Design Patterns for High Availability

Lessons learned building Amazon CloudFront

CTD303

Alex Smith
Head of M&E Architecture, APAC
Amazon Web Services

Harvo Jones
Sr. Software Development Engineer
Amazon CloudFront

11/30/2016

amazon web services

What to Expect from the Session

- Learn about the design patterns for high availability of Amazon CloudFront

- Learn how you can implement the patterns in your own services or applications built on top of AWS

What is a Content Delivery Network (CDN)?

Amazon CloudFront locations worldwide

68 POPs

21 Countries

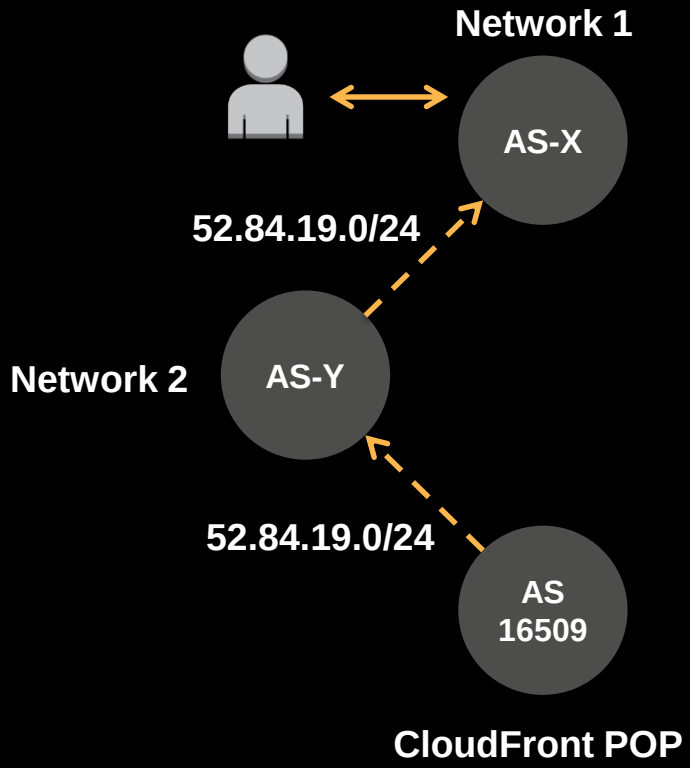
43 Cities

5 Continents

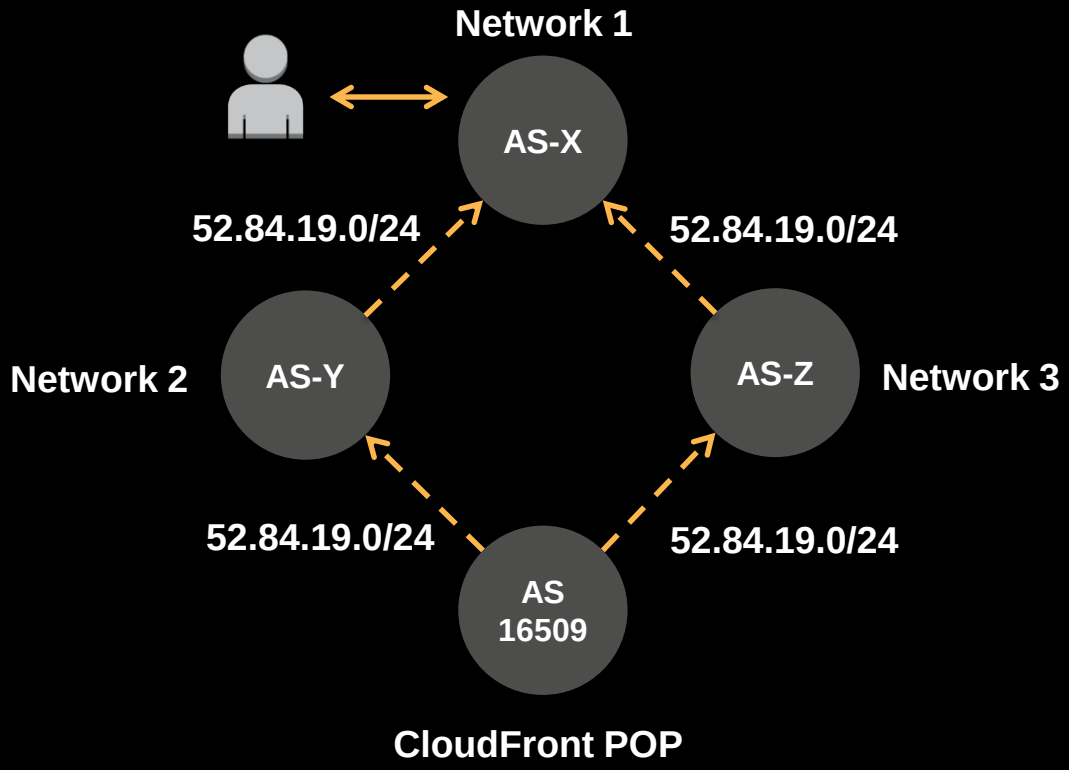
North America | South America | EMEA | APAC

AWS Region CloudFront Edge Location

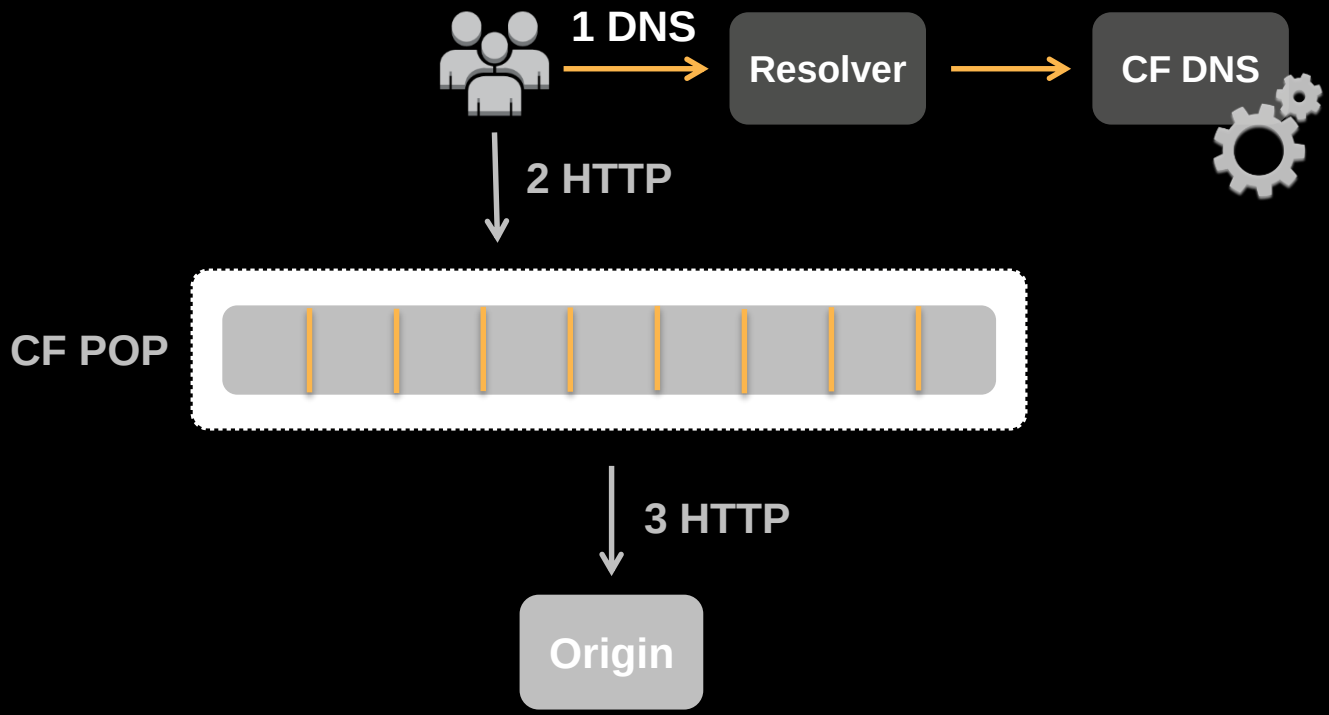
How does Amazon CloudFront work?



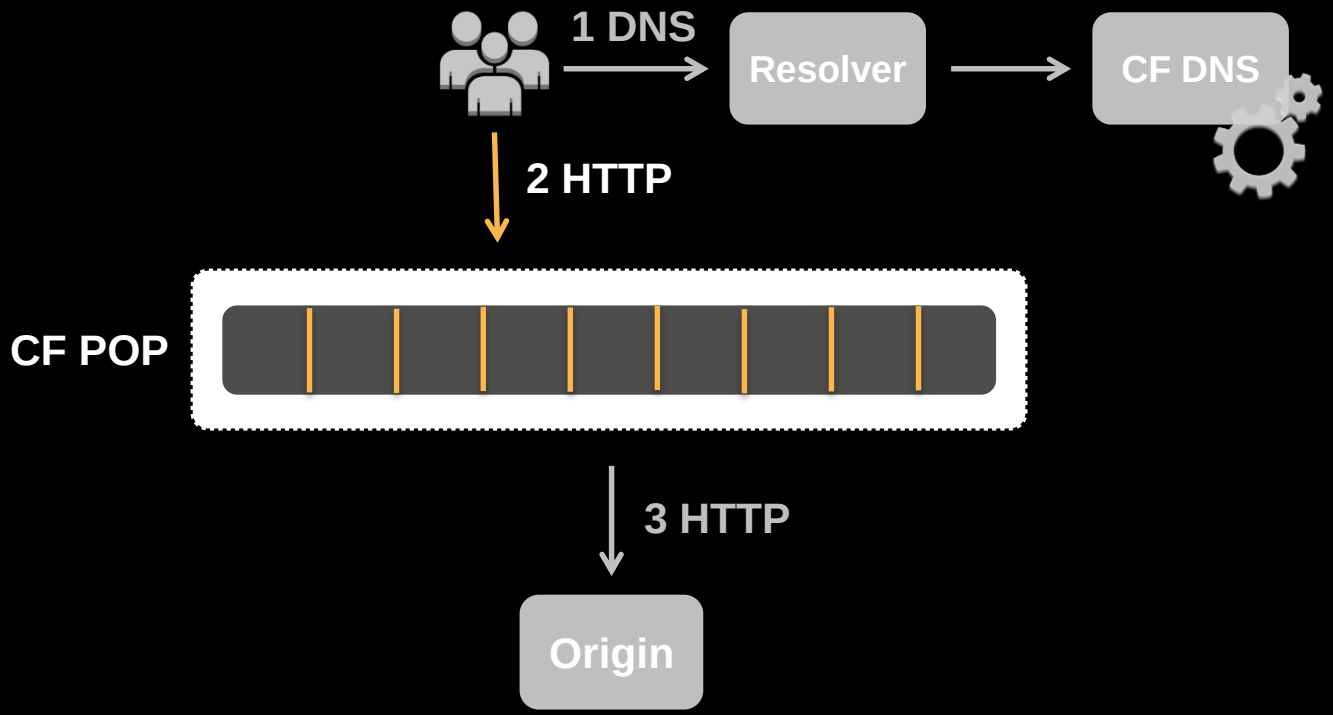
How does Amazon CloudFront work?



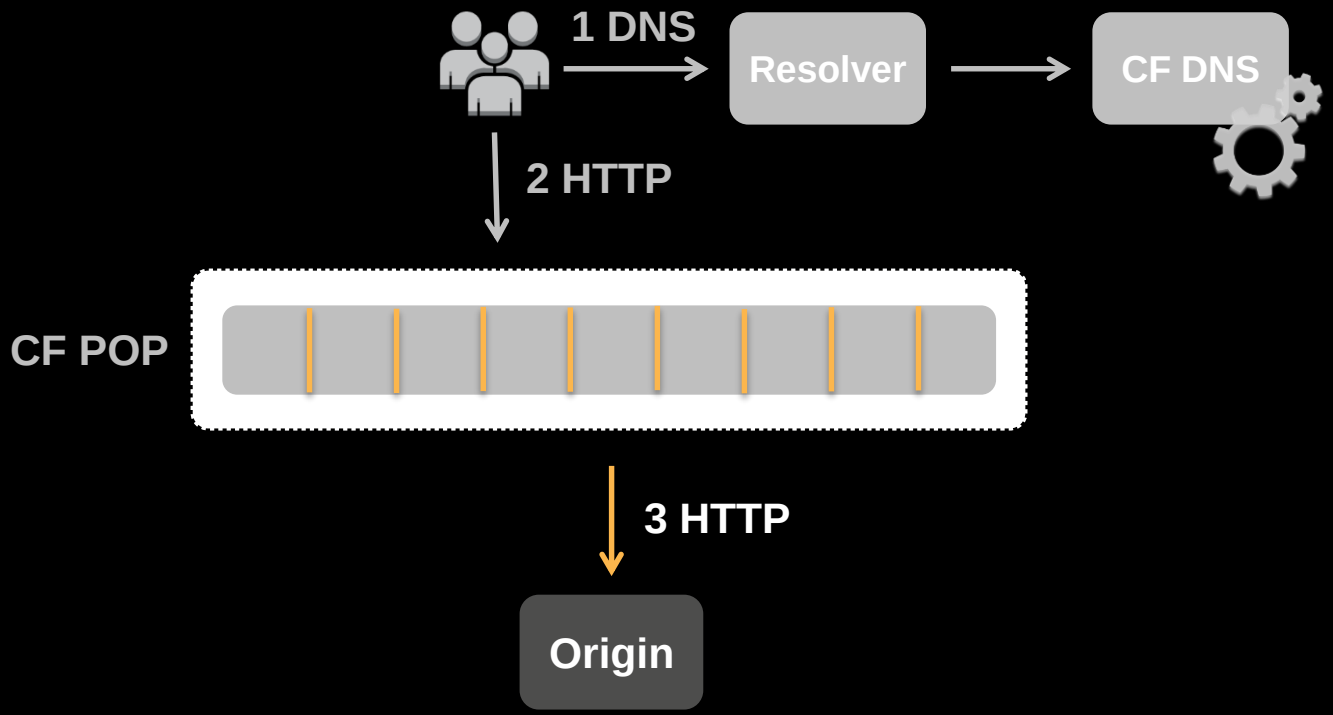
Sequence of events



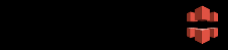
Sequence of events



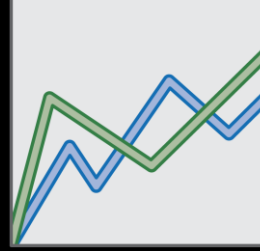
Sequence of events



How to monitor availability

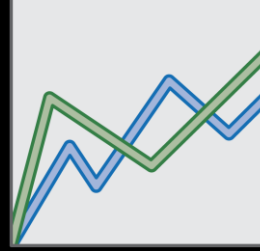


How CloudFront monitors availability



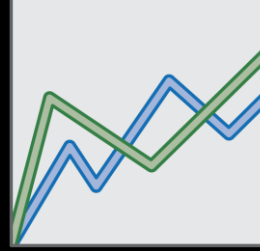
1. Analysis of server-side metrics

How CloudFront monitors availability



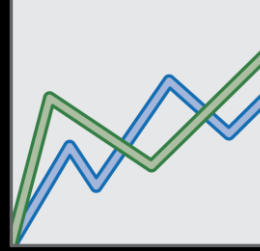
1. Analysis of server-side metrics
2. **Canaries**

