

Today's Lecture

1. Why transactions?
2. Transactions
3. Properties of Transactions: ACID
4. Logging



Lecture: Concurrency & Locking for Transactions

Example

Monthly bank interest transaction

Money

Account	Balance (\$)
3001		500
4001		100
5001		20
6001		60
3002		80
4002		-200
5002		320
...	...	
30108		-100
40008		100
50002		20

Money (@4:29 am day+1)

Account	Balance (\$)
3001		550
4001		110
5001		22
6001		66
3002		88
4002		-220
5002		352
...	...	
30108		-110
40008		110
50002		22



'T-Monthly-423'

Monthly Interest 10%

4:28 am Starts run on 10M bank accounts
Takes 24 hours to run

```
UPDATE Money  
SET Balance = Balance * 1.1
```

Other Transactions

10:02 am Acct 3001: Wants 600\$
11:45 am Acct 5002: Wire for 1000\$
.....
.....

2:02 pm Acct 3001: Debit card for \$12.37

Q: How do I not wait for a day to access my \$\$\$?

Big Idea LOCKS!

Big Idea: LOCKs

- ▷ Intuition:
 - 'Lock' each record for shortest time possible
 - (e.g, Locking Money Table for a day is not good enough)
- ▷ Key questions:
 - Which records? For how long? What's algorithm?



Many kinds of LOCKs. We'll study some simple ones!

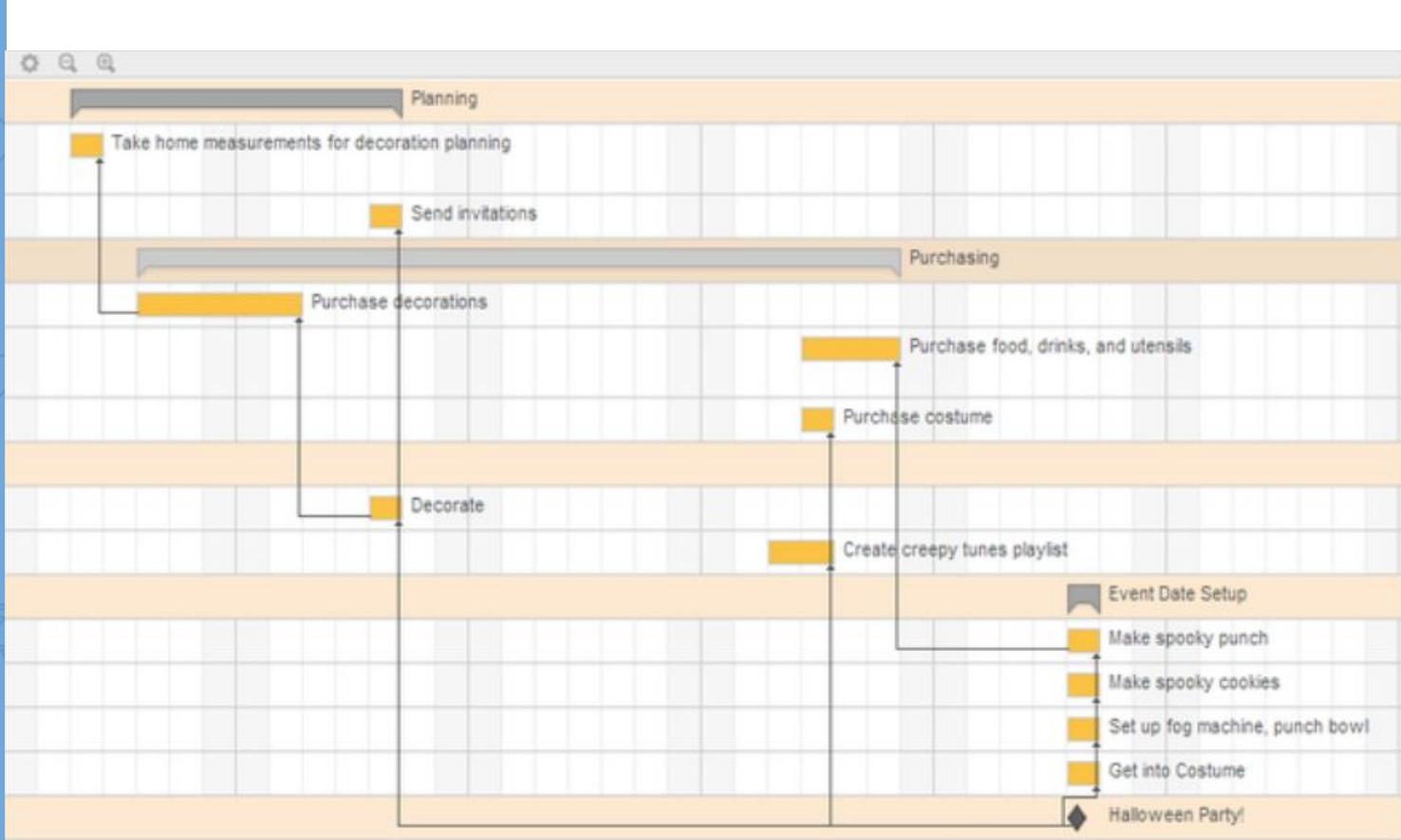


CS Concept Reminder: DAGs & Topological Orderings

- A **topological ordering** of a directed graph is a linear ordering of its vertices that respects all the directed edges
- A directed acyclic graph (DAG) always has one or more **topological orderings**
 - (And there exists a topological ordering *if and only if* there are no directed cycles)

Example TODO list dependencies

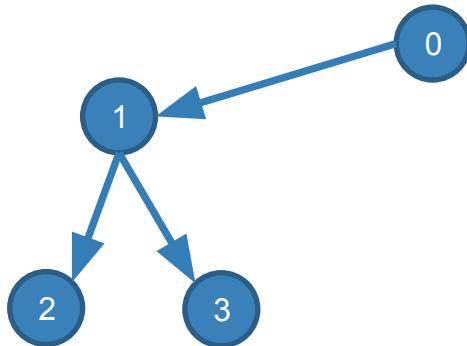
(Intuition for DAGs/
TopoSort)



How would you plan?
What if there are cycles? (dependencies)

DAGs & Topological Orderings

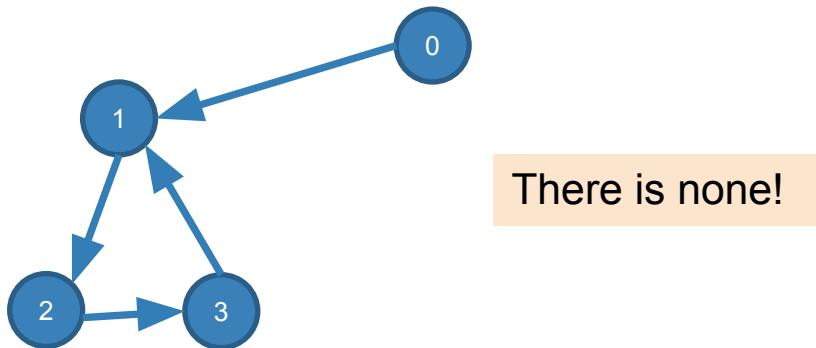
- Ex: What is one possible topological ordering here?



Ex: 0, 1, 2, 3 (or: 0, 1, 3, 2)

DAGs & Topological Orderings

- Ex: What is one possible topological ordering here?





What you will
learn about in
this section

1. Concurrency

- ▷ Interleaving & scheduling (Examples)
- ▷ Conflict & Anomaly types (Formalize)

2. Locking: 2PL, Deadlocks (Algorithm)



Concurrency, Scheduling & Anomalies



Concurrency: Isolation & Consistency

- DBMS maintains
 1. **Isolation:** Users execute each TXN **as if they were the only user** ACID
 2. **Consistency:** TXNs must leave the DB in a **consistent state** ACID



Next 30 mins

1. We'll start with 2 TXNs and 2 rows 'A' and 'B'
2. Then generalize for more TXNs and more rows
3. Next week, how to do the LOCKing



Note the hard part...

...is the effect of *interleaving* transactions and *crashes*.
See 245 for the gory details!

