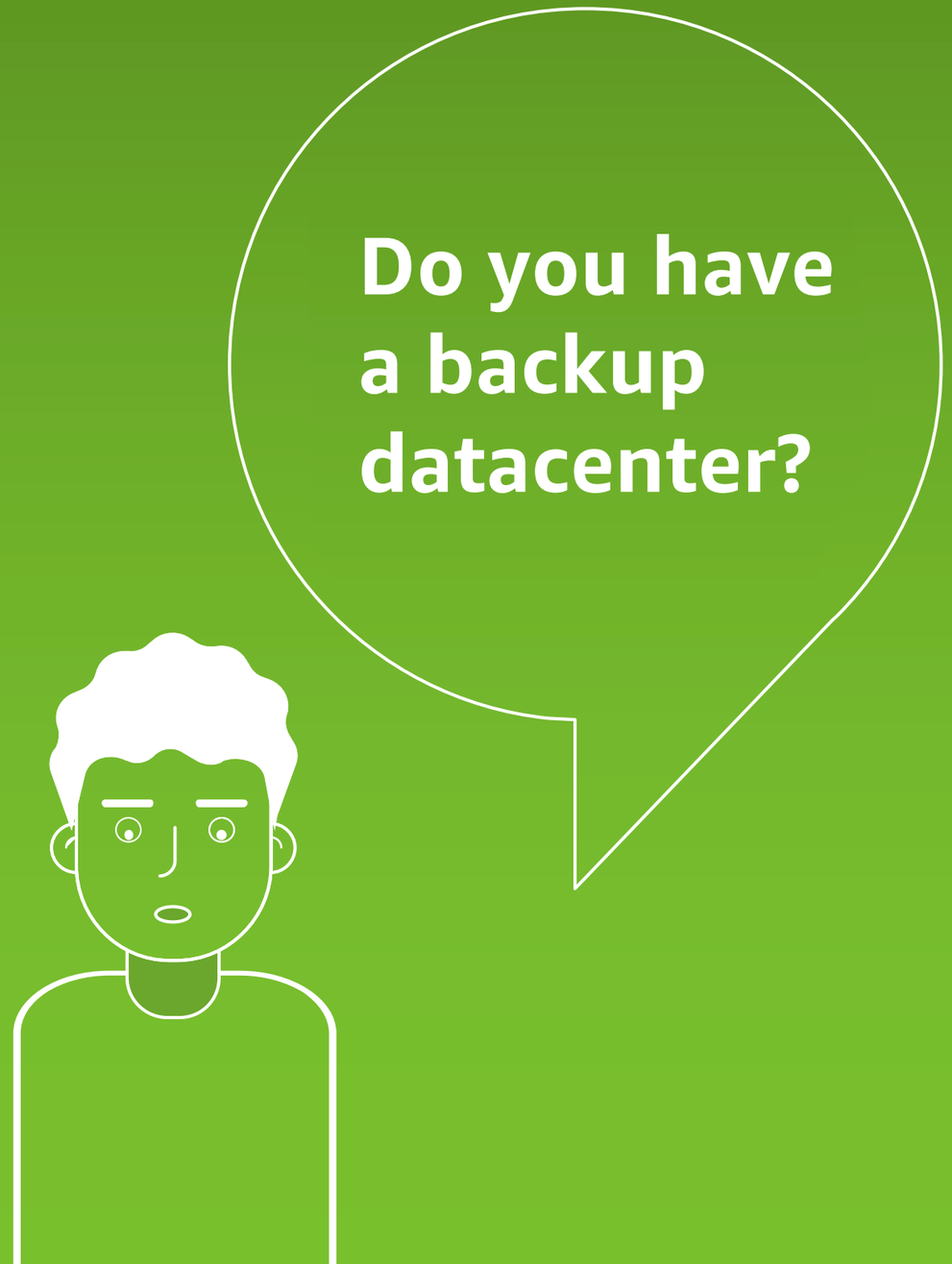
Failing Over Without Falling Over

Adrian Cockcroft

@adrianco

AWS VP Cloud Architecture Strategy



How often do you failover apps to it?

How often do you failover the **whole datacenter** at once?

“Availability Theater”

“Major publicly recorded outages are now more likely to be caused by IT and network problems than a year or two years ago, when power problems were a bigger cause.”

https://uptimeinstitute.com/publicly-reported-outages-2018-19

“IT and network problems…”

The inspiration for Perrow's book [Normal Accidents 1984] was the 1979 Three Mile Island accident, where a nuclear accident resulted from an unanticipated interaction of multiple failures in a complex system. The event was an example of a normal accident because it was

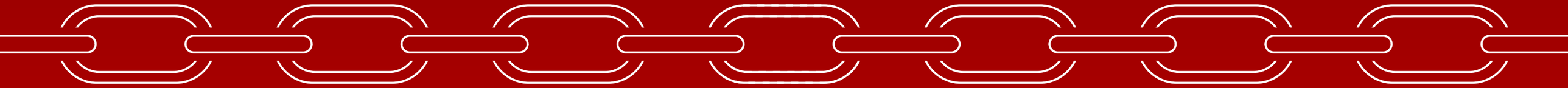
"unexpected, incomprehensible, uncontrollable and unavoidable".

https://en.wikipedia.org/wiki/Normal_Accidents

We build redundancy into systems so that if something fails, we can fail-over to an alternative

However, our ability to fail-over is complex and hard to test, so often the whole system falls over

How can we do better?

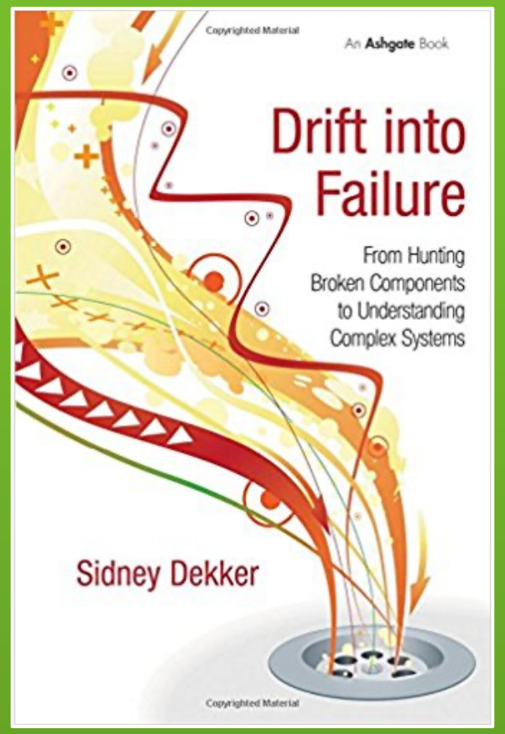


You can only be as strong as your
weakest link

Dedicated teams are needed to find weaknesses before they take you out!

But the last strand that breaks is not the cause of failure!

Build resilient systems like a rope, not a chain, but make sure you know how much margin you have, and how “frayed” your system is…



Drift into Failure

Sydney Dekker

Everyone can locally optimize for the right outcome at every step, and you may still get a catastrophic failure as a result…

We need to capture and learn from near misses, test and measure the safety margins, before things go wrong.

