北京大學
PEKING UNIVERSITY

BCStore: Bandwidth-Efficient In-memory KV-Store with Batch Coding

Shenglong Li, Quanlu Zhang, Zhi Yang and Yafei Dai

Peking University

Outline

- Introduction and Motivation
- Our Design
- System and Implementation
- Evaluation

Outline

- Introduction and Motivation
- Our Design
- System and Implementation
- Evaluation

In-memory KV-Store

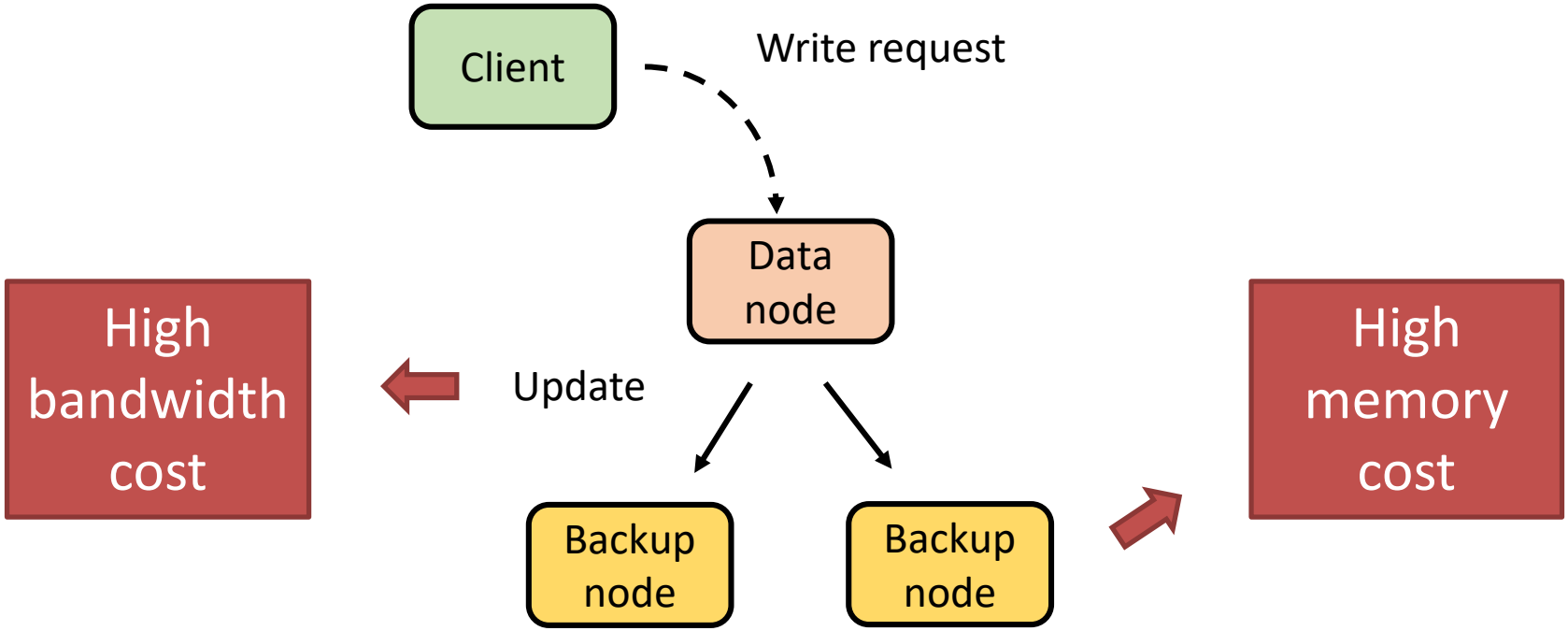
- A crucial building block for many systems
  - Data cache (e.g. Memcached and Redis in Facebook, Twitter)
  - In-memory database

- Availability is important for in-memory KV-Stores
  - Facebook reports that it takes 2.5-3 hours to recover 120GB data of an in-memory database from disk to memory

Data redundancy in distributed memory is essential for fast failover

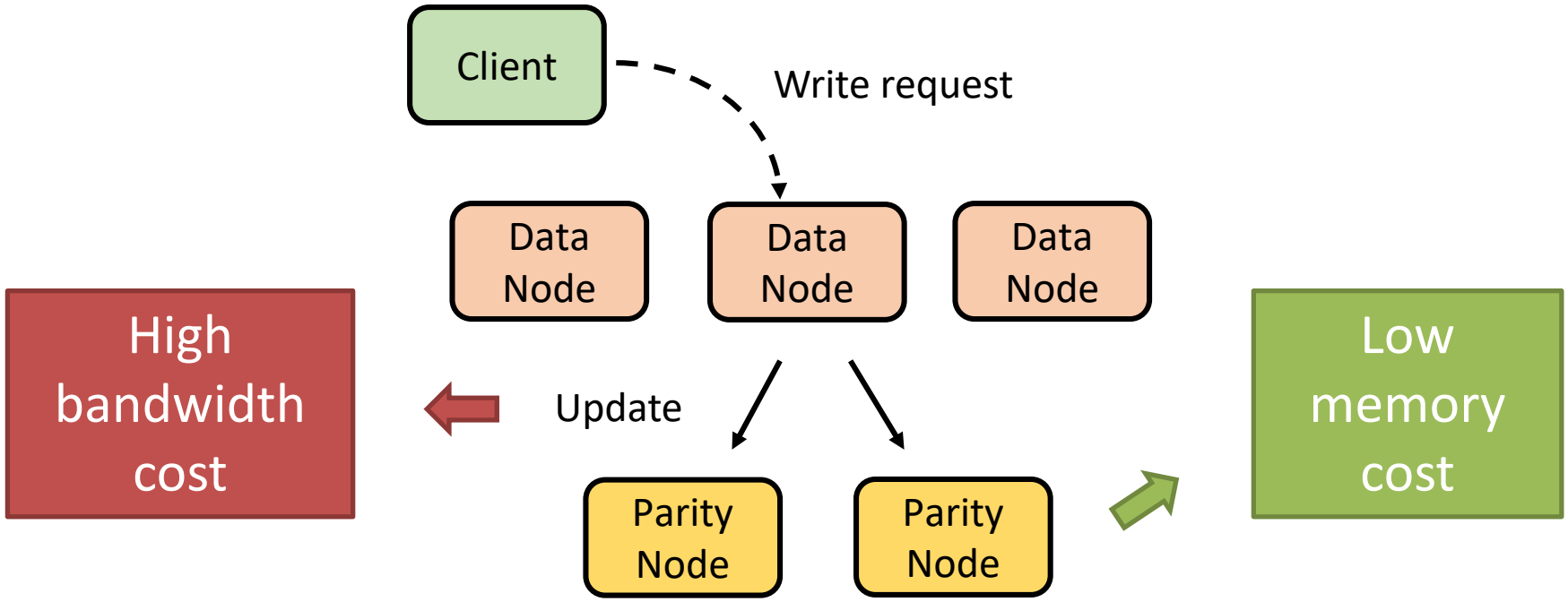
Two redundancy schemes

- Replication is a classical way to provide data availability
  - E.g., Repcached, Redis



Two redundancy schemes

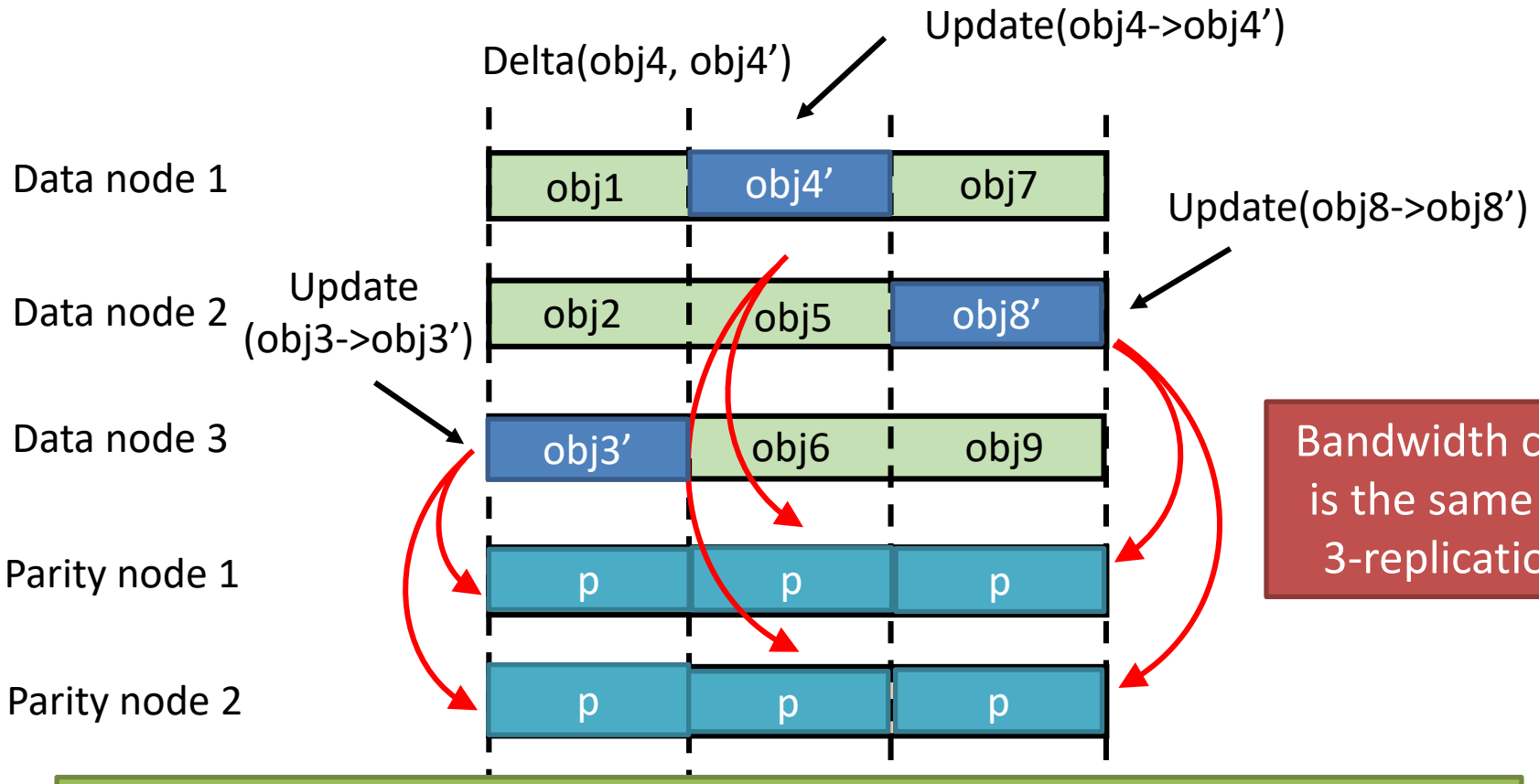
- Erasure coding is a space-efficient redundancy scheme
- The increase of CPU speed enables fast data recovery
  - Encoding/Decoding rates can reach 40Gb/s on single core [1]



