

Failures and Resiliency Principles of Distributed Computing

course 2024
arnaud-nauwynck@gmail.com

This document:
[https://github.com/Arnaud-Nauwynck/presentations/pres-bigdata/
3-failure-and-resiliency-pinciples-distributed-computing](https://github.com/Arnaud-Nauwynck/presentations/pres-bigdata/3-failure-and-resiliency-pinciples-distributed-computing)

MTBF

M.T.B.F = Mean Time Between Failure

For HDD ~ 500 000 hours

≥ 50 years

May look good at home, to save your data

(Mean average may be smaller/larger)

MTBF « at Scale »

In DataCenter with 10 000 servers x 4 disks

=> 1 failure every 1h 15mn

= 19 failures per day

becomes a recurrent task / job



When Failure(s) Happens ?

=> Program fails ?

=> System fails ?

=> need relaunch manually ?

=> diagnostic hardware/software ?

=> interrupt all, repair ?



Studying « When Failure » 1/5

=> Program fails ?

=> System fails ?

=> need relaunch manually ?

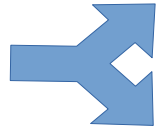
=> diagnostic hardware/software ?

=> interrupt all, repair ?

When Failure..

=> Program fails ?

Individual Component : Obviously FAIL



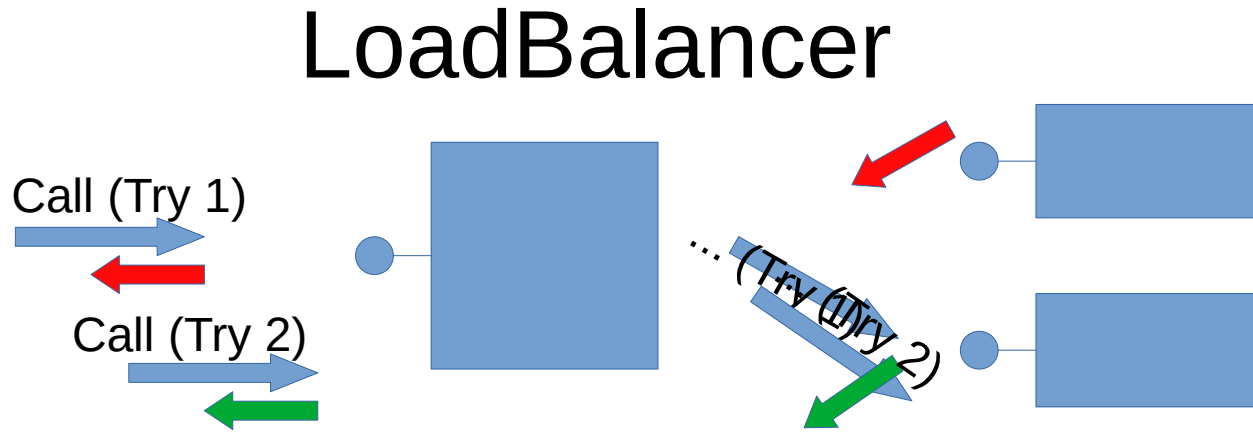
Distributed Architecture : RESILIENT

resist (hopefully) to (some) failure

Failure is not « Exceptional »
.. is a « normal » path
consider it everywhere in code

```
public void doSomething() throws Exception {  
    int maxRetry = 5;  
    for(int retry = 0; retry < maxRetry; retry++) {  
        try {  
            anyTreatmentCanFail();  
            break; // success!  
        } catch(Exception ex) {  
            Log.warn("Failed .. retry " + retry + "/" + maxRetry, ex);  
            sleep(100 * Math.pow(2, maxRetry)); // wait a little  
        }  
    }  
}
```

Retrying with a LoadBalancer .. might just work

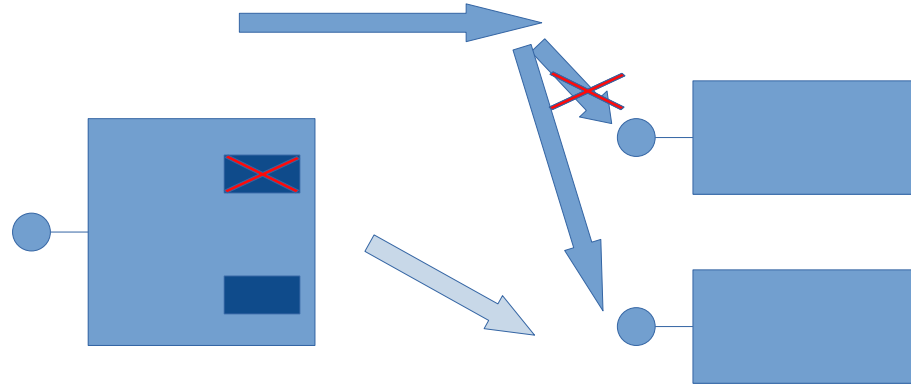


Dispatch to underlying servers
Using Round-Robin

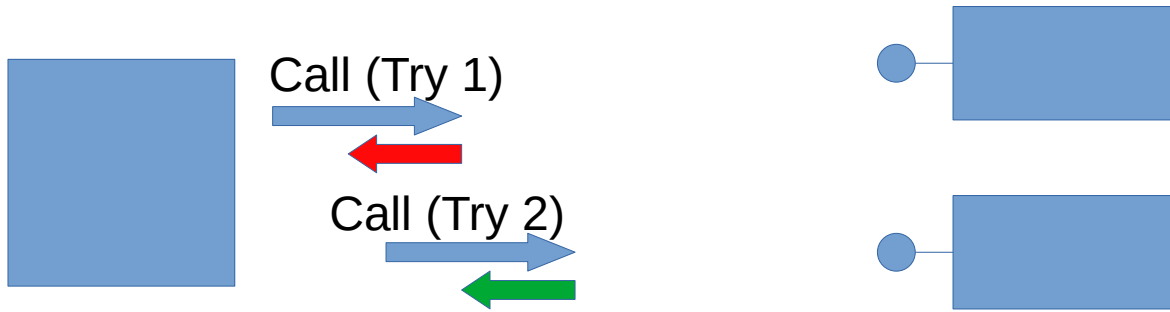
Server Health Check

Temporary evict from RoundRobin Pool

Health Check



Client-Side « LB »



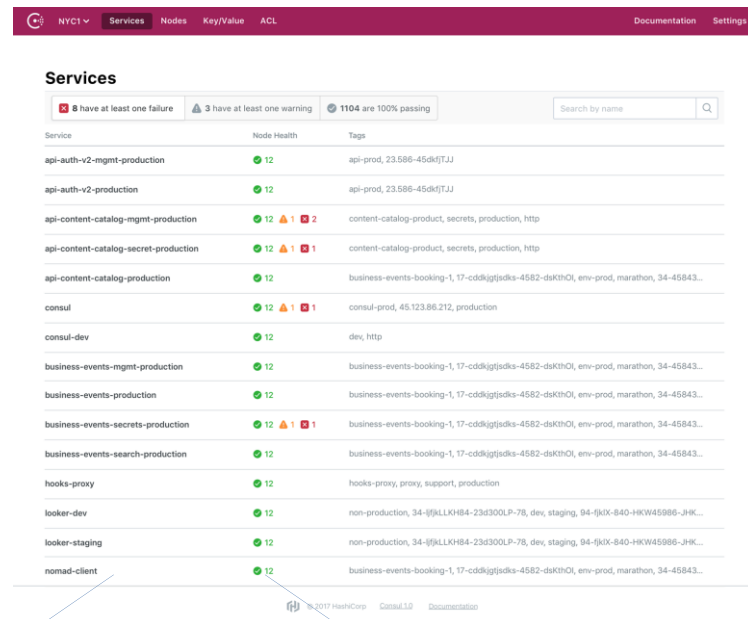
NO « LB »

Need discovery mechanism on client

Service >= Nodes

Examples of Service Discovery

DNS, ServiceMesh,
Zookeeper, Consul.io,
HAProxy, Kubernetes, ..



The screenshot shows the Consul.io web interface. At the top, there's a navigation bar with 'Services' selected. Below it, a summary bar indicates '8 have at least one failure', '3 have at least one warning', and '1104 are 100% passing'. The main content is a table with columns for Service, Node Health, and Tags. The table lists various services like 'api-auth-v2-mgmt-production', 'api-content-catalog-mgmt-production', etc., along with their health status (green circle for healthy, yellow triangle for warning, red square for failure) and the number of nodes (12 for most). Tags are also listed for each service.

Service	Node Health	Tags
api-auth-v2-mgmt-production	12	api-prod, 23.586-45dk[TJJ
api-auth-v2-production	12	api-prod, 23.586-45dk[TJJ
api-content-catalog-mgmt-production	12 1 2	content-catalog-product, secrets, production, http
api-content-catalog-secret-production	12 1 1	content-catalog-product, secrets, production, http
api-content-catalog-production	12	business-events-booking-1, 17-cddk(ggjsdks-4582-dsKHOI, env-prod, marathon, 34-45843...
consul	12 1 1	consul-prod, 45.123.86.212, production
consul-dev	12	dev, http
business-events-mgmt-production	12	business-events-booking-1, 17-cddk(ggjsdks-4582-dsKHOI, env-prod, marathon, 34-45843...
business-events-production	12	business-events-booking-1, 17-cddk(ggjsdks-4582-dsKHOI, env-prod, marathon, 34-45843...
business-events-secrets-production	12 1 1	business-events-booking-1, 17-cddk(ggjsdks-4582-dsKHOI, env-prod, marathon, 34-45843...
business-events-search-production	12	business-events-booking-1, 17-cddk(ggjsdks-4582-dsKHOI, env-prod, marathon, 34-45843...
hooks-proxy	12	hooks-proxy, proxy, support, production
looker-dev	12	non-production, 34-@LLKH84-23d300LP-78, dev, staging, 94-9kx-840-HKW45886-JHK...
looker-staging	12	non-production, 34-@LLKH84-23d300LP-78, dev, staging, 94-9kx-840-HKW45886-JHK...
nomad-client	12	business-events-booking-1, 17-cddk(ggjsdks-4582-dsKHOI, env-prod, marathon, 34-45843...

