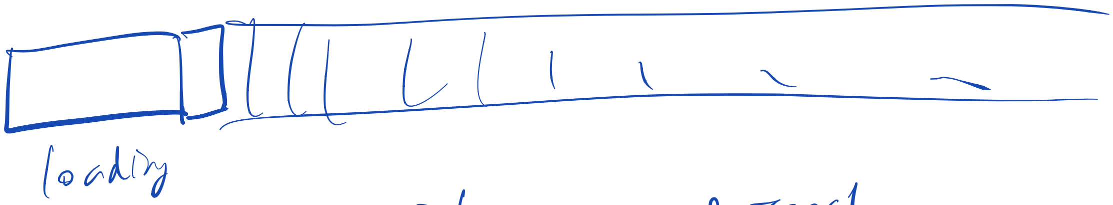
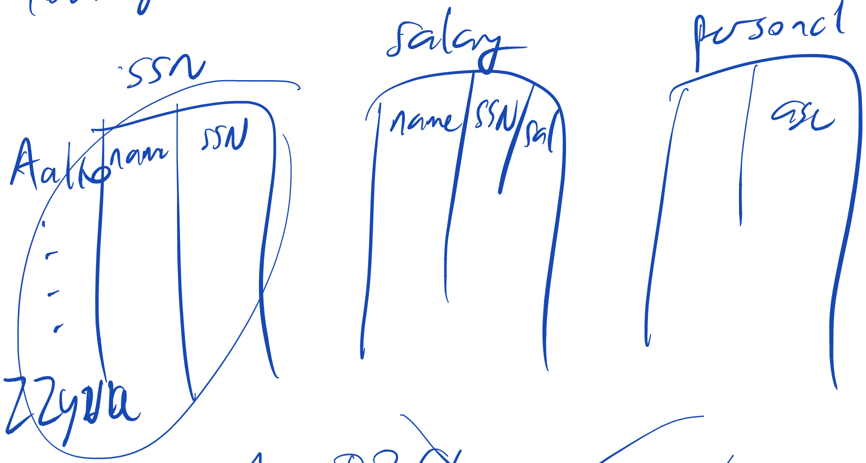
Parallel DBs

