AWS re:Invent

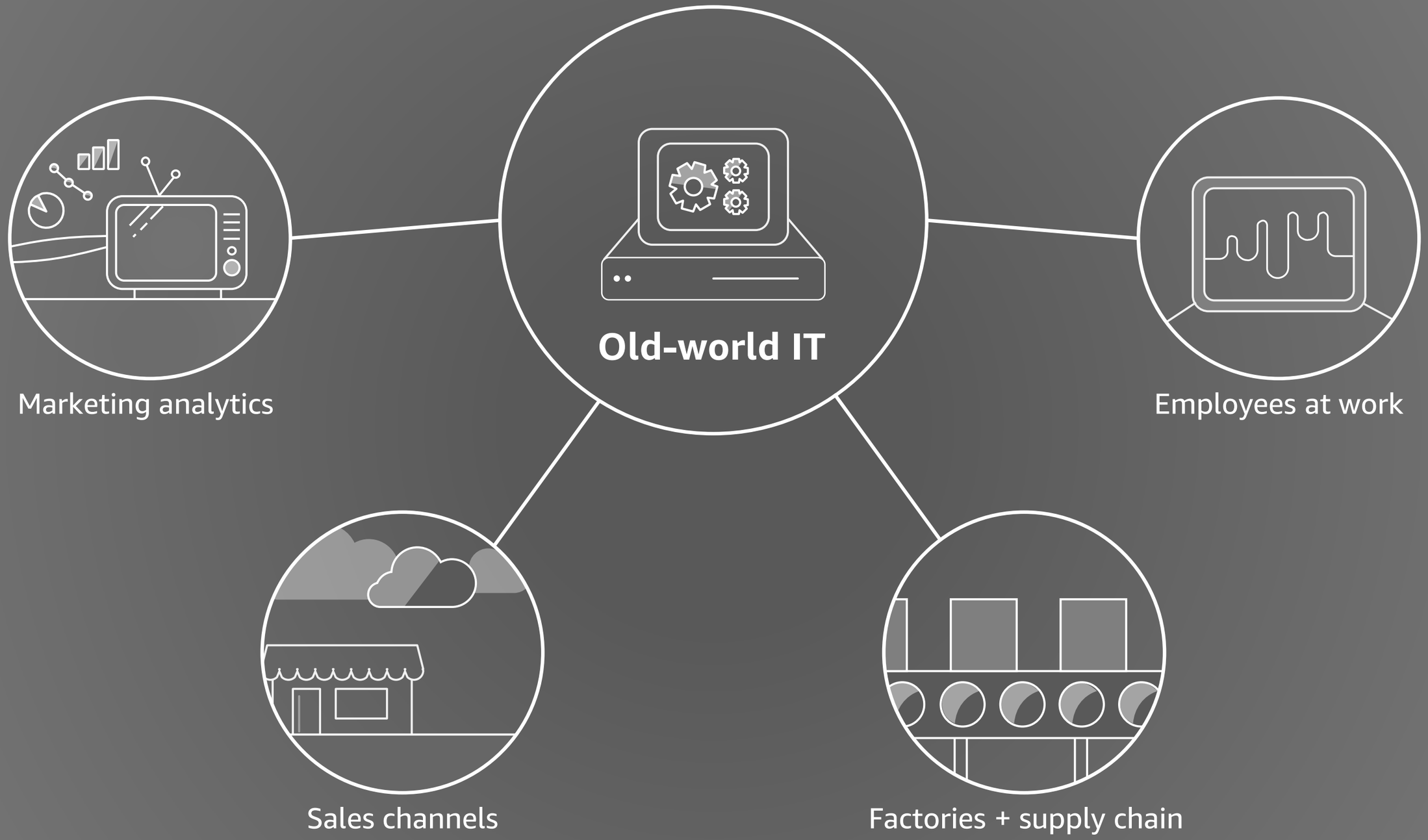
ARC203

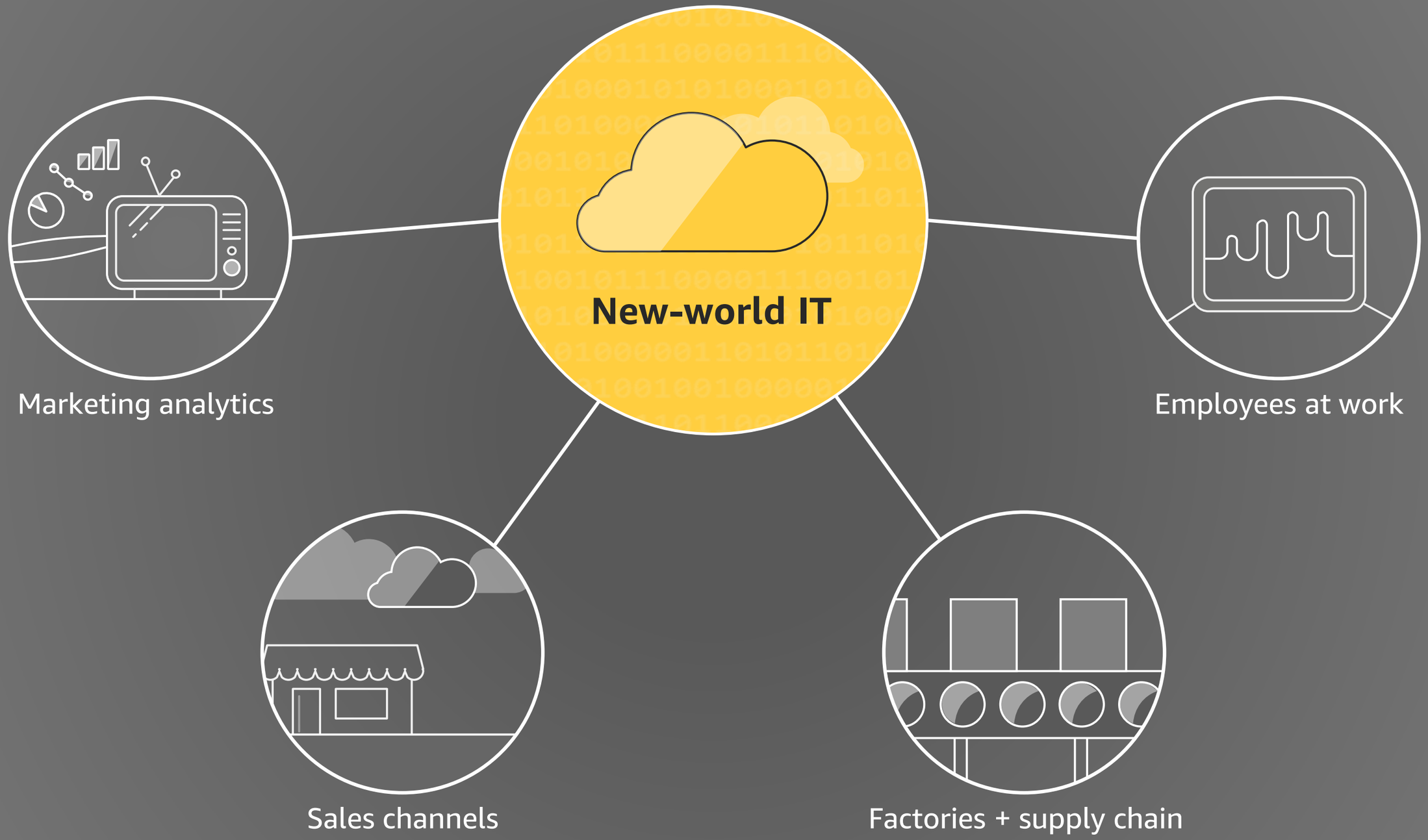
Innovation at speed

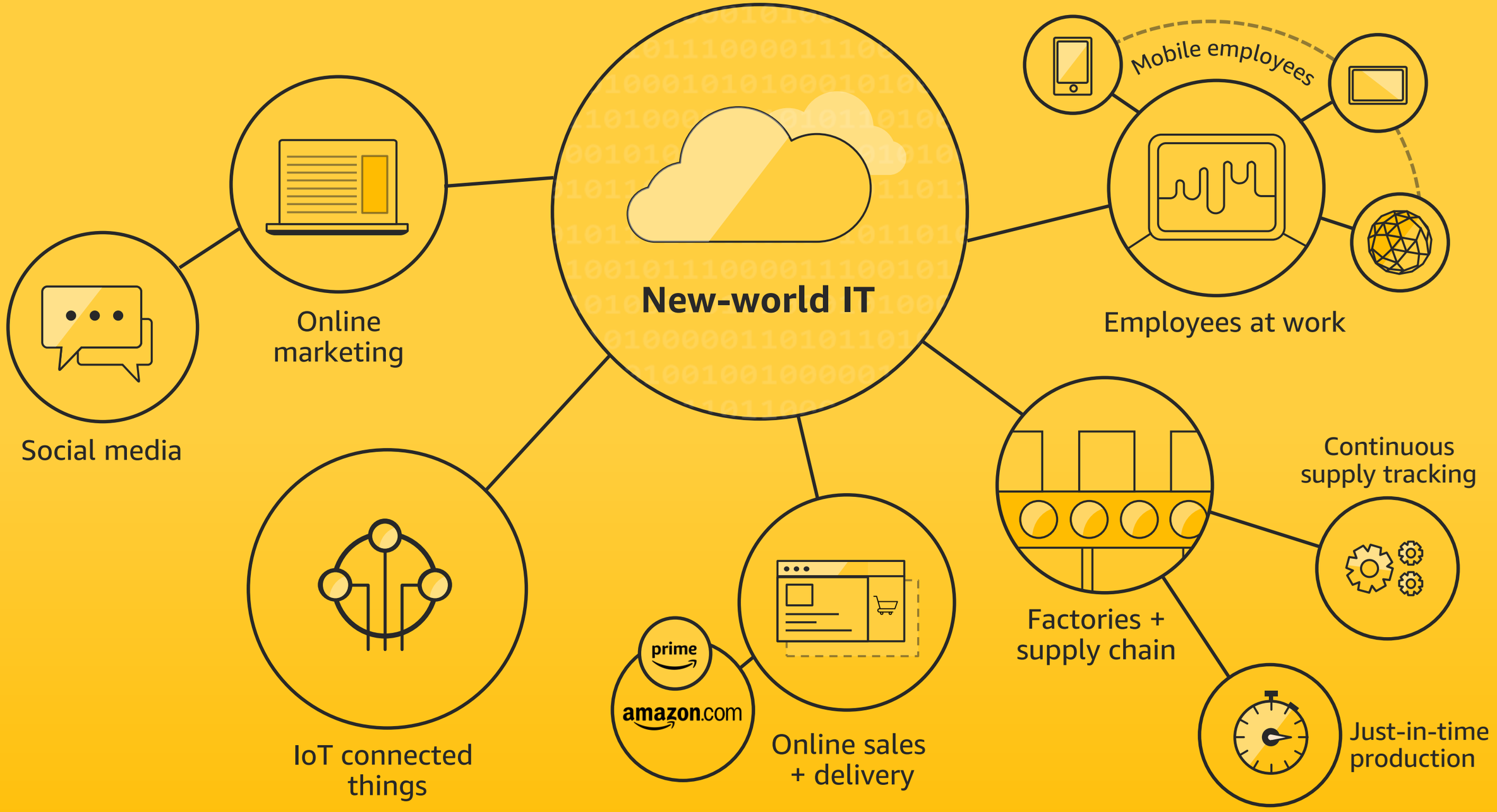
Adrian Cockcroft
VP, Cloud Architecture Strategy
Amazon Web Services

AWS re:Invent

aws


Old-world IT
Marketing analytics
Employees at work
Sales channels
Factories + supply chain


New-world IT
Marketing analytics
Employees at work
Sales channels
Factories + supply chain


New-world IT
Mobile employees
Employees at work
Online marketing
Social media
IoT connected things
prime
amazon.com
Online sales + delivery
Factories + supply chain
Continuous supply tracking
Just-in-time production

New needs

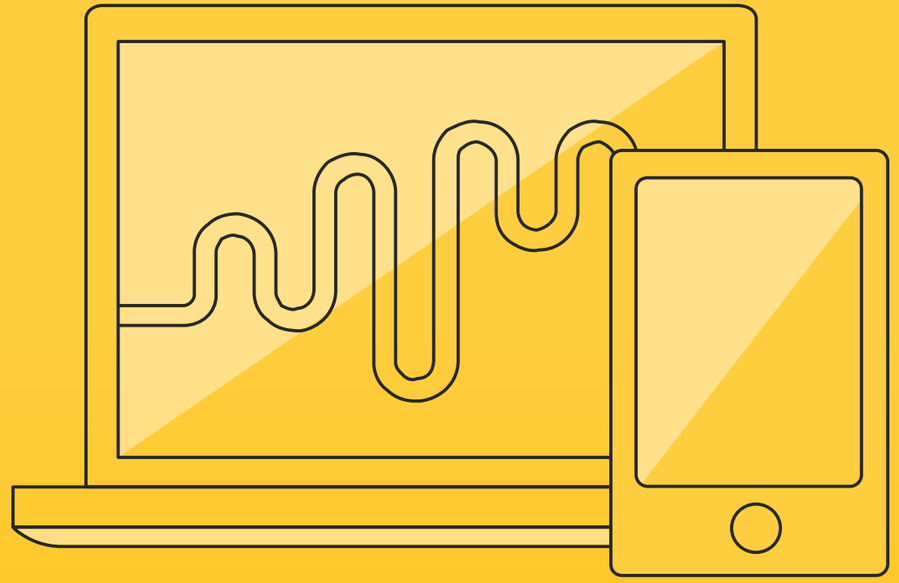
Personalization

Customer analytics

New channels direct to customer

More things, more scale, rapid change

AWS: Unblocking innovation for digital transformation with enterprise customers



Blockers for innovation

Culture	Skills	Organization	Risk
Leadership systems and feedback	Training and compensation	Silos project to product	Finance and board level concerns

Leadership systems and feedback problems

Centralized, slow decision-making

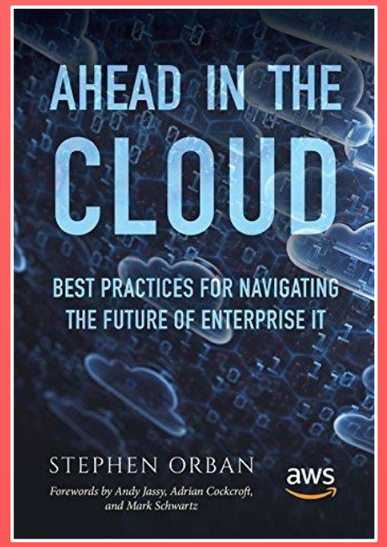
Lack of trust

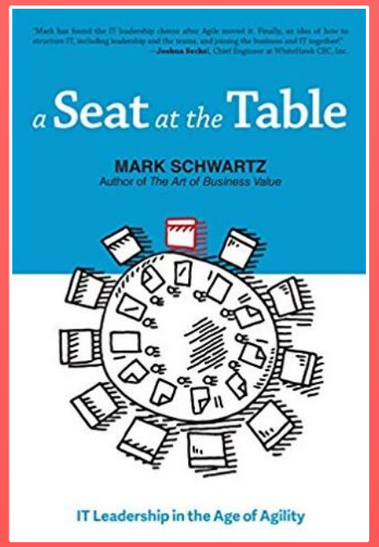
Inflexible policies and processes

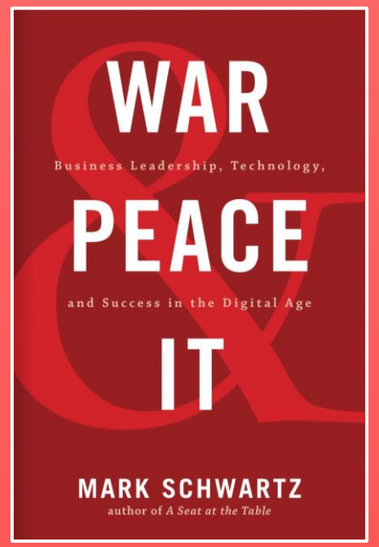
Leadership systems and feedback

Ahead in the Cloud
Stephen Orban

A Seat at the Table and *War and Peace and IT*
Mark Schwartz







Culture

If you want to build a ship, don't
drum up the people to gather wood,
divide the work, and give orders.
Instead, teach them to yearn
for the vast and endless sea.

