Database Techniques to Manage a Distributed POSIX Namespace and Associated Metadata

David Boomer
IBM

Introduction

- Dave Boomer
  - boomerd@us.ibm.com
  - Information Architect for the High Performance Storage System (HPSS) project (www.hpss-collaboration.org)
  - DB2 Specialist

Agenda

- The Challenge
- Utilizing relational database technology
  - POSIX Namespace
  - User Defined Attributes
- Prototype
  - Configuration
  - Results

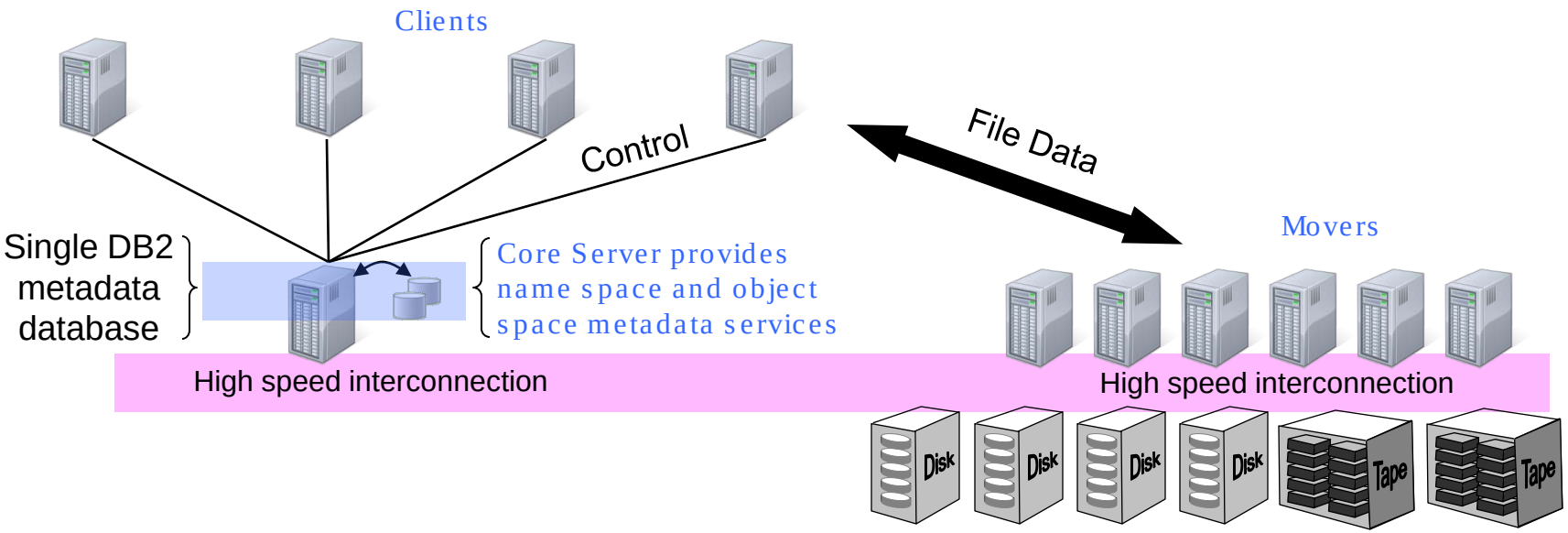
HPSS program goals for capacity and performance

	HPSS V7 2009	HPSS V7.4 2010	HPSS V8 2011	HPSS V8.x 2012+
Storage capacity	10^16 10s of PB	10^17 100 PB	10^17 100s of PB	**10^18 1 Exabyte**
Number of files	10^8 100s of Millions	10^9 1 Billion	10^10 10s of B	**10^12 Trillions**
Peak File creates per second	10^2 Hundreds	10^3 1 Thousand	10^4 10s of K	**10^4 10s of K**
Sustained file creates per year	10^8 100s of M	10^9 1s of B	10^10 10s of B	**10^12 1 Trillion**
Metadata size	10^11 100s of GB	10^12 1s of TB	10^13 10s of TB	**10^15 1s of PB**

Single metadata engine architecture

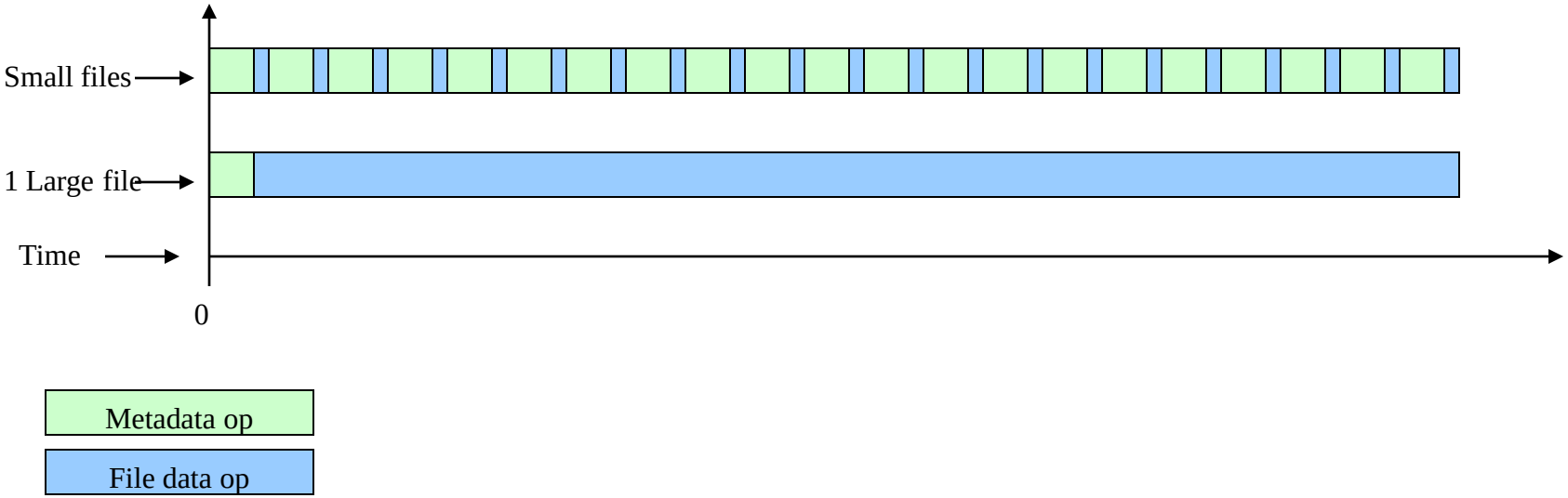
- HPSS Versions 1 thru 7 have a single metadata server
- This is sufficient for all HPSS installations to date
- HPSS mission to be ready for the next generation of data-intensive computing

- Therefore, the next step in HPSS capability will be to replace the single metadata server with a cluster of metadata servers



The Challenge – Small Files and Metadata

- Single Core Server architecture fine for "large" files…
  - Data movement for large files is the significant component of file ingest transaction
- However…
  - Metadata overhead significant percentage of complete file ingest transaction for small files
  - Adds database transaction overhead to core server

