Eötvös Loránd University (ELTE)
Faculty of Informatics (IK)
Pázmány Péter sétány 1/c
1117 Budapest, Hungary

DATA STORAGE #1: MONGODB, LOGSTASH & CASSANDRA

Open Source Technologies for Real-Time Data Analytics

Imre Lendák, PhD, Associate Professor

2020
Budapest, Hungary

Key storage requirements

- Ability to handle extremely large amounts of data → distributed storage
- Ability to handle both structured and unstructured data
- Open source access → vibrant community driven by business needs developing additional tools (a'la Hadoop, as opposed to MapReduce which was internal to Google)
  - Commercial use
  - Distribution
  - Modification
  - Patent use
  - The open source 'consumer' might not want to open source the new code

Data storage timeline

Database world

- PostgreSQL (from – to)
- MySQL (from – to)
- SQLite (from – to)

‘Big data’ world

- MapReduce (from – to)
- Hadoop (from – to)
- MongoDB (from – to)
- Cassandra (from – to)
- CouchDB (from – to)
- Neo4j (from – to)
- Logstash (from – to)

Data storage timeline

Database world

- PostgreSQL (1996 – now)
- MySQL (1995 – now)
- SQLite (2000 – now)

‘Big data’ world

- MapReduce (2003-2014)
- Hadoop (2004 – now)
- MongoDB (2009 – now)
- Cassandra (2008 – now)
- CouchDB (2005 – now)
- Neo4j (2007 – now)
- Logstash (2010 – now)

RDBMS timeline

	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	…	2016	2017	2018	2019	2020
MySQL	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
PostreSQL		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
SQLite						🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

‘Big data’ storage timeline

	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020
MapReduce	■	■	■	■	■	■	■	■	■	■	■	■						
Hadoop*		■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
CouchDB			■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
Neo4j					■	■	■	■	■	■	■	■	■	■	■	■	■	■
Cassandra						■	■	■	■	■	■	■	■	■	■	■	■	■
MongoDB							■	■	■	■	■	■	■	■	■	■	■	■
Logstash								■	■	■	■	■	■	■	■	■	■	■

* Hadoop lives as a distributed data storage platform, not as data processor

Chosen technologies

- **MongoDB** is a general purpose, document-based, distributed database
- **Logstash** is the log management element of the ELK stack
- **Cassandra** is a decentralized structured storage system

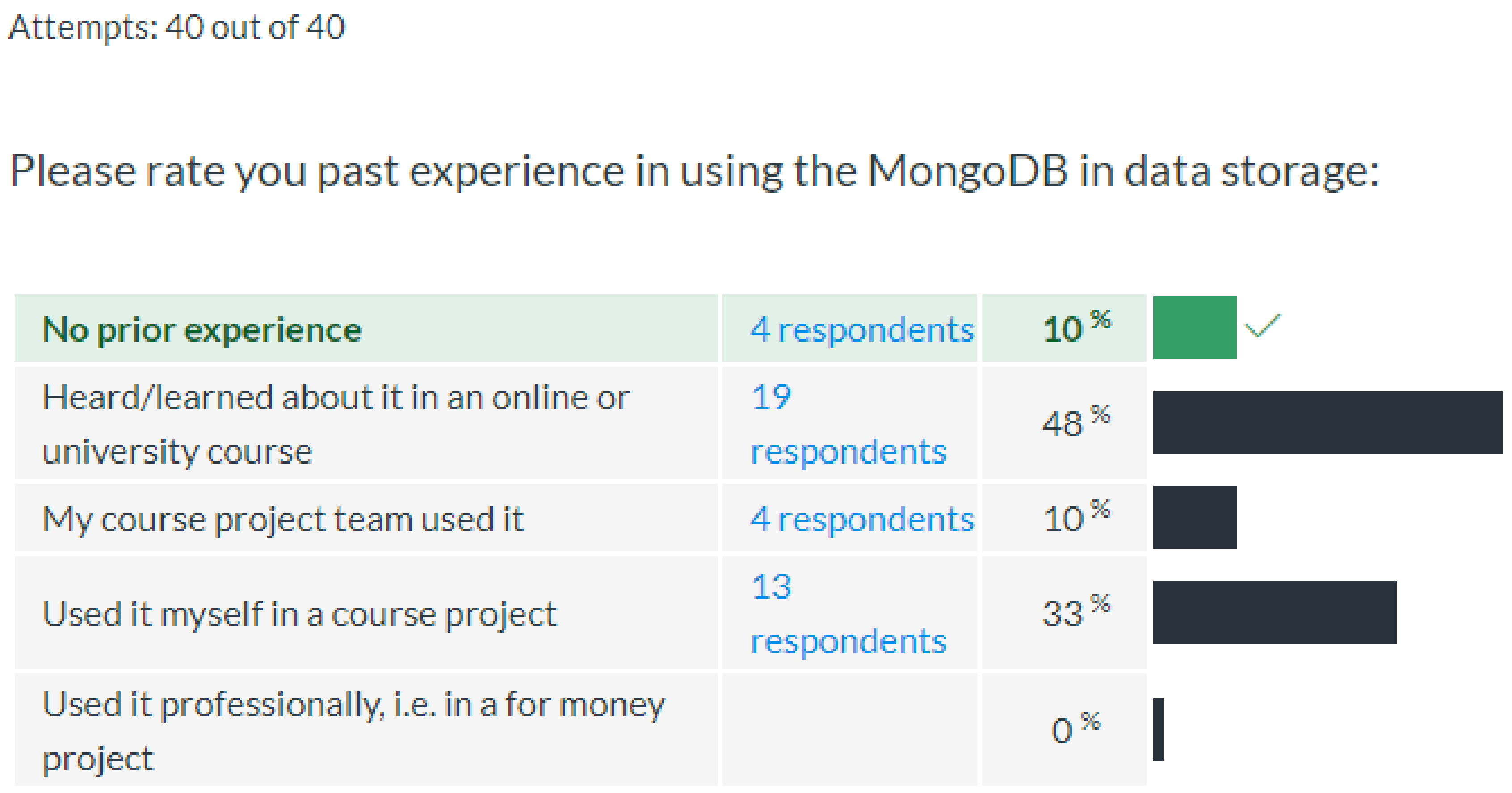
Mongo DB

MongoDB in the timeline

	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020
MapReduce	■	■	■	■	■	■	■	■	■	■	■	■						
Hadoop*		■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
CouchDB			■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
Neo4j					■	■	■	■	■	■	■	■	■	■	■	■	■	■
Cassandra						■	■	■	■	■	■	■	■	■	■	■	■	■
MongoDB							■	■	■	■	■	■	■	■	■	■	■	■
Logstash								■	■	■	■	■	■	■	■	■	■	■

* Hadoop lives as a distributed data storage platform, not as data processor

Why MongoDB @ OST?



Attempts: 40 out of 40

Please rate you past experience in using the MongoDB in data storage:

No prior experience	4 respondents	**10 %**	✓
Heard/learned about it in an online or university course	19 respondents	48 %	
My course project team used it	4 respondents	10 %	
Used it myself in a course project	13 respondents	33 %	
Used it professionally, i.e. in a for money project		0 %	

History

- **DEF:** MongoDB is a general purpose, document-based, distributed database
- mongoDB = “Hu**mongo**us DB”
  - Open-source
  - Document-based
  - “High performance, high availability”
  - Automatic scaling
  - C-P on CAP

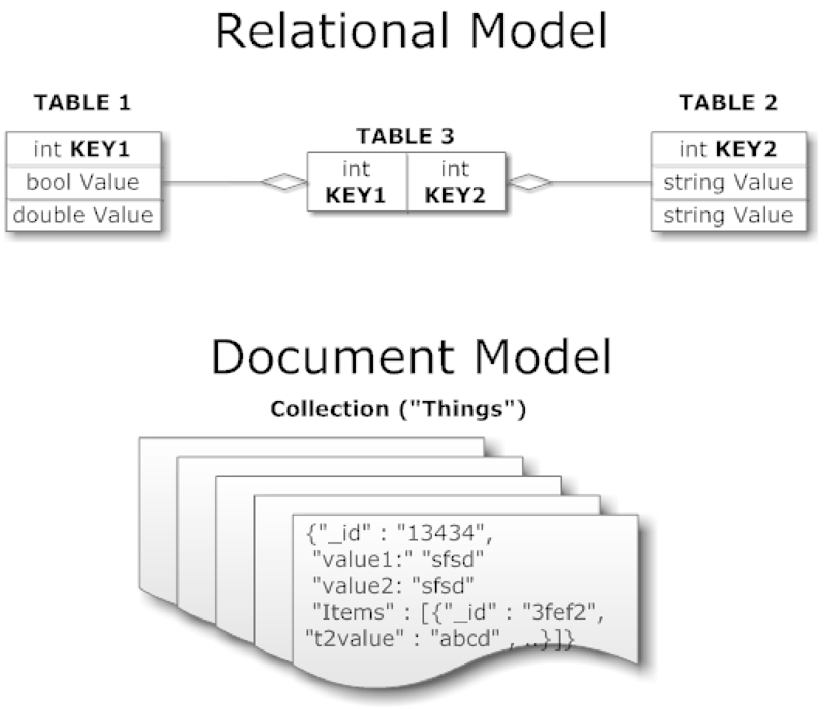
-blog.mongodb.org/post/475279604/on-distributed-consistency-part-1
-mongodb.org/manual

Other NoSQL Types

Key/value (Dynamo)

Columnar/tabular (HBase)

Document (mongoDB)



http://www.aaronstannard.com/post/2011/06/30/MongoDB-vs-SQL-Server.aspx

Motivations

Problems with SQL

- Rigid schema
- Not easily scalable (designed for 90’s technology or worse)
- Requires unintuitive joins

Perks of mongoDB

- Easy interface with common languages (Java, Javascript, PHP, etc.)
- DB tech should run anywhere (VM’s, cloud, etc.)
- Keeps essential features of RDBMS’s while learning from key-value noSQL systems

http://www.slideshare.net/spf13/mongodb-9794741?v=qf1&b=&from_search=13

Data Model

- Document-Based (max 16 MB)
- Documents are in BSON format, consisting of field-value pairs
- Each document stored in a collection
- Collections
  - Have index set in common
  - Like tables of relational DB’s.
  - Documents do not have to have uniform structure

What is BSON?

