Databases at Scale

Paul Talaga

Scalability Rules - Abbott, Fisher

Web Database Applications - Williams, Lane

Wikipedia

UNIVERSITY OF Cincinnati

Database Background

Relational databases popular: MySQL, MS SQL, Oracle, etc

- Rows
- Column
- Table - Set of tuples (rows) sharing same attributes (columns)
- Use *primary* or *foreign* keys to ‘link’ rows across tables.

UNIVERSITY OF Cincinnati

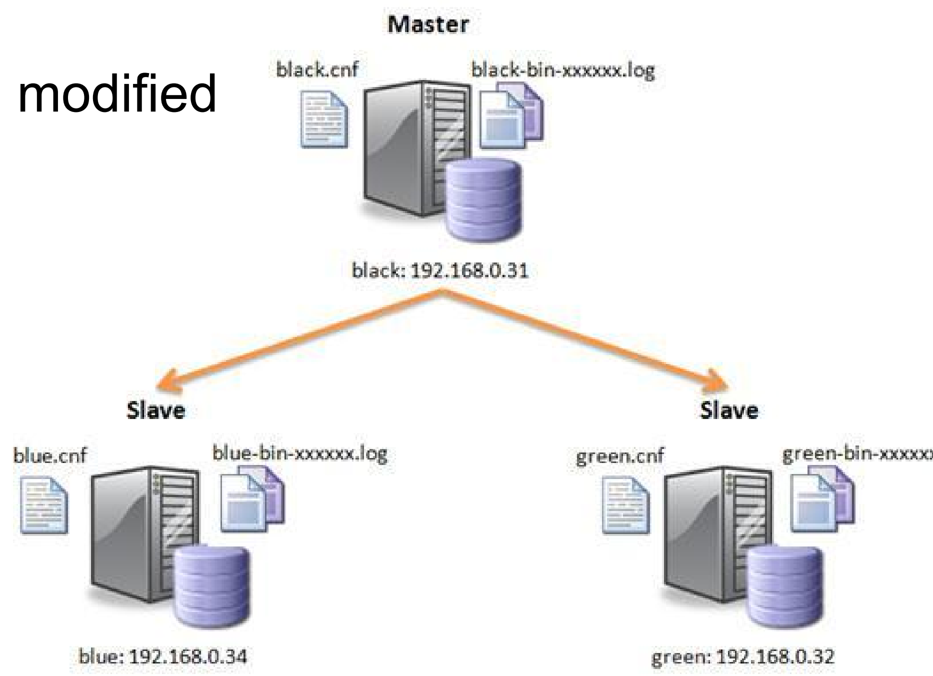
Size Constraints?

- How many rows can I add to a table?
- Methods of making larger tables?
  - Bigger machine (forklift upgrade)
  - Buy more expensive software - Oracle etc
  - Sharding - Break the table up into smaller tables, possibly on different machines
    - Pro: Puts off forklift upgrade
    - Con: App must be changed
      - Any joins must be repeated and manually joined in app
      - Not scalable

UNIVERSITY OF Cincinnati

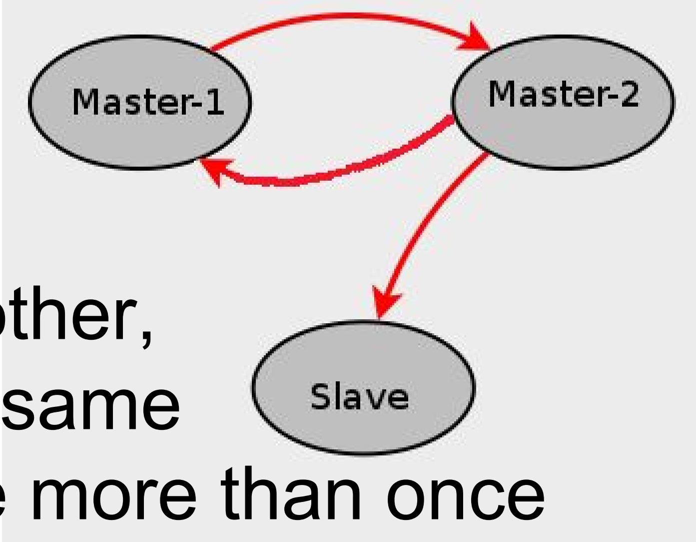
Load Constraints

- Write or Read load too high?
  - Bigger machine (forklift upgrade)
  - Replication
    - Master - Slave(s) - Streams log file to slaves
      - Pro: Increases read capacity
        - Scales reads
      - Con: Doesn't increase write capacity
        - Slaves lag
        - App must be modified



