SQL – Transactions

Carla Teixeira Lopes

Bases de Dados
Mestrado Integrado em Engenharia Informática e Computação, FEUP
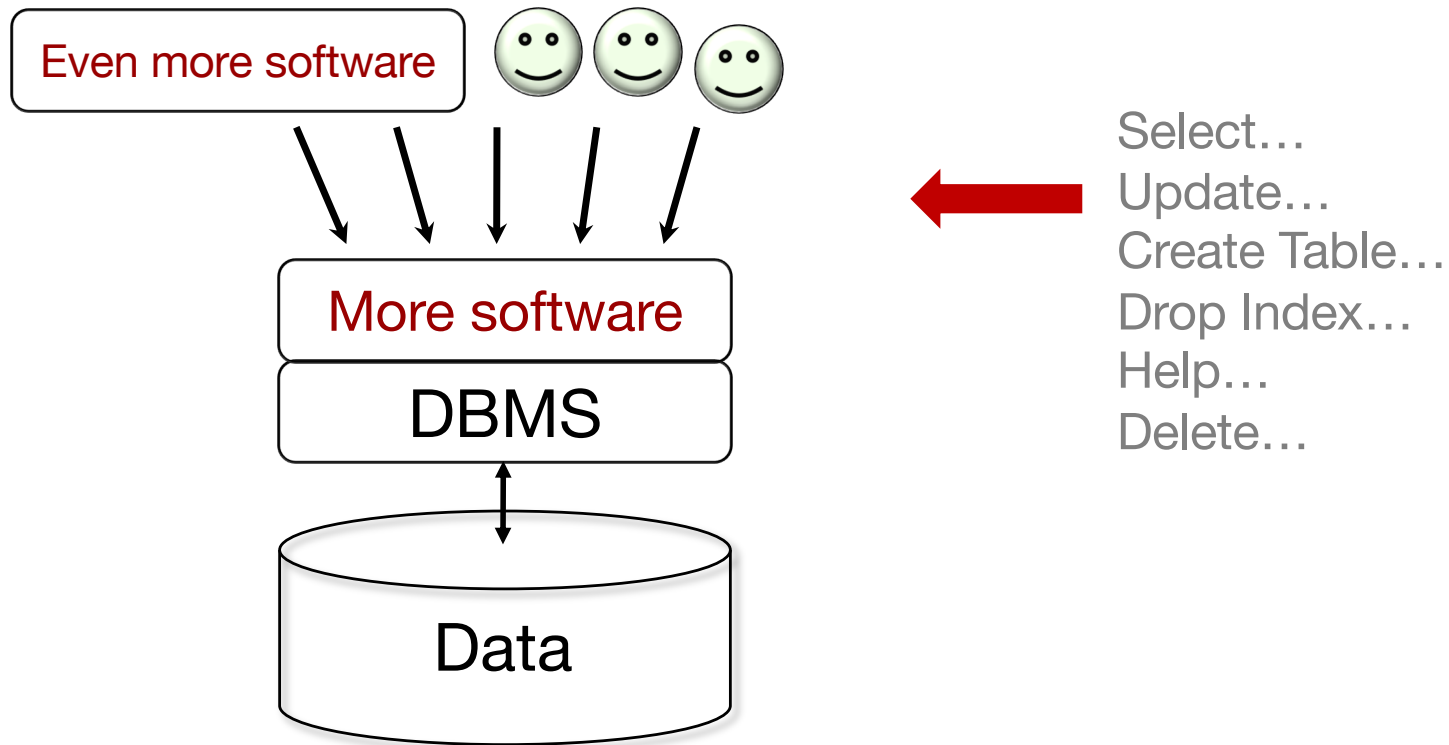
Agenda

Introduction

Properties

Isolation levels

Motivation for transactions

Concurrent database access

Resilience to system failures

Concurrent Database Access

Even more software 😊 😊 😊

More software

DBMS

Data

Select…
Update…
Create Table…
Drop Index…
Help…
Delete…

Attribute-level Inconsistency

Update College Set enr = enr + 1000 Where cName = 'Stanford'

concurrent with …

Update College Set enr = enr + 1500 Where cName = 'Stanford'

