Lec 3 DB Storage I

DB Management

DBMS — DB management sys

App ⟷ (SQL) Query Planning ← query optim
Operator Execution ← run query.
Access Methods
Buffer Pool Manager ← cache
Disk manager

this stack

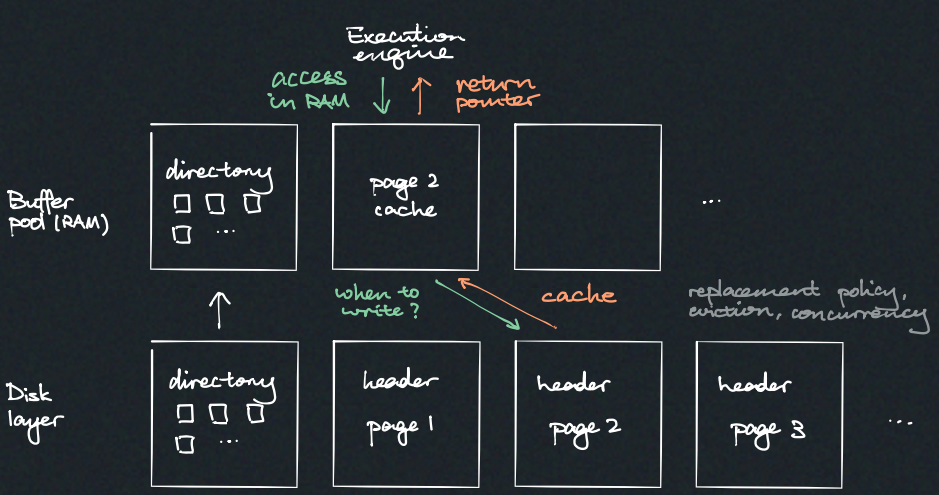
Disk-based Archi.

- storage hierarchy
- on non-volatile disk
  - SSD, HDD, cloud, etc.
  - usually only block-addressable
    - eg. 4KB blocks
  - sequential access faster than random
    - need good locality for performance
- multithreading to use multiple cores
  - write conflict handling

reg, CPU cache, DRAM — byte addressable
SSD ← CXL, HDD ← RNS, Cloud — block addressable
* some newer tech

DB Files

eg. one file per table / all tables in one file



[bring disk file to virtual mem (at least appear so)

? OS provides mmap, why not use it?
- OS mmap doesn't handle file eviction the way we want
  we want to be in control using semantic context in the DB
  ↳ usually DB process just malloc its entire cache
  (buffer pool size from config)
  ↳ sometimes tell OS to not cache automatically

- more mmap problems:
  - transaction safety — OS page flush may be bad
  - IO stalls — stall on page fault
  - Error handling — mmap gives SIGBUS
  - Performance issue — TLB shootdown
    └ TLB change on one core and cores have to sync
    └ translation lookaside buffer

In general, OS isn't helpful

File Storage

Early systems — run in privilege mode to do more manage

Mordern — put everything in one db file

Superimposed Structure:

Array of bytes → Array of pages
Each page gets unique ID
Set some page size

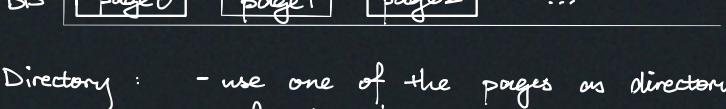
Page layers: Hardware page, OS page, DB page

[in the "dump it there" sense

Heap File Structure

[in the logical sense, not that they lack index

Unorder collection of pages
- create / delete / write / get page
- supports iterating



Directory:
- use one of the pages as directory that refer to other pages
- keep track of metadata
- keep track of free pages

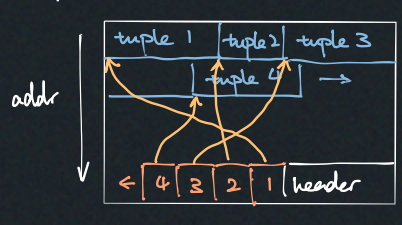
Page header:
- checksum
- page size
- DB version
- transaction data
- metadata / aggregation
  - e.g. min/max date for quicker query

Self-contained page — all info in the page
some systems do this

Tuple oriented page

If fixed len — just array of records

In practice for variable len: slotted pages

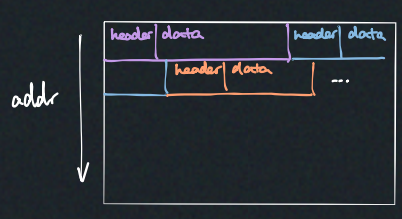


→ in case of deletion, move ▭ cells only, leave ▢ open until knowing outside world won't refer to deleted ▢
→ can do batch compacting

Postgre: use ctid column to get (page, slot) tuples
... delete some records
... run compaction
⇒ slots & tuples moved around

Tuple Layout

Keep tuple headers like malloc



some sys: if frequently joined, might even stored joined entries directly

Research Frontier

Can we do more with less

→ arrange bits differently so multiple records processed by single cycle in same register