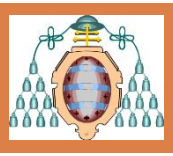
PROOF ANALYSIS FRAMEWORK

Isidro González Caballero
& Javier Delgado Fernández



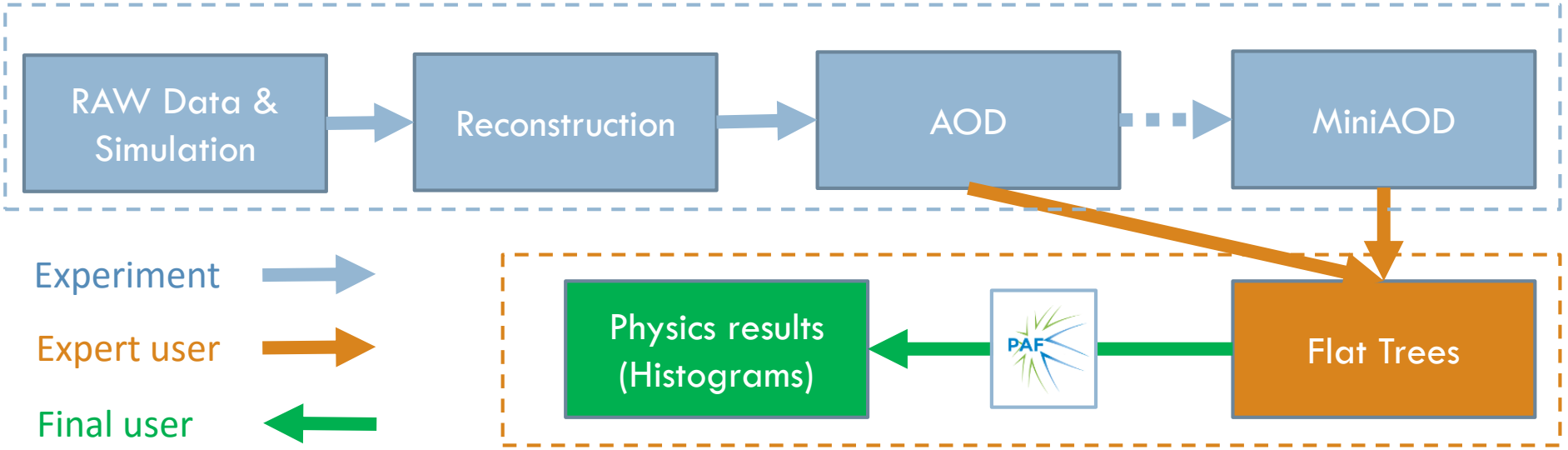
Universidad de Oviedo

ROOT Users' Workshop 2015

Outline

- Why? – The use cases
- What? – Description
- How? – The details and examples
- Where? – PAF Environments
- When? – The present situation and the future
- Who? – Developers, users, documentation…

Why? – A typical HEP analysis scenario



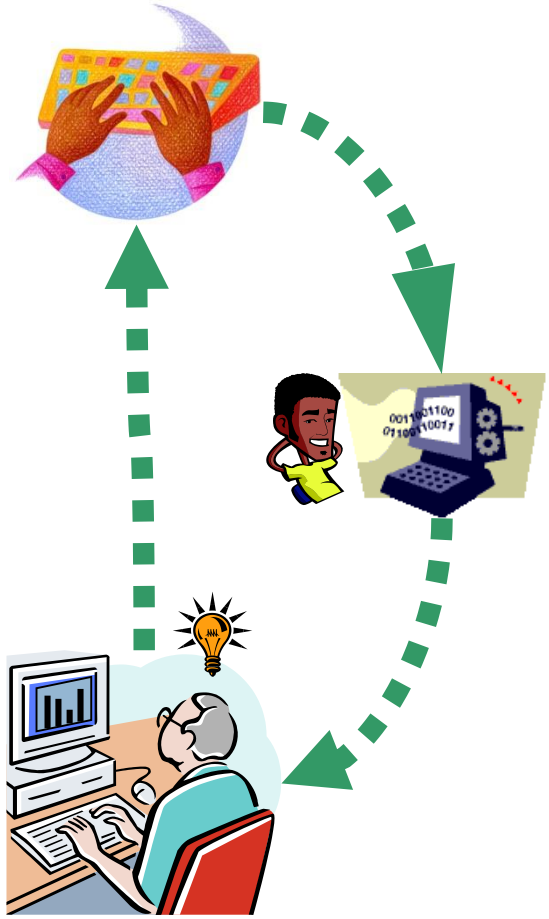
- After reconstruction, skimming, slimming, … what is left is a (more or less) flat tree with the relevant variables and events
  - Size: 100 MB – 10 GB per sample
  - Total data to process: 100 GB – 10 TB per analysis

