SNAPSHOTS FOR IBRIX – A HIGHLY DISTRIBUTED SEGMENTED PARALLEL FS

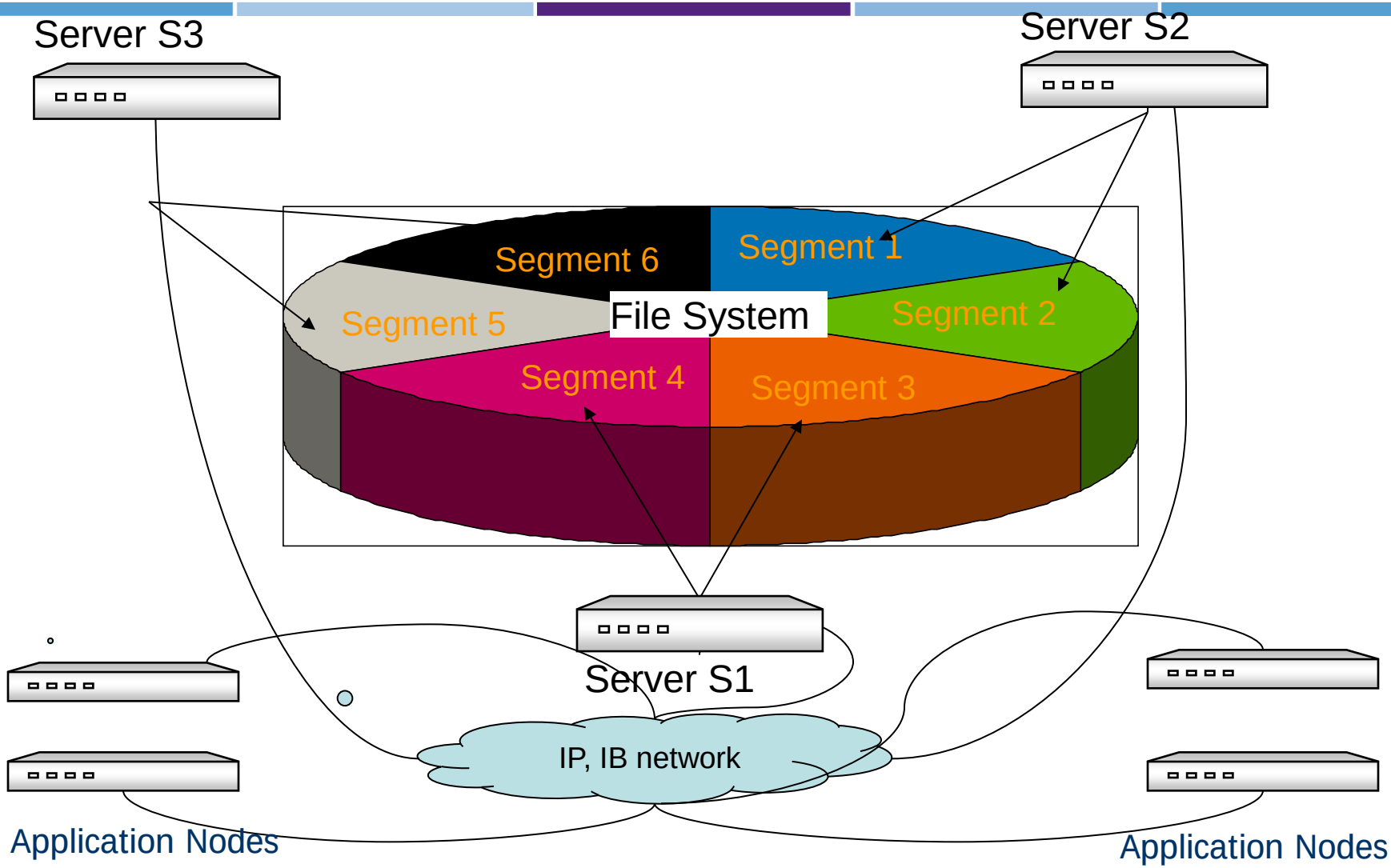
Boris Zuckerman
Hewlett-Packard Co.

Objectives

- Review the challenges of implementing snapshots in highly distributed FS environment
- Expose fundamentals of highly distributed segmented parallel file system architecture
- Show key points of the snapshot design:
  - Snap Identities - dynamically inheritable attributes
  - Preserve name components in snapshots
  - Avoid large scale data flushes at snap time
  - Support for instantaneous restores

How to scale high?

