GIGA+: Scalable Directories for Shared File Systems

Swapnil V. Patil

Garth Gibson and Milo Polte (PDL @ CMU)

Sam Lang (Argonne National Lab)

Carnegie Mellon
Parallel Data Laboratory

Pushing Scale: *googol* = 10^{100}

- If things get big - really big - most systems can break easily
  - "Push the limits" in scalability by targeting numbers that break current designs

- Start with building a file system that has really, really huge directories
  - Scale to store billions to trillions of files in a dir
  - Handle more than 100K operations/second

Why do this - "huge directories"?

- Parallel FS already scale file size and concurrent access, what's next …
  - Customers want directories with more than million entries
- Applications sometimes use the file system as a fast, lightweight "database"
- Large number of small files written in a directory
  - Logging phone records
  - Check-pointing large clusters
  - Scientific experiments (genomics, physics)

Increasing parallelism

- Applications becoming highly parallel
  - Large compute clusters
    - Today 1000s of nodes, soon 10000s nodes
  - More cores per CPU

- So, solutions must scale in concurrency and shared memory should not be assumed

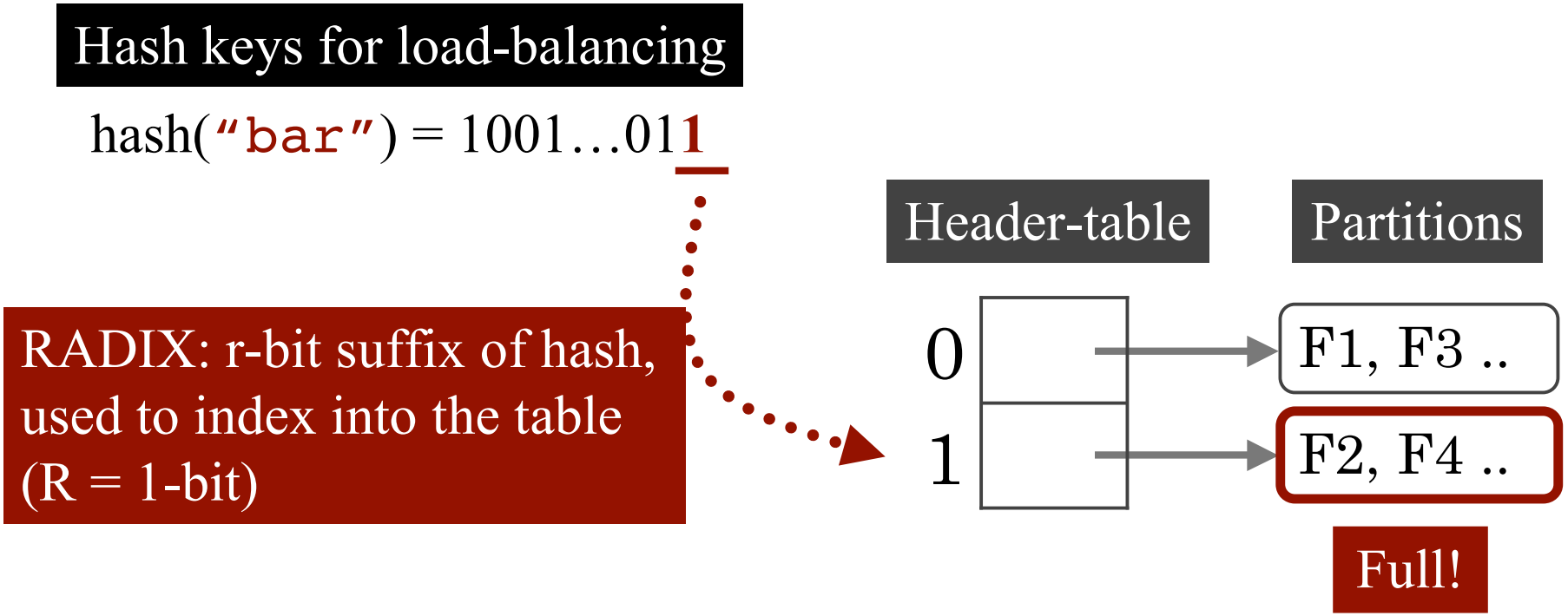
Outline

- Introduction and motivation
- Related work
  - Current systems and how they limit scalability
- GIGA+ in action
- GIGA+ techniques
- Status and summary

Out-of-core indexing structures

- B-trees vs hash-table
  - XFS [Sweeney96], Ext2/Ext3 [Tso02]
  - Use hash-table for O(1) lookups
  - B-trees support range queries, hash-tables don't
    - File system API doesn't support range queries
- Need incremental growth of the directory
  - Small directory performance not penalized
- Use extendible hashing [Fagin79]

Extendible Hashing [Fagin79]



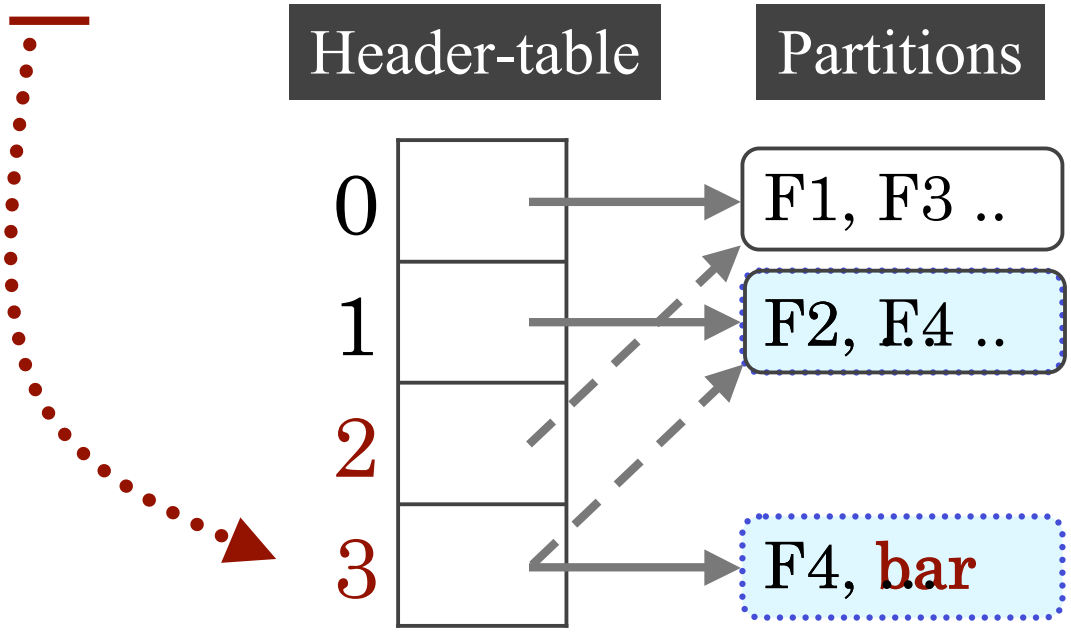
- Header-table points to partitions
  - Each entry holds a pointer to a single partition
  - One partition can be pointed to by multiple entries