JSON

- "JavaScript Object Notation"
- Easy for humans to write/read, easy for computers to parse/generate
- Objects can be nested
- Built on
  - name/value pairs
  - Ordered list of values
- E.g. AlienVault threat intelligence feed
- http://json.org/

BSON

- "Binary JSON"
- Binary-encoded serialization of JSON-like docs
- Also allows "referencing"
- Embedded structure reduces need for joins
- Goals
  - Lightweight
  - Traversable
  - Efficient (decoding and encoding)
- http://bsonspec.org/

Deserialized BSON example

```
{
“_id" :         “XXXXX"
"city" :        “Budapest",
"pop" :         1660,
"state" :       “HUN",
“professor” : {
                   name: “Péter Péterffy”
                   address: “Pázmány Péter sétány 1/a”
                 }
}
```

BSON Types

Type	Number
Double	1
String	2
Object	3
Array	4
Binary data	5
Object id	7
Boolean	8
Date	9
Null	10
Regular Expression	11
JavaScript	13
Symbol	14
JavaScript (with scope)	15
32-bit integer	16
Timestamp	17
64-bit integer	18
Min key	255
Max key	127

The number can be used with the $type operator to query by type!

http://docs.mongodb.org/manual/reference/bson-types/

The _id Field

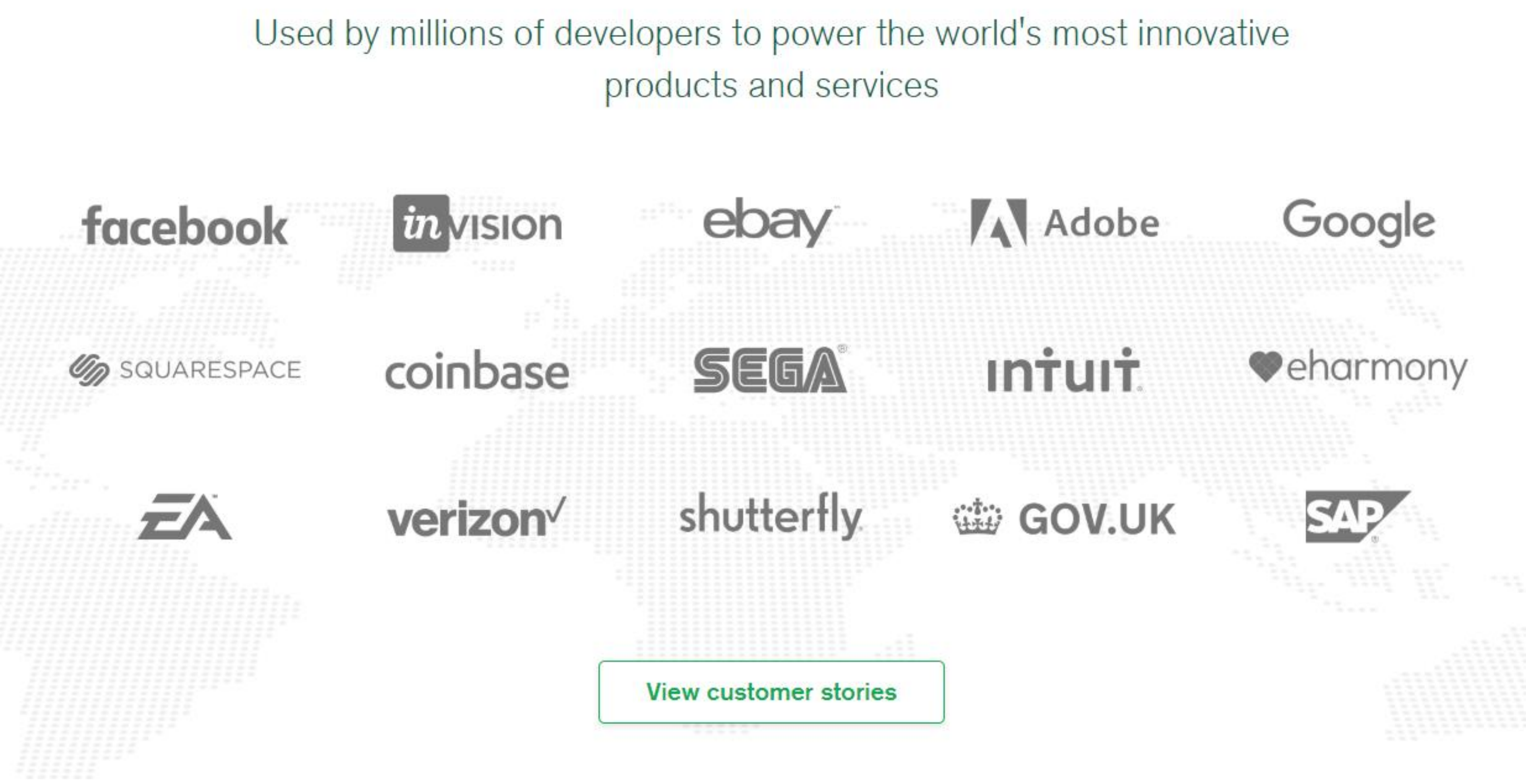
- By default, each document contains an _id field
- _id serves as primary key for collection.
- Its value is unique, immutable, and may be any non-array type.
- The default data type is ObjectId, which is “small, likely unique, fast to generate, and ordered.” Sorting on an ObjectId value is roughly equivalent to sorting on creation time.

http://docs.mongodb.org/manual/reference/bson-types/

mongoDB vs. SQL

	mongoDB	SQL
Data ‘atom’	Document	Tuple
Data group	Collection	Table/View
Identifier	PK: _id Field	PK: Any attribute(s)
Schema	Uniformity not Required	Uniform Relation Schema
Efficiency	Index	Index
Linking	Embedded Structure	Joins
Distribution	Shard	Partition

Who uses Mongo DB?



Used by millions of developers to power the world's most innovative products and services

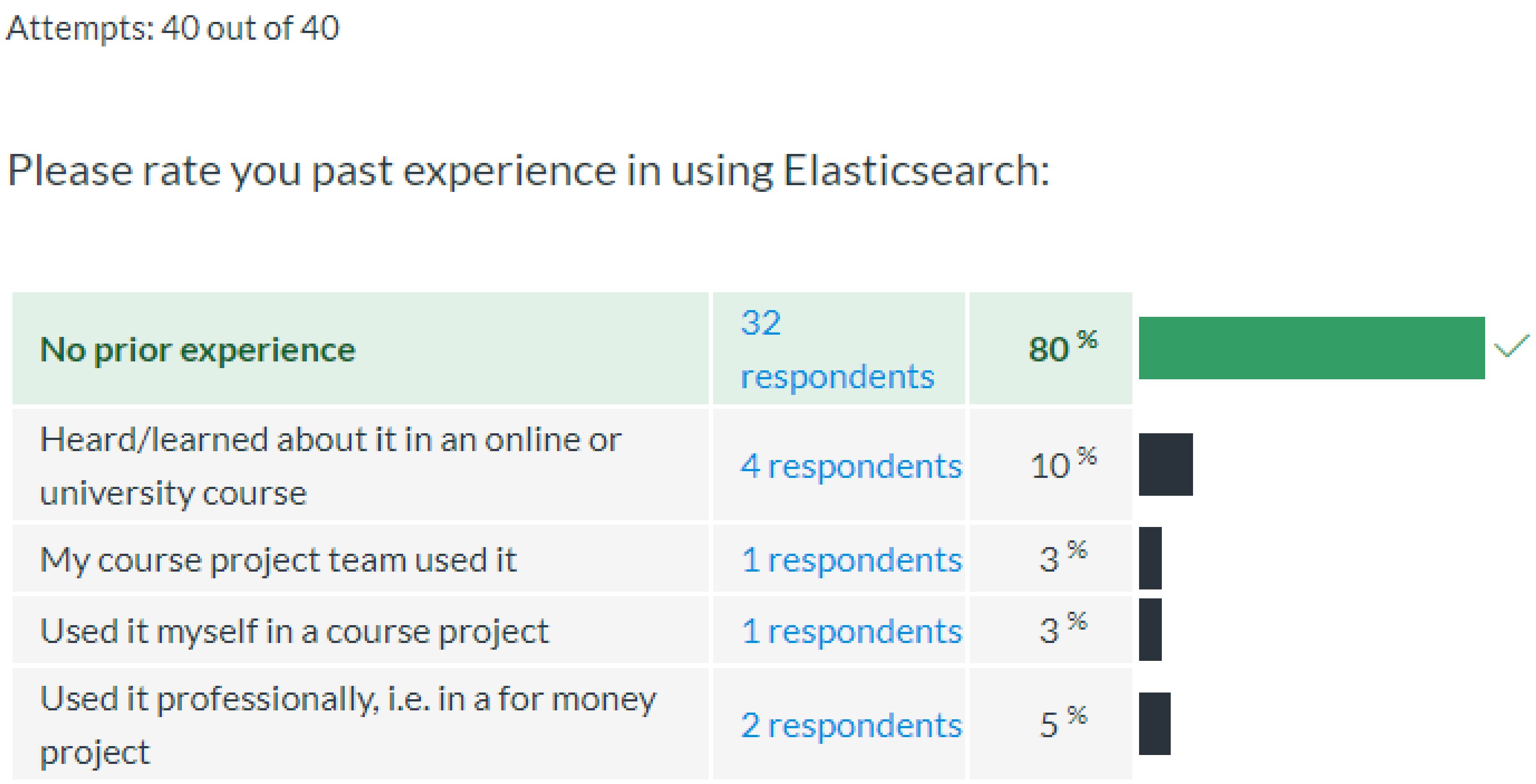
Logstash

Logstash in the timeline

	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020
MapReduce	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦						
Hadoop*		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
CouchDB			🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
Neo4j					▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Cassandra						🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪
MongoDB							🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧
Logstash								🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

* Hadoop lives as a distributed data storage platform, not as data processor

Why Logstash @ OST?

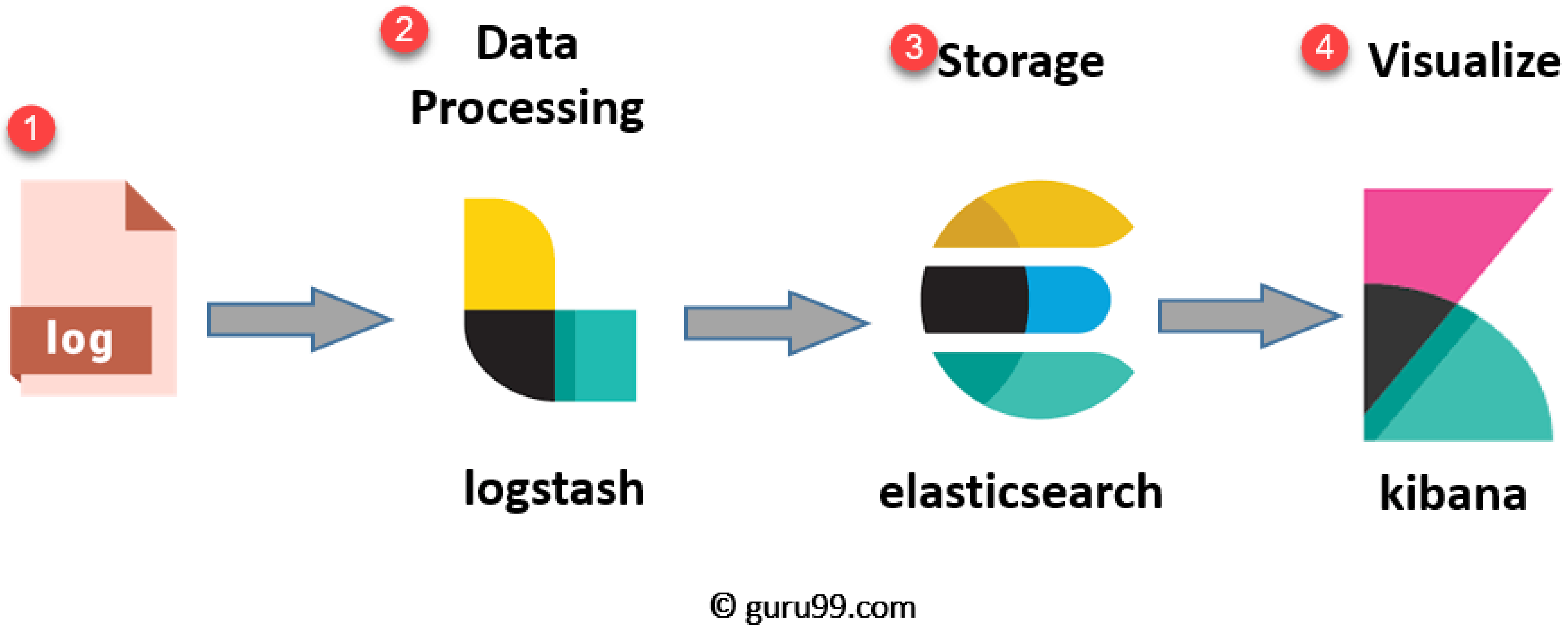


Attempts: 40 out of 40

Please rate you past experience in using Elasticsearch:

No prior experience	32 respondents	**80 %**	✓
Heard/learned about it in an online or university course	4 respondents	10 %	
My course project team used it	1 respondents	3 %	
Used it myself in a course project	1 respondents	3 %	
Used it professionally, i.e. in a for money project	2 respondents	5 %	

ELK stack



https://www.guru99.com/elk-stack-tutorial.html

Logstash

Components & Features

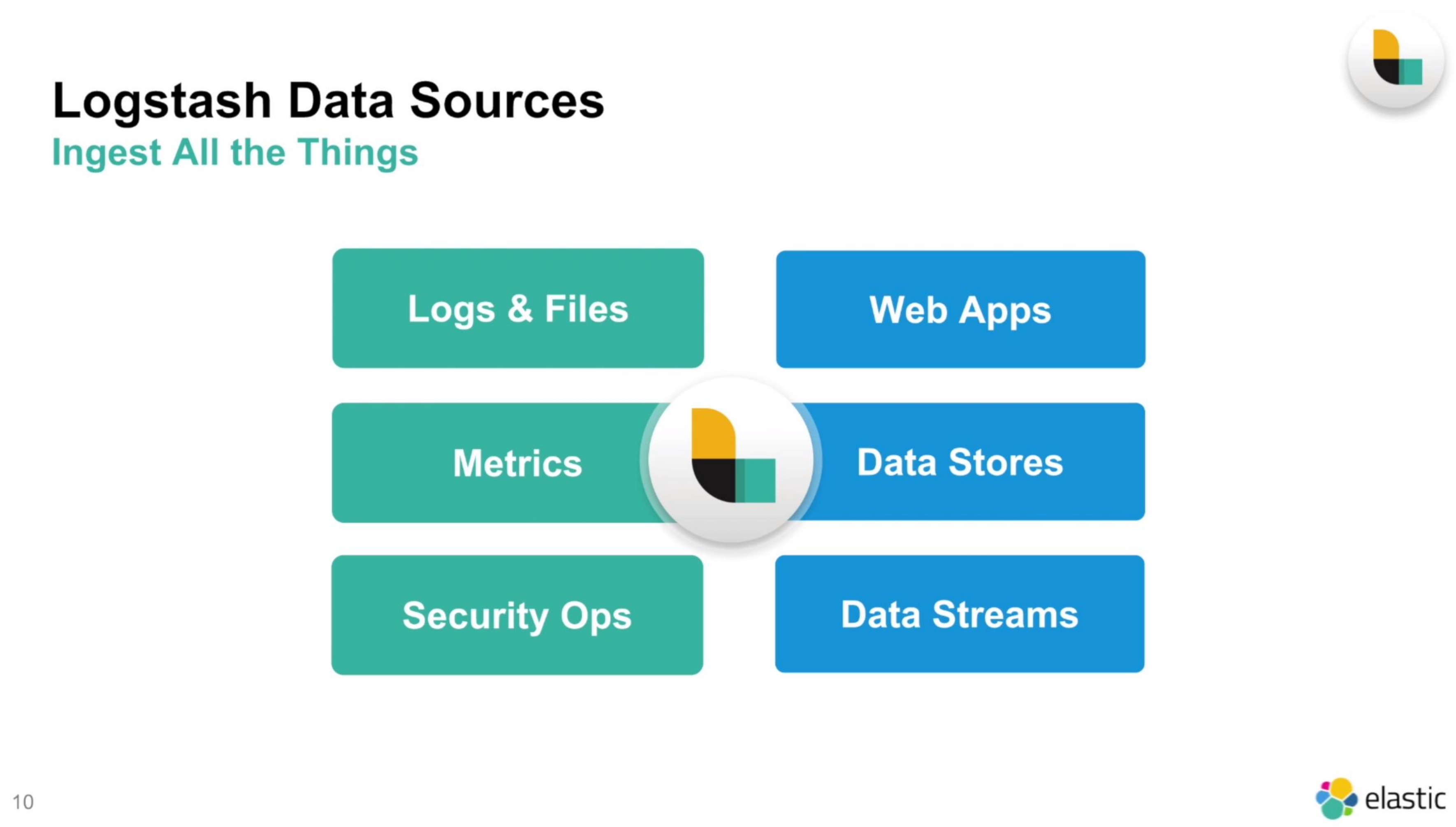
- Is a **data collection** pipeline
- Parse and **normalize** different kinds of logs into machine-based analysis ready format
- Advanced input filtering
- One instance can handle multiple pipelines of related events

- Instance = single LS process
- Event = the primary data unit in Logstash
- Pipeline = separate data flows
- Queue = input data received

Strong points

- Central data processing, like syslog servers earlier
- Accepts a wide variety of structured data
- Accepts unstructured data
- Plugins for various input sources and platforms
- Written in Java → cross platform
- In-memory and on-disk (durable) queues

Logstash use cases

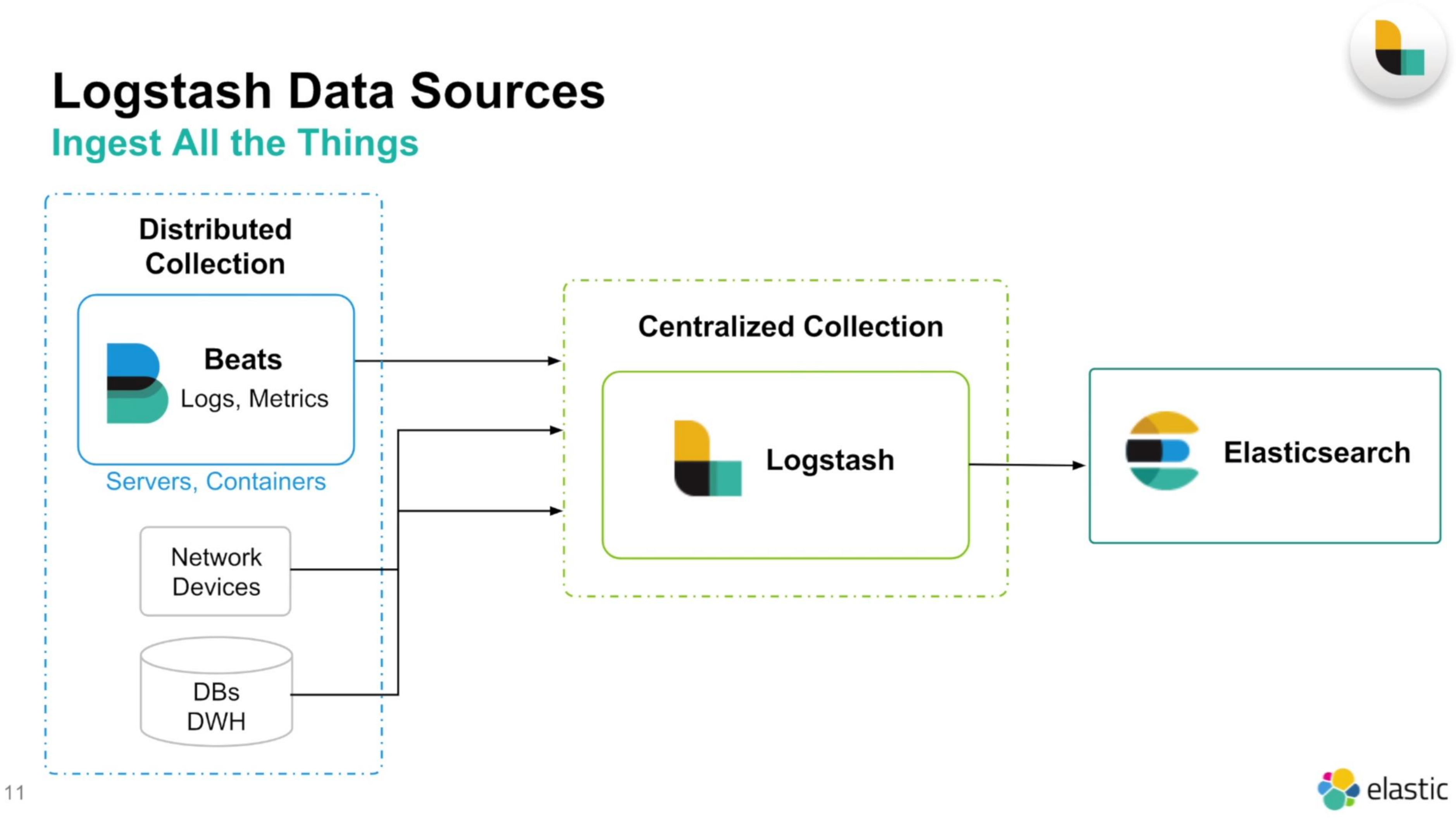


Logstash Data Sources

Ingest All the Things

https://www.elastic.co/webinars/getting-started-logstash

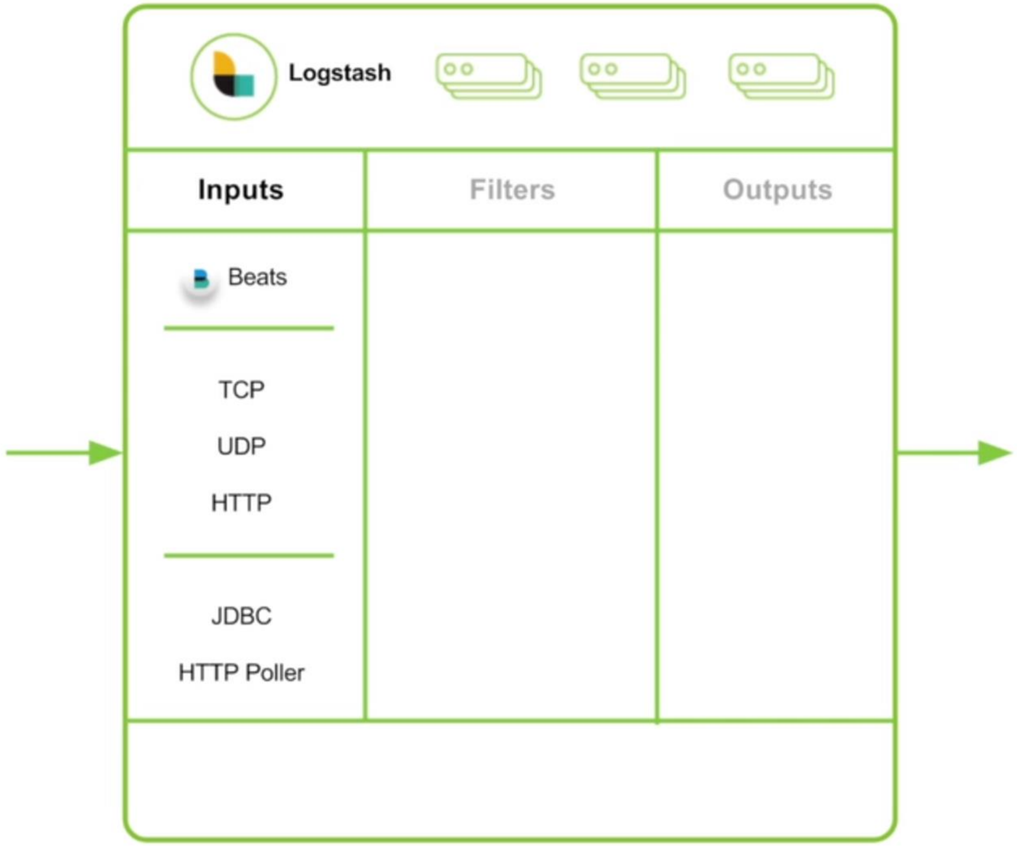
Logstash data sources



https://www.elastic.co/webinars/getting-started-logstash

Logstash Data Collection

- Collect and deserialize data with input plugins
- Codecs may be used for deserialization