Divide and Distribute



Scalable architecture

- Splits physical and logical space into segments
- Assigns control over segments to segment servers
- Segment servers are entirely responsible for file (inode) and block allocation **within the boundaries of the individual segments**
- segment servers coordinate caches and activities, associated with objects **they maintain**
- Files and directories are distributed though the sets of segments according to placement rules

Entry and Destination Servers

- Hosts that receive FS (VFS/IFS) requests from applications, NFS, SMB and other servers are called Entry Servers (**ES**)
- Some requests may be handled locally because objects are local or because ES has trusted objects in cache
- If objects are remote, ES makes RPC requests to Destination Servers (**DS**)

Sessions and delegations

- Objects returned by DS to ES are delegated: have associated rights to keep them in ES cache
- There are shared and exclusive rights
- Rights/delegations can be revoked
- ES must not stall revocation requests
- ES is known to be alive, if it maintains network heartbeats – maintains session
- ES-DS session starts with ES handshake offer
- If ES is disconnected, DS considers session broken and may release all delegations

Challenges of snapshot implementation

- Limited scope of individual FS operations
  - Snapshots by their nature have to affect the state of multiple, perhaps millions of objects
- Time of snap command should be small:
  - No detection of all objects
  - No freezing
  - No flushing
- Time of restore should be small and should not require significant additional storage

Snapshot design goals

- To be applied at any level and selecting any subsets of objects
- To be applied as frequently, as needed
- To withstand disconnects and failures
- To support quick logical rollback and cleanup in the background
- To provide ability to query all historical content of directories (versioning)

How to identify snapshots?

Snapshot Time Mark as Snap ID

- The most direct way to identify snapshots is the time when snapshot is requested; we call it Snapshot Time Mark (STM)
- STM can be recorded on any object at any place of the name space
- STM is used to mark when objects are created
- STMs are used to decide if object can be deleted or preserved in the past
- STM can be used to query preserved content (@STMn)

Static inheritance

- Before snapshots all properties of Ibrix objects were known and maintained by DS nodes
- ES could propagate some properties from parent directories to direct descendants during create or mkdir calls, but inherited properties were recorded statically
- Changes at higher level did not affect previously created descendants

