

Administrivia

HW4 due today

Project 2 due 5/9, no late days.

Why? 5/9 is the most time we can give you.
Grades due to registrar 5/13!

Exam 2: 5/13 10:10AM-12:40PM.

Same format as Exam 1

SQLite >

Analyzing iMessage with SQL



<https://arctype.com/blog/search-imessage/>

SQLite >

Analyzing iMessage with

message

ROWID	INTEGER
guid	TEXT
text	TEXT
replace	INTEGER
service_center	TEXT
handle_id	INTEGER
subject	TEXT
country	TEXT
attributedBody	BLOB
version	INTEGER
type	INTEGER
service	TEXT
account	TEXT
account_guid	TEXT
error	INTEGER
date	INTEGER
date_read	INTEGER
date_delivered	INTEGER
is_delivered	INTEGER
is_finished	INTEGER
is_emote	INTEGER
is_from_me	INTEGER
is_empty	INTEGER
is_delayed	INTEGER
is_auto_reply	INTEGER
is_prepared	INTEGER
is_read	INTEGER
is_system_message	INTEGER
is_sent	INTEGER
has_dd_results	INTEGER

handle

ROWID	INTEGER
id	TEXT
country	TEXT
service	TEXT
uncanonicalized_id	TEXT
person_centric_id	TEXT

chat_message_join

chat_id	INTEGER
message_id	INTEGER
message_date	INTEGER

chat_handle_join

chat_id	INTEGER
handle_id	INTEGER

chat

ROWID	INTEGER
guid	TEXT
style	INTEGER
state	INTEGER
account_id	TEXT
properties	BLOB
chat_identifier	TEXT
service_name	TEXT

is_service_message	INTEGER
is_forward	INTEGER
was_downgraded	INTEGER
is_archive	INTEGER
cache_has_attachme...	INTEGER
cache_roomnames	TEXT
was_data_detected	INTEGER
was_deduplicated	INTEGER
is_audio_message	INTEGER
is_played	INTEGER
date_played	INTEGER
item_type	INTEGER
other_handle	INTEGER
group_title	TEXT
group_action_type	INTEGER
share_status	INTEGER
share_direction	INTEGER
is_expirable	INTEGER
expire_state	INTEGER
message_action_type	INTEGER
message_source	INTEGER
associated_message...	STRING
balloon_bundle_id	STRING
payload_data	BLOB
associated_message...	INTEGER
expressive_send_st...	STRING
associated_message...	INTEGER
associated_message...	INTEGER
time_expressive_se...	INTEGER
message_summary_in...	BLOB
ck_sync_state	INTEGER
ck_record_id	TEXT
ck_record_change_t...	TEXT

service_name	TEXT
room_name	TEXT
account_login	TEXT
is_archived	INTEGER
last_addressed_han...	TEXT
display_name	TEXT
group_id	TEXT
is_filtered	INTEGER
successful_query	INTEGER
engram_id	TEXT
server_change_token	TEXT
ck_sync_state	INTEGER
last_read_message...	INTEGER
ck_record_system_p...	BLOB
original_group_id	TEXT
sr_server_change_t...	TEXT
sr_ck_sync_state	INTEGER
cloudkit_record_id	TEXT
sr_cloudkit_record...	TEXT
last_addressed_sim...	TEXT
is_blackholed	INTEGER
syndication_date	INTEGER
syndication_type	INTEGER

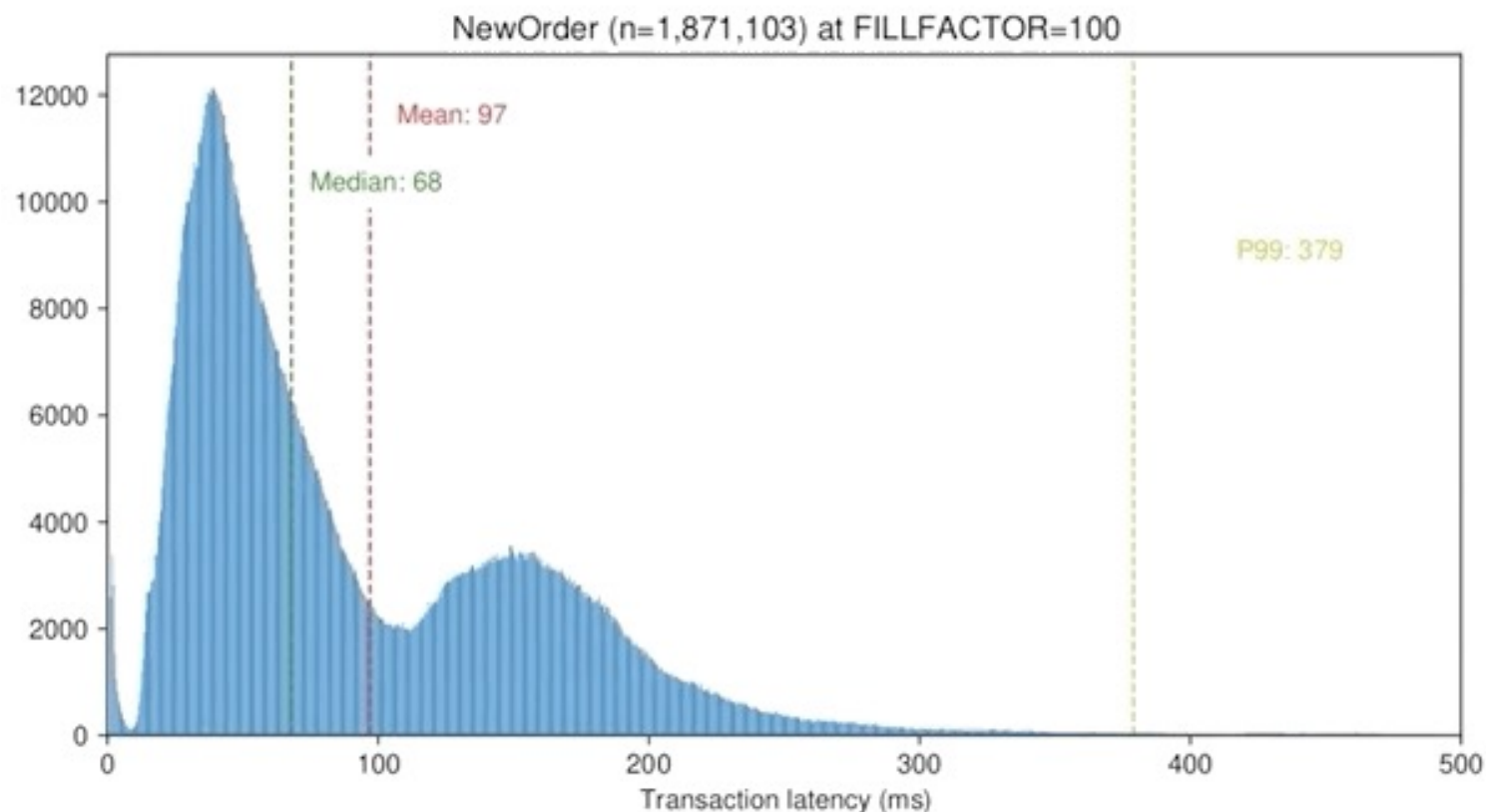
<https://arctype.com/blog/search-imessage/>

