

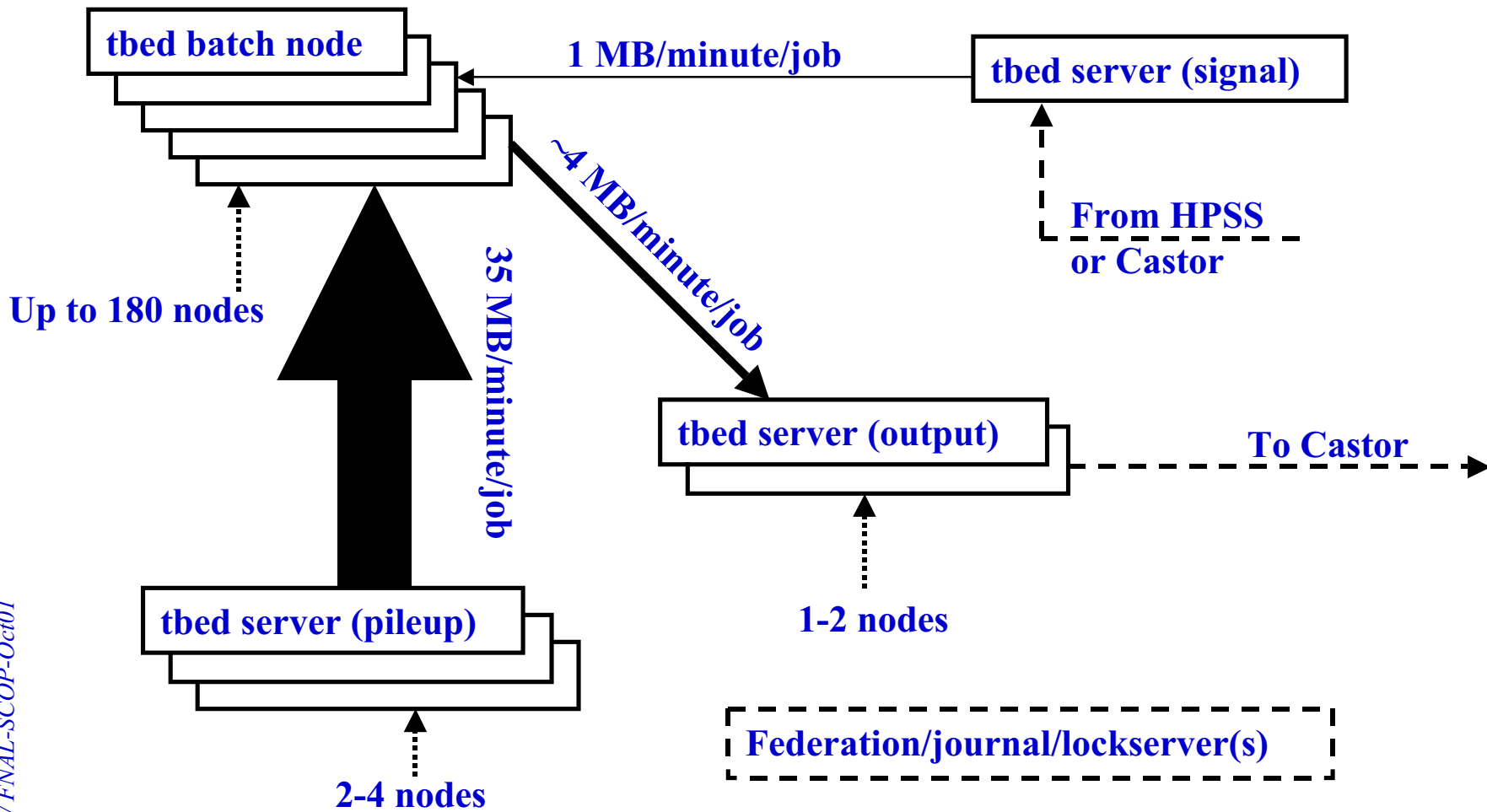
# **CMS Software, Near Term Technical Challenges**

*In particular issues related to the layers of SW required  
for persistency and routes to common solutions*

