FORD: Fast One-sided RDMA-based Distributed Transactions for Disaggregated Persistent Memory

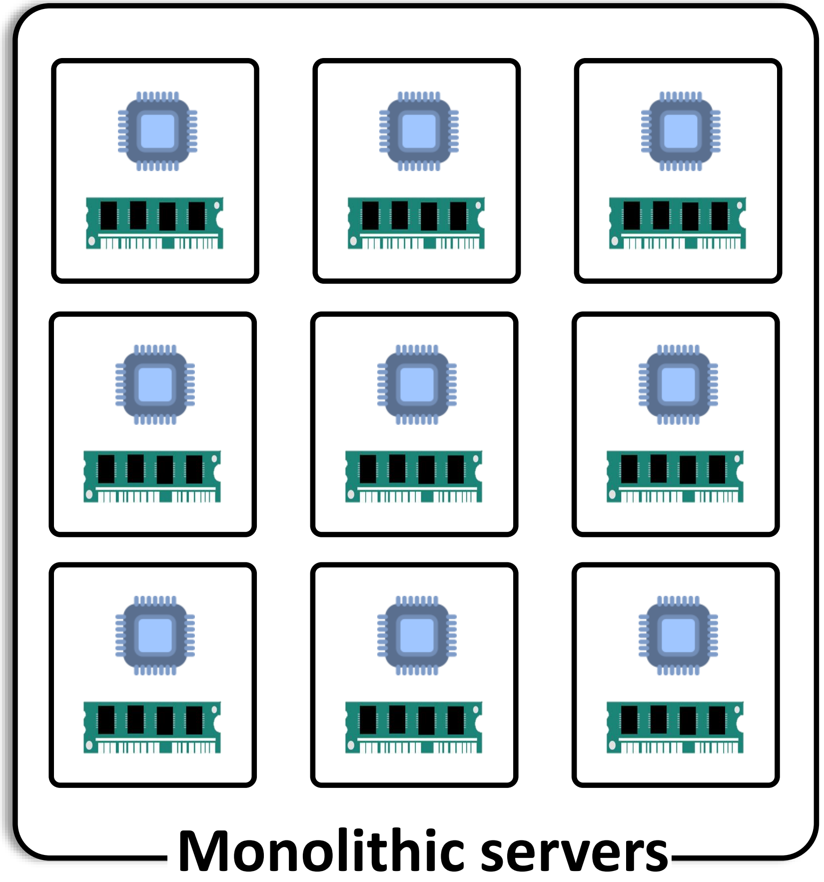
Ming Zhang, Yu Hua, Pengfei Zuo, Lurong Liu

Huazhong University of Science and Technology, China

20th USENIX Conference on File and Storage Technologies (**FAST**), 2022

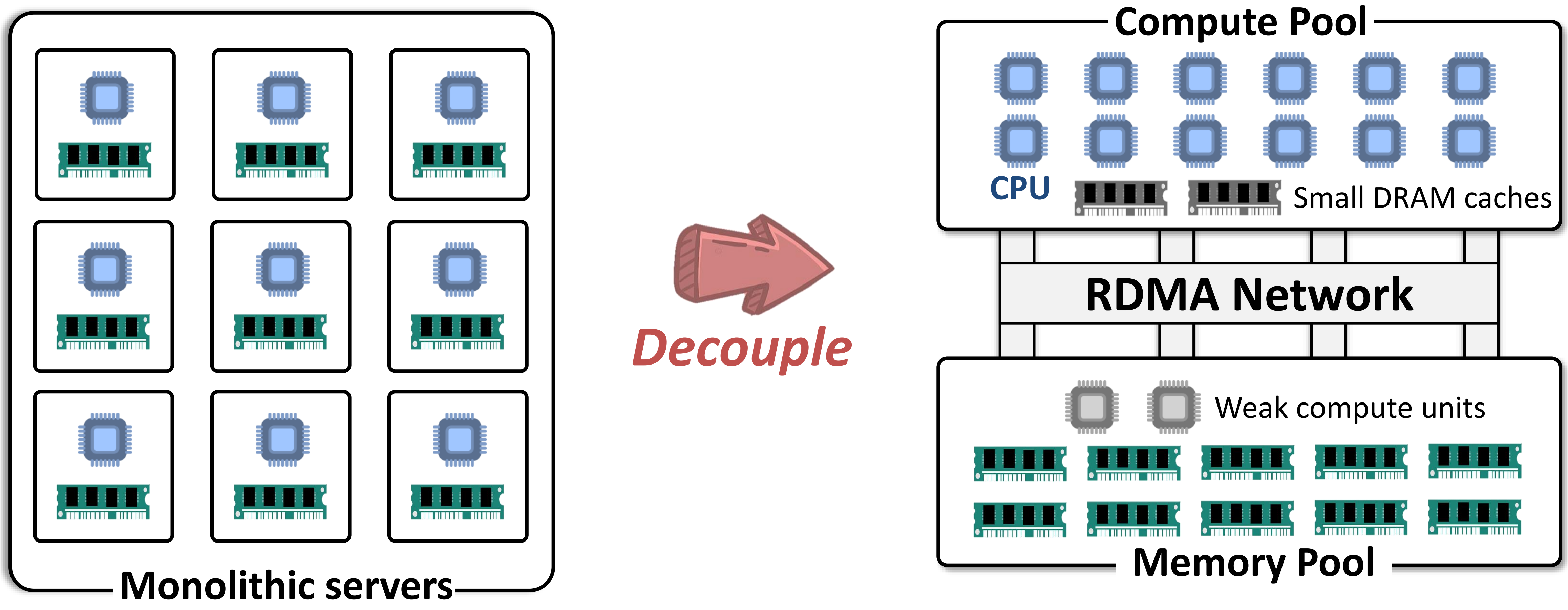
Disaggregated Persistent Memory

- Memory disaggregation



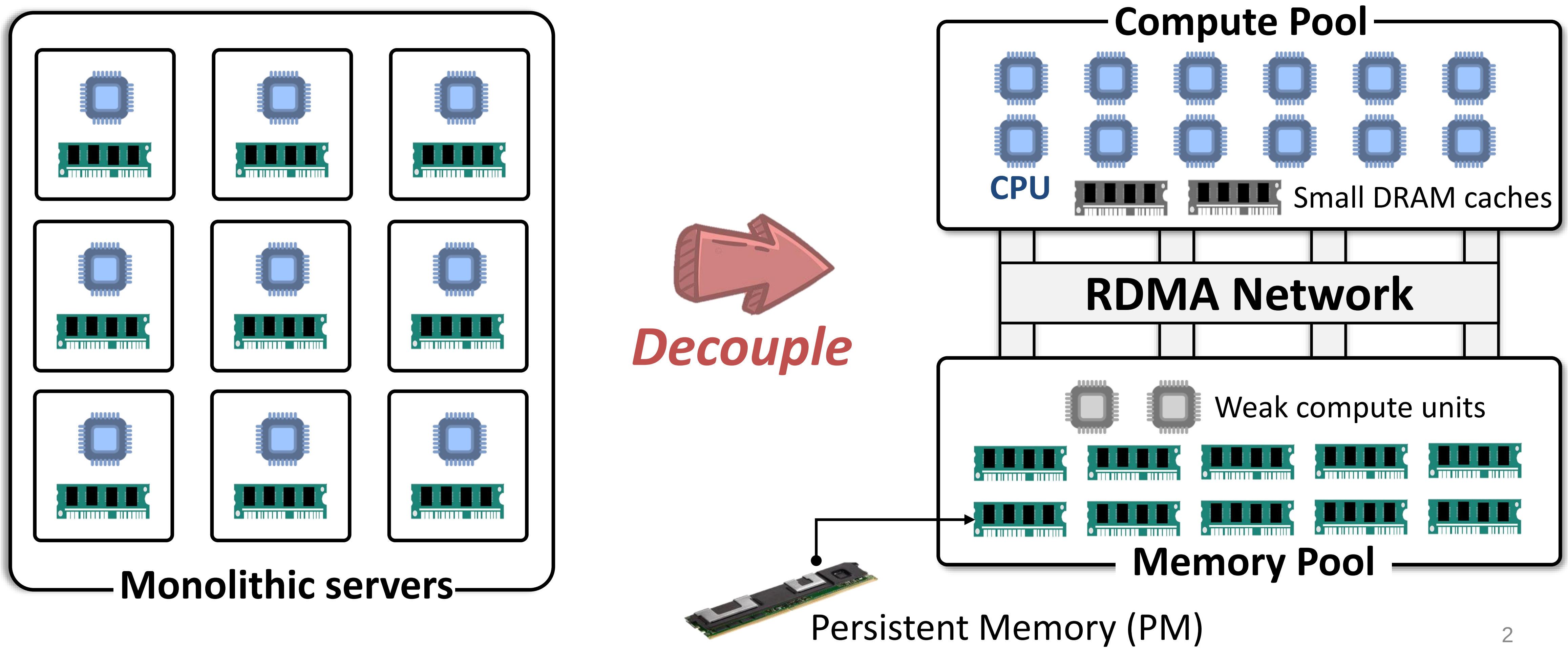
Disaggregated Persistent Memory

- Memory disaggregation

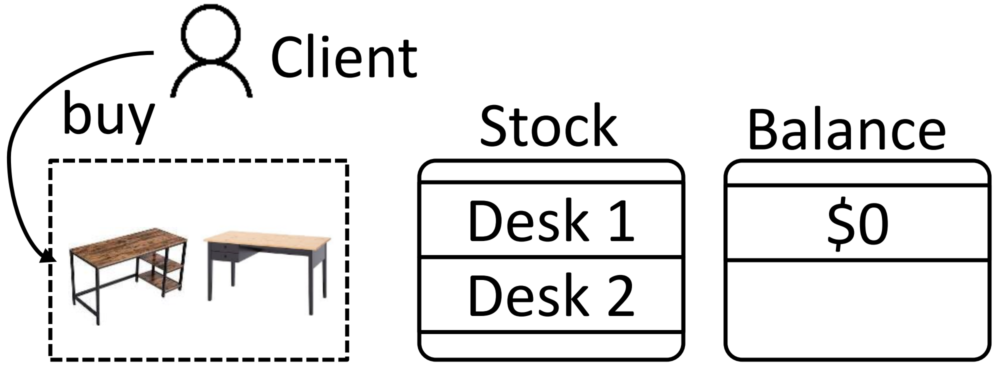


Disaggregated Persistent Memory

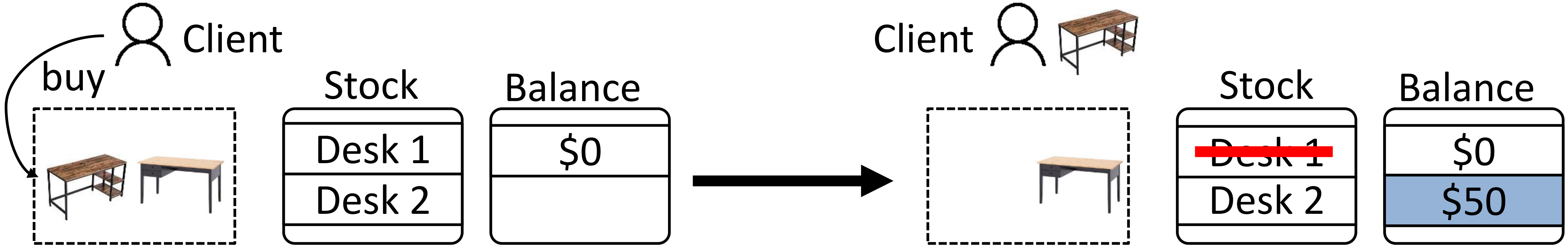
- Memory disaggregation



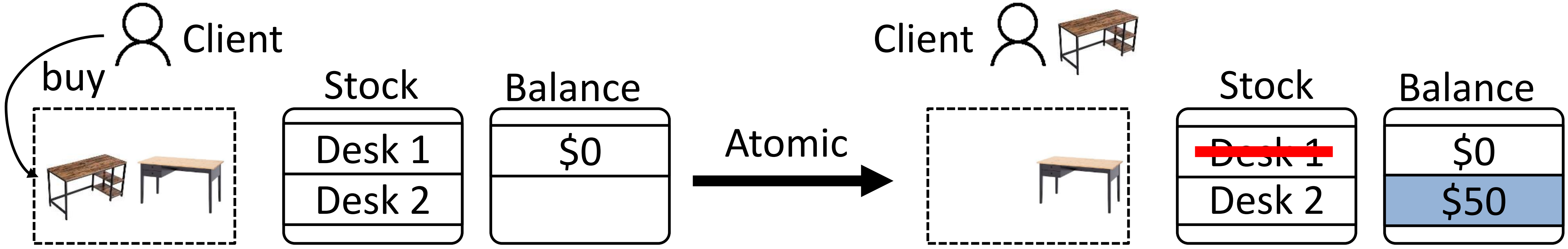
Transaction



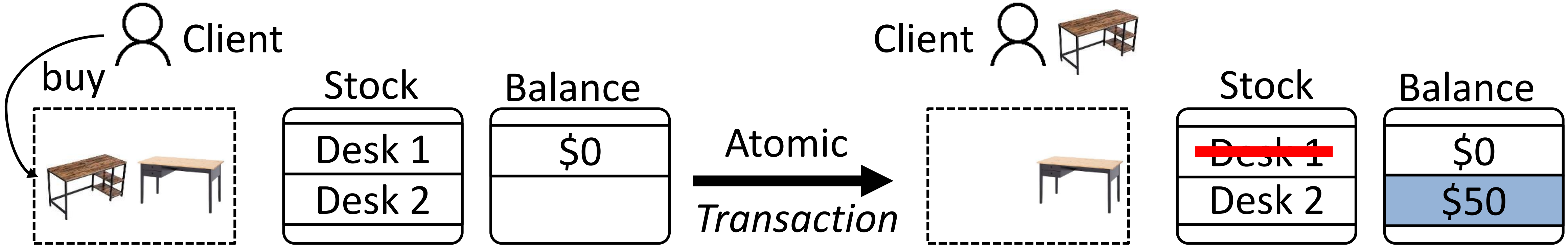
Transaction



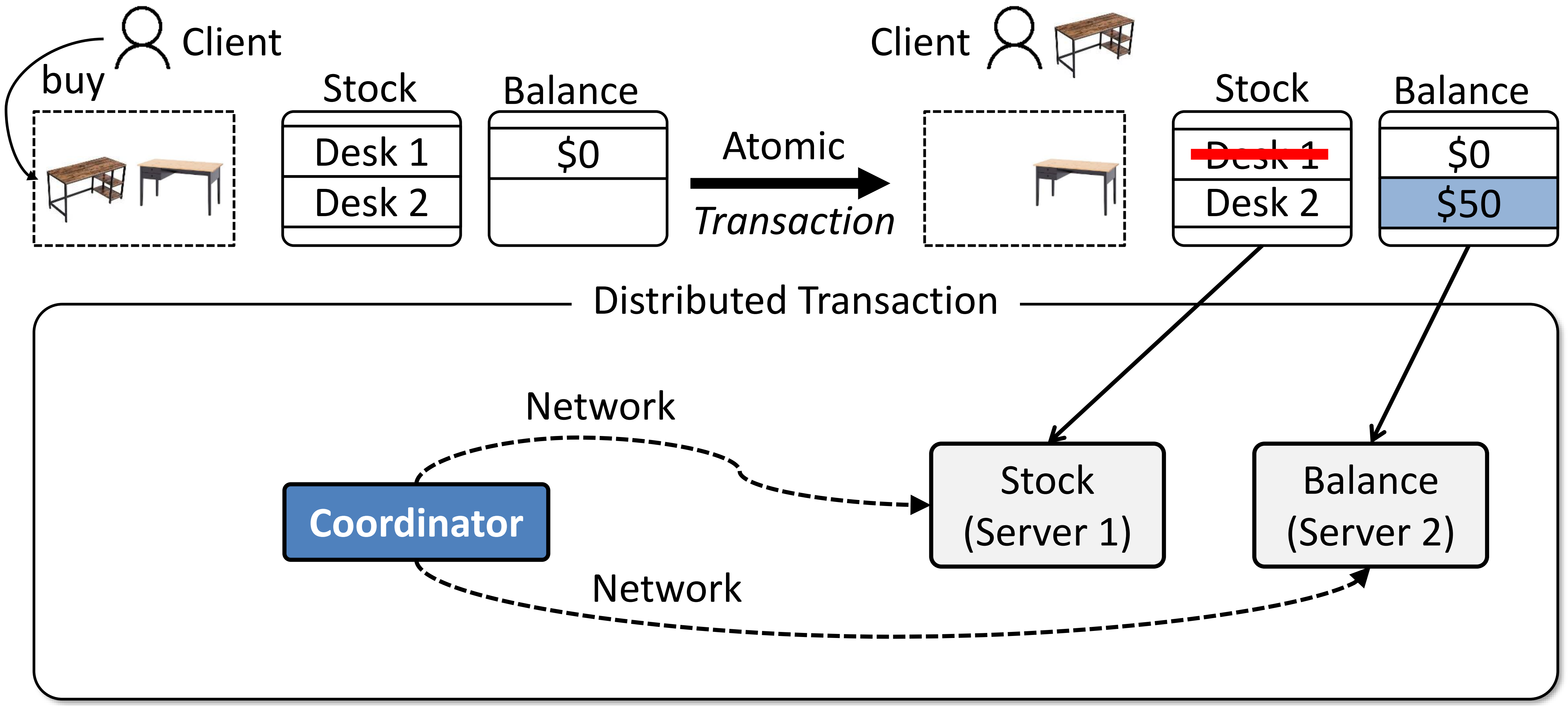
Transaction



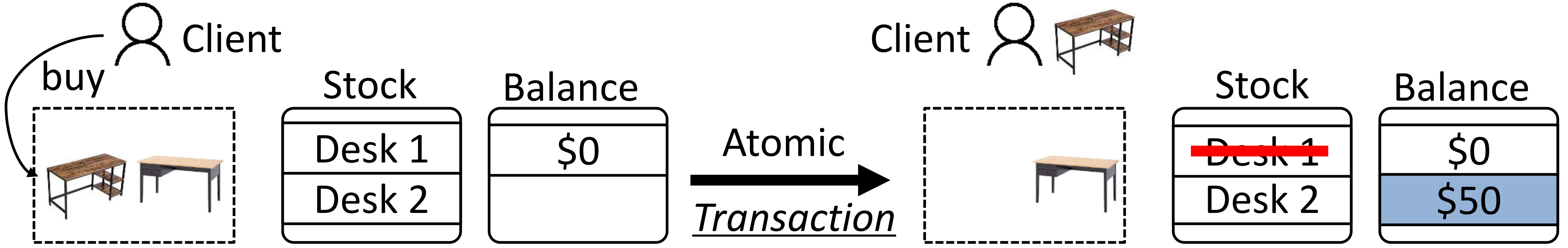
Transaction



Distributed Transaction



Distributed Transaction



Distributed Transaction: **A key building block**

Google Cloud Spanner

Amazon Aurora

Microsoft Azure

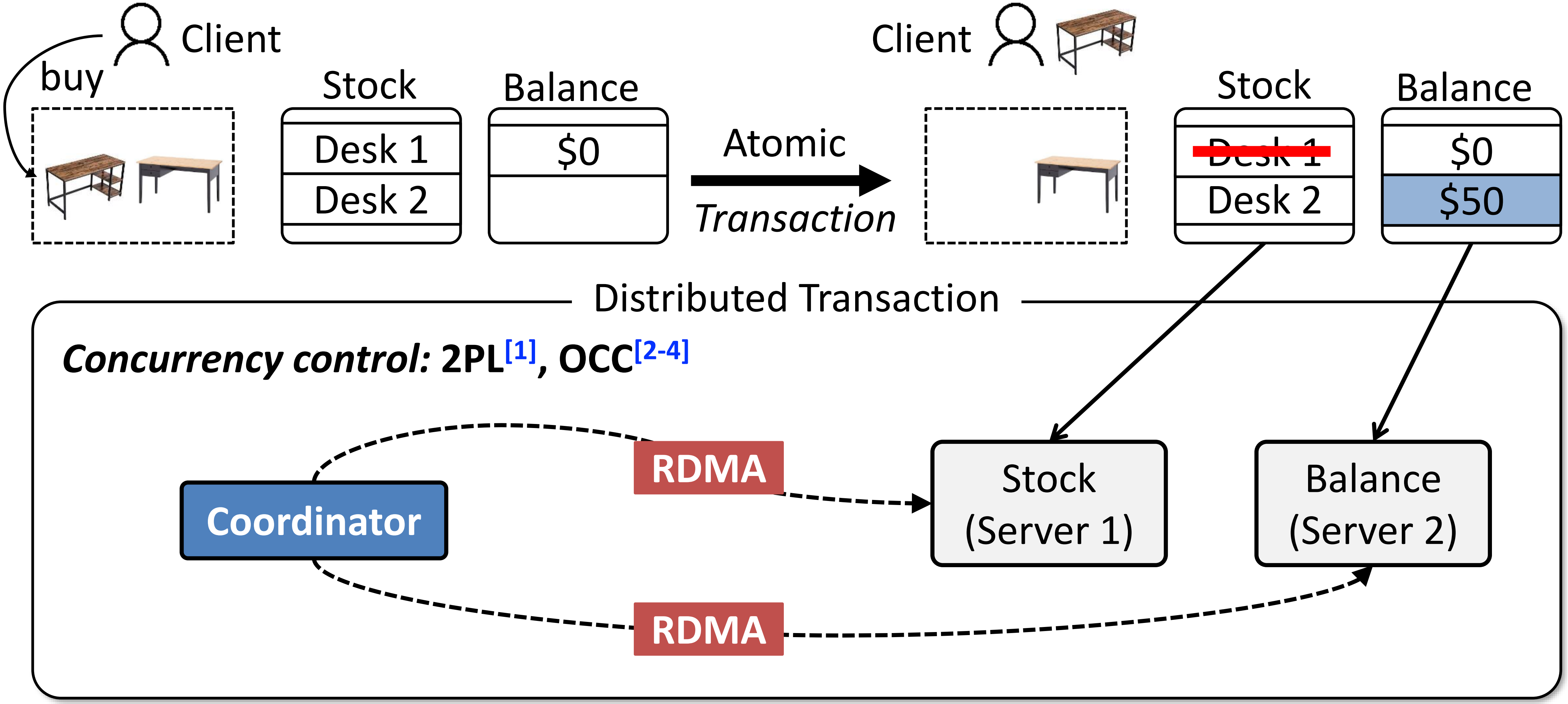
HUAWEI

HUAWEI CLOUD

Alibaba.com

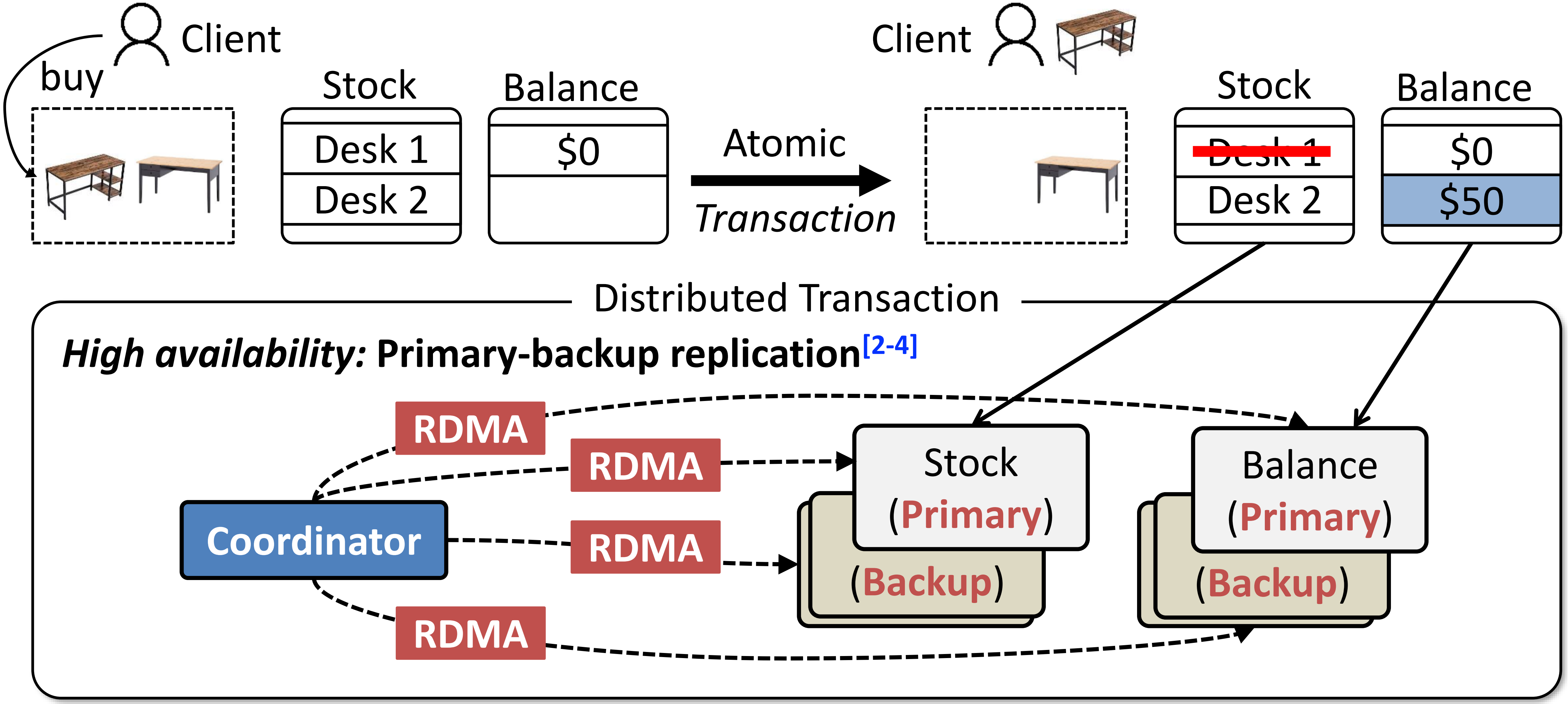
PayPal

RDMA-based Distributed Transaction



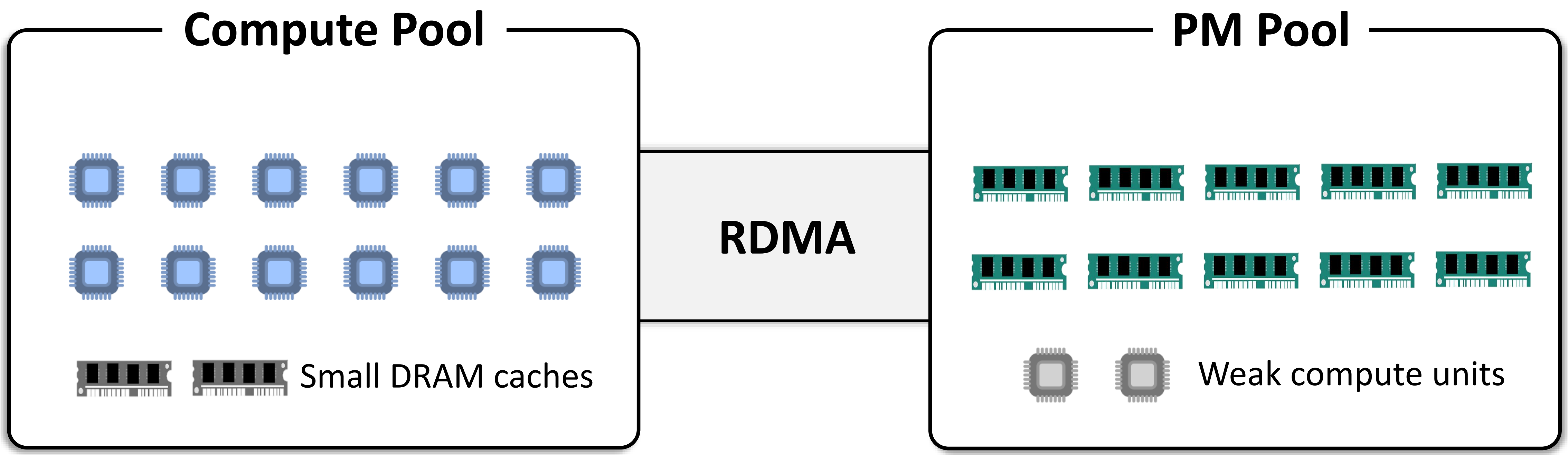
[1] DrTM@SOSP'15 [2] FaRM@SOSP'15 [3] FaSST@OSDI'16 [4] DrTM+H@OSDI'18

