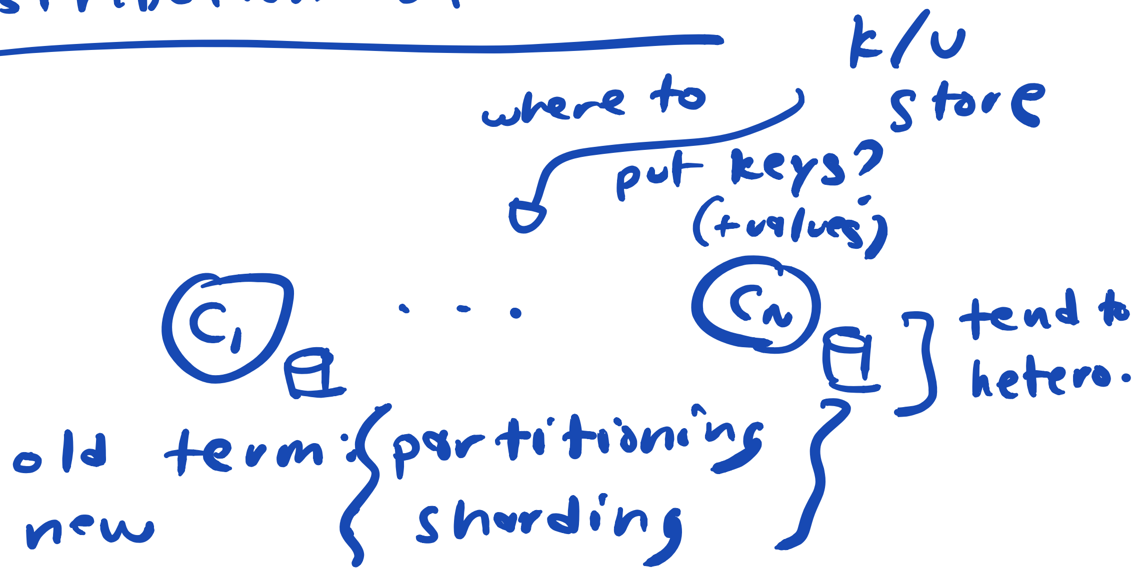
start @ 1:05 ← "Minutes Remain"

Load Balancing

⇒ load-balance.py

Distribution of data



Goals:

⇒ even (distribution)

- ⇒ space utilization (even w/ heterogeneous storage)
- ⇒ performance

=> handle growth / shrink

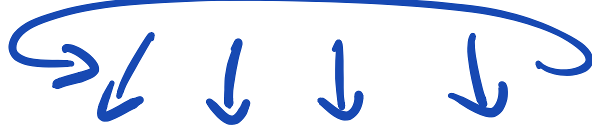
=> lookup : speed

=> scale

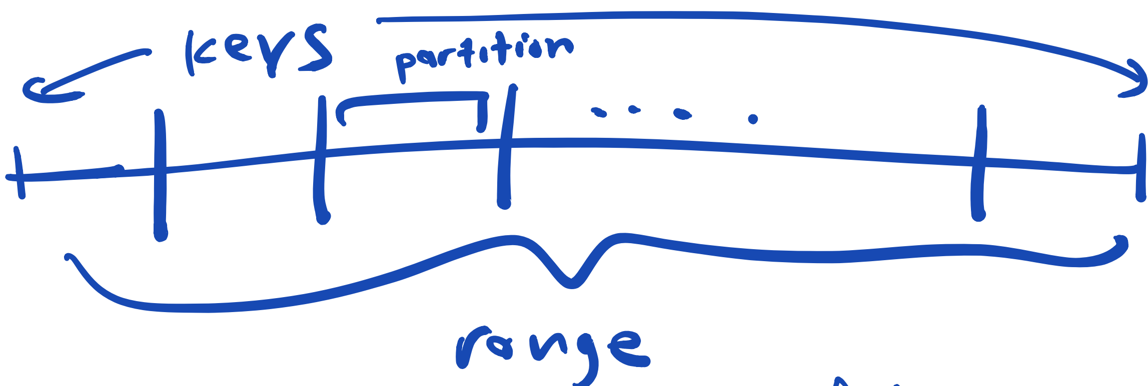
(Sharding)

