Data Storage: File Systems and Storage Software

Rainer Schwemmer

DAQ @ LHC Workshop 14.03.2013

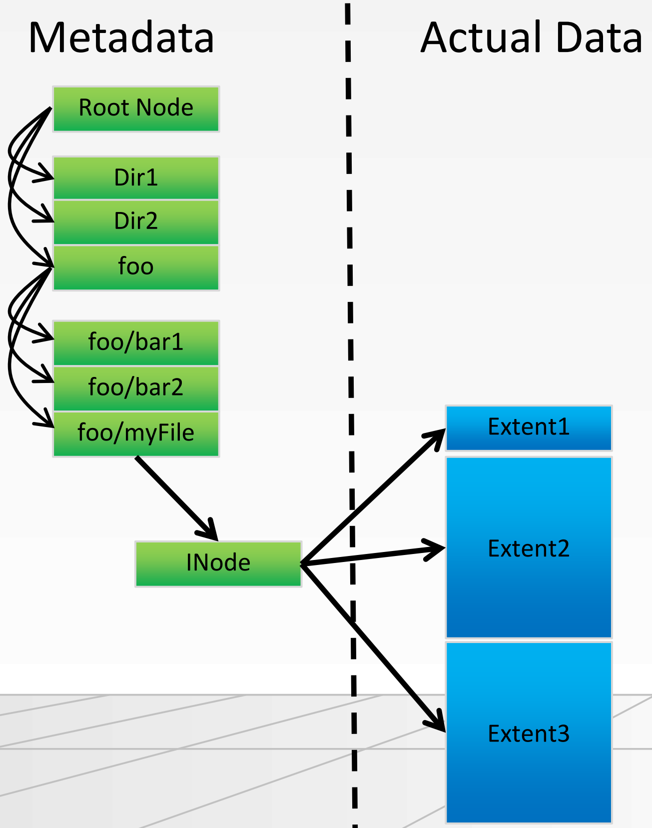
Short Intro to File Systems

- What is a (Computer) File System?
  - ~~The~~ A method for storing and retrieving ~~files~~ data on a hard disk.
  - The file system manages a folder/directory structure, which provides an index to the files, and it defines the syntax used to access them.
  - It is system software that takes commands from the operating system to read and write the disk clusters (groups of sectors).

- What's the difference to a database?
  - It is a database
  - There is usually only one fixed schema though:

Path/Name	Meta Information	Location(s)
/etc/passwd	Size, owner, access, …	0x1234567890

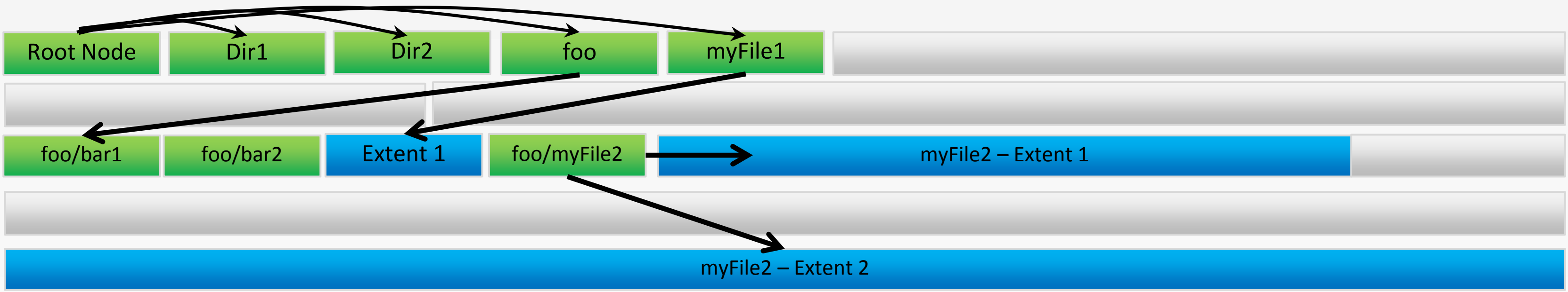
How Does it Work? (conceptual)



- Typically based on B-Trees
  - Optimized search tree
  - Reading/Writing is the timing dominant factor, not compare
  - Self balancing
- Attributes are stored in INodes
- Small files are typically also stored directly inside INodes
- Large files are split up into clusters or, more recently: Extents
- Extents for large files are allocated with rather complex algorithms
  - These are usually the algorithms that make or break the FS
  - The patterns in which data is stored determine how fast it can be read later
  - Most of the tuning parameters of an FS are here
- Metadata: All the other information that is necessary to describe the layout of your data
  - Beware: This can eat a significant amount of your disk space!

