

CSE 350

Advanced Data Structures

Topic 16: Enforcing Durability

What is Durability?

↳ power failure
↳ program crash

↳ Tr-Rex attack

↳ Axe Murderer

↳ Explosions

Not a prob
for us

↳ RAID

↳ Log shipping

Challenges

1: Atomicity

Buffer



Changes

Disk



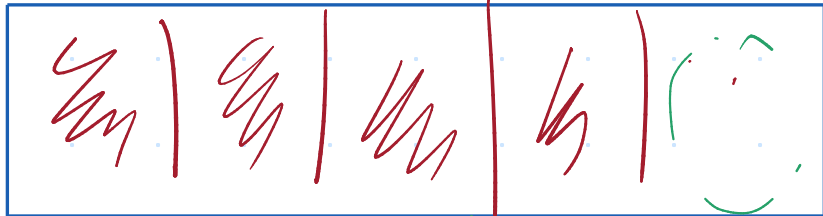
ERROR!

Not Atomic

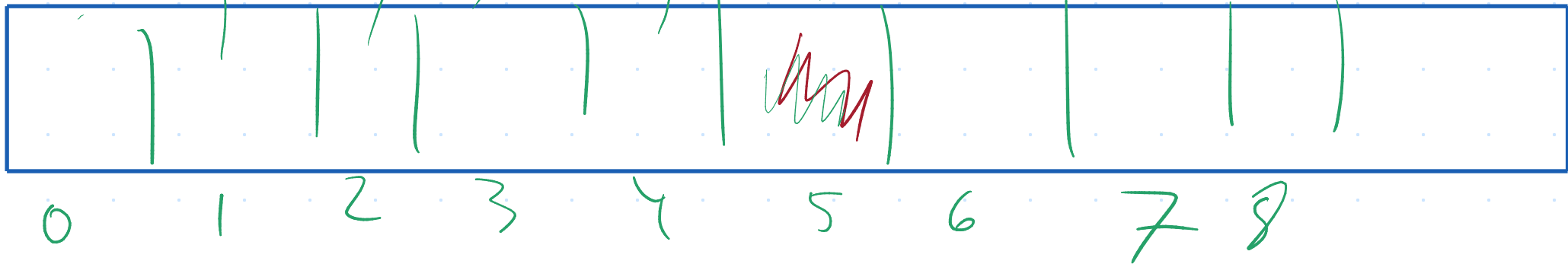
2: Abort



Buffer



Disk



Ground Rules



→ atomic

↳ write(address, data)

↳ read(address) → data

↳ flush

} guarantees all writes are "safe"