[1] Efficient and Available In-memory KV-Store with Hybrid Erasure Coding and Replication, FAST '16

In-place Update

- A traditional mechanism for encoding small objects



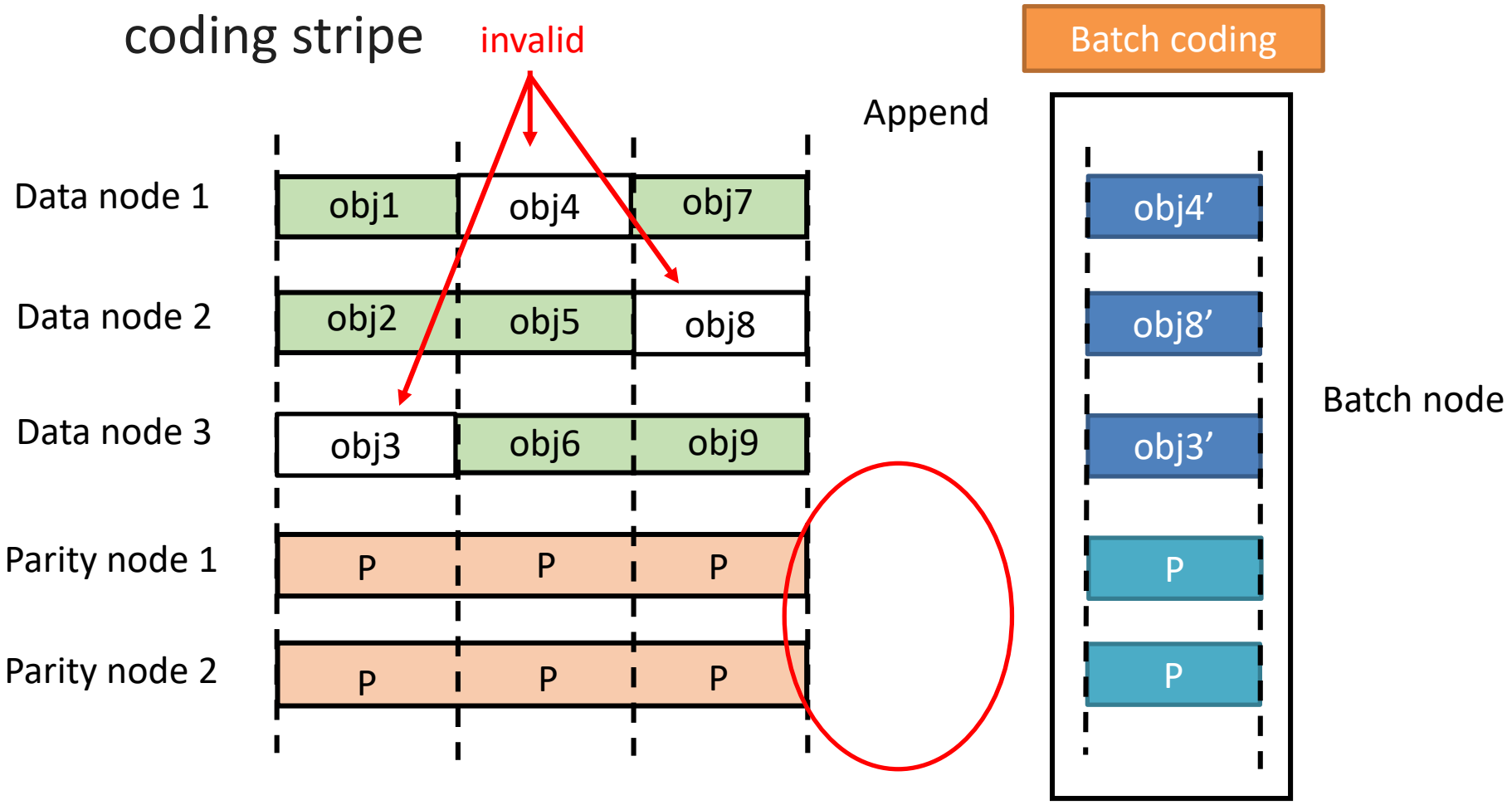
Our goal: both memory efficiency and bandwidth efficiency

Outline

- Introduction and Motivation
- Our Design
- System and Implementation
- Evaluation

Our Design

- Aggregate write requests and encode objects in a new coding stripe



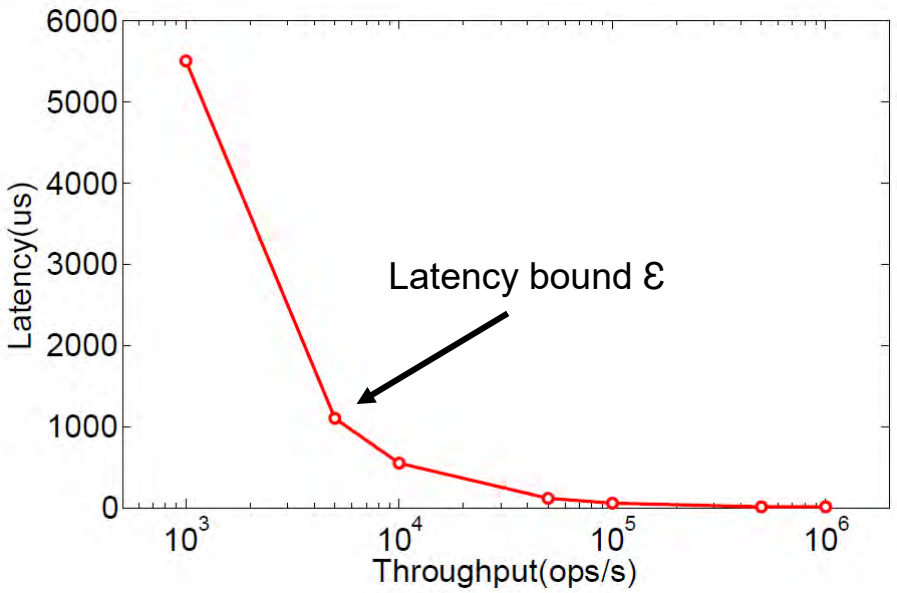
Latency Analysis

- Batch coding induces extra request waiting time
- Formalize the waiting time W

$W = f(T, k)$

Request throughput

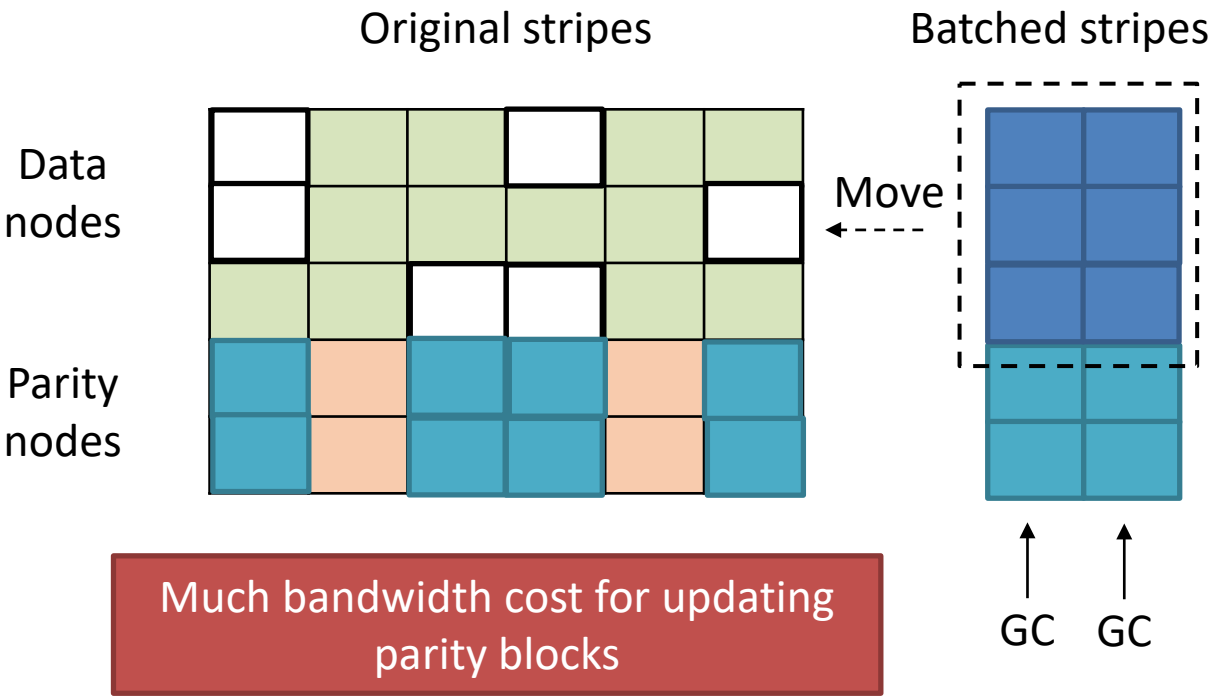
number of data nodes



K = 3

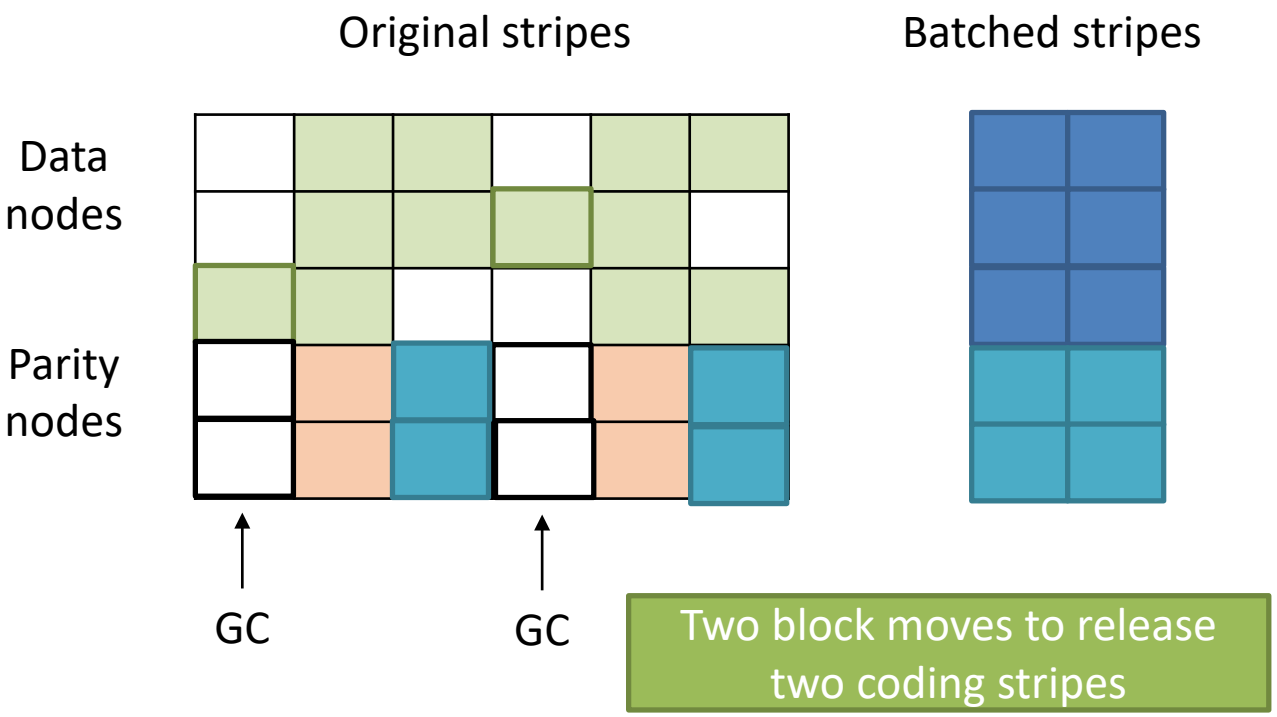
Garbage Collection

- Recycle updated or deleted blocks and release extra parity blocks
- Move-based garbage collection



