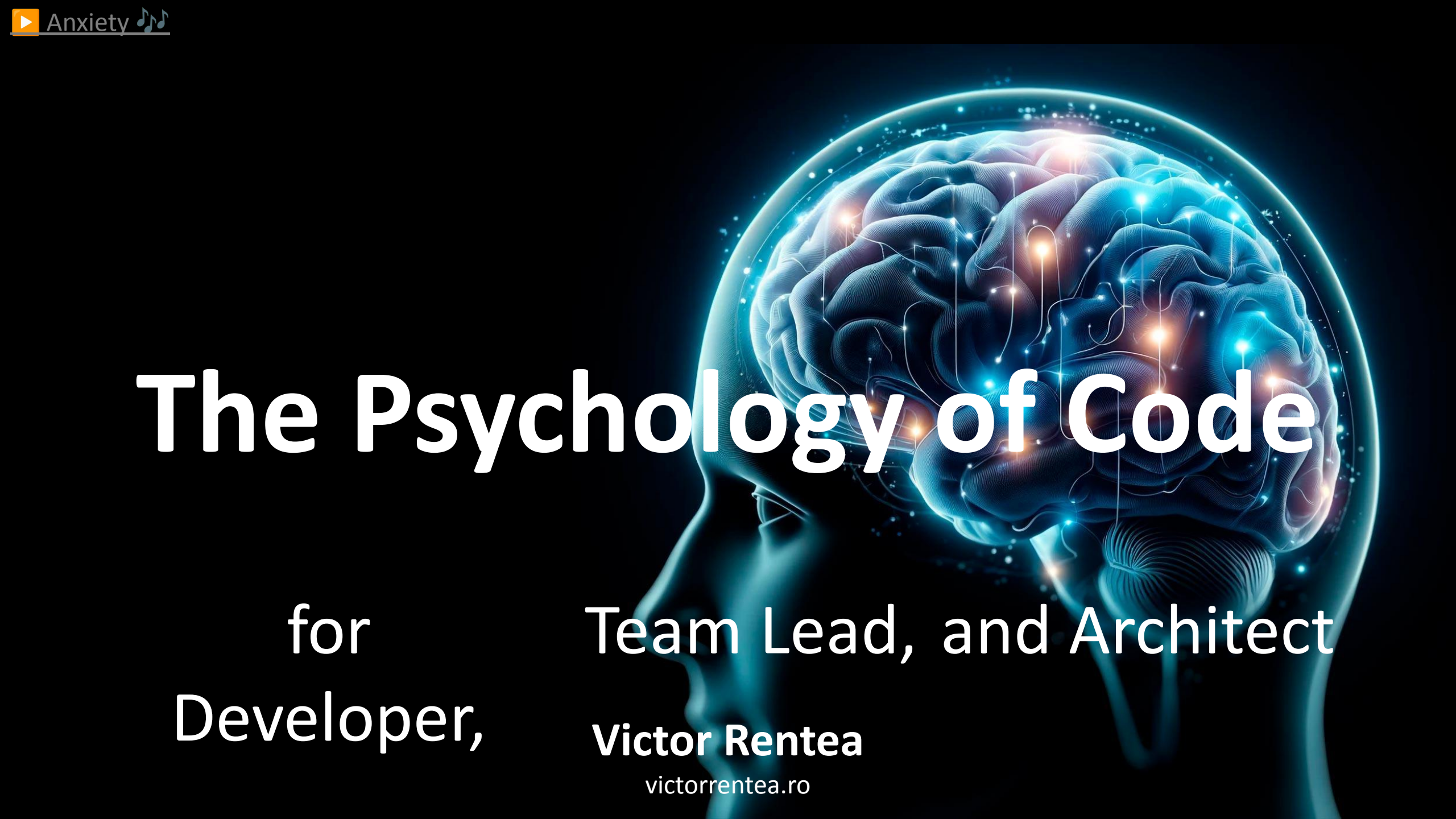
▶️ Anxiety 🎶



The Psychology of Code

for Developer, Team Lead, and Architect

Victor Rentea
victorrentea.ro

Imposter Syndrome

Ever felt **unqualified** for your task?