Approach

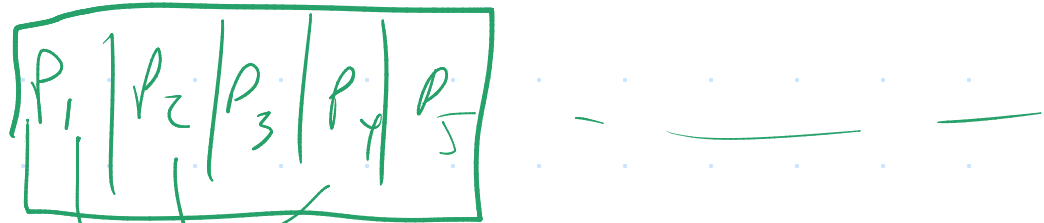
Deferred Write

↳ write data some where

↳ write a commit record

repeat
as
needed

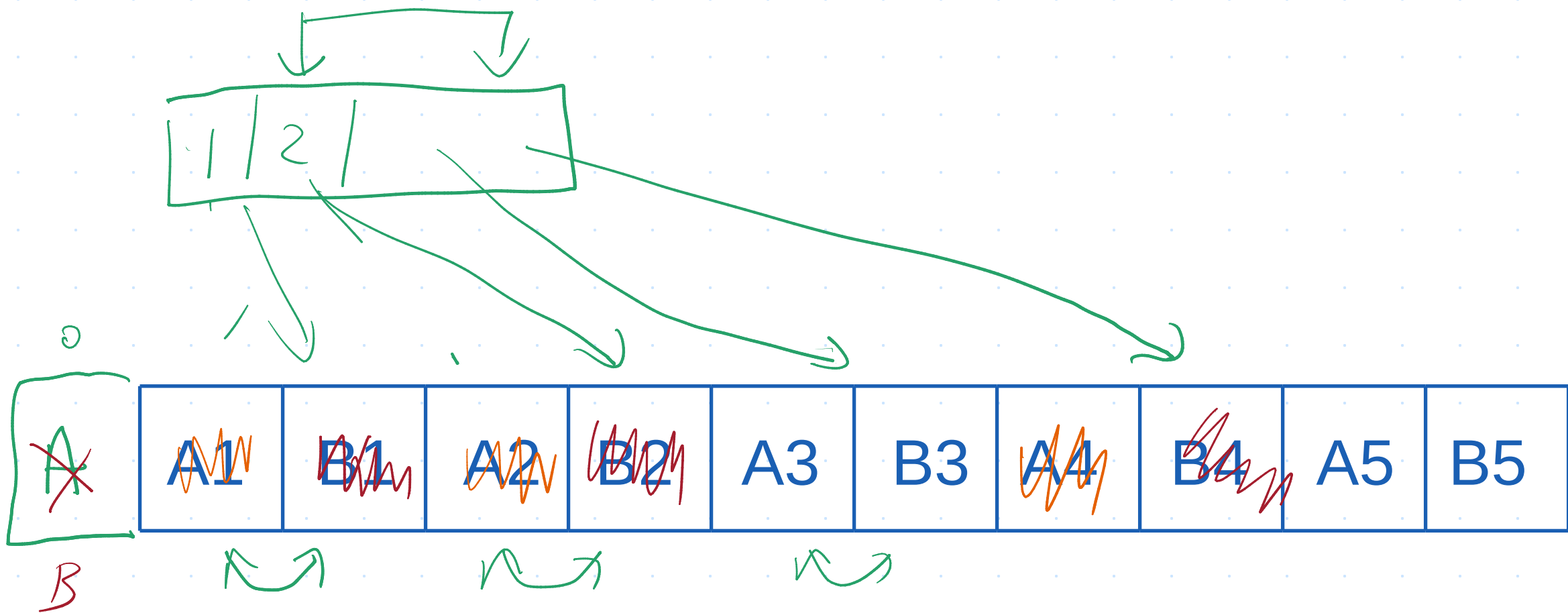
Logical



Physical



Twin Block



Abort

Commit

Commit

- Flush

- Swap $A \leftrightarrow B$

- Flush

→ report to caller
- Copy changes that data
back to twin safe

Abort

- Don't do anything to $A \leftrightarrow B$

- Restore changes from twins

Pros

- No data structures needed for lookup table
- Spatial locality is preserved (clustered)
- Easy abort

Cons

- 2 copies of all data
- No direct support for concurrency

Shadow Page

get_Pages

0	1	2	3
---	---	---	---

Lookup Table

0 → 5
 1 → 4
 2 → 3
 3 → 6

Read only

CoW

	B1	B2	B3	B4	B5	B6	B7 ^{B4'}	B8	B9	B10
--	----	----	----	----	----	----	------------------------------	----	----	-----


Read write

0	1	2	3
---	---	---	---

Lookup Table

0 → 5
 1 → ~~4~~
 2 → 3
 3 → 6

Commit

- write lookup table to disk
- flush
- update reference to current lookup table
- flush  New data is "safe"
- cleanup (garbage collect)

Abort

- do nothing (delete lookup table)
- cleanup

Pros

- Only duplicating differences ($\ll 2x$)
- Easy abort
- More than one lookuptable \rightarrow Better concurrency

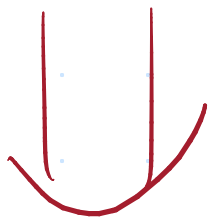
Cons

- Logical spatial locality \neq physical spatial locality
 - Need data structure for lookup table
- \rightarrow Fragmentation