RDMA-based Distributed Transaction

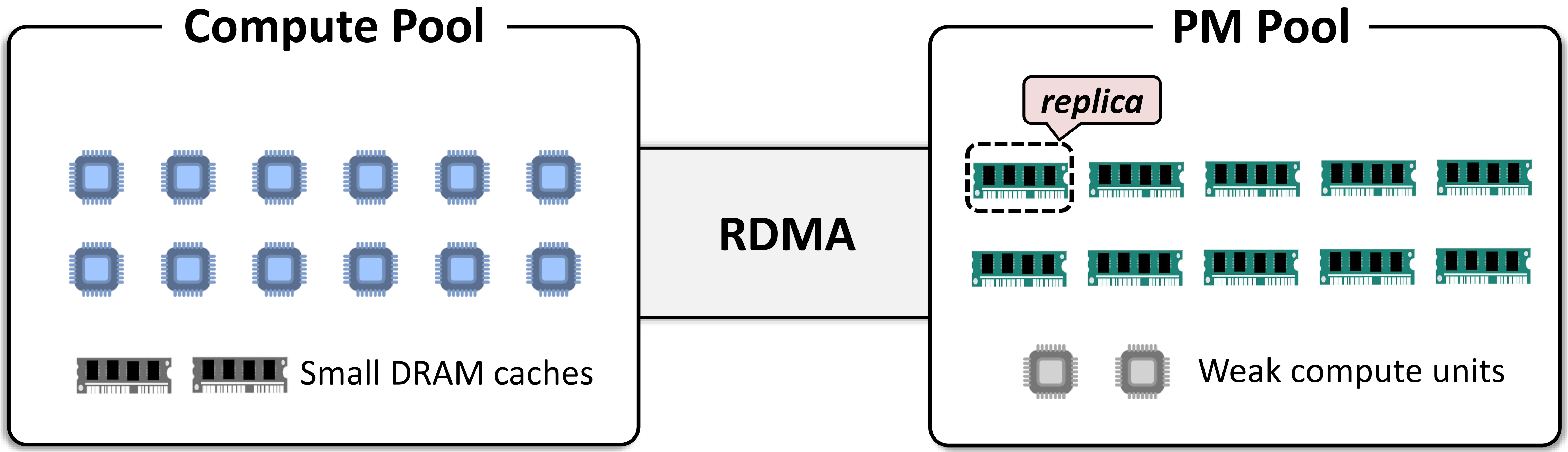


[1] DrTM@SOSP'15 [2] FaRM@SOSP'15 [3] FaSST@OSDI'16 [4] DrTM+H@OSDI'18

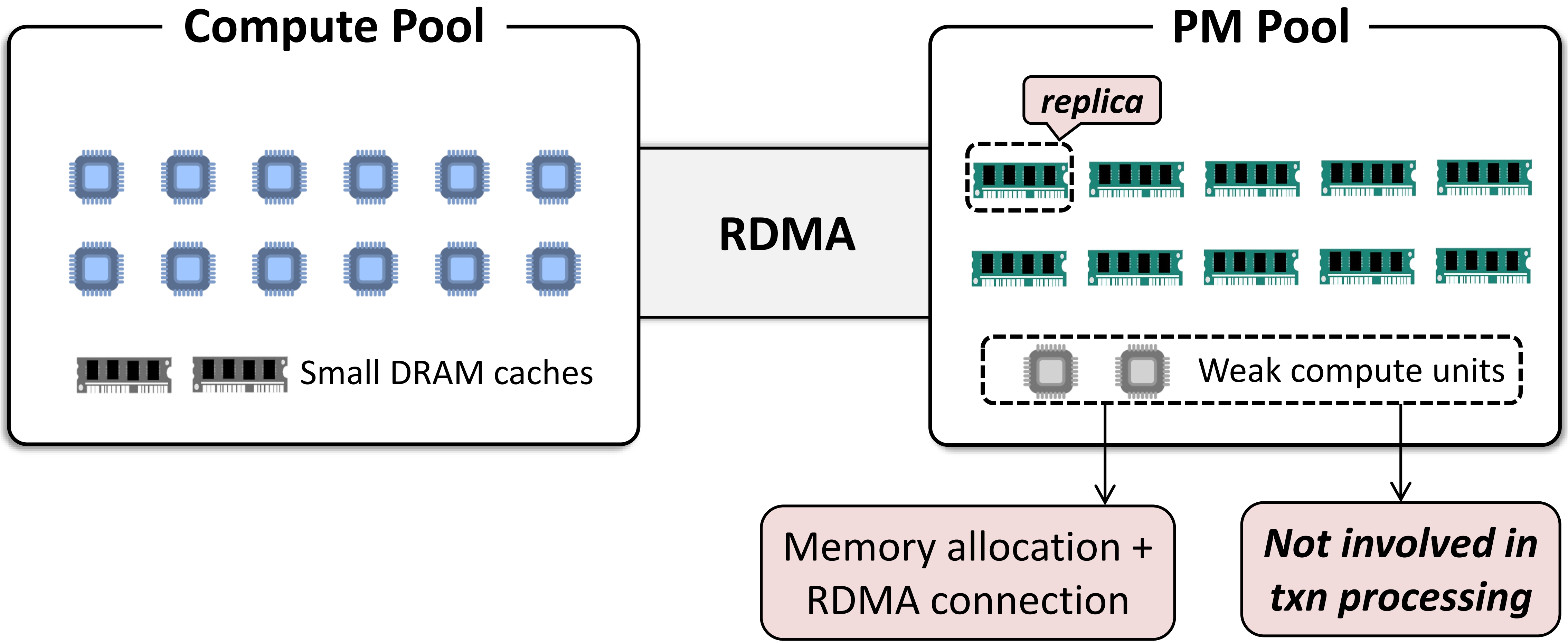
Transaction on Disaggregated PM



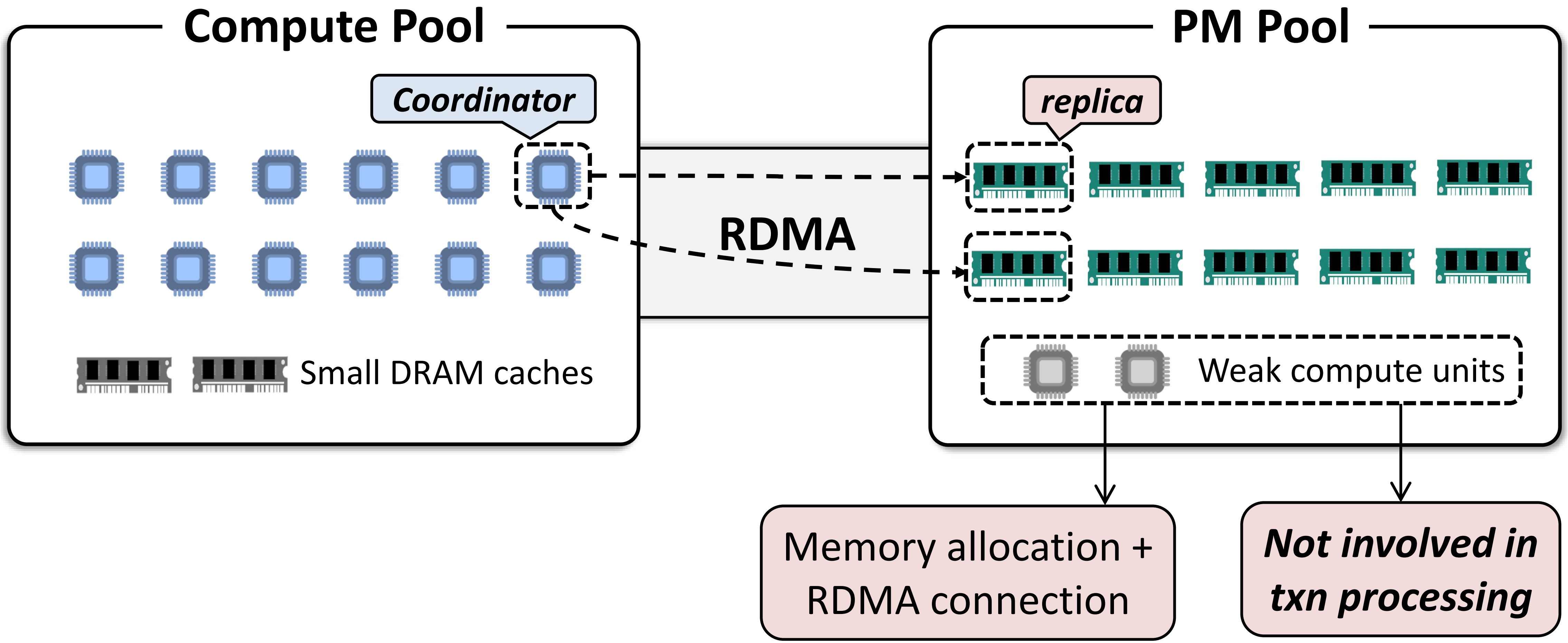
Transaction on Disaggregated PM



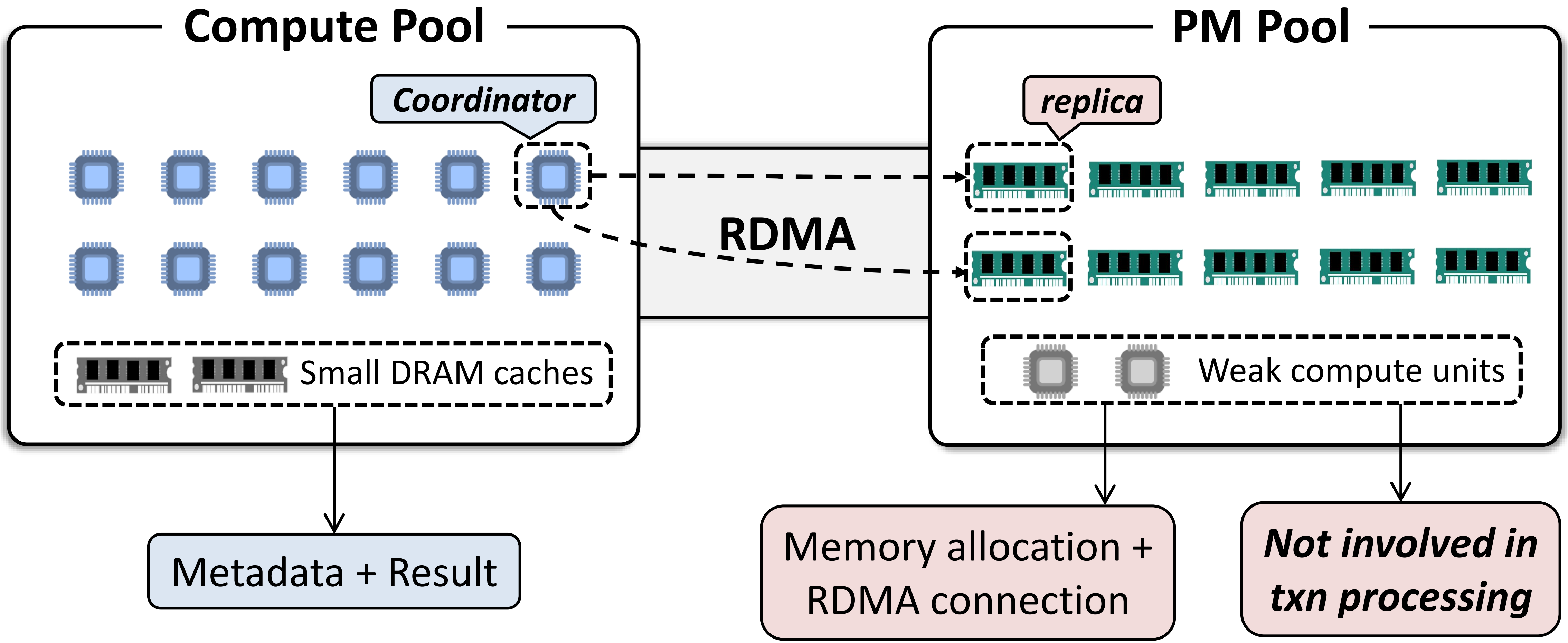
Transaction on Disaggregated PM



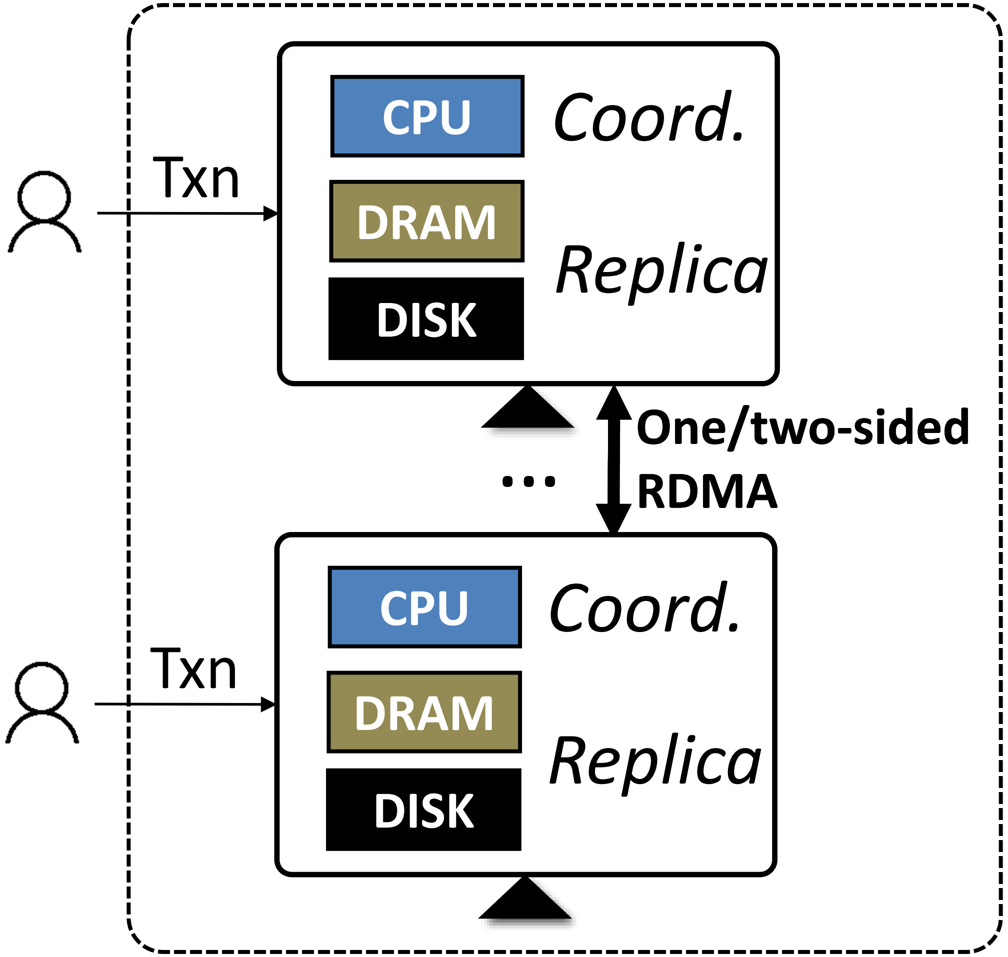
Transaction on Disaggregated PM



Transaction on Disaggregated PM



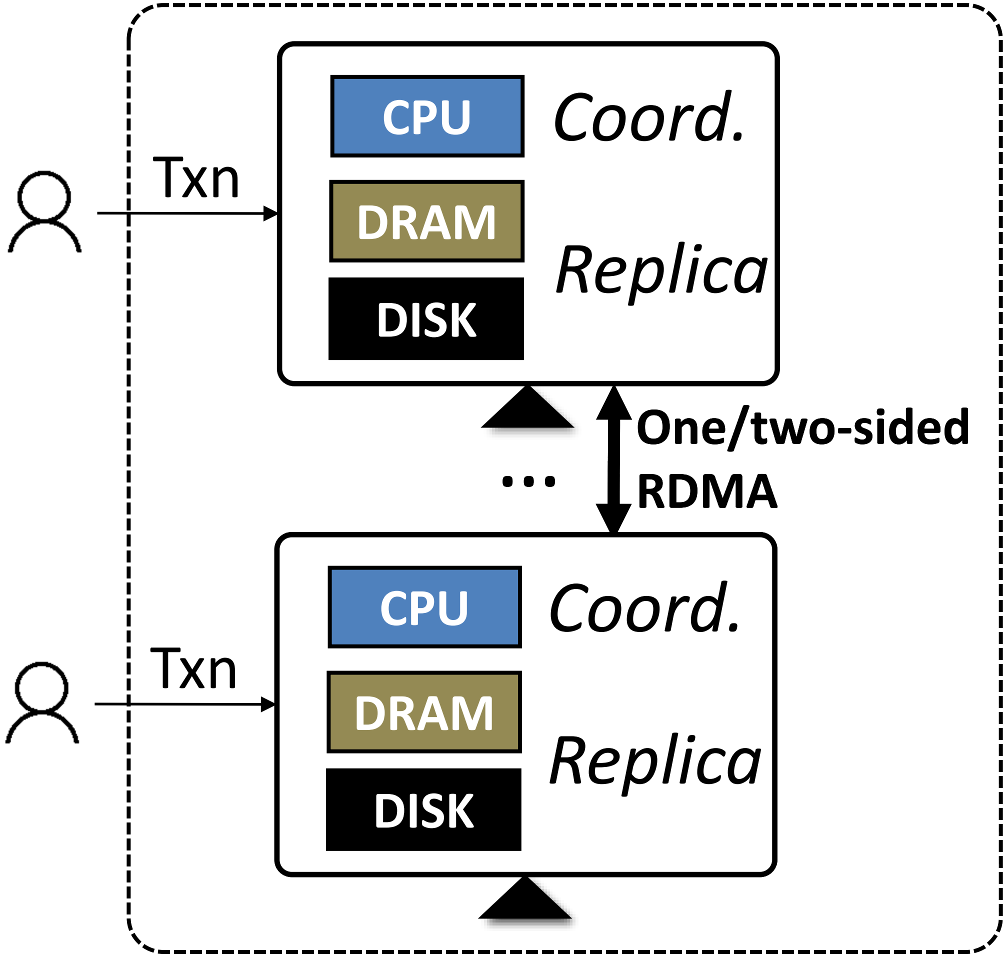
State-of-the-art



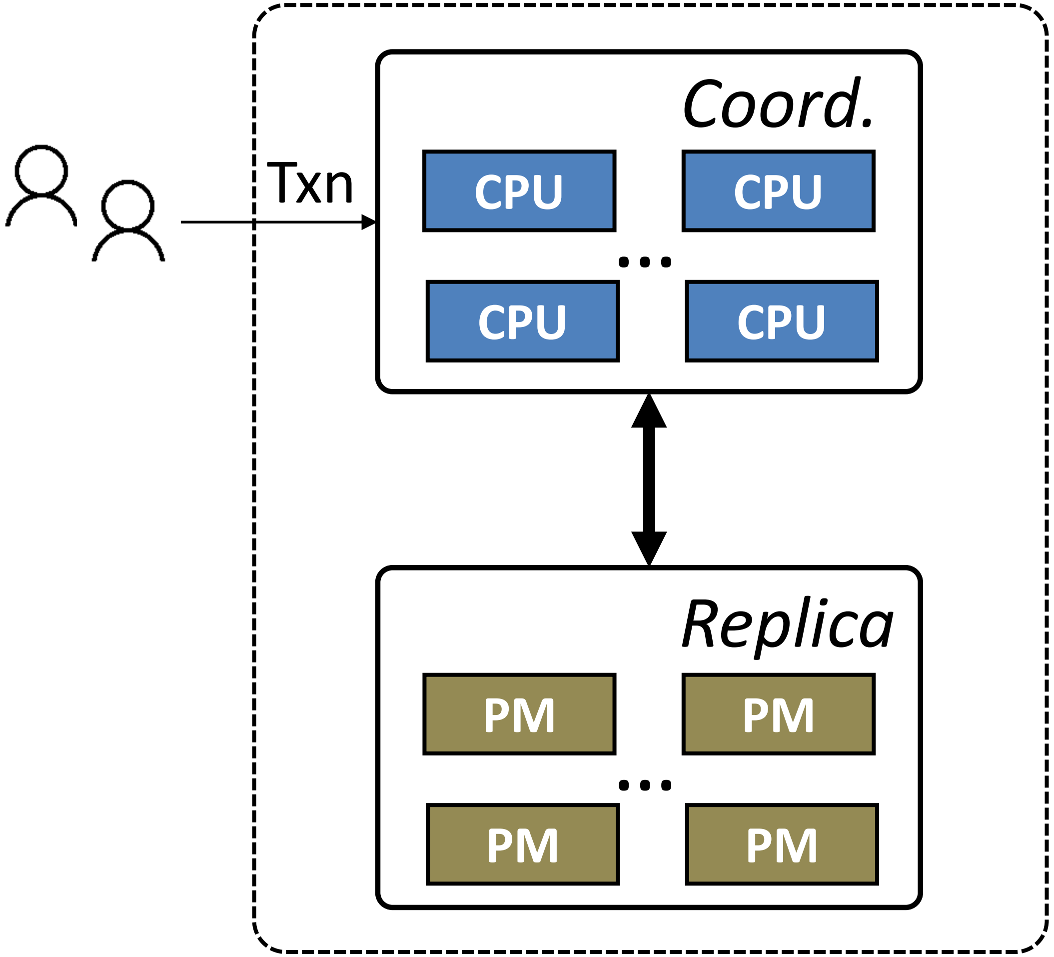
Monolithic architecture[1-4]

[1] DrTM@SOSP'15 [2] FaRM@SOSP'15 [3] FaSST@OSDI'16 [4] DrTM+H@OSDI'18

State-of-the-art



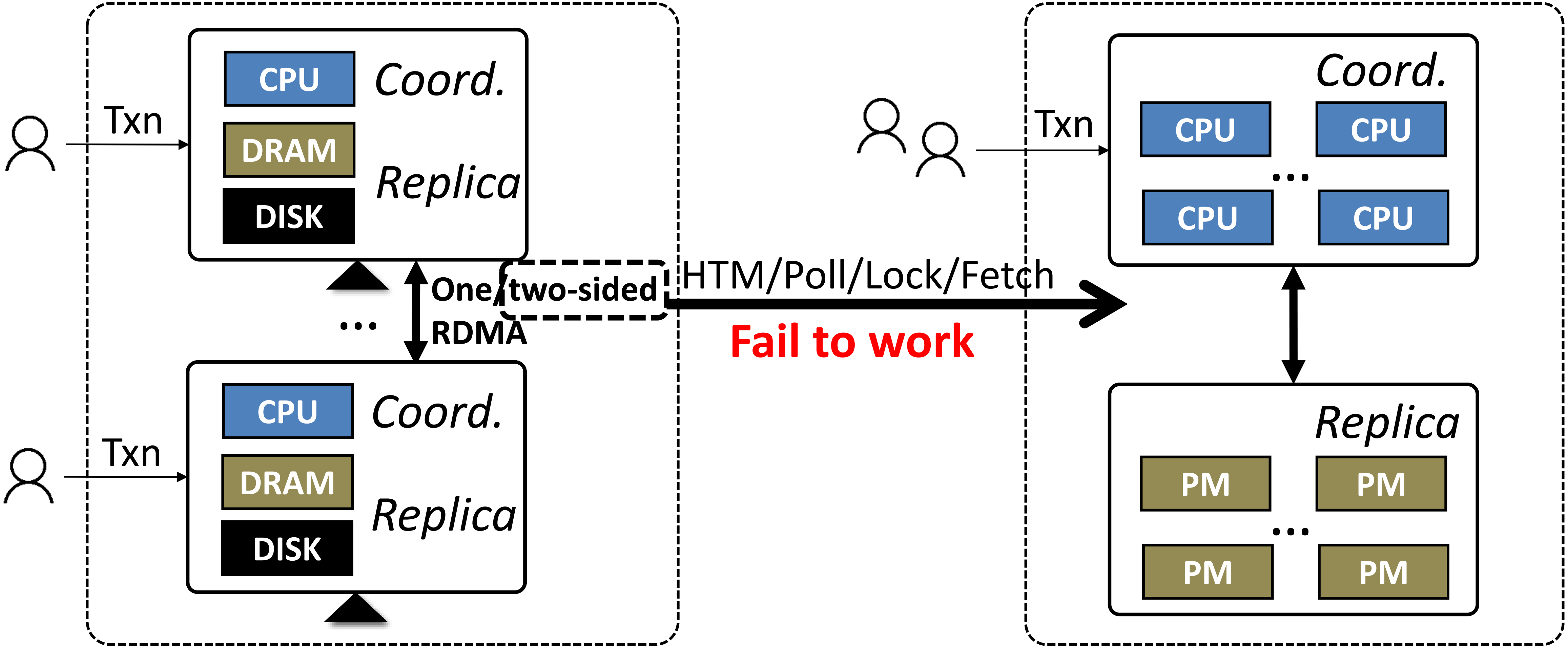
Monolithic architecture[1-4]



Disaggregated architecture

[1] DrTM@SOSP'15 [2] FaRM@SOSP'15 [3] FaSST@OSDI'16 [4] DrTM+H@OSDI'18

State-of-the-art

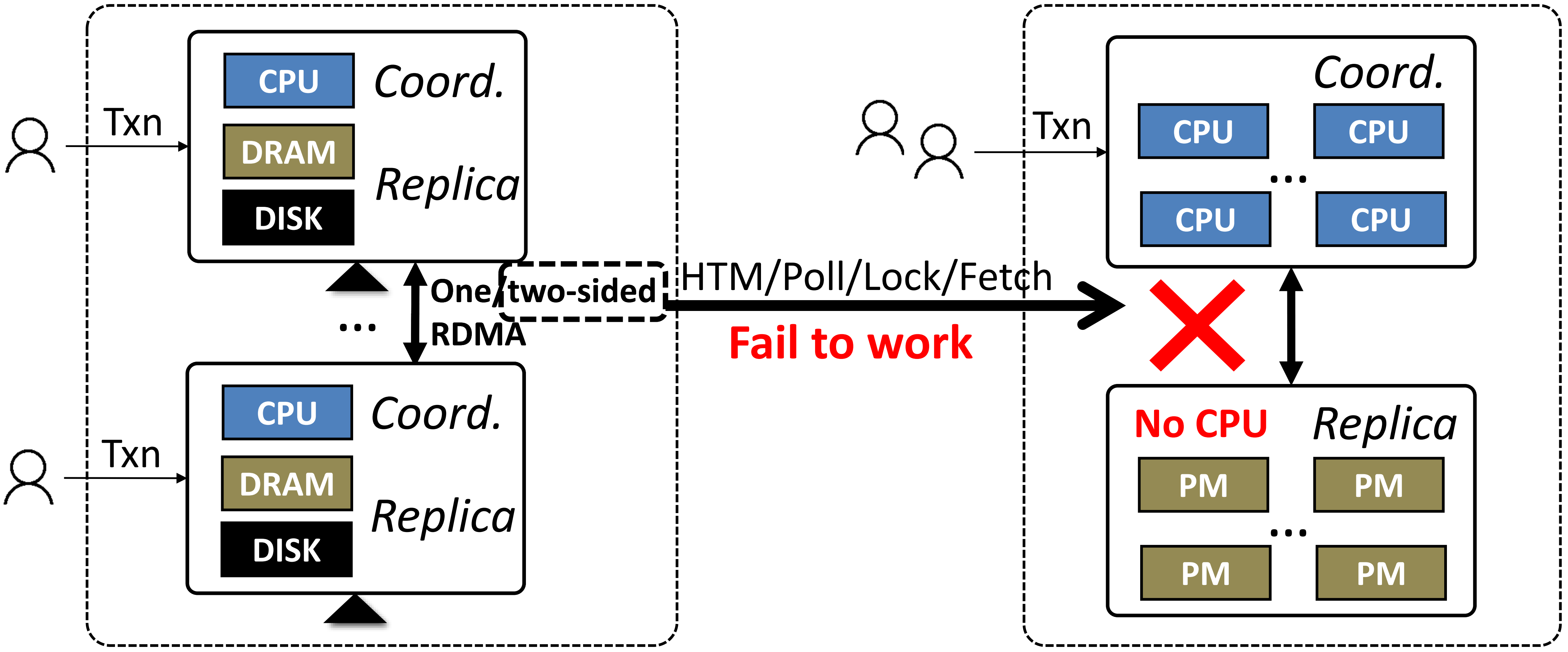


Monolithic architecture[1-4]

Disaggregated architecture

[1] DrTM@SOSP'15 [2] FaRM@SOSP'15 [3] FaSST@OSDI'16 [4] DrTM+H@OSDI'18

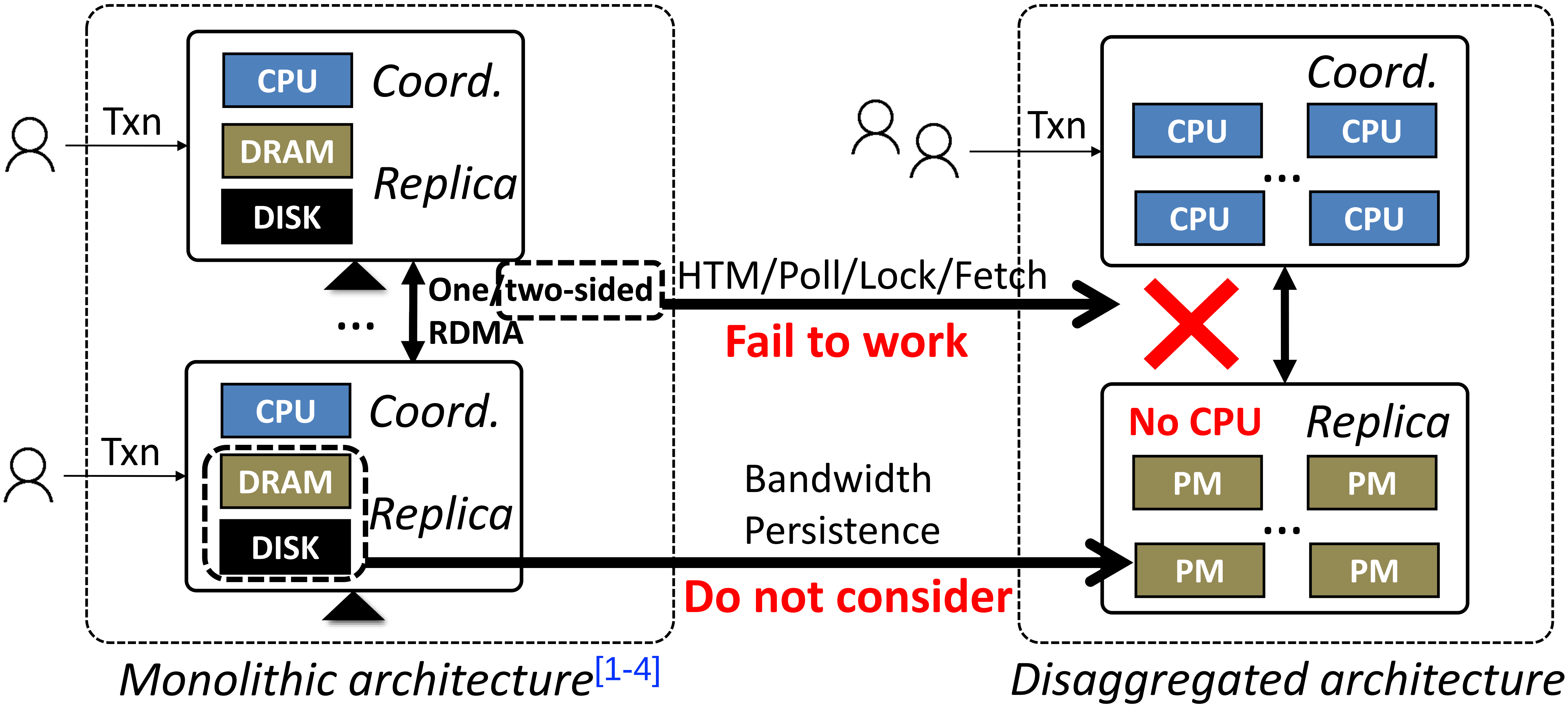
State-of-the-art



Monolithic architecture[1-4] *Disaggregated architecture*

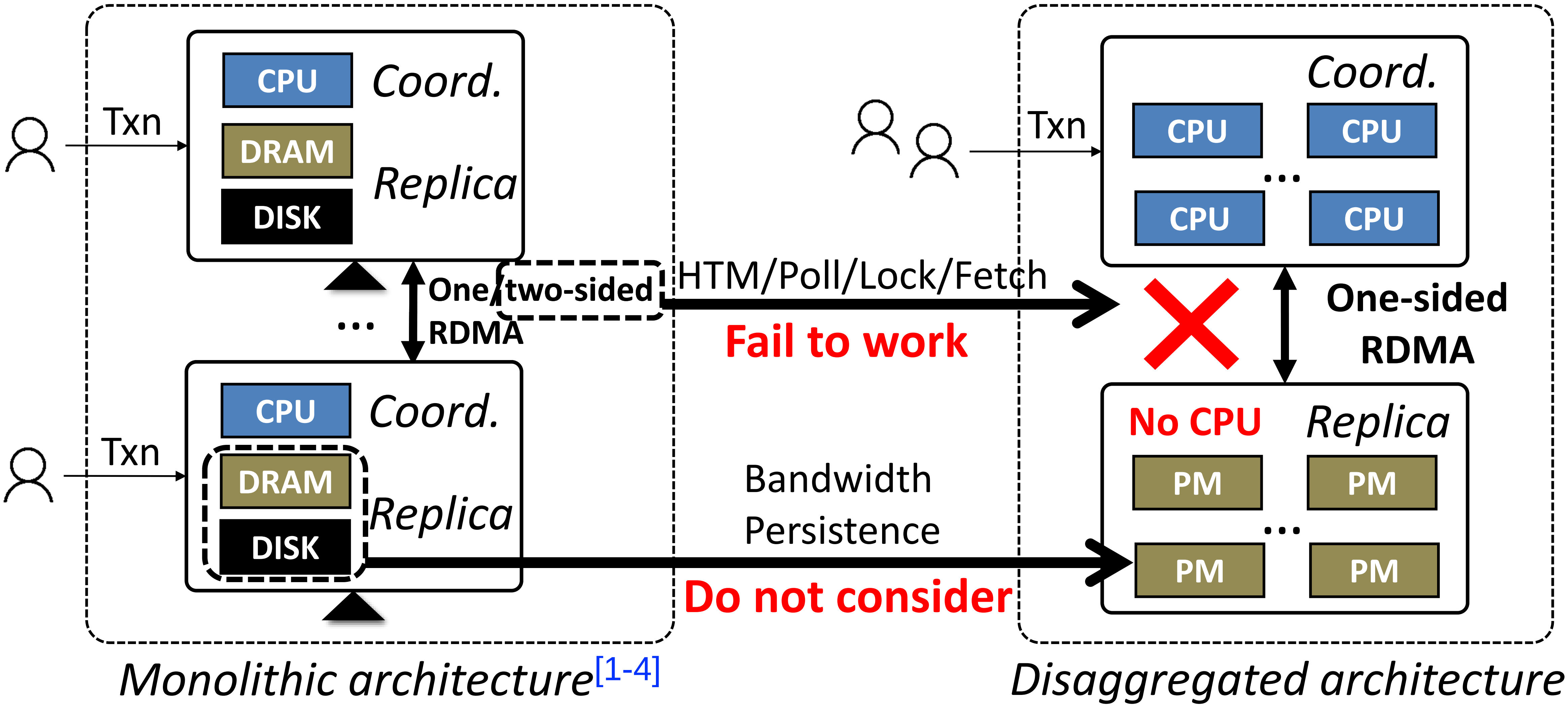
[1] DrTM@SOSP'15 [2] FaRM@SOSP'15 [3] FaSST@OSDI'16 [4] DrTM+H@OSDI'18

State-of-the-art



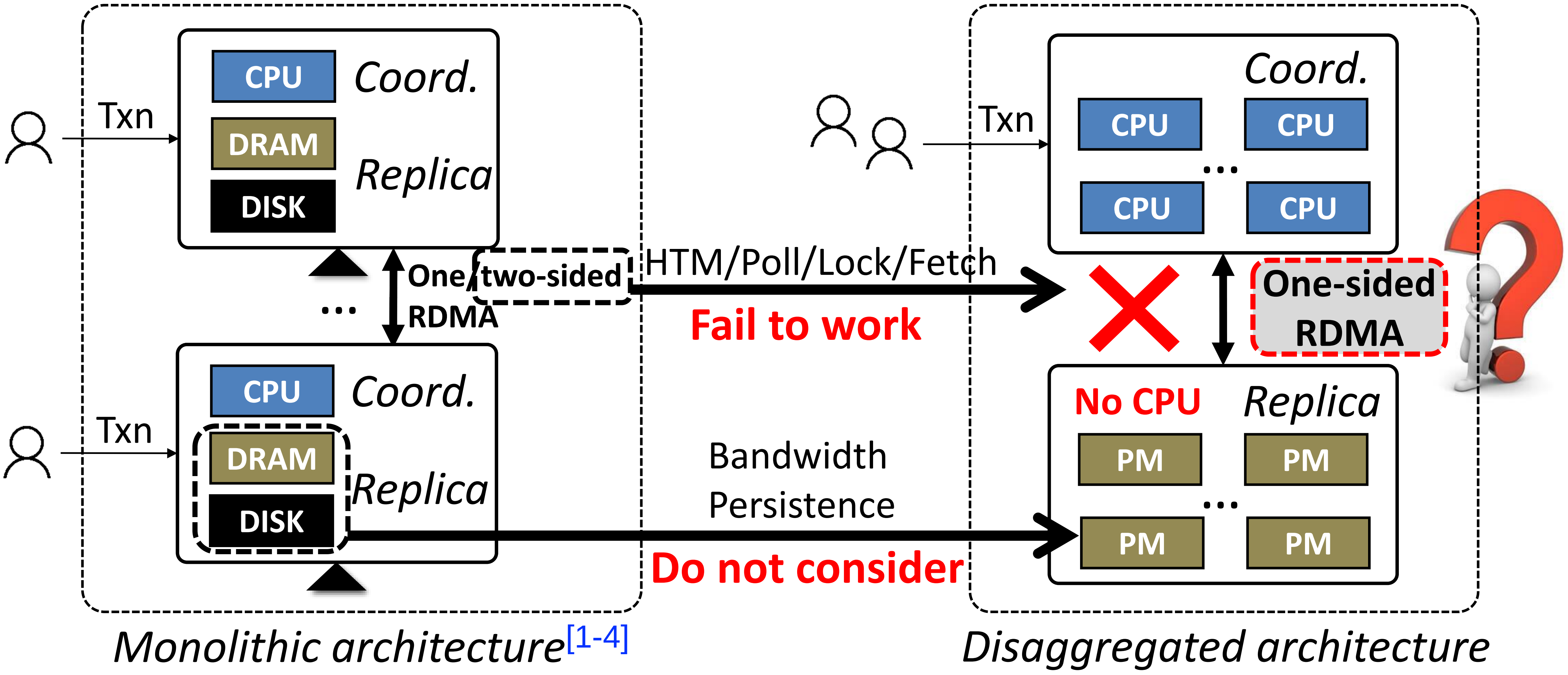
[1] DrTM@SOSP'15 [2] FaRM@SOSP'15 [3] FaSST@OSDI'16 [4] DrTM+H@OSDI'18

State-of-the-art



[1] DrTM@SOSP'15 [2] FaRM@SOSP'15 [3] FaSST@OSDI'16 [4] DrTM+H@OSDI'18

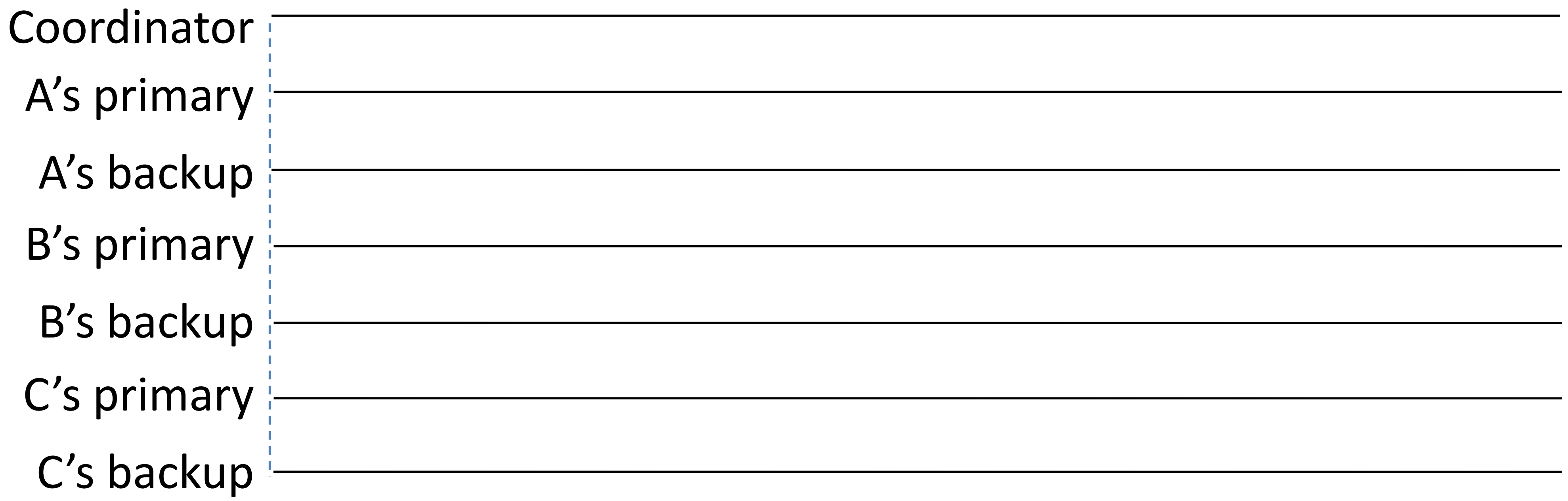
State-of-the-art



[1] DrTM@SOSP'15 [2] FaRM@SOSP'15 [3] FaSST@OSDI'16 [4] DrTM+H@OSDI'18

Challenge 1

➢ **<u>Long-latency processing:</u>** *Many round trips*



Txn begin
Read A,B,C
Write A=A+C
Write B=B-C
Txn commit

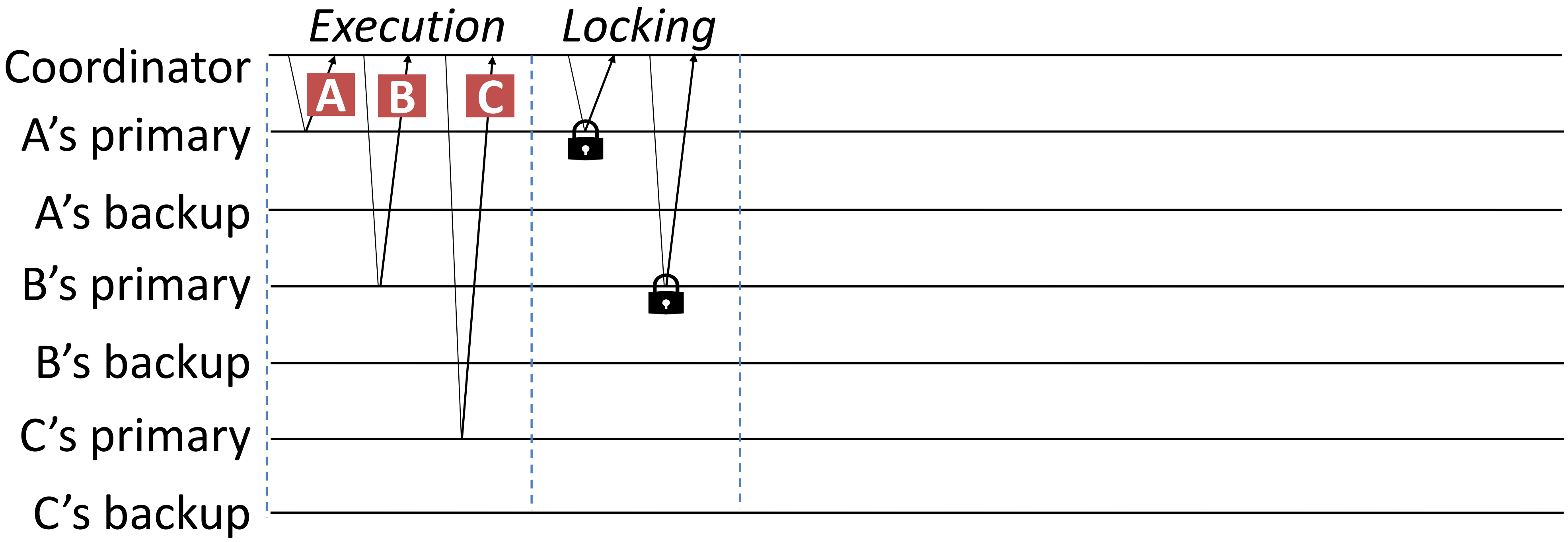
Challenge 1

➢ **Long-latency processing:** *Many round trips*

Txn begin
Read A,B,C
Write A=A+C
Write B=B-C
Txn commit

Challenge 1

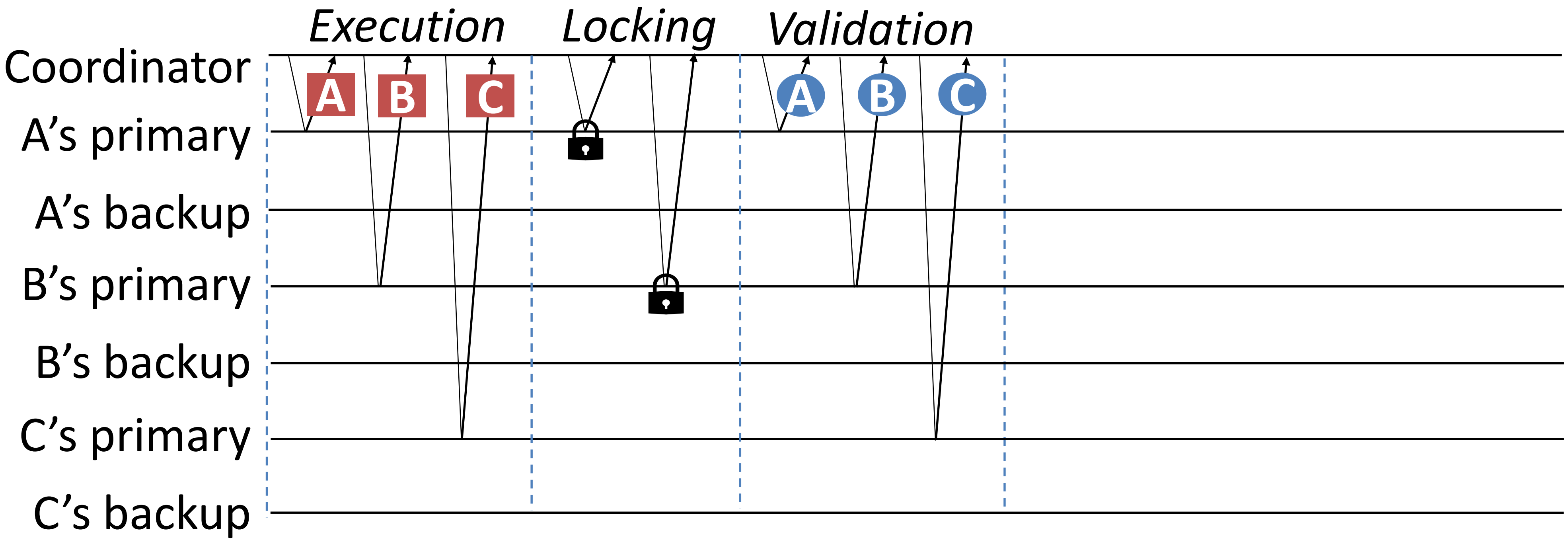
➢ **<u>Long-latency processing:</u>** *Many round trips*



