Architectural Patterns of Resilient Distributed Systems

 OMG Strangeloop 2015!!

Ines Sombra

fastly®

@Randommood
ines@fastly.com

$$f(x) = x$$

Globally distributed & highly available

fastly®

Today's Journey

- Why care?
- Resilience literature
- Resilience in industry
- Conclusions

fastly

@randommood

OBLIGATORY DISCLAIMER SLIDE

Things you may see in this talk

- Pugs
- Fast talking
- Life pondering
- Un-tweetable moments
- Rantifestos
- What surprised me this year
- Wedding factoids and trivia

fastly®

@randommood

Why Resilience?

fastly

How can I make a system more resilient?

@randommood

Resilience is the ability of a system to **adapt** or **keep working** when challenges occur

fastly®

@randommood

Defining Resilience

fastly

- Fault-tolerance
- Evolvability
- Scalability
- Failure isolation
- Complexity management

It's what really matters

fastly

@randommood

Resilience in **Literature**

Harvest, Yield, and Scalable Tolerant Systems

Armando Fox
Stanford University
fox@cs.stanford.edu

Eric A. Brewer
University of California at Berkeley
brewer@cs.berkeley.edu

Abstract

The cost of reconciling consistency and state management with high availability is highly magnified by the unprecedented scale and robustness requirements of today's Internet applications. We propose two strategies for improving overall availability using simple mechanisms that scale over large applications whose output behavior tolerates graceful degradation. We characterize this degradation in terms of harvest *and* yield*, and map it directly onto engineering mechanisms that enhance availability by improving*

motivate a broader research program in this area.

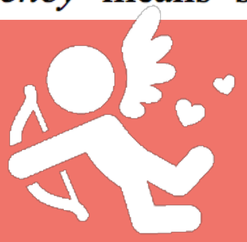
degrading functionality rather than lack of availability of the service as a whole. The approaches were developed in the context of cluster computing, where it is well accepted [22] that one of the major challenges is the nontrivial software engineering required to automate partial-failure handling in order to keep system management tractable.

2. Related Work and the CAP Principle

In this discussion, *strong consistency* means single-

given consumer of the data can always reach *some* replica.

Harvest & Yield Model



Yield

Fraction of successfully answered queries

Close to uptime but more useful because it directly **maps to user experience** (uptime misses this)

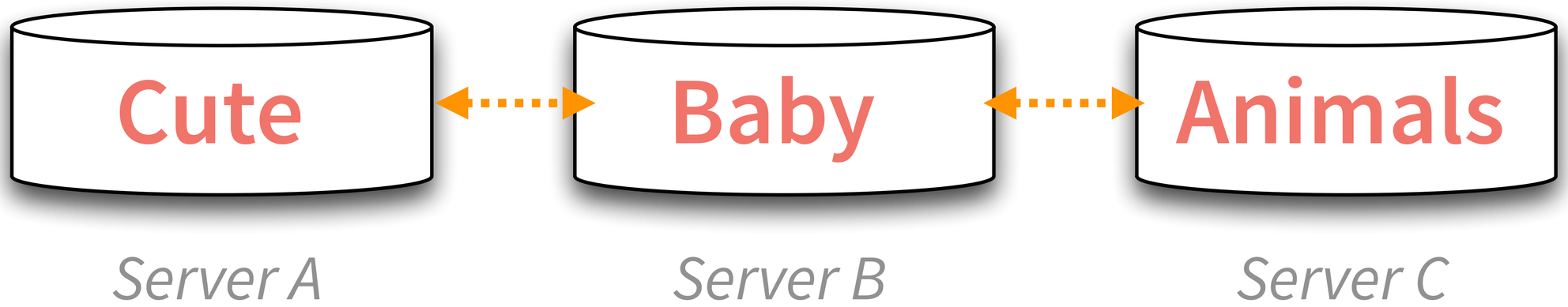
Focus on yield rather than uptime

fastly

Harvest

Fraction of the complete result

$$harvest = \frac{data\ available}{total\ data}$$



fastly®

From Coda Hale's "You can't sacrifice partition tolerance"

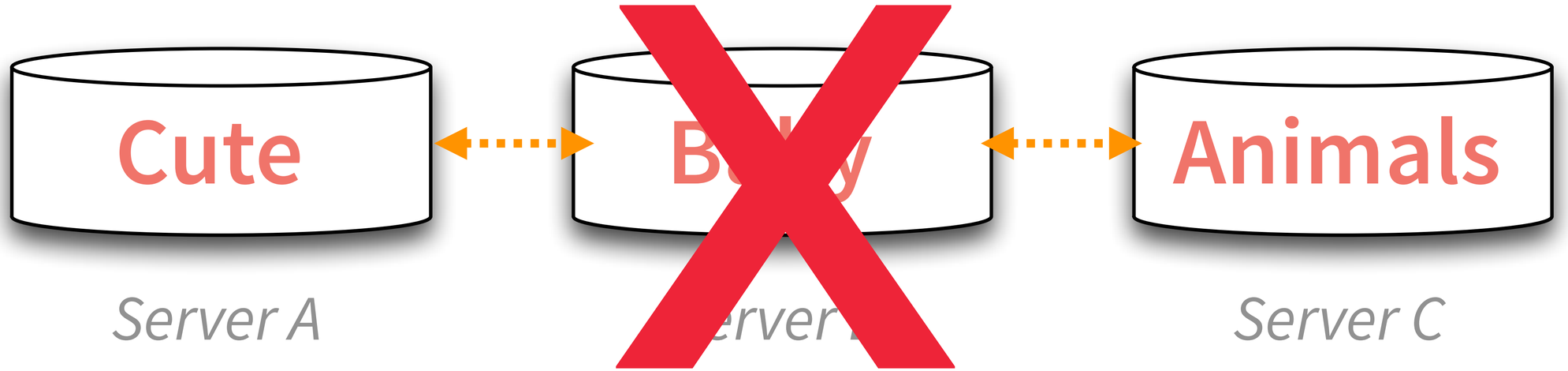
@randommood

Harvest

Fraction of the complete result

$$harvest = \frac{data\ available}{total\ data}$$

66% harvest



fastly®

From Coda Hale's "You can't sacrifice partition tolerance"

@randommood

#1: Probabilistic Availability

Graceful** harvest **degradation under faults

***Randomness** to make the worst-case & average-case the same*

***Replication of high-priority data** for greater harvest control*

*Degrading **results based on client** capability*

fastly®

#2 Decomposition & Orthogonality

Decomposing into subsystems independently intolerant to harvest degradation but the ***application can continue if they fail***

You can only provide ***strong consistency for the subsystems*** *that need it*

Orthogonal mechanisms *(state vs functionality)*

fastly®

“If your system favors yield or harvest is an **outcome of its design**”

Fox & Brewer

fastly®

"Going solid": a model of system dynamics and consequences for patient safety

R Cook, J Rasmussen

Qual Saf Health Care 2005;**14**:130–134. doi: 10.1136/qshc.2003.009530

Rather than being a static property of hospitals and other healthcare facilities, safety is dynamic and often on short time scales. In the past most healthcare delivery systems were loosely coupled—that is, activities and conditions in one part of the system had only limited effect on those elsewhere. Loose coupling allowed the system to buffer many conditions such as short term surges in demand. Modern management techniques and information systems have allowed facilities to reduce inefficiencies in operation. One side effect is the loss of buffers that previously accommodated demand surges. As a result, situations occur in which activities in one area of the hospital become critically dependent on seemingly insignificant events in seemingly distant areas. This tight coupling condition is

current conditions in modern healthcare delivery and the way these conditions may lead to accidents.

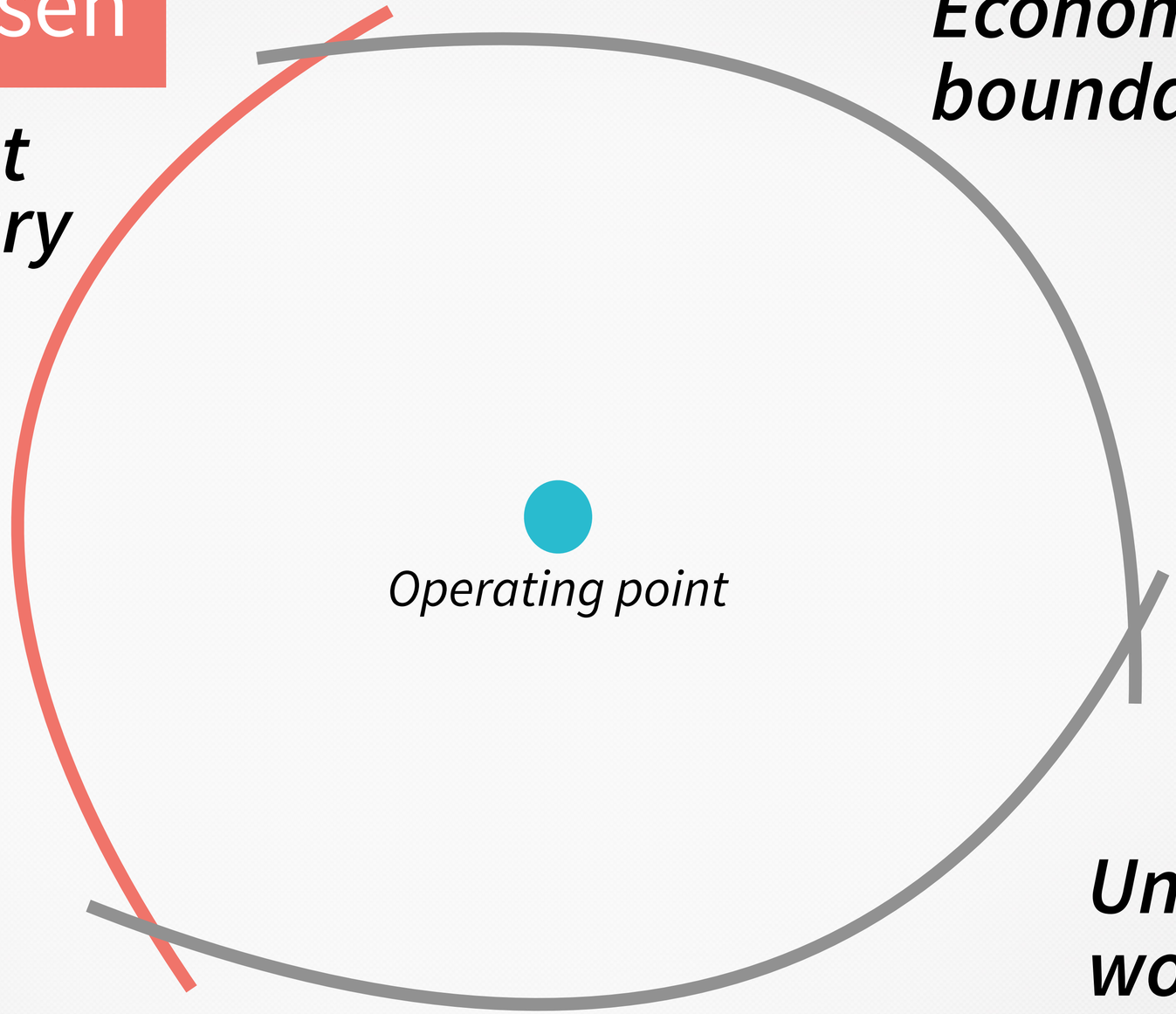
condition. The anesthetic was started because it had become routine to begin surgery in anticipation of resources *becoming* available rather than waiting for them to *be* available. The patient would have required an ICU bed for recovery after the procedure and the practitioners elected to halt the operation when it became apparent that no ICU bed would be available. Similar situations are reported by others (box 1). Such situations create opportunities for incidents and accidents.

