2017 USENIX Annual Technical Conference

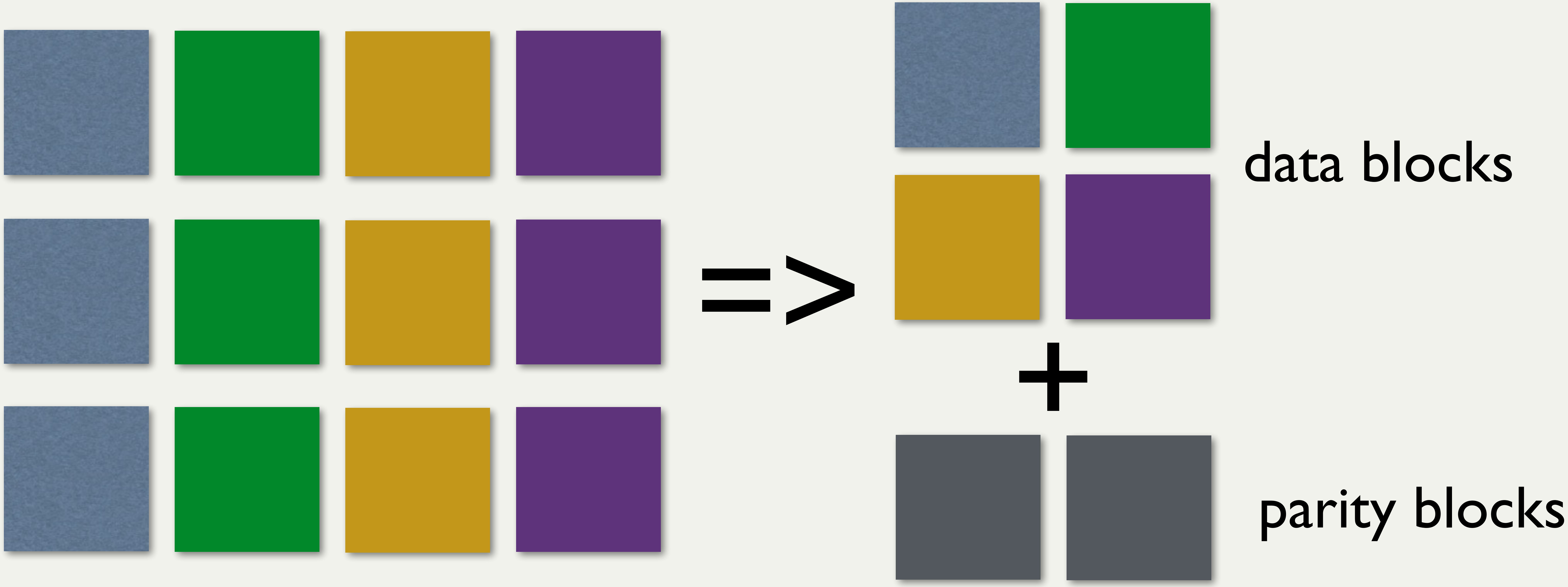
美团云 MEITUAN OPEN SERVICES 技术突破边界 服务传递价值

PARIX: Speculative Partial Writes in Erasure-Coded Systems

Huiba Li*, Yiming Zhang[θ], Zhiming Zhang*, Shengyun Liu[θ], Dongsheng Li[θ], Xiaohui Liu[θ], Yuxing Peng[θ]

*Meituan Open Service, [θ]NUDT

Erasure Coding (EC)



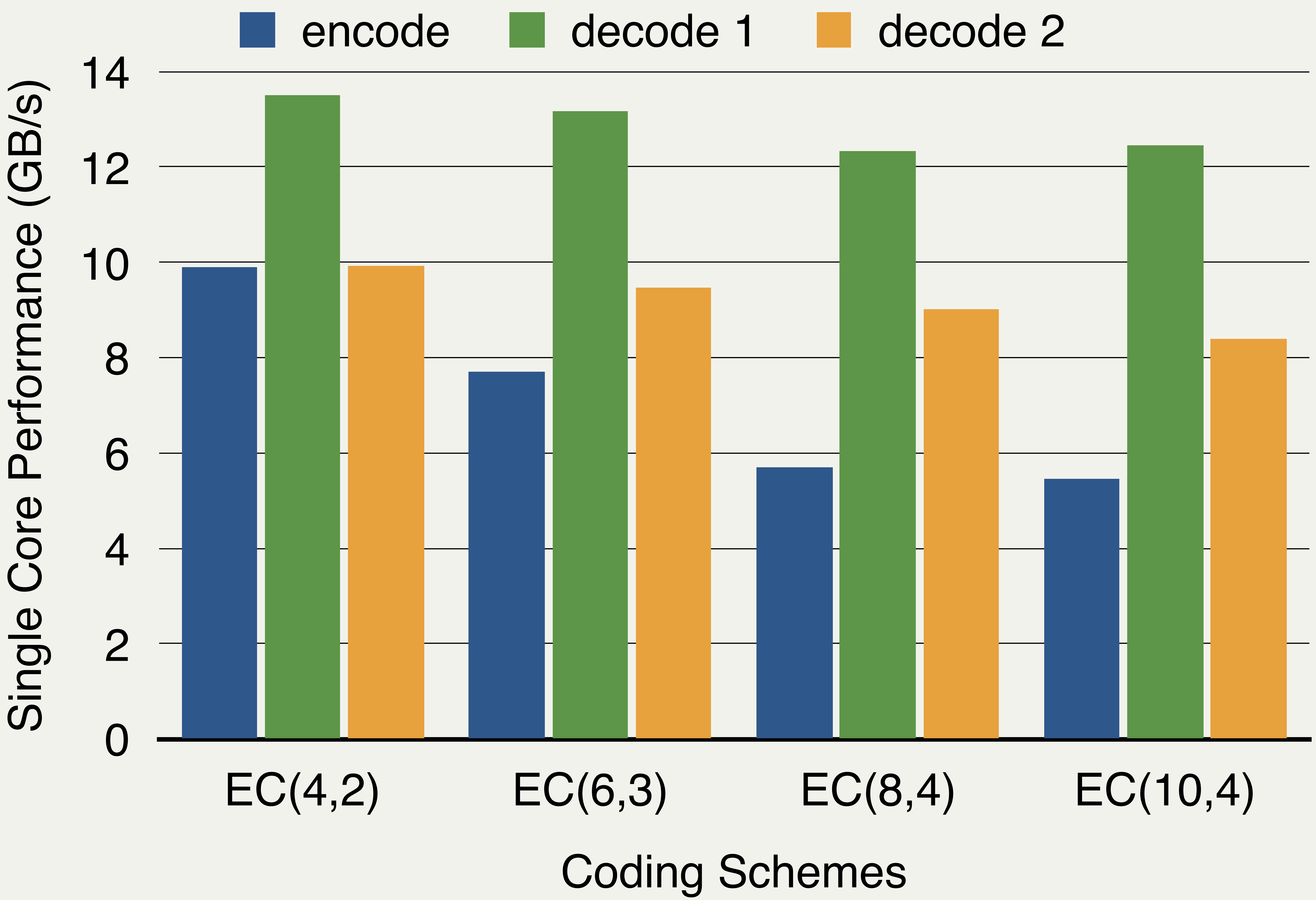
4*3=12
(300% redundancy)

4+2=6
(150% redundancy)

Usage of EC

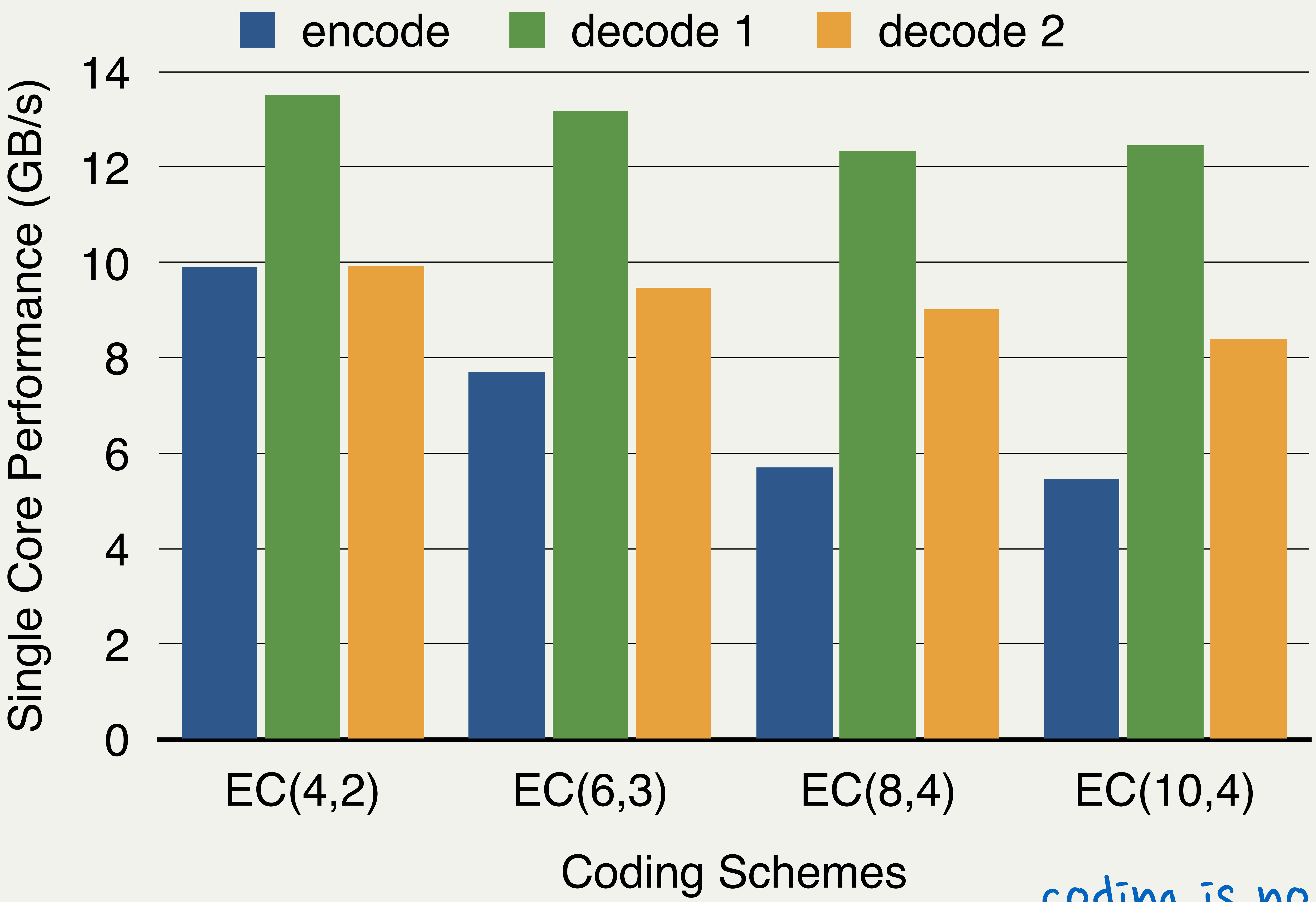
- Widely used in distributed storage systems
  - *Especially large-scale cloud storage services*
- Except in read-write hot storage
- Because of performance
- Overheads include:
  - *Coding calculation*
  - *I/O pattern deterioration, especially for partial writes*

