Exploiting concurrency to speed-up ROOT data analysis

caching, PROOF and multi-threading

G. Ganis, CERN

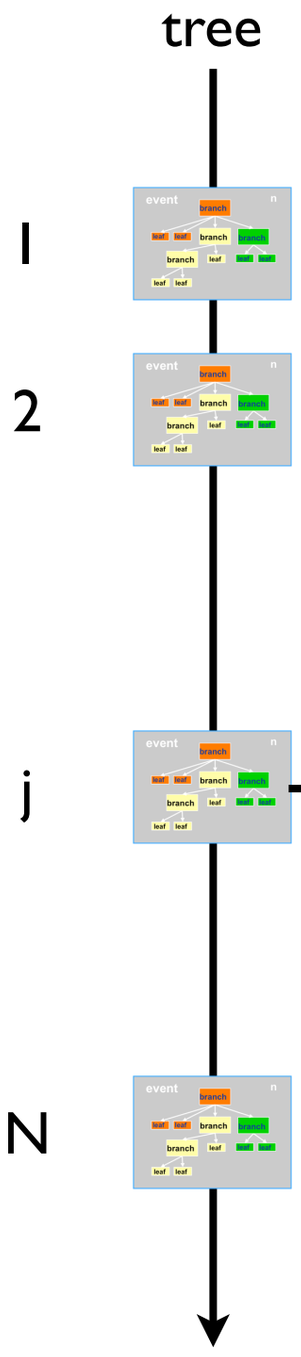
Hardware context

- Large increase in processing power
  - Multi-/Many-cores, easy access to GPUs
  - CPU is (probably) the cheapest component
  - CPU-bound is the ideal case
    - Real time ≅ CPU time / N_{proc}

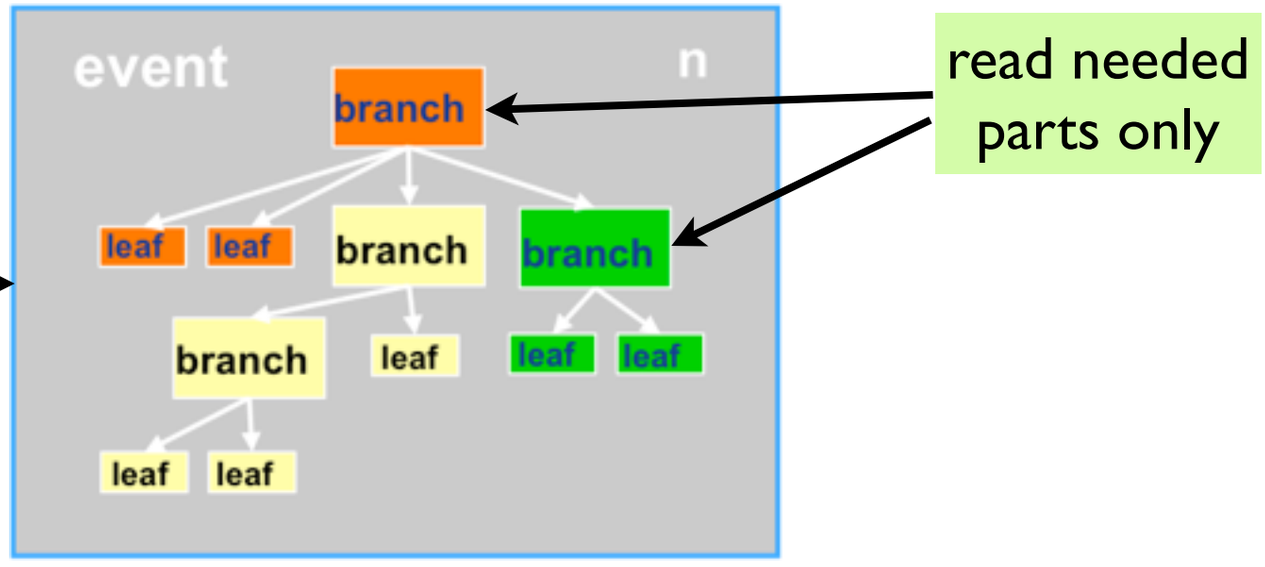
Data Analysis needs I/O

- End-User analysis profits from CPU (simulations, fits, ...)
- But in HEP data analysis I/O is important
  - 10 TB in ~1 day implies ~200 MB/s
- How does the currently affordable I/O hardware technology compare with the available CPU?