Extendible Hashing [Fagin79]

Hash keys for load-balancing

hash("bar") = 1001…011

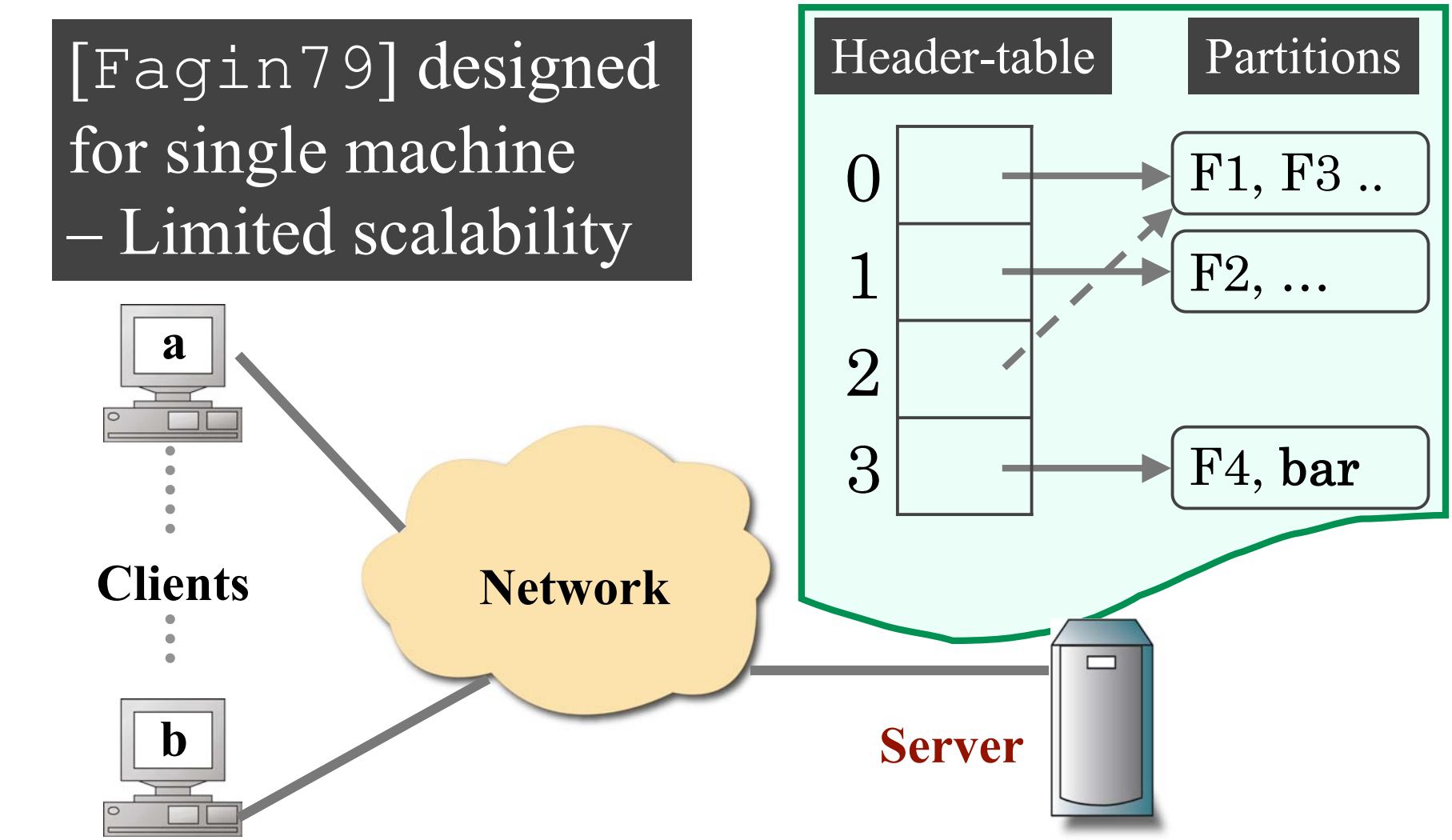
RADIX increases, that uses the growing table (R = 2 bits)



- Header-table doubles, if necessary
  - On splitting, the new partitions distribute their keys

Extendible hashing on single server

[Fagin79] designed for single machine – Limited scalability



Extendible hashing on >1 machine

- GPFS [Schmuck02] is a parallel file system
  - Uses extendible hashing on multiple machines
- Partitions are stored on the server
  - Directory is represented as a large file
  - Large files striped on many servers
- Header-table (mapping information) at clients
- How to lookup partitions and get up-to-date partition-to-server mapping?

