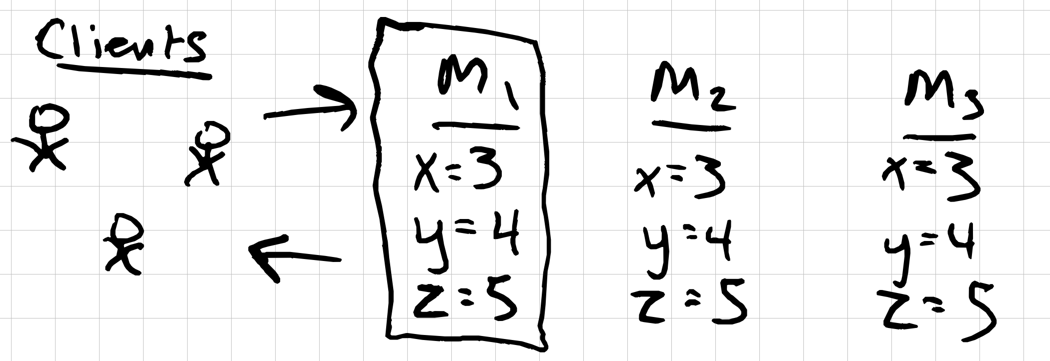
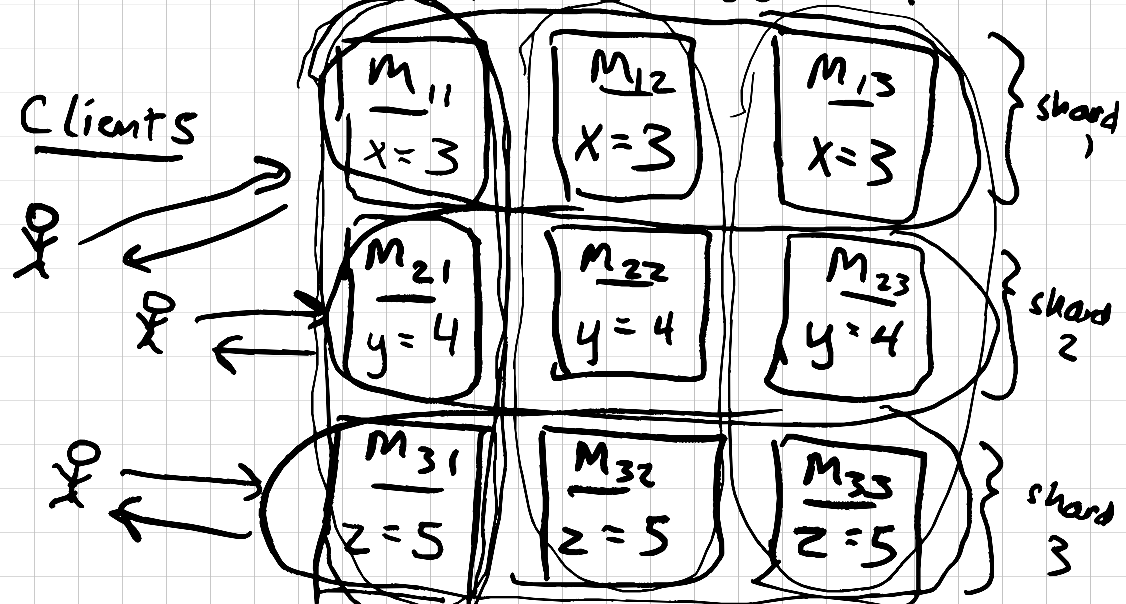
CSE138 Lecture 16

- intro to sharding (aka data partitioning)
- consistent hashing



One issue with storing all data on all replicas:

- What if there's more data than can fit on one machine?

