INSTalytics: Cluster Filesystem Co-design for Big-data Analytics

Muthian Sivathanu, **Midhul Vuppalapati**, Bhargav S. Gulavani,
Kaushik Rajan, Jyoti Leeka, Jayashree Mohan, Piyus Kedia

Microsoft Research India

Big-data Analytics: Motivation

- Queries to measure, understand & derive intelligence from data

- Huge business value (billion $ industry)
  - Large internet companies -> massive data
    - Store & process **Exabytes of data** per week
  - Analytics as a Service offerings

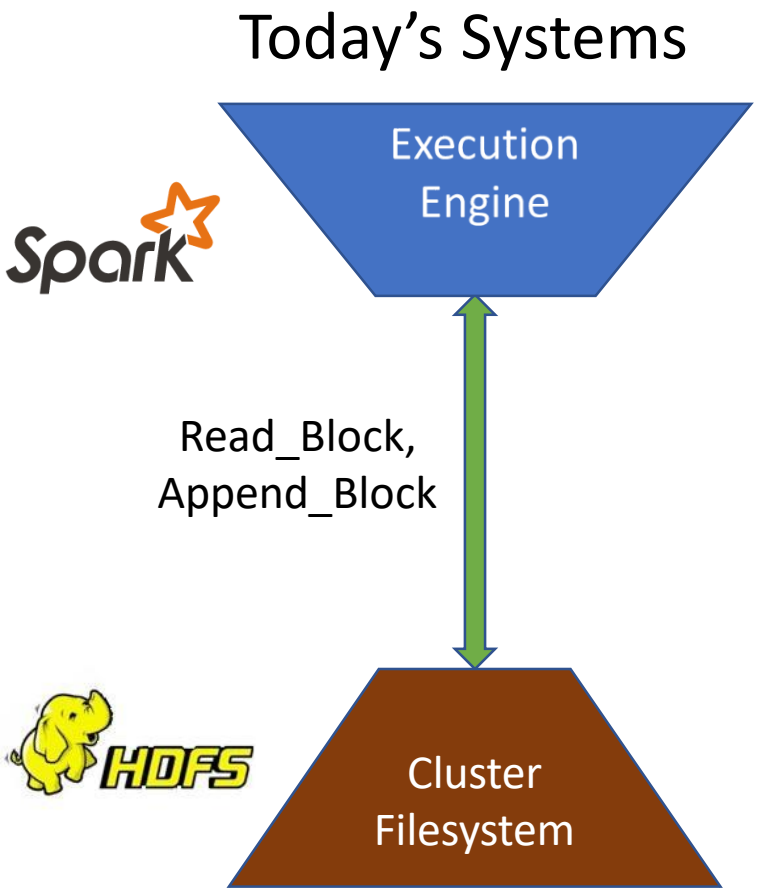
- Several Frameworks
  - Extensive research work over past decade

Problem statement

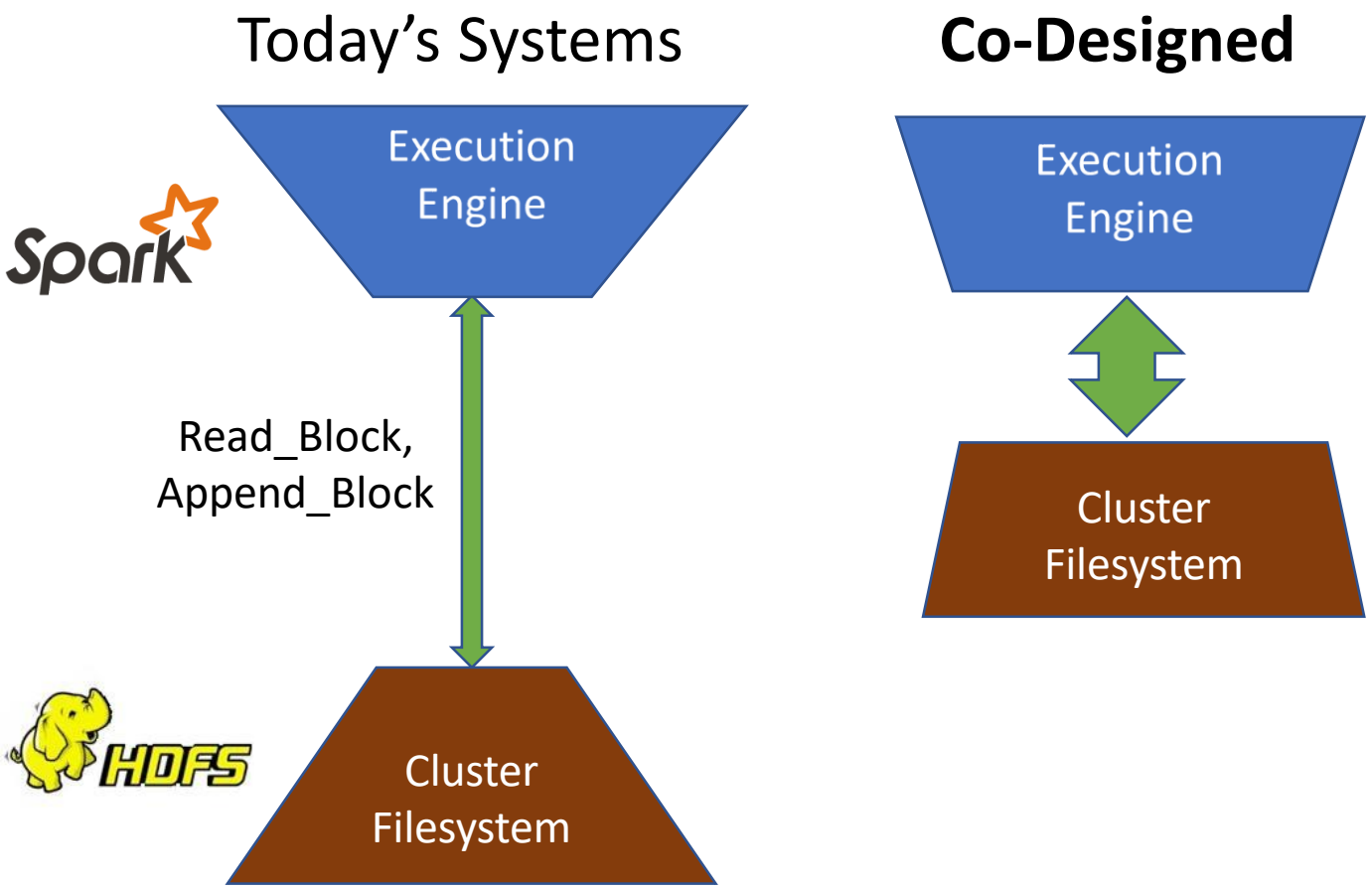
- Large-scale analytics queries (100TBs - PBs)
  - Very expensive to store in DRAM / on SSD
  - Take several hours to execute (on 1000s of machines)
  - Consume significant CPU, Disk, Network resources
- **Two problems**
  - High latency for users
  - Huge resource/machine cost for service provider

- **Goal: Improve efficiency of large scale analytics processing**

Approach at a glance

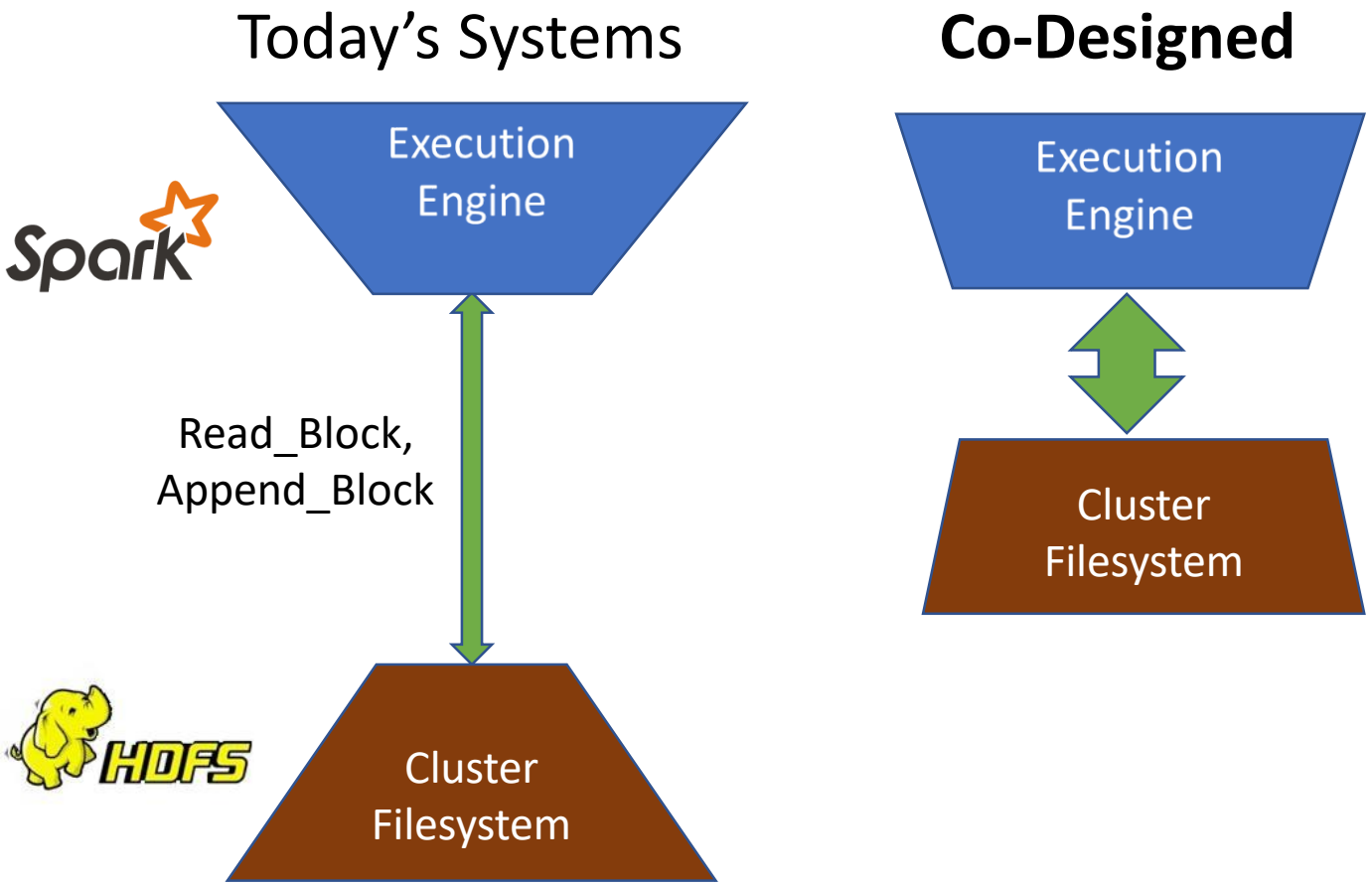


Approach at a glance



Compute-aware Storage can drive significant efficiency in analytics

Approach at a glance



INSTalytics

(Intelligent Store-powered Analytics)

Improves Query Performance

Latency +
Execution cost

No strings attached!

Compute-aware Storage can drive significant efficiency in analytics

Outline

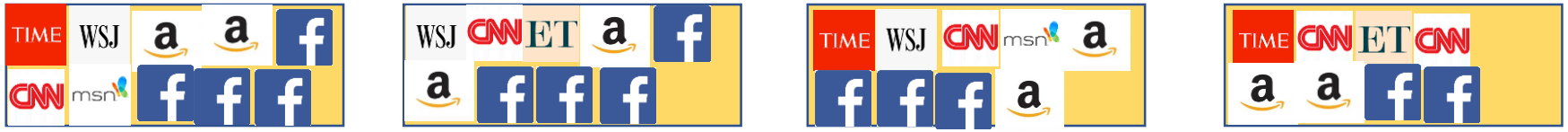
- *Introduction*
- Design & Evaluation
  1.) Key mechanism at storage layer
  2.) Efficient Query Execution
- Implementation
- Summary

Common Techniques used today

- **Partitioning**

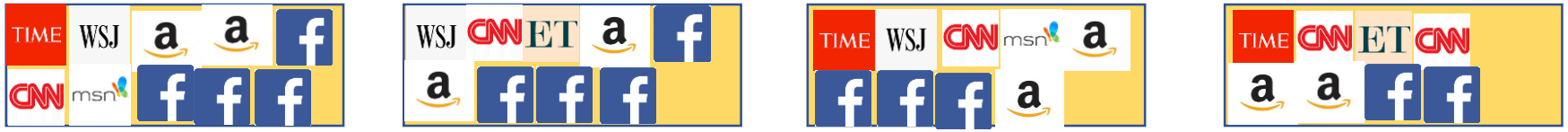
Common Techniques used today

- **Partitioning**



Common Techniques used today

- **Partitioning**

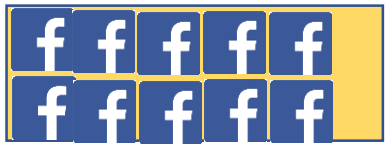


Retrieve all click records with domain == "cnn"

(Filter Query)

Common Techniques used today

- **Partitioning**

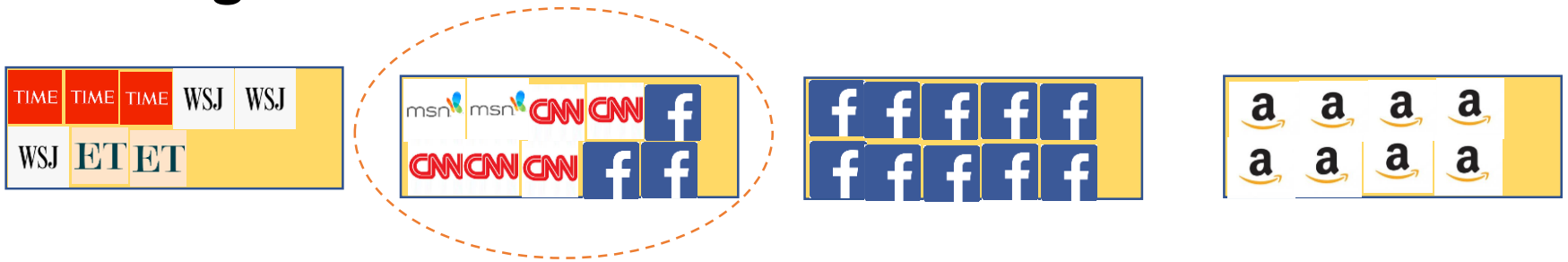


Retrieve all click records with domain == "cnn"

(Filter Query)

Common Techniques used today

- **Partitioning**



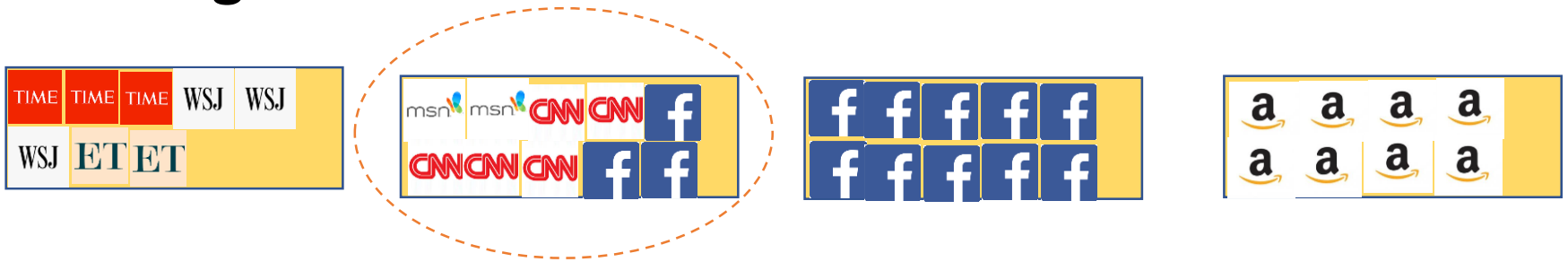
Retrieve all click records with domain == "cnn"

(Filter Query)

- **Partitioning + Co-location**

Common Techniques used today

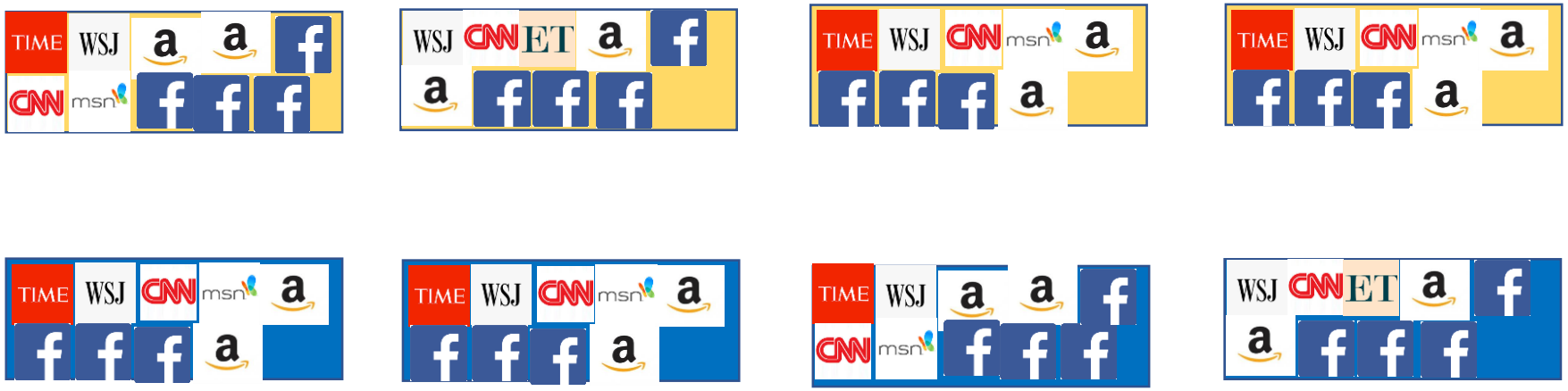
- **Partitioning**



Retrieve all click records with domain == "cnn"

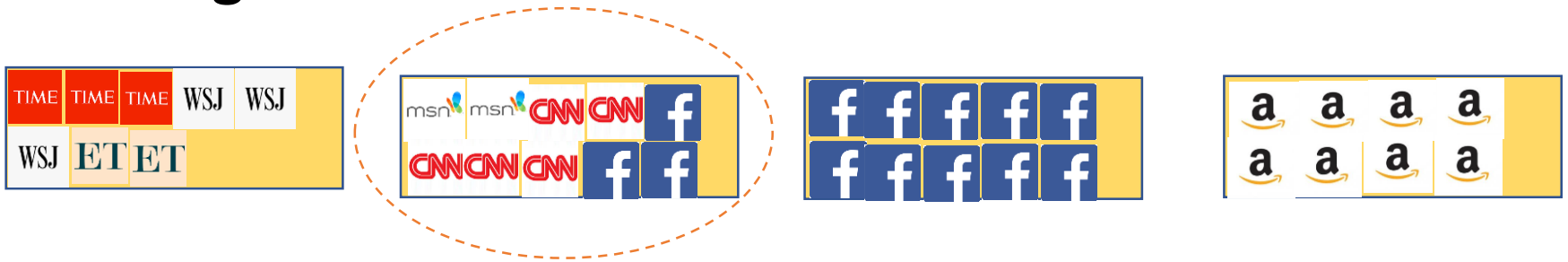
(Filter Query)

- **Partitioning + Co-location**



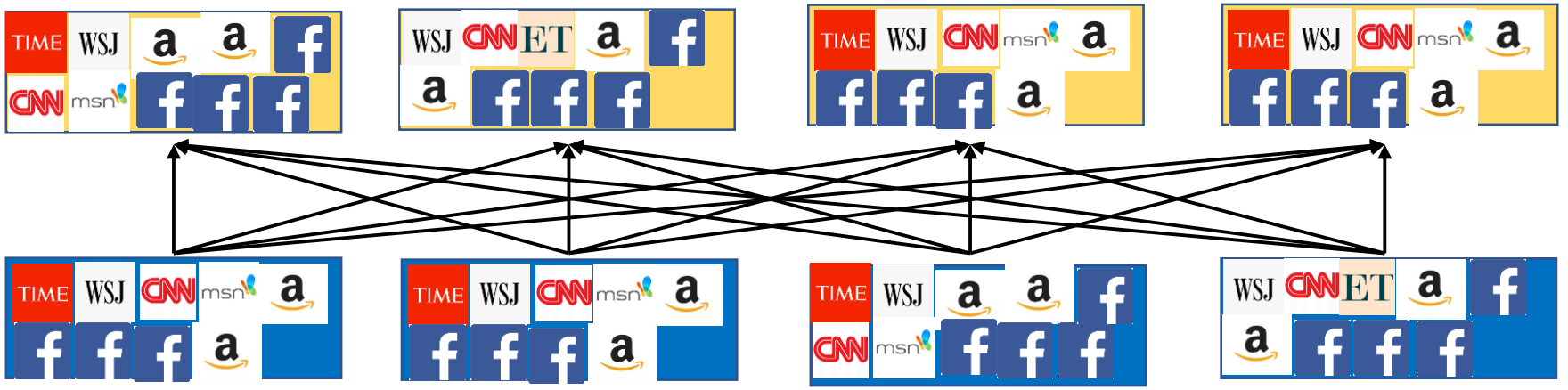
Common Techniques used today

- **Partitioning**



(Filter Query)

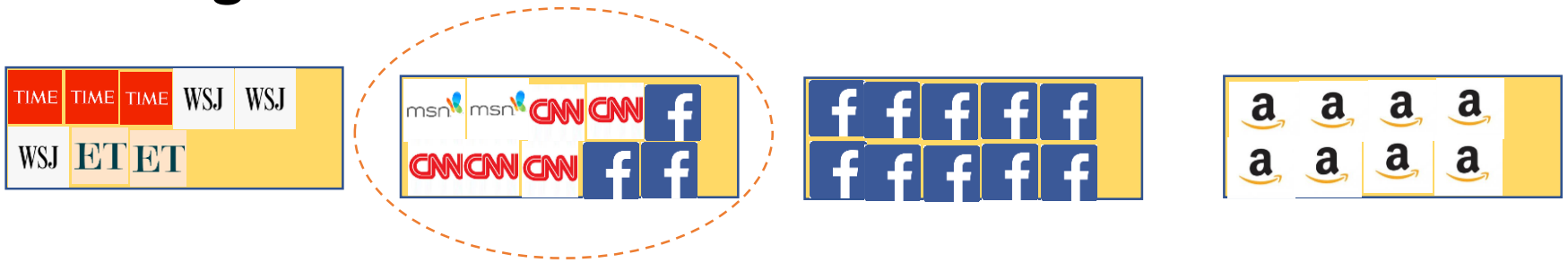
- **Partitioning + Co-location**



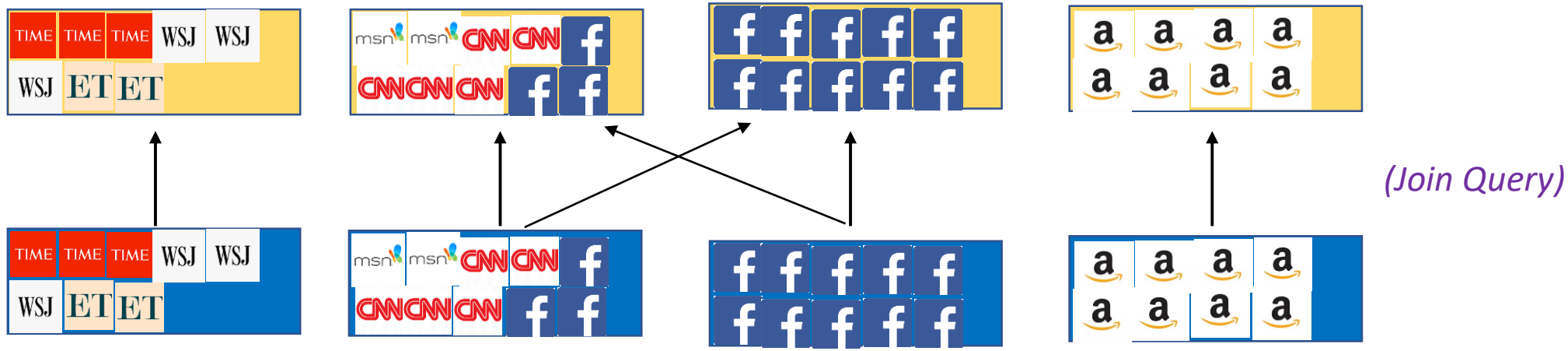
(Join Query)

Common Techniques used today

- **Partitioning**

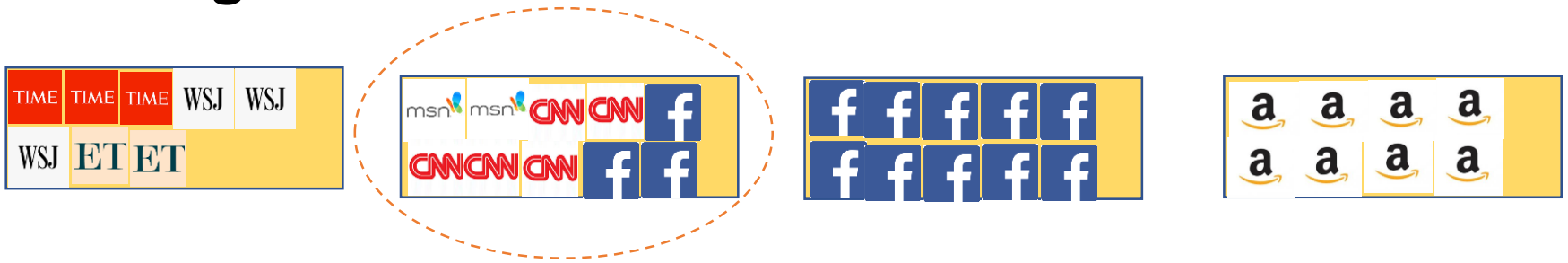


- **Partitioning + Co-location**



Common Techniques used today

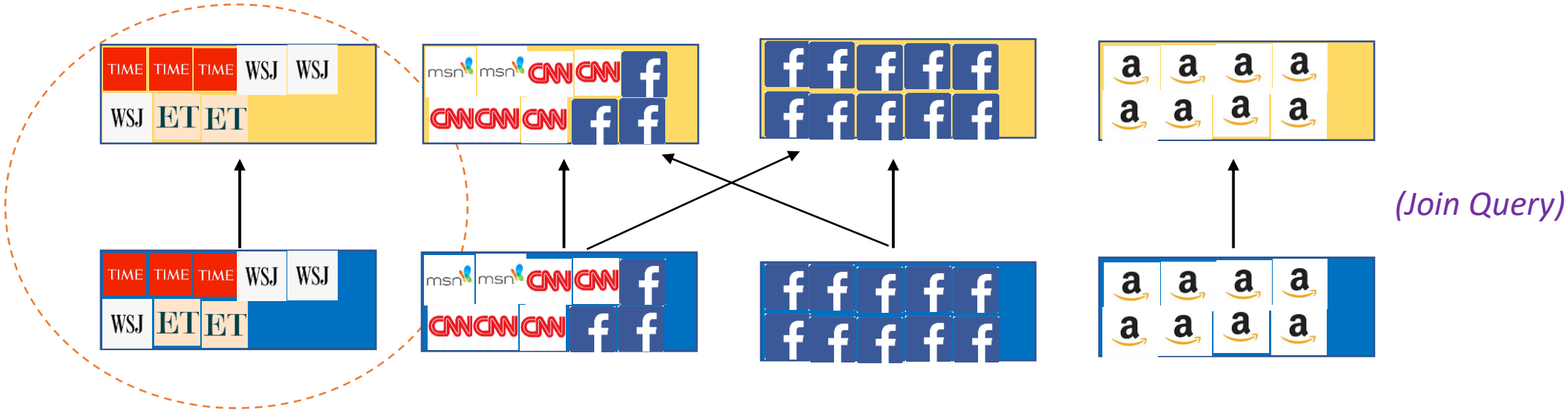
- **Partitioning**



Retrieve all click records with domain == "cnn"

(Filter Query)

- **Partitioning + Co-location**



(Join Query)

But, utility is limited

- Only one column can be chosen for partitioning or collocation
  - Helps only small set of queries that happen to filter/join on that column
  - Queries on other columns still slow!