Partitioning -> parallel databases (volcano, Gamma, RDB)

=> Round Robin

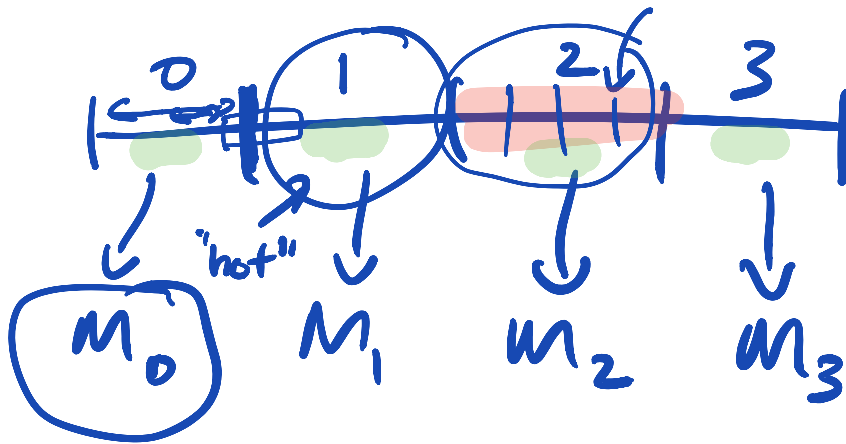


=> Range Partitioning



1) chop range => partitions

2) assign partitions => nodes



Positives

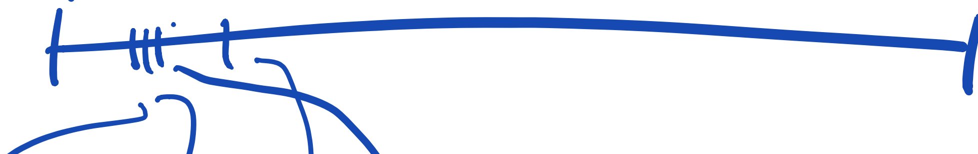
- ⇒ simple – minus except
- ⇒ efficient for range queries
- ⇒ user can know about "locality"

