UNIVERSITY of WASHINGTON

BIGDATA 210: Introduction to Data Engineering

Autumn 2018

Module 7: NoSQL Databases Part I

Jerry Kuch

jkuch@uw.edu

Week 7 Agenda

- Project Discussion
- Big Data Management, remarks on:
  - Modeling, governance, quality, security
- What is NoSQL?
- Introduction to some NoSQL systems
  - Redis
  - HBase

Final Project

- Overview:
  - Self-directed project using technologies discussed in the course…
  - …on a data set of sufficiently large size.
  - Very open ended.
- Objectives:
  - Demonstrate ability to use skills from class on real data
  - Find something that interests you and dig in!
- Requirements:
  - Form groups and notify instructor of your group members
  - Choose a suitable dataset (say > 1GB, or else smaller but challenging)
  - Develop use case or set of questions you want to solve with the data
- Deliverables:
  - Source of data
  - Any code/scripts/notebooks developed
  - In class presentation (length TBD based on how many groups)

Big Data Management

INTERACTIVE HYPE CYCLE

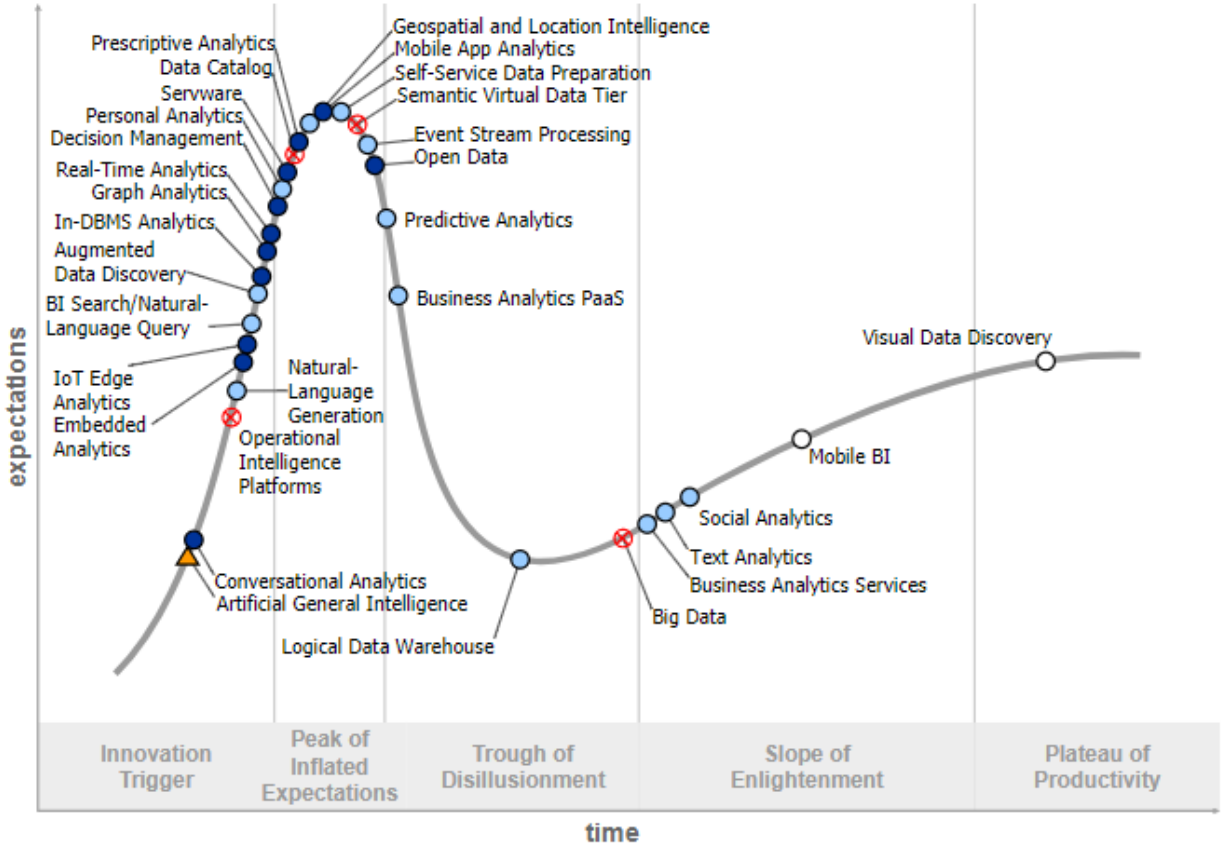
Hype Cycle for Analytics and Business Intelligence, 2017

27 July 2017 G00314848

VIEW FULL DOCUMENT

Analyst(s): *Kurt Schlegel* | *Jim Hare*

About This Hype Cycle | Glossary | Technology Priority Matrix



Big Data

Definition:

Big data is a pervasive phenomenon in information technology and IT practices. It is composed of various solution architectures, collections of tools and platforms that address radical changes in volume, velocity and variety of information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making. Big data itself is not a market or technology stack.