Consul.io Services

Nodes

Studying « When Failure » 2/5

=> Program fails ?

=> System fails ?

=> need relaunch manually ?

=> diagnostic hardware/software ?

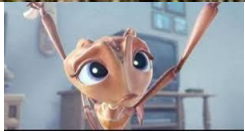
=> interrupt all, repair ?

System = Union of independent
components

Each component can be killed
... The system must survive each failure

Be Confident in « System »

Walk on a « Ant » ...



... « Anthill » not in danger



How to Check System does not « Fail » ?

Very difficult to « prove » system correctness

Easy to test : « kill and see »

Not an exhaustive test... Repeat + Changes

Test Kill with Chaos Engineering

Initiated by Netflix

Goal : Randomly kill Process / VM / Datacenter



SLA : 99.99 Up-time ?



= Daily: 8s

Weekly: 1m 0s

Monthly: 4m 22s

Quarterly: 13m 8s

Yearly: 52m 35s

3 nines : 99.999 = Yearly : 5mn15s

4 nines : 99.9999 = Yearly : 31s

Studying « When Failure » 3/5

=> Program fails ?

=> System fails ?

=> need relaunch manually ?

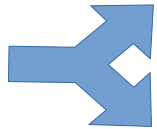
=> diagnostic hardware/software ?

=> interrupt all, repair ?

When Failure ...

=> need relaunch manually ?

Hard-Coded Launch to specific server.. fail



Auto Distributed: RESILIENT

don't launch manually on a specific server

Let the system select one

Pet vs Cattle

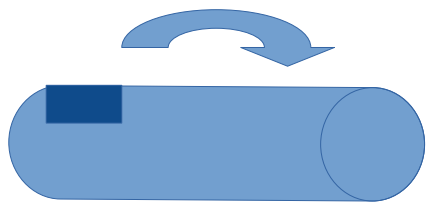


Pet have id=name
You take extra care of it



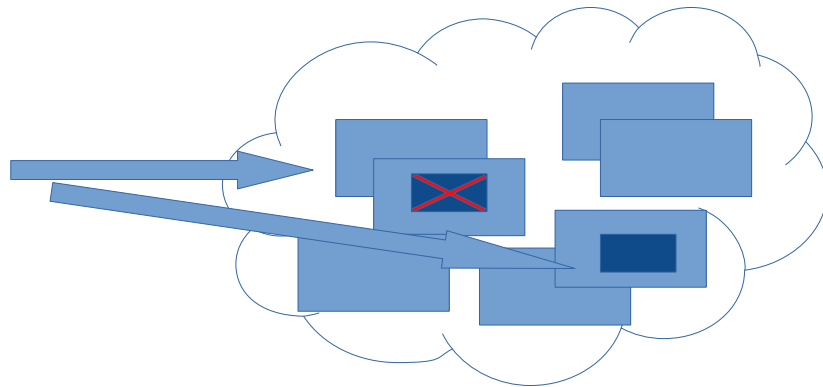
Cattle ≥ 1000 : anonymous
Id=Number, interchangeable

Launching => Scheduling on Allocated Resource



You submit
Something to run

You don't run yourself



Maybe wait for resource
Allocate resource
Try launch
retry

Studying « When Failure » 4/5

=> Program fails ?

=> System fails ?

=> need relaunch manually ?

=> diagnostic hardware/software ?

=> interrupt all, repair ?

When Failure ...

=> diagnostic hardware/software ?

NO ? ... difficult anyway

Bugs may exist

BUT failure is « not » a bug

Studying « When Failure » 5/5

=> Program fails ?

=> System fails ?

=> need relaunch manually ?

=> diagnostic hardware/software ?

=> interrupt all, repair ?

When Failure ...

=> interrupt all, repair ?

NO ...

Immutable-Infrastructure

Do not edit infra once created

Drop and re-create new VMs

When Failure ...

Hardware HOT-PLUG :
unplug old disk, and plug new one
Without interruption



When Failure ...

Idem for Software :
add/remove servers to cluster
(no hard-coded topology/configs)

Duplicate Failable Component for Fewer System Failure

If a component has 0.01 chance to Fail today

Adding another (independent) component...

=> Probability that 2 fail today = $0.01 * 0.01 = 0.0001$

=> Probability that 3 fail today = $0.01^3 = 0.000001 = 1e-6$

« If » your system still works with 1 working components out of 2 ... better

System \geq Component

If No Correlated / Dispatch / No Spof

Electric power : when fail ... all fails (correlated failures)

Need reliable network + dispatching to working components (retry/detect failed ones)

Everything between the components can be a « SPOF »

Network may be a « Single Point Of Failure »

as all forgotten components not redundant

Story of Arianne 501

Duplicated / « Correlated » Errors...



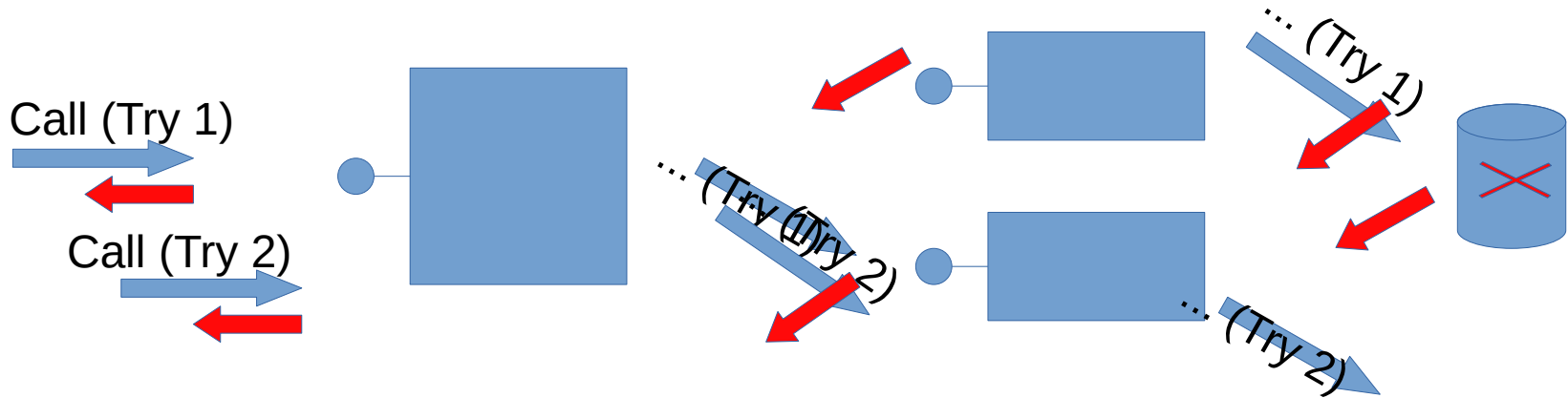
Flight computation performed twice in parallel on 2 isolated hardwares

But using same program...

Both programs throw same «overflow exception » at exact same millis

=> Lessons learned : use 2 hardwares + 2 independent softwares + ..

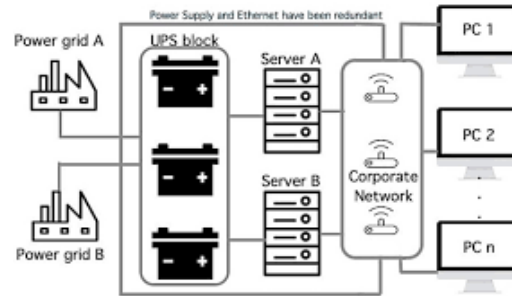
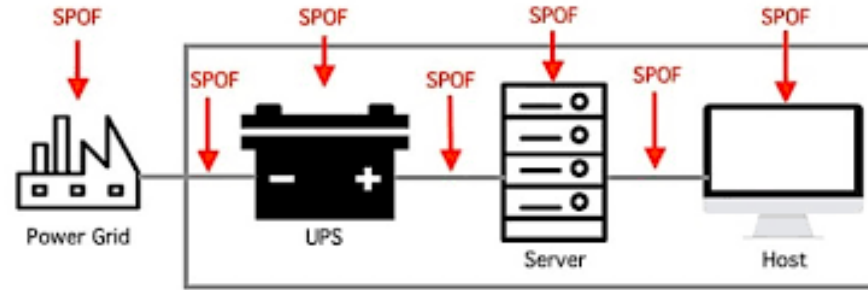
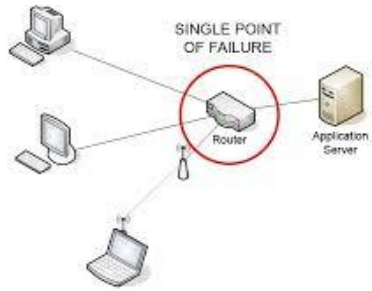
Single Point Of Failure ?



High Availability Servers...

But Shared State / Database .. SPOF

Examples of SPOF

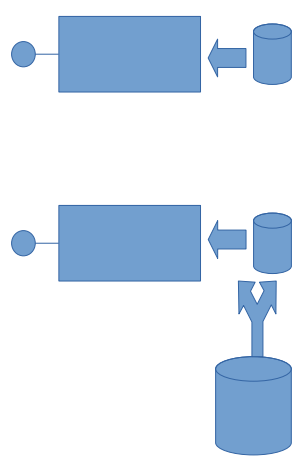


NO SPOF : Duplicate Everything

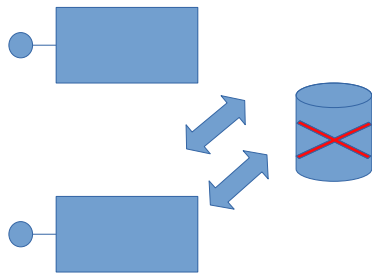
But how to duplicate a single Source of Truth « Data » ?

Copy => stale data / replication / distributed lock ?

