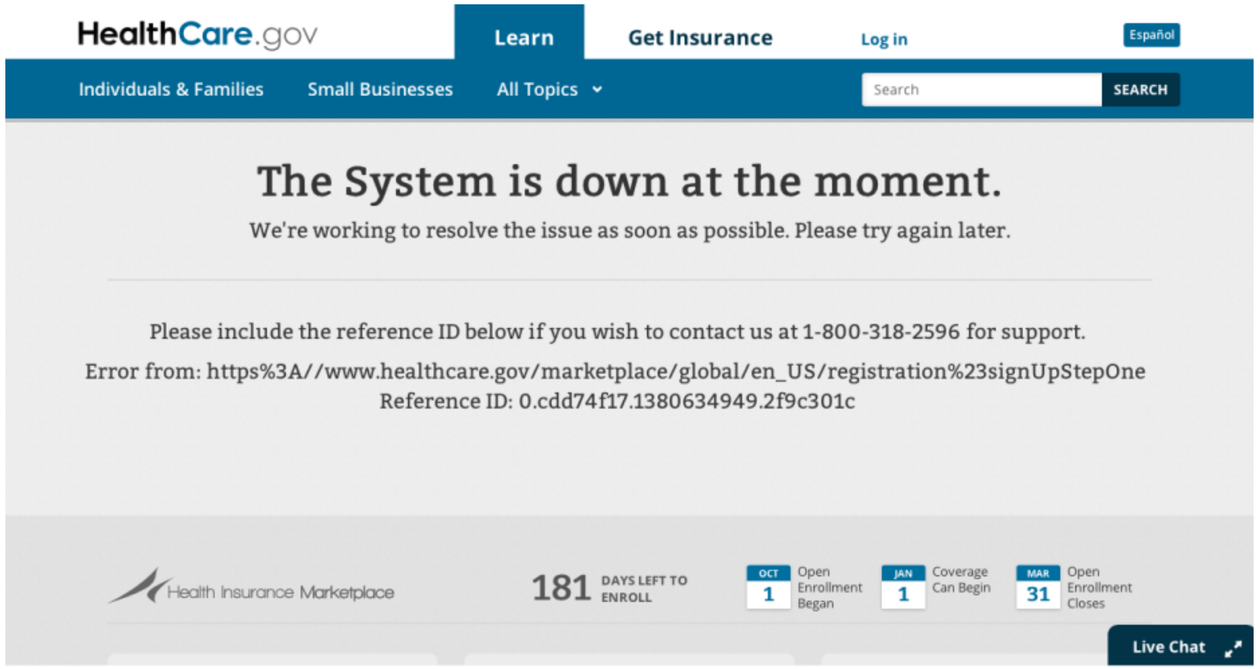
CTO Program Scalability

A Model of Scalability

Why do we care about Scalability?

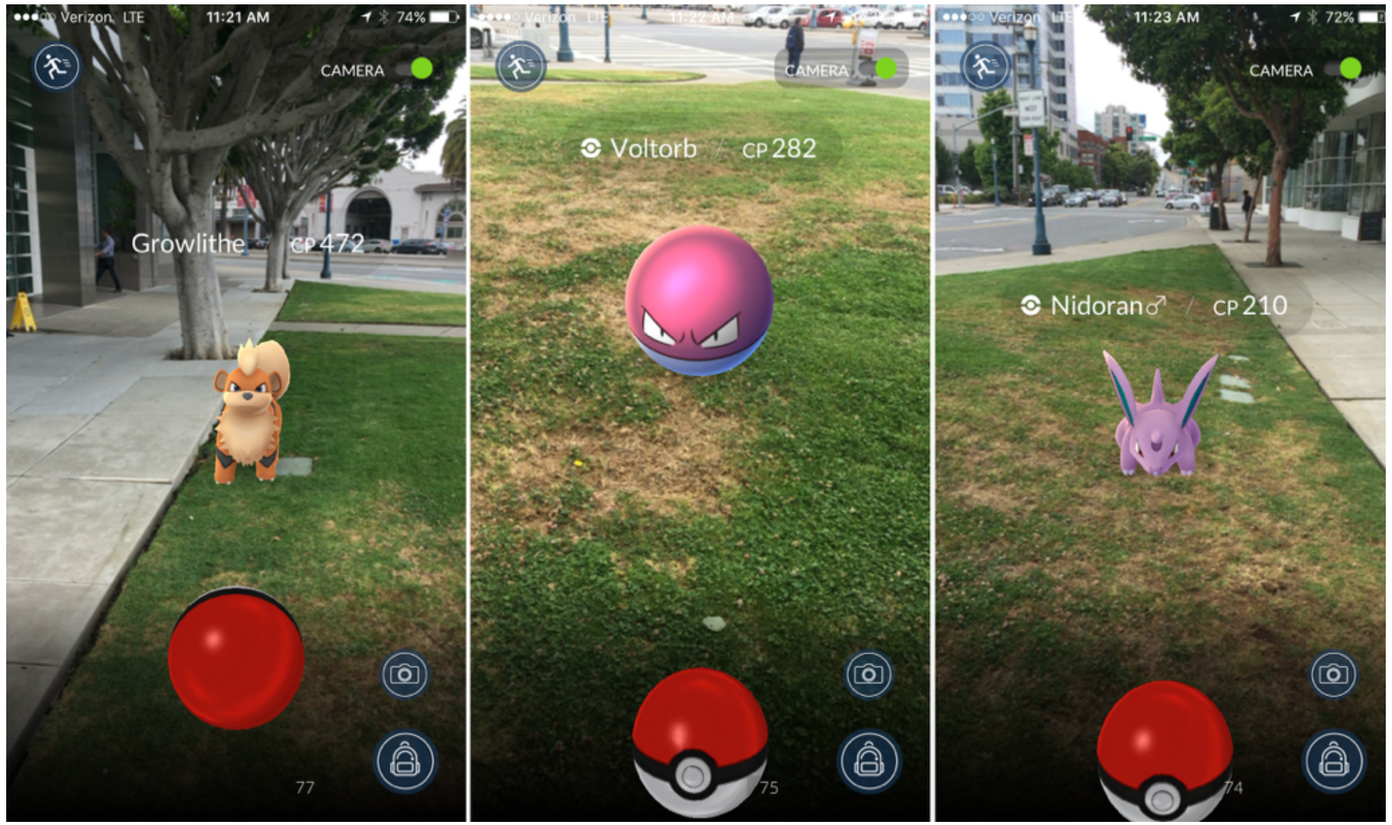
"We had millions of Friendster members begging us to get the site working faster so they could log in."

Why do we care about Scalability?



If any part of the web of systems fails to work properly, it could lead to a traffic jam blocking most users from the marketplace. That's just what happened: On Oct. 2, officials identified a bottleneck where those systems intersect at a software component sold by Oracle Corp. that still hasn't been cleared.

Why do we care about Scalability?



The global rollout of Pokémon GO has been halted. If you've already managed to install the game you've probably come across server delay screens already Well, that's the culprit. Pokémon GO has been so popular that its developer Niantic Labs' servers can't keep up with demand.

Definition of Scalability

- **Systems service requests for work (HTTP, DB query, file, computation)**
- **More of "X" leads to more request handling**
  - X could be machines, databases, CPUs, hard drives
  - Clear relationship between X and number of requests
  - Perfect Scaling:
    - 1 machine gives you 1000 r/s, installing 10 machines give you 10000 r/s
- **Bottlenecks are identified**
  - What part of the system are they located?
  - Do we have a mitigation strategy?
- **Metrics**
  - Are we measuring performance?

Bottlenecks

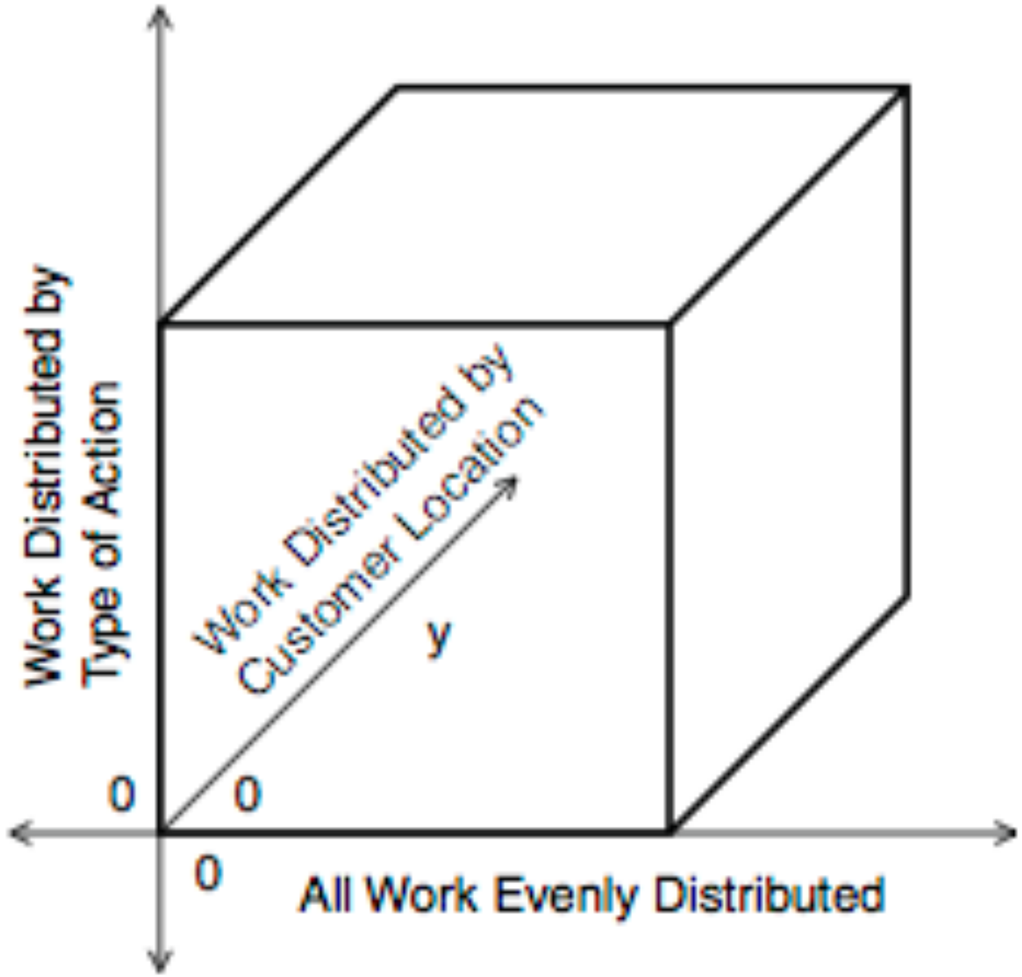
- **As we add X to service more requests do we start breaking external systems/dependencies**
- **Classic Web Startup Bottlenecks:**
  - More users mean more data => Matching algorithm slows to a crawl (Friendster)
  - More app server machines handles more requests => Database queries slow down
  - More developers means more modules => Response time for bugs slows down

Where the Industry Sees Bottlenecks

Scaling for More Work

- What are ways that we can scale up a system to handle more requests for work?
- What type of work are we getting?