Coding with Vectorial Instructions



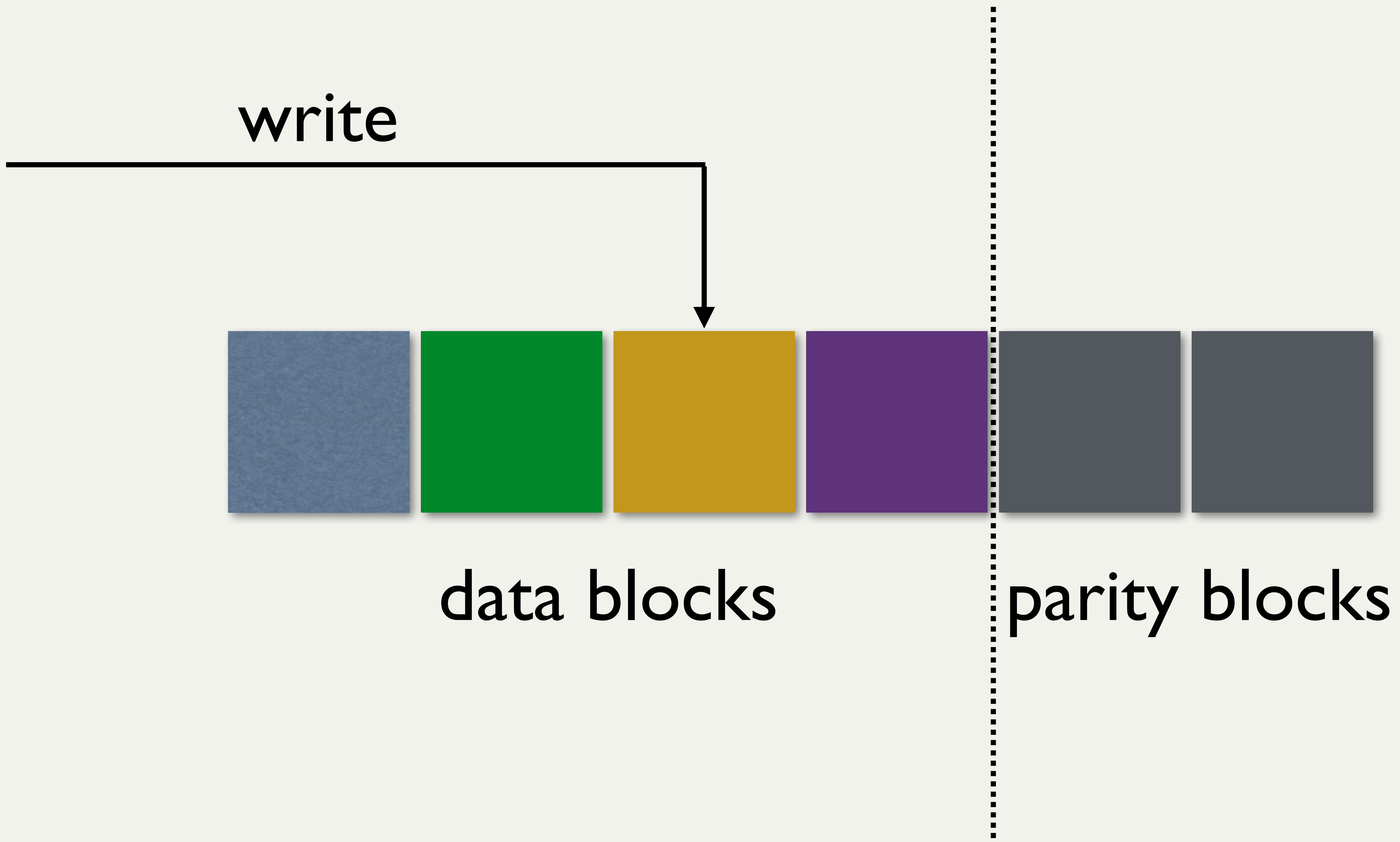
Coding with Vectorial Instructions

美团云
MEITUAN OPEN SERVICES
技术突破边界 服务传递价值

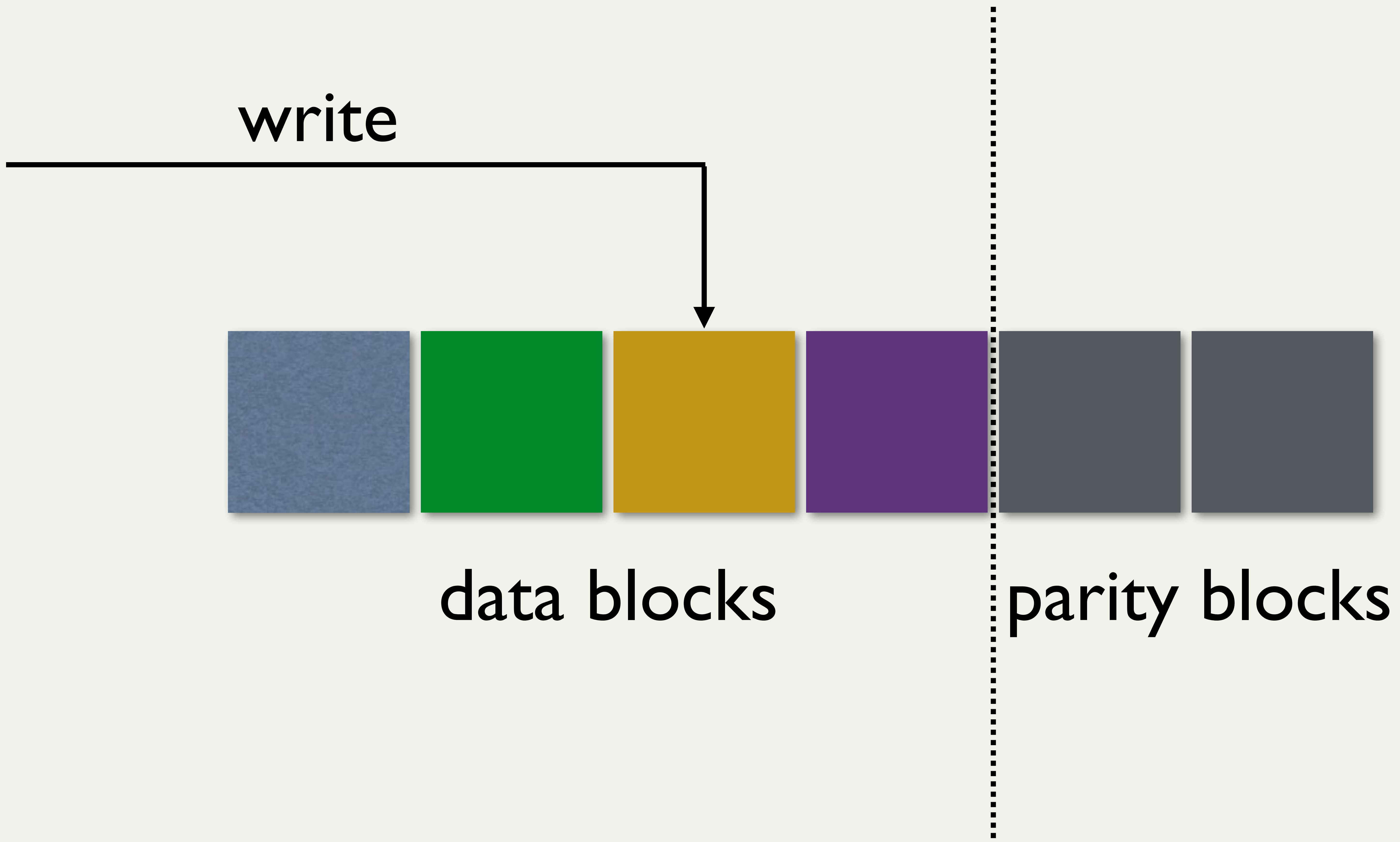


coding is no longer the bottleneck

Partial Write in EC



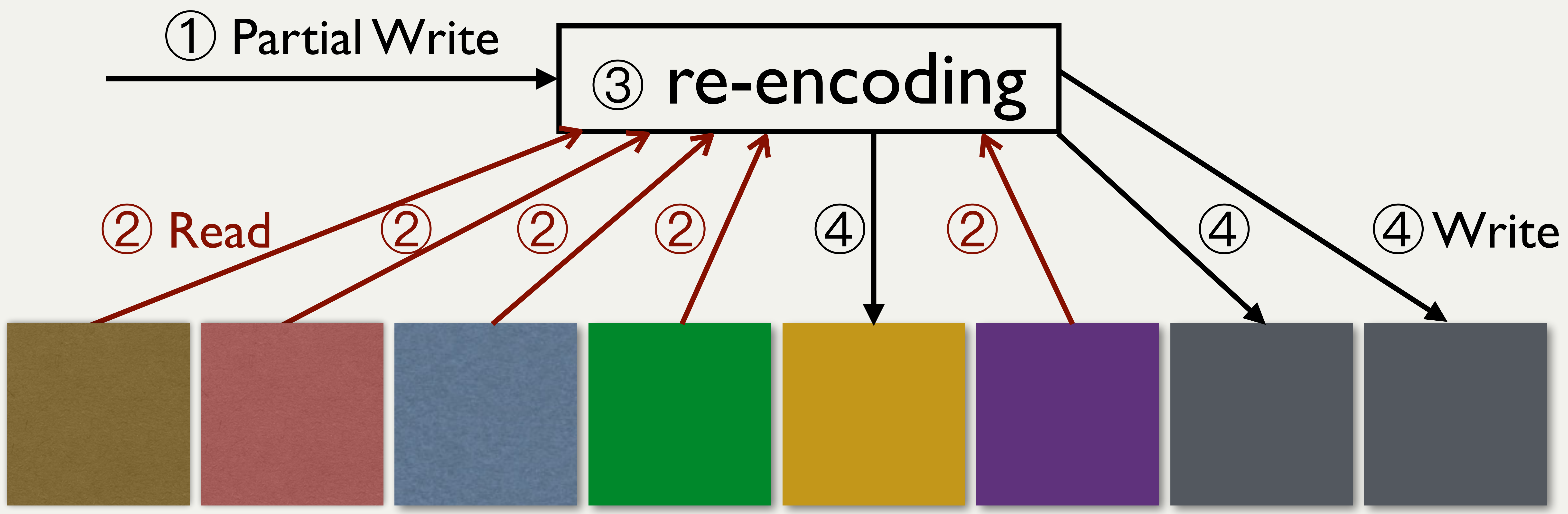
Partial Write in EC



unaligned write

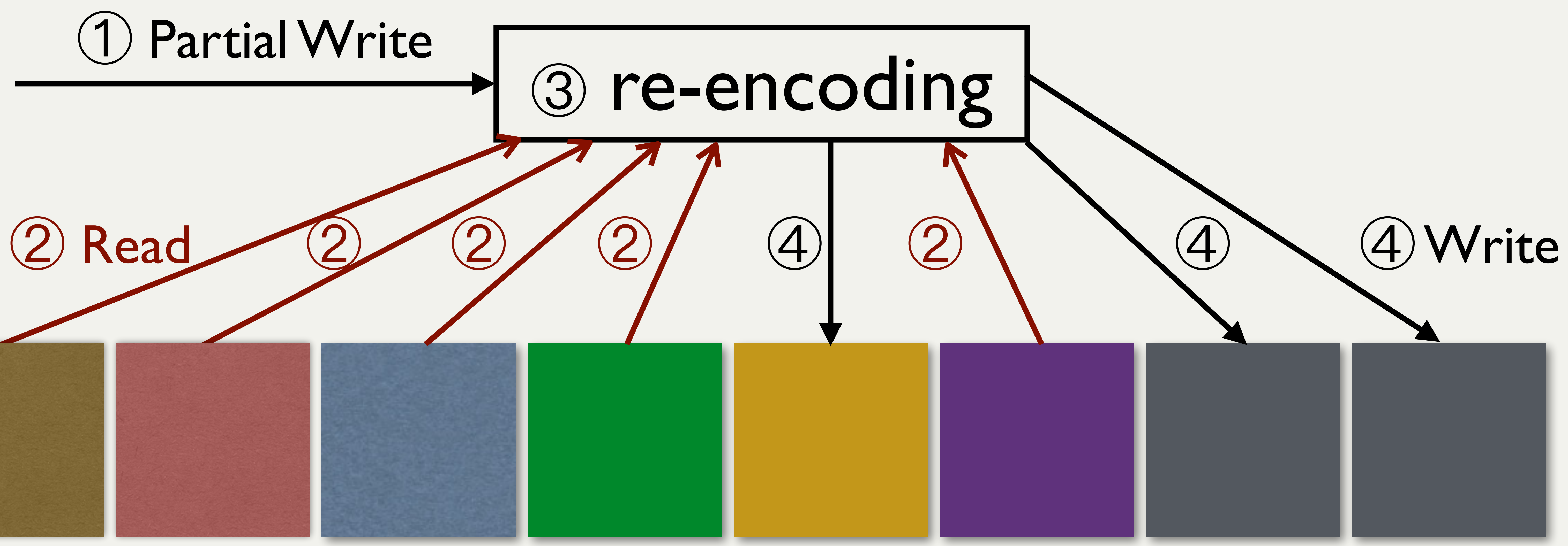
A Simple Approach to Partial Write

- By fully re-encoding



A Simple Approach to Partial Write

- By fully re-encoding

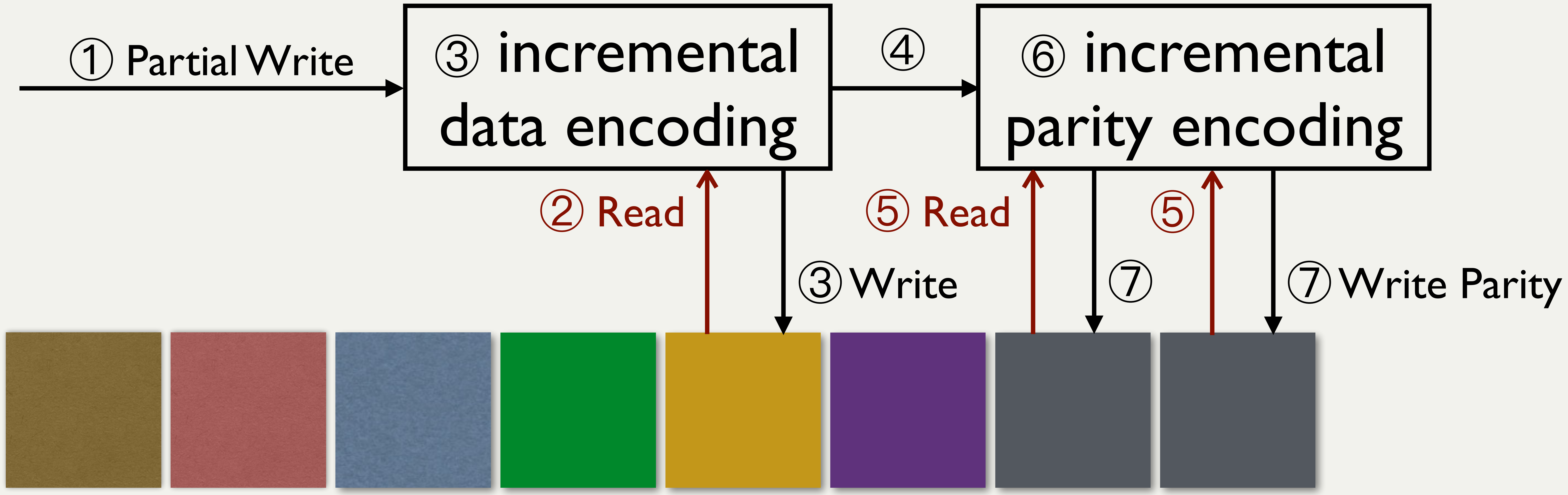


high I/O amplification

unfriendly to parallelism

An Incremental Approach to Partial Write

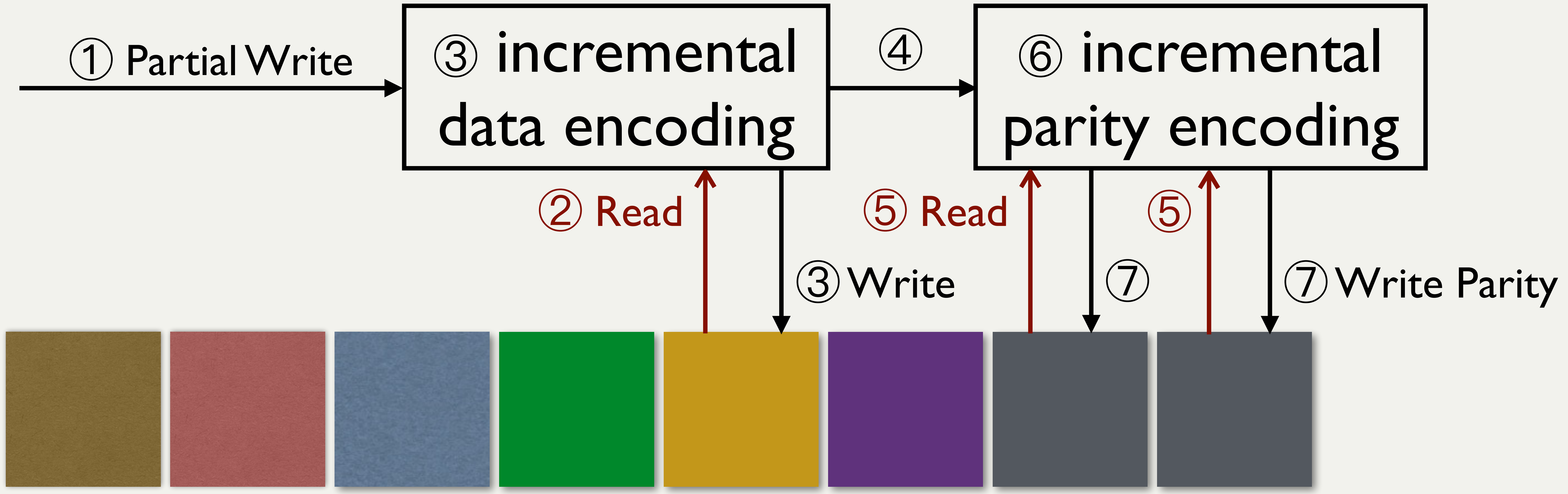
- By incremental encoding



An Incremental Approach to Partial Write

美团云 MEITUAN OPEN SERVICES 技术突破边界 服务传递价值

- By incremental encoding

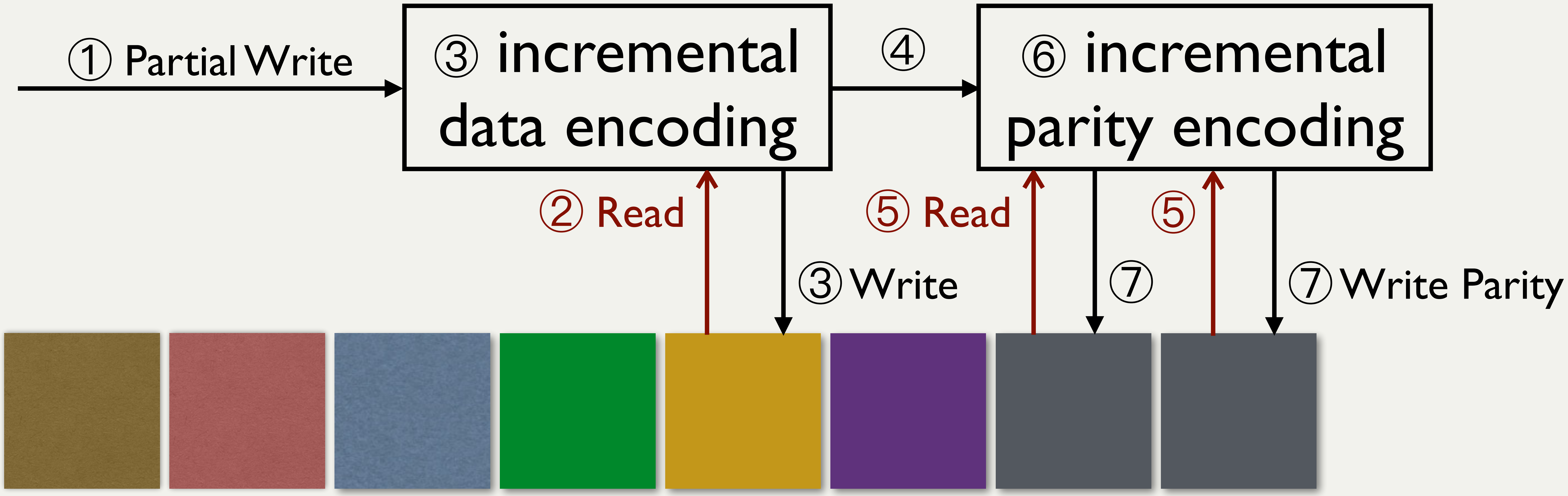


moderate I/O amplification