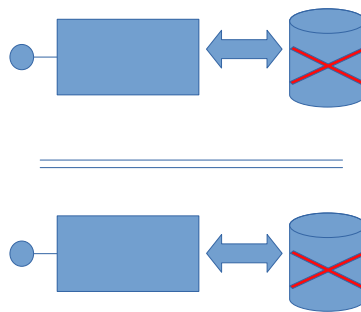
Stateless, Spof, Sharded (easy) vs Statefull (difficult)



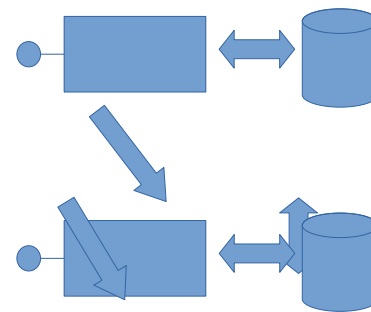
Stateless
(or stale Read-Only
cached data)



Spof



Sharded... 2 spofs
(ex : id modulo 2)
Horizontal scale : OK
But not replicated

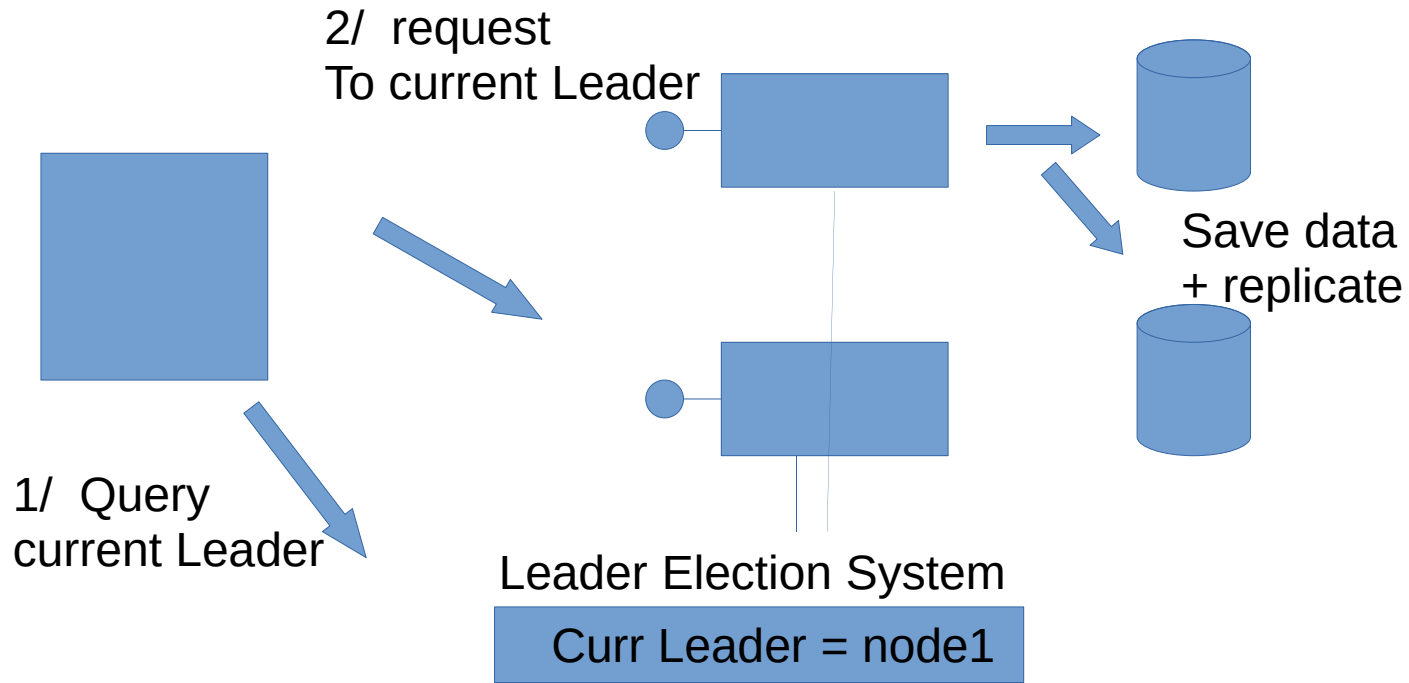


Statefull
Concurrent Distributed Data

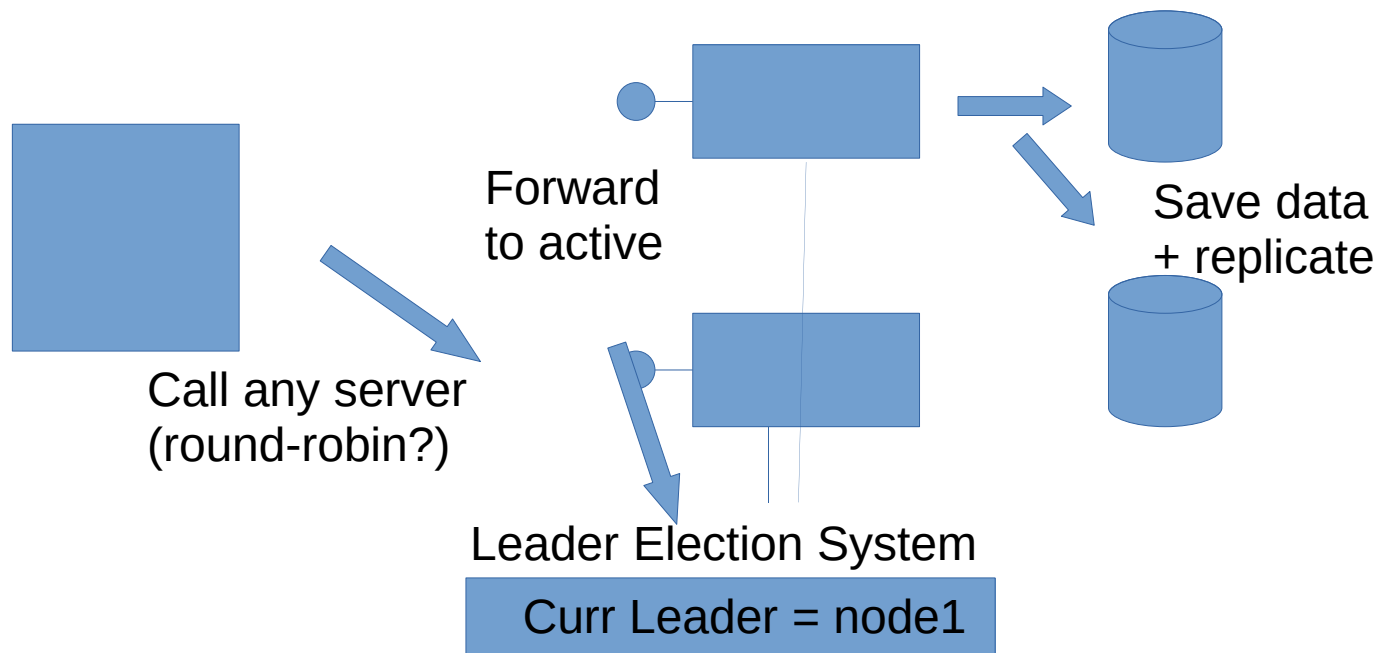
Distributed Lock ?
Replication ?
Source of truth ?

« HA » High Availability

Active – Standby + Replication



Transparent Delegate to Active



Synonym Terms...

Master – Slaves (not politically correct)

Leader – Followers

Active – StandBys

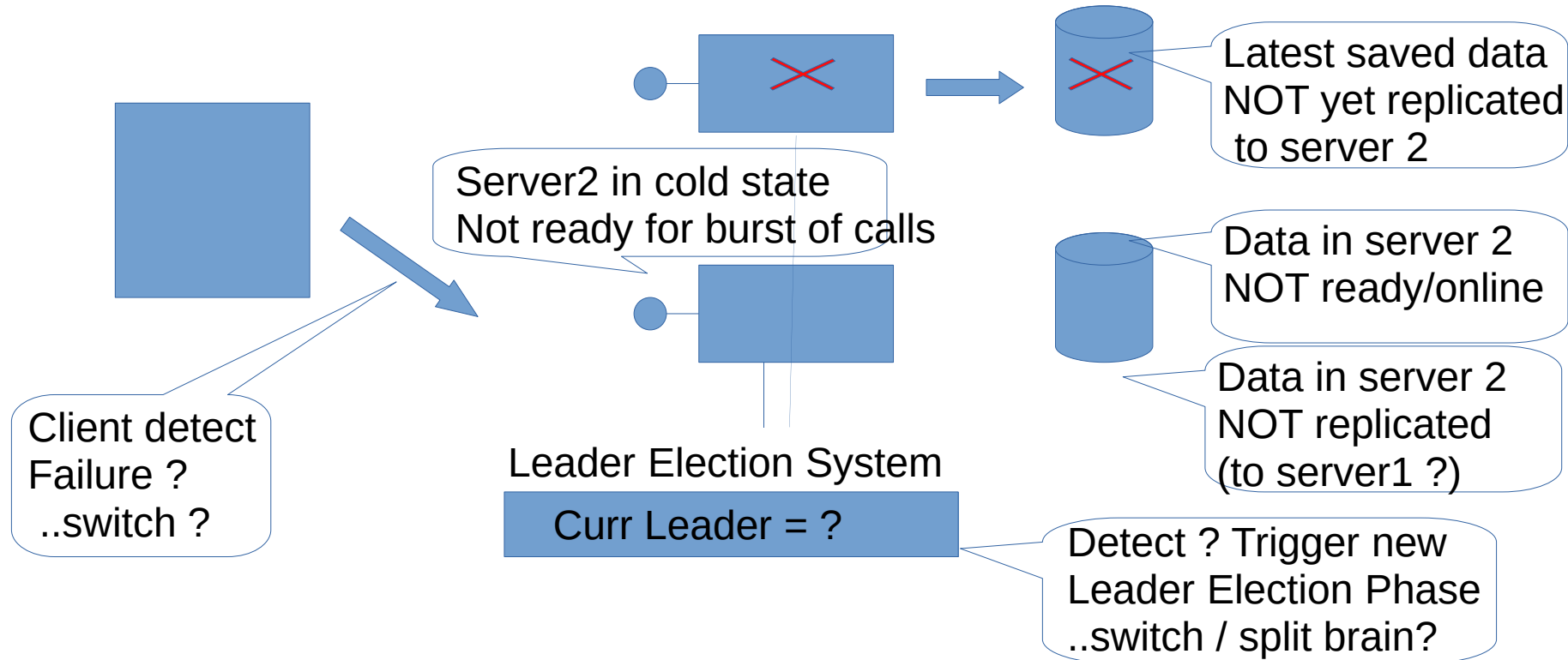
Primary – Secondaries

Main – Replica

MainSite - Disaster Recovery

MainDisk - BackupDisk

Switching from Active to StandBy



Problem with Leaders same as in Political

works only when there is exactly 1 leader

≥ 2 leaders ... conflicts / Split-Brains / Network Partitioning

0 Leader ... nothing works

Imposters pretend to be Leader

Leader staying leader too long after new election

Slow to organize election

Trust the result of election ?

