Fast Abort-Freedom for Deterministic Transactions

Chen Chen[1], Xingbo Wu[2], Wenshao Zhong[1], Jakob Eriksson[1]

[1] University of Illinois at Chicago

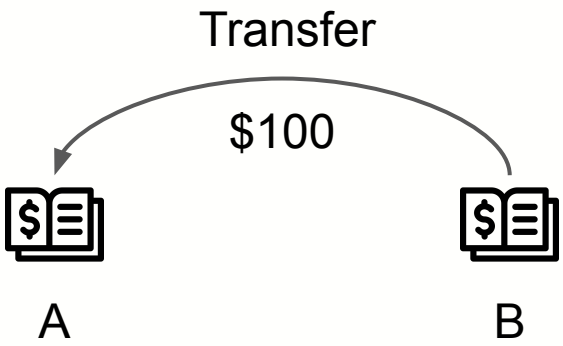
[2] Microsoft Research

Concurrent Transactions

- Transaction: a logical unit of multiple data access operations

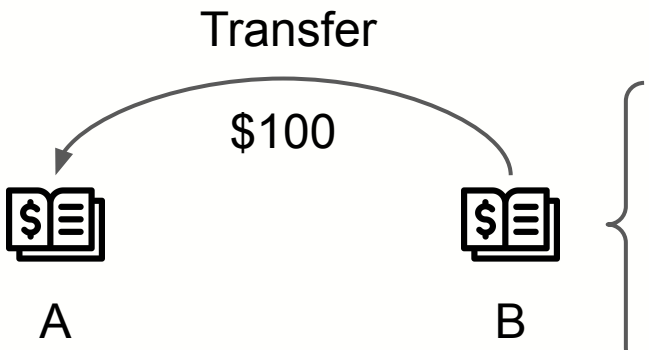
Concurrent Transactions

- Transaction: a logical unit of multiple data access operations



Concurrent Transactions

- Transaction: a logical unit of multiple data access operations



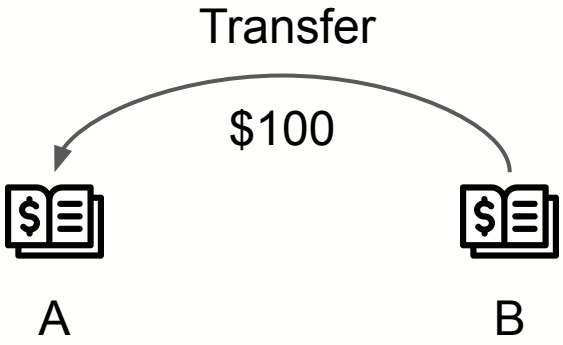

```
Procedures

1) Check balance of B
2) Subtract $100 from B
3) Add $100 to A
```

Concurrent Transactions

- Transaction: a logical unit of multiple data access operations

Atomic **Consistent** **Isolated**




```
Procedures

1) Check balance of B
2) Subtract $100 from B
3) Add $100 to A
```