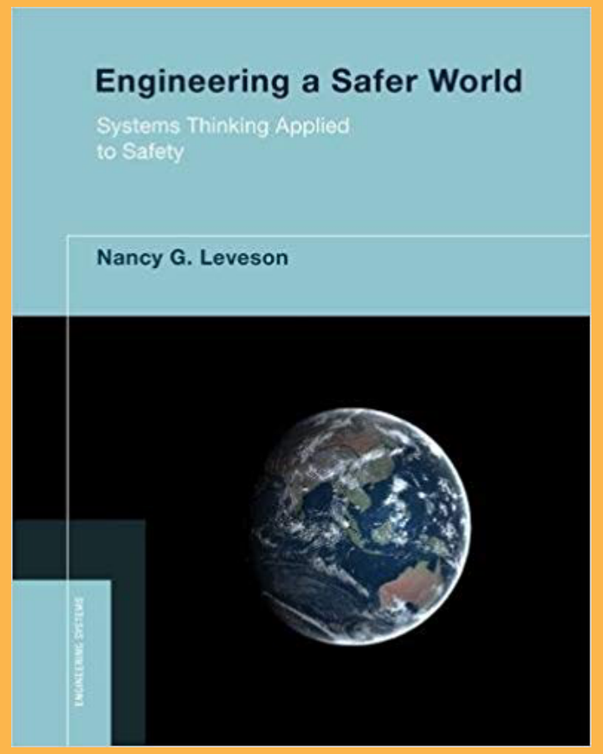
“You can’t legislate against failure, focus on fast detection and response.”

—Chris Pinkham

Control

Observability

Let’s see what we can learn from experts who’ve been working on controlling safety critical systems for decades…



Engineering a Safer World

Systems Thinking Applied to Safety - 2012

Nancy G. Leveson

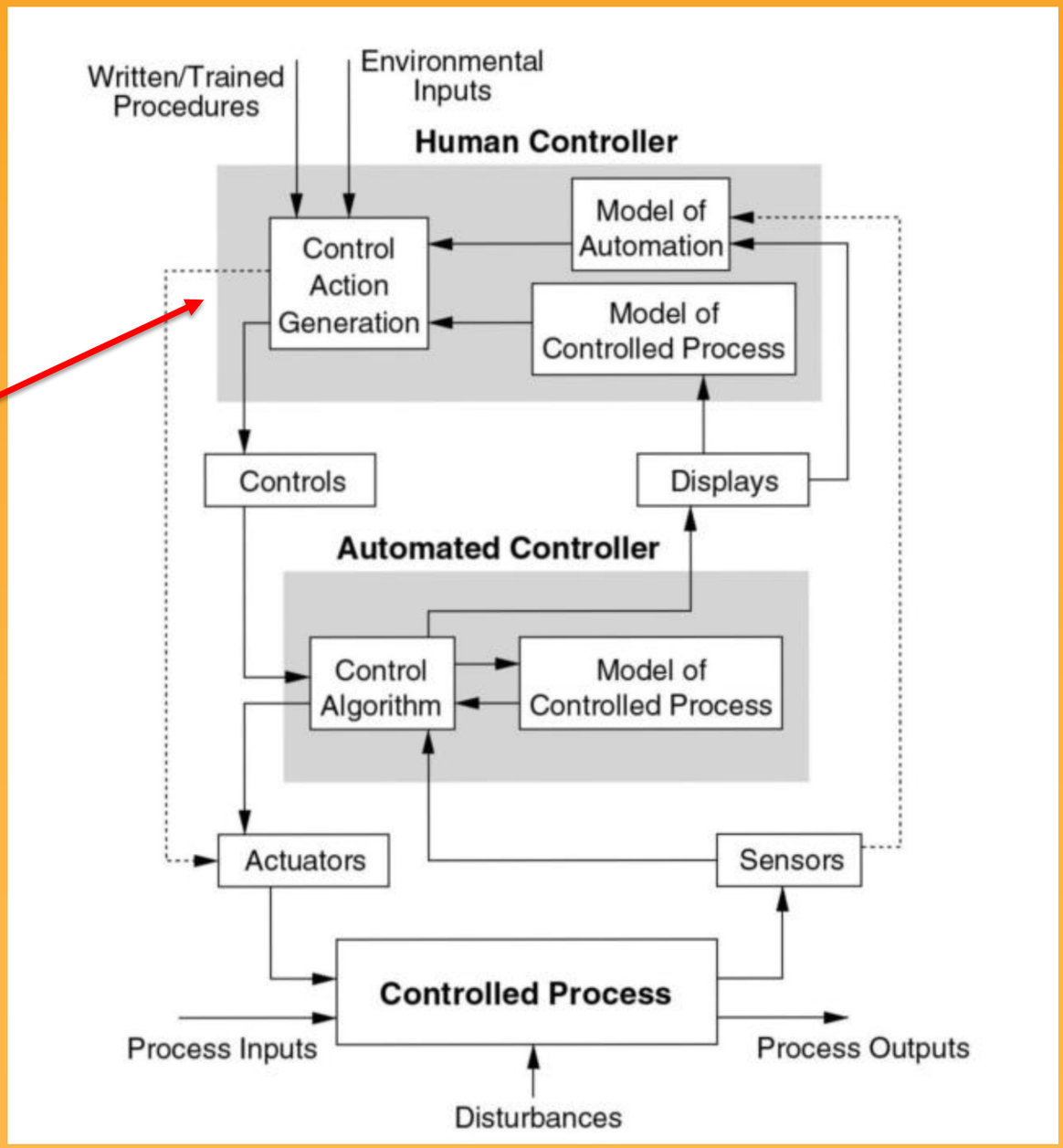
STPA – Systems Theoretic Process Analysis

