Chaos

Addressing the challenges of Complex Distributed Systems at Scale

IEEE Reliability Roundtable 2015

A Little About Me

- Founder of Chaos Engineering at Netflix
- Scaled Netflix systems from 8M subscribers to 60M
- Computer Science background
- Technical Leadership

@bruce_m_wong

A Little About Netflix

- 33+% of North America Internet Traffic at Peak
- Amazon Web Services, one of the largest customers
- Over 1B hours of Netflix viewed every 2 weeks (as of Q1.2015 earnings call)
- Very diverse device interactions
  - Mobile, Laptops, TVs, Set-Top-Boxes

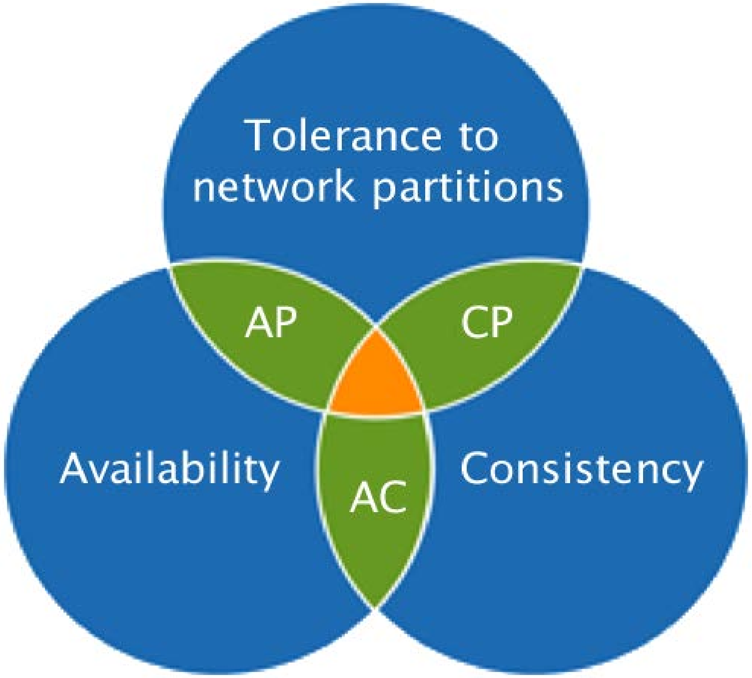
NETFLIX


SCALE
@bruce_m_wong

Scale Presents Challenges

- Vertical Scale has limits (bigger more expensive hardware)
- Horizontal Scale has complexity
- Large Monolithic systems are difficult to change and maintain reliability
- Micro-Services add complexity

CAP Theorem



- Databases – CAP Theorem
- Availability or Consistency? Orange doesn't exist

Complex Systems

- Very Difficult to model
- Impossible to simulate scale

Modeling

Attempting to represent a system with the purpose of predicting behavior

- Human Behavior interacting with systems
  - Social Media: live events, tv-shows, news, etc.
  - Popularity of Goods, entertainment, etc.
- System Failures
  - Network partitions
  - Hard Drives Fail
  - Power Outages
- Natural Disasters

@bruce_m_wong

Simulation

Simulating conditions of a system often with the purpose of testing

- Lack of modeling and scale make this impossible

- Network Conditions
  - Latency, new interconnections, shared infrastructure
- Simulation at scale
  - Would effect and change the Internet Network Conditions
- Data and Capacity
  - Likely too expensive to replicate
  - Constant stream of new data