**David Stickland**  
**CMS SCOP Review, October 01**



# Production test configuration



DPS/FNAL-SCOP-Oct01

**N.B. Widths of solid arrows are to scale!**



# Powerful AMS features

- λ **AMS reads/writes files, serves pages. User never sees files.**
  - ➔ User entry-point to data: 'federation bootfile' and 'owner/dataset' names (keys to navigate metadata).
  - ➔ User really has no idea where the data is coming from.
  - ➔ Events split over many files to try to match analysis access patterns.
    - **Very complex relationship between files and runs/events.**
- λ **Multiple concurrent readers and/or writers.**
  - ➔ Lockserver avoids conflicts, controls access.
  - ➔ Many jobs writing to same (meta-) data files during production.
  - ➔ Framework (COBRA) takes care of closing files at 2GB limit, and asks the AMS to create new files as containers grow.
    - **Production jobs never know which files they are going to write to!**
- λ **Transparent MSS interface.**
  - ➔ HPSS, now Castor. Files retrieved from MSS by AMS on demand, purged by disk-pool manager when space needed.
  - ➔ MSS may be the grid! Already have all the tools we need (HPSS interface + GDMP will do).
  - ➔ Knowledge of which files are on disk exists, but outside AMS ('cost of access' in grid terms).
- λ **Host-based file redirection (RRP).**
  - ➔ Load-balancing, same file served (R/O) by many hosts.
  - ➔ Reliability, switch servers in/out of use transparently.
  - ➔ Can and do reconfigure running farms on the fly.
  - ➔ 'Shallow-copies' managed centrally without knowing they exist!



# Copies and Files

## λ Shallow-copies

- ➔ Shallow-copy in the C++ sense.
- ➔ Users who want to add their own data or schema do so by making a shallow-copy of the federation.
  - ❑ Upgrade schema on the clone.
  - ❑ Add private databases to it.
  - ❑ Full original datasets available from the original locations via the catalogue.
- ➔ Originally a kludge for access control, but also very useful as a step towards 'deep-copies' and standalone federations.

## λ Deep-copies

- ➔ Shallow-copy with some of the bulk data copied too.
  - ❑ Better performance by using local disk.
  - ❑ Recluster events or parts of events from sparse analyses ⇒ reduce MSS access, dependence on remote server...
  - ❑ Take data offline ('laptop on a plane').
- ➔ Also have totally standalone federations for small private analyses.
  - ❑ These still need the file-management capabilities of the full production.
  - ❑ File-level Summary

## λ Users never see/ask for files.

- ➔ Shallow/deep copies a special case.
- ➔ Users write to containers, lockserver ensures multiple writers can write to the same file.

## λ Federation catalogue + RRP + MSS manage file movement.

- ➔ Same file may be served to different users in different ways, on different servers etc.



## Other Lessons from Productions

- λ **At CERN, tested up to 300 CPU's simultaneously writing into a single Objectivity federation**
  - ➔ **Worst case tests:**
    - ❑ **Single owner/dataset ( CMS terminology for a coherent dataset sharing same Objectivity system files, collection files etc)**
    - ❑ **Contention for allocating files and containers**
      - Can result in excessive startup times
      - Can reduce farm-efficiency as jobs become auto-synchronized
  - ➔ **Solutions exist**
    - ❑ **Reducing job coupling**
    - ❑ **Compartmentalizing production farm into manageable unit sizes**
      - Effectively we do this with world-wide production farms
      - Methods exist, but not used in production yet, to build single collections of the recombined outputs from the different farm
- λ **Very large Federations limited at 64k files**
  - ➔ **Objectivity solution is long-refs in multi-federations**
- λ **Different requirements of Production and Analysis “clustering”**
  - ➔ **Production, complete and close files rapidly to reduce exposure to hardware errors**
  - ➔ **Analysis, cluster objects according to (different) use cases**



# COBRA

## λ Continuation of prototype (CARF)

- ➔ Separated from ORCA/OSCAR/FAMOS in last 6 months
- ➔ But, 10-20% is traditional “event-loop” part of framework, the rest is occupied with Objectivity specifics
  - ❑ (“get the name of next file for this dataset”, etc)
- ➔ DBA specific code should be split out into yet another layer
  - ❑ Will clean up COBRA from its prototype remnants
  - ❑ Will allow better use of our human resources by dividing responsibilities along cleaner lines
  - ❑ Will clarify DB system-specific and system-generic partition

