

L19

Disk, Storage, and Indexing

Example

Branch, Customer, banker Name, Office
BCNO

Name \rightarrow Branch, Office N \rightarrow BO
Customer, Branch \rightarrow Name CB \rightarrow N

CB is the key (determines everything)

3NF

F^{\min} = minimal cover of F

Run BCNF using F^{\min}

for $X \rightarrow Y$ in F^{\min} not in projection onto $R_1 \dots R_N$

create relation XY

BCNO $BC \rightarrow N, N \rightarrow BO$

NBO, CN using $N \rightarrow BO$... oops create BCN

NBO, CN, BCN ... remove redundant CN relation

NBO, BCN ... BCN: BC is key; $N \rightarrow B$ violates BCNF

Keys and Functional Dependencies

A key must determine all column values

Must be the left hand side (aka input or source) of FDs

$BC \rightarrow N, N \rightarrow BO$

NBO key is N: determines all other fields

Banker name determines branch, office

NBO is in BCNF: All FDs are keys

$$BC \rightarrow N, N \rightarrow BO$$

BCN key?

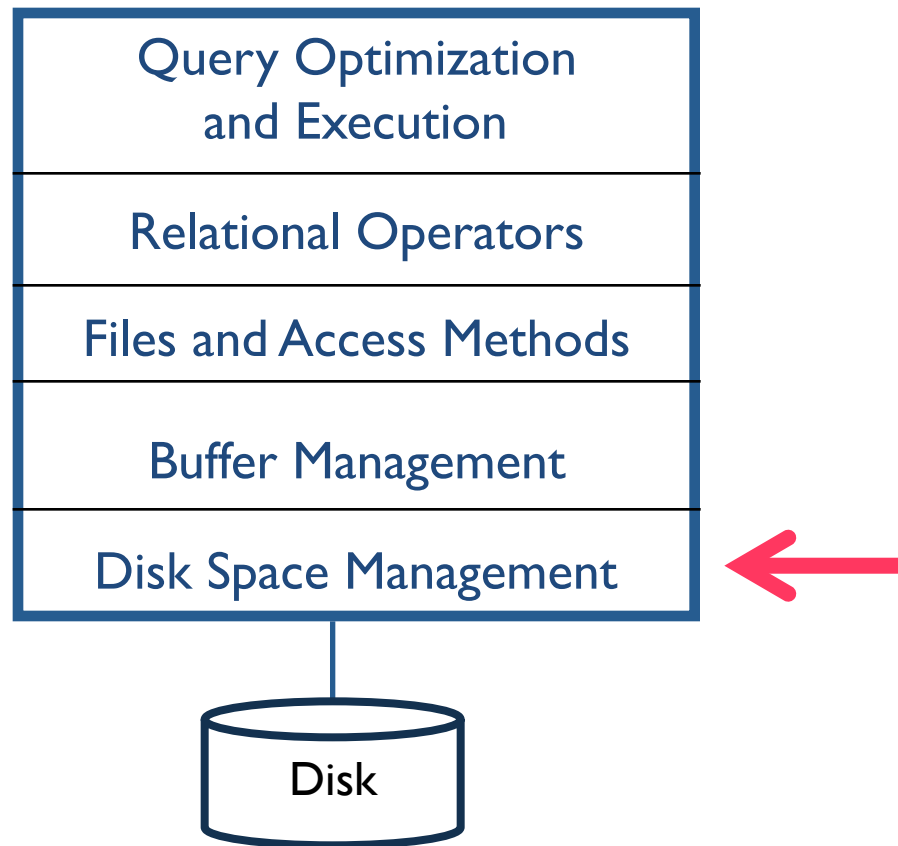
Maybe N? N determines B, but not C!

Problem: A banker has multiple customers!

Must be BC

$N \rightarrow B$ violates BCNF; Okay for 3NF

Work from the bottom up



\$ Matters

Why not store all in RAM?

Costs too much

High-end Databases today ~Petabyte (1000TB) range.
~60% cost of a production system is in the disks.

