Reproducible Pattern Recognition and Machine Learning

André Anjos - Biometrics Group
<andre.anjos@idiap.ch>

July 22, 2015

Reproducible Pattern Recognition and Machine Learning

Let's understand what that means[1]:

- Pattern Recognition/Machine Learning
  - Subfield of **computer science**
  - Construction and study of **algorithms**
  - Learn from and make predictions on **data**
- Reproducible (Research)
  - Ability to *reproduce* work
  - By him/herself or someone else working **independently**

[1] Source: Wikipedia

Practical Details

Hands-on without an exam:

- The course will be given in 10-12 hours
- There will be **no** formal evaluation
- There will be hands-on you're encouraged to try on the provided virtual image

Python Disclaimer

We'll use an implementation language (Python), but:

- You're not required to be familiar with it
- It is not the most important aspect of the course
- Tools and techniques can be deployed in many other scenarios

What you should be taking home

That you are **urgently** required to think about **reproducibility**, whichever the language of application is. Failing to do so will diminish the importance of your contribution at short, medium **and** long term.

Step 1: Install VirtualBox and the course image

Get one of the USB keys:

- Choose the VirtualBox executable it is suitable for your OS and copy it over to your machine.
- Copy the course image (`*.ova`) from the key
- Pass the USB key to the next person
- Install VirtualBox on your computer
- Install the `*.ova` file by going to `File -> Import Appliance`

Step 2: Check network connectivity & keyboard

- Make sure you **can use the Internet** by clicking on the Firefox web browser on the panel (top-left);
- Verify you **can use your keyboard**. The virtual machine comes pre-configured with a US QWERTY keyboard. If your computer has a different one, you will need to configure it to get it right. Symbols like [(open square brackets),] (close square brackets), ' (single quotes), " (double quotes) and : (colon) will be used through the course.

Step 3: Open a Terminal

```
$ cd Desktop/Course
```

Notice!

- Yellow blobs in this talk refer to commands/text you should execute yourself or enter on an editor.
- If the line starts with a "$" marker, it means it is a shell command, to be typed on a Terminal session.
- The shell prompt may look slightly different on your setup

Alternatively

Right-click on the "Course" folder in your Desktop and select "Open Terminal here".

Troubleshooting

- Keyboard configuration: If the QWERTY keyboard does not seem to work, you will need to right-click on that icon and make sure you choose both the keyboard layout and the language correctly, so they match the model/language of your keyboard
- OVA file cannot be imported: Some older versions of VirtualBox do not accept OVA input files like the one for our custom image. Please upgrade to the latest VirtualBox version available in this case

Disclaimer

We'll introduce a simple problem so that you understand the urge for reproducibility in computer sciences. Keep in mind:

- This is a *toy problem*, it is here only to draw examples
- In real-life^{TM}, problems are more sophisticated
- The task of describing and providing data and examples is much more complex

The Iris Flower Database (1/3)

Task: Discriminate between 3 types of Iris Flowers:

Examples (setosa, versicolor, virgnica)



Reference

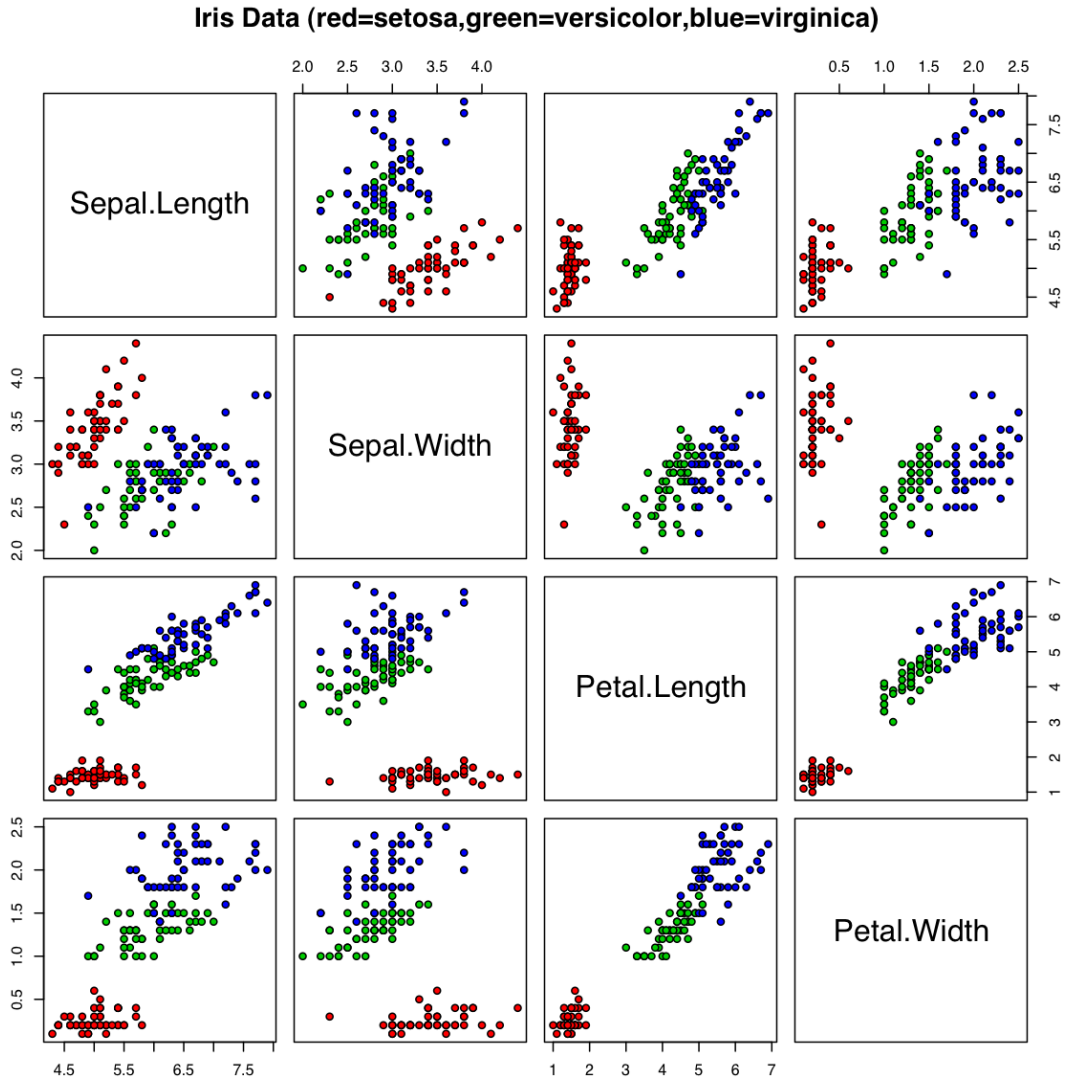
First description of Linear Discriminant Analysis: R. A. Fisher (1936). "The use of multiple measurements in taxonomic problems", Annals of Eugenics 7 (2): 179–188 `http://digital.library.adelaide.edu.au/coll/special/fisher/138.pdf`

The Iris Flower Database (2/3)

There are 50 examples of each class. 4 variables are described: Petal length and width, Sepal length and width (`plen`, `pwid`, `slen`, `swid`)

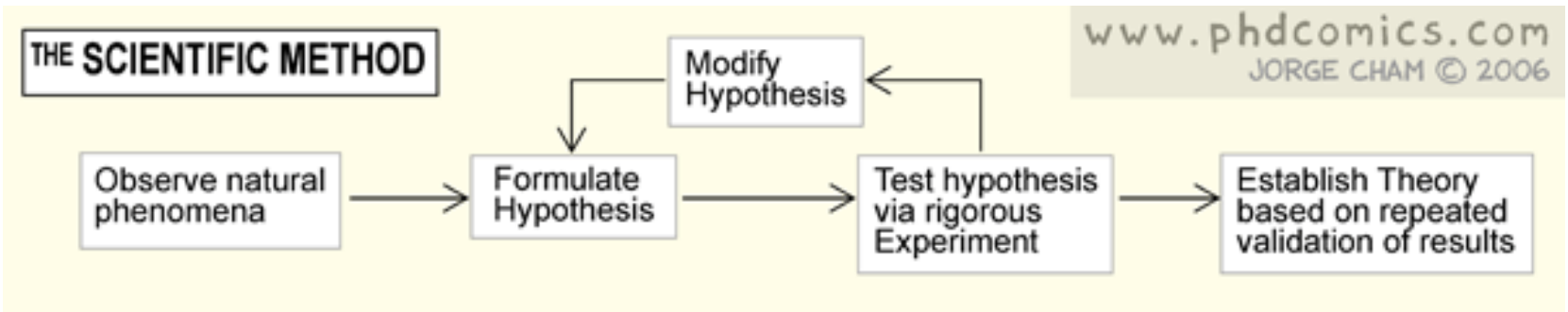
The Iris Flower Database (3/3)

The "Setosa" variant is linearly separable from the other two.



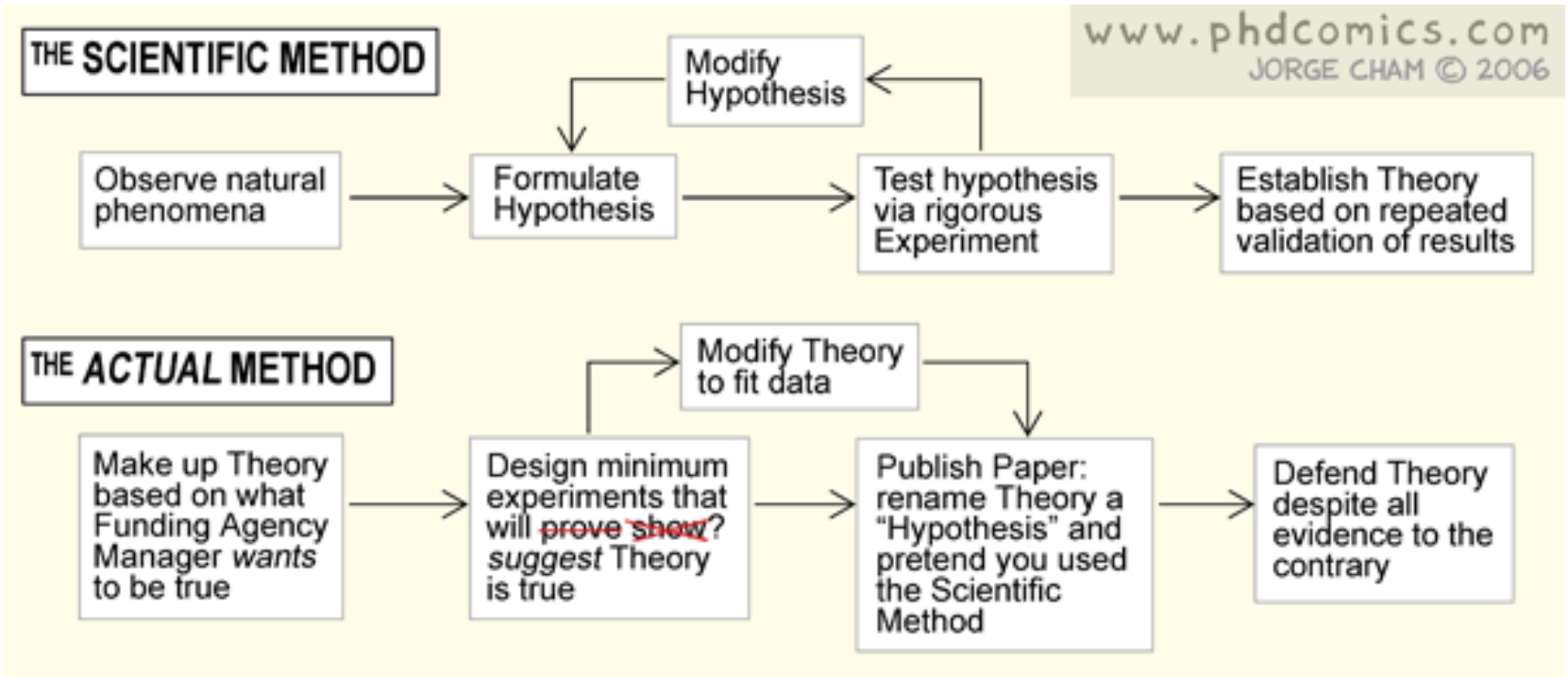
The Scientific Method

"Graphical view":



The Scientific Method

"Graphical view":



Hypothesis and Merit

Naive Working Hypothesis

The features provided on the database allow for a simple linear classifier to separate between the 3 types of flowers.

Hypothesis and Merit

Naive Working Hypothesis

The features provided on the database allow for a simple linear classifier to separate between the 3 types of flowers.

Figure of Merit

The figure we'll use for attesting this will be the *Total Classification Error Rate* (CER).

$$CER = \frac{\sum^{N} \text{Errors}}{N}$$

Hypothesis and Merit

Naive Working Hypothesis

The features provided on the database allow for a simple linear classifier to separate between the 3 types of flowers.

Figure of Merit

The figure we'll use for attesting this will be the *Total Classification Error Rate* (CER).

$$CER = \frac{\sum^{N} \text{Errors}}{N}$$

Notice

- You should constantly discuss the hypothesis and the merit
- They are essential bits of your analysis and your contribution

Validating the Hypothesis

In order to validate the hypothesis, a colleague (anonymous) tried to linearly separate the Iris Flower types using *Multiclass (linear) Logistic Regression*

Notice

- There is no reason why other classification methods cannot work equally well
- For example, your colleague could have used *Linear Discriminant Analysis* or *Support Vector Machines* with a linear kernel for the same purpose!

First Exercise

You read a paper that says:

> *It is possible to separate the Iris Flower Database using a Logistic Regression Classifier with a 0% CER.*

Now you reproduce it:

```
$ cd ex0
$ # figure it out
```

What is missing?

What is missing?

- How was the system trained?
- Was there any pre-processing?
- What was the data/protocol used to verify these results?

In brief...

So that you can verify the claim from the writer, you need **data**, **software** to reproduce the claimed results.

You contact the author

You decide to e-mail the claim's author and ask for more information:

- The author points you to the data, that you download
- He said they split the data 60% for training and 40% for testing
- They explain you that they just applied "the Logistic Regression Classifier" available on "Statistics and Machine Learning Toolbox" from Matlab

You think...

That is it! Now, I can try it out...

Second Exercise

Now we reproduce it:

```
$ cd ../ex1
$ cat email.txt
$ matlab
$ # start matlab
```

Issues?

Issues?

- Well, you have to buy a Matlab license...
- Are there untold parameters?
- How was the analysis carried out? What is it?
- Would it work for a different protocol?
- How can I apply those findings on my work?
- ...

In brief...

So that you can verify the claim from the writer, you need not only the **data**, but also (preferably free) **software** and **instructions** to reproduce the claimed results.

Statistics

Let's look at some statistics which are freely available on the net[2]:

	Country	Documents	Citable documents	Citations	Self-Citations	Citations per Document	H index
1	United States	674.876	660.385	6.863.682	2.098.285	12,10	588
2	China	533.724	530.256	1.195.832	570.771	3,90	189
3	Japan	170.877	168.377	645.155	166.574	4,79	171
4	Germany	170.090	166.330	1.039.985	218.161	8,54	235
5	United Kingdom	159.312	154.399	1.269.899	231.099	10,47	267
6	France	132.068	129.571	823.842	183.243	8,82	224
7	Canada	110.523	107.989	859.973	128.162	10,32	233
8	Italy	98.727	96.022	628.951	140.139	8,78	194
9	South Korea	92.957	91.526	394.934	68.034	6,17	146
10	Spain	84.041	82.127	434.934	108.506	7,97	161
11	Taiwan	83.238	81.978	457.728	109.171	8,14	155
12	India	80.013	79.018	283.005	49.560	7,55	132
13	Australia	69.835	67.978	443.081	61.258	9,53	177
14	Netherlands	49.674	48.229	414.325	61.569	11,73	187
15	Singapore	35.950	35.210	263.852	30.829	9,37	139
16	Brazil	35.778	35.064	138.660	30.311	6,21	106
17	Russian Federation	35.122	34.921	78.095	14.900	2,31	79

[2]Source: `http://www.scimagojr.com/countryrank.php`

Analysis

	Country	Documents	Citable documents	Citations	Self-Citations	Citations per Document	H index
1	United States	674.876	660.385	6.863.682	2.098.285	12,10	588
2	China	533.724	530.256	1.195.832	570.771	3,90	189
3	Japan	170.877	168.377	645.155	166.574	4,79	171
4	Germany	170.090	166.330	1.039.985	218.161	8,54	235
5	United Kingdom	159.312	154.399	1.269.899	231.099	10,47	267
6	France	132.068	129.571	823.842	183.243	8,82	224
7	Canada	110.523	107.989	859.973	128.162	10,32	233
8	Italy	98.727	96.022	628.951	140.139	8,78	194
9	South Korea	92.957	91.526	394.934	68.034	6,17	146
10	Spain	84.041	82.127	434.934	108.506	7,97	161
11	Taiwan	83.238	81.978	457.728	109.171	8,14	155
12	India	80.013	79.018	283.005	49.560	7,55	132
13	Australia	69.835	67.978	443.081	61.258	9,53	177
14	Netherlands	49.674	48.229	414.325	61.569	11,73	187
15	Singapore	35.950	35.210	263.852	30.829	9,37	139
16	Brazil	35.778	35.064	138.660	30.311	6,21	106
17	Russian Federation	35.122	34.921	78.095	14.900	2,31	79

- The sheer number of papers is huge (millions of documents)
- Emerging countries have a non-negligeable role
- English-speaking countries seem to have a larger number of citations per document

Analysis

	Country	Documents	Citable documents	Citations	Self-Citations	Citations per Document	H index
1	United States	674.876	660.385	6.863.682	2.098.285	12,10	588
2	China	533.724	530.256	1.195.832	570.771	3,90	189
3	Japan	170.877	168.377	645.155	166.574	4,79	171
4	Germany	170.090	166.330	1.039.985	218.161	8,54	235
5	United Kingdom	159.312	154.399	1.269.899	231.099	10,47	267
6	France	132.068	129.571	823.842	183.243	8,82	224
7	Canada	110.523	107.989	859.973	128.162	10,32	233
8	Italy	98.727	96.022	628.951	140.139	8,78	194
9	South Korea	92.957	91.526	394.934	68.034	6,17	146
10	Spain	84.041	82.127	434.934	108.506	7,97	161
11	Taiwan	83.238	81.978	457.728	109.171	8,14	155
12	India	80.013	79.018	283.005	49.560	7,55	132
13	Australia	69.835	67.978	443.081	61.258	9,53	177
14	Netherlands	49.674	48.229	414.325	61.569	11,73	187
15	Singapore	35.950	35.210	263.852	30.829	9,37	139
16	Brazil	35.778	35.064	138.660	30.311	6,21	106
17	Russian Federation	35.122	34.921	78.095	14.900	2,31	79

- The sheer number of papers is huge (millions of documents)
- Emerging countries have a non-negligeable role
- English-speaking countries seem to have a larger number of citations per document
- In short, your work is a "*needle in a haystack*"

Analysis

	Country	Documents	Citable documents	Citations	Self-Citations	Citations per Document	H index
1	United States	674.876	660.385	6.863.682	2.098.285	12,10	588
2	China	533.724	530.256	1.195.832	570.771	3,90	189
3	Japan	170.877	168.377	645.155	166.574	4,79	171
4	Germany	170.090	166.330	1.039.985	218.161	8,54	235
5	United Kingdom	159.312	154.399	1.269.899	231.099	10,47	267
6	France	132.068	129.571	823.842	183.243	8,82	224
7	Canada	110.523	107.989	859.973	128.162	10,32	233
8	Italy	98.727	96.022	628.951	140.139	8,78	194
9	South Korea	92.957	91.526	394.934	68.034	6,17	146
10	Spain	84.041	82.127	434.934	108.506	7,97	161
11	Taiwan	83.238	81.978	457.728	109.171	8,14	155
12	India	80.013	79.018	283.005	49.560	7,55	132
13	Australia	69.835	67.978	443.081	61.258	9,53	177
14	Netherlands	49.674	48.229	414.325	61.569	11,73	187
15	Singapore	35.950	35.210	263.852	30.829	9,37	139
16	Brazil	35.778	35.064	138.660	30.311	6,21	106
17	Russian Federation	35.122	34.921	78.095	14.900	2,31	79

- The sheer number of papers is huge (millions of documents)
- Emerging countries have a non-negligeable role
- English-speaking countries seem to have a larger number of citations per document
- In short, your work is a "*needle in a haystack*"

Mission Impossible

Your task, if you choose to accept it, is to allow other researchers to find your work.

Your contribution

Your research is a contribution to the world's knowledge base:

- You write a paper, because you think your idea is relevant

Your contribution

Your research is a contribution to the world's knowledge base:

- You write a paper, because you think your idea is relevant
- You want it to get to the attention of the biggest possible number of parties

Your contribution

Your research is a contribution to the world's knowledge base:

- You write a paper, because you think your idea is relevant
- You want it to get to the attention of the biggest possible number of parties
- You want researchers to be able to try your ideas in cleaner/faster/cheaper ways

Your contribution

Your research is a contribution to the world's knowledge base:

- You write a paper, because you think your idea is relevant
- You want it to get to the attention of the biggest possible number of parties
- You want researchers to be able to try your ideas in cleaner/faster/cheaper ways
- Simply put, you want to lower the "entrance barrier", as much as possible, to your ideas

How to do that?

You can approach the problem in many ways but, possibly, you want to do it through **all** possible means:

- You want to write papers well (clear ideas, contributions)
- You want to submit to prestigious conferences and journals

How to do that?

You can approach the problem in many ways but, possibly, you want to do it through **all** possible means:

- You want to write papers well (clear ideas, contributions)
- You want to submit to prestigious conferences and journals
- You want to make it dead-simple to reproduce your claims and transpose your ideas to other problems

Motivation

A Scalable Formulation of Probabilistic Linear Discriminant Analysis: Applied to Face Recognition

Laurent El Shafey, Chris McCool, Roy Wallace, and Sébastien Marcel

Appendix A
Mathematical Derivations

The goal of the following section is to provide more detailed proofs of the formulae given in the article for both training and computing the likelihood.

The following proofs make use of a formulation of the inverse of a block matrix that uses the Schur complement. The corresponding identity can be found in [1] (Equations 1.11 and 1.10),

$$\begin{bmatrix} \boldsymbol{L} & \boldsymbol{M} \\ \boldsymbol{N} & \boldsymbol{O} \end{bmatrix}^{-1} = \begin{bmatrix} \boldsymbol{R}, & -\boldsymbol{RMO}^{-1} \\ -\boldsymbol{O}^{-1}\boldsymbol{NR}, & \boldsymbol{O}^{-1}+\boldsymbol{O}^{-1}\boldsymbol{NRMO}^{-1} \end{bmatrix}, \quad (51)$$

where we have substituted $\boldsymbol{R} = \left(\boldsymbol{L} - \boldsymbol{MO}^{-1}\boldsymbol{N}\right)^{-1}$.

Another related expression is the Woodbury matrix identity (Equation $C.7$ of [2]), which states that,

$$\left(\boldsymbol{L}+\boldsymbol{MON}\right)^{-1} = \boldsymbol{L}^{-1} - \boldsymbol{L}^{-1}\boldsymbol{M}\left(\boldsymbol{O}^{-1}+\boldsymbol{NL}^{-1}\boldsymbol{M}\right)^{-1}\boldsymbol{NL}^{-1}. \quad (52)$$

A. Scalable training

The bottleneck of the training procedure is the expectation step (E-Step) of the Expectation-Maximization algorithm. This E-Step requires the computation of the first and second order moments of the latent variables.

1) Estimating the first order moment of the Latent Variables: The most computationally expensive part when estimating the latent variables is the inversion of the matrix $\tilde{\boldsymbol{\mathcal{P}}}$ (Equation (27)). This matrix is block diagonal, the two blocks being $\boldsymbol{\mathcal{P}}_0$ (Equation (28)) and (a repetition of) $\boldsymbol{\mathcal{P}}_1$ (Equation (29)),

$$\tilde{\boldsymbol{\mathcal{P}}} = \begin{bmatrix} \boldsymbol{\mathcal{P}}_0 & 0 & \cdots & 0 \\ 0 & \boldsymbol{\mathcal{P}}_1 & \ddots & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \boldsymbol{\mathcal{P}}_1 \end{bmatrix}. \quad (53)$$

The inverse of $\boldsymbol{\mathcal{P}}_1$ is equal to the matrix $\boldsymbol{\mathcal{G}}$, defined by (30). This matrix is of constant size $(D_G \times D_G)$, irrespective of the number of training samples for the class. In addition, the inversion of $\boldsymbol{\mathcal{P}}_0$ can be further optimised using the block matrix inversion identity introduced at the beginning of this section, leading to

$$\boldsymbol{\mathcal{P}}_0^{-1} = \begin{bmatrix} \boldsymbol{\mathcal{F}}_{J_i} & \sqrt{J_i}\boldsymbol{\mathcal{H}}^T \\ \sqrt{J_i}\boldsymbol{\mathcal{H}} & \left(\boldsymbol{I}_{D_G} - J_i\boldsymbol{\mathcal{H}}\boldsymbol{F}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{G}\right)\boldsymbol{\mathcal{G}} \end{bmatrix}, \quad (54)$$

where $\boldsymbol{\mathcal{F}}_{J_i}$ is defined by (33) and $\boldsymbol{\mathcal{H}}$ by (37).

Then, the computation of $\tilde{\tilde{\boldsymbol{\mathcal{P}}}}^{-1}\tilde{\boldsymbol{A}}^T\tilde{\boldsymbol{\Sigma}}^{-1}$ gives a block diagonal matrix, the first block being

$$\begin{bmatrix} \sqrt{J_i}\boldsymbol{\mathcal{F}}_{J_i}\boldsymbol{F}^T\boldsymbol{\mathcal{S}} \\ \boldsymbol{\mathcal{G}}\boldsymbol{G}^T\boldsymbol{\Sigma}^{-1}\left(\boldsymbol{I}_{D_x} - J_i\boldsymbol{F}\boldsymbol{\mathcal{F}}_{J_i}\boldsymbol{F}^T\boldsymbol{\mathcal{S}}\right) \end{bmatrix},$$

and the other ones being equal to $\boldsymbol{\mathcal{G}}\boldsymbol{G}^T\boldsymbol{\Sigma}^{-1}$.

As explained in section III.B.a of the article, $\boldsymbol{h}_i$ corresponds to the upper sub-vector of $\tilde{\tilde{\boldsymbol{y}}}_i$ and is not affected by the change of variable, as depicted in (21). Therefore, the first order moment of $\boldsymbol{h}_i$ is directly obtained by multiplying the first block-rows of the matrix $\tilde{\tilde{\boldsymbol{\mathcal{P}}}}^{-1}\tilde{\boldsymbol{A}}^T\tilde{\boldsymbol{\Sigma}}^{-1}$ with $\tilde{\tilde{\boldsymbol{x}}}_i$, which gives (31).

Considering only the $\tilde{\tilde{\boldsymbol{w}}}_i$ (lower) sub-vector of $\tilde{\tilde{\boldsymbol{y}}}_i$, the corresponding (lower) part $\tilde{\tilde{\boldsymbol{B}}}$ of the matrix $\tilde{\tilde{\boldsymbol{\mathcal{P}}}}^{-1}\tilde{\boldsymbol{A}}^T\tilde{\boldsymbol{\Sigma}}^{-1}$ can be decomposed into a sum of two matrices, the first one being sparse with a single non-zero block (upper left) equal to $\boldsymbol{\mathcal{B}}_0 = -J_i\boldsymbol{\mathcal{G}}\boldsymbol{G}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{F}\boldsymbol{\mathcal{F}}_{J_i}\boldsymbol{F}^T\boldsymbol{\mathcal{S}}$, and the second one being diagonal by blocks with identical blocks $\boldsymbol{\mathcal{B}}_1 = \boldsymbol{\mathcal{G}}\boldsymbol{G}^T\boldsymbol{\Sigma}^{-1}$,

$$\tilde{\tilde{\boldsymbol{B}}} = \begin{bmatrix} \boldsymbol{\mathcal{B}}_0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} \boldsymbol{\mathcal{B}}_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \boldsymbol{\mathcal{B}}_1 \end{bmatrix}. \quad (55)$$

Furthermore, the first order moment of the variables $\tilde{\boldsymbol{w}}_i$ is given by

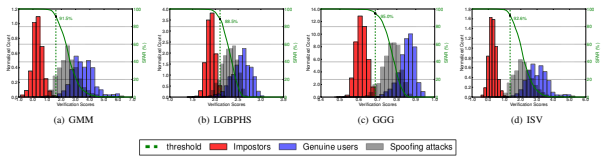
$$E\left[\tilde{\boldsymbol{w}}_i|\tilde{\tilde{\boldsymbol{x}}}_i, \boldsymbol{\Theta}\right] = \left(\tilde{\boldsymbol{U}}^T \otimes \boldsymbol{I}_{D_G}\right)\begin{bmatrix} \boldsymbol{\mathcal{B}}_0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\tilde{\tilde{\boldsymbol{x}}}_i + \left(\tilde{\boldsymbol{U}}^T \otimes \boldsymbol{I}_{D_G}\right)\begin{bmatrix} \boldsymbol{\mathcal{B}}_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \boldsymbol{\mathcal{B}}_1 \end{bmatrix}\left(\tilde{\boldsymbol{U}} \otimes \boldsymbol{I}_{D_x}\right)\tilde{\tilde{\boldsymbol{x}}}_i. \quad (56)$$

The previous decomposition greatly simplifies the computation, and leads to the following expression for each $\boldsymbol{w}_{i,j}$,

$$E\left[\boldsymbol{w}_{i,j}|\tilde{\tilde{\boldsymbol{x}}}_i, \boldsymbol{\Theta}\right] = \boldsymbol{\mathcal{G}}\boldsymbol{G}^T\boldsymbol{\Sigma}^{-1}\tilde{\boldsymbol{x}}_{i,j} - \boldsymbol{\mathcal{G}}\boldsymbol{G}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{F}\boldsymbol{\mathcal{F}}_{J_i}\boldsymbol{F}^T\boldsymbol{\mathcal{S}}\sum_j \tilde{\boldsymbol{x}}_{i,j} \quad (57)$$

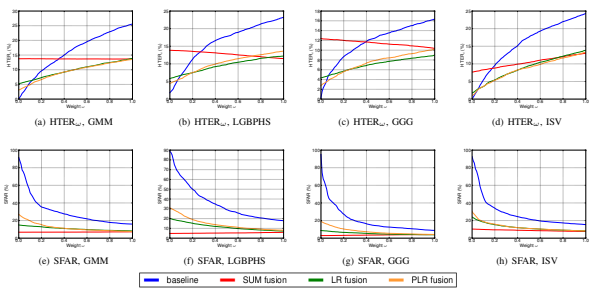
L. El Shafey is with Idiap Research Institute and Ecole Polytechnique Fédérale de Lausanne, Switzerland e-mail: laurent.el-shafey@idiap.ch
C. McCool, R. Wallace and S. Marcel are with Idiap Research Institute, Martigny, Switzerland e-mail: {christopher.mccool,roy.wallace,sebastien.marcel}@idiap.ch

The research leading to these results has received funding from the European Communitys Seventh Framework Programme (FP7) under grant agreements 238803 (BBfor2) and 257289 (TABULA RASA).



(a) GMM (b) LGBPHS (c) GGG (d) ISV

Fig. 10: Score distributions of baseline face verification systems. The full green line shows how SFAR changes with moving the threshold.



(a) HTER$_\omega$, GMM (b) HTER$_\omega$, LGBPHS (c) HTER$_\omega$, GGG (d) HTER$_\omega$, ISV

(e) SFAR, GMM (f) SFAR, LGBPHS (g) SFAR, GGG (h) SFAR, ISV

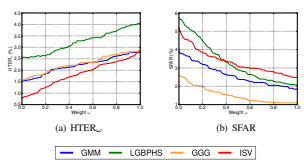
Fig. 12: EPSC for comparison of fusion techniques of baselines with LBP anti-spoofing algorithm

D. Performance of fused systems

In our last experiment, we compare the four face verification systems when fused with ALL counter-measures using PLR fusion scheme. Firstly, we illustrate how fusion changes the score distribution for each of them separately in Figure 14. Then, in Figure 15 we compare which of the fused systems performs the best.

While Figure 10 shows that the spoofing attacks of Replay-Attack are in the optimal category when fed to the baseline face verification systems, Figure 14 illustrates that their effectiveness has vastly changed after fusion. The score distribution of the spoofing attacks is now mostly overlapping with the score distribution of the zero-effort impostors, allowing for better discriminability between the positive class and the two negative classes. The results are reflecting this observation: even when the threshold is obtained using the licit scenario, SFAR has dropped to less then 6%.

The comparison between the EPSC curves given in Figure 11(a) and Figure 15(a), confirms the above observations:

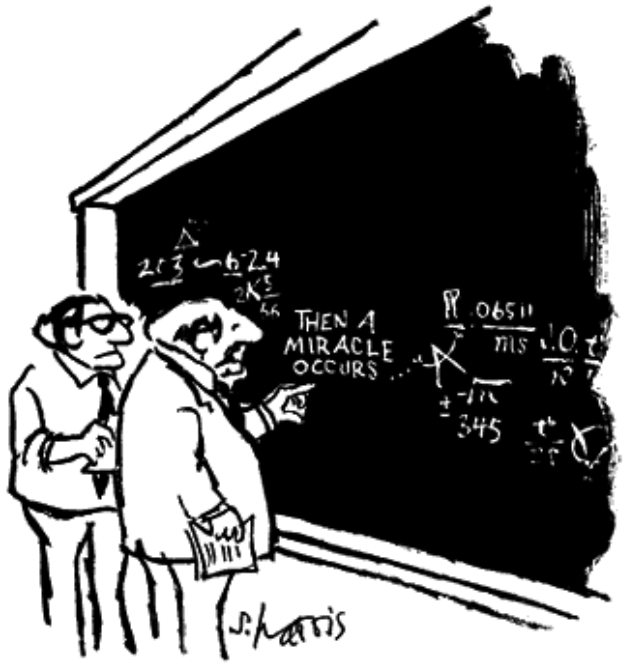


(a) HTER$_\omega$ (b) SFAR

Fig. 15: EPSC curves to compare fused systems

while HTER$_\omega$ increases rapidly with ω and reaches up to 25% for some of the baseline systems, it increases very mildly and does not exceed 4.1% for the fused systems. The major augmentation of the robustness to spoofing of the systems after

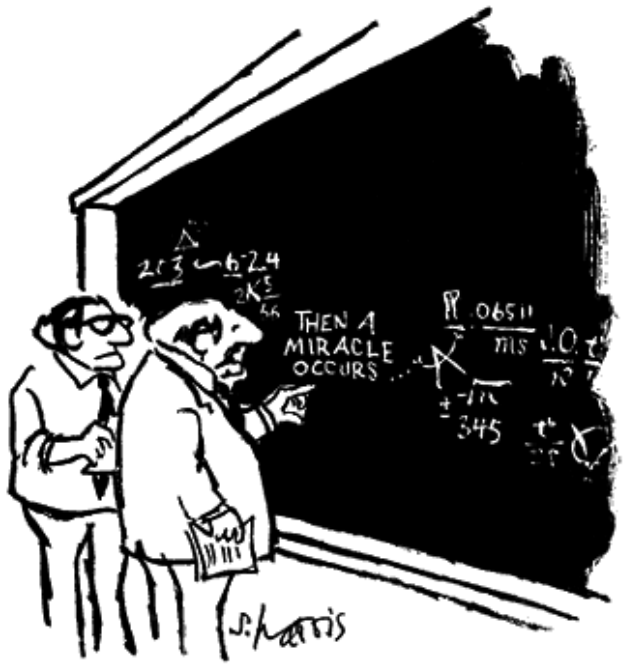
Motivation



"I think you should be more explicit here in step two."

How many times?

Motivation



"I think you should be more explicit here in step two."

Crossed a publication and openly decided to **ignore it because it would be too hard to apply** *those doubtful results on your research?*

Motivation

Worked day and night to **incorporate some results** *on your own work but:*

- There were **untold parameters** that needed adjustment and you couldn't get hold of them?
- Realized the proposed algorithm **worked only on the specific data** shown at the original paper?
- Realized that something did **not quite add up** in the end?