"I have **no idea** what I'm doing!"

But others think
you're doing a great job.

Imposter Syndrome - Why?

- Fast Changing tools and paradigms
- Job Requiring superhero knowledge & skills
- Toxic Environments promoting elitism
- AI-generated kung-fu code
- Will AI steal my job? No! An AI-skilled dev will. 😈

Victor Rentea

- **18y of Coding**, Java Champion
- **10y of Training**
- At **150 companies** in

for hirE

victorrentea.ro

- **100+ Educational Conference Talks** on YouTube
- **Meet me online: victorrentea.ro/community**

The more you know,

The less you feel you know

unknown

Junior Knowledge

Senior Knowledge

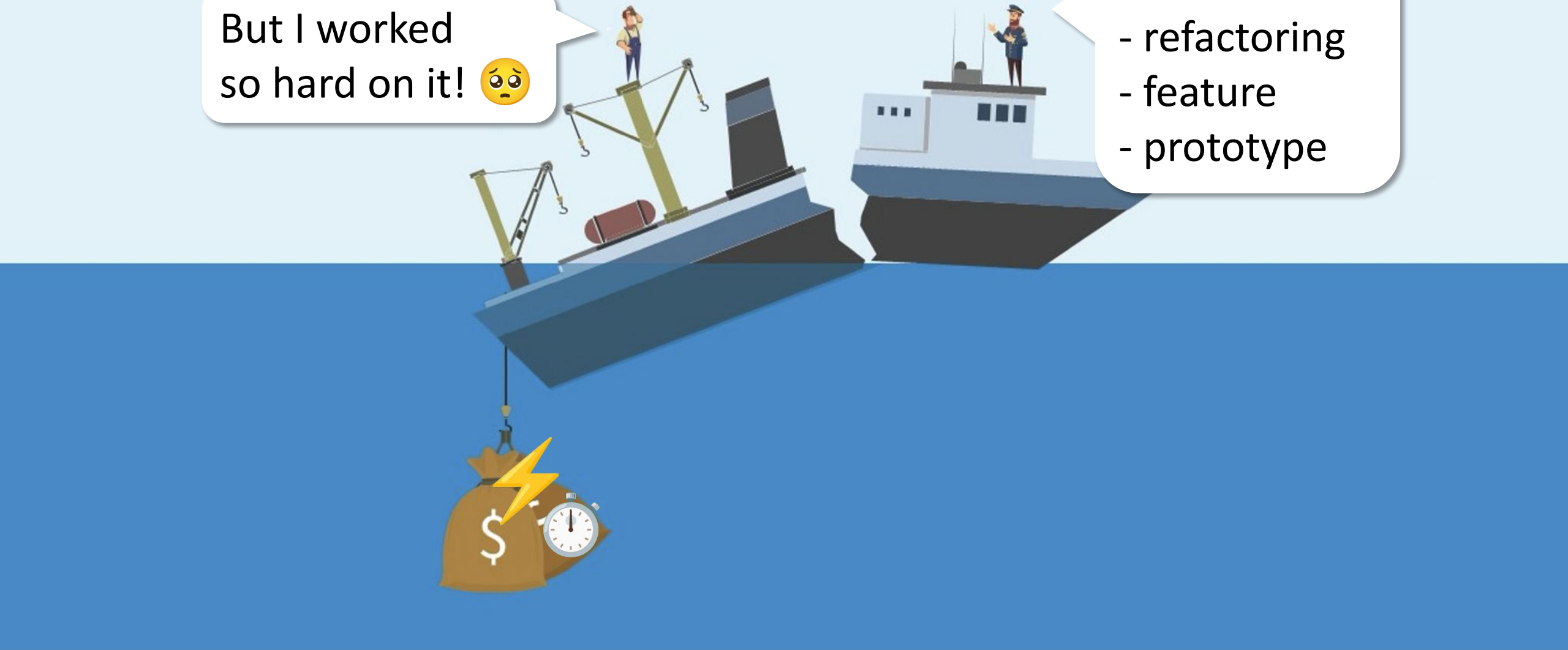
larger contact surface □

Surviving Imposter Syndrome

- ✔ **Accept it** – be **humble** & Ack **your limits**
- ✔ **Celebrate small victories** 🏆
- ✔ **Seek Social Contact** 🍻
- ✔ **Teach Someone**