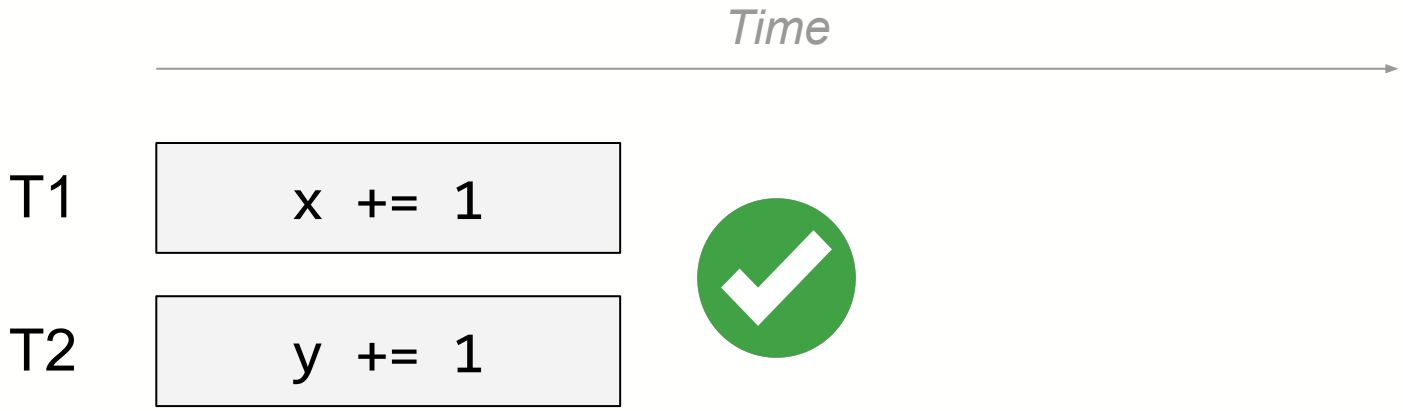
A lot of transactions concurrently

Concurrency Control

- Properly utilizing hardware parallelism

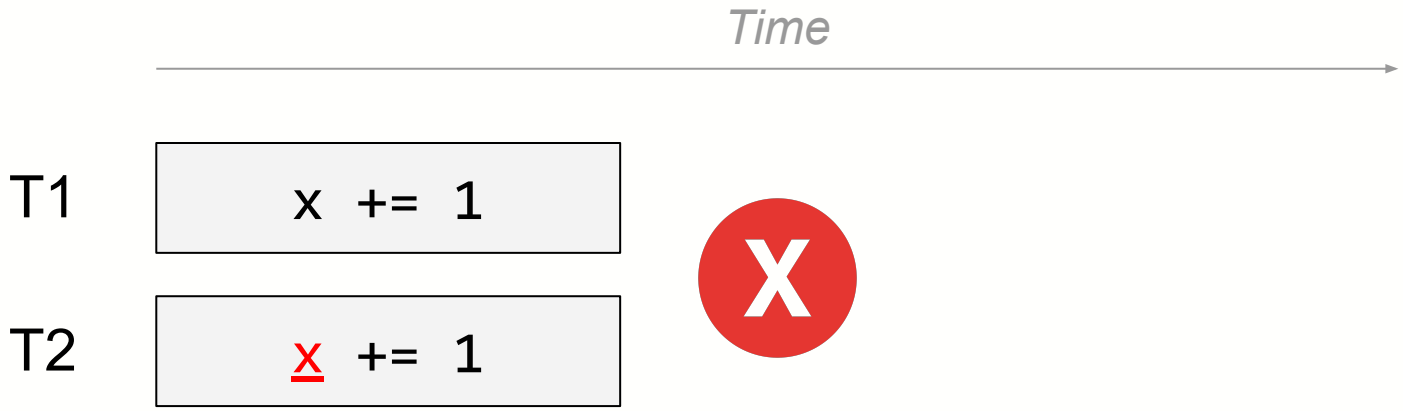
Concurrency Control

- Properly utilizing hardware parallelism



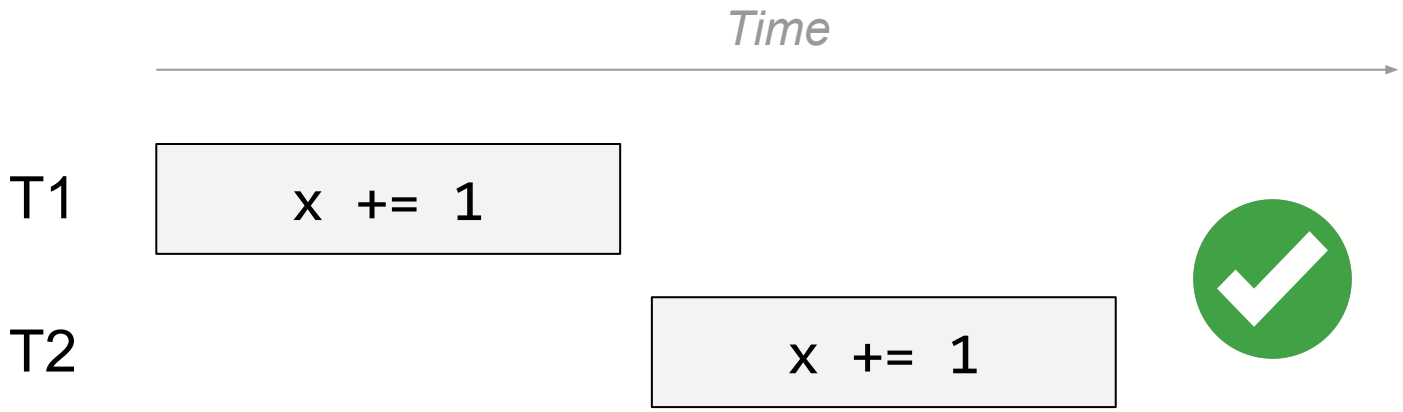
Concurrency Control

- Properly utilizing hardware parallelism



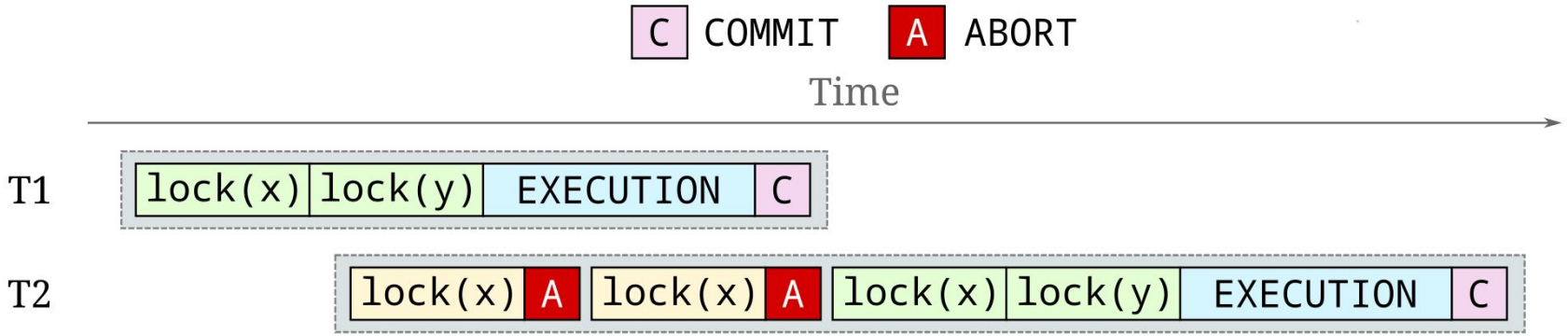
Concurrency Control

- Properly utilizing hardware parallelism



Concurrency Control

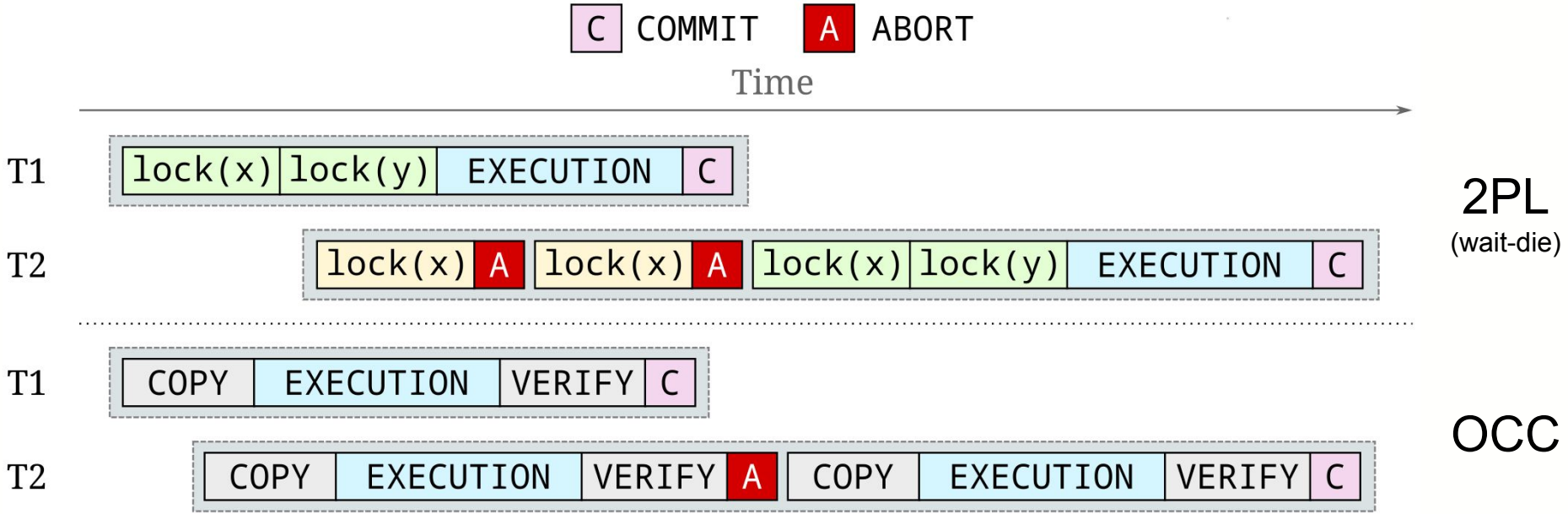
- Pessimistic, locking based: 2-phase locking (2PL)



2PL
(wait-die)

