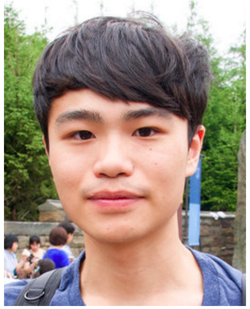
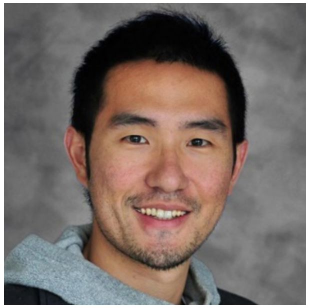
MgCrab

Transaction Crabbing for Live Migration in Deterministic Database Systems





Yu-Shan Lin[1] Shao-Kan Pi[1] Meng-Kai Liao[1] Ching Tsai[1] Aaron J. Elmore[2] Shan-Hung Wu[1]

National Tsing Hua University, Taiwan[1]

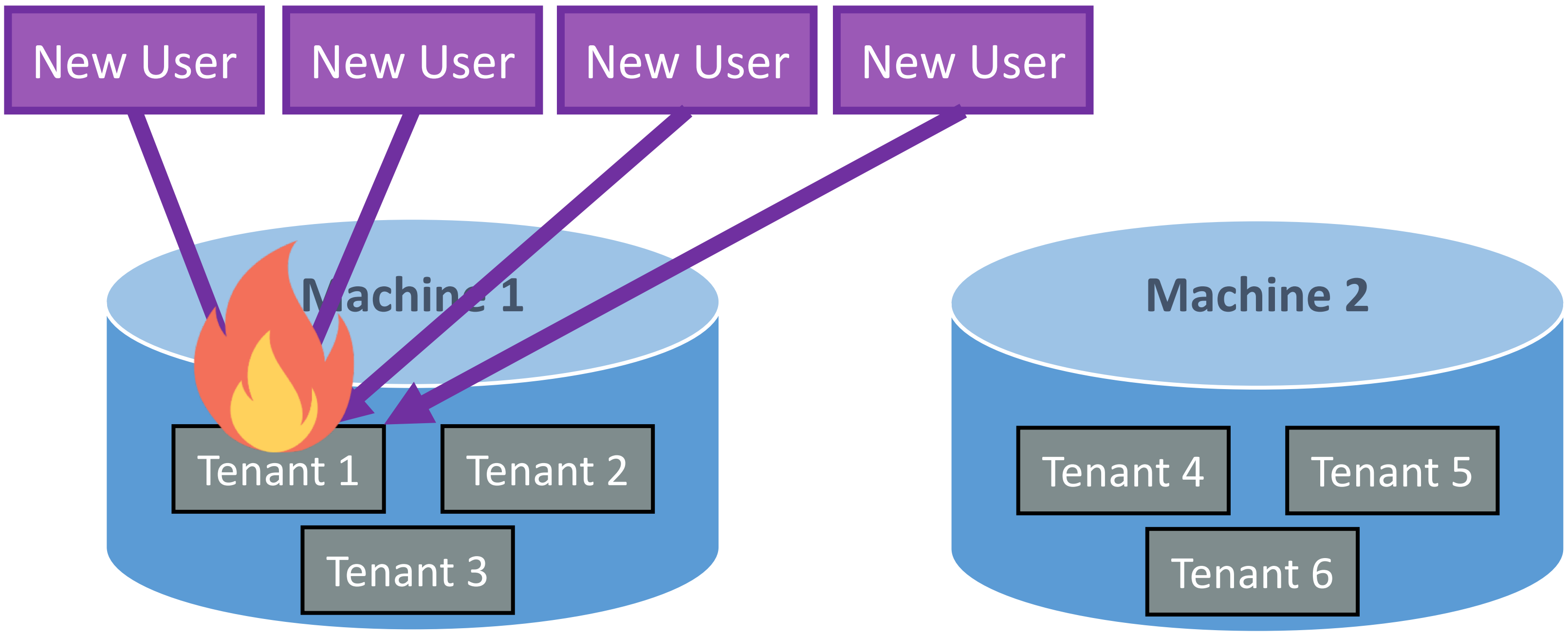
University of Chicago [2]

Outline

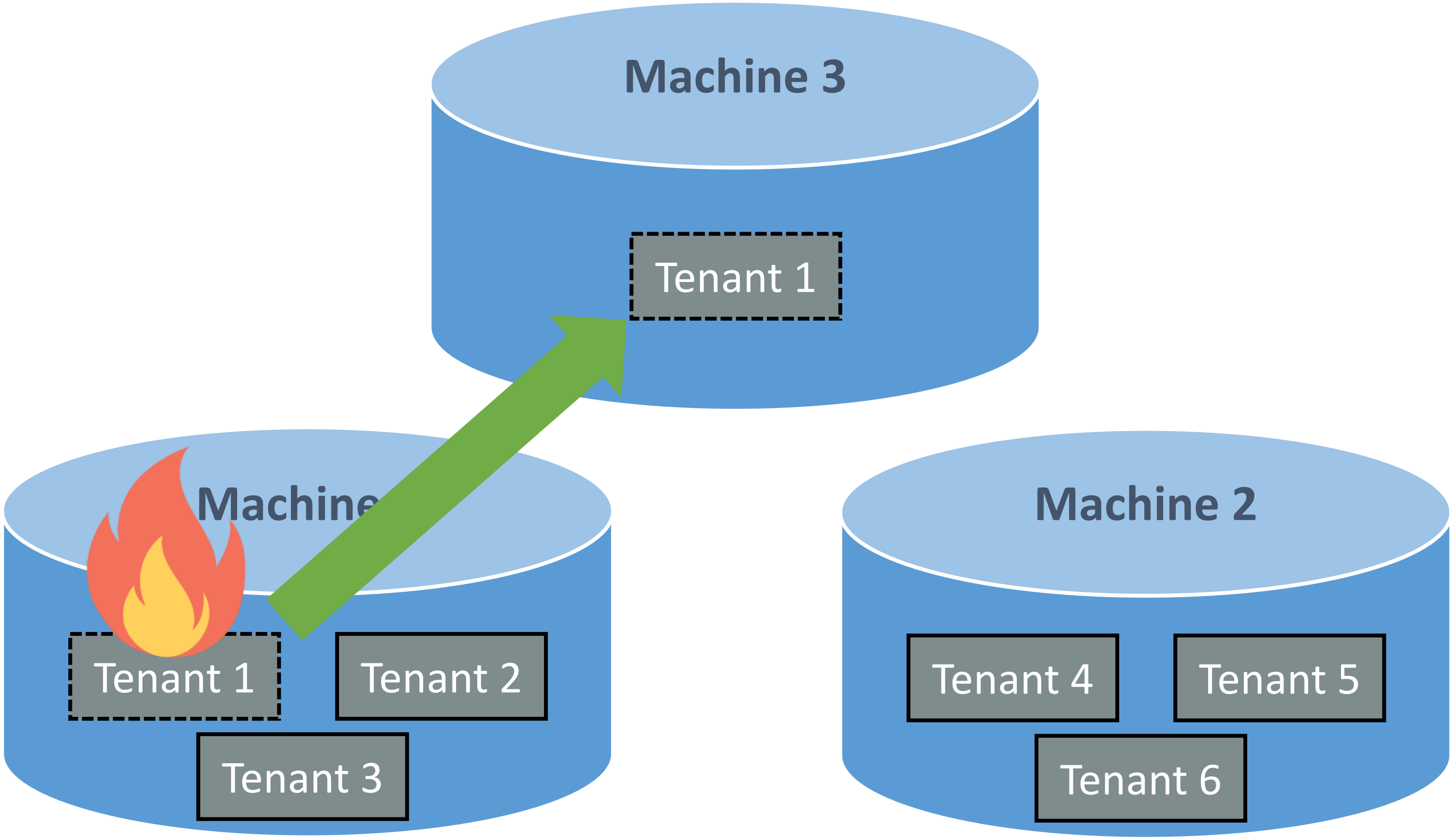
- Background
  - Elastic Load Balancing
- Related Work
- MgCrab
- Experiments Results
- Conclusion

Motivation: Hot Tenants

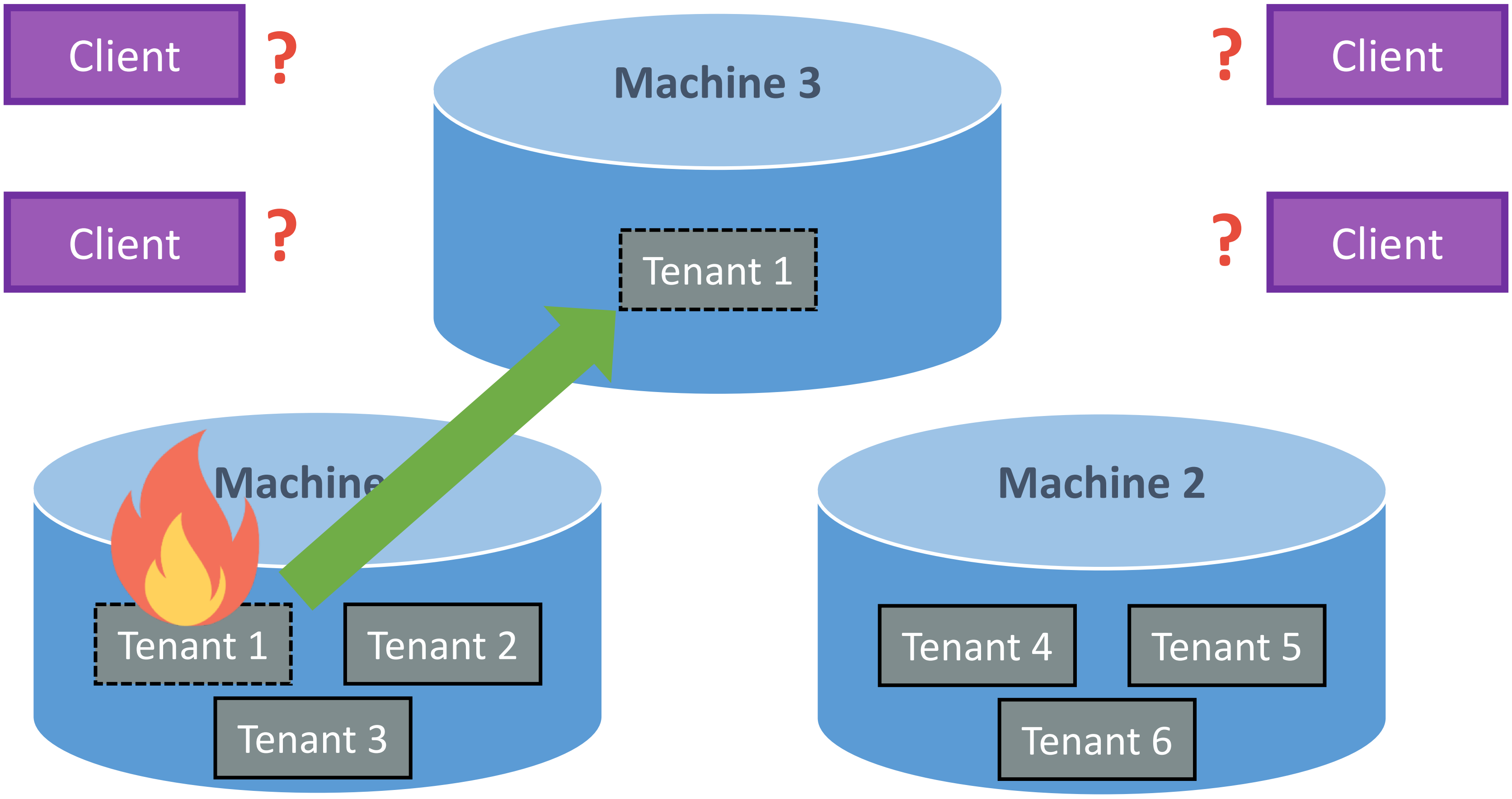
- An application gains flash crowds originating from viral popularity.