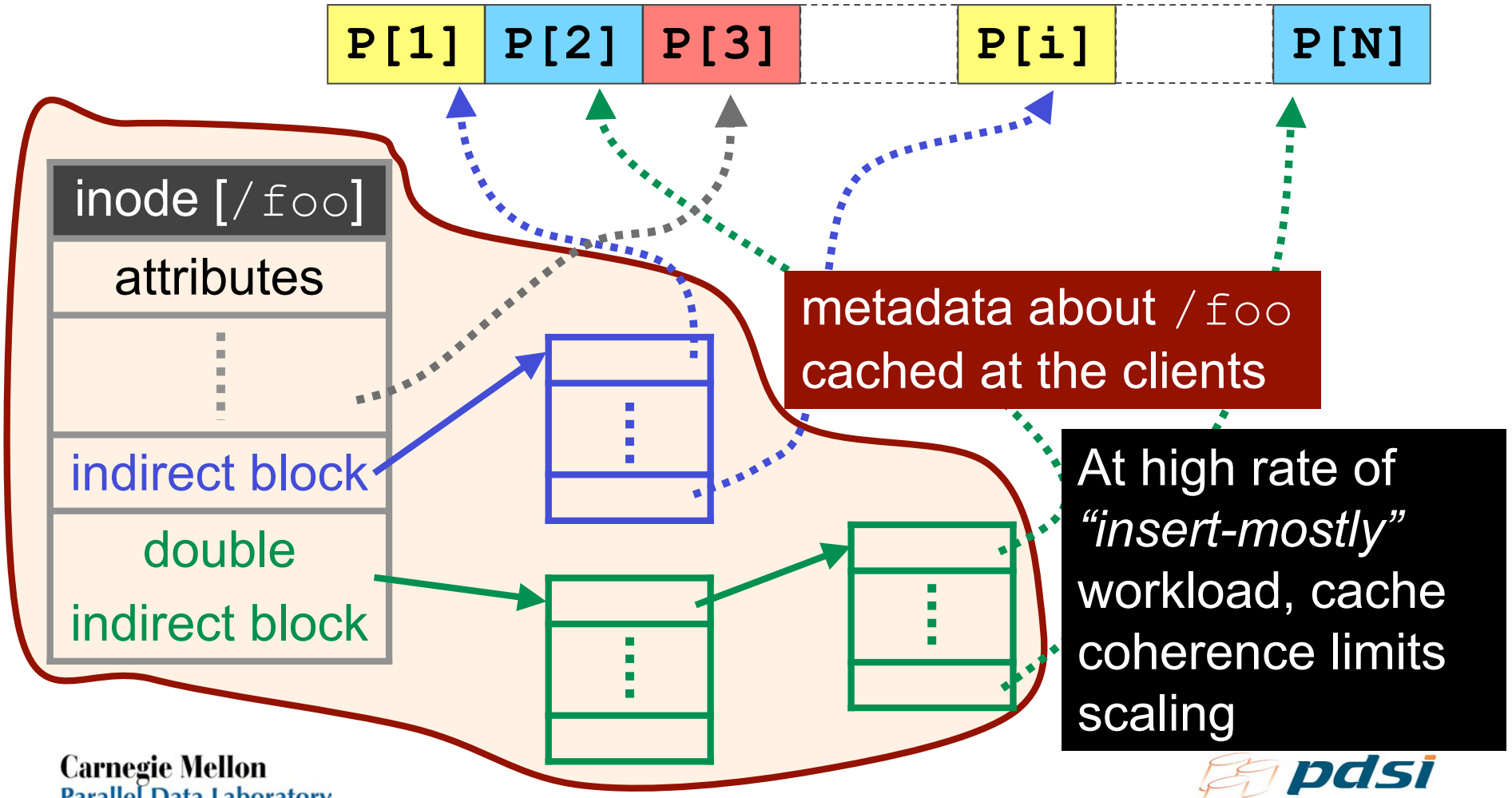
Data cache coherence in GPFS

Dir `/foo`, represented as a file, striped across servers

- Concurrent access to file data
  - Divides a file into multiple regions
  - Assign a server to lock these regions during concurrent access
- Clients get the lock on the region, update it and write it back
  - Data cache coherence can limit scalability

Caching metadata maps in GPFS

Dir `/foo`, represented as a file, striped across servers



Carnegie Mellon
Parallel Data Laboratory

pdsi

Outline

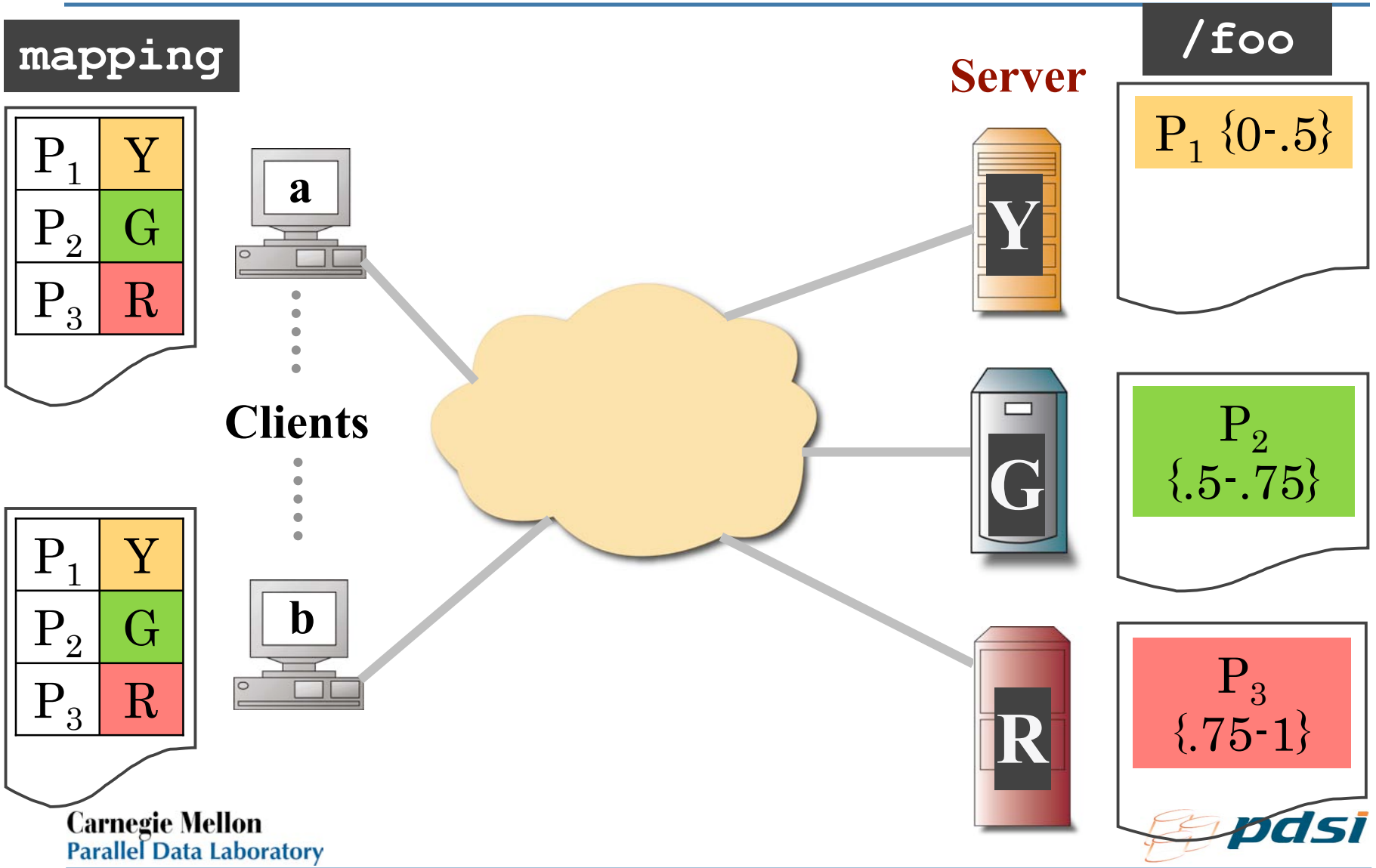
- Introduction
- Related work
- GIGA+ in action
  - Example
- GIGA+ techniques
- Status and summary

