UNIVERSITY of WASHINGTON

BIGDATA 210: Introduction to Data Engineering

Autumn 2018

Module 8: NoSQL Databases Part II

Jason Kolter

jkolter@uw.edu

Week 8 Agenda

- Project Updates / Discussion
- Introduction to NoSQL Databases
  - Cassandra
  - Elasticsearch

Final Project Discussion

- All but a few have topics / teams
- Does anybody need something from me to proceed?
- Next week, two options:
  - Another lecture
  - In-class workshop / co-working time

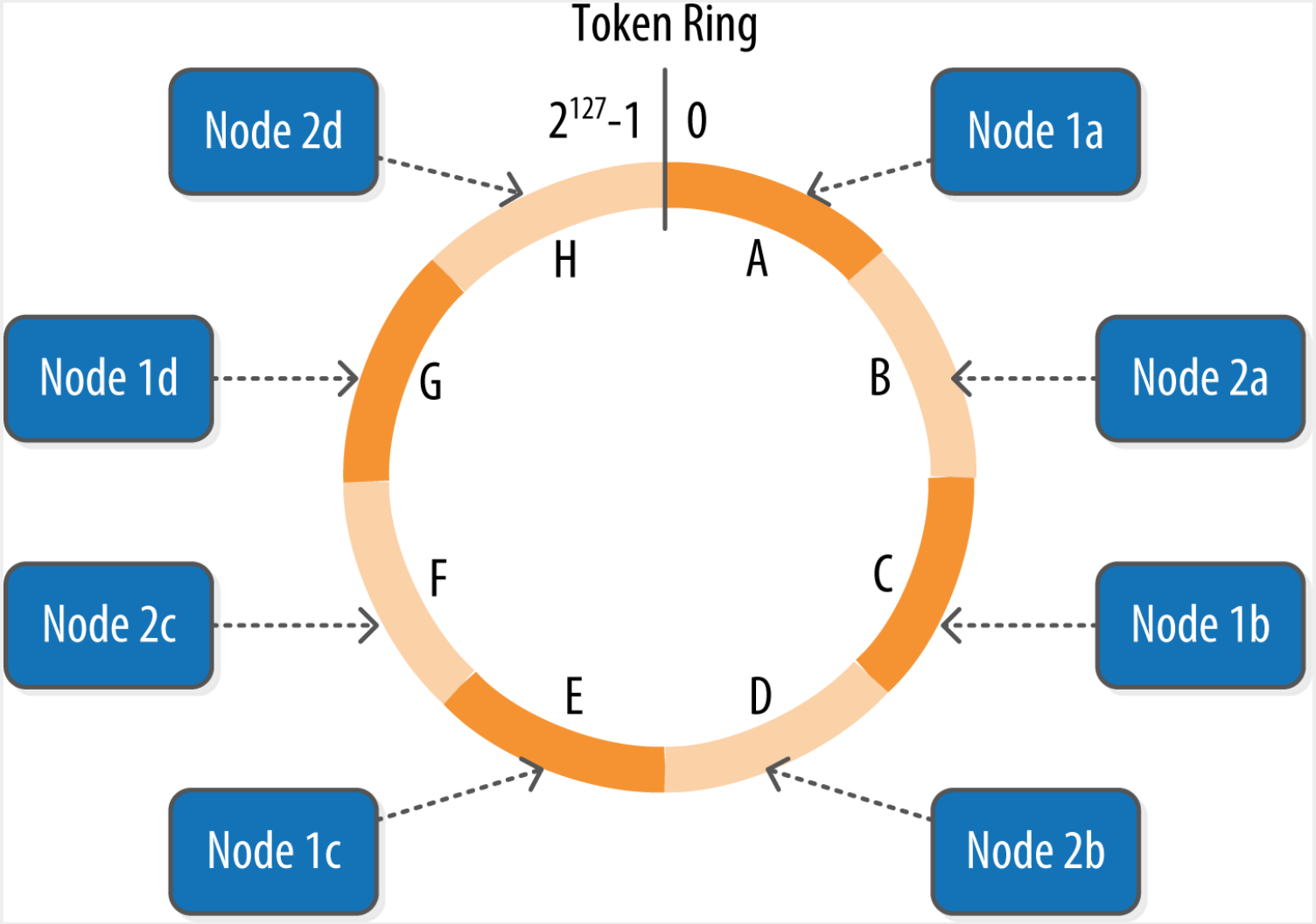
Cassandra: In a Nutshell

- Row-oriented (supports sparseness): data model inspired by BigTable
- Distributed: system structure inspired by Amazon Dynamo
- Decentralized
  - Masterless architecture => Extremely high (linear) scalability
  - No single point of failure or single point of bottleneck
- Elastically scalable
  - Reads and writes scale well
- Highly available and Fault tolerant
- Tunably consistent
- Lightweight transactions (“compare and set”)
- Tables created with a schema (since 3.0)
  - Schema can contain complex data types
  - Has a query language (CQL)