In cs145, we'll focus on a simplified model

Example- consider two TXNs:

```
T1: START TRANSACTION  
    UPDATE Accounts  
    SET Amt = Amt + 100  
    WHERE Name = 'A'  
  
    UPDATE Accounts  
    SET Amt = Amt - 100  
    WHERE Name = 'B'  
  
    COMMIT
```

T1 transfers \$100 from B's account to A's account

```
T2: START TRANSACTION  
    UPDATE Accounts  
    SET Amt = Amt * 1.06  
    COMMIT
```

T2 credits both accounts with a 6% interest payment

Note:

1. DB does not care if T1 → T2 or T2 → T1 (which TXN executes first)
2. If developer does, what can they do? (Put T1 and T2 inside 1 TXN)

Example

T_1

A += 100

B -= 100

T1 transfers \$100 from B's account to A's account

T_2

A *= 1.06

B *= 1.06

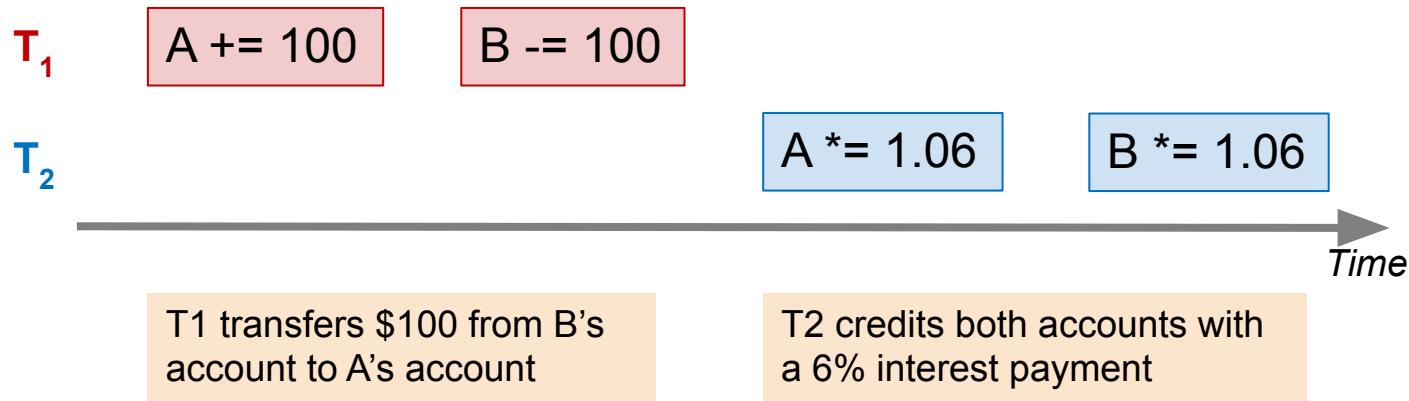
T2 credits both accounts with a 6% interest payment

Goal for scheduling transactions:

- Interleave transactions to boost performance
- Data stays in a good state after commits and/or aborts (ACID)

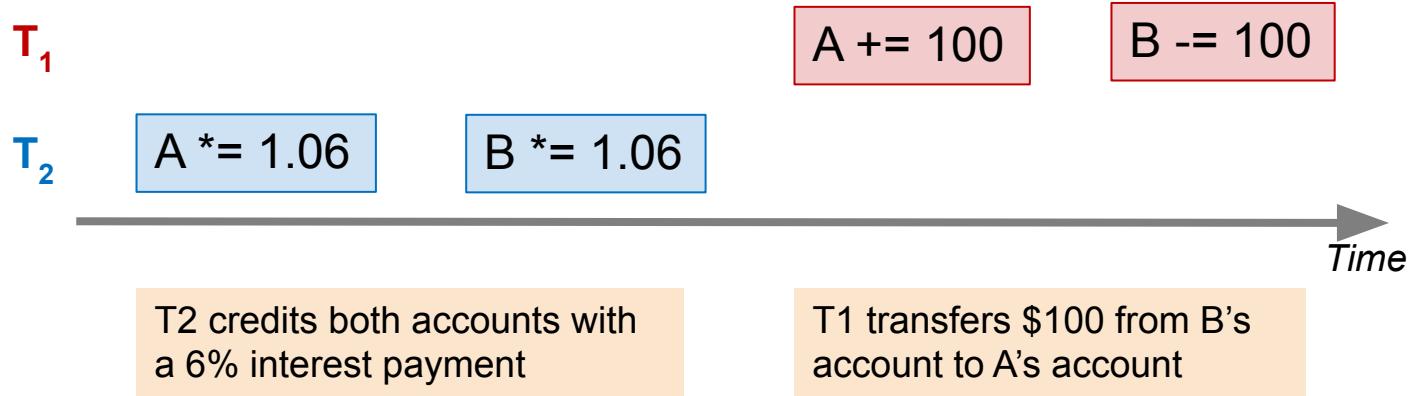
Example- consider two TXNs:

We can look at the TXNs in a timeline view- serial execution:



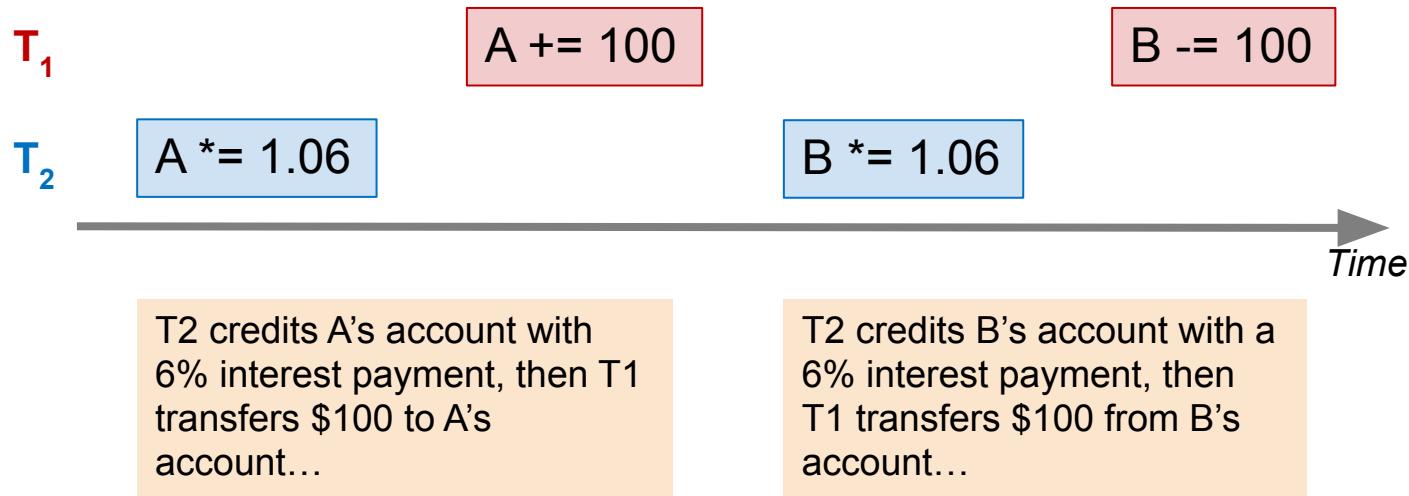
Example- consider two TXNs:

The TXNs could occur in either order... DBMS allows!