Solution: Adding More Resource by Increasing Provisioning Machines



How to Move Data On The Fly While Keeping Serving Transactions?



For such cases, we need **Live Migrations** techniques!

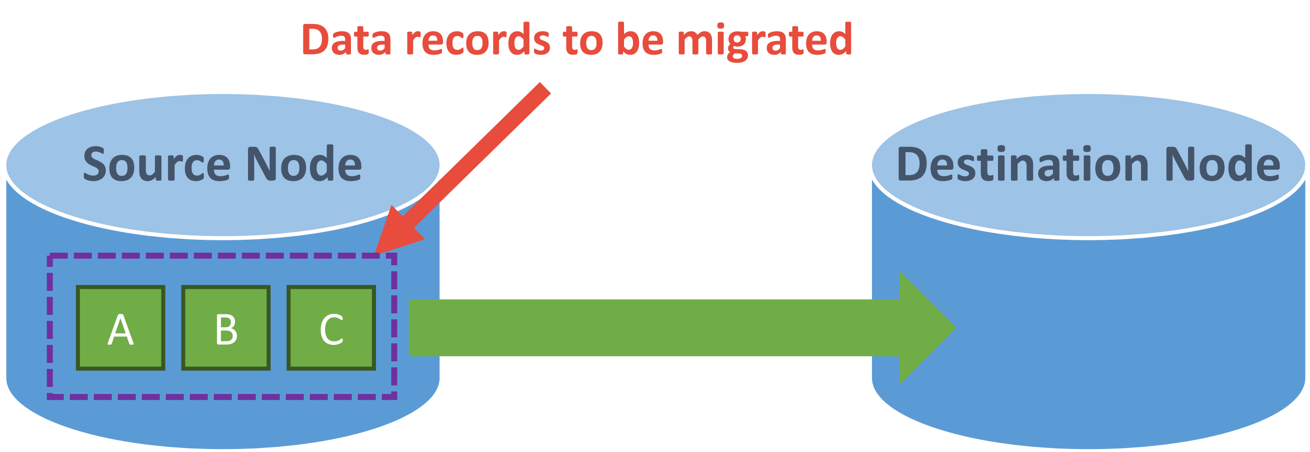
Given a migration plan, to migrate data from the source node to the destination node while **continuously serving** incoming transactions.

Outline

- Background
- Related Work
  - Source-based Approaches
  - Destination-based Approaches
  - Either-node Approaches
- MgCrab
- Experiments Results
- Conclusion

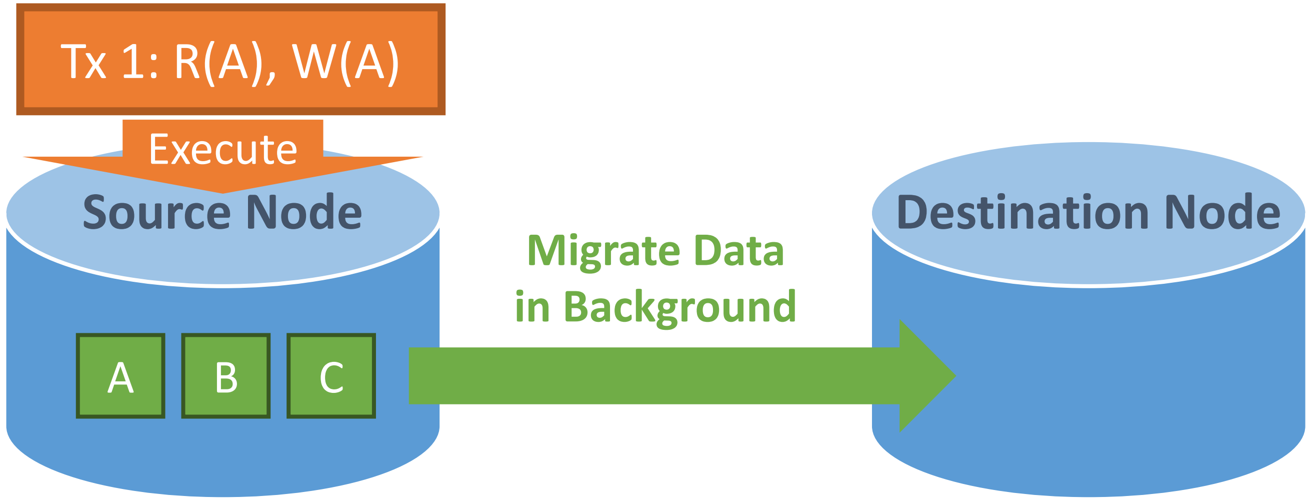
3 Variants

- Based on how they serve incoming transactions…
  - Source-based approaches
  - Destination-based approaches
  - Either-node approaches



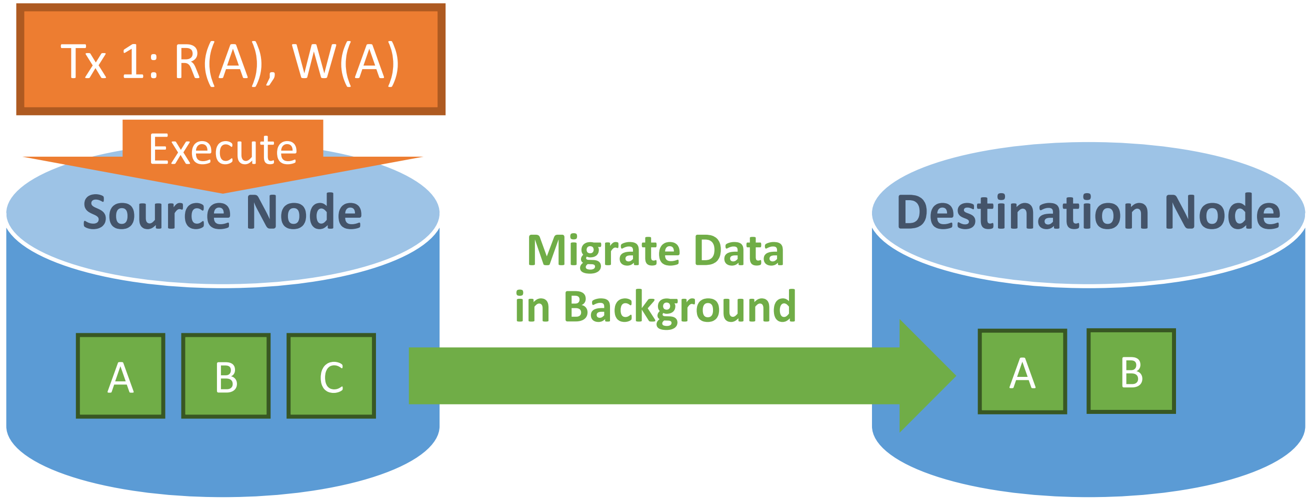
Source-based Approaches

- Pro
  - The cache is warm
  - Data are likely available on source nodes.
- Example: Albatross [VLDB’11]



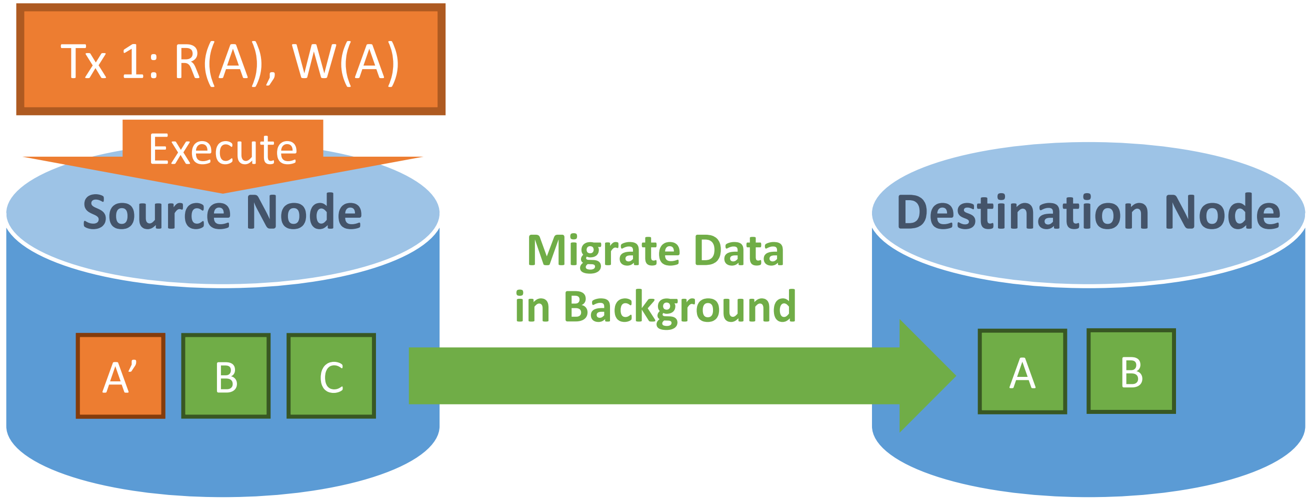
Con: Termination Problem

- Updates are always on the source node.
  - Need to be sent to the destination node.



