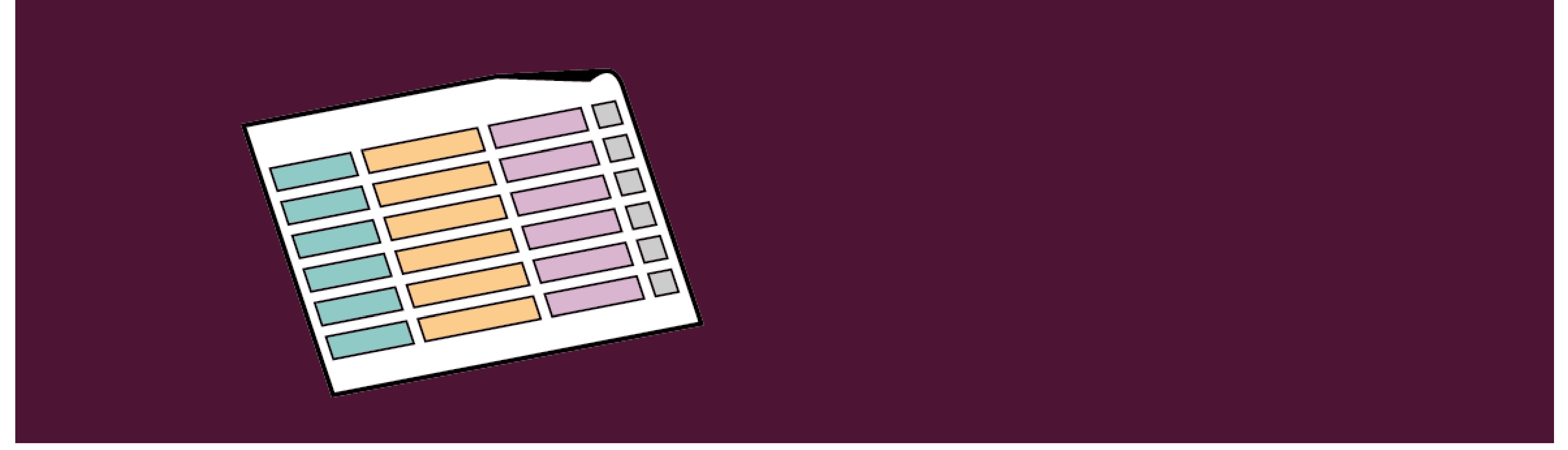
A DEEP DIVE INTO A DATABASE ENGINE INTERNALS

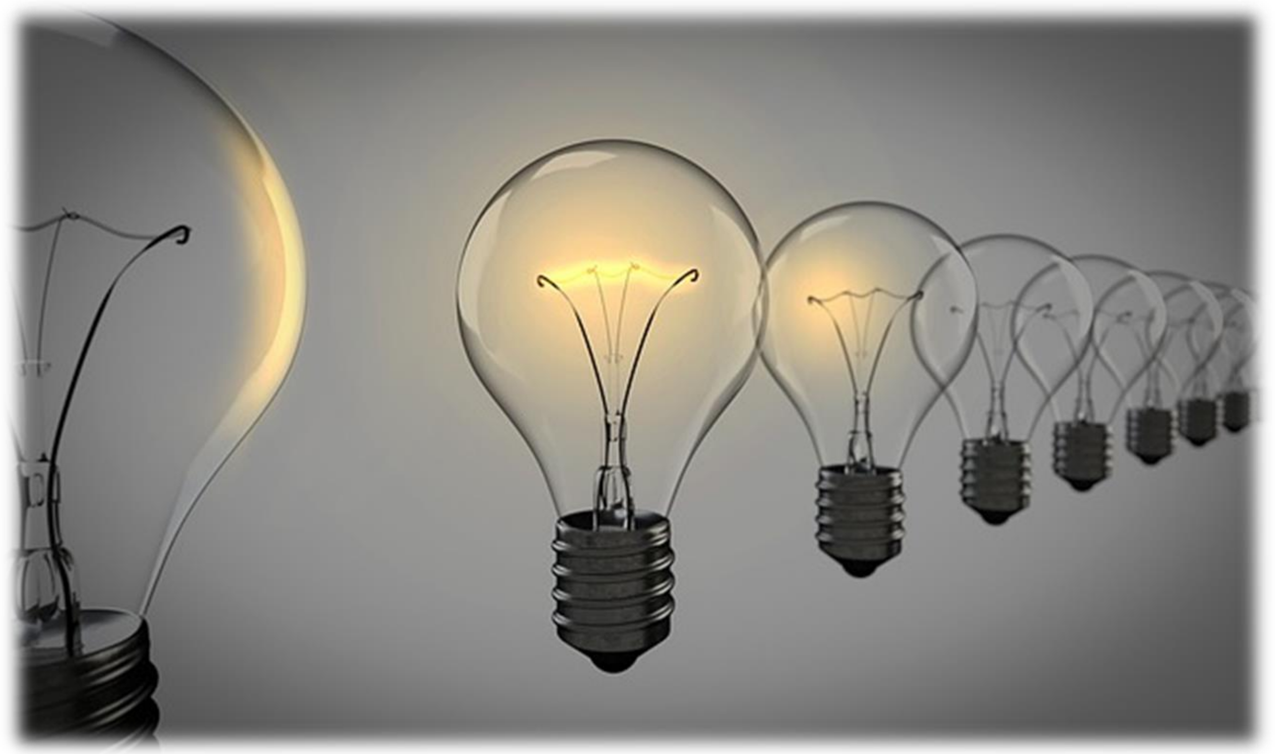
OREN EINI

OREN@RAVENDB.NET



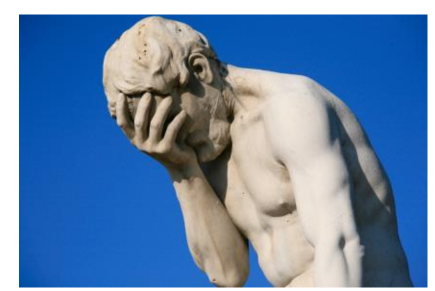
STRUCTURE

- Part 1: Internal structure
- Part 2: Durability
- Part 3: Transactions & Concurrency
- Part 4: Tricks & Optimizations



I BUILD DATABASES FOR A LIVING

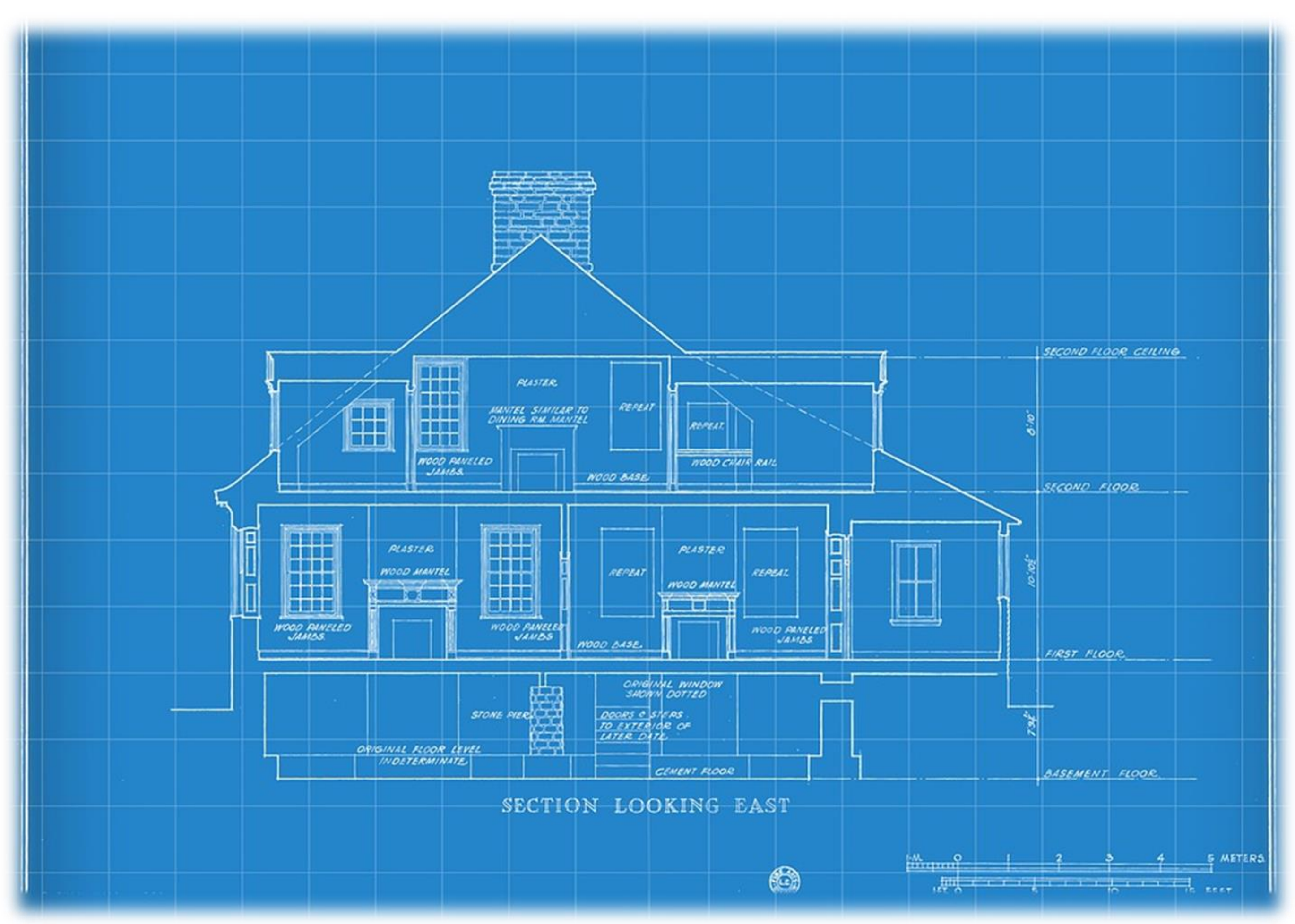
- Interview question:
  - Build a persistent phone book



- Database is a black box, full of magic

WHAT CONSTITUTE A DATABASE?

