MySQL For Frappe Devs

Credits

This is a "Stripped down" version of CMU's "Database Systems" course which is available online for free.

Check it out: https://15445.courses.cs.cmu.edu/fall2023/

P.S. - Professor of this class is fucking awesome!

Disclaimer

This slide deck was used for internal training. You can not get the same experience or “knowledge transfer” by just reading the slides.

- If you’ve seen this before, then you can probably use it to recall something.
- If you’re seeing this for the first time, then you should use this deck as a “teaser” and consider self-studying the relevant concepts.

Warning: abstraction alert

- Conceptual abstraction of SQL database <- Most DB users
- **high level implementation** <- Where we want to be
- low level implementation details <- MySQL devs

A LOT of hand-wavy explanations incoming.

Outline

- Storage
- Bufferpool
- Indexing (B+Tree)
- Query planner and execution
- Concurrency, locking and MVCC
- Logging and recovery

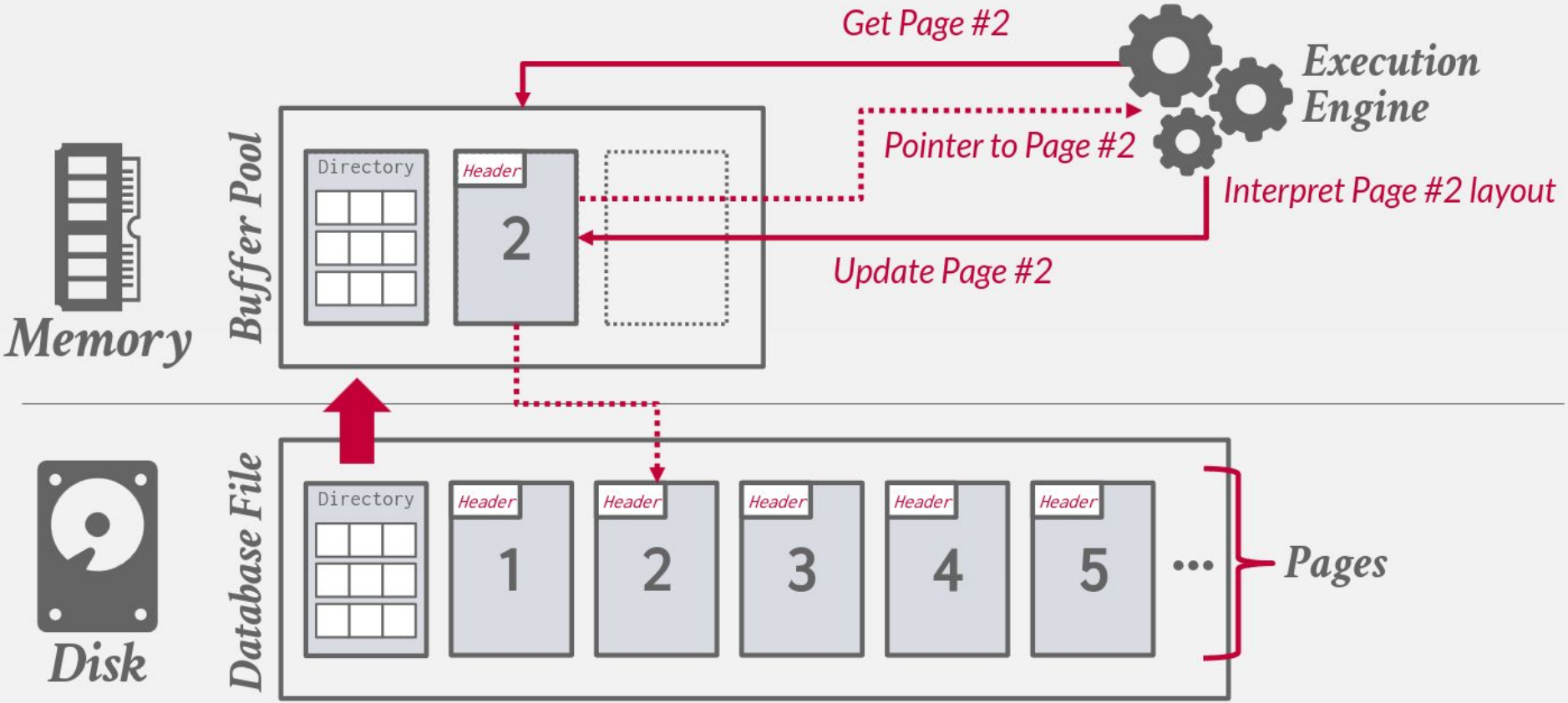
Storage - RAM vs Disk

Device	Latency (ns)
DRAM	100
SSD	16,000
HDD	2,000,000

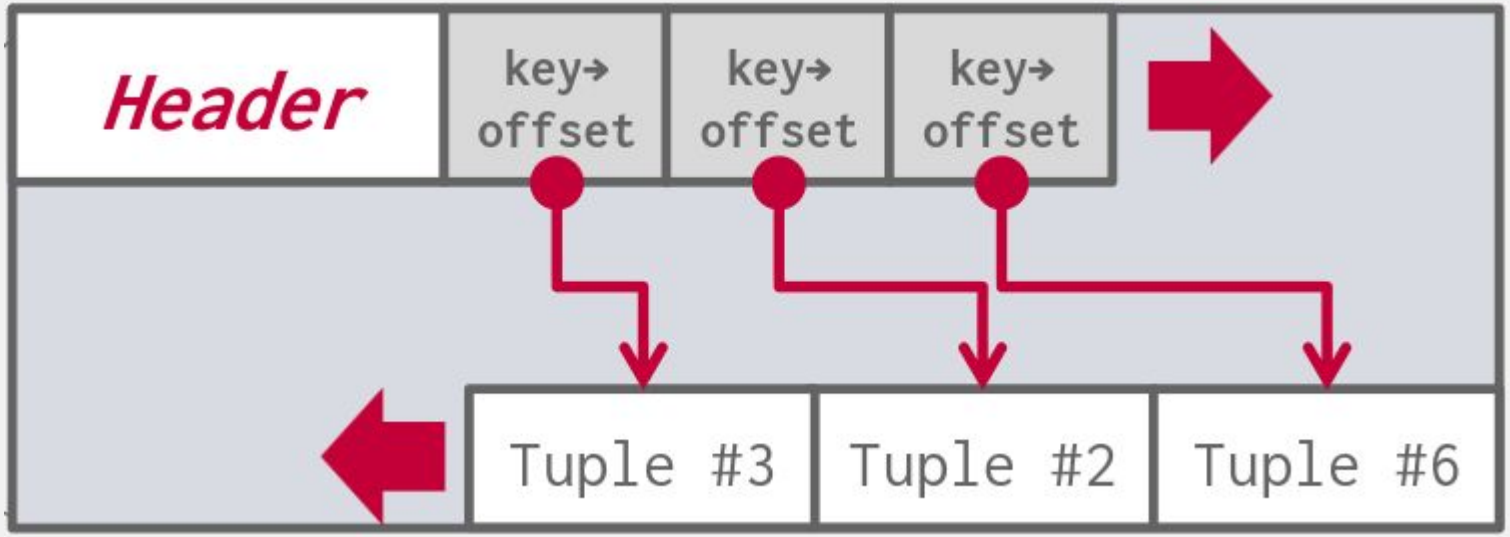
Storage - RAM vs Disk

Device	Latency (ns)	Relative
DRAM	100	1 second
SSD	16,000	2.6 minutes
HDD	2,000,000	5.5 hours

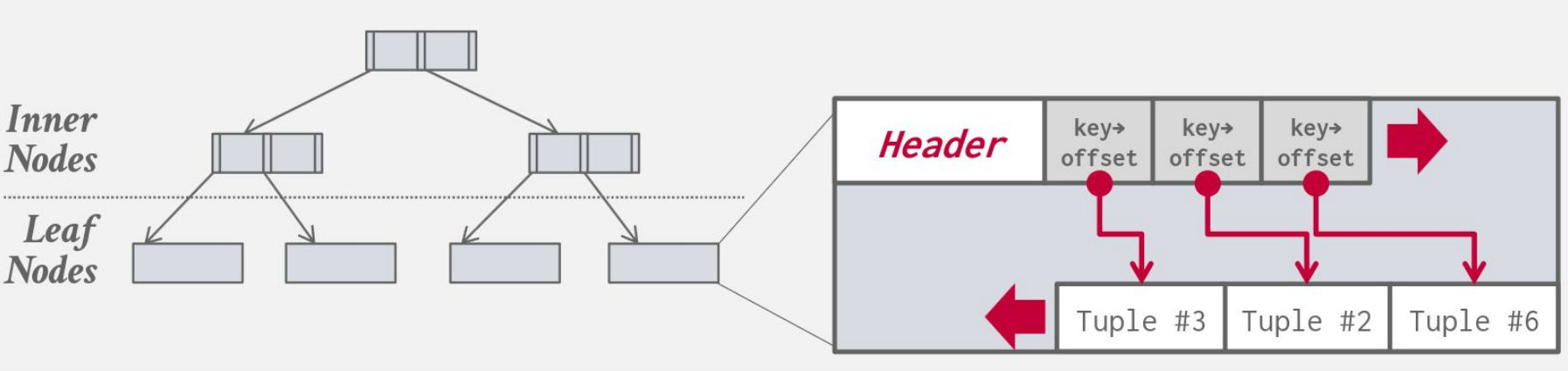
Disk Oriented Database



MySQL Page - 16KB

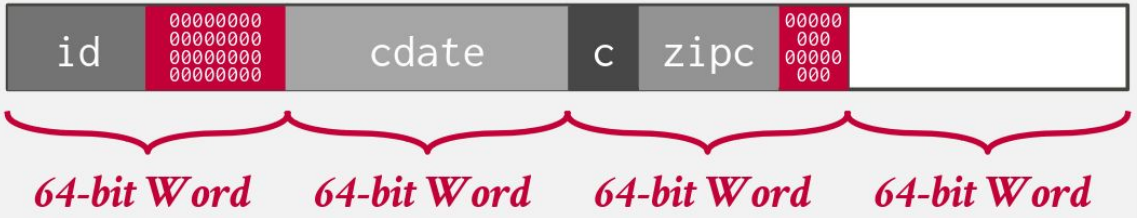


InnoDB's "Index-Organized" structure



Storing individual tuple (aka row)

```
CREATE TABLE dj2pl (
  id INT PRIMARY KEY,      -- 32-bits
  cdate TIMESTAMP,         -- 64-bits
  color CHAR(2),           -- 16-bits
  zipcode INT              -- 32-bits
);
```

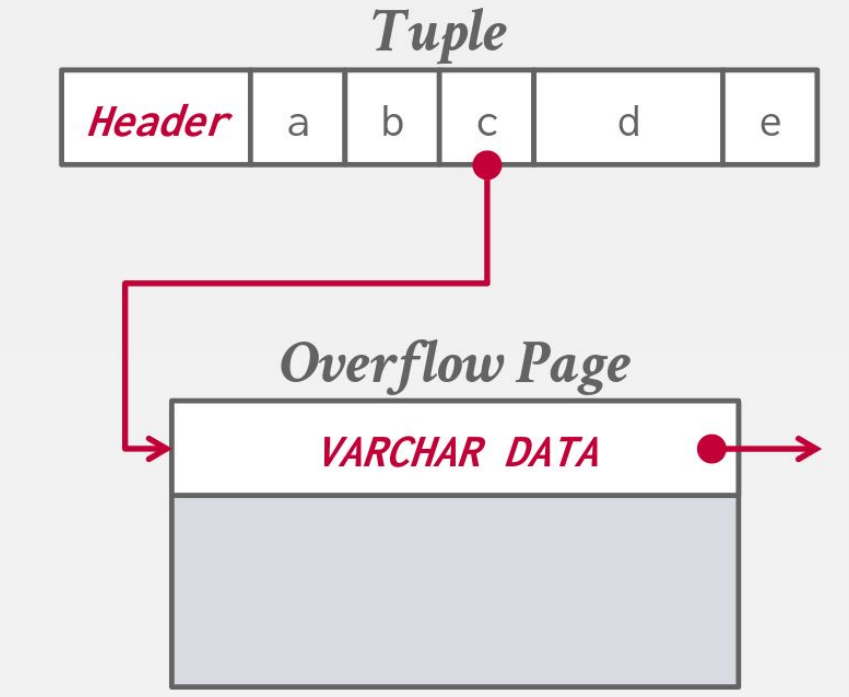


Datatypes and their representation

- Integers / Float -> same as C
- VARCHAR -> 2 or 3 byte header + actual data*
- Date(time) - encoding each parts as integer
- Text/Long Text - pointer to some other “overflow page”
- Decimal - fixed precision number
- UUID - 16 bytes binary data.