UNIVERSITY OF Cincinnati

Load Constraints



- Replication
  - Master - Master: Each streams log file to each other, marked by originator, so same DB doesn't apply change more than once
  - Can be in ring formation
  - Pros: Increases write and read capacity, though not linearly because all changes must be applied to all DB's
  - Cons: Consistency can be an issue
    - More complex - harder to maintain

UNIVERSITY OF Cincinnati

Why can't we scale? Some background...

- ACID properties - assures correct DB operations
  - **A**tomicity
  - **C**onsistency
  - **I**solation
  - **D**urability
- Applies to a transaction - diff granularity
  - Course (`BEGIN -> COMMIT/ROLLBACK`)
  - Fine - by command (MySQL < v 5.5)

UNIVERSITY OF Cincinnati

ACID Properties

- Atomicity - “All or nothing”
  - Either all the commands succeeded, or none did.
  - Resilient to power failures, disk errors, etc
  - DB correctly reflects the application/ or not
- Consistency - DB stays in valid state
  - DB implementation doesn’t corrupt DB
- Isolation - To a transaction, the DB doesn’t change. All transactions are like they were serialized.

ACID Properties

- Durability - Once transaction completed, DB shows the change, even with power loss or disk errors. MUST be stored in non-volatile memory.

UNIVERSITY OF Cincinnati

ACID Failure Examples

- Atomicity
  - Fund transfer: -10 from A, +10 to B.
- Consistency
  - Class scheduling - places 2 classes in the same room at the same time (or slightly off)
- Isolation
  - Simultaneous fund transfer: $10 A->B and $10 from B->A
    - Sub 10 from A
    - Add 10 to B
    - Sub 10 from B
    - Add 10 to A

- Sub 10 from A
- Sub 10 from B
- Add 10 to A
- Add 10 to B

UNIVERSITY OF Cincinnati

ACID Failure Examples

- Durability
  - I issue money transfer from my account to another (ex: rent) and get confirmation
  - Receiver claims money never sent

- Gaining ACID compliance isn’t that hard, but doing it quickly and in parallel is.
- Distributed?!?! Require distributed transactions (Hard and slow)

Is ACID always needed?

- Yes, in some situations:
  - Finance software
  - Asset Management (UPS, Airlines, Scheduling)
  - ….. anywhere the data MUST be right
- No, in others:
  - My FB post is before yours for everyone
  - Perfect # logged in users
  - Perfect # of products on a shelf >100
  - Regenerable data

UNIVERSITY OF Cincinnati

Other Background..

CAP Theorem

(Eric Brewer in 2000)

A distributed system can not satisfy:

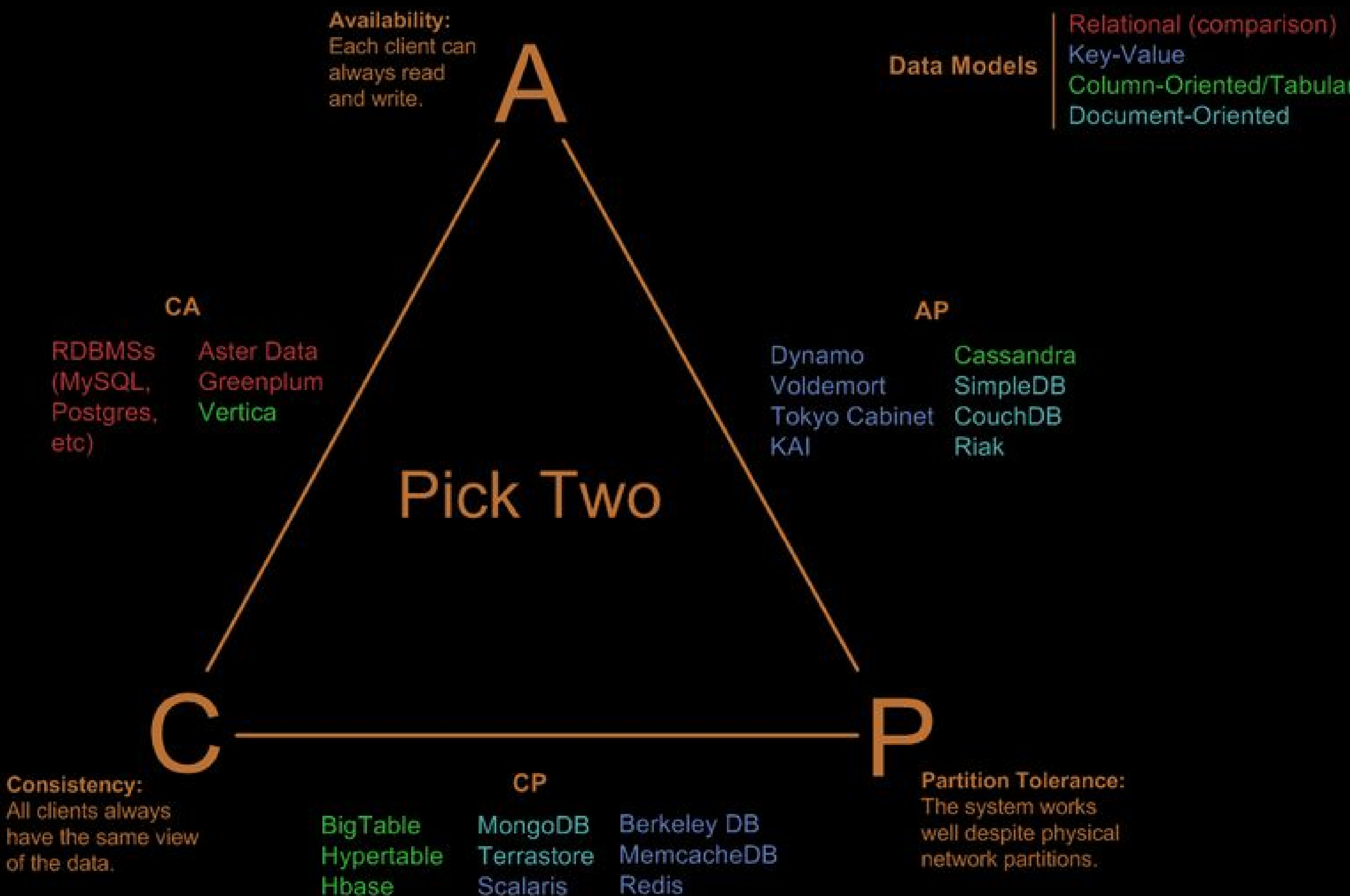
- Consistency - all nodes see same data
- Availability - All requests acked
- Partition tolerance - OK with message loss or network partitioning

You'll see many (distributed) databases claim to prioritize these guarantees.

UNIVERSITY OF Cincinnati

Visual Guide to NoSQL Systems

http://blog.nahurst.com/visual-guide-to-nosql-systems



The Birth of NoSQL

- Not technically No-SQL, but a relaxation of ACID for performance gains
  - No-RELational possibly a better term
  - Typically they do not have an SQL-like interface (but some do)
- Taxonomy (wiki)
  - Column Stores
  - Document Stores
  - Key-value Stores
  - Graph Stores

NoSQL Strengths/Weaknesses

by Ben Scofield

Data Model	Performance	Scalability	Flexibility	Complexity	Functionality
Column	High	High	Moderate	Low	minimal
Document	High	variable (high)	High	Low	variable (low)
Key-value	High	High	High	Low	variable (none)
Graph	Variable	Variable	High	High	Graph Theory
Relational	Variable	Variable	Low	Moderate	Relational Alg.

