Distributed Systems

Spring Semester 2020

Lecture 10: Optimistic Concurrency Control (Thor)

John Liagouris
liagos@bu.edu

Why this paper

- Focus on optimistic concurrency control (OCC)
  - Distributed OCC very interesting -- still active research
- Use of wall-clock time for transaction ordering to check serializability and provide external consistency
  - Rely on Network Time Protocol
  - Thor's use of time has been influential (e.g. Spanner)

http://research.microsoft.com/en-us/um/people/mcastro/publications/liskov96safe.pdf

http://www.pmg.csail.mit.edu/papers/ecoop99.pdf

http://www.pmg.csail.mit.edu/thor/Thor.0_User_Guide.ps

Why this paper

- Focus on optimistic concurrency control (OCC)
  - Distributed OCC very interesting -- still active research
- Use of wall-clock time for transaction ordering to check serializability and provide external consistency
  - Rely on Network Time Protocol
  - Thor's use of time has been influential (e.g. Spanner)

http://research.microsoft.com/en-us/um/people/mcastro/publications/liskov96safe.pdf

http://www.pmg.csail.mit.edu/papers/ecoop99.pdf

http://www.pmg.csail.mit.edu/thor/Thor.0_User_Guide.ps

Transaction (Databases)

- A unit of work (read-write operations) that is:
  - Atomic: Either the whole transaction succeeds or fails
  - Consistent: Does not violate database restrictions after completion
  - Isolated: Results of incomplete transactions are not visible to other transactions
  - Durable: After the transaction completes (commits) it cannot be lost

Serializability

- Not to be confused with linearizability!
- A correctness condition
- Concurrent execution of transactions must be equivalent to a serial execution
- If no equivalent serial execution exists, a transaction have read results of other incomplete transactions
- If serial order respects the real time the transactions were committed by the client, then:
  - External consistency

OCC vs Locking (Optimistic vs Pessimistic)

```
l.lock();
 val=read(x);
 val++;
 write(x);
l.unlock();
```

```
tx.begin()
 val=read(x);
 val++;
 write(x);
tx.end();
```

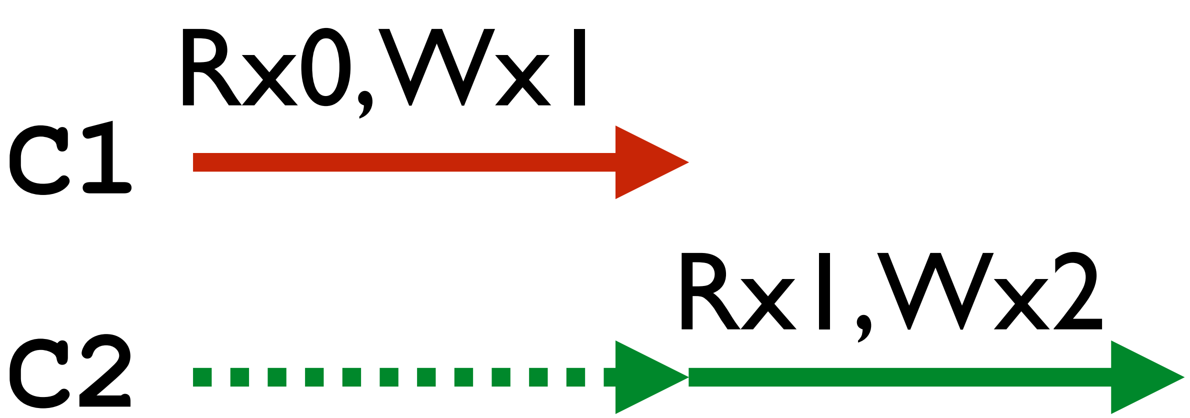
What is the difference between these two?

OCC vs Locking
(Optimistic vs Pessimistic)

```
l.lock();
 val=read(x);
 val++;
 write(x);
l.unlock();
```

```
tx.begin()
 val=read(x);
 val++;
 write(x);
tx.end();
```

