https://github.com/emeryberger/Systems-DS-UPC-2017

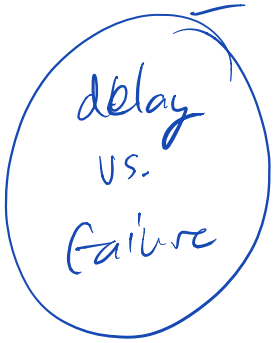
fail-stop vs. incorrect results

what about delays?

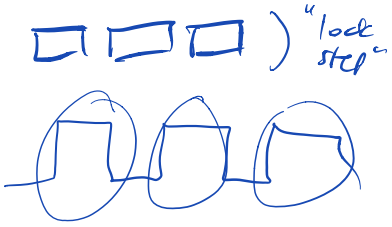
FLP result Fischer, Lynch, Paterson

asynchronous systems vs. synchronous

— distributed consensus impossible



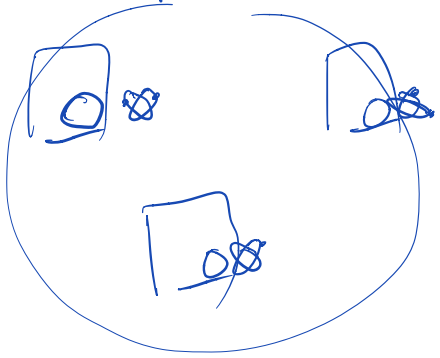
clock skew/ NTP drift



requires a clock

timeout

Spanner



databases

— semantics serializability transactions

— query language

(DB admin — indexes)

— scalable? & (MapReduce vs. DBMS)

scale up — takes adv. of all cores on a node

scale out — takes adv. of all nodes

P processors
N nodes

perf(P·2) ≈ 2·perf(P)

perf(N·2) ≈ 2·perf(N)

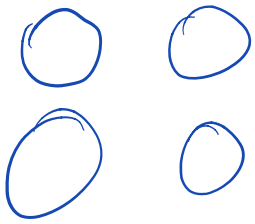
- linear scalability
- no scalability
- super linear scalability (!?)

```
for(i=0; i<10000; i++) {
  q = q * k;
}
```

5s on one processor — 5s on N processors

r1 r2 r3 r4
q = r1*r2*r3*r4

10,000/4



⇒ merge ⇒ 5s/4

[automatic parallelization

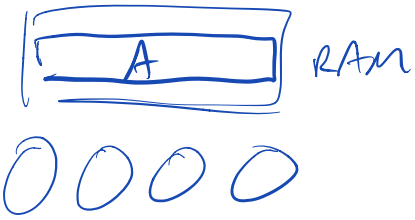
→ explicit parallelism (OpenMP)

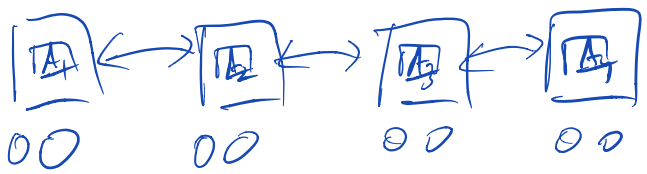
"embarrassingly parallel"