The AKF Scale Cube



X-Axis Scale

- Cloned Identical Services
- No difference between increased number of service handlers
- Typically referred to as a “resource pool”
  - App Server Pool
- “heroku ps:scale web=2”
- Easiest to scale in this direction

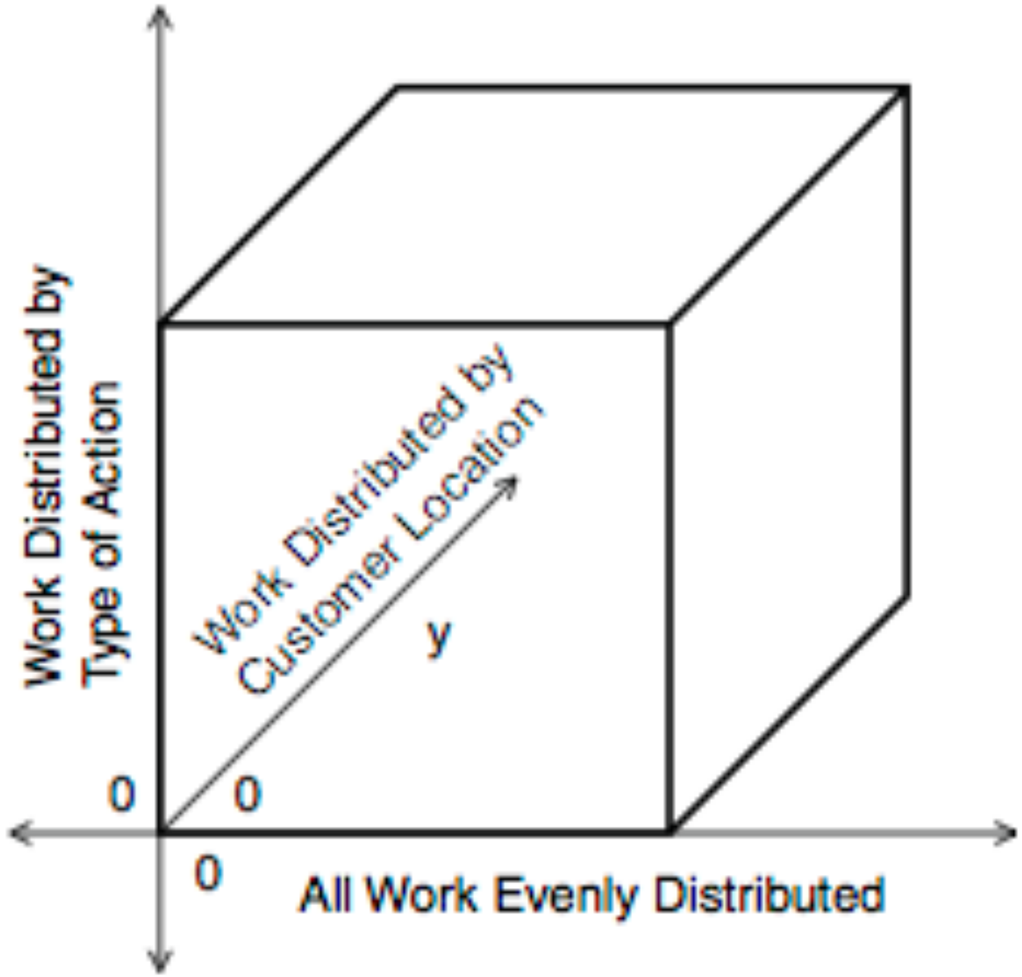
Y-Axis

- Split Services by Responsibility, Action or Data
- Specialization
  - Conversion from generalists to specialists
- Using Node.js for WebSockets and Rails for your CRUD app

Z-Axis

- **Split Services by Requestor**
- **Customer splits**
  - Geography
  - Bucketing / Sharding (ID)
  - Distributed System
- **Amazon West and Amazon East**