Sidebar: Copy on Write/Merge on Write

Cow

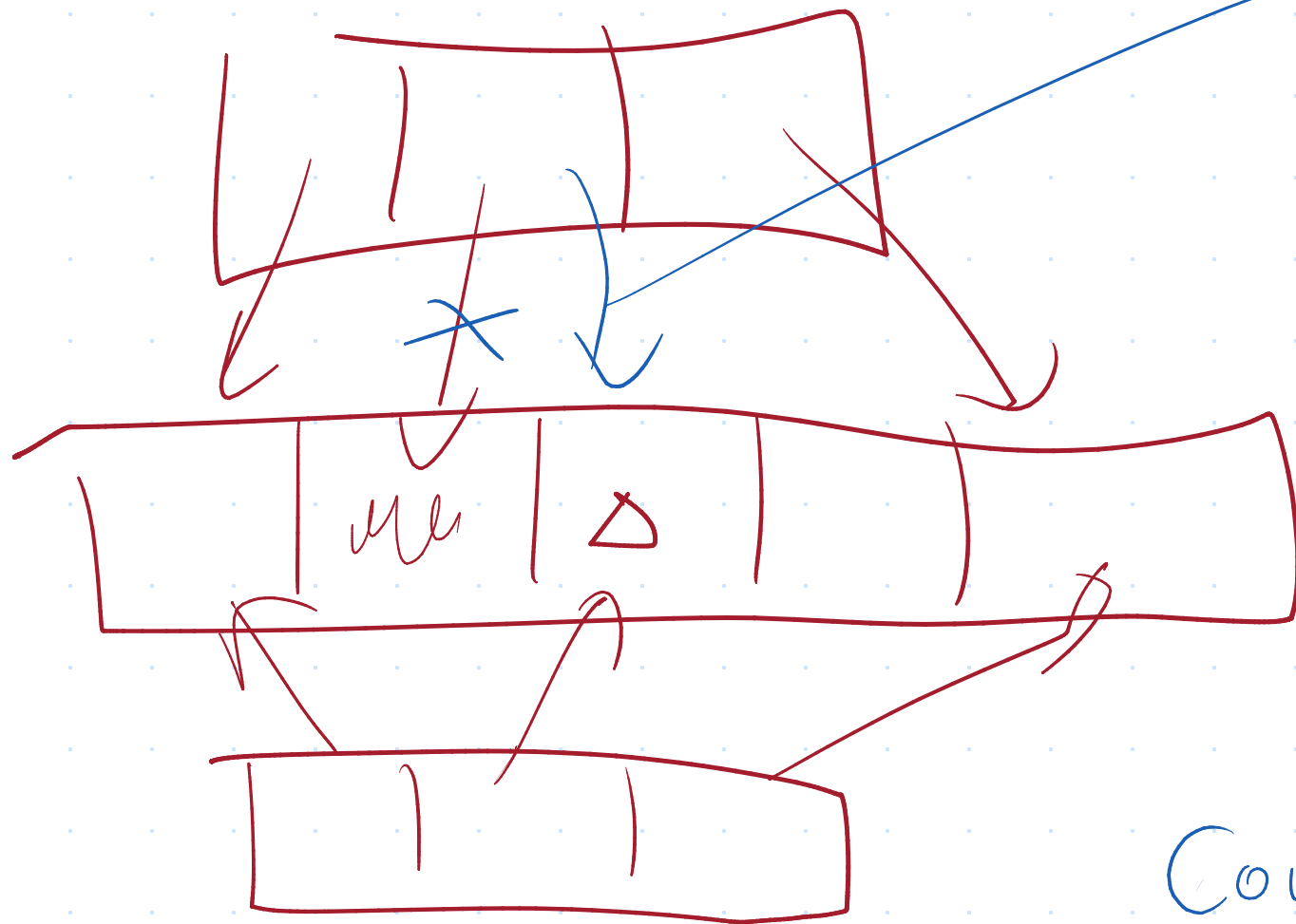
Fragments of Memory (disk, hash buckets)
↳ modifying some but not all



Logical/Physical indirection

↳ share data until it changes

↳ copy only when it changes



Merge or Write
 ↳ Detect write of
 a duplicate value
 (e.g. id fragment)
 w/ hash