Txn begin
Read A,B,C
Write A=A+C
Write B=B-C
Txn commit

Challenge 1

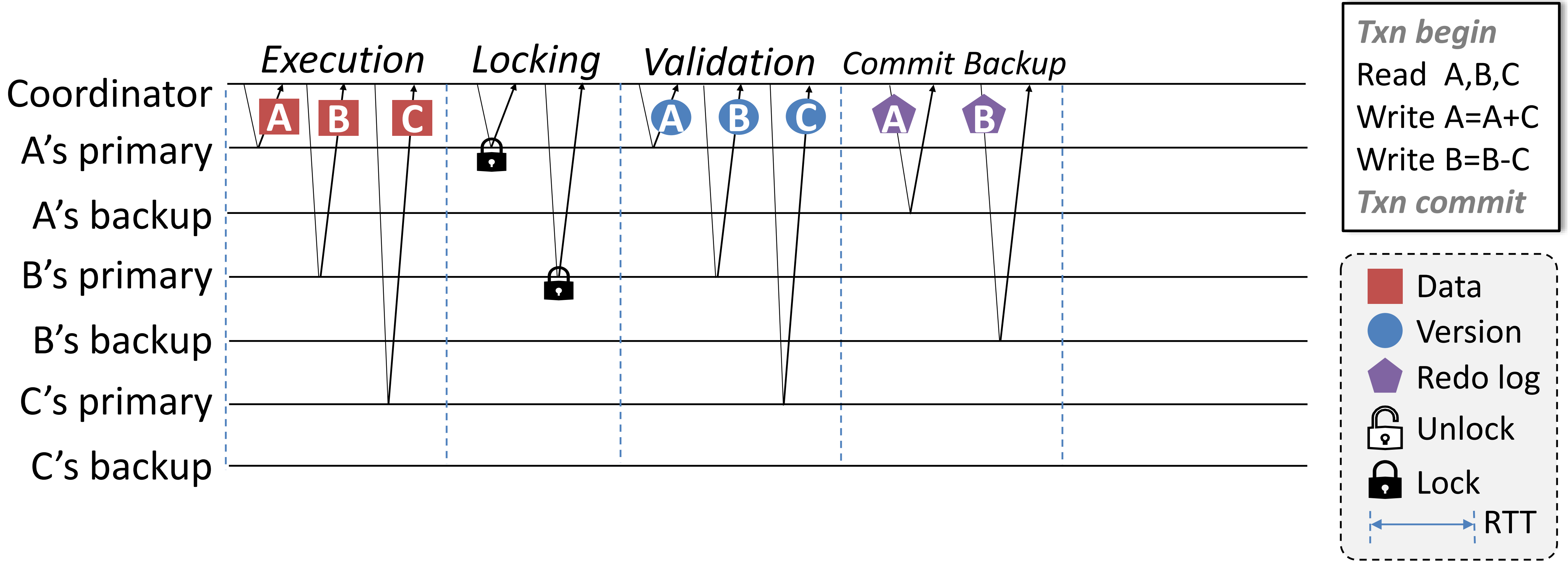
➢ **<u>Long-latency processing:</u>** *Many round trips*



Txn begin
Read A,B,C
Write A=A+C
Write B=B-C
Txn commit

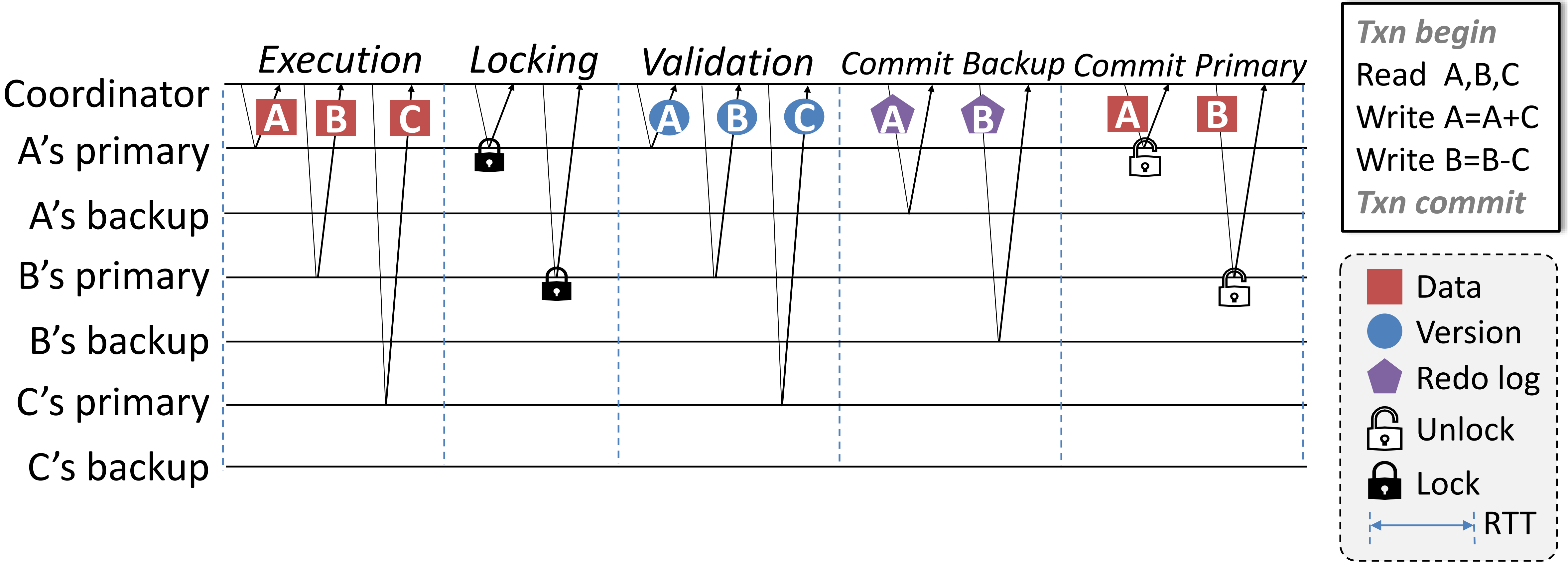
Challenge 1

➢ **Long-latency processing:** *Many round trips*



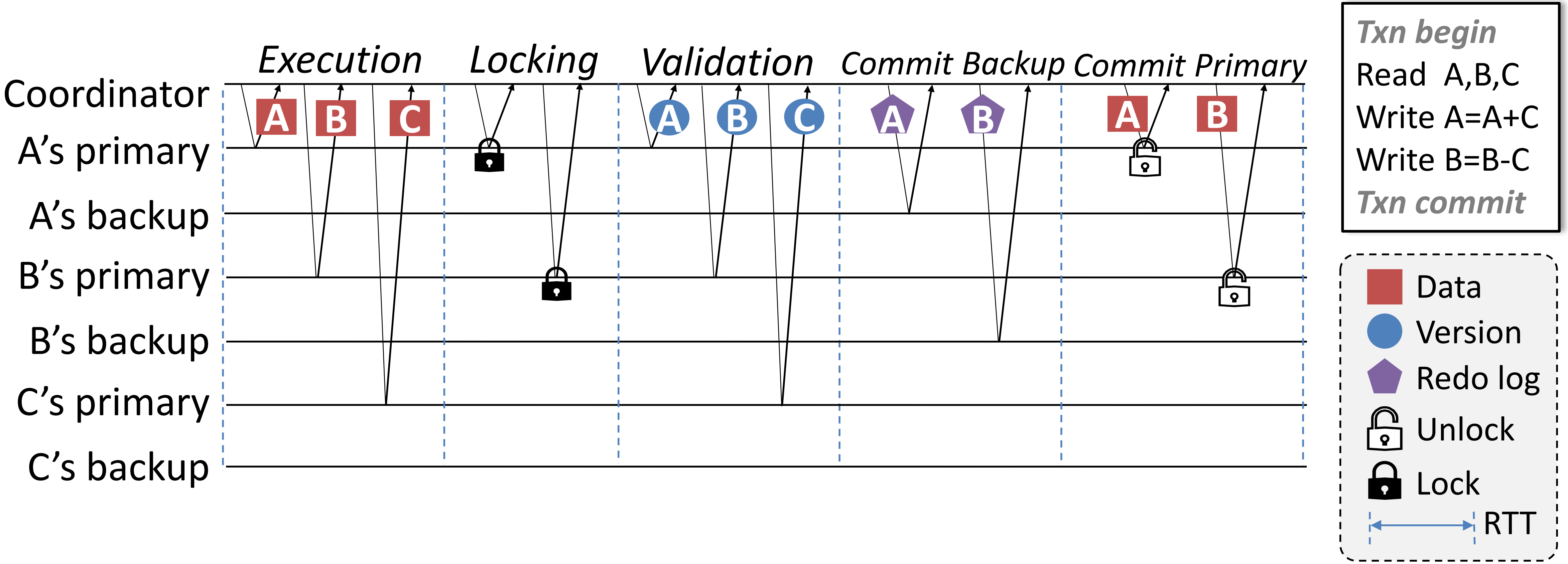
Challenge 1

➢ **Long-latency processing:** *Many round trips*



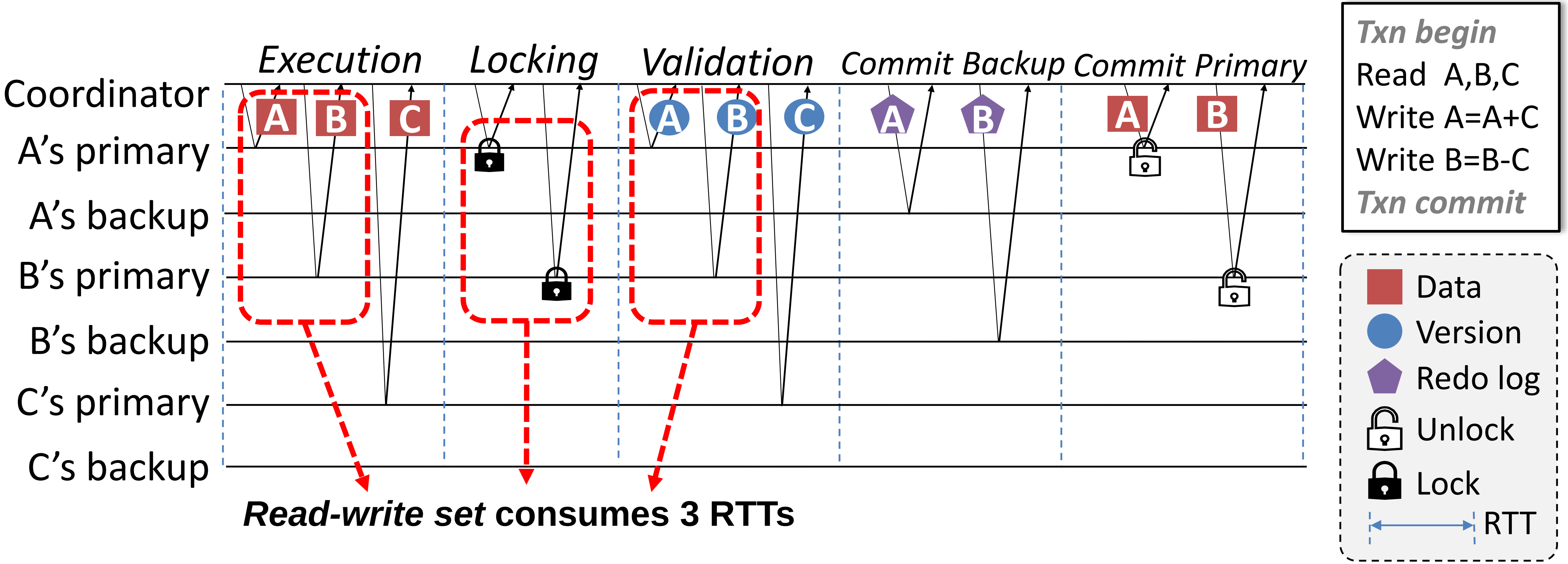
Challenge 1

➢ **<u>Long-latency processing:</u>** *Many round trips*



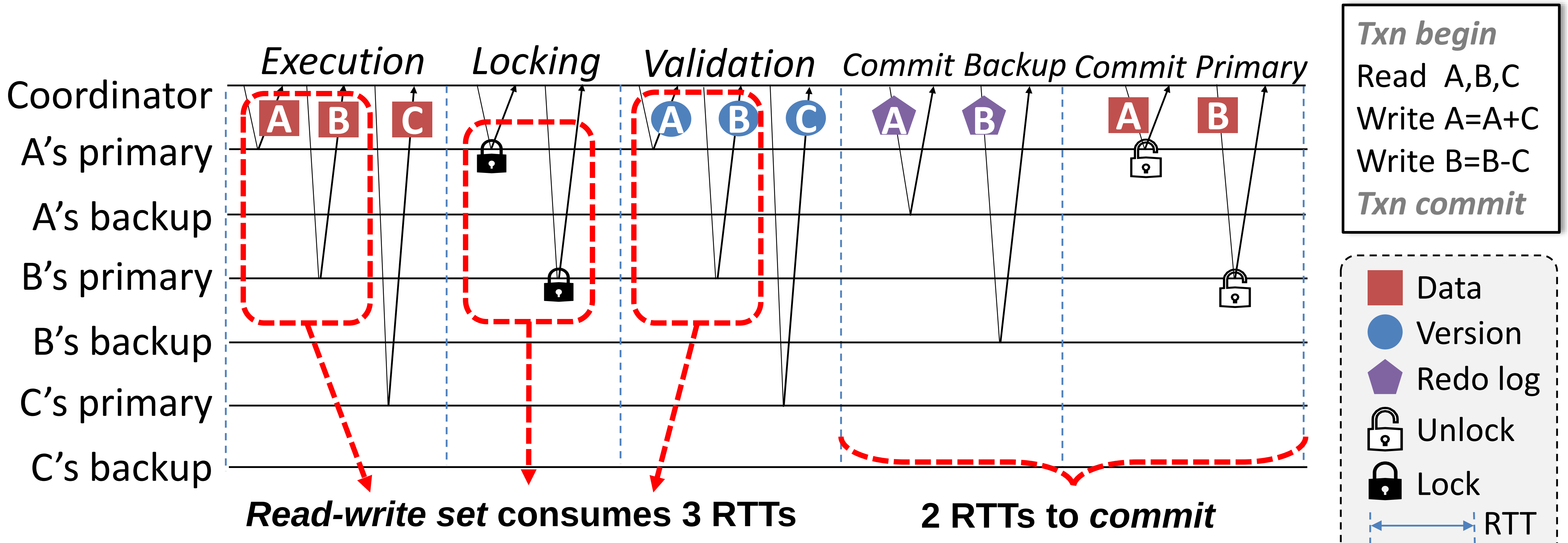
Challenge 1

➢ **<u>Long-latency processing:</u>** *Many round trips*



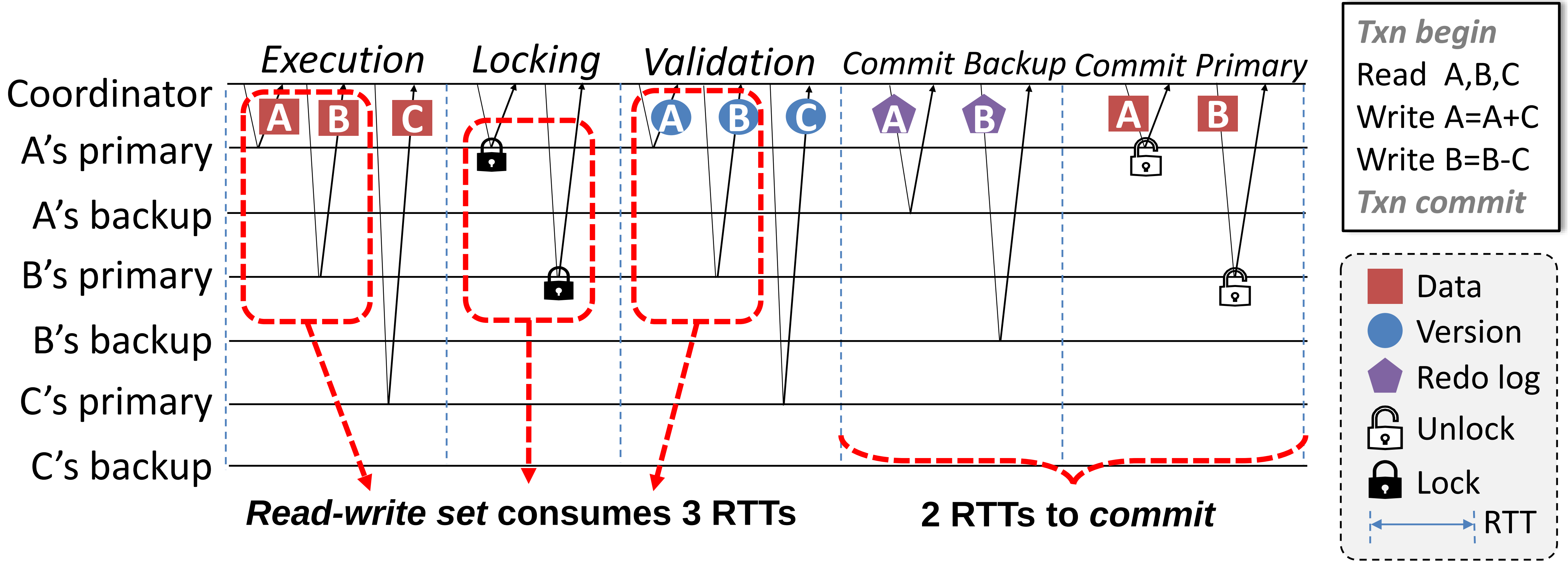
Challenge 1

➢ **<u>Long-latency processing:</u>** *Many round trips*



Challenge 1

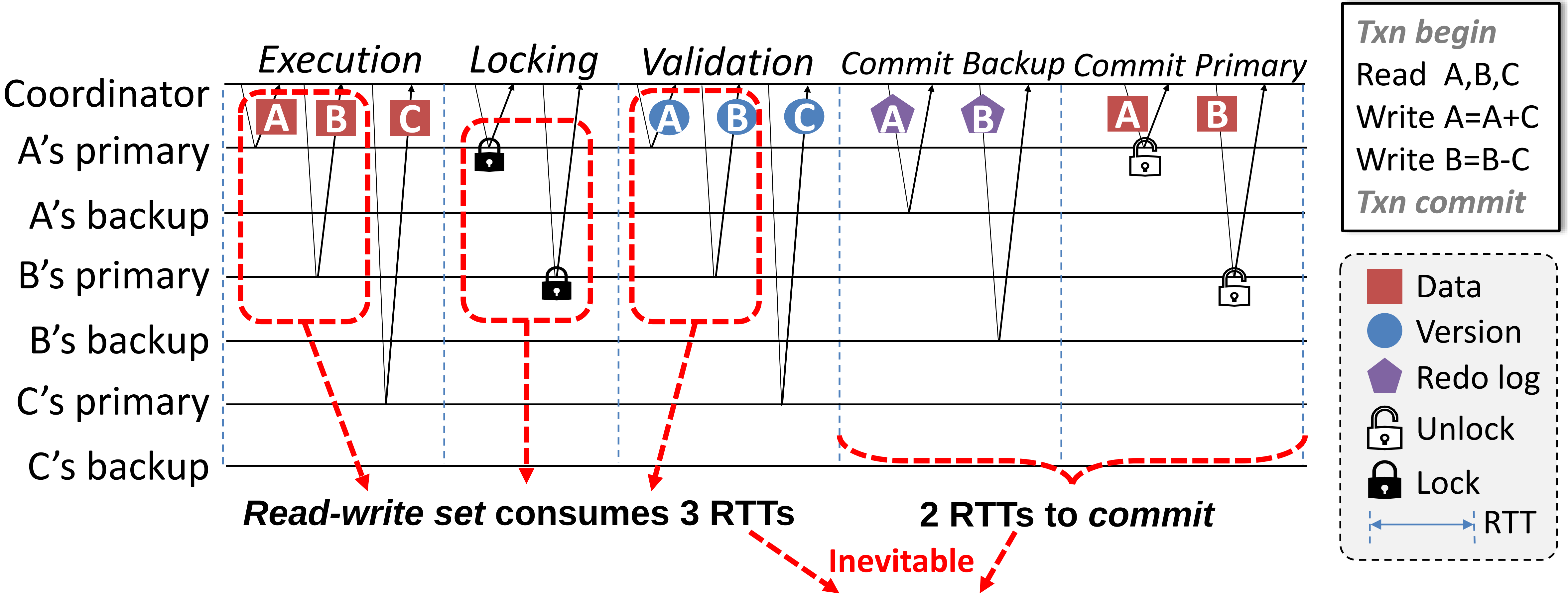
➢ **<u>Long-latency processing:</u>** *Many round trips*



Coordinators and replicas are separated: **All transactions are distributed**

Challenge 1

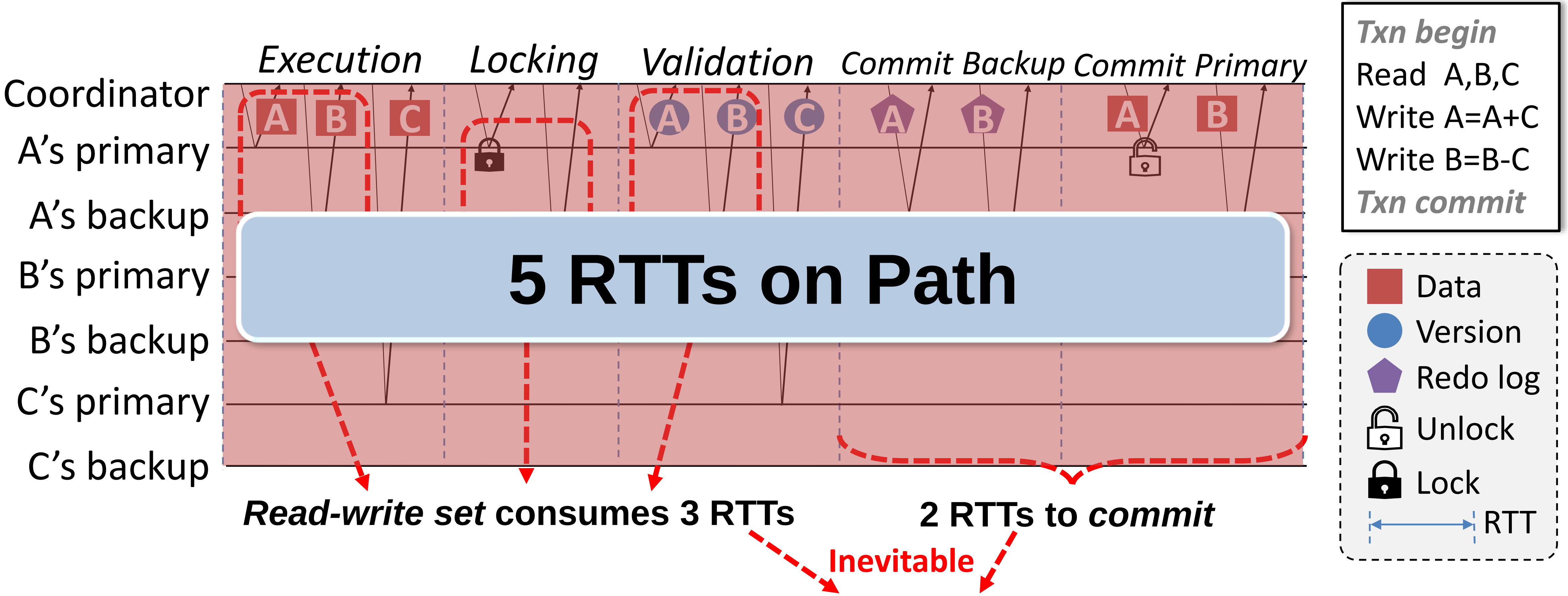
➢ <u>**Long-latency processing:**</u> *Many round trips*



Coordinators and replicas are separated: **All transactions are distributed**

Challenge 1

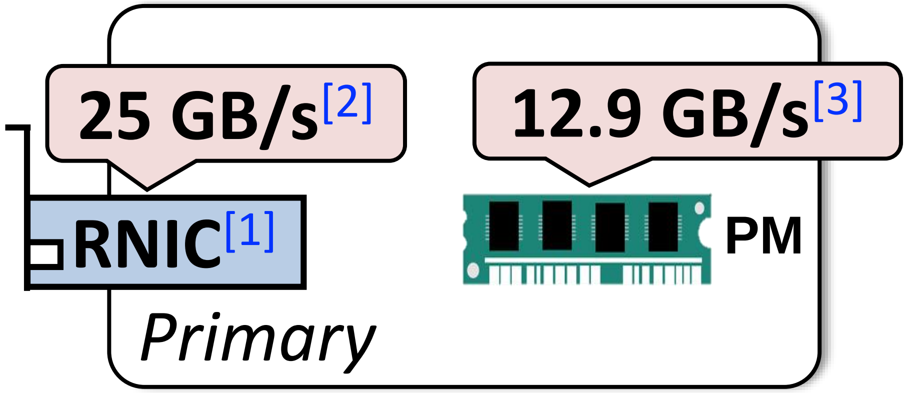
➢ **<u>Long-latency processing:</u>** *Many round trips*



Coordinators and replicas are separated: **All transactions are distributed**

Challenge 2

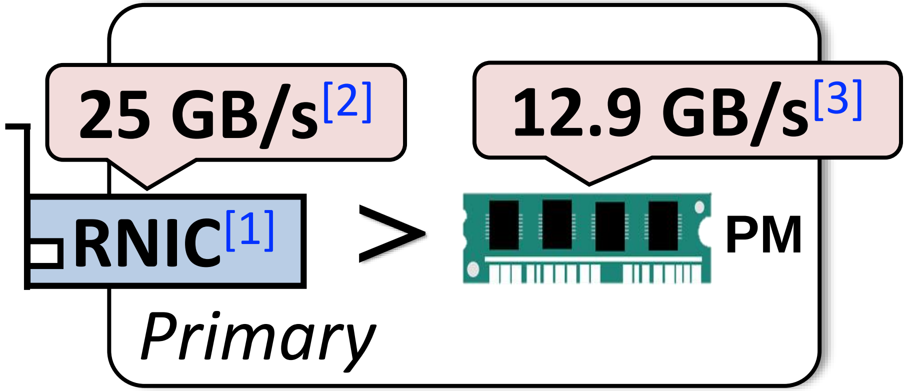
- **<u>Limited PM bandwidth:</u>** *High loads on primary*



[1] RDMA NIC [2] Mellanox ConnectX-6/dual-port ConnectX-5 VPI Card
[3] Six interleaved 256GB Intel Optane PM [Jian Yang, et al.@FAST'20]

Challenge 2

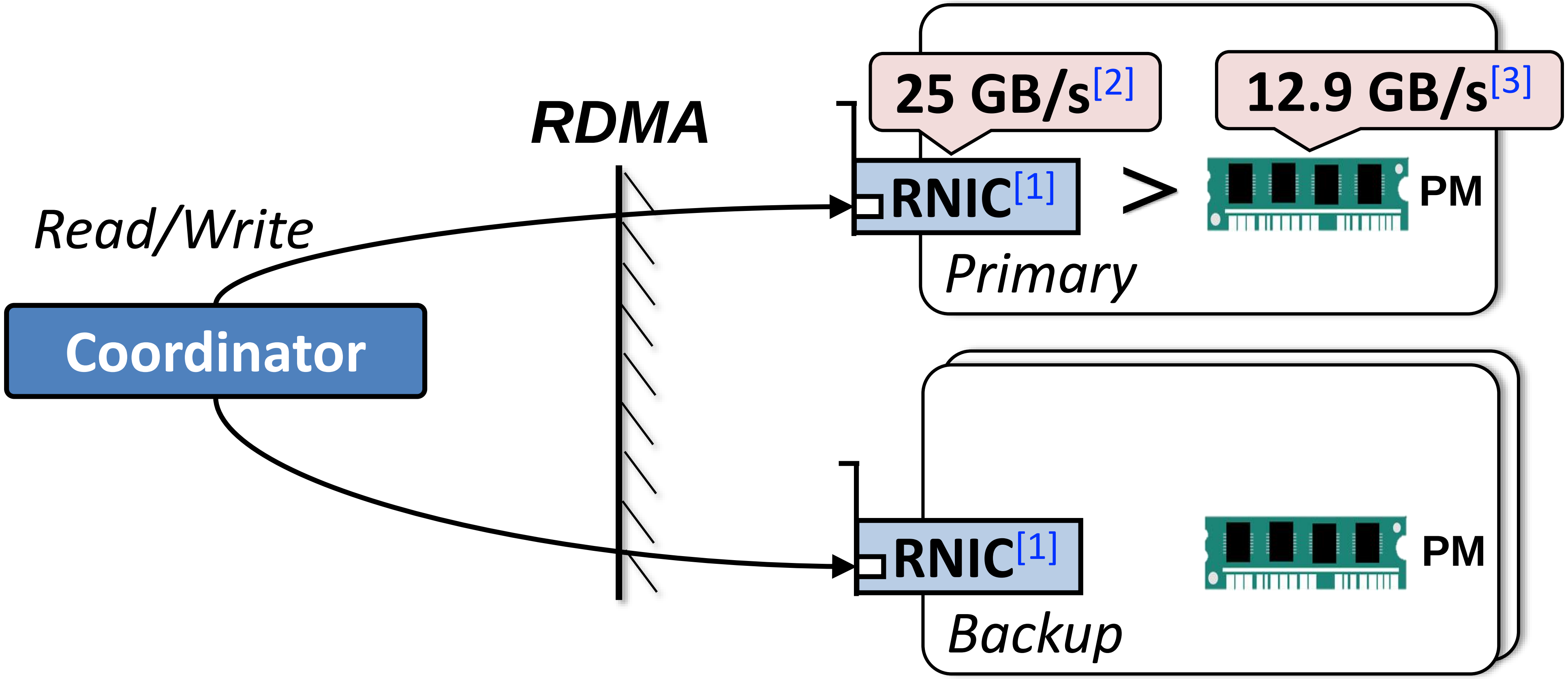
➢ **<u>Limited PM bandwidth:</u>** *High loads on primary*



[1] RDMA NIC [2] Mellanox ConnectX-6/dual-port ConnectX-5 VPI Card
[3] Six interleaved 256GB Intel Optane PM [Jian Yang, et al.@FAST'20]

Challenge 2

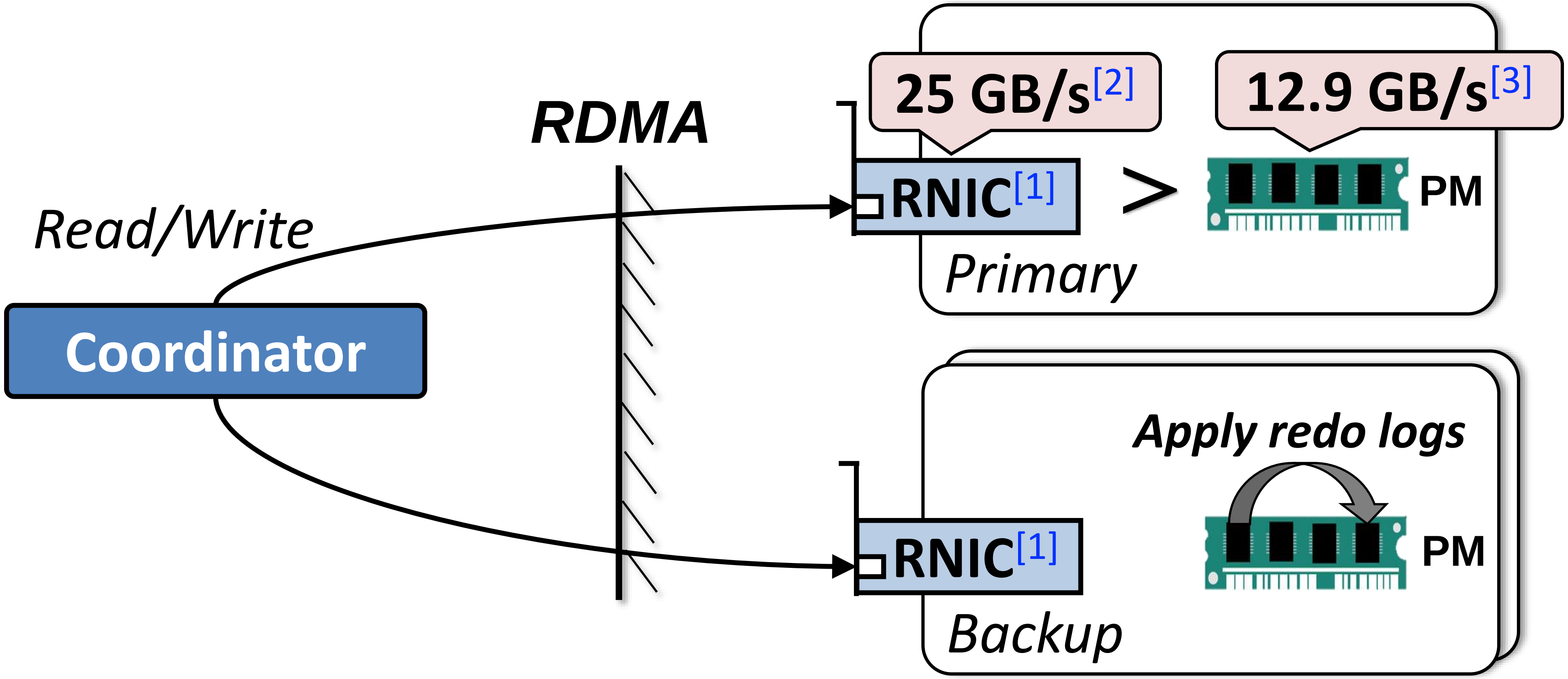
- **Limited PM bandwidth:** *High loads on primary*