- multiple processors

  100 computers ~ 100x faster

- multiple disks



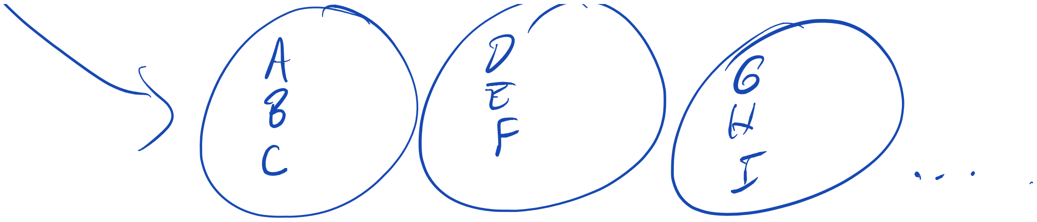


~~1 DB file → machine~~

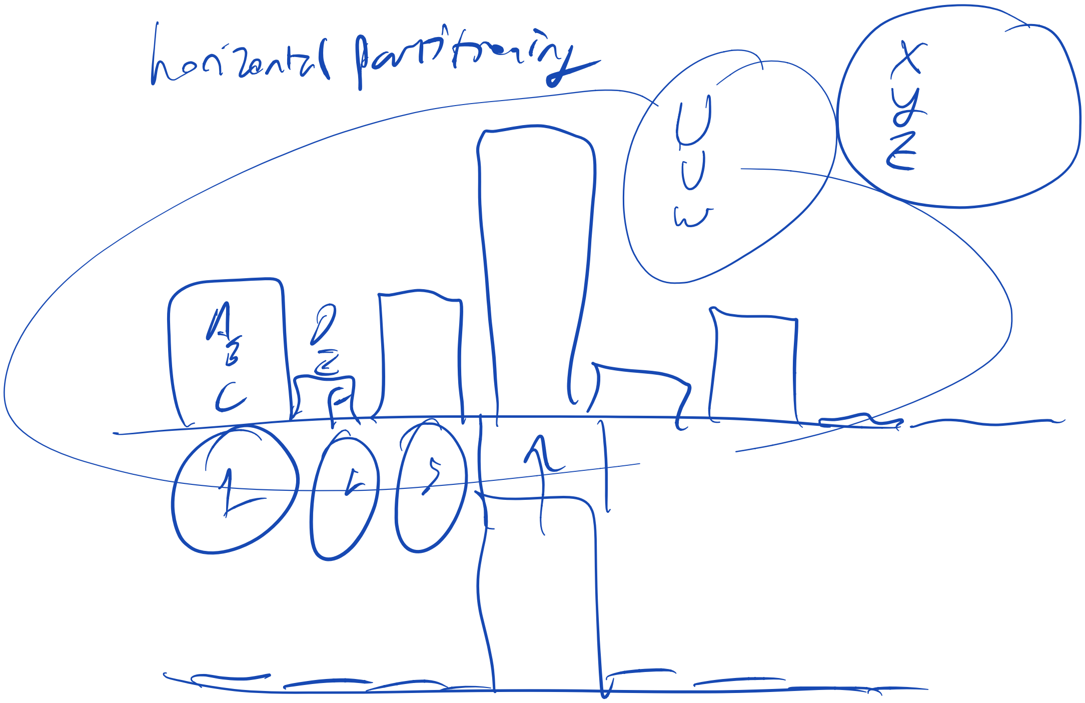
not much ||ism

may not fit

partitioning



horizontal partitioning



load balance

database(s) spread evenly

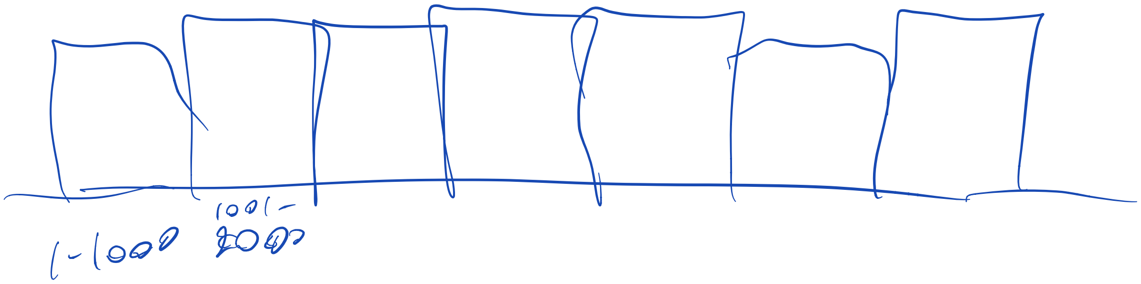
goal: hash fn "looks random"

hash

$$P[h(x) = \boxed{i}] = 1/N$$

1...N

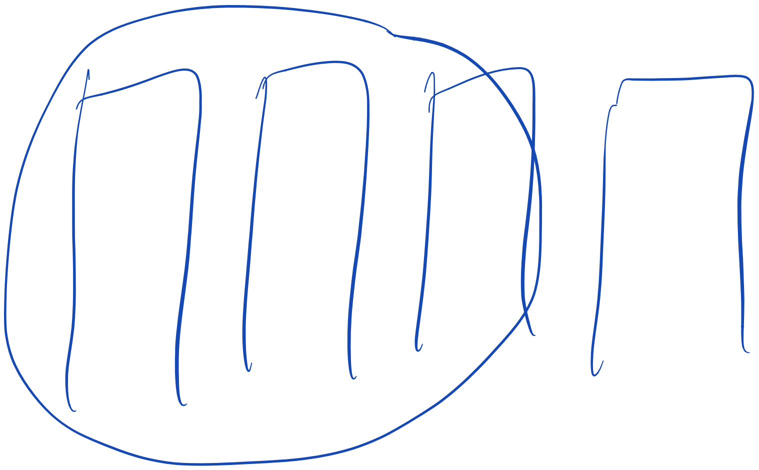
N machines

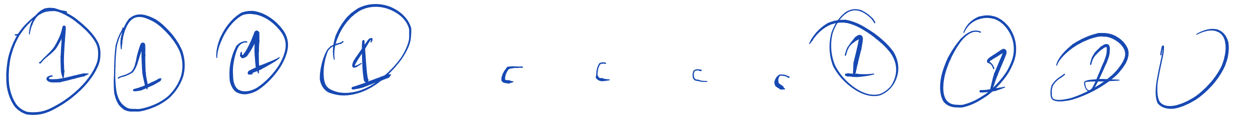


h("Joe Franklin") → 939715 % 100

N

machine 15





trade-off

maximize local comp.

||ism

maximize
maximize RAM utilization
minimize communication

DBMS

Schema

NAME STRING (30)
SSN STRING (9)
SALARY NUM . . . .