UNIVERSITY OF Cincinnati

Column Stores

- Similar schema to relational databases
- Tables w/ rows - all same columns
- Primary keys allowed
- Joins NOT possible
  - Must denormalize data
- Interaction is similar to SQL
- Ex: Cassandra, BigTable, HBase

UNIVERSITY OF Cincinnati

Cassandra Example (link)

CQL - Cassandra Query Language

```
CREATE TABLE songs (
  id uuid PRIMARY KEY,
  title text,
  album text,
  artist text,
  data blob
 );
```

```
CREATE TABLE playlists (
  id uuid,
  song_order int,
  song_id uuid,
  title text,
  album text,
  artist text,
  PRIMARY KEY  (id,
song_order ) );
```

```
SELECT * FROM playlists;
```

id	song_order	album	artist	song_id	title
62c36092...	1	Tres Hombres	ZZ Top	a3e64f8f...	La Grange
62c36092...	2	We Must Obey	Fu Manchu	8a172618...	Moving in Stereo
62c36092...	3	Roll Away	Back Door Slam	2b09185b...	Outside Woman Blues

```
CREATE INDEX ON playlists(artist );

SELECT * FROM playlists WHERE artist = 'Fu Manchu';
```

id	song_order	album	artist	song_id	title
62c36092...	2	We Must Obey	Fu Manchu	8a172618...	Moving in Stereo
62c36092...	4	No One Rides for Free	Fu Manchu	7db1a490...	Ojo Rojo

UNIVERSITY OF Cincinnati

Cassandra Features

- Compound keys & clustering - determines where row is stored
- Collection Columns - allow `set, list, map` structures to row
- Decentralized architecture
- Asynchronous replication
- **Linear scaling**
- Tunable consistency
- Very popular: Apple (75k nodes 10+PB), Netflix, Ebay, Twitter, Reddit, Digg

UNIVERSITY OF Cincinnati

Document Stores

- Lack consistent columns
- A document is like a row, but can differ
- Document is encoded: XML, JSON, etc..
- Can add ‘columns’ at any time

```
{
    FirstName: "Bob",
    Address: "5 Oak St.",
    Hobby: "sailing"
 }
```

```
{
   FirstName: "Jonathan",
   Address: "15 Wanamassa Point Road",
   Children: [
        {Name: "Michael", Age: 10},
        {Name: "Jennifer", Age: 8},
        {Name: "Samantha", Age: 5},
        {Name: "Elena", Age: 2}
   ]
}
```

UNIVERSITY OF Cincinnati

Document Stores

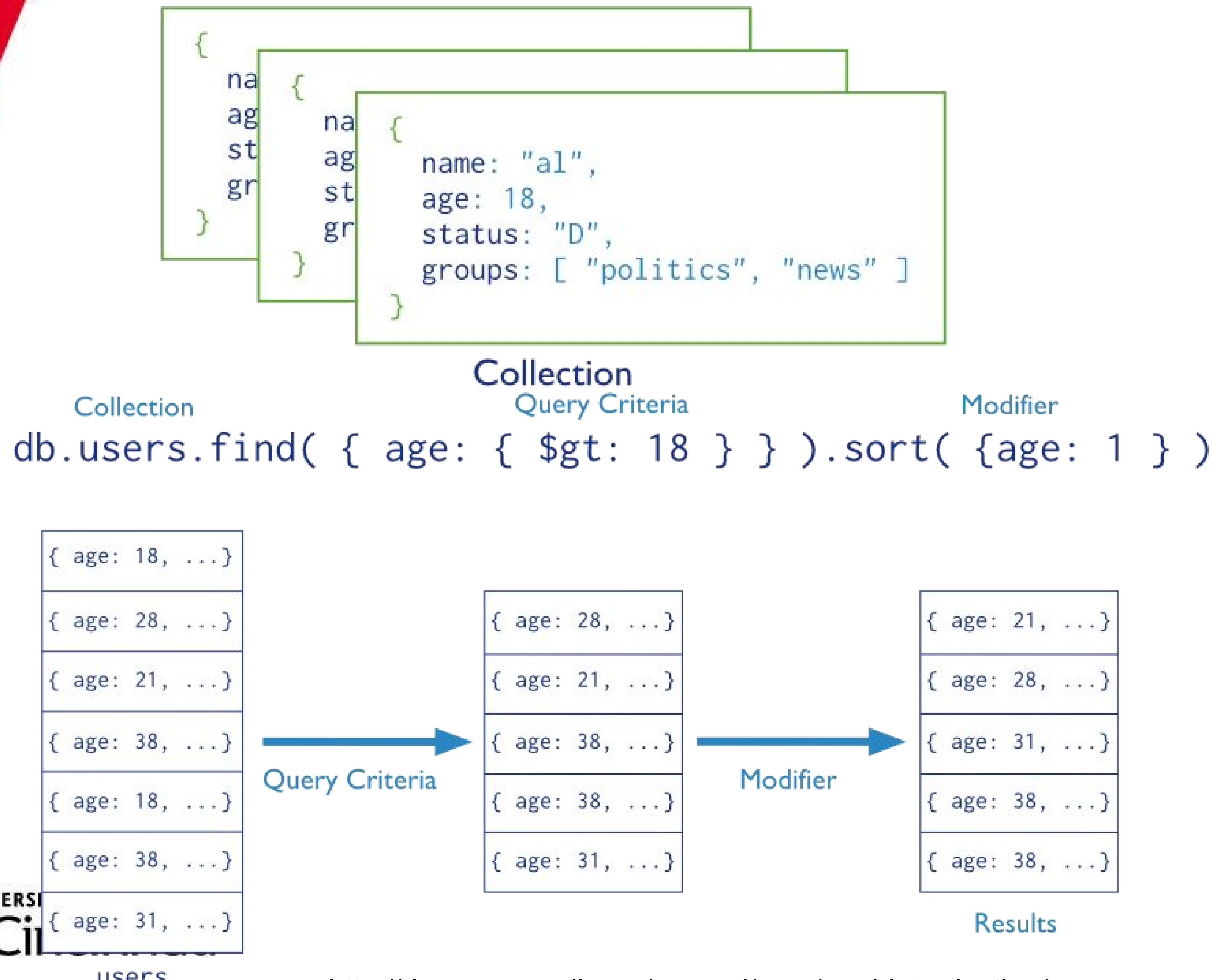
- Contain a key per document (like key/val)
- Query method can vary:
  - Can query by key….or
  - Query function applied to documents
- Good for situations where you may be adding unforeseen properties to docs
- Query performance not great
- Easy mapping from OO
- Ex: MongoDB, CouchDB

