CS 839: Design the Next-Generation Database

Lecture 2: Transaction Basics

Xiangyao Yu

1/23/2020

Announcements

Course website

http://pages.cs.wisc.edu/~yxy/cs839-s20/index.html

Email me if you are not in HotCRP

https://wisc-cs839-ngdb20.hotcrp.com

Today’s Agenda

OLTP vs. OLAP

ACID properties

- Atomicity
- Consistency
- Isolation
- Durability

OLTP vs. OLAP

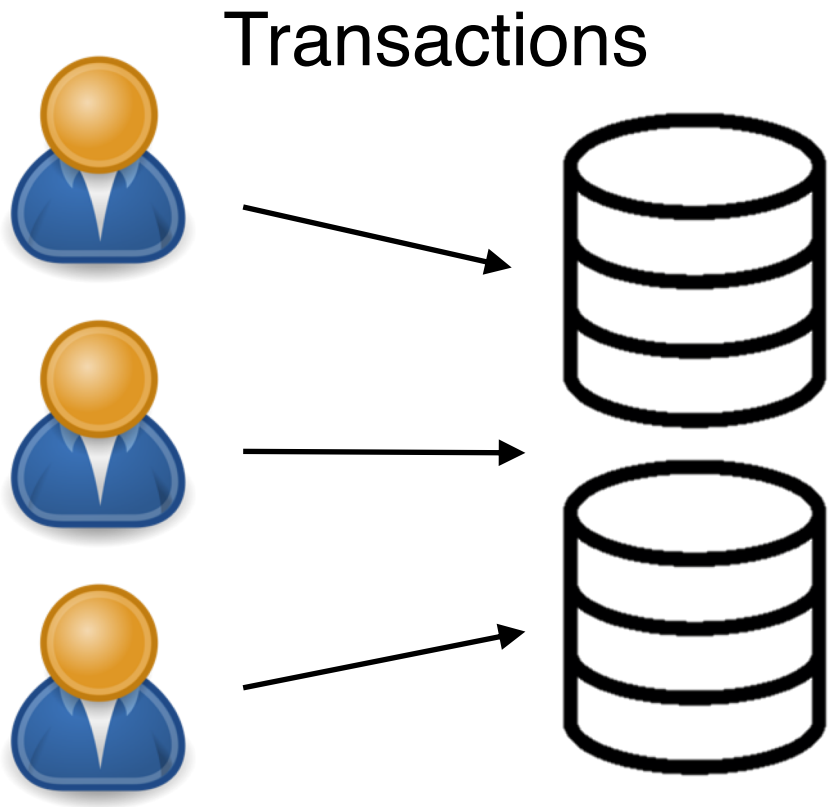
OLTP: On-Line Transaction Processing

- Users submit transactions that contain simple read/write operations
- Example: banking, online shopping, etc.

OLAP: On-Line Analytical Processing

- Complex analytics queries that reveal insights behind data
- Example: business report, marketing, forecasting, etc.

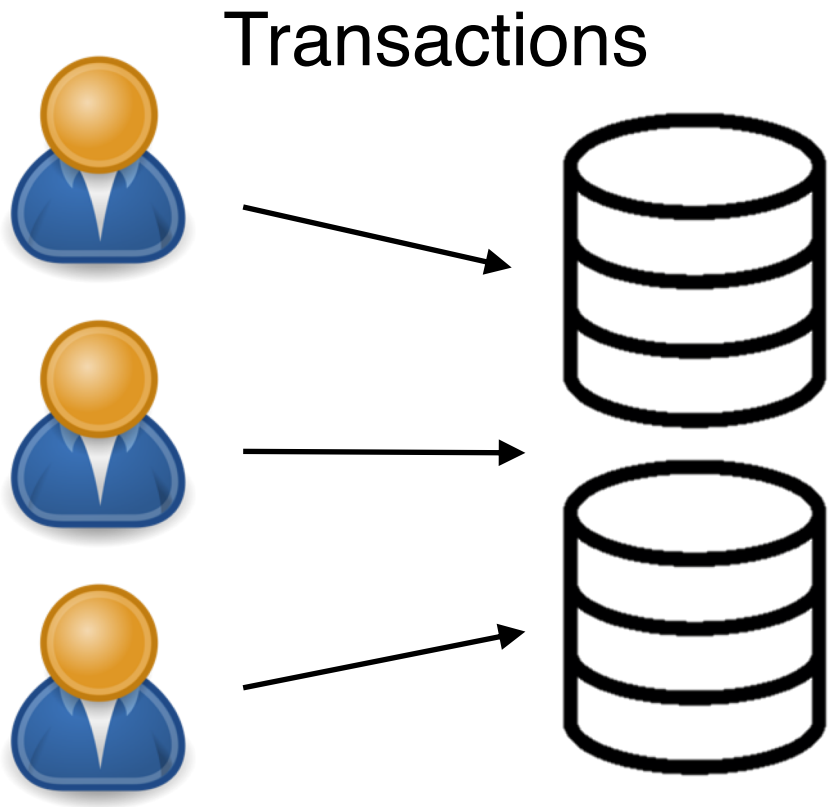
OLTP vs. OLAP



OLTP database

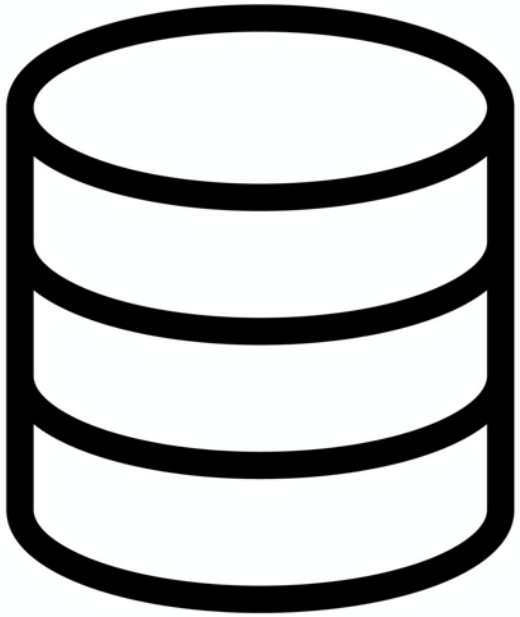
(Update Intensive)