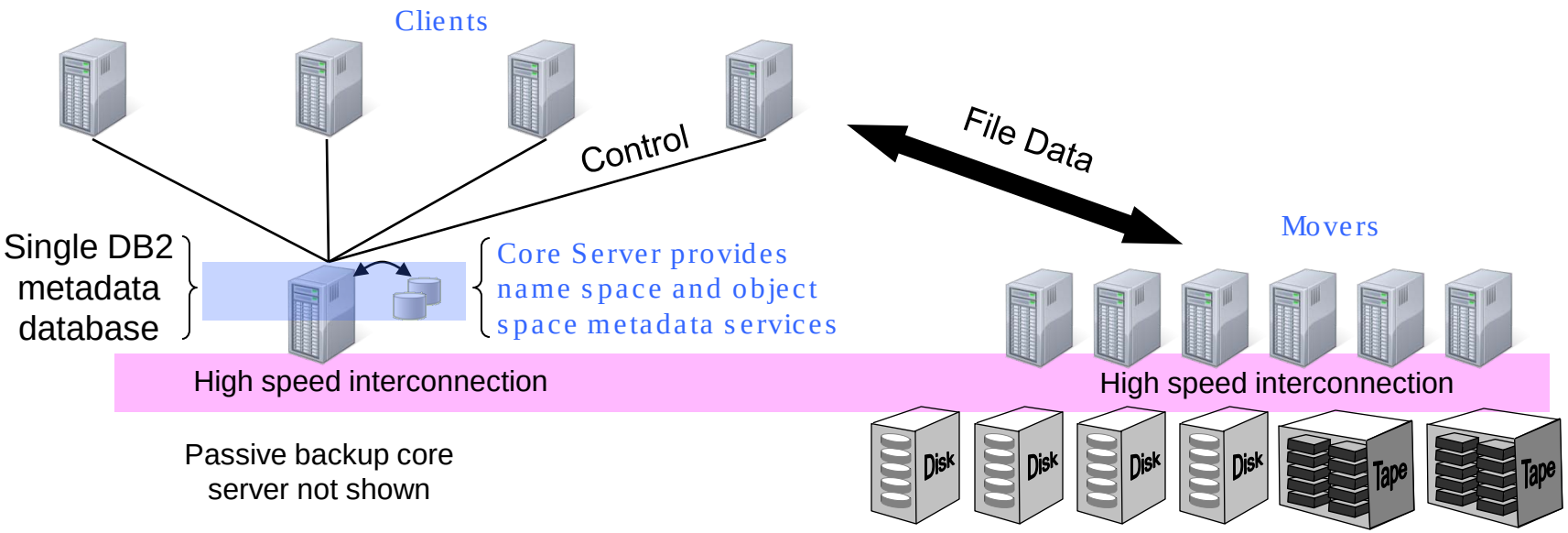


Agenda

- The Challenge
- Utilizing relational database technology
  - POSIX Namespace
  - User Defined Attributes
- Prototype
  - Configuration
  - Results

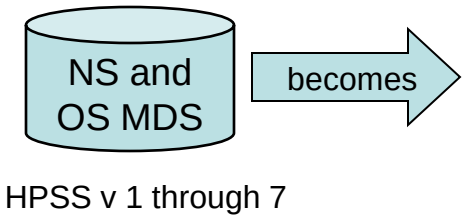
Requirements for HPSS v8 metadata

- Overcome Single Core Server in a ***balanced, "self-leveling"*** manner
- Provide transaction scalability beyond a single metadata server
- Provide distributed metadata management
- Provide distributed POSIX namespace
- Enable virtually unlimited growth in number of files
- Utilize commodity hardware

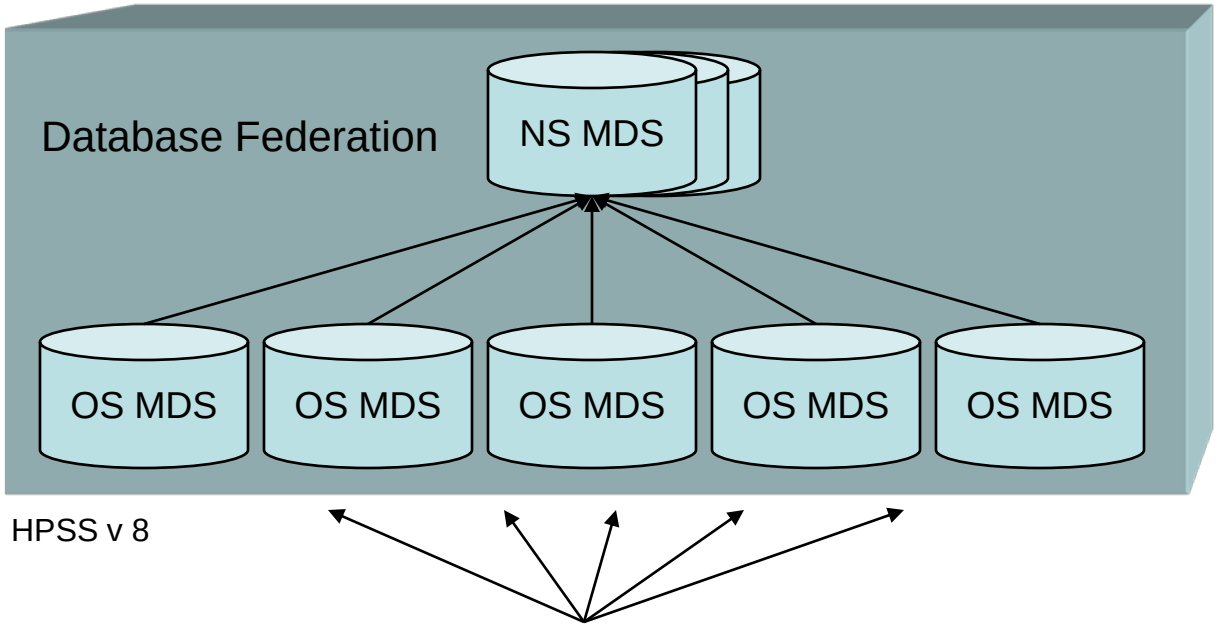


For HPSS v8, the metadata becomes a federation of databases…

- Removes Single system bottleneck
- Extremely scalable across all components
- Metadata and database transactions distributed across cluster
- NS & OS layers independently expandable