The AKF Scale Cube

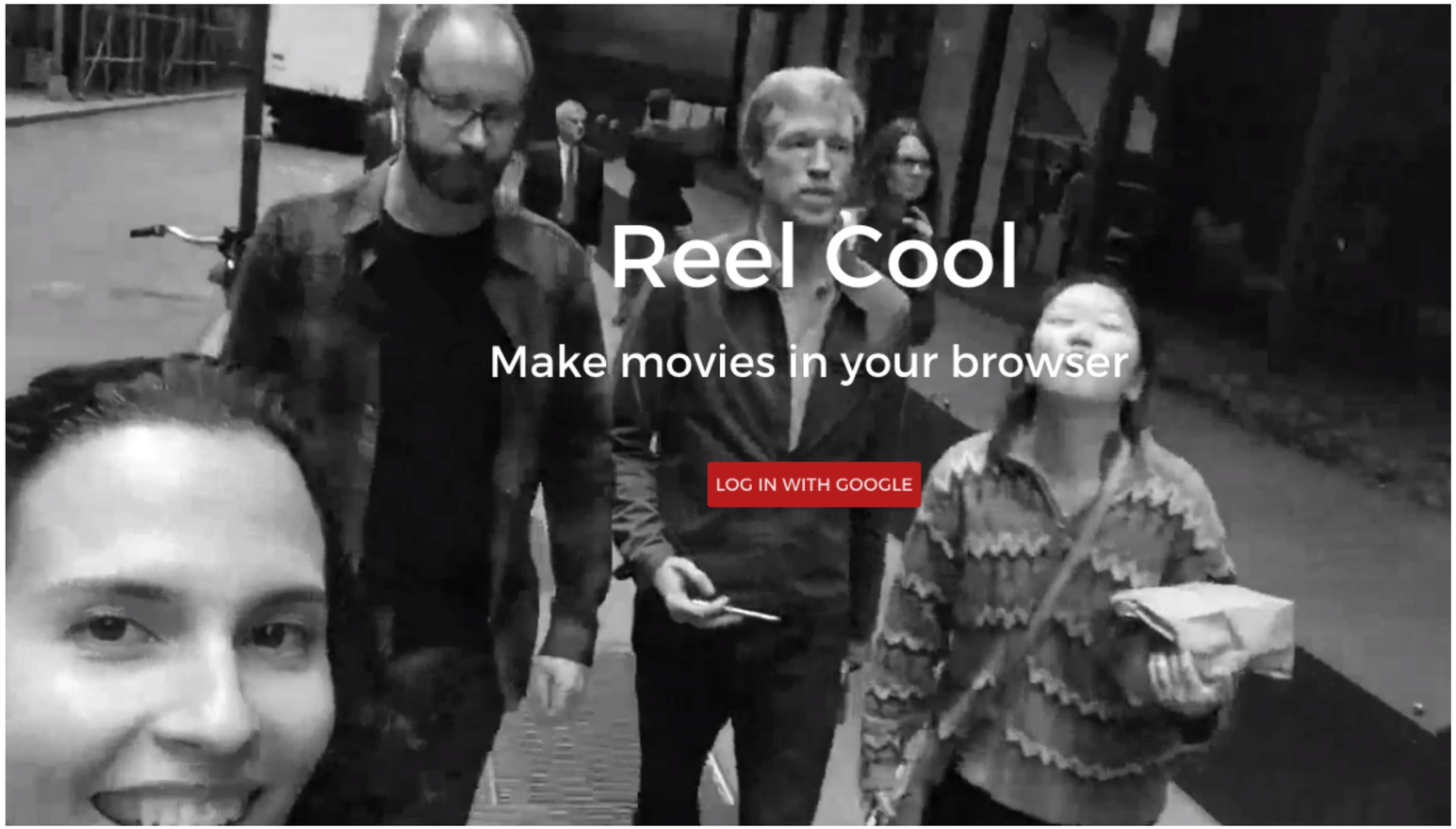



WHOLE FOODS MARKET
WHOLE FOODS

Waiting Lines

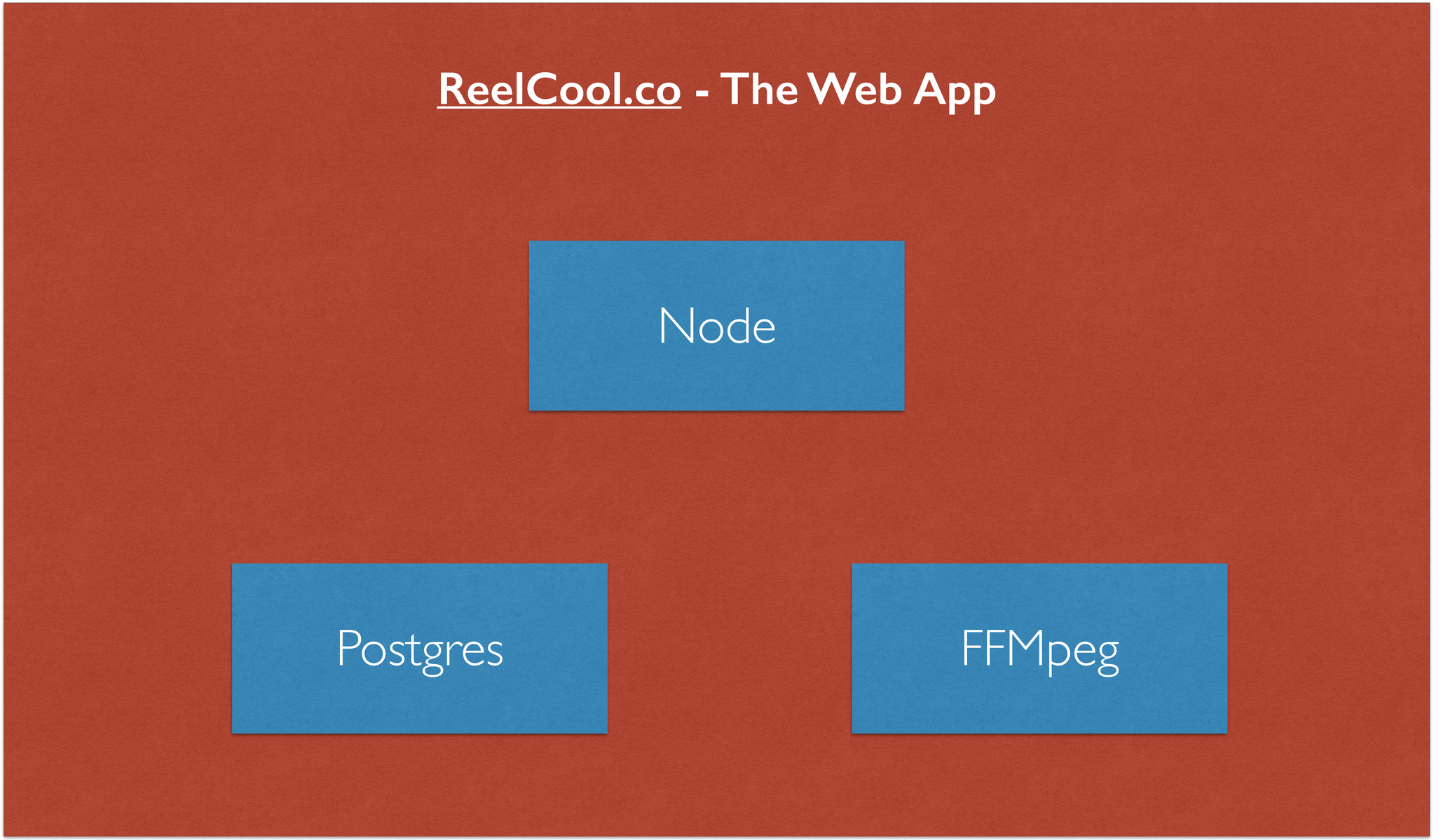
- Add Checkout Lanes (X-Axis)
- Add Customer Service/Returns Desk (Y-Axis)
- Add Express Lane (Y-Axis)
- Add Self-Service Checkout (Y-Axis)
- Add More Stores (Z-Axis)
- Add Delivery (Z-Axis)

Scalability Challenges for the Web



ReelCool.co - The Web App

ReelCool.co - The Web App

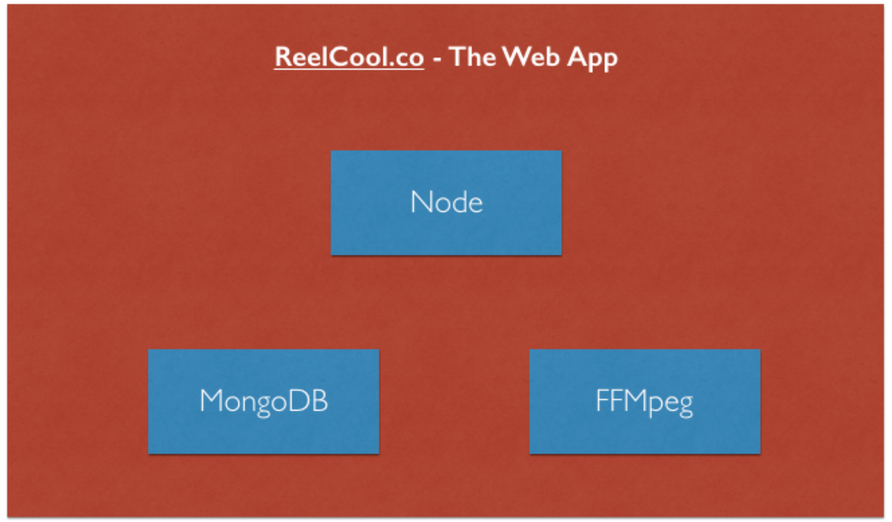


Amazon Managed Server inside Amazon Network

Amazon EC2

EC2 CPU

File System (Storage)



ReelCool - X-Axis

- **Run on more instances of EC2 (more CPU for FFMpeg)**
- **Each EC2 Instance duplicates everything**
  - Node.js Instance
  - FFMpeg Compiler
  - Share a database