Assume x=0

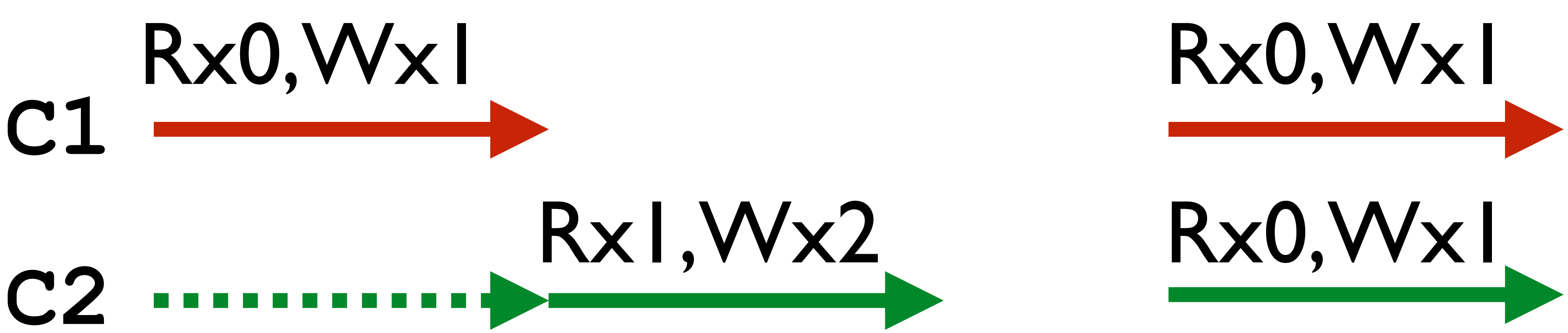


OCC vs Locking (Optimistic vs Pessimistic)

```
l.lock();
 val=read(x);
 val++;
 write(x);
l.unlock();
```

```
tx.begin()
 val=read(x);
 val++;
 write(x);
tx.end();
```

`Assume x=0`



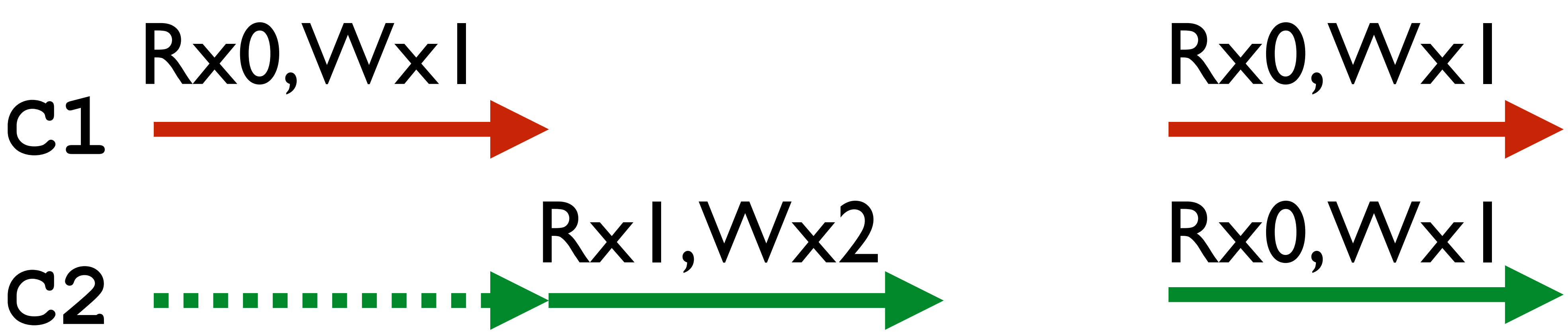
OCC vs Locking
(Optimistic vs Pessimistic)

```
l.lock();
 val=read(x);
 val++;
 write(x);
l.unlock();
```

```
tx.begin()
 val=read(x);
 val++;
```

OCC : Fast but WRONG

Assume



OCC vs Locking (Optimistic vs Pessimistic)

```
l.lock();
 val=read(x);
 val++;
 write(x);
l.unlock();
```

```
tx.begin()
 val=read(x);
 val++;
```

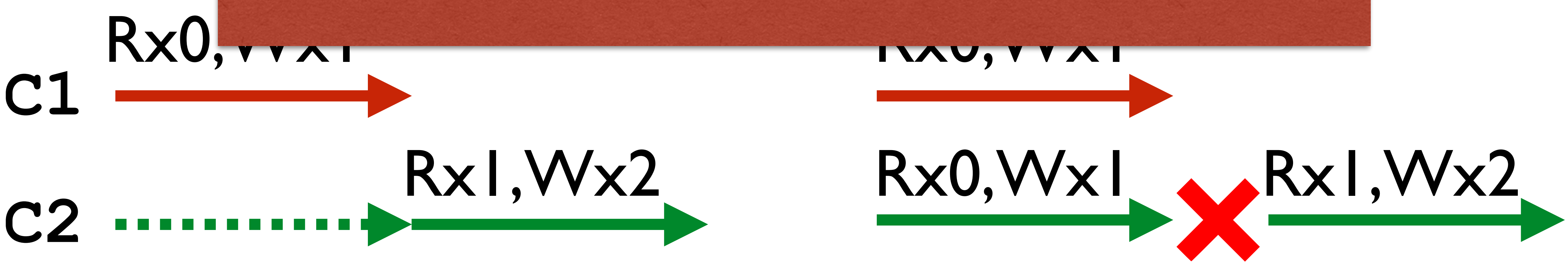
OCC :Must detect, abort one and retry — longer and more work