Main memory not persistent

Obviously important if DB stops/crashes

Some systems are *main-memory* DBMSes, topic for advanced DB course

\$ Matters

Newegg enterprise \$1000

RAM: 64-96 GB

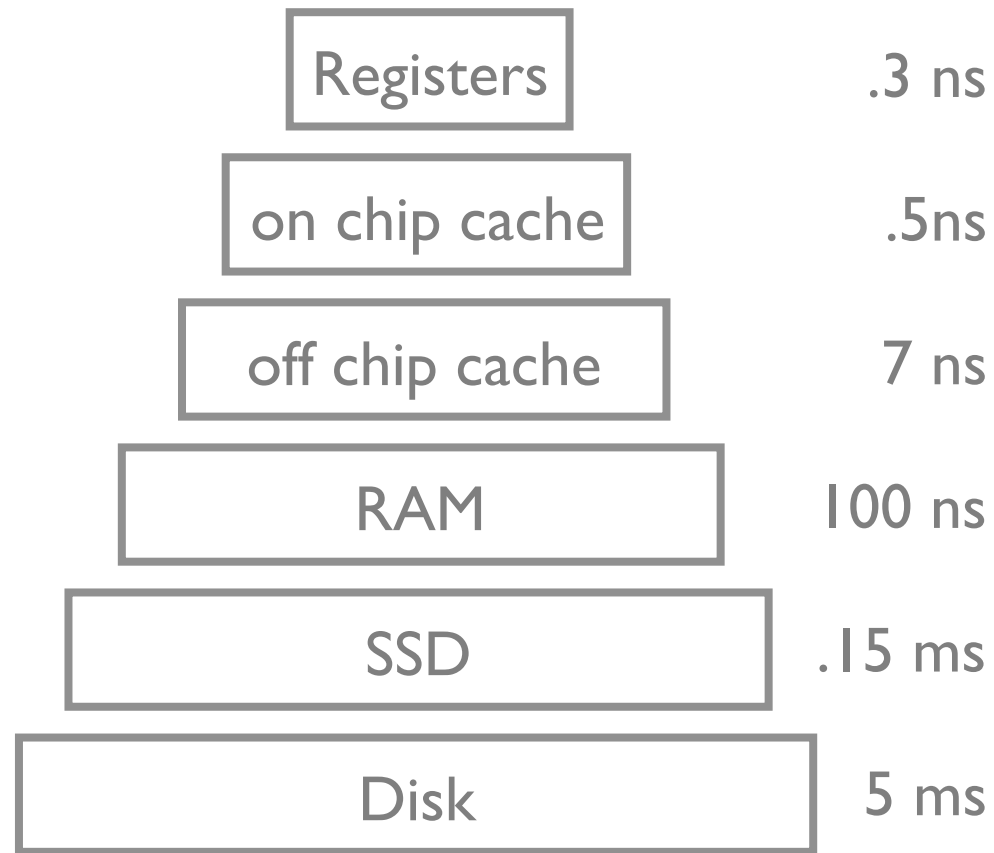
SSD: 400-1000

Disk: 24000

RAM for active data

Disk for main database
secondary storage

Tapes for archive