Cassandra: Physical Model

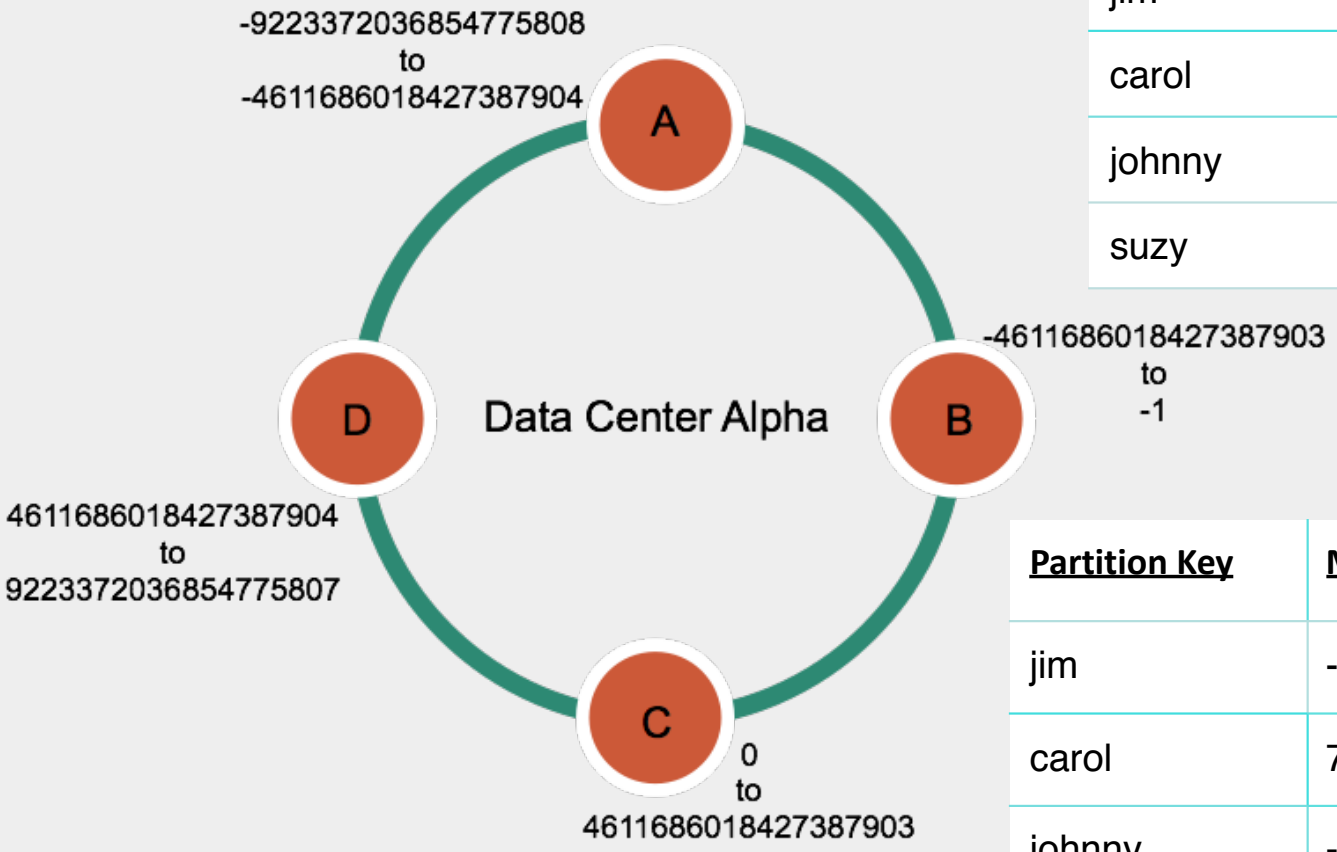
- Cluster
  - Rack and data center awareness
- Nodes
  - Individual servers that store the data for the cluster
  - All nodes are equal
  - Nodes organized into a “ring” architecture
- Token
  - Hashed 64-bit integer ID that identifies partitions (Murmur3)
    - Consistent hashing to minimize reorganization of data
  - Each node is responsible for a range of data by token
- Virtual nodes
  - Break partition space into smaller chunks

Cassandra: Physical Model
How Data is Distributed Across Nodes



Source: Cassandra: The Definitive Guide

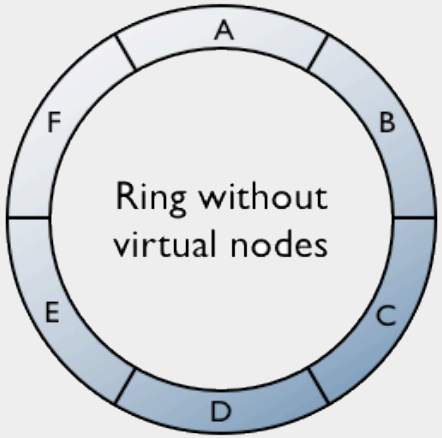
Cassandra: Physical Model
Example of Distributing Data via Hashing

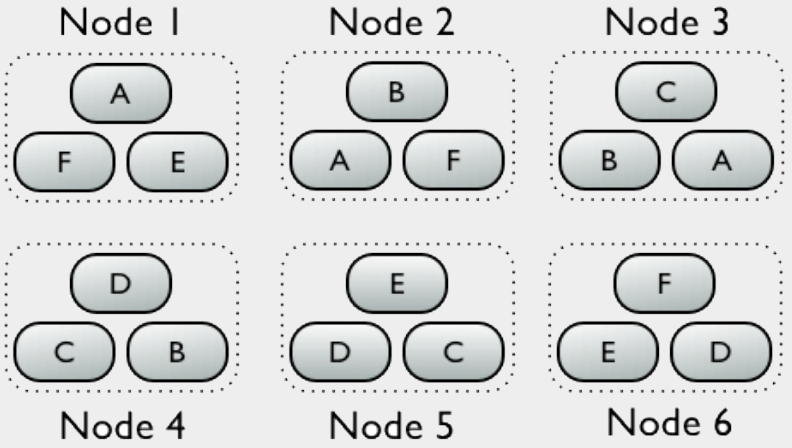


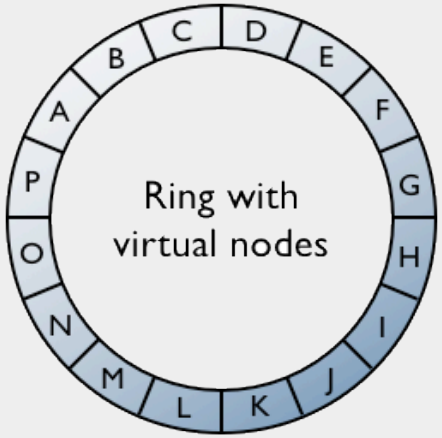
Partition Key	Murmur3 hash value
jim	-2245462676723223822
carol	7723358927203680754
johnny	-6723372854036780875
suzy	1168604627387940318

