CS 686: Special Topics in Big Data

DHTs and Data Models

Lecture 9

Today's Agenda

- DHT Replication Strategies
- Reducing latency, boosting performance
- Virtual Nodes
- Data Models

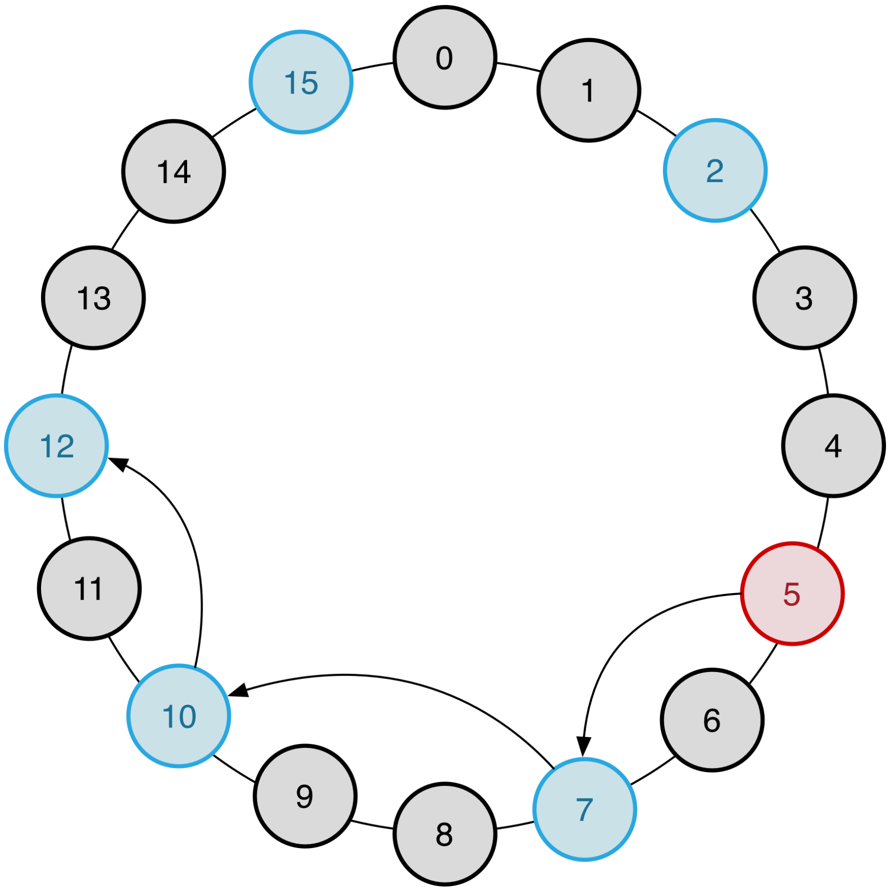
Today's Agenda

- **DHT Replication Strategies**
- Reducing latency, boosting performance
- Virtual Nodes
- Data Models

Replication

- We've seen from the HDFS paper that maintaining **3** total copies of each file is our gold standard
  - In some situations, **5** is warranted
  - …And sometimes having 0 copies is the way to go (like in the case of poetry/artwork by the instructor)
- It's always worth thinking about the cost of maintaining these, though
- How do we do replication in DHTs?

Replicate to Successors



- Send a copy to R **successors**
- If Node 5 goes down, Node 7 will take its load
  - Great! Promote replica to primary file
- Doesn't account for query traffic, physical locations, etc.

Query Paths

- Rather than replicating immediately to a certain set of nodes, wait for queries to come in
- Cache the replicas at nodes that forwarded the query
  - Reduces the latency of frequent queries that originate at the same node
  - Let's say my client always contacts the node in San Francisco, which then retrieves from a node in Texas
    - Store a replica in SF
- Better for query performance, not absolute safety

Salting

- For each file, add a **salt**
  - Random data used as an additional input to the hash function
  - SALT_REPLICA1 = "Hi there!"
  - SALT_REPLICA2 = "What what what"
- put(key + SALT_REPLICA1, value)
- Now we can deterministically locate the replicas associated with a key

Today’s Agenda

- DHT Replication Strategies
- **Reducing latency, boosting performance**
- Virtual Nodes
- Data Models

Zero-Hop DHTs (1/2)