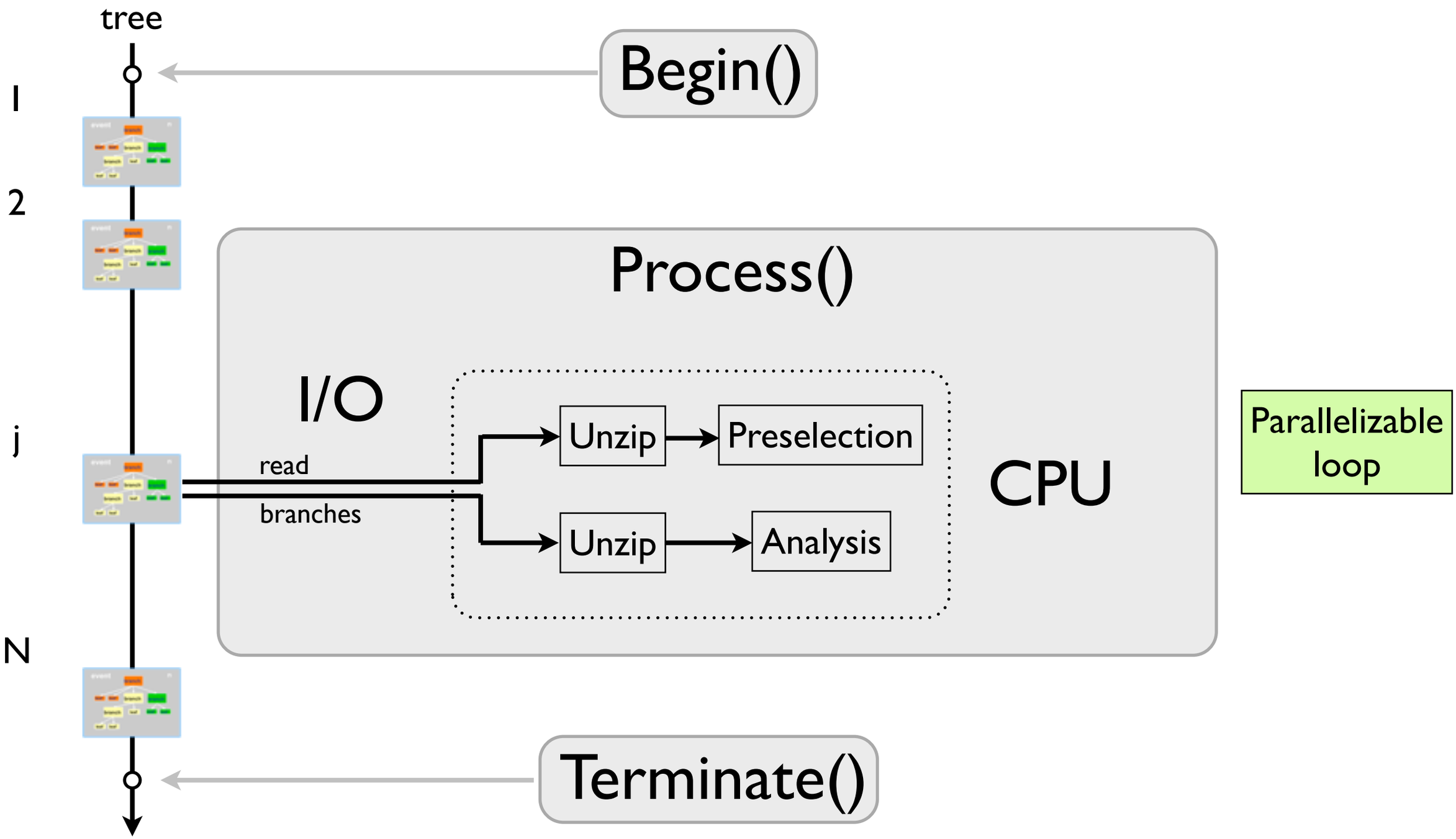
ROOT Trees



- Optimized for fast and random access to any part of an entry



Processing trees



Rate scaling (schematically)

- Global rate R for N process, $N_{I/O}$ devices

$$N \cdot R \,/\, [1 + (N\text{-}1) \cdot R \,/\, (N_{I/O} \cdot R_{I/O})]$$

where

$$1 \,/\, R = 1 \,/\, R_{I/O} + 1 \,/\, R_{dec} + 1 \,/\, R_{proc}$$

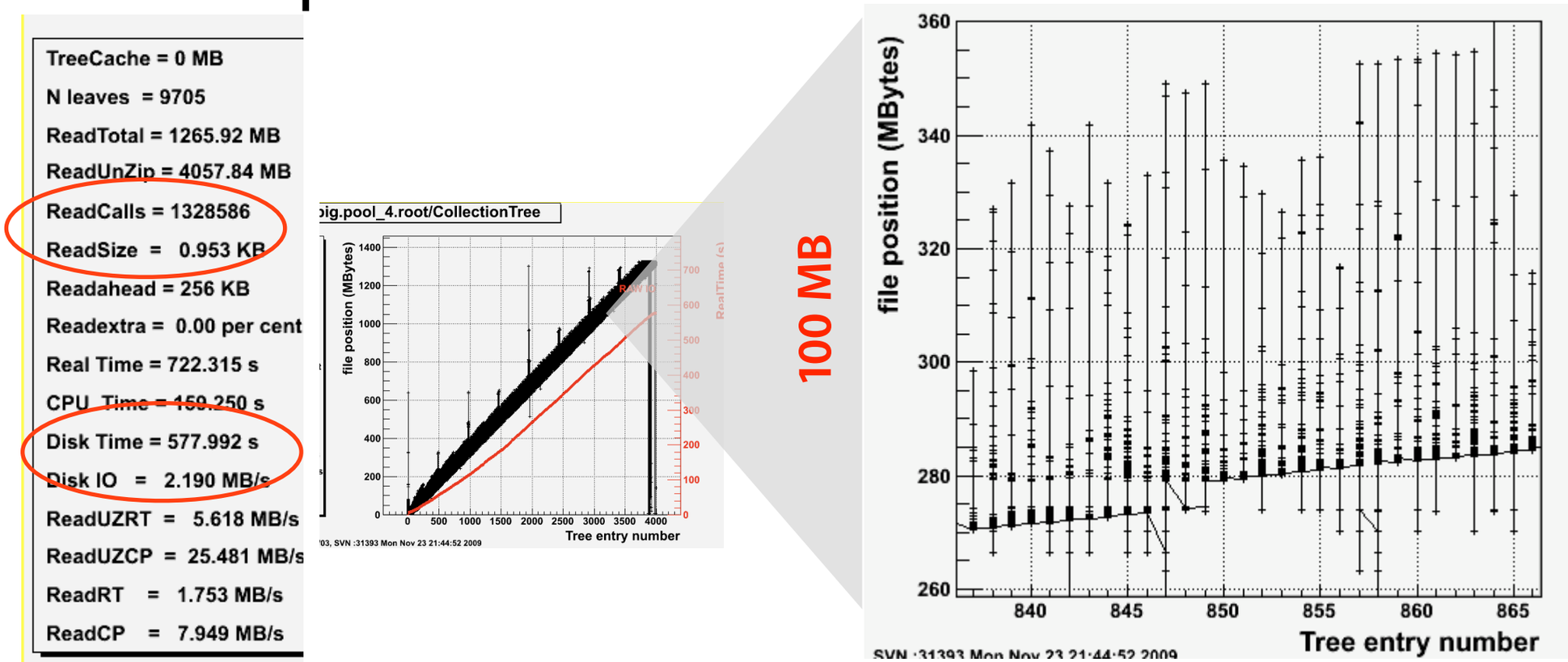
- Saturation term small for large I/O bandwidth or small I/O weight ($R << R_{I/O}$)
- $R \rightarrow N_{I/O} \cdot R_{I/O}$ for large N

Decompression

- Decompression requires a fair amount of CPU (~20 MB/s per core)
- Compression factor ~ 3 : cost of network still justifies it