How Does it Work? (rough example)

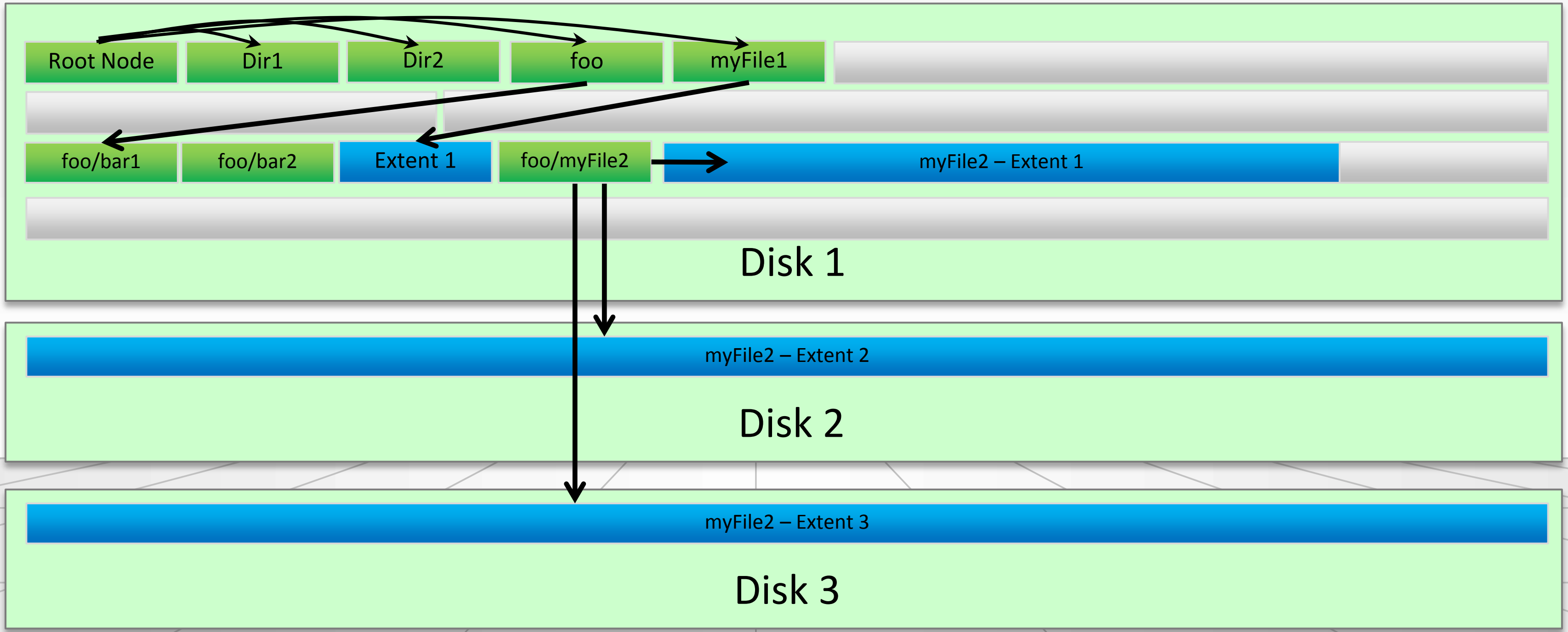
- Usually things are not so neatly organized



- File fragmentation is still a big issue
  - Writing many files at the same time
  - At approximately the same speed
  - For a long time
  - → Will bring your File System to its knees
- Luckily our data is mostly transient
- If you ever plan on having a central log server: Beware!

