Steve Riley   Sr. Technical Program Manager   steriley@amazon.com

How to “Think Cloud”

Architectural Design Patterns for Cloud Computing

Cloud Best Practices Whitepaper

Prescriptive guidance to Cloud Architects

http://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf

Amazon Web Services - Architecting for The Cloud: Best Practices January 2010

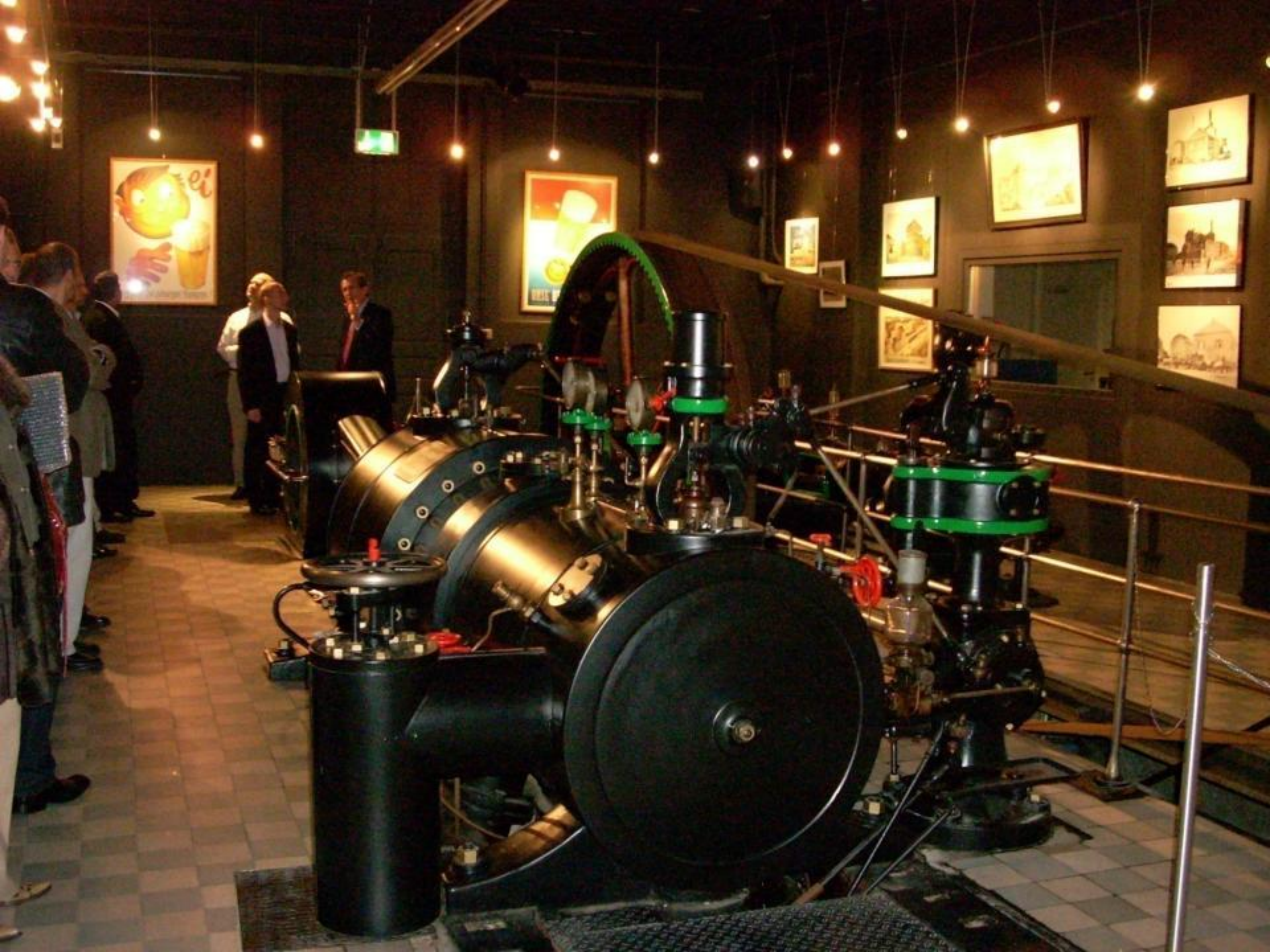
Architecting for the Cloud: Best Practices

January 2010

Jinesh Varia

jvaria@amazon.com

1

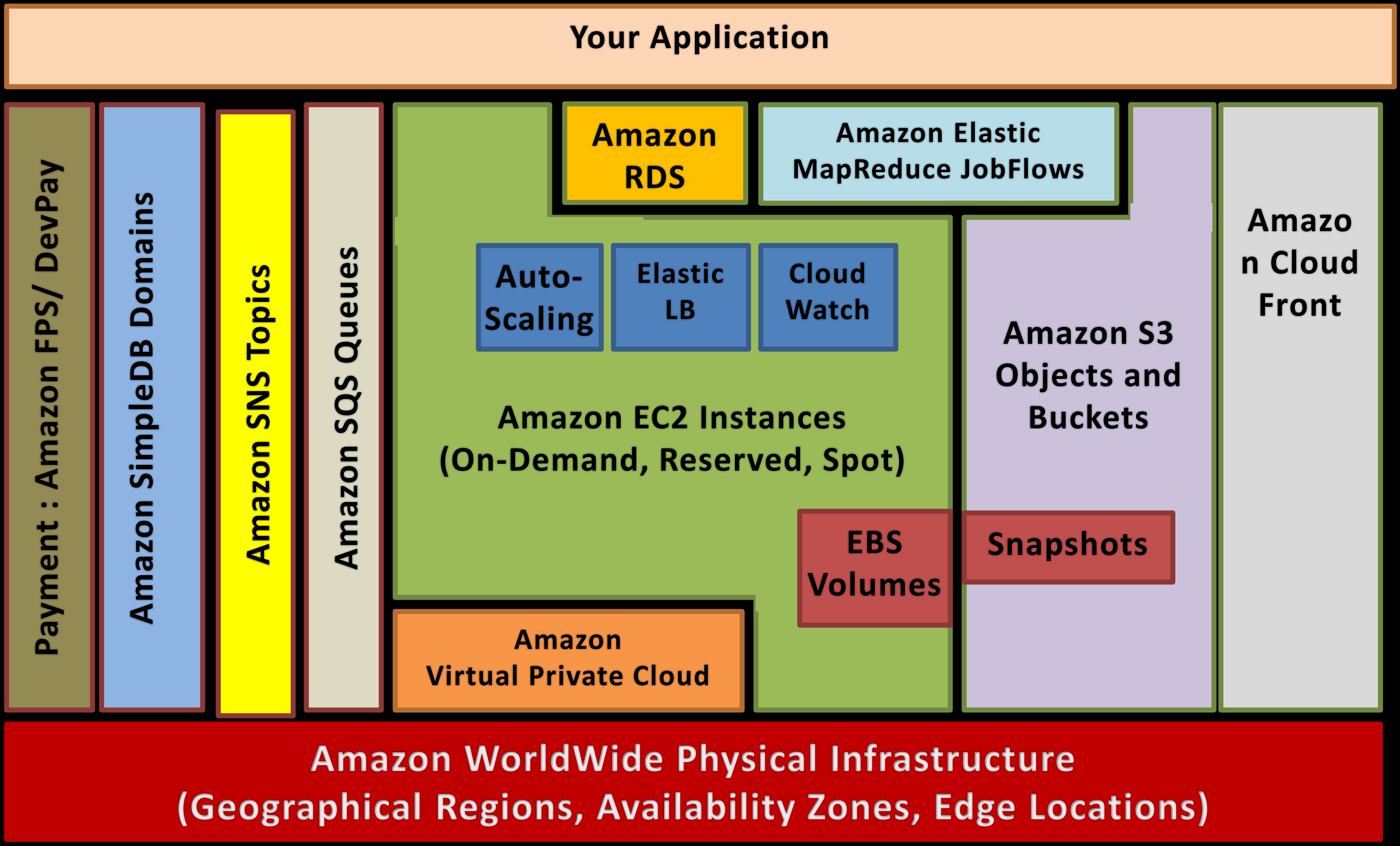


The “Living and Evolving” Cloud

AWS services and basic terminology

Most Applications Need:

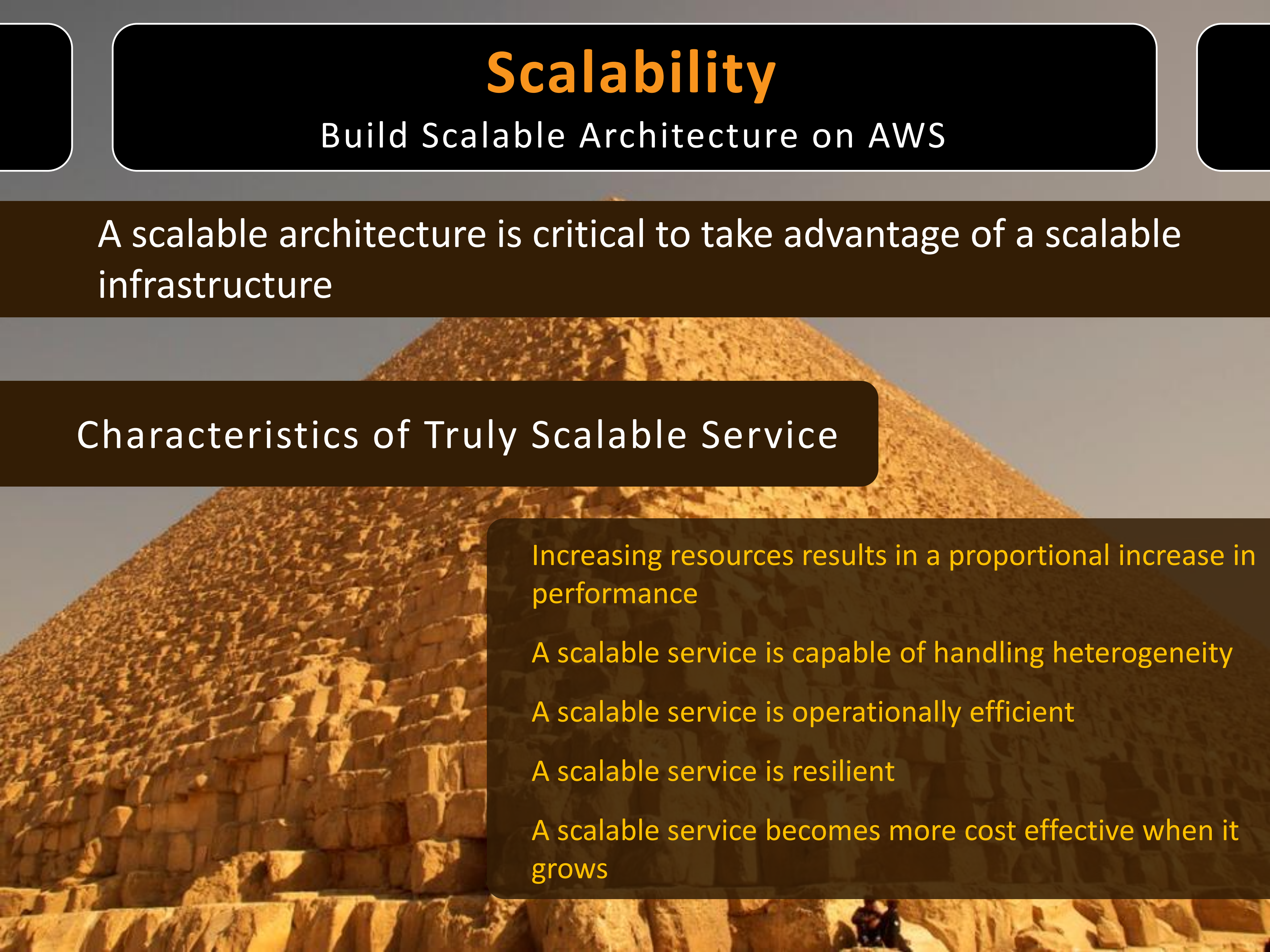
1. Compute
2. Storage
3. Messaging
4. Payment
5. Distribution
6. Scale
7. Analytics


Your Application
Payment : Amazon FPS/ DevPay
Amazon SimpleDB Domains
Amazon SNS Topics
Amazon SQS Queues
Amazon RDS
Amazon Elastic MapReduce JobFlows
Auto-Scaling
Elastic LB
Cloud Watch
Amazon EC2 Instances (On-Demand, Reserved, Spot)
EBS Volumes
Amazon Virtual Private Cloud
Amazon S3 Objects and Buckets
Snapshots
Amazon Cloud Front
Amazon WorldWide Physical Infrastructure
(Geographical Regions, Availability Zones, Edge Locations)

Cloud Computing Attributes

What makes the Cloud so attractive

Abstract Resources	Focus on your needs, not on hardware specs. As your needs change, so should your resources.
On-Demand Provisioning	Ask for what you need, exactly when you need it. Get rid of it when you don't need
Scalability in minutes	Scale out or in depending on usage needs.
Pay per consumption	No long-term commitments. Pay only for what you use.
Efficiency of Experts	Utilize the skills, knowledge and resources of experts.