OLTP vs. OLAP



OLTP database

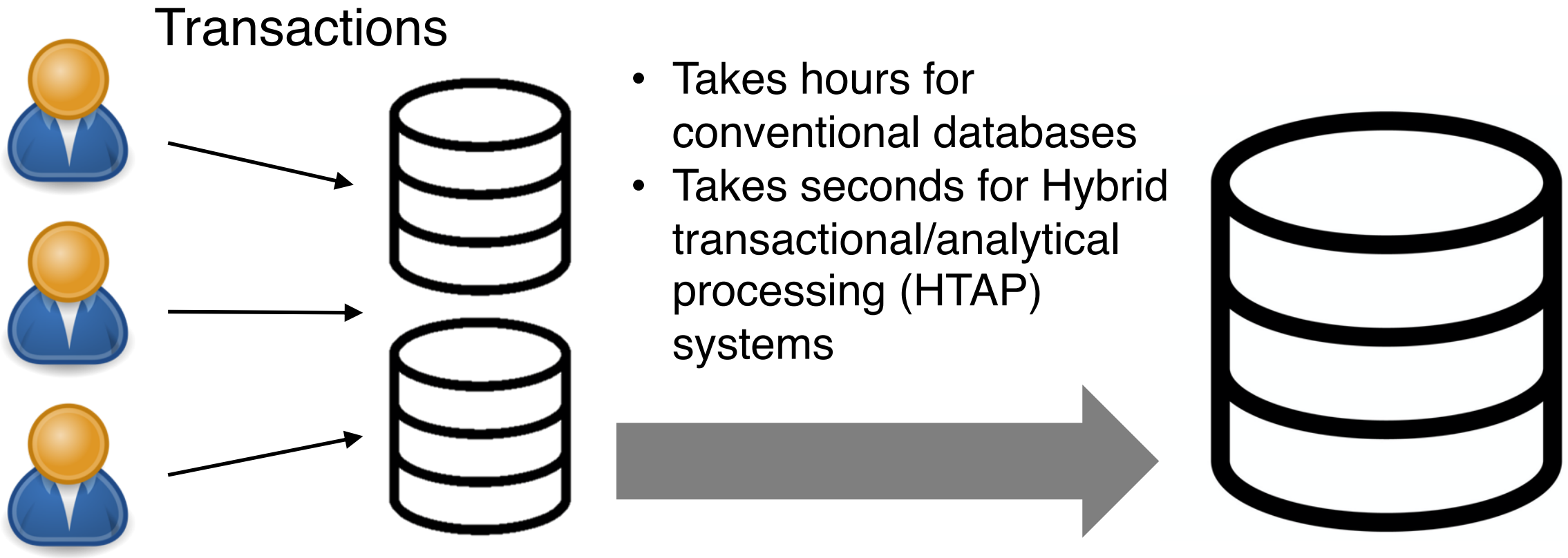
(Update Intensive)



OLAP database

(Read Intensive, rare updates)

OLTP vs. OLAP



- Takes hours for conventional databases
- Takes seconds for Hybrid transactional/analytical processing (HTAP) systems

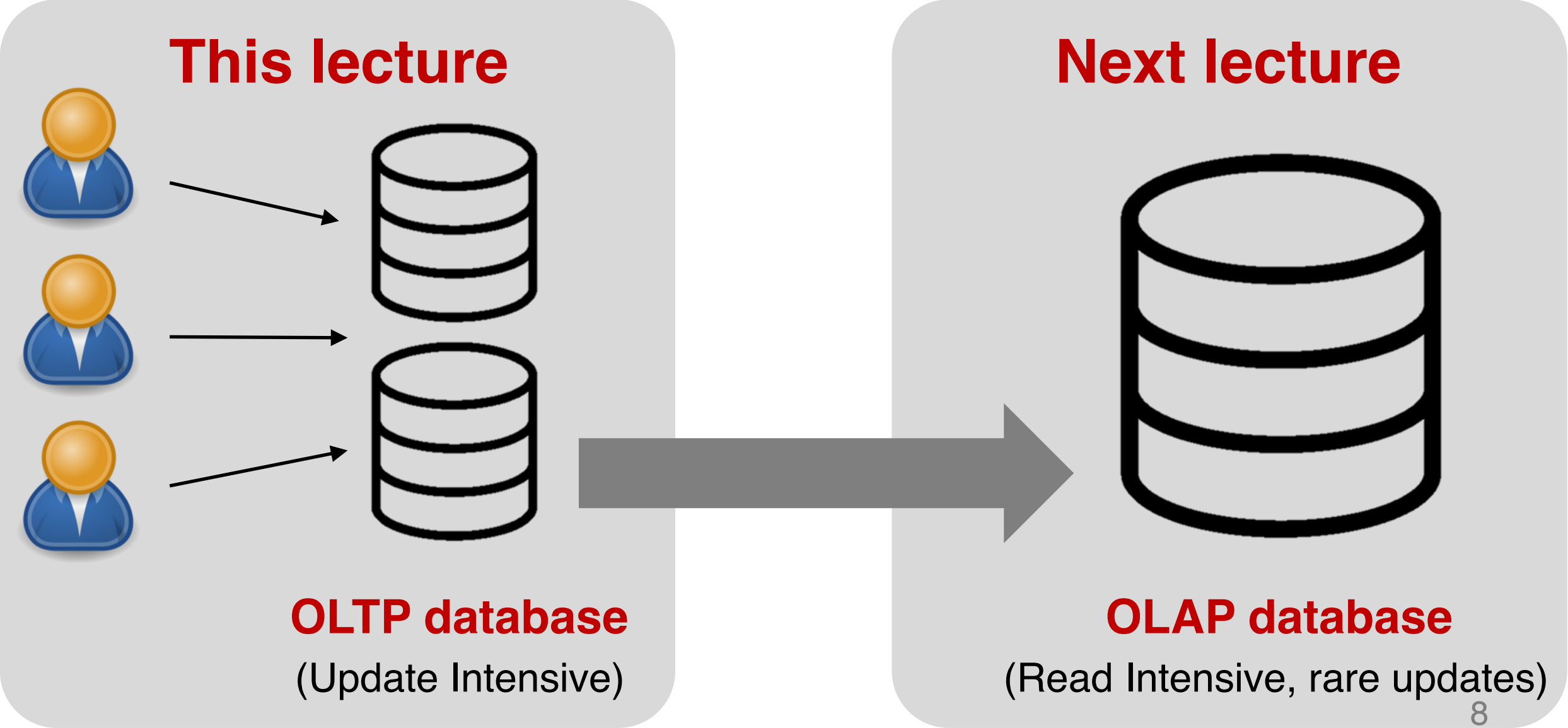
OLTP database

(Update Intensive)

OLAP database

(Read Intensive, rare updates)

OLTP vs. OLAP


This lecture
Next lecture
OLTP database
(Update Intensive)
OLAP database
(Read Intensive, rare updates)

Transaction Definition

trans·ac·tion

/tranˈzakSH(ə)n,tranˈsakSH(ə)n/

noun

noun: **transaction**; plural noun: **transactions**

an instance of buying or selling something; a business deal.
"in an ordinary commercial transaction a delivery date is essential"

Similar: deal | business | agreement | undertaking | affair

Transaction Definition

What are the required properties of a database transaction?

ACID

Transaction Definition

What are the required properties of a database transaction?

