NoSQL like There is No Tomorrow

Khawaja

Head of Engineering, NoSQL

Swaminathan Sivasubramanian

GM, NoSQL

@swami_79

@ksshams

how can you build your own DynamoDB Scale service?

@swami_79

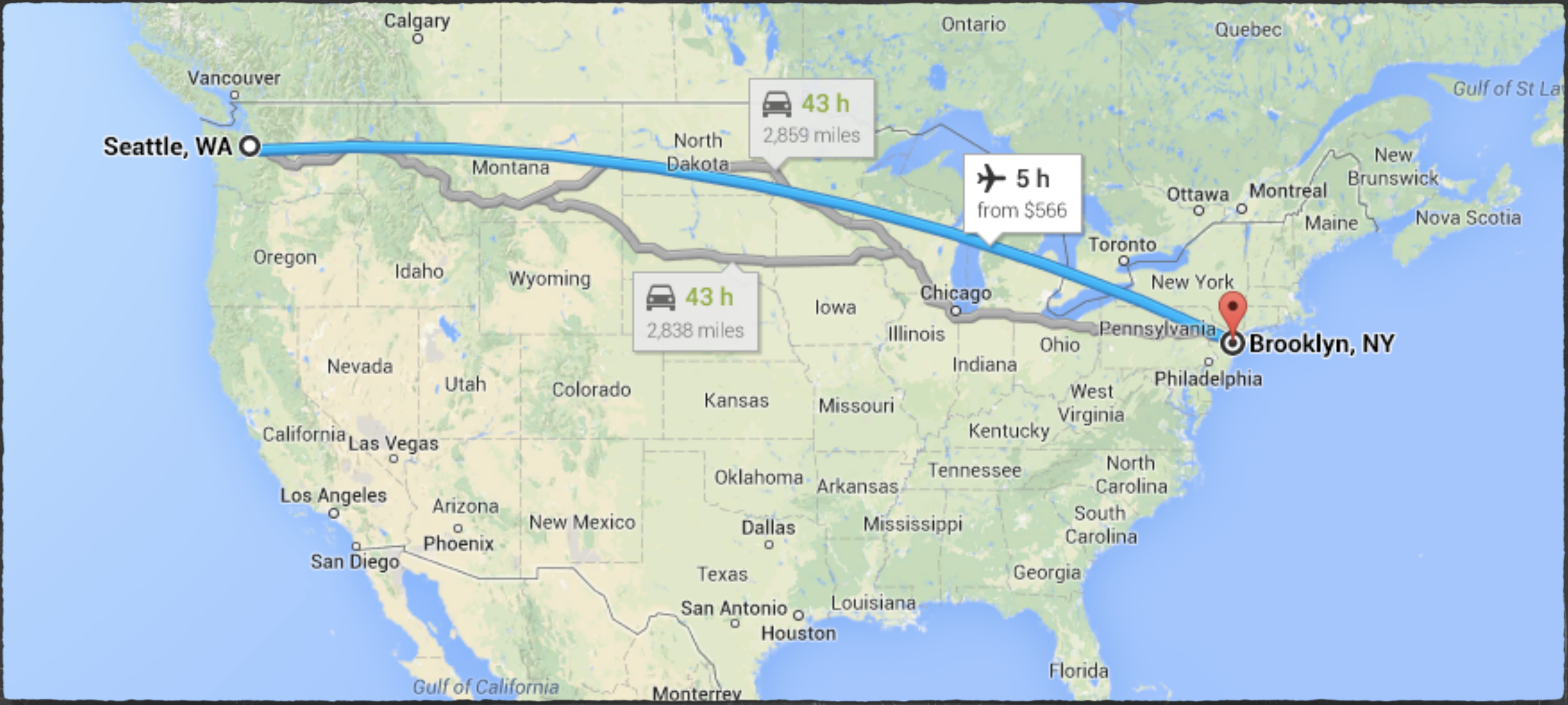
@ksshams

let's start with a **story** about a little company called **amazon.com**

episode 1

once upon a time…
(in 2000)

a few thousand miles away… (seattle)



@swami_79 @ksshams

amazon.com - a rapidly growing Internet based retail business relied on **relational databases**

we had 1000s of independent services

@swami_79 @ksshams

each service managed its state in RDBMs

RDBMs are actually kind of cool

first of all... SQL!!

so it is easier to query..

@swami_79 @ksshams

easier to learn

as versatile as a swiss army knife

key-value access

complex queries

analytics

transactions

RDBMs are too similar to Swiss Army Knives

but sometimes.. swiss army knifes.. can be more than what you bargained for

partitioning

easy

re-partitioning

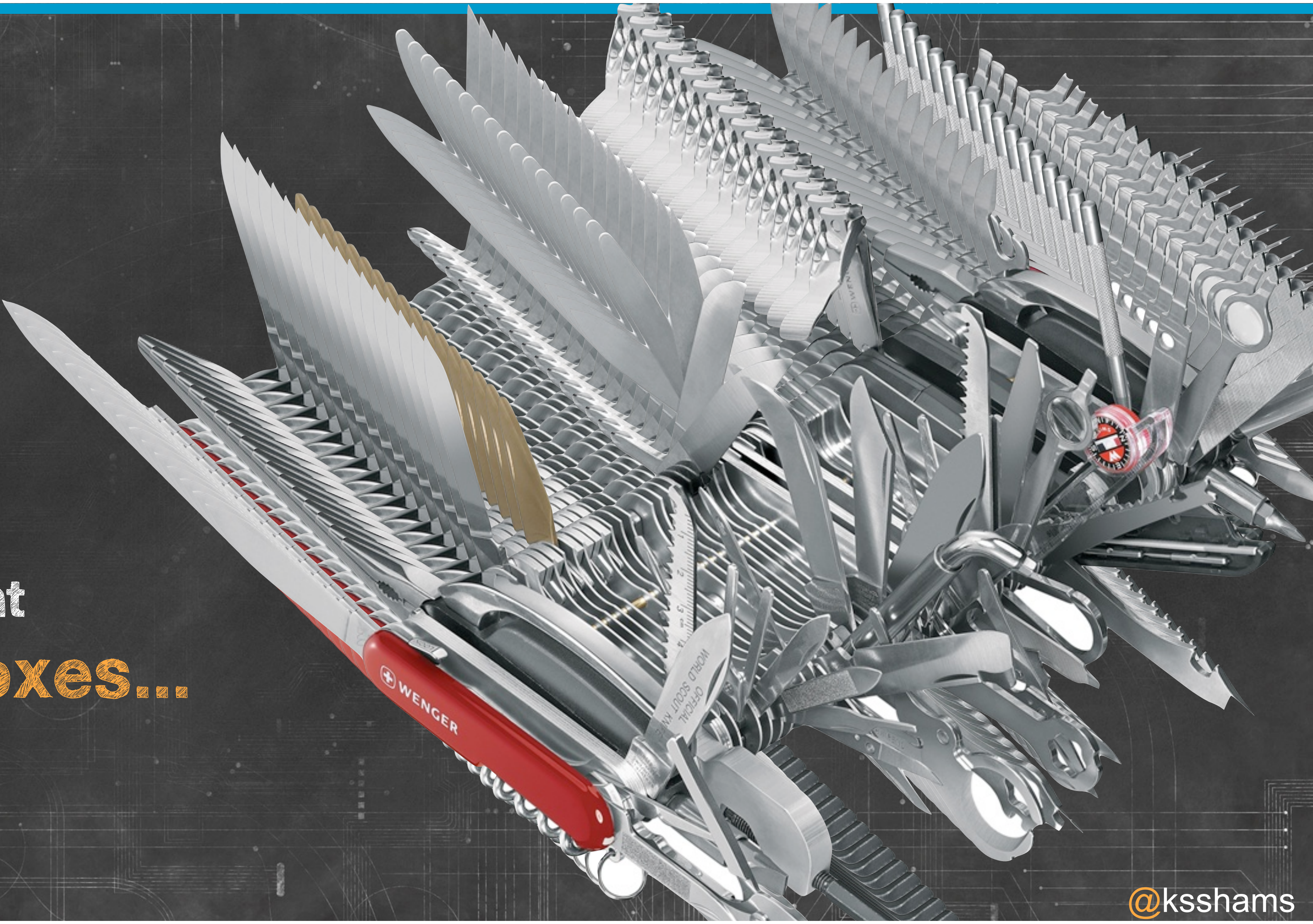
hard..

@swami_79

@ksshams

so we bought

bigger boxes...



benchmark new hardware

migrate to new hardware

Q4 was hard-work at Amazon

repartition databases

pray …

@swami_79

@ksshams

RDBMs availability challenges..

