PROOF tutorial
Introduction to PROOF

Gerardo Ganis, CERN, PH-SFT
gerardo.ganis@cern.ch

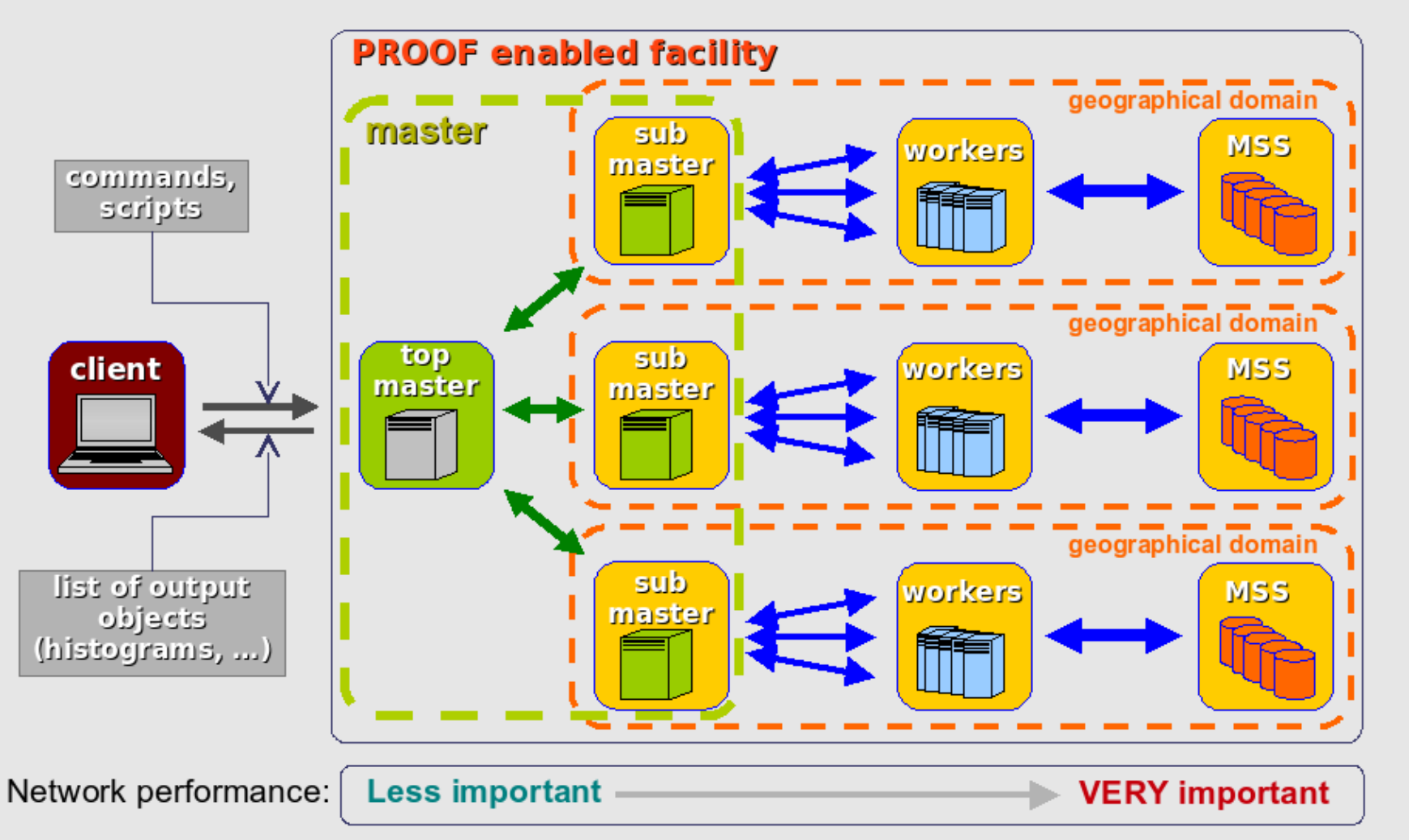
PROOF goals

- Primary target of PROOF was to speed-up TTree processing by going parallel
  - HEP events independent → can split
- Multiple machines → multiply disk I/O
- Multiple machines → multi-process parallelism
- Multi-thread would anyhow been difficult in ROOT
  - Too many part not really thread-safe, although things may change in the future

PROOF – Parallel ROOT Facility

- Parallel coordination of distributed ROOT sessions
  - Scalability: small serial overhead
  - Transparent: extension of the local shell
- Multi-Process Parallelism
  - Easy adaptation to broad range of setups
  - Less requirements on user code
- Process data where they are, if possible
  - Minimize data transfers
- Event-level dynamic load balancing via a pull architecture
  - Minimize wasted cycles
- Real-time feedback
  - Output snapshot sent back at tunable frequency
- Automatic merging of results