Example- consider two TXNs:

The DBMS can also **interleave** the TXNs



Interleaving & Isolation

- The DBMS has freedom to interleave TXNs
- However, it must pick an interleaving or schedule such that isolation and consistency are maintained
- **⇒ Must be as *if* the TXNs had executed serially!**

“With great power comes great responsibility”

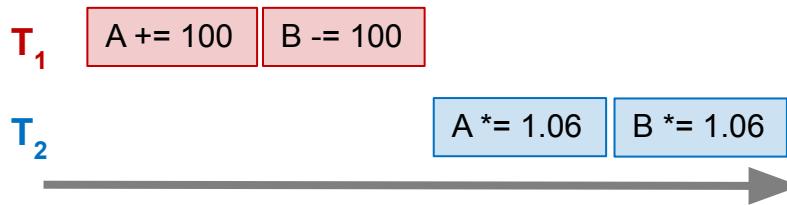
ACID

DBMS must pick a schedule which maintains isolation & consistency

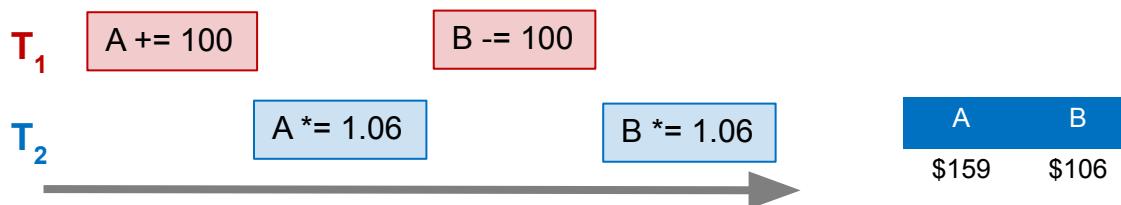
Scheduling examples

<i>Starting Balance</i>	A	B
	\$50	\$200

Serial schedule T_1, T_2 :



Interleaved schedule A:



Same
result!

Scheduling examples

Starting Balance	
A	B
\$50	\$200

Serial schedule T_1, T_2 :



A	B
\$159	\$106

Interleaved schedule B:



A	B
\$159	\$112

Different result than serial T_1, T_2 !

Scheduling examples

Serial schedule T_2, T_1 :

T_1

A += 100 B -= 100

T_2

A *= 1.06 B *= 1.06

Starting Balance	A	B
\$50	\$200	

A	B
\$153	\$112

Interleaved schedule B:

T_1

A += 100 B -= 100

T_2

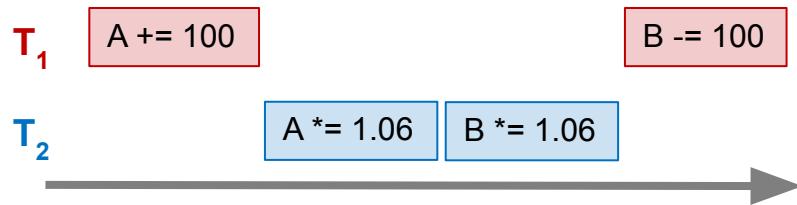
A *= 1.06 B *= 1.06

A	B
\$159	\$112

Different result than serial T_2, T_1
ALSO!

Scheduling examples

Interleaved schedule B:



This schedule is different than *any serial order!* We say that it is not serializable



Scheduling Definitions

- A **serial schedule** is one that does not interleave the actions of different transactions
- A and B are **equivalent schedules** if, ***for any database state***, the effect on DB of executing A **is identical** to the effect of executing B
- A **serializable schedule** is a schedule that is equivalent to ***some*** serial execution of the transactions.

The word “***some***” makes this definition powerful & tricky!

Serial Schedules

T1	A += 100	B -= 100	
T2			A *= 1.06 B*= 1.06

S1

T1			A += 100	B -= 100
T2		A *= 1.06	B *= 1.06	

S2

Interleaved Schedules

T1			A += 100	B -= 100
T2			A *= 1.06	B*= 1.06

S3

T1				A += 100	B -= 100
T2		A *= 1.06		B *= 1.06	

S4

T1					A += 100	B -= 100
T2		A *= 1.06				B*= 1.06

S5

T1			A += 100			B -= 100
T2			A *= 1.06	B *= 1.06		

S6

Serial Schedules	S1, S2
Serializable Schedules	S3, S4 (And S1, S2)
Equivalent Schedules	<S1, S3> <S2, S4>
Non-serializable (Bad) Schedules	S5, S6



What you will
learn about in
this section

1. Concurrency

- ▷ Interleaving & scheduling (Examples)
- ▷ Conflict & Anomaly types (Formalize)

2. Locking: 2PL, Deadlocks (Algorithm)



Conflicts and Anomalies



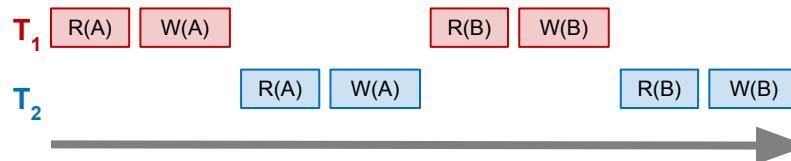
General DBMS model: Concurrency as Interleaving TXNs

Serial Schedule



Each action in the TXNs *reads a value from global memory and then writes one back to it* (e.g., $R(A)$ reads ‘A’)

Interleaved Schedule



For our purposes, having TXNs occur concurrently means **interleaving their component actions (R/W)**

We call the particular order of interleaving a **schedule**



Conflict Types

Two actions **conflict** if they are part of different TXNs, involve the same variable, and at least one of them is a write

Thus, there are three types of conflicts:

- Read-Write conflicts (RW)
- Write-Read conflicts (WR)
- Write-Write conflicts (WW)

Why no “RR Conflict”?

Note: **conflicts** happen often in many real world transactions. (E.g., two people trying to book an airline ticket)



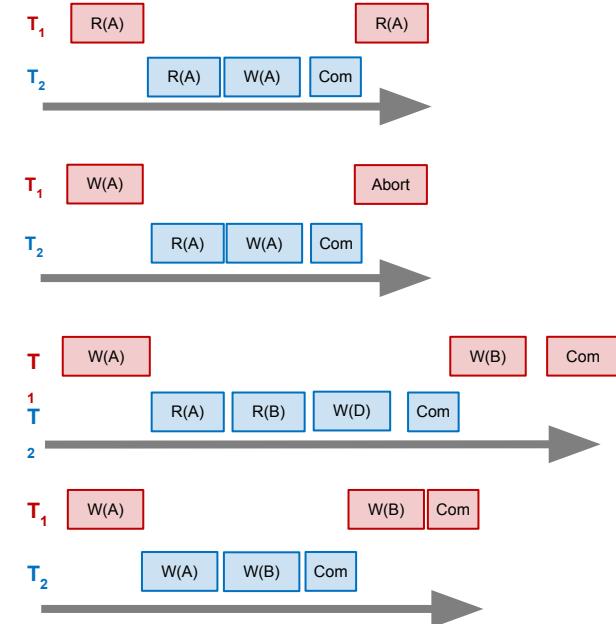
Classic Anomalies with Interleaved Execution

“Unrepeatable read”:

“Dirty read” / Reading uncommitted data:

“Inconsistent read” / Reading partial commits:

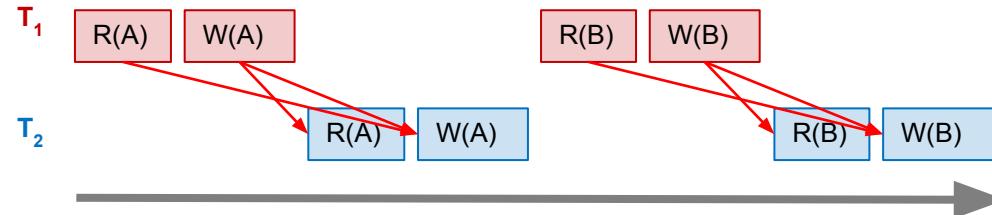
Partially-lost update:





Conflicts

Two actions **conflict** if they are part of different TXNs, involve the same variable, and at least one of them is a write



All “conflicts”!



Note: Conflicts vs. Anomalies

Conflicts are in both “good” and “bad” schedules
(they are a property of transactions)

Goal: Avoid Anomalies while interleaving transactions with conflicts!

- Do not create “bad” schedules where isolation and/or consistency is broken (i.e., Anomalies)



Conflict Serializability, Locking & Deadlock



Conflict Serializability

Two schedules are **conflict equivalent** if:

- Every *pair of conflicting actions* of TXNs are *ordered in the same way*
 - (And involve *the same actions of the same TXNs*)

Schedule S is **conflict serializable** if S is *conflict equivalent* to some serial schedule

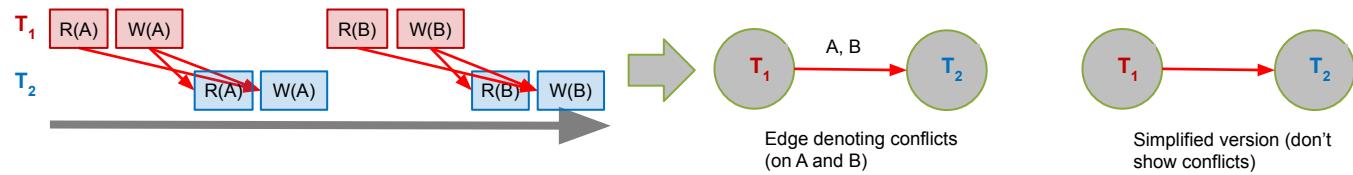
Conflict serializable \Rightarrow serializable

So if we have conflict serializable, we have consistency & isolation!