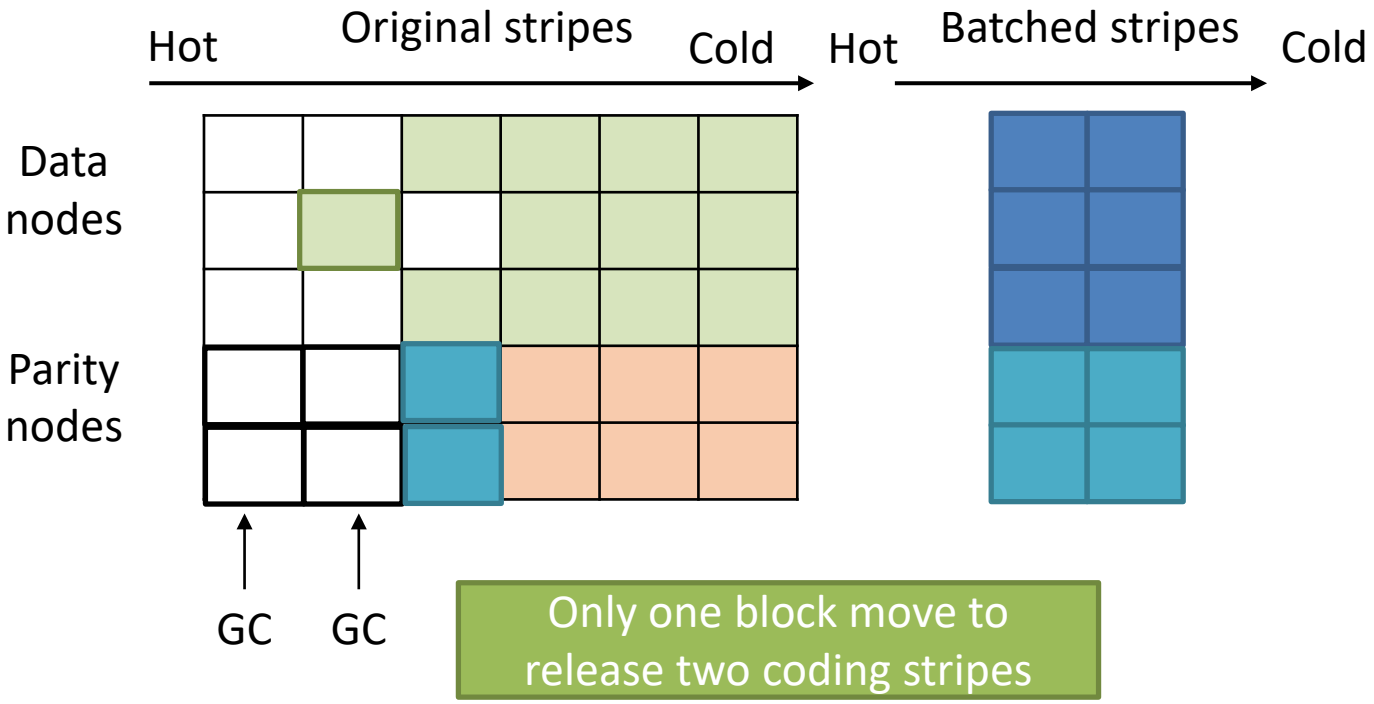
Garbage Collection

- How to reduce the GC bandwidth cost?
  - Intuition: GC the stripes with the most invalid blocks
- Greedy block moving



Garbage Collection

- How to further reduce block move?
  - Intuition: make the updates focus on few stripes
- Popularity-based data arrangement



Bandwidth Analysis

- **Theorem**

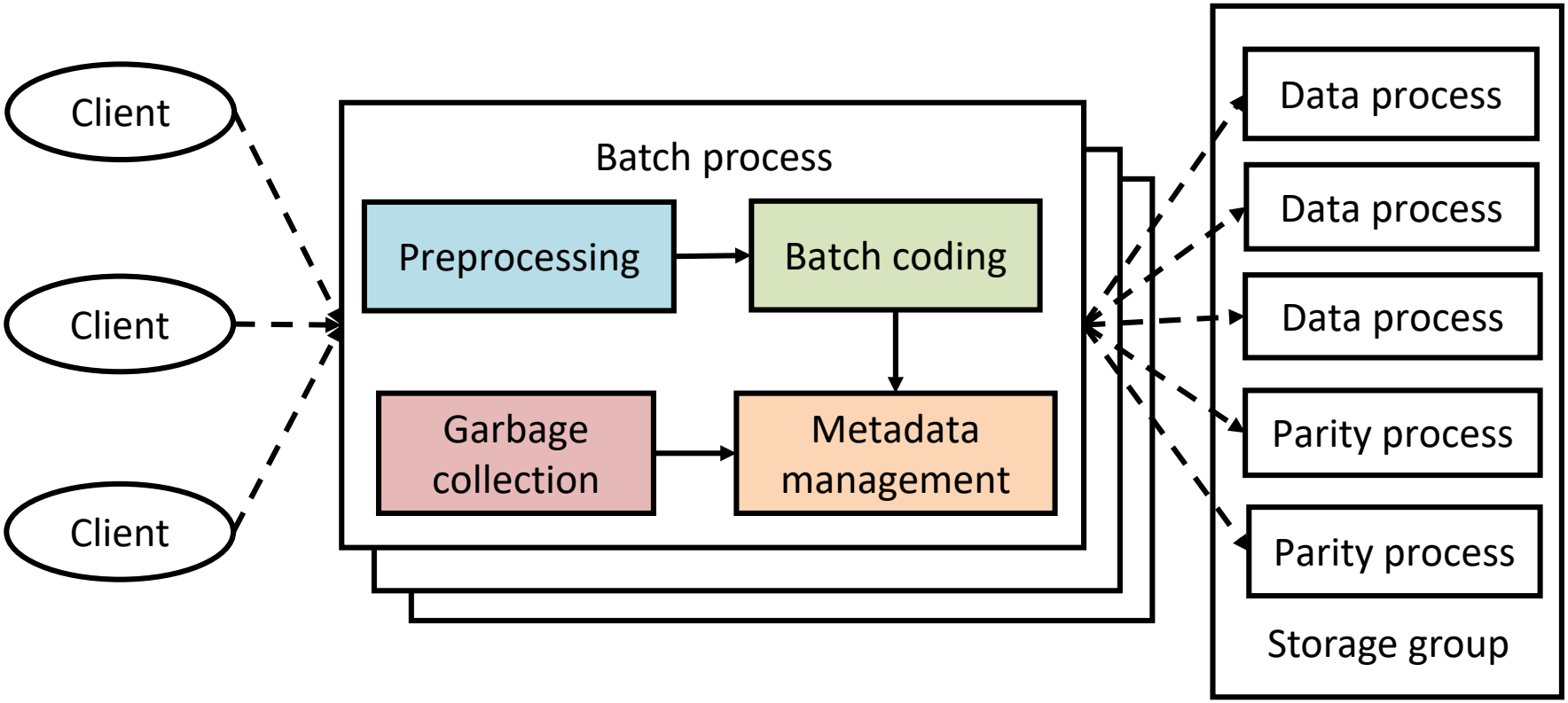
GC bandwidth + Coding bandwidth <= In-place update bandwidth

Detailed proof can be found in our paper

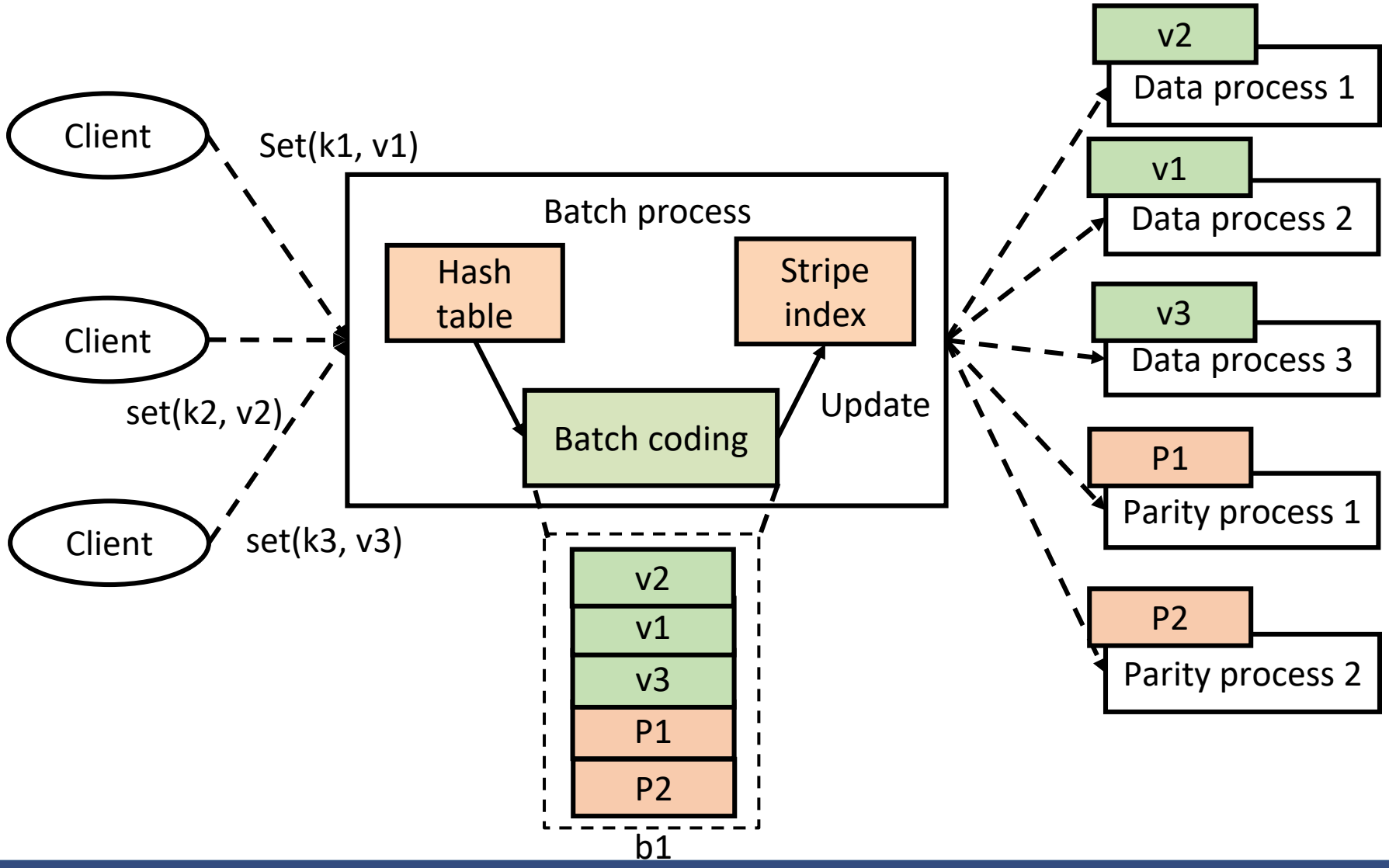
Outline

- Introduction and Motivation
- Our Design
- System and Implementation
- Evaluation