- Store and retrieve data
  - Format of the data
- Resource management
- Transactions
  - ACID
- Query engine



LET'S BUILD A DATABASE

- What do we need for a phone book application?

```
Name             ,   Phone
Allison Neu      ,   555-8396
Bob Newhall      ,   555-4344
Canine College   ,   555-7201
Canine Therapy   ,   555-2849
```

CSV FOR THE WIN!!!

- Easy to work with
- Human readable

- How do you search?
  - O(N)
- How do you modify a record?
  - Rewrite the whole file

IN MEMORY ALGORITHMS BADLY SUITED FOR PERSISTENCE

- Sorted data structures:
  - AVL Tree
  - Skip Lists

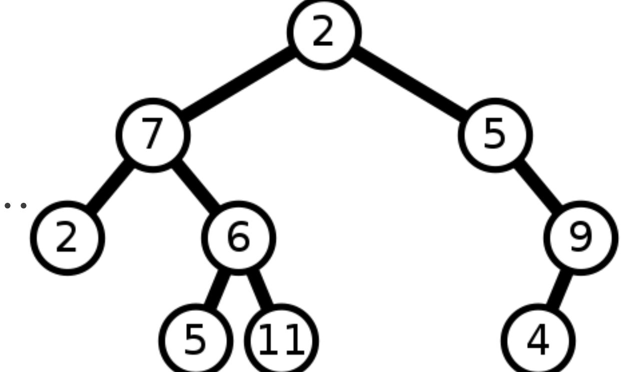
- O(logN)
- Optimal *

- * Assuming same access speed for each datum

Hardware	Latency (4KB read)
DRAM	50 nanoseconds
NVMe (Optane)	100 microseconds 100,000 nanoseconds
SSD (Kingston)	500 microseconds 500,000 nanoseconds
Hard Disk	8 milliseconds 8,000,000 nanoseconds

SEEK TIME DOMINATES PERSISTENT DATA STRUCTURES

- Cost of finding a value?
  - O(log2N)
- But what is N?
  - Random read
- Let's do the math…

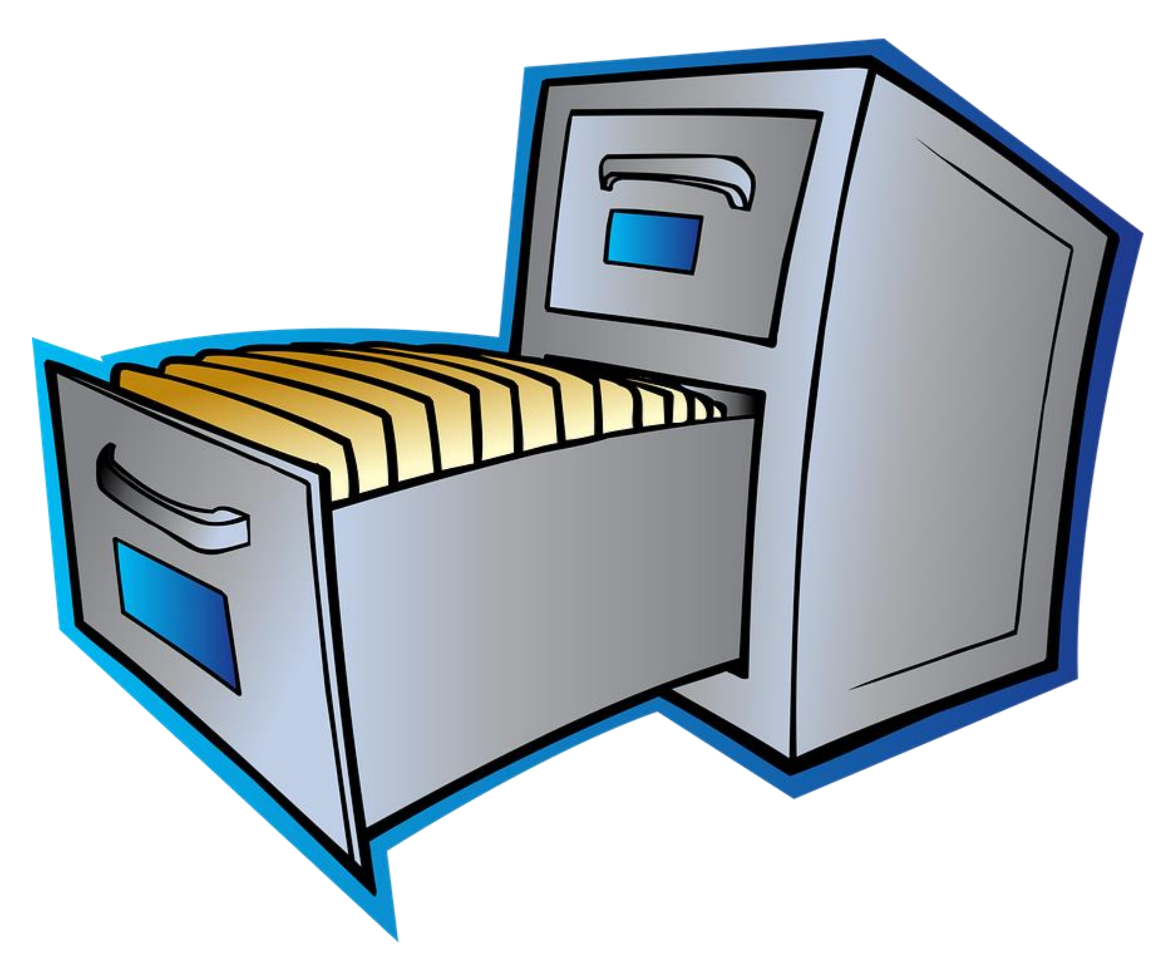


Count	Search time (log2N)	Seeks (SSD)	Seeks (HDD)
1,000	10	5 ms	80 ms
1,000,000	20	10 ms	800 ms

BATCH ACCESS TO MEMORY: PAGES

- Internal space management in the file
- Divide to pages (common 4KB – 4MB)
- Pages are loaded to memory as a single unit
- Modified in memory

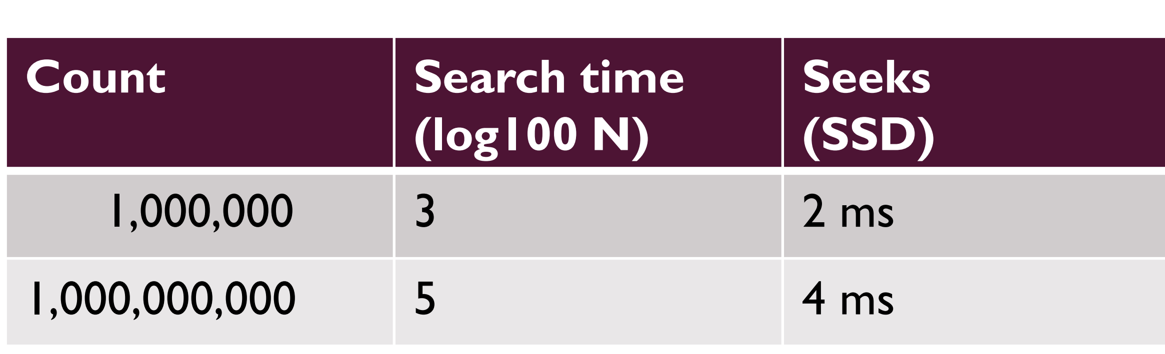
- Locality of reference as a core concept



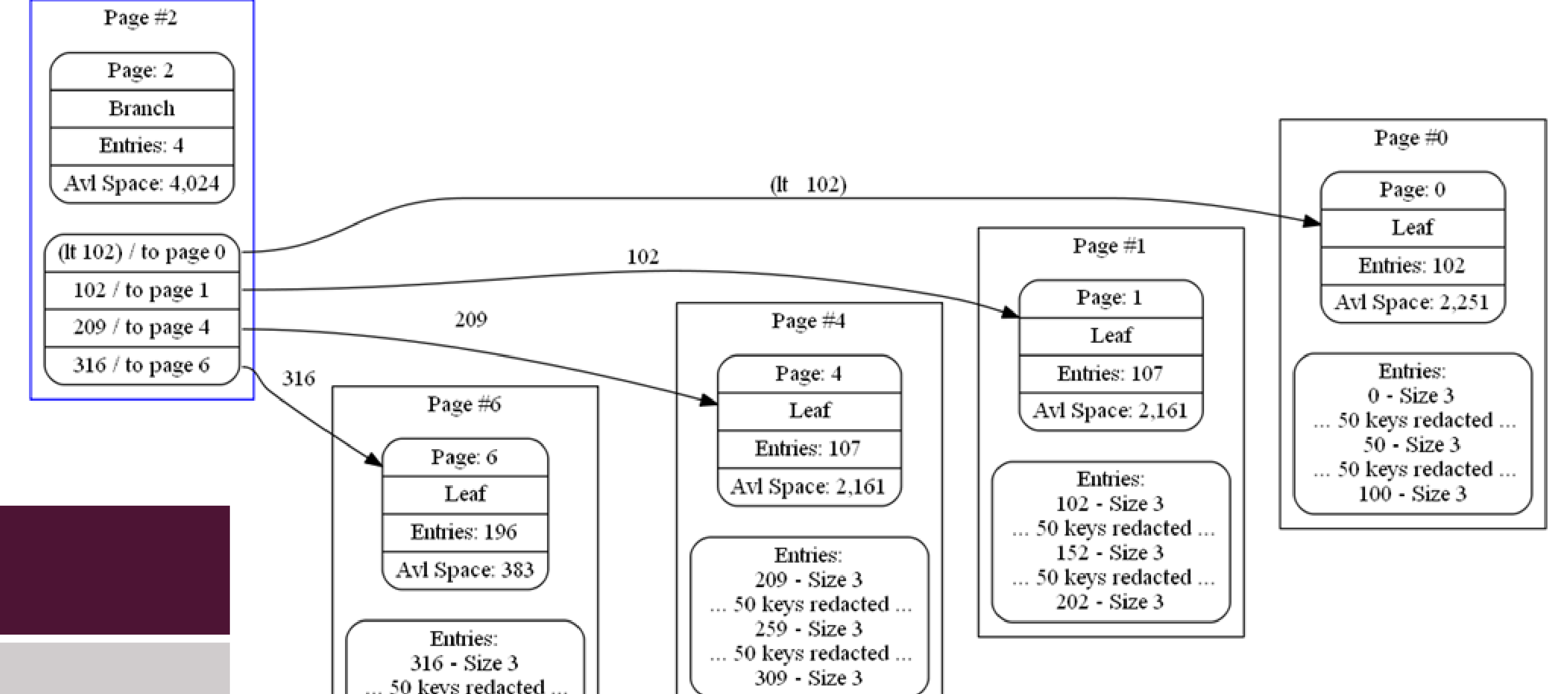
DISK OPTIMIZED ALGORITHM: B+TREE

The Ubiquitous B-Tree - 1979

- Seeks are expensive, so let's avoid them.
- Bring many results in one disk seek.
- Cost is now (O(logF N) + O(log2 F))
  - Where F is fill factor
- Assume F is 100
  - Searching in page is free



