# Preserving Reciprocal Consistency in Distributed Graph Databases

*Jack Waudby*[1], Paul Ezhilchelvan[1], Jim Webber[2] & Isi Mitrani[1]

April 27, 2020

[1]Newcastle University

[2]Neo4j

# Graph Databases

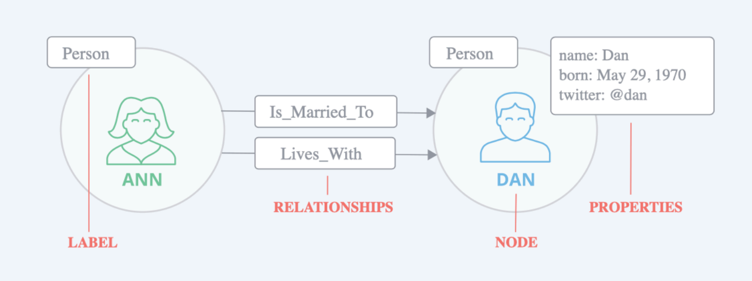


**Figure 1:** Labeled property graph[1]

[1] `https://neo4j.com`

# Graph Databases

- **Efficient graph analysis:** reachability, pattern matching, shortest path search and clustering

# Graph Databases

- **Efficient graph analysis:** reachability, pattern matching, shortest path search and clustering
- **Use cases:** telecommunications, pharma, publishing, finance, social media

# Graph Database Popularity

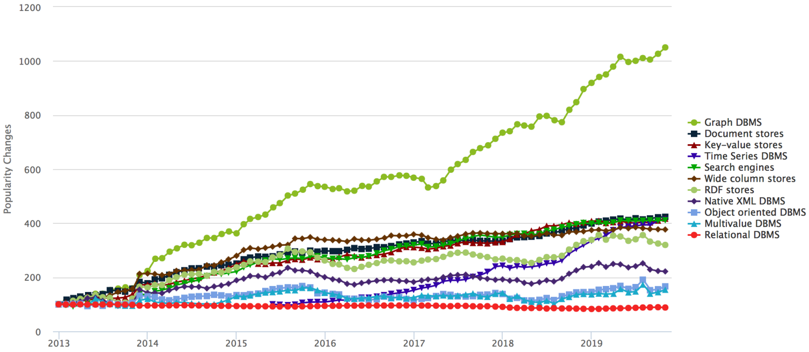


**Figure 2:** Database popularity by data model[2]

[1] `https://db-engines.com/en/ranking_categories`

# Distributed Graph Databases

- Graph exceeds the storage capacity of a single machine

# Distributed Graph Databases

- Graph exceeds the storage capacity of a single machine
- Partition the graph across multiple machines in a cluster, with partitions replicated for fault-tolerance and availability

# Distributed Graph Databases

- Graph exceeds the storage capacity of a single machine
- Partition the graph across multiple machines in a cluster, with partitions replicated for fault-tolerance and availability
- Graph partitioning inevitably leads to **distributed edges**

# Distributed Graph Databases: Partitioning

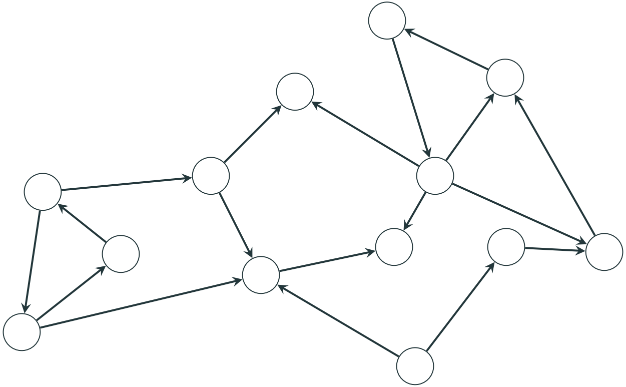


Figure 3: Connected graph

# Distributed Graph Databases: Partitioning

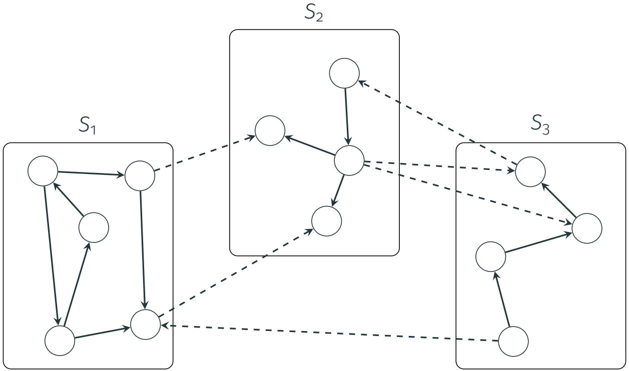


**Figure 4:** Graph partitioned across 3 servers

# Reciprocal Consistency

A logical edge is represented by 2 physical records

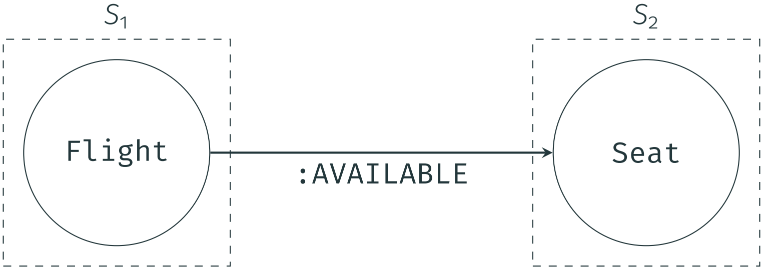


**Figure 5:** A reciprocal consistent edge

# Reciprocal Consistency

A logical edge is represented by 2 physical records

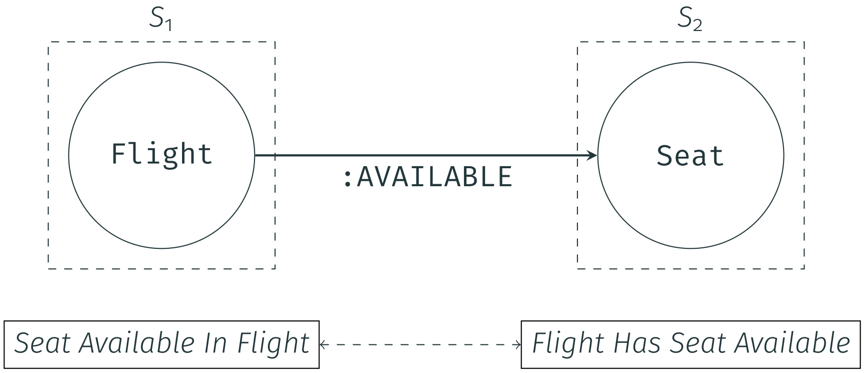


**Figure 5:** A reciprocal consistent edge

# Reciprocal Consistency Violation

Two concurrent transactions:

- Mr Red requests to book the seat, $T_R$
- Mr Blue requests to book the seat, $T_B$