Motivation

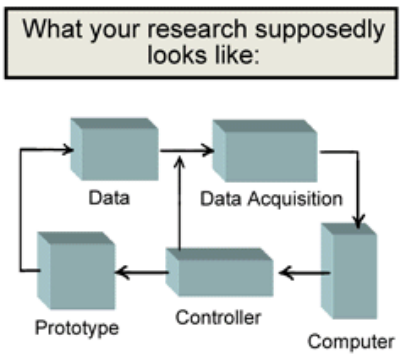


Figure 1. Experimental Diagram

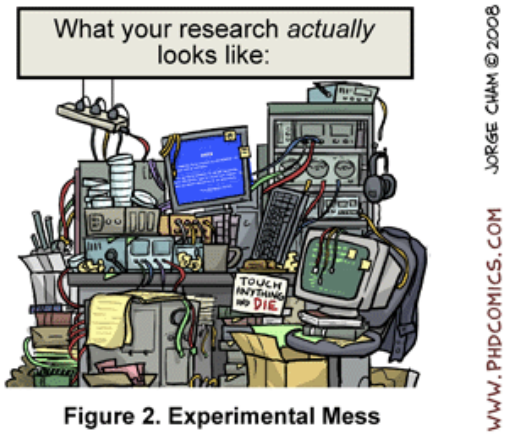


Figure 2. Experimental Mess

Had **to take over** *the work from another colleague that left and had to start from scratch - months into programming to make things work again?*

Motivation

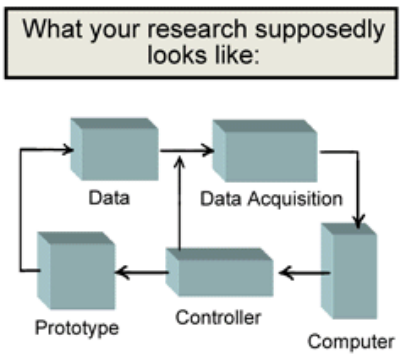


Figure 1. Experimental Diagram

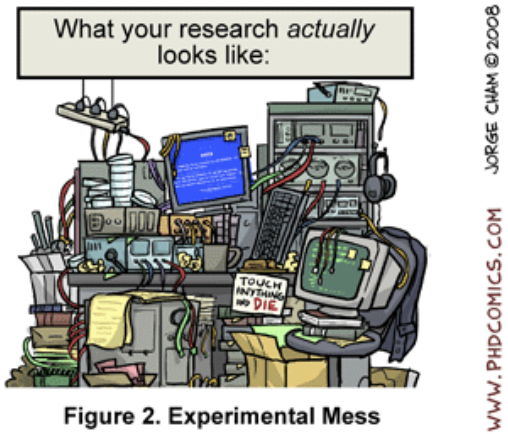


Figure 2. Experimental Mess

Would have liked to **replay to someone about your work**, *but you couldn't really remember all details when you first made it work? Or you* **could not make it work** *at all?*

An article about computational science in a scientific publication is **not the scholarship** *itself, it is merely* **advertising** *of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.*

D. Donoho, 2010[3]

[1] `http://biostatistics.oxfordjournals.org/content/11/3/385.long`

Enter "Reproducible Research" (RR)[4]

One term that aggregates work comprising of:

- a **paper**, that describe your work in all relevant details
- **code** to reproduce all results
- **data** required to reproduce the results
- **instructions**, on how to apply the *code* on the *data* to replicate the results on the *paper*.

[2] `http://reproducibleresearch.net`

With respect to "Replication"[5]

- RR is the practice of presenting computational research such that members of a scientific community may easily reproduce and verify the results.
- Distinct from "scientific replication"
- Reproducibility i.e. RR verifies an experimental result with the **same** data and procedures
- Replication strengthens evidence about a scientific theory through **different** data and procedures

[3] `http://jblevins.org/log/rep`

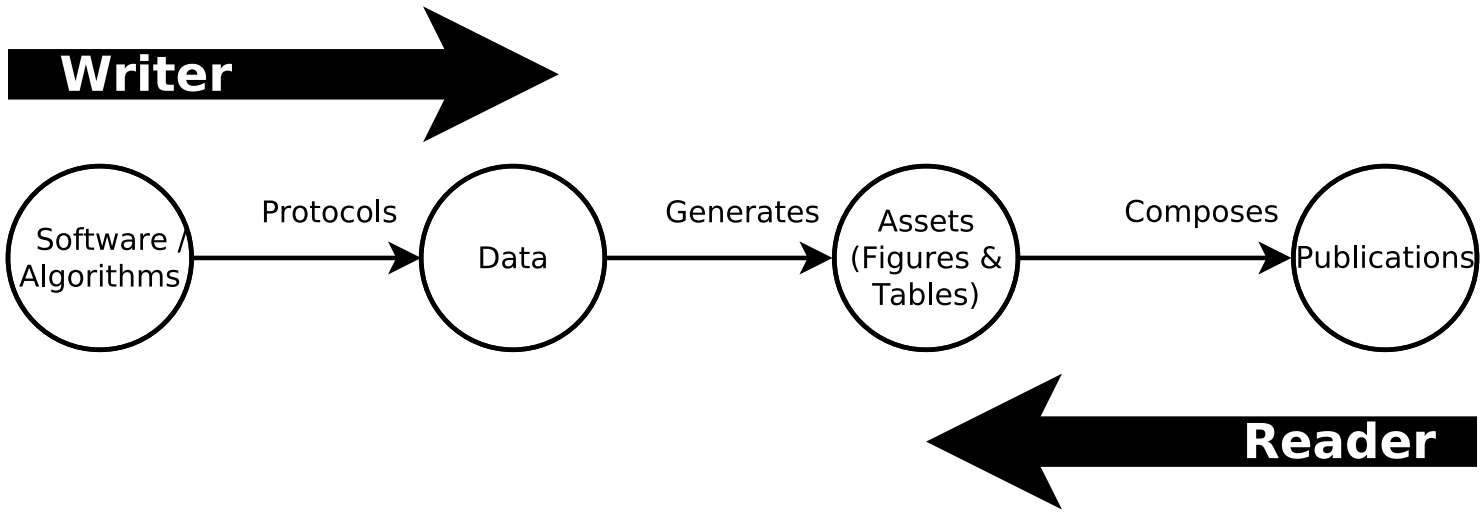
Levels of Reproducibility[6]

With respect to an independent researcher (reader):

0. Irreproducible
1. Cannot seem to reproduce
2. Reproducible, with extreme effort ($>$ 1 month)
3. Reproducible, with considerable effort ($>$ 1 week)
4. Easily reproducible ($\sim$ 15 min.), but requires proprietary software (e.g. Matlab)
5. **Easily reproducible ($\sim$ 15 min.), only free software**

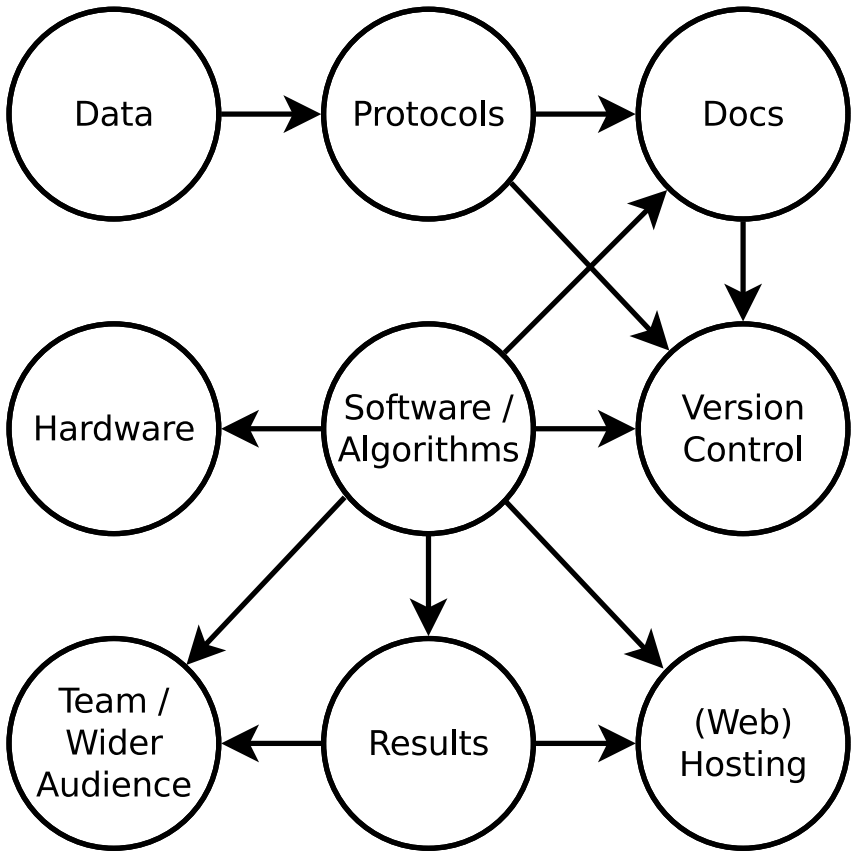
[4] *Reproducible Research in Signal Processing: What, why and how*, Vandewalle, Kovacevic and Vetterli, 2012

Pipeline


Writer
Software / Algorithms
Protocols
Data
Generates
Assets (Figures & Tables)
Composes
Publications
Reader

Why?

Finally, writing and distributing code and data takes time...



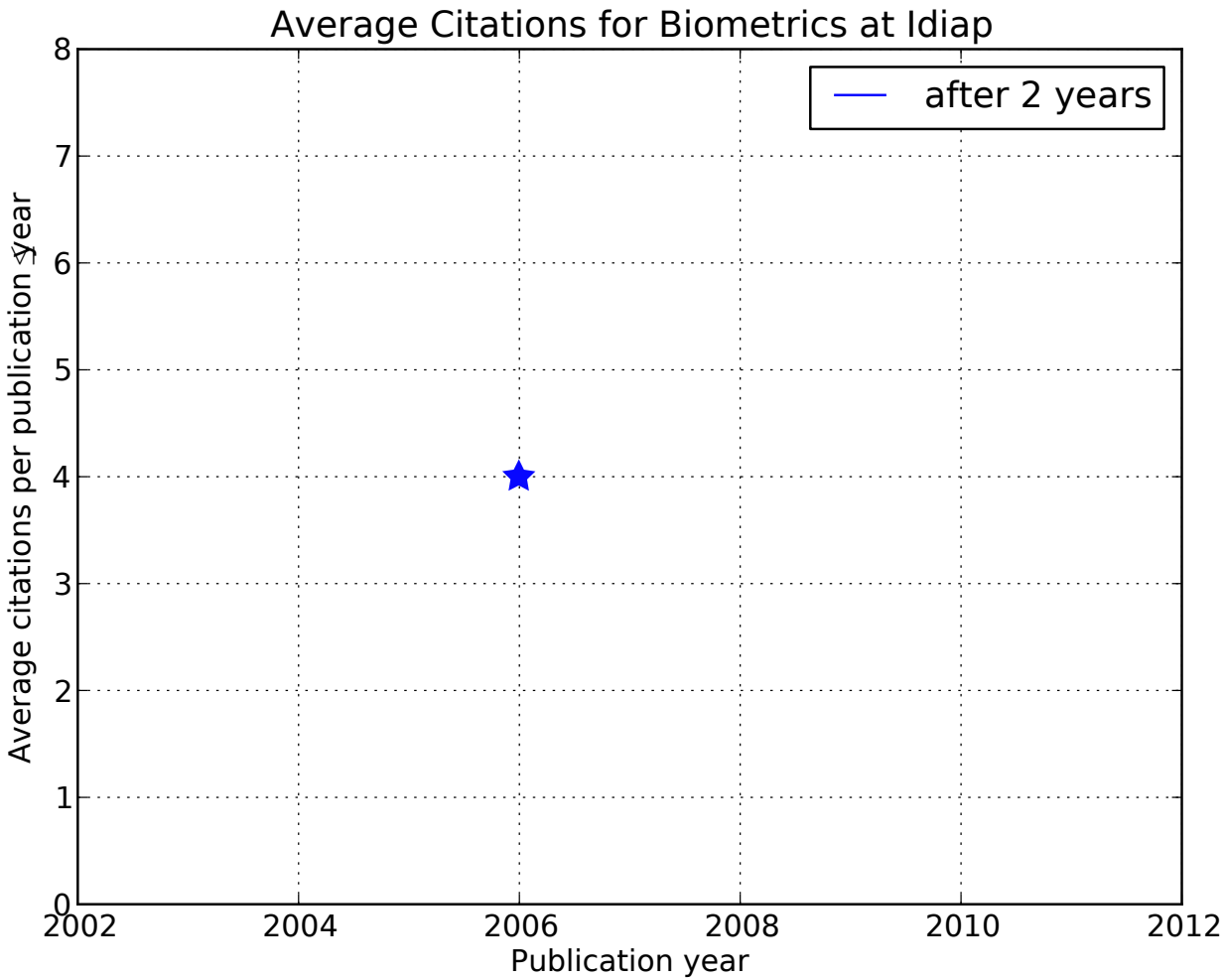
Why?

Boost your research **impact (visibility)**:

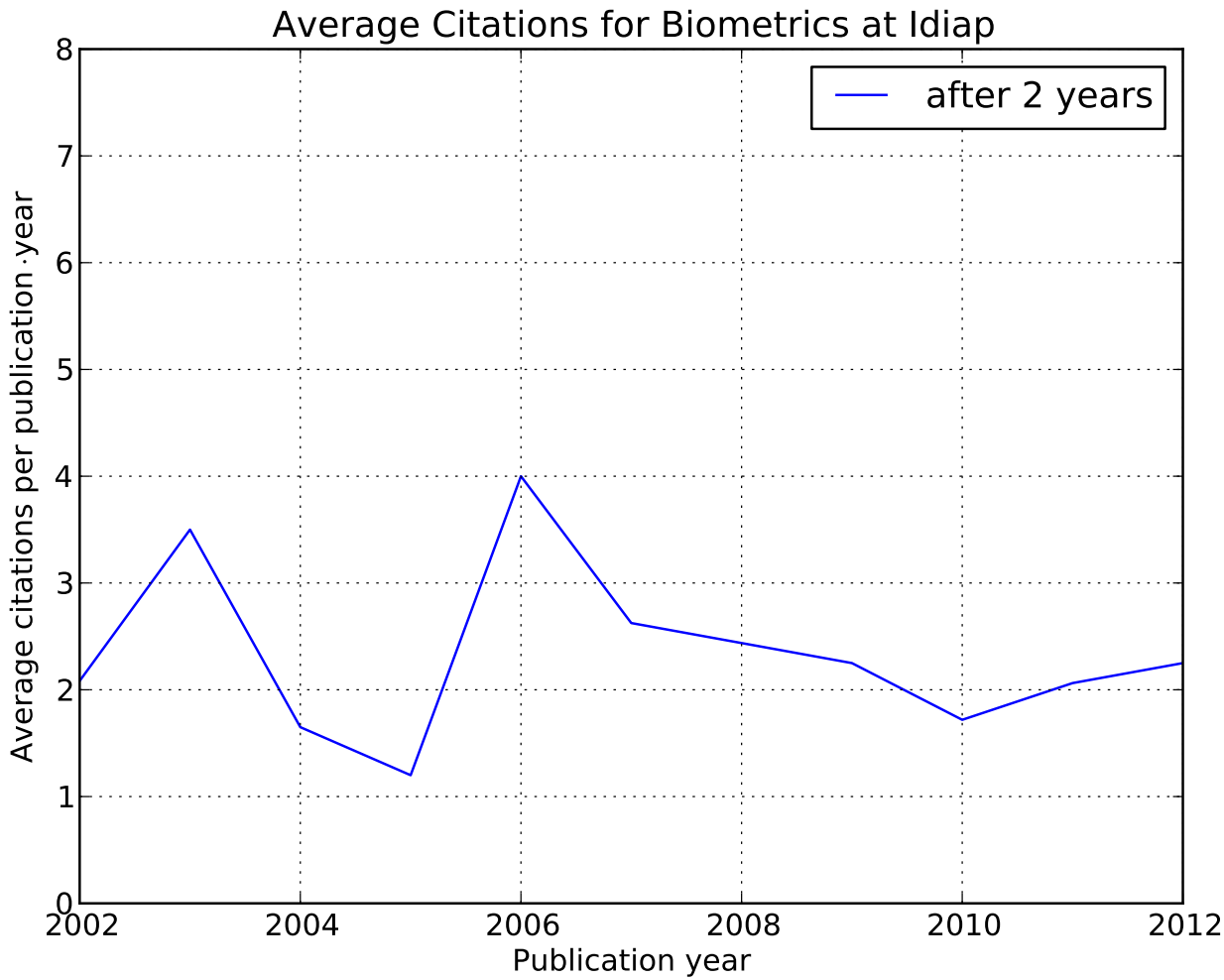
- **Lower entrance barrier** to your publications
- The current number of RR papers is **rather small** - you have a clear chance to stand out today:
  - Only **10% of TIP** papers provide source code[7].
- Statistically, your work is **more valuable** if it is RR:
  - **13 out of the top 15 most cited** articles in TPAMI or TIP provide (at least) source code
  - The average number of citations for papers that provide source-code in TIP is **7 fold** that of papers that don't.

[5] *Code Sharing is Associated with Research Impact in Image Processing*, Patrick Vandewalle, 2012

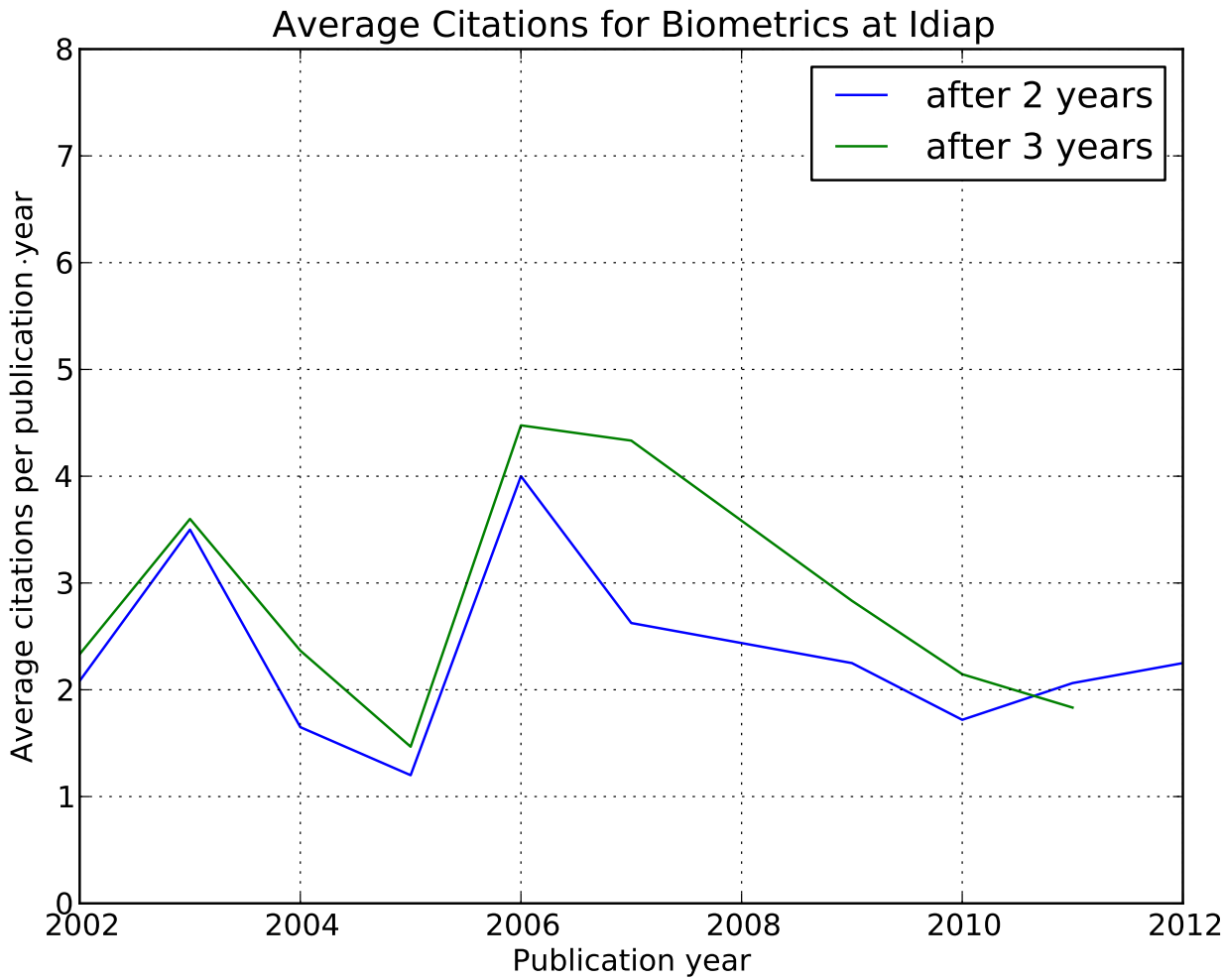
Example: Our Group's Citation Score



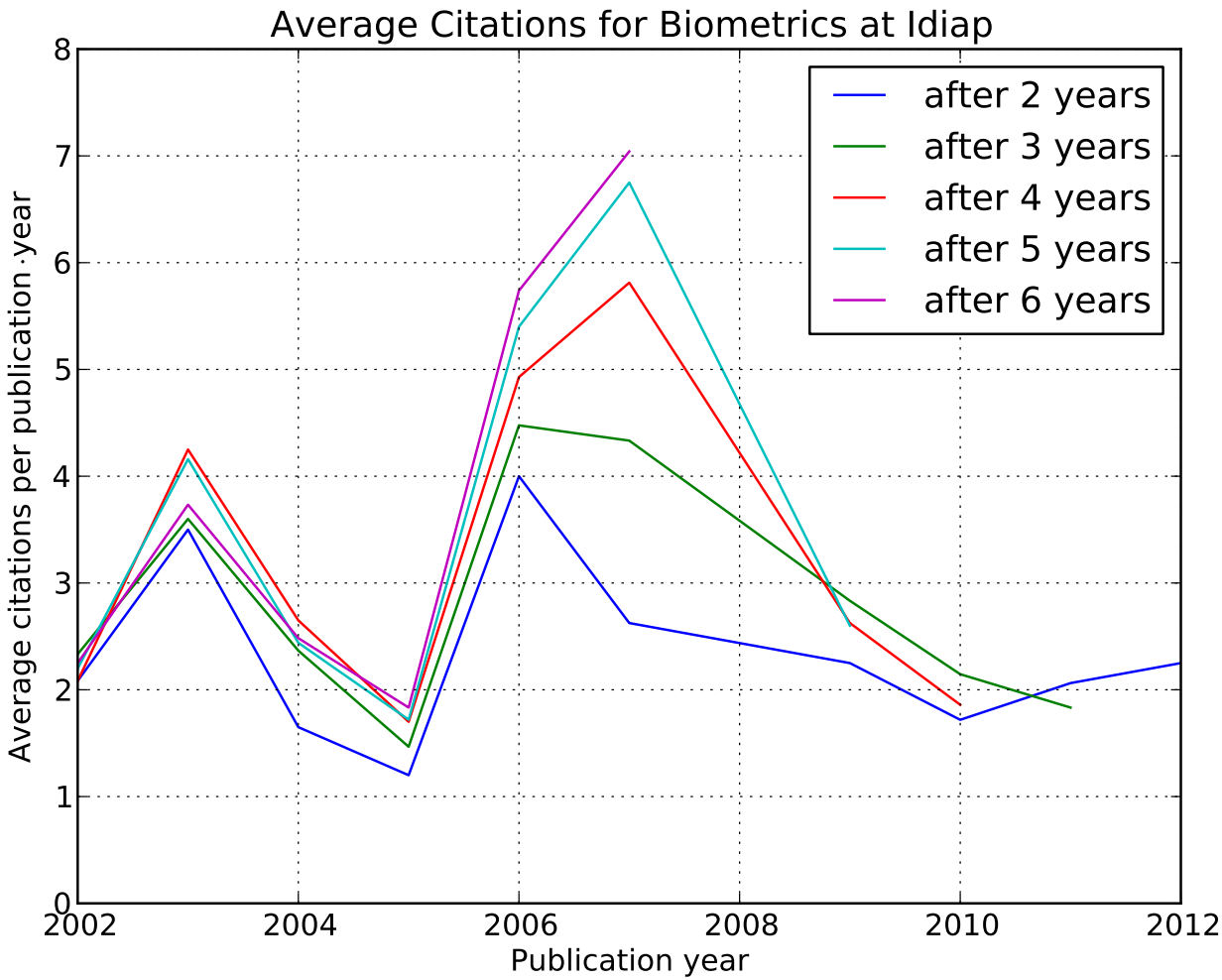
Example: Our Group's Citation Score



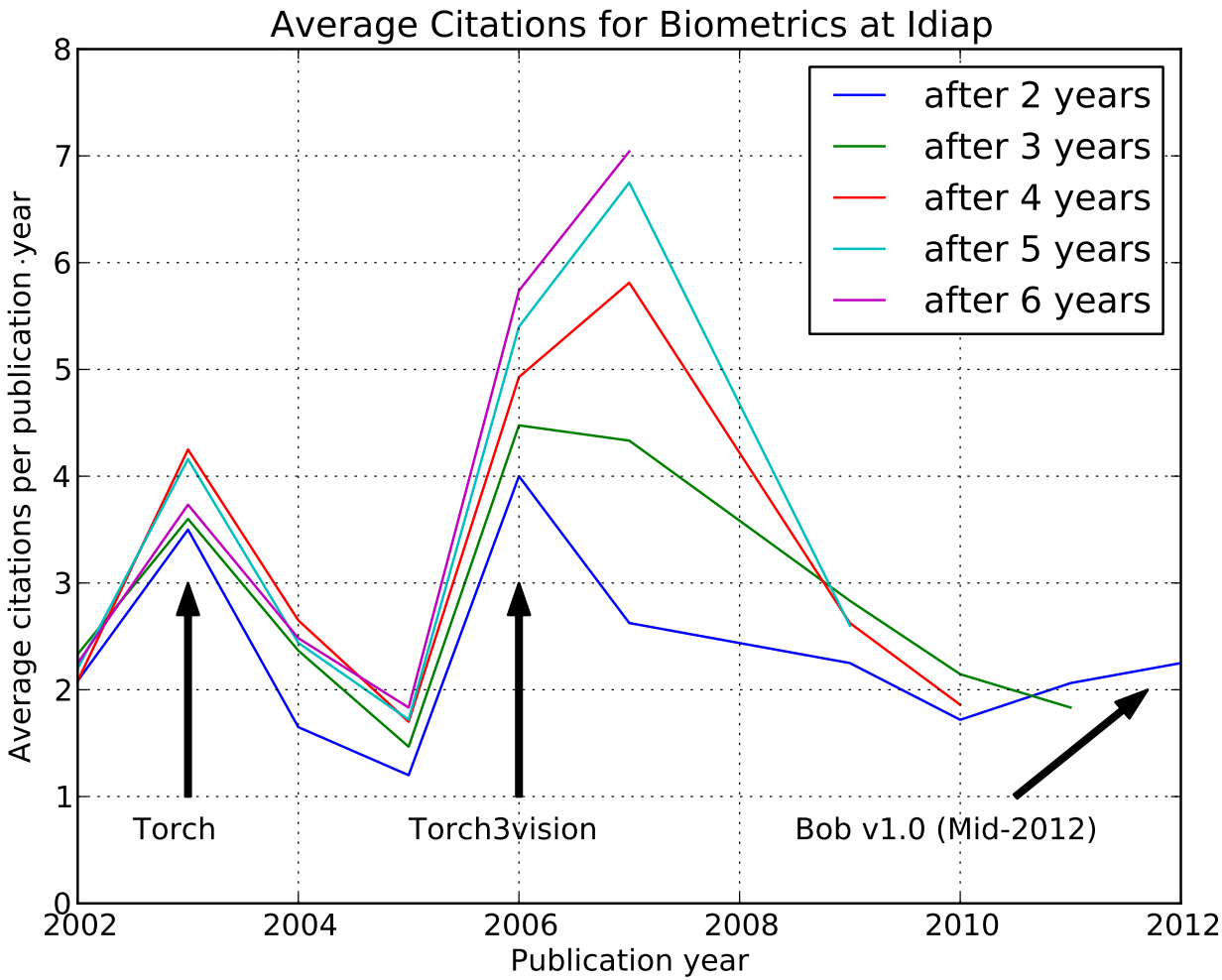
Example: Our Group's Citation Score



Example: Our Group's Citation Score



Example: Our Group's Citation Score



Productivity Boost

- Don't start from scratch - start from the state-of-the-art
- Start to work fast (useful for visiting staff and collaborators)
- Facilitate collaborative work with external partners
- Create something **durable**: Avoid re-doing things. Do it once and ask your team members to:
  - Improve it
  - Test it
  - Document it
  - Keep it operational.

  All for the benefit of all members.

In practice

Organize yourself so you are **always** *doing RR*

Work as a team to:

- Organize **basic tools** so that you have a library that is documented and re-usable by all
- Organize **the data** so it is easy to replay analysis protocols by all
- Write and distribute **applications** that use your basic tools and data to **generate interesting results**.

Benefits:

- Research is always kept reproducible
- People help each other in case of problems
- New colleagues can start (nearly) immediately to produce high-quality results.

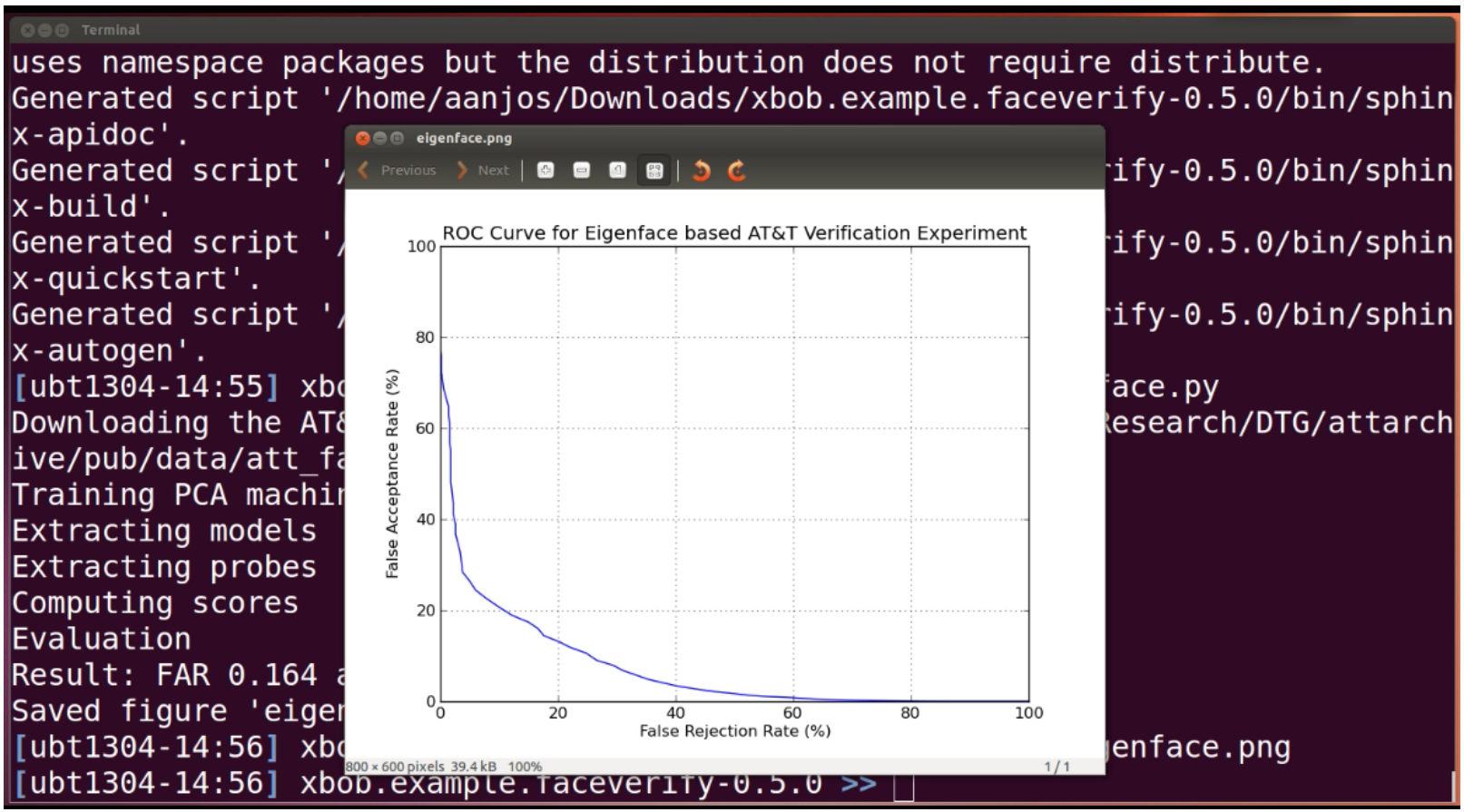
At our group:

We handle **Level 5** RR (free software, easy deployment) in 3 ways:

- Develop and maintain a basic set of tools that we all share: database protocol APIs, signal (image, audio) processing, machine learning - **Bob**
- Provide most of our **databases publicly**, free of charge
- Provide end-user **applications** wrapped in an **easy to deploy** packaging system that users (potential citers) can download, install and extend - **Bob packages**.

Example Deployment

Basic Face Recognition Example (∼7000 downloads since Sep./2011)

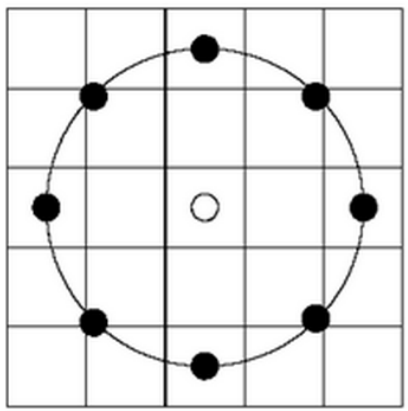


(Click to launch external video)

It's not just about throwing data at the web: Case Study

Local Binary Patterns, **cited by 6068** - Matlab Toolbox (UOULU)[8]

$LBP_{8,r2}$



Evaluate on position

Compare

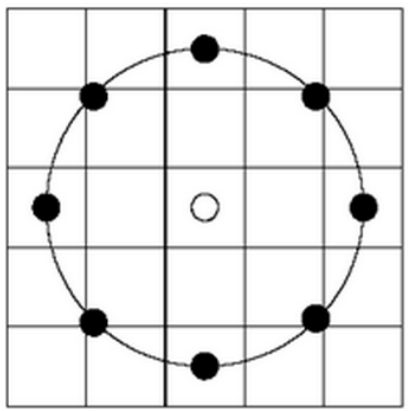
Get a code

[6] *Multiresolution gray-scale and rotation invariant texture classification with local binary patterns*, Ojala, 2002 (TPAMI)

It's not just about throwing data at the web: Case Study

Local Binary Patterns, **cited by 6068** - Matlab Toolbox (UOULU)[8]

$LBP_{8,r2}$



Bug at Matlab bi-linear interpolation

Evaluate on position

Interpolation Error

Compare

Error Propagated

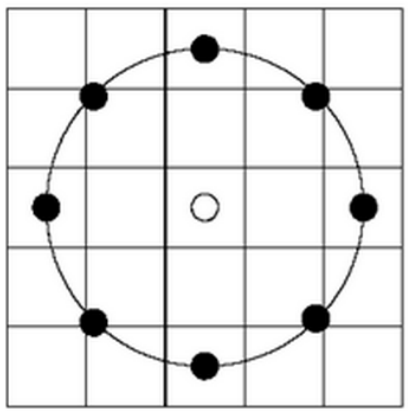
Get a code

[6] *Multiresolution gray-scale and rotation invariant texture classification with local binary patterns*, Ojala, 2002 (TPAMI)

It's not just about throwing data at the web: Case Study

Local Binary Patterns, **cited by 6068** - Matlab Toolbox (UOULU)[8]

$LBP_{8,r2}$



Bug at Matlab bi-linear interpolation

Evaluate on position

Interpolation Error

Compare

Error Propagated

Get a code

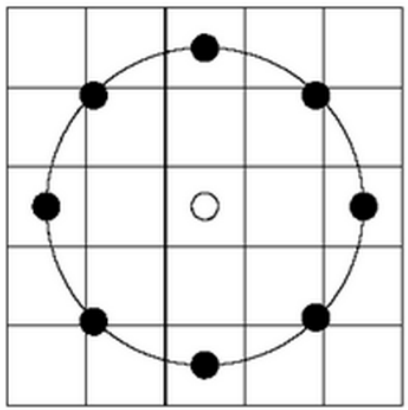
In natural images, differences can amount to ∼5% of codes

[6] *Multiresolution gray-scale and rotation invariant texture classification with local binary patterns*, Ojala, 2002 (TPAMI)

It's not just about throwing data at the web: Case Study

Local Binary Patterns, **cited by 6068** - Matlab Toolbox (UOULU)[8]

$LBP_{8,r2}$



Bug at Matlab bi-linear interpolation

Evaluate on position

Interpolation Error

Compare

Error Propagated

Get a code

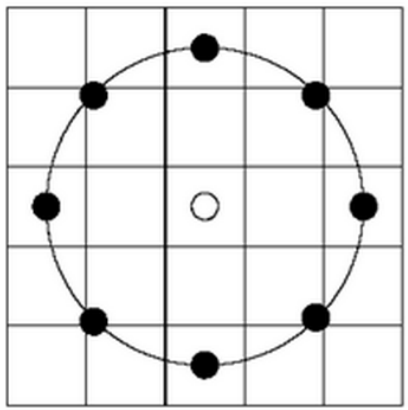
How to plan for errors?