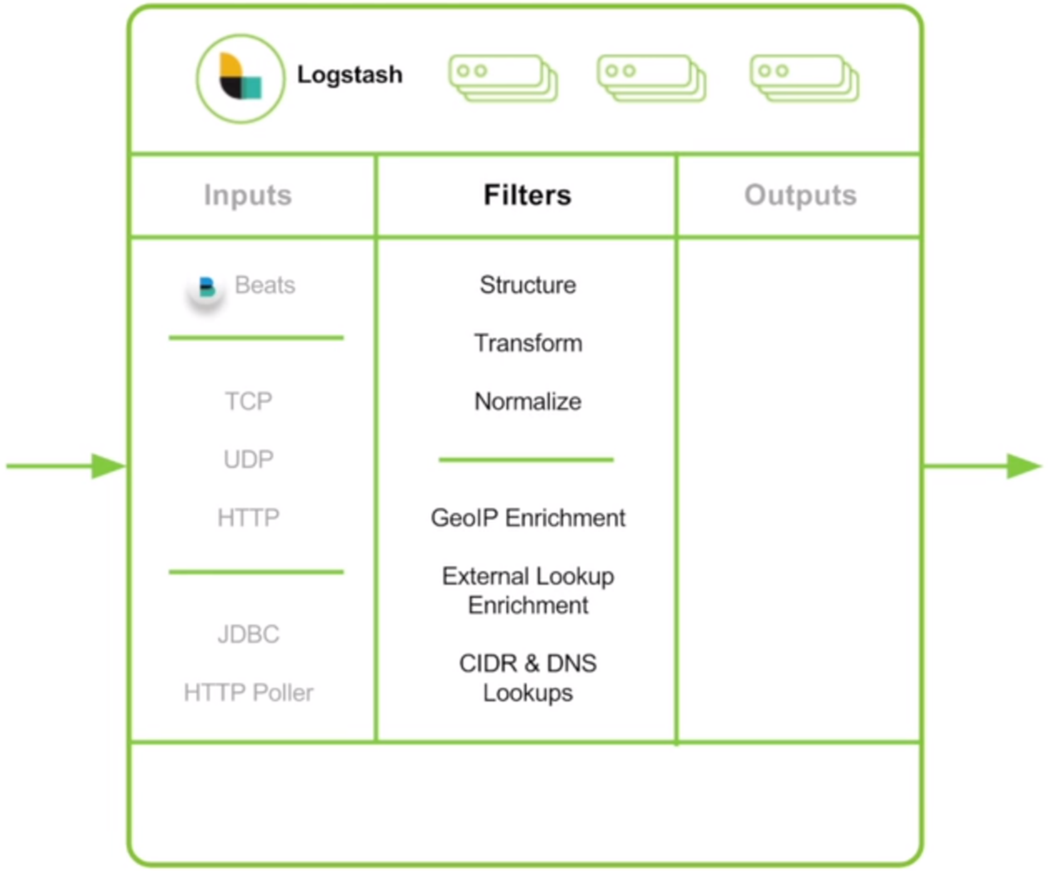


https://www.elastic.co/webinars/getting-started-logstash

Logstash Data Processing

- Filter plugins

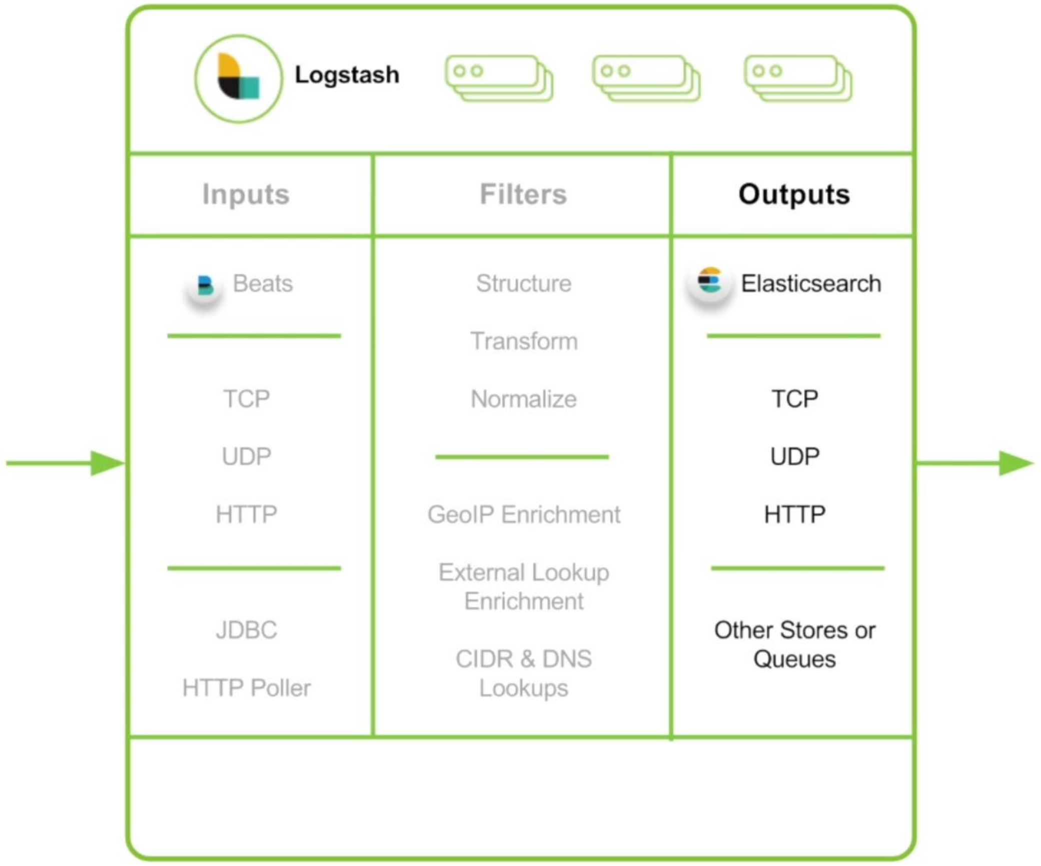
- Structure,
- Transform, and
- Enrich data



https://www.elastic.co/webinars/getting-started-logstash

Logstash Data Processing

- Emit (i.e. output) data to ElasticSearch or other destinations
- Use of output plugins



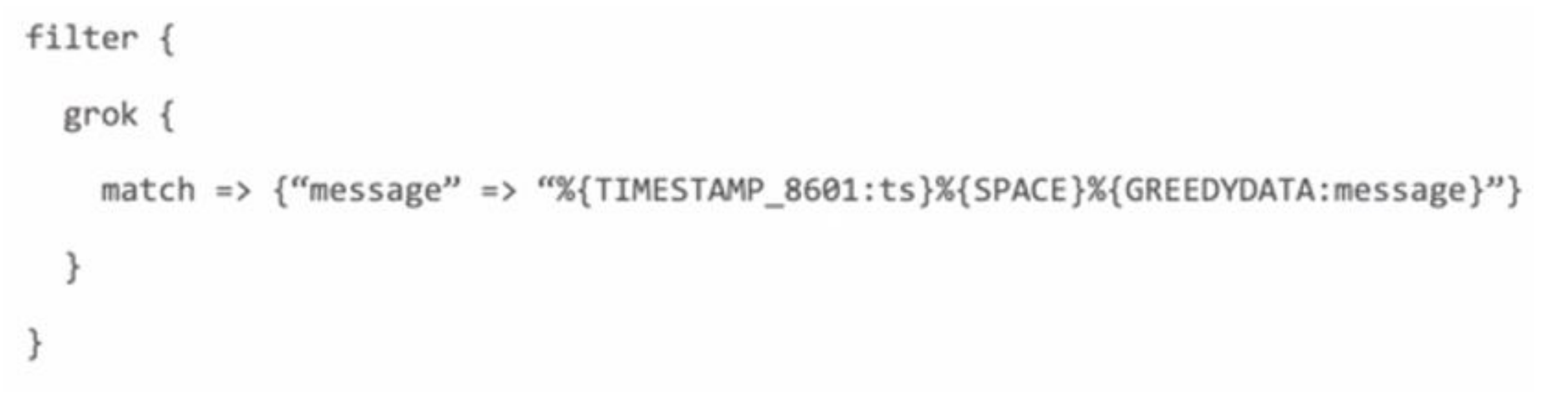
https://www.elastic.co/webinars/getting-started-logstash

Logstash filtering

- Grok filter – for parsing fields
- Date filter
- Dissect filter – like Grok, but minimal
- KV filter – key-value pair