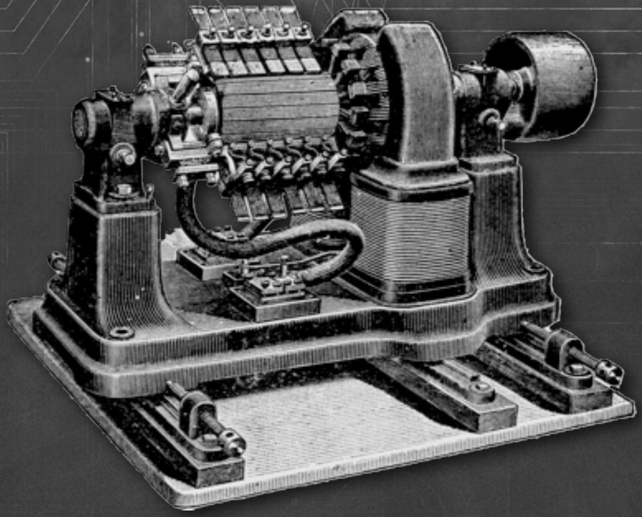
@swami_79 @ksshams

episode 2

then.. (in 2005)

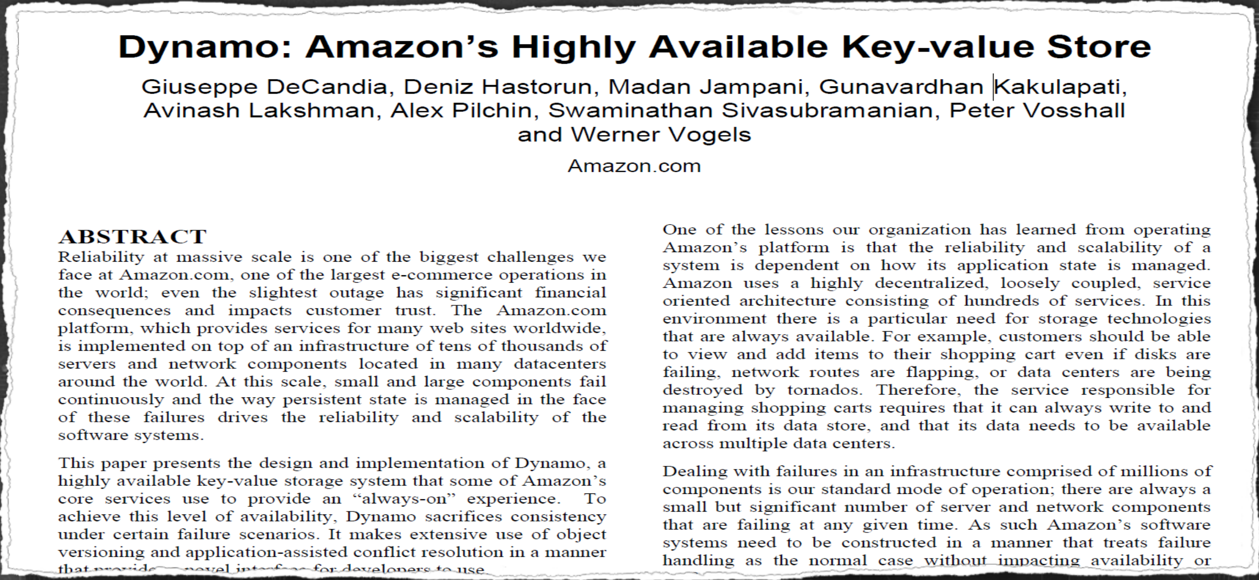
amazon dynamo

predecessor to dynamoDB



replicated DHT with consistent hashing
optimistic replication
“sloppy quorum”
anti-entropy mechanism
object versioning

specialist tool :
- limited querying capabilities
- simpler consistency



Dynamo: Amazon’s Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

Amazon.com

ABSTRACT

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

This paper presents the design and implementation of Dynamo, a highly available key-value storage system that some of Amazon’s core services use to provide an “always-on” experience. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios. It makes extensive use of object versioning and application-assisted conflict resolution in a manner that provides a novel interface for developers to use.

One of the lessons our organization has learned from operating Amazon’s platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers.

Dealing with failures in an infrastructure comprised of millions of components is our standard mode of operation; there are always a small but significant number of server and network components that are failing at any given time. As such Amazon’s software systems need to be constructed in a manner that treats failure handling as the normal case without impacting availability or

@swami_79 @ksshams

dynamo had many benefits

- higher availability
  - we traded it off for eventual consistency
- incremental scalability
  - no more repartitioning
  - no need to architect apps for peak
  - just add boxes
- simpler querying model ==>> predictable performance

but dynamo was not perfect...

lacked strong consistency

but dynamo was not perfect…

scaling was easier, but…

but dynamo was not perfect...

steep learning curve

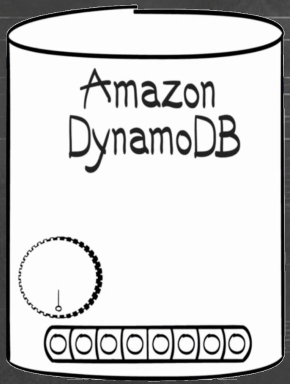
but dynamo was not perfect...

dynamo was a product ... ==>> not a service...

episode 3

then.. (in 2012)

DynamoDB



- NoSQL database
- fast & predictable performance
- seamless scalability
- easy administration

"Even though we have years of experience with large, complex NoSQL architectures, we are happy to be finally out of the business of managing it ourselves." - Don MacAskill, CEO

SmugMug

@swami_79

@ksshams

build services not software!!

@swami_79

@ksshams

amazon.com’s experience with services

how do you create a successful service?