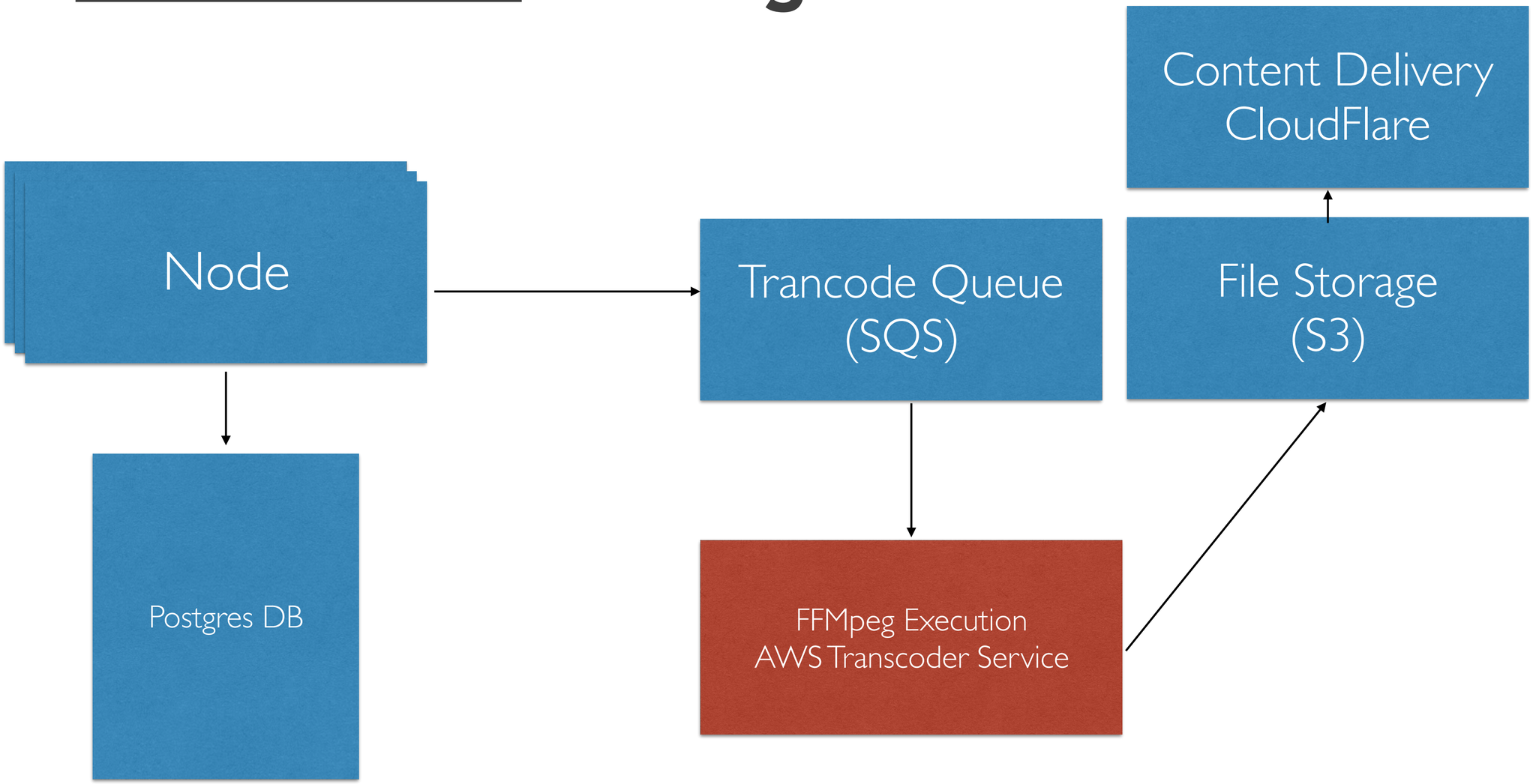
ReelCool Y-Axis

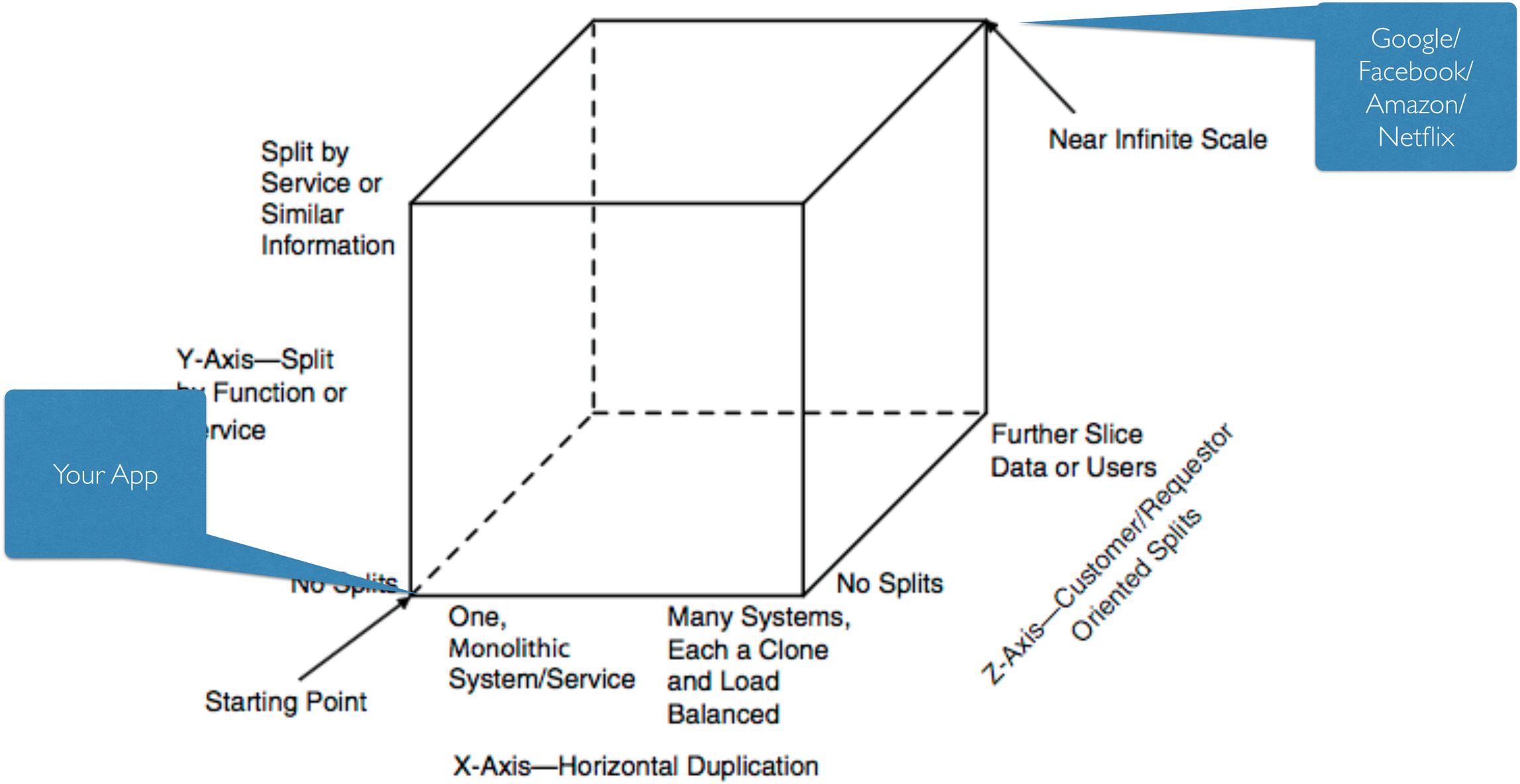
- **Split Services By Responsibility**
- **Services:**
  - Process Uploaded Video/Audio
  - Generate Download Files
  - Host Files
- **Amazon has solutions to all these things:**
  - Process Uploaded Video/Audio - EC2
  - Generate Download Files - Transcoder Service
  - Host Files - S3: Simple Storage Service

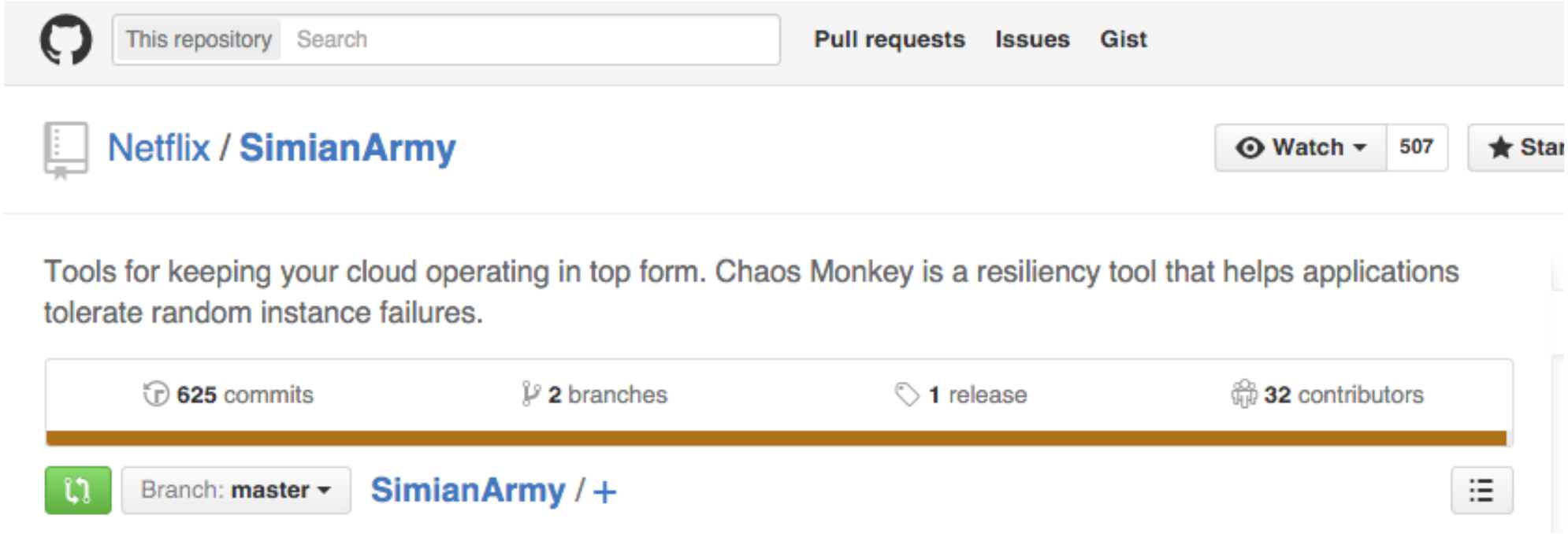
ReelCool Z-Axis

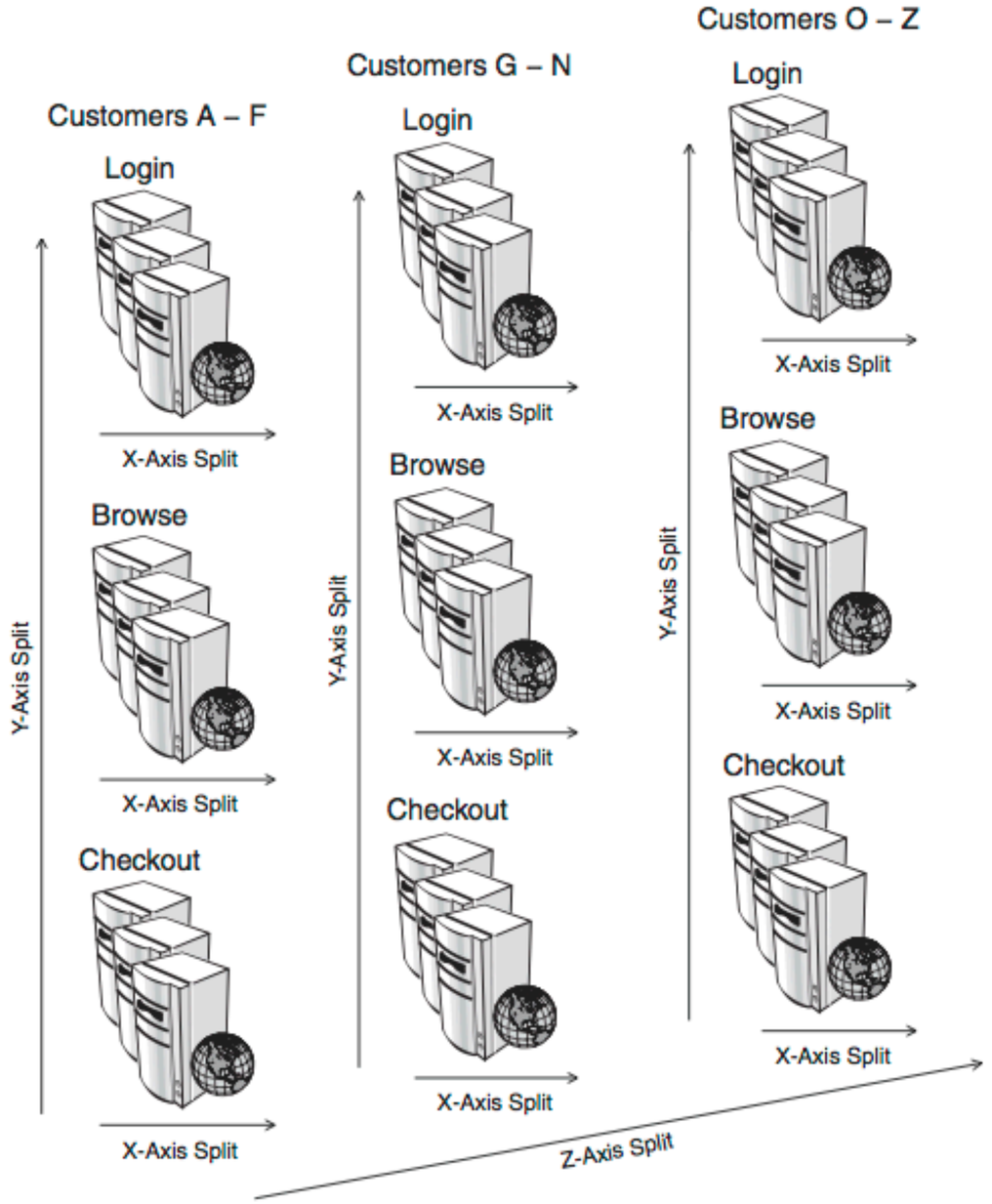
- Uploads/downloads are slow
- Content Delivery Networks
- Cloudflare/CDNjs
- Moving GB around the internet can still be a "slow" part of your app