# Reciprocal Consistency Violation

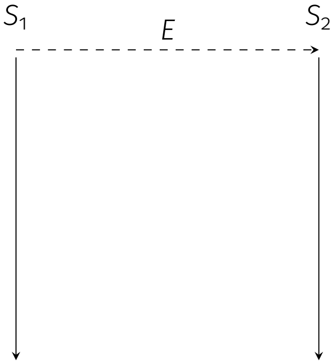


**Figure 6:** Concurrent transactions interleave and violate reciprocal consistency

# Reciprocal Consistency Violation

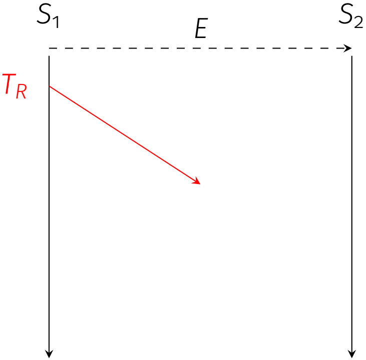


**Figure 6:** Concurrent transactions interleave and violate reciprocal consistency

# Reciprocal Consistency Violation

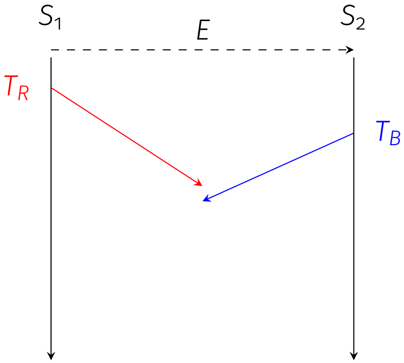


**Figure 6:** Concurrent transactions interleave and violate reciprocal consistency

# Reciprocal Consistency Violation

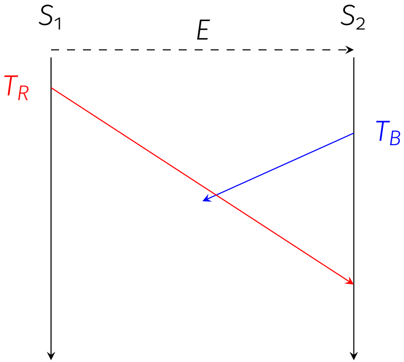


**Figure 6:** Concurrent transactions interleave and violate reciprocal consistency

# Reciprocal Consistency Violation

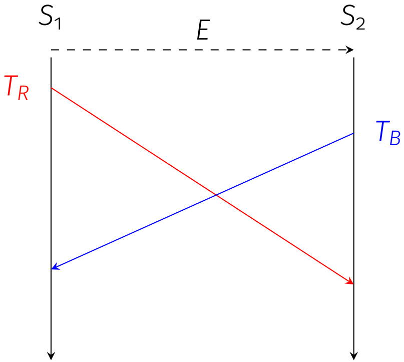


**Figure 6:** Concurrent transactions interleave and violate reciprocal consistency

# Reciprocal Consistency Violation

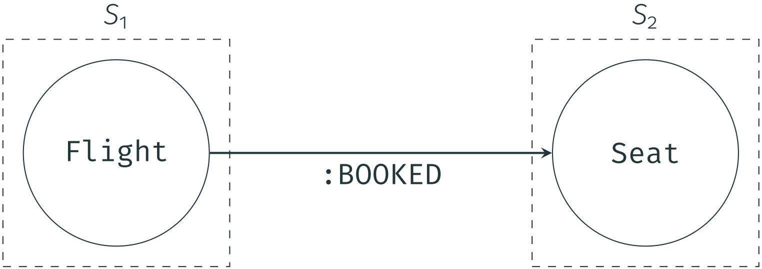


**Figure 7:** Reciprocal inconsistent distributed edge

# Reciprocal Consistency Violation

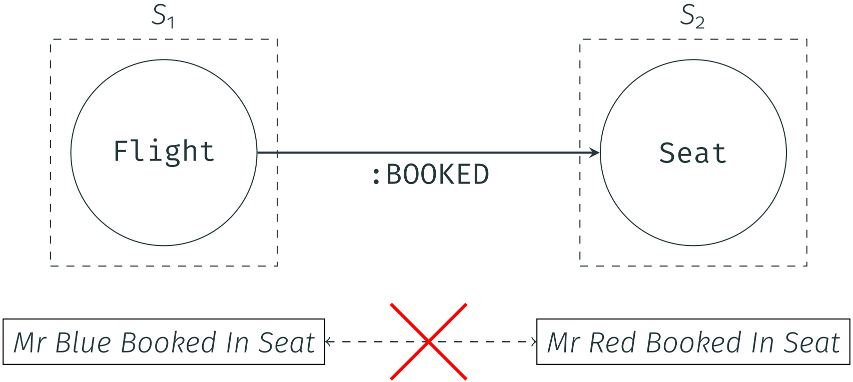


**Figure 7:** Reciprocal inconsistent distributed edge

# Distributed Edge Reciprocal Consistency

- Without concurrency control, distributed edges can become reciprocally inconsistent
  - A distributed edge's reciprocal information is separated by the network

[3] Ezhilchelvan, P. et al, *On the degradation of distributed graph databases with eventual consistency*. European Performance Engineering Workshop 2018.

# Distributed Edge Reciprocal Consistency

- Without concurrency control, distributed edges can become reciprocally inconsistent
  - A distributed edge's reciprocal information is separated by the network
- Reciprocal inconsistency is a source of corruption[3]

[3] Ezhilchelvan, P. et al, *On the degradation of distributed graph databases with eventual consistency*. European Performance Engineering Workshop 2018.

# Distributed Edge Reciprocal Consistency

- Without concurrency control, distributed edges can become reciprocally inconsistent
  - A distributed edge's reciprocal information is separated by the network
- Reciprocal inconsistency is a source of corruption[3]
- No known concurrency control protocol exists specific to graph databases and maintaining reciprocal consistency

[3] Ezhilchelvan, P. et al, *On the degradation of distributed graph databases with eventual consistency*. European Performance Engineering Workshop 2018.

# Our Solution: Collision Detection

- Carefully abort transaction(s)

# Our Solution: Collision Detection

- Carefully abort transaction(s)
- No centralized control or synchronized clock

# Our Solution: Collision Detection

- Carefully abort transaction(s)
- No centralized control or synchronized clock
- Permits interleavings that preserve reciprocal consistency

# Collision Detection General Rules

- Updates are initially provisional

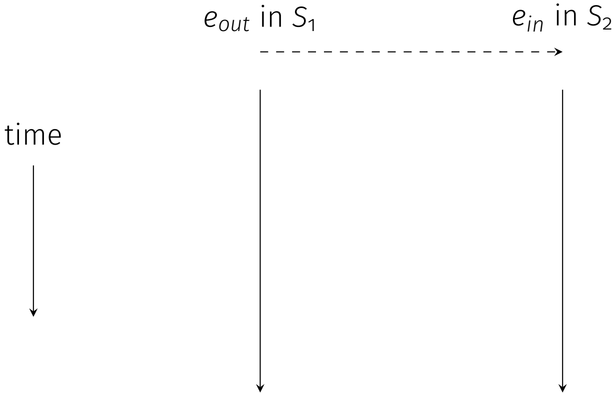
# Collision Detection General Rules

- Updates are initially provisional
- Transactions distinguish between their first and second updates using labels

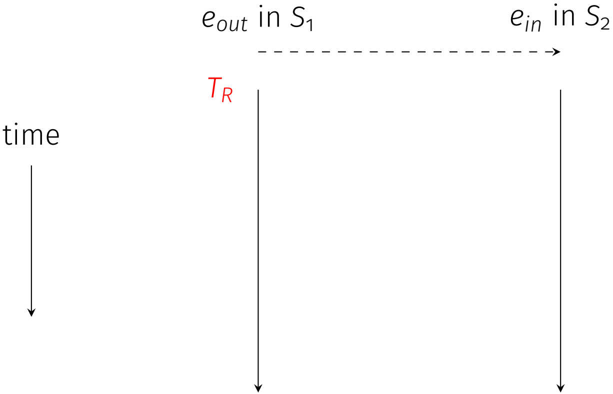
# Collision Detection General Rules

- Updates are initially provisional
- Transactions distinguish between their first and second updates using labels
- When a transaction attempts to update a given record, it can identify all other transactions that have earlier updated that record provisionally