with great services, comes great responsibility

@swami_79

@ksshams

DynamoDB Goals and Philosophies

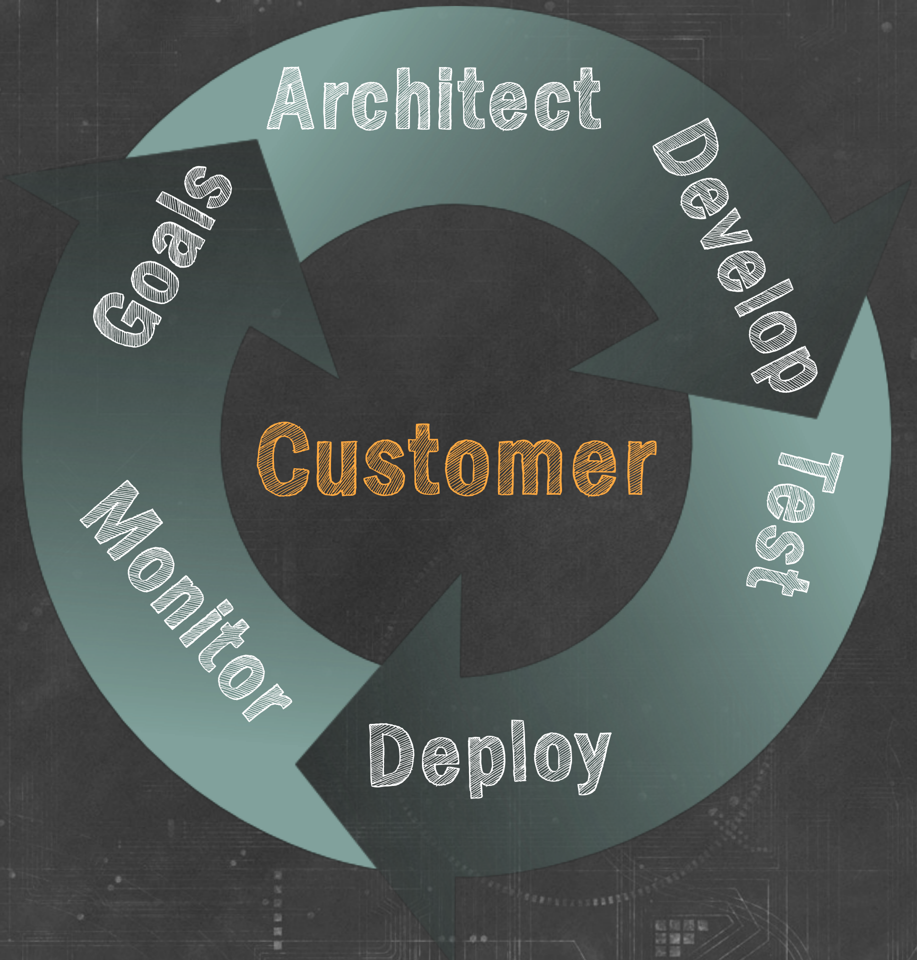
never compromise on durability

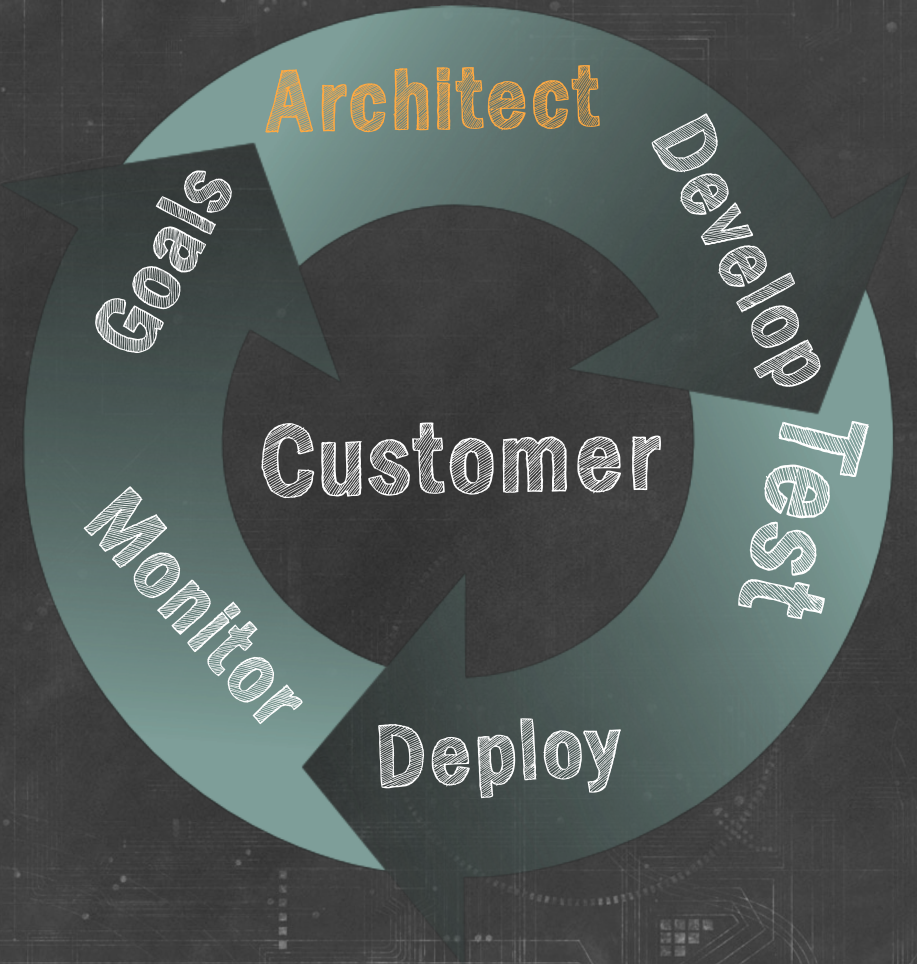
scale is our problem

easy to use

consistent and low latencies

scale in rps


Architect
Develop
Test
Deploy
Monitor
Goals
Customer


Architect
Develop
Test
Deploy
Monitor
Goals
Customer

Sacred Tenets in Services

don’t compromise durability for performance

plan for success - plan for scalability

plan for failures - fault -tolerance is key

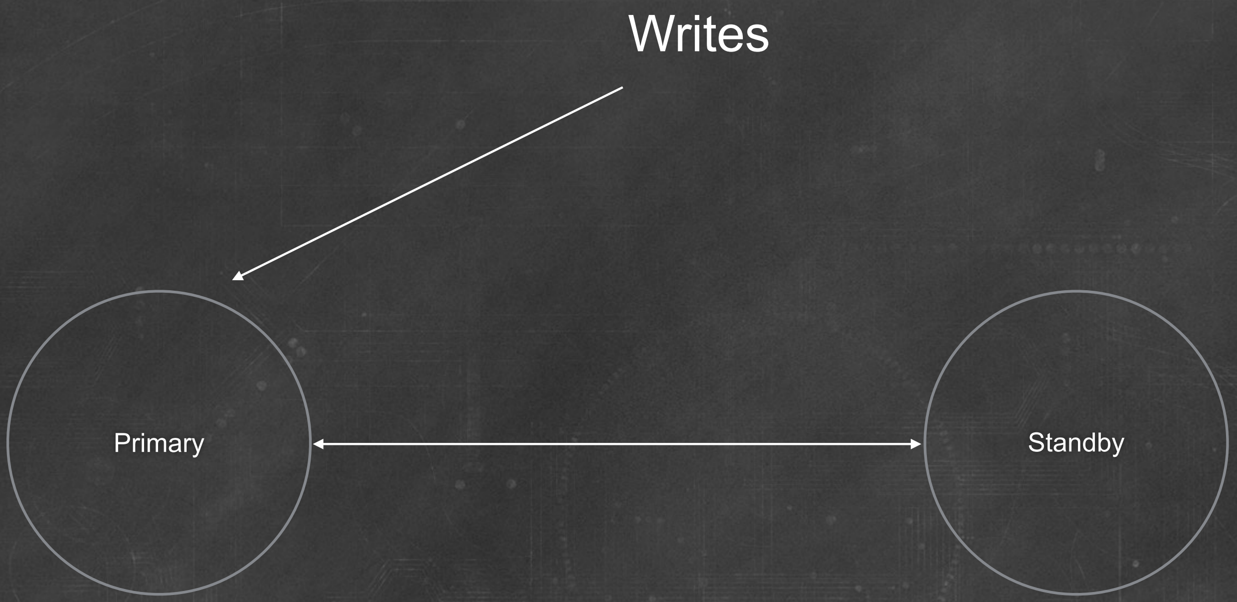
consistent performance is important

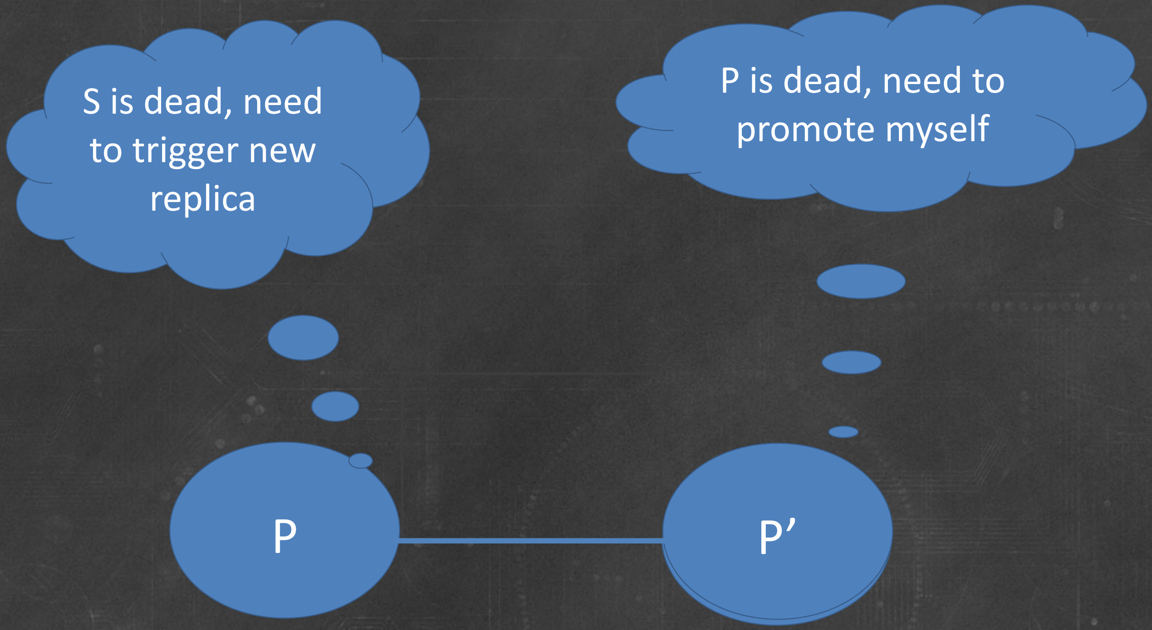
design - think of blast radius

insist on correctness

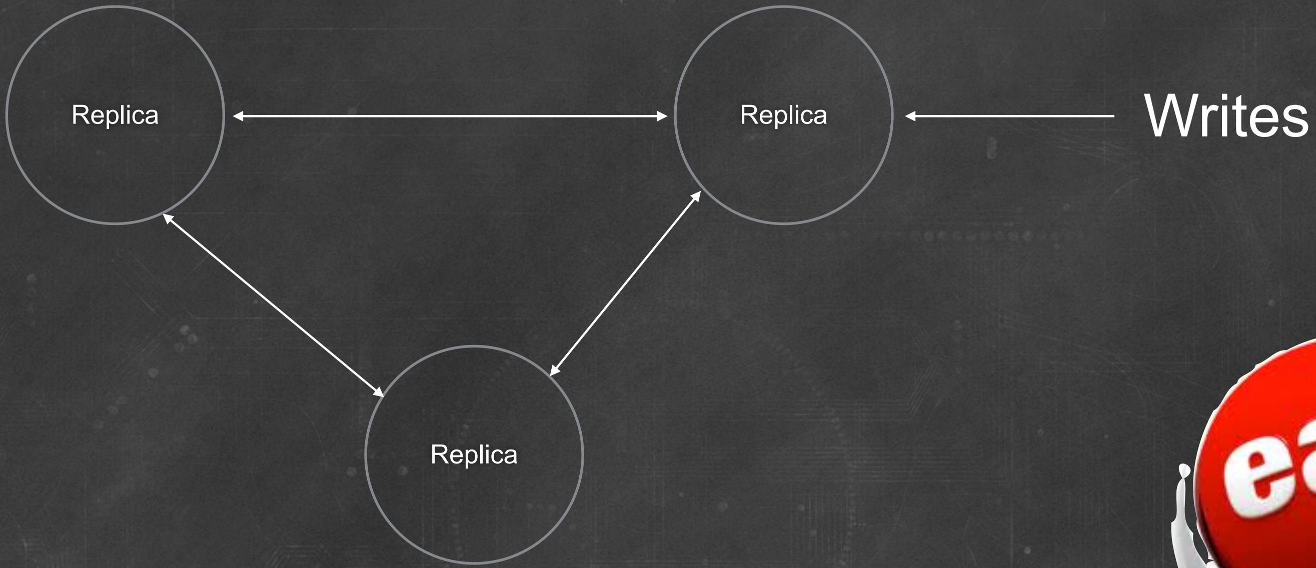
fault tolerance is a lesson best learned offline

a simple 2-way replication system of a traditional database...



