CS 564: Database Management Systems

Lecture 31: Concurrency control

Xiangyao Yu

4/10/2024

Module B4 Transactions

Concurrency control

Optimistic concurrency control

Logging

ARIES recovery

Outline

Transaction basics

ACID properties

Two-phase locking (2PL)

Outline

Transaction basics

ACID properties

Two-phase locking (2PL)

Basic Concept: **Transaction**

Transaction: A sequence of many actions considered to be one atomic unit of work

- Action: read, scan, update, insert, delete, etc.
- Special action: **begin**, **commit**, **abort**
- Central building block of a database

Basic Concept: **Transaction**

Transaction: A sequence of many actions considered to be one atomic unit of work

- Action: read, scan, update, insert, delete, etc.
- Special action: **begin**, **commit**, **abort**
- Central building block of a database

```
BEGIN TRANSACTION ;
      UPDATE account
        SET balance = balance – 1000
        WHERE account_no = 1;
      UPDATE account
        SET balance = balance + 1000
        WHERE account_no = 2;
COMMIT ;
```

Use cases

- Transfer money between accounts
- Purchase a group of products online
- Book flight tickets

Outline

Transaction basics

ACID properties

- Atomicity
- Consistency
- Isolation (serializability)
- Durability

Two-phase locking (2PL)

ACID Properties

ACID properties

- **Atomicity**: all actions in the transaction happen, or none happen

ACID Properties

ACID properties

- **Atomicity**: all actions in the transaction happen, or none happen
- **Consistency**: a database in a consistent state will remain in a consistent state after the transaction

ACID Properties

ACID properties

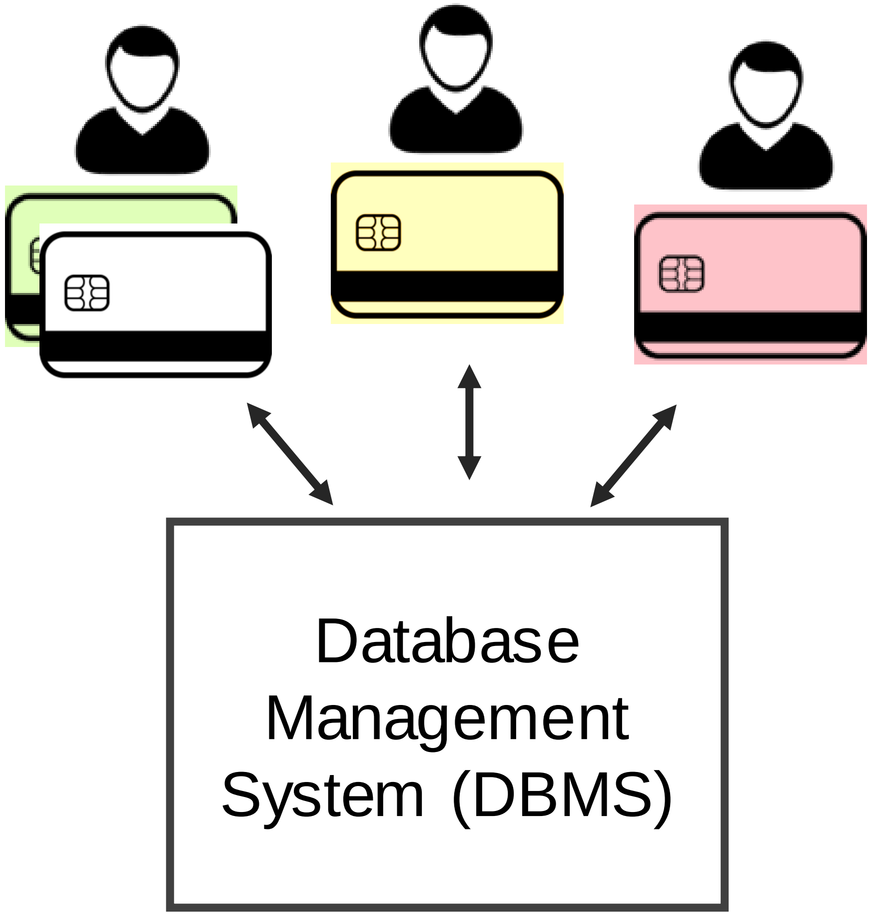
- **A**tomicity: all actions in the transaction happen, or none happen
- **C**onsistency: a database in a consistent state will remain in a consistent state after the transaction
- **I**solation: the execution of one transaction is isolated from other (possibly interleaved) transactions

