Can we Archive a Trillion things?!

Distributing & Scaling HPSS Metadata Operations

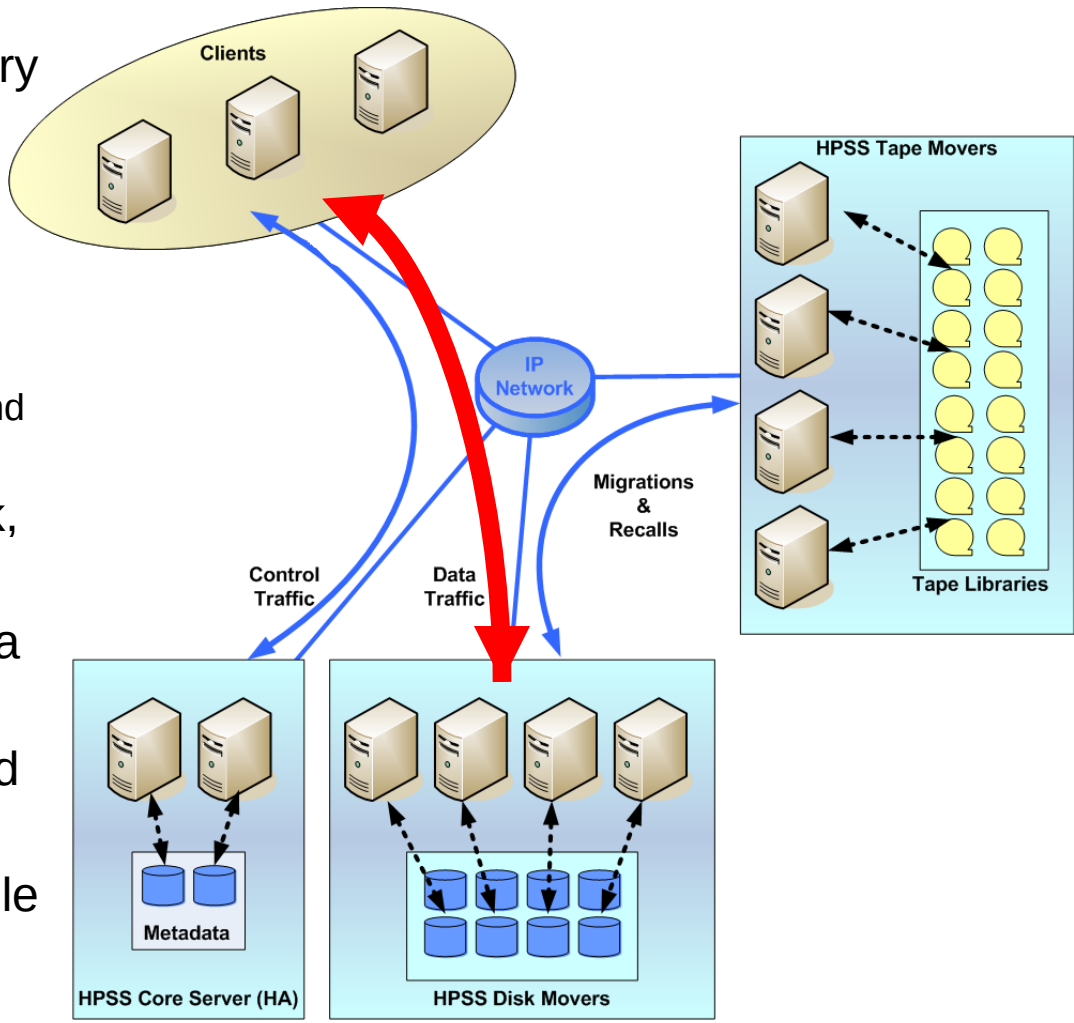
Dave Boomer

Agenda

- What is HPSS?
- Driving HPSS Requirements
- Derived Metadata Management Requirements
- The Challenge
- The Goal
- Distributing HPSS Metadata

What is HPSS?

- HPSS is an HSM, coarse grain parallel file system and file repository
  - Based on the IEEE Mass Storage Reference Model v5
  - Capabilities include:
    - Managing billions of files
    - Managing petabytes of data
    - 1,000 - 2,000 file creates per second (configuration dependent)
- Blend of compute, storage, network, and software technologies
- Transactional-secure DB2 metadata server
- Distributed movers and modularized architecture
- Supports striping, parallel I/O and file aggregation



Driving HPSS Scalability Requirements

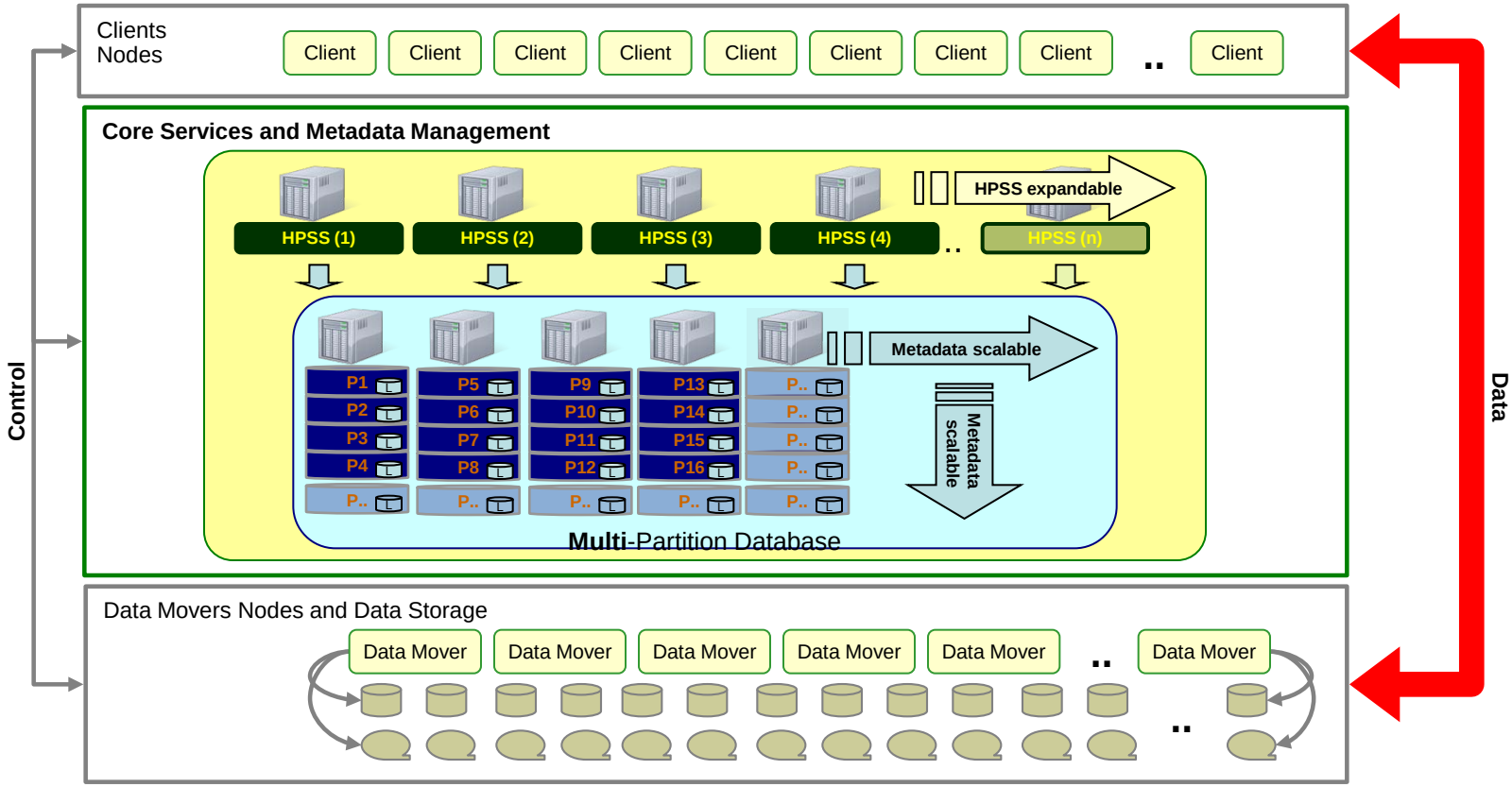
- Increase file ingest **40x**
- Manage **1 trillion files** in a single name space
- Manage **exabytes** of data