Why? – The computing resources around

- Heterogeneous situation among institutions
  - CPU: From multicore workstations to local clusters with batch, grid or cloud systems
  - Storage: From few local TB to dedicated disk servers with O(100 TB)

- In many situations these resources are underused during the analysis development phases
  - At the beginning only process a (few selected) sample(s)
  - Only when things are stable go for the whole data

- Drawbacks of usual strategies:
  - ➔ Local root macro only uses a core
  - ➔ Batch system response time driven by biggest sample
    - ➔ If sample split in several jobs, manually merge results at the end

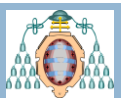


What? – PROOF

- PROOF stands for Parallel ROOT Facility
  - It is an extension of ROOT enabling interactive analysis of large sets of ROOT files in parallel on clusters of computers or many-core machines.
- The main design goals for the PROOF system are:
  - Transparency: running a PROOF session should not be very different from running a ROOT session
  - Scalability
  - Adaptability to variations in the remote environment
- It is based on the `TSelector` model by ROOT
- IMHO, PROOF is a very nice and complete framework
  - But using it at the level required for a serious analysis introduces some complexities that may scare new adopters

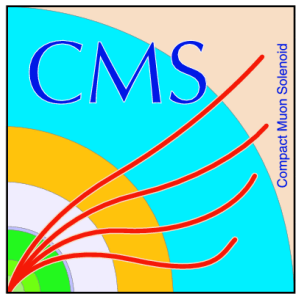
What – PROOF Analysis Framework (PAF)

- The PROOF Analysis Framework (PAF) is a tool to **easily and quickly** implement **distributed analysis** over ROOT trees
  - Physicists should concentrate on analysis rather than on software or computing
  - Migration from a typical ROOT based sequential analysis should be very easy
- PAF **hides as much as possible the inherent complexities** of parallel paradigms to the final users
  - Taking care of the tedious and repetitive tasks as much as possible (setting the environments, packaging and uploading code, passing information to the WNs, …)
  - Setting sensible default values for configurable parameters (still maintaining access to them)
  - Making smart decisions when possible
- PAF **provides a common framework for** different distributed computing technologies:
  - Uniformly exposing the PROOF related configurations across technologies
  - Taking advantage of all the cores in modern CPUs through **PROOF Lite**
  - Building dynamic PROOF clusters through **PROOF Cluster** or **PROOF on Demand**
  - (or even pure sequential processing)