—Antoine de Saint-Exupéry,
Author of "Le Petit Prince" ("The Little Prince")

NORDSTROM
Culture

Nordstrom Technology
NorDNA Culture Deck

NETFLIX

Culture

Seven aspects of Netflix culture

1. Values are what we value
2. High performance
3. Freedom and responsibility
4. Context, not control
5. Highly aligned, loosely coupled
6. Pay top of market
7. Promotions and development

amazon

Culture

Amazon leadership principles

- Customer obsession
- Ownership
- Invent and simplify
- Are right, a lot
- Hire and develop the best
- Insist on the highest standards
- Think big
- Bias for action
- Frugality
- Learn and be curious
- Earn trust of others
- Dive deep
- Have backbone; disagree and commit
- Deliver results

Culture

Intentional

Appropriate

Judgment

Blockers for innovation

Culture	Skills	Organization	Risk
Leadership systems and feedback	Training and compensation	Silos project to product	Finance and board level concerns

Training and compensation

Train existing staff on cloud tech

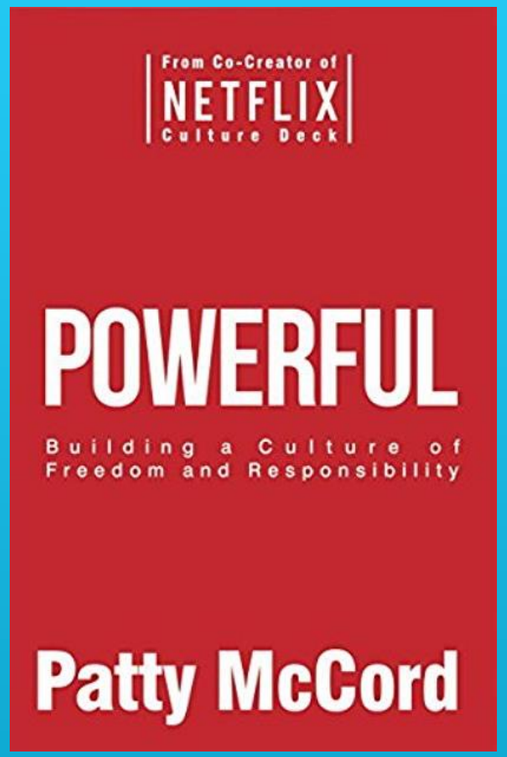
Fund pathfinder teams

Be prepared to create incentives to keep the best people after training!

Training and compensation

Get out of the way of innovation

Read the recent book *Powerful*
by Patty McCord
Ex-Netflix Chief Talent Officer



Blockers for innovation

Culture

Leadership systems and feedback

Skills

Training and compensation

Organization

Silos project to product

Risk

Finance and board-level concerns

Move from projects to product teams

Long-term product ownership

Continuous delivery

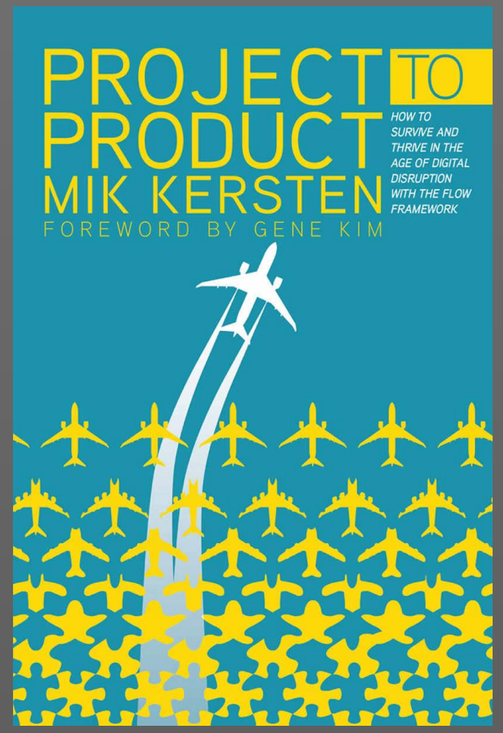
DevOps and “run what you wrote”

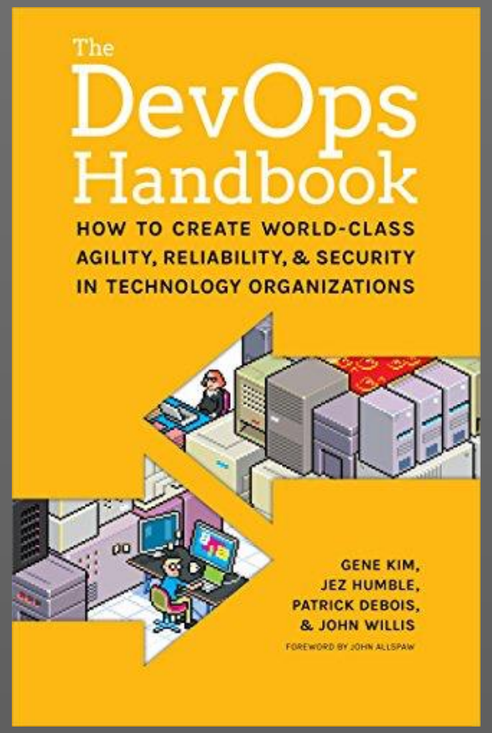
Reduce tech debt and lock-in

Move from projects to product teams

Project to Product
by Mik Kersten

The DevOps Handbook
by Gene Kim et al.





Integrate business with DevOps

AWS service teams—BusProdDevOps?

Business

Budget, headcount, goals

Product

Customer input, roadmap

Development

Continuous delivery of features

Operations

Automated global support

Integrate business with DevOps

Organization structure and APIs

- CEO
  - Sales VP
  - Marketing VP
    - Product Marketing Manager
  - Services VP
    - Services GM
      - Service Manager (two-pizza team)
        - BusProdDevOps (API)

Reporting and learning

Monday – Service team

Review last week's operations dashboards
Review last week's revenue/growth/goals

Tuesday – Groups of services

Roll-up reviews

Wednesday – CEO/VP-level view of everything

Review all operations, spin the wheel*, learnings
Review entire business revenue/growth/goals

*https://aws.amazon.com/blogs/opensource/the-wheel/

Blockers for innovation

Culture

Leadership systems and feedback

Skills

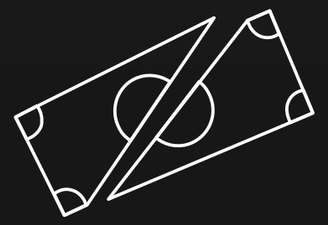
Training and compensation

Organization

Silos project to product

Risk

Finance and board-level concerns



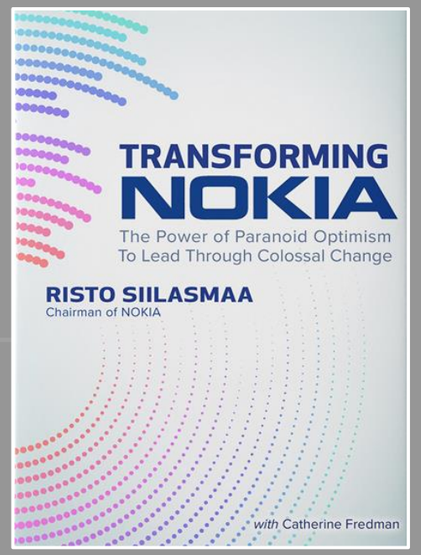
Finance – Capex vs. opex

Capitalized data center to expensed cloud

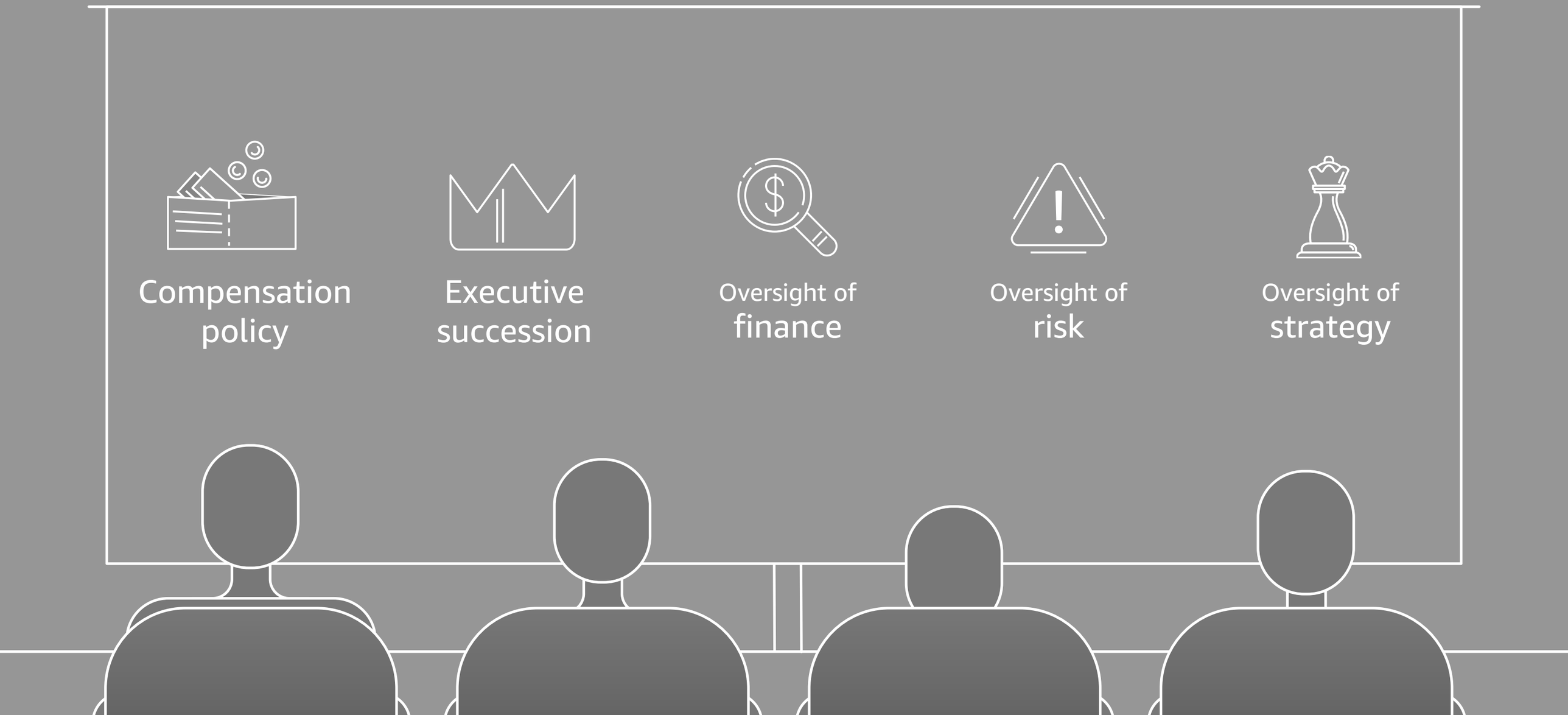
Capitalized development, expensed operations, to combined DevOps

Plan ahead, don’t surprise the CFO or your shareholders!

What is the role of boards in the long-term success of their company?



Board-level concerns



Board-level concerns

All of these affect the ability to innovate and build long-term value

Compensation policy

Executive succession

Oversight of finance

Oversight of risk

Oversight of strategy

Board-level concerns

All of these affect the ability to innovate and build long-term value

Compensation policy

Executive succession

Oversight of finance

Oversight of risk

Oversight of strategy

Many board best practices *reduce innovation*

Strategy

“Be more innovative”

Like these companies

NETFLIX amazon Capital One

Gartner study found that 47% of
CEOs face pressure from their board to digitally transform (2017)