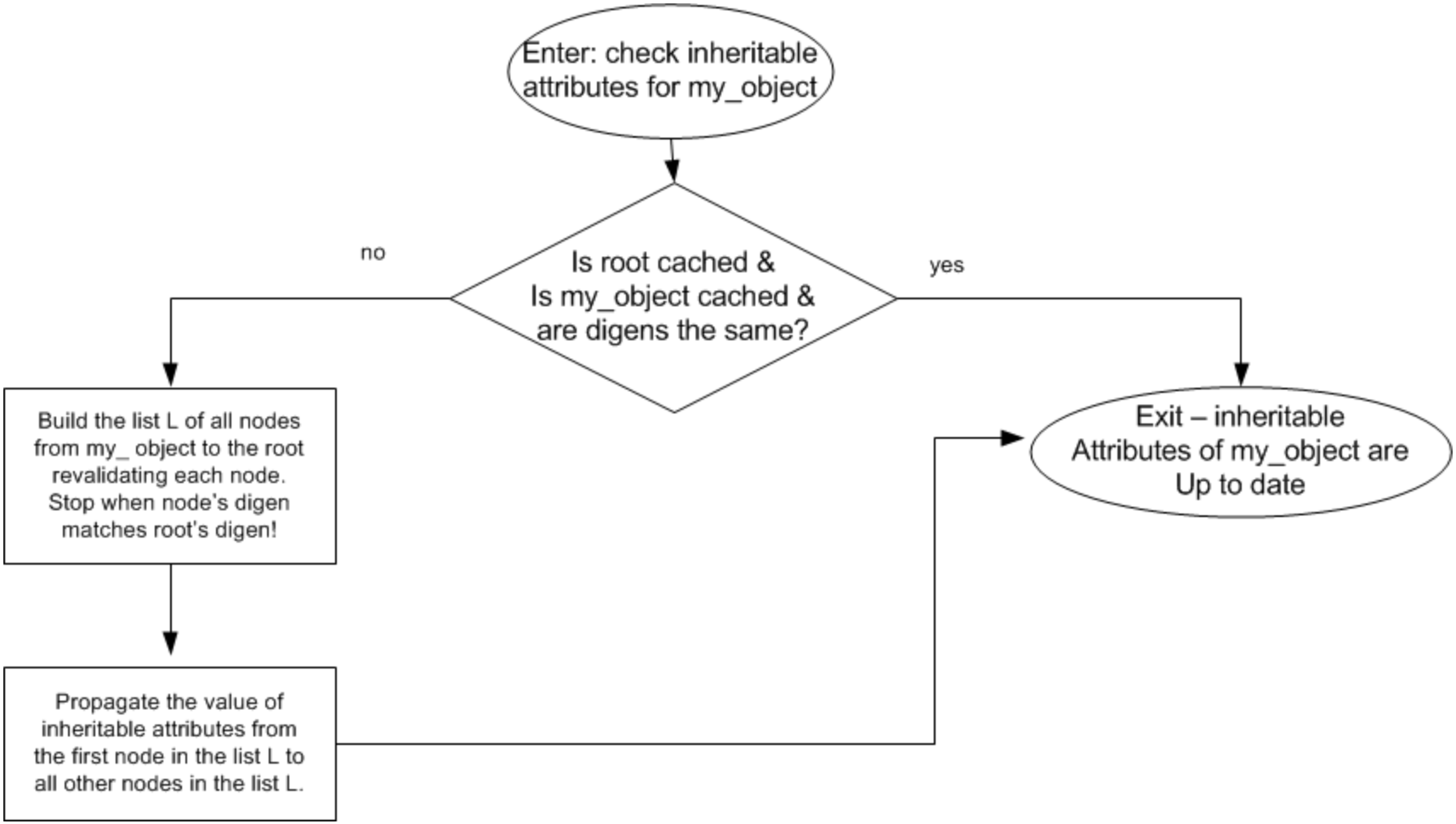
Dynamically Inheritable STM

- To rapidly propagate STMs we designed a mechanism of dynamic inheritance
- Dynamically inheritable attributes are calculated or revalidated by ES
- ES is responsible for propagating these attributes down the hierarchical lines at ‘run-time’ and revalidating them when it becomes necessary
- When DS actions depend on values of such attributes ES passes them to DS in RPC calls

Dynamic Inheritance: Efficiency

- The main goal of the algorithm below is to avoid traversing an entire chain of nodes for every check even when some nodes in the chain are not protected by delegations
- It is based on Dynamically Inherited Generation (digen) that we keep in the **in-core** inode
- ES copies digen from a parent of a child inode
- digen is monotonically incremented only on the root of the FS (DS side) and then propagated to other nodes during validation of dynamically inheritable attributes

