PROOF in Atlas Tier 3 model

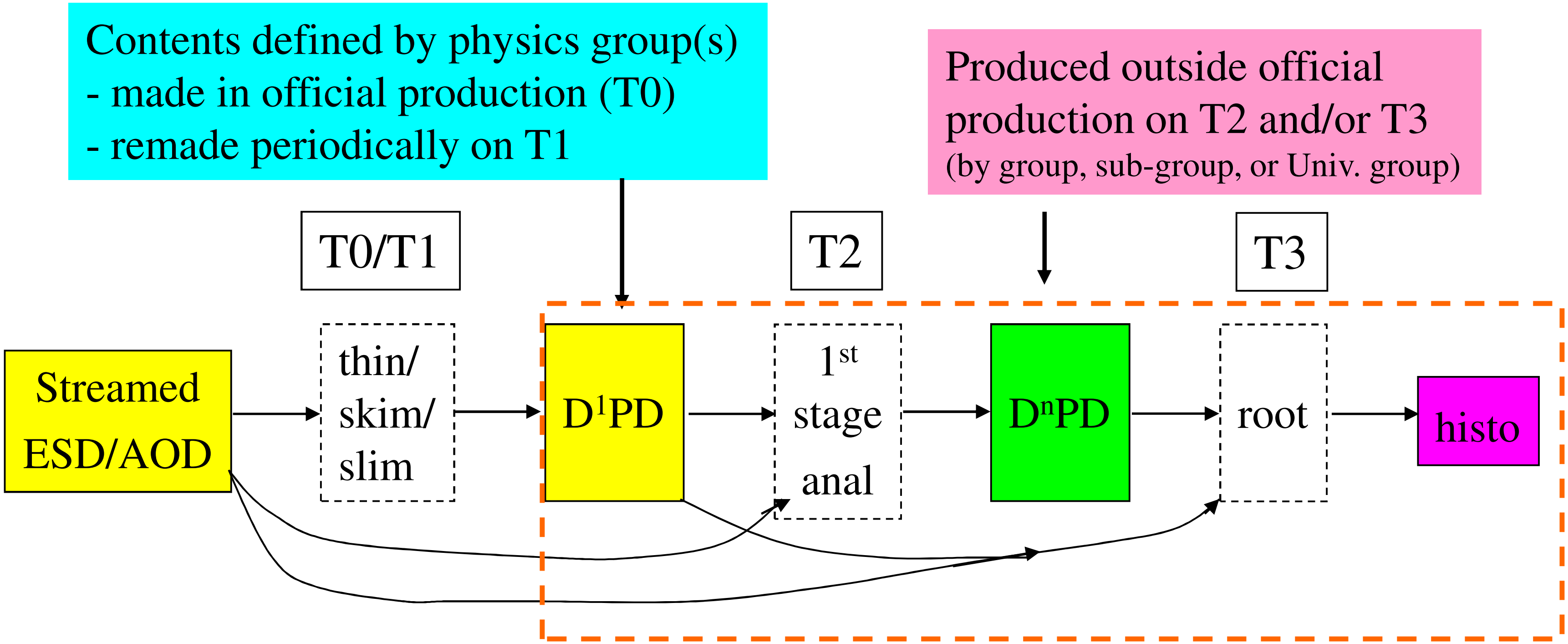
Sergey Panitkin

BNL

Outline

Part One: Proof in T3 environment
Part Two: I/O Issues in analysis on multi-core hardware

ATLAS Analysis Model – analyzer view



Contents defined by physics group(s)
- made in official production (T0)
- remade periodically on T1

Produced outside official production on T2 and/or T3
(by group, sub-group, or Univ. group)

ESD/AOD, D^{1}PD, D^{2}PD - POOL based

D^{3}PD - flat ntuple

Jim Cochran’s slide about the Analysis Model

Post-AOD analysis in Atlas

- Advantages of DPDs for T3:
  - Faster to analyze due to a smaller size and faster I/O
  - Less demand on storage space and infrastructure (1 year worth of DPD ~40TB)
  - Well suited for T3 types of analyses and scales
- How to make Tier 3 affordable, but still effective for Atlas analysis?
- How to analyze ~10^9 DPD events efficiently in Root?
- How to ensure fast analysis turnaround?
- **Use PROOF!** Parallel Root Facility - Root's native system for parallel distributed analysis**!**

PROOF Advantages

- Native to Root
- Easy to install
- Comes with build-in high performance storage solution - Xrootd
- Efficient in extracting maximum performance from cheap hardware
- Avoids drawbacks of “classical” batch systems
- Scalable in many ways, allows multi-cluster federations
- Transparent for users. System complexity is hidden
- Comes with its own event loop model (TSelector)
- Development driven by physics community
- Free, open source software
- Can nicely coexist with batch queues on the same hardware

Native to Root

- A system for the **interactive** or **batch** analysis of **very large sets** of **Root** data files on a **cluster of computers**
- Optimized for processing of data in Root format
- Speed up the query processing by employing inherent parallelism in event data
- Parallel Root Facility, originally developed by MIT physicists about 10 years ago
- PROOF is an integral part of Root now.
- Developed and supported by Root team.
- Distributed with Root. If you installed Root you have Proof.

PROOF and Xrootd

- One of the main advantages of modern PROOF implementation is its close cooperation with Xrootd
- PROOF is just a plug-in for Xrootd
- Typically PROOF uses Xrootd for communication, clustering, data discovery and file serving
- Note that PROOF can be used without Xrootd based storage. It works with many types of SE (dCache, Lustre, NFS boxes, etc...)
- But PROOF is at its best when Xrootd provides high performance distributed storage
- When PROOF is used in conjunction with Xrootd SE it automatically ensures data discovery and local data processing: a job running on a given node reads data stored on that node. This typically provides maximum analysis rate and optimal farm scalability.

PROOF and Xrootd

```
# General section that applies to all servers
#
all.export /atlas

if redirector.slac.stanford.edu
all.role manager
else
all.role server
fi
all.manager redirector.slac.stanford.edu 3121
# Cluster management specific configuration
#
cms.allow *.slac.stanford.edu
# xrootd specific configuration
#
xrootd.fslib /opt/xrootd/prod/lib/libXrdOfs.so
xrootd.port 1094

### Load the XrdProofd protocol:
if exec xrootd
xrd.protocol xproofd:1093
/opt/xrootd/prod/lib/libXrdProofd.so
fi
```

Easy configuratioon

A few extra lines in Xrootd config. file
Plus a simple PROOF config. file

Node1 master

Node2 worker
Node2 worker
Node3 Worker
Node3 Worker

Efficient Processing

- Efficient in extracting maximum performance from (cheap) hardware
- Easy, self-organized clustering
- Well suited for (if not geared to) analysis farms with distributed **local** storage
    - Local data processing is encouraged – automatic matching of code with data. This typically means optimal I/O
- A job is automatically split in optimal number of sub-jobs
- All jobs are dynamically managed to insure maximum resource utilization
    - Load is actively managed and balanced by master (packetizer)
    - Pull architecture for load management
    - When local processing is exhausted remote access automatically begins so all CPUs are utilized