- When nodes enter and leave the network in a controlled fashion, **zero-hop** DHTs may be a good fit
- O(1) routing hops rather than O(log n)
- Every node must maintain an entire copy of the routing table
  - Synchronous updates are not required
  - If an old route is used, just forward the request to the correct node
  - Node down? Try the predecessor

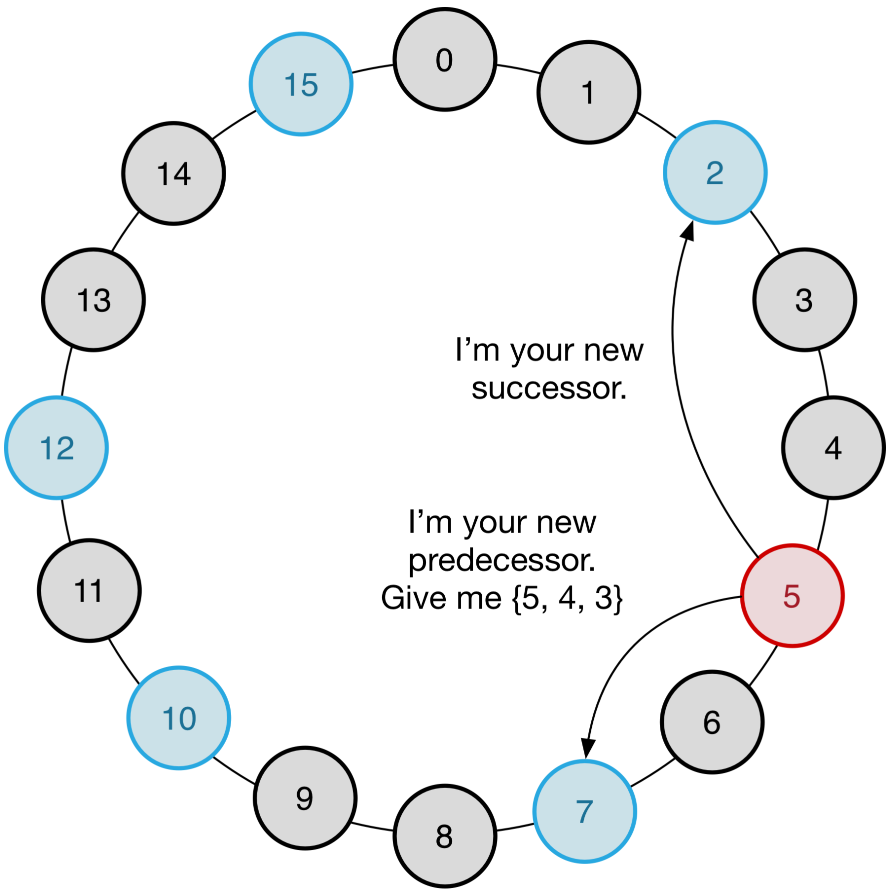
Zero-Hop DHTs (2/2)

- Zero-Hop DHTs are a great example of finding a compromise in the middle
- Retain many good aspects of regular DHTs, but are also easier to implement
  - May sacrifice some scalability, but in general they target a different use case
- Some implementations: Dynamo, Cassandra, Riak
  - Dynamo: Amazon & SLAs

GlusterFS

- Unlike most of the distributed file systems we've surveyed, GlusterFS is actually mountable
  - Backed by Zero-Hop DHT
- Hashes directory ID + file ID to place/locate files
- When we use a regular file system, move operations are common
  - When the usual lookup fails, broadcast to everyone
- Supports **linkfiles**, which are essentially a symlink to redirect lookup requests to another node
  - Great for dealing with file migrations

Eventual Consistency (1/2)



- Joining or leaving the Chord network causes **inconsistency**
- In this example, it may take a bit for node **15** to learn about node **5**
- The system will eventually reach a steady state (usually in ms)

Eventual Consistency (2/2)

- Eventual consistency is a mainstay of distributed systems
- It’s easier to accept that things will be inconsistent (sometimes) rather than trying to prevent it
  - Amazon: shopping cart vs billing
- You can often achieve much better performance if you relax consistency
  - But remember to ask yourself: are your customers/clients okay with that?