How CloudFront monitors availability

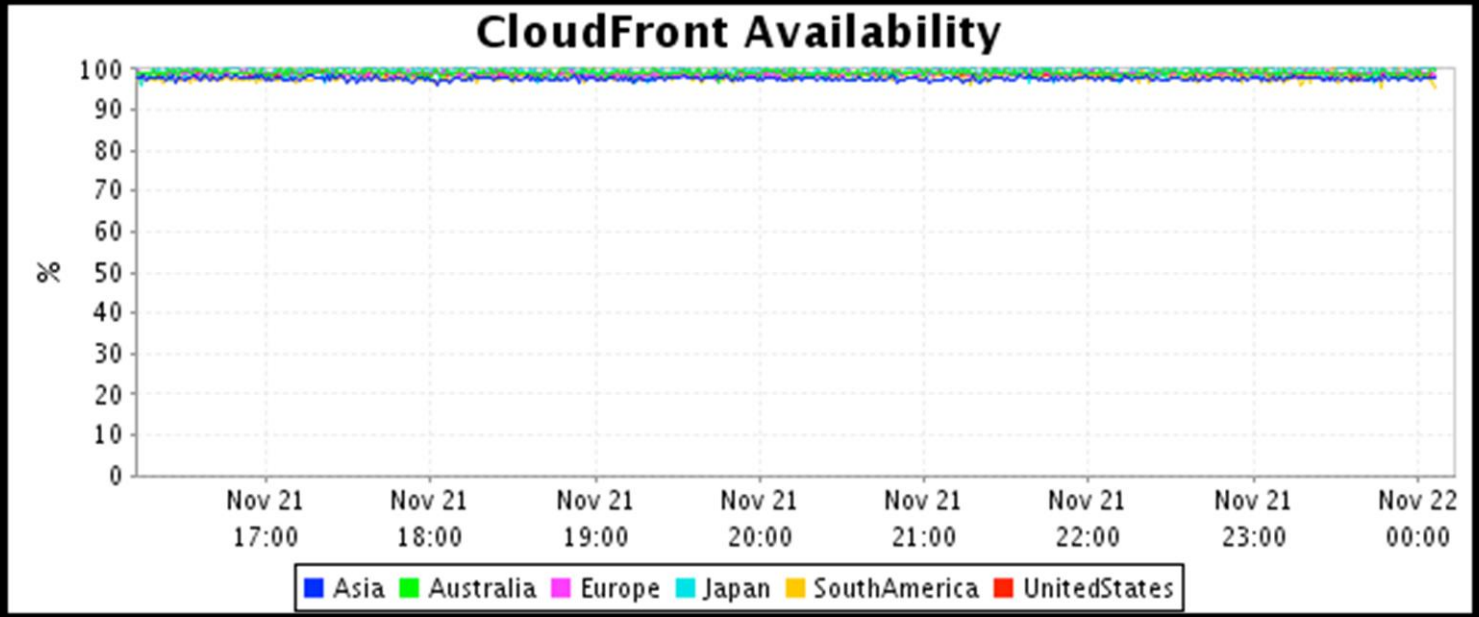


1. Analysis of server-side metrics
2. Canaries
3. **Third-party global HTTP tests**

How CloudFront monitors availability



1. Analysis of server-side metrics
2. Canaries
3. Third party global HTTP tests

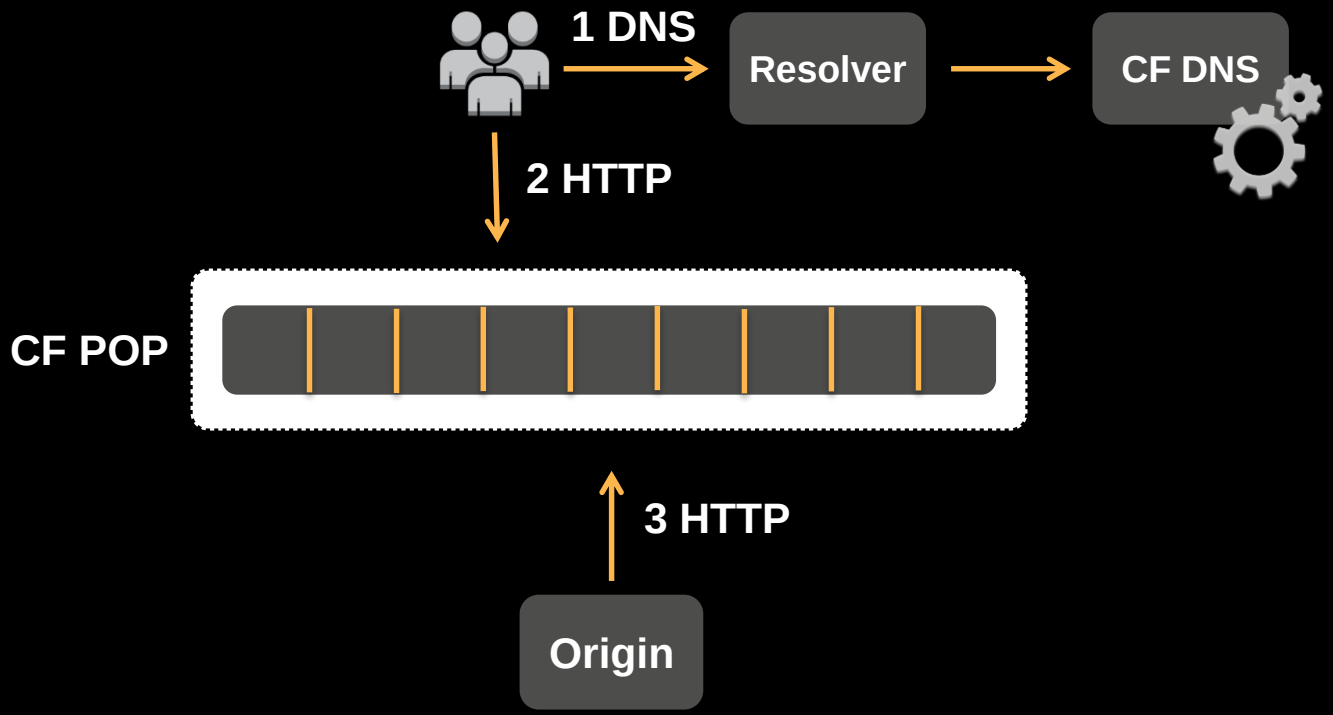


Availability interruption

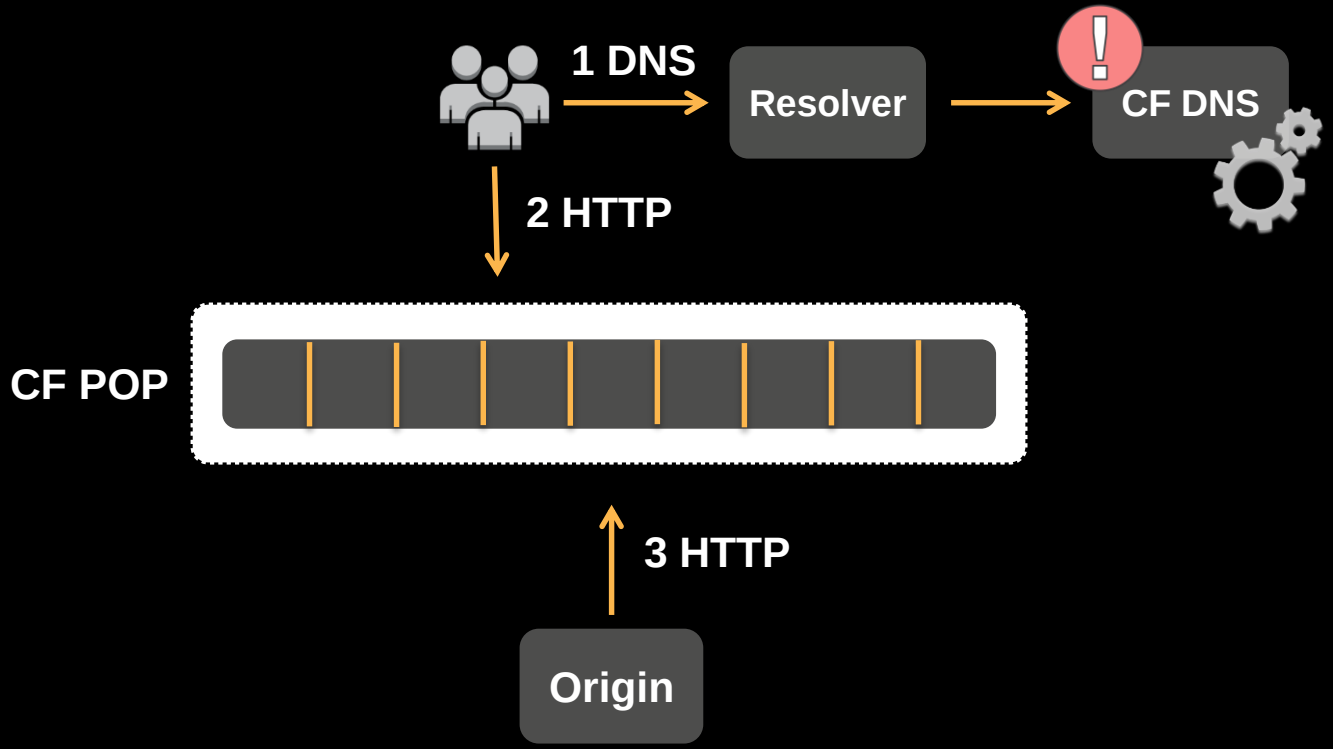
Segfault in CloudFront DNS server

```
27: int dns_get_domain_length(const char *domain) {  /* <== domain is NULL */
28:     const char *pos;
29:     unsigned char ch;
30:
31:     pos = domain;
32:     while ((ch = *pos++) != 0)          /* <== SEGFAULT */
33:         pos += (unsigned int) ch;
34:
35:     return pos - domain;
36: }
```