Who can organize if there is no leader?

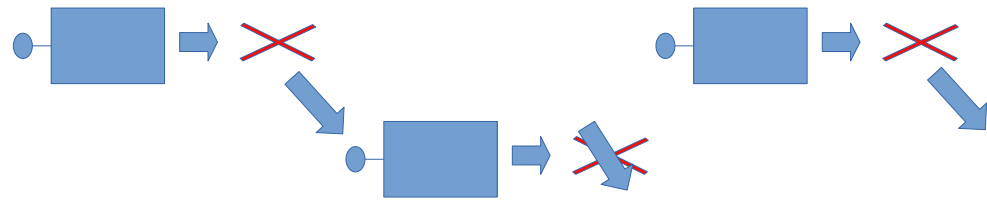
Can't decide election in 50 % / 50 % equality ... need an odd number : 3, 5, 7..

There must be at least 50 % participation (no Abstention)

Otherwise both candidates pretend to majority (not absolute majority) with 50 %

Leader fail to « Start Lead » ... re-electing too Often ?

Example of Catastrophic scenario :



1/ Server « N » become Leader

2/ It needs huge memory to serve requests

There are too many requests (burst of waiting requests)

3/ « Full GC » occurs, take $\geq 30s$, and « stop-the-world »

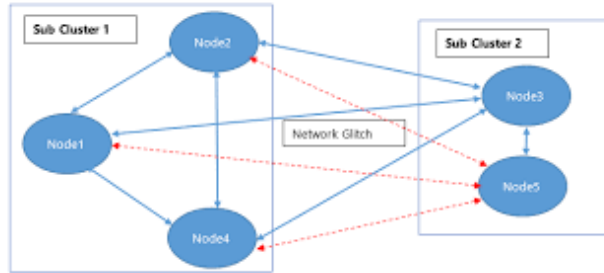
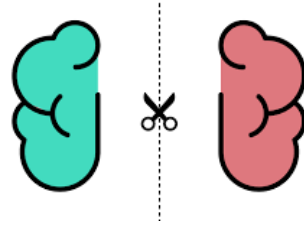
30s is the default Timeout for socket connection / read response

... During this time, Leader fails to answer to « Health check »

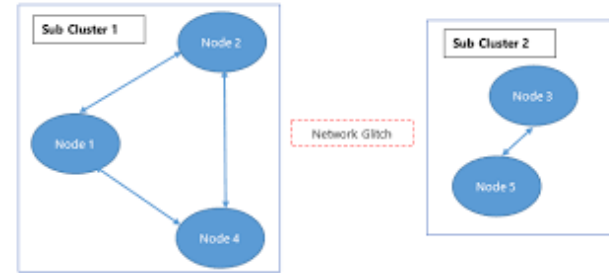
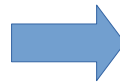
4/ Leader is considered not healthy => not leader any more

=> 5/ Electing another Leader « N+1 » => back to beginning 1/

Split Brain / Network Partitioning



1 cluster, 1 master
status=OK
... but network partition happens



2 clusters fully isolated each
Each have 1 master
Each have status=OK
... data diverge, will discover conflict on merge

Quorum of 50 %



Quorum of >50 % reaches => elect candidate1



NO Quorum of 50 % reaches
=> re-organize new election
(proba=0.5 that 2 of 3 vote for same)



Majority but NO Quorum
... high risk of Split-Brain
(all others may vote and you are not aware)
=> wait enough voters (re-organize / accept vote)

Waiting Quorum

= system « Not Available »