Derived Metadata Management Design Requirements

- Distribute metadata enabling parallel access and operations
  - Do so in a balanced and self-leveling manner
    - Minimize skew across data partitions
  - Implement intelligent Co-location of related metadata items
    - Consider
      - Co-location of related metadata items
      - Co-location of transaction metadata
  - Ensure simplified metadata topology alteration
    - Ease of expansion
    - Must maintain metadata balance and co-location
  - Increase metadata capacity
  - Optimize support operations (backup/recovery/statistics)

HPSS Design Outlook

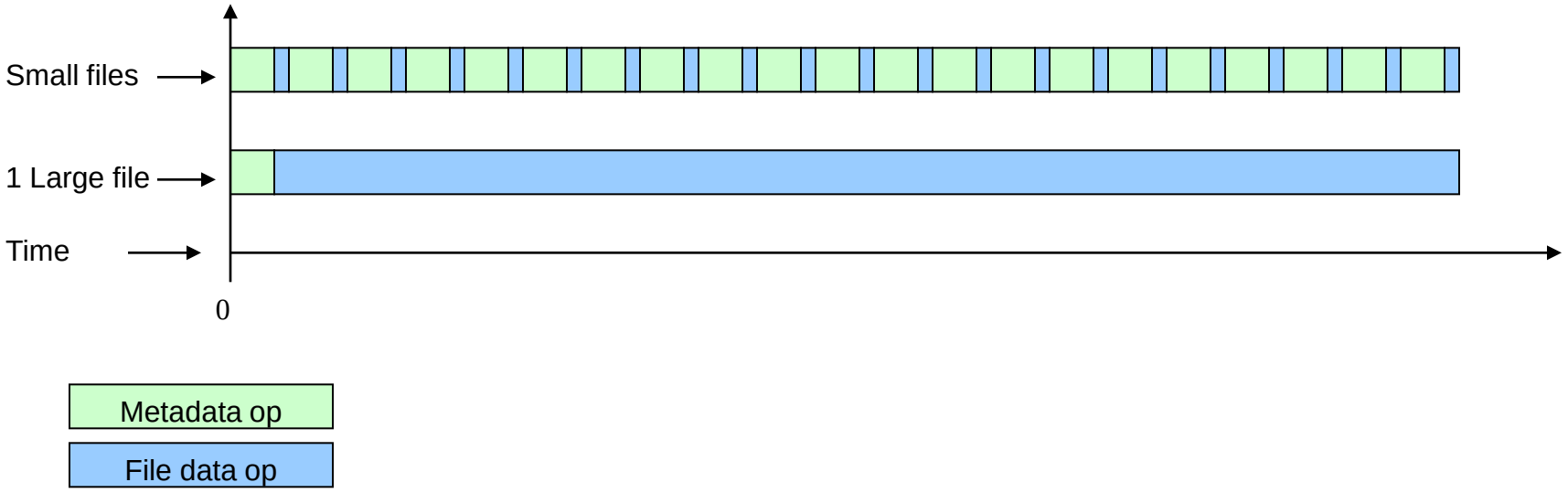
- Flexible HPSS design - preparing the architecture for exascale computing
  - The IEEE Mass Storage Reference Model (MSRM) v5 baseline allows/supports replication (parallel processing) and distribution of key functions
    - This flexibility allow HPSS evolution without major design modifications
  - Enterprise class relational database support for distribution of metadata
    - This flexibility allows HPSS evolution without major design modifications
  - IEEE MSRM and DB2 enable scalability through vertically & horizontally growth



