Enhancing Throughput of

NCA 2017

Zhongmiao Li, Peter Van Roy and Paolo Romano

UCL Université catholique de Louvain

inesc id lisboa

TÉCNICO LISBOA

Enhancing Throughput of Partially Replicated State Machines via

NCA 2017

Zhongmiao Li, Peter Van Roy and Paolo Romano

UCL Université catholique de Louvain

inesc id lisboa

TÉCNICO LISBOA

Enhancing Throughput of Partially Replicated State Machines via Multi-Partition Operation Scheduling

NCA 2017

Zhongmiao Li, Peter Van Roy and Paolo Romano

UCL Université catholique de Louvain

inesc id lisboa

TÉCNICO LISBOA

Background

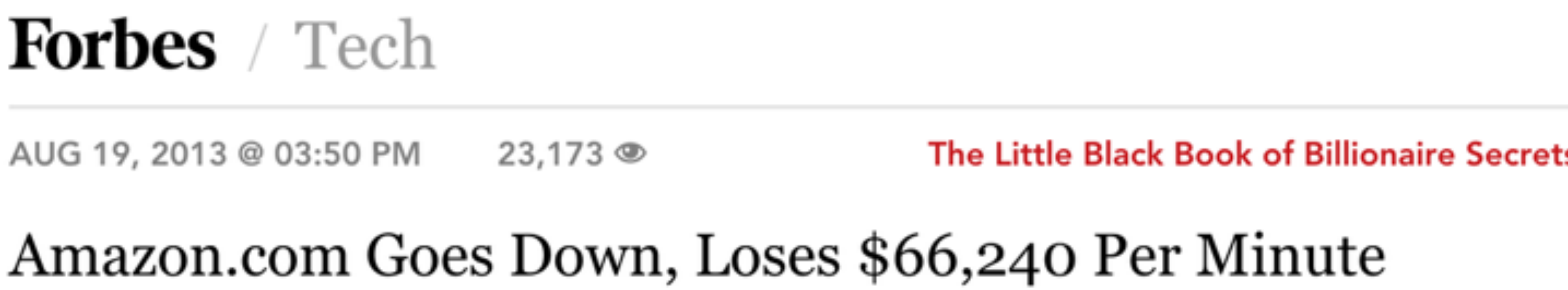
- Online services strive to have 7*24 availability.

- Replication is crucial to ensure availability.

- State-machine replication (SMR) is a key technique to implement fault-tolerant services.

Background

- Online services strive to have 7*24 availability.



Forbes / Tech

AUG 19, 2013 @ 03:50 PM 23,173

The Little Black Book of Billionaire Secrets

Amazon.com Goes Down, Loses $66,240 Per Minute

- Replication is crucial to ensure availability.
- State-machine replication (SMR) is a key technique to implement fault-tolerant services.

Background

State-machine replication

- Applications are abstracted as ‘deterministic state machines’
  - All replicas store application state
  - Replicas agree on operation order (e.g. using *Paxos*), then execute
  - Deterministic operation => equivalent final state of replicas

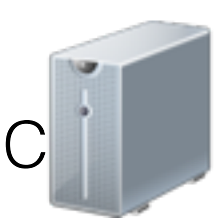
A, B, C

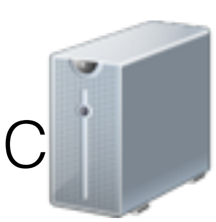
Background

State-machine replication

- Applications are abstracted as ‘deterministic state machines’
  - All replicas store application state
  - Replicas agree on operation order (e.g. using *Paxos*), then execute
  - Deterministic operation => equivalent final state of replicas

A, B, C 

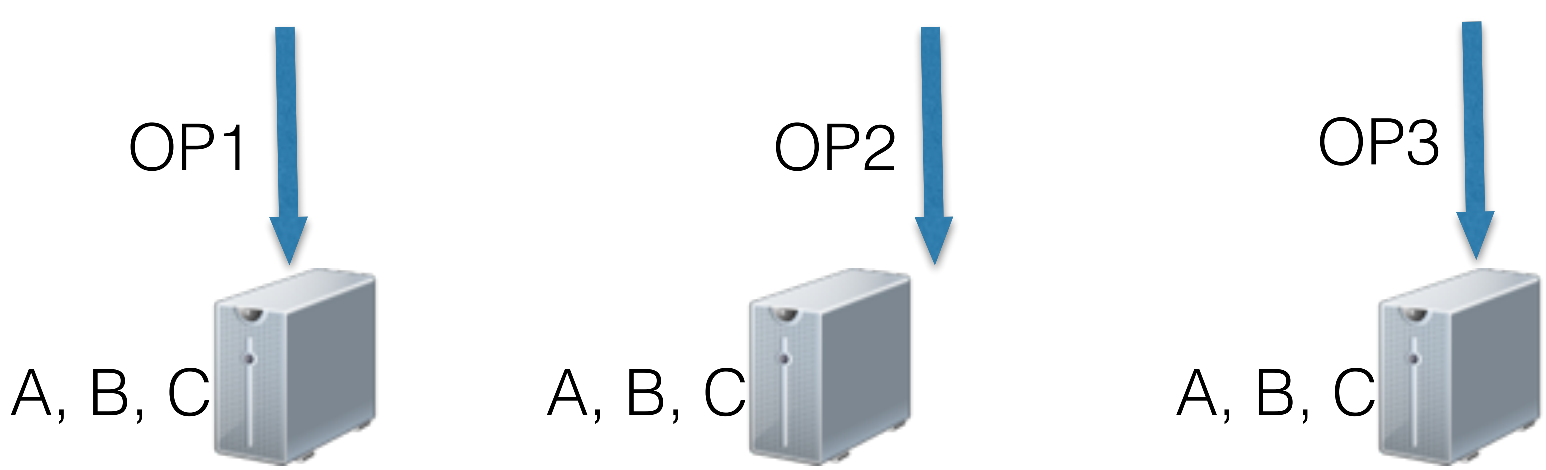
A, B, C 

A, B, C 

Background

State-machine replication

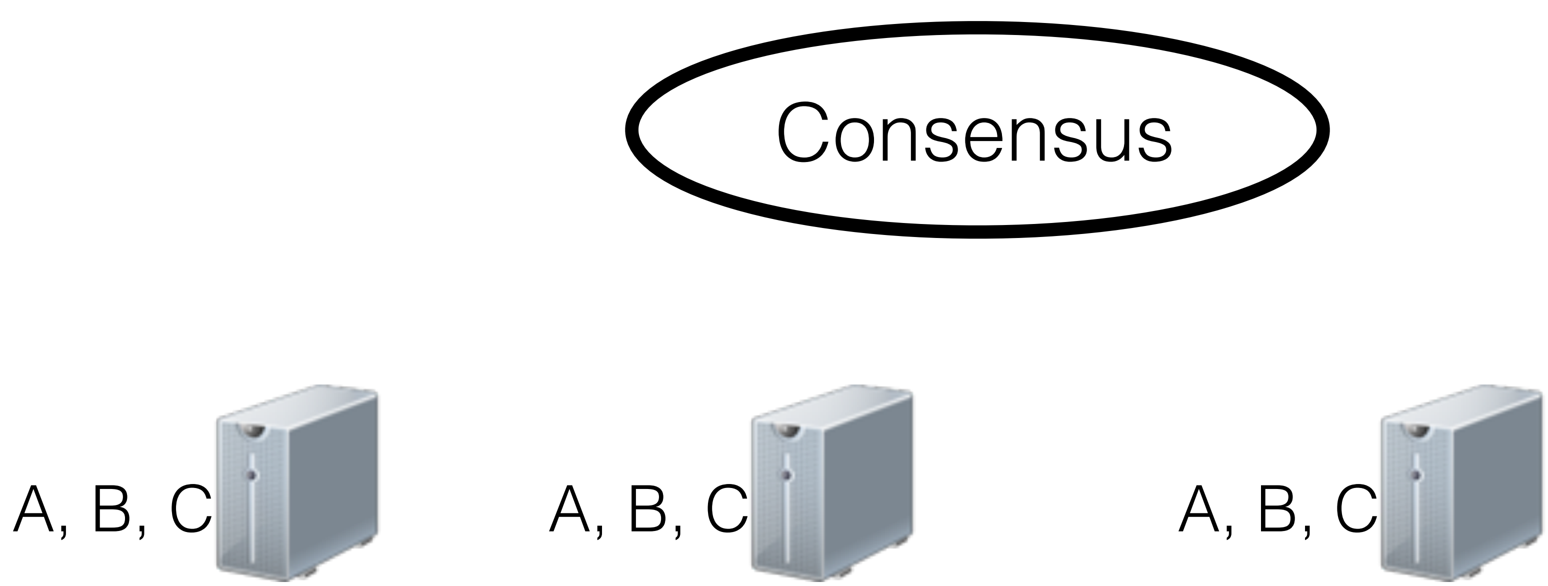
- Applications are abstracted as 'deterministic state machines'
  - All replicas store application state
  - Replicas agree on operation order (e.g. using *Paxos*), then execute
  - Deterministic operation => equivalent final state of replicas



Background

State-machine replication

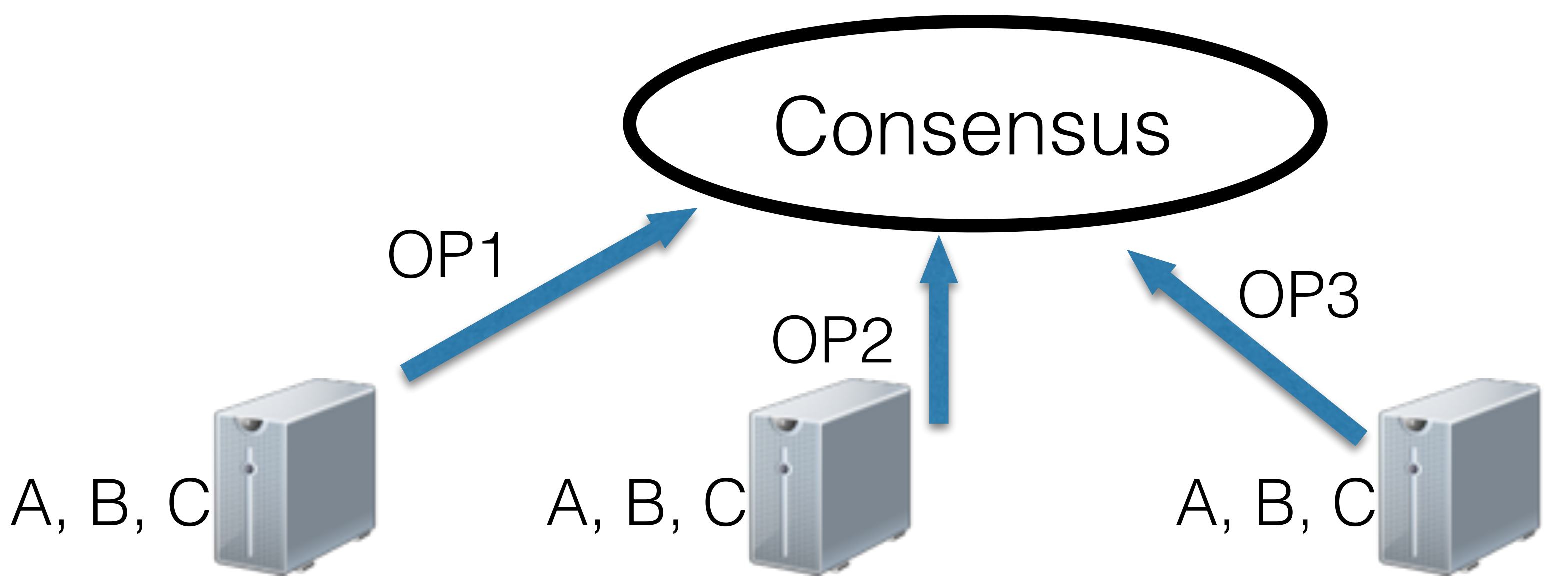
- Applications are abstracted as ‘deterministic state machines’
  - All replicas store application state
  - Replicas agree on operation order (e.g. using *Paxos*), then execute
  - Deterministic operation => equivalent final state of replicas



Background

State-machine replication

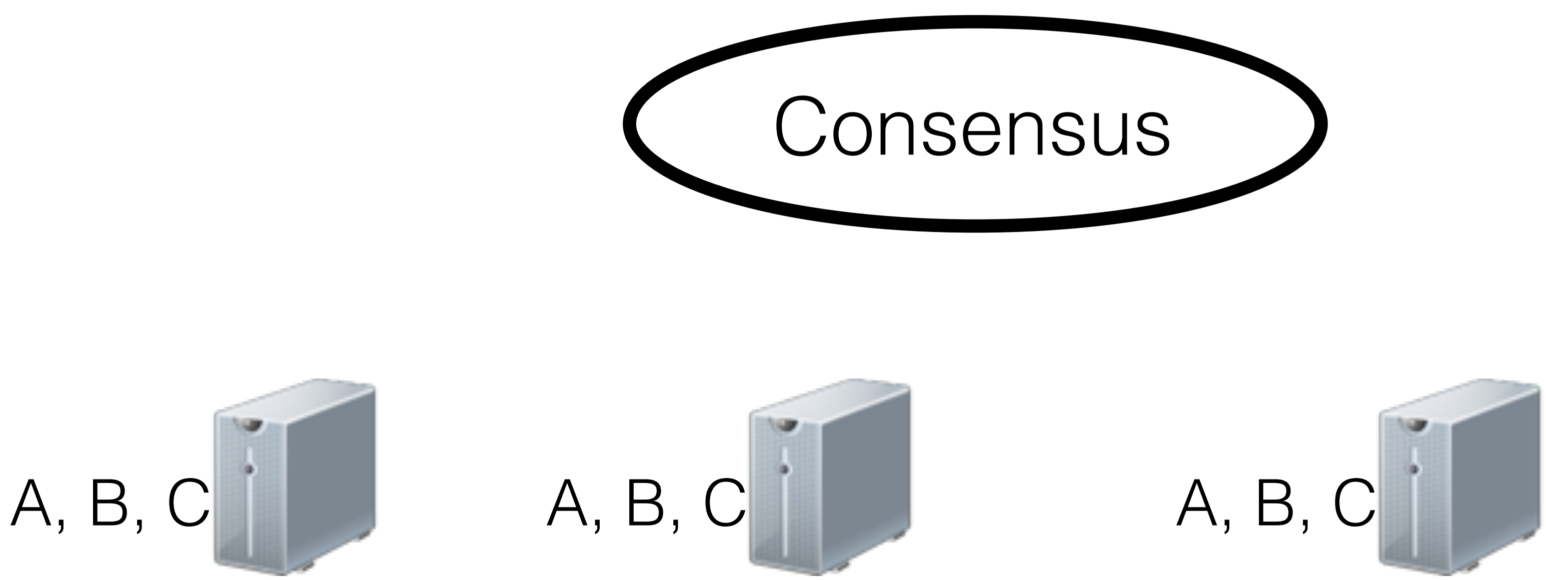
- Applications are abstracted as ‘deterministic state machines’
  - All replicas store application state
  - Replicas agree on operation order (e.g. using *Paxos*), then execute
  - Deterministic operation => equivalent final state of replicas



Background

State-machine replication

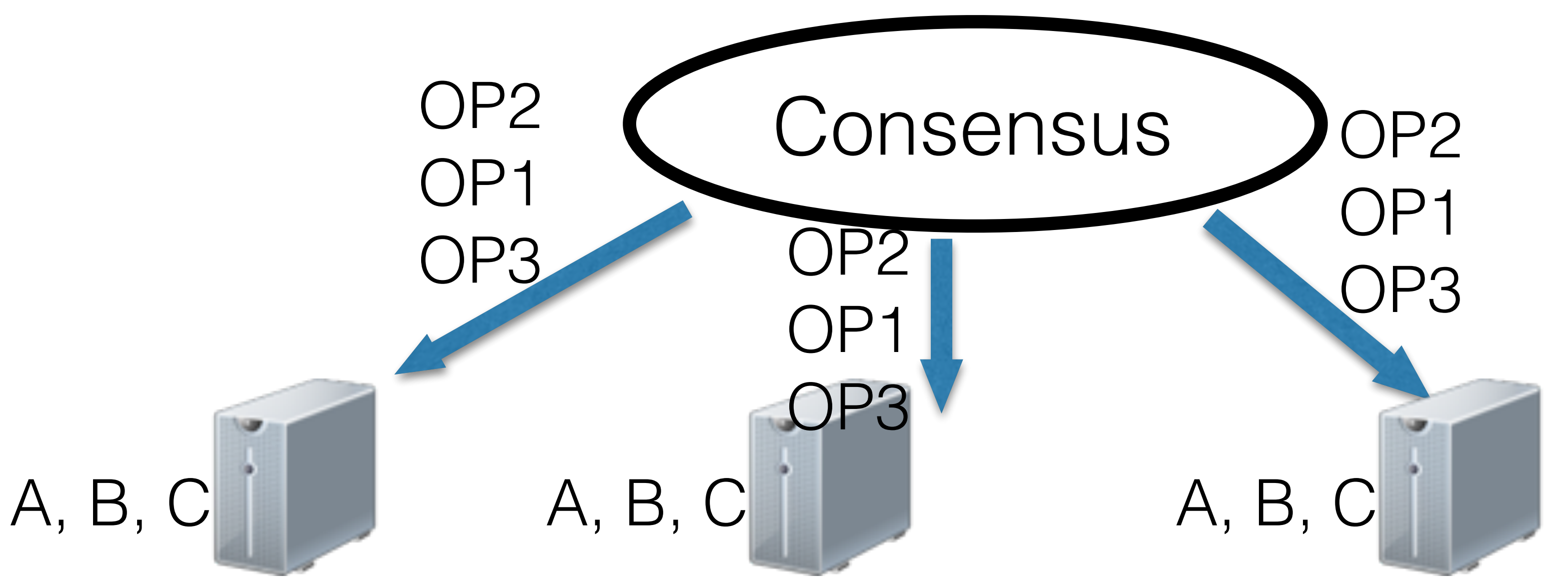
- Applications are abstracted as ‘deterministic state machines’
  - All replicas store application state
  - Replicas agree on operation order (e.g. using *Paxos*), then execute
  - Deterministic operation => equivalent final state of replicas



