ROOT and Federated Data Stores

What Features We Would Like

Fons Rademakers
CERN

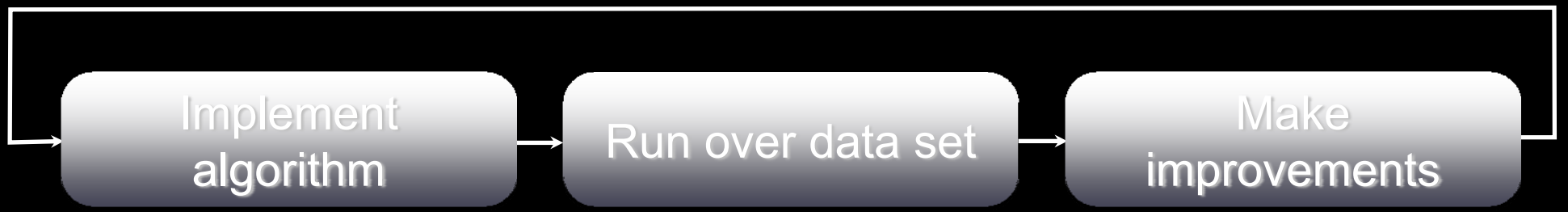
CC-IN2P3, 21-22 Nov, 2011, Lyon, France.

Outline

- Optimizing WAN file access
- Local caching
- To Merge or not to merge
- Conclusions