[6] *Multiresolution gray-scale and rotation invariant texture classification with local binary patterns*, Ojala, 2002 (TPAMI)

It's not just about throwing data at the web: Case Study

Local Binary Patterns, **cited by 6068** - Matlab Toolbox (UOULU)[8]

$LBP_{8,r2}$



Bug at Matlab bi-linear interpolation

Evaluate on position

Interpolation Error

Compare

Error Propagated

Get a code

How to plan for errors?: **Software Re-distribution**

[6] *Multiresolution gray-scale and rotation invariant texture classification with local binary patterns*, Ojala, 2002 (TPAMI)

Fields of Application

Requirements

- Data that serves as input must be copiable
- Procedure must be easily copied:
  - Computer-based routines
  - Statistical or Deterministic methods

Counter-Examples

- Theoretical Physics (or disciplines of any sort)
- Biological Experimentation (see "replicability")
- Humanities
- ...

Python

In this course, we'll review RR techniques applied to *Machine Learning Tasks* that can be implemented using the *Python* programming language.

Disclaimer

RR *can* be implemented in other programming languages

Reasoning

Why Python?

- full programming language (classes, exceptions, lambda functions, complex types)
- extensible via C, C++
- modular
- free
- cross-platform
- widely used
- well documented
- well supported

Python

On a terminal, type the following to start the Python interpreter:

```
$ python
Python 2.7.3 (default, Sep 26 2012, 21:51:14)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more ...
>>> 3 + 4
7
>>> print("Hello, World!")
Hello, World!
```

Try it

You can try all examples in yellow boxes.

Help is built-in:

```
>>> help() #help on help ;-)
...
help> quit
```

Modules

Most of Python magic lives in outside modules. You need to `import` modules to use their functionality:

```
>>> import os
```

If you don't know it, ask for help:

```
>>> help(os)
```

To access an object inside another module (or other object), use the `.` (dot) operator:

```
>>> help(os.listdir)
```

Standard Modules

`http://docs.python.org/2/library/index.html`

Exiting the interpreter

To leave the interpreter, use the `exit()` function:

```
>>> exit()
```

Tip

You can also use the keyboard sequence `CTRL-D`, to exit.

Objects (== variables)

Objects in Python are *dynamically* typed:

```
>>> a = 7
>>> type(a)
<type 'int'>
>>> a/2
3
>>> b = 7.
>>> type(b)
<type 'float'>
>>> b/2
3.5
>>> c = 7+5j
>>> type(c)
<type 'complex'>
>>> c/2
(3.5+2.5j)
>>> d = '7'
>>> type(d)
<type 'str'>
```

Objects (part 2)

Everything **is** an object:

```
>>> import os
>>> type(os)
<type 'module'>
>>> andre = os
>>> help(andre)
>>> type(os.listdir)
<type 'builtin_function_or_method'>
>>> foo = os.listdir
>>> type(foo)
```

Sequences

Two main types of sequences: lists and tuples (read-only lists). We cover lists, but you can read more online about tuples.

```
>>> lst = [7, 3., 'string']
>>> lst[0] # N.B.: indexing is 0-based
7
>>> lst[1:3] # or lst[1:]
[3.0, 'string']
>>> lst[-1] # or lst[2]
'string'
>>> lst.remove(3.) #powerful
>>> print lst
[7, 'string']
>>> lst.append('x')
>>> print lst
[7, 'string', 'x']
>>> len(lst)
3
>>> 3. in lst
False
>>> lst.index('x')
2
```

Sequences (part 2)

Explore: Use TAB completion

```
>>> lst.<TAB><TAB>
```

Remember

In doubt call `help`, *e.g.*, `help(lst.remove)`

More information

`http://docs.python.org/2/library/stdtypes.html`

Number Operations

Operators

Operation	Meaning	Example
`+, -, /, *`	standard (note: integer division)	`4 / 3` (=1)
`+=, -=, /=, *=`	standard	`x = 3; x += 1`
`%, %=`	reminder of division	`7 % 2`
`**, **=`	exponent	`4**2`
`//, //=`	floor division	`9.0//2.0`

Use () to mark priority

More information

`http://www.tutorialspoint.com/python/python_basic_operators.htm`

Programming

Scope

Scope in Python is determined by **indentation**:

```
>>> <control-flow statement> :
...     <statement>
...     <statement>
... [RETURN] #marks end of block
```

About indentation

- Good practice in **any** programming language
- **Enforced** in Python (default: 4 spaces)
- Never mix spaces and tabs $\Rightarrow$ editor setup

More information

`http://docs.python.org/2.7/reference/lexical_analysis.html#indentation`

Functions

To create a function use the `def` keyword. Let's try it:

```
>>> def f(x, y=2, z=3):
...     return (x * y) + z
... <RETURN>
```

Call *f*, what happens? → Quiz!

- `f(1.1)`
- `f(1, 5)`
- `f(0, z=4)`

Functions (answers)

To create a function use the `def` keyword. Let's try it:

```
>>> def f(x, y=2, z=3):
...     return (x * y) + z
... <RETURN>
```

Call *f*, what happens? → Quiz!

- `f(1.1)` → `x=1.1; y=2; z=3` → `f=5.2`, output is `float`
- `f(1, 5)` → `x=1; y=5; z=3` → `f=8`, output is `int`
- `f(0, z=4)` → `x=0; y=2; z=4` → `f=4`, output is `int`

Your Modules

You can create your own modules. Try this:

```
$ gedit first.py
```

```
def f(x, y=2, z=3):
    return (x * y) + z
```

Save, the file, open the Python prompt and type:

```
>>> import first
>>> first.f(1.1)
5.2
```

More information

http://docs.python.org/2/tutorial/modules.html

Loops with `for`

To create a `for` loop, use the `for` keyword:

```
>>> for <variable> in <iterable> :
...     <statement>
...     <statement>
... [RETURN] #marks end of block
```

Try this:

```
>>> for k in [1, 2, 3.14, 'bla']:
...     print k
... [RETURN] #marks end of block
1
2
3.14
bla
```

Formal definition

http://docs.python.org/2/reference/compound_stmts.html#the-for-statement

Python: NumPy and `ndarrays`

NumPy and `ndarrays`

- Fundamental package for scientific computing in Python
- Provides a multi-dimensional (or N-dimensional) array type: `ndarray`
- Linear algebra support for these types;
- Vectorized (pre-compiled) routines for most operations.

Python: NumPy and ndarrays

Creating arrays:

```
>>> import numpy
>>> help(numpy.ndarray)
>>> # basics: numpy.ndarray(<SHAPE>, <DATA-TYPE (dtype)>)
>>> X = numpy.ndarray((2,2), dtype=numpy.uint8)
>>> print X
[[168 247]
 [ 20 161]] #notice: uninitialized
>>> X = numpy.zeros((2,2), dtype=numpy.float64)
>>> print X
[[0.,  0.]
 [0.,  0.]]
>>> # other interesting initializers
>>> help(numpy.ones) #equivalent to Matlab's
>>> help(numpy.array) #initializes from almost any object!
```

More information

http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html

Python: NumPy and ndarrays

Interesting methods and properties:

```
>>> X = numpy.array([[1,2,3],[4,5,6]])
>>> print X
[[1 2 3]
 [4 5 6]]
>>> X.reshape(3,2)
array([[1, 2],
       [3, 4],
       [5, 6]])
>>> X.sum()
21
>>> X.mean()
3.5
>>> X.std()
1.707825127659933
>>> X.shape
(2, 3)
>>> X.dtype
dtype('int64')
>>> len(X)
2 #notice: number of rows == dimension 0 extent
>>> X.<TAB> #for much more
```

Python: NumPy and ndarrays

Indexing and slicing (1D):

```
>>> X = numpy.array(numpy.arange(5.))
>>> print X
array([ 0.,  1.,  2.,  3.,  4.])
>>> X[1] #second element
1.0 #note: indexing starts at 0!
>>> X[-1] #last element
4.0
>>> X[1:3] #slice: second and third elements
array([ 1.,  2.])
>>> X[2:] #slice: from the third element
array([ 2.,  3.,  4.])
>>> X[-4:-1] # what do you think happens?
```

Python: NumPy and ndarrays

Indexing and slicing (1D):

```
>>> X = numpy.array(numpy.arange(5.))
>>> print X
array([ 0.,  1.,  2.,  3.,  4.])
>>> X[1] #second element
1.0 #note: indexing starts at 0!
>>> X[-1] #last element
4.0
>>> X[1:3] #slice: second and third elements
array([ 1.,  2.])
>>> X[2:] #slice: from the third element
array([ 2.,  3.,  4.])
>>> X[-4:-1] # what do you think happens?
array([ 1.,  2.,  3.])
```

Python: NumPy and ndarrays

Indexing and slicing (2D, continued):

```
>>> X = numpy.array(numpy.arange(25.)).reshape(5,5)
>>> print X
array([[  0.,   1.,   2.,   3.,   4.],
       [  5.,   6.,   7.,   8.,   9.],
       [ 10.,  11.,  12.,  13.,  14.],
       [ 15.,  16.,  17.,  18.,  19.],
       [ 20.,  21.,  22.,  23.,  24.]])
>>> X[1] #second row
array([ 5.,  6.,  7.,  8.,  9.])
>>> X[-1]
array([ 20.,  21.,  22.,  23.,  24.])
>>> X[:,0] #first column
array([  0.,   5.,  10.,  15.,  20.]) #note: row vector
>>> X[:,1:3] #second and third colums
array([[  1.,   2.],
       [  6.,   7.],
       [ 11.,  12.],
       [ 16.,  17.],
       [ 21.,  22.]])
>>> Z = X[:,1:3]
>>> Z[0,0] = 3.1416
>>> print X # what do you think happens?
```

Python: NumPy and `ndarrays`

Matlab Gotchas

- Array operations are applied element-wise. Example: `a * b` means multiply a with b elementwise. Use the `numpy.matrix` class if you want the default to mean *matrix multiplication* (not advised though);
  - Element Broadcasting
- 0-based indexing (instead of 1-based);
- slicing gives a reference to existing data, **not** a copy.

More information

`http://www.scipy.org/NumPy_for_Matlab_Users`

NumPy: Vectorization

vectorization

- Disclaimer: This is an abuse of the the term in most cases
  - It means "*parallelize*" normally.
- What we mean is: *use pre-compiled routines to operate on matrices and vectors so as to speed-up computation*
- This is mostly deployed for linear algebra operations. Examples:
  - Dot products between vectors, vectors and matrices or matrices;
  - Linear system solving;
  - Value Decomposition;
  - etc.
- Also possible to speed-up simple mathematical operations, such as "*square all values of this array*".

NumPy: Vectorization

We will only cover the dot-product

- NumPy: `http://docs.scipy.org/doc/numpy/reference/`
- Scipy: `http://docs.scipy.org/doc/scipy/reference/`

NumPy: Matrix-Vector operations

Matrix $(m \times n)$ Matrix $(n \times o)$ multiplication

$$A \times B = AB = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ a_{2,1} & \cdots & a_{2,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{bmatrix} \begin{bmatrix} b_{1,1} & \cdots & b_{1,o} \\ b_{2,1} & \cdots & b_{2,o} \\ \vdots & \ddots & \vdots \\ b_{n,1} & \cdots & b_{n,o} \end{bmatrix}$$

$$= \begin{bmatrix} a_{1,1}b_{1,1} + \cdots + a_{1,n}b_{n,1} & \cdots & a_{1,1}b_{1,o} + \cdots + a_{1,n}b_{n,o} \\ \vdots & \ddots & \vdots \\ a_{m,1}b_{1,1} + \cdots + a_{m,n}b_{n,1} & \cdots & a_{m,1}b_{1,o} + \cdots + a_{m,n}b_{n,o} \end{bmatrix}$$

In Python

```
numpy.dot(A, B)
```

NumPy: Matrix-Vector operations

Matrix ($m \times n$) Vector (n) multiplication

$$A \times \mathbf{b} = A\mathbf{b} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

$$= \begin{bmatrix} b_1 a_{1,1} + b_2 a_{1,2} + \cdots + b_n a_{1,n} \\ b_1 a_{2,1} + b_2 a_{2,2} + \cdots + b_n a_{2,n} \\ \vdots \\ b_1 a_{m,1} + b_2 a_{m,2} + \cdots + b_n a_{m,n} \end{bmatrix}$$

In Python

```
numpy.dot(A, b)
```

NumPy: Matrix-Vector operations

Transpose A^T $(n \times m)$ of A $(m \times n)$

$$A = \begin{bmatrix} 49 & 9 & 2 & 6200 \\ 43 & 10 & 10 & 283 \\ 51 & 9 & 3 & 169 \end{bmatrix} \quad A^T = \begin{bmatrix} 49 & 43 & 51 \\ 9 & 10 & 9 \\ 2 & 10 & 3 \\ 6200 & 283 & 169 \end{bmatrix}$$

In Python

`A.transpose()`, or `numpy.transpose(A)`

NumPy: Matrix-Vector operations

Inverse A^{-1} $(n \times n)$ of A $(n \times n)$

$$A \times A^{-1} = I = A^{-1} \times A$$

In Python

`numpy.linalg.inv(A)`, or
`numpy.linalg.pinv(A)` (for the -Moore-Penrose- pseudo-inverse)

NumPy: Vectorization

Remember

Array operations are applied element-wise.

That means:

```
>>> t = numpy.array(numpy.arange(4))
>>> t
array([0, 1, 2, 3])
>>> t*t
array([0, 1, 4, 9])
>>> # use the function numpy.dot() for dot-products
>>> numpy.dot(t, t)
14
>>> # if both objects are 1D, numpy will do what you expect
>>> col = t.reshape(4,1)
>>> # if one of the is a column vector, it may go wrong
>>> numpy.dot(t, col) # ok
array([14])
>>> numpy.dot(col, t) # not ok
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: objects are not aligned
```

NumPy: Vectorization

Can use it with 2D arrays (matrices):

```
>>> m = numpy.array(numpy.arange(4)).reshape(2,2)
>>> m
array([[0, 1],
       [2, 3]])
>>> numpy.dot(m, m)
array([[ 2,  3],
       [ 6, 11]])
>>> # matrix versus vector
>>> v = numpy.array([2.0, 3.0])
>>> numpy.dot(v, m) # means v * m
array([  6.,  11.]) # notice automatic type casting
>>> numpy.dot(m, v) # means m * v', using v's shape
array([  3.,  13.])
>>> numpy.dot(m, v.reshape(2,1)) # means m * v' literally
array([[  3.],
       [ 13.]])
>>> numpy.dot(v.reshape(2,1), m) # not Ok => error
...
ValueError: matrices are not aligned
```

NumPy: Vectorization

Interesting operators and linear algebra:

```
>>> m = numpy.array(numpy.arange(4)).reshape(2,2)
>>> m
array([[0, 1],
       [2, 3]])
>>> m.transpose()
array([[0, 2],
       [1, 3]])
>>> numpy.linalg.inv(m)
array([[-1.5,  0.5],
       [ 1. ,  0. ]]) #notice automatic type casting
```

More information

- `help(numpy.linalg)`
- `http://docs.scipy.org/doc/numpy/reference/routines.linalg.html`

Python: Other Resources

Often, researchers don't know Python is *at least* as powerful as other software packages:

- Vector/Matrix/Array (use NumPy): `http://www.numpy.org`
- Optimization, distance metrics, signal processing, statistics, etc (use SciPy): `http://www.scipy.org`
- Symbolic maths (use SimPy): `http://www.sympy.org/en/index.html`
- Plotting 2D or 3D complex surfaces (use MatplotLib): `http://matplotlib.org`
- Integrated documentation (use Sphinx): `http://sphinx-doc.org`
- Machine Learning (use Bob, Scikit-Learn): `http://scikit-learn.org/stable/`
- Etc.

Free code

After some struggling, you have finally obtained some code from the author, which does not require Matlab

You go and execute it:

```
$ cd ex2
$ cat email2.txt
# figure it out
```

OK, so what?

Here are some questions for you:

- How was the system trained?
- Was there any pre-processing?
- What was the data/protocol used to verify these results?
- Are there untold parameters?
- How was the analysis carried out? What is it?
- Would it work for a different protocol?
- How can I apply those findings on my work?

Figuring things out

Go ahead and figure it out by opening the file provided by the author.

```
$ gedit doit.py
$ # figure it out
```

Figuring things out

Go ahead and figure it out by opening the file provided by the author.

```
$ gedit doit.py
$ # figure it out
```

Problems

Why can't you figure this out that easily?

Figuring things out

Go ahead and figure it out by opening the file provided by the author.

```
$ gedit doit.py
$ # figure it out
```

Problems

Why can't you figure this out that easily?

Need more

You need more than just the bare software:

- You need to code to be organized
- You need to instructions on how to run it
- You need the code to be documented

Work flow

Let's start by organizing the code so it is easy to find things. How would you start it?

Work flow

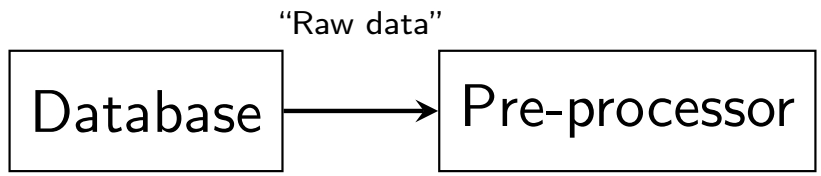
Let's start by organizing the code so it is easy to find things. How would you start it?

Work flow

Let's start by organizing the code so it is easy to find things. How would you start it?

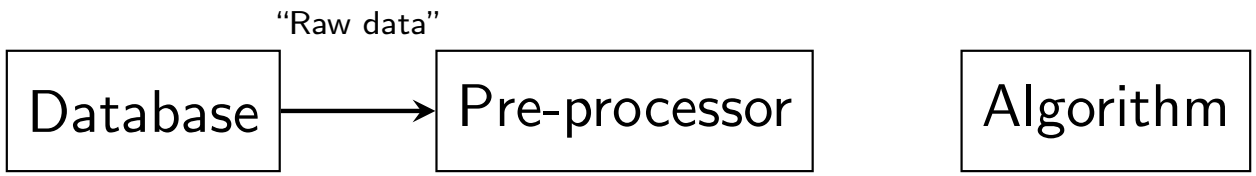
Work flow

Let's start by organizing the code so it is easy to find things. How would you start it?



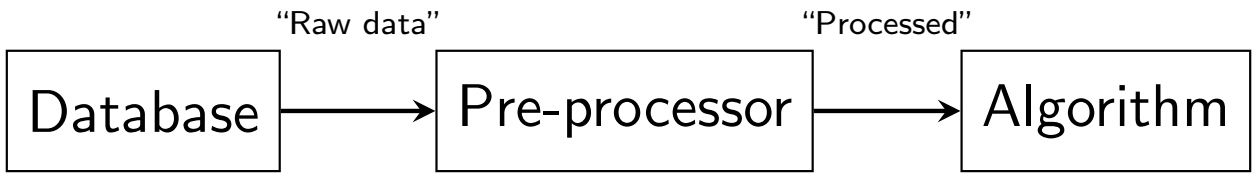
Work flow

Let's start by organizing the code so it is easy to find things. How would you start it?



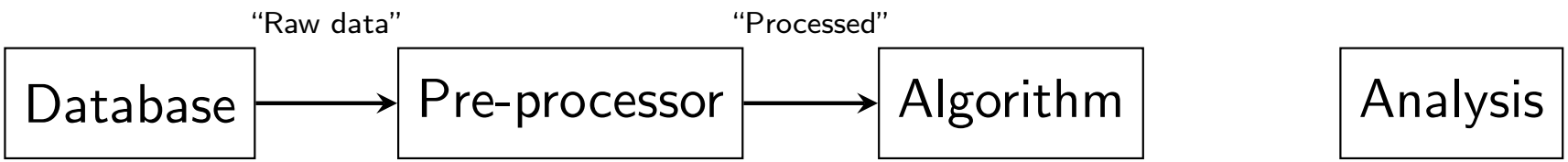
Work flow

Let's start by organizing the code so it is easy to find things. How would you start it?



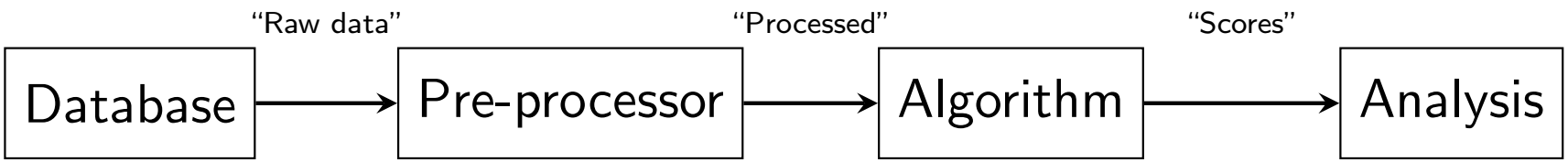
Work flow

Let's start by organizing the code so it is easy to find things. How would you start it?



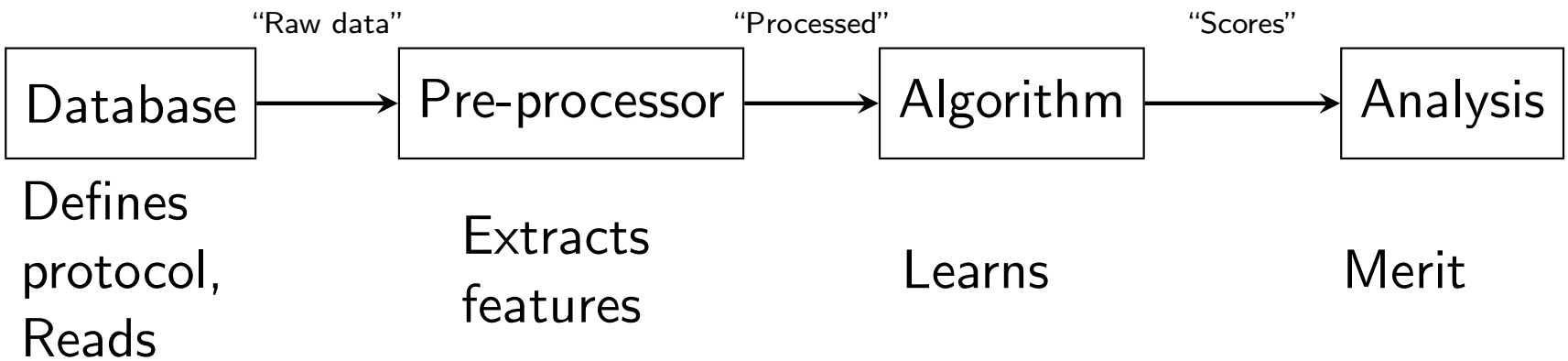
Work flow

Let's start by organizing the code so it is easy to find things. How would you start it?



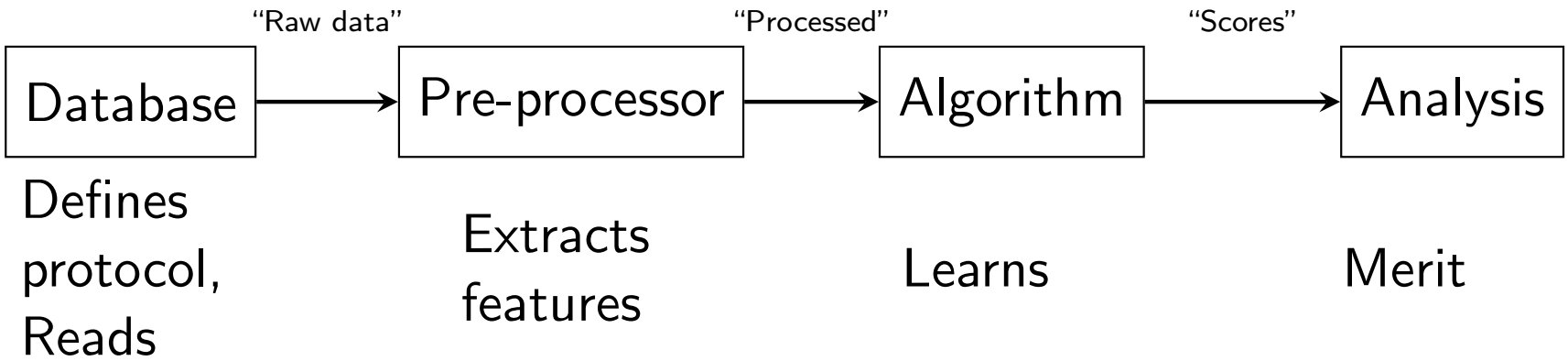
Work flow

Let's start by organizing the code so it is easy to find things. How would you start it?



Work flow

Let's start by organizing the code so it is easy to find things. How would you start it?



How would you break down this problem?

Packaging

This was an engineering solution to this:

- We broke down the problem into smaller, easily solvable units
- Each of these units will (as you'll see), result in a package with an API you can **re-use** in different projects
- Combine all packages into a single **powerful** program that performs your experiment(s)
- Combine several of these programs and generate a **full paper**!

The Database

Where all starts. You need to think about:

1. How is the data going to be served to the algorithm?
2. What is the evaluation protocol?

We'd like to emphasize: Simplicity and reproducibility

The Database: A solution

Here is a way to organize it eloquently:

1. How is the data going to be served to the algorithm?
   - Use native arrays as raw data objects. Example: Iris Flowers $\rightarrow$ 1D arrays with 4 elements; Images $\rightarrow$ 2D arrays, etc.
2. What is the evaluation protocol?
   - Organize the data in subsets
   - Provide functions that return data/label for a given protocol

The Database: A solution

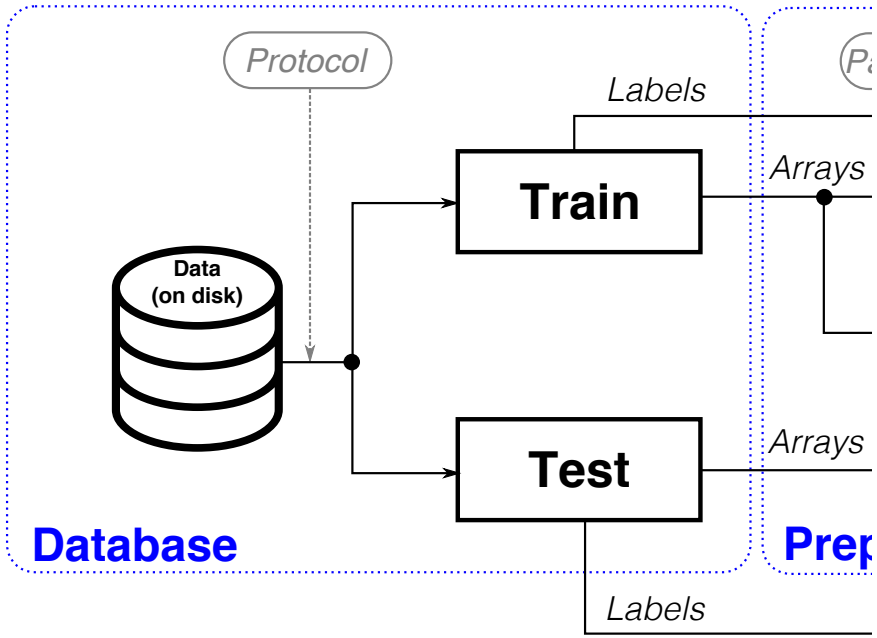
Here is a way to organize it eloquently:

1. How is the data going to be served to the algorithm?
   - Use native arrays as raw data objects. Example: Iris Flowers $\rightarrow$ 1D arrays with 4 elements; Images $\rightarrow$ 2D arrays, etc.
2. What is the evaluation protocol?
   - Organize the data in subsets
   - Provide functions that return data/label for a given protocol

What if...

...you don't have a protocol?
You must create one so that your evaluations are *reproducible*

Database: Sought design



Notice

Despite a current practice in many fields, the use of separate development and test data is not required *for reproducibility*.

Pre-processor

A pre-processing step is used in many methods. In our particular problem, the author applied Z-normalization on the input data only.

Z-norm

If X is a 2D matrix containing **training** examples in each row and one feature per column, then Z-norm:

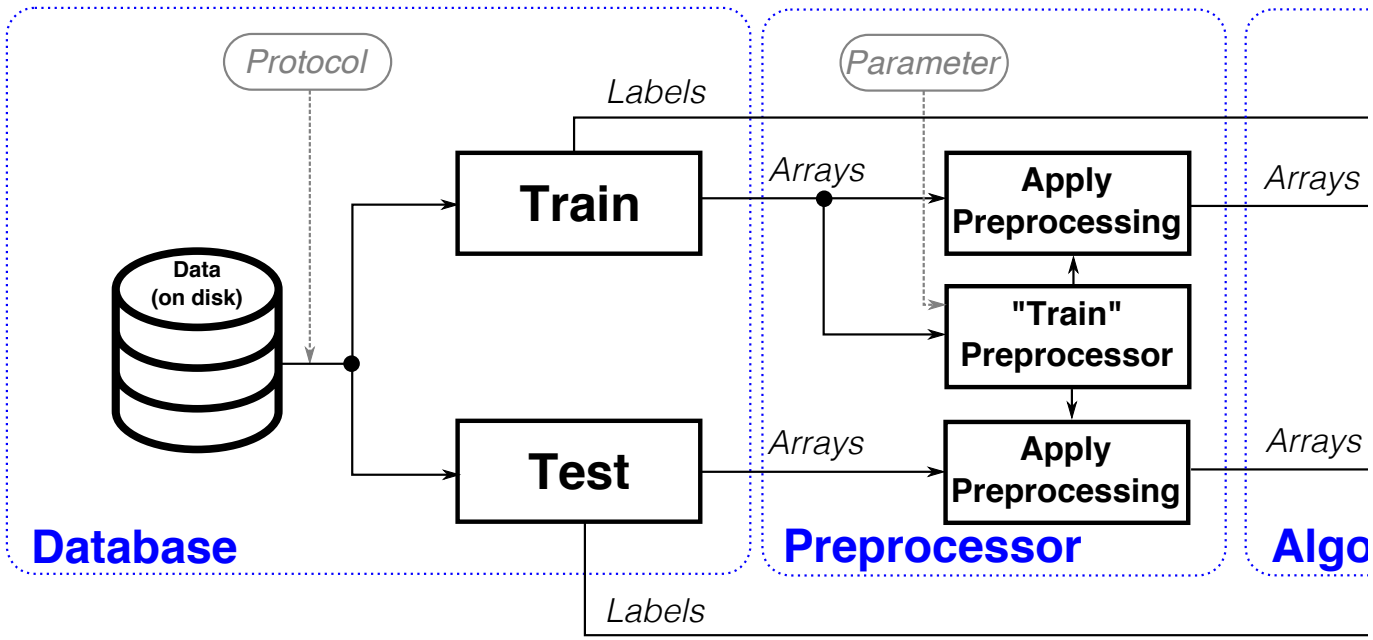
- Estimates the average μ and standard deviation σ across rows of the training (or design) matrix
- Applies the average and deviation to **both** the training and test data:

$$X' = \frac{X - \mu}{\sigma}$$

Other Feature Extraction

Without loss of generality, the pre-processing stage could also extract new features (e.g. apply linear/non-linear transformations) from the data.

Pre-processor: Sought design

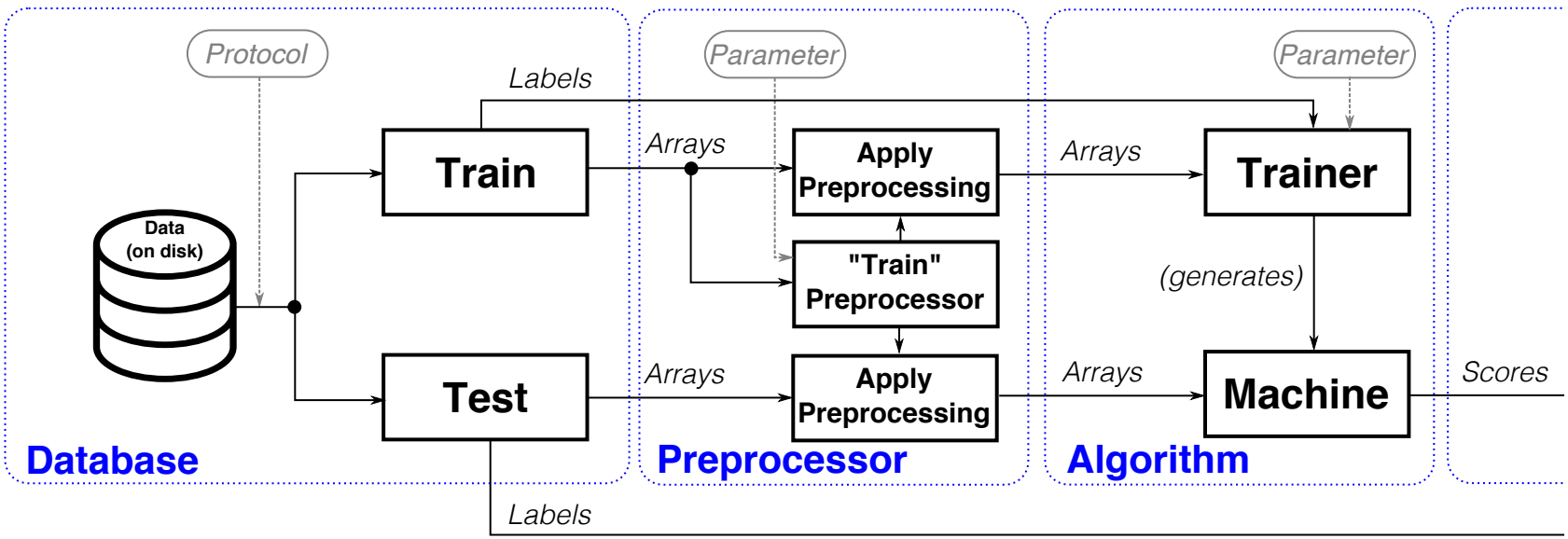


Algorithm

The machine learning algorithm you're going to implement:

- Will use the training data (and labels) to **train** a machine (or set of machines) for your problem
- Training an ML system normally implies setting **parameters** that specify the final machine design (e.g. number of neurons in MLPs, the C parameter in SVMs, etc)
- The machine will be **deployed** on the test data (at least) to produce *scores* for each input array (representing one flower)

Algorithm: Sought design

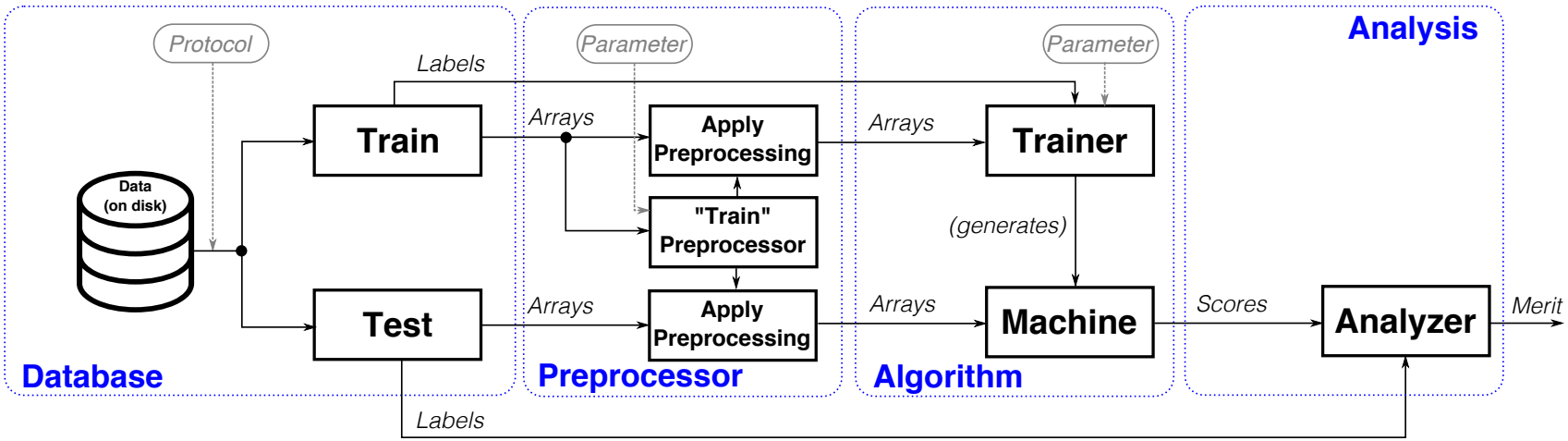


Analysis

Analysis is probably the simpler of the three core functions you'll write:

- For each input sample you analyzed, a single score (scalar) is output by your machine
- For every separate subset (development or test), your analyzer inputs the scores and *ground truth* and will address your merit figure
- The output of the analysis should be a **simple**, easily digestable number or plot you can use to verify your hypothesis or not
- You can *compare* systems that are trained and tested on the same Protocol by comparing their merit figures

Full System: Sought design



Notice

The way we have organized things is pretty general and fits a great number of problems in Machine Learning and Pattern Recognition!