ReelCool.co Scaling Points




Google/
Facebook/
Amazon/
Netflix
Near Infinite Scale
Split by
Service or
Similar
Information
Y-Axis—Split
by Function or
Service
Your App
Further Slice
Data or Users
Z-Axis—Customer/Requestor
Oriented Splits
No Splits
No Splits
One,
Monolithic
System/Service
Many Systems,
Each a Clone
and Load
Balanced
Starting Point
X-Axis—Horizontal Duplication


This repository
Search
Pull requests
Issues
Gist
Netflix / SimianArmy
Watch
507
Tools for keeping your cloud operating in top form. Chaos Monkey is a resiliency tool that helps applications tolerate random instance failures.
625 commits
2 branches
1 release
32 contributors
Branch: master
SimianArmy / +

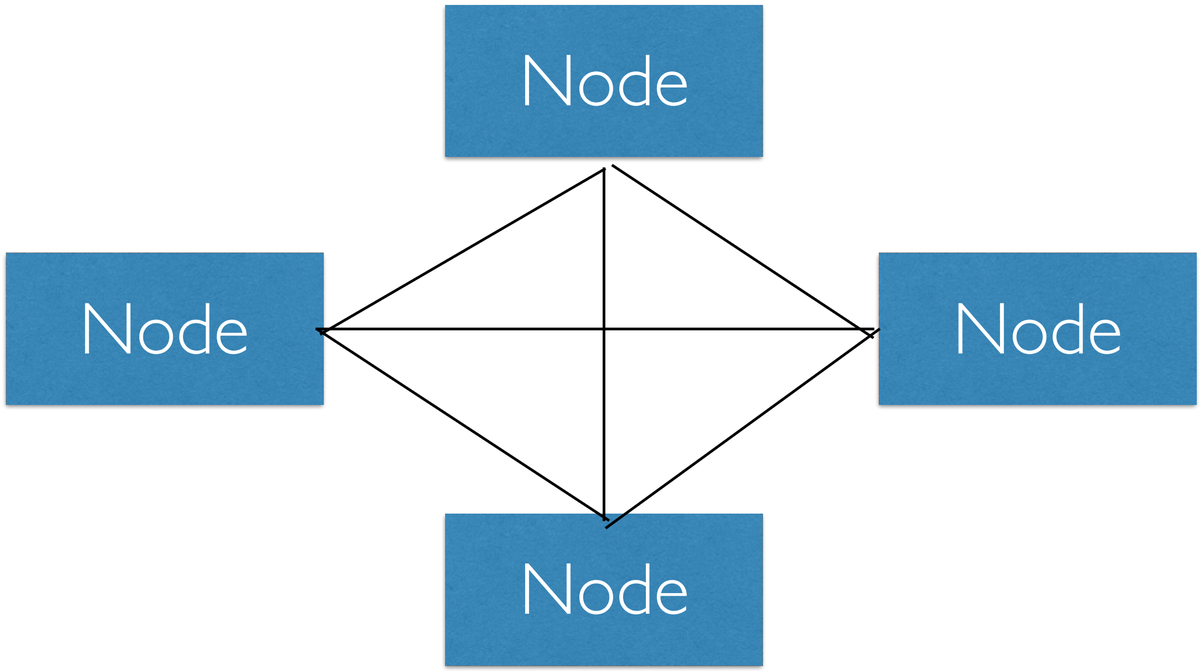

Customers A – F
Login
Browse
Checkout
Customers G – N
Login
Browse
Checkout
Customers O – Z
Login
Browse
Checkout
X-Axis Split
Y-Axis Split
Z-Axis Split

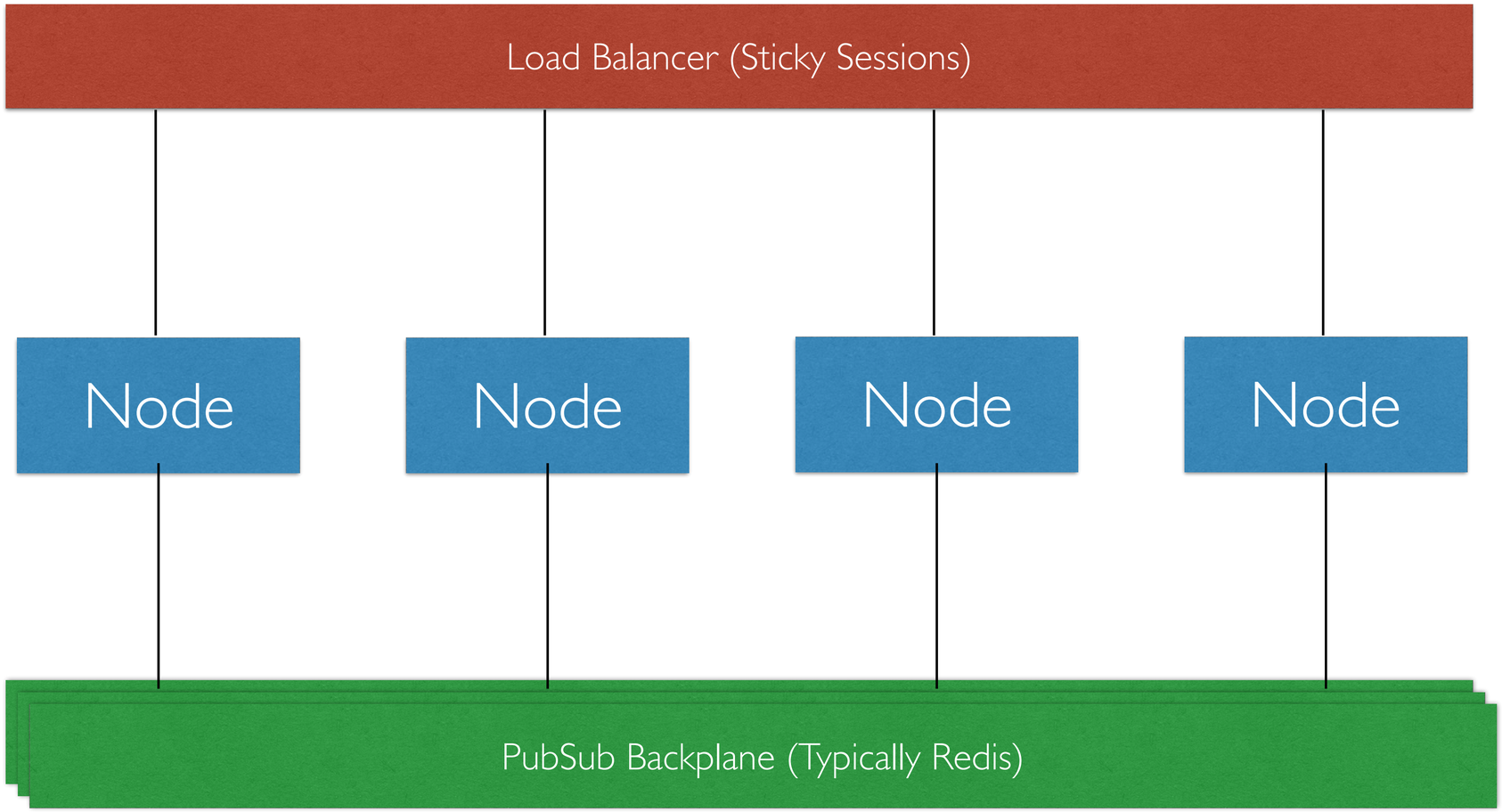
Databases

- **Databases have different types of challenges because:**
- **State is hard**
- **People expect their data to be correct**
- **Data changes are non-local**
  - More than one table, one machine, one data center
  - Data interacts in hard to predict ways (Tweet fanning out to 2M people)

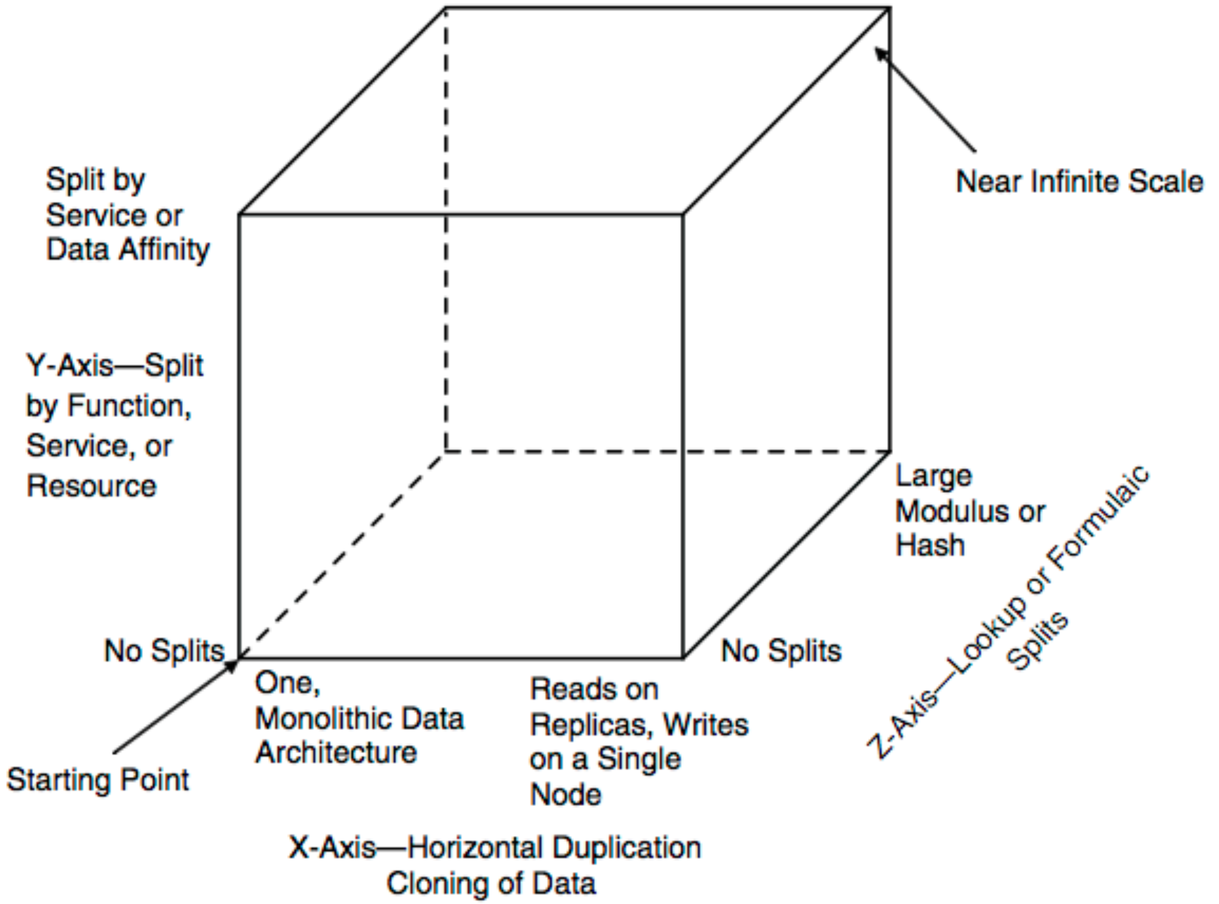
State is hard (socket.io)