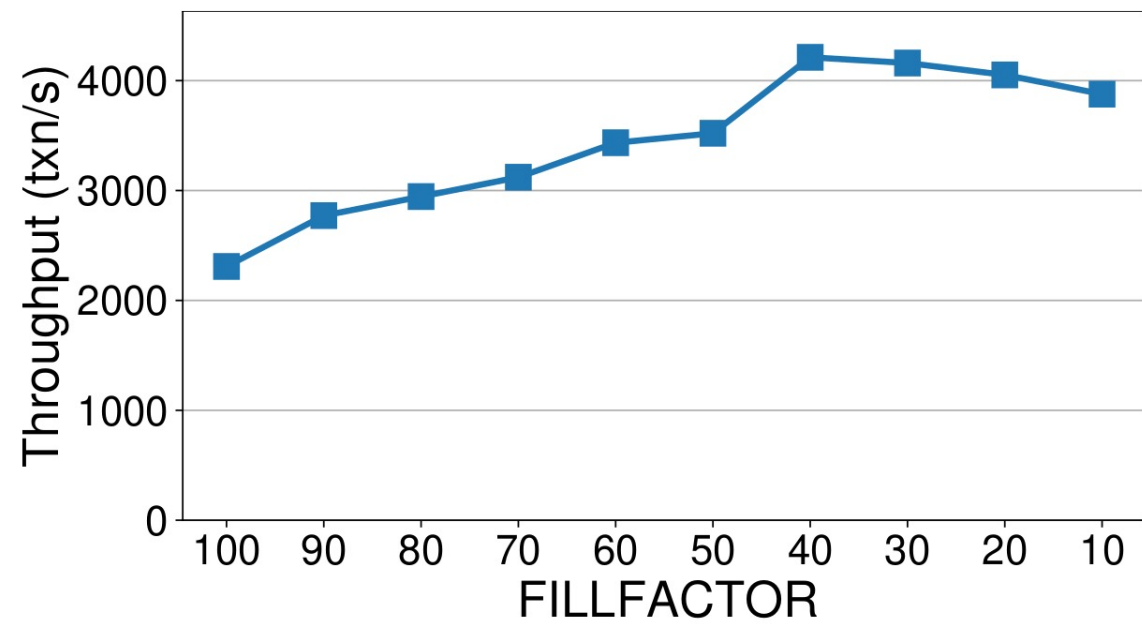
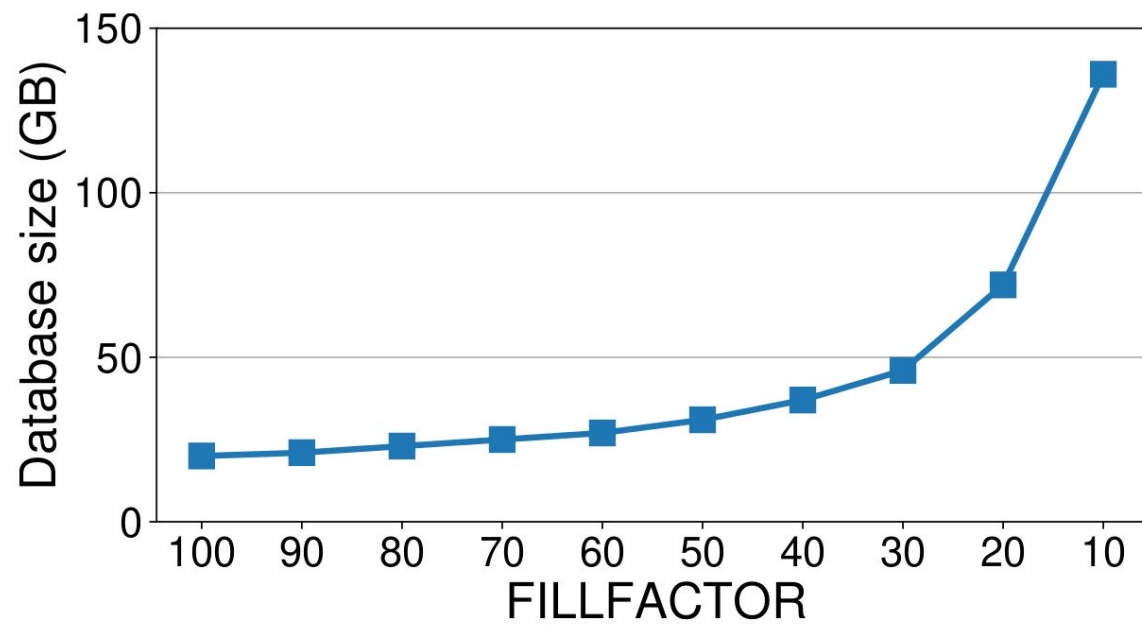


Matt Butrovich

@butro

We've been playing with [@PostgreSQL](#)'s fillfactor for tables. This is a hard setting to get right because it depends on workload patterns. This graph shows TPC-C NewOrder latency as you change fillfactor on 5 tables. p99 gets tighter. Not sure of bimodal pattern at fillfactor=100.





L10

Transactions, Concurrency, Recovery

Eugene Wu

Overview

Why do we want transactions?

What guarantees do we want from transactions?

Why Transactions?

Concurrency (for performance)

N clients, no concurrency

1st client runs fast

2nd client waits a bit

3rd client waits a bit longer

Nth client walks away

N clients, concurrency

client 1 runs $x += y$

client 2 runs $x -= y$

what happens?

Can we prevent stepping on toes? *Isolation*

User 1

$x \ += \ y$

$a1 = \text{read}(x)$

$b1 = \text{read}(y)$

$\text{store}(a1 + b1)$

User 2

$x \ -= \ y$

$a2 = \text{read}(x)$

$b2 = \text{read}(y)$

$\text{store}(a2 - b2)$

Good Execution Order

$a1 = \text{read}(x)$	// $x=0$
$b1 = \text{read}(y)$	// $y=1$
$\text{store}(a1 + b1, x)$	// $1 \rightarrow x$
$a2 = \text{read}(x)$	// $x=1$
$b2 = \text{read}(y)$	// $y=1$
$\text{store}(a2 - b2, x)$	// $0 \rightarrow x$

result:

$x=0$

User 1

$x \ += \ y$

$a1 = \text{read}(x)$

$b1 = \text{read}(y)$

$\text{store}(a1 + b1)$

User 2

$x \ -= \ y$

$a2 = \text{read}(x)$

$b2 = \text{read}(y)$

$\text{store}(a2 - b2)$

Bad Execution Order

$a1 = \text{read}(x)$	// $x=0$
$a2 = \text{read}(x)$	// $x=0$
$b2 = \text{read}(y)$	// $y=1$
$\text{store}(a2 - b2, x)$	// $-1 \rightarrow x$
$b1 = \text{read}(y)$	// $y=1$
$\text{store}(a1 + b1, x)$	// $1 \rightarrow x$

result:

$x=1$

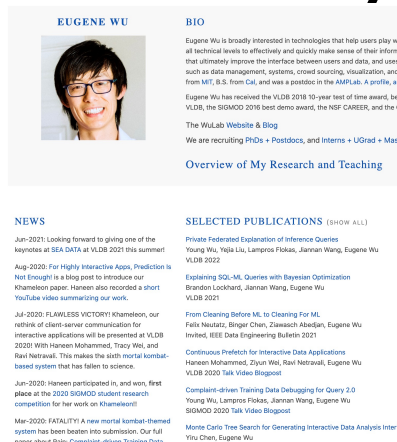
Who... would ever do this?

Real \$1B+ Companies...

Store extracted data in a file

Every change → rewrite the file

Text extraction and transform tasks



EUGENE WU

BIO

Eugene Wu is broadly interested in technologies that help users play with all technical levels to effectively and quickly make sense of their information that ultimately improve the interface between users and data, and uses such as data management, systems, crowd sourcing, visualization, and from MIT, B.S. from Cal, and was a postdoc in the AMPLab. A profile, an Eugene Wu has received the VLDB 2018 10-year test of time award, the VLDB, the SIGMOD 2018 best demo award, the NSF CAREER, and the G

The WuLab Website & Blog

We are recruiting PhDs • Postdocs, and interns • UGrad • Mast

[Overview of My Research and Teaching](#)

NEWS

Jun-2021: Looking forward to giving one of the keynote at SIGMOD at VLDB 2021 this summer!

Aug-2020: Four Highly Interactive Apps, Prediction is Not Enough! is a blog post to introduce our Khamaleon paper. Hansen also recorded a short YouTube video summarizing our work.

Jul-2020: FLAWLESS VICTORY! Khamaleon, our rethink of client-server communication for interactive applications will be presented at VLDB 2020! With Hansen Mohammed, Tracy Wei, and Raul Nallawala. This makes the sixth mortal combat-based system that has fallen to science.

Jun-2020: Hansen participated in, and won, first place at the 2020 SIGMOD student research competition for her work on Khamaleon!

Mar-2020: FATALITY! A new mortal combat-themed system has been beaten into submission. Our full paper about Fate: FATALITY-driven Traversal Trees

SELECTED PUBLICATIONS (SHOW ALL)

Private Federated Explanation of Inference Queries
Young Wu, Yiqi Liu, Lampson Fokas, Jianan Wang, Eugene Wu
VLDB 2022

Explaining SQL-ML Queries with Bayesian Optimization
Brandon Lockhart, Jianan Wang, Eugene Wu
VLDB 2021

From Clearing Before ML to Clearing For ML
Felix Neudatz, Bing Chen, Zsolt Abadi, Eugene Wu
Invited, IEEE Data Engineering Bulletin 2021

Continuous Prefetch for Interactive Data Applications
Hansen Mohammed, Ziyu Wei, Raul Nallawala, Eugene Wu
VLDB 2020 Talk Video Blogpost

Complaint-driven Training Data Debugging for Query 2.0
Young Wu, Lampson Fokas, Jianan Wang, Eugene Wu
SIGMOD 2020 Talk Video Blogpost


Monte Carlo Tree Search for Generating Interactive Data Analysis Interf
Yiqi Chen, Eugene Wu

[{
obj: region
box: 36,34,100,120
text: "Private F..
}, { .. }, ..]

[{
obj: authorlist
box: 40,40,100,20
text: "Eugene W..
}, { .. }, ..]

Browser

New Tab



EUGENE WU

BIO

Eugene Wu is broadly interested in technologies that help users play with their data. His goal is for all technical levels to effectively and quickly make sense of their information. He is interested in solutions that ultimately improve the interface between users and data, and uses techniques borrowed from fields such as data management, systems, crowd sourcing, visualization, and HCI. Eugene Wu received his MSc from MIT, B.S. from Cal, and was a postdoc in the AMP-Lab. A profile, an obit.