We describe this shift in operations as *going solid*. "Going solid" is a nuclear power slang term used to describe a difficult to manage technical situation.[3] Manageable behavior of a steam boiler depends on having both steam and liquid water present in the boiler. When a boiler becomes completely filled with liquid (goes solid), its operating characteristics shift suddenly and

the hospital together so that events in one place have direct implications for the operations of all

Cook & Rasmussen model

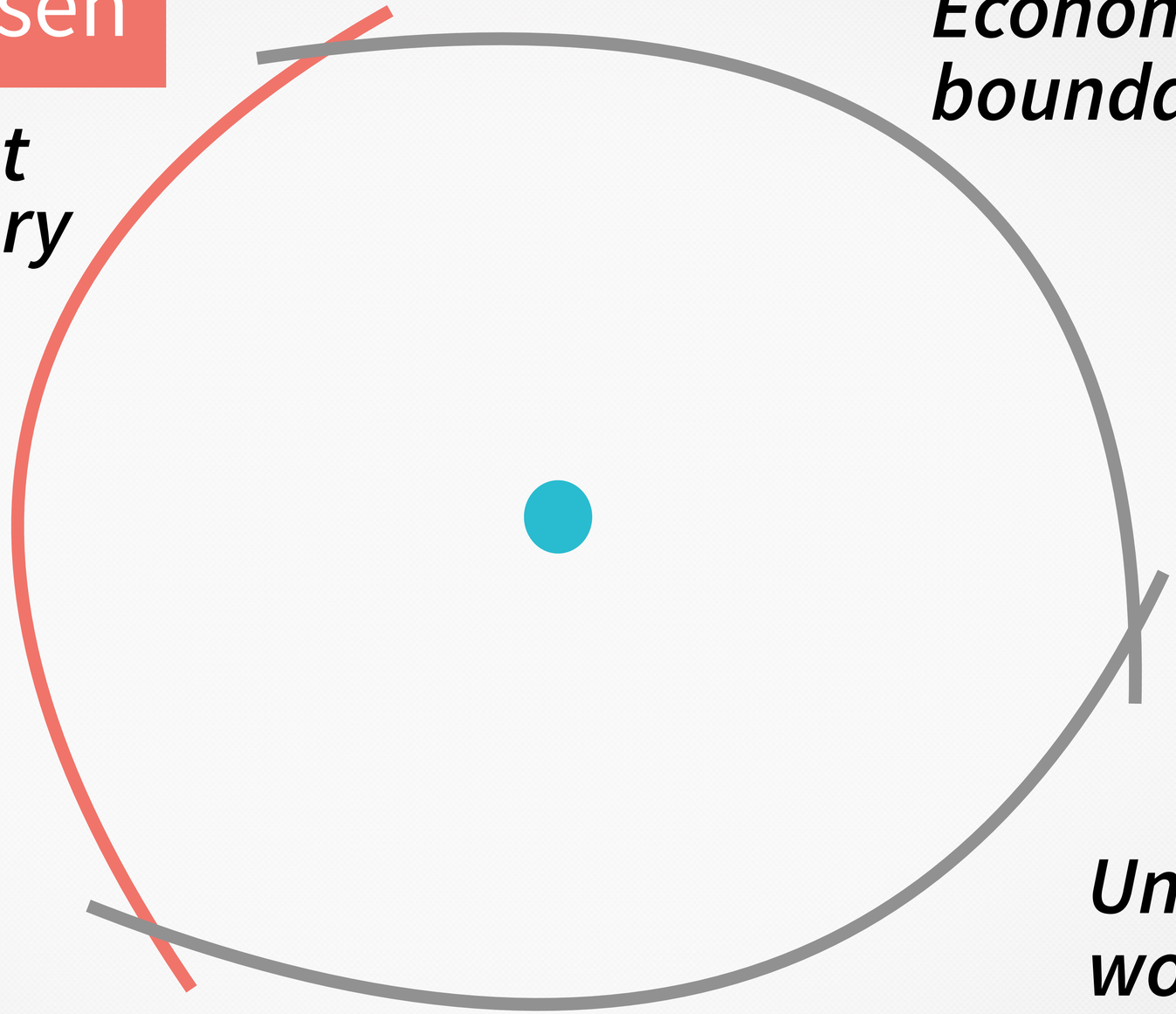
Cook & Rasmussen



Cook & Rasmussen

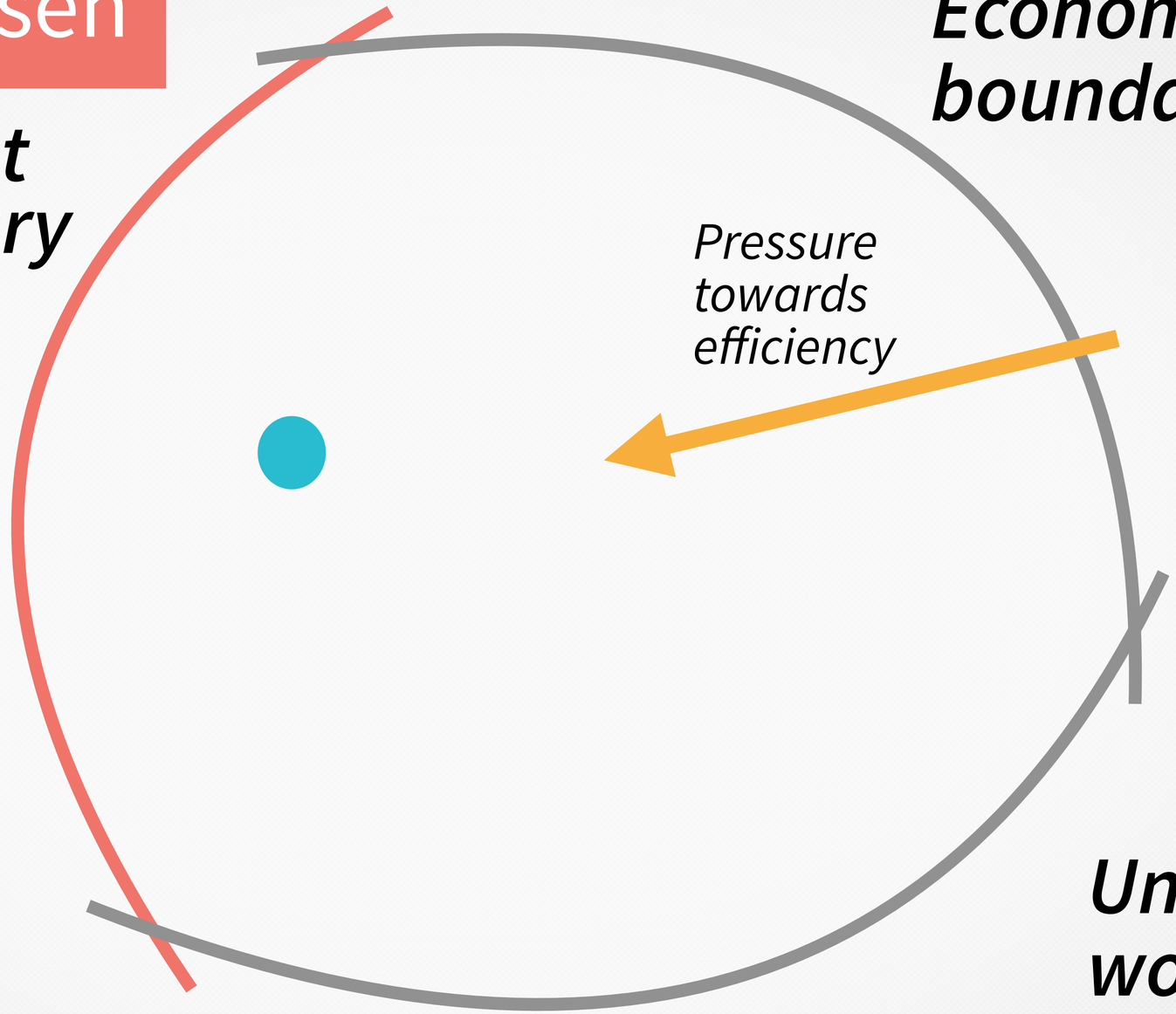
Economic failure boundary

Accident boundary

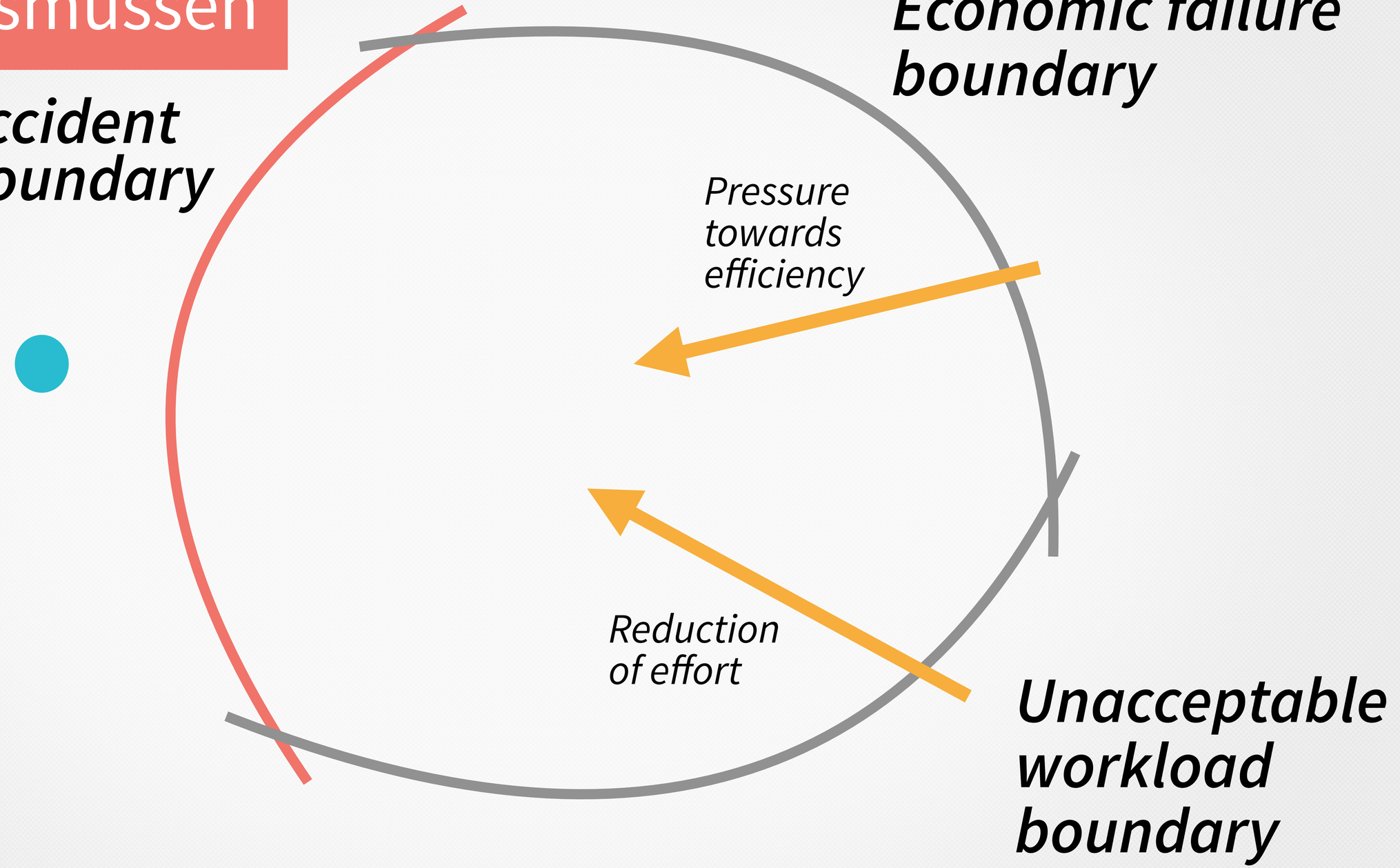


Unacceptable workload boundary

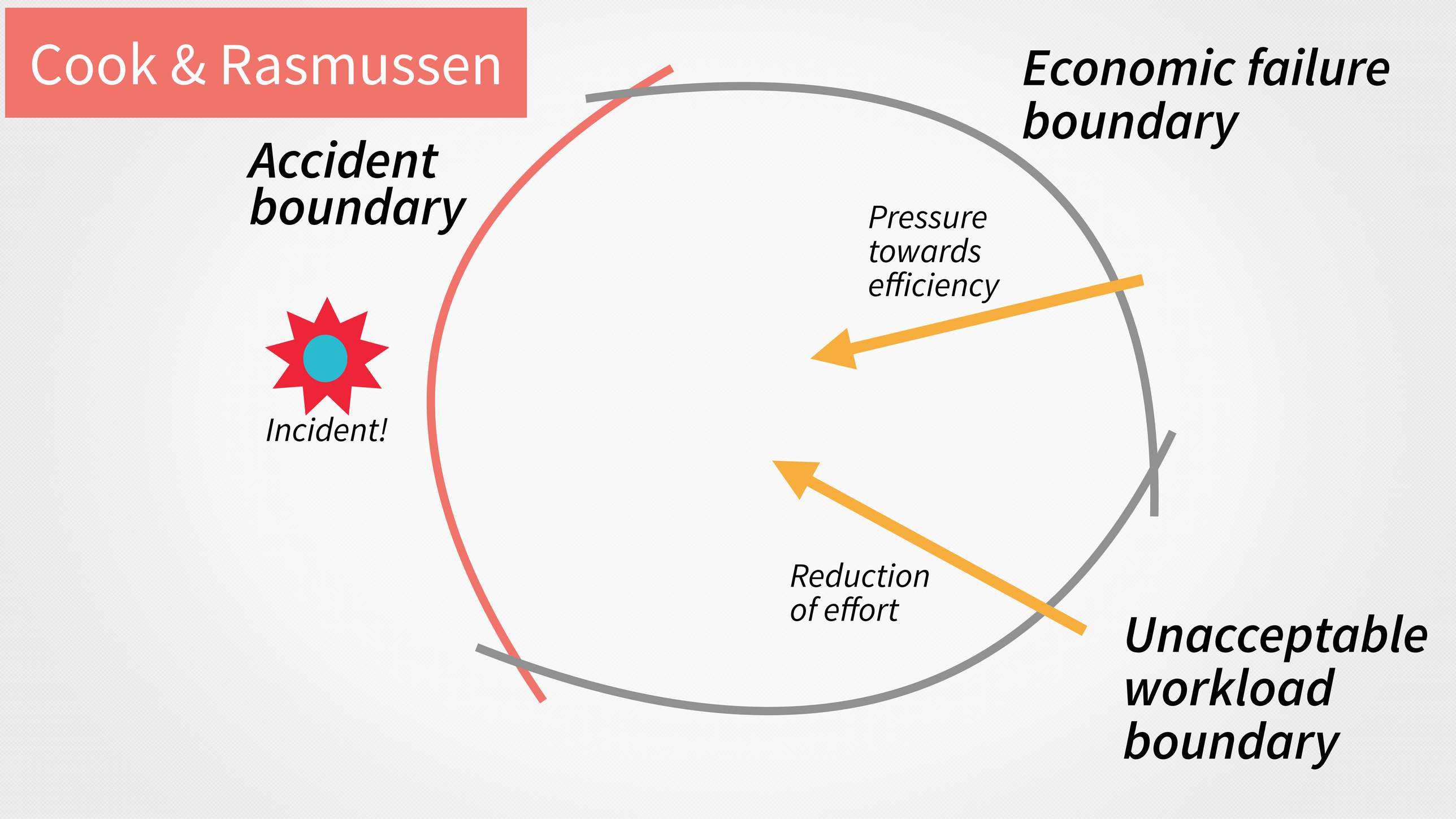
Cook & Rasmussen



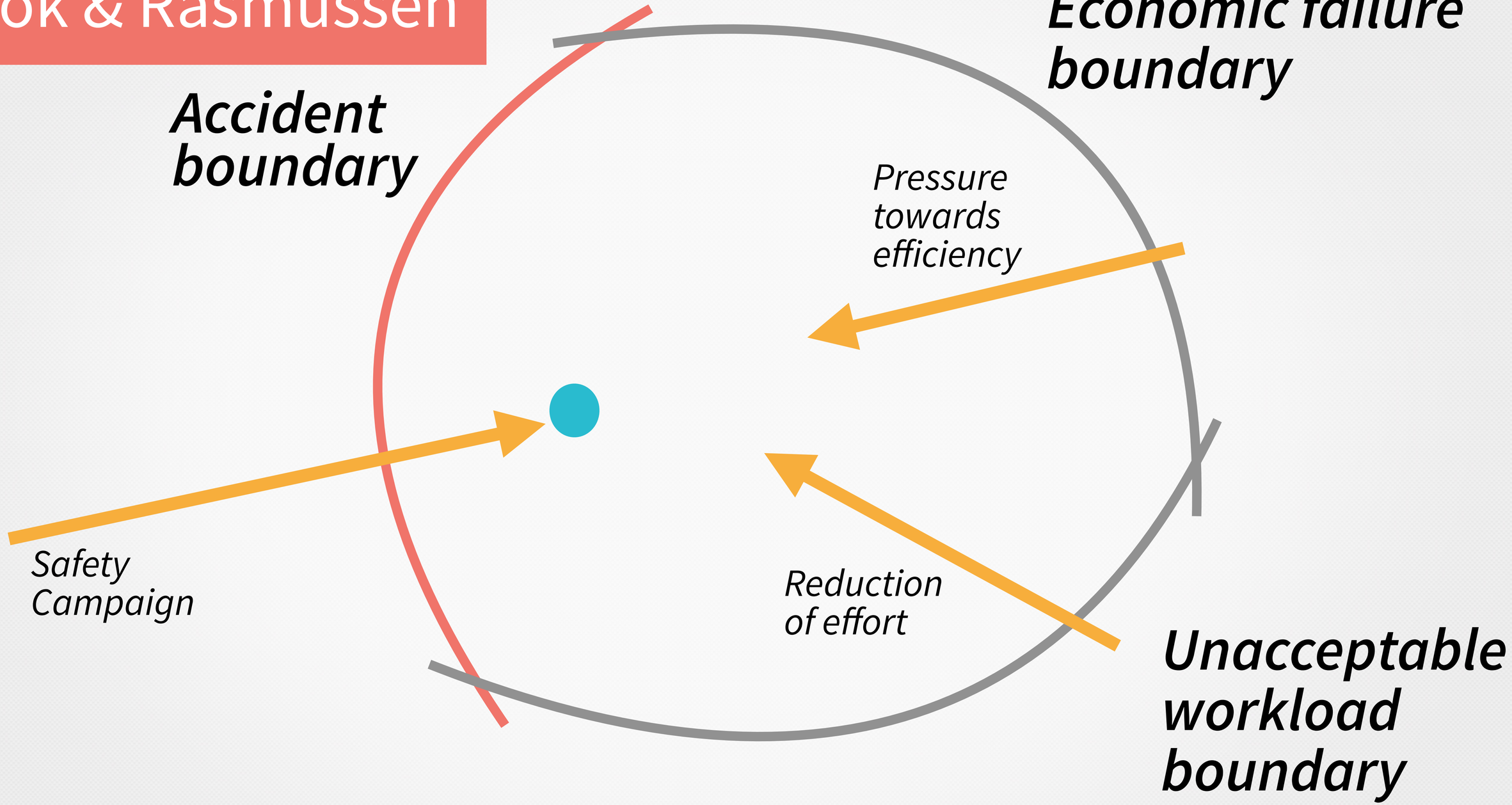
Cook & Rasmussen



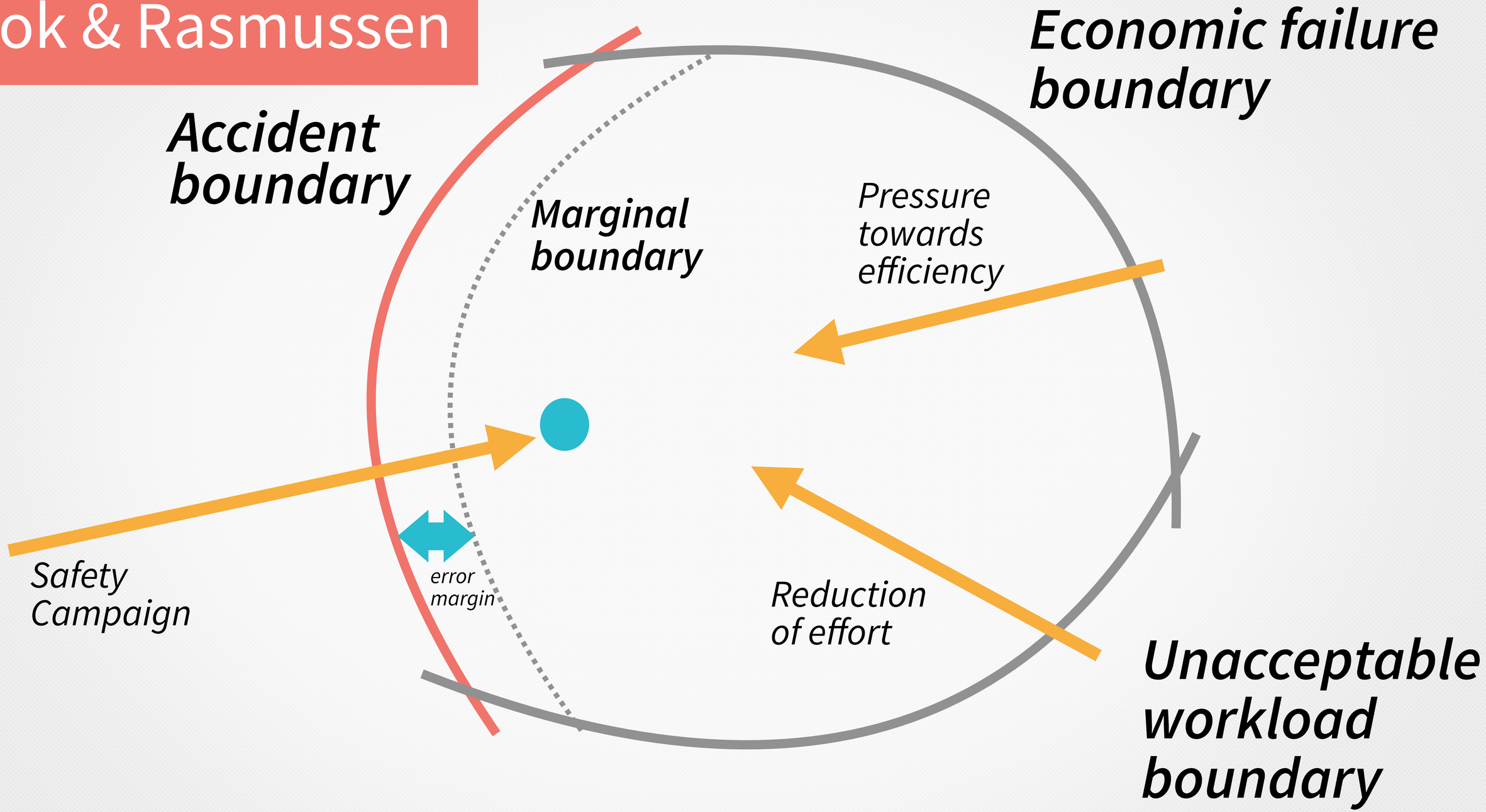
Cook & Rasmussen



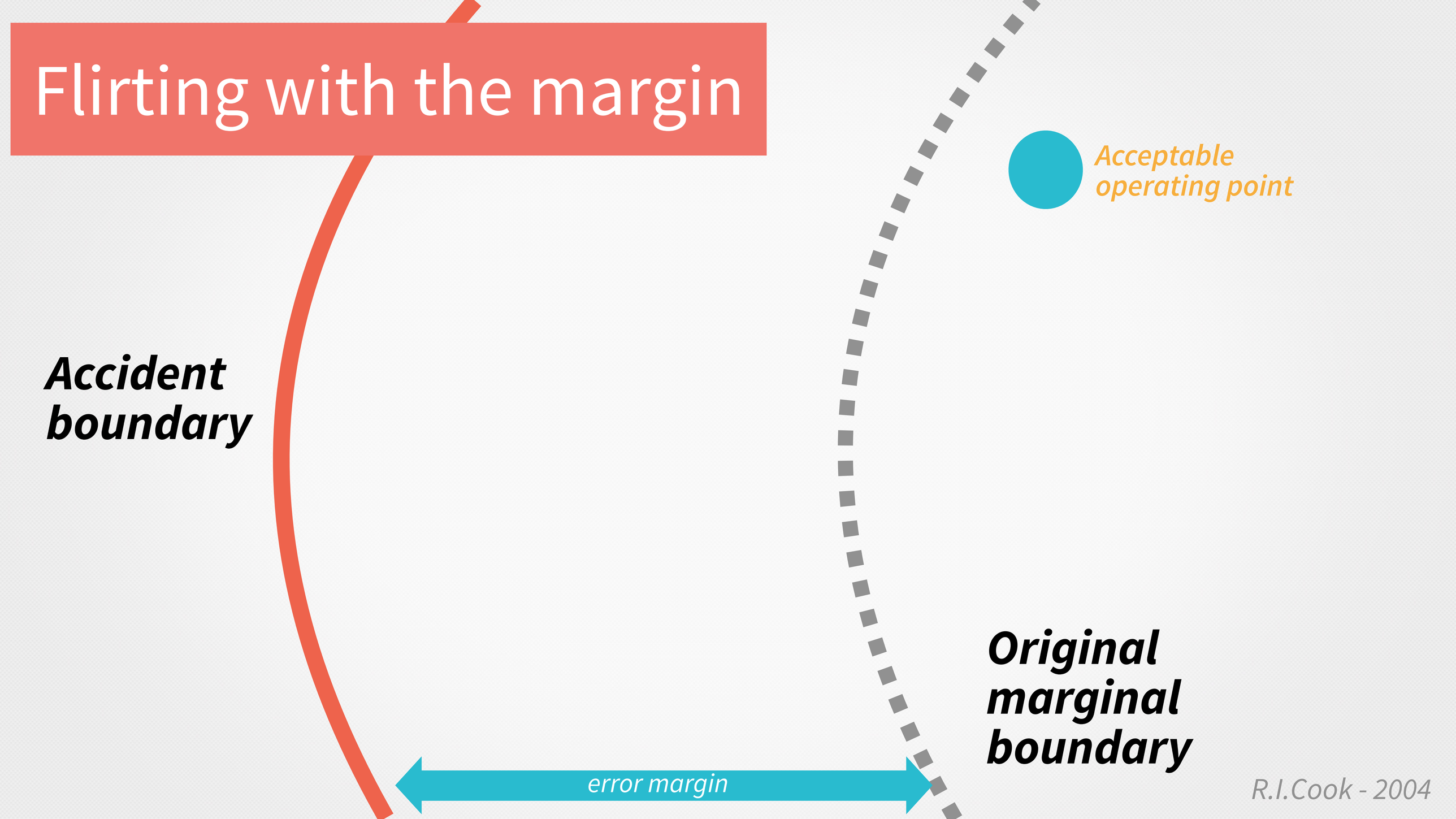
Cook & Rasmussen



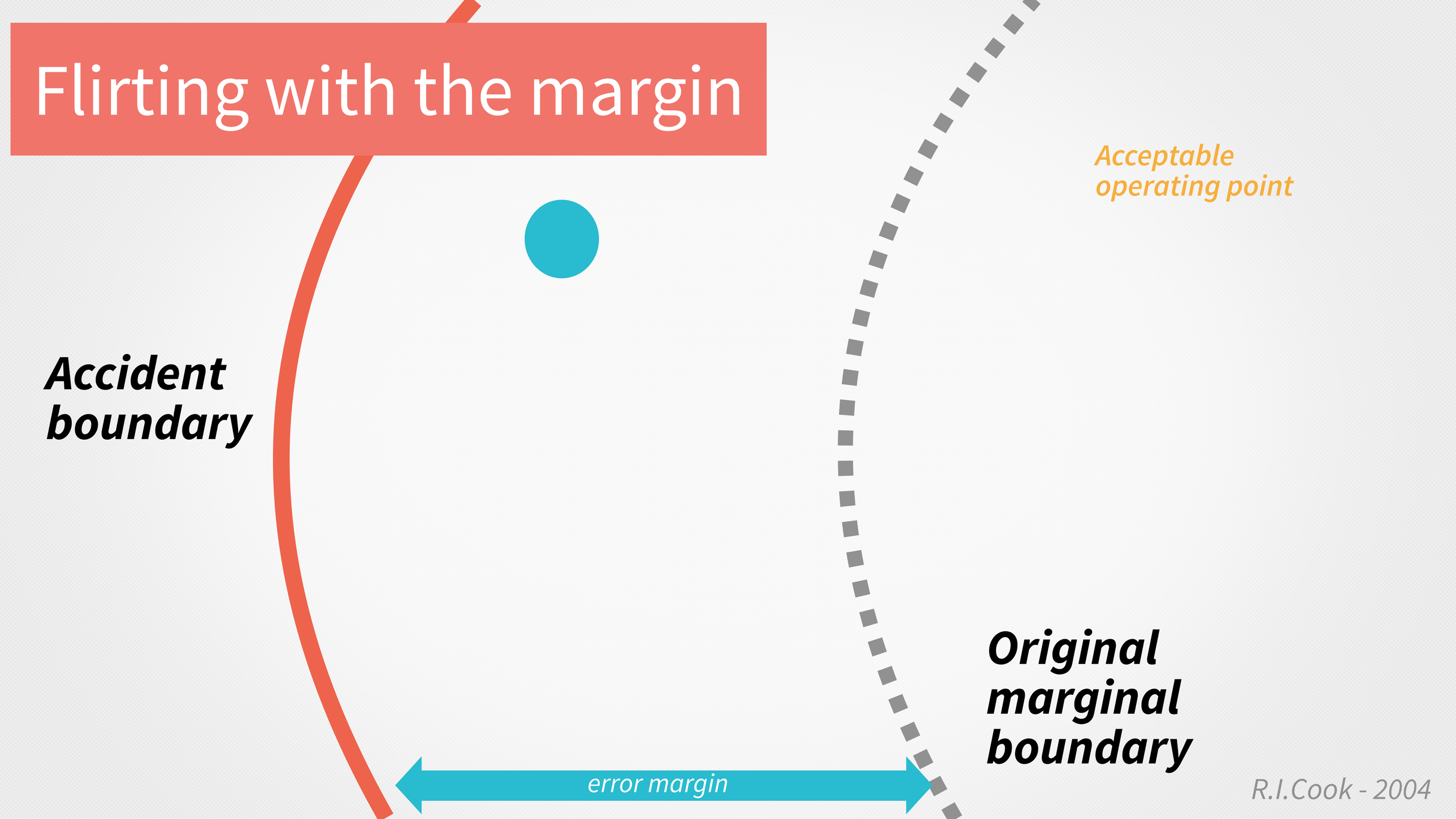
Cook & Rasmussen



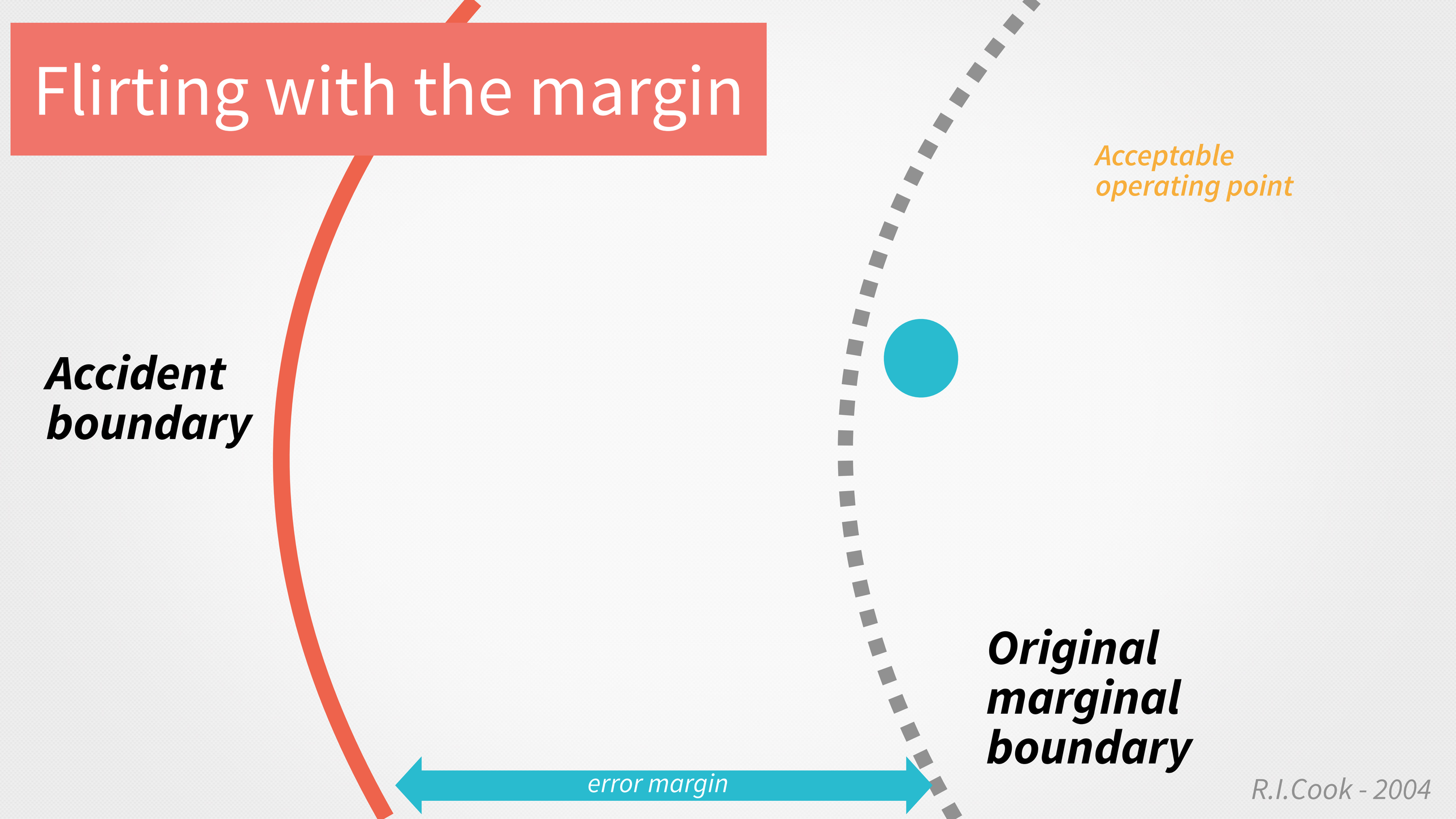
Flirting with the margin



Flirting with the margin

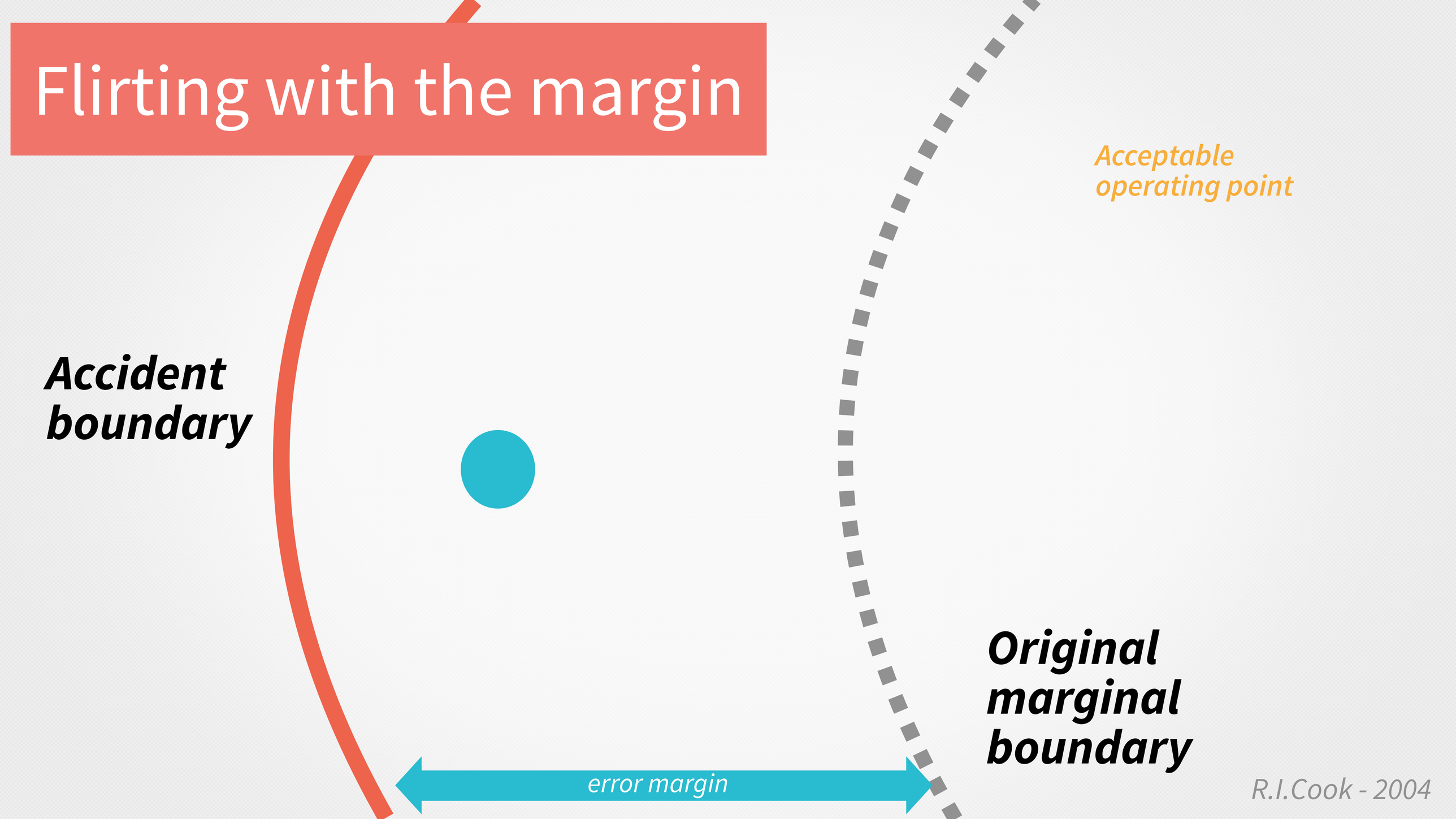


Flirting with the margin

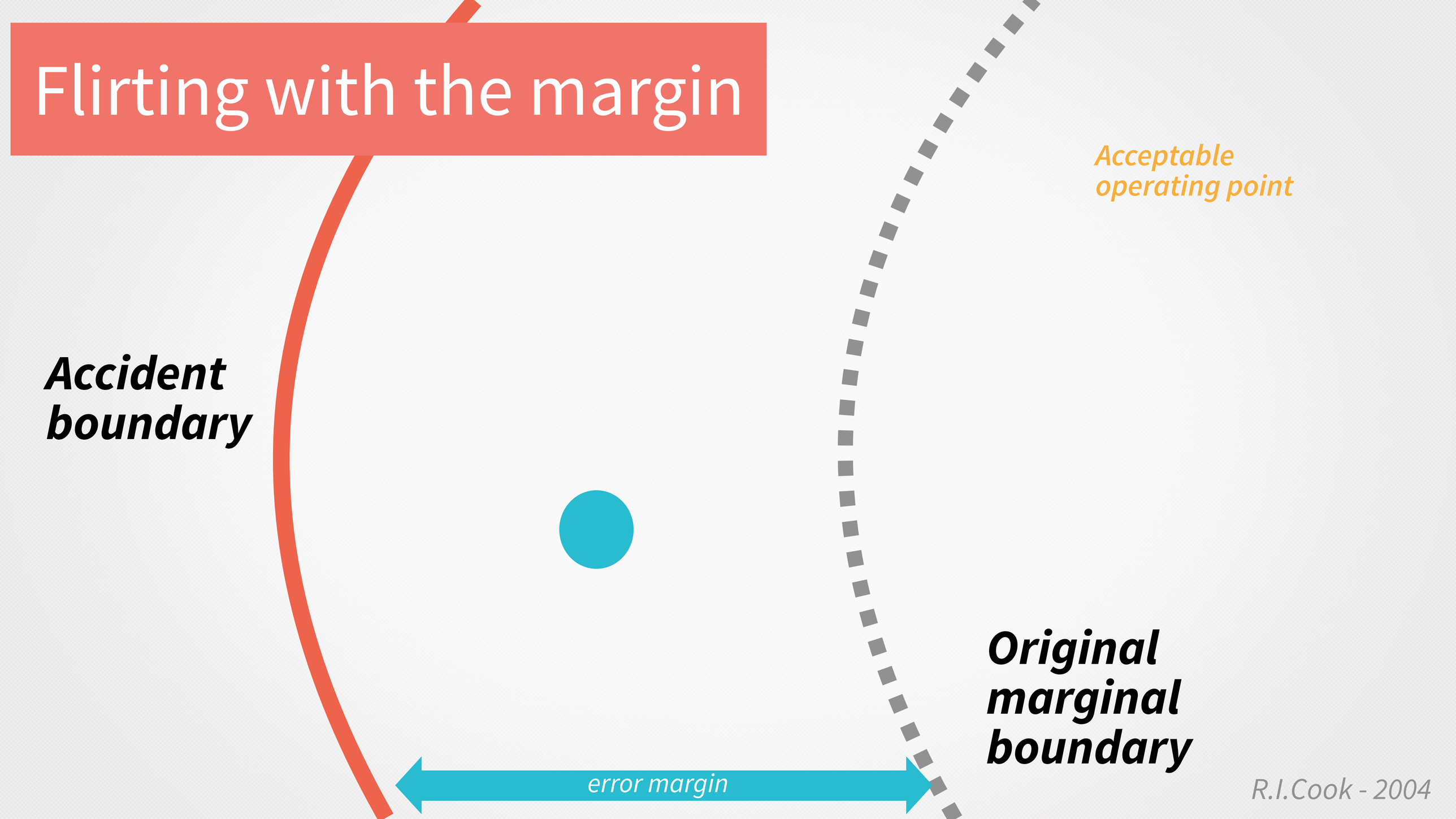


R.I.Cook - 2004

Flirting with the margin

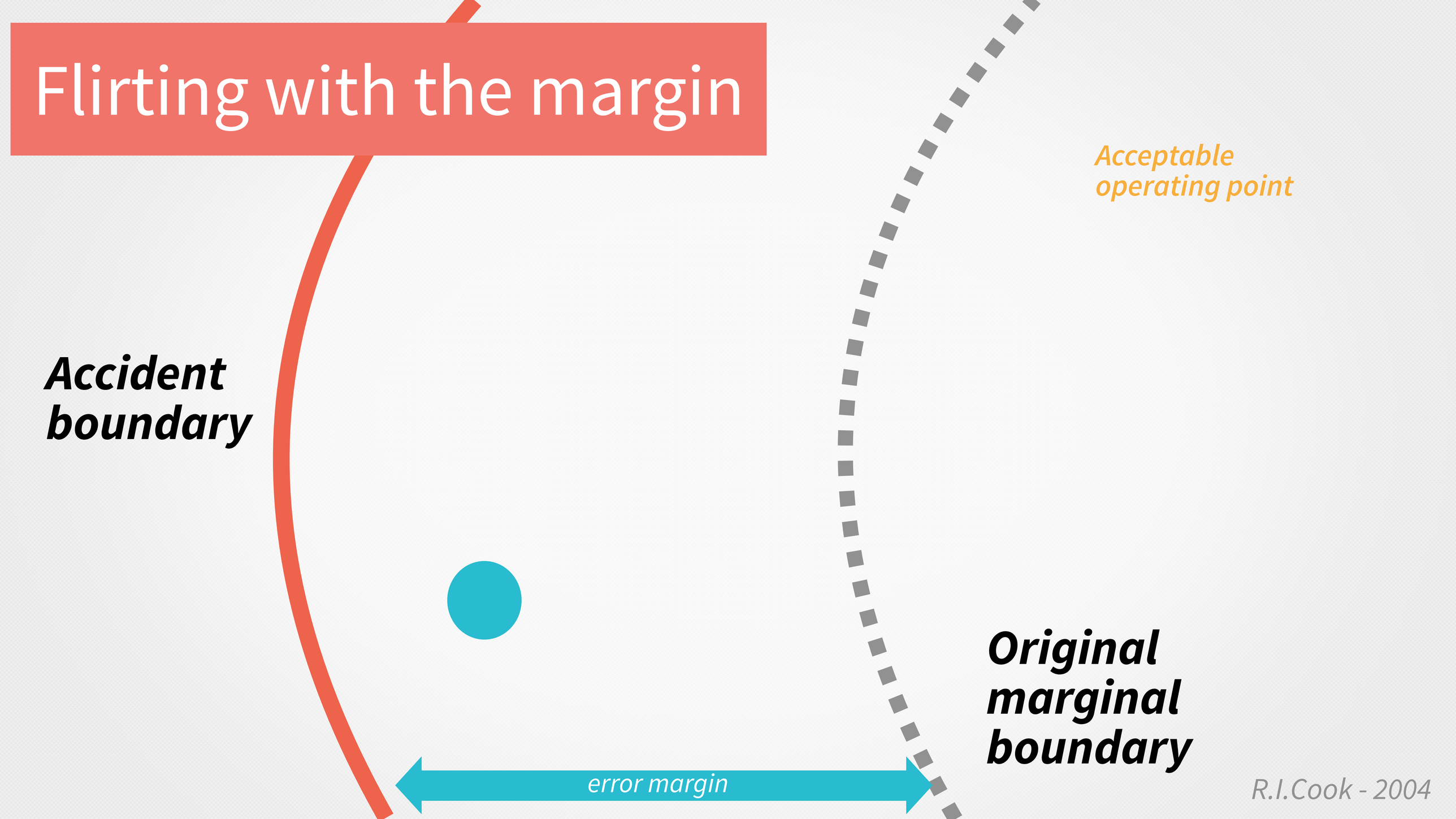


Flirting with the margin

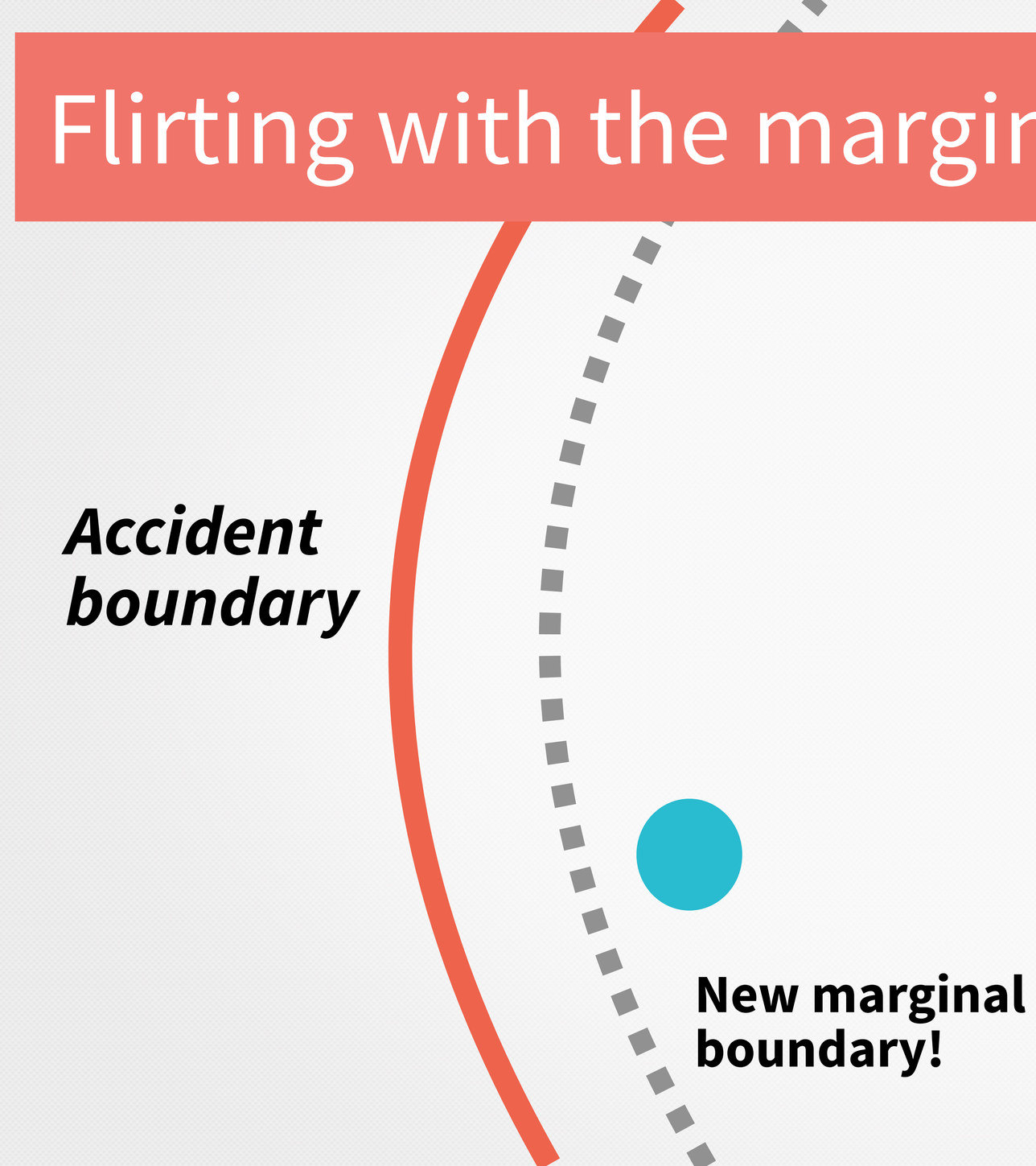


R.I.Cook - 2004

Flirting with the margin



Flirting with the margin



R.I.Cook - 2004

Flirting with the margin

Accident boundary

New marginal boundary!

R.I.Cook - 2004

Insights from Cook's model

Engineering resilience requires a model of safety based on: ***mentoring, responding, adapting,*** *and* ***learning***

System safety is about what can happen, where the operating point actually is, and ***what we do under pressure***

Resilience is operator community focused

