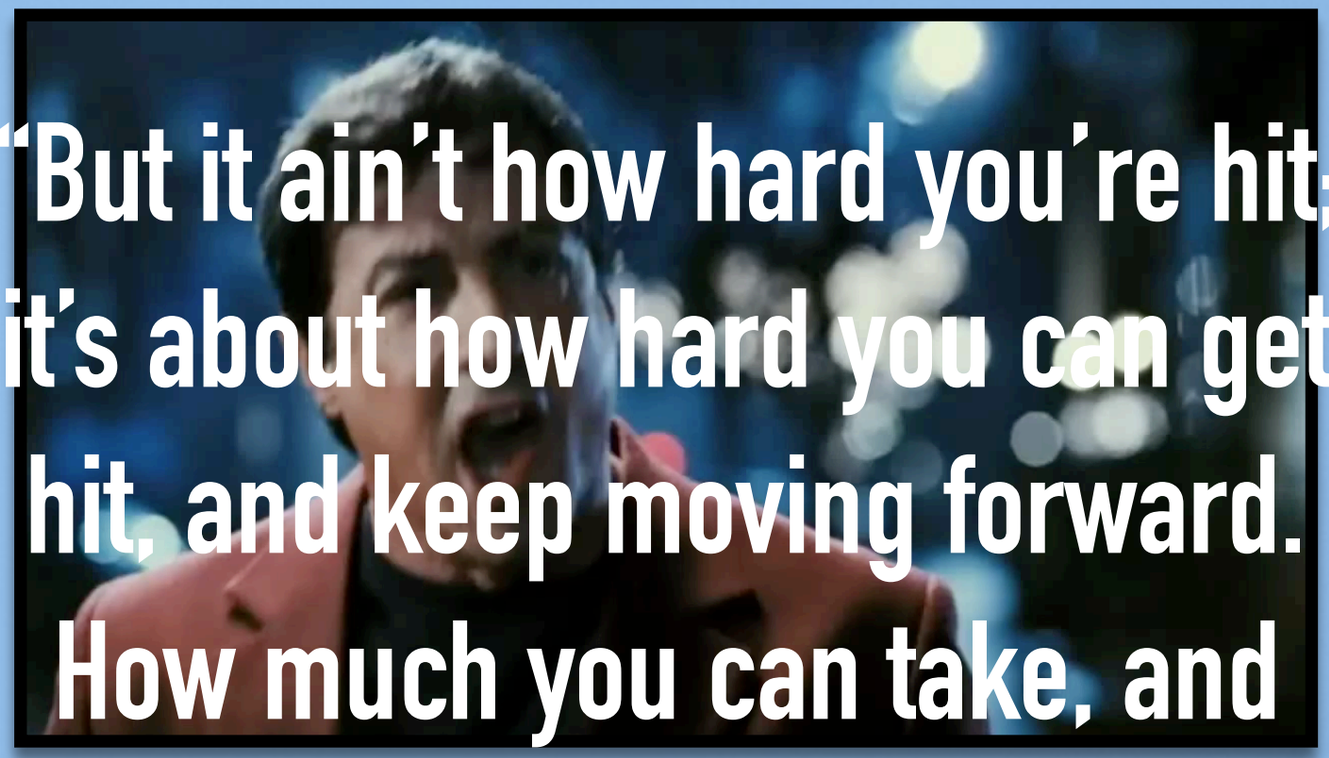
Without Resilience Nothing Else Matters

JONAS BONÉR
CTO Lightbend
@jboner



"But it ain't how hard you're hit; it's about how hard you can get hit, and keep moving forward. How much you can take, and keep moving forward. That's how winning is done."

- ROCKY BALBOA

This Is Fault Tolerance

Resilience Is Beyond Fault Tolerance

Resilience

"The ability of a substance or object to spring back into shape. The capacity to recover quickly from difficulties."

-MERRIAM WEBSTER

Antifragility

"Antifragility is beyond resilience and robustness. The resilient resists shock and stays the same; the antifragile gets better."

- NASSEM NICHOLAS TALEB

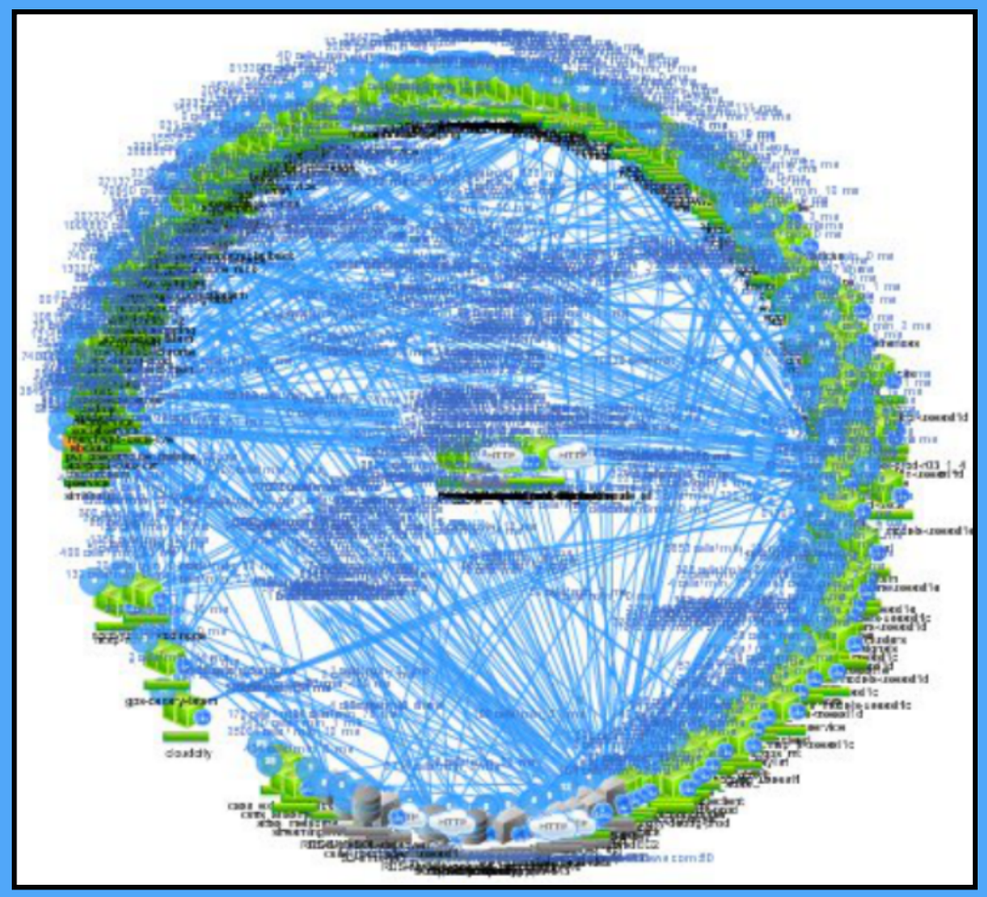
Antifragile: Things That Gain from Disorder - Nassim Nicholas Taleb

"We can model and understand in isolation.
But, when released into competitive nominally
regulated societies, their connections proliferate,
their interactions and interdependencies multiply,
their complexities mushroom.
And we are caught short."