Efficient processing

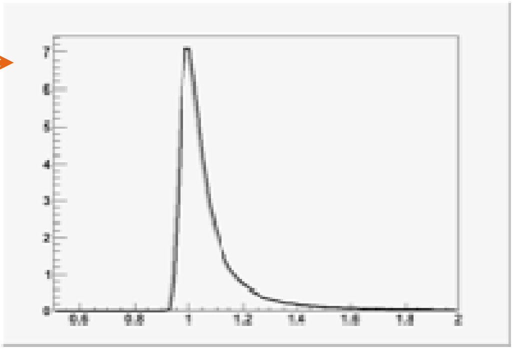
- The PROOF packetizer is the heart of the system
- It runs on the client/master and hands out work (packets) to the workers
- Packet can be a few hundred events
- The packetizer takes data locality and storage type into account
  - Matches workers to local data when possible
- By managing workers load it makes sure that all workers end at the same time

Pull architecture

workers ask for work, no complex worker state in the master

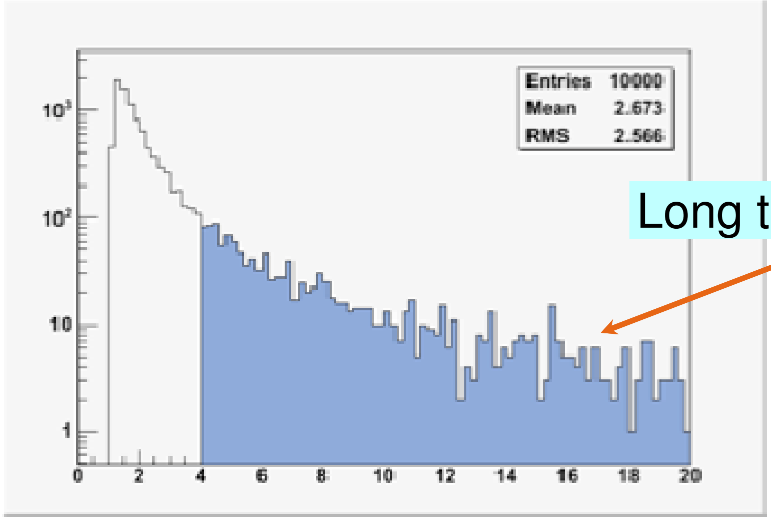
PROOF Pull Technology Avoids Long Tails

- In push approach last job determines the total execution time
  - Basically a Landau distribution



- Example:
  - Total expected time 20h, target 1h
  - 20 sub-jobs, 1h +/- 5%

10000 toy experiments



Long tails, e.g. 15% > 4h

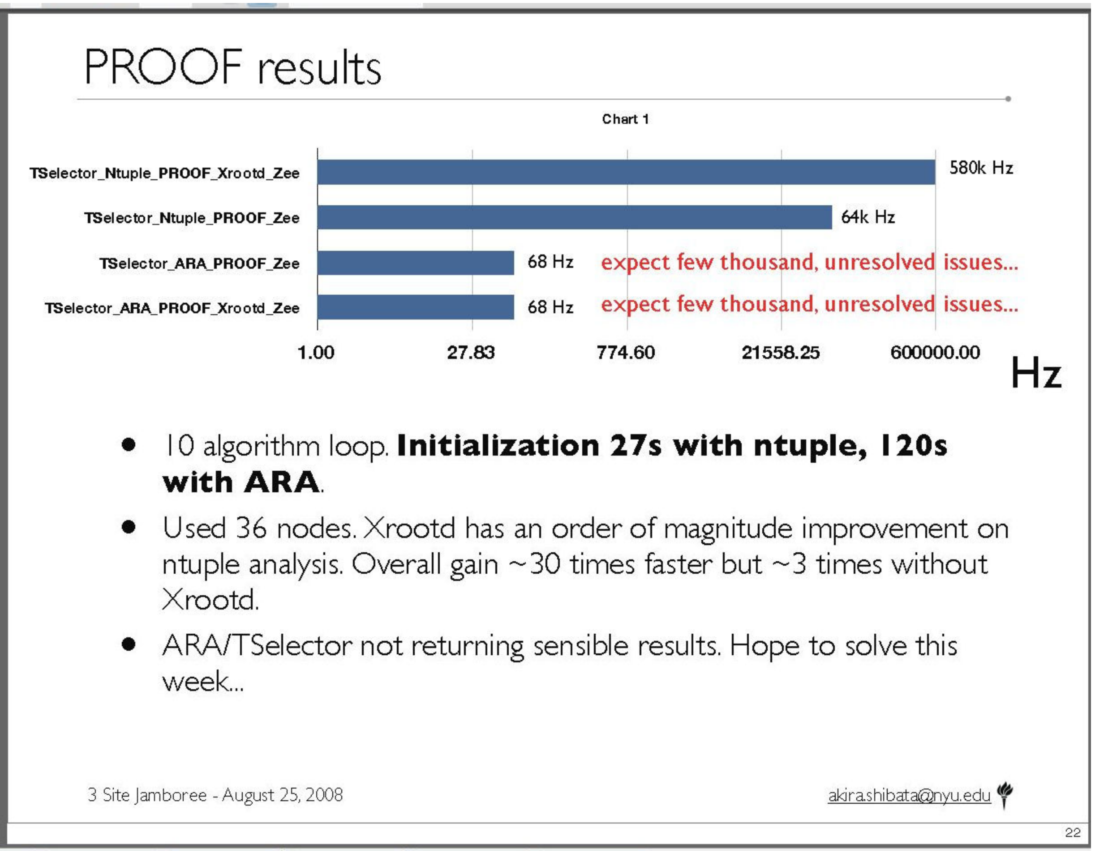
Time of slowest sub-job

Courtesy Fons Rademakers, Gerri Ganis

Efficient Processing

Akira's talk about ROOT analysis comparison at the Jamboree

http://indico.cern.ch/getFile.py/access?contribId=10&sessionId=0&resId=0&materialId=slides&confId=38991



PROOF results

Chart 1
TSelector_Ntuple_PROOF_Xrootd_Zee
TSelector_Ntuple_PROOF_Zee
TSelector_ARA_PROOF_Zee
TSelector_ARA_PROOF_Xrootd_Zee
580k Hz
64k Hz
68 Hz
expect few thousand, unresolved issues...
68 Hz
expect few thousand, unresolved issues...
1.00
27.83
774.60
21558.25
600000.00
Hz

- 10 algorithm loop. **Initialization 27s with ntuple, 120s with ARA**.
- Used 36 nodes. Xrootd has an order of magnitude improvement on ntuple analysis. Overall gain ~30 times faster but ~3 times without Xrootd.
- ARA/TSelector not returning sensible results. Hope to solve this week...