Scalability

Build Scalable Architecture on AWS

A scalable architecture is critical to take advantage of a scalable infrastructure

Characteristics of Truly Scalable Service

Increasing resources results in a proportional increase in performance

A scalable service is capable of handling heterogeneity

A scalable service is operationally efficient

A scalable service is resilient

A scalable service becomes more cost effective when it grows

Cloud Architecture Lessons

using Amazon Web Services

1. Design for failure and nothing fails
2. Loose coupling sets you free
3. Implement “Elasticity”
4. Build Security in every layer
5. Don't fear constraints
6. Think Parallel
7. Leverage different storage options

1. Design for Failure

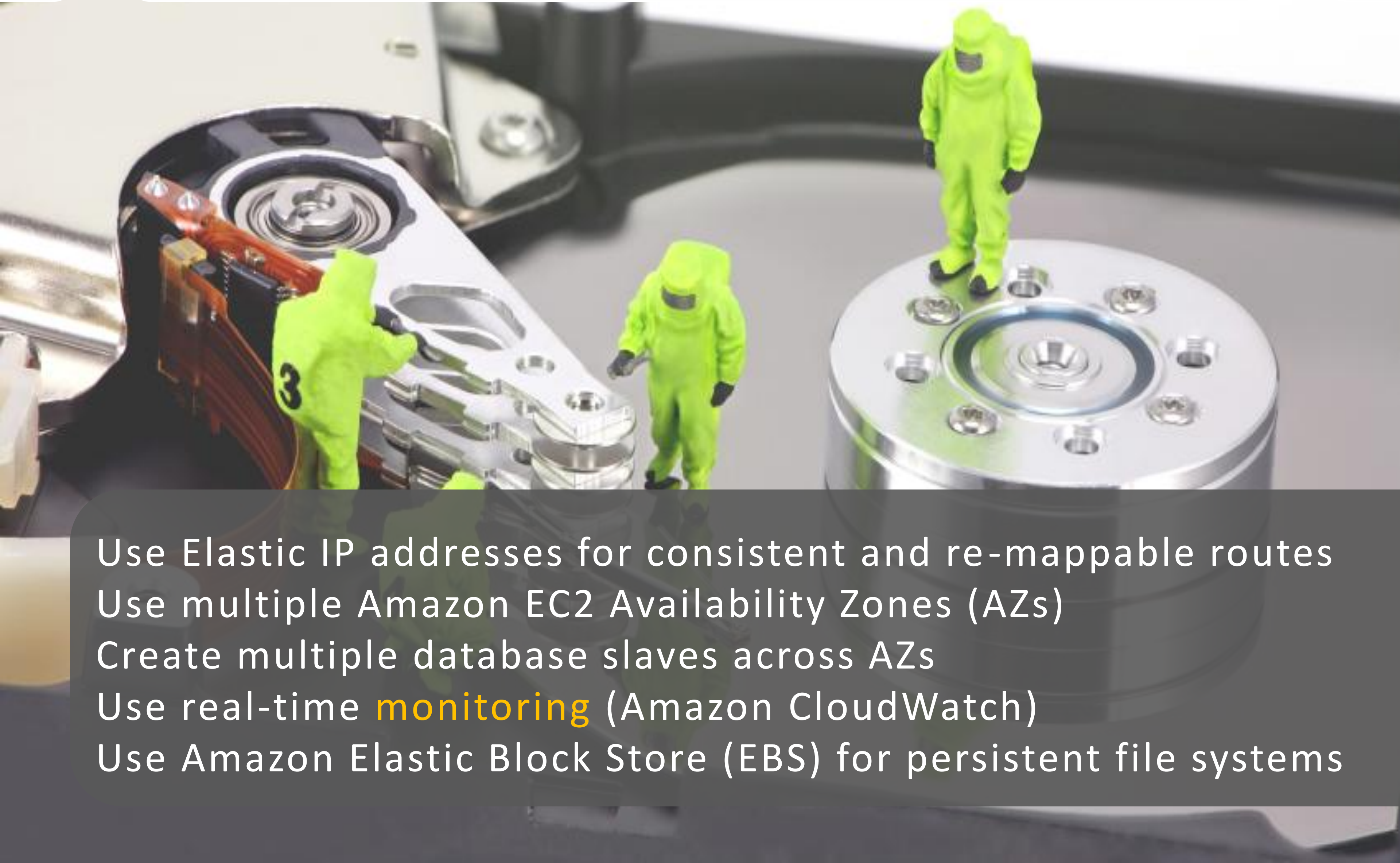
and nothing will really fail

"Everything fails, all the time"
Werner Vogels, CTO Amazon.com

Avoid single points of failure
Assume everything fails, and design backwards
Goal: Applications should continue to function even if the underlying physical hardware fails or is removed or replaced.

Design for Failure with AWS

Tools to make your life easier



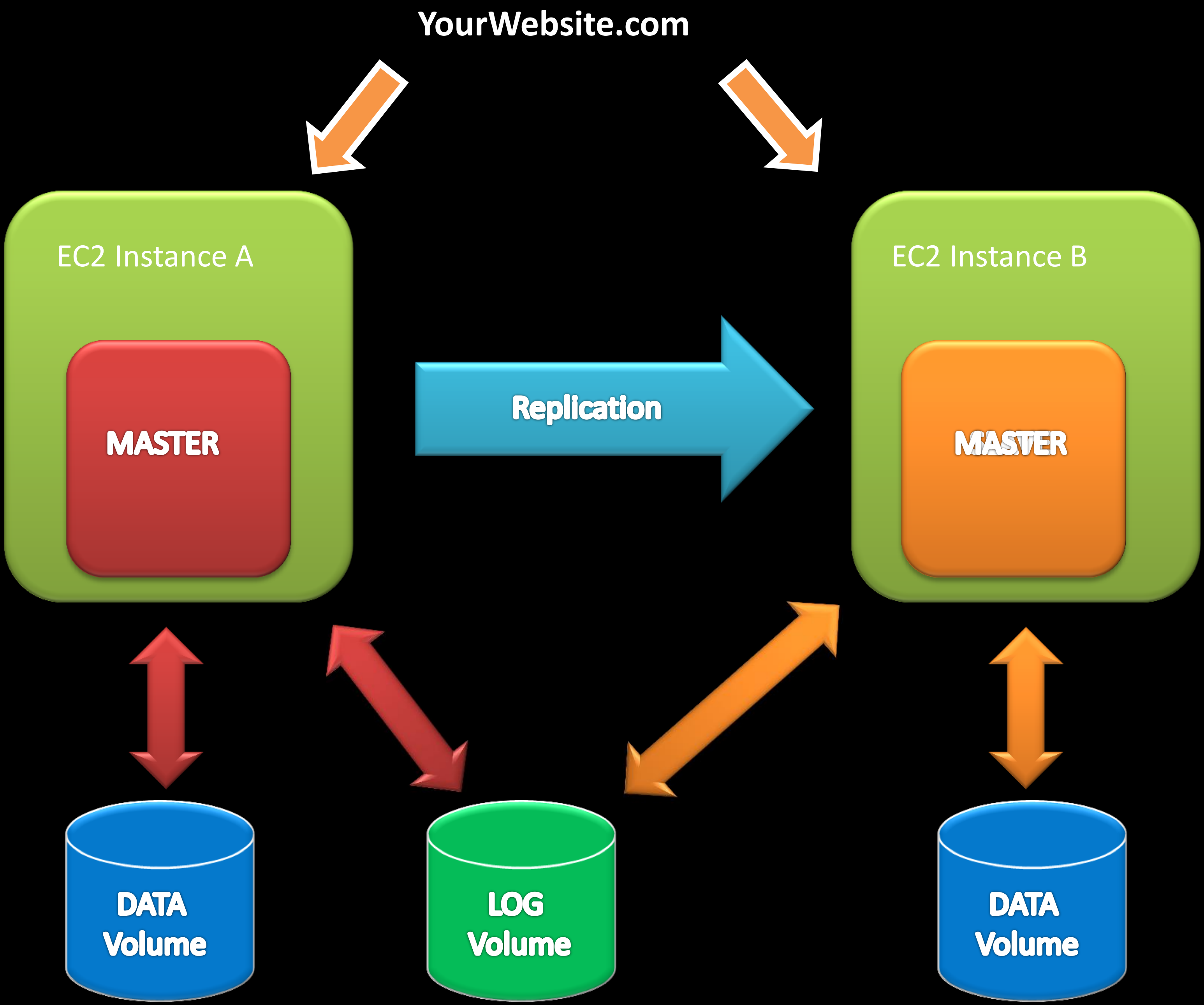
Use Elastic IP addresses for consistent and re-mappable routes

Use multiple Amazon EC2 Availability Zones (AZs)

Create multiple database slaves across AZs

Use real-time monitoring (Amazon CloudWatch)