Irrational Artifact Attachment

"Sunk-cost Fallacy"



Fall in love with Problem Solving, not Typing Code.

Private Code Ownership

Watch out, you're touching Darian's code !!

Collective Code Ownership

Establish Trust by **Pairing**, **Mobbing**

Review This !

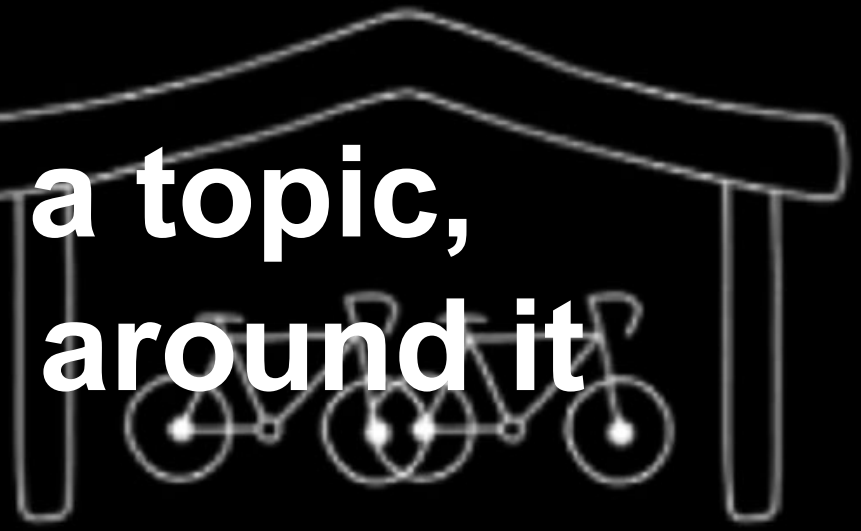
- That's not Java-style {
- Add { after for/if
- Don't import *
- Use boolean primitive
- Use Streams
- Tabs or spaces?
- Use s.isEmpty()

```
import java.util.*;

Boolean hasEmpty(List<String> list)
{
    for (var s : list)
        if ("".equals(s)) return true;
    return false;
}
```

Bike-Shedding

(Parkinson's Law of triviality)