ACID

Example transactions:

Begin
Read(X)
Write(Y)
Insert(Z)
Commit

```
If checking.balance > 100
     checking.balance -= 100
     saving.balance += 100
```

ACID: Atomicity

```
If checking.balance > 100

     checking.balance -= 100

     saving.balance += 100
```

Atomicity: Either all operations occur, or nothing occurs (All or nothing)

ACID: Consistency

```
If checking.balance > 100
     checking.balance -= 100
     saving.balance += 100
```

Checking	$50	RunTxn()	-$50
Saving	$1000		$1100

Consistency: Integrity constraints must be maintained.

Example Integrity constraint:

balance of checking account must be above $0

ACID: Durability

Example transaction:

```
If checking.balance > 100
     checking.balance -= 100
     saving.balance += 100
```

Checking	$1000
Saving	$1000

ACID: Durability

Example transaction:

```
If checking.balance > 100
     checking.balance -= 100
     saving.balance += 100
```

Checking	$1000	RunTxn()	$900
Saving	$1000		$1100

ACID: Durability

Example transaction:

```
If checking.balance > 100
    checking.balance -= 100
    saving.balance += 100
```

Checking	$1000	RunTxn()	$900		???
Saving	$1000		$1100		???

CRASH !!

ACID: Durability

Example transaction:

```
If checking.balance > 100
     checking.balance -= 100
     saving.balance += 100
```

Checking	$1000	RunTxn()	$900		???
Saving	$1000		$1100		???

CRASH !!

Durability: A transaction's updates persist in case of system failure

ACID: Isolation

Initailly
checking.balance = 1000

ACID: Isolation

Initailly
checking.balance = 1000

```
If checking.balance > 100

    bal = checking.balance

    bal = bal – 100

    checking.balance = bal
```

```
If checking.balance > 100

    bal = checking.balance

    bal = bal – 100

    checking.balance = bal
```

ACID: Isolation

Initailly
checking.balance = 1000

```
If checking.balance > 100
```

```
bal = checking.balance
```

1

bal = 1000

```
bal = bal – 100

checking.balance = bal
```

```
If checking.balance > 100

bal = checking.balance

bal = bal – 100

checking.balance = bal
```

ACID: Isolation

Initailly
checking.balance = 1000

```
If checking.balance > 100
```

```
bal = checking.balance
```

1

bal = 1000

```
bal = bal – 100

checking.balance = bal
```

```
If checking.balance > 100
```

```
bal = checking.balance
```

2

bal = 1000

```
bal = bal – 100

checking.balance = bal
```

ACID: Isolation

Initailly
checking.balance = 1000

```
If checking.balance > 100
```

```
bal = checking.balance
```

1

bal = 1000

```
bal = bal – 100
```

bal = 900

```
checking.balance = bal
```

```
If checking.balance > 100
```

```
bal = checking.balance
```

2

bal = 1000

```
bal = bal – 100
```

bal = 900

```
checking.balance = bal
```

ACID: Isolation

Initailly
checking.balance = 1000

```
If checking.balance > 100
```

```
bal = checking.balance
```

1

bal = 1000

```
bal = bal – 100
```

bal = 900

```
checking.balance = bal
```

3

checking = 900

```
If checking.balance > 100
```

```
bal = checking.balance
```

2

bal = 1000

```
bal = bal – 100
```

bal = 900

```
checking.balance = bal
```

ACID: Isolation

Initailly
checking.balance = 1000

```
If checking.balance > 100
```

```
bal = checking.balance
```

1

bal = 1000

```
bal = bal – 100
```

bal = 900

```
checking.balance = bal
```

3

checking = 900

```
If checking.balance > 100
```

```
bal = checking.balance
```

2

bal = 1000

```
bal = bal – 100
```

bal = 900

```
checking.balance = bal
```

4

checking = 900

ACID: Isolation

Strong isolation: **Serializability (Focus of this course)**

Schedule of concurrent transactions is equivalent to some serial schedule

Weak isolation: **Snapshot Isolation, Read Committed, Read Uncommitted, etc.**

Weaker isolation levels allow more interleaving of transactions

ACID: Isolation – Why Strong Isolation?

MongoDB & Bitcoin: How NoSQL design flaws brought down two exchanges

DZone April 2014

Attackers stole **896 Bitcoins ≈ 3 million US dollars**

Why you should pick strong consistency, whenever possible

Google Cloud January 2018

> *Systems that don't provide strong consistency …*
> ***create a burden for application developers***

SQL (before 2000) -> NoSQL (since 2000) -> NewSQL (since 2010s)