3 Site Jamboree - August 25, 2008

akira.shibata@nyu.edu

22

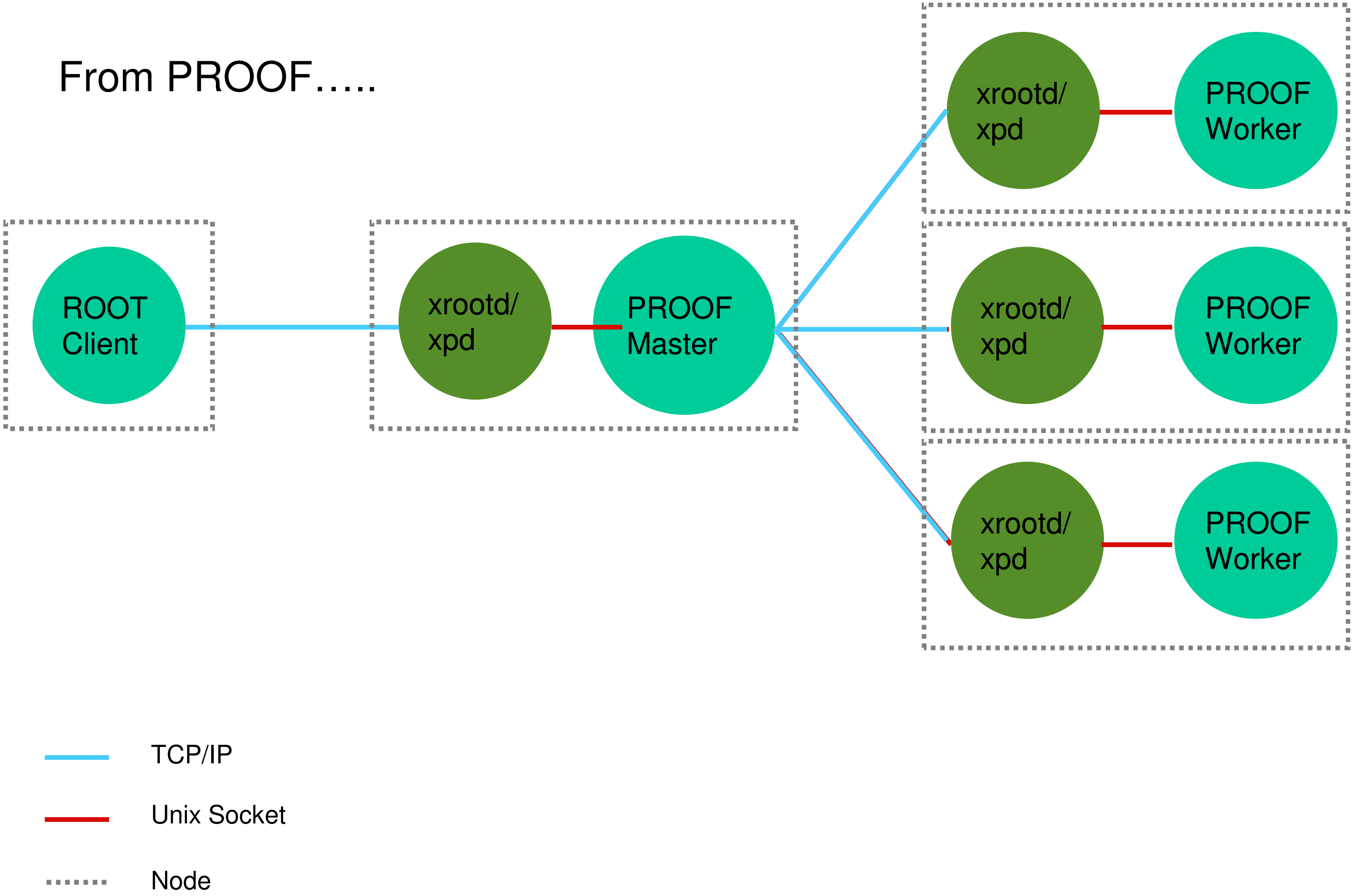
100 KHz analysis rate

- Akira Shibata reported his tests with PROOF
- Root analysis benchmark package

- Good results for Ntuple based analysis
- Xrootd improves I/O performance

Scalability

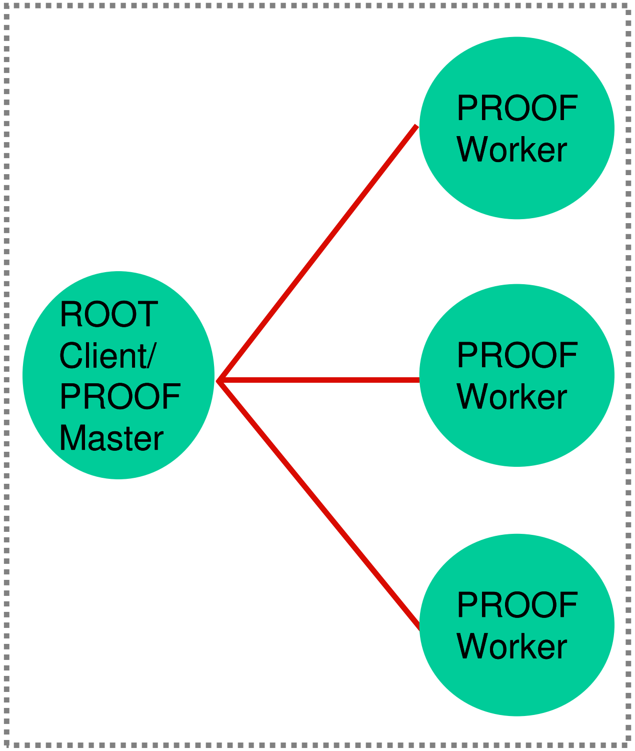
From PROOF…..



Courtesy Fons Rademakers, Gerri Ganis

Scalability

…To PROOF Lite…



— Unix Socket

⋯ Node

Perfect for multi-core desktops/laptops

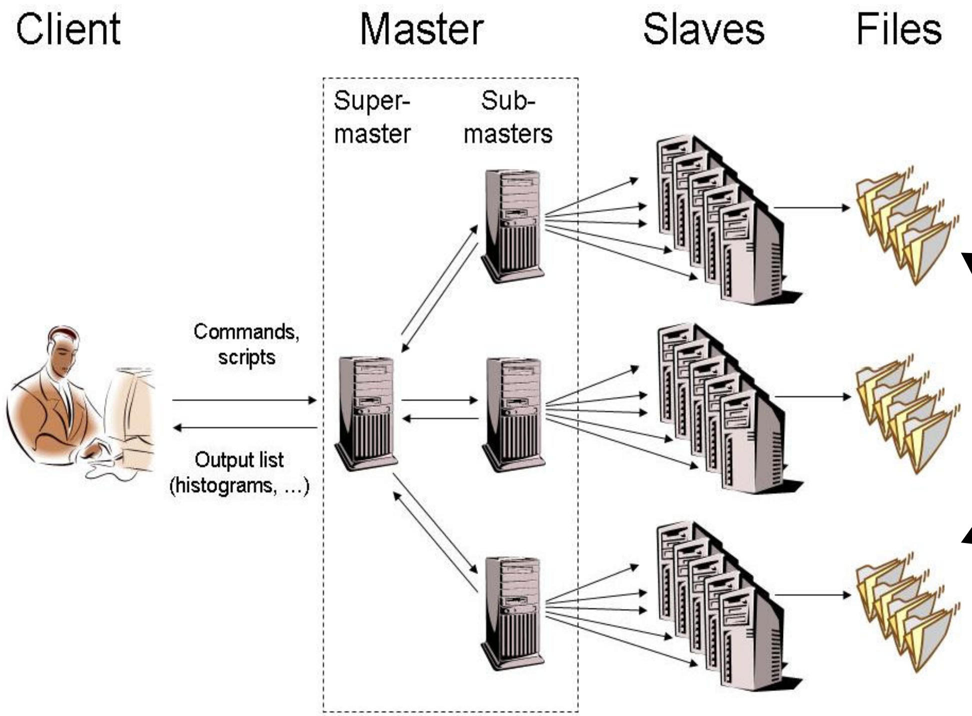
Courtesy Fons Rademakers, Gerri Ganis

PROOF Lite

- PROOF optimized for single many-core machines
- Zero configuration setup (no config. files and no daemons)
- Workers are processes and not threads for added robustness
- Like PROOF it can exploit fast disks, SSD’s, lots of RAM, fast networks and fast CPU’s
- Once your analysis runs on PROOF Lite it will also run on PROOF
  - Works with exactly the same user code as PROOF

Scalability

….To Federated Clusters….