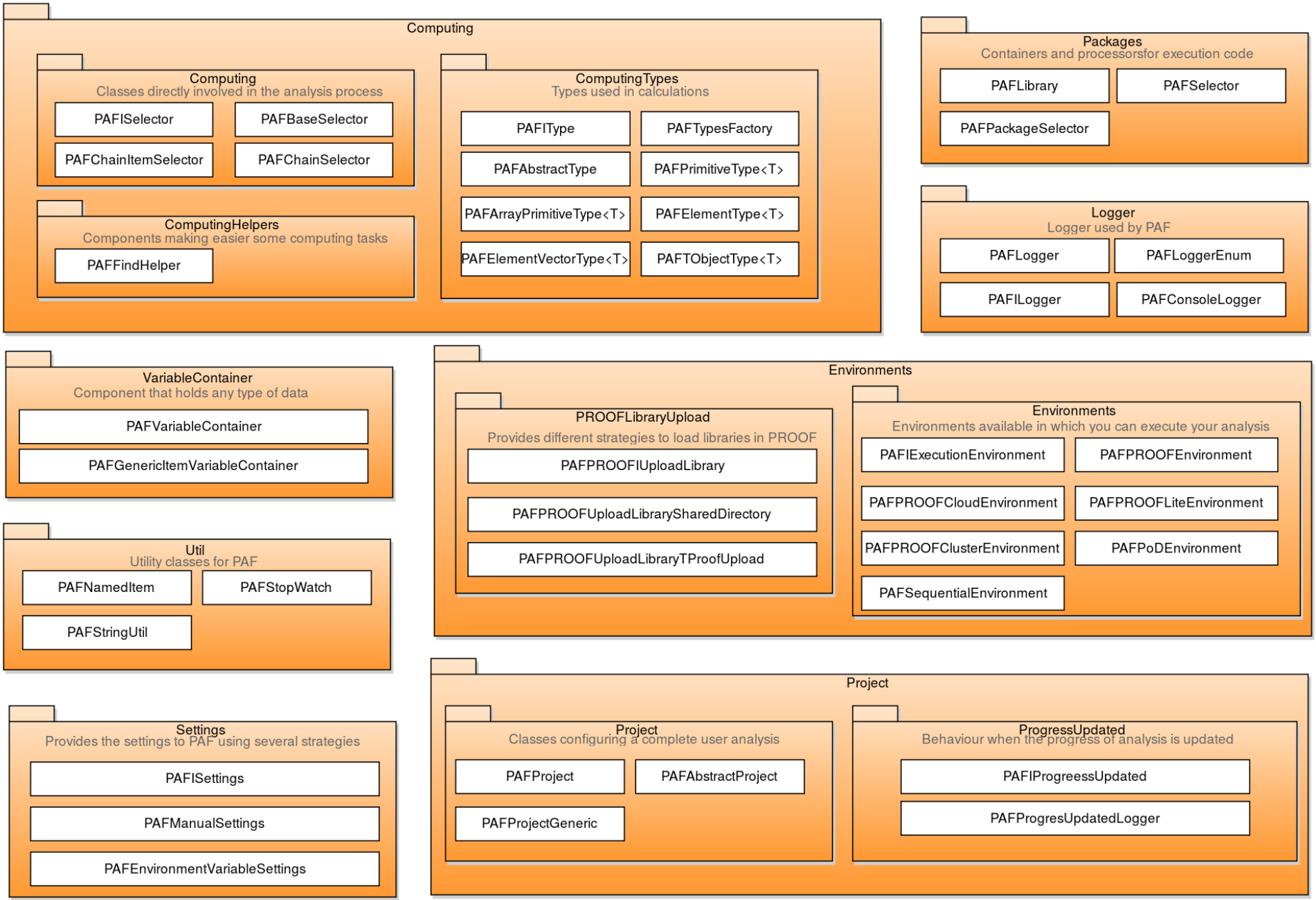


What? – PAF history

- Initial release in 2006
  - With little changes until now
  - It has served well the CMS community at U. Oviedo and IFCA during LHC Run I
- During the last 6 months we have completely re-engineered PAF → V5.0.1
  - PAF has now a strong object orientation:
    - → Easier to understand and modify the system, particularly for new developers
  - Enforcing a modular architecture design
    - → Flexibility to adapt to new scenarios
  - Providing interfaces to change almost any functionality