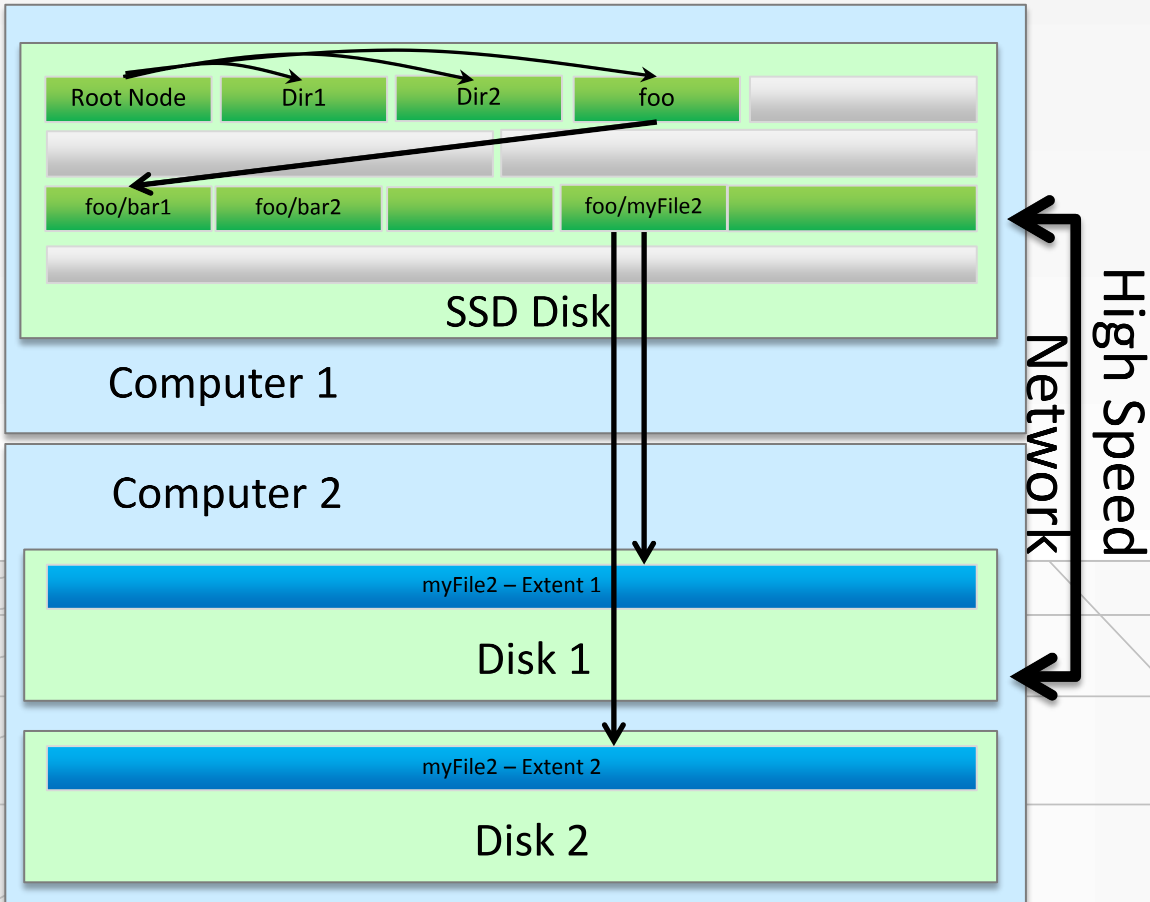
Scaling it up: Distributed File Systems

- File Systems are not limited to single storage units



Scaling it up: Distributed File Systems

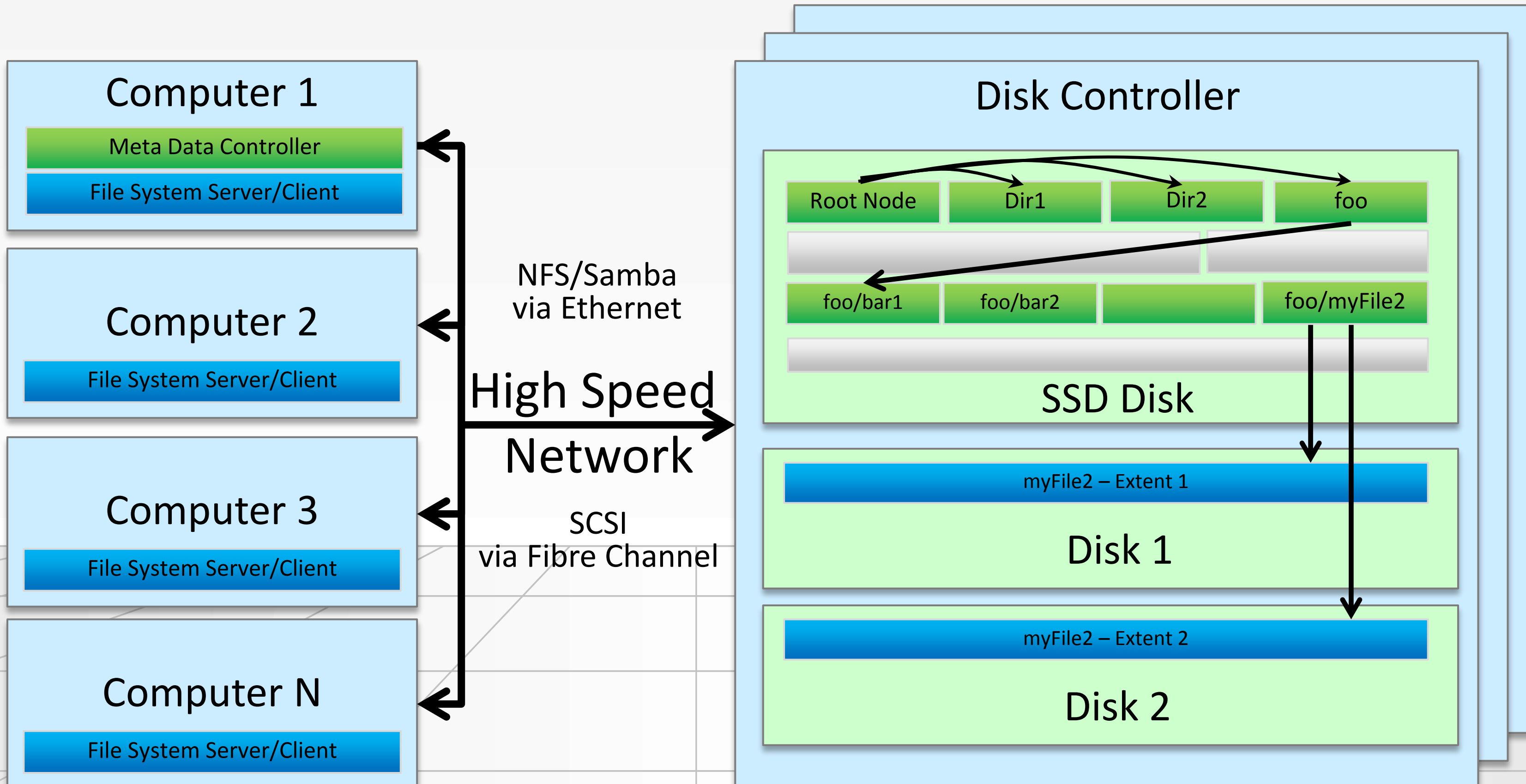
- Not even to the same computer



- Meta Data Operations are the biggest bottleneck
- They are also the most critical ones
- Corrupted Meta Data can potentially destroy all your data
- Data is still there, but without structural information it has no meaning anymore
- You want to be sure that before you write any actual data, the Metadata is on disk and up to date
- Put it on super fast disks
- Put it on a very fast computer