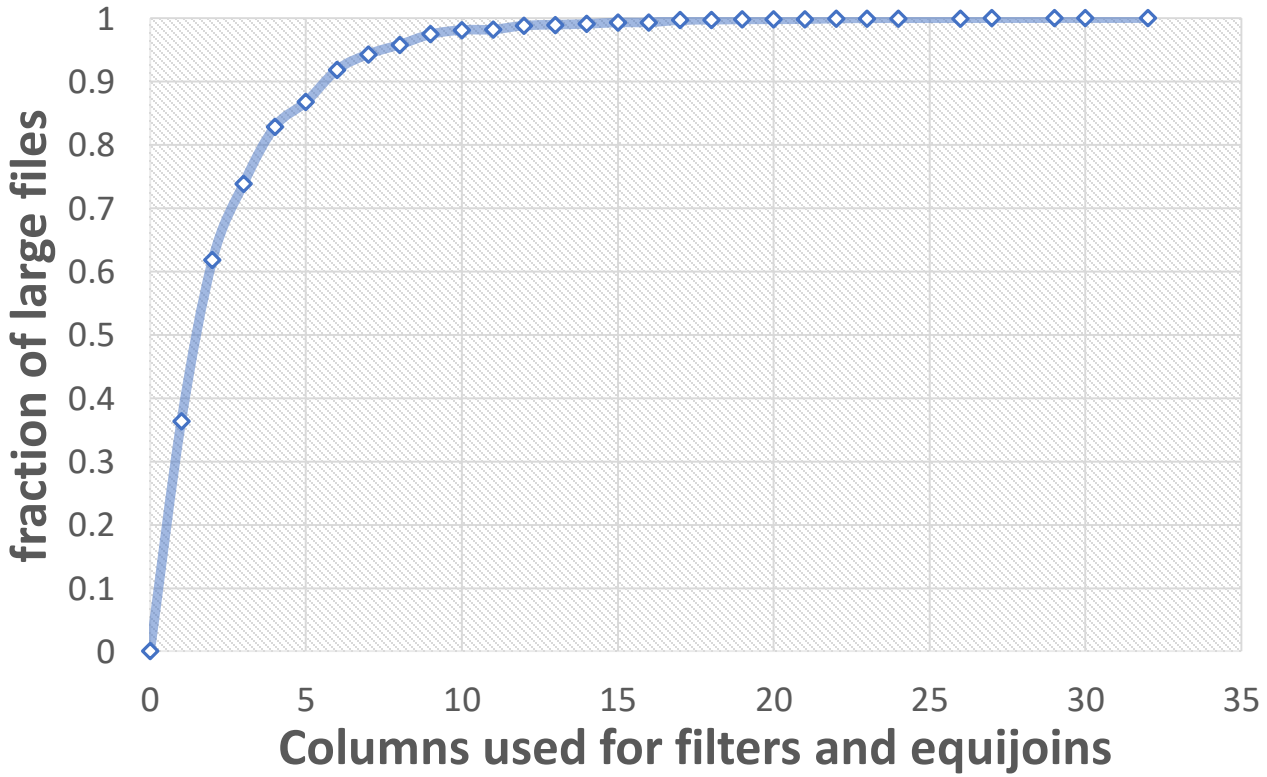
- How to get multiple partitioning/co-location strategies?
  - Only option: Maintain multiple copies of file
  - **Prohibitive storage cost**
  - **Cost of maintaining consistency**

Logical Replication

- Can we get multiple partition orders without extra storage cost?
  - Answer: Yes!
  - **Key insight**: Piggyback on replication done by cluster filesystem
- Today: **Physical replication**
  - All 3 copies of a file are identical byte-wise replicas
- **Logical replication:** Each replica of file partitioned differently
  - Benefit: 3 partition orders with no extra storage cost!

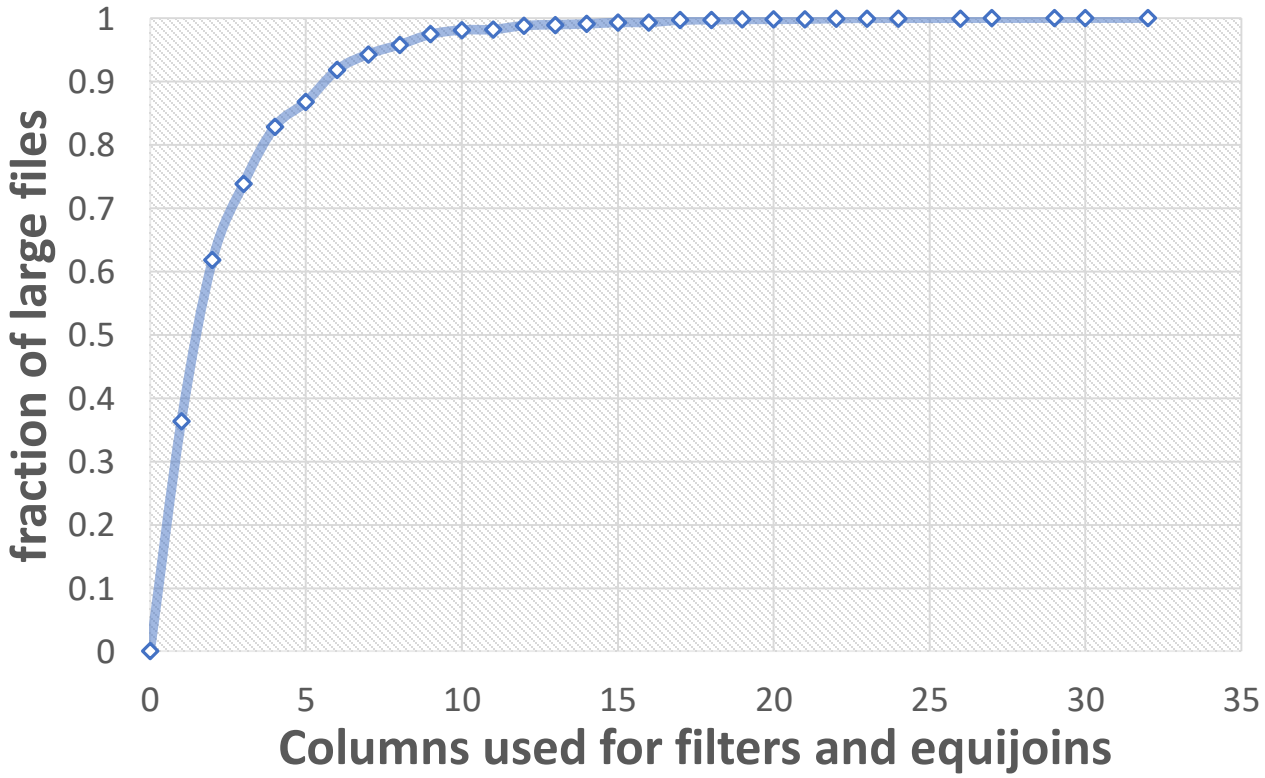
Are 3 partition orders enough?

- Analyzed one week of jobs on a production cluster
- Large input files (100GB+): How many columns used in filters / joins?



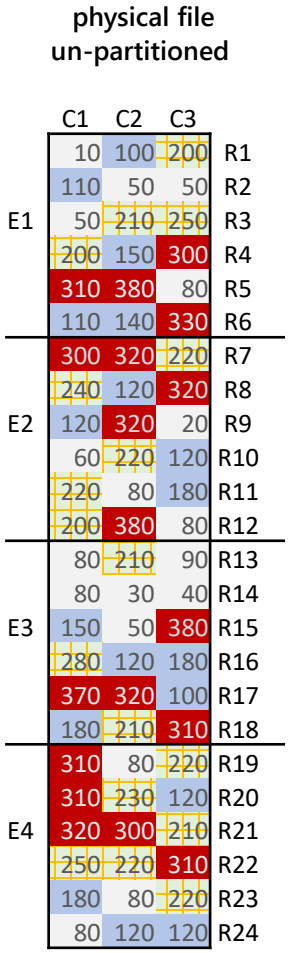
Are 3 partition orders enough?

- Analyzed one week of jobs on a production cluster
- Large input files (100GB+): How many columns used in filters / joins?

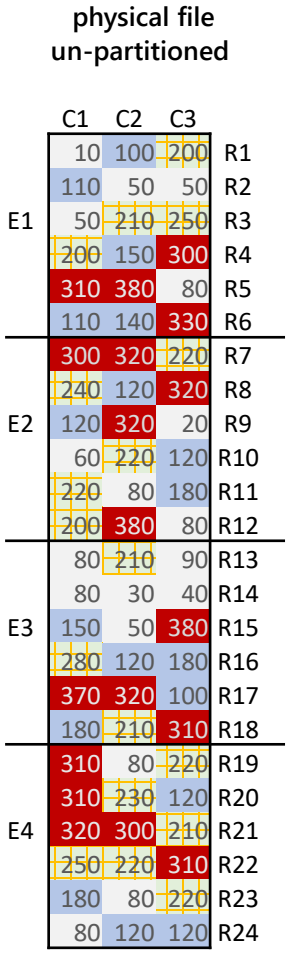


- One partition order covers only 35% of files
- **3 diff. partition orders cover 75% of files**

Challenge: Recovery cost



Challenge: Recovery cost

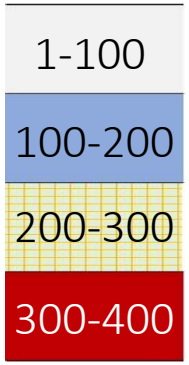


Physical Replication

↓

Recovery: Copy from another replica
(Extent: 250MB)

Challenge: Recovery cost



physical file
un-partitioned

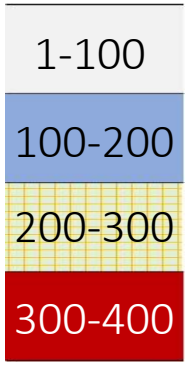
	C1	C2	C3	
	10	100	200	R1
	110	50	50	R2
E1	50	210	250	R3
	200	150	300	R4
	310	380	80	R5
	110	140	330	R6
	300	320	220	R7
	240	120	320	R8
E2	120	320	20	R9
	60	220	120	R10
	220	80	180	R11
	200	380	80	R12
	80	210	90	R13
	80	30	40	R14
E3	150	50	380	R15
	280	120	180	R16
	370	320	100	R17
	180	210	310	R18
	310	80	220	R19
	310	230	120	R20
E4	320	300	210	R21
	250	220	310	R22
	180	80	220	R23
	80	120	120	R24

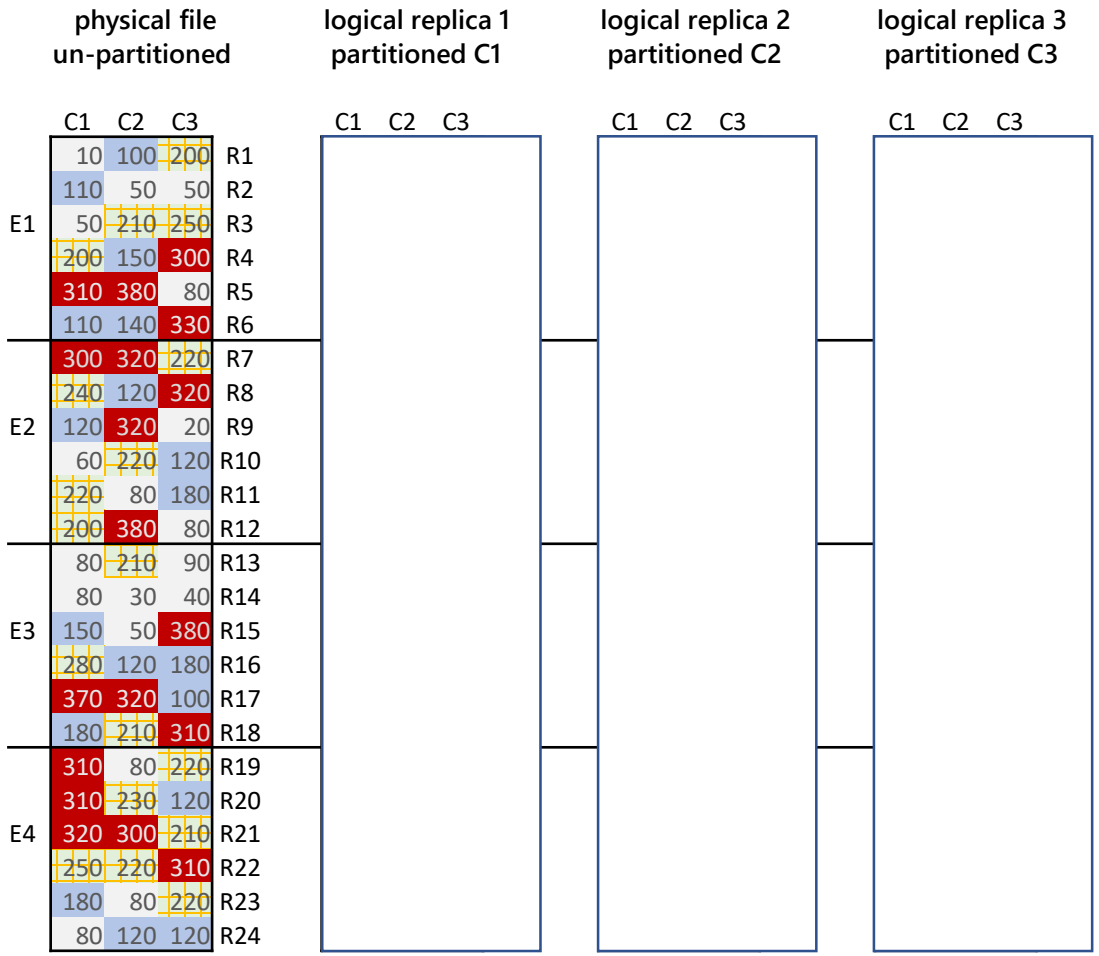
Physical Replication

↓

Recovery: Copy from another replica
(Extent: 250MB)

Challenge: Recovery cost



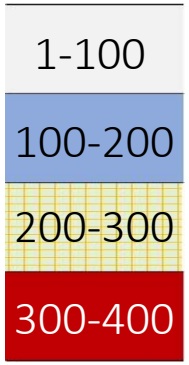


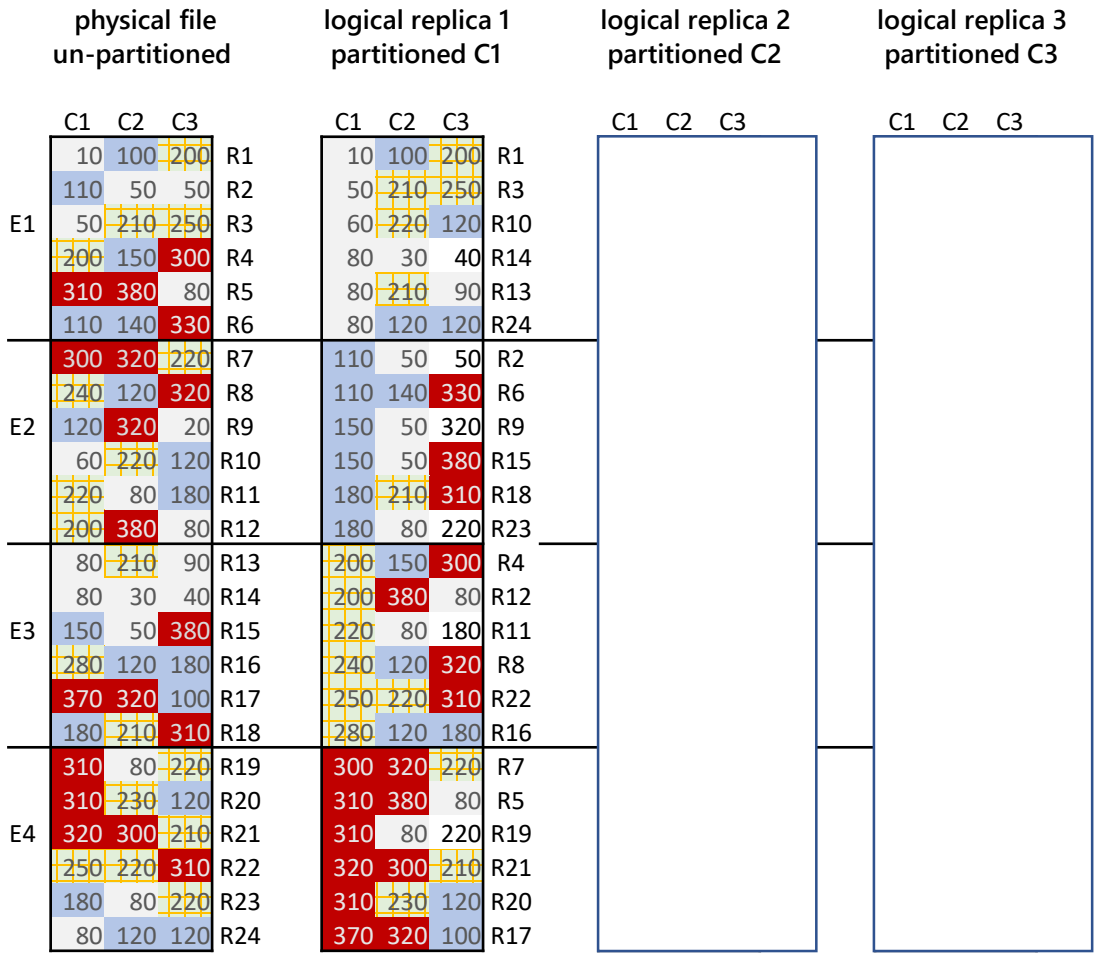
Physical Replication

↓

Recovery: Copy from another replica
(Extent: 250MB)

Challenge: Recovery cost



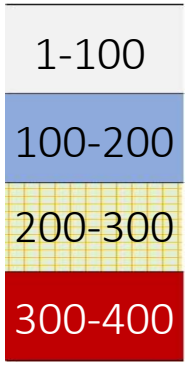


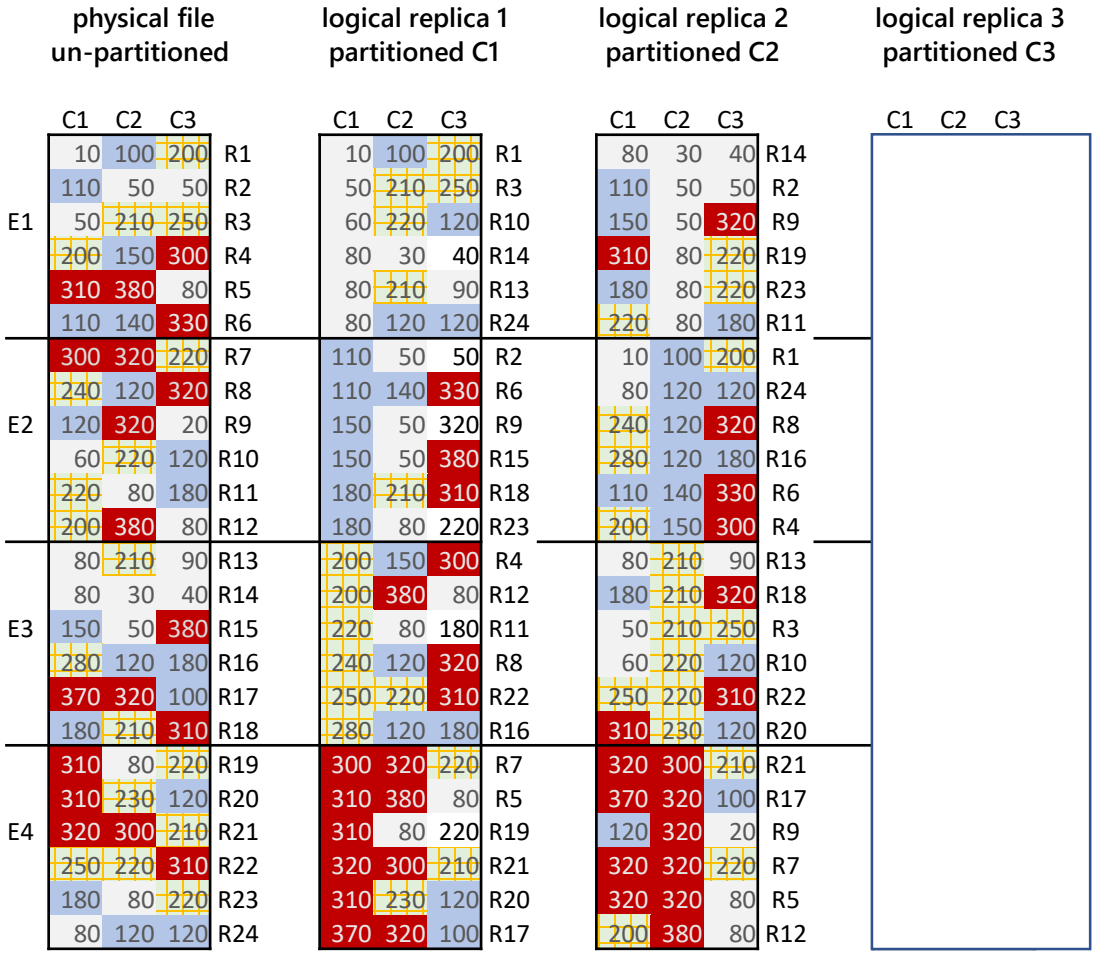
Physical Replication

↓

Recovery: Copy from another replica
(Extent: 250MB)

Challenge: Recovery cost



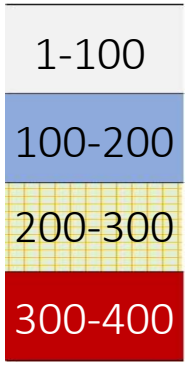


Physical Replication

↓

Recovery: Copy from another replica
(Extent: 250MB)

Challenge: Recovery cost



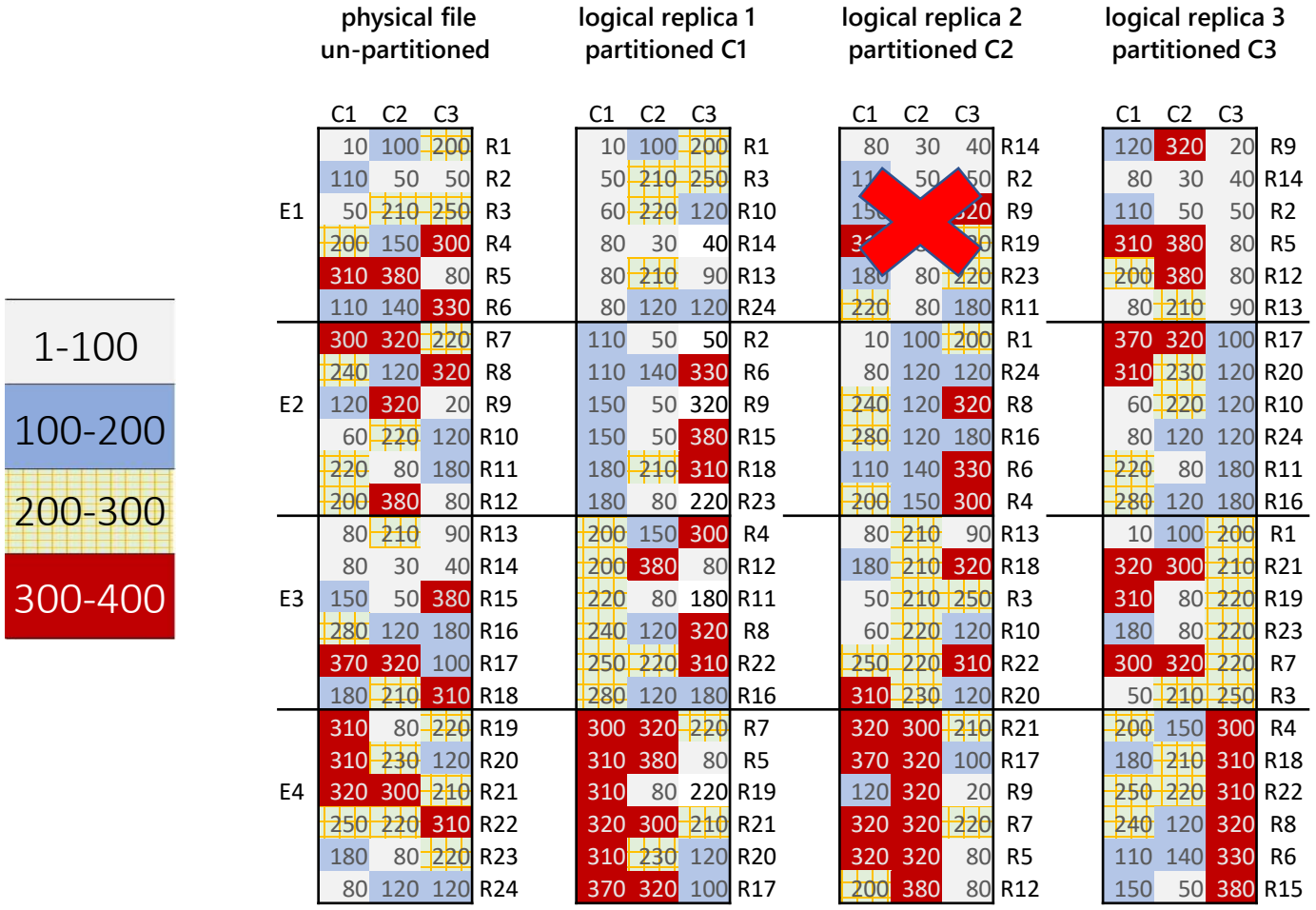
	physical file un-partitioned				logical replica 1 partitioned C1				logical replica 2 partitioned C2				logical replica 3 partitioned C3			
	C1	C2	C3		C1	C2	C3		C1	C2	C3		C1	C2	C3	
	10	100	200	R1	10	100	200	R1	80	30	40	R14	120	320	20	R9
	110	50	50	R2	50	210	250	R3	110	50	50	R2	80	30	40	R14
E1	50	210	250	R3	60	220	120	R10	150	50	320	R9	110	50	50	R2
	200	150	300	R4	80	30	40	R14	310	80	220	R19	310	380	80	R5
	310	380	80	R5	80	210	90	R13	180	80	220	R23	200	380	80	R12
	110	140	330	R6	80	120	120	R24	220	80	180	R11	80	210	90	R13
	300	320	220	R7	110	50	50	R2	10	100	200	R1	370	320	100	R17
	240	120	320	R8	110	140	330	R6	80	120	120	R24	310	230	120	R20
E2	120	320	20	R9	150	50	320	R9	240	120	320	R8	60	220	120	R10
	60	220	120	R10	150	50	380	R15	280	120	180	R16	80	120	120	R24
	220	80	180	R11	180	210	310	R18	110	140	330	R6	220	80	180	R11
	200	380	80	R12	180	80	220	R23	200	150	300	R4	280	120	180	R16
	80	210	90	R13	200	150	300	R4	80	210	90	R13	10	100	200	R1
	80	30	40	R14	200	380	80	R12	180	210	320	R18	320	300	210	R21
E3	150	50	380	R15	220	80	180	R11	50	210	250	R3	310	80	220	R19
	280	120	180	R16	240	120	320	R8	60	220	120	R10	180	80	220	R23
	370	320	100	R17	250	220	310	R22	250	220	310	R22	300	320	220	R7
	180	210	310	R18	280	120	180	R16	310	230	120	R20	50	210	250	R3
	310	80	220	R19	300	320	220	R7	320	300	210	R21	200	150	300	R4
	310	230	120	R20	310	380	80	R5	370	320	100	R17	180	210	310	R18
E4	320	300	210	R21	310	80	220	R19	120	320	20	R9	250	220	310	R22
	250	220	310	R22	320	300	210	R21	320	320	220	R7	240	120	320	R8
	180	80	220	R23	310	230	120	R20	320	320	80	R5	110	140	330	R6
	80	120	120	R24	370	320	100	R17	200	380	80	R12	150	50	380	R15