	15000	

get; modify; put

15 000 + 2 500 = 17 500

15 000 + 1 000 = 16 000

15 000 + 1 500 = 16 500

Tuple-level Inconsistency

Update Apply **Set** major = 'CS' **Where** sID = 123

concurrent with …

Update Apply **Set** dec = 'Y' **Where** sID = 123

sID	major	dec
123		

get; modify; put

both changes

One of the two changes

Table-level Inconsistency

Update Apply Set decision = 'Y'
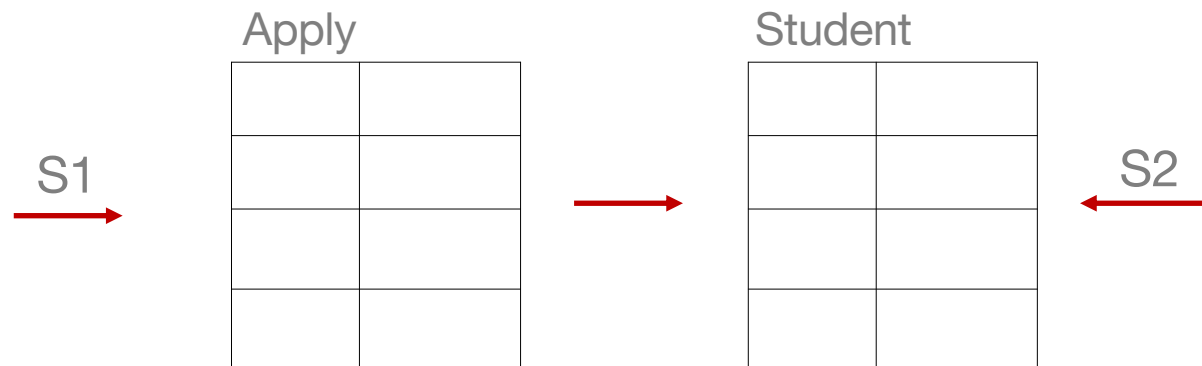
Where sID In (Select sID From Student Where GPA > 3.9)   S1

concurrent with …

Update Student Set GPA = (1.1) * GPA Where sizeHS > 2500   S2
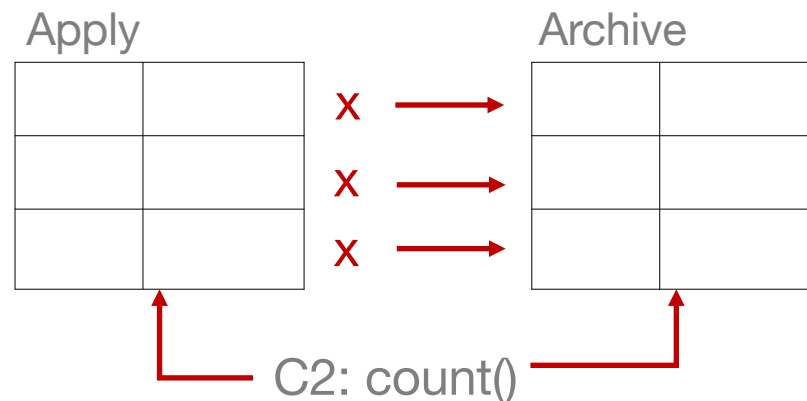
Multi-statement Inconsistency

Insert Into Archive Select * From Apply Where decision = 'N';

Delete From Apply Where decision = 'N';

C1

concurrent with …

Select Count(*) From Apply;

Select Count(*) From Archive;

C2

Concurrency goal

Execute sequence of SQL statements so they appear to be running in isolation
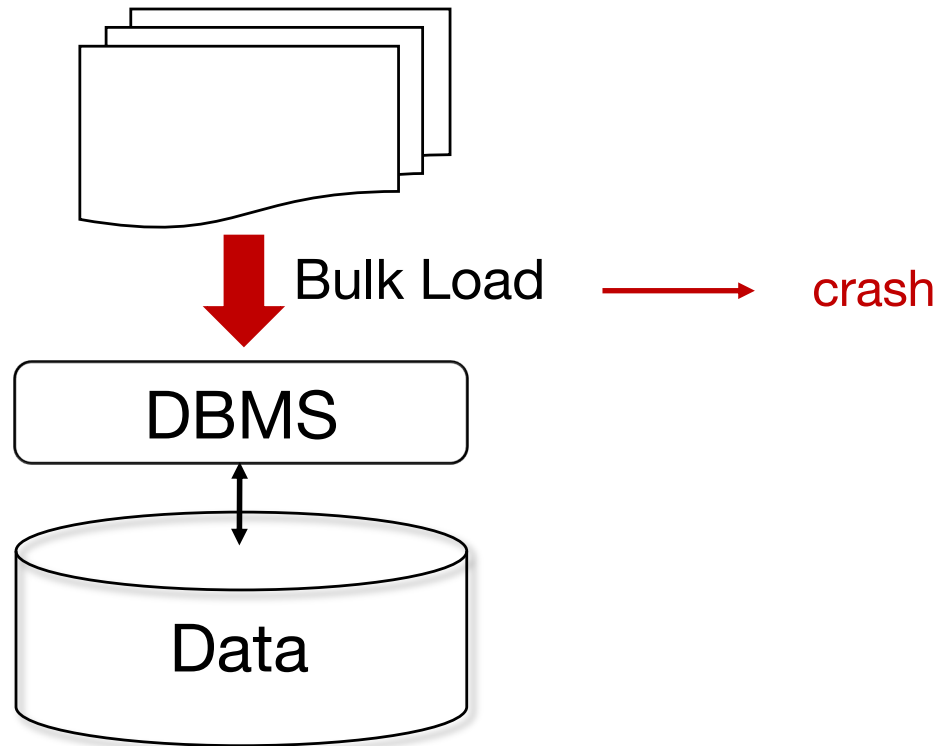
> Simple solution: execute them in isolation