friendly to inter-disk parallelism

An Incremental Approach to Partial Write

- By incremental encoding



moderate I/O amplification

friendly to inter-disk parallelism

but, in-place read-and-write is expensive!

Cost of in-place read-and-write

美团云
MEITUAN OPEN SERVICES
技术突破边界 服务传递价值

- its latency is equivalent to that of random seek
- performance hurts a lot
  - ✤ *random write: reduced by half*
  - ✤ *sequential write: reduced to that of random write*
- the major obstacle for EC
  - ✤ *to get used in read-write hot storage*

7,200 RPM

Cost of in-place read-and-write

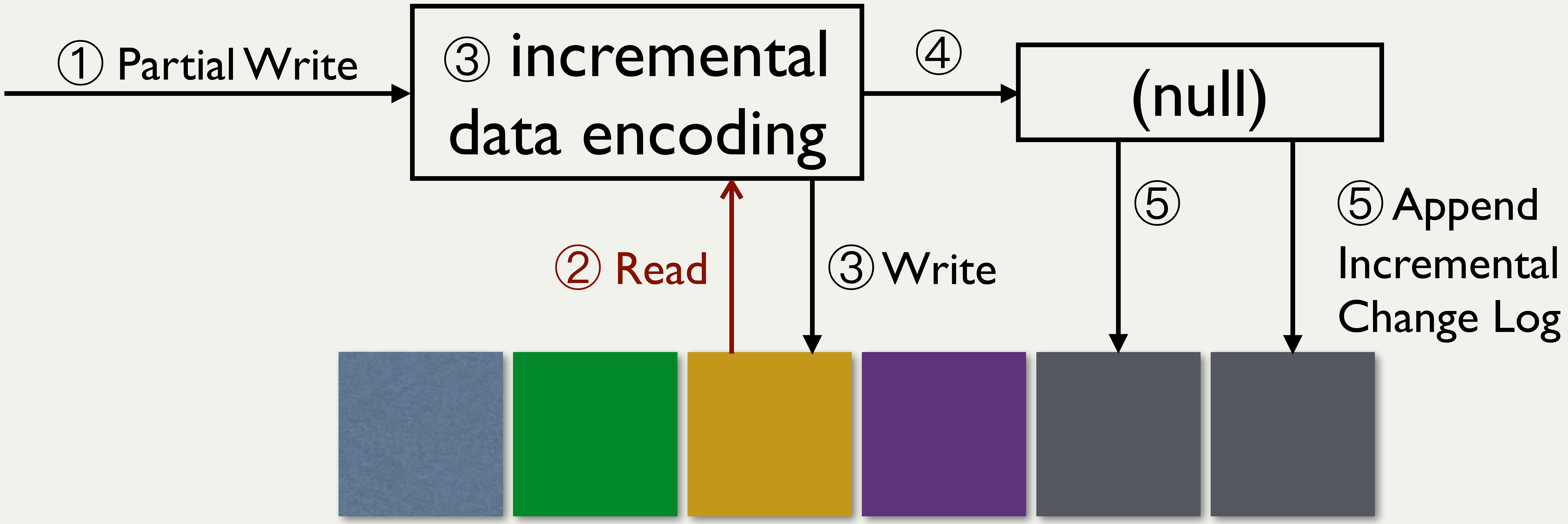
- its latency is equivalent to that of random seek
- performance hurts a lot
  - ✤ *random write: reduced by half*
  - ✤ *sequential write: reduced to that of random write*
- the major obstacle for EC
  - ✤ *to get used in read-write hot storage*