The more basic a topic,
the more debates around it

Nuclear plant
cost: $28,000,000
discussion: 2,5 minutes

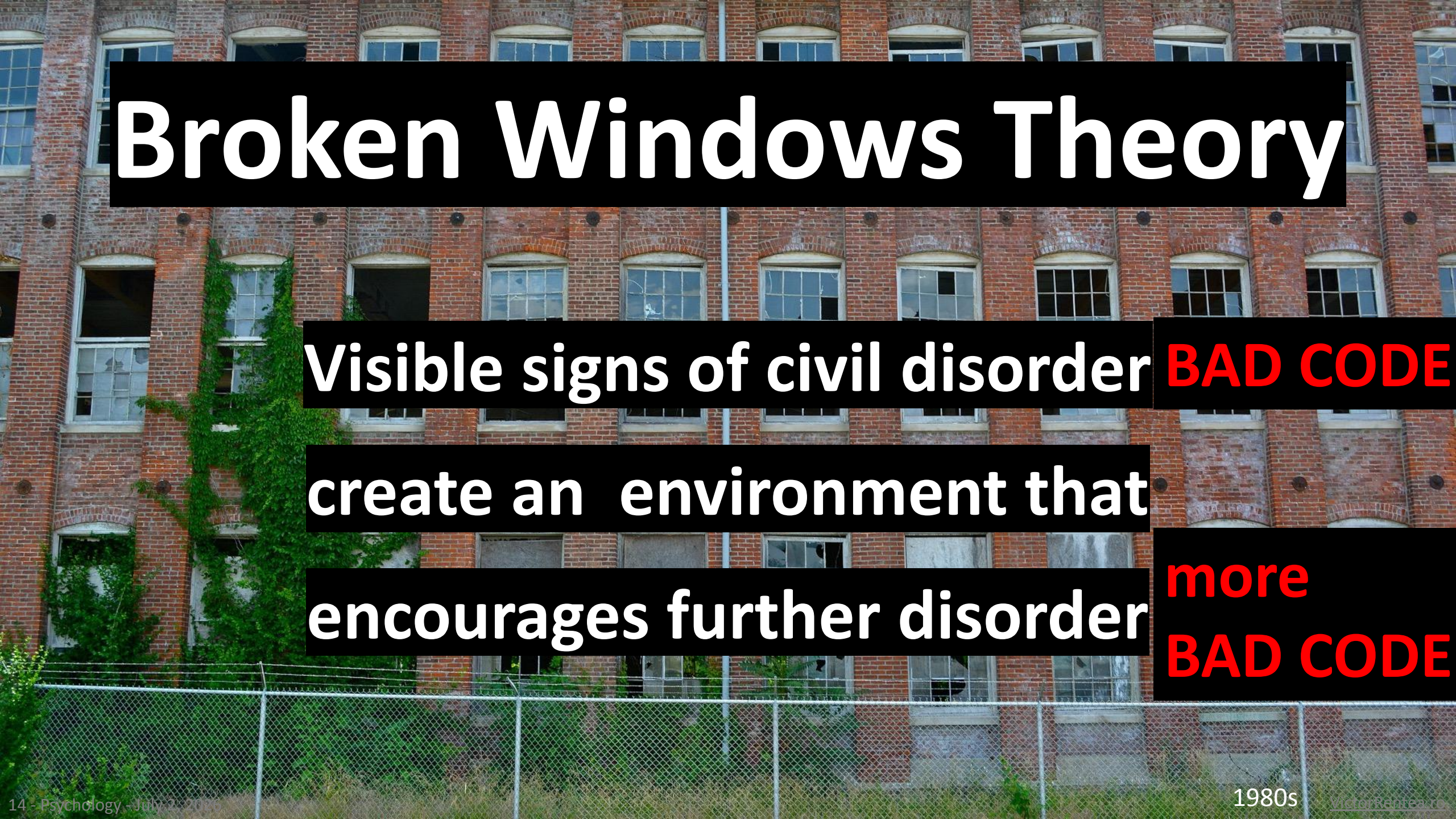
Bike shed
cost: $1000
discussion: 45 minutes

Careless Code

Bike-Shedding

VS

The more trivial a topic, the more debates around it



1980s

Dunning–Kruger Effect

People **with limited competence** in a particular domain might **overestimate their abilities**.

It's a simple task. It should take you max 2 days.

