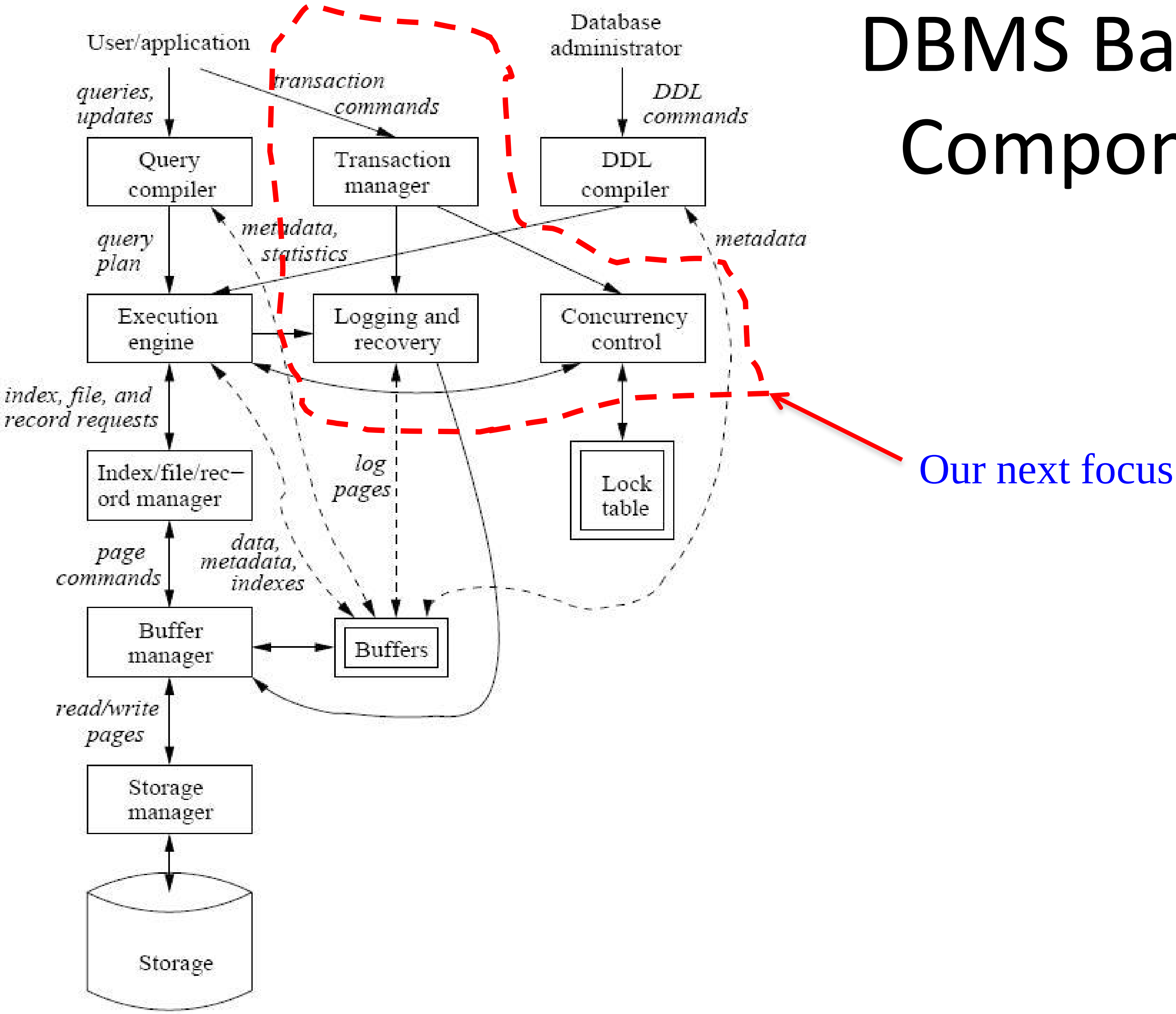
Transaction Management

Motivation

DBMS Backend Components



Database management system components

Transactions

- A *transaction* = sequence of operations that either all succeed, or all fail
- Basic unit of processing in DBMS
- **Transactions have the ACID properties:**