7,200 RPM

Parity Logging

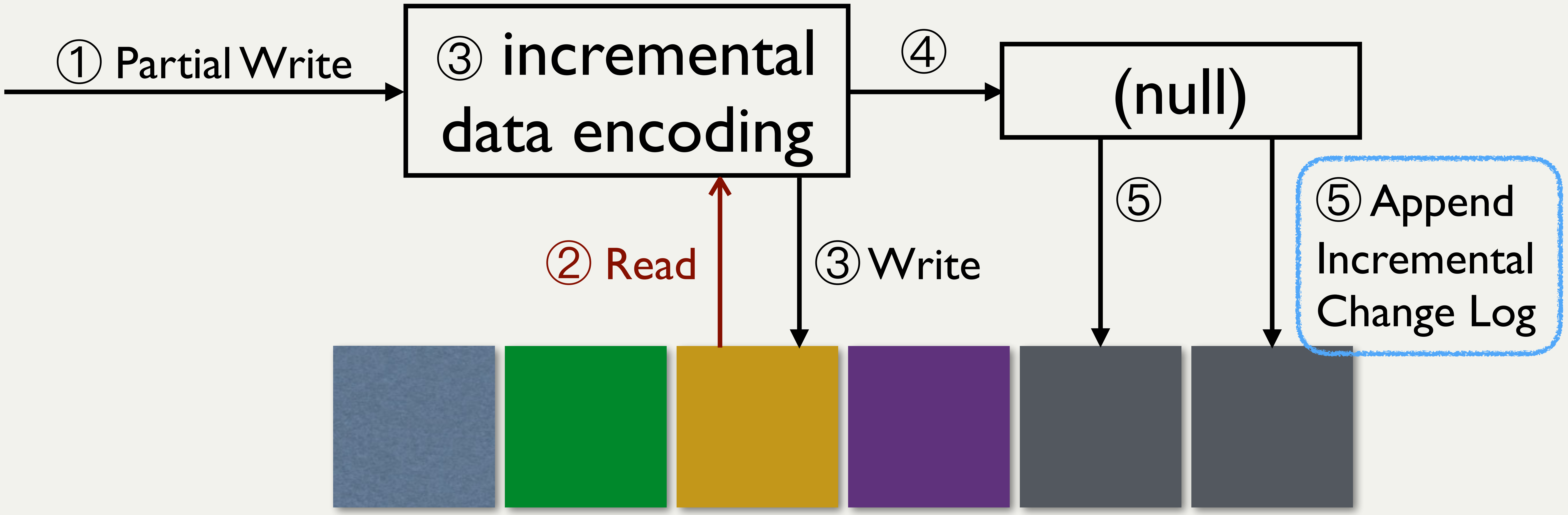
美团云
MEITUAN OPEN SERVICES
技术突破边界 服务传递价值

- An approach to accelerate parity writing

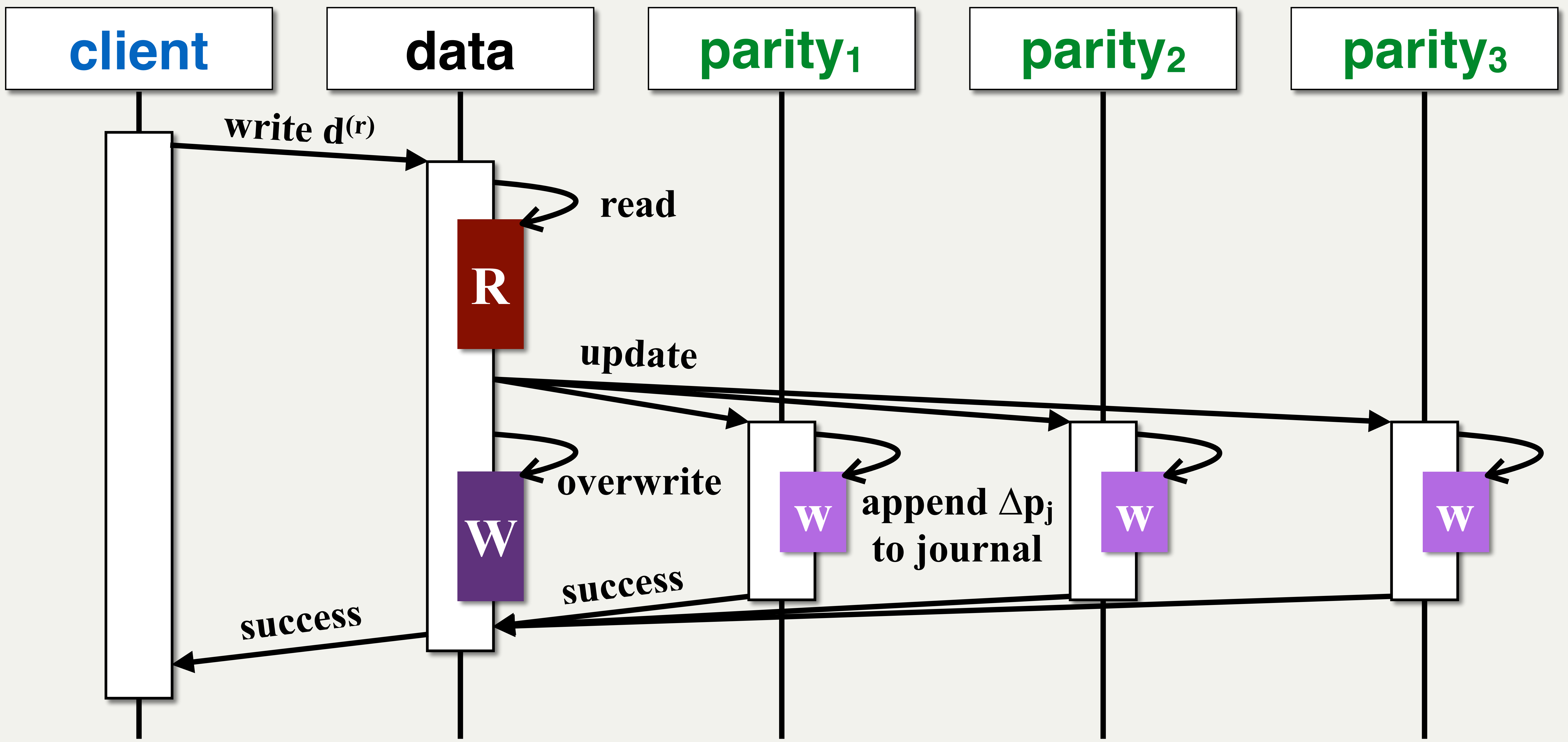


Parity Logging

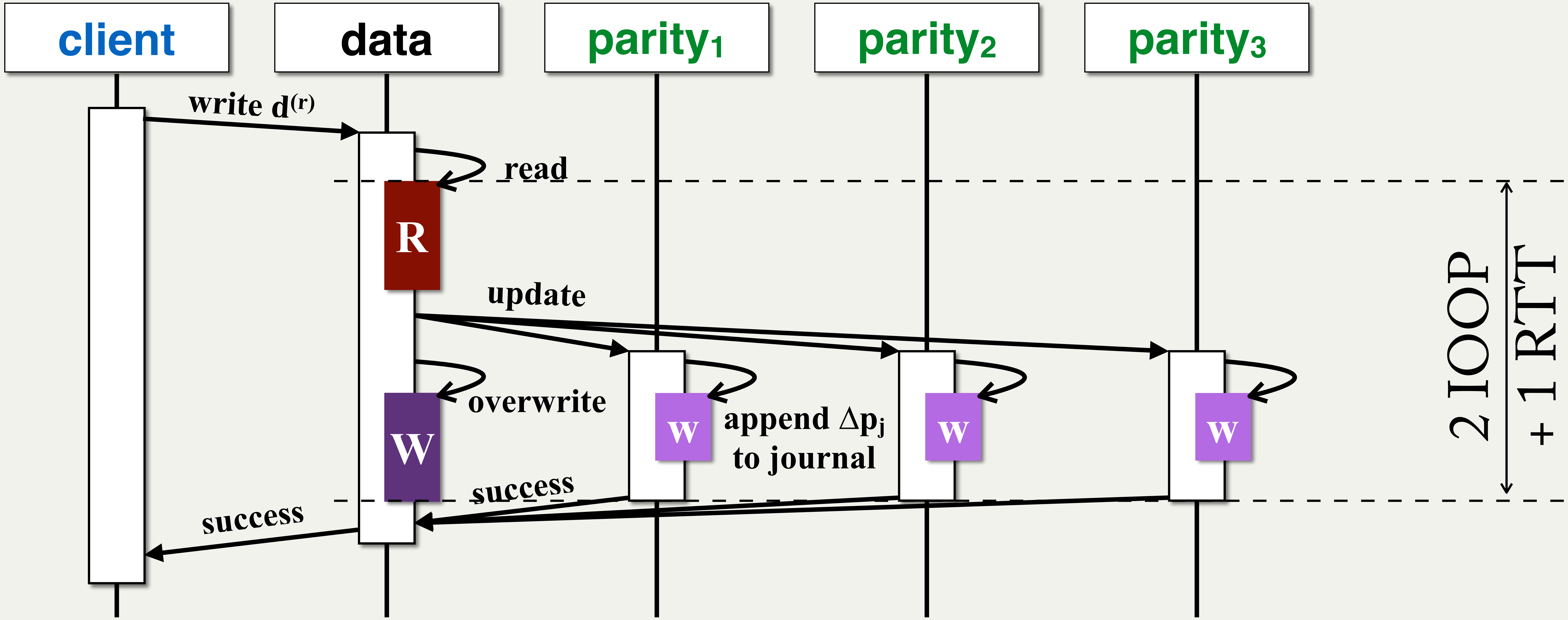
- An approach to accelerate parity writing



Parity Logging



Parity Logging



There's still one in-place read-and-write.

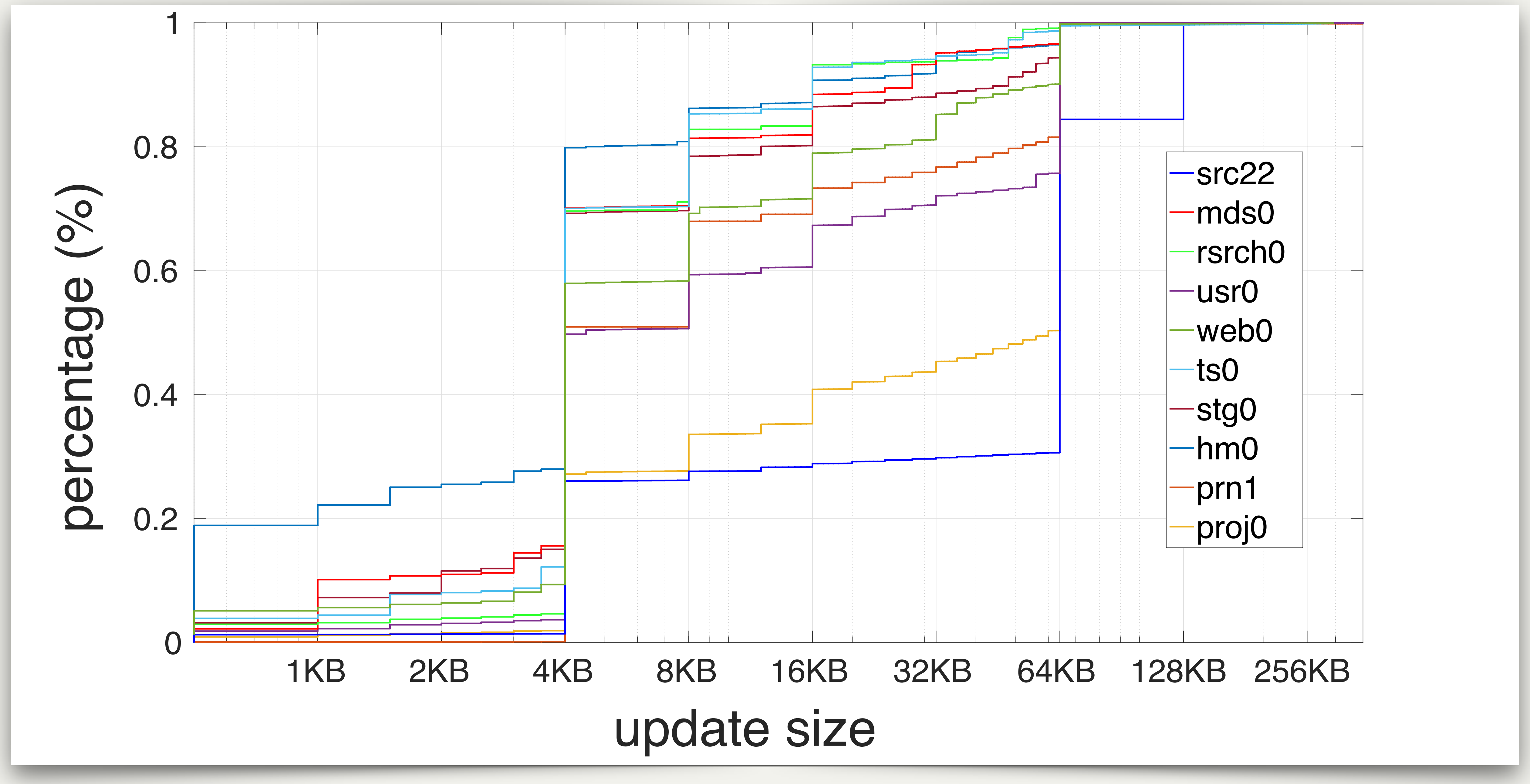
It's the source of overhead.

Pervasiveness of Partial Write

美团云
MEITUAN OPEN SERVICES
技术突破边界 服务传递价值

- partial write is caused by unalignment
- unalignment is the norm
  - *the upper layers of I/O stack just don't know the fact of EC and its alignment*
- It is the major obstacle
  - *for EC to get used in read-write hot storage systems*
- Must be addressed!

An analysis of the MSR I/O traces



PariX: Eliminating the Read

- Consider a series of r writes to a same data block d_i:

$$d_i^{(0)} \leftarrow d_i^{(1)}, d_i^{(2)}, \cdots, d_i^{(r)}$$

- The parities are p_j $(j=1,2,\ldots,k)$

- By incremental parity update equation, we have:

in Galois Field GF(2^8)

$$\Delta p_j^{(1)} = a_{ij} \times \Delta d_i^{(1)}, \ \Delta p_j^{(2)} = a_{ij} \times \Delta d_i^{(2)}$$

$$p_j^{(r)} = p_j^{(0)} + \sum_{x=1}^{r} \Delta p_j^{(x)} = p_j^{(0)} + a_{ij}(d_i^{(1)} - d_i^{(0)} + d_i^{(2)} - d_i^{(1)} + d_i^{(3)} - d_i^{(2)} + \ldots)$$

$$= p_j^{(0)} + a_{ij} \times (d_i^{(r)} - d_i^{(0)})$$

ParIX: Eliminating the Read

美团云 MEITUAN OPEN SERVICES 技术突破边界 服务传递价值

- Consider a series of r writes to a same data block d_i :

$$d_i^{(0)} \leftarrow d_i^{(1)}, d_i^{(2)}, \cdots, d_i^{(r)}$$

- The parities are p_j (j=1,2,…,k)

- By incremental parity update equation, we have:

in Galois Field GF(2^8)

$$\Delta p_j^{(1)} = a_{ij} \times \Delta d_i^{(1)}, \ \Delta p_j^{(2)} = a_{ij} \times \Delta d_i^{(2)}$$

$$p_j^{(r)} = p_j^{(0)} + \sum_{x=1}^{r} \Delta p_j^{(x)} = p_j^{(0)} + a_{ij}(d_i^{(1)} - d_i^{(0)} + d_i^{(2)} - d_i^{(1)} + d_i^{(3)} - d_i^{(2)} + \ldots)$$

$$= p_j^{(0)} + a_{ij} \times (d_i^{(r)} - d_i^{(0)})$$

intermediate values are ALL reduced, no need to read it at all!