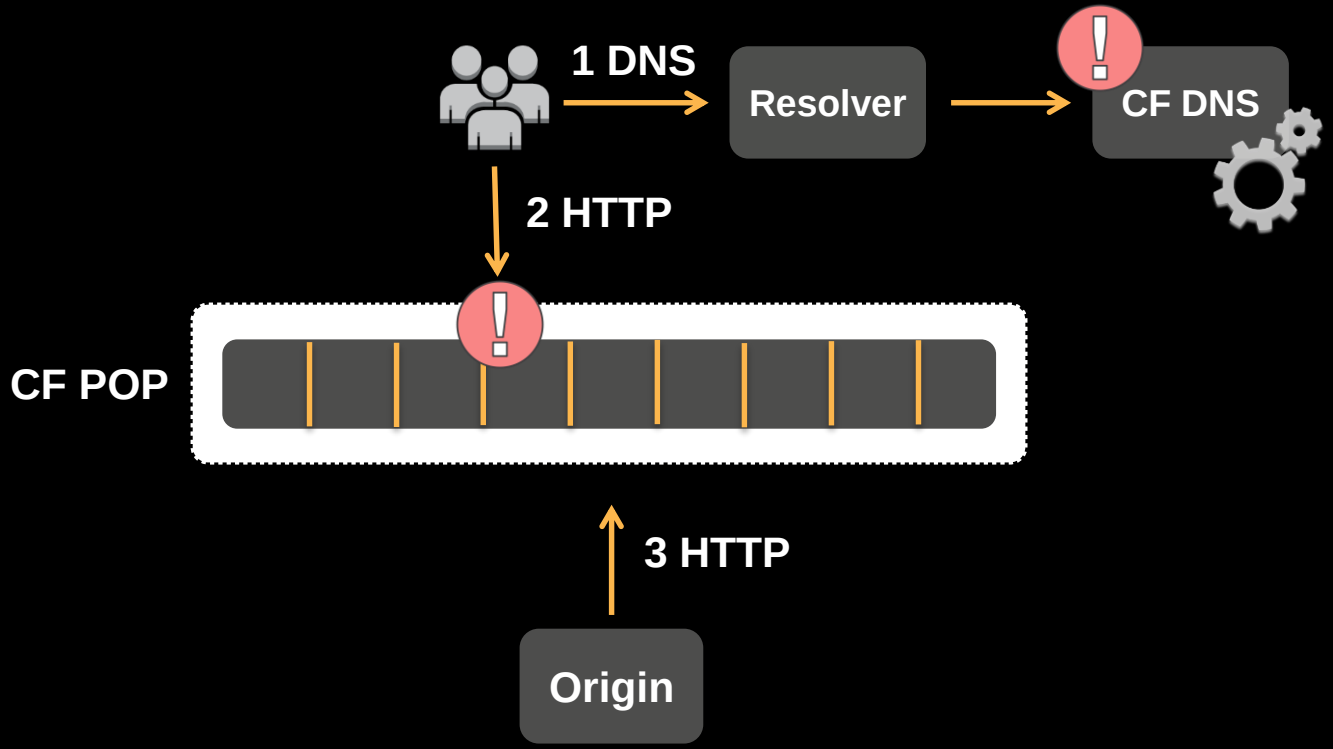
Potential causes of application failure



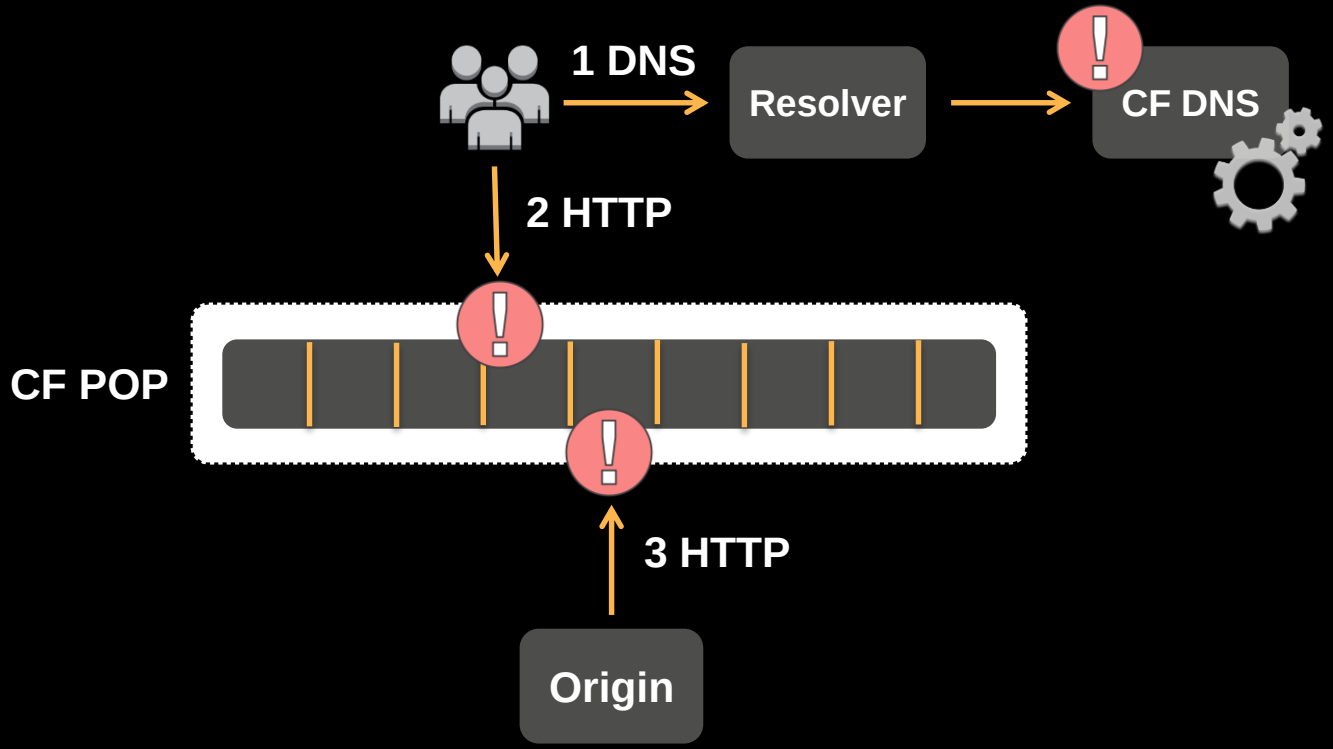
Potential causes of application failure



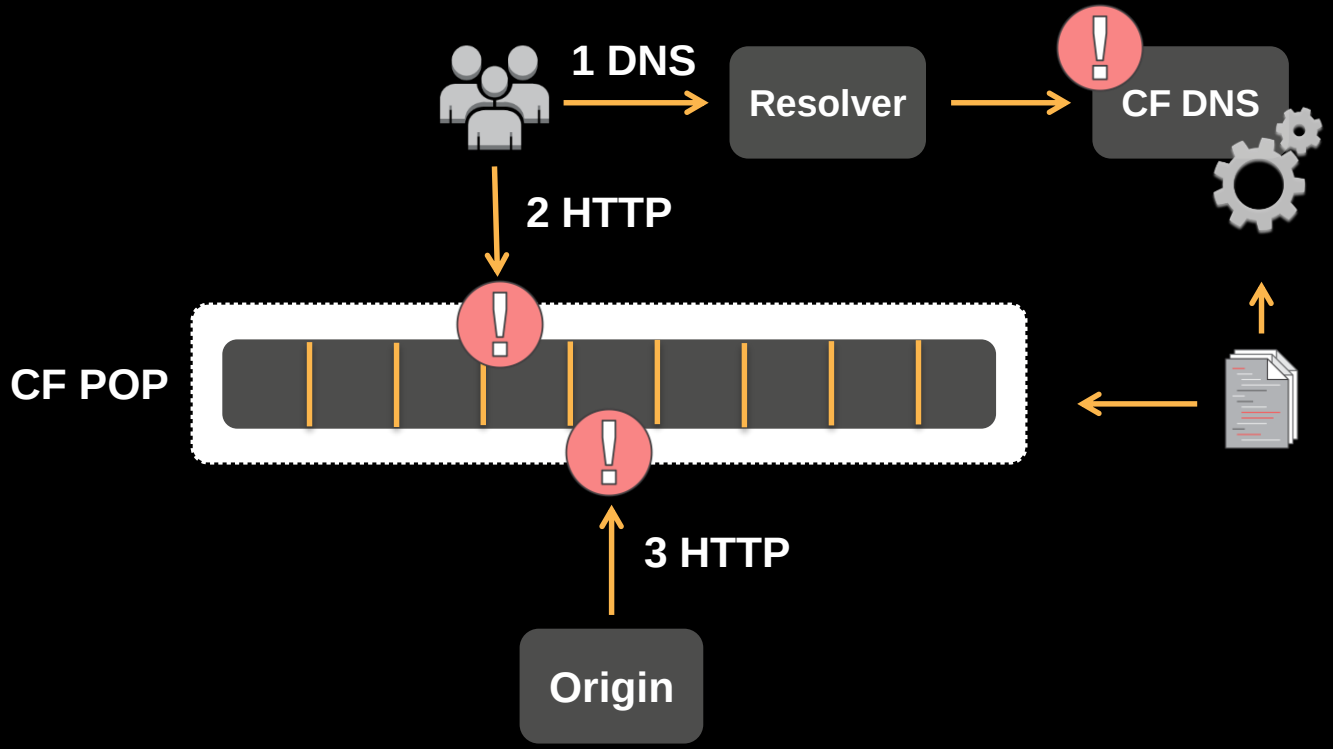
Potential causes of application failure



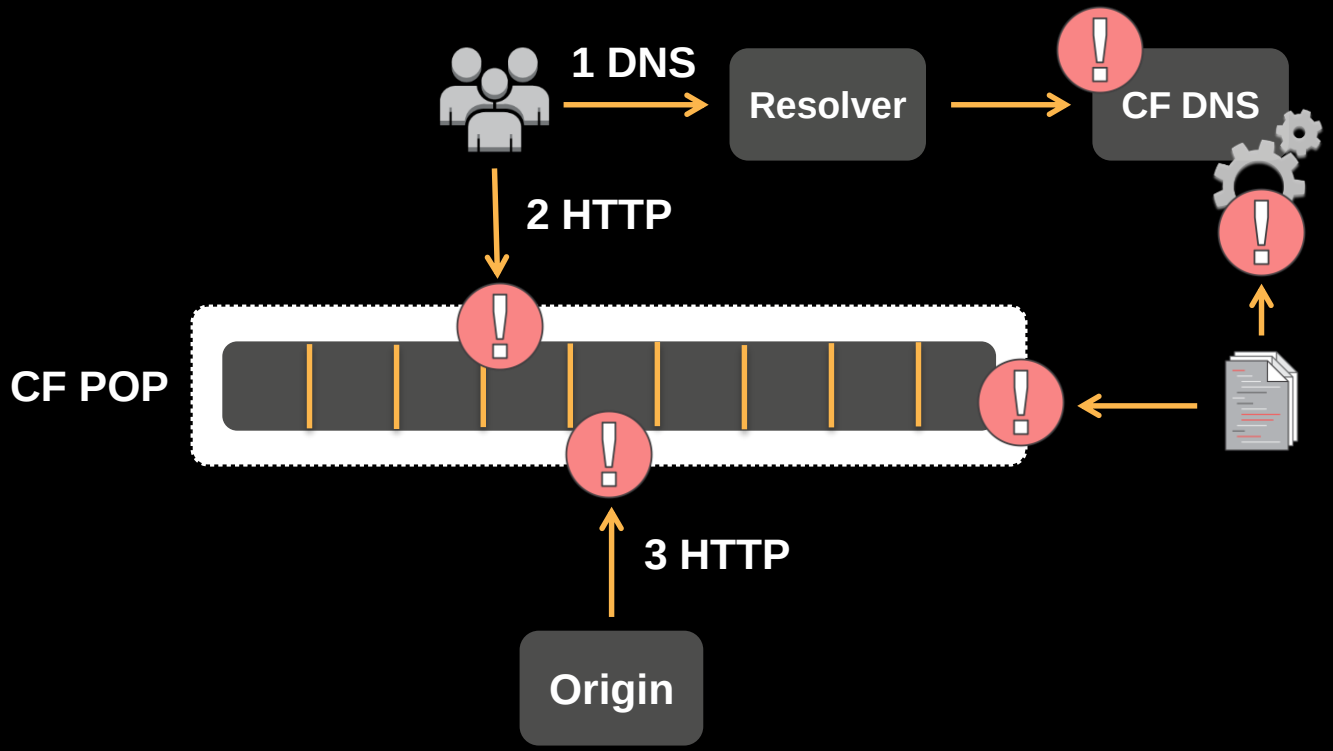
Potential causes of application failure



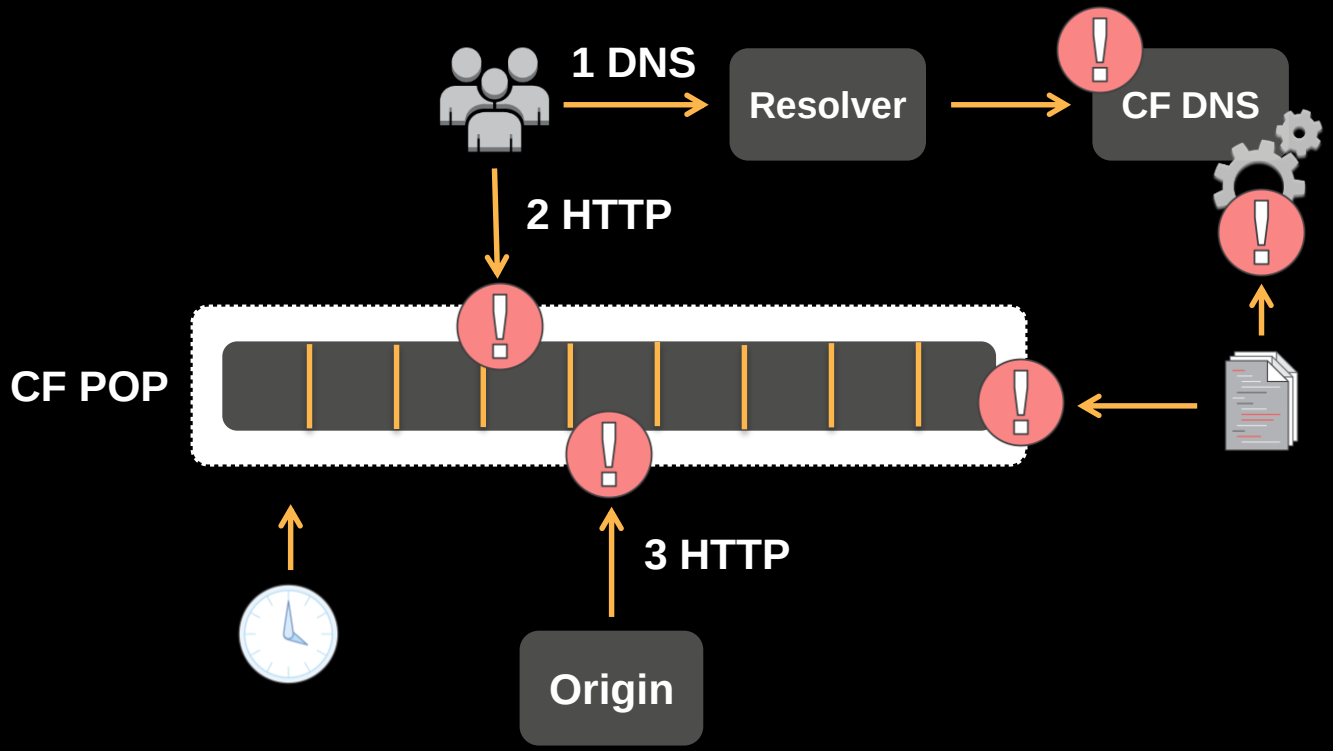
Potential causes of application failure



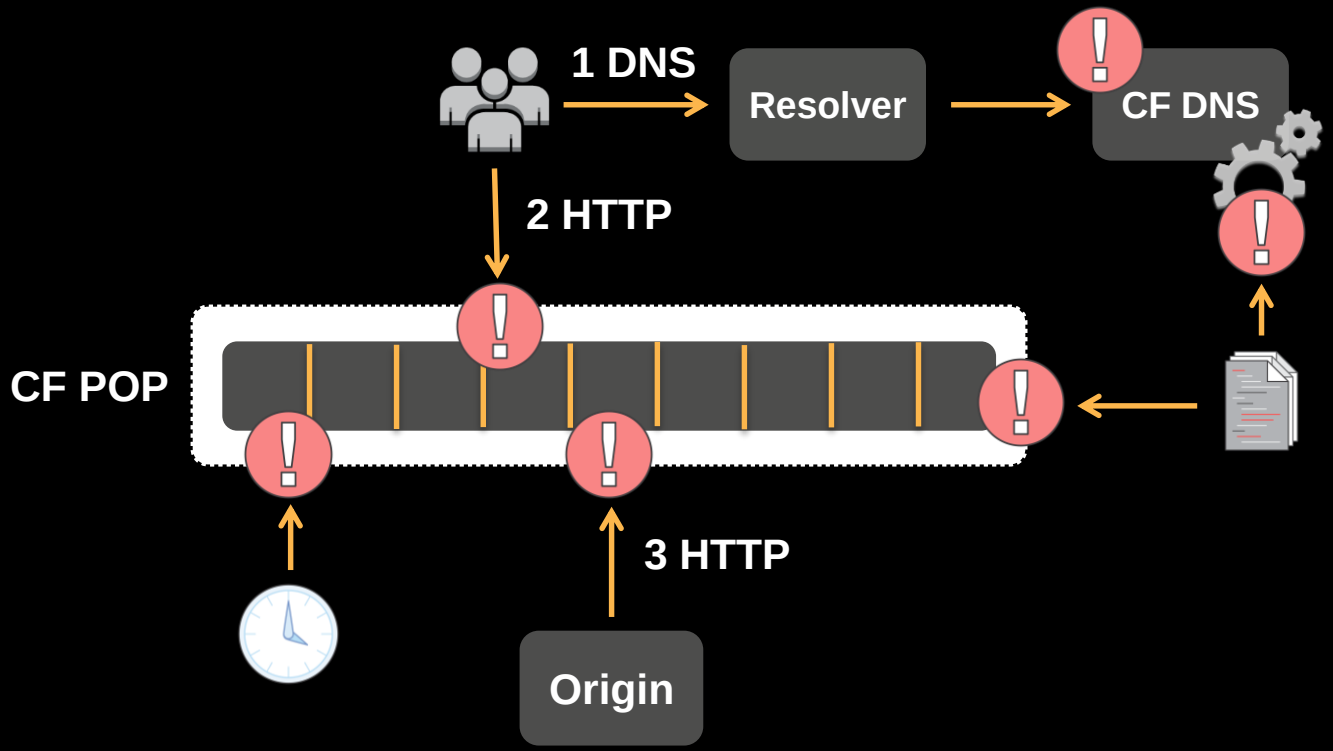
Potential causes of application failure



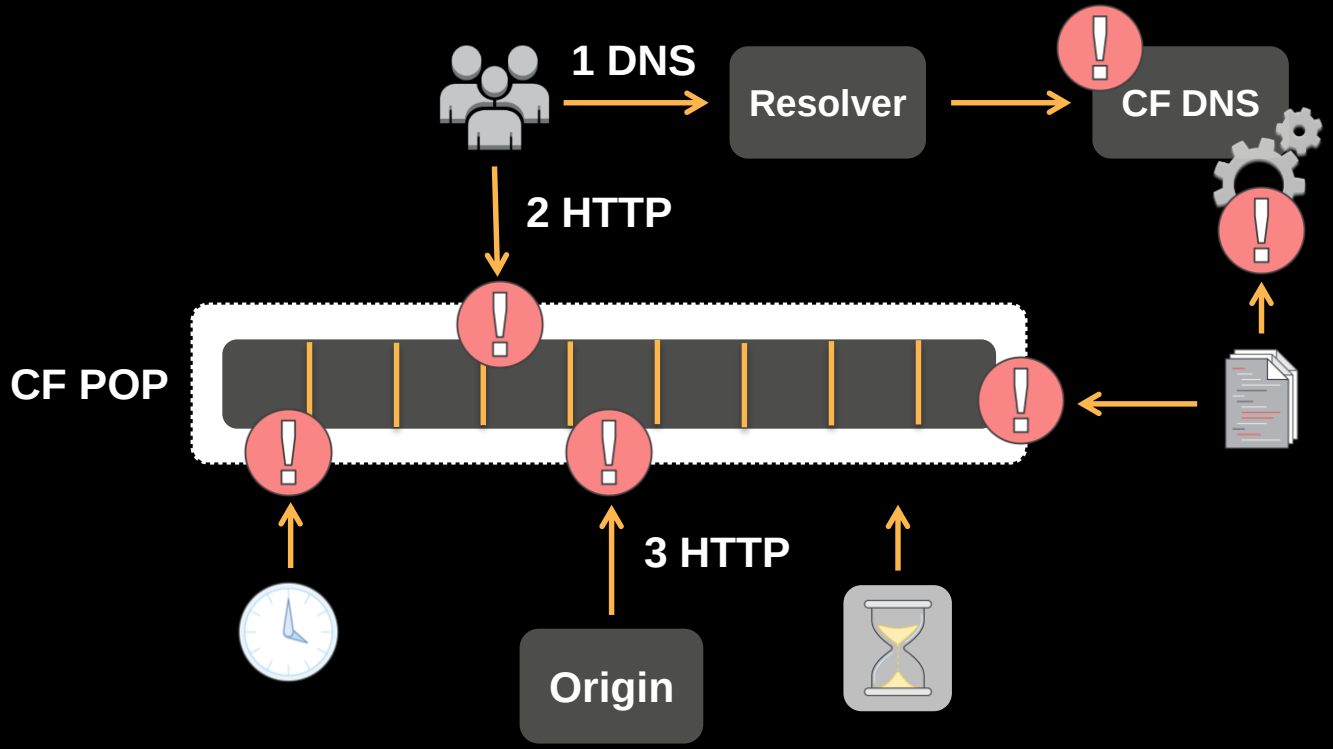
Potential causes of application failure



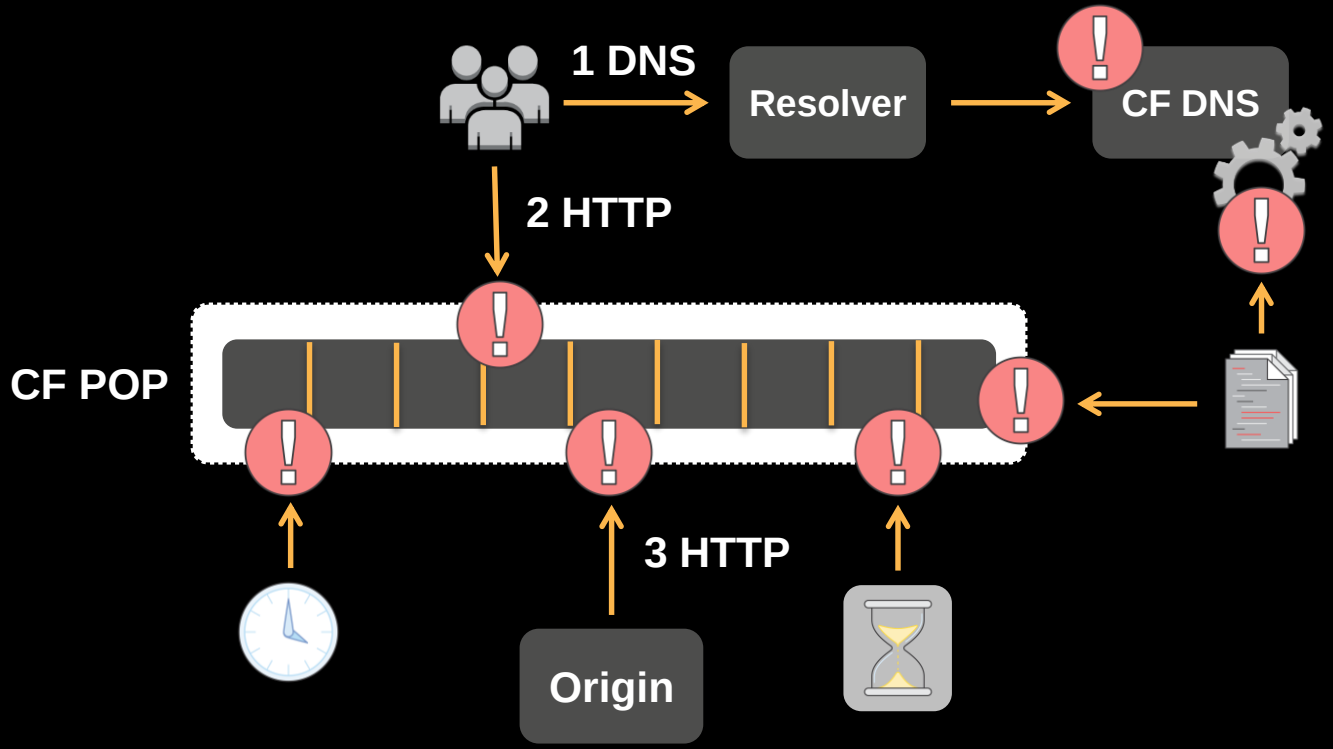
Potential causes of application failure



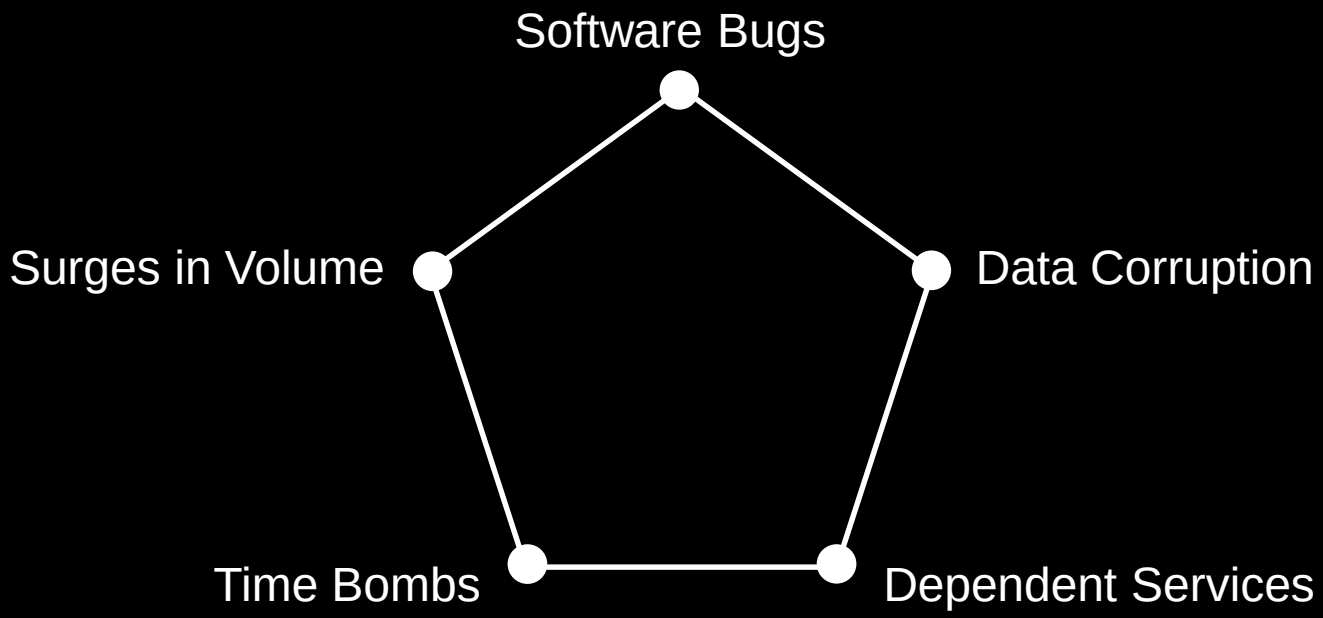
Potential causes of application failure



Potential causes of application failure



Risks to availability



Rapid Iteration on Capabilities

2008

- CloudFront Launched with 14 POPs

2009

- Access Logs
- Lower Pricing Tiers
- Management Console
- Private Content
- RTMP Streaming

2010

- 1-Hour TTLs
- Custom Origins
- Default Root Object
- HTTPS Support
- Invalidations
- Price Drop, SLA
- Private Streaming
- RTMP Access Logs
- Singapore
- New York City
- Jacksonville

2011

- File Size Increase
- IAM Integration
- Live Streaming
- Price Drop
- Paris
- Stockholm
- Sao Paulo
- San Jose
- South Bend
- Los Angeles (2nd)
- New York City (2nd)

2012

- Geo-Blocking
- Live Streaming Update
- Multiple Cache Behaviors
- Multiple Origins
- QueryString Forwarding
- Zero TTL Support
- Osaka
- Milan
- Virginia (2nd)
- Singapore (2nd)
- Frankfurt (2nd)
- London (2nd)
- Dallas (2nd)

2013

- Cookie Based Caching
- Custom SSL Support
- Enhanced Logs
- HTTP 1.1
- Lower Inter-Region Pricing
- Price Classes
- Private Content Console Support
- Zone Apex Support
- Chennai & Mumbai
- Seoul
- Hayward
- Madrid
- Sydney
- Paris (2nd)
- Amsterdam (2nd)
- Tokyo (2nd)
- Hong Kong (2nd)
- New York City (3rd)
- Virginia (3rd)

2014

- Smooth Streaming
- SNI Custom SSL
- HTTP to HTTPS Redirect
- Usage Charts
- Free Tier
- EDNS Client Subnet
- CloudTrail
- Device Detection, Geo Targeting, Host Header Forwarding
- Advanced SSL Features
- Wildcard Cookies
- Monitoring & Alarming
- Content Charts
- Price Reduction
- Zero Rate AWS Origin Traffic
- Directory Path as Origin
- Viewer Charts
- Rio de Janeiro
- Taipei
- Melbourne

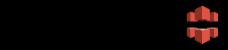
2015

- Signed Cookie Support
- Smart TV Device Detection
- Devices Report
- Usage Metrics CSV Export
- Wildcard Invalidations
- Default TTLs
- Max TTLs
- PCI DSS Compliance
- AWS WAF Integration
- Auto Gzip Compression
- Modify Request Headers
- Chicago
- Seoul (2nd)

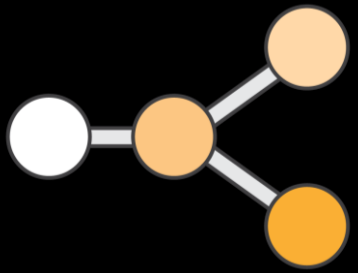
2016

- Enforce HTTPS Connections
- TLSv1.1 and TLSv1.2 to Origin
- AWS Certificate Manager Integration
- Cost Allocation Tagging
- Query String Whitelisting
- HTTP/2
- Toronto
- Montreal
- New Delhi
- Atlanta (2nd)
- Mumbai (2nd)
- Seoul (3rd)
- Frankfurt (2nd)
- Japan (4th)
- Hong Kong (3rd)
- London (4th)
- Berlin
- Minneapolis
- IPv6
- ACM Cert Import
- Regional Edge Caches
- […]

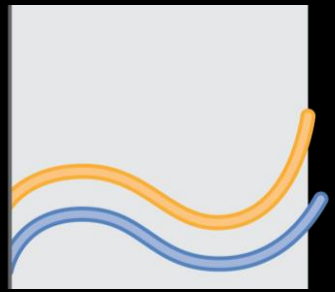
Design patterns for availability



Design patterns for availability



Maximizing availability with Food Tasting

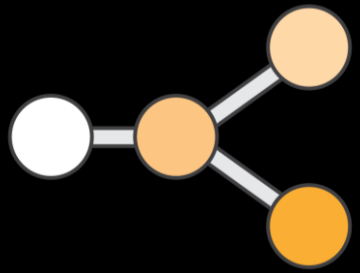


Flash Crowds without scaling for the peak

Defense in Depth Strategies

Time Bomb Jitter Protection

Design pattern 1: FoodTaster



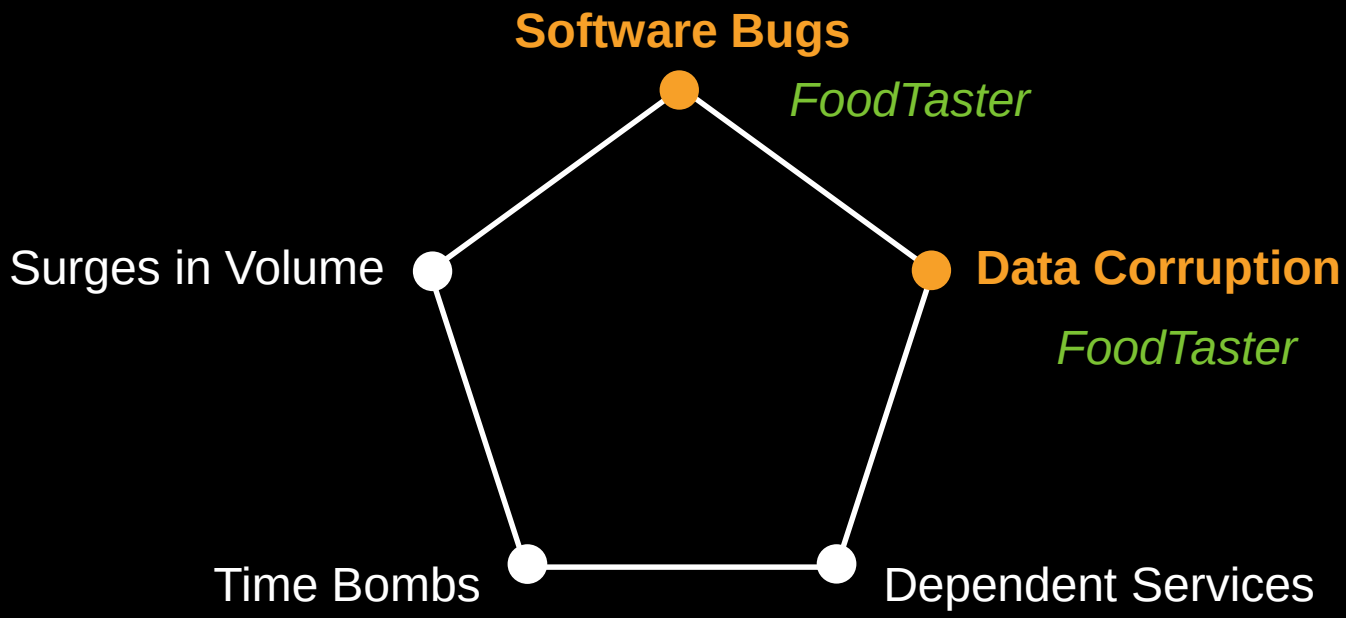
Maximizing availability with Food Tasting

Flash Crowds without scaling for the peak

Defense in Depth Strategies

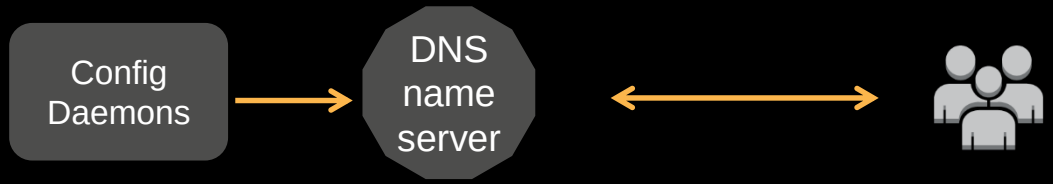
Time Bomb Jitter Protection

Risks to availability

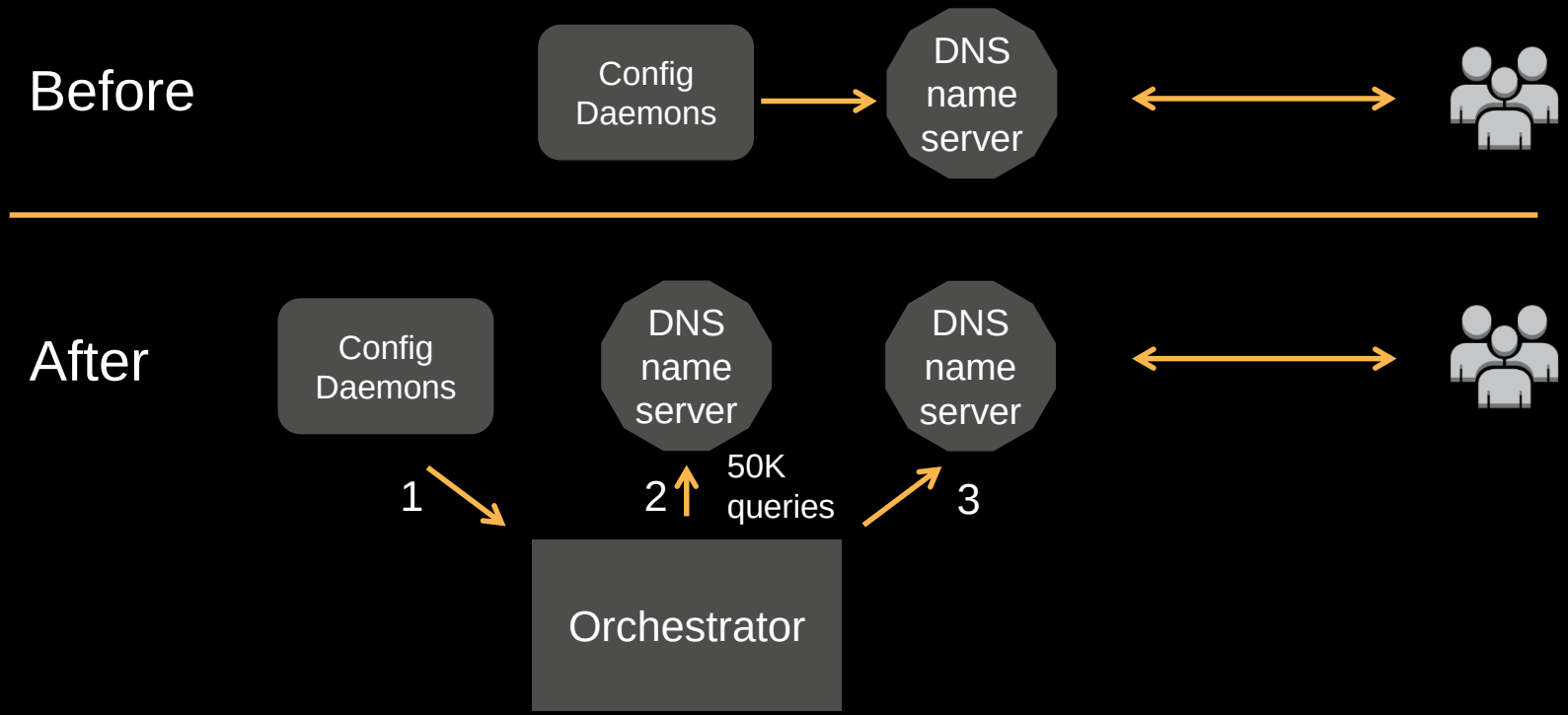


Praegustator – DNS FoodTaster

Before



Praegustator – DNS FoodTaster



Praegustator – DNS FoodTaster

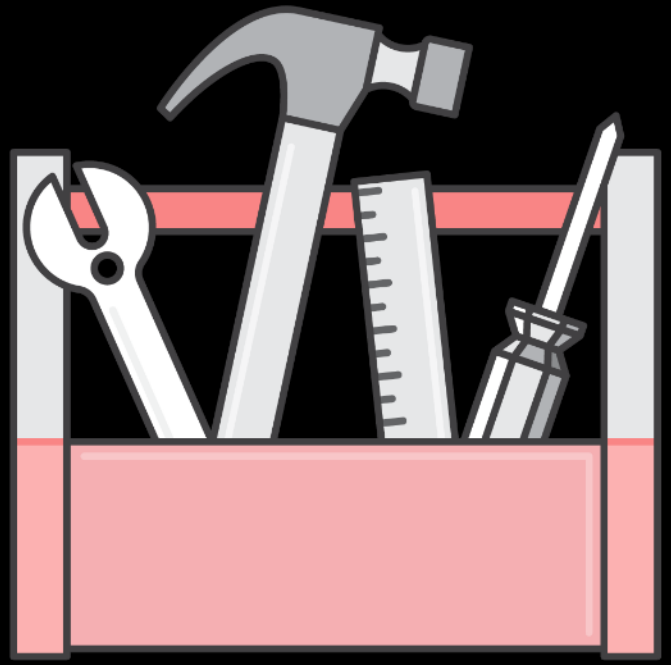
```
$ ls
dns/ foodtaster/ landing/

$ ls foodtaster/
customer.data@      pop.data@
resolvers.data@     routing.data

$ ls dns/
customer.data       pop.data
resolvers.data      routing.data
```