Scaling it up: Distributed File Systems

- While we are at it …



Modern Cluster File Systems

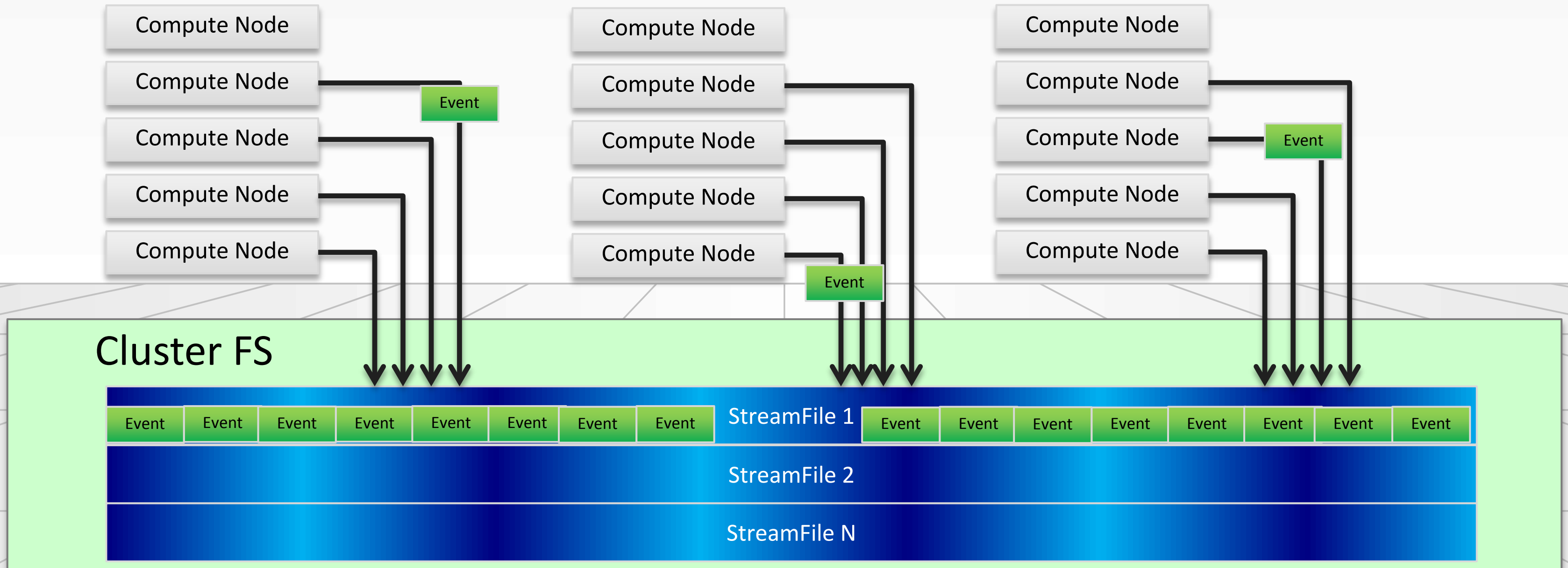
- Load Balancing:
  - By distributing files over multiple disks throughput and IOPS can be improved ALMOST infinitely
- Tiered Storage:
  - Can migrate data between fast (Disk) and slow (Tape) storage depending on usage patterns
- Redundancy, Replication and Fault Tolerance:
  - Can replicate data on the fly to cope with node or disk failures (Not Raid!) or just to improve speed of local access
- De-duplication:
  - Recognizes that files are identical and stores a file only once with multiple pointers to the same data
- Copy on Write / Snapshotting:
  - If a de-duplicated file is modified, a new Extent will be created, covering the modification, while the rest of the file is still shared
  - A certain state of the FS can be locked. All future modifications trigger the Copy on Write mechanism → Versioned history of all files

Unfortunately…

- Still can't write efficiently into a single file from many sources → Does this sound familiar to you?
- In fact: Everything that concerns Metadata is essentially single threaded (cluster wide)
  - Allocating more space to a file is a Metadata operation
  - There are tricks around this but they only last up to a certain scale and usually make life "interesting"
- There are also problems in the underlying storage layer
  - Encapsulation used to be good but it's not cutting it anymore → see modern IP/Ethernet devices
  - The Block Device model (essentially arrays) has become quite a crutch
  - File System and Storage are not really communicating enough with each other
  - Blatantly obvious if you ever had to setup and tune a multi purpose RAID system