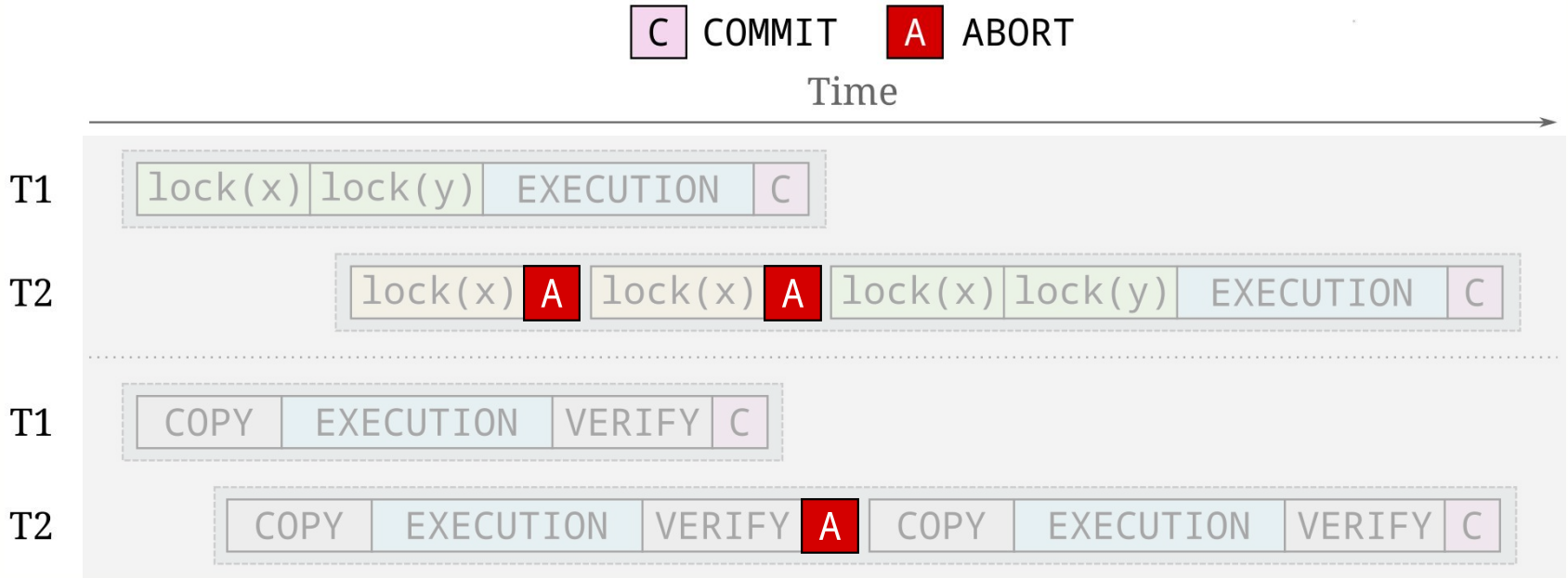
Concurrency Control

- Pessimistic, locking based: 2-phase locking (2PL)
- Optimistic concurrency control (OCC)



Concurrency Control

- Pessimistic, locking based: 2-phase locking (2PL)
- Optimistic concurrency control (OCC)



Abort and deterministic transactions

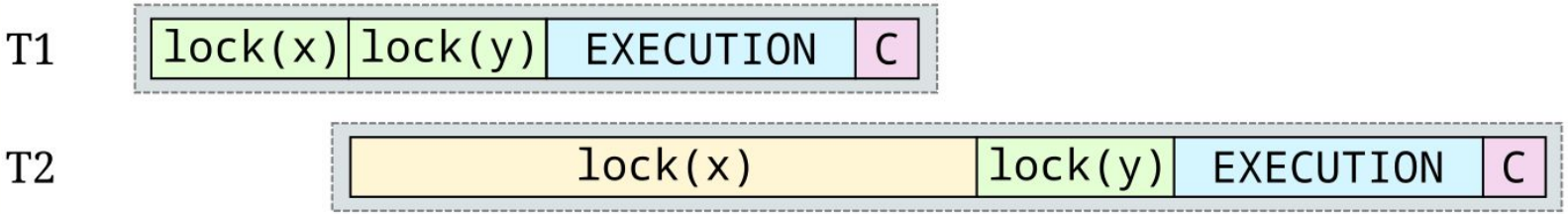
- Abort is inevitable if:
  - The data access of a transaction is unknown

Abort and deterministic transactions

- Abort is inevitable if:
  - The data access of a transaction is unknown
- Deterministic transactions bring new opportunities
  - w/ known read set and write set

Abort and deterministic transactions

- Abort is inevitable if:
  - The data access of a transaction is unknown
- Deterministic transactions bring new opportunities
  - w/ known read set and write set
- Exploit determinism:
  - Ordered locking

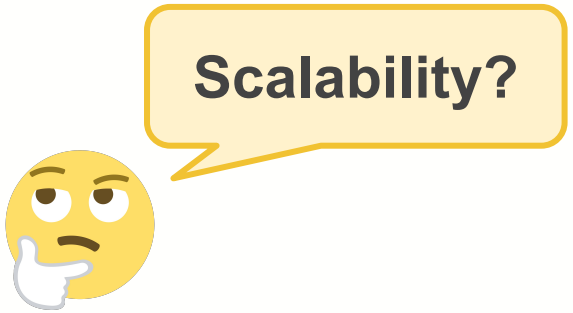


Abort and deterministic transactions

- Abort is inevitable if:
  - The data access of a transaction is unknown
- Deterministic transactions bring new opportunities
  - w/ known read set and write set
- Exploit determinism:
  - Ordered locking
  - Batched scheduling

Abort and deterministic transactions

- Abort is inevitable if:
  - The data access of a transaction is unknown
- Deterministic transactions bring new opportunities
  - w/ known read set and write set
- Exploit determinism:
  - Ordered locking
  - Batched scheduling



Idea

- Get the best of 2PL, scheduling and OCC

Idea

- Get the best of 2PL, scheduling and OCC
  - 2PL: locking
  - Scheduling: centralized scheduler
  - OCC: opportunistic commit

Idea

- Get the best of 2PL, scheduling and OCC
  - 2PL: ~~locking~~ **queuing**
  - Scheduling: ~~centralized scheduler~~ **decentralized scheduling**
  - OCC: ~~opportunistic commit~~ **deterministic commit order**

Idea

- Get the best of 2PL, scheduling and OCC
  - 2PL: ~~locking~~ **queuing**
  - Scheduling: ~~centralized scheduler~~ **decentralized scheduling**
  - OCC: ~~opportunistic commit~~ **deterministic commit order**

- Each transaction can observe their global order individually, with minimal cost.