[1] RDMA NIC [2] Mellanox ConnectX-6/dual-port ConnectX-5 VPI Card
[3] Six interleaved 256GB Intel Optane PM [Jian Yang, et al.@FAST'20]

Challenge 2

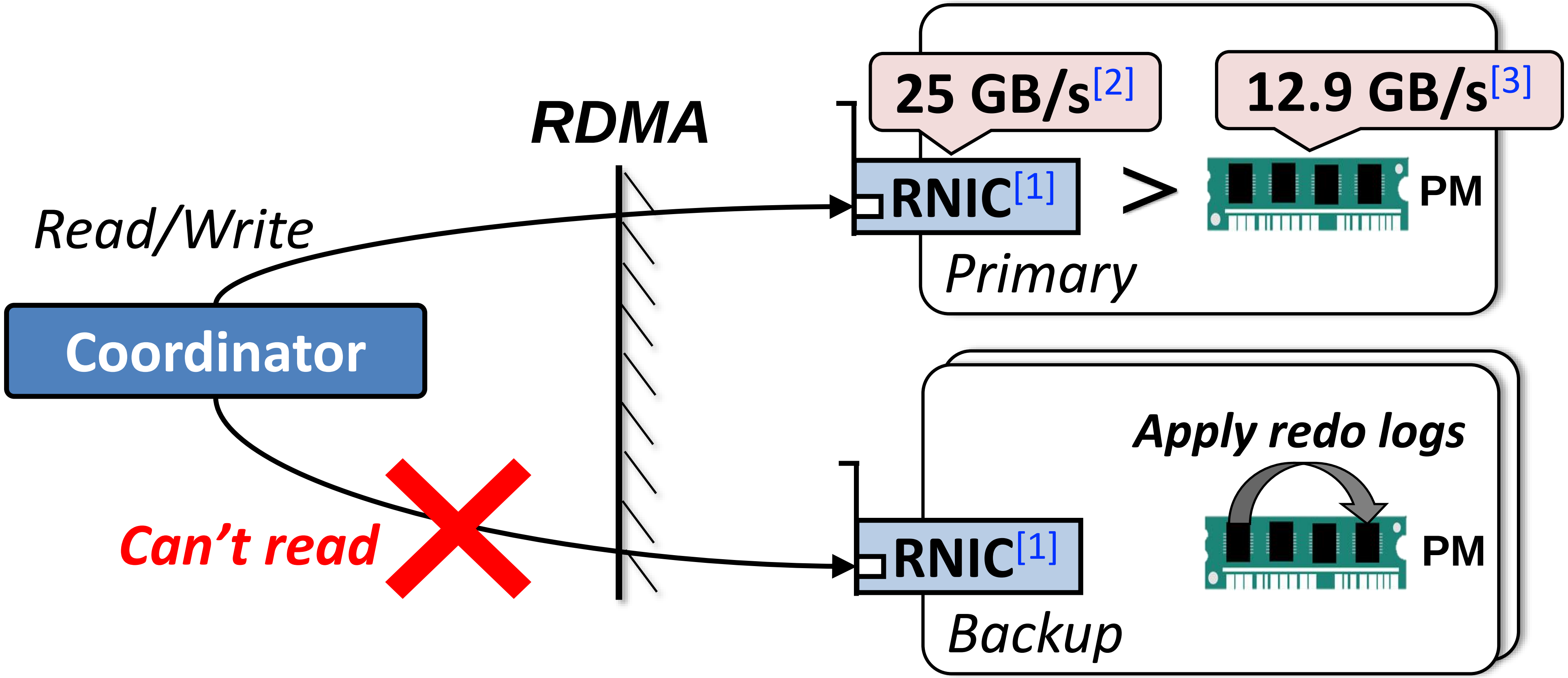
➢ **Limited PM bandwidth:** *High loads on primary*



[1] RDMA NIC [2] Mellanox ConnectX-6/dual-port ConnectX-5 VPI Card
[3] Six interleaved 256GB Intel Optane PM [Jian Yang, et al.@FAST'20]

Challenge 2

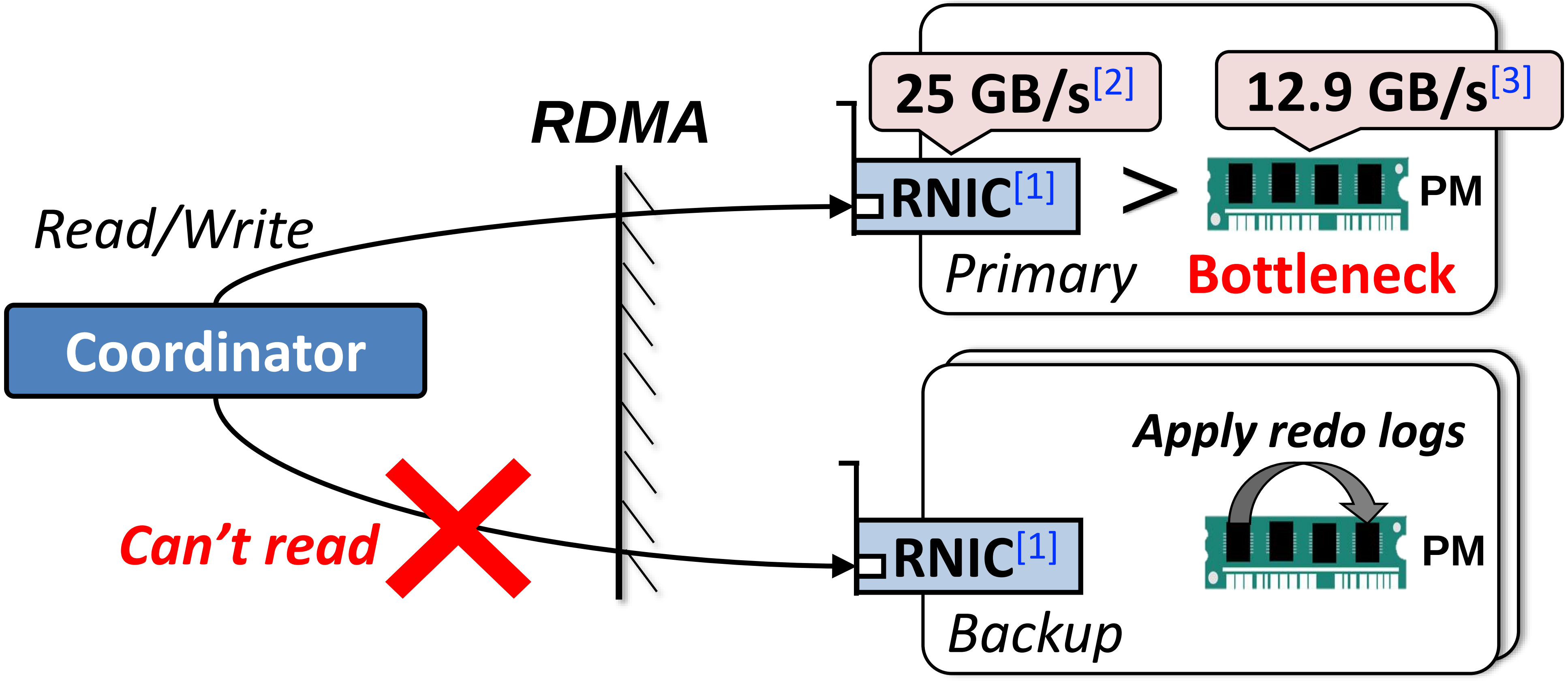
➢ **<u>Limited PM bandwidth:</u>** *High loads on primary*



[1] RDMA NIC [2] Mellanox ConnectX-6/dual-port ConnectX-5 VPI Card
[3] Six interleaved 256GB Intel Optane PM [Jian Yang, et al.@FAST'20]

Challenge 2

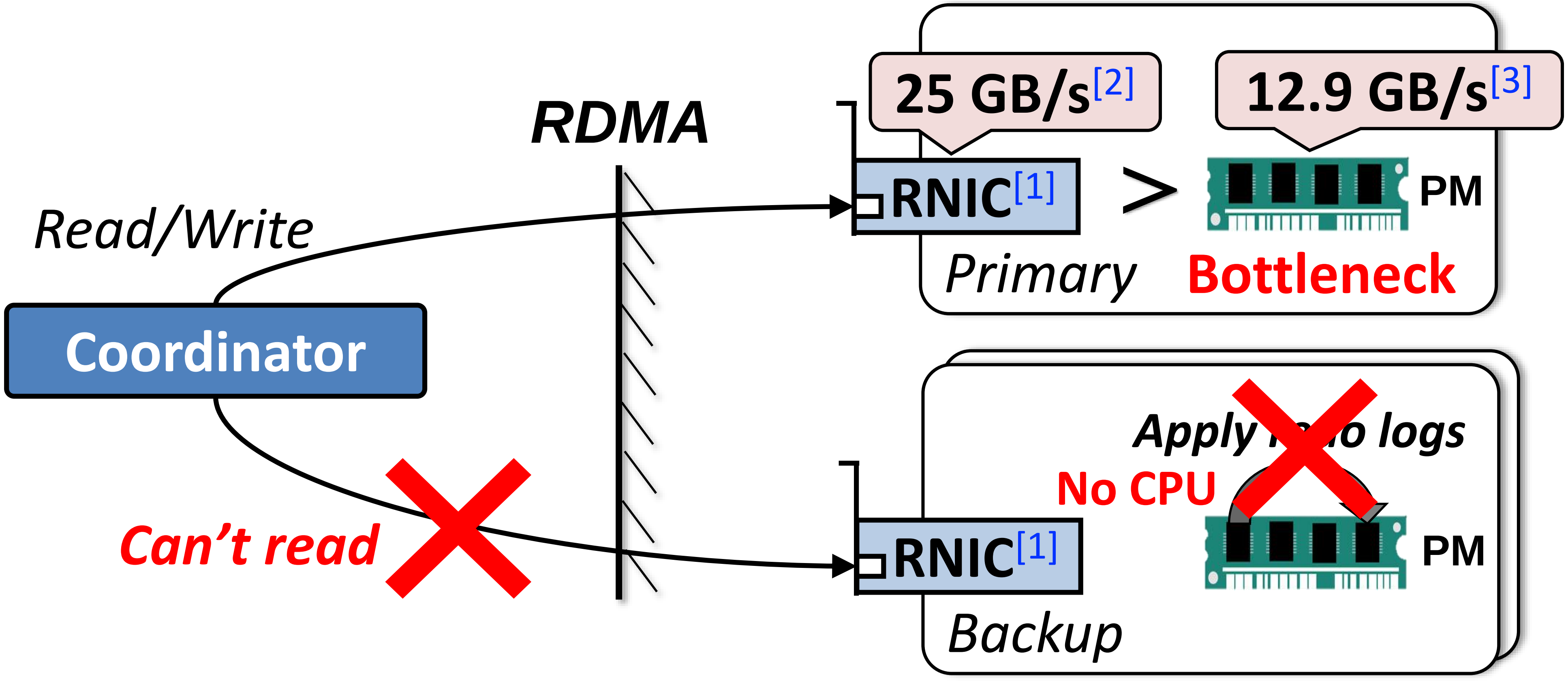
➢ **<u>Limited PM bandwidth:</u>** *High loads on primary*



[1] RDMA NIC [2] Mellanox ConnectX-6/dual-port ConnectX-5 VPI Card
[3] Six interleaved 256GB Intel Optane PM [Jian Yang, et al.@FAST'20]

Challenge 2

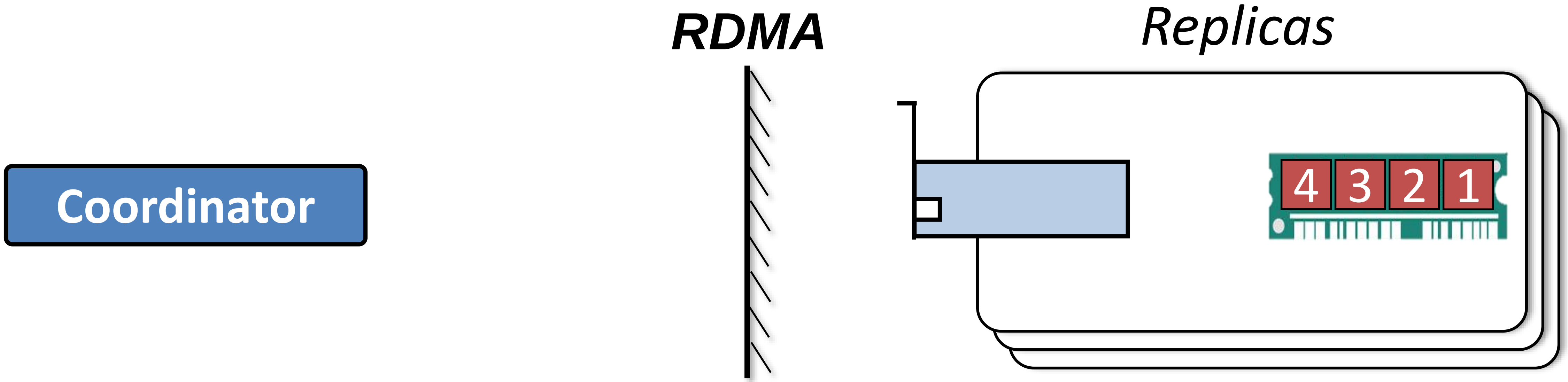
➢ **<u>Limited PM bandwidth:</u>** *High loads on primary*



[1] RDMA NIC [2] Mellanox ConnectX-6/dual-port ConnectX-5 VPI Card
[3] Six interleaved 256GB Intel Optane PM [Jian Yang, et al.@FAST'20]

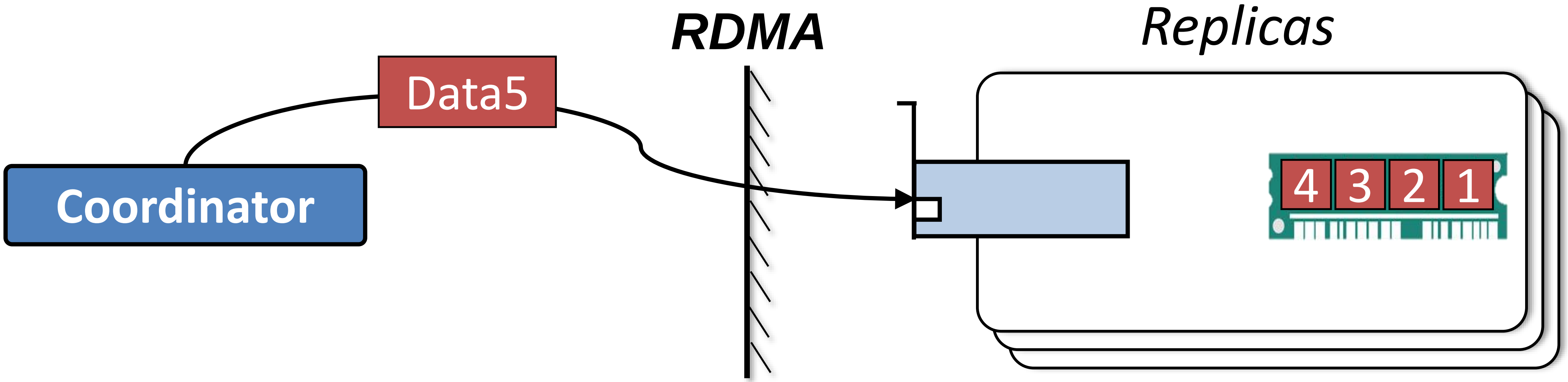
Challenge 3

- **<u>Lack of remote persistency guarantee:</u>** *Inconsistent write*



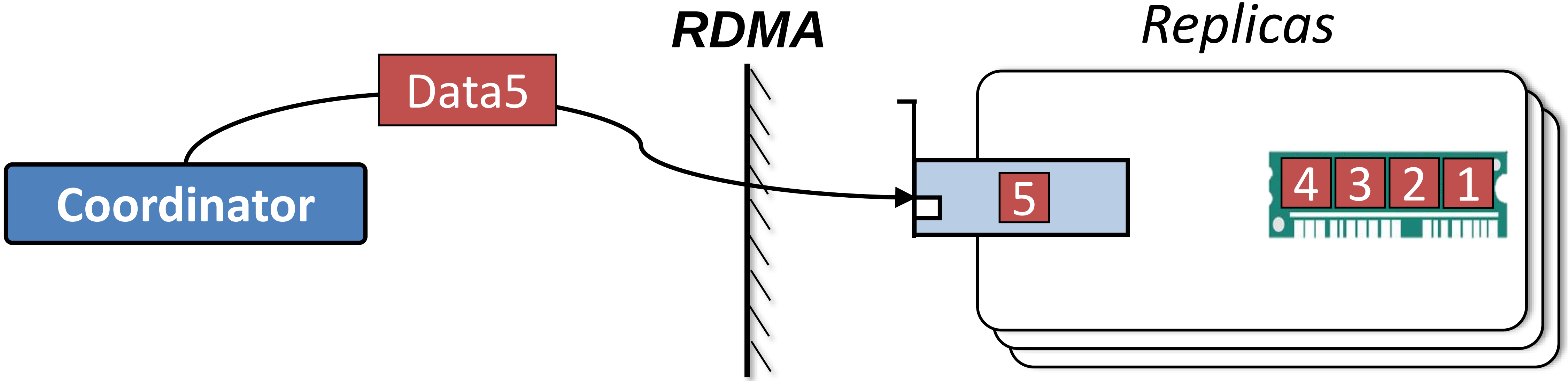
Challenge 3

➢ **Lack of remote persistency guarantee:** *Inconsistent write*



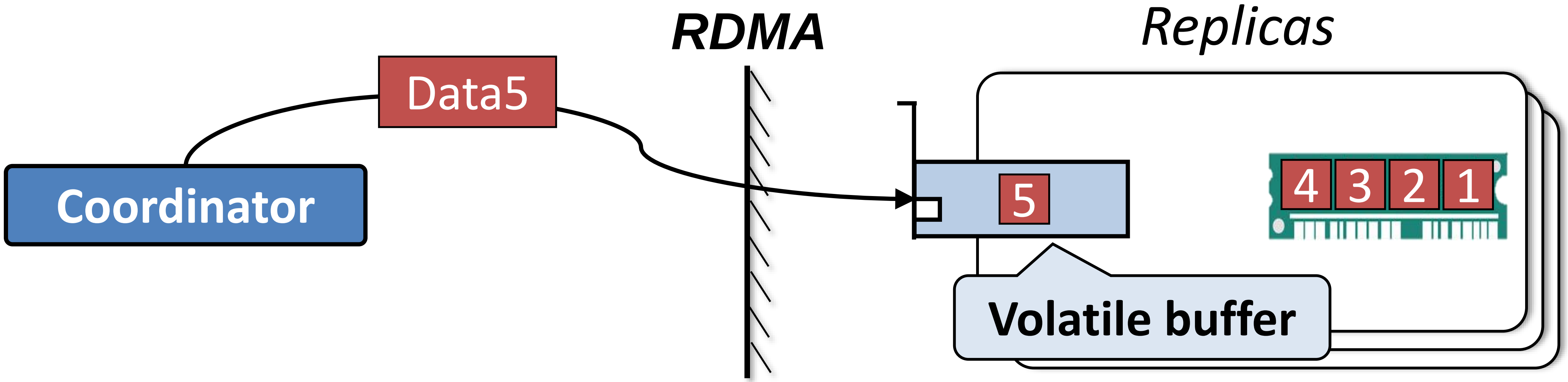
Challenge 3

➢ <u>**Lack of remote persistency guarantee:**</u> *Inconsistent write*



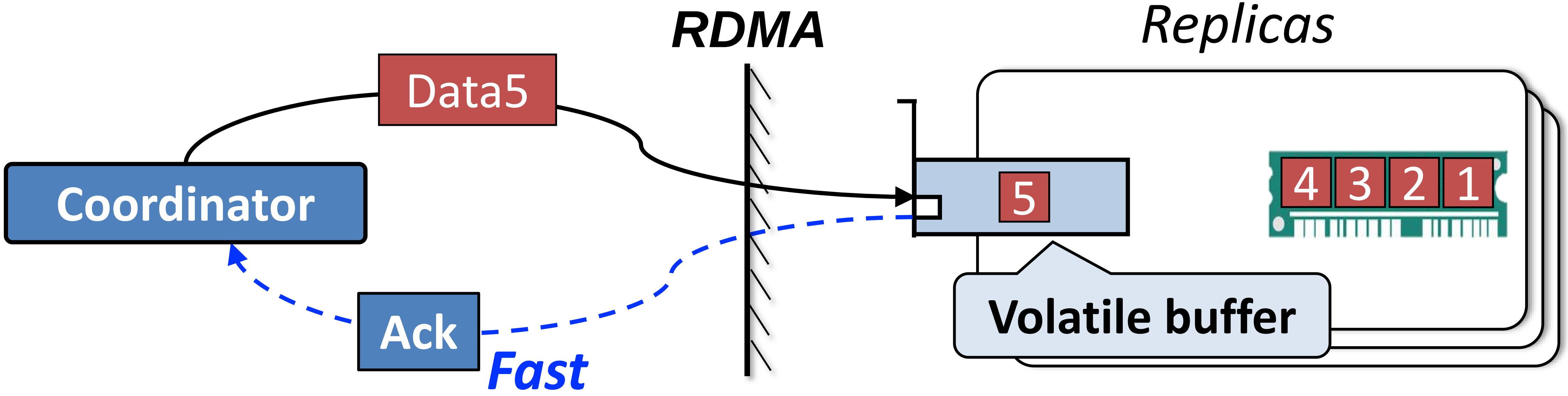
Challenge 3

➢ **<u>Lack of remote persistency guarantee:</u>** *Inconsistent write*



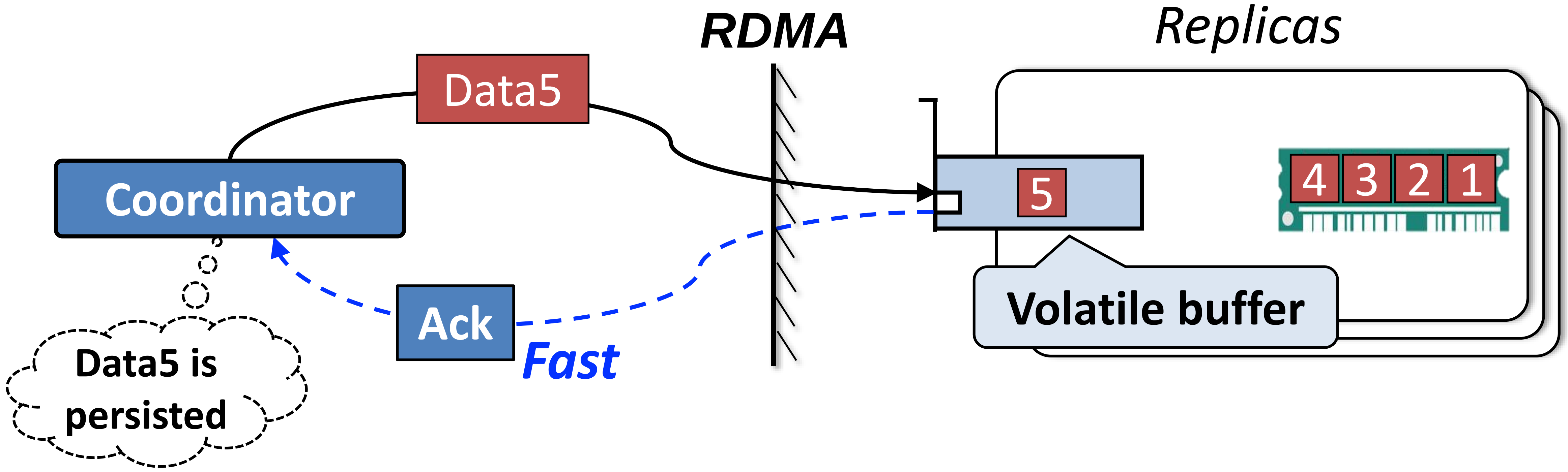
Challenge 3

➢ **<u>Lack of remote persistency guarantee:</u>** *Inconsistent write*



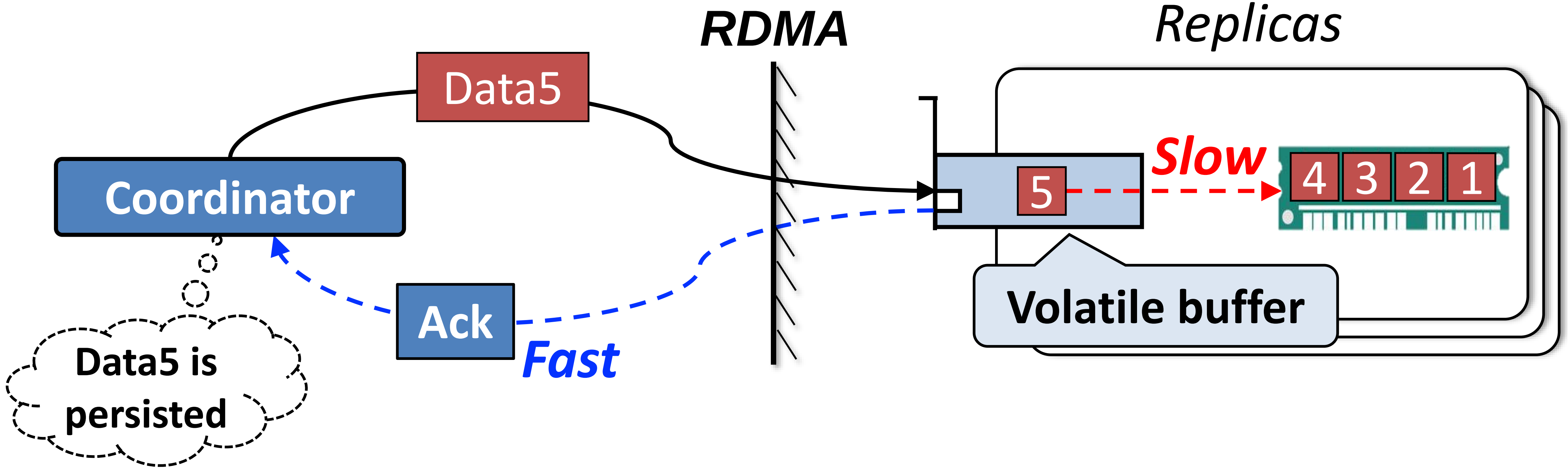
Challenge 3

- **<u>Lack of remote persistency guarantee:</u>** *Inconsistent write*



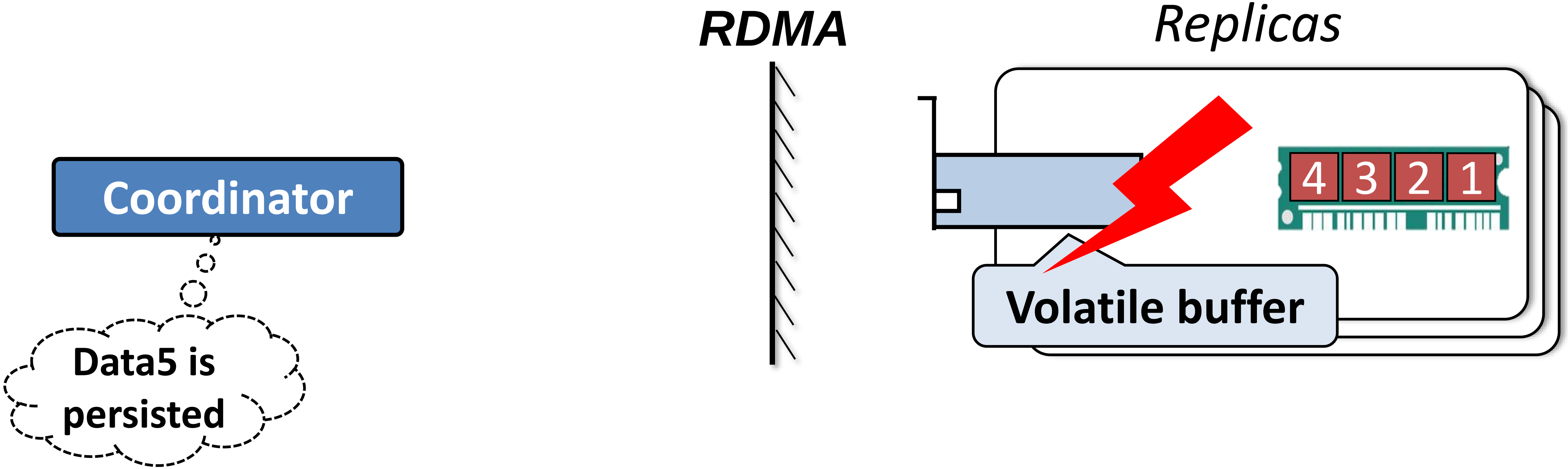
Challenge 3

➢ **<u>Lack of remote persistency guarantee:</u>** *Inconsistent write*



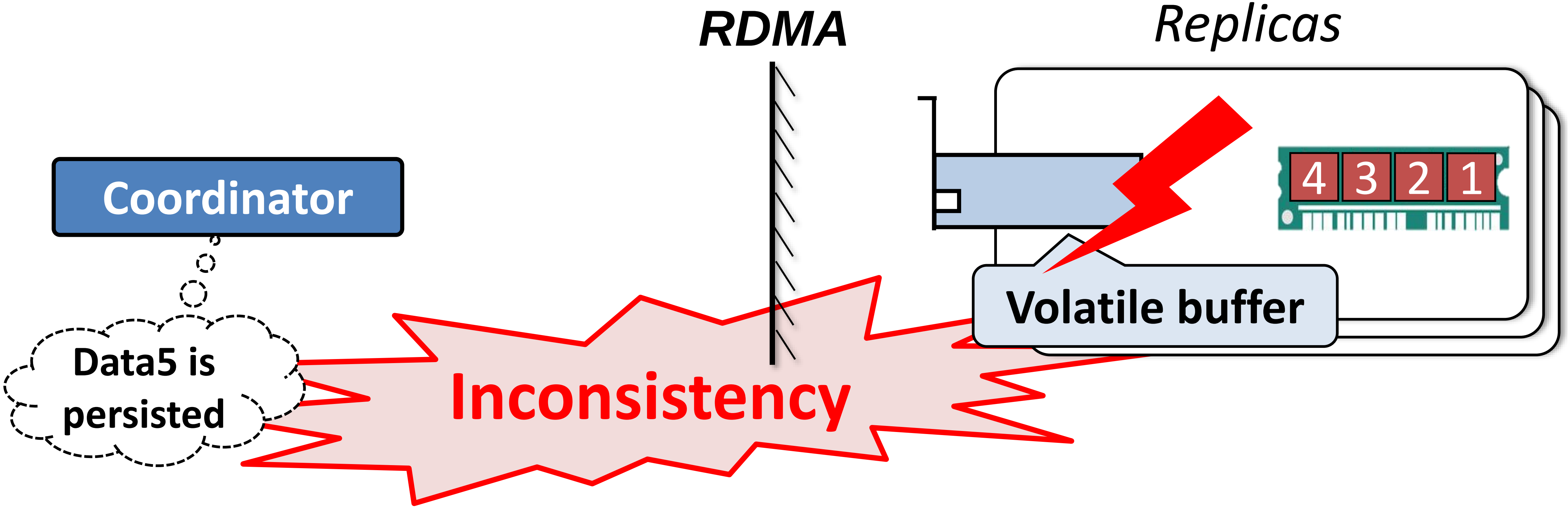
Challenge 3

➢ **<u>Lack of remote persistency guarantee:</u>** *Inconsistent write*



Challenge 3

- **<u>Lack of remote persistency guarantee:</u>** *Inconsistent write*



FORD: Solutions

- **Long-latency processing:** *Many round trips*

- **Limited PM bandwidth:** *High loads on primary*

- **Lack of remote persistency guarantee:** *Inconsistent write*

FORD: Solutions

- **Long-latency processing:** *Many round trips*
  - Hitchhiked Locking
  - Coalescent Commit

  Reduce round trips to decrease latency

- **Limited PM bandwidth:** *High loads on primary*

- **Lack of remote persistency guarantee:** *Inconsistent write*

FORD: Solutions

- **Long-latency processing:** *Many round trips*
  - **Hitchhiked Locking**
  - **Coalescent Commit**

  Reduce round trips to decrease latency

- **Limited PM bandwidth:** *High loads on primary*
  - **Backup-enabled Read**

  Balance load to improve throughput

- **Lack of remote persistency guarantee:** *Inconsistent write*

FORD: Solutions

- **Long-latency processing:** *Many round trips*
  - **Hitchhiked Locking**
  - **Coalescent Commit**

  ➔ Reduce round trips to decrease latency

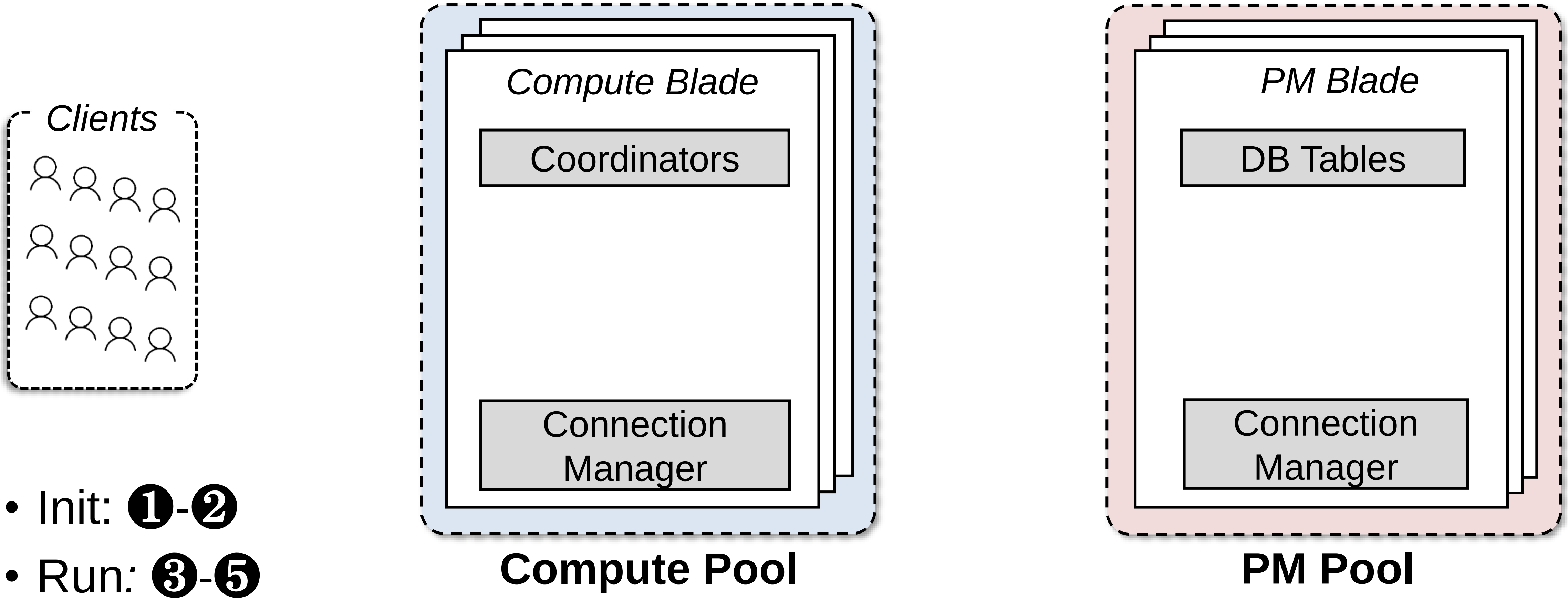
- **Limited PM bandwidth:** *High loads on primary*
  - **Backup-enabled Read**