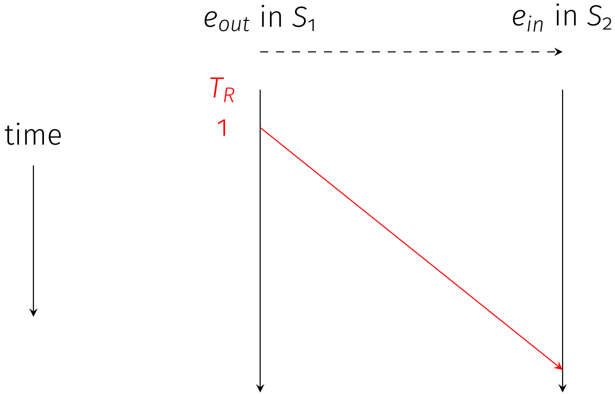
# Interference-free Update



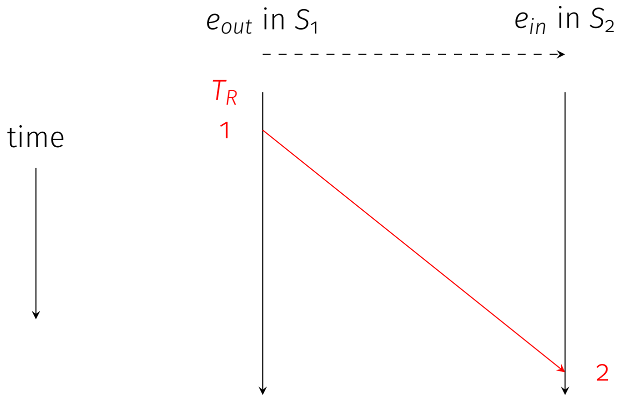
# Interference-free Update



# Interference-free Update



# Interference-free Update

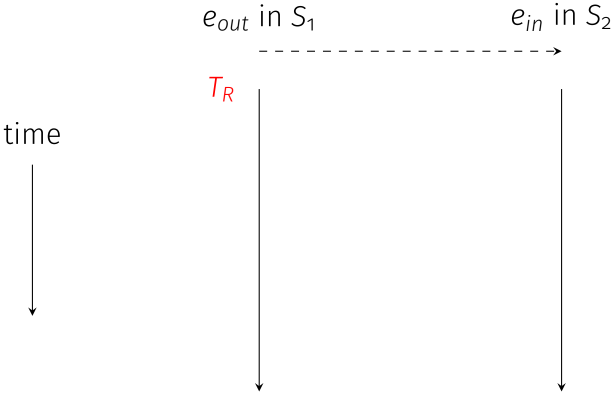


# Collision Detection

Rule: For any “1” seen, there must be a “2” in the opposite end. Else, abort

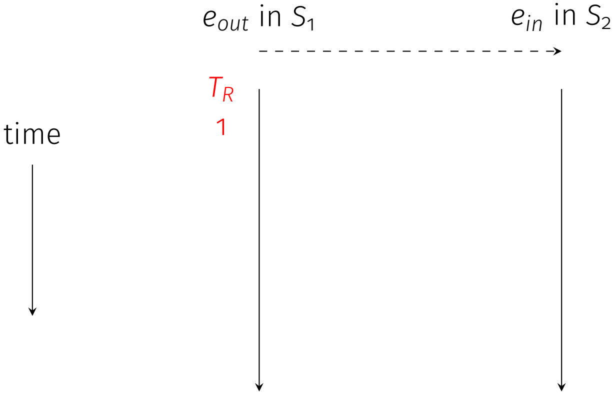
# Collision Detection

Rule: For any “1” seen, there must be a “2” in the opposite end. Else, abort



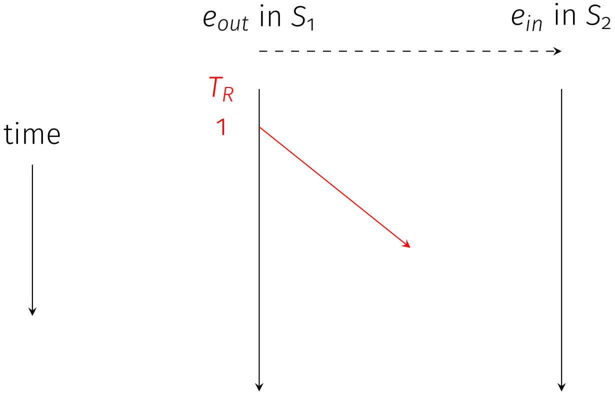
# Collision Detection

Rule: For any “1” seen, there must be a “2” in the opposite end. Else, abort



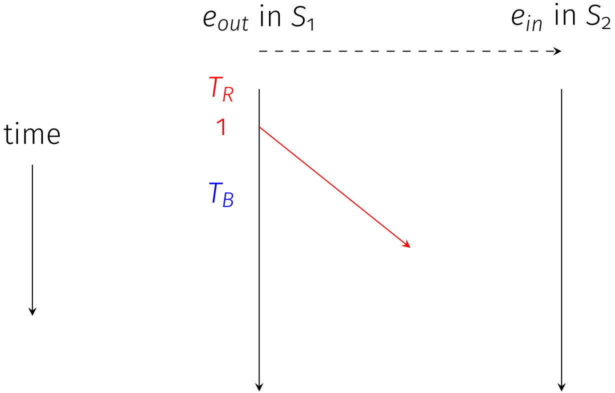
# Collision Detection

Rule: For any “1” seen, there must be a “2” in the opposite end. Else, abort



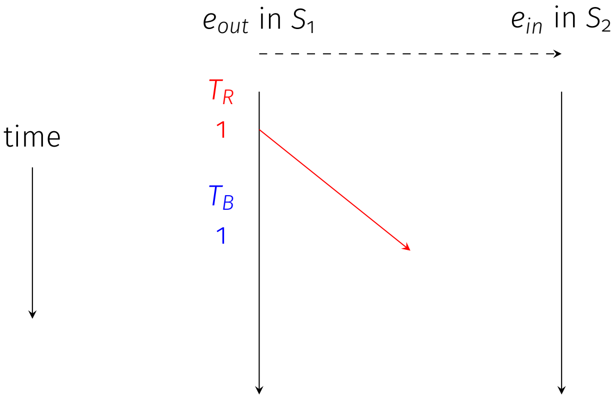
# Collision Detection

Rule: For any “1” seen, there must be a “2” in the opposite end. Else, abort



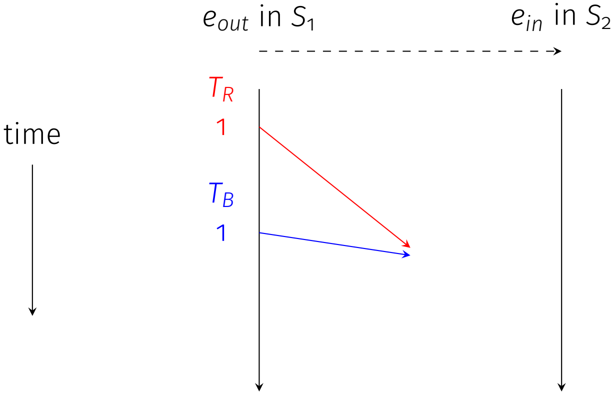
# Collision Detection

Rule: For any “1” seen, there must be a “2” in the opposite end. Else, abort



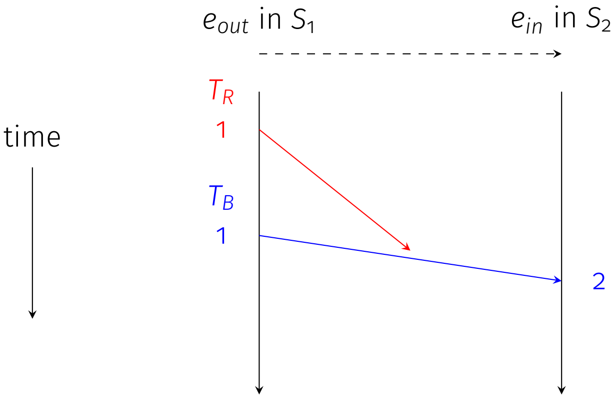
# Collision Detection

Rule: For any “1” seen, there must be a “2” in the opposite end. Else, abort



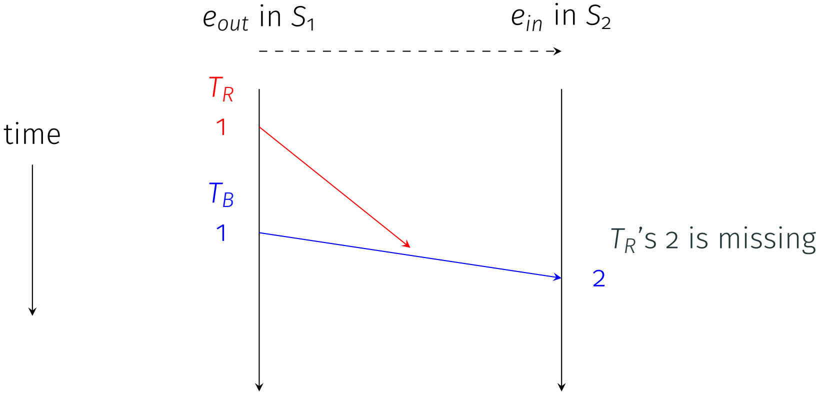
# Collision Detection

Rule: For any “1” seen, there must be a “2” in the opposite end. Else, abort



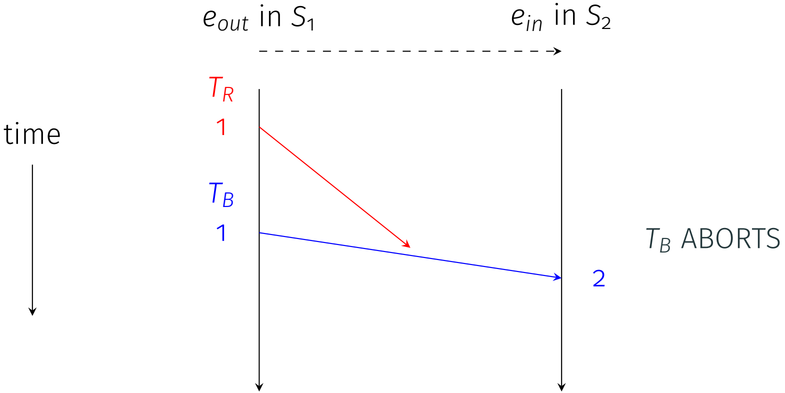
# Collision Detection

Rule: For any “1” seen, there must be a “2” in the opposite end. Else, abort