Recommended Reading

Identifying and Selecting the Optimal Persistent Data Store for Big Data Initiatives

Solution Path for Evolving Your Business Analytics Program

Big Data Management: Questions to Ask About Your Data

- What does the data look like?
  - How are different pieces related?
- How will we ingest the data?
- Where and how do we store it?
- How do we ensure/enforce data quality?
- What operations do we need to perform on it?
- How will we scale up to handle Big Data?
- How do we keep data secure?

Big Data Management

- Questions to Ask
  - **What does the data look like?**
    - **How are different pieces related?**
  - How will we ingest the data?
  - Where and how do we store it?
  - **How do we ensure/enforce data quality?**
  - What operations do we need to do to the data?
  - How will we scale up to handle Big Data?
  - **How do we keep data secure?**

Data Modeling: What does it mean in NoSQL?

- NoSQL != “no schema”
- Big Data != NoSQL
- Big Data CAN be modeled

Data Modeling: Characteristics of Data Models

- What is a data model?
  - Describe data *structure*
    - e.g. Person{firstName:String, lastName:String, DOB:date, SSN:String}
  - Impose/enforce *Constraints*
    - All DOB must be > 21
    - DOB must have format YYYY-MM-DD
    - SSN must be unique
  - *Relationships*
    - Person and Tax Returns are related
      - Even if field names differ
      - SSN vs SocialSecNum

Data Modeling: When to Impose Schema

- Schema on write
  - Schema defined upfront
    - Setup table
    - Map relationships from data to actual columns
    - Write sql insert statement for data
  - Benefits
    - Result speed
    - Data consistency
  - Concerns
    - How to handle changes?
    - Similar but different data?

Data Modeling: When to Impose Schema

- Schema on read (schema on demand)
  - Write data to data store
    - "Schema" applied at read time based on application
  - Benefits
    - Flexibility
  - Concerns
    - Queryability
    - Data consistency

Data Governance/Quality

- Data Governance
  - A defined process for an organization to ensure high quality as you collect, manage and archive data in order to derive value

- Data Quality
  - Metadata
  - Standardization
  - Cleansing
  - Matching
  - Enrichment