Physical Replication

↓

Recovery: Copy from another replica
(Extent: 250MB)

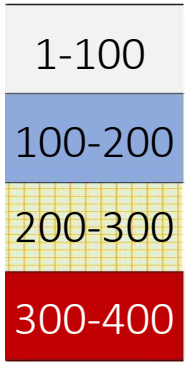
Challenge: Recovery cost

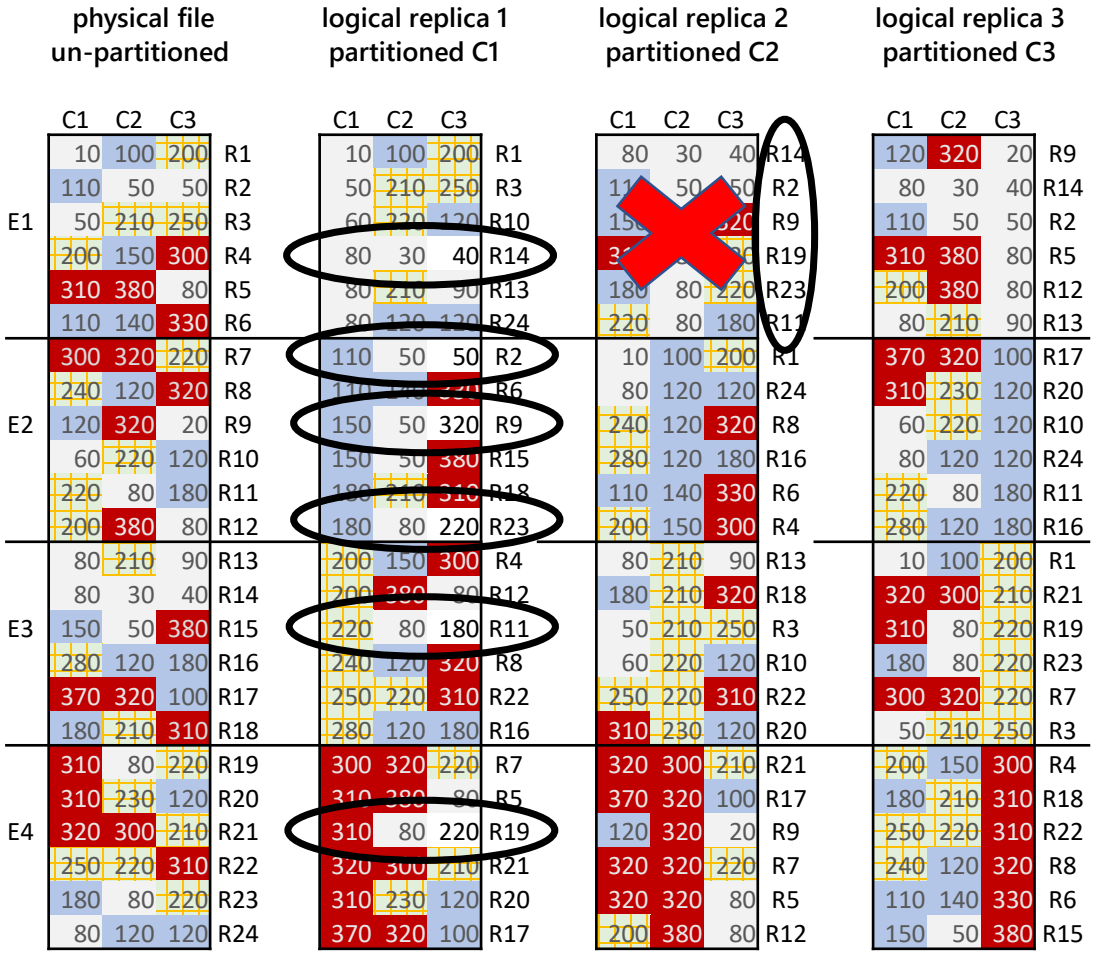


Physical Replication

Recovery: Copy from another replica (Extent: 250MB)

Challenge: Recovery cost

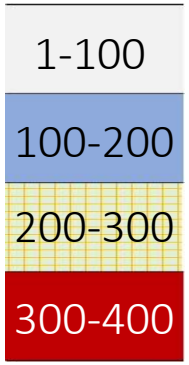


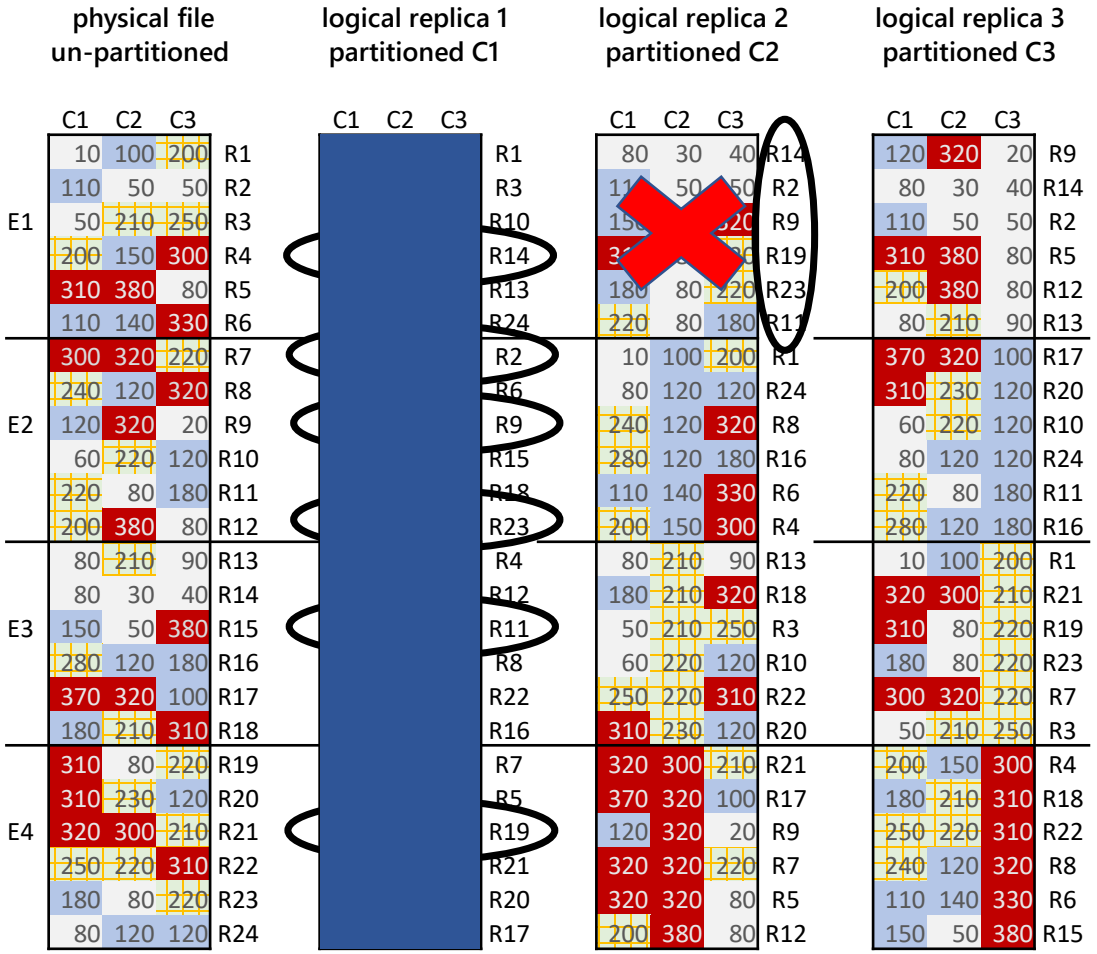


Physical Replication

Recovery: Copy from another replica
(Extent: 250MB)

Challenge: Recovery cost

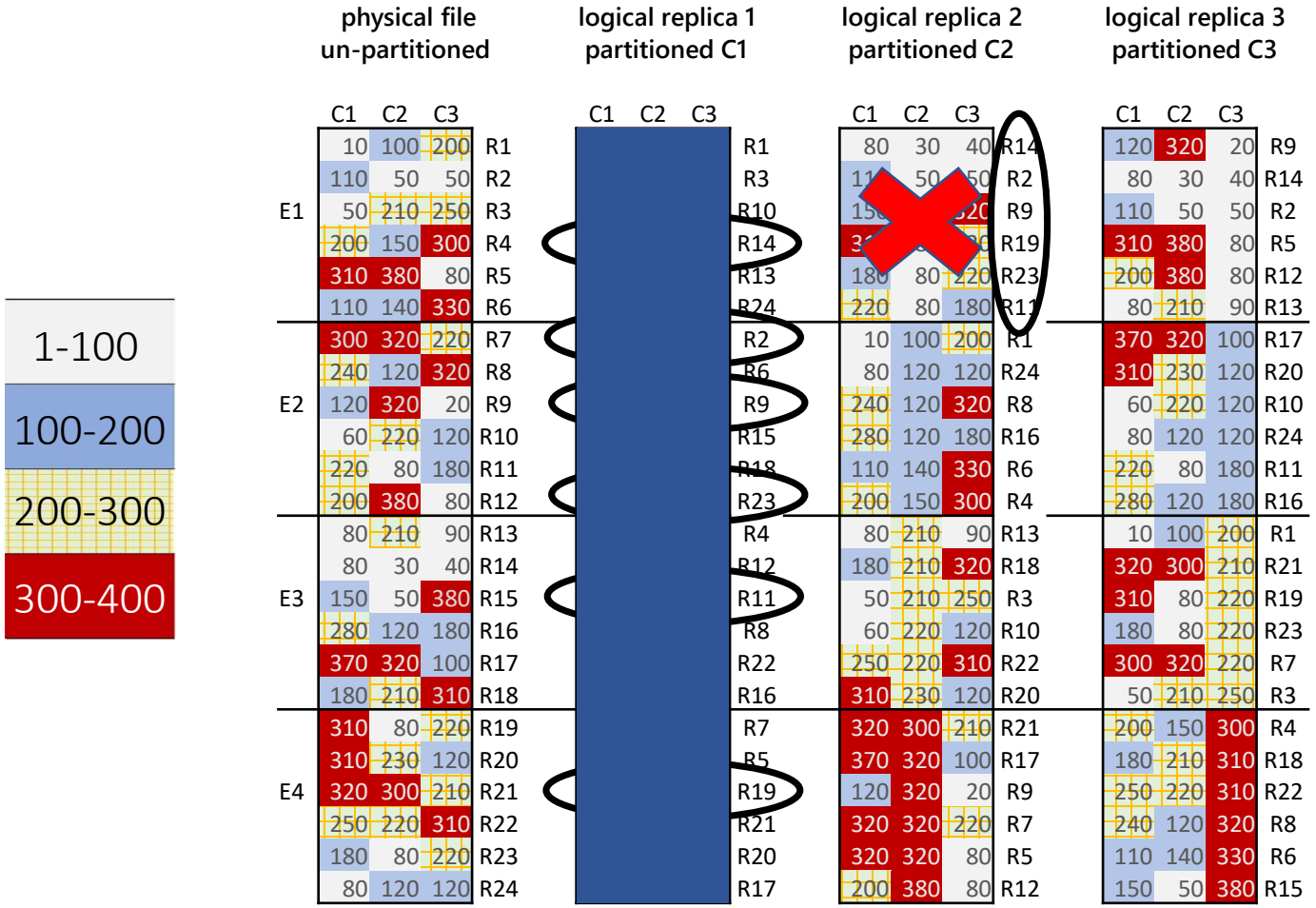




Physical Replication

Recovery: Copy from another replica
(Extent: 250MB)

Challenge: Recovery cost



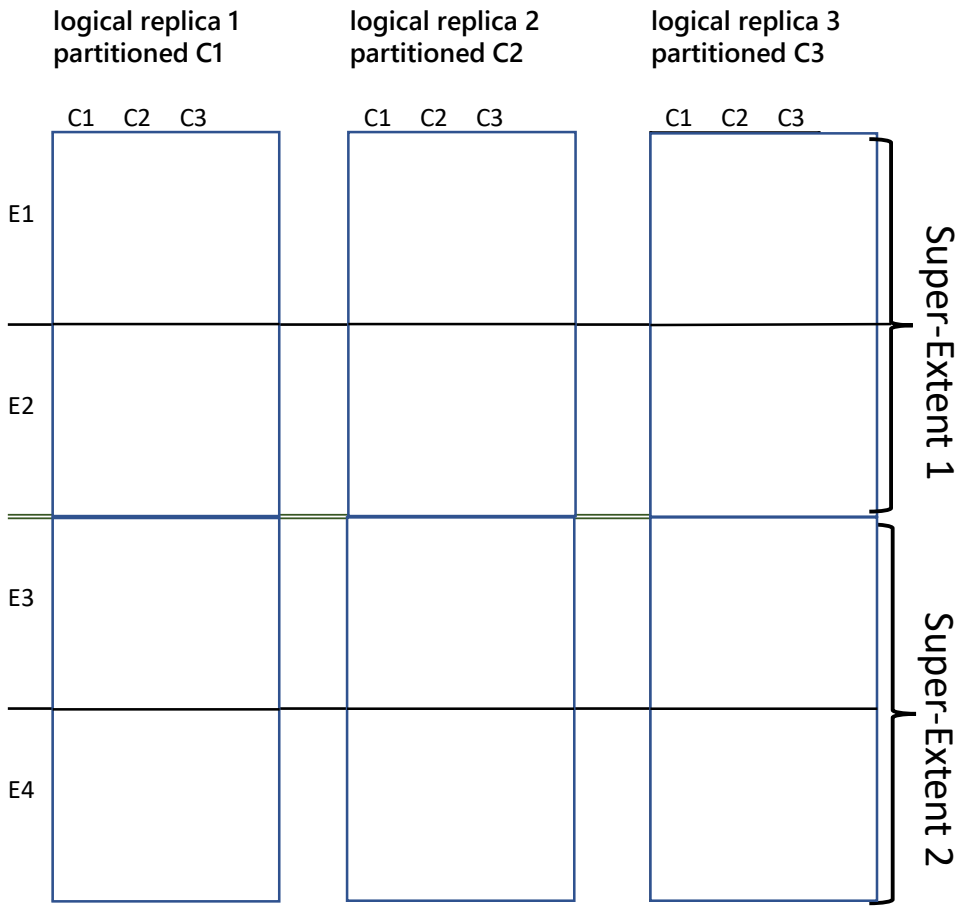
Physical Replication

Recovery: Copy from another replica (Extent: 250MB)

Naïve Logical Replication

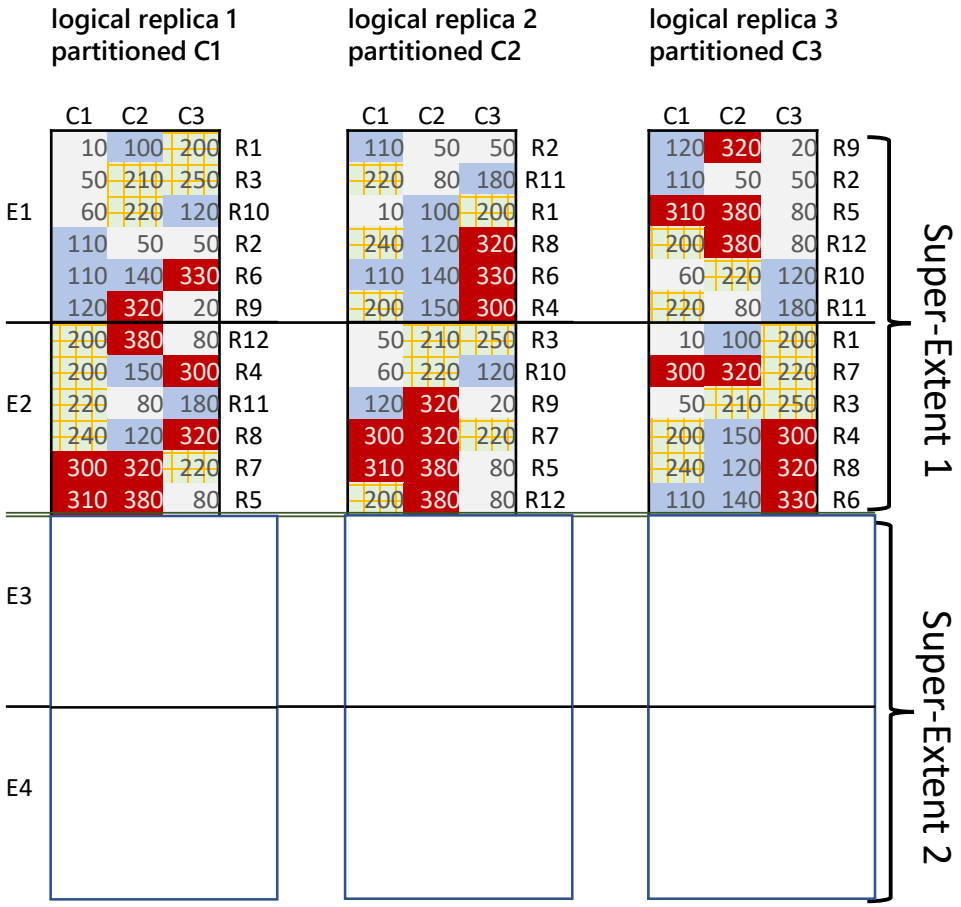
Prohibitive recovery cost!

Super Extents



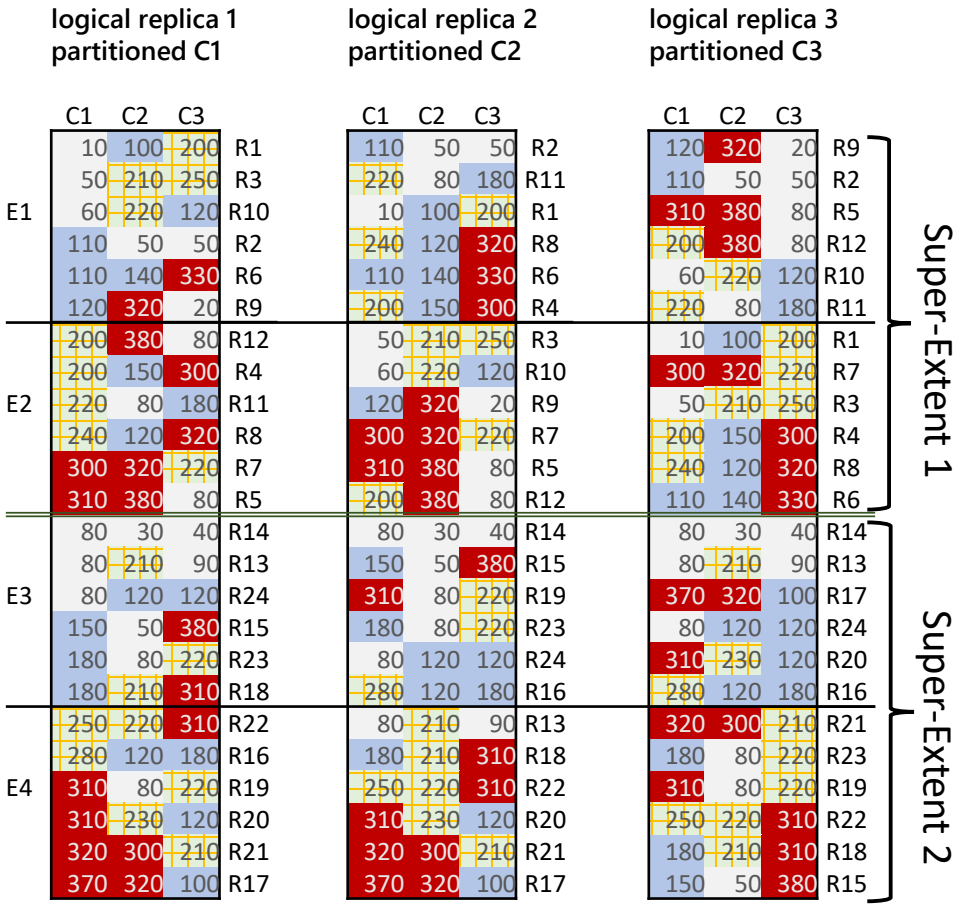
- ***Super Extent***
  - Contiguous group of fixed # of extents
  - Super extent size
  - Re-order records at super-extent level

Super Extents



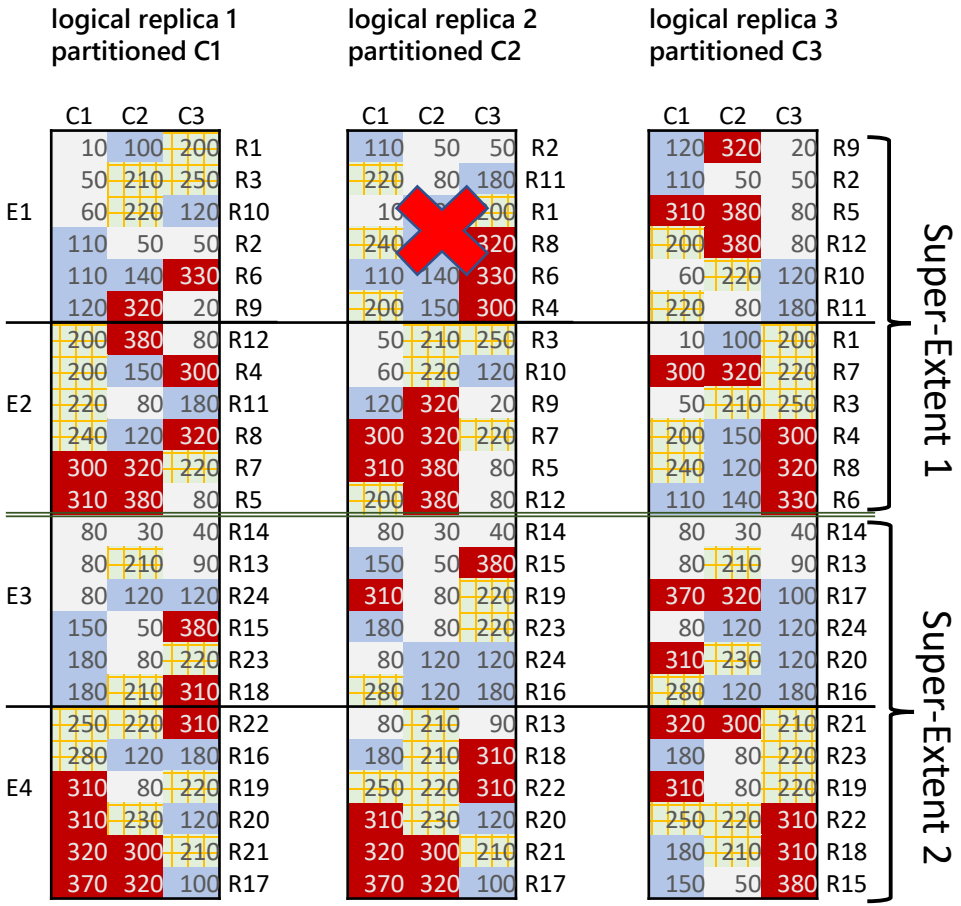
- ***Super Extent***
  - Contiguous group of fixed # of extents
  - Super extent size
  - Re-order records at super-extent level

Super Extents



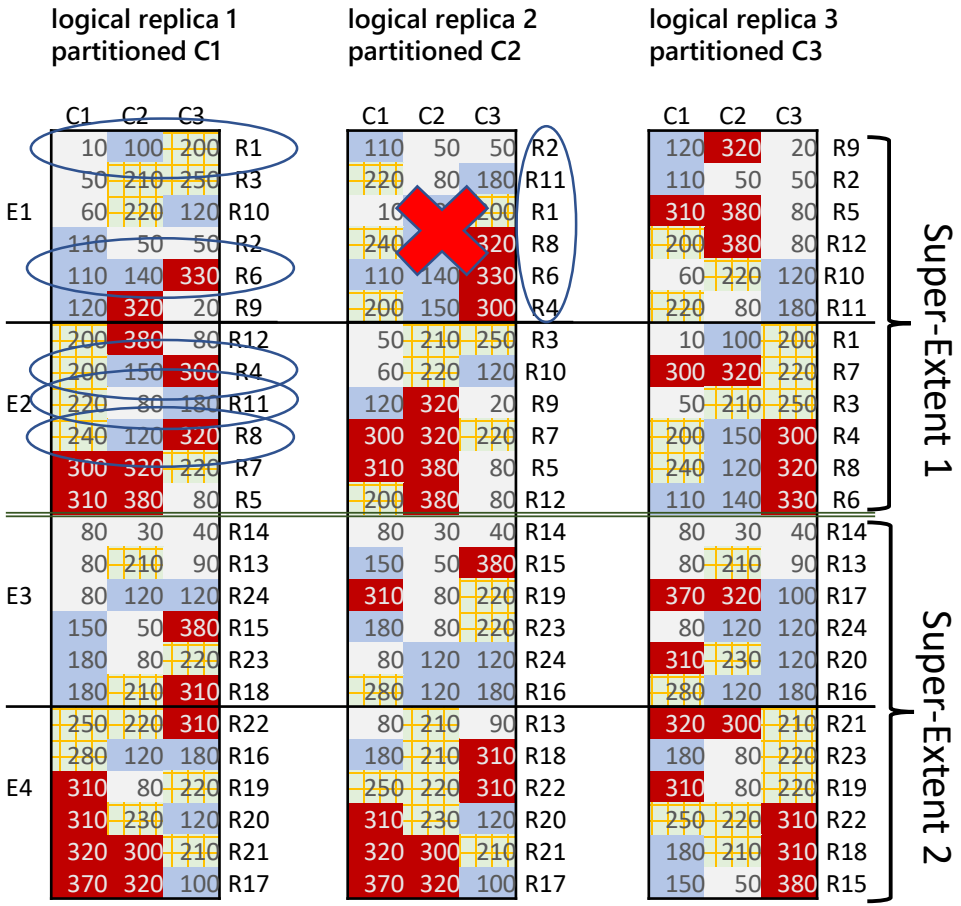
- ***Super Extent***
  - Contiguous group of fixed # of extents
  - Super extent size
  - Re-order records at super-extent level