Use Amazon Elastic Block Store (EBS) for persistent file systems



YourWebsite.com

EC2 Instance A

MASTER

Replication

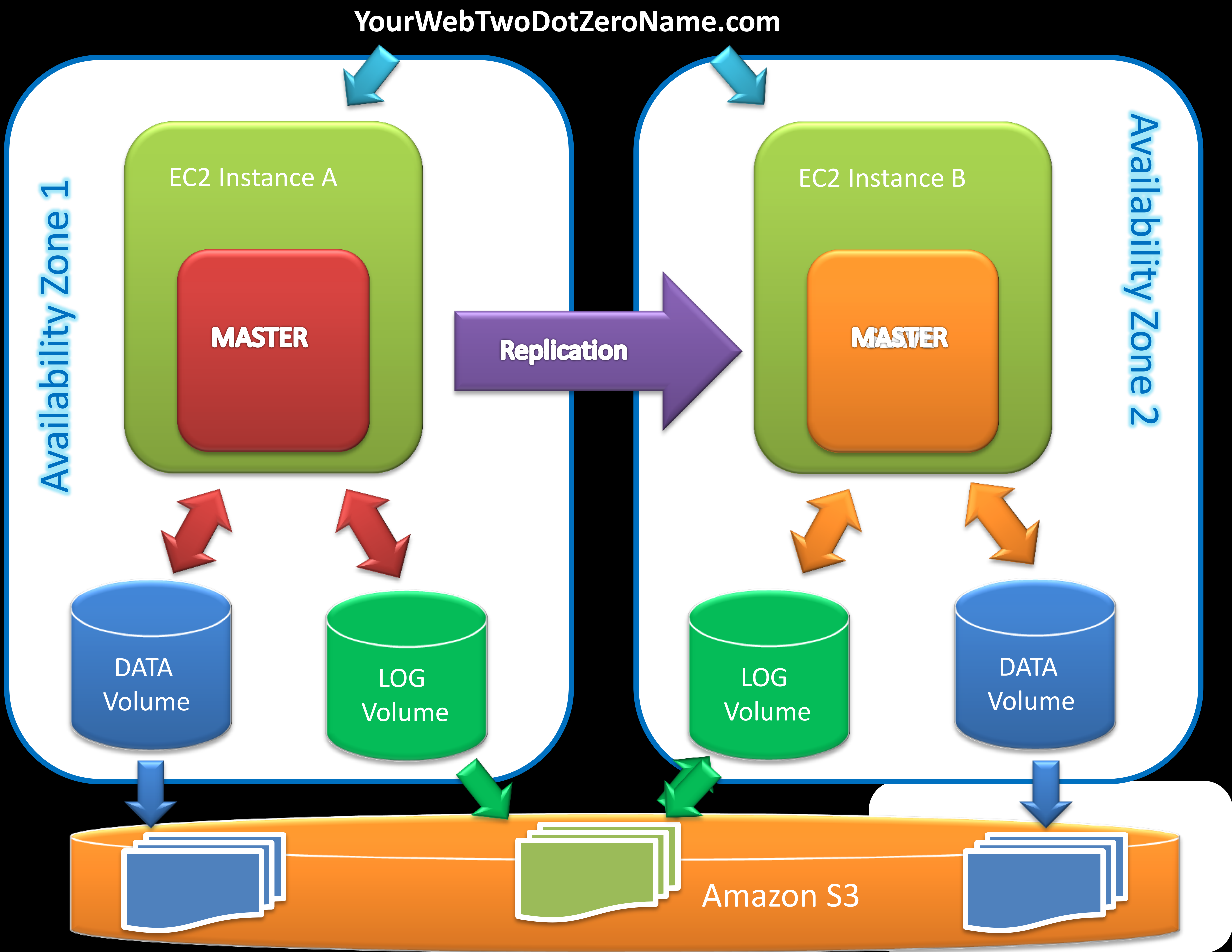
EC2 Instance B

MASTER

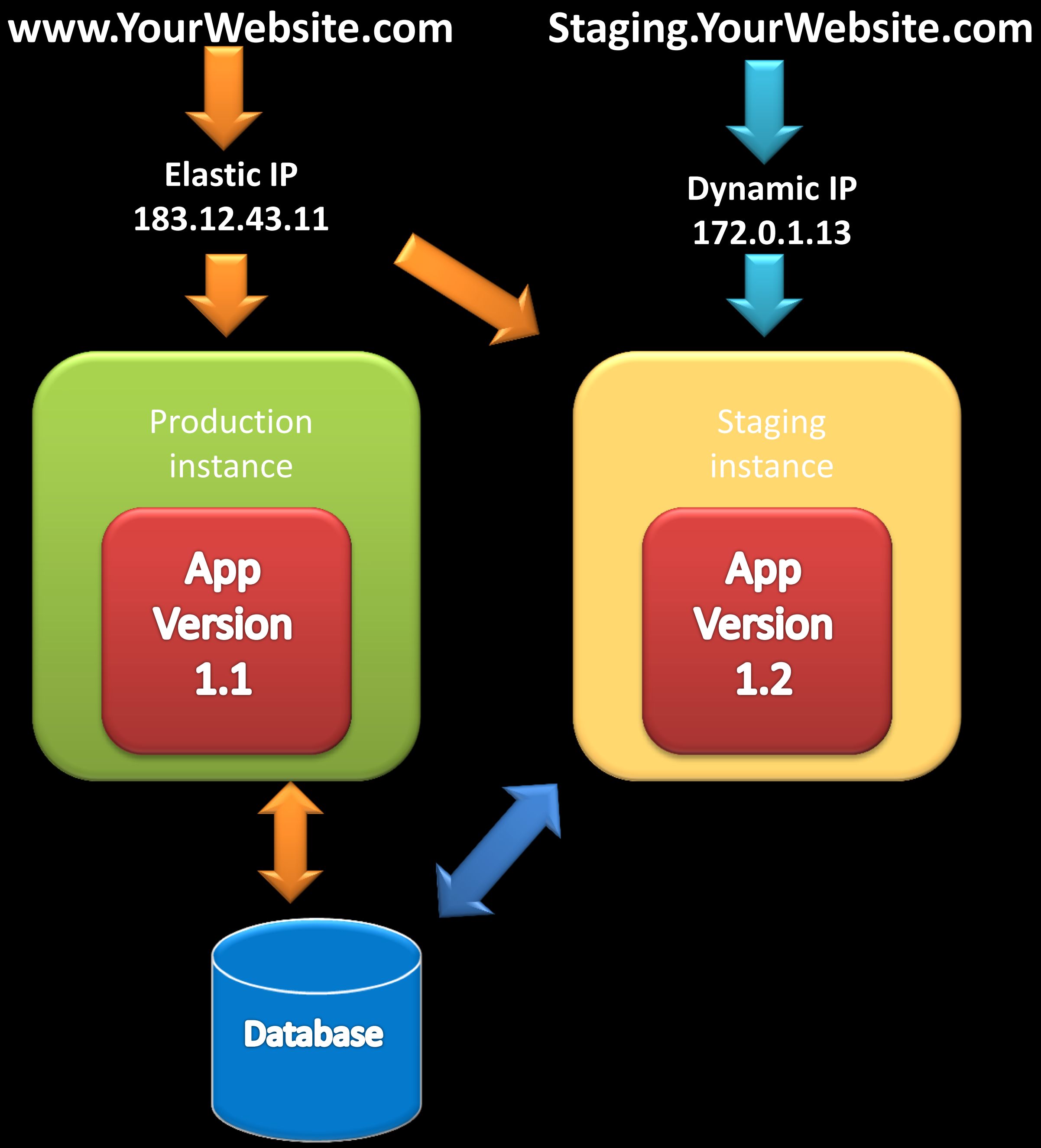
DATA Volume

LOG Volume

DATA Volume



YourWebTwoDotZeroName.com
Availability Zone 1
EC2 Instance A
MASTER
Replication
EC2 Instance B
MASTER
Availability Zone 2
DATA Volume
LOG Volume
LOG Volume
DATA Volume
Amazon S3



www.YourWebsite.com

Elastic IP
183.12.43.11

Production instance

App Version 1.1

Staging.YourWebsite.com

Dynamic IP
172.0.1.13

Staging instance

App Version 1.2

Database

2. Build Loosely Coupled Systems

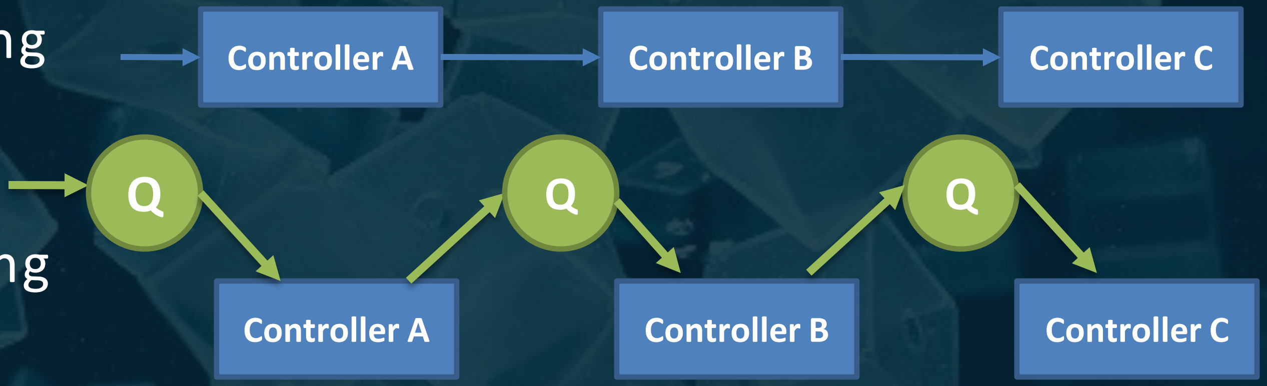
The looser they're coupled, the bigger they scale

Independent components
Design everything as a Black Box
De-coupling for Hybrid models
Load-balance clusters

Use Amazon SQS as Buffers

Tight Coupling

Loose Coupling using Queues



3. Implement Elasticity

Elasticity is fundamental property of the Cloud

Don’t assume health or fixed location of components

Use designs that are resilient to reboot and re-launch

Bootstrap your instances: Instances on boot will ask a question *“Who am I & what is my role?”*

Enable dynamic configuration

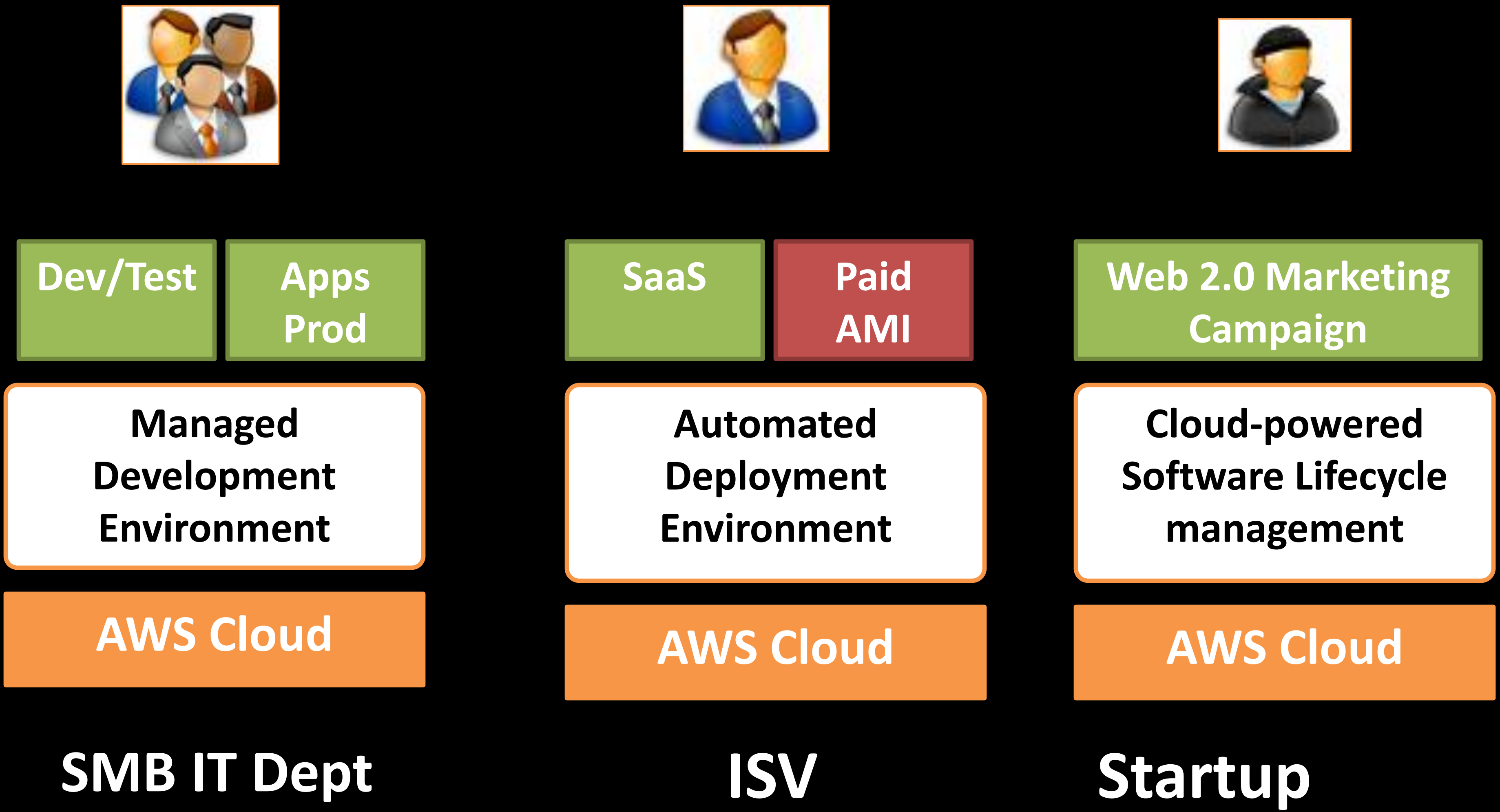
Use Auto-scaling (Free)

Use Elastic Load Balancing on multiple layers

Use configurations in SimpleDB to bootstrap instance

3. Implement Elasticity

Automate everything



Dev/Test

Apps Prod

Managed Development Environment

AWS Cloud

SMB IT Dept

SaaS

Paid AMI

Automated Deployment Environment

AWS Cloud

ISV

Web 2.0 Marketing Campaign

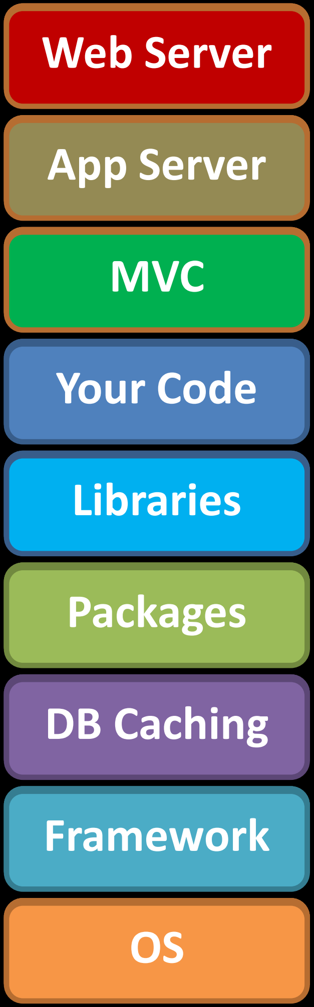
Cloud-powered Software Lifecycle management

AWS Cloud

Startup

3. Implement Elasticity

Standardized Application Stacks



Web Server

App Server

MVC

Your Code

Libraries

Packages

DB Caching

Framework

OS

Java Stack

.NET Stack

RoR stack

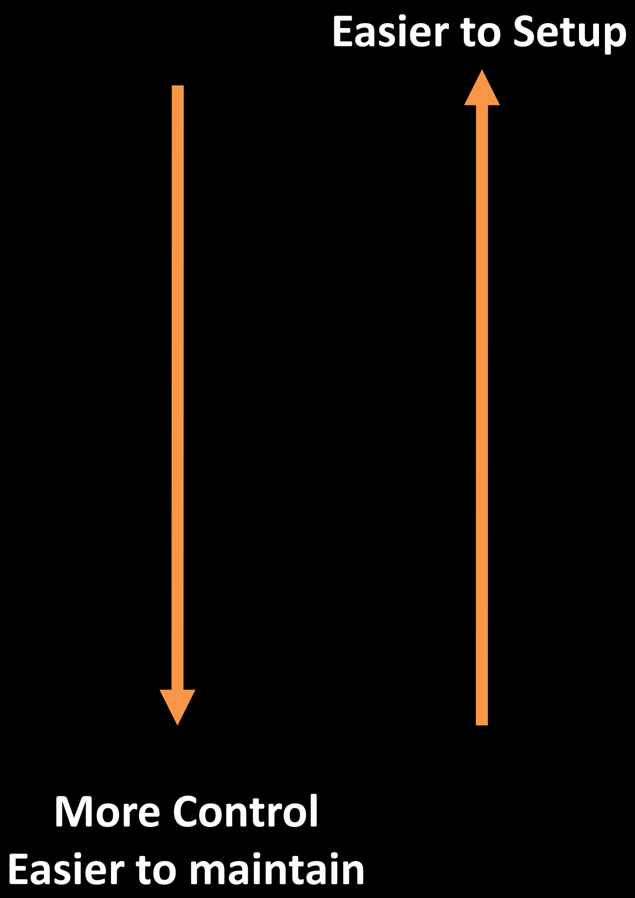
3. Implement Elasticity

3 approaches to designing your AMIs

1 Inventory of fully baked AMIs
(Frozen Pizza Model)

2 “Golden AMIs” with fetch on boot
(Take N’ Bake Papa Murphy Model)