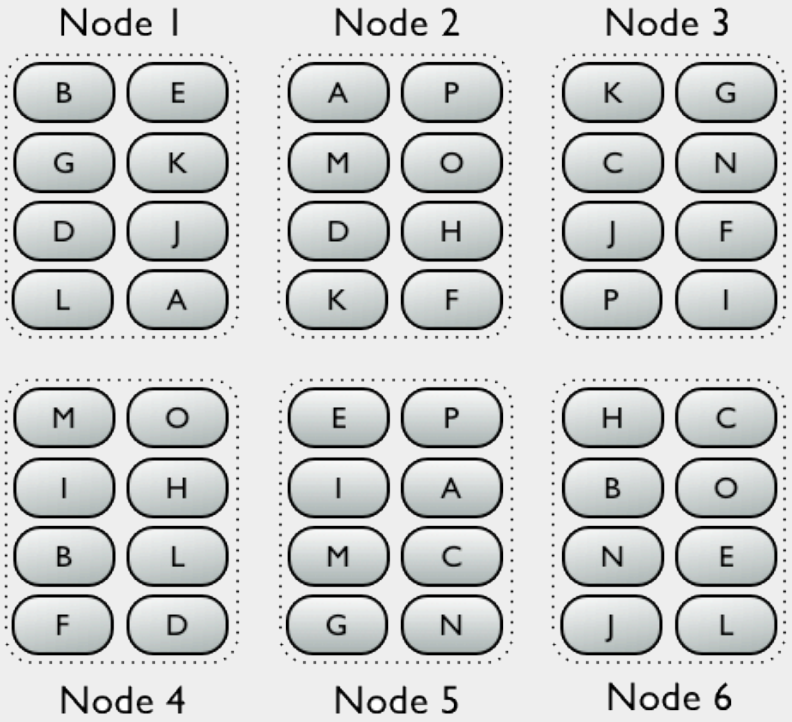
Partition Key	Murmur3 hash value	Node
jim	-2245462676723223822	B
carol	7723358927203680754	D
johnny	-6723372854036780875	A
suzy	1168604627387940318	C

Cassandra: Physical Model
Physical Nodes and Virtual Nodes









Cassandra: How Things are Stored

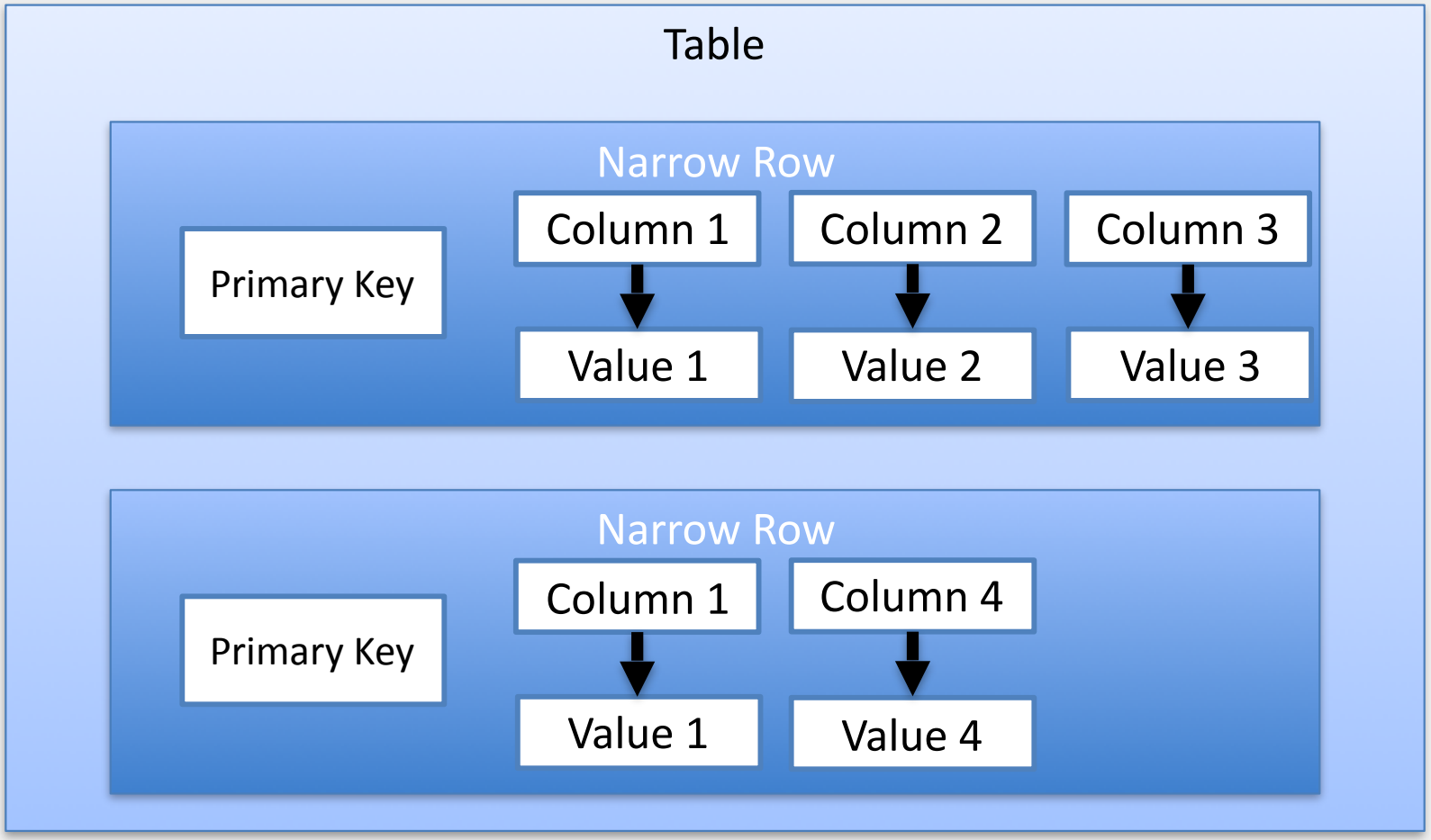
- Cassandra stores data both in memory (for speed) and on disk (for durability)—the principal structures involved are:
- ***Commit logs***
  - A write first goes to a commit log to enable crash recovery
- ***memtables***
  - After write enters commit log it goes to a memory resident *memtable* (stored off the JVM heap)
- ***SSTables***
  - When a memtable hits a threshold size, the memtable is flushed to disk in a file called an SSTable
  - Then a new memtable is created
  - The flush operation is non-blocking
  - Can be multiple SSTables for one memtable, the current one and the rest waiting to flush