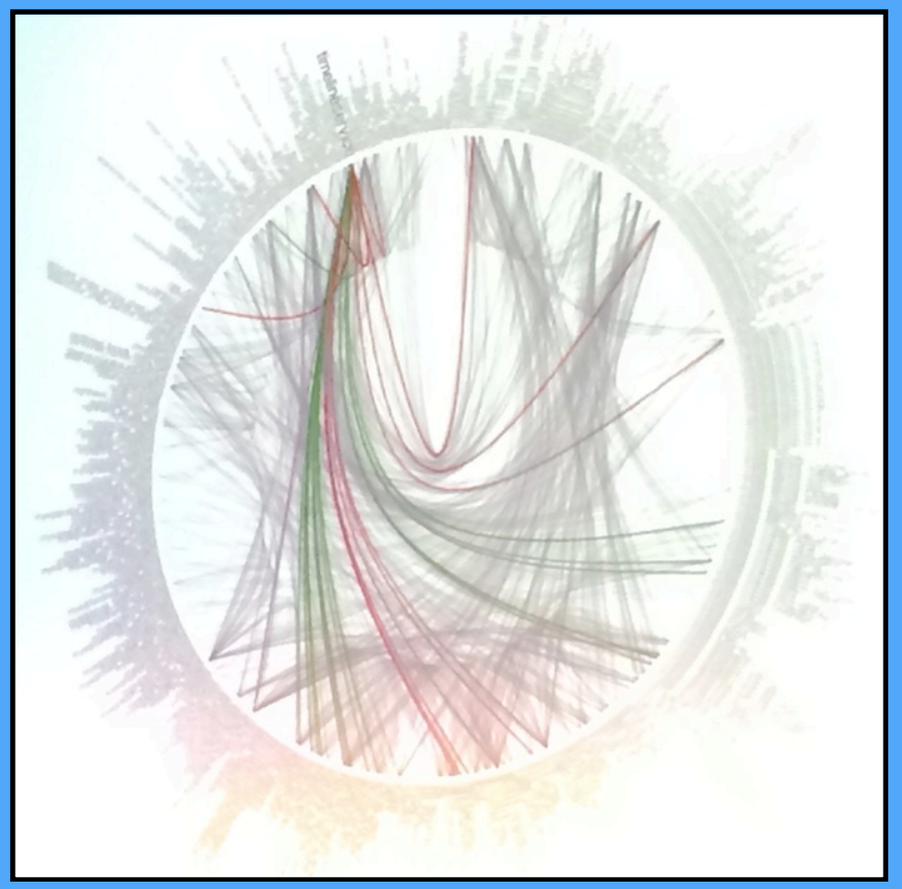
- SIDNEY DEKKER

Drift into Failure - Sidney Dekker

Software Systems Today Are Incredibly Complex



NETFLIX

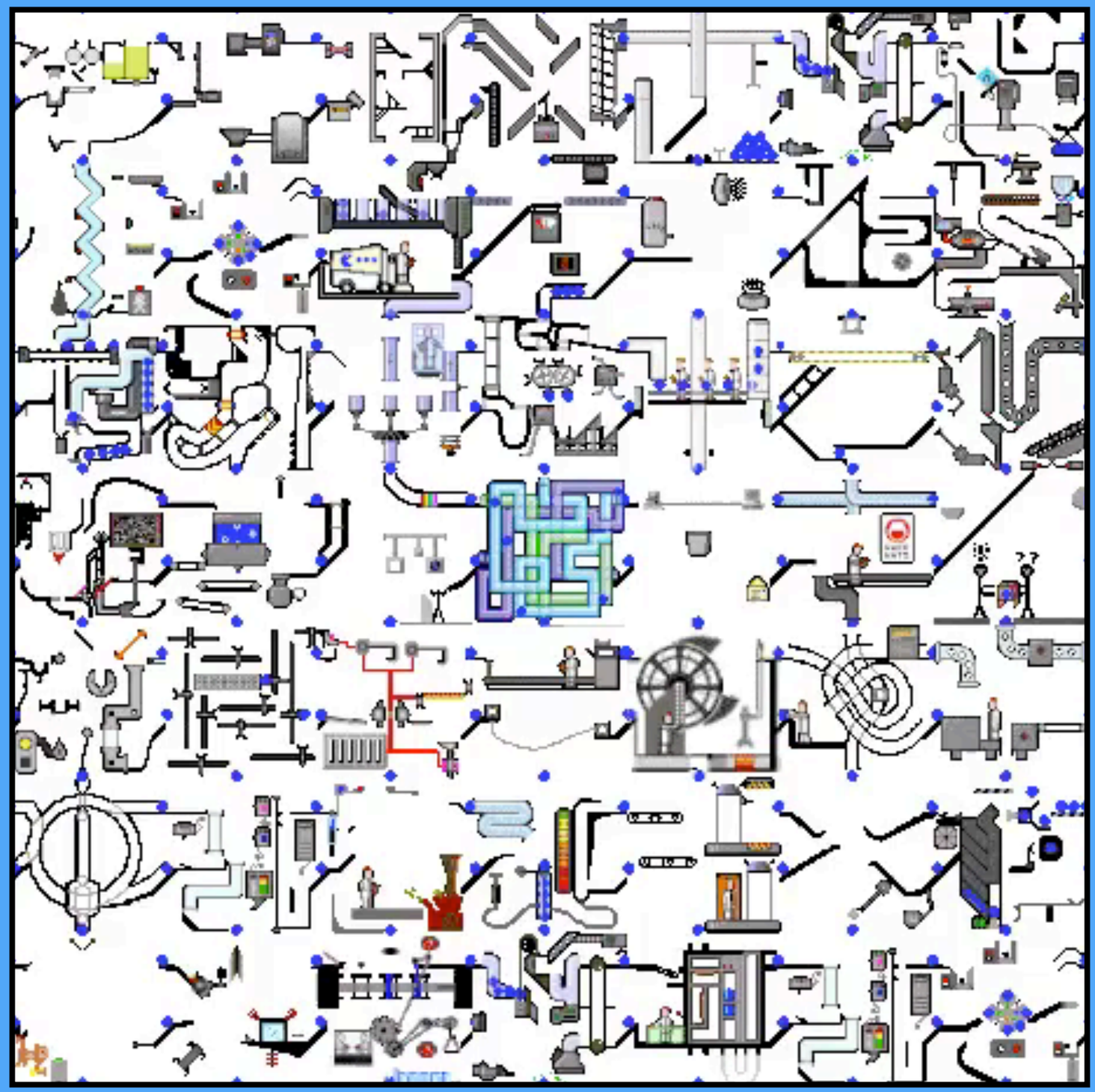


TWITTER

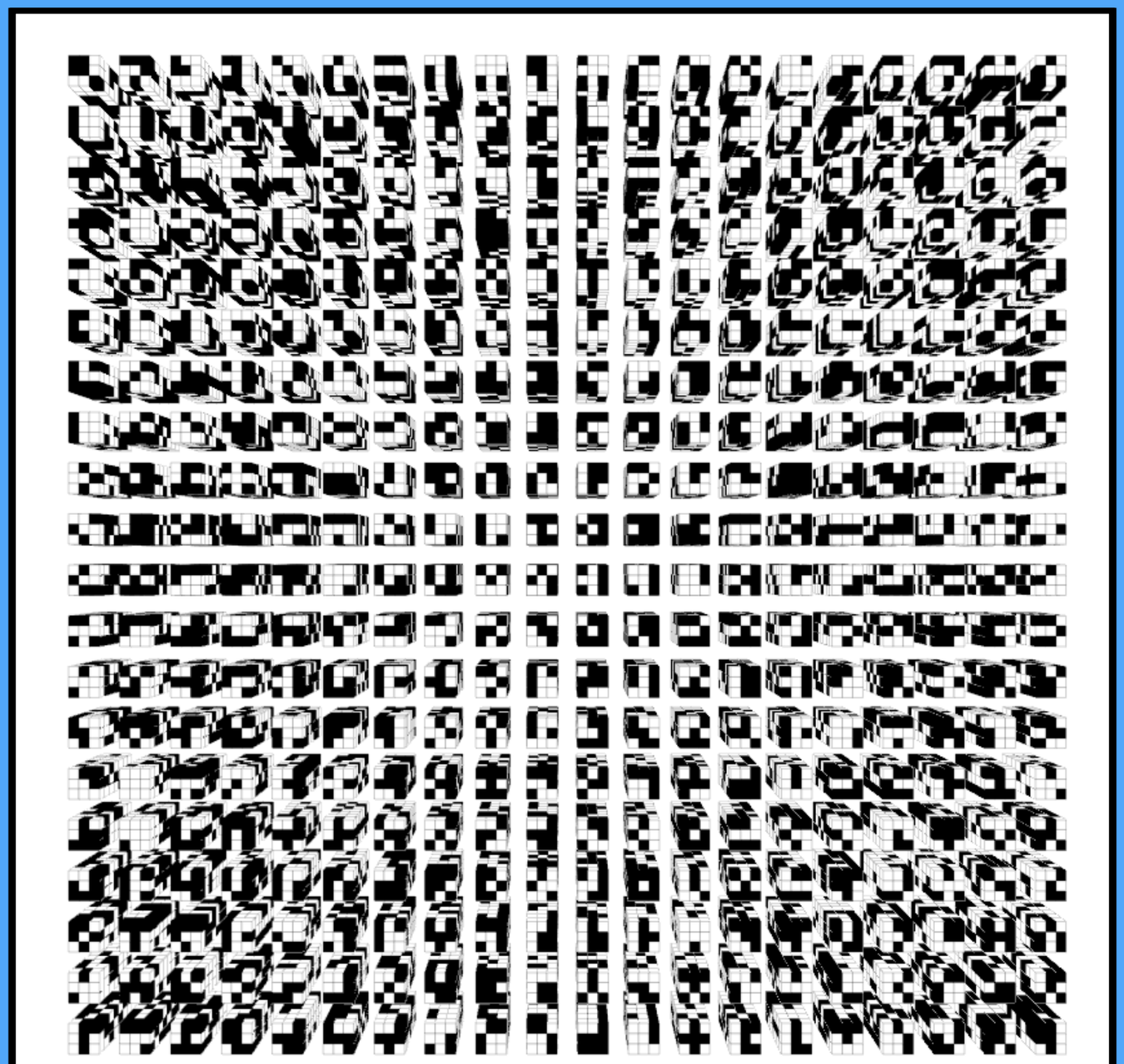
WE NEED TO STUDY

Resilience in Complex Systems

Complicated System



Complex System



COMPLICATED ≠ COMPLEX

“Complex systems run in degraded mode.”

“Complex systems run as broken systems.”

- RICHARD COOK

How Complex Systems Fail - Richard Cook

"Counterintuitive. That's [Jay] Forrester's word to describe complex systems. Leverage points are not intuitive. Or if they are, we intuitively use them backward, systematically worsening whatever problems we are trying to solve."

- DONELLA MEADOWS

Leverage Points: Places to Intervene in a System - Donella Meadows

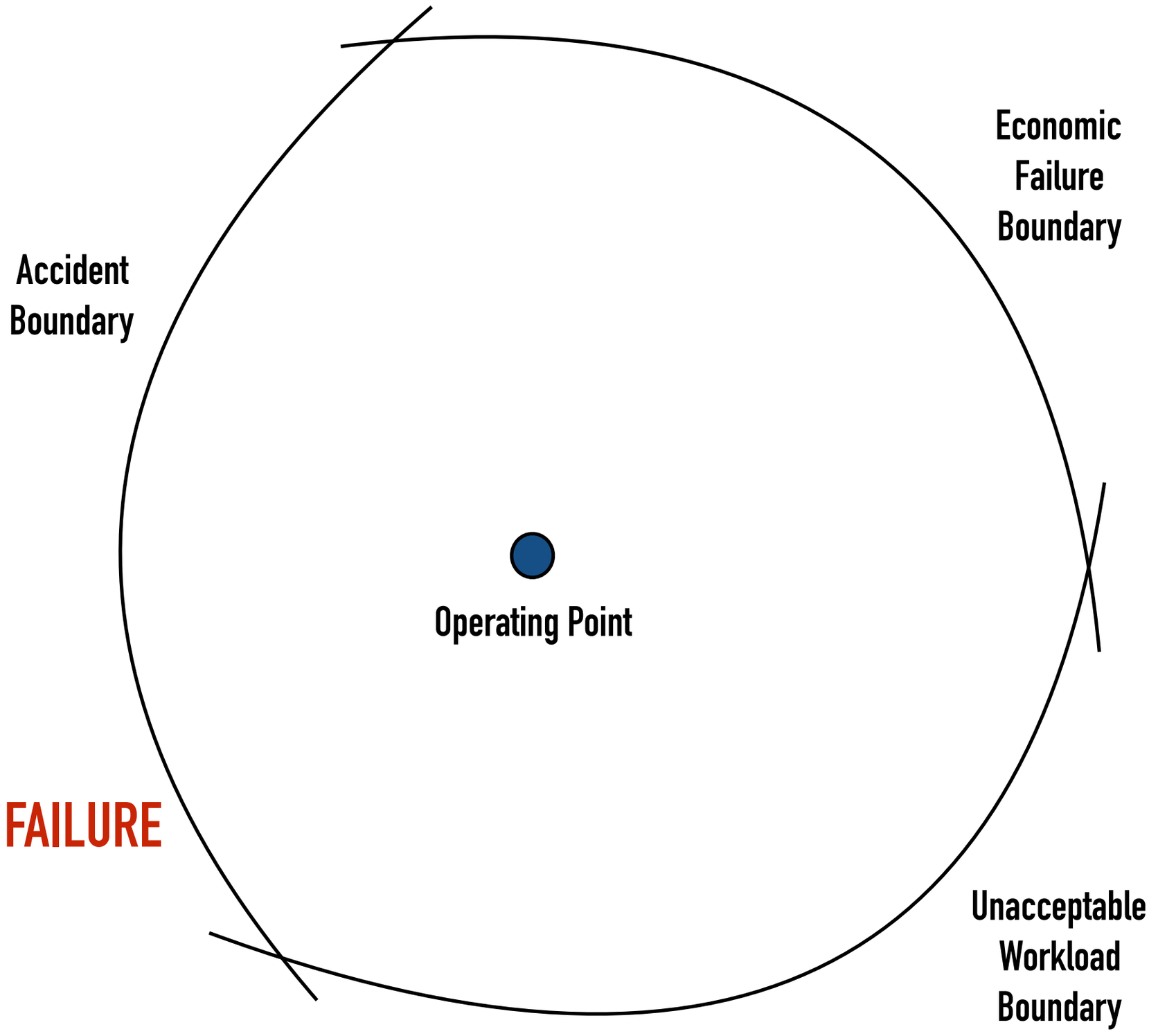
“Humans should not be involved in setting timeouts.”

“Human involvement in complex systems is the biggest source of trouble.”

- BEN CHRISTENSEN, NETFLIX

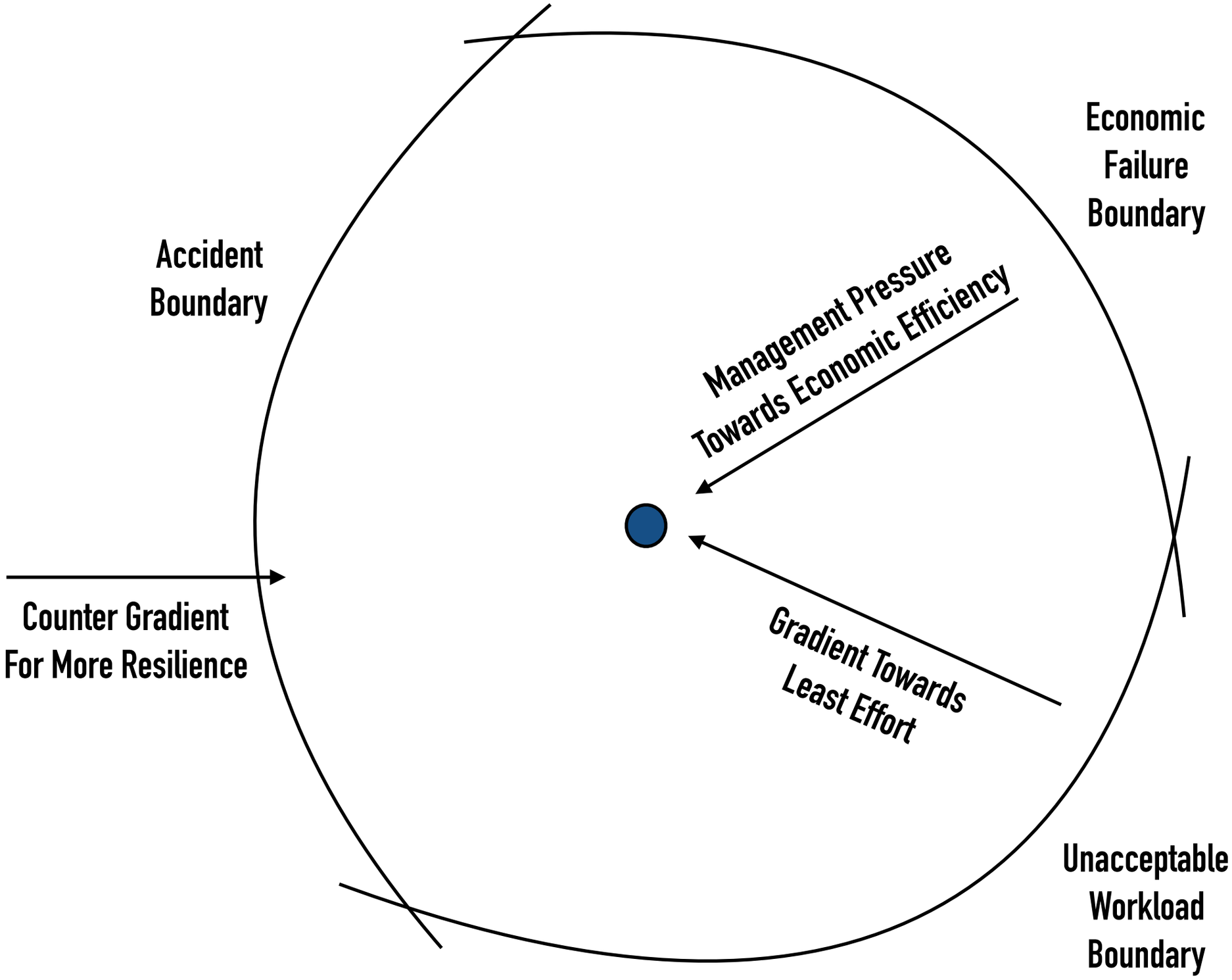
Humans Generally Make Things Worse

Operating at the Edge of Failure



"Going solid": a model of system dynamics and consequences for patient safety - R Cook, J Rasmussen
Resilience in complex adaptive systems: Operating at the Edge of Failure - Richard Cook - Talk at Velocity NY 2013

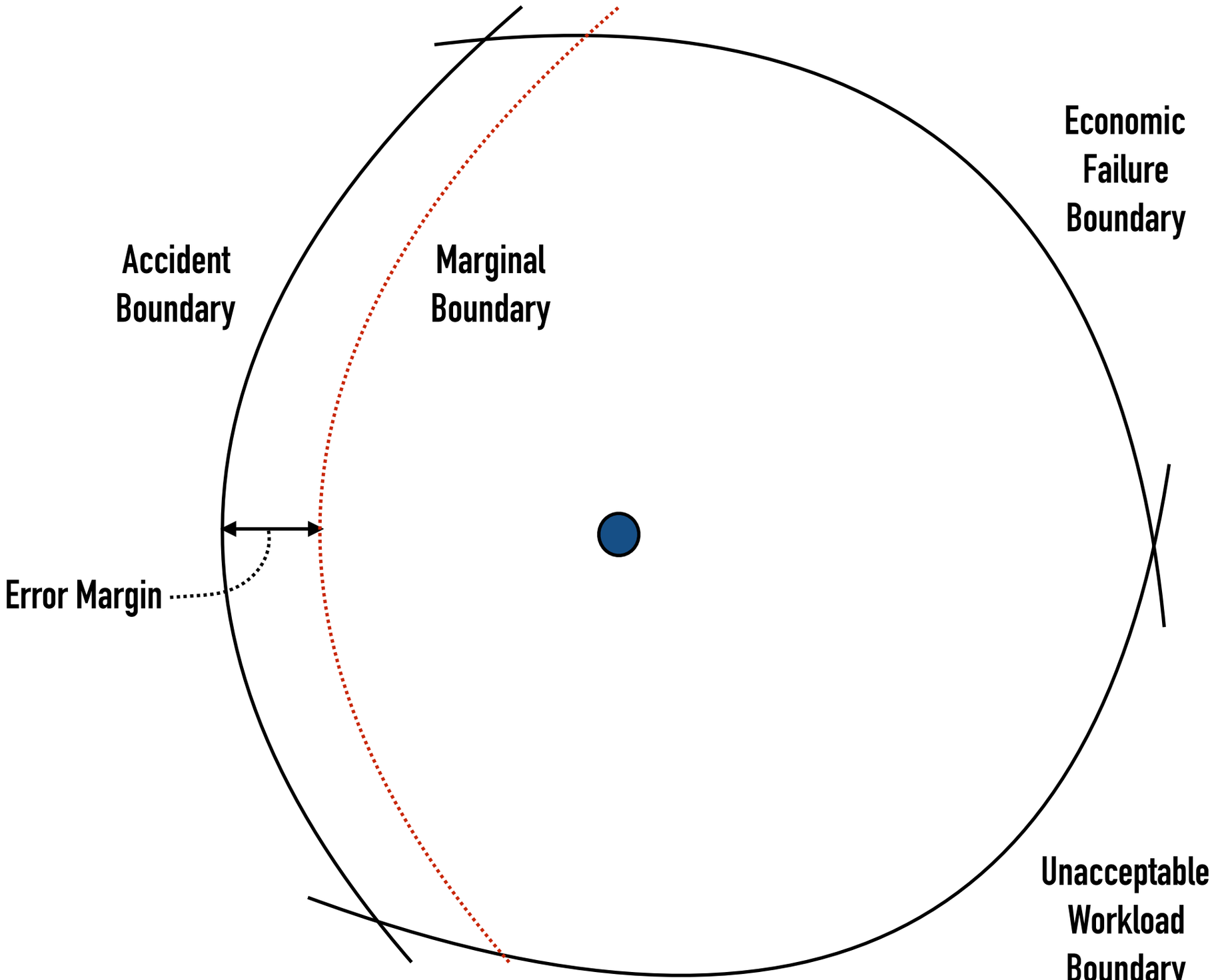
Operating at the Edge of Failure



"Going solid": a model of system dynamics and consequences for patient safety - R Cook, J Rasmussen

Resilience in complex adaptive systems: Operating at the Edge of Failure - Richard Cook - Talk at Velocity NY 2013