ACID Properties

ACID properties

- **Atomicity**: all actions in the transaction happen, or none happen
- **Consistency**: a database in a consistent state will remain in a consistent state after the transaction
- **Isolation**: the execution of one transaction is isolated from other (possibly interleaved) transactions
- **Durability**: once a transaction commits, its effects must persist

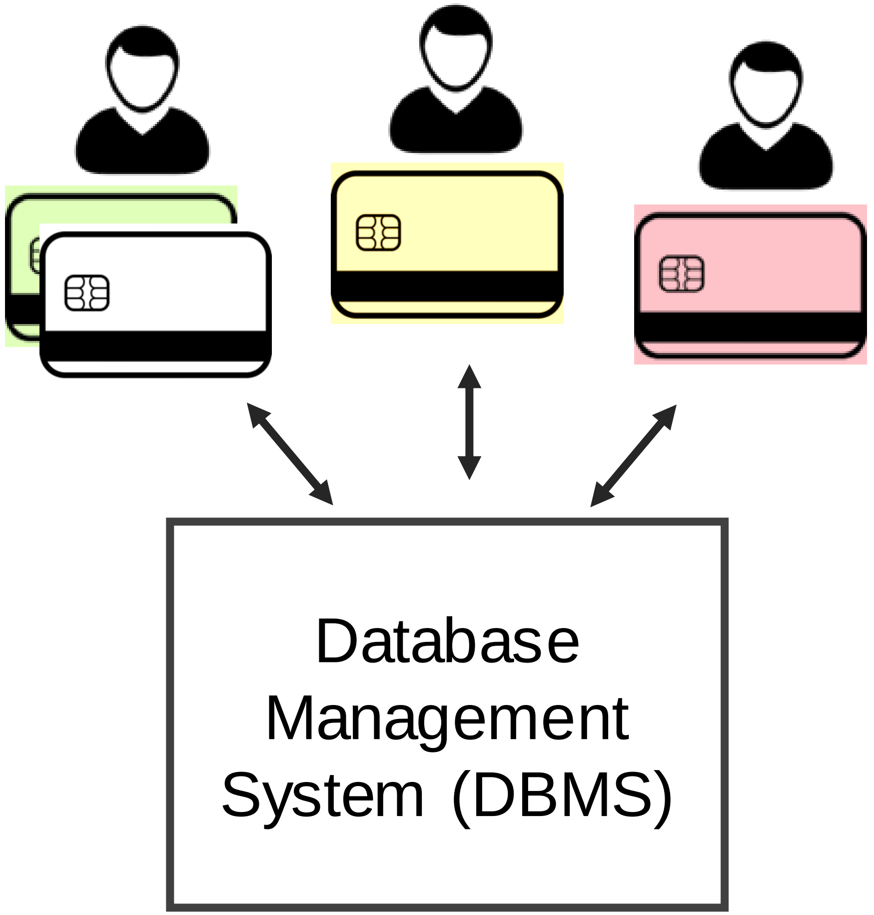
A Naïve Implementation of Transactions



NaïveDB

- Admit one transaction T
- Execute T to completion
- Flush T's writes to disk
- Commit T