Food Tasting in AWS

- Straightforward approach
- Deployments are inexpensive, redeployments more too
- Approaches like CodeDeploy make life even easier



Food Tasting in AWS

- Example - **GeoIP Database**
- Automatically pushed out to every host
- Likely already have checks
- More to consider than just invalid user configuration

Food Tasting in AWS

- Simple is better (Works for Amazon CloudFront!)
- Assume we already use a deployment system (e.g., AWS CodeDeploy)
- Complete in-situ tests before returning complete

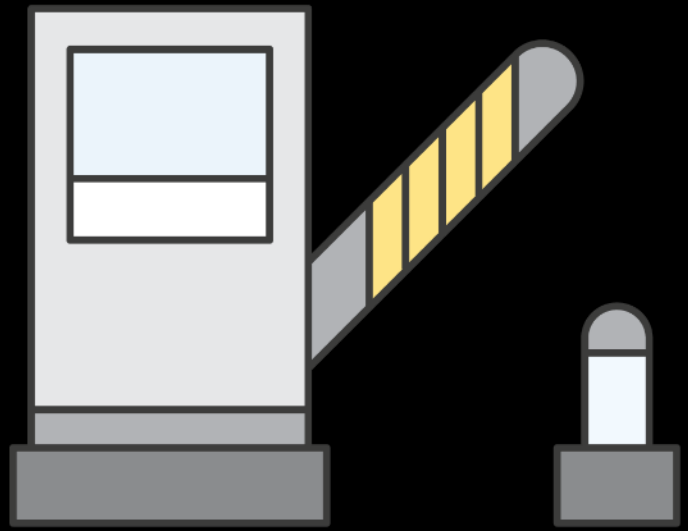
CodeDeploy AppSec example

```
hooks:
    BeforeInstall:
        - location: Scripts/UnzipResourceBundle.sh
        - location: Scripts/UnzipDataBundle.sh
    AfterInstall:
        - location: Scripts/RunResourceTests.sh
          timeout: 180
    ApplicationStart:
        - location: Scripts/RunFunctionalTests.sh
          timeout: 3600
    ValidateService:
        - location: Scripts/MonitorService.sh
          timeout: 3600
          runas: codedeployuse
```

CodeDeploy AppSec example

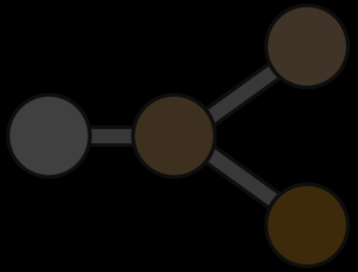
```
hooks:
    BeforeInstall:
        - location: Scripts/UnzipResourceBundle.sh
        - location: Scripts/UnzipDataBundle.sh
    AfterInstall:
        - location: Scripts/RunResourceTests.sh
          timeout: 180
    ApplicationStart:
        - location: Scripts/RunFunctionalTests.sh
          timeout: 3600
    ValidateService:
        - location: Scripts/MonitorService.sh
          timeout: 3600
          runas: codedeployuse
```

Food Tasting in AWS

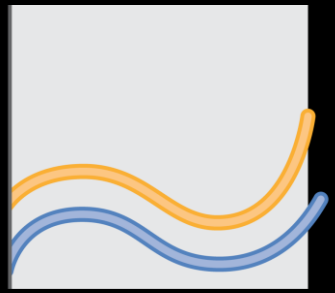


- Acts as a quality gate
- Roll-back can be automatic
- Verification never affects user facing traffic