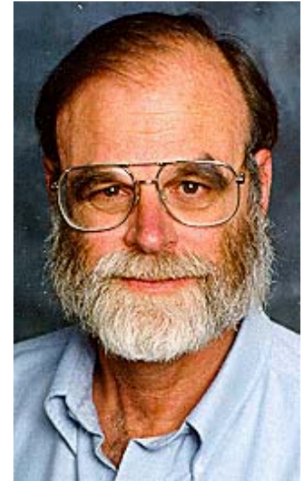


Interesting numbers

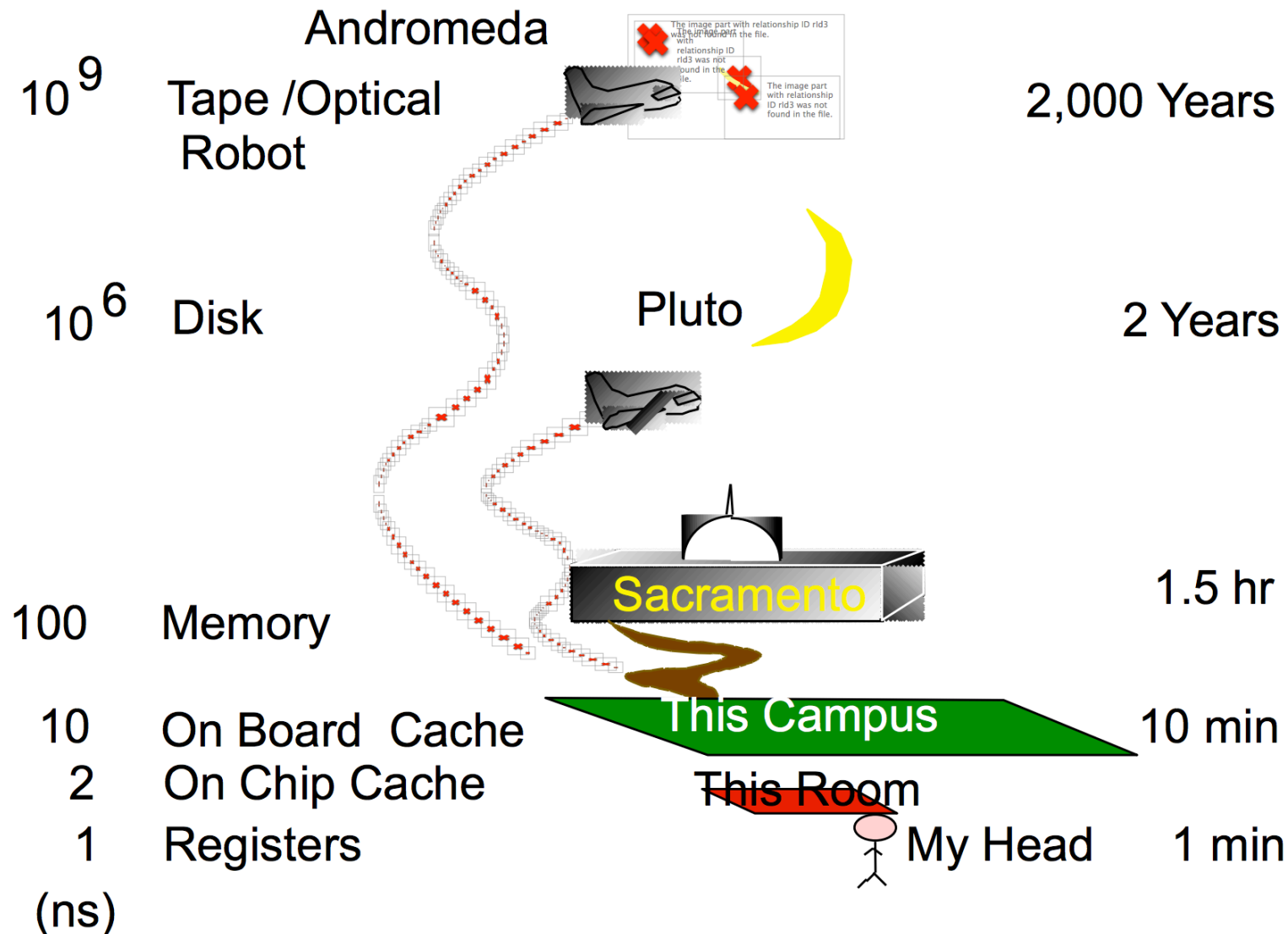
compress 1k bytes: 3000 ns

roundtrip in data center: .5 ms

Jim Gray's Storage Latency Analogy: How Far Away is the Data?

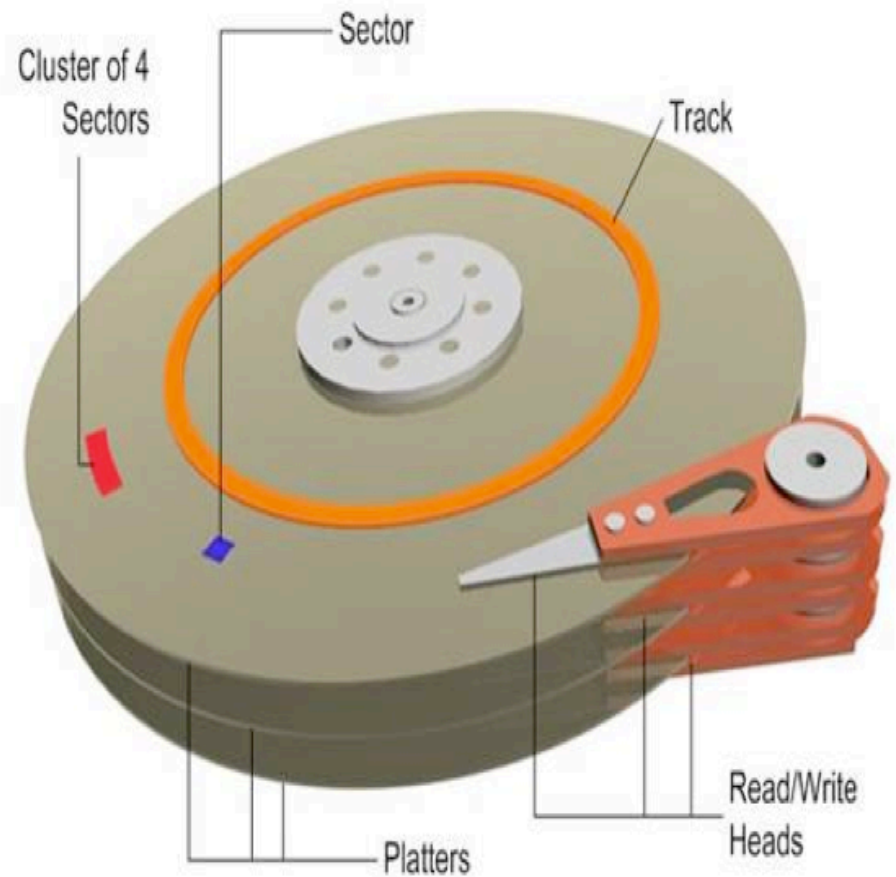


Jim Gray
Turing Award
B.S. Cal 1966
Ph.D. Cal 1969!