3 AMIs with JeOS and “Chef” Agent
(Made to Order Pizza Model)

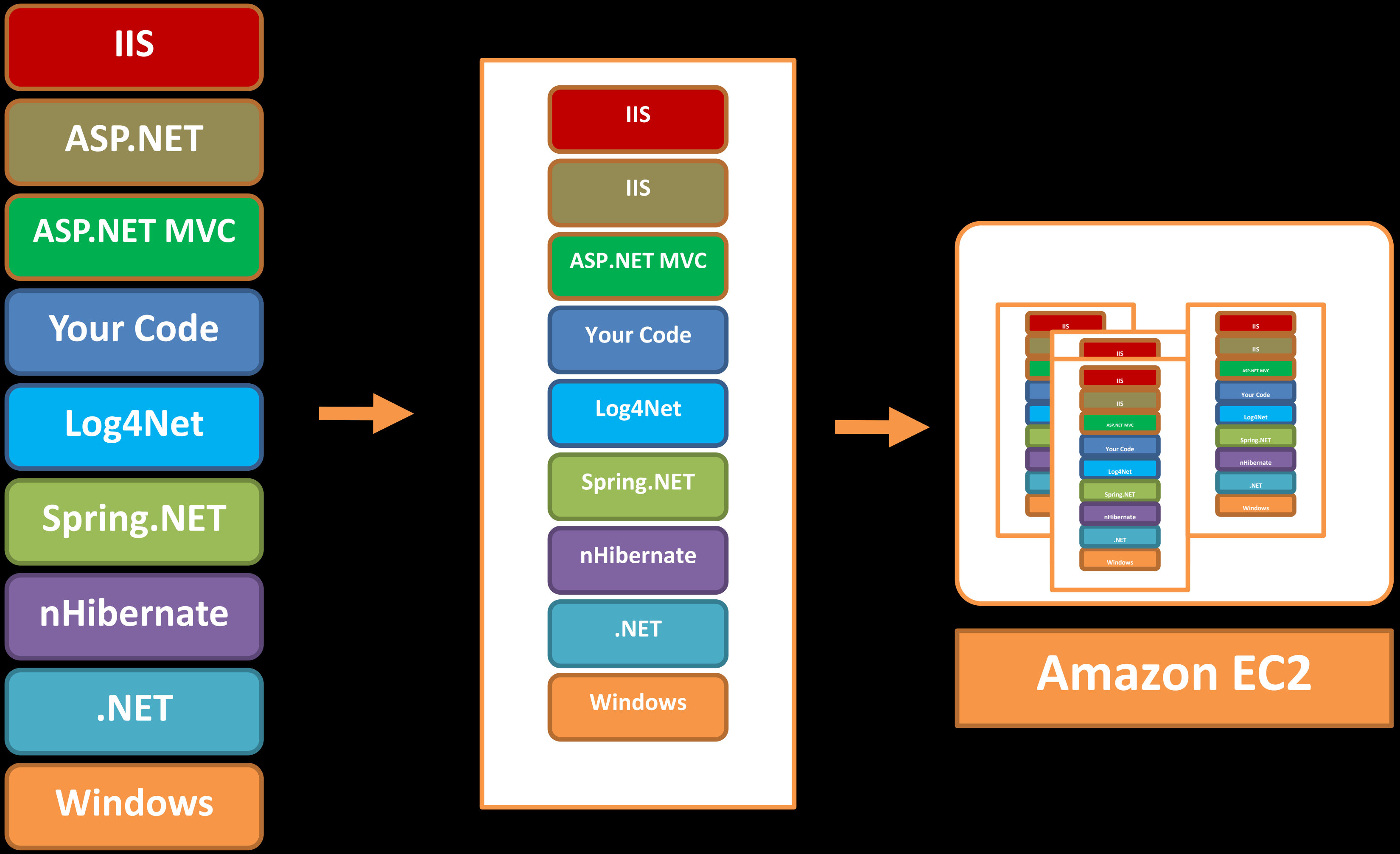


Easier to Setup

More Control
Easier to maintain

3. Implement Elasticity

1. Frozen Pizza Model



IIS

ASP.NET

ASP.NET MVC

Your Code

Log4Net

Spring.NET

nHibernate

.NET

Windows

IIS

IIS

ASP.NET MVC

Your Code

Log4Net

Spring.NET

nHibernate

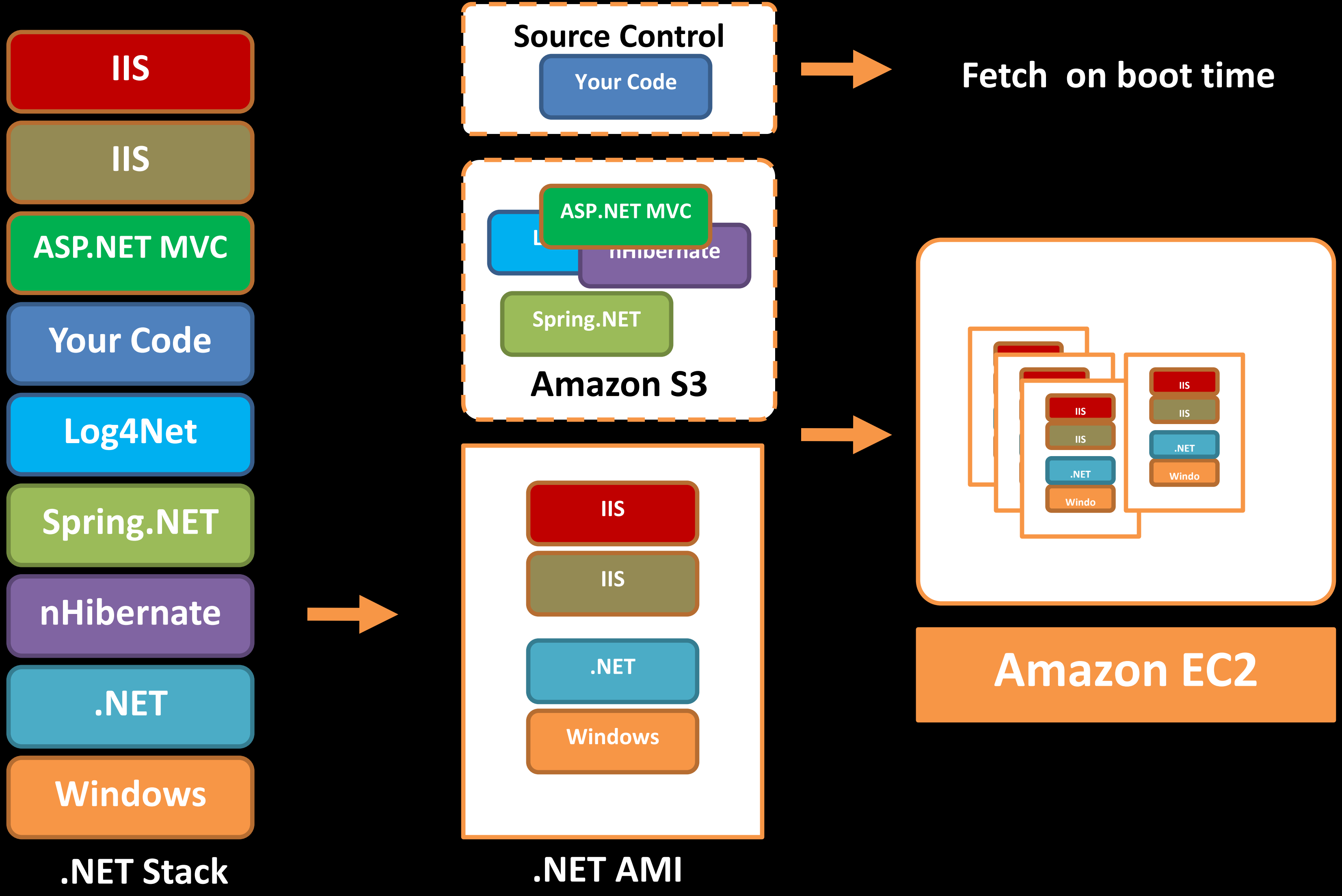
.NET

Windows

Amazon EC2

3. Implement Elasticity

2. Papa Murphy Pizza Model



IIS

IIS

ASP.NET MVC

Your Code

Log4Net

Spring.NET

nHibernate

.NET

Windows

.NET Stack

Source Control

Your Code

Fetch on boot time

ASP.NET MVC

nHibernate

Spring.NET

Amazon S3

IIS

IIS

.NET

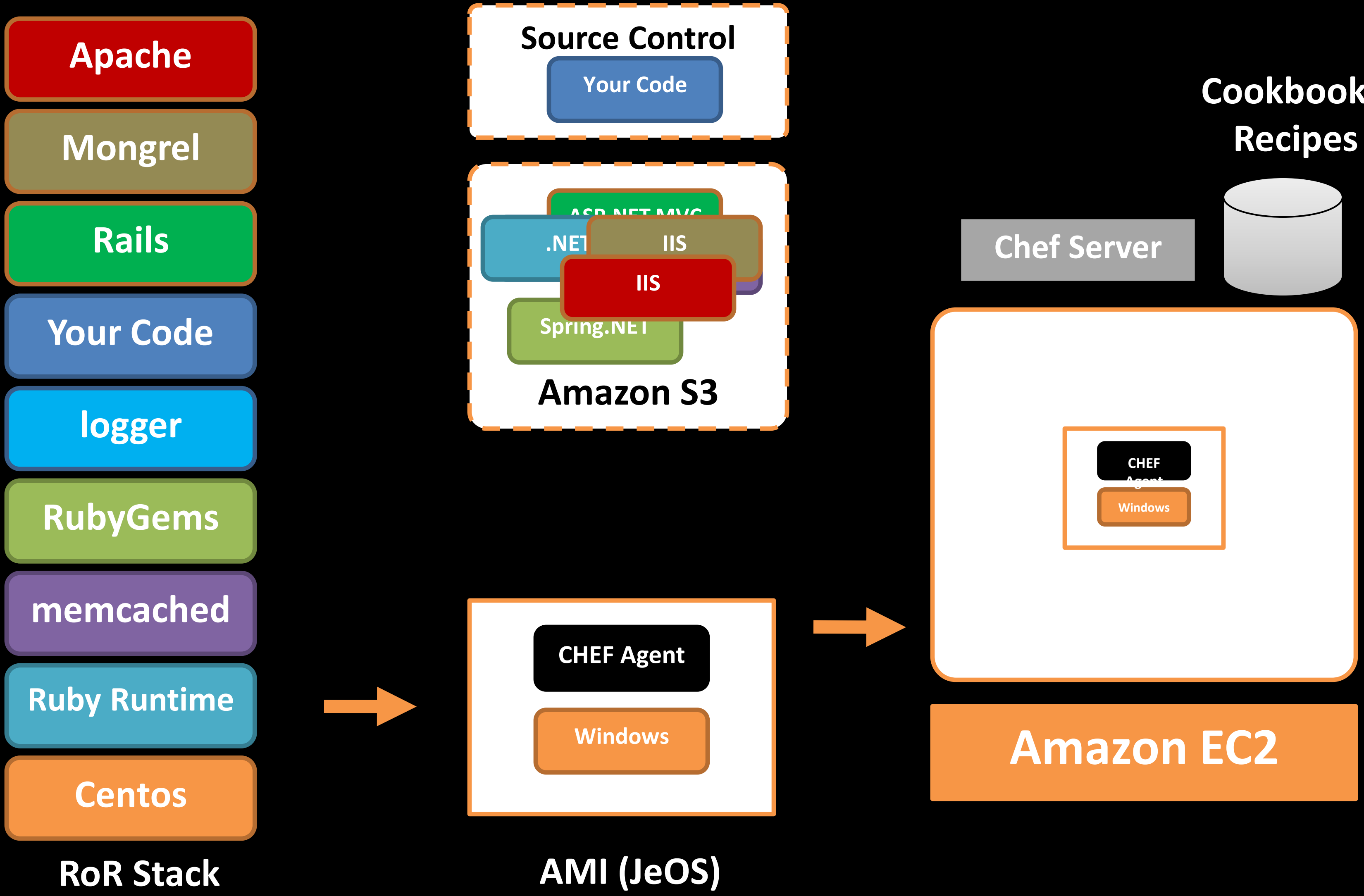
Windows

.NET AMI

Amazon EC2

3. Implement Elasticity

3. Made to Order Pizza Model



Apache

Mongrel

Rails

Your Code

logger

RubyGems

memcached

Ruby Runtime

Centos

RoR Stack

Source Control

Your Code

ASP.NET MVC

.NET

IIS

IIS

Spring.NET

Amazon S3

CHEF Agent

Windows

AMI (JeOS)

Cookbooks

Recipes

Chef Server

CHEF Agent

Windows

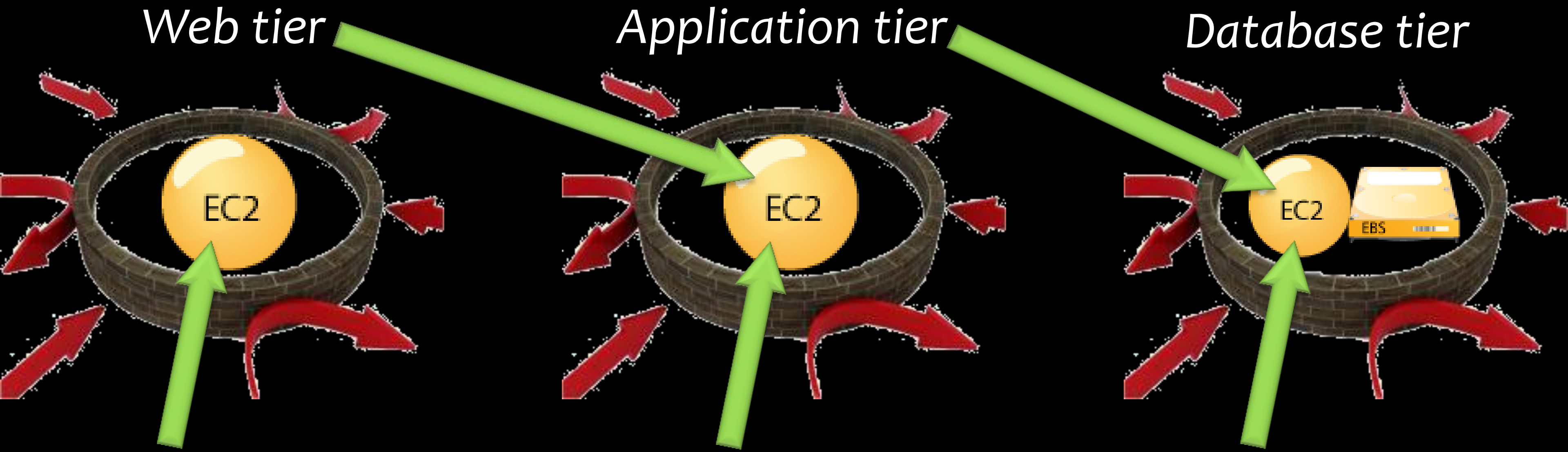
Amazon EC2

4. Build Security in every layer

Design with Security in mind

With cloud, you lose a little bit of physical control but not your ownership

Create distinct Security Groups for each Amazon EC2 cluster
Use group-based rules for controlling access between layers
Restrict external access to specific IP ranges
Encrypt data "at-rest" in Amazon S3
Encrypt data "in-transit" (SSL)
Consider encrypted file systems in EC2 for sensitive data
Rotate your AWS Credentials, Pass in as arguments encrypted
Use MultiFactor Authentication