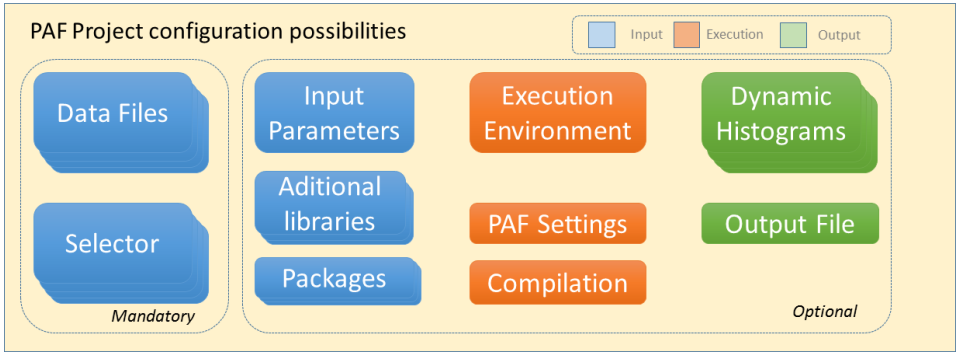
How? – The big (probably useless) picture



How? – Everything starts with a project

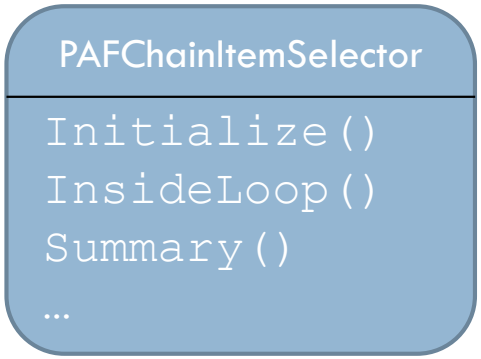
- Project paradigm to configure an execution:
  - Easy and intuitive configuration
- `PAFProject` has lots of parameters
  - Only a couple are mandatory:
    - Input data
    - Selector name (see next slide)
  - Default values and smart decision taking for most of them
    - Almost any aspect is configurable

```
void MyProject()
{
    PAFProject p;
    p.AddDataFile("ROOT file name");
    p.AddSelectorPackage("SelectorName");
    p.Run();
}
```



How? – Selector for physics

- The physics code is encapsulated into one (or several) selectors
  - Inheriting from `PAFChainItemSelector`
  - The processing is split in the usual three hooks



- `Initialize()`
  - → Actions needed before going through the events
    - For example: create and register histograms, trees, profiles…
- `InsideLoop()`
  - → Actions performed for each event
    - For example: Select events, fill histograms, …
    - Lazy loading of data
- `Summary()`
  - → Actions needed after processing all the events
    - For example: Print some summary output

```
TTree* CreateTree(const char* name, const char* title);
TH1F*  CreateH1F(const char* name, const char* title,
                 Int_t nbinsx, Axis_t xlow, Axis_t xup);
TH1D*  CreateH1D(const char* name, const char* title,
                 Int_t nbinsx, Axis_t xlow, Axis_t xup);
TH2F*  CreateH2F(const char* name, const char* title,
                 Int_t nbinsx, Float_t* xbins,
                 Int_t nbinsy, Float_t* ybins);
```