STAMP – Systems Theoretic Accident Model & Processes

http://psas.scripts.mit.edu for handbook and talks

Observability

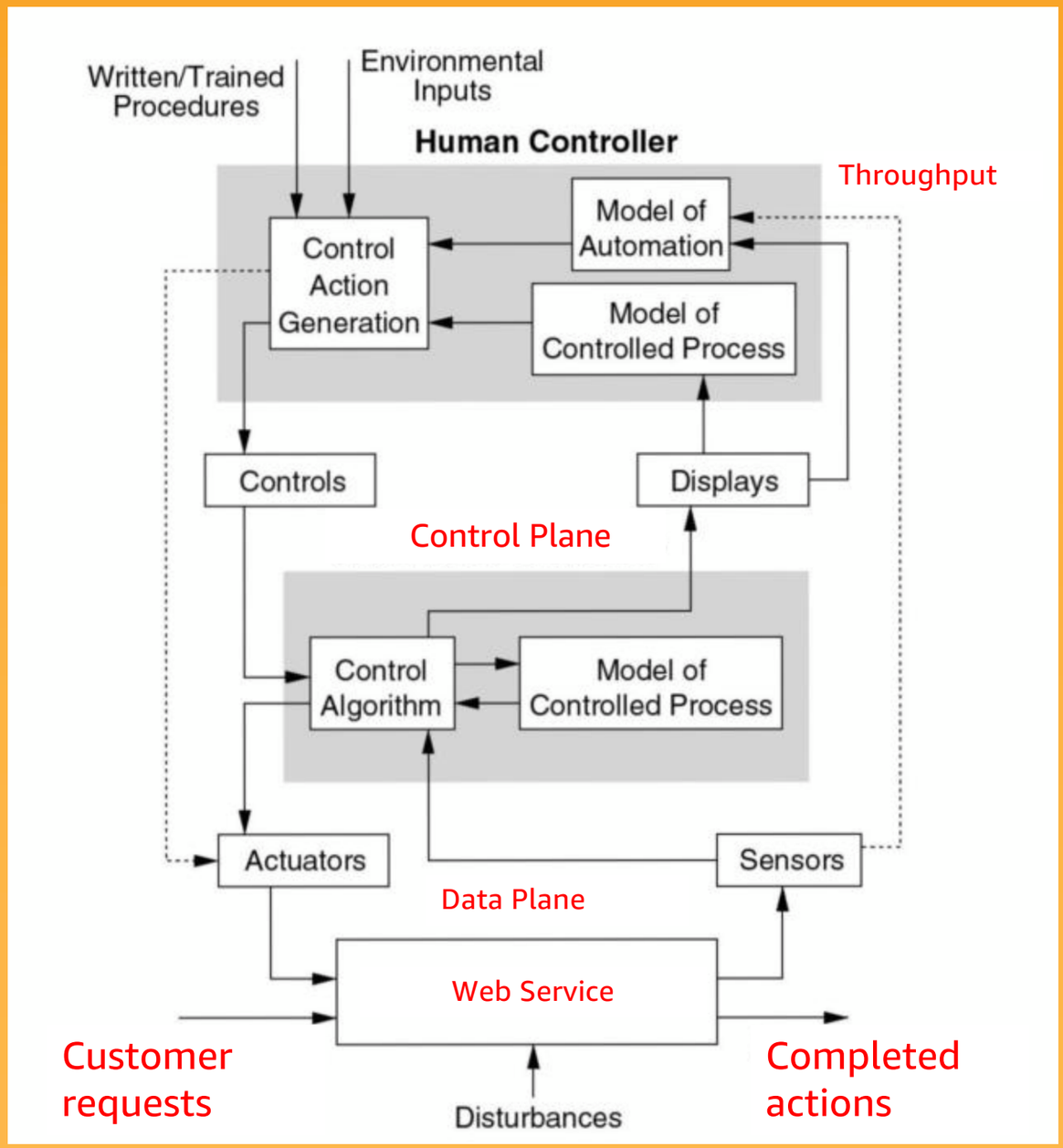
STPA Model Control Structure (System Theoretic Process Analysis)



Observability

STPA Model

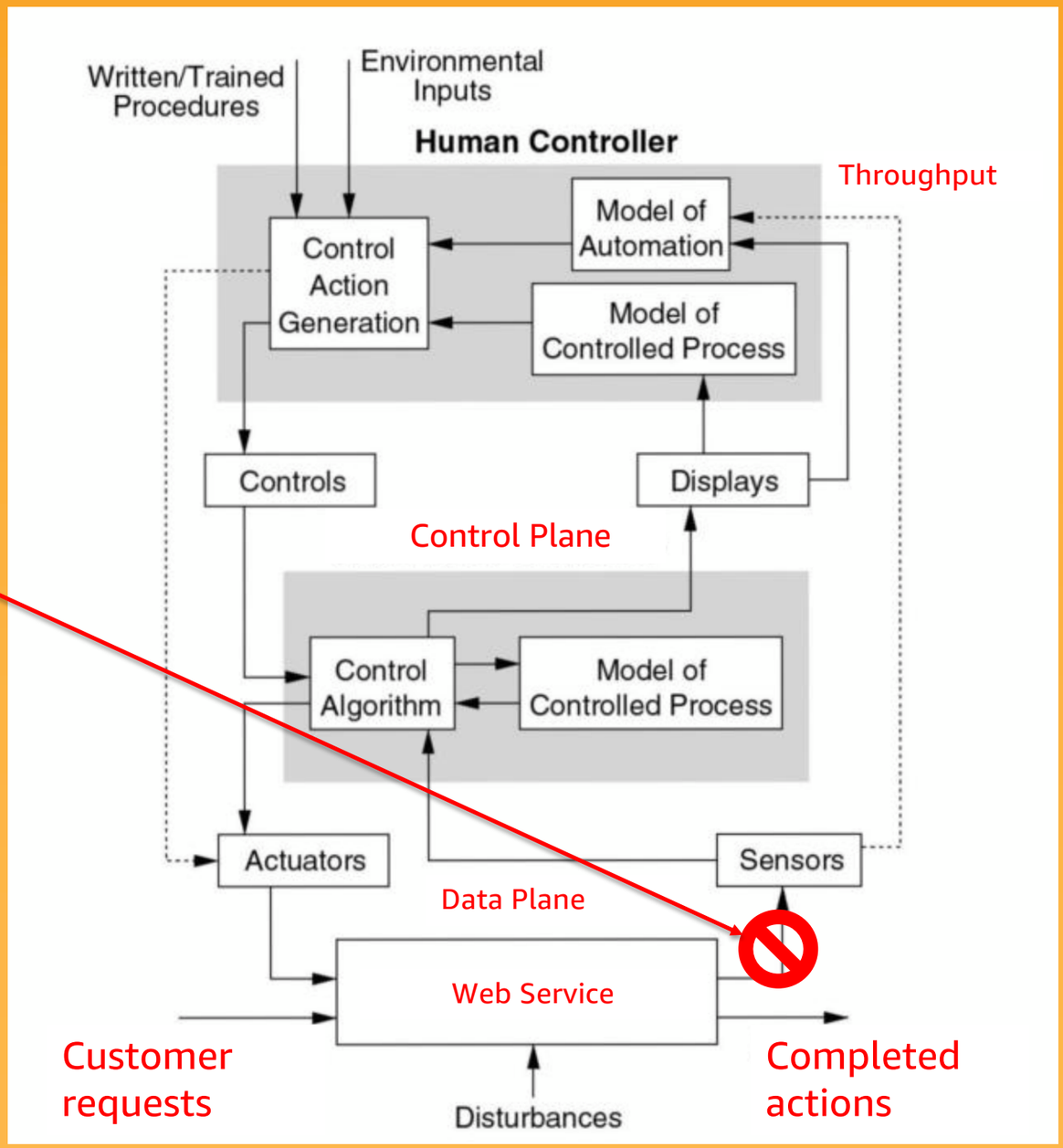
Understand Hazards that could disrupt successful application processing



STPA Hazards

Sensor Metrics:

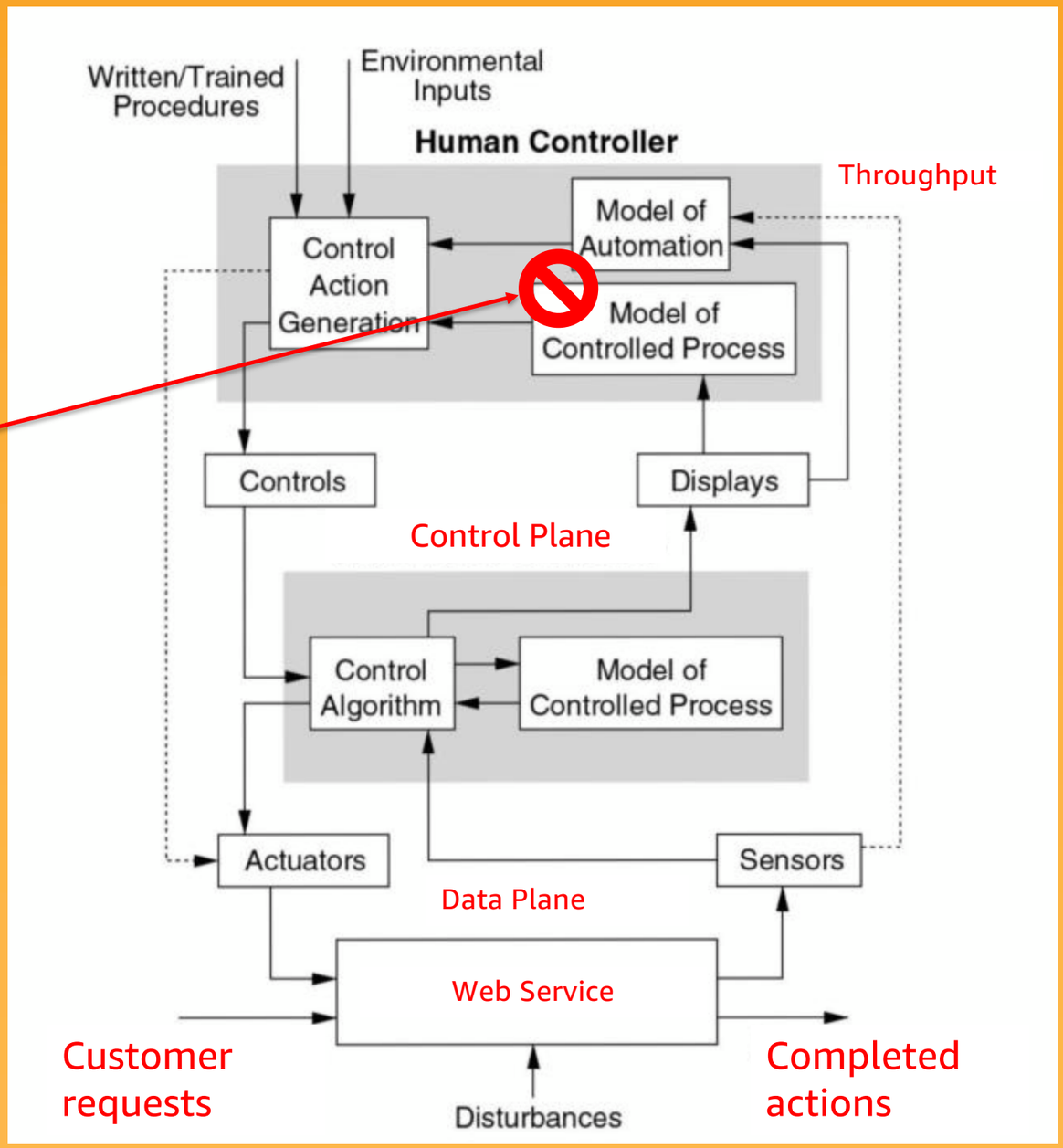
Missing updates
Zeroed
Overflowed
Corrupted
Out of order
Updates too rapid
Updates infrequent
Updates delayed
Coordination problems
Degradation over time



STPA Hazards

Model problems:

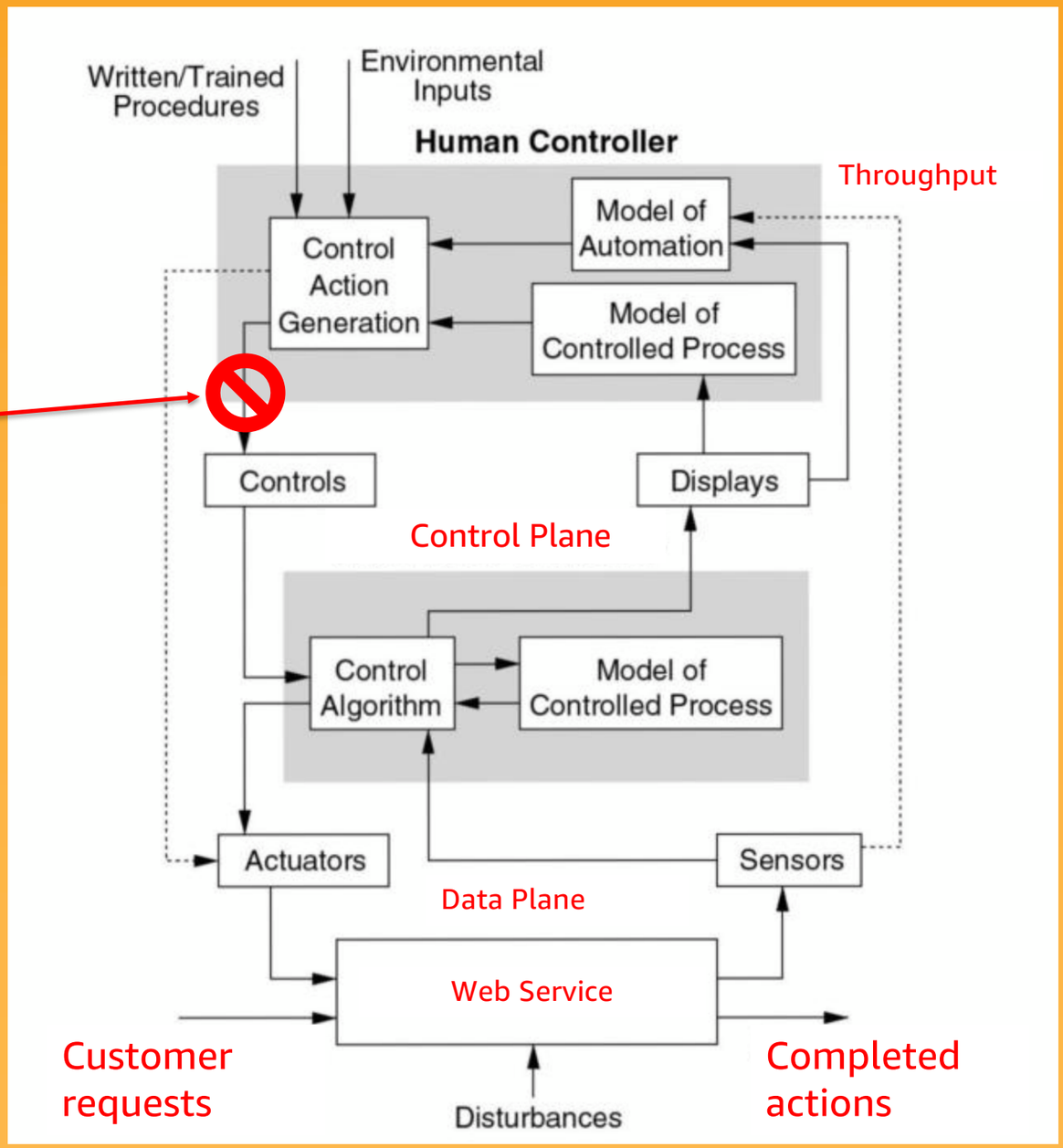
- Model mismatch
- Missing inputs
- Missing updates
- Updates too rapid
- Updates infrequent
- Updates delayed
- Coordination problems
- Degradation over time