  A = atomicity

  C = consistency

  I = independence (Isolation)

  D = durability

Goal: The ACID properties

- **A tomicity:** All actions in the transaction happen, or none happen.
- **C onsistency:** If each transaction is consistent, and the DB starts consistent, it ends up consistent.
- **I solation:** Execution of one transaction is isolated from that of all others.
- **D urability:** If a transaction commits, its effects persist.

Integrity & Consistency of Data

- Data in the DB should be always **correct** and **consistent**

Is this data correct (consistent)?

Name	Age
White	52
Green	3421
Gray	1

How DBMS decides if data is consistent?

Integrity & Consistency **Constraints**

- Define **predicates** and **constraints** that the data must satisfy

- ***Examples:***
  - x is key of relation R
  - $x \rightarrow y$ holds in R
  - Domain(x) = {Red, Blue, Green}
  - No employee should make more than twice the average salary

Defining constraints (CS3431)

Schema-level
Add Constraint command

```
CREATE TABLE Students
          (sid: CHAR(20),
           name: CHAR(20) NOT NULL,
           login: CHAR(10),
           age: INTEGER,
           gpa: REAL Default 0,
  Constraint pk Primary Key (sid),
  Constraint u1 Unique (login));
```

Business-constraint
Use of Triggers

```
Create Trigger EmpBonus
Before Insert Or Update On Employee
For Each Row
Begin
    :new.bonus := :new.salary * 0.03;
End;
/
```

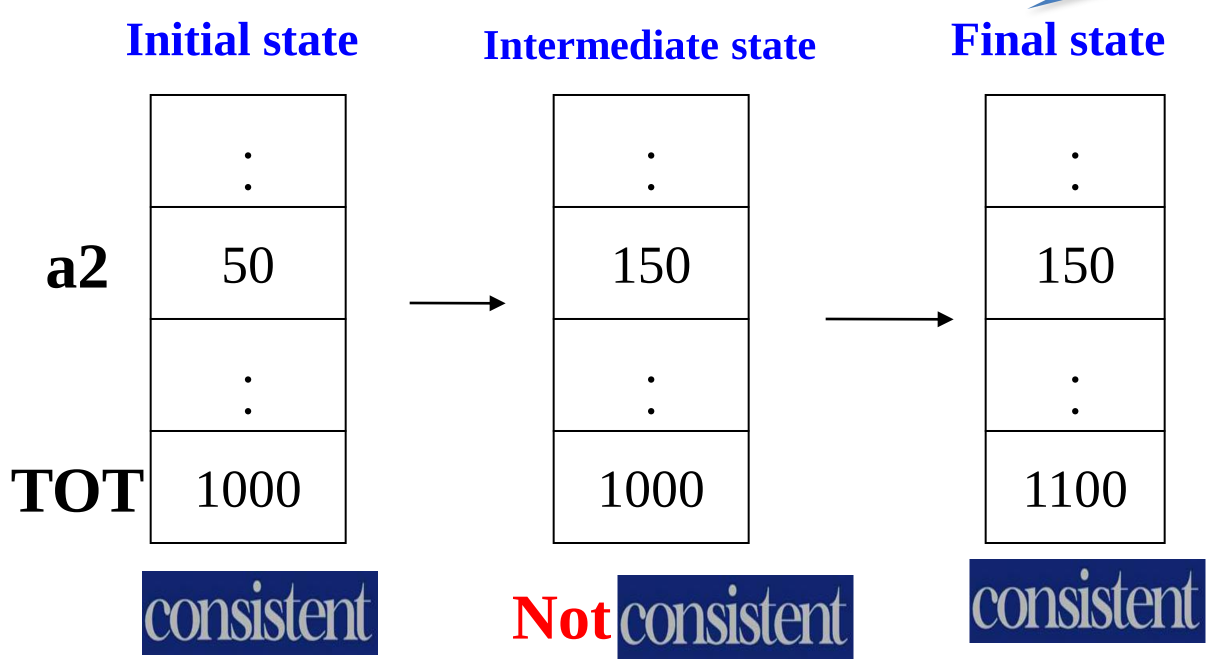
FACT: DBMS is Not Consistent All the Time