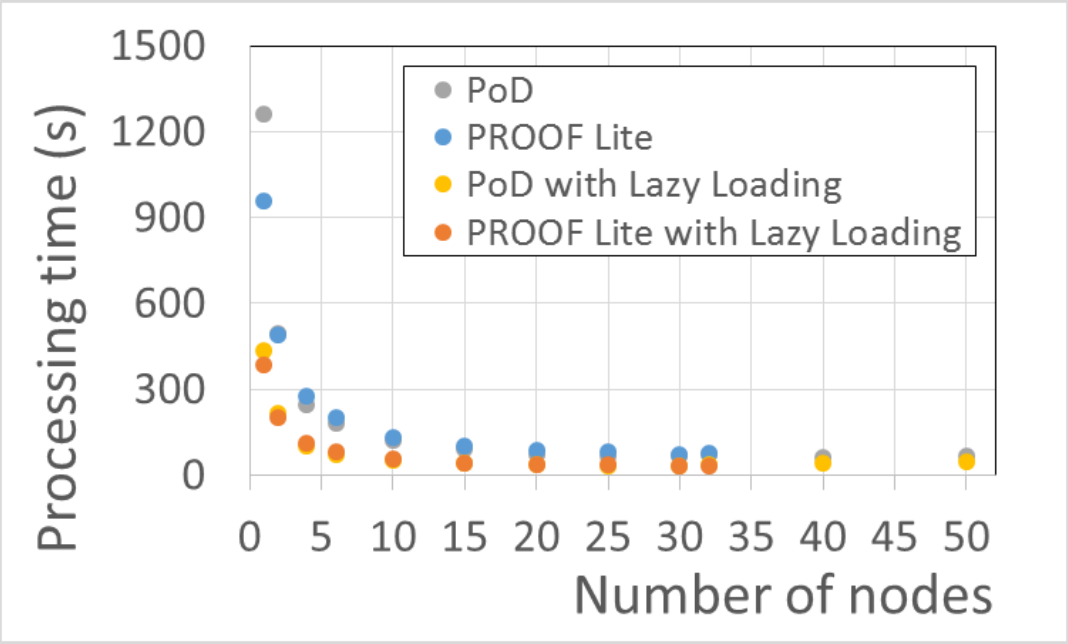
```
template<typename T>
T Get(const char* key);
Int_t GetInt(const char* key);
Float_t GetFloat(const char* key);
Double_t GetDouble(const char* key);
```

How – A basic selector


```
#include "BasicSelector.h"
#include "TCanvas.h"

ClassImp(BasicSelector);

void BasicSelector::Initialise() {
  etHisto = CreateH1F("etHistogram","#slash{E}_{T}", 100, 0., 200.);
}

void BasicSelector::InsideLoop() {
  float t_metpf_et = Get<float>("ootpum2");
  etHisto->Fill(t_metpf_et);
}

void BasicSelector::Summary() {
  TCanvas* canvas = new TCanvas("canvas", "Proof ProofFirst canvas");
  TH1F* result = FindOutput<TH1F*>("etHistogram");
  result->Draw();
  canvas->Update();
}
```

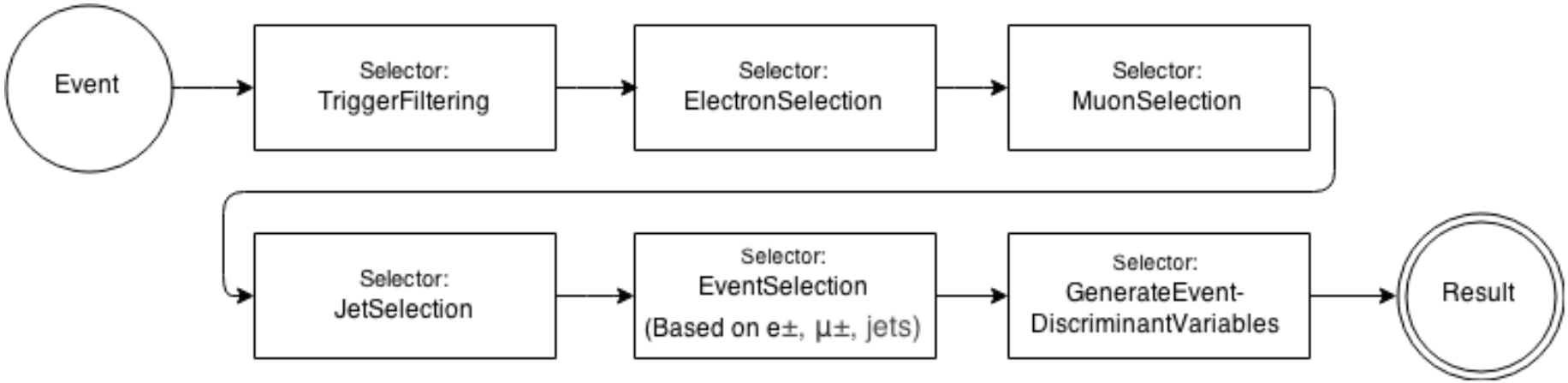
How? – Lazy Loading

- Smart trick to dynamically tell ROOT which branches are used
  - … and therefore read from the file
  - … faster I/O
- Speed up by a factor 2-10 the whole processing time