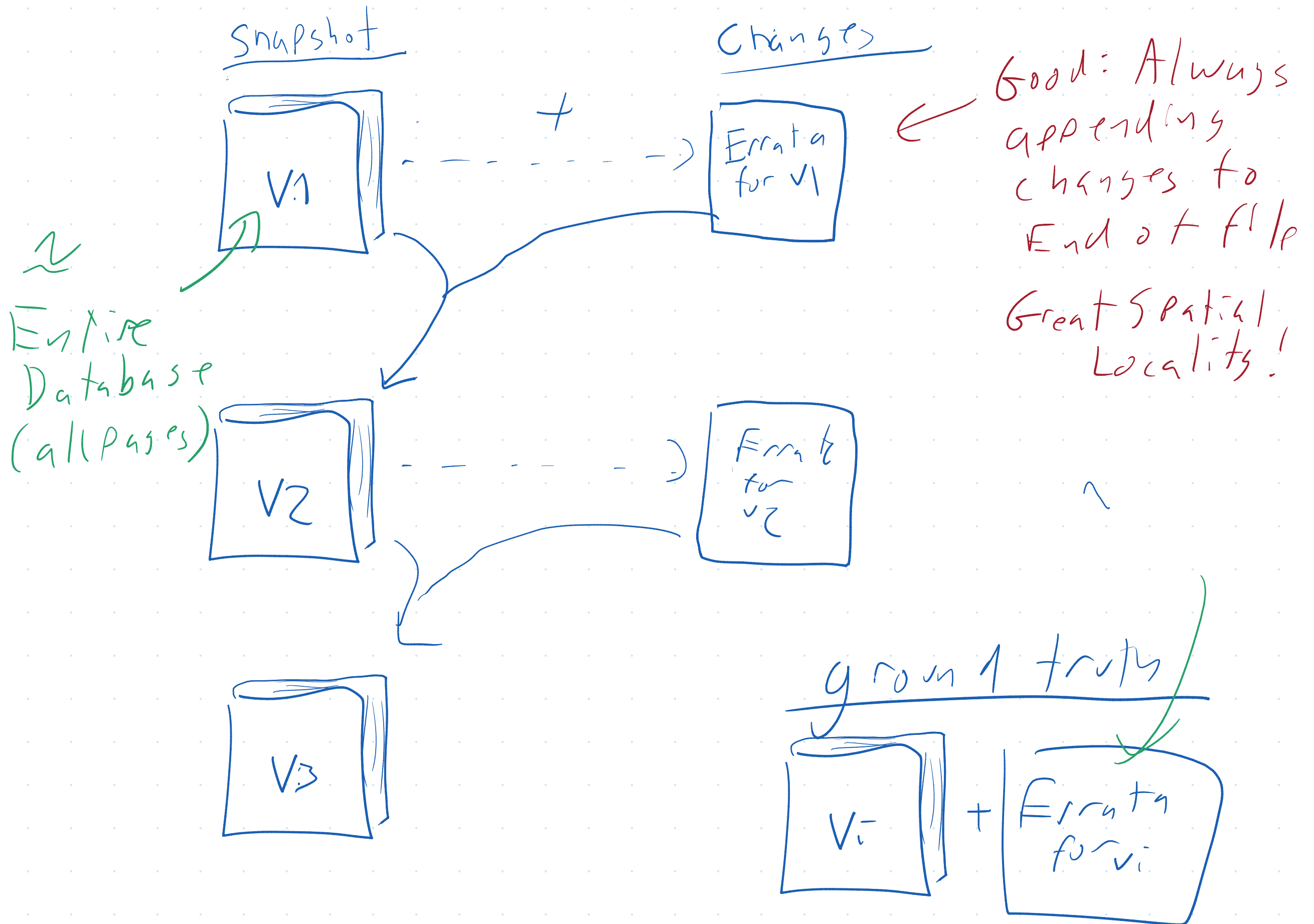
Cow / Mow

Virtuals | Memory

Git (cow + mow)

Filesystems

Differential Files



Changes

page 23 is now

- byte, 4-byte 23 are now "foo"

- field A of record 7 on page 23 is "foo"

- add 2 to field C of ---

Commit

flush (after writing
changes to log)

write "commit" to log

Abort

write "Abort"