A Naïve Implementation of Transactions

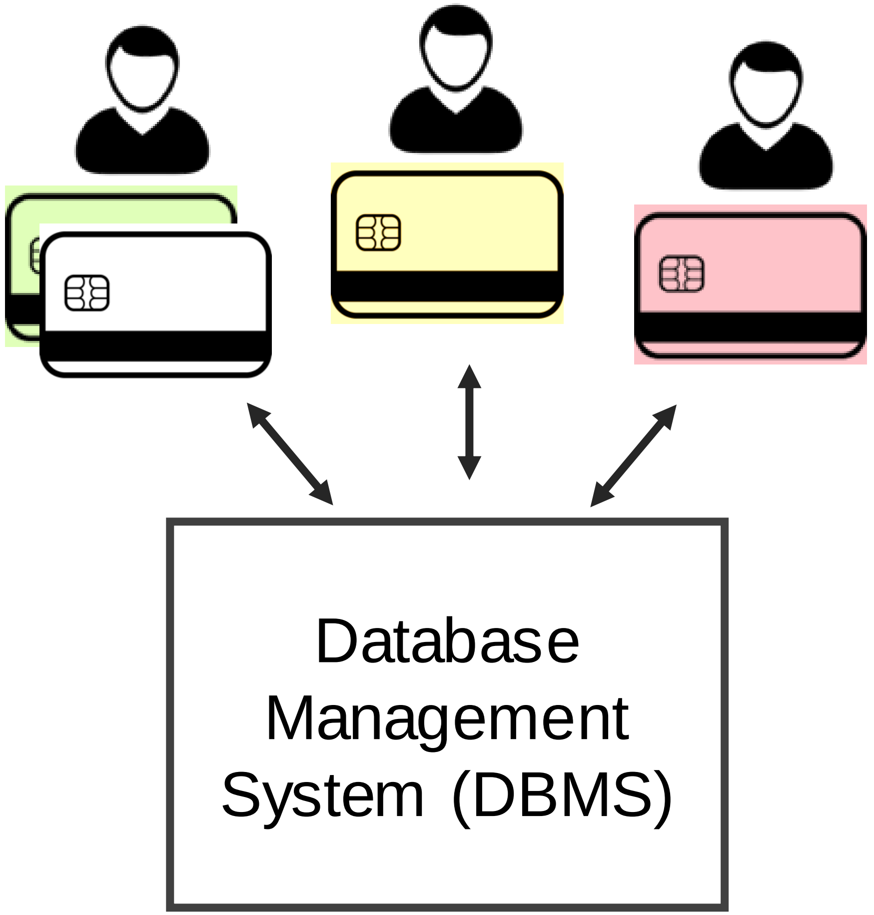


NaïveDB

- Admit one transaction T
- Execute T to completion
- Flush T's writes to disk
- Commit T

Naïve DB satisfies **Atomicity**, **Serializable Isolation**, and **Durability**

A Naïve Implementation of Transactions



NaïveDB

- Admit one transaction T
- Execute T to completion
- Flush T's writes to disk
- Commit T

Naïve DB satisfies **Atomicity**, **Serializable Isolation**, and **Durability**

Disadvantage of NaiveDB?

Performance!

Online Transaction Processing (OLTP)

NaïveDB
- Admit one transaction T
- Execute T to completion
- Flush T's writes to disk
- Commit T

Performance goal of OLTP DBMS: **Logically equivalent to NaïveDB** and yet achieve **better performance**

Isolation Level

We focus on strong isolation level (i.e., **serializability**)

Isolation Level

We focus on strong isolation level (i.e., **serializability**)

Serializability: Execution of transactions produces the same results as some serial execution

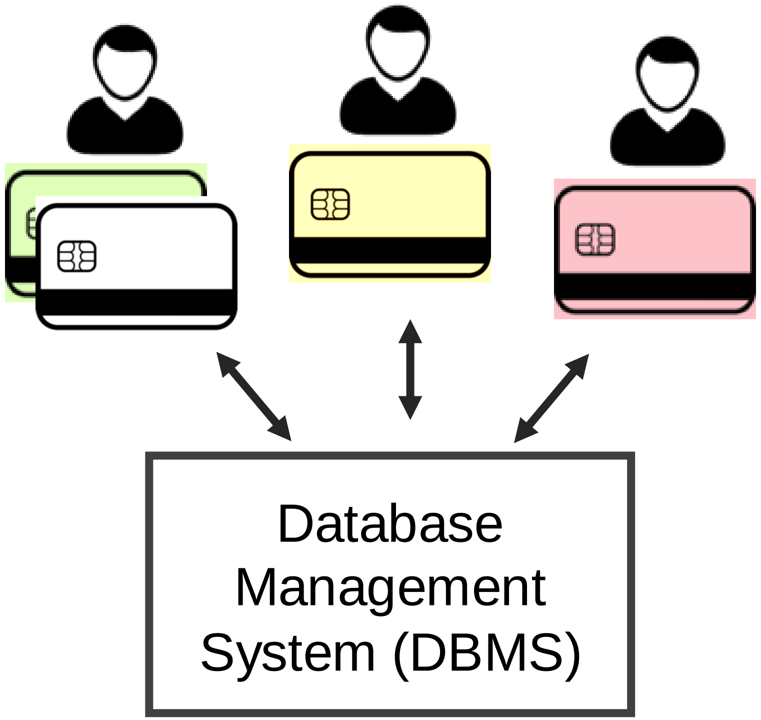
- Equivalent to NaiveDB

Isolation Level

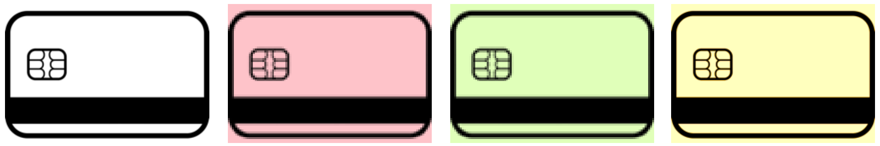
We focus on strong isolation level (i.e., **serializability**)