Eugene Wu has received the VLDB 2018 10-year test of time award, best-of-conference citations at VLDB, the SIGMOD 2016 best demo award, the NSF CAREER, and the Google and Amazon faculty awards.

[The WuLab Website & Blog](#)

We are recruiting **PhDs + Postdocs**, and **Interns + UGrad + Masters!**

Overview of My Research and Teaching

NEWS

Jun-2021: Looking forward to giving one of the keynotes at SEA DATA at VLDB 2021 this summer!

Aug-2020: For [Highly Interactive Apps](#), Prediction is Not Enough! is a blog post to introduce our Kameleon program. Haneen also recorded a short YouTube video summarizing our work.

Jul-2020: [FLAWLESS VICTORY!](#) Kameleon, our

SELECTED PUBLICATIONS (click to view all)


[Privacy-Enforced Exploration of Inference Outcomes](#)

Your VLDB 2021

[Explaining SQL-ML Queries with Bayesian Optimization](#)

Brandon Lockhard, Jannan Wang, Eugene Wu

VLDB 2021



EUGENE WU

BIO

Eugene Wu is broadly interested in technologies that help users play with their data. His goal is for all technical levels to effectively and quickly make sense of their information. He is interested in solutions that ultimately improve the interface between users and data, and uses techniques borrowed from fields such as data management, systems, crowd sourcing, visualization, and HCI. Eugene Wu received his PhD from MIT, B.S. from Cal, and was a postdoc in the AMPLab. [A profile, an obit.](#)

Eugene Wu has received the VLDB 2018 10-year test of time award, best-of-conference citations at VLDB, the SIGMOD 2016 best demo award, the NSF CAREER, and the Google and Amazon faculty awards.

The WuLab Website & Blog

We are recruiting **PhDs + Postdocs**, **Interns + UGrad + Masters!**

Overview of My Research and Teaching

NEWS

Jan-2021: Looking forward to giving one of the keynotes at SEA DATA at VLDB 2021 this summer!

Aug-2020: For Highly Interactive Apps, Prediction Is Not Enough! is a blog post to introduce our Kameleon paper. Hansen also recorded a short YouTube video summarizing our work.

Jul-2020: FLAWLESS VICTORY! Kameleon, our

SELECTED PUBLICATIONS (SHOW ALL)

Private, Extended Explanation of Inference Queries
 Your
 VLDB

Explaining SQL-ML Queries with Bayesian Optimization
 Brandon Lockhard, Jiaman Wang, Eugene Wu
 VLDB 2021

Annotation tasks



EUGENE WU

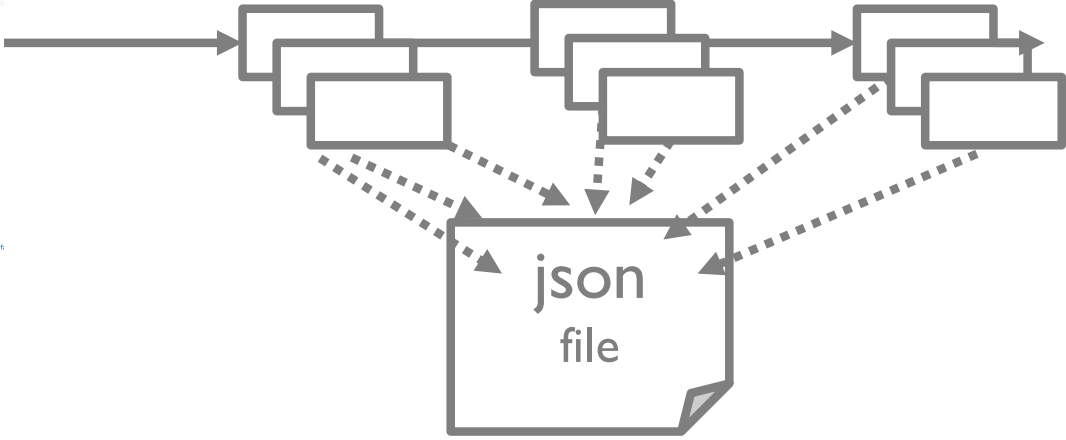
BIO

Eugene Wu is broadly interested in technologies that help users play well with technical tools to effectively and rapidly make sense of their informatics that ultimately improve the interface between users and data, and uses such as data management, systems, cloud sourcing, visualization, and from MIT, U.S. from Cal, and was a postdoc in the AMP-Lab, a profile, Eugene Wu has received the VLDB 2018 10-year best of time award, best VLDB, the SIGMOD 2016 best demo award, the NSF CAREER, and the C

The WuLab Website & Blog

We are recruiting PhDs + Postdocs, and Interns + UGrad + Master

Overview of My Research and Teaching



Why Transactions?

What about 1 client, no concurrency?

Client runs big update query

`update set x += y`

Power goes out

What is the state of the database?

Why Transactions?

What about 1 client, no concurrency?

Client runs big update query

update set $x += y$

Aborts the query (e.g., ctrl-c)

What is the state of the database?

If an abort happens, can the database recover to something sensible? *Atomicity, Durability*

What does “sensible” mean?

Transactions = r/w over objects

Transaction: a sequence of actions

action = read object, write object, commit, abort

API between app semantics and DBMS's view

From app/user' point of view, the transaction (and its effects) are only “correct” once the DBMS has told the app/user that the transaction is COMMITed

Transactions = r/w over objects

Transaction: a sequence of actions

action = read object, write object, commit, abort

API between app semantics and DBMS's view

User's view

T1: begin $A = A + 100$ $B = B - 100$ END

T2: begin $A = 1.5 * A$ $A = 1.5 * B$ END

DBMS's logical view

T1: begin $r(A)$ $w(A)$ $r(B)$ $w(B)$ END

T2: begin $r(A)$ $w(A)$ $r(B)$ $w(A)$ END

ACID: Transaction Guarantees

A

Atomicity

users never see in-between xact state.
only see a xact's effects once it's committed
as if a xact runs instantaneously

C

Consistency

database always satisfies ICs.
xacts move from valid database to valid database

I

Isolation:

from xact's point of view, it's the only xact running

D

Durability:

if xact commits, its effects *must persist*

Concepts

Concurrency Control (CC)

techniques to ensure **correct** results when running transactions concurrently

what does this mean?

Recovery

On crash or abort, how to get back to a consistent (**correct**) state?

The two are intertwined! The CC mechanism dictates the complexity of recovery!

What Does Correct Execution Mean?

Serializability

Regardless of the interleaving of operations, end result same as a serial ordering (no concurrency)

Schedule

One specific interleaving of the operations

T1: R(A) R(B) W(D) COMMIT

How a Schedule Works

T1: A += 1

T2: A -= 1

Before T1 and T2, A is 0

T1: R(A) W(A,1) COMMIT

T2: R(A) W(A, -1) COMMIT

State:	A=0	A=0	A=1	A=1	A=-1	A=-1
			uncommitted	committed	uncommitted	committed

How a Schedule Works

T1: A += 1

T2: A -= 1

Before T1 and T2, A is 0

T1:	R(A)		W(A)		COMMIT	
T2:		R(A)			W(A)	COMMIT
State:	A=0	A=0	A	A	A	A
			uncommitted	committed	uncommitted	committed

The value that is written doesn't matter

Serial Schedules

Logical facts

T1: r(A) w(A) r(B) w(B)

T2: r(A) w(A) r(B) w(B)

No concurrency (**serial 1**)

T1: r(A) w(A) r(B) w(B)

T2:

r(A) w(A) r(B) w(B)

No concurrency (**serial 2**)

T1:

r(A) w(A) r(B) w(B)

T2: r(A) w(A) r(B) w(B)

Are serial 1 and serial 2 equivalent?

More Example Schedules

Logical xacts

T1: r(A) w(A) **r(A)** w(B)

T2: r(A) w(A) r(B) w(B)

Concurrency (not equivalent to any serial schedule)

T1: r(A) w(A) r(A) w(B)

T2: r(A) w(A) r(B) w(B)

Concurrency (same as serial I!)

T1: r(A) w(A) r(A) w(B)

T2: r(A) w(A) r(B) w(B)

Important Concepts

Serial schedule

single threaded model. no concurrency.

each xact finishes (commits or aborts) before next xact runs

Equivalent schedule

the database state same at end of both schedules

Serializable schedule (gold standard)

equivalent to a serial schedule

These are just definitions.

How to *ensure* that schedules are serializable?

SQL → R/W Operations

```
UPDATE    accounts
SET       bal = bal + 1000
WHERE     bal > 1M
```

Read all balances for every tuple

Update those with balances > 1000

Does the access method matter?

YES!