## λ There is a missing service-level layer that used to be in CERNLIB but is now being written separately by everyone

- ➔ For example old “TIMEL” functionality
- ➔ New functionalities like exception handling
- ➔ We propose to work with GAUDI and IT/API to find common solutions, or even just adopt their solutions for these problems

## λ Experiment specific developments have a valid place as the experiments try to optimize according to what they think will be important

- ➔ That’s why we have different detector choices

## λ But no reason, in fact every reason not, to duplicate service parts



## CMS-ROOTIO Workshop (Oct 10/11)

- λ **ROOT team has built a powerful system for object streaming and object description in the IO files**
- λ **Users have built, and now ROOT is implementing, inter-file references (OIDS!)**
  - ➔ See <http://www.AmbySoft.com/mappingObjects.pdf> for example, strong emphasis on having control of your “oids”
    - ❑ Should have no “Business meaning” (that will anyway change)
    - ❑ Do not allow a proprietary layer to do this !
    - ❑ Much easier to change persistency later
    - ❑ HIGH/LOW OID generation can reduce bottlenecks due to central allocation
- λ **Need a true DB layer at least for the OID mapping, collections etc**
- λ **CMS believes we also need the functionalities of the Objectivity AMS/RRP etc to delegate “files” and “file responsibilities” to a very low level of the system**
  - ➔ GRID should concentrate the mind on this issue as the whole concept of physical and logical files requires addressing
  - ➔ This layer will have to be built by LHC community
- λ **With these three layers one could build a sustainable persistency solution that could satisfy most reasonable use-cases for LHC**
  - ➔ Try to build an LHC common approach, sharing of work



# ROOTIO/CMS Mismatches ?

λ **CMS has some important technical issues to resolve with the ROOT team**

➔ **Degree of intrusiveness**

❑ Resolvable.

➔ **Technical c++ issues**

❑ Global-state, threads, exception handling..

➔ **Use of external software**

❑ Rather than subsumption of it

zlib example

➔ **Definitions of modularity have to be agreed**

❑ Binary only versus Binary+Source+Logical

❑ Long term scalability and maintenance is at issue here

➔ **Development model**

❑ management, oversight, SW packaging, priorities, "ownership", and so on

➔ **Legacy support**

❑ May be an issue by 2005,2010

λ **But it is my belief that these issues *should* be tractable**



# Starting a Process

## λ **Develop these ideas within CMS and in collaboration with**

- ➔ ROOT Team
- ➔ ALICE, ATLAS, LHCb
- ➔ LHC Computing Grid Project
- ➔ IT/DB, IT/API, IT/PDP
- ➔ Eventually clearly one or more RTAGs in SC2, but get started now to seed SC2 discussion

## λ **This process should start with:**

- ➔ a clear exposition of the requirements and primary use-cases, with full consultation of all four LHC experiments
  - without getting bogged down in this process to the complete detriment of SW development
- ➔ specification of architectural principals (modularity, etc.)
- ➔ discussion of development model
  - management, oversight, SW packaging, priorities, "ownership", ..
- ➔ a look at relevant work in CMS and beyond: including IT/DB's experience (Espresso), other experiments (DOOM...), public-domain, etc..



## Concrete First Steps?

- λ **The physical and logical format of the object store should be an LHC (or even HEP?) spec.**
  - ➔ **ROOTIO format may be a concrete area we can start on now**
    - **Document and specify ROOTIO physical and logical format**
    - **Identify missing concepts and iterate with ROOT team,**
      - For example the OID issues, long and short references**
      - Intention is to keep the ROOTIO solutions, possibly with some extensions**
    - **Establish this format and an agreed mechanism for changing it**
      - We must know that we can write/read/interpret this data and schema “forever”**
- λ **With this standard agreed,**
  - ➔ **We establish an important component of ROOT as a guaranteed layer**
  - ➔ **We start to collaborate in a concrete technical way**
  - ➔ **ROOT itself of course interfaces to this format now, so is in a strong position**
  - ➔ **But, one can imagine building now, or in the future, other products**
    - **ROOT++, and/or something quite different**