How to Enforce ACID

Atomic & Isolation: Concurrency control

Consistency: Check integrity for transactions

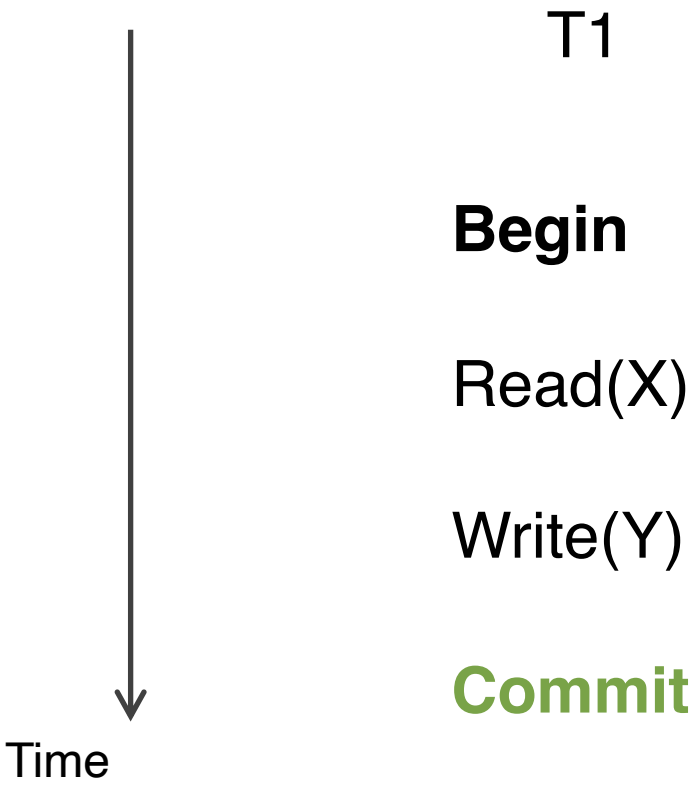
Durability: Logging

Concurrency Control

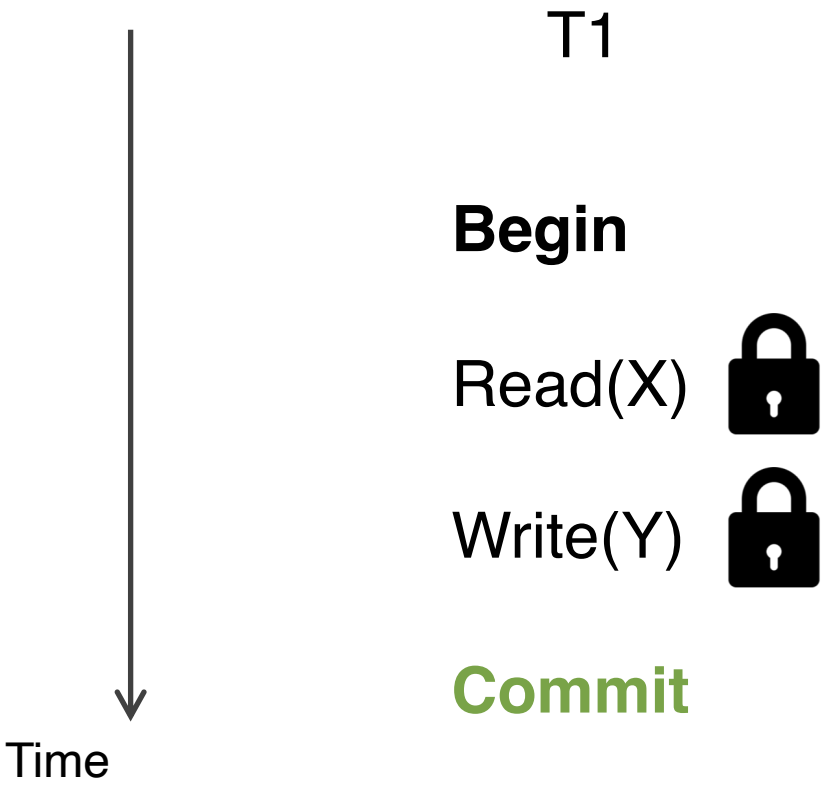
Pessimistic

Optimistic

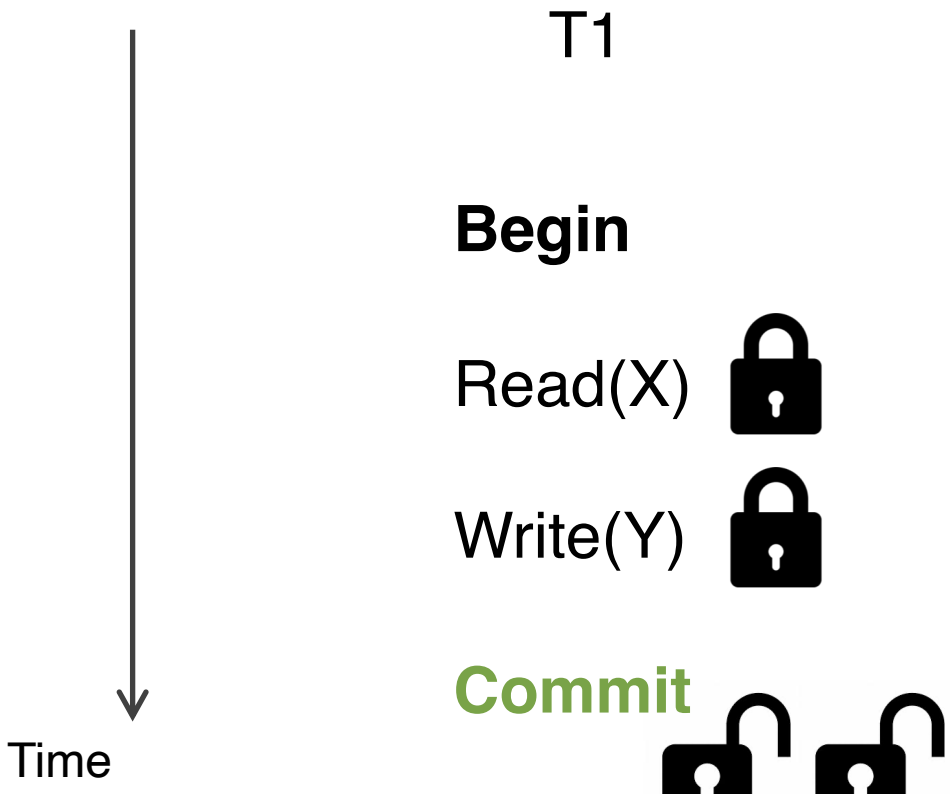
Pessimistic – Two Phase Locking (2PL)



Pessimistic – Two Phase Locking (2PL)



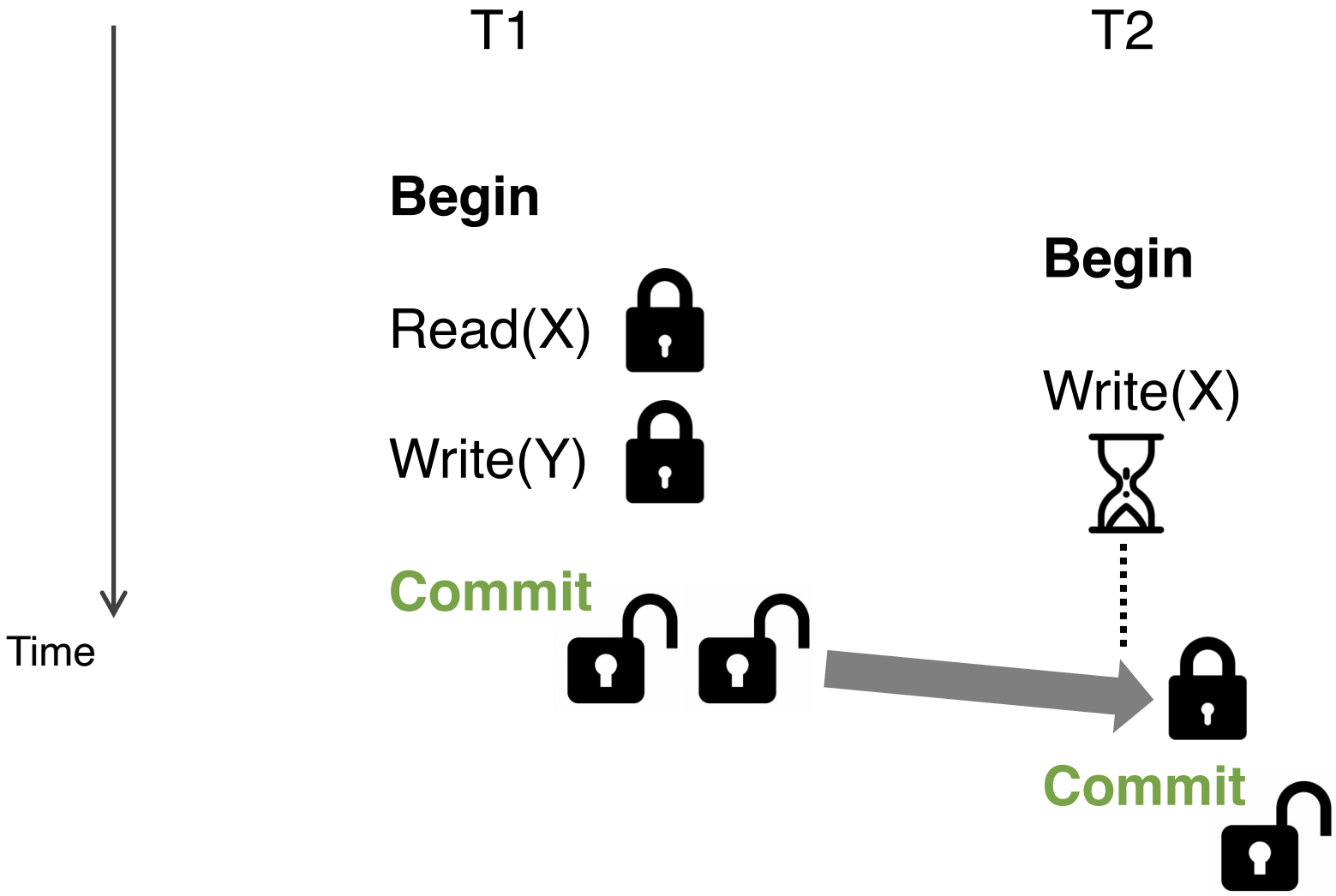
Pessimistic – Two Phase Locking (2PL)