File Layout Issues

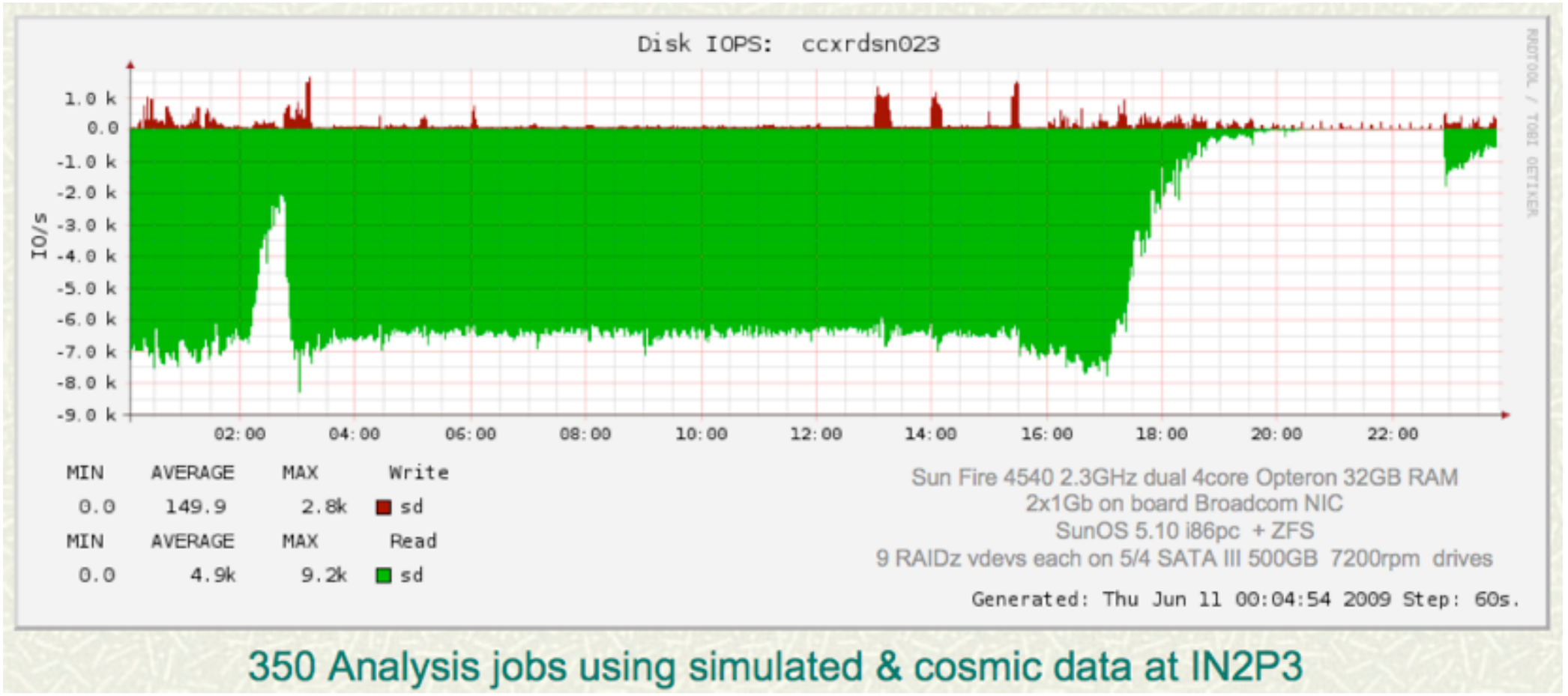
- Access patterns like this



Can kill performances: very inefficient I/O if local (2 MB/s); unsustainable rate of requests on remote

Server side

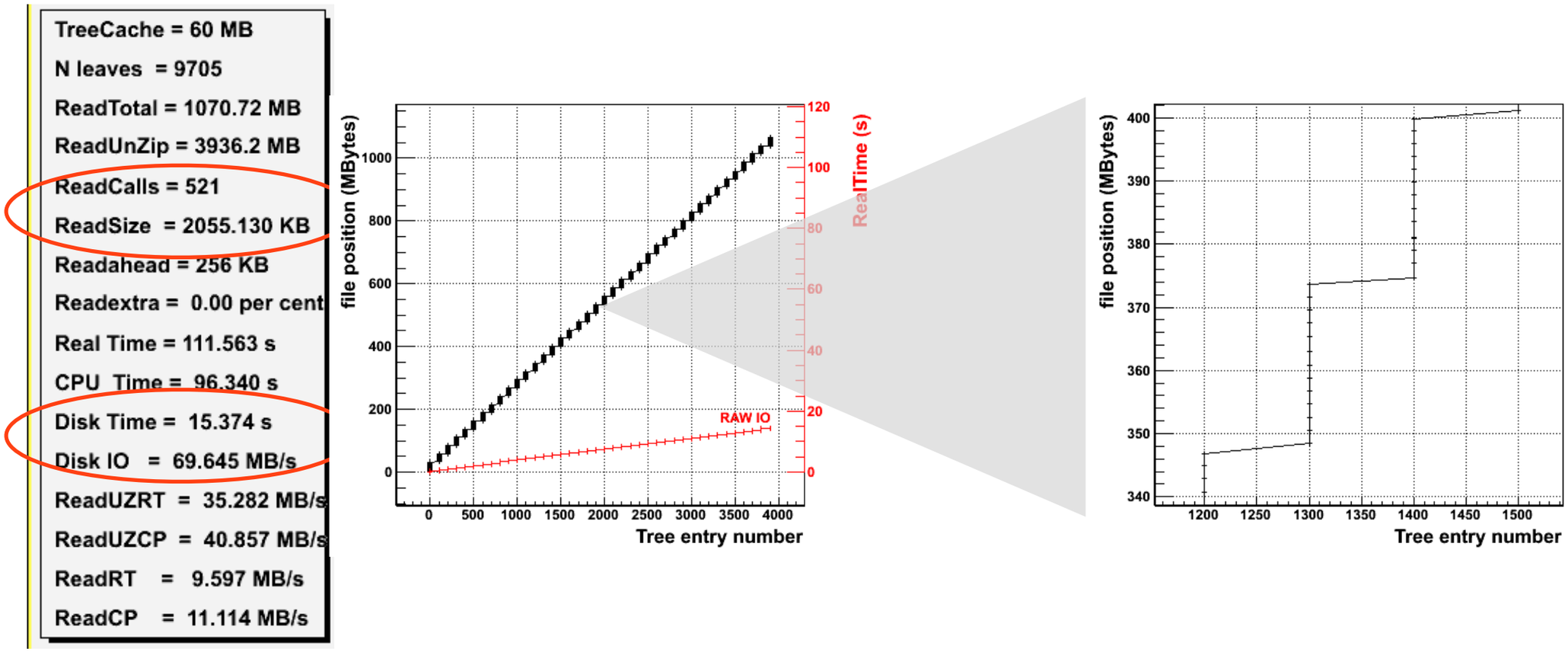
A. Hanushevsky



- Average 5000 I/O requests per second

Improved layout

- Like this is better



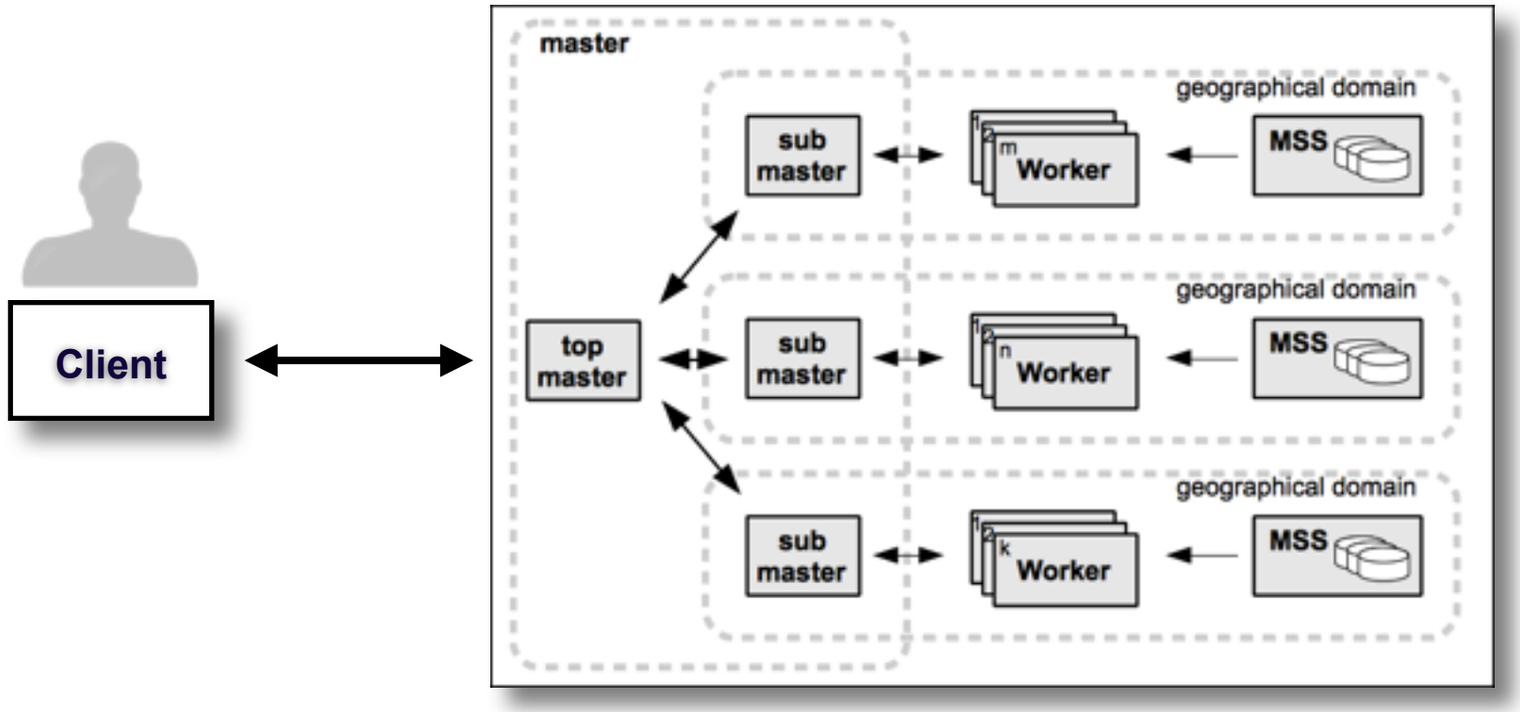
- Better device utilization (70 MB/s), reduce number of access to the device