  ➔ Balance load to improve throughput

- **Lack of remote persistency guarantee:** *Inconsistent write*
  - **Selective Remote Flush**

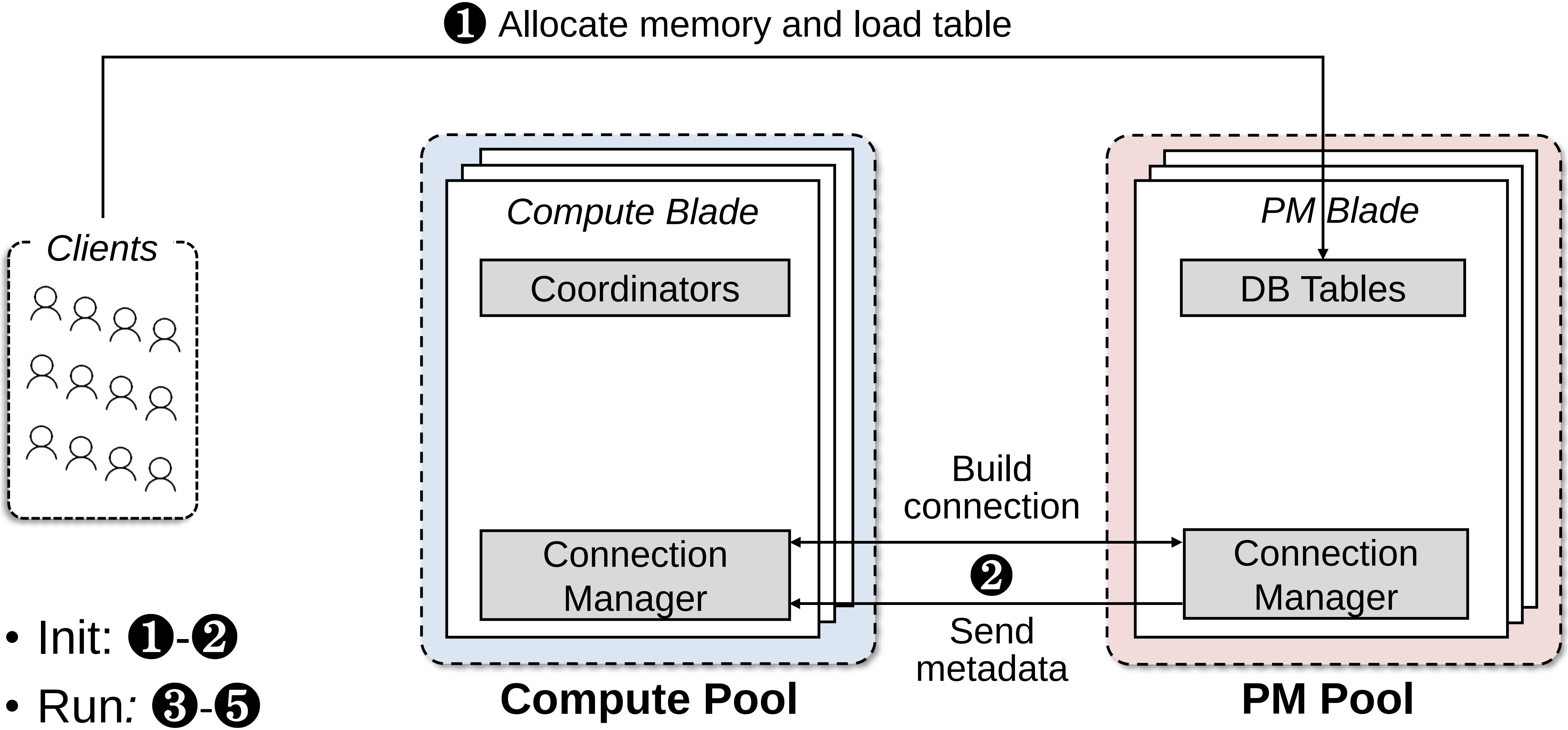
  ➔ Guarantee remote persistency with low overhead

Overview



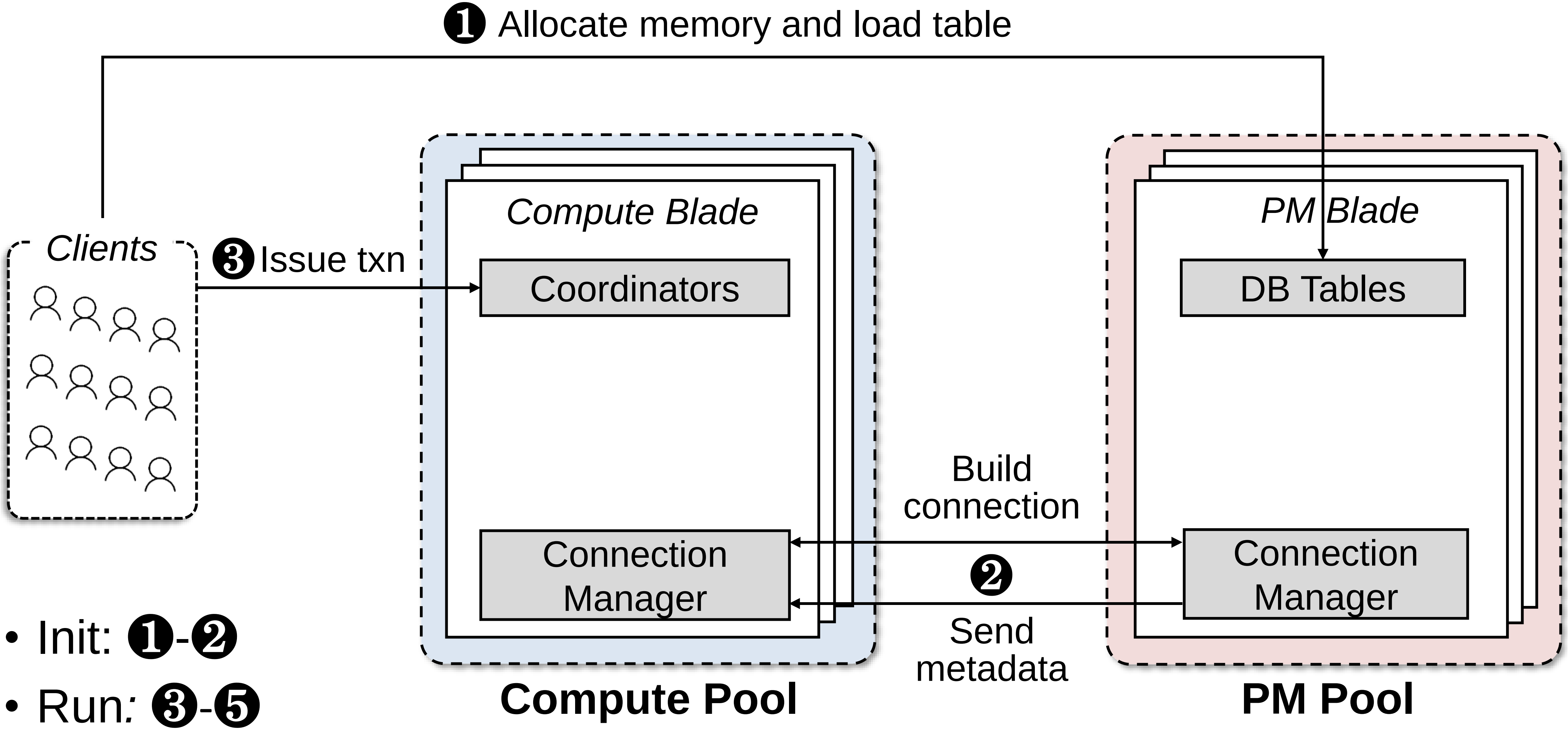
- Init: ❶-❷
- Run: ❸-❺

Overview



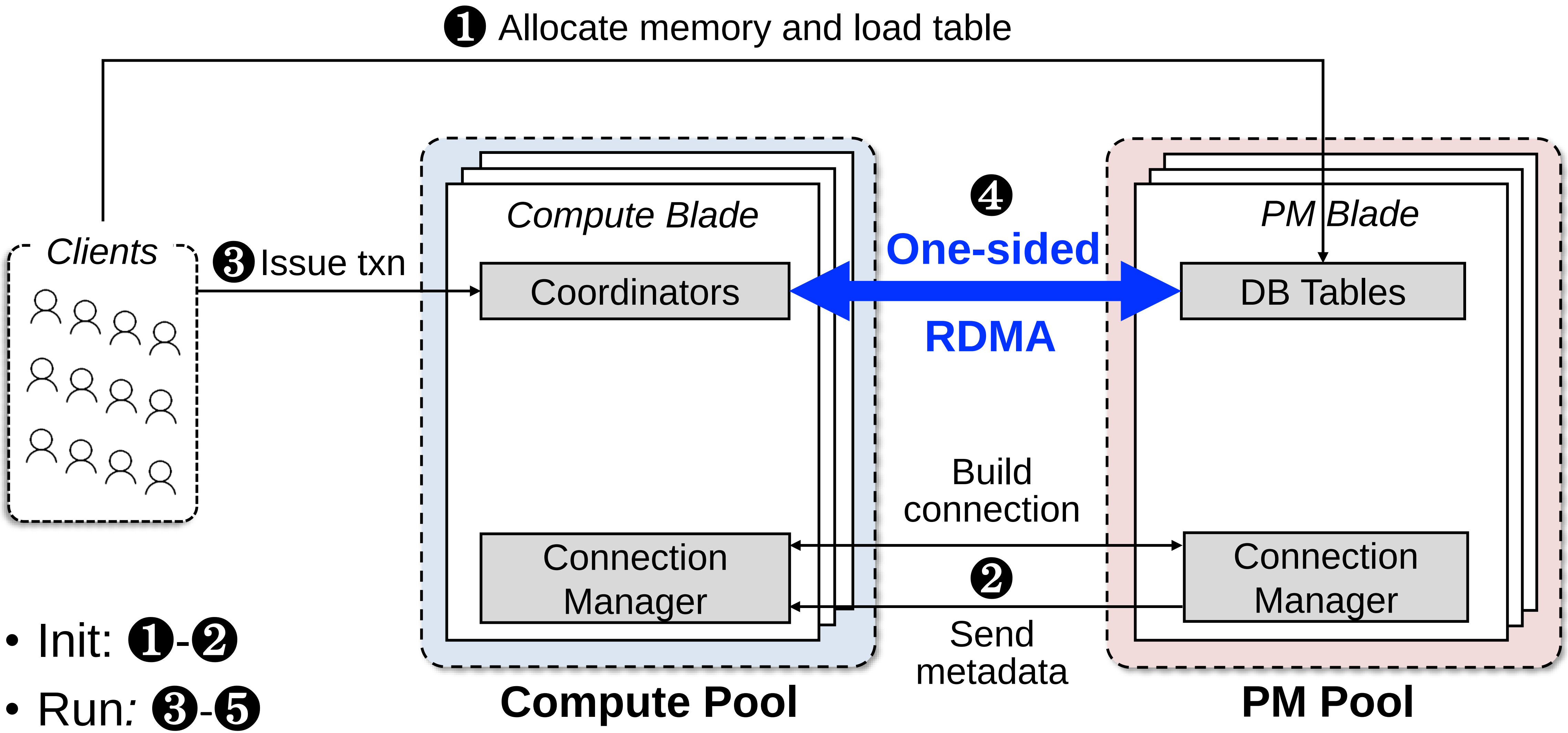
- Init: ❶-❷
- Run: ❸-❺

Overview



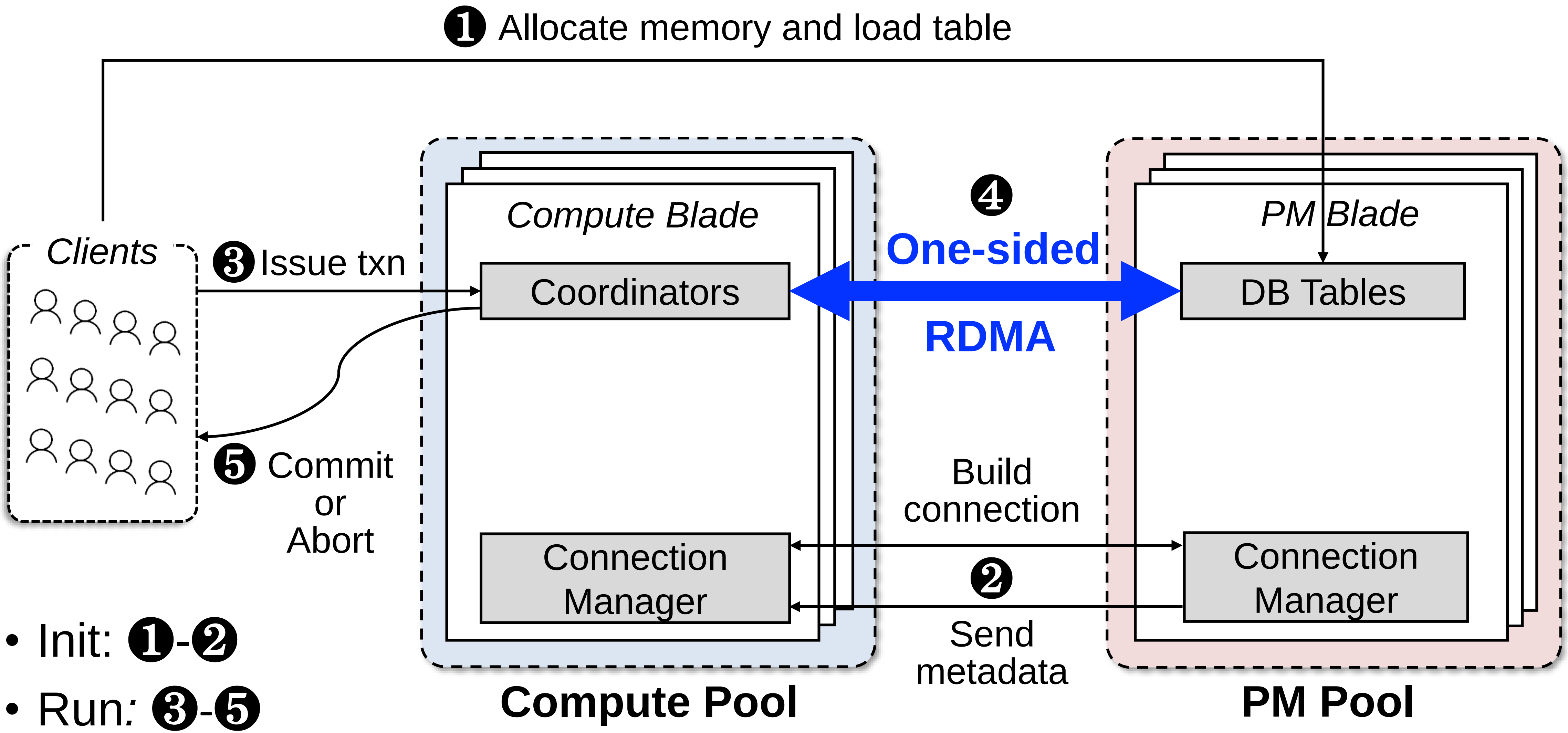
- Init: ❶-❷
- Run: ❸-❺

Overview



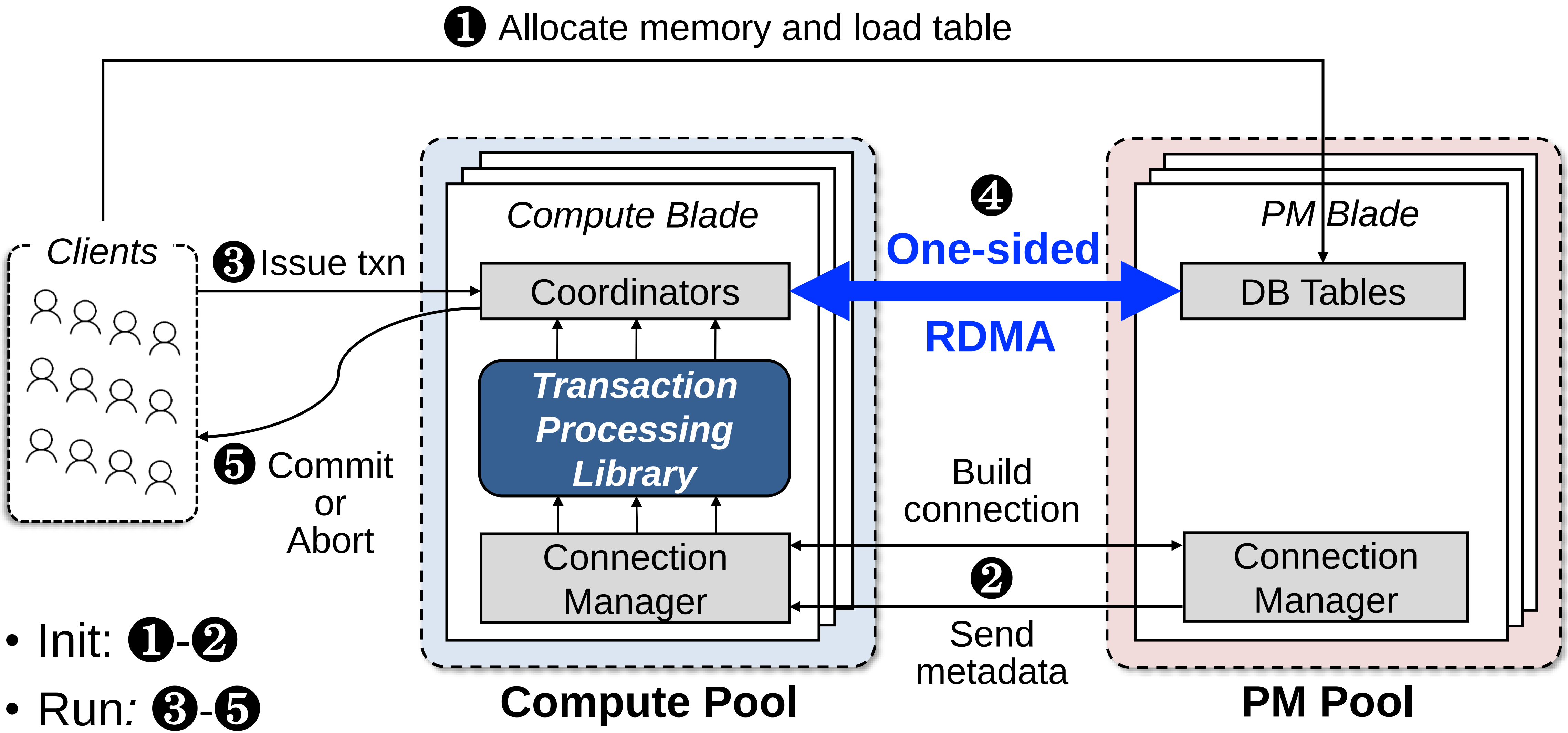
- Init: ❶-❷
- Run: ❸-❺

Overview



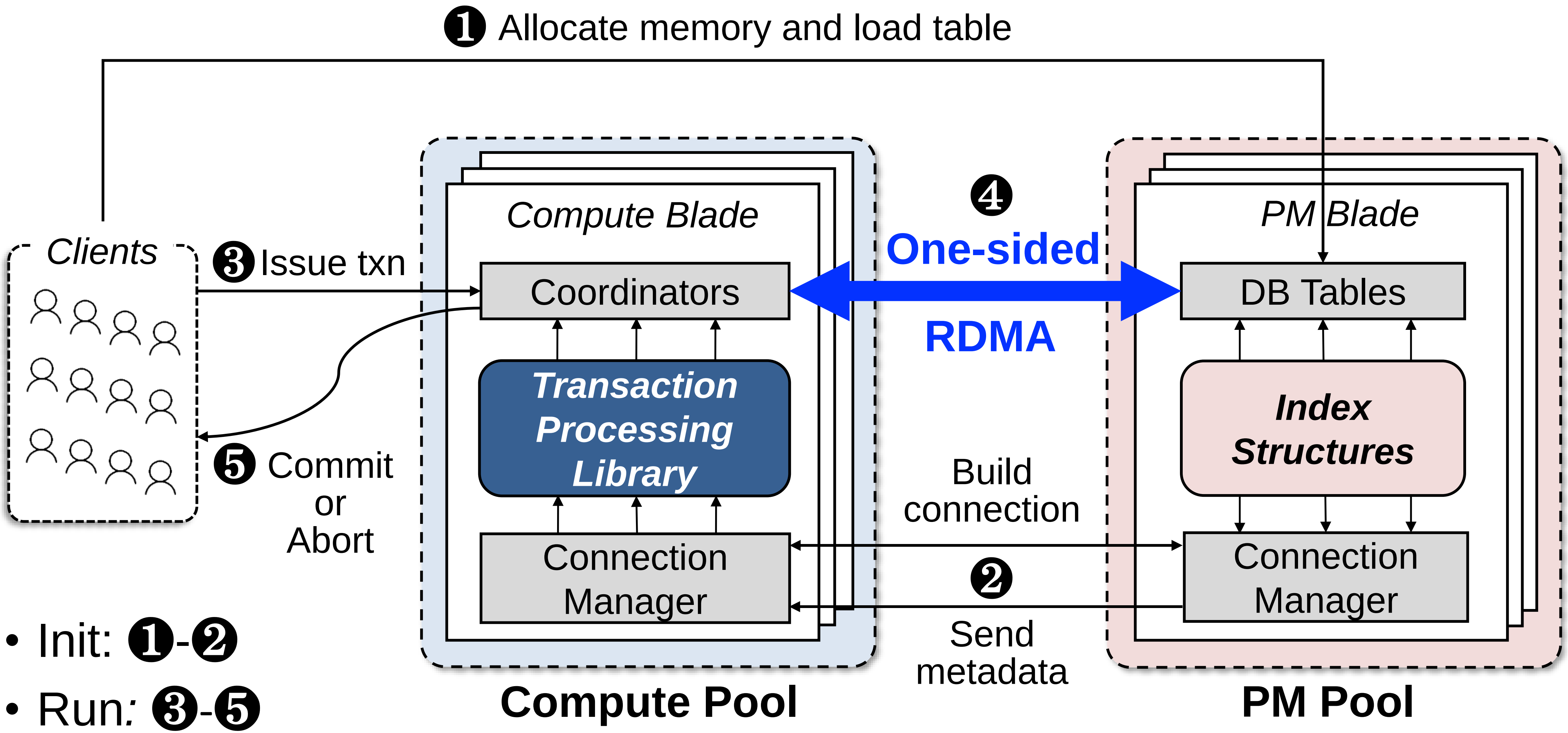
- Init: ❶-❷
- Run: ❸-❺

Overview



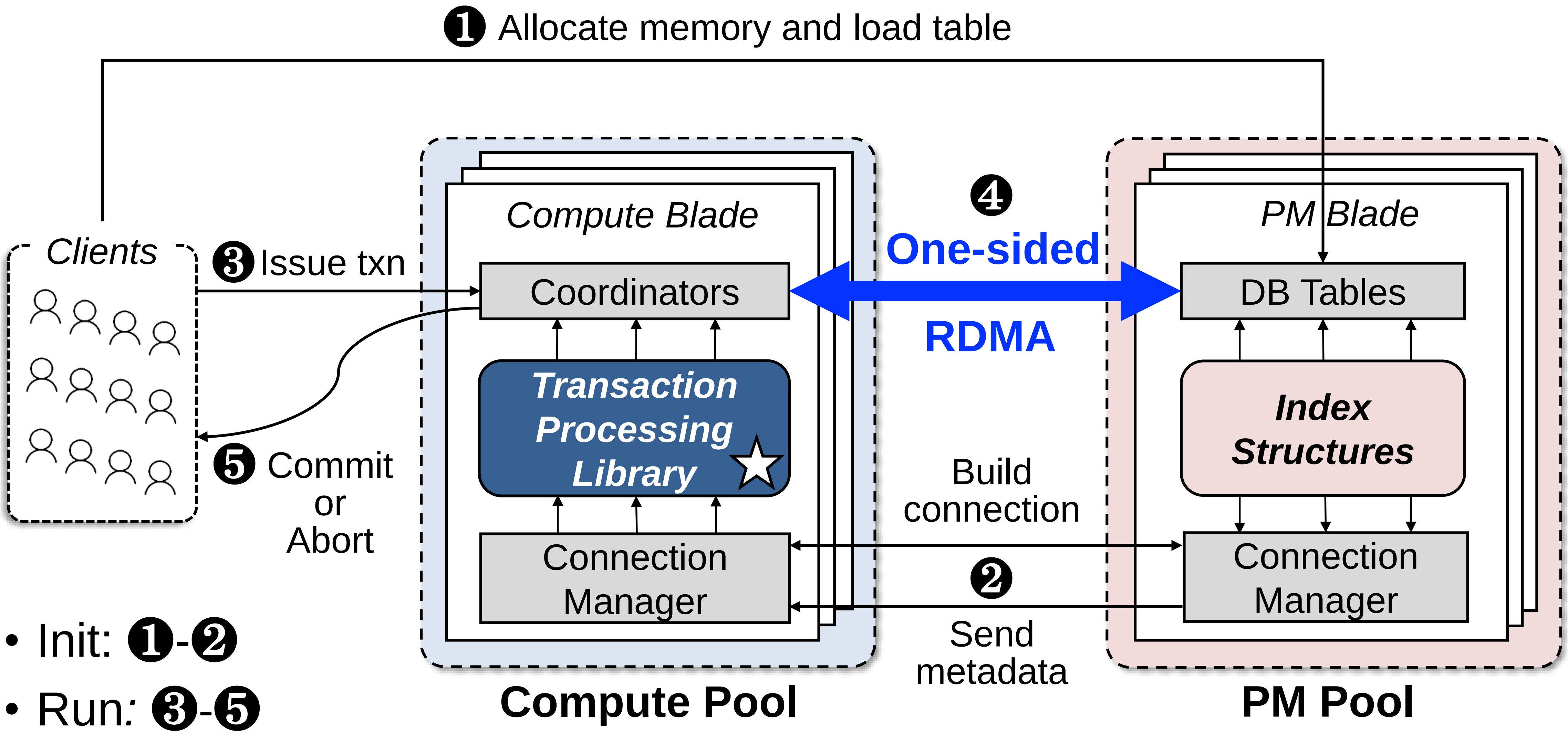
- Init: ❶-❷
- Run: ❸-❺

Overview



- Init: ❶-❷
- Run: ❸-❺

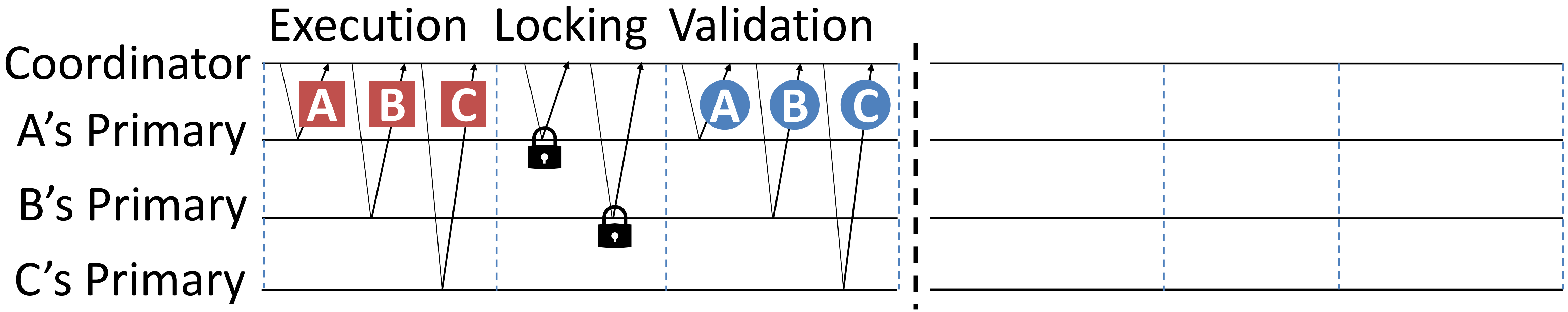
Overview



- Init: ❶-❷
- Run: ❸-❺

Hitchhiked Locking

- ➢ Read and lock the **read-write (RW) set** in execution
  - Avoid subsequent locking and validations
  - No lock on the read-only data

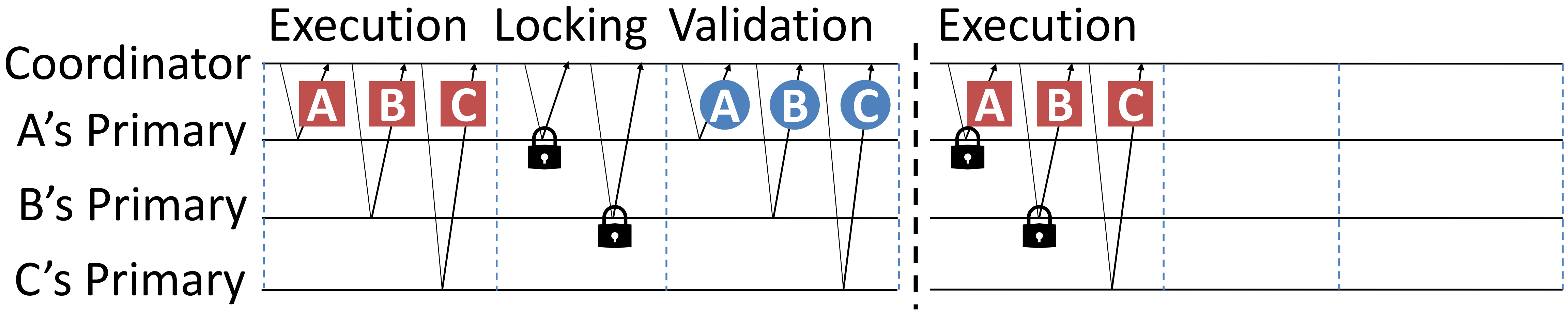


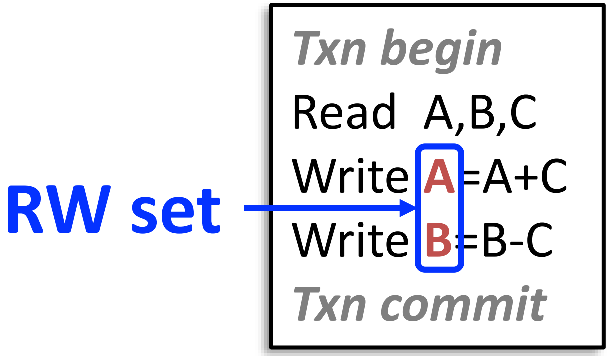
Txn begin
Read A,B,C
Write A=A+C
Write B=B-C
Txn commit

RW set

Hitchhiked Locking

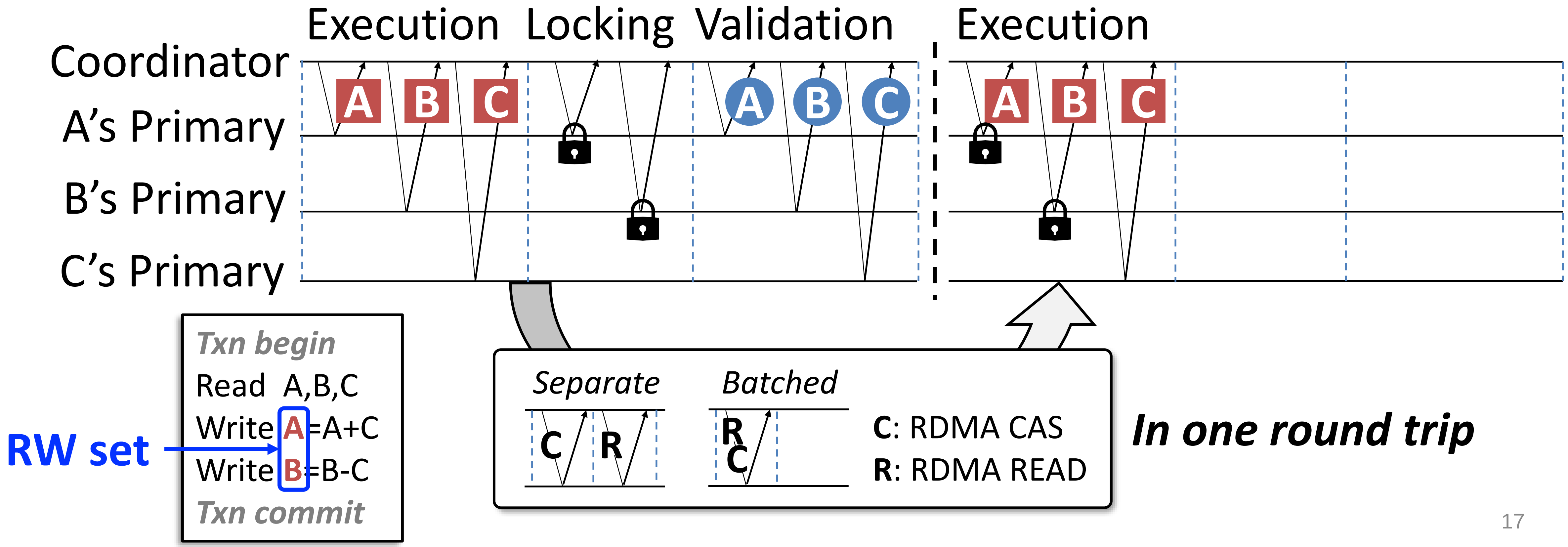
- Read and lock the **<u>read-write (RW) set</u>** in execution
  - Avoid subsequent locking and validations
  - No lock on the read-only data





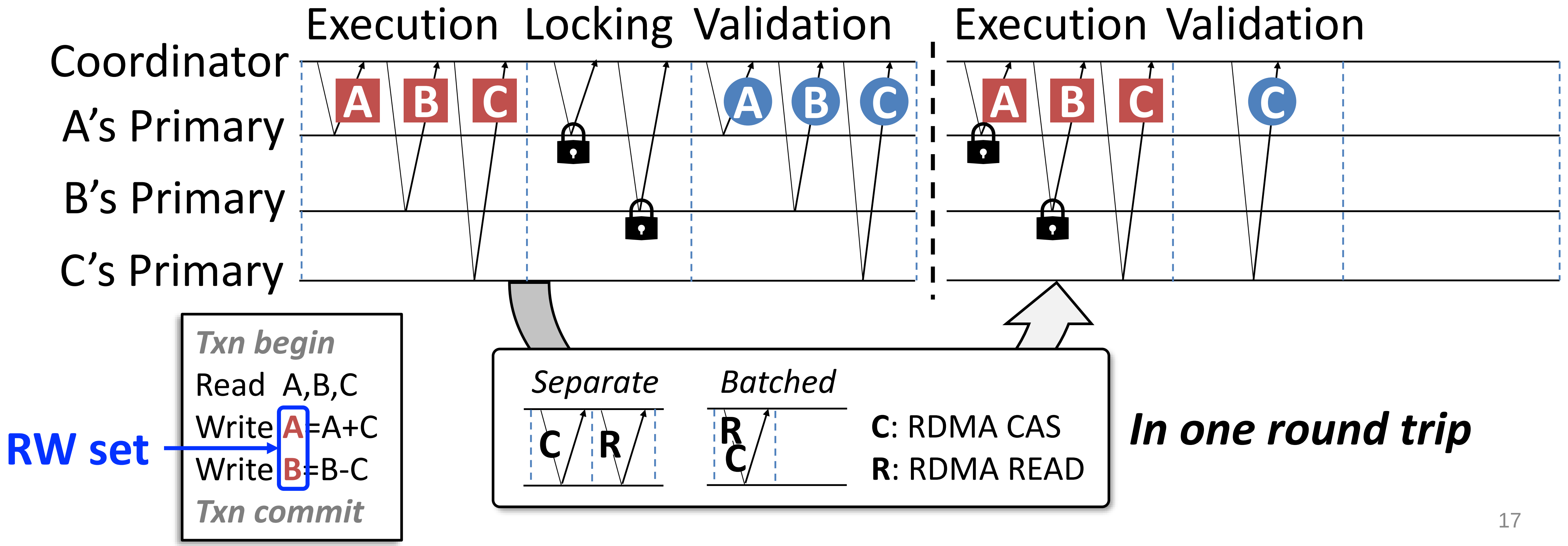
Hitchhiked Locking

- Read and lock the **read-write (RW) set** in execution
  - Avoid subsequent locking and validations
  - No lock on the read-only data