Example: $a_1 + a_2 + \ldots a_n = TOT$ (constraint)

Deposit \$100 in a_2: $a_2 \leftarrow a_2 + 100$

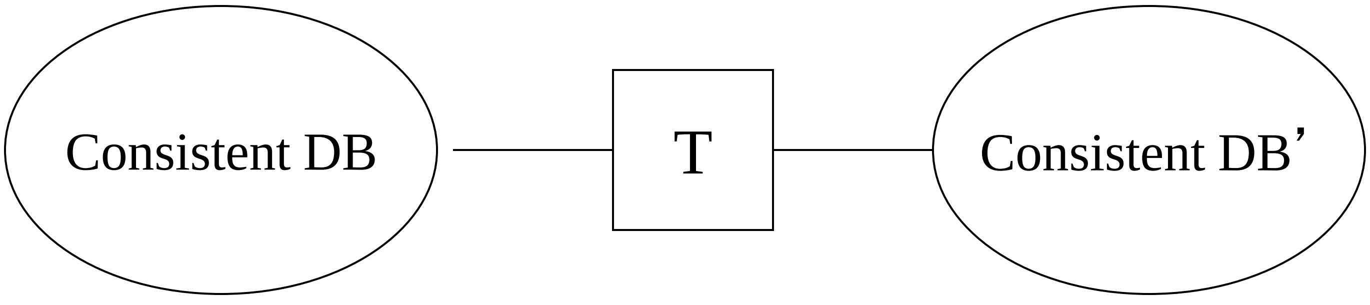
$TOT \leftarrow TOT + 100$

A transaction hides intermediate states
(Even under failure)



Concept of Transactions

Transaction: a collection of actions that preserve consistency



Main Assumption

If T starts with *consistent state*

AND

T executes in *isolation*

THEN

$\Rightarrow$ T leaves consistent state

How Can Constraints Be Violated?

- **Transaction Bug**
  - The semantics of the transaction is wrong
  - E.g., update a2 and not ToT
- **DBMS Bug**
  - DBMS fails to detect inconsistent states
- **Hardware Failure**
  - Disk crash, memory failure, …
- **Concurrent Access**
  - Many transactions accessing the data at the same time
  - E.g., T1: give 10% raise to programmers

    T2: change programmers $\Rightarrow$ systems analysts

DBMS can easily detect and prevent that (if constraints are defined)

Should not use this DBMS

Our focus & Major components in DBMS

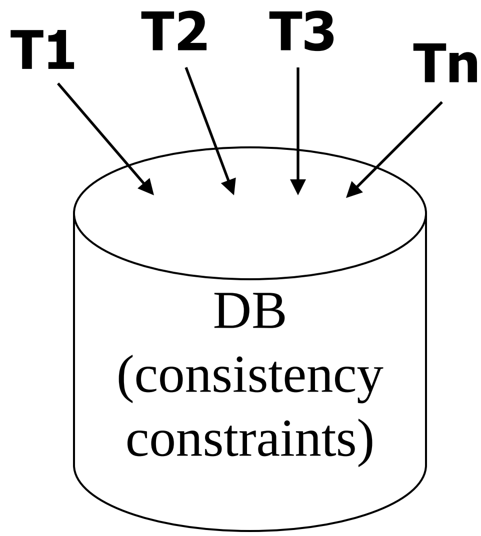
How Can We Prevent/Fix Violations?