The Challenge

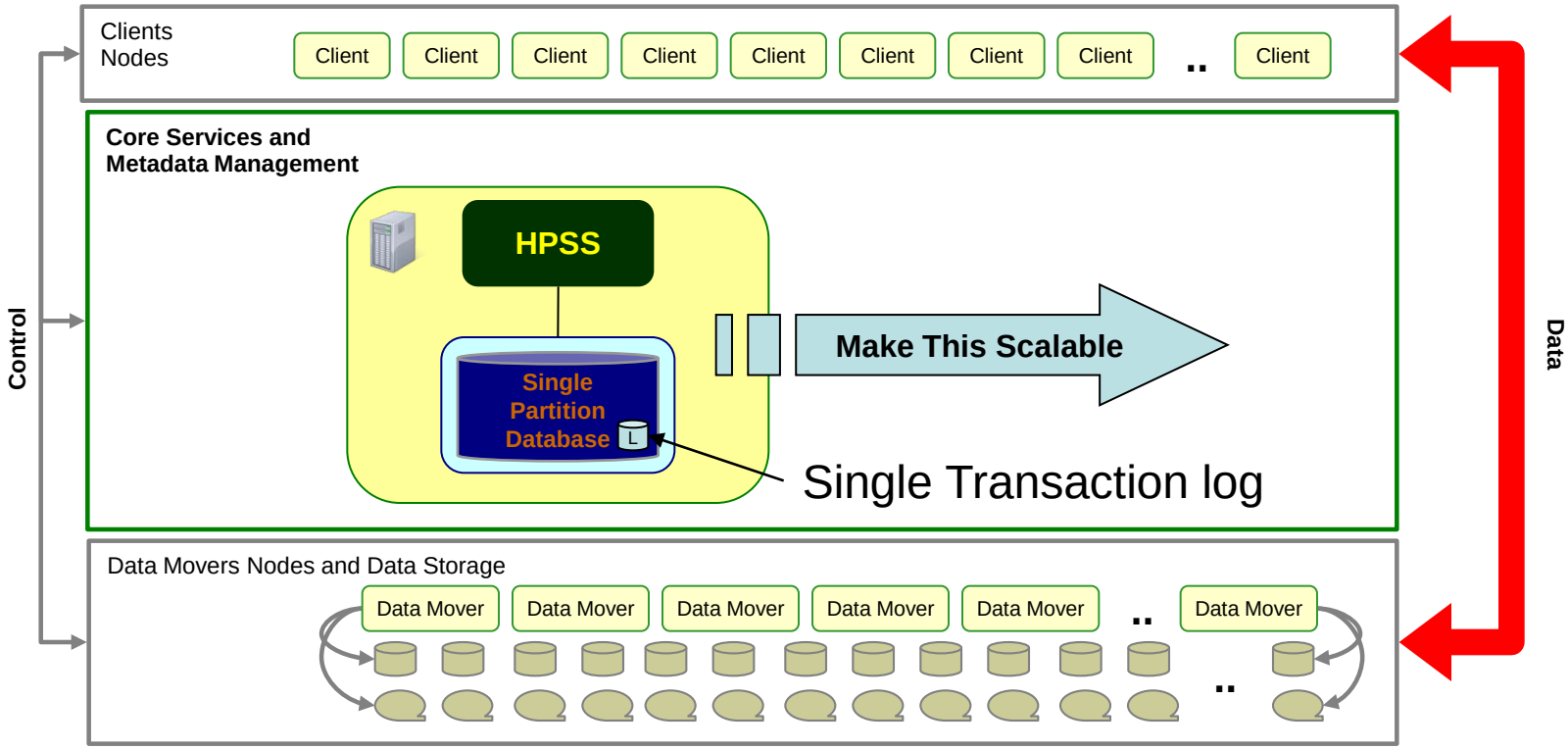
Provide Extreme Scalability to Metadata Transactions

- Single metadata manager architecture fine for “large” file transaction rates… (not fine if you have 1 trillion large files)
- Data movement for large files is the significant component of file ingest transaction
  - Metadata transaction small percentage of large file create overhead
- However…
  - Metadata overhead significant percentage of complete file ingest transaction for small files
  - Adds database transaction overhead to metadata manager
- **Small Files and or More files = More Metadata Operations**



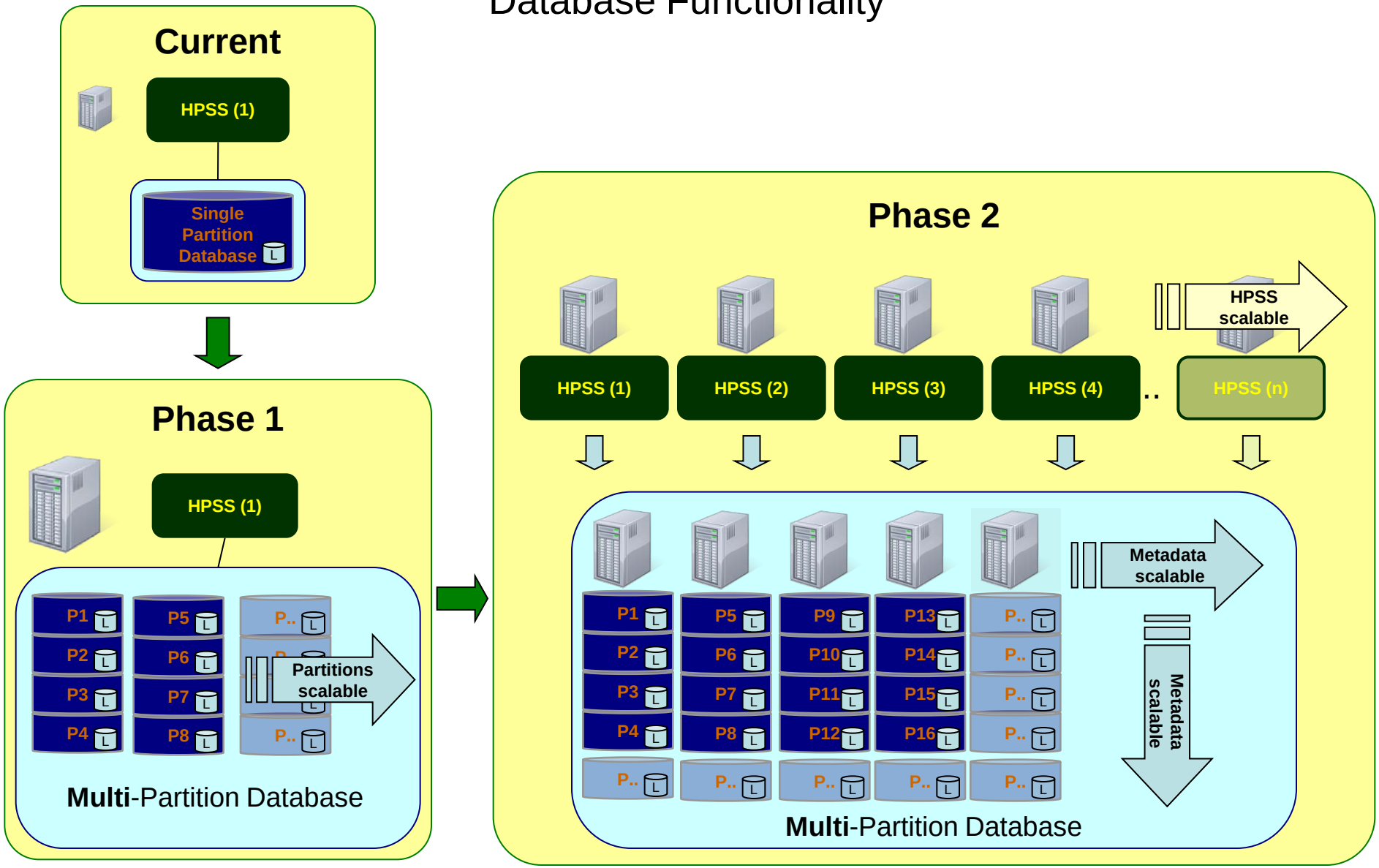
The Challenge cont'd

- Address single metadata database bottleneck
  - **1 Database partition (1 transaction log)**
- **Take advantage of DB2 "off the shelf" functionality**
  - **Significant capability reduces development effort**
  - **Extreme reliability ("ACID")**