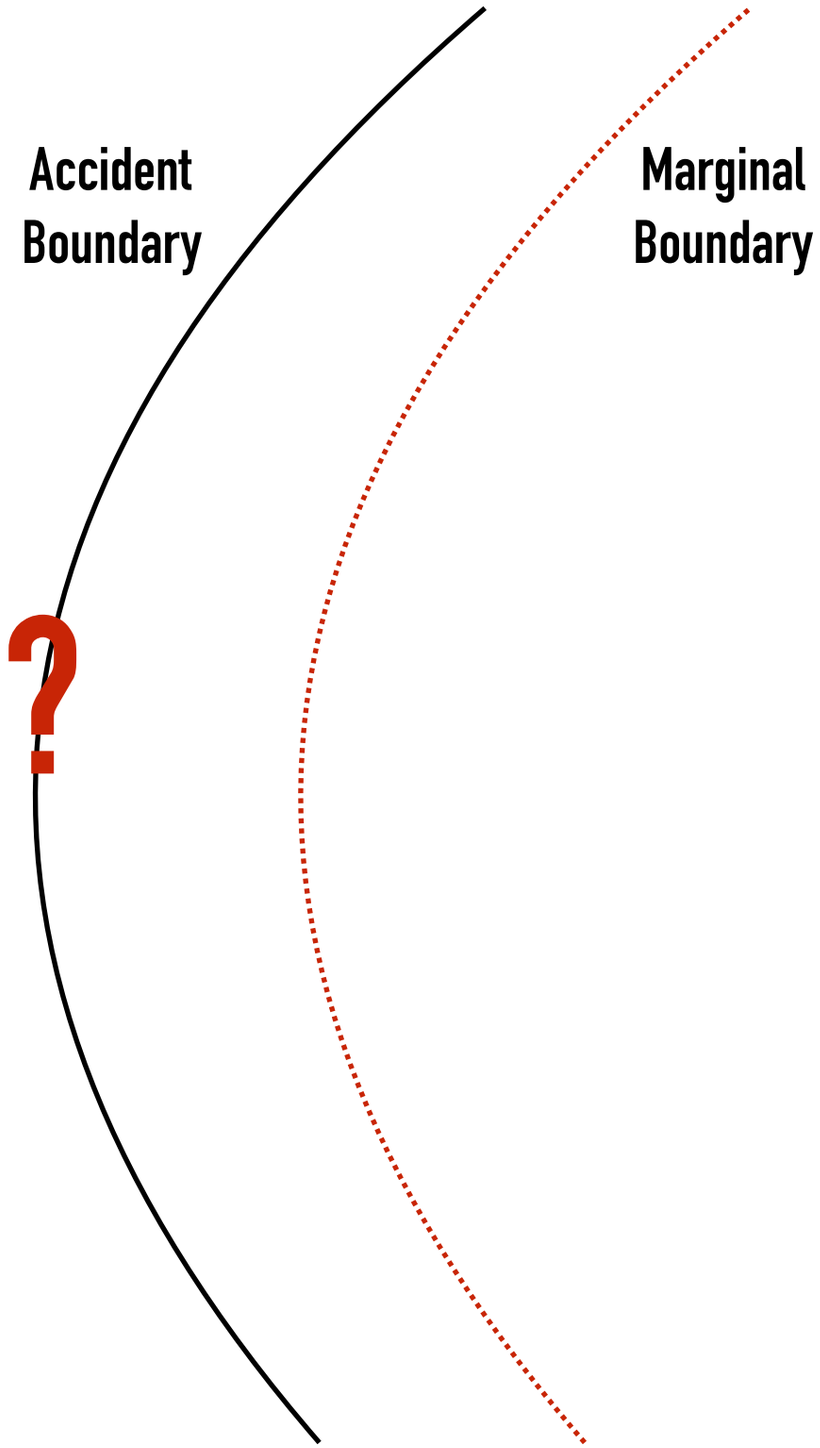
Operating at the Edge of Failure



"Going solid": a model of system dynamics and consequences for patient safety - R Cook, J Rasmussen

Resilience in complex adaptive systems: Operating at the Edge of Failure - Richard Cook - Talk at Velocity NY 2013

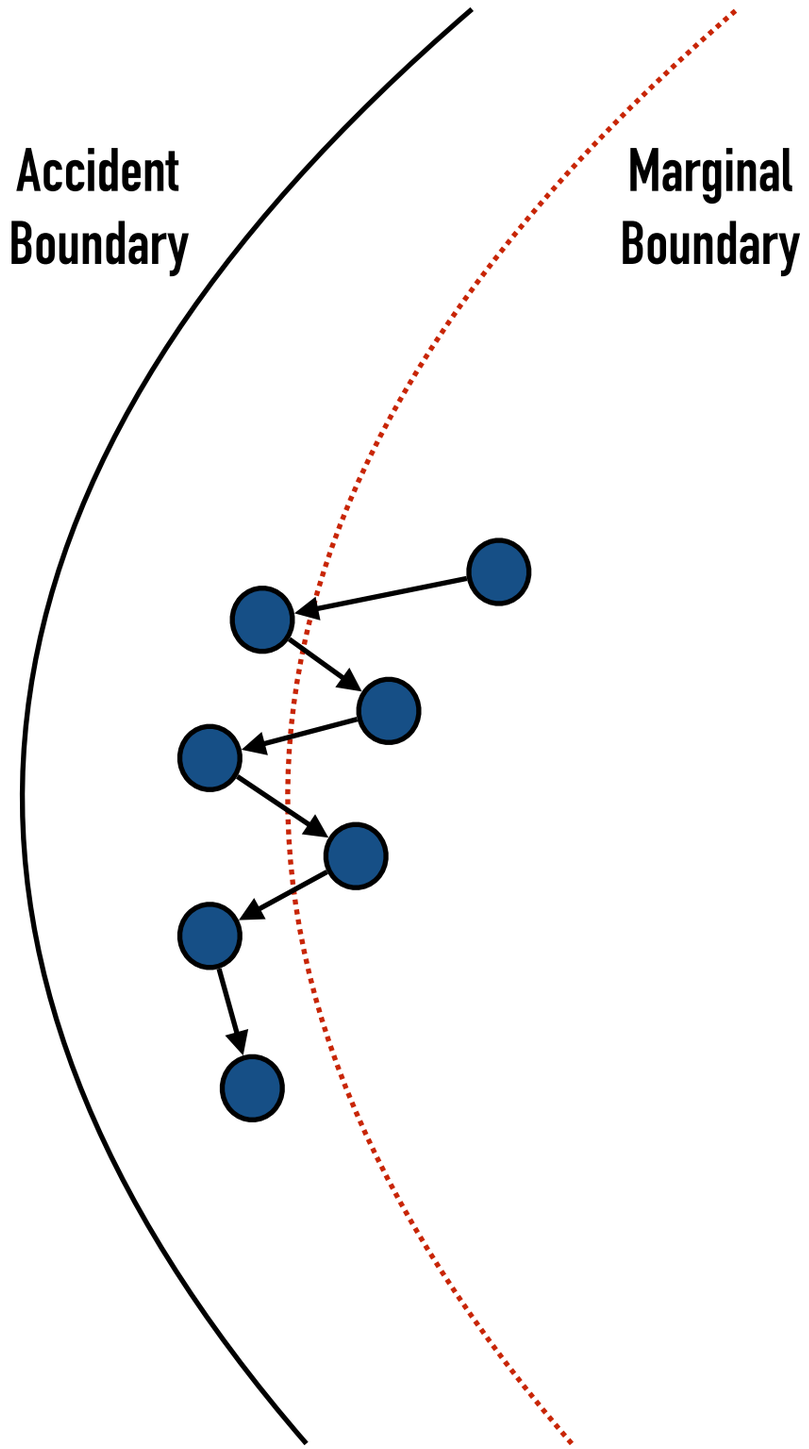
Operating at the Edge of Failure



"Going solid": a model of system dynamics and consequences for patient safety - R Cook, J Rasmussen

Resilience in complex adaptive systems: Operating at the Edge of Failure - Richard Cook - Talk at Velocity NY 2013

Operating at the Edge of Failure



"Going solid": a model of system dynamics and consequences for patient safety - R Cook, J Rasmussen

Resilience in complex adaptive systems: Operating at the Edge of Failure - Richard Cook - Talk at Velocity NY 2013

Embrace Failure

Resilience is by Design

Photo courtesy of FEMA/Joselyne Augustino

"Autonomy makes information local, leading to greater certainty and stability."

- MARK BURGESS

In Search of Certainty - Mark Burgess

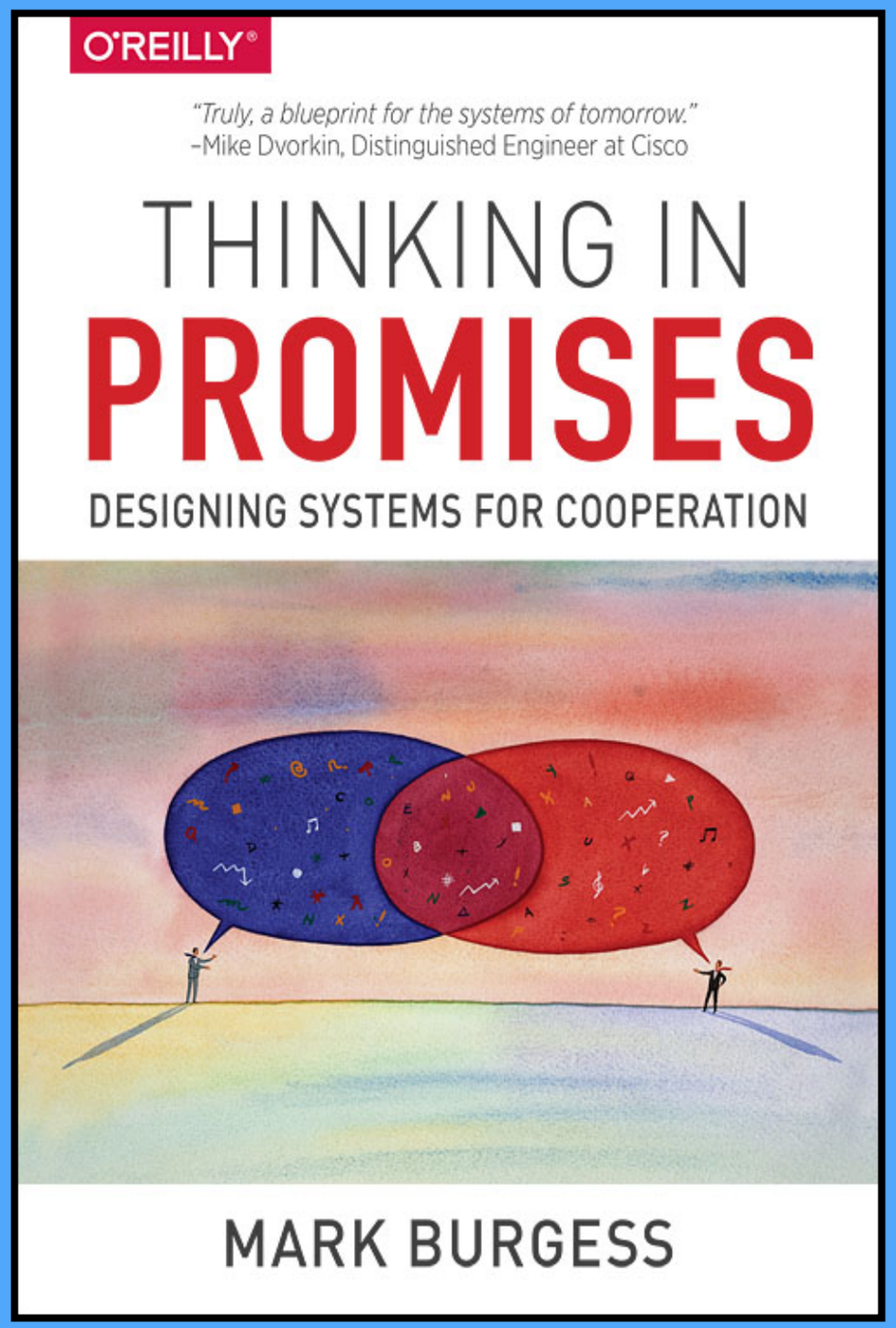
Promise Theory

THINK IN

PROMISES

NOT

COMMANDS



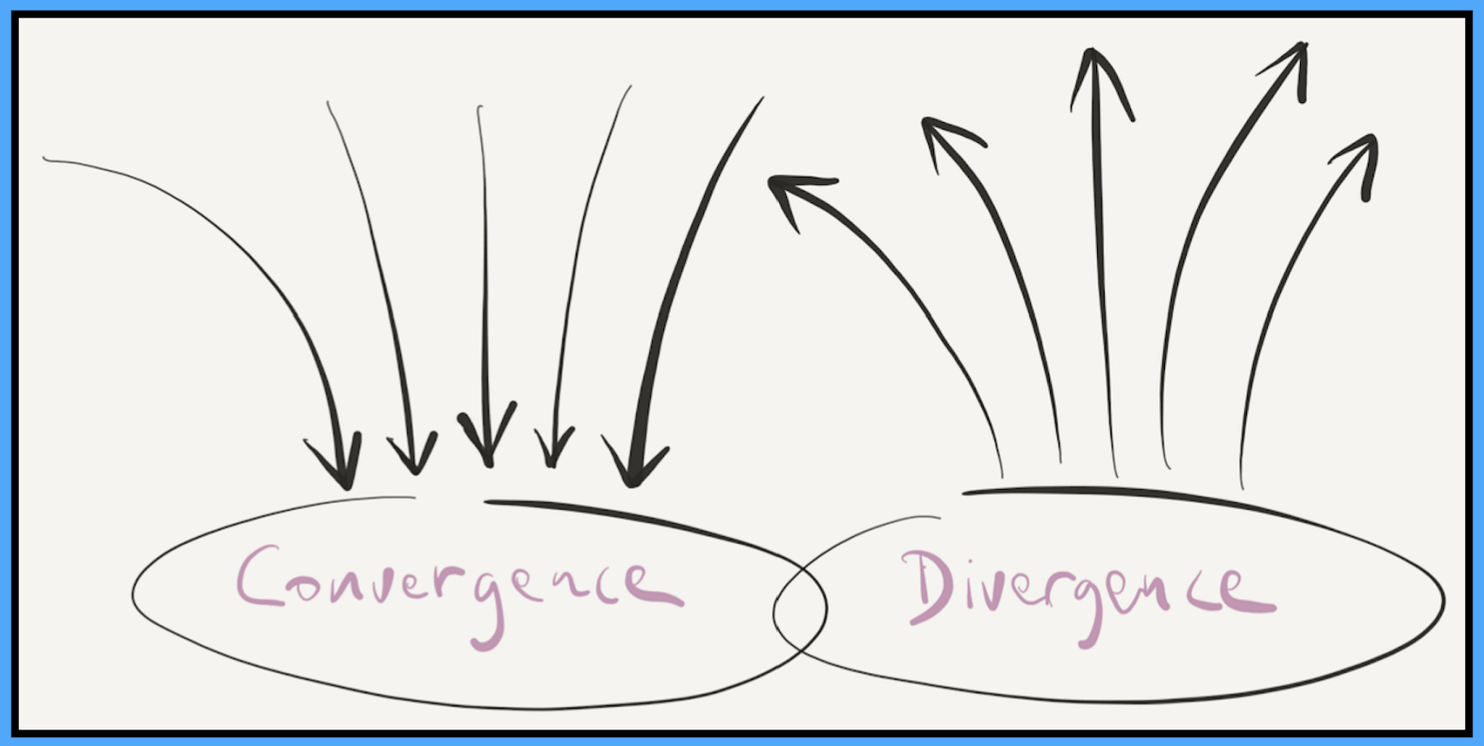
Promise Theory

PROMISES CONVERGE TOWARDS
A DEFINITE OUTCOME FROM
UNPREDICTABLE BEGINNINGS

⇒ IMPROVED STABILITY

COMMANDS DIVERGE INTO
UNPREDICTABLE OUTCOMES FROM
DEFINITE BEGINNINGS

⇒ DECREASED STABILITY



Resilience in Biological Systems

Meerkats

Puppies! Now that I've got your attention, complexity theory - Nicolas Perony, TED talk

“In three words, in the animal kingdom, simplicity leads to complexity which leads to resilience.”