UNIVERSITY OF Cincinnati

MongoDB

- Stores BSON (Binary JSON)
- Normalized data allowed
- Atomic writes at the document level
- Sharding possible with shard keys
- Indexing possible on common fields
- TTL possible
- Users: Craigslist, eBay, SourceForge, New York Times, Cisco

MongoDB Example



http://docs.mongodb.org/manual/core/crud-introduction/

Key-Value Stores

- AKA associative array, map, or dictionary
- Set ‘Value’ by ‘Key’, Get by ‘Key’, Delete ‘Key’
- Typically ANY data can be key or value
- Different types:
  - Eventually consistent: Cassandra, Dynamo, Voldemort, Riak
  - Hierarchical: GT.M
  - RAM: Memcache(d), Redis
  - Persistent: MemcacheDB, Tokyo Cabinet
  - Ordered: Berkeley DB, MemcacheDB, SimpleDB

UNIVERSITY OF Cincinnati

Key-Value Stores

- Very simple interface (ex: memcache)
  - `get <key>`
  - `set <key>, add <key>, replace <key> <val>`
  - `append <key> <val>, prepend <key> <val>`
  - `incr <key> (<val>), decr <key> (<val>)`
  - `delete <key>`
  - `flush` - removes all items
- Supports expiration: timestamp or seconds
- Easy to distribute using DHT