Super Extents



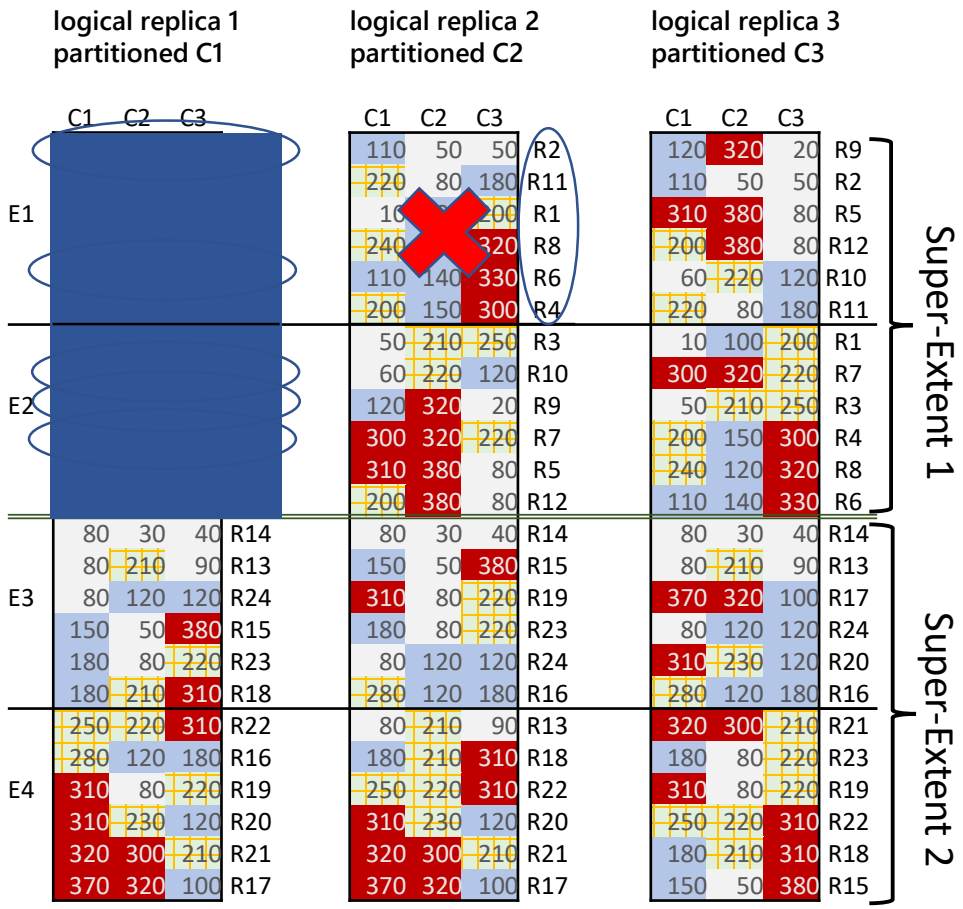
- ***Super Extent***
  - Contiguous group of fixed # of extents
  - Super extent size
  - Re-order records at super-extent level

Super Extents



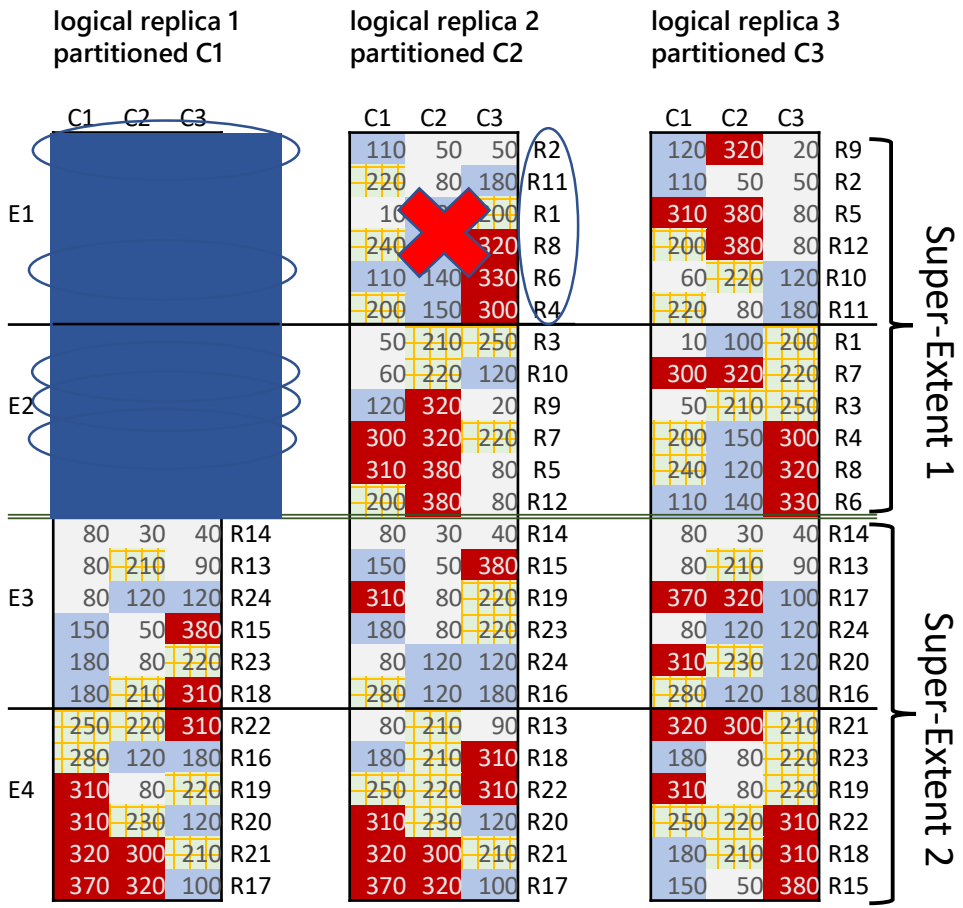
- ***Super Extent***
  - Contiguous group of fixed # of extents
  - Super extent size
  - Re-order records at super-extent level

Super Extents



- ***Super Extent***
  - Contiguous group of fixed # of extents
  - Super extent size
  - Re-order records at super-extent level

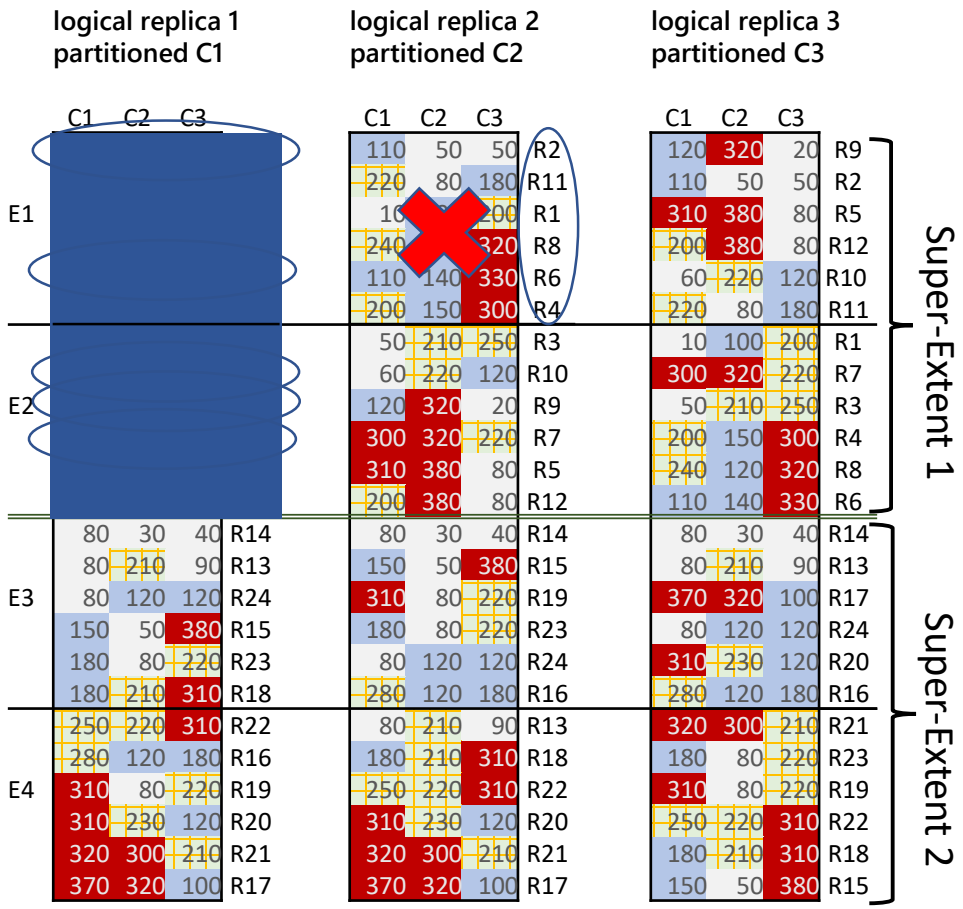
Super Extents



- ***Super Extent***
  - Contiguous group of fixed # of extents
  - Super extent size
  - Re-order records at super-extent level

- Consequence:
  - partial ordering v/s global ordering
  - Benefits = func(super extent size)

Super Extents

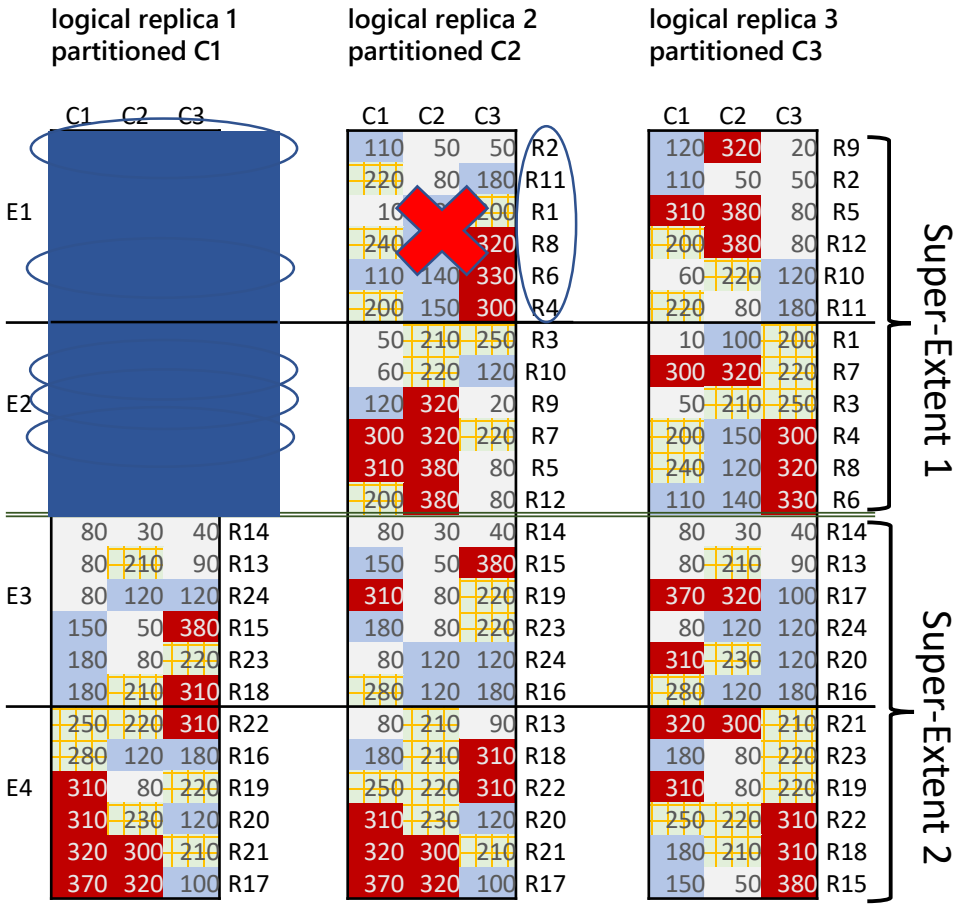


- ***Super Extent***
  - Contiguous group of fixed # of extents
  - Super extent size
  - Re-order records at super-extent level

- Consequence:
  - partial ordering v/s global ordering
  - Benefits = func(super extent size)

- In practice: **Super extent size = 100**

Super Extents



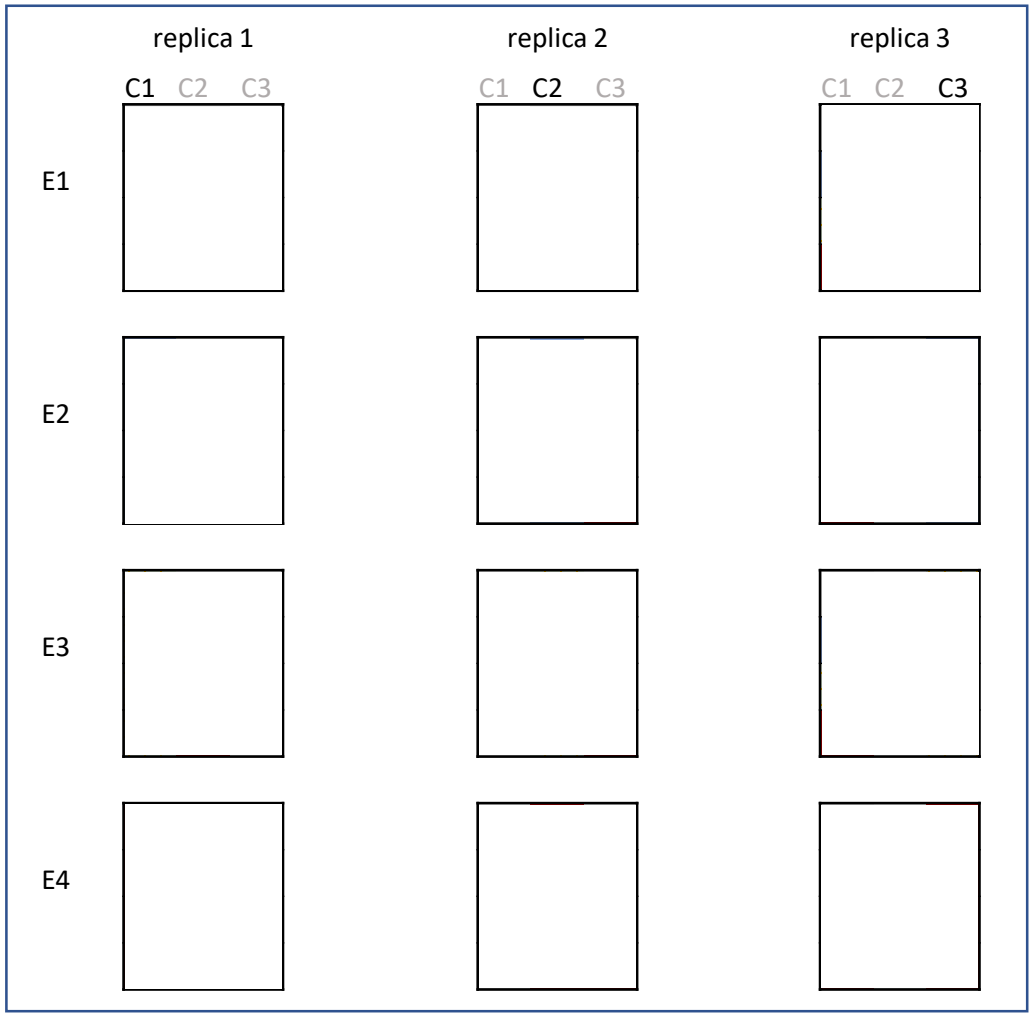
- ***Super Extent***
  - Contiguous group of fixed # of extents
  - Super extent size
  - Re-order records at super-extent level

- Consequence:
  - partial ordering v/s global ordering
  - Benefits = func(super extent size)

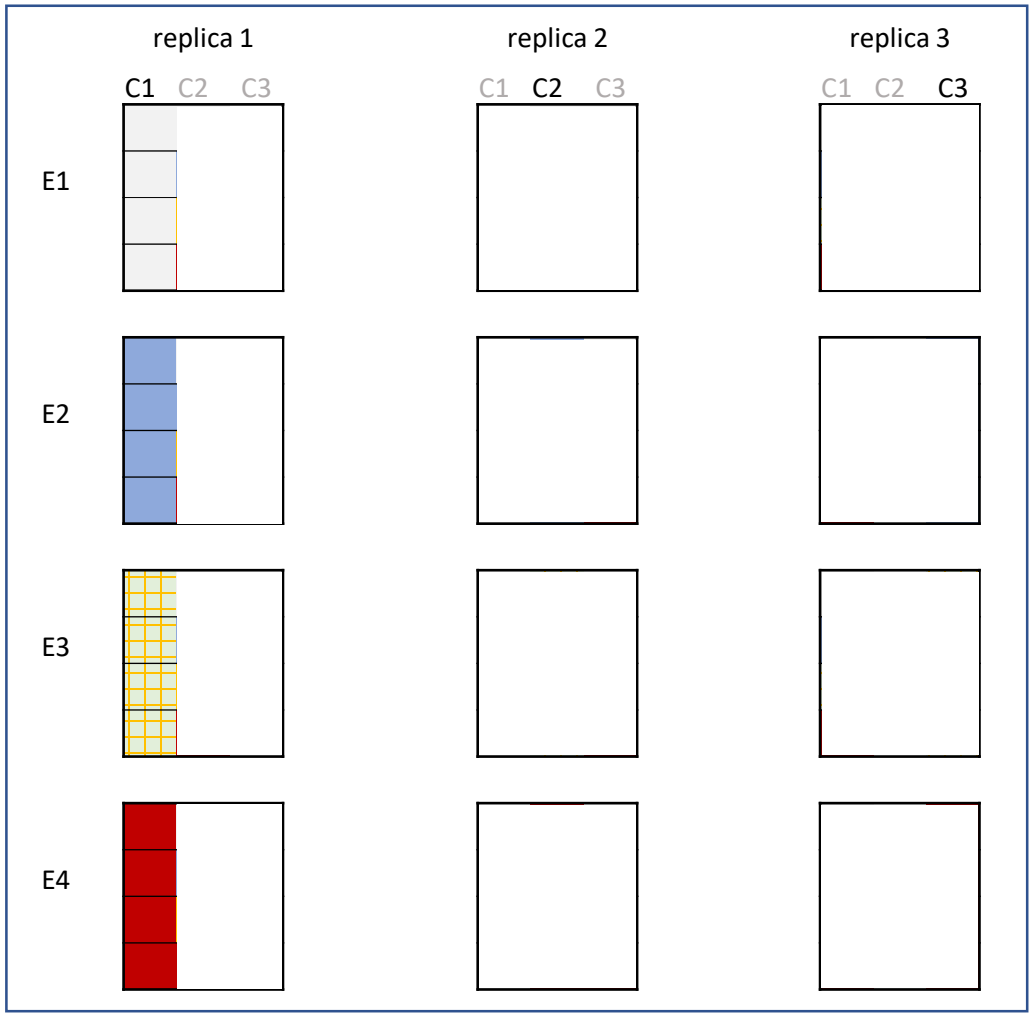
- In practice: **Super extent size = 100**

Recovery cost still 100x!

