Architecting Distributed Databases for Failure

A Case Study with Druid

Fangjin Yang

Cofounder @ imply

Overview

The Bad

The Really Bad

The Catastrophic

Best Practices: Operations

Everything is going to fail!

Requirements

Scalable

- Tens of thousands of nodes
- Petabytes of raw data

Available

- 24 x 7 x 365 uptime

Performant

- Run as smoothly as possible when things go wrong

Druid

Open source distributed data store

Column oriented storage of event data

Low latency OLAP queries & low latency data ingestion

Initially designed to power a SaaS for online advertising (in AWS)

Our real-world example case study

The Bad

Single Server Failures

Common

Occurs for every imaginable and unimaginable reason

- Hardware malfunction, kernel panic, network outage, etc.
- Minimal impact

Standard solution: replication

Druid Segments

Timestamp	Dimensions	Measures
2015-01-01T00		
2015-01-01T01		
2015-01-02T05		
2015-01-02T07		
2015-01-03T05		
2015-01-03T07		

Timestamp	Dimensions	Measures
2015-01-01T00		
2015-01-01T01		

Segment_2015-01-01/2014-01-02

Timestamp	Dimensions	Measures
2015-01-02T05		
2015-01-02T07		

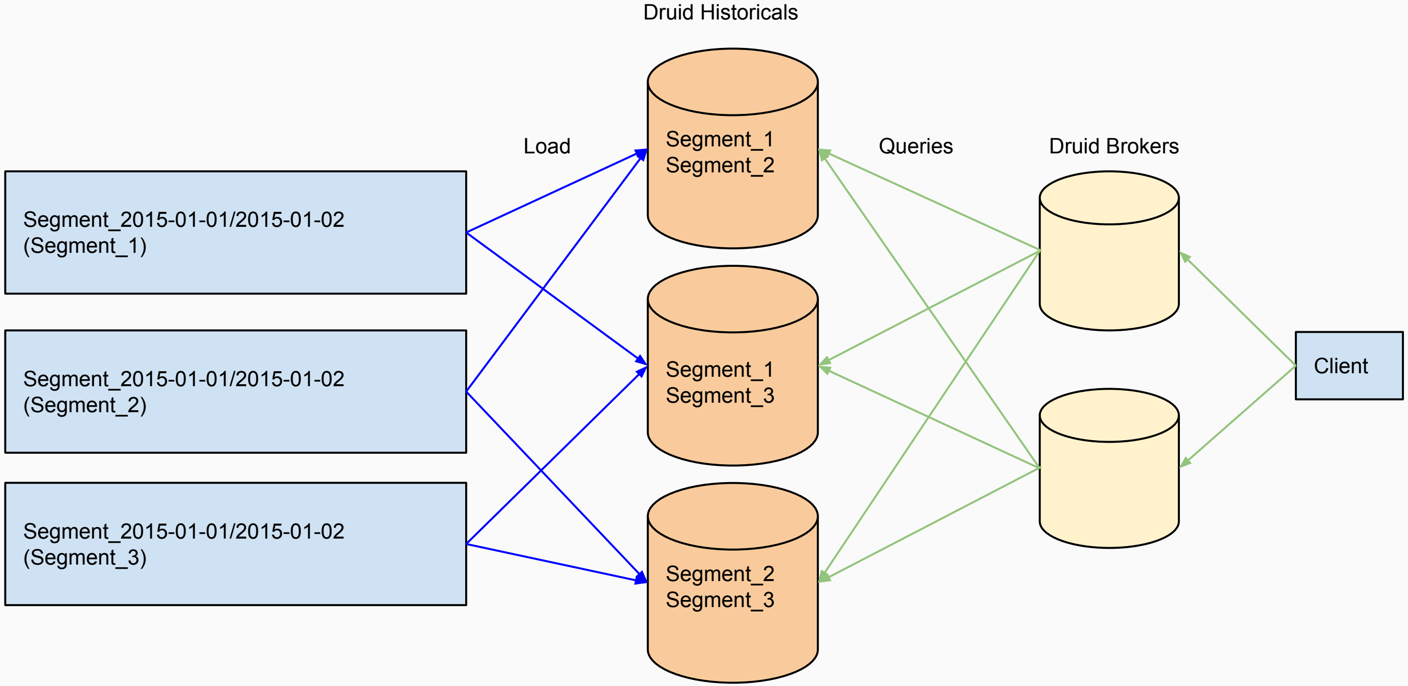
Segment_2015-01-02/2014-01-03

Timestamp	Dimensions	Measures
2015-01-03T05		
2015-01-03T07		

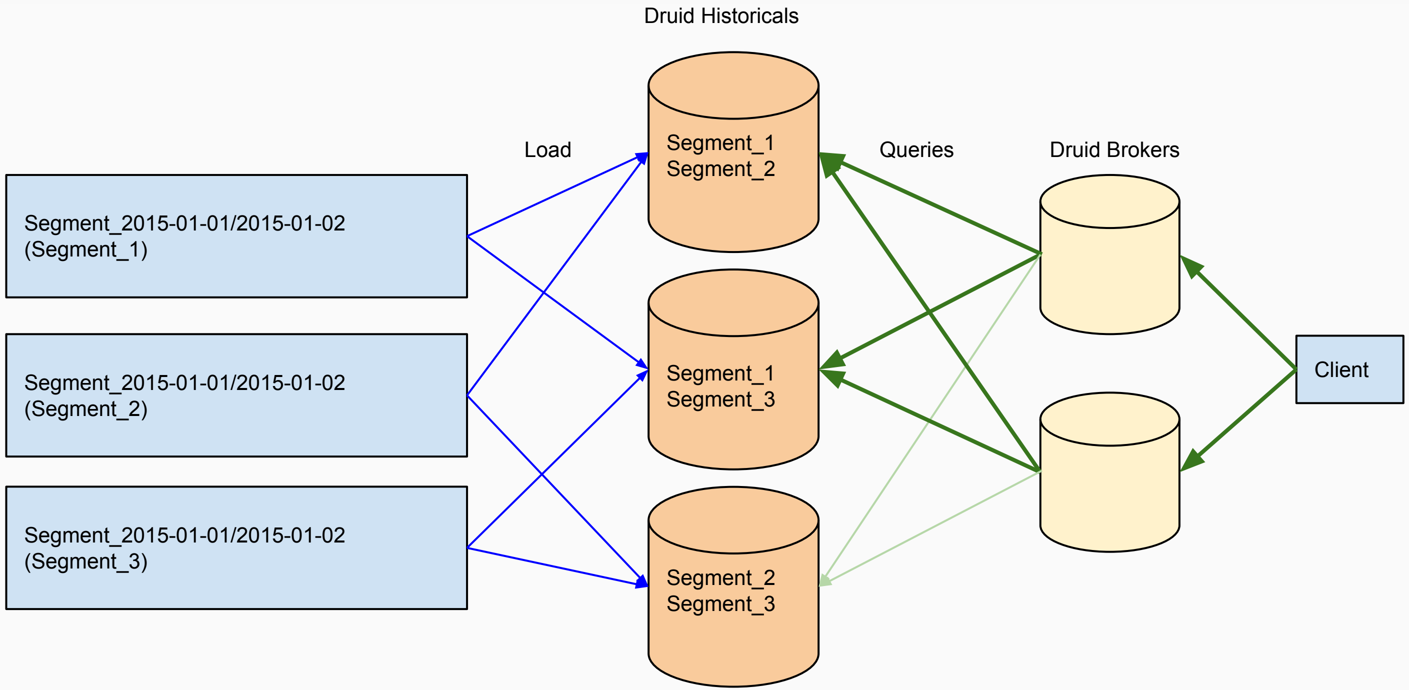
Segment_2015-01-03/2014-01-04

Partition by time

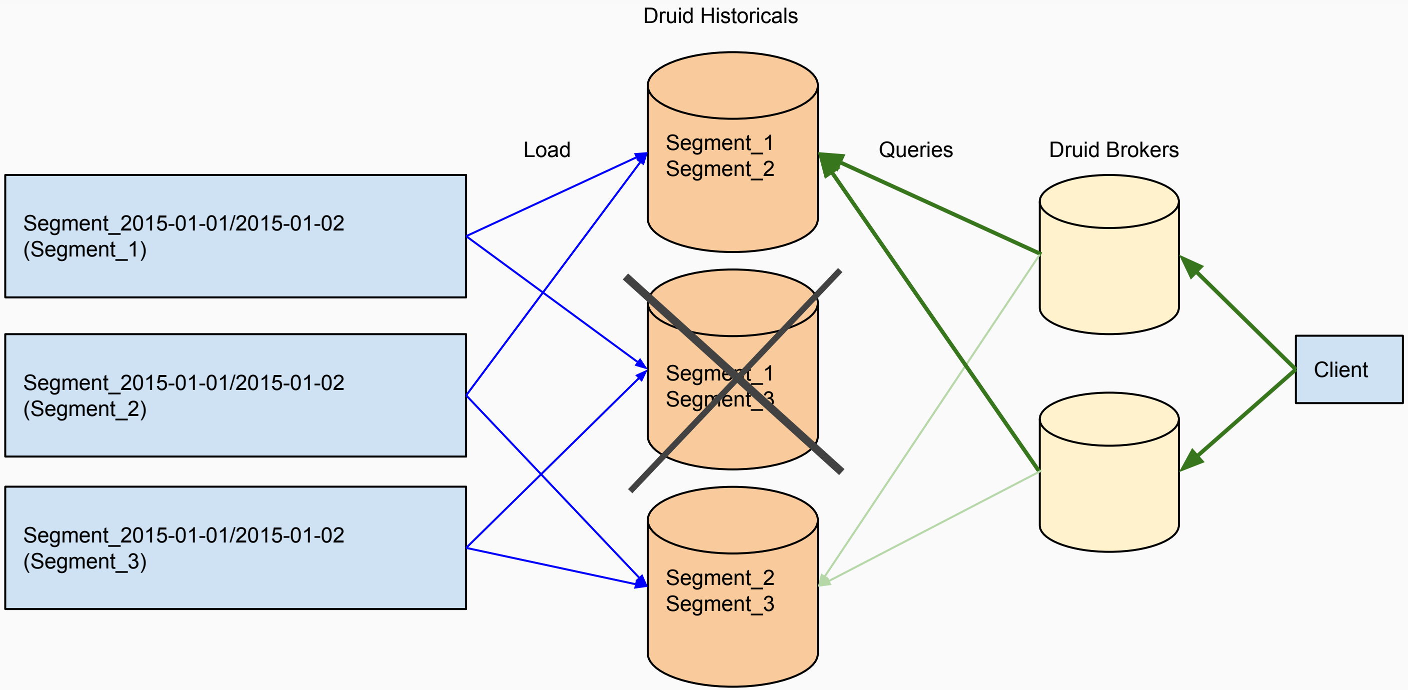
Replication Example



Query Segment_1



Query Segment_1



Multi-Server Failures

Common: 1 server fails
Less common: >1 server fails

Data center issues (rack failure)

Two strategies:

- fast recovery
- multi-datacenter replication

Fast Recovery

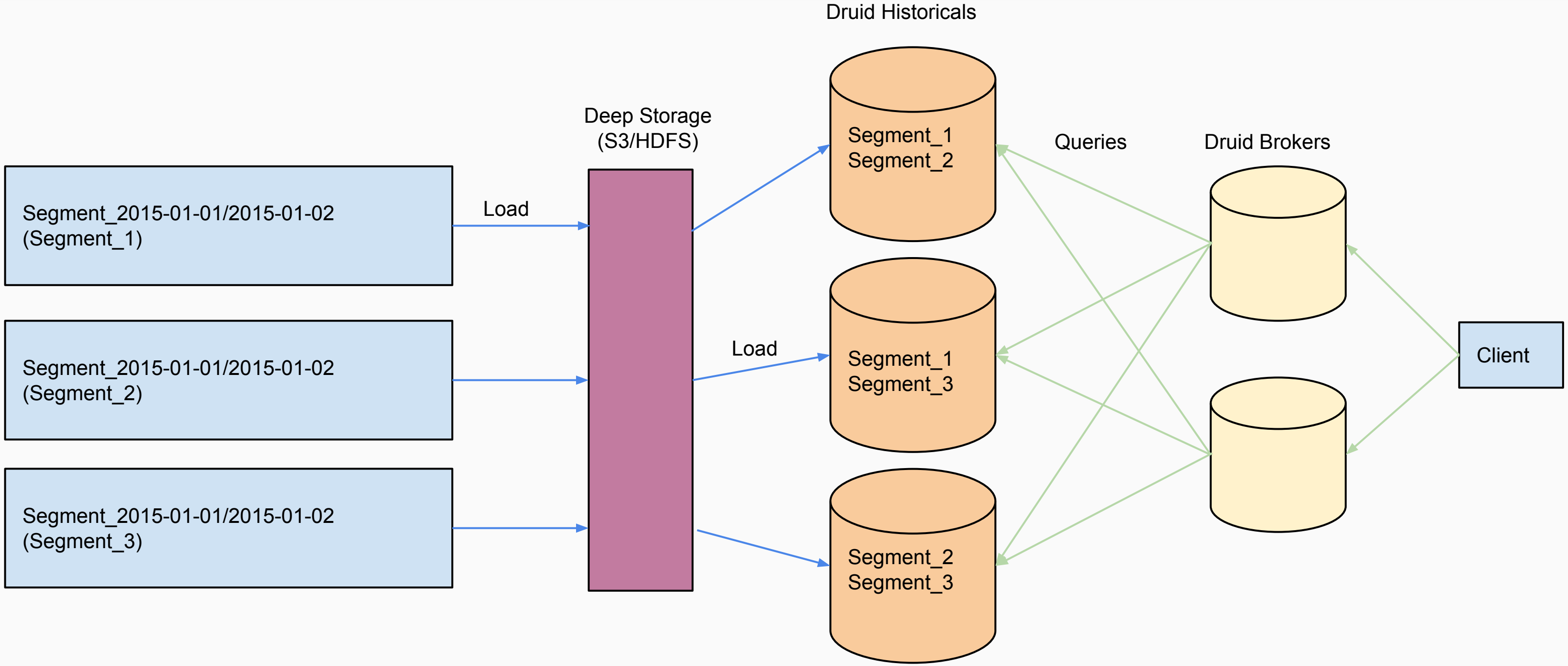
Complete data availability in the face of multi-server failures is hard!

Focus on fast recovery instead

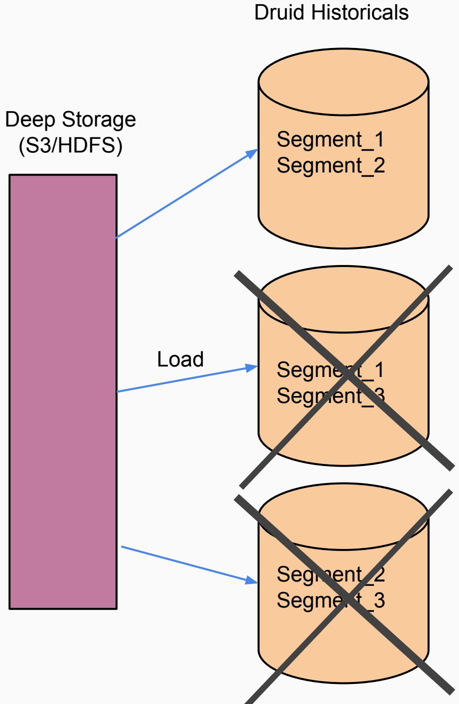
Be careful of the pitfalls of fast recovery