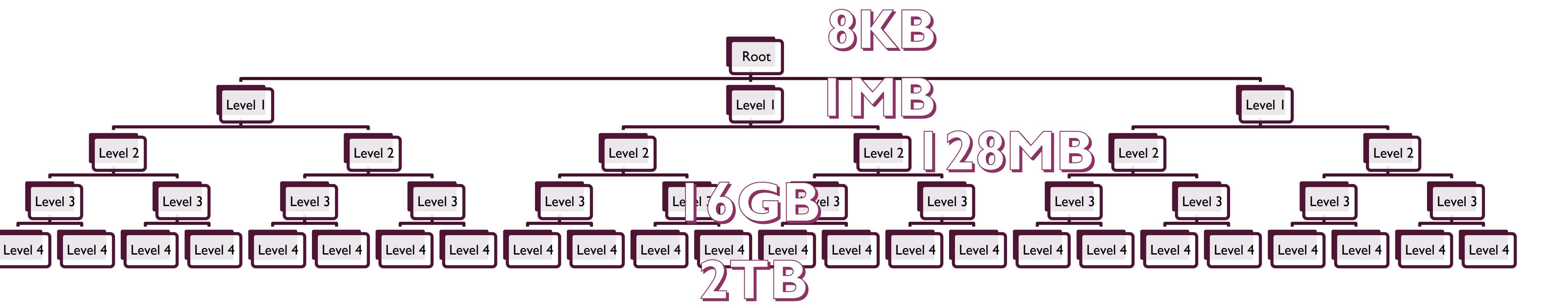
Count	Search time (log100 N)	Seeks (SSD)
1,000,000	3	2 ms
1,000,000,000	5	4 ms



MEMORY HIERARCHY

Count	Search time (log100 N)	Seeks (HDD)
1,000,000	3	24 ms
1,000,000,000	5	40 ms

Fanout: 128

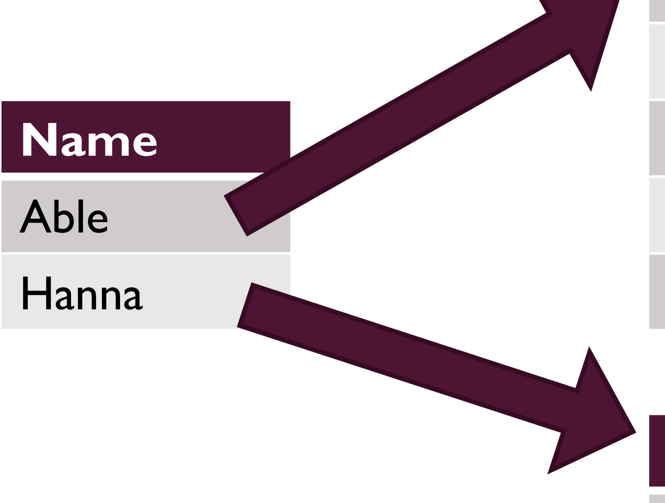


Cache: 128MB

Seeks: 2 – Cost 16 ms (hdd)

WHAT IS A PAGE SPLIT? 1/2

Name
Able
Brett
Cannes
Derek
Erik
Frank
George

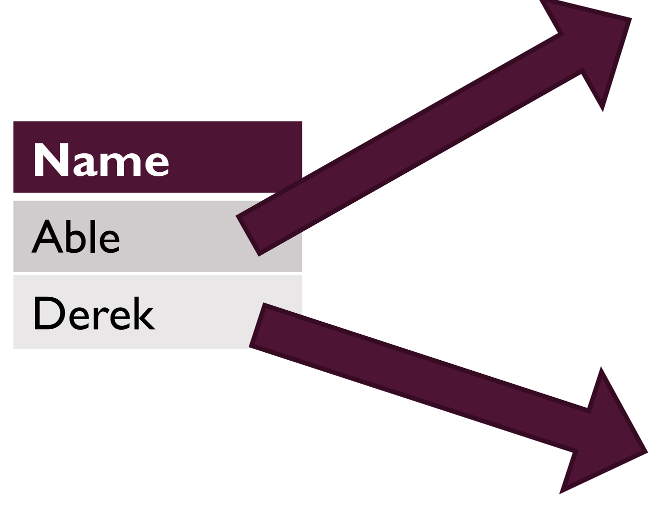


Name
Able
Brett
Cannes
Derek
Erik
Frank
George

Name
Hanna
6 x empty

WHAT IS PAGE SPLIT? 2/2

Name
Able
Brett
Cannes
Derek
Erik
Frank
George



Name
Able
Ben
Brett
Cannes
2 x empty

Name
Derek
Erik
Frank
George
2 x empty

THE IMPORTANCE OF ORDER

Root

A,B
C,D,E
F

A1,B2
C1,D2
E1,E2
F1, F2

A1,A2,B1
B2
C2,D1
D2

Root

A,B,C
D,E,F

A1,A2
B1,B2
C1,C2
D1,D2,
E1,E2
F1,F2

WHAT DO YOU PUT IN THE B+TREE?

Pos	Data
20	Email,Name, etc (Cannes)
31	Email,Name, etc (Able)
55	Email,Name, etc (Brett)
80	Email,Name, etc (Derek)

Clustered

Embedded data

C

A,B

C,D

Able (email, name, etc)

Brett (email, name, etc)

Cannes(email, name, etc)

Derek(email, name, etc)

References

A,B,C,D

Able (31)

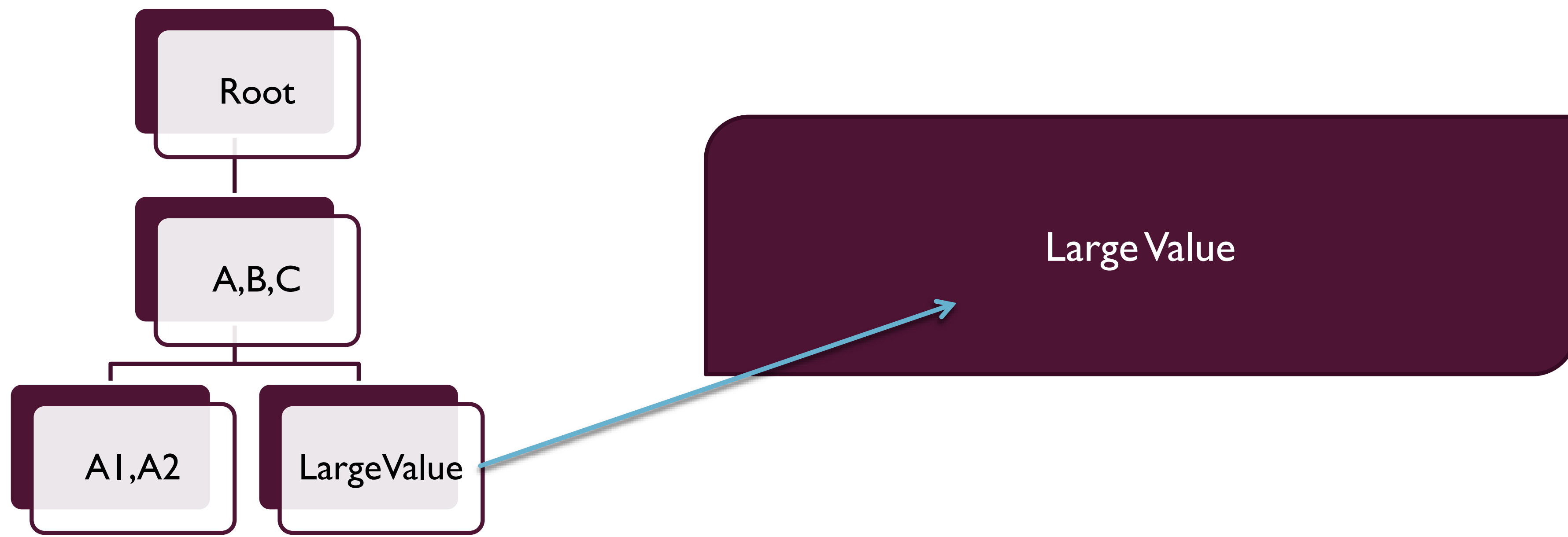
Brett(55)

Cannes(20)

Derek(80)

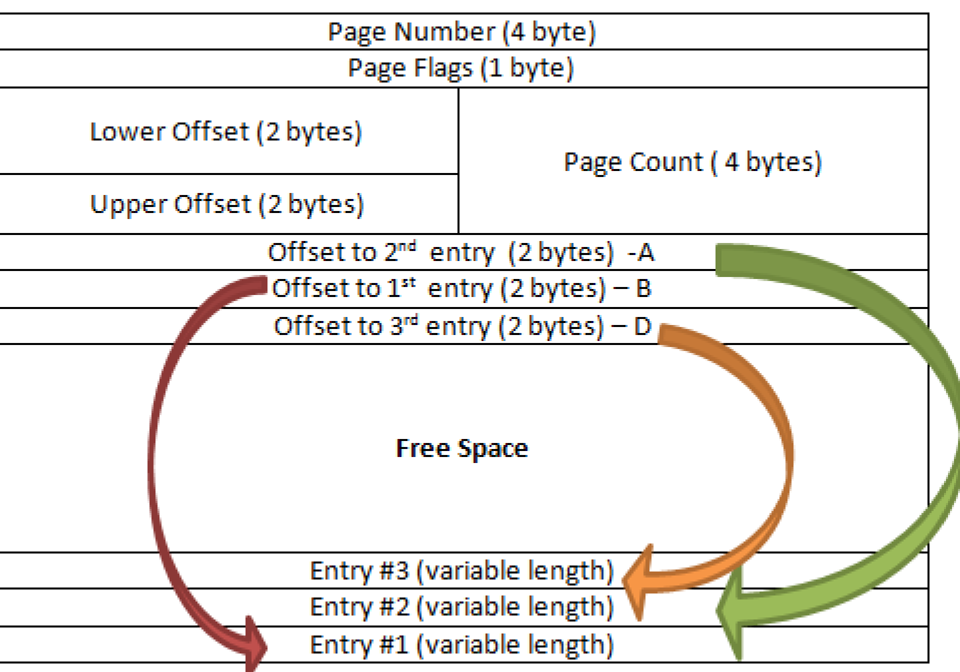
DEALING WITH LARGE VALUES…

- Overflow – If value too big to put in B+Tree page
- Allocate N pages to fit it.
- Record pointer to it from B+Tree