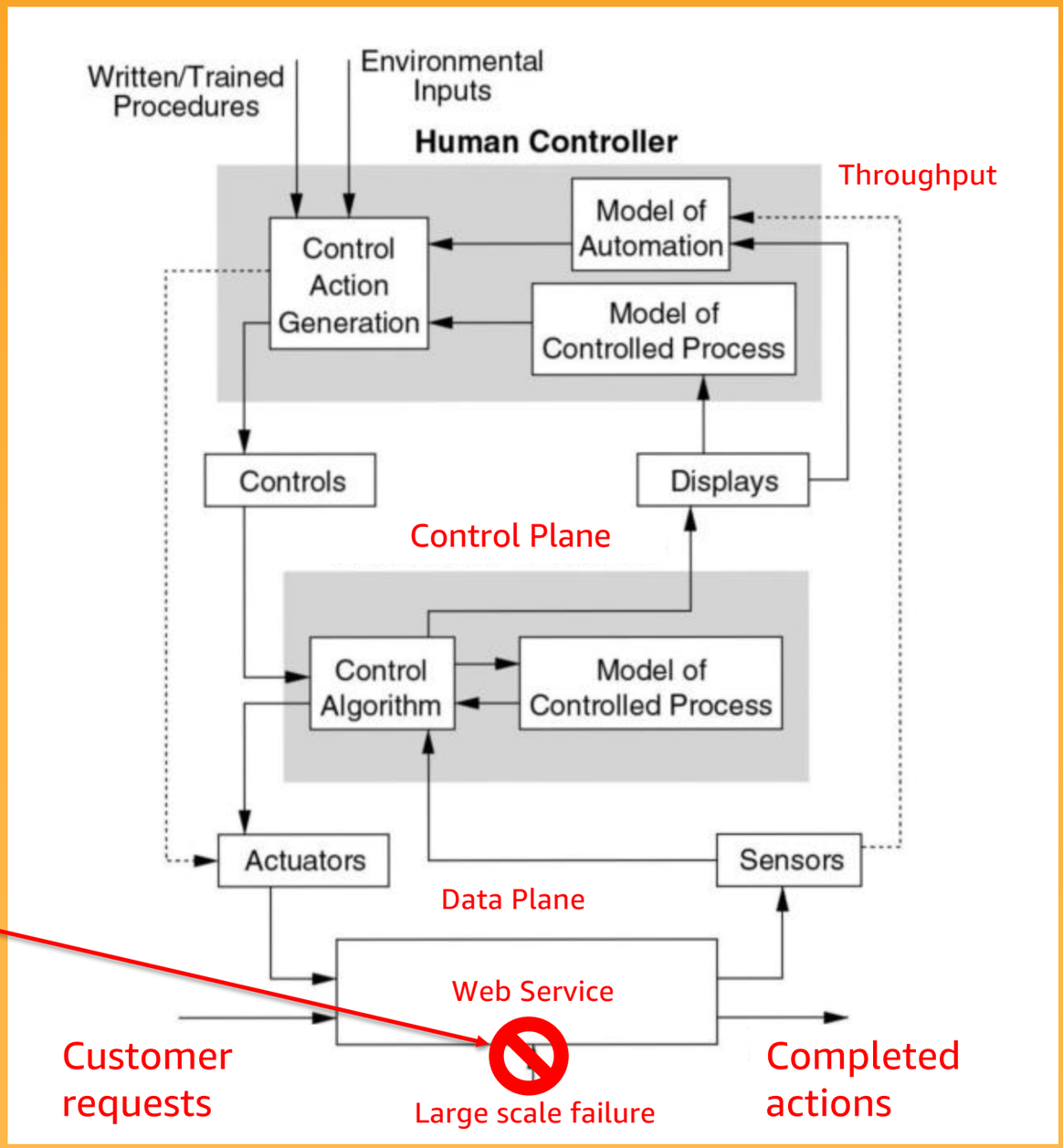
STPA Hazards

Human Control Action:

Not provided
Unsafe action
Safe but too early
Safe but too late
Wrong sequence
Stopped too soon
Applied too long
Conflicts
Coordination problems
Degradation over time



What happens if there is a big enough Disturbance to break the Web Service?