Some consequences

- In ROOT the file layout is automatically optimized with ad hoc flushing techniques
- Close-by chunks are collapsed in a slightly bigger buffer read at once
- Server side (Xroot) started investigating the usage of SSD as additional layer to face these issues

PROOF: multi-process parallelism with ROOT

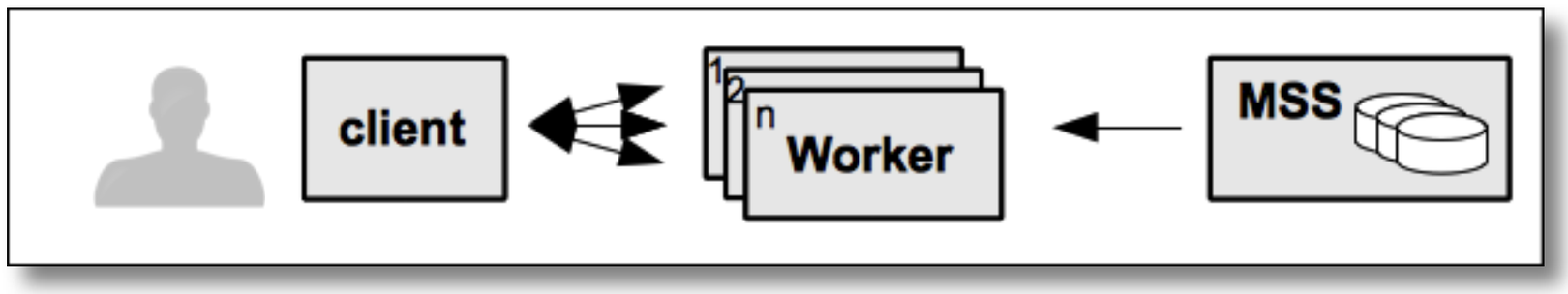
- Multi-Tier architecture



- Dynamic load-balancing (pull architecture)
- Developed for distributed data access exploiting data locality

On Multi-Cores: PROOF-Lite

- Version optimized for multi-cores



- 0-config setup (no daemons)
- Compatible with full PROOF

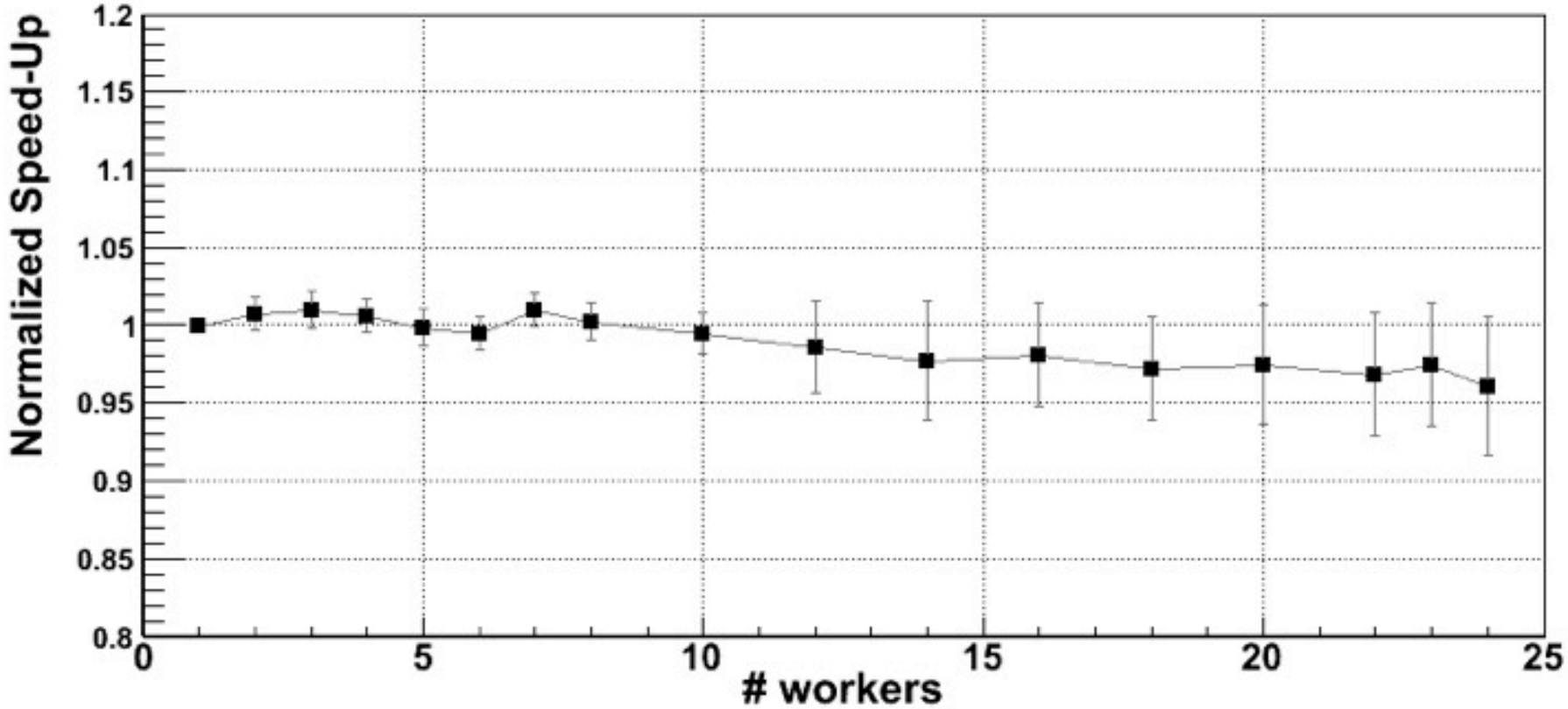
I/O devices

- Local
  - Hard Drives (40÷50 MB/s)
  - Solid State Disks (120 MB/s)
  - PCI Express SSD (ioDrive) (>700 MB/s)
  - RAM (> GB/s)
- Remote access via network:
  - {1 GB/s, 10 GB/s} NIC (100 MB/s, 700 MB/s)