Serializability: Execution of transactions produces the same results as some serial execution

– Equivalent to NaiveDB



Serializable execution



ACID – Isolation Example

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 Example

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 Example

Initailly
checking.balance = 1000

```
If checking.balance > 100
    bal = checking.balance
    bal = bal - 100
    checking.balance = bal
```

1

bal = 1000

```
If checking.balance > 100
    bal = checking.balance
    bal = bal - 100
    checking.balance = bal
```

2

bal = 1000

ACID – Isolation Example

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 Example

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 Example

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 Example

Not Serializable!

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

Serializability – Example

Is the following execution serializable?

T1.read(X) T2.write(X) T2.read(Y) T1.read(Y) C1 C2

Serializability – Example

Is the following execution serializable?

T1.read(X) T2.write(X) T2.read(Y) T1.read(Y) C1 C2

T1.write(X) T2.read(X) C1 C2

Serializability – Example

Is the following execution serializable?

T1.read(X) T2.write(X) T2.read(Y) T1.read(Y) C1 C2

T1.write(X) T2.read(X) C1 C2

T1.read(X) T2.read(Y) T1.write(Y) T2.write(X) C1 C2

Serializability – Example

Is the following execution serializable?

T1.read(X) T2.write(X) T2.read(Y) T1.read(Y) C1 C2

T1.write(X) T2.read(X) C1 C2

T1.read(X) T2.read(Y) T1.write(Y) T2.write(X) C1 C2

T1.write(X) T2.read(X) A1 C2

Serializability – Example

Is the following execution serializable?

T1.read(X) T2.write(X) T2.read(Y) T1.read(Y) C1 C2

T1.write(X) T2.read(X) C1 C2

T1.read(X) T2.read(Y) T1.write(Y) T2.write(X) C1 C2

T1.write(X) T2.read(X) A1 C2

T1.read(X) T2.read(X) T2.write(X) T1.write(X) C1 C2

Serializability – Example

Is the following execution serializable?

T1.read(X) T2.write(X) T2.read(Y) T1.read(Y) C1 C2 ✓

T1.write(X) T2.read(X) C1 C2 ✓

T1.read(X) T2.read(Y) T1.write(Y) T2.write(X) C1 C2 ✗

T1.write(X) T2.read(X) A1 C2 ✗

T1.read(X) T2.read(X) T2.write(X) T1.write(X) C1 C2 ✗

Outline

Transaction basics

ACID properties

Two-phase locking (2PL)

- Shared and exclusive lock
- Strict 2PL
- Deadlock

Two-Phase Locking (2PL)

NaiveDB is equivalent to exclusively locking the entire database for each transaction

Two-Phase Locking (2PL)

NaiveDB is equivalent to exclusively locking the entire database for each transaction

Use **fine-grained locks** (e.g., page granularity) to improve concurrency

- A lock is in either **Shared** (for read) or **Exclusive** (for write) mode

Two-Phase Locking (2PL)

NaiveDB is equivalent to exclusively locking the entire database for each transaction

Use **fine-grained locks** (e.g., page granularity) to improve concurrency

– A lock is in either **Shared** (for read) or **Exclusive** (for write) mode

Transaction T1	Transaction T2
SH-lock(A) Unlock(A)	
	EX-lock(A) Unlock(A)
SH-lock(A) Unlock(A)	

Locking by itself does not guarantee serializability

Two-Phase Locking (2PL)

NaiveDB is equivalent to exclusively locking the entire database for each transaction

Use **fine-grained locks** (e.g., page granularity) to improve concurrency

Two-phase locking (2PL) ensures serializability

- **Growing phase**: acquire but do not release locks
- **Shrinking phases**: release but do not acquire locks

Two-Phase Locking (2PL)

NaiveDB is equivalent to exclusively locking the entire database for each transaction