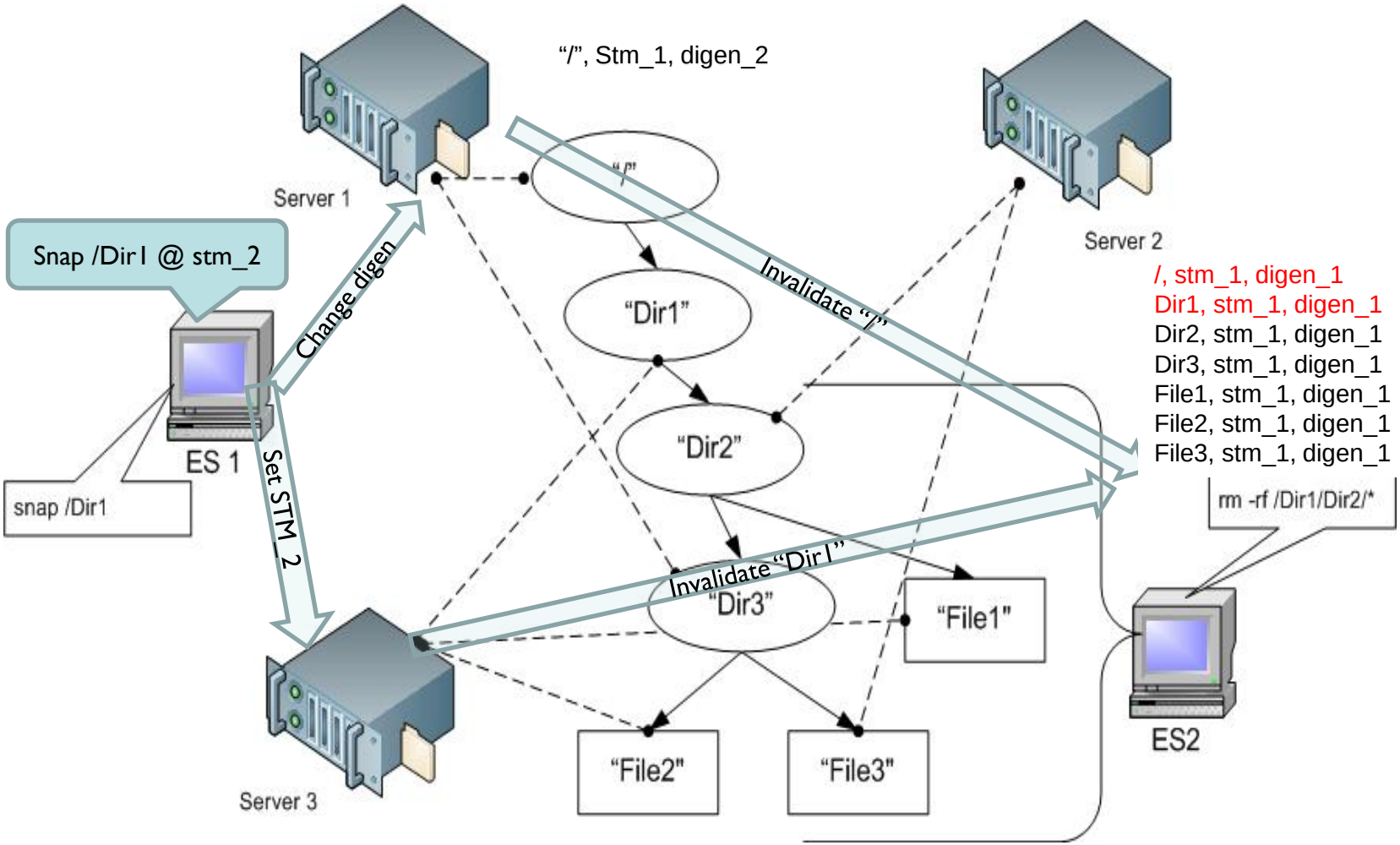
Propagation of Inheritable Attributes



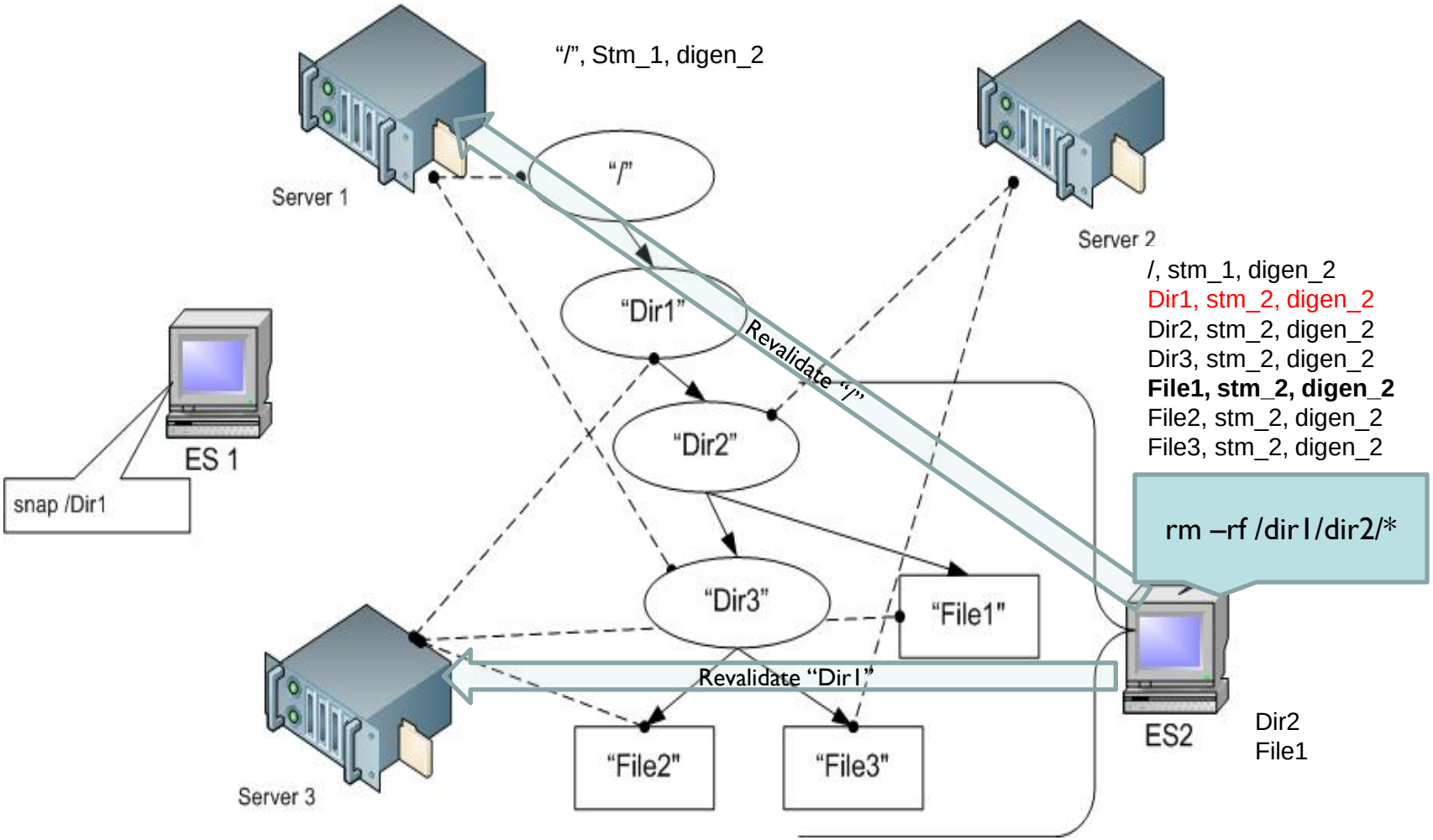
Key Points of the Algorithm

- The algorithm does not check values of inheritable attributes for all nodes of the hierarchy. The processing stops when digen of any node matches the digen of the root.
- STM propagation is based on the fact that time moves in one direction and STM for snapshots grow monotonically. A later request to preserve an object overwrites a previous request - the **effective** update **STM** is always the largest requested STM on the chain of all predecessors.

Example of propagating STM



Example of propagating STM



Name Space and Data Preservation

- Snapshot birth and death STM-s, were added to each dentry stored in directory files
- Birth STM is set to the effective STM dynamically inherited from the object's predecessors when a name is created
- When name is removed (unlinked or rename) and the current effective STM matches the birth STM, we remove the dentry from the directory
- If the effective STM is different from the birth STM the dentry is marked as "killed at effective STM" by setting the death STM (DTM)

Name Space and Data Preservation

- Modifications of files have no effect on names
- Modifications of files may be treated as snap events or non-snap event
- Coordinate snap requests with the stable state of such files. Microsoft's VSS architecture addresses this on software level

Identifiable write caches

- ES clients maintain write (dirty) caches of data
- It's very undesirable and impractical to force a flush of all caches **before** setting a snap identity
- Every time a write request is directed to cache we validate STM
- If the STM changes, we flush existing data cache into a "previous state" and start caching with the new STM

How to look into past?

- Navigate into the preserved state by explicitly specifying QSTM (ls /myRoot@QSTM)
- Record a pointer to the snap in a symlink

./.snap

Snap_09_16_2013_11_27_14 ->../@1375360549

- Inherit .snap to appropriate descendants

How to restore?

- Copy over?
  - impractical - we may have to copy a huge amount of data that is already exists in our FS
  - we may simply not have enough space
  - does not address discarding of files created after the point of restore
- Wait for a special tool to remove newly created objects and revive killed?
  - It can take a long time…