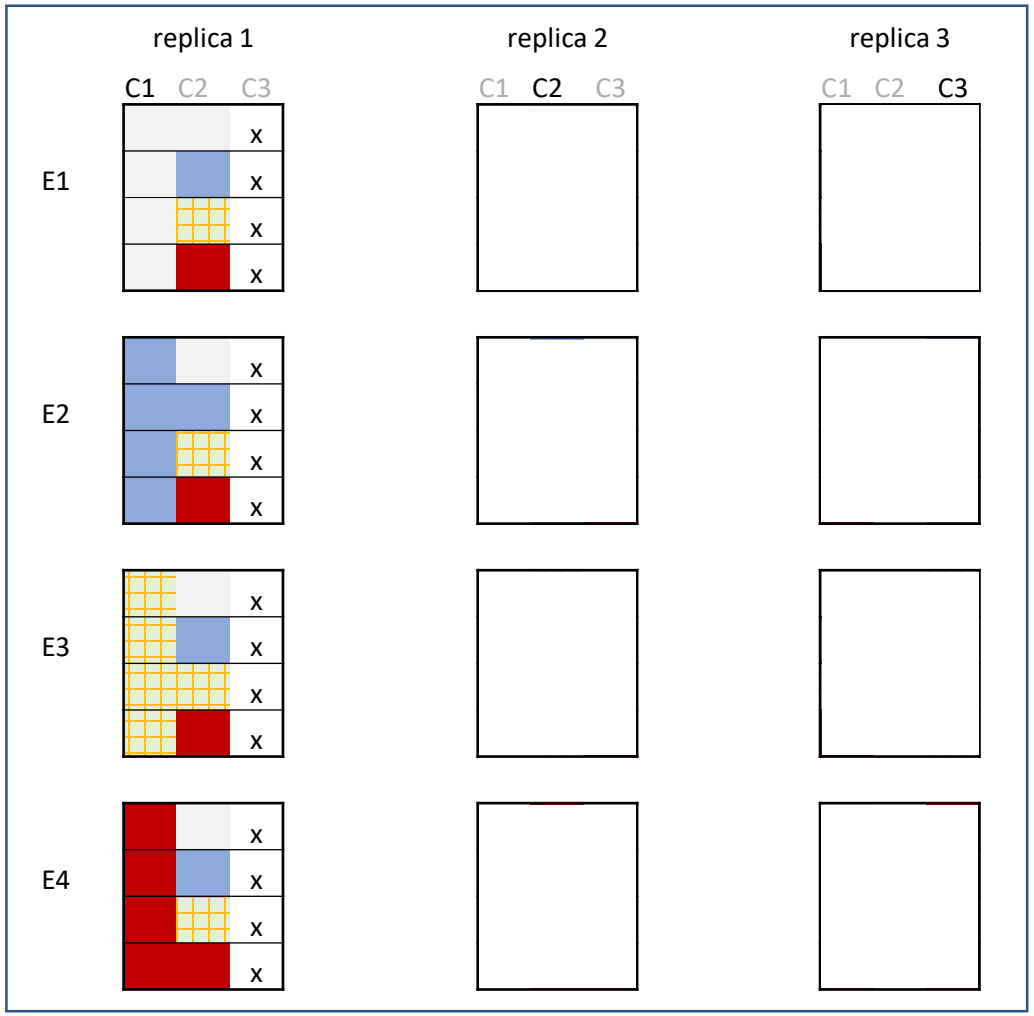
Chained Intra-extent bucketing



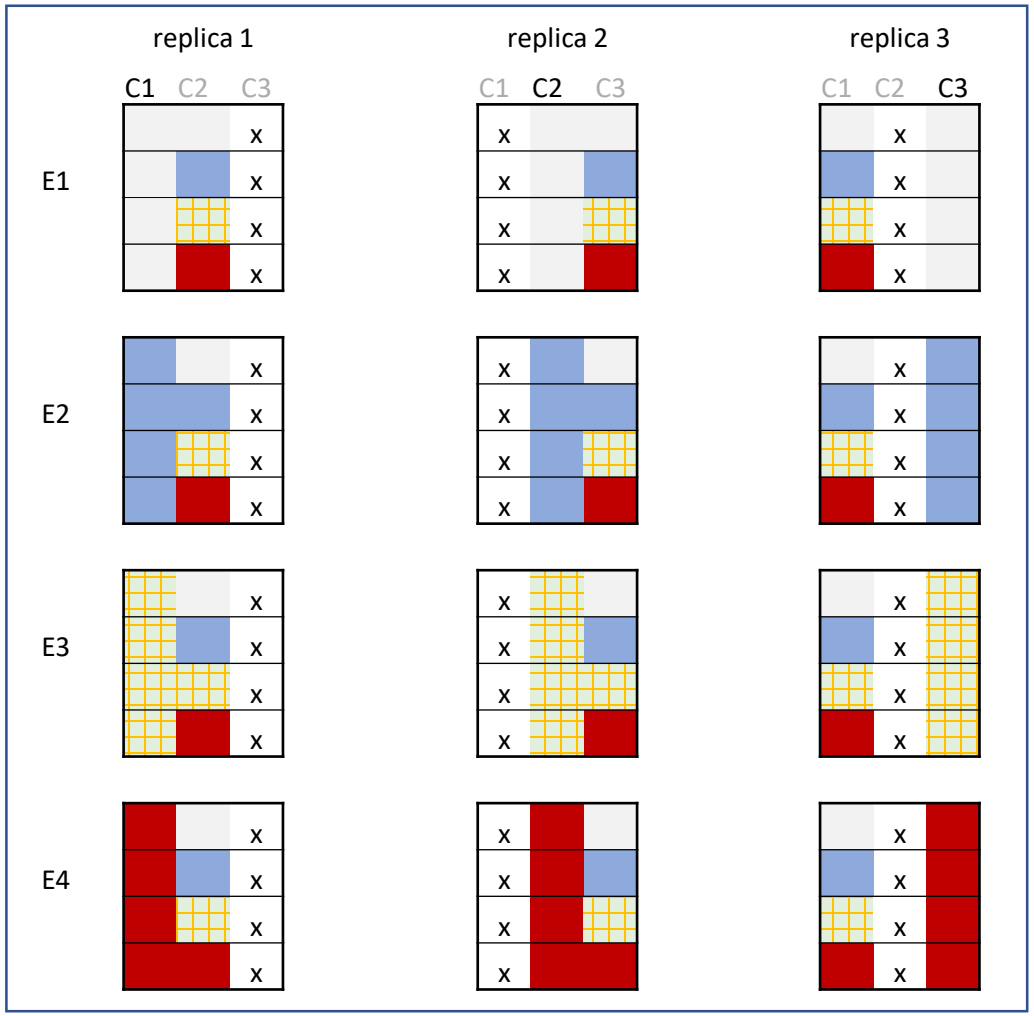
Chained Intra-extent bucketing



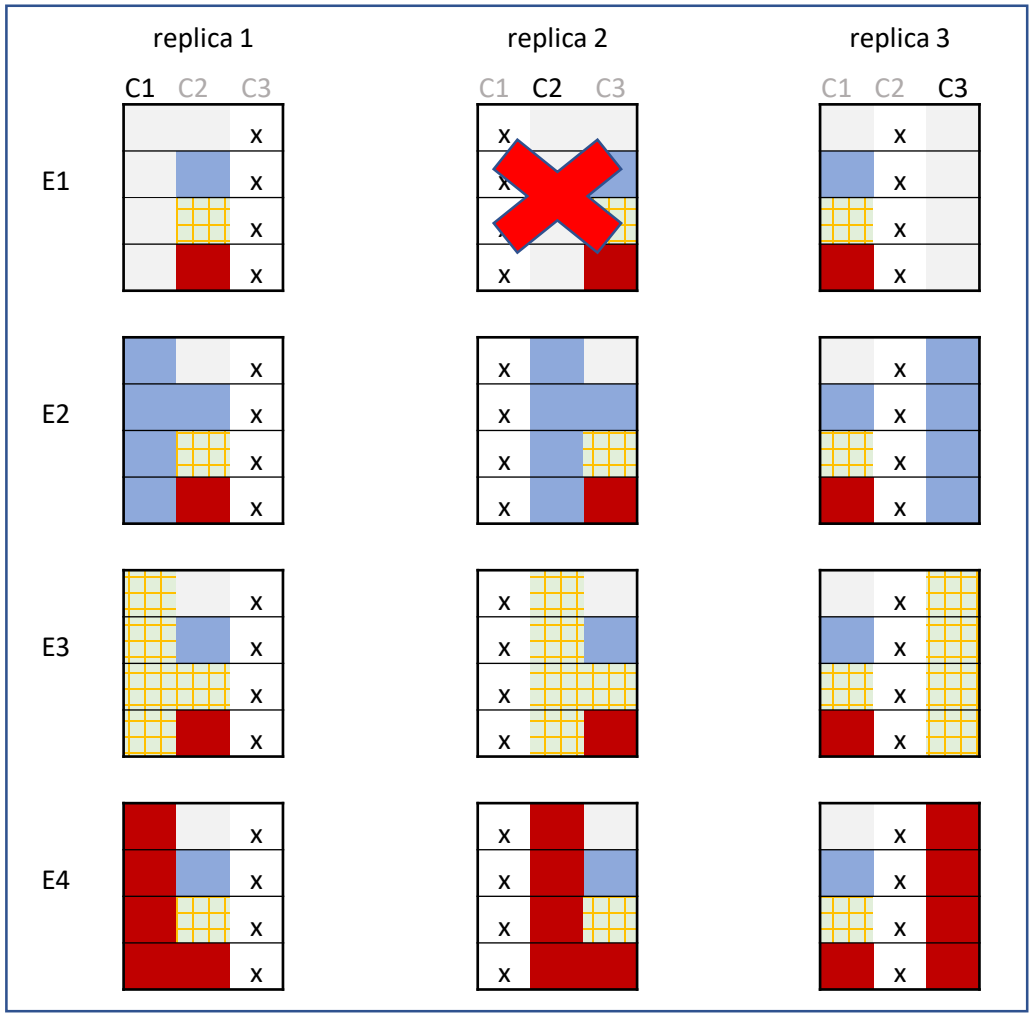
Chained Intra-extent bucketing



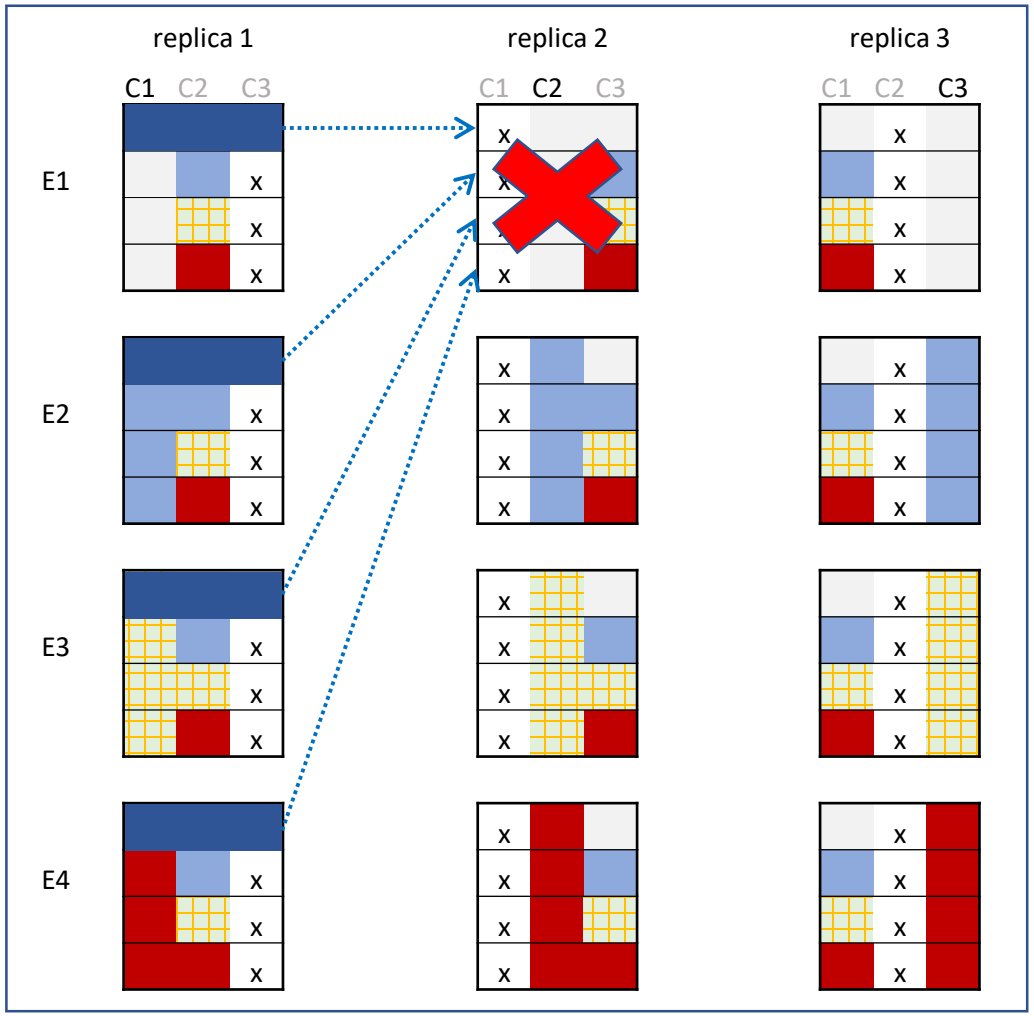
Chained Intra-extent bucketing



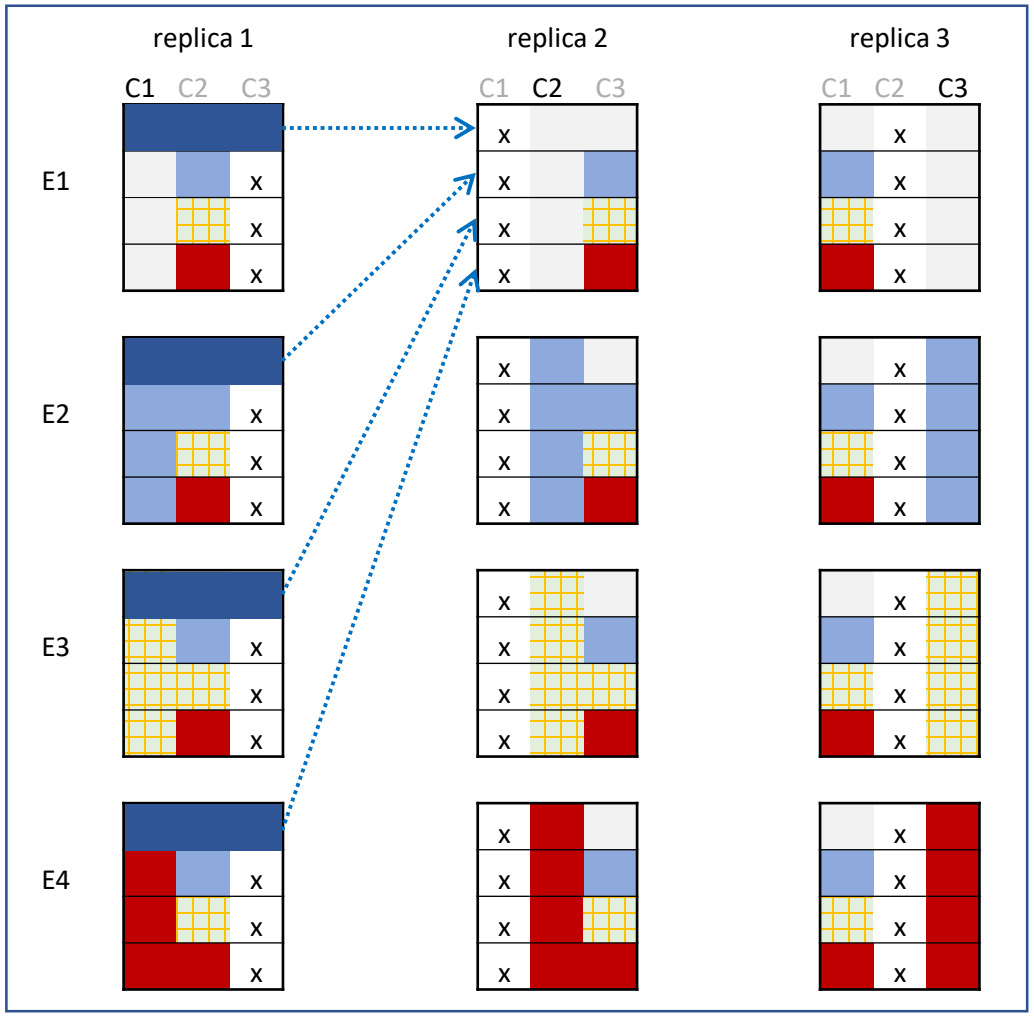
Chained Intra-extent bucketing



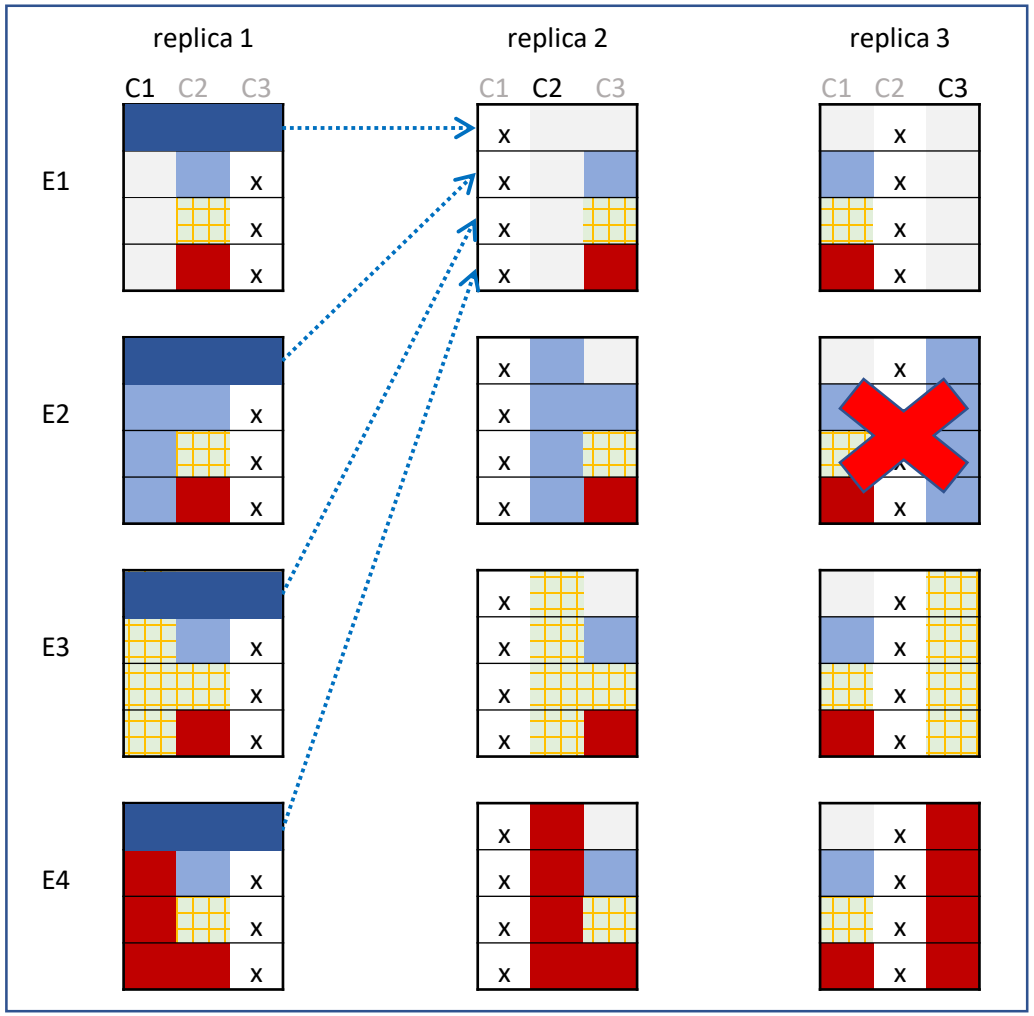
Chained Intra-extent bucketing



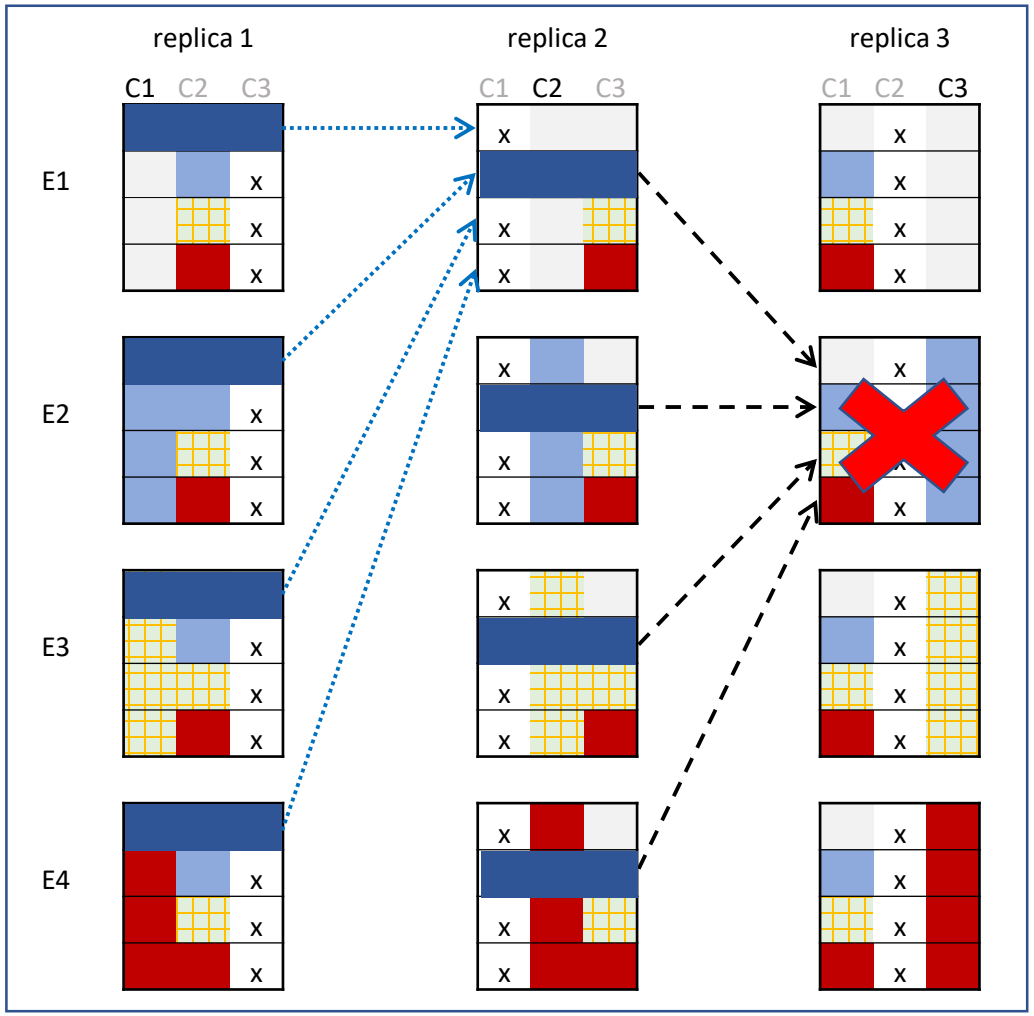
Chained Intra-extent bucketing



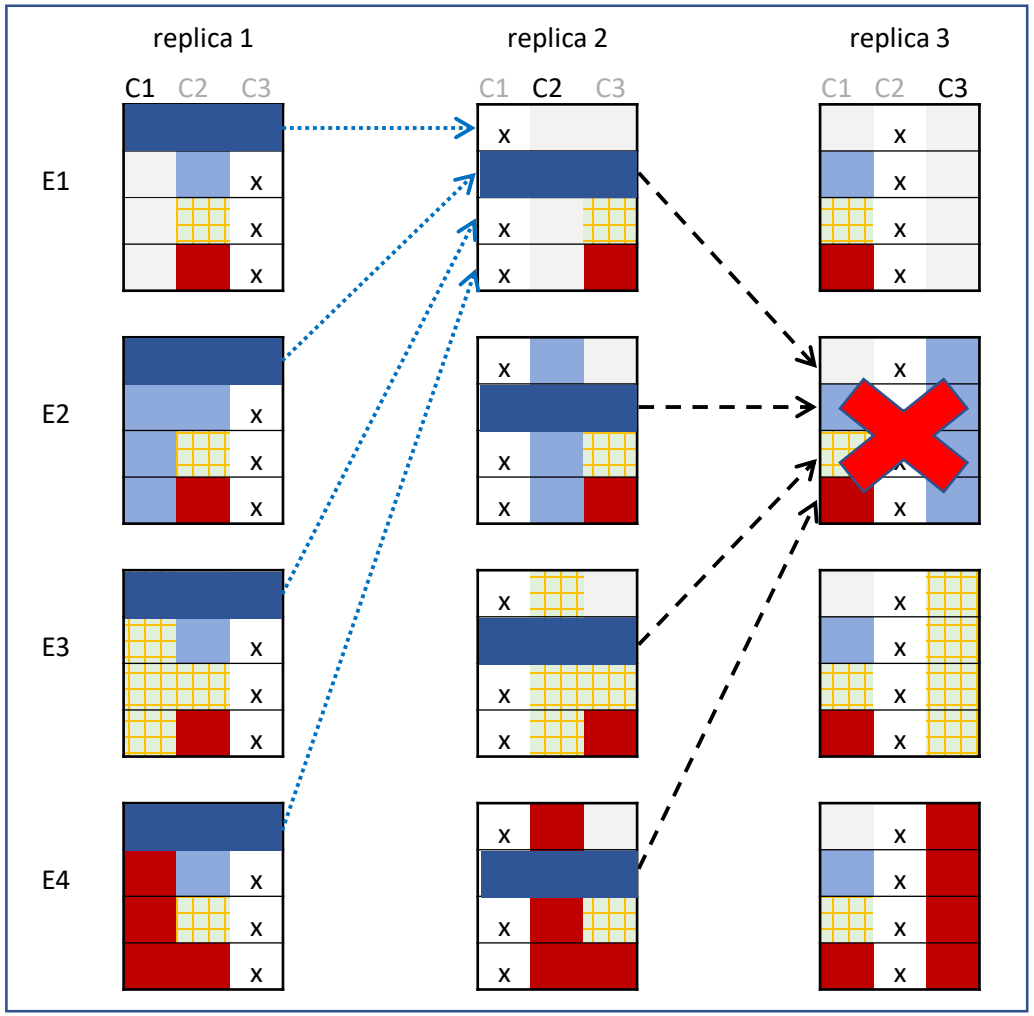
Chained Intra-extent bucketing



Chained Intra-extent bucketing



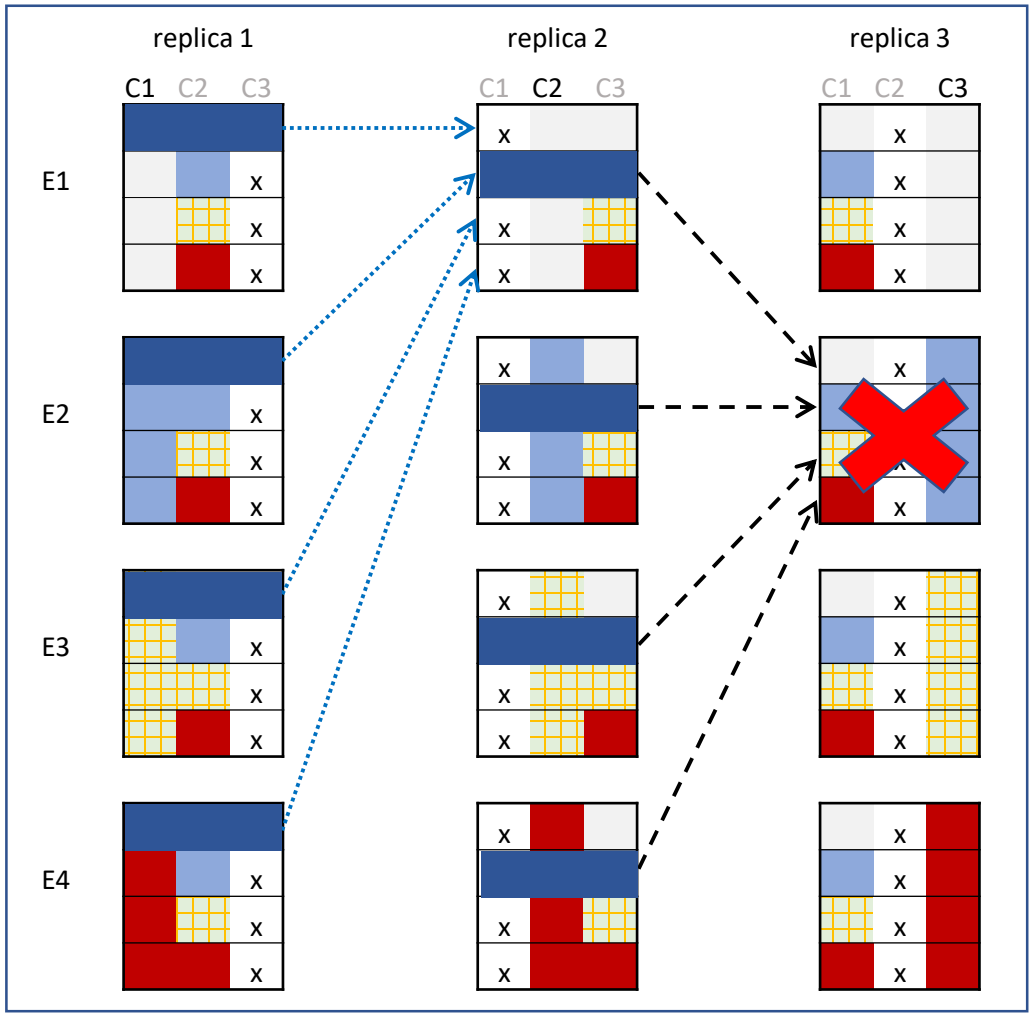
Chained Intra-extent bucketing



Same recovery cost
as Physical Replication

(in terms of Disk & Network I/O)

Chained Intra-extent bucketing



Same recovery cost
as Physical Replication

(in terms of Disk & Network I/O)

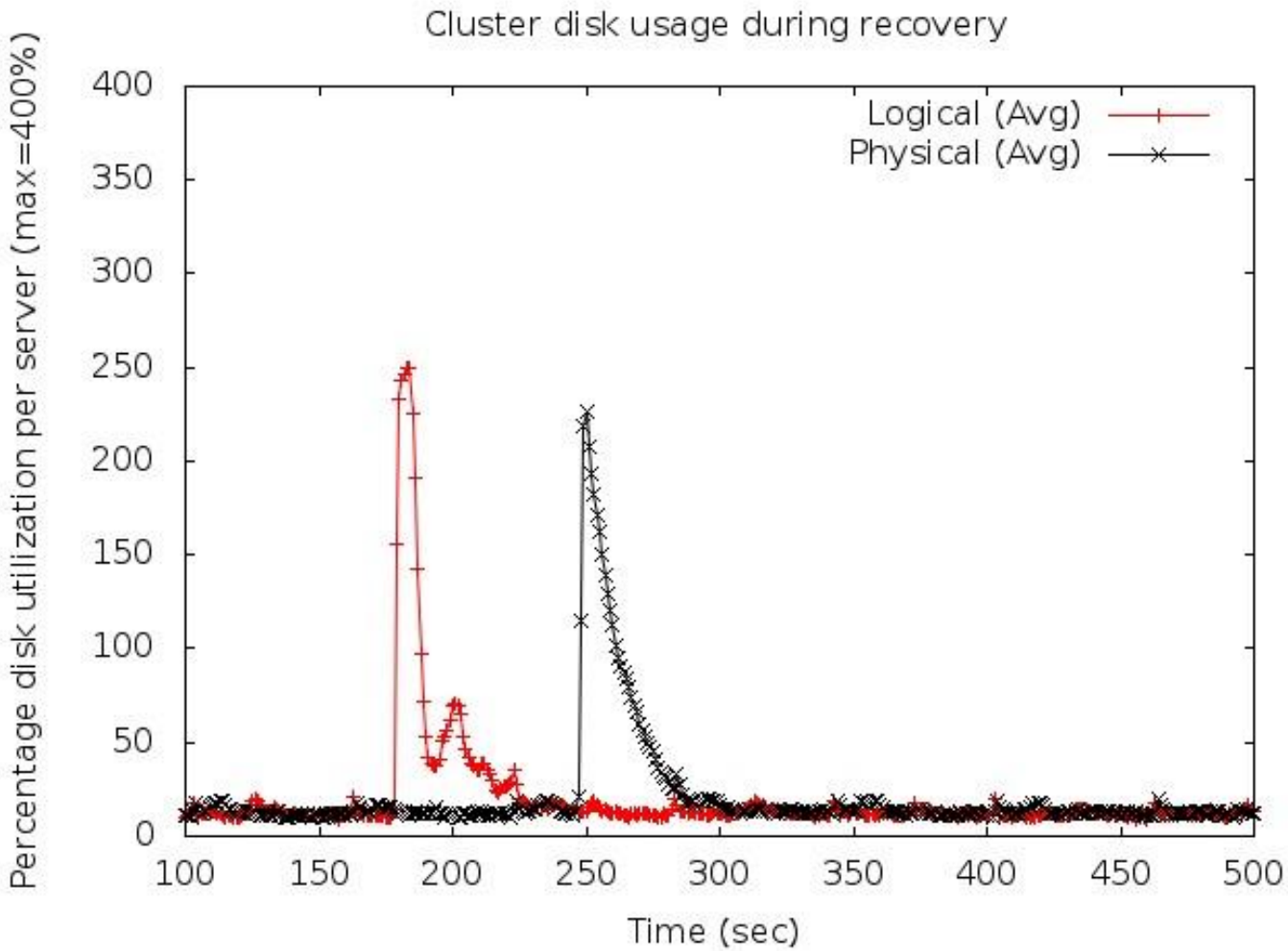
- Super extent size = 100
  - => Size(Intra-bucket) = 2.5MB
  - Disk seek amortized over transfer