The Conflict Graph

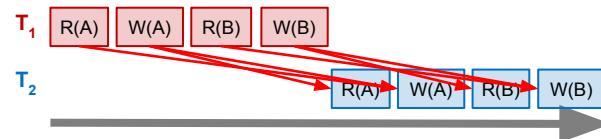
- Let's now consider looking at conflicts **at the TXN level**
- Consider a graph where the **nodes are TXNs**, and there is an edge from $T_i \rightarrow T_j$ if **any actions in T_i precede and conflict with any actions in T_j**



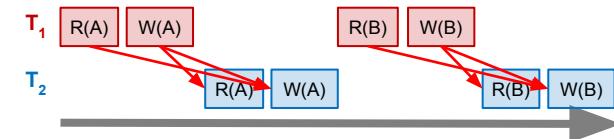


What can we say about “good” vs. “bad” conflict graphs?

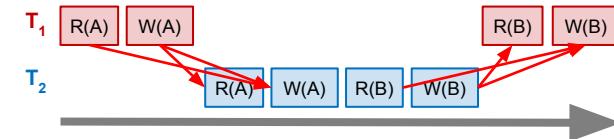
Serial Schedule:



Interleaved Schedules:



A bit complicated...



Conflict serializability provides us with an operative notion of “good” vs. “bad” schedules! “Bad” schedules create data Anomalies



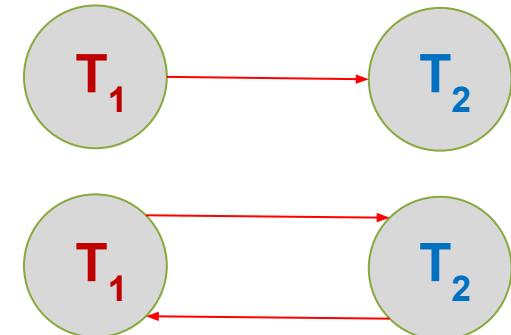
What can we say about “good” vs. “bad” conflict graphs?

Serial Schedule:



Simple!

Interleaved Schedules:



Theorem: Schedule is **conflict serializable** if and only if its conflict graph is acyclic



Connection to conflict serializability

- In the conflict graph, a **topological** (sort) ordering of nodes corresponds to a **serial ordering of TXNs**

Theorem: Schedule is **conflict serializable** if and only if its conflict graph is **acyclic**

Example with 5 Transactions

Given: Schedule S1

	w1(A)	r2(A)	w1(B)	w3(C)	r2(C)	r4(B)	w2(D)	w4(E)	r5(D)	w5(E)
--	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

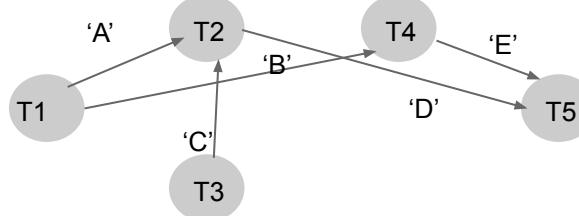
E.g. w3(C) is short for "T3 Writes on C"

Good or Bad schedule?
Conflict serializable?

T1	w1(A)		w1(B)							
T2		r2(A)			r2(C)		w2(D)			
T3				w3(C)						
T4						r4(B)		w4(E)		
T5									r5(D)	w5(E)

Step1

Find conflicts
(RW, WW, WR)



Acyclic
⇒ Conflict serializable!
⇒ Serializable

Step2

Build Conflict graph
Acyclic? Topo Sort

Step3

Example serial schedules
Conflict Equiv to S1

	T3	T1	T4	T2	T5		SerialSched (SS1)
	w3(C)	w1(A)	w1(B)	r4(B)	w4(E)	r2(A)	w2(D) r5(D) w5(E)
	T1	T3	T2	T4	T5		SerialSched (SS2)

	T3	T1	T4	T2	T5		SerialSched (SS1)
	w1(A)	w1(B)	w3(C)	r2(A)	r2(A)	w2(D)	w4(E) r5(D) w5(E)
	T1	T3	T2	T4	T5		SerialSched (SS2)

Summary

- Concurrency achieved by **interleaving TXNs** such that **isolation & consistency** are maintained
 - We formalized a notion of **serializability** that captured such a “good” interleaving schedule
 - We defined **conflict serializability**



Put all this
machinery
together NEXT
week

Quick intuition for use cases ?

1. Construction

Locking algorithms to produce good schedules

2. Optimization

Optimizer may take a schedule and reorder (if disk is slow, etc.)



Strict 2PL: One Simple Locking algorithm

(now that we understand properties of schedules we want)

Big Idea

LOCKS!

Big Idea: LOCKs

- ▷ Intuition:
 - 'Lock' each record for shortest time possible
 - (e.g, Locking Money Table for a day is not good enough)
- ▷ Key questions:
 - Which records? For how long? What's algorithm?



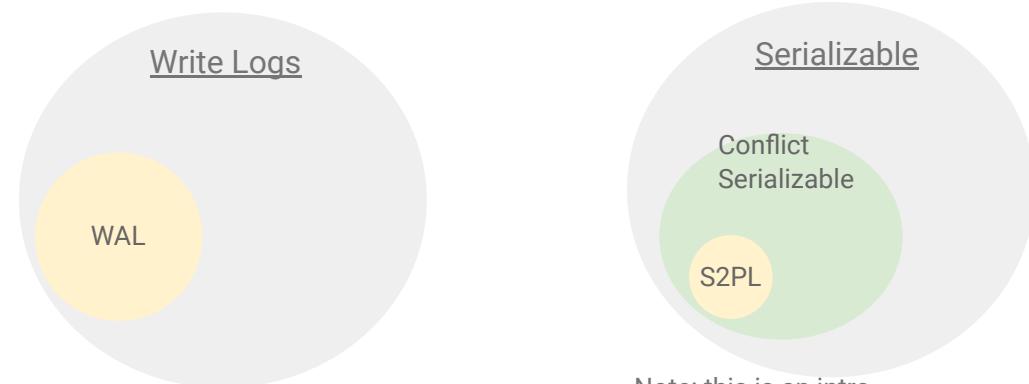
We now have the tools to BUILD such locks. Next week!

Plan for
Today

Summary

Strict 2 PL (S2PL) Locking

Putting it all together -- ACID Transactions



Note: this is an intro
Next: Take 245/346 (Distributed Transactions) or read
[Jim Gray's](#) classic



Strict Two-phase Locking (S2PL) Protocol

TXNs obtain:

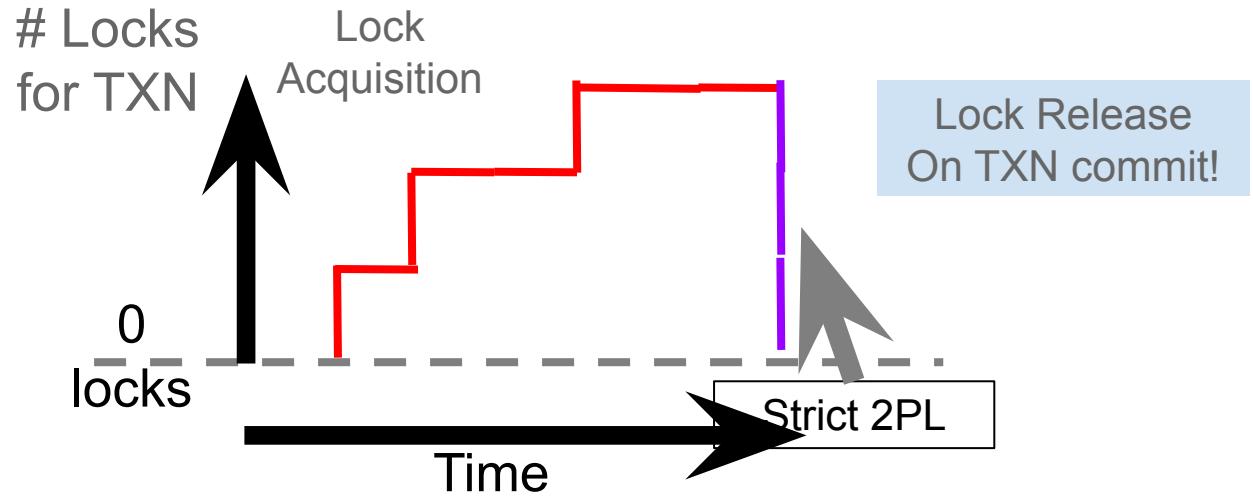
1. An **X (exclusive) lock** on object before **writing**.
⇒ No other TXN can get a lock (S or X) on that object.
(e.g., X('A') is an exclusive lock on 'A')

2. An **S (shared) lock** on object before **reading**
⇒ No other TXN can get an X lock on that object

All locks held by a TXN are released when TXN completes.