Board-level concerns

Compensation policy

Executive succession

Oversight of finance

Oversight of risk

Oversight of strategy

So what’s the connection?
Connect the dots…

Board-level patterns

Short term focus

Capitalization issues

Limited investment

But PLEASE be more innovative

Compensation policy

Executive succession

Non technical non-founder

Oversight of finance

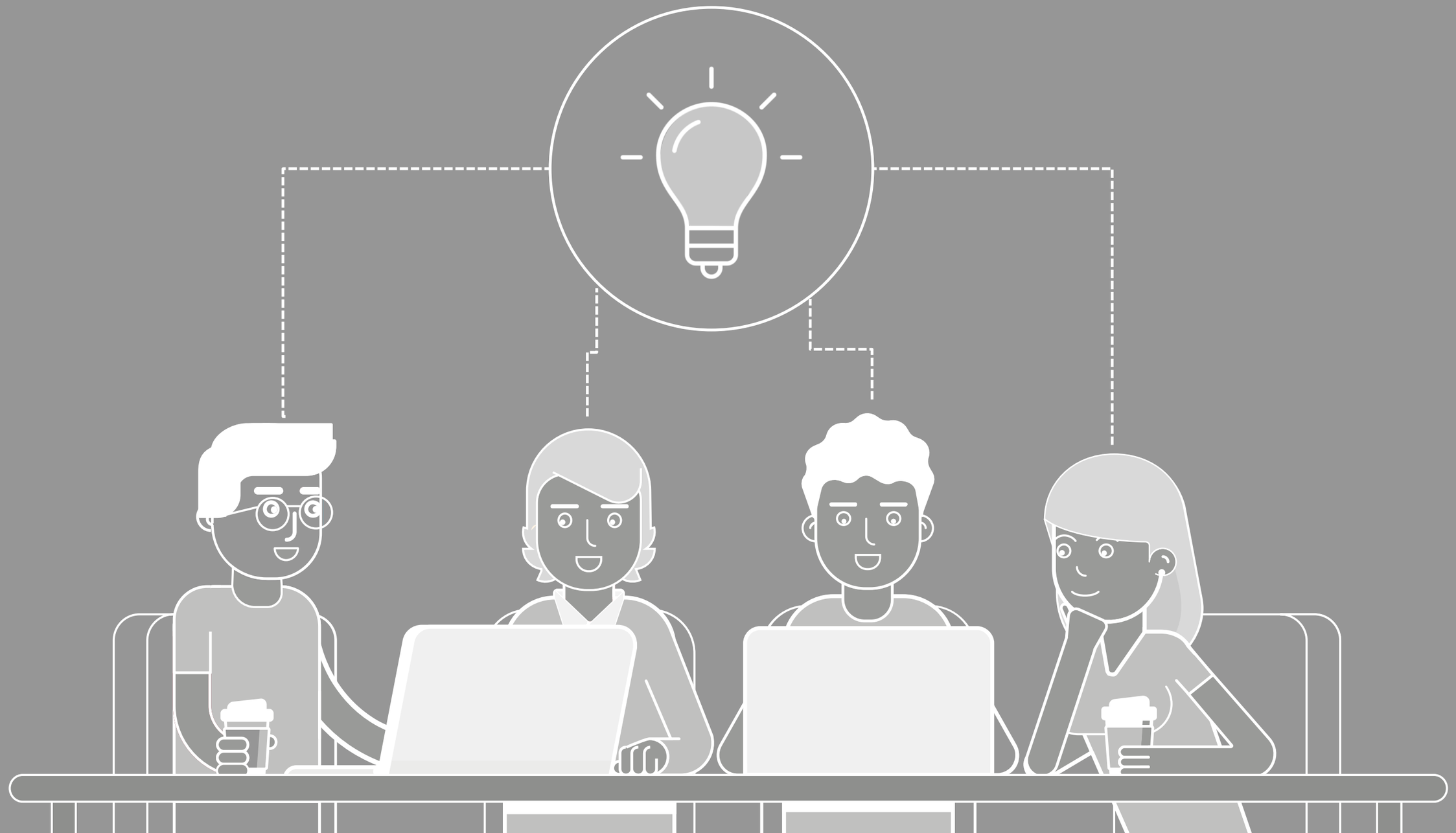
Oversight of risk

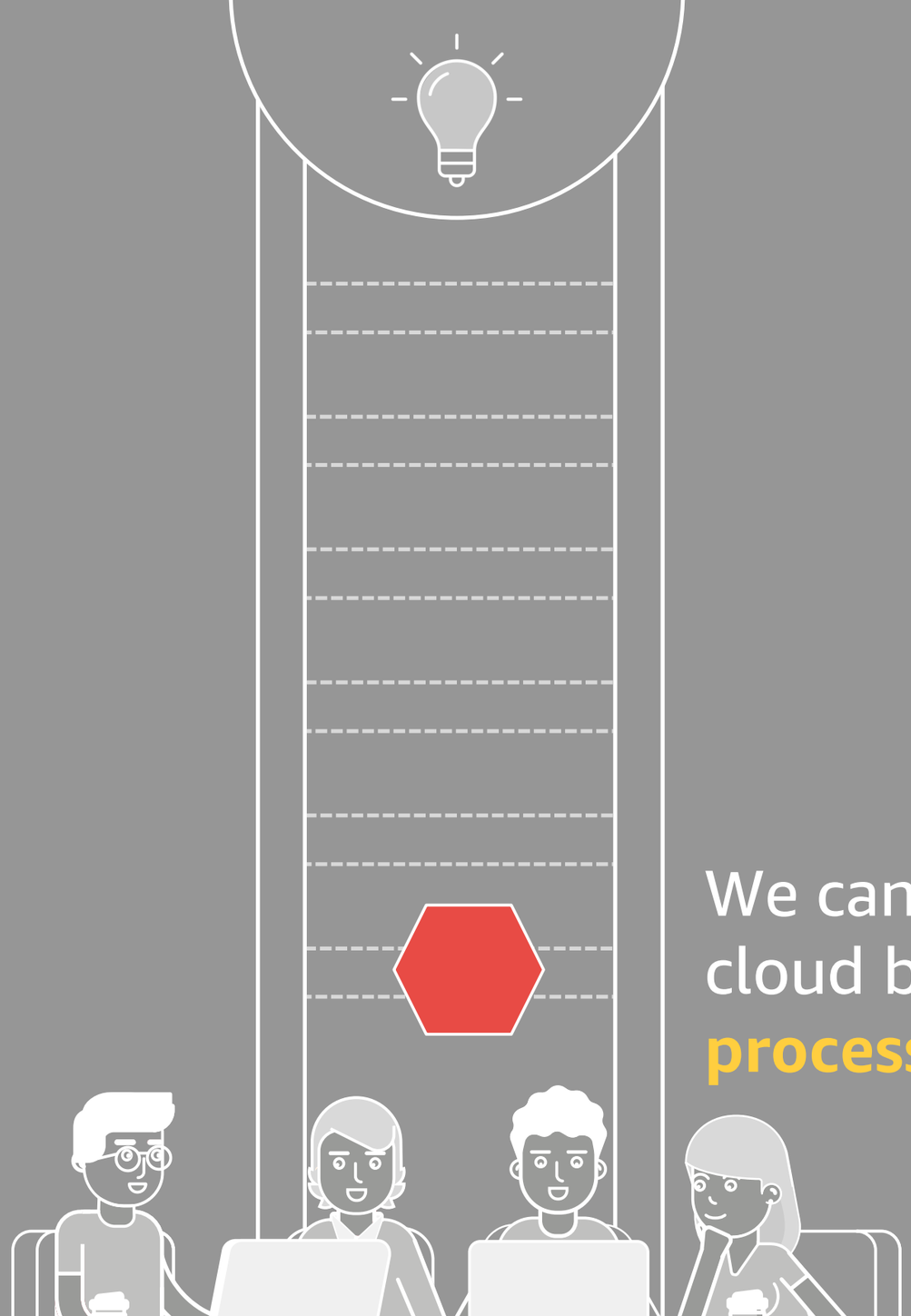
Risk adverse

NO FAILURES

Oversight of strategy

Product team would like to leverage technologies (e.g., cloud) to make a better product faster





We can’t leverage cloud because of **processes and policies**

We can’t change processes and policies because of our **org chart**

DevOps is a reorg.

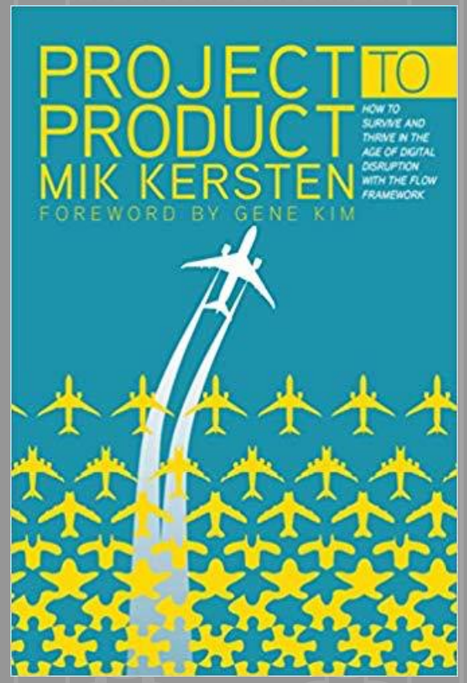
(for most enterprises)

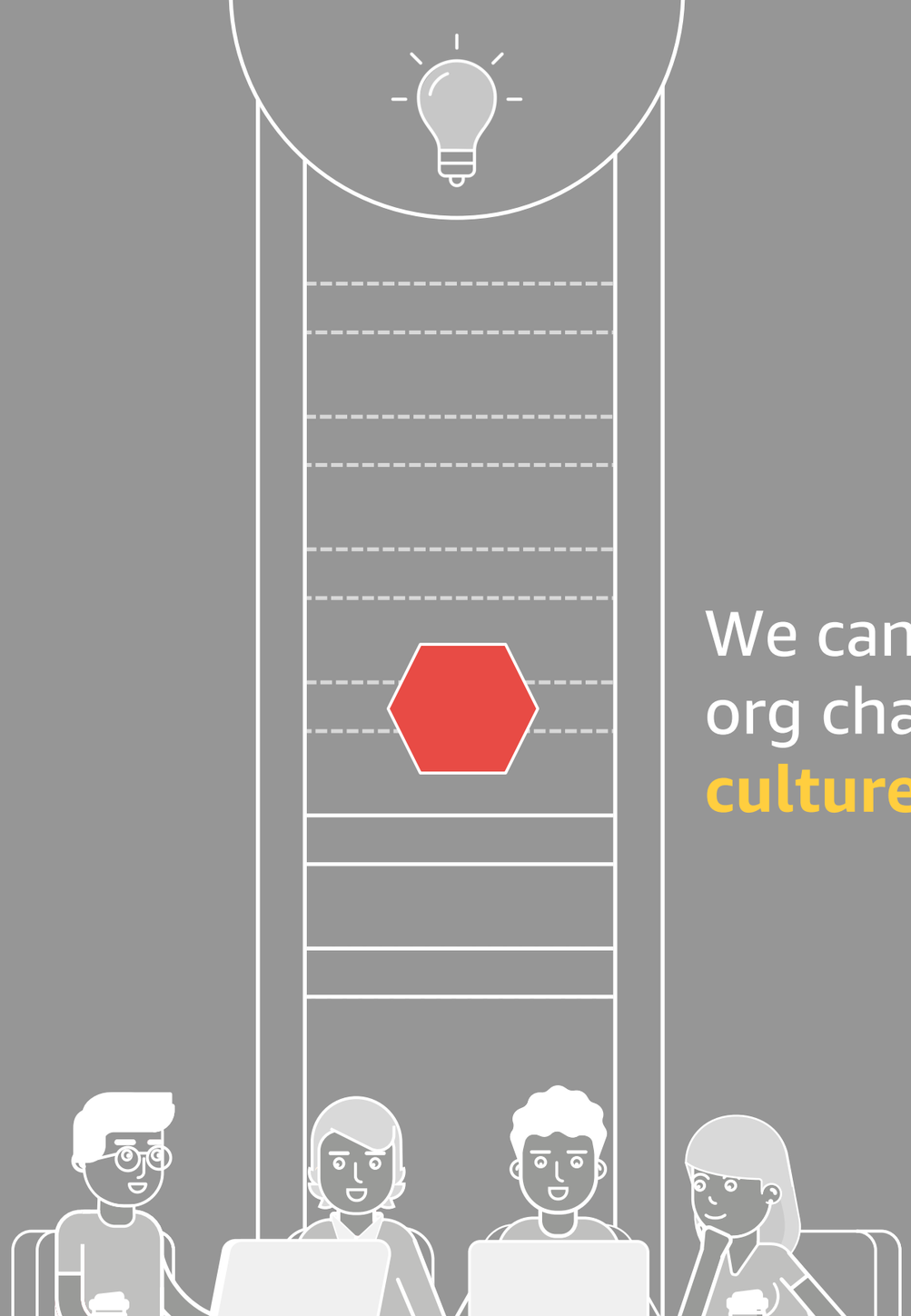
We can’t change processes and policies because of our **org chart**

Change from project to product

(huge reduction in project management staff)

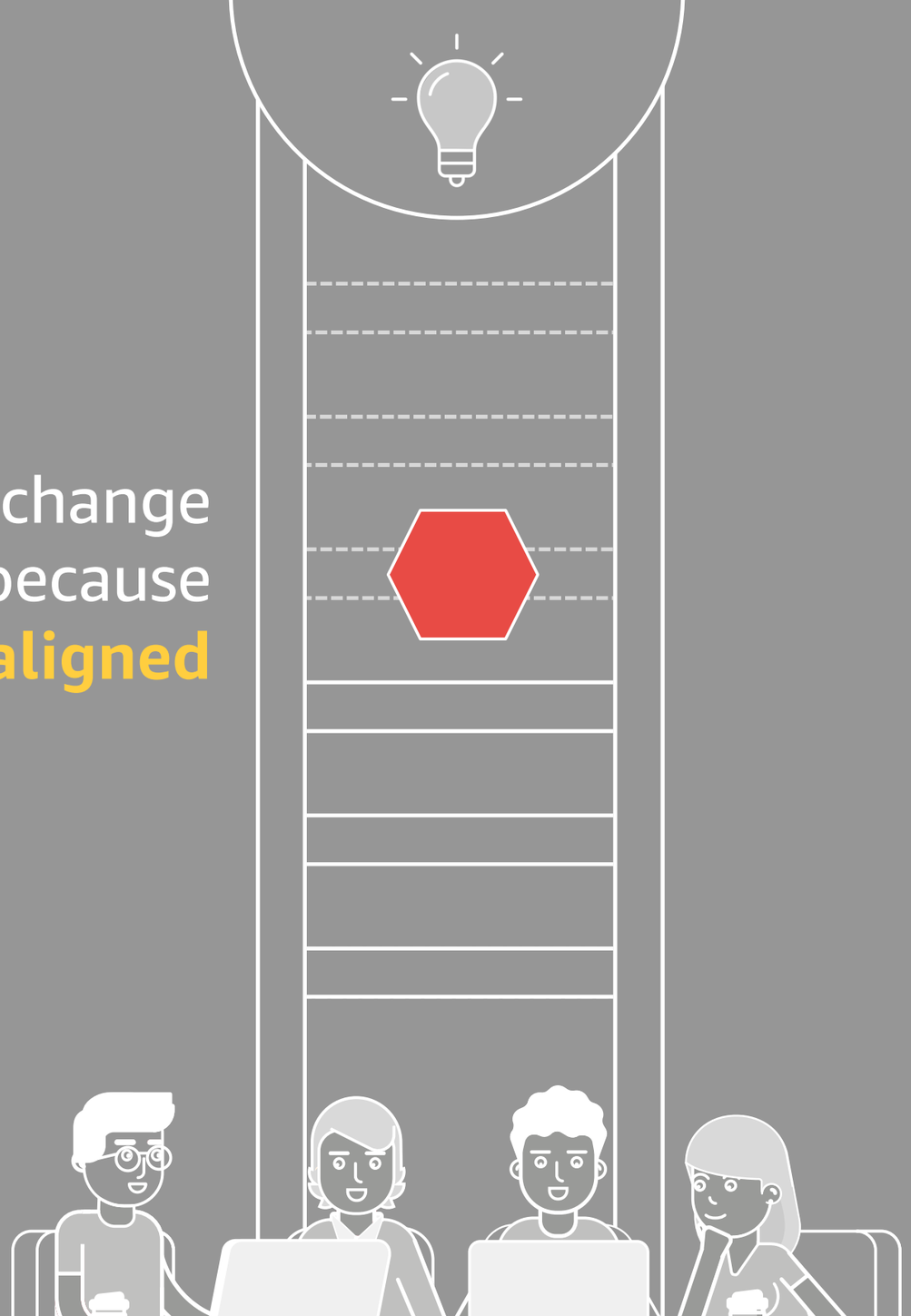
We can't change processes and policies because of our **org chart**