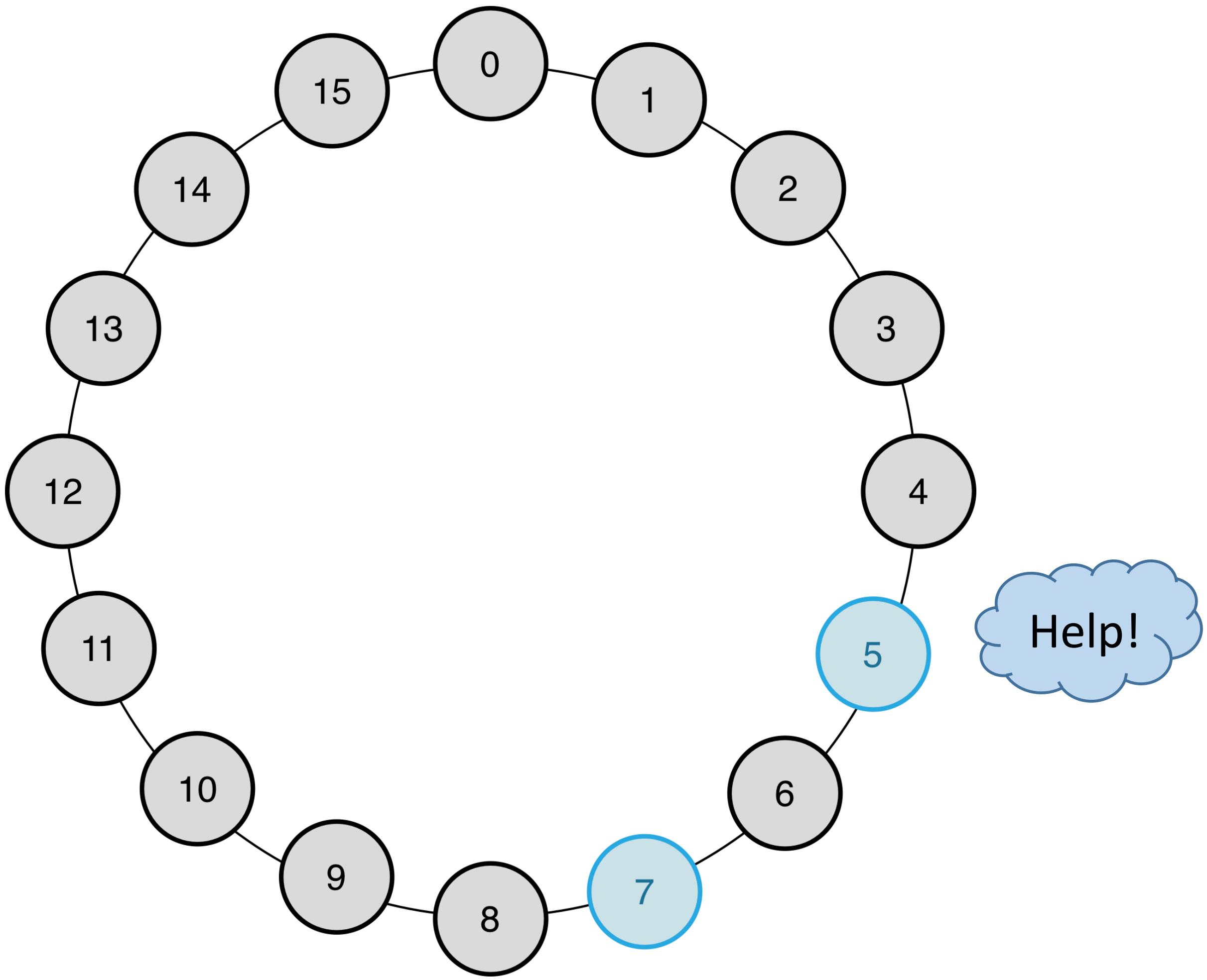
Today's Agenda

- DHT Replication Strategies
- Reducing latency, boosting performance
- **Virtual Nodes**
- Data Models