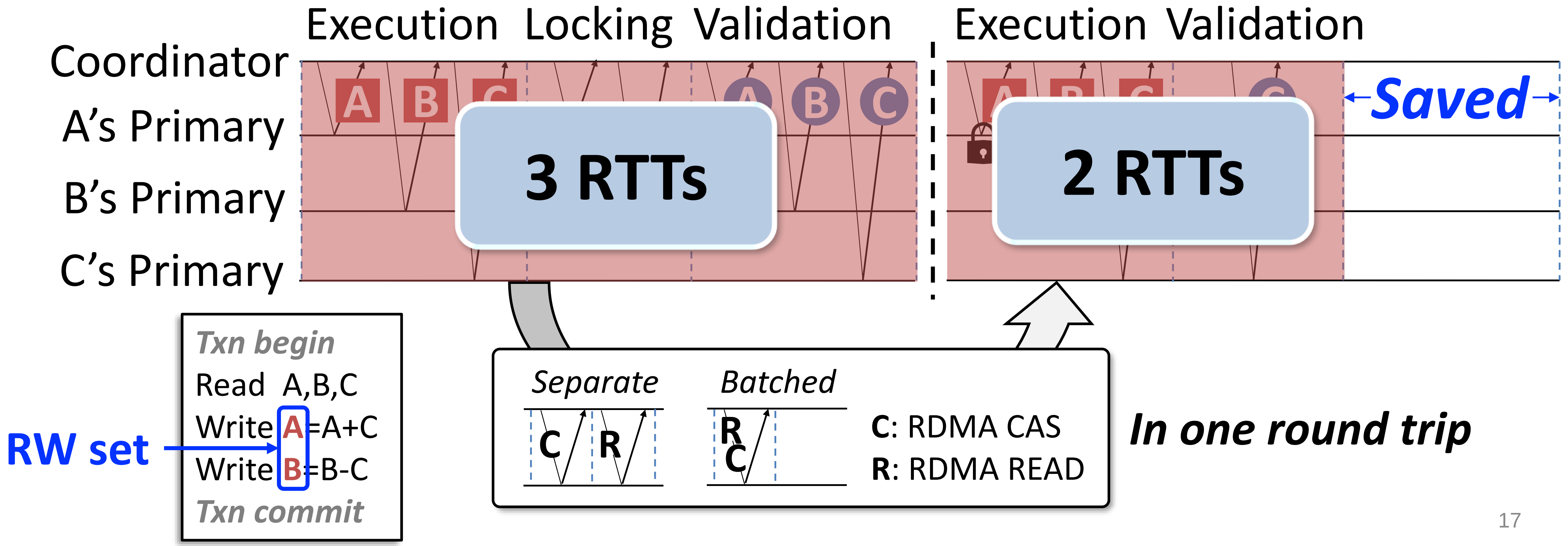
Hitchhiked Locking

- Read and lock the **<u>read-write (RW) set</u>** in execution
  - Avoid subsequent locking and validations
  - No lock on the read-only data



Hitchhiked Locking

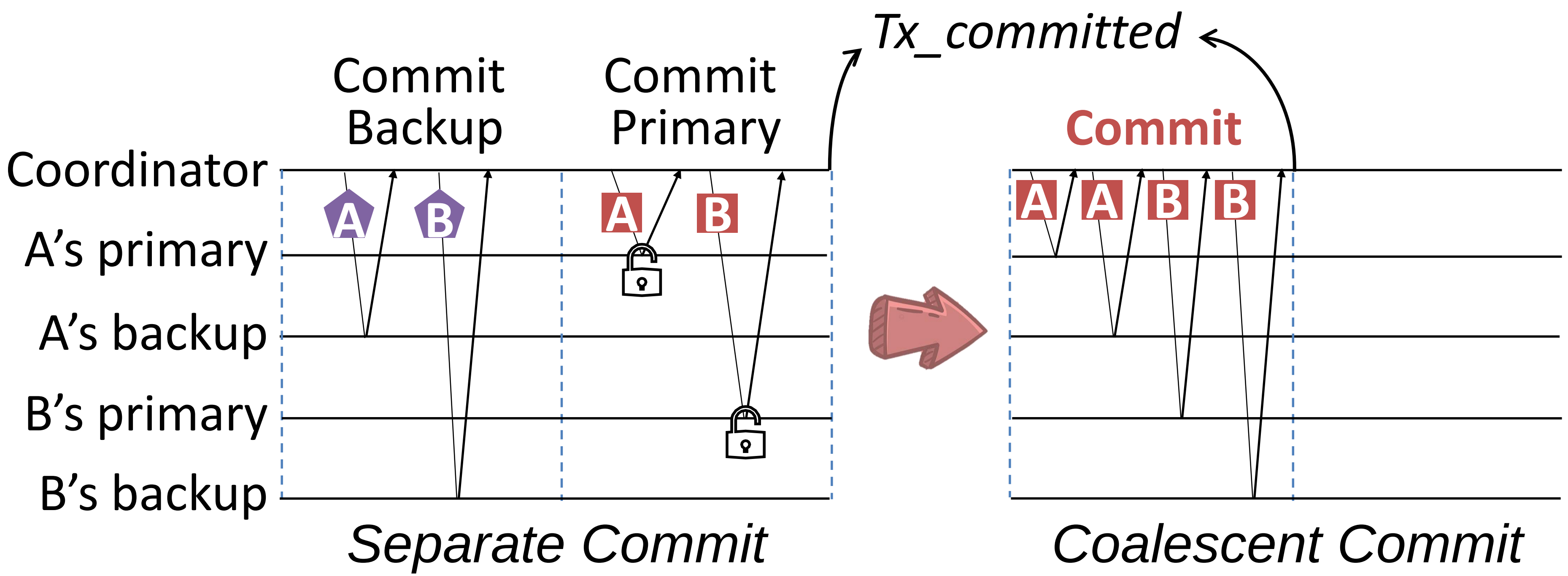
- ➢ Read and lock the **read-write (RW) set** in execution
  - Avoid subsequent locking and validations
  - No lock on the read-only data



Coalescent Commit

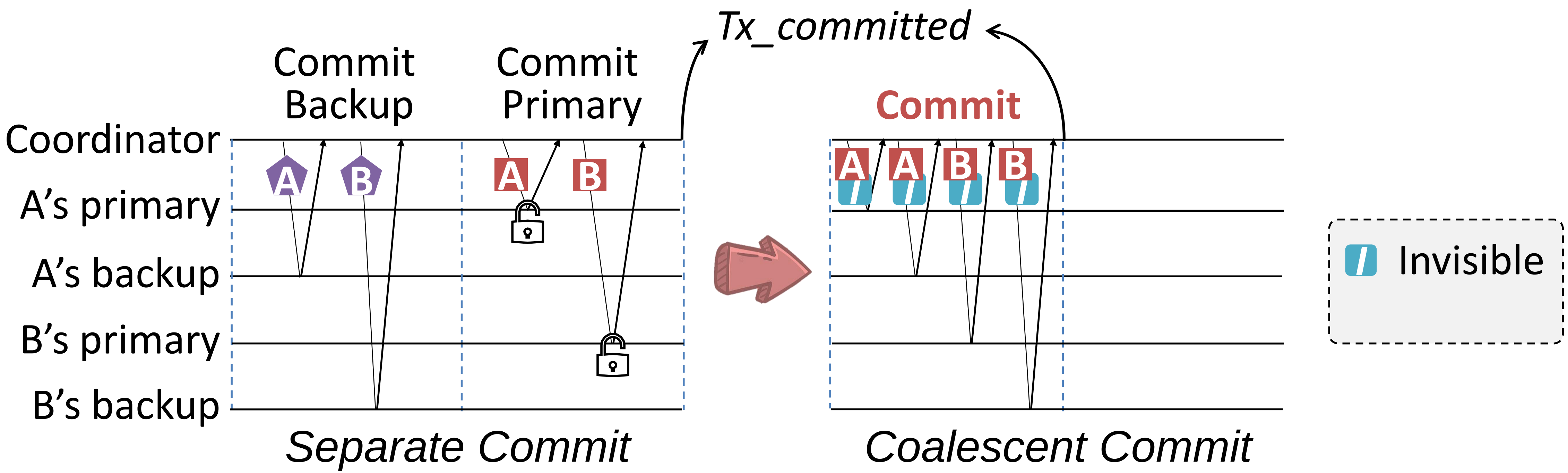
➢ Commit all replicas **together** in one round trip

- In-place update: Parallel undo logging in execution phase
- Prevent reading partial updates: Control data visibility



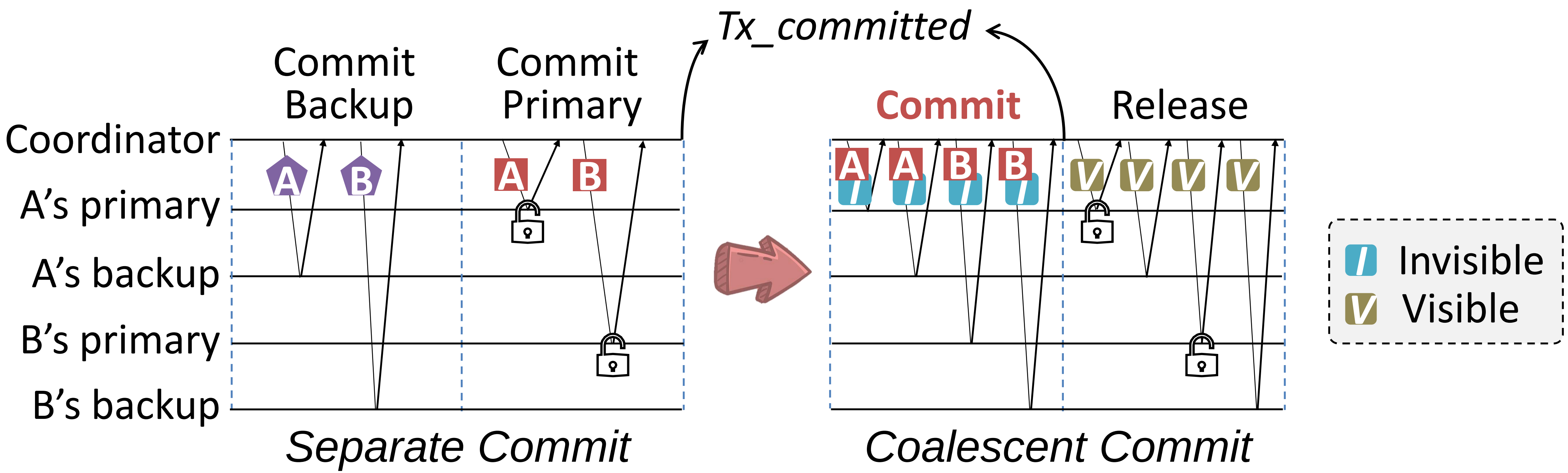
Coalescent Commit

- Commit all replicas **together** in one round trip
  - In-place update: Parallel undo logging in execution phase
  - Prevent reading partial updates: Control data visibility



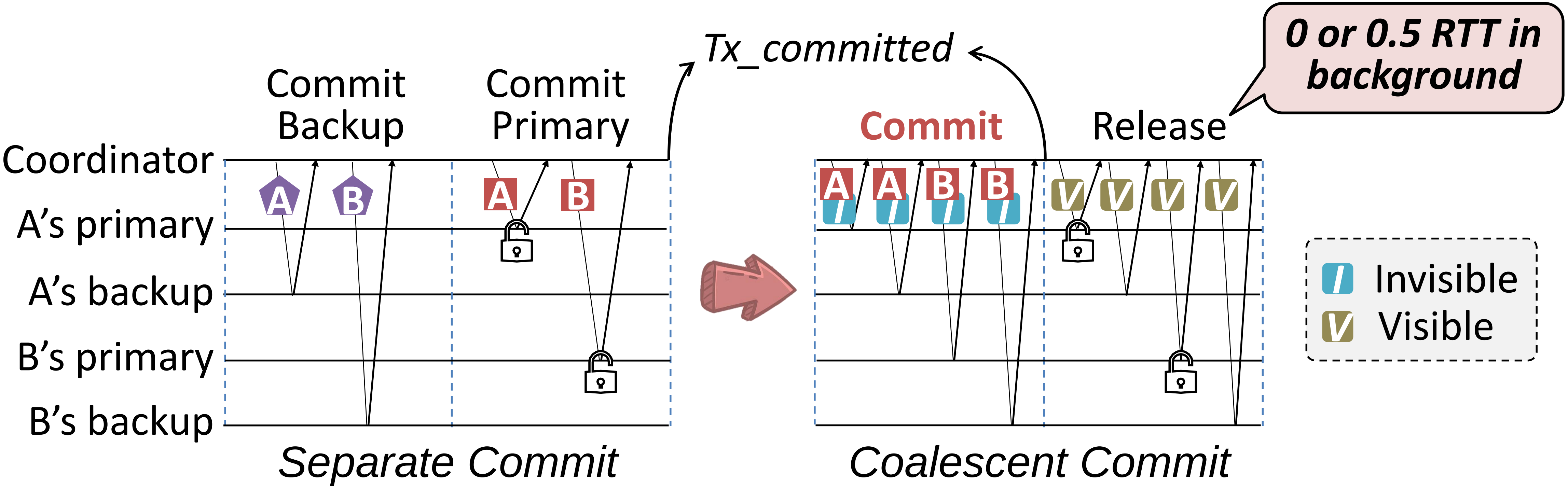
Coalescent Commit

- Commit all replicas **<u>together</u>** in one round trip
  - In-place update: Parallel undo logging in execution phase
  - Prevent reading partial updates: Control data visibility



Coalescent Commit

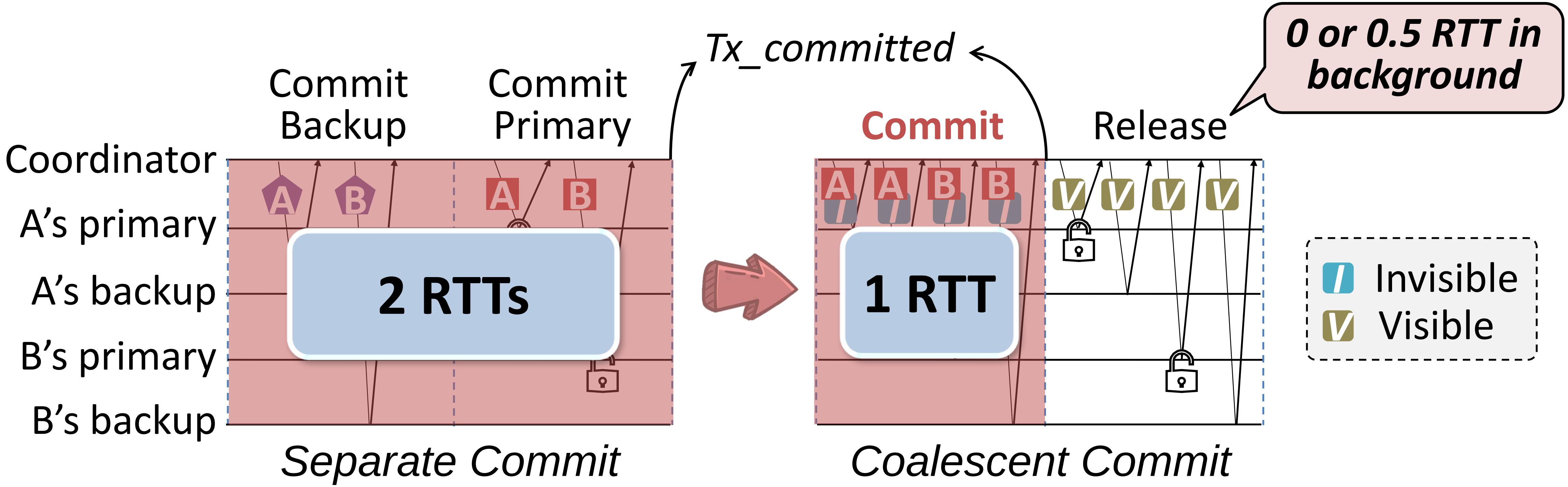
- ➢ Commit all replicas **together** in one round trip
  - In-place update: Parallel undo logging in execution phase
  - Prevent reading partial updates: Control data visibility



Coalescent Commit

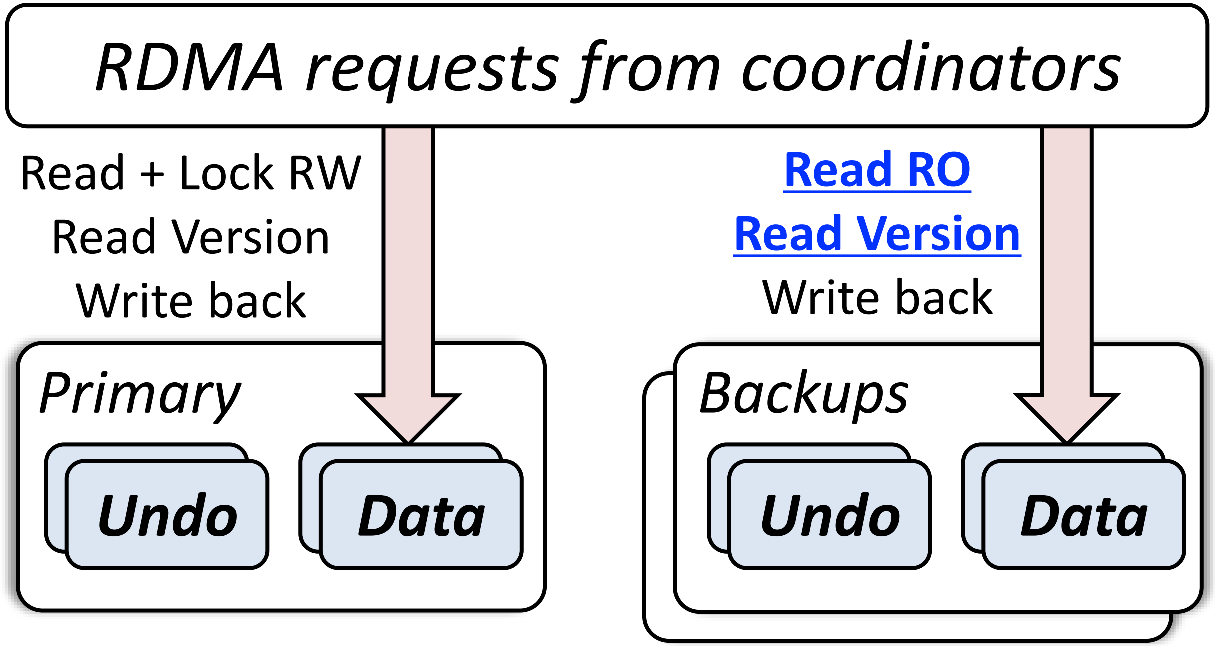
➢ Commit all replicas **together** in one round trip

- In-place update: Parallel undo logging in execution phase
- Prevent reading partial updates: Control data visibility



Backup-enabled Read

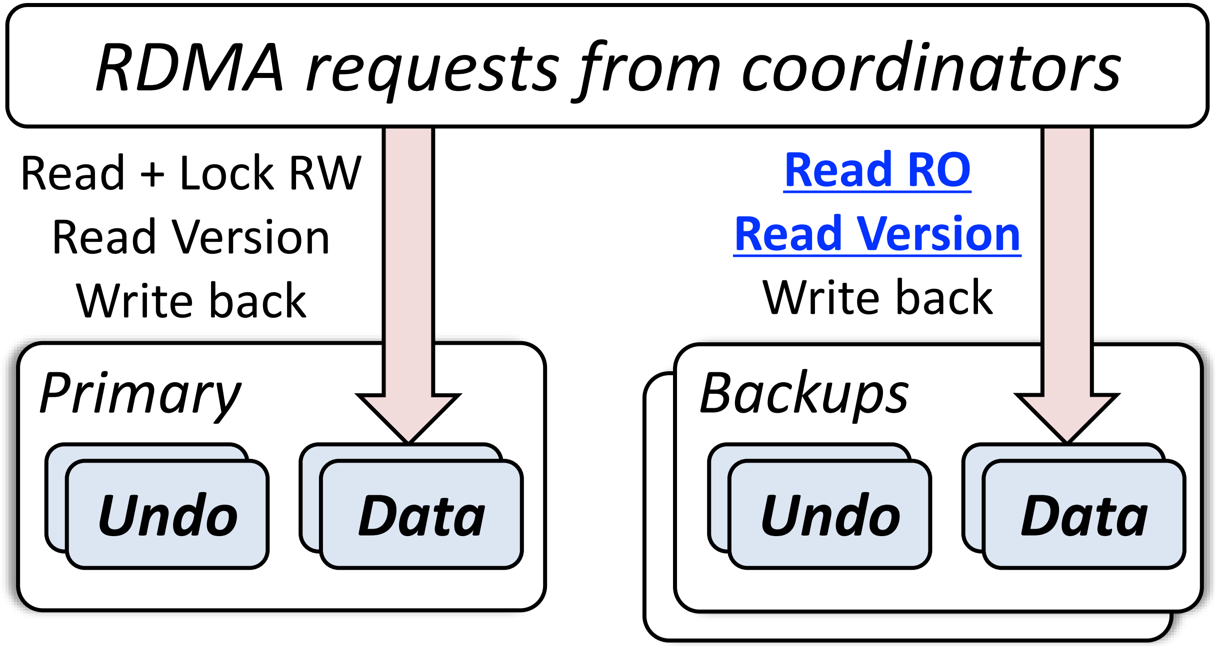
- Allows **backups** to serve **read-only (RO)** data
  - In-place update → No address redirection
  - Undo logging → No data migration → No CPU involvement in PM pool



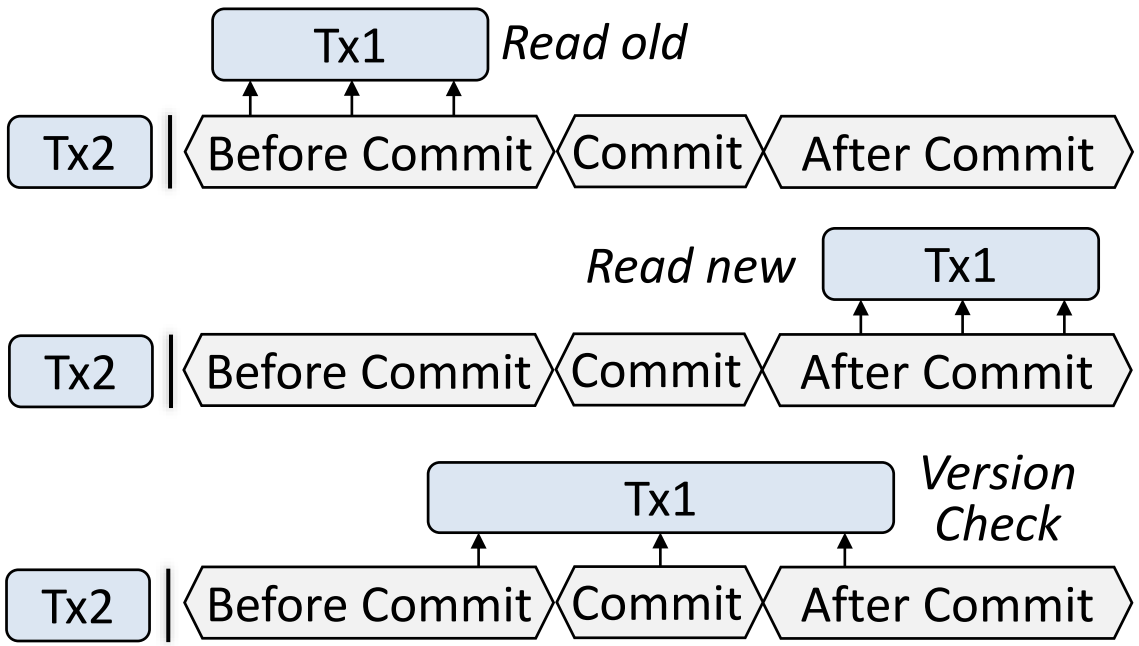
Balance load → Improve throughput

Backup-enabled Read

- Allows **backups** to serve **read-only (RO)** data
  - In-place update → No address redirection
  - Undo logging → No data migration → No CPU involvement in PM pool



Balance load → Improve throughput



Correctness guarantee

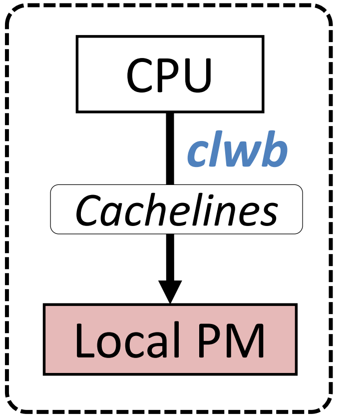
Selective Remote Flush

- Only issue remote flushes to **<u>backups</u>** after the **<u>final</u>** write
  - Guarantee remote persistency
  - Ensure recoverability by backups
  - Reduce flushing round trips
  - Compatible with different flush primitives [1]

[1] One-sided RDMA **FLUSH** or **READ** (*READ-after-WRITE*)

Selective Remote Flush

- ➢ Only issue remote flushes to **backups** after the **final** write
  - Guarantee remote persistency
  - Ensure recoverability by backups
  - Reduce flushing round trips
  - Compatible with different flush primitives [1]



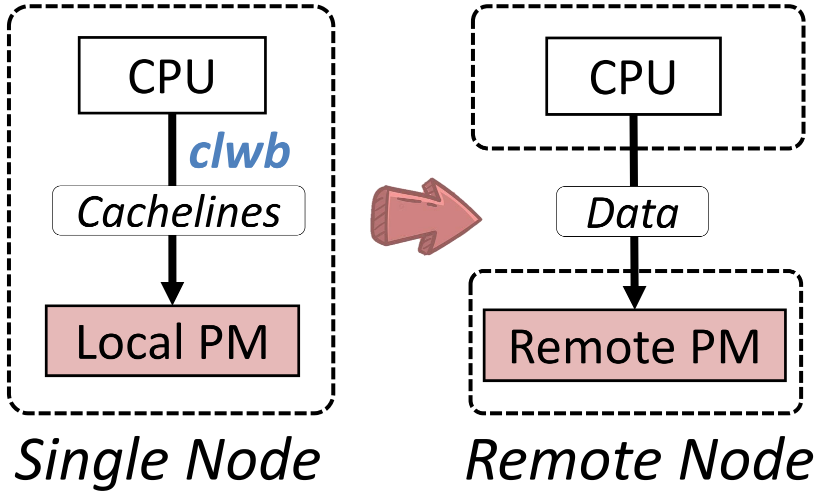
Single Node

[1] One-sided RDMA **FLUSH** or **READ** (*READ-after-WRITE*)

Selective Remote Flush

➢ Only issue remote flushes to **backups** after the **final** write

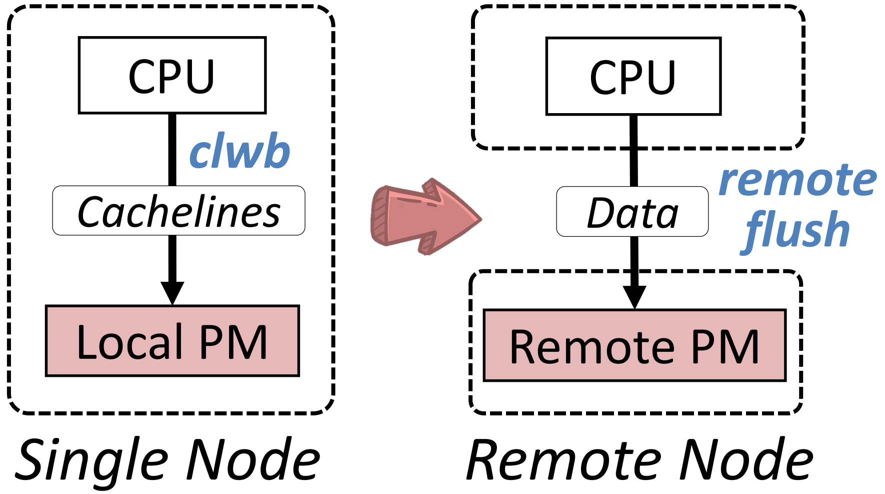
- Guarantee remote persistency
- Ensure recoverability by backups
- Reduce flushing round trips
- Compatible with different flush primitives [1]



[1] One-sided RDMA **FLUSH** or **READ** (*READ-after-WRITE*)

Selective Remote Flush

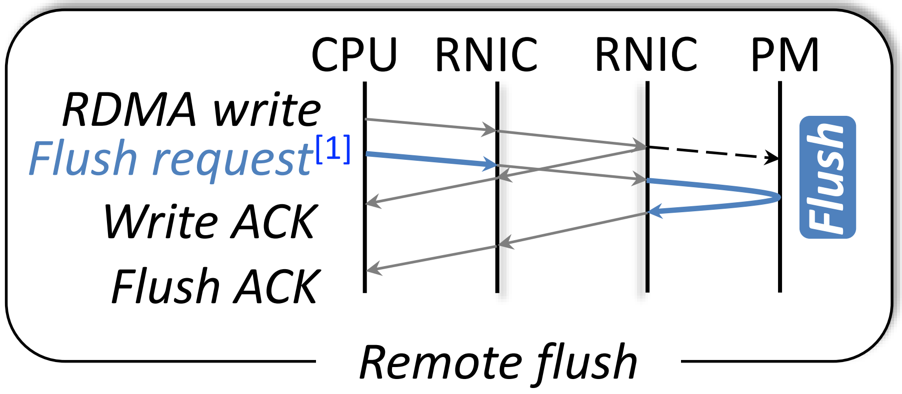
- Only issue remote flushes to **backups** after the **final** write
  - Guarantee remote persistency
  - Ensure recoverability by backups
  - Reduce flushing round trips
  - Compatible with different flush primitives [1]

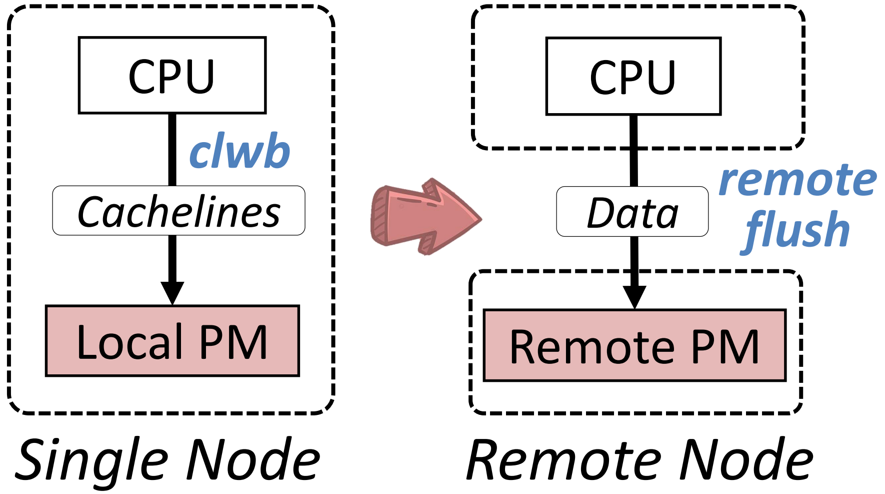


[1] One-sided RDMA **FLUSH** or **READ** (*READ-after-WRITE*)

Selective Remote Flush

- Only issue remote flushes to **backups** after the **final** write
  - Guarantee remote persistency
  - Ensure recoverability by backups
  - Reduce flushing round trips
  - Compatible with different flush primitives [1]

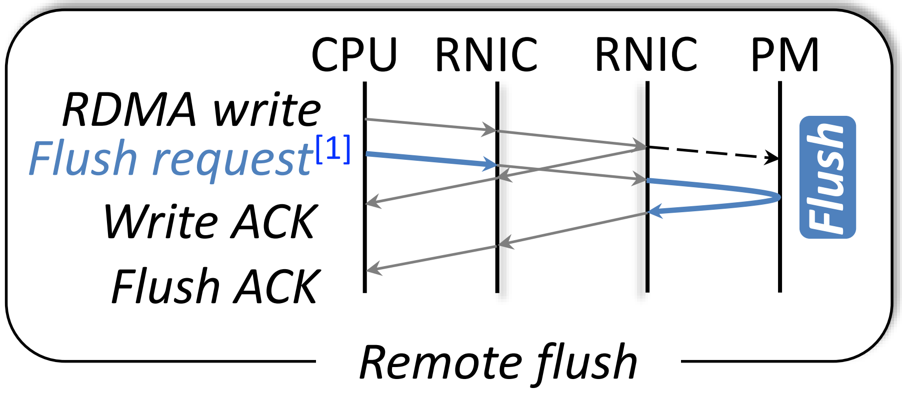


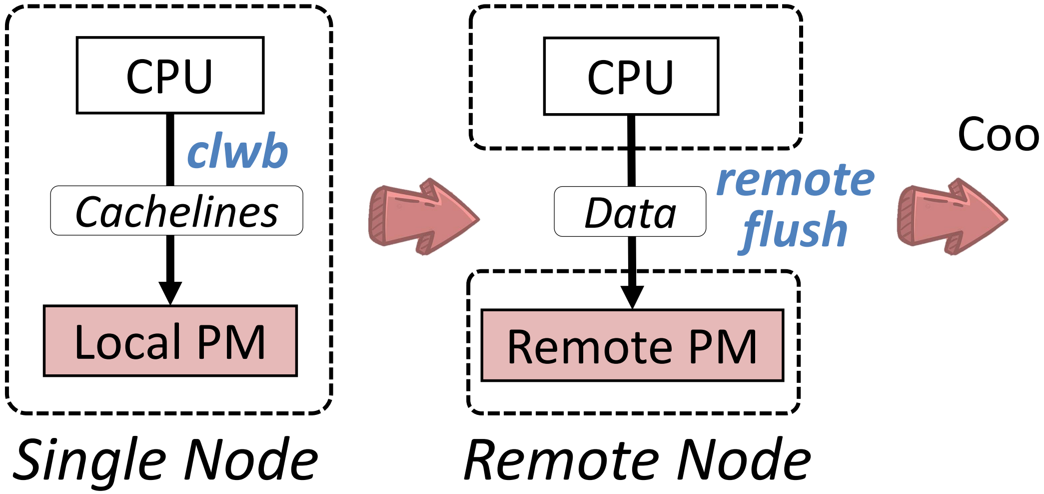


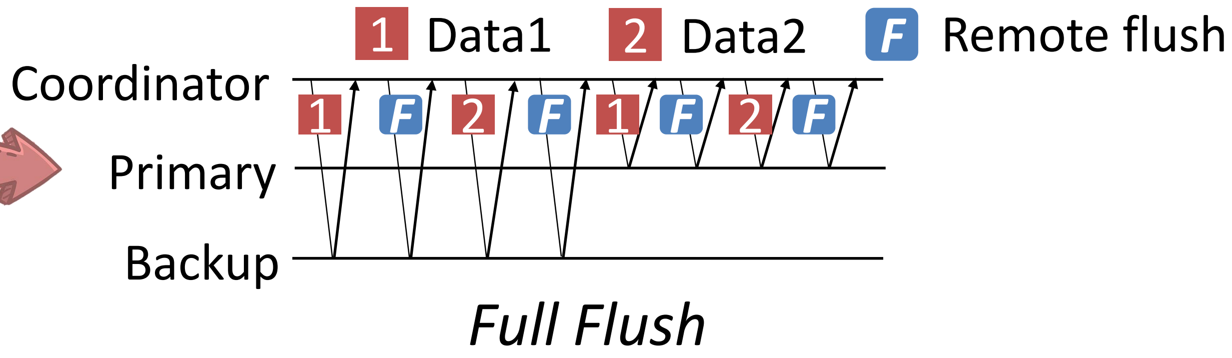
[1] One-sided RDMA **FLUSH** or **READ** (*READ-after-WRITE*)