PROOF-Lite test machine

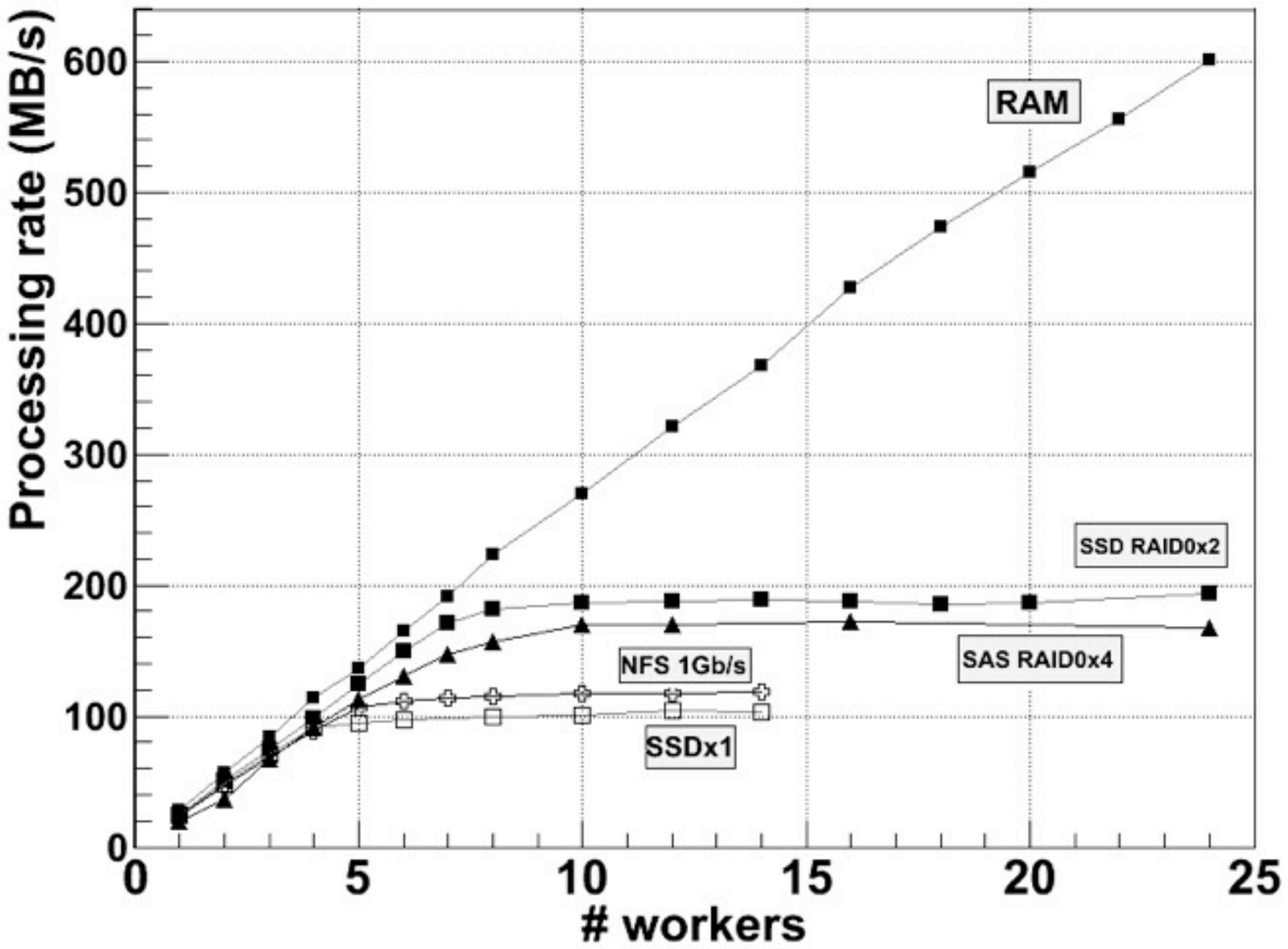
- 24 core HP DL580, 48 GB RAM
- I/O devices tested
  - 4 SAS disks 74 GB 10k RPMs, RAID0
  - 2 SSD disks 160 GB Intel X25-M, RAID0
  - 1 SSD disk 160 GB Intel X25-M
  - NFS via 1 GB/s NIC