With minor modifications, you can adapt this design to your own problems.

Documentation

Good coding habits and documentation standards:

PEP8: `https://www.python.org/dev/peps/pep-0008/`

- Use spaces for indentation (4)
- Use docstrings
- Functions should be like `my_function()`
- Classes should be like `MyClass`
- Avoid `from bla import *`

Preferred (PEP8) Docstring usage

```
def my_function(par1, par2):
    """The first line of the docstring contains a short description.

    Then we skip one line and continue on the following. The short
    description should only have ~50-80 characters. It should be succint
    and describe what the object does. The long string should be a bit
    more verbose.
    """

    # skip one line after the docstring to start your code
    pass


def other_function(par2):
    """Skip 2 lines between objects for clarity"""
    pass
```

How to use docstrings (1)

Docstrings are free text and that means you can use it in any way you like.

Currently, there seems to be 2 "preferred" ways to organize docstrings for best readability and automated parsing:

- The Google Standard: `http://sphinxcontrib-napoleon.readthedocs.org/en/latest/example_google.html`
- The NumPy Standard: `http://sphinxcontrib-napoleon.readthedocs.org/en/latest/example_numpy.html#example-numpy`

In this course, we'll adopt the "Google" one. Let's have a look at it.

ReestructuredText (reST)

Standard for Python Documentation (see PEP287):
ReestructuredText (`reST`)
(`http://docutils.sourceforge.net/rst.html`)

- It is a mark-up language: you write in plain text, parsers can generate HTML or even LaTeX output out of it!
- Quick Reference: `http://docutils.sourceforge.net/docs/user/rst/quickref.html`
- Primer: `http://sphinx-doc.org/rest.html`
- Full Specifications: `http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html`

Tip

We format docstrings using reST so as to generate beautiful and more readable documentation.

reST Paragraphs

Paragraphs are the basic block in reST

```
This is the first paragraph. This paragraph can have as many lines as
I want. It will end when I skip, at least, one line.

This is the second paragraph.
```

reST Inline Markup

You can mark words as you type:

- one asterisk: `*text*` for emphasis (*italics*),
- two asterisks: `**text**` for strong emphasis (**boldface**), and
- backquotes: ``` ``text`` ``` for code samples (`code font`)

```
While writing, one can use **strong**, *italic* or ``code``.
```

reST Lists and Enumerations

List markup is natural: just place an asterisk at the start of a paragraph and indent properly.

```
Lists:

* First item
* Second item
  * nested item 1
  * nested item 2

Enumerations:

1. First item
2. Second item
   * Can also nest lists
   * Second item
```

reST Definitions

Definition lists are created as follows:

```
term (up to a line of text)
  Definition of the term, which must be indented

  and can even consist of multiple paragraphs

next term
  Description.
```

Tip

Definition lists will be heavily used to describe the parameters of methods and functions.

reST Source

Source code can be created with an indented paragraph preceeded by a double-colon (::) on the previous paragraph:

```
This is a normal text paragraph. The next paragraph is a code sample::

   import sys
   print sys.version

   #It can span multiple lines (just keep the indentation)

This is a normal text paragraph again.
```

Notice

Stock reST does not process the code blocks. It just indents them and make sure the selected font to represent it is monospaced.

reST Tables (1)

Two forms of tables are supported. For grid tables, you have to "paint" the cell grid yourself. They look like this:

```
+------------------------+------------+----------+----------+
| Header row, column 1   | Header 2   | Header 3 | Header 4 |
| (header rows optional) |            |          |          |
+========================+============+==========+==========+
| body row 1, column 1   | column 2   | column 3 | column 4 |
+------------------------+------------+----------+----------+
| body row 2             | ...        | ...      |          |
+------------------------+------------+----------+----------+
```

reST Tables (2)

Simple tables are easier to write, but limited: they must contain more than one row, and the first column cannot contain multiple lines. They look like this:

```
=====  =====  =======
A      B      A and B
=====  =====  =======
False  False  False
True   False  False
False  True   False
True   True   True
=====  =====  =======
```

reST Hyperlinks (1)

The easiest way to write hyperlinks is just to type them!

```
If it is a simple URL you want to link, you can just type it
like in http://example.com. The parser will figure it out.

Otherwise, you can defined the link text that will displayed by writing
‘Link text <http://example.com>‘.
```

reST Hyperlinks (2)

You can also separate the link and the target definition, like this:

```
This is a paragraph that contains `a link`_.

.. _a link: http://example.com/
```

reST Sections (1)

Section headers are created by underlining (and optionally overlining) the section title with a punctuation character, at least as long as the text:

```
================
This is a heading
================
```

Tip

There exists no special characters. However, for Python documentation, this convention is used:

- # with overline, for parts
- * with overline, for chapters
- =, for sections
- -, for subsections
- ^, for subsubsections
- ¨, for paragraphs

reST Sections (2)

Here is a longer example:

```
##############################
Reproducible Research Course
##############################

This package contains material for a 12-hour hands-on course on
Reproducible Research using Python, Bob and the BEAT platform.

Build notes
===========

In order to build an abstract for the course, do the following::

  $ make abstract
```

Tip

You don't have to use reST only to document Python code, you can use it everywhere a little README is required.

reST Images

reST supports an image directive, used like so:

```
.. image:: gnu.png
   :height: 100px
   :width: 200 px
   :scale: 50 %
   :alt: alternate text
   :align: right
```

Tip

The values following the image declaration are optional. If you need a caption, use a `figure` instead (`http://docutils.sourceforge.net/docs/ref/rst/directives.html#figure`).

reST Citations

Standard reST citations are supported. Use them like so:

```
Lorem ipsum [Ref]_ dolor sit amet.

.. [Ref] Book or article reference, URL or whatever.
```

reST Comments

Every explicit markup block which isn't a valid markup construct is regarded as a comment. For example:

```
.. This is a comment.

..
   This whole indented block
   is a comment.

   Still in the comment.
```

reST Admonitions

Indented, specially treated, blocks of text are called “admonitions”. For example, you can create a note like this:

```
.. note:: This is a note admonition.
   This is the second line of the first paragraph.

   - The note contains all indented body elements
     following.
   - It includes this bullet list.
```

Tip

Other types of admonitions are natively supported: “attention”, “caution”, “danger”, “error”, “hint”, “important”, “note”, “tip”, “warning”.

Tools for reST

The Python package "`docutils`" provides basic tools for the parsing of reST code. It can generate HTML output, for example:

```
$ cd ex3
$ rst2html README.rst > README.html
$ xo README.html
# Now reproduce it!
# Inspect the package and the source code
```

Wrapping-up

What are your impressions up to this point?

- Do you think that the author is doing better?
- Do you think all elements provided are enough?

Next course:

- Unit testing
- Version control
- Packaging Python code
- Deploying Python code
- Continuous Integration
- Bob

Going back to the last exercise

```
$ cd ex3
$ python ./paper.py
...
```

Have you noticed something weird?

Going back to the last exercise

```
$ cd ex3
$ python ./paper.py
...
```

Have you noticed something weird?

Well...

... the CER is always zero for one!

There was an error!

Your colleague, unwillingly, has introduced a **programming error** on his analyzer code.

Can you guess where it is?

Tip

In Python, a division with integers, gives an integer output!

There was an error!

Your colleague, unwillingly, has introduced a **programming error** on his analyzer code.

Can you guess where it is?

Tip

In Python, a division with integers, gives an integer output!

```
return errors/len(prediction)
```

To err is human

Unfortunately, programming is a tricky bussiness. You're bound to make mistakes:

- Buggy, untested cases which become important later
- Naive "improvements" which cause errors somewhere else
- Bug fixes may also introduce more bugs
- ...

But how to prevent these errors?

To err is human

Unfortunately, programming is a tricky bussiness. You're bound to make mistakes:

- Buggy, untested cases which become important later
- Naive "improvements" which cause errors somewhere else
- Bug fixes may also introduce more bugs
- ...

But how to prevent these errors?

For starters...

(from experience) Be humble! Believe you, like anyone else, can make mistakes!

Defensive Programming

That basically means: "Prevent problems before they appear". For example:

```
def division(a, b):
    return a/b
```

Could be best written like this:

```
def division(a, b):
    assert b != 0
    return a/b
```

Unit Testing

In unit testing we test each functional unit of our code with a tuple of **expected inputs and outputs**. The behaviour of the unit must respect the expected input and output or the test **fails**.

Each unit test is encoded in a single function. For example:

```
def test_one():
    assert analysis.CER(numpy.array([1,1]), numpy.array([0,1])) == 0.5
```

Fix the code

Your task now is to fix the code so that the tests run correctly.

```
$ cd ex4
$ gedit README.rst #figure out how to run tests!
FF
...
Ran 2 tests in 0.002s

FAILED (failures=2)
$ gedit analysis.py
# re-run tests until all good
```

Solution

Here is the solution to the problem:

```
return float(errors)/len(prediction)
```

Running the tests should now get you:

```
$ nosetests ./test.py
..
----------------------------------------------
Ran 2 tests in 0.001s

OK
```

Unit Testing in Python

You need a couple of tools:

- A program to run your tests (a.k.a. "test runner")
- A set of utility functions that allow you to check for conditions

Unit Testing in Python

In Python, there are two leading packages for this:

- The native `unittest` module:
  `https://docs.python.org/2/library/unittest.html`
  (utility functions)
- And the external package `nose`:
  `https://nose.readthedocs.org/en/latest/`

In this course, we'll use `nose` because it is simpler.

Introduction to Nose

The package `nose` extends the native `unittest` with plugins and other useful functionality, also providing a simpler way to write tests.

The `nose` runner searches for objects on a given directory structure that match a search expression (basically, anything with `test` on their name). It considers those objects to be test units and executes them.

`nose` provides a set of utilities to help encoding test units in a readable manner. They mostly sit in the subpackage `nose.tools`.

Basic Skeleton of a test program

```
import nose.tools

def test_one(): #no parameters!
    assert True #use the stock Python assertion

def test_two(): #notice no parameters!
    assert False #this test will fail

def test_equal():
    a = 5
    b = 5
    assert a == b, "%r != %r" % (a, b)

def test_same():
    a = 5
    b = 5
    nose.tools.eq_(a, b)

@nose.tools.nottest
def test_skipped():
  assert False
```

Tests with setup/teardown

Some tests require a common setup and/or clean-up routines to take place before and/or after the test is run.

```
value = False

def setup():
    global value
    value = True

def cleanup():
    value = False

@with_setup(setup, cleanup)
def test_value()
    assert value #will pass
```

Other useful testing tools

`nose.readthedocs.org/en/latest/testing_tools.html`

- `nose.tools.raises`: makes sure the test raises the said exception
- `nose.tools.timed`: makes sure the test takes, at most, a given amount of time

Exercise

Open the file `test.py`. Try to figure it out.

```
$ gedit test.py
# do you understand what will happen?
```

How much should I test?

This is a common question when you are confronted for the first time with unit testing.

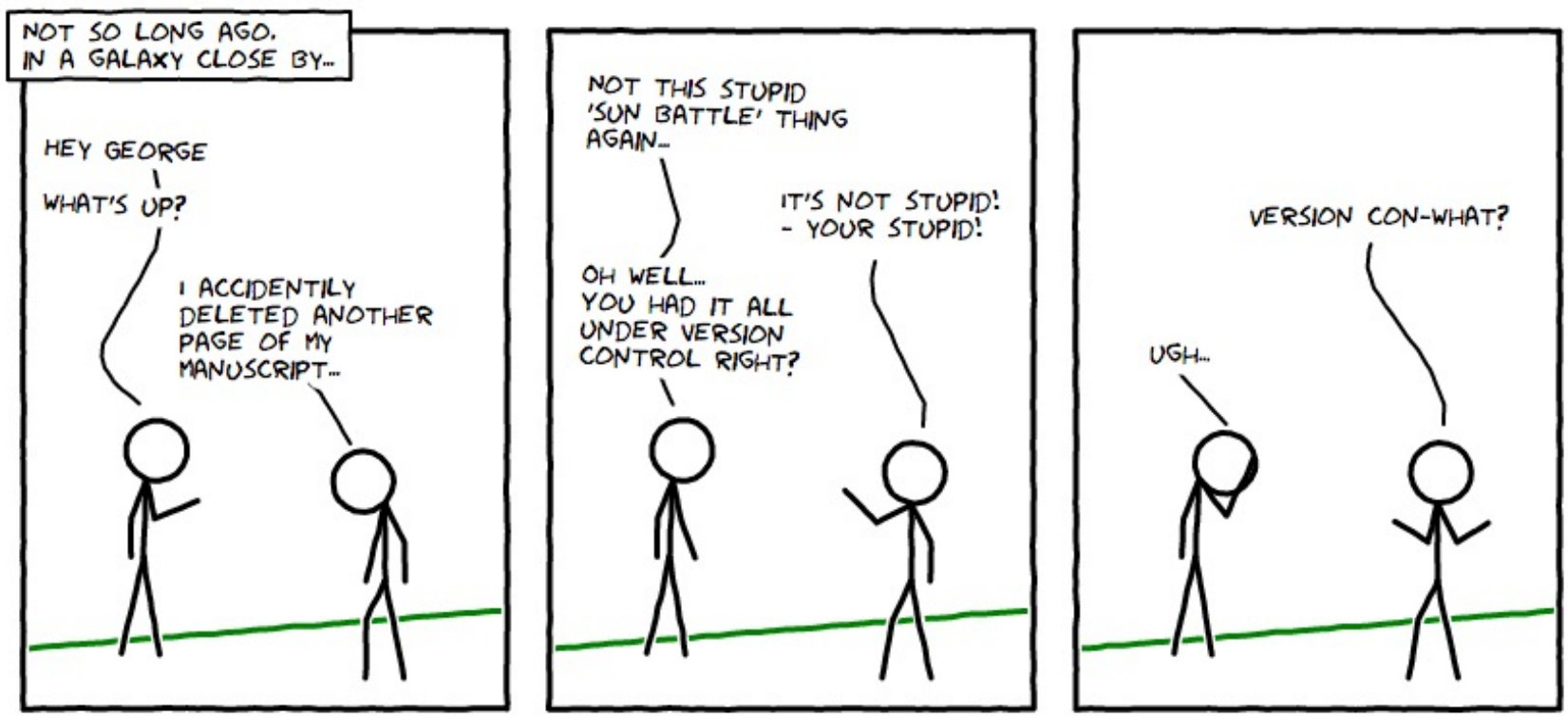
There is no *right* answer to this, but:

- Avoid complicated functions, but if you can't, test them thorougly
- Use test units to *pin* behaviour you'd like to be tested when changes occur
- Test as much as you can (don't believe you aren't prone to errors!)

Tip

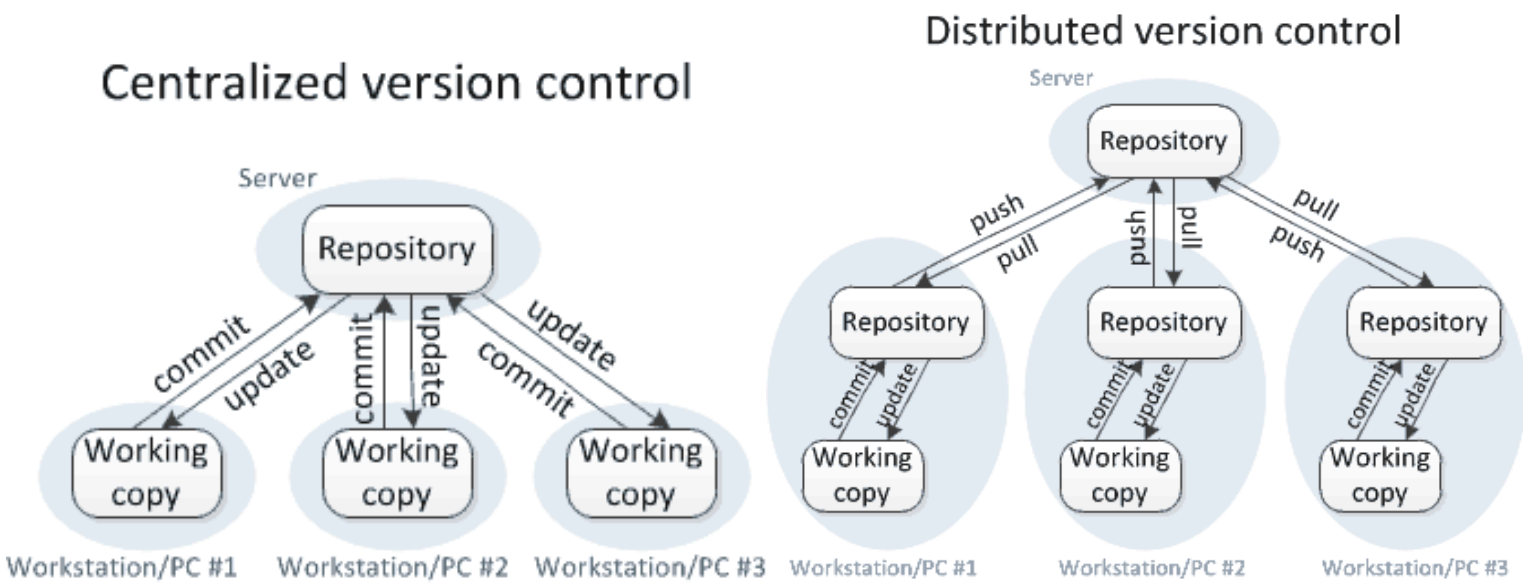
Test units allow for you to have an overall feeling everything looks OK, **quickly**.

Revision/Version Control



What is Revision Control?

- Management of changes to documents/code or any sorts of collections of information
- It is normally done by specialized software packages such as `git`
- There are two types:
  - Centralized: Revision history is kept on a remote server
  - Distributed: History is copied with the repository



Why is it necessary?

Imagine a world w/o version control:

- You released version 1.0 of your software. It has a bug. Which other versions are affected?
- When was the last time I touched this file? Which changes did I do?
- You introduced a bug on the software: Where is that *fracking* backup?

It is possible!

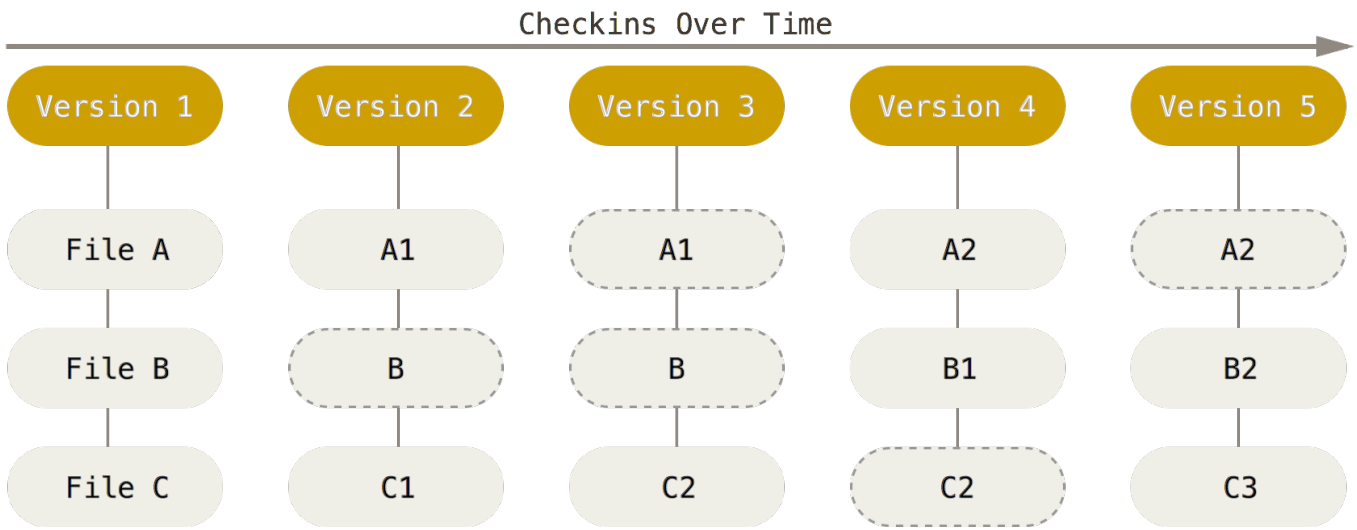
Actually, the Linux project stayed 11 years w/o version control!

This was possible thanks to an "extremely" organized procedure for diff/patching changes that gave birth to what is "Git" today!

Git

What is Git?

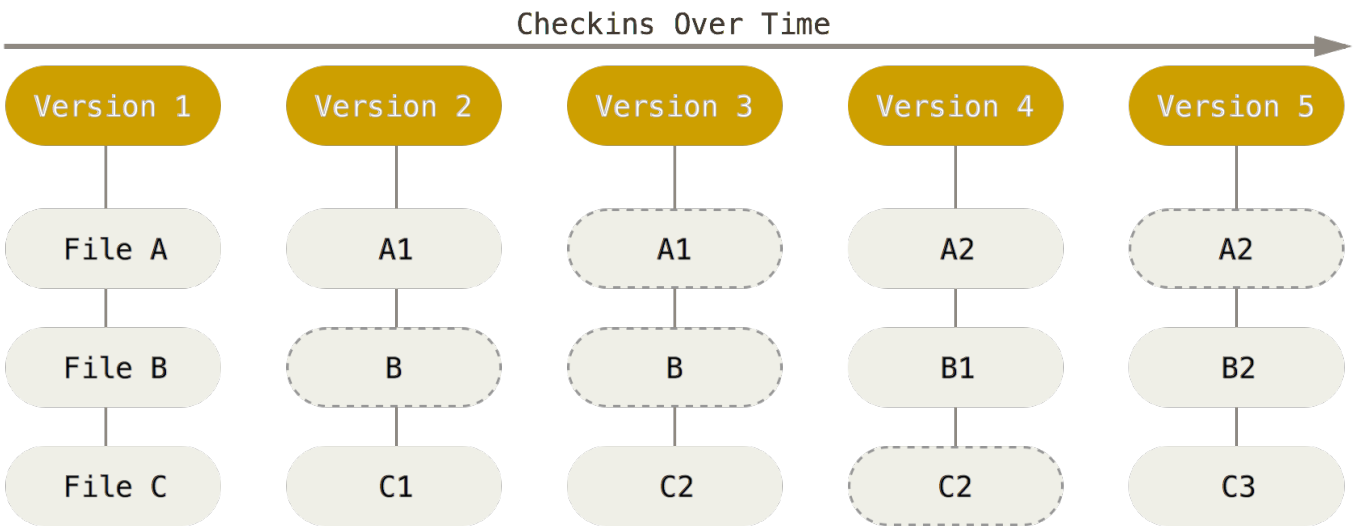
Git is a distributed revision control system. It keeps snapshots of **the entirety** of your versioned directory through time using patches.



Git

What is Git?

Git is a distributed revision control system. It keeps snapshots of **the entirety** of your versioned directory through time using patches.



Old tools, new usage

In order to create a snapshot, git uses *diffs*, *patches* and (SHA-1) *hashes*

Diff/Patch

A *diff* is a set of textual differences between files.

`file1.txt:`

```
I need to go to the store.
I need to buy some apples.
When I get home, I'll wash the dog.
```

`file2.txt:`

```
I need to go to the store.
I need to buy some apples.
I also need to buy grated cheese.
When I get home, I'll wash the dog.
```

To create a *patch*, use the `diff` command:

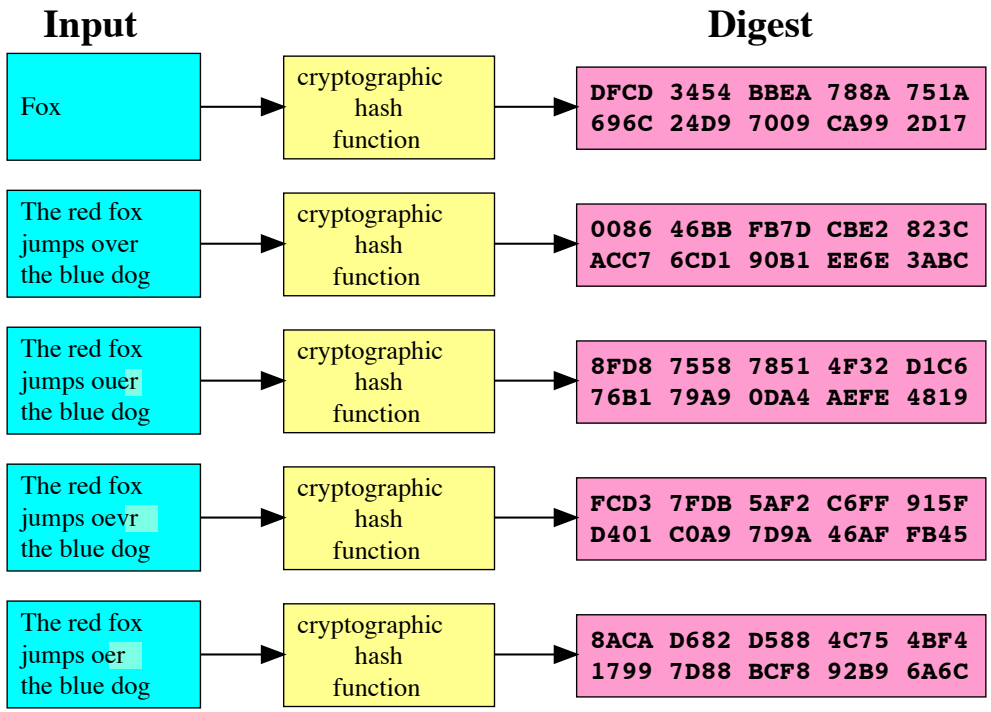
```
$ diff file1.txt file2.txt > patch.txt
$ cat patch.txt
2a3
> I also need to buy grated cheese.
```

Translating

After line 2 in the first file, a line needs to be added: line 3 from the second file.

Hash

A (crypto) hash function is a function that can be used to map digital data of arbitrary size to a fixed length string, that is practically impossible to invert.



Notice that small changes on the input make the hash change a lot.

Hash (collision)

Nearly impossible to clash

It is nearly **impossible** that two natural sequences collide on the **same** repository.

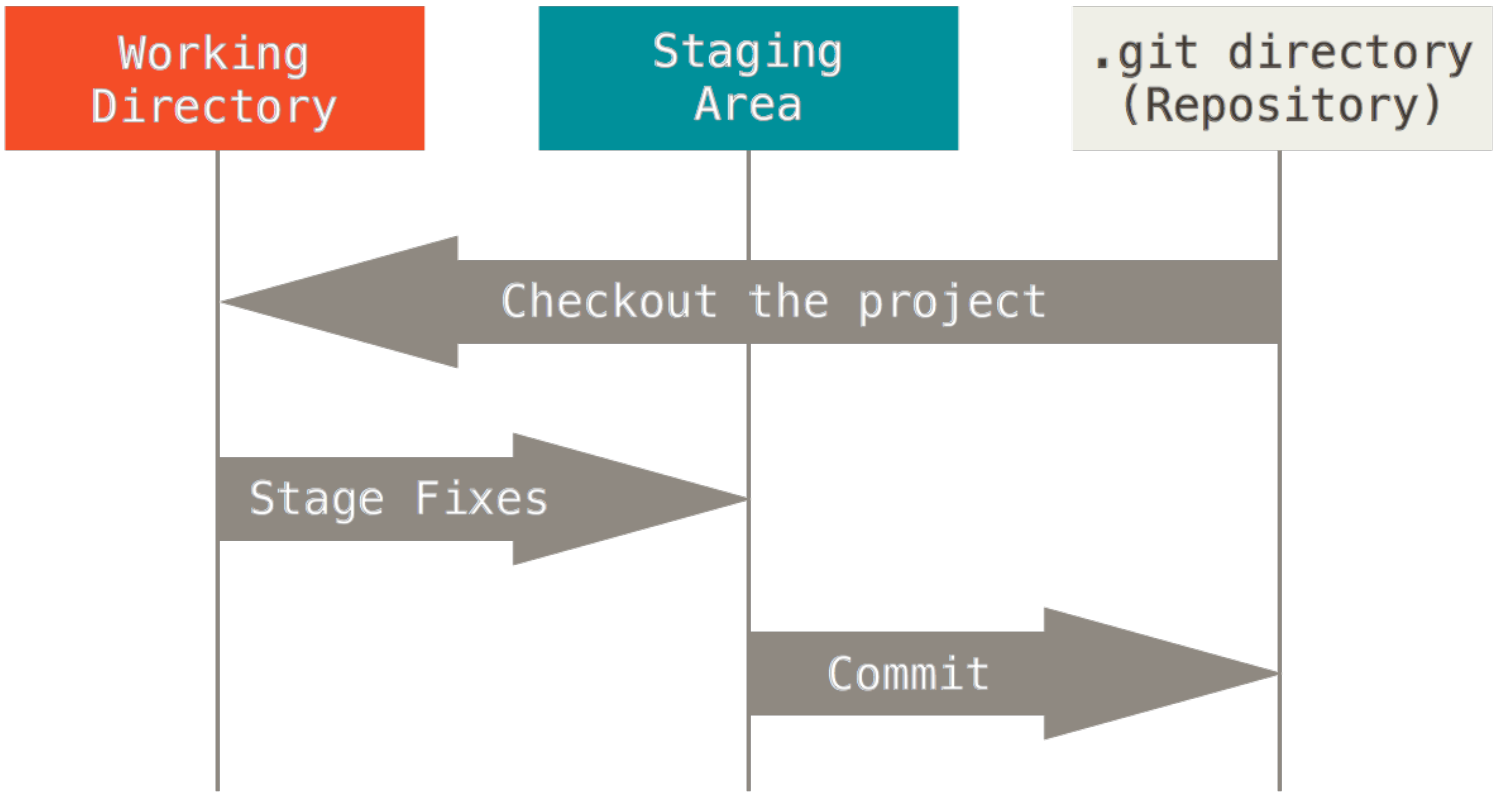
If all world population would be developers and every one of them would commit to the **same** repository every second, the probability of 50% collision would be reached in[9]:

$$6.6 \times 10^6 \text{years}$$

[9] http://diego.assencio.com/?index=eacd6eedf46c9dd596a5f12221ad15b8

Git states

Git contains 3 states for your project.



Git workflow

Easy

- You modify files in your working directory.
- You stage the files, adding snapshots of them to your staging area.
- You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

Configuring `git` (first time only)

To tell git it should log every commit using your name and e-mail, you need to configure it once:

```
$ git config --global user.name "First Last"
$ git config --global user.email first.last@example.com
# to list all configuration set for you
$ git config --list
```

Tip

Tab-like auto-completion works out-of-the-box!

Initializing a new repository

To initialize a repository just use `git init`. Let's try it!

```
$ cp -r ex4 myproj
$ cd myproj
$ git init
Initialized empty Git repository in ...
```

What is staged?

The `status` command gives an overview of the staging area.

```
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add ...
```

Let's do the first commit

The `commit` command instructs git to register the snapshot (patch) to its `.git` directory.

```
$ git add . #adds all files to staging area
$ git commit -m "My first commit with git"
$ git status
On branch master
nothing to commit, working directory clean
```

Tip: Configuring the default editor

```
$ git config --global core.editor /usr/bin/nano #default
$ git config --global core.editor /usr/bin/gedit
$ git config --global core.editor /usr/bin/vim
$ git config --global core.editor /usr/bin/gvim
```

Making changes

The power of version control can be shown when you make changes. Let's go back to the file `analysis.py` and undo the fix you just did, so the bug we corrected is re-introduced.

```
$ gedit analysis.py #undo the change
# now we use git to problem for the modification
$ git diff
diff --git a/analysis.py b/analysis.py
index d4d5b3e..2697486 100644
--- a/analysis.py
+++ b/analysis.py
@@ -20,4 +20,4 @@ def CER(prediction, true_labels):
    """

    errors = (prediction != true_labels).sum()
-   return float(errors)/len(prediction)
+   return errors/len(prediction)
```

Note: The commit hashes my differ

Commiting changes (faster)

You can stage and commit changes with one command.

```
$ git commit -m "Re-added nasty bug" -a
```

Logs and Diffs

At all times, you have access to history and can revert back.

```
$ git log --oneline
df7bc06 Re-added nasty bug
ccb2d42 My first commit with git
$ #figured out I re-added the bug, so will revert
$ git revert df7bc06
# edit the comment, and save (<ESC>:wq)
$ git log --oneline
a43f4c6 Revert "Re-added nasty bug"
df7bc06 Re-added nasty bug
ccb2d42 My first commit with git
$ git diff df7bc06..a43f4c6
# OK!
```

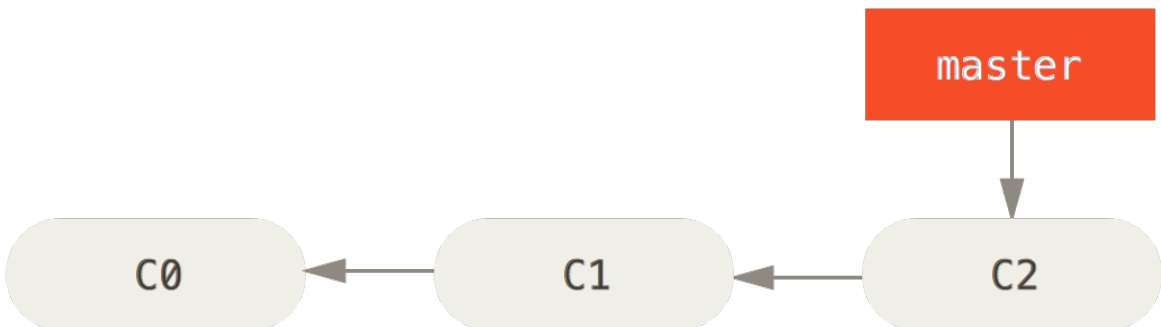
Tags

Git allows you to set labels to refer to repository versions (instead of hash initials). You should use the `tag` command to do so.

```
$ git tag final #makes final == a43f4c6..
$ git tag buggy df7bc06 #makes buggy == df7bc06...
$ git diff buggy..final
...
$ git tag
buggy
final
```

Branches

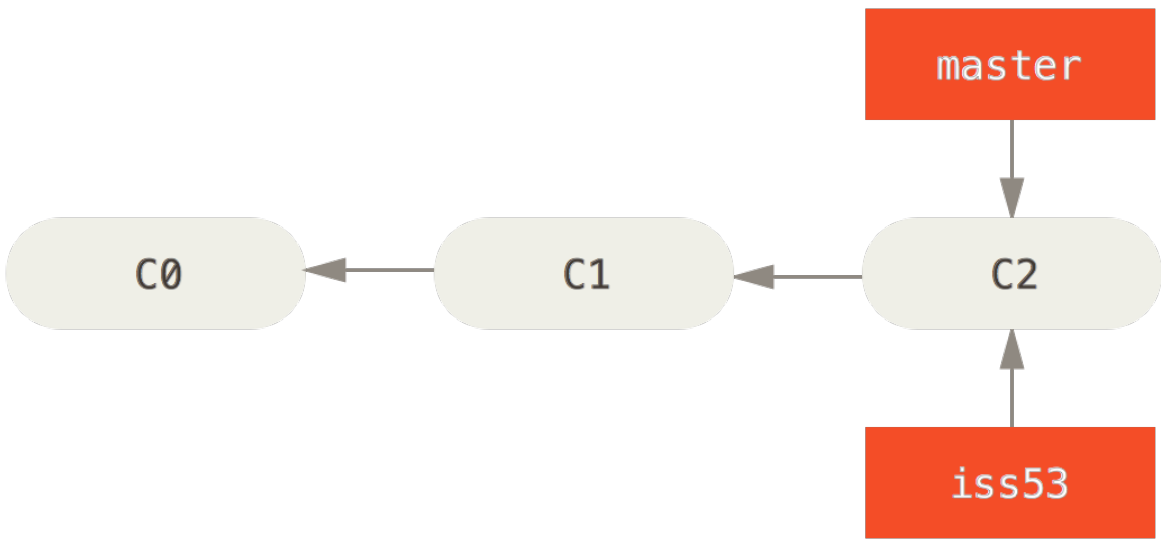
A branch is marker that identifies where the current changes will be patched on. By default, there is only one branch (called `master`) on a git repository.