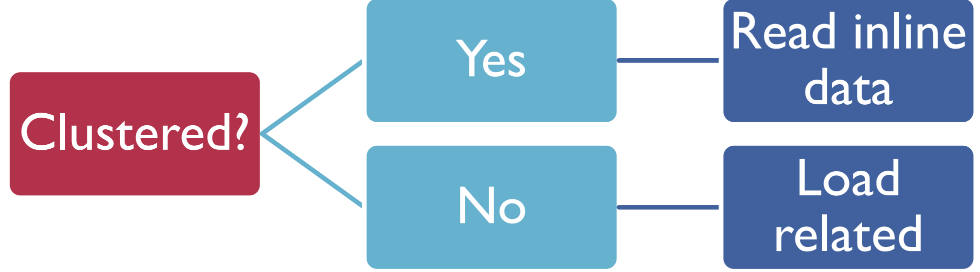
THE STRUCTURE OF A B+TREE PAGE

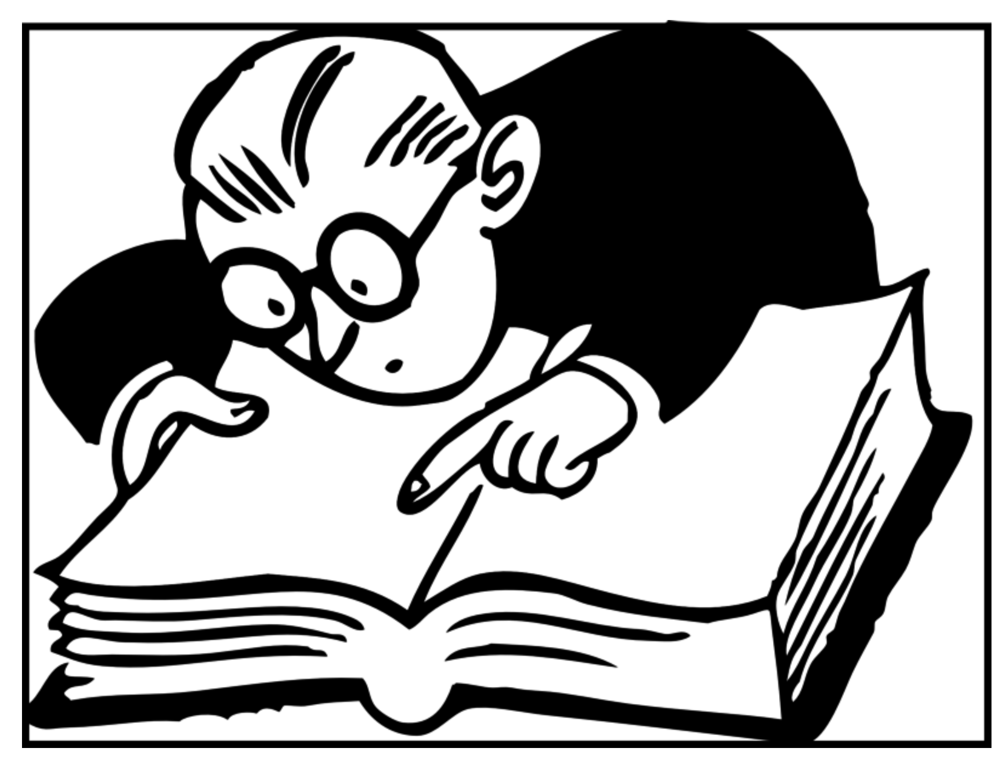
- Slotted pages



B+TREE IS A SORTED DATA STRUCTURE

- What can we do with it?
- Find by key
- Find by prefix
- Find by range





SELECT FROM Users ORDER BY Registered DESC, Name ASC

Registered (DESC)	Name (ASC)	Row
2021-03-21	Cannes	21
2021-03-21	Emily	49
2021-03-20	Able	58
2021-03-20	George	84
2021-03-19	Derek	98

Scan INDEX

For each record: Load via **rowid**

```
def index_cmp(x, y):
    a = cmp(x.Registred, y.Registered) * -1
    if a != 0:
        return a
    return cmp(x.Name, y.Name)
```

WHAT IS rowid?

Logical

- Numeric value that represent the row
- O(logN)

Physical

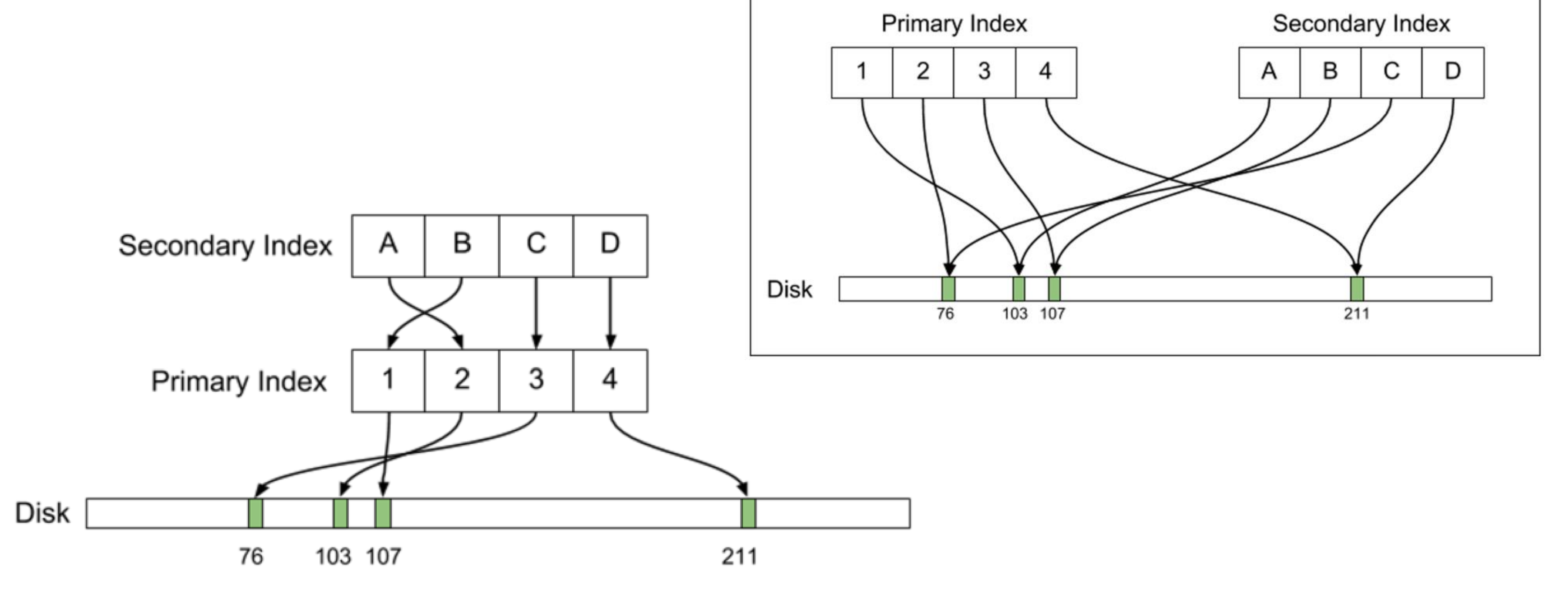
- Physical location on disk
- O(1)

SELECT FROM Users ORDER BY Registered DESC, Name ASC

- O(N * logN)
- O(N * 1)