HEP Data Analysis

- Typical HEP analysis needs a continuous algorithm refinement cycle

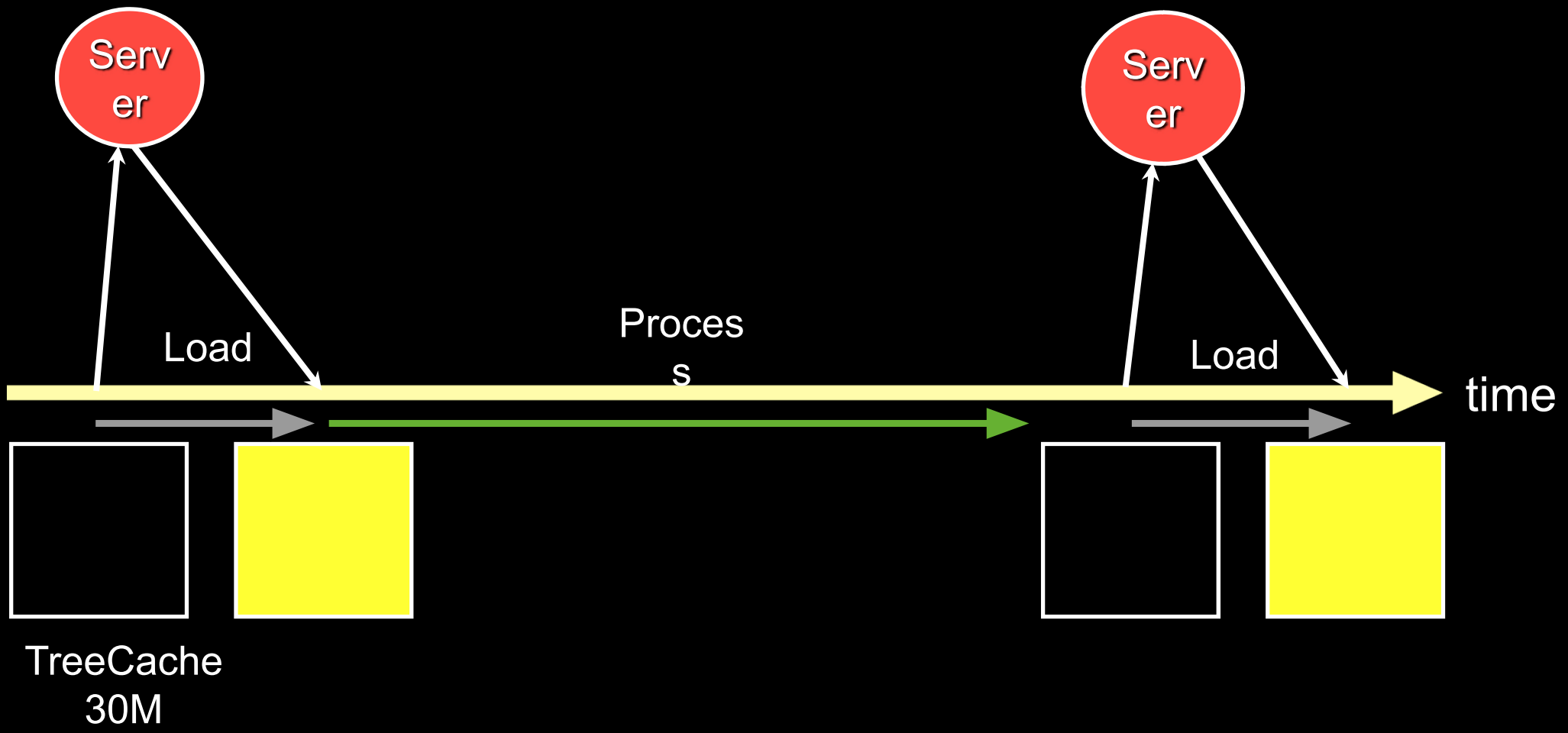


- Ranging from I/O bound to CPU bound
- The faster the network the higher the I/O rate
- The lower the network latency the higher the I/O rate
- The more disks the higher the I/O rate
- The more RAM the more can be cached
- The more CPUs the faster the processing