- the manager who did web design when he was young

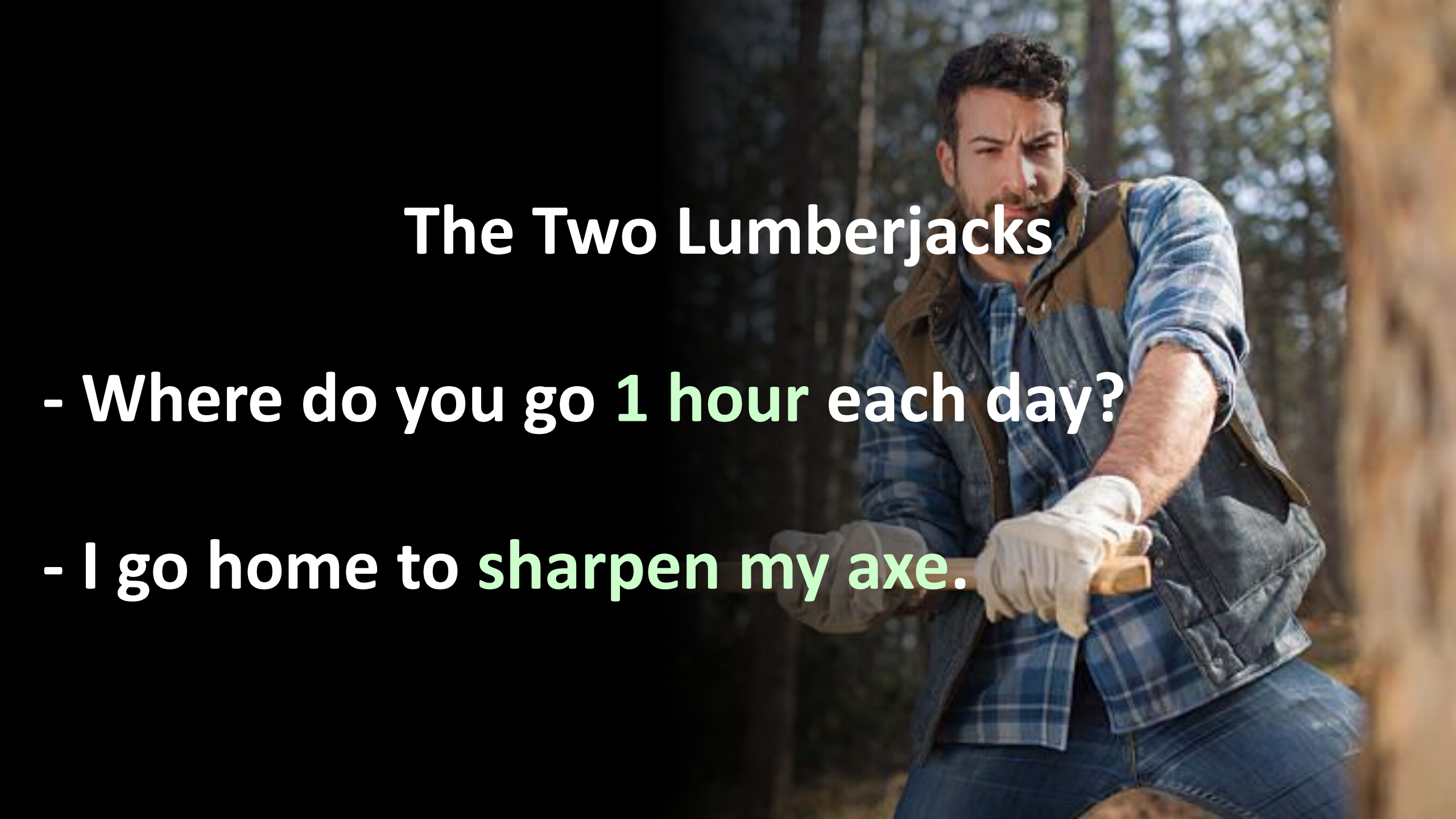
https://en.wikipedia.org/wiki/Dunning%E2%80%93Kruger_effect