→ MT

```
for(   ) {
  x[i] .... x[i-1]
}
```

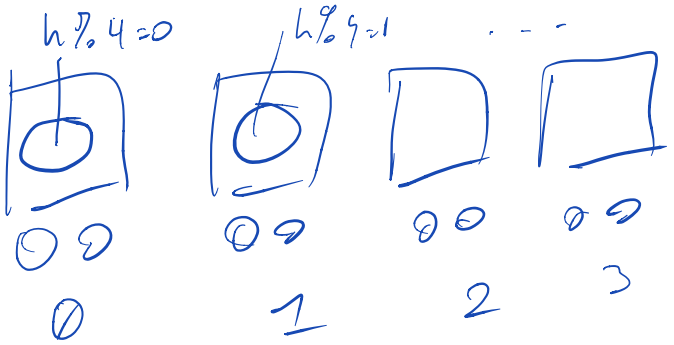
"shared memory system" multiple threads



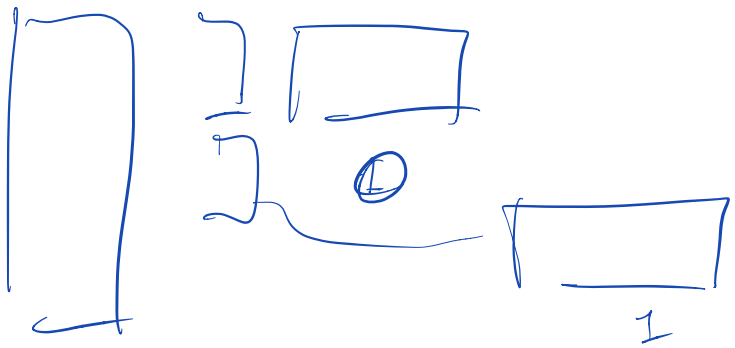


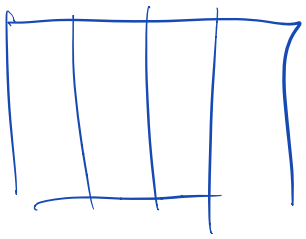
---

databases



Sharding





Scalability?

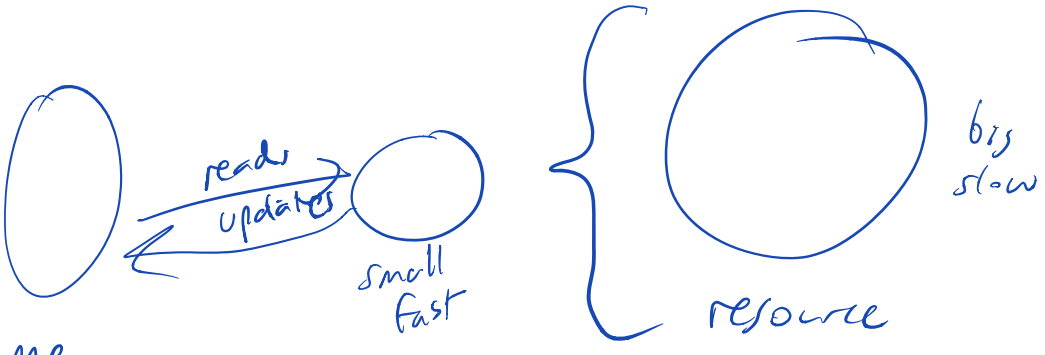
DB scale ~ 100

speed?

latency → high

generality

abstraction penalty



CACHE

mini DB

"hit"

in RAM

On disk

(ACID makes things slow)

Cache API:

v = get(key) ← string

set(key, value)

string

dict

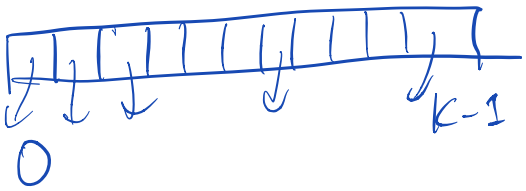
v = x[key]

x[key] = value

hash table

SHA-1 (str) ⇝ hash

string * arr[K];




```
get(key) {
    h = hash(key)
    ind = h % K;
    return arr[ind];
}
```

string *

```
set(key, value) {
    h = hash(key);
    ind = h % K;
    arr[ind] = value;
}
```

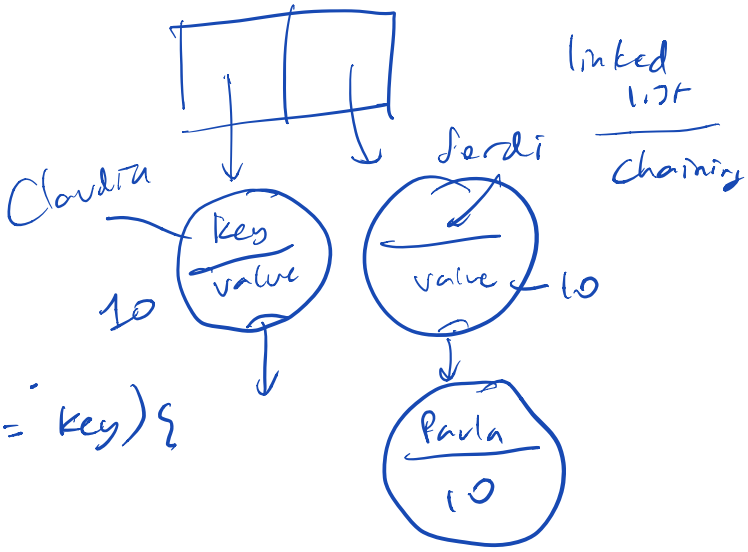
string *

set("Claudia", 10)
set("Jordi", 10)
set("Paula", 10)