Data Governance/Quality

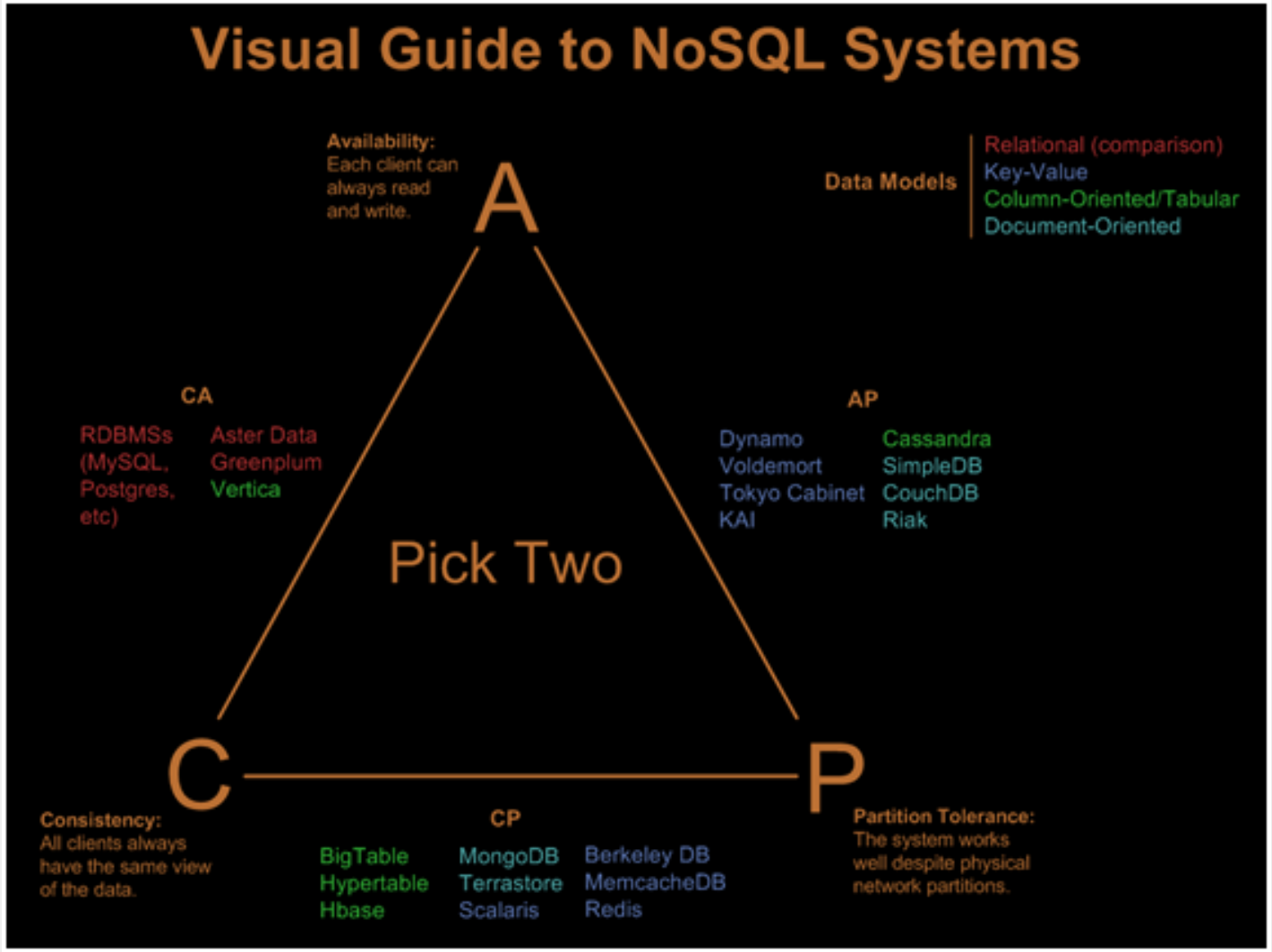
- Big Data/NoSQL Governance
  - Open source tools
    - Explicit Governance tooling
      - Ranger, Atlas, Falcon
    - Frameworks that provide governance
      - NiFi
  - Vendor options
    - Legacy Data Warehouses
      - Teradata, Informatica, etc…
    - Big Data-centric
      - Cloudera, Numerous startups….

Data Security

- Historically limited security options
- Hadoop ecosystem security layers
  - Perimeter Security
    - Firewalls, Apache Knox (proxy)
  - Authentication & Authorization
    - Kerberos, ACLs
  - Encryption
    - HDFS Transparent Encryption, Cloud encryption
  - OS Security
    - File Permissions
    - Process/Container isolation

NoSQL:
Aside: The CAP Theorem

- CAP Theorem



NoSQL: Guarantees
ACID vs BASE

- ACID
  - Atomicity, Consistency, Isolation, Durability
  - Hard to enforce on distributed system… *why?*
- BASE
  - Relaxes ACID
  - BA: Basic availability
  - S: Soft state
  - E: Eventual consistency (strong or weak)

NoSQL vs Old School SQL

- NoSQL != “not RDBMS” (necessarily)
- Dimensions / Properties where NoSQL systems vary or are novel:
  - Data Model
    - Key/value, columnar document, graph, etc..
  - Storage Model
    - In-memory or persistent
  - Consistency Model
    - Eventual or strict consistency? Tunable?
  - Physical Model
    - Share-nothing, master/slave, master/replica
  - Read/Write Performance Tradeoffs
    - Read-heavy use case, write-heavy use case, lookups or scans
  - Indexes
    - Secondary Indexes
  - Failure Handling and High Availability
    - CAP theorem
  - Client Usage
    - Query-language, API, CLI

NoSQL:
Extended Example, Imagine Building Canvas…

- Storing data for an assignment gradebook
  - Week 1
    - Student
    - Simple text box input
    - Grade

NoSQL:
Extended Example, Imagine Building Canvas…

- Storing data for an assignment gradebook
  - Week 1
    - Student
    - Simple text box input
    - Grade
  - Week 2
    - Student
    - Multiple choice questions
    - Grade

NoSQL:
Extended Example, Imagine Building Canvas…

- Storing data for an assignment gradebook
  - Week 1
    - Student
    - Simple text box input
    - Grade
  - Week 2
    - Student
    - Multiple choice questions
    - Grade
  - Week 3
    - Student
    - Link to source code
    - Description of project
    - Binary of JAR file
    - Grade