BLOBs

binary large object

IMAGE
variable-sized

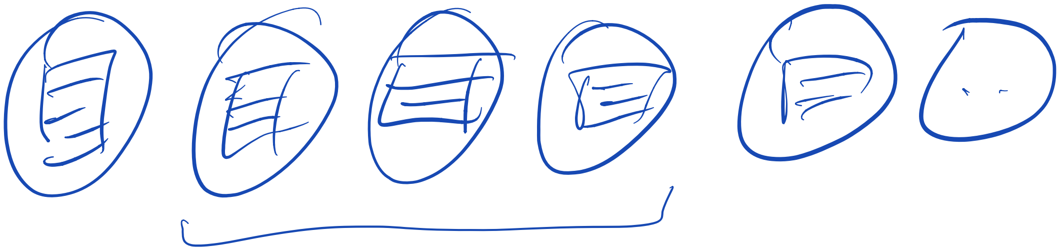
MapReduce

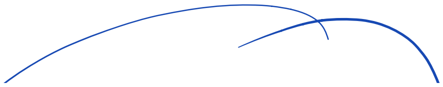
schemaless

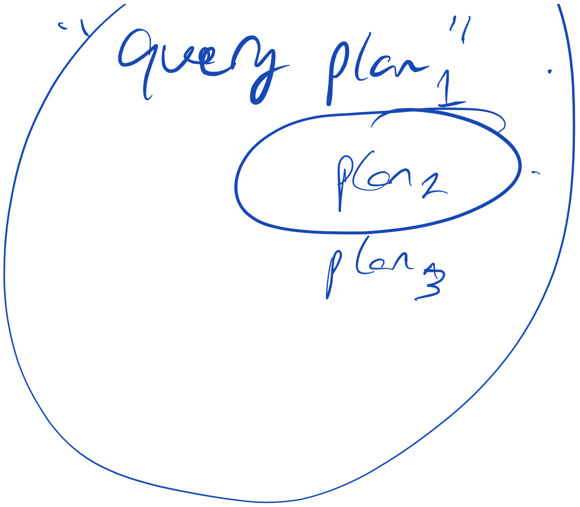
web pages
– "structured"
hierarchical
– "XML"/HTML

URL	word
	the
	news

hash partitioned







distribute, collect, ...

issue

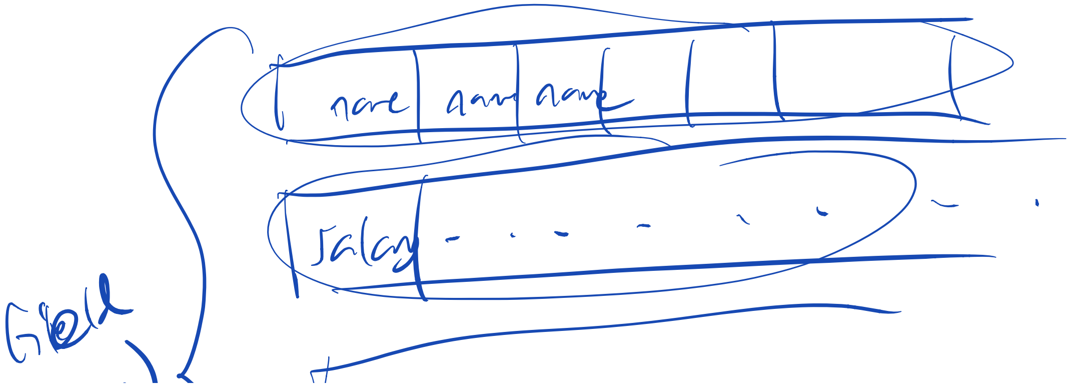
more comp.

more comm.

row-based

Column-based

name	salary	age	favorite col	BLOB portrait



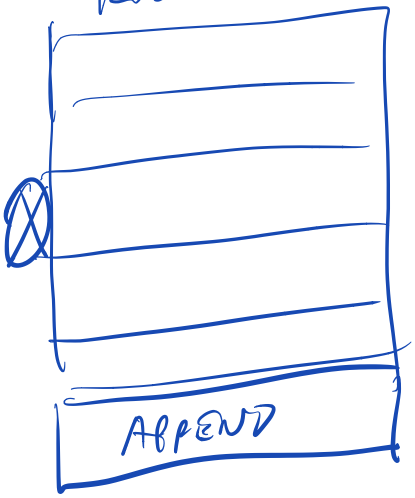
(column) in separate file

$$\text{I/O util.} \qquad \frac{\text{I/O I use}}{\text{I/O I do}}$$

$$\frac{16 \text{ bytes}}{4096 \text{ bytes}}$$

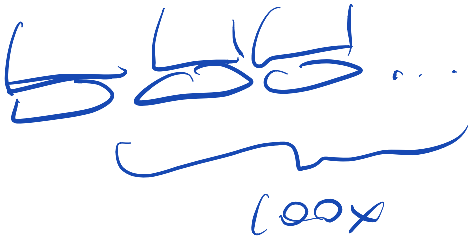
$$\text{100\% util.} \qquad \frac{4096 \text{ bytes}}{4096 \text{ bytes}}$$

ROW-BASED



COL-BASED

100 column



" it depends "