pigeonhole principle

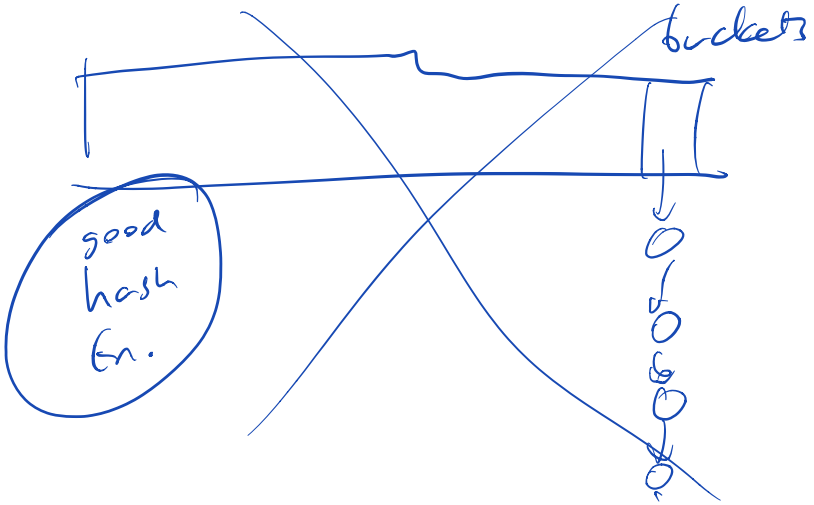
⟶ collisions

```
get(key)
  h = hash(key)
  ind = h % K
  p = arr[ind];
  while ( p
          != null
       && p->key != key) {
    p = p->next;
  }
  return p->value;
}
```