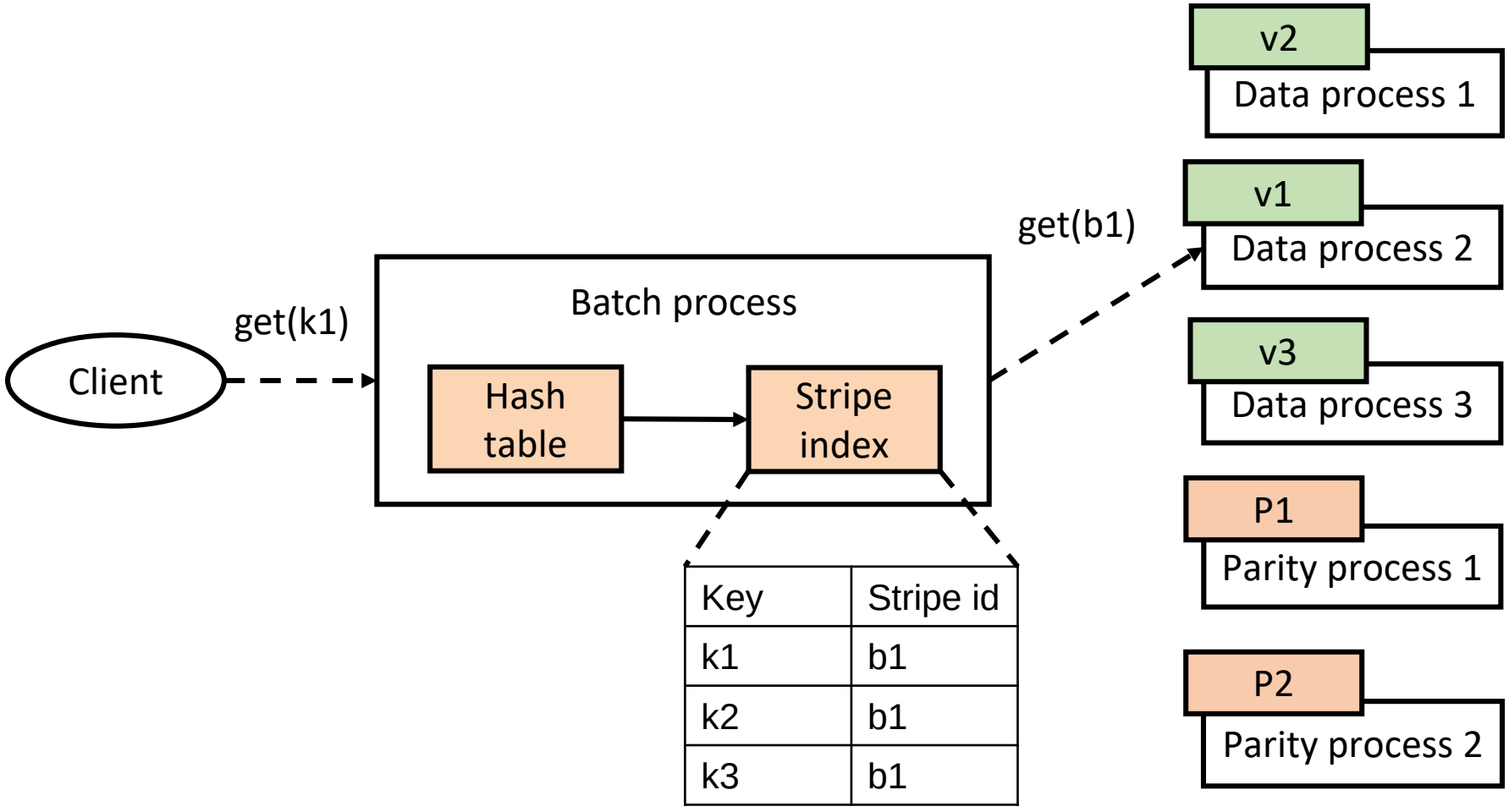
System Architecture



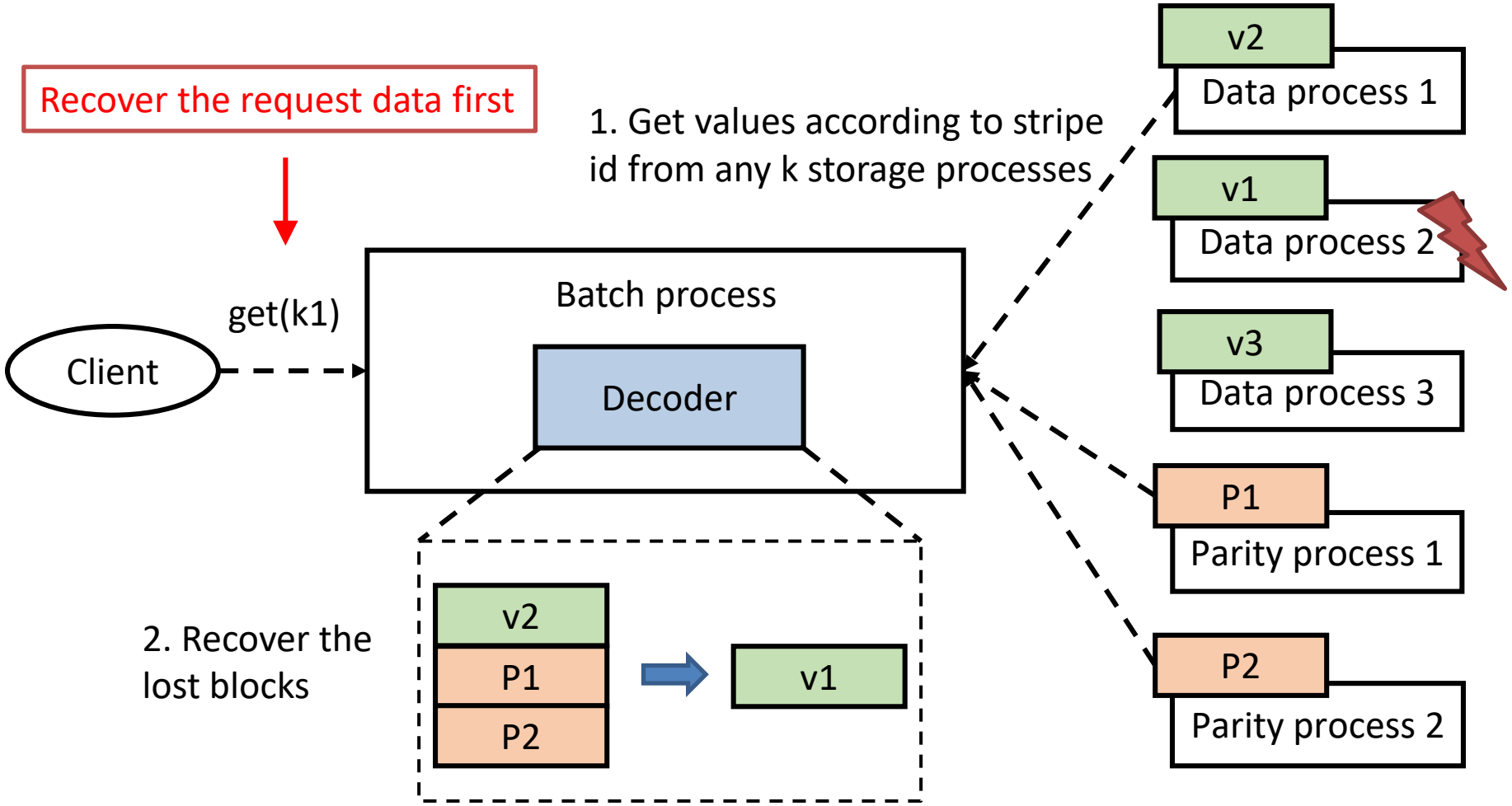
Handle Write Requests



Handle Read Requests



Recovery


Recover the request data first
1. Get values according to stripe id from any k storage processes
v2
Data process 1
v1
Data process 2
v3
Data process 3
P1
Parity process 1
P2
Parity process 2
get(k1)
Client
Batch process
Decoder
2. Recover the lost blocks
v2
P1
P2
v1

Outline

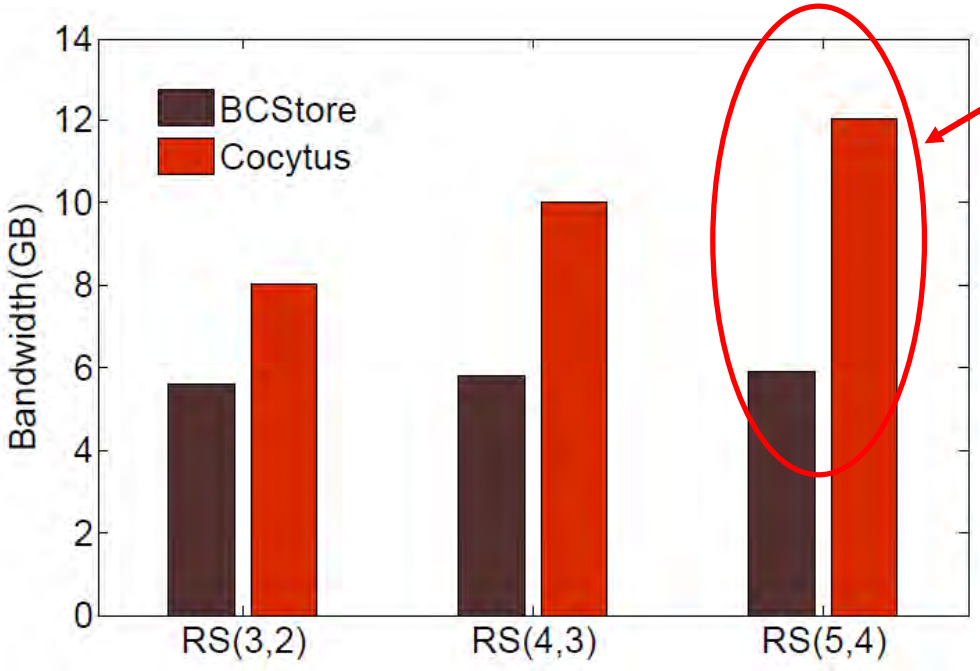
- Introduction and Motivation
- Our Design
- System and Implementation
- Evaluation