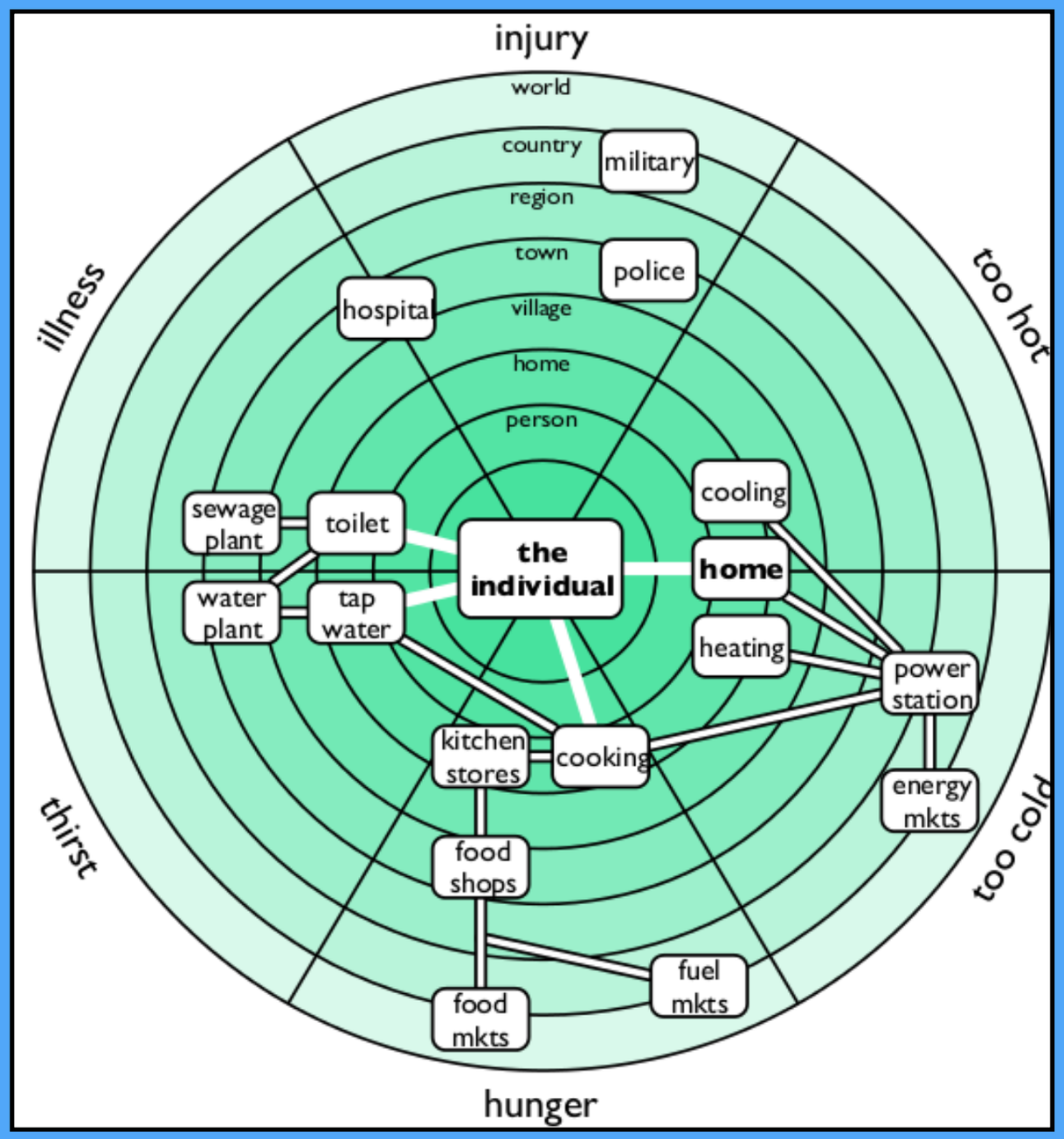
- NICOLAS PERONY

Puppies! Now that I’ve got your attention, complexity theory - Nicolas Perony, TED talk

Resilience in Social Systems

Dealing in Security

UNDERSTANDING VITAL SERVICES, AND HOW THEY KEEP YOU SAFE



1 INDIVIDUAL (YOU)

6 WAYS TO DIE

3 SETS OF ESSENTIAL SERVICES

7 LAYERS OF PROTECTION

WHAT WE CAN LEARN FROM

Resilience in Biological and Social Systems

1. FEATURE DIVERSITY AND REDUNDANCY
2. INTER-CONNECTED NETWORK STRUCTURE
3. WIDE DISTRIBUTION ACROSS ALL SCALES
4. CAPACITY TO SELF-ADAPT & SELF-ORGANIZE

Applying resilience thinking: Seven principles for building resilience in social-ecological systems - Reinette Biggs et. al.

Toward Resilient Architectures 1: Biology Lessons - Michael Mehaffy, Nikos A. Salingaros

Resilience in Computer Systems

Our Disaster Recovery Plan Goes Something Like This...

DILBERT

By Scott Adams

We Need To

Manage Failure

Not Try To Avoid It

Let It Crash

Crash Only Software

STOP = CRASH SAFELY
START = RECOVER FAST

Crash-Only Software - George Candea, Armando Fox

Recursive Restartability

TURNING THE CRASH-ONLY SLEDGEHAMMER INTO A SCALPEL

Recursive Restartability: Turning the Reboot Sledgehammer into a Scalpel - George Candea, Armando Fox

Traditional State Management

Client

Object

Critical state that needs protection

Thread boundary

Synchronous dispatch

Thread boundary

?

UTTERLY BROKEN

“Accidents come from relationships not broken parts.”

- SIDNEY DEKKER

Drift into Failure - Sidney Dekker

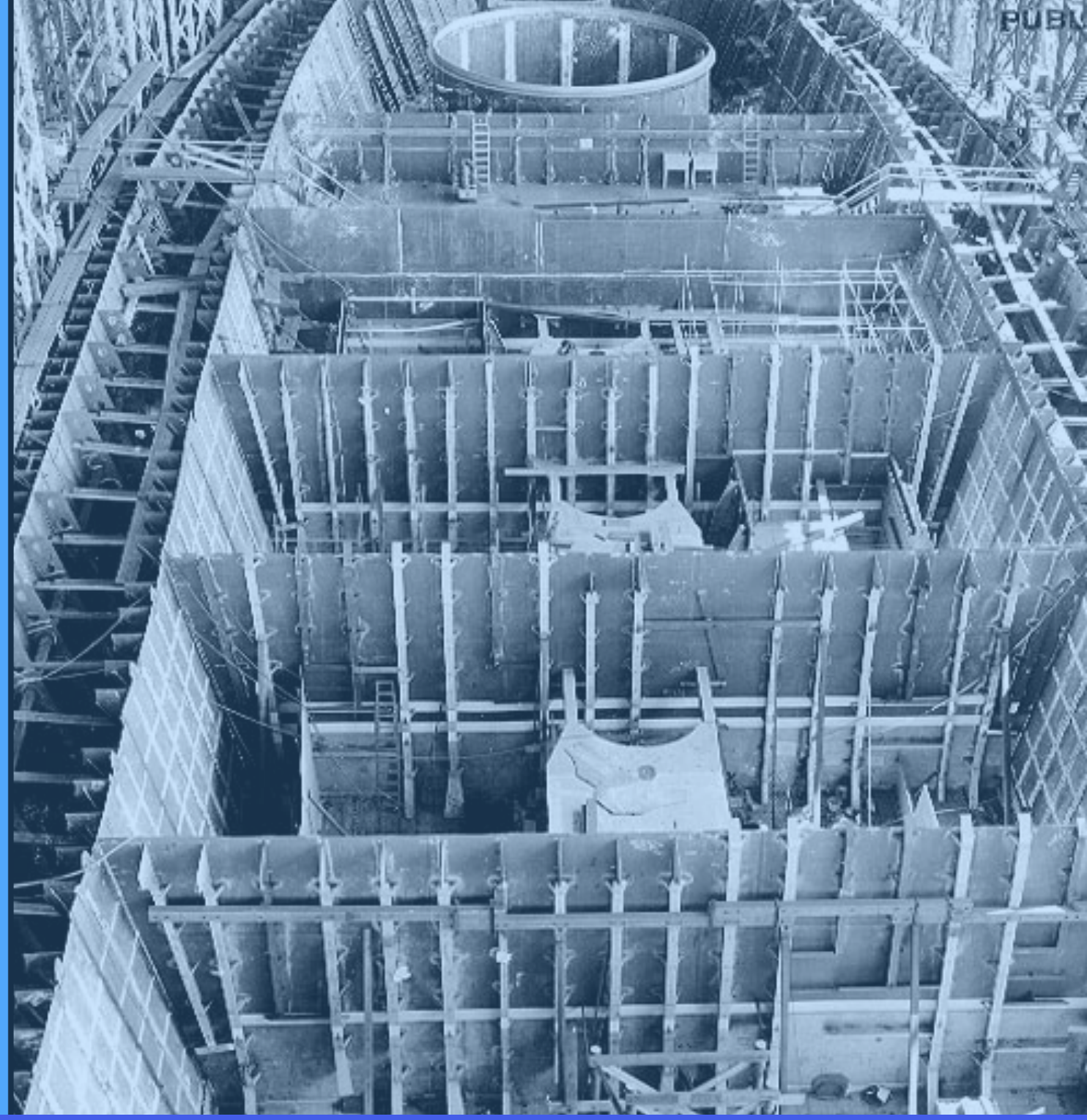
REQUIREMENTS FOR A

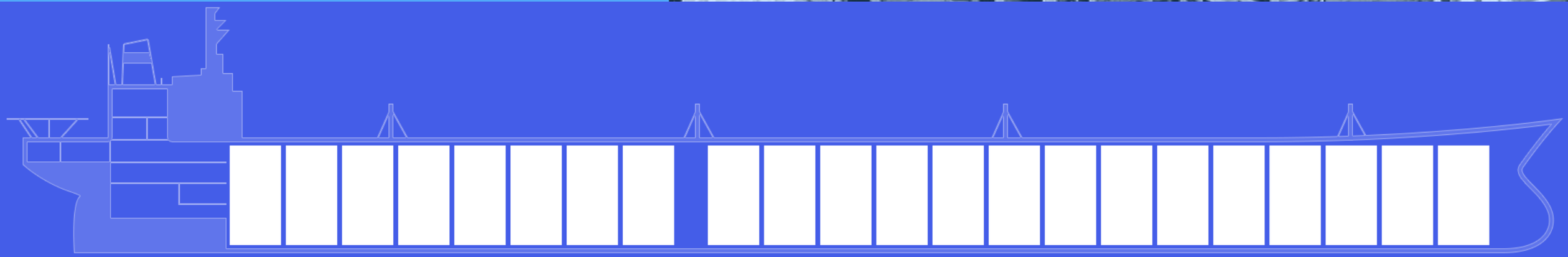
Sane Failure Model

FAILURES NEED TO BE

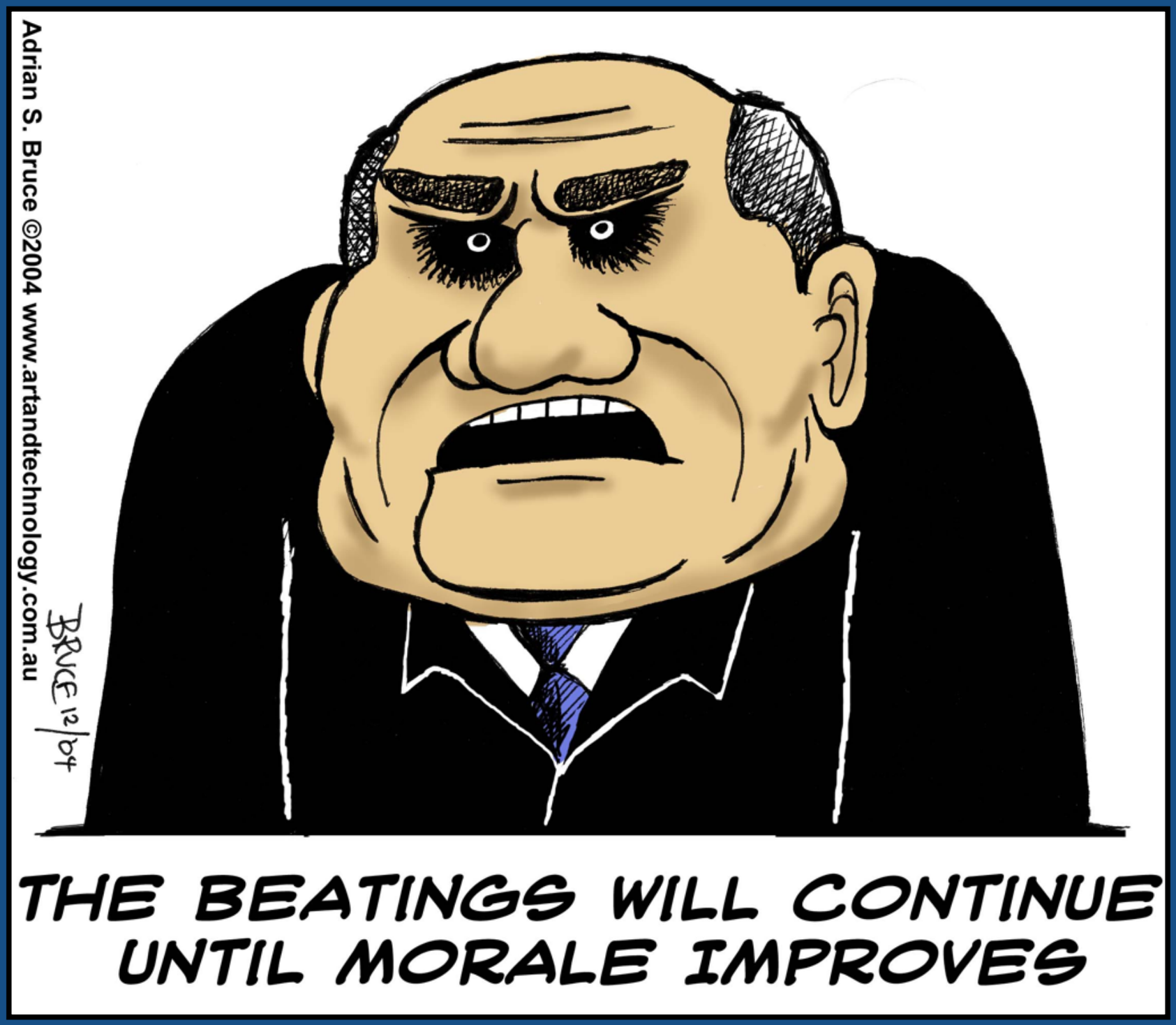
1. CONTAINED—AVOID CASCADING FAILURES
2. REIFIED—AS MESSAGES
3. SIGNALLED—ASYNCHRONOUSLY
4. OBSERVED—BY 1-N
5. MANAGED—OUTSIDE FAILED CONTEXT