Branches (2)

You may create a new branch, to develop something new while keeping the master stable.

```
$ git branch iss53
```



Branches (3)

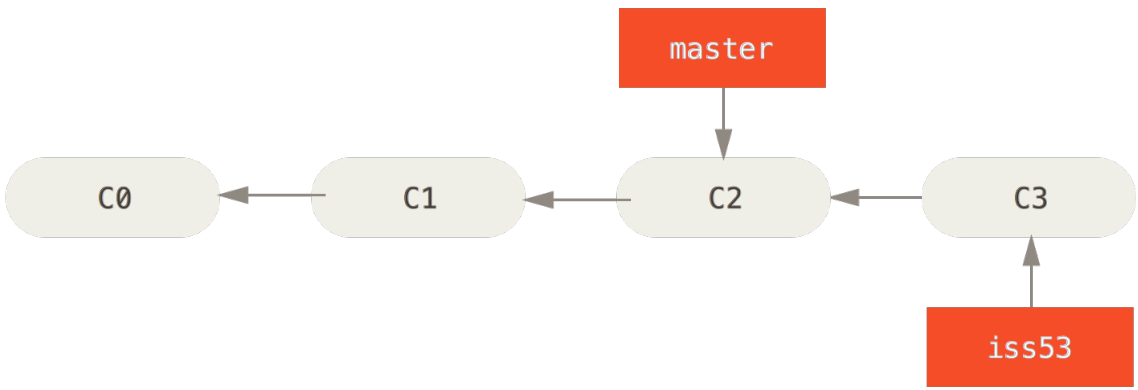
To tell Git to consider a new branch as the default for commiting, use the `checkout` command:

```
$ git checkout iss53
Switched to a branch "iss53"
```

Branches (4)

If you commit a change, only the marker `iss53` will be modified:

```
$ gedit ...
$ git commit -m "My change to the special branch" -a
$ git log --oneline #has the new commit
...
$ git log --oneline master #as before
...
$ git checkout master #To go back to the master
$ git merge iss53 #To merge all changes back
$ git branch -d iss53 #To remove the branch
```



How to Properly Use Tag/Branches

As rule of thumb:

- Everytime you make a **new release**, you **tag** your repository so you know what was distributed
- You use branches to:
  - Test new features w/o disturbing the stability of `master`
  - Fix problems with old versions of the software:

```
$ git branch old-release tag-1.2.4
$ git checkout old-release #state of version 1.2.4
# edit the changes
$ git commit -m "..."
# release version 1.2.5 from that point
```

More Git

Here is a list of resources if you're interested to know more about this powerful tool:

- Reference website: `https://git-scm.com/documentation`
  - Detailed tutorials
  - Videos
  - Reference documentation

  videos)
- ProGit Book (2nd Edition): `http://git-scm.com/book/en/v2`
- MOOC: `https://www.codeschool.com/courses/try-git`
- YouTube clips from GitHub: `https://www.youtube.com/user/GitHubGuides`

GitHub

GitHub is a website that integrates a powerful interface to git repositories, a system of wiki pages and integrated issue lists.

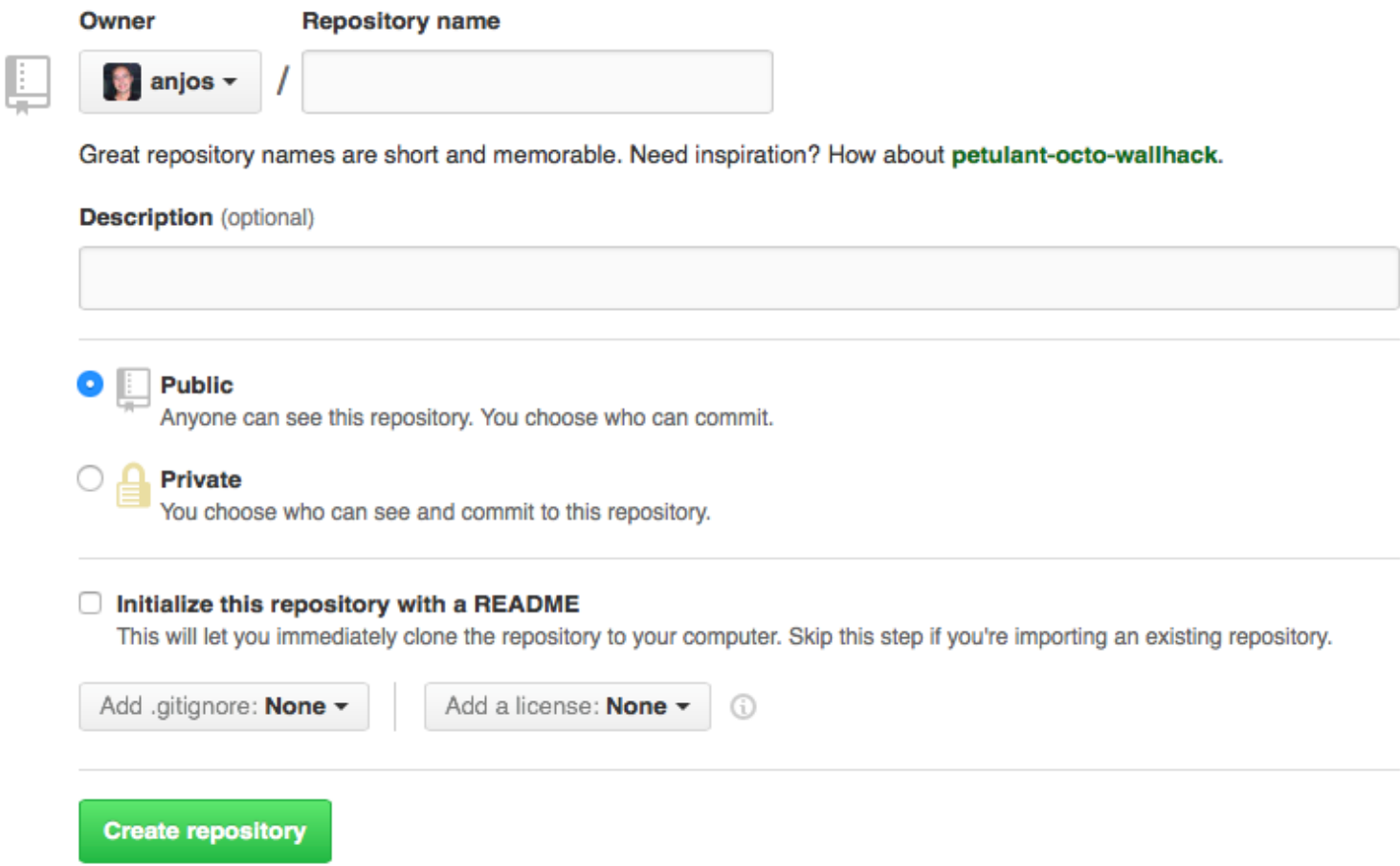
- It's free for open-source projects
- Up to 10 private repositories for academic usage
- Allows easy code sharing and free web-hosting for your projects
- Integrates wiki so you can setup various sorts of guides
- Integrates an issue tracker so people can report bugs and you can keep track of development
- Allows the creation of a website for your project for free (GitHub pages)

Interact

Open a web-browser and go to `https://github.com`.
Open an account.

GitHub: New repository

We're going to move your "`myproj`" project into GitHub. Create a new repository by clicking on the + sign on the top-left.



GitHub: Push the code

You can now push your local code to the remote repository.

```
$ git remote add origin https://github.com/USERNAME/myproj.git
$ git push -u origin master
Username for 'https://github.com': USERNAME
Password for 'https://USERNAME@github.com':
Counting objects: 14, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (14/14), done.
Writing objects: 100% (14/14), 8.34 KiB | 0 bytes/s, done.
Total 14 (delta 3), reused 0 (delta 0)
To https://github.com/anjos/myproj.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

Reload GitHub

Reload the GitHub repository page and see what happens!

GitHub: Cloning/Pulling code

You told about Git/GitHub to your anonymous colleague - he decided to use it for their code. Use the `clone` command to retrieve the new package:

```
$ cd .. #go back to the root of the Course exercises
# Let's download a repository
$ git clone https://github.com/anjos/rrbasic.git ex5
$ cd ex5
$ nosetests ./test.py
$ python ./paper.py
```

GitHub: Cloning/Pulling code

You told about Git/GitHub to your anonymous colleague - he decided to use it for their code. Use the `clone` command to retrieve the new package:

```
$ cd .. #go back to the root of the Course exercises
# Let's download a repository
$ git clone https://github.com/anjos/rrbasic.git ex5
$ cd ex5
$ nosetests ./test.py
$ python ./paper.py
```

Notice

- You were already able to reproduce somebody's (very simple) work in pretty quickly
- This is (nearly) maintenance free: You can have as many repositories on GitHub as you like

GitHub: Explore more!

Here is a list of resources you should explore:

- Issue Tracker: `https://guides.github.com/features/issues/`
- Wiki: `https://help.github.com/articles/about-github-wikis/`
- Host your project's website: `https://pages.github.com`
- Setup SSH access: `https://help.github.com/articles/generating-ssh-keys/`
- Private repositories for academic usage: `https://education.github.com`
- Be Social: `https://help.github.com/articles/be-social/`

What to do?

Here is a list of tips:

- **Always** keep your source code under revision control
- **Commit** often
- **Tag** to keep track of important moments (paper status for example)
- Use **Branches** to apply fixes to distributed software
- Use GitHub as much as you can to host your code. It is free, robust and requires no maintenance efforts from you!
- Alternative services:
  - Atlassian BitBucket: `https://bitbucket.org`
  - GitLab: `https://gitlab.com/users/sign_in` (also installable at your institution!)

Packaging - Why?

Remember from the README?

What happens if?

This is a simple project with a small number of dependencies. What if:

- You have a project with several dependent packages?
- These packages depend on other packages being installed?

Messy

You guessed it right!

Python Packaging

Python has a standard for tracking the inter-dependencies between packages and more!

- Dependence tracking: `https://pypi.python.org/pypi/setuptools`
- Version numbering: `https://www.python.org/dev/peps/pep-0386/`
- Package Repository: `https://pypi.python.org/pypi`

Let's explore each of these concepts and package our code!

Anatomy of a Python Package

A Python Package is a zip or a tar archive containing an organized directory structure.

```
.
+-- MANIFEST.in     # extras to be installed, besides the Python files
+-- LICENSE         # licensing terms (recommended)
+-- README.rst      # basic information
+-- setup.py        # installation + requirements
+-- <package>/      # your code in a directory
|   +-- __init__.py # package marker
|   +-- ...         # other files
```

The setup.py file

Tells setuptools what is inside the package, how to install it.

```
# you told the author about setuptools!
# it is almost a dream - they took it:
$ git clone https://github.com/anjos/rrpack.git ex6
$ cd ex6
$ ls
README.rst  setup.py  rr/  LICENSE  MANIFEST.in
$ ls rr/
test.py          database.py  analysis.py   __init__.py
preprocessor.py  data.csv     algorithm.py  paper.py
$ gedit setup.py #open it!
```

Note

There are a couple of other files in the root directory of this package. We'll talk about them later.

Other observations

Here are a few more insights:

- The code for the package was moved to a subdirectory called `rr`. An empty `__init__.py` marks this is a package.
- The file `rr/paper.py` (our script) now imports modules relatively to its location
- The file `MANIFEST.in` lists **extra** non-python files to be shipped with the package
- It is prudent to ship a `LICENSE` file together with your code. This is also marked on the `setup.py`
- If you open the `rr/database.py` file, you'll notice we now read the datafile using a special technique (so the package is moveable)

Now try to run the code: `python ./rr/paper.py`

What just happened?

What just happened?

```
$ python rr/paper.py
Traceback (most recent call last):
  File "rr/paper.py", line 15, in <module>
    from . import database
ValueError: Attempted relative import in non-package
```

Need to install

You can't just execute the code from a package, *directly from it*. We need need to install it using `buildout`[10].

Buildout does the following, auto*magically*:

1. **Locally** downloads and installs all packages required **both** directly and indirectly by the package you're developing
2. Prepares a copy of the Python interpreter and all required scripts so that they can **prioritarily** access the **locally** installed packages.

Why?

Someone has to control the required dependencies! This "check" is the installation.

[10] `https://pypi.python.org/pypi/zc.buildout`

Disclaimer

There are other ways to install Python packages. Using `buildout` is simpler. For more information and alternatives, please have a look at:

- setuptools: `https://pypi.python.org/pypi/setuptools`
- pip: `https://pypi.python.org/pypi/pip`
- virtualenv: `https://virtualenv.pypa.io/en/latest/`

Usefulness of `buildout`

It is not immediate, but here are some advantages from this behaviour:

1. Every buildout is independent
2. You can override system package versions locally

Learn this

One package == one task

Where to use `buildout`

We use `buildout` everywhere a work environment needs to be setup. Some examples:

- Each package we ship has a **development environment** that can be re-created using `buildout`
- Each paper we submit, includes a **production environment** that allows users to re-build the necessary environment locally and run the experiments

How can I ship an environment?

Very easy, you must include 2 files within (the root of) your package:

1. A bootstrap file that allows the user to download the latest version of `buildout` (stock)
2. A text file, normally named `buildout.cfg` that describes how to build the environment. (you write it)

In practice

```
# don't need to do it now:
$ wget https://bootstrap.pypa.io/bootstrap-buildout.py
$ gedit buildout.cfg #open it!
```

A buildout recipe

```
1  [buildout]
2  parts = scripts
3  develop = .
4  eggs = rrpack
5  newest = false
6
7  [scripts]
8  recipe = bob.buildout:scripts
```

It is a (Windows) ini-style file, divided into sections:

1. The first section is **obligatory**, and must be called `buildout`. It indicates what other sections are relevant and global options
2. This buildout only has 1 other section called `scripts`
3. It must consider the local `setup.py` file first
4. It must **install** scripts for the package `rrpack` (which happens to be the current one)
5. If it finds versions of dependencies which are not the latest possible value, it is OK
6. The second section
7. `buildout` is **extensible** with recipes. The `bob.buildout:scripts`[11] installs all scripts required for normal development (nosetests, a python interpreter, sphinx documentation generator, etc).

[11] `https://pypi.python.org/pypi/bob.buildout`

How to use it?

This can't be easier:

```
# downloads it and creates ./bin/buildout
$ python bootstrap-buildout.py
# the next command will "follow" your recipe
$ ./bin/buildout
```

And so?

So, you can now run!

```
$ ./bin/paper.py
...
$ ./bin/nosetests #even simpler!
...
```

Notice

The beauty and magic that just happened:

- Check system, download required packages
- Installs scripts to execute what you need, pointing to locally installed packages if required

Where are all packages coming from?

They come from a central Python Package Index, or simply said `PyPI`.

- PyPI is a standard Python egg[12] distribution mechanism
- The web address is: `https://pypi.python.org/pypi`
- We **strongly** recommend you distribute your packages using it instead of GitHub:
  - Supports documentation hosting
  - Supports automatic dependence tracking
  - Can hold large packages (50MB OK)
  - Separates development from ready-to-use states

[12] Synonim for Python package

Versioning

Version numbers are easy in Python[13]:

```
0.4       0.4.0  (these two are equivalent)
0.4.1
0.5a1
0.5b3
0.5
0.9.6
1.0
1.0.4a3
1.0.4b1
1.0.4
```

In short

- 2 or 3 numbers separated by dots
- (optional) `[abc]N`, indicating *alpha*, *beta* or *candidate* versions

[13] https://www.python.org/dev/peps/pep-0386/

Workflow

Recommended workflow:

- Keep your package at GitHub or similar
- Develop on the `master` branch
- Everytime you release, tag the package (push to GitHub)
- Upload the package to PyPI

Register at PyPI and TestPyPI

Let's all register at these two services for the next exercises. Access (tip: use the same username):

PyPI: `https://pypi.python.org/pypi?%3Aaction=register_form`

TestPyPI:
`https://testpypi.python.org/pypi?%3Aaction=register_form`

After completing the registration

Modify the file `/.pypirc`, set your PyPI and TestPyPI credentials:

```
$ gedit ~/.pypirc
```

Register package at PyPI (first time)

```
$ python setup.py register -r pypitest #use test index
```

What will happen

- Your `README.rst` will be uploaded
- Your package will be **described** at PyPI from that moment: `https://testpypi.python.org/pypi/rrpack`

Notice

- For tests, use the Test PyPI server
- If everybody tries to register the same package, the registration will fail (modify name for your test)

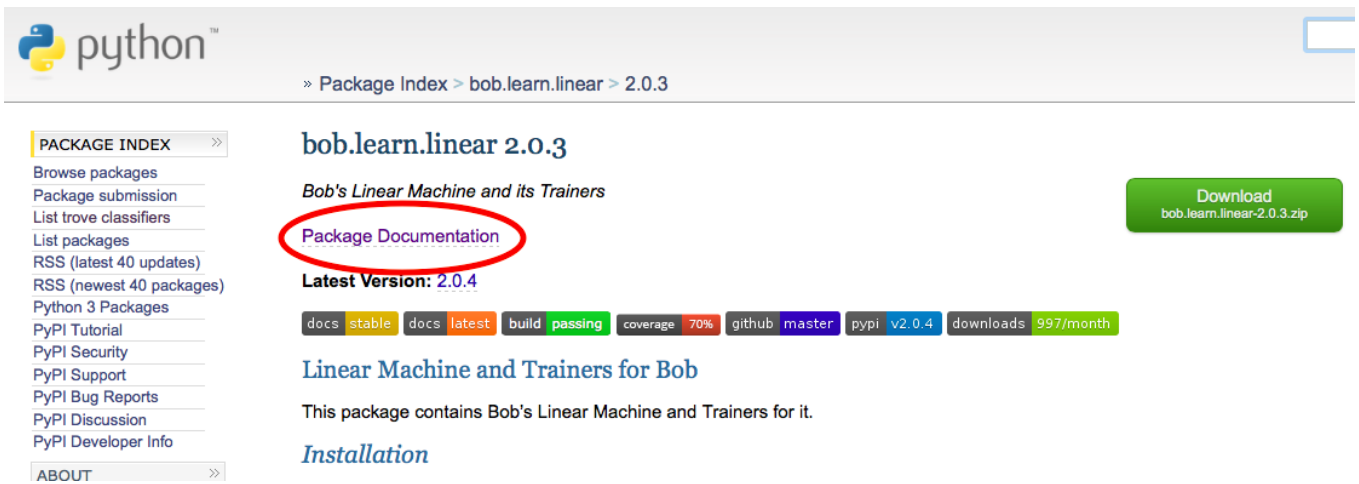
Upload package to PyPI (at every release)

```
# create a zip of the package
$ python setup.py sdist --formats=zip
# upload to server
$ python setup.py upload -r pypitest
```

Your package will be **available** to PyPI from that moment:
`https://testpypi.python.org/pypi/rrpack`

Package Documentation

You can have more complete documentation for your package. For example, look at `https://pypi.python.org/pypi/bob.learn.linear`:



Sphinx

Sphinx is a documentation generator for Python packages. What can it do:

- Automatically scan/format docstrings from your functions to generate beautiful documentation
- Read extra documents to create sections and chapters as you see fit
- Syntax Highlight code
- Print LaTeXformulas
- Include test-able python code
- Uses the reST format you're now familiar with
- Much more!

Checkout: `http://sphinx-doc.org`

Using Sphinx

To start a documentation project on your package, do the following:

```
$ ./bin/sphinx-quickstart doc #don't have to do it
# follow the instructions
$ ./bin/sphinx-build doc sphinx
$ xo ./sphinx/index.html
```

Tip: Serving your docs

Once your documentation is tested and updated, PyPI offers a hosting place for it. Pack the sphinx output directory (the file `index.html` must be on the root of the package) and upload it using the **PyPI admin page** for your package.

A short word on *doctests*

Documentation tests (a.k.a. *doctests*) are bits of Python code in your documentation, that are specially marked for testing.
Example:

```
.. doctest::

   >>> from rr.analysis import CER
   >>> import numpy
   >>> a = numpy.array([0,1])
   >>> b = numpy.array([1,1])
   >>> CER(a, b)
   0.5
```

Running documentation tests

Documentation tests (a.k.a. *doctests*) are not run automatically. You need to select a special Sphinx builder for it:

```
$ ./bin/sphinx-build -b doctest doc sphinx
$ xo ./sphinx/index.html
...

Doctest summary
===============
    5 tests
    0 failures in tests
    0 failures in setup code
    0 failures in cleanup code
build succeeded.
```

More Sphinx

You can get more information about Sphinx, extensions and techniques at:

`http://sphinx-doc.org`

Do we have everything?

Let's check what we have so far:

- Your code is organized
- Your code is documented
- Your peers can now download the source code + dependencies (from PyPI)
- You keep track of changes and tag stable releases
- The code is hosted on a free website that also gets you a Wiki and an Issue tracker (GitHub)
- Your code is tested

Do we have everything?

Let's check what we have so far:

- Your code is organized
- Your code is documented
- Your peers can now download the source code + dependencies (from PyPI)
- You keep track of changes and tag stable releases
- The code is hosted on a free website that also gets you a Wiki and an Issue tracker (GitHub)
- **Your code is tested?** Really? Did you remember it before releasing it last time?

Do we have everything?

Let's check what we have so far:

- Your code is organized
- Your code is documented
- Your peers can now download the source code + dependencies (from PyPI)
- You keep track of changes and tag stable releases
- The code is hosted on a free website that also gets you a Wiki and an Issue tracker (GitHub)
- **Your code is tested?** Really? Did you remember it before releasing it last time?
- Have you tested for Python 3.x as well?

Enter Continuous Integration (CI)

The concept is: test at every (GitHub) push, so problems are caught as early as possible.

- A green build indicates the tests are passing
- If problems occur, you get an e-mail and a red build flag

Why?

- It can test on many Python **versions**
- It will test **whenever** people push code to GitHub
- It will make sure your tests are **always OK**

Implementing CI

There are many ways to implement CI. The easiest for GitHub projets, is to use Travis (`https://travis-ci.org`).

- Deeply integrated with GitHub (same login)
- At every push, runs a set of commands **you define** on a "new" virtual machine running Ubuntu 12.04
- Provides a "badge" icon, you can stick to your `README.rst` file, so you can keep track of quality
- Can run a build matrix including **various versions** of Python

Which commands

The "commands" to run depend on your project. For our toy problem, the check is pretty simple:

```
$ python bootstrap-buildout.py
$ ./bin/buildout
$ ./bin/nosetests -sv
$ ./bin/sphinx-build -b doctest doc sphinx
$ ./bin/sphinx-build -b html doc sphinx
```

Implementation at Travis-CI

Put these commands on a YAML formatted file called `.travis.yml` on the root of your package and enable it through `https://travis-ci.org`:

```
$ gedit .travis.yml #let's check it together
```

After pushing the package, the builds at Travis will start. Visit:

`https://travis-ci.org/anjos/rrpack`

To view the build status. **Browse around a bit.**

More Travis

You can get more information about Travis and techniques from their website:

- Python getting started: `http://docs.travis-ci.com/user/languages/python/`
- PyPI deployment (from tags): `http://docs.travis-ci.com/user/deployment/pypi/`
- Coverage integration: `https://coveralls.io`
- More badge services: `http://shields.io`
- Building on multiple OSes: `http://docs.travis-ci.com/user/multi-os/` (still beta)

What is Bob?

Put simply, Bob is **all** that you have seen, wrapping a number of Signal Processing, Pattern Recognition & Machine Learning tools into an easy to use software environment.

Packages in the Wild

Bob is a disjoint set of packages[14] built using a common, uniform, `bob.extension` support.

Packages are all available in PyPI for easy download and installation: `https://pypi.python.org/pypi?:action=browse&show=all&c=590`

Tip

Your virtual images **already** contain most Bob packages pre-installed.

[14] `https://github.com/idiap/bob/wiki/Packages`

Installation

We have installation procedures for a number of OSes[15]:

- Linux[16]: Ubuntu, Debian
- Mac OSX

No MS Windows (yet)

Working on a Windows port since sometime - contributions are welcome.

[15] https://github.com/idiap/bob/wiki/Installation
[16] Should work on most Linux-based distros

What is there?

Each package contains a specific set of utilities. You use them as you like:

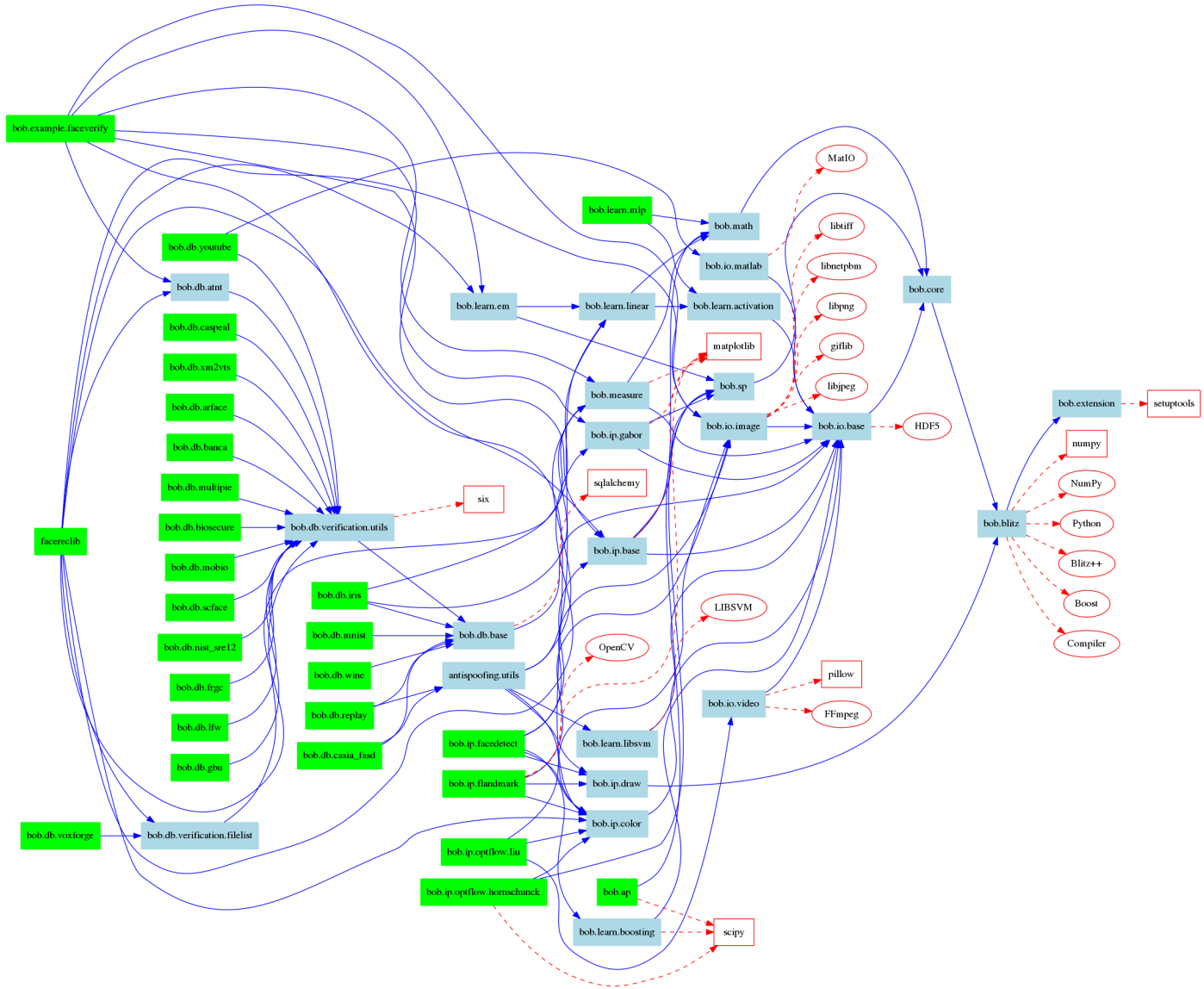
- Input/Output: Extensible support HDF5, Images, Videos, Audio, Matlab
- Audio/Signal processing: MFCC, FFT, DCT
- Image processing: Local Binary Patterns (LBPs), Gabor Jets, SIFT, HOG, GLCM, TanTriggs, etc
- Machine Learning: Linear models, Neural Nets, SVMs, GMMs, Boosting, Bayesian intra/extra (personal) classifier, Inter-Session Variability modeling (ISV), Joint Factor Analysis (JFA), Probabilistic Linear Discriminant Analysis (PLDA), I-Vectors, etc.
- Database interfaces: Toy databases, M-NIST character recognition, Various Biometric databases (face, speaker, anti-spoofing, vein)
- Metrics: F1-Score, DET, ROC, ROC-Convex Hull, EPC, CMC, EER and WER minimization, etc.

Bob: Figures

Statistics:

- $\sim$**5'000 updates** on repository
- $>$**22'000** new front page **visits** since April 2012
- $>$**50 satellite packages** (tutorials and published papers)
- $>$30 database APIs for different purposes (more coming)

Interdependencies



Quality Control

An automated framework controls the quality of code added to key Bob packages:

- Checks at every code modification
- More than >**1200 test units**
- Tests code and installation infrastructure every night, on > 10 **different operating systems**

How can I make use of that?

You already are!

We're using the `bob.buildout` recipe extension on your buildout environment.

But you can go further, of course!

The final exercise

You sent an e-mail to your anonymous colleague once more, telling about Bob. The author rewrote the package, created a new GitHub repo and then uploaded it to PyPI, along side its documentation.

You now start at: `https://pypi.python.org/pypi/rrbob`

You now, you *know* what to do!

Keeping in touch

We created a Google Groups forum for discussion. Join to:

- Keep up-to-date with the framework development
- Discuss issues and other problems you may be facing

Details:

- View posts: https://groups.google.com/d/forum/bob-devel
- Subscribe: bob-devel+subscribe@googlegroups.com
- Post: bob-devel@googlegroups.com

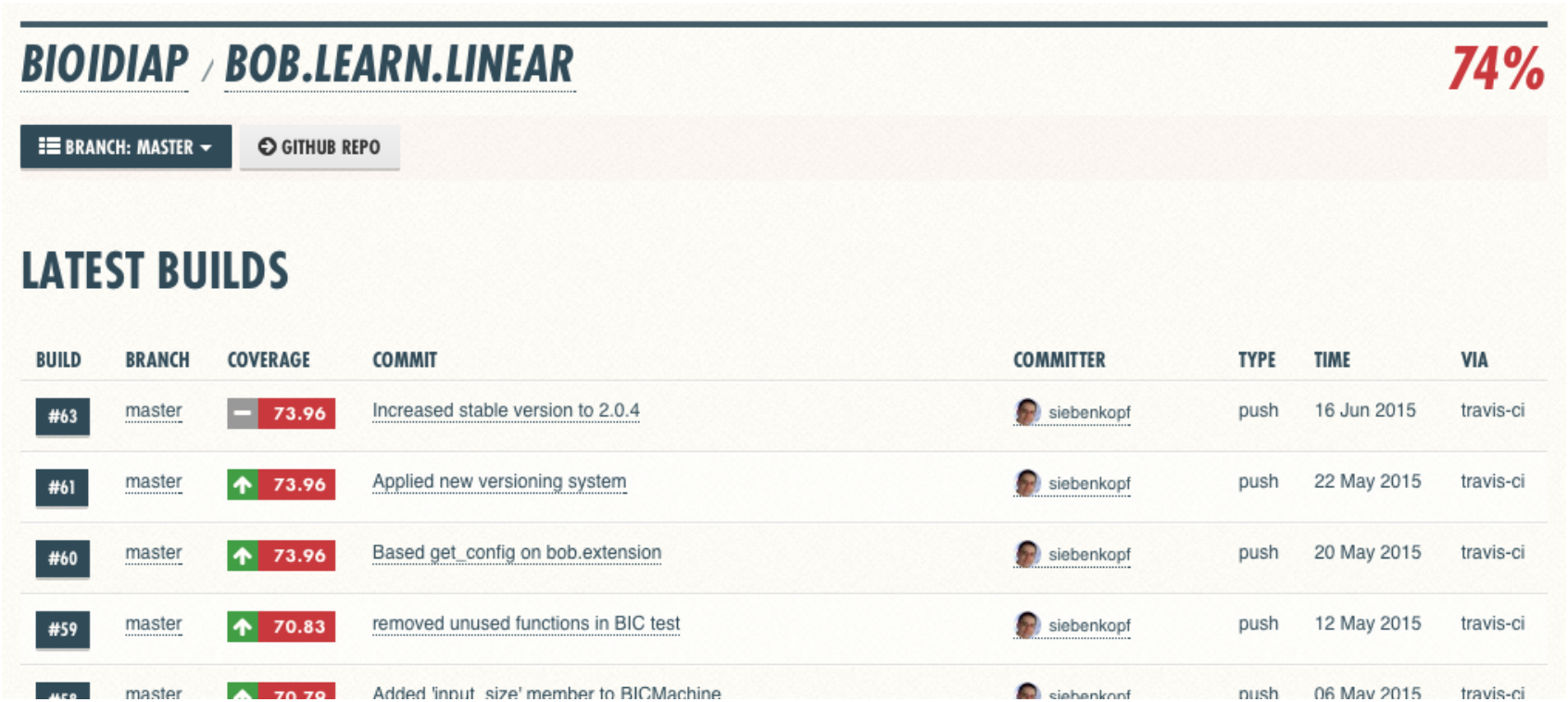
Test Coverage

If you'd like to check how much of your **tested code** is actually being tested.