- Chapter 17: Due to failures only
- Chapter 18: Due to concurrent access only
- Chapter 19: Due to failures and concurrent access

Plan of Attack (ACID properties)

- First we will deal with "I", by focusing on concurrency control.
- Then we will address "A" and "D" by looking at recovery.
- **What about "C"?**
  - Well, if you have the other three working, and you set up your integrity constraints correctly, then you get "C" for free

Concurrent Transactions



- Many transactions access the data at the same time
- Some are ***reading***, others are ***writing***
- May conflict

Transactions: Example

T1: Read(A)	**T2:** Read(A)
$A \leftarrow A + 100$	$A \leftarrow A \times 2$
Write(A)	Write(A)
Read(B)	Read(B)
$B \leftarrow B+100$	$B \leftarrow B \times 2$
Write(B)	Write(B)

Constraint: A=B

- How to execute these two transactions?
- How to ***schedule*** the ***read/write*** operations?

A Schedule

An ordering of operations (reads/writes) inside one or more transactions over time

What is correct outcome ?

Leads To

What is good schedule ?

Schedule A

T1: Read(A)
A ¬ A + 100
Write(A)
Read(B)
B ¬ B+100
Write(B)

T2: Read(A)
A ¬ A´ 2
Write(A)
Read(B)
B ¬ B´ 2
Write(B)

Constraint: A=B

T1	T2	A	B
		25	25
Read(A); A $\leftarrow$ A+100			
Write(A);		125	
Read(B); B $\leftarrow$ B+100;			
Write(B);			125
	Read(A);A $\leftarrow$ A$\times$2;		
	Write(A);	250	
	Read(B);B $\leftarrow$ B$\times$2;		
	Write(B);		250
		250	250

Serial Schedule: T1, T2

Schedule B

T1:	**T2:**
Read(A)	Read(A)
A ¬ A + 100	A ¬ A ´ 2
Write(A)	Write(A)
Read(B)	Read(B)
B ¬ B+100	B ¬ B ´ 2
Write(B)	Write(B)

Constraint: A=B

T1	T2	A	B
		25	25
	Read(A);A ¬ A´ 2;		
	Write(A);	50	
	Read(B);B ¬ B´ 2;		
	Write(B);		50
Read(A); A ¬ A+100			
Write(A);		150	
Read(B); B ¬ B+100;			
Write(B);			150
		150	150

Serial Schedule: T2, T1

Serial Schedules !

- **Definition:** A schedule in which transactions are performed in a serial order (no interleaving)

- **The Good:** Consistency is guaranteed
- ➔Any serial schedule is "good".

- **The Bad:** Throughput is low, need to execute in parallel

Solution ➔ Interleave Transactions in A Schedule…

Schedule C

T1: Read(A)
A ¬ A + 100
Write(A)
Read(B)
B ¬ B+100
Write(B)

T2: Read(A)
A ¬ A ´ 2
Write(A)
Read(B)
B ¬ B ´ 2
Write(B)

Constraint: A=B

T1	T2	A	B
		25	25
Read(A); A ¬ A+100			
Write(A);		125	
	Read(A);A ¬ A´ 2;		
	Write(A);	250	
Read(B); B ¬ B+100;			
Write(B);			125
	Read(B);B ¬ B´ 2;		
	Write(B);		250
		250	250

Schedule C is ***NOT*** serial but its ***Good***

Schedule D

T1:		T2:	
Read(A)		Read(A)	
	A ¬ A + 100		A ¬ A ´ 2
	Write(A)		Write(A)
	Read(B)		Read(B)
	B ¬ B+100		B ¬ B ´ 2
	Write(B)		Write(B)

Constraint: A=B

T1	T2	A	B
		25	25
Read(A); A ¬ A+100			
Write(A);		125	
	Read(A);A ¬ A´ 2;		
	Write(A);	250	
	Read(B);B ¬ B´ 2;		
	Write(B);		50
Read(B); B ¬ B+100;			
Write(B);			150
		250	150

Schedule C is ***NOT*** serial but its ***Bad***

Not Consistent

Schedule E

Same as Schedule D but with **new T2'**

T1	T2'	A	B
		25	25
Read(A); A ¬ A+100			
Write(A);		125	
	Read(A);A ¬ A´ 1;		
	Write(A);	125	
	Read(B);B ¬ B´ 1;		
	Write(B);		25
Read(B); B ¬ B+100;			
Write(B);			125
		125	125

Consistent

Same schedule as D, but this one is ***Good***

What Is A ‘Good’ Schedule?