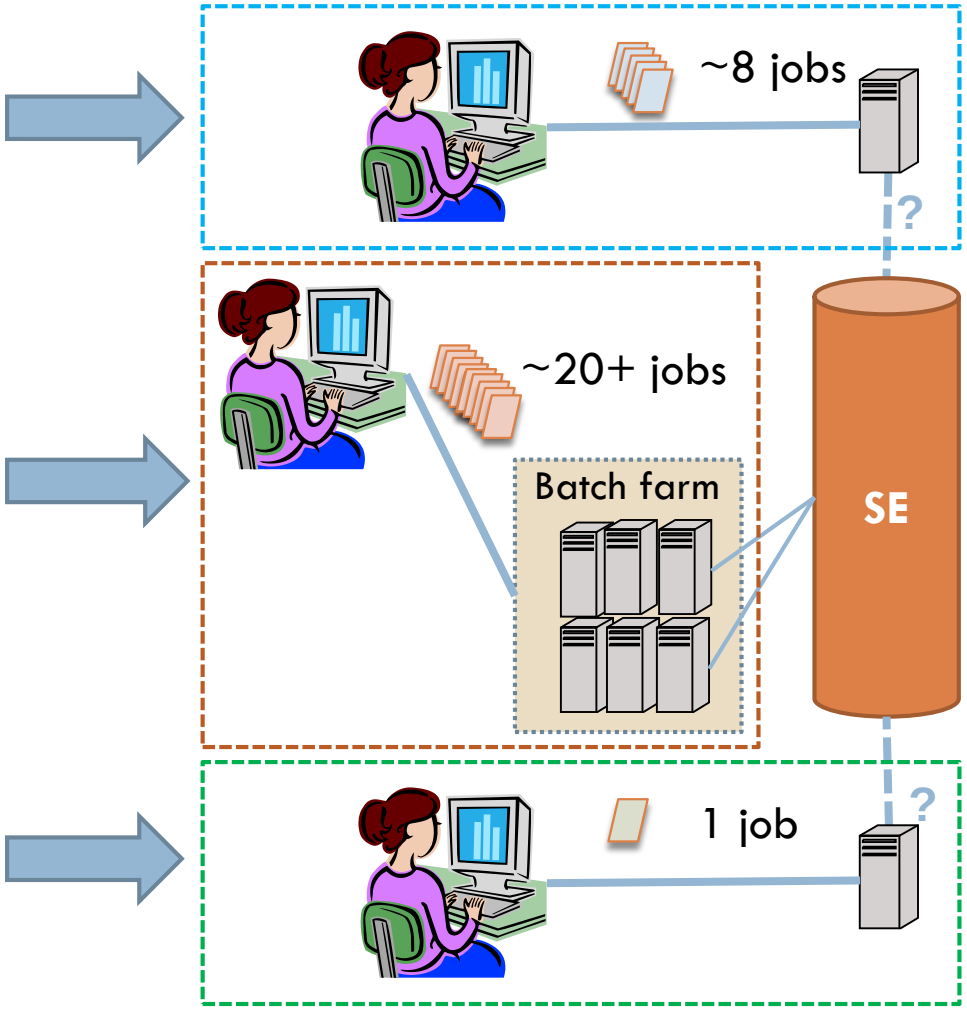
How? – Selector extended for modular analysis

- Support for chained selectors introduced in new version:
  - Modularization: Cleaner, easier to understand, atomic tasks…
  - Reusability and sharing of the selectors in different analysis
- Mechanism to pass messages among sub-selectors introduced
  - The same used to get information from the project