The Long Journey

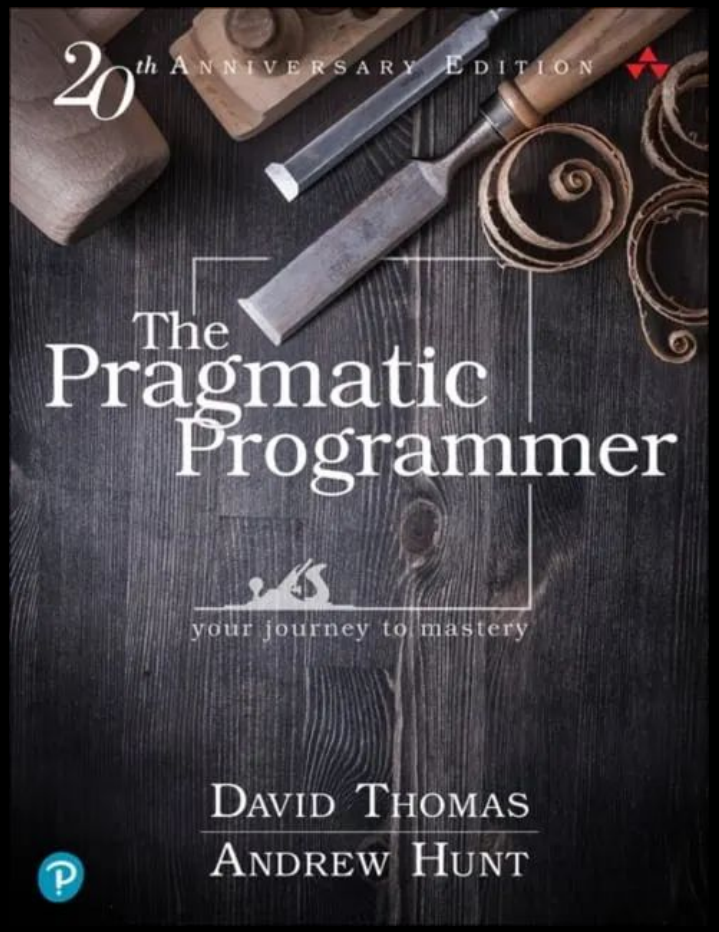
The Two Lumberjacks

- Where do you go 1 hour each day?

- I go home to sharpen my axe.