- **Does not depend only on the sequence of operations**
  - Schedules D and E have the same sequence
  - D produced inconsistent data
  - E produced consistent data

Transaction semantics played a role

- **We want schedules that are guaranteed “good” regardless of:**
  - The initial state and
  - The transaction semantics

- **Hence we consider only:**
  - The order of read/write operations
  - Any other computations are ignored (transaction semantics)

Example:
Schedule S = $r_1(A)$ $w_1(A)$ $r_2(A)$ $w_2(A)$ $r_1(B)$ $w_1(B)$ $r_2(B)$ $w_2(B)$

Example: Considering Only R/W Operations

T1	T2’
Read(A); A ¬ A+100	
Write(A);	
	Read(A);A ¬ A´ 1;
	Write(A);
	Read(B);B ¬ B´ 1;
	Write(B);
Read(B); B ¬ B+100;	
Write(B);	

Schedule S = $r_1(A)\ w_1(A)\ r_2(A)\ w_2(A)\ r_2(B)\ w_2(B)\ r_1(B)\ w_1(B)$

Concept: Conflicting Actions

Conflicting actions: Two actions from two different transactions on the same object are conflicting iff one of them is write

Conflict	r1(A) ←→ W2(A)	→ Transaction 1 reads A, Transaction 2 write A
Conflict	w1(A) ←→ r2(A)	→ Transaction 1 writes A, Transaction 2 reads A
Conflict	w1(A) ←→ w2(A)	→ Transaction 1 writes A, Transaction 2 write A
No Conflict	r1(A) ←→ r2(A)	→ Transaction 1 reads A, Transaction 2 reads A

Conflicting actions can cause anomalies…Which is Bad

Anomalies with Interleaving

Reading Uncommitted Data (WR Conflicts, "dirty reads"):

e.g. T1: A+100, B+100, T2: A*1.06, B*1.06

T1:	R(A), W(A),		R(B), W(B), Abort	
T2:		R(A), W(A), C		

Unrepeatable Reads (RW Conflicts):

E.g., T1: R(A), ...R(A), decrement, T2: R(A), decrement

T1:	R(A),		R(A), W(A), C
T2:		R(A), W(A), C	

Overwriting Uncommitted Data (WW Conflicts):

T1:	W(A),		W(B), C
T2:		W(A), W(B), C	

We need schedule that is anomaly-free

Our Goal

- We need schedule that is ***equivalent*** to **any** ***serial schedule***

It should allow interleaving

Any serial order is good

Produces consistent result & anomaly-free

Given schedule S:

If we can shuffle the *non-conflicting* actions to reach a serial schedule L

→ S is equivalent to L

→ S is good

Example: Schedule C

T1: Read(A)	**T2:** Read(A)
A ¬ A + 100	A ¬ A´ 2
Write(A)	Write(A)
Read(B)	Read(B)
B ¬ B+100	B ¬ B´ 2
Write(B)	Write(B)

Constraint: A=B

T1	T2	A	B
		25	25
Read(A); A ¬ A+100			
Write(A);		125	
	Read(A);A ¬ A´ 2;		
	Write(A);	250	
Read(B); B ¬ B+100;			
Write(B);			125
	Read(B);B ¬ B´ 2;		
	Write(B);		250
		250	250

Example: Schedule C

T1	T2
Read(A); A ¬ A+100	
Write(A);	
	Read(A);A ¬ A´ 2;
	Write(A);
Read(B); B ¬ B+100;	
Write(B);	
	Read(B);B ¬ B´ 2;
	Write(B);

S_c= r1(A) w1(A) r2(A) w2(A) r1(B) w1(B) r2(B) w2(B)

Can be switched because they are not conflicting

S_c"= r1(A) w1(A) r1(B) w1(B) r2(A) w2(A) r2(B) w2(B)

T1 T2

➔ Schedule C is equivalent to a serial schedule ➔ So it is "Good"

Why Schedule C turned out to be Good ?