Filter #1: Grok filter

- Field parsing filter

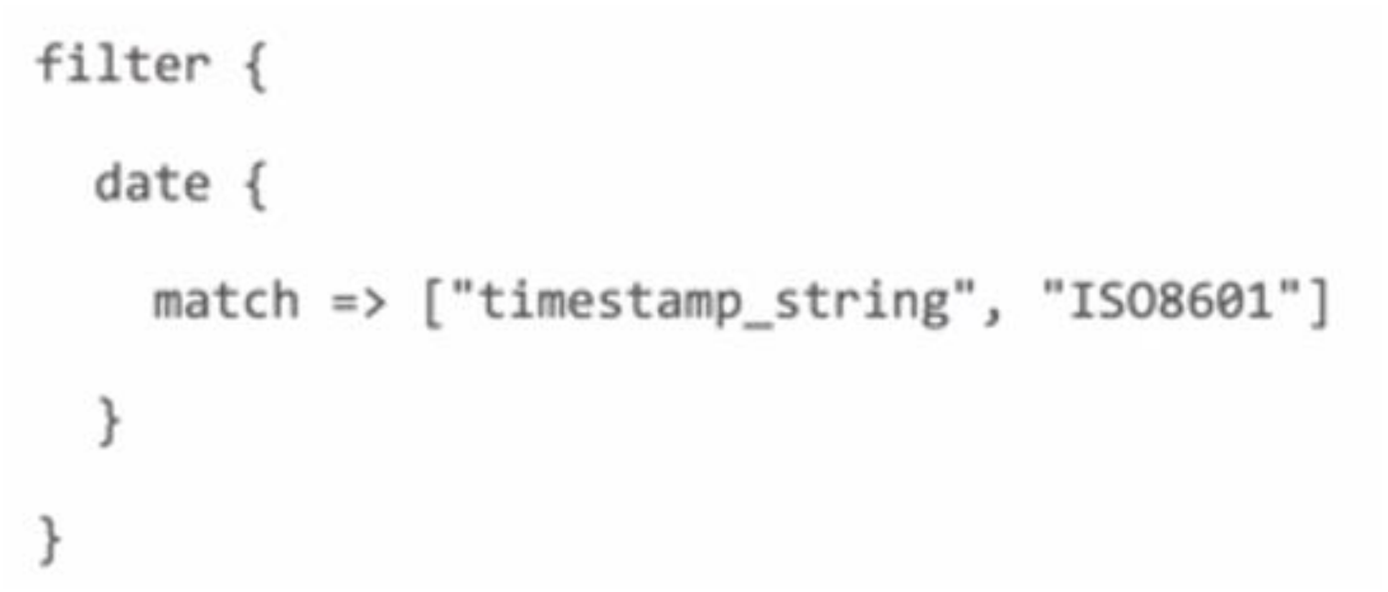


```
filter {
  grok {
    match => {“message” => “%{TIMESTAMP_8601:ts}%{SPACE}%{GREEDYDATA:message}”}
  }
}
```

https://www.elastic.co/webinars/getting-started-logstash

Filter #2: Date filter

- User strings for setting @timestamp based on



```
filter {
  date {
    match => ["timestamp_string", "ISO8601"]
  }
}
```

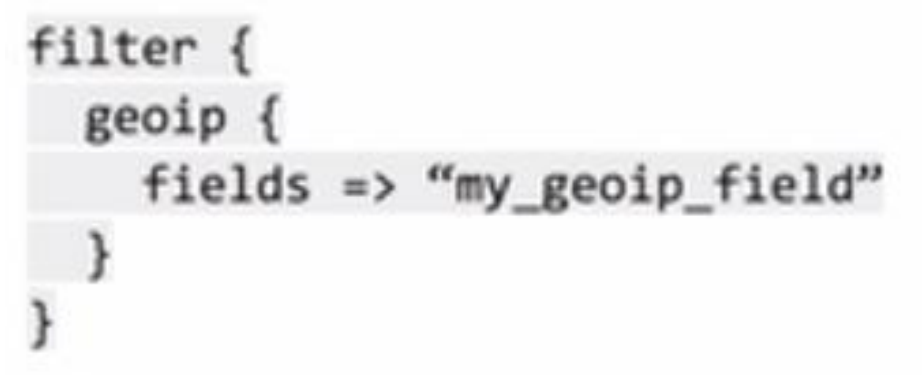
https://www.elastic.co/webinars/getting-started-logstash

Enriching data in Logstash

- GeoIP filter: Enrich IP address information with geographical context
- DNS filter: Enrich hostname with DNS info
- User Agent filter: Enrich user agent (i.e. browser info)
- Translate filter: Translate numerical codes and/or foreign languages

Enrich #1: GeoIP filter

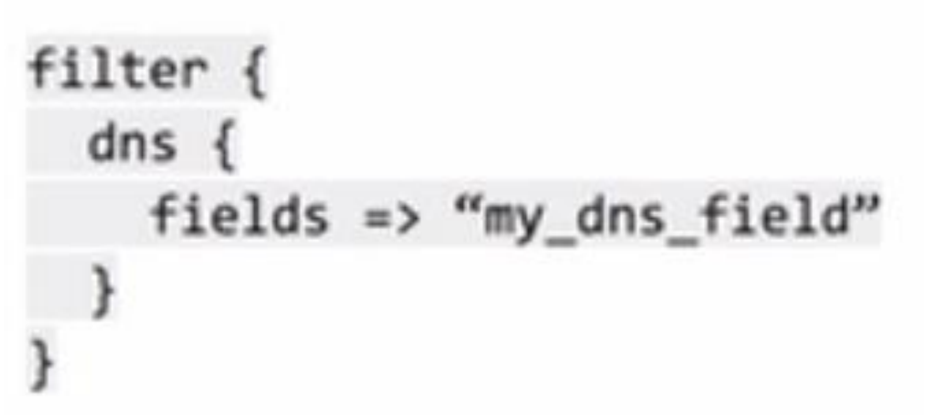
- Enrich IP address information



```
filter {
  geoip {
    fields => “my_geoip_field”
  }
}
```

Enrich #2: DNS filter

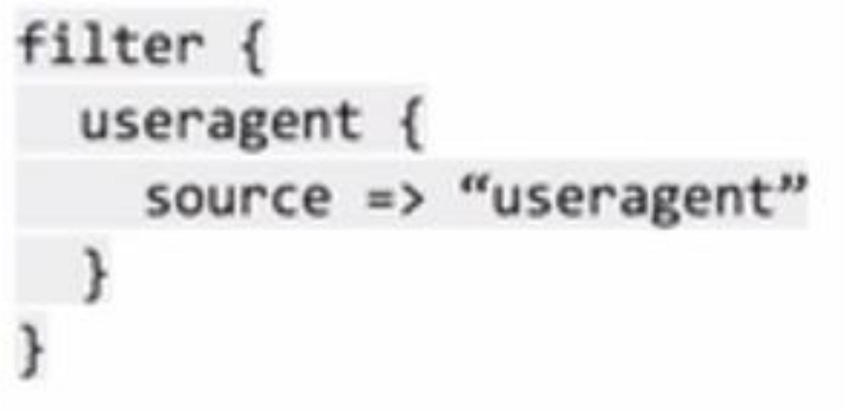
- Enrich hostname information with DNS info



```
filter {
  dns {
    fields => “my_dns_field”
  }
}
```

Enrich #3: User Agent filter

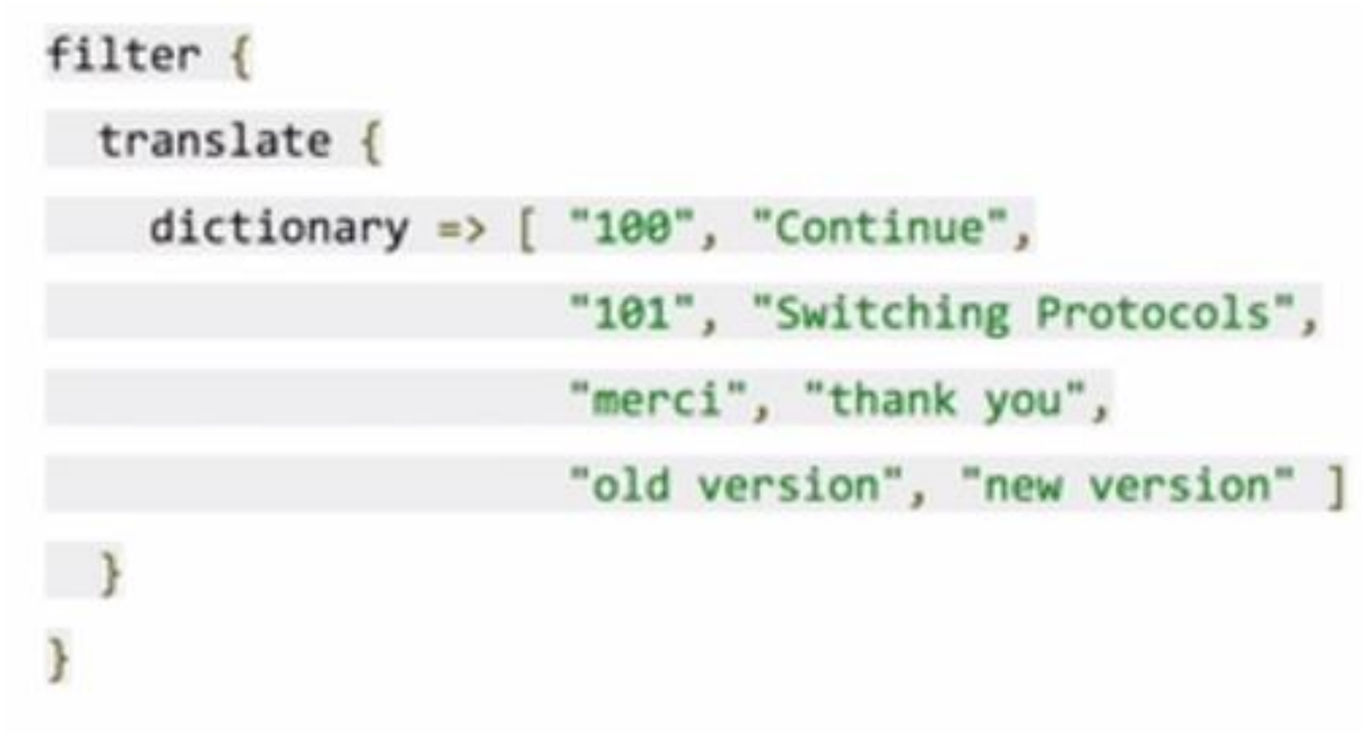
- Enrich browser user agent information



```
filter {
  useragent {
    source => “useragent”
  }
}
```

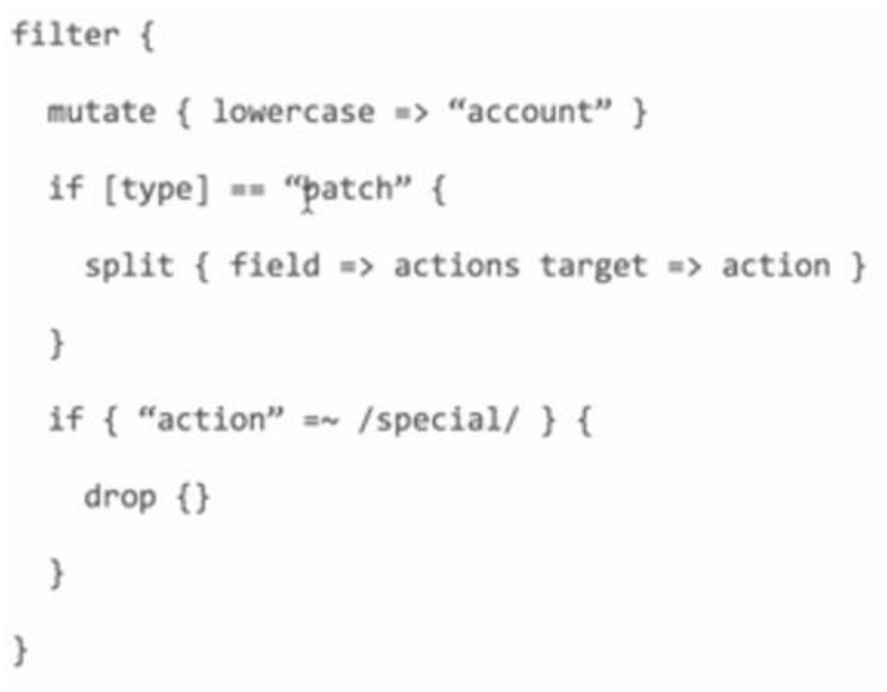
Enrich #4: Translate filter

- Use local data to map / enrich event descriptions



```
filter {
  translate {
    dictionary => [ "100", "Continue",
                    "101", "Switching Protocols",
                    "merci", "thank you",
                    "old version", "new version" ]
  }
}
```