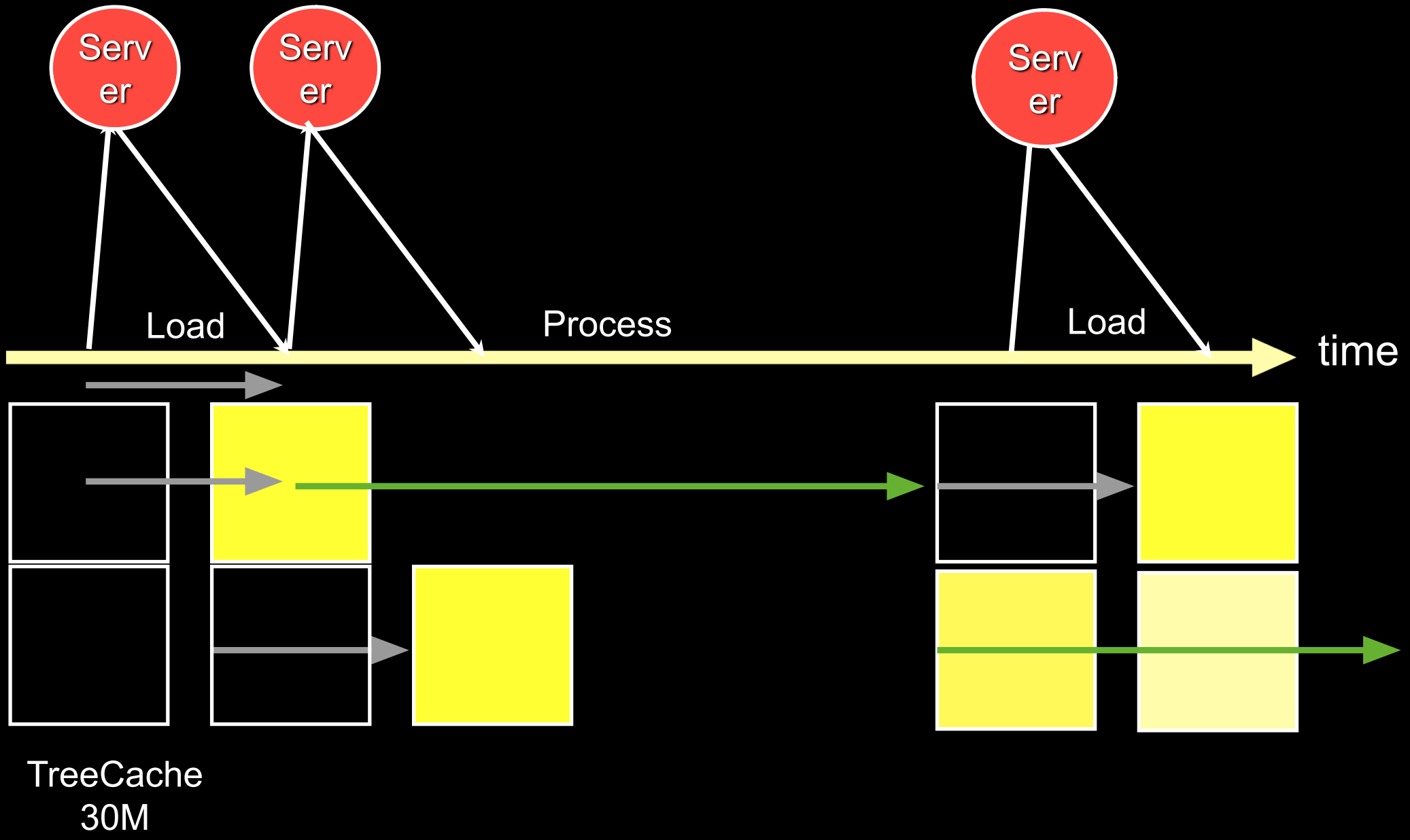
ROOT Optimizations for WAN

- Load phase (where data is fetched from an SE into the TreeCache) is
  - Short for LAN transfers
  - Significant for WAN transfers (latency, bandwidth)
- Gain in WAN by asynchronous (double buffering) transfer technique
  - Independent of access protocol (xrootd, httpd, etc)
- In addition local file caching
- And site proxy server
- Implemented in v5.30 by Elvin Alin Sindrilaru (fellow IT-