Strict 2-Phase Locking (S2PL)



2-Phase Locking: A transaction can not request additional locks once it releases any locks. [Phase1: “growing phase” to get more locks. Phase2: “shrinking phase”]

Strict 2-PL: Release locks only at COMMIT (COMMIT Record flushed) or ABORT



Strict 2PL

S2PL produces **conflict serializable** schedules...

- ...and thus serializable
- ...and we get isolation & consistency!

Popular implementation

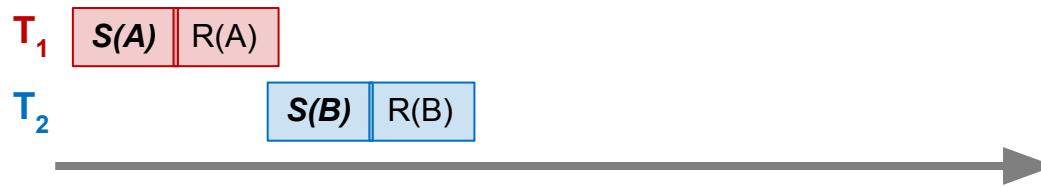
- Simple !
- Produces subset of *all* conflict serializable schedules
- There are MANY more complex LOCKING schemes with better performance. (See CS 245/ CS 345)
- One key, subtle problem (next)

Example: Deadlock Detection



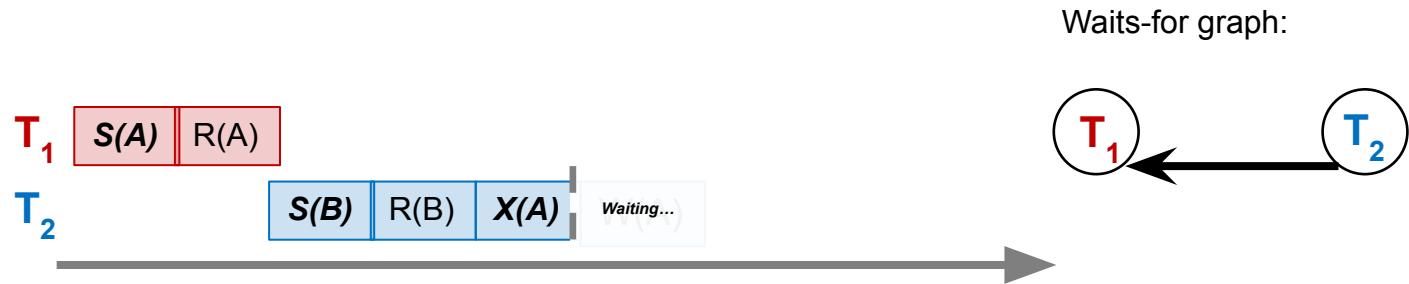
First, T_1 requests a shared lock
on A to read from it

Deadlock Detection: Example



Next, T_2 requests a shared lock
on B to read from it

Deadlock Detection: Example

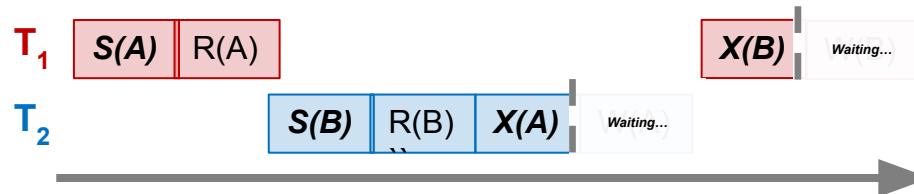


T₂ then requests an exclusive lock on A to write to it- **now T₂ is waiting on T₁...**

Waits-For graph: Track which Transactions are waiting

IMPORTANT: WAITS-FOR graph different than CONFLICT graph we learnt earlier !

Deadlock Detection: Example



Waits-for graph:



Cycle =
DEADLOCK

Finally, T_1 requests an exclusive lock on B to write to it- now T_1 is waiting on T_2 ... DEADLOCK!



Deadlocks

Deadlock: Cycle of transactions waiting for locks to be released by each other.

Two ways of dealing with deadlocks:

1. Deadlock prevention
2. Deadlock detection



Deadlock Detection

Create the **waits-for graph**:

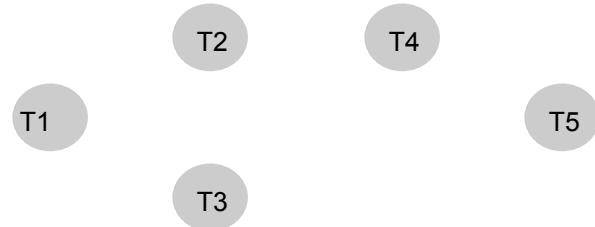
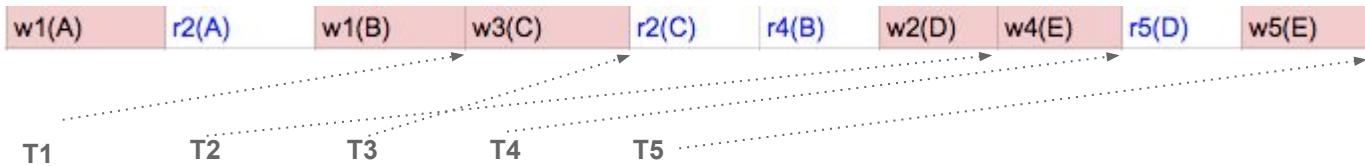
- Nodes are transactions
- There is an edge from $T_i \rightarrow T_j$ if T_i is *waiting for T_j to release a lock*

Periodically check for (**and break**) cycles in the waits-for graph

Example with 5 Transactions (S2PL)

Given: Schedule S1

Execute with S2PL



Waits- For Graph

Example with 5 Transactions (S2PL)

Given: Schedule S1

Execute with S2PL

Step 0

X (A)
w1(A)

Req S(A)

Step 1

X (B)
w1(B)
Unl B, A

Step 2

Get S(A)
r2(A)

Step 3

X (C)
w3(C)
Unl C

Step 4

S(C)
r2(C)

Step 5

S(B)
r4(B)

Step 6

X(D)
w2(D)
Unl A, C, D

Step 7

X(E)
w4(E)
Unl B, E

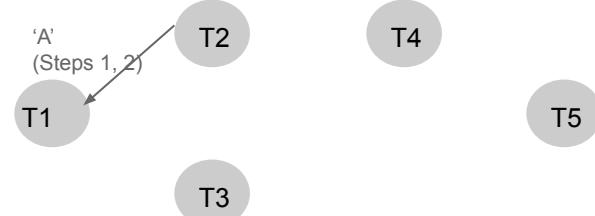
Step 8

S (D)
r5(D)