REAL WORLD SCENARIO: UBER

https://eng.uber.com/postgres-to-mysql-migration/

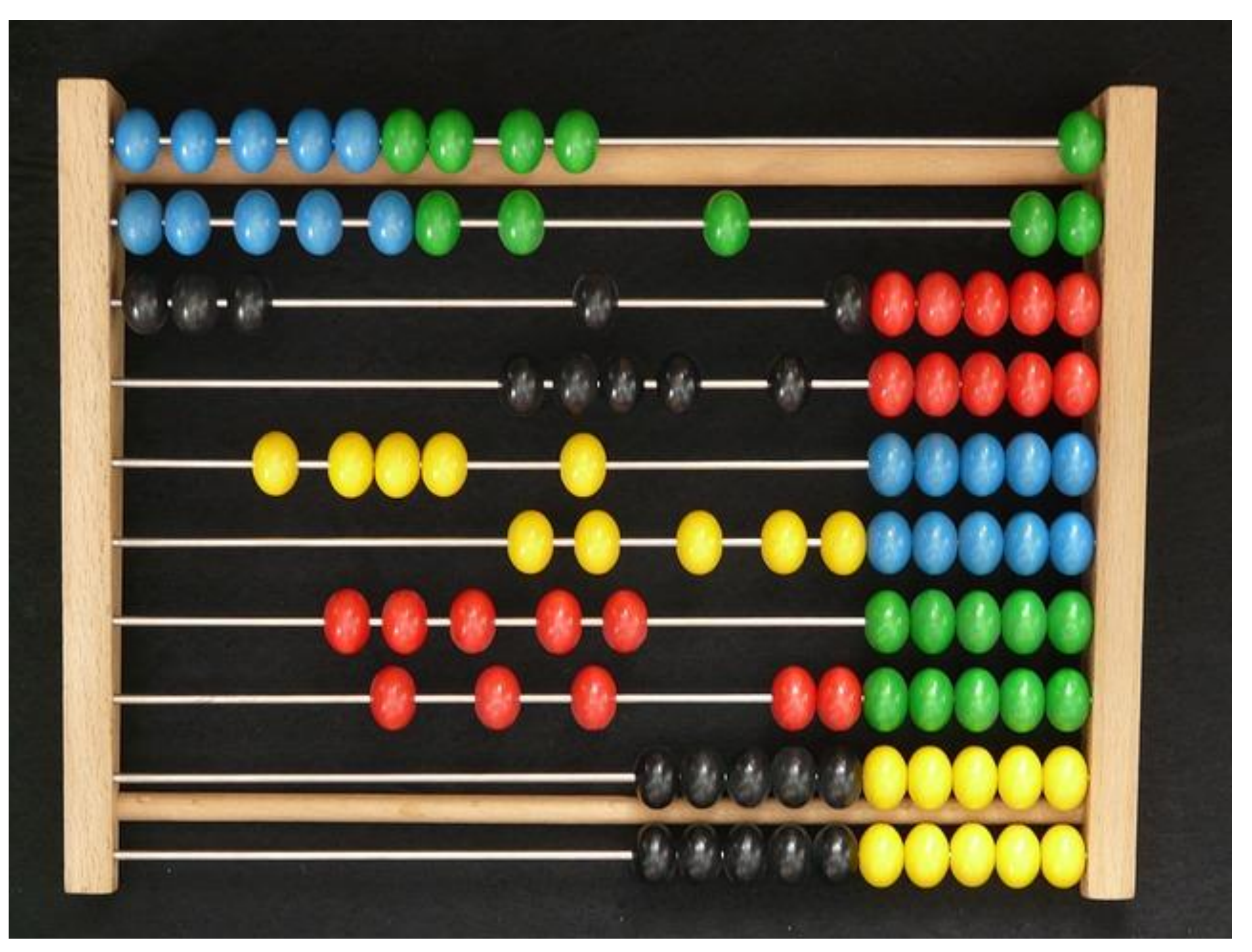


WHAT CAN THE DATABASE OPTIMIZE?

- Find by key
- Find by prefix
- Find by range

- Scan by primary key / secondary key
  (if there is an index)

- All else: COMPUTE!



PART 2: DURABILITY

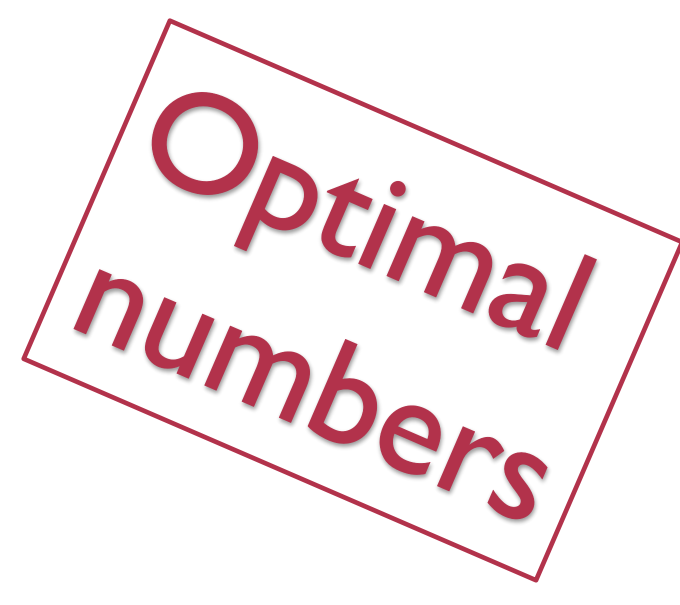
DURABILITY

- Disks sucks
- A lot
- They suck a lot and are horrible
  - Yes, SSD too
  - Yes, NVMe too
- Also, everything you know about I/O is a lie

LATENCY NUMBERS…

- L1 reference - 1 nanosecond
- NVMe Read – 100 microseconds (10,000 nanoseconds)
- SSD Read – 300 microseconds (30,000 nanoseconds)
- HDD Read – 8 milliseconds (8,000,000 nanoseconds)

- On Cloud – Assume latency is x5 worse



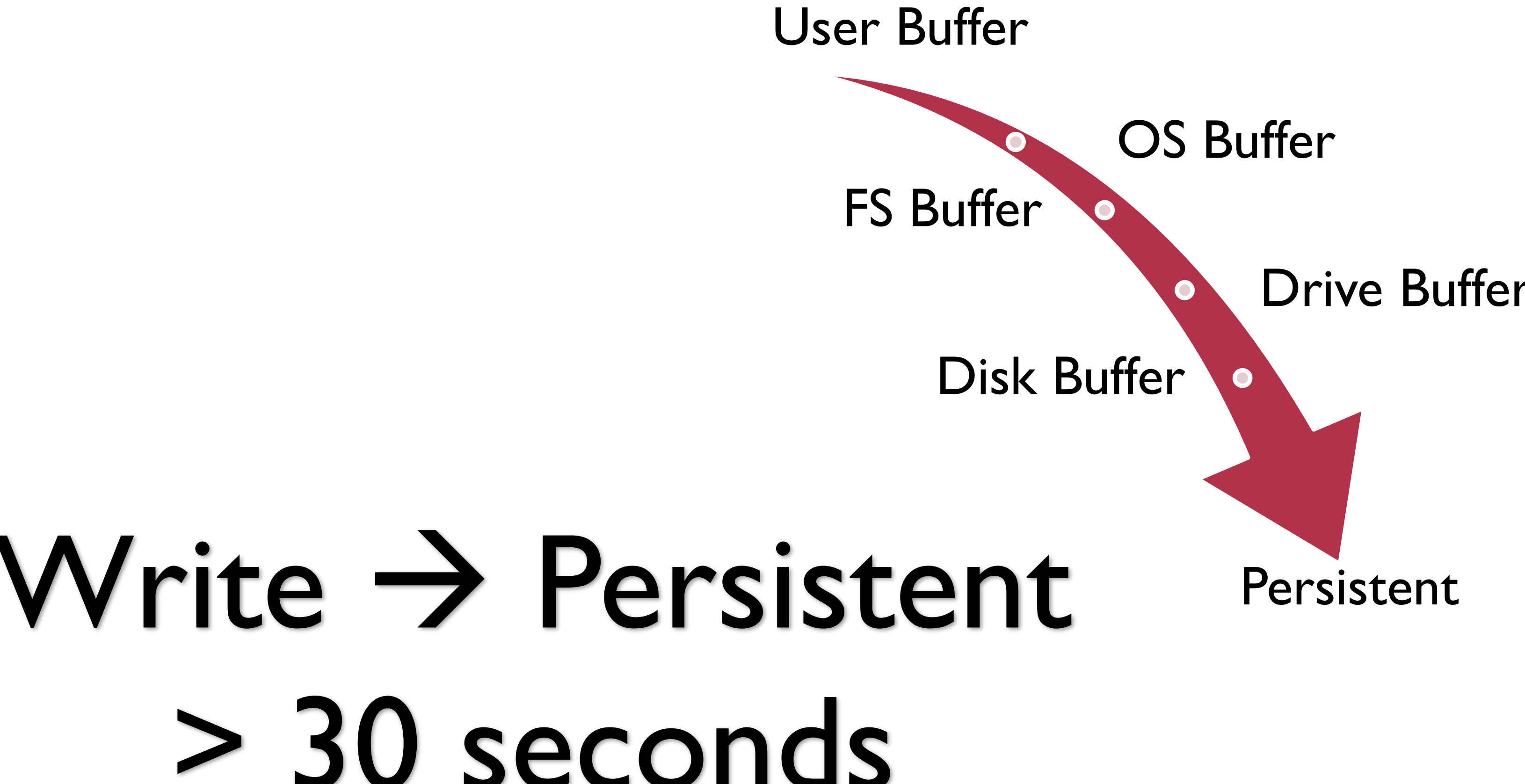
https://www.marvell.com/content/dam/marvell/en/public-collateral/fibre-channel/marvell-fibre-channel-nvme-over-fabrics-white-paper.pdf

WHAT IS THE ISSUE?

Unsafe!

```
bool write_1(const char* file, void* data, size_t len)
{
    int fd = open(file, O_CREATE | S_IRWXU | O_RDWR);
    if(fd < 0)
        return false;
    ssize_t ret = write(fd, data, len);
    close(fd);
    return ret != -1;
}
```

IT'S BUFFERS ALL THE WAY DOWN…



USE FYSNC?

Unsafe!

```
bool write_1(const char* file, void* data, size_t len)
{
    int fd = open(file, O_CREATE | S_IRWXU | O_RDWR);
    if(fd < 0)
        return false;
    ssize_t ret = write(fd, data, len);
    fsync(fd);
    close(fd);
    return ret != -1;
}
```

FSYNC GATE

- Fsync can **fail**
  - Bad things happen then
- https://danluu.com/file-consistency/
- https://danluu.com/deconstruct-files/

- Can Applications Recover from fsync Failures?
  - https://www.usenix.org/conference/atc20/presentation/rebello
  - TLDR – nope.

1. fd = open("/path/to/file“)
2. write(fd, data, len);
3. fsync(fd)
4. close(fd)
5. pfd = open(“/path/to”)
6. fsync(fpd)
7. close(pfd)

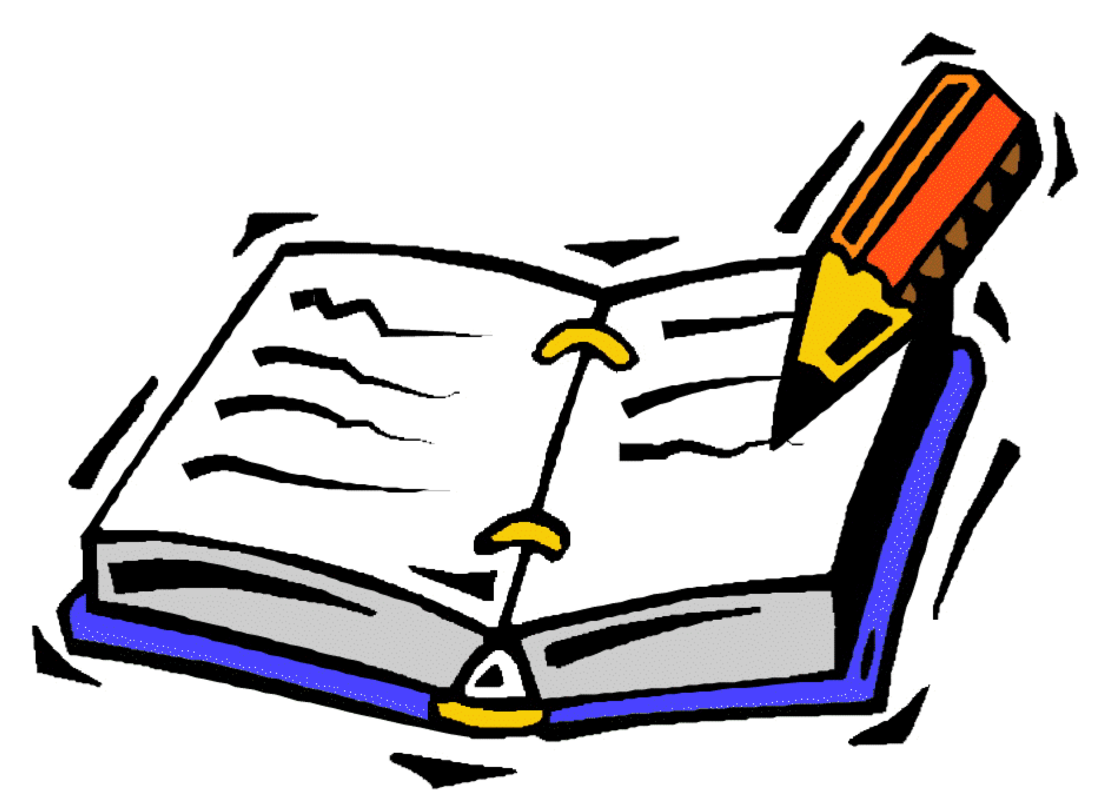
DURABILITY? WHAT ABOUT ROLLBACK?