Use **fine-grained locks** (e.g., page granularity) to improve concurrency

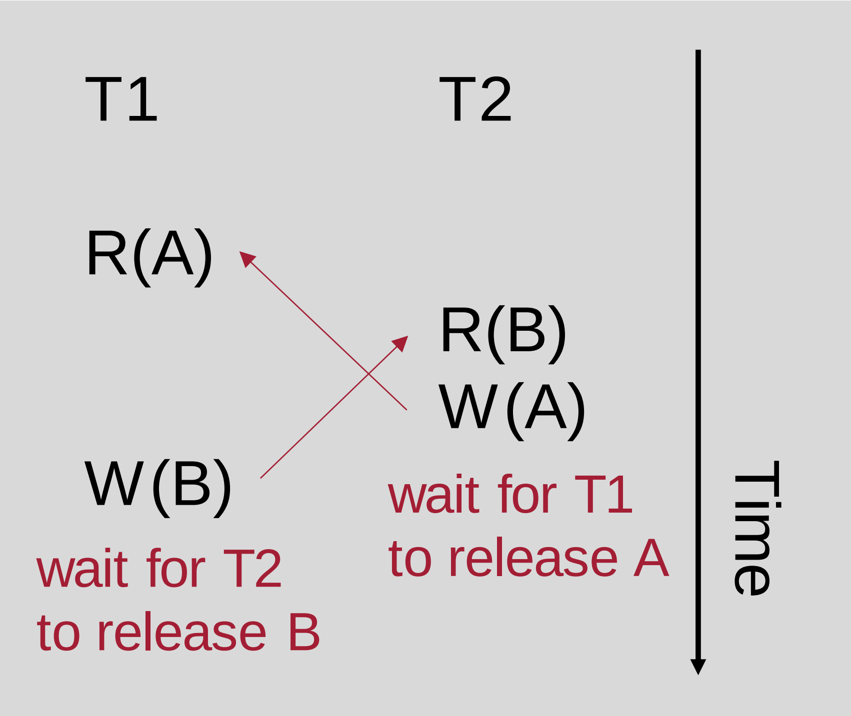
Two-phase locking (2PL) ensures serializability

- **Growing phase**: acquire but do not release locks
- **Shrinking phases**: release but do not acquire locks