Fault-Tolerant Systems

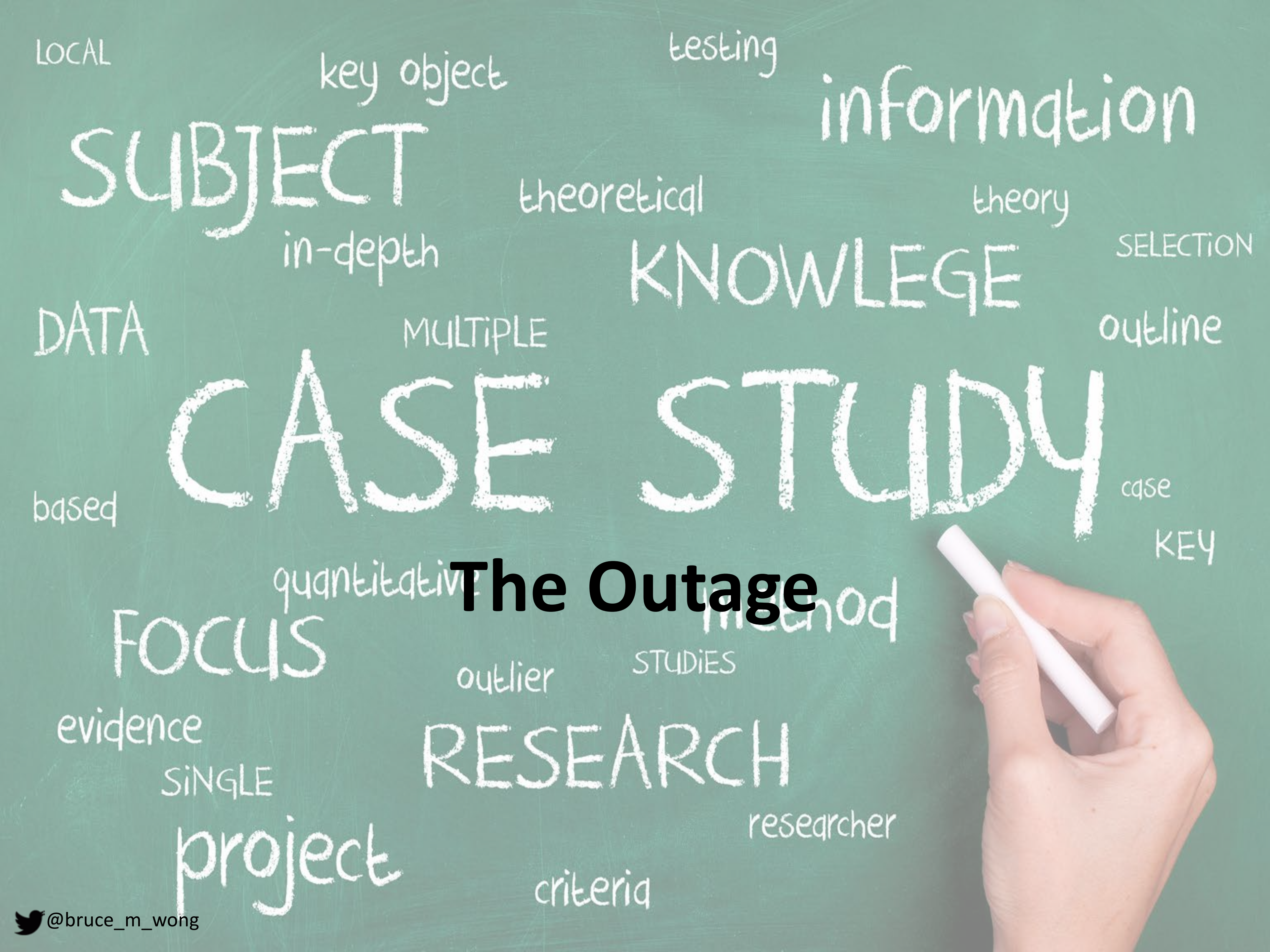
Designing a system to handle failure gracefully

- Eliminating Single-Points of Failure
- Allowing different aspects or micro-services to fail independently (Failure Isolation)
- Prevent propagation (Failure Containment)

Fault-Tolerant Systems

How do you validate a fault-tolerant system can indeed fail gracefully?

- If you can't model it
- If you can't simulate it



The Outage

@bruce_m_wong

Case Study: The Outage

Lets take a User Preferences Service (UPS)

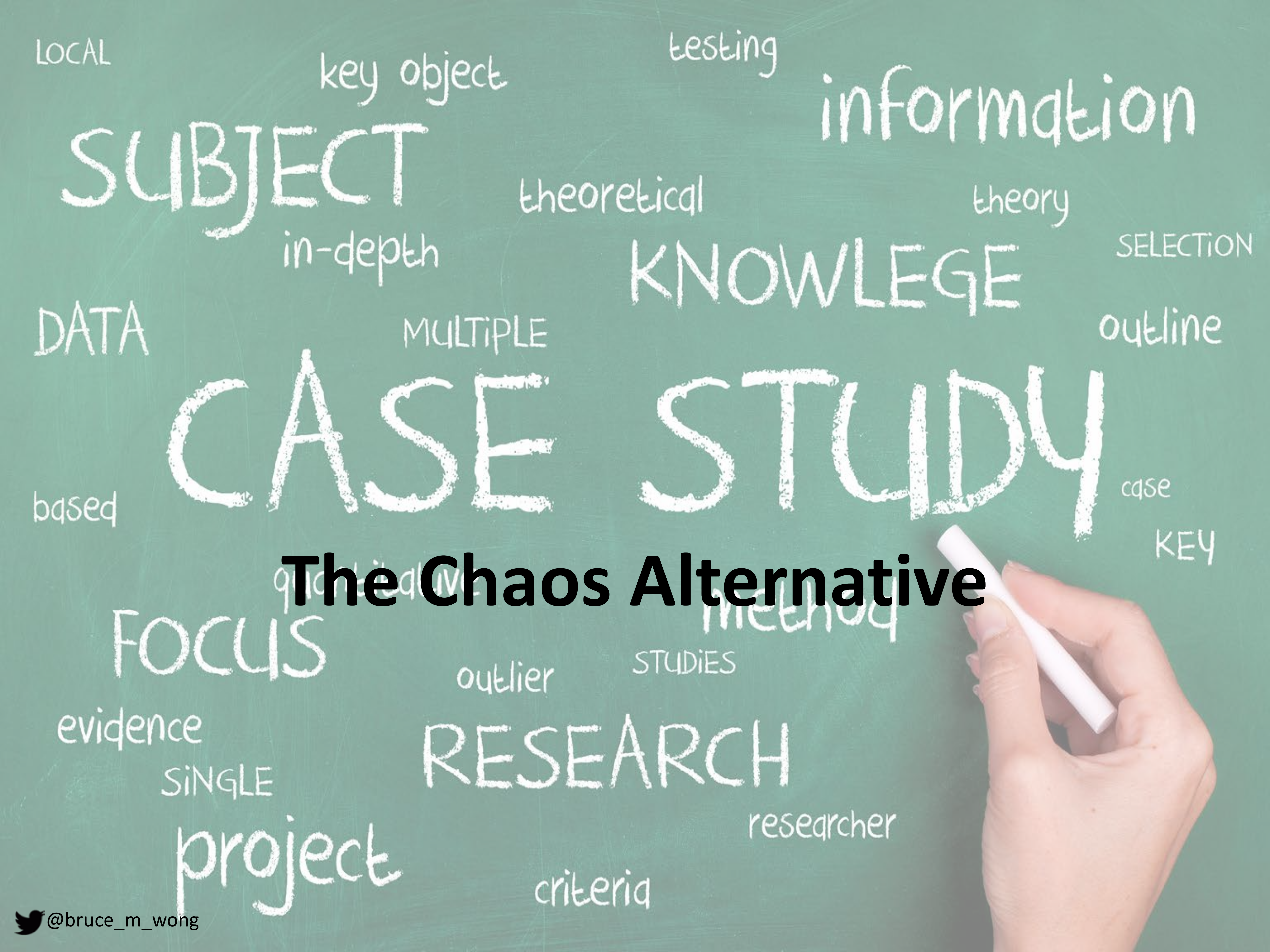
- Well Architected, Fault-Tolerant Design
- When unavailable users can't update their preferences, but product still has their last known preferences
- UPS can fail independently of the rest of the system