PROOF architecture



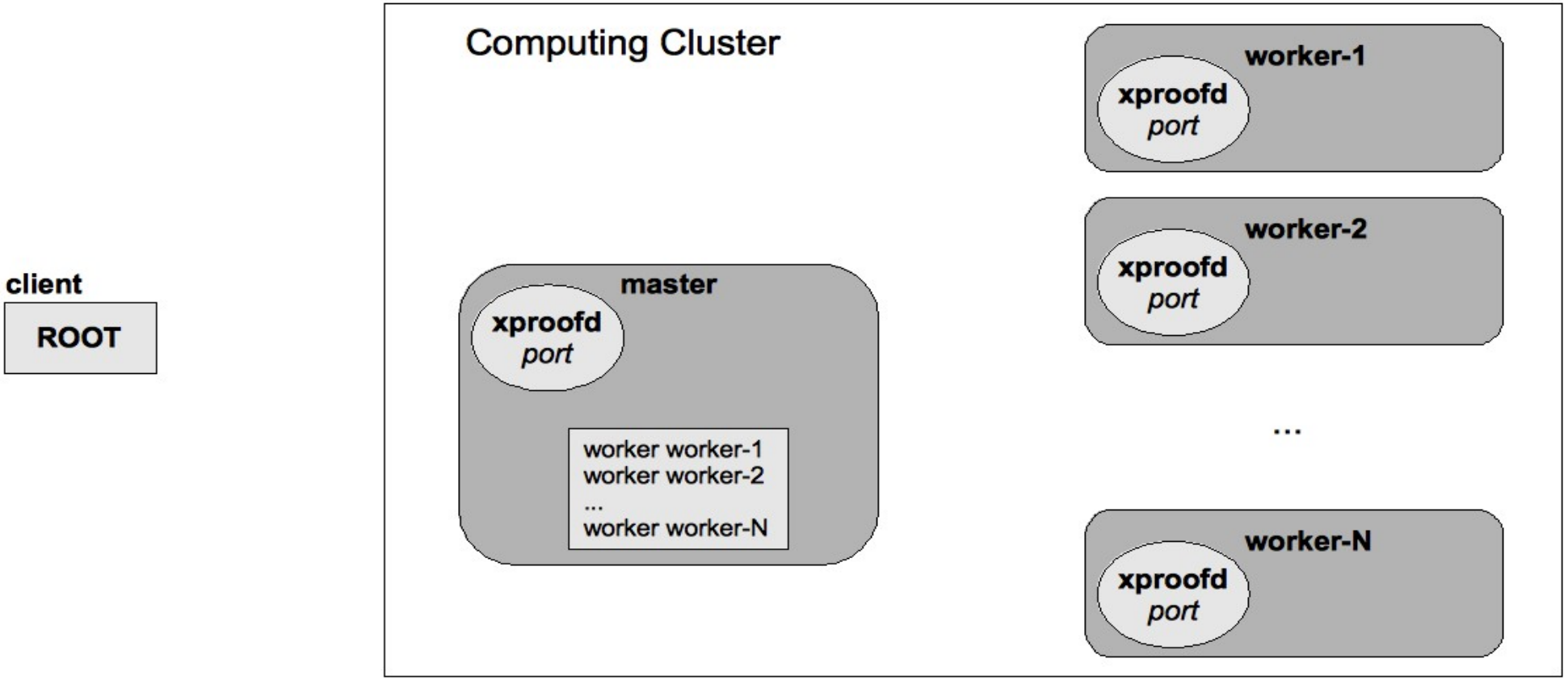
PROOF architecture

- Three-tier Client-Master-Worker architecture
- Flexible Master tier
  - Adapt to heterogeneous configurations
  - Distribute load of reduction (merging) phase

Applications running on the master and worker nodes are ROOT applications similar to the one running on the local machine: the only difference is that they take input from a network socket instead of the keyboard