Background

State-machine replication

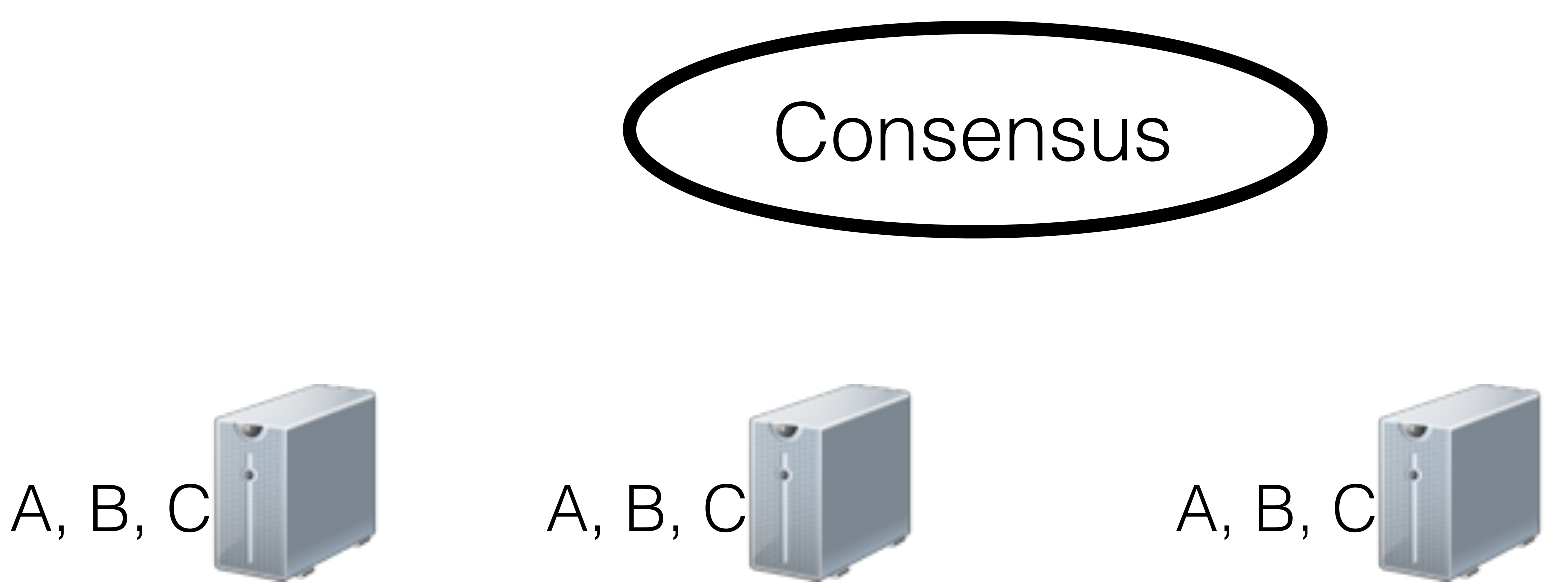
- Applications are abstracted as ‘deterministic state machines’
  - All replicas store application state
  - Replicas agree on operation order (e.g. using *Paxos*), then execute
  - Deterministic operation => equivalent final state of replicas



Background

State-machine replication

- Applications are abstracted as ‘deterministic state machines’
  - All replicas store application state
  - Replicas agree on operation order (e.g. using *Paxos*), then execute
  - Deterministic operation => equivalent final state of replicas



Background

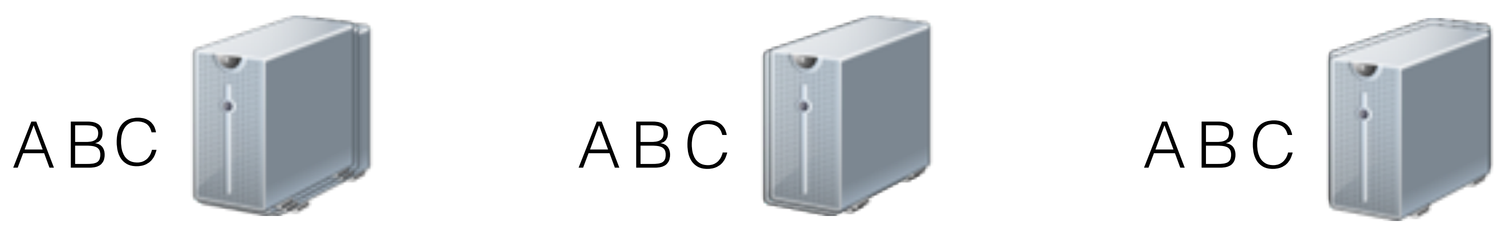
Partially-replicated state machines(i)

- The classical SMR does not scale
  - Replicas store full state & execute all update ops

    => throughput limited by single replica’s capacity & speed!
- Recent work propose to *partially-replicate* state machines to enhance scalability
  - “High performance state- machine replication”, DSN’11
  - “Calvin: fast distributed transactions for partitioned database systems”, SIGMOD’12
  - “Scalable state-machine replication”, DSN’14

Background

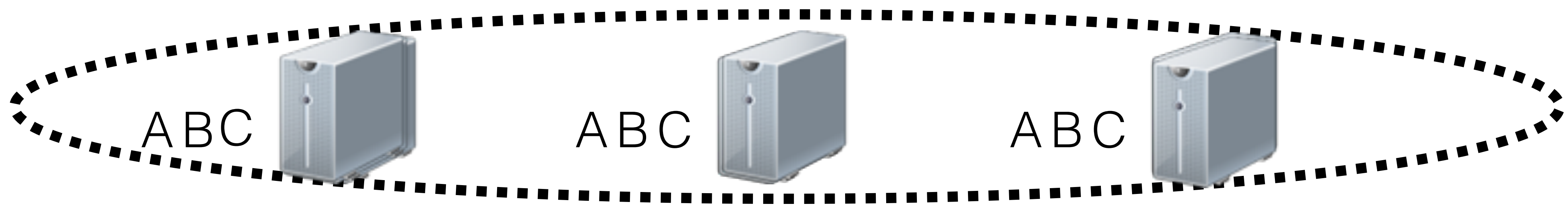
Partially-replicated state machines(ii)



Background

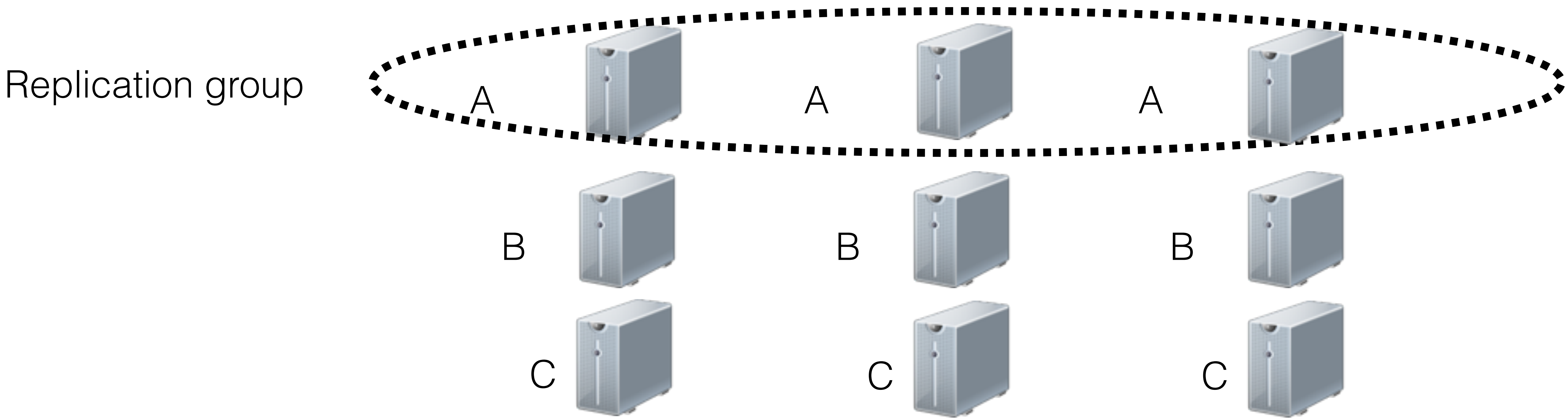
Partially-replicated state machines(ii)

Replication group



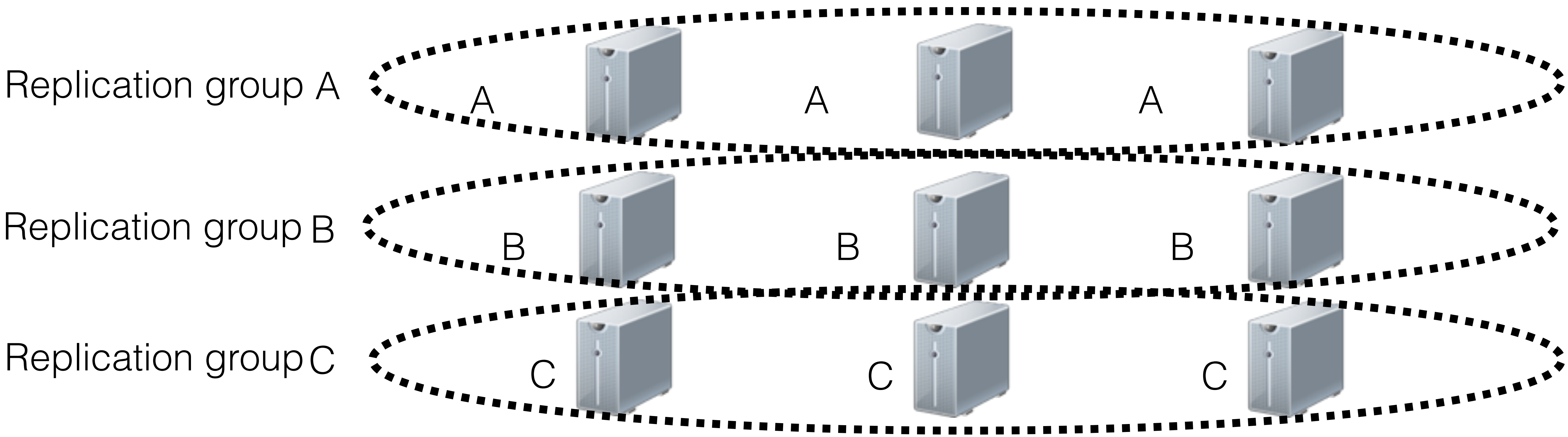
Background

Partially-replicated state machines(ii)



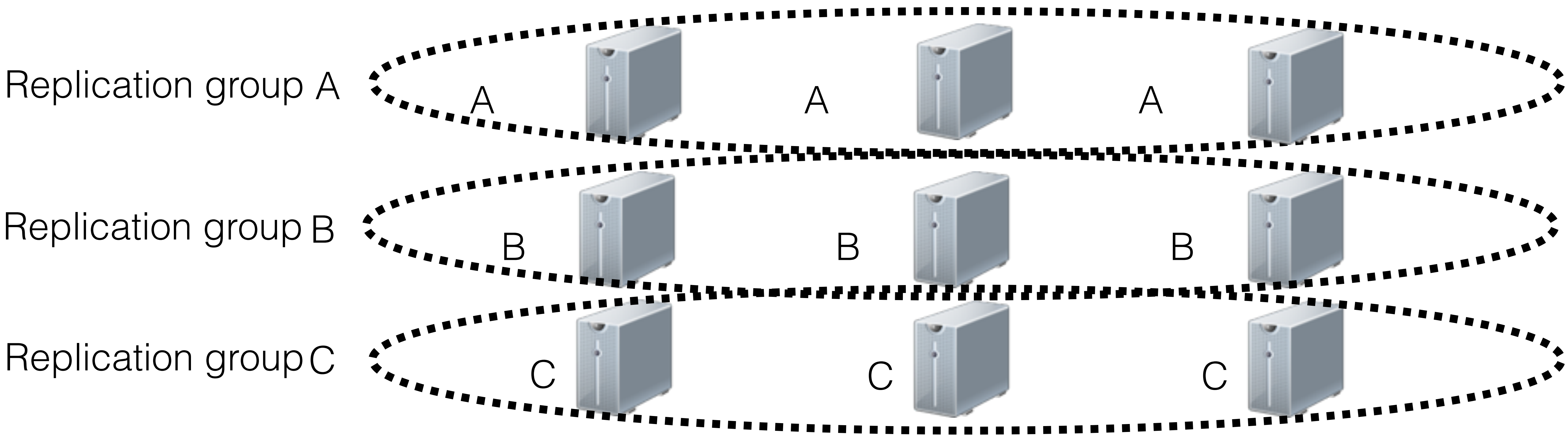
Background

Partially-replicated state machines(ii)



Background

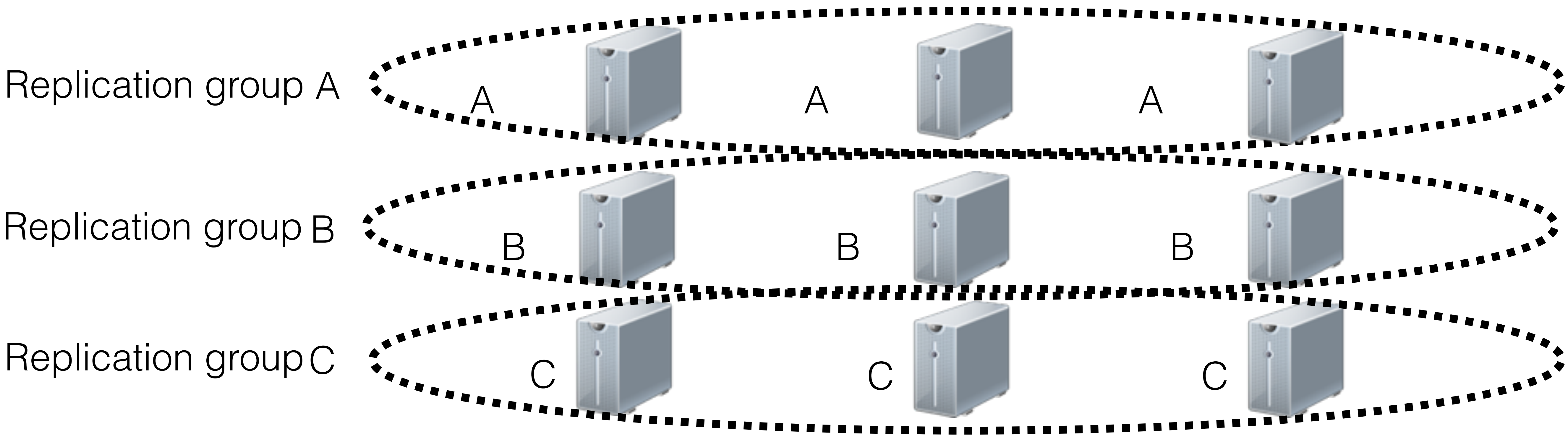
Partially-replicated state machines(ii)



- Each replica splits their state to multiple partitions
- Ops involving single partition (SPOs) only executed by that partition
- Ops involving multiple partitions (MPOs) coordinated and then executed by involved partitions

Background

Partially-replicated state machines(ii)



- Each replica splits their state to multiple partitions
- Ops involving single partition (SPOs) only executed by that partition
- Ops involving multiple partitions (MPOs) coordinated and then executed by involved partitions
- But.. can we scale linearly by adding more partitions?

Problems

Coordinating MPOs (i)

- Partitions have to agree on the order of MPOs.

Problems

Coordinating MPOs (i)

- Partitions have to agree on the order of MPOs.

OP1: A=10 B=10

A

B

OP2: A=5   B=5

Problems

Coordinating MPOs (i)

- Partitions have to agree on the order of MPOs.