- workload
- machines

- interconnect
- drives
- memory

STATE

STATE
MA
MA
MA
MA
MA

10000

RLE

run length encoding

MA 10000

TIFF

Columns

⇒ compression

data analytics

read-only

read-mostly

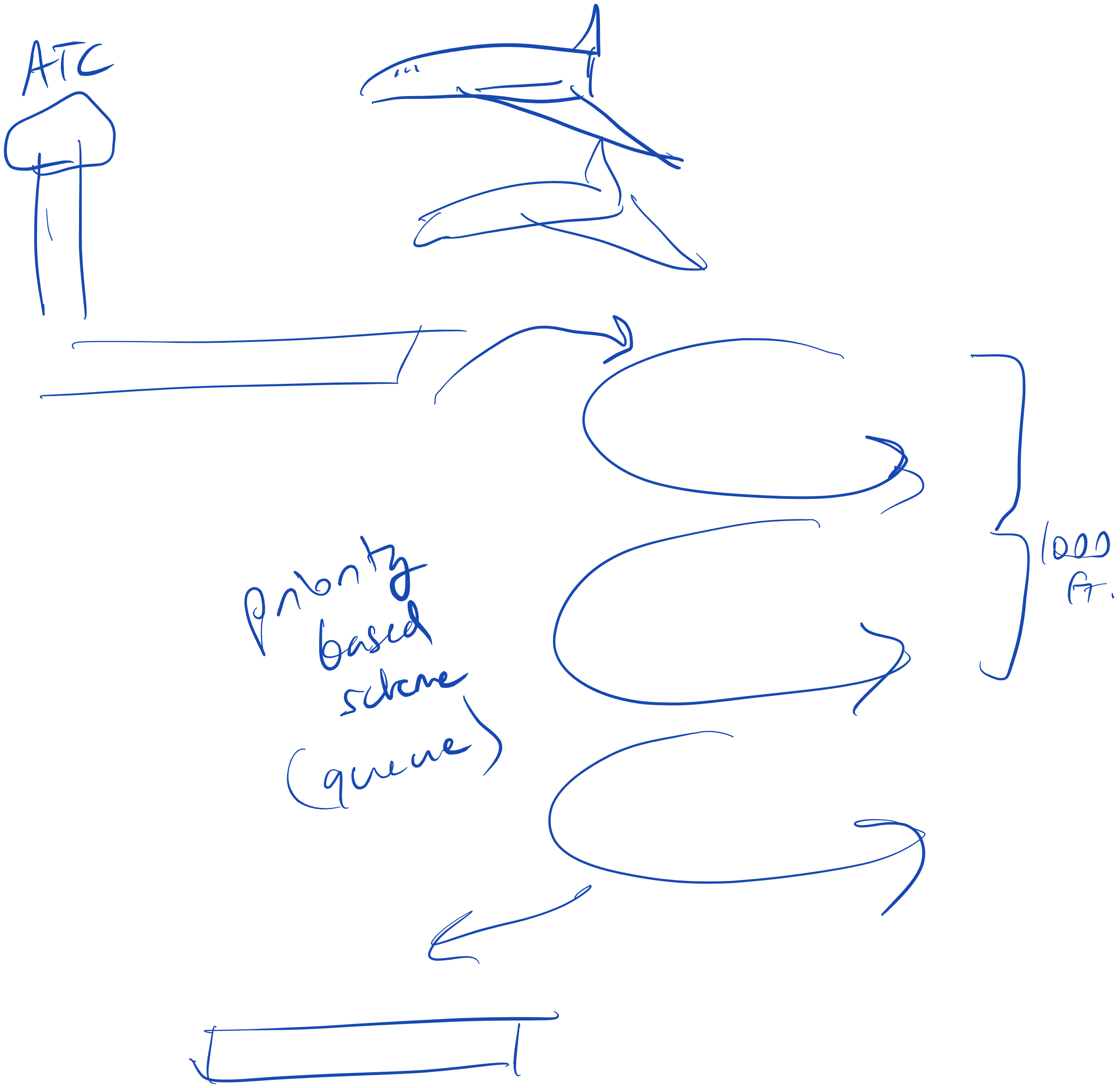
update-heavy

ETL

transaction processing

OLTP

DBMS



UPDATE WHERE ...
NAME "~~John Franklin~~" — YOUR NAME GOES HERE

SALARY = SALARY + 1,000,000

UPDATE ~~[illegible]~~ X

SALARY = SALARY * 1.02

transactions

ACID

Atomic	all or nothing
Consistent	good state → good state
Isolated	xactions can't see updates
Durable	saved to persistent store

FB

ACID overkill

"eventual consistency"

Where ~~[illegible]~~ ACID

[illegible] using ACID xactions

- Locking
- Optimizations for xactions
  (lock-free)

NoSQL — distrib. hash tables

key-value stores

put(k, v) stores k, v pair

get(k) → v