Recovery Cost Evaluation

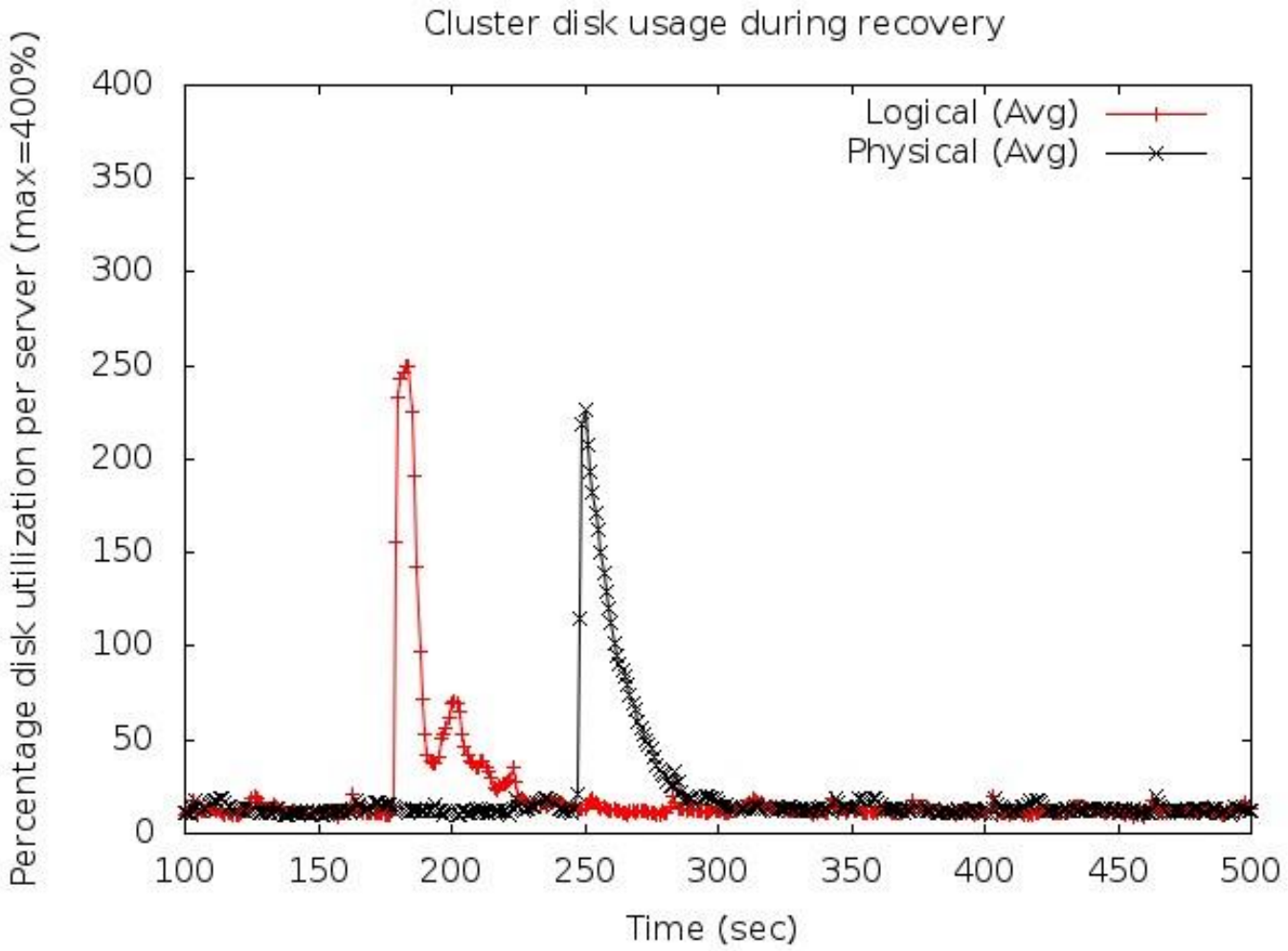
- Setup
  - Dedicated cluster of 500 machines (20 racks x 25 machines)
  - Machine configuration
    - 2.4GHz Xeon processor w/ 24 H/W threads
    - 128GB RAM
    - 4x 5TB HDD
    - 4x 500GB SSD

- Recovery Experiment
  - Ingested large amount of data
  - Took down 1 rack of machines
  - Measured disk & network utilization

Recovery cost: Disk I/O

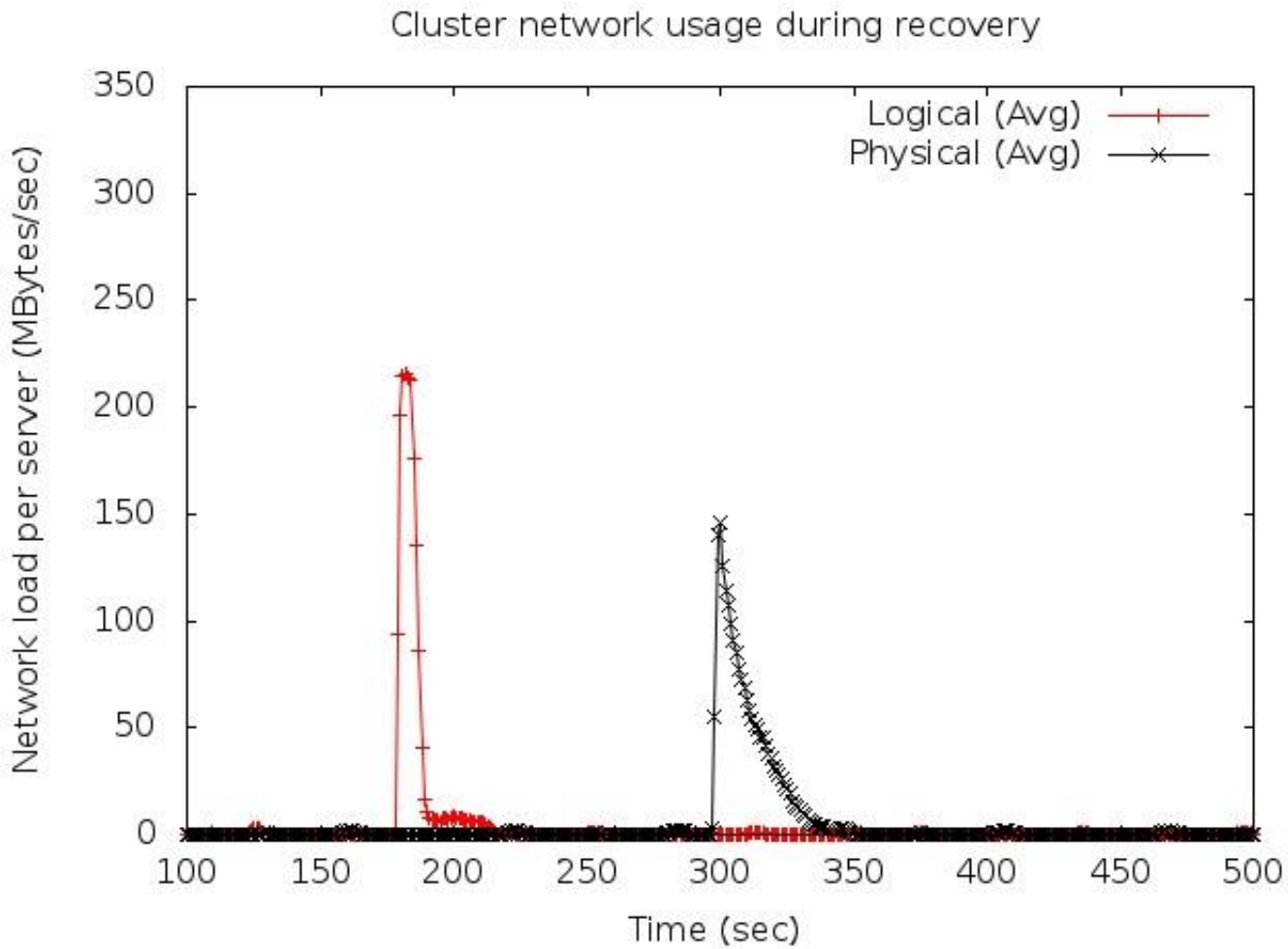


Recovery cost: Disk I/O

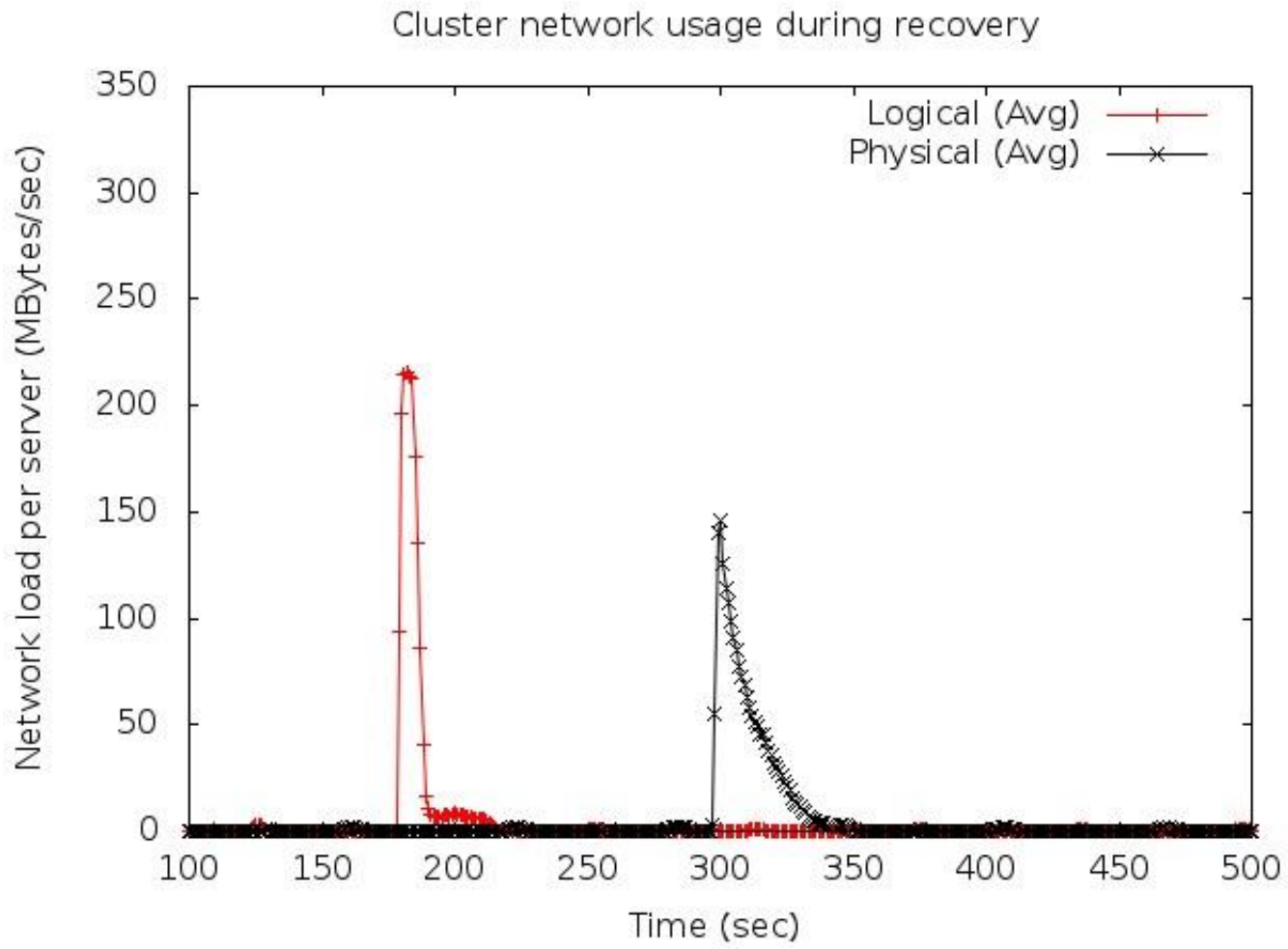


Area under the curves is same

Recovery cost: Network I/O



Recovery cost: Network I/O



Area under the curves is same

Other storage challenges

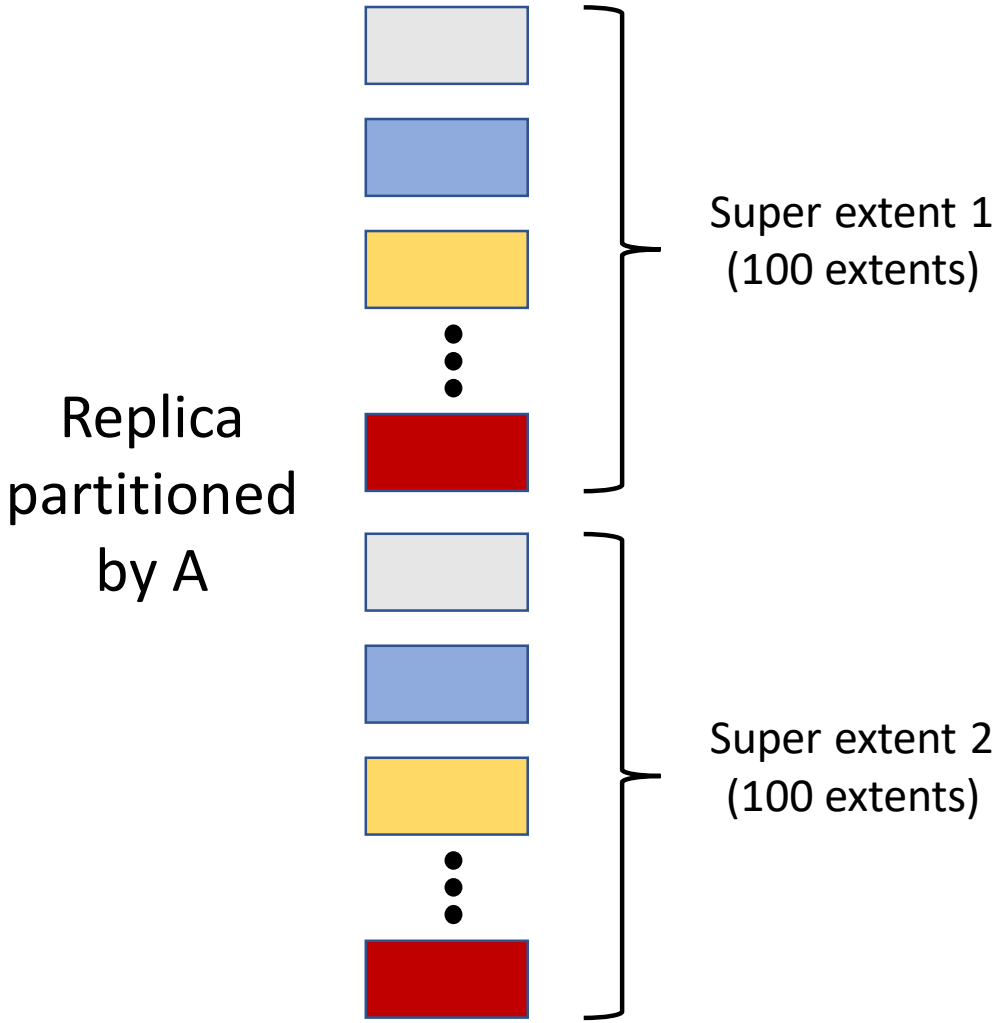
- Availability properties
- Fault isolation

Please refer to paper for details

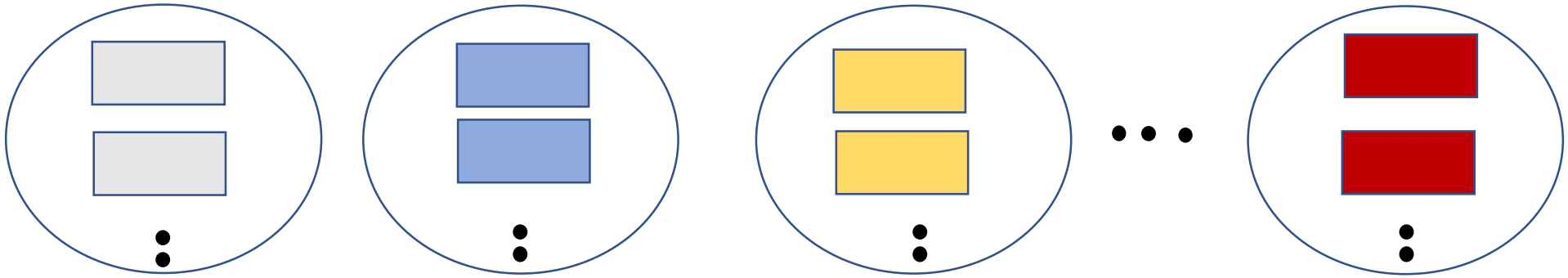
Outline

- *Introduction*
- Design & Evaluation
  1.) Key mechanism at storage layer
  2.) Efficient Query Execution
- Implementation
- Summary