fastly®

Engineering system resilience

*Build support for **continuous maintenance***

Reveal control of system to operators

*Know it's going to get moved, replaced, and **used in ways you did not intend***

*Think about **configurations as interfaces***

fastly®

Optimal Design, Robustness, and Risk Aversion

M. E. J. Newman,[1,2] Michelle Girvan,[1,3] and J. Doyne Farmer[1]
[1]*Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, New Mexico 87501*
[2]*Department of Physics, University of Michigan, Ann Arbor, Michigan 48109-1120*
[3]*Department of Physics, Cornell University, Ithaca, New York 14853-2501*
(Received 19 February 2002; published 21 June 2002)

Highly optimized tolerance is a model of optimization in engineered systems, which gives rise to power-law distributions of failure events in such systems. The archetypal example is the highly optimized forest fire model. Here we give an analytic solution for this model which explains the origin of the power laws. We also generalize the model to incorporate risk aversion, which results in truncation of the tails of the power law so that the probability of disastrously large events is dramatically lowered, giving the system more robustness.

DOI: 10.1103/PhysRevLett.89.028301 PACS numbers: 05.65.+b, 05.45.–a, 89.75.–k

In a series of recent papers, Carlson and Doyle [1–3] have proposed a model for designed systems which they call "highly optimized tolerance," or HOT. The fundamental idea behind HOT is that systems designed for high performance naturally organize into highly structured, statistically unlikely states that are robust to perturbations they were designed to handle, yet fragile to rare perturbations and design flaws. As an example, they consider an idealized model of forest fires [2]. In this model, a forester is charged with finding the optimal distribution of the trees