↙ Restore ↘

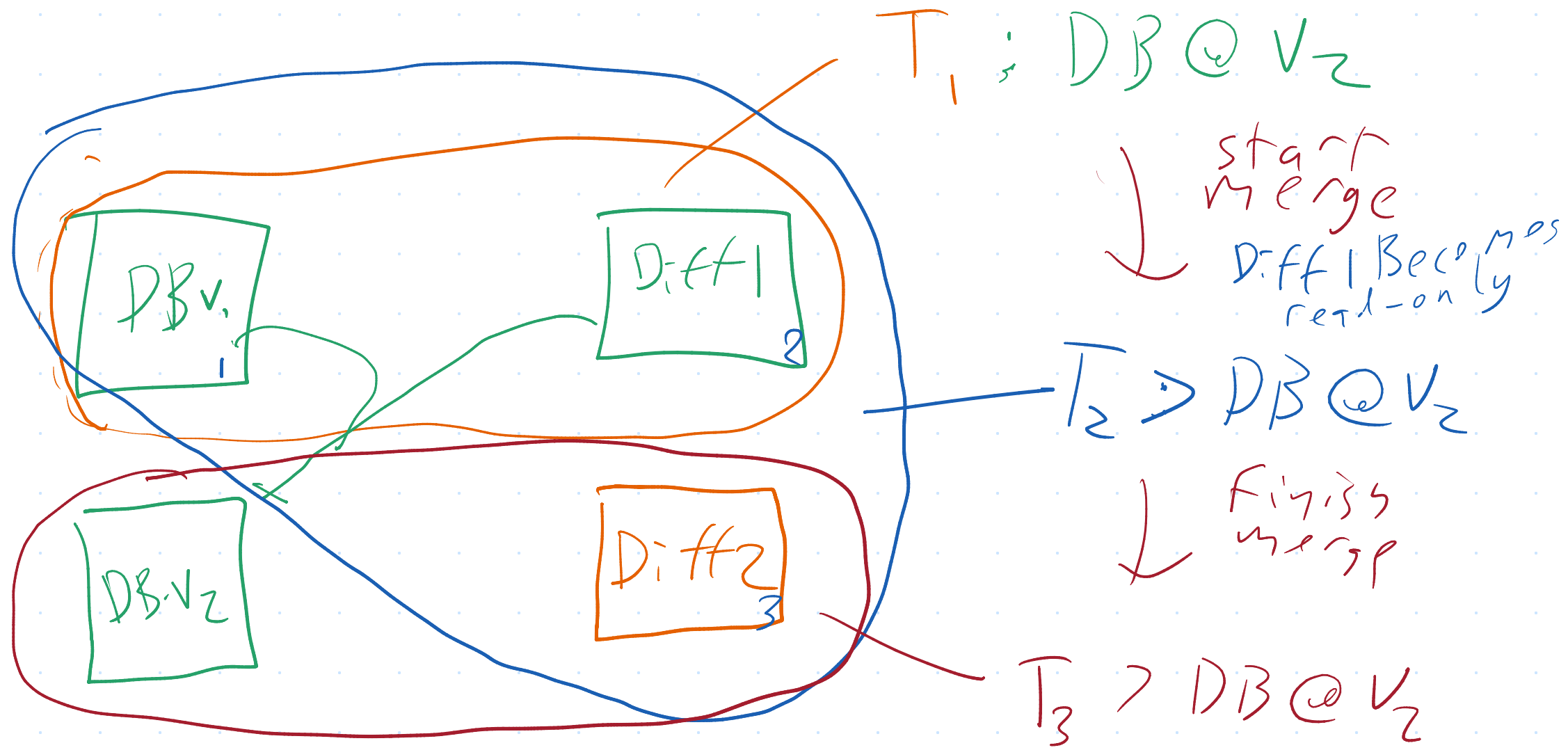
only apply changes
as of the last
commit

Pro

- Really good spatial locality for writes
- Diff Files are incremental backups
- Easy to recover snapshots of DB
- "Easy"

Con

- More space (need to store all changes)
- Reads go to PB AND ^{Differential File} ~~Errata~~
ENTIRE



(cont) While merging, we need to look at DB & 2 diff files

Logging / WAL

Diff Files

DB (Read)