More viable in the cloud

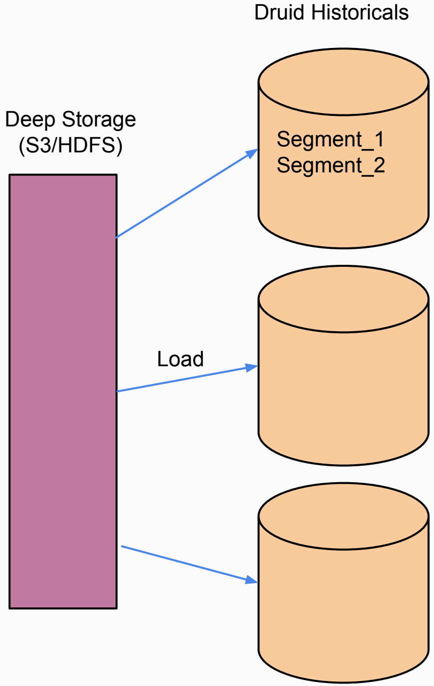
Fast Recovery Example



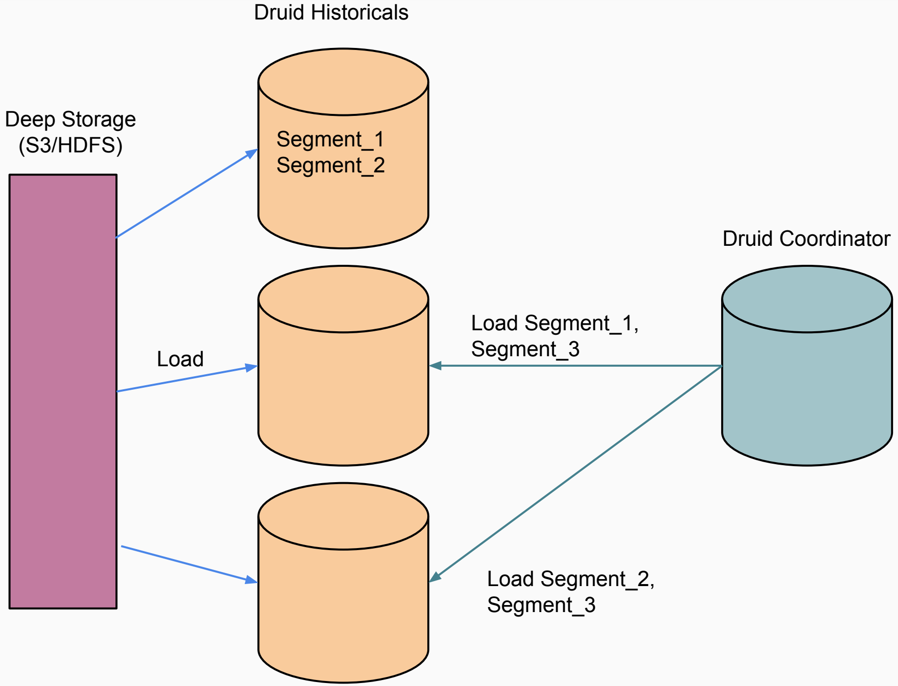
Fast Recovery Example



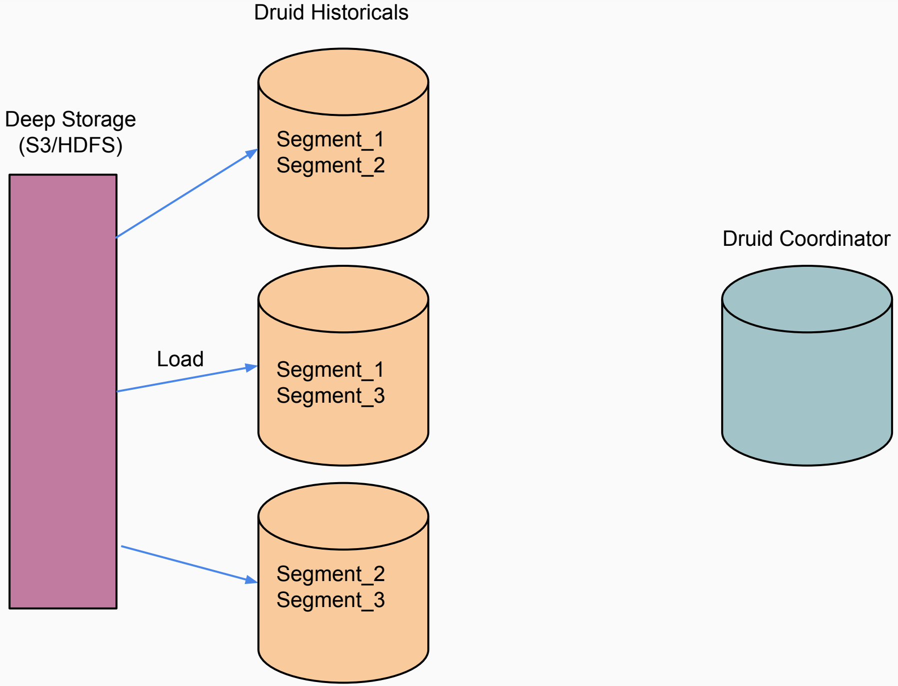
Fast Recovery Example



Fast Recovery Example



Fast Recovery Example



Dangers of Fast Recovery

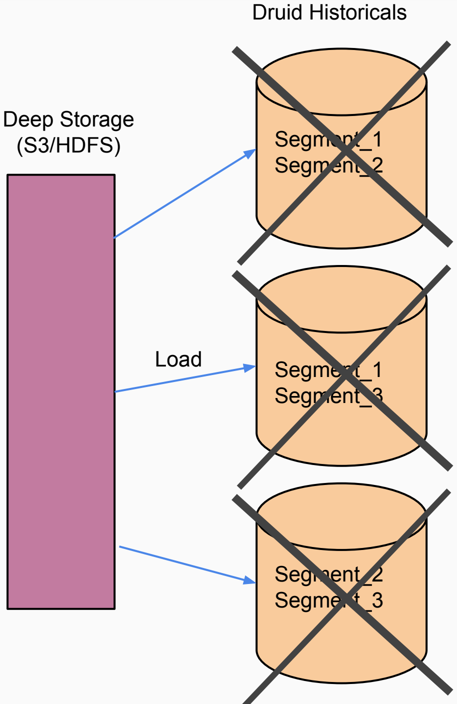
Easy to create bottlenecks

- Prioritize how resources are spent during recovery
- Druid prioritizes data availability and throttles replication