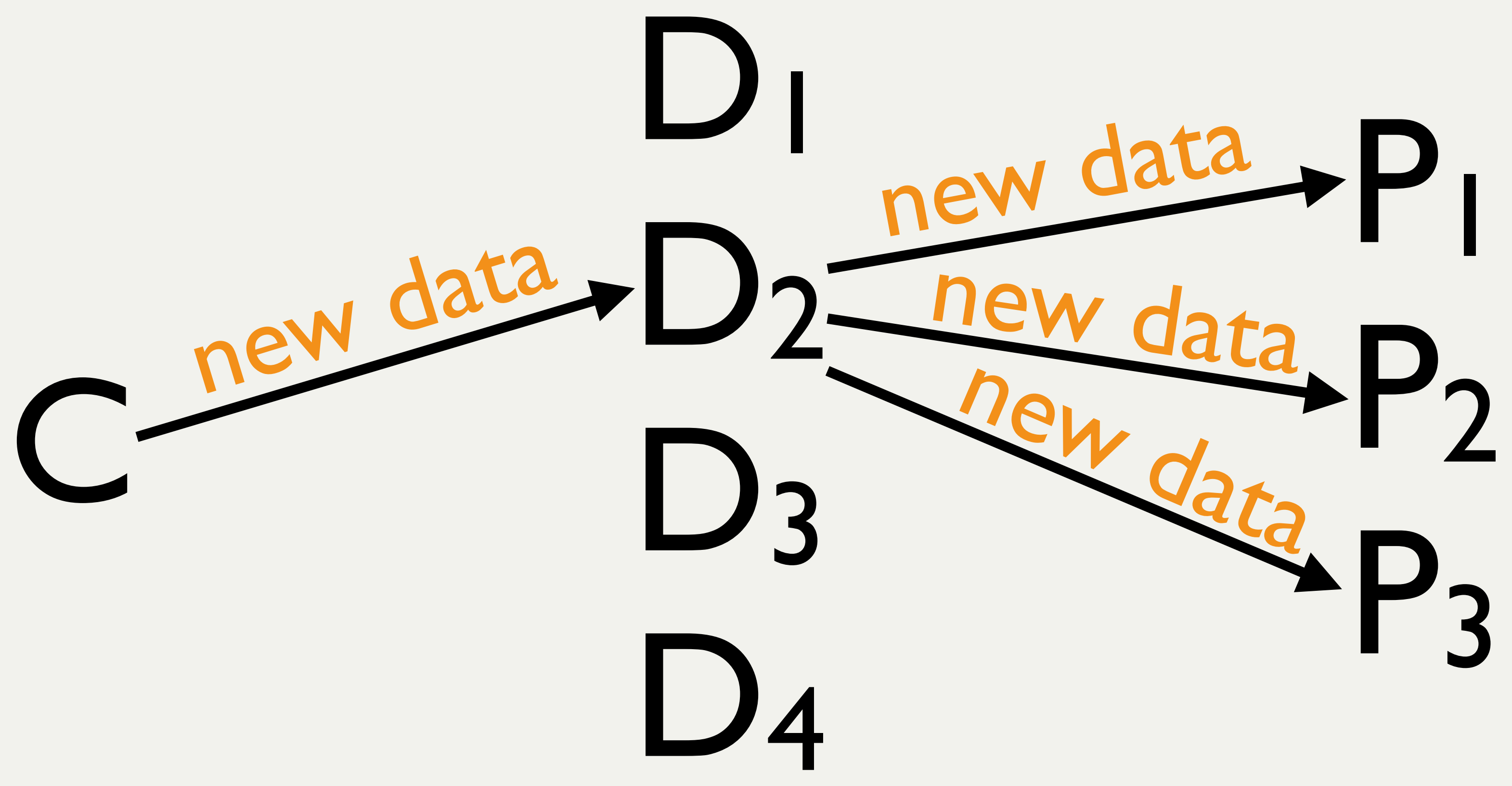
PariX: Logging Data on Parities

- Instead of parities deltas, as in parity logging
- Each parity records m series of change logs
  - *respectively for m data blocks*
  - *stored as a single journal file*
  - *interleaved with each other*
  - *every $d^{(0)}$ always comes after corresponding $d^{(1)}$*
- Example:

$$d_2^{(1)}, d_1^{(1)}, d_2^{(0)}, d_4^{(1)}, d_1^{(0)}, d_1^{(2)}, d_2^{(2)}, ..., d_i^{(k)} (i = 1..m, k = 0..r_i), ...$$

The Speculation

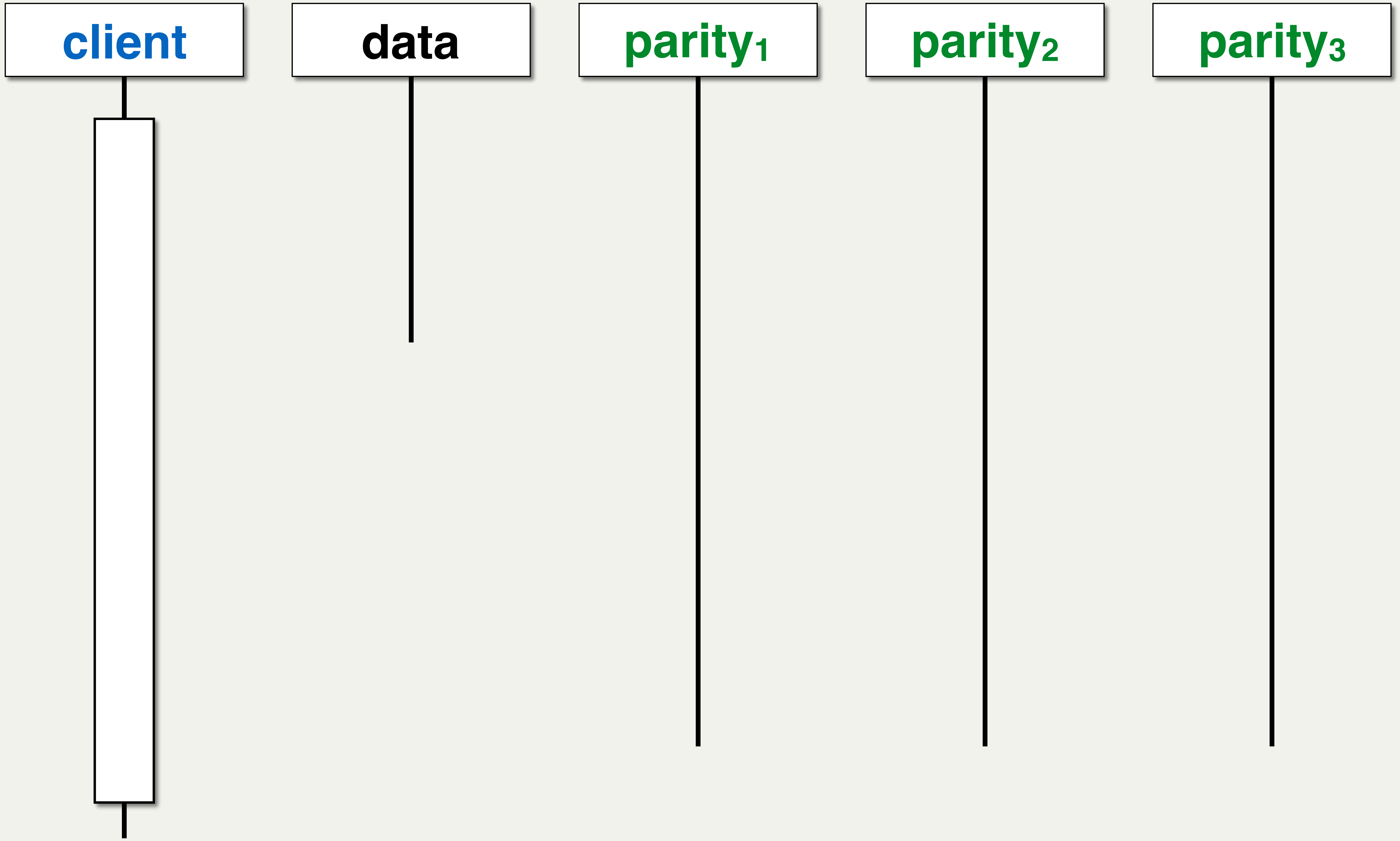
- Whether the parities need $d_i^{(0)}$ or not?



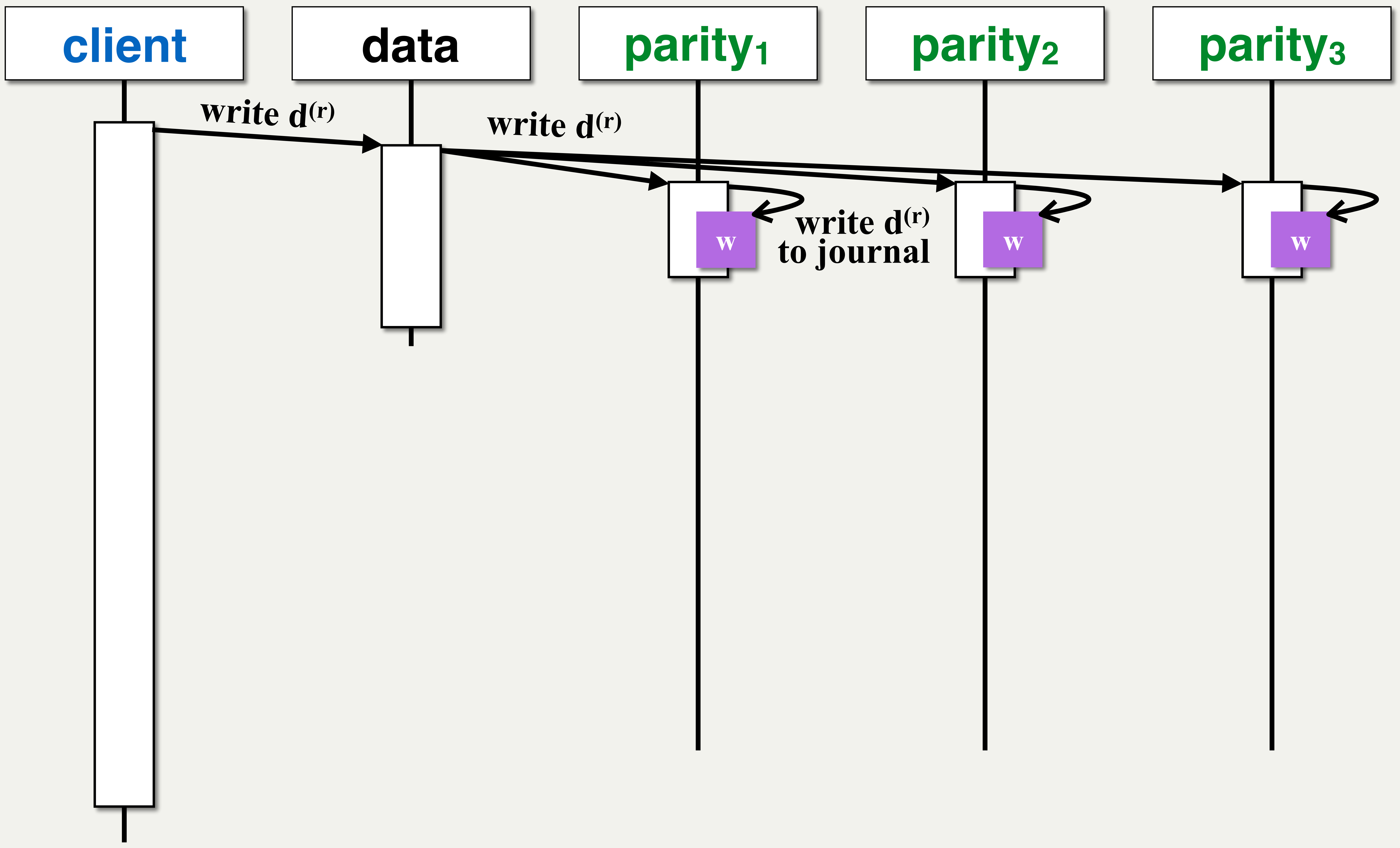
- It is too costly to maintain consensus among nodes about this

- Instead, we speculate about it:
  - *Assume* $d_i^{(0)}$ *is NOT needed (mostly right)*
  - *Send* $d_i^{(0)}$ *only when it is actually needed (sometimes only)*