```
$ ./bin/nosetests --with-coverage --cover-package=rr -sv
rr.test.test_50_50 ... ok
rr.test.test_20_80 ... ok

Name           Stmts   Miss  Cover   Missing
--------------------------------------------
rr                 0      0   100%
rr.analysis        3      0   100%
--------------------------------------------
TOTAL              3      0   100%
--------------------------------------------
Ran 2 tests in 0.049s

OK
```

Coveralls

A service to host coverage reports.



As you can now expect, it integrates nicely with GitHub and Travis.

Coveralls + Python

Here you'll find some material available on the wild:

- python-coveralls:
  `https://pypi.python.org/pypi/python-coveralls`
- coveralls-python:
  `https://pypi.python.org/pypi/coveralls`

C/C++ Extensions

It is possible to ship your packages with C/C++ extensions. There are many ways to do it:

Easy	**Cython**: `http://cython.org`
Harder	**Boost.Python**: `http://www.boost.org/doc/libs/1_55-_0/libs/python/doc/`
Flexible	**Python - C API**: `https://docs.python.org/2/extending/extending.html`
Bob's	http://pythonhosted.org/bob.extension/extension.html

Re-cap

Up to this point, you learned:

- What is RR and its importance for CS/Engineering
- You can do RR with Python
- How to design your Machine Learning problem
- How to document using reST/Python
- How to perform unit testing
- How to package your stuff using Python
- How to integrate Continous Integration
- What is and how to use Bob

Starting...

Here is how you should start:

1. Use git/GitHub or similar for keeping track of your projects
2. Try to depend on stable libraries in the wild
3. Document, organize so it is easy to share your libraries
4. Add unit testing
5. Package it properly, setup distribution mechanism
6. Add continuous integration

Wrapping-up

What are your impressions up to this point?

Do you think it is hard to do RR?

Next course:

- A new paradigm in RR: The BEAT Platform

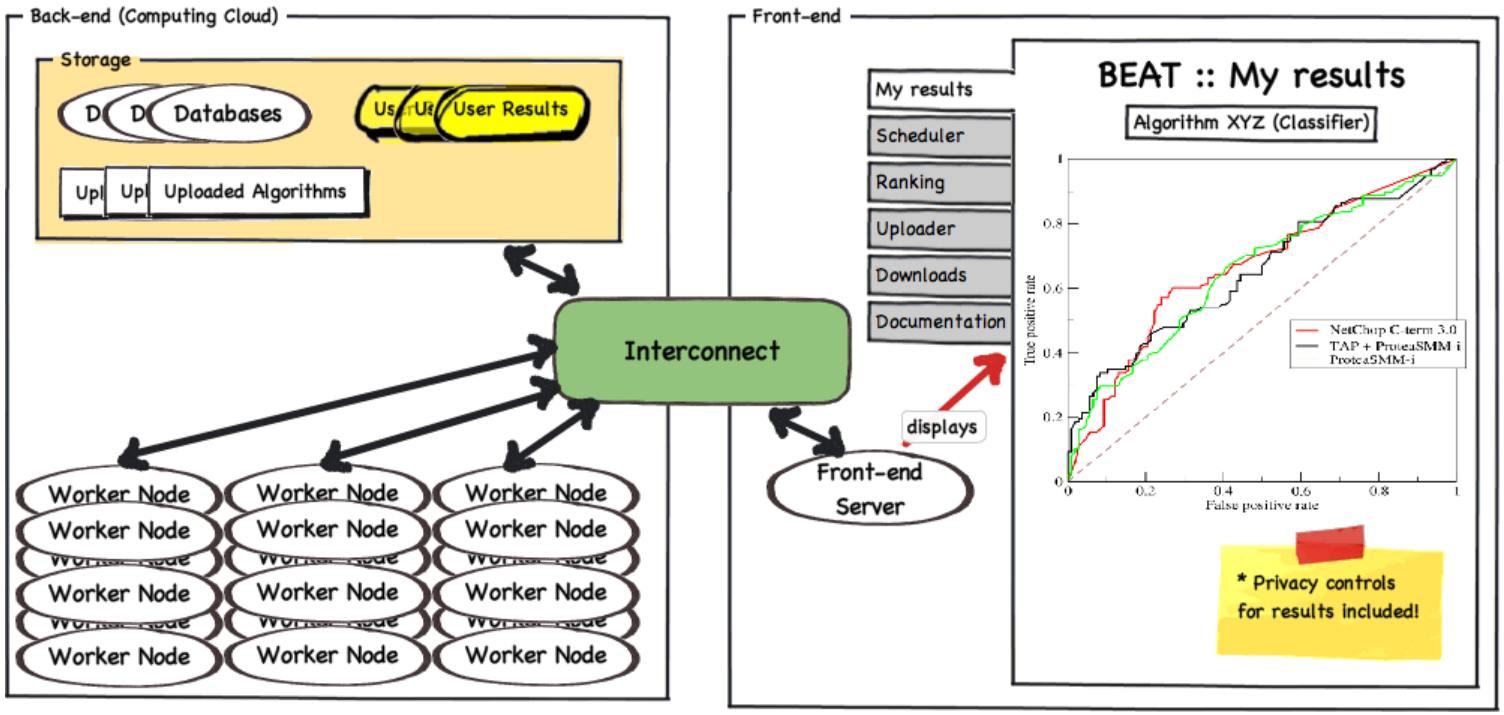
What can be improved?

- Downloading and storing **data** may be a privacy concern in many countries:
  - Need to work-out space for the growing number of samples
  - Not all databases are distributable (e.g. *forensic data*)
- **Software** management and installation can be hard
  - Software gets outdated: constant quality and integration
  - Plan for errors: re-distribution mechanism
- **Computing** can be limited

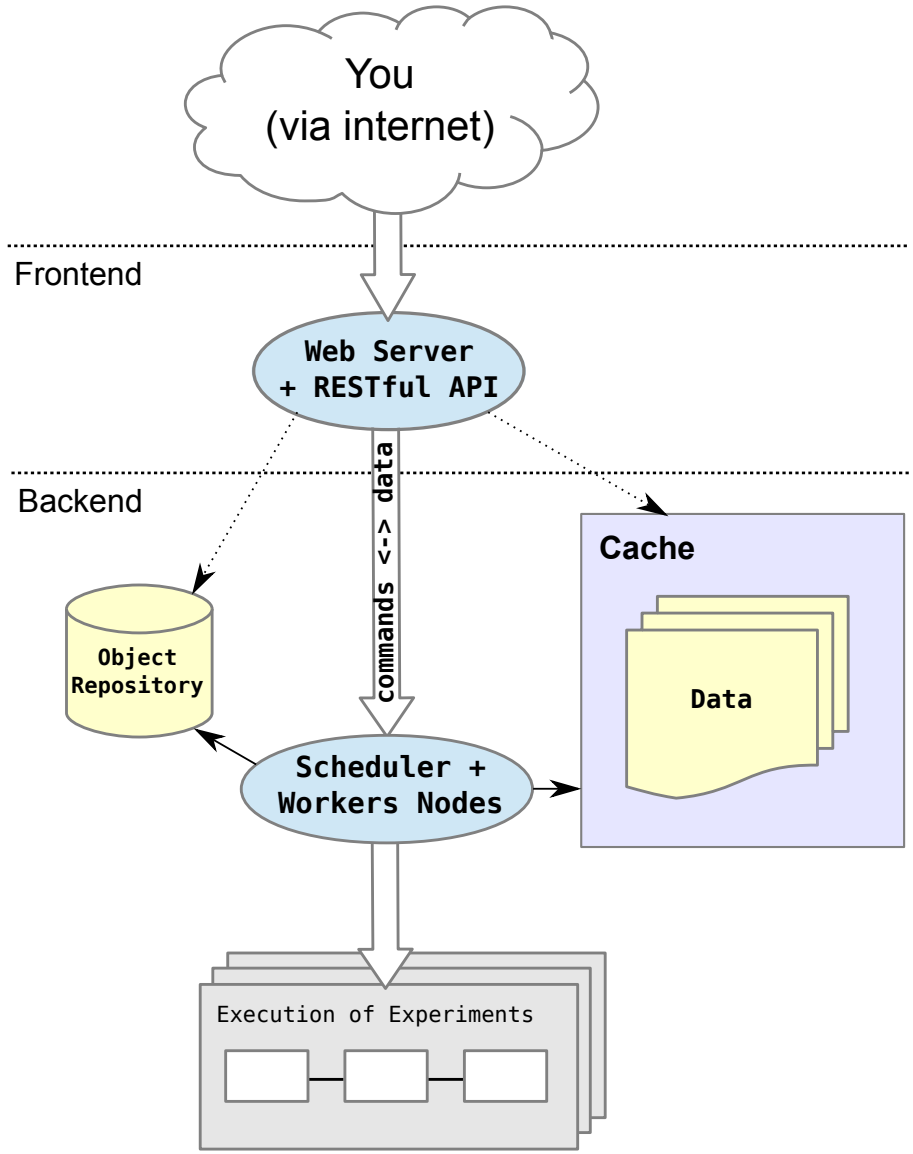
BEAT: A web platform for RR

- **Accessible**: no need to install extra software
- **Intuitive**: graphically connect blocks to run experiments
- **Social**: engagement gets you more processing power
- **Productive**: search the state-of-the-art by any filtering criteria
- **Data Privacy**:
  - No need to handle large-scale databases
  - Can run on un-distributable data (e.g. proprietary databases)
- **Assurance**:
  - fair (reproducible) evaluations of algorithms
  - online attestations for all produced results
- **Free**: build on open-source software and standards

EU-FP7 Project Proposal

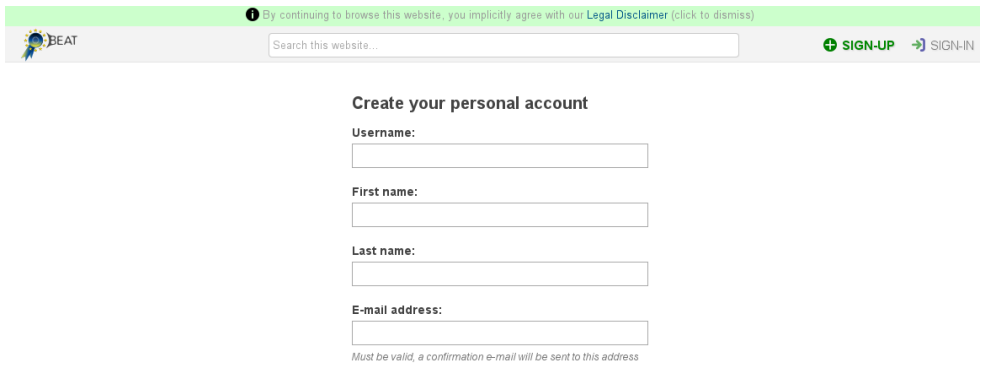


Final Design



Let's all register

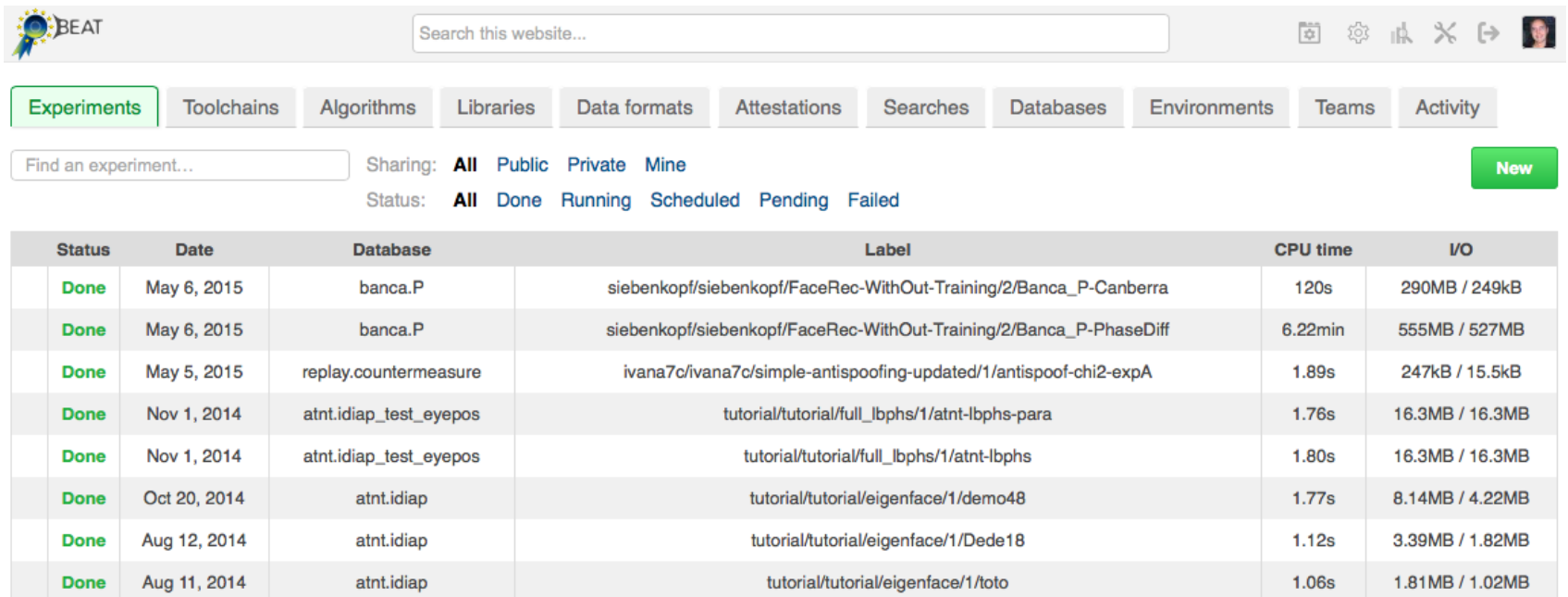
This should allow you to contribute to the platform.



- Go to https://beat-eu.org/platform
- To register, click on the green button "SIGN-UP"
- Fill-in all fields and check the ToS box
- Go to your e-mail box and confirm the registration
- Log-in with your password and go to your user page
- Browse publicly available objects

Experiment list

At the first contact as a logged-in user with the platform, you immediately have access to all experiments produced by you or shared with you.



Status	Date	Database	Label	CPU time	I/O
Done	May 6, 2015	banca.P	siebenkopf/siebenkopf/FaceRec-WithOut-Training/2/Banca_P-Canberra	120s	290MB / 249kB
Done	May 6, 2015	banca.P	siebenkopf/siebenkopf/FaceRec-WithOut-Training/2/Banca_P-PhaseDiff	6.22min	555MB / 527MB
Done	May 5, 2015	replay.countermeasure	ivana7c/ivana7c/simple-antispoofing-updated/1/antispoof-chi2-expA	1.89s	247kB / 15.5kB
Done	Nov 1, 2014	atnt.idiap_test_eyepos	tutorial/tutorial/full_lbphs/1/atnt-lbphs-para	1.76s	16.3MB / 16.3MB
Done	Nov 1, 2014	atnt.idiap_test_eyepos	tutorial/tutorial/full_lbphs/1/atnt-lbphs	1.80s	16.3MB / 16.3MB
Done	Oct 20, 2014	atnt.idiap	tutorial/tutorial/eigenface/1/demo48	1.77s	8.14MB / 4.22MB
Done	Aug 12, 2014	atnt.idiap	tutorial/tutorial/eigenface/1/Dede18	1.12s	3.39MB / 1.82MB
Done	Aug 11, 2014	atnt.idiap	tutorial/tutorial/eigenface/1/toto	1.06s	1.81MB / 1.02MB

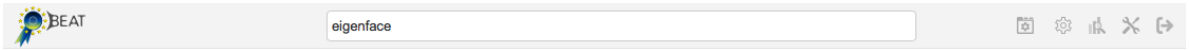
Click any of them!

Experiment display

In the experiment display, you can access information about the experiment that was run:

- Results produced by its analysis, including plots and single numbers
- Parameters used by this experiment (single values, processing environments, execution time, work flow, etc)
- Each experiment produces an enormous amount of information that the platform tracks

Searching



Viewing a single experiment is not very useful. Let's try to get an overview of results, for example, on "eigenface".

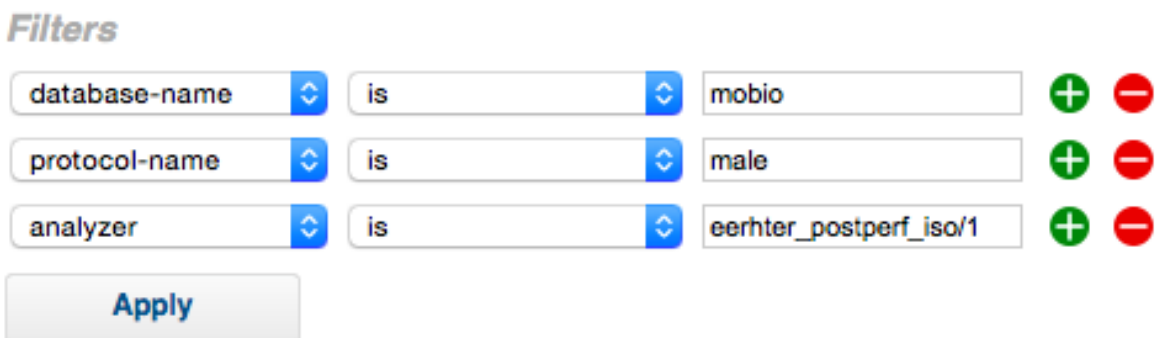
- At the top search bar, type "eigenface"
- The platform will search **any** component with that string on the name
- If required, refine your search using the filters you have available - *you can refine your search quite substantially using these selectors*
- Notice you can only compare results produced with the same analysis algorithm
- After filtering, the table shows a few columns (these are the default for the analysis for these experiments), to display more, click on the "gear" icon. You can display the aggregated plots, for example.

Another test

Let's search for the best results on MOBIO (protocol "male") *visible to you* at the platform.

On the filter bar:

- choose `database-name` and then `is` `mobio/1`
- choose `protocol-name` and then `is` `male`
- choose `analyzer` and then `is` `eerhter_postperf_iso`.



Start contributing

We now want to run an experiment on our own, using the eigenface/AT&T setup as basis:

- Go to the user micro-site (2 icons left to your user icon)
- Search/click on the experiment `tutorial/tutorial/eigenface/1/eigenfaces_5comp`
- On the top-right, click on "Fork"
- Tune the experiment parameters on the right
- Choose a label
- Hit "Go!"
- Wait for the experiment to complete
- Hit "Similar experiments"

Do you do better or worse? Did you check your ROC? (gear icon)

Going deeper

The BEAT Platform provides an easy way to run pattern recognition experiments involving software components.

We hope that after some iterations, public material will allow for a large number of baselines and state-of-the-art results to be **easily accessible** and **reproducible**.

You can contribute in many levels:

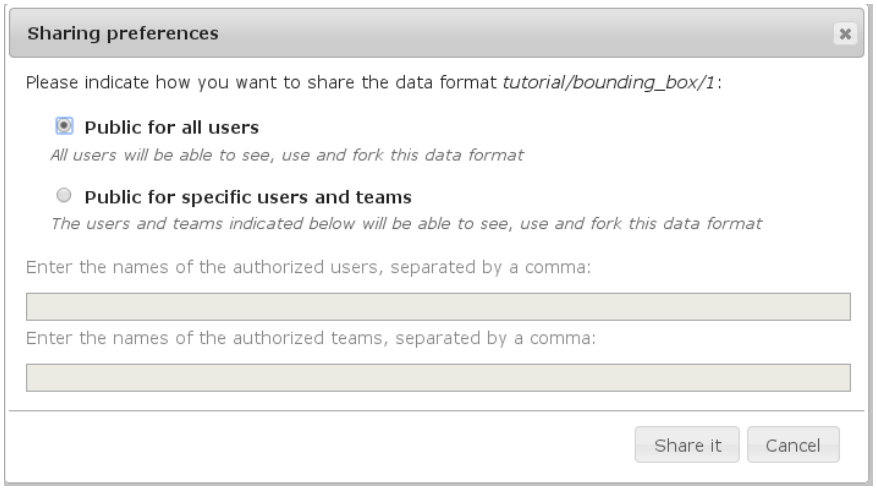
- running experiments produced by others.
- **contributing** back to create your own unique experiments, shared to any number of parties you want.

Note: Confidentiality

BEAT is designed as an *opt-in* platform

- All your actions and results are kept private until you choose to change visibility, by clicking on the "Private" icon
- Once you (or an authorized peer) use an item, it cannot be modified (or erased) anymore
- So that an item is modifiable, it must not be **in use**
- If an item is not modifiable, it can be made modifiable again by removing all items pointing to it

Teams



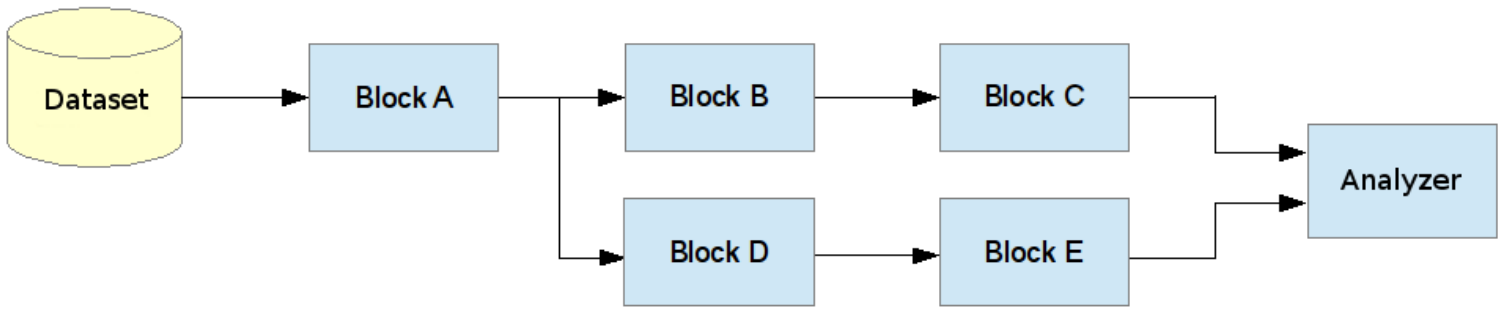
It is possible for users to organize themselves using “Teams”.

Interact: Open the “Teams” tab, try to create a team and add users.

You can use “Teams” to share your contributions with a specific set of users.

Semantics: Toolchains (or Workflows)

A key concept in experimentation at the platform is the idea of *Toolchains*.



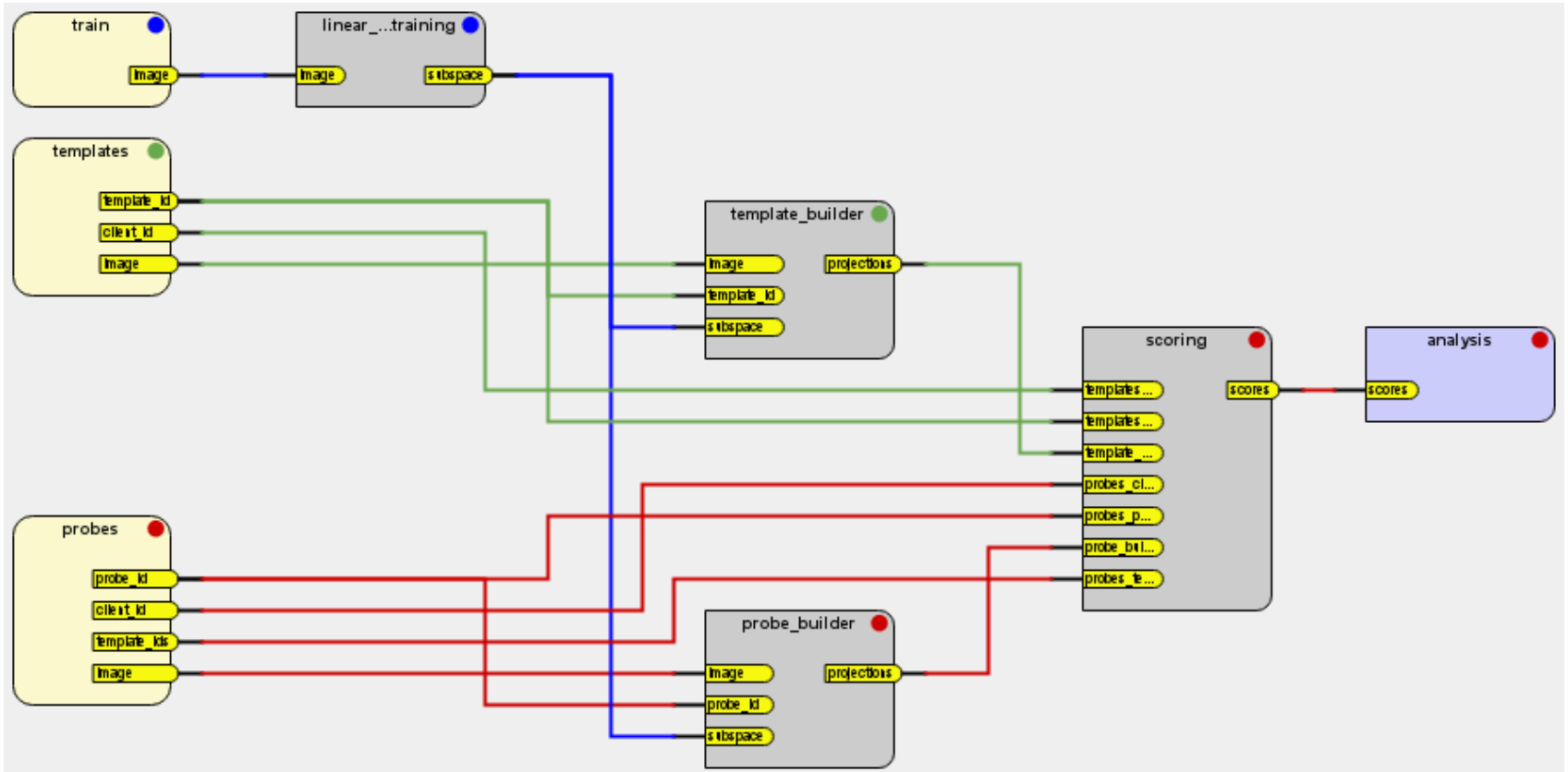
Toolchains are decomposable into blocks.

Toolchain Example: trivial Eigen-faces

Simple: no evaluation (test) set, threshold *a posteriori*

Toolchain Example: trivial Eigen-faces

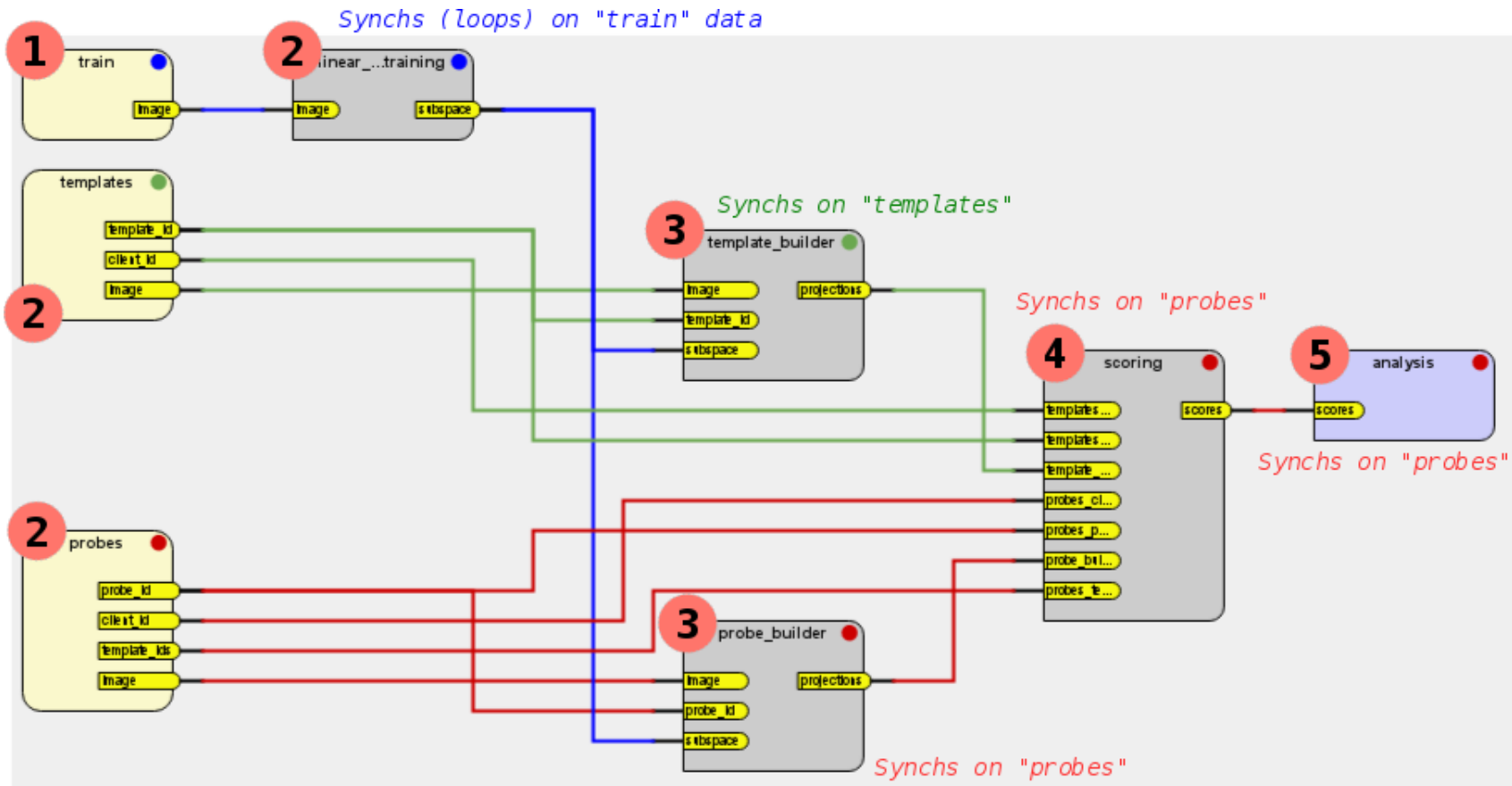
Translation as a BEAT toolchain



- All relations are explicit: I/O, work flow
- We **do not** define here which algorithms, datasets, analyzers, formats or other results to run and produce!

Toolchain Example: trivial Eigen-faces

Translation as a BEAT toolchain



- All relations are explicit: I/O, work flow
- We **do not** define here which algorithms, datasets, analyzers, formats or other results to run and produce!

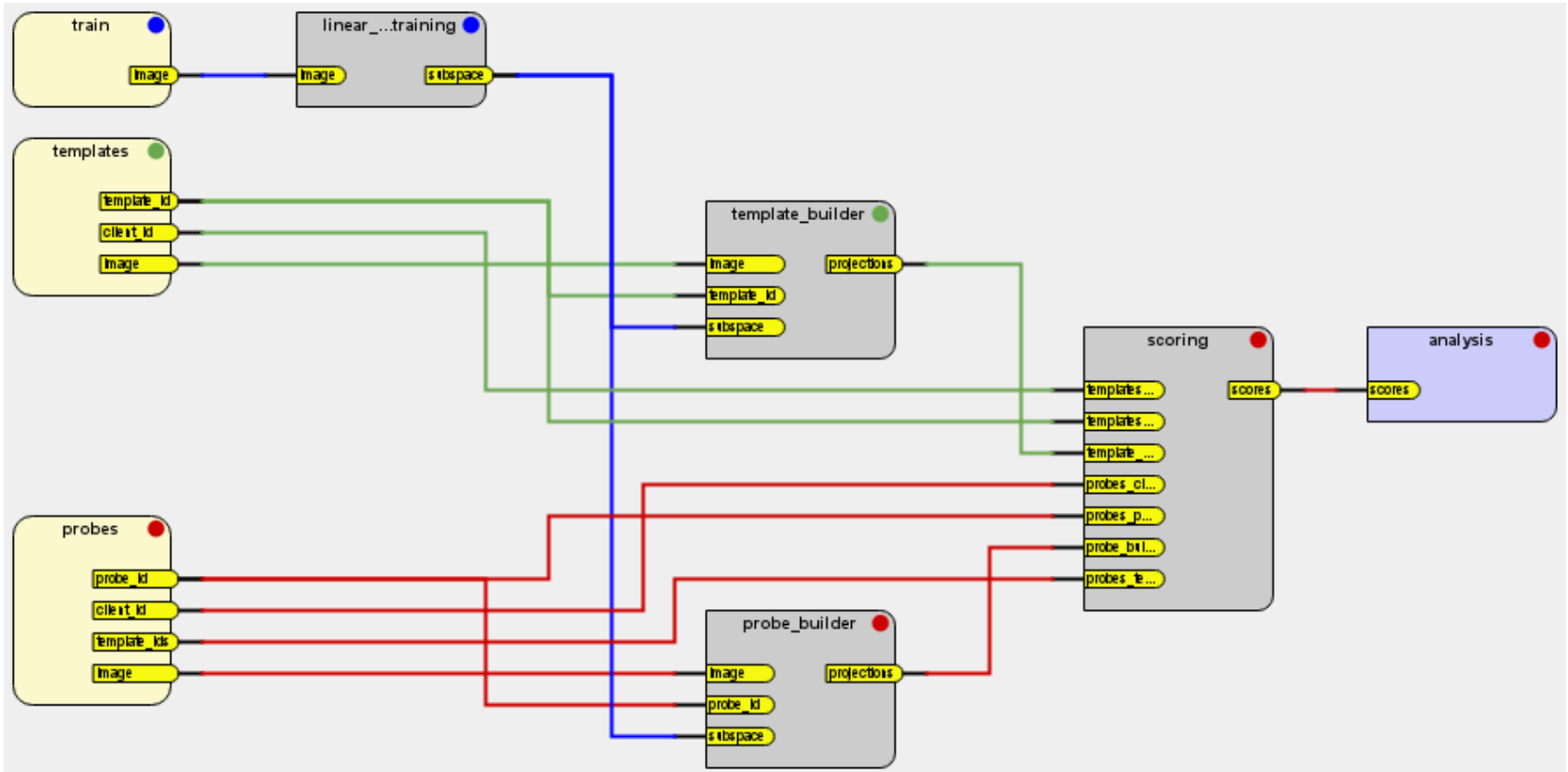
Browsing

Interact: In your browser, try to click on the "Toolchains" tab:

- Use the small search bar under the tab to refine the search and look for "eigenface" (thanks to the "tutorial" user, it is avaiable publicly);
- You can zoom in/out with a scroll button on your mouse or with an equivalent gesture on your laptop.
- You can move the canvas by middle-clicking and dragging (option-drag on a Mac, left-alt-drag on a PC)
- There are a few experiments that use this toolchain (click on any of them)

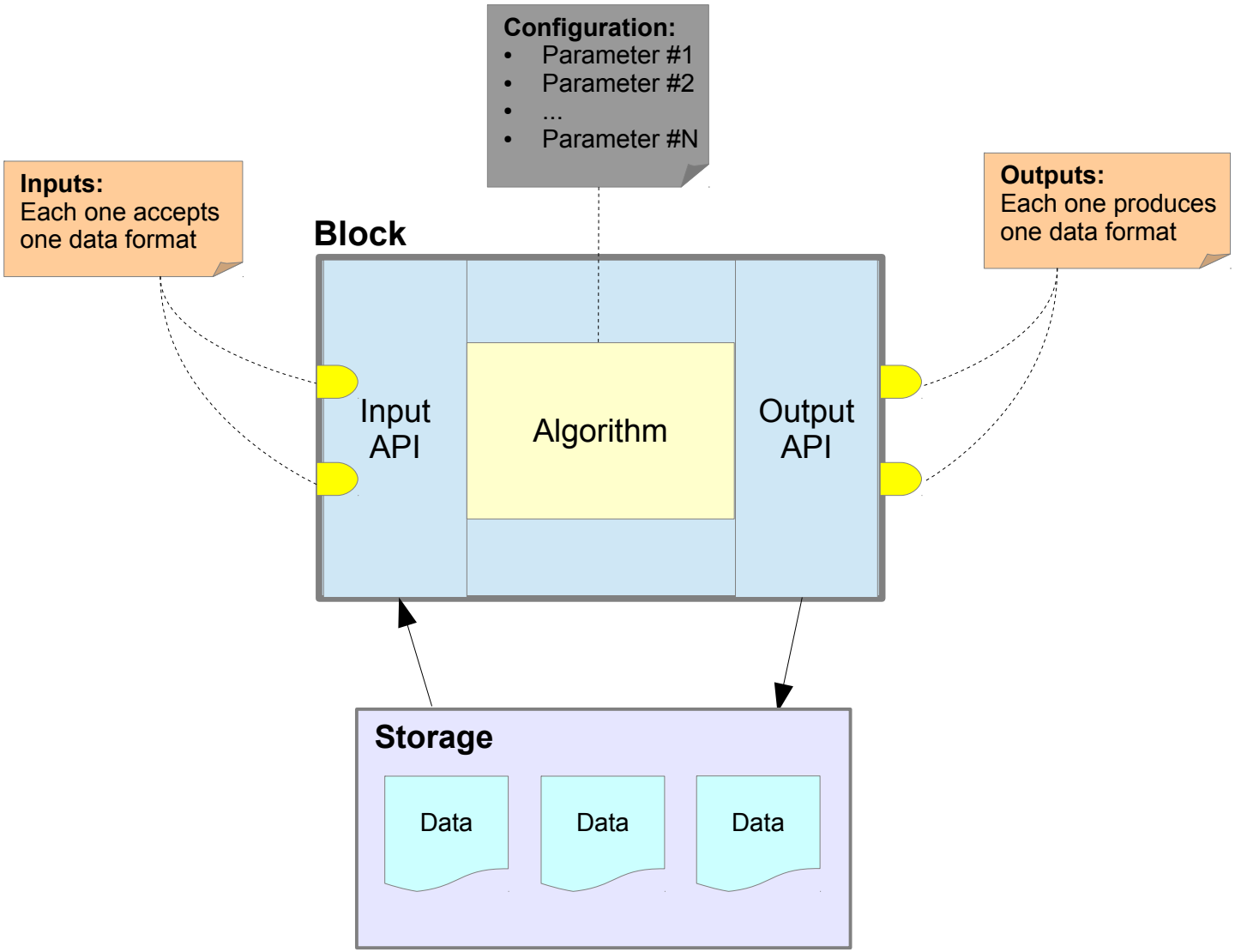
Blocks

Most elements in a toolchain are *ordinary* blocks



Blocks

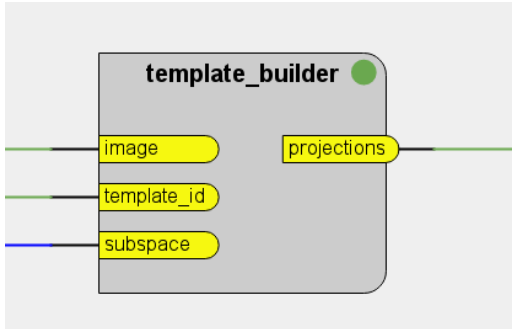
A look inside a block:



Blocks: Features

- Blocks can be equipped with **arbitrary** code
  - Potential to implement back-ends to support compiled code, any scripting language
  - We picked Python as our first implemented back-end
- Blocks can be executed on **arbitrary** architectures
- Blocks have inputs and outputs
- Data transmitted from block to block is formally defined (*Data Formats*)
- Dataset blocks only have outputs, they define the possible data flows (loops)
- Analyzer blocks don't output to any other block, they provide the results of your workflow

Blocks



A block is composed of:

- an **unique name** (in the Toolchain)
- one or more **inputs**
- one or more **outputs**
- a synchronization channel (where the *for* loop is)

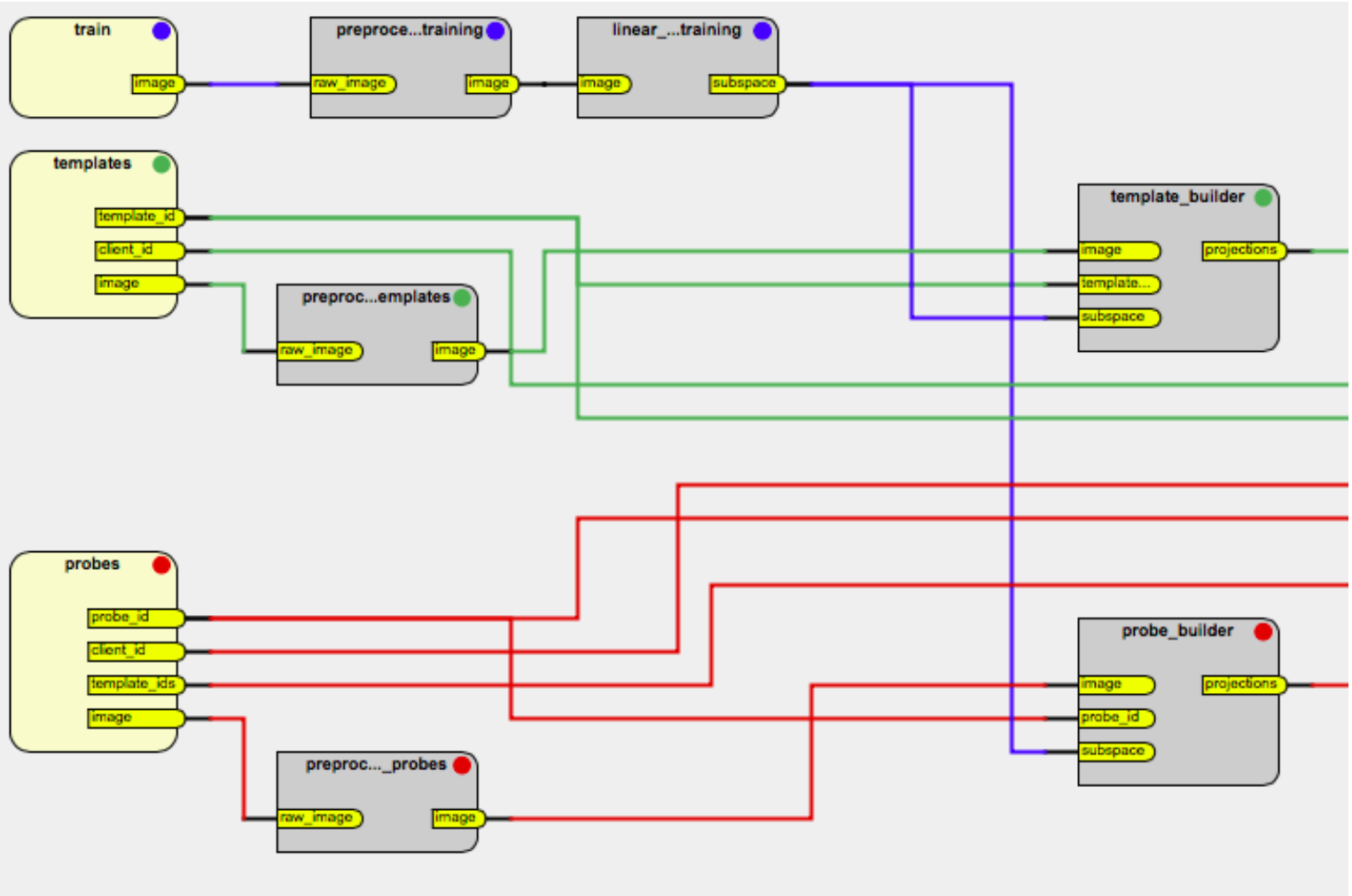
To perform an **experiment**, a *compatible* **algorithm** must be associated to each block.

Contributing

Contribute: Like you did it for the experiment, let's fork this toolchain and add a block for image pre-processing.

- Use the "Fork" button available on the right of the page.
- You'll be re-directed to the editor page, eventually.
- Using your mouse, draw additional blocks for the "image" inputs, just after they have been emitted by the database
- Save your work regularly

Hint


train
image
preproce...training
raw_image
image
linear_...training
image
subspace
templates
template_id
client_id
image
preproc...emplates
raw_image
image
template_builder
image
template...
subspace
projections
probes
probe_id
client_id
template_ids
image
preproc..._probes
raw_image
image
probe_builder
image
probe_id
subspace
projections

Dataformats

A **dataformat** describes the structure of an atomic data unit: which fields it contains, and what their type is.

Interact: On your tabs, select "Data formats" and look around.

Example: A single floating-point value (*system/float/1*)