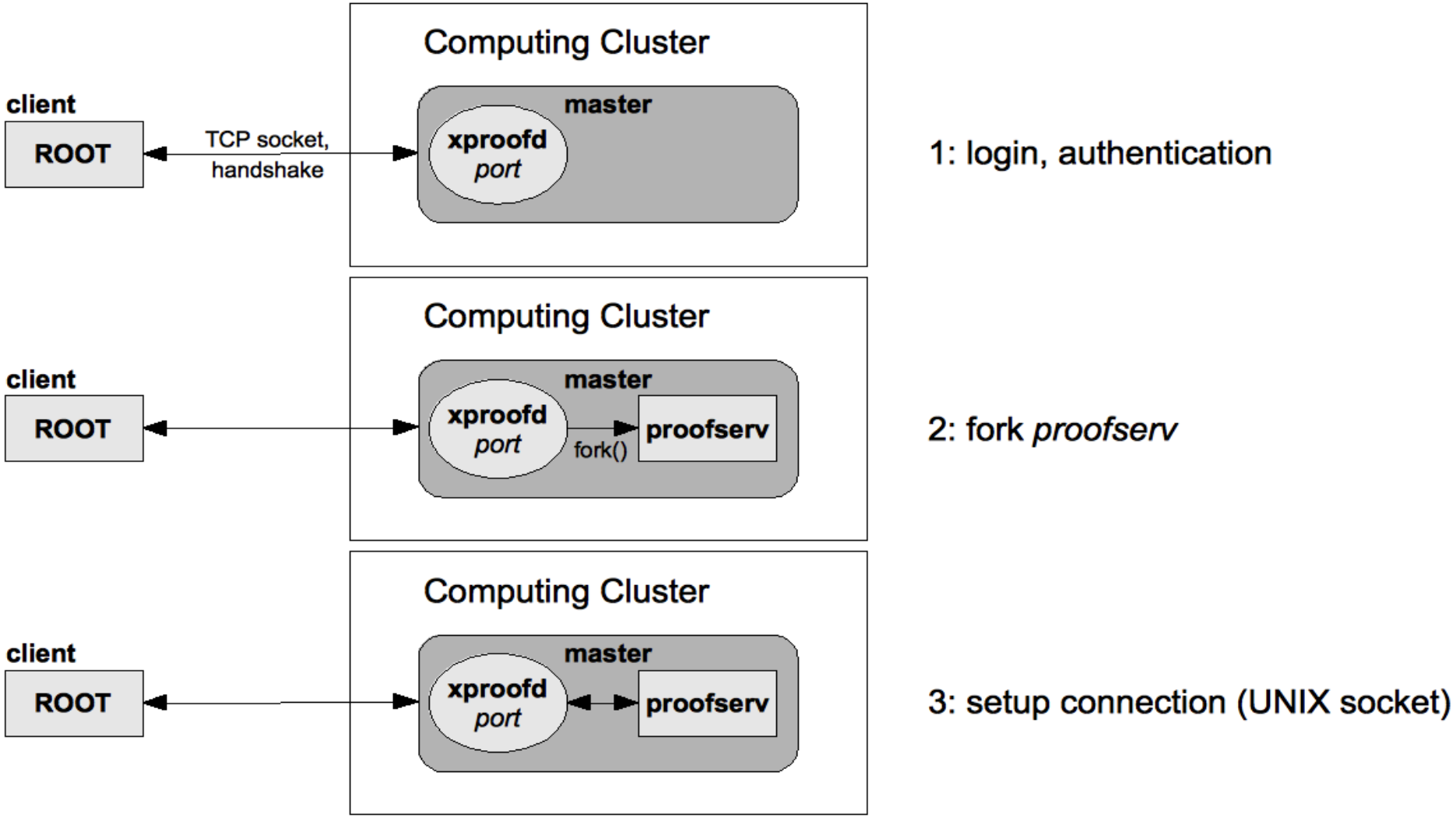
PROOF as a network service

- PROOF is a network service and has its own daemon
- An idle cluster would look like this:



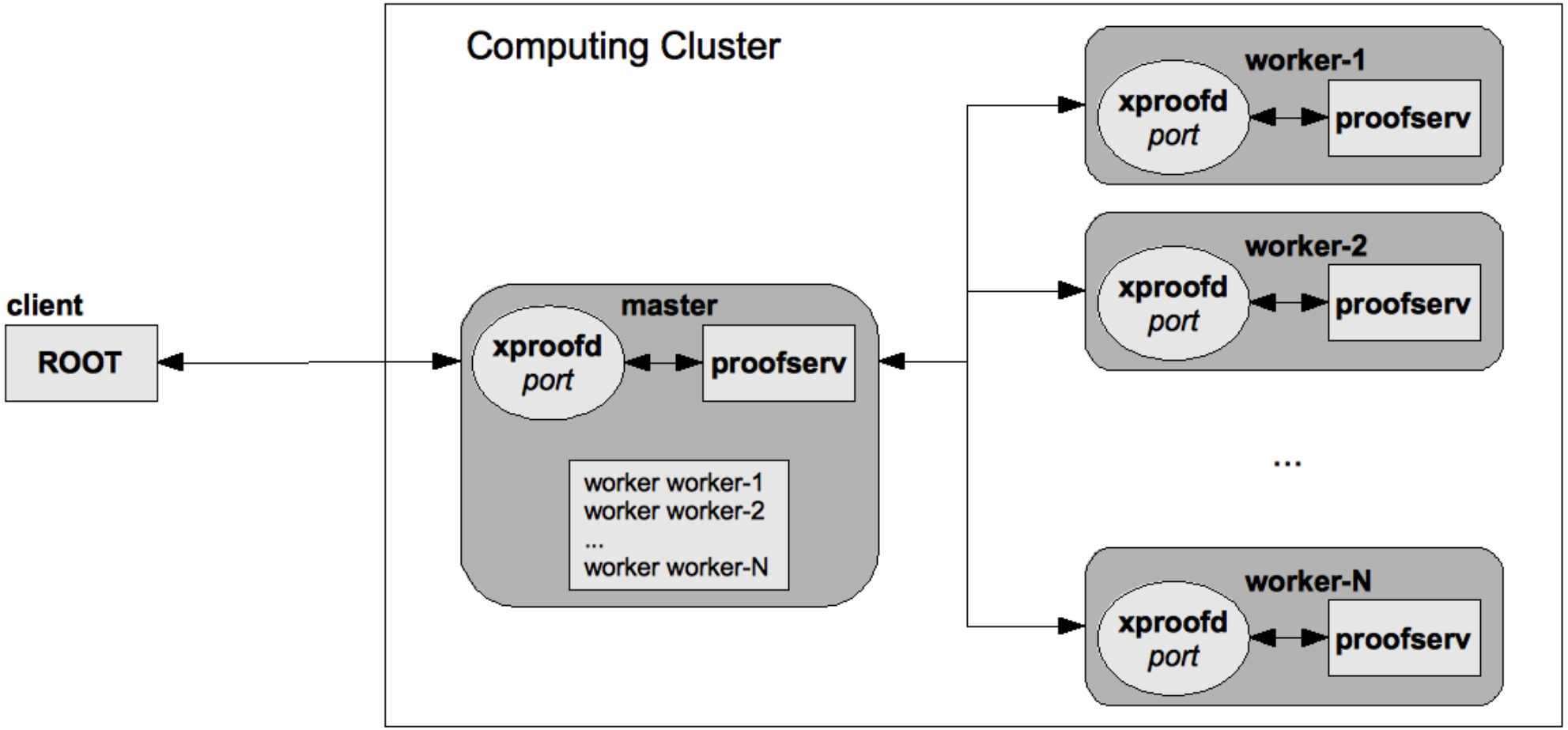
Setting up a PROOF connection

- Setting up a PROOF connection involves a few steps



A PROOF session

- The proofserv are ROOT applications net-connected



What can you expect from your resources

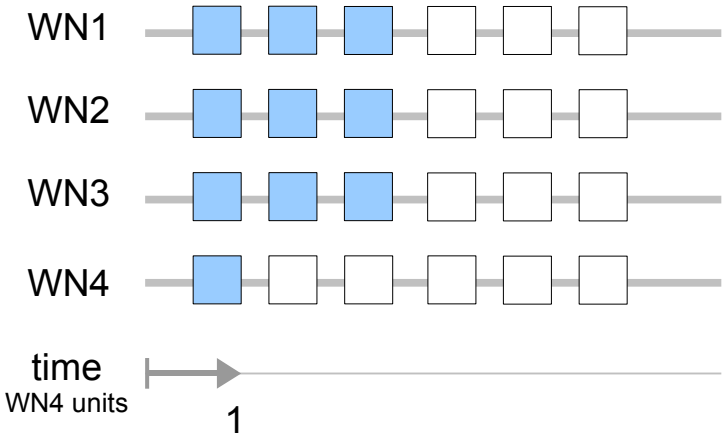
- C cores, U users
- N files to process
- Processing time (T_{proc})

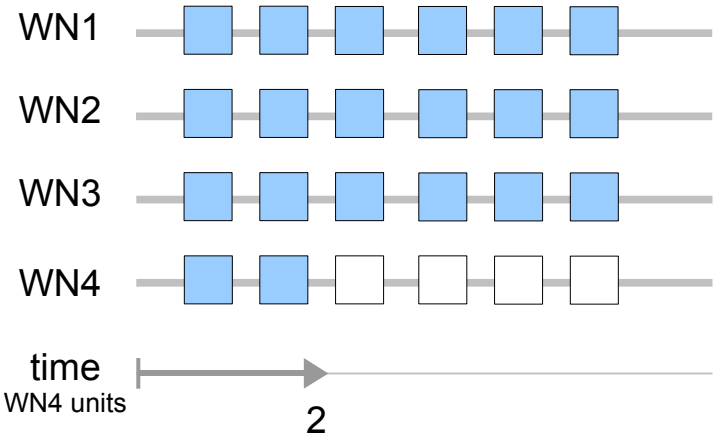
$$T_{proc} = T_{init} + \frac{U \cdot N}{C} \cdot T_{file} + T_{term}$$