Tuples(objects) read depend on access method

R/W Operations Depend On Access Paths

```
UPDATE  accounts
SET     bal = bal + 1000
WHERE   id = 123
```

If 1000 tuples in accounts, how many tuples are read:

If no indexes?

If index on bal?

If hash index on id?

if B+-tree index on id?

R/W Operations Depend On Access Paths

```
UPDATE  accounts
SET     bal = bal + 1000
WHERE   id = 123
```

If 1000 tuples in accounts, how many tuples are read:

If no indexes? 1000 tuples

If index on bal? 1000 tuples

If hash index on id? # tuples in hash bucket

if B+-tree index on id? # tuples in a page

NonSerializable Schedule → Anomalies

Reading in-between (uncommitted) data

T1: R(A) W(A) R(B) W(B) abort

T2: R(A) W(A) commit

WR conflict or dirty reads

Reading same data gets different values

T1: R(A) R(A) W(A) commit

T2: R(A) W(A) commit

RW conflict or unrepeatable reads

NonSerializable Schedule → Anomalies

Stepping on someone else's writes

T1: W(A) W(B) commit

T2:  W(A) W(B) commit

WW conflict or lost writes

Note: all anomalies involve writing to data that is read/written to.

If we track our writes, maybe can prevent anomalies

Conflict Serializability

cheaply prevent non-serializable scheds

Over-conservative: some serializable schedules disallowed.

Intuition: if xacts don't touch the same records, should be OK.

Conflict Serializability

What is a conflict?

For 2 operations, if run in different order, get different results

Conflict?	R(A)	W(A)
R(A)	NO	YES
W(A)	YES	YES
R(B)	NO	NO
W(B)	NO	NO

Conflict Serializability

def: possible to swap non-conflicting operations to derive a serial schedule.

\forall conflicting operations O_1 of T_1 , O_2 of T_2

O_1 always before O_2 in the schedule or

O_2 always before O_1 in the schedule

Operation O_i is a read or write of an object

	1	2	3	4
T1:	R(A)	W(A)	R(B)	W(B)
	5	6	7	8
T2:	R(A)	W(A)	R(B)	W(B)

Logical

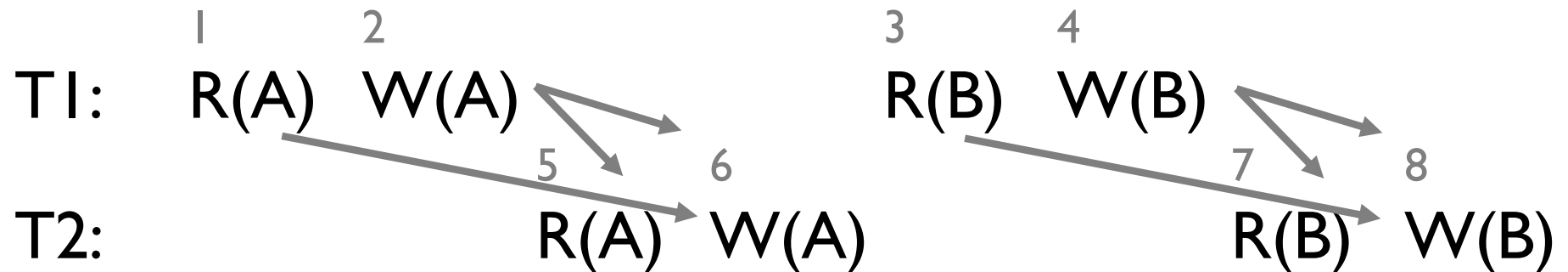
Conflicts

1,6 2,5 2,6 3,8 4,7 4,8

Logical

	1	2	3	4
T1:	R(A)	W(A)	R(B)	W(B)
	5	6	7	8
T2:	R(A)	W(A)	R(B)	W(B)

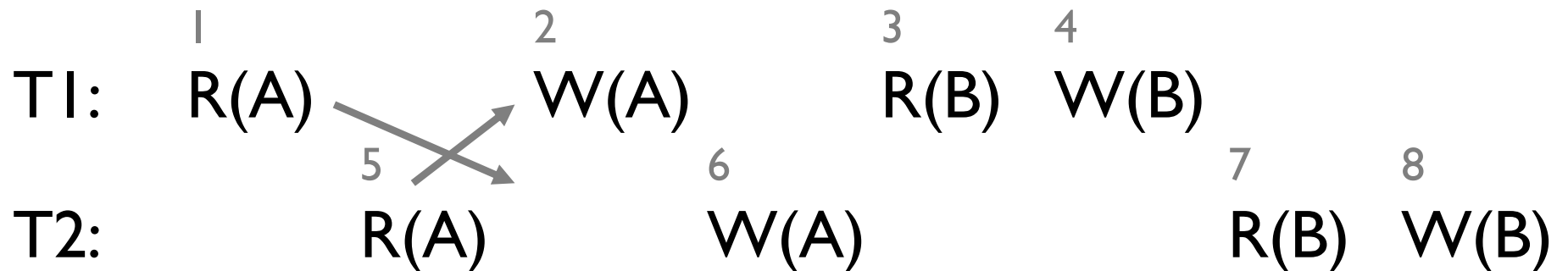
Conflict Serializable



Logical

	1	2	3	4
T1:	R(A)	W(A)	R(B)	W(B)
	5	6	7	8
T2:	R(A)	W(A)	R(B)	W(B)

Not Conflict Serializable



Conflict Serializability

Transaction Precedence Graph

Nodes are transactions

Edge $T_i \rightarrow T_j$ if:

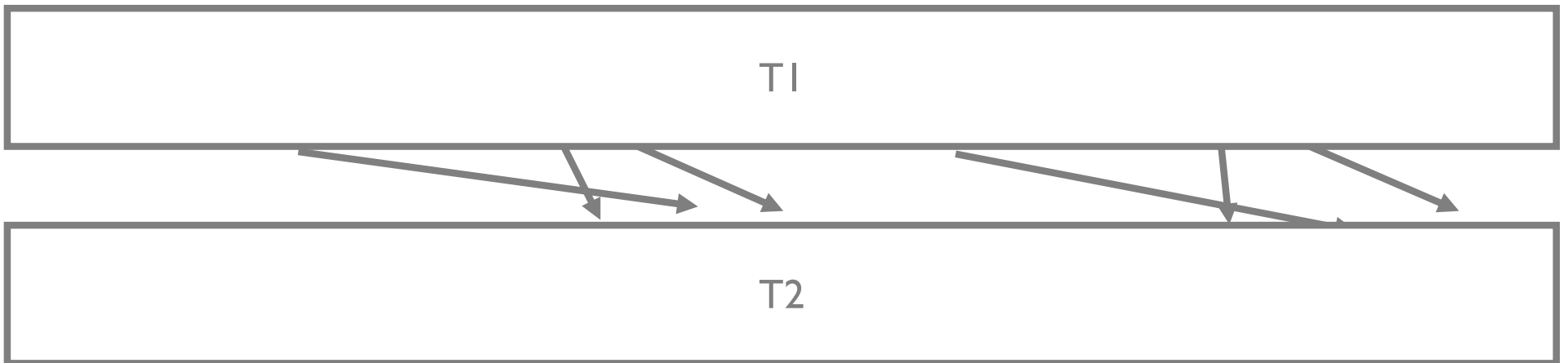
1. T_i read/write A before T_j writes A or
2. T_i writes some A before T_j reads A

Acyclic graph (no cycles) = conflict serializable!

Logical

	1	2	3	4
T1:	R(A)	W(A)	R(B)	W(B)
	5	6	7	8
T2:	R(A)	W(A)	R(B)	W(B)

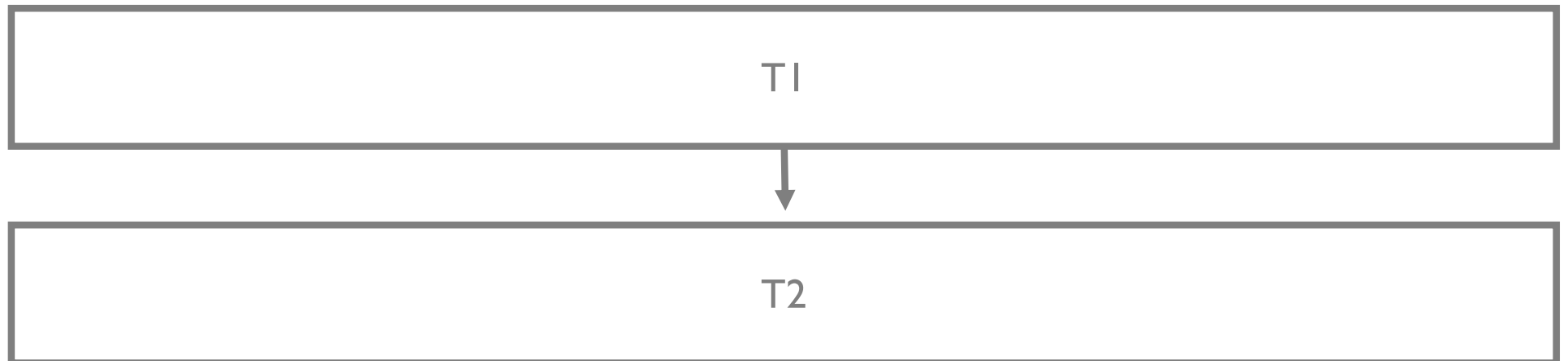
Conflict Serializable



Logical

	1	2	3	4
T1:	R(A)	W(A)	R(B)	W(B)
	5	6	7	8
T2:	R(A)	W(A)	R(B)	W(B)