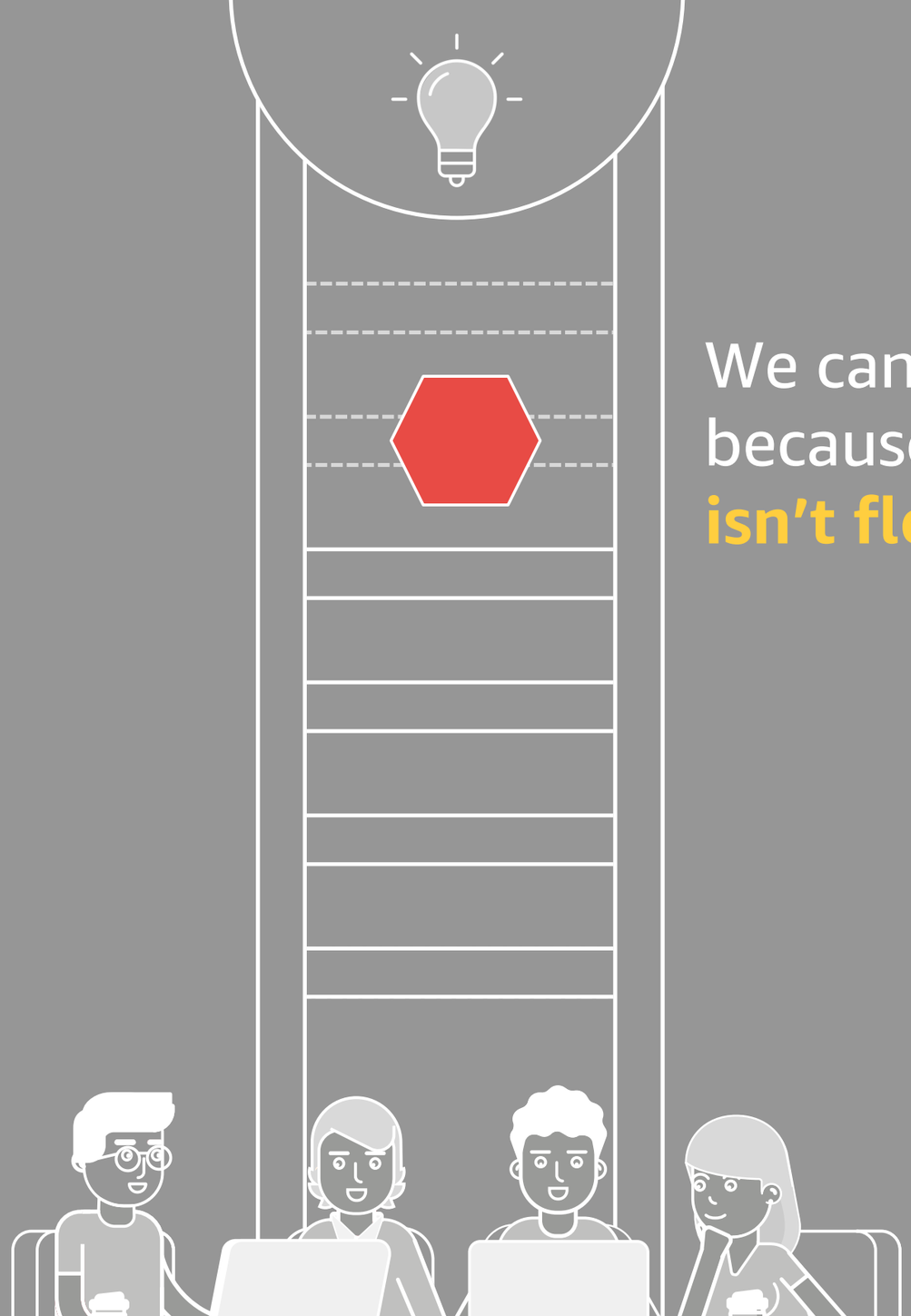
We can’t change our org chart because of our **culture or lack of culture**

We can’t change
culture because
incentives aren’t aligned



We can’t change culture because incentives aren’t aligned

You get the culture you pay for



We can’t change incentives because **compensation isn’t flexible enough**

Board won’t change compensation policy because common **“best practices” are seen as low risk**

Successful board-level patterns

Focus on fewer, higher paid staff and more ownership

Manage capex/opex expectations

Invest in differentiators

Get out of the way of innovation

Compensation policy

Executive succession

Oversight of finance

Oversight of risk

Oversight of strategy

Include technical competence as a must-have

Smaller projects with incremental returns

Time to value

Board-level concerns

"Best practices" that minimize short-term risk get in the way of successful strategic innovation

Pathway for innovation

Speed → Scale → Strategic

Time to value

Distributed optimized capacity

Critical workloads data center replacement

You don’t add innovation to an organization

You get out of its way!

What is the fundamental metric for **innovation?**

Time to value

Do some work

How long?

Value to a customer

Time to value

Do some work

Months?

Value to a customer

Time to value

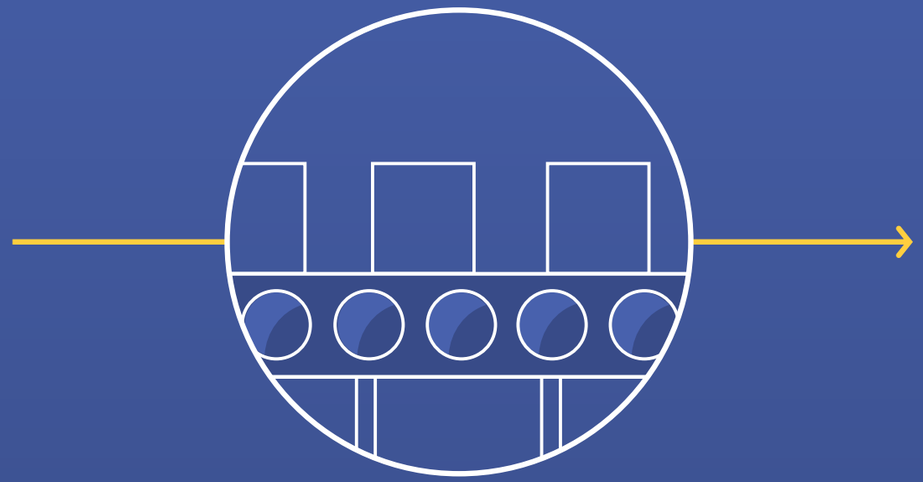
Do some work

Days?

Value to a customer

Time to value

Do some work



Minutes?

Value to a customer

There is no economy of scale in software

Smaller changes are better

Lots of small changes

Automated continuous-delivery pipeline

Tagging feature flags, A/B tests

Rapid cheap builds

Lots of small changes

Hours to Seconds

Move from **Java monolith** to **Go microservices**

Slow build

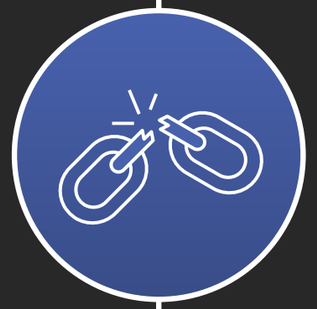
Big build

Fast build

Small build

Change one small thing at a time

Easier to



tell if it breaks

Easier to

roll back to previous version

Easier to

measure time to value

Decouple

New code from **new feature**

Incrementally change system with many small safe updates

Turn on features for testing and when it works—for everyone

Less risk

Faster problem detection

Faster repair

Less work in progress

Less time merging changes

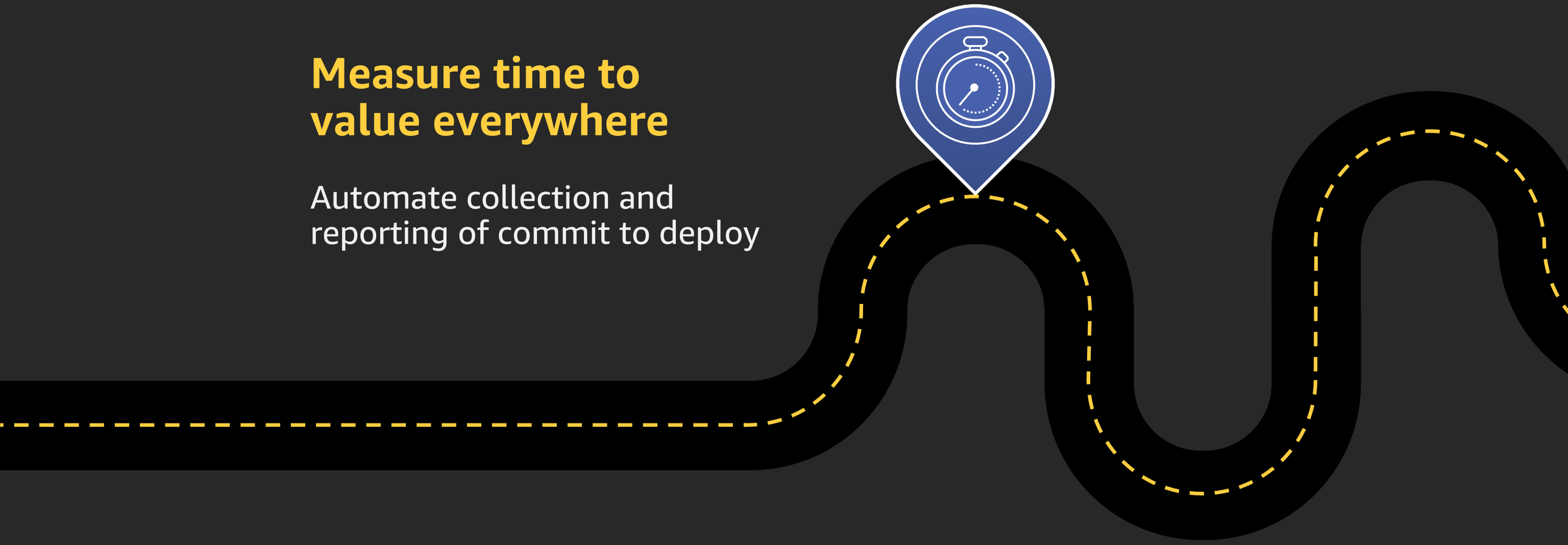
Happier developers

Faster flow

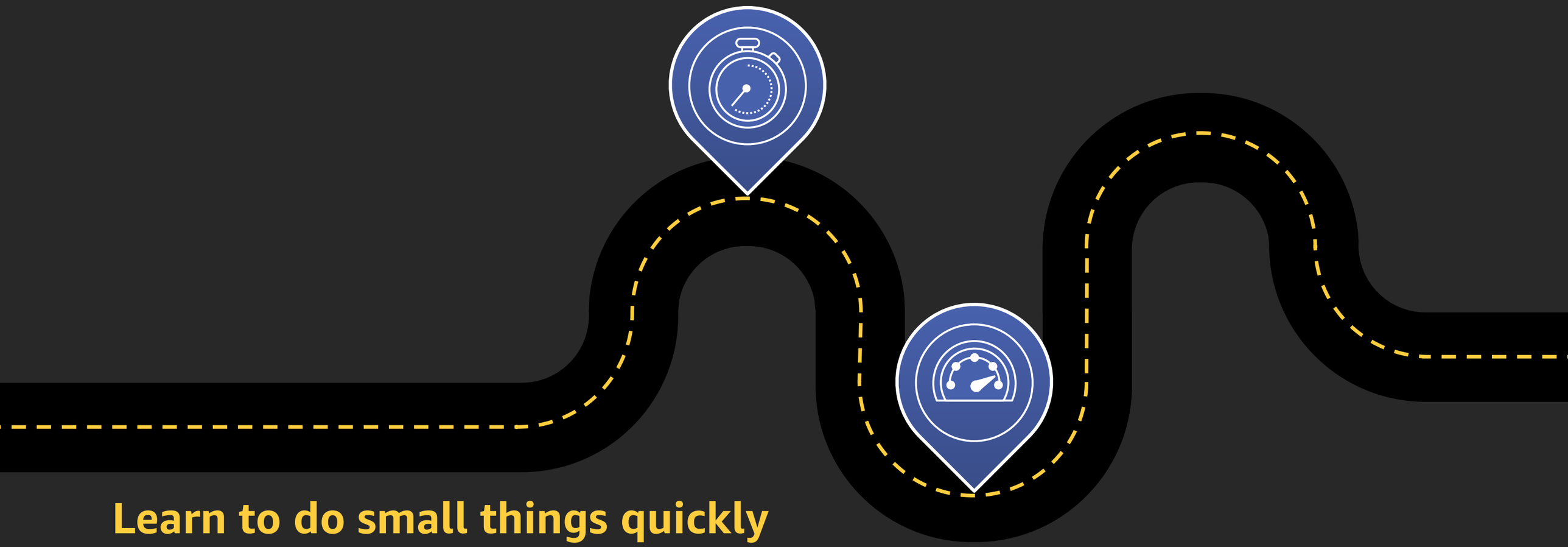
How do we get there?

Measure time to value everywhere

Automate collection and reporting of commit to deploy



How do we get there?

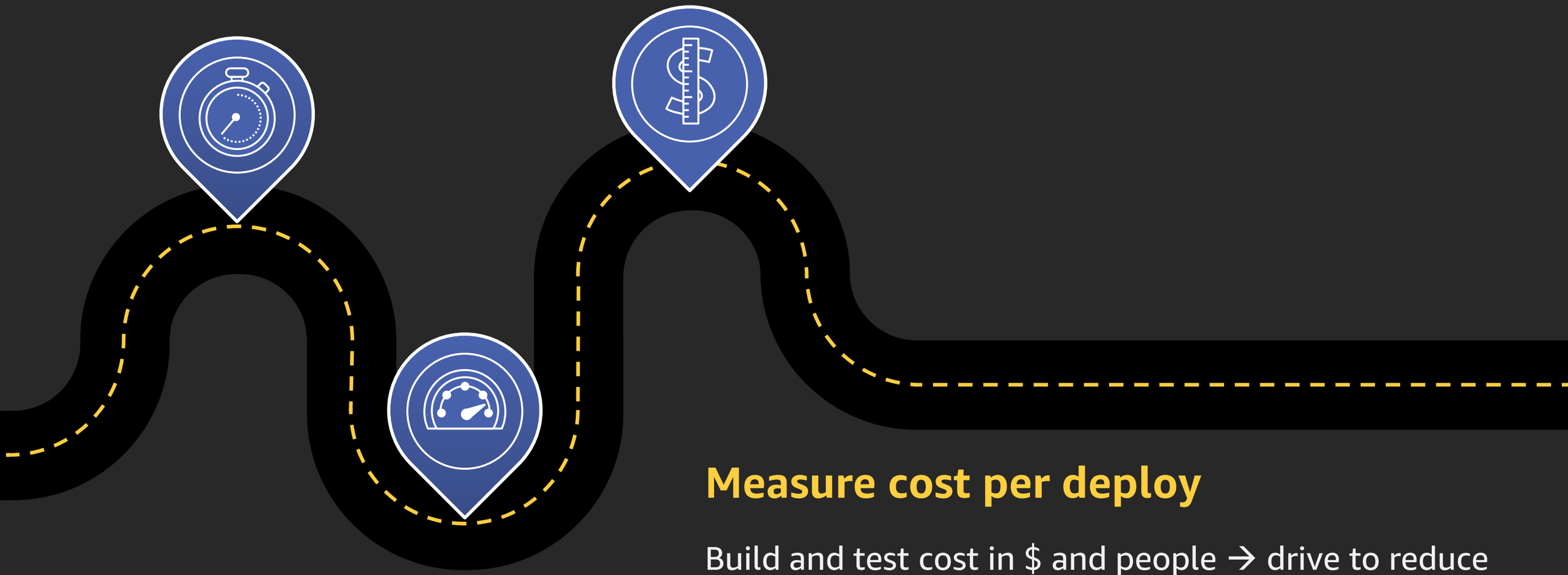


Learn to do small things quickly

Don’t get bogged down speeding up everything.