Assume

OCC vs Locking
(Optimistic vs Pessimistic)

OCC better if few conflicts otherwise overheads could be larger than lock waits in case of conflicts

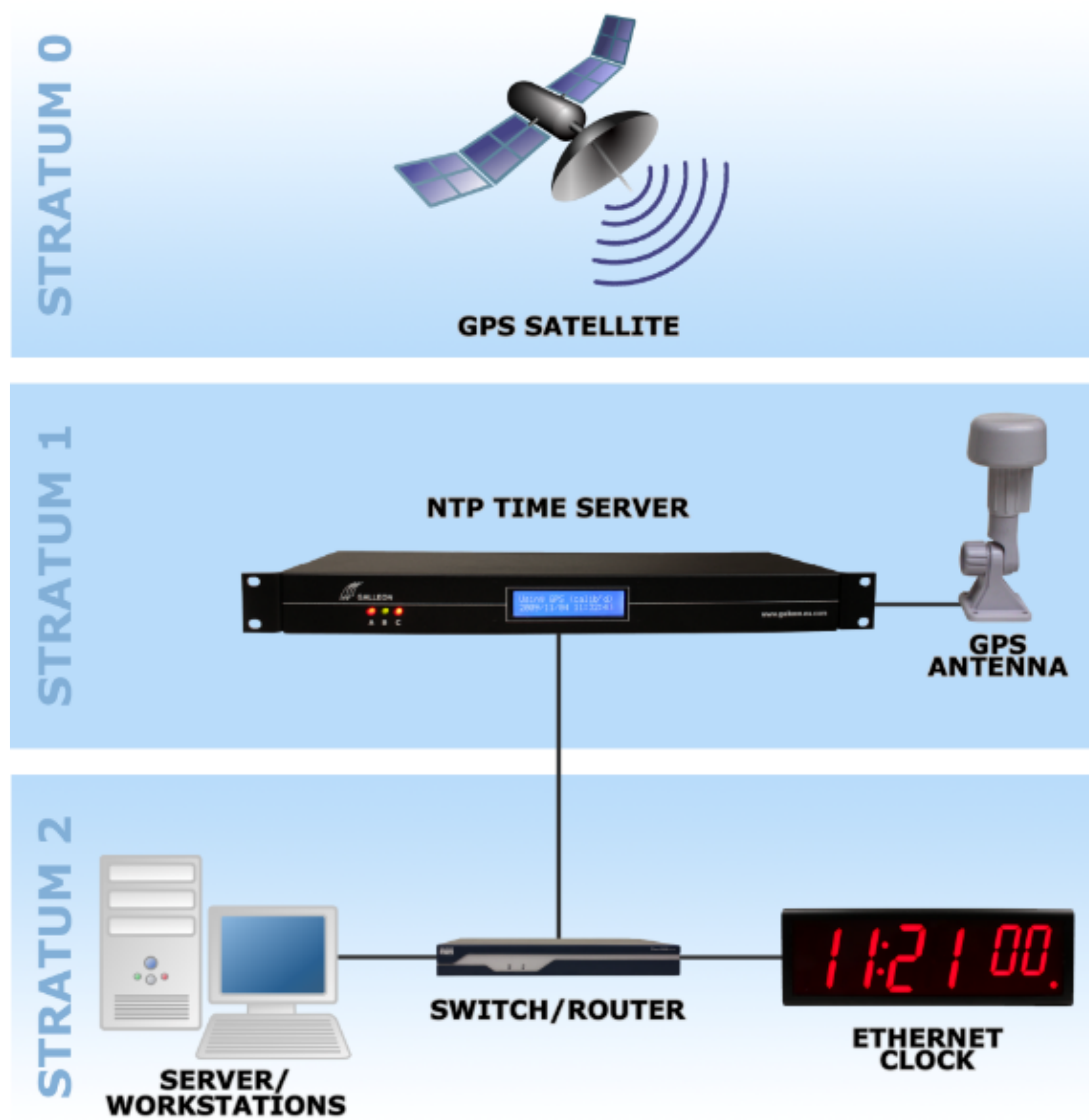
, abort
longer
rk



Network Time Protocol

- Clocks in different machines are not perfectly synchronized (skew, drift)
- Causes of clock skew: temperature variations, material imperfection, etc.
- Atomic clocks: Very accurate. Will not gain or lose a second in 300 million years!
- NTP uses an atomic clock as reference to allow machines in a network synchronise their clocks