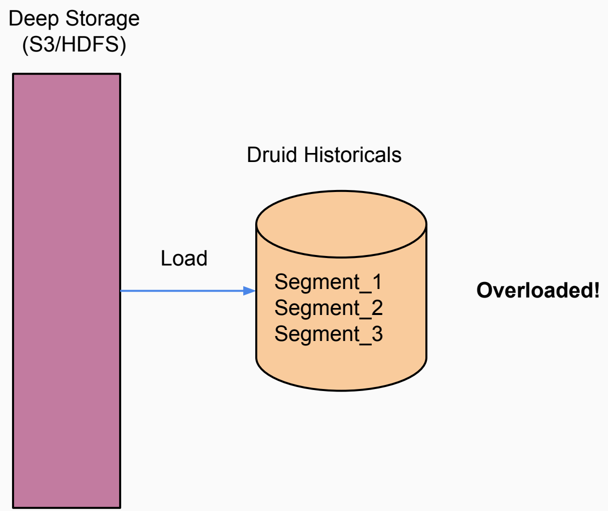
Beware query hotspots

- Intelligent load balancing during recovery is important

Fast Recovery Example



Fast Recovery Example



The Really Bad

Data Center Outage

Very uncommon

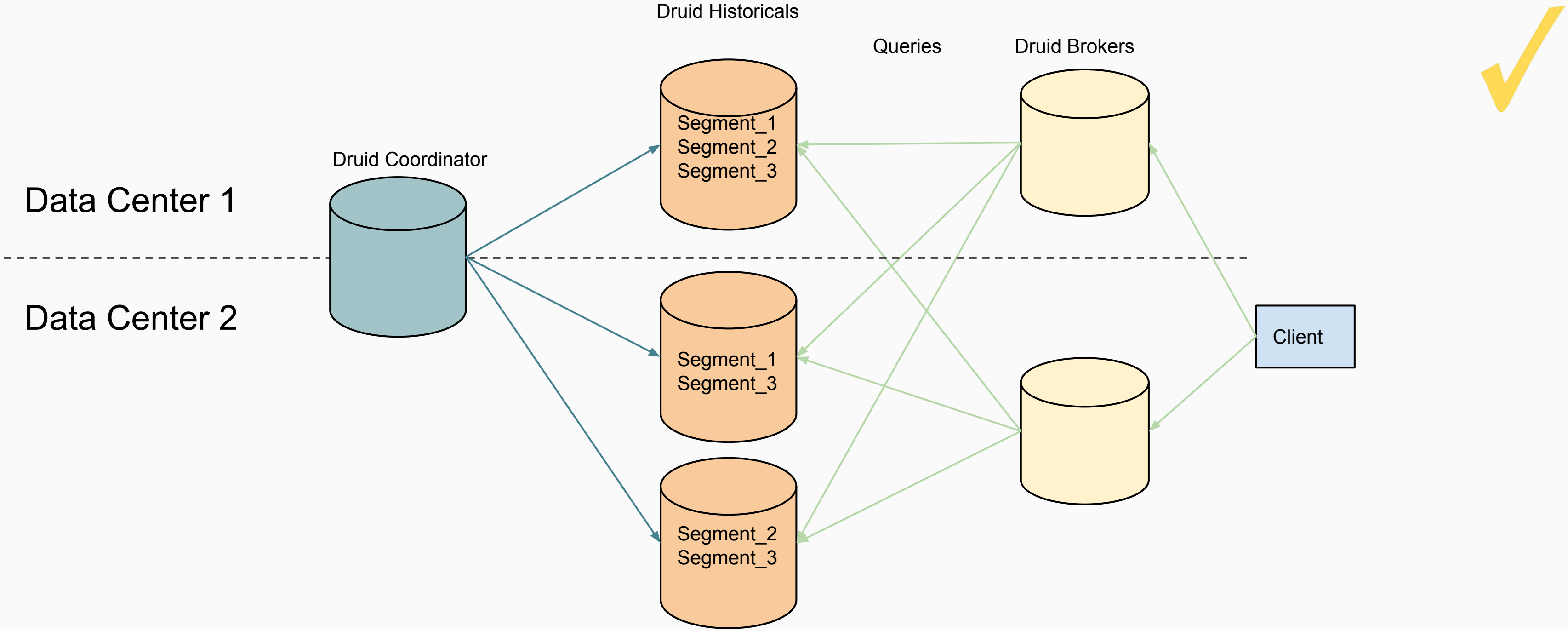
Power loss

Can be extremely disruptive without proper planning

Solution: Multi-datacenter replication

Beware pitfalls of multi-datacenter replication

Multi-Datacenter Replication



Multi-Datacenter Pitfalls

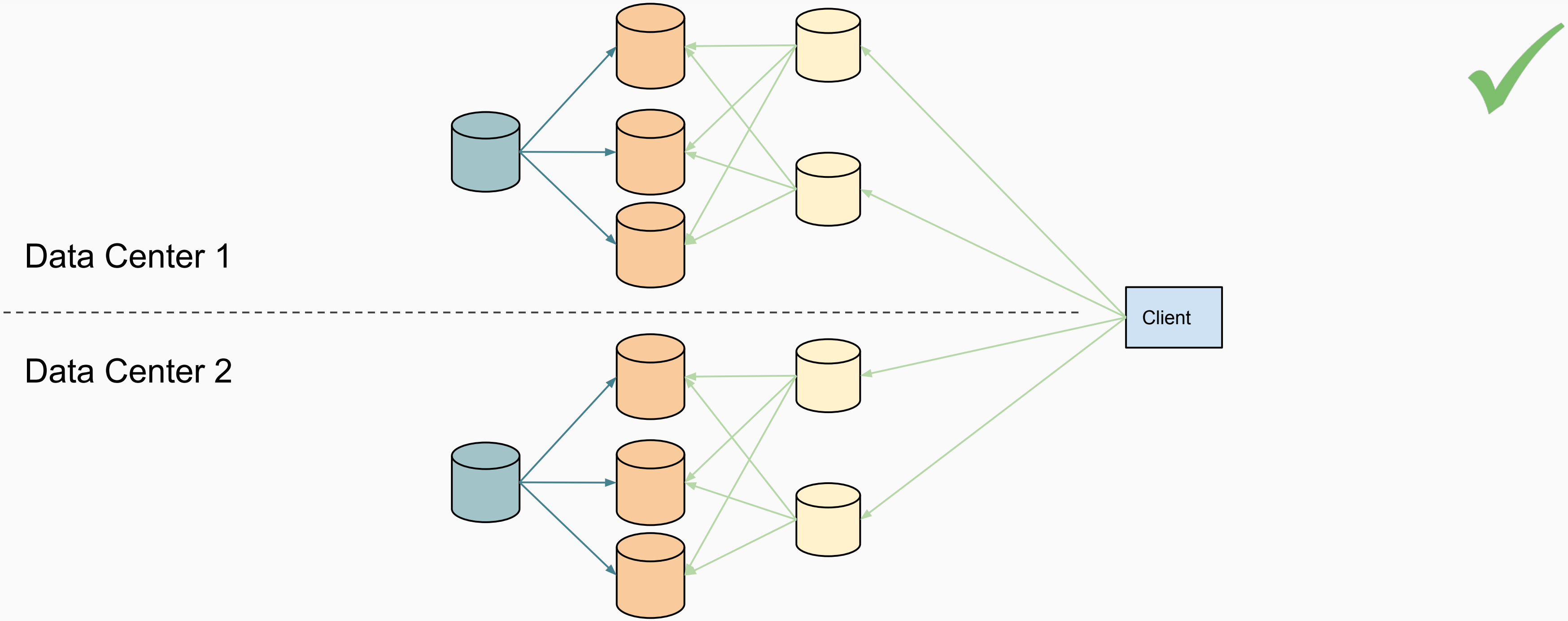