Create a fast path for simple and safe changes.

How do we get there?

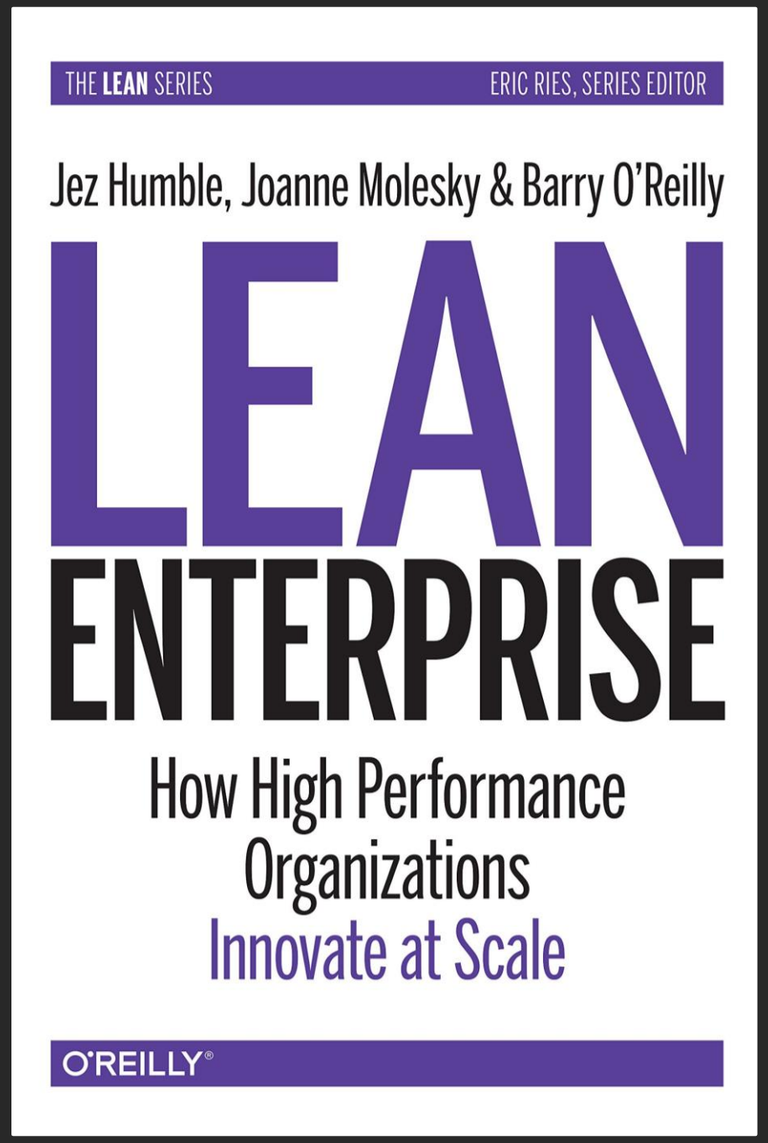


Measure cost per deploy

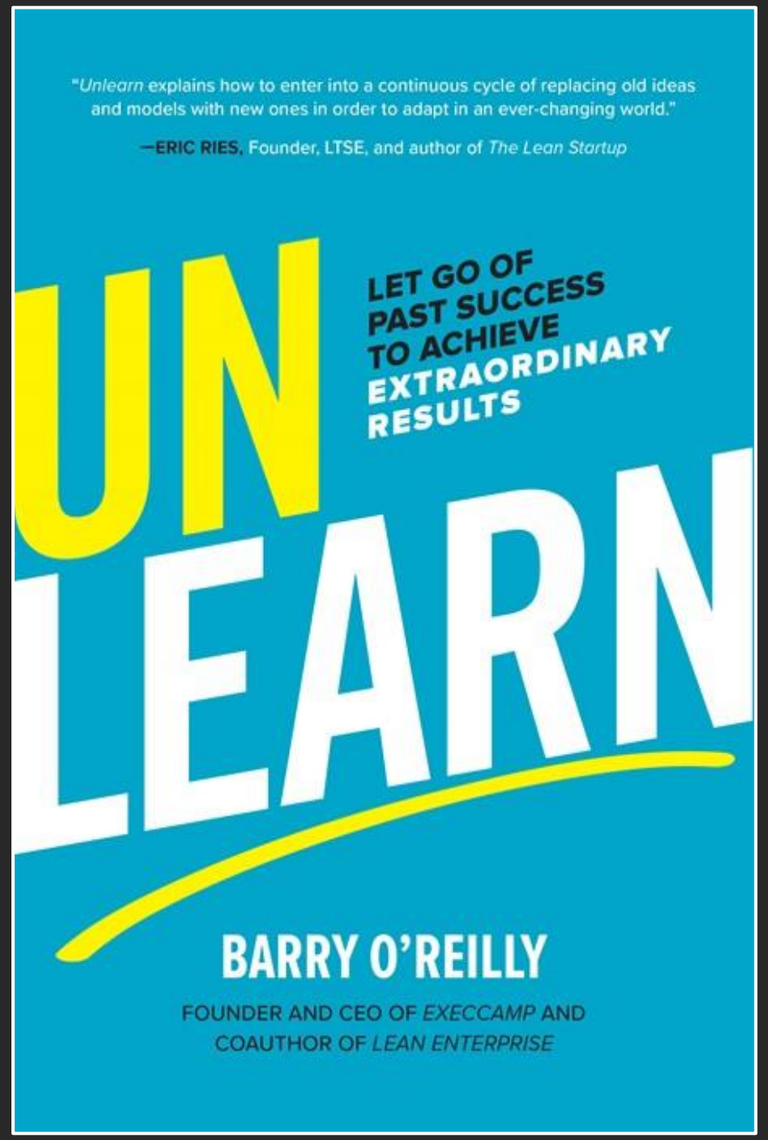
Build and test cost in $ and people → drive to reduce

Number of tickets filed per deploy → drive to one

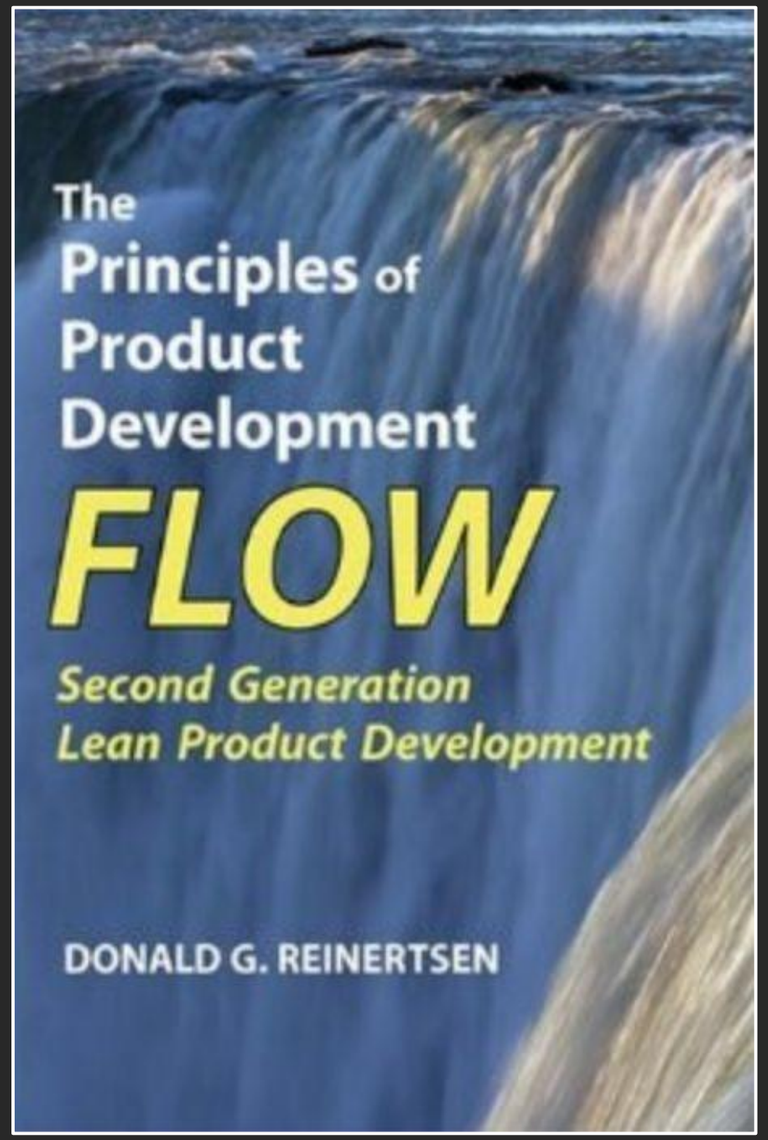
Number of meetings per deploy → drive to zero



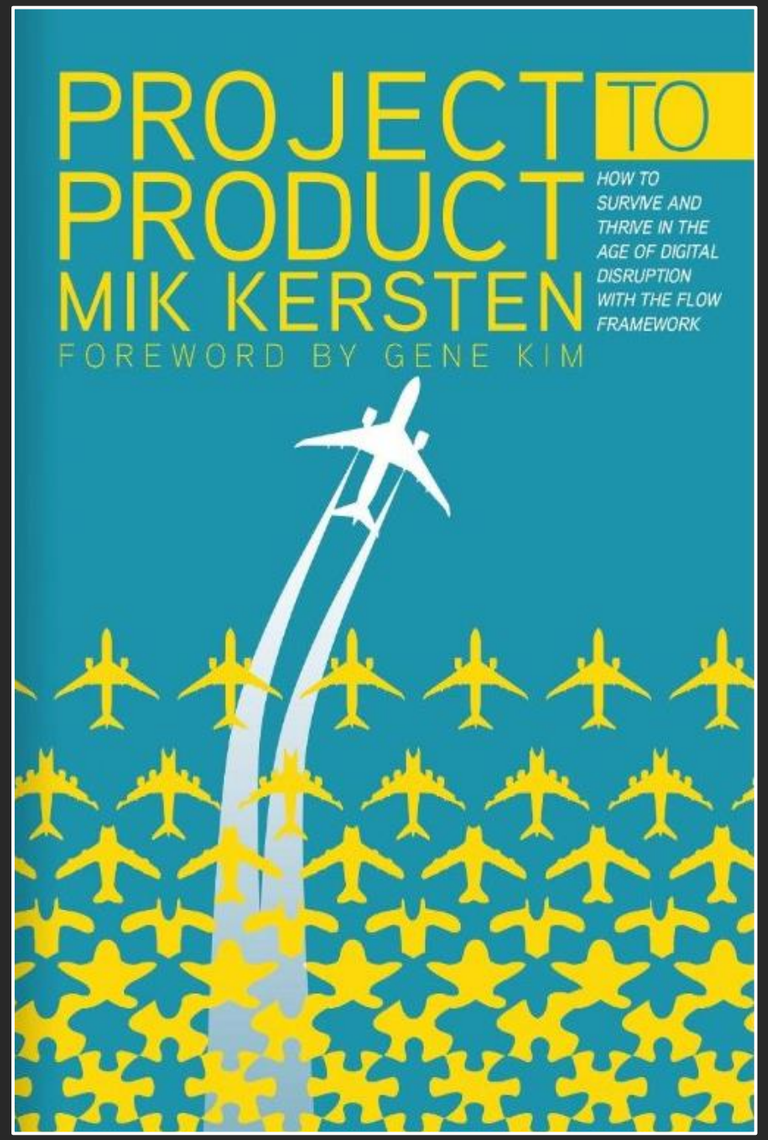
Hypothesis-
driven
development



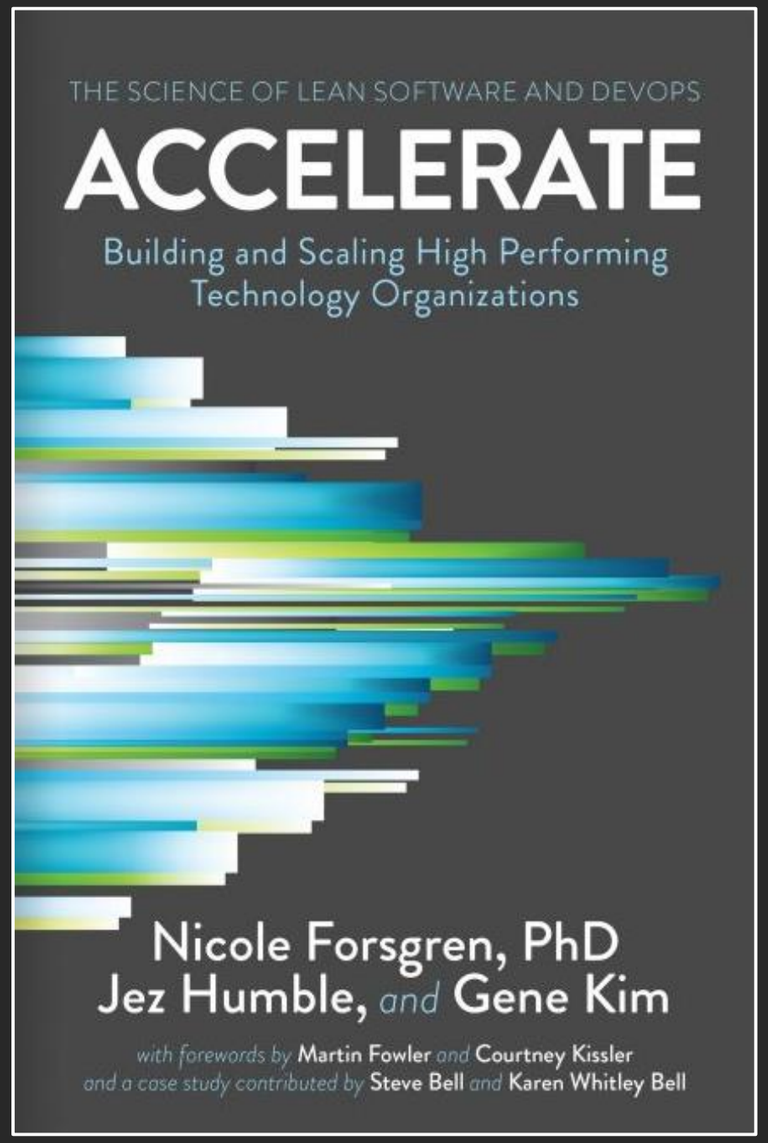
Break away
from your old
ways of working



Theoretical basis for using consistently **small changes**

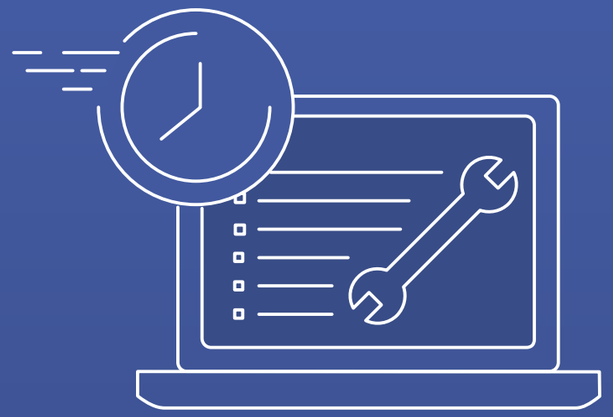


Get rid of 90% of your project managers as you move to **continuous improvement**