Case Study: The Outage

- Changes to UPS happen
  - Features, system configuration, growth, etc.
- A change gets introduced that breaks the ability for the product to function when UPS is unavailable
- Months Pass before UPS experiences downtime
- Surprise system wide outage

Case Study: The Outage

- Team scrambles to bring back service
  - All hands on deck, people woken up
  - Resources spent troubleshooting and trying to determine what went wrong
  - Customers impacted
- Post-Mortem(s) happen
  - Talk and design how to prevent recurrence
  - Changes Implemented



The Chaos Alternative

@bruce_m_wong

Case Study: The Chaos Alternative

Lets take the same UPS

- Changes to UPS happen
  - Features, Configuration, etc.
- Chaos Exercises Regularly scheduled to validate resilience design

Case Study: The Chaos Alternative

- Exercise exposes misconfiguration that breaks graceful degradation
- Configuration is fixed right away
- Another Chaos Exercise is scheduled to validate

Case Study Summary

The Outage

- Big user impact
- Resource intensive
- Uncontrolled
- Unpredicted
- Unintended failure

The Chaos Alternative

- Microscopic user impact
- Resource efficient
- Controlled
- Planned
- Intended failure

Chaos

Chaos is the discipline and practice of intentionally injecting failure into a production system

- Validation of Resilience Design
- Reduce Risk of Drift caused by change and growth
- Controlled and Planned
- Effective to Validate both Isolation and Containment Strategies

Chaos Exercise

Understand failure and prove resilience through introducing controlled failure

- Returning a % of Errors
- Introducing latency
- Find single-points of failure
- Availability-Zone Failure Evacuation
- Regional Failure Evacuation

Chaos Proven: Eliminating SPOF

In Q3.2014 a vulnerability was found that required AWS to reboot ~10% of all instances

Over 10% of database nodes were rebooted, 1% didn't come back.

Zero Downtime

Chaos Proven: Isolation

Learning more from 1-minute of controlled chaos than a multi-hour unpredicted, uncontrolled outage

- A single Critical Micro-service had many issues causing multiple system-wide outages over the course of months
- Multiple Chaos Exercises allowed the team to iterate on it's resilience design and eventually validate and prove resilience in the face of failure.

Chaos Proven: Containment

Measures to prevent the propagation of failure.

- The goal is to keep failure impact contained as small as possible
- Instance > Cluster > Availability Zone > Region

In 2014, Netflix executed 12 Regional evacuation exercises

- Confidence to use evacuation procedures at a moment's notice

Confidence in Containment

- Simplifies recovery steps in the face of system outages
- After Detection, Time is usually spent in investigating and analysis
- With robust containment and evacuation, impact can be mitigated while investigation and analysis is done.

Fault-Tolerant Systems meet Chaos

Fault-Tolerant Principles

- Eliminating Single-points of failure
- Allowing different aspects or micro-services to fail independently (Failure Isolation)
- Prevent propagation (Failure Containment)

Chaos Principles

- Discovery of single-points of failure
- Validate failure isolation design and prevent drift
- Proactively prove containment

@bruce_m_wong