NoSQL:
Extended Example, Imagine Building Canvas…

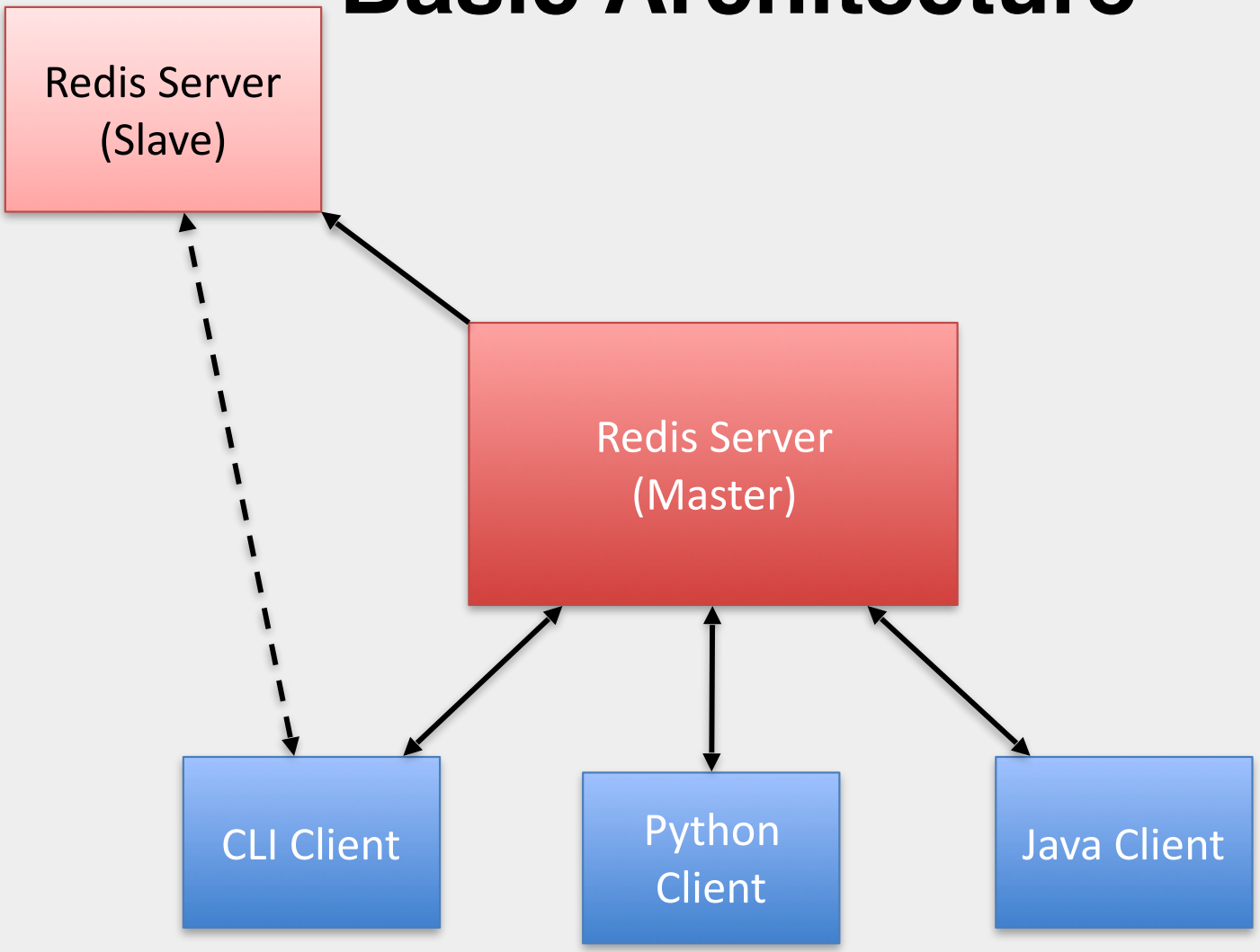
Key: Student

CF1	Grade	Text			
CF2	Grade	Answer 1	Answer 2	Answer 3	Answer 4
CF3	Grade	Link	Text	BLOB	