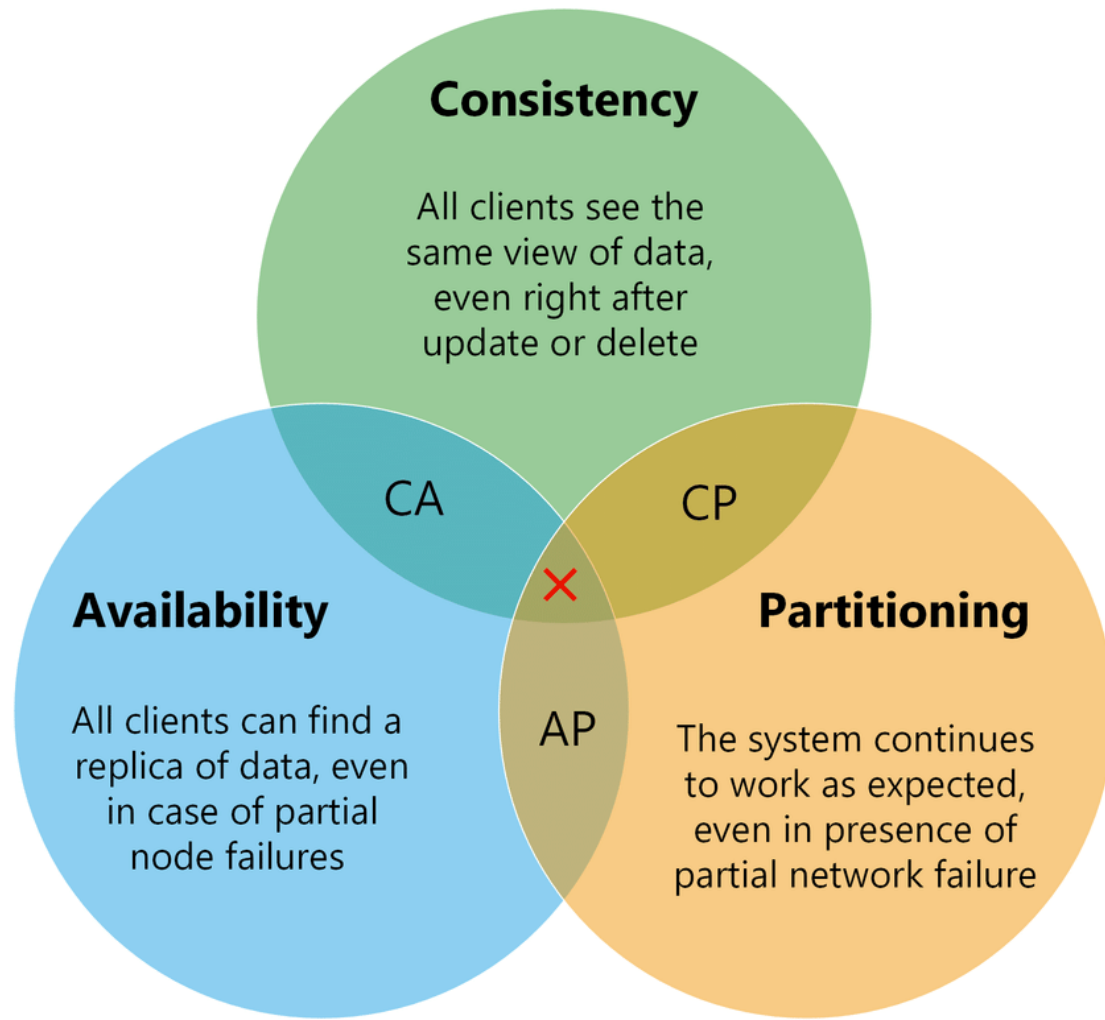


Waiting Quorum

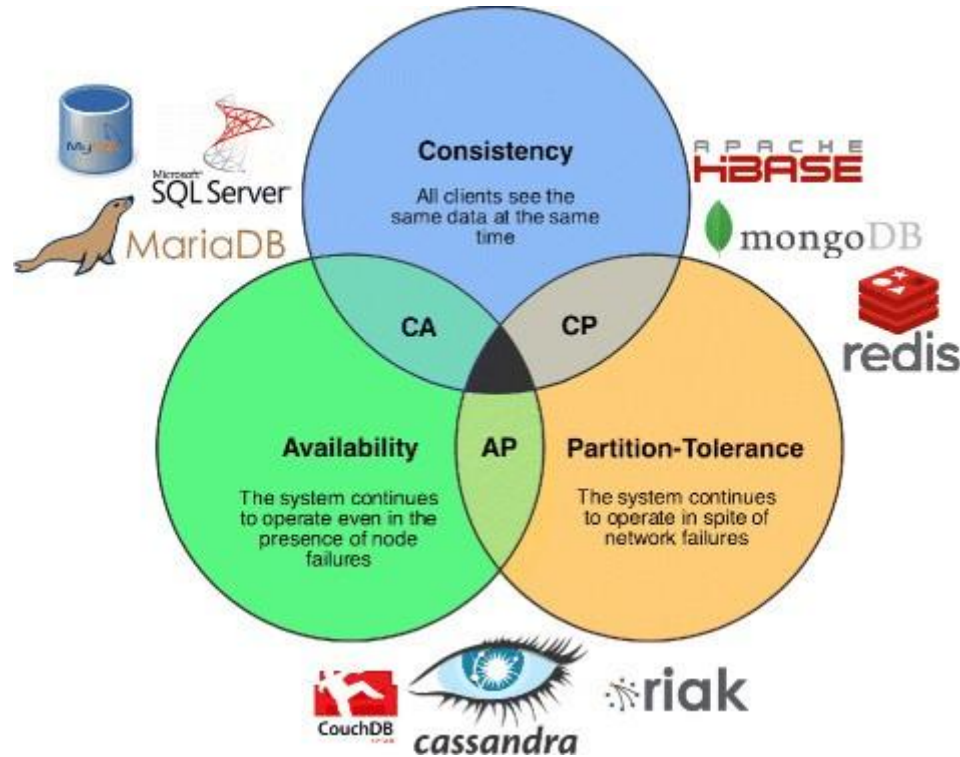
$\leq 50 \%$

... means cluster 0 % working

CAP Theorem .. Choose 2 or 3



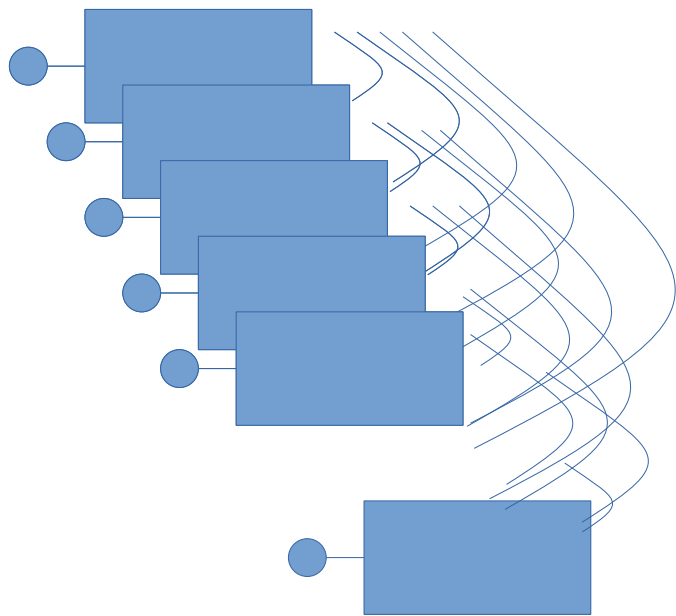
Databases choices



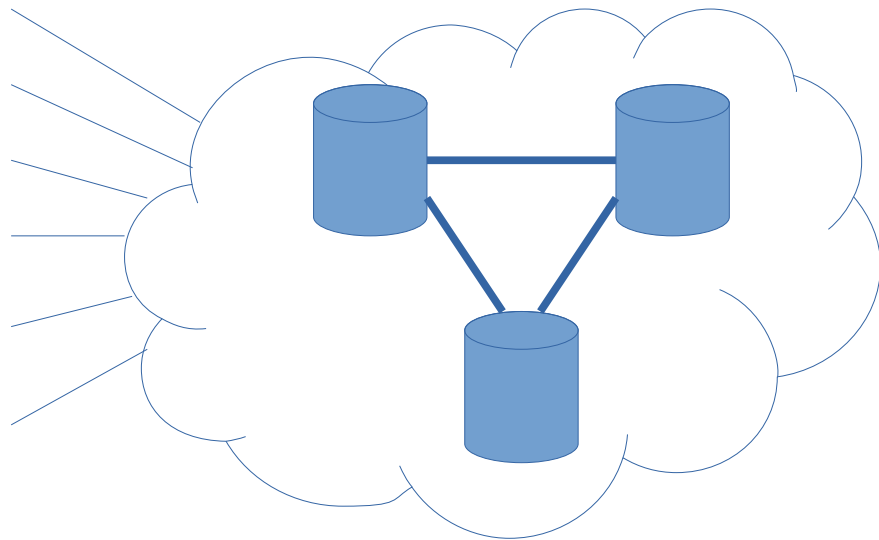
« Eventually Consistent »

.. Means « NOT » always consistent ... but ends to be

Distributed Coordination Server(s)



Difficult to synchronize data on N servers
N servers => $\frac{1}{2} * N * (N-1)$ connections .. too many
Slow to organize election



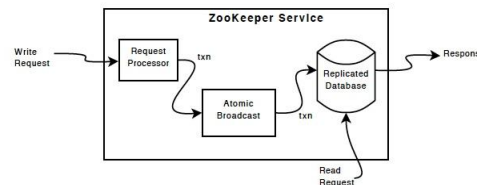
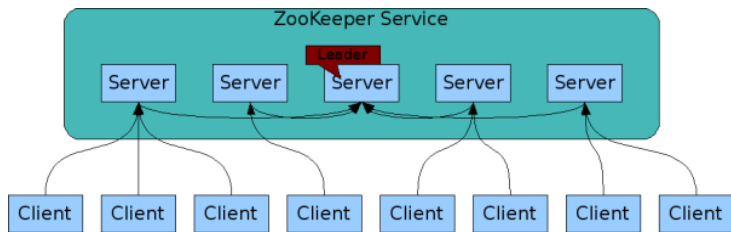
Small subsystem (3,5,7 nodes)
for quorum / election
Storing coordination data
N servers => N+3 connections

ZooKeeper



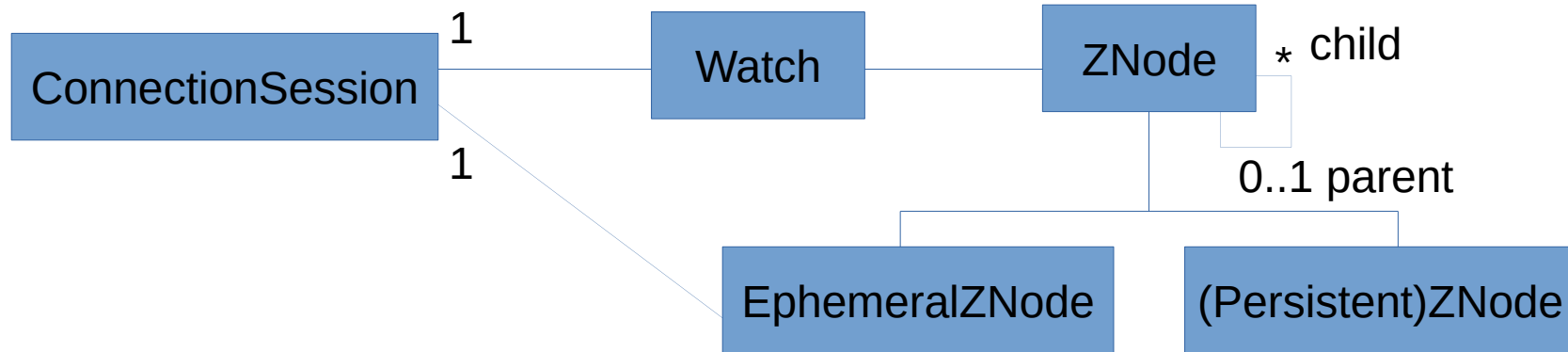
Because Coordinating Distributed Systems is a Zoo

Zookeeper is **CP** (Consistent and Partition Tolerant)
Not **A** (Available)



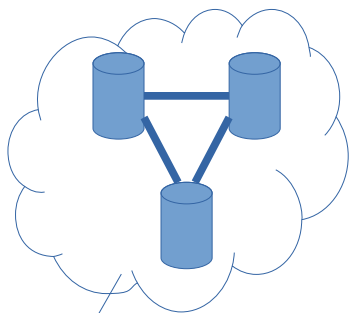
All writes are serialized to Leader
(exclusive lock)
+ replicated to quorum for committing

ZooKeeper Features



Zookeeper kernel : « key=value » database, with atomic features
Organized as directories hierarchy, called Znode
With Listening capabilities
And Ephemeral Znode for connections

ZooKeeper



Get key..
Set key=value
List key/*
Listen Key

Znode « / »

Znode « /data »

« /prop » = value

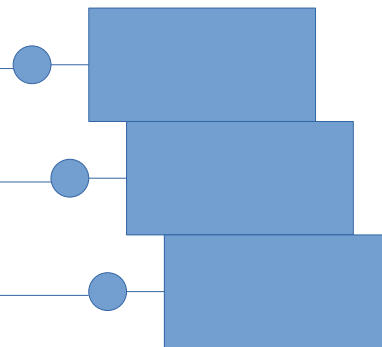
Znode « /servers »

EphemeralZNode « /node1 »

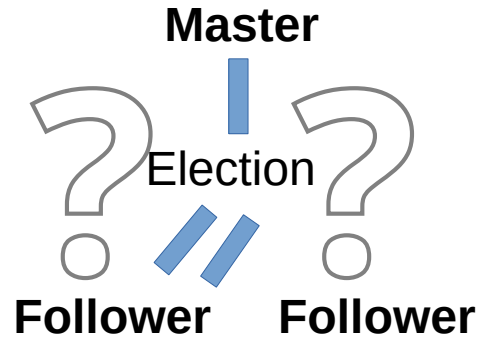
EphemeralZNode « /node2 »

EphemeralZNode « /node3 »

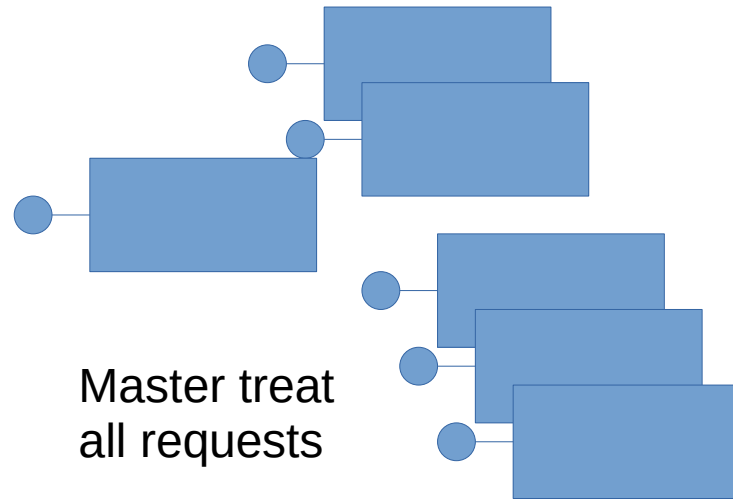
Example for service discovery :
Listen « /servers/* »