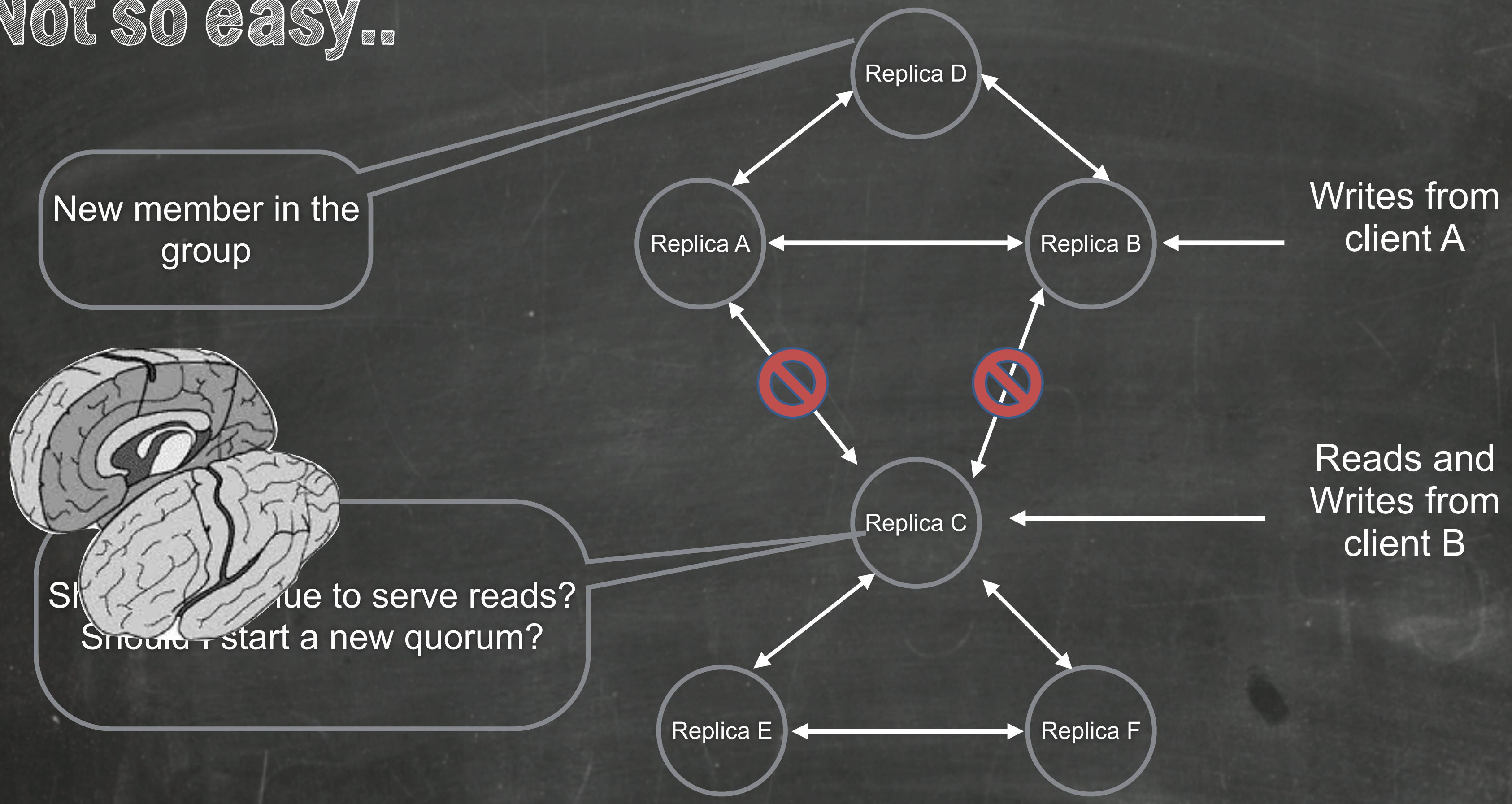

S is dead, need to trigger new replica
P is dead, need to promote myself
P
P'

improved Replication: quorum



Quorum: Successful write on a majority

Not so easy..



Classic Split Brain Issue in Replicated systems leading to lost writes!

Building correct distributed systems is not straight forward..

- *How do you handle replica failures?*
- *How do you ensure there is not a parallel quorum?*
- *How do you handle partial failures of replicas?*
- *How do you handle concurrent failures?*

correctness is hard, but necessary

Formal Methods

Formal Methods

to minimize bugs, we must have a precise description of the design

Formal Methods

code is too detailed

design documents and diagrams are vague & imprecise

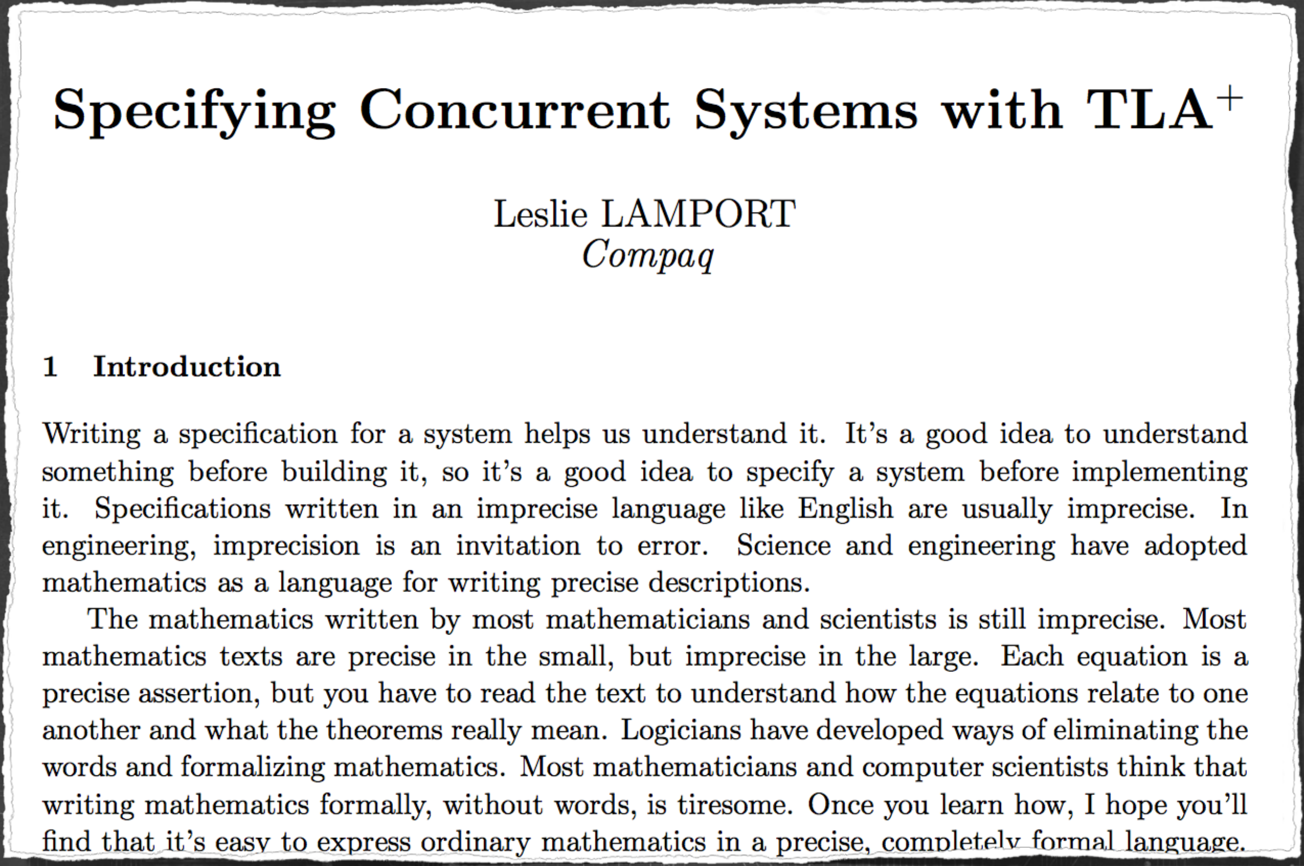
how would you express partial failures or concurrency?