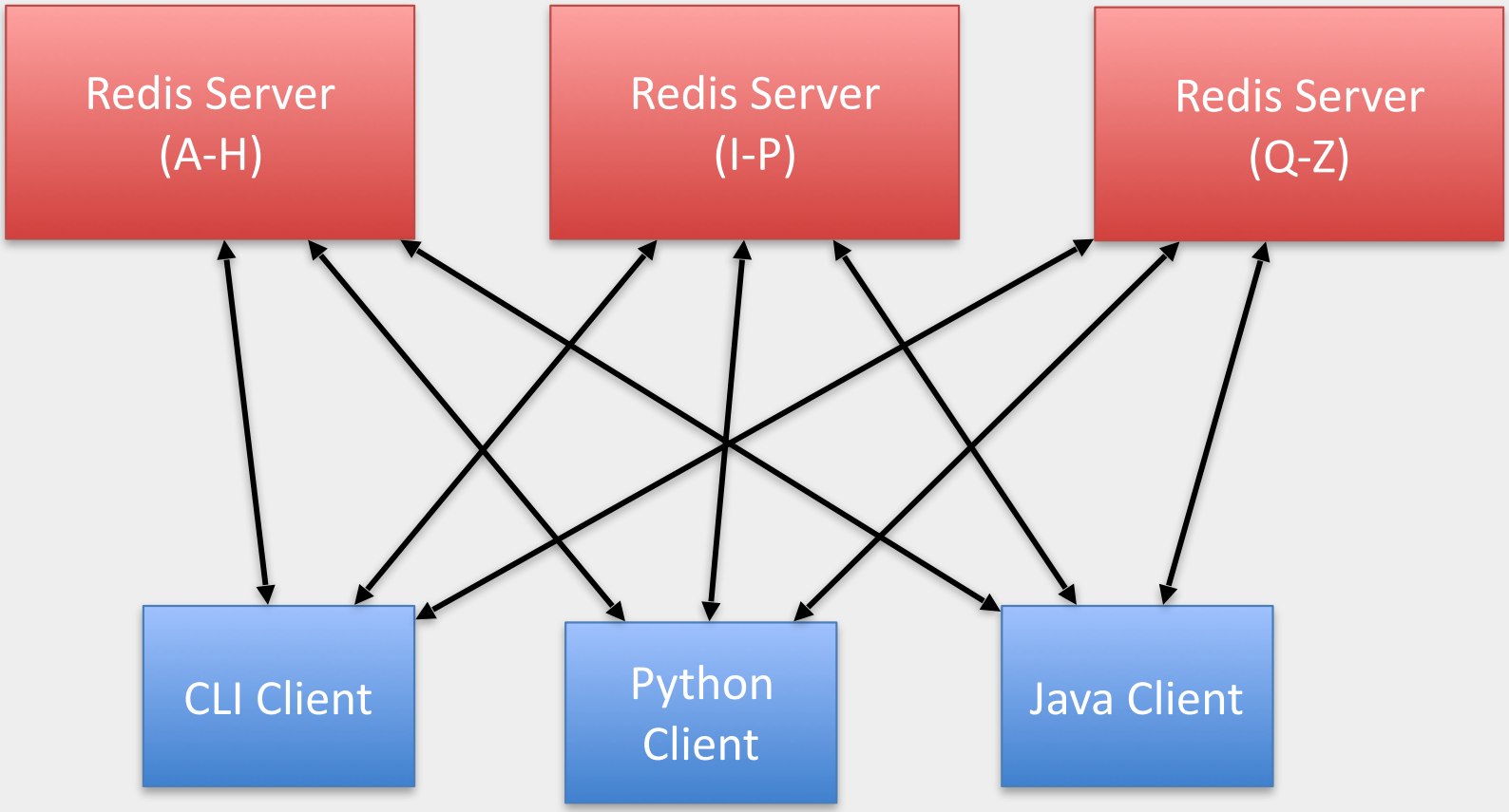
Redis: A Key Value Store

- An **in-memory** key/value store
  - Fast, because it's in-memory…
    - Disk flush for persistence
  - Used variously as a database, cache, and/or message broker (with varying suitabilities)
  - Supports several complex data structures
    - Hash, set, scored sets, geospatial data
    - Keys can expire
  - Master-slave replication
  - Cluster mode for scalability
    - Handles partitioning for you
    - Can manually partition multiple servers well