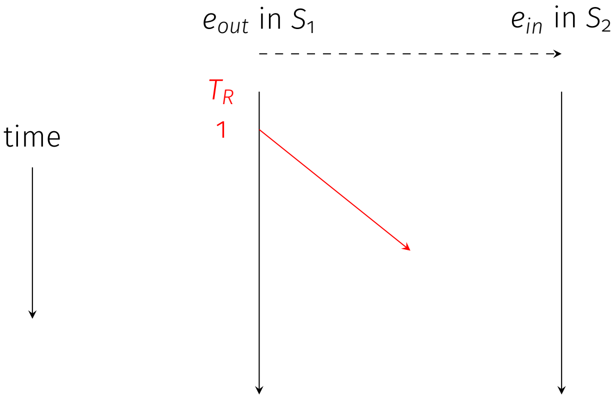
# Collision Detection

Rule: For any “1” seen, there must be a “2” in the opposite end. Else, abort



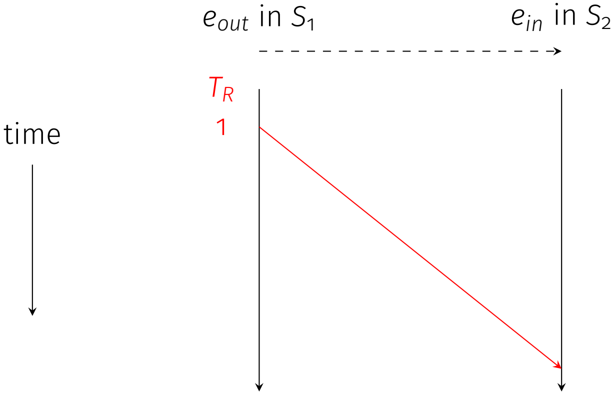
# Collision Detection

Rule: For any “1” seen, there must be a “2” in the opposite end. Else, abort



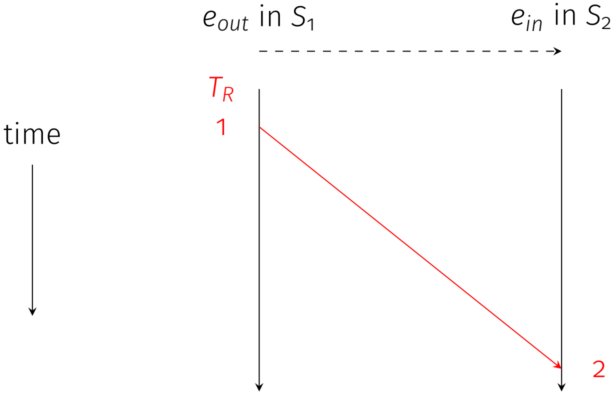
# Collision Detection

Rule: For any “1” seen, there must be a “2” in the opposite end. Else, abort



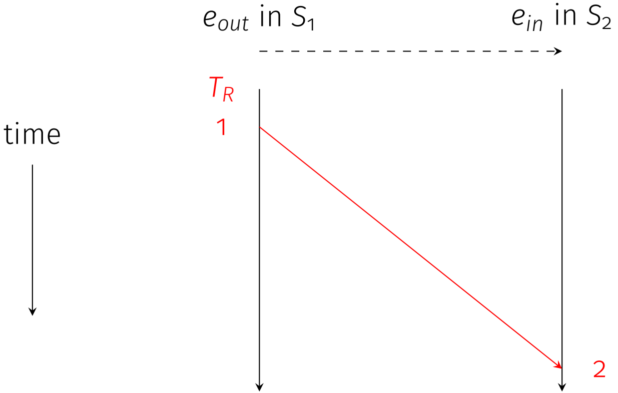
# Collision Detection

Rule: For any “1” seen, there must be a “2” in the opposite end. Else, abort



# Collision Detection

Rule: For any “1” seen, there must be a “2” in the opposite end. Else, abort



$T_R$ SUCCEEDS

# Edge-Order Consistency

A second consistency problem related to distributed edges:

# Edge-Order Consistency

A second consistency problem related to distributed edges:

- Consider 2 transactions that update 2 or more of the *same* distributed edges

# Edge-Order Consistency

A second consistency problem related to distributed edges:

- Consider 2 transactions that update 2 or more of the *same* distributed edges
- Each update is reciprocally consistent

# Edge-Order Consistency

A second consistency problem related to distributed edges:

- Consider 2 transactions that update 2 or more of the *same* distributed edges
- Each update is reciprocally consistent
- Then if $T_R$ updates before $T_B$ on a given edge then this order should be preserved across all edges