ZooKeeper : for Master Election + Persist Topology/Metadata Infos



1 Master for All => does not scale



Followers in StandBy Mode
(waste of cpu)

Sharding for Horizontal Scaling

Sharding = choose responsible server from data Id
(shared rule for clients and servers)

ServerId = Id modulo 3



Example

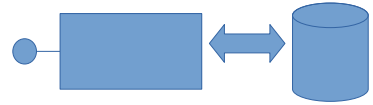
id=5



Hash: 2



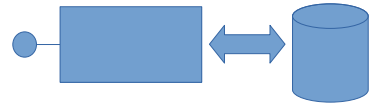
When $Id \% 3 = 0$



When $Id \% 3 = 1$



When $Id \% 3 = 2$



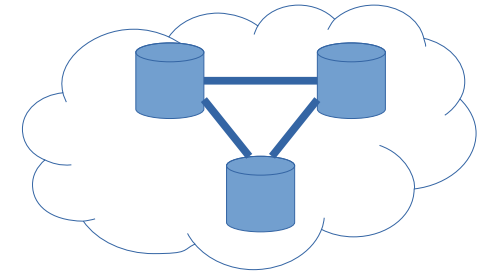
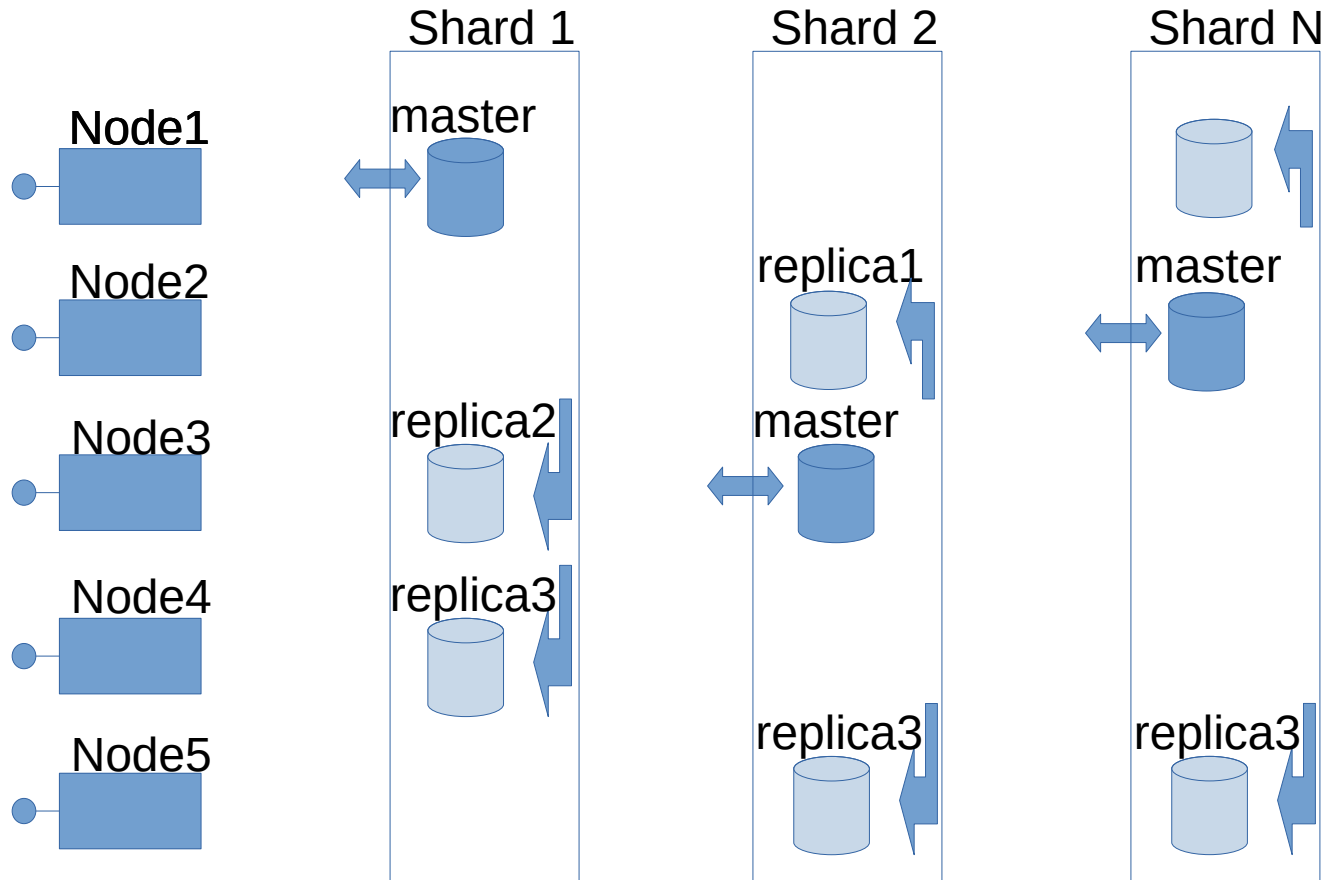
Sharding .. Pros/Cons

Scale very well « linearly » with number of servers

ONLY for request with known « ID »
(not for search / full scan)

Difficult to redimension « N » at runtime... need re-shuffle all data !

1 Master per Shard



Shard1 : master on 1
Replica on 3, 4

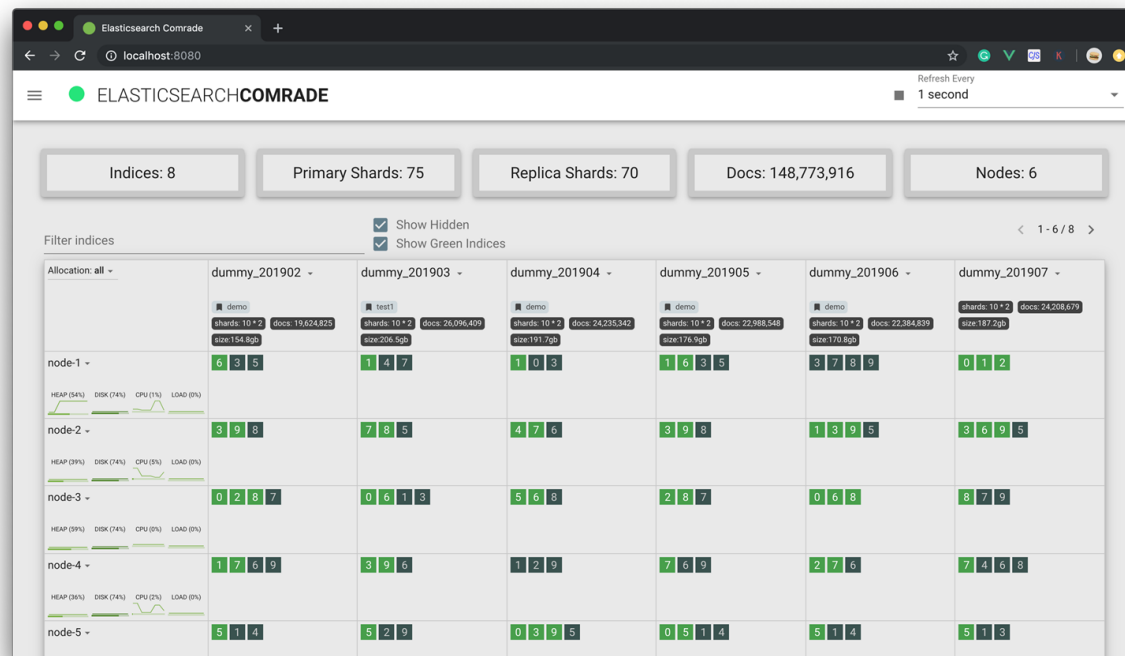
Shard2 : master on 3
Replica on 2, 5

ShardN : ...

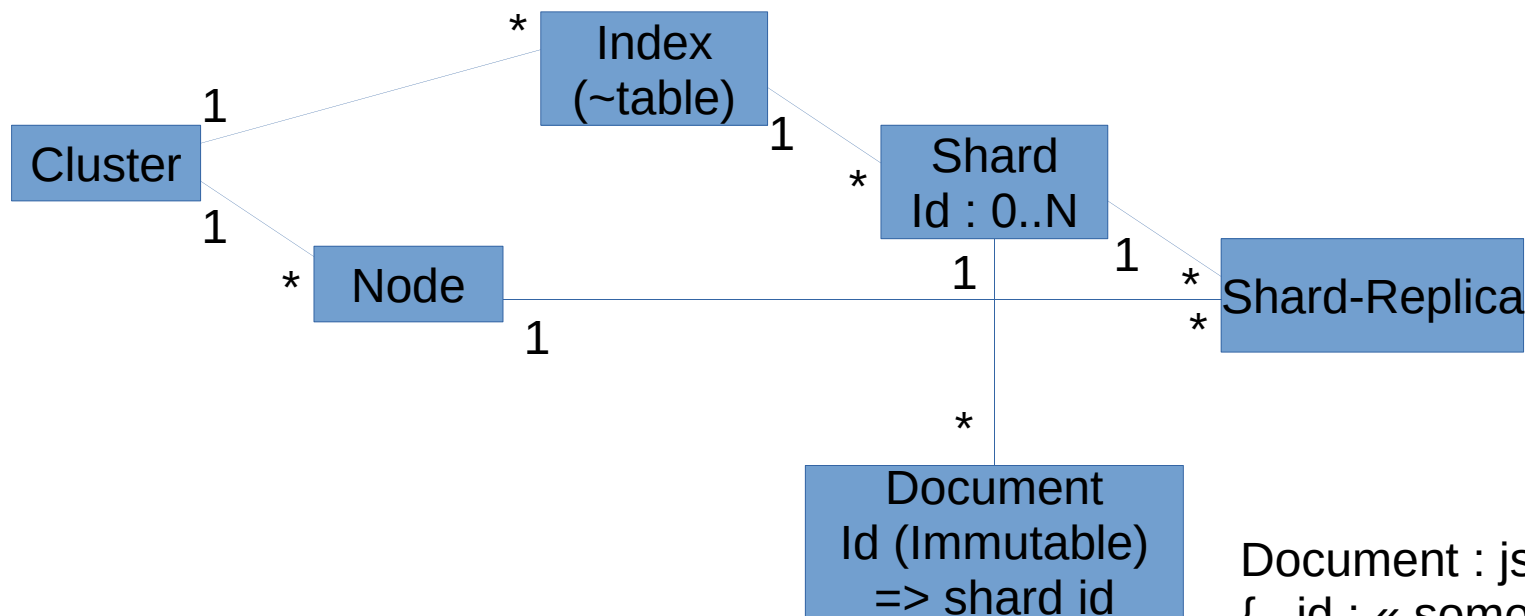
3 Examples & Comparisons for Sharding + Master/Replica