OP1: A=10

A 

B 

OP2: A=5 B=5

OP1: B=10

Problems

Coordinating MPOs (i)

- Partitions have to agree on the order of MPOs.

OP1: A=10
OP2: A=5

A 

B 

OP2: B=5
OP1: B=10

Problems

Coordinating MPOs (i)

- Partitions have to agree on the order of MPOs.

OP1: A=10
OP2: A=5

A


B


OP2: B=5
OP1: B=10

A=5

B=10

Problems

Coordinating MPOs (i)

- Partitions have to agree on the order of MPOs.

OP1: A=10
OP2: A=5

A=5

A 

B 

OP2: B=5
OP1: B=10

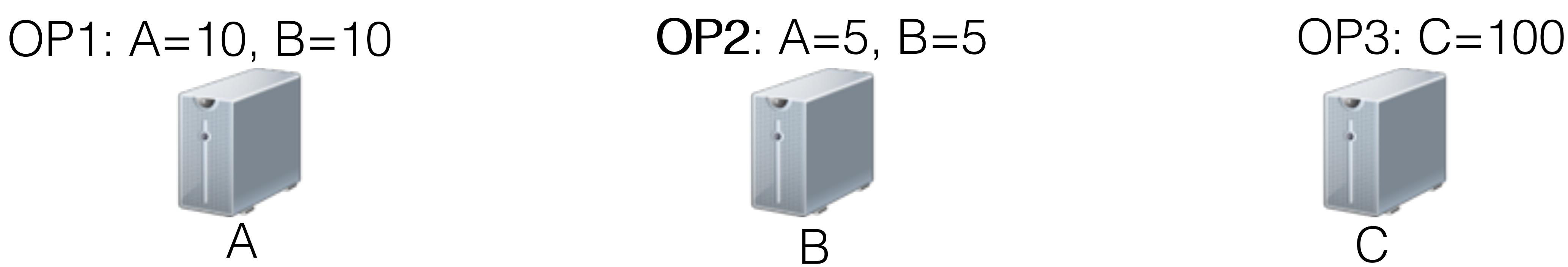
B=10

- Coordinating MPOs is slow
  - Replication + multiple inter-group communication
- In existing systems, the coordination of MPOs lies on the critical path of execution!
  - Partitions sit idle while coordinating MPOs=> throughput reduced

Problems

Coordinating MPOs (ii)

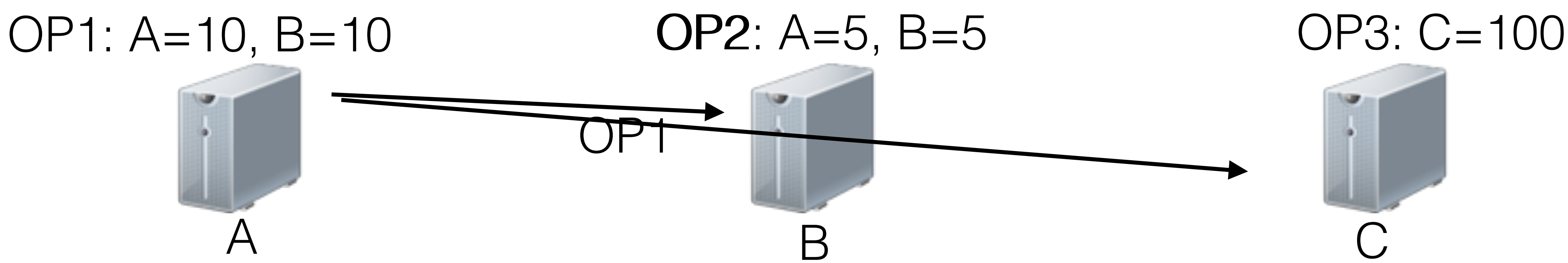
- Calvin requires all-to-all synchronization to order ops
  - Progresses in round, in each round:



Problems

Coordinating MPOs (ii)

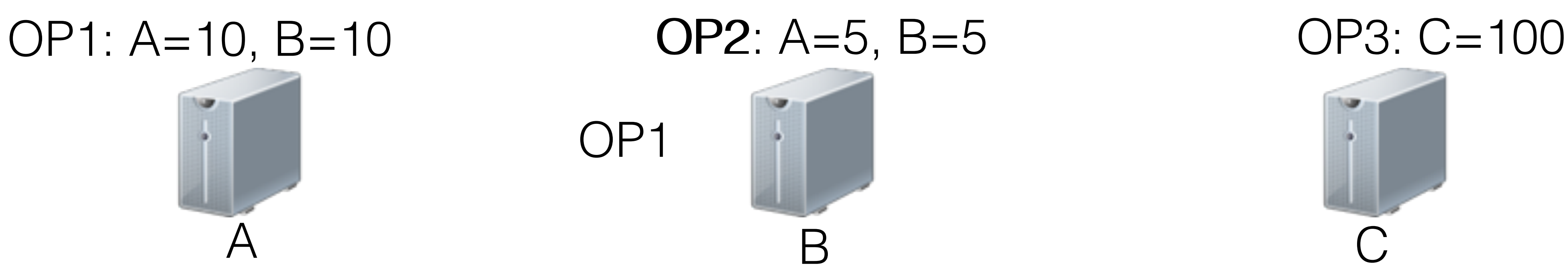
- Calvin requires all-to-all synchronization to order ops
  - Progresses in round, in each round:



Problems

Coordinating MPOs (ii)

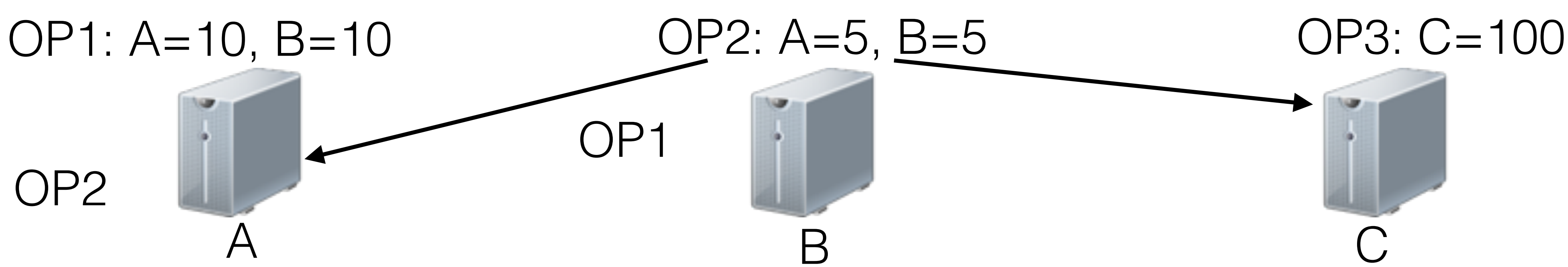
- Calvin requires all-to-all synchronization to order ops
    - Progresses in round, in each round:



Problems

Coordinating MPOs (ii)

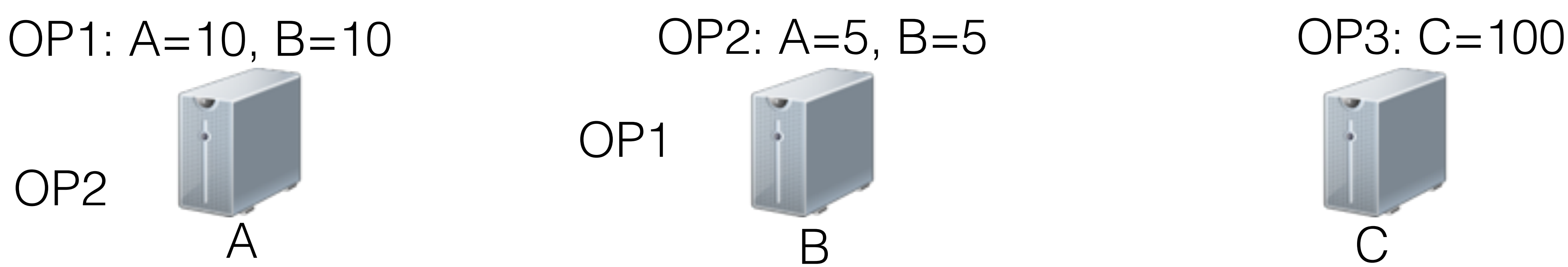
- Calvin requires all-to-all synchronization to order ops
    - Progresses in round, in each round:



Problems

Coordinating MPOs (ii)

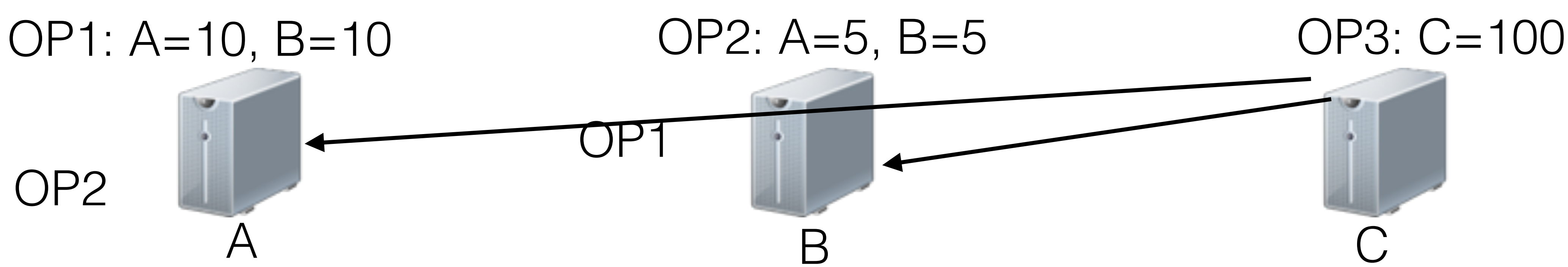
- Calvin requires all-to-all synchronization to order ops
  - Progresses in round, in each round:



Problems

Coordinating MPOs (ii)

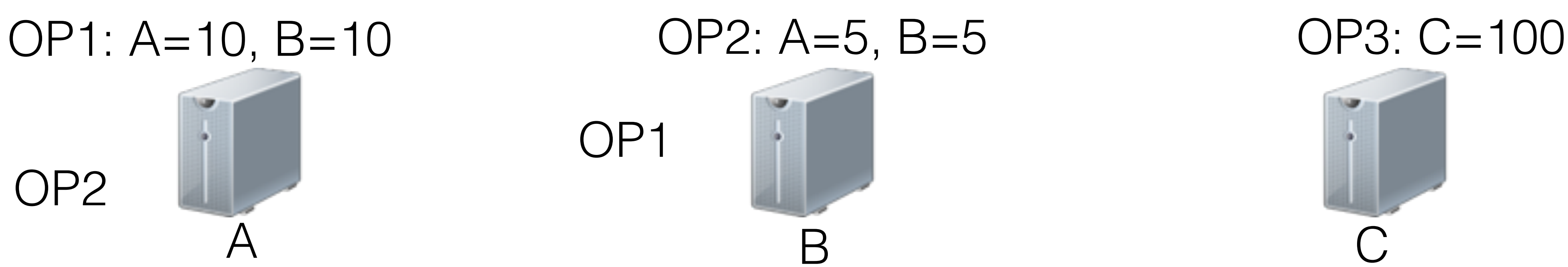
- Calvin requires all-to-all synchronization to order ops
  - Progresses in round, in each round:



Problems

Coordinating MPOs (ii)

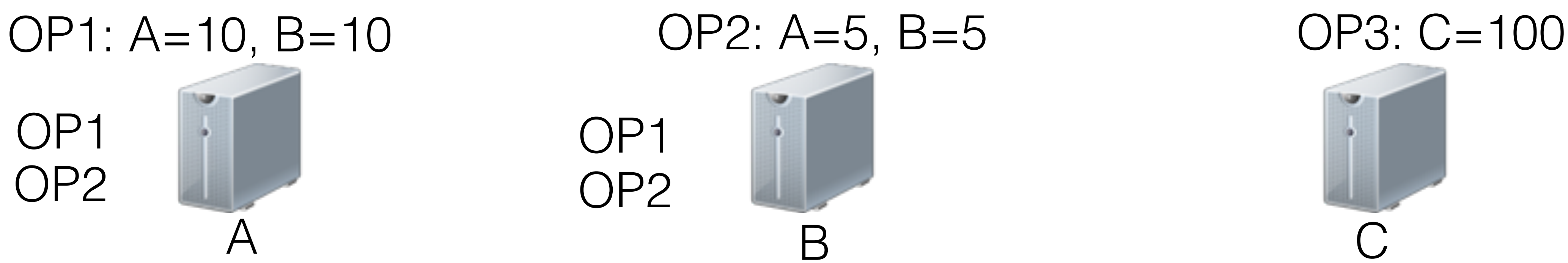
- Calvin requires all-to-all synchronization to order ops
    - Progresses in round, in each round:



Problems

Coordinating MPOs (ii)

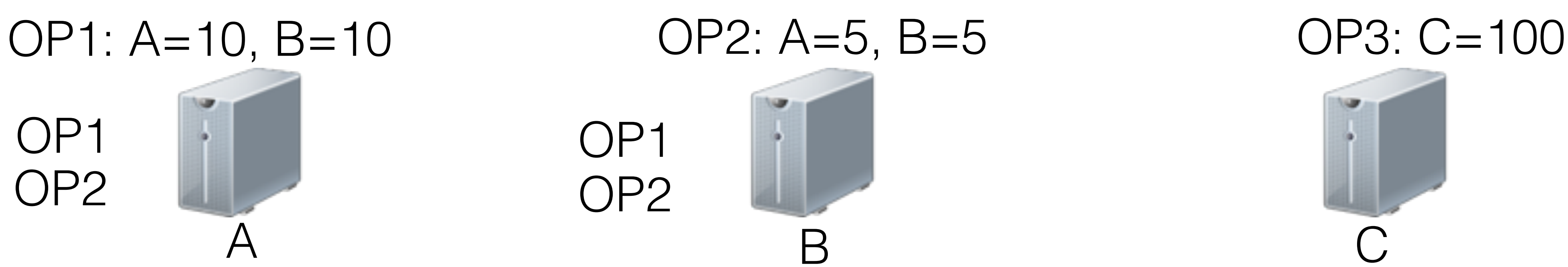
- Calvin requires all-to-all synchronization to order ops
  - Progresses in round, in each round:



Problems

Coordinating MPOs (ii)

- Calvin requires all-to-all synchronization to order ops
    - Progresses in round, in each round:

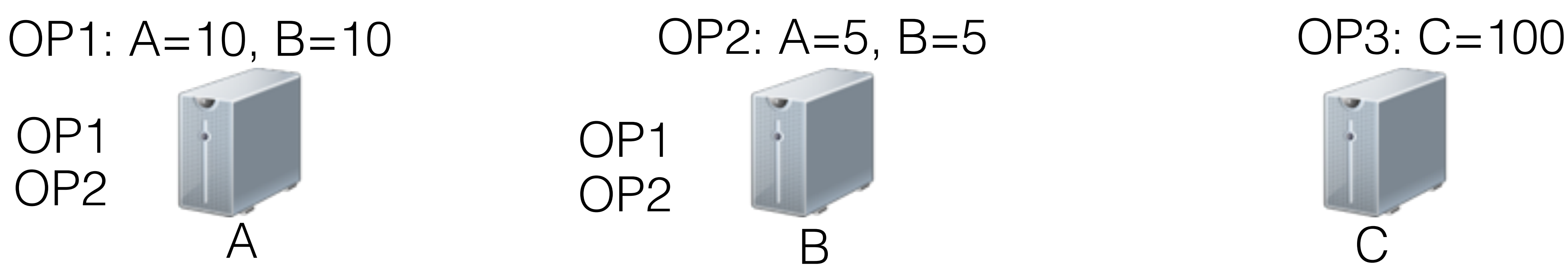


- Ordering lies on the critical path of execution
- Non-scalable

Problems

Coordinating MPOs (ii)

- Calvin requires all-to-all synchronization to order ops
  - Progresses in round, in each round:



  - Ordering lies on the critical path of execution
  - Non-scalable

- Scalable SMR leverages atomic multicast to order ops
  - More scalable than Calvin, but ordering still lies on the critical path of execution
  - Additional messages exchanged between partitions to ensure linearizability*

*Omitted due to time constraints; refer to paper if interested

Solution

Genepi

- Remove the coordination of MPOs from the critical path of operation execution by:
  - Schedule MPOs to future round

    => overlap the ordering of MPOs & processing of ordered ops

- Genepi:
  - Efficient execution protocol ensuring linearizability*
  - Scraper: an ordering building block for Genepi

Solution

Genepi

- Remove the coordination of MPOs from the critical path of operation execution by:
  - Schedule MPOs to future round

    => overlap the ordering of MPOs & processing of ordered ops
- Genepi:
  - Efficient execution protocol ensuring linearizability*
  - Scraper: an ordering building block for Genepi