# Edge-Order Consistency Violation

Consider two edges,

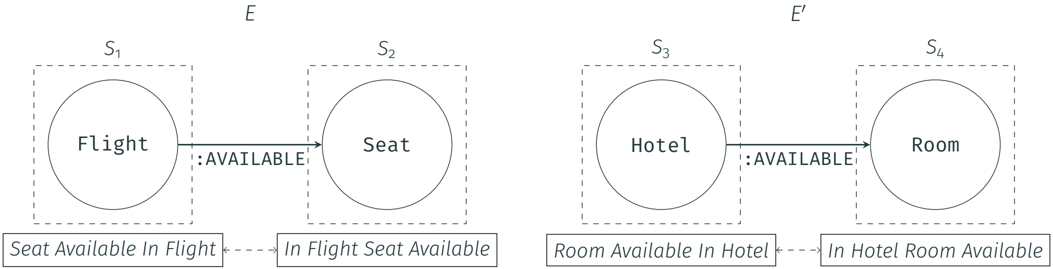


**Figure 8:** Two distributed edges

# Edge-Order Consistency Violation

Two concurrent transactions by a travel agent:

- Requests to book room and seat for Mr Red, $T_R$
- Requests to book room and seat for Mr Blue, $T_B$

# Edge-Order Consistency Violation

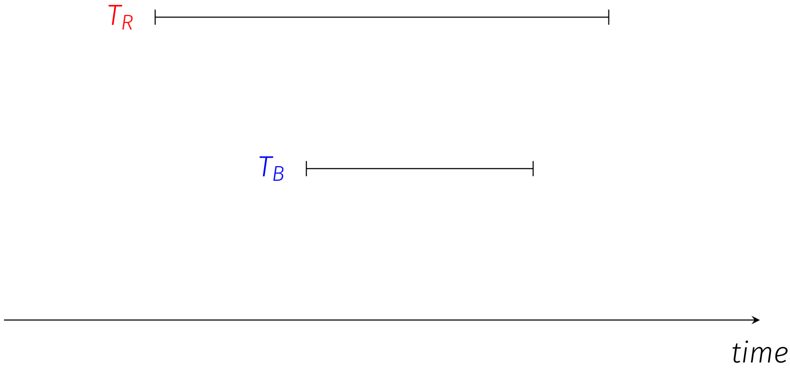


**Figure 9:** Edge-order consistency violation

# Edge-Order Consistency Violation

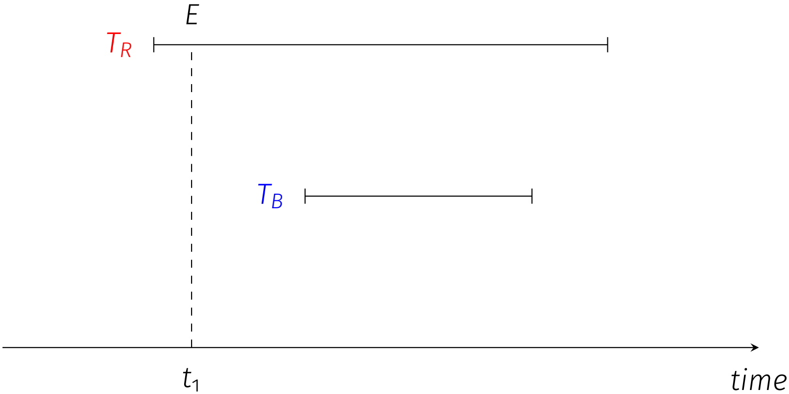


**Figure 9:** Edge-order consistency violation

# Edge-Order Consistency Violation

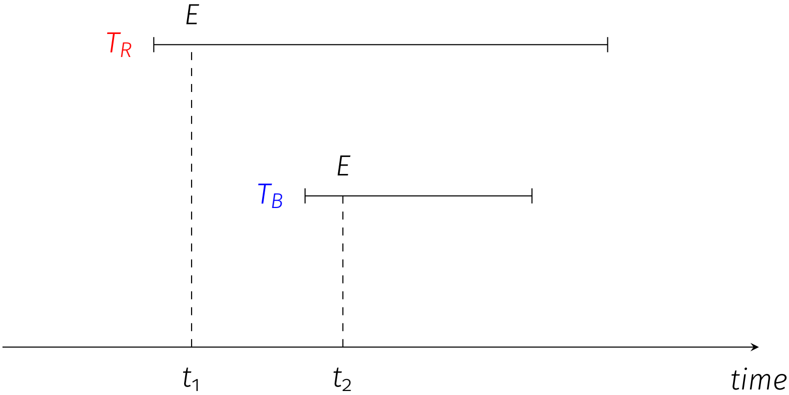


**Figure 9:** Edge-order consistency violation

# Edge-Order Consistency Violation

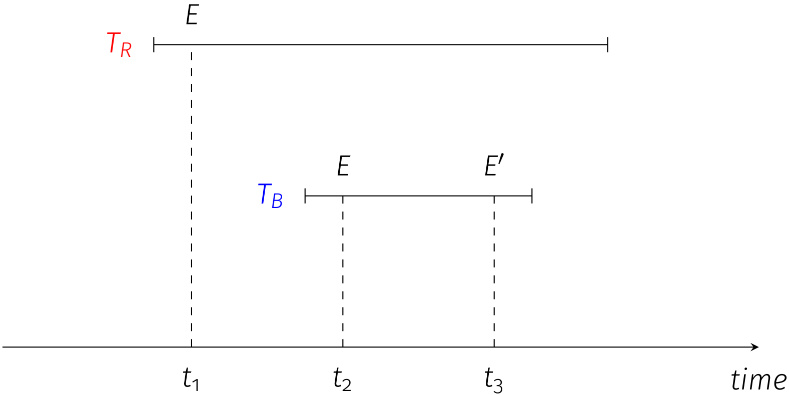


**Figure 9:** Edge-order consistency violation

# Edge-Order Consistency Violation

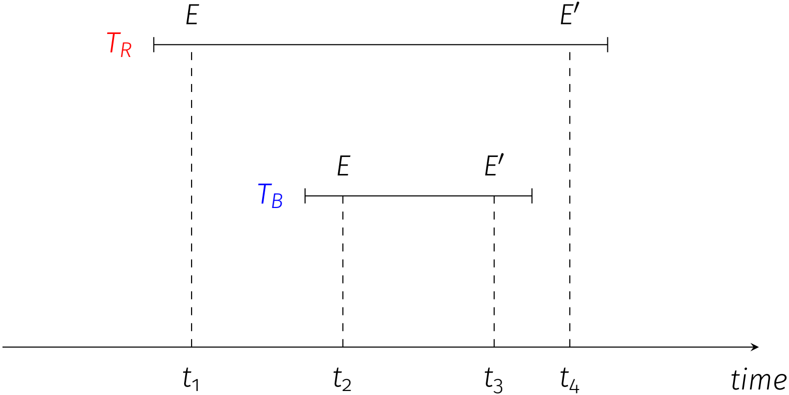


**Figure 9:** Edge-order consistency violation

# Edge-Order Consistency Violation

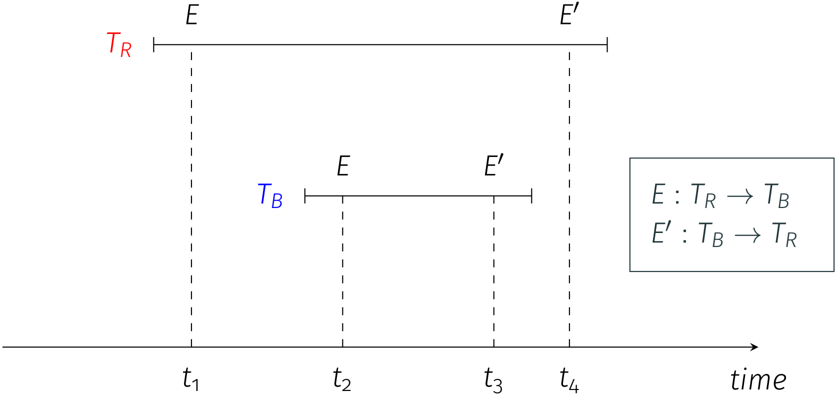


**Figure 9:** Edge-order consistency violation

# Edge-Order Consistency Violation

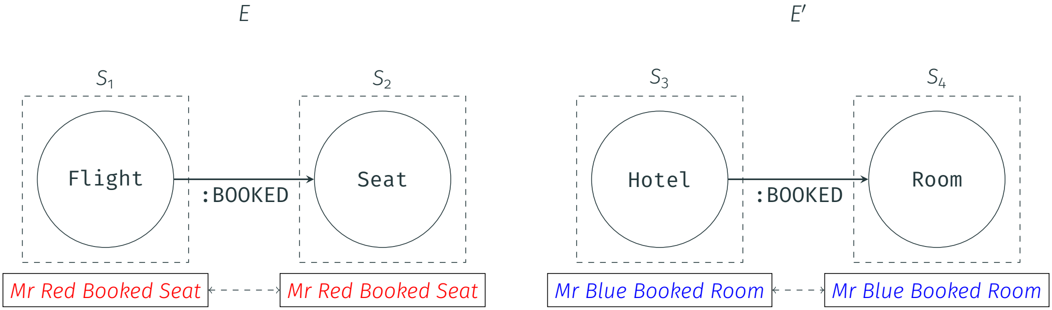


**Figure 10:** Edge-order consistency violation

# Our Solution: Order Arbitration

- Transactions collect predecessors from each update