Formal Methods

law of large numbers is your friend,
until you hit large numbers

so design for scale

TLA+ to the rescue?



Specifying Concurrent Systems with TLA$^+$

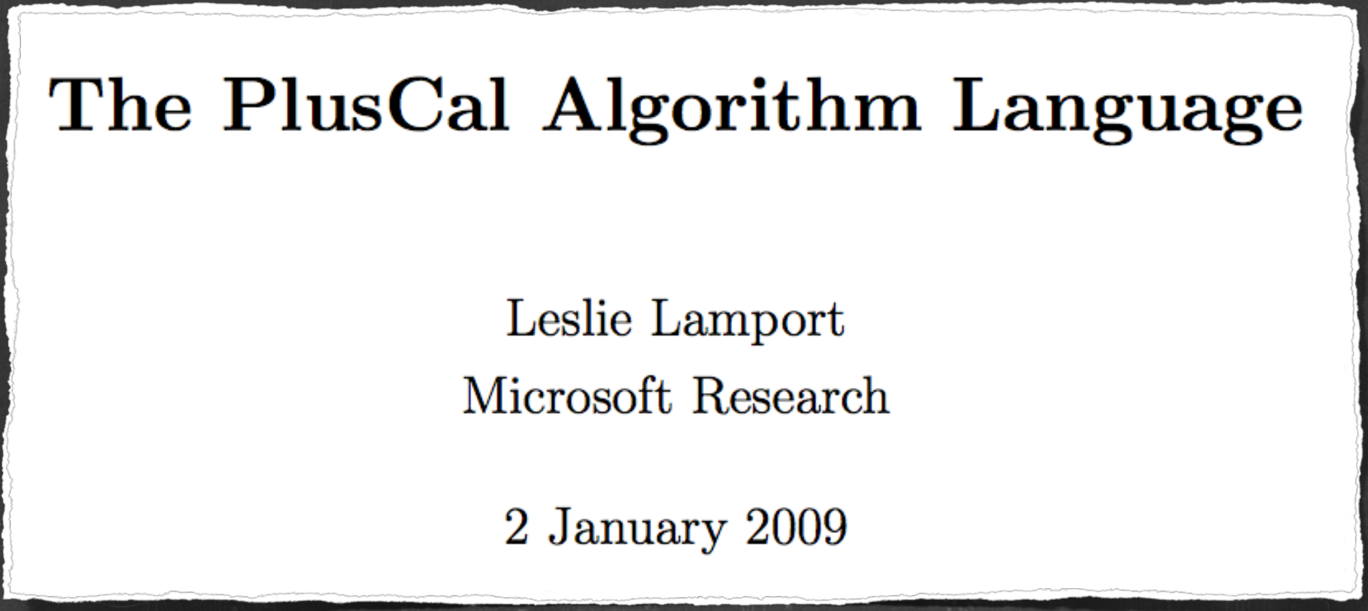
Leslie LAMPORT
Compaq

1 Introduction

Writing a specification for a system helps us understand it. It's a good idea to understand something before building it, so it's a good idea to specify a system before implementing it. Specifications written in an imprecise language like English are usually imprecise. In engineering, imprecision is an invitation to error. Science and engineering have adopted mathematics as a language for writing precise descriptions.

The mathematics written by most mathematicians and scientists is still imprecise. Most mathematics texts are precise in the small, but imprecise in the large. Each equation is a precise assertion, but you have to read the text to understand how the equations relate to one another and what the theorems really mean. Logicians have developed ways of eliminating the words and formalizing mathematics. Most mathematicians and computer scientists think that writing mathematics formally, without words, is tiresome. Once you learn how, I hope you'll find that it's easy to express ordinary mathematics in a precise, completely formal language.

PlusCal

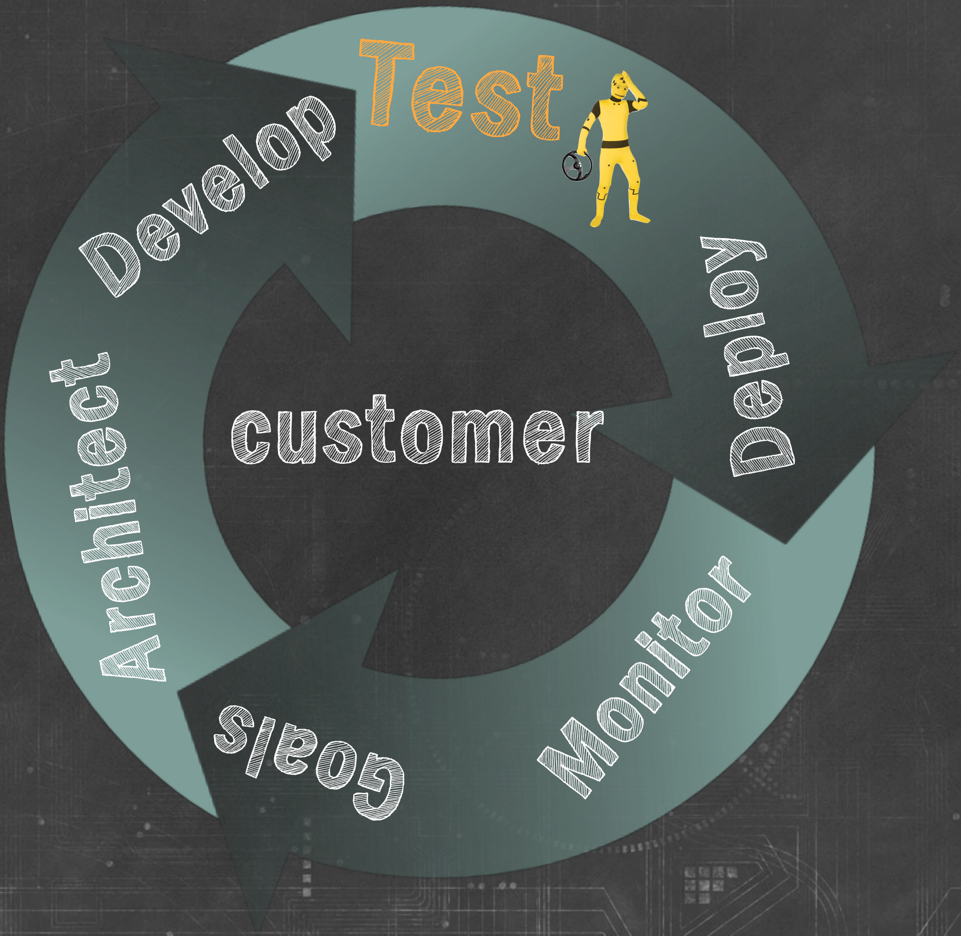


The PlusCal Algorithm Language

Leslie Lamport
Microsoft Research

2 January 2009

formal methods are **necessary**
but not **sufficient..**



@swami_79 @ksshams

don't forget to test - no, seriously..