Adapts to wide area *virtual* clusters

Geographically separated domains, heterogeneous machines

Super master is users' single point of entry. System complexity is hidden

Automatic data discovery and job matching with local data

Support and documentation

- Main PROOF Page at CERN, PROOF worldwide forum
    - http://root.cern.ch/twiki/bin/view/ROOT/PROOF
- USAtlas Wiki PROOF page
    - http://www.usatlas.bnl.gov/twiki/bin/view/ProofXrootd/WebHome
- Web page/TWIKI at BNL with general farm information, help, examples, tips, talks, links to Ganglia page, etc.
    - http://www.usatlas.bnl.gov/twiki/bin/view/AtlasSoftware/ProofTestBed
- Hypernews forum for Atlas PROOF users created

  hn-atlas-proof-xrootd@cern.ch

  https://hypernews.cern.ch/HyperNews/Atlas/get/proofXrootd.html

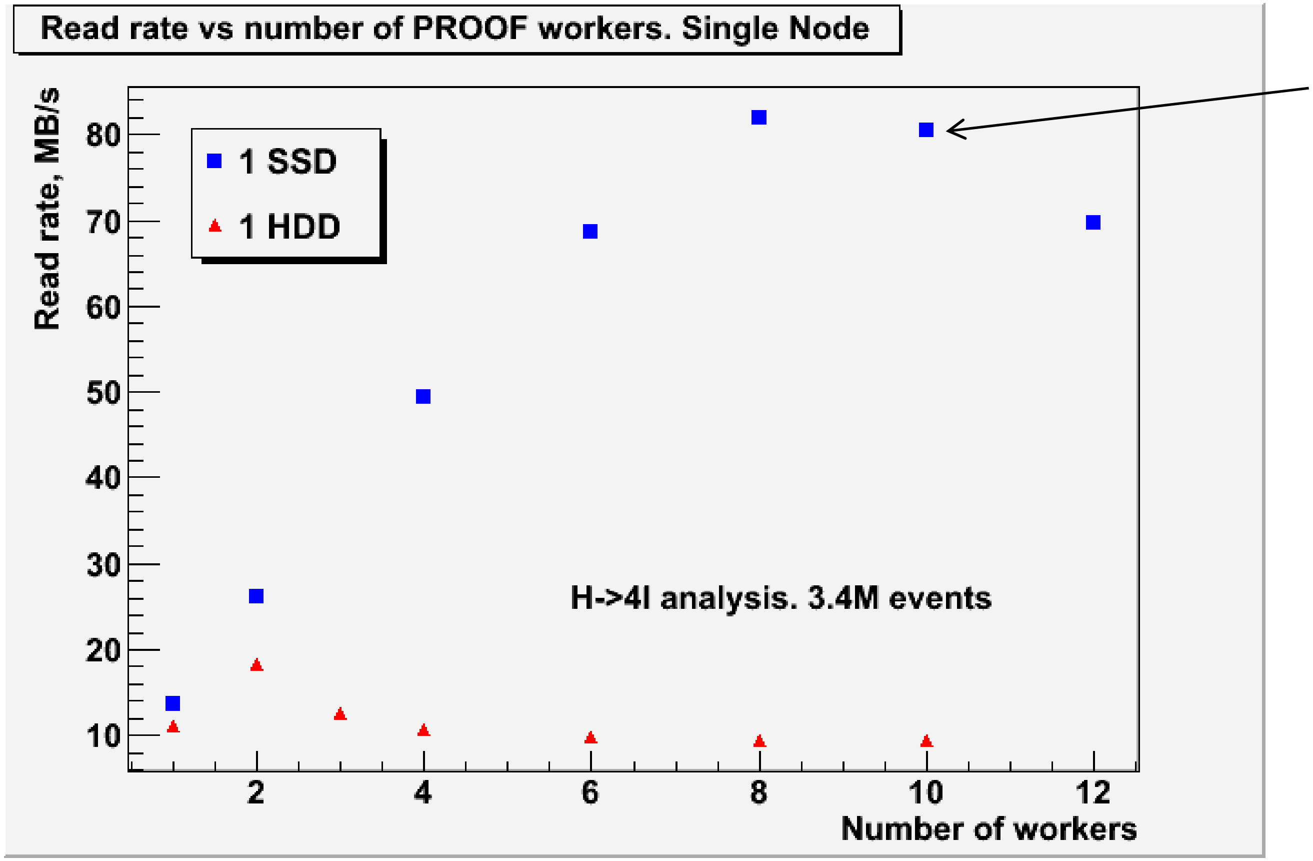
PROOF at LHC

- Alice
  - will run PROOF on CAF for calibrations, alignment, analysis, etc
  - PROOF farm at GSI T2
  - Various local T3s
  - Integration of PROOF with AliRoot
- Atlas
  - PROOF farm at T2 at Munich LMU (and German NAF?)
  - PROOF farm(s) at BNL T1
  - PROOF test farm at UTA T2
  - Proof test farm at Universidad Autonoma de Madrid T2
  - PROOF farm at Wisconsin T3
- CMS
  - PROOF at NAF Germany
  - PROOF cluster at Purdue - USCMS T2
  - PROOF farm at Oviedo
  - Planned farms at ETH and PSI (T3 or T2)

Part Two: I/O Issues

- PROOF can effectively exploit local processing (I/O localization)
- How does it work with multi-core hardware?
- When disk subsystem becomes a bottleneck?
- Next few slides are from my CHEP09 talk about PROOF and SSDs:
- It can be found here: http://indico.cern.ch/contributionDisplay.py?contribId=395&sessionId=61&confId=35523

