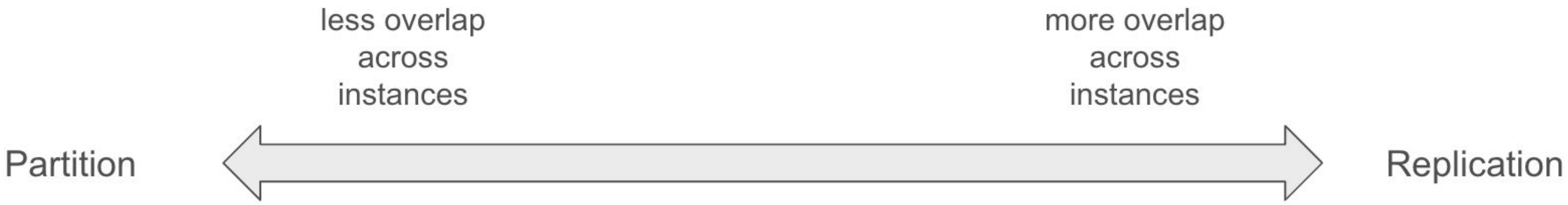
NoSQL Databases

Spring 2025

Last time…

Distributed databases:

- fault tolerance — serving data even if database instances fail
- localization — keeping relevant data geographically close to users
- scalability — serve more data to more users



Partition

less overlap across instances

more overlap across instances

Replication

CAP Theorem

A distributed system can only guarantee at most 2 of the following properties at the same time:

Consistency	all nodes have the same data
Availability	every node can respond to any request
Partition tolerance	works even is messages are slow or dropped

P is pretty much required, because internet

So you can either guarantee C or A — but not both

Relational databases focus on CP — what about AP?

Today…

Introduction to NoSQL databases

- focus on availability over consistency
- because of that, much simpler structurally

History

Early 2000s — Web 2.0

- move from informational sites to social sites
- user generated content, participatory culture
- posts, likes, shares
- important to get right, but not *critical*
- database can be inconsistent for a period of time
- availability and high-throughput is more important
- easy scalability — add a new instance quickly if load spikes

BASE properties

ACID properties for relational database enforce transactions and consistency

NoSQL databases focus instead of the BASE properties

Basically **A**vailable nodes are available, but not necessarily all data in them
Soft State data read from any node may not be the latest
Eventual consistency all nodes eventually hold the same data

No joins, no transactions

- any complexity is pushed to the apps using the database

Less database than data store — persistent distributed data structure

- one reason why devs like them so much?

Key-Value Stores

Dynamo, Redis

Distributed dictionary / hash map

f : key → value

Operations:

- get value associated with a key
- put (associate) a value with a key

Goes all the way back to Thompson's dbm on Unix!

Key-Value Stores

Often distributed by sharding

- recall that sharding is a horizontal fragmentation mechanism

Values stored in different instances (nodes) of a KV store based on key

Naive hashing to find which node to put the value into

- h_{naive} : key → {1, 2, …, K}

Adding or removing a node requires creating a new hash function

- scan and re-hash all objects!

Consistent Hashing

A clever way to hash that supports adding / removing nodes

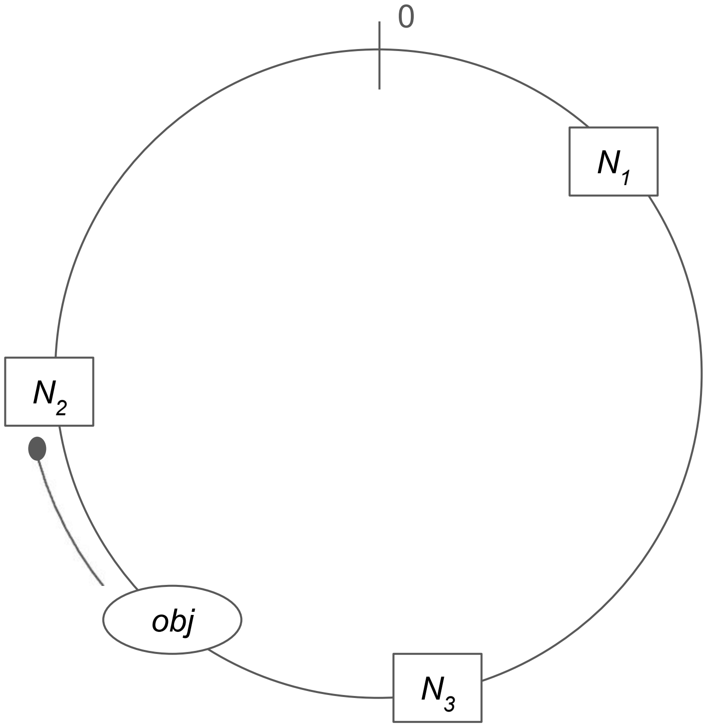
- without needing to rehash *everything*
- used for peer-to-peer networks, content delivery networks (Akamai)
- also known as a distributed hash tables

Intuition:

Do not hash into the discrete set {1, …, k}

Hash into the interval [0, 1] !