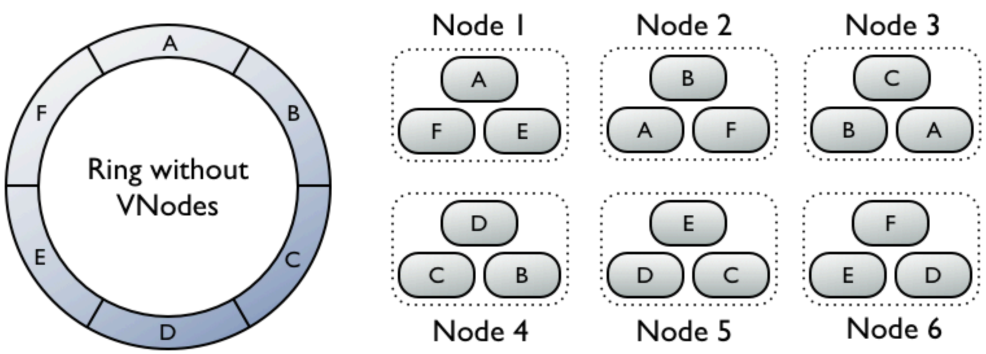
Avoiding Hotspots

- Our cluster may be heterogeneous or have **hotspots** that receive a disproportionate amount of load
- To help fill in the gaps and even out the load, nodes may be required to initially represent several IDs
  - Used frequently in large deployments – hundreds of IDs are assigned to each node
  - Allows variations on the default load level: new node could take on 1.2 nodes' worth of keys