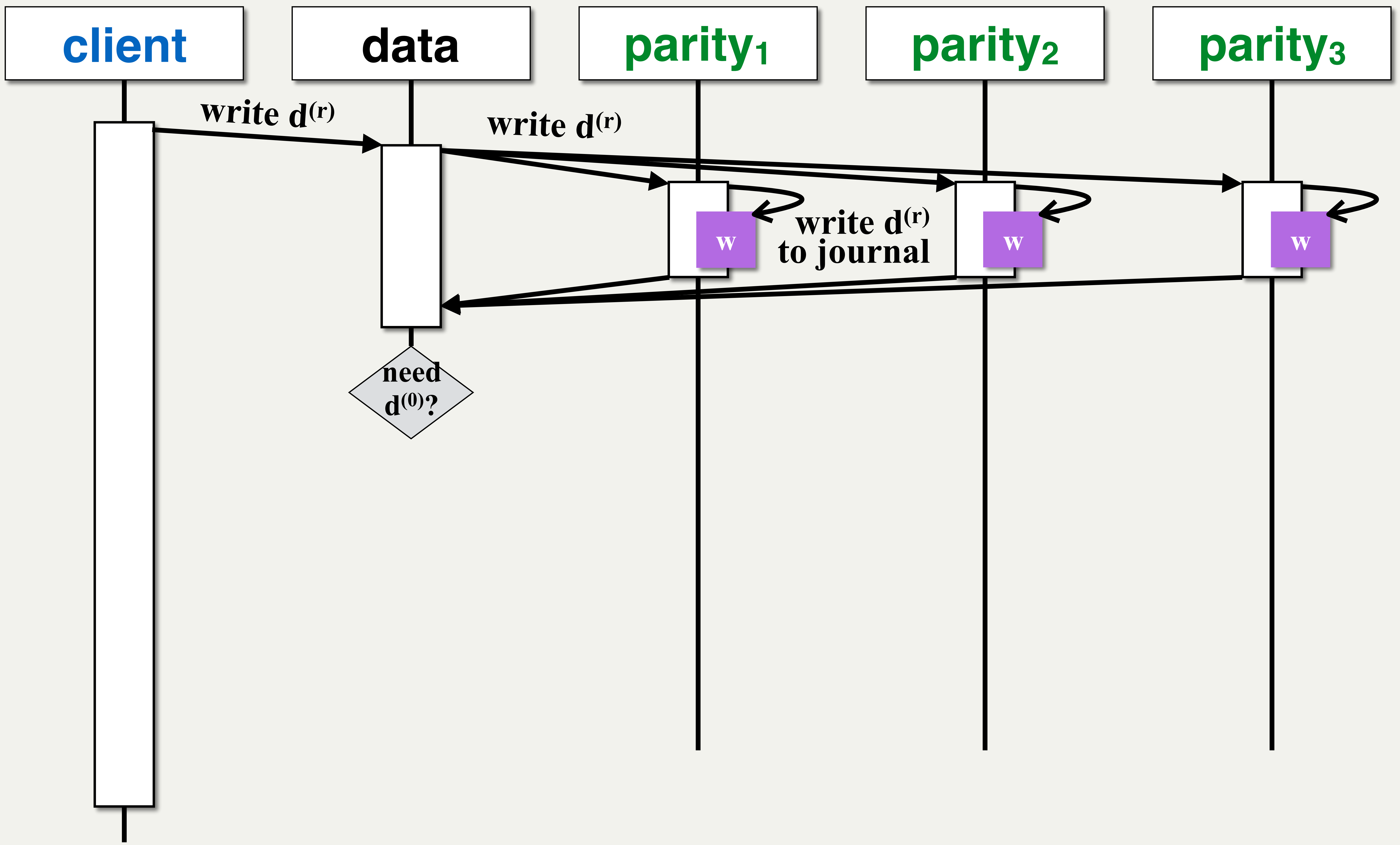
PARIX: Eliminating the Read



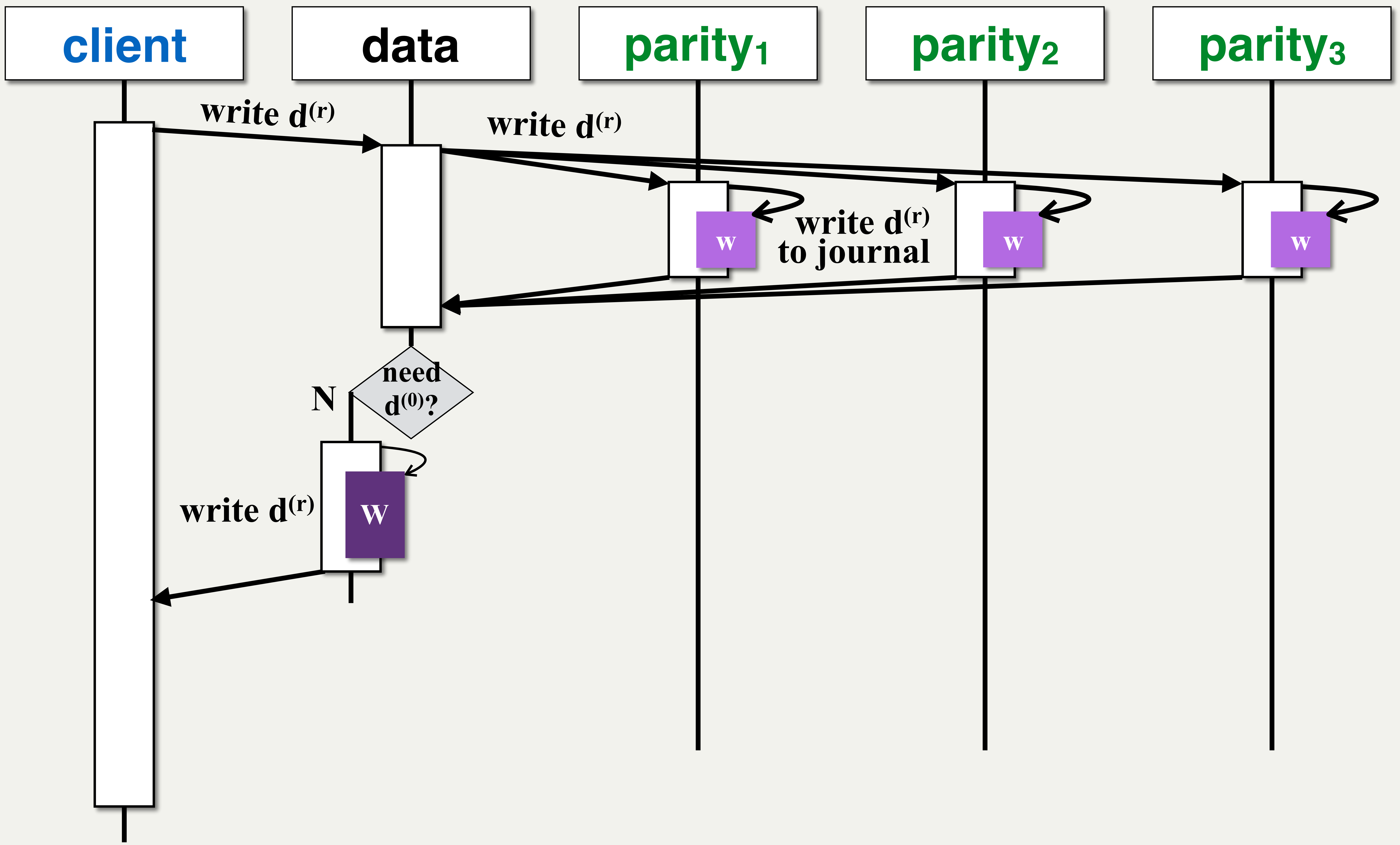
PariX: Eliminating the Read



ParIX: Eliminating the Read

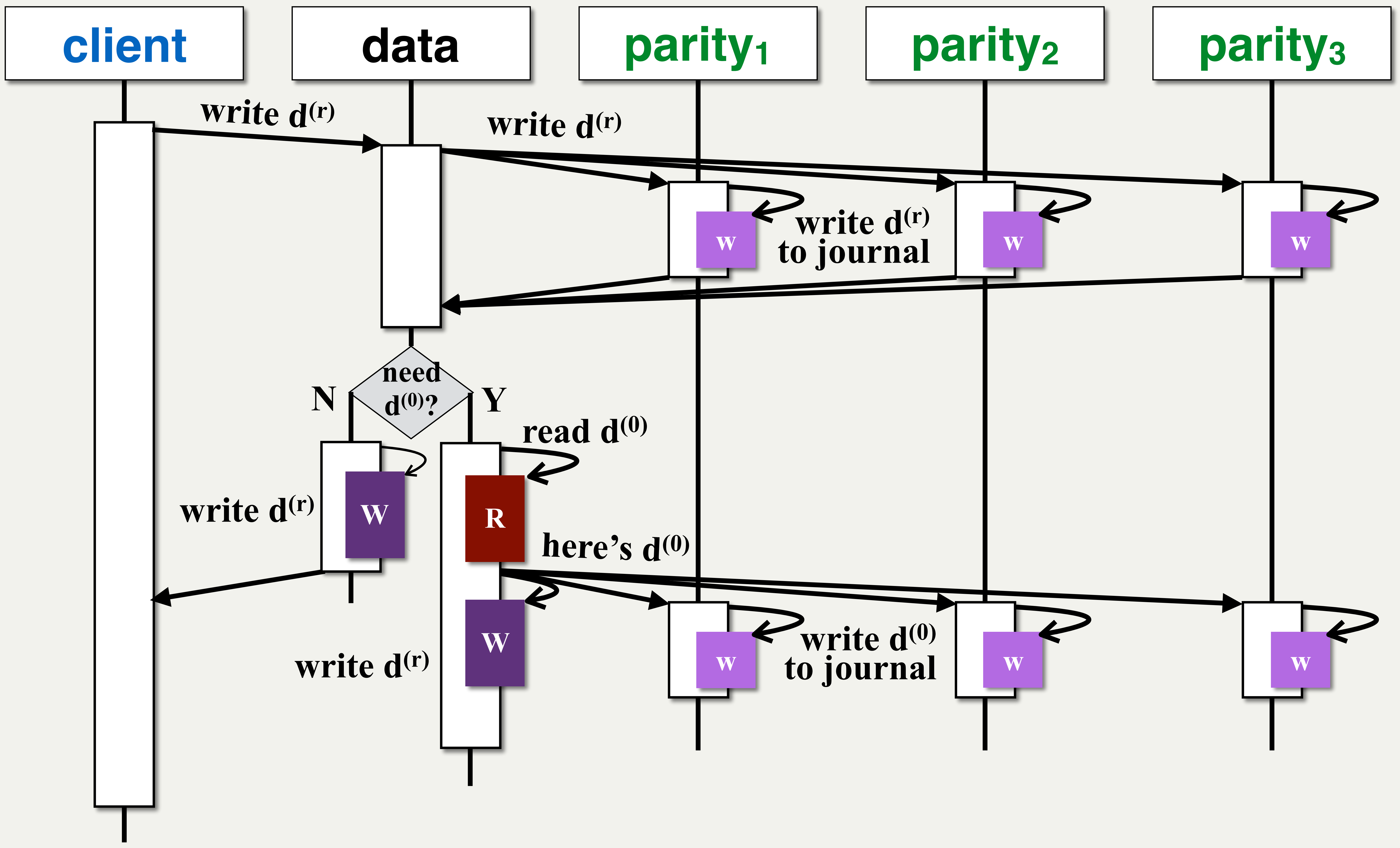


ParíX: Eliminating the Read

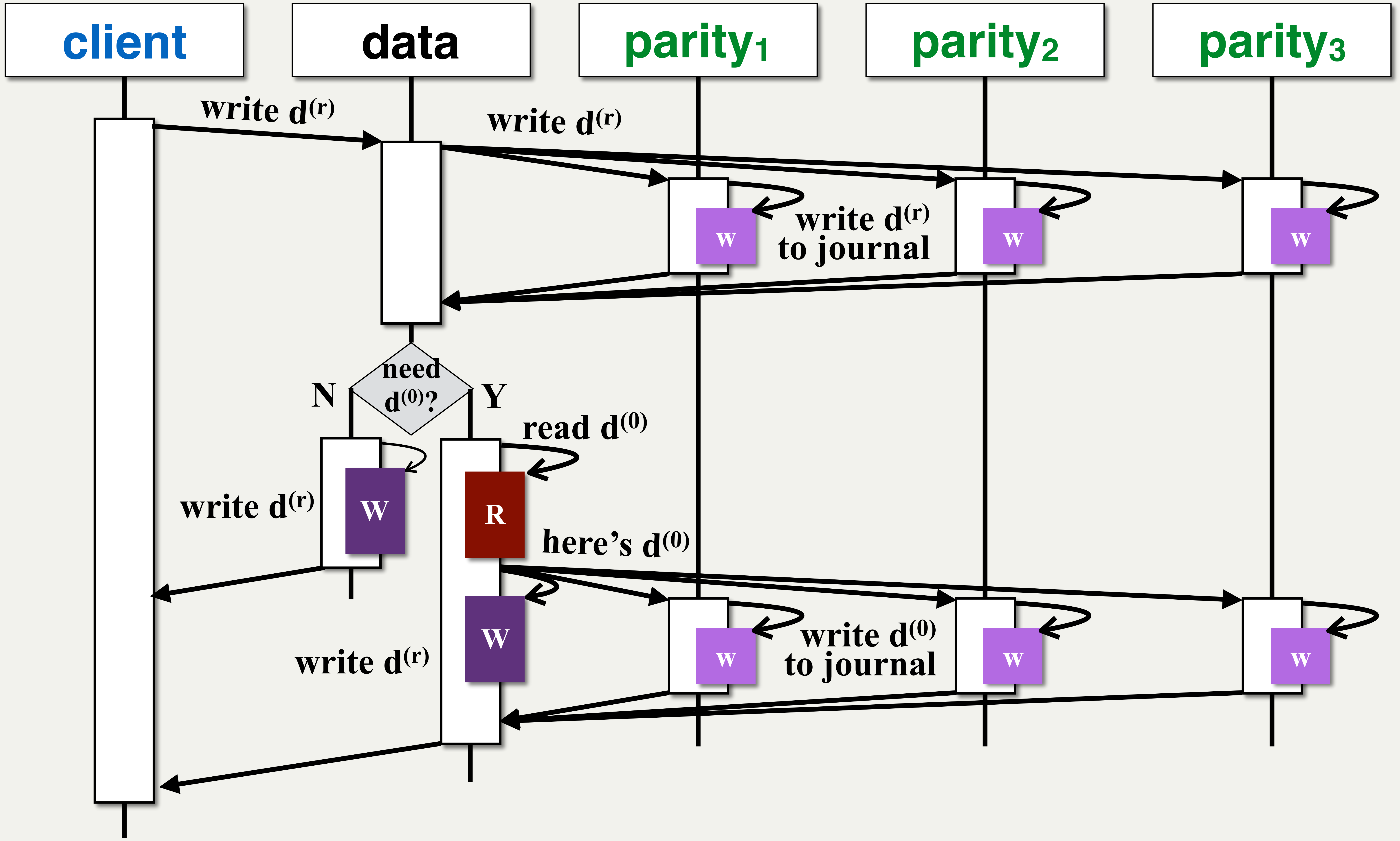


PariX: Eliminating the Read