Spin speed: ~7200 RPM

Arm moved in/out to
position head over a
track



Time to access (read or write) a disk block

seek time	2-4 msec avg (arm movement)
rotational delay	2-4 msec (based on rotation speed)
transfer time	0.3 msec/64kb page

Throughput

read	~150 MB/sec
write	~50 MB/sec

Key: reduce seek and rotational delays

HW & SW approaches

Pre-fetching

Next block concept (in order of speed)

- blocks on same track

- blocks on same cylinder

- blocks on adjacent cylinder

Sequentially arrange files

- minimize seek and rotation latency

When sequentially scanning: Pre-fetch

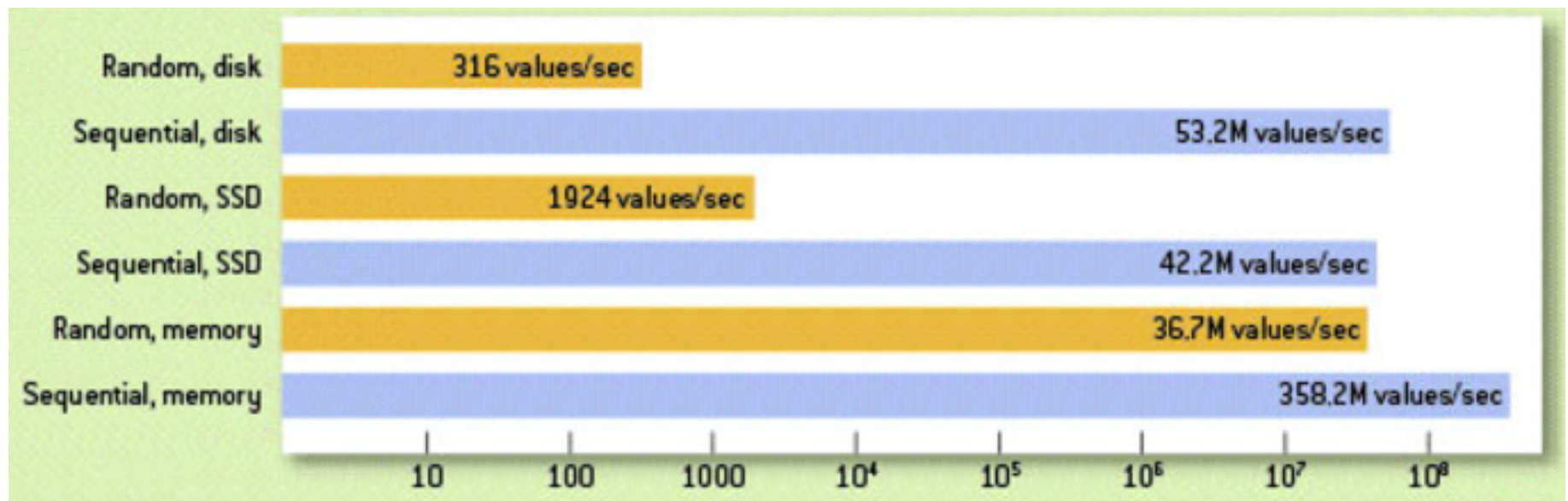
- > 1 page/block at once

SSD versus Hard Drives

Disks are not dead!

	HDD WD Black 6 TB	SDD Samsung 850 Pro	SDD Factor
Sequential Throughput	214 MB/s	496 MB/s	2.3X
Random Throughput	0.5 MB/s	273 MB/s	546X
Random IO Latency	4800 us	8 us	600X
\$/GB	\$0.05	\$0.46	0.1X

4 byte values read per second



5 orders of magnitude

Pragmatics of Databases

Most databases are pretty small

All global daily weather since 1929: 20GB

2000 US Census: 200GB

2009 english wikipedia: 14GB

Data sizes grow faster than Moore's law:

Record everything; storage is cheap?