- socket.io
  - State: Which Node.js Server holds the connection to which Socket
  - Node.js Servers need to communicate now to share which sockets they hold
  - Three challenging problems:
    - I want to broadcast something to all sockets in a "room"
    - A user reconnects, I want to make sure they connect to the same Node server
    - Communication Overhead - Grows at n^2 (where n = number of Node.js servers)


Node
Node
Node
Node


Load Balancer (Sticky Sessions)
Node
Node
Node
Node
PubSub Backplane (Typically Redis)

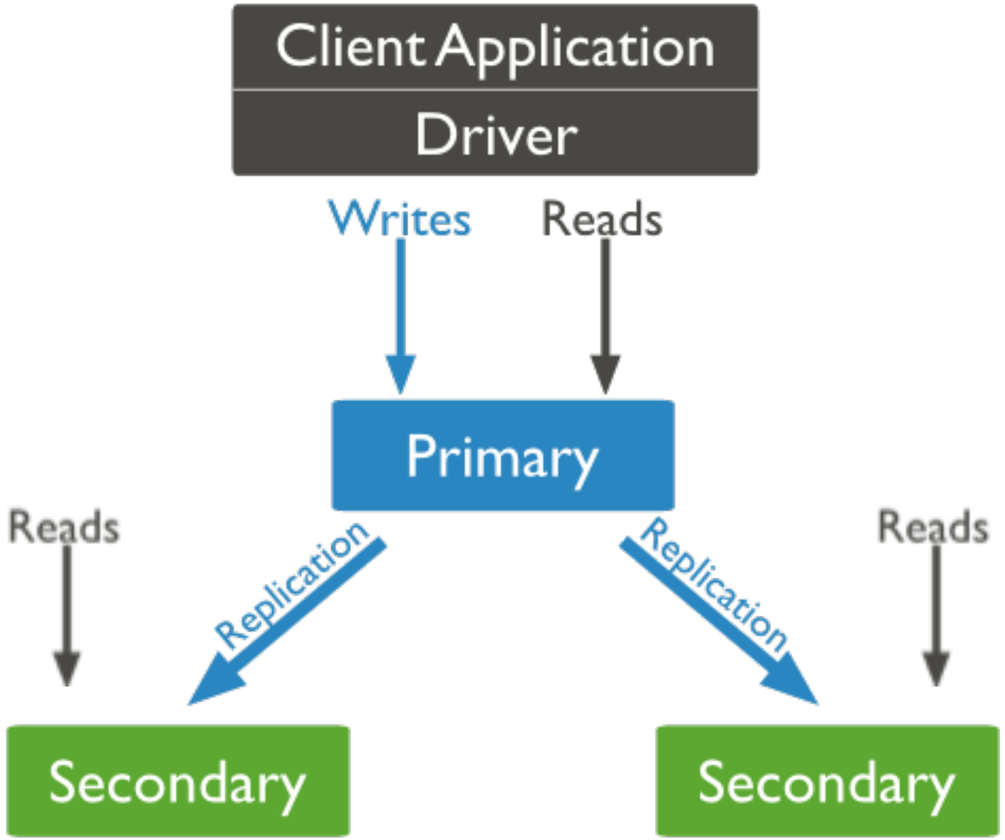
AKF for Databases



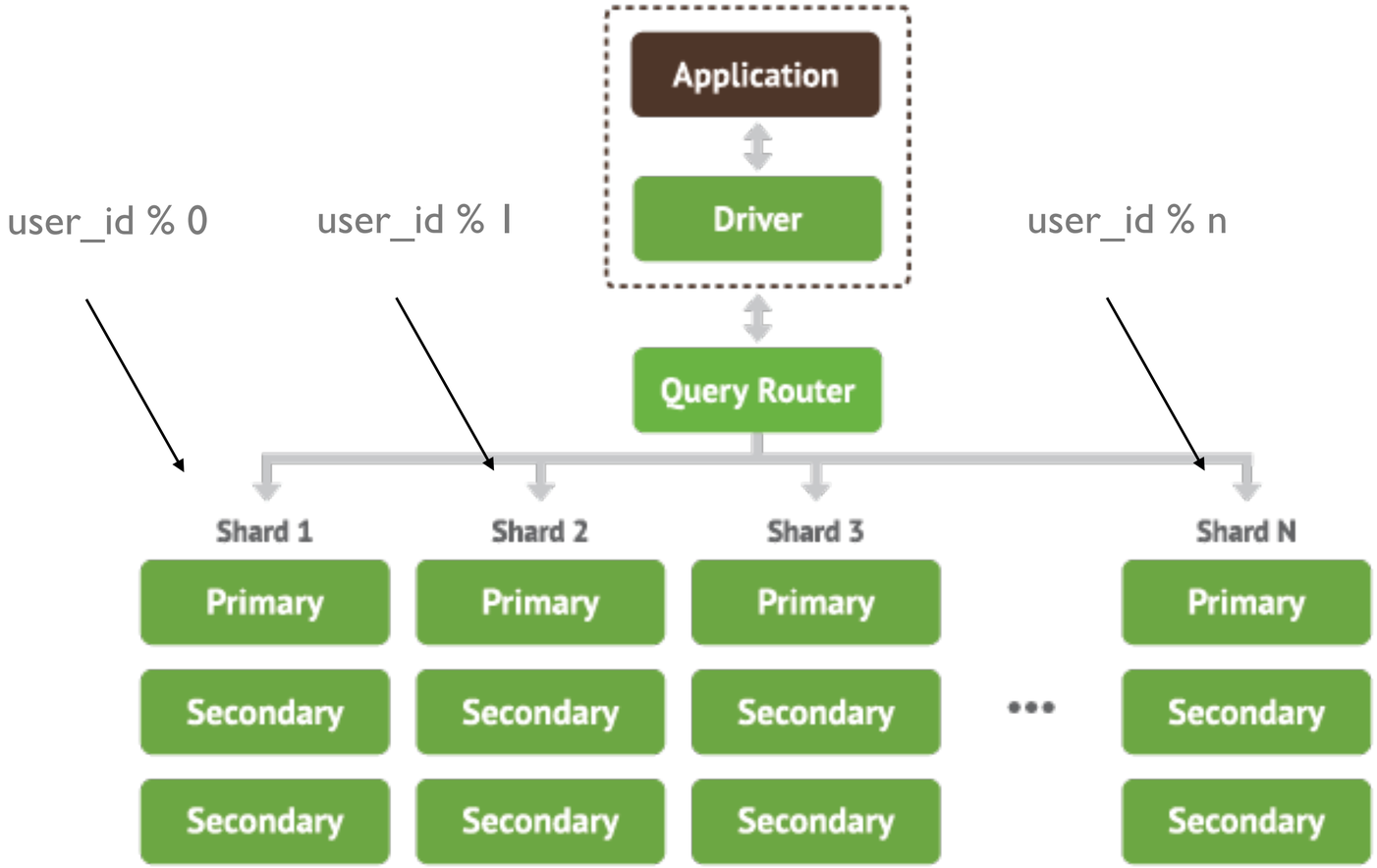
AKF for Databases

- **X-Axis**
  - Read Replicas
- **Y-Axis**
  - Postgres/Mongo
  - Redis
  - Memcached
- **Z-Axis**
  - Sending data to different databases based on the information itself
  - Referred to as sharding

Read Replicas (X-Axis)