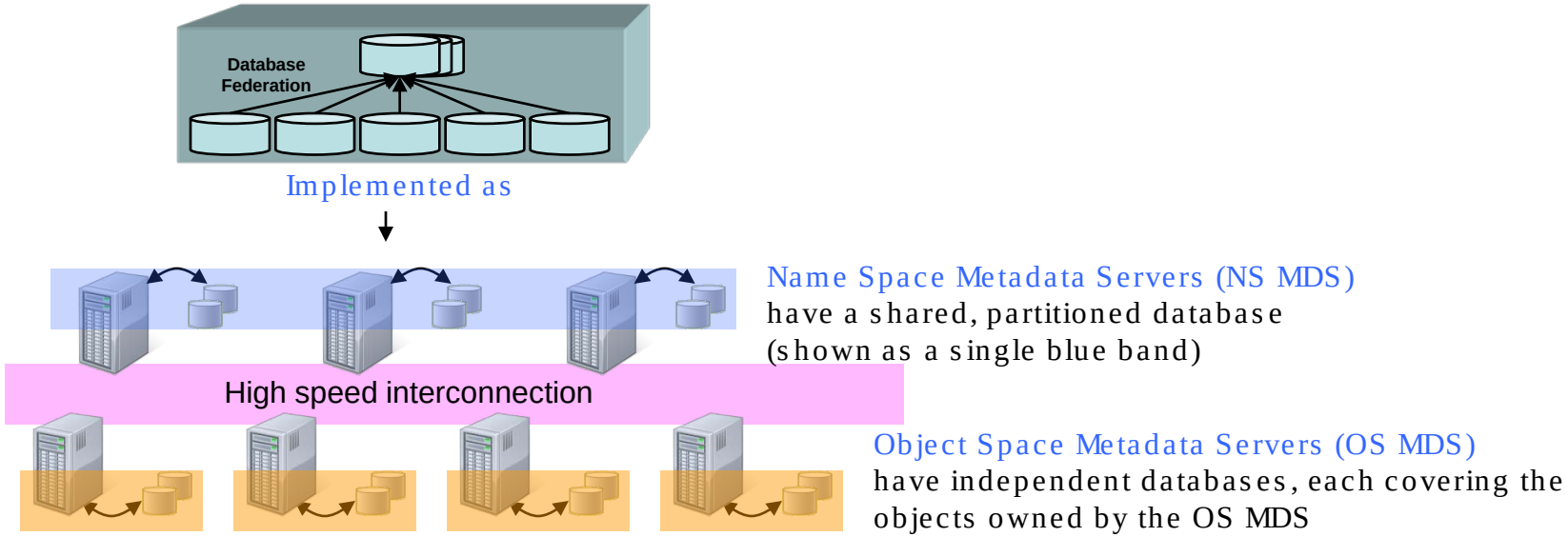


HPSS v 1 through 7



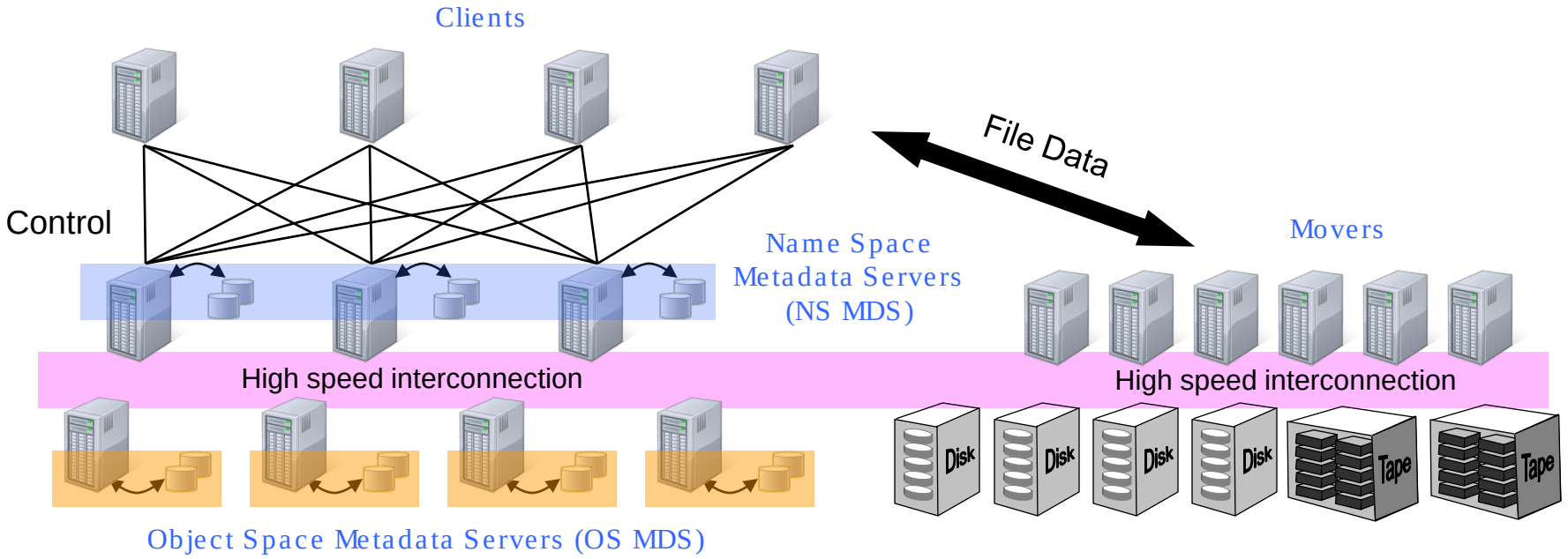
… and the single metadata server becomes a cluster of metadata servers

- A cluster of Name Space and Object Storage Metadata Servers replaces the single "core server" metadata server used by HPSS from v1 through v7
- Entire name space must be visible from any name server, implemented as a single database that is visible across all the name space metadata servers
- On the other hand, each object server is responsible only for managing the data objects assigned to it, so each object space metadata server is assigned a database that is not shared.
- And of course a passive backup architecture will provide a replacement to quickly take over for any failed components.



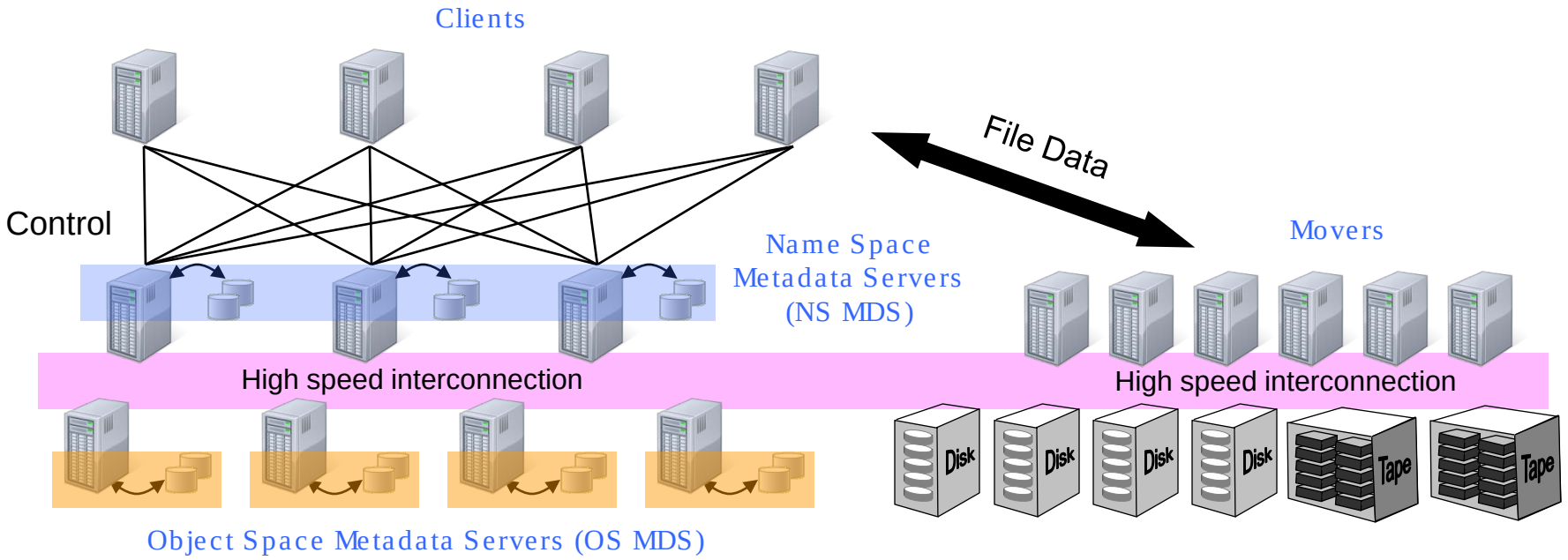
Distributing POSIX Namespace metadata across multiple systems

- Support POSIX rules
  - Unique object names within a directory
- Balanced and "self-leveling"
- Data co-location
- Limit transaction to single partition or database (no distributed transactions)
- File path management
- Utilizing off the shelf database technology
- Minimizing network hops