Example 1/3 : Elasticsearch



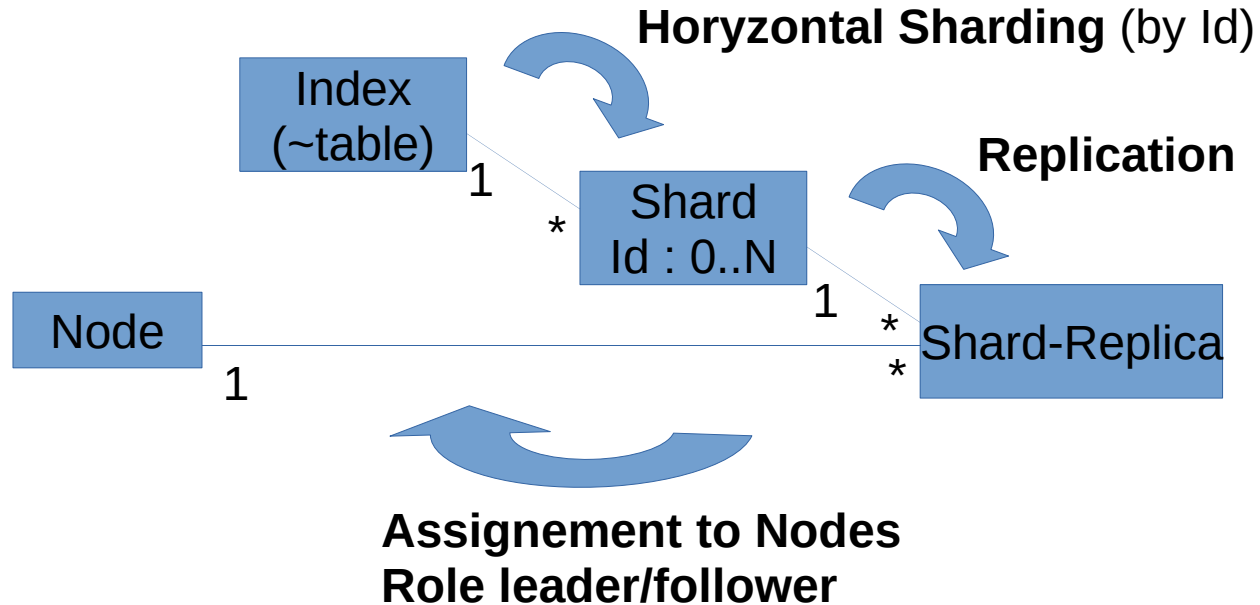
ElasticSearch ... UML model



Document : json text
{
 _id : « some-id1 »,
 field1 : 123,
 field2 : { subField : [« a »] }
}

ElasticSearch ...

Zooming Relations



HDFS



Hadoop

Overview

Datanodes

Datanode Volume Failures

Snapshot

Startup Progress

Utilities

Datanode Information

In service

Down

Decommissioned

Decommissioned & dead

In operation

Show

25

entries

Search:

Node	Http Address	Last contact	Last Block Report	Capacity	Blocks	Block pool used	Version
<div><div></div><div>node-1:50010 (10.128.7.122:50010)</div></div>	node-1:50075	4s	14m	18.74 GB <div><div></div></div>	320	348.61 MB (1.82%)	2.7.3.2.6.4.0-91
<div><div></div><div>node-3:50010 (10.128.6.1:50010)</div></div>	node-3:50075	3s	52106m	18.74 GB <div><div></div></div>	1	486.1 MB (2.53%)	2.7.3.2.6.4.0-91
<div><div></div><div>node-4:50010 (10.129.2.1:50010)</div></div>	node-4:50075	4s	29m	8.73 GB <div><div></div></div>	473	596.96 MB (6.67%)	2.7.3.2.6.4.0-91
<div><div></div><div>node-5:50010 (10.131.2.1:50010)</div></div>	node-5:50075	4s	76m	8.73 GB <div><div></div></div>	715	1.32 GB (15.1%)	2.7.3.2.6.4.0-91
<div><div></div><div>node-7:50010 (10.128.0.1:50010)</div></div>	node-7:50075	4s	52106m	8.73 GB <div><div></div></div>	0	833.62 MB (9.32%)	2.7.3.2.6.4.0-91

Showing 1 to 5 of 5 entries

Previous

1

Next

File information - 6e584bed53254094b2d2e1cc1b643b63

Download

Block information - Block 1

Block ID: 1212161965

Block Pool ID: BP-1065830962-192-196-114-35-1494702271527

Generation Stamp: 1195

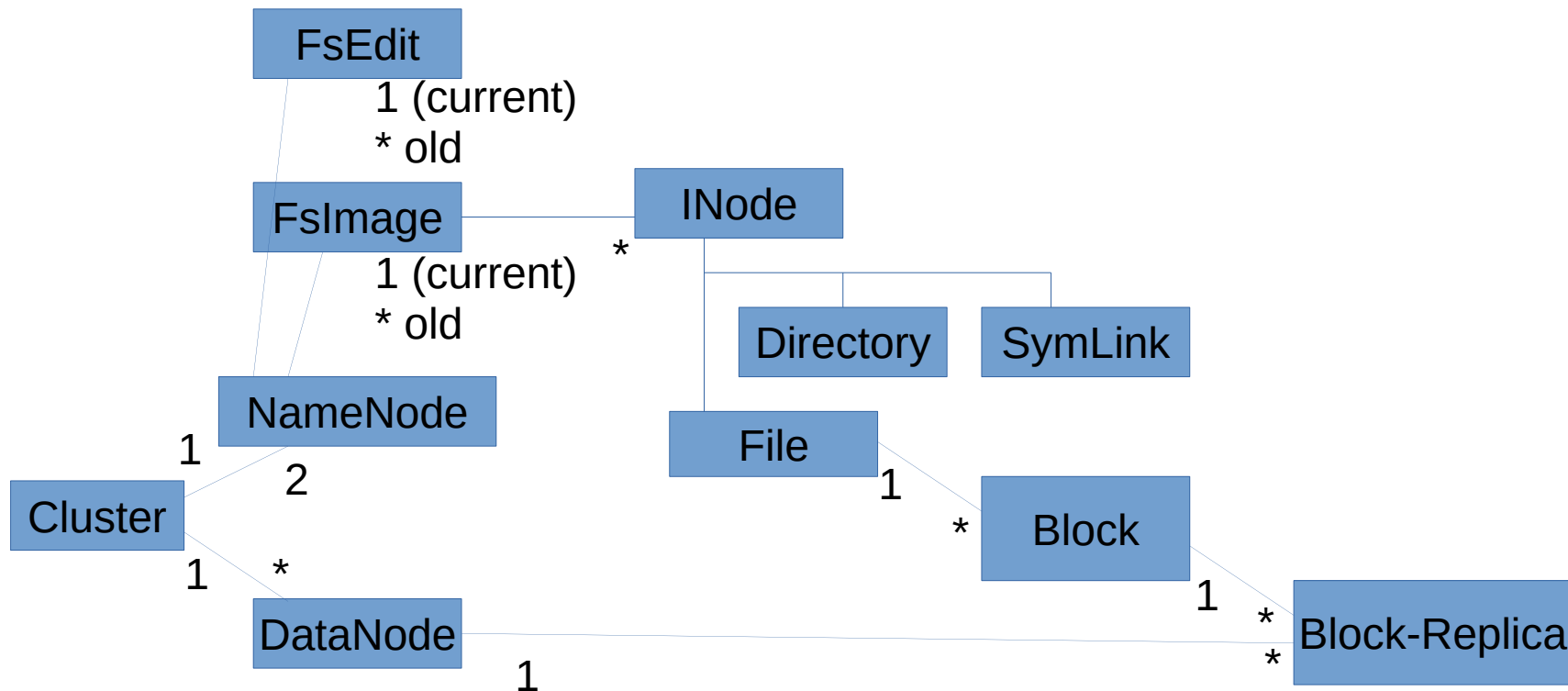
Size: 6053

Availability:

- node-1:50010 (10.128.7.122:50010)
- node-3:50010 (10.128.6.1:50010)
- node-4:50010 (10.129.2.1:50010)