* - read UTF-8 blog by Joel Spolsky

“Text” representation



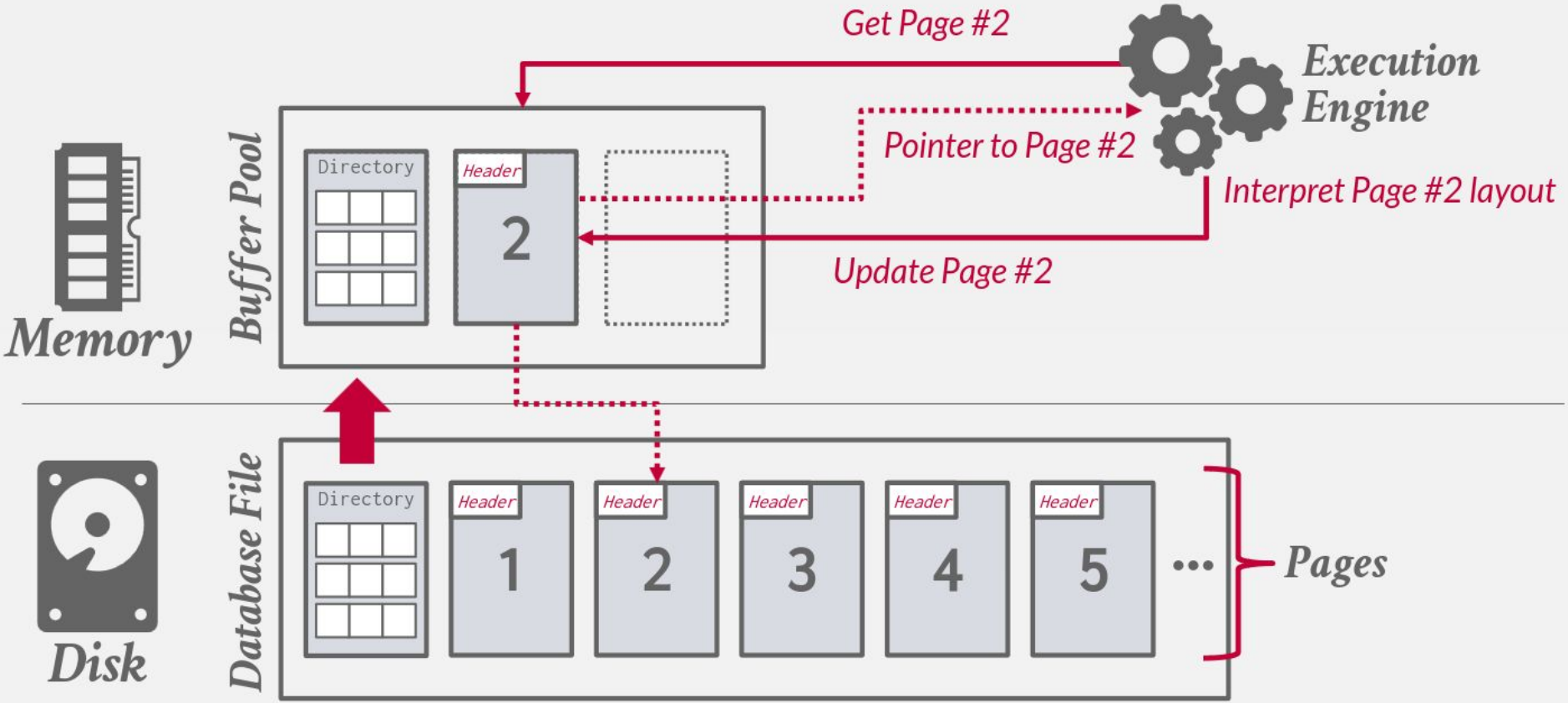
Practical Implications

- “Max row size reached” - https://docs.erpnext.com/docs/user/manual/en/maximum-number-of-fields-in-a-form
- Our config: https://github.com/frappe/press/blob/master/press/playbooks/roles/mariadb/templates/mariadb.cnf

Outline

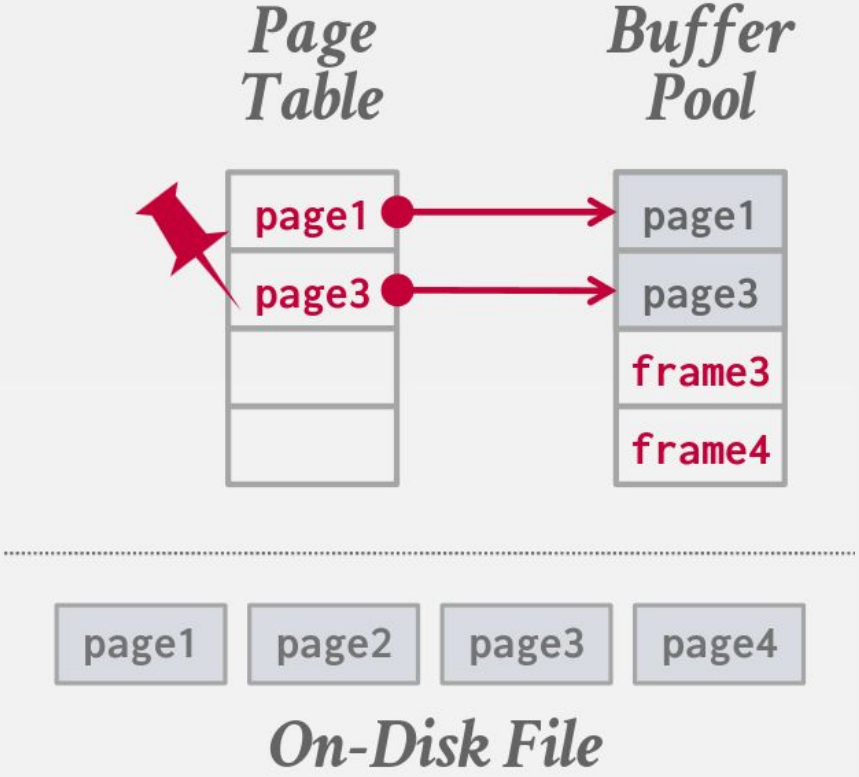
- ~~Storage~~
- Bufferpool
- Indexing (B+Tree)
- Query planner and execution
- Concurrency, locking and MVCC
- Logging and recovery

Buffer pool


Get Page #2
Execution Engine
Pointer to Page #2
Interpret Page #2 layout
Update Page #2
Memory
Buffer Pool
Directory
Header
2
Disk
Database File
Directory
Header
1
Header
2
Header
3
Header
4
Header
5
...
Pages

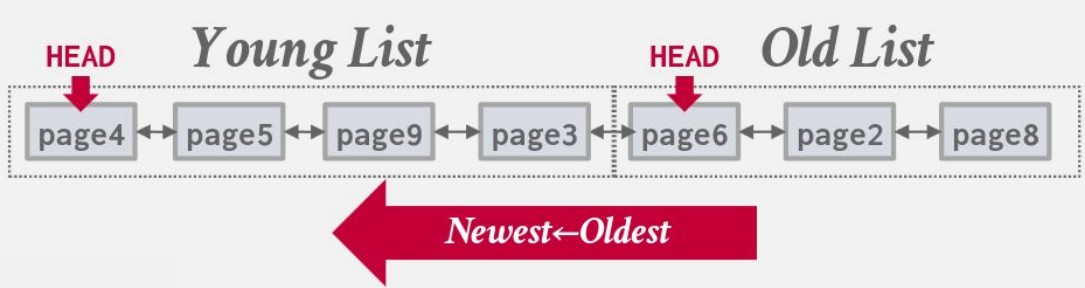
Page Table

- hashtable that keeps track of pages
- Track dirty pages
- metadata about accesses and locking



Bufferpool size and eviction

- Bufferpools have fixed size
- What happens when you run out memory?
- Enter LRU-**K**



Practical Implications

- Size specified in our config: mariadb.cnf
- How do you recommend DB server size?
- Lets `monitor` some of this:
  - Bufferpool size
  - BP miss ratio
  - LRU Sub-chain churn

Questions

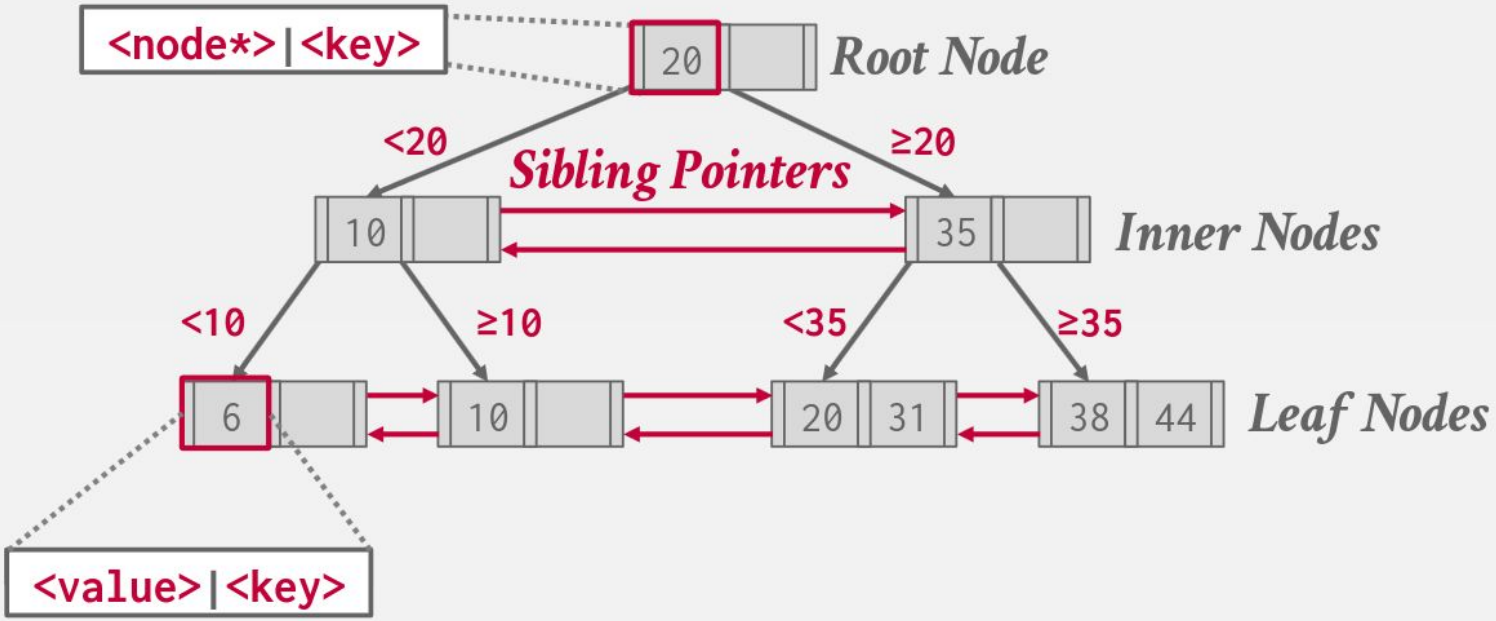
- How backups affect bufferpool?
- Why is swap bad?
- What should be the size of DB server?