But want to enable concurrency whenever safe to do so

> Multiprocessor system
>
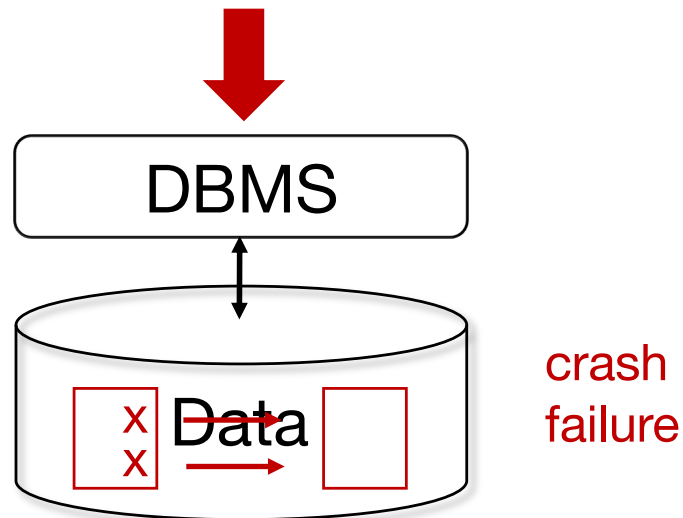> Multithreaded system
>
> Asynchronous I/O

Resilience to System Failures

Resilience to System Failures

Insert Into Archive
 Select * From Apply Where decision = 'N';

Delete From Apply Where decision = 'N';

DBMS

Data

crash
failure

Resilience to System Failures

Lots of updates
buffered in memory

DBMS
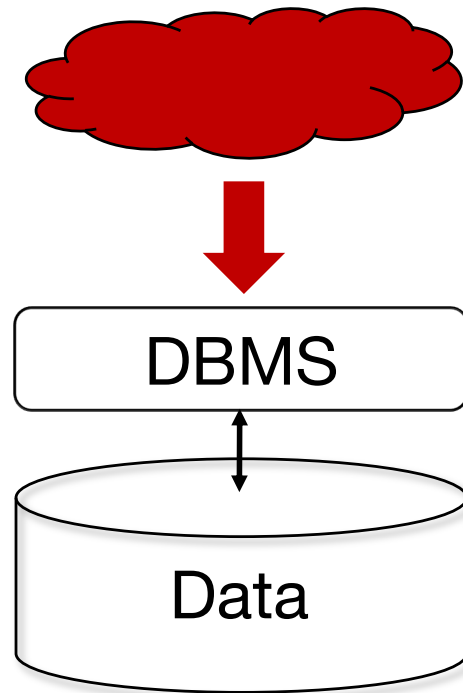
Data
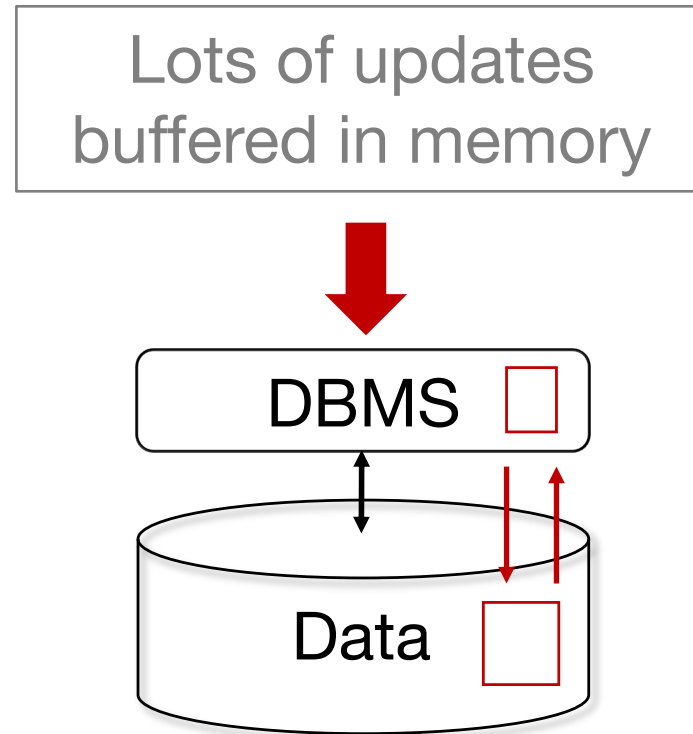
System-Failure Goal

Guarantee all-or-nothing execution, regardless of failures

Transactions

Solution for both concurrency and failures

A transaction is a sequence of one or more SQL operations treated as a unit

- Transactions appear to run in isolation

- If the system fails, each transaction's changes are reflected either entirely or not at all