PROOF-Lite and CPU bounded tasks



- Close to ideal
- Deviations due to the result merging which is serial

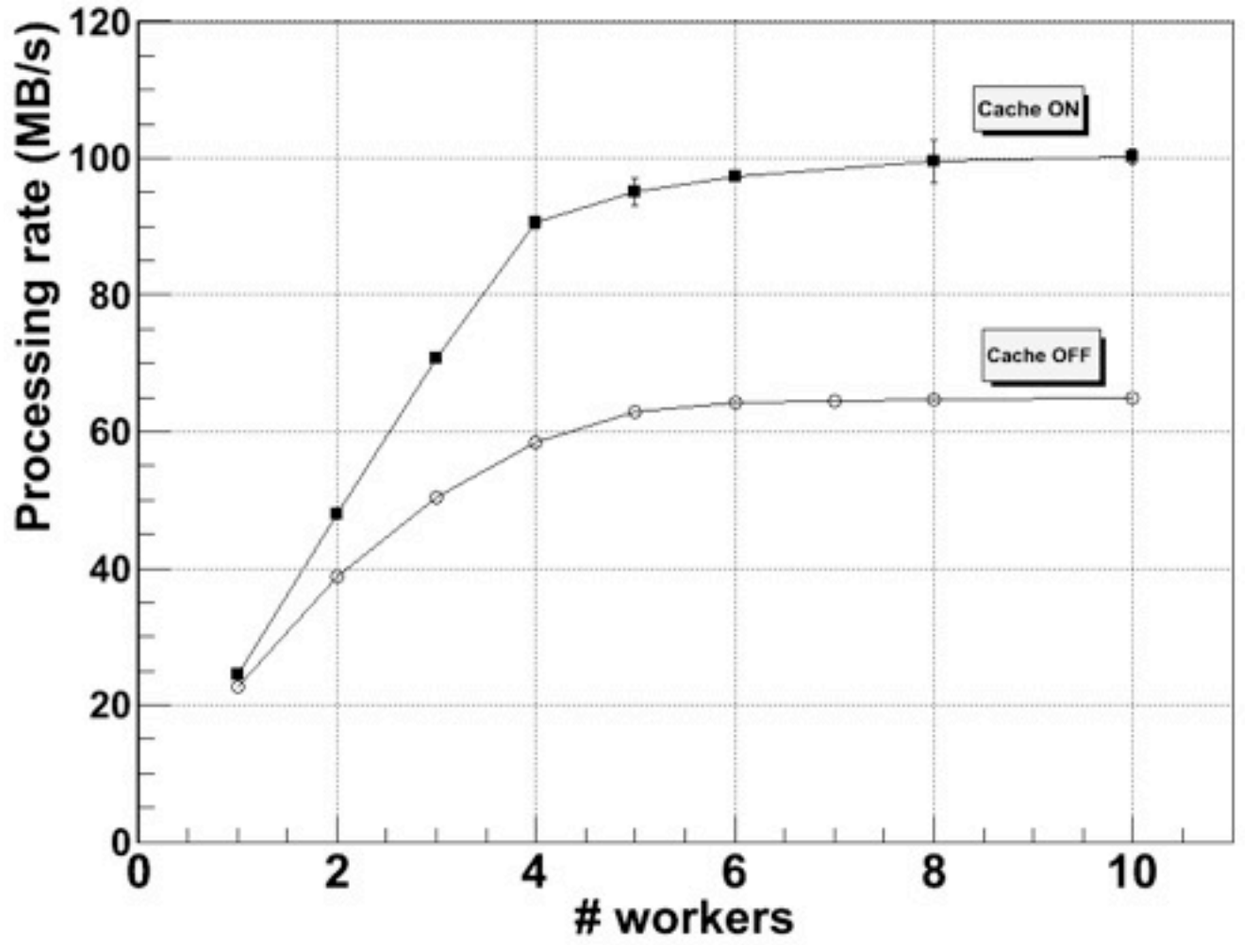
I/O device(s) on a single machine



Some observations

- HW RAID0 seems to scale well
- 4 SAS ≈ 2 SSD in RAID0 but cheaper
- RAM important for repeated runs on datasets fitting in it
- GB/s NIC fully exploited
  - 10 GB/s close to RAM ?

Client Cache for local files

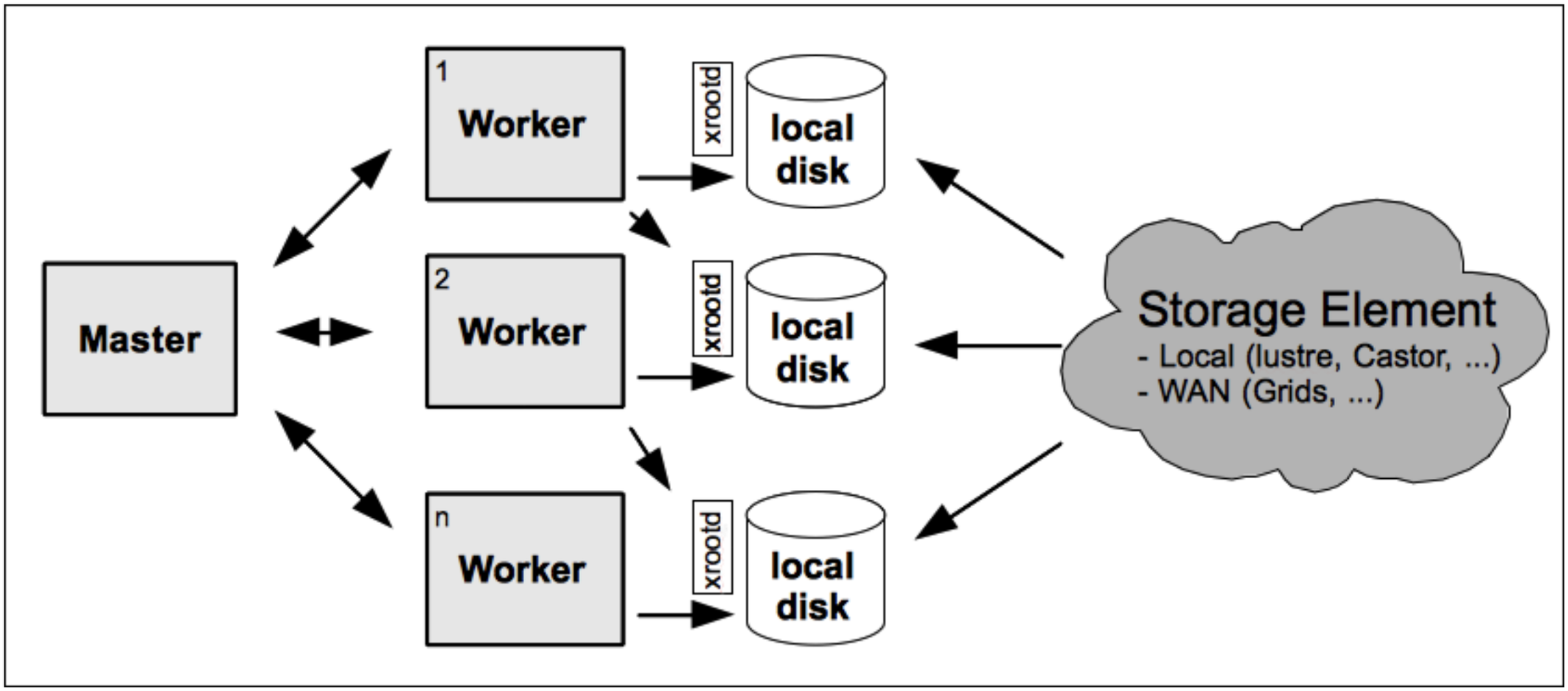


- Reduces the number of accesses to disks increasing the efficiency of device sharing

Increasing the effective number of I/O devices

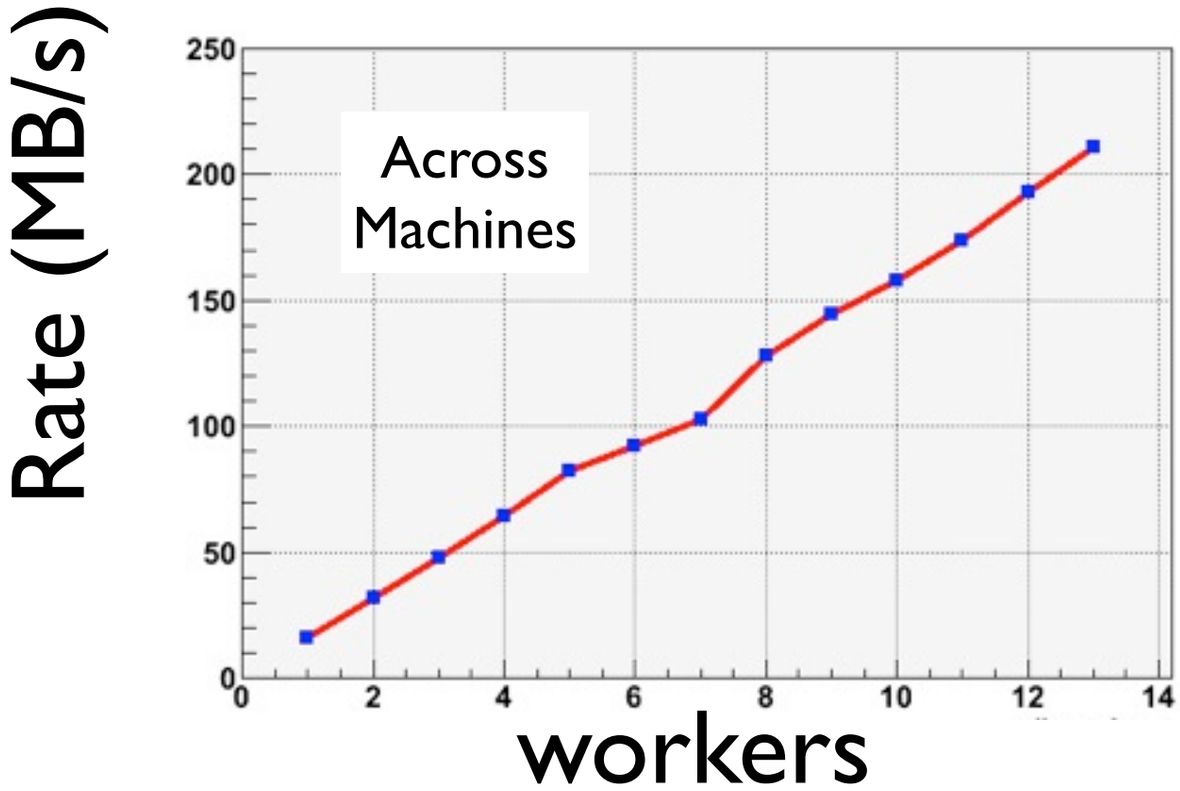
- LAN cluster w/
  - Pools on nodes aggregated via Scalla
  - Efficient MSS (Scalla, Lustre, GPFS,...)
- Populated from the Grid
  - Virtual MSS (see F. Furano)
- (Elastic) clouds for *ad hoc* cluster
  - Issue with data access (remote?)