Strict 2PL (S2PL): release locks only after a transaction commits or aborts

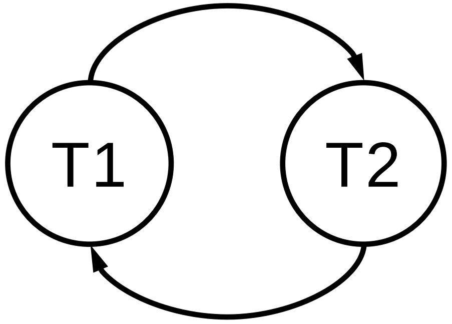
Deadlock in 2PL

Deadlock is a significant challenge in 2PL

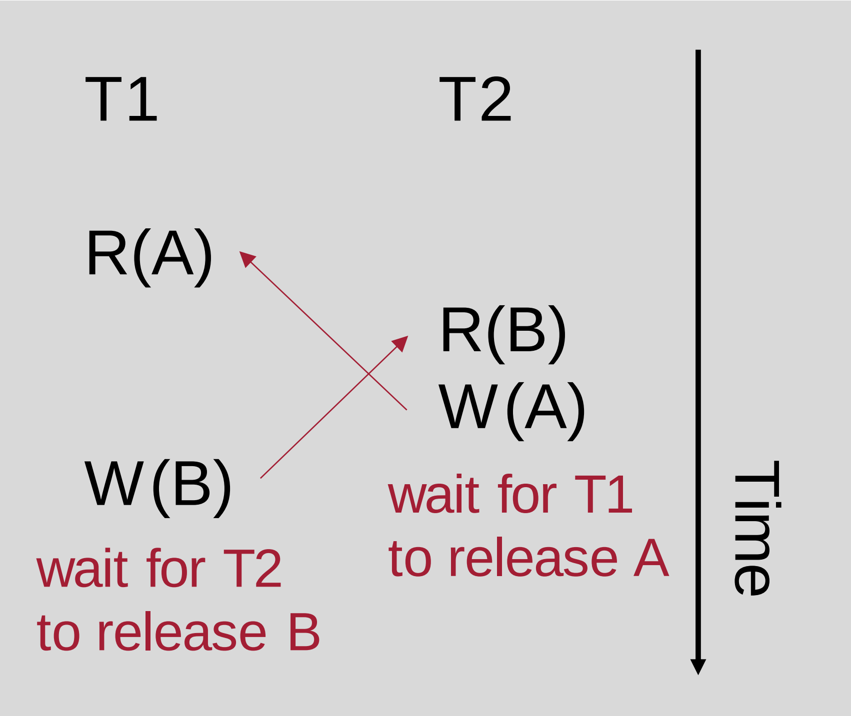


Deadlock in 2PL

Deadlock is a significant challenge in 2PL



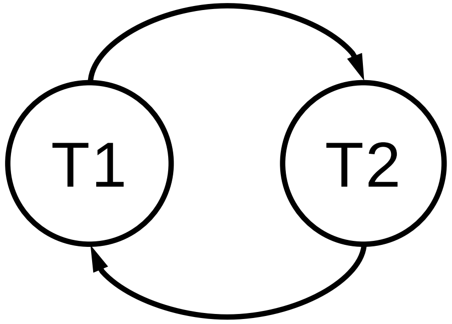
Waits-for graph



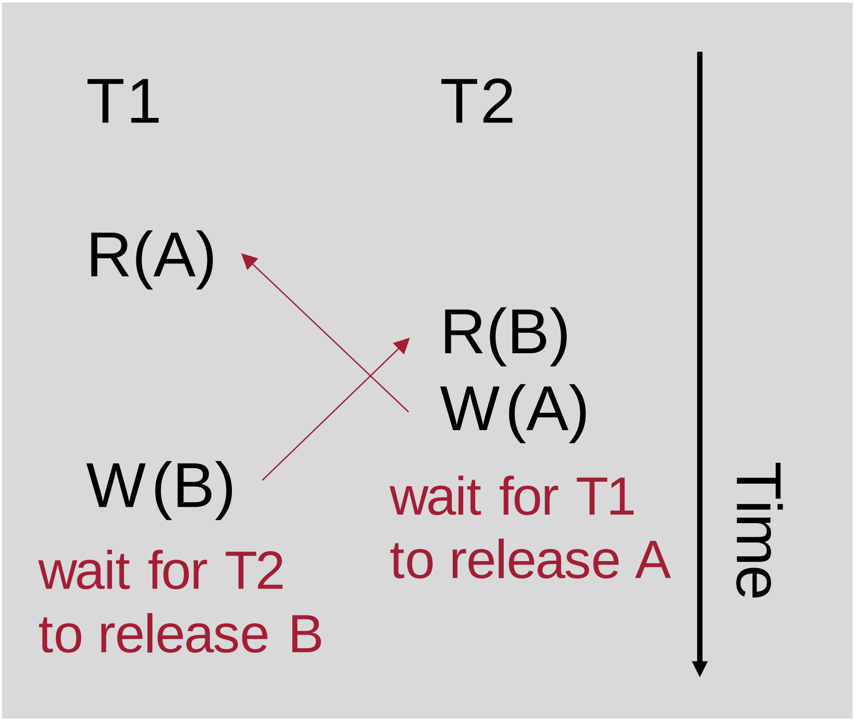
Deadlock in 2PL

Deadlock is a significant challenge in 2PL

A cycle in the waits-for graph -> **Deadlock**!



Waits-for graph



Deadlock Detection and Prevention

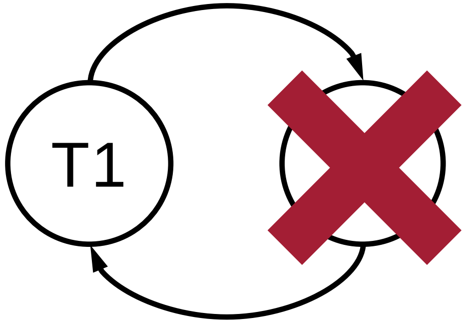
Deadlock detection: Once a cycle is detected, abort one transaction to break the cycle