Disk Space Management

VLDBs

Small DBs

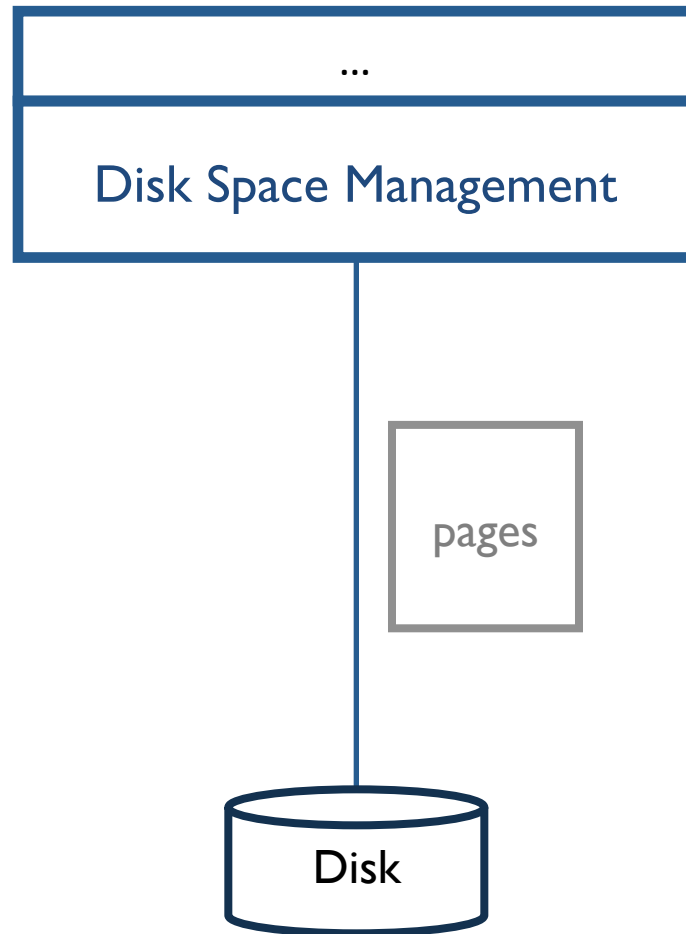
Huge data exists

Many interesting data are small

People still worry about magnetic disk

May not care about it

Work from the bottom up



What is a page?

Unit of transfer between storage and database

Typically fixed size

Small enough for one I/O to be fast

Big enough to not be wasteful

Usually a multiple of 4 kB

- Intel virtual memory hardware page size

- Modern disk sector size (minimum I/O size)

Random example sizes

SQLite: 1 kB

IBM DB2: 4 kB

Postgres: 8 kB pages

SQL Server: 8 kB

MySQL: 16 kB

MongoDB (Wired Tiger): 32 kB

Disk Space Management

Manages space on disk, IO, and caching

Sequential performance desirable

- hidden from rest of DBMS

- some algorithms assume sequential performance

Example Disk Space Interface

DiskInterface:

readPage(page_id): data

writePage(page_id, data)

newPage(): page_id

freePage(page_id)

Records and Files

Record: “application” storage unit

e.g. a row in a table

Page: Collection of records

File: Collection of pages

insert/delete/modify record

get(record_id) a record

scan all records

May be in multiple OS files spanning multiple disks

Units that we'll care about

Ignore CPU cost

Ignore RAM cost

B # *data* pages on disk for relation

R # records per data page

D avg time to read/write data page to/from disk

Simplifies life when computing costs

Very rough approximation, but OK for now

ignores prefetching, bulk writes/reads, CPU/RAM

Unordered Heap Files

Unordered collection of records

Pages allocated as collection grows

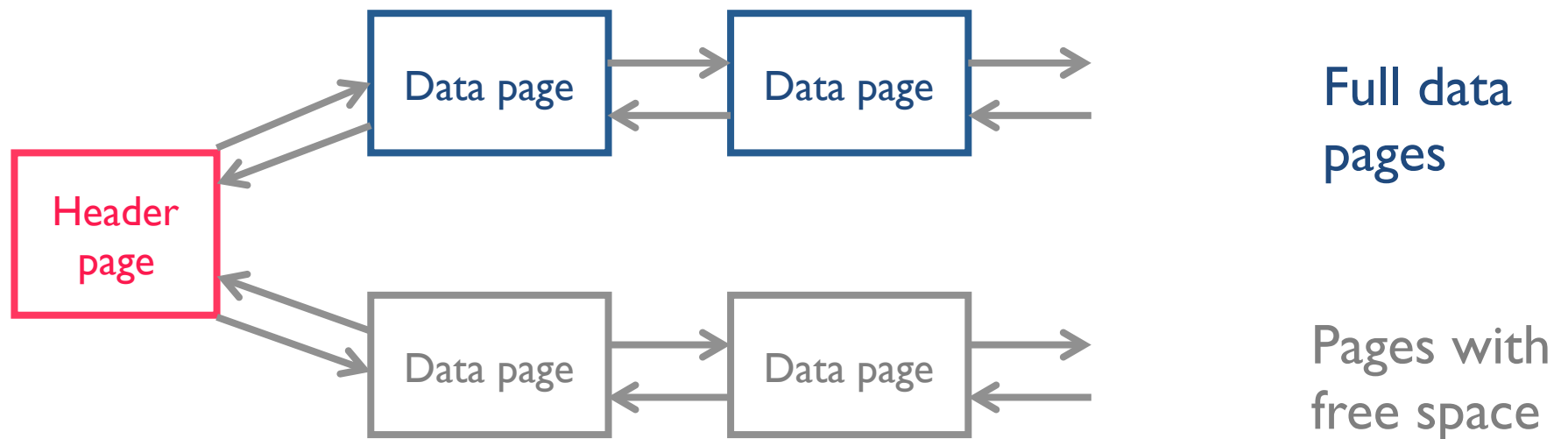
Need to track:

- pages in file

- free space on pages

- records on page

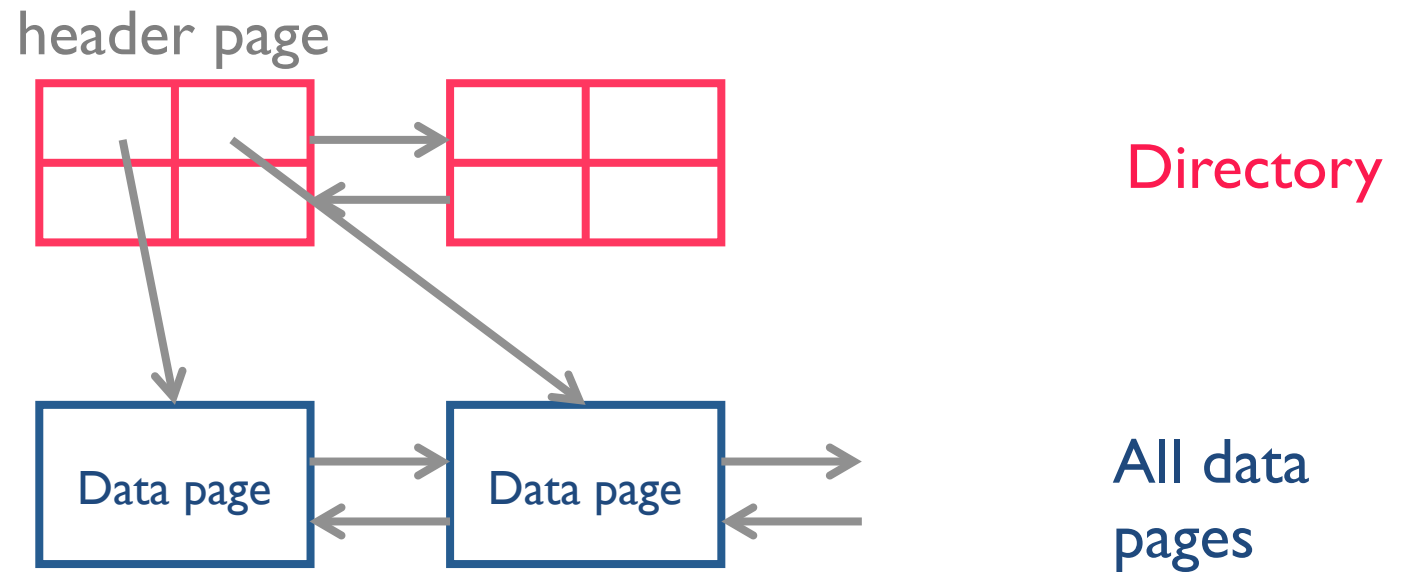
Potential Heap File Implementation



Header page location? Typically in the “catalog”

Data page = 2 pointers + data

Alternative: Use a directory



Directory entries track #free bytes on data pages

Directory is collection of pages

Indexes

“If I had eight hours to chop down a tree,
I'd spend six sharpening my ax.”

Abraham Lincoln

Indexes

Heap files can get data by sequential scan

Queries use *qualifications* (predicates)

- find students in “CS”

- find students from CA

Indexes

- file structures for value-based queries

- B+-tree index (~1970s)