simulate failures at
unit test level

fault injection testing

scale testing

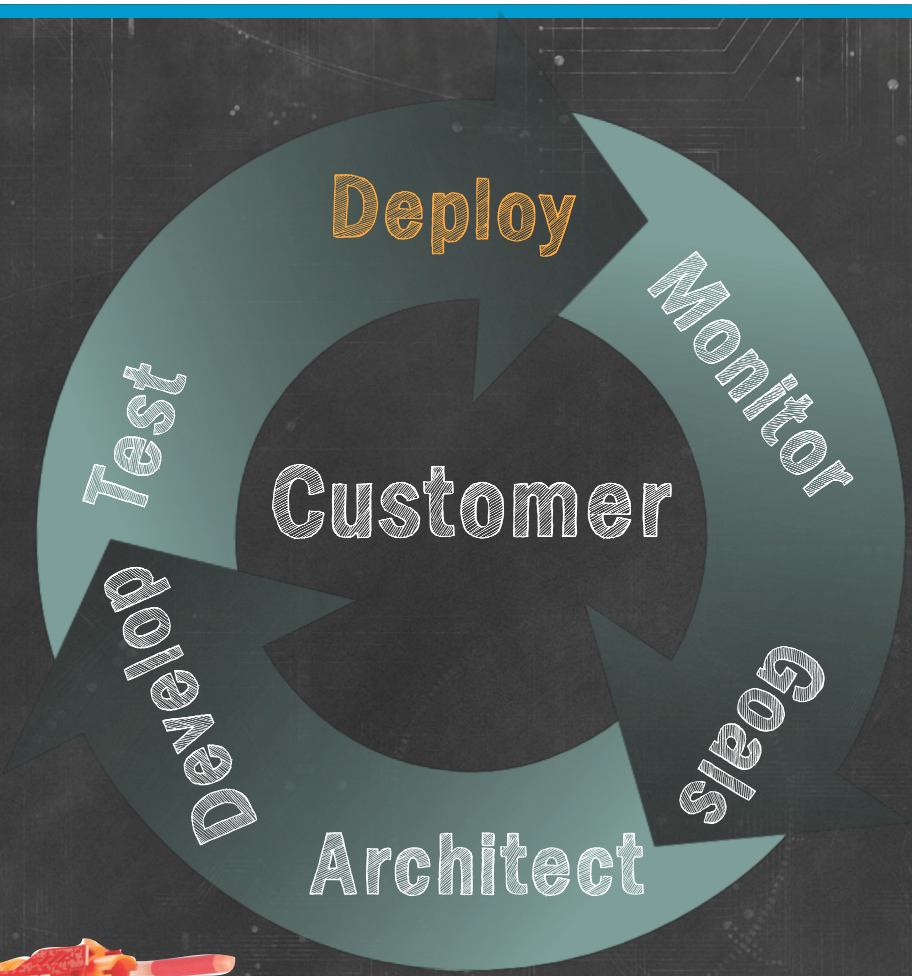
embrace failure and don’t be surprised

datacenter testing

network brown out testing

testing is a lifelong journey

testing is necessary but not sufficient..



@swami_79

@ksshams

gamma
simulate real world

one box
does it work?

release cycle

phased deployment
treading lightly

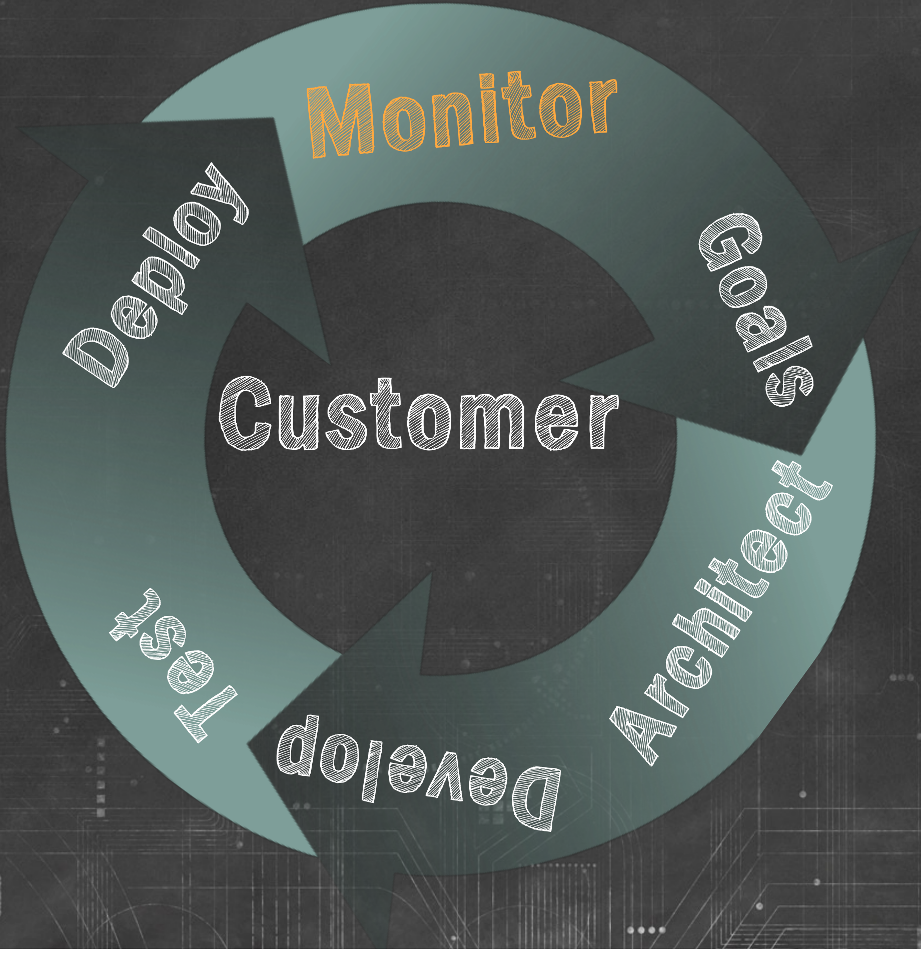
monitor
does it still work?