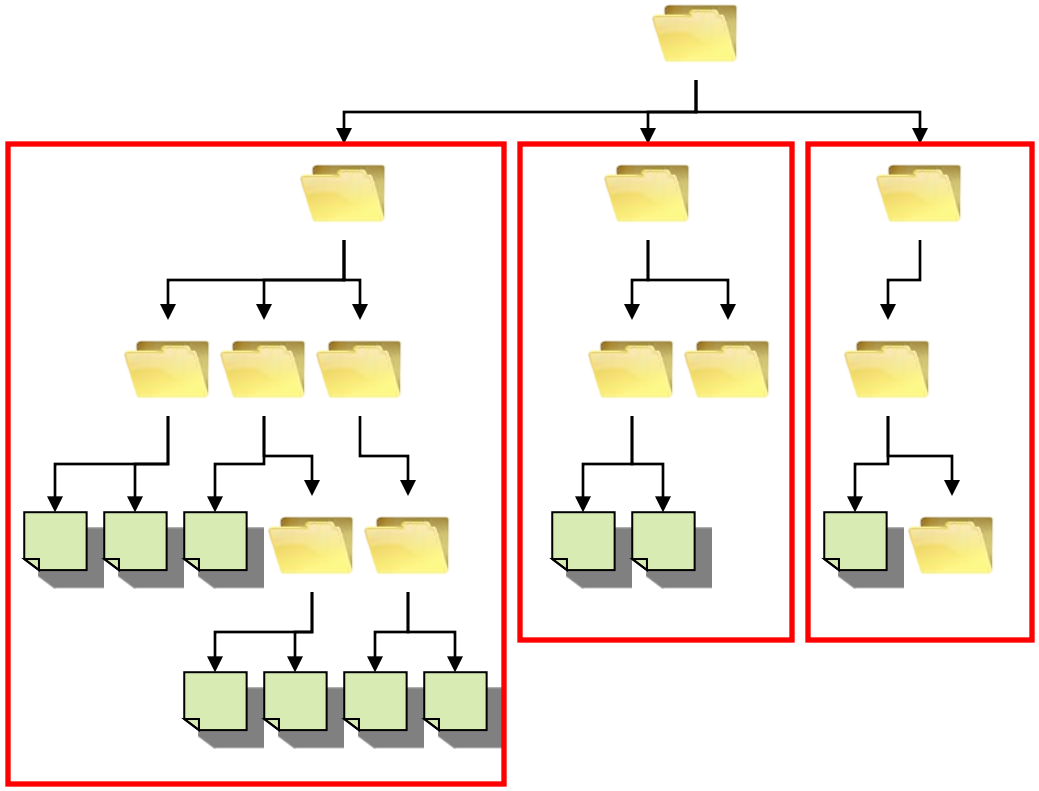
Managing Petabyte Class Relational Datastores

- Maintenance: Backups, Statistics, Reorgs
- Minimize transaction/logging overhead per file
- Reducing size of metadata per file
- Integrating user supplied content knowledge
- Move towards an Object-based Cluster File System architecture



Similar challenge faced by other distributed file systems

- Techniques:
  - Static Subtree Partitioning (HPSS V7) [3]
  - Dynamic Subtree Partitioning [2]
  - Name/Path Hashing [1,2,3]
  - Lazy Hybrid (i.e., Name hashing w/lazy update) [1]
  - Hierarchical Bloom Filter Arrays [4]
  - Ceph [5]
  - GIGA+ [6]
- We will hash on parent directory ID and file name
  - Results in even distribution
  - DB2 major enabler



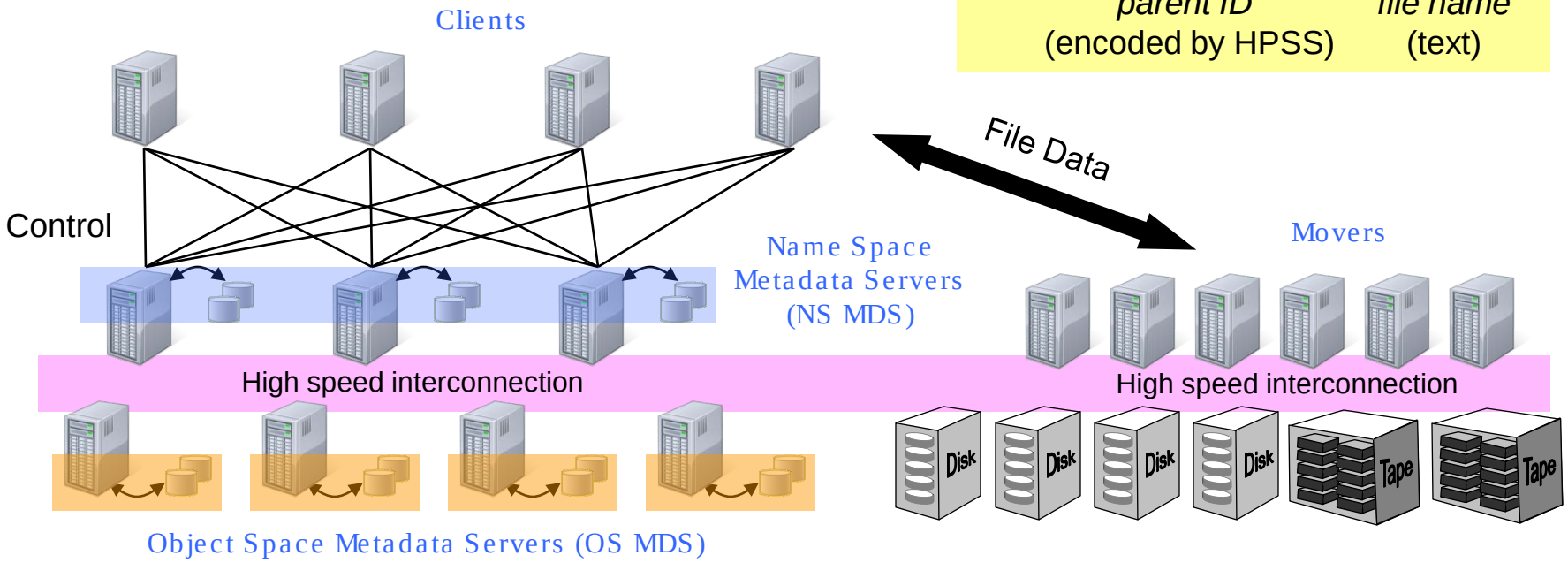
See last page of this presentation for reference citations on hashing methods. References are in gray brackets [n]

Which Name Space Metadata Server should a client use?

- Clients will direct work to the appropriate metadata server based on a hash value.
- Hash value is created from the *file name* and *parent ID**.
- Should yield a reasonably uniform distribution of MDS accesses without hot spots.
- Data is transferred between client and HPSS mover (supporting separation of control and data).