some loss in average system performance can effectively limit the large deviations in the event size distribution so that disasters are rare. We call this variation on the HOT theme "constrained optimization with limited deviations," or COLD. By avoiding total ruin, a COLD design is more robust than a HOT one, even in a world of perfect error-free optimization.

To demonstrate the difference between HOT and COLD, we first revisit the HOT forest fire model. We give an analytic solution for the model which shows that the distri-

Highly Optimized Tolerance: Robustness and Power Laws in Complex Systems

J.M. Carlson
Department of Physics, University of California, Santa Barbara, CA 93106

John Doyle
Control and Dynamical Systems, California Institute of Technology, Pasadena, CA 91125
(October 6, 2013)

We introduce *highly optimized tolerance* (HOT), a mechanism that connects evolving structure and power laws in interconnected systems. HOT systems arise, e.g., in biology and engineering, where design and evolution create complex systems sharing common features, including (1) high efficiency, performance, and robustness to designed-for uncertainties, (2) hypersensitivity to design flaws and unanticipated perturbations, (3) nongeneric, specialized, structured configurations, and (4) power laws. We introduce HOT states in the context of percolation, and contrast properties of the high density HOT states with random configurations near the critical point. While both cases exhibit power laws, only HOT states display properties (1-3) associated with design and evolution.

mental idea behind HOT is that systems designed for high performance naturally organize into highly structured, statistically unlikely states that are robust to perturbations they were designed to handle, yet fragile to rare perturbations and design flaws. As an example, they consider an idealized model of forest fires [2]. In this model, a forester is charged with finding the optimal distribution of the trees