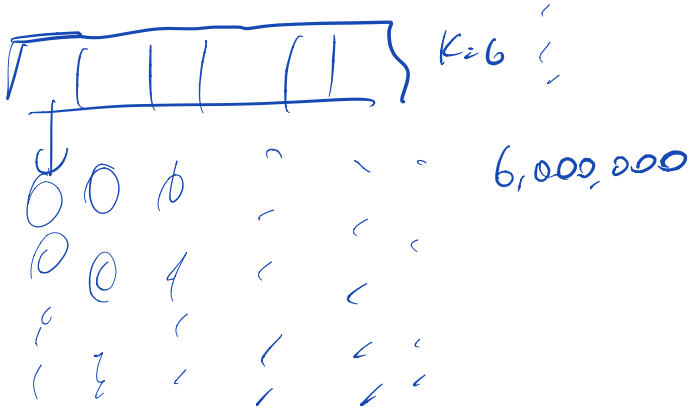
DoS attack

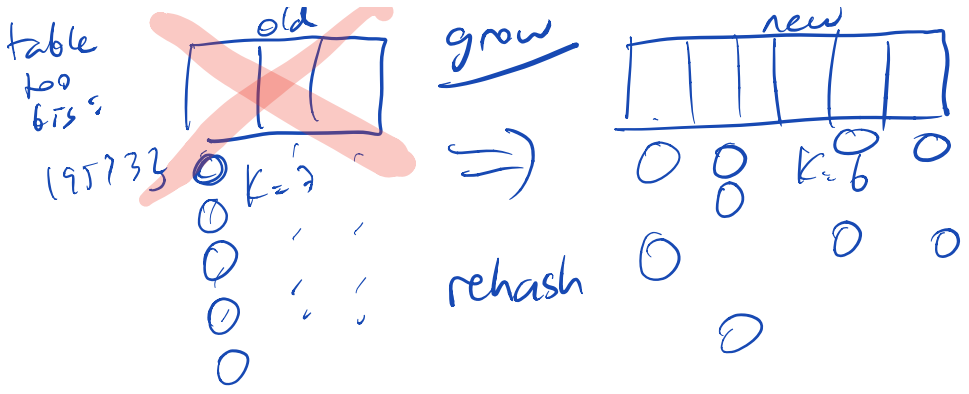
denial of service



DDoS

distributed

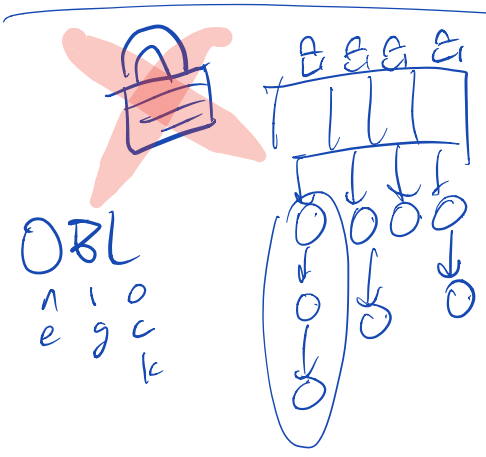




dynamic hash table

key = /upc/system-ds/paula/ims1.jpg

value = 1 MB



OBL

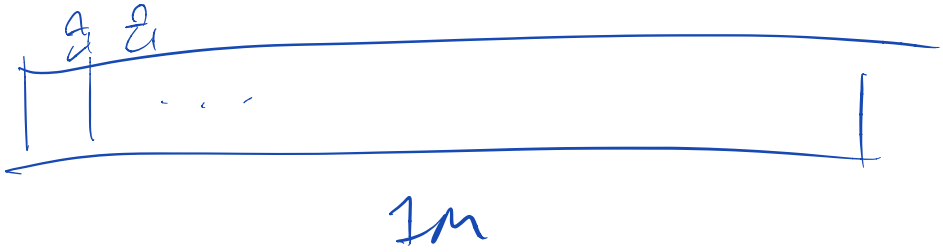
per-bucket locks

```
get(key) {
  h = hash(key)
  ind = h % K
  lock(table)   tablelock[ind]
    {arr[ind] ... }
  unlock(table) tablelock[ind]
}
```