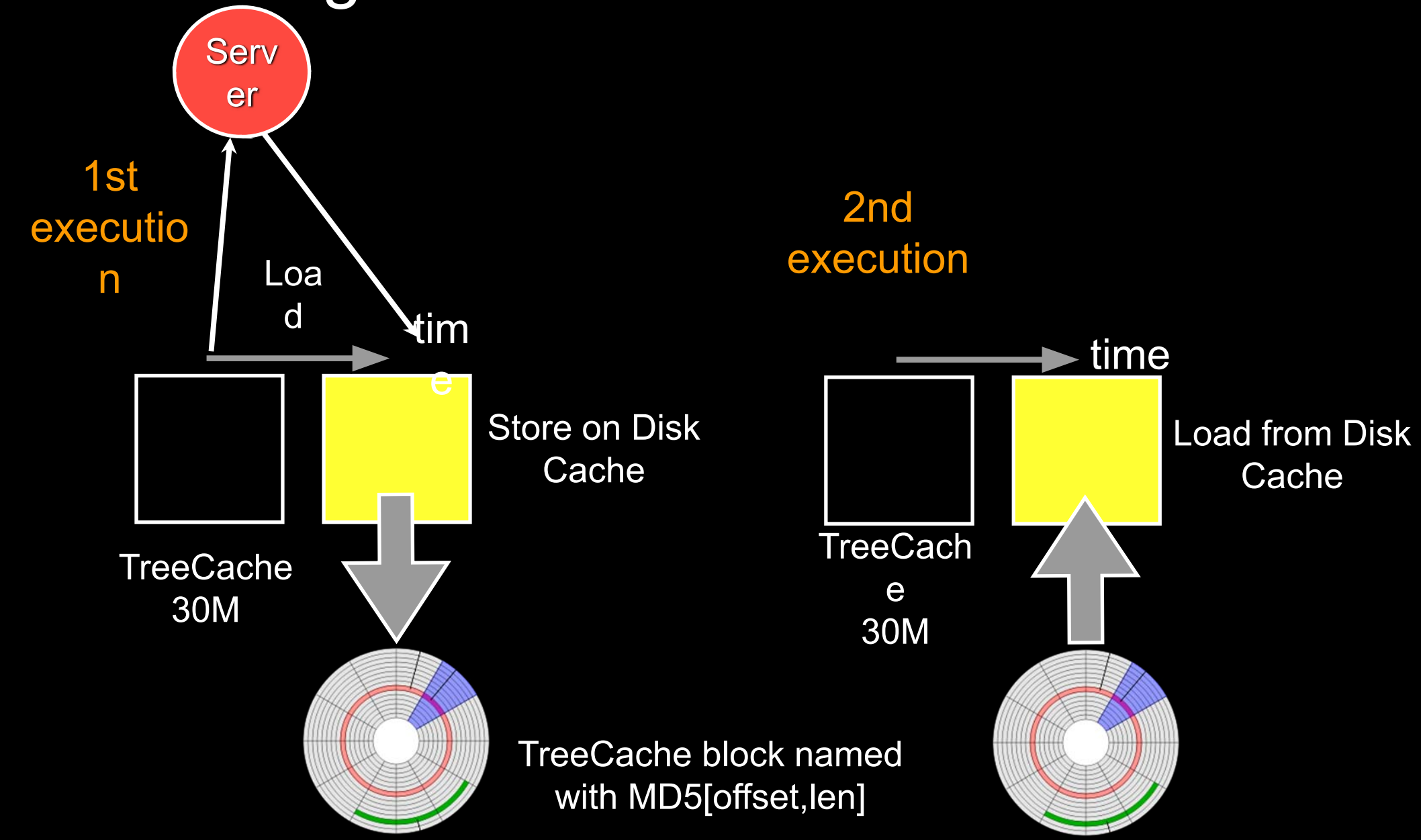
Tree Processing is Synchronous



Asynchronous Pre-Fetching

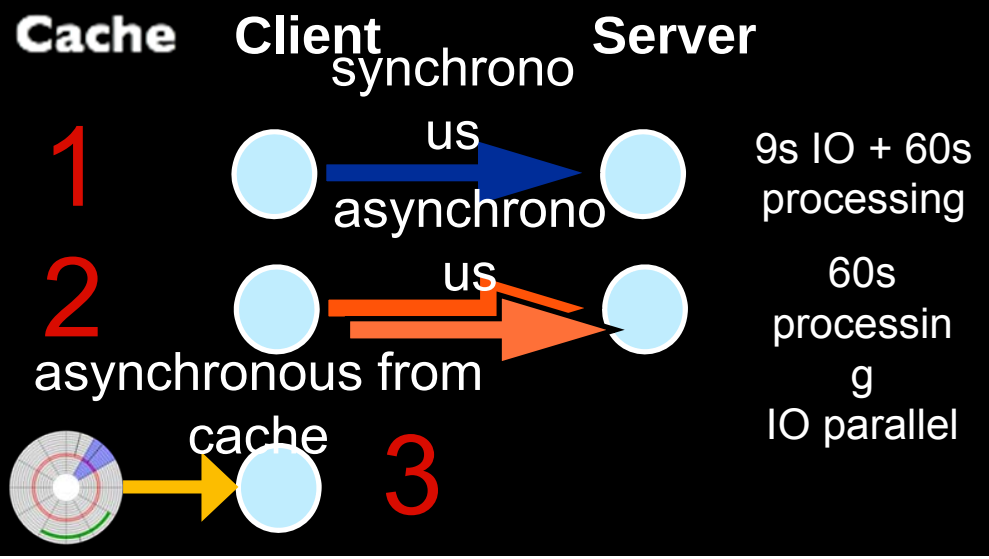