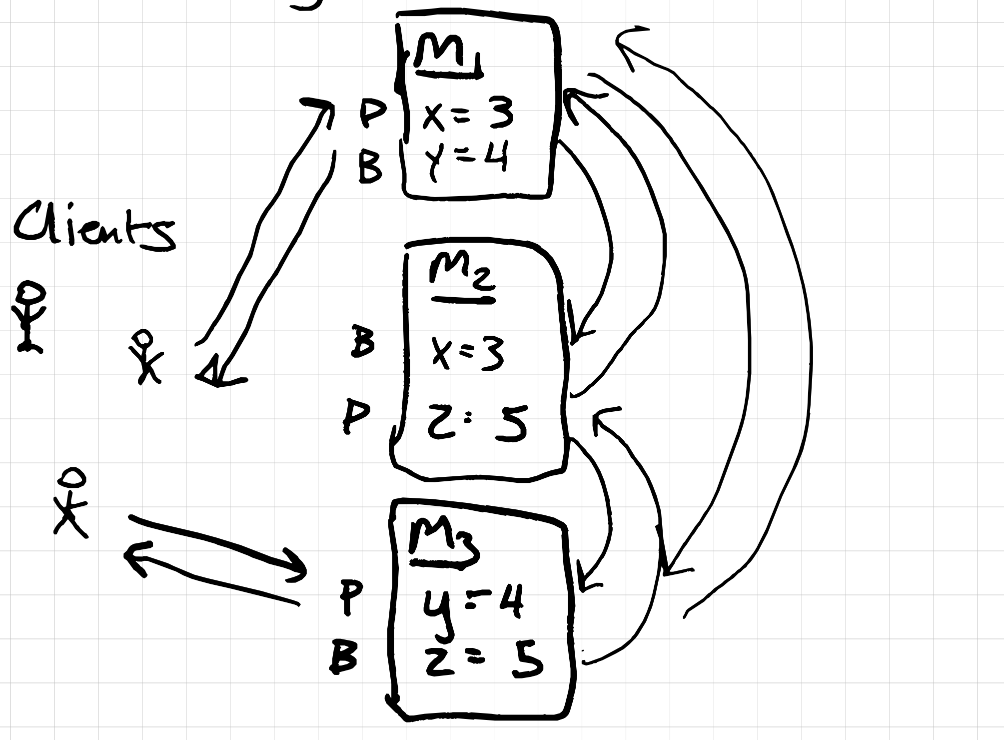


Shards (different data on each) are rows

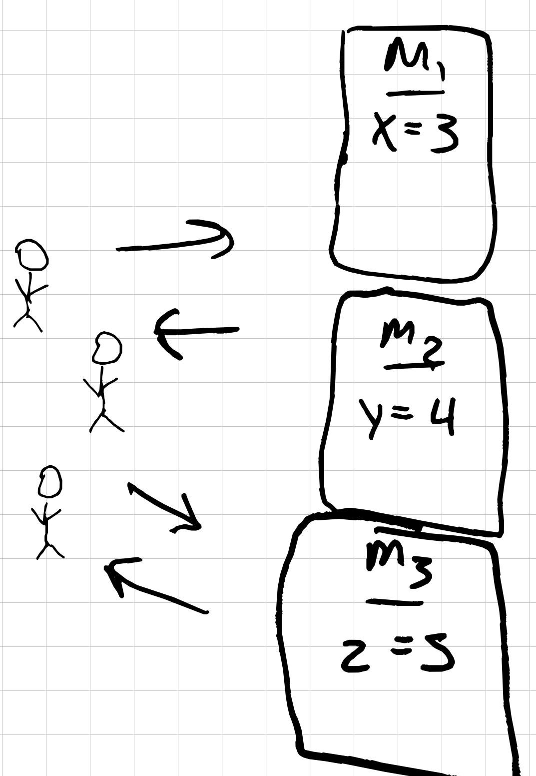
replicas (same data on each) are columns.

Reasons to do sharding:

- ⁎ – store more data than can fit on one machine
- ⁎ – improving throughput



Focus on partitions:



Goals for our partitioning strategy:

- Evenly spread data across the nodes.
- Make it fast and easy to find data we want!

Partitioning by key range:

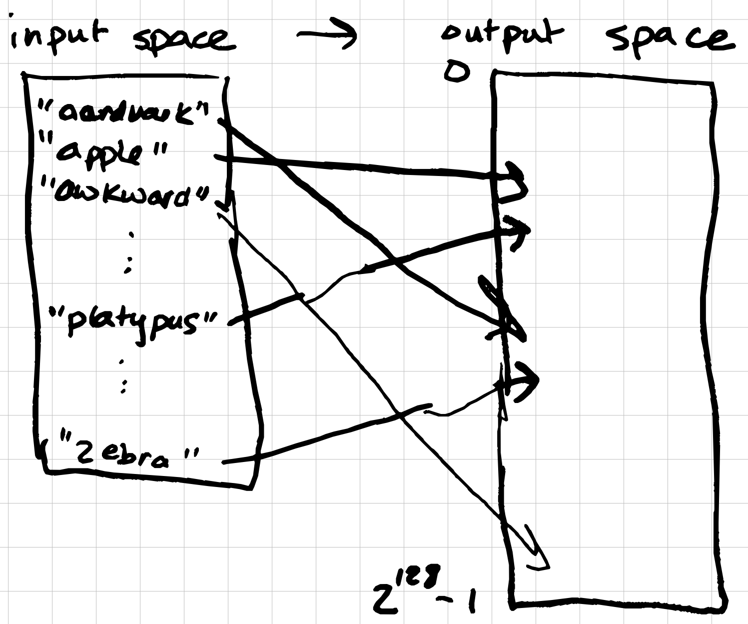
M_1	M_2	M_3
keys a–h	i–r	s–z
b=3	p=5	y=4