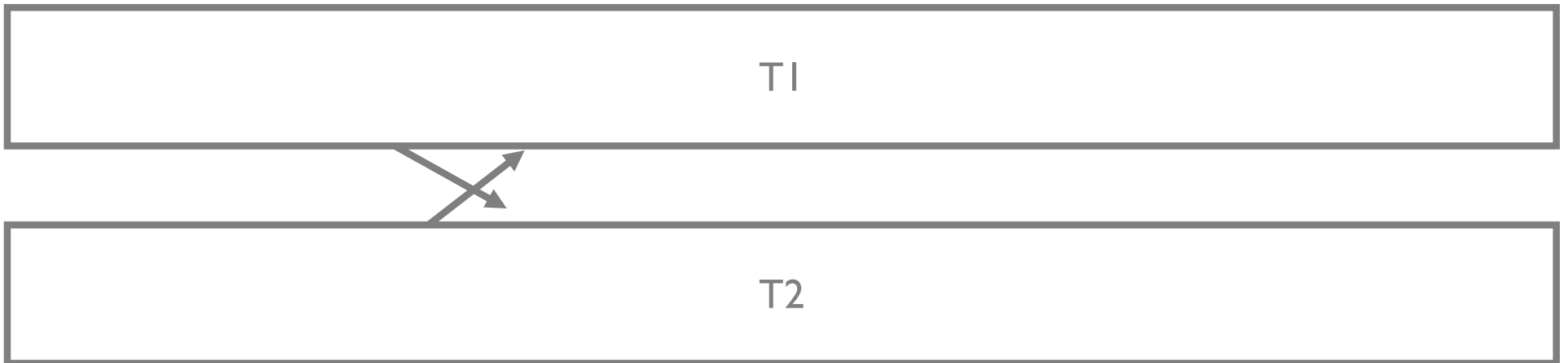
Conflict Serializable



Logical

	1	2	3	4
T1:	R(A)	W(A)	R(B)	W(B)
	5	6	7	8
T2:	R(A)	W(A)	R(B)	W(B)

Not Conflict Serializable



Commits/Aborts Complicate Things

So far, focused on schedule equivalence assuming that all transactions will commit.

But some transactions may abort and want to undo the changes.

These are OK right?

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

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

Fine, but what about COMMITing?

T1	R(A) W(A)	R(B) ABORT
T2	R(A) COMMIT	

Not recoverable

Promised T2 everything is OK. IT WAS A LIE.

T1	R(A) W(B) W(A)	ABORT
T2	R(A) W(A)	

Cascading Rollback.

T2 read uncommitted data → T1's abort undos T1's ops & T2's

Lock-based Concurrency Control

Everything so far has been definitions.

Want a *procedure* that will guarantee serializable schedules

Naïve approach:

- Lock database when starting xact
- Unlock database when xact ends

Want something similar, but by locking objects (like records)

Lock-based Concurrency Control

Must get **S**hared(read) or e**X**clusive(write) lock BEFORE op
 If other xact has lock, can acquire if lock table says so

		T1		Can this schedule happen?			
Allowed?	S	X					
T2	S	Y	N	T1	R(A)	W(A)	
	X	N	N	T2			
							R(B) ABORT
						R(A)	COMMIT

YES

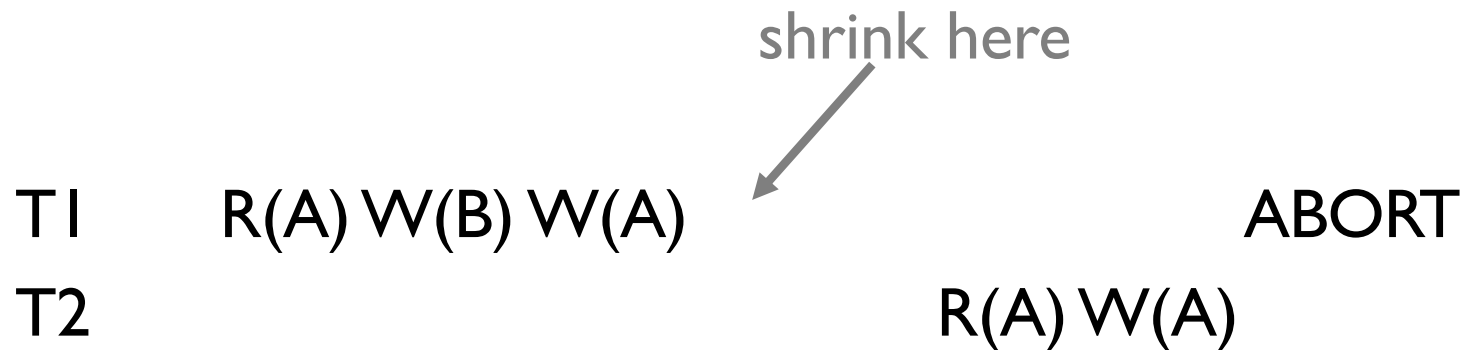
didn't discuss when to release locks

Lock-based Concurrency Control

Two-phase locking (2PL)

Growing phase: acquire locks

Shrinking phase: release locks



Uh Oh, same problem

Lock-based Concurrency Control

Strict two-phase locking (Strict 2PL)

Growing phase: acquire locks

Shrinking phase: release locks

Hold onto locks until commit/abort



Why? Which problem does it prevent?

T1	R(A) W(B)	W(A)	ABORT
T2		R(A) W(A)	

Guarantees serializable schedules! Avoids cascading rollbacks!

Review

Issues

WR: dirty reads

RW: unrepeatable reads

WW: lost writes

Schedules

Equivalence

Serial

Serializable

Serializability

Conflict serializability

how to detect

Conflict Serializable Issues

Not recoverable

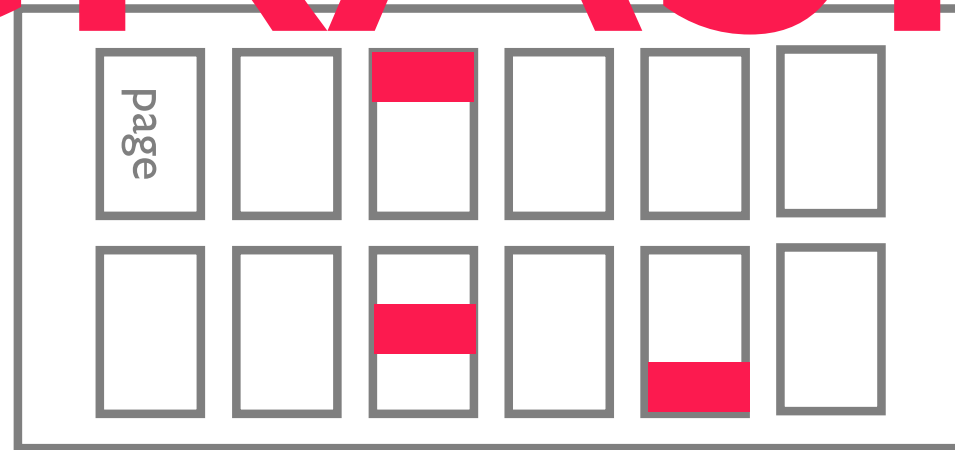
Cascading Rollback

Strict 2 phase locking (2PL)