Web tier

Application tier

Database tier

HTTP/HTTPS
from Internet

SSH/RDP management
from corpnet

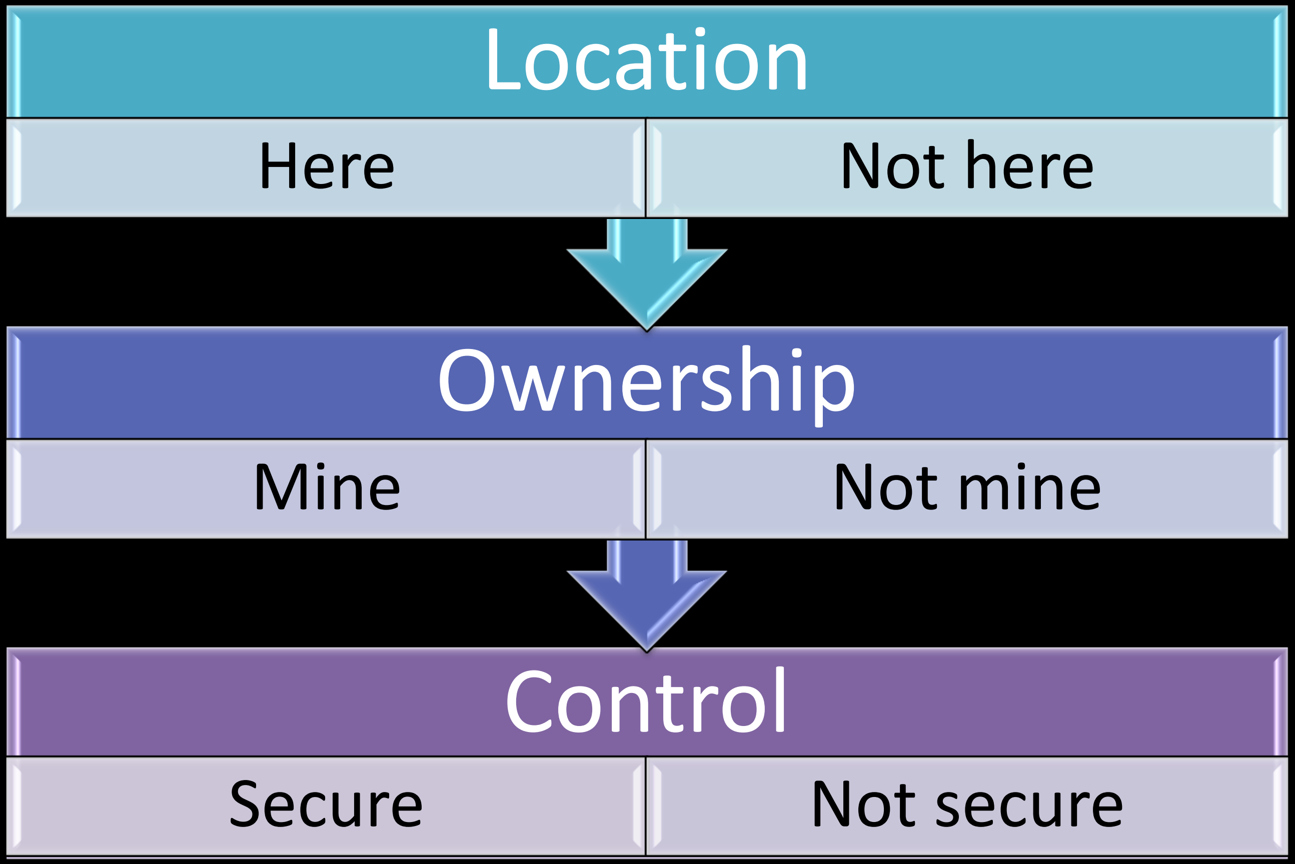
SSH/RDP management
from corpnet, vendor

```
ec2-authorize WebSG -P tcp -p 80 -s 0.0.0.0/0
ec2-authorize WebSG -P tcp -p 443 -s 0.0.0.0/0

ec2-authorize AppSG -P tcp -p AppPort -o WebSG
ec2-authorize AppSG -P tcp -p 22|3389 -s CorpNet

ec2-authorize DBSG -P tcp -p DBPort -o AppSG
ec2-authorize DBSG -P tcp -p 22|3389 -s CorpNet
ec2-authorize DBSG -P tcp -p 22|3389 -s Vendor
```

Traditional security model



Location

Here | Not here

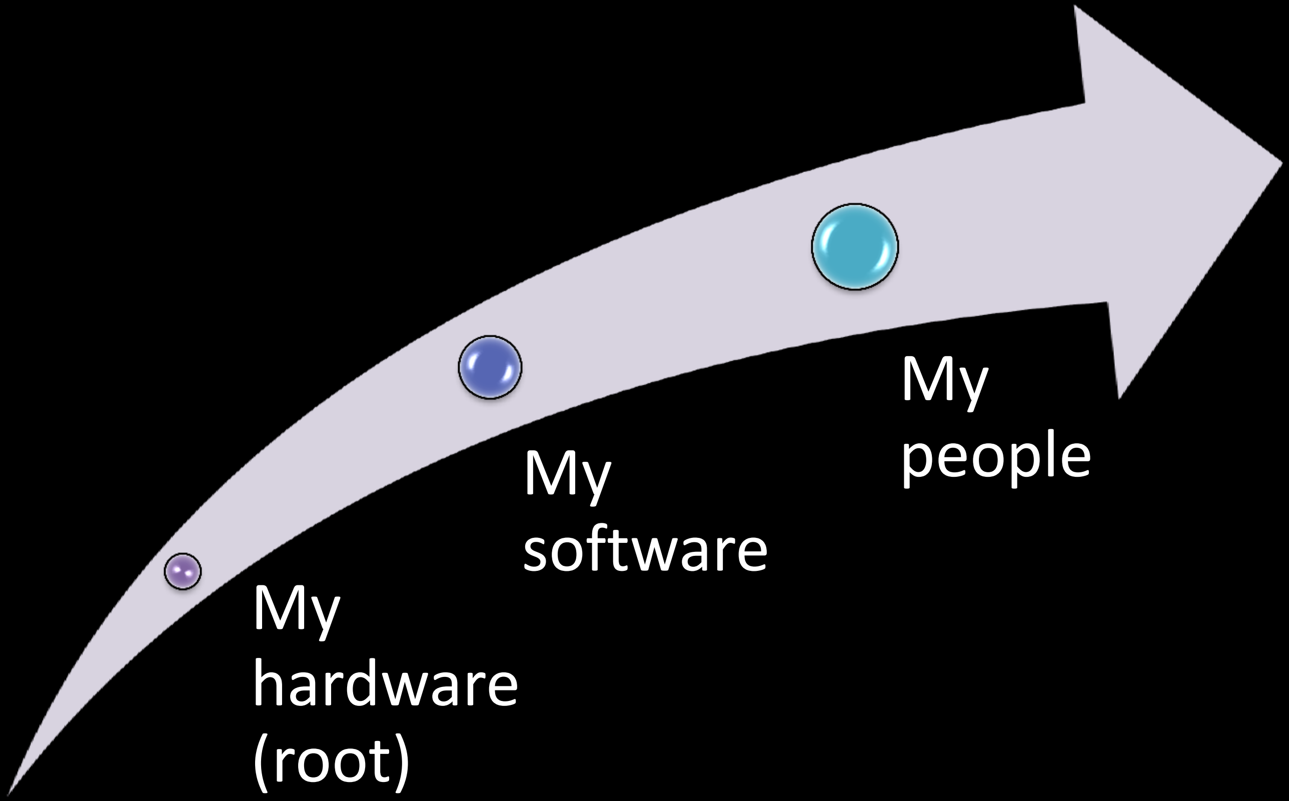
Ownership

Mine | Not mine

Control

Secure | Not secure

Layers of trust



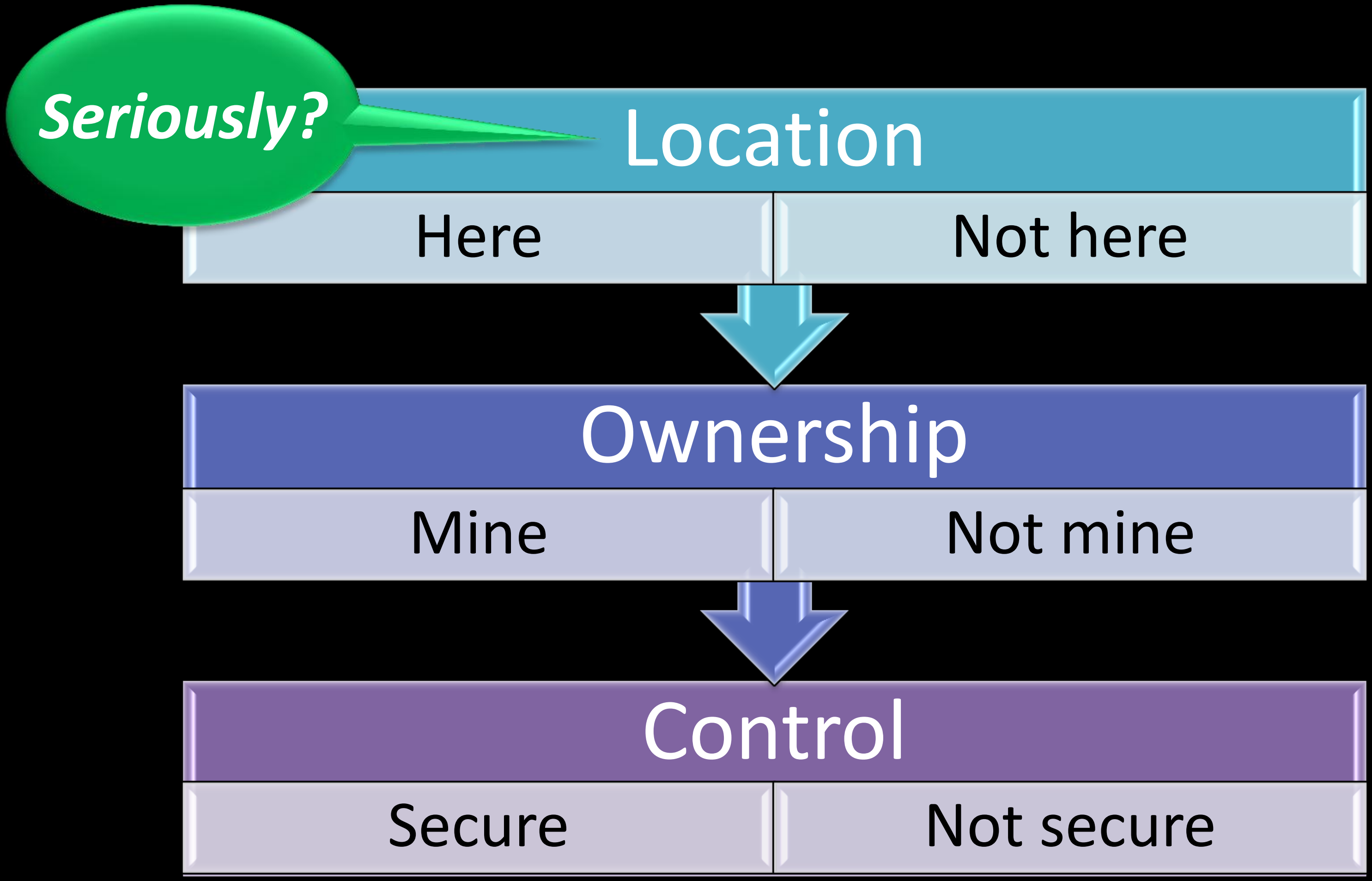
My hardware (root)

My software

My people

Perimeters separate trusted (owned, local) from untrusted (other, remote)

The model is breaking



Seriously?