Step 9

X (E)
w5(E)
Unl D, E

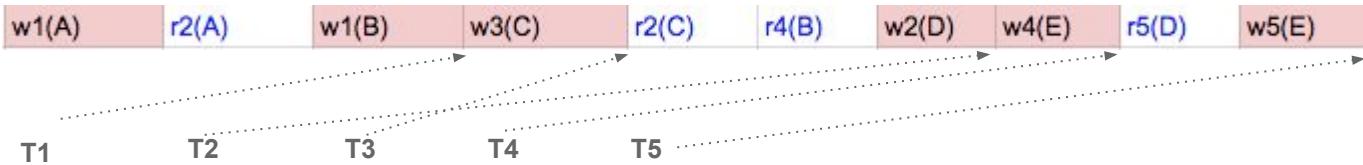
Step 10



Waits-For Graph

Example with 5 Transactions (S2PL)

Schedule S1



Execute with S2PL

Step 0

X (A)
w1(A)

Step 1

X (B)
w1(B)

Unl B, A

Step 2

Step 3

Step 4

Step 5

Step 6

Step 7

Step 8

Step 9

Step 10

X(D)
w2(D)

Unl A, C, D

S(B)
r4(B)

X(E)

w4(E)

Unl B, E

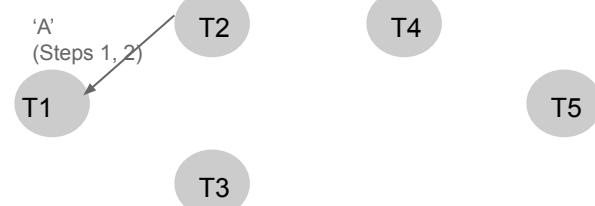
S (D)

r5(D)

X (E)

w5(E)

Unl D, E



Waits-For Graph

Example: What happened?

Input Schedule

w1(A)	r2(A)	w1(B)	w3(C)	r2(C)	r4(B)	w2(D)	w4(E)	r5(D)	w5(E)
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

S2PL's Reordered Schedule Executed

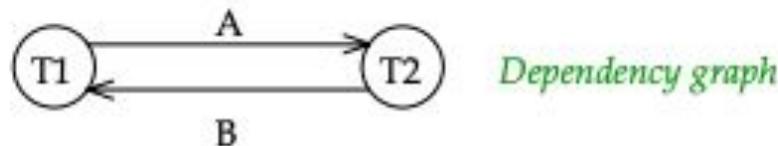
w1(A)	w1(B)	r2(A)	w3(C)	r2(C)	r4(B)	w2(D)	w4(E)	r5(D)	w5(E)
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

In general, given a set of TXNs, i.e., schedules
S2PL() produces conflict-serializable schedules.

Example2

- A schedule that is not conflict serializable:

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



If you Input above schedule into S2PL(), what would happen?

[Ans: R2(A) blocked until after W1(B). Therefore, conflict serialized. i.e. $T1 \rightarrow T2$]

Focus for cs145

Strict 2PL vs 2PL?

- 2PL releases locks faster than S2PL
- Pros: higher performance (e.g., for the monthly Bank Interest updating TXN, S2PL will hold all LOCKs till end of TXN, 2PL will release faster).
- Cons: subtle problems which Strict 2PL gets around by waiting to release locks (read: cascading rollbacks after class)

For cs145 in Fall'21,

- Focus on Strict 2PL for our tests, homeworks

Recall

Quick intuition for use cases ?

1. Construction

Locking algorithms to produce good schedules

2. Optimization

Optimizer may take a schedule and reorder (if disk is slow, etc.)

Transactions

Summary

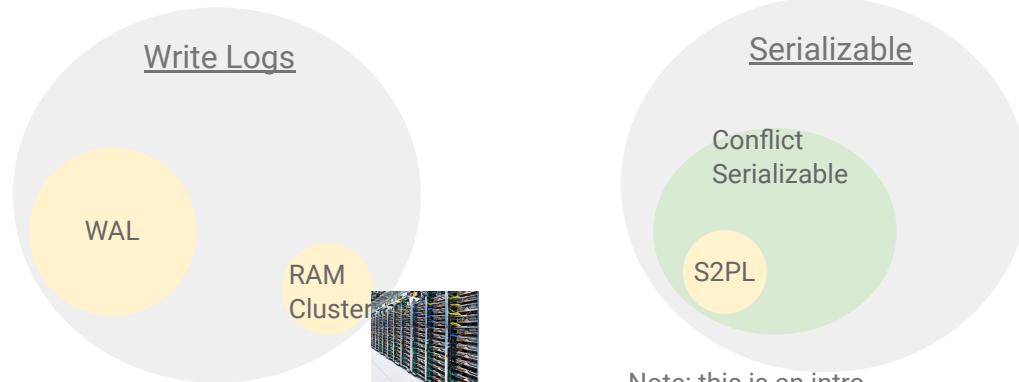
Why study Transactions?

Good programming model for parallel applications on shared data !

- Atomic
- Consistent
- Isolation
- Durable

Design choices?

- Write update Logs (e.g., WAL logs)
- Serial? Parallel, interleaved and serializable?



Note: this is an intro
Next: Take 346 (Distributed Transactions) or read [Jim Gray's](#) classic



Putting it all together

Example

Monthly bank interest transaction

Money

Account	Balance (\$)
3001		500
4001		100
5001		20
6001		60
3002		80
4002		-200
5002		320
...	...	
30108		-100
40008		100
50002		20

Money (@4:29 am day+1)

Account	Balance (\$)
3001		550
4001		110
5001		22
6001		66
3002		88
4002		-220
5002		352
...	...	
30108		-110
40008		110
50002		22



'T-Monthly-423'

Monthly Interest 10%

4:28 am Starts run on 10M bank accounts
Takes 24 hours to run

```
UPDATE Money  
SET Balance = Balance * 1.1
```

Other Transactions

10:02 am Acct 3001: Wants 600\$
11:45 am Acct 5002: Wire for 1000\$
.....
.....

2:02 pm Acct 3001: Debit card for \$12.37

Q: How do I not wait for a day to access \$\$\$s?

Transactions

Summary

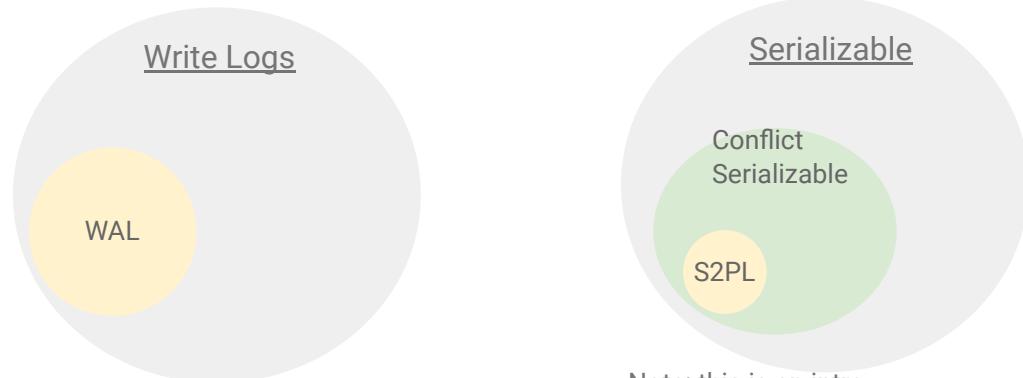
Why study Transactions?

Good programming model for parallel applications on shared data !

Atomic
Consistent
Isolation
Durable

Design choices?

- Write update Logs (e.g., WAL logs)
- Serial? Parallel, interleaved and serializable?