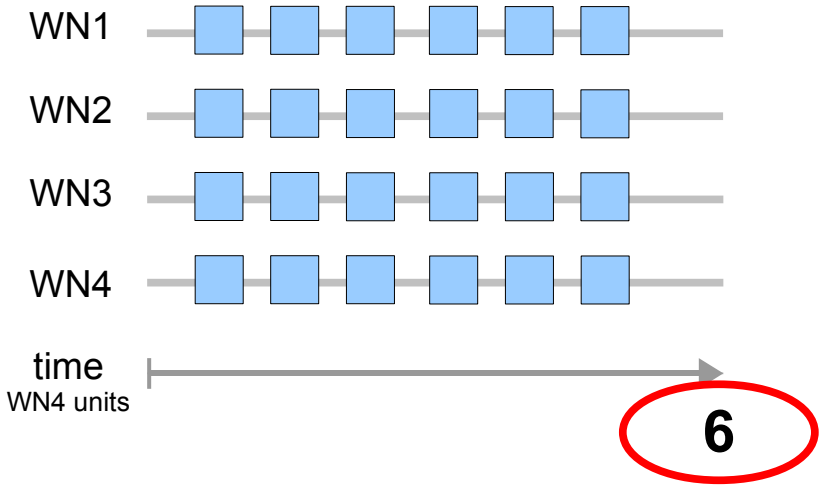
- Assuming
  - Optimal splitting (~ C/U concurrent jobs at a time)
  - Processing time per file (T_{file}) independent of the status of resources
- Large tails in the T_{file} distribution ...

Static … (e.g. multi-process on Grid)