*Omitted due to time constraints; refer to paper if interested

Solution

Genepi

- Remove the coordination of MPOs from the critical path of operation execution by:
  - Schedule MPOs to future round

    => overlap the ordering of MPOs & processing of ordered ops

- Genepi:
  - Efficient execution protocol ensuring linearizability*
  - Scraper: an ordering building block for Genepi

Scalable consensus for partial replication

*Omitted due to time constraints; refer to paper if interested

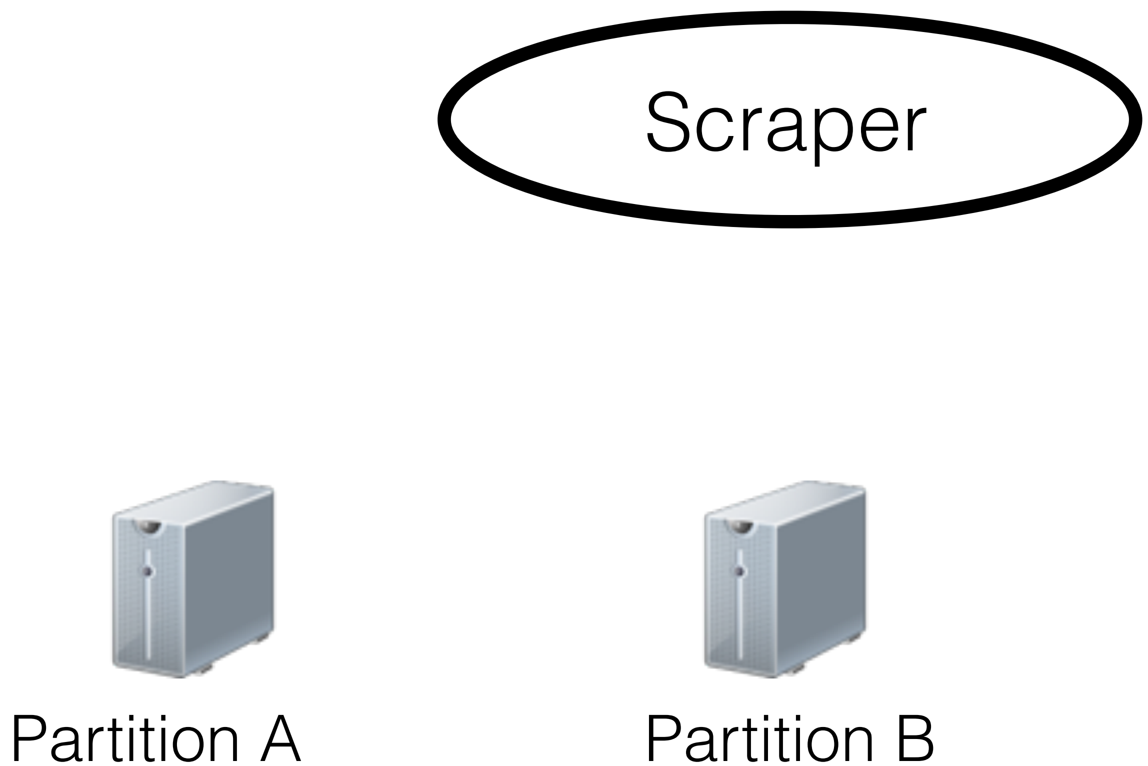
Solution

Scraper abstraction

- Formal specifications can be found in the paper
- S-Propose(SPOs, Rs, MPOs, Rm)
  - Propose accumulated ops for each round
  - **Rs** current round & **Rm** a future round => only low bound on final round
- S-Decide(OPs, R)
  - Triggered when the operations for **R** has been decided
  - **R** can only be decided if **1, 2, …, R-1** have all been decided