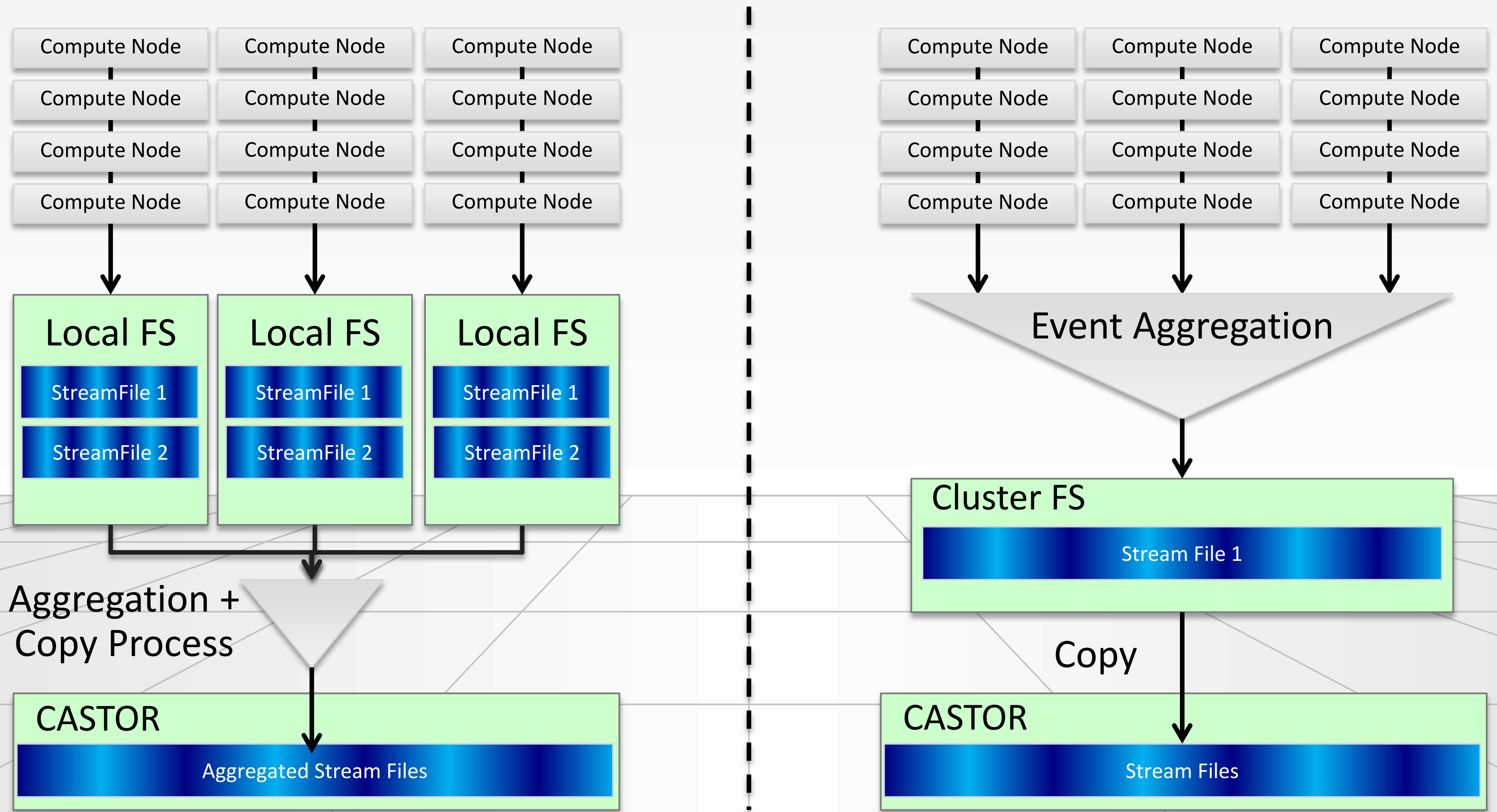
Our Problem

- Many Sources producing fragments of data that logically belong together
  - For a change, I'm not talking about event building here
  - Fully built and accepted events belonging to a Lumi Section / Block / Run
- What we ideally would like to do:

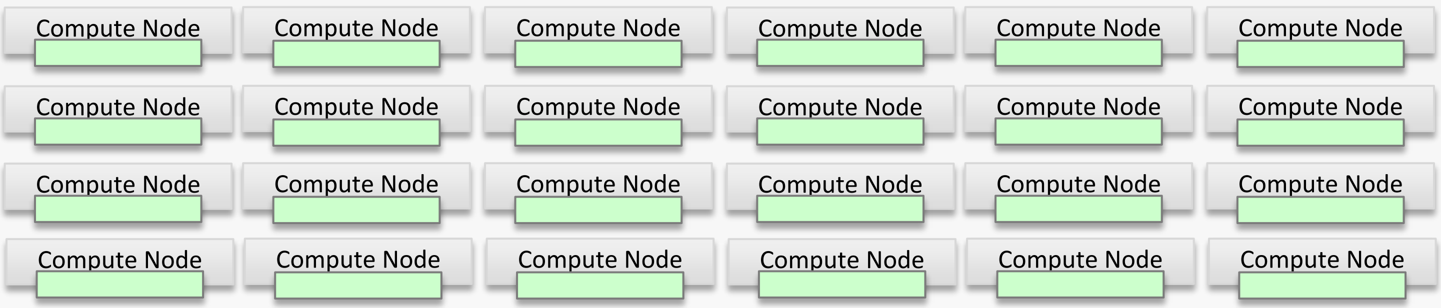


Our Solution

What we are actually doing:

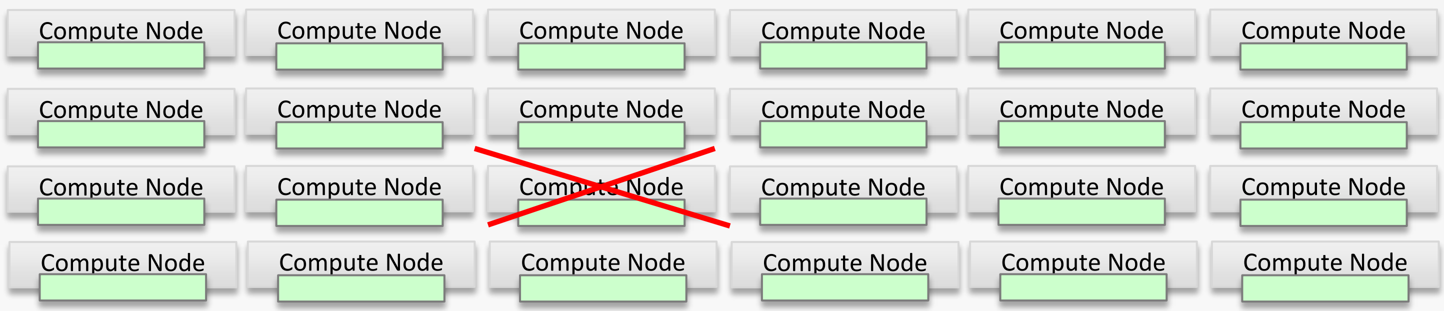


Deferred Processing



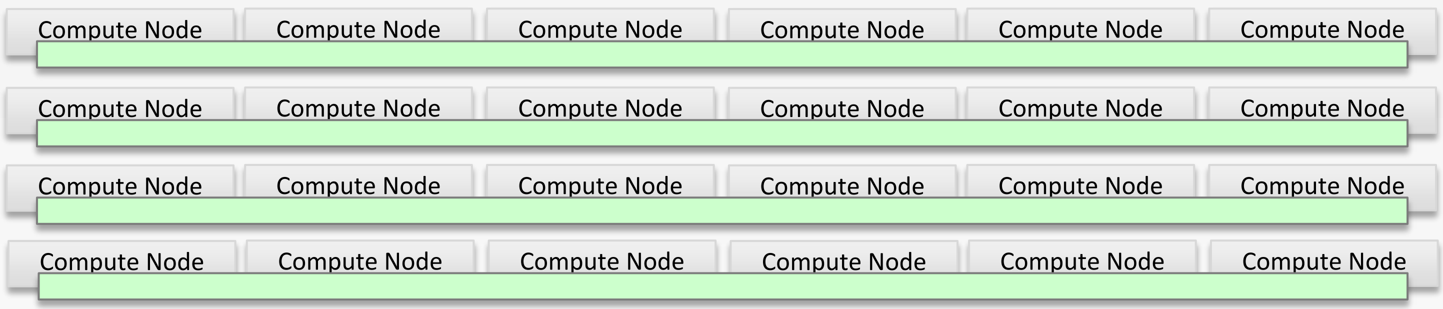
- Farm Nodes usually come with a little bit of local storage
- (A little bit of storage) * (1000+ Machines) = A lot
  - LHCb farm currently has > 1 PB of local storage
- Accelerator duty cycle < 100%
- Store data which we currently can’t process on local FS and process in inter-fill gap/technical stop
  - LHCb: > 30% gain in physics performance

What happens if?



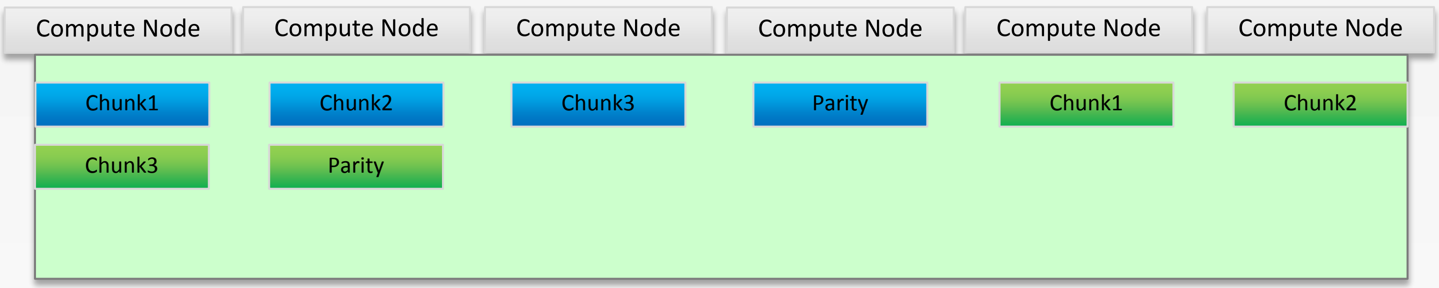
- One machine dies
  - Data is still intact
  - Problem is fixed much later
  - Machine comes back but everybody else is done → annoying
- Local Hard disk dies
  - Data is now gone
  - Do we care?
  - It’s only a small fraction
  - On the other hand we suddenly have a lot of opportunity for broken disks
- Some machines are faster than others

Unified Farm File system



- What happens when one machine dies?
- Local Hard disk dies?
  - → Use replication to cover failures
  - – Replication takes up a lot of space though
  - – Software raid5/6 would be possible but not on 100s of disks
- Some machines are faster than others
  - → Start pulling in files from other machines