Network Time Protocol



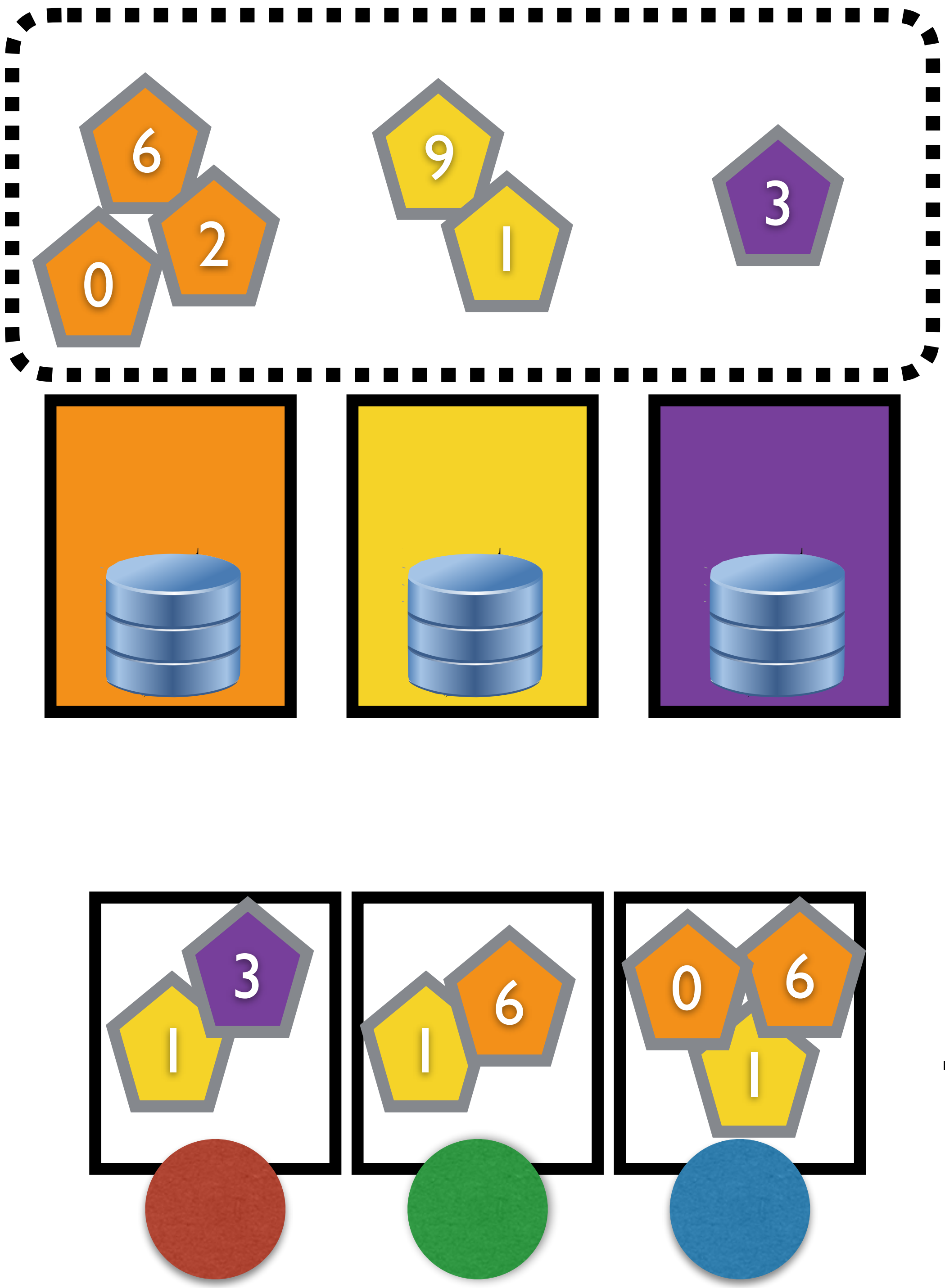
https://www.galsys.co.uk/news/what-is-ntp-a-beginners-guide-to-network-time-protocol/#What-is-NTP?