Transactions: SQL standard

Transaction begins automatically on first SQL statement
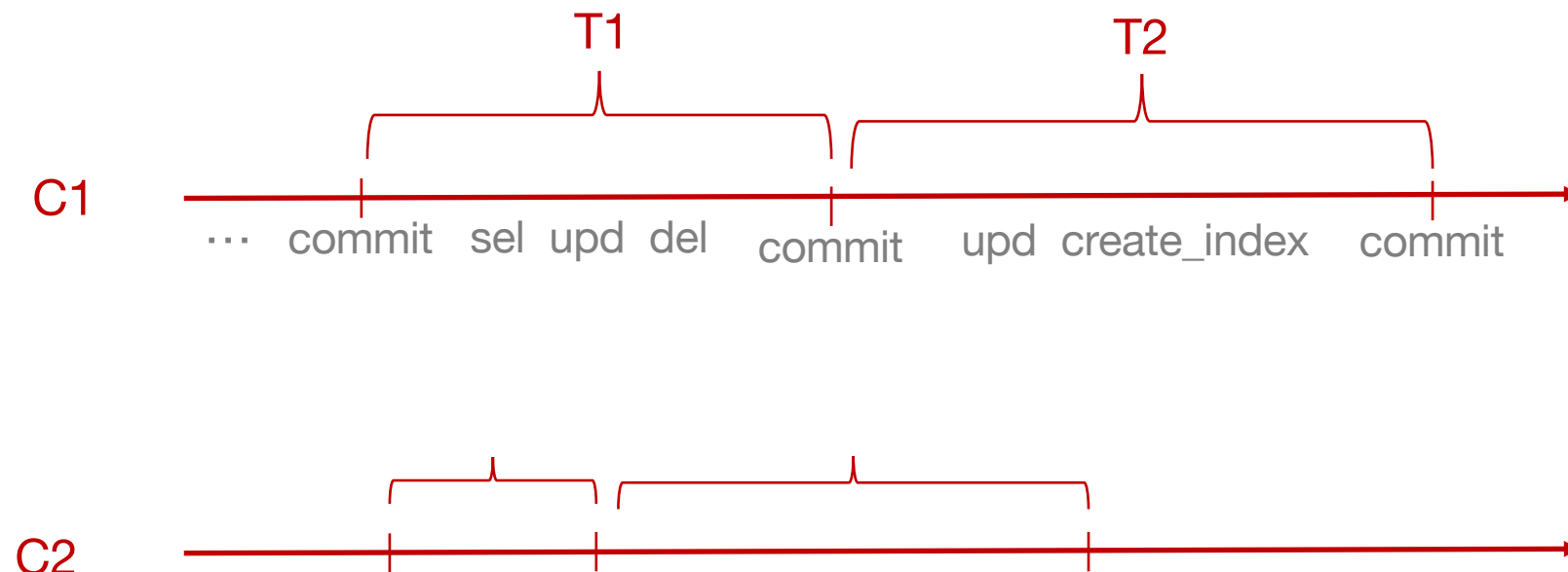
On "commit" transaction ends and new one begins