Your Learning Day



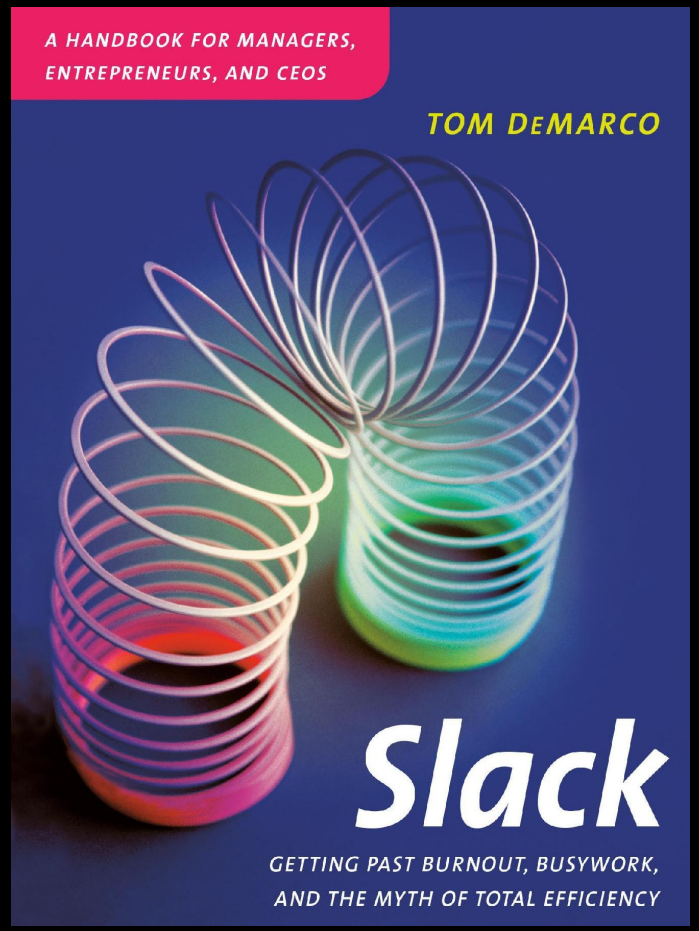
1999

Sharpen Skills

Automate Chores

Study Fundamentals

Improve Workflow

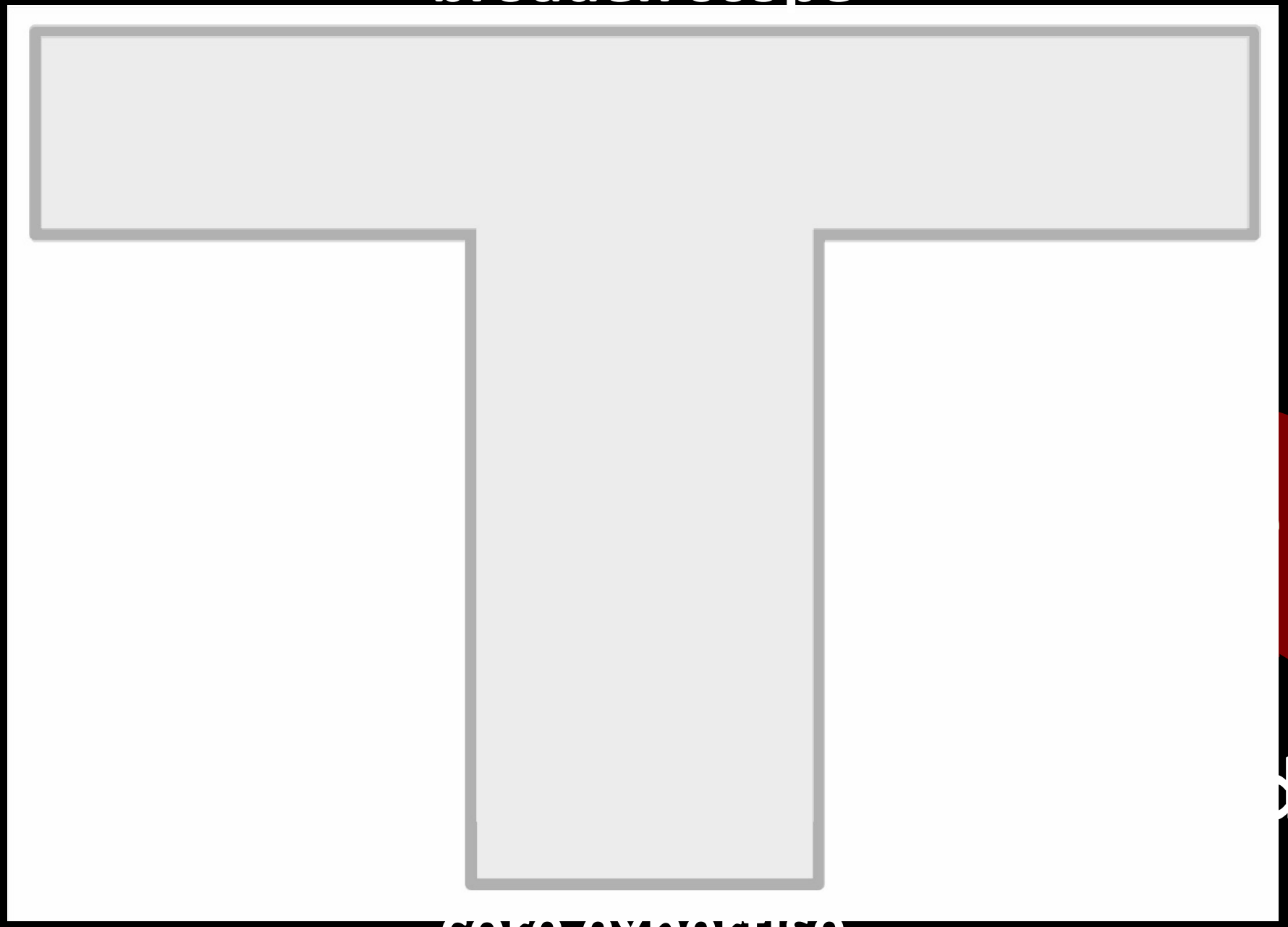


Your Learning Day

- We had one?!
- We use it to fix more bugs

= Stealing from your boss!