Idea

- Get the best of 2PL, scheduling and OCC
  - 2PL: ~~locking~~ **queuing**
  - Scheduling: ~~centralized scheduler~~ **decentralized scheduling**
  - OCC: ~~opportunistic commit~~ **deterministic commit order**

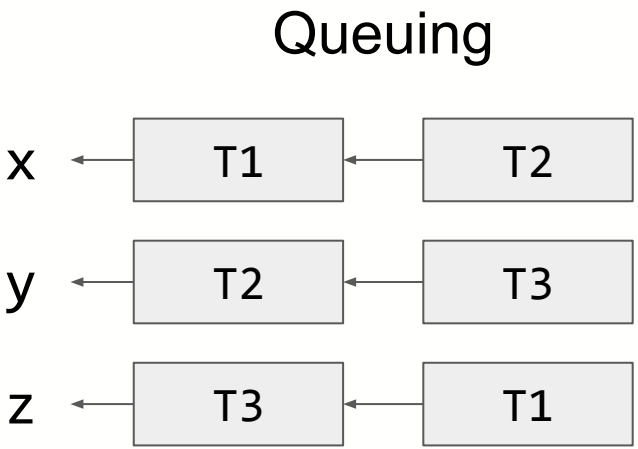
- Each transaction can observe their global order individually, with minimal cost.
- **DecentSched**

DecentSched

- Three transactions accessing three variables
  - `T1: x, z`
  - `T2: x, y`
  - `T3: y, z`

DecentSched

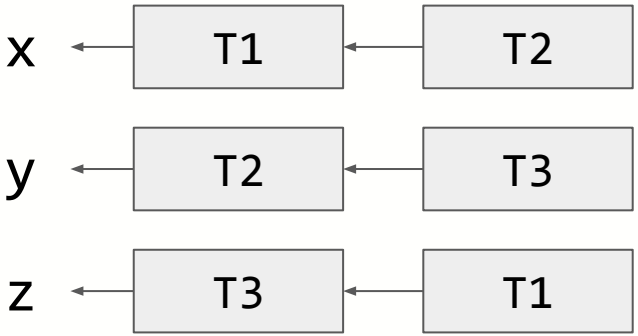
- Three transactions accessing three variables
  - `T1: x, z`
  - `T2: x, y`
  - `T3: y, z`



DecentSched

- Three transactions accessing three variables
  - `T1: x, z`
  - `T2: x, y`
  - `T3: y, z`

Queuing



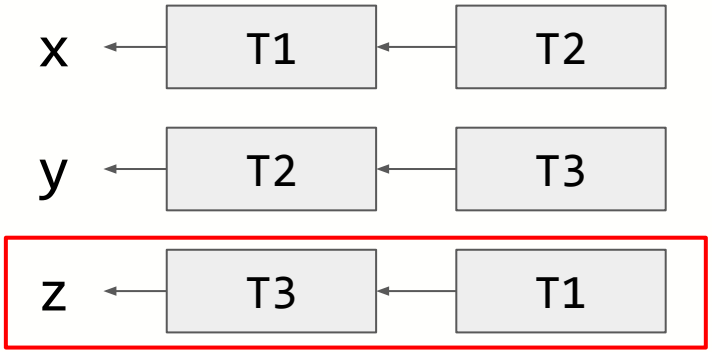
Dependency tracking

T1 T2 T3

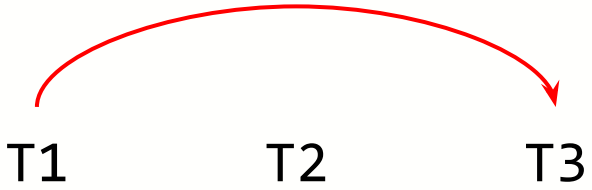
DecentSched

- Three transactions accessing three variables
  - `T1: x, z`
  - `T2: x, y`
  - `T3: y, z`

Queuing



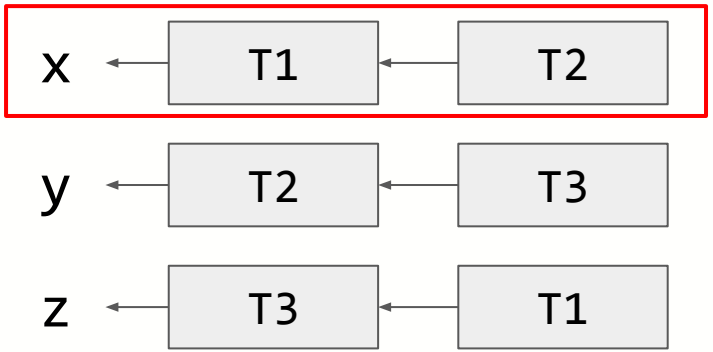
Dependency tracking



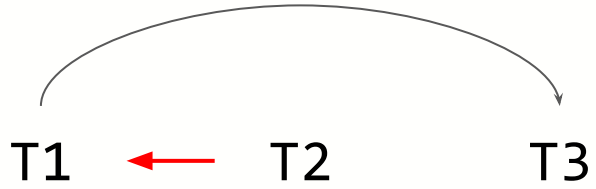
DecentSched

- Three transactions accessing three variables
  - `T1: x, z`
  - `T2: x, y`
  - `T3: y, z`

Queuing



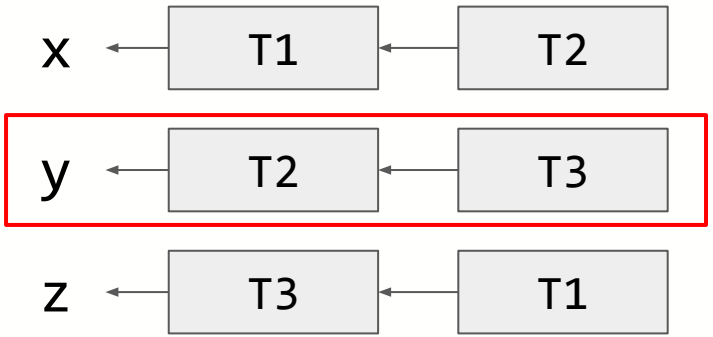
Dependency tracking



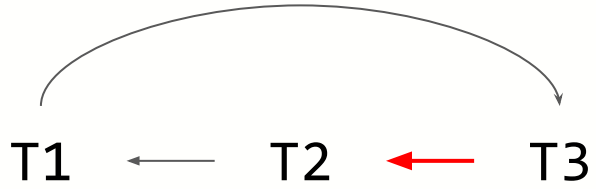
DecentSched

- Three transactions accessing three variables
  - `T1: x, z`
  - `T2: x, y`
  - `T3: y, z`