- Hash index

Overview! Details in 4112

Indexes

Defined wrt a *search key*

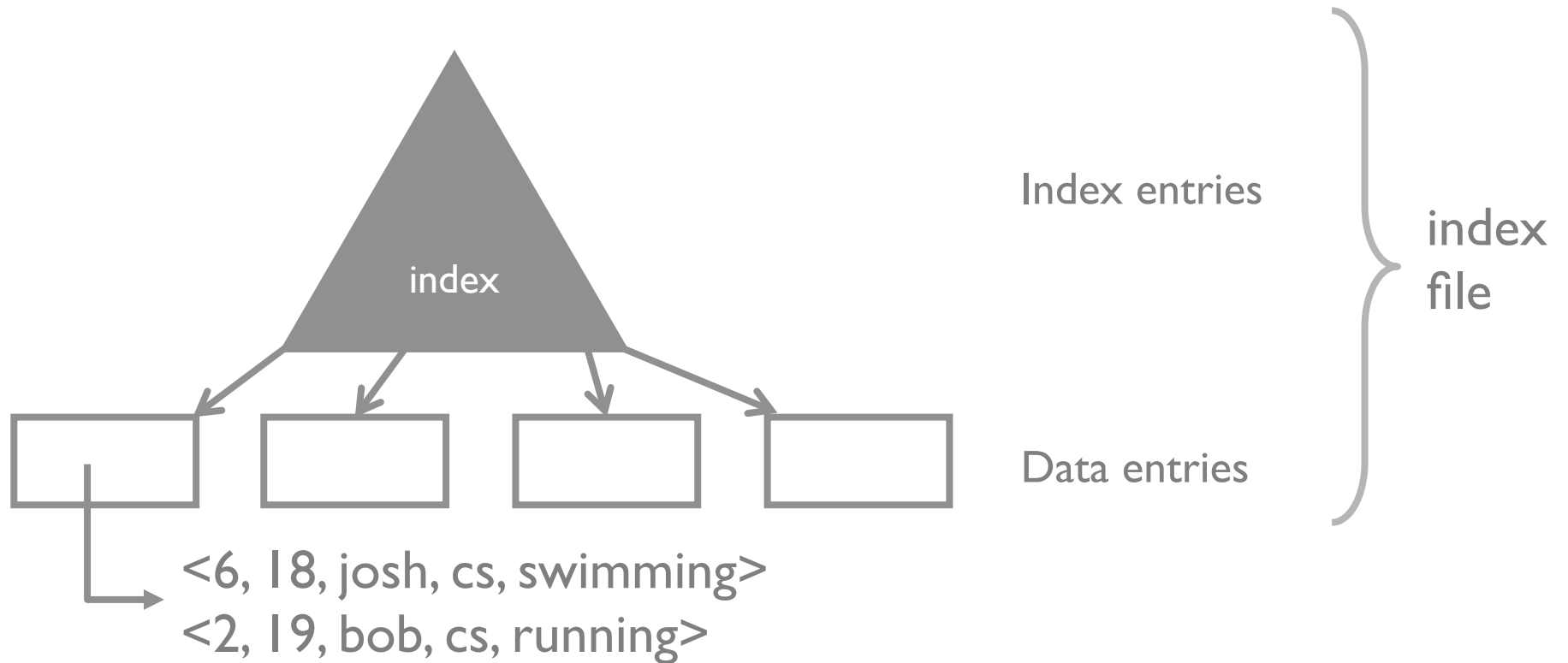
different than a candidate key!

Faster access for WHERE clauses w/ search key

```
CREATE INDEX idx1 ON users USING btree (sid)
CREATE INDEX idx2 ON users USING hash  (sid)
CREATE INDEX idx3 ON users USING btree (age,name)
```