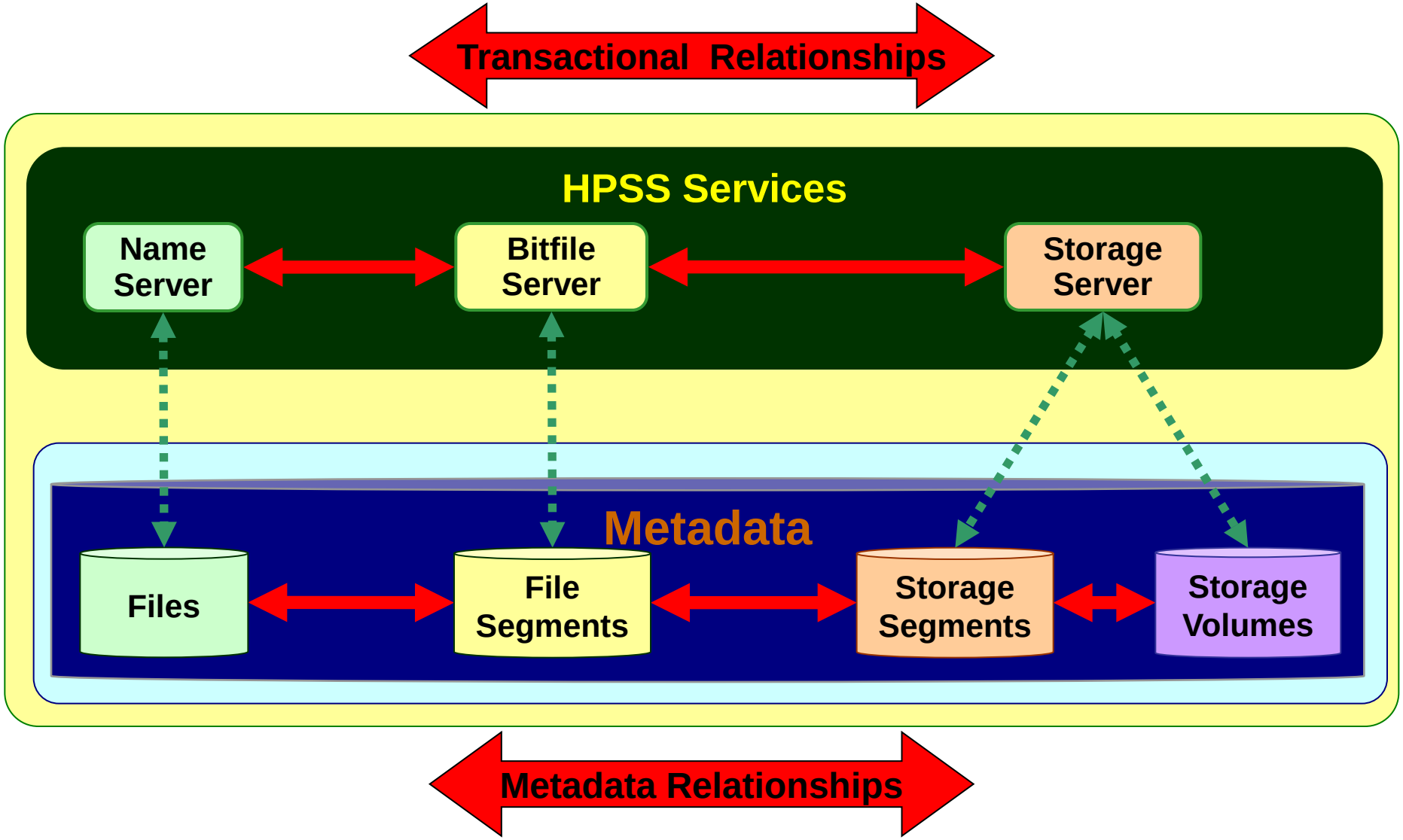
The Goal

Add Metadata Scalability in Phases, Using Native DB2 Distributed Database Functionality

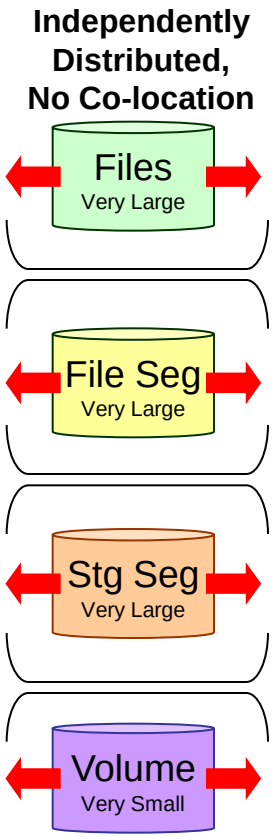


Metadata Distribution Method Based on Relationships

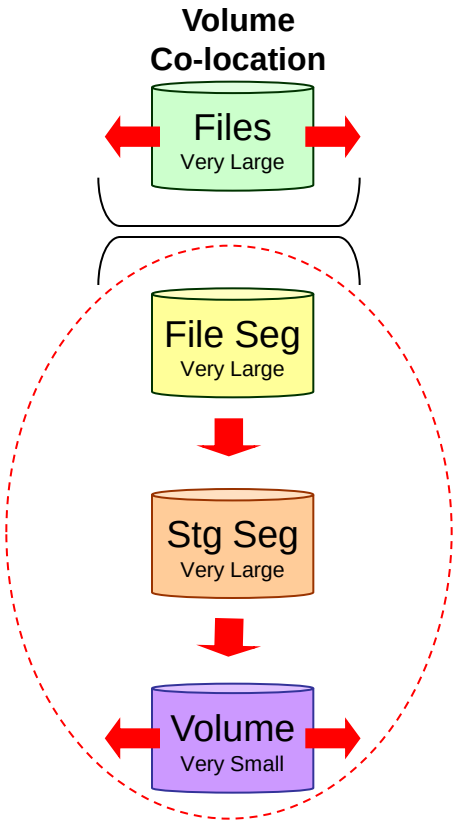
Goal: Minimize Distributed Transactions and Maximize Object Metadata Co-location