```
{
  "value": "float64"
}
```

Dataformats: Example

Example: Two integer fields (*system/coordinates/1*)

```
{
  "x": "int32",
  "y": "int32"
}
```

In our Python backend, user code can assign values to an object of this type like this:

```
obj.x = numpy.uint32(42)
obj.y = numpy.uint32(53)
```

Dataformats: Primitive types

The following types are available to declare new dataformats:

- Integers: *int8*, *int16*, *int32*, *int64*
- Unsigned integers: *uint8*, *uint16*, *uint32*, *uint64*
- Floating-point numbers: *float32*, *float64*
- Complex numbers: *complex64*, *complex128*
- *bool*
- *string*

Note

With a Python backend, those primitive types are implemented using their *NumPy* counterparts.

Dataformats: Objects

A dataformat can declare whole objects as the type of its fields.

For instance, the coordinates of a rectangular region in an image could be represented by the following dataformat:

```
{
  "coords": {
    "x": "int32",
    "y": "int32"
  },
  "size": {
    "width": "int32",
    "height": "int32"
  }
}
```

The fields can then be used in Python like this:

```
data.coords.x = 10
data.coords.y = 20
data.size.width = 100
data.size.height = 100
```

Dataformats - Composition

It is possible to use an existing dataformat as the type of a field.

Example: *system/eye_positions/1*

```
{
  "right": "system/coordinates/1",
  "left": "system/coordinates/1"
}
```

The fields can the be used in Python as in the object example:

```
data.left.x = 10
data.left.y = 50
data.right.x = data.left.x + 30
data.right.y = data.left.y
```

Dataformats - Array types (1)

A field can also declare a multi-dimensional array of any other type.

For instance, consider the following example:

```
{
  "field1": [10, "int32"],
  "field2": [10, 5, "bool"]
}
```

Here we declare that *field1* is a one-dimensional array of 10 integers (*int32*), and *field2* is a two-dimensional array of 10x5 booleans.

Note

In Python, those arrays are implemented using `numpy.ndarray`.

Dataformats - Array types (2)

It is also possible to declare an array without specifying the number of elements, by using a size of 0:

```
{
  "field1": [0, "int32"],
  "field2": [0, 0, "bool"],
  "field3": [10, 0, "float32"]
}
```

Here *field1* is a one-dimensional array of integers (*int32*), *field2* is a two-dimensional array of booleans, and *field3* is a two-dimensional array of floating-point numbers (*float32*) where the first dimension is fixed to 10.

Dataformats - Array types (3)

Note that the following declaration isn't valid (you can't fix a dimension if the preceding one isn't fixed too):

```
{
  "error": [0, 10, "int32"]
}
```

When determining if that a data unit corresponds to a dataformat containing an array, the platform will check that:

- the number of dimensions is correct
- the size of each declared dimension that isn't 0 is correct
- the type of each value in the array is correct

Dataformats: Extension (1)

When, by adding a few fields to an existing dataformat, another useful one can be created, it is possible to **extend** it.

For instance, from the *user/rectangular_area/1* dataformat:

```
{
  "coords": {
    "x": "int32",
    "y": "int32"
  },
  "size": {
    "width": "int32",
    "height": "int32"
  }
}
```

we can create a new dataformat that describes a face:

```
{
  "#extends": "user/rectangular_area/1",
  "eyes": "system/eye_positions/1"
}
```

Dataformats - Extension (2)

A data unit of this dataformat can the be used like that:

```
data.coords.x     = 20
data.coords.y     = 20
data.size.width   = 100
data.size.height  = 100
data.eyes.left.x  = 40
data.eyes.left.y  = 50
data.eyes.right.x = 100
data.eyes.right.y = 50
```

Dataformats: Your turn

Interact: Try to create yourself, a data format for a (2D) **rectangle** (for example, the bounding box for an object in an image):

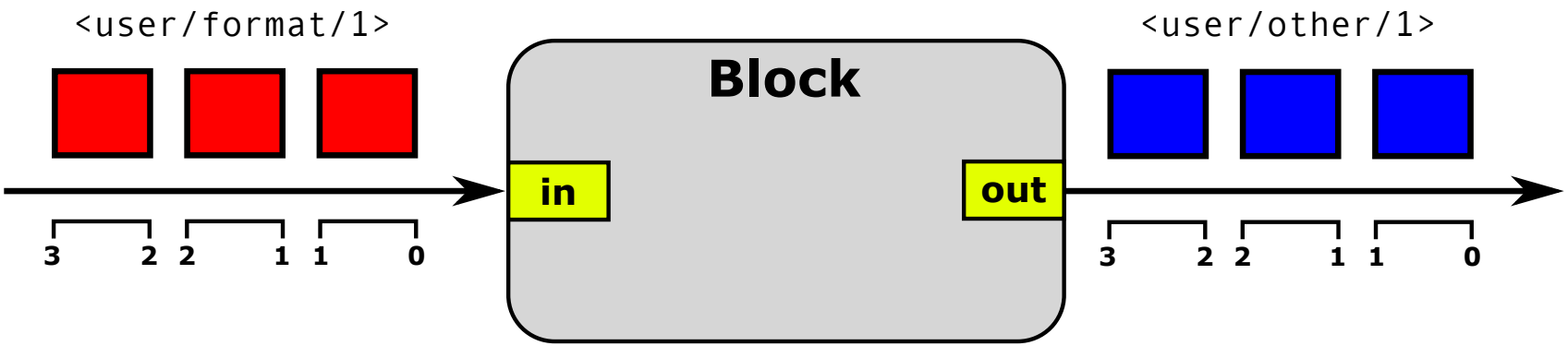
Tips:

1. A rectangle typically contains 4 integers (top-left coordinates, width and height)
2. Less is more: use other dataformats by opening a second tab on your browser and checking what is available
3. Click on the formats to visualize them, read their documentation (if one was written)
4. Try to document your data format well, so you remember what it was for

Filling Blocks with Algorithms

- So that the block we added to our forked toolchain is useful, we must implement at least one algorithm that we can use it with
- Algorithms can be implemented to any number of complex tasks
- For this tutorial, we'll use the *Python* programming language to examplify and create an algorithm for our image pre-processing block.
- We can classify algorithms by their inputs/outputs and their applicability to different types of blocks
- Let's review some of the most common possibilities

Case study #1: "**1-to-1**"



For each **object** received on its sole input, the algorithm should generate one corresponding **object** on its sole output.

Examples

Feature extractors, image pre-processors

Algorithm Example

N.B.: no loops!

```
...
class Algorithm:

    def __init__(self):
        self.extractor = bob.ip.DCTFeatures(12, 12, 11, 11, 45)

    def process(self, inputs, outputs):

        # Convert the image to grayscale
        grayscale_image = bob.ip.rgb_to_gray(inputs['image'].data.pixels)

        # Convert the grayscale image to a floating-point representation
        float_image = grayscale_image.astype('float64') / 255.0

        # Compute the features
        features = numpy.vstack(self.extractor(float_image))

        # Generate a data unit on the output
        outputs["features"].write({'value': features})

        return True
```

Configurable algorithm

This algorithm implementation uses some magic numbers:

```
class Algorithm:

    def __init__(self):
        self.extractor = bob.ip.DCTFeatures(12, 12, 11, 11, 45)
```

It is better to allow the creator of the experiment to choose those values.

Configurable algorithm: Skeleton

A configurable algorithm must have the following structure:

```
class Algorithm:

    def setup(self, parameters):
        # Here, you process user parameters and store
        return True

    def process(self, inputs, outputs):
        # Here, you use parameters to perform the task
        return True
```

Configurable algorithm

```
class Algorithm:

    def setup(self, parameters):
        block_size = parameters.get('block-size', 12)
        block_overlap = parameters.get('block-overlap', 11)
        number_of_components = parameters.get('number-of-components', 45)

        self.extractor = bob.ip.DCTFeatures(
            block_size,
            block_size,
            block_overlap,
            block_overlap,
            number_of_components,
            )

        return True

    def process(self, inputs, outputs):
        # Same as before
```

Data unit generation

In the Python backend, write output data like this:

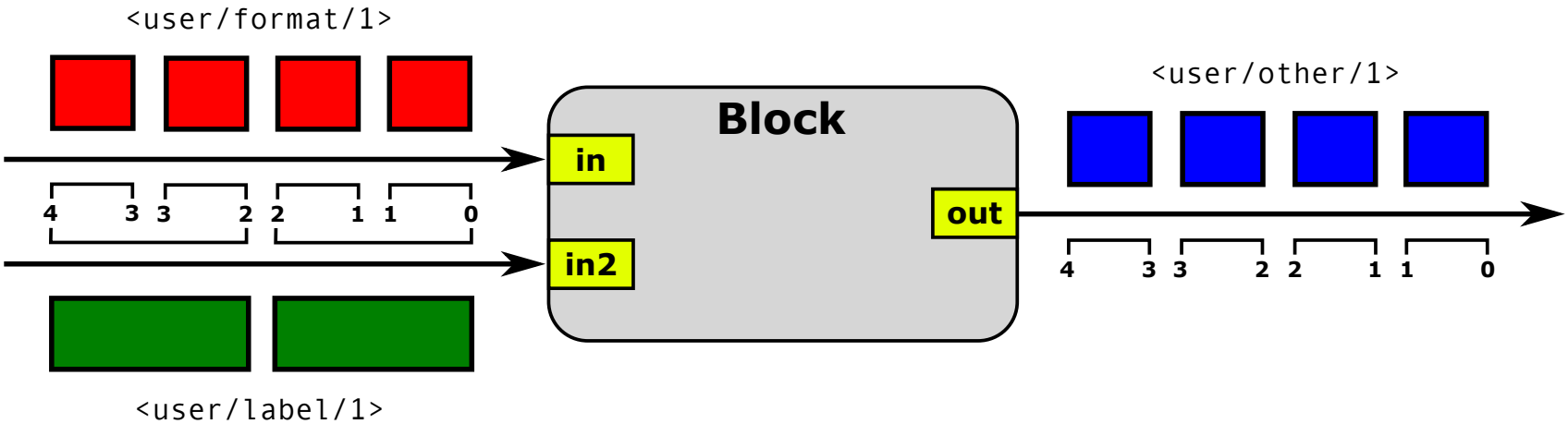
```
outputs["coordinates"].write({
  'y': numpy.int32(20),
  'x': numpy.int32(10)
})
```

The data written must match the structure defined for the type, in terms of keys names for each field **and** the value types. Unsafe casting is **not** allowed. In doubt, wrap the values with the "numpy" homologues of their supposed types.

```
{
  "y": "int32",
  "x": "int32"
}
```

All operations must be **explicit**.

Case study #2: *fancier* "**1-to-1**"



The inputs of this block are **synchronized**: the platform will ensure that, at each execution of the algorithm, the object at the **in** input is related to the one on the *in2* input.

For each synchronized **pair of objects** received on its inputs, the algorithm inside this block generates one corresponding **object** on its "**out**" output.

Examples

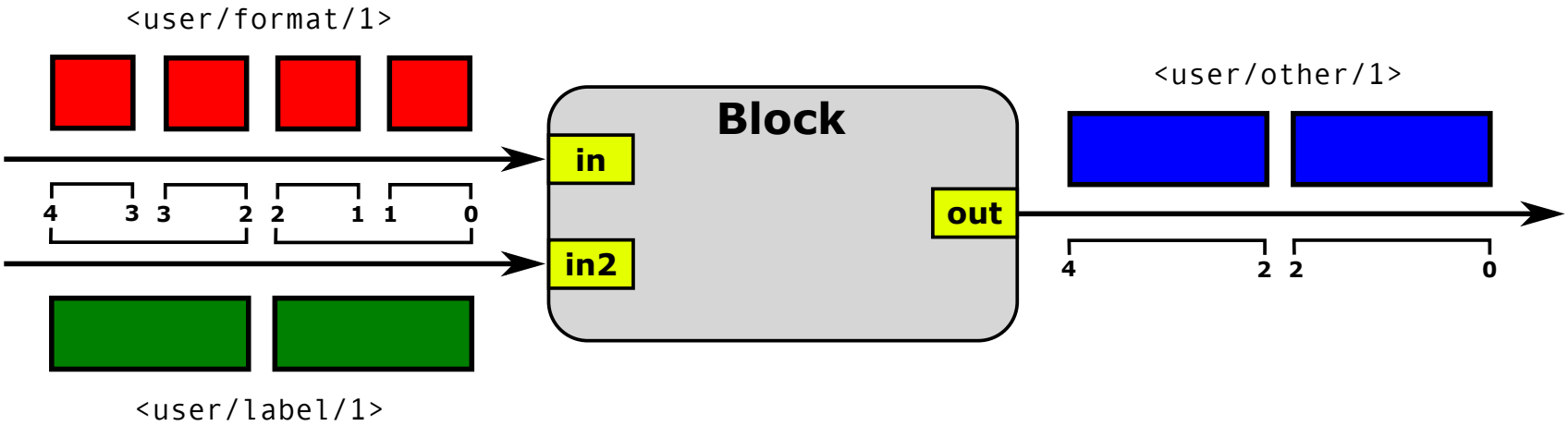
More complex feature extractors and image pre-processors which depend on input parameters (e.g. image quality).

Case study #2: Algorithm (2)

N.B.: no loops!

```
import numpy

class Algorithm:

    def process(self, inputs, outputs):

        pixels = inputs['image'].data.pixels
        label = inputs['label'].data.text

        ...

        features = numpy.ndarray((10, 10), dtype=numpy.float64)
        features[0][0] = 4.0

        ...

        outputs["features"].write({
          'value': features
        })

        return True
```

Case study #3: Synchronized "**N-to-1**"



The inputs of this block are **synchronized**: the platform will ensure that, at each execution of the algorithm, the **object** on the input "**in**" is related to the one on the "**in2**" input.

The algorithm inside this block generates one **object** on its output for each object received on its "**in2**" input.

Examples

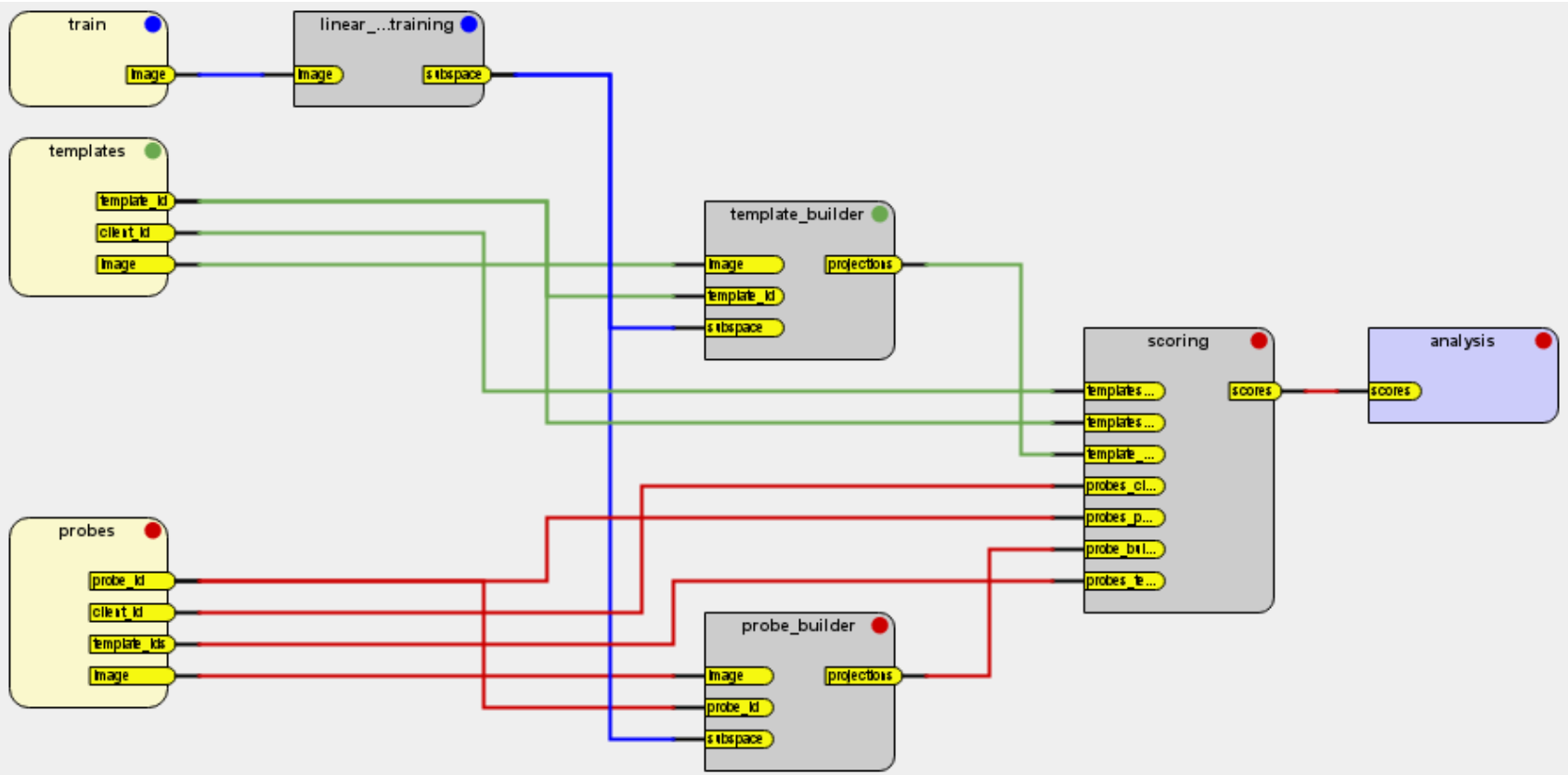
Object-class model training.

Case study #3: Synchronized "**N-to-1**"

N.B.: No loops, branch on `isDataUnitDone()`

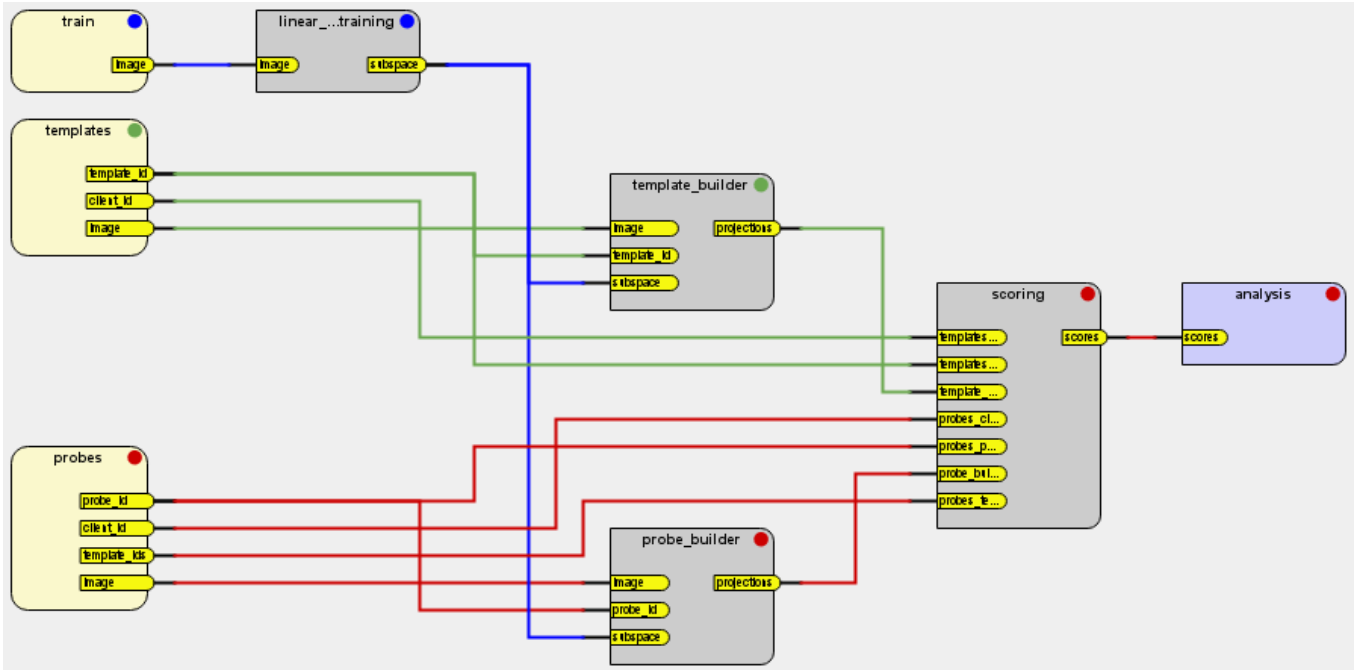
```
class Algorithm:

    def __init__(self):
        self.images = []

    def process(self, inputs, outputs):

        self.images.append(inputs['image'].data.pixels) #accumulates

        if inputs['label'].isDataUnitDone():
            label = inputs['label'].data.text
            ...
            outputs["features"].write({'value': features})
            self.images = [] #reset

        return True
```

Case study #4: **Unsynchronized** inputs



The inputs of the many blocks are connected to two different datasets: **they aren't all synchronized together!**

Synchronization (1)

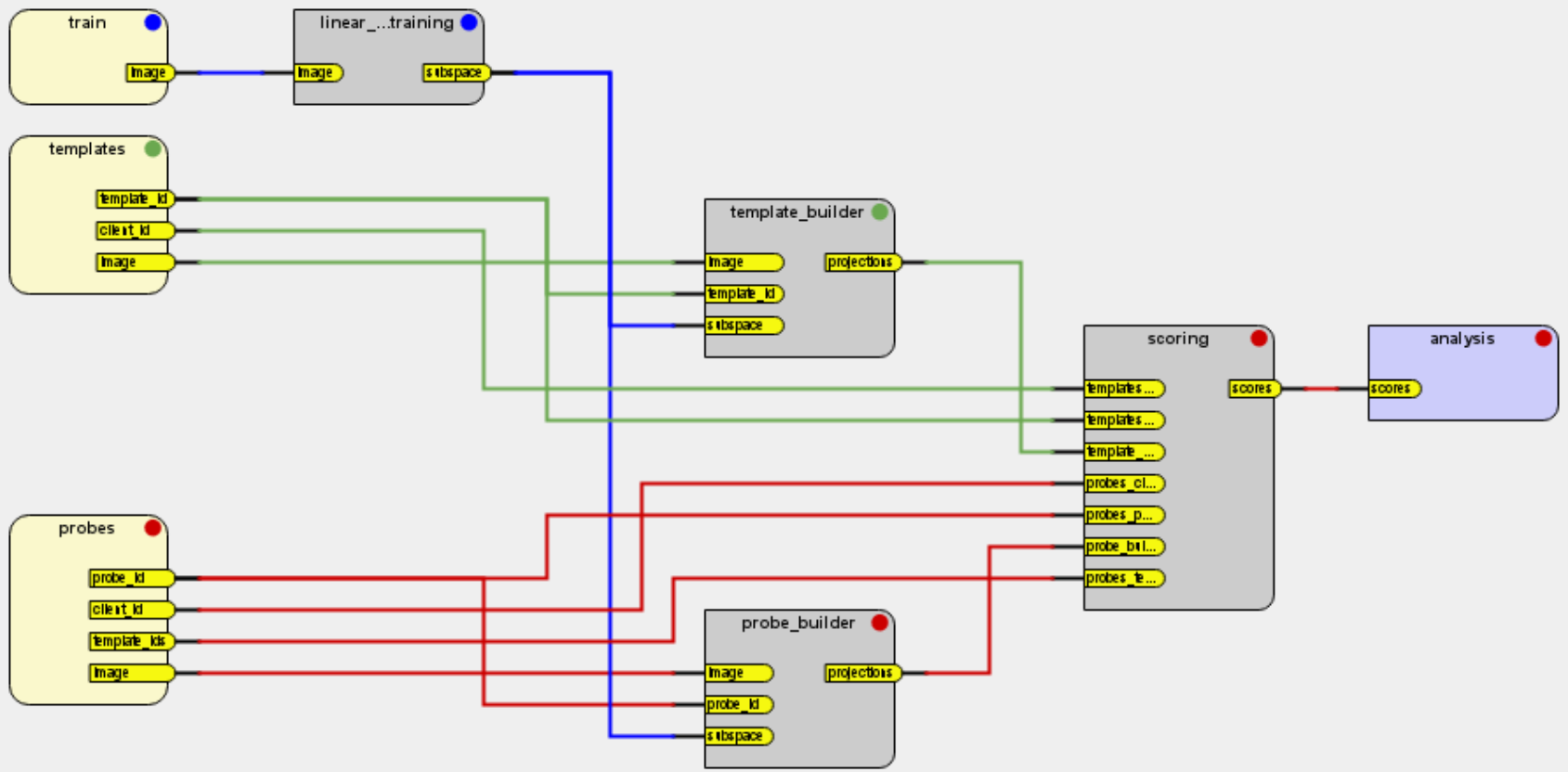


Each *dataset block* is the origin of a **synchronization channel**, sharing its name.

All the outputs of a *dataset block* are **synchronized** together: the *data units* on one output are related to the ones on the other outputs.

Each *connection* is assigned to a **synchronization channel**.

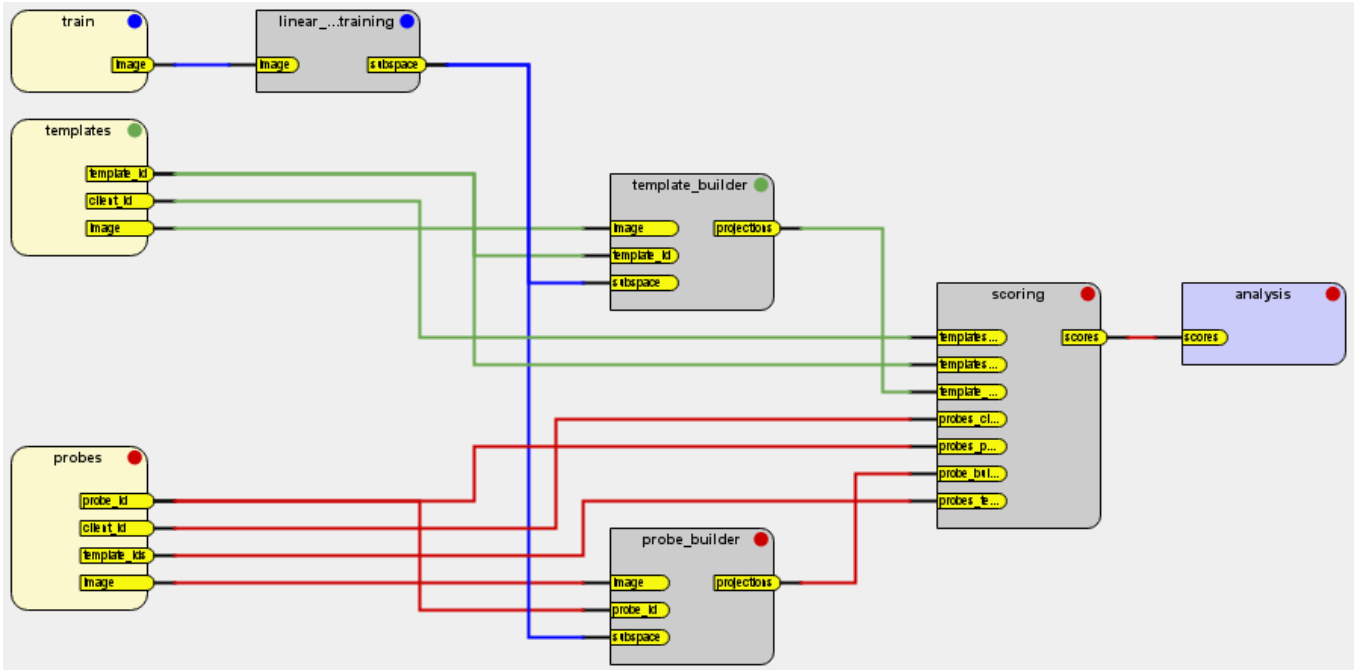
Synchronization (2)



Each *data unit* is assigned an **index** (or a **range of indexes**).

The platform uses this information to determine the relationships between *data units* on different *connections* **that have the same synchronization channel**.

Synchronization (3)

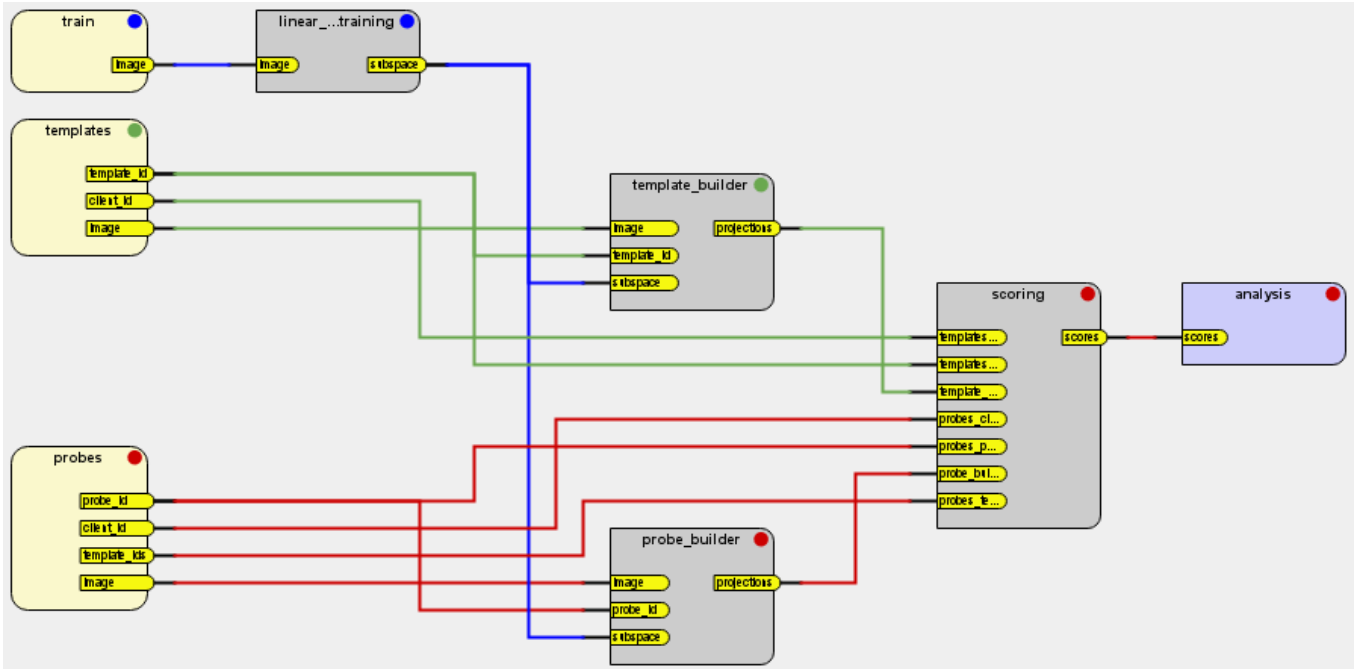


The *inputs* of a *block* are **grouped** by their *synchronization channel*.

Groups aren't synchronized together (not the same *synchronization channel*), but the *inputs* inside each *group* are still able to be synchronized.

Each *block* must select one **group** of its *inputs* as its main one.

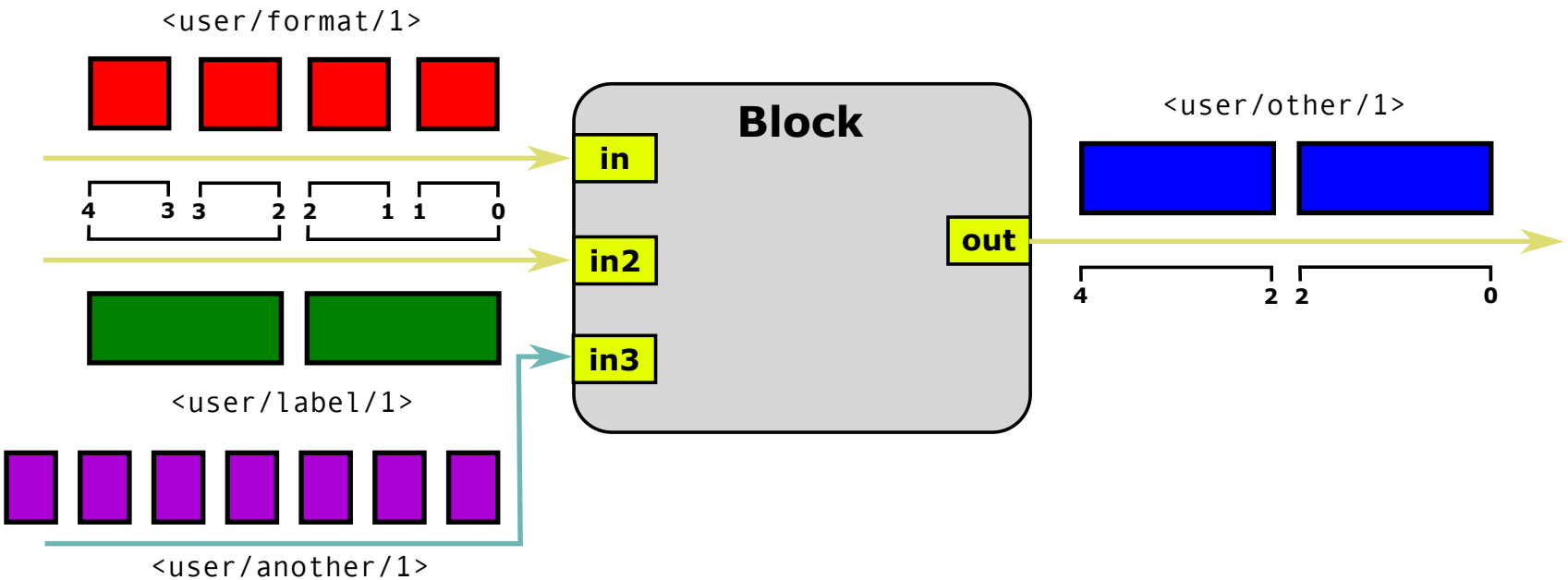
Synchronization (4)



Each time new *data units* are available on the *main group of inputs* of the *block*, the platform will invoke the `process(...)` method of its algorithm.

The *algorithm* is expected to take care of the other *groups of inputs* itself.

Case study #4: Unsynchronized inputs



Case study #4 - Algorithm (1)

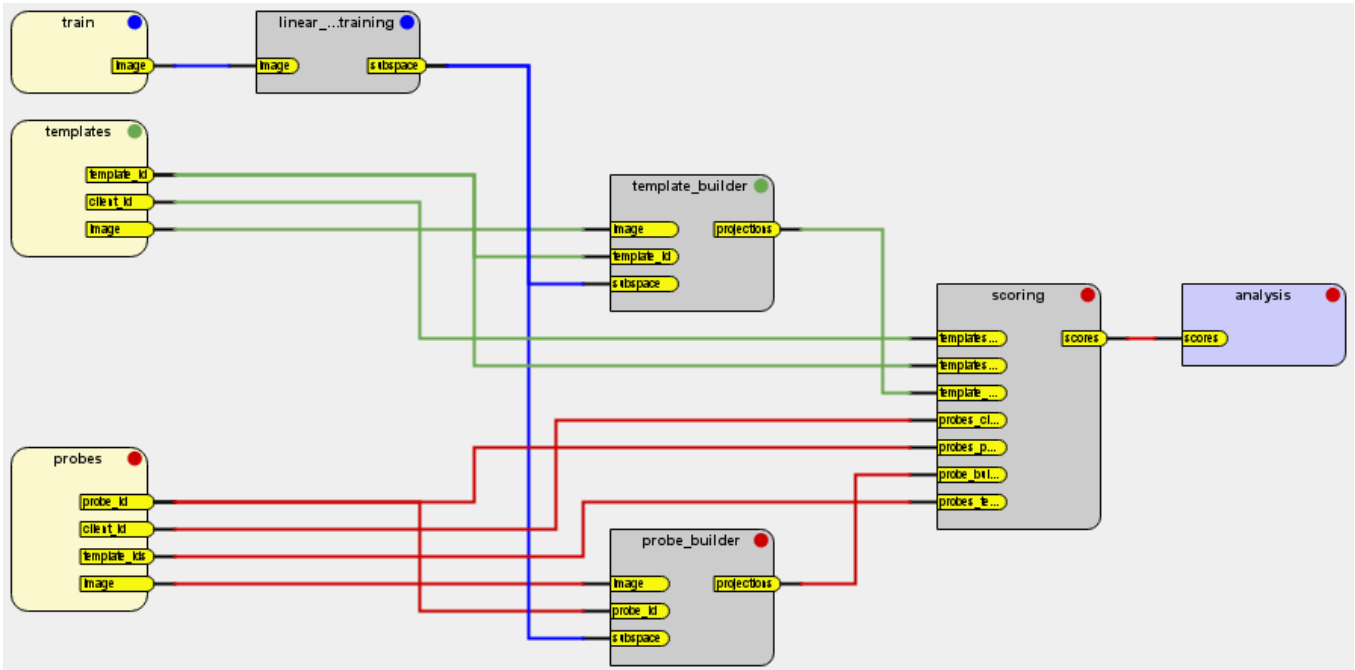
N.B.: loop only on **unsynchronized** inputs