Lonely Node 5



Cassandra: VNodes

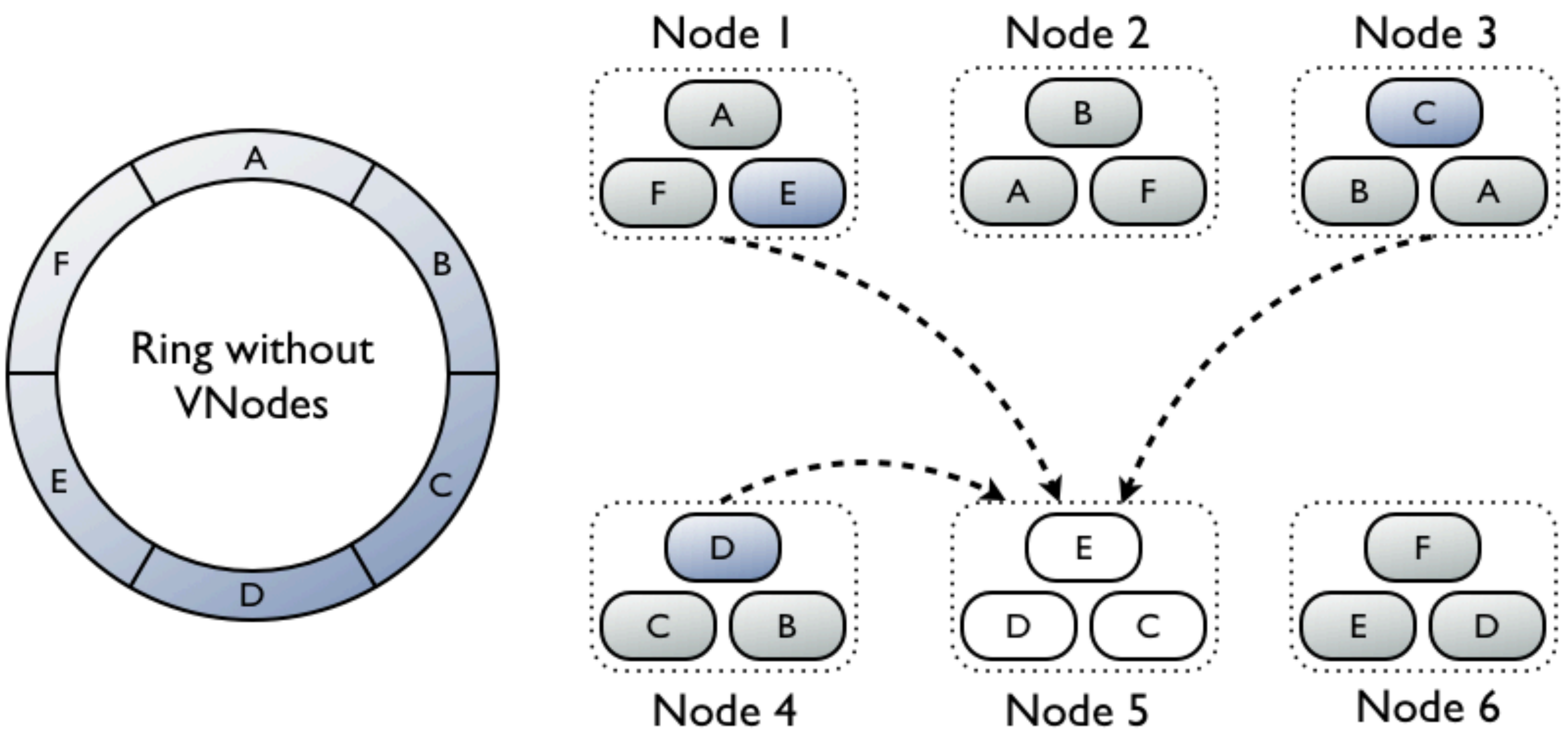


See: https://www.datastax.com/dev/blog/virtual-nodes-in-cassandra-1-2

VNodes

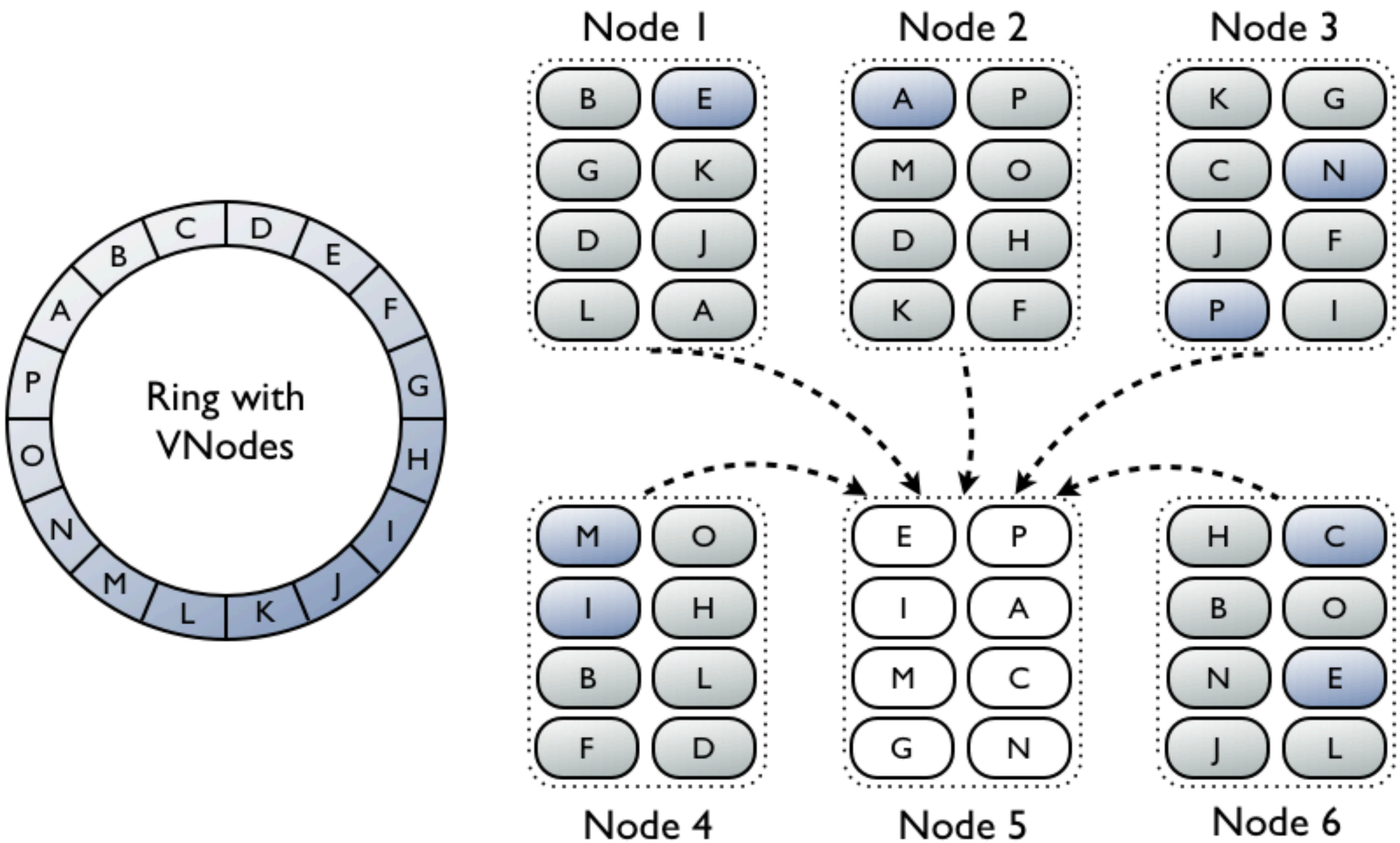
- With virtual nodes, each physical host is responsible for many more portions of the overall hash space
- Common approach: randomize the vnode locations
- More coverage means less of a chance that one node gets stuck with too much load
- But wait, wasn't localizing network changes one of the **pros** of using DHTs?
  - Yes. But more coverage ***can*** be a good thing too.