- Cannot modify in place, what would happen on partial failure?
- Need secondary location.
- Need to durably write there first.

- Models:
  - Write Ahead Log
  - Append only
    - Requires compaction

THE WRITE AHEAD LOG

- Sequential writes
- Durable mode
- Write changes to the data file before making them
- Replay operations from the log on startup
  - Redo log
  - Undo log



THE WRITE AHEAD LOG & THE PROTOCOL

Chat

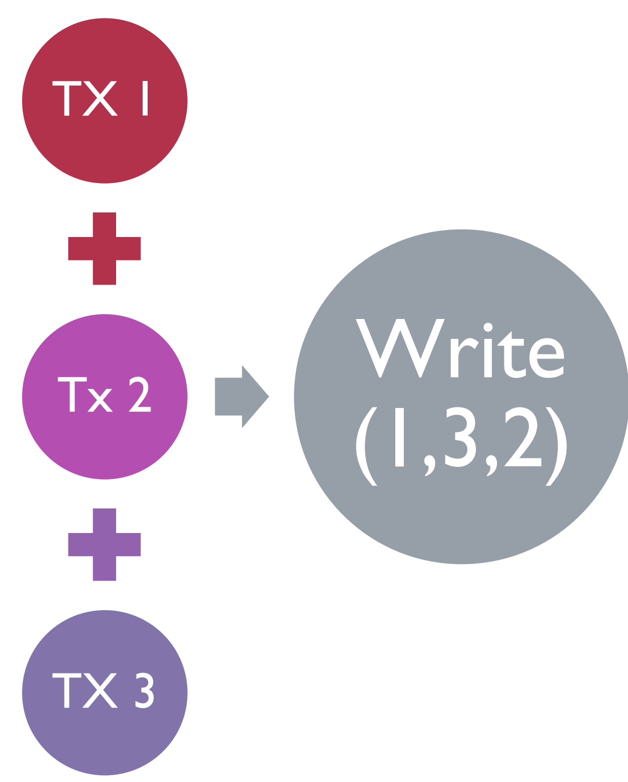
Message

WRITING TO THE LOG…

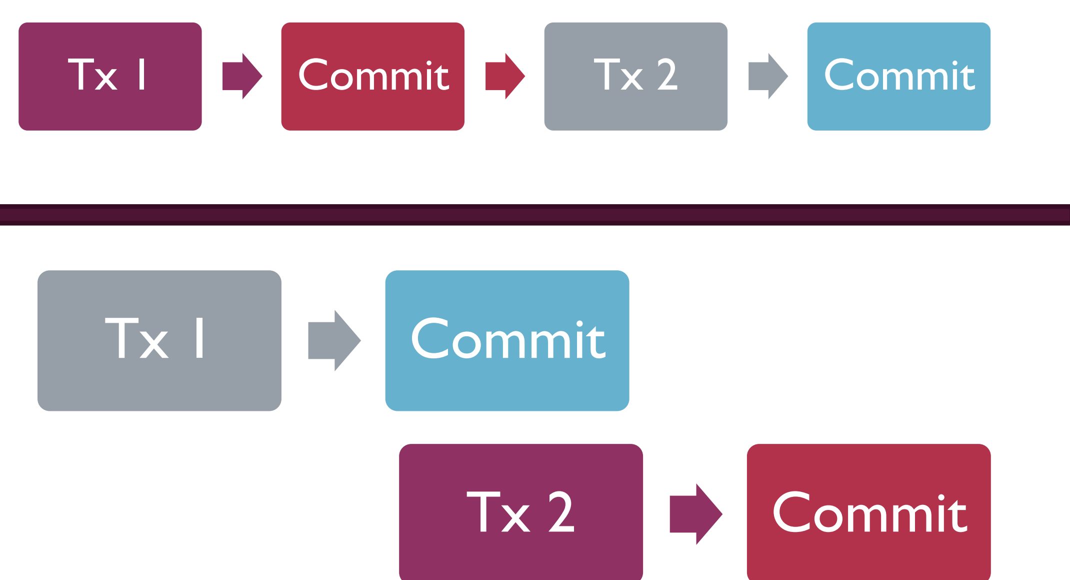
- Must be durable
  - write() + fsync
  - open(O_DSYNC) + write
- Durable writes:
  - O_DIRECT + O_DSYNC
  - 4KB aligned
- Commit happens after done with log write

- Protect from partial writes
- Auto extend file or allocate in advance?
- Hopefully never read…
- What is the cost of replay the log?

WRITING TO THE LOG IS **SLOOOOOW**



- Can optimize in several ways
- Merge concurrent transactions to a single write
- Speculatively execute transactions



PART 3: TRANSACTIONS & CONCURRENCY

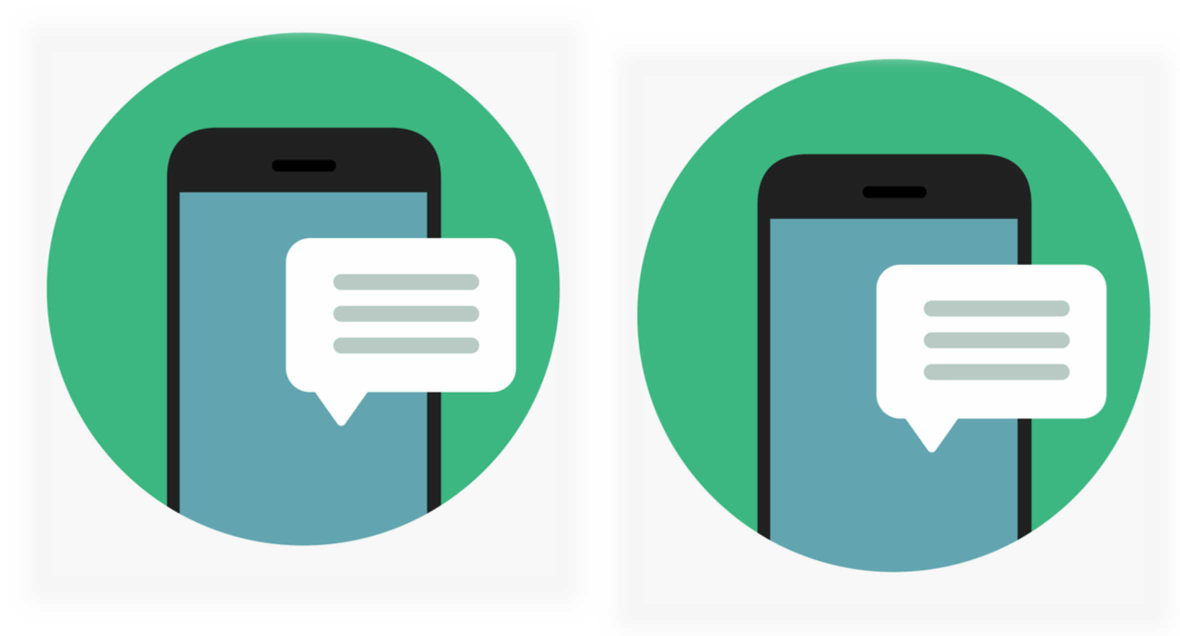
TRANSACTIONS

- Concurrent transactions?
  - Reads + Write?
  - Reads + Writes?
- What is the protocol?
- Single threaded write preferred

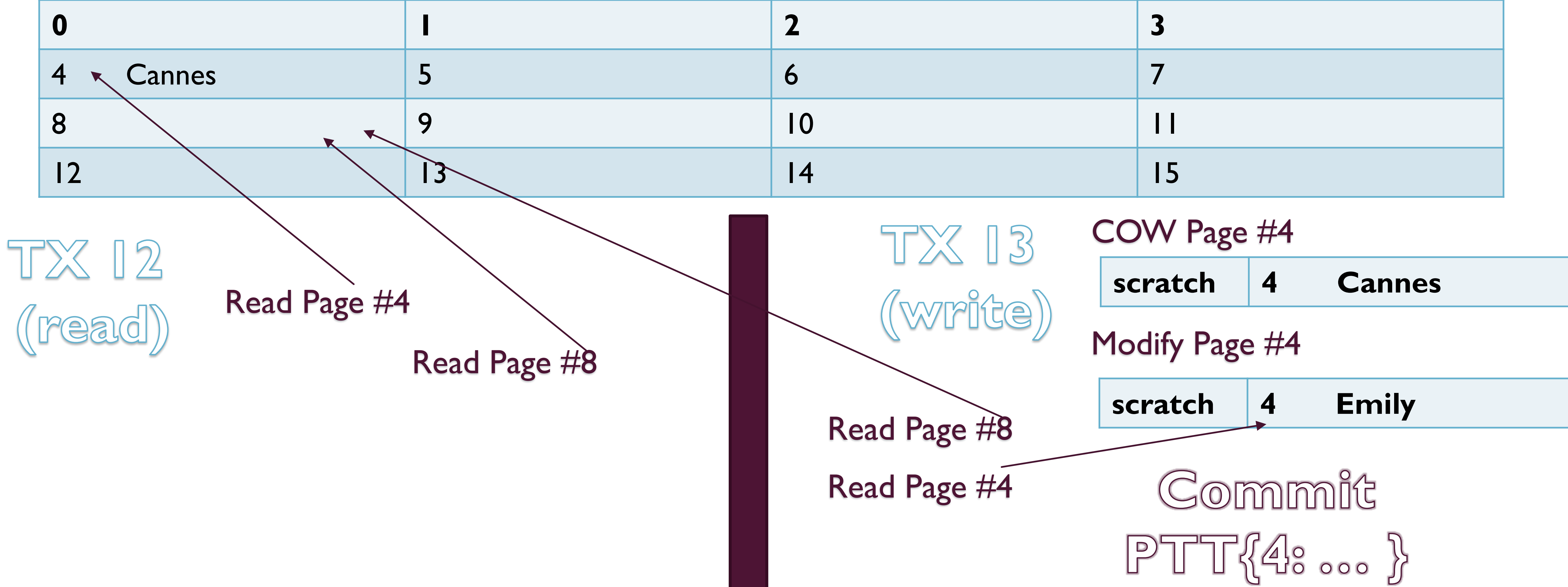
- Concurrent writes transactions == locks
  - How expensive are these?