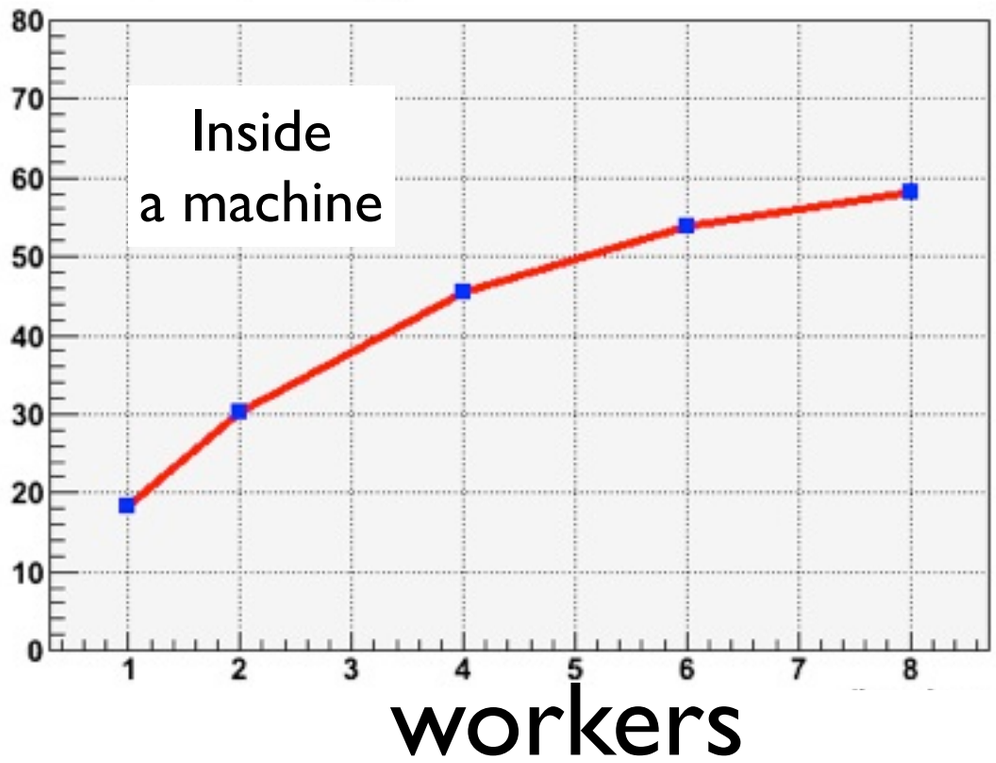
The LAN model



- Xrootd pools: ALICE@{CAF, SKAF, LAF}, ATLAS@{Wisconsin, BNL}, ...
- Lustre: ALICE@Darmstadt, NAF@Desy, ...

CAF scalability



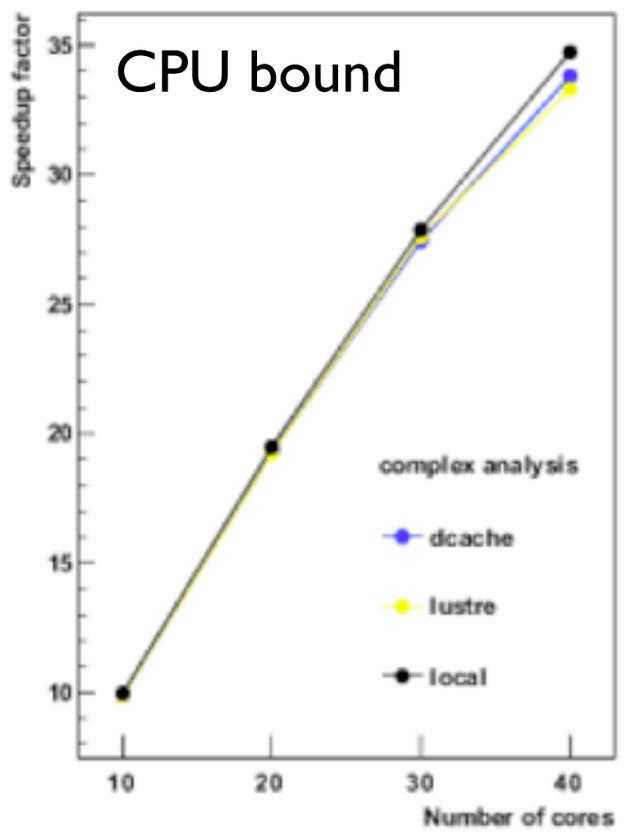


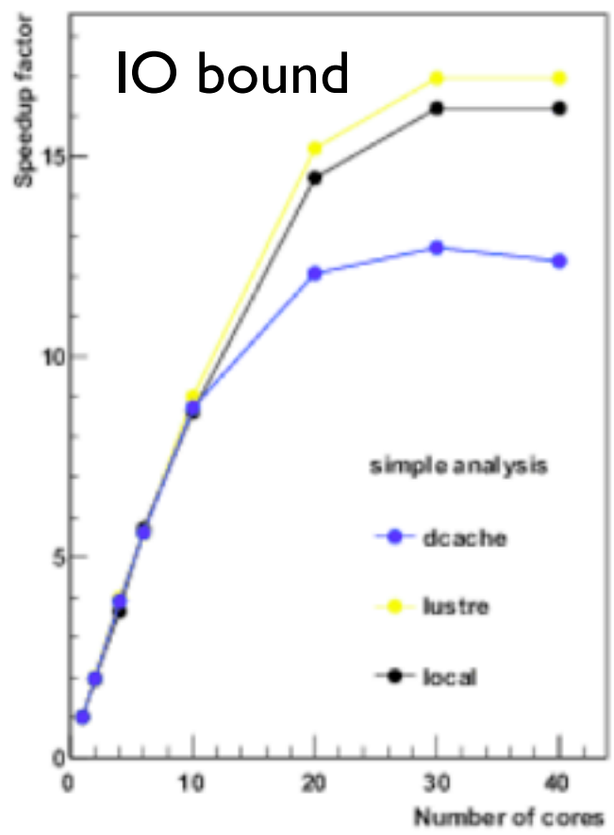
- ALICE ESD analysis on standard 8 core CERN machines