GIGA+ key ideas

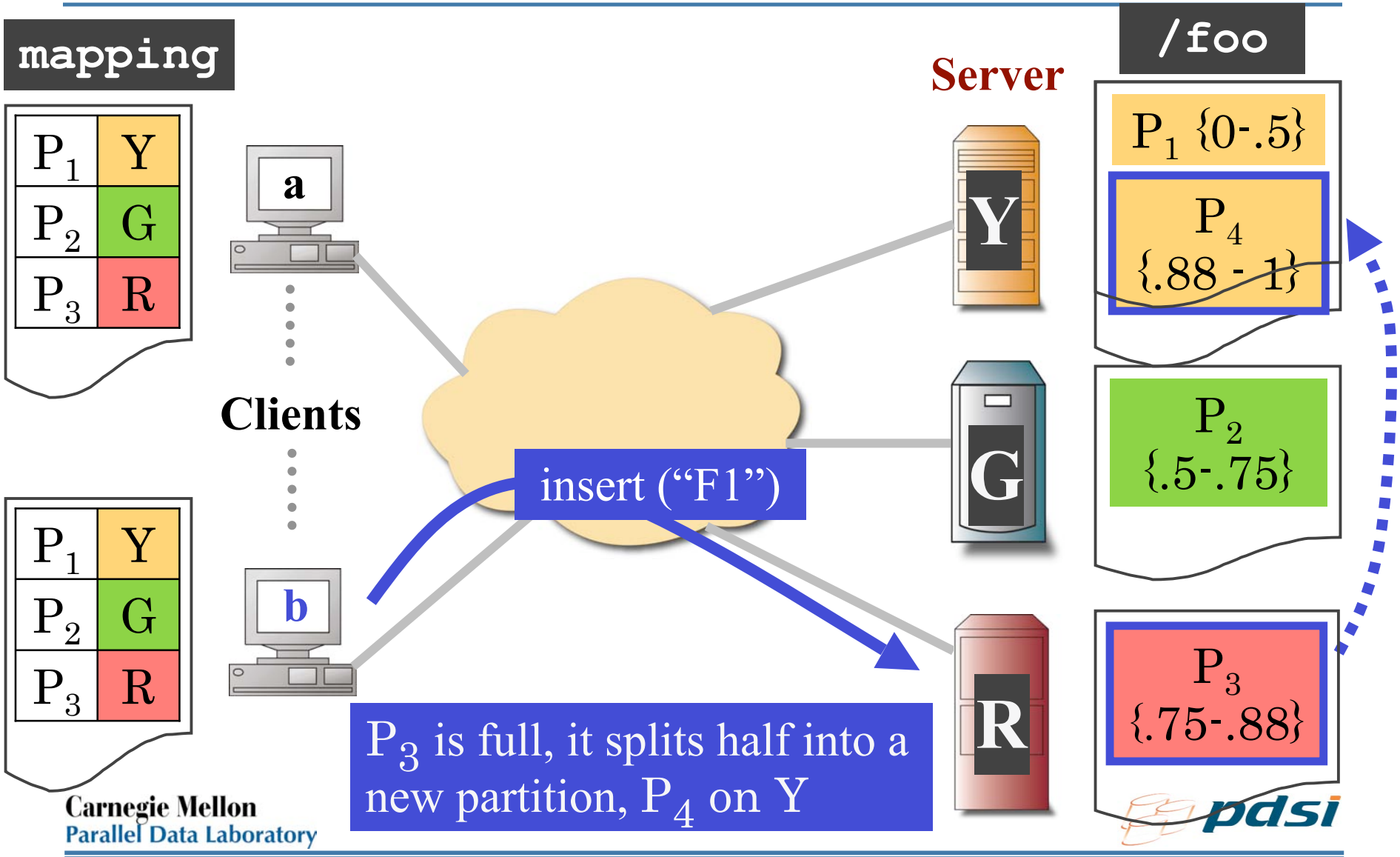
- Highly decentralized and parallel growth of the directory
  - Highly decentralized: Decentralized splitting
  - Load-balanced: Hash the key

- High concurrency through minimal synchronization overhead
  - Indexing technique that tolerates the use of stale metadata information