Bulkhead Pattern





Enter Supervision



WE NEED A WAY OUT OF THE

State Tar Pit

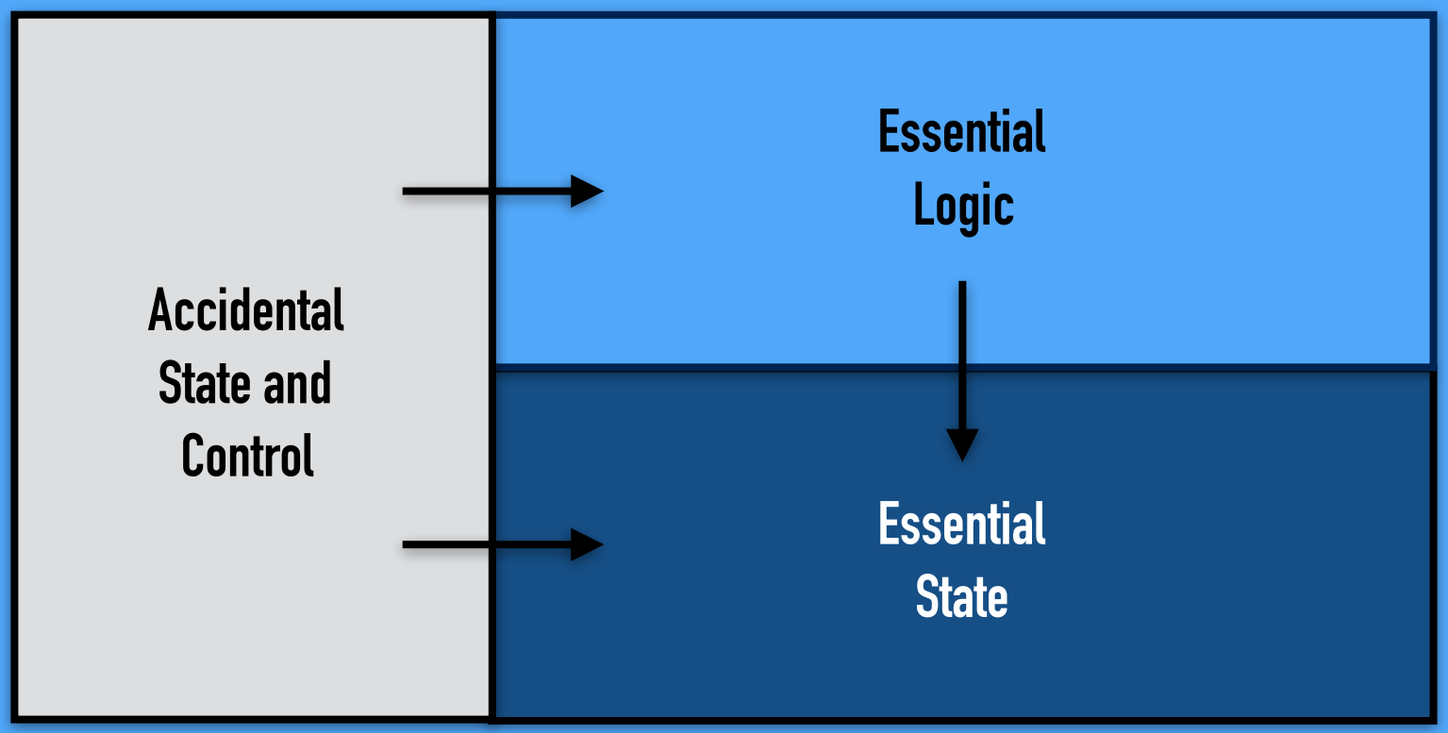
Critical

- INPUT DATA
- DERIVED DATA

Out of the Tar Pit - Ben Moseley , Peter Marks

WE NEED A WAY OUT OF THE

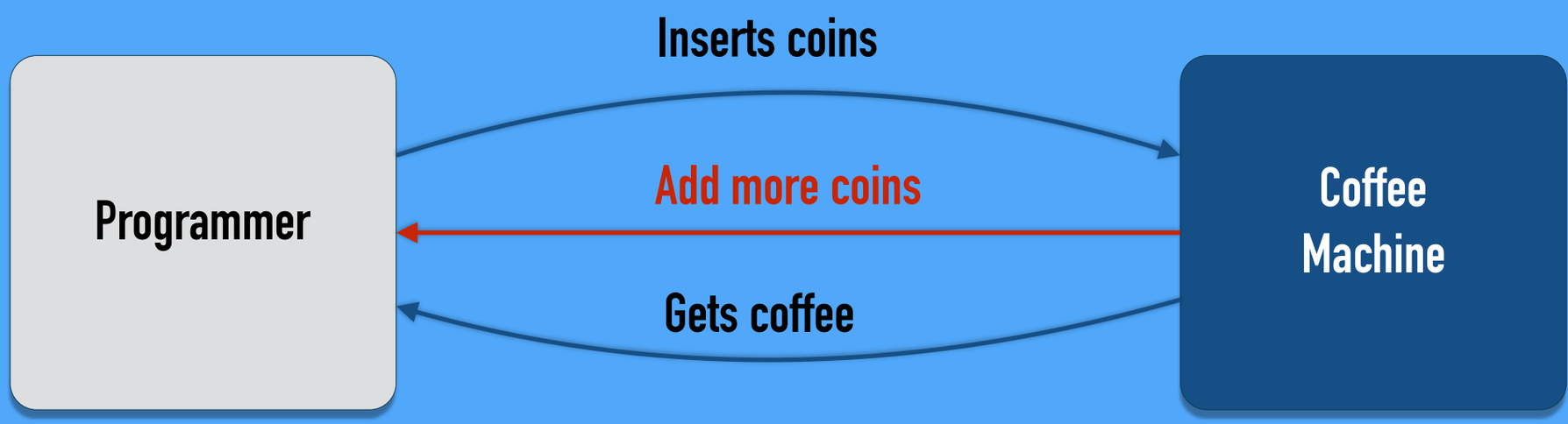
State Tar Pit



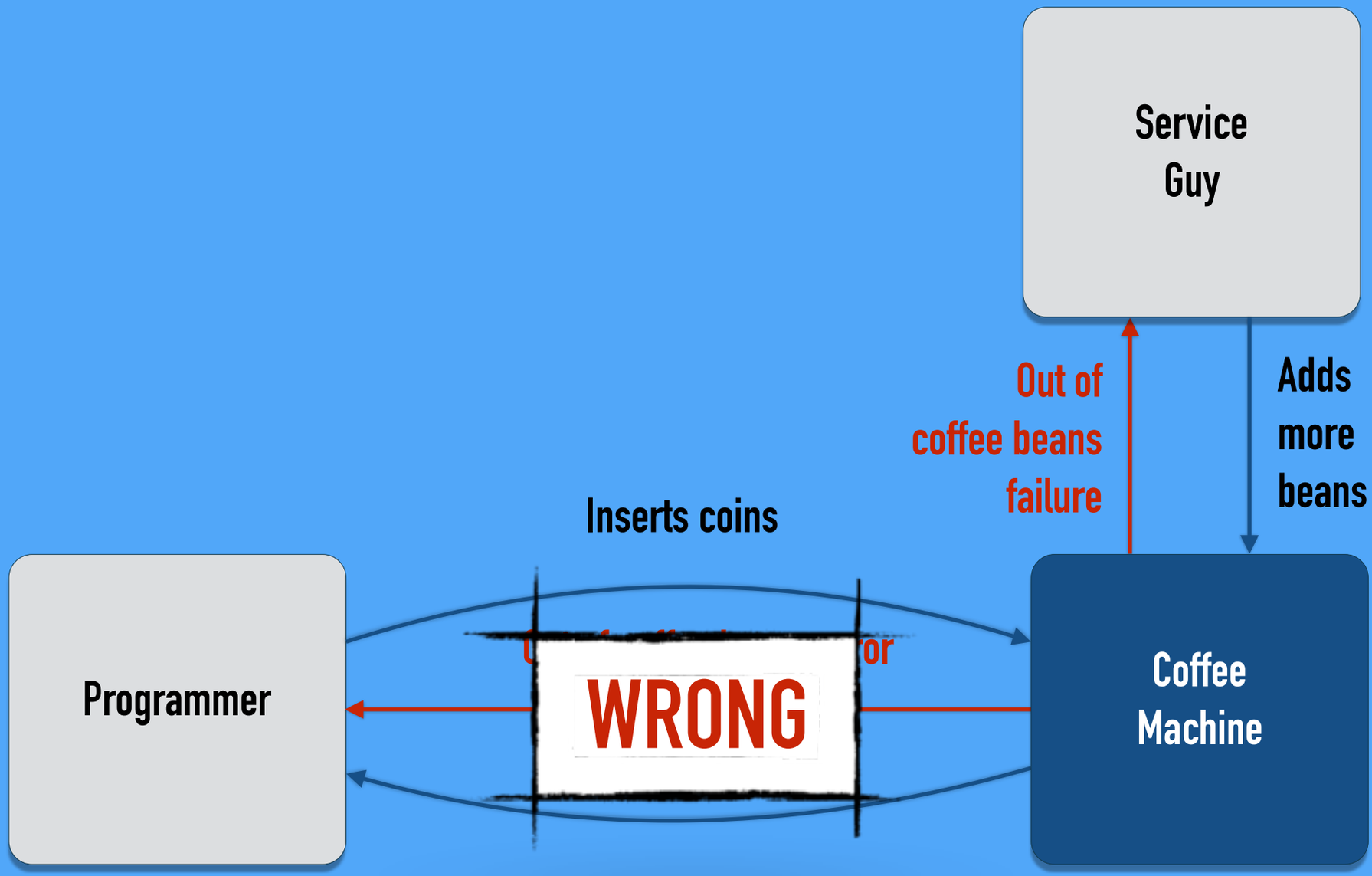
Out of the Tar Pit - Ben Moseley , Peter Marks

THE Vending Machine Pattern

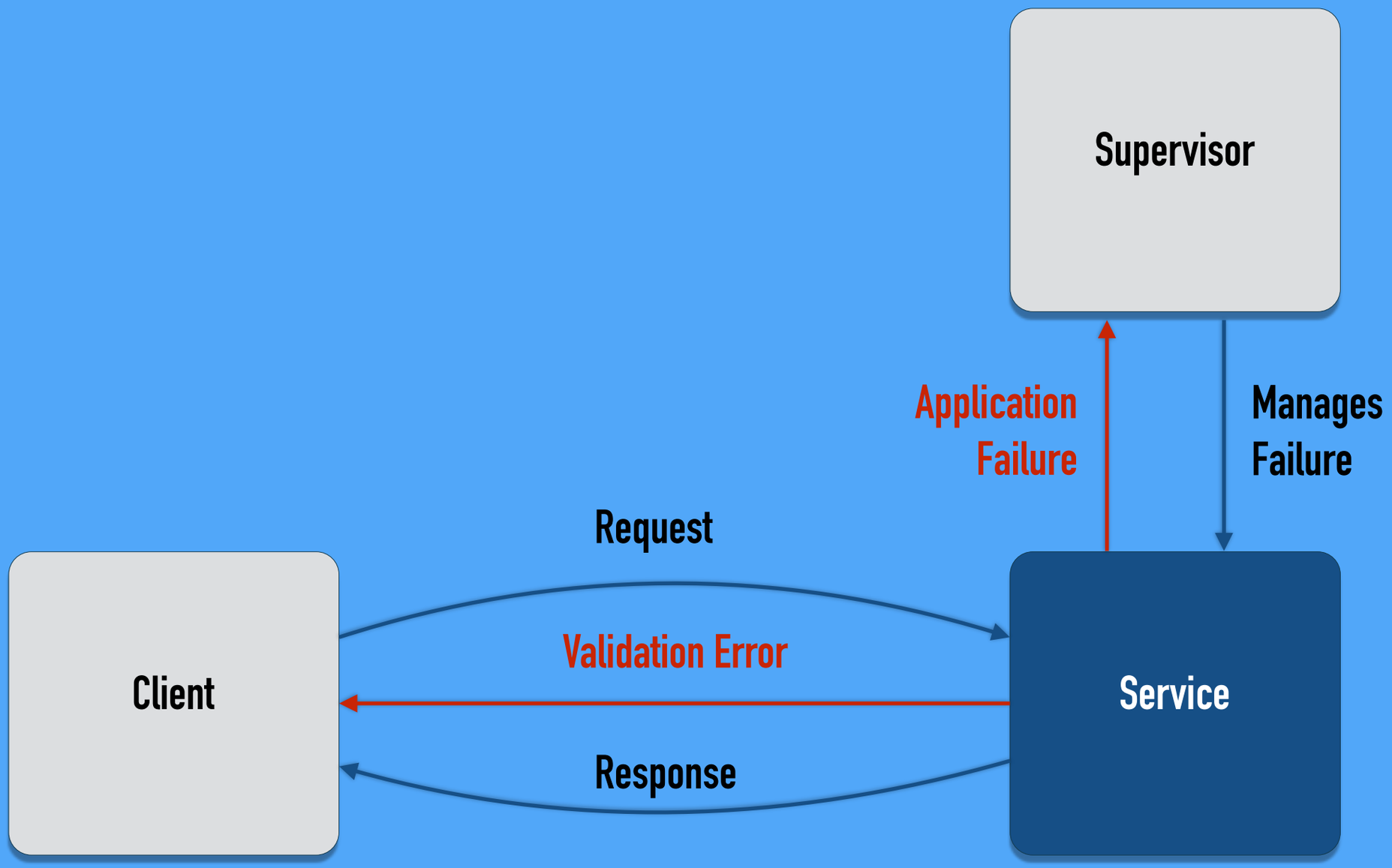
Think Vending Machine



Think Vending Machine



Think Vending Machine




Error
Kernel
Pattern
ONION-LAYERED STATE & FAILURE MANAGEMENT
Making reliable distributed systems in the presence of software errors - Joe Armstrong
On Erlang, State and Crashes - Jesper Louis Andersen