Network Time Protocol

- With NTP clock differences are in the order of 10s of *milliseconds*
- Geng et al. presented an approach to reduce clock skew to10-100s *nanoseconds*.
- Might have implications to the design and implementation of distributed systems algorithms

https://www.usenix.org/system/files/conference/nsdi18/nsdi18-geng.pdf

Thor: Distributed DB



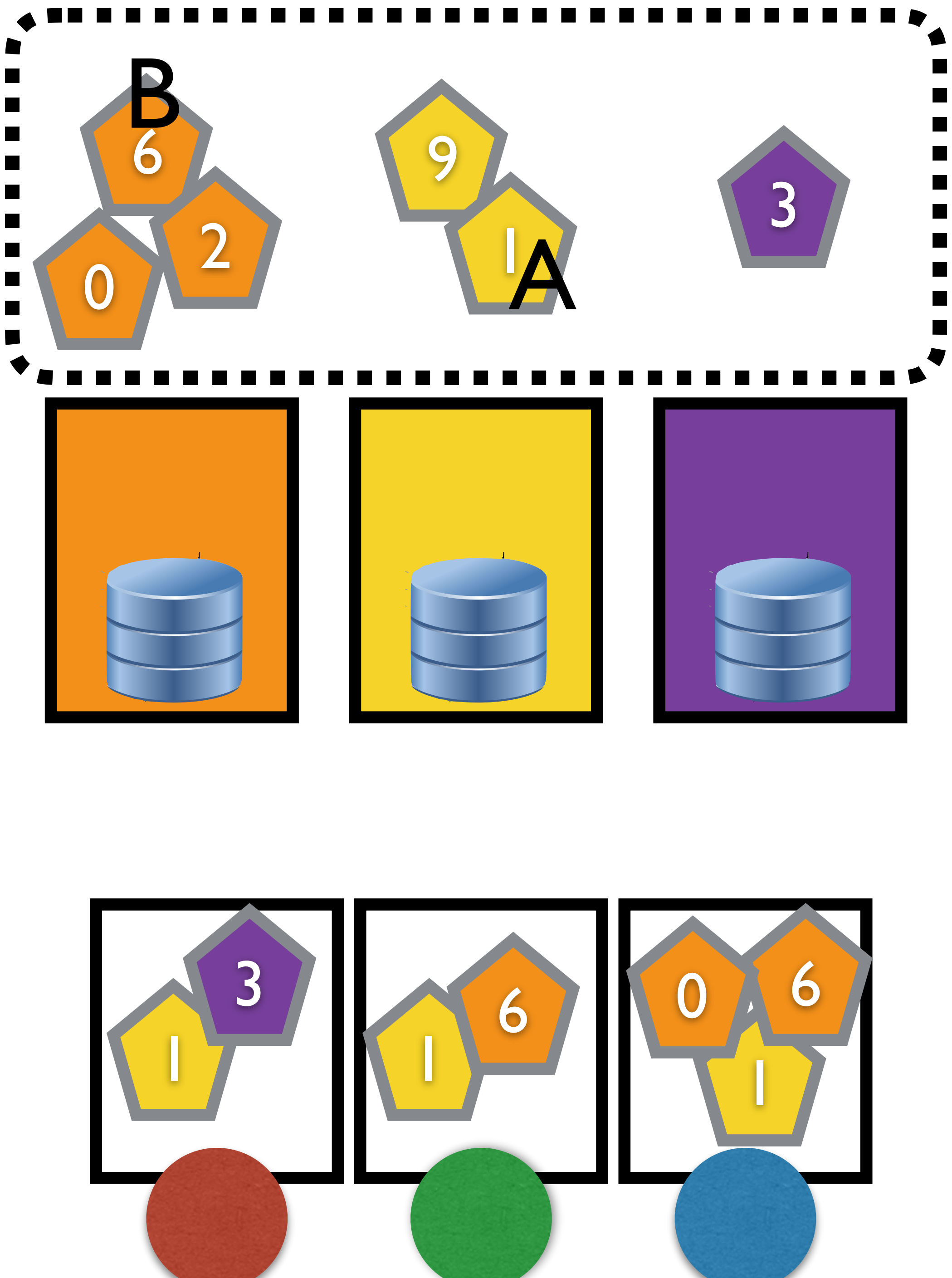
Thor Objects

Thor Servers

Thor $
Thor FE
Clients

- Universe of persistent Thor Objects
- Data/Object state partitioned across servers
- methods/operations run on clients (not RPC)
- clients read/write object state to from server DB's
- client cache for fast access