Cassandra: Data Model
How Data is Addressed

- Keyspace
  - Container for tables
    - Similar to relational database
- Tables
  - Container for an ordered collections of rows
- Row
  - Identified by a primary key—*this is mandatory*
    - All data access requires using the primary key
      - SELECT, INSERT, UPDATE, DELETE in CQL
      - Queries also need primary key unless define a secondary index
  - Contains a collection of columns
- Column
  - Name/value pairs
  - Value defined to be a static type when table created
    - Simple types
      - text, int, float, boolean, etc…
    - Collections
      - set, list, map
  - Also can specify a Time To Live (TTL) at insertion time
    - If want an entire row to expire must set each column to expire

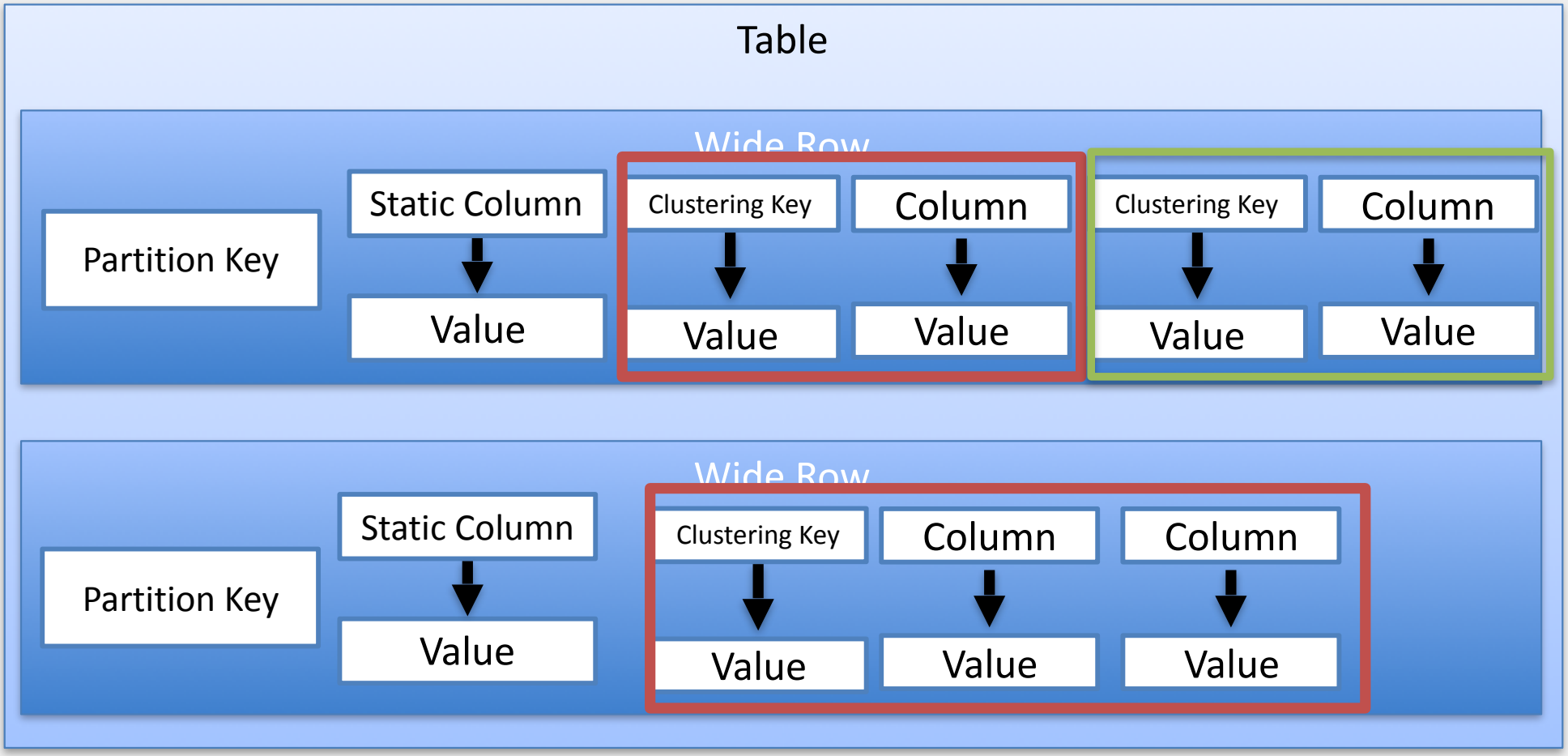
Cassandra: Data Model Skinny Rows

Narrow or "***skinny***" row model have a single *primary key*



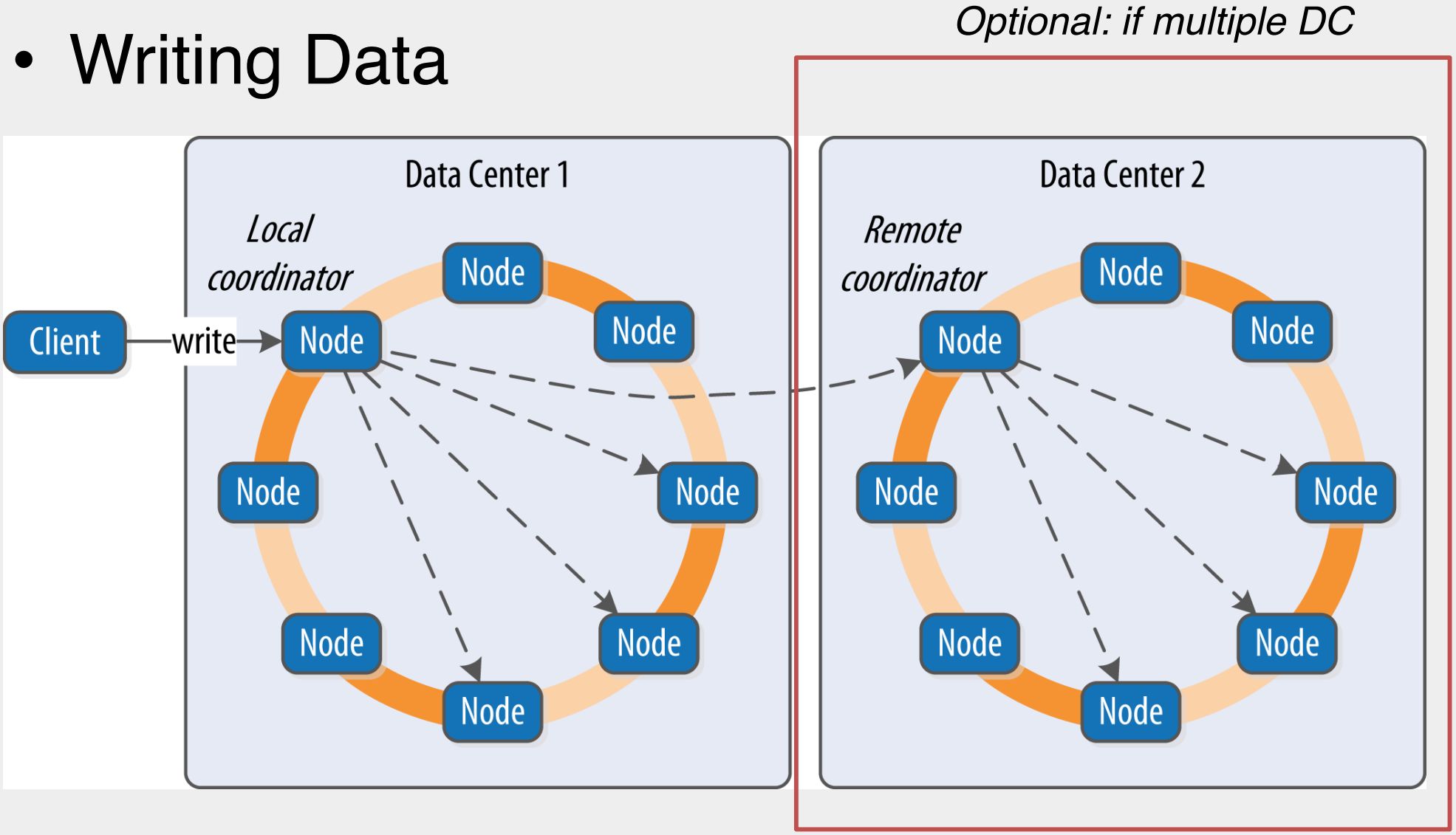
Cassandra: Data Model
Wide Rows

Wide or “fat” Rows use a *composite key*



Cassandra: Write Process

- Writing Data

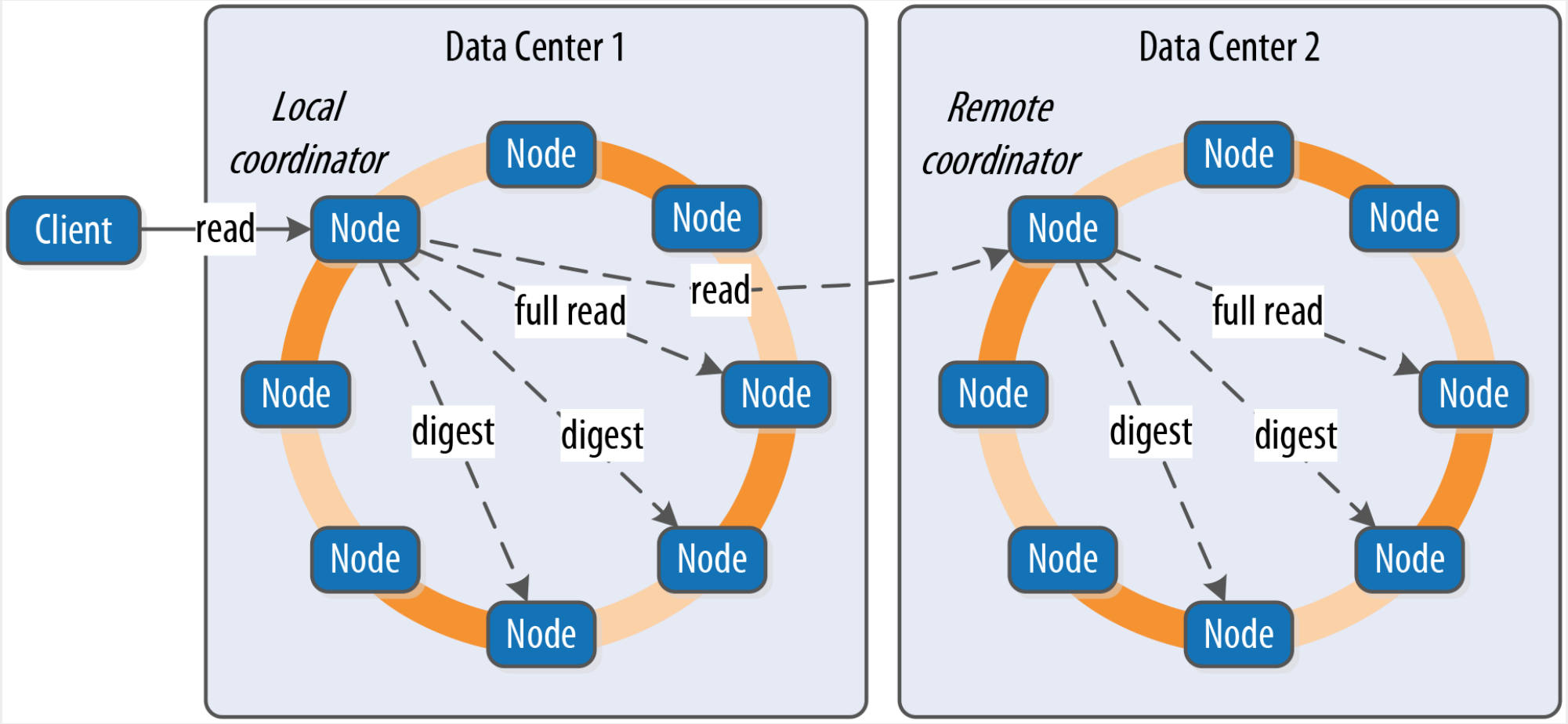


Source: Cassandra: The Definitive Guide

Cassandra: Write Consistency Levels

Consistency level	Behavior
ANY	Ensure that the value is written to a minimum of one replica node before returning to the client