Canaries

@swami_79 @ksshams

Alarms

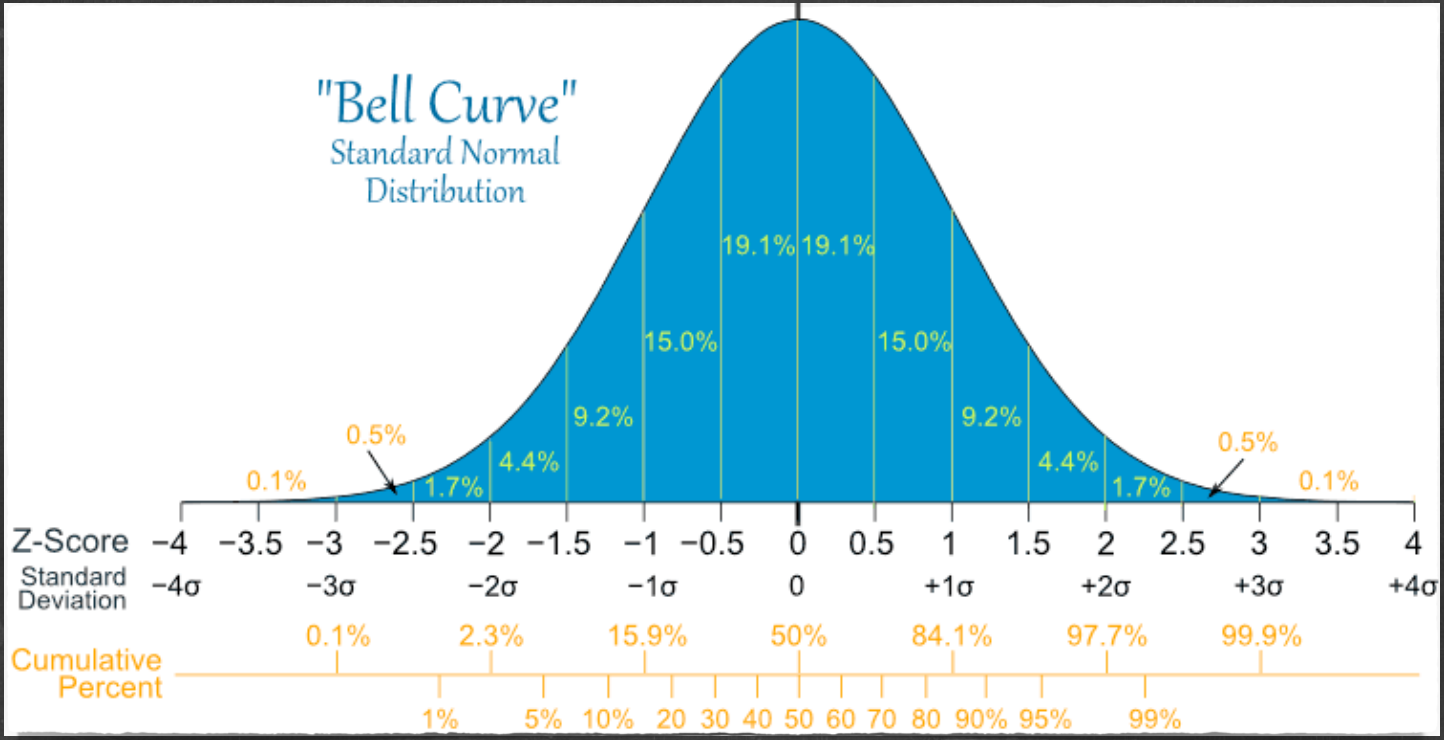
Monitor customer behavior



@swami_79

@ksshams

measuring customer experience is key



don't be satisfied by average - look at 99 percentile

@swami_79 @ksshams

understand the scaling dimensions

@swami_79

@ksshams

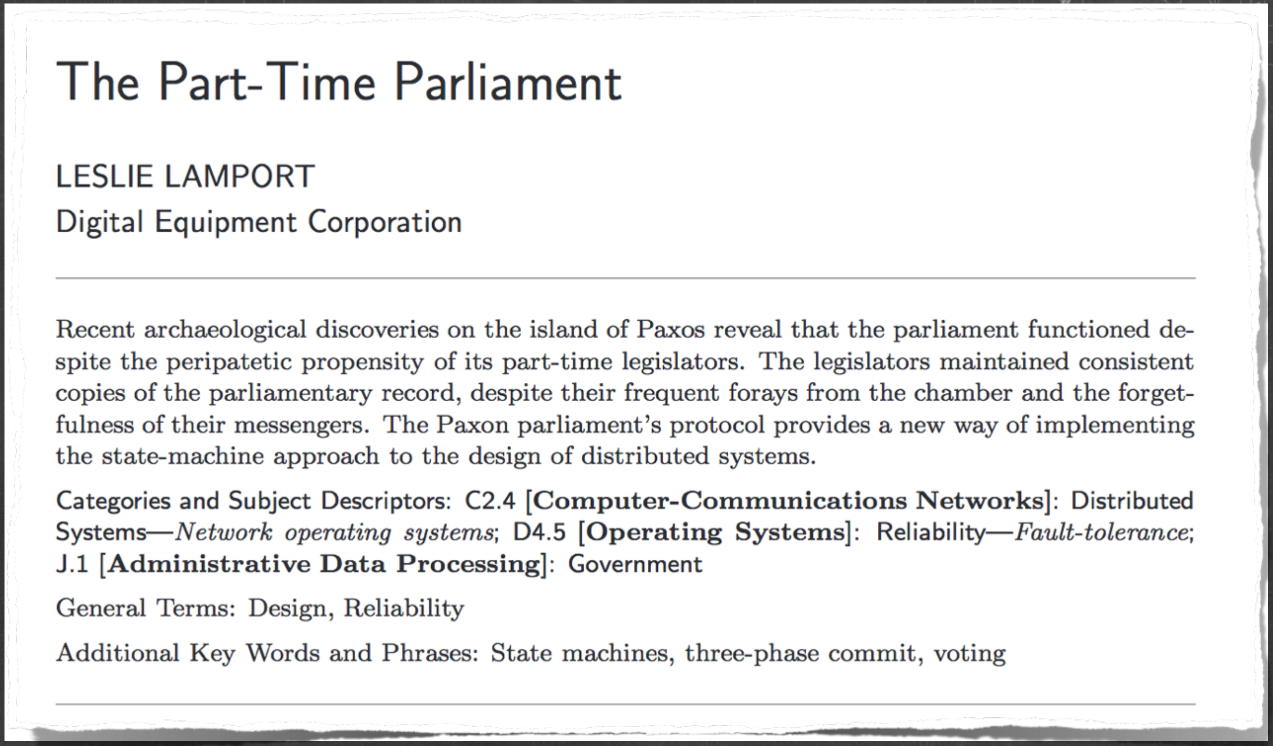
understand how your service will be abused

let's see these rules in action through a true story

@swami_79 @ksshams

we were building distributed systems all over amazon.com

we needed a uniform and correct way to do consensus..



The Part-Time Parliament

LESLIE LAMPORT
Digital Equipment Corporation

Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxon parliament's protocol provides a new way of implementing the state-machine approach to the design of distributed systems.

Categories and Subject Descriptors: C2.4 [**Computer-Communications Networks**]: Distributed Systems—*Network operating systems*; D4.5 [**Operating Systems**]: Reliability—*Fault-tolerance*; J.1 [**Administrative Data Processing**]: Government

General Terms: Design, Reliability

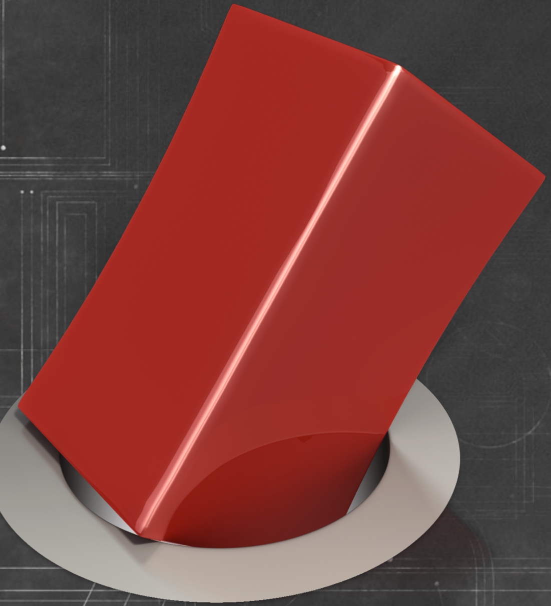
Additional Key Words and Phrases: State machines, three-phase commit, voting

so we built a paxos lock ~~library~~ service

@swami_79 @ksshams

such a service is so much more useful than just leader election..

it became a distributed state store



such a service is so much more useful than just leader election..

or a distributed state store

Wait, wait... you're telling me...
If I pee in the yard, I get a TREAT?

wait wait.. you're telling me if I poll,
I can detect node failure?

@ksshams

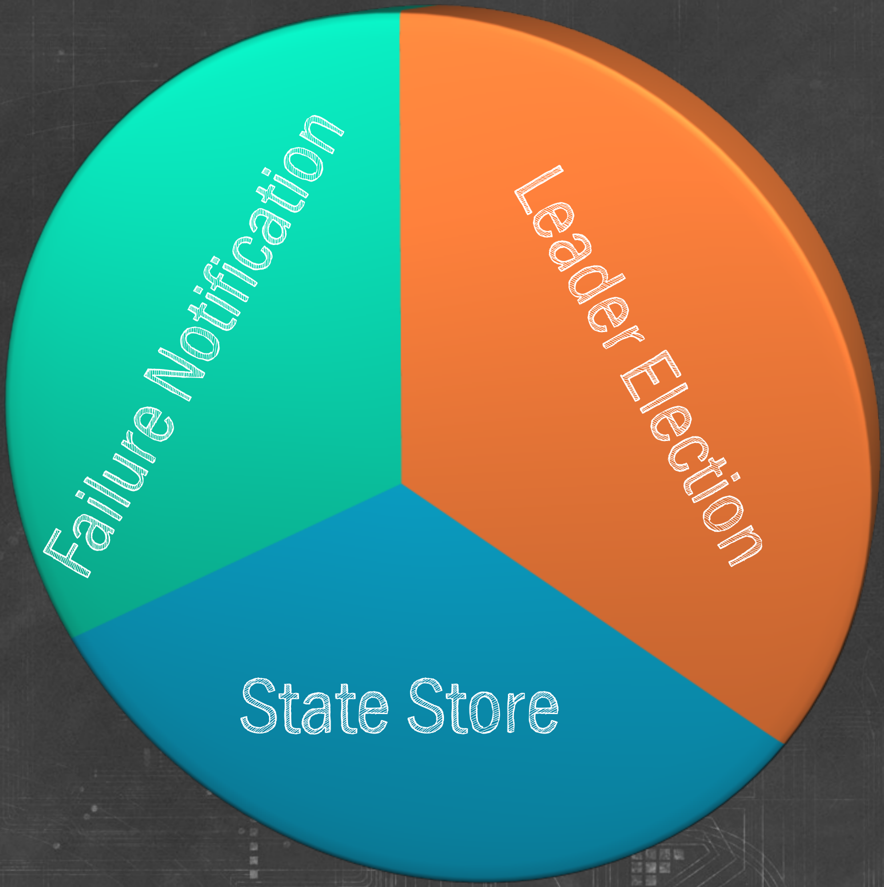
we acted quickly - and scaled up our entire fleet with more nodes

we slowed consensus...