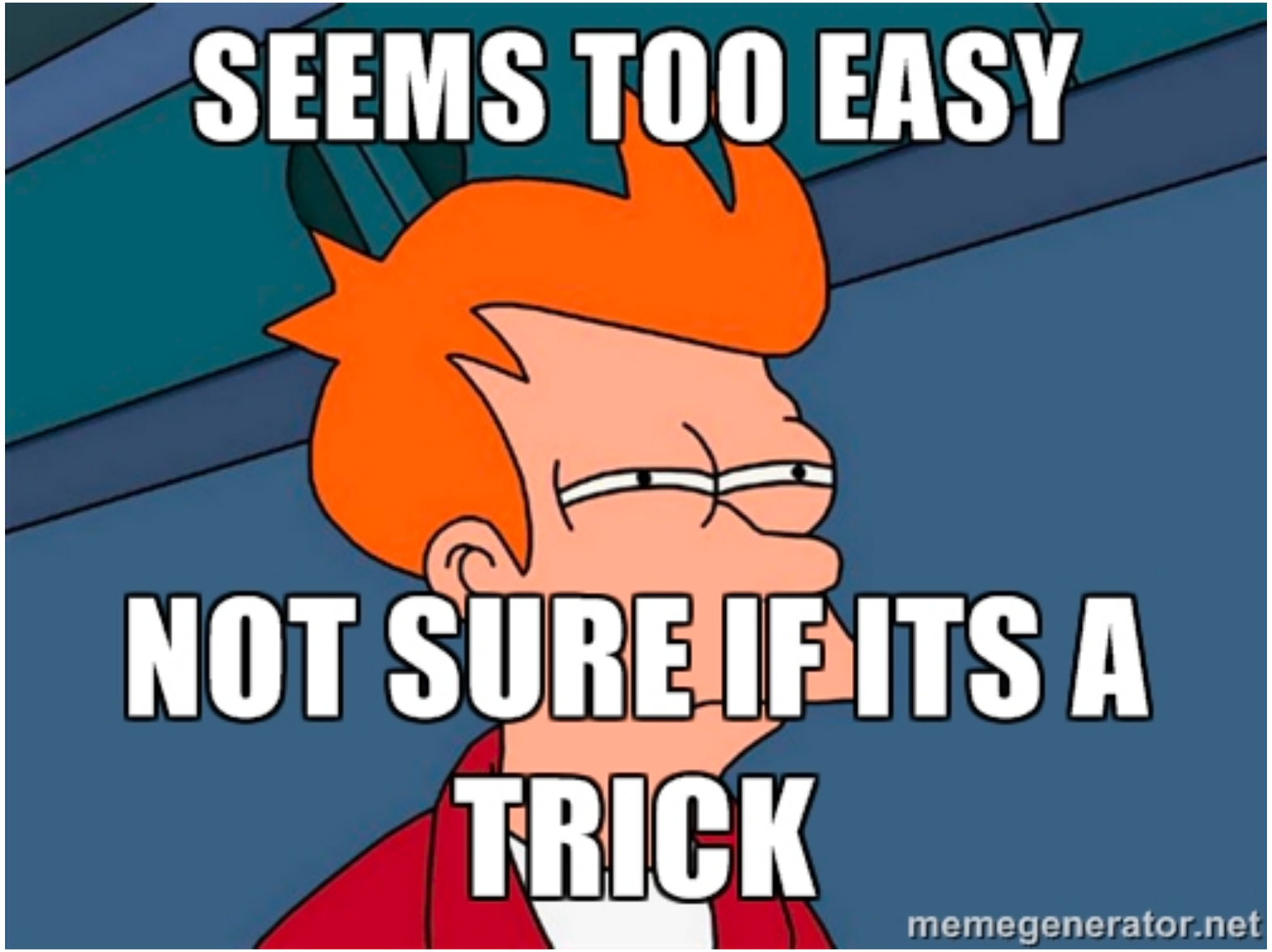


Z-Axis


Application
Driver
Query Router
user_id % 0
user_id % 1
user_id % n
Shard 1
Shard 2
Shard 3
Shard N
Primary
Primary
Primary
Primary
Secondary
Secondary
Secondary
Secondary
Secondary
Secondary
Secondary
Secondary

Benefits

- X-Axis > Infinite Read Scalability
- Z-Axis > Infinite Storage Scalability


SEEMS TOO EASY
NOT SURE IF ITS A TRICK
memegenerator.net

ACID

- **For years, database developers struggled to make their DB systems "ACID"**
  - Atomic: Everything in a transaction succeeds or the entire transaction is rolled back.
  - Consistent: A transaction cannot leave the database in an inconsistent state.
  - Isolated: Transactions cannot interfere with each other.
  - Durable: Completed transactions persist, even when servers restart etc.

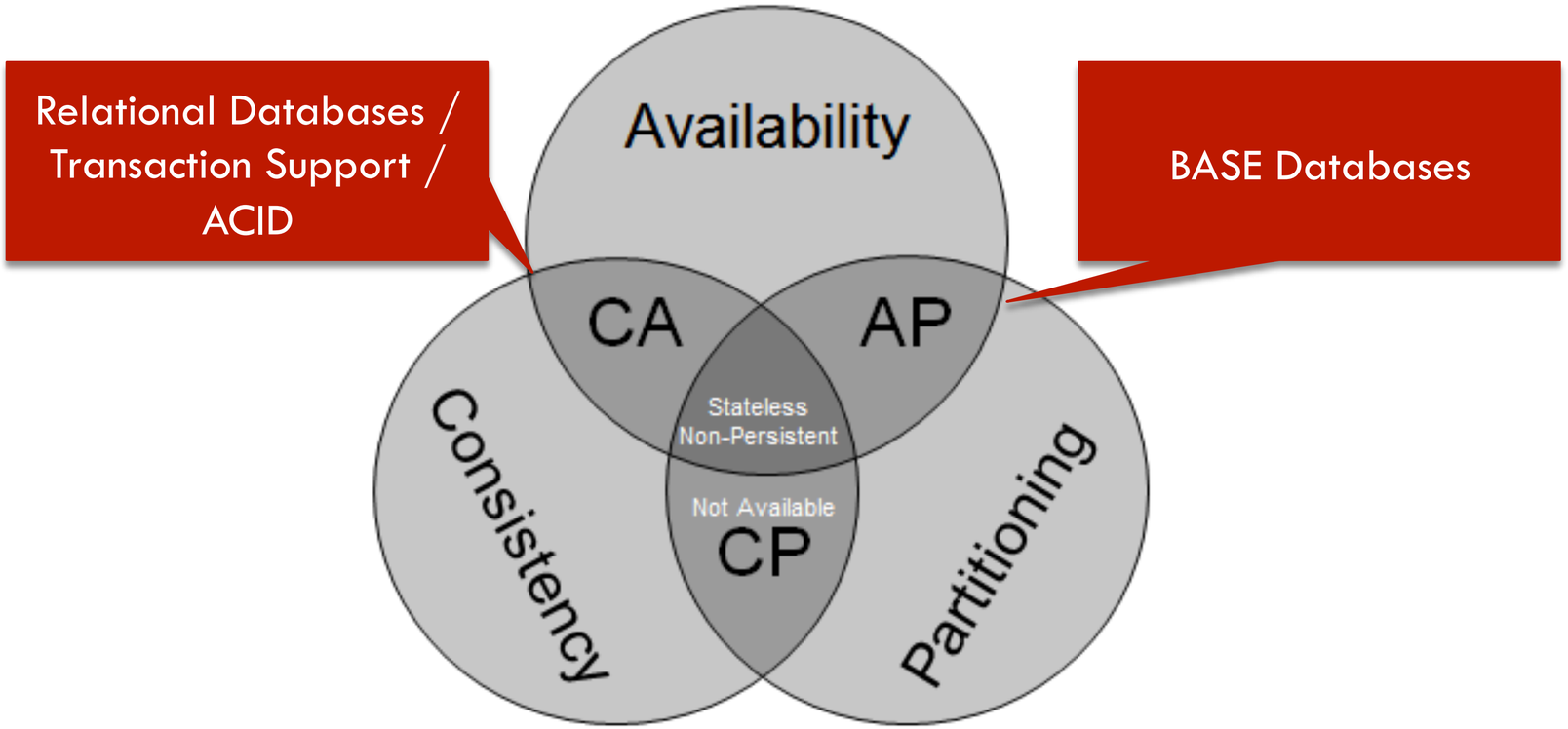
What's after Acid?

What's after Acid?

- BASE
  - Basic Availability
    - The database appears to be available even though the
  - Soft-state
  - Eventual Consistency

CAP Theorem

- Consistency, Availability, Partition Tolerance (choose 2)



Implications of the AKF Cube

- Need for Repeatability
- Microservices

Repeatability

- X-axis scaling requires repeatability
- Scalability requires repeatable inputs
- DevOps
  - Heroku
  - Docker

Heroku

- **I. Codebase**
  - One codebase tracked in revision control, many deploys
- **II. Dependencies**
  - Explicitly declare and isolate dependencies
- **III. Config**
  - Store config in the environment
- **IV. Backing Services**
  - Treat backing services as attached resources
- **V. Build, release, run**
  - Strictly separate build and run stages
- **VI. Processes**
  - Execute the app as one or more stateless processes
- **VII. Port binding**
  - Export services via port binding
- **VIII. Concurrency**
  - Scale out via the process model
- **IX. Disposability**
  - Maximize robustness with fast startup and graceful shutdown
- **X. Dev/prod parity**
  - Keep development, staging, and production as similar as possible
- **XI. Logs**
  - Treat logs as event streams
- **XII. Admin processes**
  - Run admin/management tasks as one-off processes

Docker

- Companies want to be able to ship their applications around their many data center machines
- They used to use Virtual Machines to accomplish this (the virtual machine says I need 2 CPUs, 500 GB of Disk, 1 Network Port)
- Problem: Virtual Machine Spin Up/Down is Slow

Docker

- Containers are lightweight virtual machines
- Isolate:
  - Processes
  - Disk
  - Network
- Allows Repeatability of the environment

Docker

23 lines (17 sloc) | 851 Bytes

Raw | Blame | History

```
1  FROM ubuntu
2  MAINTAINER Kimbro Staken
3
4  RUN apt-get install -y python-software-properties python python-setuptools ruby rubygems
5  RUN add-apt-repository ppa:chris-lea/node.js
6  RUN echo "deb http://us.archive.ubuntu.com/ubuntu/ precise universe" >> /etc/apt/sources.list
7  RUN apt-get update
8  RUN apt-get install -y nodejs
9
10 RUN apt-key adv --keyserver keyserver.ubuntu.com --recv 7F0CEB10
11 RUN echo "deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen" | tee -a /etc/apt/sources.list.d/10gen.list
12 RUN apt-get -y update
13 RUN apt-get -y install mongodb-10gen
14
15 RUN easy_install supervisor
16 RUN echo_supervisord_conf > /etc/supervisord.conf
17 RUN printf "[include]\nfiles = /var/www/Supervisorfile\n" >> /etc/supervisord.conf
18
19 ADD . /var/www
20
21 RUN cd /var/www ; npm install
22
23 CMD ["/usr/local/bin/supervisord", "-n", "-c", "/etc/supervisord.conf"]
```

- **If you're on a production machine and running bash commands, you're doing it wrong.**
- **At the minimum, use bash scripts to set up your production servers**
  - Also your dev boxes and dev environments (Vagrant)

Y-Axis Split

- Another implication of Y-Axis Splits...