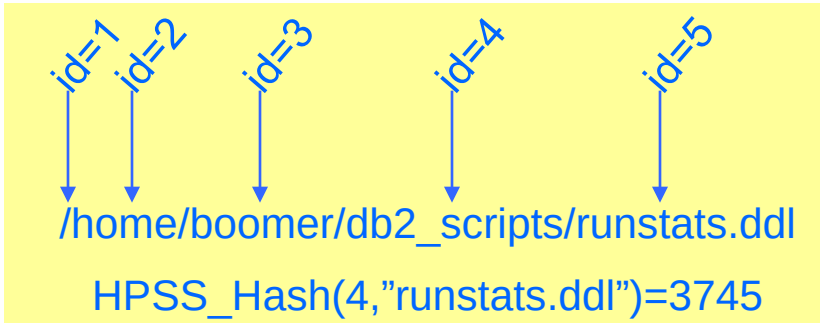
* `/dir1/dir2/dir3/myfile`

parent ID (encoded by HPSS) | *file name* (text)



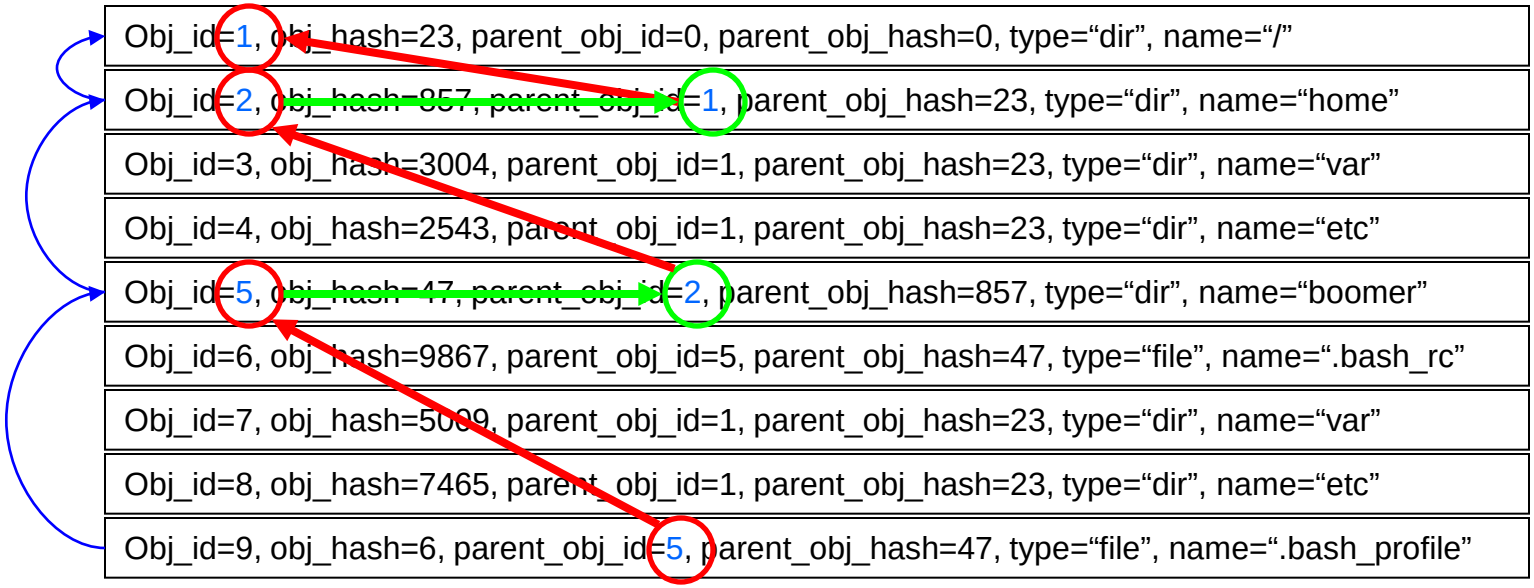
Distributing the POSIX namespace

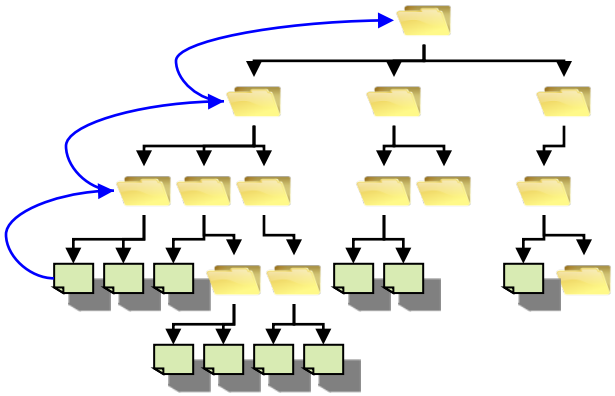
- Hashing the combination of:
  - Parent directory identifier
    - Unique across the namespace
    - Does not change
  - Object name



- Why?
  - Database enforces the "unique name within directory" POSIX constraint *within a database partition* (very scalable)
    - 2 files with same name in same directory generate same hash value
    - Will be inserted into the same database partition
  - DB2 will *hash* our calculated hash value to determine partition
  - Namespace objects within a directory are balanced across cluster
    - Metadata balance
    - Database transaction balance
- Based on DB2 partitioning feature using a Linux cluster