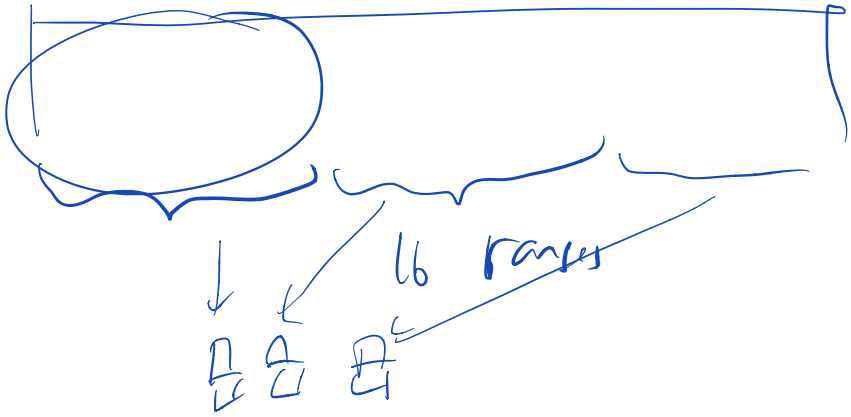
MTF

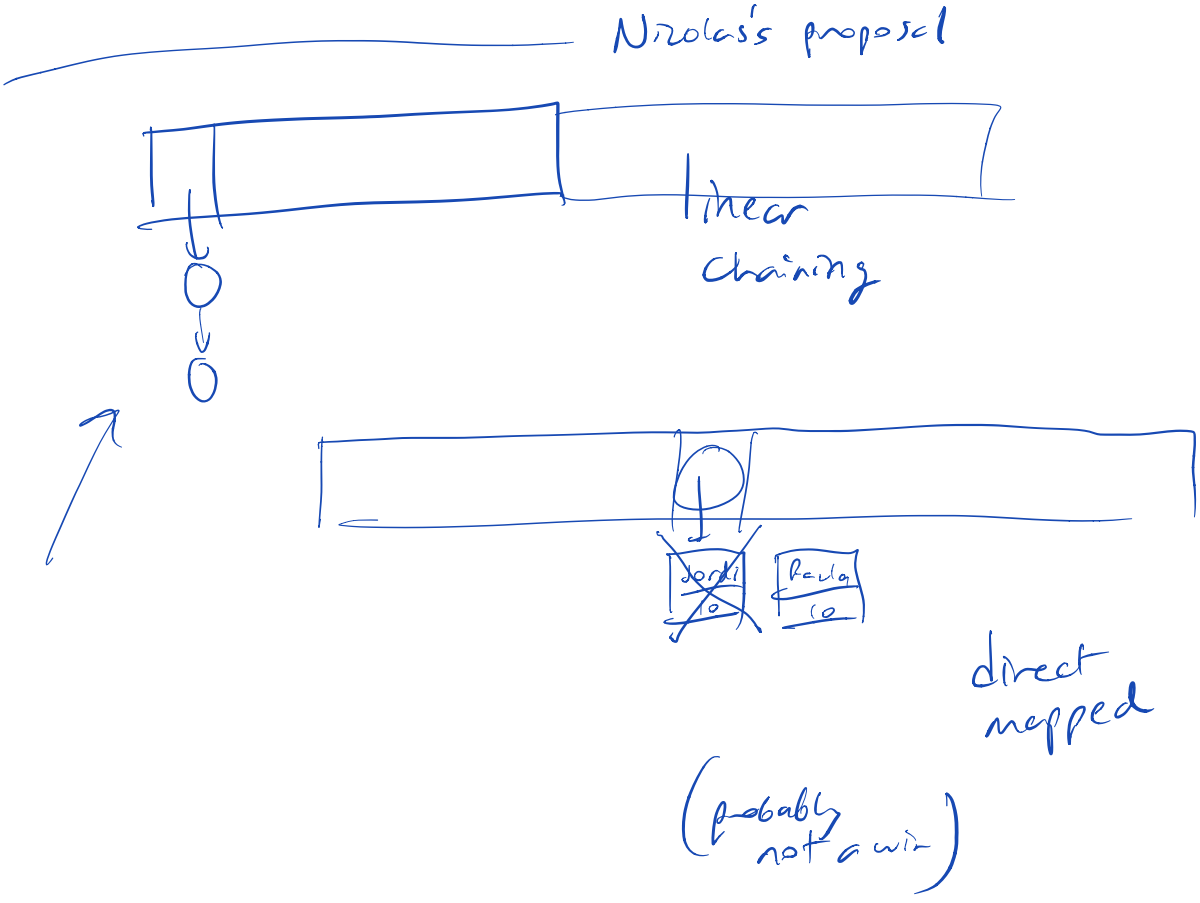
move-to-front

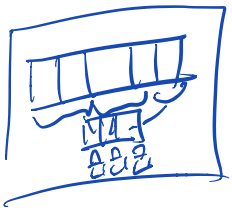
last item = LRU



1M

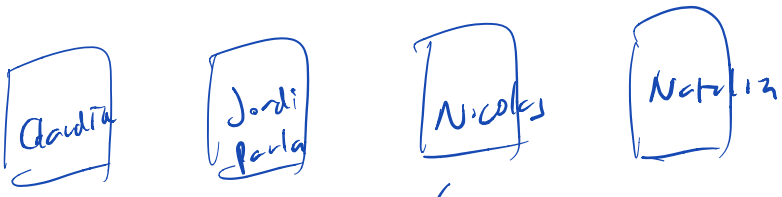

16 ranges


Nicolas's proposal
linear chaining
Jacki
Paula
direct mapped
(probably not a win)



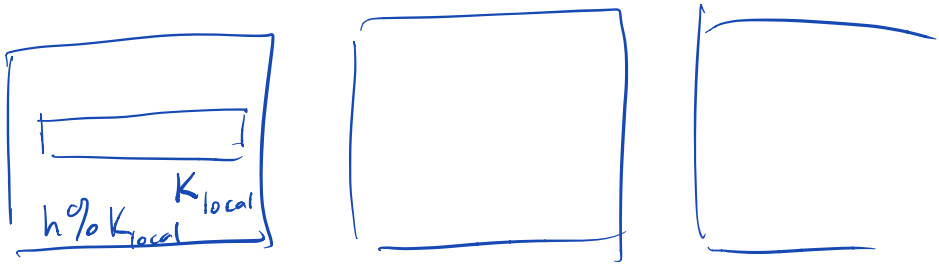
CPUs

distributed hash table




```
distribget(key) {
    h = hash(key)
    machine = h % 4
    v = machine.get(key)
    return v;
}
```