Current transaction ends on session termination

"Autocommit" turns each statement into transaction

Transactions

A transaction is a sequence of one or more SQL operations treated as a unit

Agenda

ACID Properties

Atomicity   3

Consistency   4

Isolation   1

Durability   2

ACID Properties: Isolation
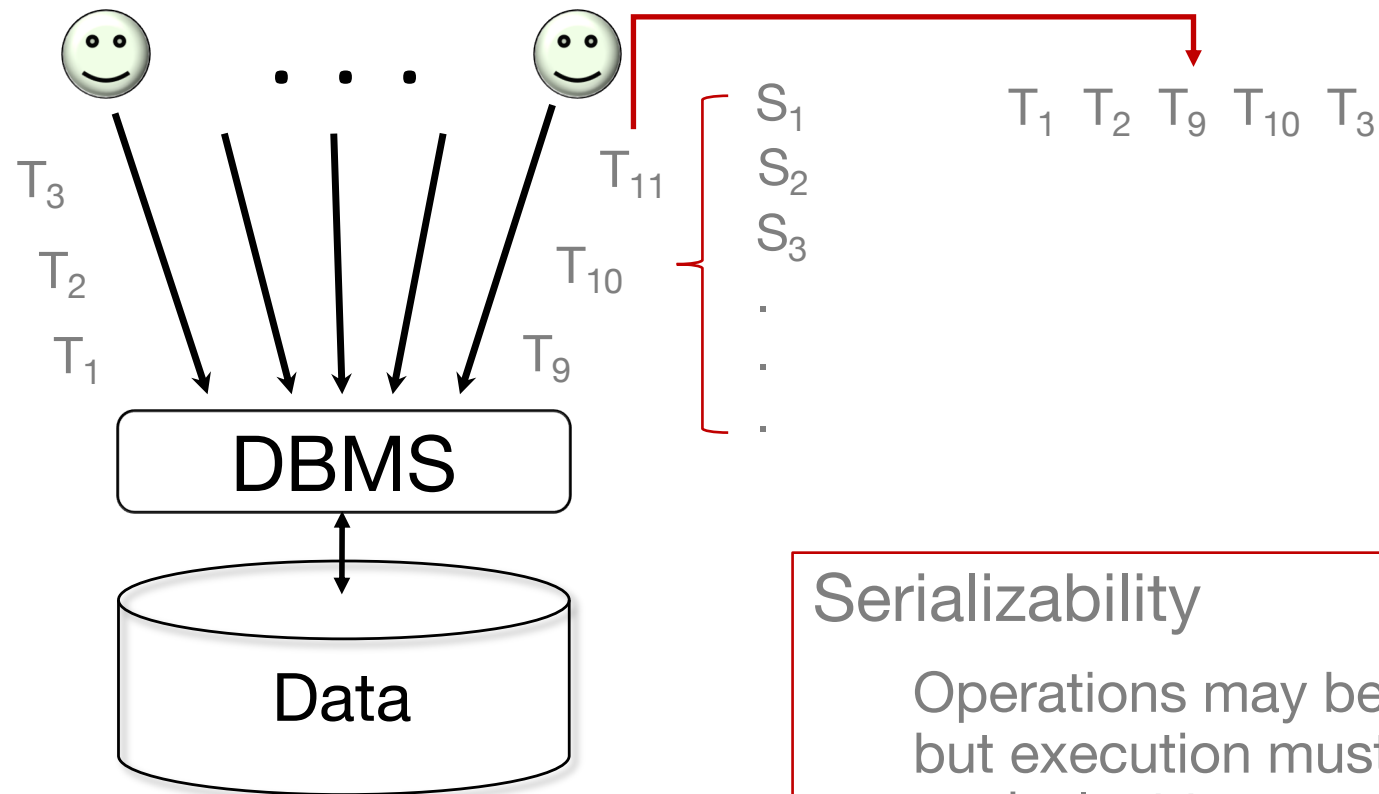
T_3

T_2

T_1

T_{10}

T_9

T_{11}

DBMS

Data

S_1 T_1 T_2 T_9 T_{10} T_3

S_2

S_3

.

.

.

Serializability

Operations may be interleaved, but execution must be equivalent to some sequential (serial) order of all transactions

Attribute-level Inconsistency

Update College **Set** enr = enr + 1000 **Where** cName = 'Stanford' T_1

concurrent with …

Update College **Set** enr = enr + 1500 **Where** cName = 'Stanford' T_2

If serializability is guaranteed
  $T_1; T_2$
  $T_2; T_1$ ⟶ 15 000 → 17 500

Tuple-level Inconsistency

Update Apply **Set** major = 'CS' **Where** sID = 123 $\quad\quad$ T_1

concurrent with …

Update Apply **Set** dec = 'Y' **Where** sID = 123 $\quad\quad$ T_2

If serializability is guaranteed

T_1; T_2
T_2; T_1 $\quad\longrightarrow\quad$ Both changes

Table-level Inconsistency

Update Apply Set decision = 'Y'

Where sID In (Select sID From Student Where GPA > 3.9) $\Big]$ T_1

concurrent with …

Update Student Set GPA = (1.1) * GPA Where sizeHS > 2500 $\Big]$ T_2

If serializability is guaranteed
    T_1; T_2
    T_2; T_1 $\longrightarrow$ Order matters $\longrightarrow$ DBMS don't guarantee the exact sequential order if the transactions are being issued at the same time

Multi-statement Inconsistency

Insert Into Archive Select * From Apply Where decision = 'N';

Delete From Apply Where decision = 'N';

T_1

concurrent with …

Select Count(*) From Apply;
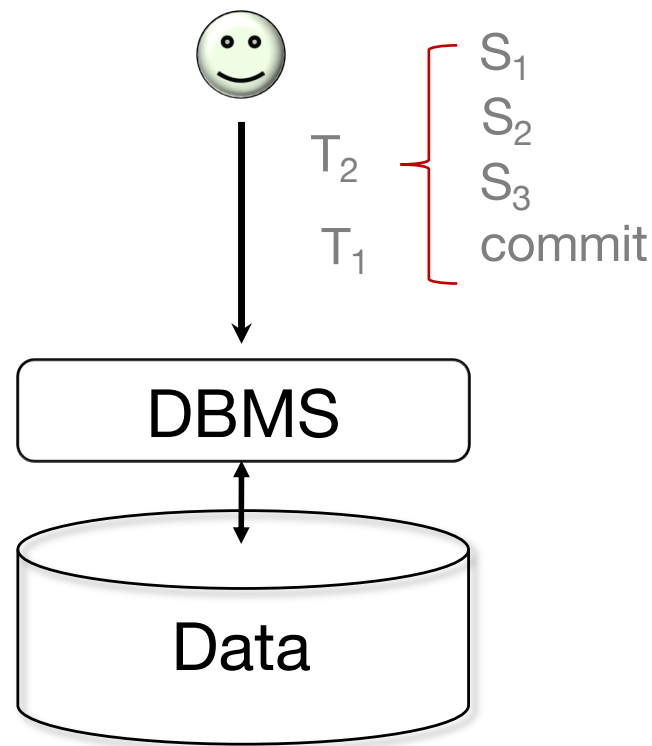
Select Count(*) From Archive;