Efficient Filter Queries



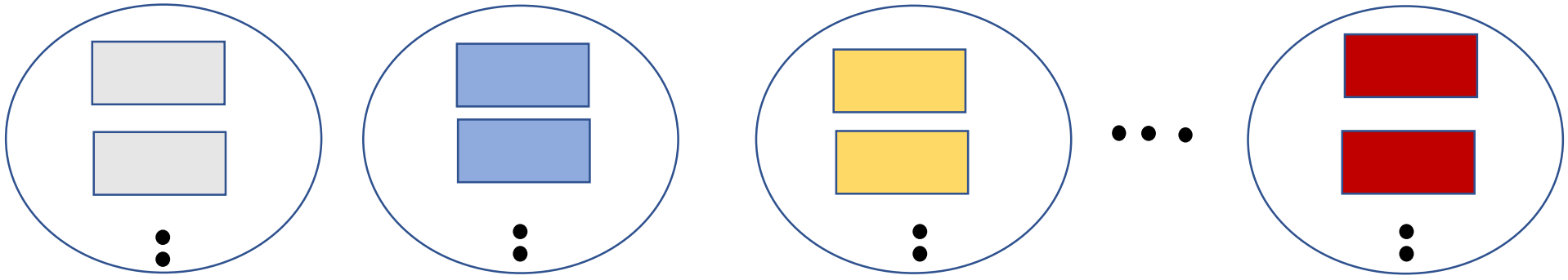
Efficient Filter Queries



Partition #1 Partition #2 Partition #3 Partition #100

Replica partitioned by A

Efficient Filter Queries

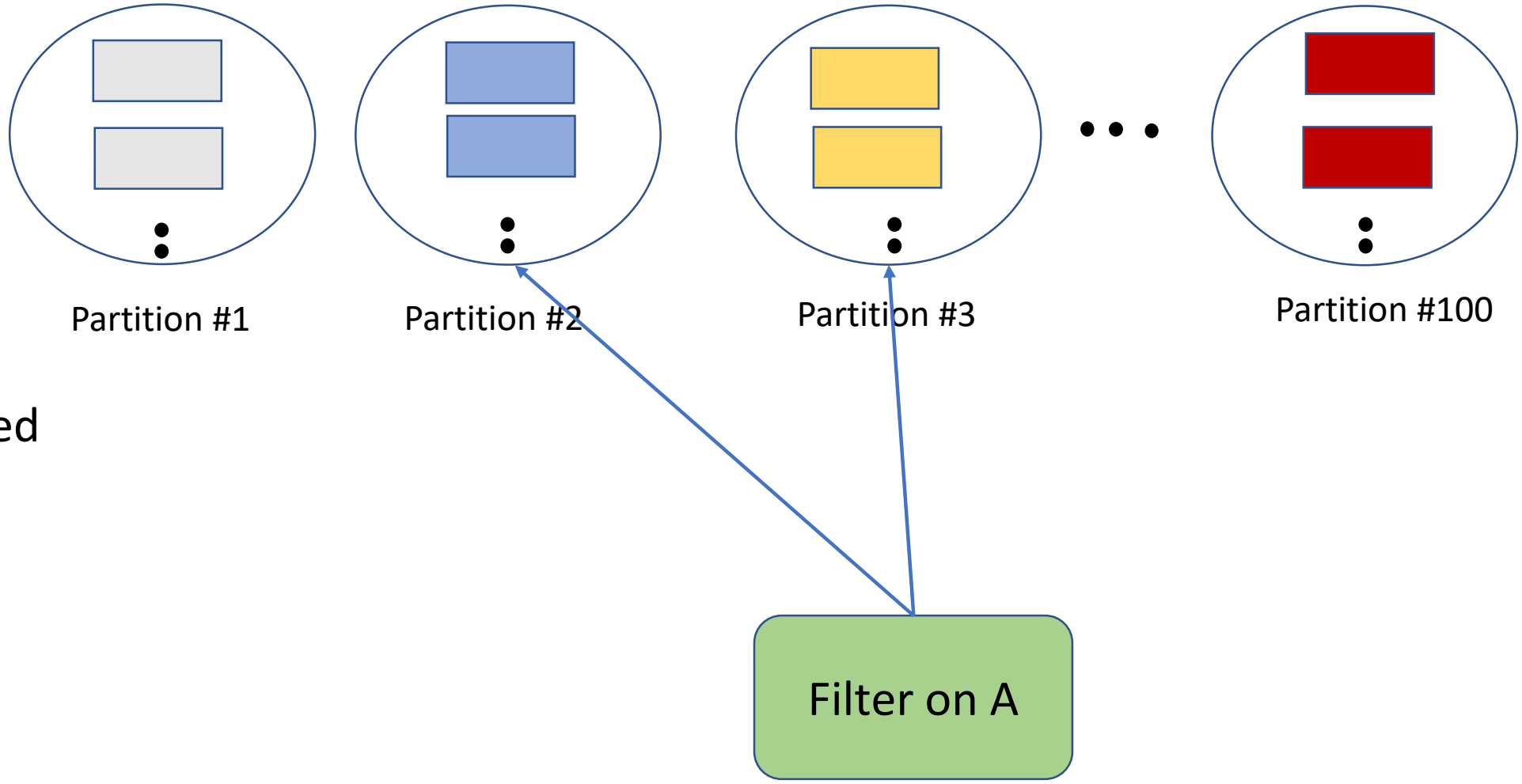


Partition #1 Partition #2 Partition #3 Partition #100

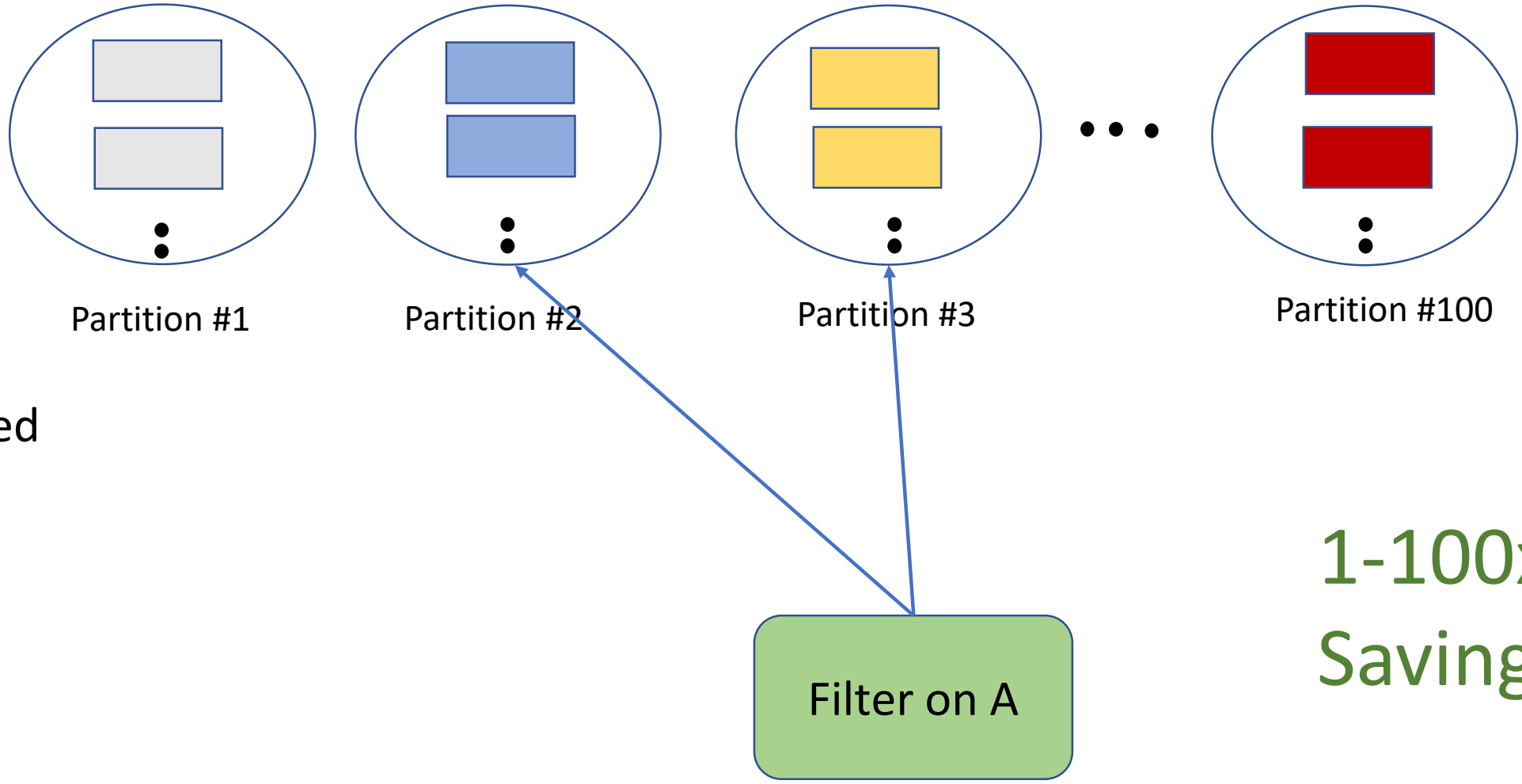
Replica partitioned by A

Filter on A

Efficient Filter Queries



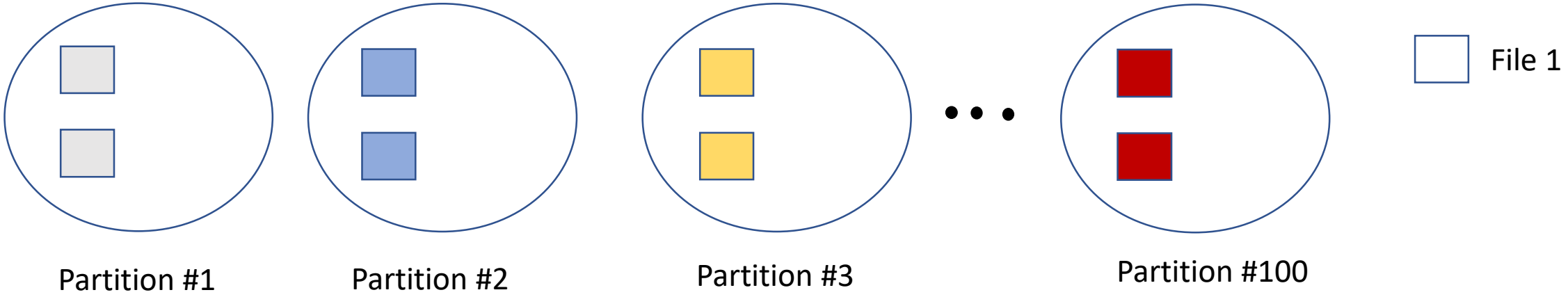
Efficient Filter Queries



1-100x Savings

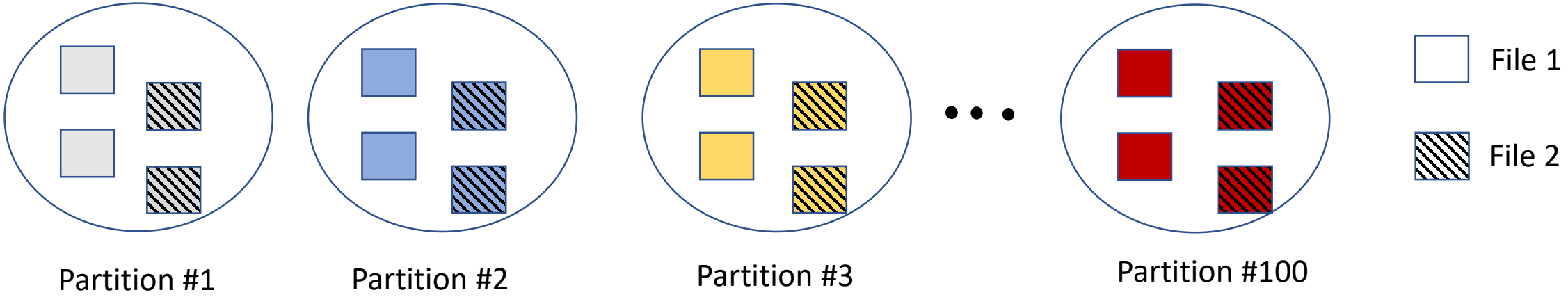
Join Queries: Heterogeneous co-location

- Rack level **co-location** of partitions across files



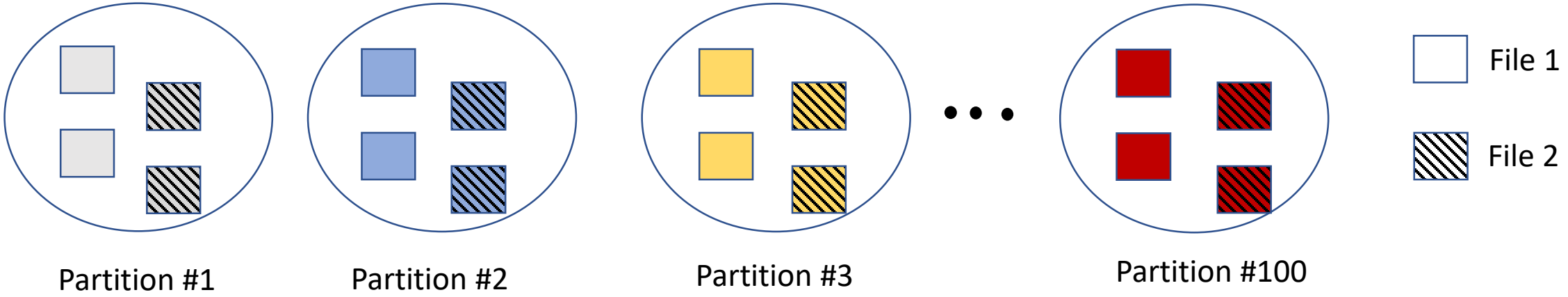
Join Queries: Heterogeneous co-location

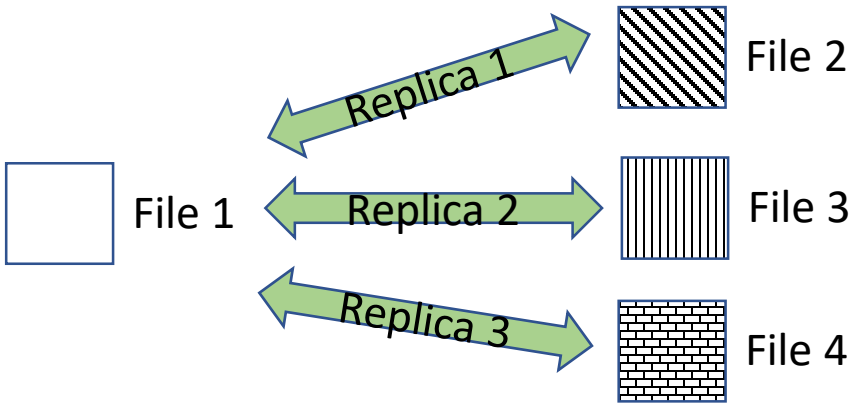
- Rack level **co-location** of partitions across files



Join Queries: Heterogeneous co-location

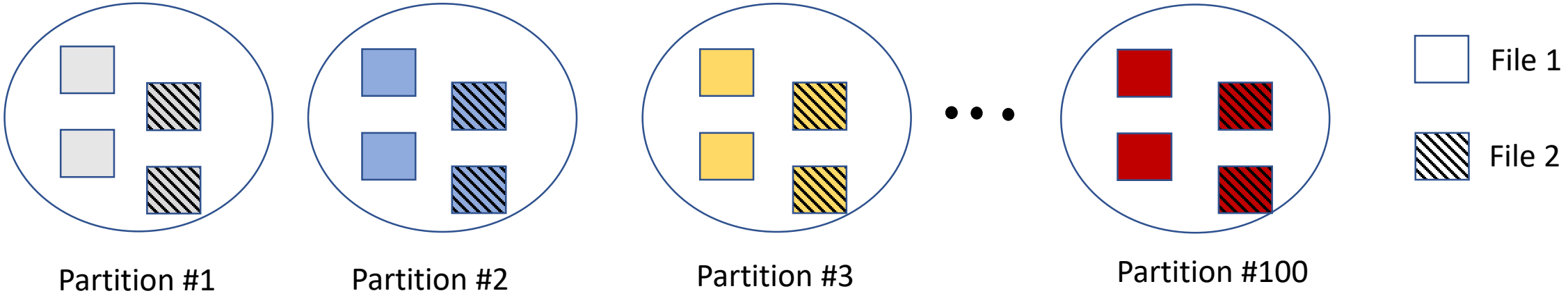
- Rack level **co-location** of partitions across files

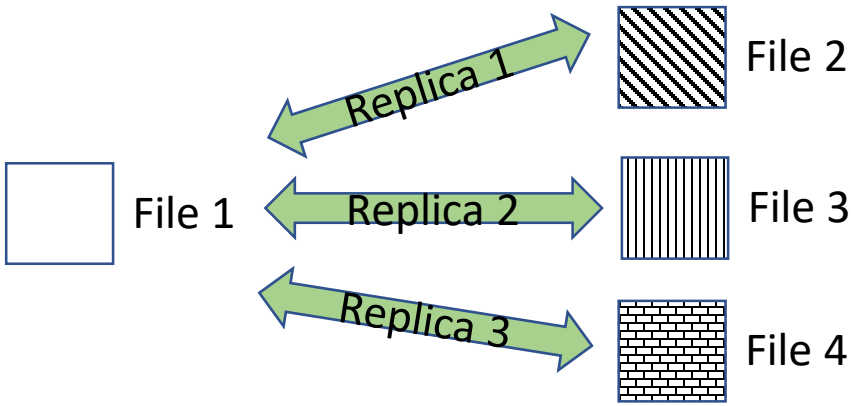




Join Queries: Heterogeneous co-location

- Rack level **co-location** of partitions across files

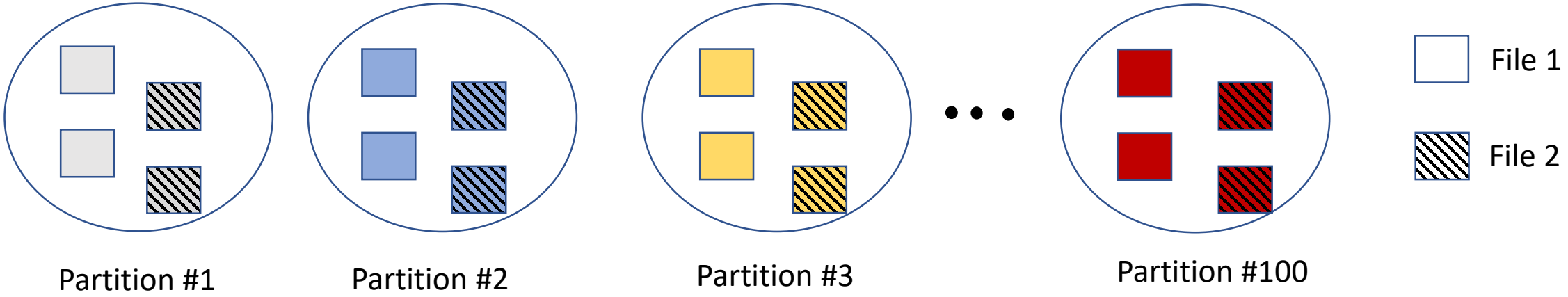




More queries get benefits of co-location

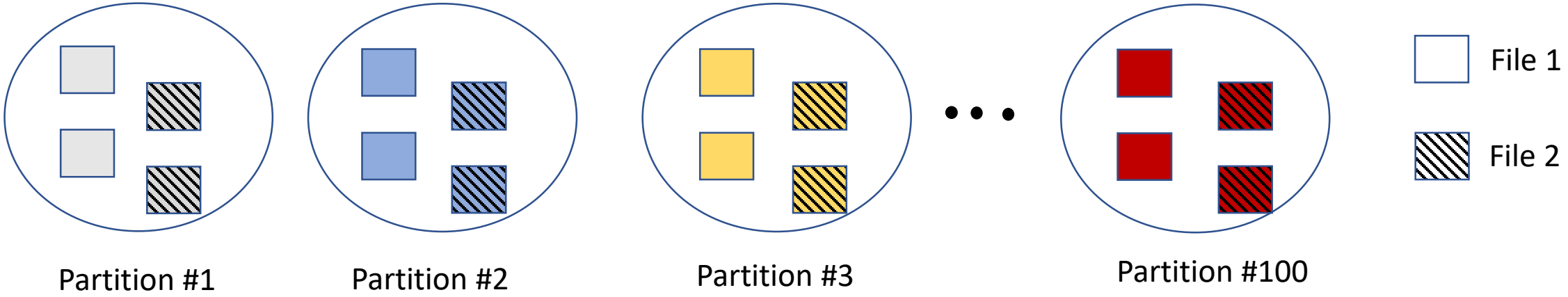
Efficient Join Queries: Sliced Reads

- *File 1* joined with *File 2* on *Column A*



Efficient Join Queries: Sliced Reads

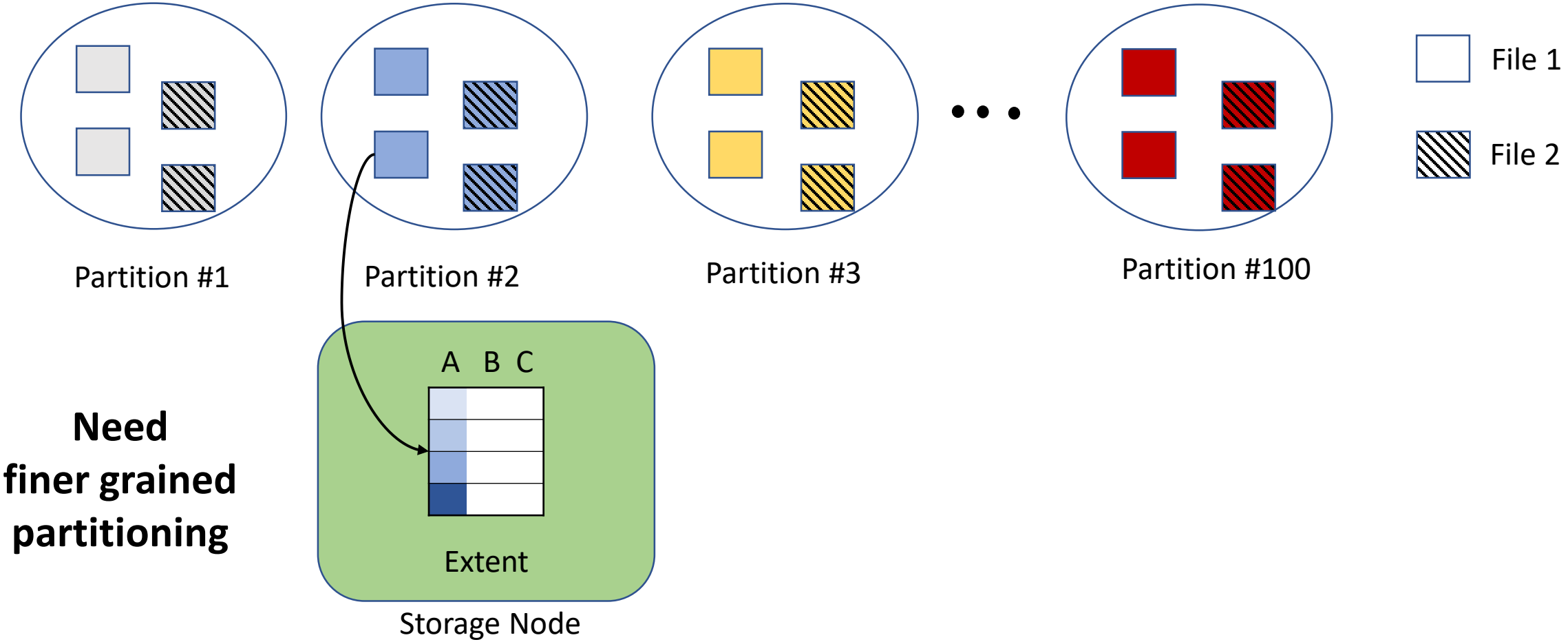
- *File 1* joined with *File 2* on *Column A*



Need finer grained partitioning

Efficient Join Queries: Sliced Reads

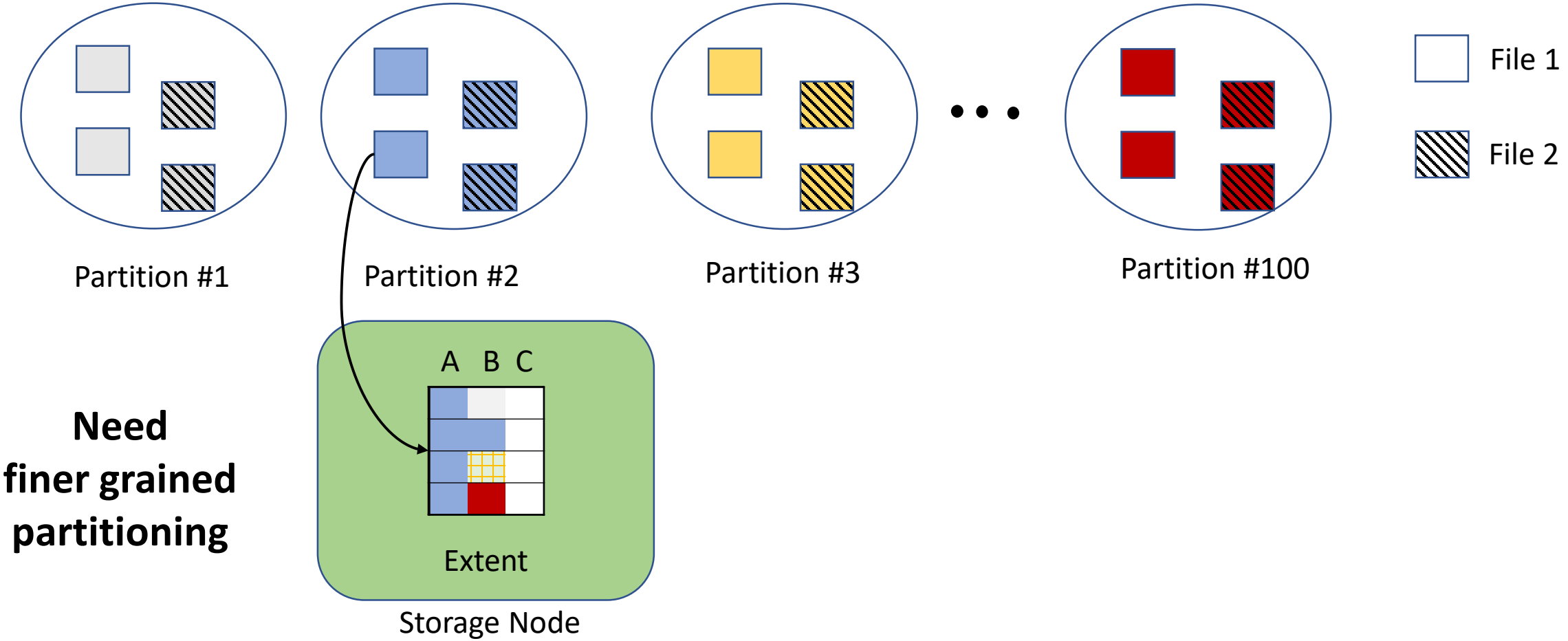
- *File 1* joined with *File 2* on *Column A*



Need finer grained partitioning

Efficient Join Queries: Sliced Reads

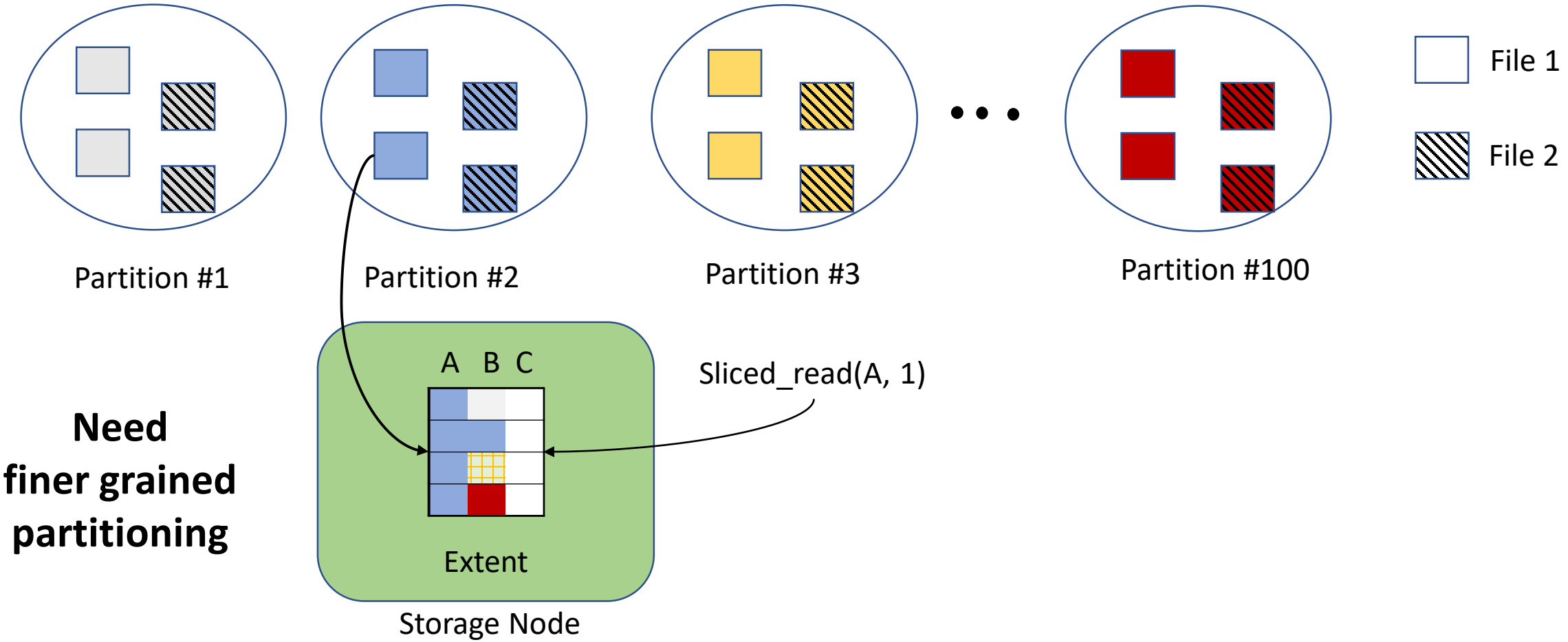
- *File 1* joined with *File 2* on *Column A*



Need finer grained partitioning

Efficient Join Queries: Sliced Reads

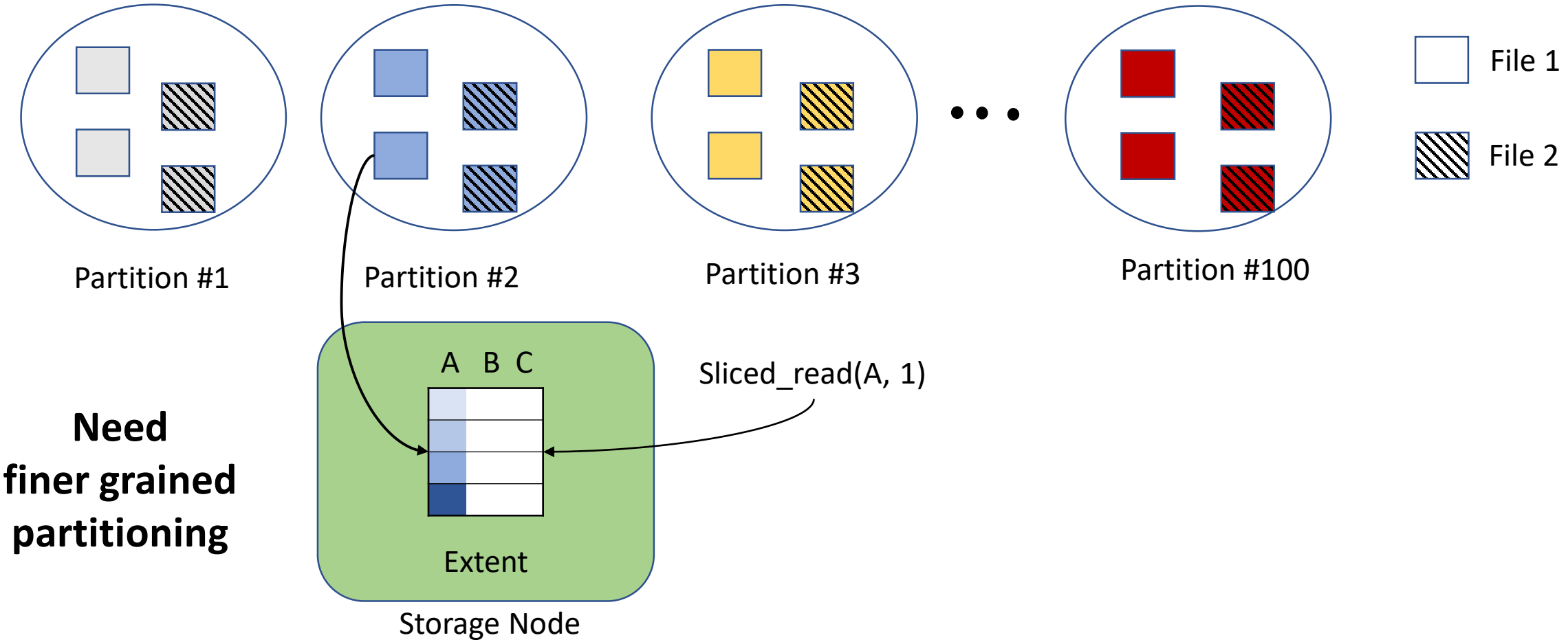
- *File 1* joined with *File 2* on *Column A*



Need finer grained partitioning

Efficient Join Queries: Sliced Reads

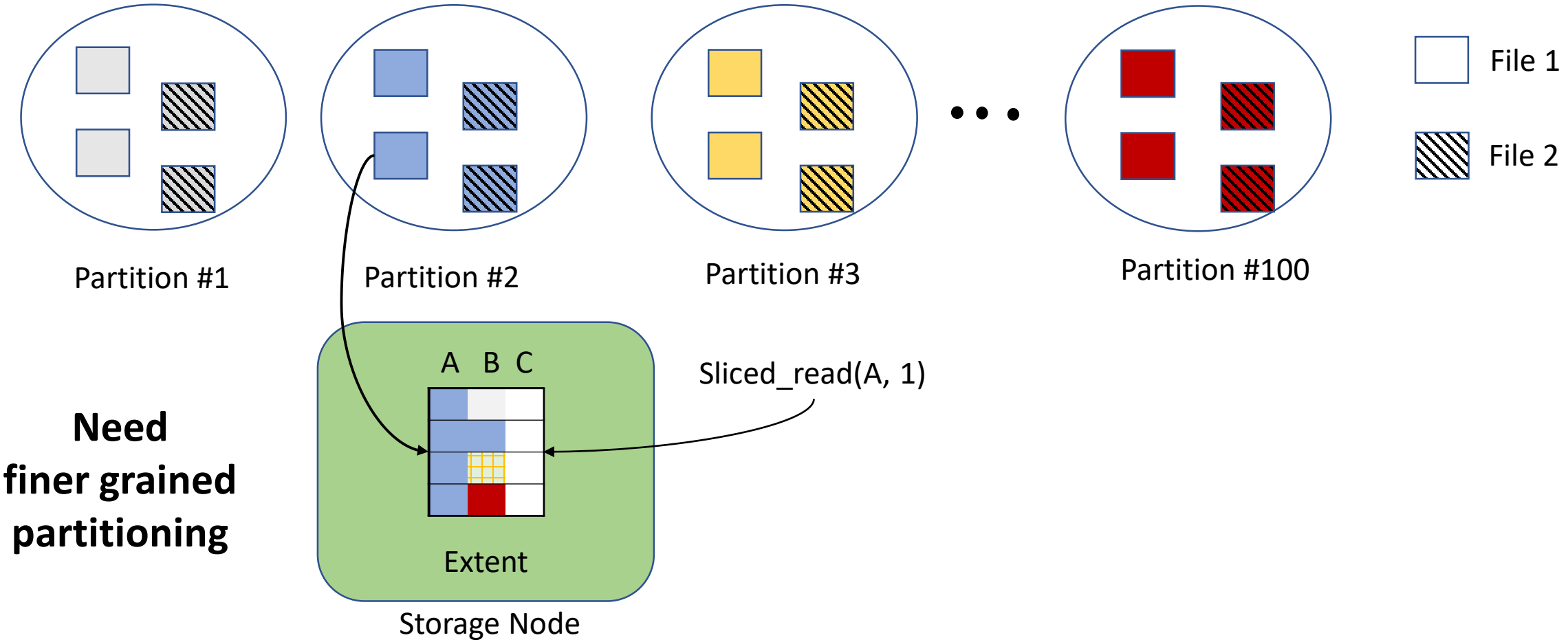
- *File 1* joined with *File 2* on *Column A*