Logstash data manipulation



```
filter {
  mutate { lowercase => "account" }
  if [type] == "batch" {
    split { field => actions target => action }
  }
  if { "action" =~ /special/ } {
    drop {}
  }
}
```

- Convert field types e.g. string → int
- Add, rename, replace, remove or copy fields
- (Upper/lower) Case transformations
- Join arrays
- Split fields → arrays
- Strip whitespace

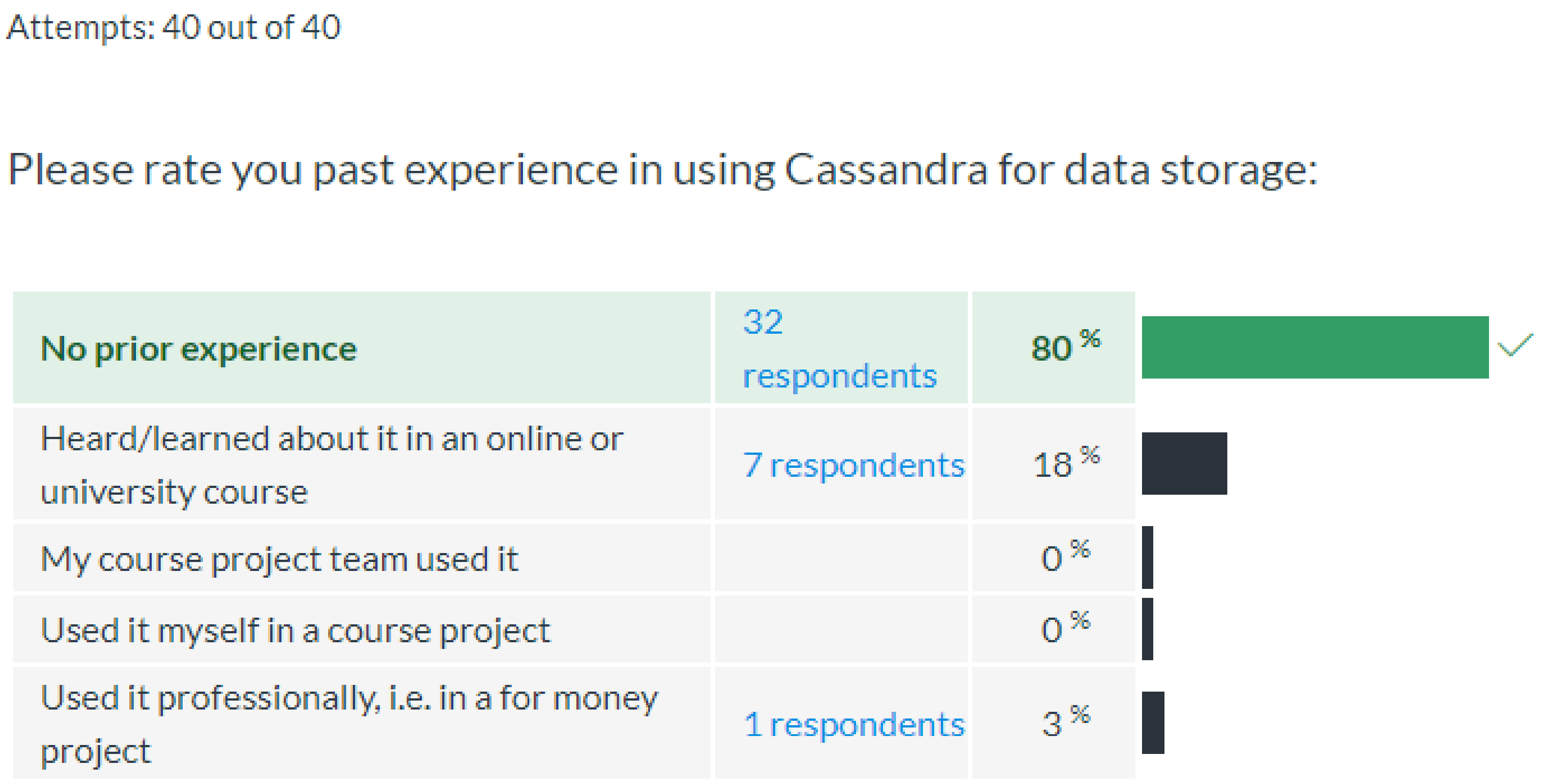
Cassandra

Cassandra in the timeline

	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020
MapReduce	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦						
Hadoop*		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
CouchDB			🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
Neo4j					▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Cassandra						🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪
MongoDB							🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧
Logstash								🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦

* Hadoop lives as a distributed data storage platform, not as data processor

Why Cassandra @ OST?



Attempts: 40 out of 40

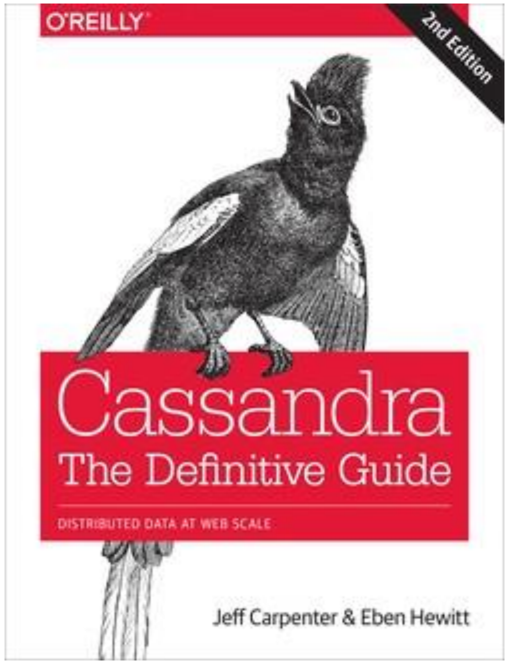
Please rate you past experience in using Cassandra for data storage:

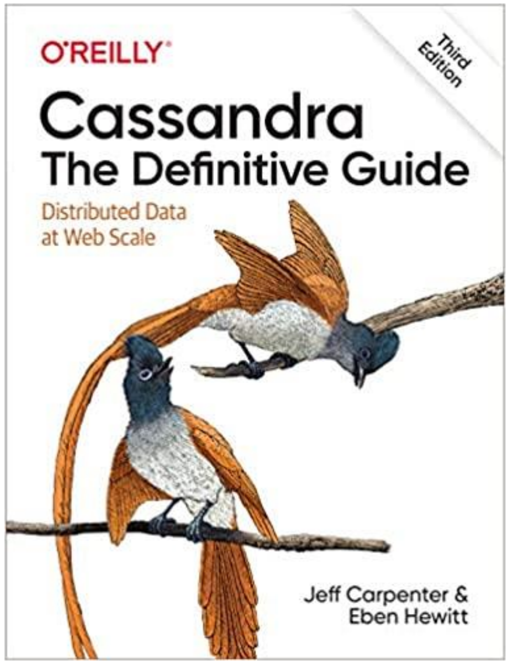
No prior experience	32 respondents	**80 %**	✓
Heard/learned about it in an online or university course	7 respondents	18 %	
My course project team used it		0 %	
Used it myself in a course project		0 %	
Used it professionally, i.e. in a for money project	1 respondents	3 %	

Key source(s)

O’Reilly: Cassandra Ed 2

O’Reilly: Cassandra Ed 3





Introduction

- "Apache Cassandra is an open-source, distributed, decentralized, elastically scalable, highly available, fault-tolerant, tunably consistent, column-oriented database that bases its distribution design on Amazon's dynamo and its data model on Google's Big Table."
  - Clearly, it is buzz-word compliant!!
- DEF: A Decentralized Structured Storage System
- Key features
  - Partitioning
  - Replication
  - Cluster management

Key RDBMS challenges

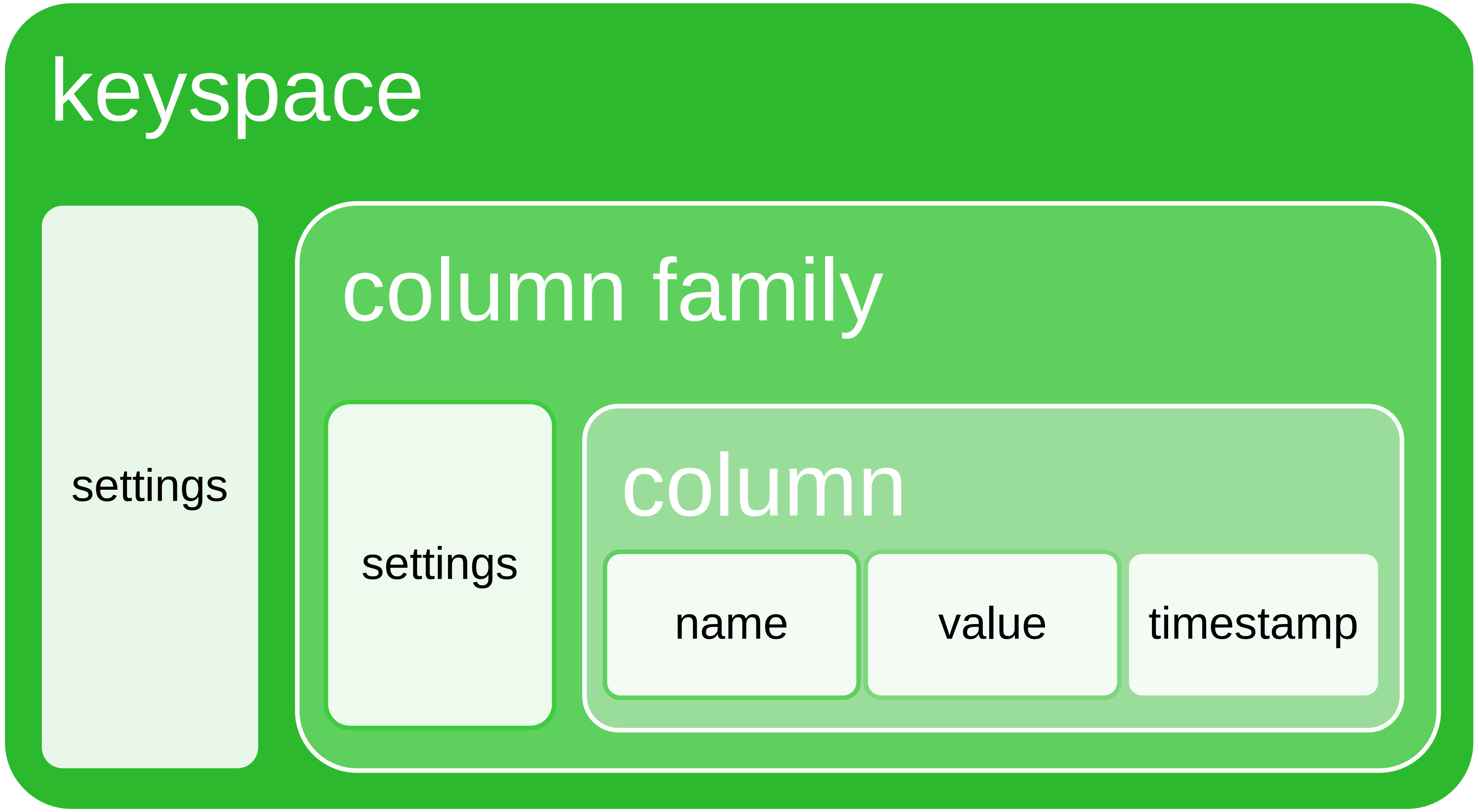
1. **Scalability issues** → what if the database (content) becomes so large that it can not be stored on the owner's own hardware & what if there are so many users that an RDBMS can not answer queries with reasonable cost in time & resources
2. **Joins are slow** → each RDBMS, even the smaller ones rely on joins which are slow → users start queueing up if the load is heavy
3. **Transactions are synchronous** → this in essence means that transactions modify multiple atomic pieces of data which are blocked during the transaction → not available to (other) users.