Con: Termination Problem

- Updates are always on the source node.
  - Need to be sent to the destination node.

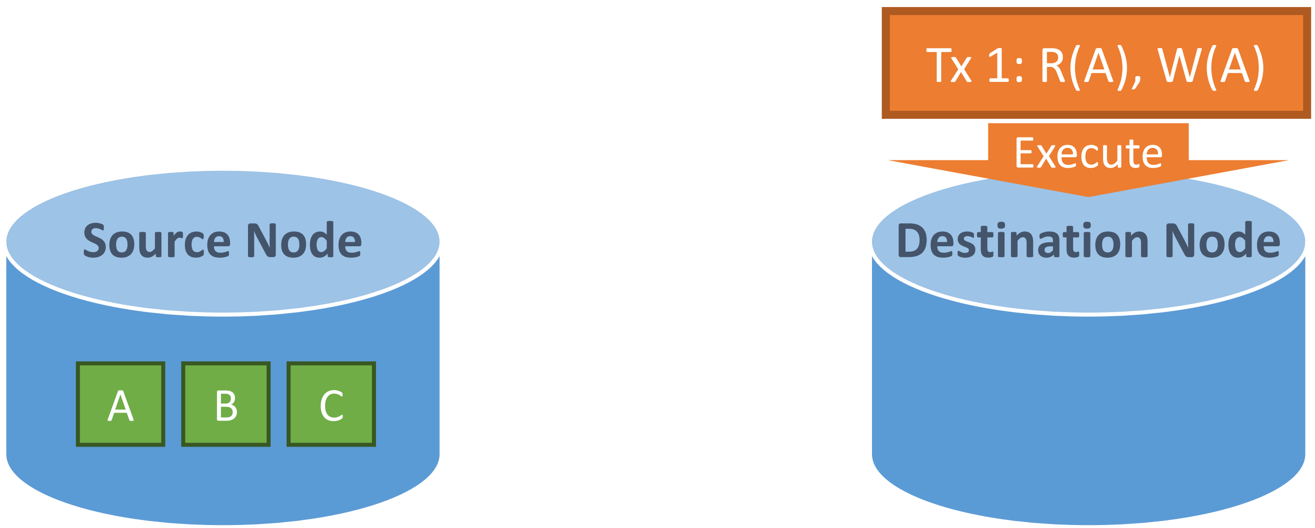


Con: Termination Problem

- Updates are always on the source node.
  - Need to be sent to the destination node.
- Needs a stop-and-copy in the end => service downtime

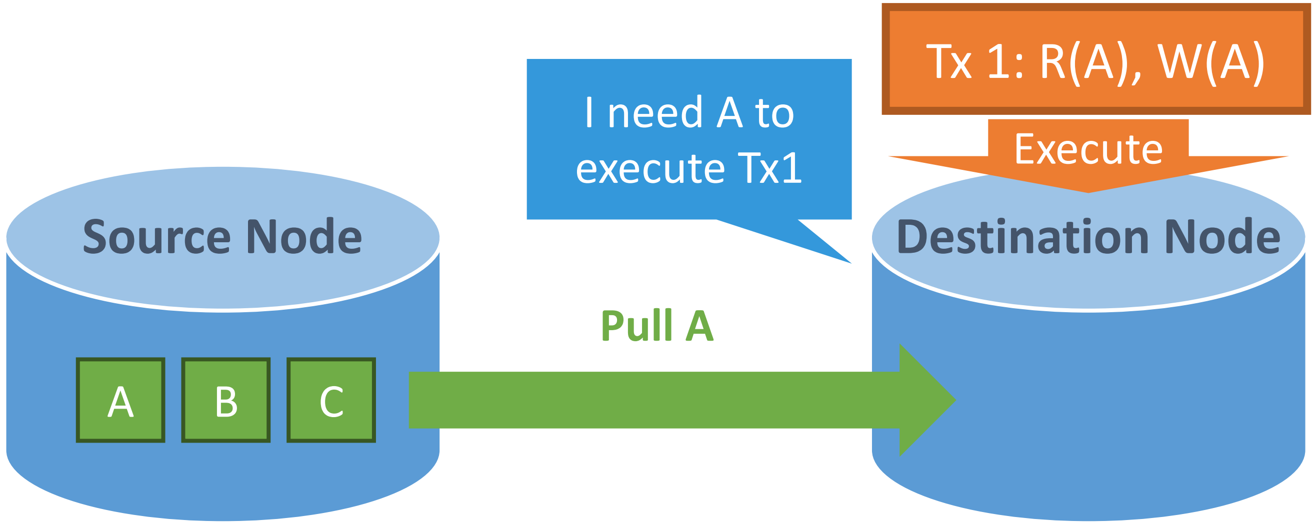
Destination-based Approaches

- Pros
  - Always terminates since it doesn't have to migrate any update.
- Example: Zephyr [SIGMOD'12]



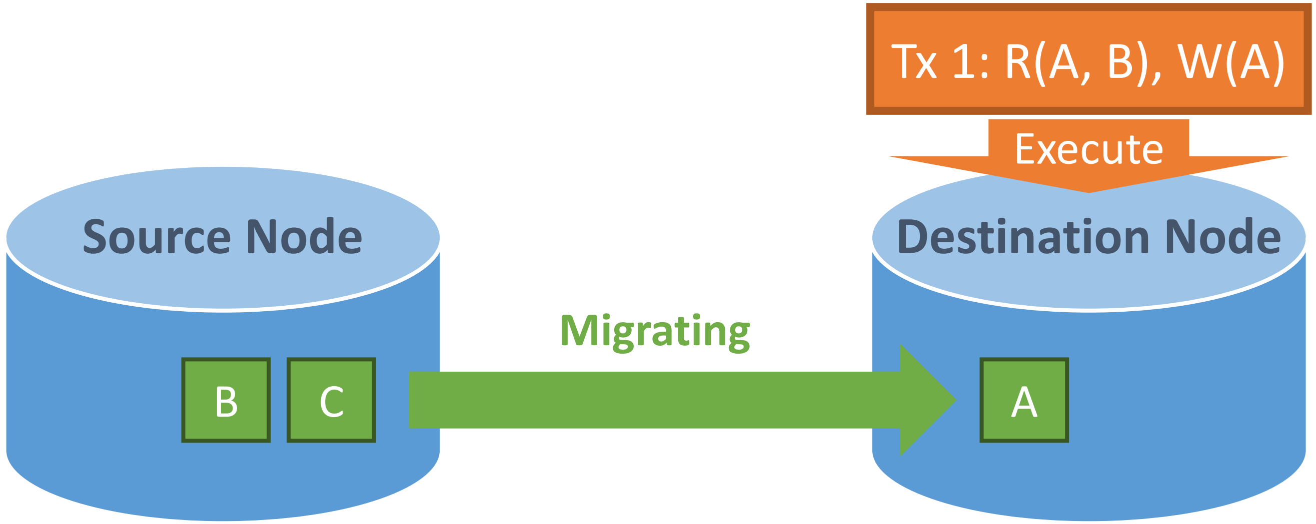
Con: **Slow** in the Beginning

- Due to the absence of data records on the destination node -> Needs to wait for pulling



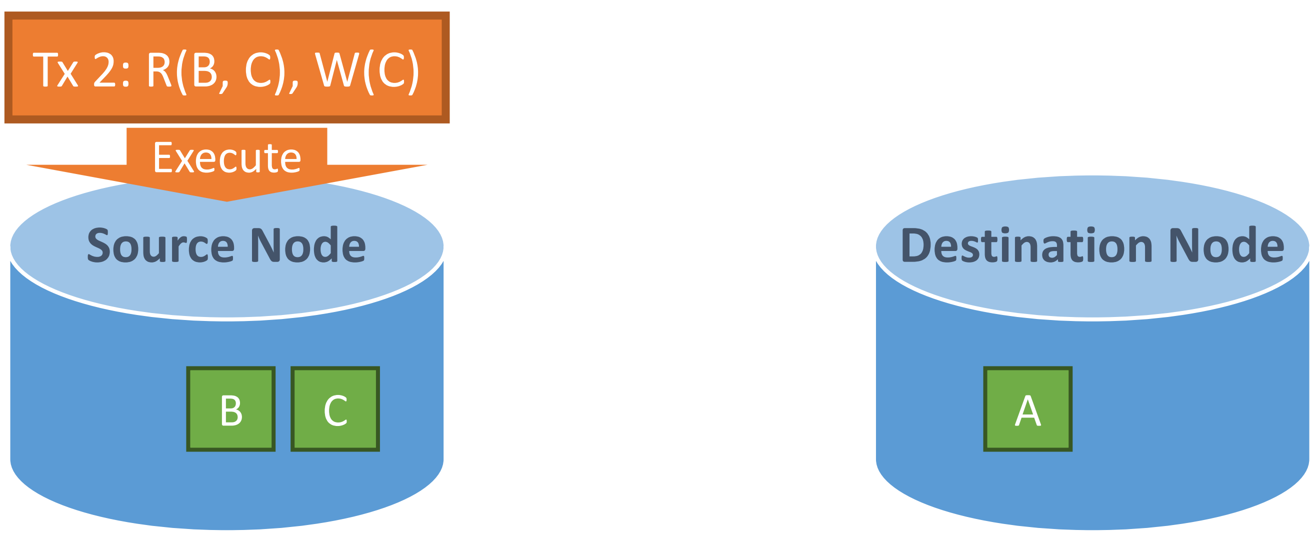
Either-Node Approaches

- Running transactions on one of nodes by carefully tracking the locations of records.
  - By default, it runs txs on destination node.
- Example: Squall [SIGMOD’15]