Client Caching and Trans



- Writes have to invalidate cached copies
- How to cope with reads of stale cached data?
- How to cope with read-modify-write races?
- Clients could lock before using each record
  - but that's slow -- probably need to contact server
  - wrecks the whole point of fast local caching in clients

Thor uses optimistic concurrency control (OCC)

- R/W local copies directly
  - Pretend sequential — no-concurrency — until commit
- On Commit:
  - send R/W info to server for "validation"
  - validation decides if it is really "OK" — serializable
    - YES: Send invalidates to cached copies
    - NO: Abort — discard changes

Thor uses optimistic concurrency control (OCC)

- Optimistic: Designed around the hope that no conflicts is common case
  - if true then Fast :-)
  - if false (validation can detect) but Slow :-(

Core of the paper is how to do Validation
and what we will focus on

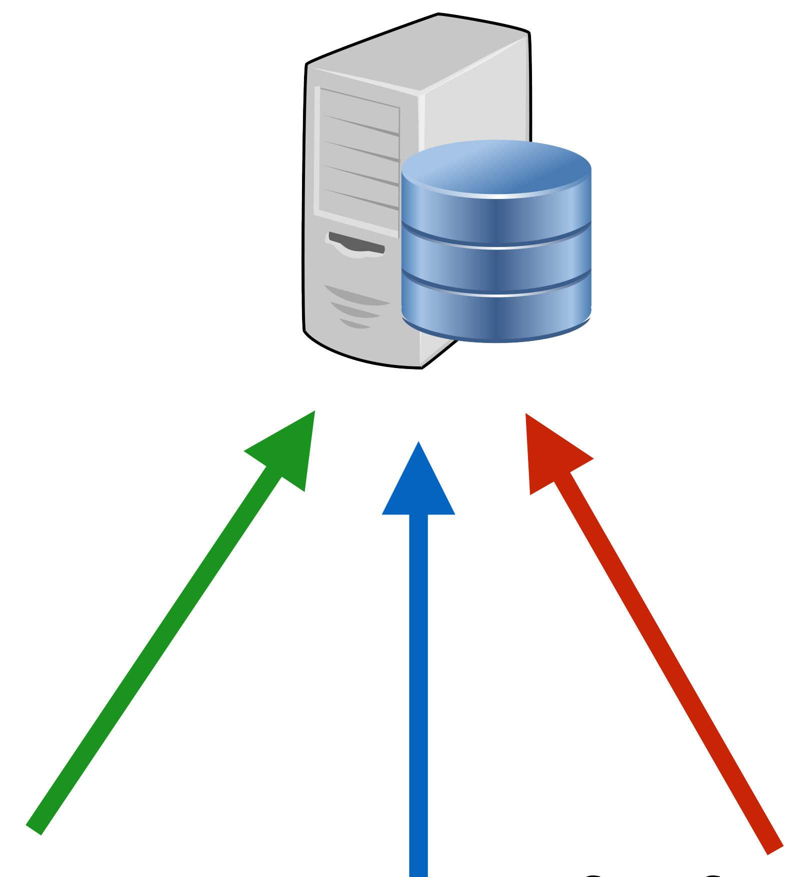
What should validation do?

- Look at what was Read and Written by the executing transactions and
  - decide if there is a serial order that would have gotten the same result as concurrent execution
- Many OCC validation algorithms

We will work our way to Thor's

Scheme 1 : Centralized

- Single Validation Server
- Clients send Read and Writes (including Values)
- Server must decide:
  - Would the results be "serializable" if we let the transactions commit?
  - Shuffle the transactions to see if we can find an order that ensures value READ are from most recent WRITES