Design pattern 2: Flash Crowds



Maximizing availability with Food Tasting

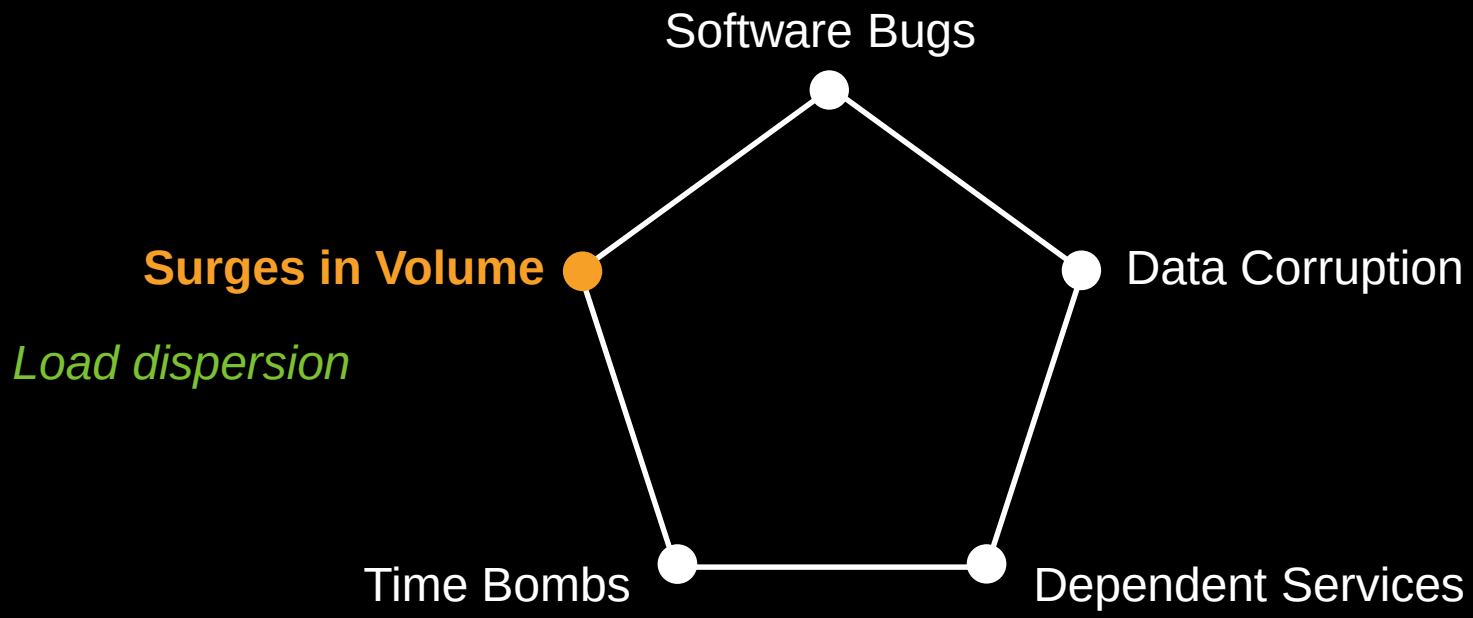


Flash Crowds without scaling for the peak

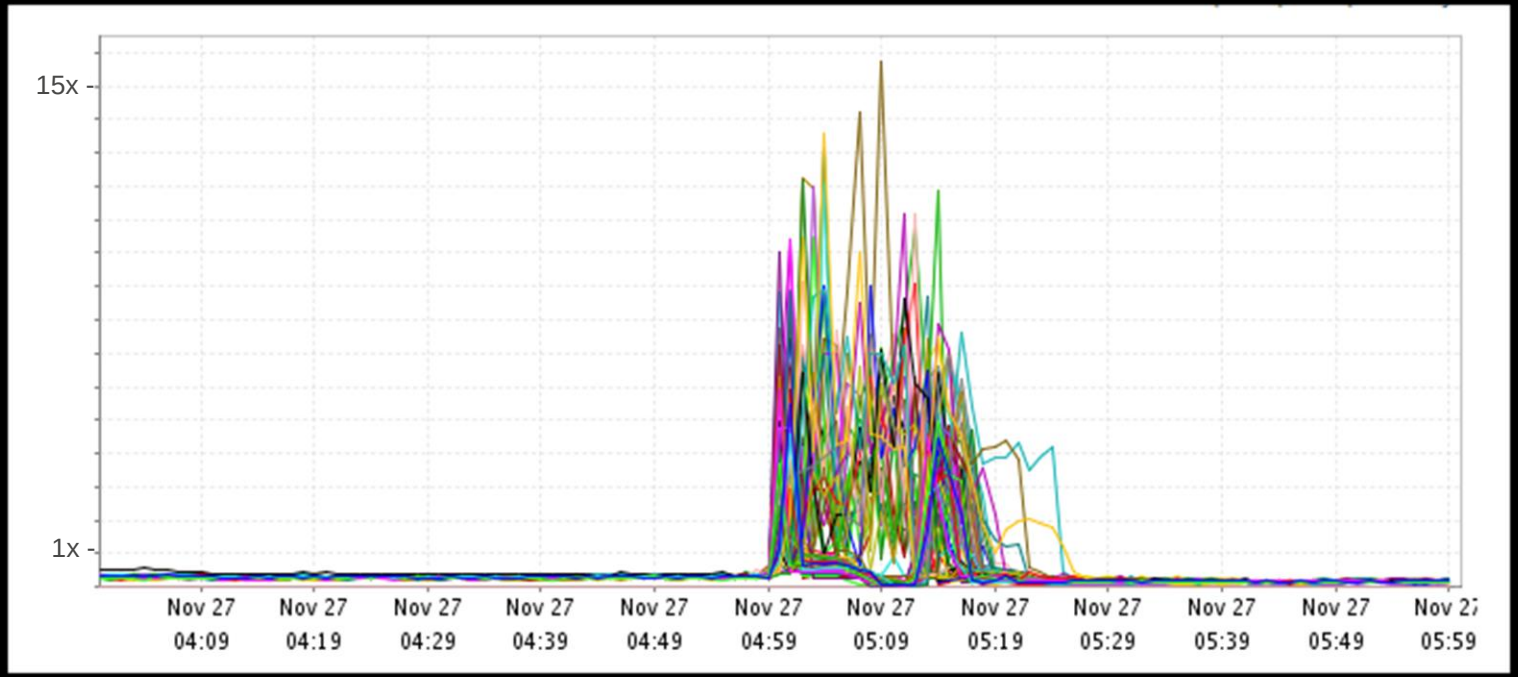
Defense in Depth Strategies

Time Bomb Jitter Protection

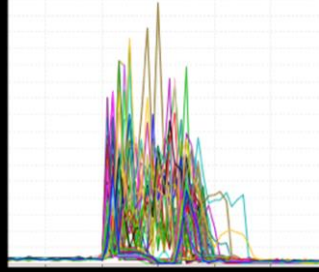
Risks to availability



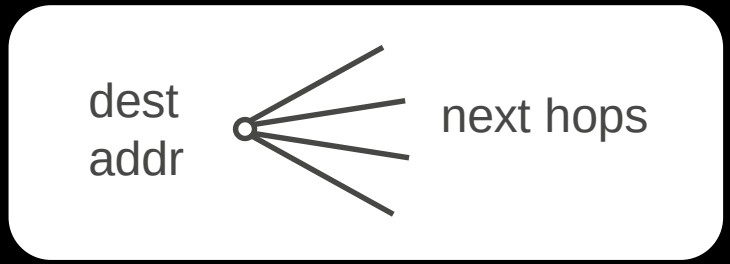
HTTP Flash Crowds



Static & Dynamic Strategies

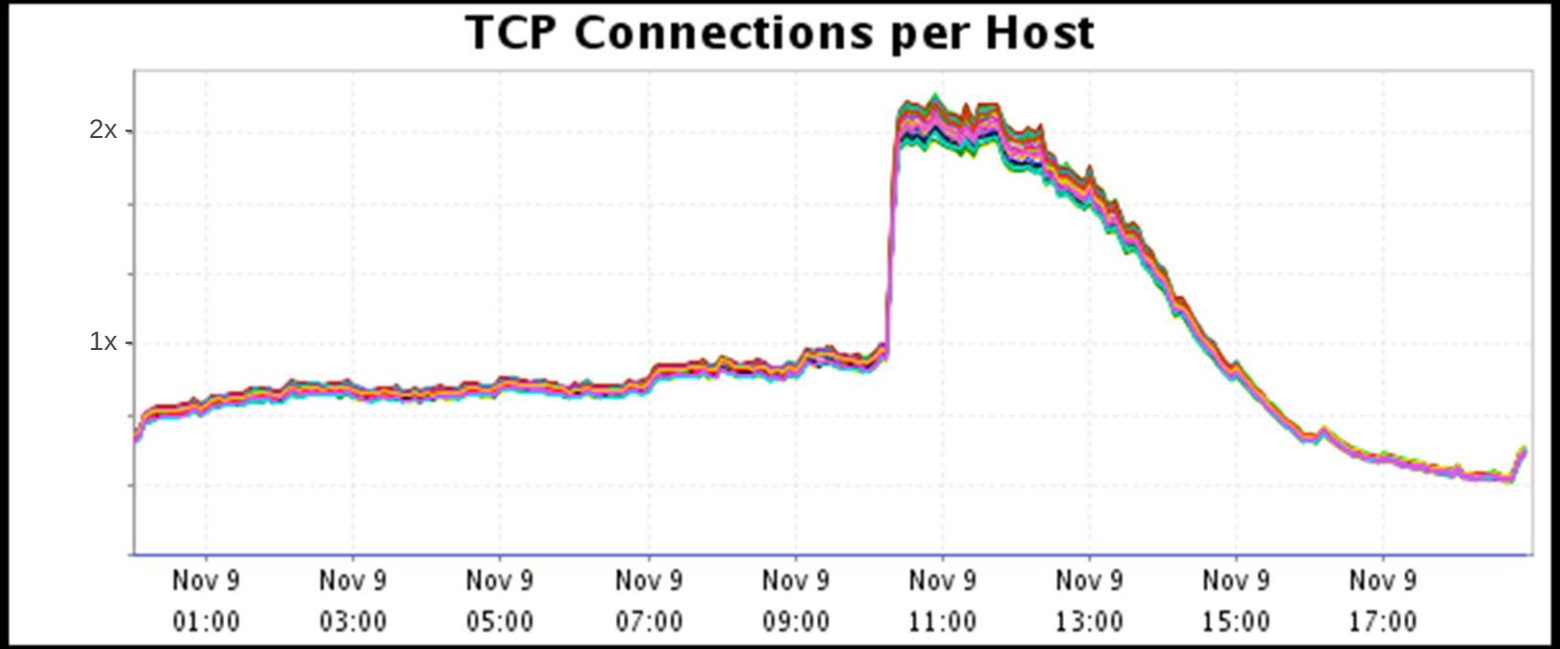


Packet-flow config



Feedback loop

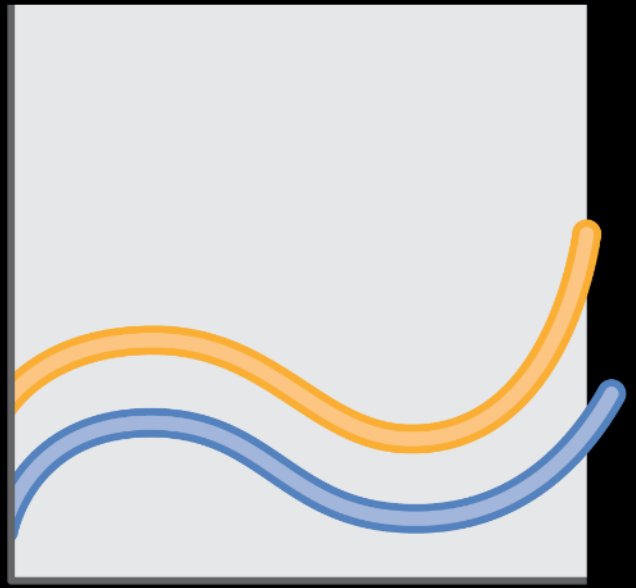
Load Dispersal

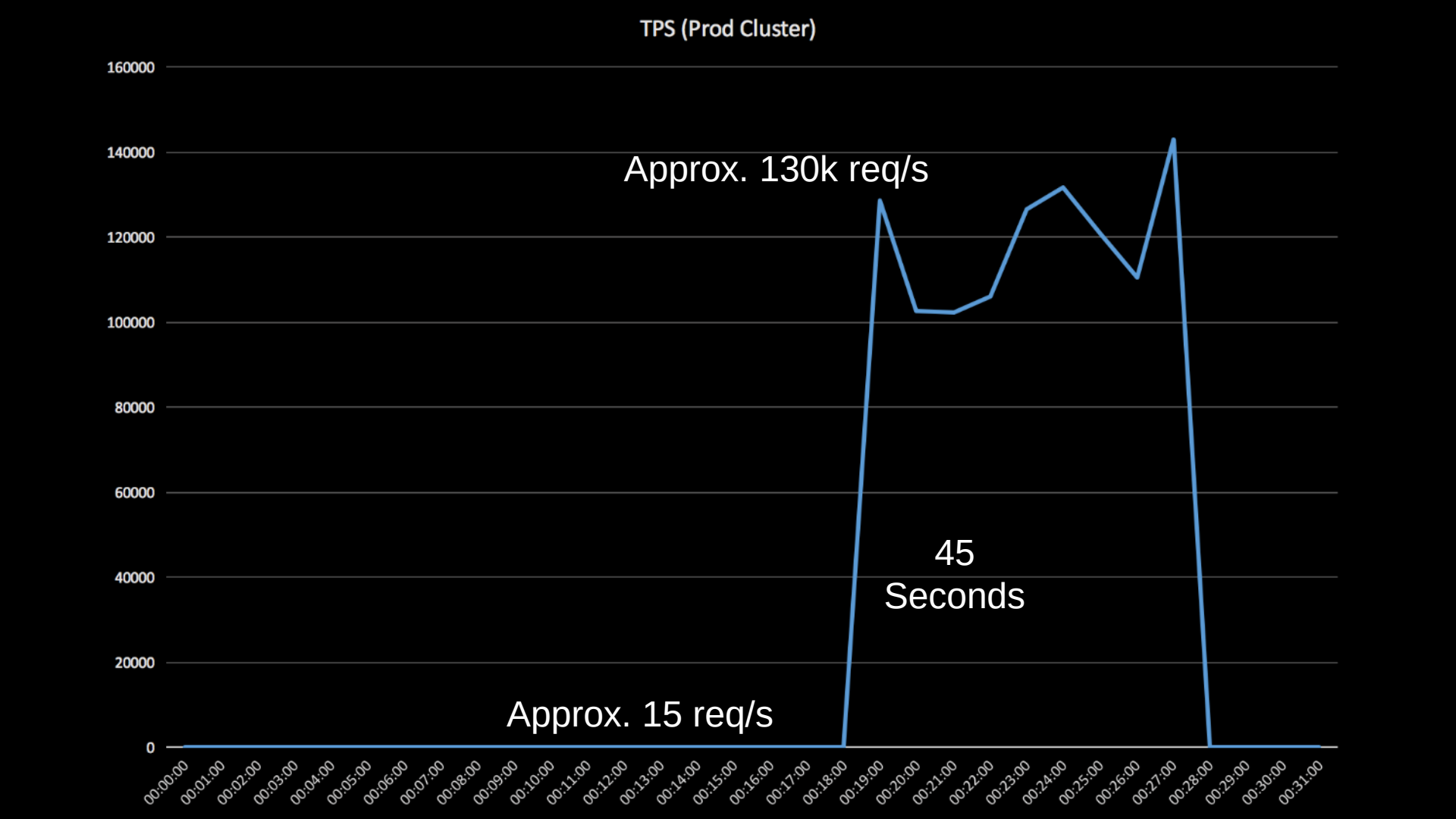


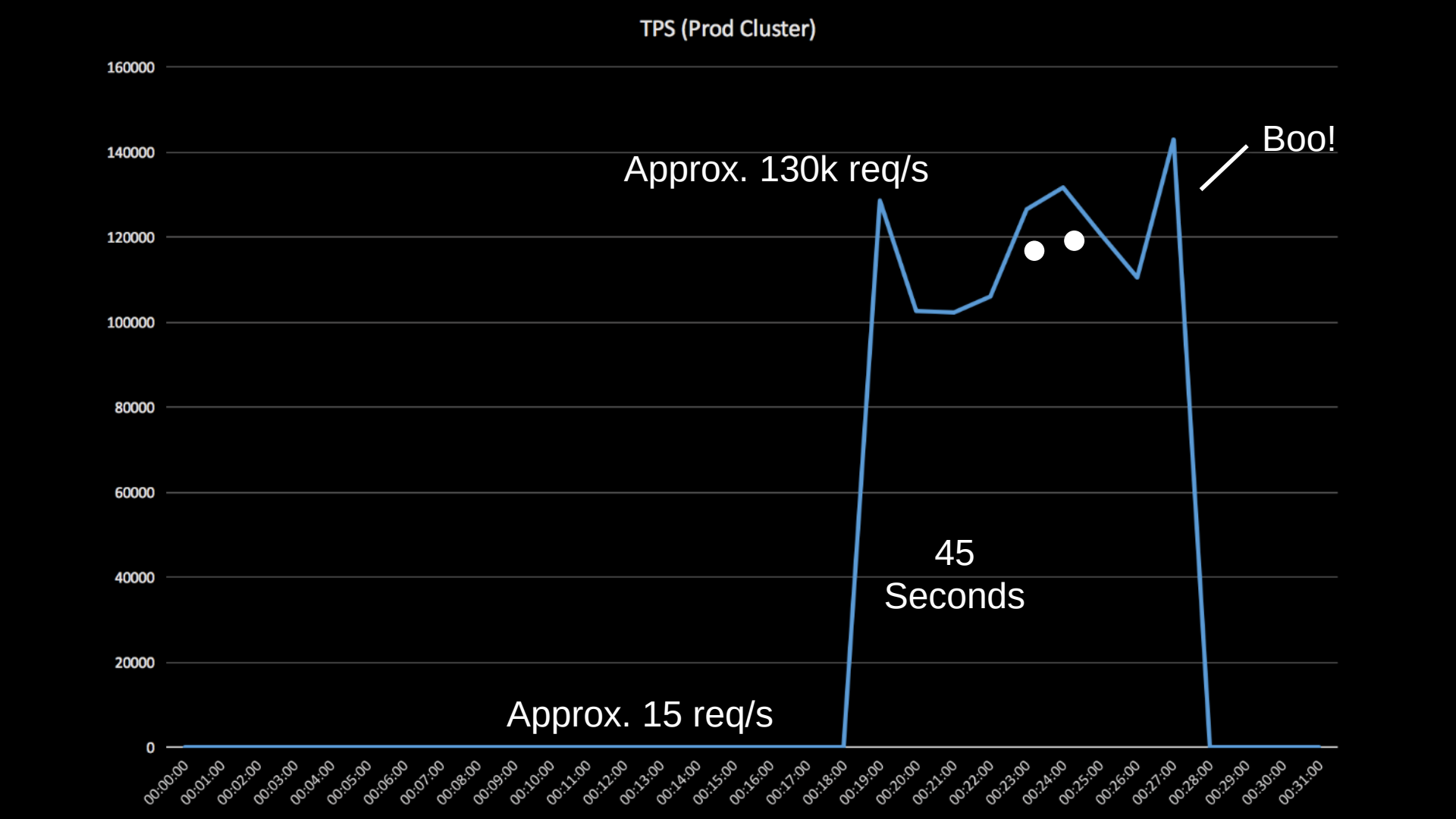
Flash Crowds within AWS

Auto Scaling works really well

But.




TPS (Prod Cluster)
160000
140000
120000
100000
80000
60000
40000
20000
0
Approx. 130k req/s
45
Seconds
Approx. 15 req/s
00:00:00
00:01:00
00:02:00
00:03:00
00:04:00
00:05:00
00:06:00
00:07:00
00:08:00
00:09:00
00:10:00
00:11:00
00:12:00
00:13:00
00:14:00
00:15:00
00:16:00
00:17:00
00:18:00
00:19:00
00:20:00
00:21:00
00:22:00
00:23:00
00:24:00
00:25:00
00:26:00
00:27:00
00:28:00
00:29:00
00:30:00
00:31:00

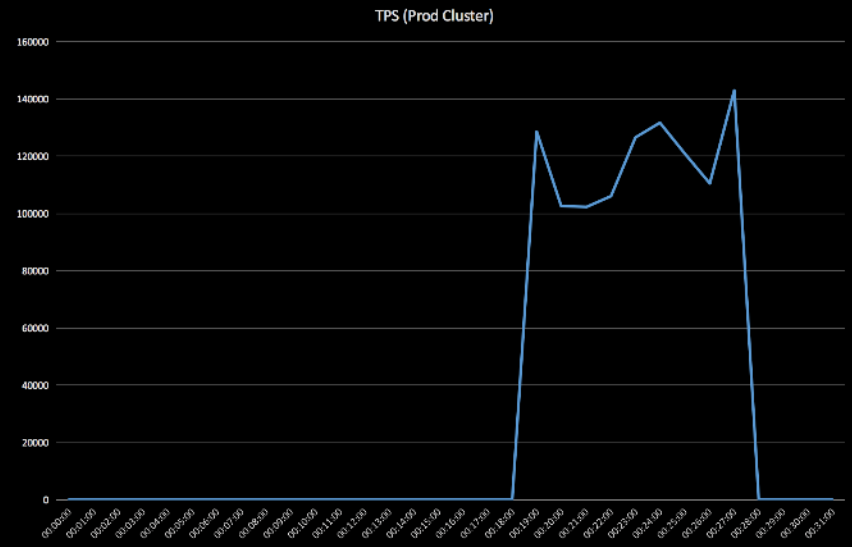

TPS (Prod Cluster)
160000
140000
120000
100000
80000
60000
40000
20000
0
Approx. 130k req/s
Boo!
45
Seconds
Approx. 15 req/s
00:00:00
00:01:00
00:02:00
00:03:00
00:04:00
00:05:00
00:06:00
00:07:00
00:08:00
00:09:00
00:10:00
00:11:00
00:12:00
00:13:00
00:14:00
00:15:00
00:16:00
00:17:00
00:18:00
00:19:00
00:20:00
00:21:00
00:22:00
00:23:00
00:24:00
00:25:00
00:26:00
00:27:00
00:28:00
00:29:00
00:30:00
00:31:00

Problem: Flash Crowds

Auto Scaling works really well

But.

Sometimes, 60s is too long.



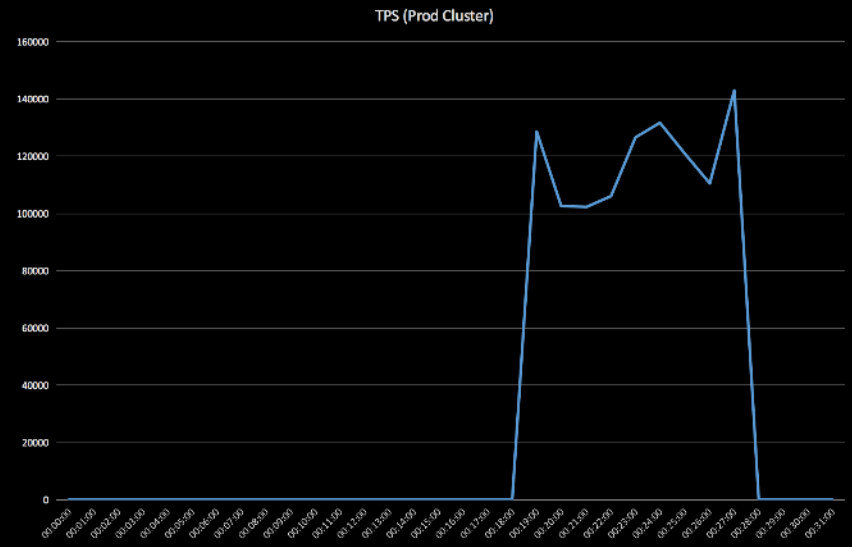
AWS solutions

- **Plan Ahead**

Planning ahead

Typical examples:

- TV programs
- Live events (sports)
- Game releases
- Established traffic patterns



Planning ahead

Typical examples:

- TV programs
- Live events (sports)
- Game releases
- Established traffic patterns

AWS Solutions:


Infrastructure Event Management


Scheduled Auto Scaling Group


Auto Scaling Integration

Schedules and the Big Red Button

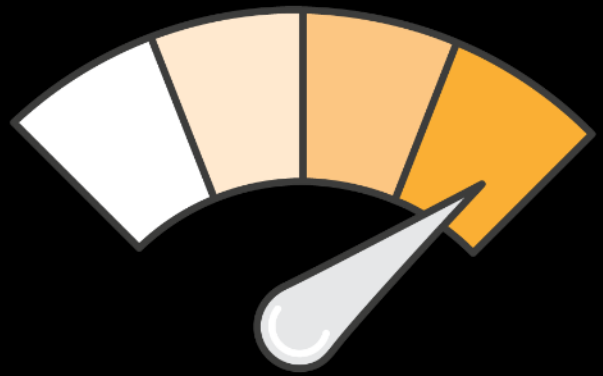
- Integration of **program scheduling** and Auto Scaling
- Program scheduling includes expected online audience parameters
- Big red button

Schedules and the Big Red Button

- Integration of program scheduling and Auto Scaling
- Program scheduling includes expected online audience parameters
- Big red button

AWS solutions

- Plan Ahead
- **Cache Things**



Cache things

Cache things

I can’t!

My website is…

Cache things

I can’t!

My website is…

DYNAMIC!

Cache things

- What's really dynamic?
  - Newspapers
  - Voting sites
  - Forum sites

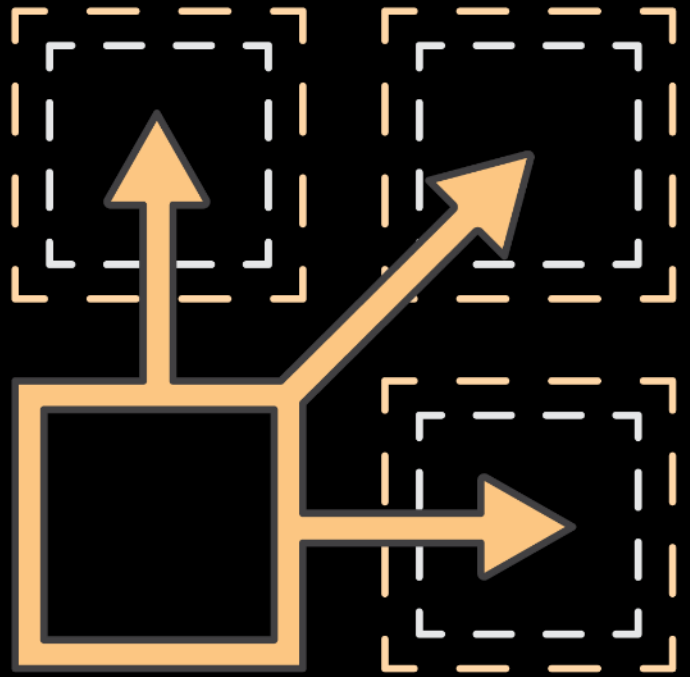
- Updates don't need to be more than once a second

Cache things (even for a second)

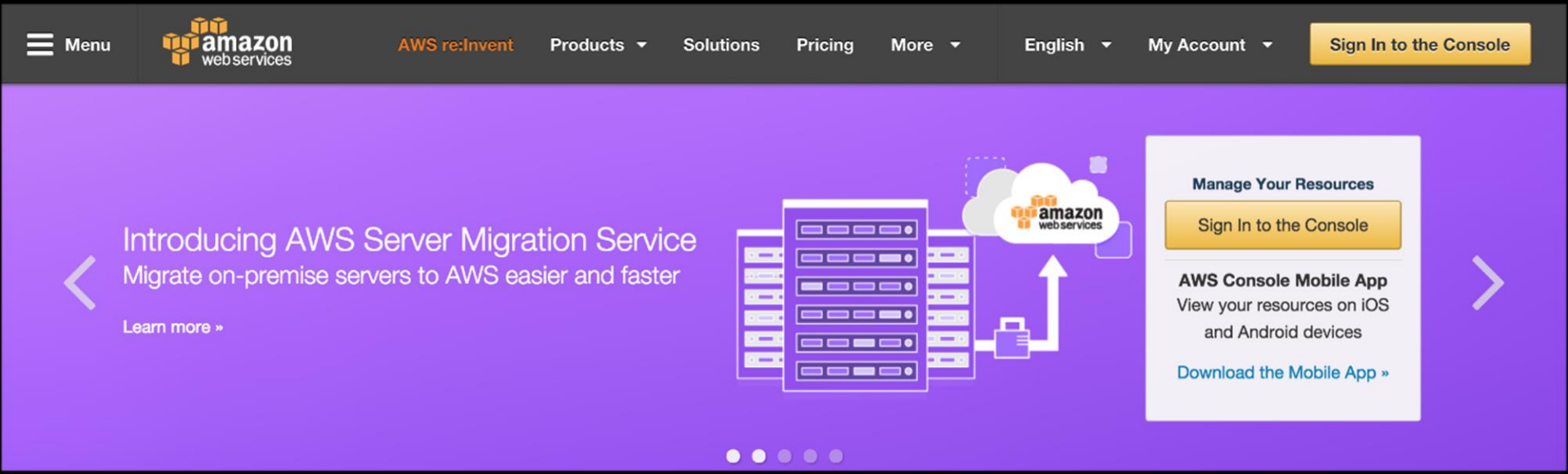
- 10000s req/s
- Assuming every Amazon CloudFront POP
- 10000s req/s -> ~ 10s req/s

AWS solutions

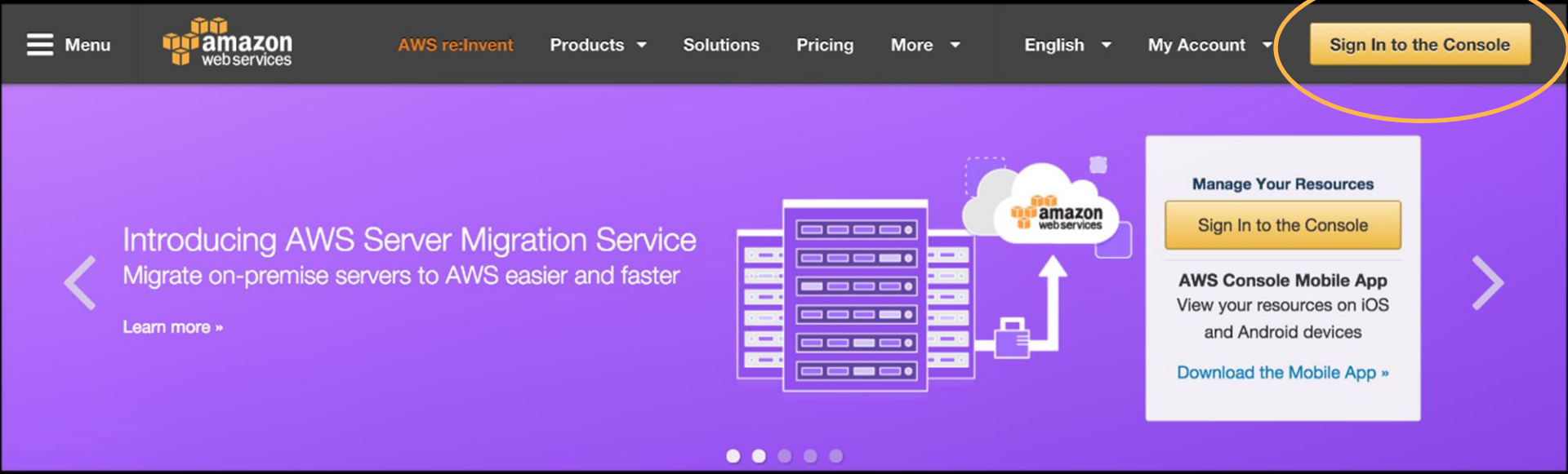
- Plan Ahead
- Cache Things
- **Serve only what you have to**