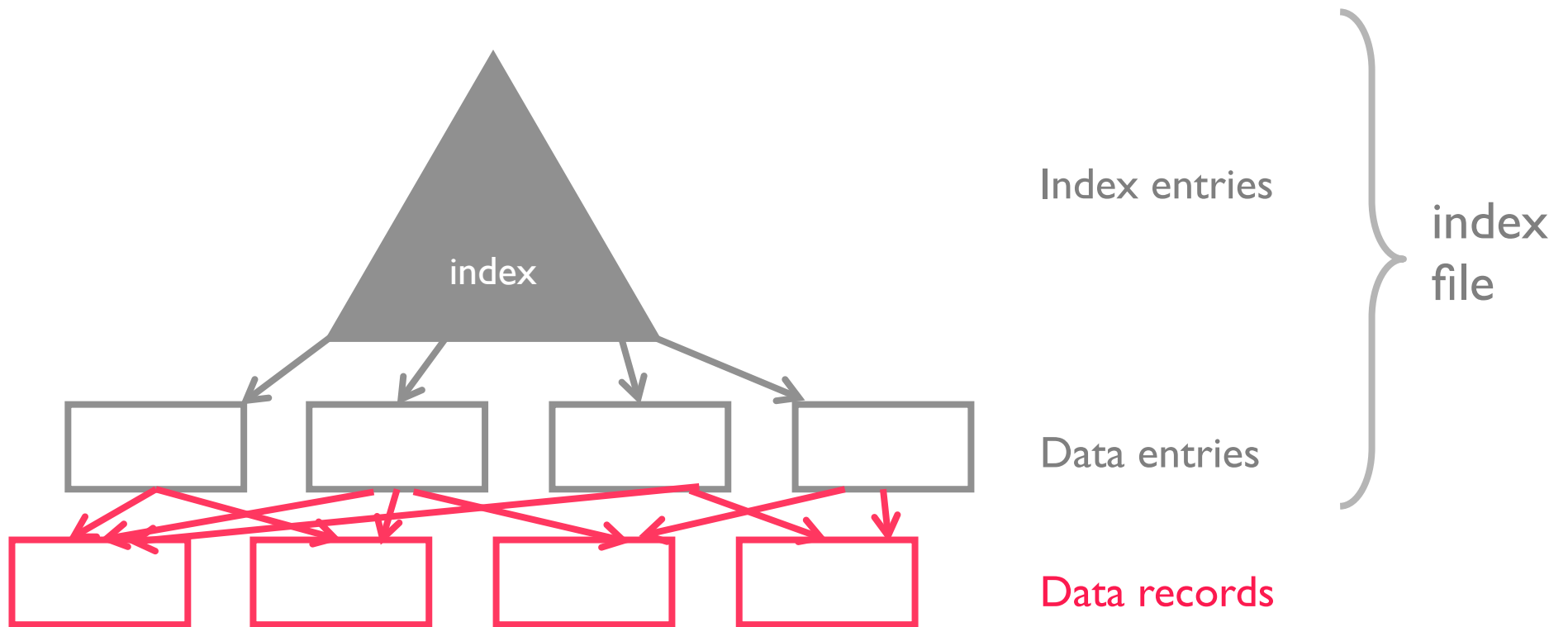
You will play around with indexes in HW4

High level (Primary) index structure



What is a data entry?
actual data record

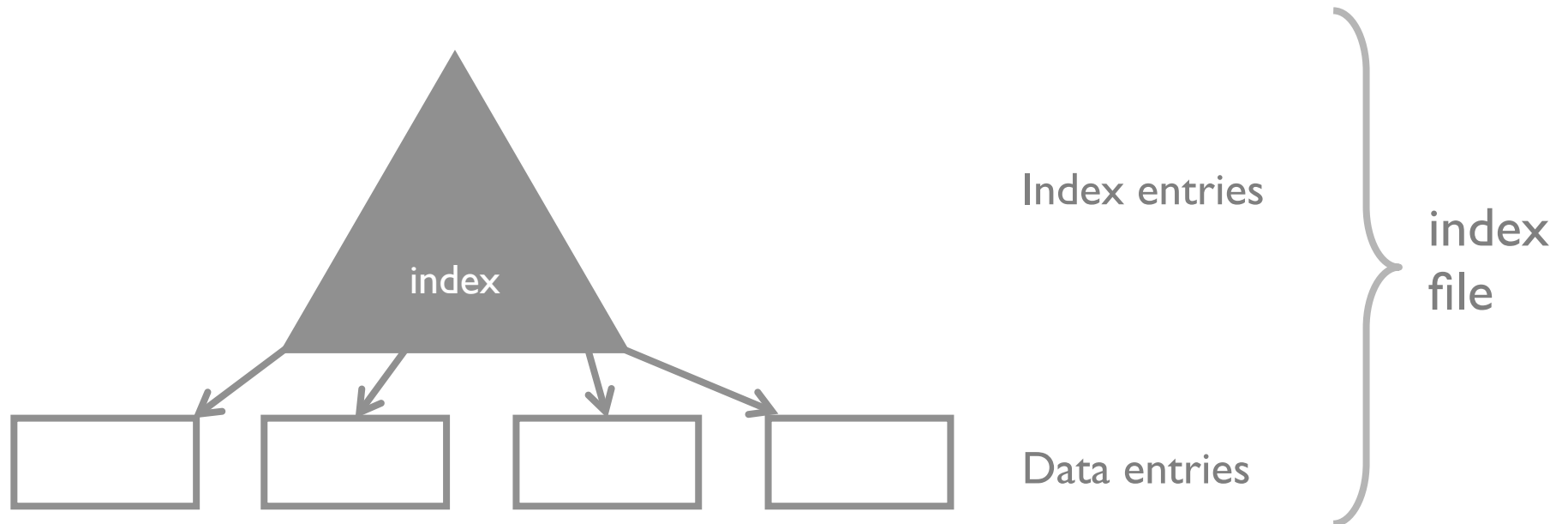
High level (Secondary) index structure



What is a data entry?
actual data record
<search key value, rid>

Tradeoffs
directly access tuple.
compact, fixed size entries

High level index structure



What is a data entry?

actual data record

<search key value, rid>

Tradeoffs

directly access tuple.

compact, fixed size entries