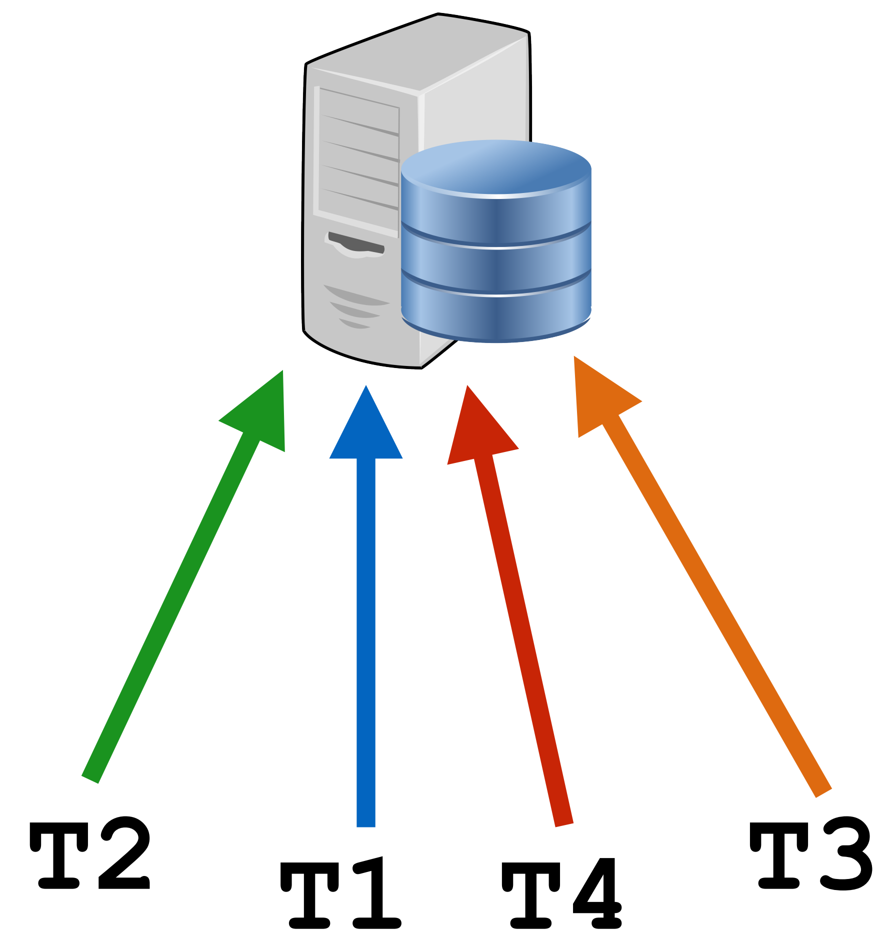
Scheme 1 : Centralized

Example 1

```
x=0   y=0   z=0

T1: Rx0 Wx1
T2: Rz0 Wz9
T3: Ry1 Rx1
T4: Rx0 Wy1
```

Validation needs to decide if this execution (reads, writes) is equivalent to some serial order