theme "constrained optimization with limited deviations," or COLD. By avoiding total ruin, a COLD design is more robust than a HOT one, even in a world of perfect error-free optimization.

To demonstrate the difference between HOT and COLD, we first revisit the HOT forest fire model. We give an ana-

Building Robust Systems
an essay

Gerald Jay Sussman
Massachusetts Institute of Technology

January 13, 2007

Highly Optimized Tolerance: Robustness and Power Laws in Complex Systems

J.M. Carlson
rsity of California, Santa Barbara, CA 93106

John Doyle
ifornia Institute of Technology, Pasadena, CA 91125

October 6, 2013)

T), a mechanism that connects evolving structure and power
ns arise, e.g., in biology and engineering, where design and
ommon features, including (1) high efficiency, performance,
es, (2) hypersensitivity to design flaws and unanticipated
perturbations, (3) nongeneric, specialized, structured configurations, and (4) power laws. We introduce
HOT states in the context of percolation, and contrast properties of the high density HOT states with
random configurations near the critical point. While both cases exhibit power laws, only HOT states
display properties (1-3) associated with design and evolution.

mental idea behind HOT is that systems designed for high performance naturally organize into highly structured, statistically unlikely states that are robust to perturbations they were designed to handle, yet fragile to rare perturbations and design flaws. As an example, they consider an idealized model of forest fires [2]. In this model, a forester is charged with finding the optimal distribution of the trees

theme "constrained optimization with limited deviations," or COLD. By avoiding total ruin, a COLD design is more robust than a HOT one, even in a world of perfect error-free optimization.