Pessimistic – Two Phase Locking (2PL)

Pessimistic – Two Phase Locking (2PL)

Pessimistic – Two Phase Locking (2PL)



Pessimistic – Deadlock in 2PL

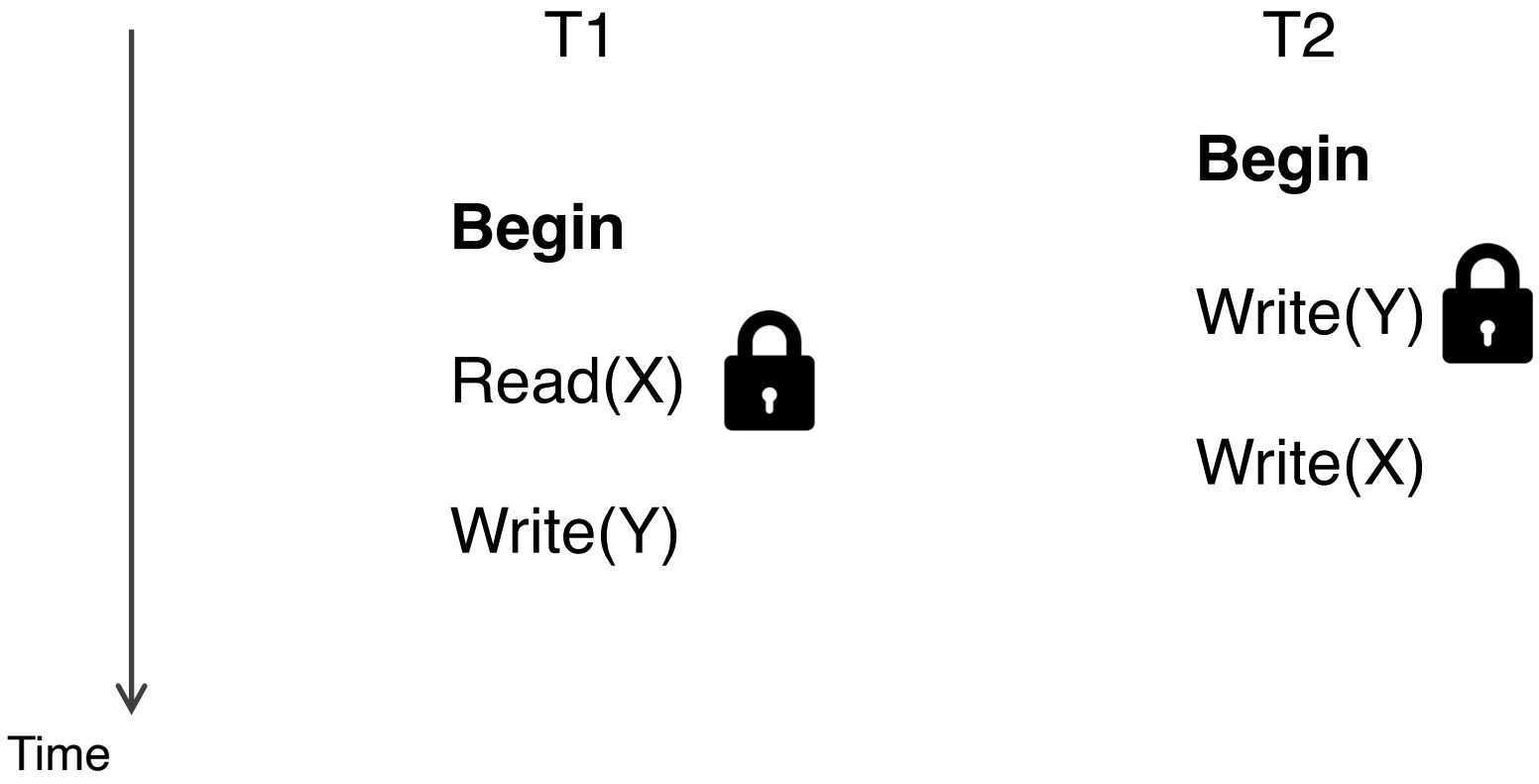
T1	T2
Begin	**Begin**
Read(X)	Write(Y)
Write(Y)	Write(X)