- DBMS maintains the **waits-for graph**

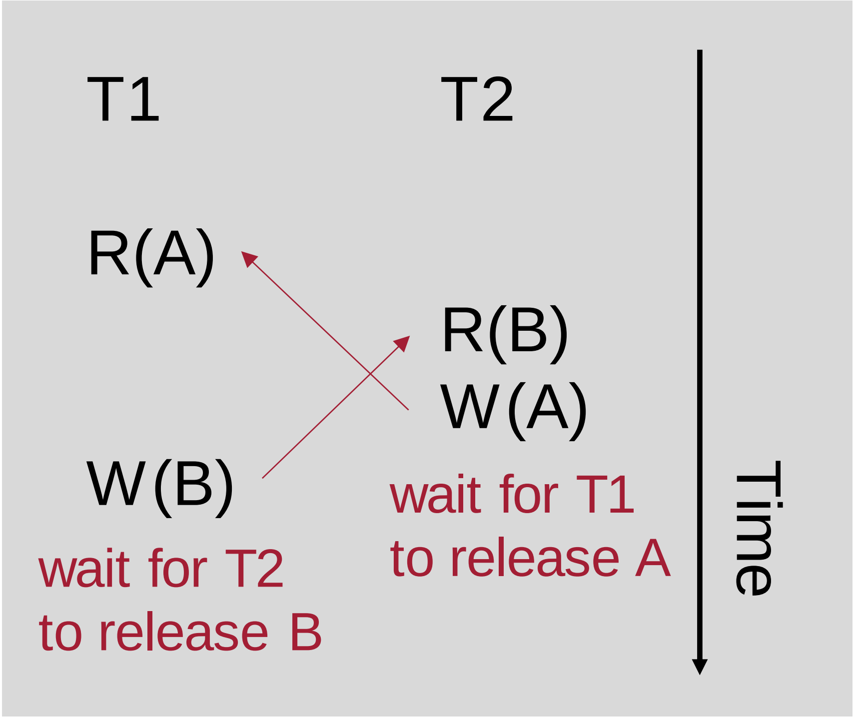
Deadlock Detection

Deadlock detection: Once a cycle is detected, abort one transaction to break the cycle

– DBMS maintains the **waits-for graph**



Waits-for graph



Deadlock Prevention

No-Wait: A transaction self-aborts when encountering a conflict

– No wait -> no deadline

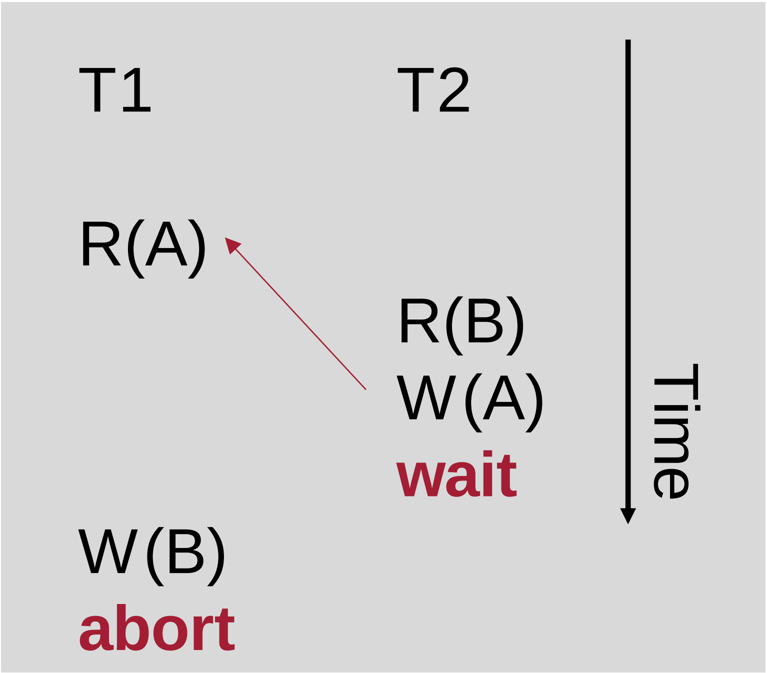
T1	T2
R(A)	
	R(B)
	W(A)
	abort
W(B)	
commit	

Time

Deadlock Prevention

Wait-Die: On a conflict, the requesting transaction **waits** if it has higher priority than the transaction that owns the lock; otherwise the requesting transaction **self-aborts**

- **No deadlock**: wait only from high-priority to low-priority txn
- **No starvation**: highest priority txn will not abort



* Assuming T2 has higher priority

Summary

Transaction basics

ACID properties

- Atomicity
- Consistency
- Isolation (serializability)
- Durability

Two-phase locking (2PL)

- Shared and exclusive lock
- Strict 2PL
- Deadlock