Example: 24 files on 4 worker nodes, one under-performing

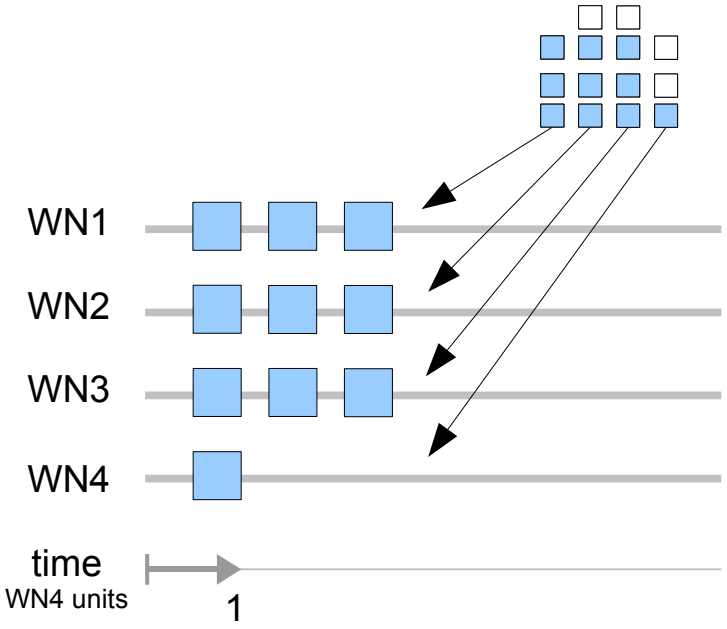


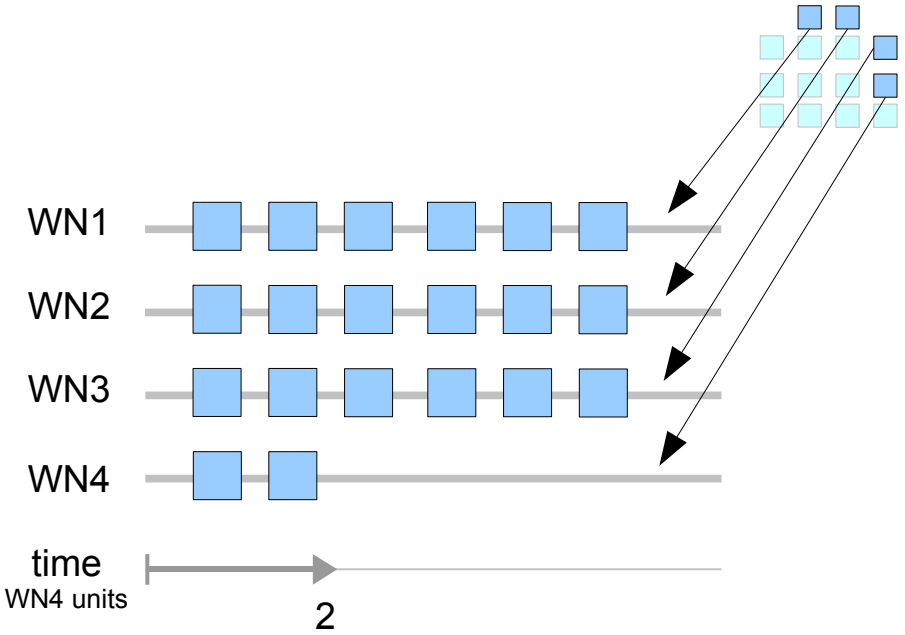


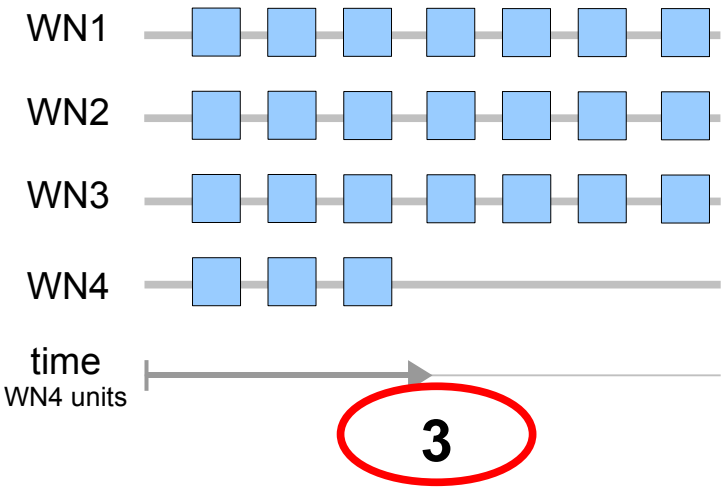


The slowest worker node sets the processing time

... vs Dynamic (e.g. Proof)