OLTP THROUGH THE LOOKING GLASS PAPER

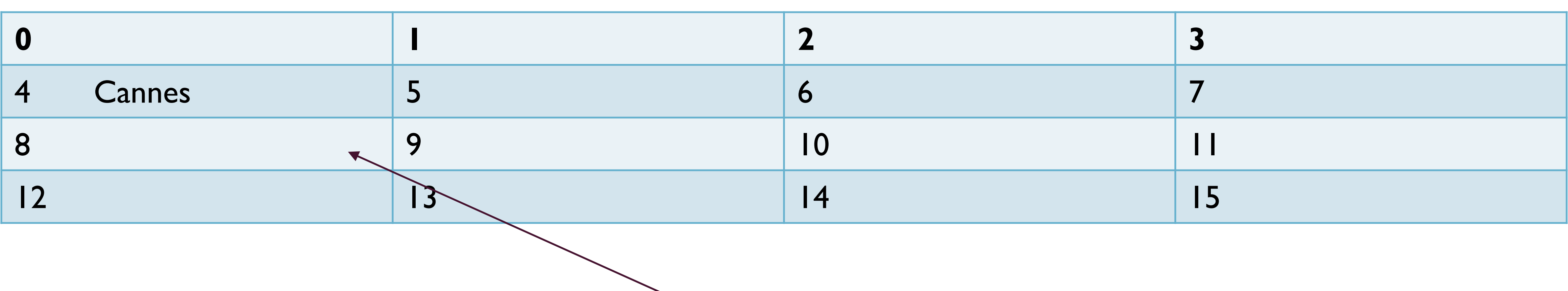
- Looking and Latching > 30% runtime
- Single threaded is preferred
- Can we model write operations without concurrency?



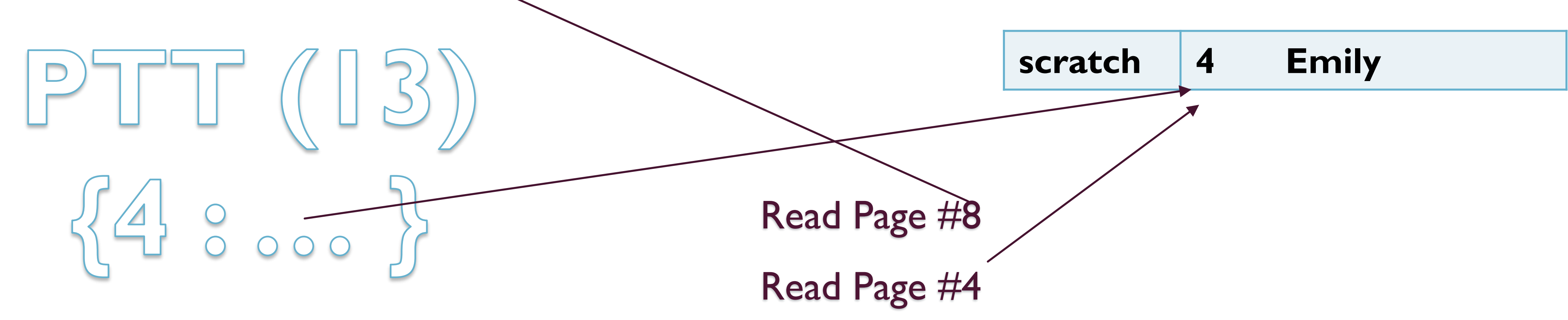
MVCC – Multi Version Concurrency Control



Page Transaction Table (PTT)

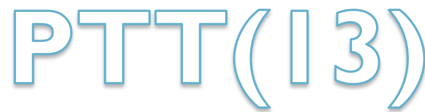


0	1	2	3
4 Cannes	5	6	7
8	9	10	11
12	13	14	15



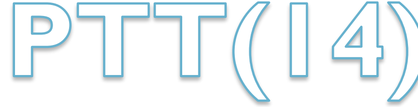
Multiple concurrent versions

0	1	2 Derek	3
4 Cannes	5	6 Brett	7



scratch	4 Emily

Scratch	4 Eddie



Scratch	2 Geogre
	6 Hanna

Scratch	2 Able

TX (R: 14)

TX (R: 14)

WRITING TO THE DATA FILE

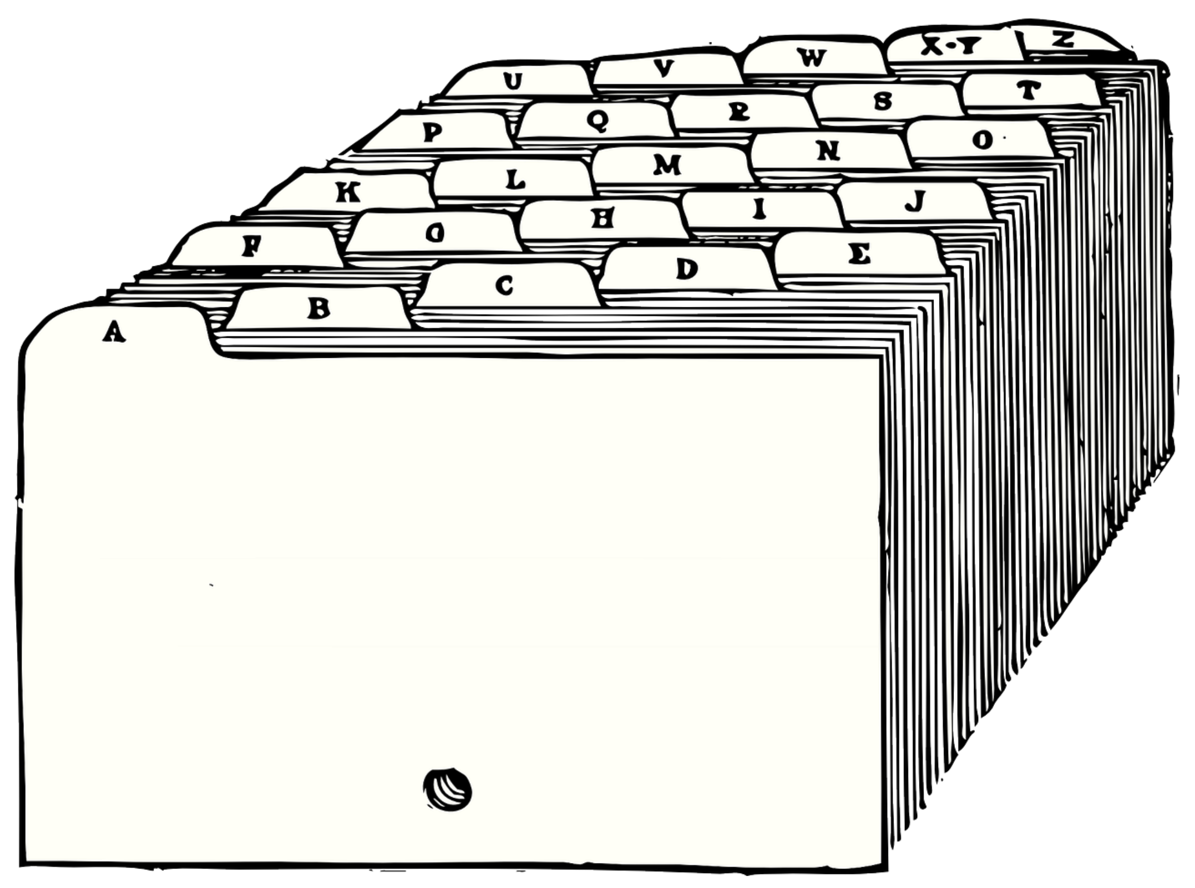
- When no one is looking…
- Copy latest version from scratch to the data file
- Update PTT to remove the reference
- New transactions will go to the data file

ACID

- Atomicity?
  - A transaction has a snapshot view of the world, enforced via the PTT
  - PTT updates happens atomically
- Isolation?
  - Each transaction is independent
  - Single write tx
  - Readers don’t block writer and vice versa
- Durability
  - The write ahead log

WRITING TO THE DATA FILE ISN'T THE END

- Data file writes are buffered
- Can do *fsync()* occasionally
- After successful *fsync*():
  - Can trim the write ahead log
  - Can free scratch buffers



LET'S COMPARE MVCC IMPLEMENTATIONS

Postgres

rowid	from_tx	to_tx	name	email (pk)
4	17	19	Cannes	cannes@foo.bz
5	20	25	Blake	cannes@foo.bz
6	26	null	Emily	cannes@foo.bz
7	16	29	George	g@baz.fi

RavenDB (Voron)