Location

Here

Not here

Ownership

Mine

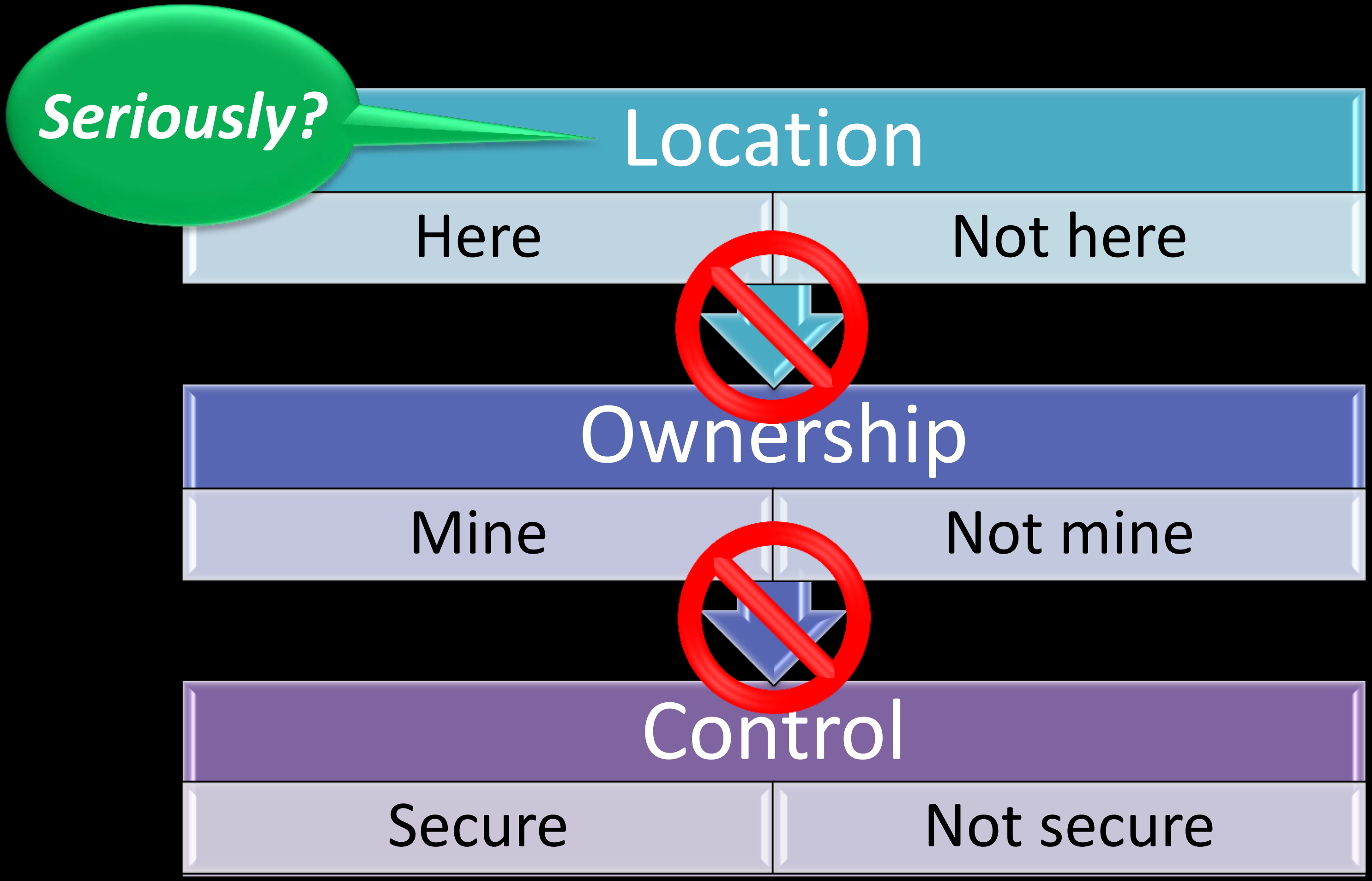
Not mine

Control

Secure

Not secure

The model is breaking



Seriously?

Location

Here

Not here

Ownership

Mine

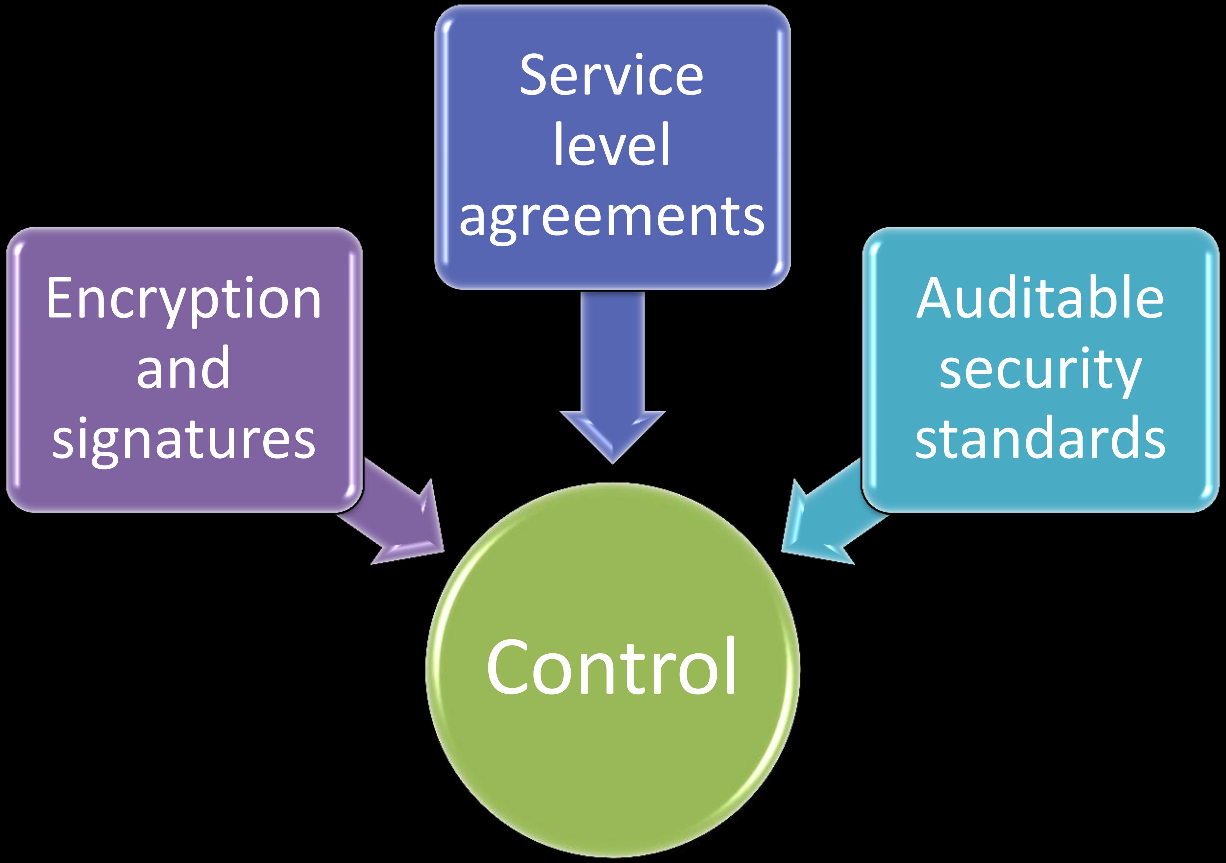
Not mine

Control

Secure

Not secure

New security model



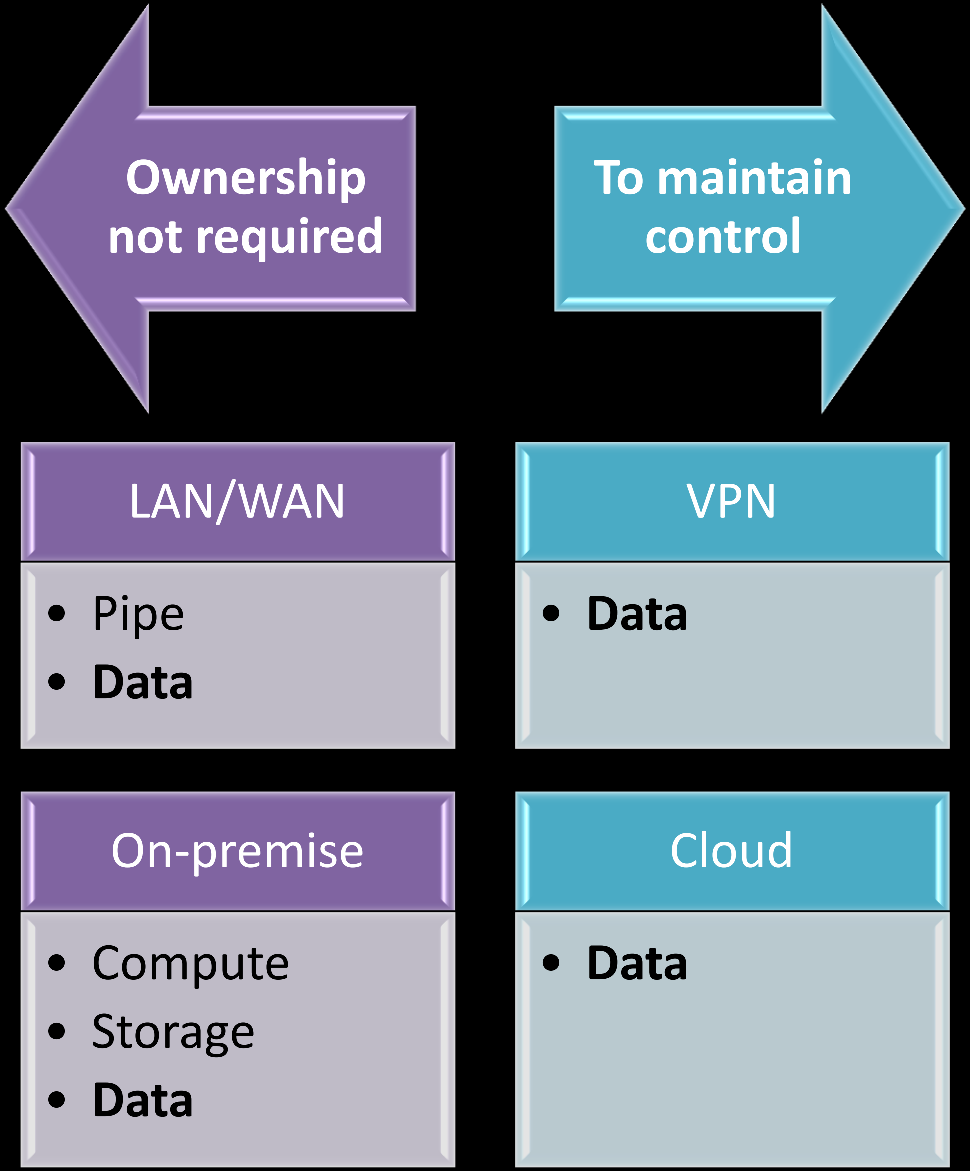
Service level agreements

Encryption and signatures

Auditable security standards

Control

Ownership vs. control



Ownership not required

To maintain control

LAN/WAN

- Pipe
- **Data**

VPN

- **Data**

On-premise

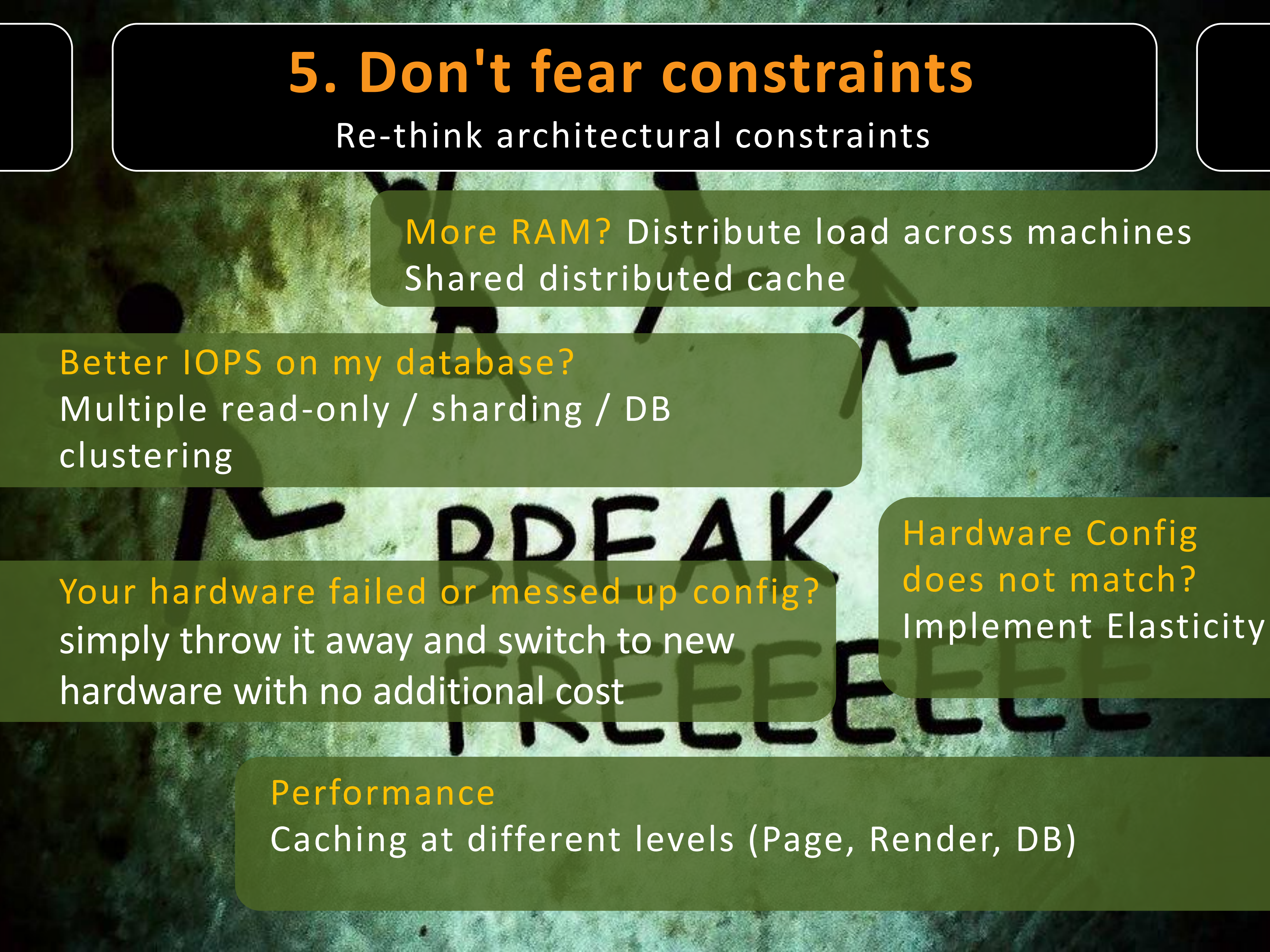
- Compute
- Storage
- **Data**

Cloud

- **Data**

5. Don't fear constraints

Re-think architectural constraints



More RAM? Distribute load across machines
Shared distributed cache

Better IOPS on my database?
Multiple read-only / sharding / DB clustering

Hardware Config does not match?
Implement Elasticity

Your hardware failed or messed up config?
simply throw it away and switch to new hardware with no additional cost

Performance
Caching at different levels (Page, Render, DB)