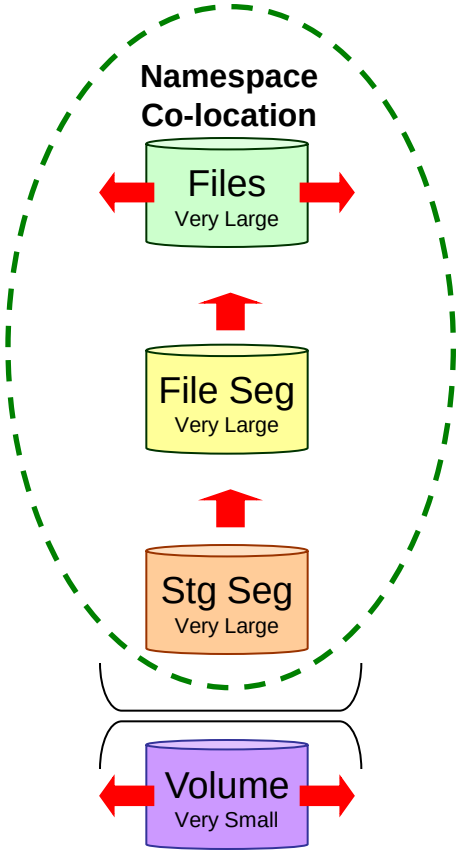
Options for Distributing HPSS Metadata



- Good Distribution
- **No Co-location**
- **Max Distributed Transactions**
- Simple Expansion



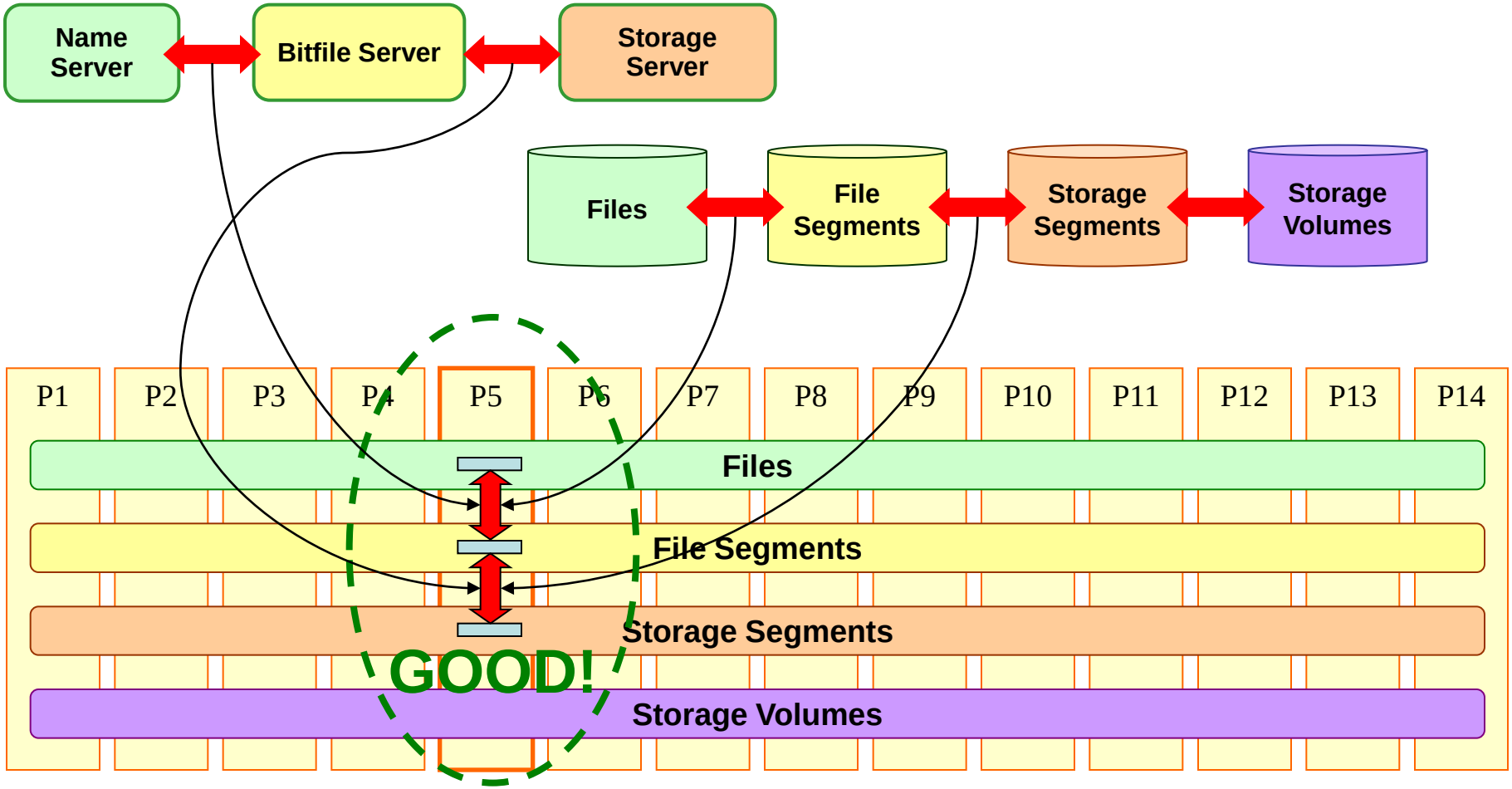
- Average Distribution
- Good Co-location
- Min Distributed Transactions
- **Difficult Expansion**



- **Good Distribution**
- **Good Co-location**
- **Min Distributed Transactions**
- **Simple Expansion**