Local vs Lustre vs dCache

- 10 Opteron nodes, 4 cores, 8 GB RAM
- dCache behind a 10 GBs switch

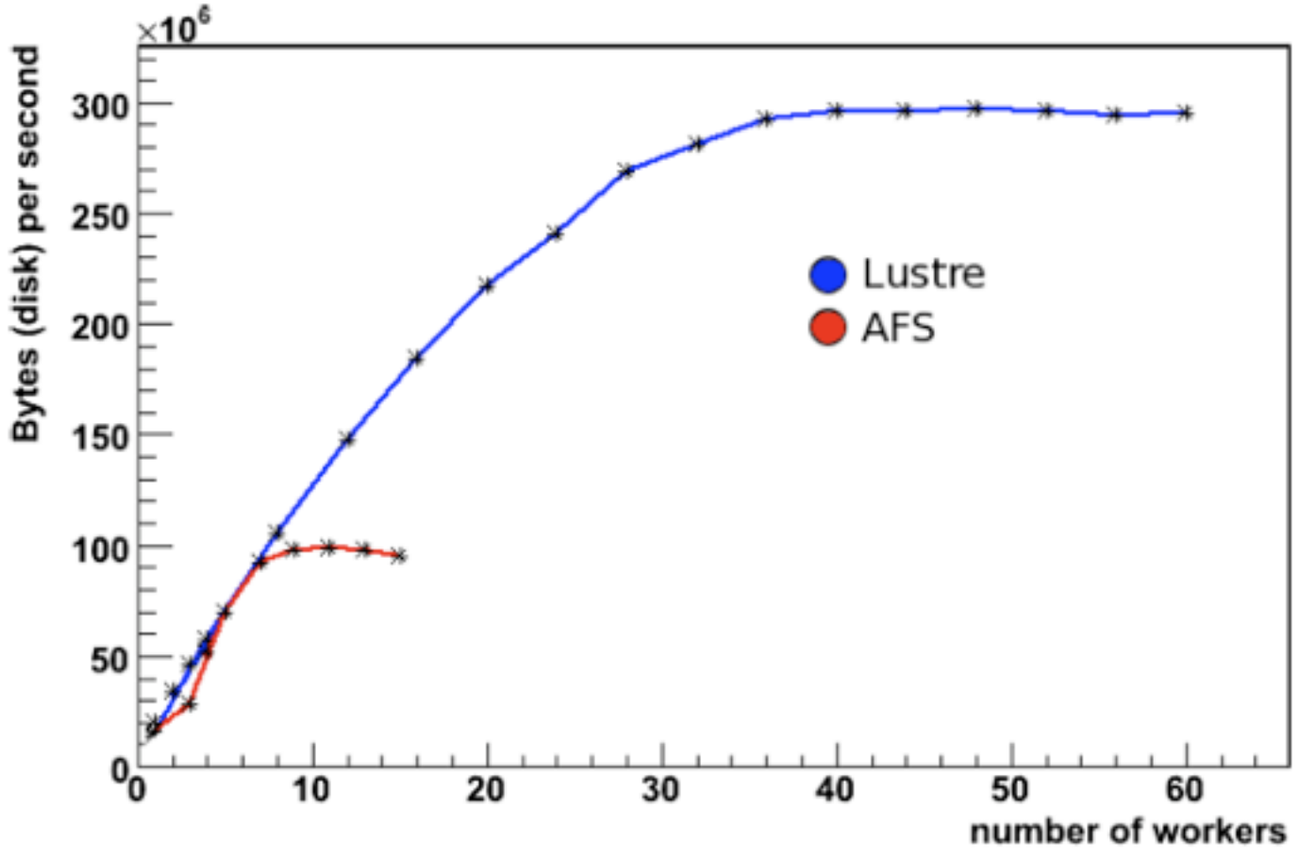
S.Panitkin, P. Calfayan, BNL





Lustre vs AFS

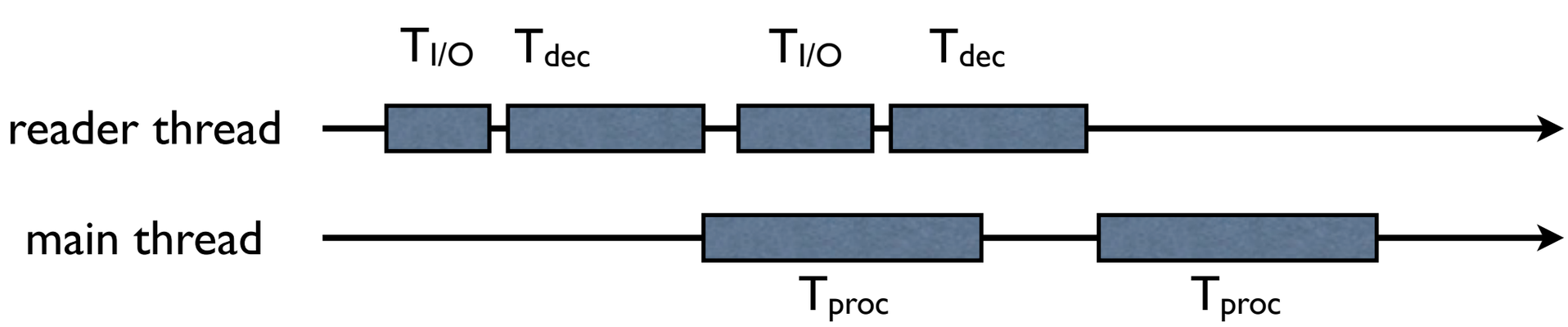
W. Behrenhoff, H. Stadie, Hamburg



As expected, Lustre via InfiniBand clearly outperforms AFS. Only one file server was used in this test. Thus, the input rate should increase further with additional servers.

Parallel Unzip

- The idea is to use a dedicated thread to speed-up decompression



- Expect to gain when $T_{proc} \sim T_{dec}$

Parallel Unzip: some preliminary results

- 4 core, 4 GB RAM
- Reading form HDD
- Rate in MB/s

N_{wrk}	1	2	4
Ref	15.1	25.5	29.3
Parallel unzip	19.5 +29%	28.2 +11%	34.5 +18%

Parallel Write?

- Interest in implementing something a la Gaudi
  - Separate process (thread?) collecting buffers to be written
- Solution outside PROOF, though the PROOF master could be a good collector

What did we learn?

- File layout, data access patterns may impact performances
- Caching (client, multi-layer) is important
- Good network requires adequate servers behind
- Distributed (and coordinated) data access is effective in increasing I/O

Thanks!