# Our Solution: Order Arbitration

- Transactions collect predecessors from each update
- Centralized arbiter maintains some global state

# Our Solution: Order Arbitration

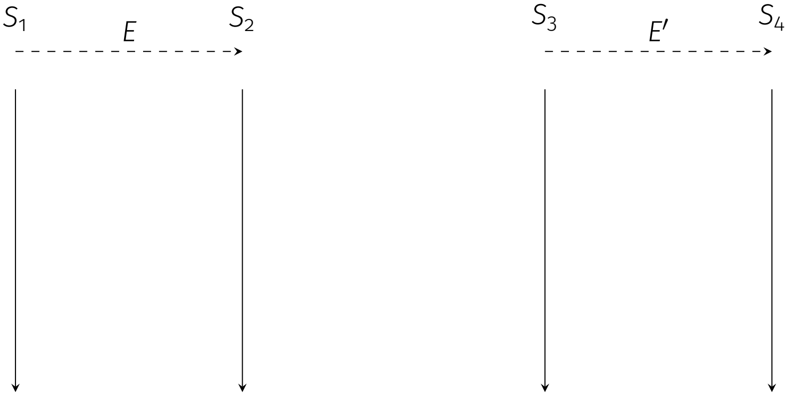
- Transactions collect predecessors from each update
- Centralized arbiter maintains some global state
- Global state used to detect edge-order violations

# Our Solution: Order Arbitration

- Transactions collect predecessors from each update
- Centralized arbiter maintains some global state
- Global state used to detect edge-order violations
- Offending transactions are aborted

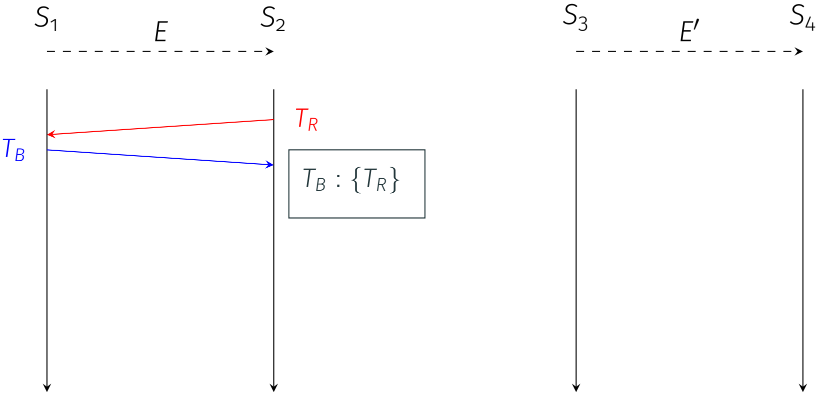
# Order Arbitration

Rule: Collect predecessors after every successful update



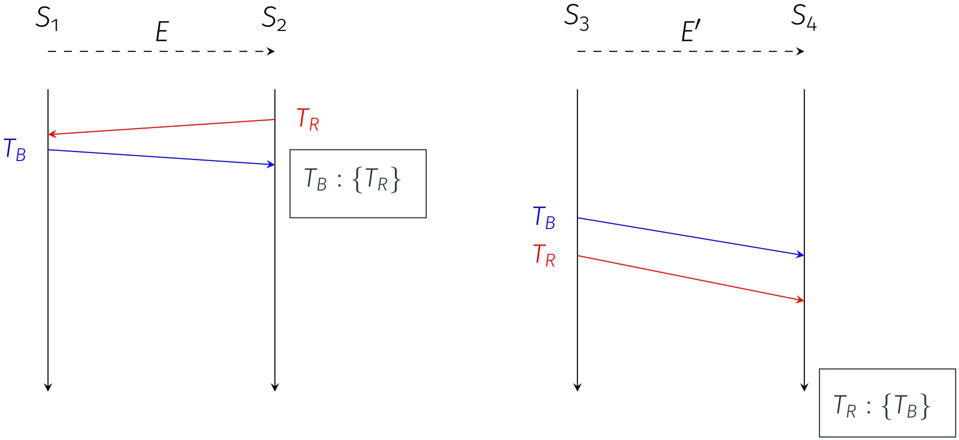
# Order Arbitration

Rule: Collect predecessors after every successful update



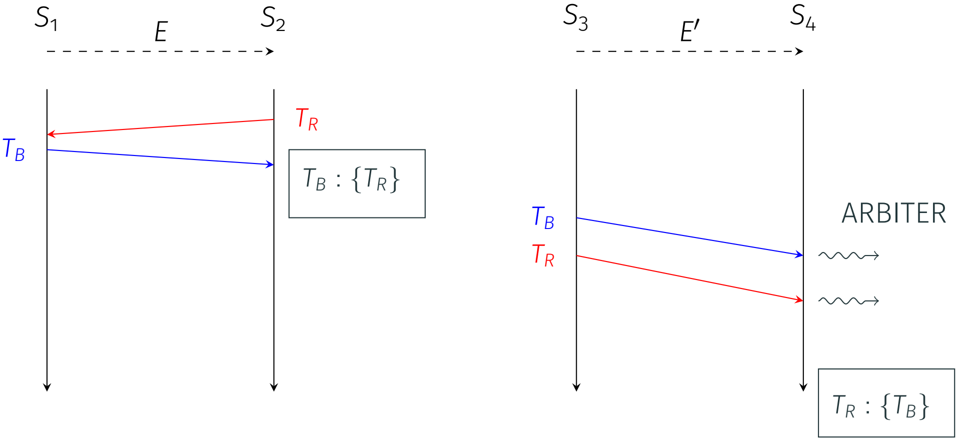
# Order Arbitration

Rule: Collect predecessors after every successful update



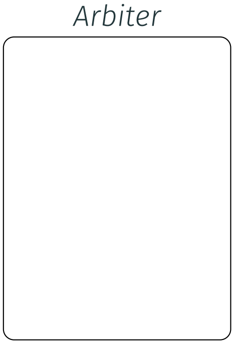
# Order Arbitration

Rule: Collect predecessors after every successful update



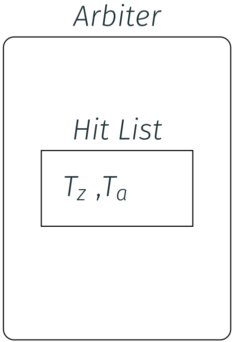
# Order Arbitration

Rule: If transaction exists in hit list, abort. Else, merge predecessors into hit list