H->4l analysis. SSD vs HDD



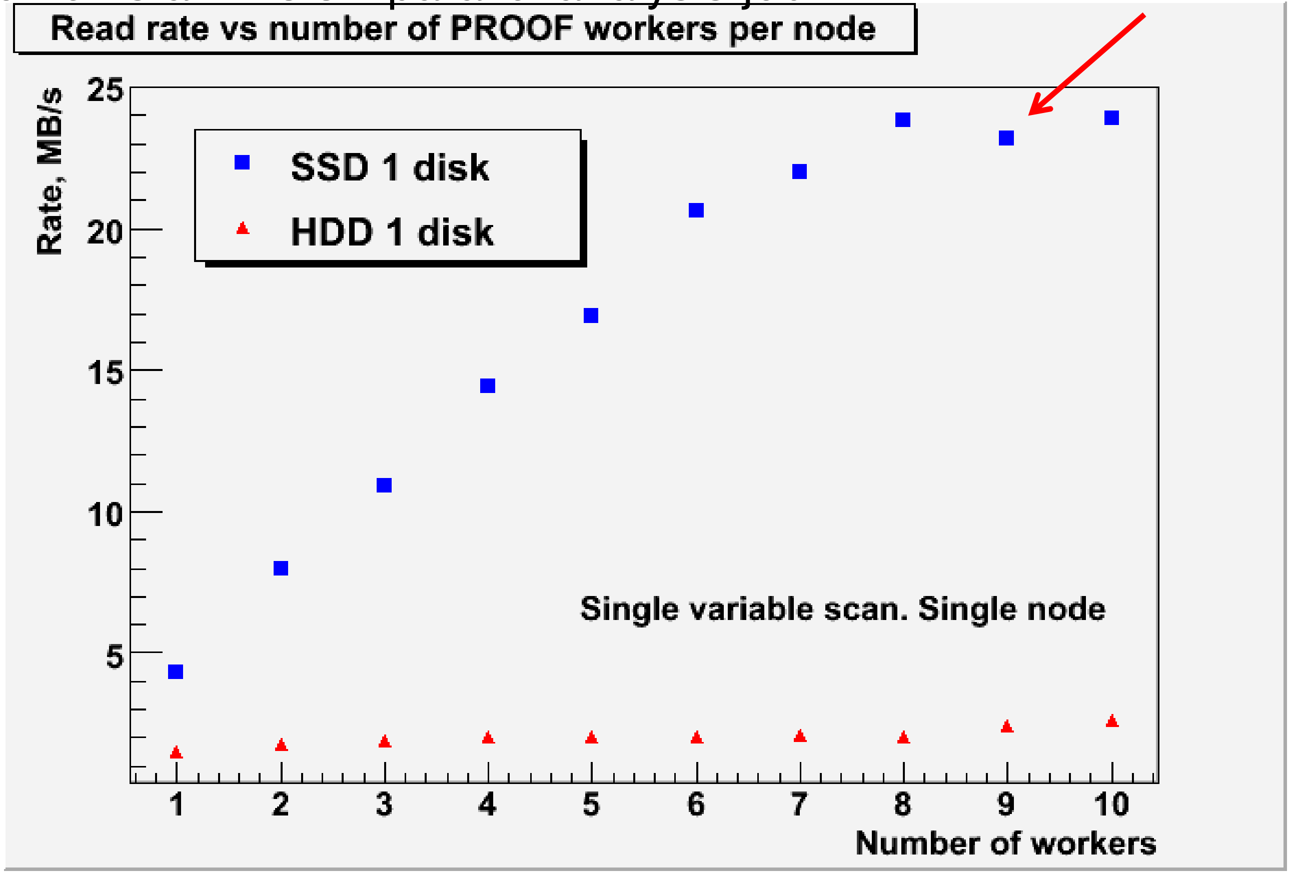
CPU Limited

Dual quad core
16GB RAM

SSD is about 10 times faster at full load
Best HDD performance at 2 worker load
Single analysis job generates ~10 -14 MB/s load with given hardware

Interactive analysis. SSD vs HDD

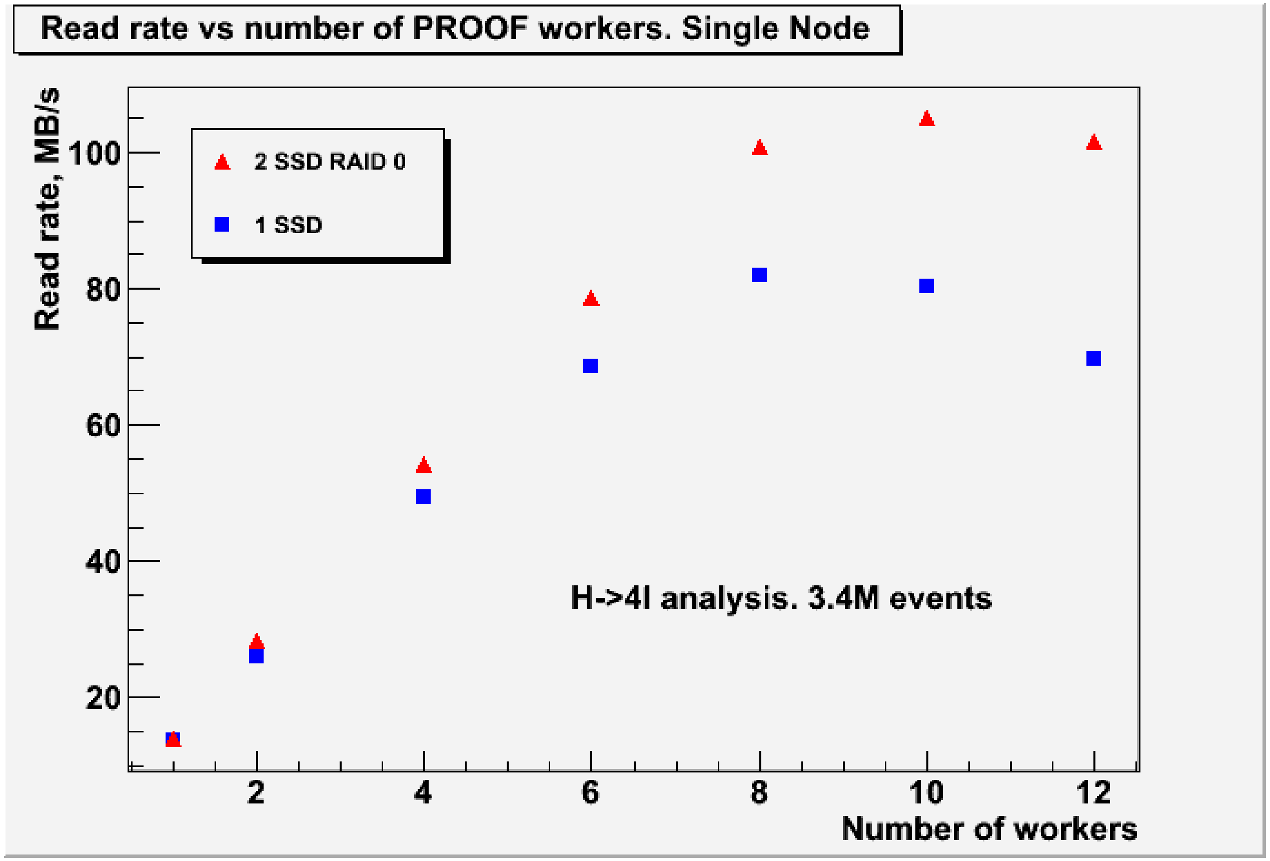
- Worker is a PROOF parallel analysis job