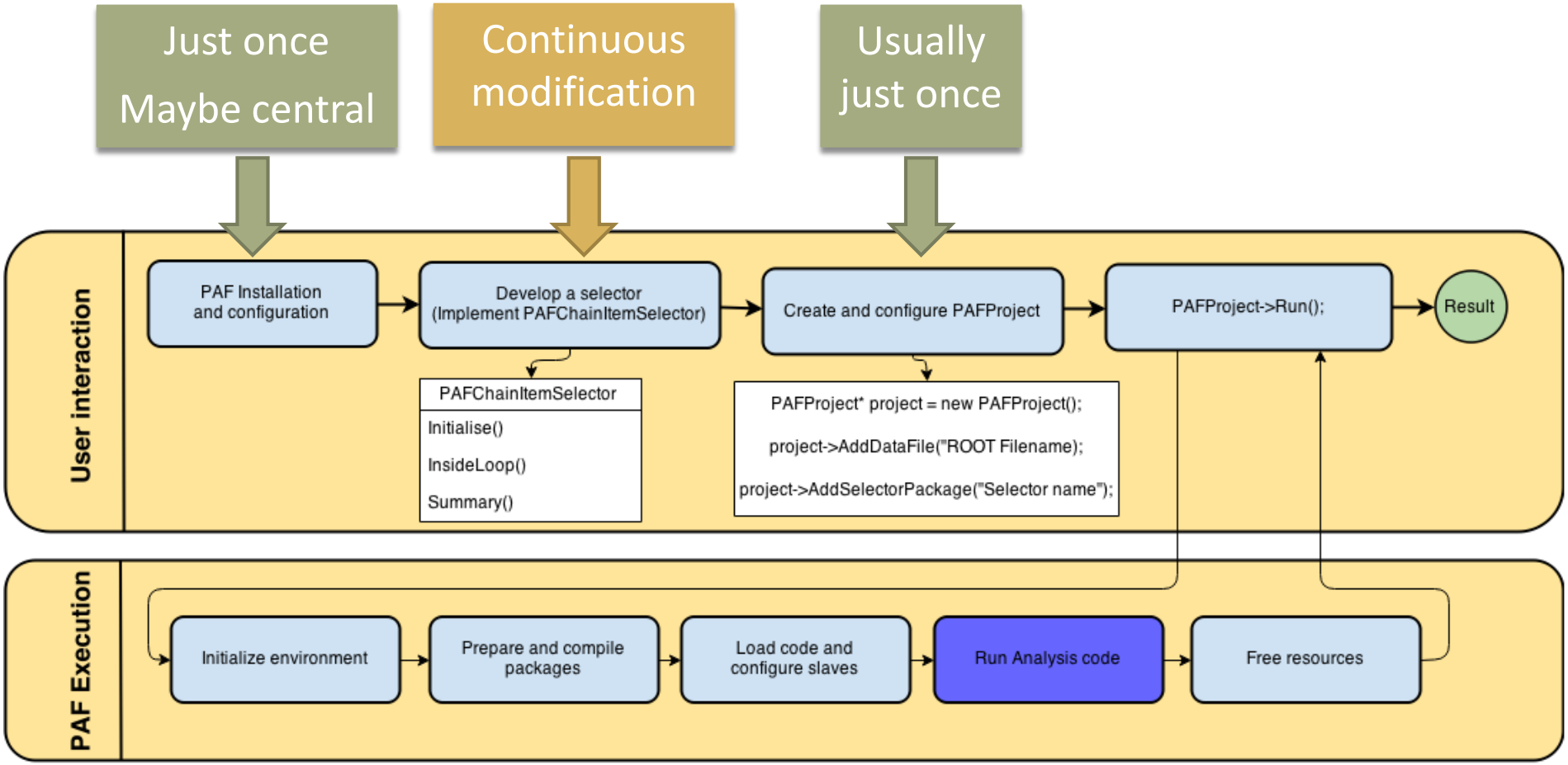


Where to use PAF? – PAF Environments

- PROOF Lite: Included in ROOT
  - Ideal for multicore machines
  - No central setup needed
- PROOF Cluster/Cloud: Devel. at IFCA
  - Suited for SGE/OGE/PBS batch systems
  - Central setup needed (just one person)
- PoD: Devel. at GSI (support by ROOT)
  - Suited for many batch systems
  - Supports ssh login in remote nodes
  - Central setup needed (just one person)
- Sequential mode: No PROOF mode
  - No change on the code required
  - Ideal for debugging