GIGA+ in action

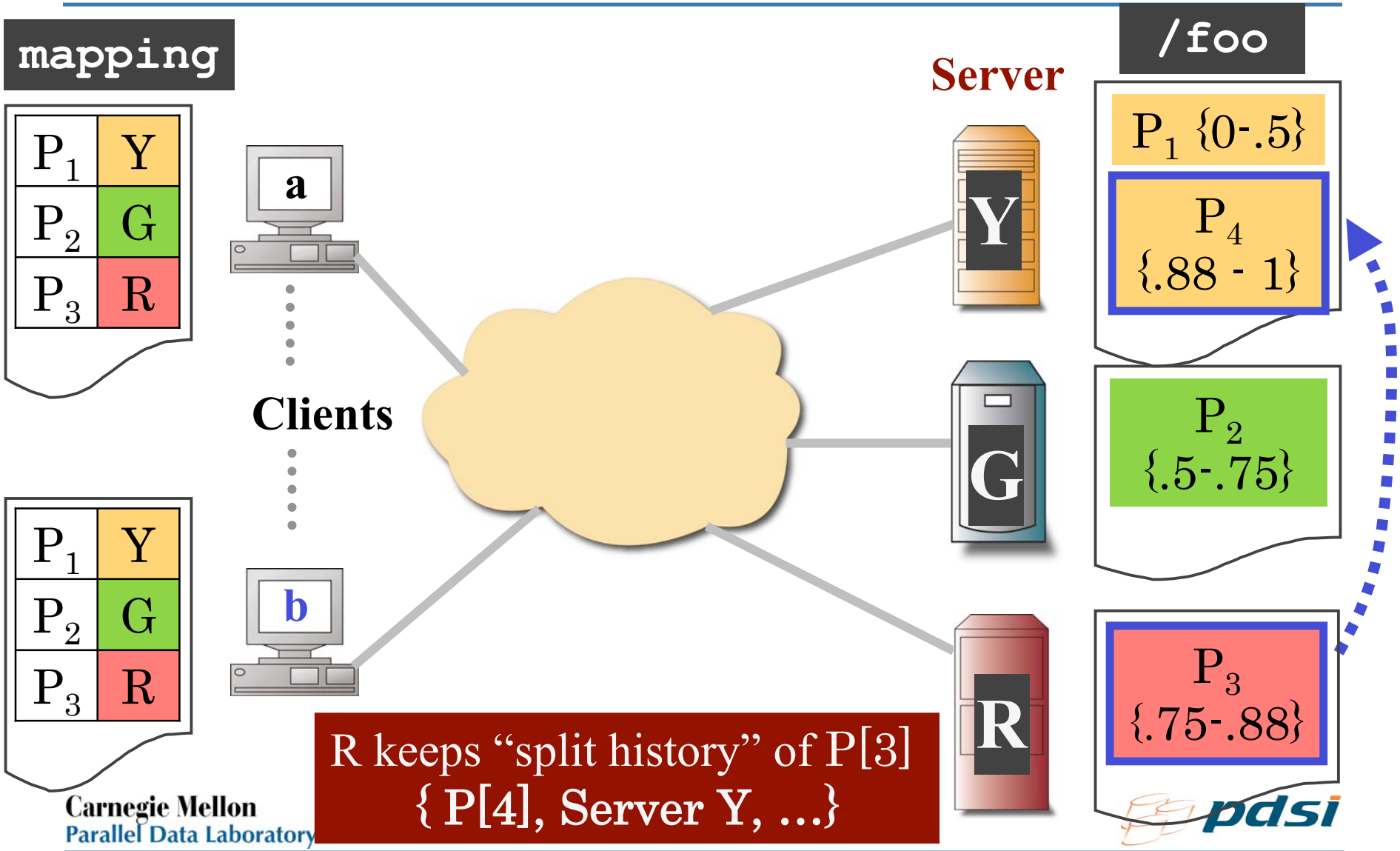


GIGA+ in action

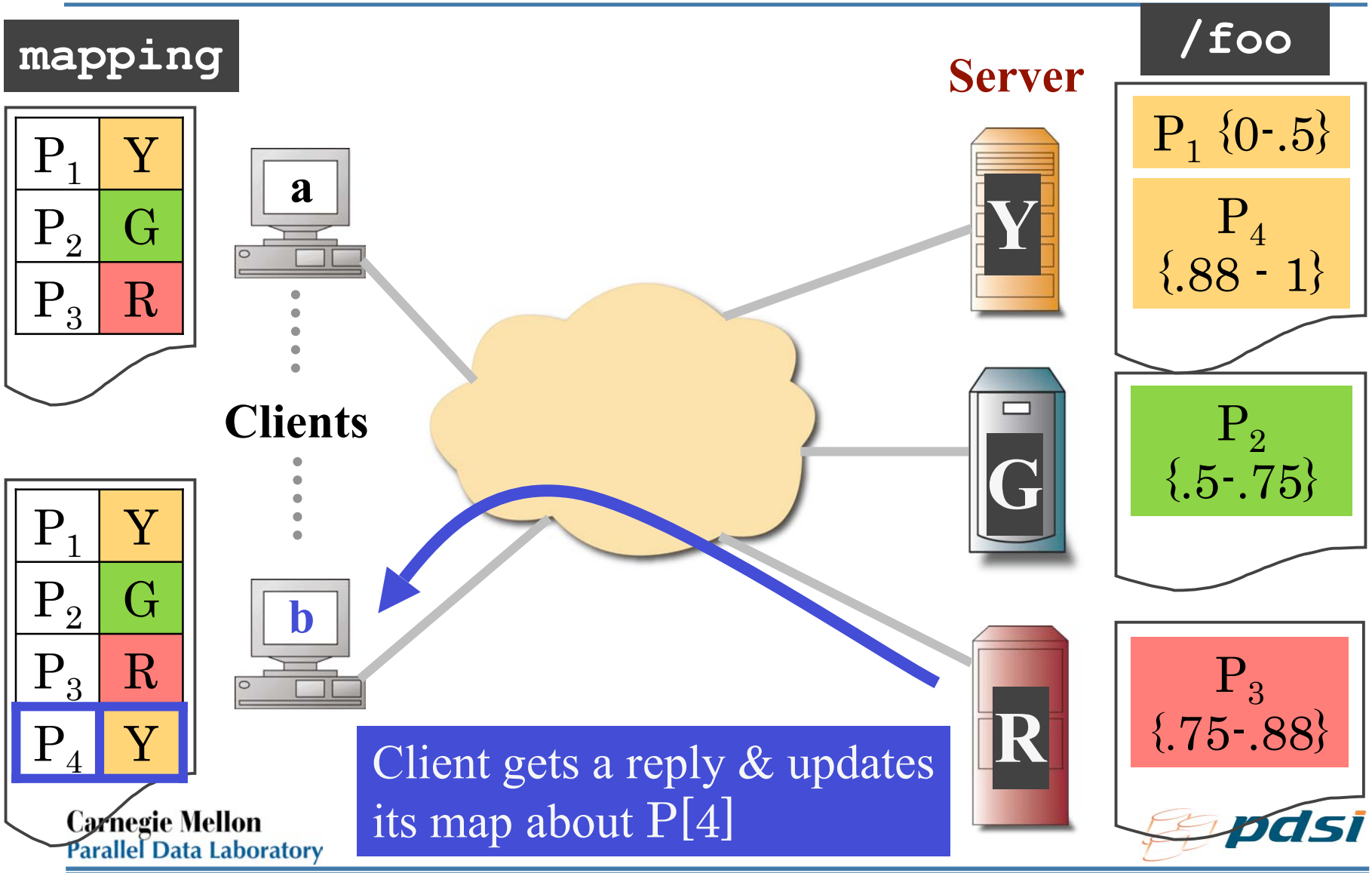


Carnegie Mellon
Parallel Data Laboratory

GIGA+ in action



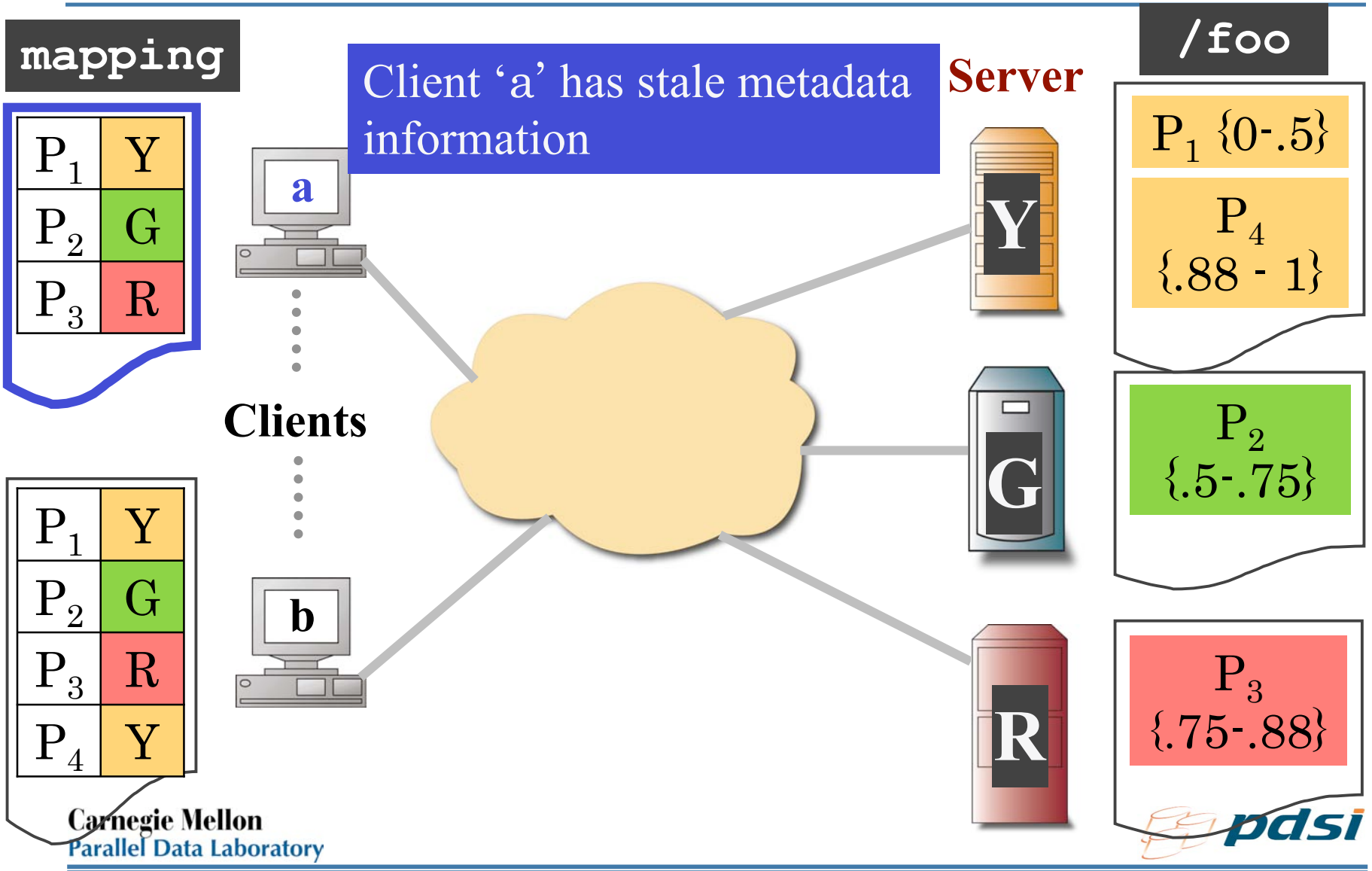