Typical solutions

- ‘Throw **more hardware** at the problem’ by adding more memory, faster CPUs or disks
  - More CPUs and disks cause scalability challenges → replication & consistency issues (might) arise
- **Custom optimizations**, e.g. SSD drives, faster communication channels, turn off unnecessary RDBMS features (journaling?)
- **Restructure the data model** → modify the model to better support high throughput at the cost of lower clarity and higher complexity
  - This often results in de-normalization, i.e. creating a (data) sub-optimal model with duplicates
- **Application optimizations** → attention put on the apps we build, improve the indices (i.e. plural of index), modify the app source code
- **Introduce a caching layer** → in-memory data with yet more added complexity, e.g. Redis

- **NOTE:** All the above might not be sufficient in intensive operations

Data Model



* Figure from Eben Hewitt's (author of Oreilly's Cassandra book) slides.

Partitioning

- Nodes are logically structured in Ring Topology.
- Hashed value of key associated with data partition is used to assign it to a node in the ring.
- Hashing rounds off after certain value to support ring structure.

- Lightly loaded nodes moves position to alleviate highly loaded nodes.

Replication

- Each data item is replicated at N (replication factor) nodes.

- Different Replication Policies
  - Rack Unaware – replicate data at N-1 successive nodes after its coordinator
  - Rack Aware – uses ‘Zookeeper’ to choose a leader which tells nodes the range they are replicas for
  - Datacenter Aware – similar to Rack Aware but leader is chosen at Datacenter level instead of Rack level.

Cluster Management (CM)

Gossip protocol

- DEF: Gossip protocols are network communication protocols inspired by real life rumor spreading
- Periodic, pairwise, inter-node communication.
- Low frequency communication ensures low cost.
- Random selection of peers.

Cassandra CM

- Uses Scuttleback (a Gossip protocol) to manage nodes.
- Uses gossip for node membership and to transmit system control state.

Basic Idea: Key-Value Store

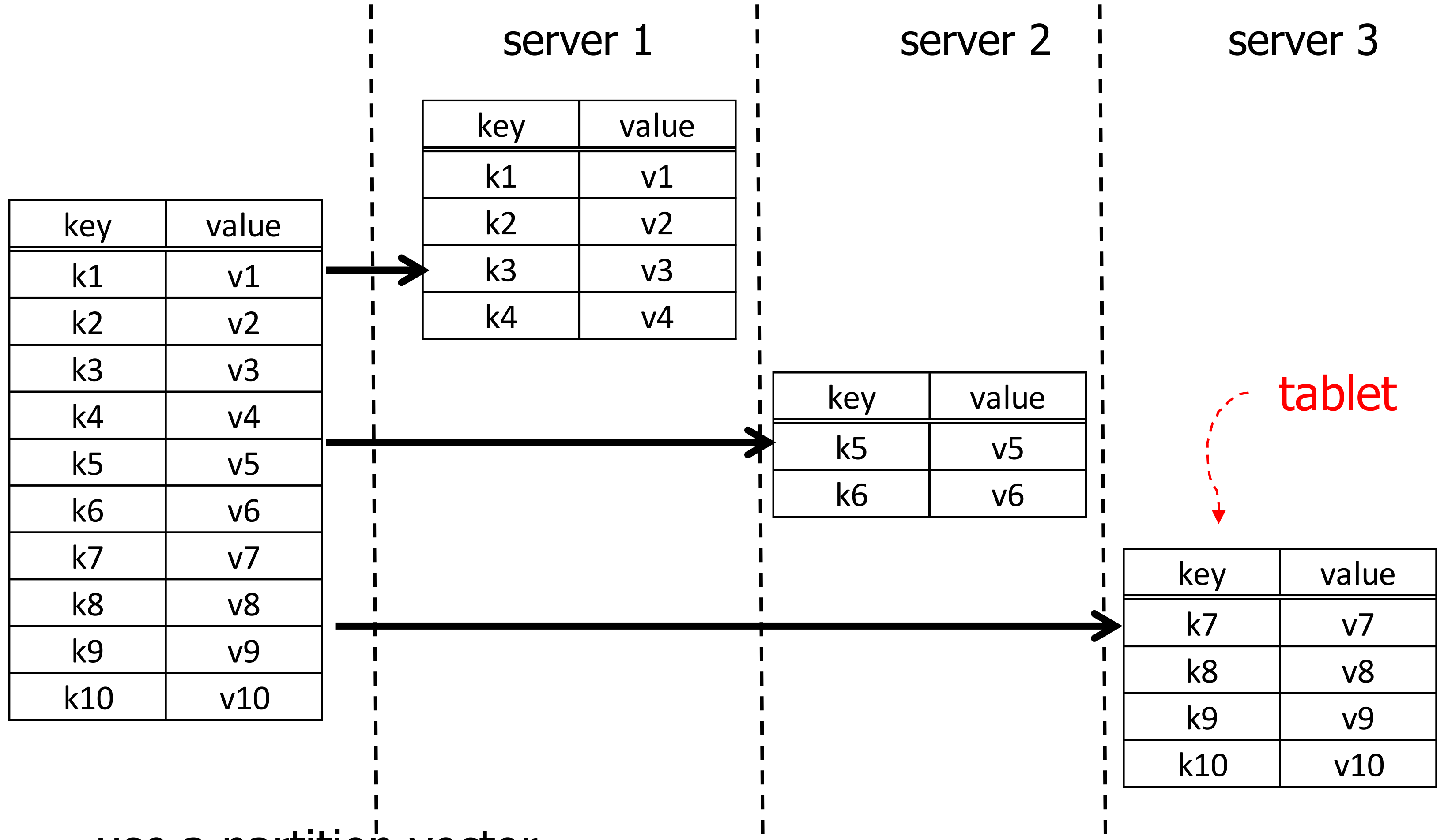
Table T:

key	value
k1	v1
k2	v2
k3	v3
k4	v4

keys are sorted

- API:
  - lookup(key) $\rightarrow$ value
  - lookup(key range) $\rightarrow$ values
  - getNext $\rightarrow$ value
  - insert(key, value)
  - delete(key)
- Each row is timestamp-ed
- Single row actions atomic
  (but not persistent in some systems?)
- No multi-key transactions
- No query language!

Fragmentation (Sharding)



key	value
k1	v1
k2	v2
k3	v3
k4	v4
k5	v5
k6	v6
k7	v7
k8	v8
k9	v9
k10	v10

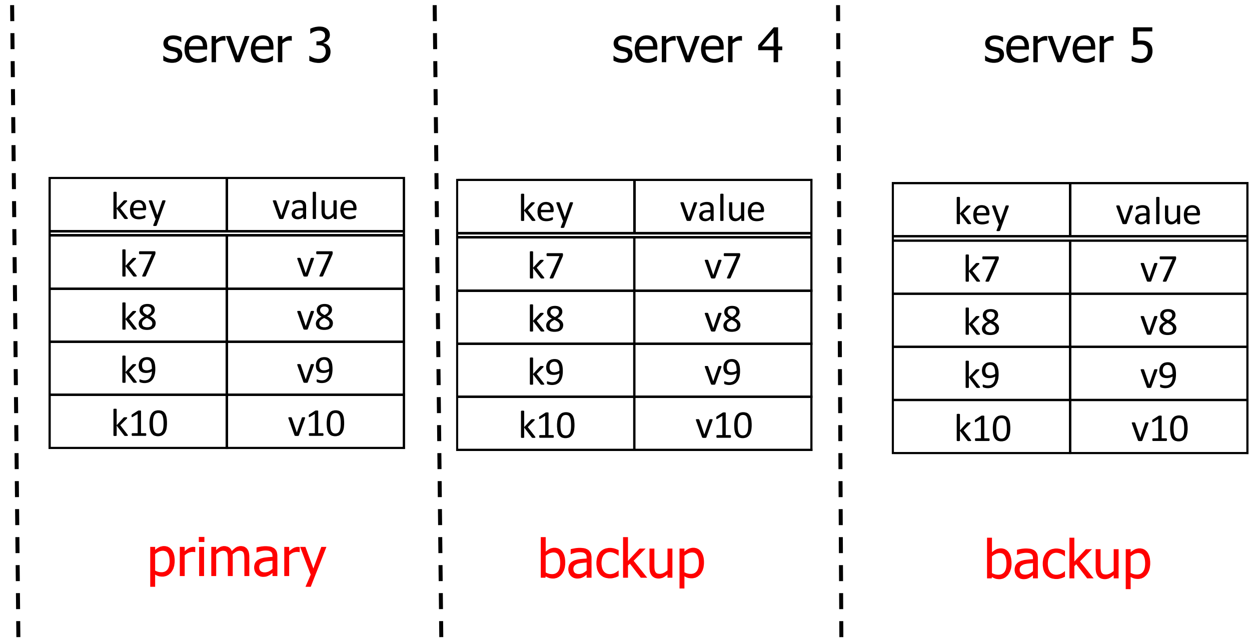
key	value
k1	v1
k2	v2
k3	v3
k4	v4

key	value
k5	v5
k6	v6

key	value
k7	v7
k8	v8
k9	v9
k10	v10

- use a partition vector
- "auto-sharding": vector selected automatically

Tablet Replication



key	value
k7	v7
k8	v8
k9	v9
k10	v10

key	value
k7	v7
k8	v8
k9	v9
k10	v10

key	value
k7	v7
k8	v8
k9	v9
k10	v10

- Cassandra:
  Replication Factor (# copies)
  R/W Rule: One, Quorum, All
  Policy (e.g., Rack Unaware, Rack Aware, ...)
  Read all copies (return fastest reply, do repairs if necessary)