Evaluation

- Cluster configuration
  - 10 machines running SUSE Linux 11 containing 12 * AMD Opteron Processor 4180 CPUs
  - 1Gb/s Ethernet
- Targets of comparison
  - In-place update EC (Cocytus[1])
  - Replication (Rep)
- Workload
  - YCSB with different key distributions
  - 50%:50% read/write ratio

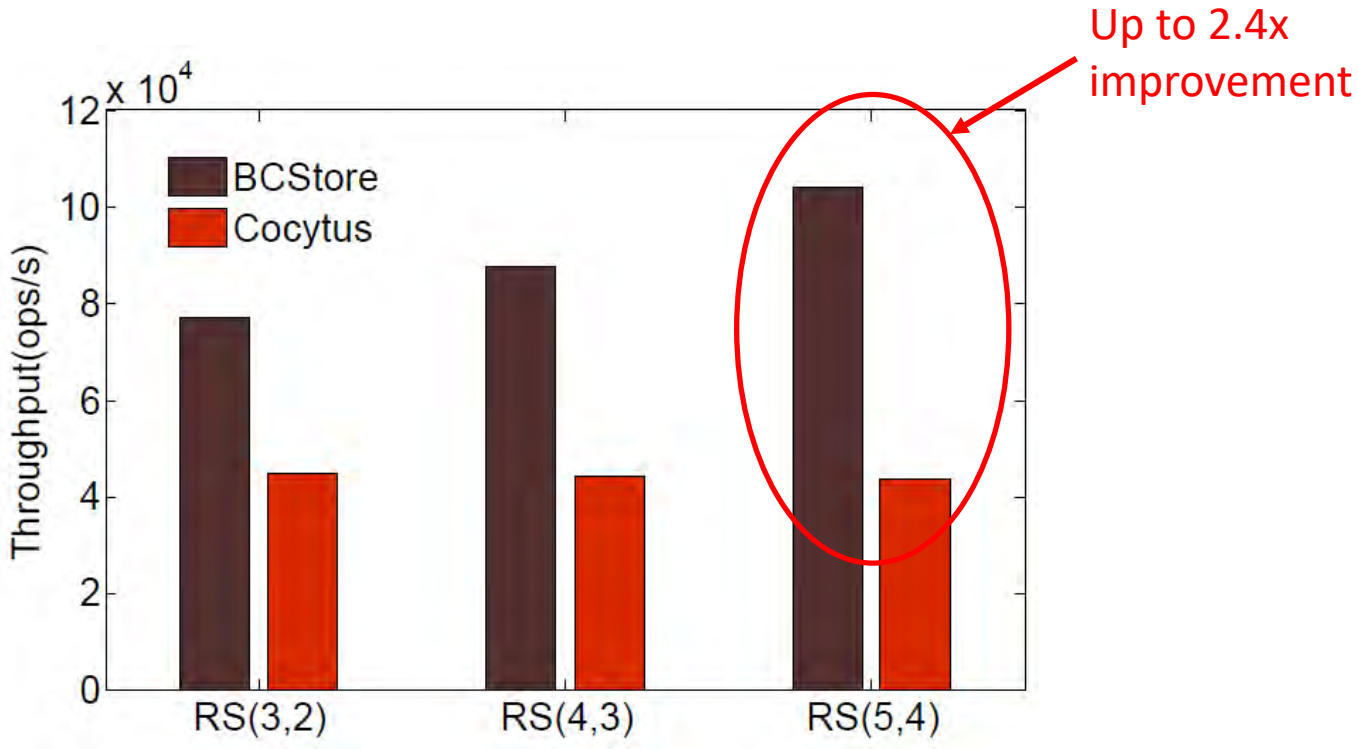
[1] Efficient and Available In-memory KV-Store with Hybrid Erasure Coding and Replication, FAST '16

Bandwidth Cost



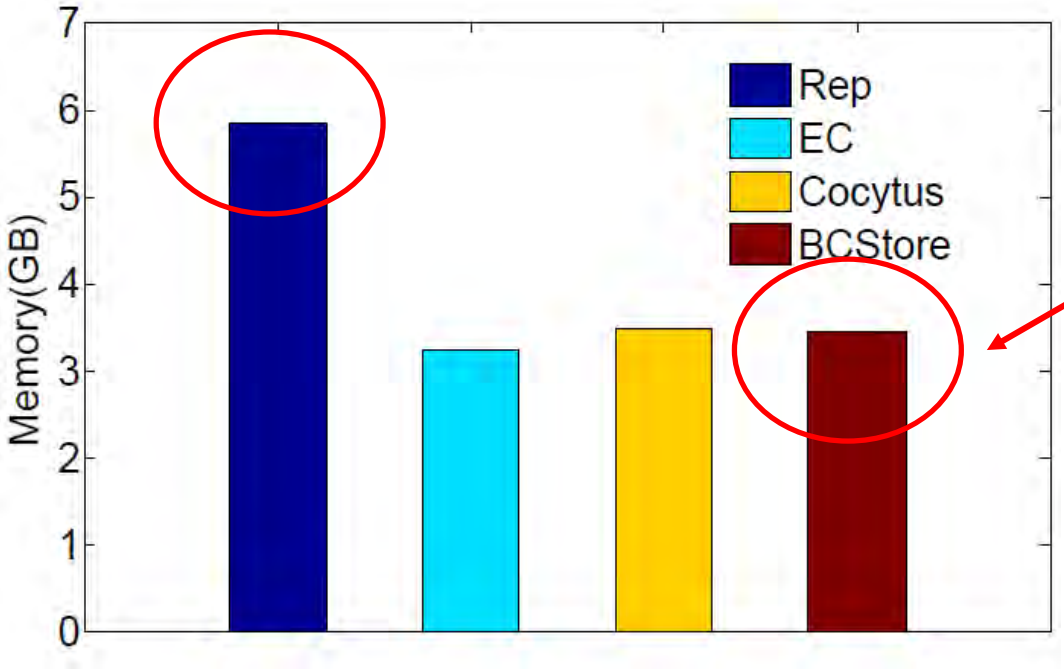
Bandwidth cost for different coding schemes.

Throughput



Throughput performance for different coding schemes.

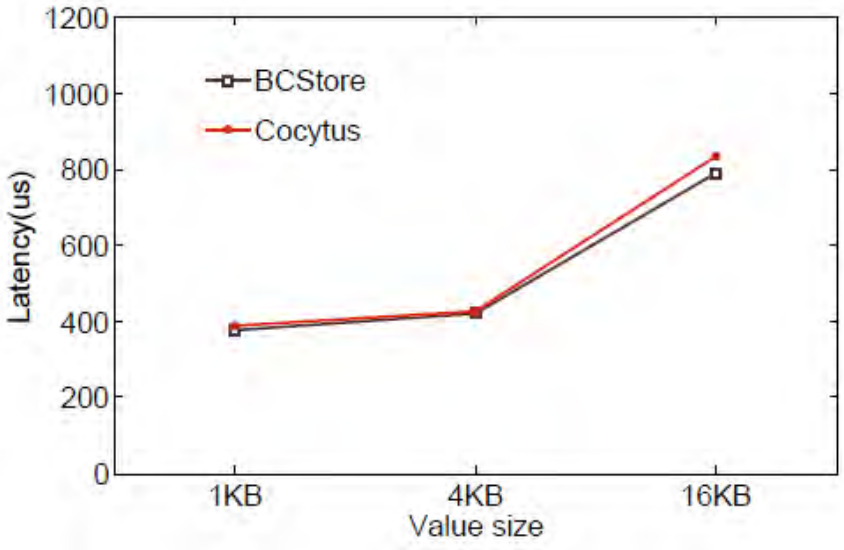
Memory



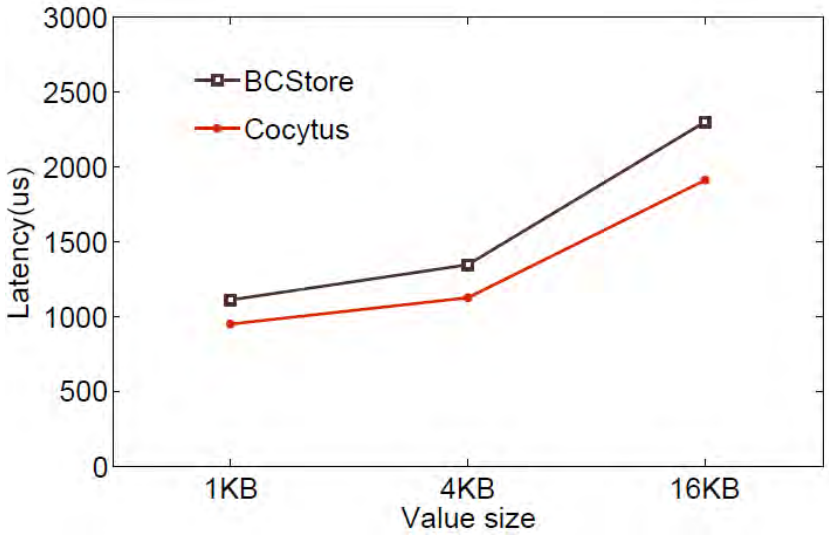
Save up to 41% memory cost

Memory consumption for different redundancy schemes

Latency



Read latency



Write latency

Conclusion

- Efficiency and availability are two crucial features for in-memory KV-Stores
- We build BCStore, an in-memory KV-Store which applies erasure coding for data availability
- We design batch coding mechanism to achieve high bandwidth efficiency for write workload
- We propose a heuristic garbage collection algorithm to improve memory efficiency

Thanks!