Queuing



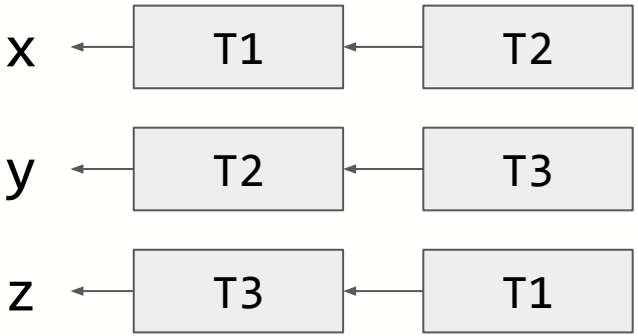
Dependency tracking



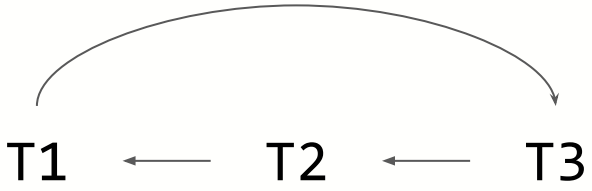
DecentSched

- Three transactions accessing three variables
  - `T1: x, z`
  - `T2: x, y`
  - `T3: y, z`

Queuing

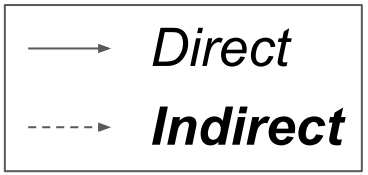


Dependency tracking

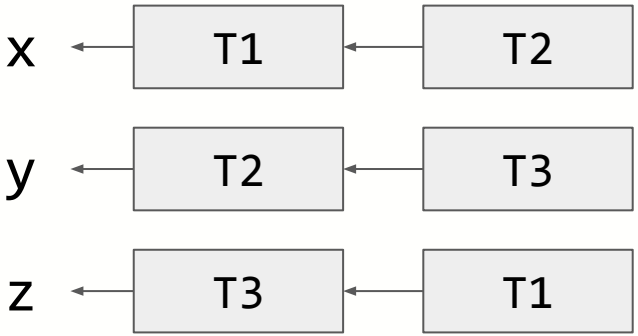


DecentSched

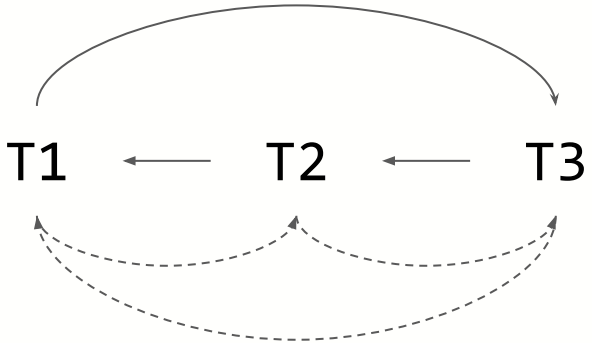
- Three transactions accessing three variables
  - `T1: x, z`
  - `T2: x, y`
  - `T3: y, z`



Queuing



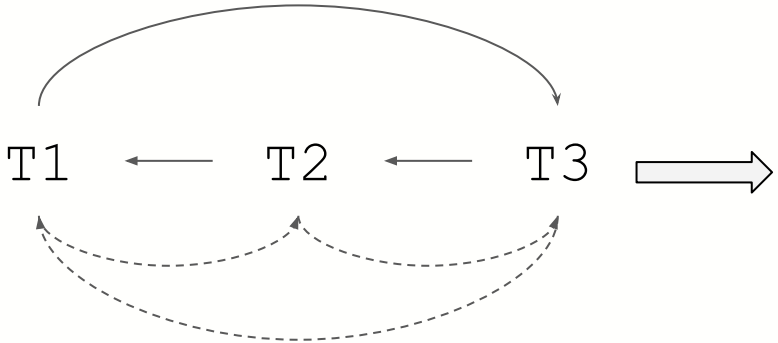
Dependency tracking



DecentSched

- Three transactions accessing three variables
  - T1: x, z
  - T2: x, y
  - T3: y, z

Dependency tracking



Dependency set

T1: { $T3_{(direct)}$, $T2_{(indirect)}$ }
T2: { $T1_{(direct)}$, $T3_{(indirect)}$ }
T3: { $T2_{(direct)}$, $T1_{(indirect)}$ }

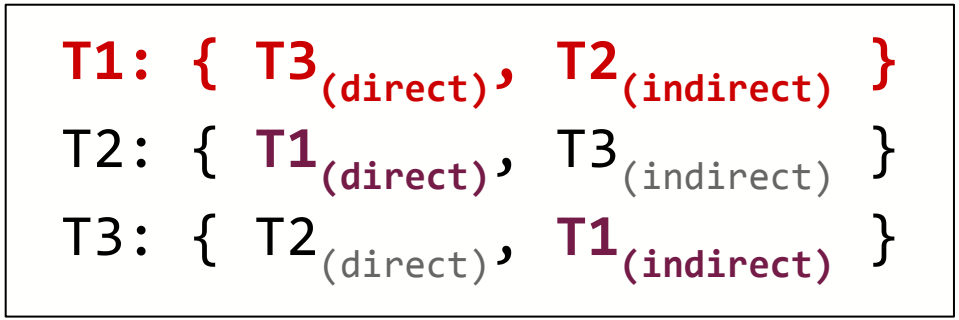
DecentSched

- Scheduling transaction T is done by evaluating each transaction T’ in its dependency set.

1. Ignore T’ if If T’ and T are not each other’s direct dependencies.
2. Otherwise:
   a. Unidirectional -> unconditional wait
   b. Bidirectional (cyclic) -> lower id executes first

DecentSched

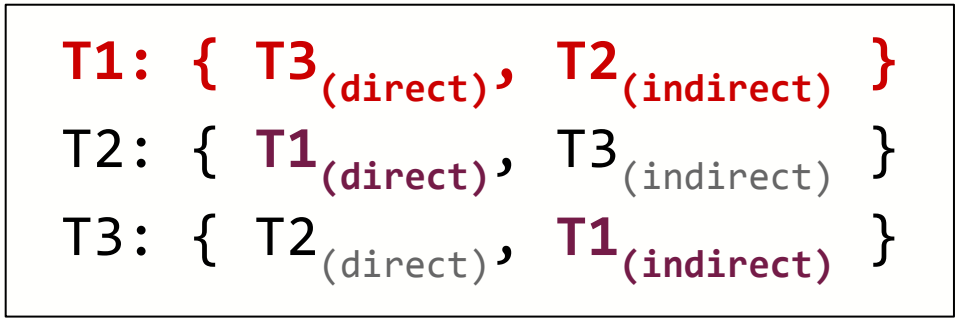
- Scheduling transaction T is done by evaluating each transaction T’ in its dependency set.