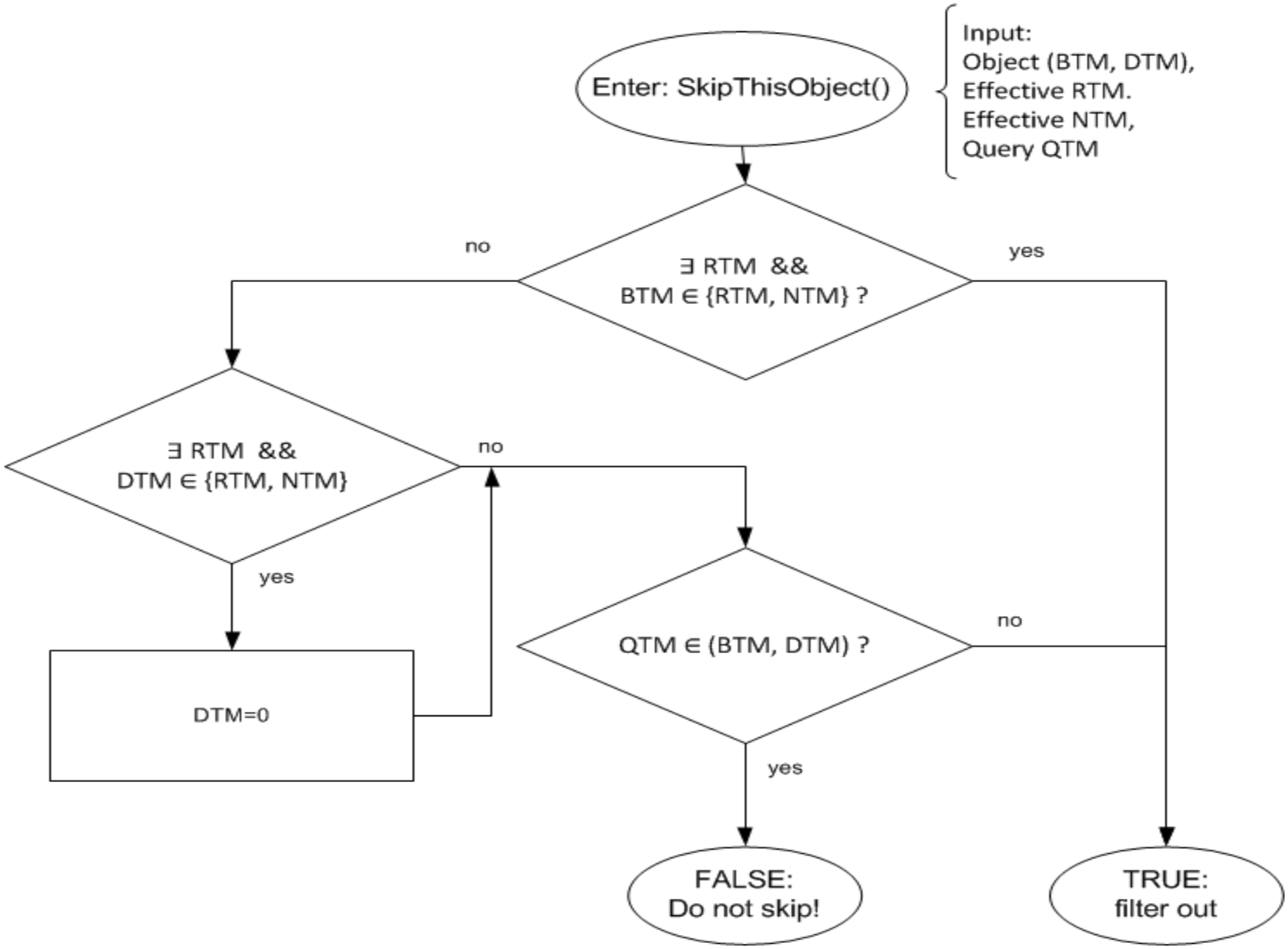
“Instant” Restore

- Birth TM and Death TM are used for filtering out objects in query processing
- We can use them to logically filter out newer updates, if restore was requested
- recode restore information in the name space and use dynamic inheritance to propagate it to descendants
  - Now Time Mark (NTM)
  - Restore_to Time Mark (RTM)

Filtering with NTM and RTM

- file system kernel can filter out all objects that were created after RTM and before NTM where BTM $\in$ {RTM,NTM} and “revive” objects with DTM in the same interval
- effective STM is being calculated combining values of STMs and RTM

Filtering out with requested roll back



Multiple Restore Requests

- Multiple restore requests could be recorded on several nested levels or on the same object
- We filter out entries applying each restore region independently.

```
skip = FALSE;
    for i in all restore regions do {
    skip = SkipThisObject(Object, NTMi, RTMi) ;
    if (skip) break;
}
```

Overlapping requests may be collapsed

- If 2 overlapping requests {NTM1, RTM1} and {NTM2, RTM2} are applied to the same object and RTM2 < NTM1, we can combine them representing them as one {NTM1, MIN(RTM1, RTM2)} request
- If we have several non-overlapping requests recorded and then we get a request overlapping both of them, we can collapse several requests into one

- There is only one applicable value of STM per node
- multiple regions of snapshot restores generally have to be recorded individually. A reasonable implementation has to limit a number of slots available for this purpose
- A highly parallelized background process collects discarded objects and history

Summary

- Reviewed architectural basis for scalability of the segmented FS
- Showed how snapshots are different from other commands
- Explained how dynamic inheritance can be used to propagate STM, RTM and NTM through nodes of name space
- Show how to avoid flushes of dirty caches at time of snap requests

Thank you!

Questions?