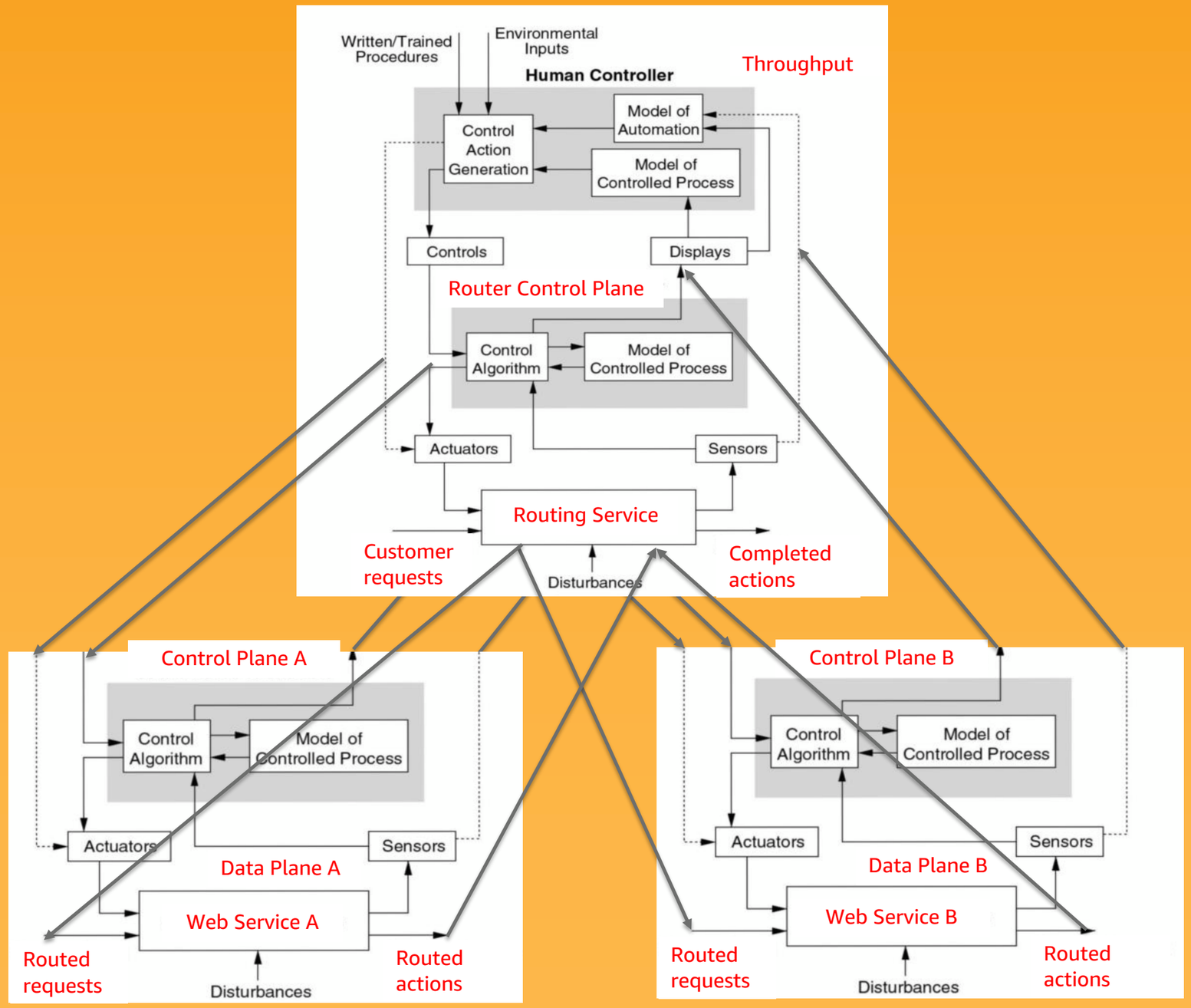


Large Scale Failures

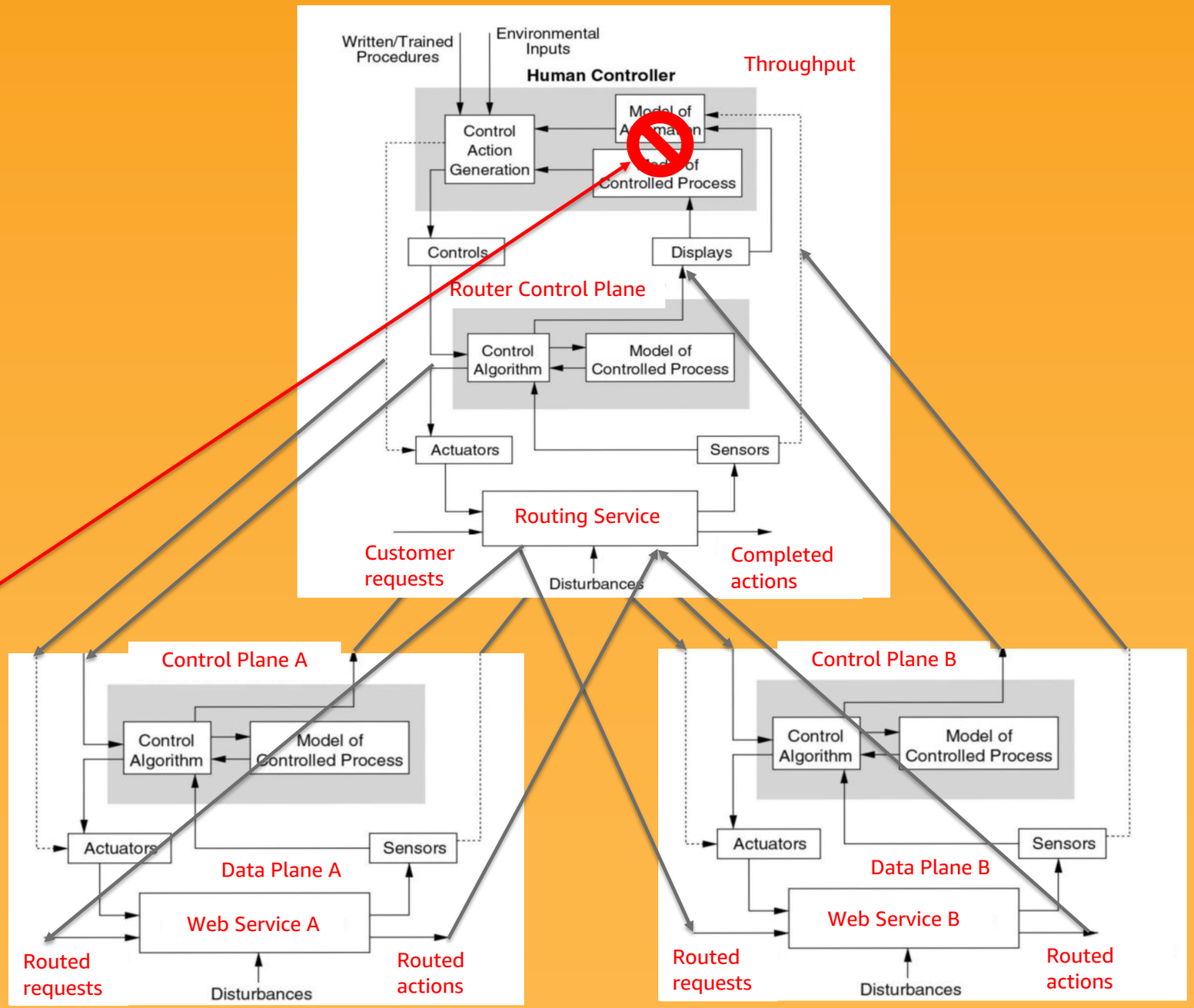
Control plane is unable to compensate for disturbance

- Control plane automation has failed, application is “out of control”
- Network partition, no route connecting application to customers
- Application crashed or corrupted, not easily restartable

Mitigation through replication, adds a router control plane and a routing service



More to go wrong, and much more complexity for the human controller to try to model



Use architectural symmetries to simplify the human controller’s mental models, and use tooling to enforce symmetry

Cloud provides consistent automation to implement modern control planes, that have useful symmetries