Minimize Distributed Transactions Through Intelligent Co-location

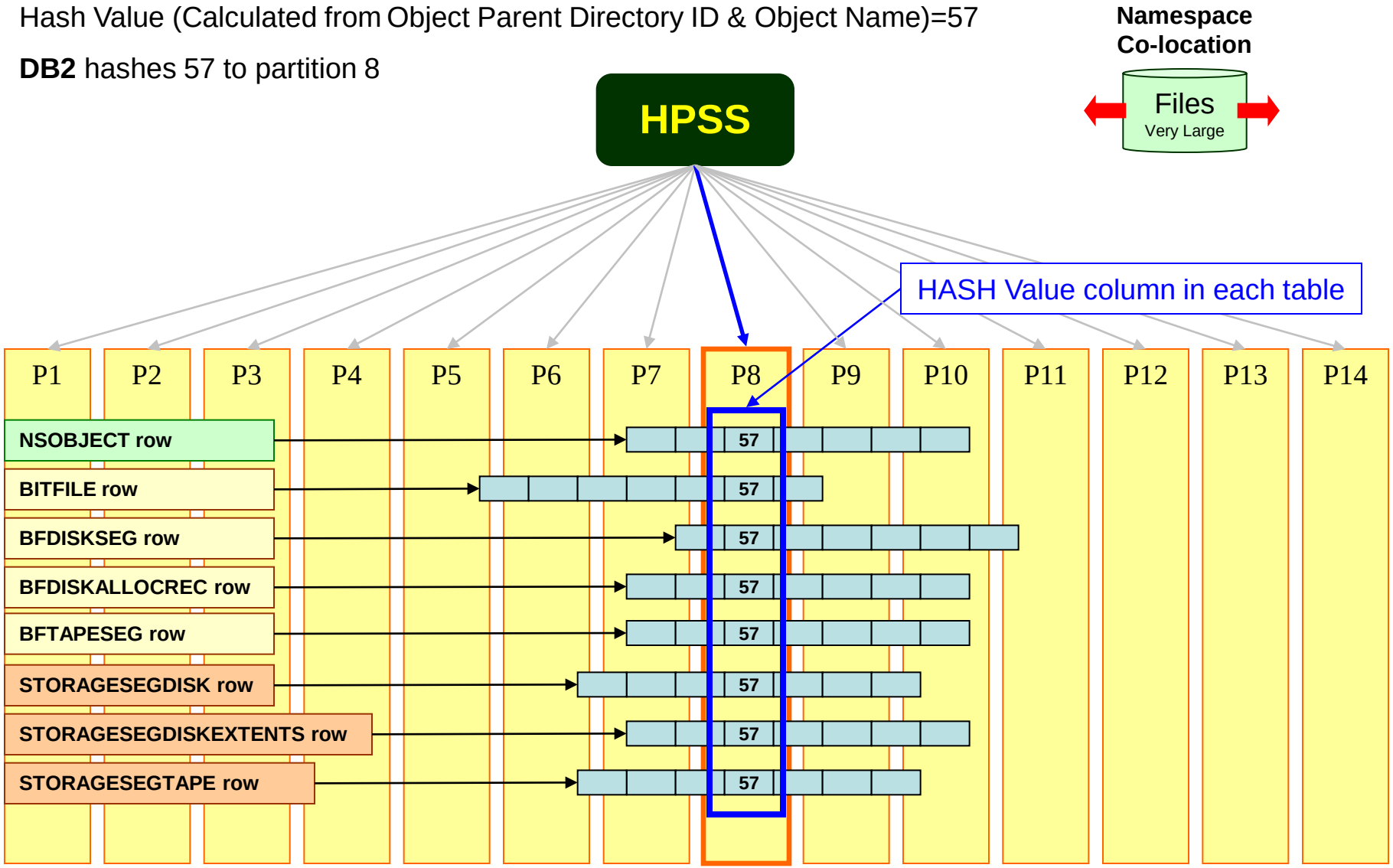
Metadata and Transaction Relationships co-located within same database partition



Co-Location based on File HASH

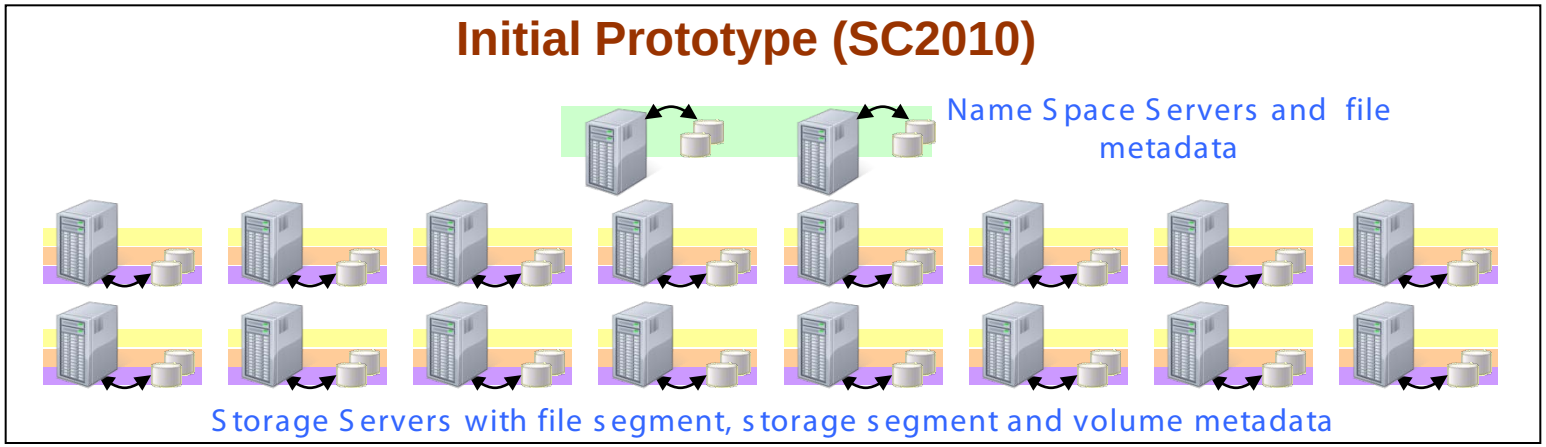
Hash Value (Calculated from Object Parent Directory ID & Object Name)=57

DB2 hashes 57 to partition 8



Status

- SC 2010 Prototype
  - **18,000** file creates per second!
  - Metadata stored in multiple separate database



- Design has evolved into a single partitioned database with an optimized distribution mechanism
  - All metadata accessible from any metadata node
  - Balanced and self-leveling
  - Maximizes object metadata co-location
  - Minimizes distributed transactions
  - Simpler maintenance/management