Namespace Table





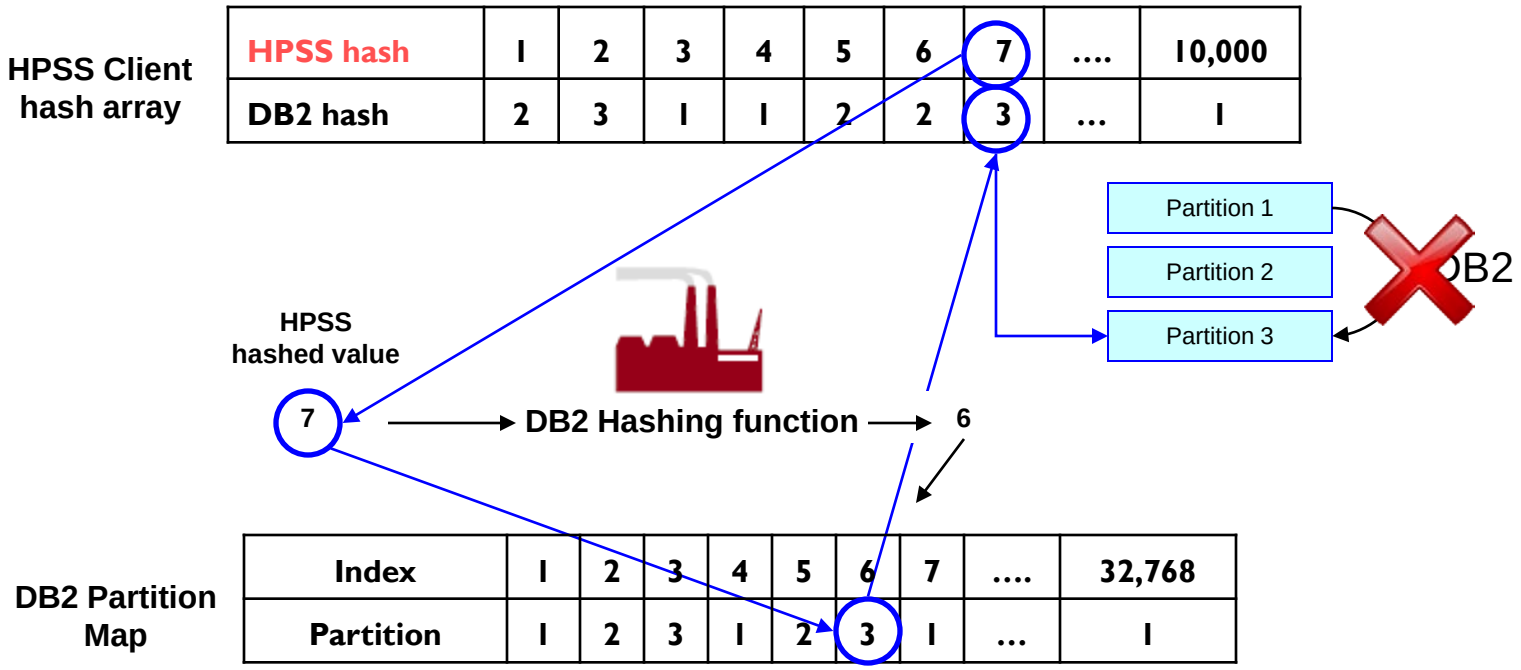
/home/boomer/.bash_profile

File path stored as recursive data relationship

Partitioned Databases

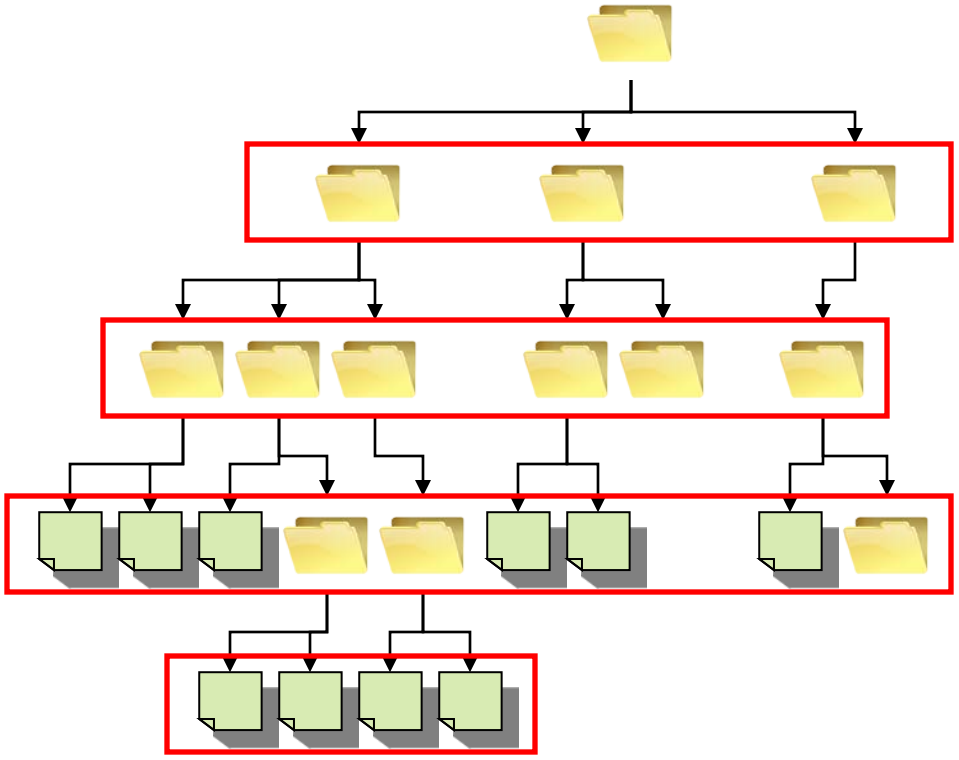
How the HPSS Client determines which NS to use

- Hashed value based on Parent ID and object name… hash(parent_id & name)=**object_ns_hash**
  - **1 <= hash output <= 10,000**
- Create table nsobject (obj_id bigint, **obj_ns_hash** smallint….) Distribute by hash (**obj_ns_hash**)
- Create unique constraint (parent_obj_id, name, **obj_ns_hash**)
- Client communicates directly with owning partition, reducing network hops



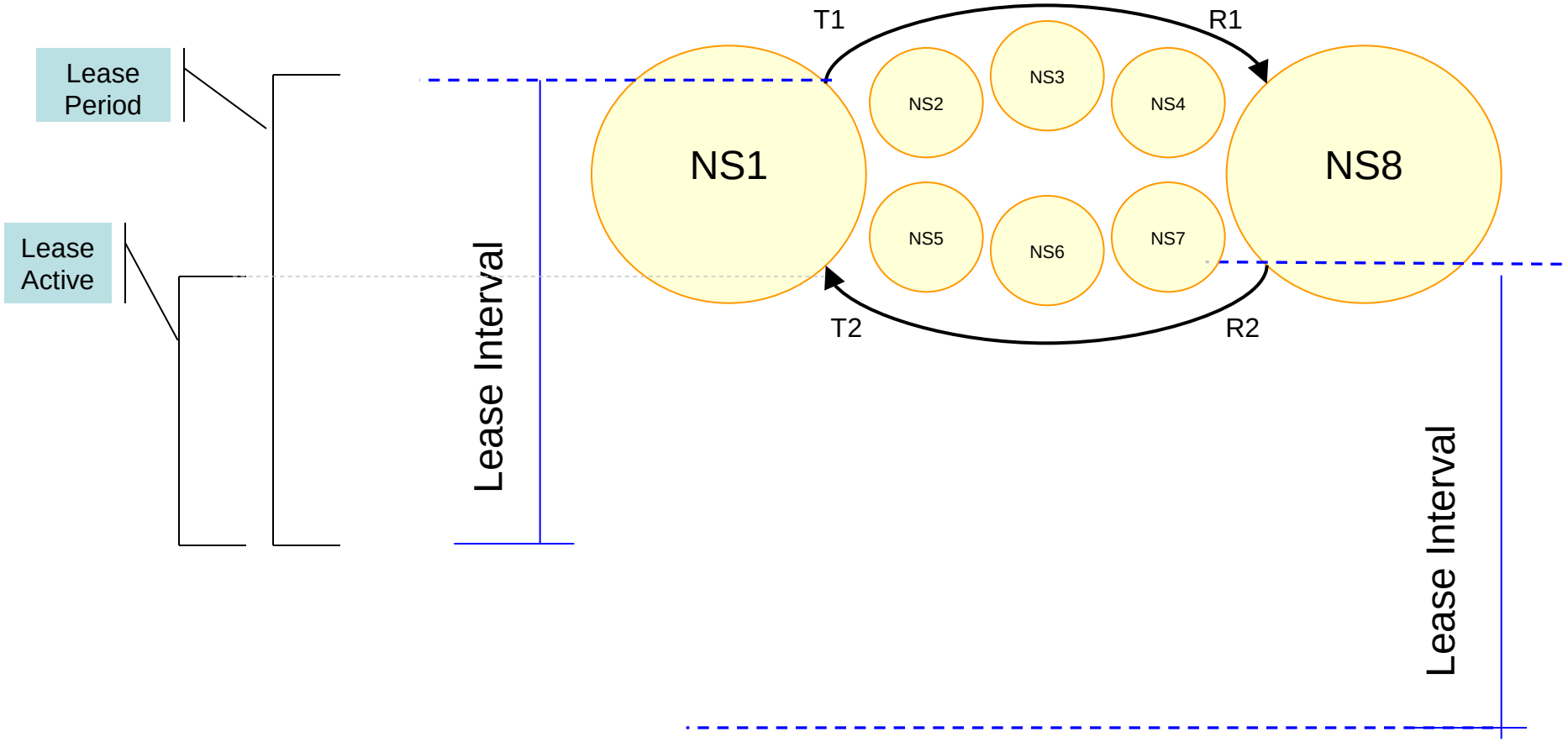
POSIX Namespace "Rehash"

- Hashing technique ensures
  - Each layer is distributed evenly
  - Reduces network "hops"
  - Supports POSIX namespace rules
- Recursive relationship
  - isolates move/rename impact
  - Reduces metadata footprint



Distributed Directory Cache Lease based read locking

Caching Active Directory objects to optimize path operations [7]



Agenda

- The Challenge
- Utilizing relational database technology
  - POSIX Namespace
  - User Defined Attributes
- Prototype
  - Configuration
  - Results

Content Knowledge (User Defined Attributes)

- Provide an extensible set of APIs that will insert/update/delete/select user defined attributes
- Provide a simple/flexible storage architecture
- Provide robust search capability

Why XML?