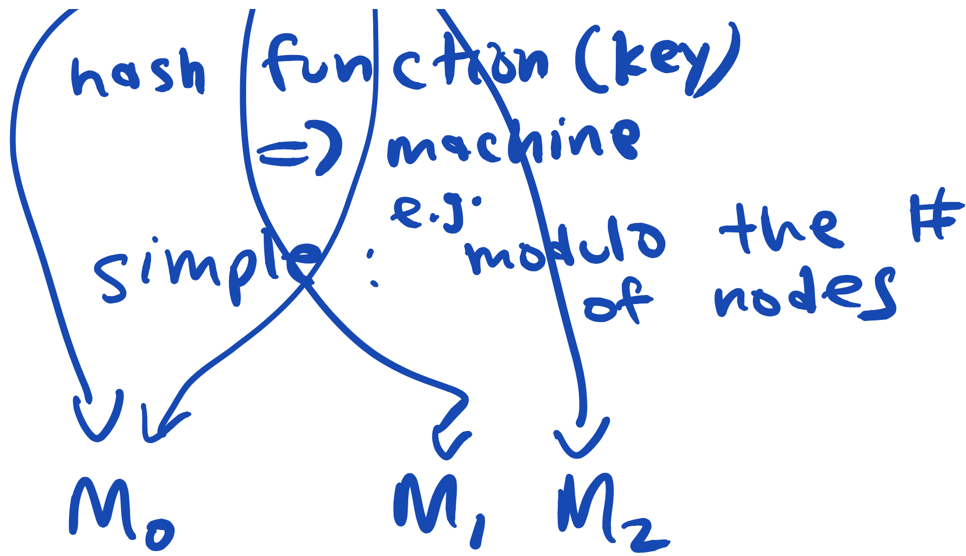
Negatives

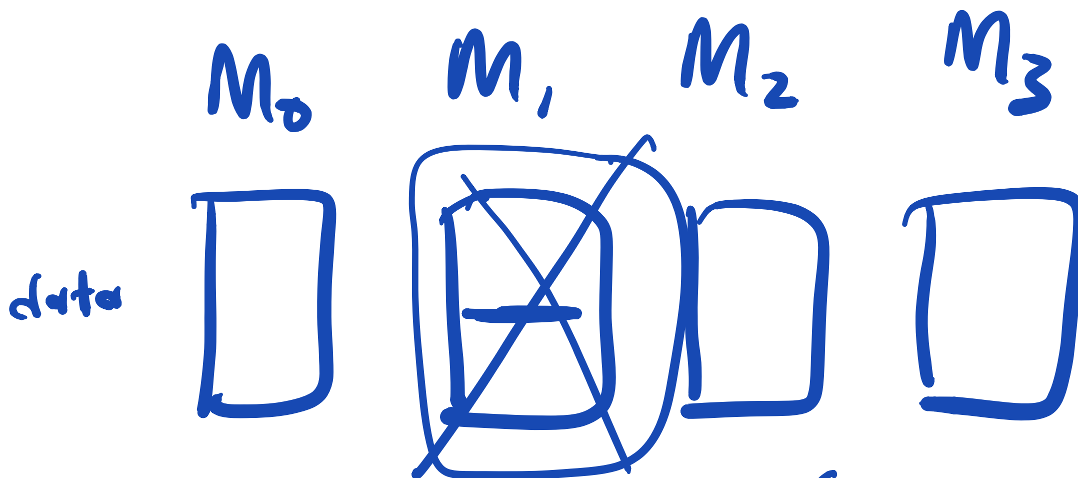
- ⇒ assumes lack of skew (load imbalance)
- ⇒ has to have mechanism to deal w/ load imbalance

⇒ Hash Partitioning

key space







pros

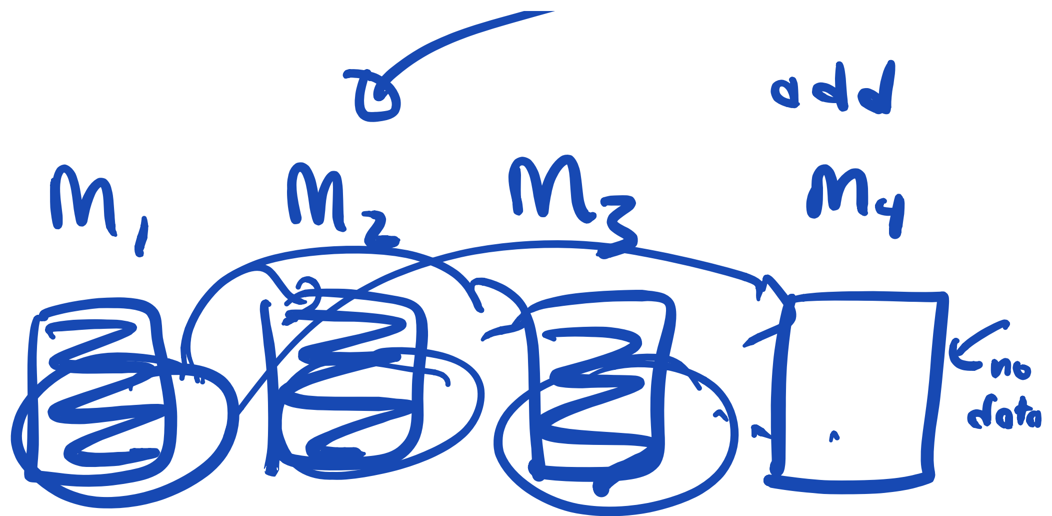
=> simple

=> distribute load pretty well

cons

=> no concept of user-managed locality

-> problems w/ growth/shrink



⇒ need: rehash all keys/values

⇒ most of data needs to move

Consistent Hashing:

⇒ hashing as basic distribution mechanism

but w/ better properties

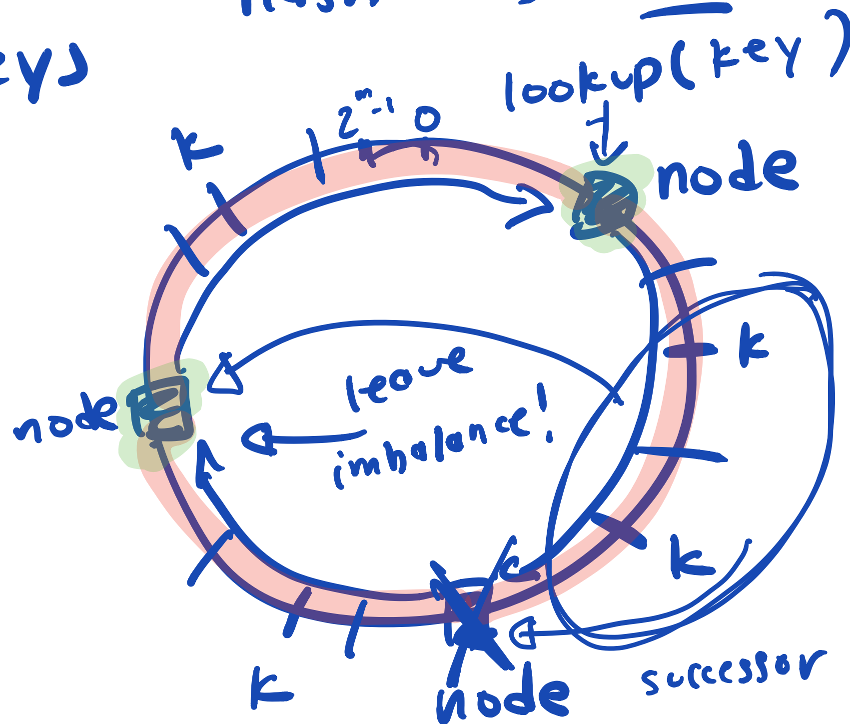
⇒ 1) node join / leave
redistribute minimal amount of data
(~ 1/N)

⇒ 2) scalability

managing info about
where keys live

Basic Approach:

nodes
hash ⇒ ID
keys



1) how to find anything?
2) " " scalably?
3) how to deal w/ change?
(e.g. node joining)
4) how to handle failure?

How to find anything?

info: node

[successor]

predecessor

lookup()

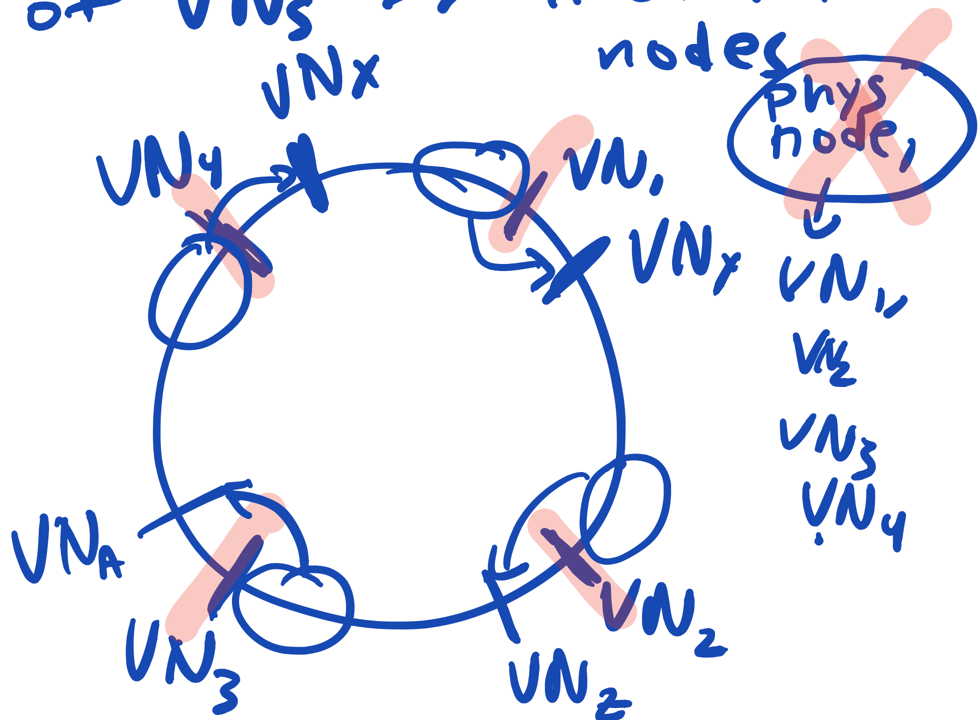
node: do I have this key?