Dependency set

DecentSched

- Scheduling transaction T is done by evaluating each transaction T’ in its dependency set.



Dependency set

T1: precedes T3 and T2

DecentSched

- Scheduling transaction T is done by evaluating each transaction T’ in its dependency set.

```
T1: { T3(direct), T2(indirect) }
T2: { T1(direct), T3(indirect) }
T3: { T2(direct), T1(indirect) }
```

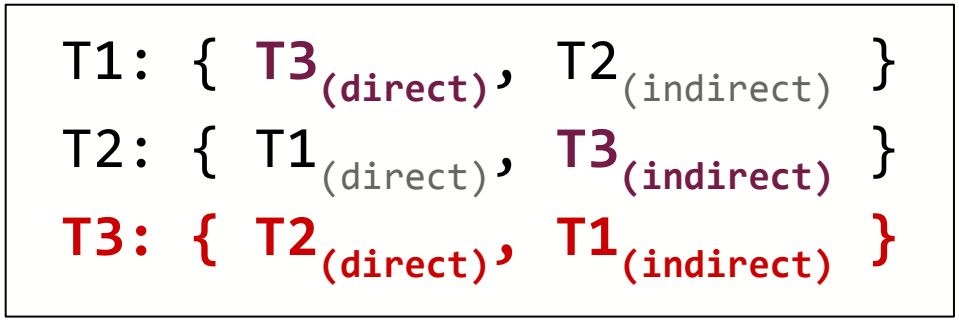
Dependency set

T1: precedes T3 and T2

T2: waits for T1, precedes T3

DecentSched

- Scheduling transaction T is done by evaluating each transaction T’ in its dependency set.



Dependency set

T1: precedes T3 and T2

T2: waits for T1, precedes T3

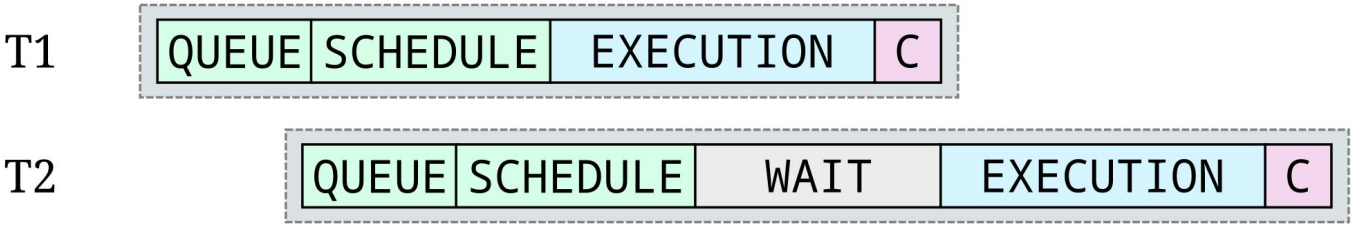
T3: waits for T1 and T2

DecentSched

- Advantages in comparison

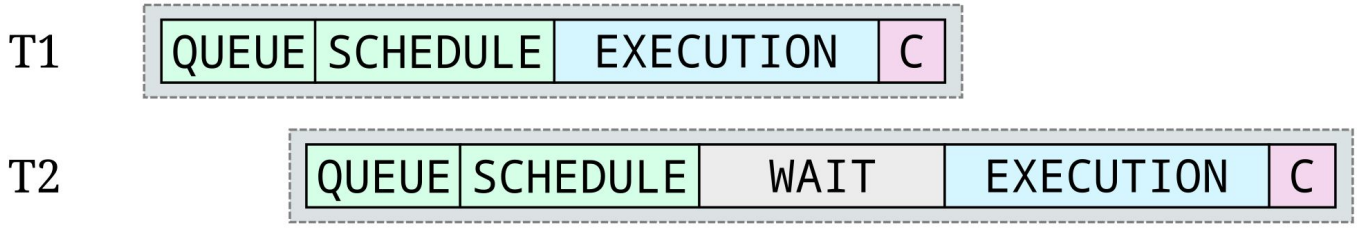
DecentSched

- Advantages in comparison
  - Compared to centralized scheduling:
    - **No scheduler bottleneck**



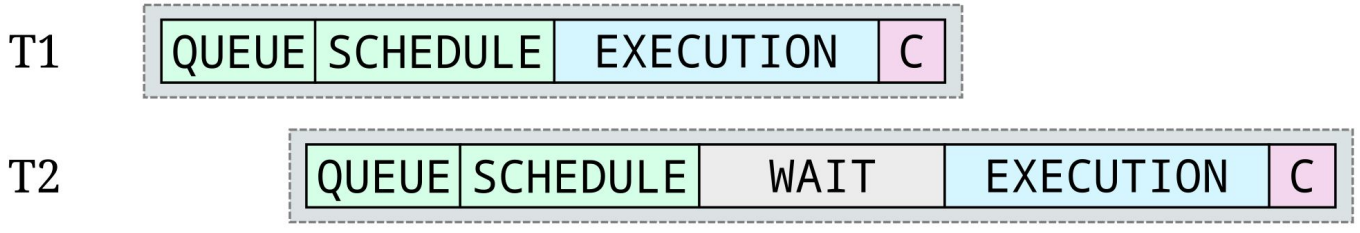
DecentSched

- Advantages in comparison
  - Compared to centralized scheduling:
    - No scheduler bottleneck
  - Compared to Locking:
    - **No lock contentions**