Onion Layered State Management

Client

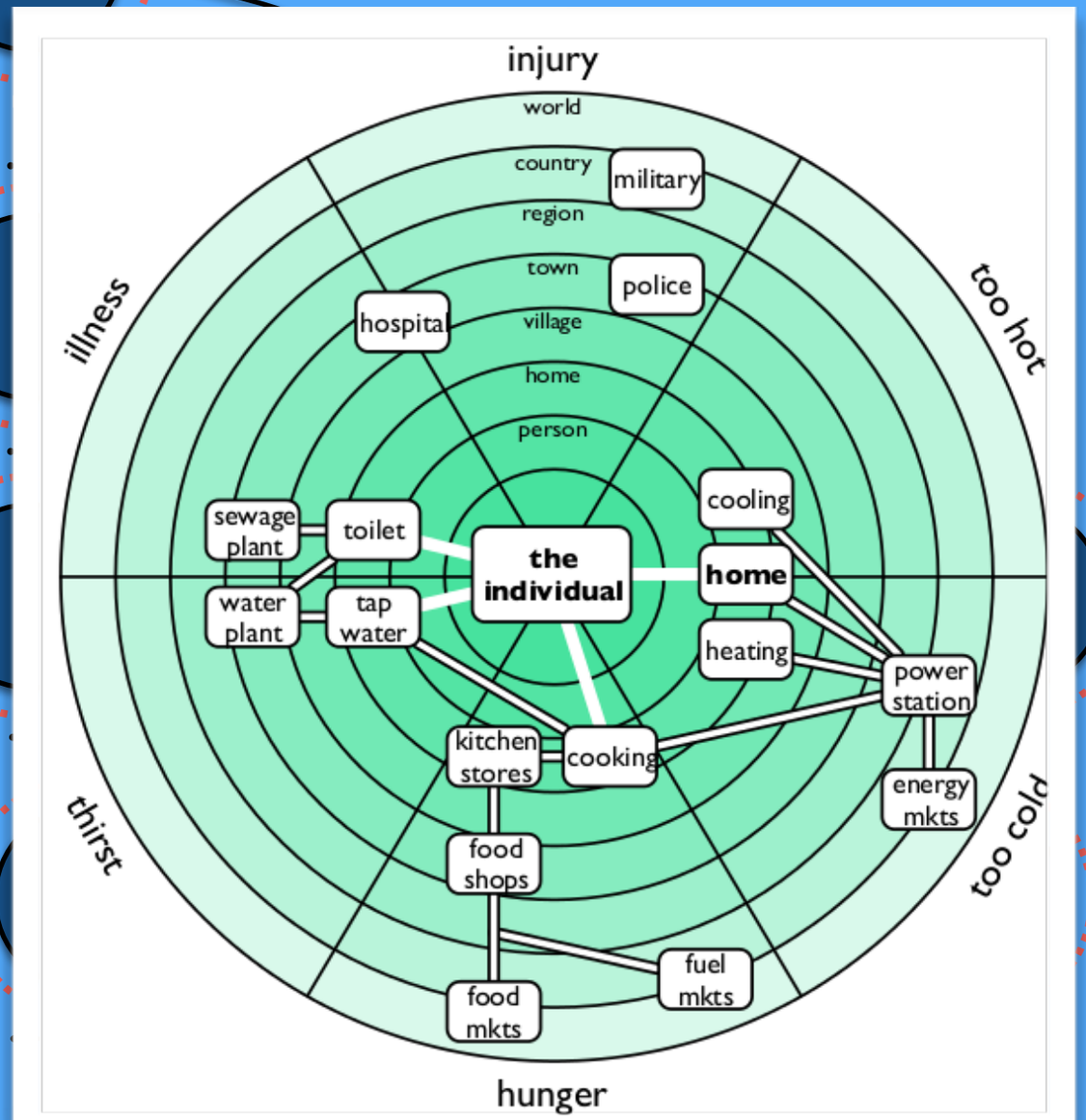
Object

Critical state that needs protection

Thread boundary

Error Kernel

Supervision



Demo Time

LET'S MODEL A RESILIENT VENDING MACHINE, IN AKKA

Demo Runner

```
object VendingMachineDemo extends App {

  val system = ActorSystem("vendingMachineDemo")
  val coffeeMachine = system.actorOf(Props[CoffeeMachineManager], "coffeeMachineManager")
  val customer = Inbox.create(system) // emulates the customer

  … // test runs

  system.shutdown()
}
```

https://gist.github.com/jboner/d24c0eb91417a5ec10a6

Test Happy Path

```
// Insert 2 coins and get an Espresso
customer.send(coffeeMachine, Coins(2))
customer.send(coffeeMachine, Selection(Espresso))
val Beverage(coffee1) = customer.receive(5.seconds)
println(s"Got myself an $coffee1")
assert(coffee1 == Espresso)
```

https://gist.github.com/jboner/d24c0eb91417a5ec10a6

Test User Error

```
customer.send(coffeeMachine, Coins(1))
customer.send(coffeeMachine, Selection(Latte))
val NotEnoughCoinsError(message) = customer.receive(5.seconds)
println(s"Got myself a validation error: $message")
assert(message == "Please insert [1] coins")
```

https://gist.github.com/jboner/d24c0eb91417a5ec10a6

Test System Failure

```
// Insert 1 coin (had 1 before) and try to get my Latte
// Machine should:
//   1. Fail
//   2. Restart
//   3. Resubmit my order
//   4. Give me my coffee
customer.send(coffeeMachine, Coins(1))
customer.send(coffeeMachine, TriggerOutOfCoffeeBeansFailure)
customer.send(coffeeMachine, Selection(Latte))
val Beverage(coffee2) = customer.receive(5.seconds)
println(s"Got myself a $coffee2")
assert(coffee2 == Latte)
```

https://gist.github.com/jboner/d24c0eb91417a5ec10a6

Protocol

```
// Coffee types
trait CoffeeType
case object BlackCoffee extends CoffeeType
case object Latte extends CoffeeType
case object Espresso extends CoffeeType

// Commands
case class Coins(number: Int)
case class Selection(coffee: CoffeeType)
case object TriggerOutOfCoffeeBeansFailure

// Events
case class CoinsReceived(number: Int)

// Replies
case class Beverage(coffee: CoffeeType)

// Errors
case class NotEnoughCoinsError(message: String)

// Failures
case class OutOfCoffeeBeansFailure(customer: ActorRef,
                                   pendingOrder: Selection,
                                   nrOfInsertedCoins: Int) extends Exception
```

https://gist.github.com/jboner/d24c0eb91417a5ec10a6

CoffeeMachine

```
class CoffeeMachine extends Actor {
  val price = 2
  var nrOfInsertedCoins = 0
  var outOfCoffeeBeans = false
  var totalNrOfCoins = 0

  def receive = { … }

  override def postRestart(failure: Throwable): Unit = { … }
}
```

https://gist.github.com/jboner/d24c0eb91417a5ec10a6

CoffeeMachine

```
def receive = {
  case Coins(nr) =>
    nrOfInsertedCoins += nr
    totalNrOfCoins += nr
    println(s"Inserted [$nr] coins")
    println(s"Total number of coins in machine is [$totalNrOfCoins]")

  case selection @ Selection(coffeeType) =>
     if (nrOfInsertedCoins < price)
       sender.tell(NotEnoughCoinsError(
         s”Insert [${price - nrOfInsertedCoins}] coins"), self)
     else {
       if (outOfCoffeeBeans)
         throw new OutOfCoffeeBeansFailure(sender, selection, nrOfInsertedCoins)
       println(s"Brewing your $coffeeType")
       sender.tell(Beverage(coffeeType), self)
       nrOfInsertedCoins = 0
     }

  case TriggerOutOfCoffeeBeansFailure =>
    outOfCoffeeBeans = true
}
```

https://gist.github.com/jboner/d24c0eb91417a5ec10a6

CoffeeMachine

```
override def postRestart(failure: Throwable): Unit = {
  println(s"Restarting coffee machine...")
  failure match {
    case OutOfCoffeeBeansFailure(customer, pendingOrder, coins) =>
      nrOfInsertedCoins = coins
      outOfCoffeeBeans = false
      println(s"Resubmitting pending order $pendingOrder")
      context.self.tell(pendingOrder, customer)
  }
}
```

https://gist.github.com/jboner/d24c0eb91417a5ec10a6

Supervisor

```
class CoffeeMachineManager extends Actor {
  override val supervisorStrategy =
    OneForOneStrategy(maxNrOfRetries = 10, withinTimeRange = 1.minute) {
      case e: OutOfCoffeeBeansFailure =>
        println(s"ServiceGuy notified: $e")
        Restart
      case _: Exception =>
        Escalate
    }

  // to simplify things he is only managing 1 single machine
  val machine = context.actorOf(
     Props[CoffeeMachine], name = "coffeeMachine")

  def receive = {
    case request => machine.forward(request)
  }
}
```

https://gist.github.com/jboner/d24c0eb91417a5ec10a6

SO.....ARE WE DONE?

SORRY...BUT NOT REALLY.

We can not keep putting all eggs in the same basket


The Network
is Reliable
NOT
Really

HERE, WE ARE LIVING IN THE

Looming Shadow of Impossibility Theorems

FLP: CONSENSUS IS IMPOSSIBLE

CAP: CONSISTENCY IS IMPOSSIBLE

Towards Resilient Distributed Systems

EMBRACE THE NETWORK

- Asynchronicity
- Location Transparency

ISOLATION

- Autonomous Microservices
- Resilient Protocols
- Virtualization

SELF-HEALING

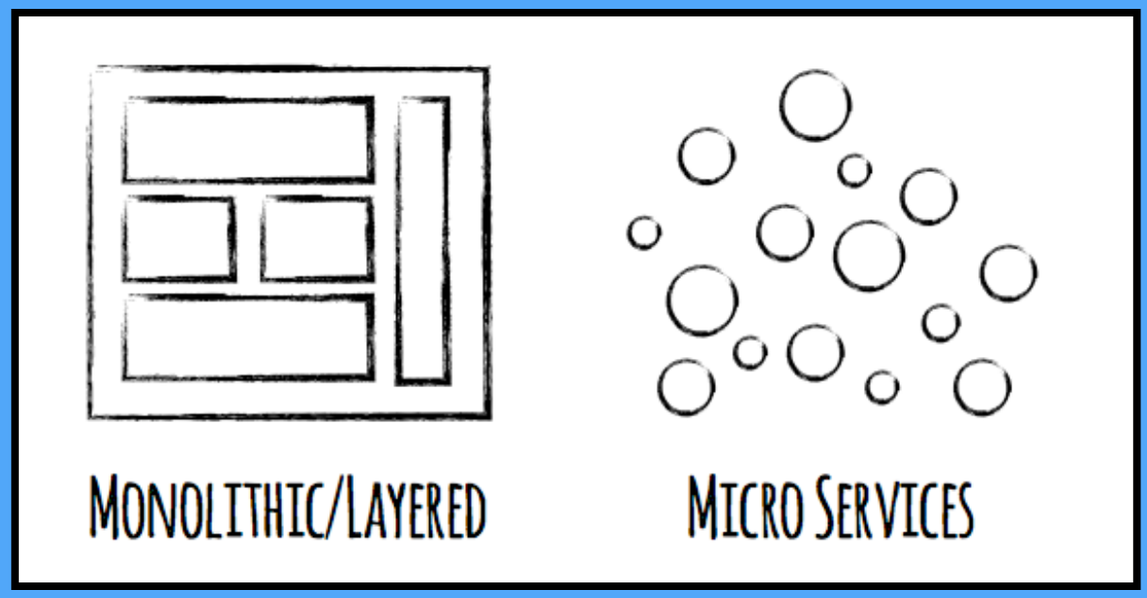
- Decentralized Architectures
- Gossip Protocols
- Failure Detection

DATA RESILIENCE

- Eventual & Causal Consistency
- Event Logging
- Flow Control / Feedback Control

Microservices

1. AUTONOMY
2. ISOLATION
3. MOBILITY
4. SINGLE RESPONSIBILITY
5. EXCLUSIVE STATE



Apply Promise Theory

AN AUTONOMOUS SERVICE
CAN ONLY PROMISE
ITS OWN BEHAVIOR

We need to decompose the system using Consistency Boundaries

Inside Data

OUR CURRENT PRESENT—STATE

Outside Data

BLAST FROM THE PAST—FACTS

Between Services

HOPE FOR THE FUTURE—COMMANDS

Data on the inside vs Data on the outside - Pat Helland

WITHIN the Consistency Boundary we can have STRONG CONSISTENCY

BETWEEN

Consistency
Boundaries
it is a

ZOO

WE NEED SYSTEMS THAT ARE DECOUPLED IN Time and Space

Embrace the Network

- GO ASYNCHRONOUS
- MAKE DISTRIBUTION FIRST CLASS
- LEARN FROM THE MISTAKES OF RPC, EJB & CORBA
- LEVERAGE LOCATION TRANSPARENCY
- ACTOR MODEL DOES IT RIGHT

Location Transparency

ONE COMMUNICATION ABSTRACTION ACROSS ALL DIMENSIONS OF SCALE

CORE ⇒ SOCKET ⇒ CPU ⇒

CONTAINER ⇒ SERVER ⇒ RACK ⇒

DATA CENTER ⇒ GLOBAL

Resilient Protocols

DEPEND ON

- ASYNCHRONOUS COMMUNICATION
- EVENTUAL CONSISTENCY

ARE TOLERANT TO

- MESSAGE LOSS
- MESSAGE REORDERING
- MESSAGE DUPLICATION

EMBRACE ACID 2.0

- ASSOCIATIVE
- COMMUTATIVE
- IDEMPOTENT
- DISTRIBUTED

"To make a system of interconnected components crash-only, it must be designed so that components can tolerate the crashes and temporary unavailability of their peers. This means we require: [1] strong modularity with relatively impermeable component boundaries, [2] timeout-based communication and lease-based resource allocation, and [3] self-describing requests that carry a time-to-live and information on whether they are idempotent."