{ if yes, reply
{ if no forward

Handling Load Imbalance:

⇒ virtual nodes (VNs)

# of VNs >> # of phys nodes



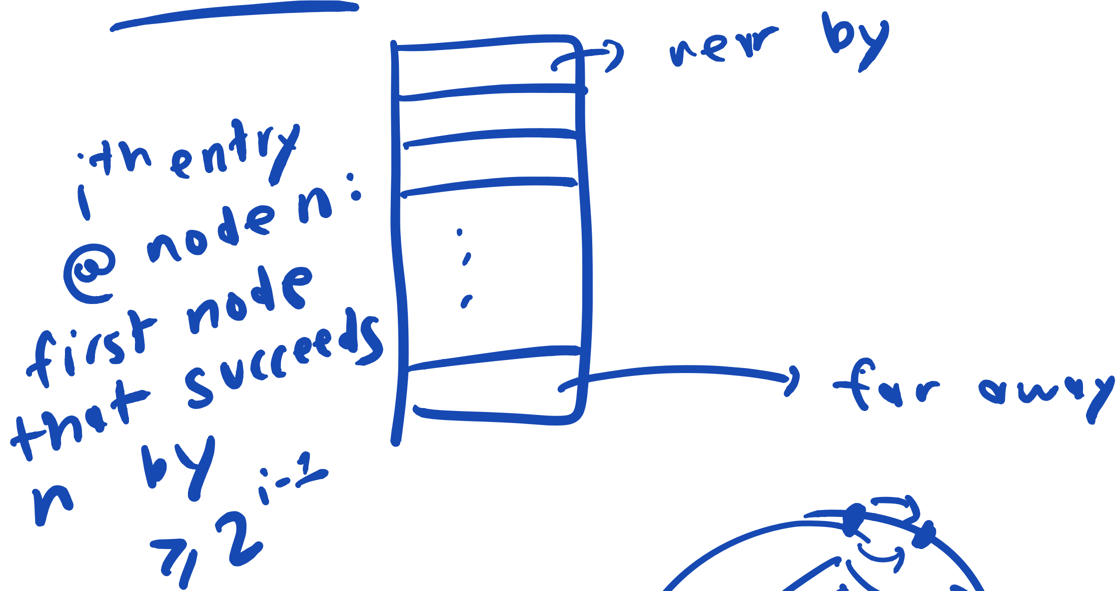
How to find info scalably?

[$O(\log N)$ not $O(N)$]

=> not linear

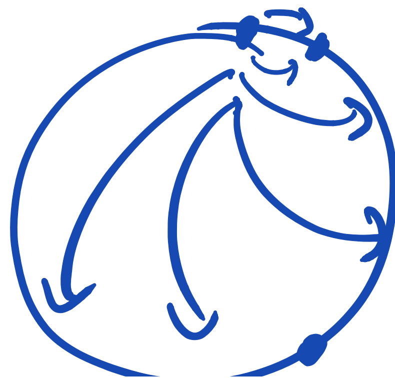
=> hop through nodes
> 1 hop @ time

finger table



routing:

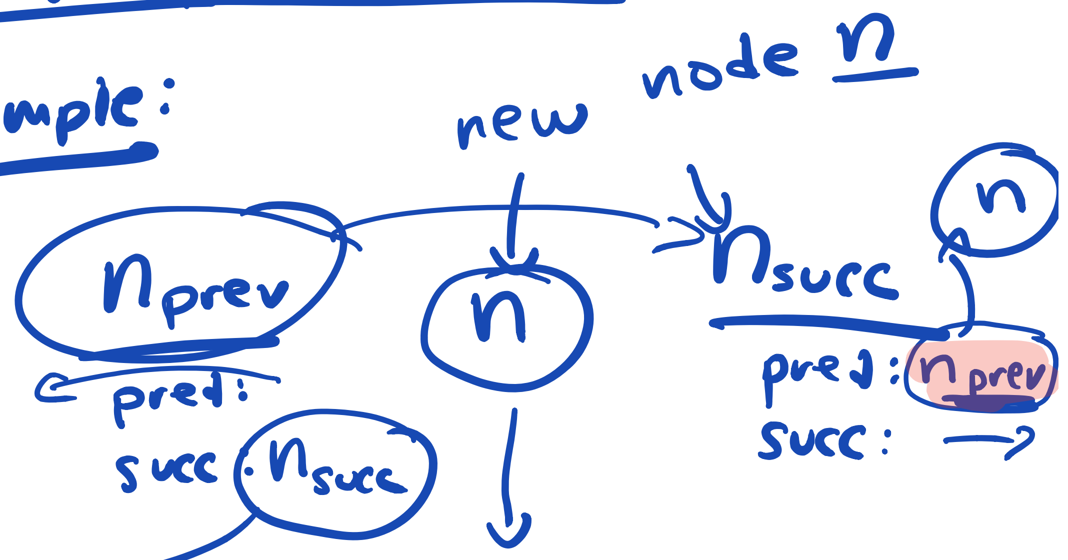
finger: get close



successors: to finalize

node join: process

Example:



not correct

call join():
⇒ route to key = n

n: from n_{succ}
pred: n_{prev}
succ: n_{succ}

nodes: periodically
call stabilize
⇒ to fix
up pred/succ

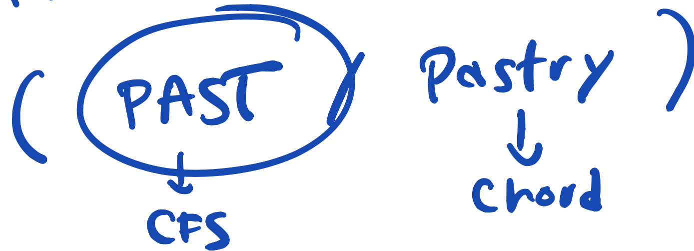
[similar for finger tables]

Fault Tolerance:

not 1 successor,
but R successors

CFS

first use of Chord



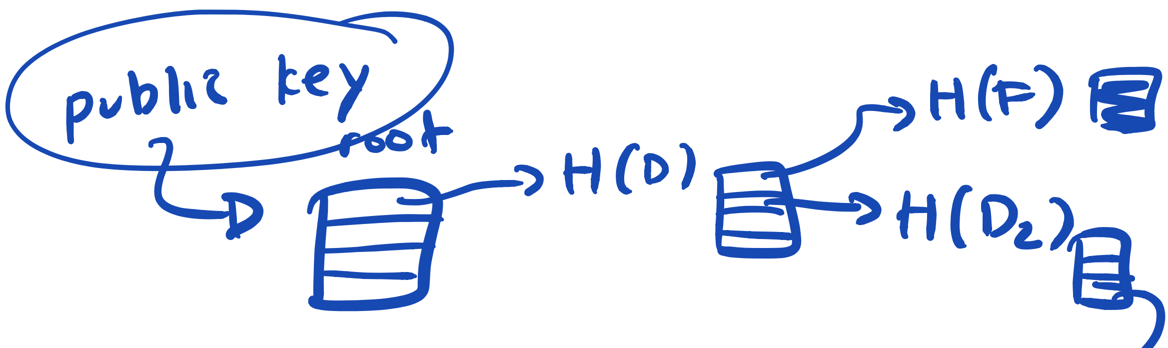
block-based storage: but "read-only"

publish once, read many times

content-based hashing:

how to refer to a block?

hash (data) => as key



Issue:
Block-level distribution

$H(F_3)$

PAST

⇒ file-level

different sizes

(might occur: file → node)

CFS

⇒ block-level

→ same (small) size

(→ lot of per-block CPU + net cost)

Replication

k copies of blocks

⇒ redundancy (not so much performance)

Caching

⇒ cache blocks along routing path

⇒ cache consistency?

easy: nothing changes

Limited Interface

(Publish (no delete))