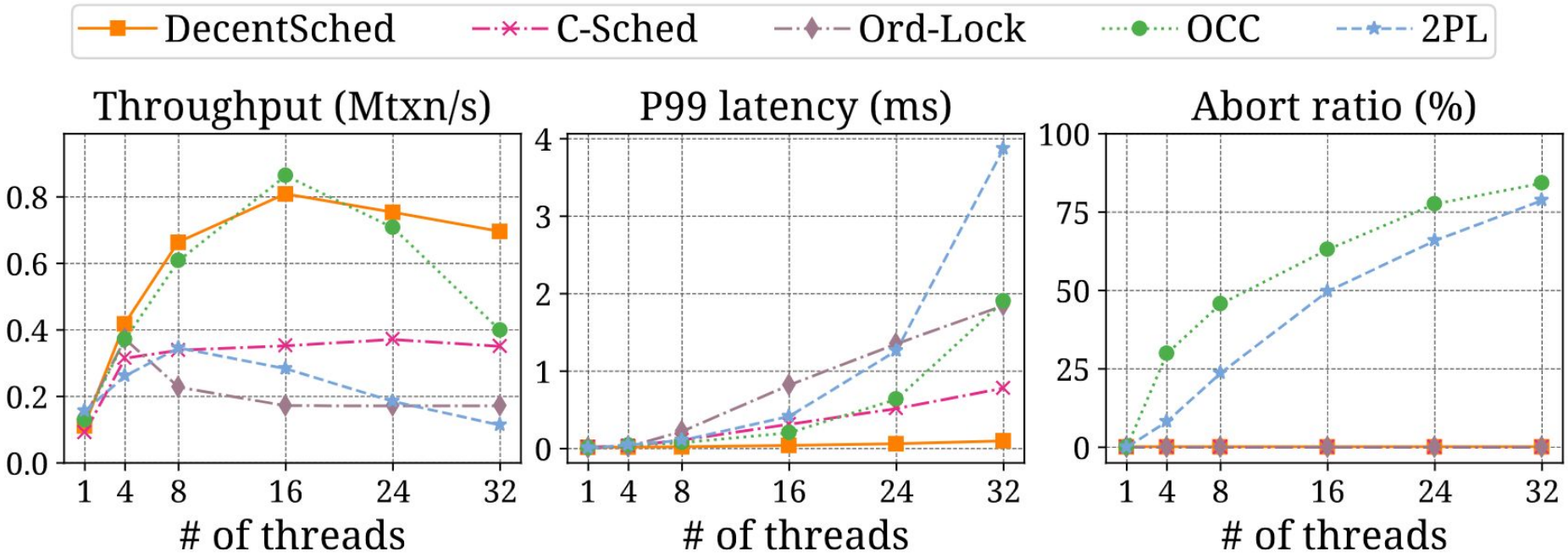


DecentSched

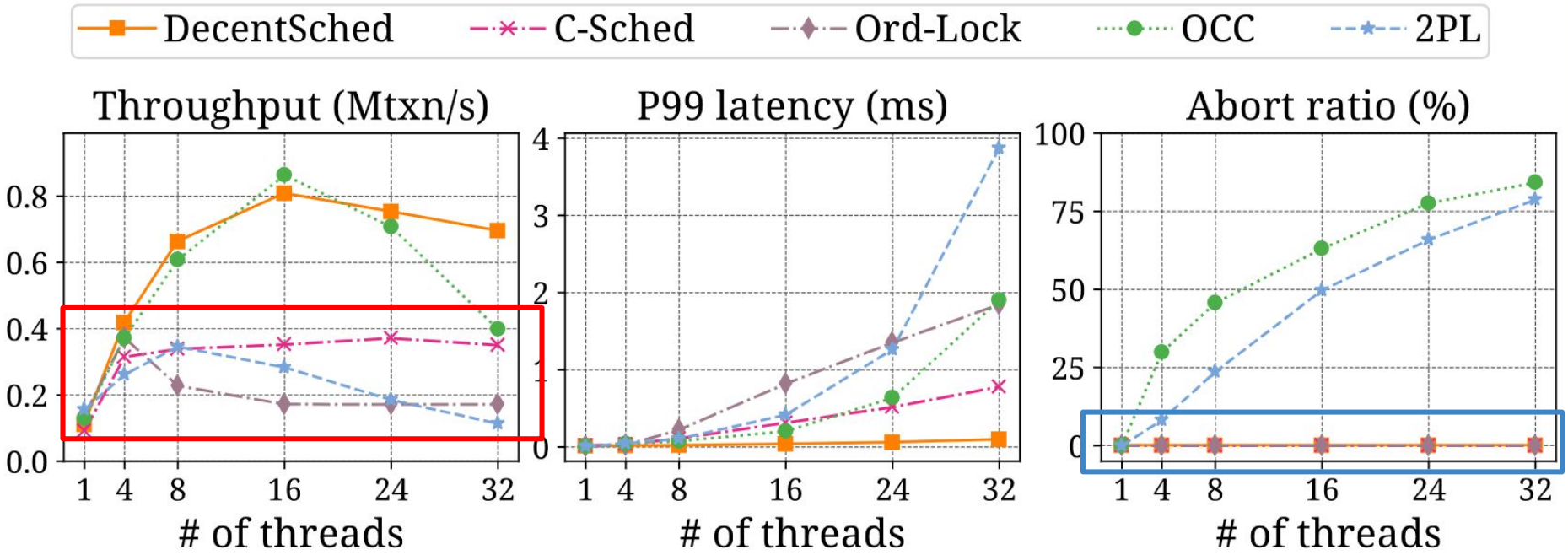
- Advantages in comparison
  - Compared to centralized scheduling:
    - No scheduler bottleneck
  - Compared to Locking:
    - No lock contentions
  - Compared to OCC:
    - **Deterministic commit order**