Replacing Node 5



See: https://www.datastax.com/dev/blog/virtual-nodes-in-cassandra-1-2

Replacing Node 5, With VNodes



See: https://www.datastax.com/dev/blog/virtual-nodes-in-cassandra-1-2

VNodes: Pros and Cons

- VNode pros:
  - Better load balancing properties
  - Better parallelism when recovering
- VNode cons:
  - Less localized faults: loss of a single node is dispersed across the hash space
  - Many more nodes participating in recovery means less resources for answering queries

Today's Agenda

- DHT Replication Strategies
- Reducing latency, boosting performance
- Virtual Nodes
- **Data Models**

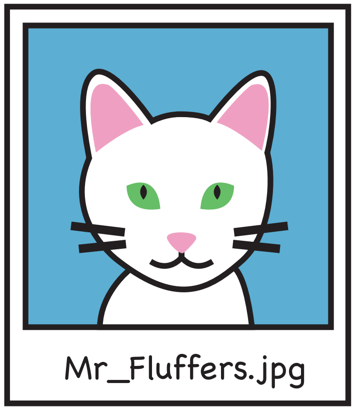
Data Models

- Key-value
- Document
- Wide Column
- Graph
- Tabular

Key-Value Stores

- Data model similar to a hash table
  - Flat namespace
- Simple functionality
  - Put(key, value)
  - Get(key)
  - No query/search support
- Frequent uses:
  - General (file) storage
  - Object caches

Key-Value Data Model



Content-Addressable Storage

- In some cases you don't want to store data with a file name or identifier
- With CAS, just use the content's hash key directly
  - put(my_file.txt) → 0x123456789
- Use cases:
  - Preventing duplicate data from being stored
  - Verifying the integrity of documents
  - Pulling in file updates

Document Stores

- Beyond key-value semantics, document stores also allow ***content-aware*** searches
- Support a wide variety of data types
  - Serialization formats, multidimensional arrays
- Generally use ***inverted indexes*** to support queries
- Index options:
  - Domain-specific indexer
  - Well-defined storage format (JSON, XML)

Document Store Data Model

(Key/Identifier) → (Document)

matthew.json →

```
{
    "name": "Matthew Malensek",
    "locale": "en-US.UTF-8",
    "status": "Teaching",
    "location": "HR 148",
}
```

stark.json →

```
{
    "name": "Tony Stark",
    "alias": "Iron Man",
    "administrator": true,
    "status": "Saving the World",
}
```