6. Think Parallel

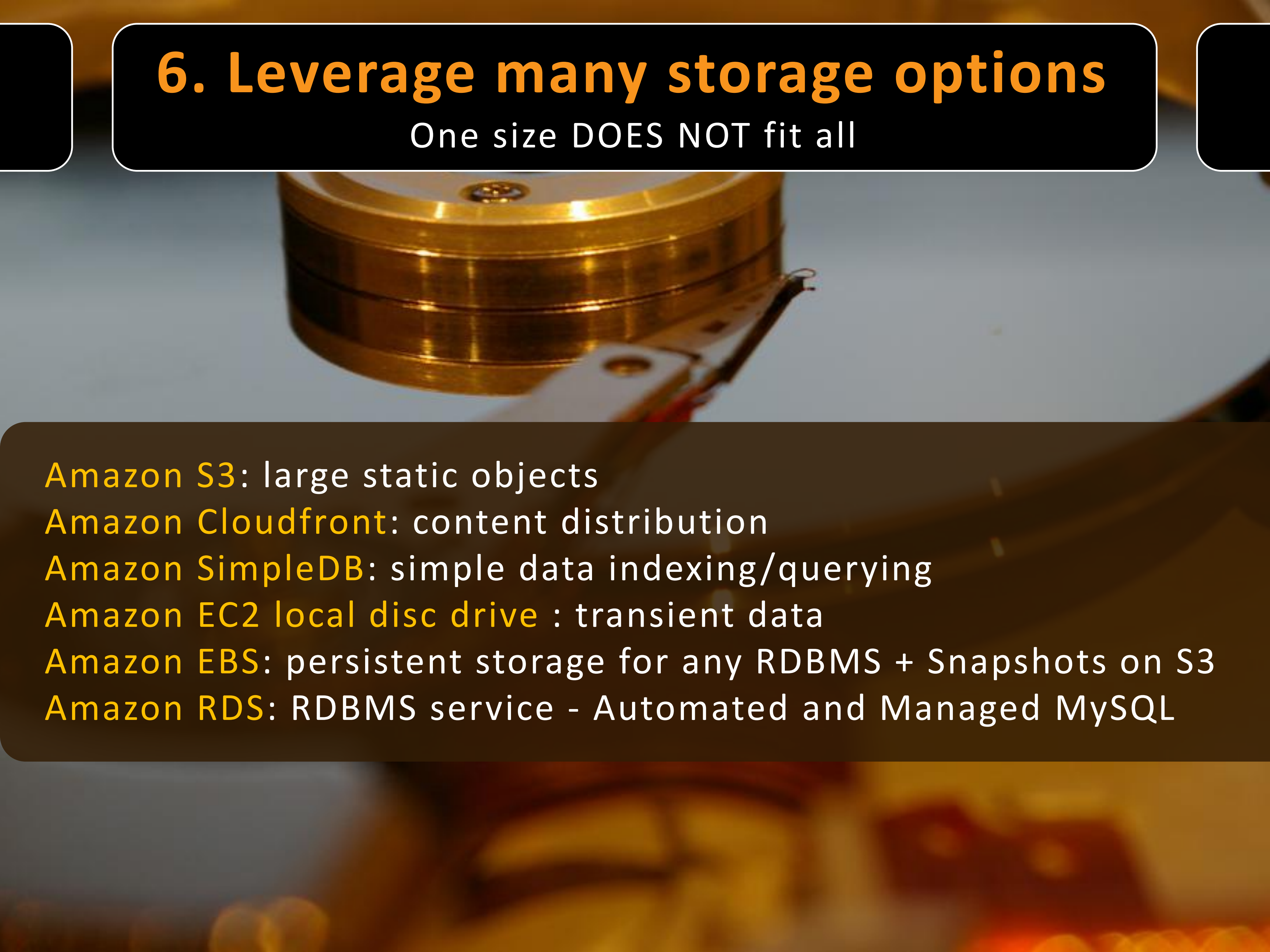
Serial and Sequential is now history

Experiment different architectures in parallel
Multi-threading and Concurrent requests to cloud services
Run parallel MapReduce Jobs
Use Elastic Load Balancing to distribute load across multiple servers
Decompose a Job into its simplest form – and with “shared nothing”

The beauty of the cloud shines when you combine elasticity and parallelization

6. Leverage many storage options

One size DOES NOT fit all



Amazon S3: large static objects
Amazon Cloudfront: content distribution
Amazon SimpleDB: simple data indexing/querying
Amazon EC2 local disc drive : transient data
Amazon EBS: persistent storage for any RDBMS + Snapshots on S3
Amazon RDS: RDBMS service - Automated and Managed MySQL

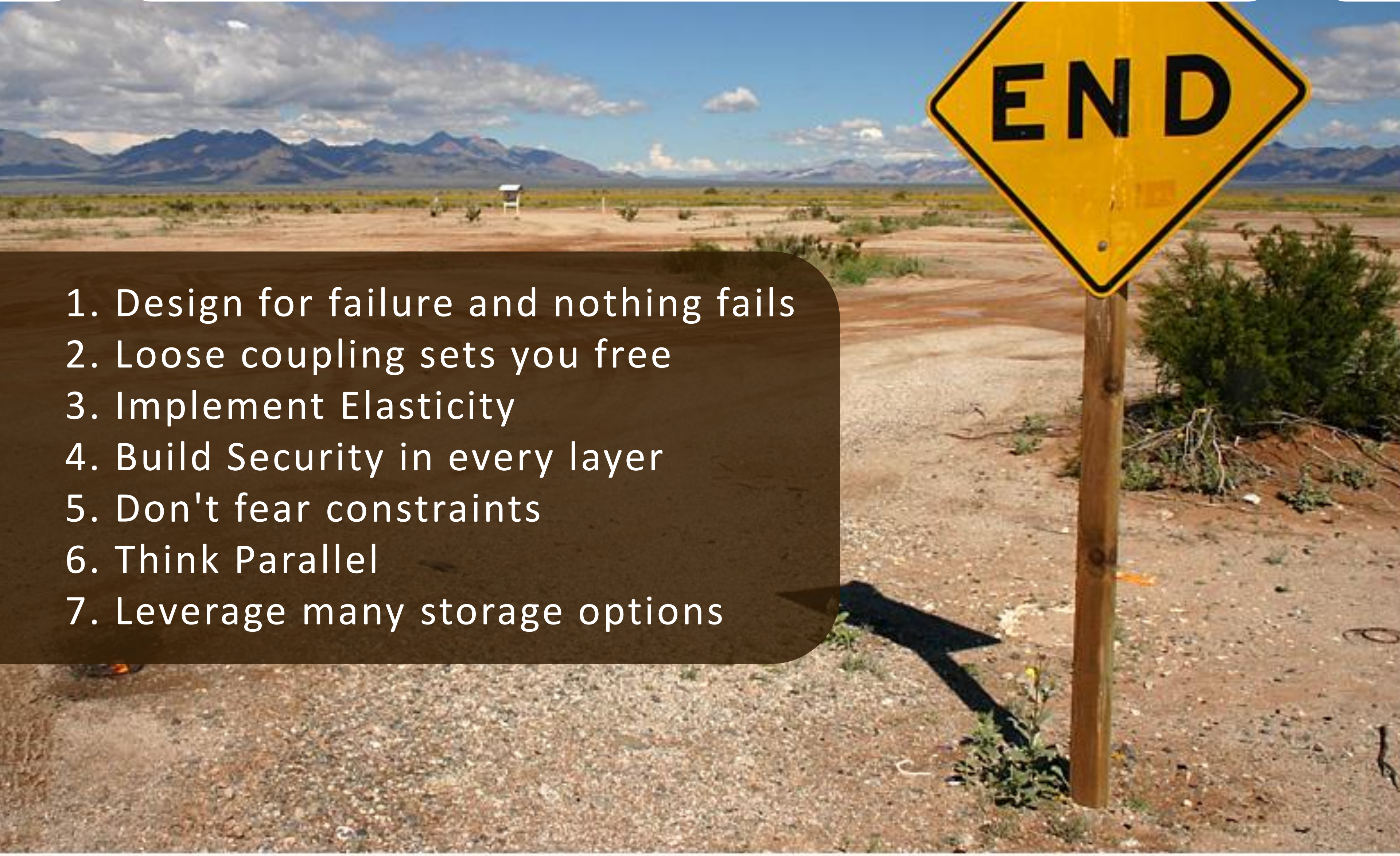
6. Leverage many storage options

Which storage option to use when?

	Amazon S3 + CF	Amazon EC2 Ephemeral Store	Amazon EBS	Amazon SimpleDB	Amazon RDS
Ideal for	Storing Large write-once, read-many types of objects, Static Content Distribution	Storing non-persistent transient updates	Off-instance persistent storage for any kind of data,	Querying light-weight attribute data	Storing and querying structured Relational and referential Data
Ideal examples	Media files, audio, video, images, Backups, archives, versioning	Config Data, scratch files, TempDB	Clusters, boot data, Log or data of commercial RDBMS like Oracle, DB2	Querying, Mapping, tagging, click-stream logs, metadata, shared-state management, indexing	Complex transactional systems, inventory management and order fulfillment systems
Not recommended for	Querying, Searching	Storing Database logs or backups, customer data		Relational (joins) query	
Not recommended examples	Database, File Systems	Sensitive data	Content Distribution	OLTP, DW cube rollups	Simple lookups

Cloud Architecture Lessons

Best Practices



1. Design for failure and nothing fails
2. Loose coupling sets you free
3. Implement Elasticity
4. Build Security in every layer
5. Don't fear constraints
6. Think Parallel
7. Leverage many storage options

Migrating your Web Application

Step by Step towards AWS

A typical Web App needs:	With AWS:
Compute Power	Amazon EC2
Storage capacity	Amazon S3
Content Distribution	Amazon CloudFront
Database storage	Amazon EBS
Messaging	Amazon SQS
Load balancing	Amazon EC2
Monitoring	Amazon CloudWatch

Amazon Web Services tools

Things you need

Web : AWS Management Console

IDE : AWS Toolkit for Eclipse

AWS SDK: .NET SDK, Java SDK

Tools : 3rd Party tools eg. CA

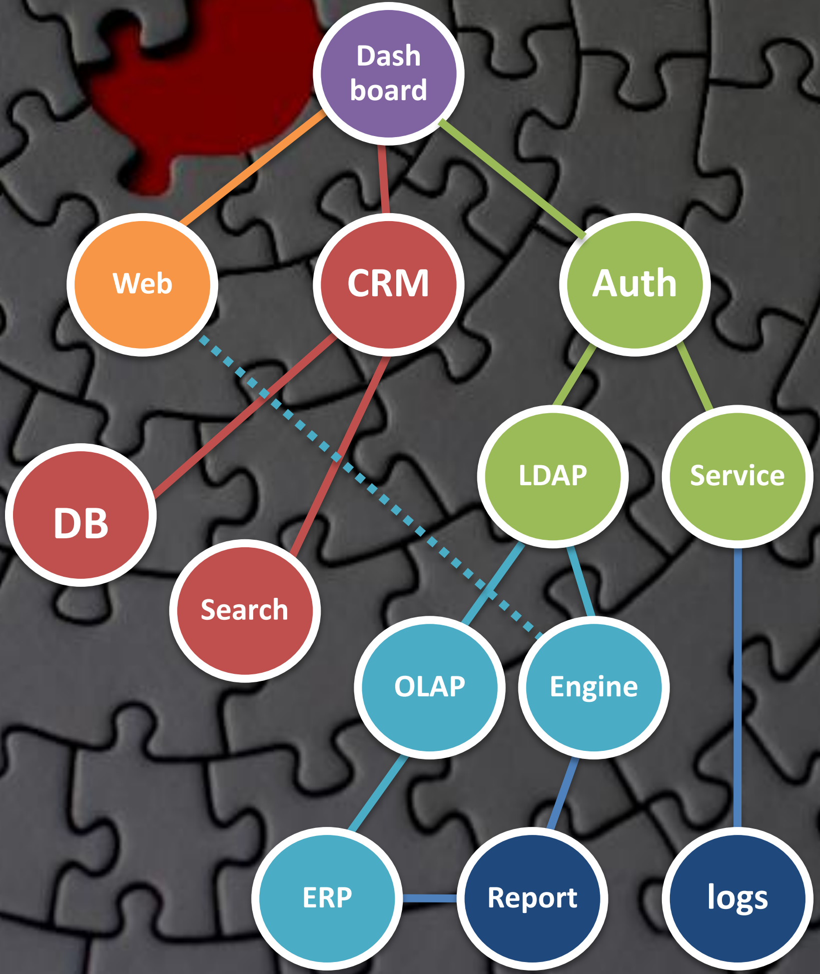
Firefox Plugins :