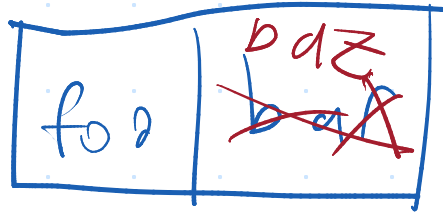
Diff (Rw)

Logging

DB Read/
write

log (w)

Butter Mgr

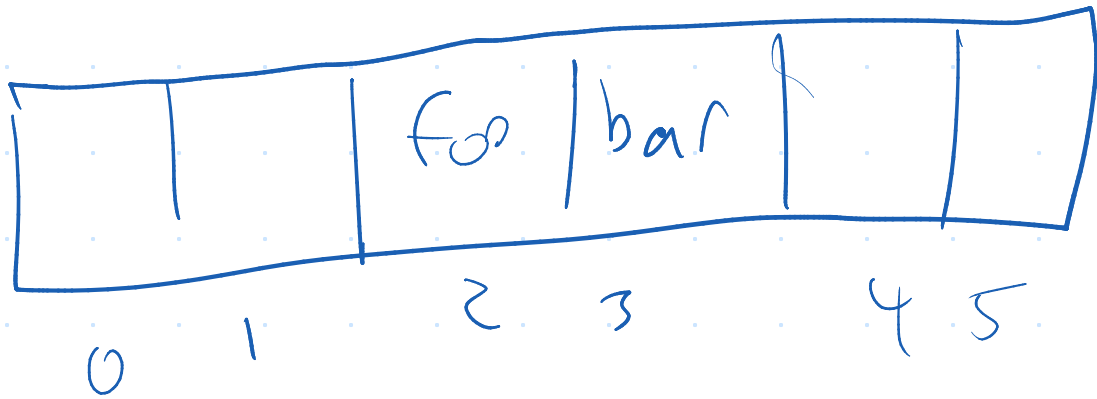


P₂ P₃

Mem

Disk

DB



Log

P₃ ← "baz"

add Ⓢ
to changes

Commit

Idempotent

Not Idempotent

On crash

replay log

commit

guarantee log written
up to & including
commit

Challenge

- Log records must be Idempotent (replay has no effect after first)
- Need to periodically checkpoint