(Some Formalization)

T1	T2
Read(A); A ¬ A+100	
Write(A);	
	Read(A);A ¬ A´ 2;
	Write(A);
Read(B); B ¬ B+100;	
Write(B);	
	Read(B);B ¬ B´ 2;
	Write(B);

$S_c = r_1(A)\ w_1(A)\ r_2(A)\ w_2(A)\ r_1(B)\ w_1(B)\ r_2(B)\ w_2(B)$

T1 → T2 *(T1 precedes T2)*

T1 → T2 *(T1 precedes T2)*

☛ No cycles ⇒ S_c is "equivalent" to a serial schedule where T_1 precedes T_2.

Example: Schedule D

T1	T2
Read(A); A ← A+100	
Write(A);	
	Read(A);A ← A×2;
	Write(A);
	Read(B);B ← B×2;
	Write(B);
Read(B); B ← B+100;	
Write(B);	

$S_D = r_1(A)\ w_1(A)\ r_2(A)\ w_2(A)\ r_2(B)\ w_2(B)\ r_1(B)\ w_1(B)$

- Can we shuffle non-conflicting actions to make T1 T2 or T2 T1 ??

Example: Schedule D

T1	T2
Read(A); A ← A+100	
Write(A);	
	Read(A);A ← A×2;
	Write(A);
	Read(B);B ← B×2;
	Write(B);
Read(B); B ← B+100;	
Write(B);	

$S_D = r_1(A)\ w_1(A)\ r_2(A)\ w_2(A)\ r_2(B)\ w_2(B)\ r_1(B)\ w_1(B)$

- **Can we make T1 first ➔ [T1 T2]?**
  - No…Cannot move $r_1(B)$ $w_1(B)$ forward
  - Why: because r1(B) conflict with w2(B) so it cannot move….Same for w1(B)

Example: Schedule D

T1	T2
Read(A); A ← A+100	
Write(A);	
	Read(A);A ← A×2;
	Write(A);
	Read(B);B ← B×2;
	Write(B);
Read(B); B ← B+100;	
Write(B);	

$S_D = r_1(A)\ w_1(A)\ r_2(A)\ w_2(A)\ r_2(B)\ w_2(B)\ r_1(B)\ w_1(B)$

- **Can we make T2 first → [T2 T1]?**
  - No…Cannot move $r_2(A)$ $w_2(A)$ forward
  - Why: because r2(A) conflict with w1(A) so it cannot move….Same for w2(A)

→ Schedule D is **NOT** equivalent to a serial schedule → So it is "Bad"

Why Schedule D turned out to be Bad?

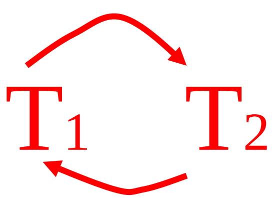
(Some Formalization)

T1	T2
Read(A); A ← A+100	
Write(A);	
	Read(A);A ← A×2;
	Write(A);
	Read(B);B ← B×2;
	Write(B);
Read(B); B ← B+100;	
Write(B);	

$S_D = r_1(A)\ w_1(A)\ r_2(A)\ w_2(A)\ r_2(B)\ w_2(B)\ r_1(B)\ w_1(B)$

T1 → T2 *(T1 precedes T2)*

T2 → T1 *(T2 precedes T1)*



☛ Cycle Exist ⇒ S_D is "Not equivalent" to any serial schedule.