CPU limited

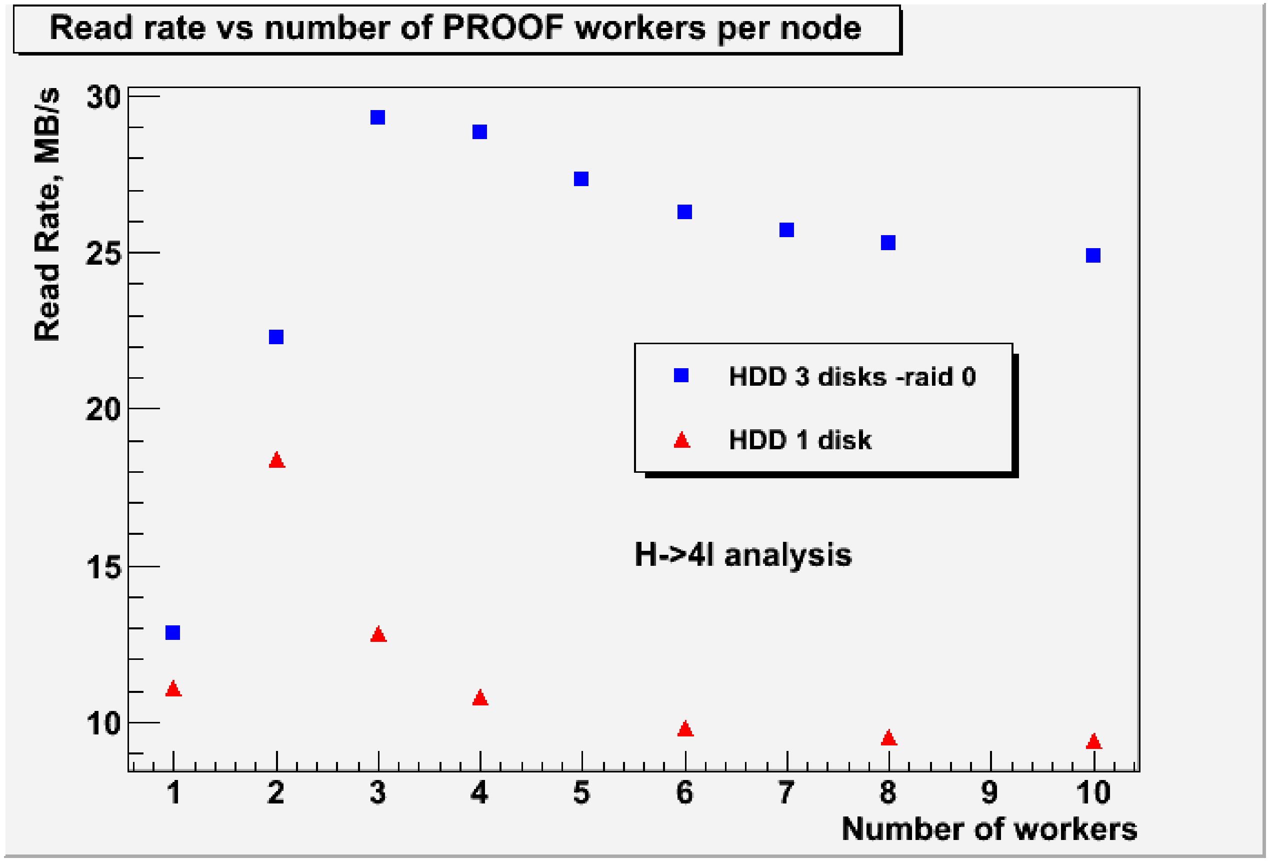
- SSD holds clear speed advantage
- ~Up to10 times faster in concurrent read scenario