Distributed Raid File System



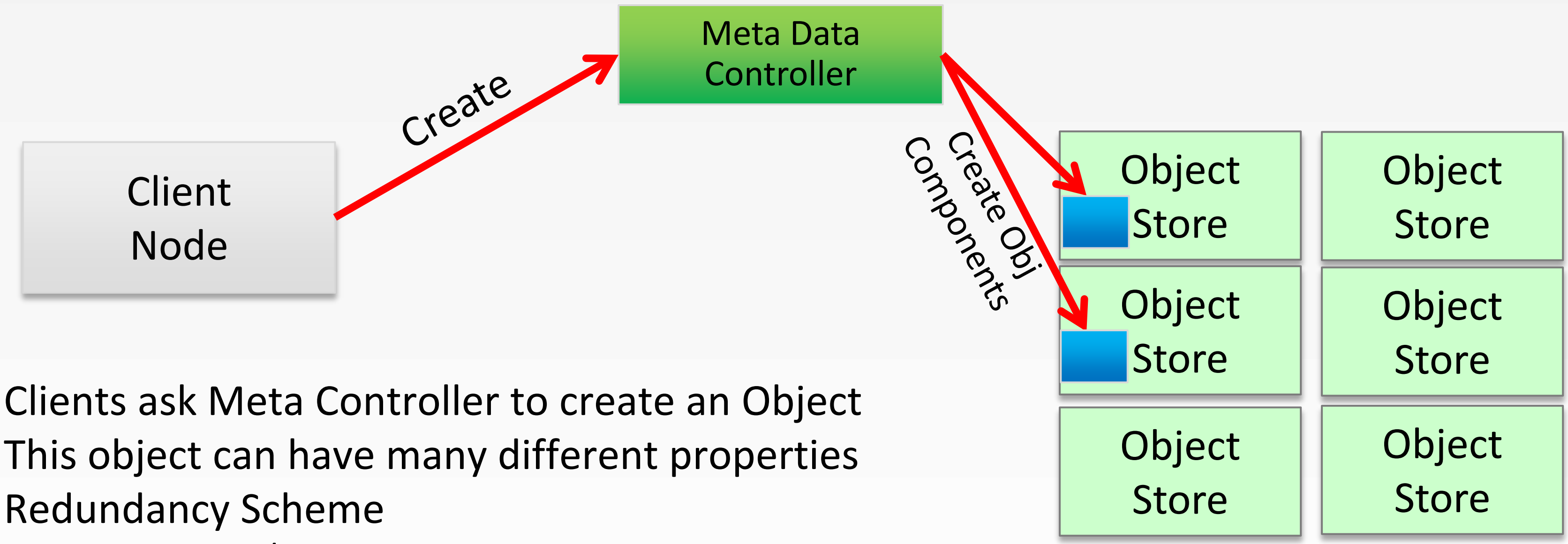
- FUSE to the Rescue
  - File System in User Space
  - Allows to easily write highly specialized File Systems
  - NTFS on Linux, SSHFS, SNMPFS, CVMFS, … is based on FUSE
- Currently under investigation in LHCb
  - File system where File Extents are actually Raid Chunks
  - Can configure how many chunks per File – don't need to stripe over all disks
  - Reading File: Read from neighbour machines in same rack
  - Disk/Machine failure: Rebuild data from parity on the fly while reading

FOR THE FUTURE

Object Storage Devices (OSD)

- Method for solving the Metadata bottleneck
- Currently a bit overhyped
  - We'll have to see what can be salvaged from the hype once it actually spreads a bit more
- Move a lot of the low level metadata operations into the storage device itself
  - Create / Destroy / Allocate
  - Storage Device can be a single Disk
  - Can also be a small set of disks
- A little bit like a File System on top of a set of other, simple File Systems
  - Master FS creates an object on a particular sub FS or set of sub FS
  - Delegates control of this Object to the sub FS

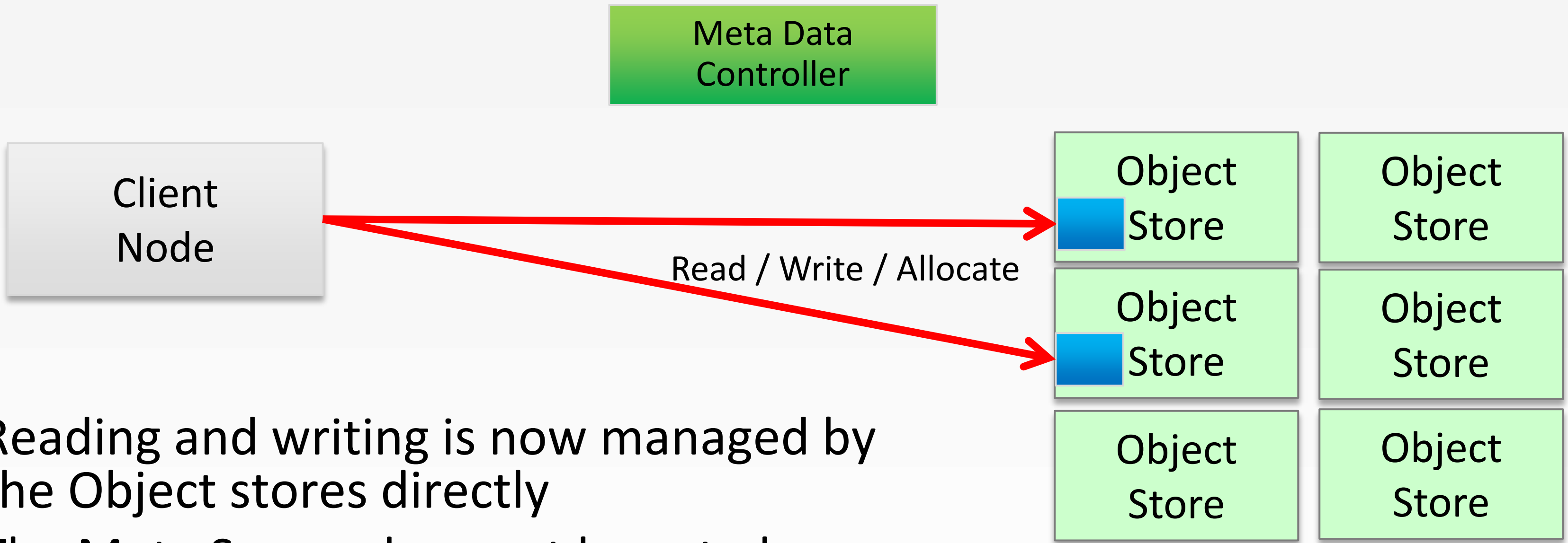
OSD: How does it work?