Solution

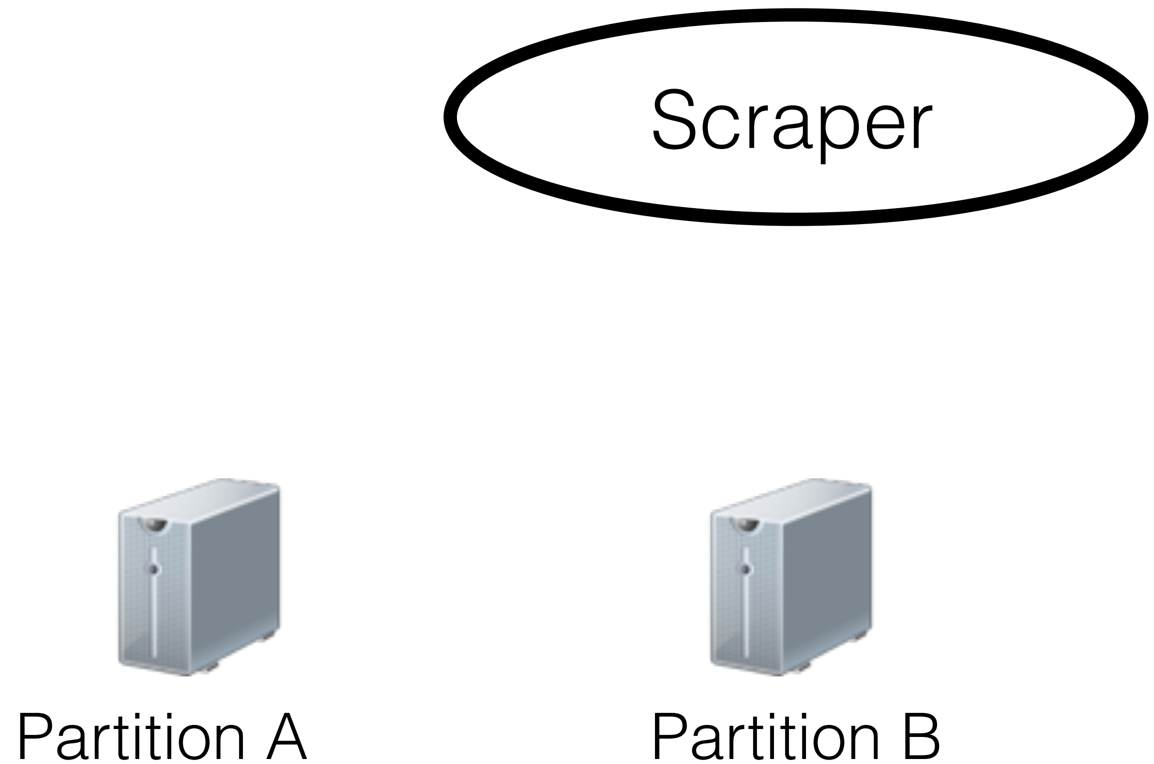
Genepi Execution



Solution

Genepi Execution

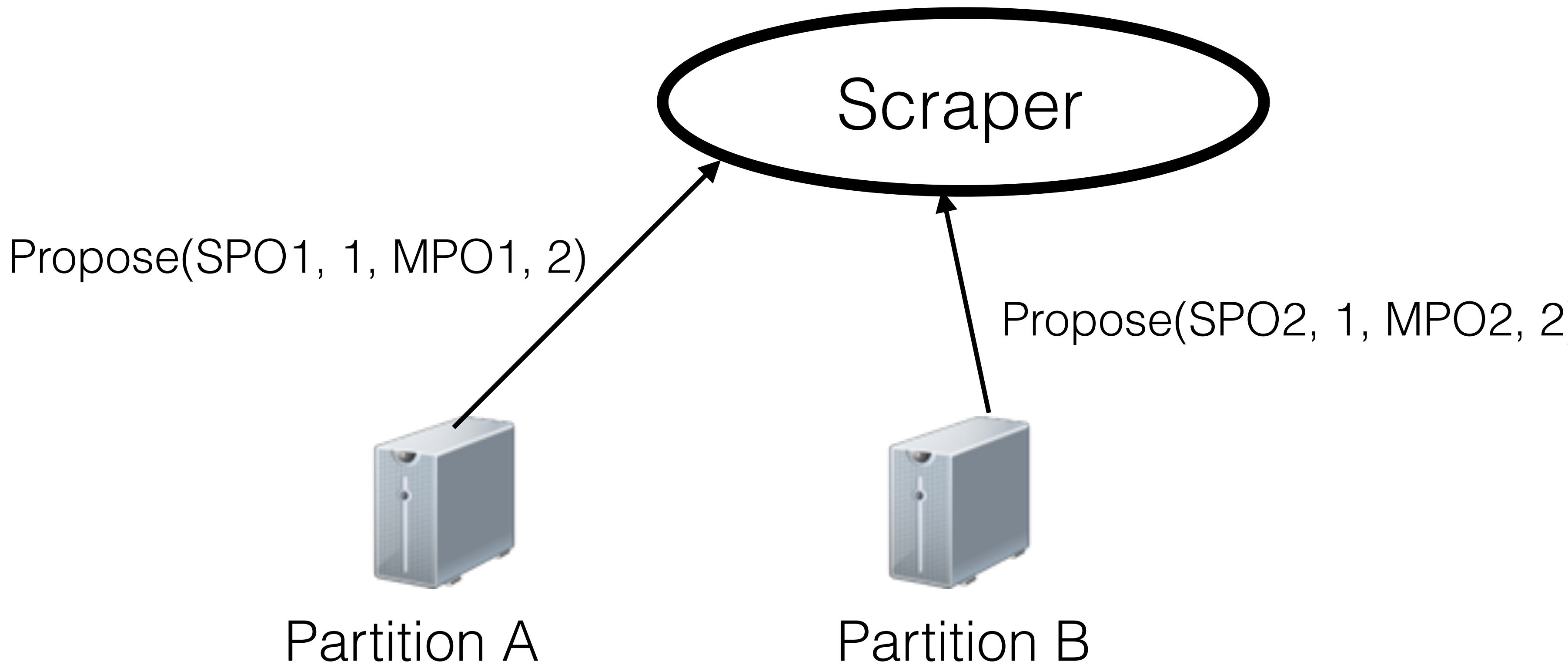
Round 1



Solution

Genepi Execution

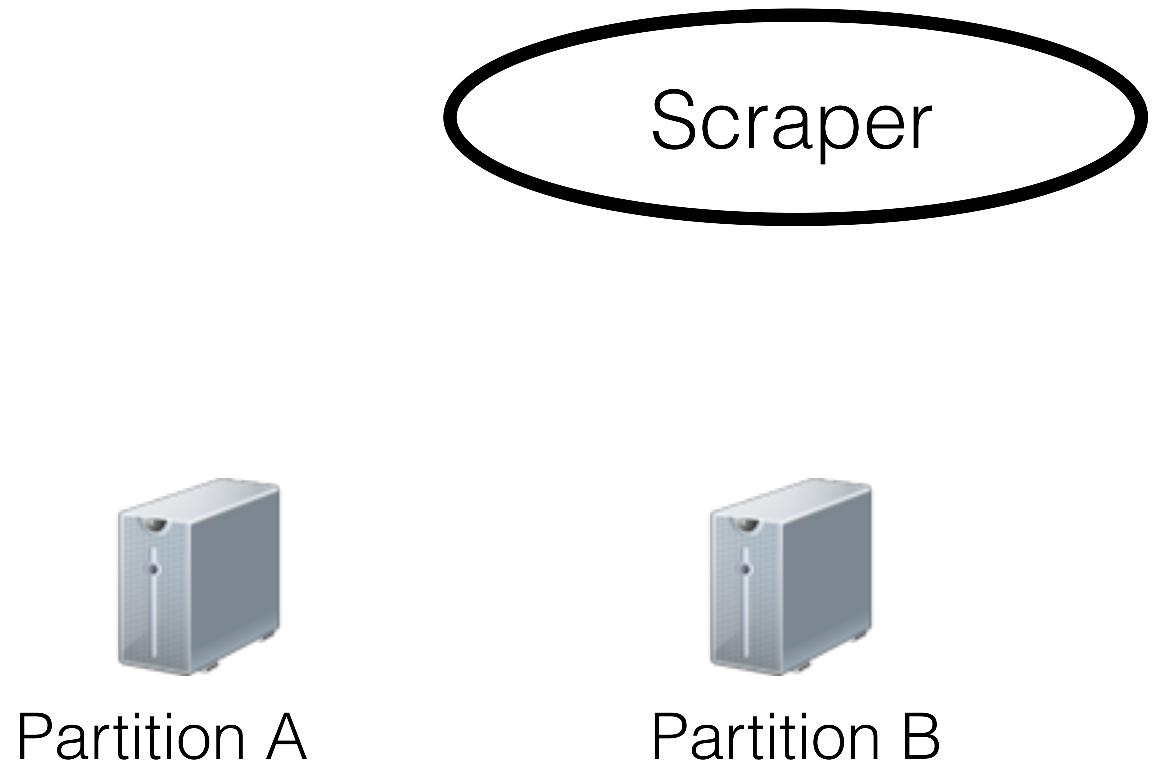
Round 1



Solution

Genepi Execution

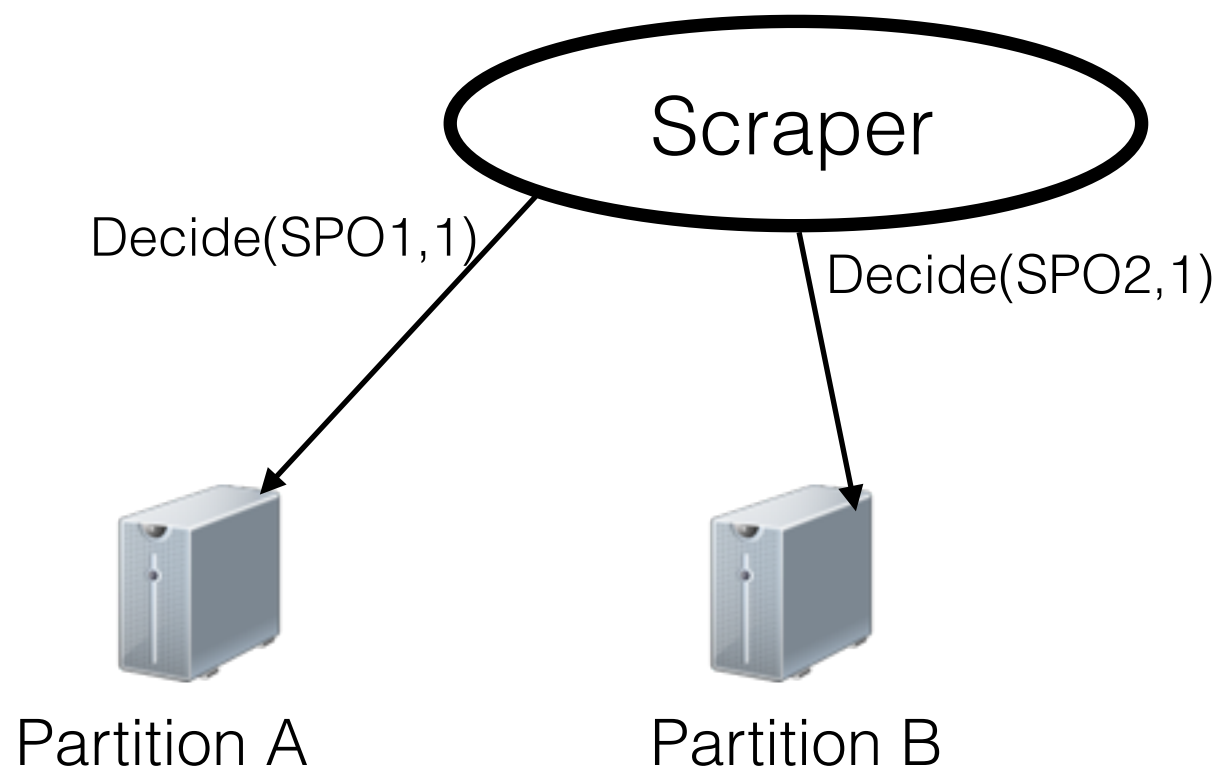
Round 1



Solution

Genepi Execution

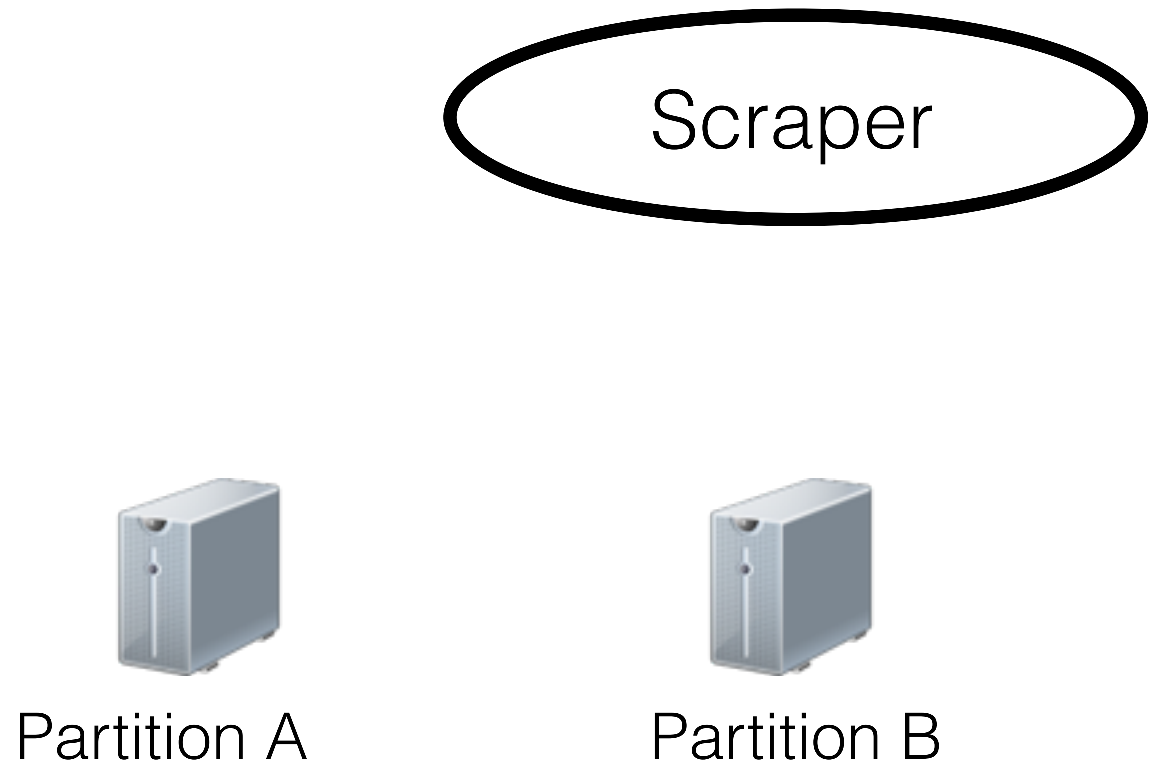
Round 1



Solution

Genepi Execution

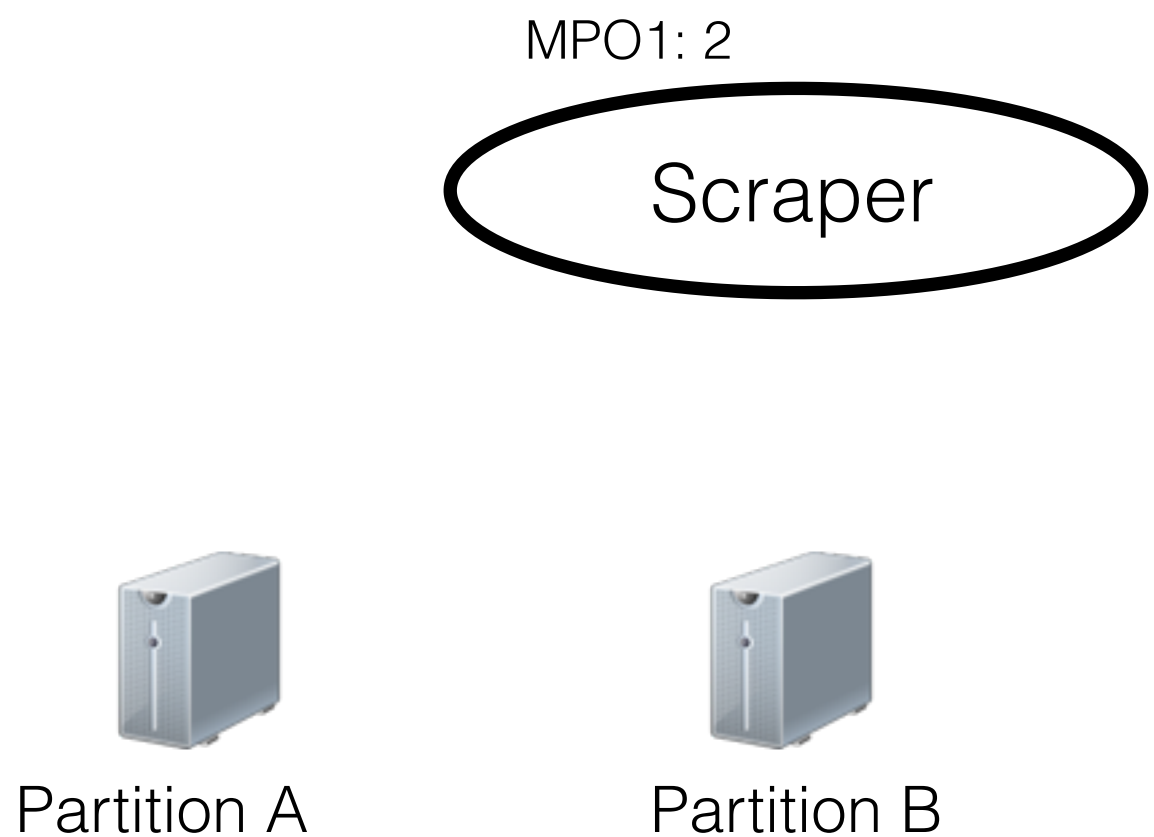
Round 1



Solution

Genepi Execution

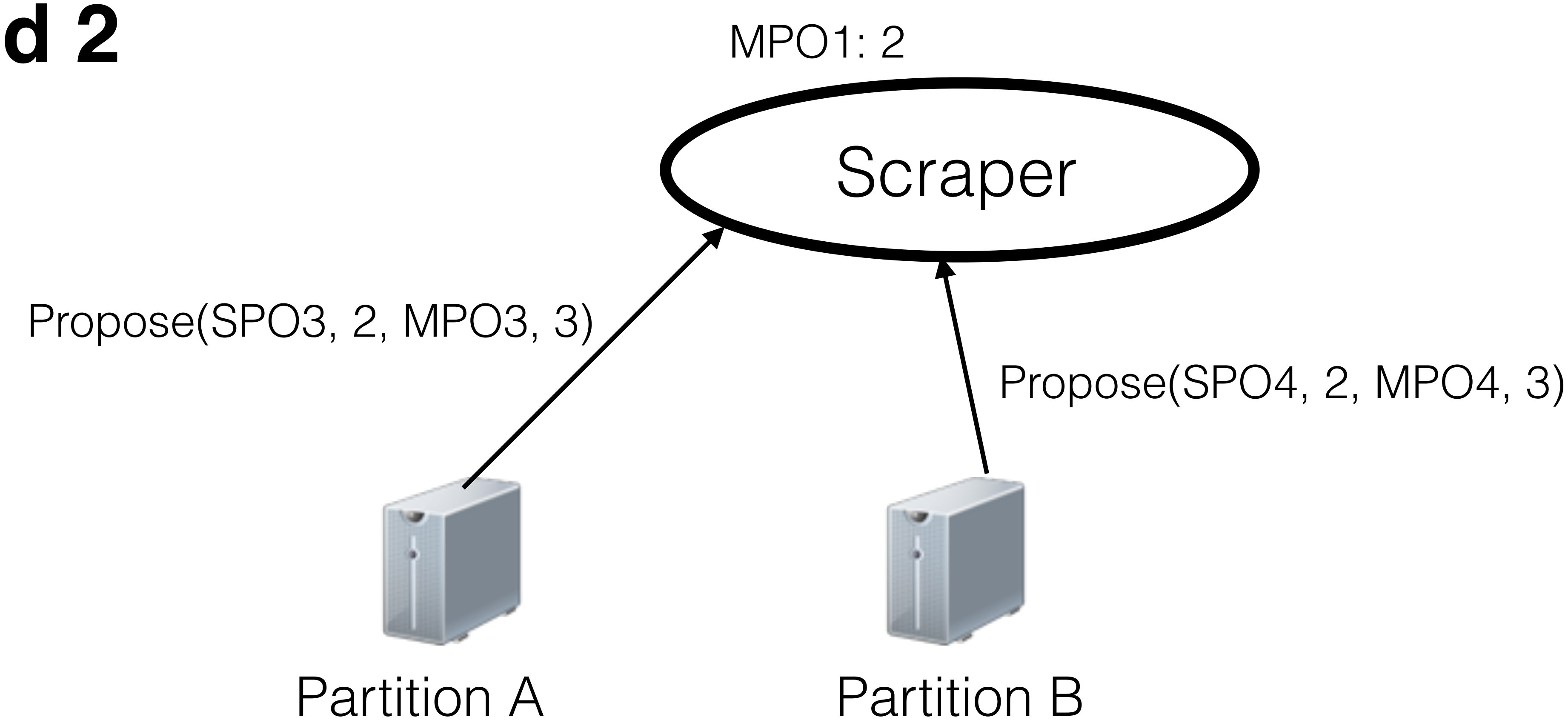
Round 2



Solution

Genepi Execution

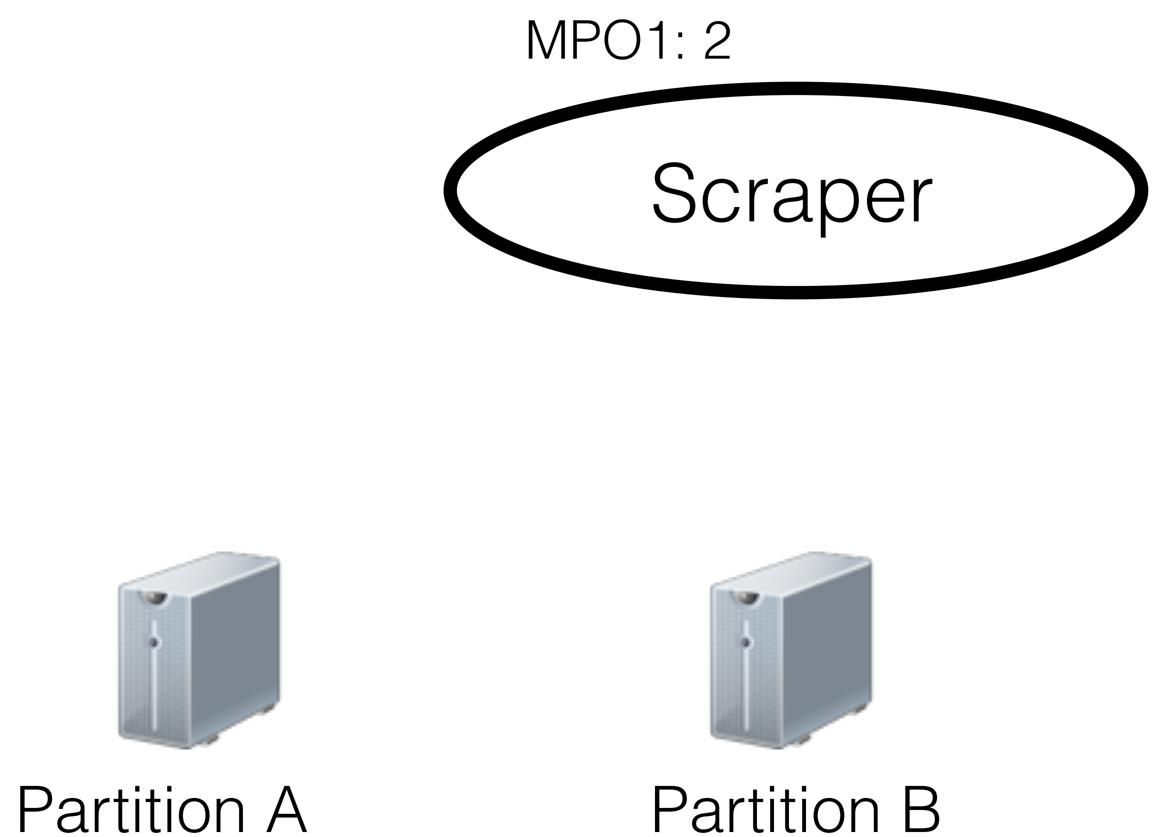
Round 2



Solution

Genepi Execution

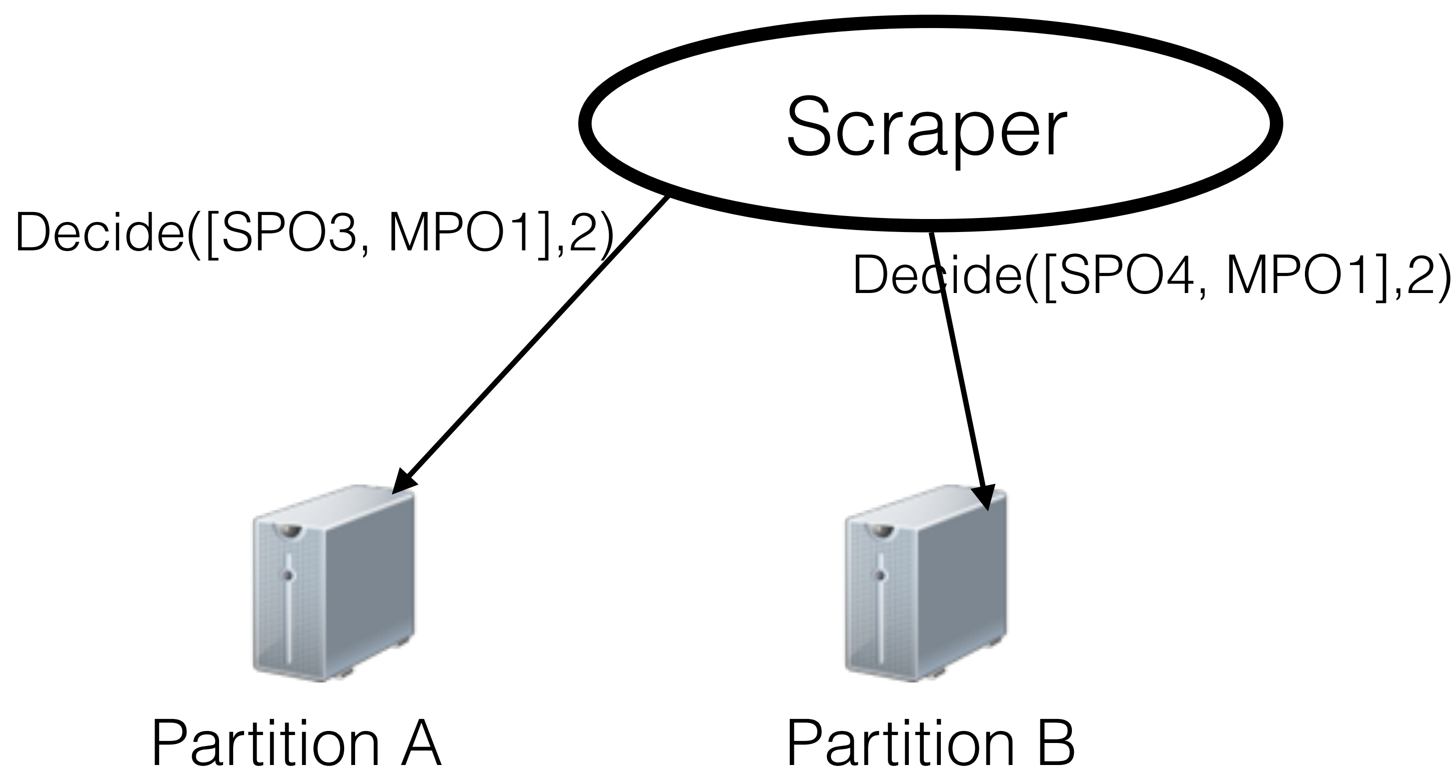
Round 2



Solution

Genepi Execution

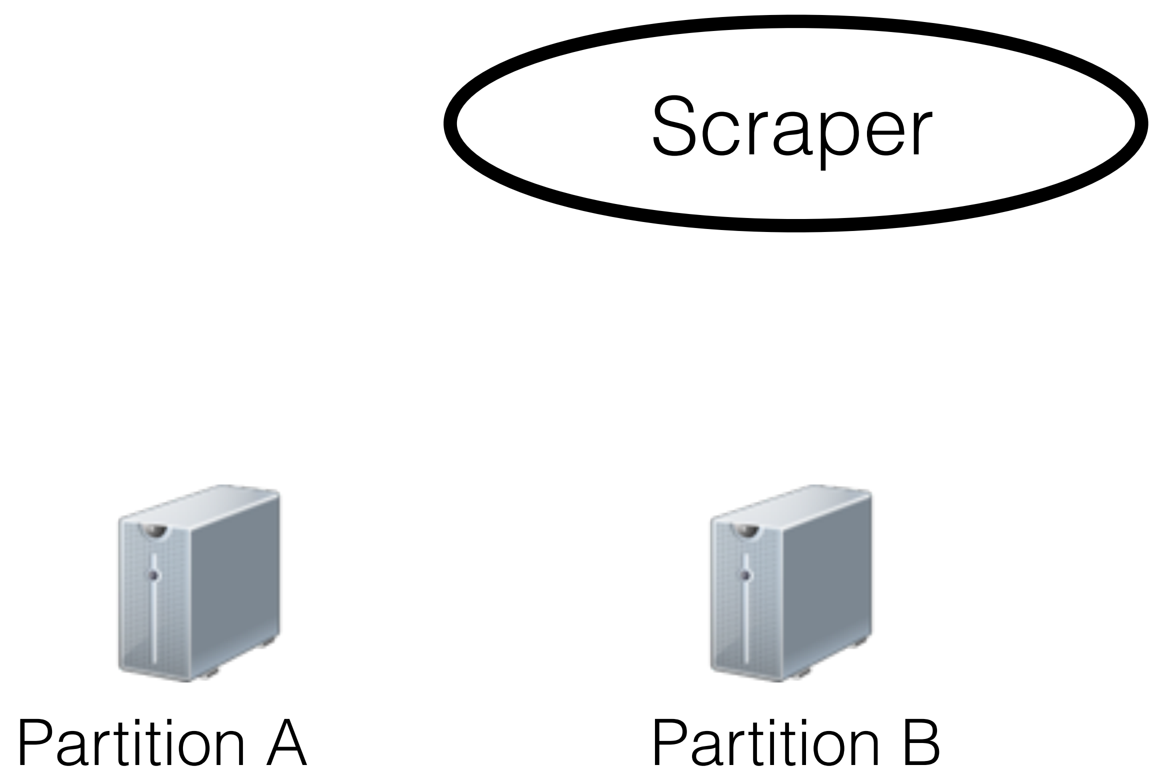
Round 2



Solution

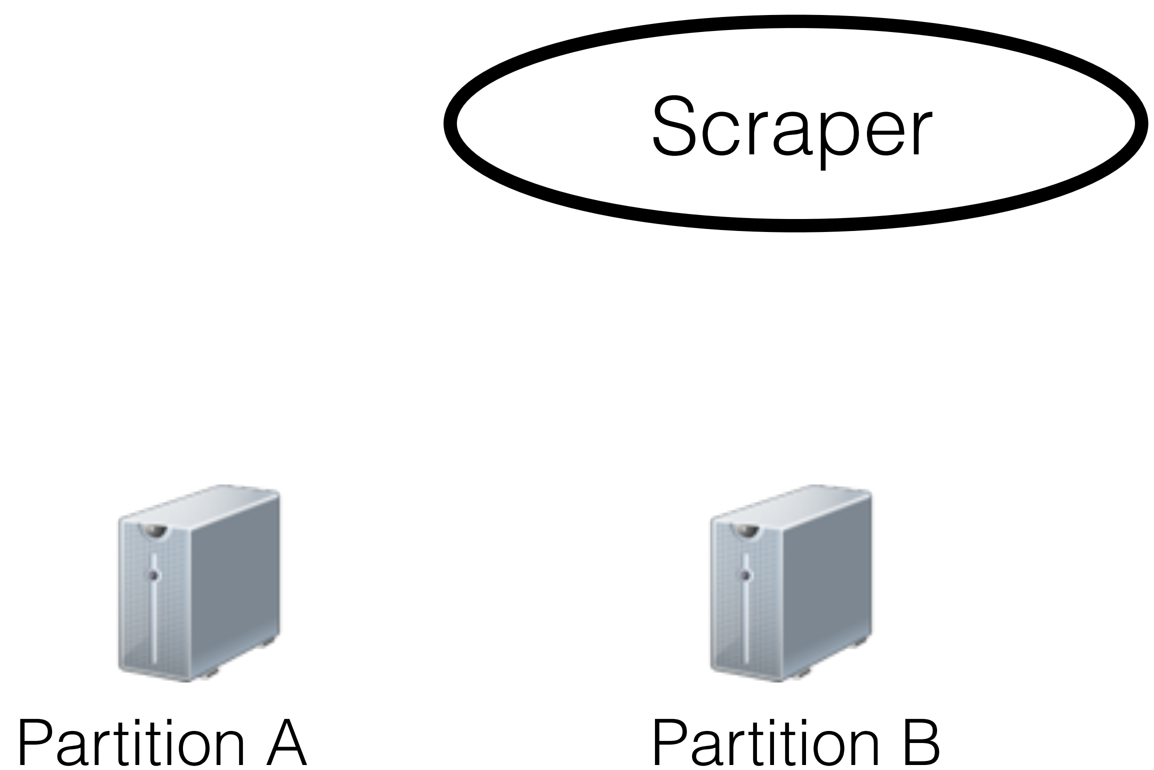
Genepi Execution

Round 2



Solution

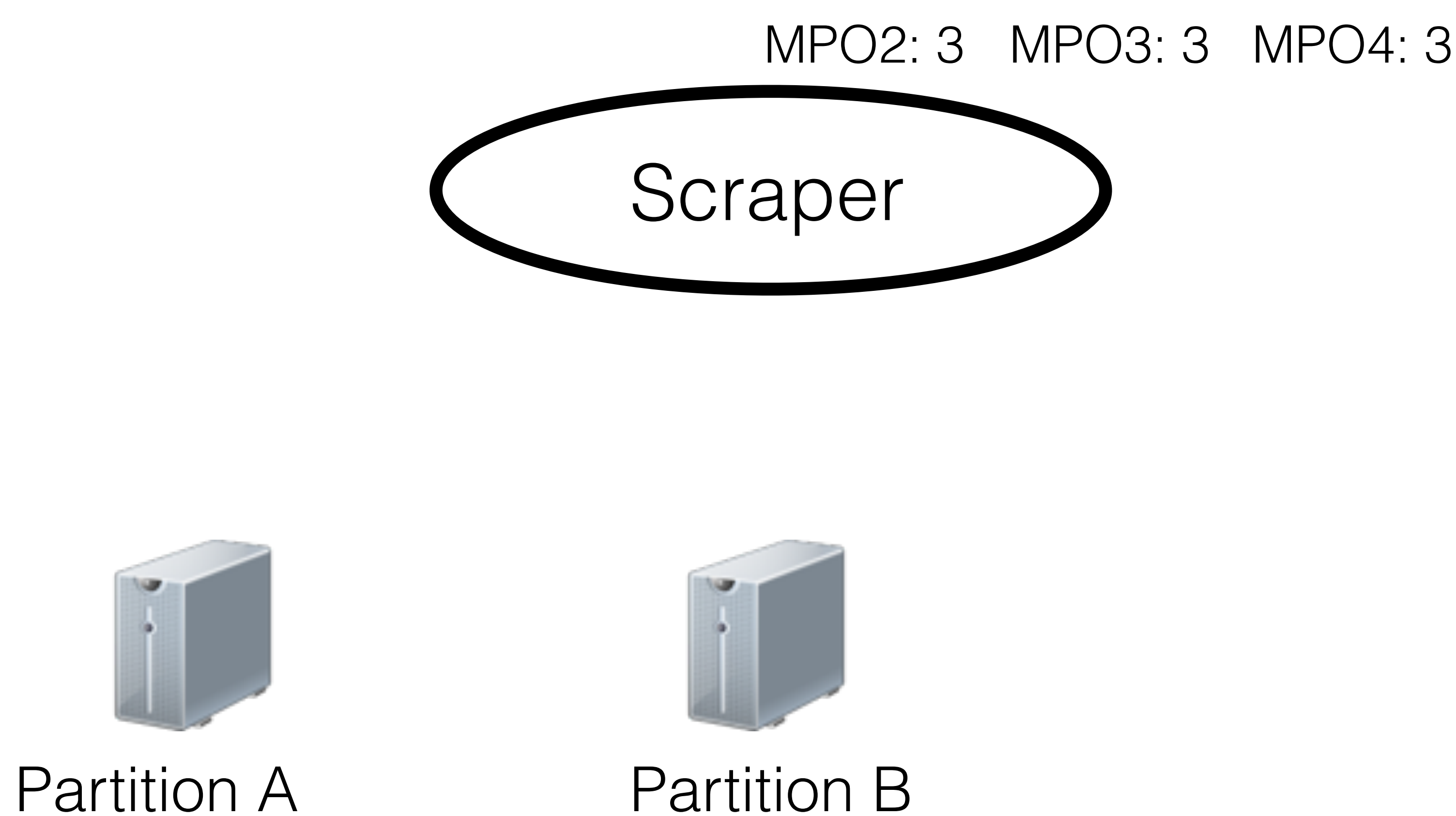
Genepi Execution



Solution

Genepi Execution

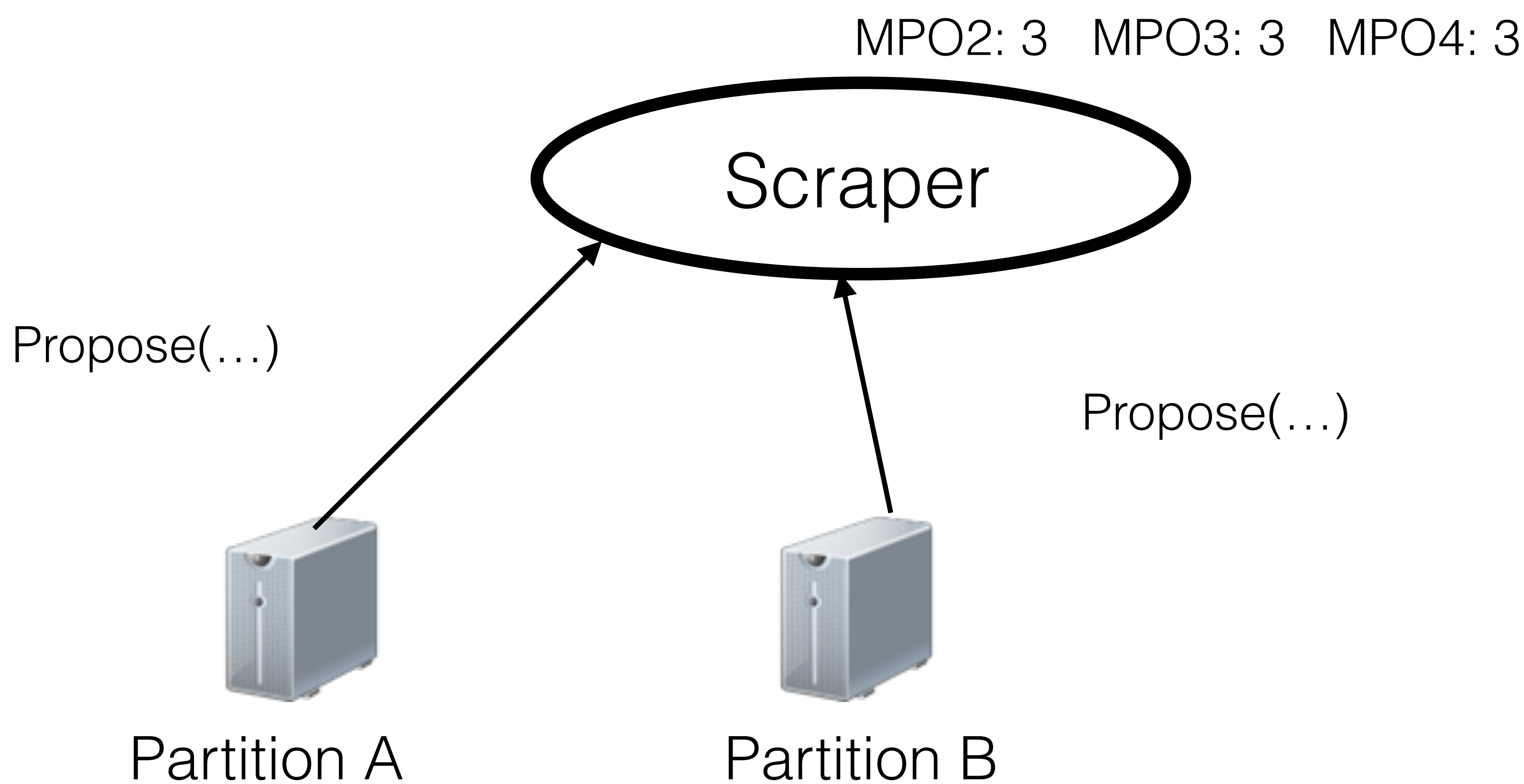
Round 3



Solution

Genepi Execution

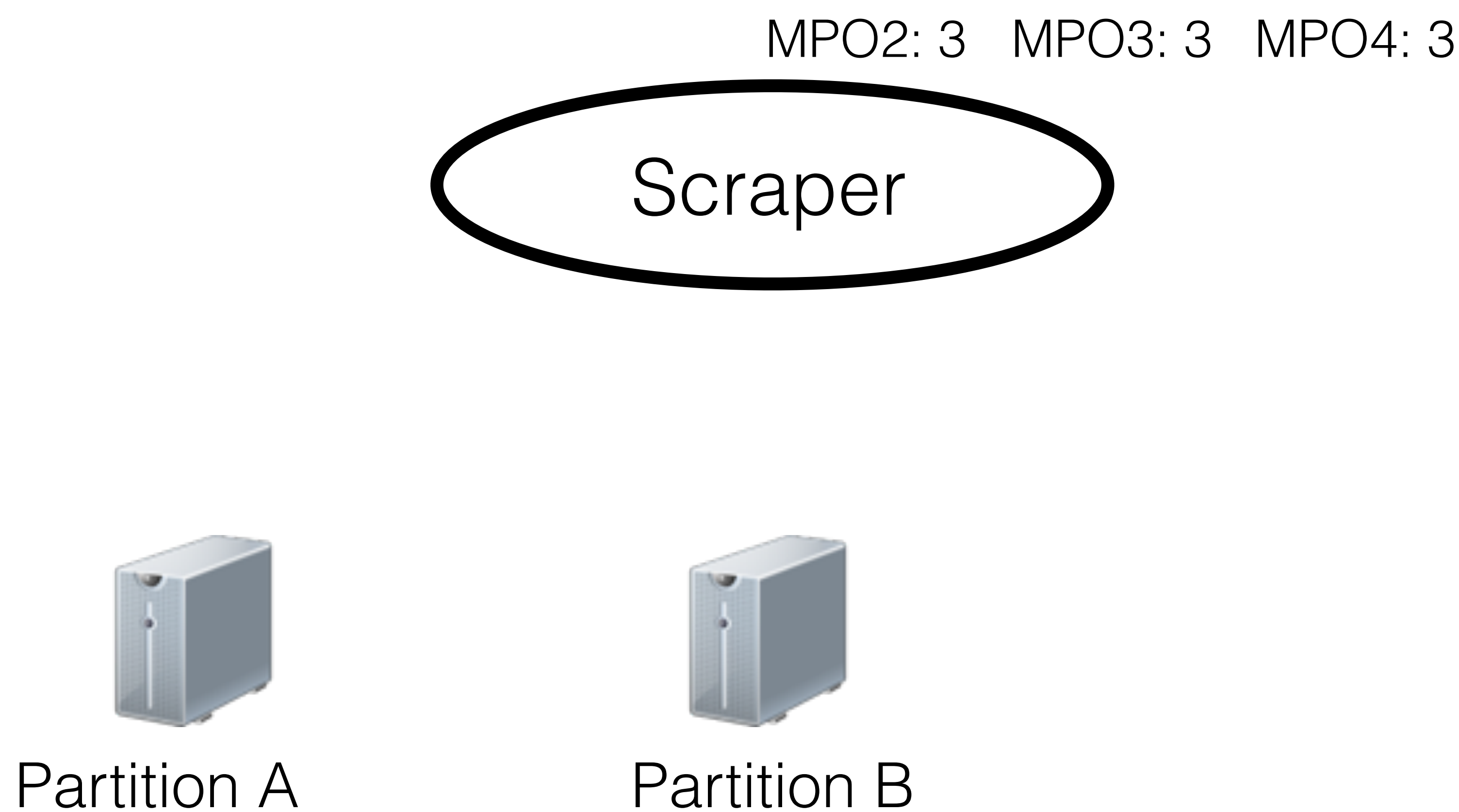
Round 3



Solution

Genepi Execution

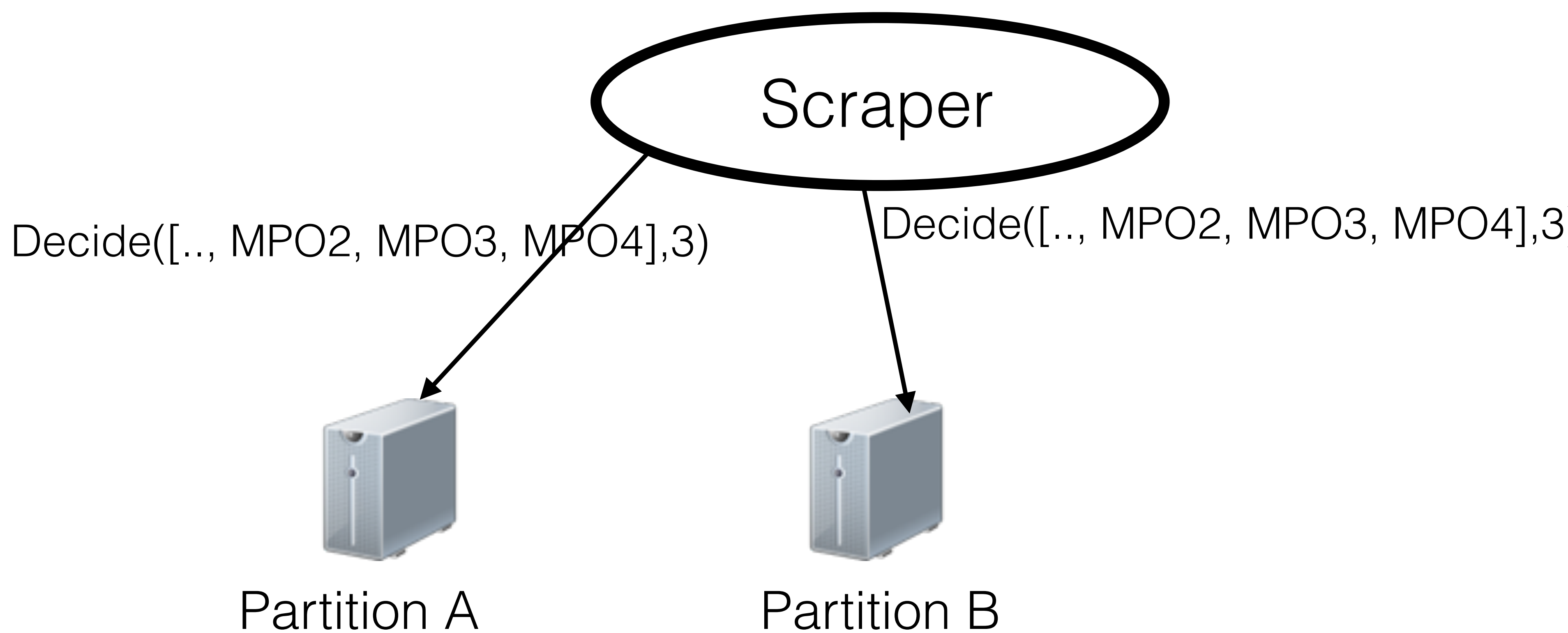
Round 3



Solution

Genepi Execution

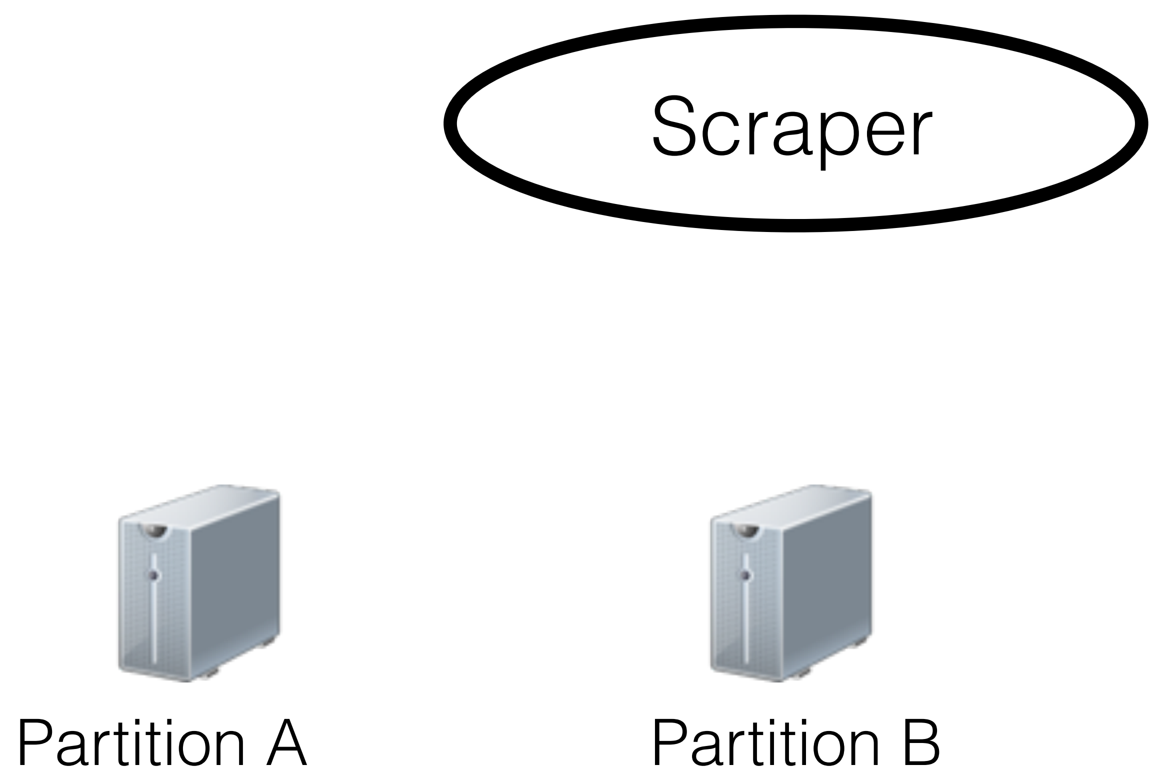