- GEORGE CANDEA, ARMANDO FOX

Crash-Only Software - George Candea, Armando Fox

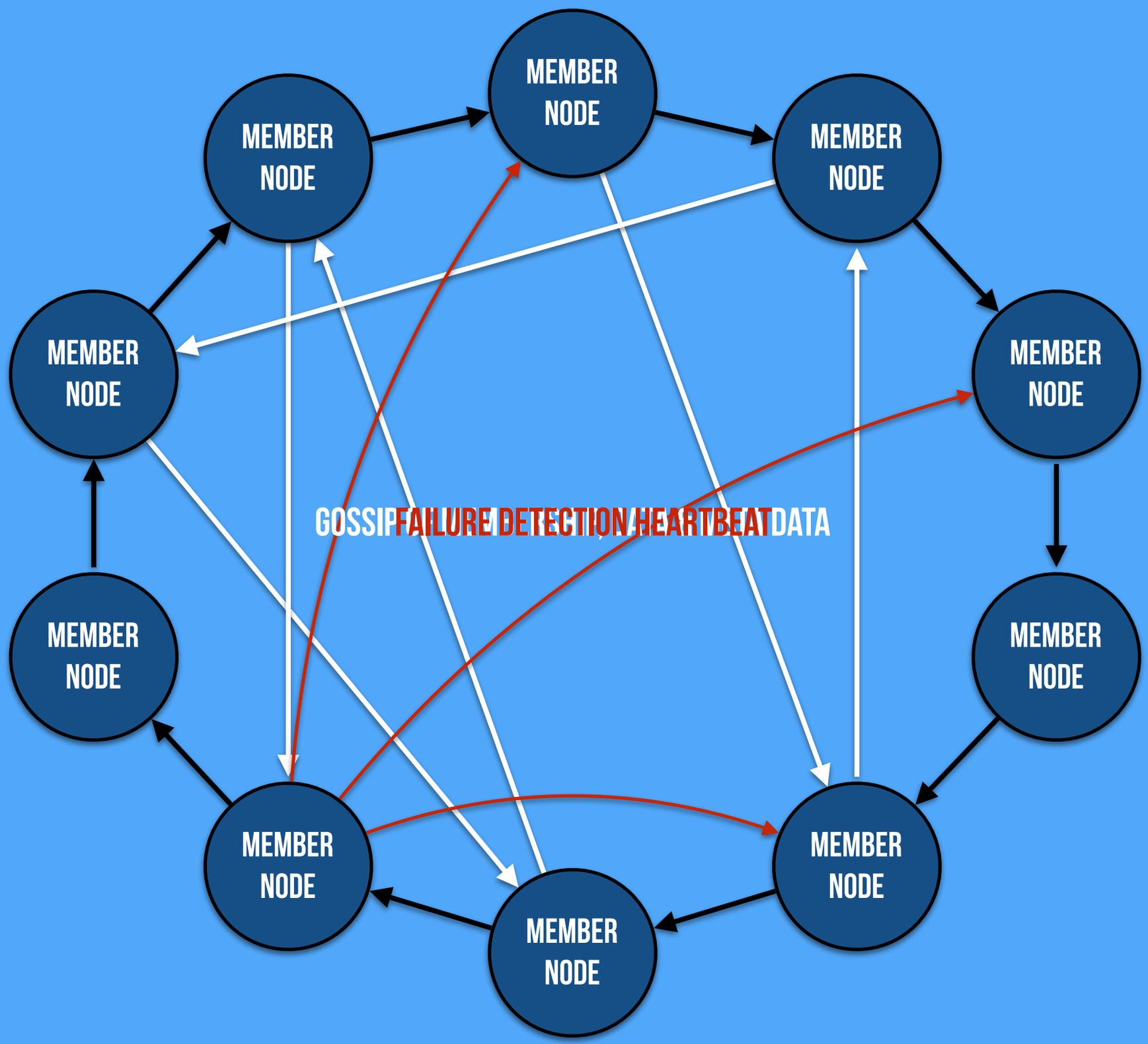
"Software components should be designed such that they can deny service for any request or call. Then, if an underlying component can say No, apps must be designed to take No for an answer and decide how to proceed: give up, wait and retry, reduce fidelity, etc."

- GEORGE CANDEA, ARMANDO FOX

Recursive Restartability: Turning the Reboot Sledgehammer into a Scalpel - George Candea, Armando Fox

Services need to learn to accept NO for an answer

Decentralized
Epidemic Gossip Protocols



STRONG

Consistency

IS THE WRONG DEFAULT

“Two-phase commit is the anti-availability protocol.”

- PAT HELLAND

WE HAVE TO RELY ON

Eventual Consistency

BUT RELAX, IT'S HOW THE WORLD WORKS

But I really need Transactions

“In general, application developers simply do not implement large scalable applications assuming distributed transactions.”

- PAT HELLAND

USE A PROTOCOL OF

Guess.

Apologize.

Compensate.

“The truth is the log. The database is a cache of a subset of the log.”

- PAT HELLAND

CRUD is DEAD


In loving memory of two
operations that changed the
world.
UPDATE and DELETE
1983 - 2013

Event Logging

- Work with facts—immutable values
- Event sourcing
- DB of facts—keep all history
- Just replay on failure
- Free auditing, debugging, replication
- Single writer principle
- Avoids OO-relational impedence mismatch
- CQRS—separate the read & write model

Demo Time

LET'S MODEL A RESILIENT & EVENT LOGGED VENDING MACHINE, IN AKKA

Event Logged CoffeeMachine

```
// Events
case class CoinsReceived(number: Int)

class CoffeeMachine extends PersistentActor {
  val price = 2
  var nrOfInsertedCoins = 0
  var outOfCoffeeBeans = false
  var totalNrOfCoins = 0

  override def persistenceId = "CoffeeMachine"

  override def receiveCommand: Receive = {
    case Coins(nr) =>
      nrOfInsertedCoins += nr
      println(s"Inserted [$nr] coins")
      persist(CoinsReceived(nr)) { evt =>
        totalNrOfCoins += nr
        println(s"Total number of coins in machine is [$totalNrOfCoins]")
      }
    …

  }

  override def receiveRecover: Receive = {
    case CoinsReceived(coins) =>
      totalNrOfCoins += coins
      println(s"Total number of coins in machine is [$totalNrOfCoins]")
  }
}
```

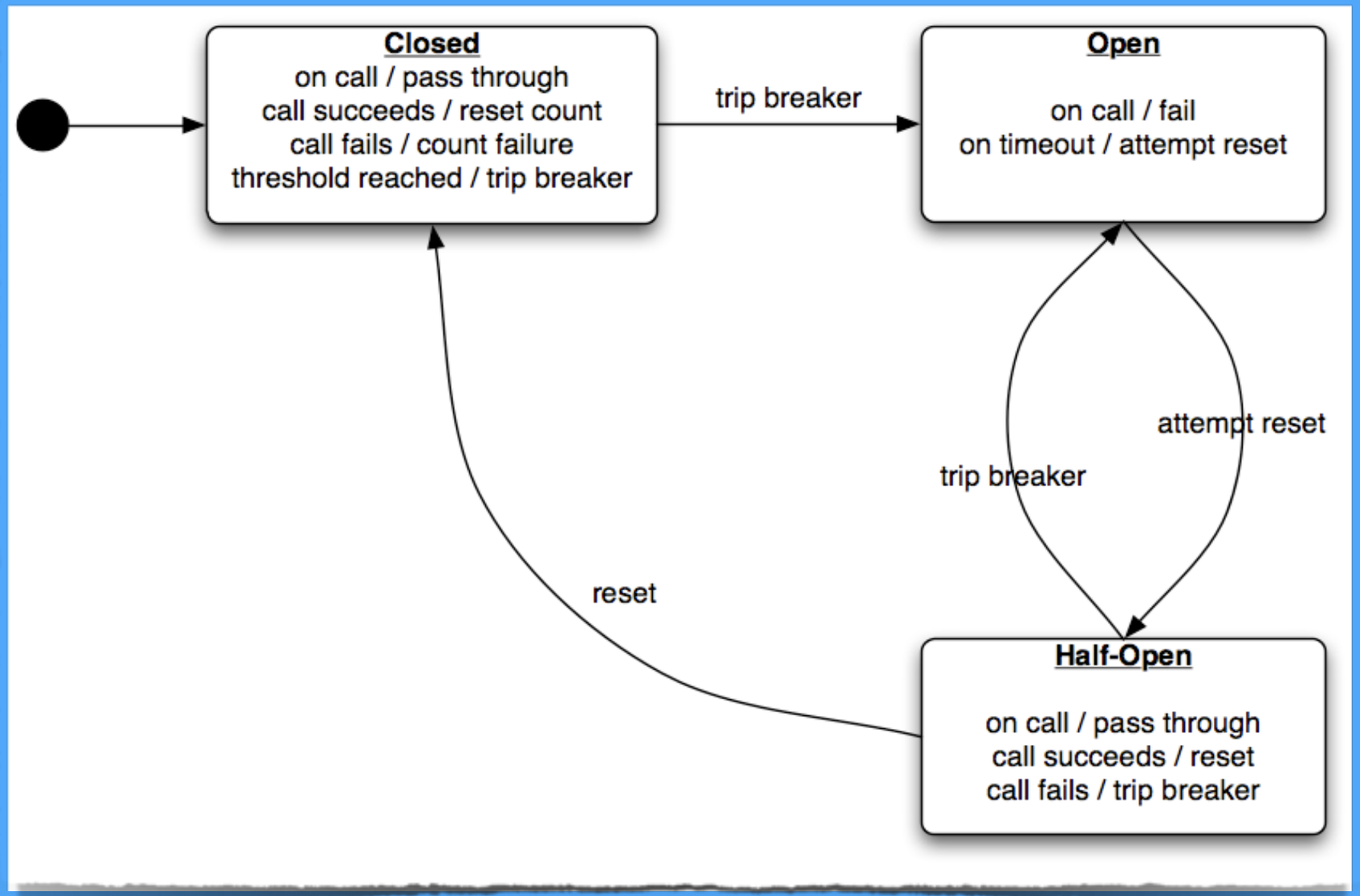
https://gist.github.com/jboner/1db37eeee3ed3c9422e4

"An escalator can never break: it can only become stairs. You should never see an Escalator Temporarily Out Of Order sign, just Escalator Temporarily Stairs. Sorry for the convenience."