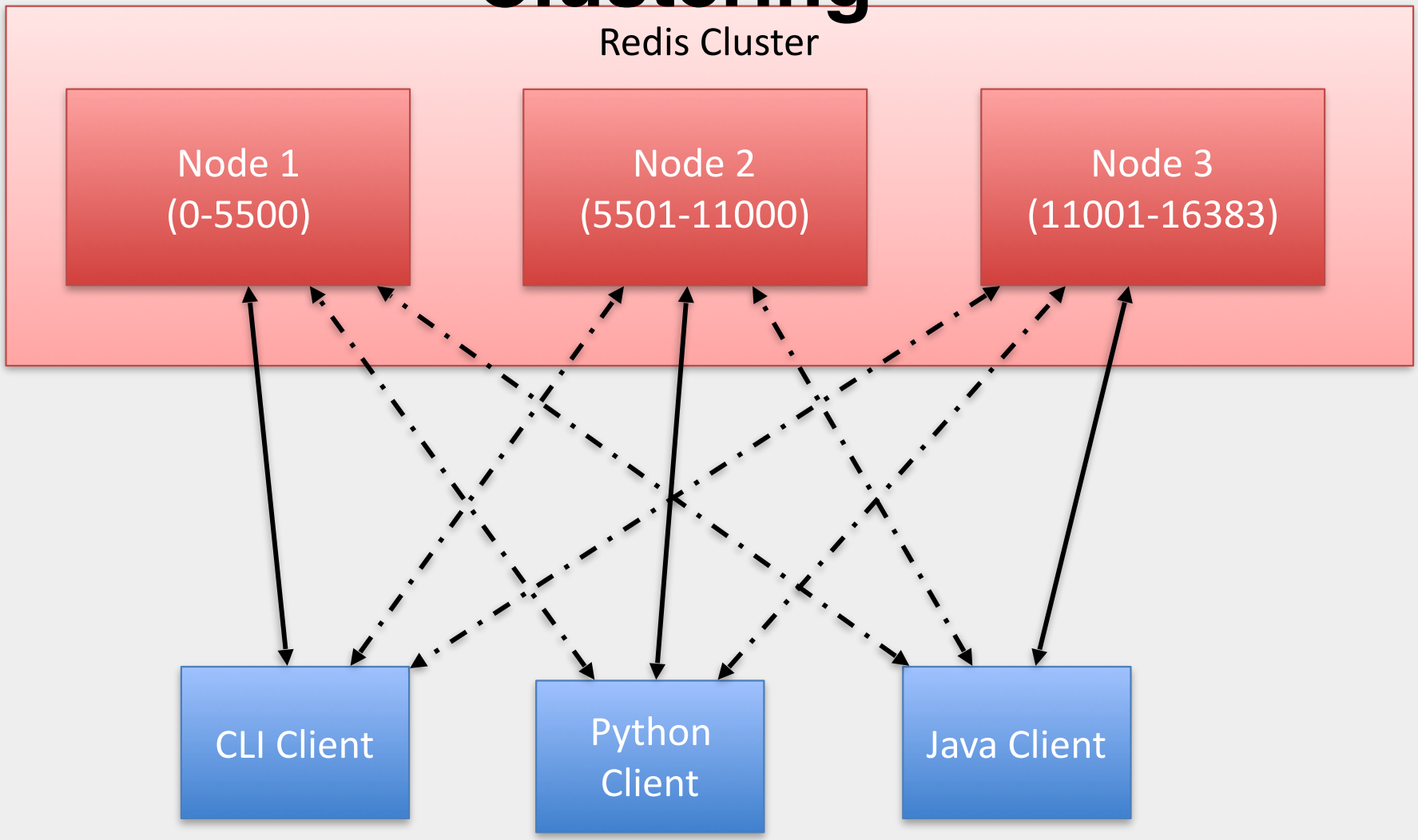
Redis: Basic Architecture



Redis: Sharding



Redis: Clustering



Hash Slot = CRC16(key) % 16384

Redis: The APIs
GET and SET

- Strings
  - The “simple” key/value
  - SET key “value”
  - GET key
    - value

Redis: The APIs
Hashes

- Hashes
  - HSET fooHash f1 “v1”
  - HGET fooHash f1
  - HSET fooHash f2 “v2”
  - HGET fooHash f2
  - HKEYS fooHash

  - Using these data structures is mostly about remembering the types and having a cheatsheet of commands handy.