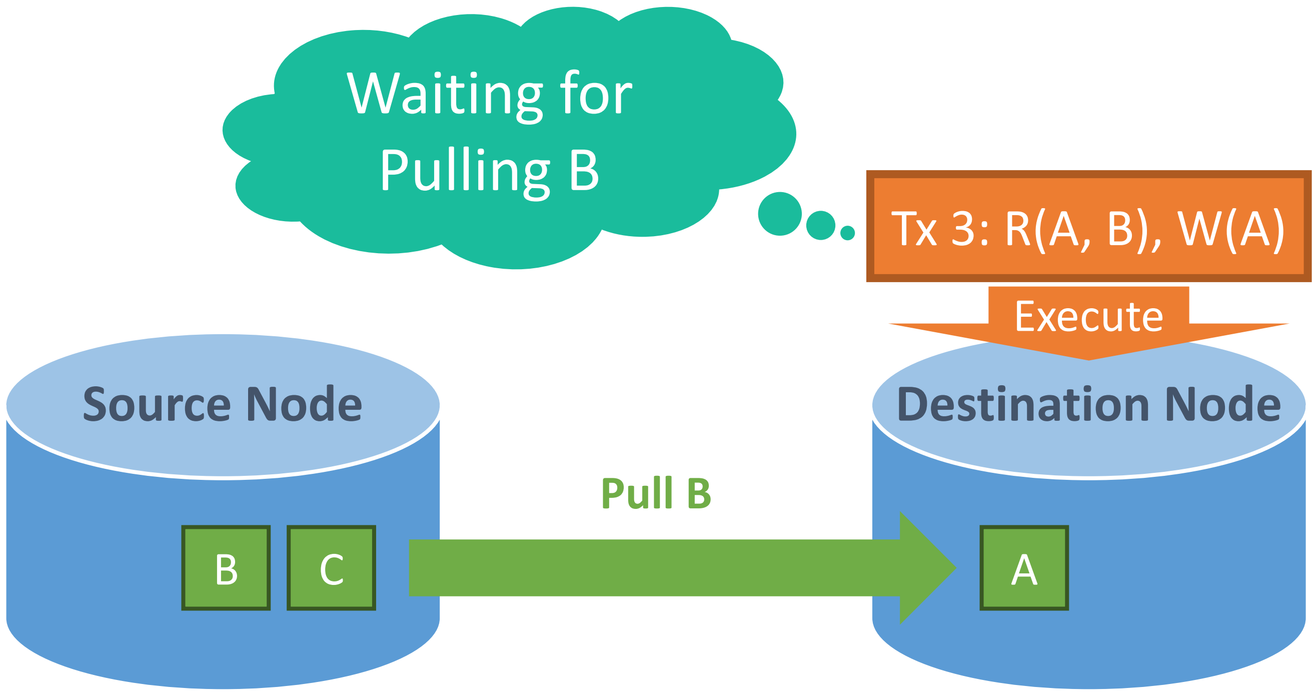
If a Tx Can Get Its All Data on the Source => Run the Tx on the Source

- Pros
  - Slightly avoids high latency in the beginning
  - No need for an atomic handover



Not Enough to Solve the Slowdown in the Beginning

- It is still possible that a transaction needs data spanning on the source and the destination.

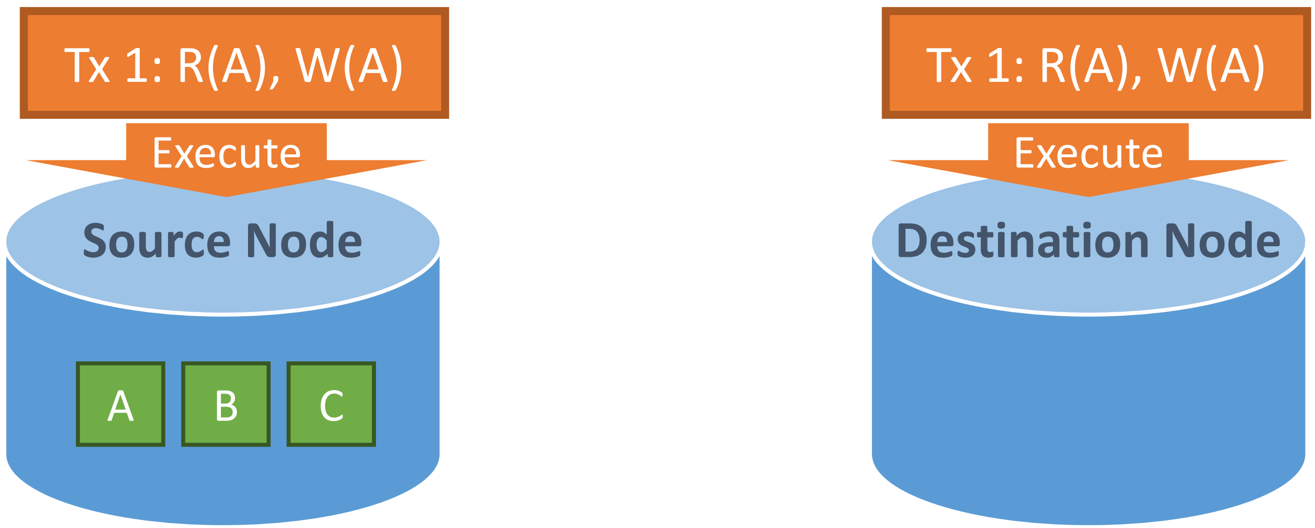


Outline

- Background
- Related Work
- MgCrab
  - Main Idea: both-nodes approaches
  - Foreground Pushes (Crabbing)
  - Two-Phase Background Pushes
- Experiments Results
- Conclusion

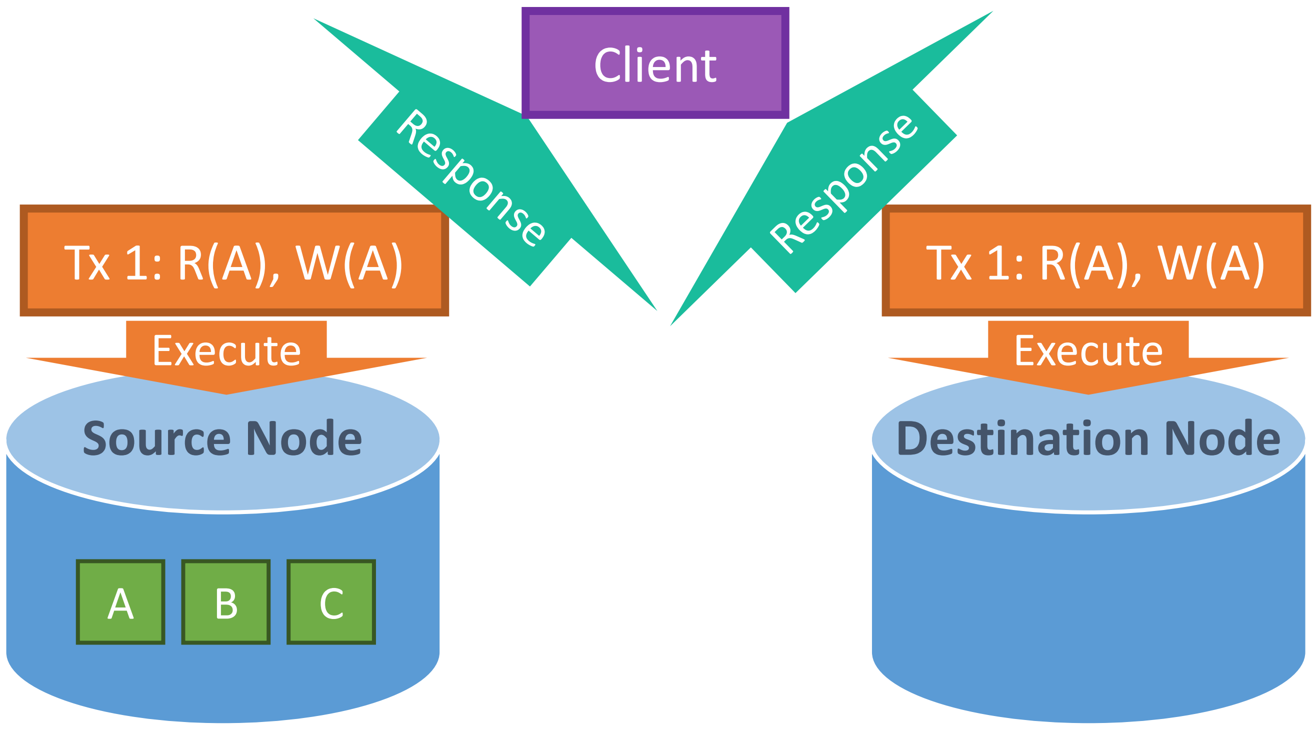
MgCrab: A **Both-Node** Approach

- Main Idea: how about running transactions on both nodes?
  - Like creating a on-demand replica.



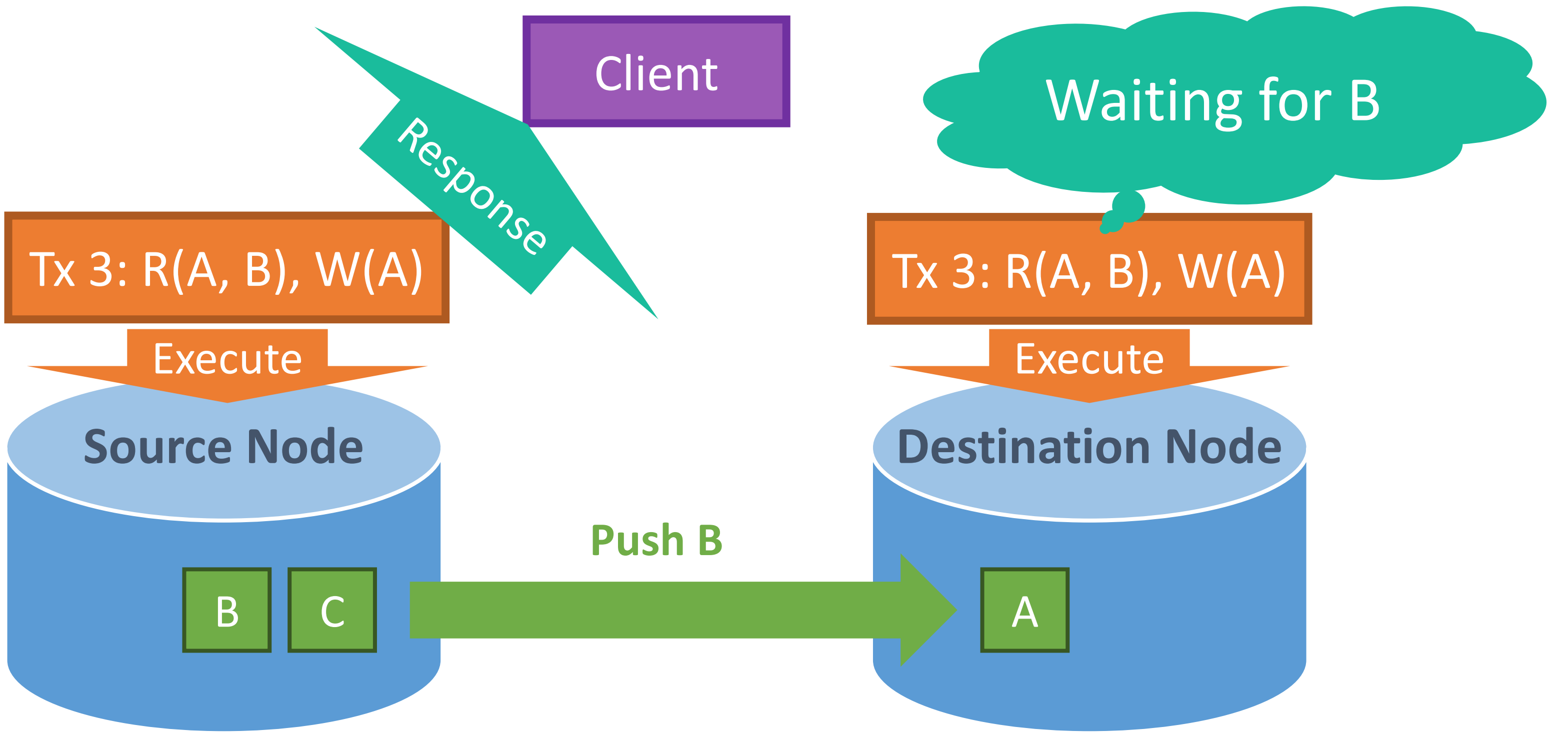
MgCrab Lets Both Nodes Response to Clients