Note: this is an intro

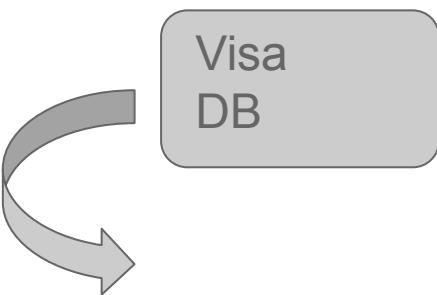
Next: Take 245/346 (Distributed Transactions) or read
[Jim Gray's](#) classic

Example Visa DB



Transaction Queue

- 60000 user TXNs/sec
- Monthly 10% Interest TXN

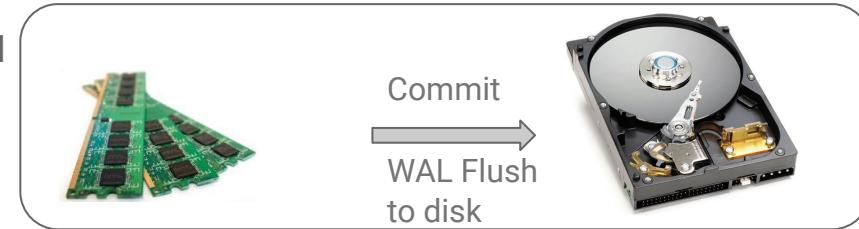


Account	...	Balance (\$)
3001		500
4001		100
5001		20
6001		60
3002		80
4002		-200
5002		320
...
30108		-100
40008		100
50002		20

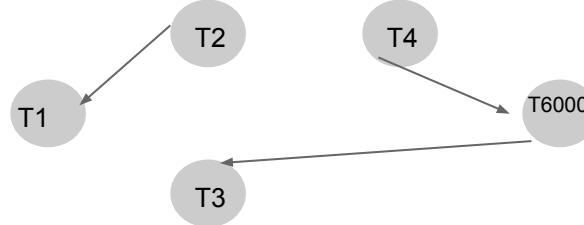
Design#1 VisaDB

For each Transaction in Queue

- For relevant records
 - Use **2PL** to acquire/release locks
 - Read, Process, Write records
 - **WAL** Logs for updates
- Commit or Abort



Example Waits-For Graph



Example WAL Logs

- for 'T-Monthly-423'

WAL (@4:29 am day+1)

Update
Records

Commit
Record

T-Monthly-423	START TRANSACTION		
T-Monthly-423	3001	500	550
T-Monthly-423	4001	100	110
T-Monthly-423	5001	20	22
T-Monthly-423	6001	60	66
T-Monthly-423	3002	80	88
T-Monthly-423	4002	-200	-220
T-Monthly-423	5002	320	352
T-Monthly-423
T-Monthly-423	30108	-100	-110
T-Monthly-423	40008	100	110
T-Monthly-423	50002	20	22
T-Monthly-423	COMMIT		



Write-Ahead Logging (WAL)

Algorithm: WAL

For each tuple update, **write Update Record** into LOG-RAM

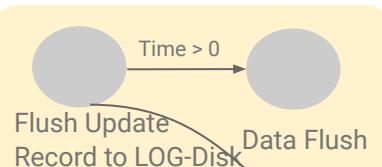
Follow two **Flush** rules for LOG

- Rule1: **Flush Update Record into LOG-Disk before** corresponding data page goes to storage
- Rule2: Before TXN commits,
 - **Flush all Update Records** to LOG-Disk
 - **Flush COMMIT Record** to LOG-Disk

→ **Durability**

→ **Atomicity**

Transaction is committed *once COMMIT record is on stable storage*

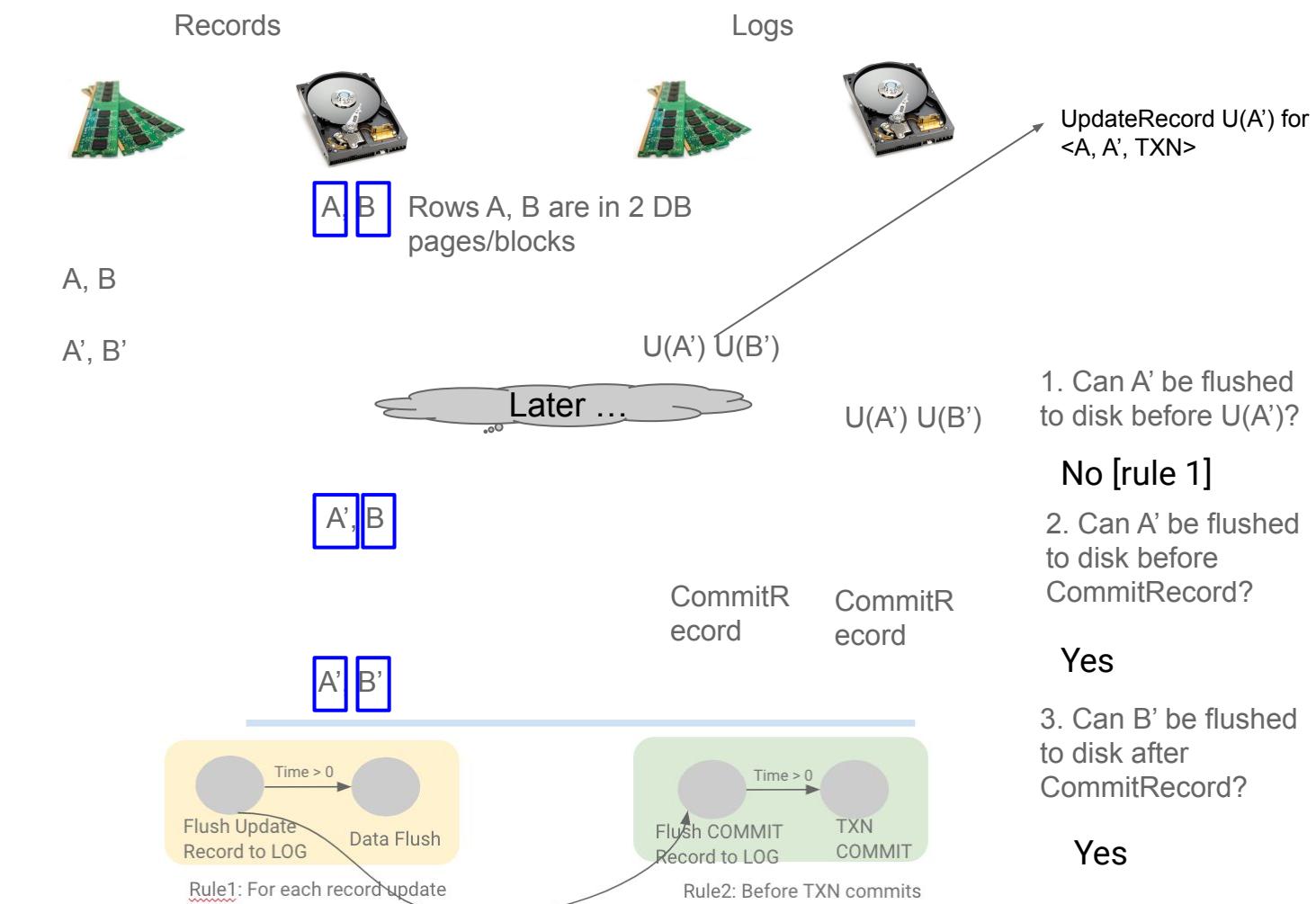


Rule1: For each tuple update



Rule2: Before TXN commits

Example WAL Sequence



Example Visa DB -- Need Higher Performance?



Transaction Queue

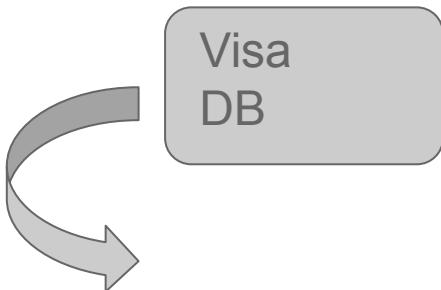
- 60000 TXNs/sec
- Monthly Interest TXN

'T-Monthly-423'

Monthly Interest 10%

4:28 am Starts run on 10M visa accounts

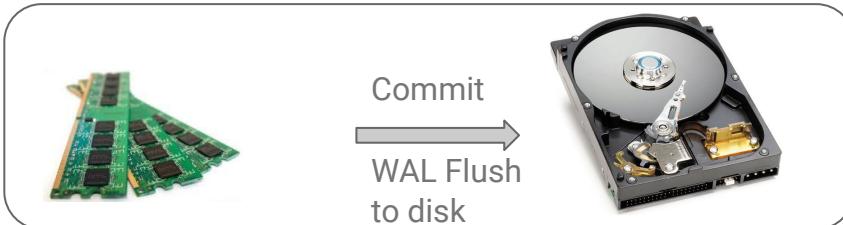
Takes 24 hours to run



Design#2 VisaDB

For each Transaction in Queue

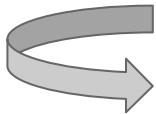
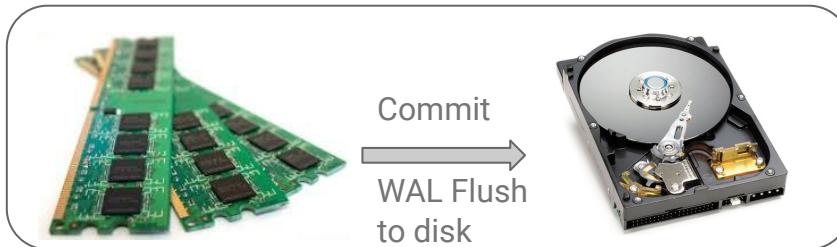
- For relevant records
 - Use 2PL to acquire/release locks
 - Process record
 - WAL Logs for updates
- Commit or Abort