Round 3



Solution

Genepi Execution

Round 3



Solution

Scraper design (i)

Avoiding synchronizing all partitions for scalability

Solution

Scraper design (i)

Avoiding synchronizing all partitions for scalability

Solution

Scraper design (i)

Avoiding synchronizing all partitions for scalability

↓

Partitions unilaterally advance rounds

Solution

Scraper design (i)

Avoiding synchronizing all partitions for scalability

↓

Partitions unilaterally advance rounds

↓

Solution

Scraper design (i)

Avoiding synchronizing all partitions for scalability

↓

Partitions unilaterally advance rounds

↓

How to ensure they agree on rounds of ops?

Solution

Scraper design (i)

Avoiding synchronizing all partitions for scalability

↓

Partitions unilaterally advance rounds

↓

How to ensure they agree on rounds of ops?

- Key idea: a two-phase-commit-like protocol for partitions to agree on the round of an operation

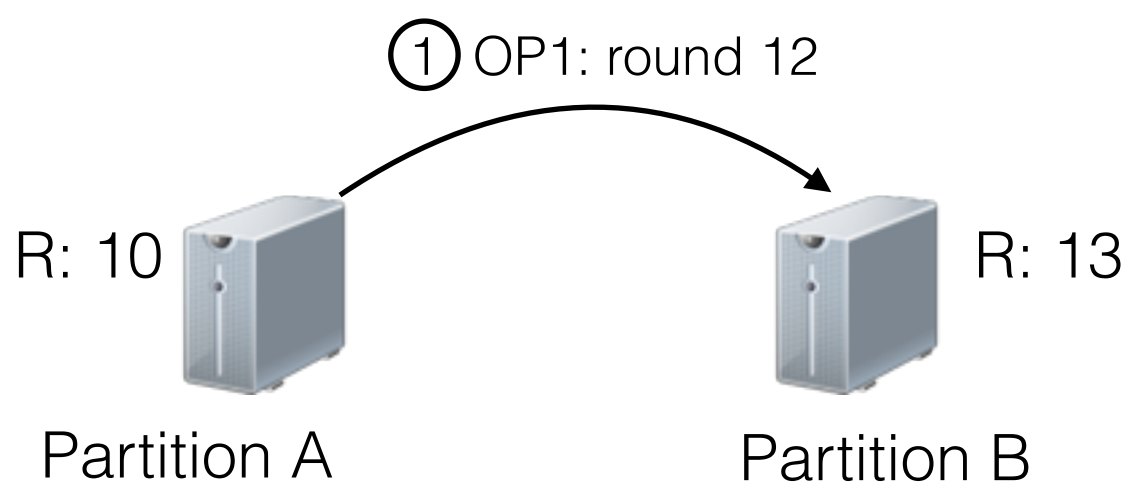
Solution

Scraper design (ii)

Solution

Scraper design (ii)

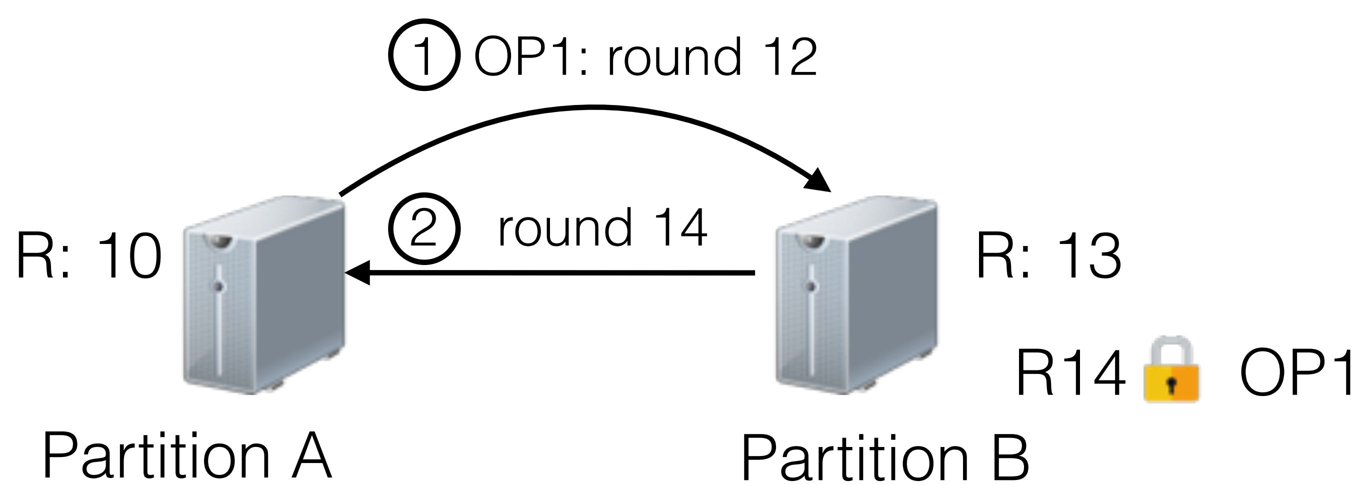
- 1. Coordinator sends request with min_round



Solution

Scraper design (ii)

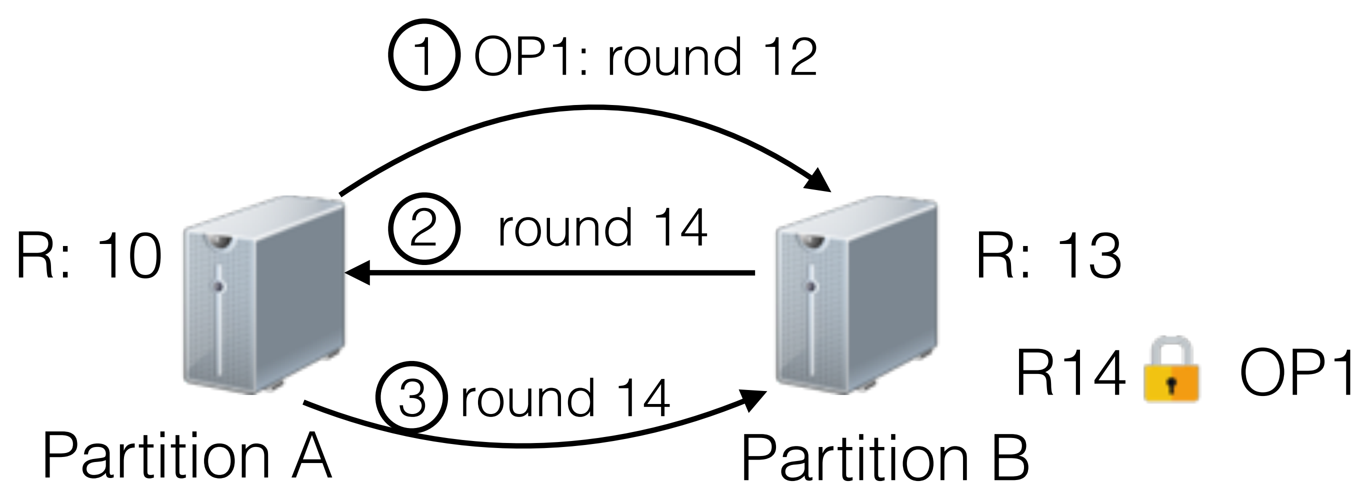