Redis: The APIs
Sets

- Sets
  - SADD key Jason
  - SADD key Erin
  - SADD key Aidan
  - SADD key Jason

  - SMEMBERS key
    - {Jason, Erin, Aidan}

Redis: The APIs
Sorted Sets

- Sorted Sets
  - ZADD grades 100 Jason
  - ZADD grades 95 Aidan
  - ZADD grades 75 Erin

  - ZRANGEBYSCORE grades 90 100
    - 1)Jason
    - 2)Aidan

Redis:
Pros and Cons

- Advantages
  - Speed
  - Simplicity
  - Redis Data Structures
- Disadvantages
  - Limited by memory
  - Cluster support is not great
  - No rich “querying” ability
- Most Common Use: *caching* as part of some bigger application

HBase: A non-relational distributed database

- Column oriented Key/value store (à la BigTable)
  - Another "Database atop Hadoop (i.e. HDFS)"
    - Fault tolerance through HDFS replication
    - For tables with billions of rows and millions of columns
  - Random real-time data access
    - If you know the key or range of keys
    - Allows table inserts, updates and deletes
  - Data is stored in cells
    - Logically a multi-dimensional map (what this means…later!)
      - There's no upfront schema
    - Can store multiple versions of each cell; history is tracked and queryable!
    - Handles *sparse data* well
  - Strong consistency
  - High availability
    - Horizontal scalability with automatic sharding and failover

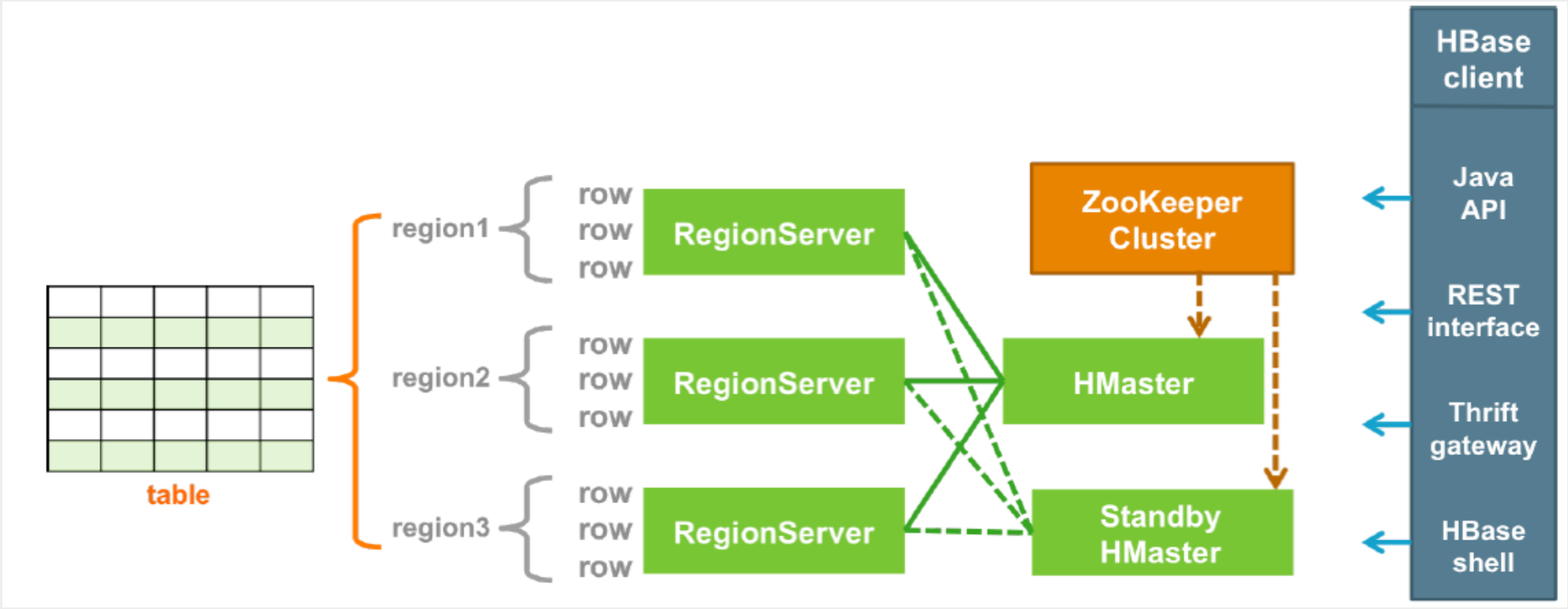
HBase:
Notable Features

- Compression
- In-memory operation
- Bloom filters on per-column basis
- Can serve as input or output for Hadoop MR jobs
- Accessible through:
  - Java API
  - REST
  - Avro and Thrift gateways
- High throughput
- Low latency
- A SQL layer exists to run atop it (Apache Phoenix)