UNIVERSITY OF Cincinnati

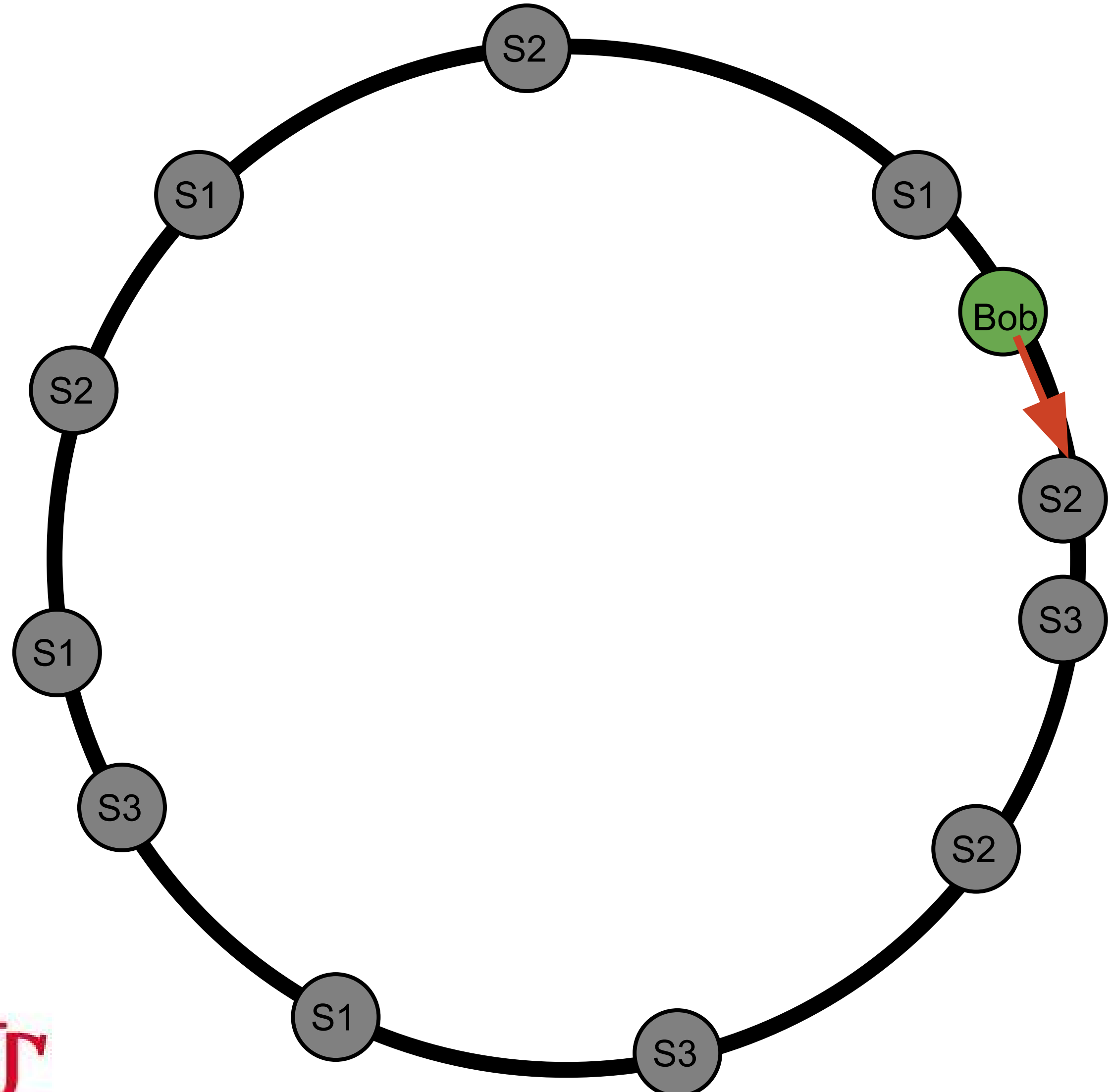
Distributed Hash Tables

- How to distribute key/value data to many servers?
  - Use hashed key as server address
    - Fast! O(1)
    - Simple
    - Problem with additions/removal of servers