Coordination + leader election can be tricky

Communication can require non-trivial network time

Coordination usually done with heartbeats and quorum decisions

Writes, failovers, & consistent reads require round trips

Multi-Datacenter Replication



The Catastrophic

“Why are things slow today?”

Poor performance is much worse than things completely failing

Causes:

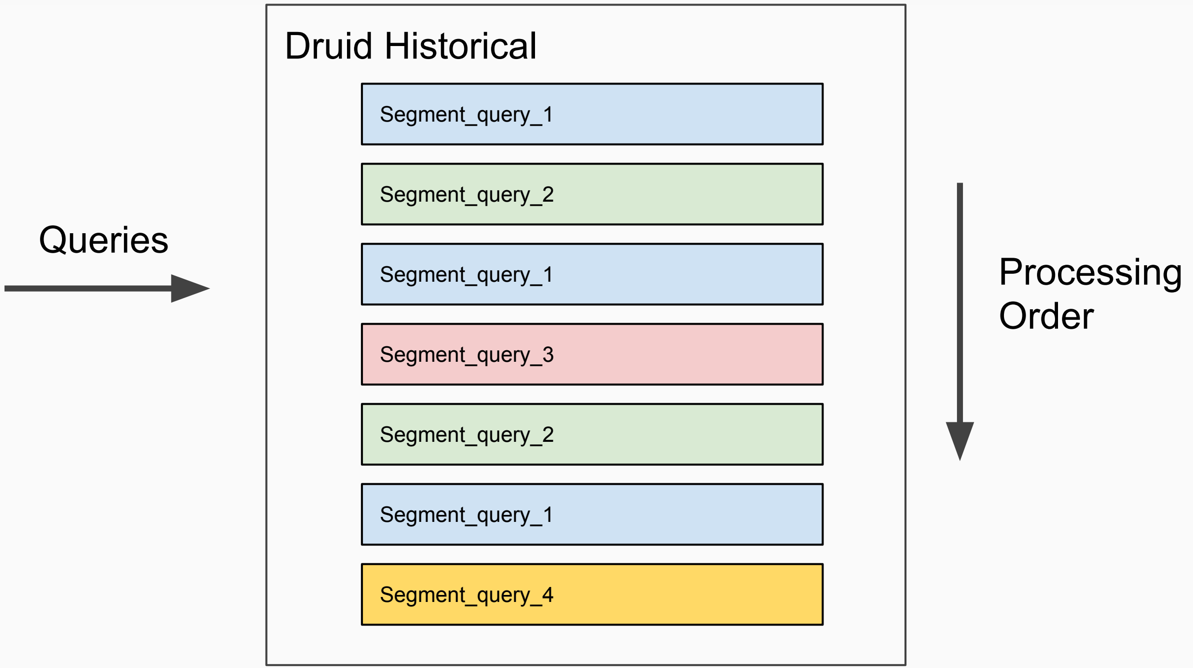
- Heavy concurrent usage (multi-tenancy)
- Hotspots & variability
- Bad software update