Survey data showing that **low latency** time-to-value works

Time to value



Learn to do simple things quickly to unblock innovation

Avoid complex one-size-fits-all processes

Time to value

The best IT architecture today:

Is minimalist, messy, and inconsistent

Provides guardrails for security, scalability, and availability

Is **designed to evolve rapidly** and explore new technologies

Supports low-latency continuous delivery

Pathway for innovation

Speed → Scale → Strategic

Time to value

Distributed optimized capacity

Critical workloads data center replacement

Distributed optimized capacity

Highly scaled

Distributed for availability

Cost-optimized high utilization

Cloud-native architecture

Cloud-native principles

Pay as you go, afterward

Self-service—no waiting

Globally distributed by default

Cross-zone/-region availability models

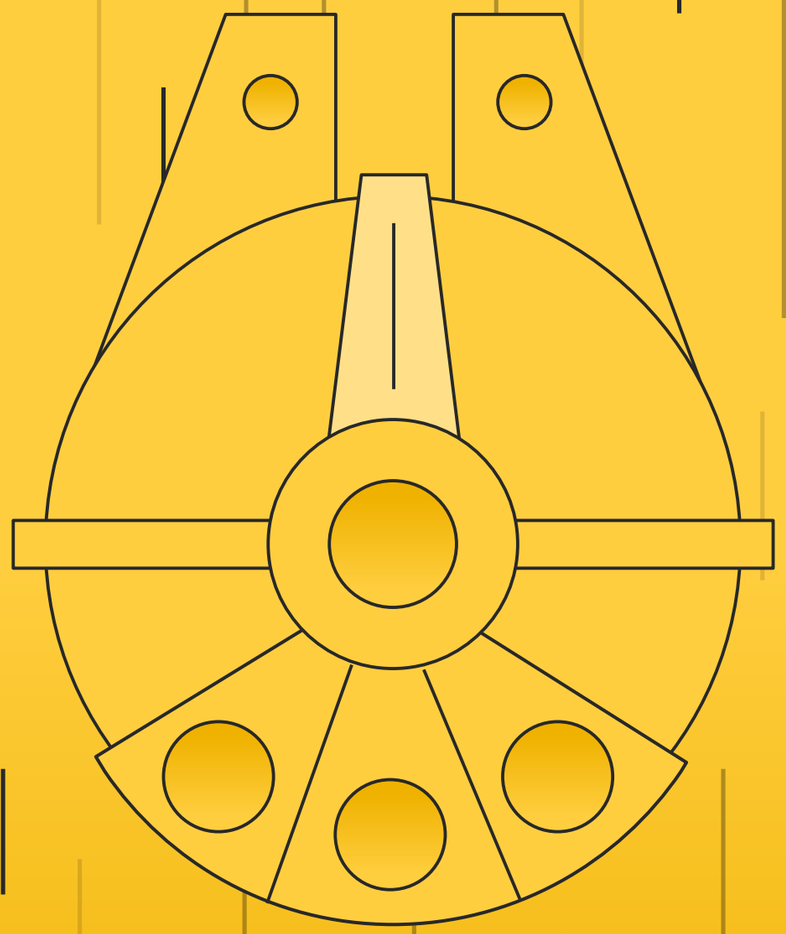
High utilization—turn idle resources off

Immutable code deployments

Containers or serverless?

Or both?

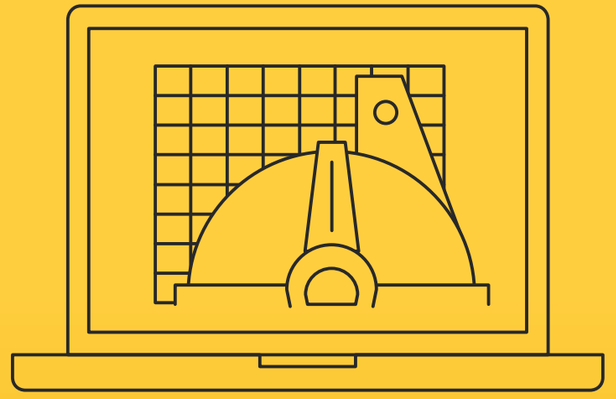
What is the user **need?**



What is the **problem** you are trying to solve?

Make a model spaceship **quickly** and **cheaply**

Traditional development

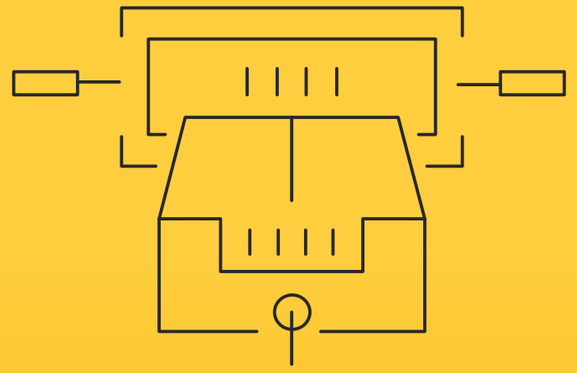


Design a prototype

Traditional development

Carve from
modeling clay

Traditional development



Make molds

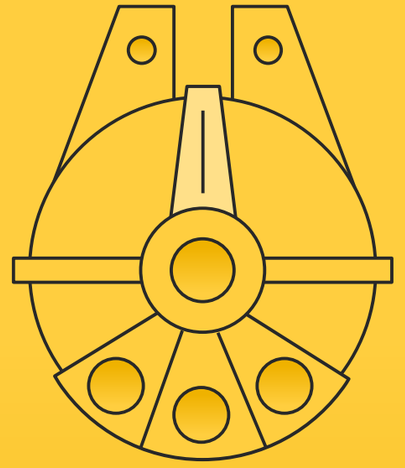
Traditional development

Produce
injection-molded parts

Traditional development

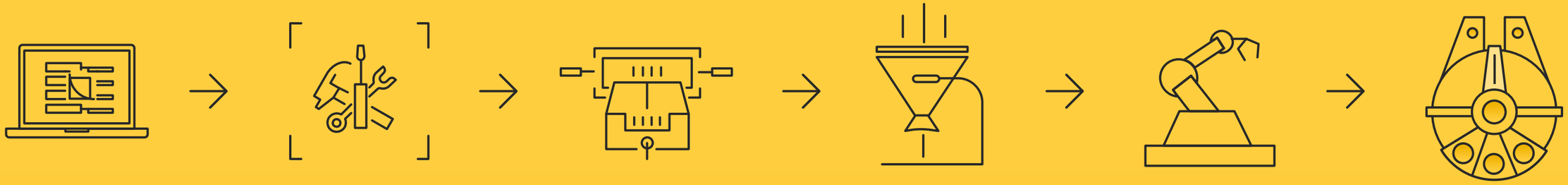
Assemble parts

Traditional development



Sell finished toy

Traditional development



Design a prototype → Carve from modeling clay → Make molds → Produce injection-molded parts → Assemble parts → Sell finished toy

Rapid development

Big bag of blocks

+

Instructions

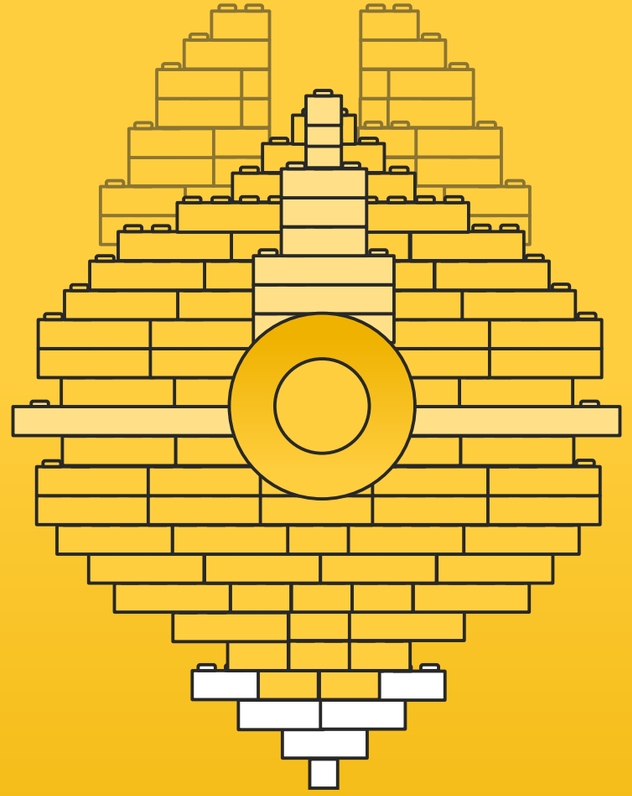
+

A few hours

Rapid development

A finished toy

Rapid development

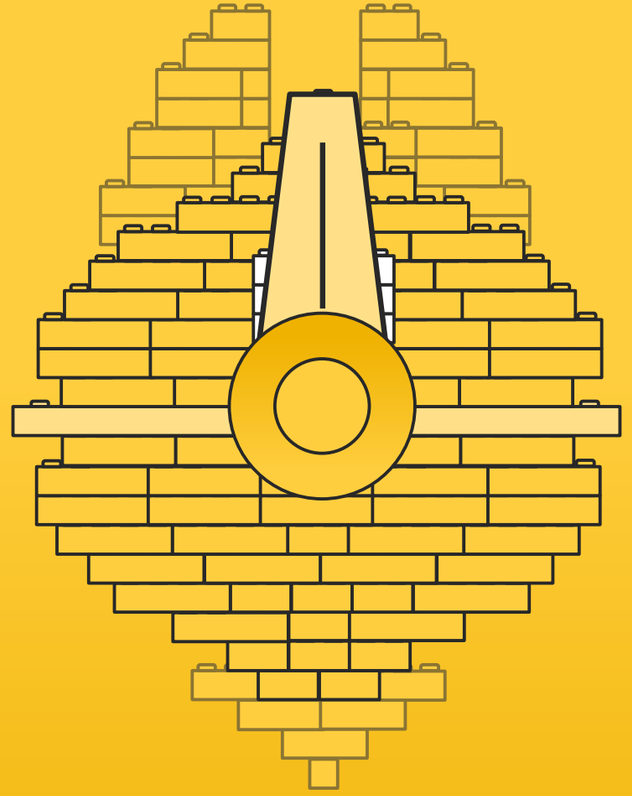


Lacks fine detail

Recognizable, but not exactly what was asked for

Easy to modify and extend

Optimization



Take a group of Lego bricks …
and form a new custom brick

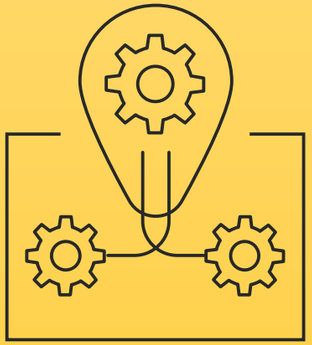
A more specialized
common component

Traditional	Rapid development
Full custom design	Building-bricks assembly
Months of work	Hours of work
Custom components may be fragile and need to be debugged and integrated	Standard reliable components scale and are well understood and interoperable
Too many detailed choices	Need to adjust requirements to fit the patterns available
Long decision cycles	Constraints tend to reduce debate and speed up decisions

Containers	Serverless
Custom code and services	Serverless events and functions
Lots of choices of frameworks and API mechanisms	Standardized choices
Where needed, optimize serverless applications by also building services using containers to solve for • Lower startup latency • Long-running compute jobs • Predictable high traffic	Combine these building blocks λ AWS Lambda Amazon API Gateway Amazon SNS, Amazon SQS Amazon DynamoDB AWS Step Functions

Observability of systems

Epidemic failure modes



Automation and continuous chaos

Observability

Kalman, 1961 paper

On the general theory of control systems

A system is observable if the behavior of the entire system can be determined by only looking at its inputs and outputs

Physical and software control systems are based on models, remember all models are wrong, but some models are useful …