Consistent Hashing



Use two hash functions:

Hash function h_1 for nodes

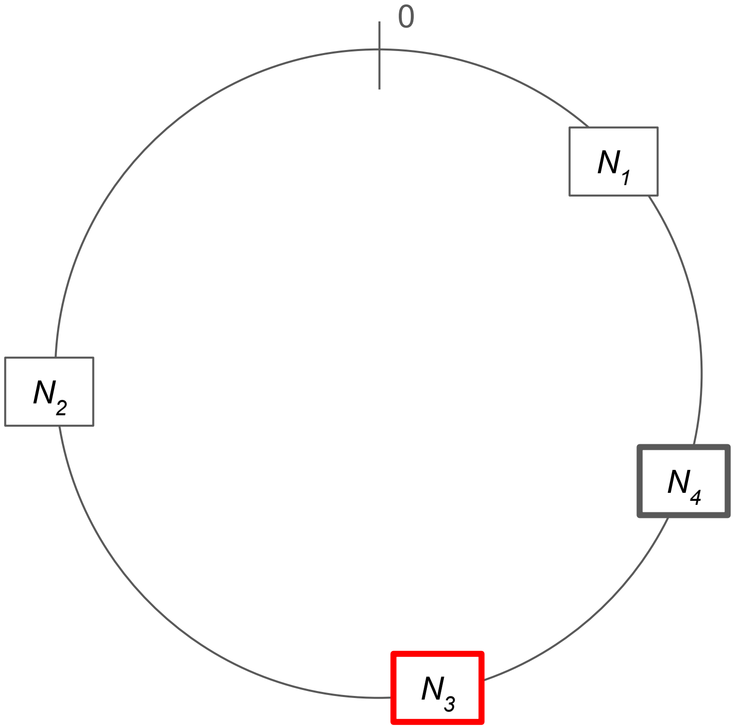
$h_1(N) \in [0, 1]$

Hash function h_2 for objects

$h_2(obj) \in [0, 1]$

obj goes into node *N* where $h_1(N)$ is first clockwise from $h_2(obj)$

Consistent Hashing



Adding a new node (N_4):

Only need to rehash the items in the node after the new node

Removing a node:

Only need to rehash the items in the node to be removed

Wide Column Stores

BigTable, Cassandra, HBase

Basically a two-dimensional key-value store

$f: (key_1, key_2) \rightarrow value$

- each entry has a primary key (for sharding)
- each entry has a secondary key

Can think of the primary key as a "row" name, secondary key as a "column" name

- main operation: get value in row/column cell
- different rows may have different "columns" filled

Wide Column Stores

	A	B	C	D	E	F	G	H	I
1	v_1						v_{10}	v_{12}	
2	v_2	v_4					v_{11}		v_{13}
3			v_6		v_8	v_9			
4	v_3	v_5		v_7					v_{14}

$f(1, A) = v_2$
$f(3, E) = v_8$
…

Can also retrieve by row (expensive)

$f(4) = \{A: v_3, B: v_5, D: v_7, I: v_{14}\}$

Wide Column Stores

Column families
(can be on their own node)

	A	B	C	D	E	F	G	H	I
1	v_1						v_{10}	v_{12}	
2	v_2	v_4					v_{11}		v_{13}
3			v_6		v_8	v_9			
4	v_3	v_5		v_7					v_{14}

$f(1, A) = v_2$
$f(3, E) = v_8$
…

Can also retrieve by row (expensive)

$f(4) = \{A: v_3, B: v_5, D: v_7, I: v_{14}\}$

Document Stores

MongoDB, CouchDB

A key-value store where values are not opaque, but instead structured (usually JSON objects)

f : key → JSON object

- objects often collected together into named collections
- but no constraints on objects in a collection

Query language:

- retrieve all objects in a collection (broadcast to all nodes!)
- retrieve all objects in a collection with a given shape

That's all, folks!