Architecting for Multi-tenancy

Small units of computation

- No single query should starve out a cluster

Druid Multi-tenancy

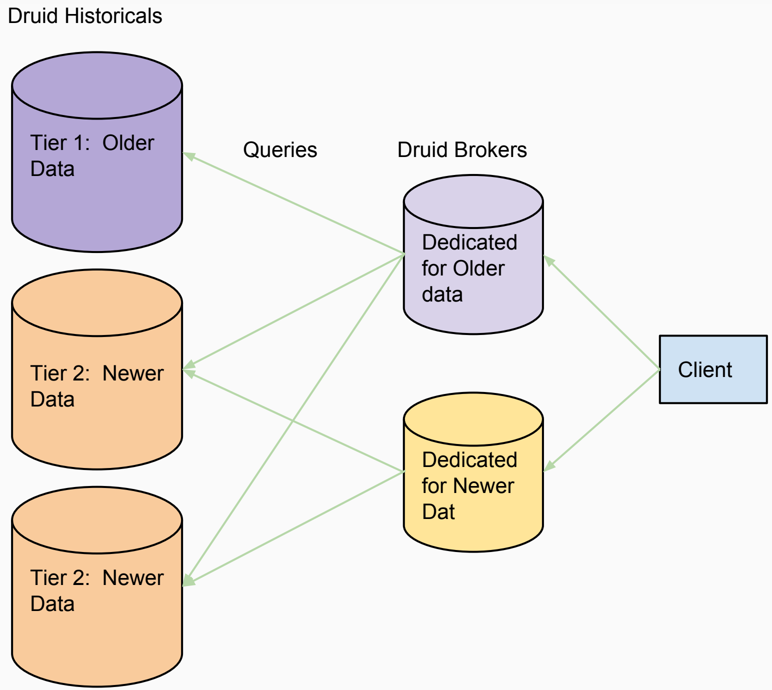


Architecting for Multi-tenancy

Resource prioritization and isolation

- Not all queries are equal
- Not all users are equal

Druid Multi-tenancy



Hotspots

Incredible variability in query performance among nodes

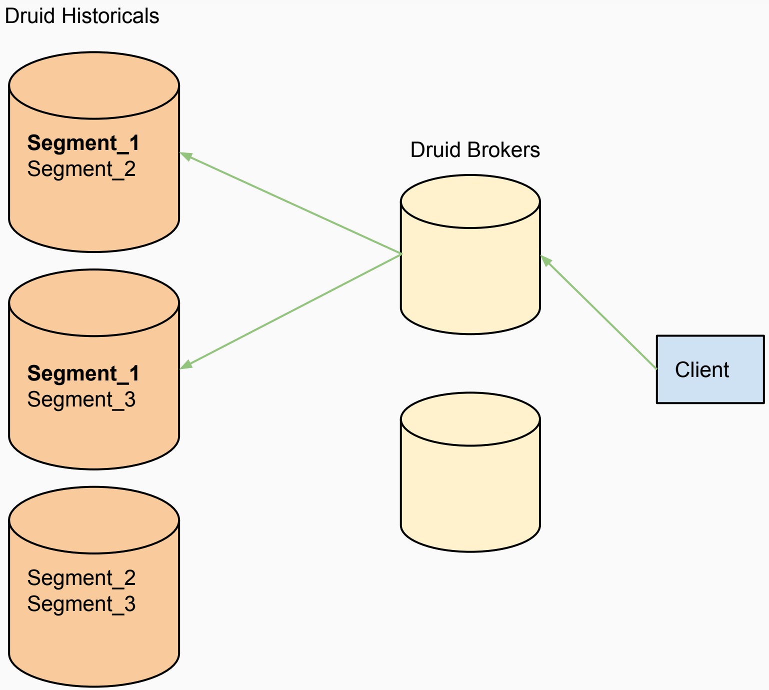
Nodes may become slow but not fail

Difficult to detect as there is nothing obviously wrong

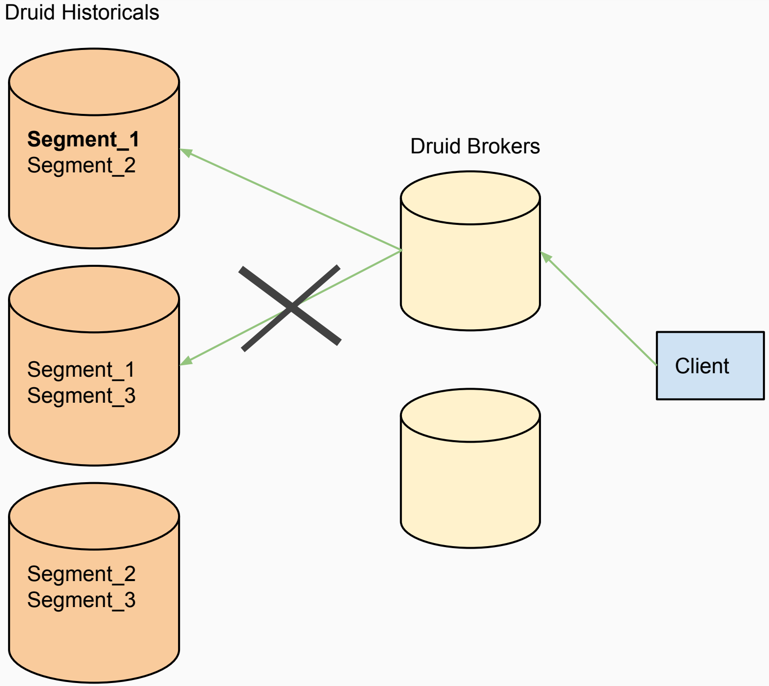
Solutions:

- Hedged requests
- Selective Replication
- Latency Induced Probation

Hedged Requests



Hedged Requests



Minimizing Variability

Selective Replication

Latency-induced probation

Great paper: https://web.stanford.edu/class/cs240/readings/tail-at-scale.pdf

Bad Software Updates

It is very difficult to simulate production traffic

- Testing/staging clusters mostly verify correctness

No noticeable failures for a long time

Common cause of cascading failures

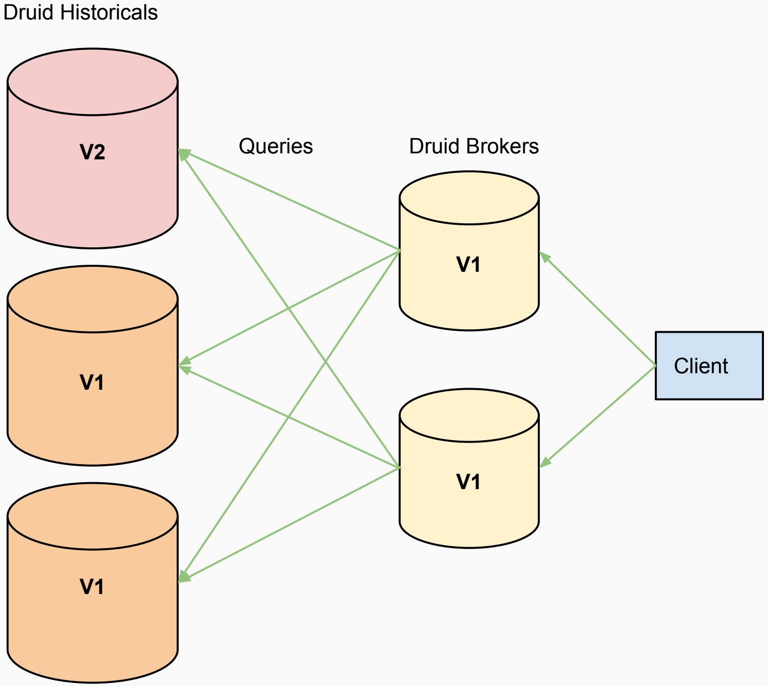
Rolling Upgrades

Be able to update different components with no down time

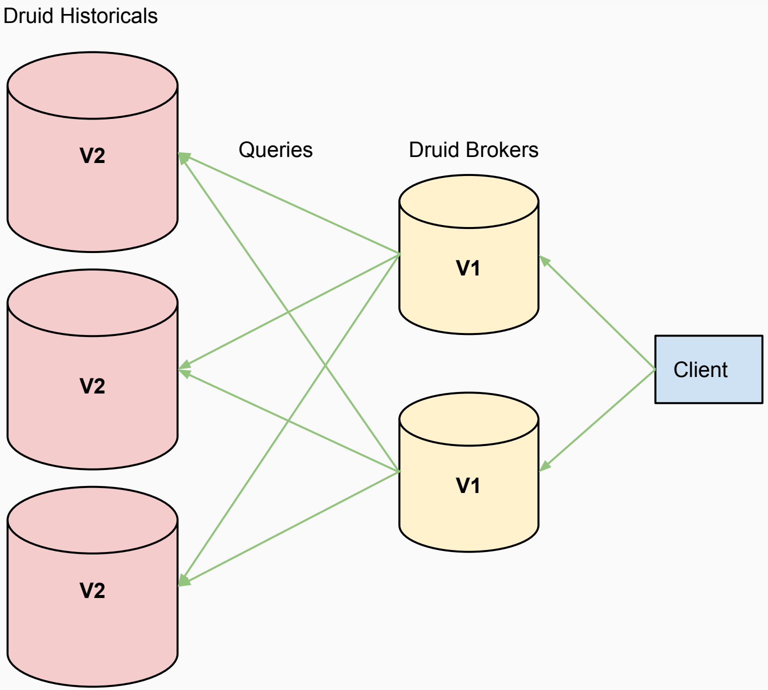
Backwards compatibility is extremely important

Roll back if things are bad

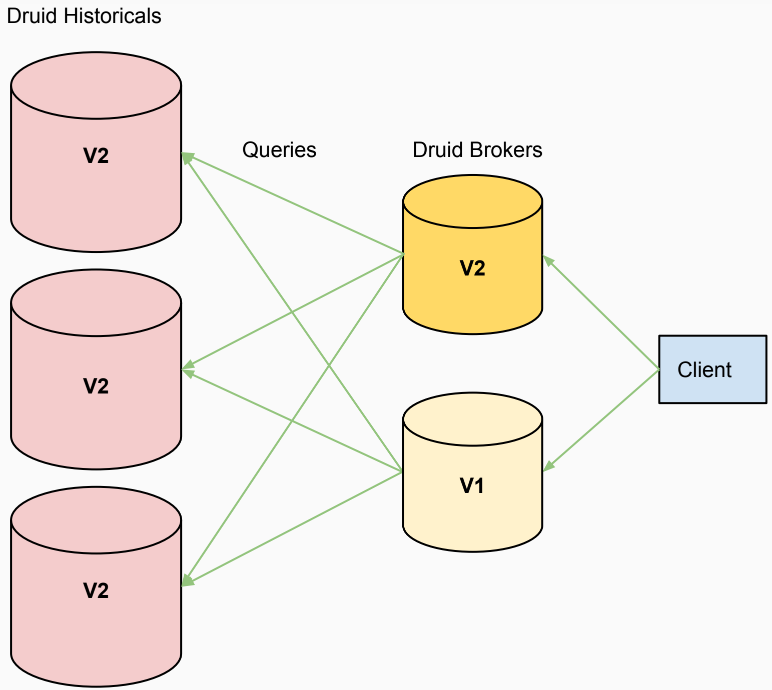
Rolling Upgrades



Rolling Upgrades



Rolling Upgrades



Best Practices: Operations

Monitoring

Detection of when things go badly

Define your critical metrics and acceptable values

Alerts

Alert on critical errors

- Out of disk space, out of cluster capacity, etc.

Design alerts to reduce “noise”

- Distinguish warnings and alerts

Exploratory Analytics

Extremely critical to diagnosing root causes quickly

Not many organizations do this

Takeaways

Everything is going to fail!

- Use replication for single server failures
- Use fast recovery for multi-server failures (when you don't want to set up another data center)
- Use multi-datacenter replication when availability really matters
- Alerting, monitoring, and exploratory analysis are critical

Thanks!

@implydata
@druidio
@fangjin

imply.io
druid.io