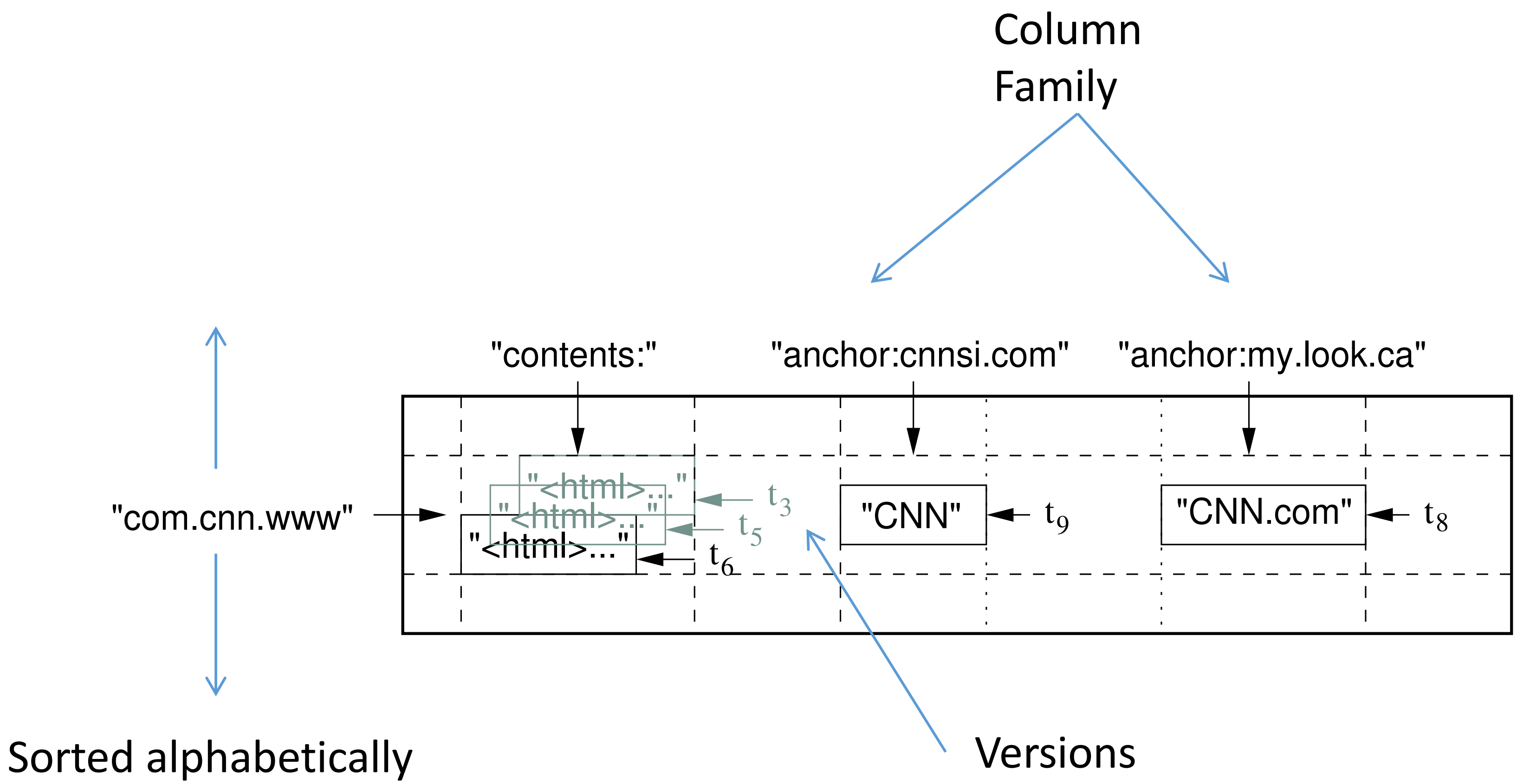
Other Document Types

- We are all familiar with JSON and XML
- Scientific document types:
  - NetCDF
    - Unidata
  - HDF5
  - GRIB
    - World Meteorological Organization
- And of course, plain text, ODF, .doc(x)

Wide-Column Stores

- Multidimensional key-value stores
  - Values are arbitrary byte arrays
- Can be sparsely populated
- A ***row key*** references a set of ***column families***
  - Writes under a row key are atomic
  - Keys are stored in lexicographic order to facilitate scanning across records
- Often include column-based ***versioning***

Wide-Column Data Model



Source: Chang et al., "Bigtable: A Distributed Storage System for Structured Data"

Graph Stores

- Relational databases provide some level of graph support: links between entities via foreign keys
  - Fairly restrictive, not performant for large graphs
- Graph stores represent data as a collection of vertices and edges
  - Models connections (relationships)
  - Can store data in both vertices, edges
  - Query via DSL or SQL
- Great for applications like Facebook friends

Tabular Stores

- Densely populated tables (relations)
- Fixed set of data types
- Schema does not frequently change
- Caveats:
  - All tables must have at least one primary key column
  - Data partitioning is specified explicitly

Tabular Data Model

Name	Address	Phone	Birth Date
Matthew	1625 W Oak St	(970) 379-4929	2/27/22
Michelle	1234 N Drury Lane	(327) 876-5309	11/16/81
Bob	1600 Pennsylvania Ave	(202) 456-1111	08/04/61

Development Timeline

- ~1970: relational databases
  - SQL, relational models
- ~2009: surge in popularity of “NoSQL” systems
  - Relaxed consistency, de-emphasizing transactions, new data models
- ~2012: “NewSQL” systems
  - Tabular data model, ACID support, but built on distributed principles

Data Models: Classifications

	Key-Value	Document	Wide-Column	Graph	Tabular
Schema	None	Ad-Hoc	Semi-Structured	Ad-hoc	Structured
Search	Key	Content	Row/Column	DSL, SQL	SQL

Wrapping Up

- These models influence both storage and retrieval
- Simple data models can allow increased automation
- Well-defined schemas provide greater query flexibility but require more configuration
- Strong consistency is most common when records are fine-grained