- Benefit: the faster machine hides the latency of the slow machine.



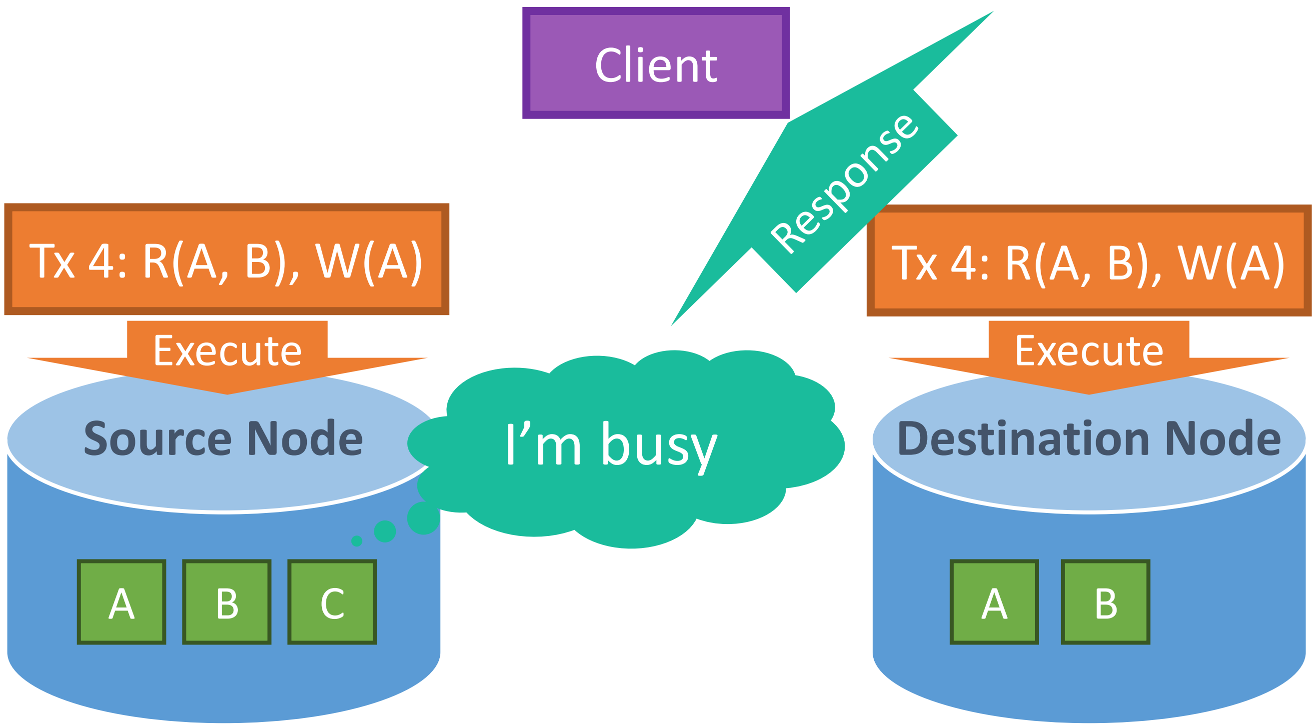
In the Beginning, MgCrab Hides the Latency of Pulling

- Totally avoids the problem of destination-based approaches.



Later, Running Txs on the Destination is Faster (scaling-out)

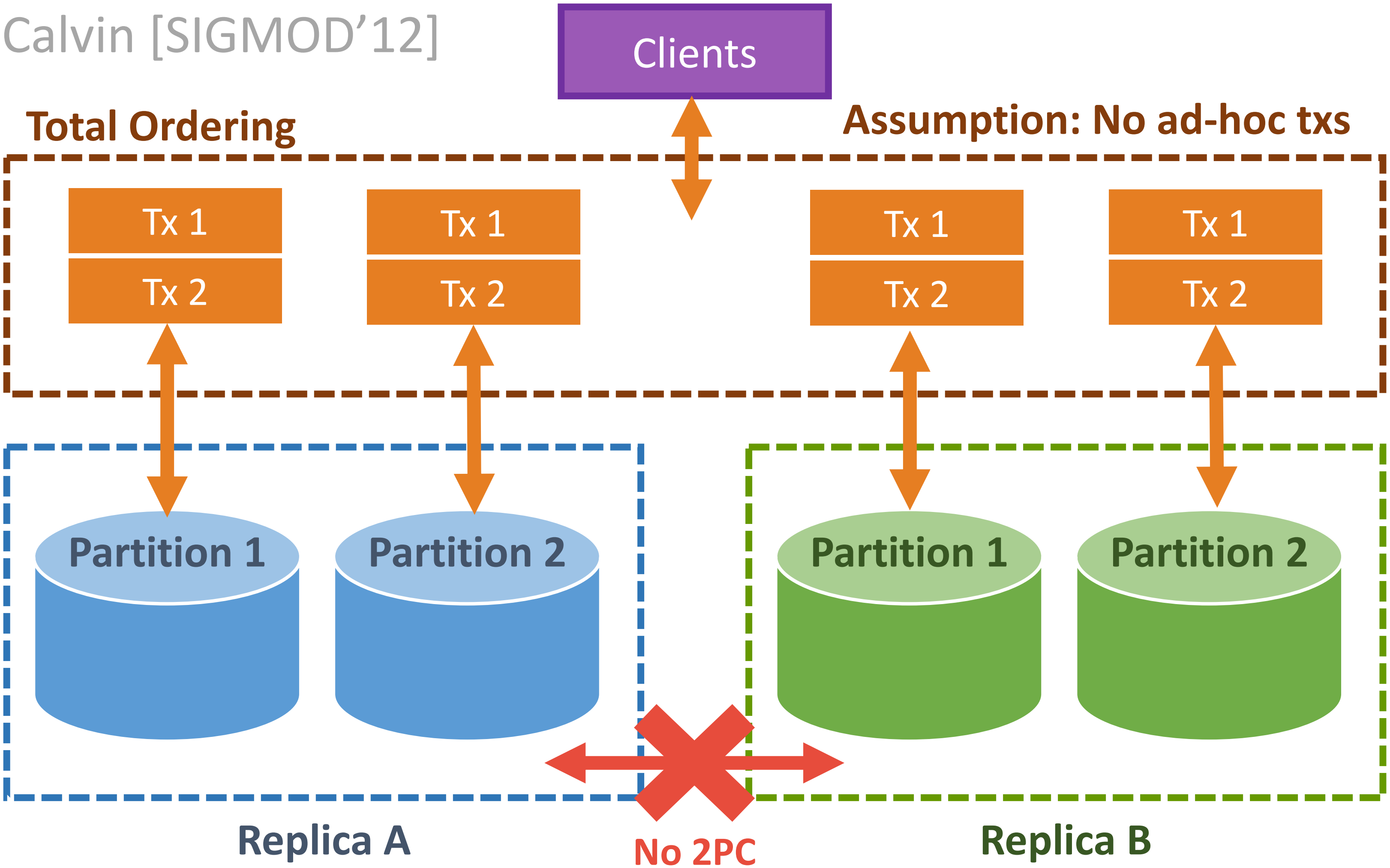
- Because the destination usually has lower loading.



Wait, it sounds costly!
How to ensure consistency?

Consistency across replicas can be easily achieved with determinism!