Selective Remote Flush

- Only issue remote flushes to **backups** after the **final** write
  - Guarantee remote persistency
  - Ensure recoverability by backups
  - Reduce flushing round trips
  - Compatible with different flush primitives [1]





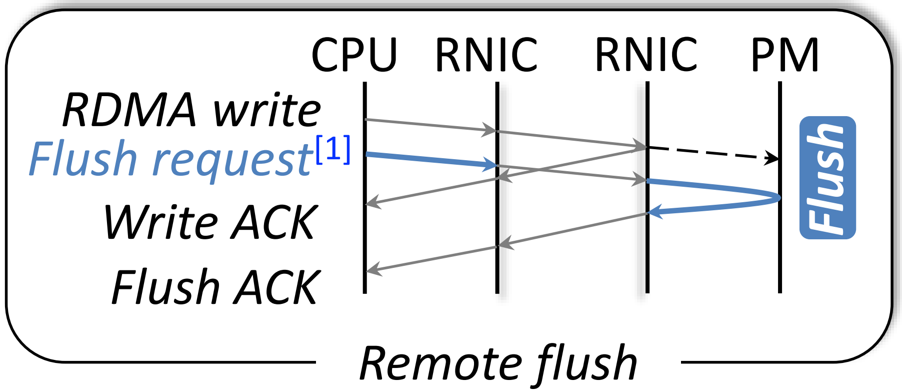


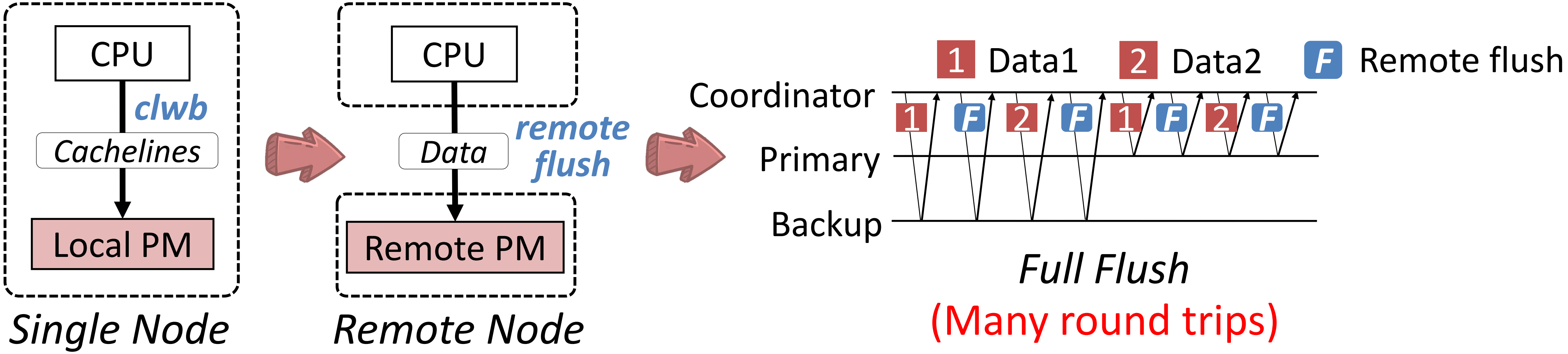
[1] One-sided RDMA **FLUSH** or **READ** (*READ-after-WRITE*)

Selective Remote Flush

➢ Only issue remote flushes to **backups** after the **final** write

- Guarantee remote persistency
- Ensure recoverability by backups
- Reduce flushing round trips
- Compatible with different flush primitives [1]

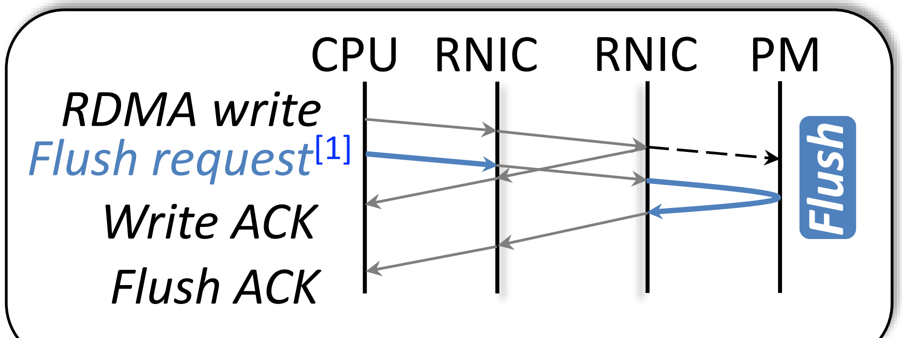




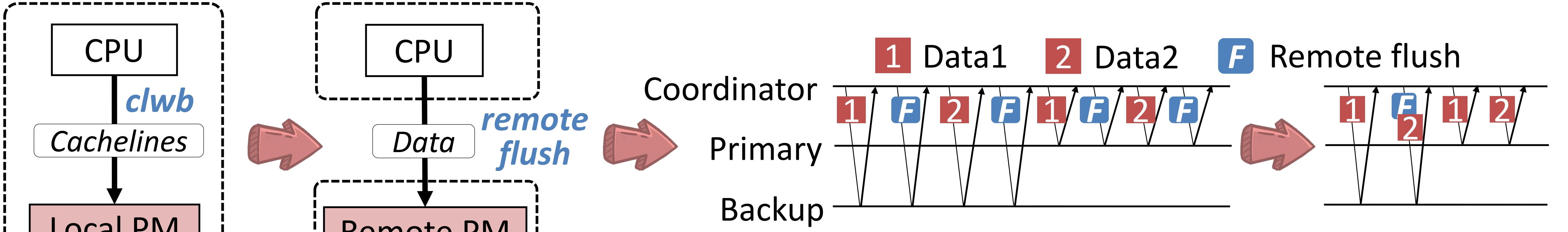
[1] One-sided RDMA **FLUSH** or **READ** (*READ-after-WRITE*)

Selective Remote Flush

- Only issue remote flushes to **backups** after the **final** write
  - Guarantee remote persistency
  - Ensure recoverability by backups
  - Reduce flushing round trips
  - Compatible with different flush primitives [1]



Remote flush



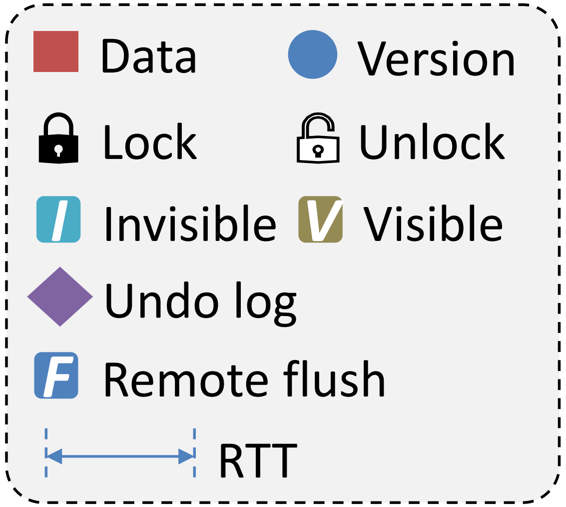
Single Node | Remote Node | Full Flush (Many round trips) | Selective Flush

[1] One-sided RDMA **FLUSH** or **READ** (*READ-after-WRITE*)

FORD Transaction

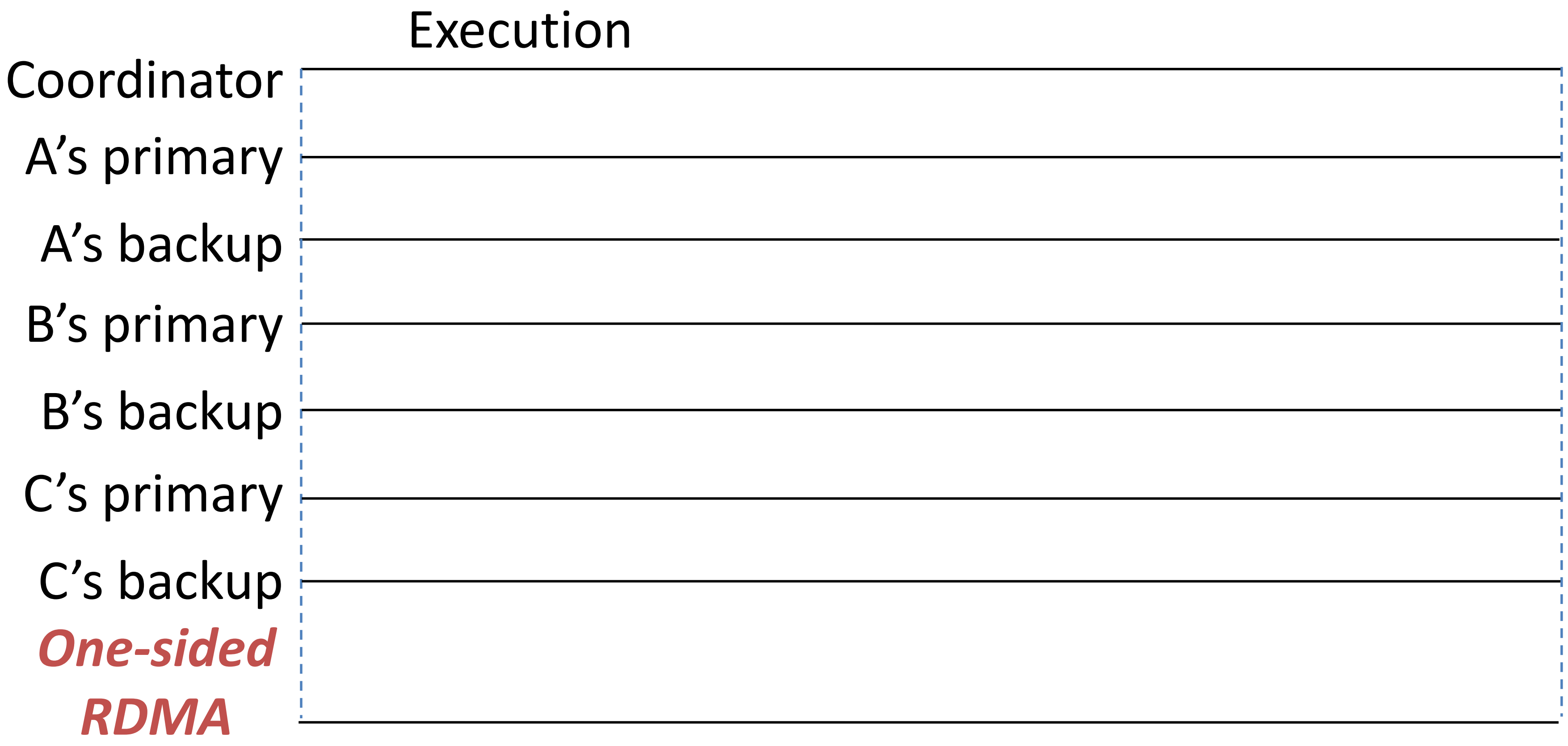
- **Put it all together:** One-sided RDMA and PM-conscious designs

Txn begin
Read A,B,C
Write A=A+C
Write B=B-C
Txn commit

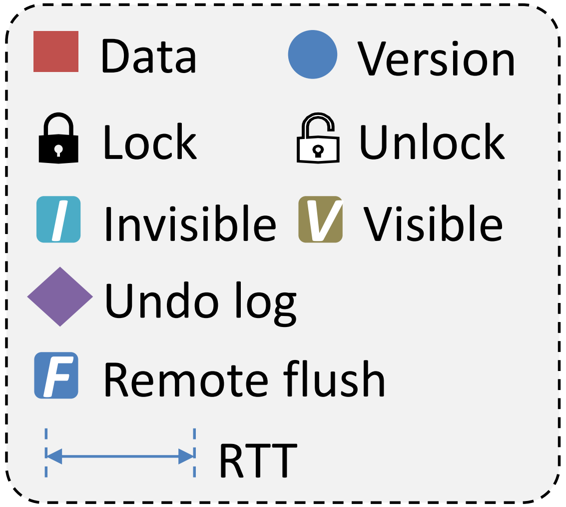


FORD Transaction

- **Put it all together:** One-sided RDMA and PM-conscious designs



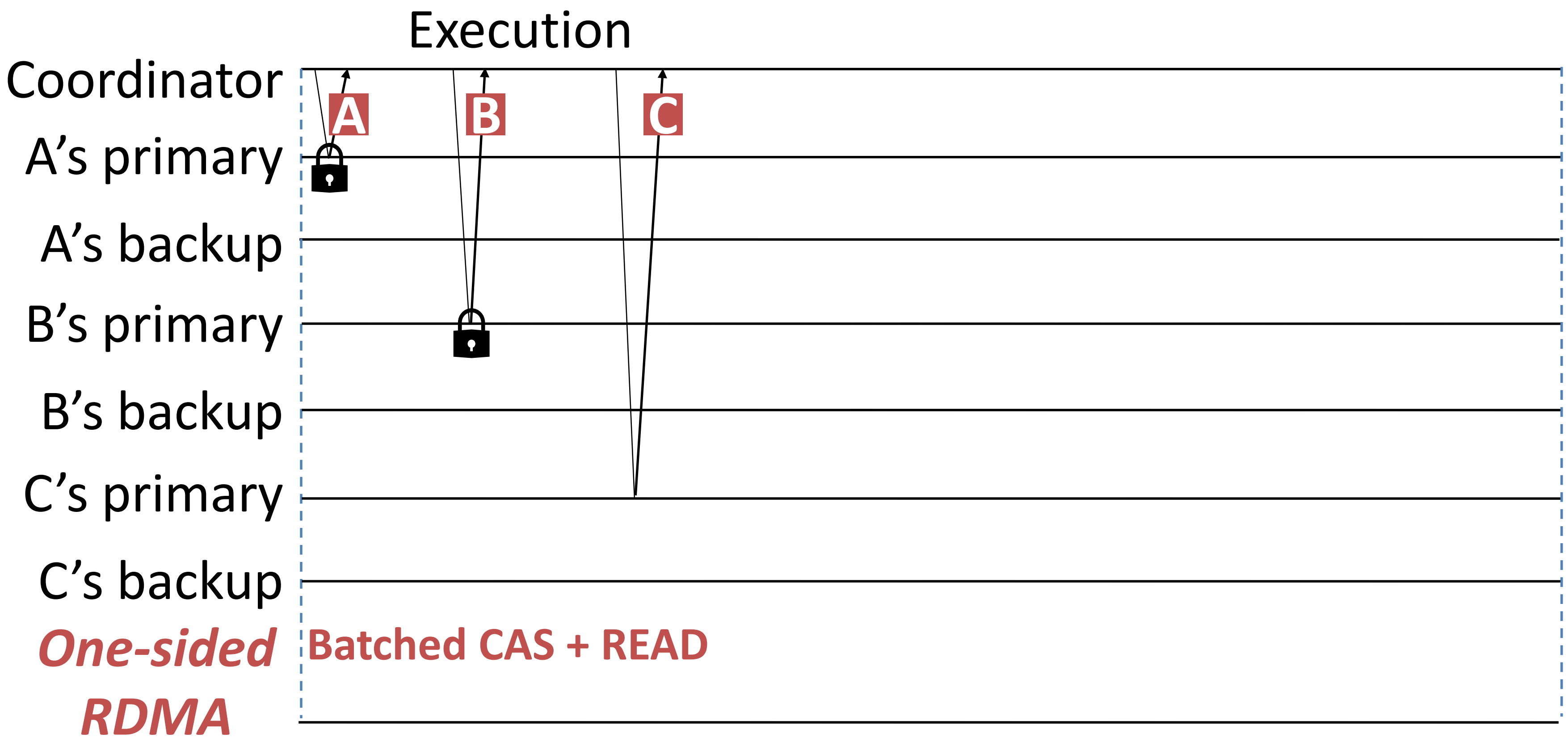
Txn begin
Read A,B,C
Write A=A+C
Write B=B-C
Txn commit

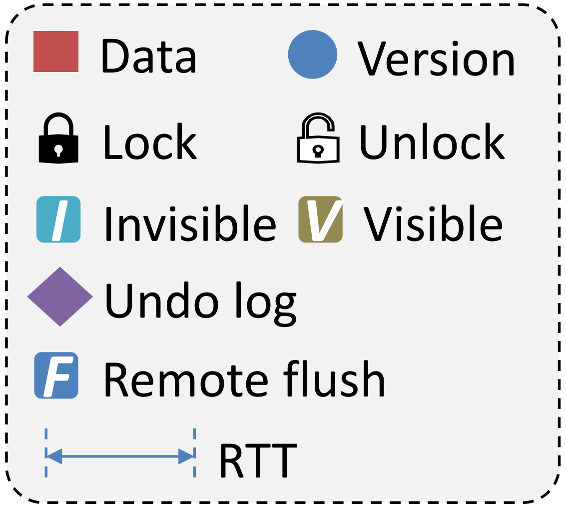


FORD Transaction

- **Put it all together:** One-sided RDMA and PM-conscious designs

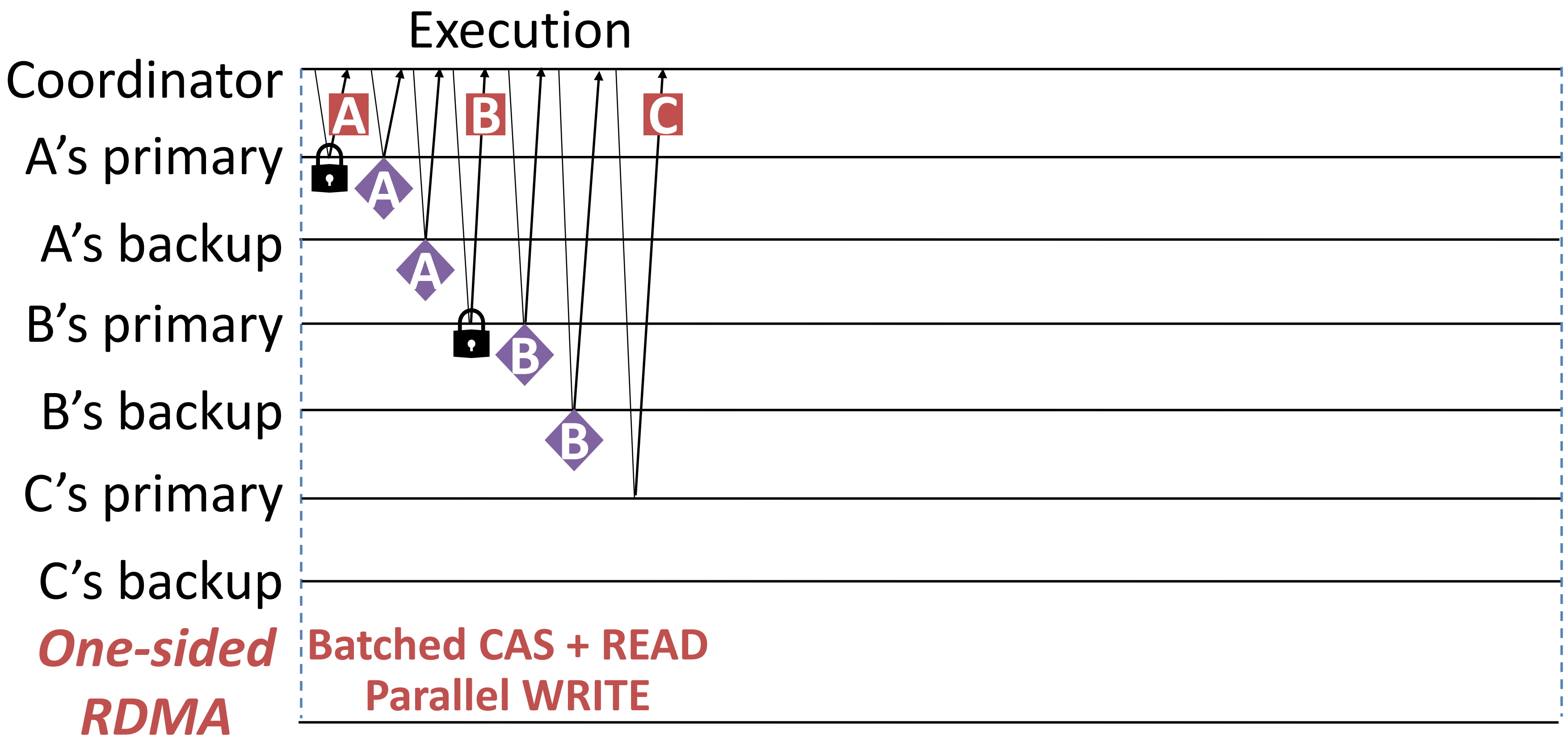
Txn begin
Read A,B,C
Write A=A+C
Write B=B-C
Txn commit