Q&A

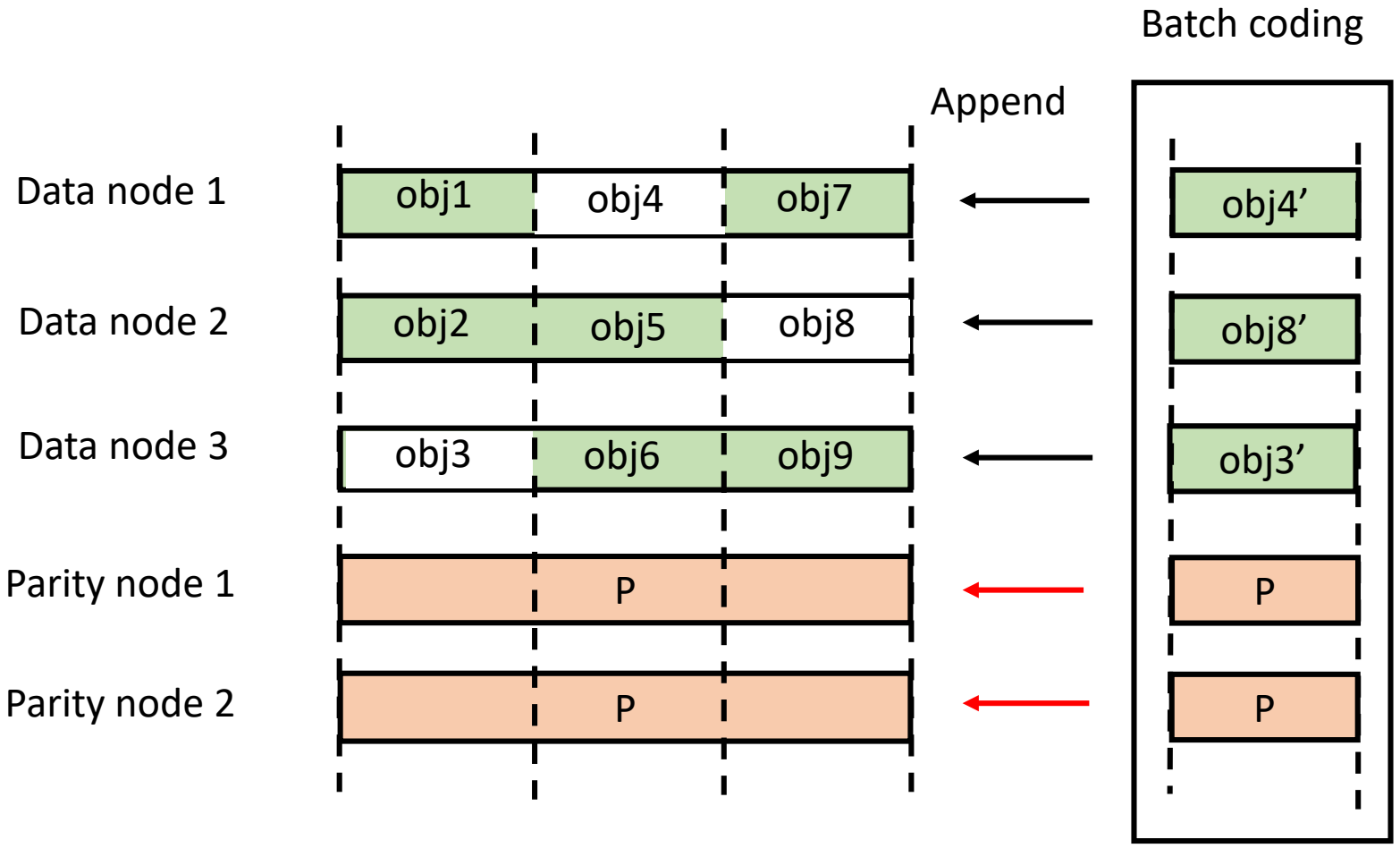
Severity of Bandwidth Cost

- Prevalence of write requests in large-scale web services
  - Peak load can easily run out of network bandwidth and degrade service performance
- Monetary cost of bandwidth becomes several times higher
  - Especially under the commonly used peak-load pricing model
  - Bandwidth amplification would be more serious with the increase of *m* (number of parity servers)
- Budget of bandwidth resource is usually limited in workload-sharing cluster

Our goal: High memory efficiency and bandwidth efficiency

Our Design

- Batch write requests and append a new coding stripe

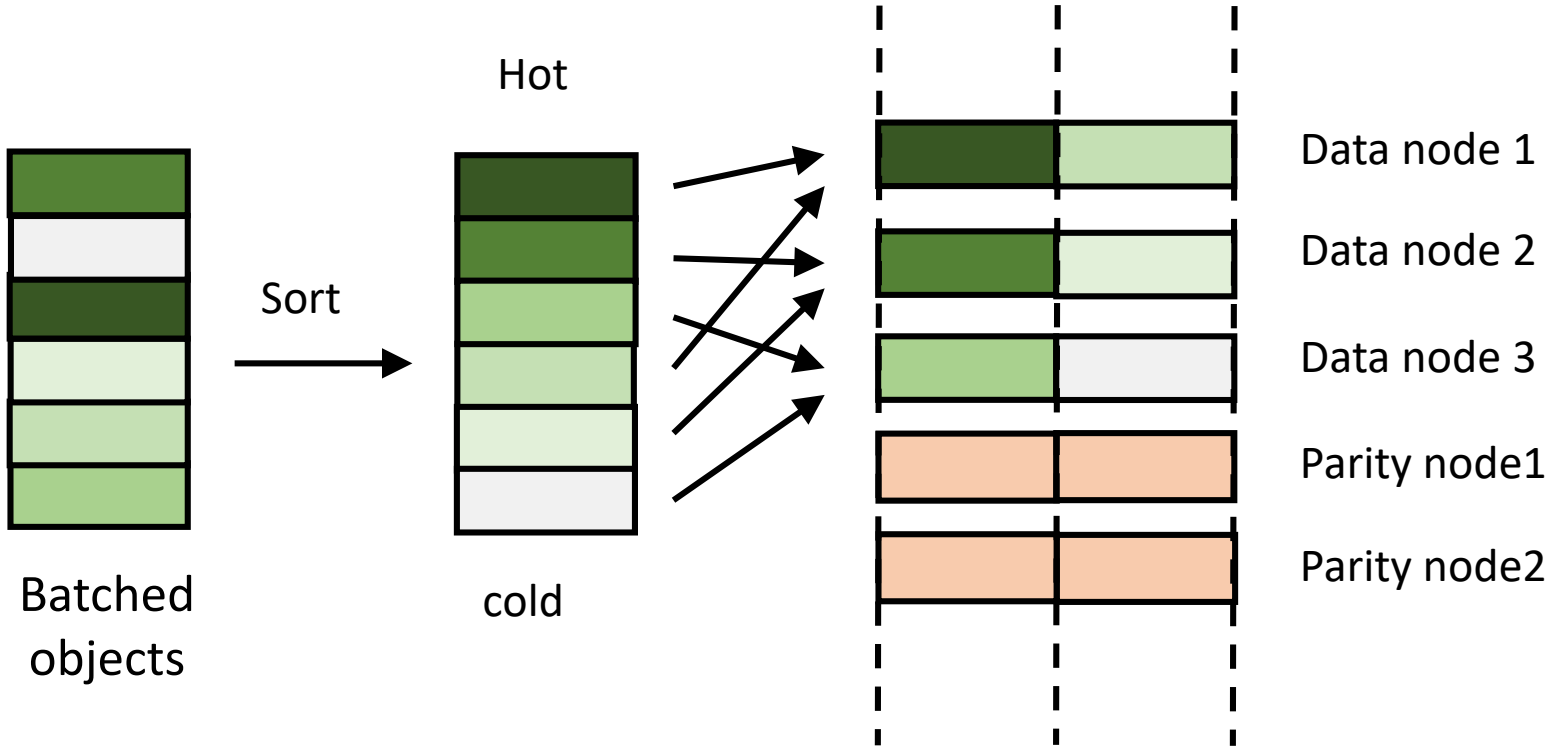


Challenges

- Recycle the memory space of data blocks which are deleted or updated
  - Data blocks and parity blocks are appended to the storage
  - Updated blocks can not be delete directly
- Encode variable-sized data efficiently
  - Variable-sized data can not be appended to previous storage space directly