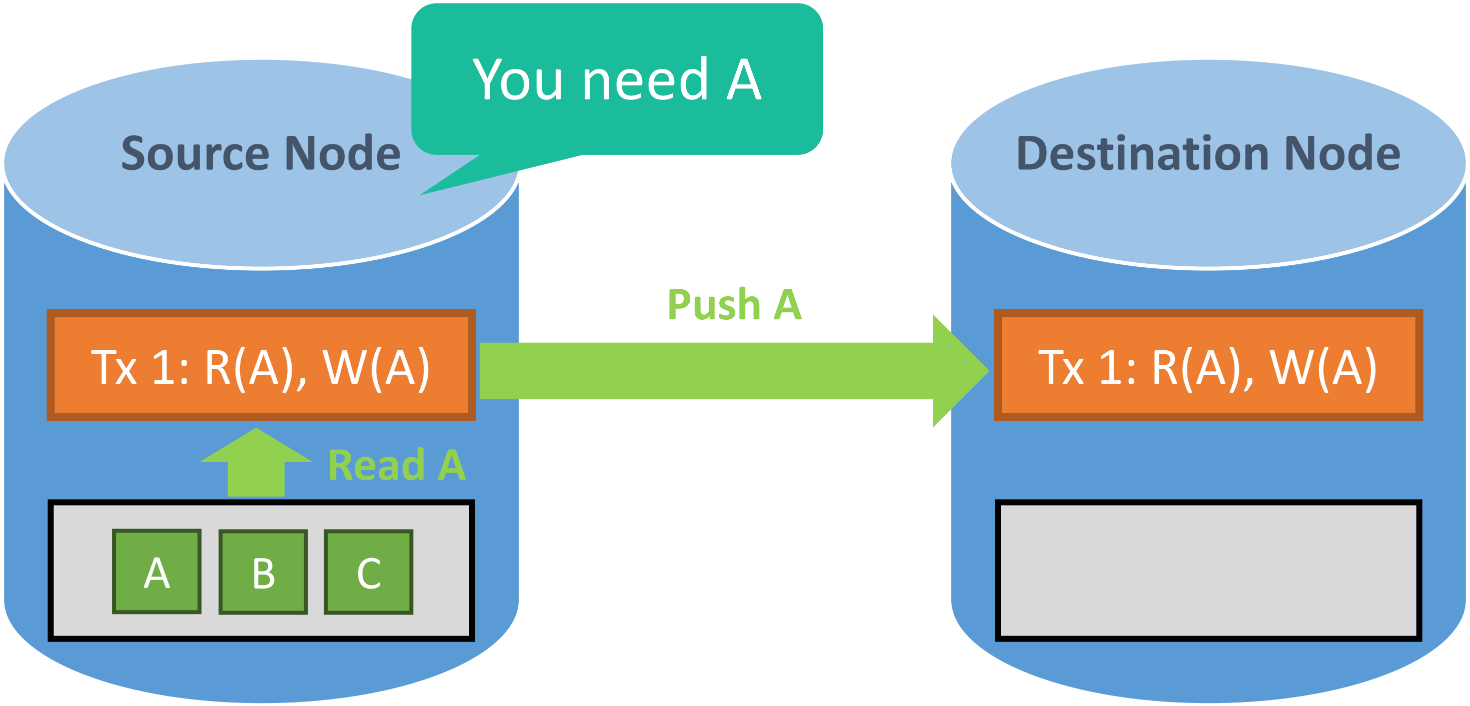
Calvin [SIGMOD'12]



How Exactly Do We Migrate Data?

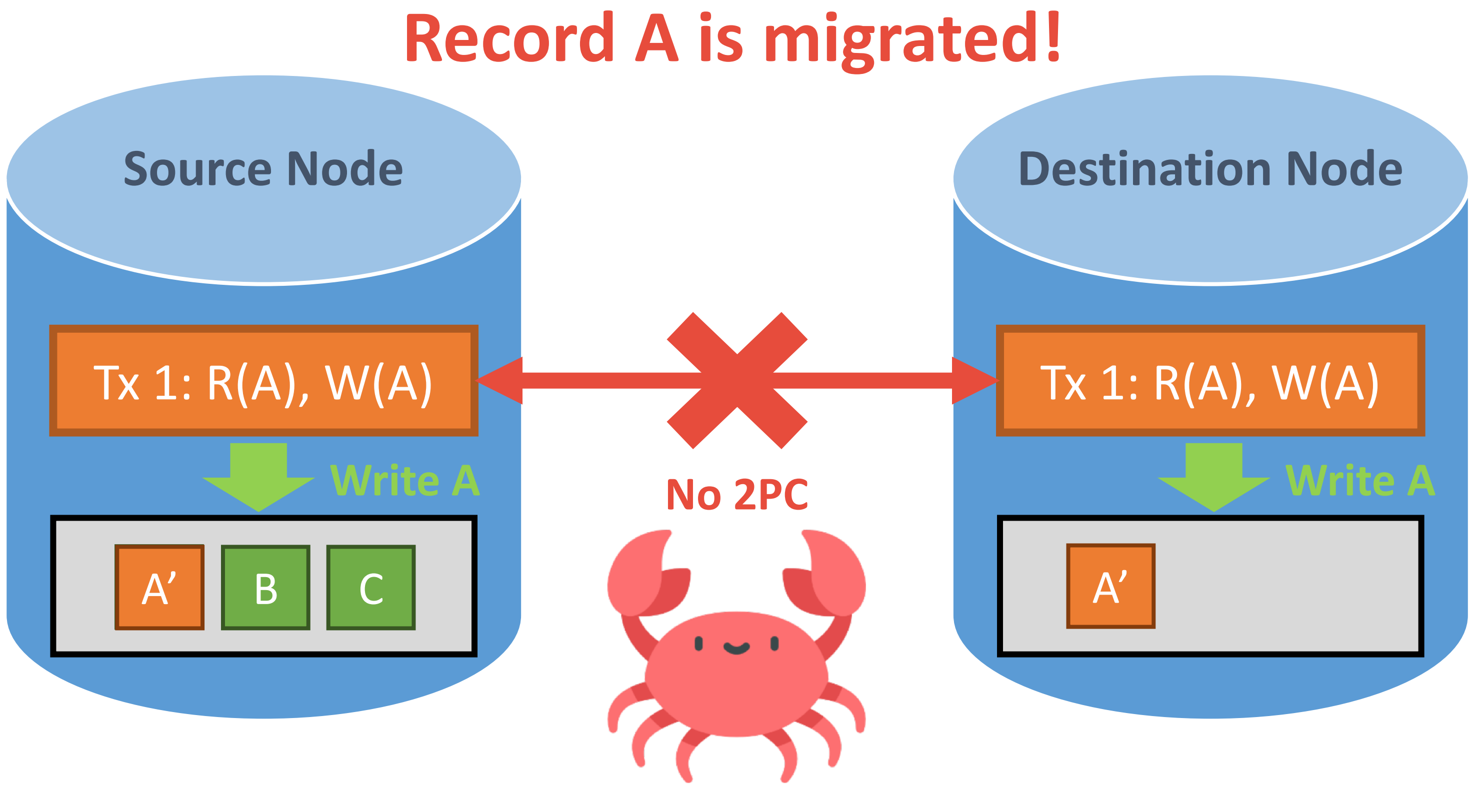
Foreground Pushes: Crabbing

- Migrates frequently used records via normal transactions.



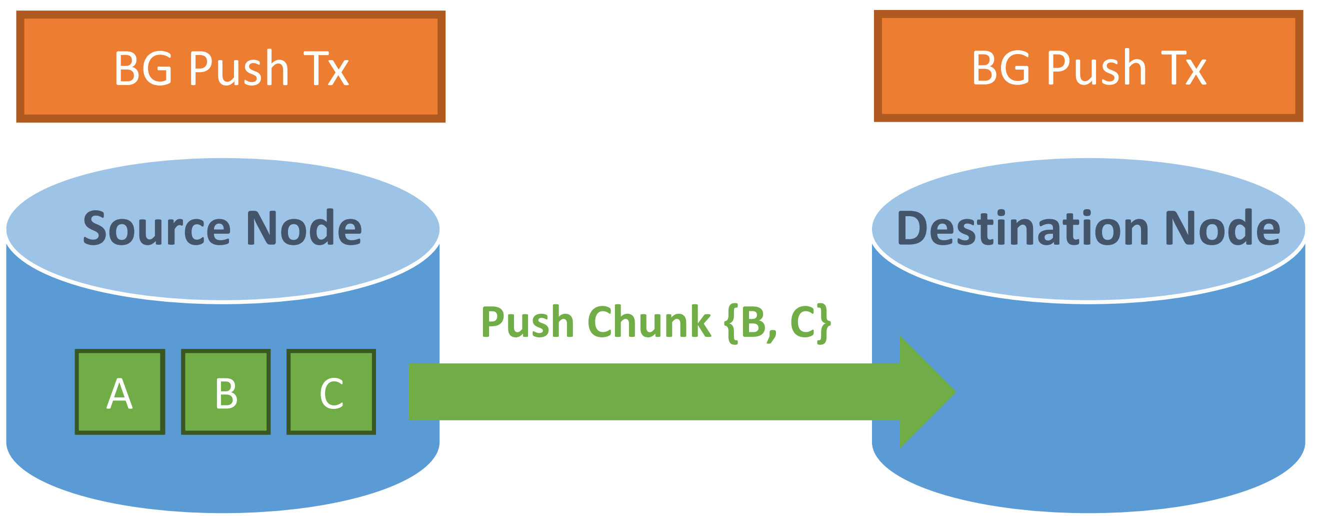
Foreground Pushes: Crabbing

- Migrates frequently used records via normal transactions.



Background Pushes

- For the rest of data (cold data), MgCrab migrates them in chunks using dedicated transactions.



Pitfall: Background pushes are more costly than we thought

How Previous Work Do Background Pushes

- Workflow in a background push:
  1. **Locks** all records in the chunk.
  2. Copies and sends the records to the destination.
  3. Applies the records to the storage.
  4. **Unlocks** the records.

Blocks bunches of transactions

We Propose: Two-Phase Background Pushes

- Observation: only needs to lock records when the records are applied to the destination.

Phase1 on Source Node

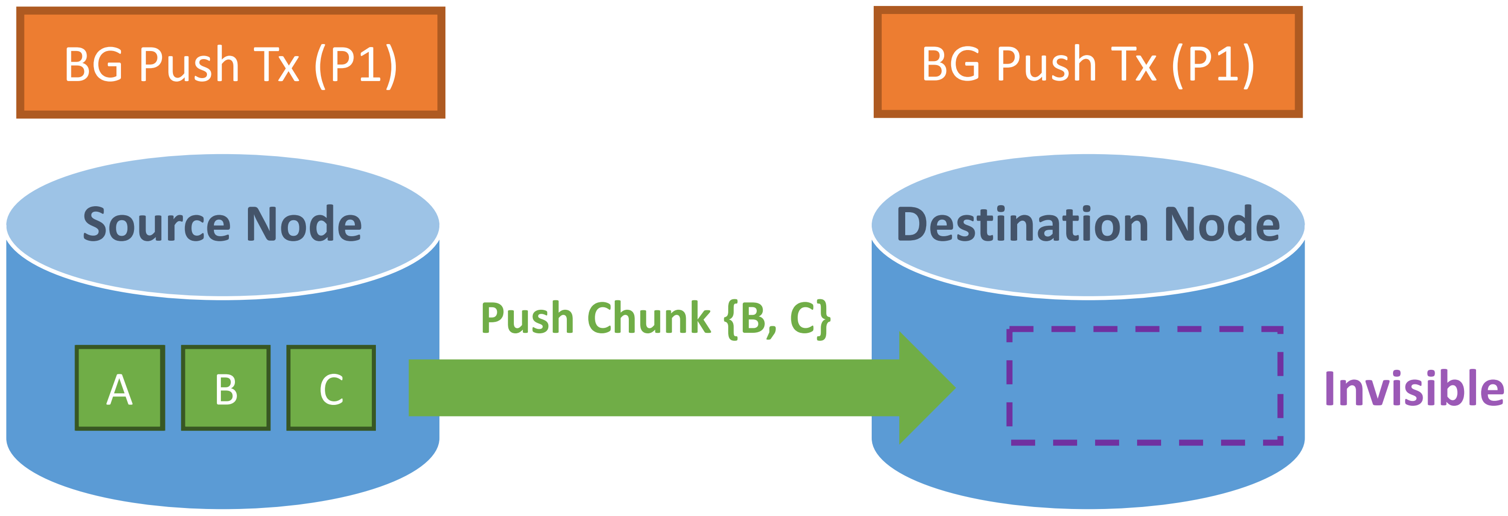
1. Copies and sends the records to the destination.