Outline

- ~~Storage~~
- ~~Bufferpool~~
- Indexing (B+Tree)
- How joins Work
- Query planner and execution
- Concurrency, locking and MVCC
- Logging and recovery

Indexes - B+Tree

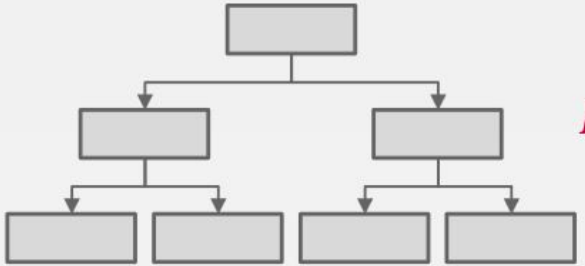
- Every index is B+Tree
- Actual table is just “primary” index, leaf nodes = data.

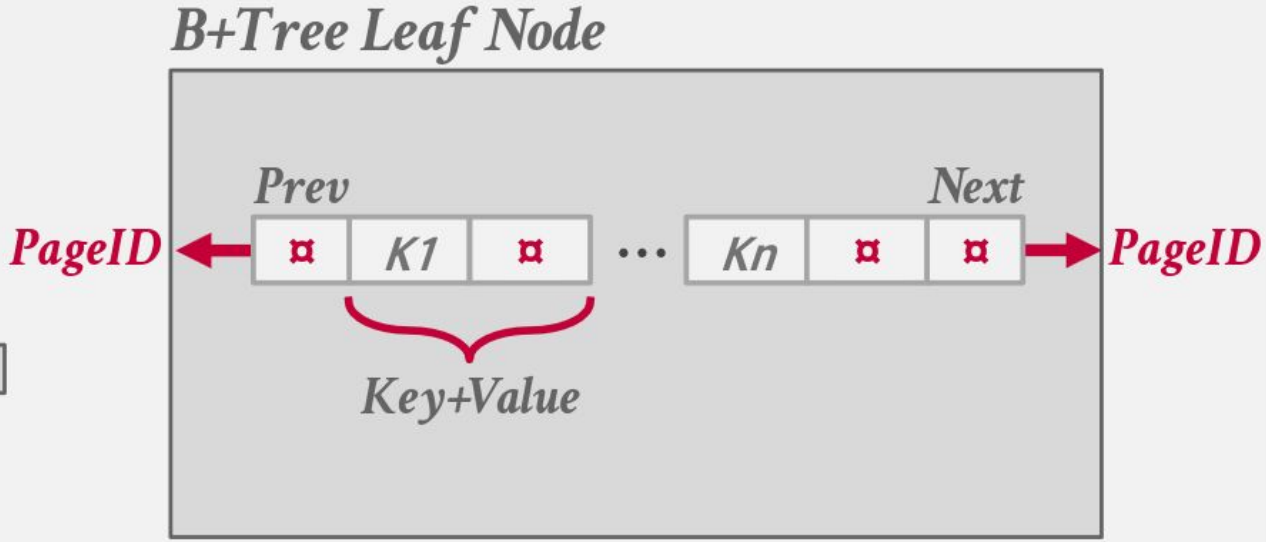


Indexes - B+Tree

"Value"

- Primary key index -> Actual tuple data
- Secondary index -> Primary key





Multicolumn index?

- Just concatenate columns
- E.g.
  - Index of first name might contain: “Alice”, “Bob”, “Zed”
  - Index of last name might contain: “Burger”, “Chains”, “Zodd”
  - Multi col index of first and last name is just concatenation (“Alice”, “Chains”), (“Bob”, “Burger”), (“Zed”, “Zodd”)

Indexes - Visualization

DEMO

Practical Implications

- Primary key naming matters - #25309
- Multi-col index -> works with prefix only.

Outline

- ~~Storage~~
- ~~Bufferpool~~
- ~~Indexing (B+Tree)~~
- Query planner and execution
- Concurrency, locking and MVCC
- Logging and recovery

Joining Tables - Naive Join

```
for o in outer_table:
    for i in inner_table:
        if o.name = i.parent:
            yield (o, i)
```

Joining Tables - Block nested join

```
for o_page in outer_table:
    for i_page in inner_table:
        for o_row in o_page:
            for i_row in i_page:
                if o.name = i.parent:
                    yield (o, i)
```

Joining Tables - Index nested join

```
for o in outer_table:
    for i in Index(i.parent = o.name):
        yield (o, i)
```

Joining Tables - Sort merge join

- Sort both tables
- Use two pointers and yield matching rows

ID	name
1	Admin
2	Garret
3	Bob

ID	role
1	Admin
2	System
3	Guest
1	System
2	Sales

Joining Tables - Sort merge join

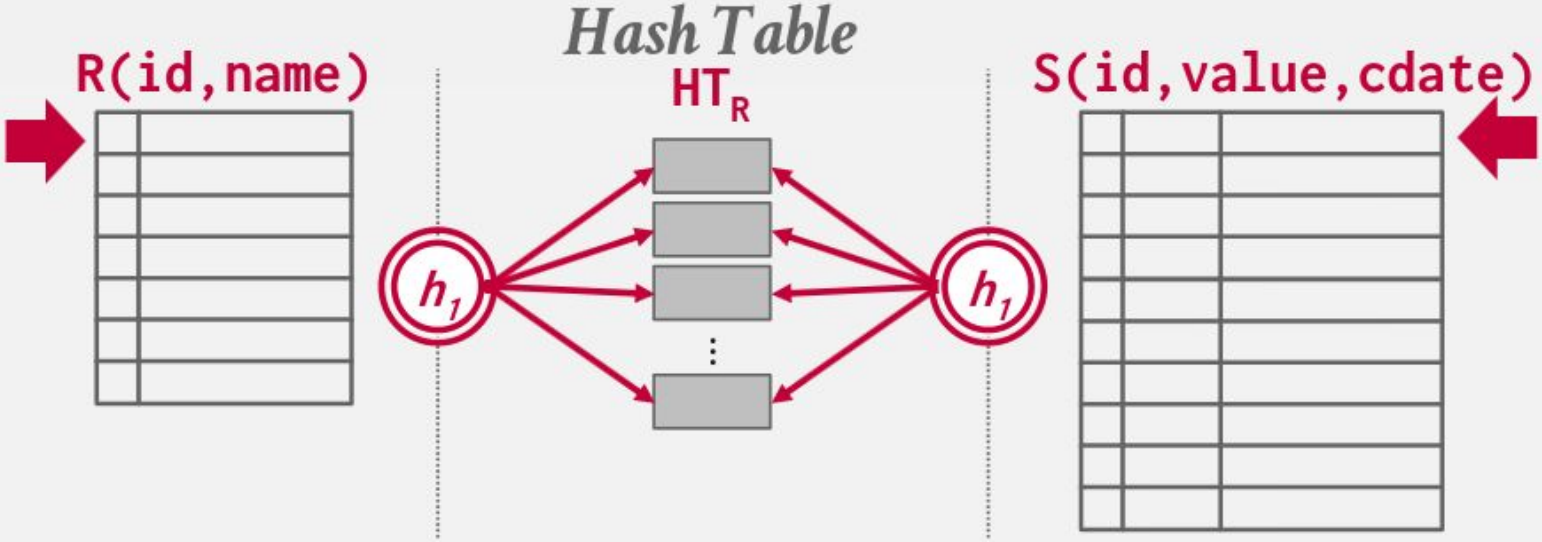
- Sort both tables
- Use two pointers and yield matching rows

ID	name
1	Admin
2	Garret
3	Bob

ID	role
1	Admin
1	System
2	Sales
2	System
3	Guest

Joining Tables - Hash Join

- Recent addition, not enabled by default.
- Also: “Adaptive hash index”



“Conceptual” Execution order

1. FROM
2. JOIN
3. WHERE
4. GROUP
5. HAVING
6. SELECT / AGGREGATE
7. ORDER
8. OFFSET
9. LIMIT

Actual Execution order

???

What’s the ideal execution ordering?

```
SELECT *

from `tabItem`;
```

What’s the ideal execution ordering?

```
SELECT *
from `tabItem`
order by `modified`
limit 20;
```

What’s the ideal execution ordering?

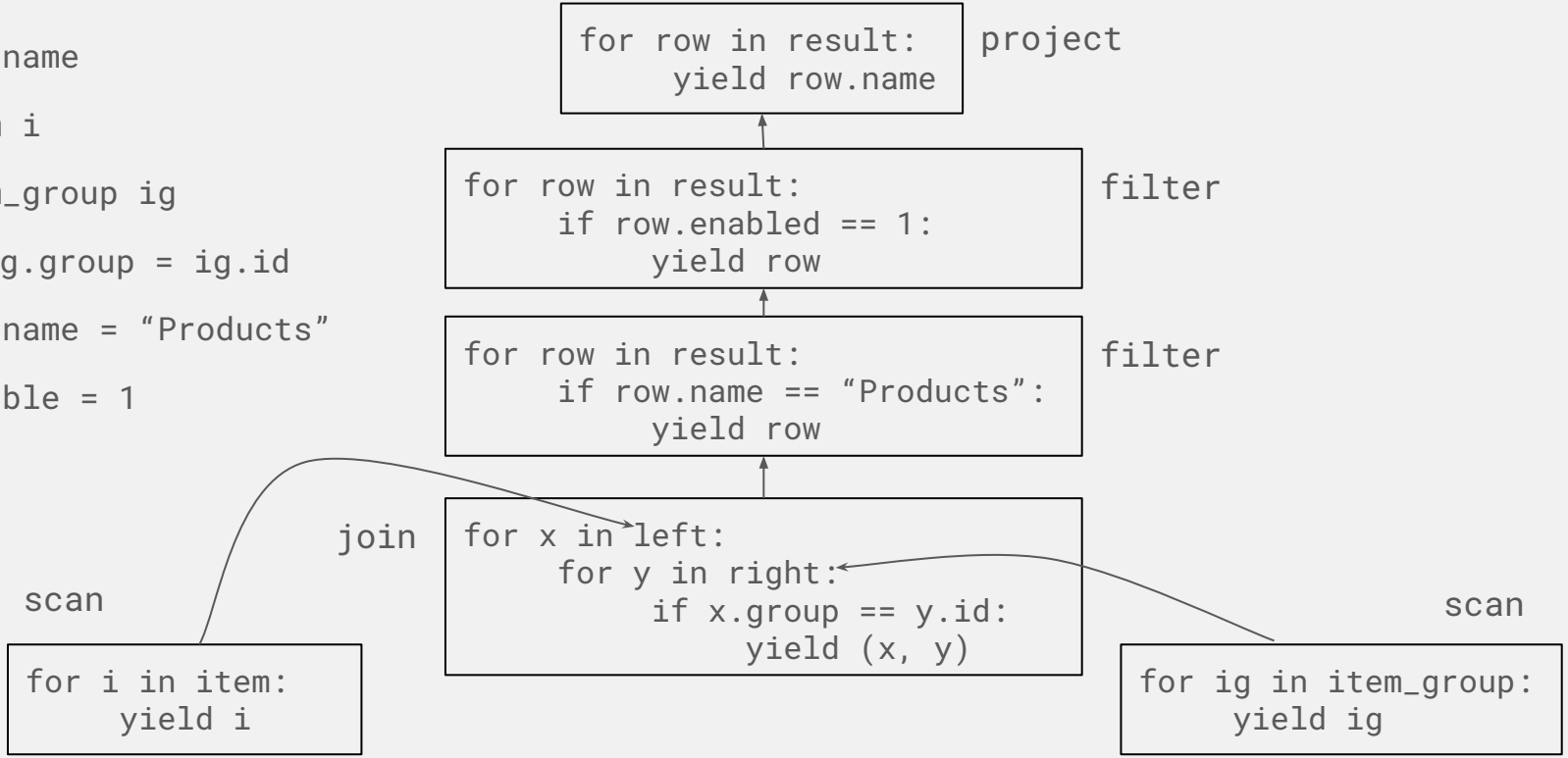
```
SELECT *
from `tabStock Ledger Entry`
where
    item_code = ‘X’
    and warehouse = ‘Y’
    and posting_datetime > ‘2024-01-01’
order by `posting_datetime` desc
limit 1;
```

Execution model - Iterators

```
select i.name
from item i
join item_group ig
     on ig.group = ig.id
where ig.name = “Products”
and i.enable = 1
```