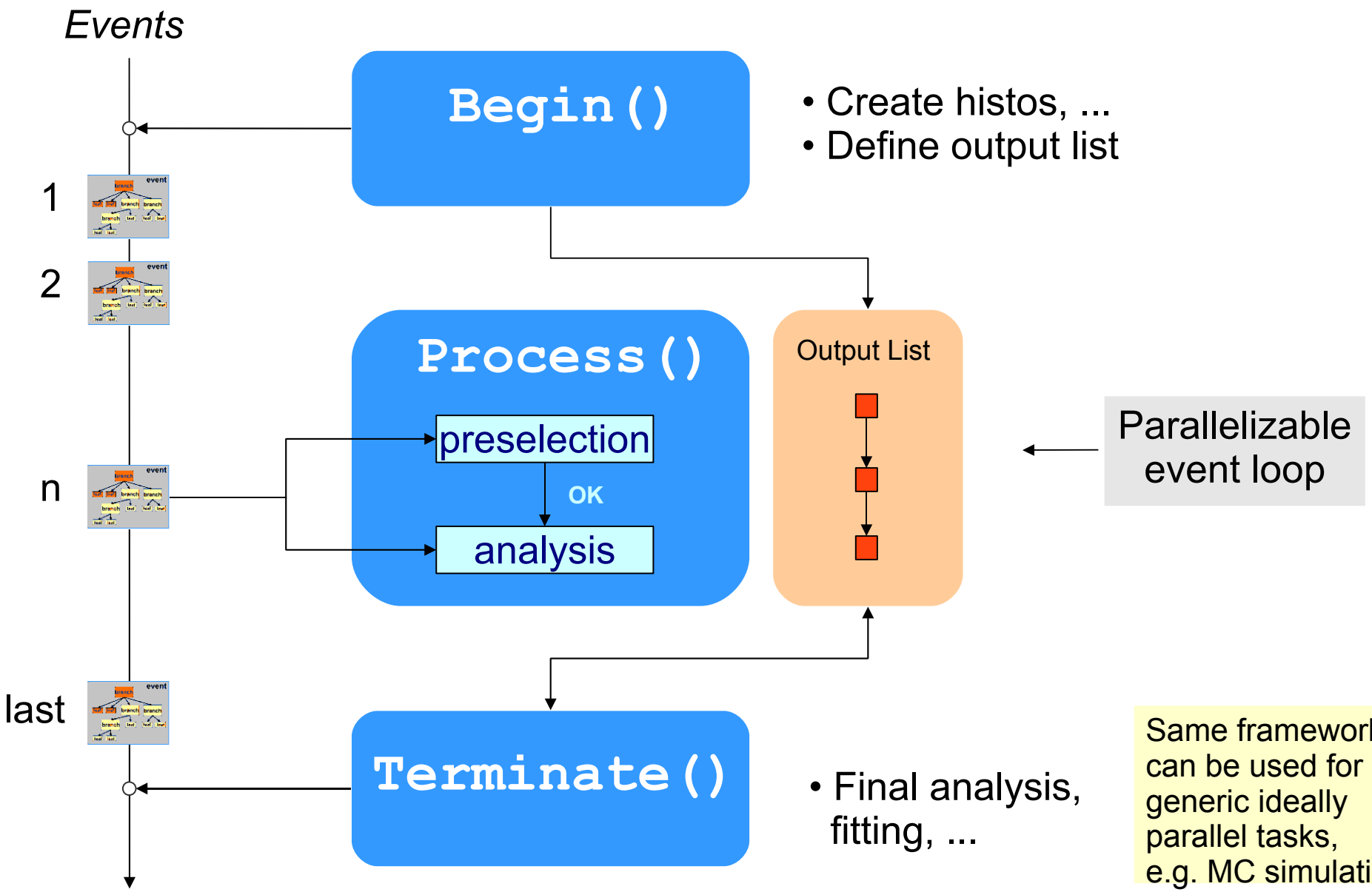




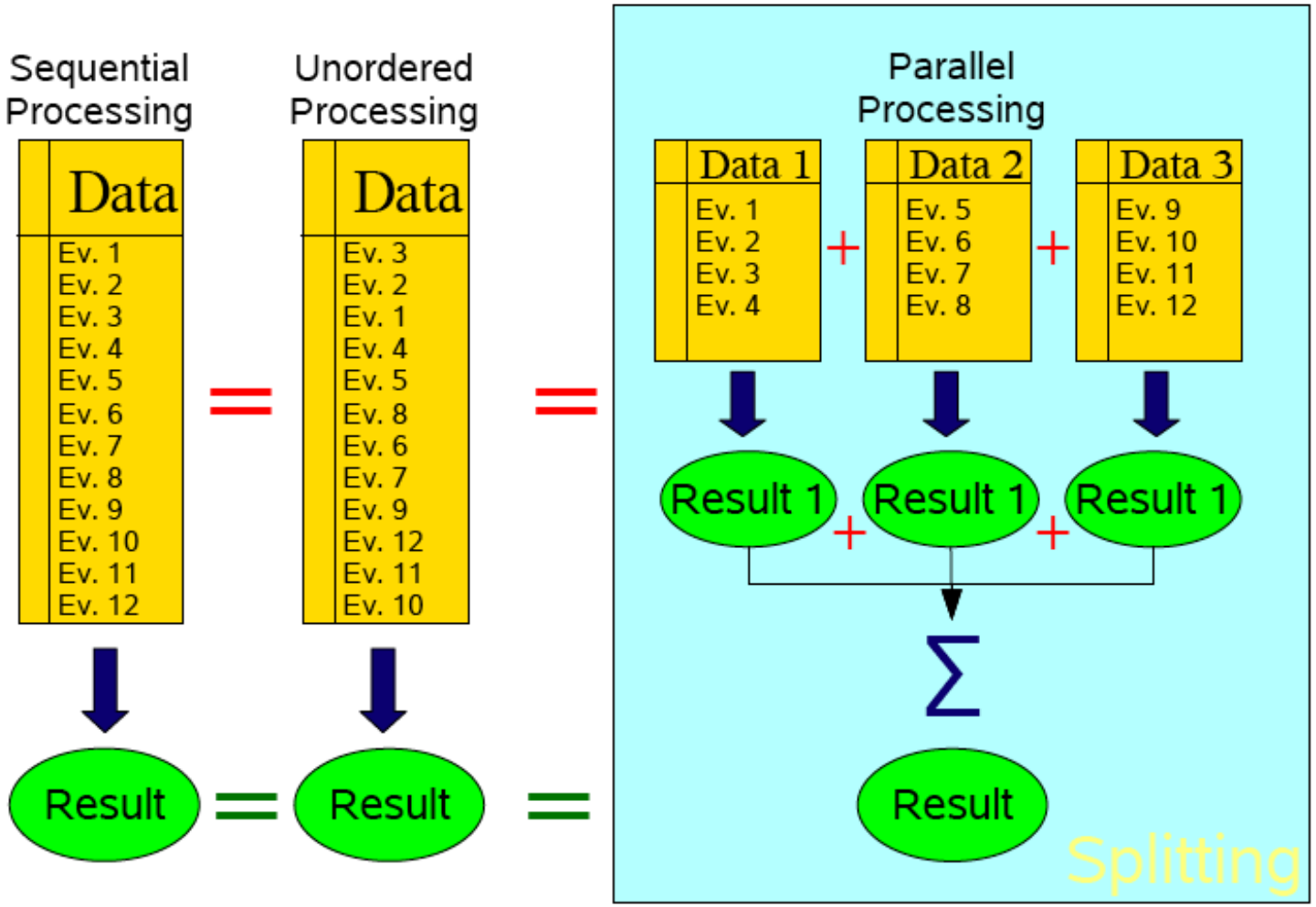


The slowest worker node gets less work to do: the processing time is less affected by its under performance