To demonstrate the difference between HOT and COLD, we first revisit the HOT forest fire model. We give an analytic solution for the model which shows that the distri-

Highly Optimized Tolerance: Robustness and Power Laws in Complex Systems

Building Robust Systems
an essay

Gerald Jay Sussman
Massachusetts Institute of Technology

January 13, 2007

perturbations, (3) nongeneric
HOT states in the context of
random configurations near t
display properties (1-3) assoc
highly optimized tolera
mental idea behind HOT is tha
performance naturally organiz
tistically unlikely states that
they were designed to handle
bations and design flaws. As a
idealized model of forest fires [2]. In this model, a forester
is charged with finding the optimal distribution of the trees

Computer Immunology

Mark Burgess – Oslo College

ABSTRACT

sent day computer systems are fragile and unreliable. Human beings are involved in the repair of computer systems at every stage in their operation. This level of human ent will be impossible to maintain in future. Biological and social systems of ble and greater complexity have self-healing processes which are crucial to their It will be necessary to mimic such systems if our future computer systems are to n a complex and hostile environment. This paper describes strategies for future research and summarizes concrete measures for the present, building upon existing software systems.

Autonomous systems

We dance for our computers. Every error, every problem that has to be diagnosed schedules us to do work on the system's behalf. Whether the root cause of the errors is faulty programming or simply a lack of foresight, human intervention is required in computing systems with a regularity which borders on the embarrassing. Operating system design is about the sharing of resources amongst a set of tasks: additional tasks

Surprisingly most system administration models which are developed and sold today are entirely based either on the idea of interaction between administrator and either user or machine; or on the cloning of existing systems. We see user graphical user interfaces of increasing complexity, allowing us to see the state of disarray with ever greater ulcer-provoking clarity, but seldom do we find any noteworthy degree of autonomy. In other words administrators are being placed

we first revisit the HOT forest fire model. We give an ana-
lytic solution for the model which shows that the distri-

Highly Optimized Tolerance: Robustness and Power Laws in Complex Systems

Building Robust Systems
an essay

Gerald Jay Sussman
Massachusetts Institute of Technology

January 13, 2007

Computer Immunology

Mark Burgess – Oslo College

ABSTRACT

sent day computer systems are fragile and unreliable. Human beings are involved in the repair of computer systems at every stage in their operation. This level of human ent will be impossible to maintain in future. Biological and social systems of ble and greater complexity have self-healing processes which are crucial to their It will be necessary to mimic such systems if our future computer systems are to n a complex and hostile environment. This paper describes strategies for future research and summarizes concrete measures for the present, building upon existing software systems.

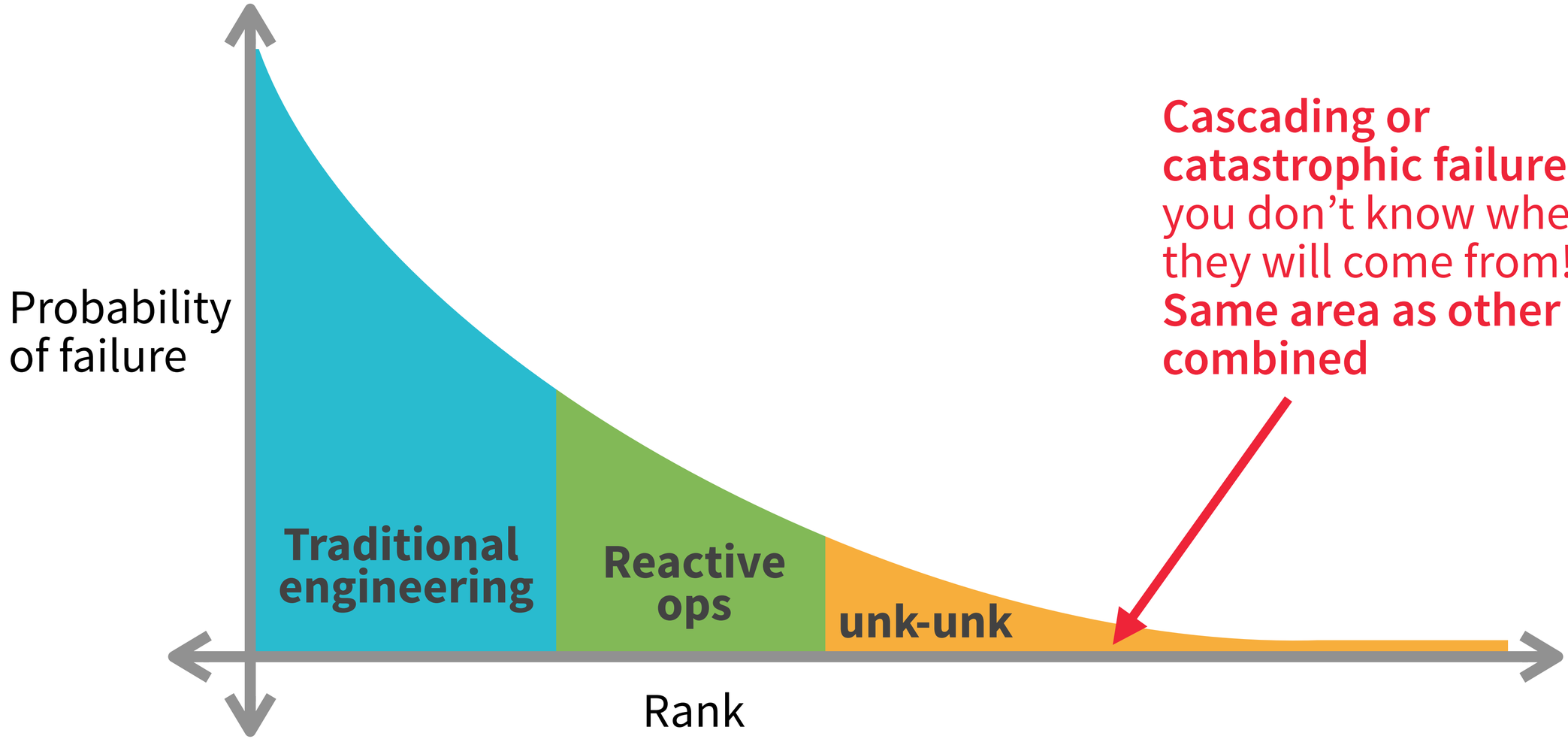
Autonomous systems

We dance for our computers. Every error, every problem that has to be diagnosed schedules us to do

Surprisingly most system administration models which are developed and sold today are entirely based either on the idea of interaction between administrator