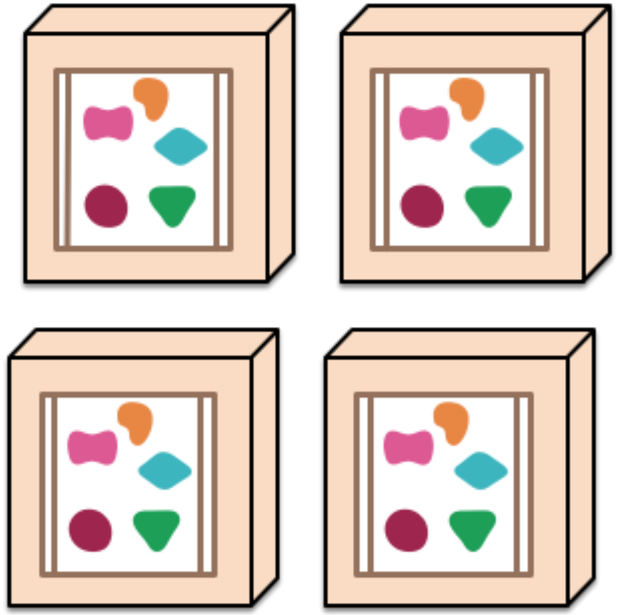
Microservices

- Y-Axis Splits become separate services
- Distribute the governance
- Distribute the data management
- Node.js as Orchestrator
- Deployment Infrastructure is Key

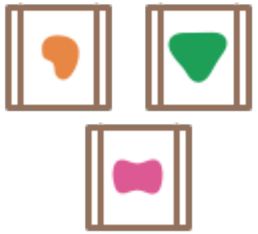
Microservices

A monolithic application puts all its functionality into a single process...

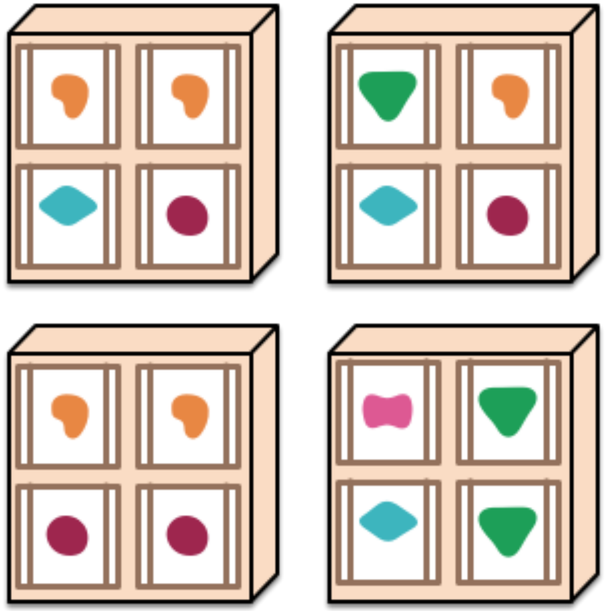
... and scales by replicating the monolith on multiple servers



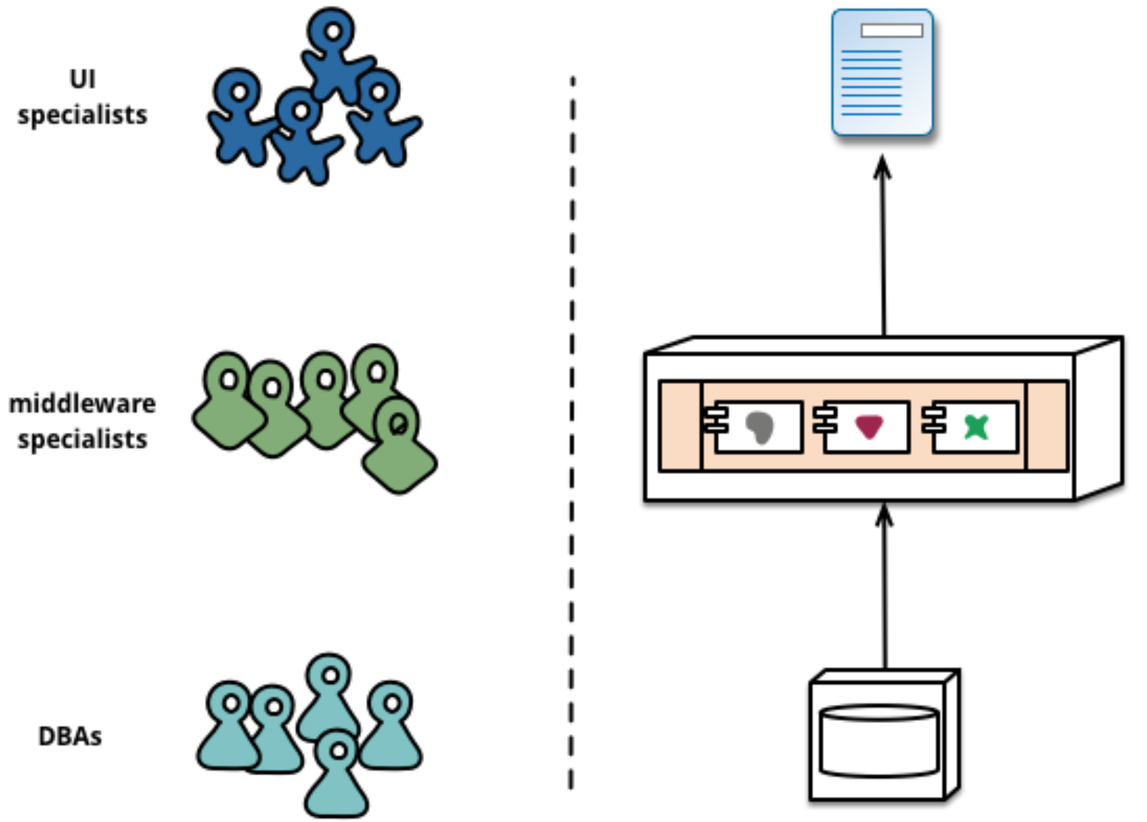
A microservices architecture puts each element of functionality into a separate service...



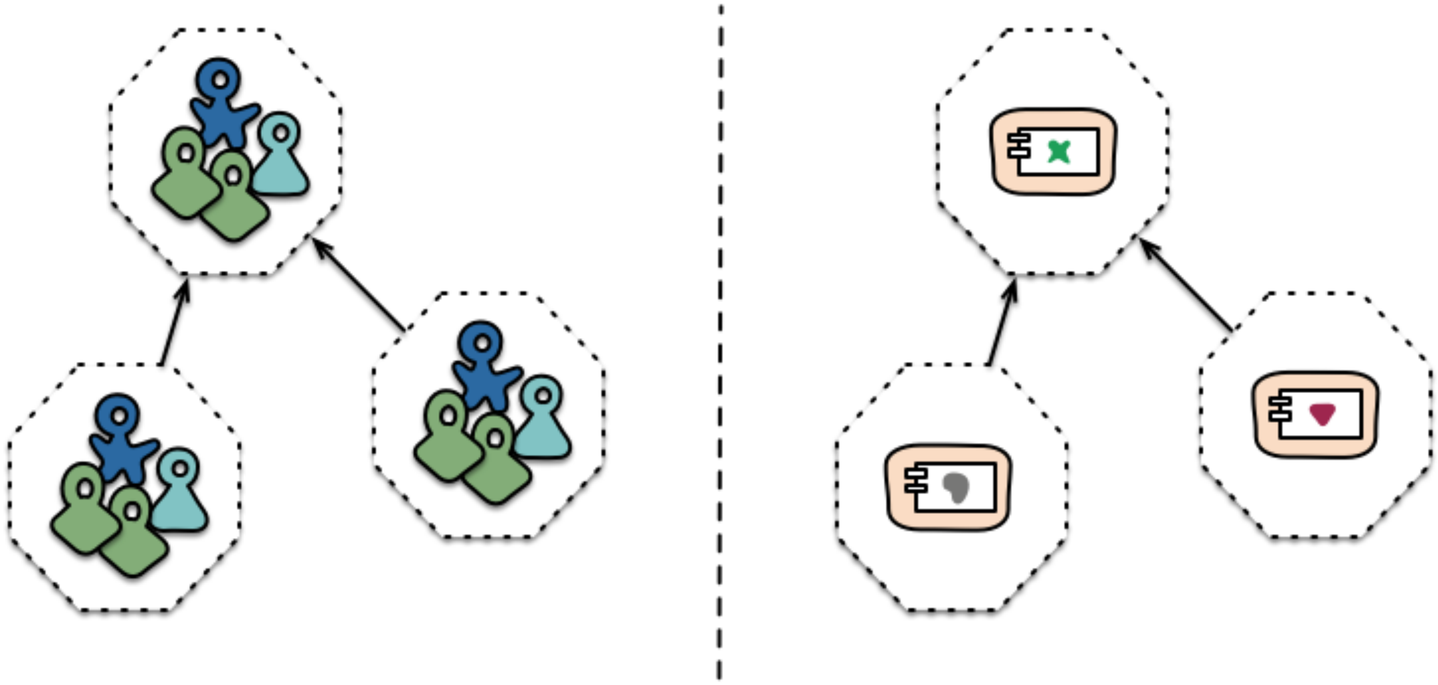
... and scales by distributing these services across servers, replicating as needed.



Microservices



Microservices



Cross-functional teams...

... organised around capabilities
Because Conway's Law