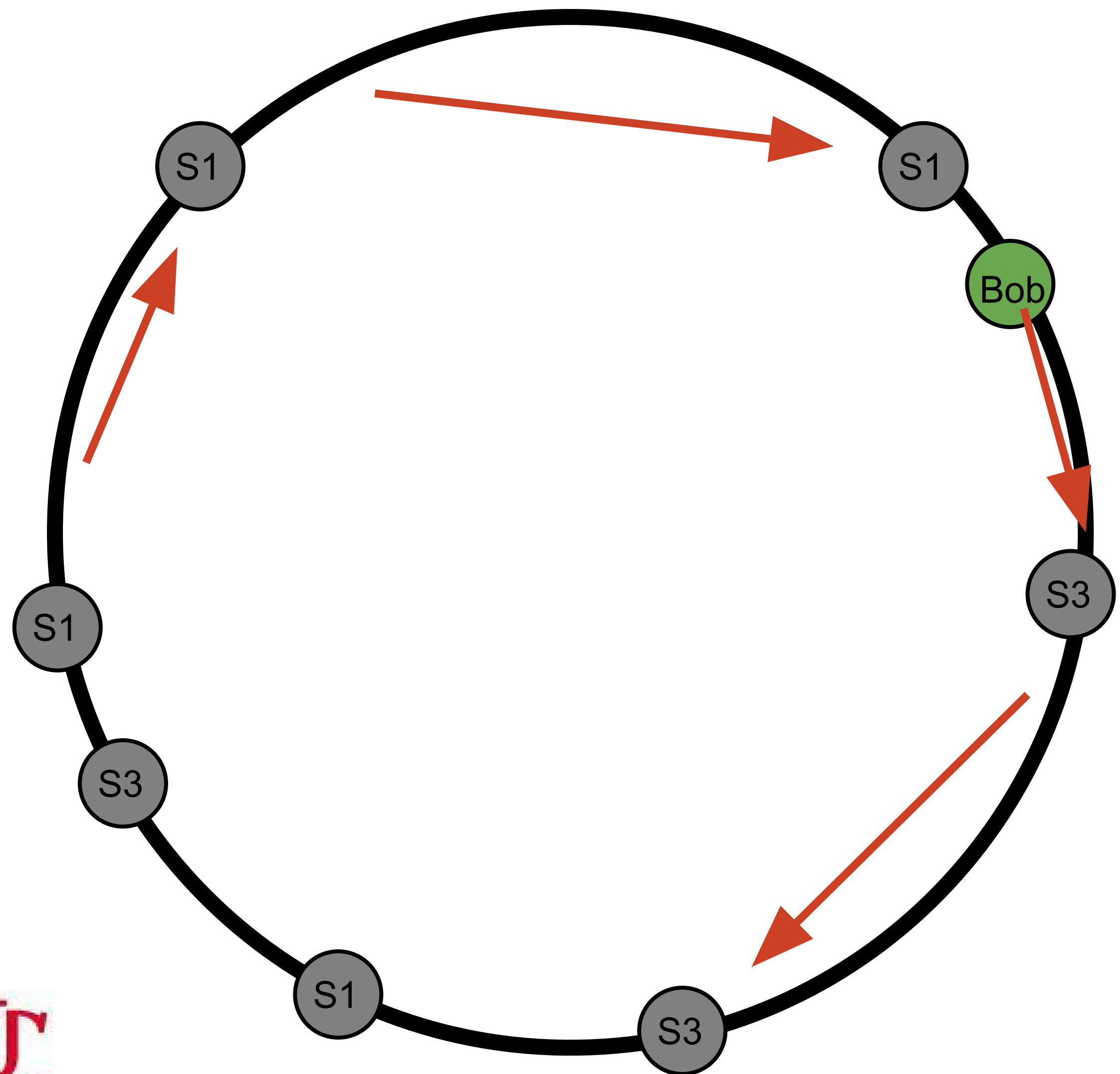
bob: stuff -> md5(bob) % 4 = Server 3
bird: table -> md5(bird) % 4 = Server 1
…
Use result of mod for server #