Pessimistic – Deadlock in 2PL

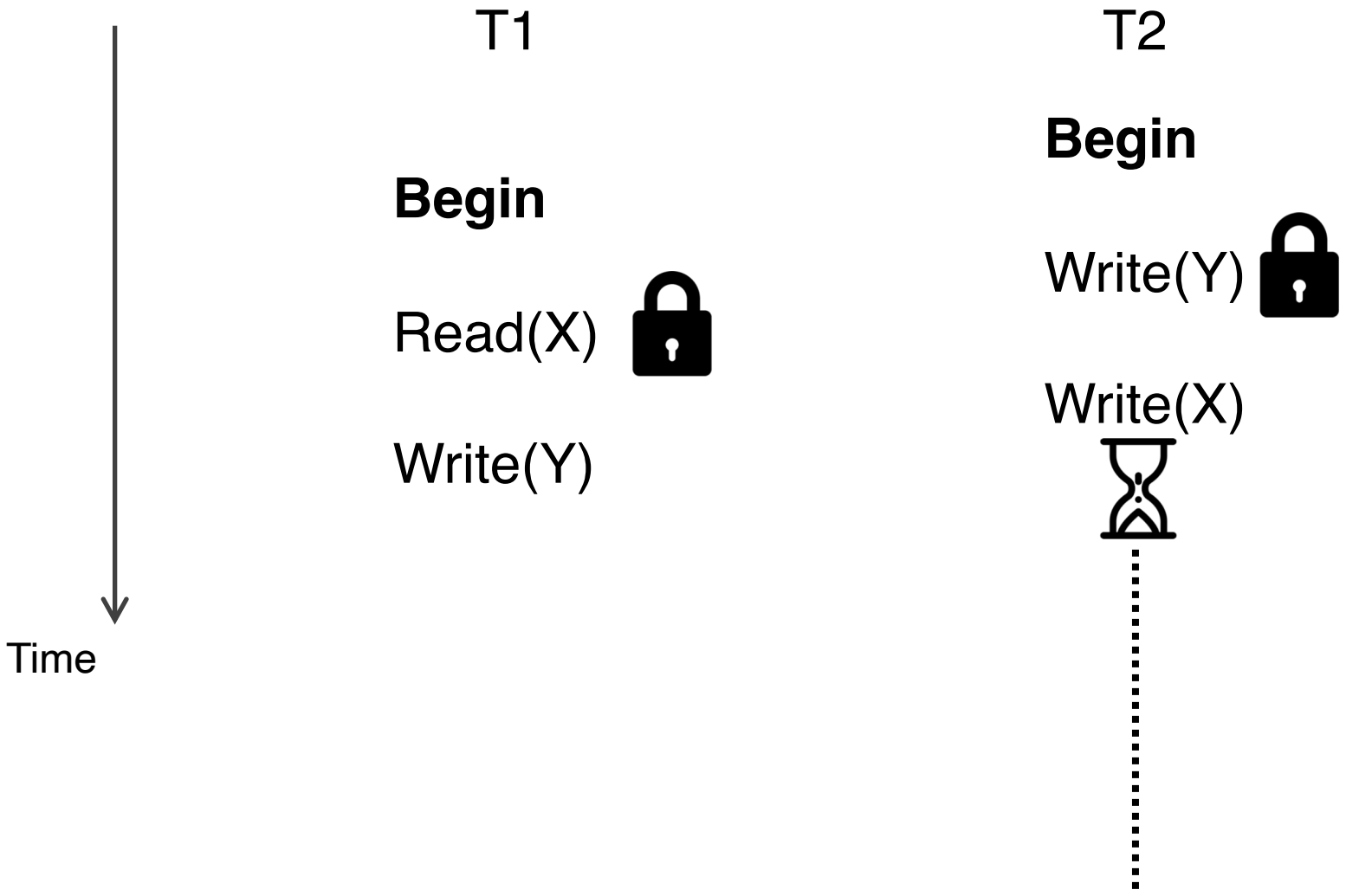
Time

T1	T2
	Begin
Begin	
	Write(Y) 🔒
Read(X)	
	Write(X)
Write(Y)	