Summary

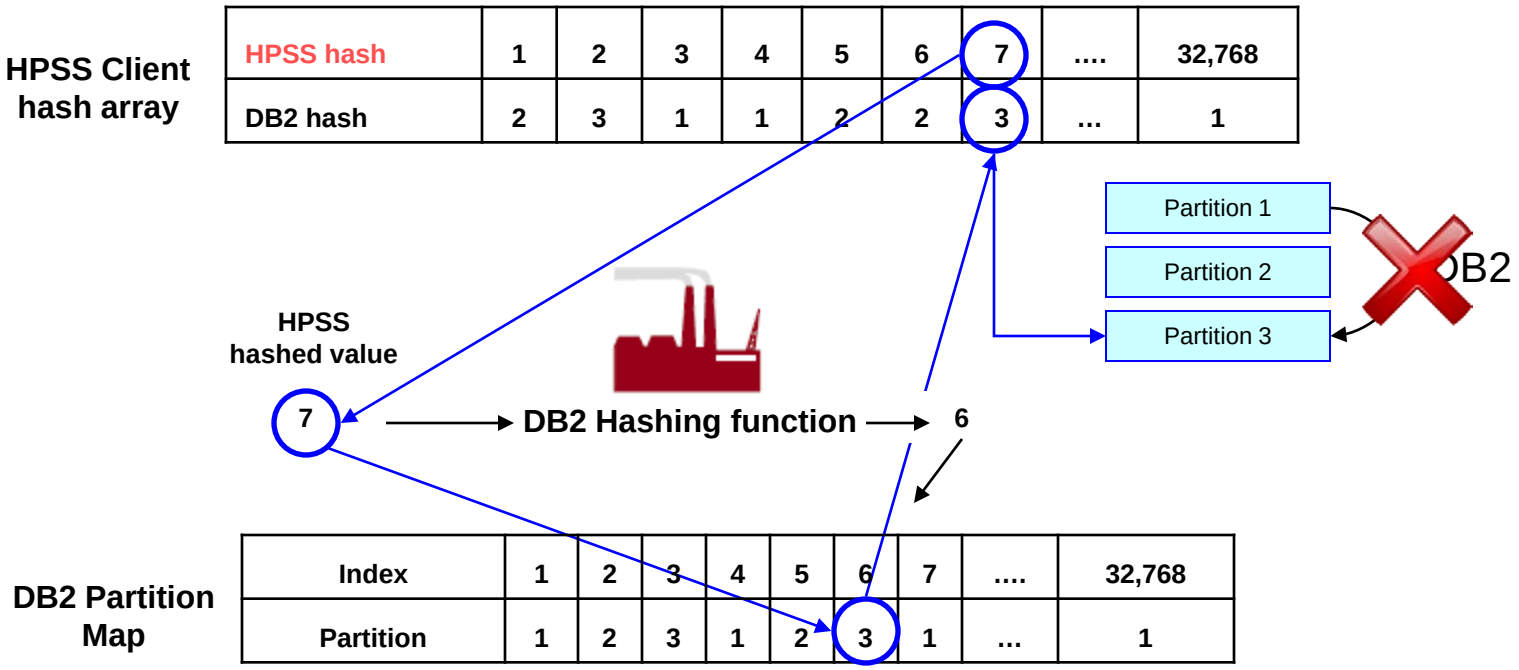
- Proof of concept shows a effective and scalable metadata distribution mechanism
  - Optimizes parallel access
  - Ensures balanced and self-leveling metadata (minimize skew)
  - Maximizes Object metadata co-location
    - Based on Relationships
      - Data
      - Transaction
  - Simplifies topology alterations
  - Simplifies maintenance/management

Backup Slides

Partitioned Databases

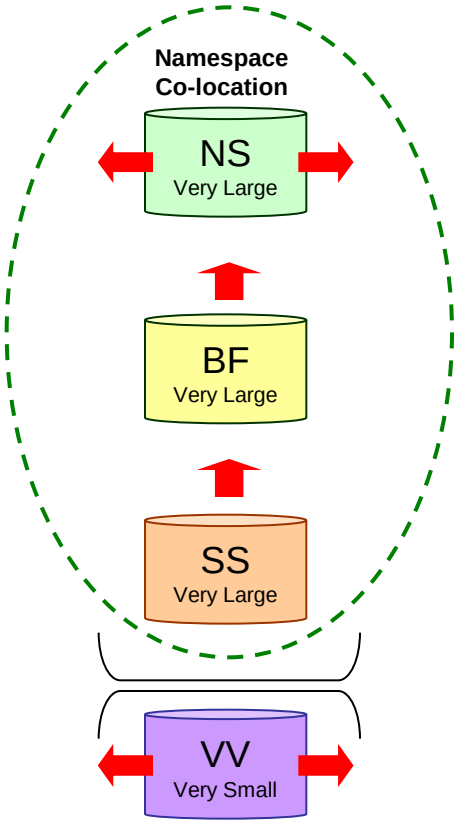
How the HPSS Client determines which NS to use

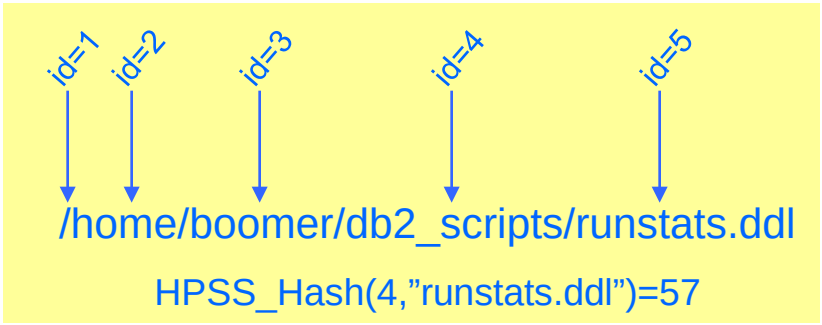
- Hashed value based on Parent ID and object name… hash(parent_id & name)=**object_ns_hash**
  - **1 <= hash output <= 32,768**
- Create table nsobject (obj_id bigint, **obj_ns_hash** smallint….) Distribute by hash (**obj_ns_hash**)
- Create unique constraint (parent_obj_id, name, **obj_ns_hash**)
- Client communicates directly with owning partition, reducing network hops