Caching of TreeCache Blocks



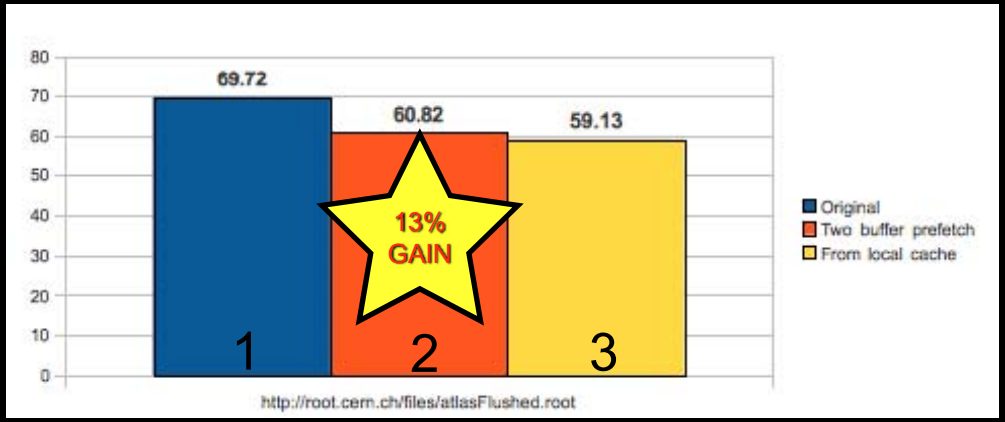
Test Results

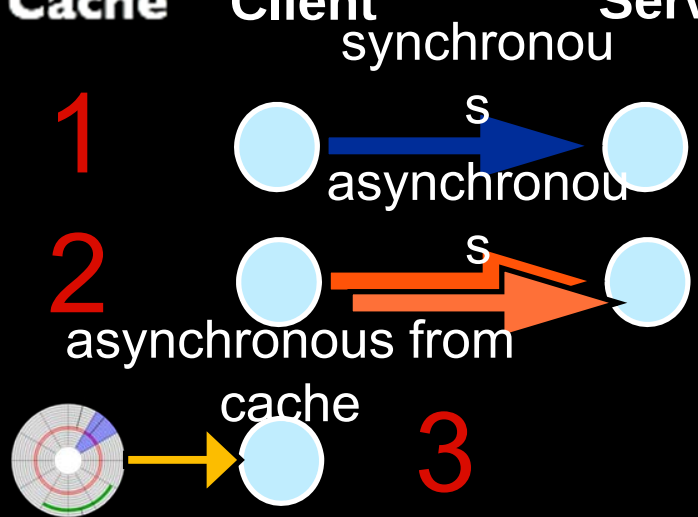
(reading an 1 GB ATLAS AOD file over http)