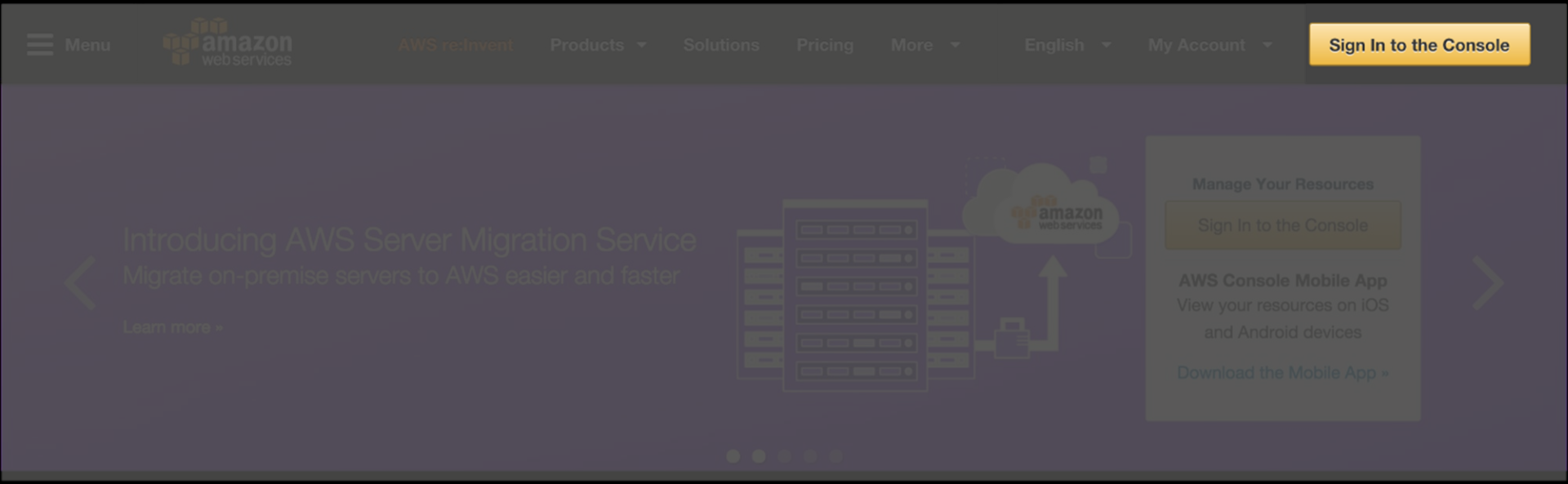
Serve only what you have to



Serve only what you have to

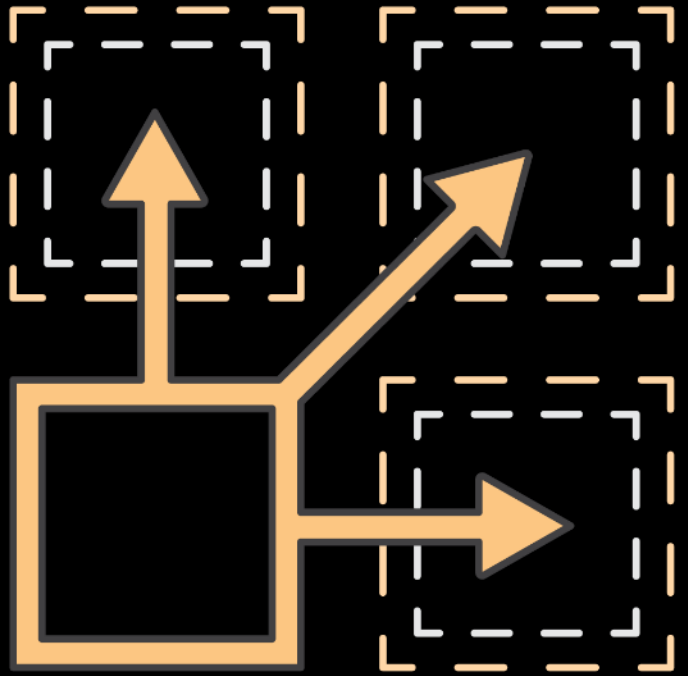


Serve only what you have to



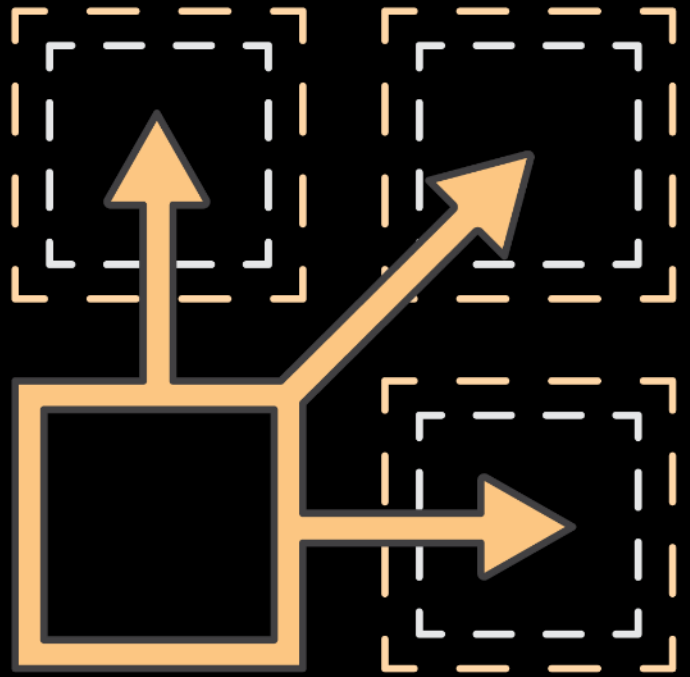
Serve only what you have to

- AJAX Includes

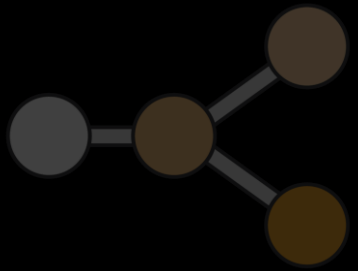


Serve only what you have to

- AJAX Includes
- CloudFront Cache Keys



Design pattern 3: Defense in Depth



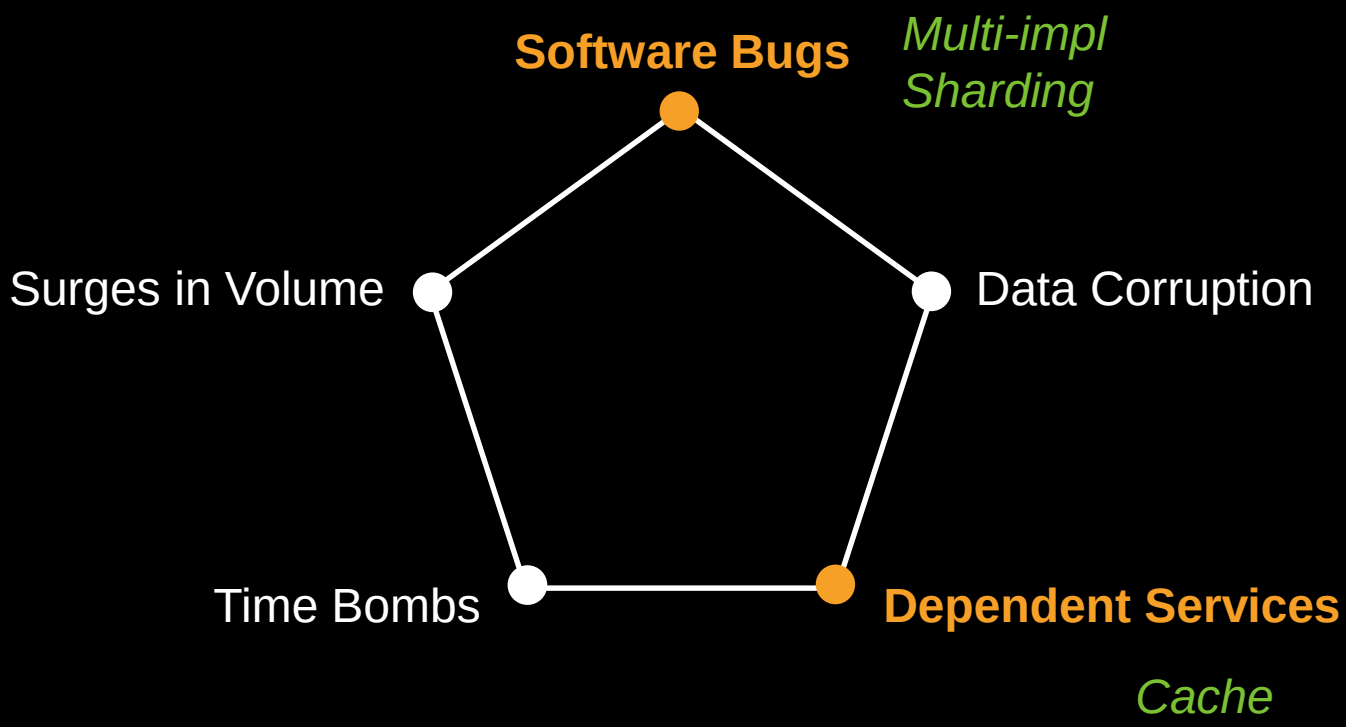
Maximizing availability with Food Tasting

Flash Crowds without scaling for the peak

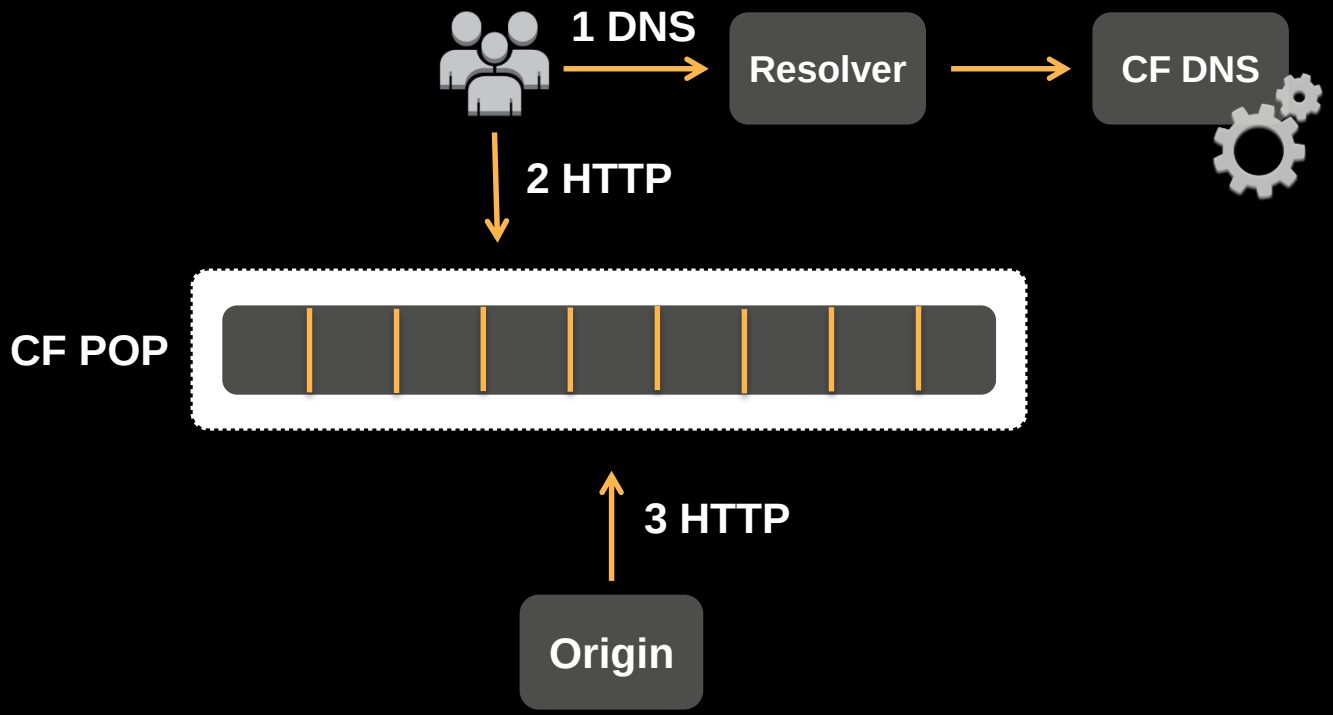
Defense in Depth Strategies

Time Bomb Jitter Protection

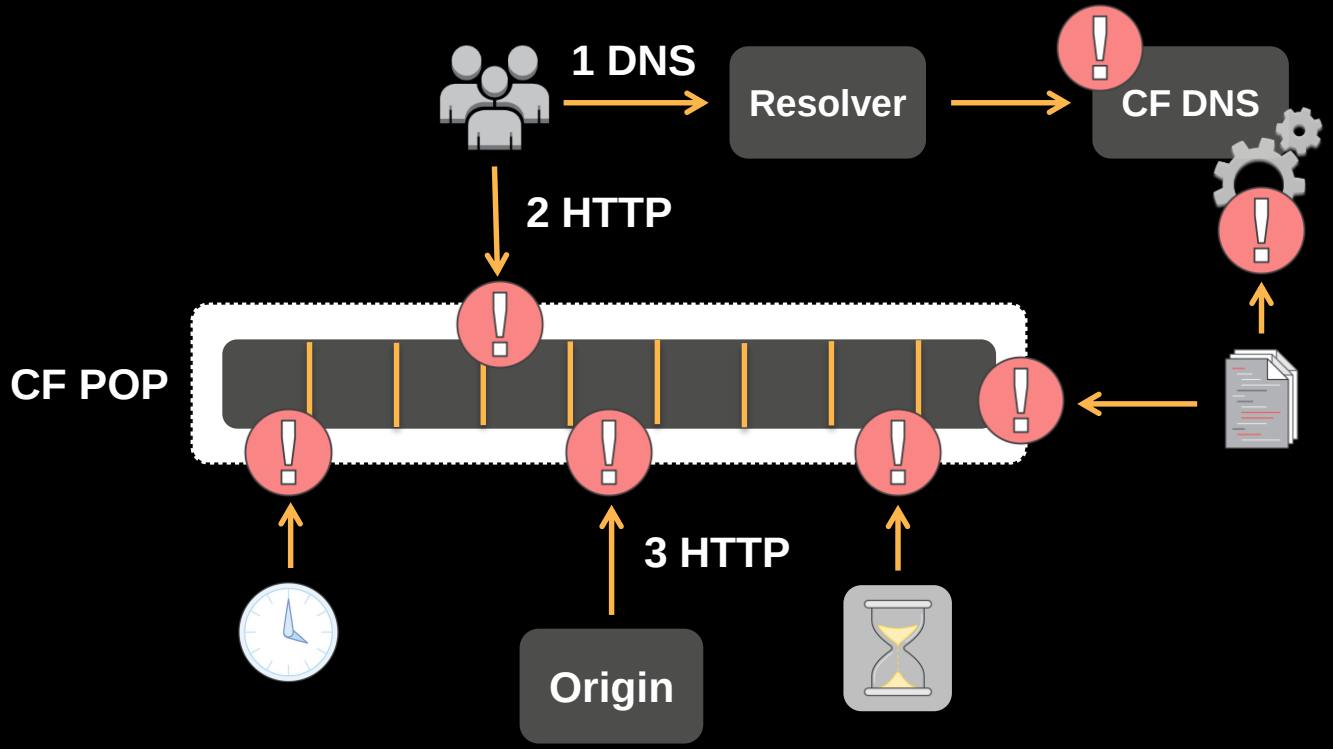
Risks to availability



Potential causes of application failure



Potential causes of application failure



Ideas to avoid crashing

- Comprehensive test coverage
- Simplify systems

Failures are going to happen.

How can we survive them when they do?

Ideas to survive crashing

- Reduce blast radius

- Reject input that previously made you crash

Ideas to survive crashing

- Reduce blast radius
    - shard customers to separate processes

- Reject input that previously made you crash

Ideas to survive crashing

- Reduce blast radius
  - shard customers to separate processes
  - recover quickly

- Reject input that previously made you crash

Ideas to survive crashing

- Reduce blast radius
  - shard customers to separate processes
  - recover quickly
  - multiple implementations
- Reject input that previously made you crash

DNS Multi-Impl

Should this be a fallback system?

Place it in front of every DNS name server?

DNS Multi-Impl

Should this be a fallback system?

- We want to know it always works

Place it in front of every DNS name server?

DNS Multi-Impl

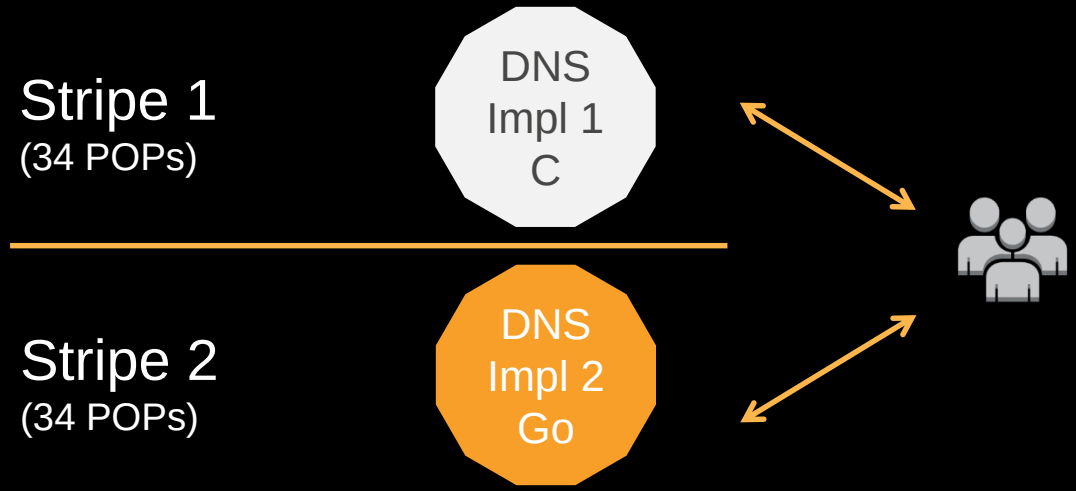
Should this be a fallback system?

- We want to know it always works

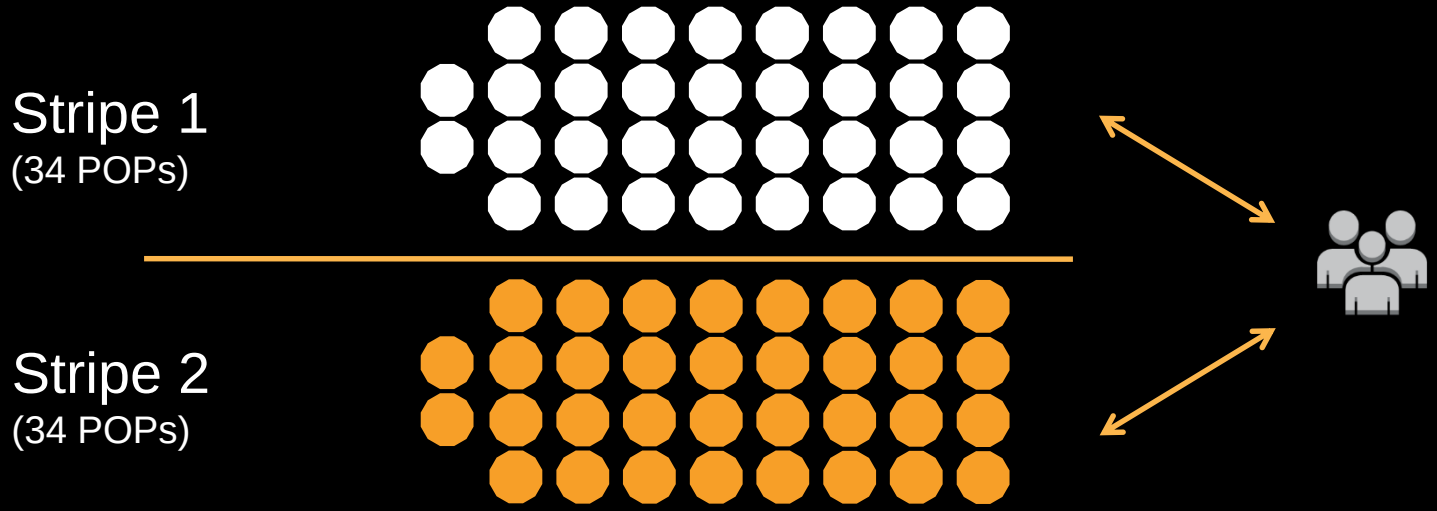
Place it in front of every DNS name server?

- Would trade point of failure one for another

DNS Stripes



DNS Stripes



CloudFront case study

Math.abs() on the lowest 32-bit signed integer, -2^31, yields a negative number.
Leading to invalid DNS configuration.

```
Two's complement: flip bits and add 1.

Math.abs(Integer.MIN_VALUE):
       -2^31 = 10000000 00000000 00000000 00000000
bits flipped = 01111111 11111111 11111111 11111111
          +1 = 10000000 00000000 00000000 00000000  <= -2^31!
```

CloudFront case study

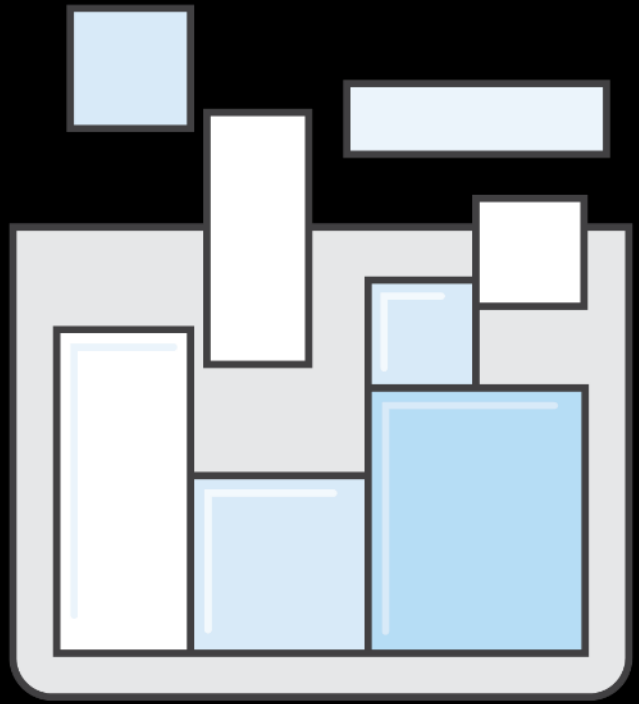
Protections in place:

1. Config process crashing in a loop
2. DNS name server defensively ignores an invalid index
3. FoodTaster crashes protect King DNS name server
4. DNS stripes reduce common failure paths

AWS solution

We can use a Load Balancer with Layer-7 awareness

So we can have multiple backend types across a fleet

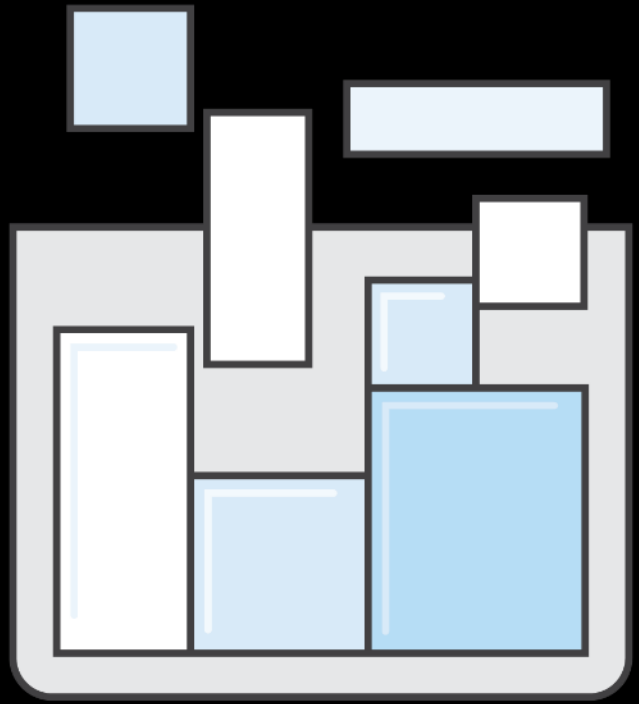


AWS solution

Enabled through microservices

Reimplementing the entire stack multiple times is a difficult cost / efficiency question