Distributing HPSS Metadata

POSIX namespace

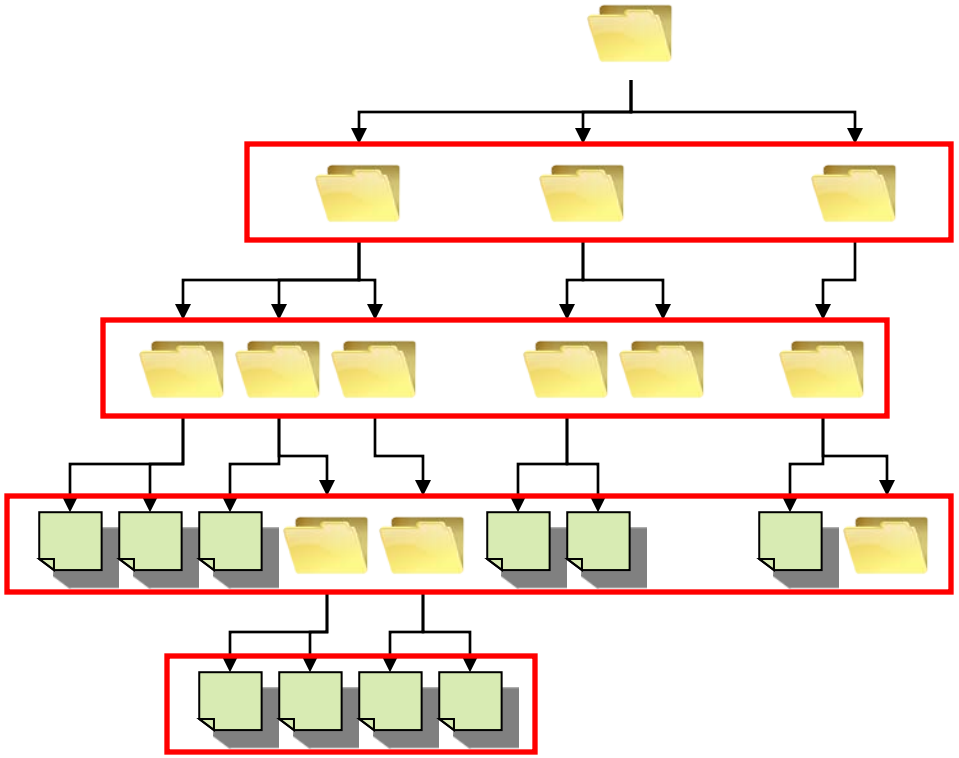


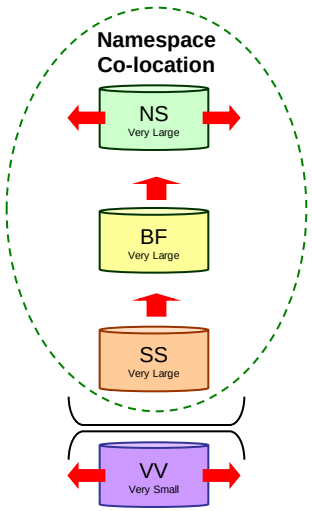


- Hashing the combination of:
  - Parent directory identifier (unique/does not change)
  - Object name
- Why?
  - Database will enforce uniqueness *within a database partition*
    - 2 files with same name in same directory generate same hash value
  - DB2 will *hash* our calculated hash value to determine partition
  - Namespace objects within a directory are balanced across cluster
    - Metadata balance
    - Database transaction balance

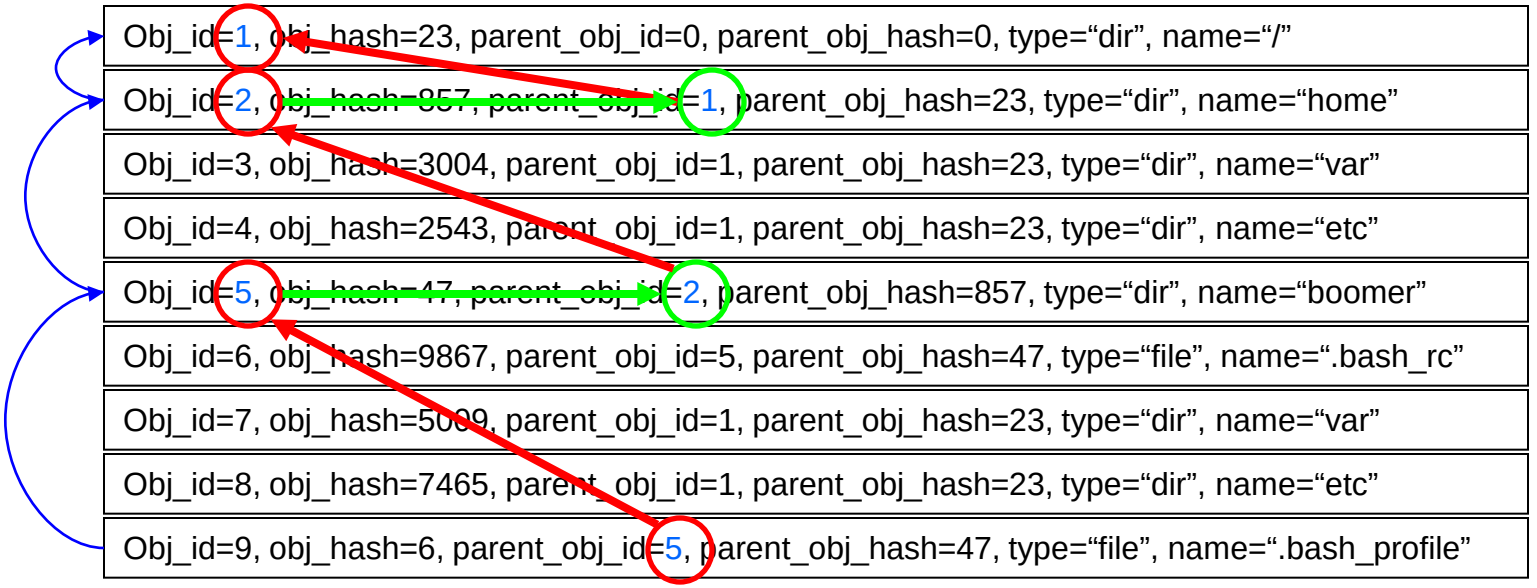
POSIX Namespace Balance

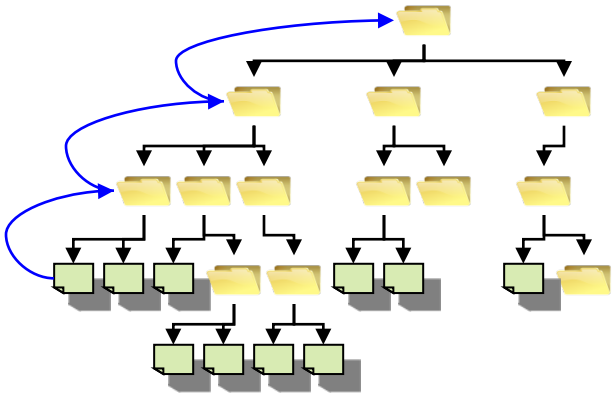
- Hashing technique ensures
  - Each layer is distributed evenly
  - Reduces network “hops”
  - Supports POSIX namespace rules
- Recursive relationship
  - isolates move/rename impact
  - Reduces metadata footprint





Namespace Table





/home/boomer/.bash_profile

File path stored as recursive data relationship