Replace with more sophisticated algorithms
(cs245/cs345)

LOG on Cluster model

A popular alternative (with tradeoffs)



Commit by replicating log and/or data to 'n' other machines (e.g. $n = 2$)

[On same rack, different rack or different datacenter]

Example

Cluster LOG model

Performance

Failure model

Main model: RAM could fail, Disk is durable

VS

Cluster LOG model:

RAM on different machines don't fail at same time
Power to racks is uncorrelated

Incremental cost to write to machine

Network speeds intra-datacenter could be 1-10 microsecs

[Lazily update data on disk later, when convenient]



Example: Youtube DB

YouTube

funny cats

About 12,100,000 results

How to Get Rid of Cat Pee Stains
Ad BISSELL • 2M views
Your cat had an accident on the carpet. BISSELL is here to help!

CATS make us LAUGH ALL THE TIME! - Ultra FUNNY CAT
Tiger FunnyWorks 100K views • 3 days ago
Ultra funny cats and kitten that will make you cry with laughter! Cats are the best laugh all the time! This is ...

You will LAUGH SO HARD that YOU WILL FAINT - FUNNY C compilation
Tiger FunnyWorks 19M views • 8 months ago
Well well well, cats for you again. But this time, even better, even funnier, even more like these furries the ...

Have you EVER LAUGHED HARDER? - Ultra FUNNY CATS
Tiger FunnyWorks 124K views • 1 week ago
Super funny cats and kitten that will make you scream with laughter! This is the LAUGH challenge ever!

Baby Cats - Funny and Cute Baby Cat Videos Compilation (2018) Gatitos Bebes Video Recopilacion
809,337 views
Animal Planet Videos Published on May 23, 2018
SUBSCRIBE 337K

Upload video
Go live

Shop Now >> DressLibs

Up next
Top Cats Vs. Cucumbers Funny Cat Videos Compilation
Animal Planet Videos 23M views

Funny Elias play with the wheel on the bus and another toys ...
Elias Adventures 291 watching

LIVE! Rescue kitten nursery!
TinyKittens.com

Puss in boots and the three diablos [HD]
Mathew Garcia 7.6M views

YouTube

cats funny

YouTube

0:34 / 15:25

4.7K 768 SHARE

Example

Youtube writes

Performance

Example: Increment number of LIKES per Video

Design 1: WAL Log for Video Likes
<videoid, like count, new like count>

WAL for Video likes

T-LIKE-4307	START TRANSACTION		
T-LIKE-4307	3001	537	538
COMMIT			
T-LIKE-4308 START TRANSACTION			
T-LIKE-4308	5309	10001	10002
COMMIT			
T-LIKE-4309 START TRANSACTION			
T-LIKE-4309	3001	538	539
COMMIT			
...
T-LIKE-4341	5309	10002	10003
T-LIKE-4351			
T-LIKE-4351	5309	10003	10004
...
T-LIKE-4383 START TRANSACTION			
T-LIKE-4383	5309	10004	10005
COMMIT			

Critique?

Correct?

Write Speed? Cost? Storage?

Bottlenecks?



Baby Shark Dance | #babysong Most Viewed Video | Animal Songs | PINKFONG Songs for Children

9,670,252,313 views • Jun 17, 2016

31M 14M SHARE

Example

Youtube writes

Performance

Design 2:

- Replicate Like count in cluster.
- Batch updates in Log (e.g. batch increase counts by 100 or 1000 to amortize writes, based on video popularity)

Update RAM on
n=3 machines
(`<videoid, #likes>`)



Critique?

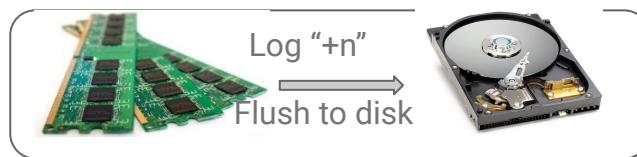
Correct?

Write Speed? Cost? Storage?

Bottlenecks?

System recovery?

Micro-batch updates		
Txn (e.g. Timestamp)	Videoid	Batch Increment
1539893189	3001	100
1539893195	5309	5000
1539893225	3001	200
	..	400
	5309	100
	...	5000
	5309	100000
	...	10
1539893289	5309	10



Example

Youtube writes

Performance

Vs Cost

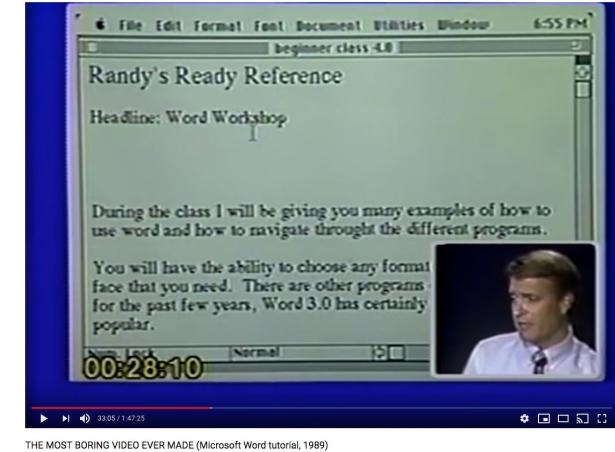
Design #3

For most videos, Design 1 (full WAL logs)

For popular videos, Design 2



Popular video



Unpopular video

Critique?

Correct?

Write Speed? Cost? Storage?
Bottlenecks?
System recovery?

Summary

Design Questions?

Correctness: Need true ACID? Pseudo-ACID? What losses are OK?

Design parameters:

Any data properties you can exploit? (e.g., '+1', popular vs not)

How much RAM, disks and machines?

How many writes per sec?

How fast do you want system to recover?

Choose: WAL logs, Replication on n-machines, Hybrid? (More in cs345)

Transactions

Summary

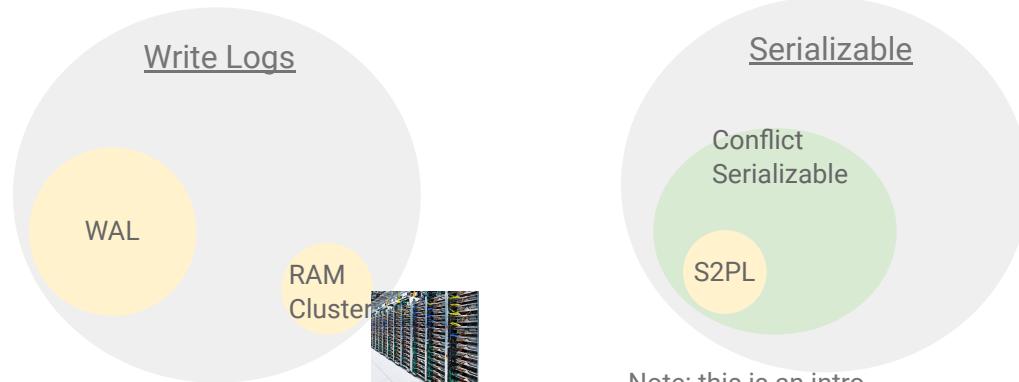
Why study Transactions?

Good programming model for parallel applications on shared data !

- Atomic
- Consistent
- Isolation
- Durable

Design choices?

- Write update Logs (e.g., WAL logs)
- Serial? Parallel, interleaved and serializable?



Note: this is an intro
Next: Take 346 (Distributed Transactions) or read [Jim Gray's](#) classic