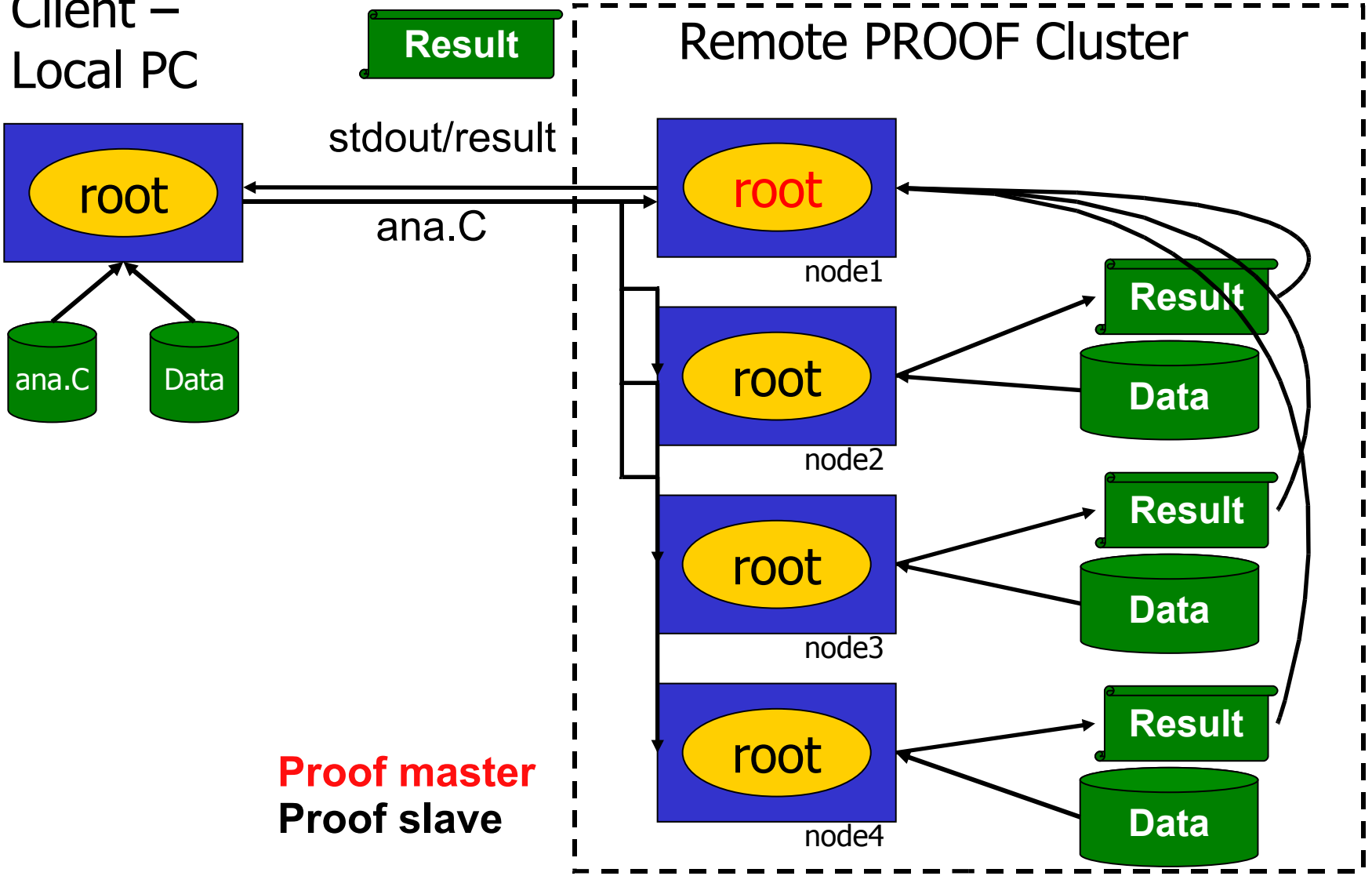
Event level parallelism: TSelector framework



Trivial / Ideal Parallelism



Workflow scheme



Merging output objects

- PROOF automatically merges objects having a method

  Long64_t Merge(TCollection *objectsToBeMerged)

  which is supposed to merge into itself the list of objects passed in the argument
- All standard output objects (TH1, TH2, TH3, TTree, ...) provide a Merge method
- For user-defined objects the method has to be provided if merging is wished
  - If the Merge method is not found, the N objects received by the workers are just sent back to the client as they are

Documentation

- PROOF documentation not (yet) complete
- The TWiki pages are the most updated source currently

  http://root.cern.ch/drupal/content/proof

- The standard ROOT class description service is also helpful

  http://root.cern.ch/root/html/ClassIndex.html

- The PROOF section on the ROOT forum allows direct interaction with the developers and other users

  http://root.cern.ch/phpBB2/index.php