Phase 2 on Destination Node

2. **Locks** all records in the chunk.
3. Applies the **untouched** records to the storage.
4. **Unlocks** the records.

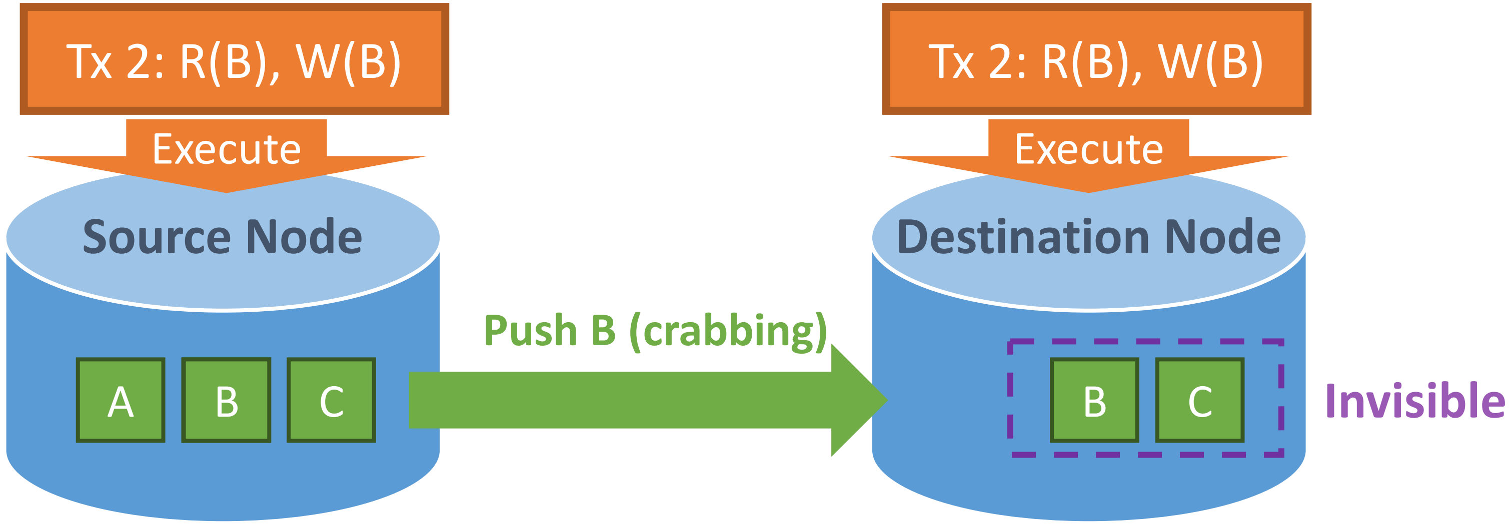
Two-Phase Background Pushes

- BG Push Phase 1: copies and sends data to the destination **without** acquiring locks.
  - The data may be inconsistent.



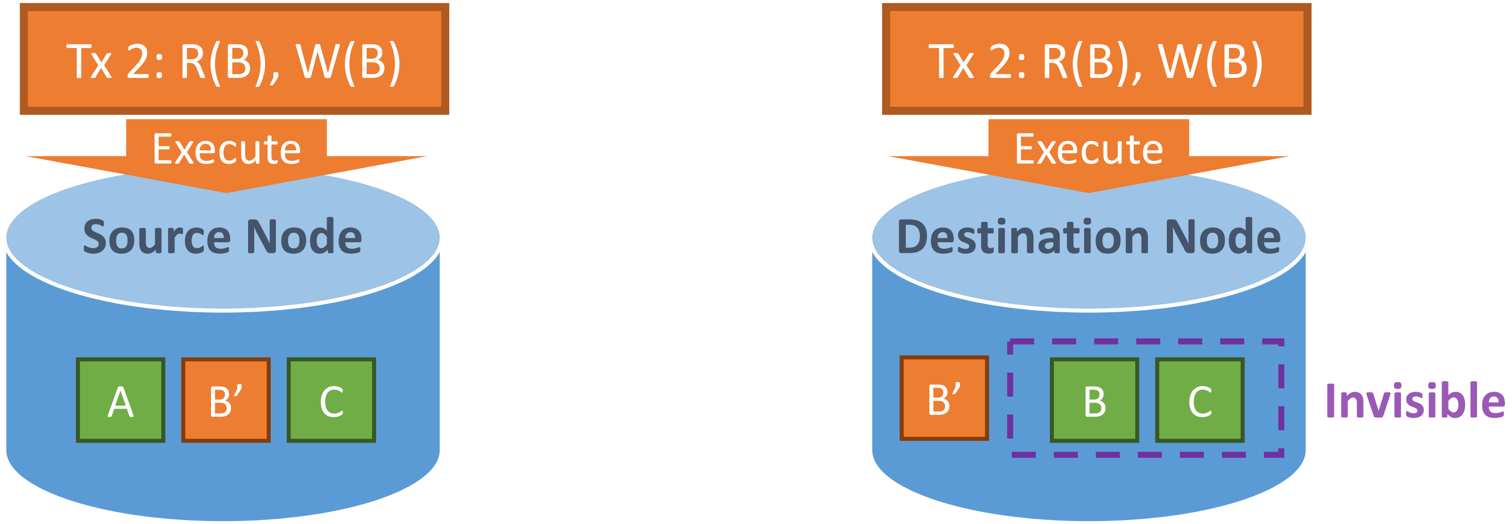
Two-Phase Background Pushes

- Other transactions can still modify the records in the chunk.



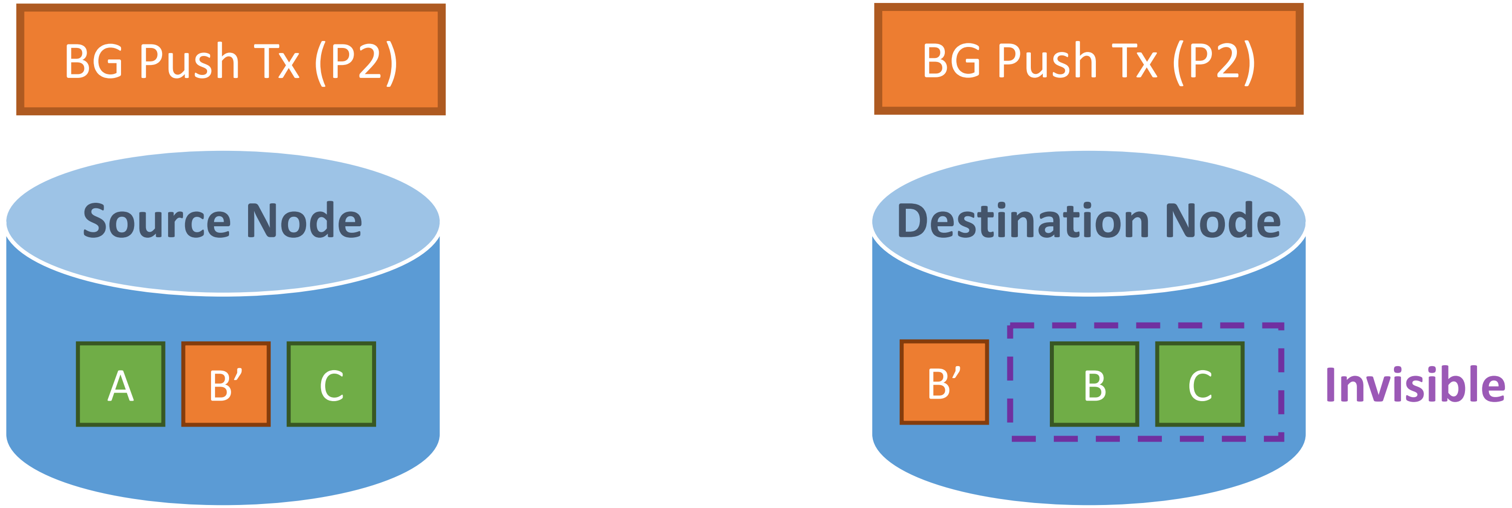
Two-Phase Background Pushes

- Other transactions can still modify the records in the chunk.



Two-Phase Background Pushes

- BG Push Phase 2: abandons outdated records, and then lock and apply the rest records to the storage
  - The outdated records have been migrated in foreground pushes anyway.



More Discussions & Optimizations

- Generalization
  - Range queries?
  - Distributed transactions?
  - Concurrent migration plans?
- Optimizations
  - Catching-up phase & Caught-up phase

- See the paper!