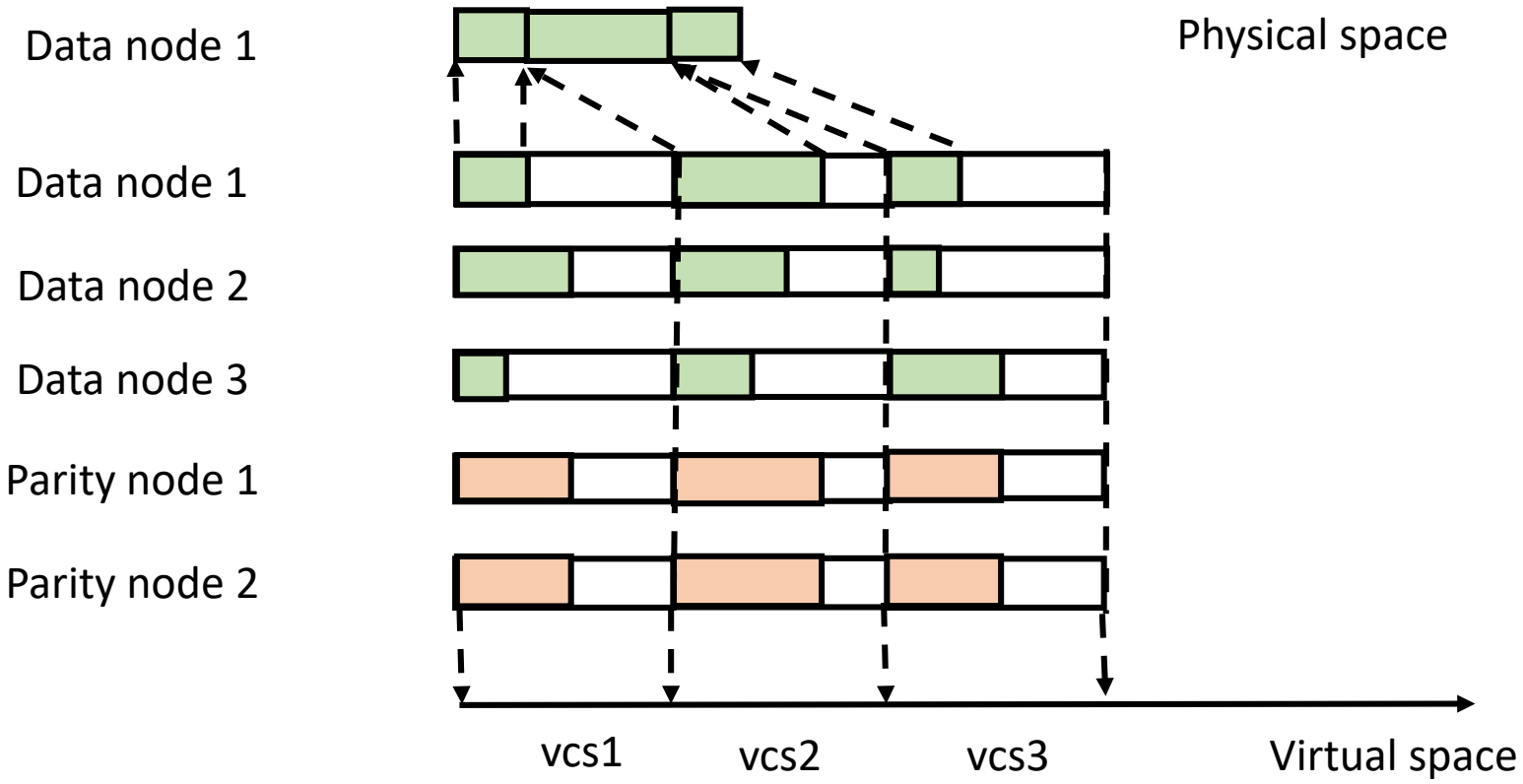
Garbage Collection

- Popularity-based data arrangement

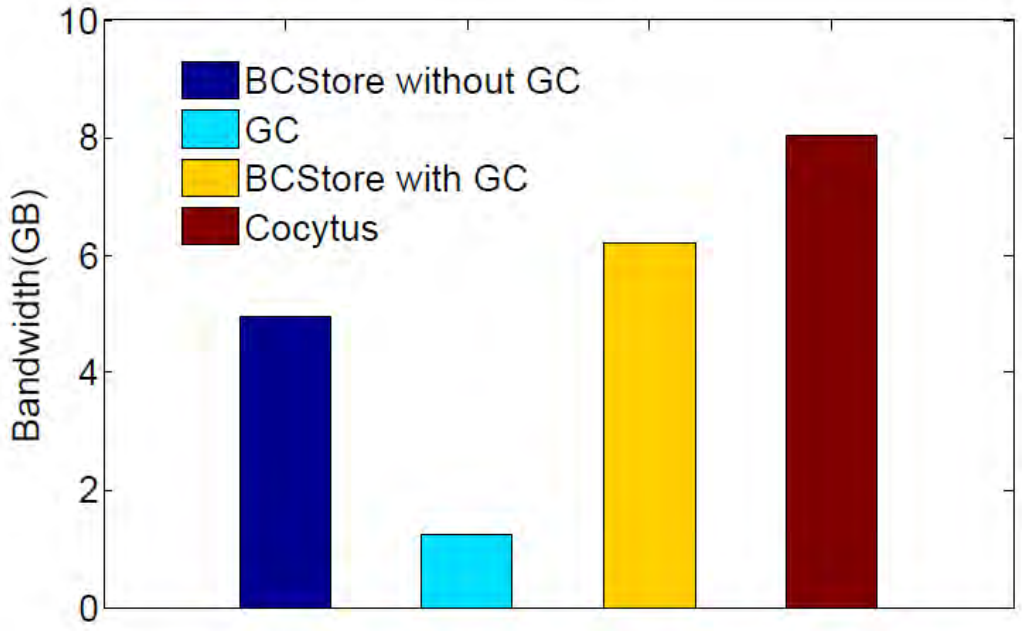


Encoding Variable-size Data

- Virtual coding stripes (vcs)
  - Each virtual coding stripe has a large fixed-length space and is aligned in virtual address

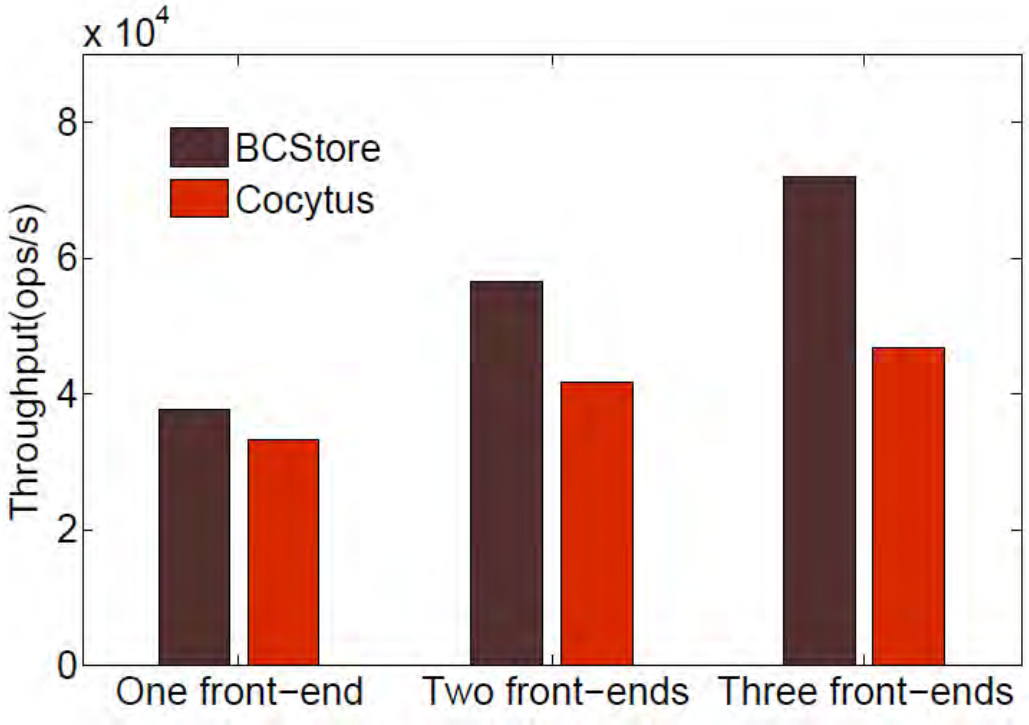


Bandwidth Cost



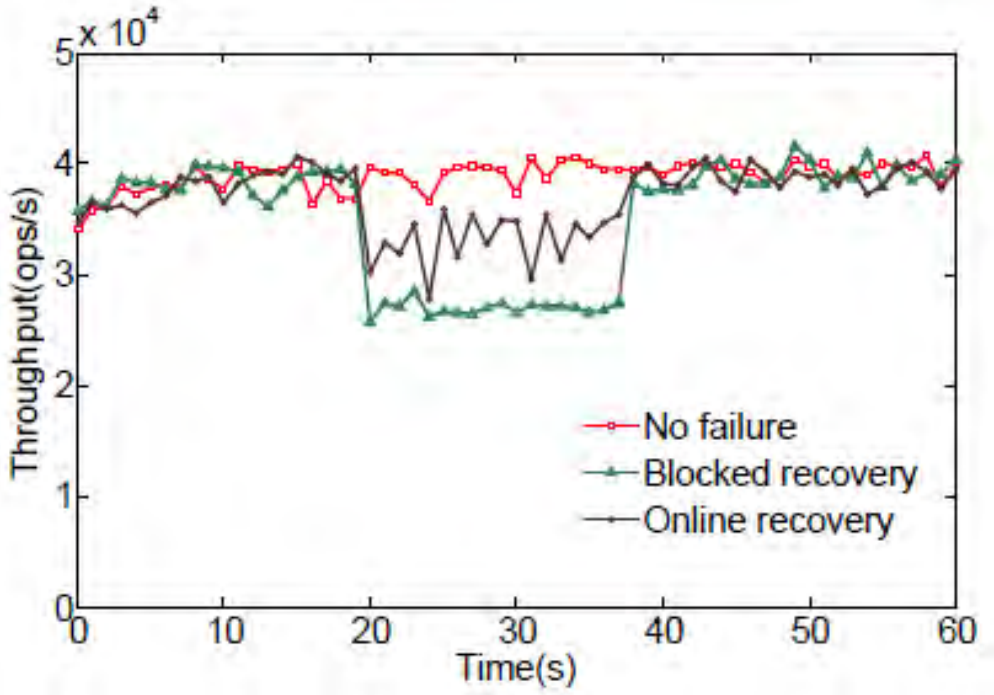
Bandwidth cost for moderate-skewed Zipfian workload (RS(3,2))