Symmetries

High level of automation, consistent configuration as code

Consistent instance types, services, versions, zones and regions

Principles

If it can be the same, make it look and act identically

If it’s different, make that clearly visible

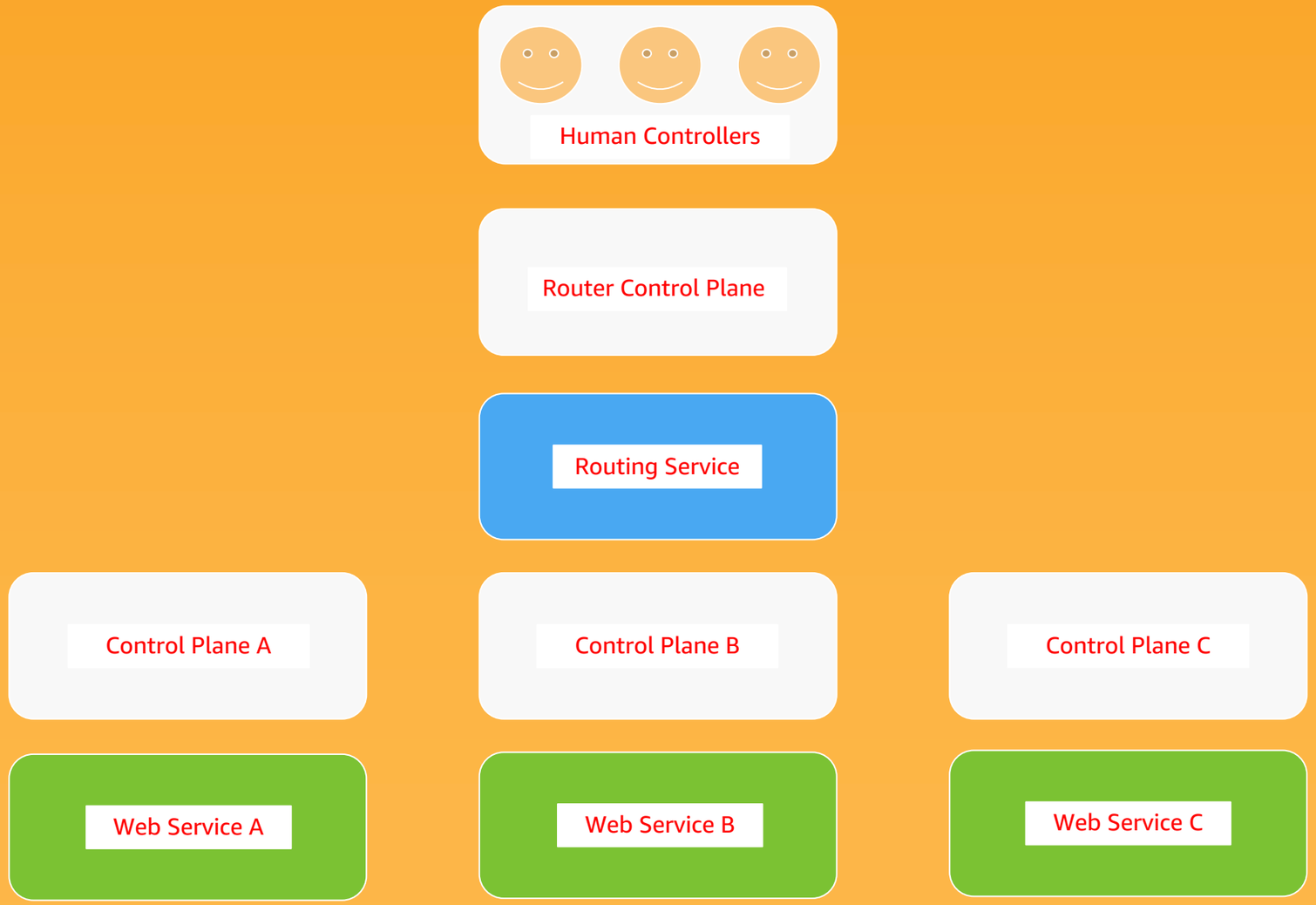
Test your assumptions continuously

Good Cloud Resilience Practices

Rule of 3 – three ways for critical operations to succeed

- Synchronous data replication over three zones in a region
- Active-Active-Active workloads across three regions
- DR failover from primary region to either of two secondary regions for very high value applications

Distilling the complex diagram and extending to triple replication



Scenario: AWS Availability Zones

Symmetry and assertions:

Services and data are consistent across three zones

Zone failure modes are independent

Application should work normally with a zone offline

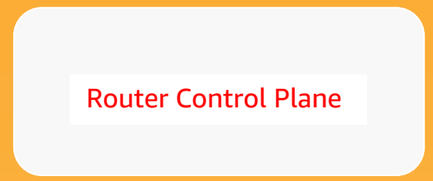
Scenario: AWS Availability Zones

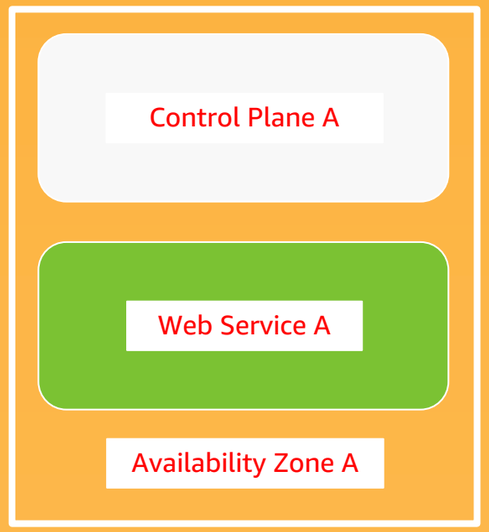
Router control plane detects offline AZ, stops routing traffic to it, retries requests on the online AZs

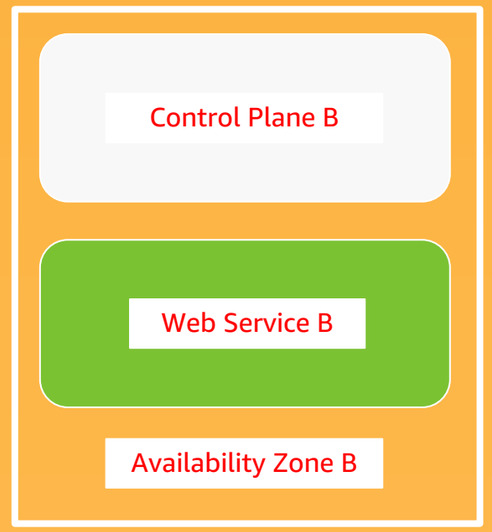
Automated response, What could go wrong?

STPA Hazards Sensor Metrics:

Missing updates
Zeroed
Overflowed
Corrupted
Out of order
Updates too rapid
Updates infrequent
Updates delayed
Coordination problems
Degradation over time







Routing control plane doesn't clearly inform humans that everything is taken care of, and offline zone delays and breaks other metrics with a flood of errors

STPA Hazards

Human Control Action:

~~Not provided~~

Unsafe action

~~Safe but too early~~

~~Safe but too late~~

Wrong sequence

~~Stopped too soon~~

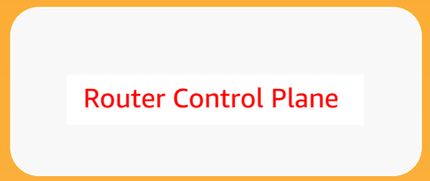
Applied too long

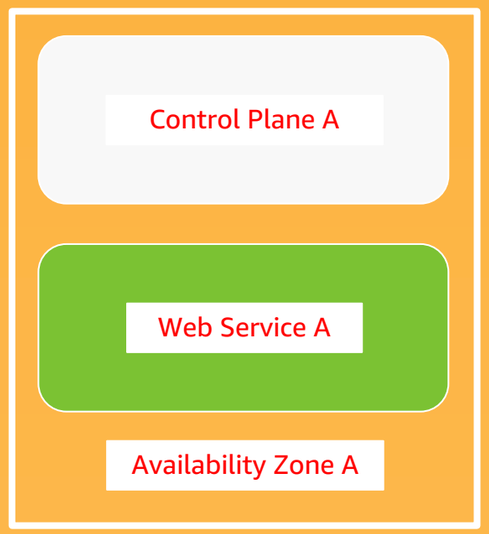
Conflicts

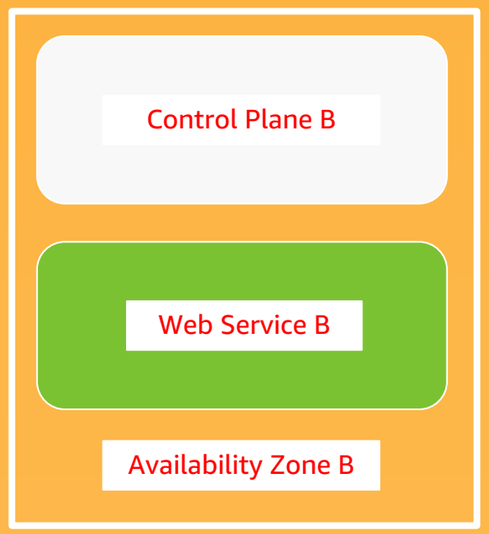
Coordination problems

Degradation over time

Human Controllers should not need to do anything! However they are confused and working separately, try to fix different problems. Some of their tools don't get used often, and are broken or misconfigured to do the wrong thing