Specific high-risk services are easier



AWS solution

Example: **2FA Platform**

API (HTTP) front-end to a back-end authentication platform

Reasonably few SLOC

If it fails – no access to platform

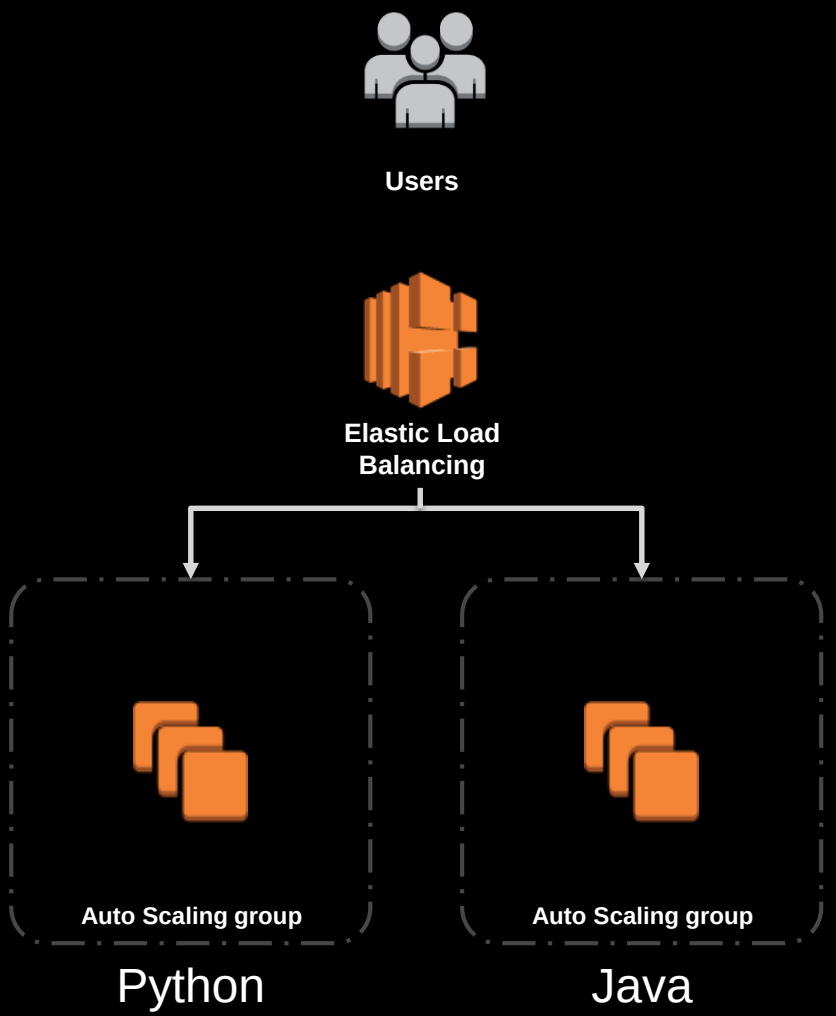
AWS solution

Example: **2FA Platform**

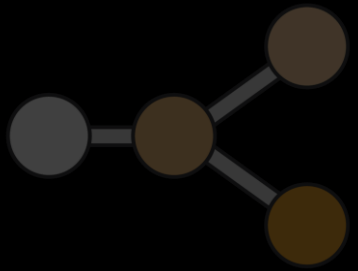
API (HTTP) front-end to a back-end authentication platform

Reasonably few SLOC

If it fails – no access to platform



Design Pattern 4: Time Bomb Jitter Protection



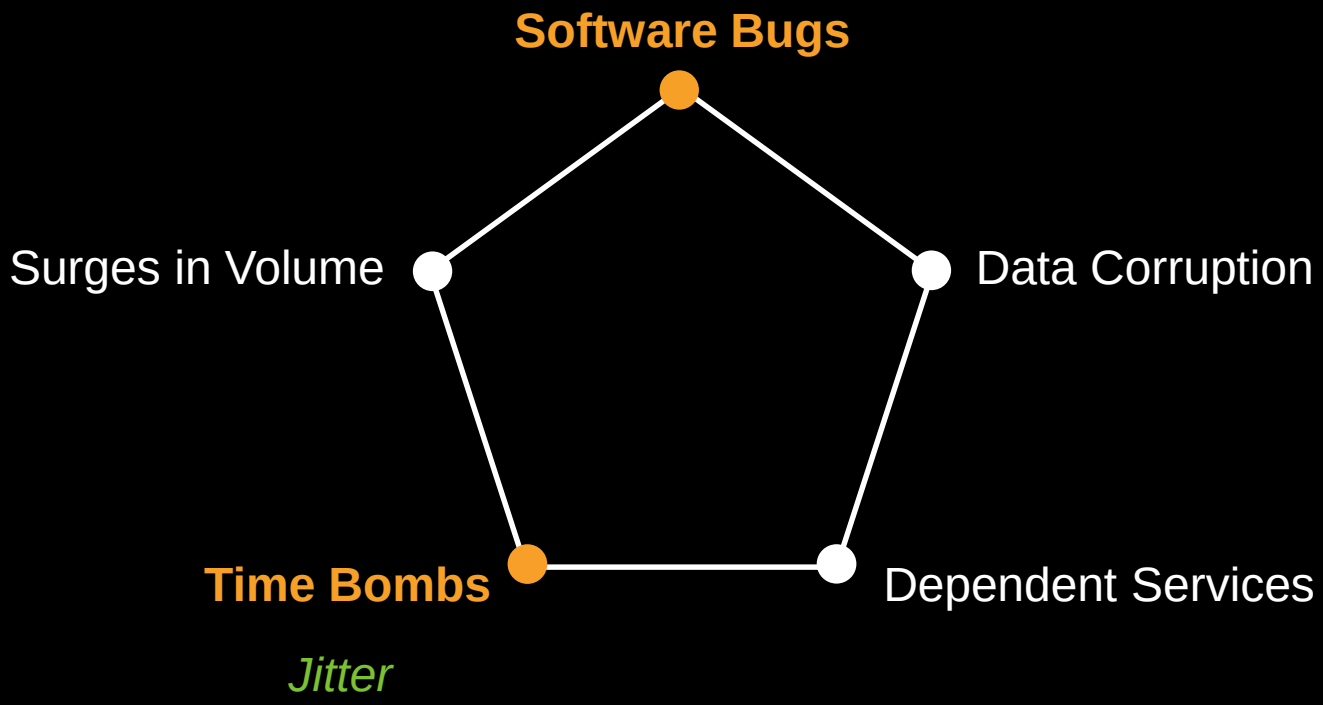
Maximizing availability with Food Tasting

Flash Crowds without scaling for the peak

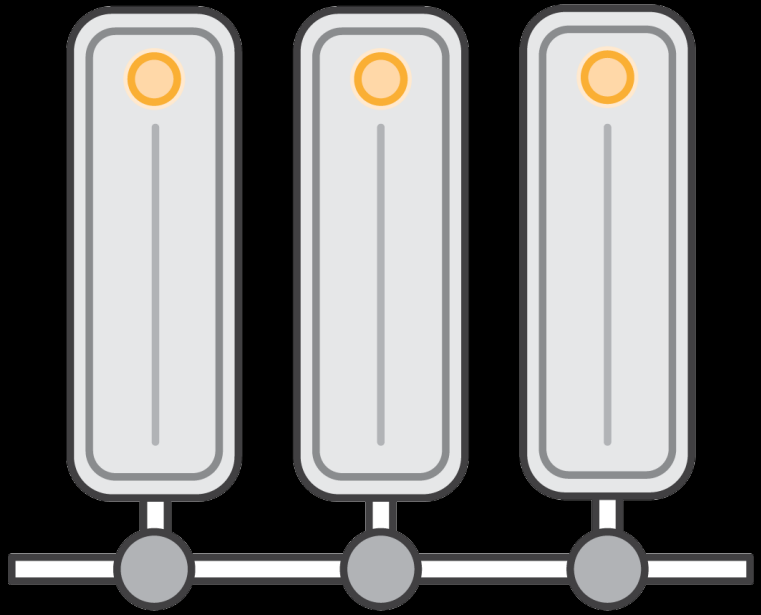
Defense in Depth Strategies

Time Bomb Jitter Protection

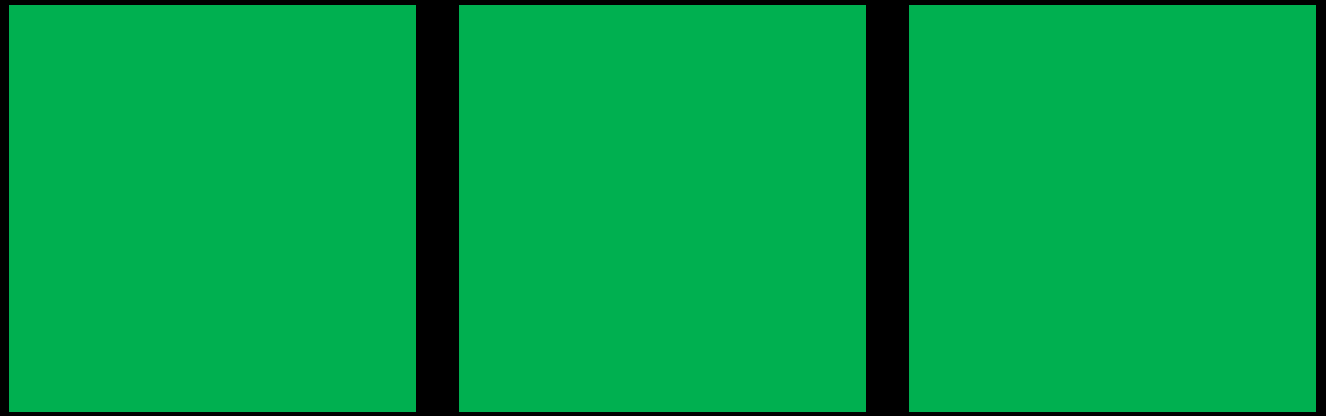
Risks to availability



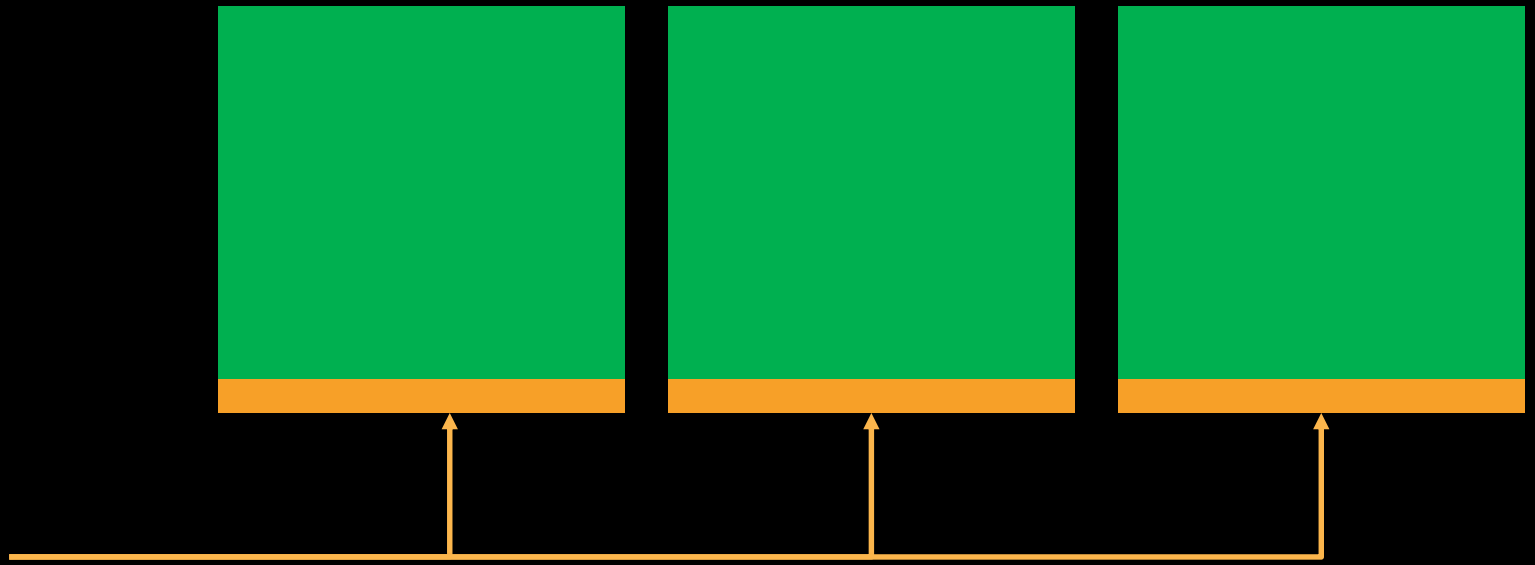
Problem: homogeneous platforms



The problem



The problem

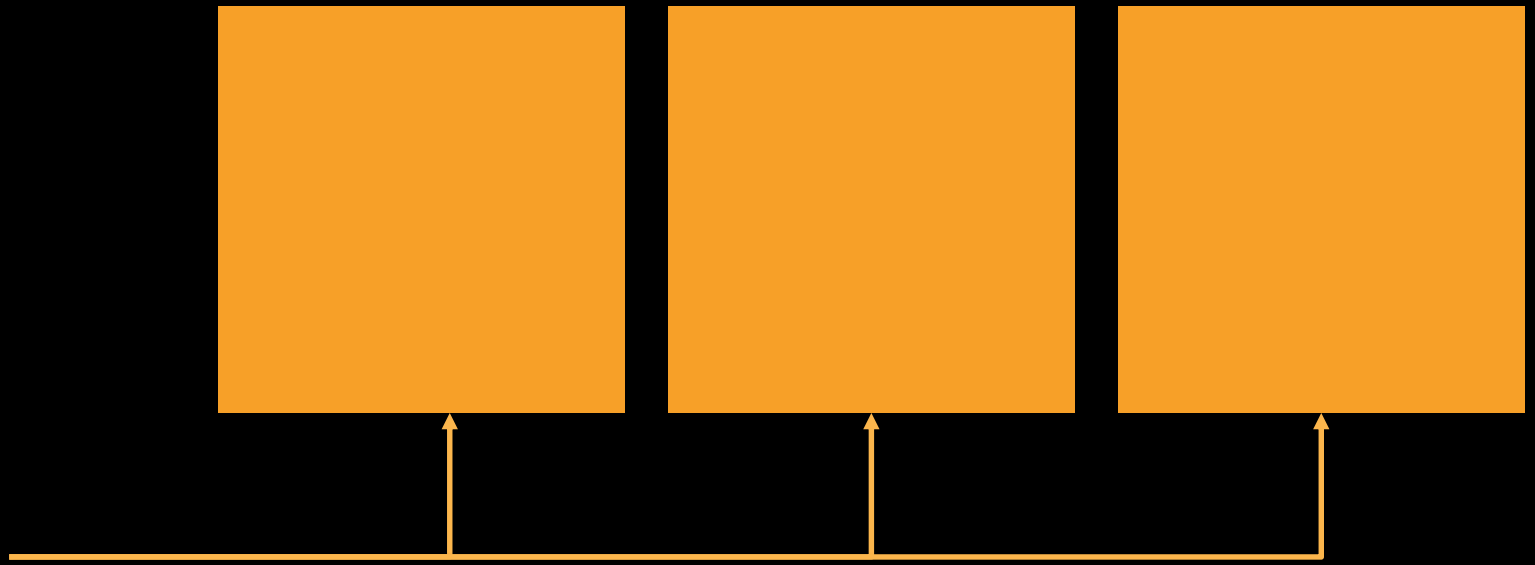


Configuration Updates

The problem

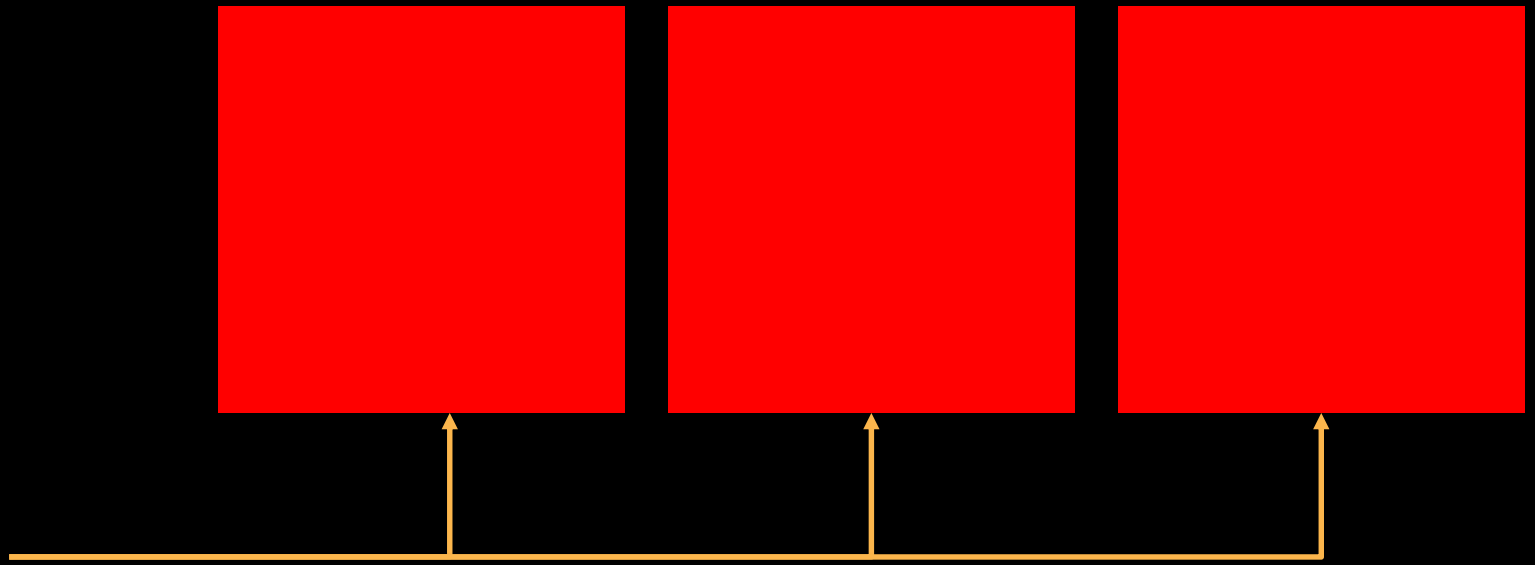
Binary Patches

The problem



Pictures of Cats

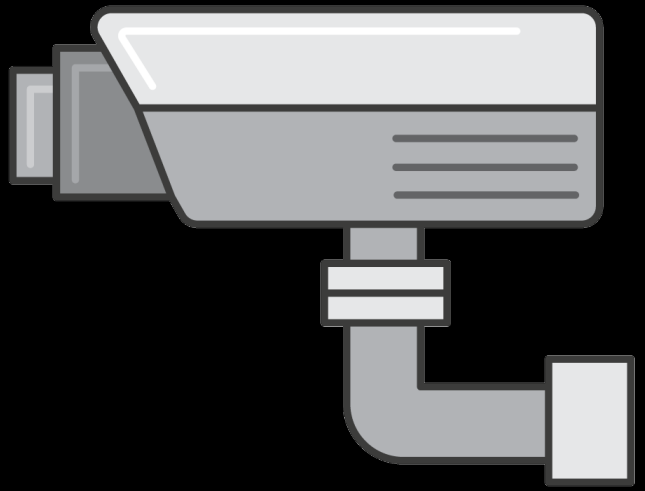
The problem



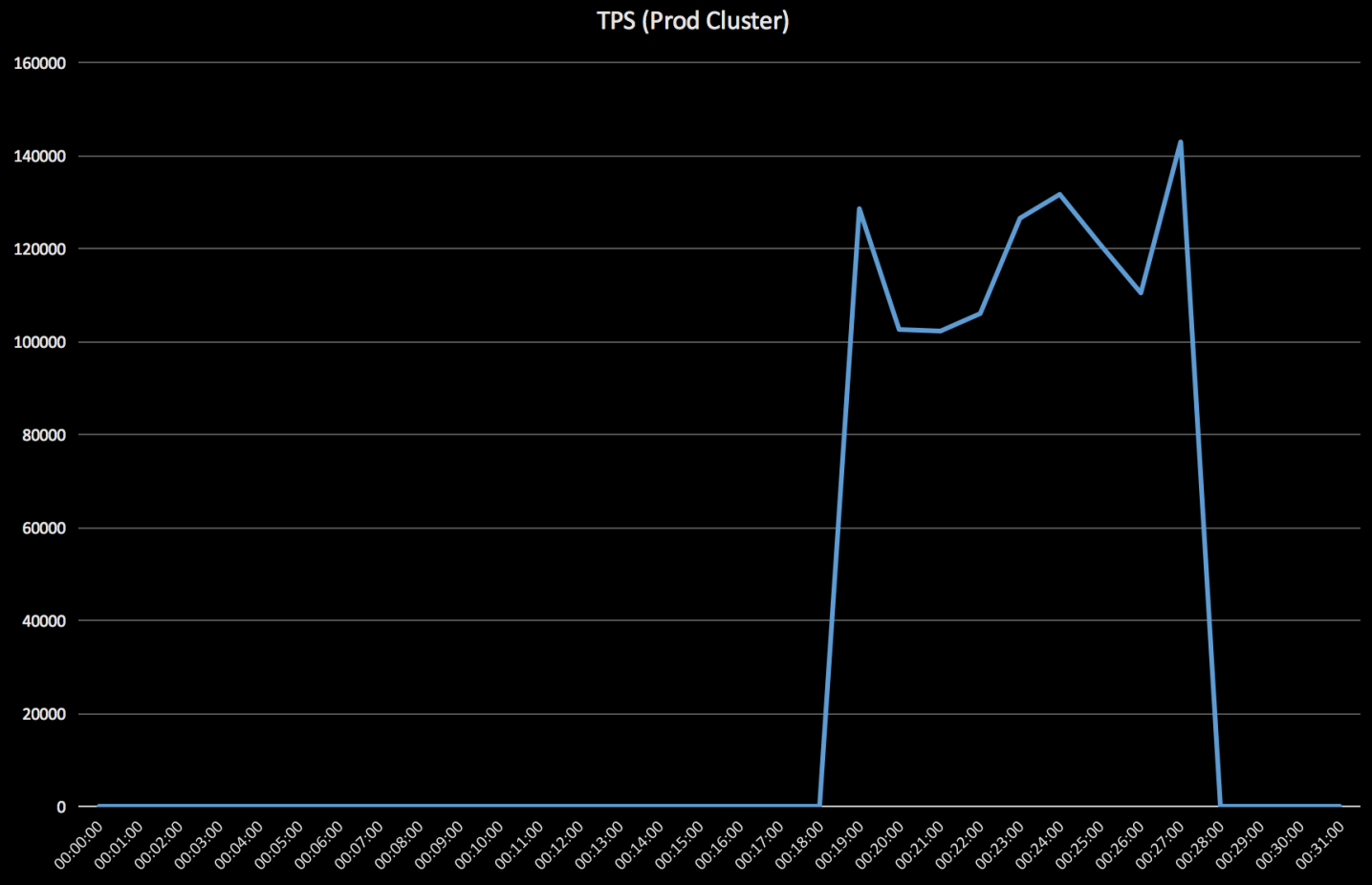
Pictures of Cats

Traditionally..