GIGA+ in action



Carnegie Mellon
Parallel Data Laboratory

pdsi

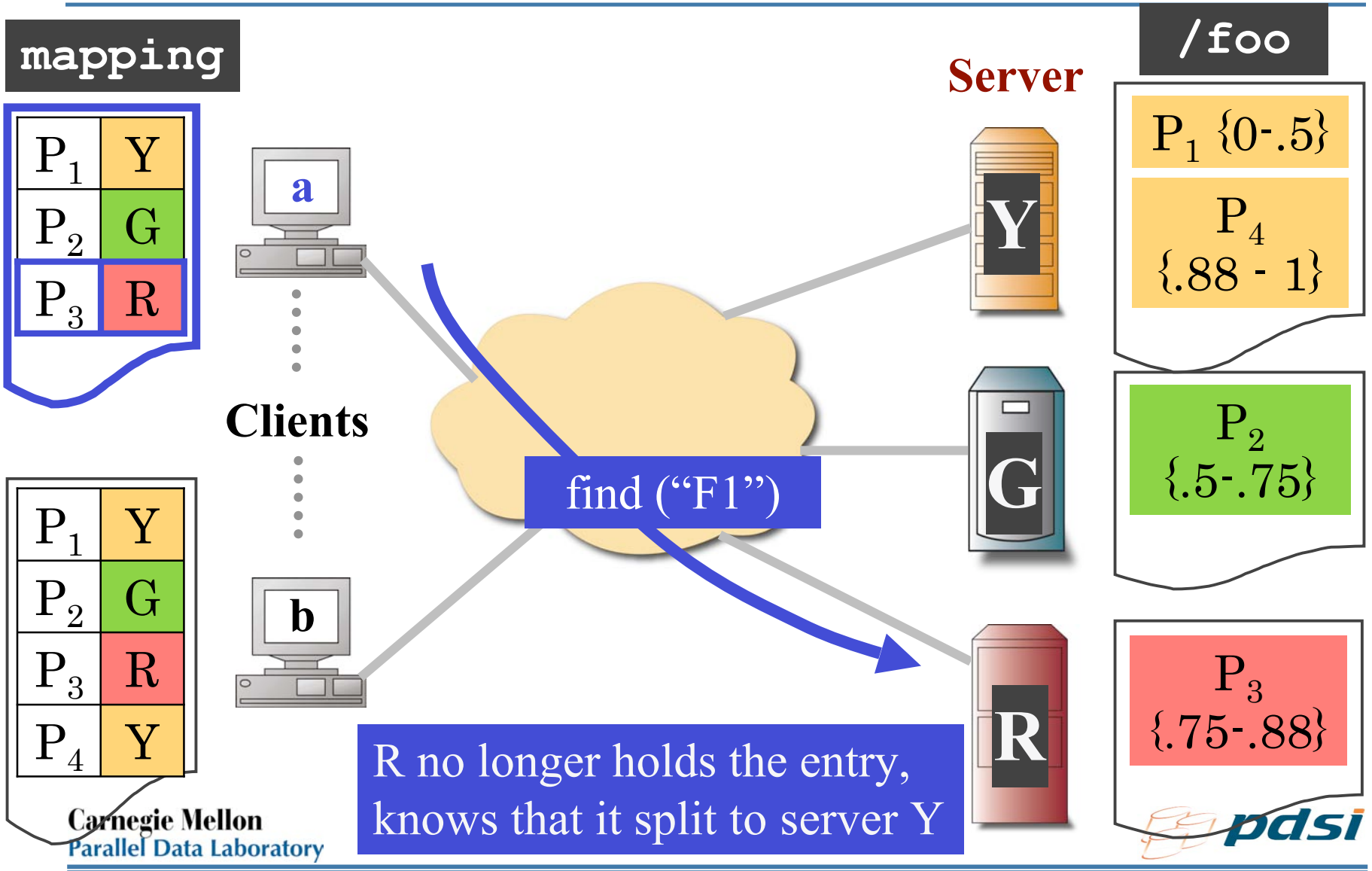
GIGA+ in action



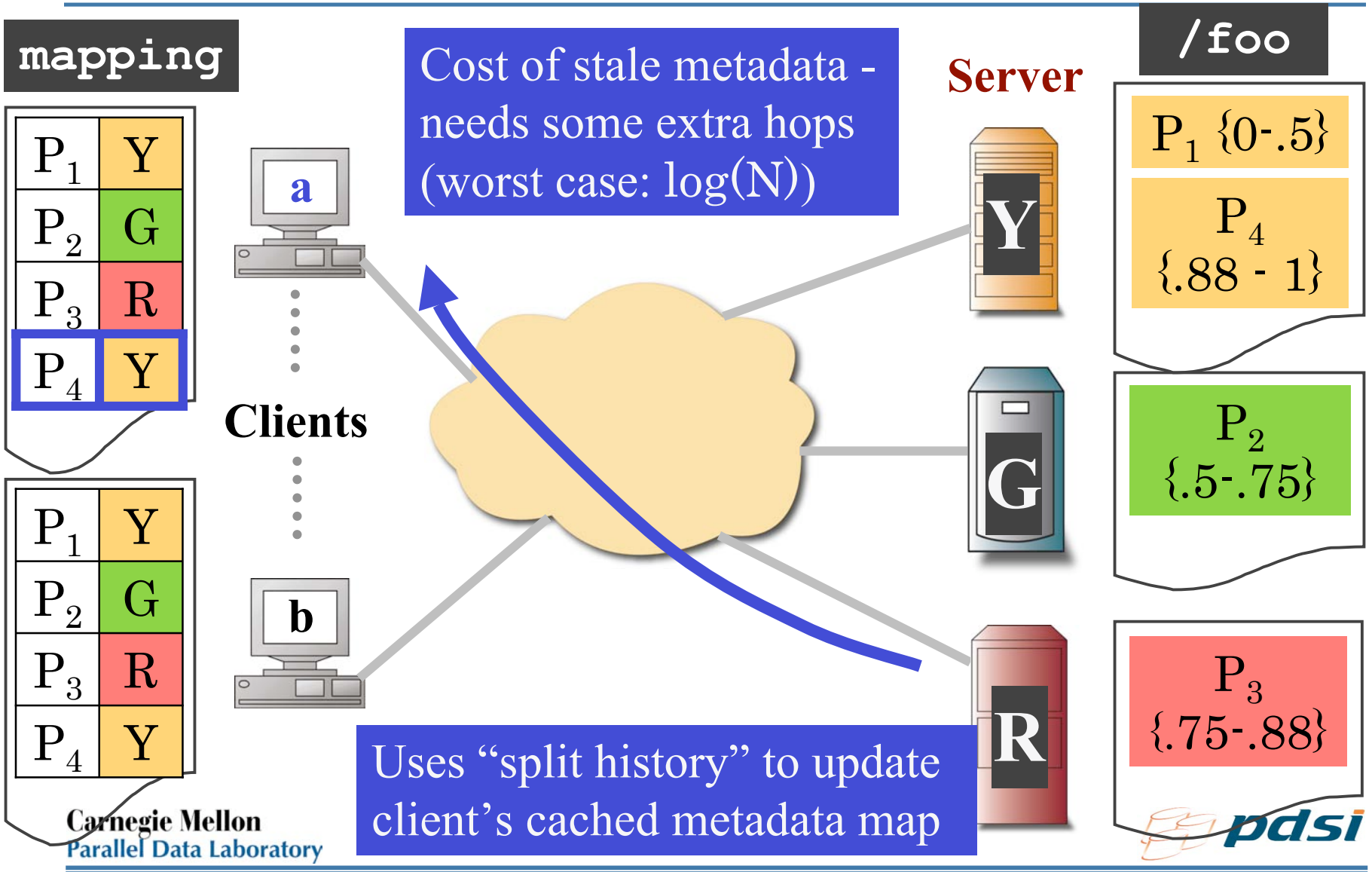
Carnegie Mellon
Parallel Data Laboratory