HBase: Components of the System

- Components of the Base system:
  - HMaster
    - Coordinates cluster and region assignment
    - 0..n standby HMasters
  - Zookeeper
    - Distributed coordination service
  - RegionServer
    - 1..n region servers
    - Manages a partition of one or more table(s)
      - HFiles (on HDFS) and MemStore (in memory)
      - Data from HBase is stored on HDFS in Hfiles (sorted)
    - Co-located with HDFS DataNode
  - Apache Phoenix
    - SQL “skin” for HBase

HBase:
Cluster Architecture



HBase: Tables

- Multi-part key identifies a “cell” with a value
- rowKey divides table into rows lexicographically
- Column family organizes the storage for a family together on disk
- Column is just a qualifier for a value
- Timestamp is used to “version” the data
- Tables can be sparse—not every cell requires a key->value mapping

	Column Family 1		Column Family 2		
rowKey	**Col1**	**Col2**	**Col3**	**Col4**	**Col5**
Key1	valueA	valueB	valueC	valueD	ValueE
Key2	valueF	valueG		valueI	valueJ
key3	valueK		valueM	valueN	

(key1, Column Family1, Col1, *timestamp*) -> valueA

(key3, Column Family2, Col3, *timestamp*) -> valueM

HBase: Regions

- Regions… recall the figure:
  - A *region* is a range of consecutive rows
  - A *table* is made up of 1..n regions
    - Each region served by only one region server
  - Number of regions is critical for load balancing—you don't want to bottleneck your operations to 1 or a few servers
  - Regions *split* as they grow
    - 10GB region splits into 2 regions
  - Split options:
    - Start with 1 and let split automatically (or can force a split manually)
    - Explicitly pre-split regions when table is created
      - Must know key distribution beforehand

```
create 'split_table', 'cf1', SPLITS => ['i', 'q']
```

  - Regions will compact from time to time
    - Merges storage files, garbage collects deleted cells

HBase: Table Creation

- Create Table
  - Specify table name, at least one c.f. and optionally versions to maintain

```
create 't1', 'cf1'
create 't1', {NAME => 'cf1', VERSIONS => 5}
create 't1', {NAME => 'cf1'}, {NAME => 'cf2'}
```

HBase: Get Operation

- ‘get’ gets some or all cells for a given key
- Specify: table name, row
  - Optionally specify a dictionary of: columns, timestamps or versions
  - Can also pass in *filters*
  - See docs or shell help for details
- Steps HBase takes to execute a ‘get’:
  - Ask Zookeeper where Metadata table is (tablename: “meta”)
  - Query hbase:meta for RegionServer handling table/rowkey
  - Request cell from RegionServer handling that row

HBase:
Put Operation

- Insert new version of a cell
- Steps HBase takes to execute a ‘put’:
  - Ask Zookeeper for metadata in hbase:meta
  - Determine region server
  - Write data to RegionServer
  - RegionServer keeps data in memstore…
    - …and simultaneously logs to HDFS the write
  - Minor compactions will flush memstore to HFile on HDFS

HBase: scan and delete

- Data Operations
  - Scan
    - Scan table row by row
      - Can also specify a subset of rows by key
  - Delete
    - Mark a cell(s) for deletion
      - Doesn't actually remove from storage until split or major compaction (uses tombstoning)
      - Pass table name, row and optionally column and timestamp

HBase: Pros, Cons, Whens and Whys

- Good use cases
  - Low latency data access with updates
  - Analytics with large scans
- Poor choices
  - Write-heavy workloads
  - Data that is written and accessed sequentially
  - Large cell values
- Interactive use not very pleasant
  - HBase shell is clunky
  - HBase shell provides little functionality (surprisingly or not?)
  - Many things require writing code pushed up to server (see *corprocessors*)

HBase: Example Use Cases

- Document-oriented database
  - e.g. web search: crawlers insert and update crawled documents, then used to create search indexes via MapReduce against HBase
- Facebook Messages (previously) benefitted from:
  - Reliability and scalability of HBase
  - High write throughput, low latency random reads
- Monitoring systems: system metrics, user interaction data, clickstreams telemetry, crash reporting
- OpenTSDB built atop HBase

Summary: NoSQL, Part I

- Issues in Big Data *management*
  - Modeling, governance, quality, security, etc.
- NoSQL
  - What is it, in so far as it's anything?
- Two very different examples of NoSQL systems:
  - *Redis*: a simple, fast in memory KV-store
  - *HBase*: a non-relational *columnar* database good for sparse data