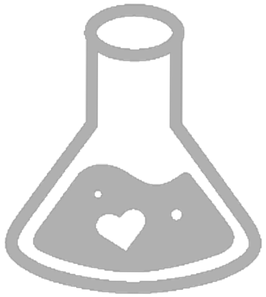
Borrill's model

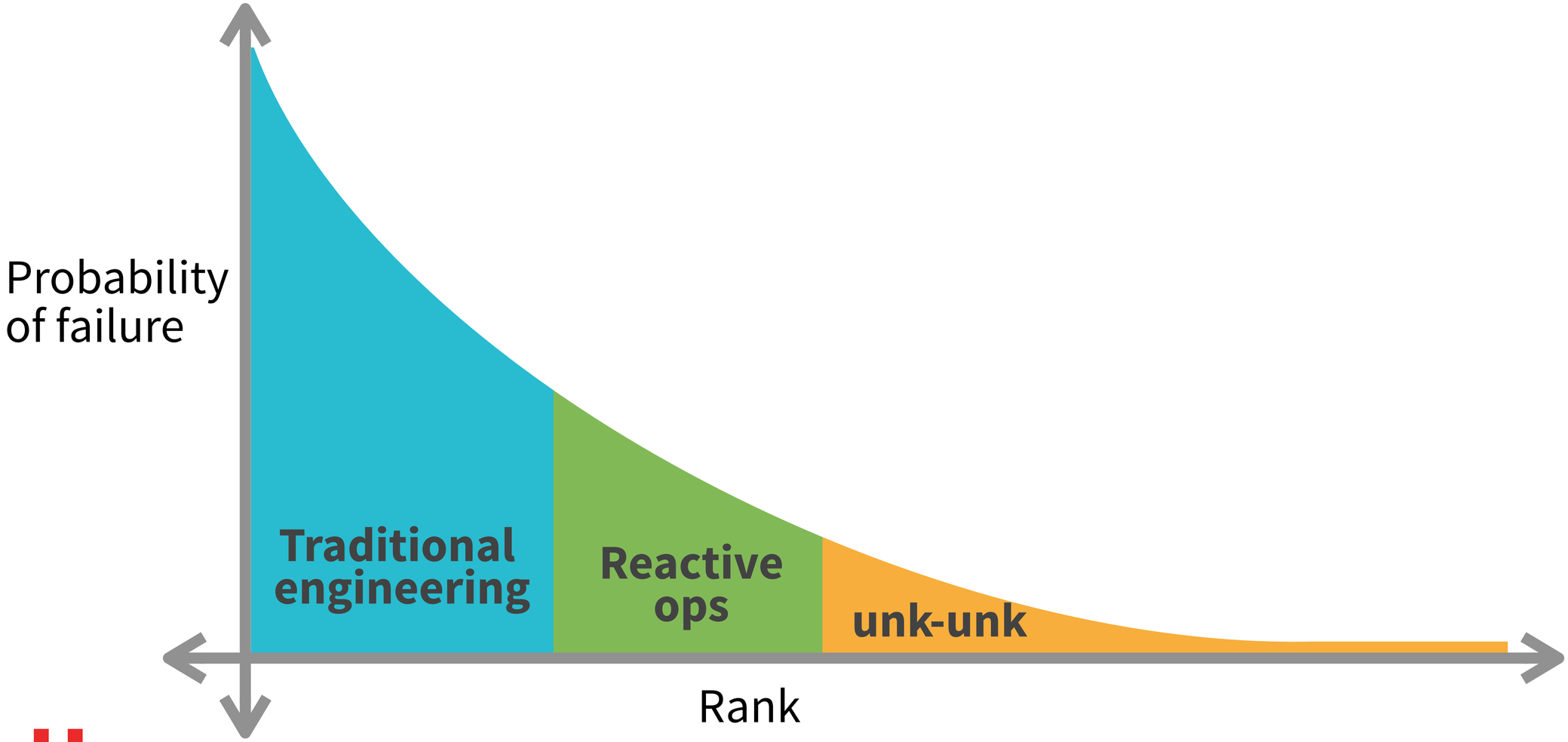
A system's complexity



fastly®

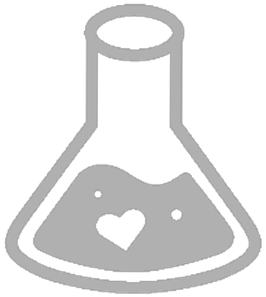
Failure areas need != strategies

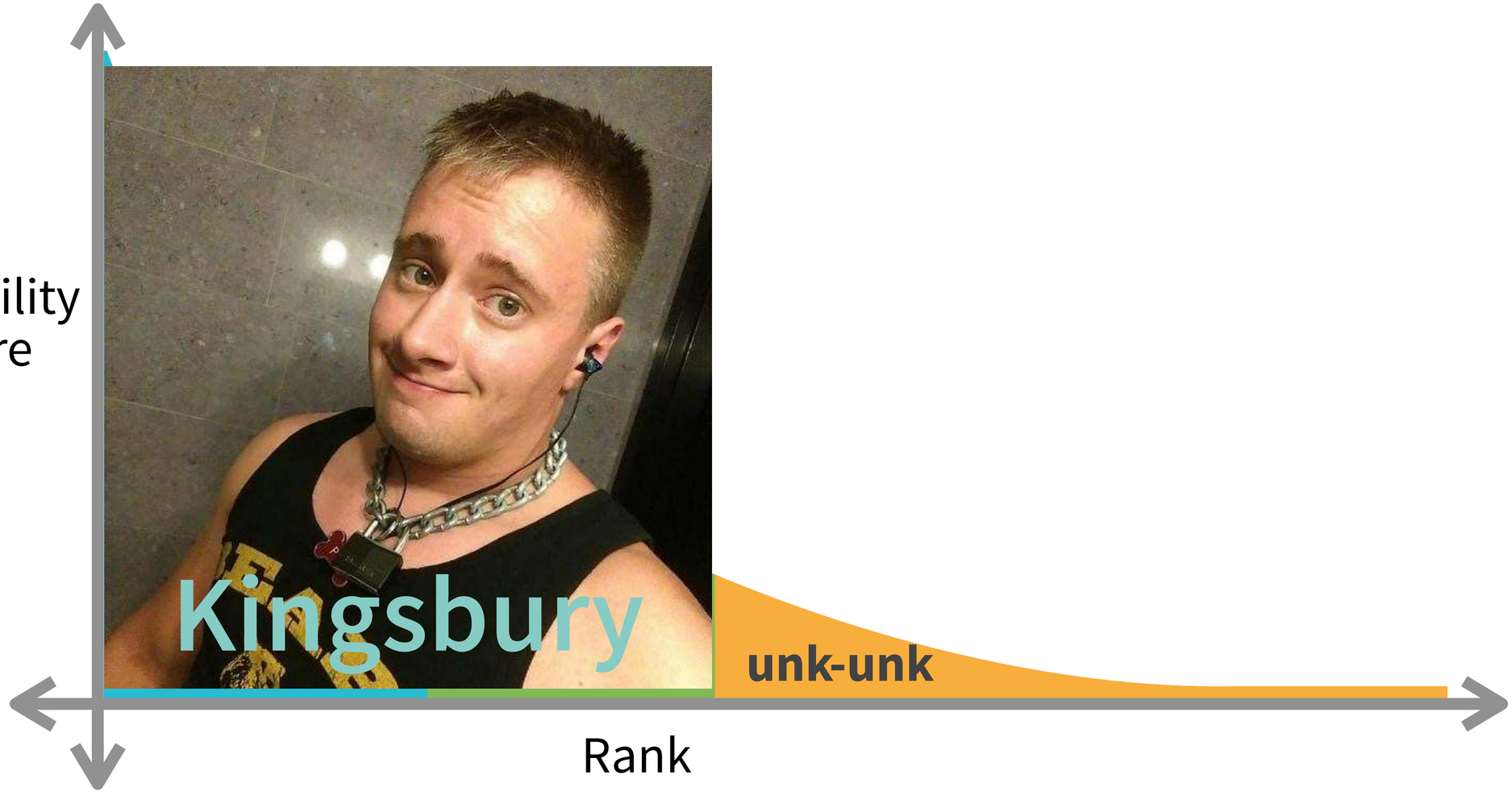




fastly®

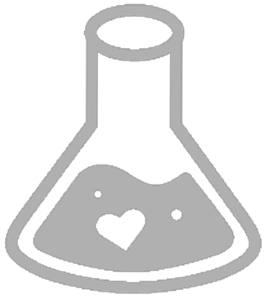
Failure areas need != strategies

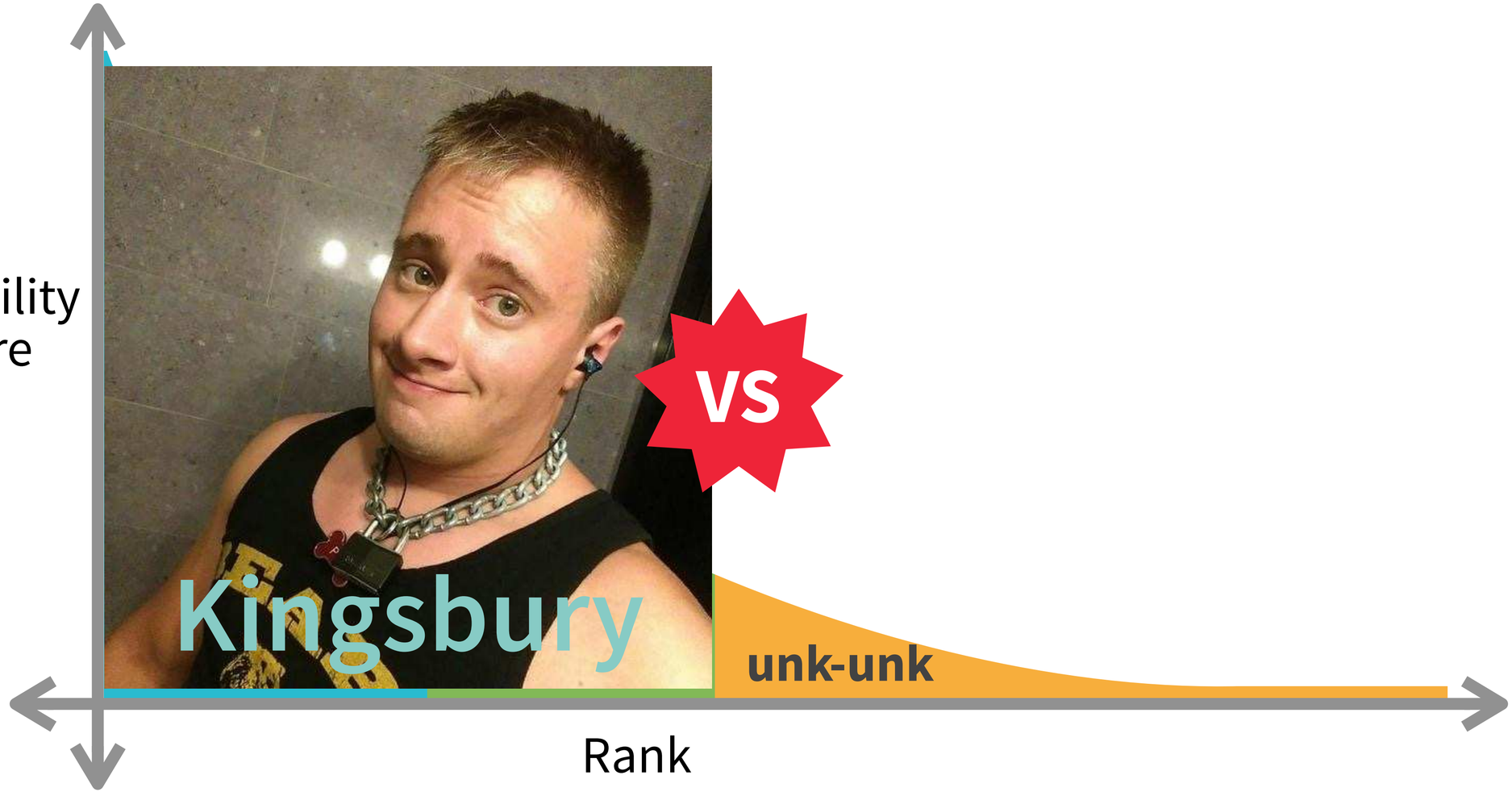




fastly®

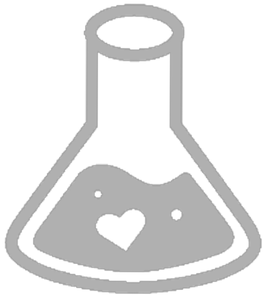
Failure areas need != strategies





fastly®

Failure areas need != strategies



fastly®

Strategies to build resilience

Classical engineering	Reactive Operations	Unknown-Unknown
Code standards	*Hazard inventories*	*System verification*
Programming patterns	*Redundancies*	*Formal methods*
Testing (full system!)	*Feature flags*	*Fault injection*
Metrics & monitoring	*Dark deploys*	
Convergence to good state	*Runbooks & docs*	The goal is to **build failure domain independence**
	Canaries	

“Thinking about building system resilience using a **single discipline is insufficient**. We need different strategies”

Borrill

Wedding Trivia!!!

Four Tier Fondant Wedding Cake, Fake Wedding Cake

$160.00+ USD

Ask a Question

Quantity

1

Options

- ✓ Select an option
- No Cut-Out Slice [$160.00]
- Cut-Out Slice [$170.00]

fastly

@randommood

Resilience in Industry

NETFLIX

fastly®

@randommood

Fault Tolerance in a High Volume, Distributed System

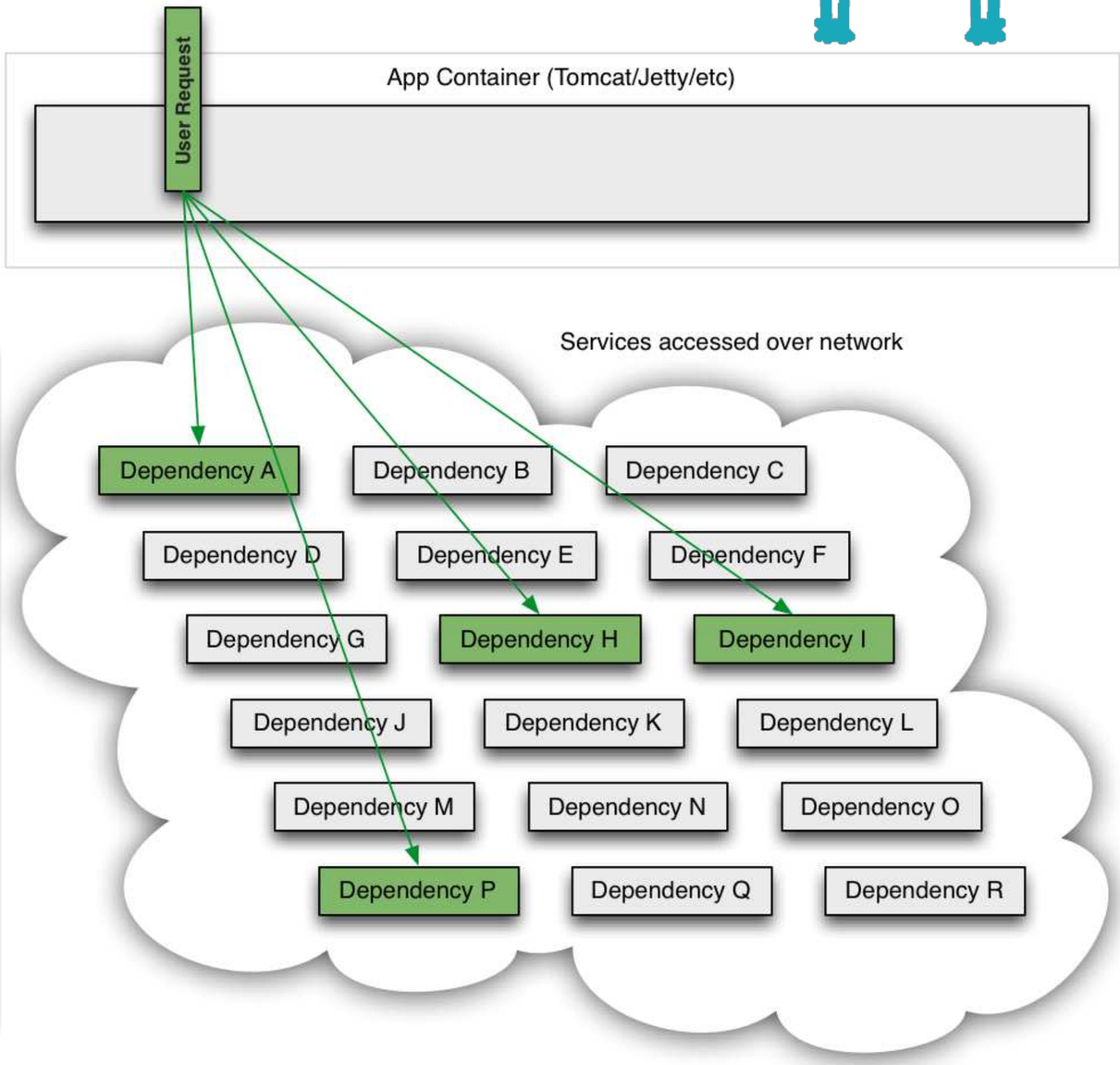
by Ben Christensen

In an earlier post by Ben Schmaus, we shared the principles behind our
that post, Ben discusses how the Netflix API interacts with dozens of sys
architecture, which makes the API inherently more vulnerable to any sys
it in the stack. The rest of this post provides a more technical deep-dive i
isolate failure, shed load and remain resilient to failures.

API **inherently more vulnerable** to any system failures or latencies in the stack

Without fault tolerance: 30 dependencies w 99.99% uptime could result in **2+ hours of downtime per month!**

Leveraged client libraries



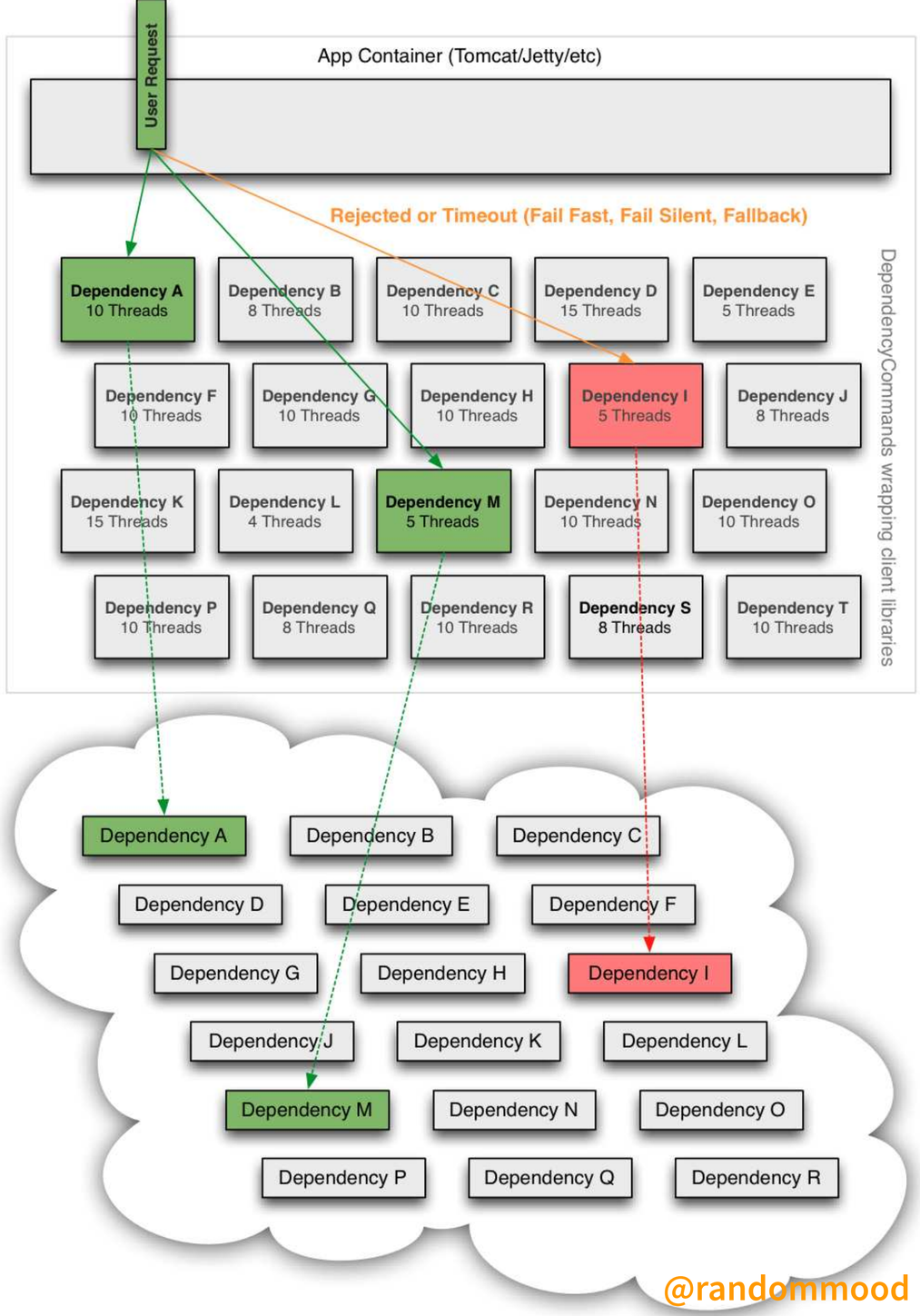
Netflix's resilient patterns

Aggressive network timeouts & retries. Use of Semaphores.