CRASH

Normal Execution

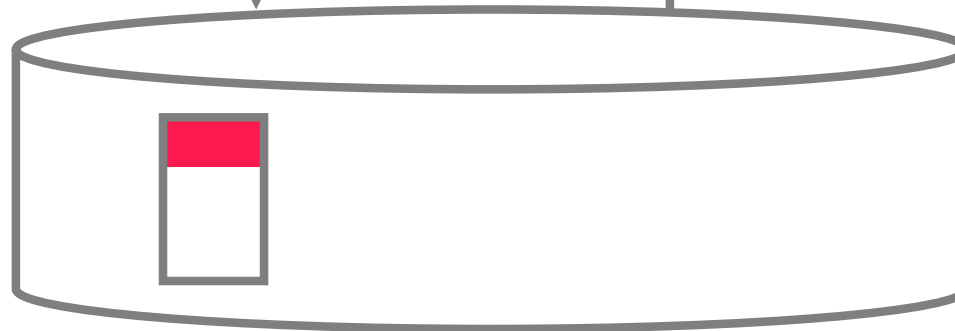
RAM



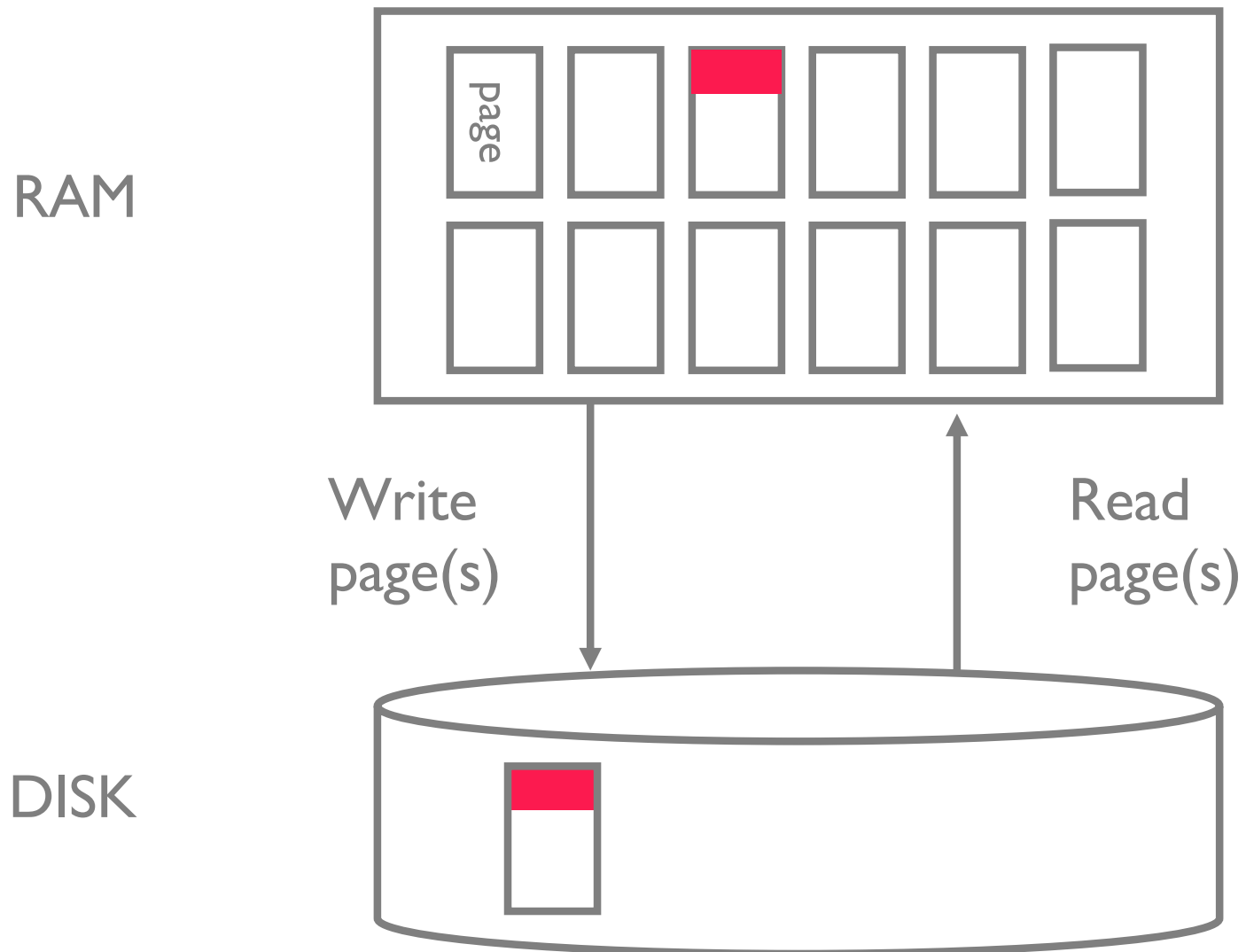
Write
page(s)

Read
page(s)