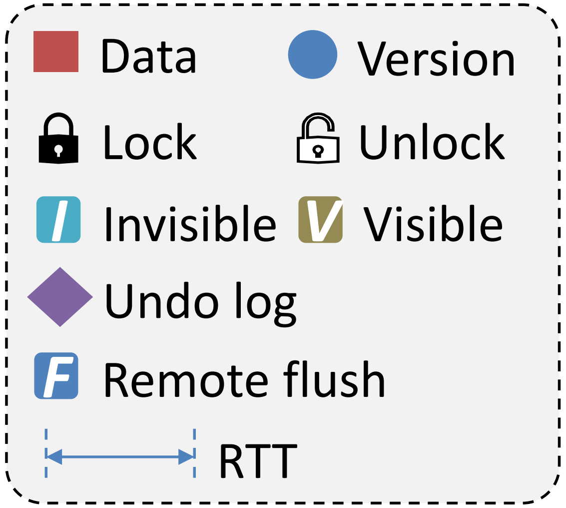


FORD Transaction

- **Put it all together:** One-sided RDMA and PM-conscious designs

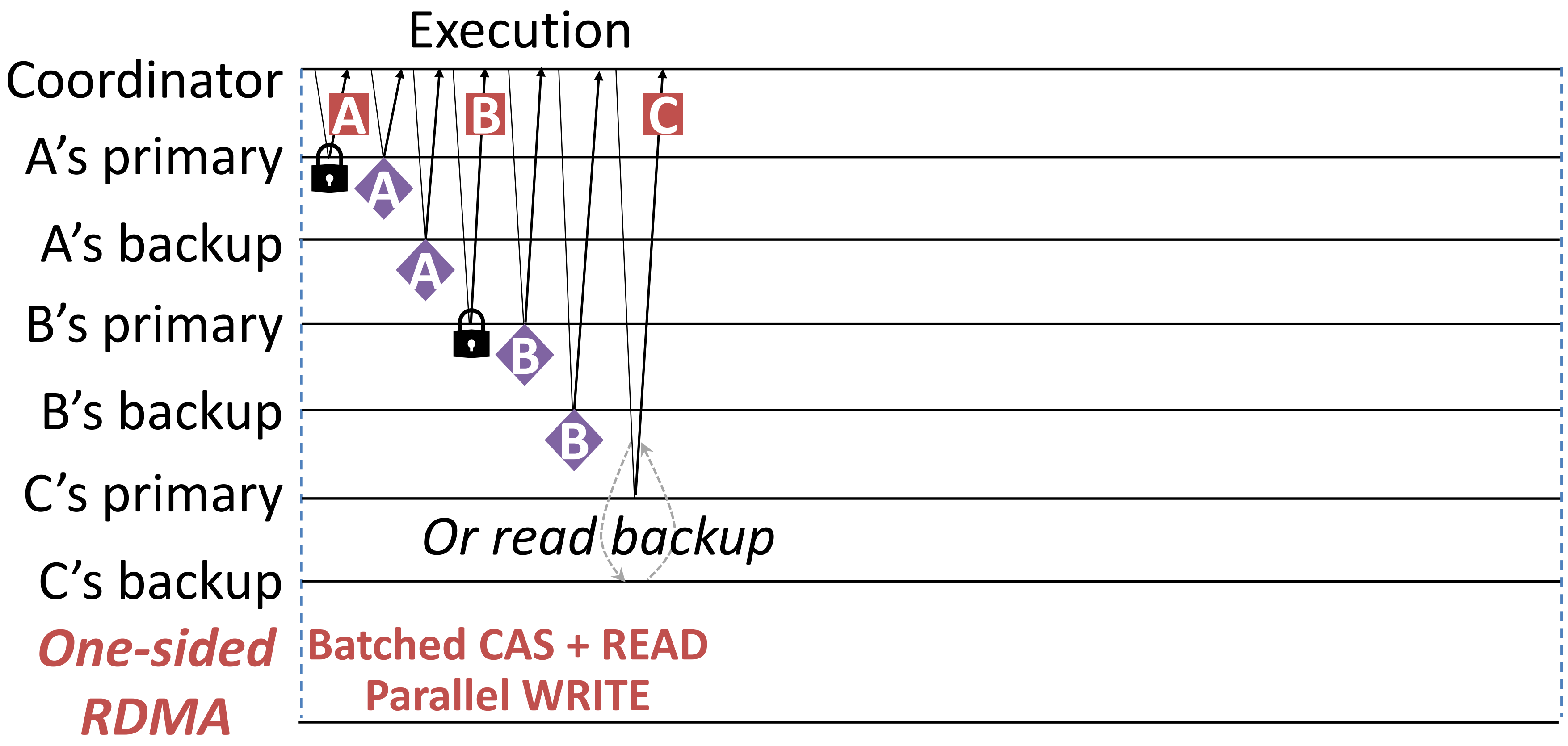


Txn begin
Read A,B,C
Write A=A+C
Write B=B-C
Txn commit

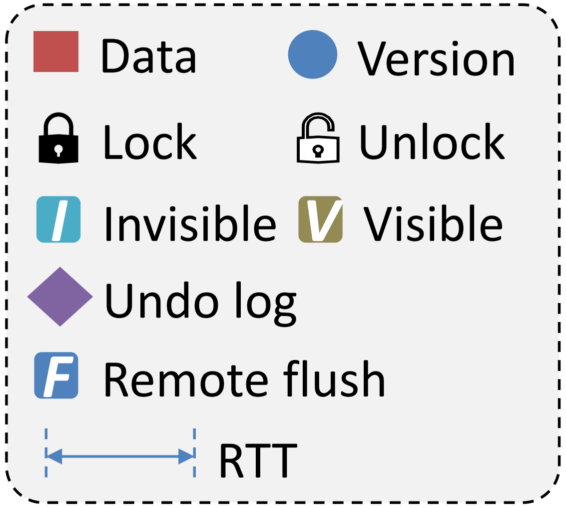


FORD Transaction

- **Put it all together:** One-sided RDMA and PM-conscious designs

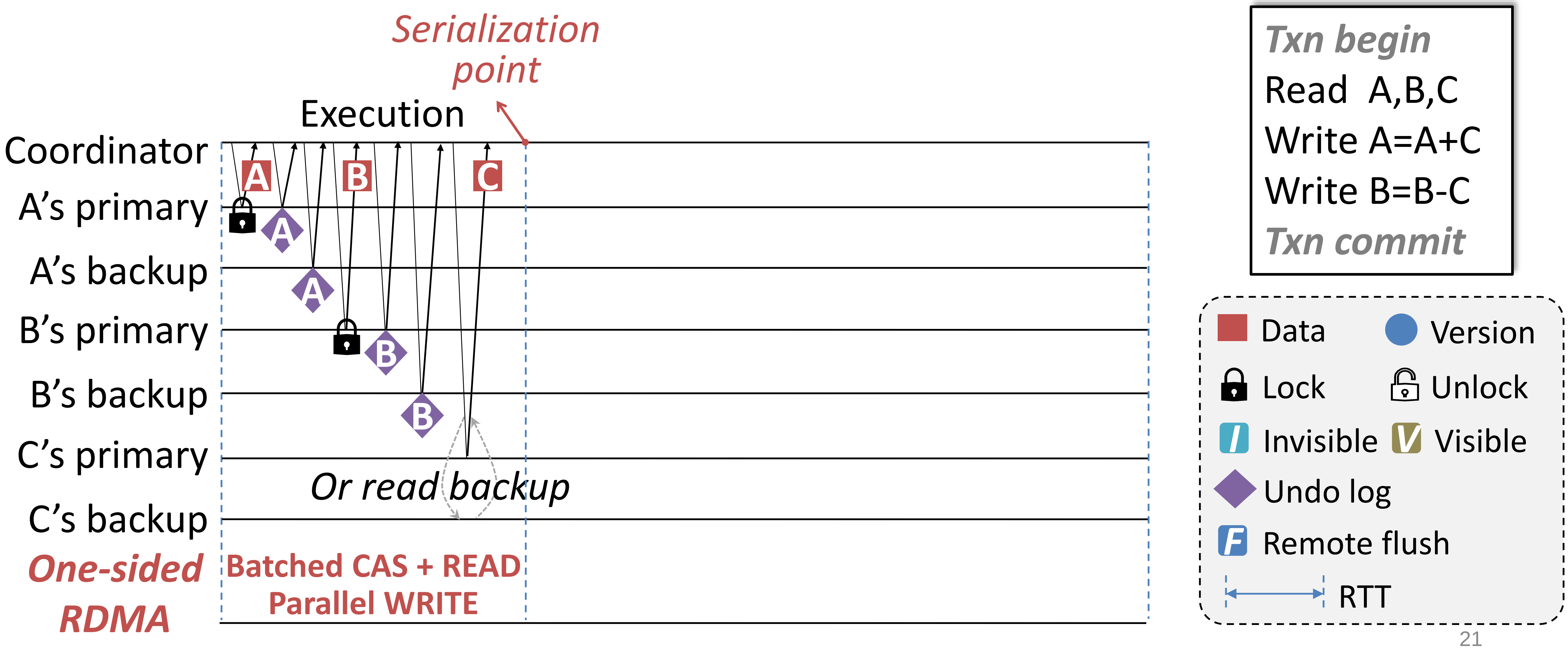


Txn begin
Read A,B,C
Write A=A+C
Write B=B-C
Txn commit



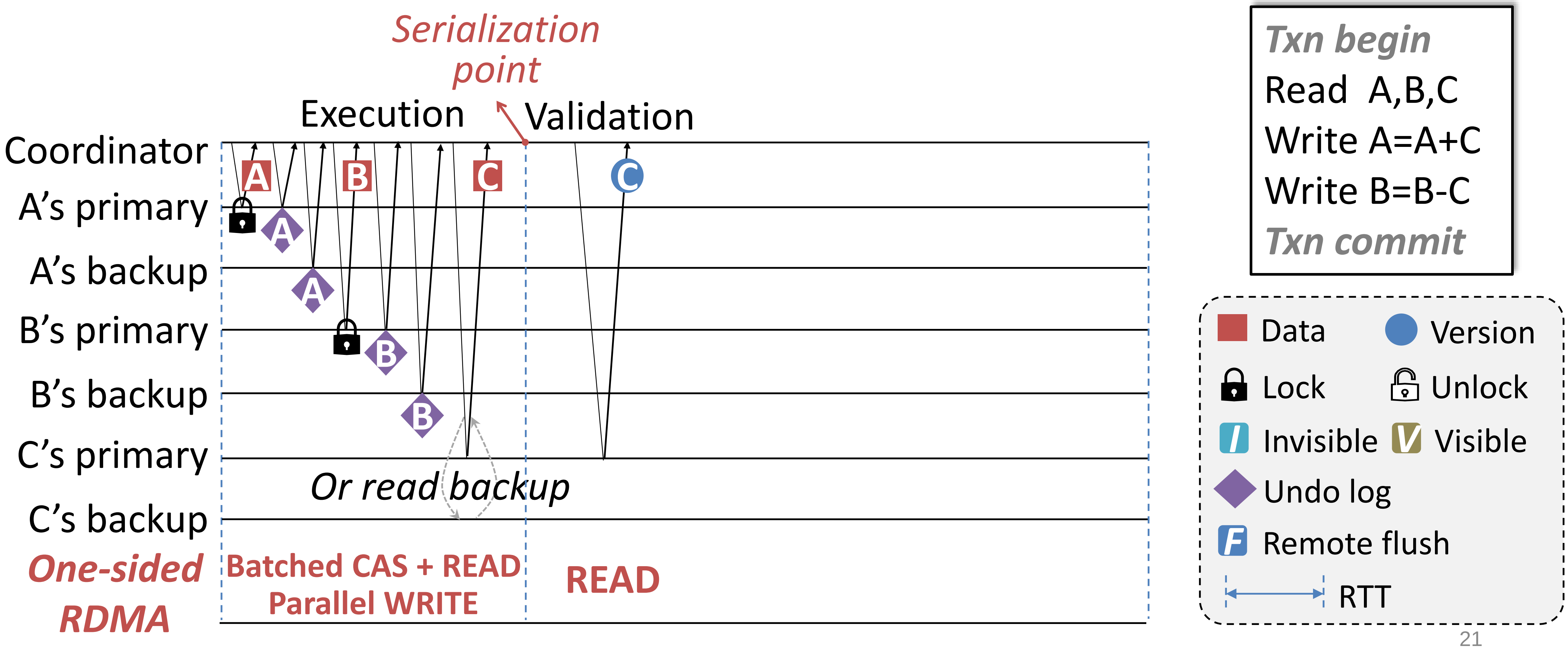
FORD Transaction

➢ **Put it all together:** One-sided RDMA and PM-conscious designs



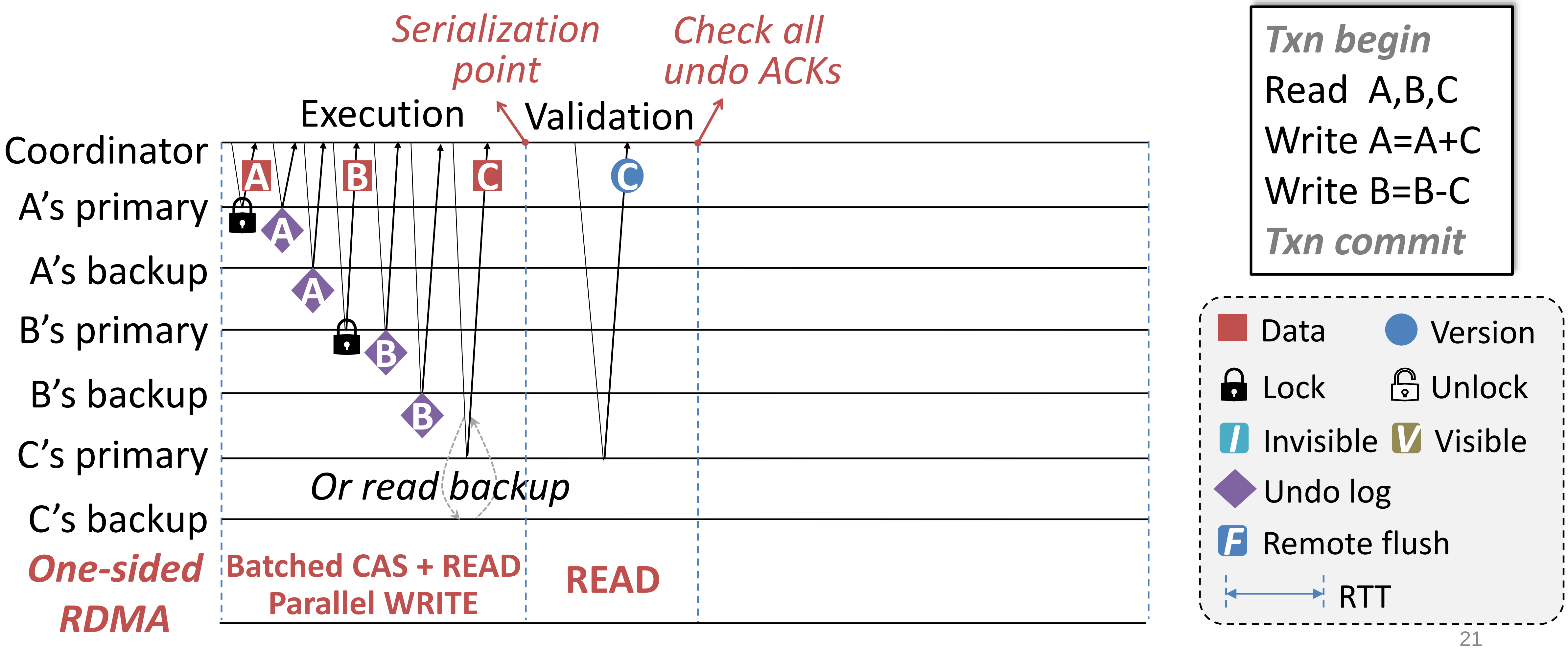
FORD Transaction

- **Put it all together:** One-sided RDMA and PM-conscious designs



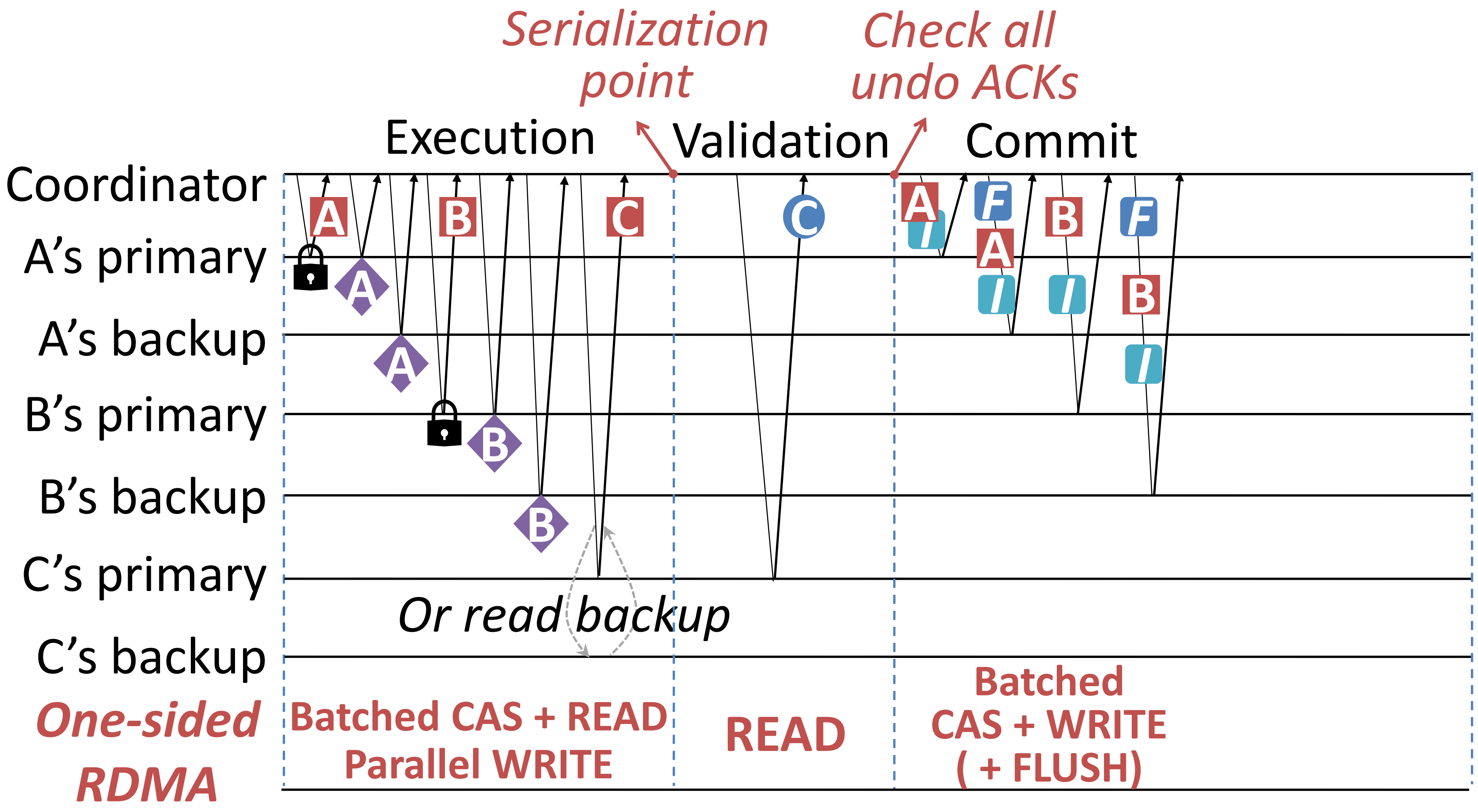
FORD Transaction

➢ **Put it all together:** One-sided RDMA and PM-conscious designs

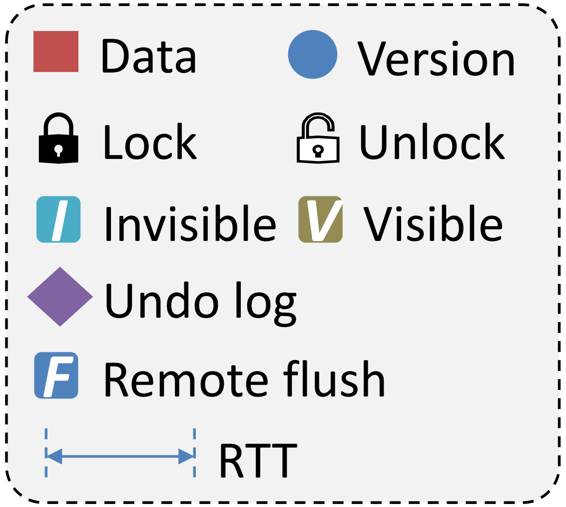


FORD Transaction

- **Put it all together:** One-sided RDMA and PM-conscious designs

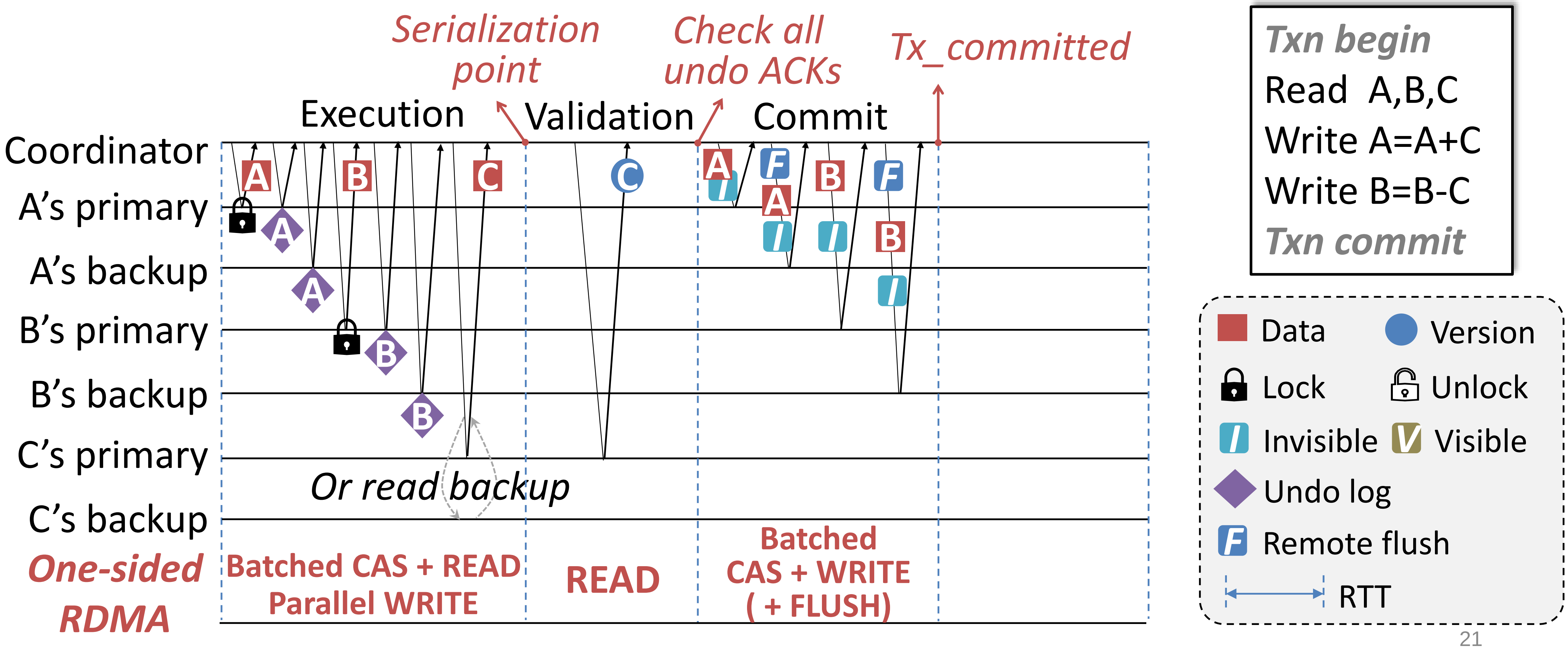


Txn begin
Read A,B,C
Write A=A+C
Write B=B-C
Txn commit



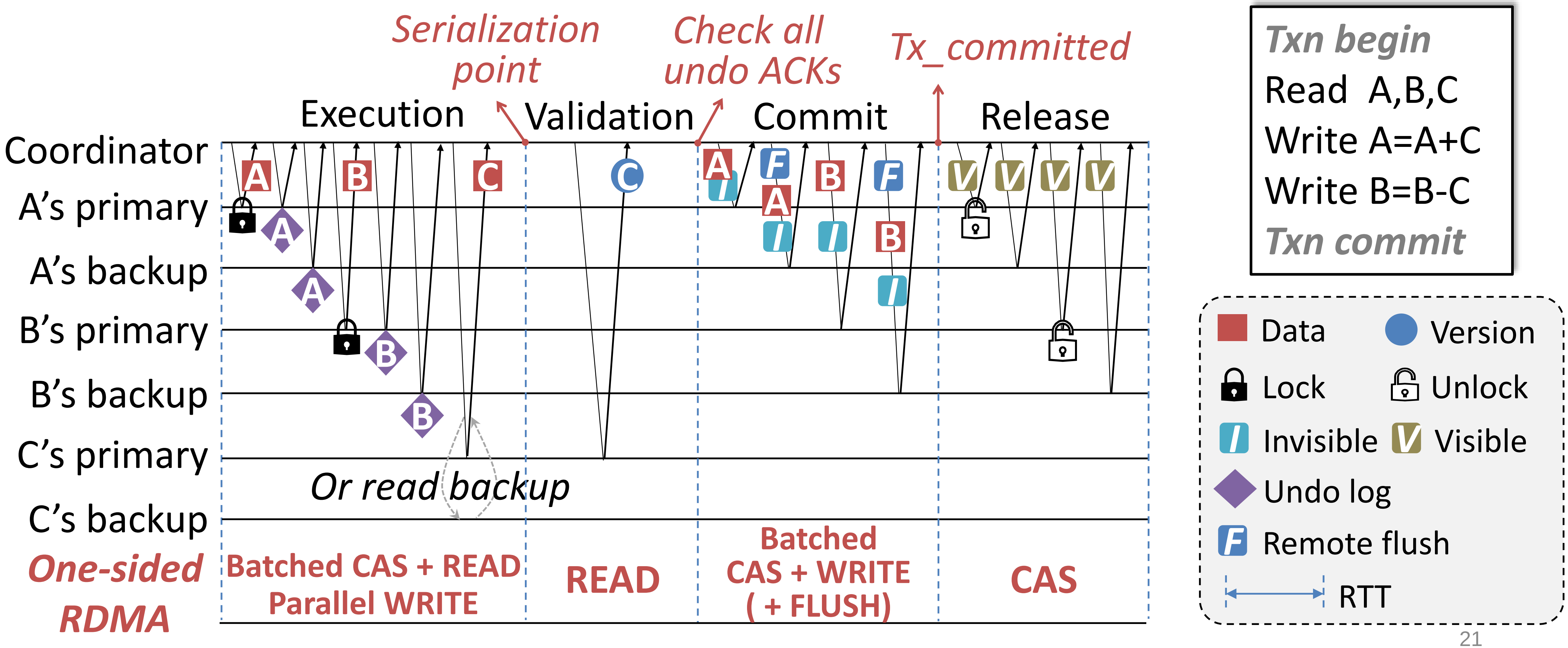
FORD Transaction

- **Put it all together:** One-sided RDMA and PM-conscious designs



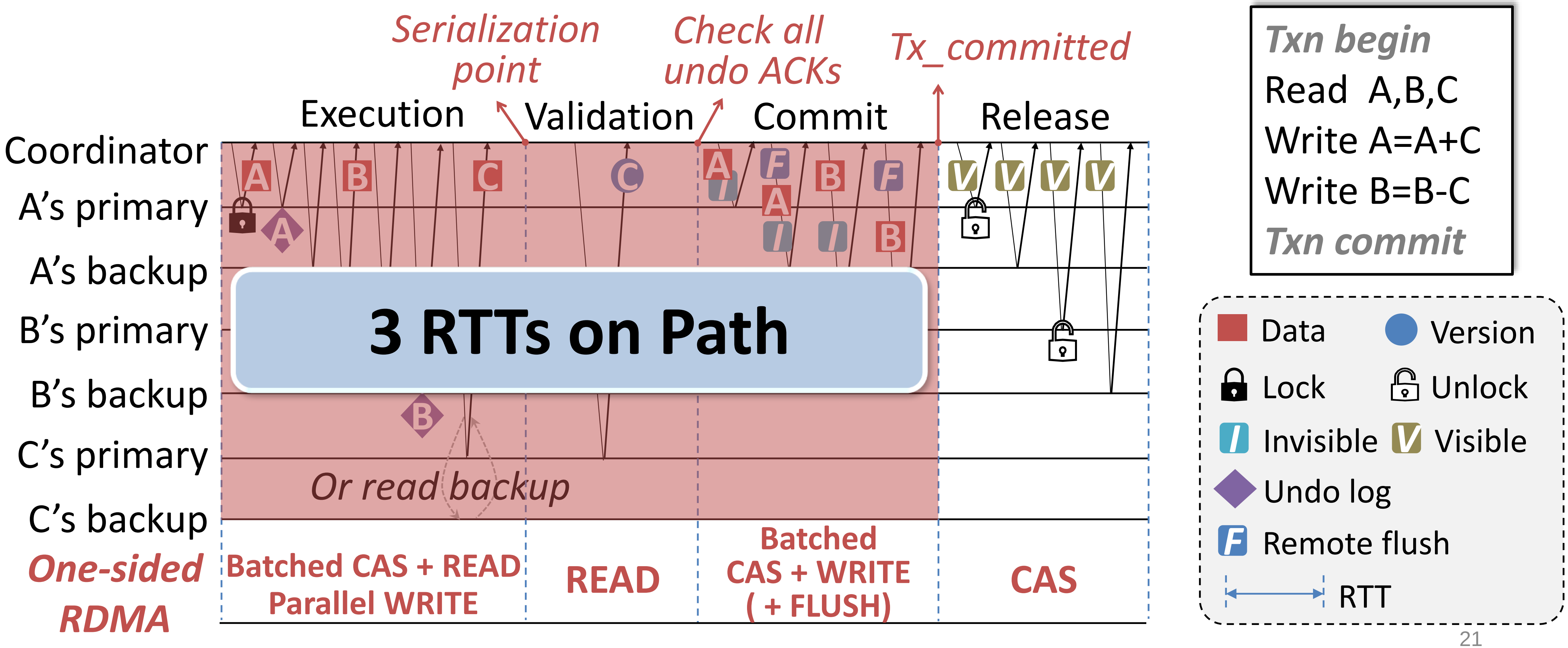
FORD Transaction

➢ **Put it all together:** One-sided RDMA and PM-conscious designs



FORD Transaction

➢ **Put it all together:** One-sided RDMA and PM-conscious designs



More Details

- ➢ Programming Interface
- ➢ Indexes in PM Pool
- ➢ Analysis on ACID and Serializability
- ➢ …

Figure 8: The hash table structure in FStore.

To enable coordinators to calculate remote addresses for one-sided RDMA in dtxn processing, the connection manager in PM pool sends the metadata (as listed below) of each hash table to the compute pool during network interconnections.

- `TableID` (8B): Global unique database table id.
- `Addr` (8B): Virtual start address of this hash table.
- `Off` (8B): Offset between `Addr` and MR's start address.
- `BucketNum` (8B): Bucket number of the hash table.
- `BucketSize` (4B): Size of a bucket (in bytes).
- `SlotNum` (4B): Number of slots per bucket.

Given the key (e.g., `K0`) of a record, if its remote address is buffered in the local cache, the coordinator directly reads the record using an RDMA `READ`. Otherwise, the coordinator reads a remote record as the following Steps:

S1: Calculate the bucket id:
`bucket_id = Hash(K0) mod BucketNum`
S2: Calculate the bucket offset in the remote MR:
`bucket_off = bucket_id × BucketSize + Off`
S3: Read the remote bucket (*bkt*) using `bucket_off`.
S4: Compare `K0` with the `SlotNum` keys in *bkt*. If a key = `K0`, the record is obtained. Then go to S7. Or else go to S5.
S5: If the `next` field of *bkt* is `NULL`, there is no such remote record. Then go to S8. Or else go to S6.
S6: Calculate the next bucket offset as below and go to S3.
`bucket_off = bkt.next − Addr + Off`
S7: Exit if the record is visible. Or else re-read it until visible.
S8: Exit with a `KEY_NOT_EXIST` hint.

Since the metadata size of a hash table is only 40B and each remote address is 8B, the local cache in compute pool can buffer all these metadata and addresses, as shown in Fig. 15b. Caching metadata is scalable, because the compute blades do not need to synchronize their metadata with each other: 1) The metadata of index does not change. 2) If the cached addresses are stale, FORD enables the coordinator to detect this and update its own cached addresses, as discussed in § 3.2.

4.2 Transaction Interfaces

FORD provides a runtime library, called *FLib*, for applications to process dtxns. Flib exposes the following interfaces:

- `TxBegin`: Start to execute a dtxn and record its id.
- `AddRO`: Add an initialized FObj to the read-only set.
- `AddRW`: Add an initialized FObj to the read-write set.
- `TxExecute`: The coordinator reads the remote data specified in read-only and read-write sets, and then executes the dtxn logic. Our hitchhiked locking and backup-enabled read schemes are leveraged.
- `TxCommit`: After execution, the coordinator commits the updated data to remote primaries and backups using our coalescent commit and selective remote flush schemes.

```
bool WriteCheck(uint64_t dtxn_id, DTXN* dtxn) {
  // The `dtxn` invokes FLib interfaces
  dtxn->TxBegin(dtxn_id);
  // Use a random account as the key
  uint64_t acct_id = RandomAccount();
  FObj* sav_obj = new FObj(SavingsTableID, acct_id);
  FObj* chk_obj = new FObj(CheckingTableID, acct_id);
  dtxn->AddRO(sav_obj);
  dtxn->AddRW(chk_obj);
  if (!dtxn->TxExecute()) return false;
  // Get record values and run transaction logic
  sav_val_t* sav = (sav_val_t*) sav_obj->value;
  chk_val_t* chk = (chk_val_t*) chk_obj->value;
  if (sav->balance + chk->balance < PredefinedAmount)
    chk->balance -= (PredefinedAmount + 1);
  else chk->balance -= PredefinedAmount;
  bool status = dtxn->TxCommit();
  delete sav_obj; delete chk_obj;
  // Report commit (true) or abort (false) to client
  return status;
}
```

Figure 9: The example of C++ code using FLib interfaces.

Figure 10: The comparisons between (a) sequential processing, and (b) interleaved processing, in one thread.

Our transaction interfaces support general transaction processing. Specifically, the developers are not required to know all the read/write sets at the beginning of each transaction. Instead, developers call `AddRO`, `AddRW`, and `TxExecute` multiple times when reading/writing data occurs during a transaction. Fig. 9 illustrates an example of using our interfaces in the `Write Check` transaction of the SmallBank benchmark [50]. This transaction reads the balances from the `Savings` and `Checking` tables, and updates the balance in the `Checking` table. It shows that our interfaces are easy-to-use.

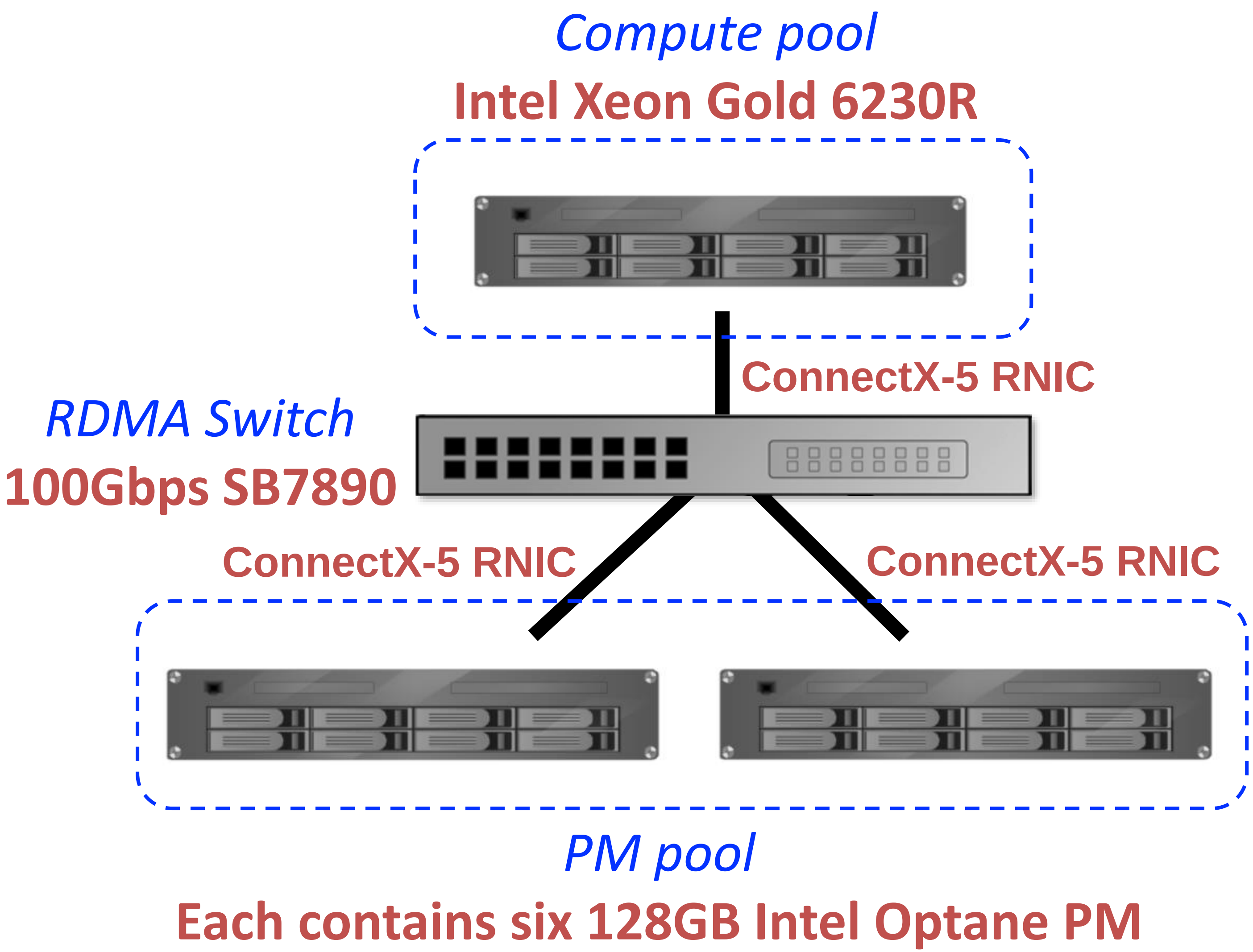
4.3 Interleaved Transaction Processing

As shown in Fig. 10a, sequentially processing dtxns in a thread wastes the CPU cycles due to waiting for RDMA ACKs, which significantly decreases the throughput. To avoid CPU idling in the compute pool, FORD leverages an interleaved processing scheme that enables multiple coordinators in one thread to process different dtxns in pipeline, as presented in Fig. 10b. In this way, the network RTTs are efficiently hided and the CPU cores in the compute pool are fulled utilized to improve the throughput.

We use coroutines [29, 60] to implement the interleaved processing. Each CPU thread generates several coroutines and each coroutine acts as a coordinator to execute dtxns. After issuing the RDMA requests, a coroutine yields its CPU core to another coroutine to process the next dtxn. A dedicated coroutine in each thread polls RDMA ACKs. If all ACKs of a coroutine arrived, FORD schedules this coroutine to occupy the CPU core to resume execution. The results in Fig. 16 show that using a proper number of coroutines improves the throughput without heavily increasing the latency.

Evaluation

- **Benchmarks**
  - KV store
    - 8B key + 40B value
    - Skewed (θ = 0.99) + Uniform
  - TATP
    - RO/RW: 80%/20%, 48B
  - SmallBank
    - RO/RW: 15%/85%, 16B
  - TPCC
    - RO/RW: 8%/92%, 672B
- **Comparisons**[1]
  - FaRM@SOSP’15
  - DrTM+H@OSDI’18

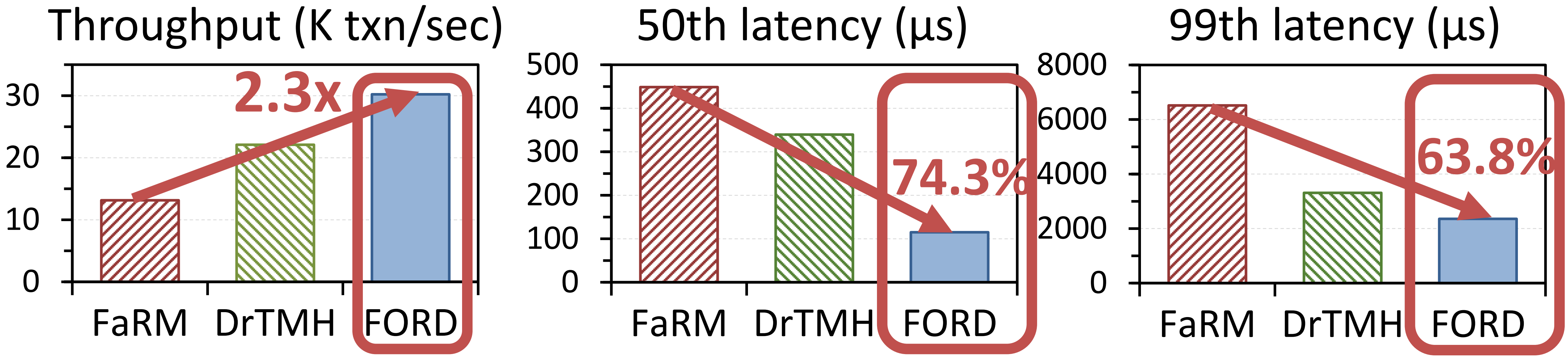


Testbed

[1] Protocols are re-implemented using one-sided RDMA

End-to-End Performance

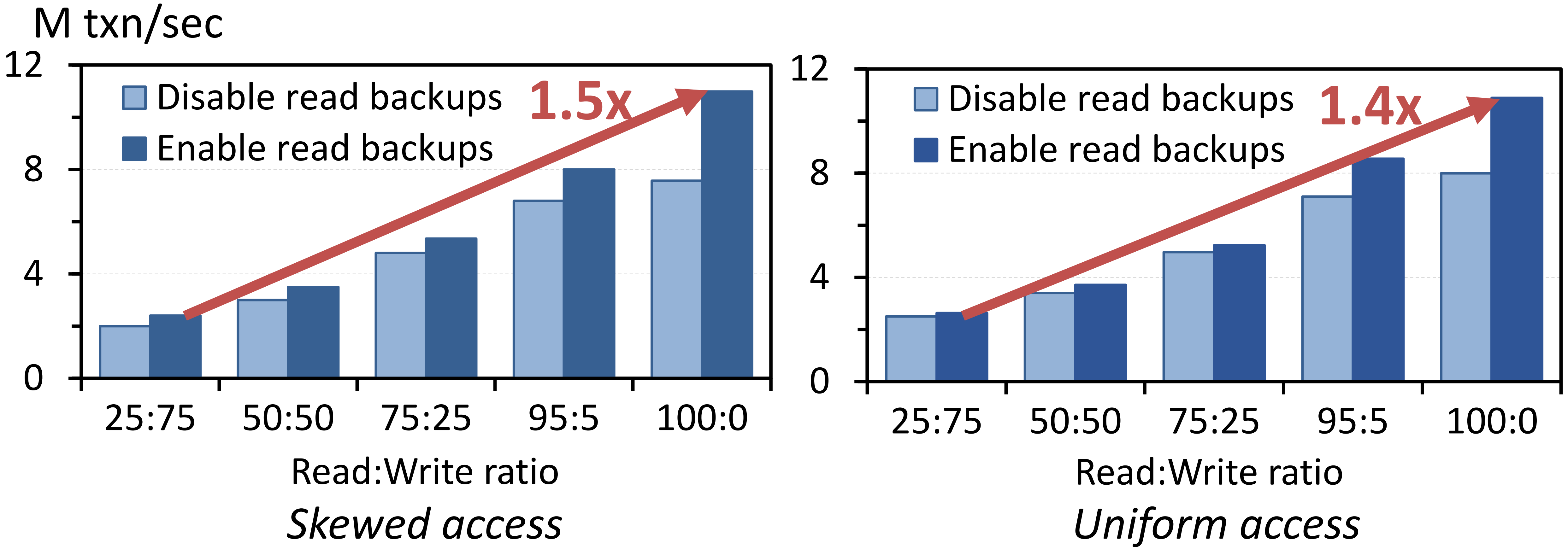
- 112 coordinators
- **Efficient round trip reduction and backup utilization**



TPCC results

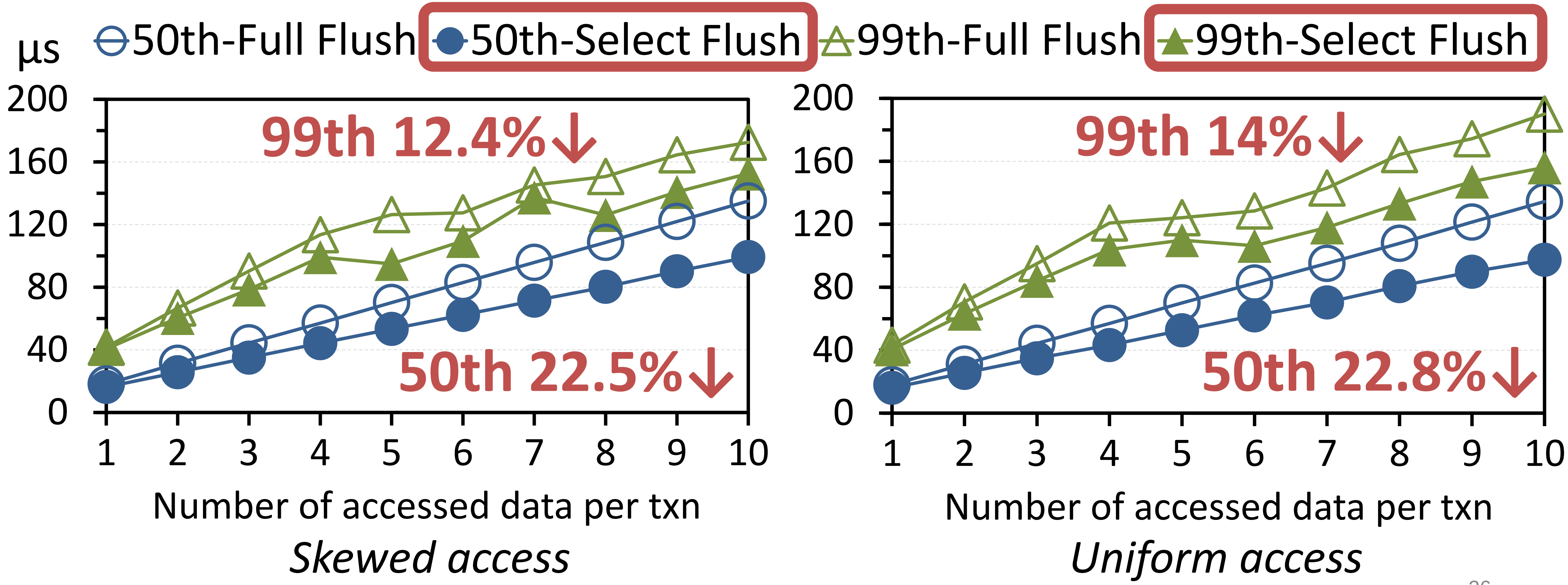
Read backup

- KV store, 1 backup

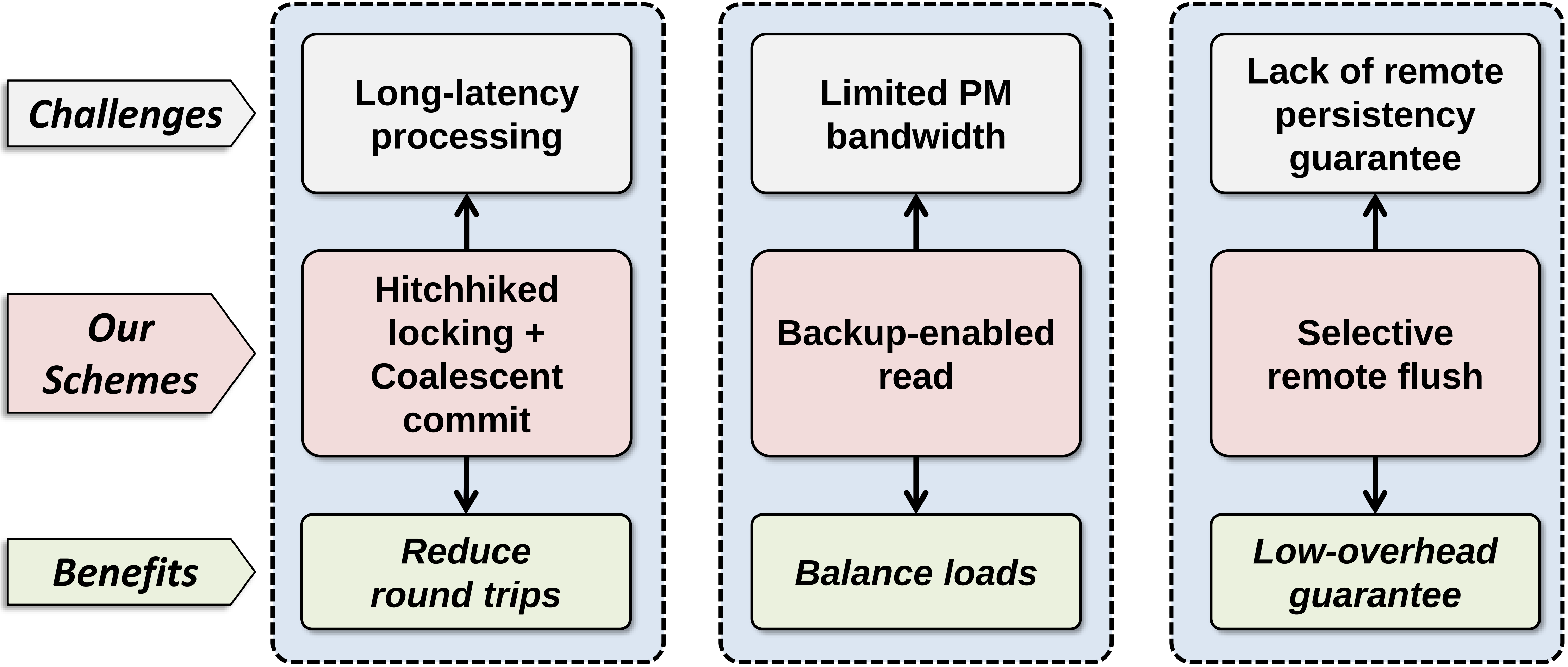


Remote flush

- KV store, 1 coordinator



Conclusion



https://github.com/minghust/ford

Thank you! Q&A