LAN

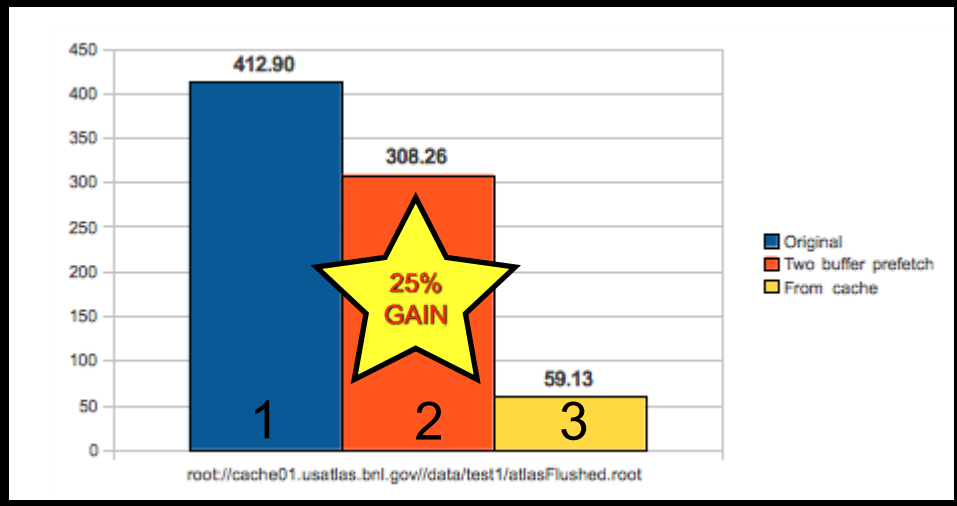
Server@CERN





WAN

Server@BNL



* although the IO is asynchronous we are limited by the available bandwidth

Pre-fetching and Caching Summary

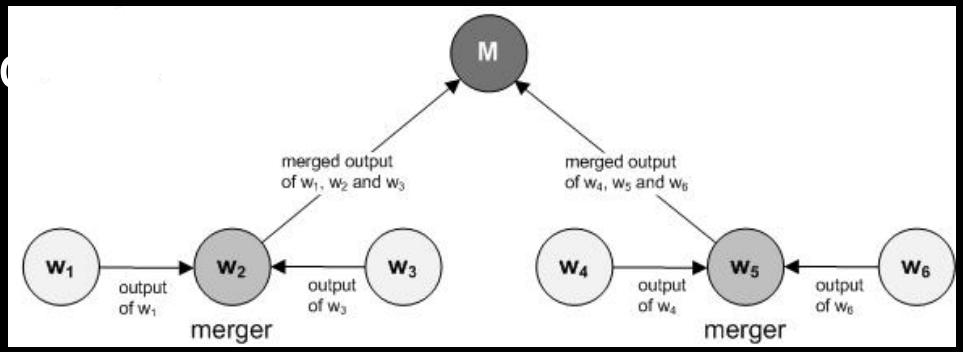
- Asynchronous pre-fetching has been demonstrated as an efficient way to improve the cpu/RT efficiency of analysis applications
  - Allows to use every synchronous protocol in asynchronous mode
  - Allows to proxy caching of TreeCache blocks on any ROOT supported file storage
    - TreeCache transforms sparse/random access into sequential local access
  - Integrated in ROOT v5.30, activated using rootrc flag:
    - TFile.AsyncPrefetching: yes

To Merge or not to Merge

- Output of Grid jobs and PROOF workers are naturally nicely split
- Ideal situation for input to next distributed analysis
- Why merge
  - Combine objects, like histograms (but not needed for trees)
  - Export
  - Simplify file management
- Why not to merge
  - Output of Grid jobs and PROOF workers are naturally nicely split
  - Good situation for input to next distributed analysis
  - Lazy merging (merge histogram only when being accessed)

Merging Large Outputs

- Large or non-optimized outputs may have dramatic effects
  - Memory explosion
  - Destroy the parallelization gains during merging
    - Many 3D histograms or 10000's of 1D histograms
- PROOF solution
  - Save outputs on files on worker nodes and
    - Automatically run a merging application (TFileMerger or your own)
    - Automatically create a dataset for further processing
  - Parallel merge: fastest w



Theoretical speedup: $\sqrt{N_{wrk}}$
Available from ROOT 5.26

Faster Parallel Merge

- Current bottleneck in merging large number of files:
  - Write to local disk on all slaves
  - Server reads remote files on workers and writes one single file
- This completely serializes the operation and wastes time and resources

Faster Parallel Merge Solution

- Completely avoid the writing on disk of the intermediary files
- Start uploading and merging of the resulting data as soon as possible
- Using a special TFile implementation
  - Write only to memory
  - Whenever the TTree is AutoFlushed, upload the data to the server
- Server will then:
  - Receive in parallel the data from the slaves and stages it in memory
  - Use the ‘fast merging’ technique to merge directly from the in-memory file to the final file without uncompressing or un-streaming the data stored in a TTree

Faster Parallel Merge Advantages

- Minimal number of I/O operations:
  - Slaves write data only to memory
  - Slaves upload data once via TSocket
  - Server writes data only once to server disk
- Merge starts as soon as a minimal (30Mb) amount of data has been accumulated by one slave
- I/O concurrent with processing
- Transparent to user code on slave
  - The special TFile, after creation, is used as any other TFile
  - Objects in the TFile are automatically Reset and Merged, avoiding data duplication (however only classes with a Merge and a ResetAfterMerge functions are supported)

Not Merging - File Set Support

- The many produced files belong to one logical single file set
- Currently several experiment and ROOT specific solutions
  - TFileCollection
  - PQ2
  - Independent of storage system
    - Storage system does not know which files belong to a file set
- Would like to see file set support in xrootd
  - Copying a file set will copy all its members
  - Deleting a file set will delete all its members
  - Migrating a file set will migrate all its members
  - Exporting a file set will export all its member

ROOT Support for Exported File Sets

- On export of a file set Xrootd just concatenates the member files
- ROOT support for concatenated ROOT files
  - TMultiFile
  - Trees in a TMultiFile are just TChain'ed
  - Objects can be merged on-the-fly
  - TMultiFile can be passed to TFileMerger for merging of all objects

Conclusions

- ROOT is mostly “on top” of Federated Data Stores
- We try to make access to remote files as efficient as possible
- File set support on the data store level would be welcome