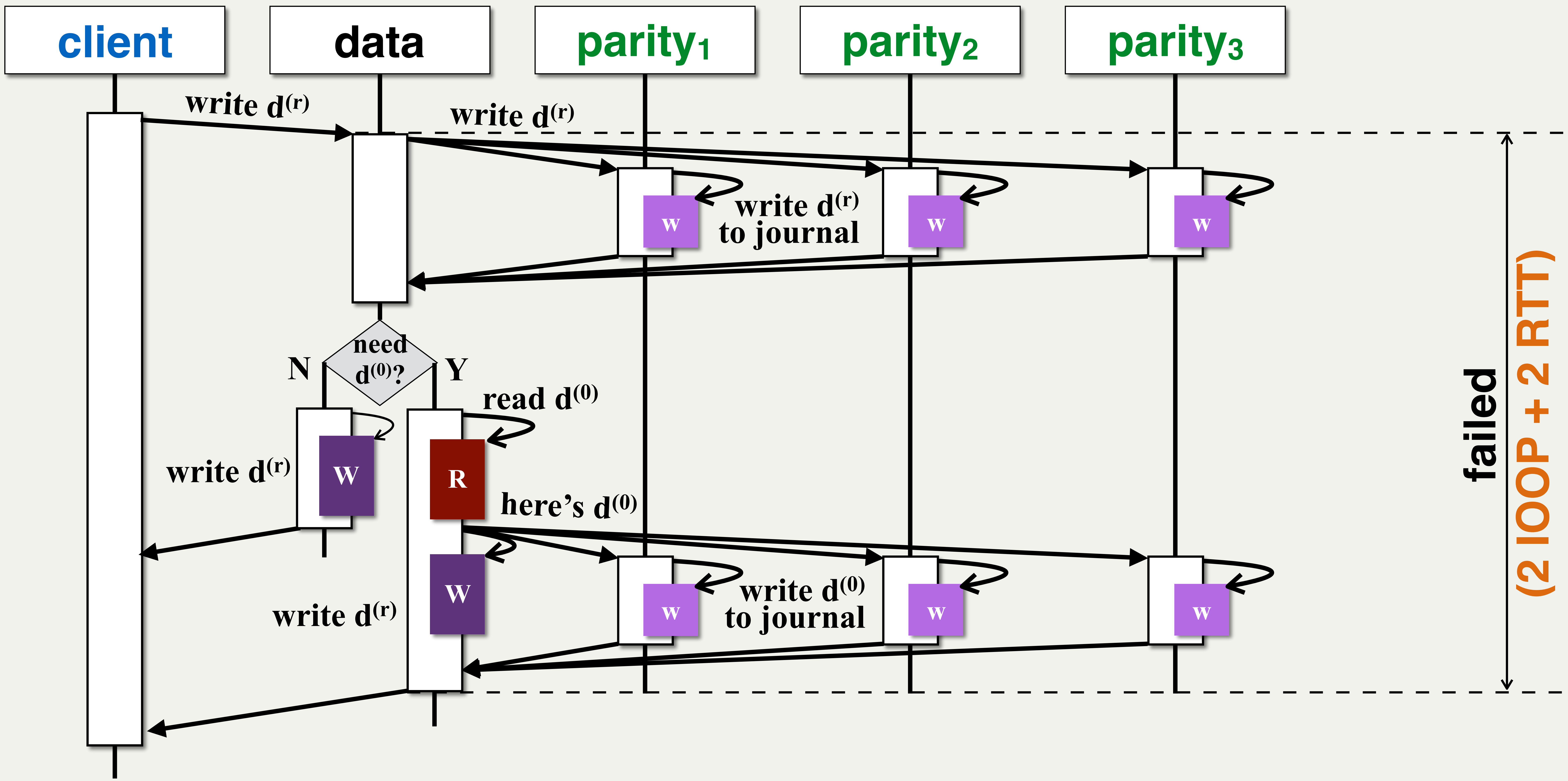
美团云
MEITUAN OPEN SERVICES
技术突破边界 服务传递价值



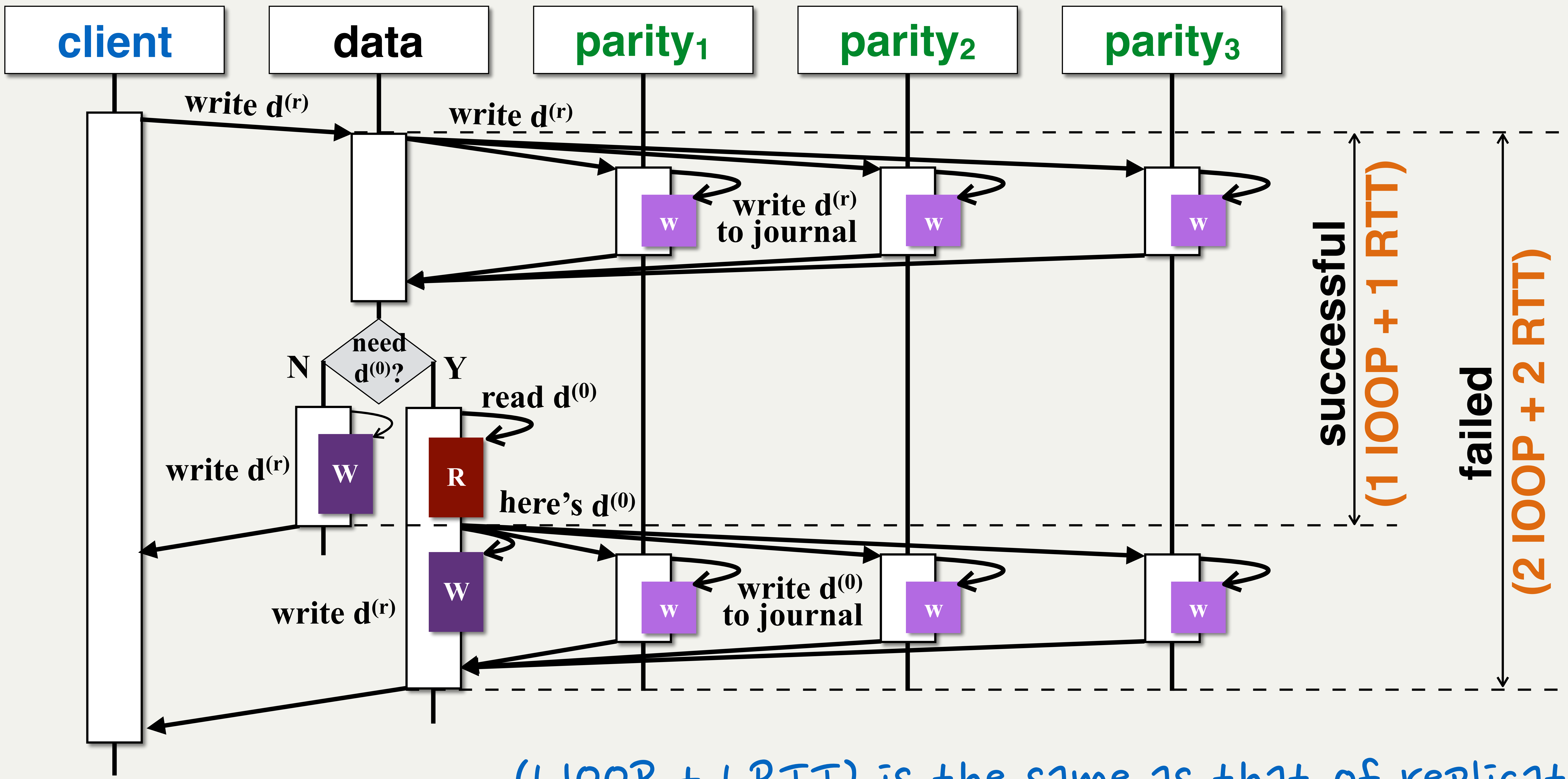
ParIX: Eliminating the Read



PariX: Eliminating the Read



ParIX: Eliminating the Read

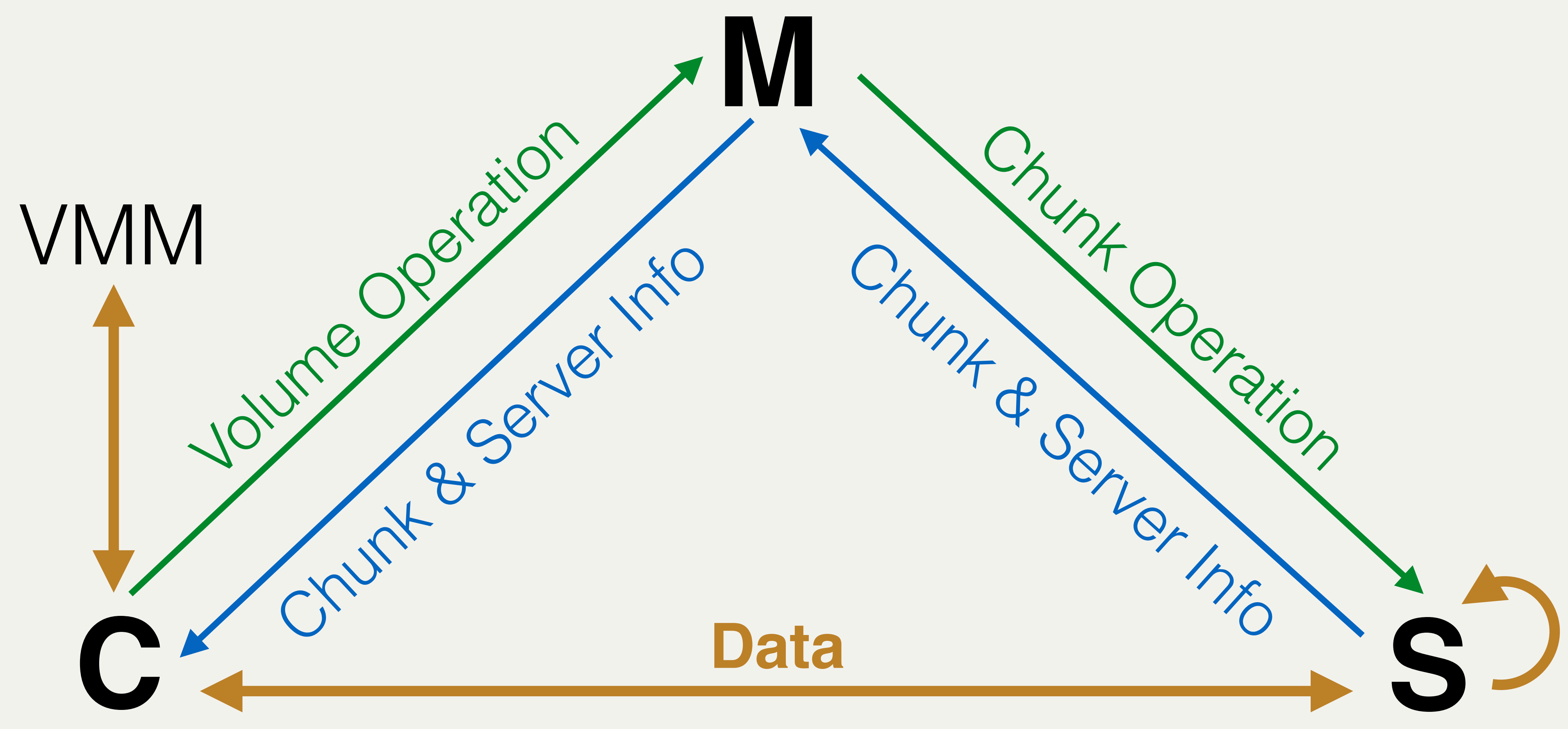


(1 IOOP + 1 RTT) is the same as that of replication !

penalty of failure is only a network RTT!