ONE, TWO, THREE	Ensure that the value is written to the commit log and memtable of at least one, two, or three nodes before returning to the client.
LOCAL_ONE	Similar to ONE, with the additional requirement that the responding node is in the local data center.
QUORUM	Ensure that the write was received by at least a majority of replicas (floor(replication factor / 2) + 1).
LOCAL_QUORUM	Similar to QUORUM, where the responding nodes are in the local data center.
EACH_QUORUM	Ensure that a QUORUM of nodes respond in each data center.
ALL	Ensure that the number of nodes specified by replication factor received the write before returning to the client. If even one replica is unresponsive to the write operation, fail the operation.

Cassandra: Read Process



Source: Cassandra: The Definitive Guide

Cassandra: Read Consistency Levels

Consistency level	Implication
ONE, TWO, THREE	Immediately return the record held by the first node(s) that respond to the query. A background thread is created to check that record against the same record on other replicas. If any are out of date, a read repair is then performed to sync them all to the most recent value.
LOCAL_ONE	Similar to ONE, with the additional requirement that the responding node is in the local data center.
QUORUM	Query all nodes. Once a majority of replicas (floor(replication factor / 2) + 1) respond, return to the client the value with the most recent timestamp. Then, if necessary, perform a read repair in the background on all remaining replicas.
LOCAL_QUORUM	Similar to QUORUM, where the responding nodes are in the local data center.
EACH_QUORUM	Ensure that a QUORUM of nodes respond in each data center.
ALL	Query all nodes. Wait for all nodes to respond, and return to the client the record with the most recent timestamp. Then, if necessary, perform a read repair in the background. If any nodes fail to respond, fail the read operation.

Cassandra: Summary

- Good Use Cases
  - Write heavy workloads
    - e.g. Logging/IoT data ingestion
  - Time series data
    - e.g. Logging/IoT
  - Things you might use a RDBMS for, but want more flexibility
    - Product Catalogs, Recommendation engines
- Poor Use Cases
  - Lots of updates
  - Full table scans
  - Want joins, need for referential integrity, want serious transactions

The Elastic Stack

- Formerly known as ELK Stack
  - Stack of components to provide a distributed search and analytics engine
- Based on ”indexing” documents
  - Very quickly (near real-time) search within documents for content
- Based on Apache Lucene
  - Lucene was just a library…
  - …so what does ELK add to Lucene?