Tablet Internals

key	value
k3	v3
k8	v8
k9	delete
k15	v15

memory

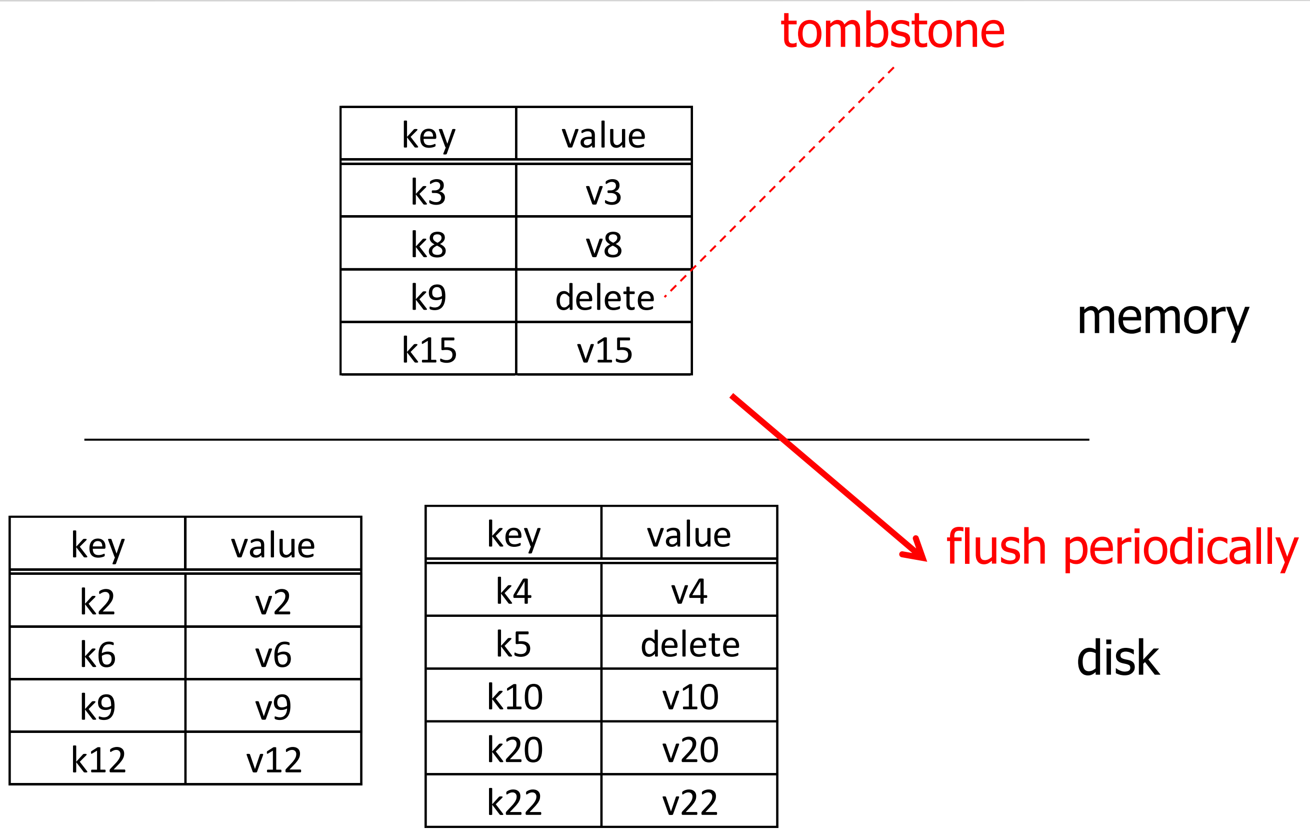
key	value
k2	v2
k6	v6
k9	v9
k12	v12

key	value
k4	v4
k5	delete
k10	v10
k20	v20
k22	v22

disk

Design Philosophy (?): Primary scenario is where all data is in memory.
Disk storage added as an afterthought

Tablet Internals



key	value
k3	v3
k8	v8
k9	delete
k15	v15

key	value
k2	v2
k6	v6
k9	v9
k12	v12

key	value
k4	v4
k5	delete
k10	v10
k20	v20
k22	v22

- tablet is merge of all segments (files)
- disk segments imutable
- writes efficient; reads only efficient when all data in memory
- periodically reorganize into single segment

Column Family

K	A	B	C	D	E
k1	a1	b1	c1	d1	e1
k2	a2	null	c2	d2	e2
k3	null	null	null	d3	e3
k4	a4	b4	c4	e4	e4
k5	a5	b5	null	null	null

Column Family

K	A	B	C	D	E
k1	a1	b1	c1	d1	e1
k2	a2	null	c2	d2	e2
k3	null	null	null	d3	e3
k4	a4	b4	c4	e4	e4
k5	a5	b5	null	null	null

- for storage, treat each row as a single “super value”
- API provides access to sub-values
  (use family:qualifier to refer to sub-values
  e.g., price:euros, price:dollars)
- Cassandra allows “super-column”:
  two level nesting of columns
  (e.g., Column A can have sub-columns X & Y)

Vertical Partitions

K	A	B	C	D	E
k1	a1	b1	c1	d1	e1
k2	a2	null	c2	d2	e2
k3	null	null	null	d3	e3
k4	a4	b4	c4	e4	e4
k5	a5	b5	null	null	null

can be <u>manually</u> implemented as

K	A
k1	a1
k2	a2
k4	a4
k5	a5

K	B
k1	b1
k4	b4
k5	b5

K	C
k1	c1
k2	c2
k4	c4

K	D	E
k1	d1	e1
k2	d2	e2
k3	d3	e3
k4	e4	e4

Vertical Partitions

K	A	B	C	D	E
k1	a1	b1	c1	d1	e1
k2	a2	null	c2	d2	e2
k3	null	null	null	d3	e3
k4	a4	b4	c4	e4	e4
k5	a5	b5	null	null	null

K	A
k1	a1
k2	a2
k4	a4
k5	a5

K	B
k1	b1
k4	b4
k5	b5

K	C
k1	c1
k2	c2
k4	c4

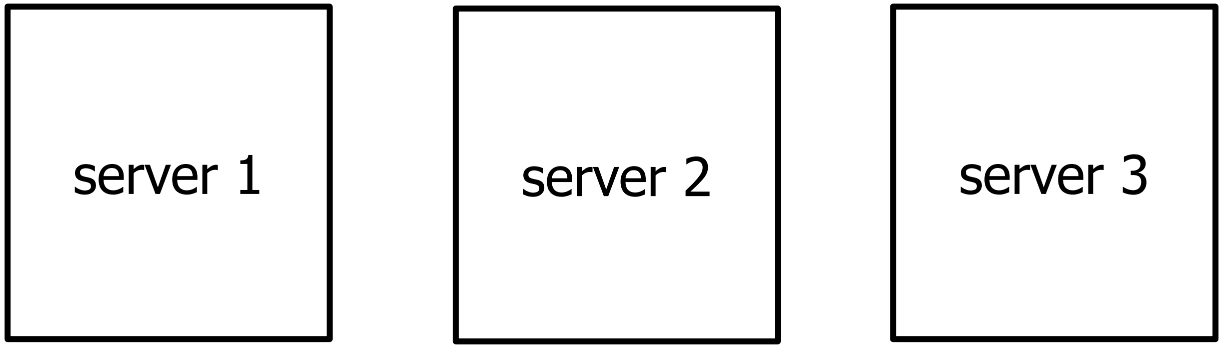
column family

K	D	E
k1	d1	e1
k2	d2	e2
k3	d3	e3
k4	e4	e4

- good for sparse data;
- good for column scans
- not so good for tuple reads
- are atomic updates to row still supported?
- API supports actions on full table; mapped to actions on column tables
- API supports column "project"
- To decide on vertical partition, need to know access patterns

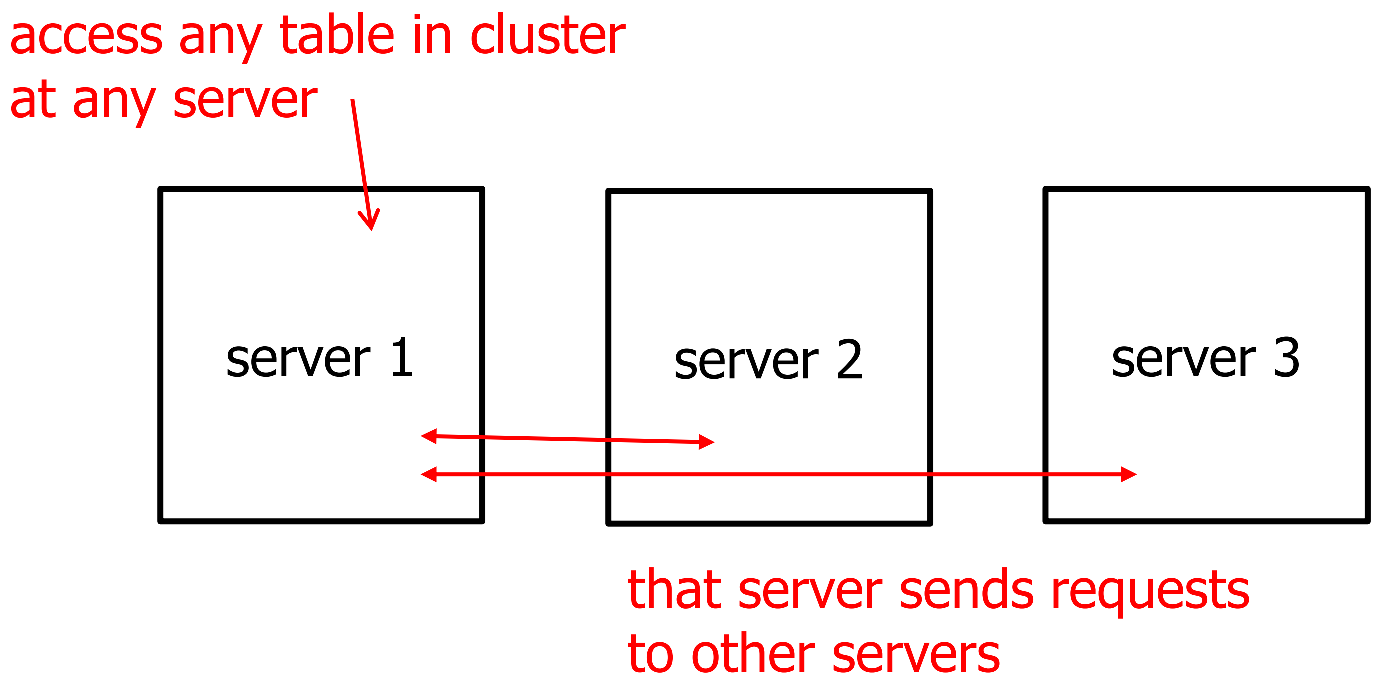
Failure recovery (Cassandra)

- No master node, all nodes in “cluster” equal



Failure recovery (Cassandra)

- No master node, all nodes in “cluster” equal



Cassandra vs MySQL

MySQL

- > 50 GB Data
- Writes Average : ~300 ms
- Reads Average : ~350 ms

Cassandra

- > 50 GB Data
- Writes Average : 0.12 ms
- Reads Average : 15 ms

- Stats calculated on Facebook data

Who uses Cassandra?

- Originally designed at Facebook
- Open-sourced
- Some of its myriad users:

Adobe

AppScale

IBM

Symantec

twitter

ebay

PBS KIDS

hp

Spotify

NETFLIX

ERICSSON

Summary

- **MongoDB** is a general purpose, document-based, distributed database
- **Cassandra** is a decentralized structured storage system
- **Logstash** is the log management system element of the ELK stack
- Coming up next:
  - Hadoop Distributed Filesystem (**HDFS**)
  - **HBase** non-relational database, (usually) runs on top of HDFS

Thank you for your attention!