pdsi

GIGA+ in action



GIGA+ in action



Outline

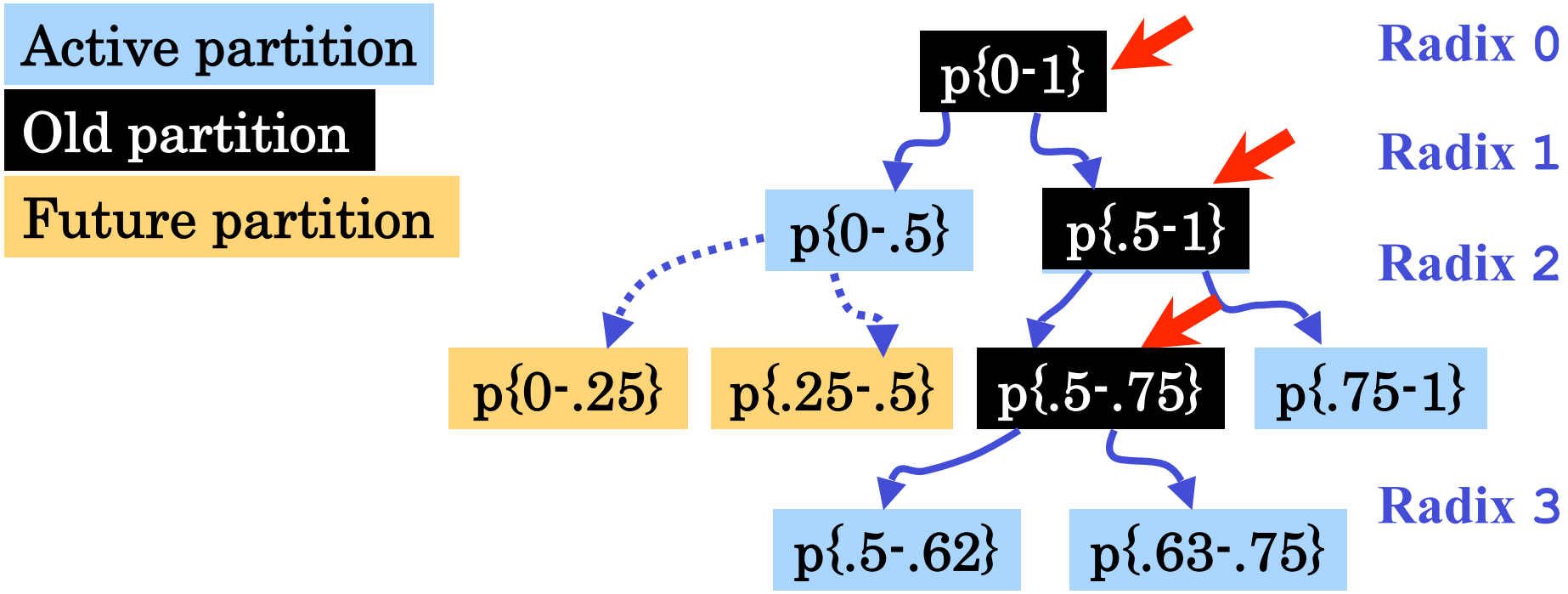
- Introduction
- Related work
- GIGA+ in action
- GIGA+ techniques
  - Two key architectural features
- Status and summary

GIGA+ Design Overview

- How to partition the directory using an index that provides high throughput?
  - Highly decentralized: Decentralized splitting
  - Load-balanced: Hash the key

- How to tolerate rapid changes to the metadata mapping?
  - Indexing technique that tolerates the use of stale metadata information

Growth of the GIGA+ index



- Each server splits its partition when the partition is full, without telling other servers