Throughput



Throughput performance for moderate-skewed Zipfian workload

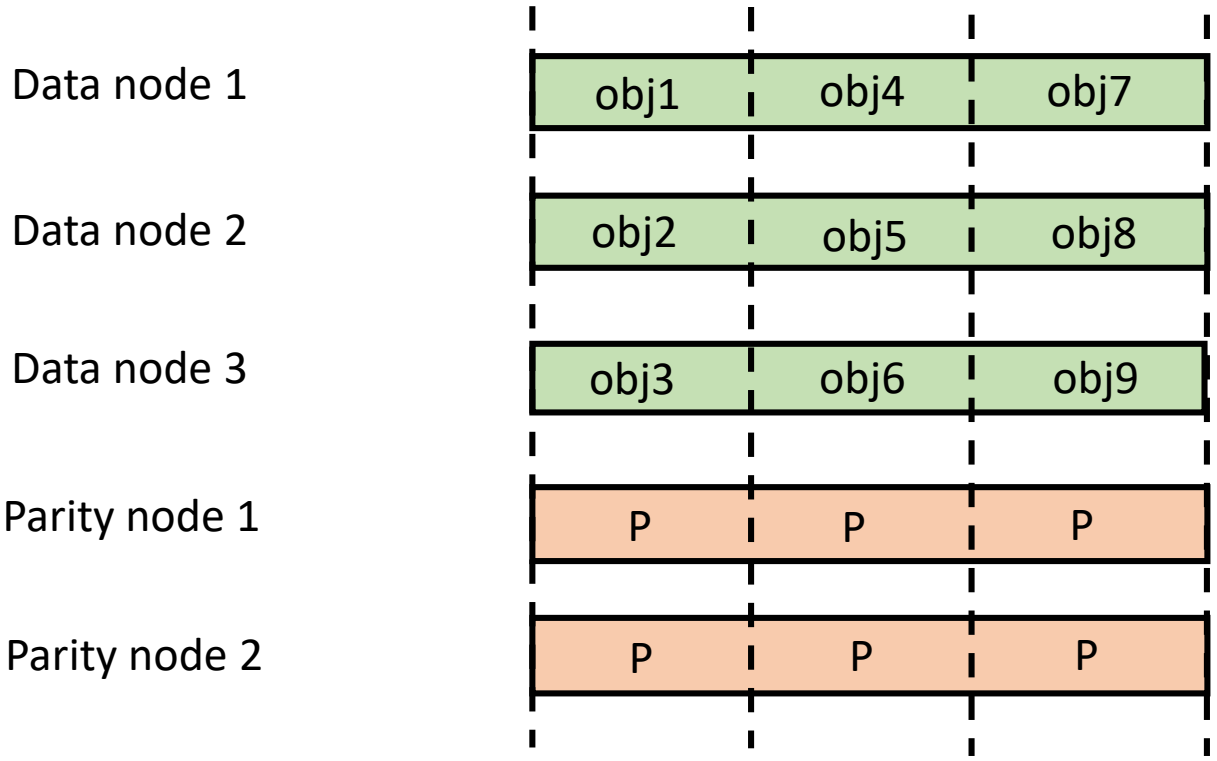
Throughput



Throughput for recovery

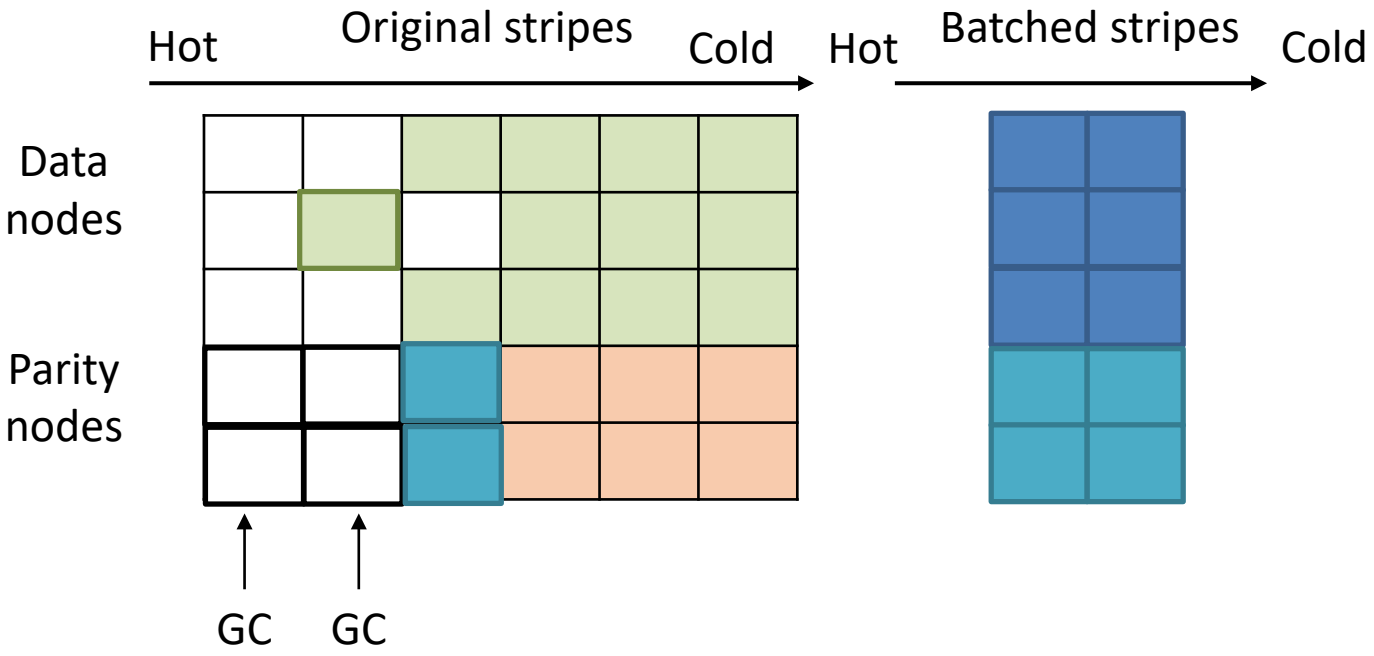
In-place Update

- A traditional mechanism for coding small objects



Garbage Collection

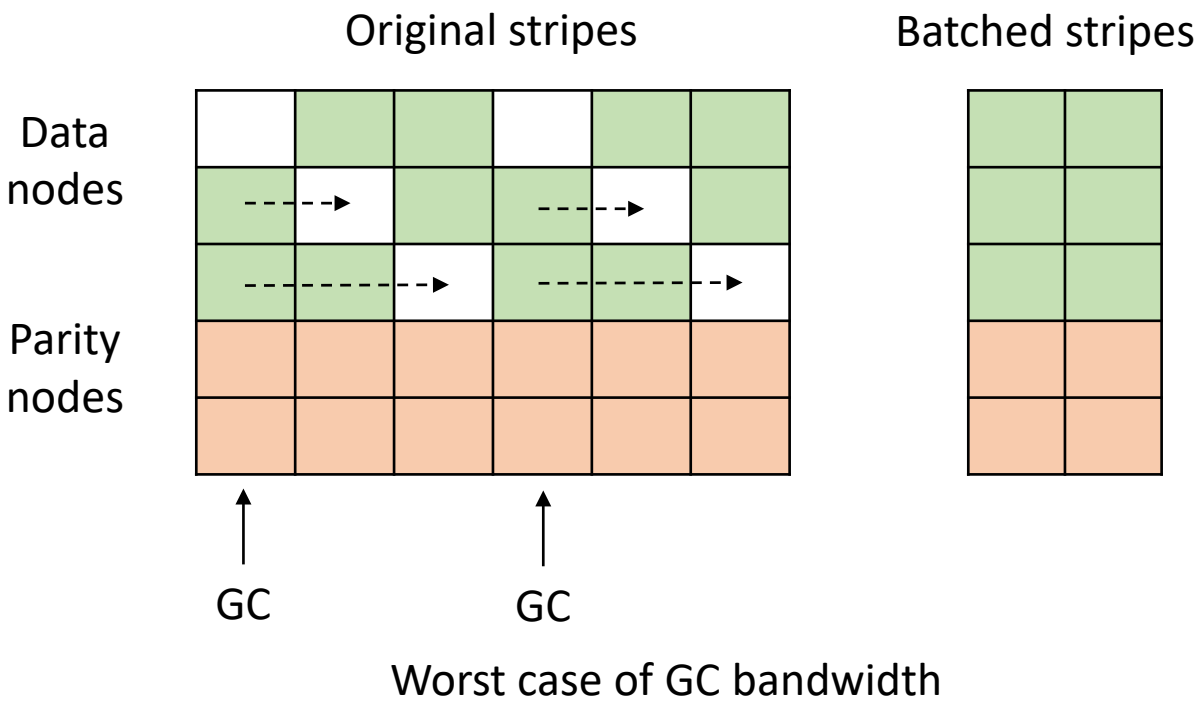
- How to further reduce block move?
  - Intuition: make the updates focus on few stripes
- Popularity-based data arrangement



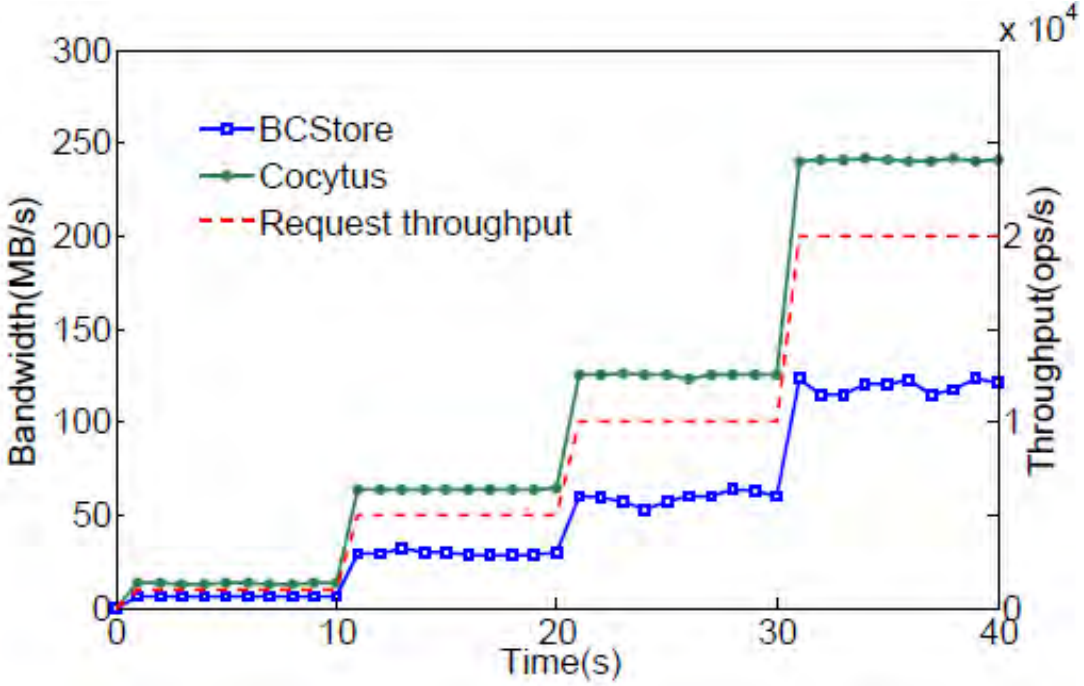
Bandwidth Analysis

- **Theorem**

GC bandwidth + Coding bandwidth <= In-place update bandwidth

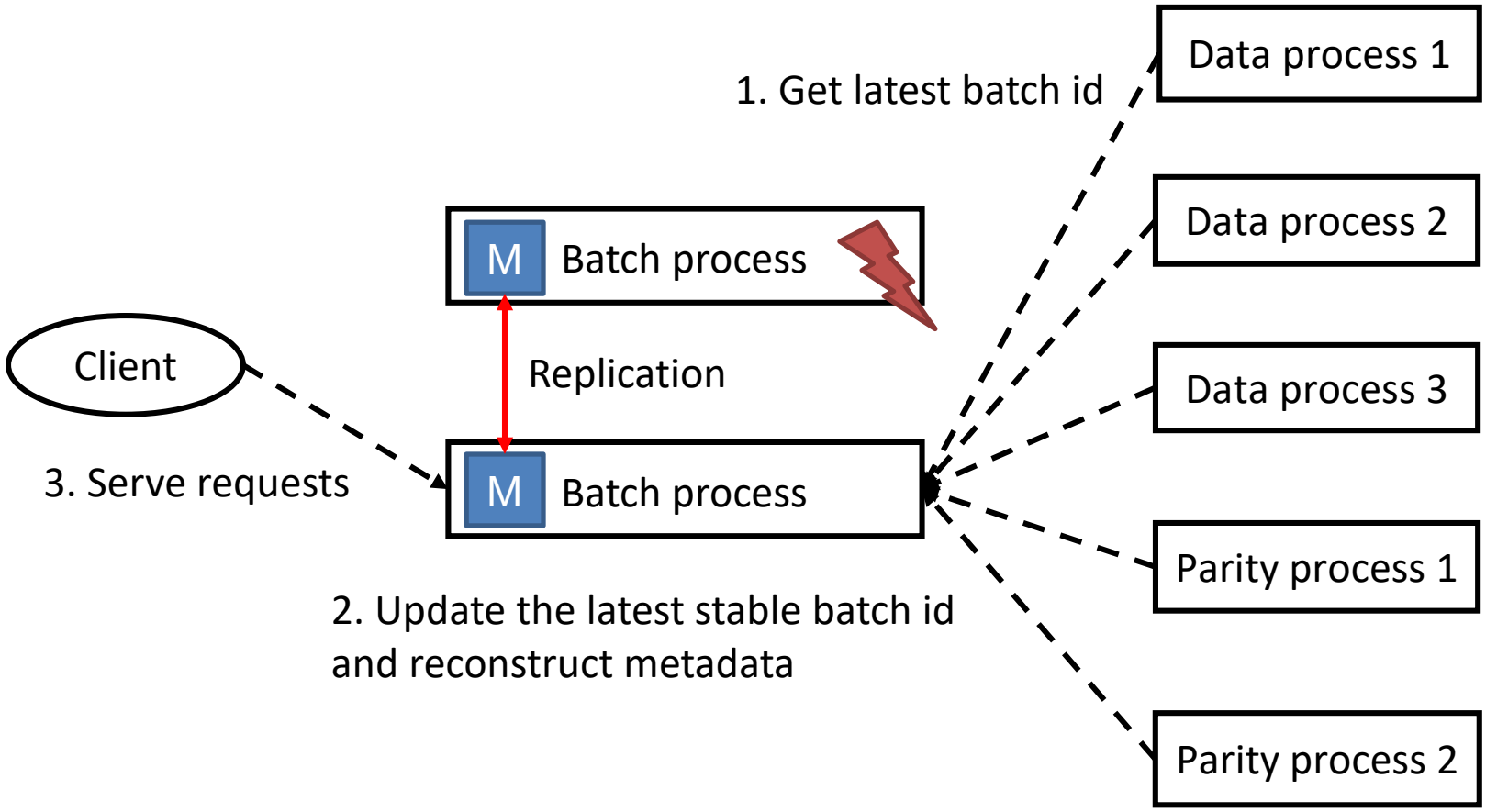


Bandwidth Cost



Bandwidth cost for different throughput. (RS(5,4))

Recovery


1. Get latest batch id
Data process 1
Data process 2
Data process 3
Parity process 1
Parity process 2
M
Batch process
Replication
Client
3. Serve requests
M
Batch process
2. Update the latest stable batch id
and reconstruct metadata