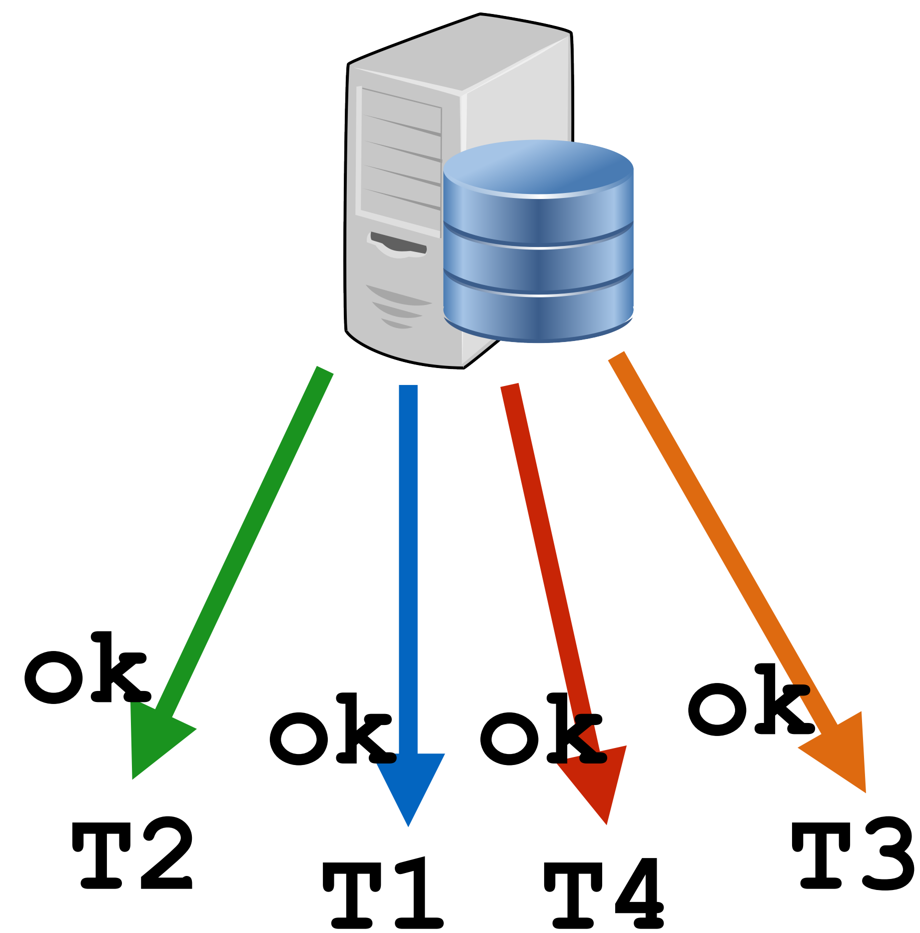


Scheme 1 : Centralized

Example 1

`x=0   y=0   z=0`

```
T1: Rx0 Wx1
T2: Rz0 Wz9
T3: Ry1 Rx1
T4: Rx0 Wy1
```



Validation needs to decide if this execution (reads, writes) is equivalent to some serial order

`YES: T4,T1,T3,T2 — Say yes to all Transactions (T2 can go anywhere)`

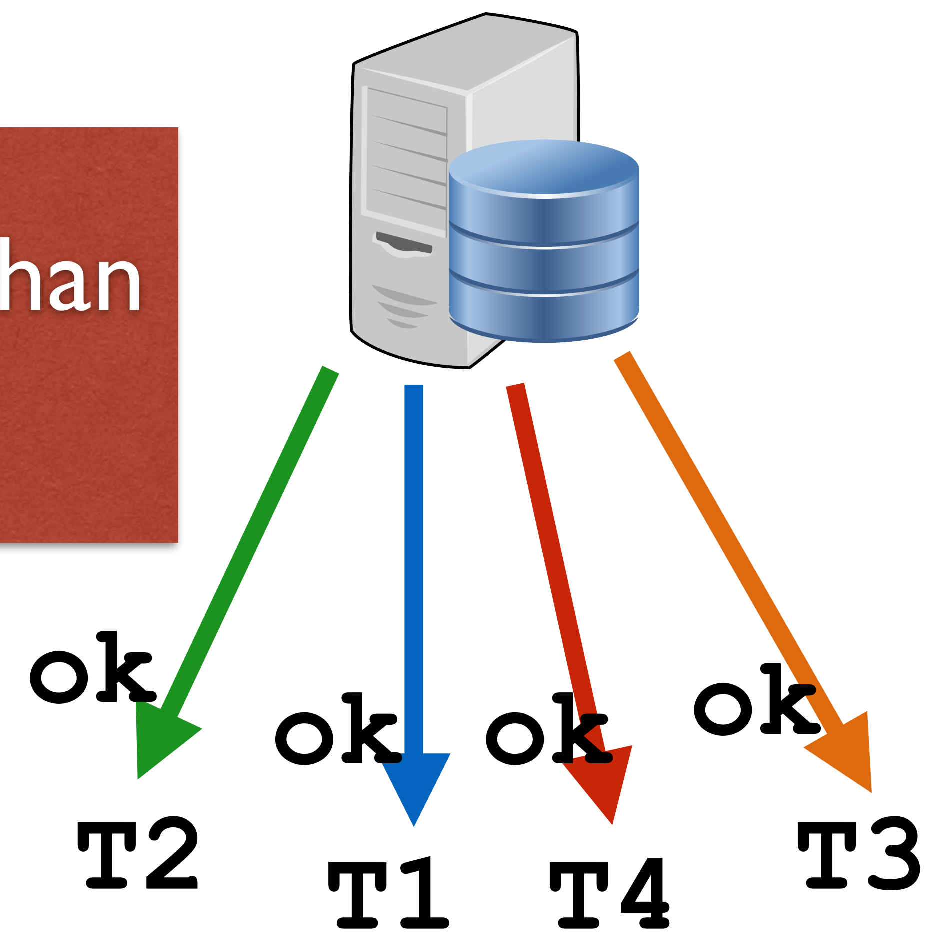
Scheme 1 : Centralized

Example 1

x=0   y=0   z=0

T1: Rx0 Wx1
T2: Rz0 Wz
T3: Ry1 Rx1
T4: Rx0 Wy1

More permissive than Thor



Validation needs to decide if this execution (reads, writes) is equivalent to some serial order

YES: T4,T1,T3,T2 – Say yes to all Transactions (T2 can go anywhere)