ElasticFox, S3Fox, SDB Tool

Several libraries: boto, cloudfusion

Identify the right candidate

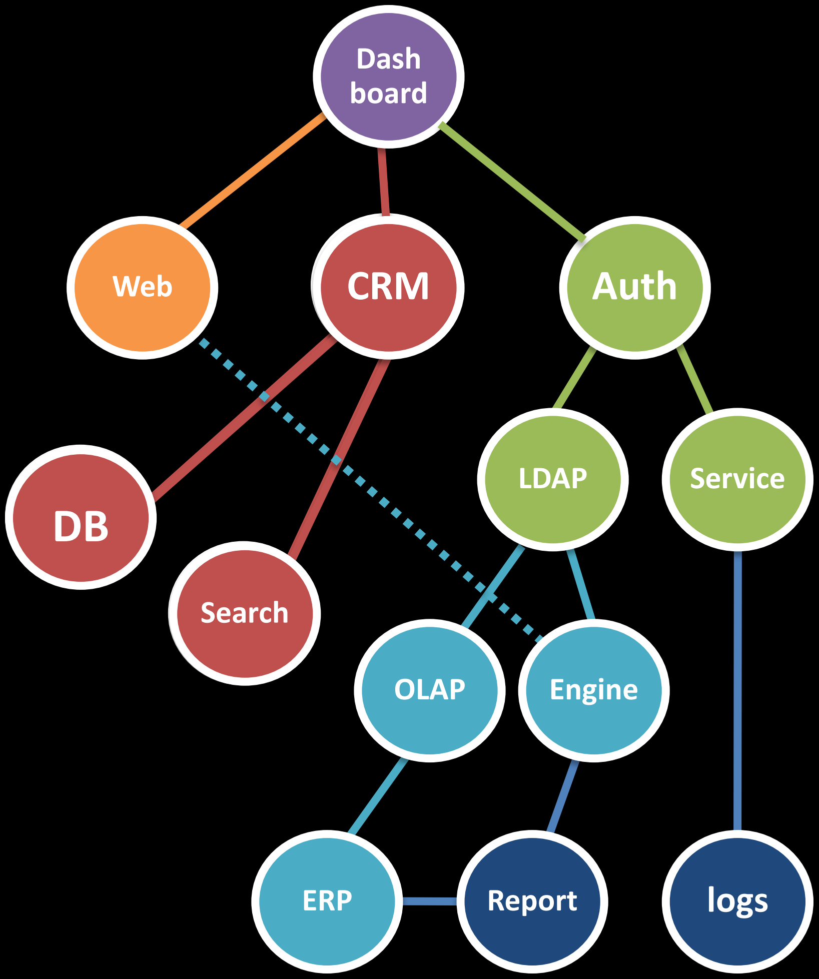
Assessment



List all your IT assets
Whiteboard your IT Assets
Identify upward and downward dependencies

Identify the right candidate

Pick one application with lower dependencies to start with



Search for under-utilized IT assets

Applications that has immediate business need to scale

Applications that are running out of capacity

Low-hanging fruits (Examples):

Web Applications

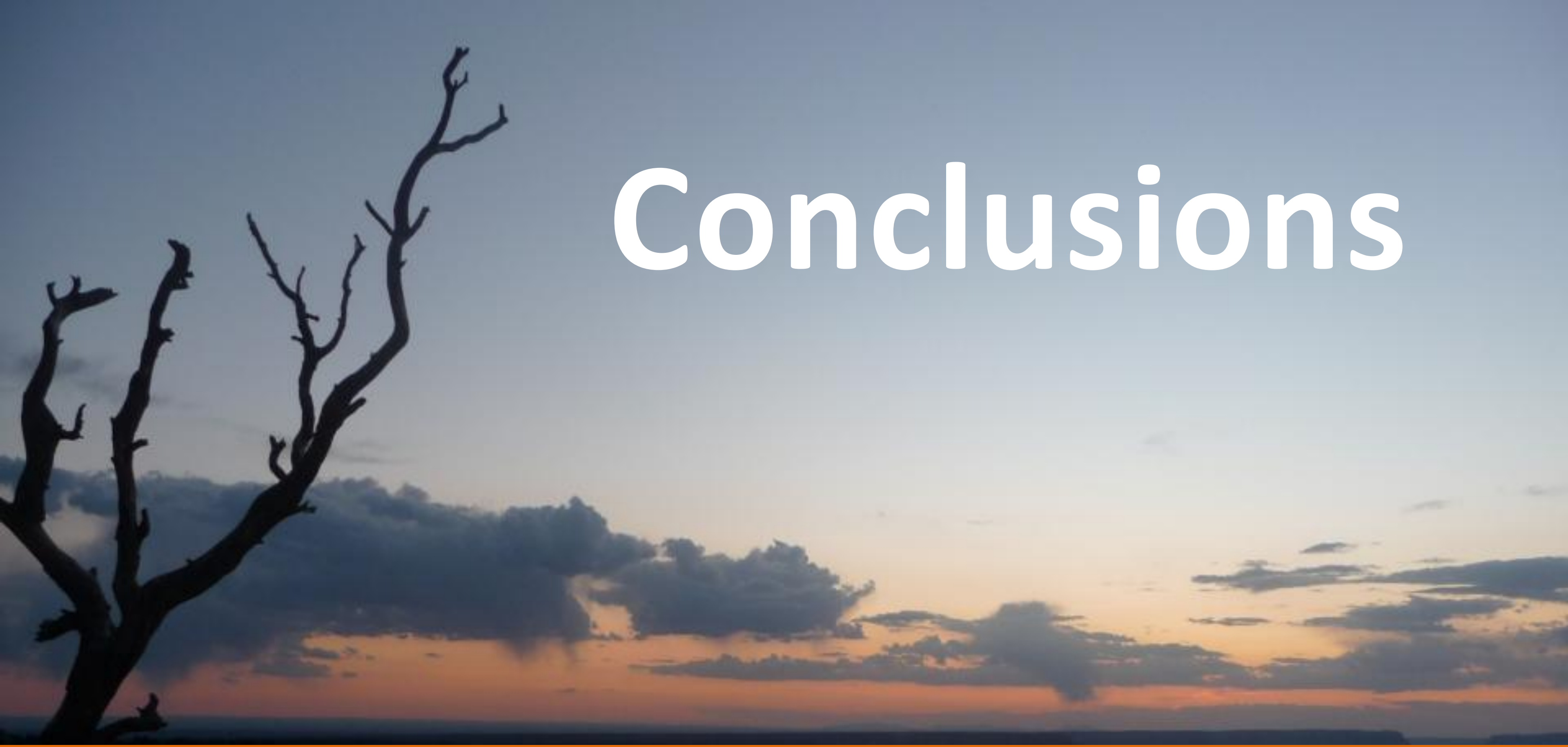
Batch Processing systems

Build/QA/Test systems

Content Management Systems

Digital Asset Management Systems

Conclusions



Most Important Lesson From Our Customers:

Start small with a well-defined proof of concept

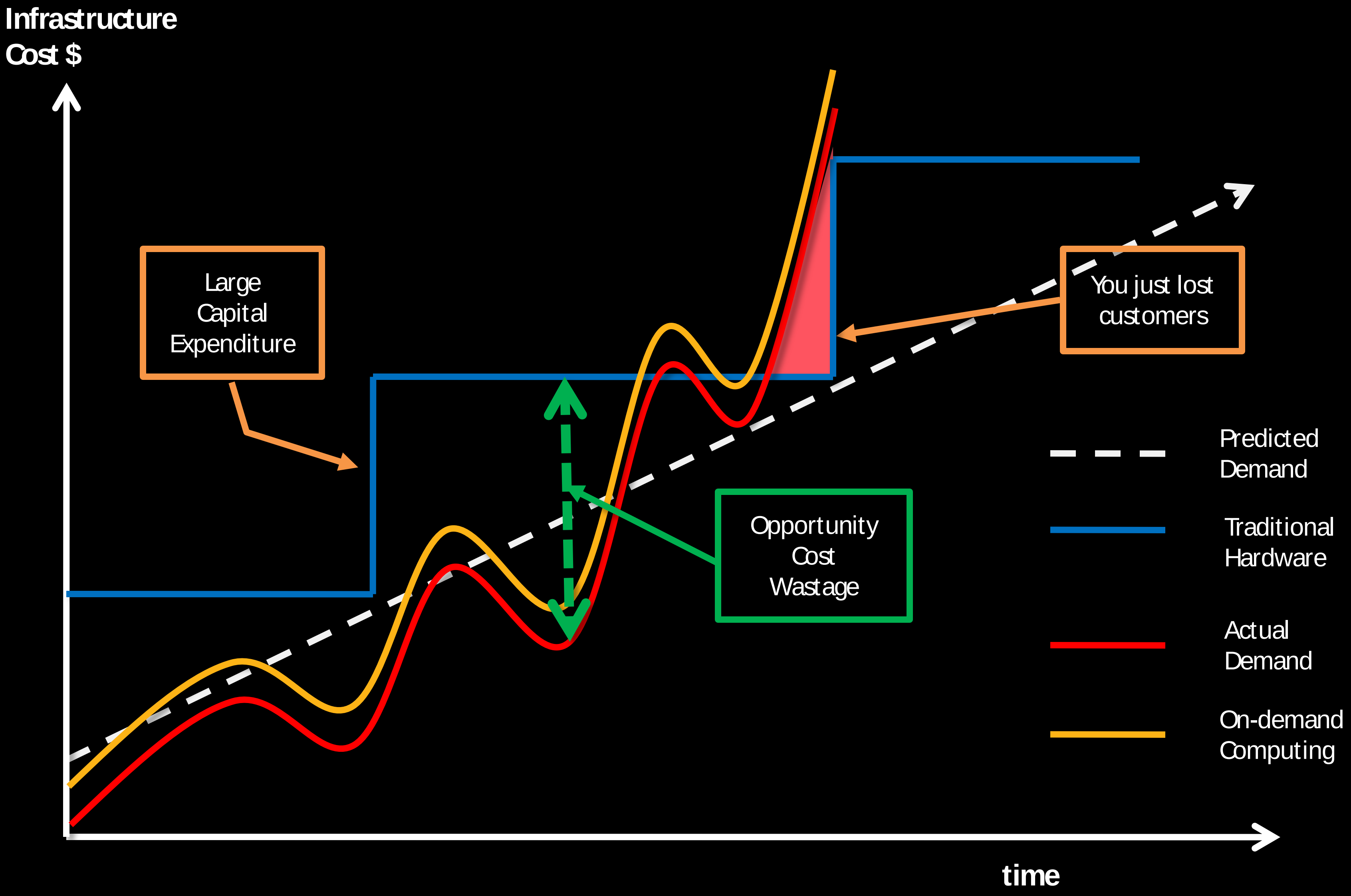
Experiment with different architectures; Keep one, throw away others

Once one application is launched others will follow…

Traditional IT roles are changing

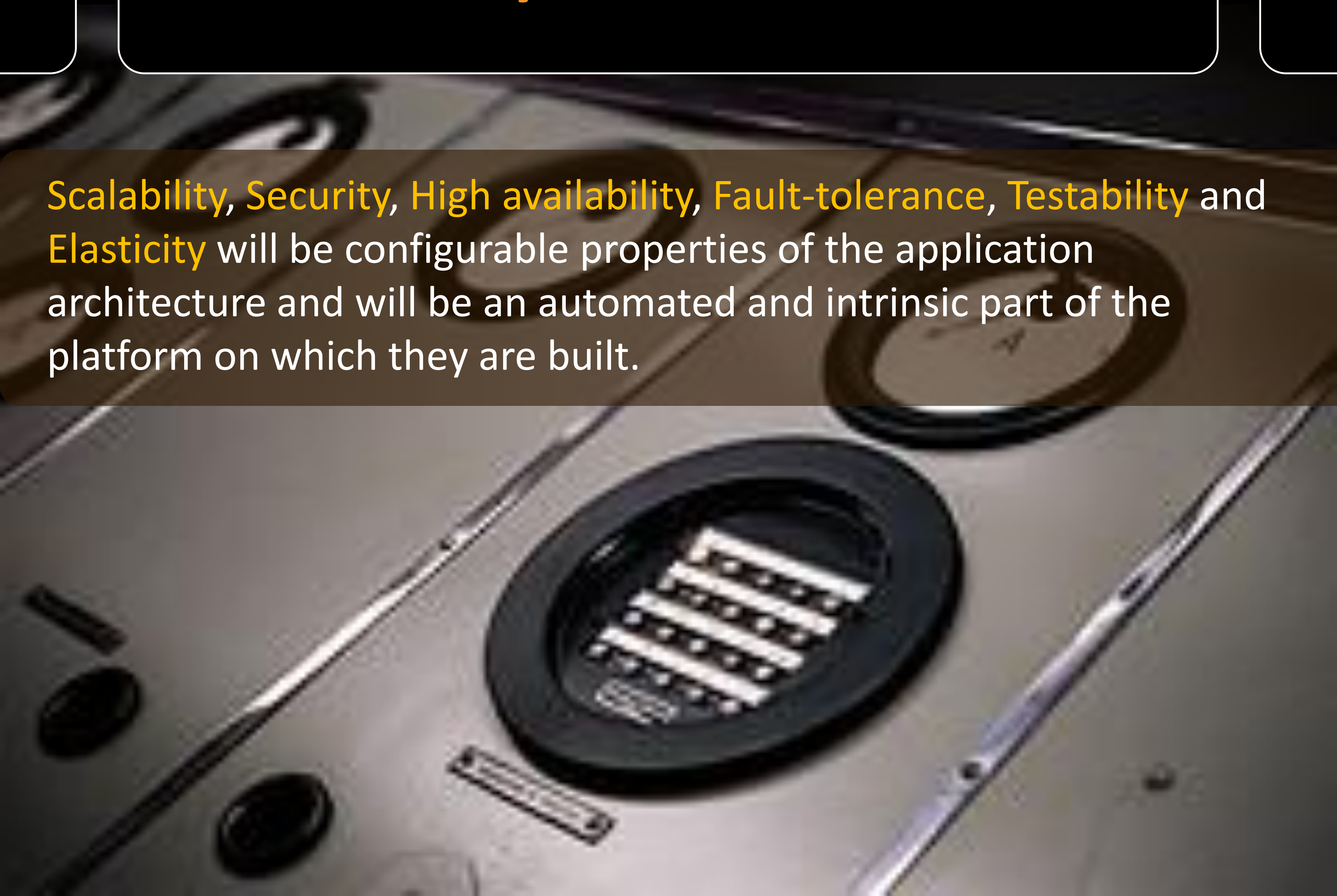
Photo: Grand Canyon Hopi Point SunSet

Predicting Infrastructure Needs



The day is not too far….

Scalability, Security, High availability, Fault-tolerance, Testability and Elasticity will be configurable properties of the application architecture and will be an automated and intrinsic part of the platform on which they are built.



Thank you!

steriley@amazon.com

@steveriley @awscloud

http://stvrly.wordpress.com

Presentation ideas and template from @simon and @jinman