The Elastic Stack

- Provides a *full text search engine* called *Elasticsearch* built atop Apache Lucene*:*
  - distributed
  - multi-tenant-capable
  - HTTP web interface
  - schema-free JSON documents
- And two other pieces for bringing data in and exposing it usefully (Logstash, Kibana)

ELK: The Elastic Stack
Elasticsearch

- Elasticsearch
  - Core component
  - Lucene client
  - Adds scalability via cluster distribution
  - Adds additional analytics capability

ELK: The Elastic Stack
Logstash

- Logstash
  - Client to send data into Elasticsearch
  - Plugins for common use cases
    - syslog
    - Kafka
    - log4j
    - http

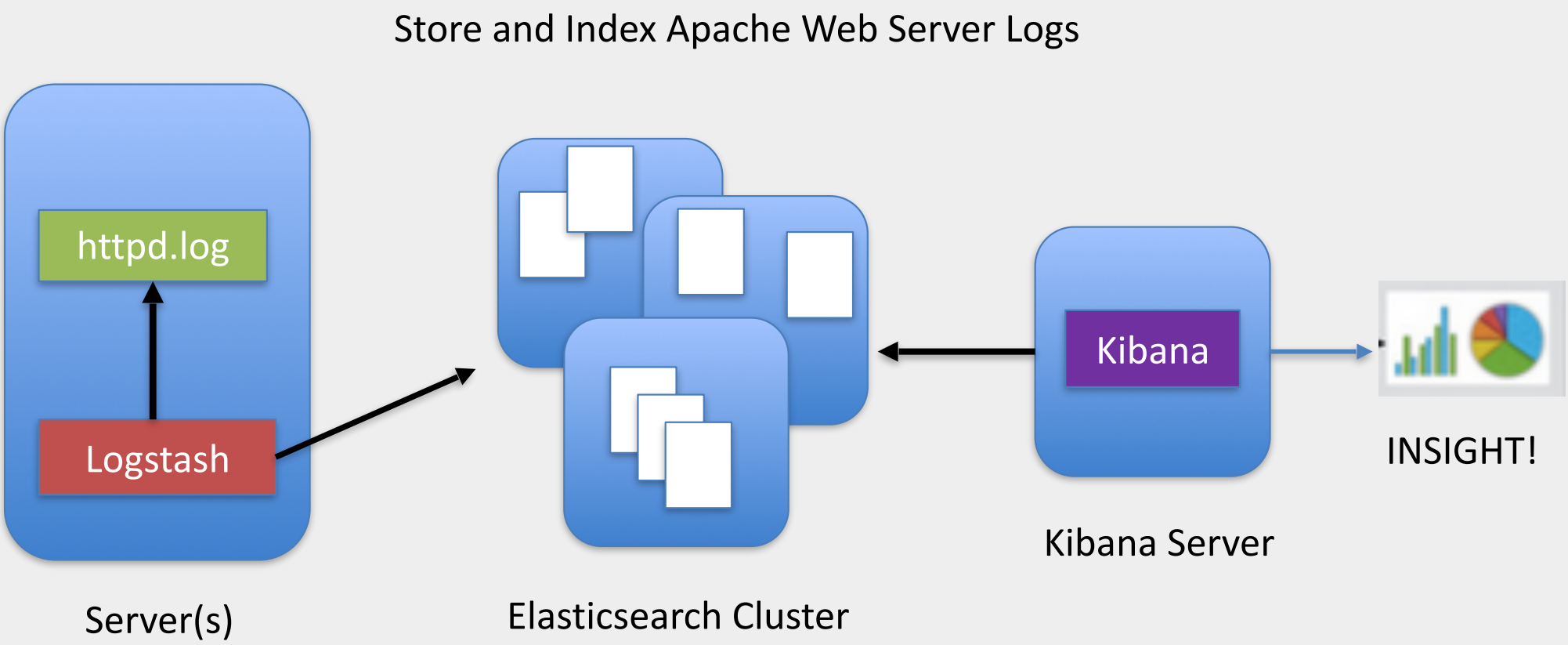
ELK: The Elastic Stack
Kibana

- Kibana
  - Visualization tool for Elasticsearch
  - Pre-integrated with Elasticsearch for common tasks
    - Histograms
    - Basic Charts
    - Machine Learning
    - Geospatial Imaging
  - Also cluster maintenance

ELK: The Elastic Stack

Source: https://www.elastic.co/products/kibana

ELK: Example Architecture



ELK: Concepts

- Document oriented NoSQL Database
  - Index, search, sort and filter DOCUMENTS
  - Documents are stored as JSON

- Cluster
  - 1 or more nodes
    - Data nodes store data and do query work
    - Requires at least one “master” to do lightweight coordination
    - Also can specify nodes specifically for “ingest” of data