{ s = 10
z = 11
x = 12

Probably bad unless you know that your keys are uniformly distributed.

But there's hope... hashing!

MD5:



hash(key) mod N,

where N is the number of Nodes you're partioning onto.

"aardvark" = $\boxed{\text{data blob}}$

hash("aardvark") mod N = 2

This lets us split up the data more or less evenly.

But there's a catch...
what if the number of nodes (N) changes?

Data that doesn't need to move will get moved around! ☹

hash("aardvark") mod 3 = 2
hash("aardvark") mod 4 = 0

0 — 1 — 2 — 3

aardvark=val ← aardvark=val

e.g.,
Some data will move from an old machine to a different old machine when we add a new machine.
Seems bad.

M_1	M_2	M_3	M_4
"apple"	"awkward"	"lagomorph"	
"platypus"	"zebra"	~~"pikachu"~~ →	"pikachu"

$\frac{6}{4}$ or 1.5 keys per node. 1 should move. (or 2)

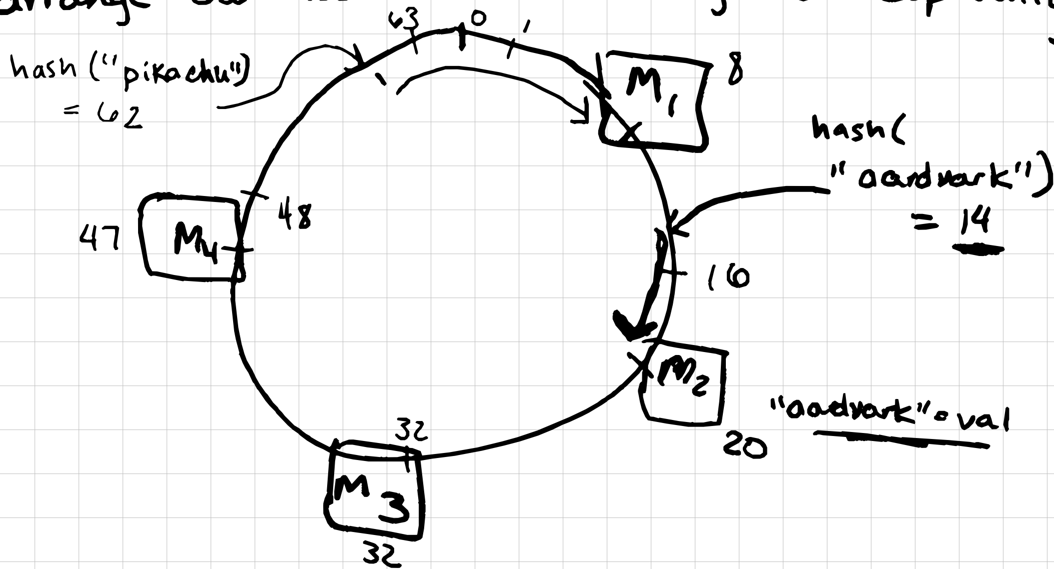
in general, if you have
K keys and N nodes,

$\frac{K}{N}$ is the number of keys that should move when a node joins or leaves.

To accomplish this,
we use $\boxed{\text{consistent}}$ hashing! ← invented for CDNs in the 90's (1997 paper) Akamai

↑ is totally different from consistency models e.g. causal consistency; totally different from consistent snapshots.

How does consistent hashing work?
arrange our nodes in a "ring" conceptually.



output space of our hash function
goes from 0 to 63 in this example.

both the keys and the node names get hashed,
and nodes get located at a point
on the ring.