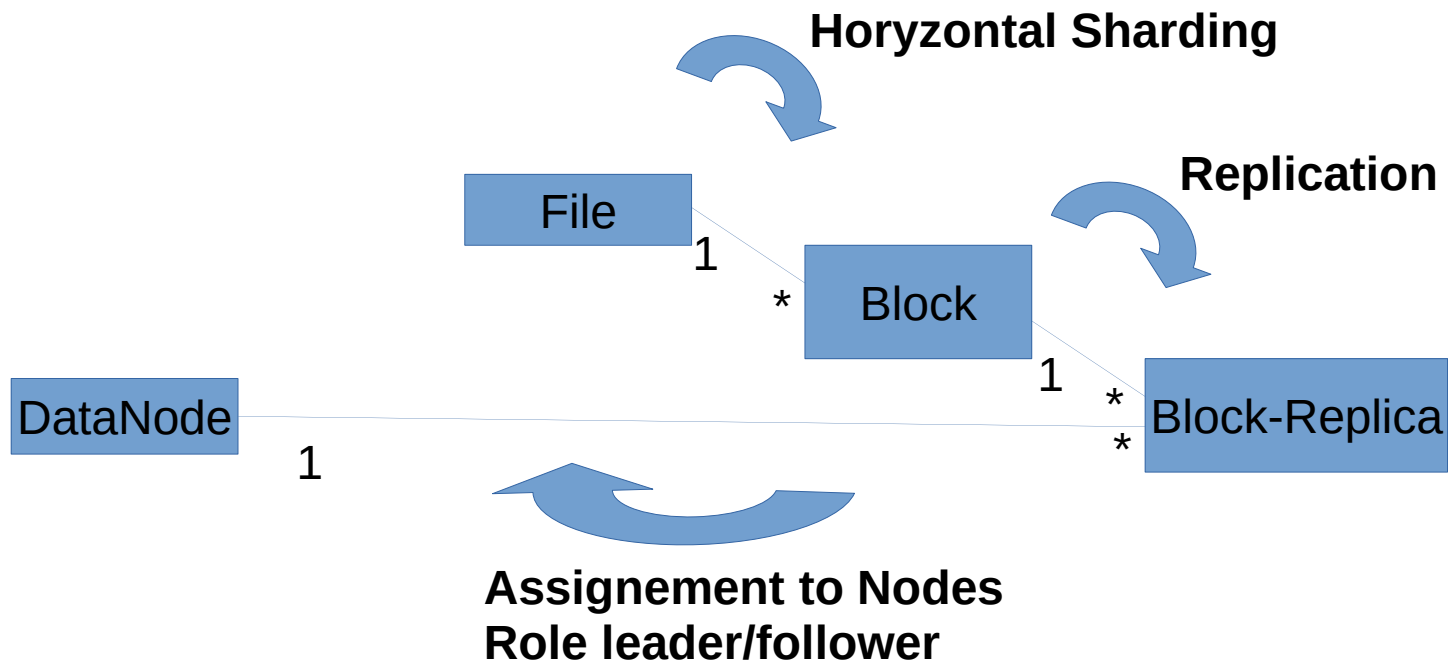
Close

HDFS ... UML Model



File: binary large object « blob »
 = metadata
 (owner, group, chmod, acl, ..)
 + list of blocks

HDFS ... Zoomin Relations



HBase



Not Secure | http://10.10.10.10:16010/master-status

HBASE Home Table Details Procedures Local Logs Log Level Debug Dump Metrics Dump HBase Configuration

Master hdp1

Region Servers

[View Data](#) [History](#) [Requests](#) [Statistics](#) [Compositions](#)

ServerName	Start time	Version	Requests Per Second	Num. Region Servers
hdp2.686020.1556711447825	Wed Feb 20 20:13:47 EST 2019	1.2.0-odh5.7.6	8	5
hdp3.686020.1556711447450	Wed Feb 20 20:13:47 EST 2019	1.2.0-odh5.7.6	8	4
hdp4.686020.1556711447450	Wed Feb 20 20:13:47 EST 2019	1.2.0-odh5.7.6	8	3
Total 3			8	12

Not Secure | http://10.10.10.10:16010/master-status

HBASE Home Table Details Local Logs Log Level Debug Dump Metrics Dump HBase Configuration

Tables

[View Tables](#) [System Tables](#) [Snapshots](#)

2 tables in set [Details](#)

Namespace	Table Name	Online Regions	Offline Regions	Failed Regions	Split Regions	Other Regions	Description
default	test	1	0	0	0	0	'test', (NAME => 'cf')
default	test2	1	0	0	0	0	'test2', (NAME => 'id'), (NAME => 'temp')

HBase ... UML Model

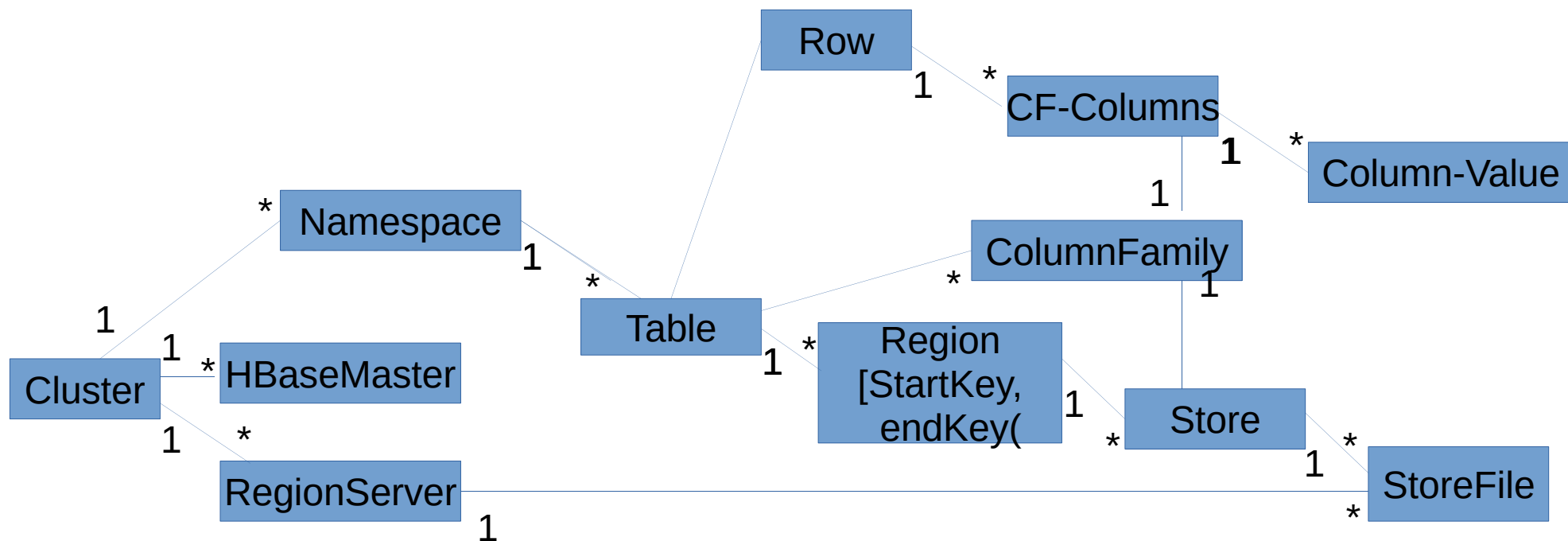


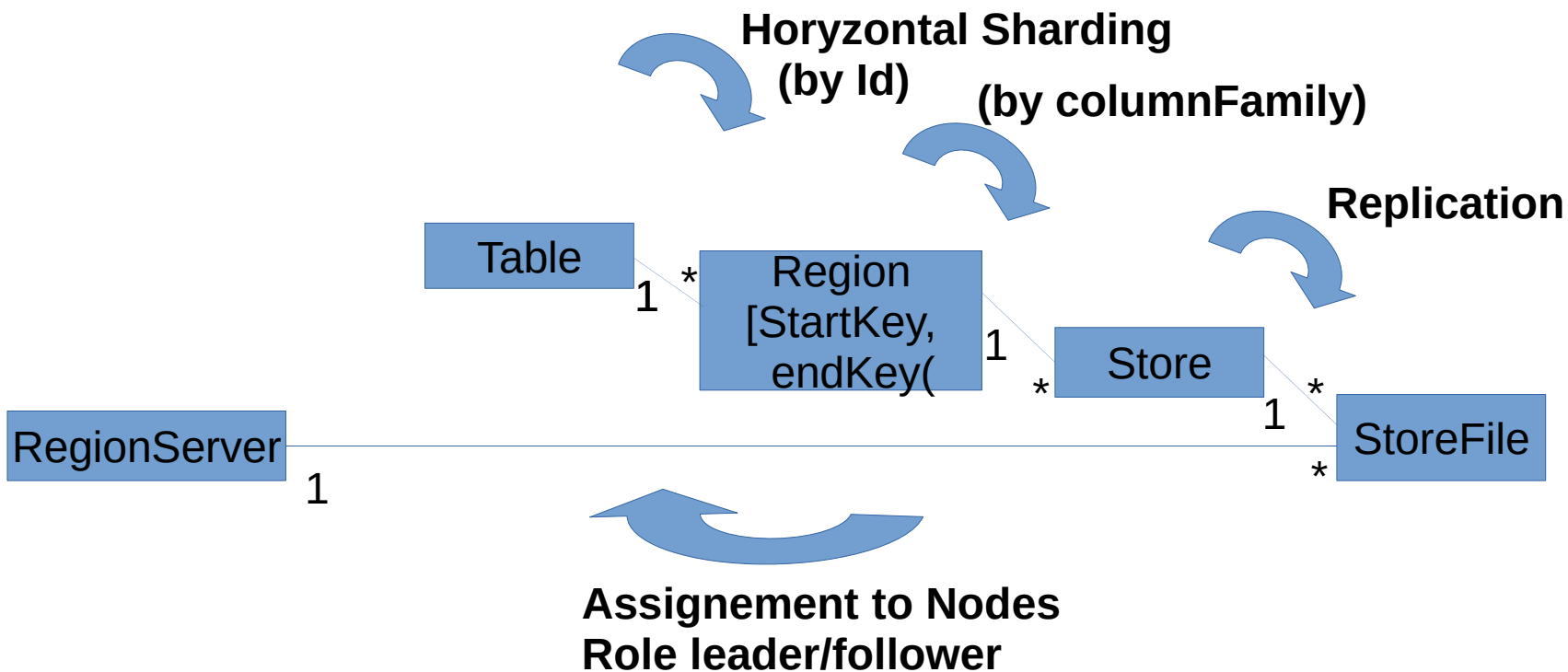
Table = Map<byte[], Map<byte[], byte[]>>

RowKey

ColumnFamily:Column

BlobValue

HBase ... Zooming Relations



Questions ?