Observability

High

Function with no side effects

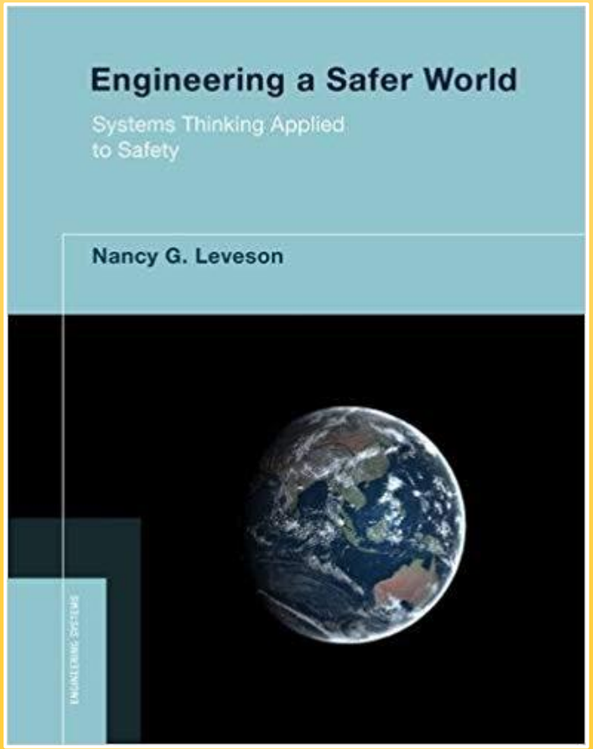
Microservice that does one thing

Medium

Monolith with tracing and logging

Low

Monolith with logging



Engineering a Safer World

Systems Thinking Applied to Safety

Nancy G. Leveson

STPA – Systems theoretic process analysis

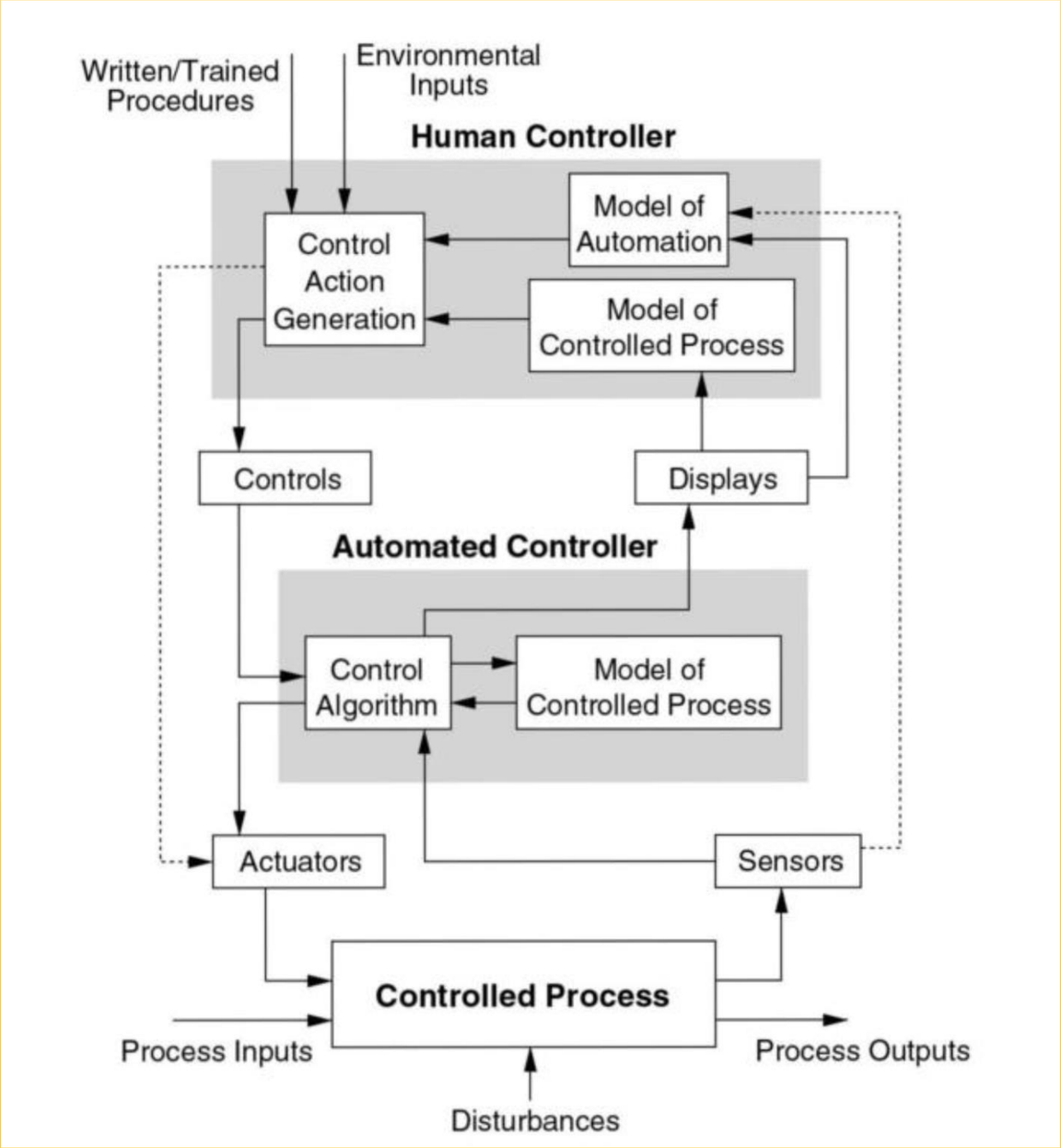
STAMP – Systems theoretic accident model and processes

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

Observability

STPA model

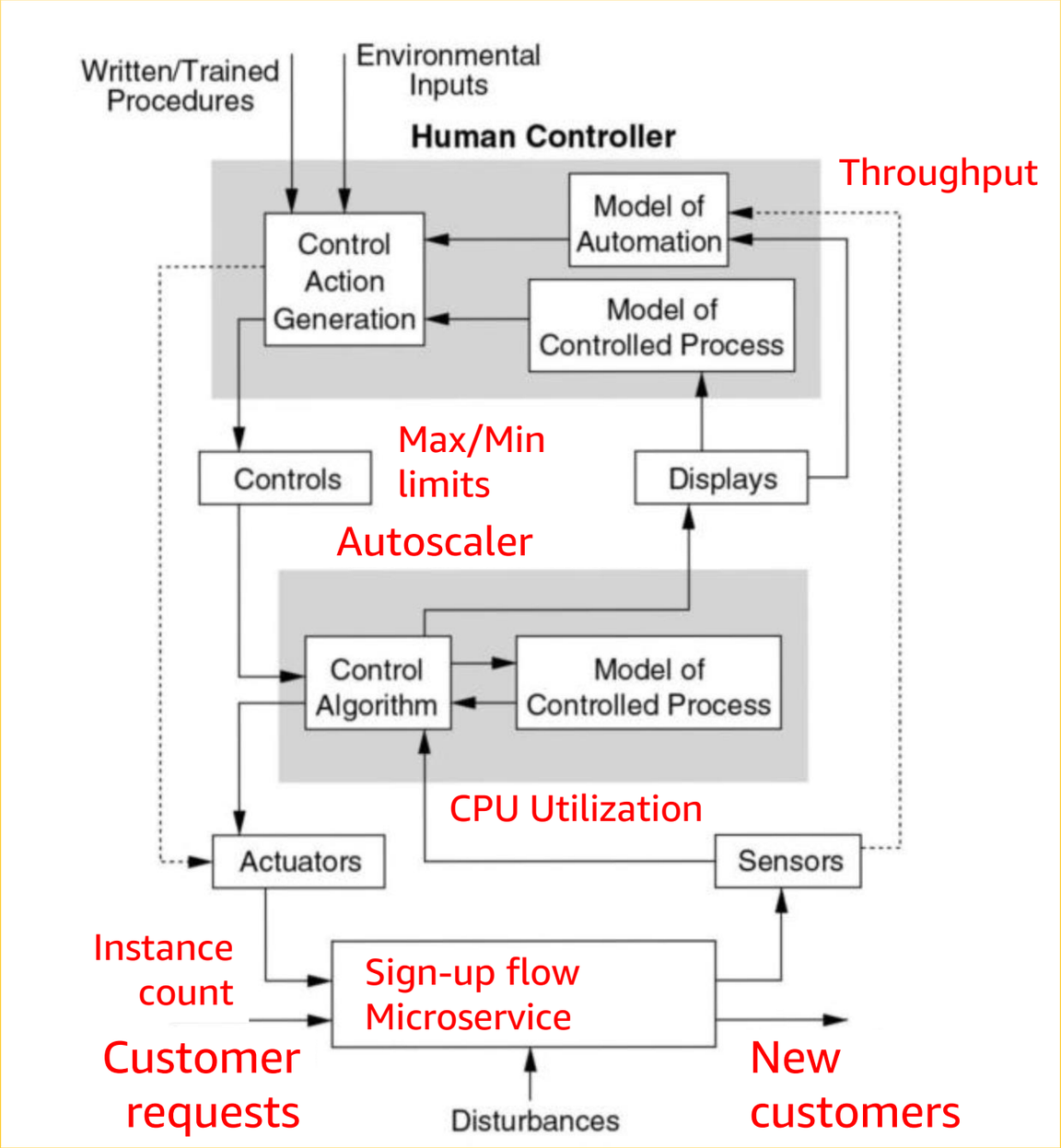
Focus on interfaces' wires, not boxes



Observability

STPA model

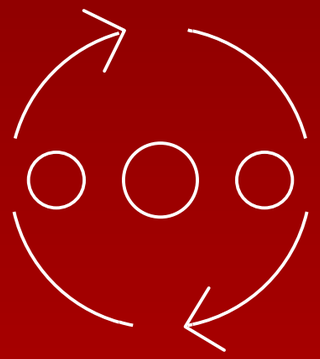
Understand hazards that could disrupt control of the process



Failures can be

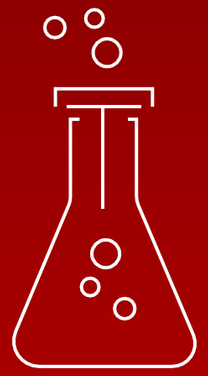
Independent

Common assumption



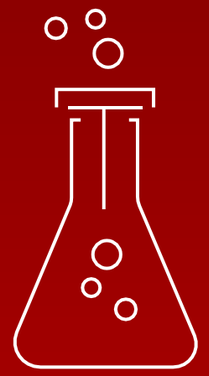
Correlated

Harder to model and mitigate knock-on effects



Epidemic

Everything breaks at once!



Epidemic examples

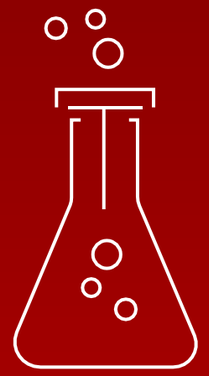
Linux leap – Second bug

Memory leak in agent

Cloud Zone or Region failure

DNS failure

Security configuration syntax error



Epidemic examples

Linux leap – Second bug

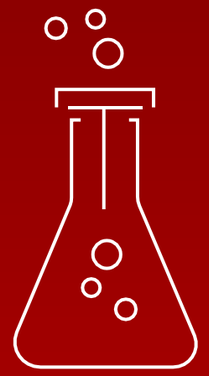
Memory leak in agent

Cloud Zone or Region failure

DNS failure

Security configuration syntax error

Quarantine needed



Epidemic examples

Linux leap – Second bug
Maintain ability to deploy on Windows

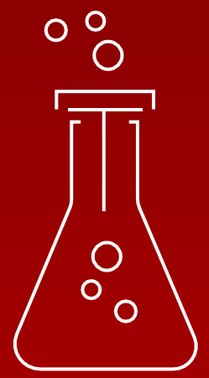
Memory leak in agent
Use multiple monitoring tools

Cloud Zone or Region failure
Cross-Zone or -Region replication

DNS failure
Multiple domains and providers

Security configuration syntax error
Limit the scope of deployments

Quarantine



Epidemic examples

Linux leap – Second bug
Maintain ability to deploy on Windows

Memory leak in agent
Use multiple monitoring tools

Cloud Zone or Region failure
Cross-Zone or -Region replication

DNS failure
Multiple domains and providers

Security configuration syntax error
Limit the scope of deployments

Quarantine

Diversity needs to be managed to contain an epidemic

Pathway for innovation

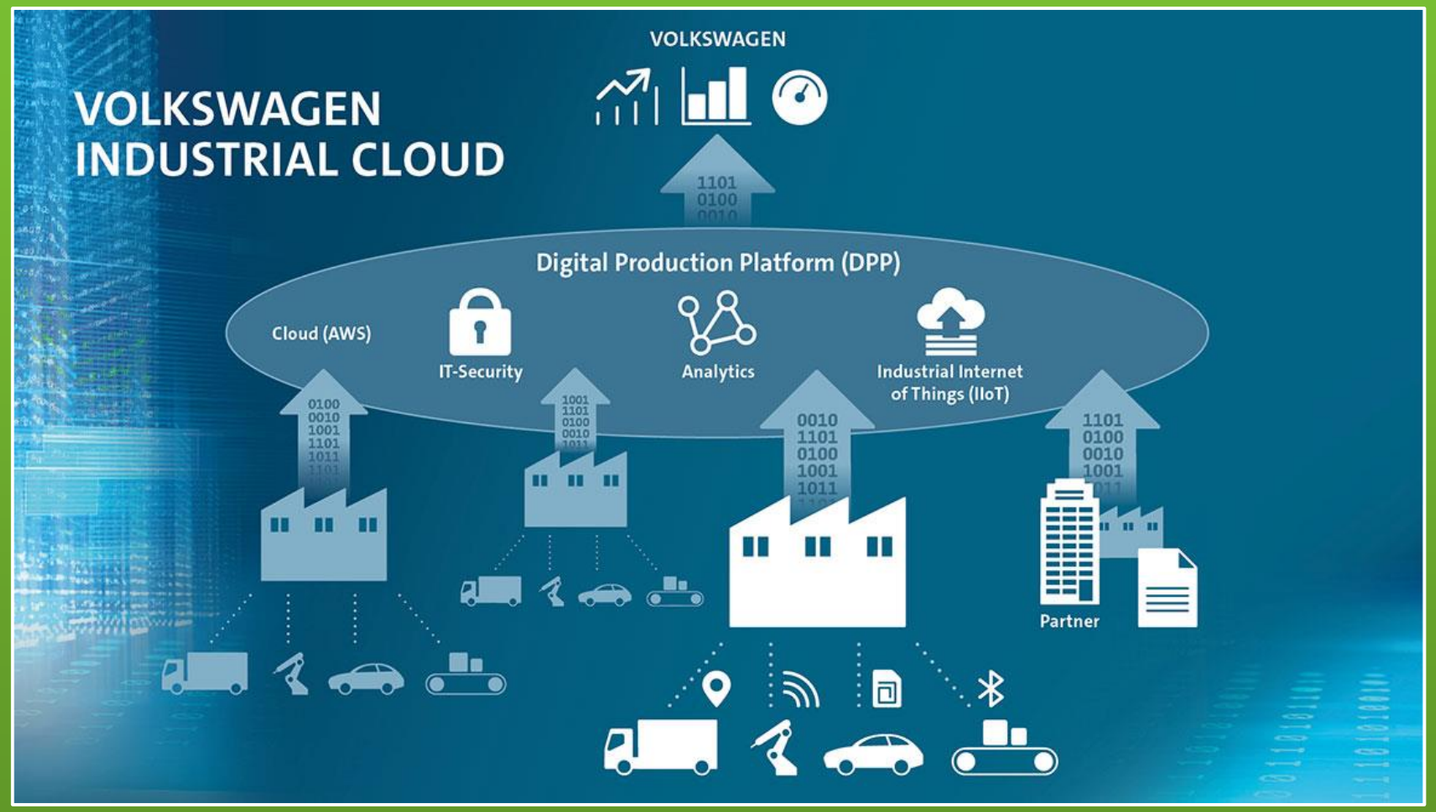
Speed → Scale → Strategic

Time to value

Distributed optimized capacity

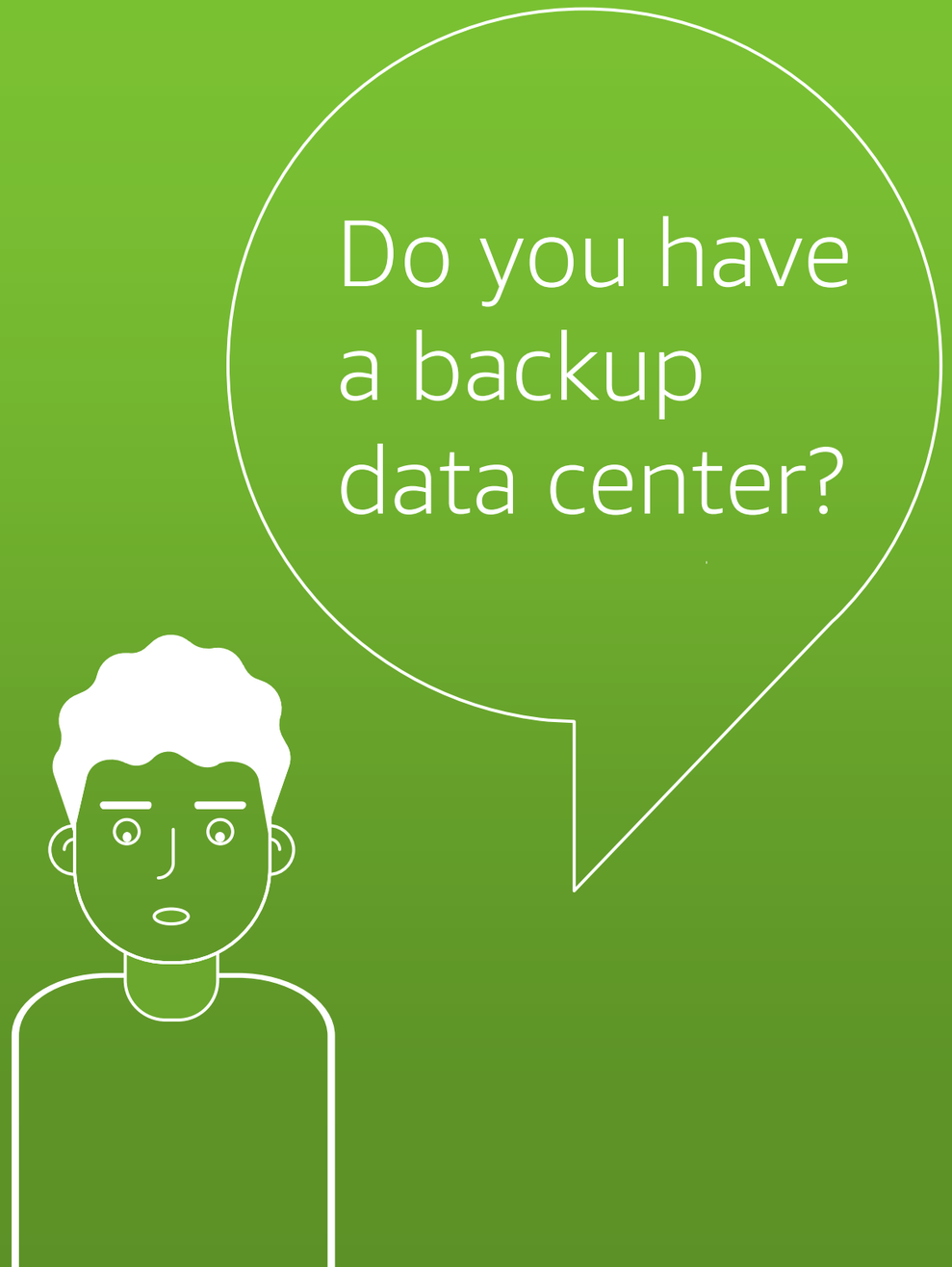
Critical workloads data center replacement

AWS and Volkswagen Industrial Cloud



Data center–to–cloud migrations are underway for the most business- and safety-critical workloads

AWS and our partners are developing patterns, solutions, and services for customers in all industries, including Travel, Finance, Healthcare, Manufacturing …



How often do you failover apps to it?