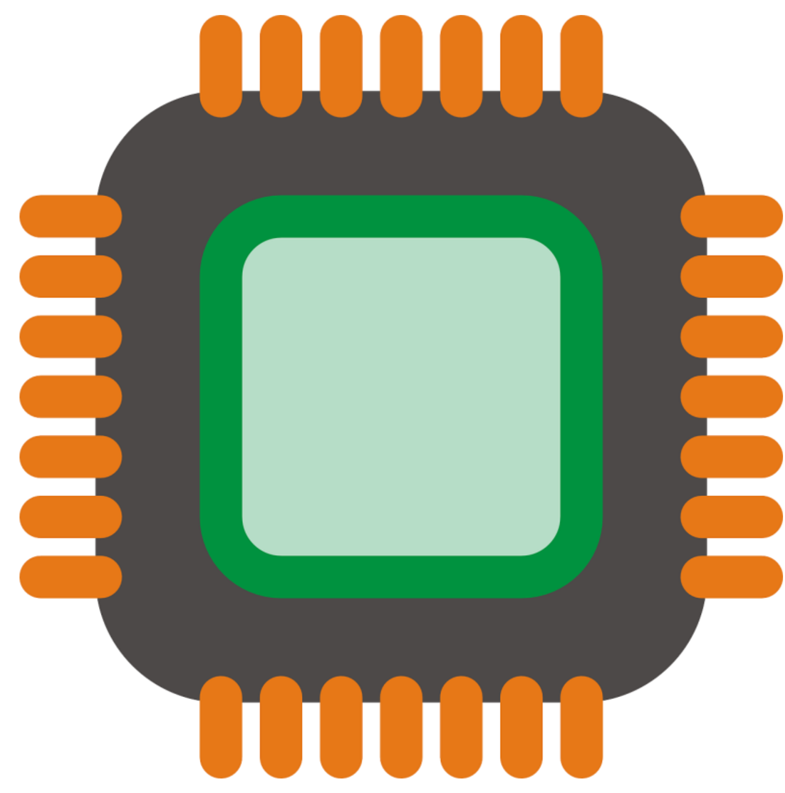
PTT (17) → 8	PTT (20) → 8	PTT (26) → 8
Cannes	Blake	Emily
Derek	Derek	Derek
Goerge	Goerge	Goerge

select name
where email = 'cannes@foo.bz`

Tx 18 – Cannes
Tx 25 – Blake
Tx 30 – Emily

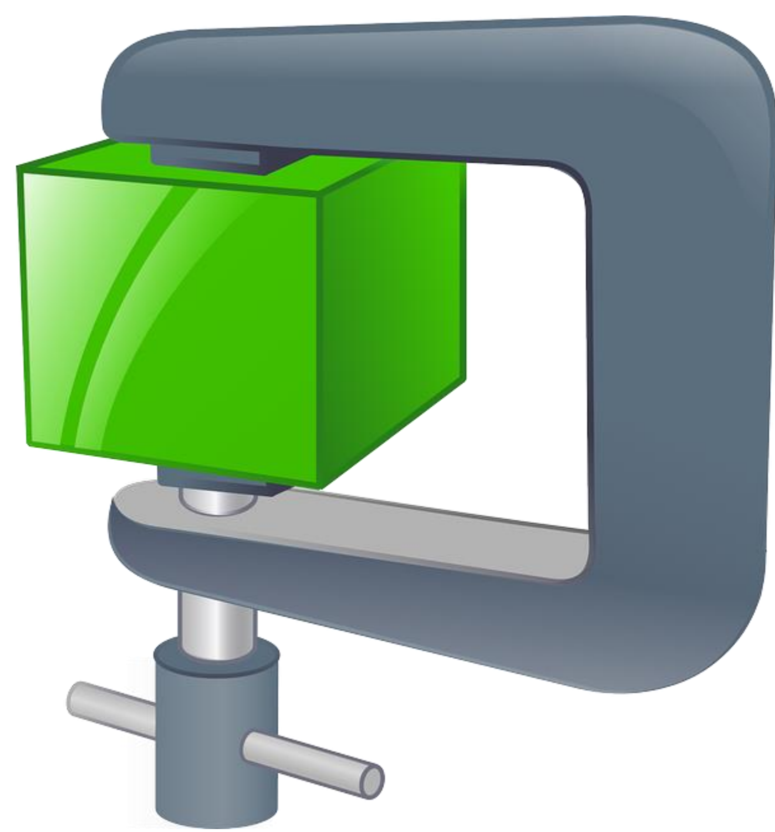
PART 4: TRICKS & OPTIMIZATIONS

WHAT IS MORE COSTLY?

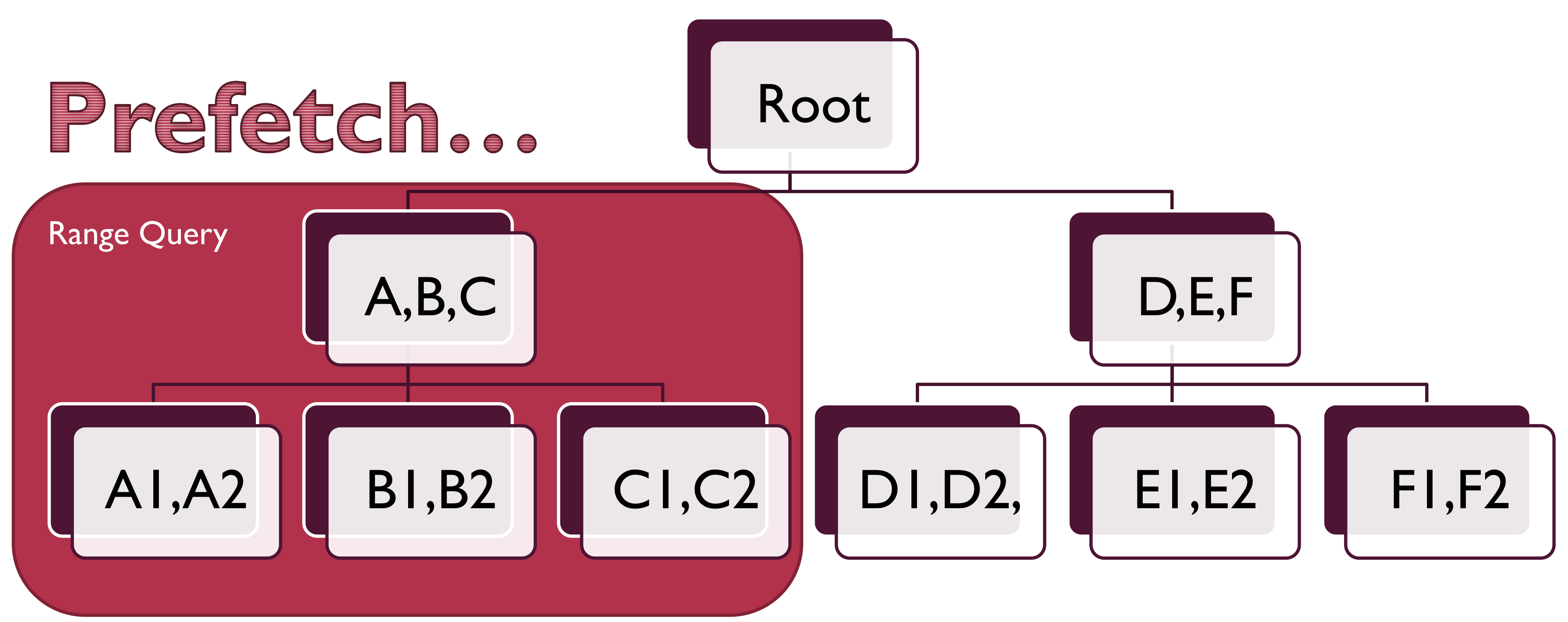


TRADEOFFS

- Writing to the log file is **expensive**
- Batch writes with transaction merging
- Write full pages to the log, why?
  - Copy on Write
  - We have previous & current versions
  - Apply DIFF – reduce log size
- Why stop at diff? Compress



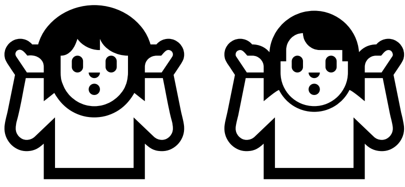
PREDICT THE FUTURE



BUFFER MANAGEMENT?

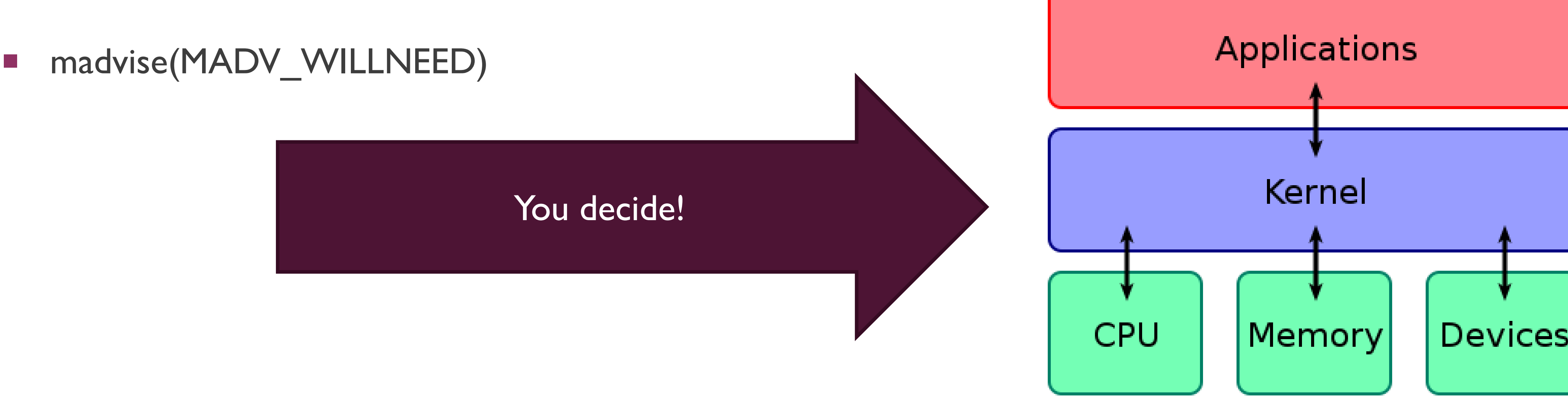
- Complex
- Balance current and future needs
- Use enough memory to optimize
- Don't hurt other aspect of the system
- Need a global view

- Postgres uses 2Q
- Others: LRU, ARC, CAR, LIRS, CLOCK-Pro



YOU CAN CHEAT…

- mmap()
- madvise(MADV_WILLNEED)



QUESTIONS?