- Simple storage architecture
  - 1 new table (3 columns)
- Extreme flexibility in UDA implementation
  - Number of UDAs
  - Structure of UDAs
  - Relationship between/among UDAs
  - Robust indexing
  - Search
  - Validation
- DB2 pureXML stores xml data in its native form
- Common – XML is "everywhere"
- Pure relational approach more cumbersome and less flexible

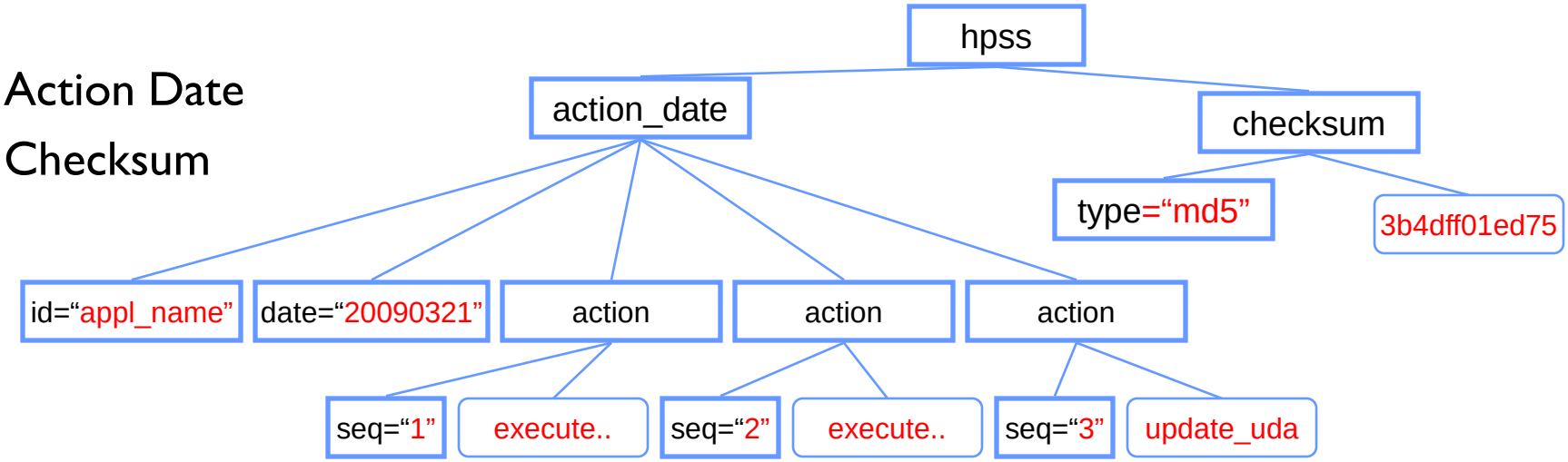
User Defined Attributes (UDAs) Implementation

- Storage based on DB2 pureXML (more later)
- 1 new table, 3 columns
  - OBJECT_ID – namespace object id
  - OBJECT_NS_HASH – for co-location
  - ATTR – XML column
- 2 interfaces:
  - Simple APIs (minimal XML knowledge) to set/get 1 or more attributes using:
    - File path info or Object Handle
      - "/dir1/dir2/file1.dat"
    - XPath representation of attribute (more later)
      - "/hpss/author"
    - Attribute value
      - "Smith"
  - Advanced APIs utilize XQuery

```
<hpss>
    <author>Smith</author>
</hpss>
```

Example

- Action Date
- Checksum



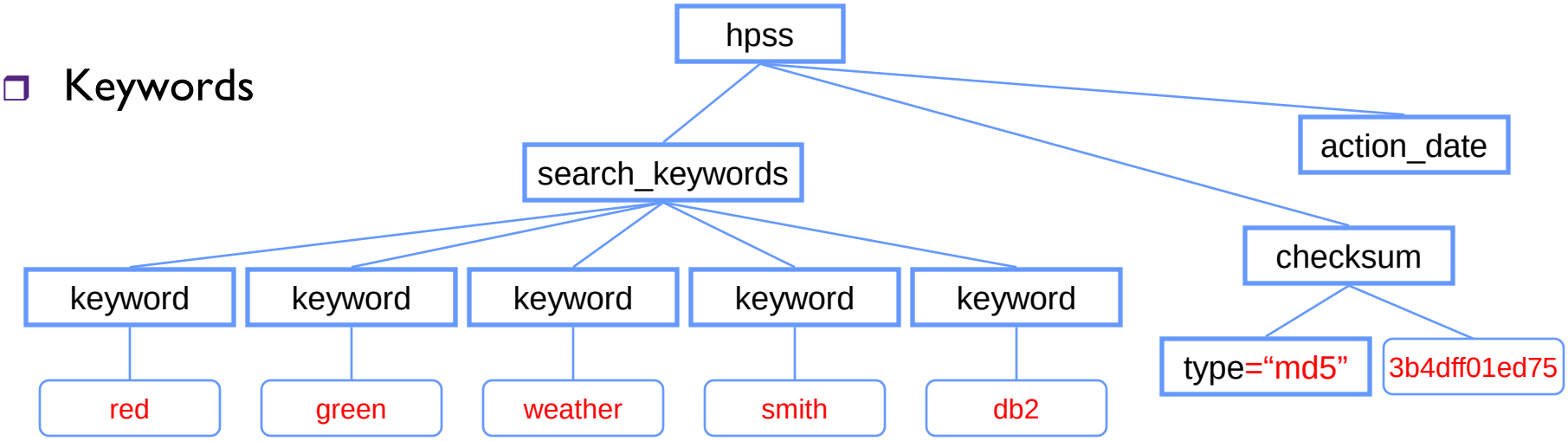

```
create unique index idx1 on
    customer(info)
generate key using
xmlpattern '/hpss/action_date/@id'
as sql varchar(40);

create unique index idx2 on
    customer(info)
generate key using
xmlpattern
    '/hpss/action_date/@date'
as sql varchar(40);
```

```
<hpss>
    <action_date id="appl_name" date="20090321">
        <action seq="1">execute..</action>
        <action seq="2">execute..</action>
        <action seq="3">update_uda</action>
    </action_date>
    <checksum type="md5"> 3b4dff01ed75</checksum>
</hpss>
```