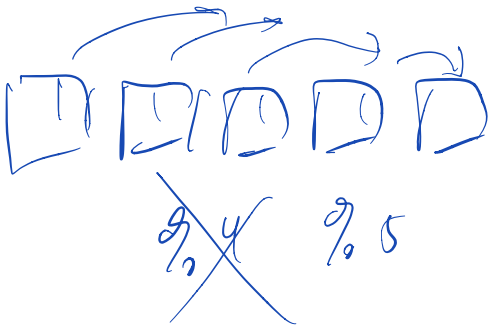
memcached



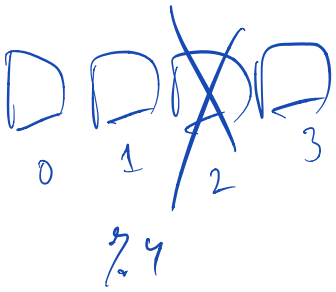
BIG PROBLEM

h % $K_{machines}$

adding or removing machines



rehash



P(failure) = p → .001

$1 - (1-p) \cdot (1-p)$

(1-p): no failure 1; (1-p): no failure 2

$1 - (1-p)^N$

at least one machine out of N fails

$N=10$: $P \sim .01$

$N=100$: $P \sim .1$

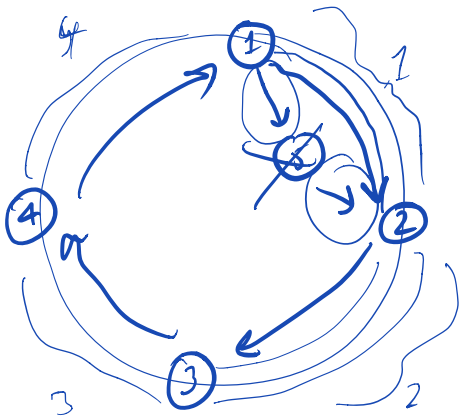
$N=1000$: $P = 1-(1-p)^{1000}$

$1 - .99^{1000}$

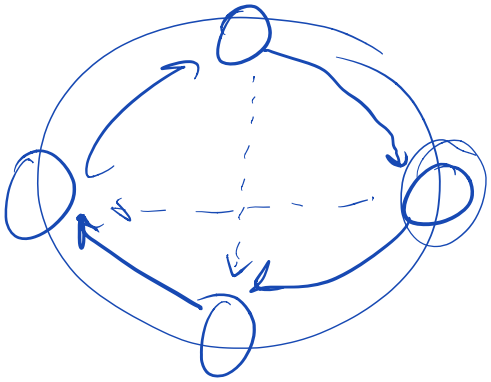
$\approx .63$

Consistent hashing

Chord

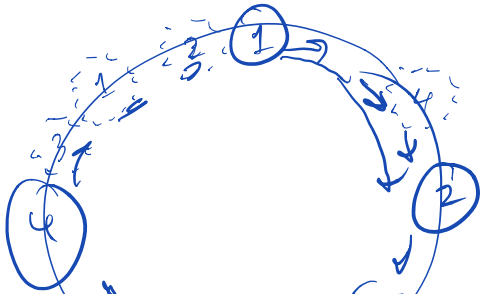


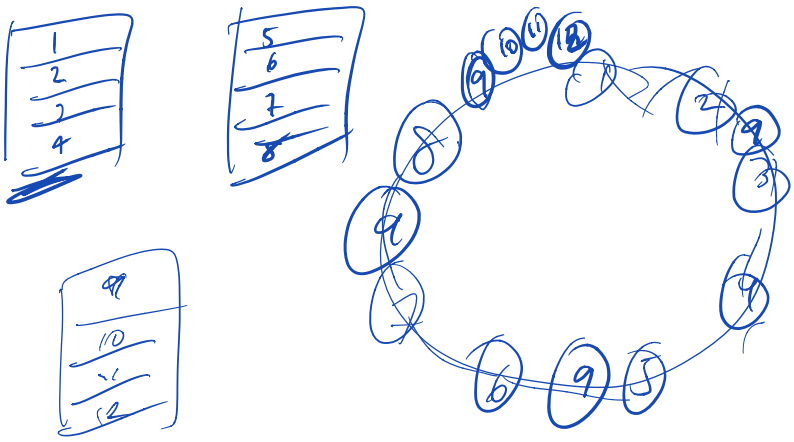
$O(N)$

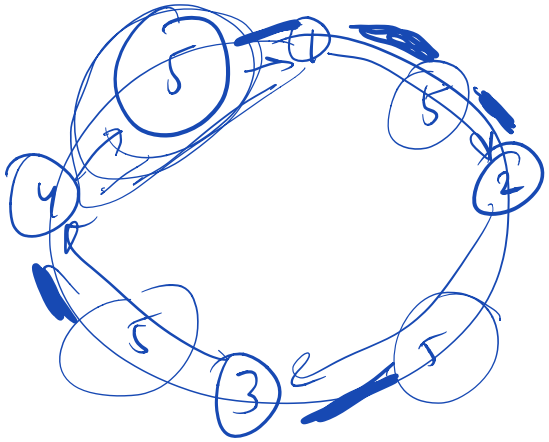


$O(\log N)$

"finger table"







1000 virtual nodes for every node

Chord

consistent hashing