UNIVERSITY OF Cincinnati

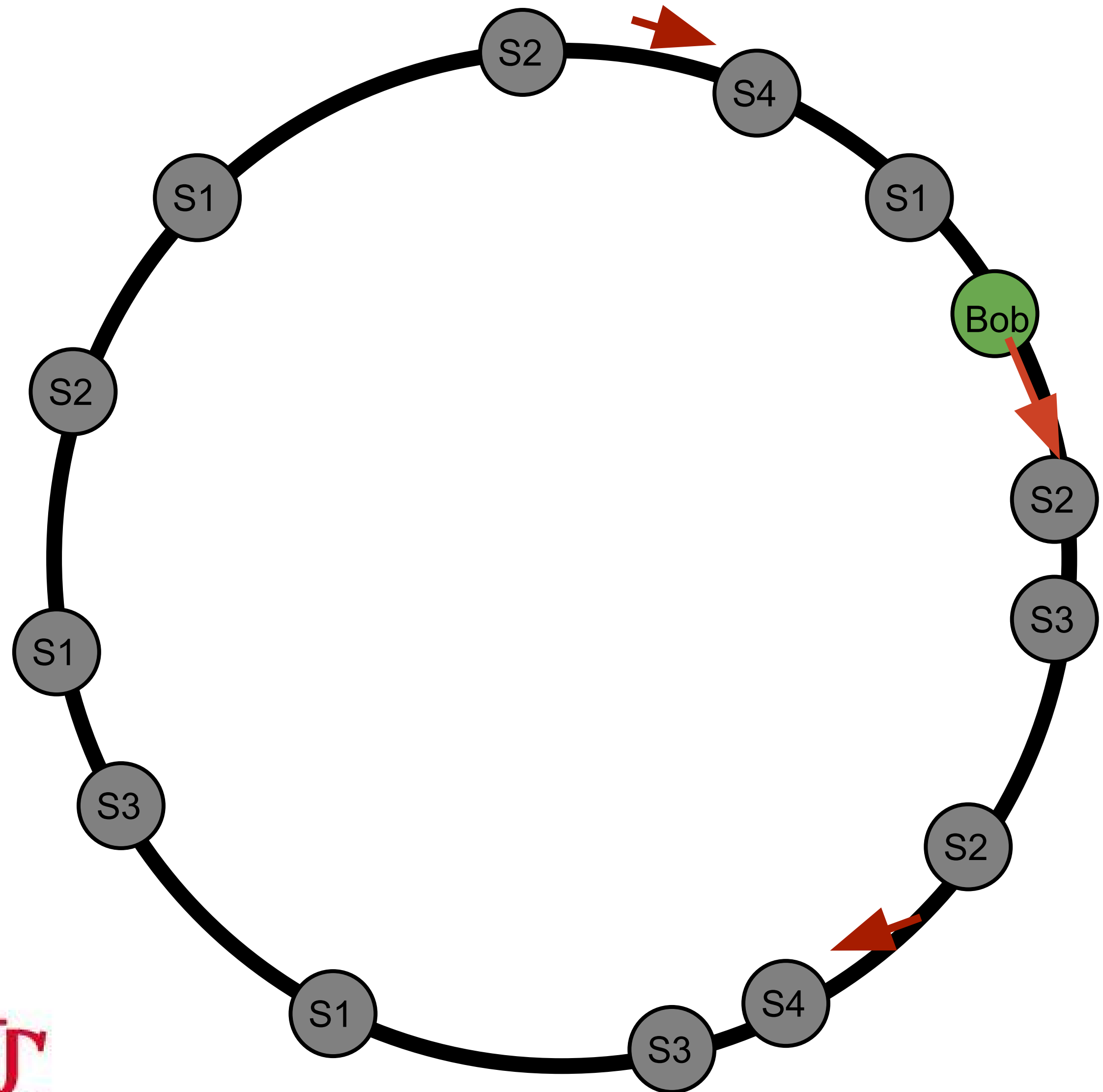
Consistent Hashing



Consistent Hashing - S2 Down



Consistent Hashing - Add S4



UNIVERSITY OF Cincinnati

Consistent Hashing

- Still fast O(1)
- Reduces data moves when adding/removing servers
- Allows different loading/size of servers
  - Allocate smaller number spots
- Redundancy can be built in - go to next 2 or 3

UNIVERSITY OF Cincinnati

SimpleDB

- Amazon’s key-value storage system
- Dr. Helmick on development team!
- Written in Erlang
- Tunable consistency (eventual or consistent)
- ‘Domain’ is like table or collection
- Key/value with attributes
- Searching possible - SQL-like (MySQL under the hood)
- BOTO API

UNIVERSITY OF Cincinnati

SimpleDB Newer Features

- Consistent Read - be sure to reflect operations just done (from this machine)
- Conditional Put - only update if the old values were <value> - warning and fail otherwise
- Conditional Delete - only delete if values match

UNIVERSITY OF Cincinnati

AWS - DynamoDB

- Replacement to SimpleDB
- Fully managed - SSD backed
- Tunable performance
  - Read Cap - 1 consistent read/sec or 2 non 4KB
  - Write Cap - 1 write/sec 1KB
- Atomic counters (like memcache)
- 64KB max value per item
- Integrates with AWS (backup, mapreduce)
- Free Tier: 100MB 5 w/s, 10 r/s
  - Regular: $0.0065 10 units write (above)
  - Regular: $0.0065 50 units read (above)
- Storage: 100 MB Free, $0.25 GB/month after

Dynamo DB - Continued

- DB, Tables, Attributes, Keys
- YOU specify what is indexed (everything in SimpleDB)
- Not as easy to change schema when running
- Provides conditional writes & atomic counters