T_2

If serializability is guaranteed

$T_1; T_2$

$T_2; T_1$ ⟶ Order matters
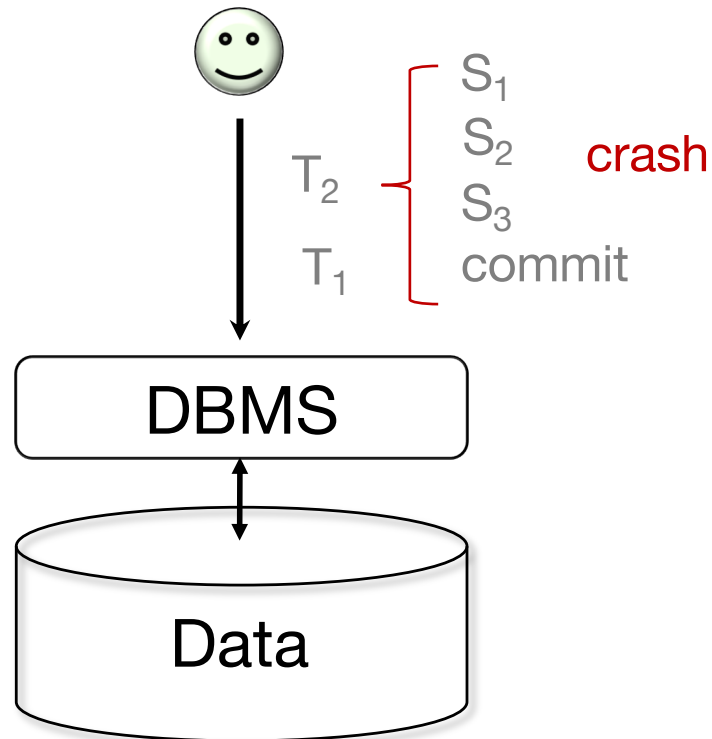
ACID Properties: Durability

If system crashes after transaction commits, all effects
of transaction remain in database

ACID Properties: Atomicity

Each transaction is "all-or-nothing", never left half done

Using a logging mechanism, partial effects of transactions at the time of crash are undone

Transaction Rollback (= Abort)

Undoes partial effects of transaction

Can be system- or client-initiated

Transactions should be constructed to run quickly

```
Begin Transaction;
<get input from user>
SQL commands based on input
<confirm results with user>
If ans='ok' Then Commit; Else Rollback;
```
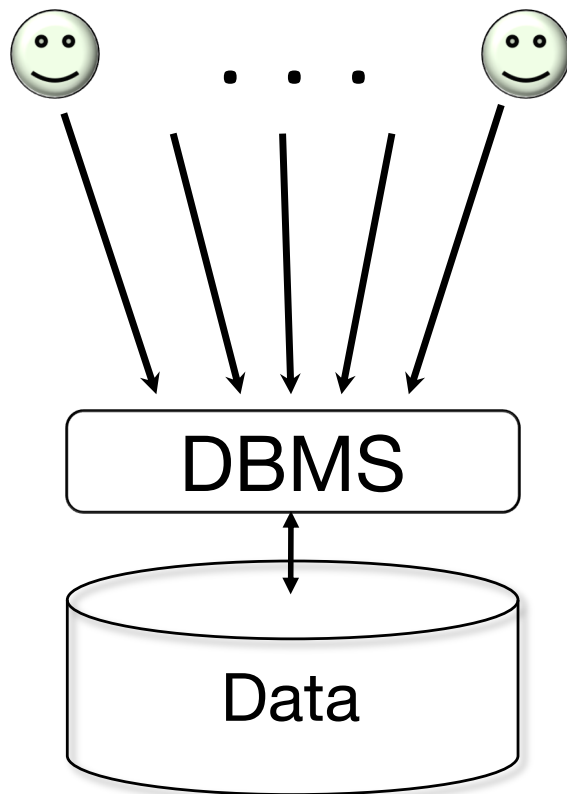
→ Not wait arbitrary amounts of time

Locking

Only undoes effects on the data itself
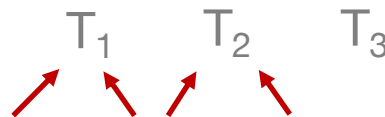
ACID Properties: Consistency

Each client, each transaction:

Can assume all constraints hold when transaction begins

Must guarantee all constraints hold when transaction ends

Serializability

Constraints always hold

T_1 T_2 T_3

Agenda

Introduction

Properties

Isolation levels

ACID Properties: Isolation
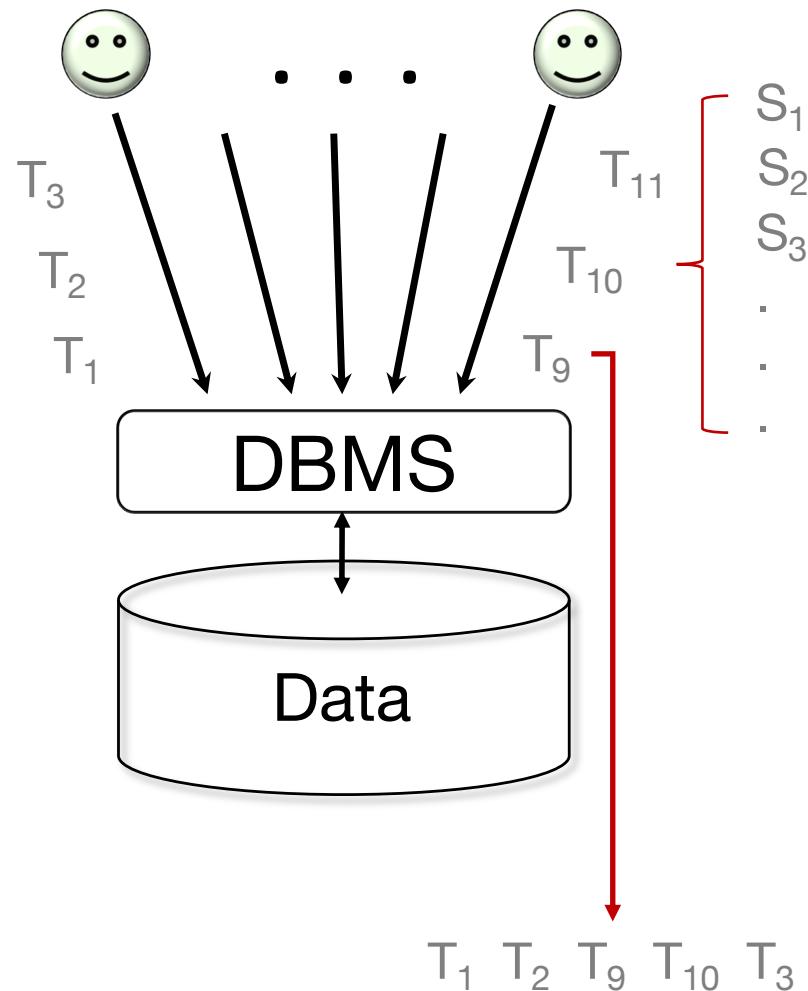
Serializability

Operations may be interleaved, but execution must be equivalent to some sequential (serial) order of all transactions

S_1
S_2
S_3
.
.
.

T_3

T_2

T_1

T_{11}

T_{10}

T_9

DBMS

Data

T_1 T_2 T_9 T_{10} T_3

Disadvantages

Overhead in locking

Reduction in concurrency

ACID Properties: Isolation
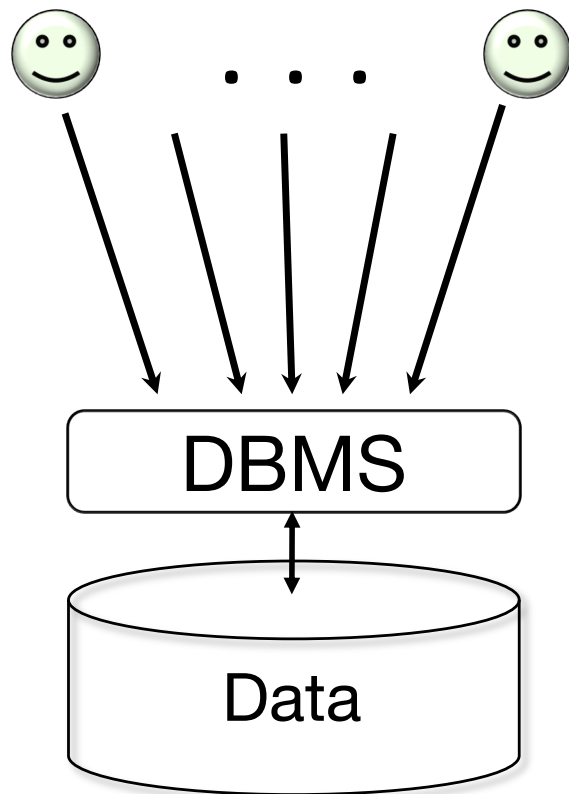
Weaker "Isolation Levels"