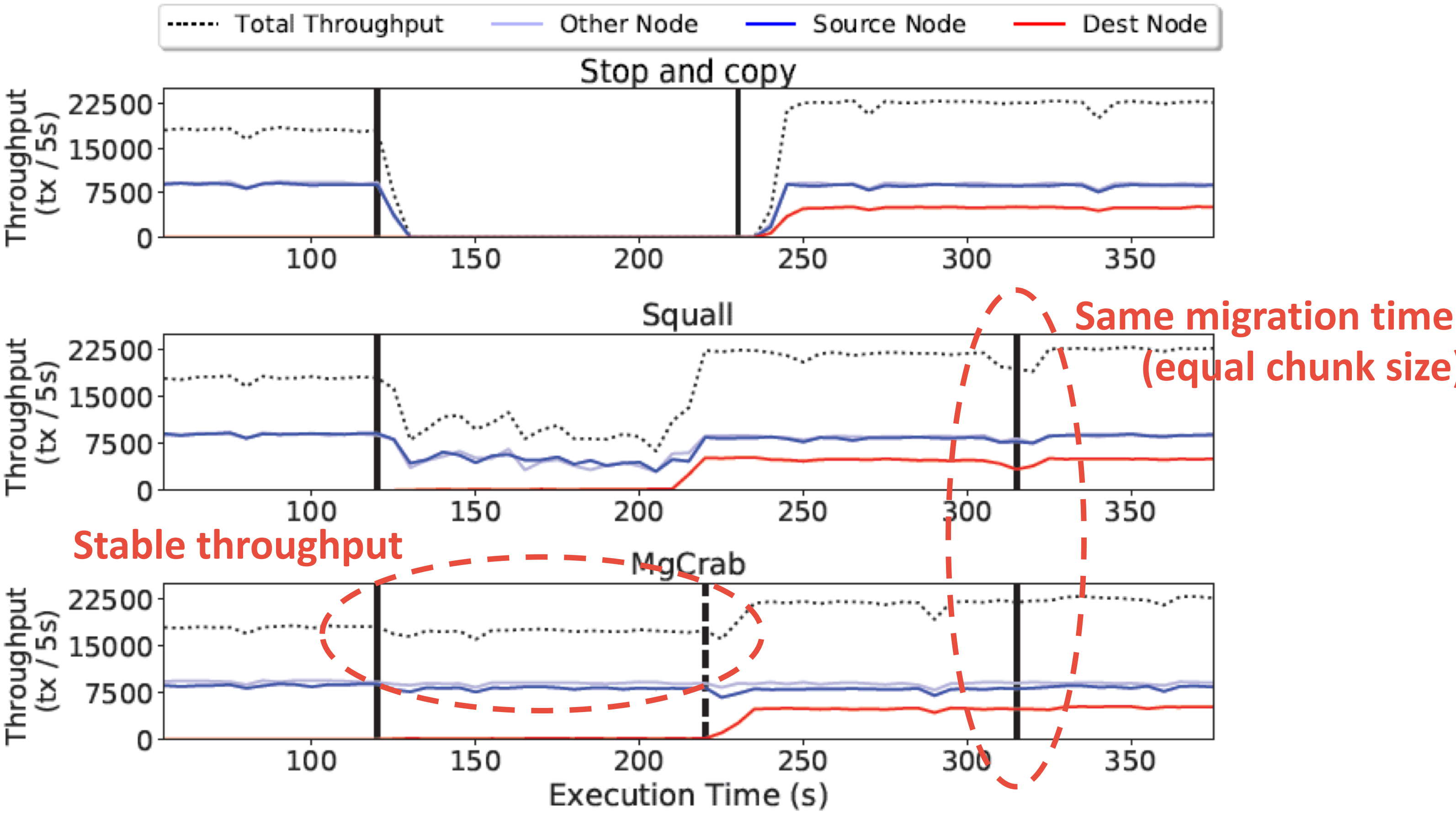
Outline

- Background
- Related Work
- MgCrab
- Experiments Results
- Conclusion

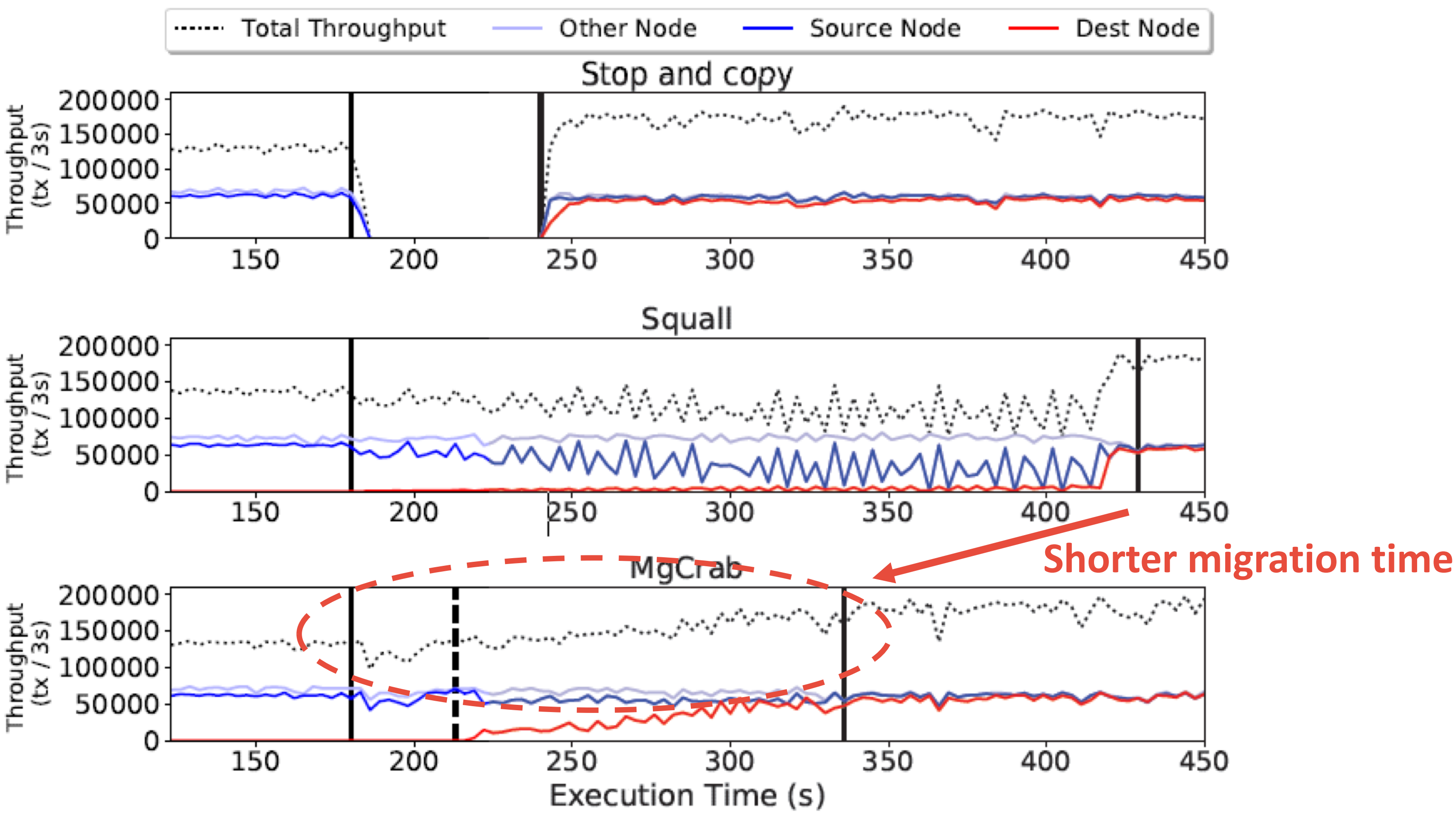
Settings

- Baselines
  - Stop-and-copy
  - Squall (either-node approach)
- Benchmarks
  - YCSB
  - TPC-C

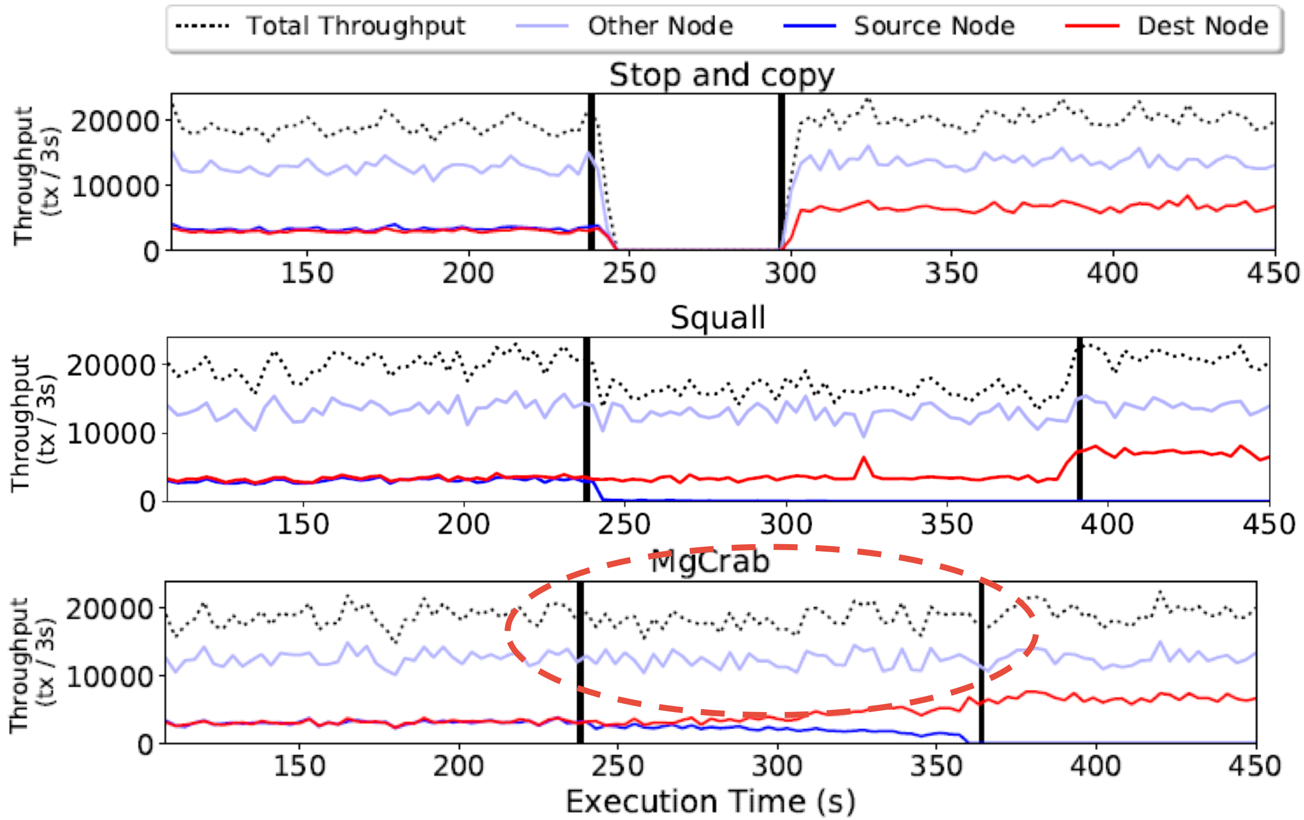
Scaling-out in TPC-C



Scaling-out in YCSB



Consolidation in YCSB



Conclusion for Takeaway

- Either-node approaches do not solve all the problems.
- Both-node execution simplifies design and reduces impact.
  - Because determinism achieves lightweight replication.
- Background pushes have pitfall, and thus we divide a push into two phases.

Thanks for listening!
Q&A

Let's meet at poster 26.2 later!

Slides →