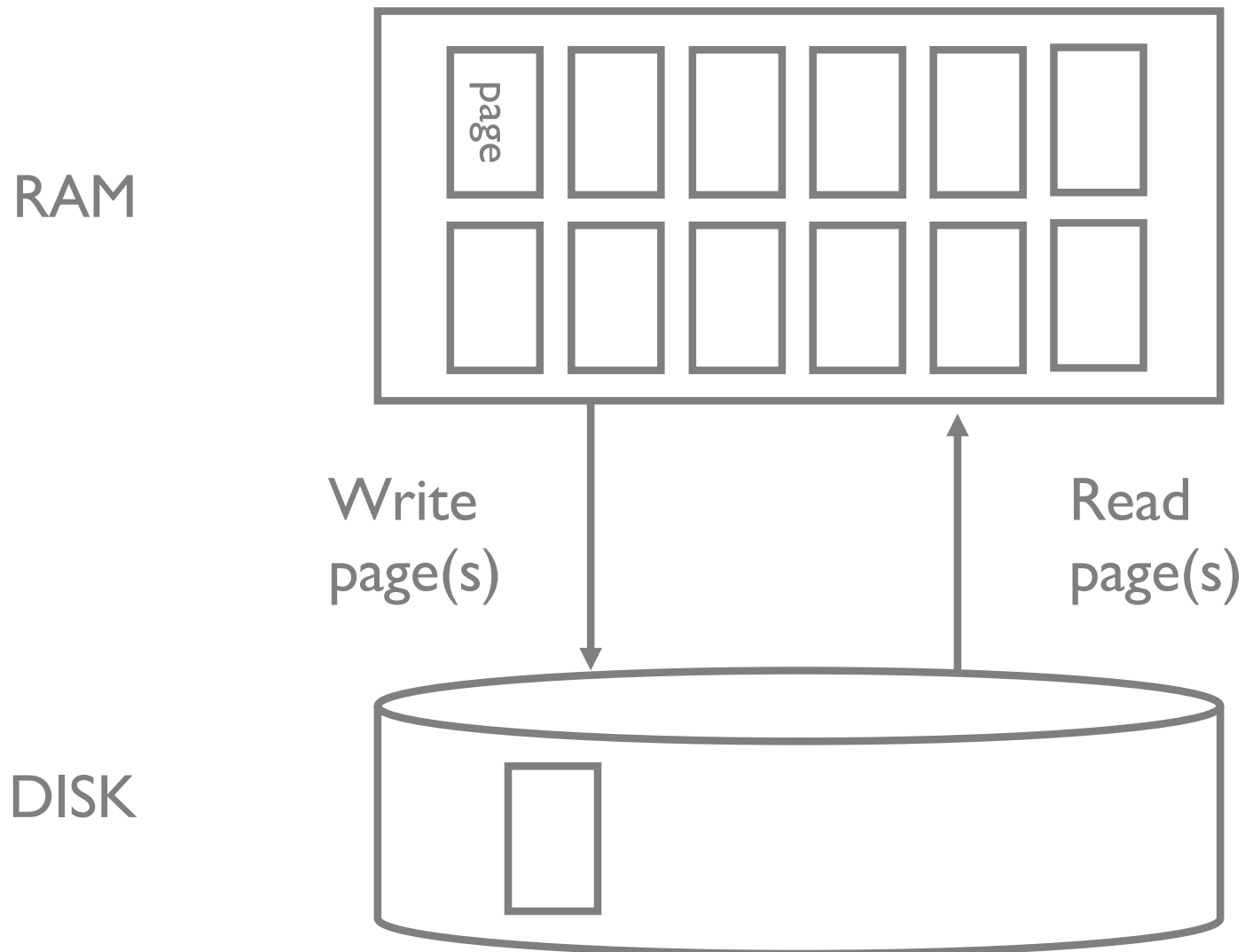
DISK



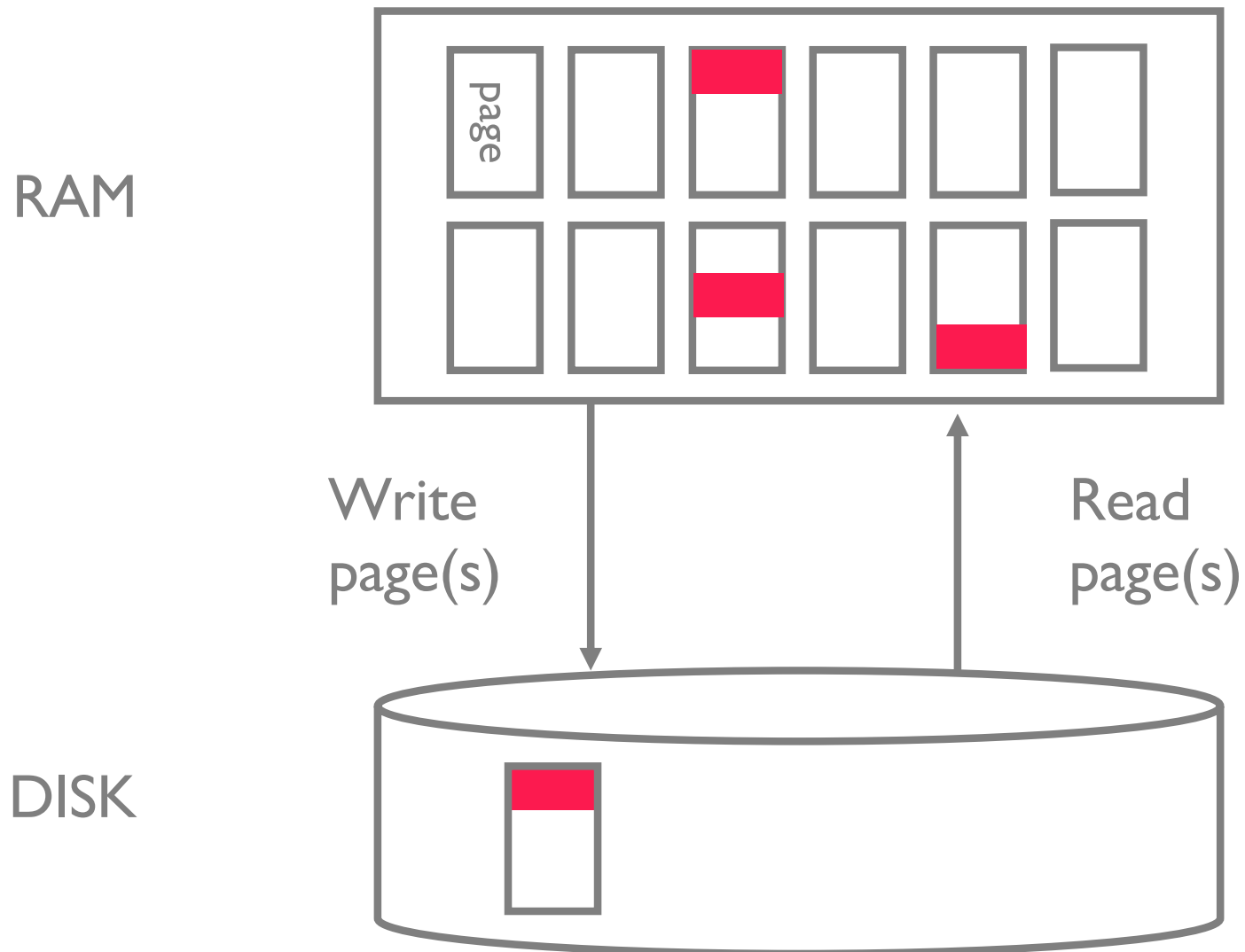
After a Crash



If DB did not say “OK, committed”
State should look like:



If T1 Committed and DB said “COMMITTED”
State should look like:



Recovery

Two properties: Atomicity, Durability

Assumption in class

Disk is safe. Memory is not.

Running strict-2PL

Need to account for

when pages are modified

when pages are flushed to disk

There's no perfect recovery, just trade-offs

Recovery

Deal with 2 cases

When could uncommitted data appear after crash?
wrote modified pages before commit

If T2 commits, what could make it not durable?
didn't write all changed pages to disk

Aborts and Undos

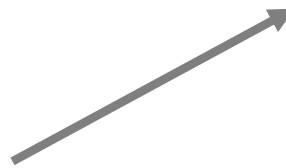
If Tx aborts, must undo all its actions

Ty that read Tx's writes must be aborted
(cascading abort)

Strict 2PL avoids cascading aborts

Use a log to know what actions to undo

aborting



1. $A = 1$
2. $B = 5$
3. $C = 10$
4. BEGIN T5
5. $A = 10$
6. $B = B + A$
7. $C = B - 2$
8. ABORT
9. undo 7
10. undo 6
- ...

Aborts and Undos

If Tx aborts, must undo all its actions

Ty that read Tx's writes must be aborted
(cascading abort)

Strict 2PL avoids cascading aborts

Use a log to know what actions to undo

On crash, abort all non-committed xacts

recovering



1. $A = 1$
2. $B = 5$
3. $C = 10$
4. BEGIN T5
5. $A = 10$
6. $B = B + A$
7. $C = B - 2$
8. CRASH
9. undo 7
10. undo 6
- ...

Logs

Log is the *ground truth*

Log records

- writes: old & new value

- commit/abort actions

- xact id & xact's previous log record

Persist log records (write to disk) *before* data pages persisted

Is this enough?

Durability

Is there an execution that
writes log records before data pages
but is incorrect?
(e.g., not ACID)

Durability

Baseline scenario

TI writes to *A* in memory

log record of write written to disk

start writing page with *A* to disk...

TI commits

Durability

OK scenario

TI writes to A in memory

log record of write written to disk

start writing page with A to disk...

crash

TI commits

Durability

OK scenario

TI writes to A in memory

log record of write written to disk

crash

start writing page with A to disk...

TI commits

Durability

Bad scenario

TI writes to A in memory

TI commits

log record of write is written to disk

start writing page with A to disk...

crash

Can undo help us?

No, need to *redo* TI, otherwise no durability!

Durability

Worse scenario

TI writes to A in memory

TI commits

crash

log record of write is written to disk

start writing page with A to disk...

Can undo help us?

Can't redo TI, no durability! Shareholders mad

Logs

Log is the *ground truth*

Log records

writes: old & new value

commit/abort actions

xact id & xact's previous log record

Write ahead logging (WAL)

1. Persist log records (write to disk) *before* data pages persisted
2. Persist all log records *before* commit
3. Log is *ordered*, if record flushed, all previous records must be flushed

(1) guarantees UNDO info

(2) guarantees REDO info

Aries Recovery Algorithm

3 phases

Analyze the log to find status of all xacts

Committed or in flight?

Redo xacts that were committed

Now at the same state at the point of the crash

Undo partial (in flight) xacts

Redo/Undo recovery ops participate in WAL

Recovery is *extremely* tricky and *must be correct*

Aries

T1 R(A) R(C) W(A)

COMMIT

T2

W(B)