Execution model - Iterators

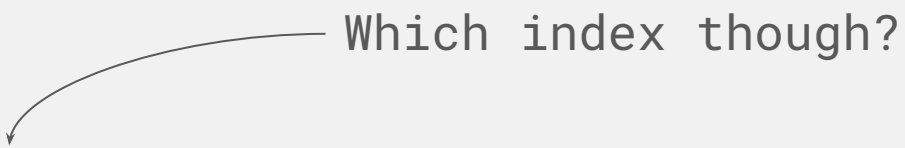
```
select i.name
from item i
join item_group ig
     on ig.group = ig.id
where ig.name = “Products”
and i.enable = 1
```



Execution model - Sequential scan

```
# “Full table scan”
for page in pages:
    for row in page:
        if matches(row, conditions):
            yield row
```

Execution model - Index scan




```
for pk in Index(predicate):
    row = get_record(pk)
    if match(row, other_conditions):
        yield row
```

Execution model - Index-merge scan

… WHERE A = “X” **OR** B = “Y”

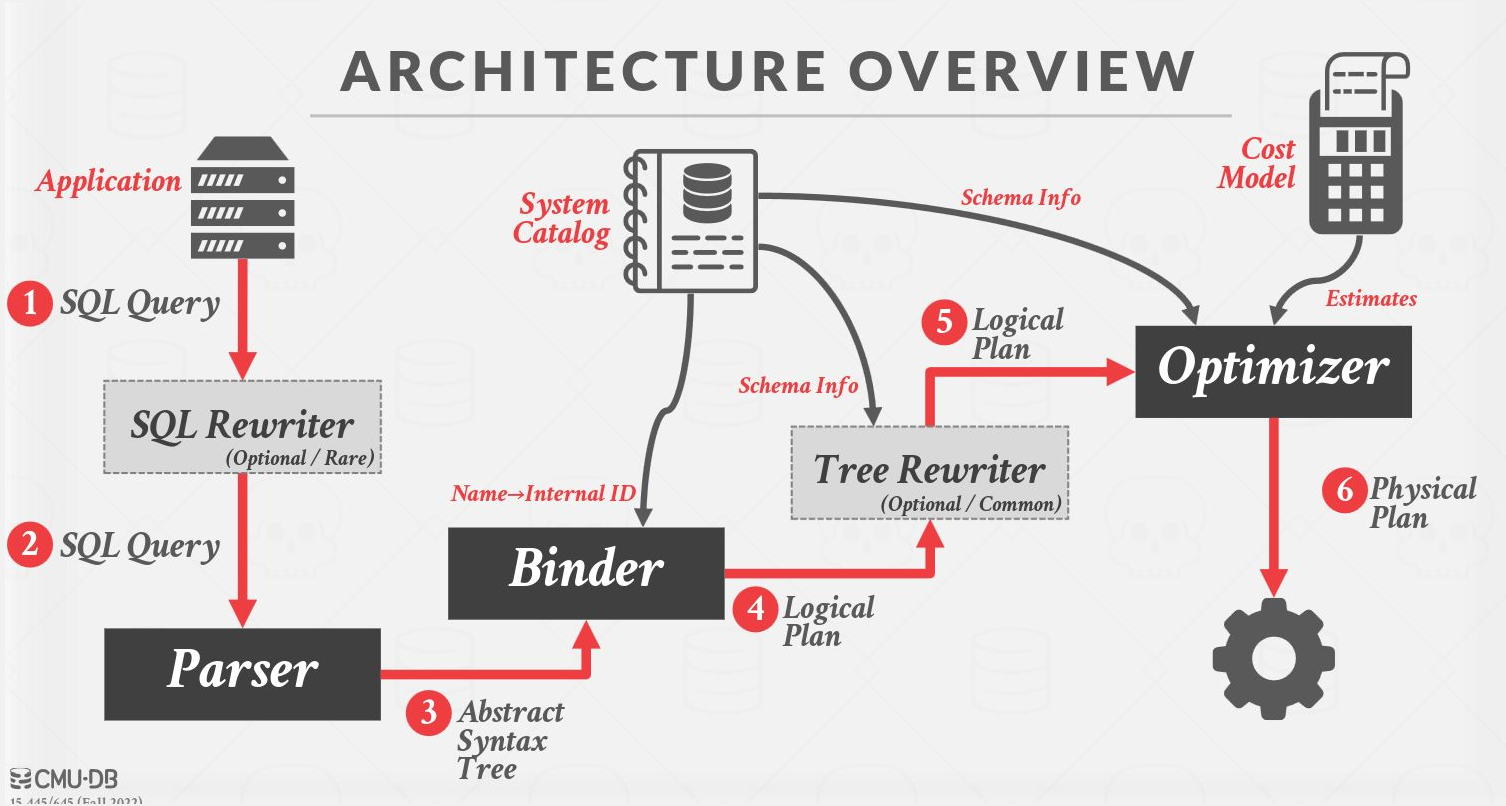
```
for pk in Index1(predicate) + Index2(predicate):
    row = get_record(pk)
    if match(row, other_conditions):
        yield row
```

Demo: Shared documents

Query optimization

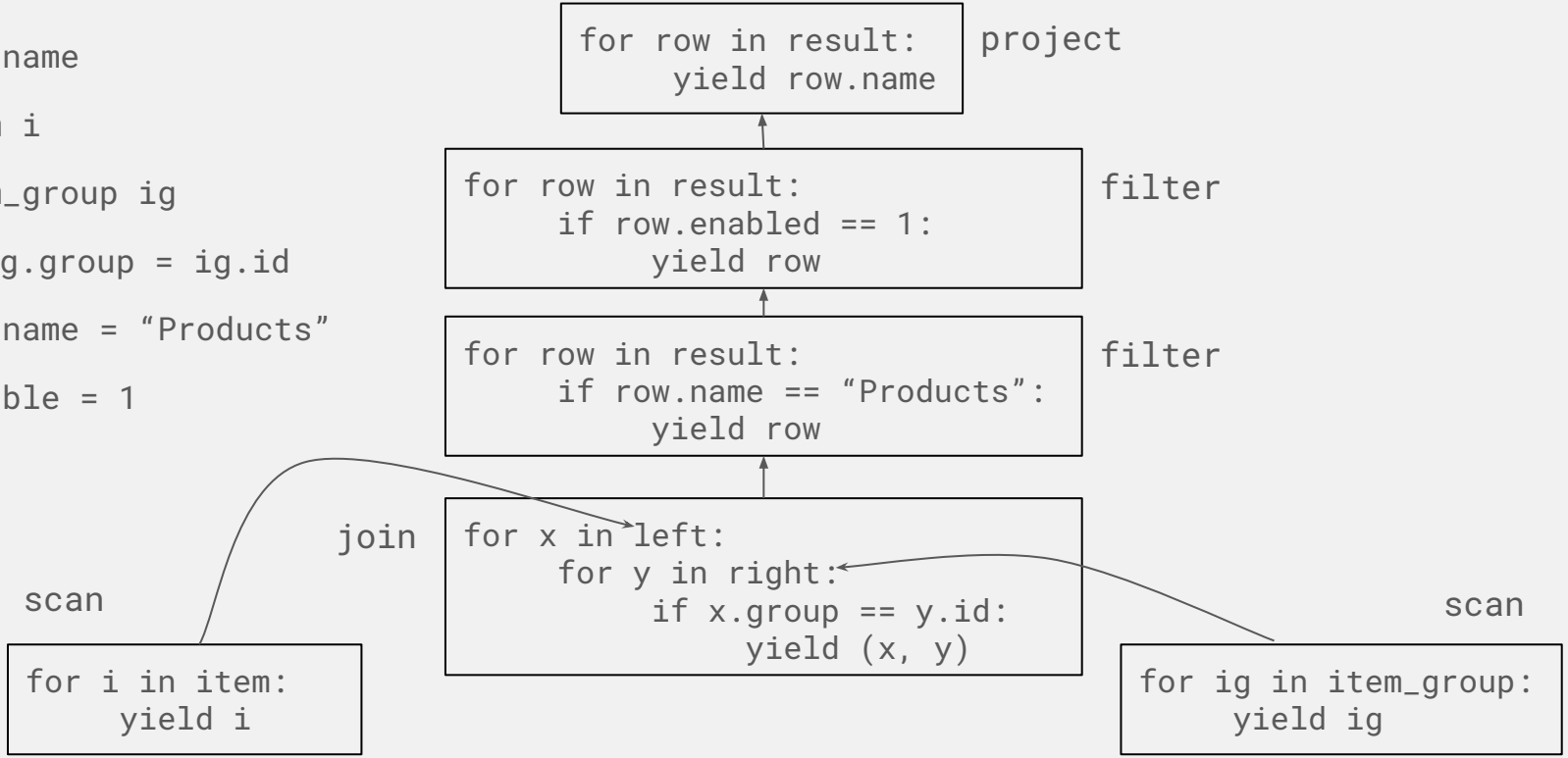
“Find **correct** execution plan with lowest cost”

Query optimization



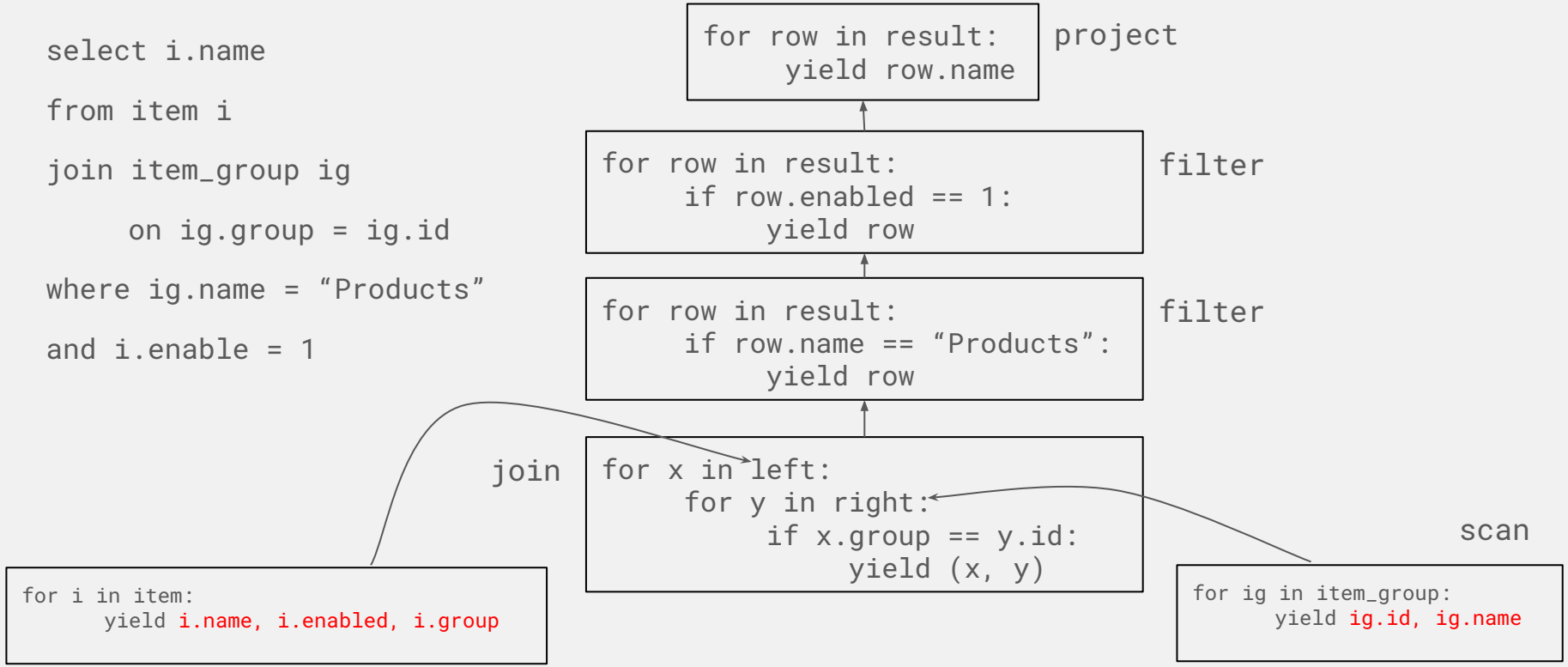
Logical Plan Optimization (~ Rules)

```
select i.name
from item i
join item_group ig
     on ig.group = ig.id
where ig.name = “Products”
and i.enable = 1
```