Diff File

Logging

Idea

Collect Changes
in a file

DB_{current}

Read Only

R/W

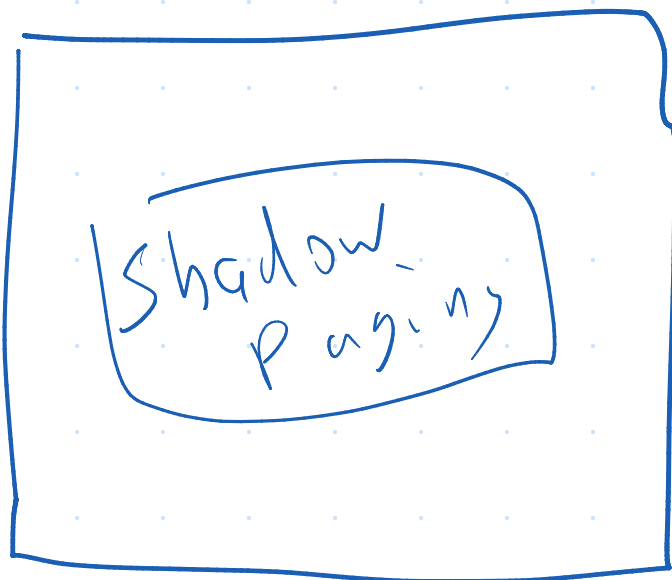
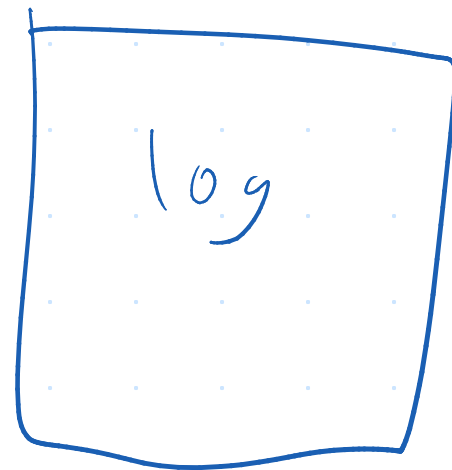
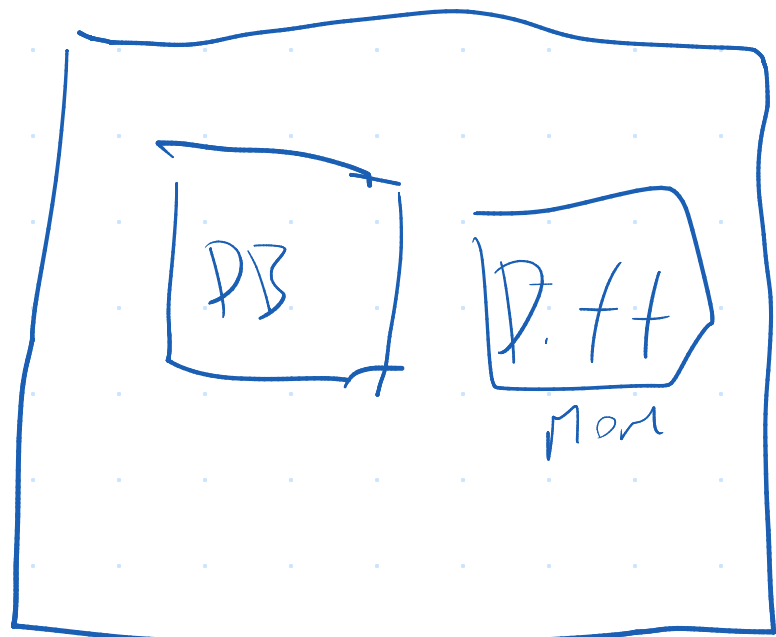
Truth

DB + Diff

DB

L

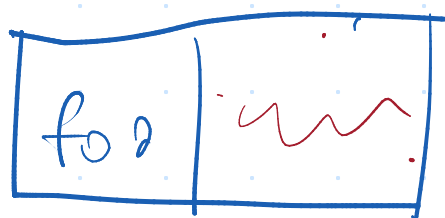
Hybrid Models



Better
spatial
locality
for
writes
✓ Benefits
of the
Base Model

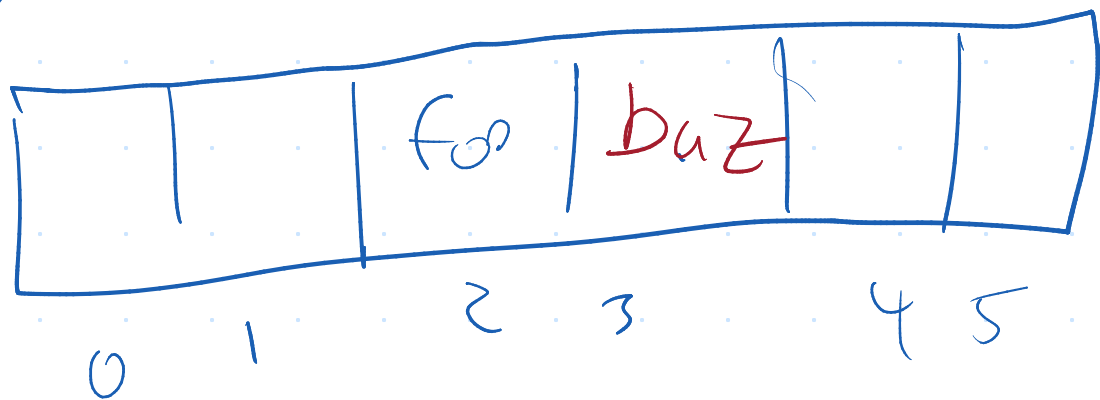
Undo Logging

BufferMgr



P2

DB



L₀

~~P3 ← "bar"~~

~~Abort~~

problem: what was on P3 before?

P3 replace "bar" with "baz"
abort

Abort

write abort to log

→ go to prev entry in log, apply undo effect
→ repeat until last commit