Implementation

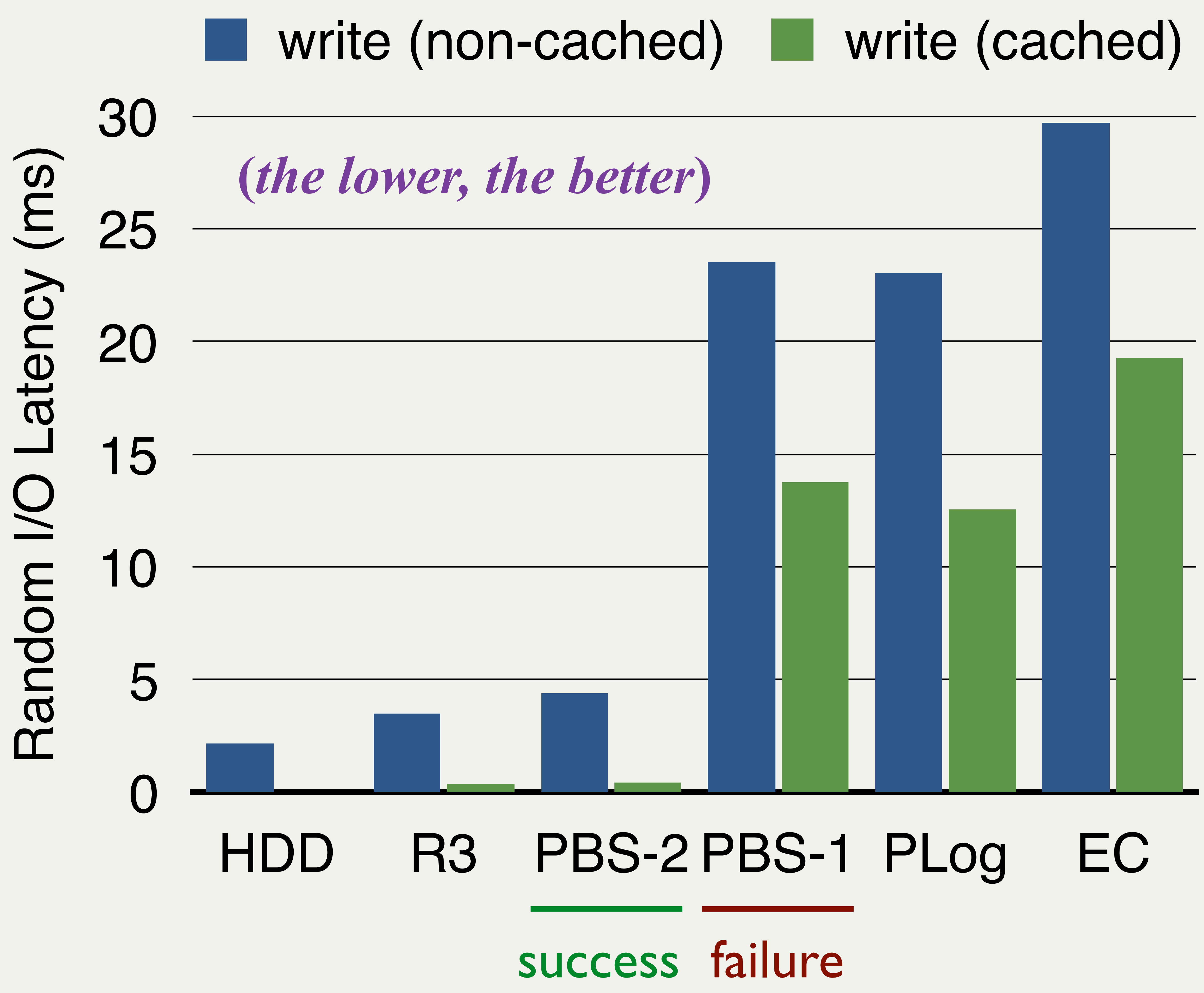
- Based on Ursa, a block store for our public & private cloud, which hosts all our businesses:
  - *food delivery: ~13M orders/day, ~67M monthly active users, ~200M total users*
  - *crowd-sourced reviews (about businesses), coupons, hotel reservation, tourism, plane & train tickets, movie tickets, payment, …*
- Architecture
  - *Master-Server-Client*
  - *No single-point-of-failure*

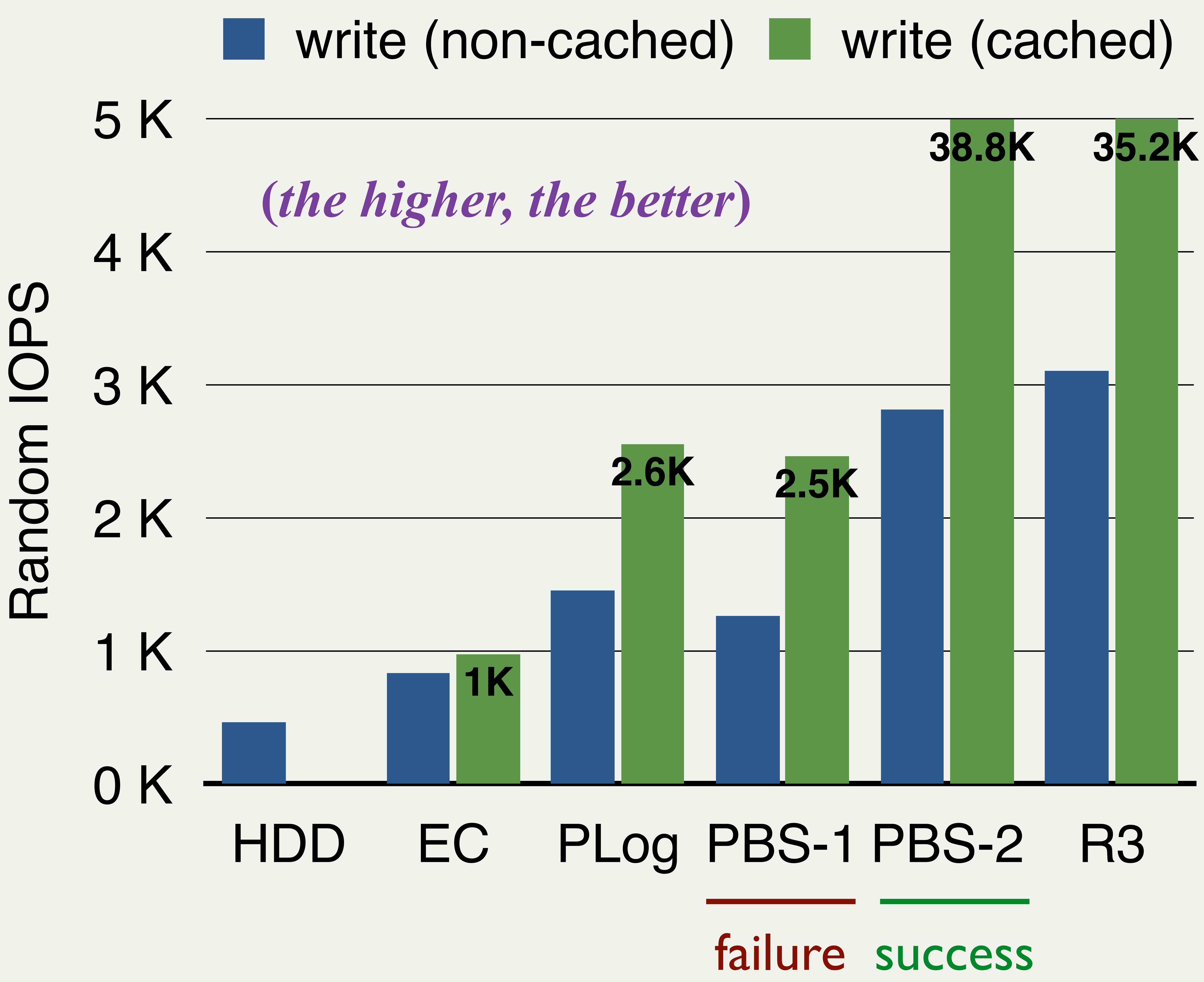


Evaluation

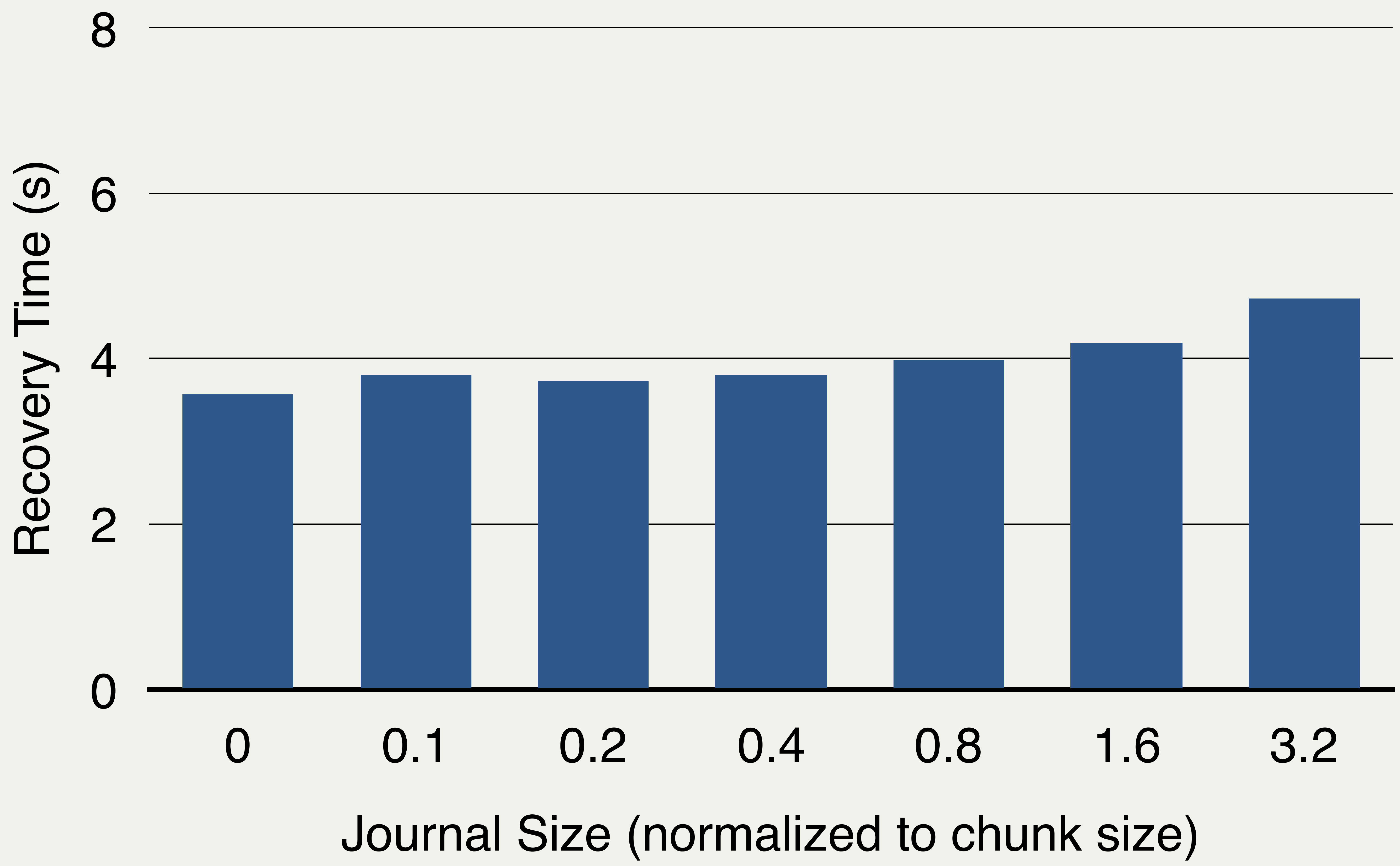
- 10 Servers, each with:
  - *12 HDD, 7200 RPM,*
    - *attached to an LSI 3008 SAS HBA, w/o flash cache*
  - *2-way Intel Xeon CPU*
  - *128GB RAM*
- 10Gb Ethernet
  - *connected with a non-blocking switch*

Evaluation


write (non-cached)
write (cached)
Random I/O Latency (ms)
30
25
20
15
10
5
0
(the lower, the better)
HDD
R3
PBS-2
PBS-1
PLog
EC
success
failure


write (non-cached)
write (cached)
Random IOPS
5 K
4 K
3 K
2 K
1 K
0 K
(the higher, the better)
38.8K
35.2K
2.6K
2.5K
1K
HDD
EC
PLog
PBS-1
PBS-2
R3
failure
success

Evaluation



8
6
4
2
0
Recovery Time (s)
0
0.1
0.2
0.4
0.8
1.6
3.2
Journal Size (normalized to chunk size)

Summary

- Performance is the major obstacle
  - *for EC to get used in read-write hot storage*
  - *especially in the case of partial write*
- PariX: a novel approach to eliminate overhead in common cases
  - *with a very small penalty in corner cases*
- Evaluations show that its performance meets expectations
  - *much better than existing approaches*
  - *close to that of 3-replica scheme*

Q&A

美团云
MEITUAN OPEN SERVICES
技术突破边界 服务传递价值

2017 USENIX Annual Technical Conference

https://www.mtyun.com

Thanks!

2017 USENIX Annual Technical Conference

美团云
MEITUAN OPEN SERVICES