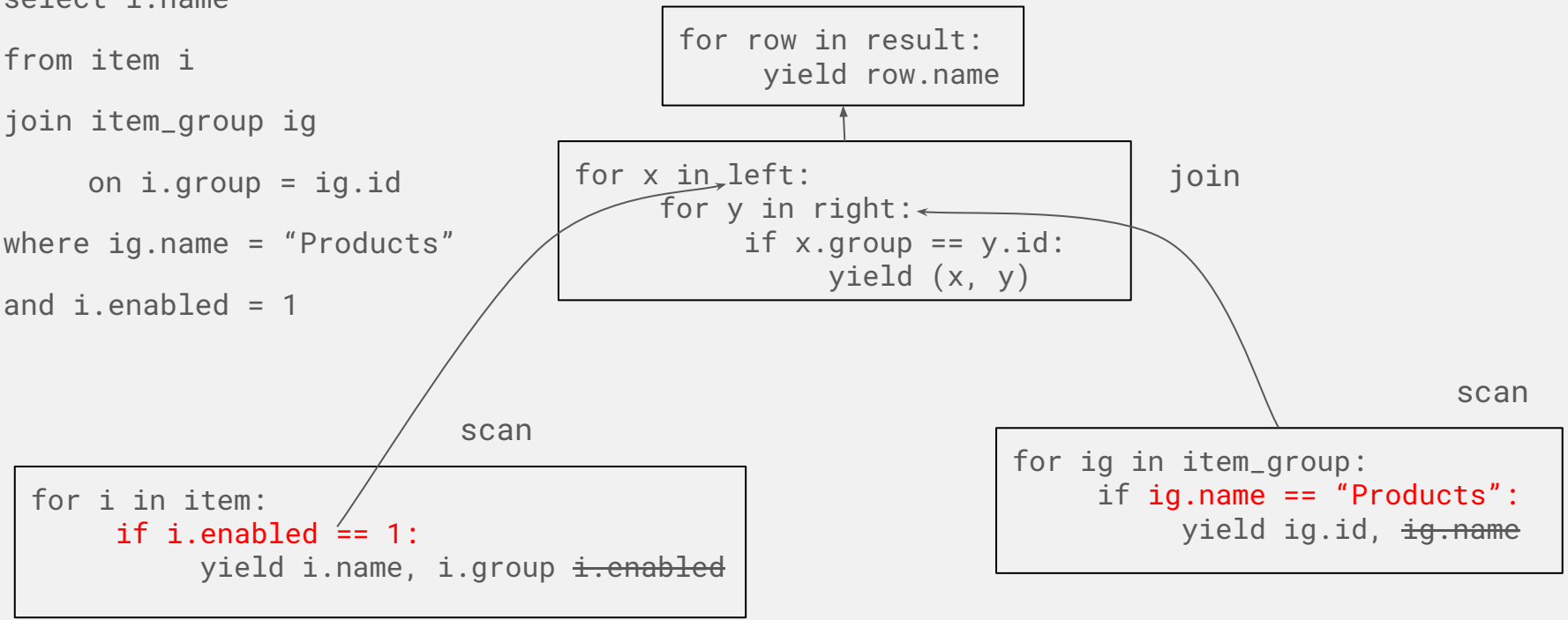
Projection Pushdown

```
select i.name
from item i
join item_group ig
     on ig.group = ig.id
where ig.name = “Products”
and i.enable = 1
```



Predicate Pushdown

```
select i.name
from item i
join item_group ig
     on i.group = ig.id
where ig.name = “Products”
and i.enabled = 1
```



SubQuery - Dumb execution

```
select i.name
from item i
where i.enable = 1 and exists (
     select *
     from item_group g
     where g.id = i.group
     and g.name = “Products”
)
```

SubQuery rewrite

```
select i.name
from item i
where i.enable = 1 and exists (
     select *
     from item_group g
     where g.id = i.group
     and g.name = “Products”
)
```

```
select i.name
from item i
join item_group ig
     on ig.group = ig.id
where ig.name = “Products”
and i.enable = 1
```

SubQuery rewrite - constant evaluation

```
select i.name
from item i
where
     i.enable = 1
     and item.group = (
          select g.id
          from item_group g
          where g.name = “Products”
     )
```

SubQuery rewrite - constant evaluation

```
select i.name
from item i
where
     i.enable = 1
     and item.group = (
          select g.id
          from item_group g
          where g.name = “Products”
     )
```

```
select i.name
from item i
where
     i.enable = 1
     and item.group = (
          5
     )
```

Cost based optimization

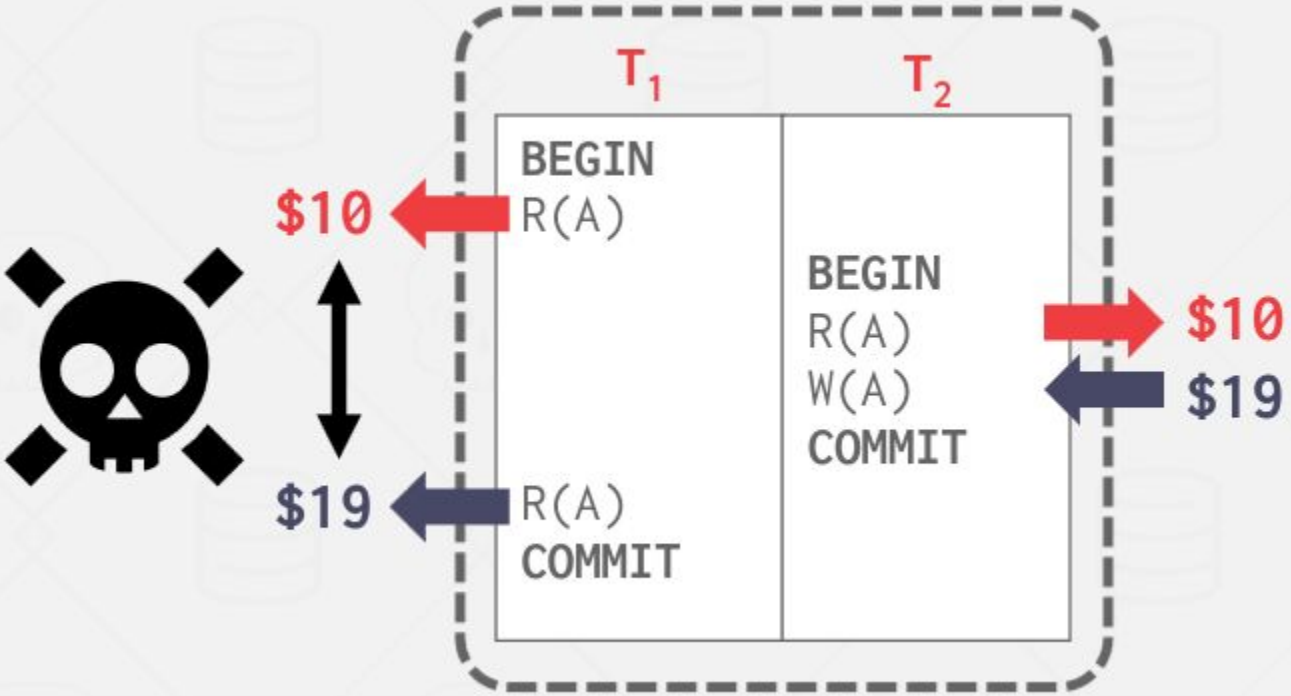
```
SELECT *
from `tabStock Ledger Entry`
where
    item_code = ‘X’
    and warehouse = ‘Y’
    and posting_datetime > ‘2024-01-01’
order by `posting_datetime` desc
limit 1;
```

- Cardinality
- Distribution

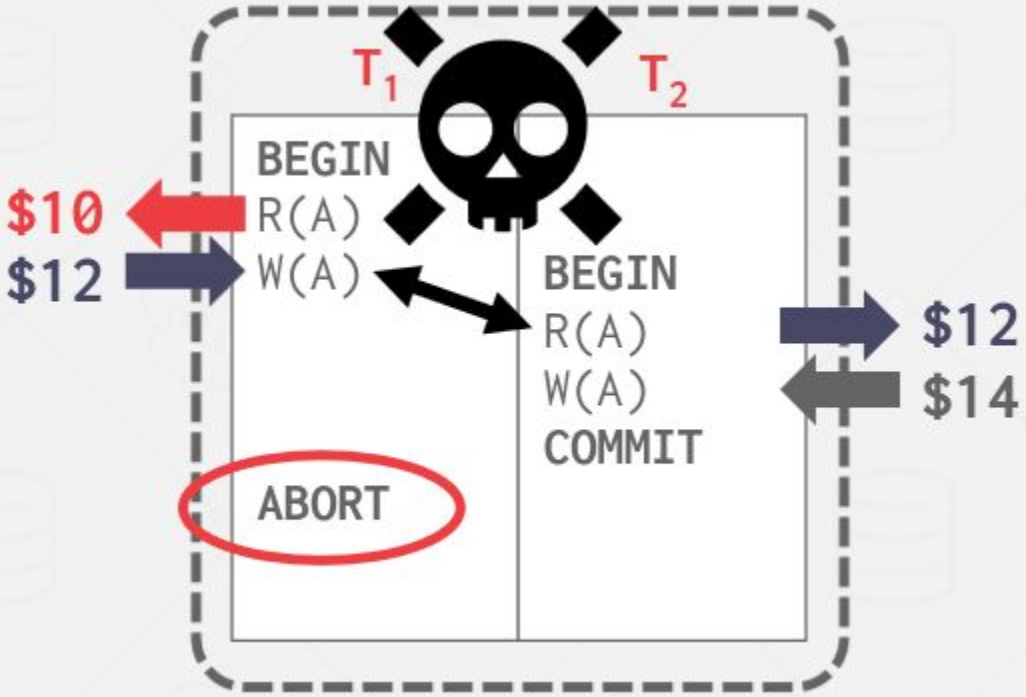
Outline

- ~~Storage~~
- ~~Bufferpool~~
- ~~Indexing (B+Tree)~~
- ~~Query planner and execution~~
- Concurrency, locking and MVCC
- Logging and recovery

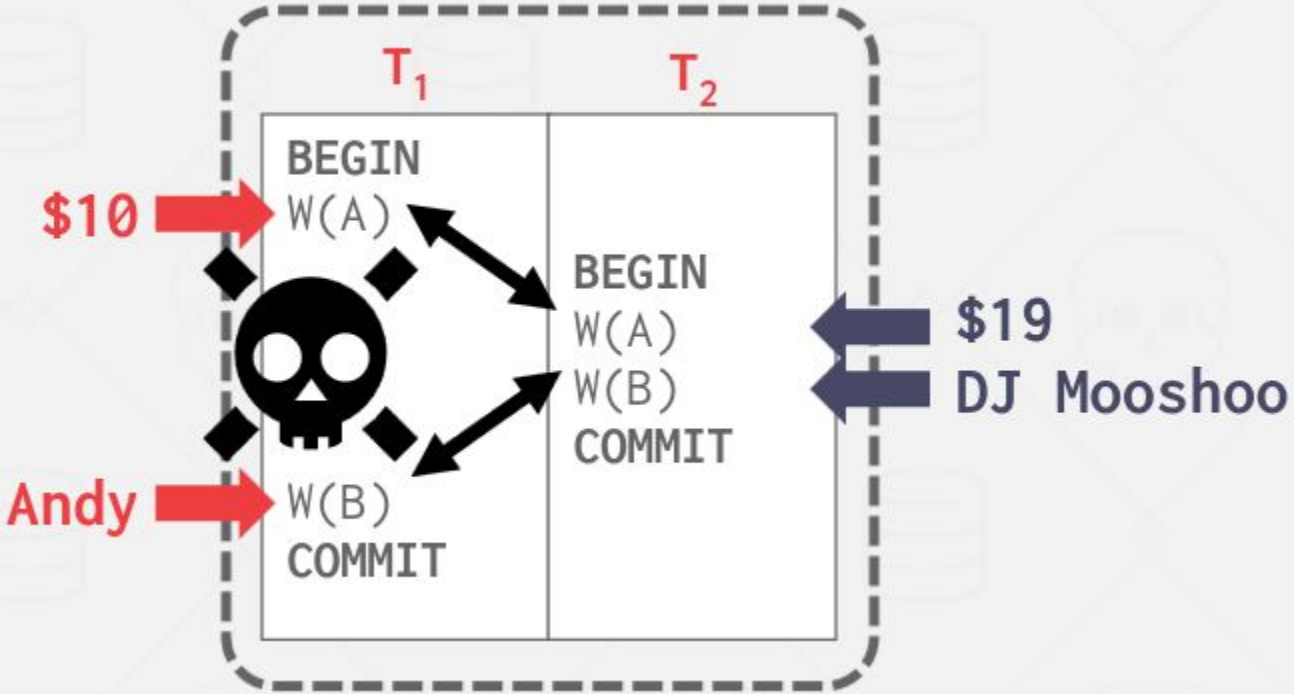
Concurrency conflicts - Unrepeatable read



Concurrency conflicts - Dirty Read



Concurrency conflicts - Lost update



ACID

Atomicity - everything happens or nothing.