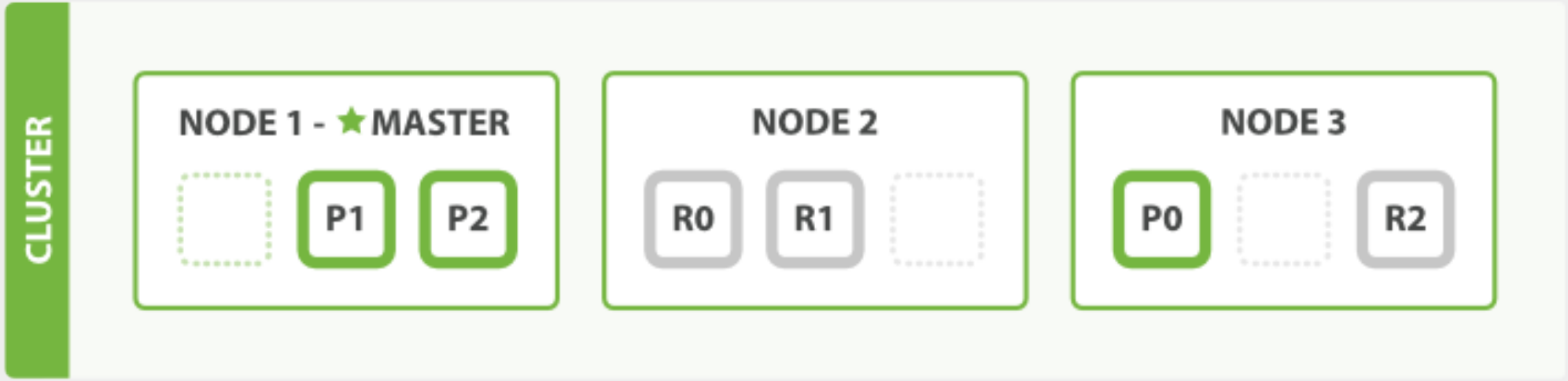
ELK: Concepts

- Document
  - Basic unit of information containing fields
- Index
  - Collection of documents with similar (not necessarily the same) characteristics
- Type
  - Logical category for an index

ELK: Concepts

- Shards
  - Subset of an index suitable for distributing among a cluster
  - Each shard is a fully functional Lucene index
  - Distribution of data among shard based on ID
- Replicas
  - Copies of data for high availability and scalability
  - Primary and replicas
    - Reads CAN be served from replicas

ELK: Concepts



ELK: Concepts

- Working with Elasticsearch
  - HTTP REST API
  - Can also use clients in most languages
    - These basically just wrap the REST calls

ELK: Create an Index

```
PUT /customer

curl -XPUT 'localhost:9200/customer
```

```
PUT /blogs
{
  "settings" :
     {
      "number_of_shards" : 3,
      "number_of_replicas" : 1
     }
 }

curl -XPUT 'localhost:9200/blogs?pretty' -H 'Content-Type: application/json' -
d'{ "settings" : {  "number_of_shards" : 3,  "number_of_replicas" : 1   }}'
```

ELK: Index a document

```
PUT /megacorp/employee/1
{
  "first_name" : "John",
  "last_name" : "Smith",
  "age" : 25,
  "about" : "I love to go rock climbing",
  "interests": [ "sports", "music" ]
}
```

```
curl -XPUT 'localhost:9200/megacorp/employee/1?pretty' -H 'Content-Type:
application/json' -d'{ "first_name" : "John", "last_name" :  "Smith", "age" :  25,
"about" :  "I love to go rock climbing",  "interests": [ "sports", "music" ]}'
```

ELK: Search an Index

Return all documents: GET /_search

curl -XGET 'localhost:9200/_search?pretty'

Find all documents that contain the word “elasticsearch” in the “tweet” field:

GET /_all/tweet/_search?q=tweet:elasticsearch

curl -XGET 'localhost:9200/_all/tweet/_search?q=tweet:elasticsearch&pretty'

ELK: Logstash

- Ingestion tool for Elasticsearch
  - More sources and sinks added
- Processing pipeline
  - Input
    - Generate events from a source plugin
  - Filter
    - Modify and enhance
  - Output
    - Send to output plugin

ELK: Logstash

- Logstash “agents”
  - Horizontally scalable
  - Will mitigate backpressure

ELK: Logstash

- Logstash “agents”
  - Configuration based
  - Horizontally scalable
  - Will mitigate backpressure
- Run multiple instances
  - Proximity to source
  - Proximity to data node(s)
- Can use with load balancer

ELK: Simple Configuration

```
input
{
    kafka  {
        bootstrap_servers => '192.168.0.33:9092'
        topics => ['logstash_test']
    }
}
output
{
    stdout {}
}
```

bin/logstash –f example.conf

ELK: Simple Configuration

```
input
{
    kafka  {
        bootstrap_servers => '192.168.0.33:9092'
        topics => ['logstash_test']
    }
}
output
{
      elasticsearch  {
            hosts => "127.0.0.1:9200"
            index => ”from_kafka"
       }
}
```

bin/logstash –f example.conf

ELK: Simple Configuration

```
input
{
    kafka  {
        bootstrap_servers => '192.168.0.33:9092'
        topics => ['logstash_test']
    }
}
filter
{
     mutate { lowercase => [ "fieldname" ] }
}
output
{
      elasticsearch  {
            hosts => "127.0.0.1:9200"
            index => ”from_kafka"
        }
}
```


bin/logstash –f example.conf

ELK: Production Considerations

- Elasticsearch is *not* a database
  - Use if if you want realtime-ish search…
  - …on semi-structured data you can index sensibly
  - Elasticsearch is probably not your desired primary warehouse
- Beware hardware requirements
  - Hundreds of nodes to handle TB of data
  - 32GB Java heap limit => creates GC problems especially in older JVMs…
  - …and Elasticsearch serves out of memory, so it's resource intensive
  - Index storage overhead
- IPv6 issues (at least until recently)
- No schemas => create mappings ahead of time

NoSQL Databases Part II Summary

- Cassandra
  - Decentralized, Dynamo-inspired system structure, BigTable-inspired data model, replicated, scalable, fault-tolerant, tunable consistency, columnar, sparse NoSQL database
- The ELK Stack
  - Suite of tools providing a full text search engine built on Lucene, plus…
  - Logstash a component for ingestion
  - Kibana for visualization, analysis, presentation.