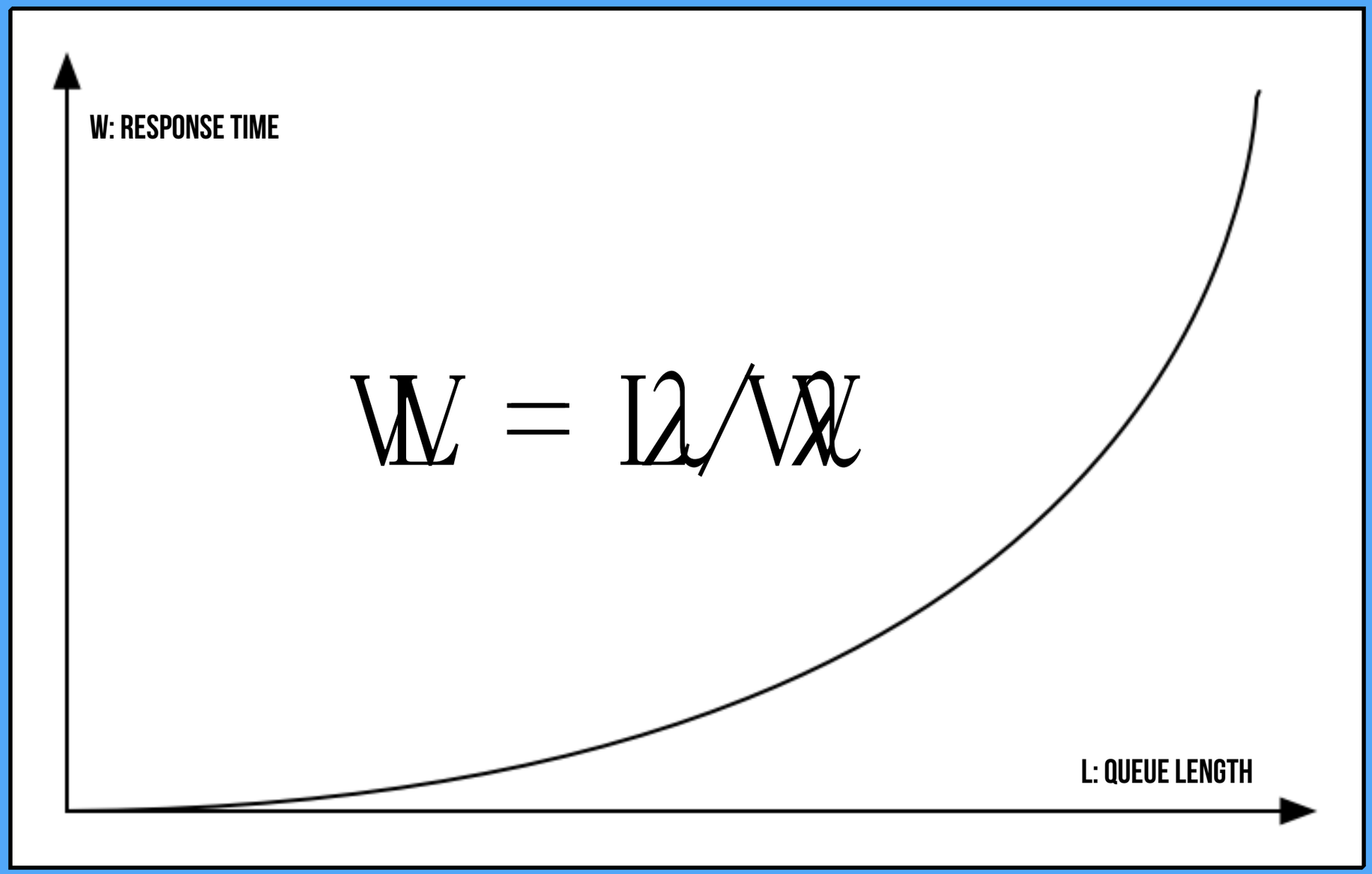
- MITCH HEDBERG

Graceful Degradation

Circuit Breaker



Little's Law



RESPONSE TIME = AVERAGE QUEUE LENGTH / RESPONSE RATE

Flow Control

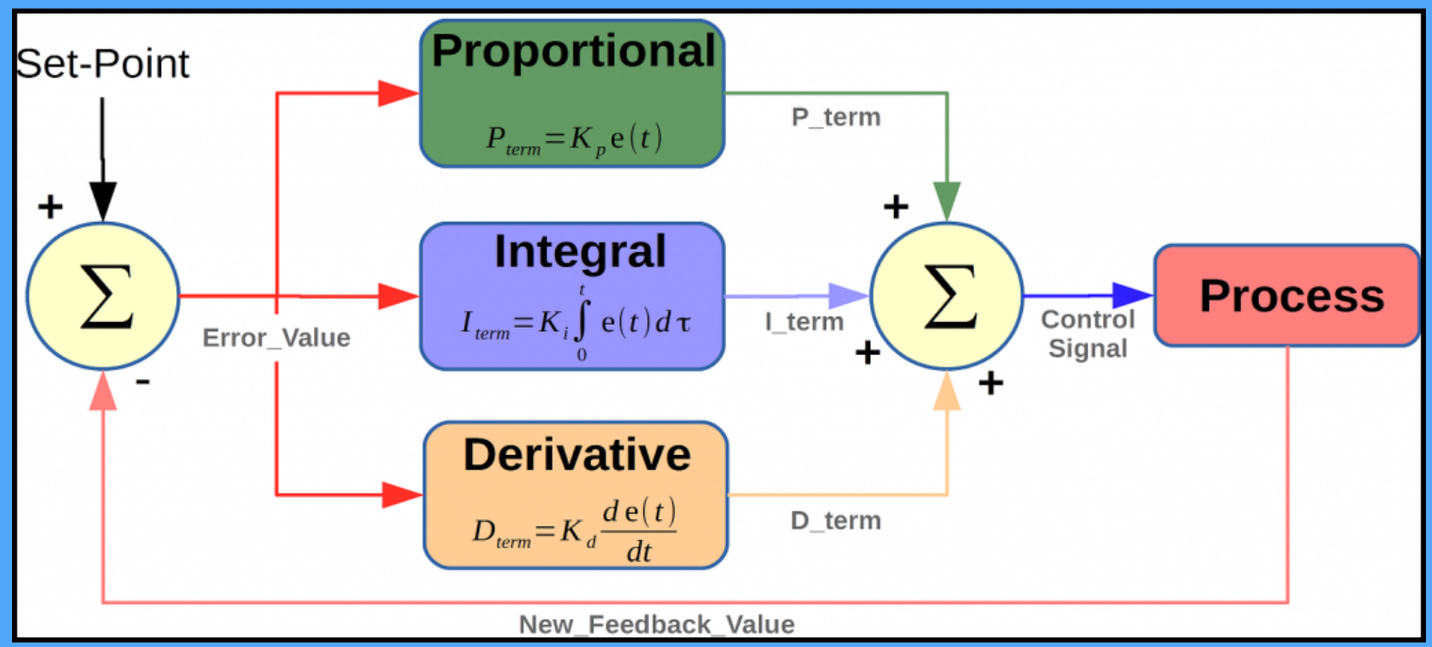
Feedback Control

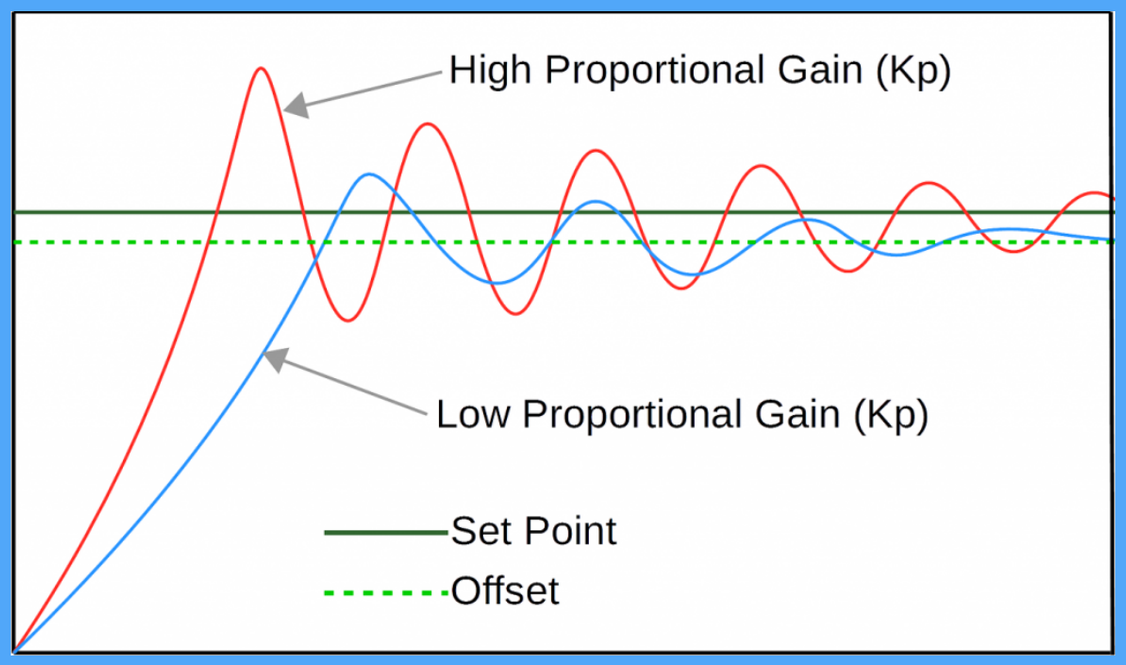
THE FEEDBACK PRINCIPLE

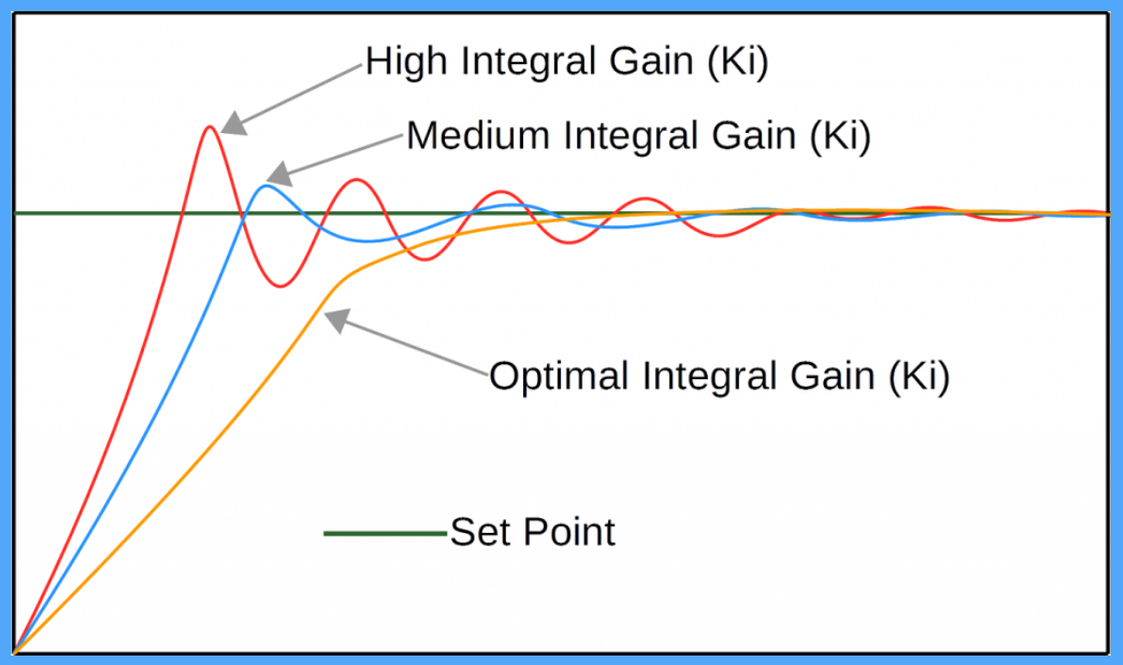
"Continuously compare the actual output to its desired reference value; then apply a change to the system inputs that counteracts any deviation of the actual output from the reference."

- PHILIPP K. JANERT

Feedback Control







Influencing a Complex System

Places to Intervene in a Complex System

1. The constants, parameters or numbers
2. The sizes of buffers relative to their flows
3. The structure of material stocks and flows
4. The lengths of delays, relative to the rate of system change
5. The strength of negative feedback loops
6. The gain around driving positive feedback loops
7. The structure of information flows
8. The rules of the system
9. The power to add, change, evolve, or self-organize structure
10. The goals of the system
11. The mindset or paradigm out of which the system arises
12. The power to transcend paradigms

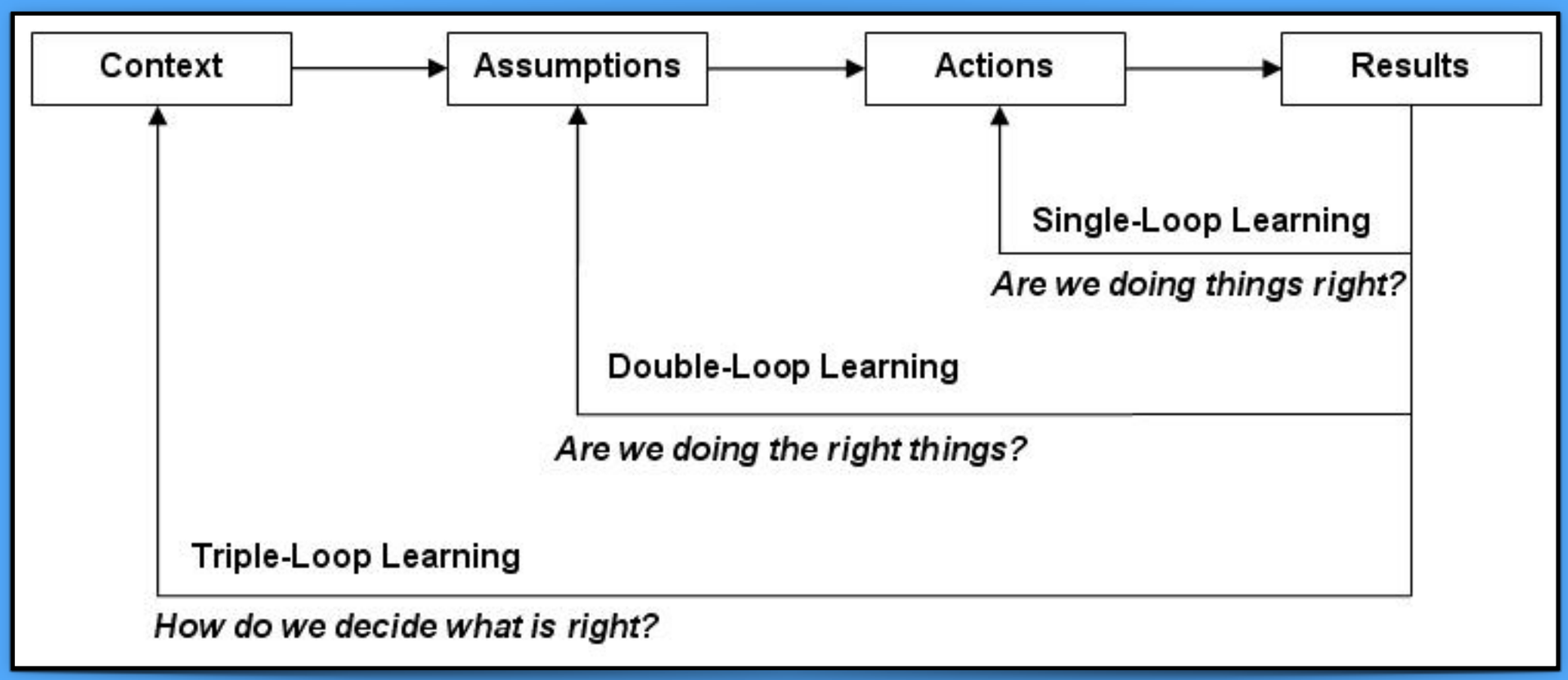
Leverage Points: Places to Intervene in a System - Donella Meadows:

Triple Loop Learning

LOOP 1: FOLLOW THE RULES

LOOP 2: CHANGE THE RULES

LOOP 3: LEARN HOW TO LEARN



Triple Loop Learning - Chris Argyris

Testing


YOU SCHEDULED THE END OF THE TEST PHASE AFTER THE START OF THE PRODUCTION PHASE.
WE'RE FEELING CONFIDENT.
IT'S TOO BAD THAT BEING SMART DOESN'T COME WITH SOME SORT OF GOOD FEELING LIKE THAT.
Dilbert.com DilbertCartoonist@gmail.com
5-17-11 ©2011 Scott Adams, Inc. Dist. by Universal Uclick

WHAT CAN WE LEARN FROM ARNOLD?

BLOW THINGS UP

Shoot Your App Down

Pull the Plug

...AND SEE WHAT HAPPENS

akka

Executive Summary

“Complex systems run as broken systems.”

- RICHARD COOK

How Complex Systems Fail - Richard Cook

Resilience is by Design

Photo courtesy of FEMA/Joselyne Augustino

Without Resilience Nothing Else Matters

References

- **Drift into Failure** - http://www.amazon.com/Drift-into-Failure-Components-Understanding-ebook/dp/B009KOKXKY
- **How Complex Systems Fail** - http://web.mit.edu/2.75/resources/random/How%20Complex%20Systems%20Fail.pdf
- **Leverage Points: Places to Intervene in a System** - http://www.donellameadows.org/archives/leverage-points-places-to-intervene-in-a-system/
- **Going Solid: A Model of System Dynamics and Consequences for Patient Safety** - http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1743994/
- **Resilience in Complex Adaptive Systems: Operating at the Edge of Failure** - https://www.youtube.com/watch?v=PGLYEDpNu60
- **Puppies! Now that I've got your attention, Complexity Theory** - https://www.ted.com/talks/nicolas_perony_puppies_now_that_i_ve_got_your_attention_complexity_theory
- **How Bacteria Becomes Resistant** - http://www.abc.net.au/science/slab/antibiotics/resistance.htm
- **Towards Resilient Architectures: Biology Lessons** - http://www.metropolismag.com/Point-of-View/March-2013/Toward-Resilient-Architectures-1-Biology-Lessons/
- **Dealing in Security** - http://resiliencemaps.org/files/Dealing_in_Security.July2010.en.pdf
- **What is resilience? An introduction to social-ecological research** - http://www.stockholmresilience.org/download/18.10119fc11455d3c557d6d21/1398172490555/SU_SRC_whatisresilience_sidaApril2014.pdf
- **Applying resilience thinking: Seven principles for building resilience in social-ecological systems** - http://www.stockholmresilience.org/download/18.10119fc11455d3c557d6928/1398150799790/SRC+Applying+Resilience+final.pdf
- **Crash-Only Software** - https://www.usenix.org/legacy/events/hotos03/tech/full_papers/candea/candea.pdf
- **Recursive Restartability: Turning the Reboot Sledgehammer into a Scalpel** - http://roc.cs.berkeley.edu/papers/recursive_restartability.pdf
- **Out of the Tar Pit** - http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.93.8928
- **Bulkhead Pattern** - http://skife.org/architecture/fault-tolerance/2009/12/31/bulkheads.html
- **Making Reliable Distributed Systems in the Presence of Software Errors** - http://www.erlang.org/download/armstrong_thesis_2003.pdf
- **On Erlang, State and Crashes** - http://jlouisramblings.blogspot.be/2010/11/on-erlang-state-and-crashes.html
- **Akka Supervision** - http://doc.akka.io/docs/akka/snapshot/general/supervision.html
- **Release It!: Design and Deploy Production-Ready Software** - https://pragprog.com/book/mnee/release-it
- **Feedback Control for Computer Systems** - http://www.amazon.com/Feedback-Control-Computer-Systems-Philipp/dp/1449361692
- **The Network in Reliable** - http://queue.acm.org/detail.cfm?id=2655736
- **Data on the Outside vs Data on the Inside** - https://msdn.microsoft.com/en-us/library/ms954587.aspx
- **Life Beyond Distributed Transactions** - http://adrianmarriott.net/logosroot/papers/LifeBeyondTxns.pdf
- **Immutability Changes Everything** - http://cidrdb.org/cidr2015/Papers/CIDR15_Paper16.pdf
- **Standing on Distributed Shoulders of Giants** - https://queue.acm.org/detail.cfm?id=2953944
- **Thinking in Promises** - http://shop.oreilly.com/product/0636920036289.do
- **In Search Of Certainty** - http://shop.oreilly.com/product/0636920038542.do
- **Reactive Microservices Architecture** - http://www.oreilly.com/programming/free/reactive-microservices-architecture-orm.csp
- **Reactive Streams** - http://reactive-streams.org
- **Vending Machine Akka Supervision Demo** - https://gist.github.com/jboner/d24c0eb91417a5ec10a6
- **Persistent Vending Machine Akka Supervision Demo** - https://gist.github.com/jboner/1db37eeee3ed3c9422e4

Thank You

Without Resilience Nothing Else Matters

JONAS BONÉR
CTO Lightbend
@jboner