- Instance level monitoring system with alerts
- Human response
- Monitor both percentage fill, and the fill rate



High availability matters

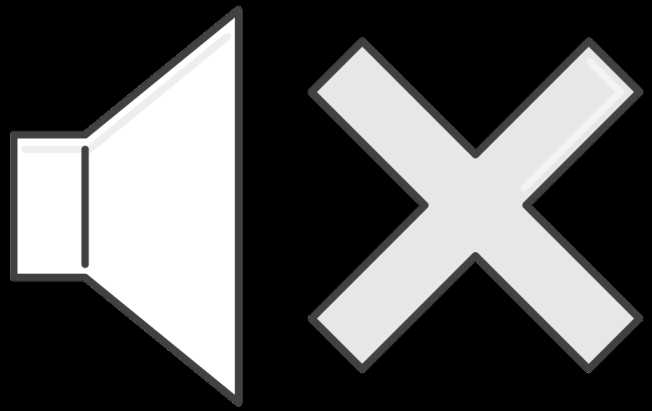


High availability matters

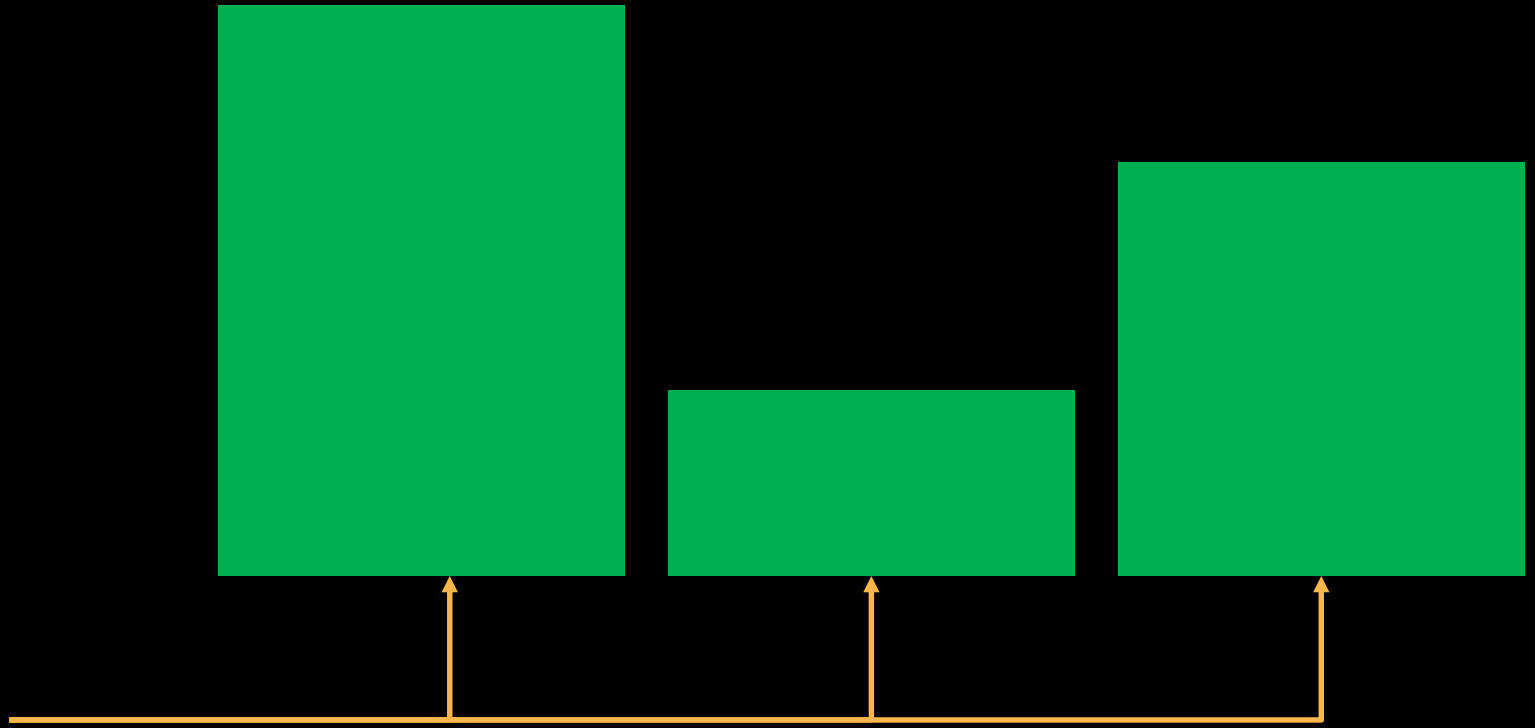
Recycling old instances is **unnecessary cost**

Automated cleanups may be **too slow**

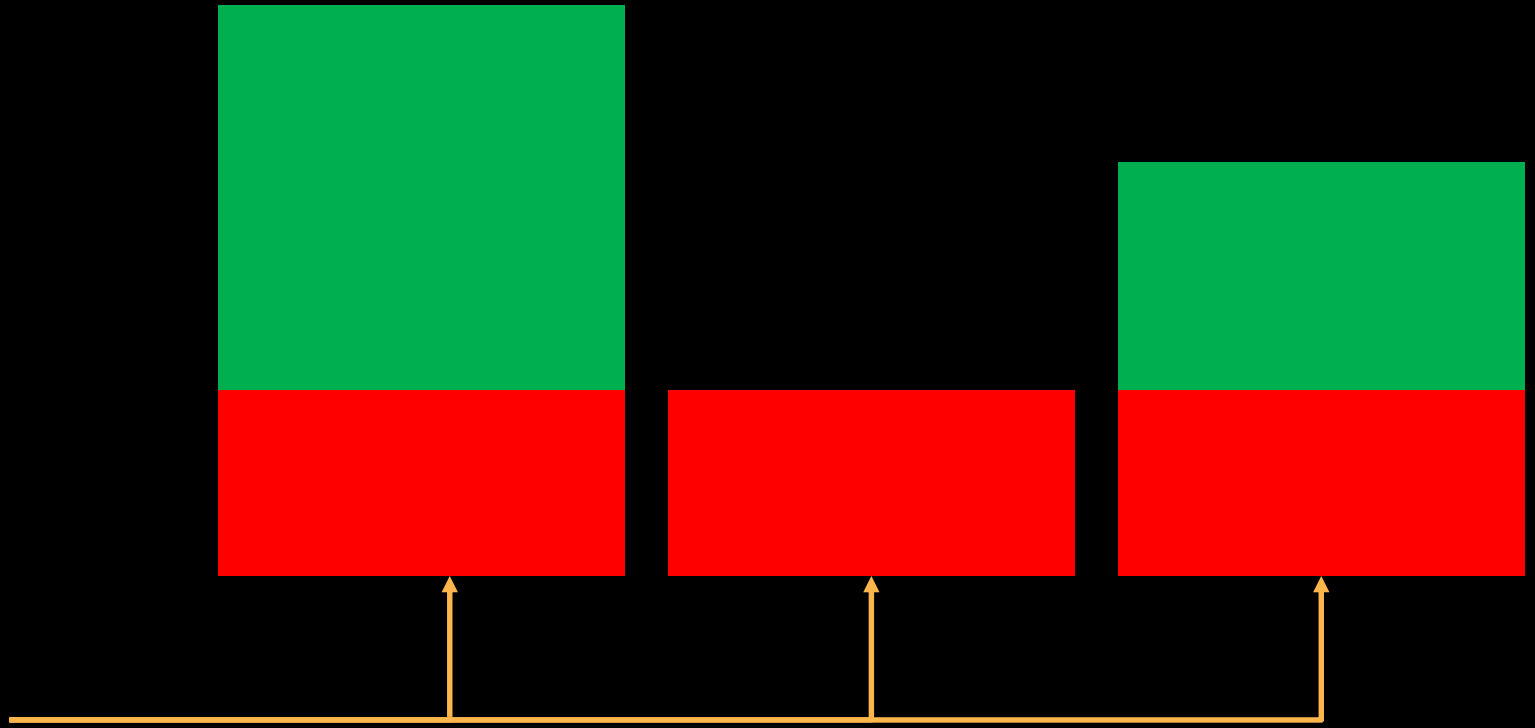
Human operators **almost certainly** too slow



The solution



The solution

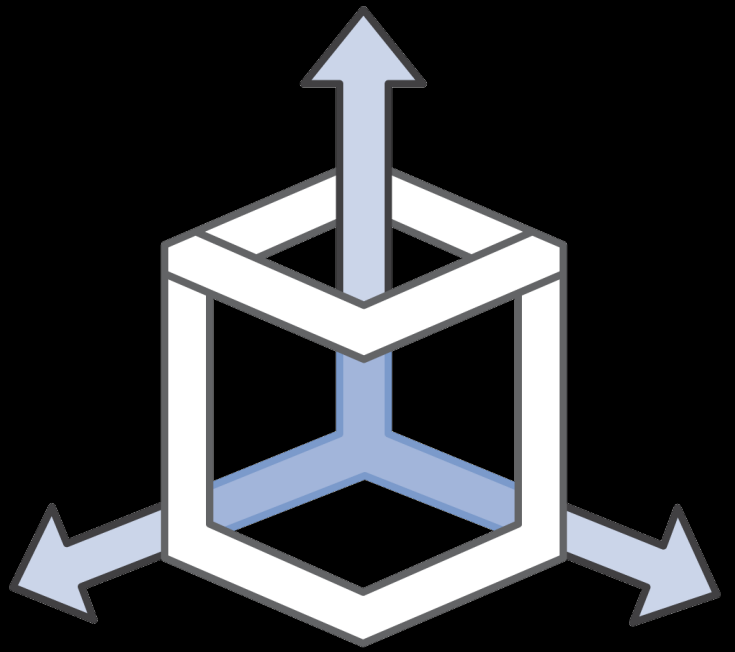


AWS solution – within an Auto Scaling group

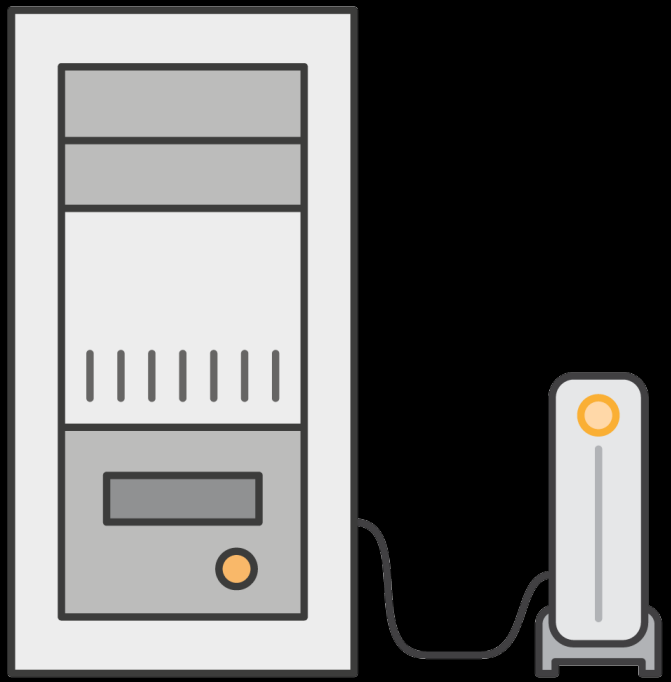
Amazon Linux AMI has /var/run (and /var/tmp) on root

Ubuntu AMI uses tmpfs (10% of RAM)

Jittering **root volume size** is tricky within an ASG



AWS solution – within Auto Scaling



Consciously segregate your files to a specified directory – on a **separate volume** – with jittering

Code will follow (check SlideShare!)

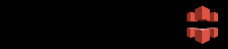
Adds <10s to system startup

What else?

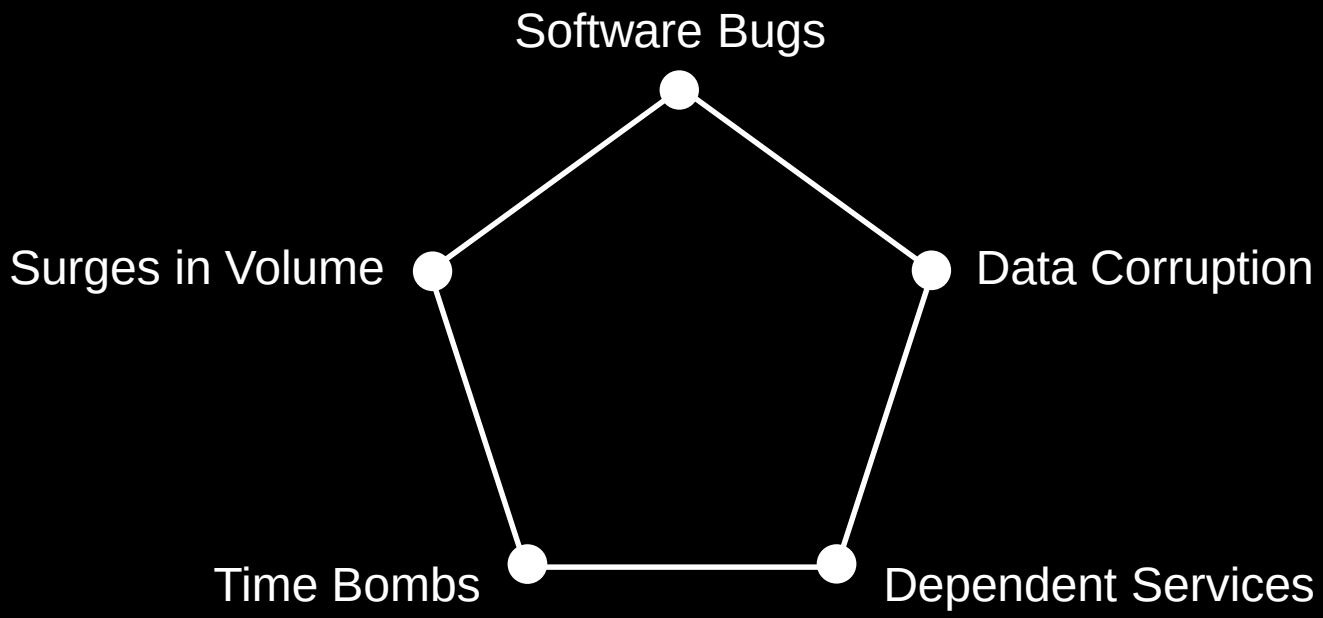
Time

- SSL Certificates
- Domain name registrations
- Deployment Schedules
- ???

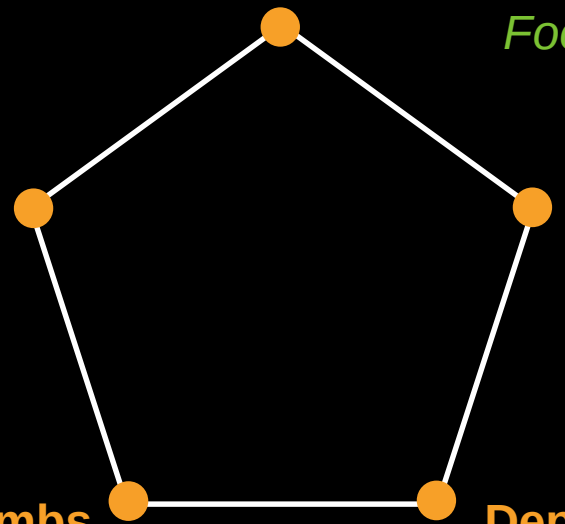
Wrap up:
Lessons learned



Risks to availability



Risks & Common Patterns



Software Bugs

Multi-impl
Sharding
FoodTaster

Surges in Volume

Load dispersion

Data Corruption

FoodTaster

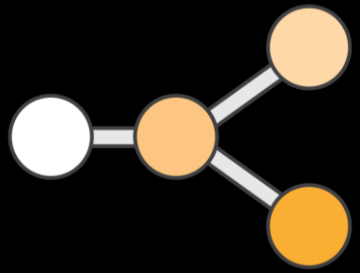
Time Bombs

Jitter

Dependent Services

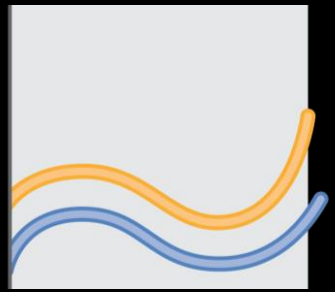
Cache
Redundancies

Lessons learned



Maximizing availability with Food Tasting

Don’t rely on validation in your main application



Flash Crowds without scaling for the peak

Auto Scaling Integration; Caching; Selective Serving

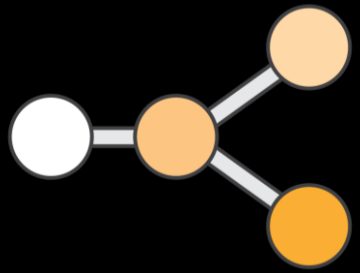
Defense in Depth Strategies

Implementation striping can save you

Time Bomb Jitter Protection

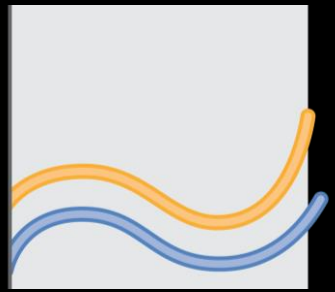
Homogeneity can hurt

Lessons learned



Maximizing availability with Food Tasting

Don’t rely on validation in your main application



Flash Crowds without scaling for the peak

Auto Scaling Integration; Caching; Selective Serving

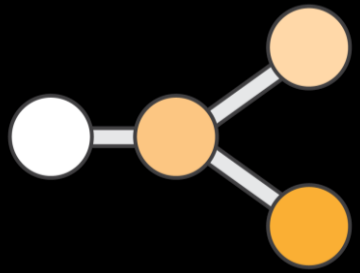
Defense in Depth Strategies

Implementation striping can save you

Time Bomb Jitter Protection

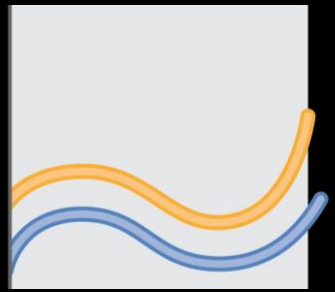
Homogeneity can hurt

Lessons learned



Maximizing availability with Food Tasting

Don’t rely on validation in your main application



Flash Crowds without scaling for the peak

Auto Scaling Integration; Caching; Selective Serving

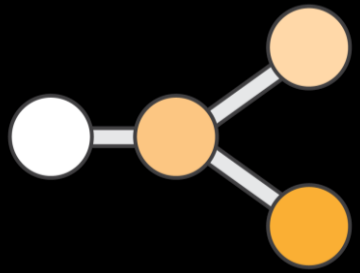
Defense in Depth Strategies

Implementation striping can save you

Time Bomb Jitter Protection

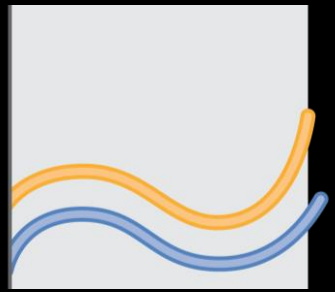
Homogeneity can hurt

Lessons learned



Maximizing availability with Food Tasting

Don’t rely on validation in your main application



Flash Crowds without scaling for the peak

Auto Scaling Integration; Caching; Selective Serving

Defense in Depth Strategies

Implementation striping can save you

Time Bomb Jitter Protection

Homogeneity can hurt

Thank you!

Remember to complete your evaluations!

Related Sessions

ARC309

Moving Mission Critical Apps from One Region to Multi-Region active/active

ARC204

From Resilience to Ubiquity - #NetflixEverywhere Global Architecture