Consistency - “stays correct” e.g. constraints

Isolation - Transactions are isolated from one another

Durability - If I commit, change is persisted for sure.

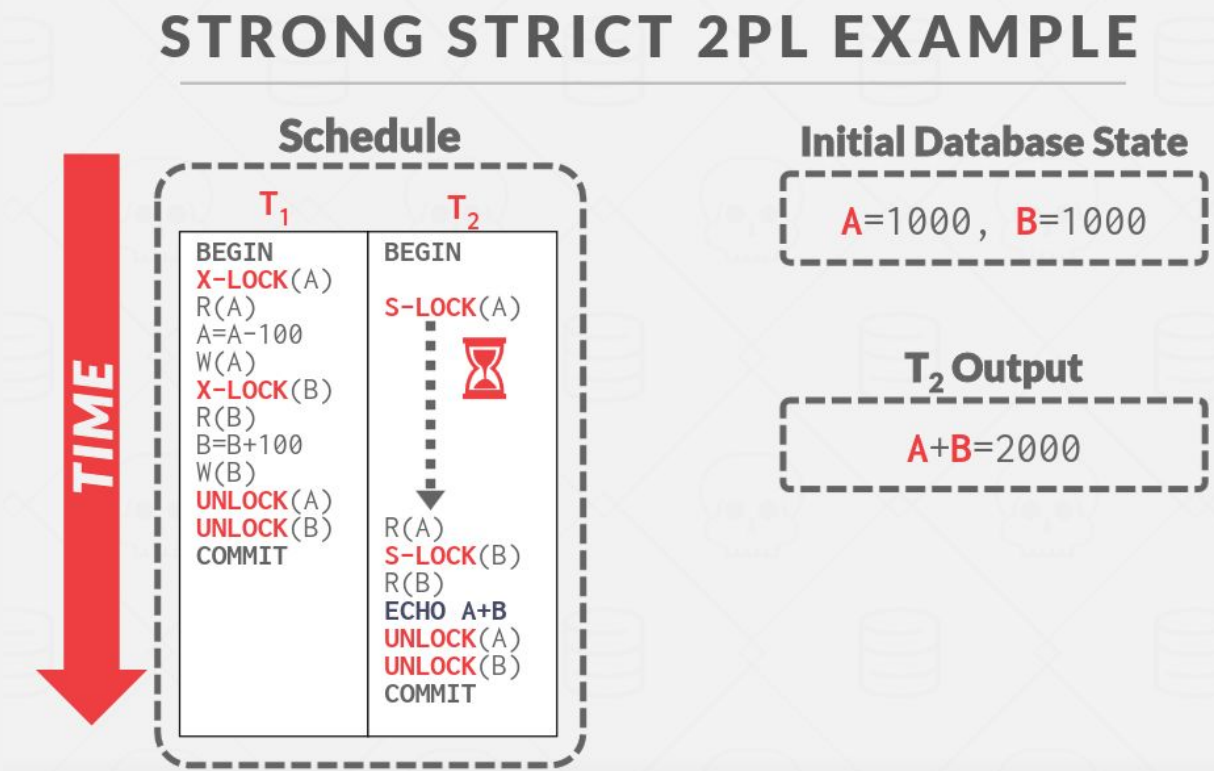
Basic Locking

S-Lock - **S**hared Lock

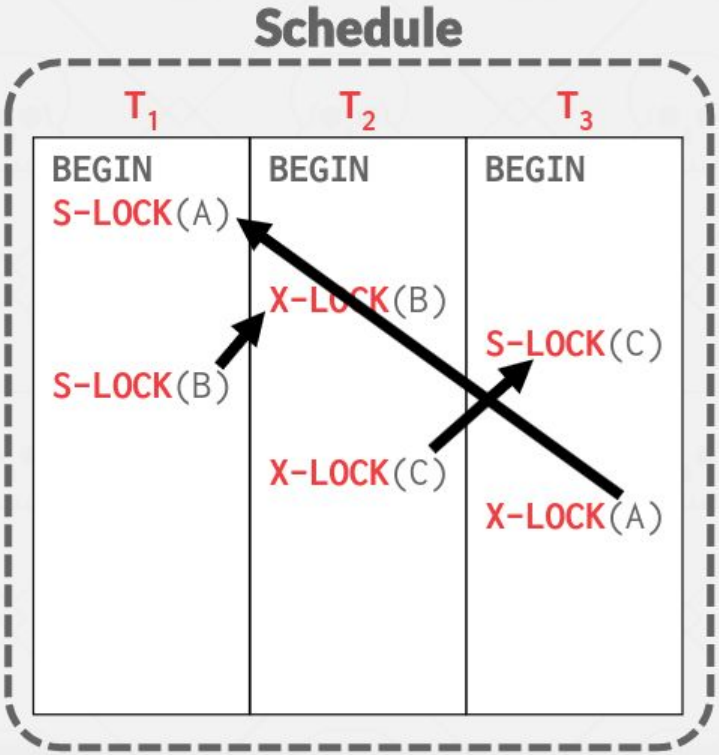
X-Lock - E**X**clusive Lock

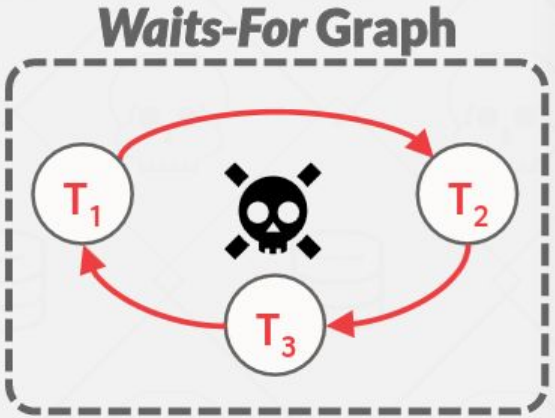
Strict Two Phase Locking

Strict Two Phase Locking

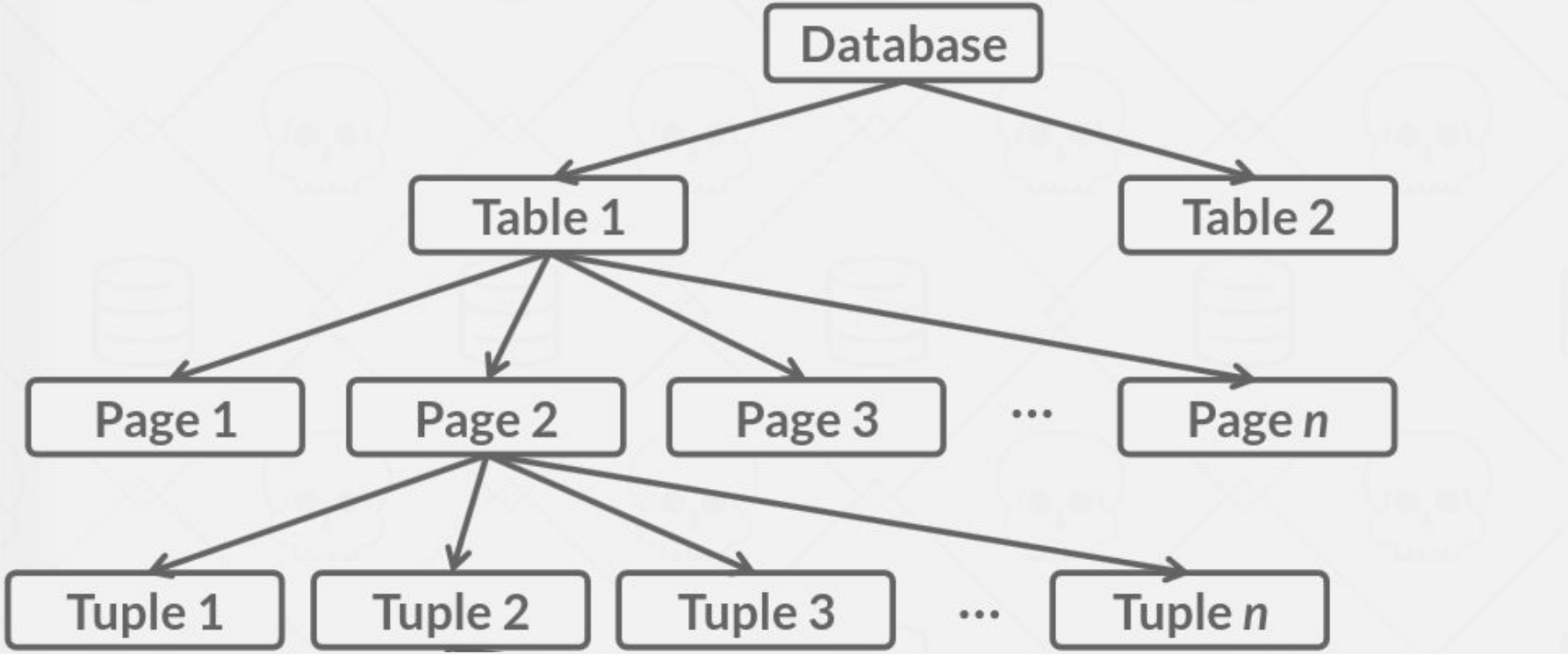


Deadlocks


Schedule
T1
T2
T3
BEGIN
BEGIN
BEGIN
S-LOCK(A)
X-LOCK(B)
S-LOCK(C)
S-LOCK(B)
X-LOCK(C)
X-LOCK(A)