Separate threads on per-dependency thread pools

Circuit-breakers to relieve pressure in underlying systems

Exceptions cause app to shed load until things are healthy



fastly

@randommood

We went on a diet just like you!

Google

fastly

@randommood

The Chubby lock service for loosely-coupled distributed systems

Mike Burrows, *Google Inc*.

Abstract

We describe our experiences with the Chubby lock service, which is intended to provide coarse-grained locking as well as reliable (though low-volume) storage for a loosely-coupled distributed system. Chubby provides an interface much like a distributed file system with advisory locks, but the design emphasis is on availability and reliability, as opposed to high performance. Many instances of the service have been used for over a year, with several of them each handling a few tens of thousands of clients concurrently. The paper describes the initial design and expected use, compares it with actual use, and explains how the design had to be modified to accommodate the differences.

1 Introduction

This paper describes a *lock service* called Chubby. It is example, the Google File System [7] uses a Chubby lock to appoint a GFS master server, and Bigtable [3] uses Chubby in several ways: to elect a master, to allow the master to discover the servers it controls, and to permit clients to find the master. In addition, both GFS and Bigtable use Chubby as a well-known and available location to store a small amount of meta-data; in effect they use Chubby as the root of their distributed data structures. Some services use locks to partition work (at a coarse grain) between several servers.

Before Chubby was deployed, most distributed systems at Google used *ad hoc* methods for primary election (when work could be duplicated without harm), or required operator intervention (when correctness was essential). In the former case, Chubby allowed a small saving in computing effort. In the latter case, it achieved a significant improvement in availability in systems that no longer required human intervention on failure.

Readers familiar with distributed computing will rec-

Key insights from Chubby

Library vs service? ***Service*** *and* ***client library*** ***control*** *+ storage of* ***small data files*** *with restricted operations*

Engineers don’t plan for: availability, consensus, primary elections, failures, their own bugs, operability, or the future. They also don’t understand Distributed Systems

fastly®

Key insights from Chubby

*Centralized services are **hard to construct** but you can dedicate effort into architecting them well and making them failure-tolerant*

Restricting user behavior increased resilience

Consumers of your service are part of your UNK-UNK scenarios

fastly®

And the family arrives!

fastly®

Key insights from Truce

O'REILLY® Velocity Web Performance & Operations CONFERENCE

Evolution of our purging system from v1 to v3

Used Bimodal Multicast (Gossip protocol) to provide extremely fast purging speed

Design concerns & system evolution

fastly®

@randommood

Key insights from NetSys

Faild allows us to fail & recover hosts via MAC-swapping and ECMP on switches

Do **immediate** or gradual **host failure & recovery**

Watch Joao's talk SREcon15 03.16.15-03.17.15 | SANTA CLARA, CA

fastly®

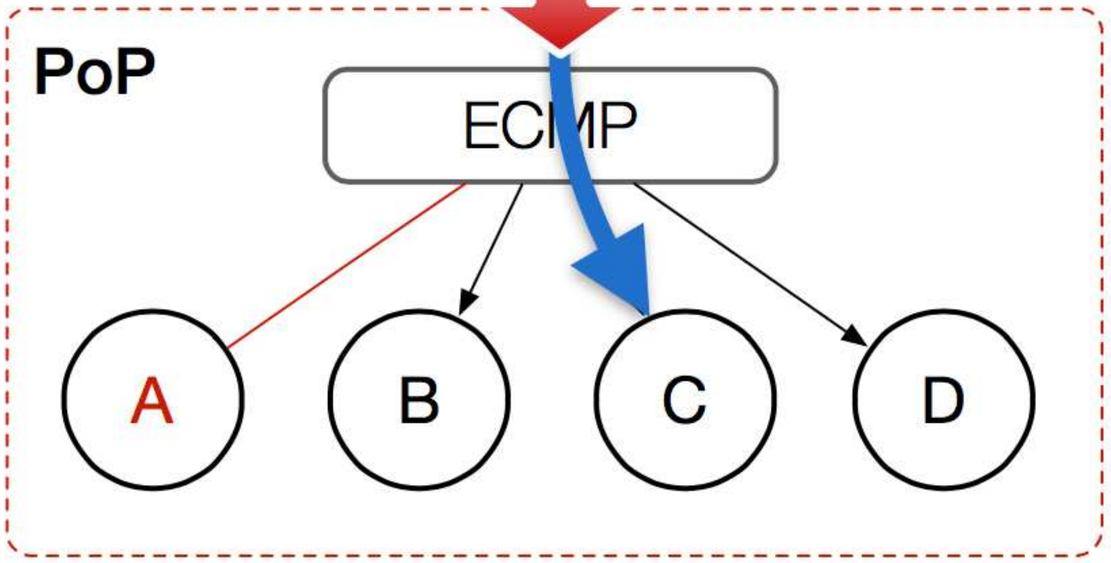
Key insights from NetSys

Faild allows us to fail & recover hosts via MAC-swapping and ECMP on switches

Do **immediate** or gradual **host failure & recovery**

Watch Joao's talk SREcon15 03.16.15-03.17.15 | SANTA CLARA, CA

fastly®



Key insights from NetSys

Faild allows us to fail & recover hosts via MAC-swapping and ECMP on switches

Do **immediate** or gradual **host failure & recovery**

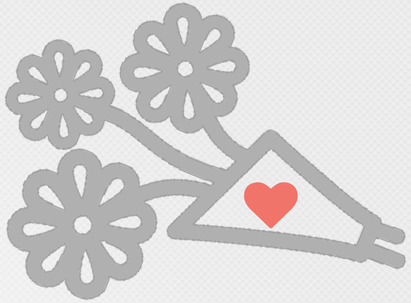
Watch Joao's talk SREcon15 03.16.15-03.17.15 | SANTA CLARA, CA

fastly®

Destination network	Next hop
10.0.0.0/24	10.1.**A**.1
10.0.0.0/24	10.1.**A**.2
10.0.0.0/24	10.1.**A**.3
…	…

IP Address	MAC
10.1.**A**.1	A:A
10.1.**A**.2	A:A
10.1.**A**.3	A:A
…	…

But wait a minute!



So we have a myriad of systems with ***different stages of evolution***

Resilient systems like Varnish, Powderhorn, and Faild have taught us many lessons but some applications have availability problems, ***why?***

fastly®

fastly®

@randommood

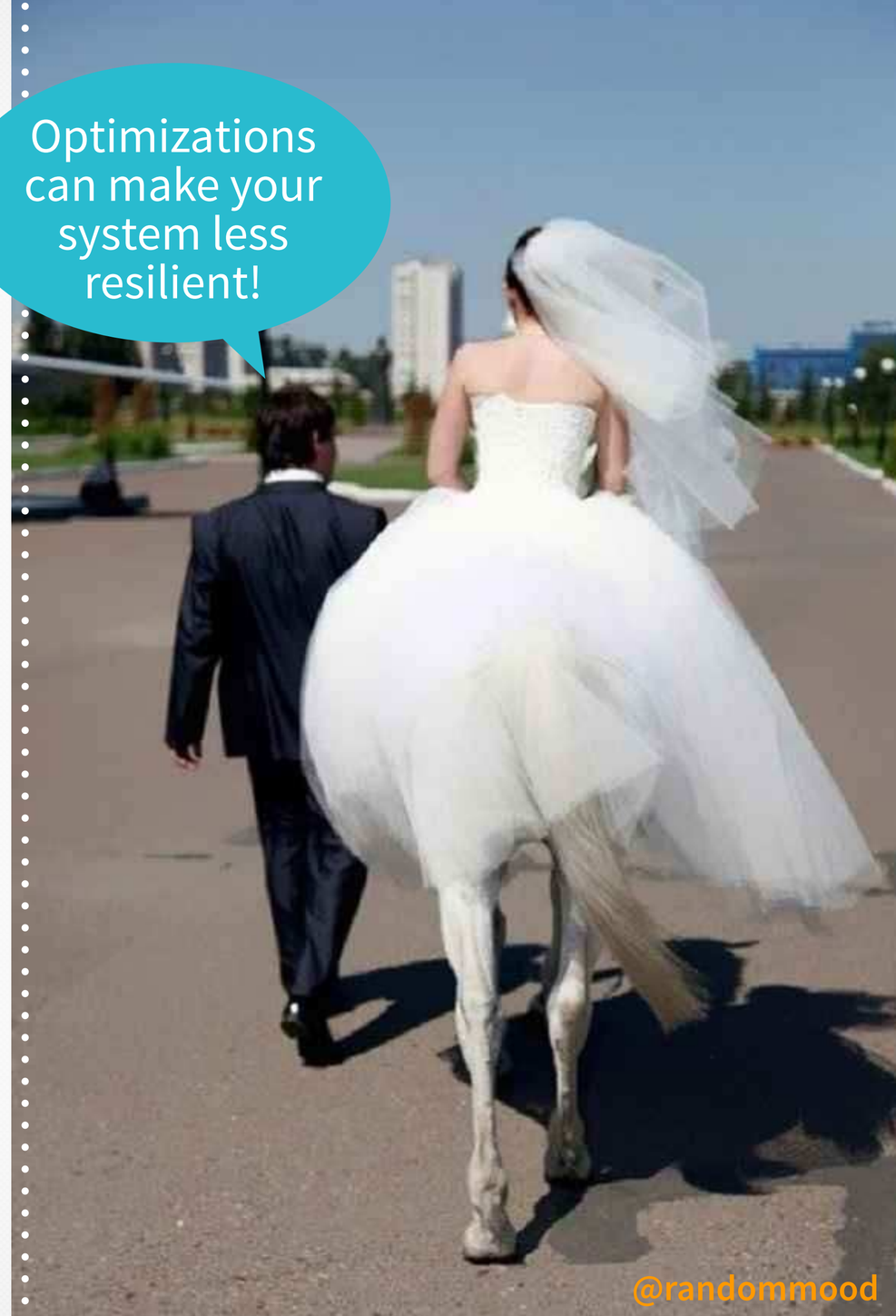
Resilient architectural patterns

Redundancies are key

Redundancies of resources, execution paths, checks, replication of data, replay of messages, anti-entropy **build resilience**

Gossip / epidemic protocols too

Capacity planning matters



fastly®

Operations matter

Unawareness of proximity to error boundary **means we are always guessing**

Complex operations make systems less resilient & more incident-prone

You design operability too!

@randommood

fastly

Not all complexity is bad

Complexity if increases safety is actually good

Adding resilience may **come at the cost of other desired goals** (e.g. performance, simplicity, cost, etc)

Leverage Engineering best practices

Resiliency and testing are correlated. TEST!

Versioning from the start - provide an upgrade path from day 1

Upgrades & evolvability of systems is still tricky. Mixed-mode operations need to be common

Re-examine the way we prototype systems

fastly®

Bringing it together

tl;dr

WHILE IN DESIGN

Are we **favoring harvest** or **yield**? Orthogonality & decomposition FTW

Do we have enough redundancies in place?

Are we resilient to our dependencies?

Theory matters!

OPERABILITY

Am I providing enough control to my operators?

Would I want to be on call for this?

Rank your services: what can be dropped, killed, deferred?

Monitoring and alerting in place?

UNK-UNK

The existence of this **stresses diligence on the other two areas**

Have we done everything we can?

Abandon hope and resort to human sacrifices

♥ tl;dr ♥

WHILE IN DESIGN

Test dependency failures

Code reviews != tests. Have both

Distrust client behavior, even if they are internal

Version (APIs, protocols, disk formats) from the start. Support mixed-mode operations.

Checksum all the things

Error handling, circuit breakers, backpressure, leases, timeouts

IMPROVING OPERABILITY

Automation shortcuts taken while in a rush will come back to haunt you

Release stability is often tied to system stability. Iron out your deploy process

Link alerts to playbooks

Consolidate system configuration (data bags, config file, etc)

Operators determine resilience

❤ TODAY'S RANTIFESTO ❤

We can't recover from lack of design. Not minding harvest/yield means **we sign up for a redesign** the moment we finish coding.

fastly®

@randommood

Special thanks to

Paul Borrill, Jordan West, Caitie McCaffrey, Camille Fournier, Mike O'Neill, Neha Narula, Joao Taveira, Tyler McMullen, Zac Duncan, Nathan Taylor, Ian Fung, Armon Dadgard, Peter Alvaro, Peter Bailis, Bruce Spang, Matt Whiteley, Alex Rasmussen, Aysulu Greenberg, Elaine Greenberg, and Greg Bako.

★Thank you!★

github.com/Randommood/Strangeloop2015