Read Uncommitted

Read Committed

Repeatable Read

Weak
↓
Strong

Serializable

↓Overhead in locking

↑Concurrency

↓Consistency Guarantees

Isolation Levels

Per transaction

It does not affect the behaviour of any other transaction

Specific to Reads
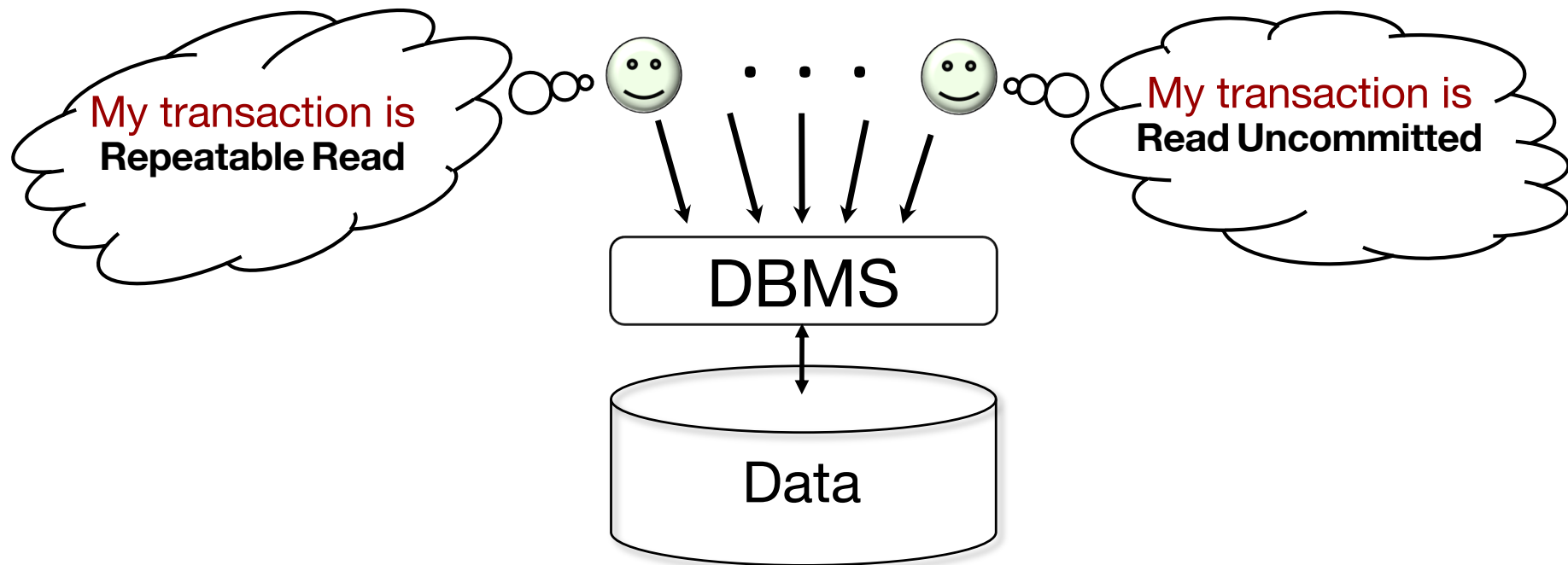
Dirty Reads

"Dirty" data item: written by an uncommitted transaction

Update College Set enr = enr + 1000 Where cName = 'Stanford' T_1

concurrent with …

Select avg(enr) From College T_2

If read before T1 commits, this value is known as dirty

Assume there is a commit at the end of each box

Dirty Reads – Example 2

Update Student Set GPA = (1.1) * GPA Where sizeHS > 2500 T_1

concurrent with …

Select GPA From Student Where sID=123 T_2

concurrent with …

Update Student Set sizeHS=2600 Where sID=234 T_3

Where can we have dirty data items?

There are no dirty reads within the same transaction

Read Uncommitted

A transaction may perform dirty reads

| Update Student Set GPA = (1.1) * GPA Where sizeHS > 2500 | T_1 |

concurrent with …

| Select avg(GPA) From Student | T_2 |

If transactions are serializable

    T1; T2 or

    T2; T1

Read Uncommitted

| Update Student Set GPA = (1.1) * GPA Where sizeHS > 2500 | T_1 |

concurrent with …

| Set Transaction Isolation Level Read Uncommitted;

 Select avg(GPA) From Student; | T_2 |

We don't have serializable behaviour

    We might don't care that much about consistency

Read Committed

A transaction may **not** perform dirty reads

Still does not guarantee global serializability

Update Student Set GPA = (1.1) * GPA Where sizeHS > 2500   T$_1$

concurrent with …

Set Transaction Isolation Level Read Committed;

Select avg(GPA) From Student

Select max(GPA) From Student   T$_2$

Repeatable Read

A transaction may **not** perform dirty reads

An item read multiple times cannot change value

Still does not guarantee global serializability

Update Student Set GPA = (1.1) * GPA Where sizeHS > 2500;

Update Student Set sizeHS=1500 Where sID = 123;

T_1

concurrent with ...

Set Transaction Isolation Level Repeatable Read;

Select avg(GPA) From Student

T_2

Select avg(sizeHS) From Student

Repeatable Read

A transaction may **not** perform dirty reads

An item read multiple times cannot change value

But a relation *can* change: "phantom" tuples

Phantom tuples

Insert into Student [100 new tuples] T_1

concurrent with …

Set Transaction Isolation Level Repeatable Read;

Select avg(GPA) From Student

Select max(GPA) From Student T_2

Repeatable Read

A transaction may **not** perform dirty reads

An item read multiple times cannot change value

But a relation *can* change: "phantom" tuples

Delete from Student [100 new tuples] T_1

concurrent with …

Set Transaction Isolation Level Repeatable Read;

Select avg(GPA) From Student T_2

Select max(GPA) From Student

Once read, values get locked and deletion is not possible in the middle of T_2

Read Only Transactions

Helps system optimize performance

Independent of isolation level

Not going to perform modifications to the database within the transaction

Set Transaction Read Only;

Set Transaction Isolation Level Repeatable Read;

Select avg(GPA) From Student

Select max(GPA) From Student

Isolation Levels: Summary

weak

strong

	dirty reads	nonrepeatable reads	phantoms
Read Uncommitted	Y	Y	Y
Read Committed	N	Y	Y
Repeatable Read	N	N	Y
Serializable	N	N	N

Isolation Levels: Summary

Standard default: Serializable

Weaker isolation levels

  Increased concurrency + decreased overhead = increased performance

  Weaker consistency guarantees

  Some systems have default Repeatable Read

Isolation level per transaction

  Each transaction's reads must conform to its isolation level

Kahoot time!

Any doubts?

Readings

Jeffrey Ullman, Jennifer Widom, A first course in Database Systems 3rd Edition

    Section 6.6 – Transactions in SQL