Waits-For Graph
T1
T2
T3

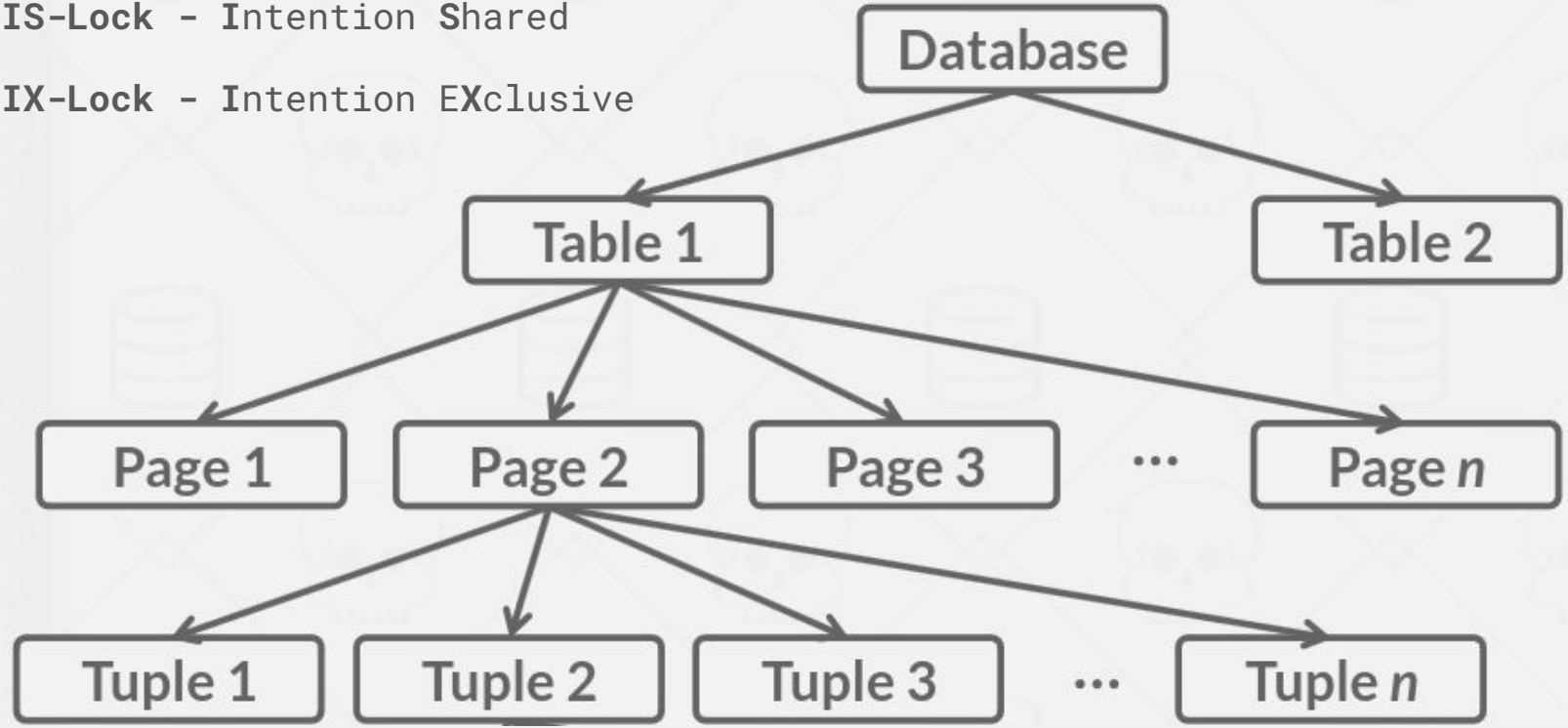
Intention Locks



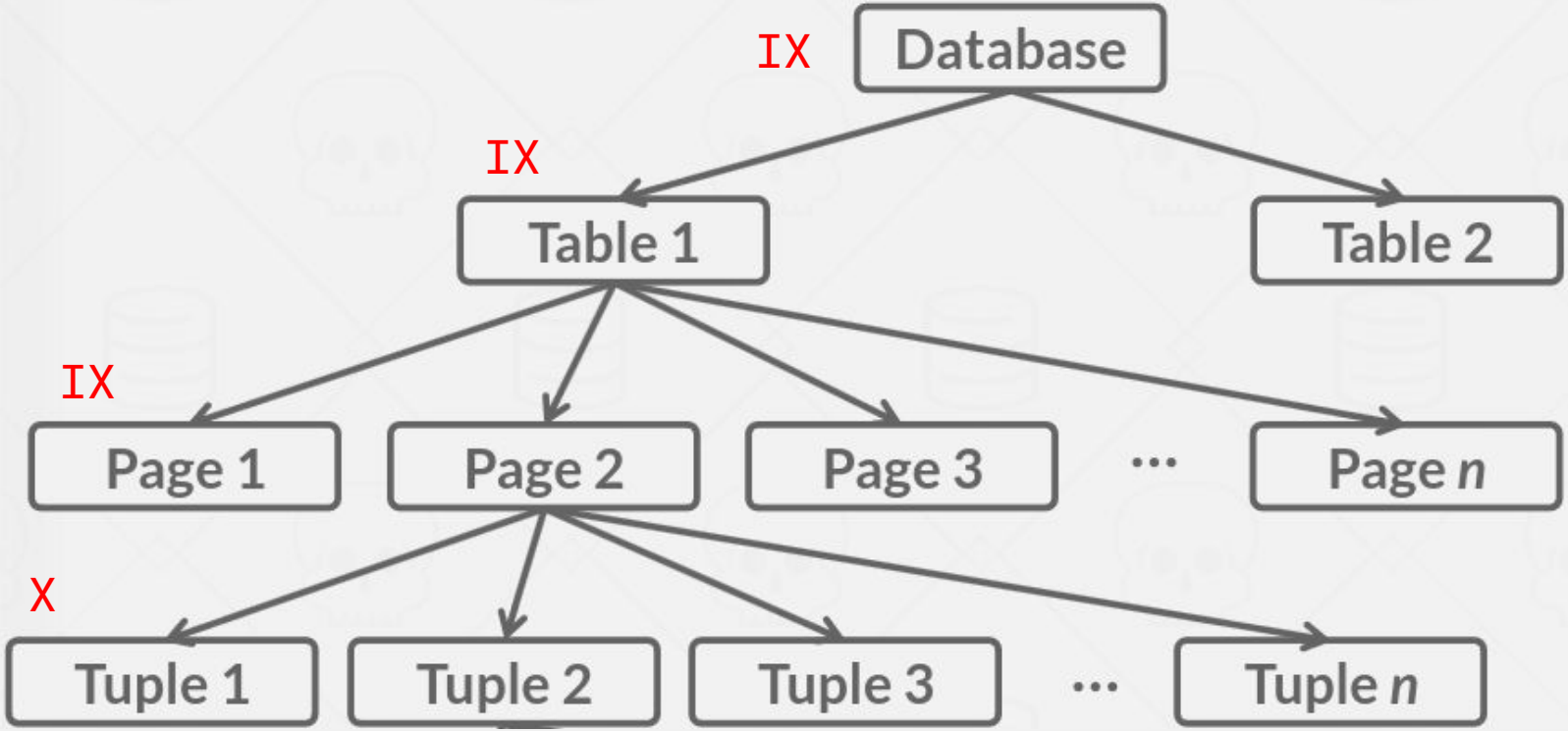
Intention Locks

IS-Lock - **I**ntention **S**hared

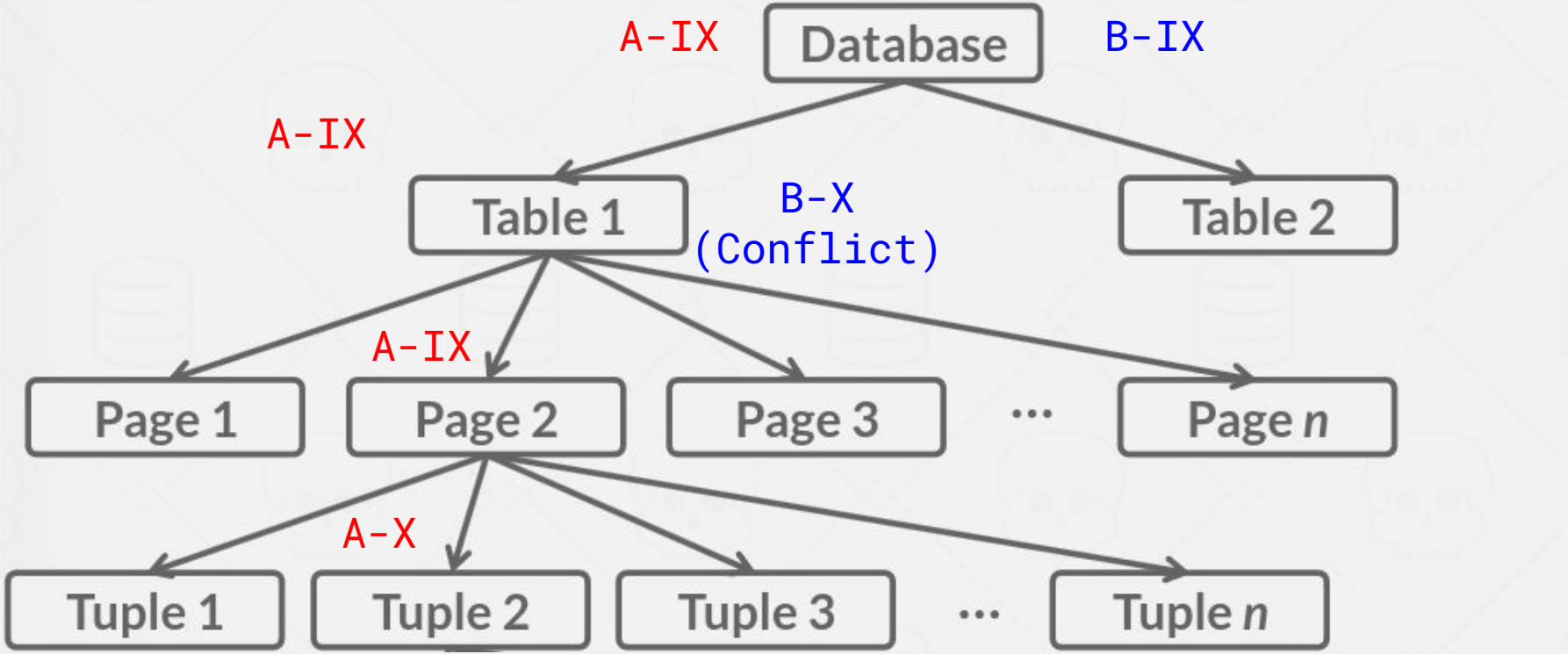
IX-Lock - **I**ntention E**X**clusive



Intention Locks - Write to a tuple



Intention Locks - Alter table



Intention Locks - Compatibility matrix

	X	IX	S	IS
X	Conflict	Conflict	Conflict	Conflict
IX	Conflict	Compatible	Conflict	Compatible
S	Conflict	Conflict	Compatible	Compatible
IS	Conflict	Compatible	Compatible	Compatible

Practical Locking

Operation	**Database**	**Table**	**Rows**
FOR UPDATE	IX	IX	X
UPDATE/DELETE	IX	IX	X
LOCK IN SHARE MODE	IS	IS	S
ALTER TABLE	IX	X	X
SELECT	-	-	-

These locks solve everything?

```
select es.employee, max(es.salary)
from `tabEmployee Salary` es;
```

These locks solve everything?

```
select es.employee, max(es.salary)
from `tabEmployee Salary` es;
```

```
insert into `tabEmployee Salary`
(employee, salary)
values (“Ankush”, 40)
```

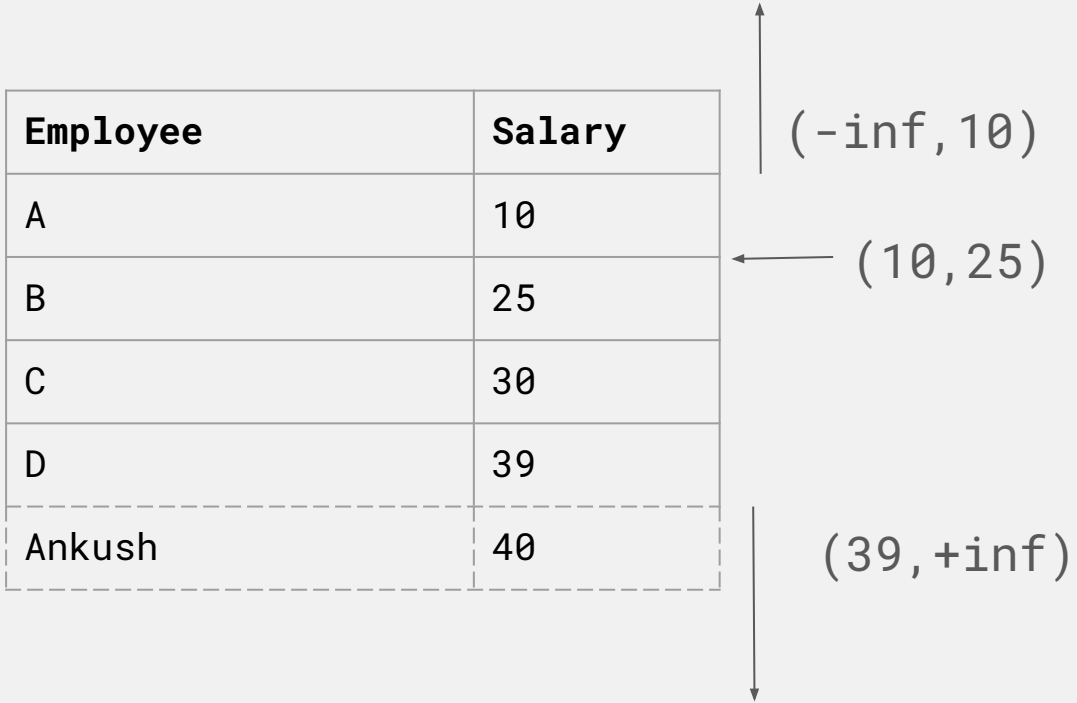
Locking things that don’t exist?