- 1. Coordinator sends request with min_round
- 2. Partitions propose max(min_round, decided round+1)



Solution

Scraper design (ii)

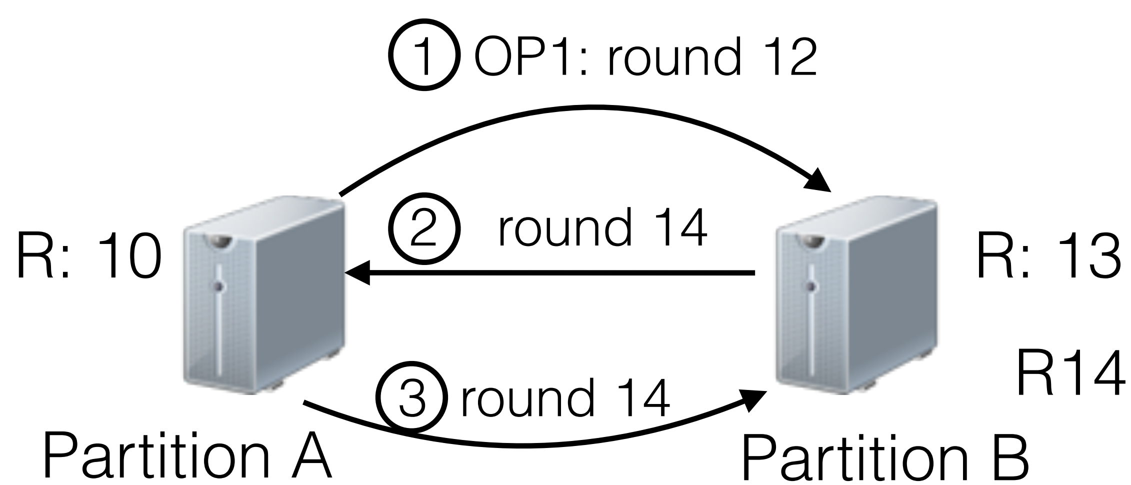
- 1. Coordinator sends request with min_round
- 2. Partitions propose max(min_round, decided round+1)
- 3. Coordinator decides max(received rounds)



Solution

Scraper design (ii)

- 1. Coordinator sends request with min_round
- 2. Partitions propose max(min_round, decided round+1)
- 3. Coordinator decides max(received rounds)
- 4. Partitions finalize proposal



Solution

Other aspects in the paper

- Replication to ensure fault-tolerance
- Lightweight mechanism to ensure linearizability
    - Delay replying to clients
- Choosing round numbers for MPOS
    - Big enough to allow ordering MPOs
    - Not too large to avoid unnecessary latency overhead

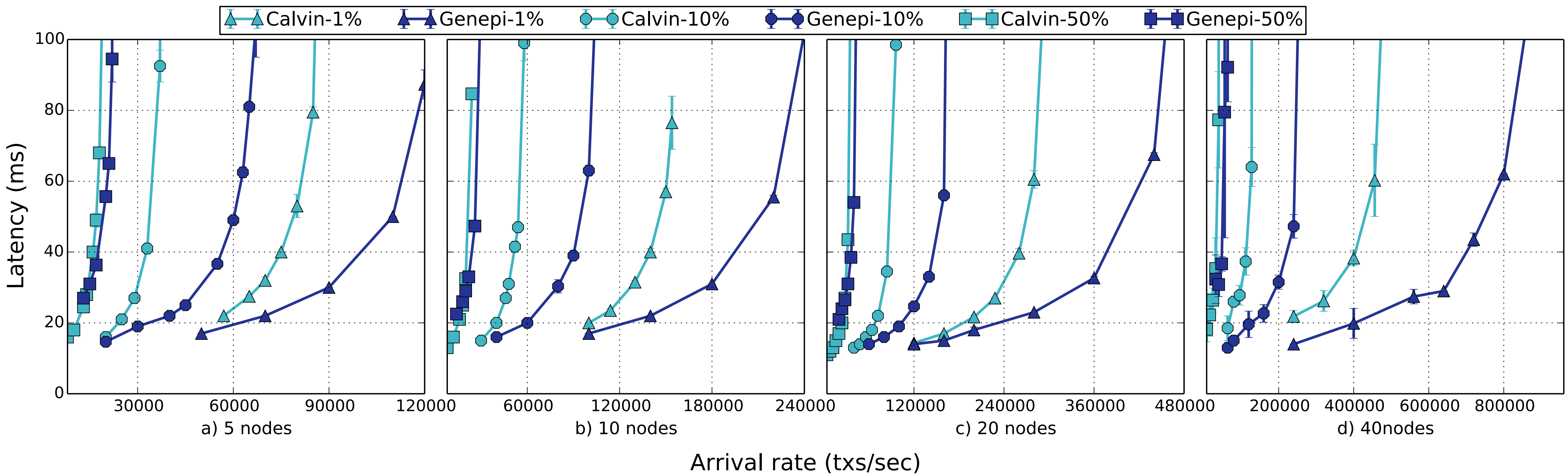
Evaluation

Experimental setup

- Implementation:
  - Calvin, S-SMR and Genepi all implemented based on Calvin's codebase (in C++)
- Deployment:
  - Deployed in Grid'5000.
  - Used up to 40 nodes in the same region; RTT is around 0.4ms
  - Replication cost emulated by injecting 3ms delay
  - 5ms round duration for batching
  - MPOs scheduled two rounds later (2*5ms)