- Clients ask Meta Controller to create an Object
- This object can have many different properties
- Redundancy Scheme
  - Replication / how many copies
  - RAID like redundancy (how many disks)
- Bandwidth guarantees
  - Assign corresponding amount of disks
- Parallel Object
  - Create Component for every client in a parallel stream
- How will the object be accessed
- Checksums

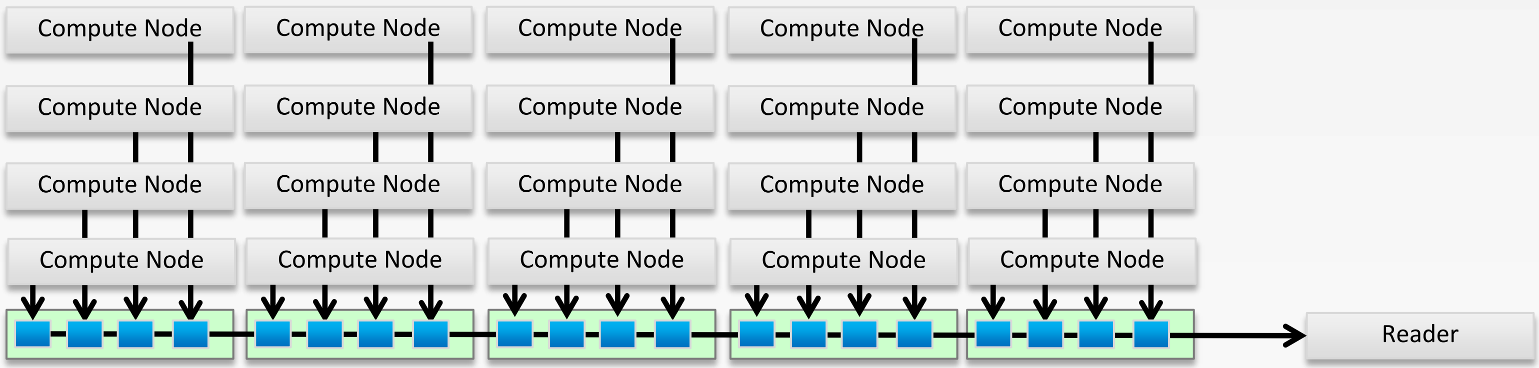
- All the tuning you usually have to set up for a file system can now be done for individual files
  → This is a huge hassle in classical systems

OSD: How does it work?



- Reading and writing is now managed by the Object stores directly
- The Meta Server does not have to be bothered anymore
- Each Object Store has its own local Meta Controller
- If a client wants to know the status of an Object, the Meta Controller will retrieve a summary from the Sub-Stores

How will it benefit us?



- DAQ example
- An Object is created for parallel writing
- Every Compute node gets its own slice of the DAQ object on a nearby storage element (can even be local)
- No need to coordinate with a meta controller besides the creation of the original object
- Object has read access defined as concatenation with arbitrary ordering
- Client (I.e. CASTOR copy process) reads the data inside the object as a continuous file from the different storage elements

Other Benefits

- Fixes a lot of the problems that are currently caused by the block device model
  - Don’t have to rebuild a lot of free space on a RAID disk anymore
  - Tuning can be adjusted to the underlying storage and individually, depending on file access patterns
- More inherent redundancy in case of individual component failure
  - pNFS is actually made for this kind of storage model
  - Built-in High Availability mechanics
- Data can be moved closer to a client in the background if necessary
- File Systems that are currently or soon going to support OSDs:
  - PanFS (PANASAS) → Appliance based on OSD
  - Lustre: Soon™ and/but OSS
  - pNFS
  - glusterFS

Closing Remarks & Things to keep in mind

- Expect to see a lot more NFS/SAMBA like File Systems in the future
  - SANs are disappearing and are being replace with specialized NAS (Network Attached Storage) appliances
  - Not only because FC is essentially an extortion racket
  - High Speed Ethernet can easily keep up with FC nowadays
  - NAS boxes are pre-tuned to the applications that you want to run on them → less work for users
- The OSD approach will probably make storage much more distributed
  - Eventually we might not even have to buy dedicated storage anymore
  - Could use local disks in our filter farms instead

Thanks

Ulrich Fuchs, Roberto Divia

Wainer Vandelli

Gerry Bauer