Need finer grained partitioning

Efficient Join Queries: Sliced Reads

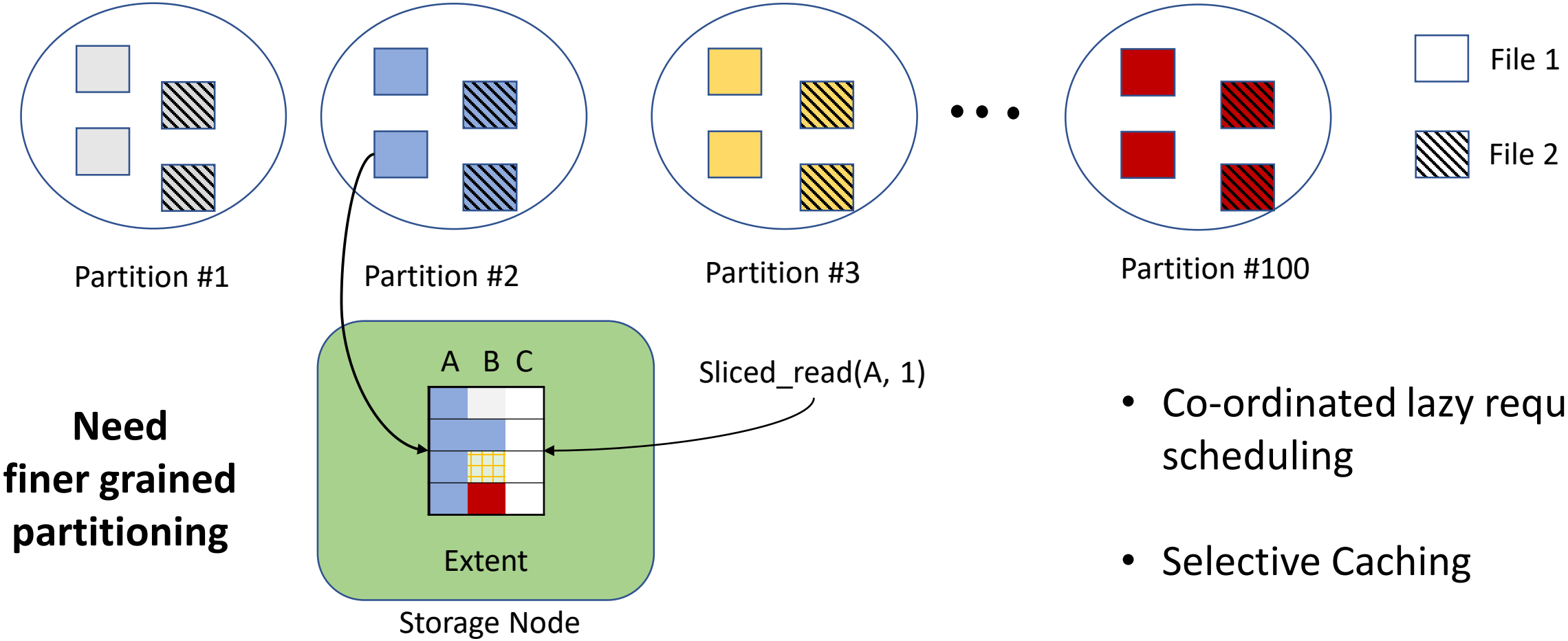
- *File 1* joined with *File 2* on *Column A*



Need finer grained partitioning

Efficient Join Queries: Sliced Reads

- *File 1* joined with *File 2* on *Column A*

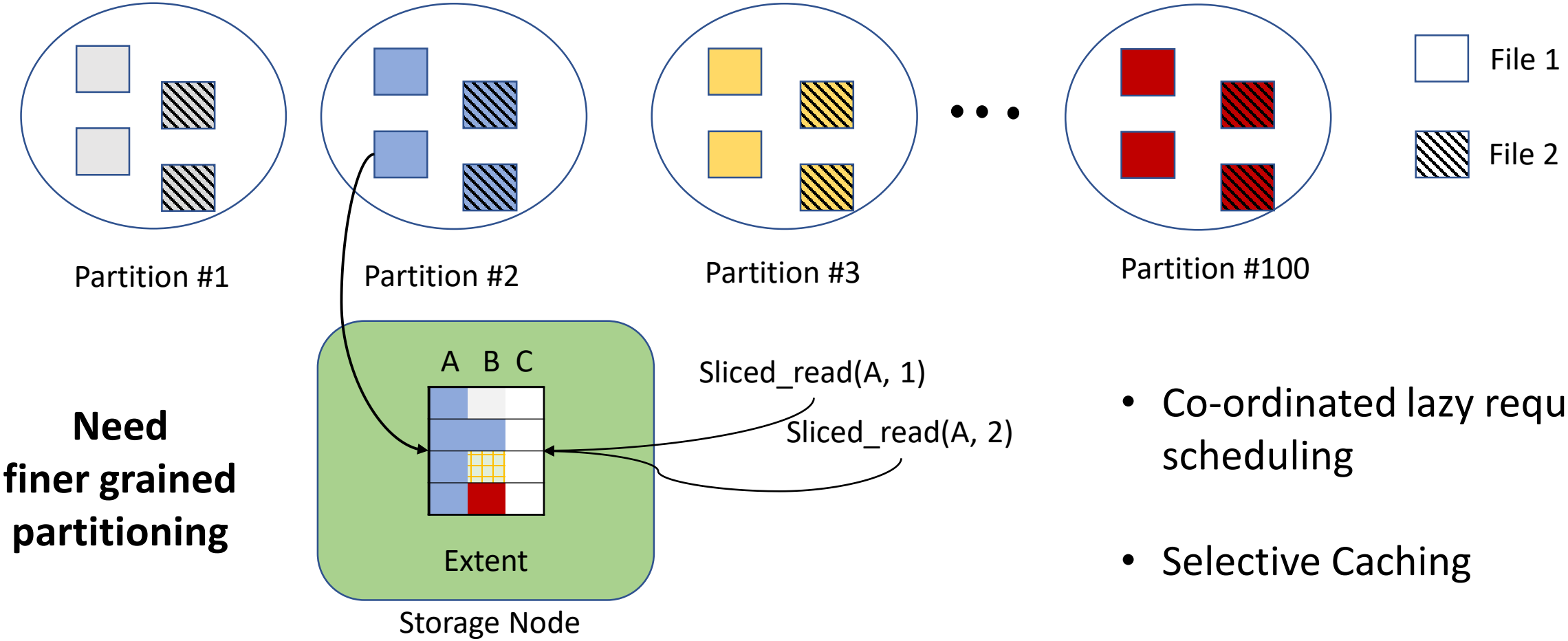


Need finer grained partitioning

- Co-ordinated lazy request scheduling
- Selective Caching

Efficient Join Queries: Sliced Reads

- *File 1* joined with *File 2* on *Column A*

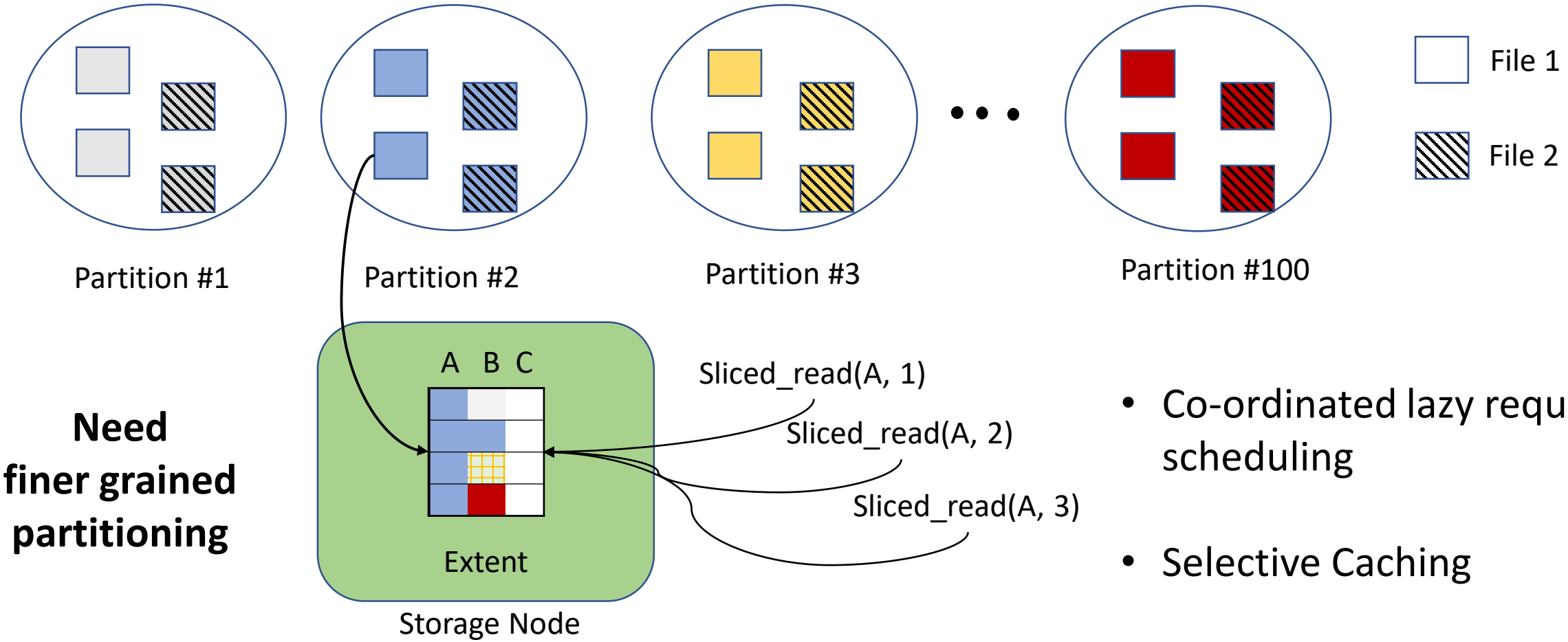


Need finer grained partitioning

- Co-ordinated lazy request scheduling
- Selective Caching

Efficient Join Queries: Sliced Reads

- *File 1* joined with *File 2* on *Column A*

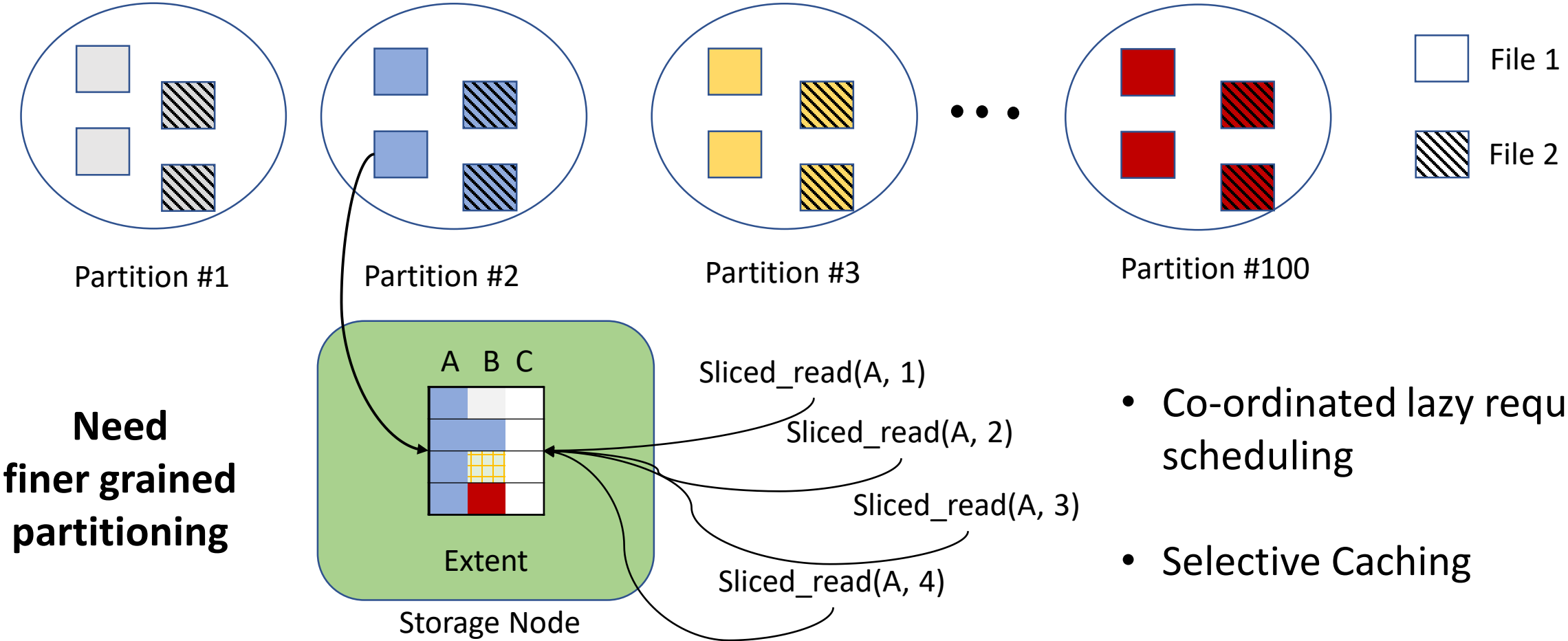


Need finer grained partitioning

- Co-ordinated lazy request scheduling
- Selective Caching

Efficient Join Queries: Sliced Reads

- *File 1* joined with *File 2* on *Column A*

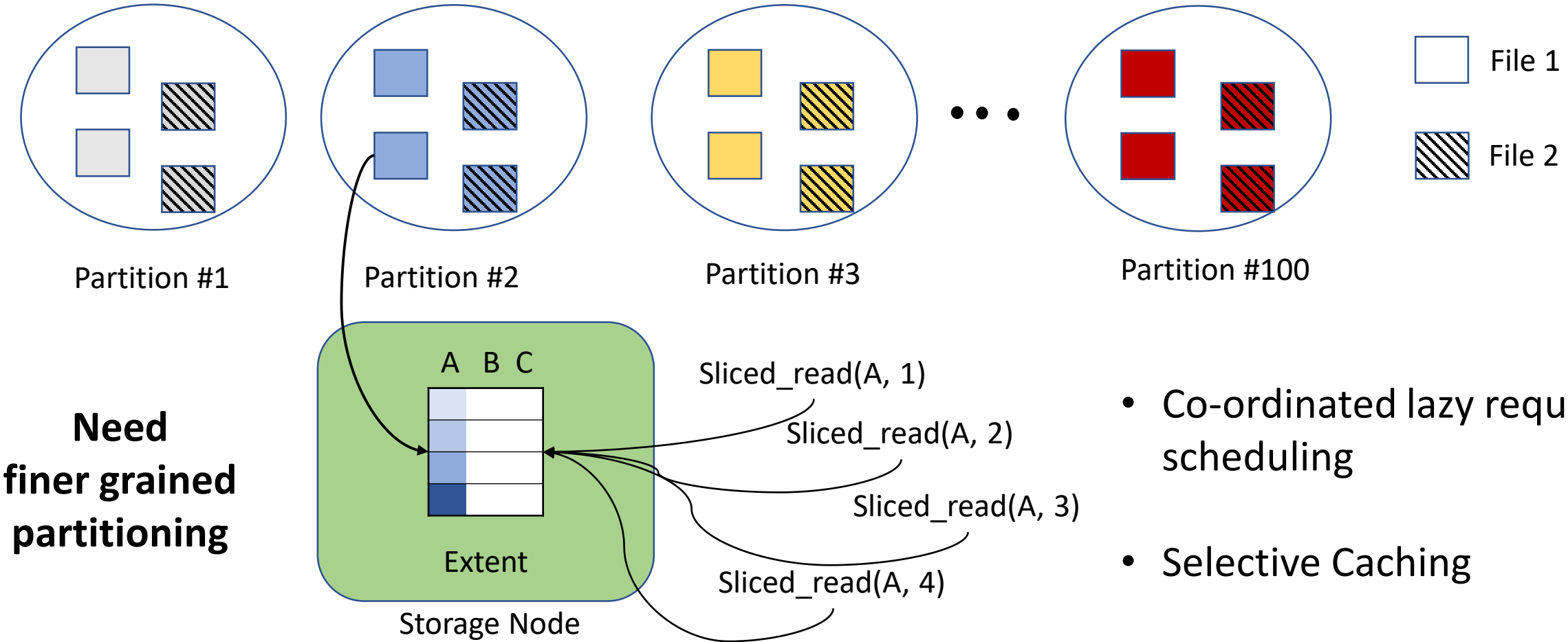


Need finer grained partitioning

- Co-ordinated lazy request scheduling
- Selective Caching

Efficient Join Queries: Sliced Reads

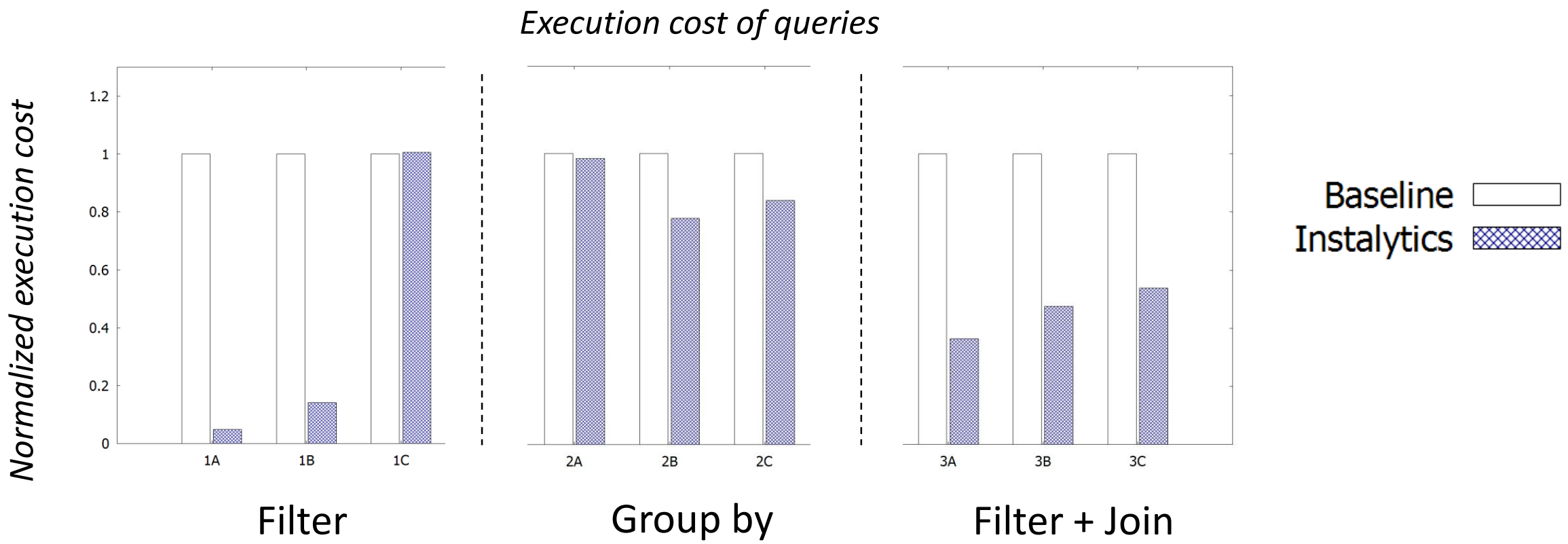
- *File 1* joined with *File 2* on *Column A*



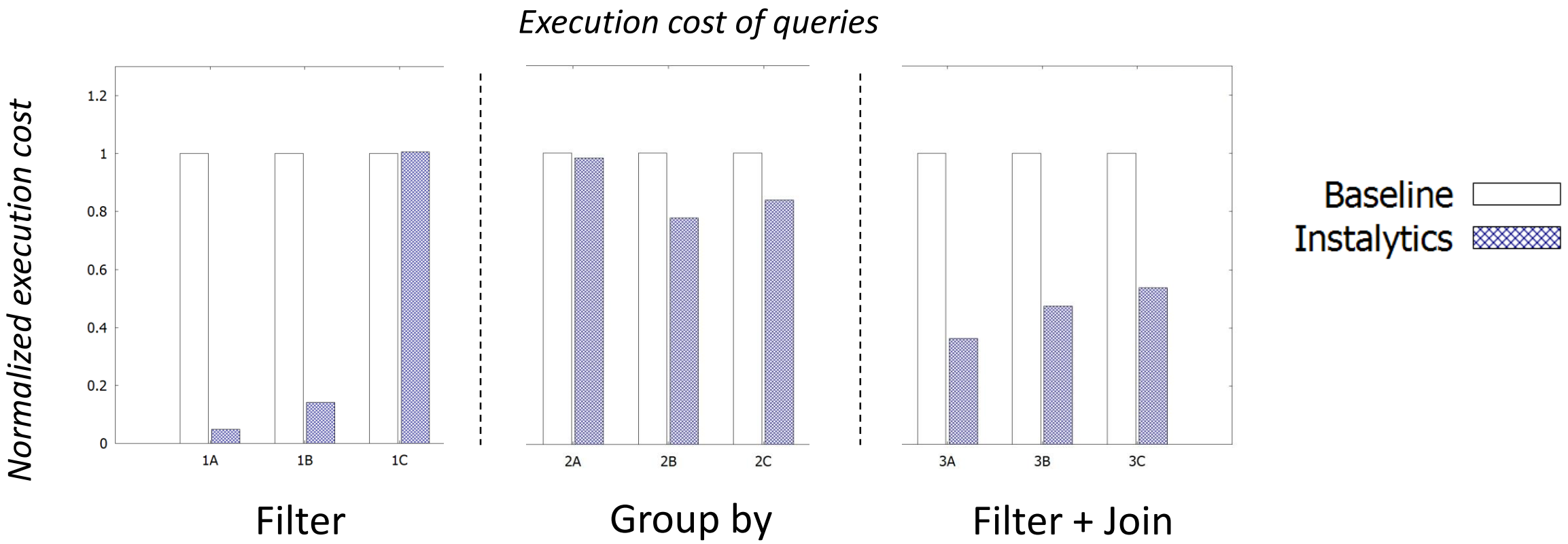
Need finer grained partitioning

- Co-ordinated lazy request scheduling
- Selective Caching

AMPLab Big Data Benchmark



AMPLab Big Data Benchmark



Simultaneous benefits on multiple columns

Production Queries

- Slice of production telemetry analytics workload

Description	Q1	Q2	Q3	Q4	Q5	Q6
$Baseline_{cost}$	251	414	22	20	398	242
$INSTalytics_{cost}$	59	206	0.3	1.1	403	239
$Baseline_{latency}$	39	66	23	4	50	20
$INSTalytics_{latency}$	7	21	1.4	2.3	51	20

- Costs are in *compute hours*
- Latencies are in *minutes*

Production Queries

- Slice of production telemetry analytics workload

Description	Q1	Q2	Q3	Q4	Q5	Q6
$Baseline_{cost}$	251	414	22	20	398	242
$INSTalytics_{cost}$	59	206	0.3	1.1	403	239
$Baseline_{latency}$	39	66	23	4	50	20
$INSTalytics_{latency}$	7	21	1.4	2.3	51	20

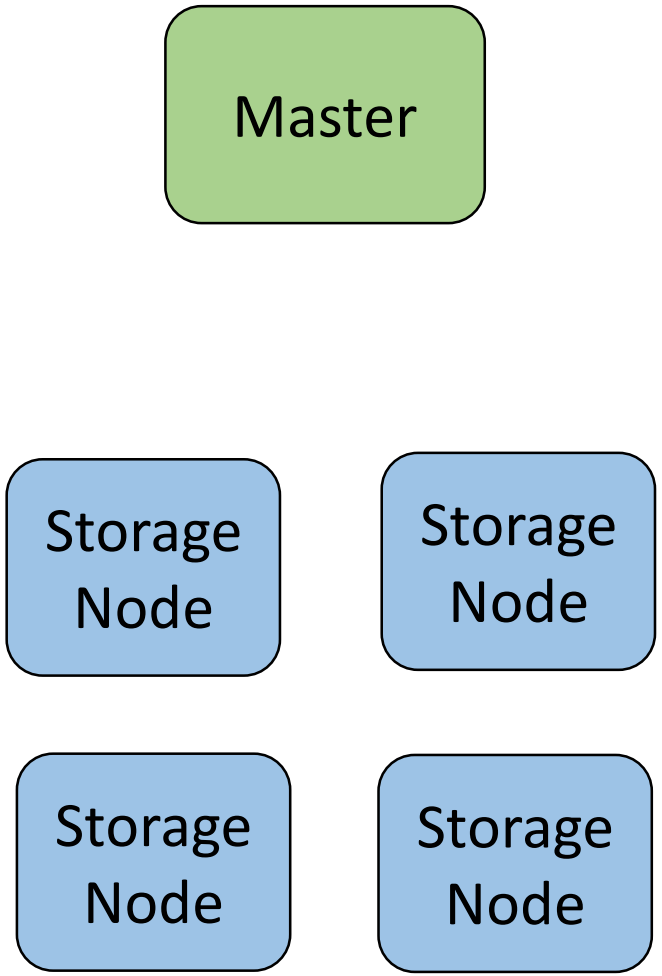
- Costs are in *compute hours*
- Latencies are in *minutes*

Outline

- *Introduction*
- Design & Evaluation
  1.) Key mechanism at storage layer
  2.) Efficient Query Execution
- Implementation
- Summary

Implementation

1.) Create Path



2.) Recovery Path

Implementation

1.) Create Path

Logically_replicate(file, **adapter**)

Master

2.) Recovery Path

CSV

Storage Node

Storage Node

Storage Node

Storage Node

Implementation

1.) Create Path

Logically_replicate(file, **adapter**)

Master

2.) Recovery Path

Recover_extent(super-extent info)

CSV

{JSON}

Parquet

Storage Node

Storage Node

Storage Node

Storage Node

Storage Node

Summary

- INSTalytics: Compute-aware cluster filesystem
  - Logical replication: Amplifies benefits of partitioning
  - Efficient processing of join queries
    - Heterogeneous co-location
    - Sliced Reads
  - Significant performance benefits
  - Recovery properties not compromised
- **Co-design of Compute & Storage layers for efficient analytics at scale**

Thank you

Questions?