H->4l analysis. SSD RAID 0



SSD 2 disk RAID 0 shows little impact up to 4 worker load

H->4l analysis. HDD: single vs RAID

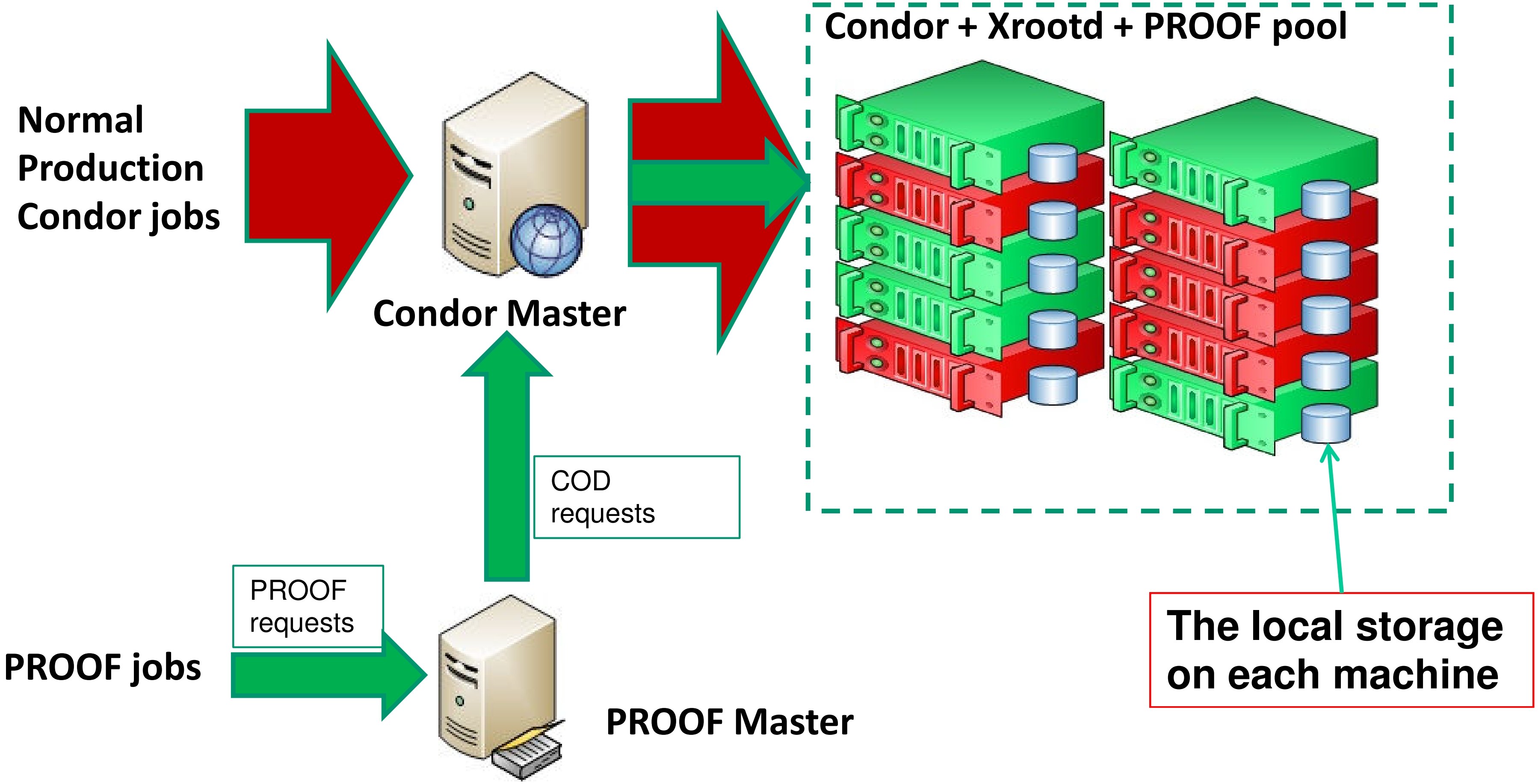


3x750 GB HDD RAID peaks at ~3 worker load

Single HDD disk peaks at 2 worker load, then performance rapidly deteriorates

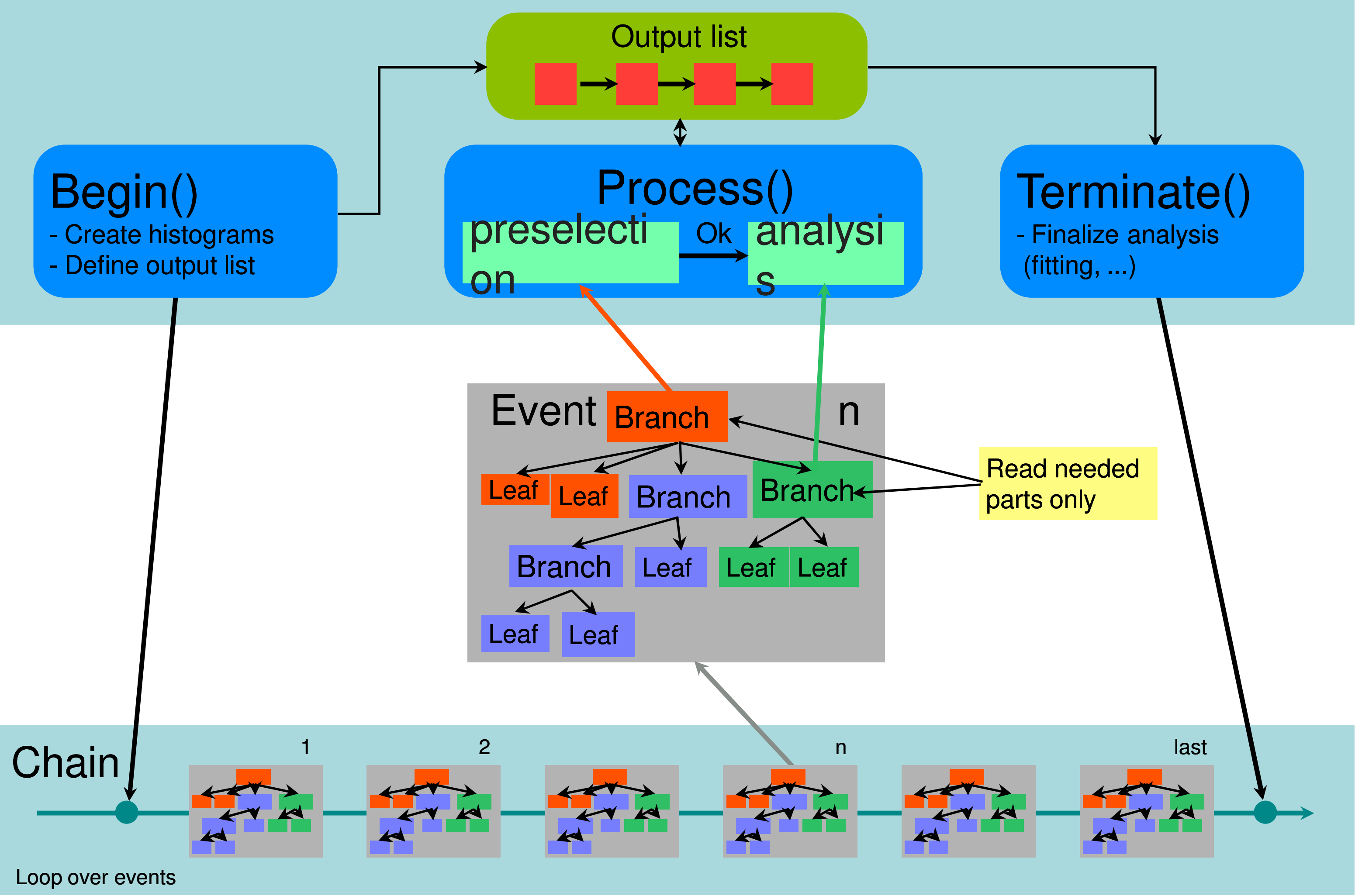
The End

The basic PROOF+Condor Integration


Condor + Xrootd + PROOF pool
Normal Production Condor jobs
Condor Master
COD requests
PROOF requests
PROOF jobs
PROOF Master
The local storage on each machine

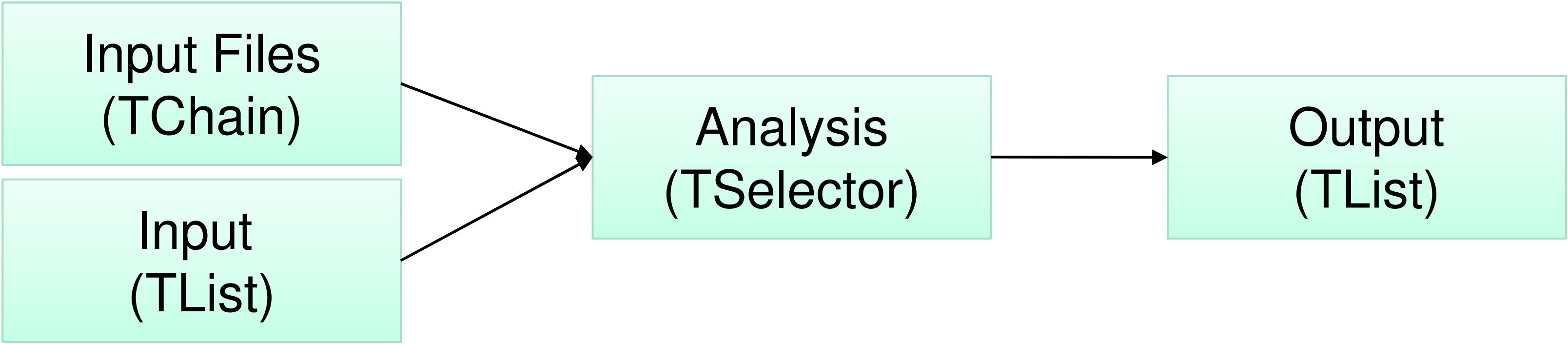
Sergey Panitkin

The ROOT Data Model
Trees & Selectors



How to use PROOF

- PROOF is designed for analysis of independent objects, e.g. ROOT Trees (basic data format in partice physics)
- Files to be analyzed are put into a chain (TChain) or a data set (TDSet), e.g. collection of files
- Analysis written as a selector
- Input/Output is sent using dedicated lists
- If additional libraries are needed, these have to be distributed as a “package”



Input Files (TChain)

Input (TList)

Analysis (TSelector)

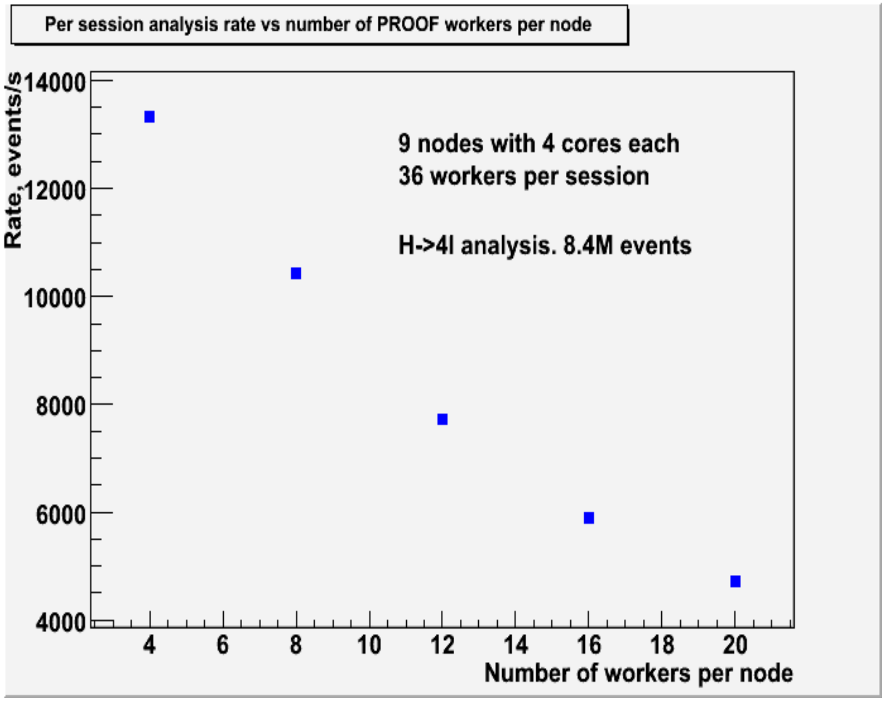
Output (TList)