GIGA+ indexing: Decentralized

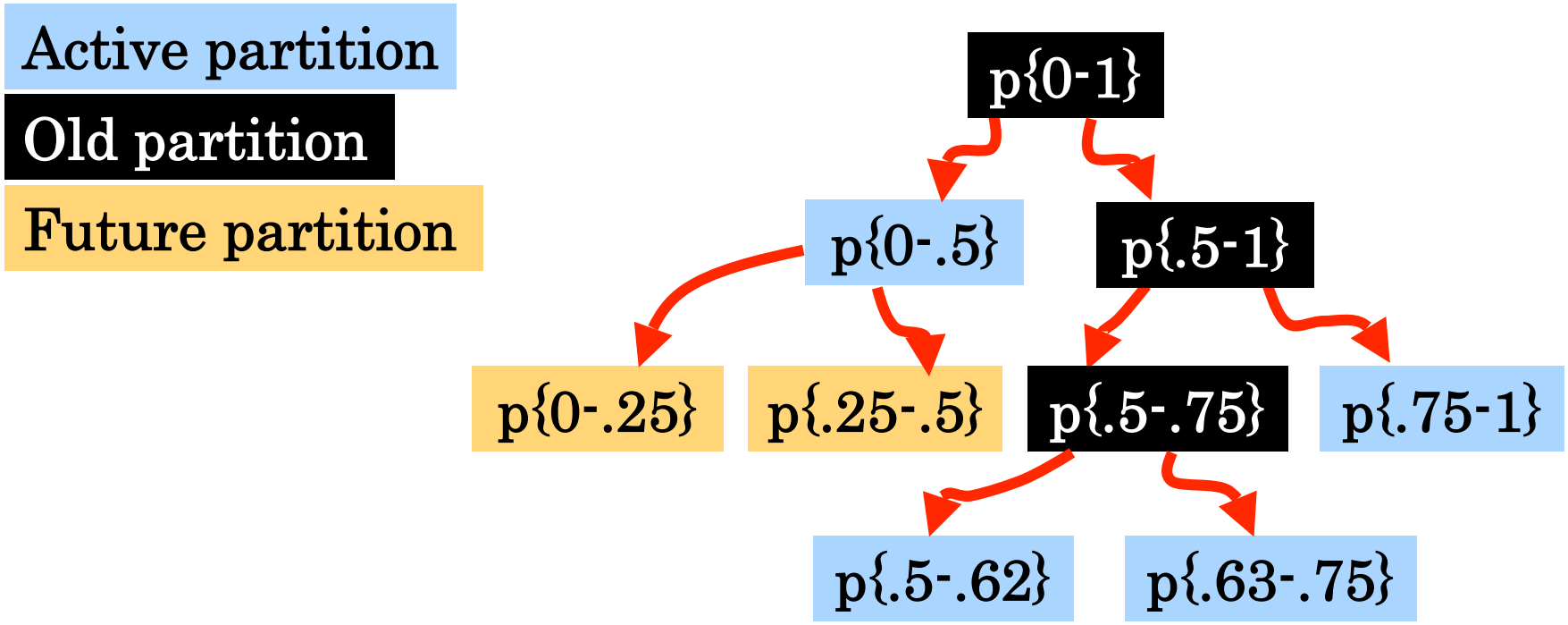
- Highly decentralized and parallel growth of the index
- GIGA+ partitions uniformly over all servers
  - Servers perform their split operation locally
  - Metadata updates happen only at splitting server
- Benefits: Reduced synchronization overhead
  - No immediate synchronization with clients/servers

GIGA+ Design Challenges

- How to partition a directory over many servers?
  - Completely decentralized splitting for maximum concurrency

- How do clients get up-to-date metadata that maps a partition to the server?
  - Tolerate high changes to the metadata mappings

How do clients reach "right" server?

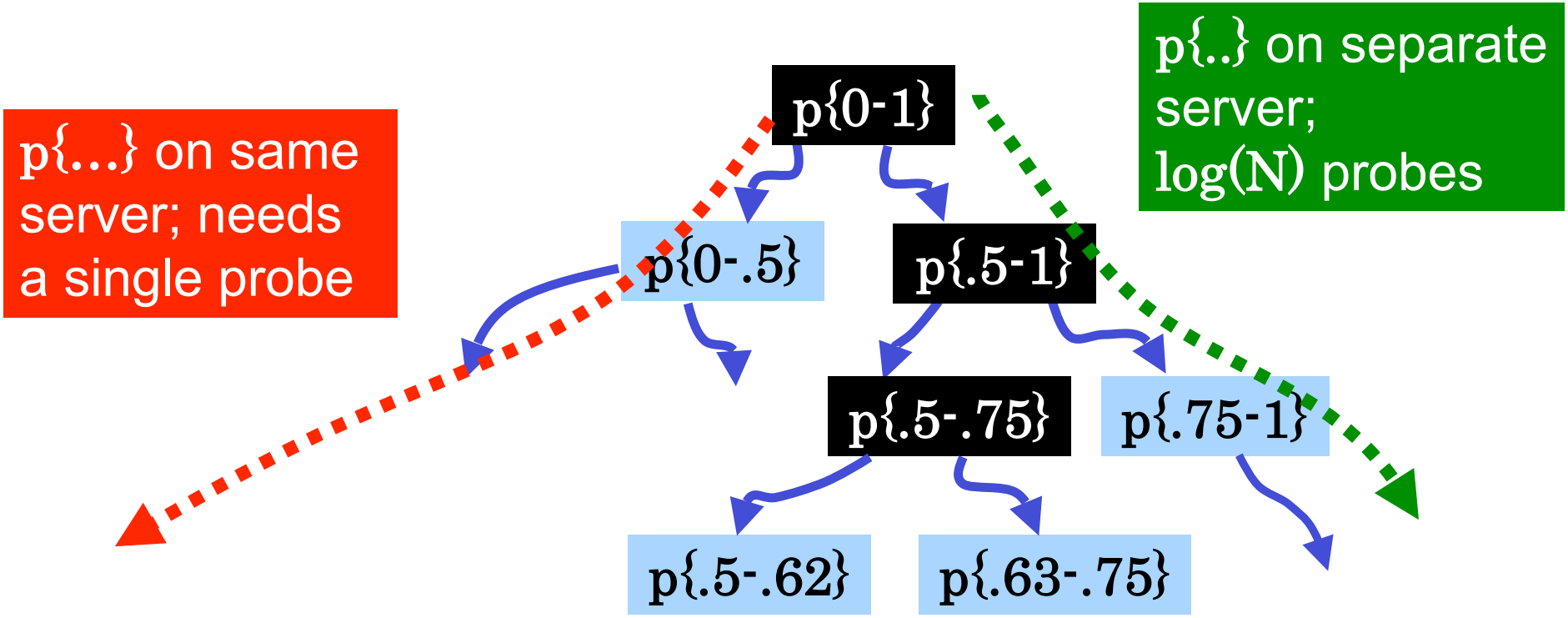


- Caching all the mapping **(links)** is ineffective
  - At high insert rates, mapping changes fast
  - Keeping an *"always consistent"* cache is expensive

GIGA+ clients: Use stale metadata

- Clients use stale partition-to-server mapping
  - Correctness despite out-of-date metadata
- Servers keep “split history” for all its partitions
  - Captures the growth of the partition
  - “history” = {new_partition, new_server, …}
- Use the “split history” to update client cache
  - Server replies with “history” of the partition that the client was looking for

Cost of using stale metadata map



- At most, log (# of partitions) extra hops
  - Lookup might traverse a path up or down a path

GIGA+ Design Summary

- Completely decentralized splitting for maximum concurrency
  - Each server splits a partition when it wants, without synchronizing with the rest of the system
- Indexing technique that allows use stale metadata mapping at clients
  - Servers update clients' mapping information

Outline

- Introduction and motivation
- Related work
- GIGA+ in action
- GIGA+ techniques
- Status and summary
  - Prototype implementation and evaluation

Implementation status

- Building prototype in PVFS
  - Open-source, user-level cluster file system
    - PVFS stores directories on a single server
- Approach
  - Implements FS operations as “state-machines”
    - Add partition splitting, client updates
  - PVFS does not always have a consistent cache
    - Clients cache the “mapping information”
    - Servers keep “split history” as an attribute of the partition

Implementation challenges

- Reducing the extra hops from using stale info
- Efficient representation of partition-to-server metadata mapping
- “Request storm” prevention to avoid overloaded servers
- Avoiding bottlenecks at central MDS

Summary: Pushing Scalability

- Directories that store billion to trillion files and handle >100K operations/second
- Decentralized and parallel growth of directory over many servers
- Indexing technique allows use of stale metadata at the clients
  - Servers update the clients' metadata maps