Employee	Salary
A	10
B	25
C	30
D	39
Ankush	40

Gap locks

Practically:

“Lock everything you read + gaps + extremes”



Gap locks

Practically:

“Lock everything you read + gaps + extremes”

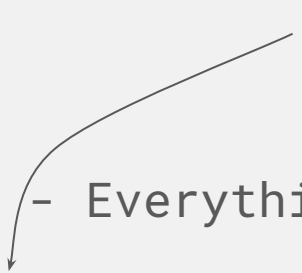
Employee	Salary
A	10
B	25
C	30
D	39
Ankush	40

(-inf,10)

(10,25)

(39,+inf)

Isolation Levels



Serializable - Everything is executed as if serialized

Repeatable Reads - Txn see “**snapshot**” of data at start

Read Committed - Txn can read already committed data.

Read Uncommitted - Txn see uncommitted writes

Multi **V**ersion **C**oncurrency **C**ontrol

- Every write creates a new “timestamped” version
- Every read query sees a version “**as of**” txn start
- MySQL keeps multiple copies of each rows
- InnoDB purges old copies when there no transaction older than that

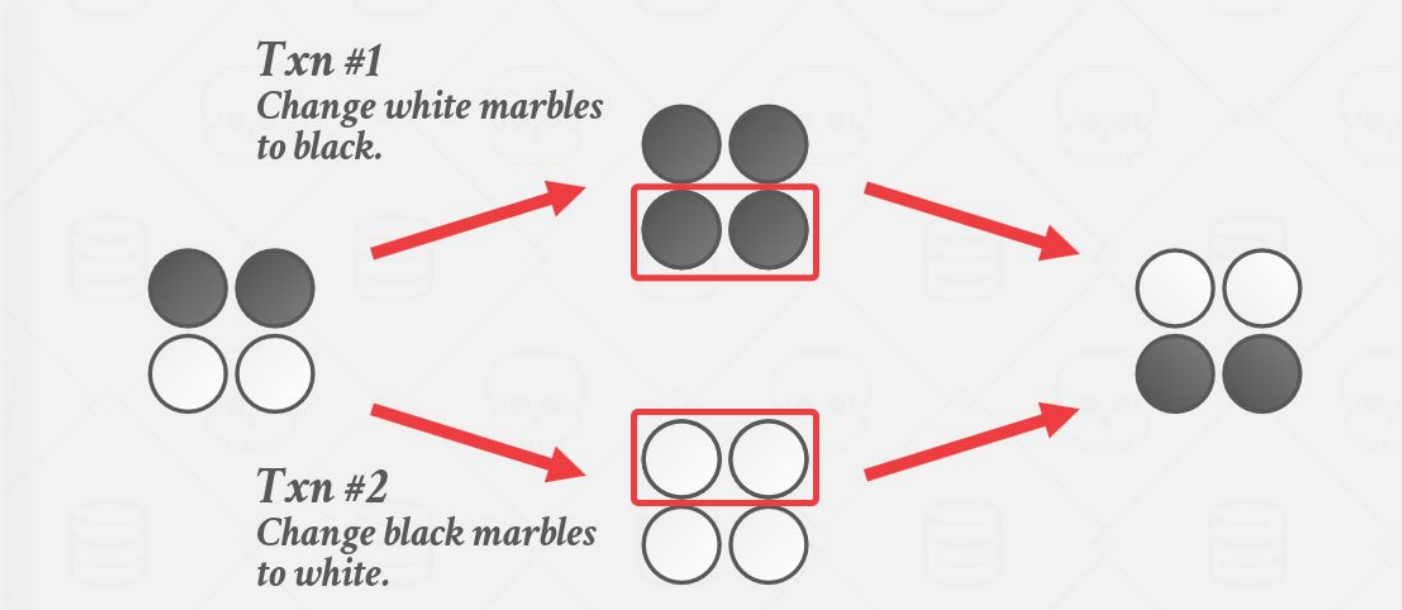
MVCC + Repeatable Read Demo

Practical Implications

- How do you think MySQLDump is consistent? `--quick` `--single-transaction`
- Why backups slow things down?
- Why long running updates slow things down?

Repeatable Read Problems

- Performance - “snapshotting”
- New type of anomaly - “**Write Skew**”



Write Skew in Practice

- User A and B create 10rs payment each

Txn A	**Txn B**
`paid_amt = sum(payment_entry.amt)` `> 0`	
`paid_amt += 10` `> 10`	`paid_amt = sum(payment_entry.amt)` `0`
	(Blocked because of write lock)
`COMMIT`	`paid_amt += 10` `> 10`
	`COMMIT`

Write Skew in Practice

- Fix: “**FOR UPDATE**” bypasses snapshots

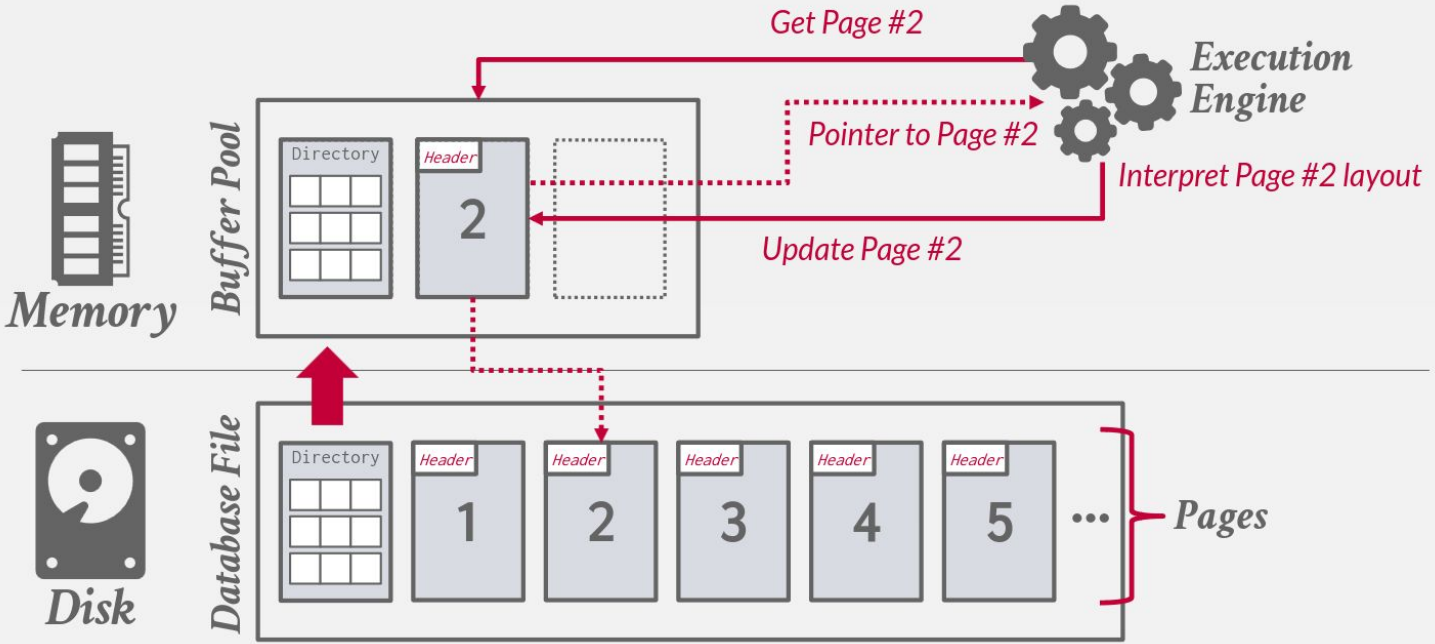
Txn A	**Txn B**
`paid_amt = sum(payment_entry.amt)` `> 0`	
`paid_amt += 10` `> 10`	`paid_amt = sum(payment_entry.amt)` `0`
	(Blocked because of write lock)
`COMMIT`	`paid_amt += 10` `> 10`
	`COMMIT`

Outline

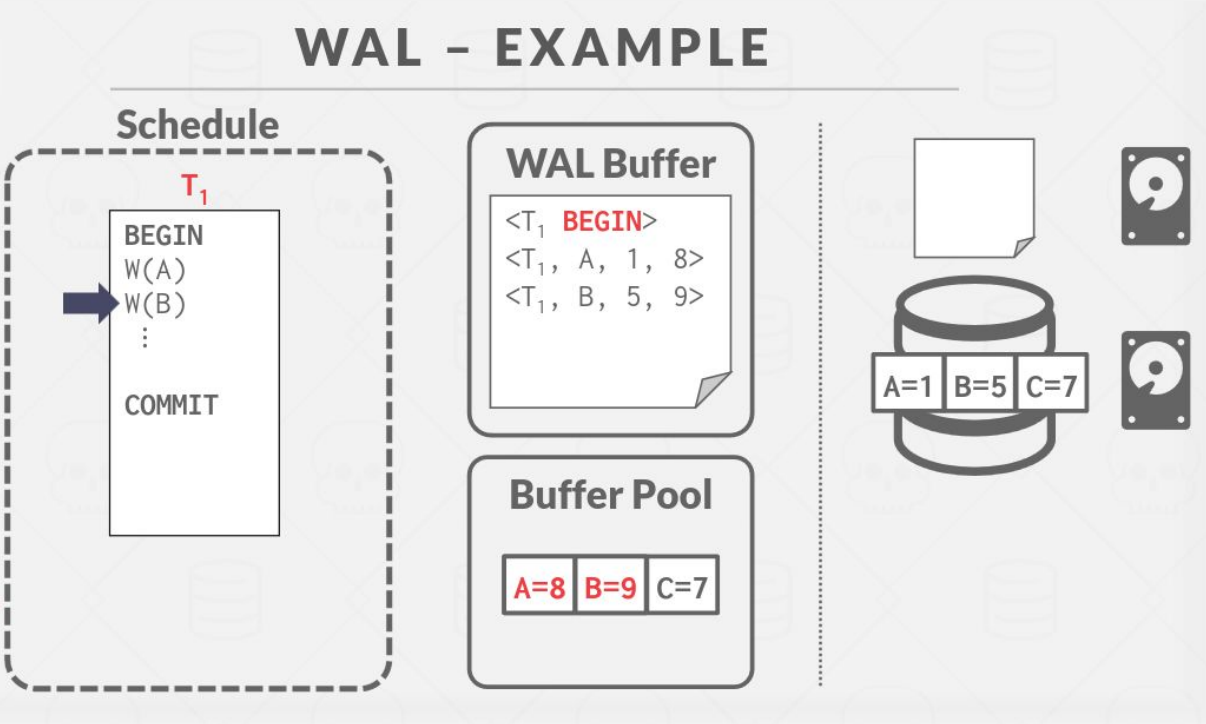
- ~~Storage~~
- ~~Bufferpool~~
- ~~Indexing (B+Tree)~~
- ~~Query planner and execution~~
- ~~Concurrency, locking and MVCC~~
- Logging and recovery

Logging

- How are dirty pages written to disk?



Write Ahead Log (WAL) aka redo log



Durability = Ensure WAL is flushed to disk on commit.

Checkpointing

- Flush all dirty pages to disk
- Truncate WAL.
- During crash recovery: replay writes from last checkpoint.

Practical Implications

- Is it safe to `kill -9` MySQL?
- Checkpointing data on monitor
- Our logging config - mariadb.cnf

Outline

- ~~Storage~~
- ~~Bufferpool~~
- ~~Indexing (B+Tree)~~
- ~~Query planner and execution~~
- ~~Concurrency, locking and MVCC~~
- ~~Logging and recovery~~

Practical Indexing

Go read

https://use-the-index-luke.com/

Questions?
Feedback?