How often do you failover the **whole data center** at once?

“Availability theater”

A fairy tale …

Once upon a time, in theory, if everything works perfectly, we have a plan to survive the disasters we thought of in advance

How did that work out?

Forgot to renew domain name …	**SaaS vendor**
Didn’t update security certificate and it expired …	**Entertainment site**
Data center flooded in Hurricane Sandy …	**Finance company, Jersey City**
Whoops!	**You, tomorrow**

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

—Chris Pinkham

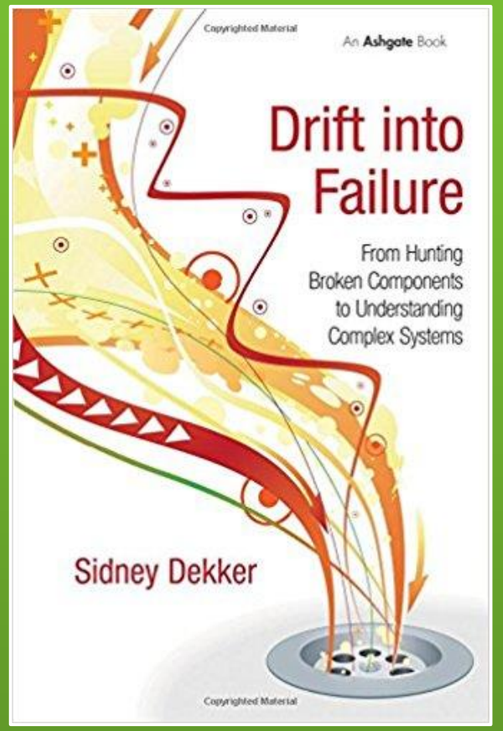
“The Network Is Reliable”

ACM Queue, 2014

Bailis & Kingsbury

@pbailis @aphyr

(Spoiler—it isn’t …)



Drift into Failure

Sydney Dekker

Everyone can do everything right at every step, and you may still get a catastrophic failure as a result …

Release It!

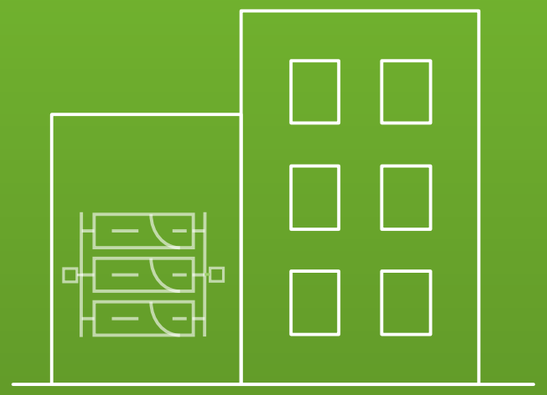
Second Edition, 2017

Michael T. Nygard

Bulkheads, circuit breakers, and some new ideas …

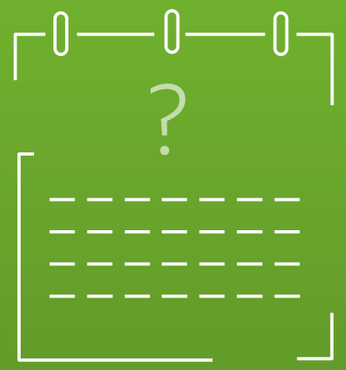
Resilience

Past — **Present** — **Future**

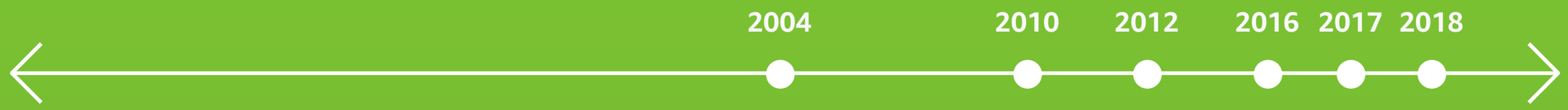


Disaster recovery

Chaos engineering



Resilient critical systems



Chaos engineering

2004 Amazon – Jesse Robbins. Master of disaster.

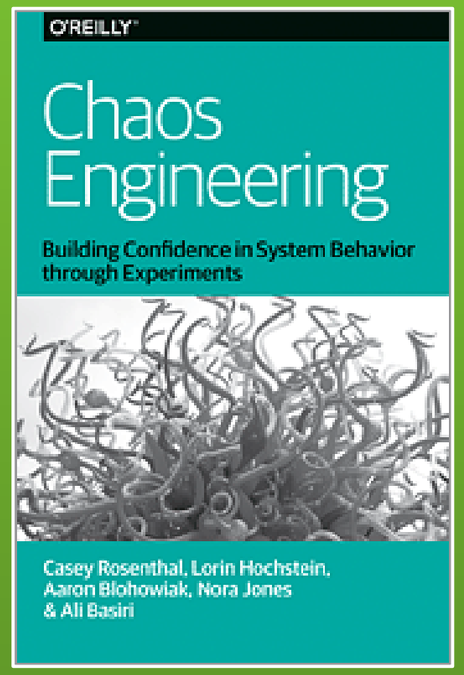
2010 Netflix – Greg Orzell @chaosimia. First implementation of Chaos Monkey to enforce use of auto-scaled stateless services.

2012 NetflixOSS open sources Simian Army.

2016 Gremlin Inc. founded.

2017 Netflix chaos engineering book. Chaos toolkit open-source project.

2018 Chaos concepts getting adoption, more startups.



Chaos engineering team

People

Application

Switching

Infrastructure

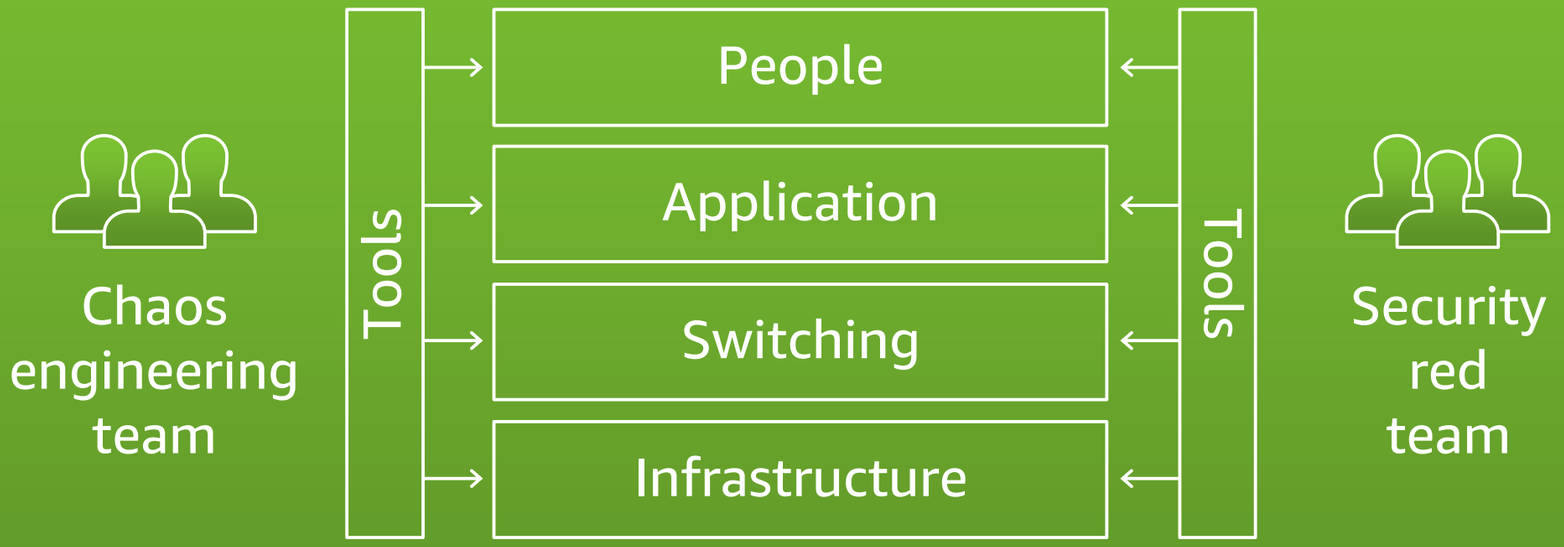
Chaos architecture

Four layers

Two teams

An attitude—

Find the weakest link



You can only be as strong as your

weakest link

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

Failures are a system problem—lack of safety margin

Not something with a root cause of component or human error

Experienced staff

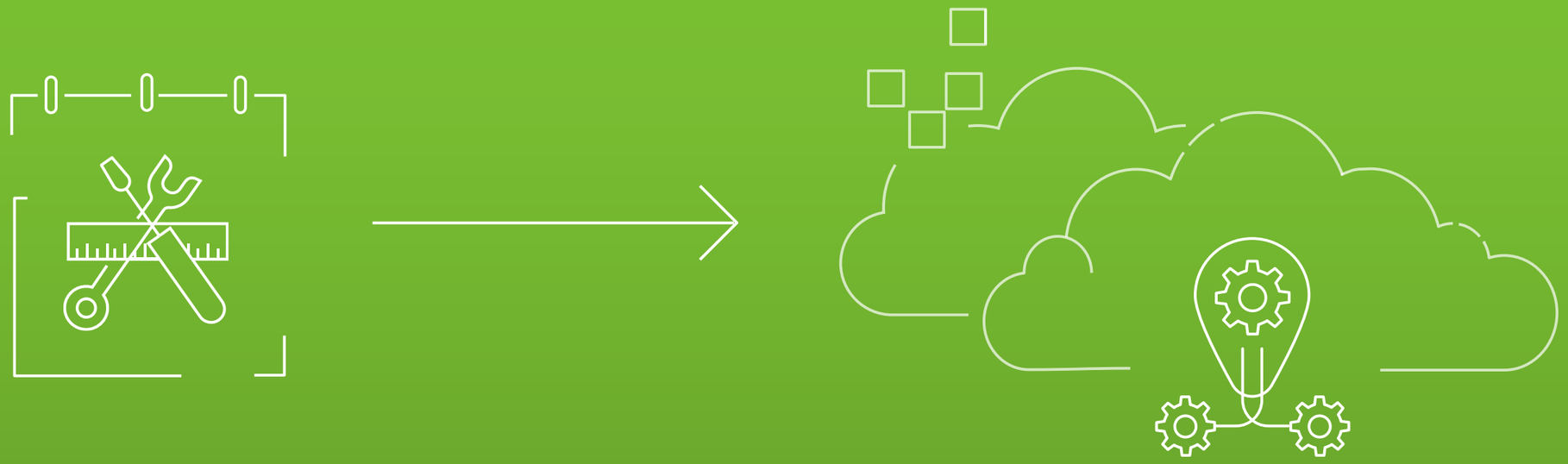
Robust applications

Dependable switching fabric

Redundant service foundation

Cloud provides the automation that leads to chaos engineering

As data centers migrate to cloud, fragile and manual disaster recovery processes can be standardized and automated



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

“Cloud for CEOs: Measure innovation with one metric”

Adrian Cockcroft

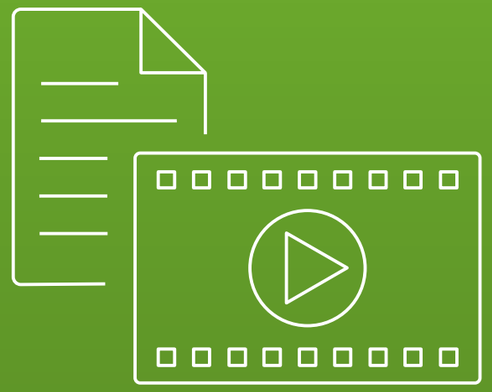
aws.amazon.com/enterprise/executive-insights/content/cloud-for-ceos

Adrian Cockcroft, VP Cloud Architecture Strategy, AWS

Digitally transforming an enterprise requires connecting with customers, understanding their needs, and responding faster than ever. The most successful organizations of today are disrupting their competitors and entering new markets by innovating more quickly and efficiently. Being an innovator requires overcoming challenges in four areas: culture, skills, organization, and risk management. Get these right, and you will be able to leverage technologies such as cloud computing and machine learning to innovate. Adrian Cockcroft, AWS VP of Cloud Architecture Strategy, provides his perspective on unlocking growth and measuring innovation with one metric: time-to-value.

Booklist

https://a.co/79CGMfB



Slides and video links

https://github.com/adrianco/slides

Best wishes for your transformation!

Speed → Scale → Strategic

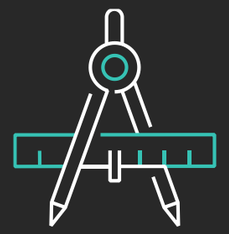
Time to value

Distributed optimized capacity

Critical workloads data center replacement

Learn to architect with AWS Training and Certification

Resources created by the experts at AWS to propel your organization and career forward



Free foundational to advanced digital courses cover AWS services and teach architecting best practices

Classroom offerings, including Architecting on AWS, feature AWS expert instructors and hands-on labs

Validate expertise with the **AWS Certified Solutions Architect - Associate** or **AWS Certification Solutions Architect - Professional** exams

Visit aws.amazon.com/training/path-architecting/

aws training and certification

Thank you!

Adrian Cockcroft

@adrianco

AWS re:Invent

aws

Please complete the session survey in the mobile app.

AWS re:Invent

aws