How? – A summary in a picture



How? – Extra goodies

- PAF includes additional tools to
  - Create the skeleton of a selector
  - Inspect trees from the shell
    - Providing code snippets that you cut & paste in your code
  - Reset and clean the whole environment
- Includes auto completion ☺

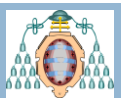
```
[iglez@fanae128 ~]$ source /opt/root/bin/thisroot.sh
[iglez@fanae128 ~]$ source /opt/PAF/PAF_setup.sh
[iglez@fanae128 ~]$ paf inspecttree -b T_Event_*umber -s Tree_TTbar_Pythia_0.root
Selecting the unique Tree: "demo/Tree"
Type: Int_t                 Variable: T_Event_RunNumber
Desc.: T_Event_RunNumber/I
Use: "Int_t T_Event_RunNumber = Get<Int_t>("T_Event_RunNumber");".

Type: Int_t                 Variable: T_Event_EventNumber
Desc.: T_Event_EventNumber/I
Use: "Int_t T_Event_EventNumber = Get<Int_t>("T_Event_EventNumber");".
```

How? – More extra goodies

- Logger
  - Configurable level of output (Debug, Warning, Error,…)
  - Stored in the output → Useful for debugging
- Same output from sequential and parallel processing
- Coherent mechanism for information passing and retrieval from project to selectors and among selectors
- External packages and libraries possible
  - Group or experiment repositories
- Works on homogeneous (optimal) and heterogeneous clusters

When? – Present and future

- **Present**
  - PAF V5.0.1 just out there
  - We will use in LHC Run-II
  - ROOT 5 and ROOT 6 supported
- **Future**:
  - Bug fixing, code cleaning and polishing, improve building process…
  - Ability to process several samples in one go
  - Further improve performance and usability: Avoid “unneeded” compilations
  - Accounting, monitoring, … → REST Service
  - Web frontend (for dissemination?) → A prototype already there
  - Integrate with other tools
  - Configuration of default parameters (ex. Batch queue name) like in .rootrc?
  - Support more complex structures (miniAOD, PAT)?

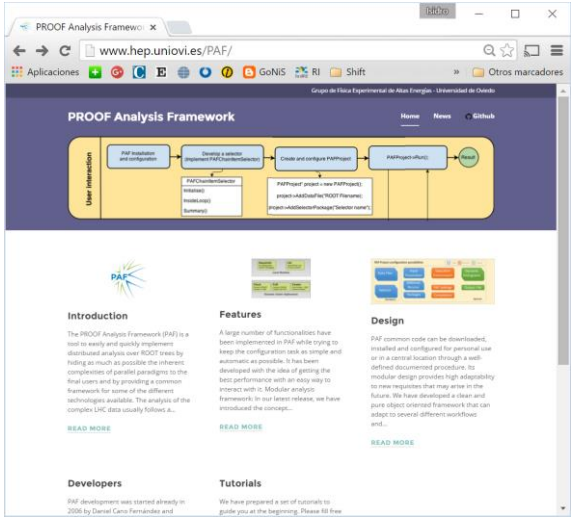


Who? – The team

- Current core team:
  - Isidro González Caballero – Project leader and initial developer
  - Javier Fernández Delgado – Current main developer
  - 2 Students joining the team soon
- Other developers
  - Enol Fernández Castillo – PROOF Cloud developer
- Past developers no more active:
  - Daniel Cano Fernández – Initial developer
  - Ana Y. Rodríguez Marrero – First redesign and PROOF Cluster
  - Alberto Cuesta – PoD integration
- Logo designer ☺ - Lara Lloret Iglesias

Thanks for your attention

- More information:
  - Main **PAF** Page (documentation, tutorials…)



PAF http://www.hep.uniovi.es/PAF

  - Github repository:

https://github.com/PROOF-Analysis-Framework

- Contact:
  - Isidro.Gonzalez.Caballero@cern.ch