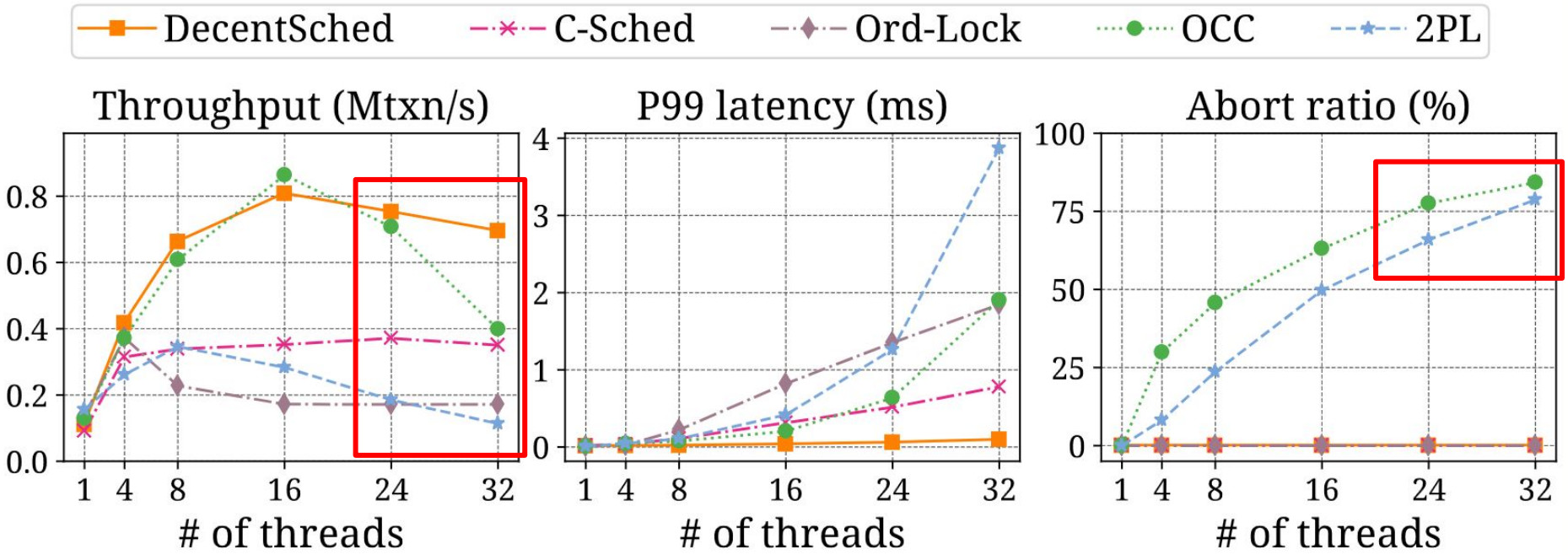
Evaluation: Transactional YCSB-A



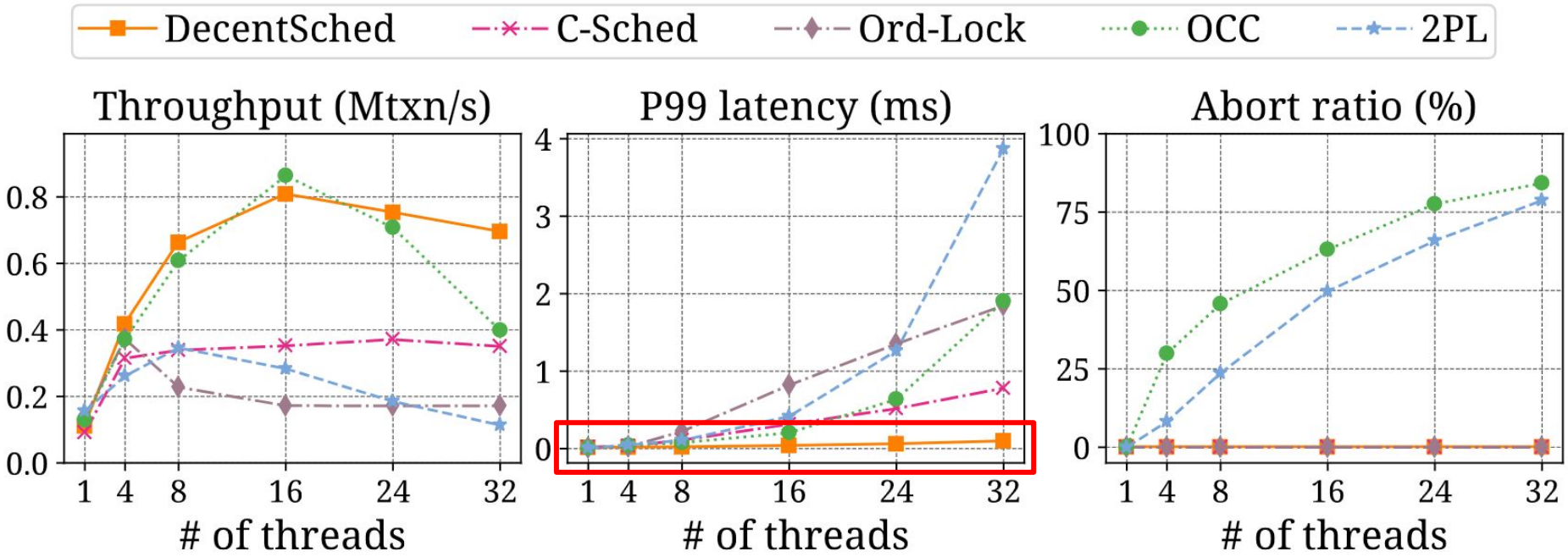
Evaluation: Transactional YCSB-A



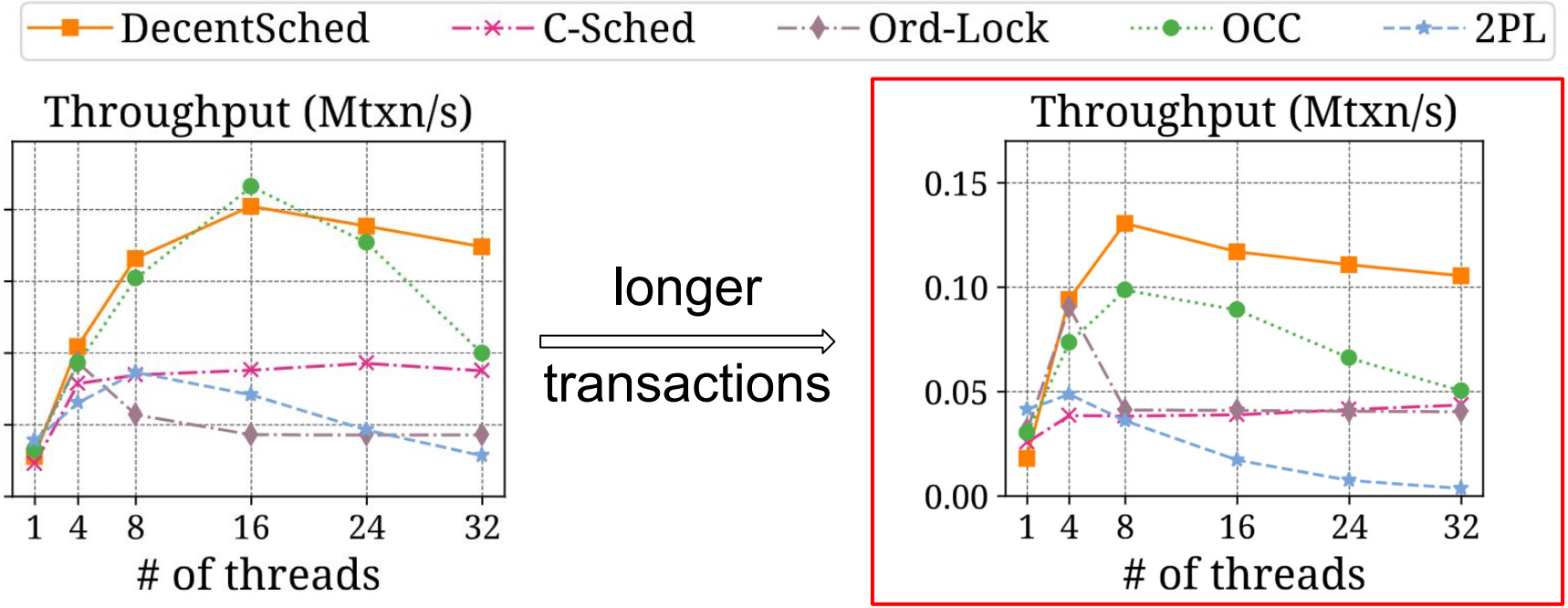
Evaluation: Transactional YCSB-A



Evaluation: Transactional YCSB-A



Evaluation: Transactional YCSB-A



More Details in the Paper

- Implementation and optimizations
- Memory management
- TPC-C Benchmark Results
- Future works

- Code and artifacts are available on 
  https://github.com/decentralized-scheduling/