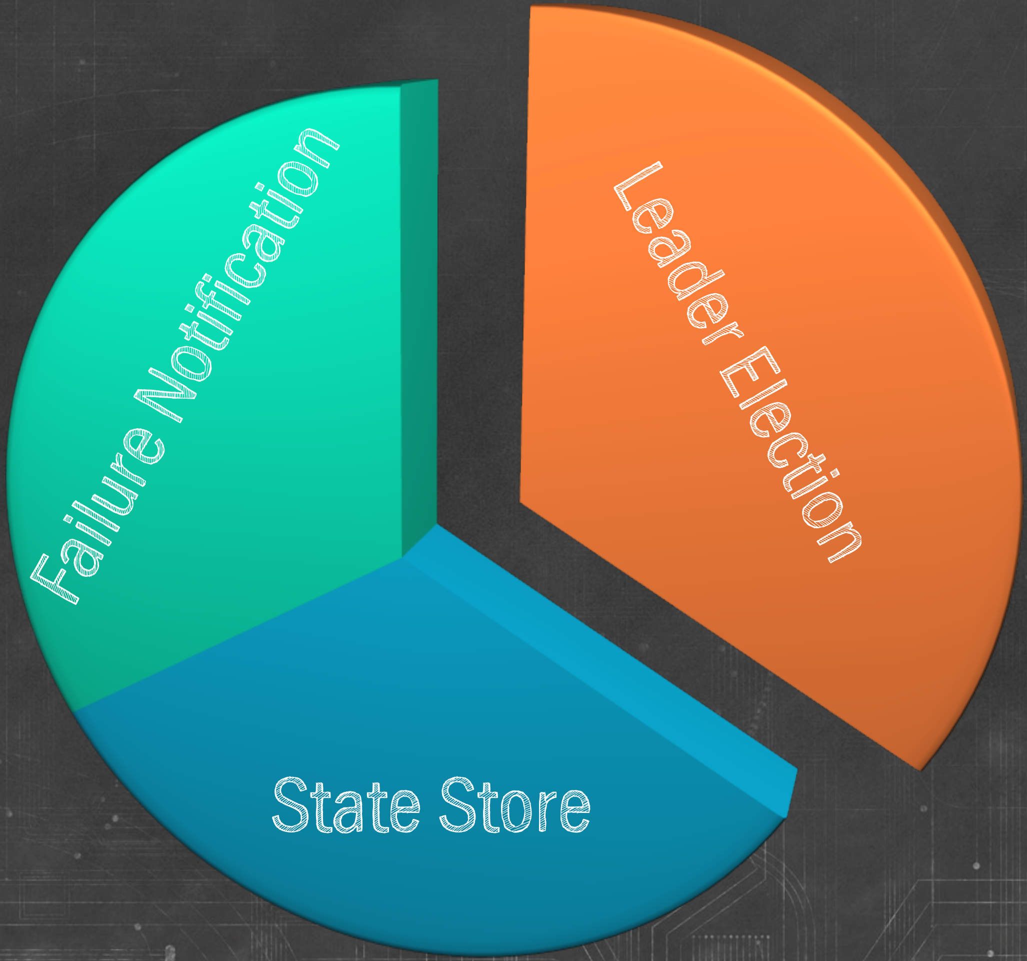
@ksshams

understand the scaling dimensions & scale them independently...

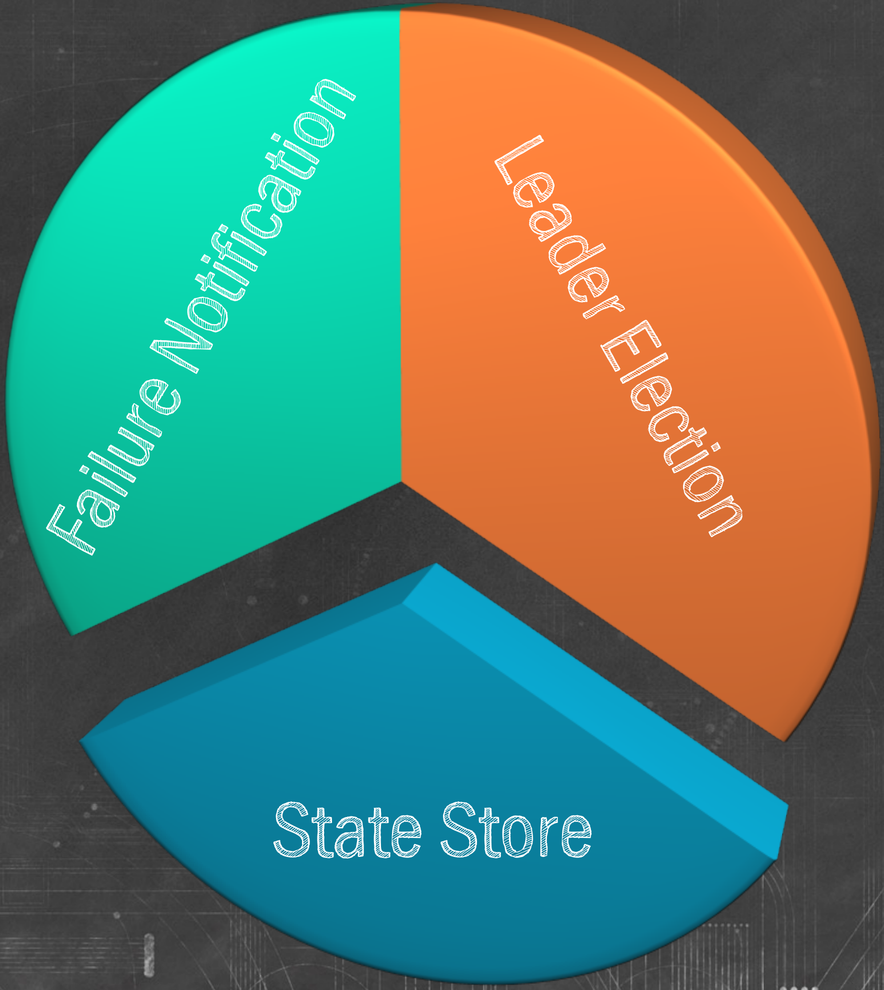
a lock service has 3 components..



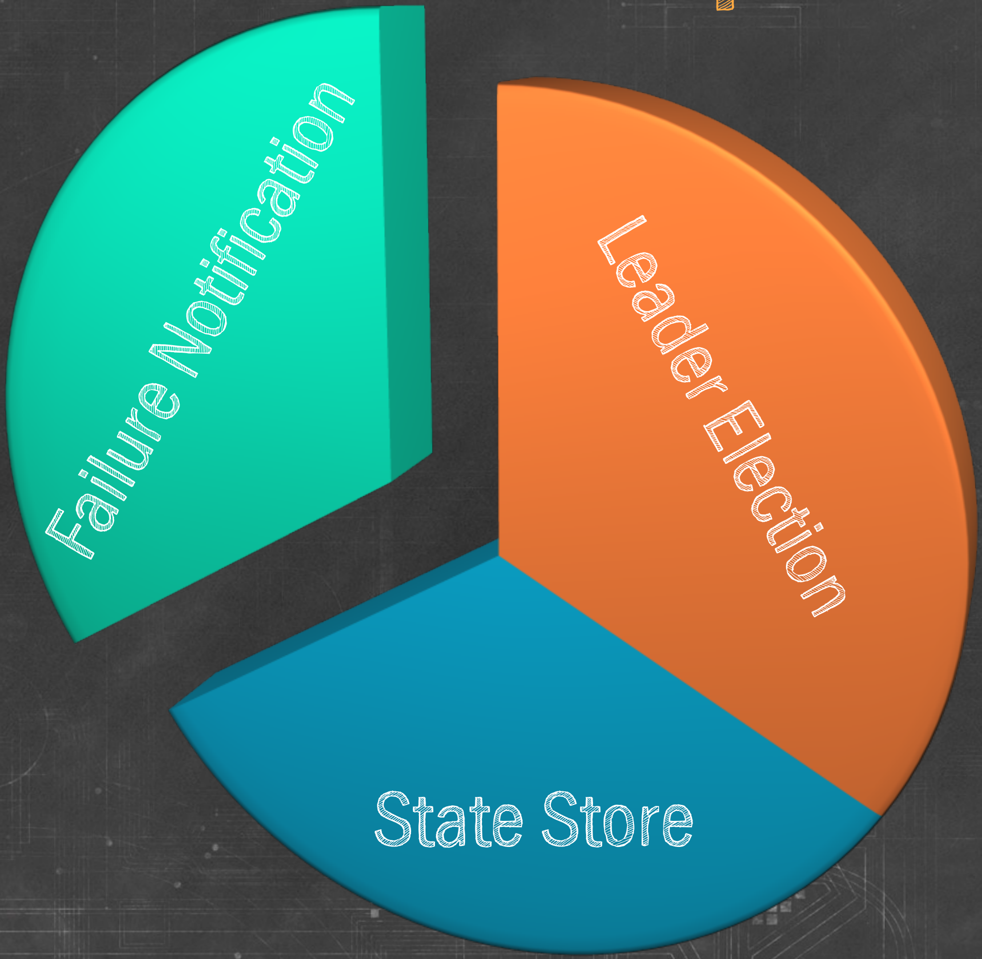
they must be scaled independently..



they must be scaled independently..



they must be scaled independently..



understand scaling dimensions

observe how service is used

relentlessly test

monitor like a hawk

strive for correctness

scalability **over** features

@swami_79

Thank You!

@kshams