Example cont'd

- Keywords



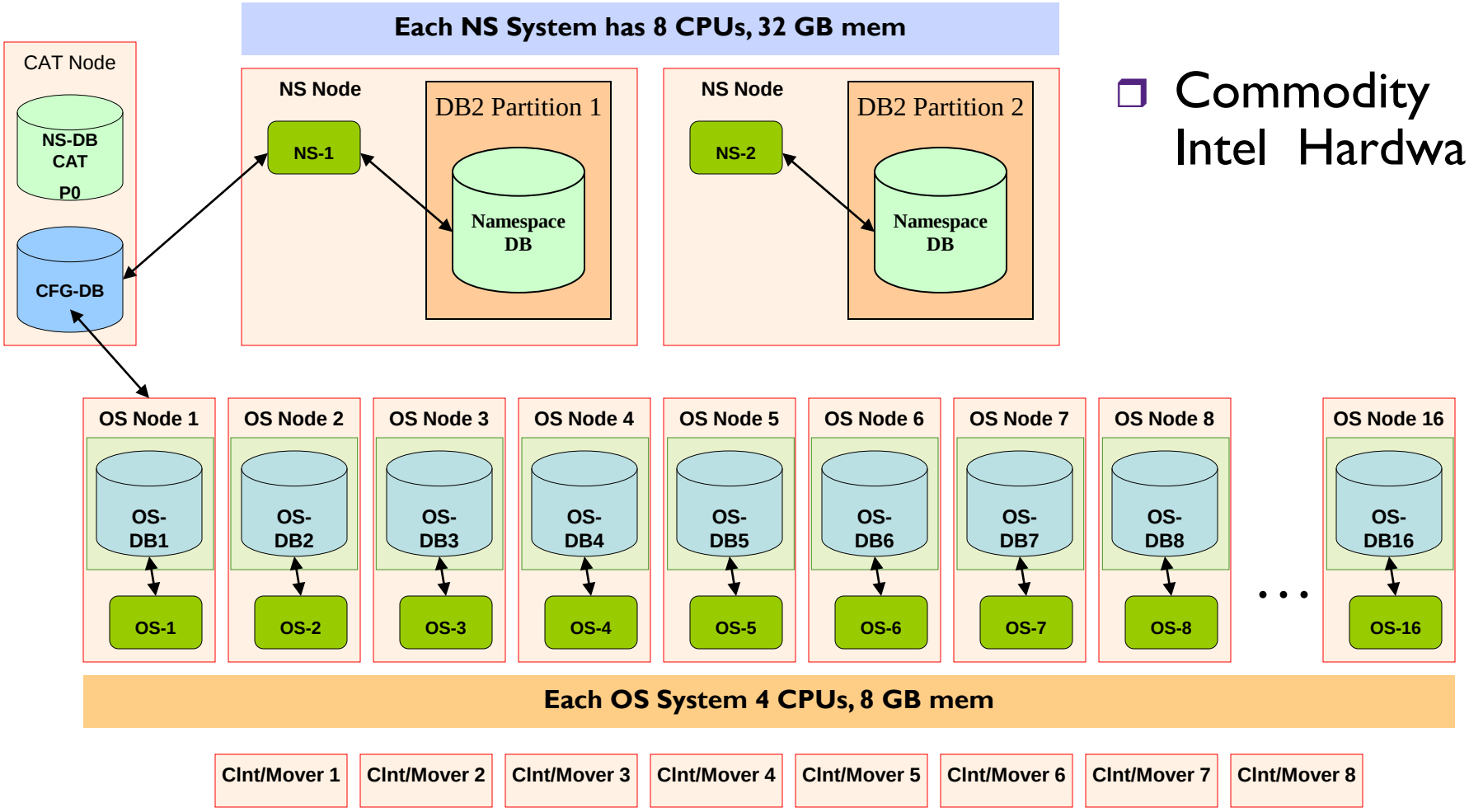

```
create unique index idx1 on
    customer(info)
generate key using
xmlpattern
    '/hpss/search_keywords/keyword'
as sql varchar(40);
```

```
<hpss>
    <action_date id="appl_name" date="20090321">
        <action seq="1">..</action>
    </action_date>
    <checksum type="md5"> 3b4dff01ed75</checksum>
    <search_keywords>
        <keyword>red</keyword>
        <keyword>green</keyword>
        <keyword>weather</keyword>
        <keyword>smith</keyword>
        <keyword>db2</keyword>
    </search_keywords>
</hpss>
```

Agenda

- The Challenge
- Utilizing relational database technology
  - POSIX Namespace
  - User Defined Attributes
- Prototype
  - Configuration
  - Results

Prototype Metadata Configuration

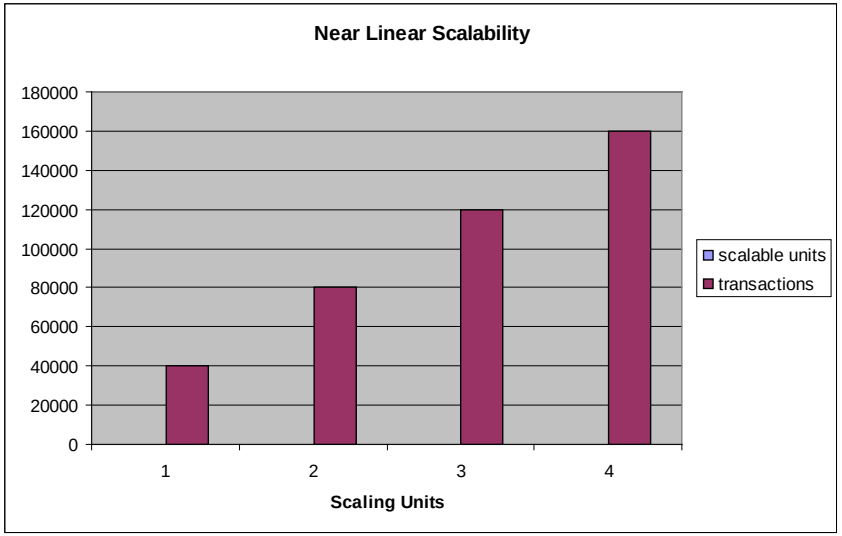


- Commodity Intel Hardware

NS=HPSS Namespace OS=Bitfile Server & Storage Server

Prototype Metadata Operations

- Scalable Unit:
  - 1 NS MDS + 8 OS MDS nodes
  - 10,000 small file (4k) creates/sec
    - 40,000 database transactions
  - 1 file create database transactions:
    - 1 MDS
    - 3 OSS



- Prototype
  - 2 Scalable units
  - Up to 20,000 Small file (4k) creates/sec
    - 80,000 database transactions

- Metadata footprint
  - ~625 bytes per file (HPSS v7 ~1500+- bytes per file, more for multiple tape copies)
    - ~105 bytes for NS MDS
    - ~520 bytes for OS MDS
  - DB2 compression enabled

Summary

- POSIX Namespace:
  - Parent_ID & Name hashing technique
    - **Balanced, self-leveling** metadata and transaction distribution
    - Filename uniqueness within directory
    - Reduces network hops and inter-partition communications
    - Transaction Isolation (no distributed database transactions)
  - Recursive relationship reduces metadata footprint (no full paths stored)
  - Utilizes off the shelf database technology (DB2 DPF)
  - Expandable
  - Near linear scalability across scaling unit
- User Defined Attribute metadata based on xml
  - Utilizes off the shelf database technology (DB2 pureXML)
  - Integrated
  - Extremely Flexible
  - Powerful search capability
  - Validation

Questions?

- Thank you

References

- [1] S.A. Brandt, L. Xue, E. L. Miller, and D.D.E. Long. "*Efficient metadata management in large distributed file systems.*" In Proceedings of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies, pages 290-298, Apr. 2003.
- [2] S.A. Weil, K.T. Pollack, S.A. Brandt, and E.L. Miller, "*Dynamic Metadata Management for Petabyte-Scale File Systems,*" Proc. ACM/IEEE Conf. Supercomputing (SC '04), p. 4, 2004.
- [3] Sage Weil, Scott A. Brandt, Ethan L. Miller, Kristal Pollack, "*Intelligent Metadata Management for a Petabyte-Scale File System*", 2nd Intelligent Storage Workshop, May 2004.
- [4] Yifeng Zhu, Hong Jiang, Jun Wang, Feng Xian, "*HBA: Distributed Metadata Management for Large Cluster-Based Storage Systems*", IEEE Transactions on Parallel and Distributed Systems Vol 19 No. 4 April 2008
- [5] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, Carlos Maltzahn, "*Ceph: A Scalable, High-Performance Distributed File System*", University of California, Santa Cruz (http://www.ssrc.ucsc.edu/Papers/weil-osdi06.pdf)
- [6] Swapnil Patil, Garth Gibson, Sam Lang, Milo Polte, "*GIGA+: Scalable Directories for Shared File Systems*", Petascale Data Storage Workshop Supercomputing '07.
- [7] Randal C. Burns, Robert M. Rees, Darrell D. E. Long, "*Safe Caching in a Distributed File System for Network Attached Storage*", In Proceedings of the 14th International Parallel & Distributed Processing Symposium (IPDPS 2000). IEEE