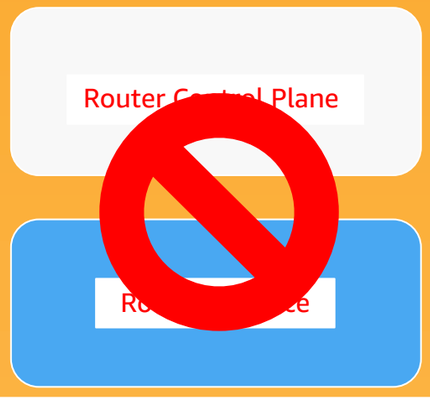


STPA Hazards

Model problems:

Model mismatch
Missing inputs
Missing updates
Updates too rapid
Updates infrequent
Updates delayed
Coordination problems
Degradation over time

Confused human controllers disagree among themselves about whether they need to do something or not, with floods of errors, displays that lag reality by several minutes, and out of date run-books



In-rush of extra traffic from failed zone, and extra work from a cross zone request retry storm causes zones A and B to struggle, and triggers a complete failure of the application. Meanwhile, the routing service also has a retry storm and is impacted

Availability Zone Failover Testing is the Biggest Win

Cross AZ data replication is synchronous, consistent

Failover should be automatic, after few retries, few seconds impact

Regularly test recovery from AZ failure in production

It will NOT work smoothly the first few times you try it…

Get this solid before attempting multi-region

Multi-Region Hazard Analysis

Multi-Region Failover is Getting Easier to do, but...

Cross region data replication is asynchronous, eventually-consistent

Failover is usually manually initiated, with visible downtime

Primary-Secondary failover appropriate for apps like marketplaces

Active-Active failover appropriate for consumer facing services

Harder to regularly test recovery from region failure in production

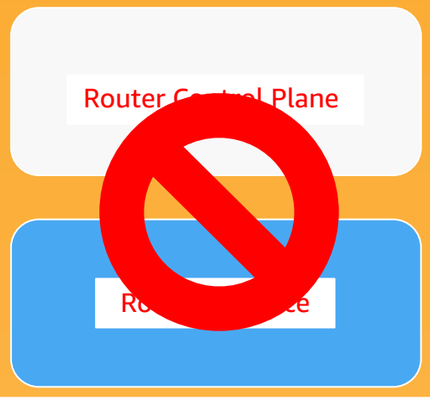
Significantly more expensive and still hard to get right

STPA Hazards

Model problems:

Model mismatch
Missing inputs
Missing updates
Updates too rapid
Updates infrequent
Updates delayed
Coordination problems
Degradation over time

Confused human controllers disagree among themselves about whether they need to do something or not, with floods of errors, displays that lag reality by several minutes, and out of date run-books



Operators redirect traffic too quickly and extra work from a cross region request retry storm causes other regions to struggle, and triggers a complete failure of the application. Meanwhile, the routing service also has a retry storm and is impacted

Good Cloud Resilience Practices

Alert Correlation

- Floods of alerts need to be reduced to actionable insights
- Observability system needs to cope with floods without failing

Retry Storms - Prevent Work Amplification

- Reduce retries to zero except at subsystem entry and exit points
- Reduce timeouts to drop orphaned requests

Good Cloud Resilience Practices

Chaos first

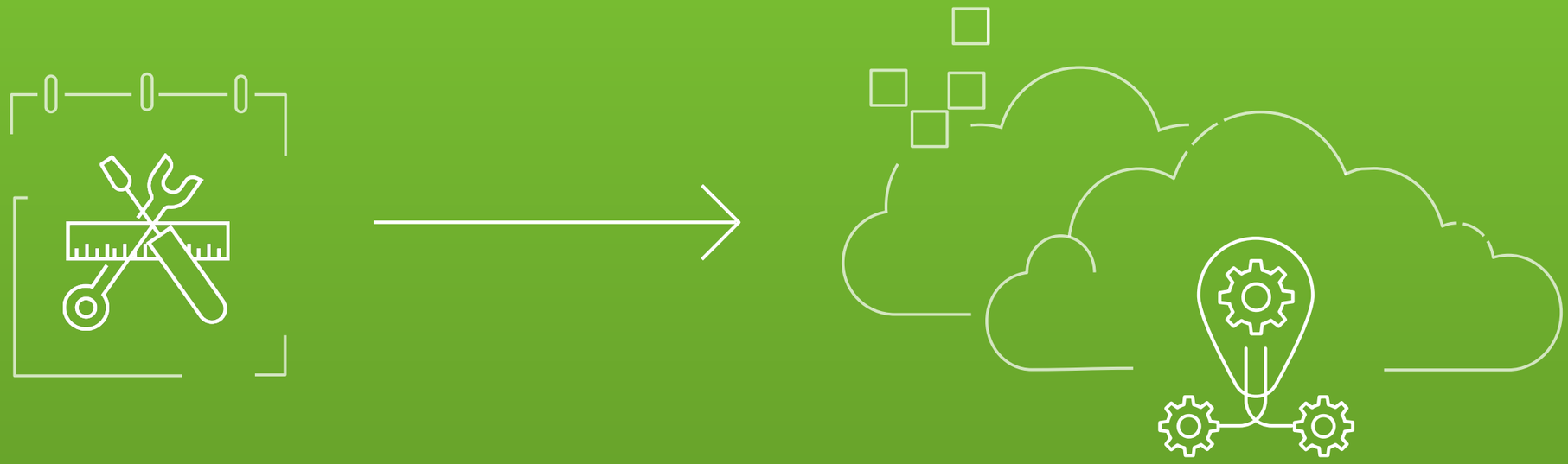
- Build your resilience environment *before* introducing apps to it
- Automated continuous zone (and region) failover testing
- Make it a “badge of honor” to have an app pass the chaos test

Good Cloud Resilience Practices

Continuous Resilience

- Continuous Delivery needs Test Driven Development and Canaries
- Continuous Resilience needs automation in both test and production
- Make failure mitigation into a well tested code path and process
- Call it Chaos Engineering if you like, it’s the same thing…

As datacenters migrate to cloud, fragile and manual disaster recovery processes should be standardized and automated



Testing failure mitigation will move from a scary annual experience to automated **continuous resilience**

Questions?

Paper: Building Mission Critical Financial Services Applications on AWS
By Pawan Agnihotri with contributions by Adrian Cockcroft

Well Architected Guide - Reliability Pillar:
https://docs.aws.amazon.com/wellarchitected/latest/reliability-pillar/welcome.html

Related blog posts by @adrianco

https://medium.com/@adrianco/failure-modes-and-continuous-resilience-6553078caad5

Response time variation: https://dev.to/aws/why-are-services-slow-sometimes-mn3

Retries and timeouts: https://dev.to/aws/if-at-first-you-don-t-get-an-answer-3e85

Source for slide decks at: https://github.com/adrianco/slides

Book recommendations, reading list at: http://a.co/79CGMfB

Adrian Cockcroft

@adrianco

Failing Over Without Falling Over

Adrian Cockcroft

@adrianco

AWS VP Cloud Architecture Strategy