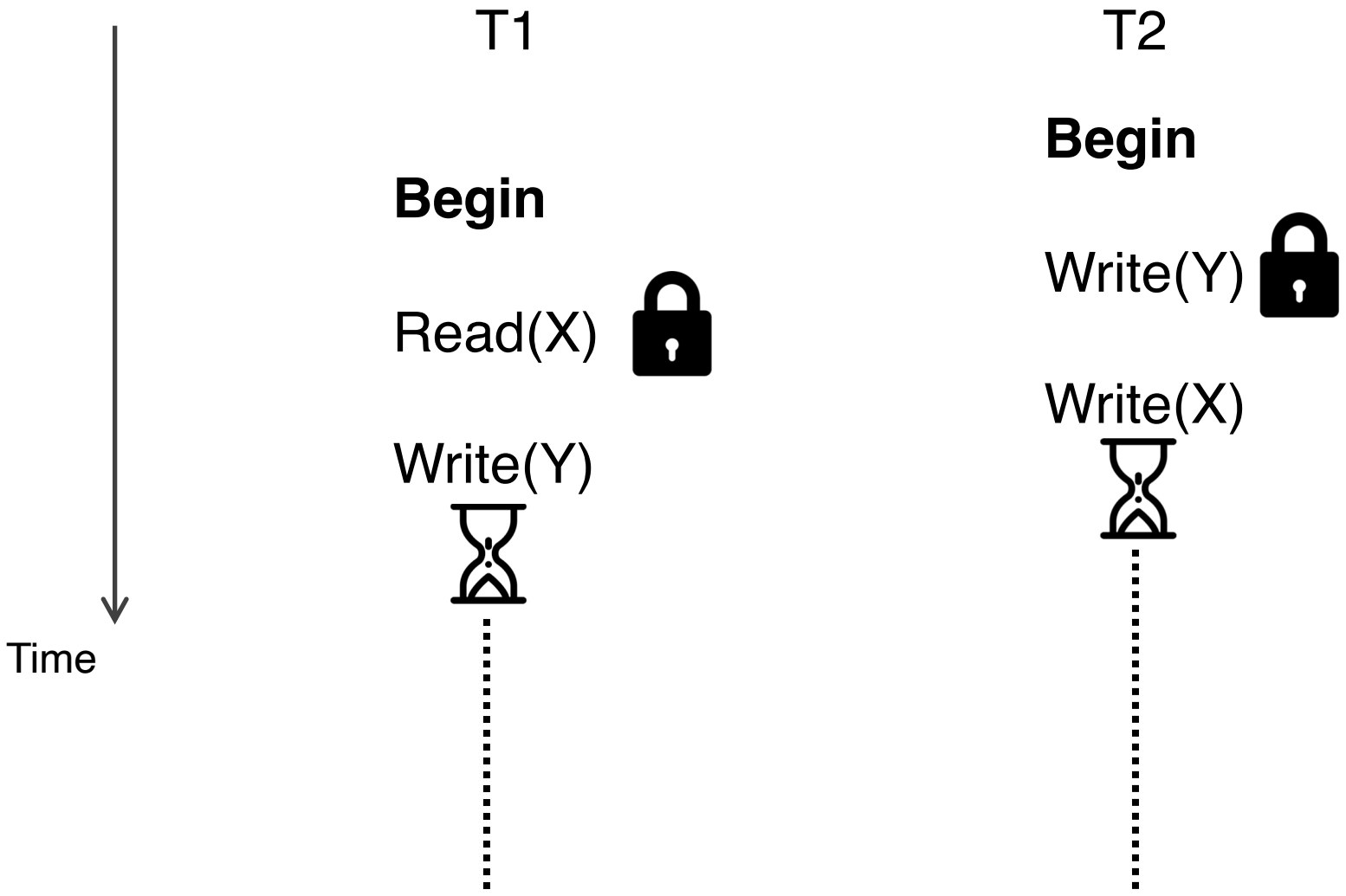
Pessimistic – Deadlock in 2PL



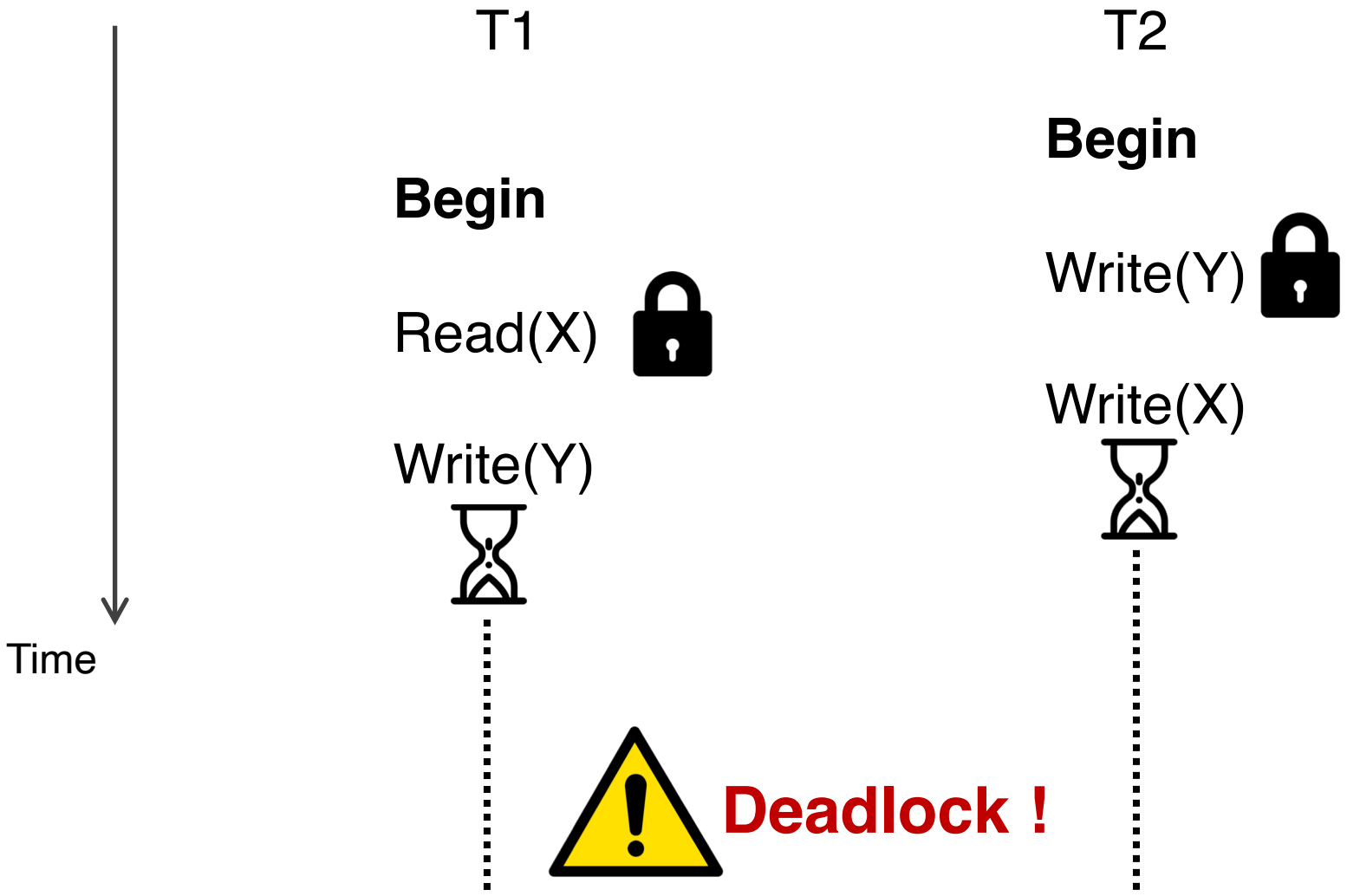
Pessimistic – Deadlock in 2PL



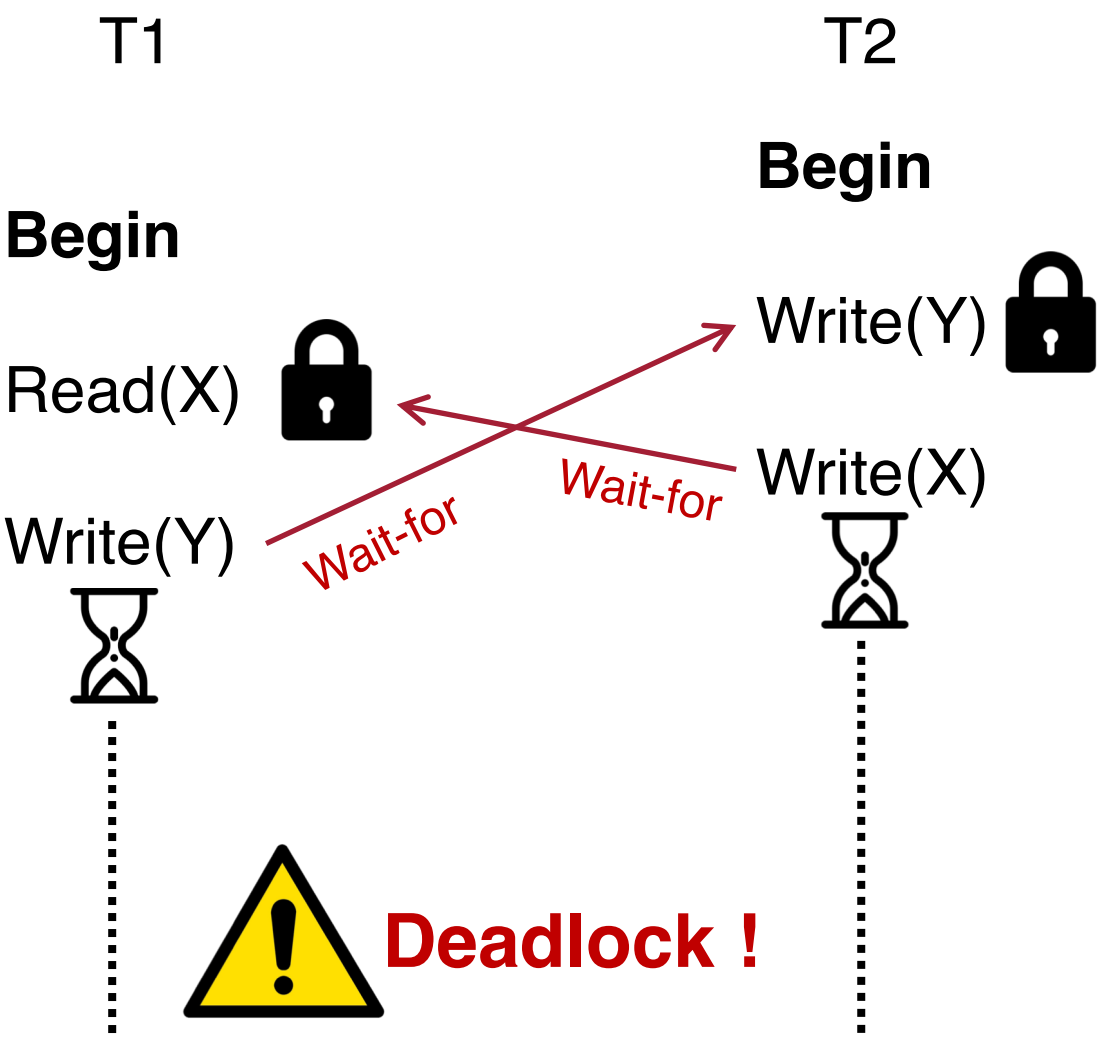
Pessimistic – Deadlock in 2PL



Pessimistic – Deadlock in 2PL



Pessimistic – Deadlock Resolution



Detect and break cycles

Allow only certain waits

- NoWait
- WaitDie
- WoundWait
- LimitedDepth
- etc.

Optimistic Concurrency Control (OCC)

T1	T2
Begin	
	Begin
Read(X)	
	Write(X)
Write(Y)	
	Commit
Validate(X)	
Abort	

How to Enforce ACID

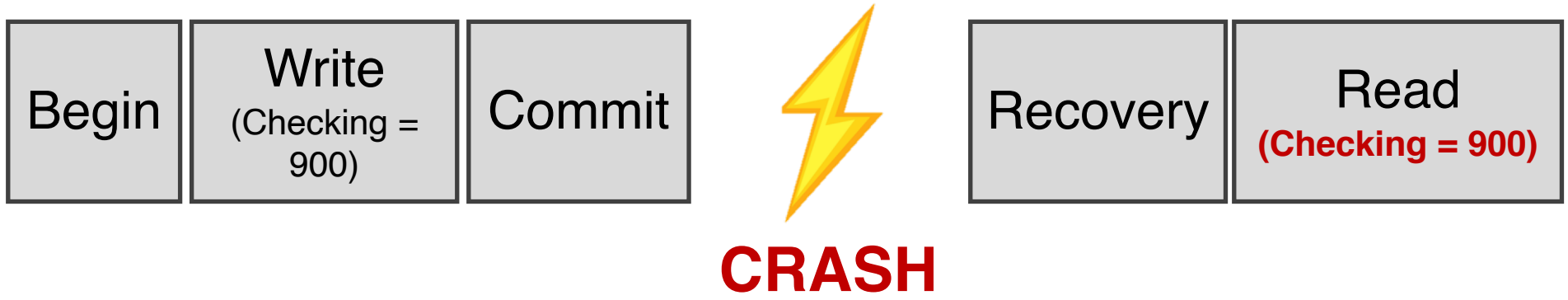
Atomic & Isolation: Concurrency control

Consistency: Check integrity for transactions

Durability: Logging

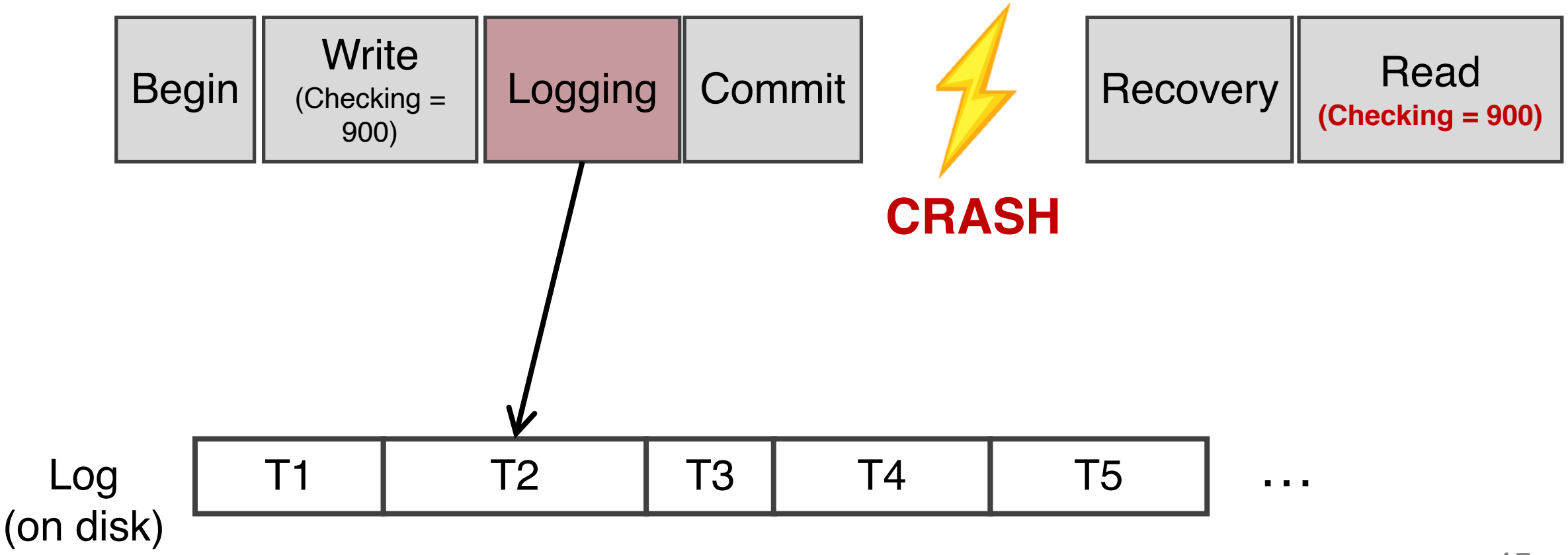
Durability: Logging

Initially
checking = 1000



Durability: Logging

Initially
checking = 1000



Summary

A transaction in an OLTP system has **ACID** properties

Atomicity, **C**onsistency, **I**solation, **D**urability

Concurrency control (enforces **Atomicity** and **Isolation**)

Two Phase Locking (2PL)

Optimistic Concurrency Control (OCC)

Logging (enforces **Durability**)

Group Discussion

Discuss the relative advantages and disadvantages of 2PL and OCC.

What is your opinion on the debate between SQL and NoSQL (strong guarantee vs. high performance)?

Do you see any scalability problem with logging? If so, any potential solution?

Before Next Lecture

Submit discussion summary to https://wisc-cs839-ngdb20.hotcrp.com

- One summary per group
- Authors: group members
- Any format is ok (e.g., pdf, doc, txt)
- Feel free to comment on others' discussion

Submit review for

[optional] C-Store: A Column-oriented DBMS