Evaluation

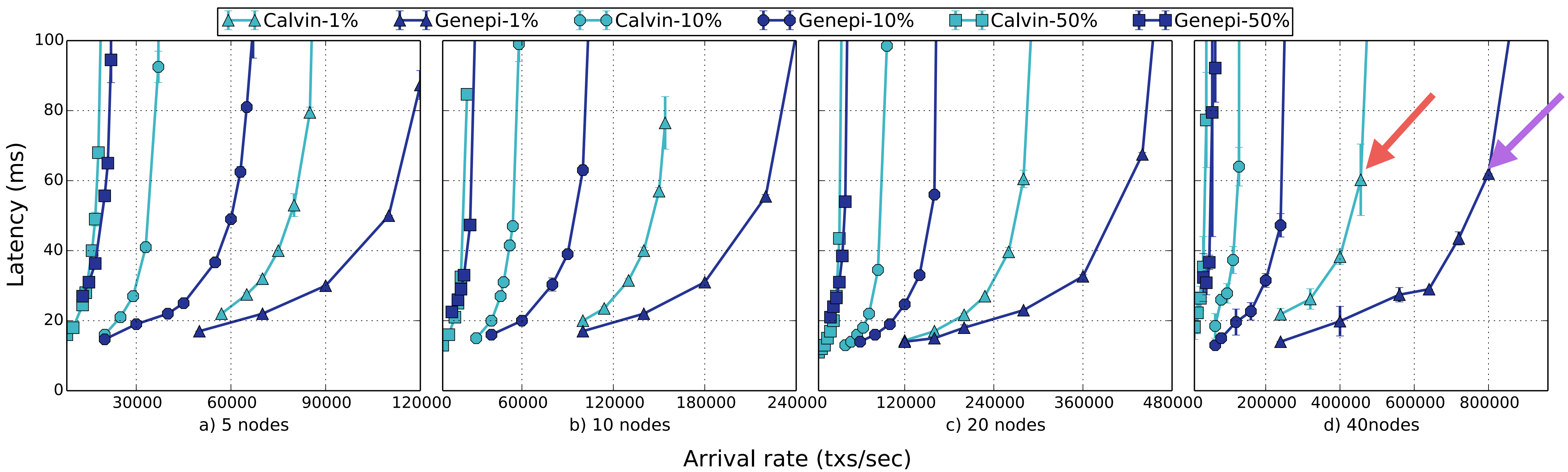
Micro benchmark



- Each op reads & updates 10 keys
- Increase number of nodes & percentage of MPOs

Evaluation

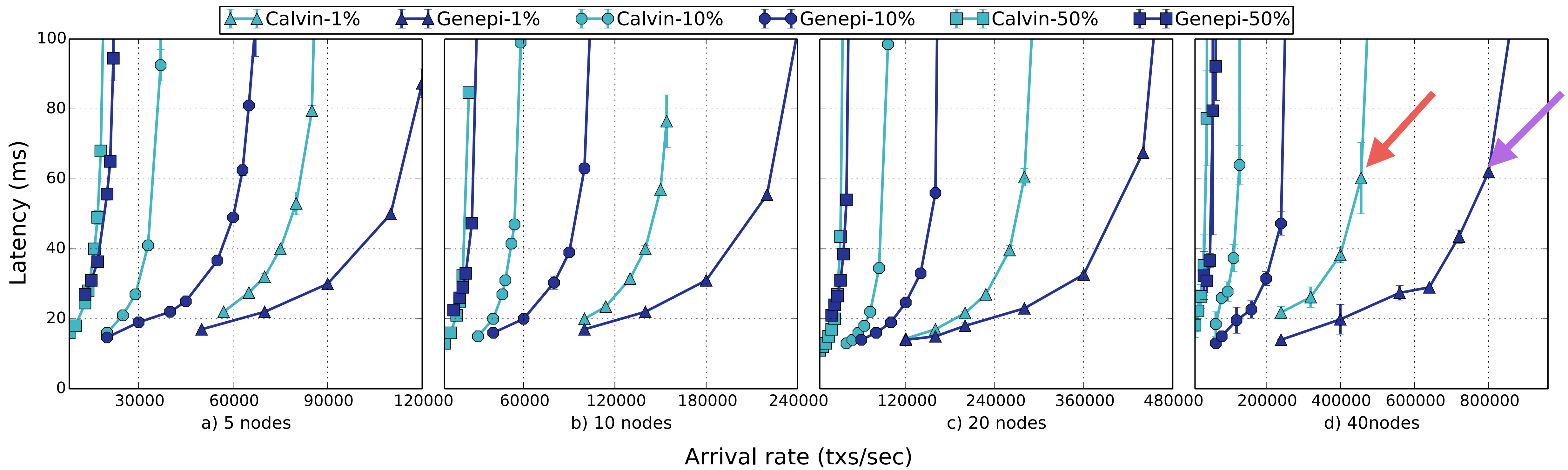
Micro benchmark



- Each op reads & updates 10 keys
- Increase number of nodes & percentage of MPOs

Evaluation

Micro benchmark



- Each op reads & updates 10 keys
- Increase number of nodes & percentage of MPOs

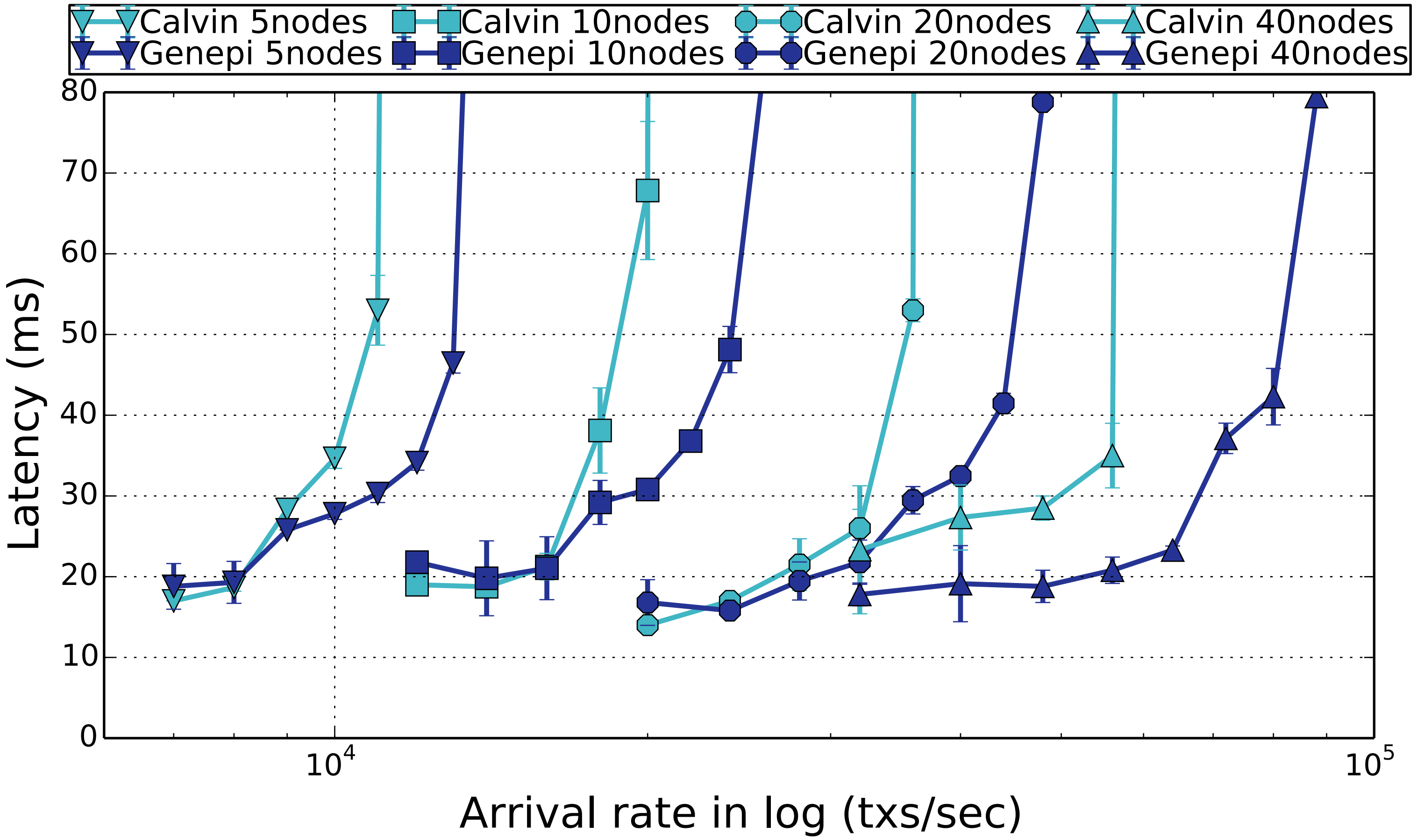
😀 : Genepi scales better than Calvin: 83% higher throughput with 40 nodes & 1% MPOs

🤔 : Latency of MPOs is 7~14 ms higher than SPOs

Evaluation

TPC-C

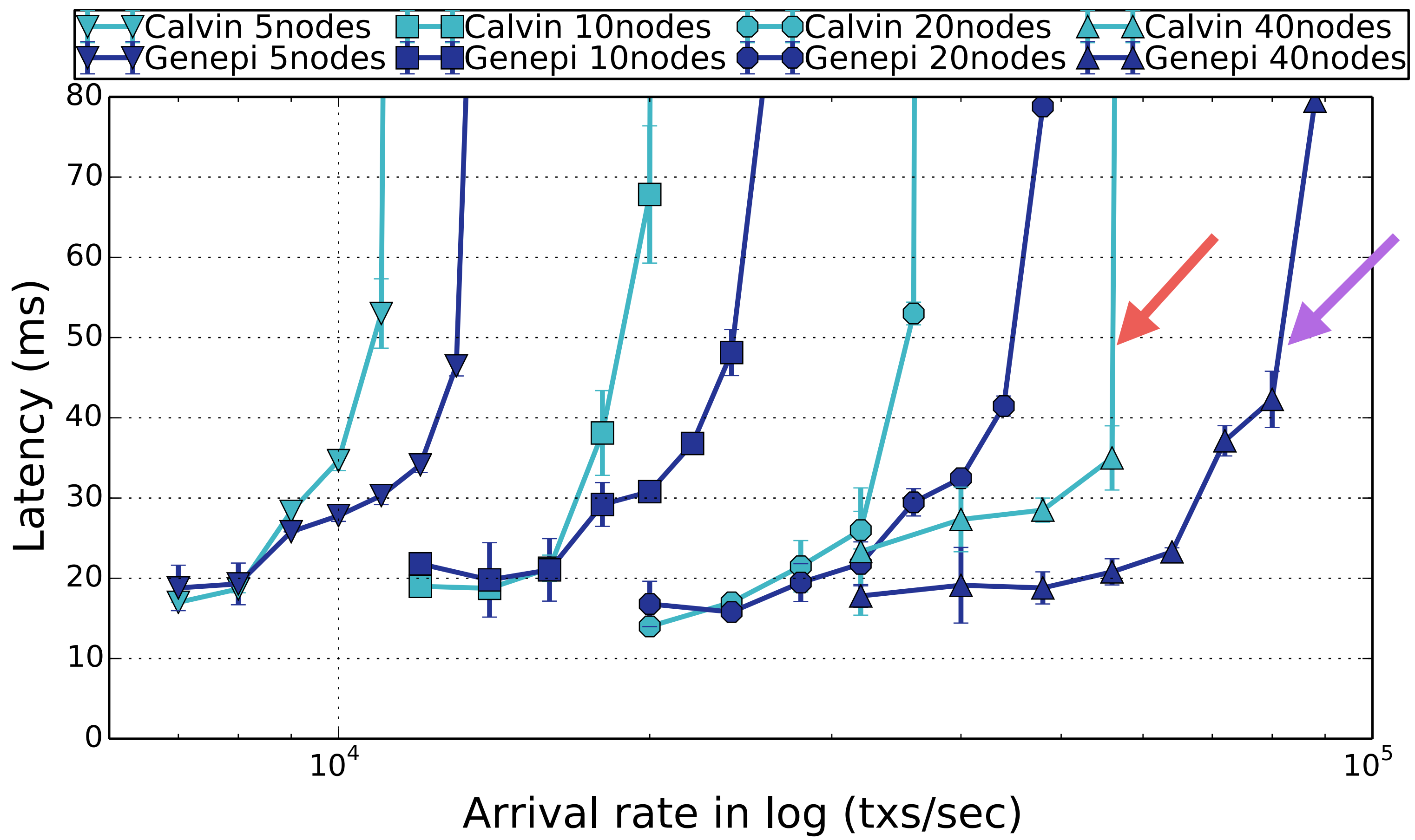
- About 10% distributed transactions
- Includes heavy-weight and/or read-only txns
- At 40 nodes, Genepi has 45% throughput gain



Evaluation

TPC-C

- About 10% distributed transactions
- Includes heavy-weight and/or read-only txns
- At 40 nodes, Genepi has 45% throughput gain



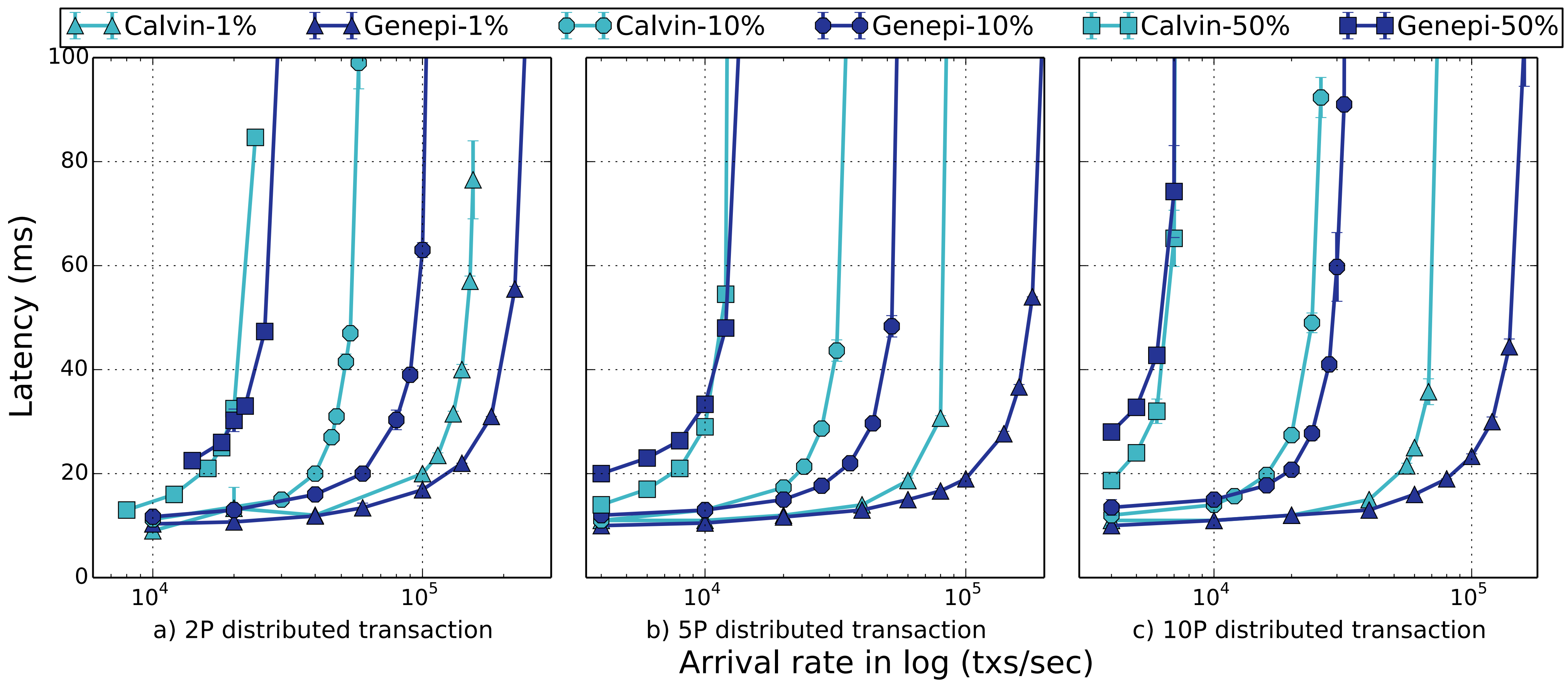
Summary

- Genepi’s idea of postponing the execution of MPOs allow remove MPO coordination from the critical path of operation execution

- Questions?

Evaluation

Micro benchmark



- 10 nodes, Varying the % of MPOs and partitions accessed by MPOs
    - Genepi is only worse for workloads with lots of MPOs that access lots of MPOs!