```
class Algorithm:

    def __init__(self):
        self.models = None
        self.score = 0.0

    def process(self, inputs, outputs):

        # At first execution, read all the models
        if self.models is None:
            self.models = {}
            group = inputs.groupOf('model')

            while group.hasMoreData():
                group.next()

                model_label = group['model_label'].data.text
                something = do_something_with(group['model'].data)
                self.models[model_label] = something

        ...
```

Case study #4 - Algorithm (2)

```
def process(self, inputs, outputs):

    ...

    # Process the current test image/label pair
    image = inputs['image'].data.pixels
    label = inputs['label'].data.text

    self.score += ...

    # After the very last test image, writes the score on the output
    if not(inputs['image'].hasMoreData()):
        outputs['score'].write({
          'value': self.score
        })

    return True
```

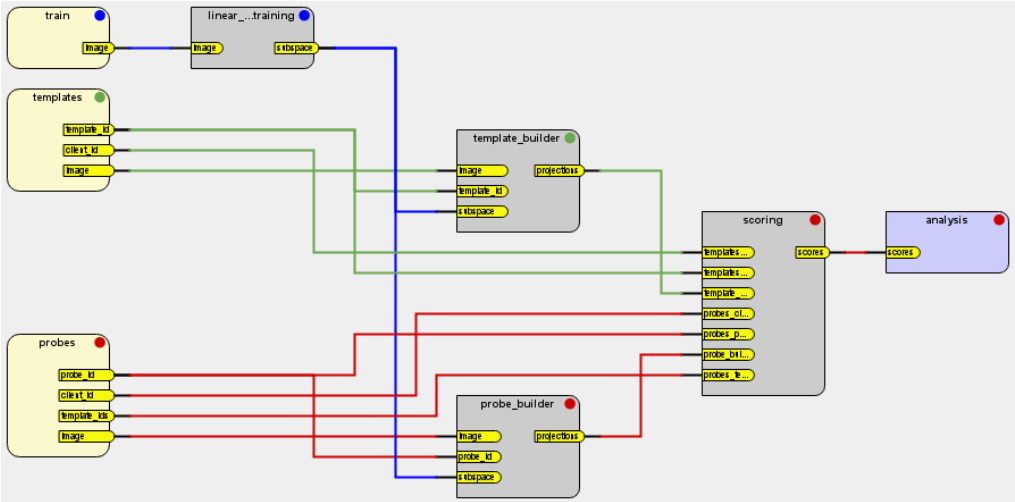
Databases



Databases provide special algorithm blocks that implement dataset blocks in Toolchains

Interact: Now click on the tab “Databases” and then select the “atnt” database, scroll down.

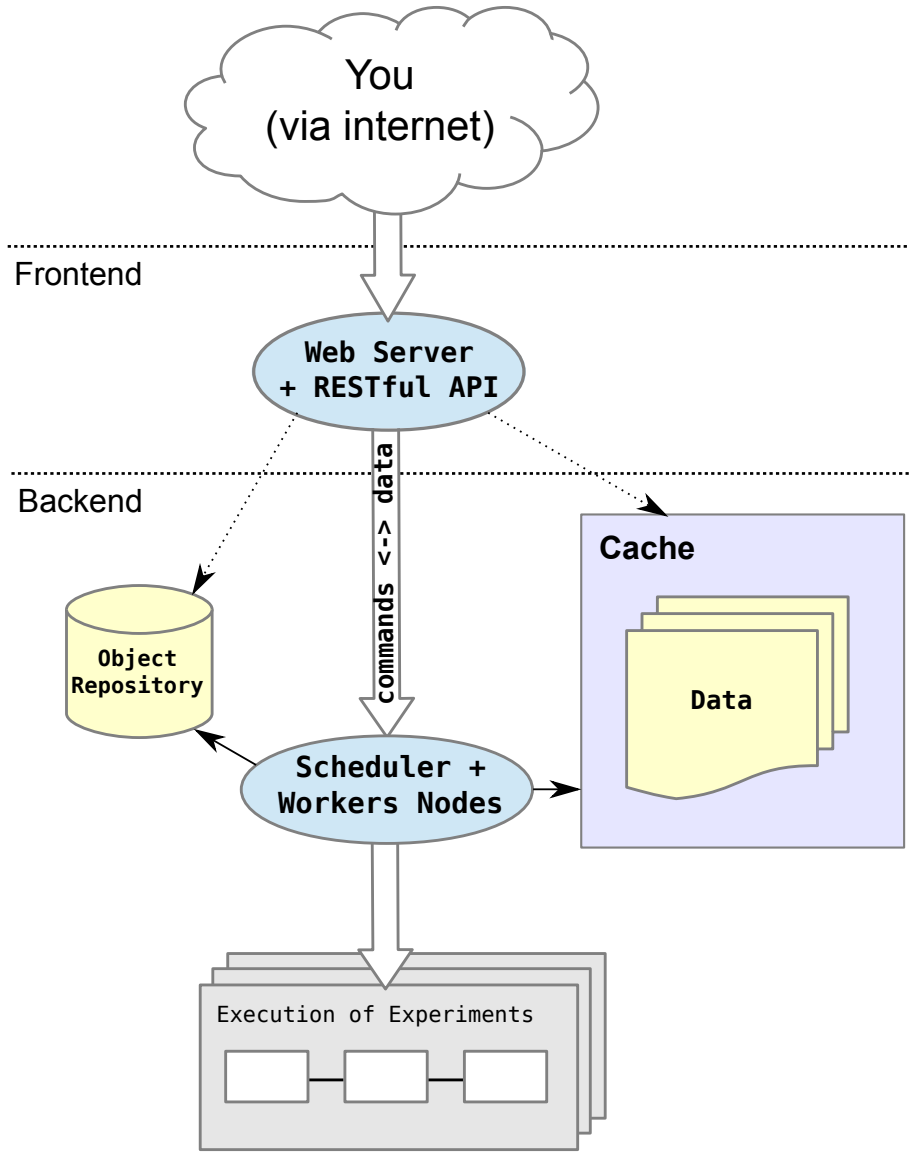
Organization



- Each Database defines a number of protocols (how to use its data in a toolchain)
- Databases have a **name** and a **version**
- Each Protocol defines a number of Sets
- Each Protocol conforms to a Template
- Each Set defines a number of outputs, with a **precise type**
- So that a Set can be used in a Toolchain (inside a Dataset), there must be a **match** between the number of outputs in the said Set as well as it must be **compatible** the input of connected blocks

Backend

The web GUI is just a front end to a powerful computing grid



Queues

Our backend is organised into processing queues.

Queues:

- Have a name
- Have a number of *slots* associated to it
- Can be shared with people or groups

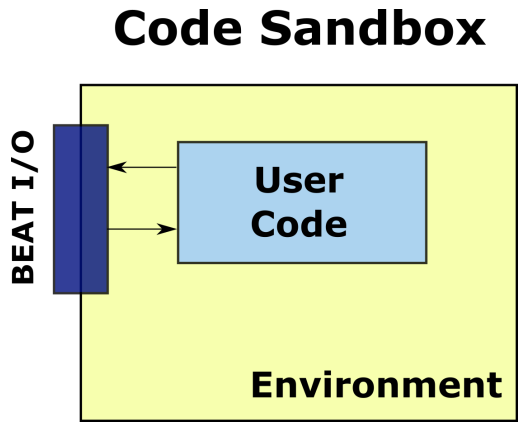
Slots

A *slot* is an abstract concept putting together:

- A number of cores in a host, with a base OS installed
- Memory (RAM)
- A base operating system
- Processing environments

Processing Environments

A processing *Environment* is a sandbox for your algorithms:



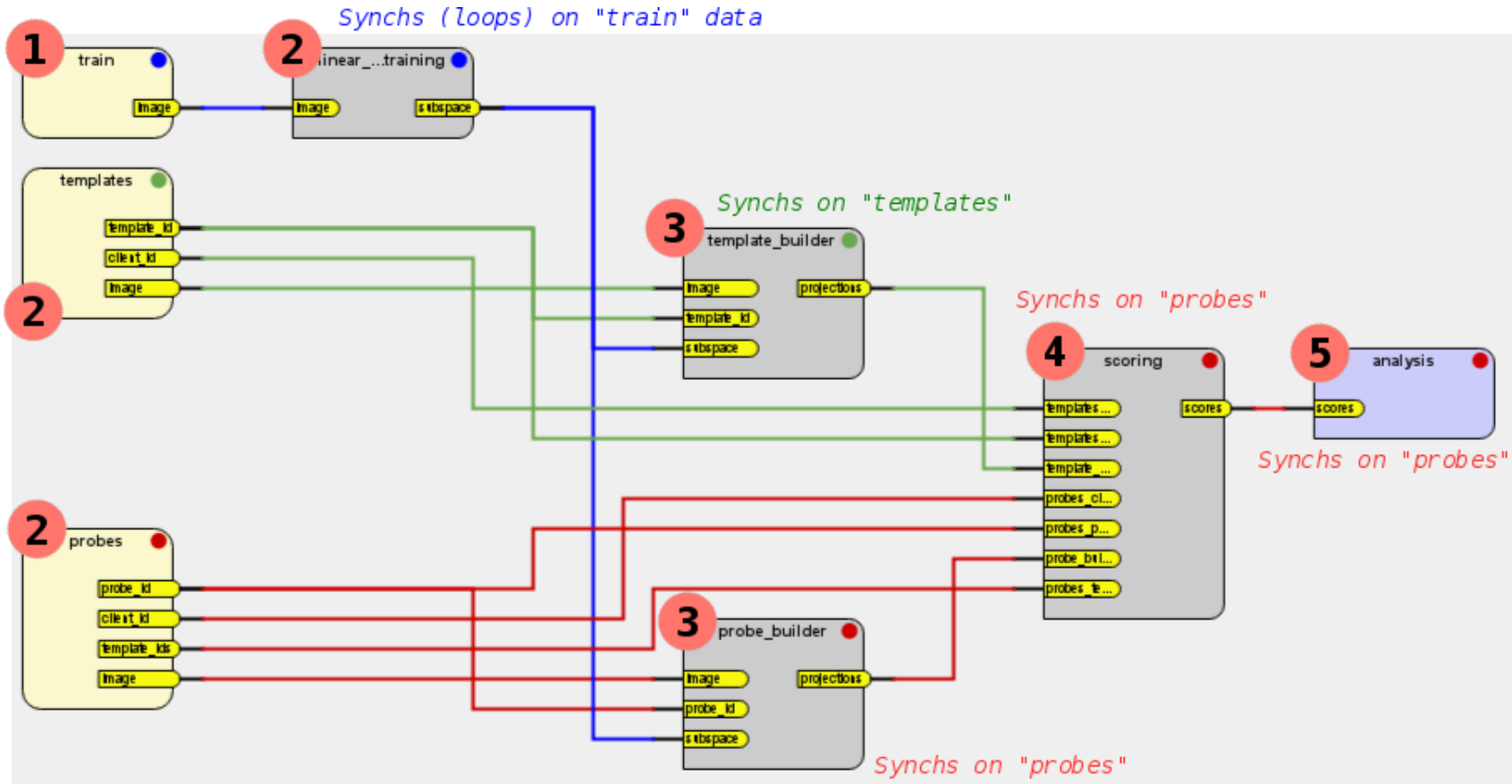
- A set of installed software that can be used on user algorithms and libraries for a given host/OS
- An API to the BEAT dataformat I/O structure
- A binary API to our Workers

Processing Strategy

Simple: 1 block $=$ 1 slot ($\sim$ one process)

- Each process is independent of each other
- The sandboxes only read and write data from a (rather big) disk cache
- This allows for an heterogenous processing farm
- This allows for **automatic** map-reduce (parallelization)

Processing Strategy (2)



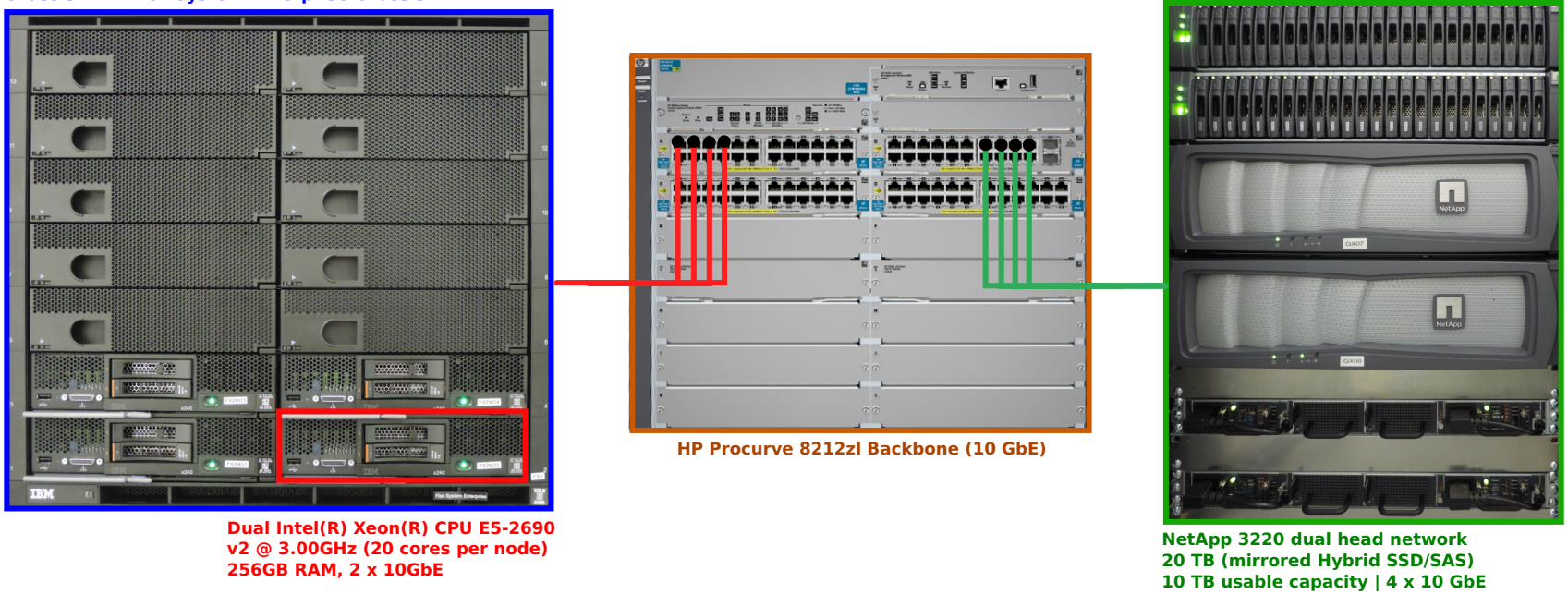
To sum-up

Because of this design:

- It is possible control computing power and privileges of our users
- You can use an heterogenous farm of resources
- You can automatically reduce processing time by applying parallelization
- A central scheduler is responsible for assigning priorities and balancing the computing farm

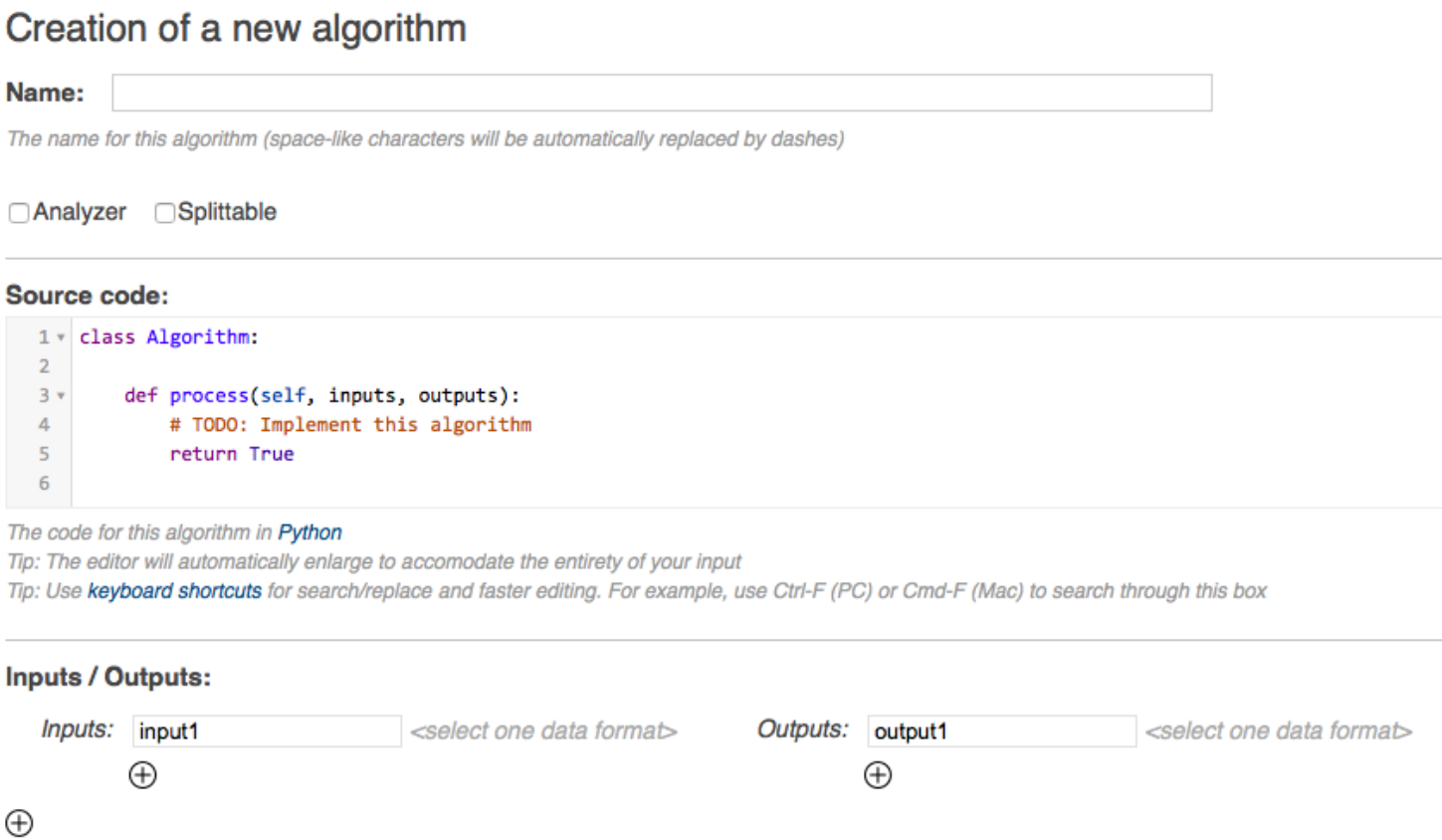
Current state

Initial configuration: 80 cores (12 Gb RAM/core), 10 Tb Cache, All nodes are virtualized



Create a parallelizable algorithm

We'll create together, an algorithm that is parallelizable and you'll use it on your new toolchain for experiments.
Interact: Click on the "Algorithms" tab, hit "New" (green button)



Example: Histogram Equalization

This is an histogram equalization algorithm that fulfills these criteria:

tutorial / histogram_equalization / 1 splittable — Fork

Status: Private

Histogram equalization for grayscale image [more] [edit]

Source code | History

```
1  import numpy
2
3  class Algorithm:
4
5      def __init__(self):
6          self.nb_bins = 256
7          self.normed = True
8
9      def process(self, inputs, outputs):
10
11         raw_image    = inputs["raw_image"].data.value
12         imhist, bins = numpy.histogram(raw_image.flatten(), self.nb_bins, normed=self.normed)
13         cdf = imhist.cumsum()
14         cdf = 255 * cdf / cdf[-1]
15         output_image = numpy.interp(raw_image.flatten(), bins[:-1], cdf).reshape(raw_image.shape).astype(numpy.uint8)
16         outputs["image"].write({
17                 'value': output_image
18             })
19         return True
20
```

Inputs / Outputs

Inputs: raw_image system/array_2d_uint8/1 *Outputs:* image system/array_2d_uint8/1

Experiment

Interact: Go back to the experiment tab, click new and now select *your* toolchain.

Let's, together, examine it details:

- Toolchain display
- Global parameter section
- Database configurator
- Default processing queue
- Local parameters and processing queue
- Parallelization
- Automated filtering of algorithms

Hit the "Go!" button.

Search

Interact: Run a few experiments.

Search for “Similar experiments” using the top-bar or hitting on the button with that name:

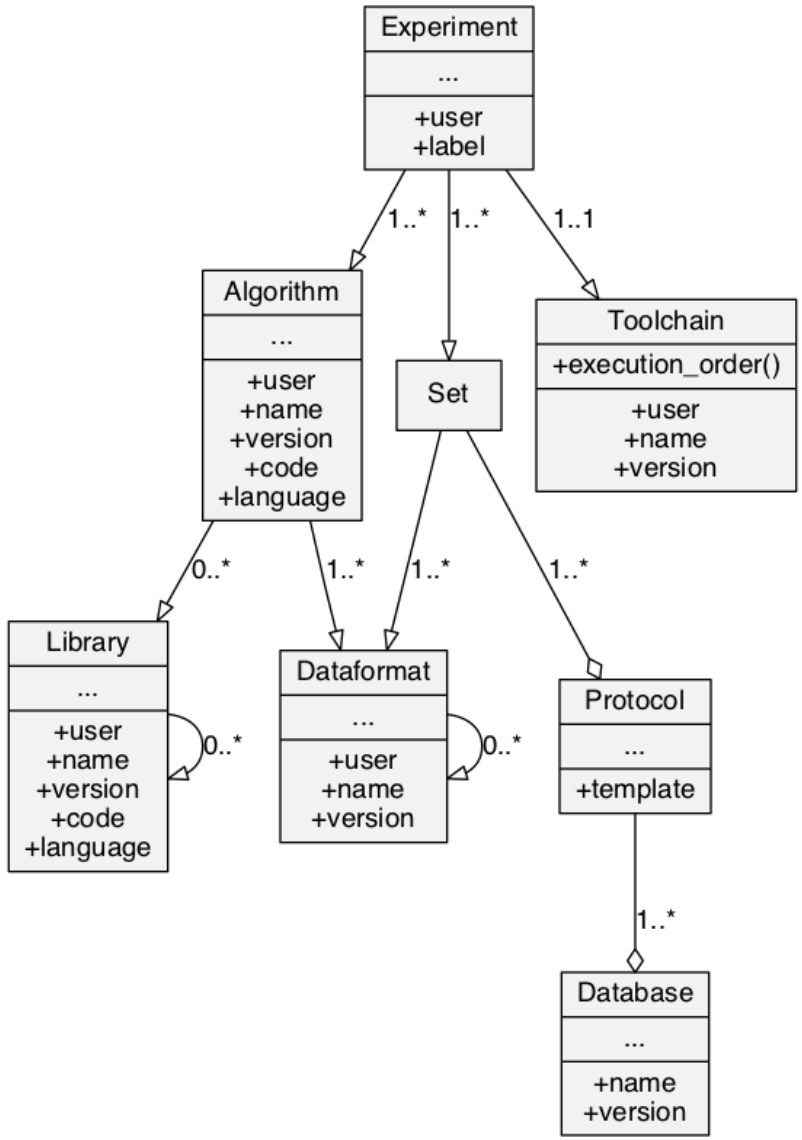
- Compare these results
- Save this search for future reference

A note on “modify-ability”

A key aspect in BEAT is “reproducibility”:

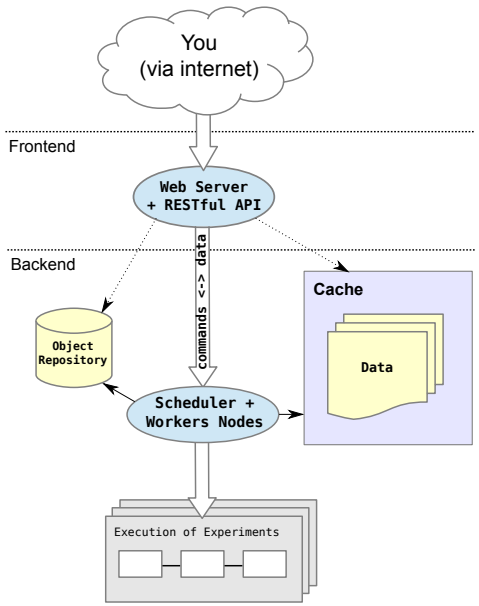
- If you use a component in any manner, you can no longer modify it
- Implications:
  - If you use an algorithm/dataformat/library in an experiment, you cannot delete/modify it any longer
  - If you attest an experiment, everything it depends on cannot be modified any more

Locking hierarchy



Caching

- BEAT keeps track of data transmitted through all stages of the processing
- It caches the data in a large disk array (∼10 Tb)
- The cache is invalidated automatically when things change:
  - Operating System or installed packages are updated
  - Toolchain changes
  - Database version changes
- One cached item is valid for a specific combination of all of the above
- If a cached item is available, it is used to speed-up processing.

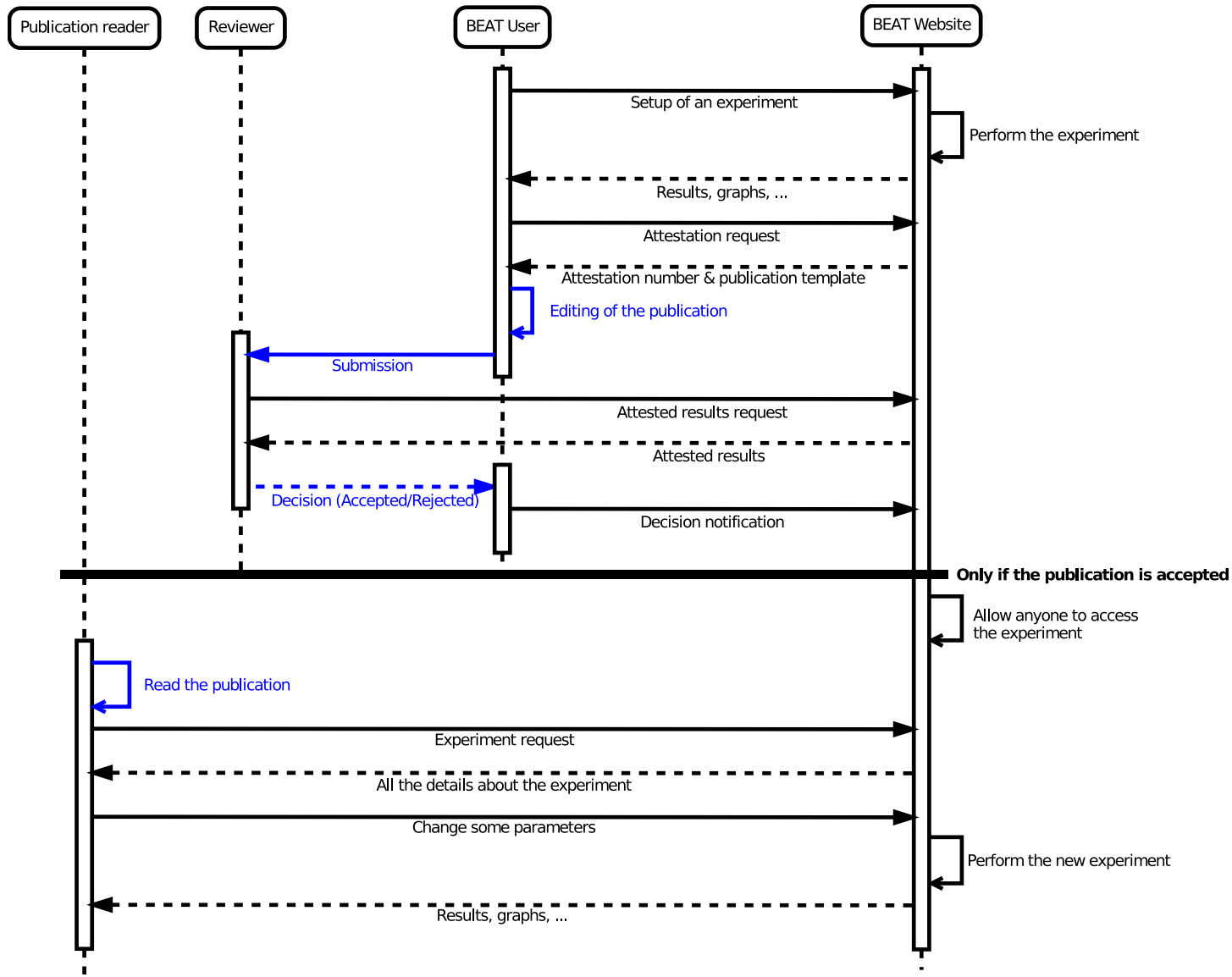


Attestation (Certification)

An attestation mechanism is available in the platform.

- Allow 3rd. party verification of results obtained with a given configuration
- Allow for a scientific *review* process to take place in confidentiality
- N.B. An attestation **permanently** locks an experiment

Attestation: Message Flow



Libraries

You can also implement libraries, to apply DRY ("Don't repeat yourself"):

- Libraries are like algorithms, except no input/output
- You can combine libraries with other libraries in order to build complex systems
- You can combine libraries wih algorithms
- Follow the same modify-ability as for other objects

Planned Activities, Issues (1)

Priority scheduling:

- Currently, our scheduler just implements FIFO scheduling
- Improve to introduce priorities:
  - Wait time
  - Number of jobs
  - Forced priority

Planned Activities, Issues (2)

Reputation

- Gamify the platform
- The more users use from your stuff, more priority on processing
- Give opportunities for people with low computing power
- More to people that return more to the community

Planned Activities, Issues (3)

Plotting

- The plotting backend is quite advanced already
- Allow users to parameterize plotting
- Make this available through a simple interface on your micro-site

Planned Activities, Issues (4)

Search, report feature

- Introduce more extensive search criteria. E.g.:
  - Show me all results with CPU time $<$ *Xminutes* for this particular block
  - Tabulate EER against a certain parameter value
- Export tables and figures (and its data) into re-usable text, images and tables for your publications (LaTeX)
- Save reports for easy reference

Planned Activities, Issues (5)

Documentation & Public Release

- Our API documentation is already pretty solid
- Work on our main user guides
- Work on administration documentation
- **Public release** planned for begin of 2016

Planned Activities, Issues (6)

Remote Software Development Kit (SDK)

- Using our API, it is possible to interact remotely with the platform
- You download/upload objects, start experiments
- We plan to integrate all this into a virtual image with the required software so that you can create run your experiments locally, on a debugging environment
- N.B.: The raw data will not be available on the SDK (privacy requirement). You must procure it yourself.

Keeping in touch

We created a Google Groups forum for discussion. Join to:

- Keep up-to-date with the platform development and the Idiap prototype
- Request features and database inclusions
- Discuss issues and other problems you may be facing

Details:

- View posts: https://groups.google.com/d/forum/beat-devel
- Subscribe: beat-devel+subscribe@googlegroups.com
- Post: beat-devel@googlegroups.com