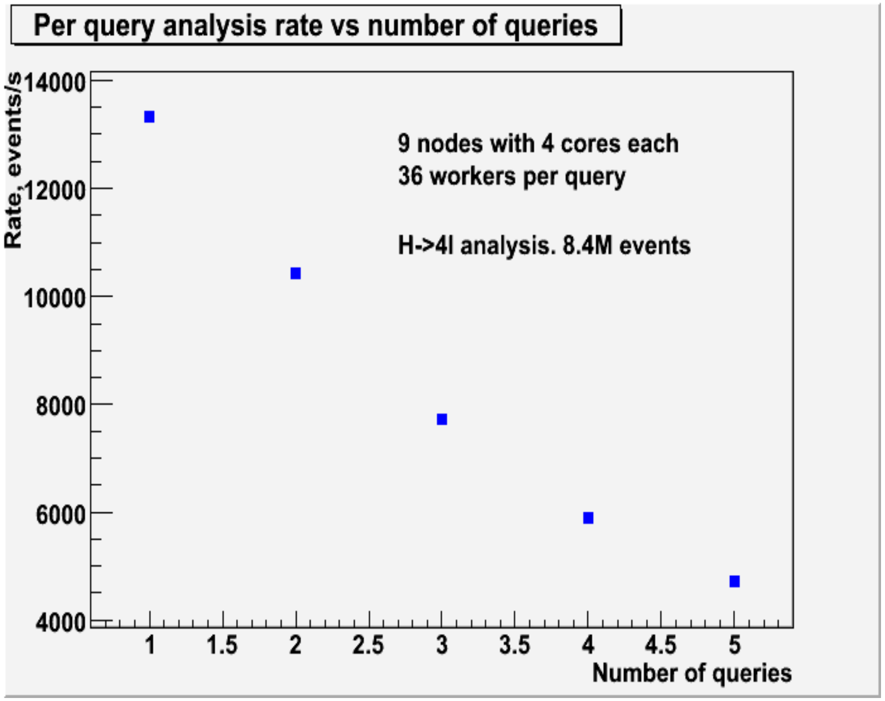
TSelector

- TSelector is a framework for analysis of event like data
- You derive from TSelector class and implement member functions with specific algorithm details
- During processing Root calls your functions in a predefined sequence
- TSelector skeleton can be automatically generated
- ROOT provides the TTree::MakeSelector function to generate a skeleton class for a given TTree.

```
root > TFile *f = TFile::Open("treefile.root")
root > TTree *t = (TTree *) f->Get("T")
root > t->MakeSelector("MySelector")
root > .!ls MySelector*
        MySelector.C MySelector.h
```

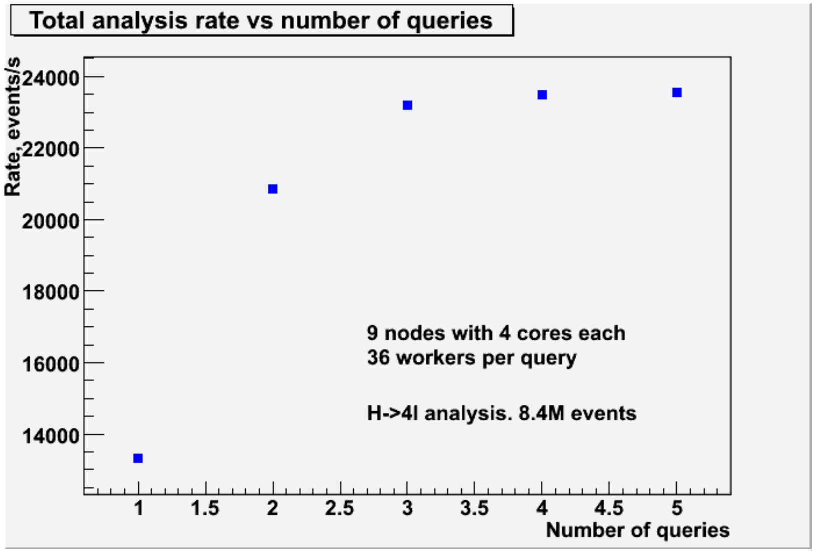
Multisession performance

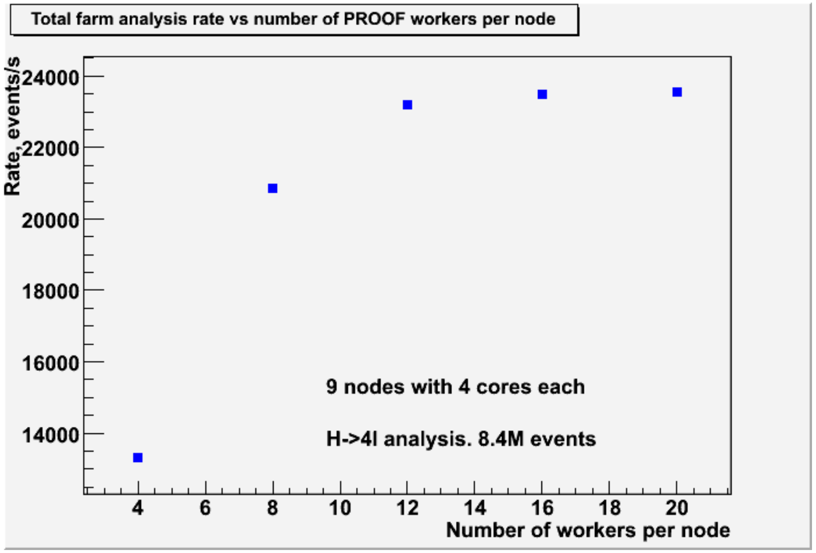


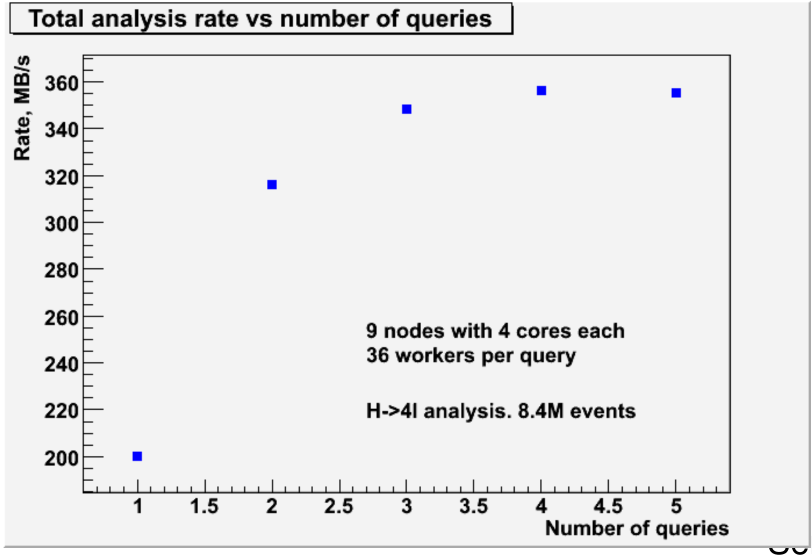


As expected per query performance drops as number of queries increases.
Resource sharing between jobs with equal priorities.

Analysis rate scaling







- Aggregate analysis rate saturates at about 3 (full load) queries
- Max analysis rate is about 360 MB/s for a given analysis type

It makes sense to run PROOF farm at optimal number of queries