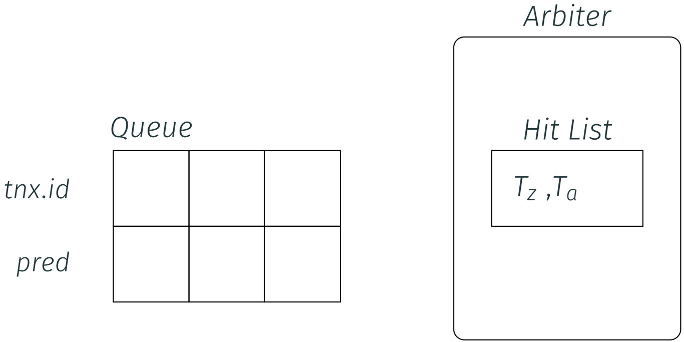
# Order Arbitration

Rule: If transaction exists in hit list, abort. Else, merge predecessors into hit list



# Order Arbitration

Rule: If transaction exists in hit list, abort. Else, merge predecessors into hit list



*Arbiter*

*Queue*

*tnx.id*

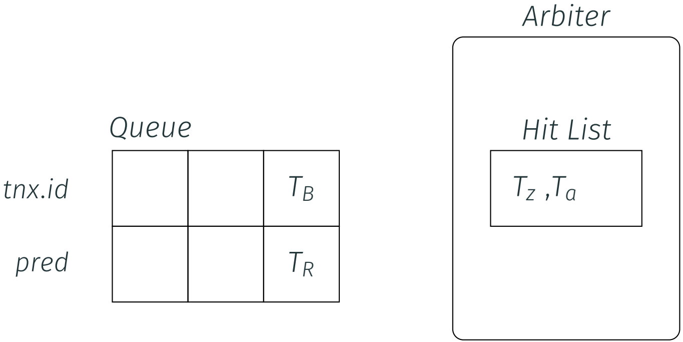
*pred*

*Hit List*

$T_z$ ,$T_a$

# Order Arbitration

Rule: If transaction exists in hit list, abort. Else, merge predecessors into hit list



*Queue*

| | | | |
|---|---|---|---|
| *tnx.id* | | | $T_B$ |
| *pred* | | | $T_R$ |

*Arbiter*

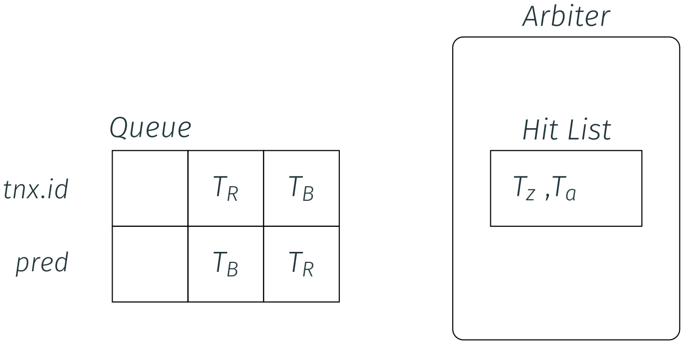
*Hit List*

$T_z$ ,$T_a$

$T_B$ ENTERS QUEUE

# Order Arbitration

Rule: If transaction exists in hit list, abort. Else, merge predecessors into hit list



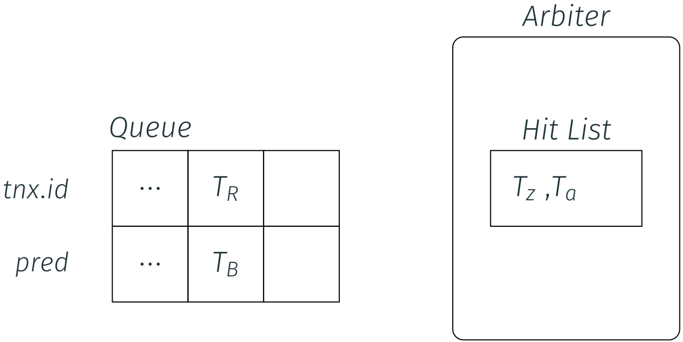
Queue

| | | | |
|---|---|---|---|
| *tnx.id* | | $T_R$ | $T_B$ |
| *pred* | | $T_B$ | $T_R$ |

$T_R$ ENTERS QUEUE

# Order Arbitration

Rule: If transaction exists in hit list, abort. Else, merge predecessors into hit list



*Queue*

| | | | |
|---|---|---|---|
| *tnx.id* | $\cdots$ | $T_R$ | |
| *pred* | $\cdots$ | $T_B$ | |

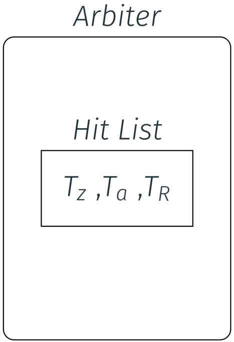
$T_B$ PROCESSED

# Order Arbitration

Rule: If transaction exists in hit list, abort. Else, merge predecessors into hit list

*Queue*

| | | | |
|---|---|---|---|
| *tnx.id* | … | $T_R$ | |
| *pred* | … | $T_B$ | |

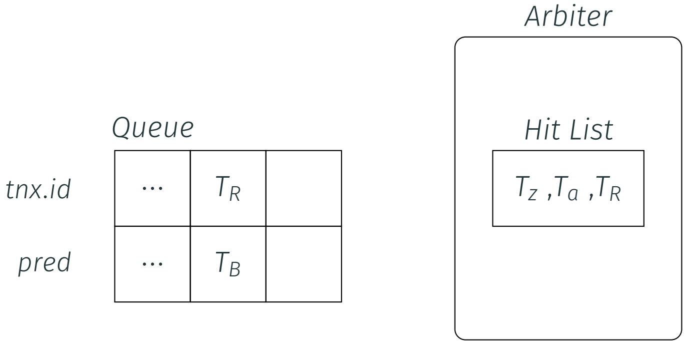


$T_B$ PROCESSED

$T_R$ ADDED TO HIT LIST

# Order Arbitration

Rule: If transaction exists in hit list, abort. Else, merge predecessors into hit list



| | | | |
|---|---|---|---|
| *tnx.id* | … | $T_R$ | |
| *pred* | … | $T_B$ | |

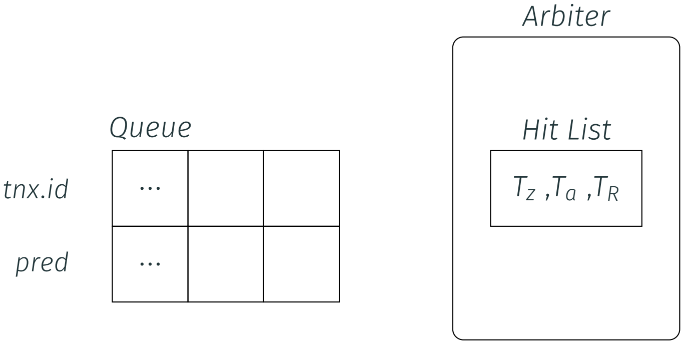
$T_B$ PROCESSED

$T_R$ ADDED TO HIT LIST

$T_B$ SUCCEEDS

# Order Arbitration

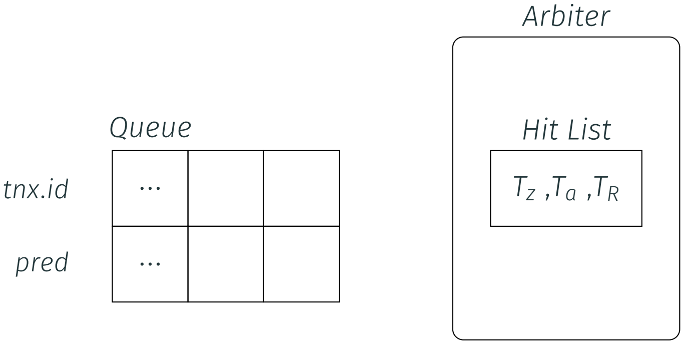
Rule: If transaction exists in hit list, abort. Else, merge predecessors into hit list