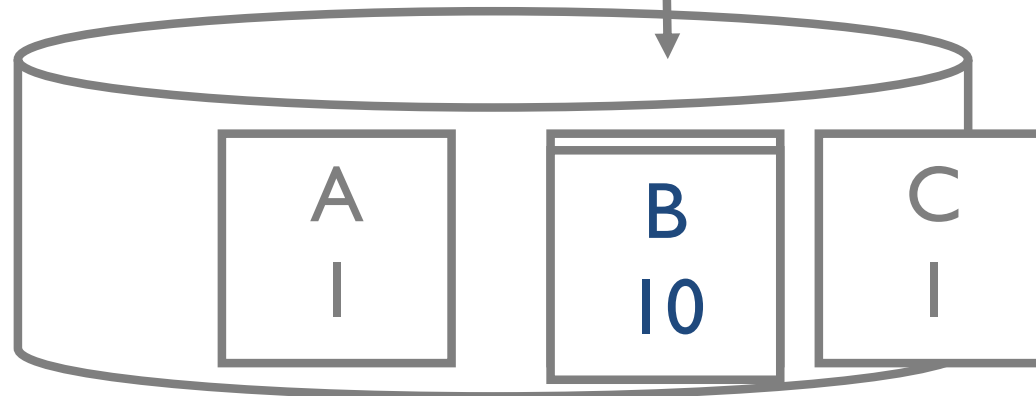
CRASH

1. A = 1
2. B = 5
3. begin T1
4. begin T2
5. A = 1 + 5
6. B = 10
7. commit

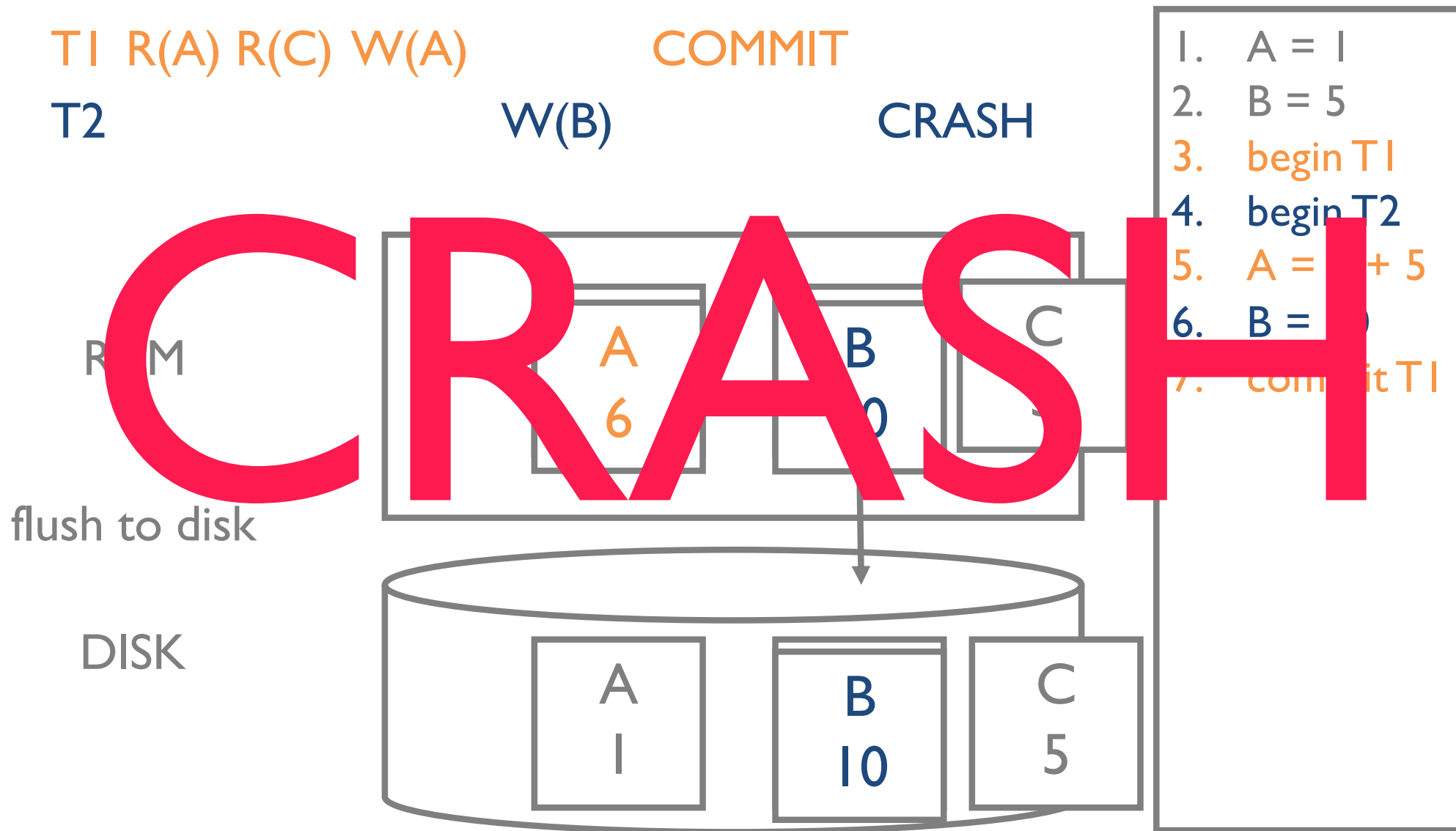
CRASH

flush to disk

DISK



Aries: alternative flushing order



Aborts and Undos (I)

T1 R(A) R(C) W(A)

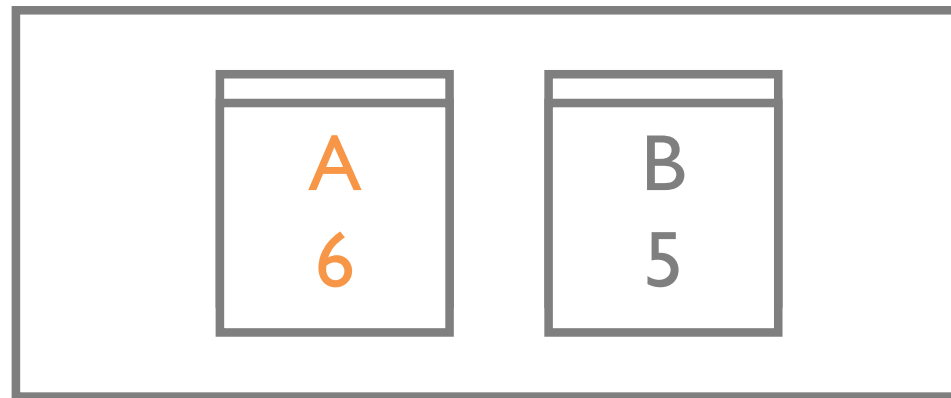
COMMIT

T2

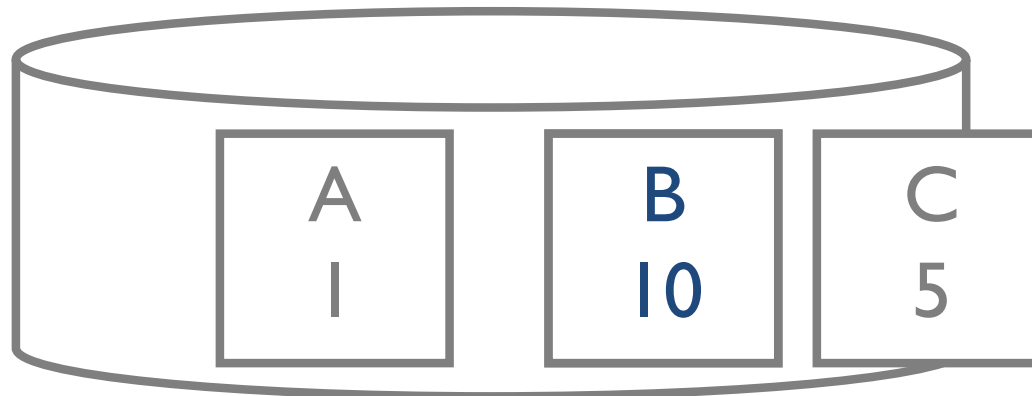
W(B)

CRASH

RAM



DISK



1. $A = 1$
2. $B, C = 5$
3. begin T1
4. begin T2
5. $A = 1 + 5$
6. $B = 10$
7. commit T1
8. redo op5
9. undo op6

Aborts and Undos (2)

T1 R(A) R(B) W(A)

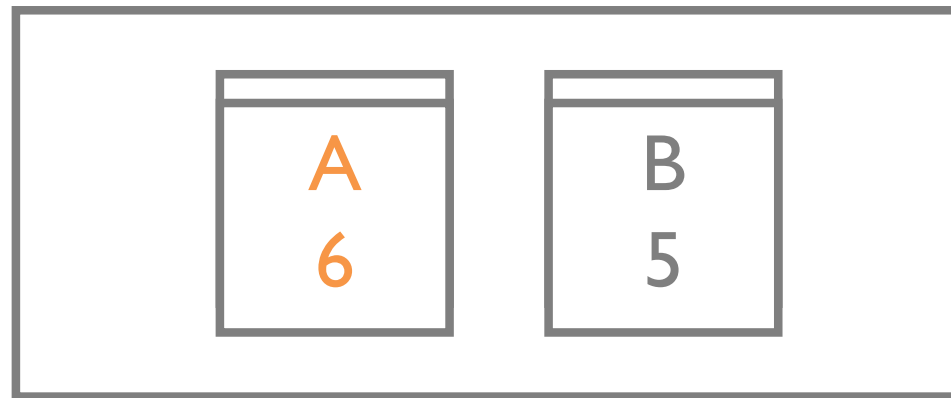
COMMIT

T2

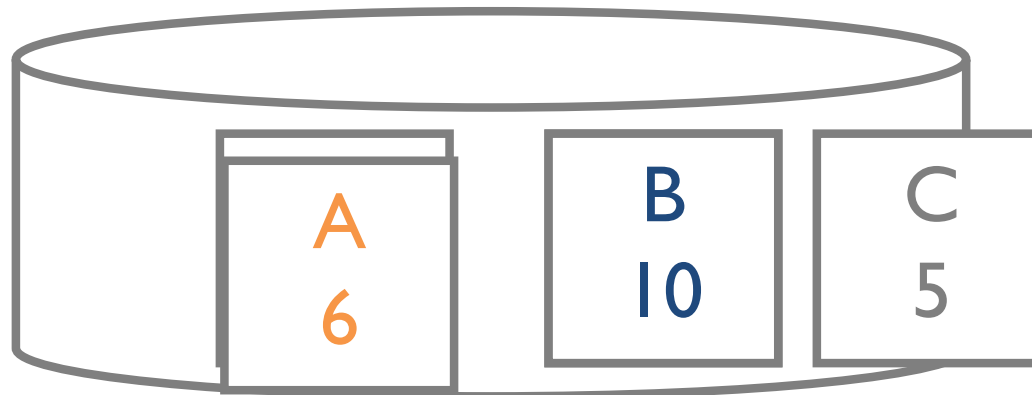
W(B)

CRASH

RAM



DISK



1. $A = 1$
2. $B = 5$
3. begin T1
4. begin T2
5. $A = 1 + 5$
6. $B = 10$
7. commit T1
8. redo op5
9. undo op6

Summary

Recovery depends on what failures are tolerable

Buffer pool can write RAM pages to disk any time

Recover to the moment of the crash, then undo all non-committed operations

WAL protocol

Recovery Manager ensures durability and atomicity via redo and undo

You should know

What transactions/schedules/serializable are

Can identify conflict serializable schedules

Can identify schedule anomalies

Can identify strict 2PL executions

Understand WAL and what it provides

Given an executed schedule, and a log file, run the proper sequence of undo/redos