$T_R$ PROCESSED

# Order Arbitration

Rule: If transaction exists in hit list, abort. Else, merge predecessors into hit list



*Queue*

| | | | |
|---|---|---|---|
| *tnx.id* | ··· | | |
| *pred* | ··· | | |

*Arbiter*

*Hit List*

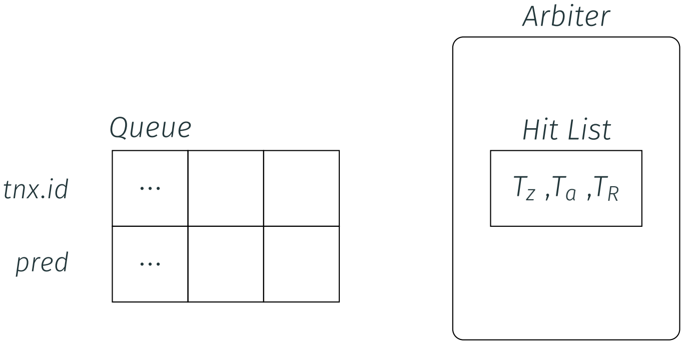
$T_z$ ,$T_a$ ,$T_R$

$T_R$ PROCESSED

$T_R$ IN HIT LIST

# Order Arbitration

Rule: If transaction exists in hit list, abort. Else, merge predecessors into hit list



*Queue*

| | | | |
|---|---|---|---|
| *tnx.id* | $\cdots$ | | |
| *pred* | $\cdots$ | | |

$T_R$ PROCESSED

$T_R$ IN HIT LIST

$T_R$ ABORTS

# Order Arbitration Protocol

If a transaction updates 1 or more distributed edges:

- Collect all predecessors

# Order Arbitration Protocol

If a transaction updates 1 or more distributed edges:

- Collect all predecessors
- Go to arbiter

# Order Arbitration Protocol

If a transaction updates 1 or more distributed edges:

- Collect all predecessors
- Go to arbiter
- If not in hit list then proceed and enter predecessors into the hit list

# Order Arbitration Protocol

If a transaction updates 1 or more distributed edges:

- Collect all predecessors
- Go to arbiter
- If not in hit list then proceed and enter predecessors into the hit list
- Else in hit list then abort

# Edge Concurrency Control Protocol Summary

1. Collision detection: enforces **reciprocal consistency** for distributed edges
2. Order arbitration: enforces **edge-order consistency** between transactions

# Performance Evaluation

Performance measures of interest:

- Average number of transactions that are aborted
- Load at the arbiter

# Evaluation Approach

1. Developed approximate model
2. Measure accuracy of model through simulation

# Parameters

- Database size
- Transaction arrival rate
- Average updates per transaction
- Average arbiter service time
- Average network delay

# Aborts per second ($R$) vs Database Size ($N$)

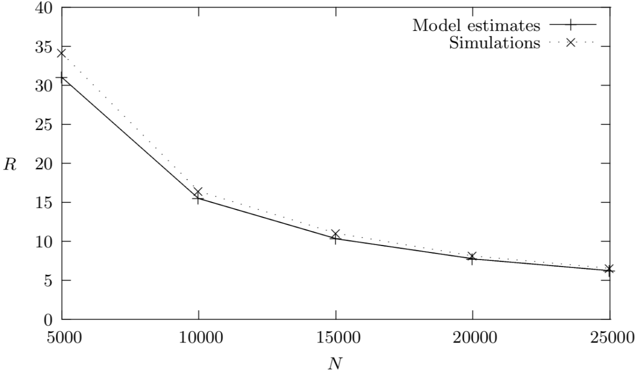


**Figure 11:** TPS $= 1000$, av updates/tnx $= 5$, av arbiter service time $= 10ms$, av network delay $= 5ms$

# Aborts per second ($R$) vs Transaction Arrival Rate ($\lambda$)

Arbiter queue unstable at $\sim$ 1500 TPS

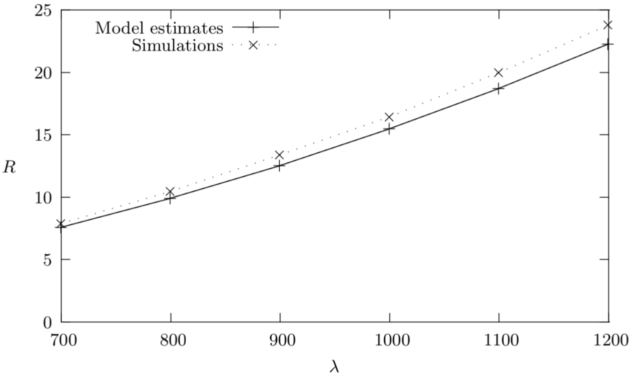


Figure 12: size $= 10K$, av updates/tnx $= 5$, av arbiter service time $= 10ms$, av network delay $= 5ms$

# Aborts per second ($R$) vs Transaction Arrival Rate ($\lambda$)

Arbiter queue unstable at $\sim$ 1100 TPS

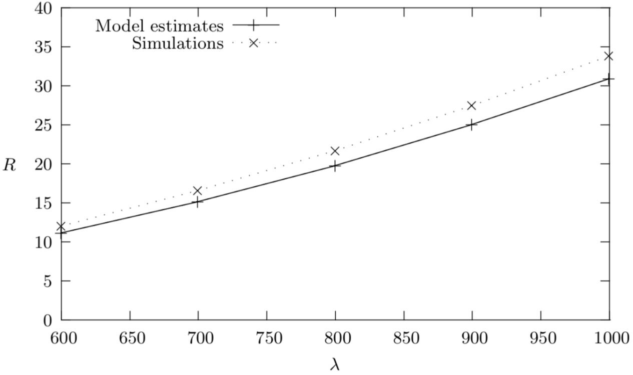


**Figure 13:** size $= 10K$, av updates/tnx $= 5$, av arbiter service time $= 10ms$, av network delay $= 10ms$

# Aborts per second ($R$) vs Transaction Arrival Rate ($\lambda$)

Arbiter queue unstable at $\sim$ 550 TPS

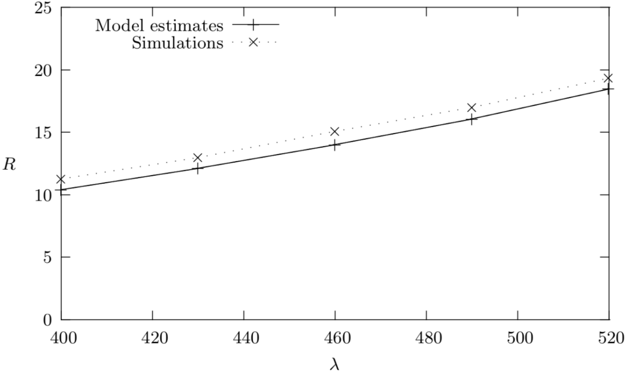


**Figure 14:** size $=$ 10$K$, av updates/tnx $=$ 10, av arbiter service time $=$ 10$ms$, av network delay $=$ 5$ms$

# Summary and Future Work

- Model accuracy similar to simulations under a variety of parameter settings

# Summary and Future Work

- Model accuracy similar to simulations under a variety of parameter settings
- Between 1-4% transactions are aborted

# Summary and Future Work

- Model accuracy similar to simulations under a variety of parameter settings
- Between 1-4% transactions are aborted
- Improve accuracy of simulation

## Thanks for listening!

## Any Questions?

Email: j.waudby2@newcastle.ac.uk

Twitter: @waudberry_7