<< broaden scope >>



r-Stack,

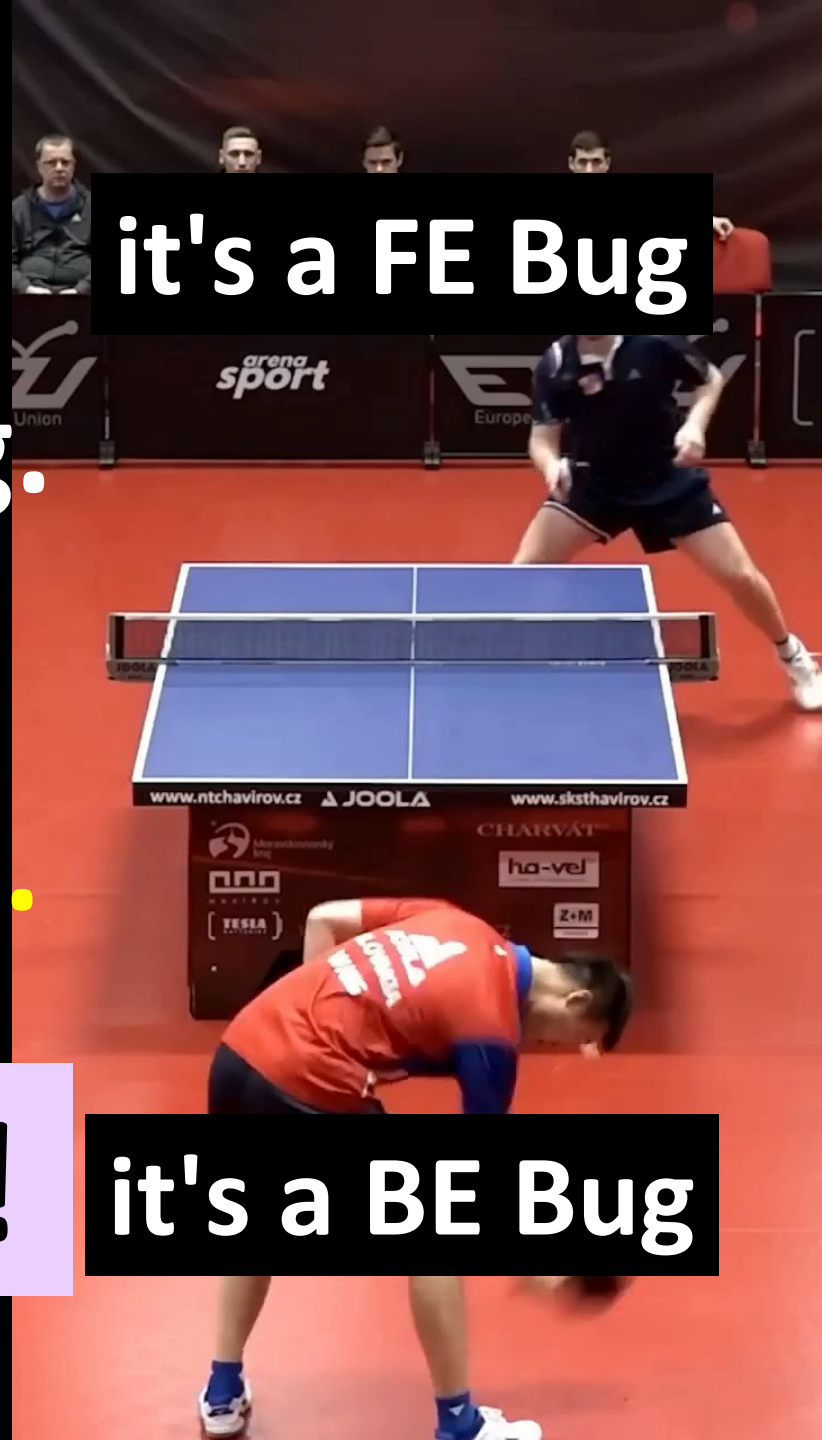
d Developer

core expertise

Less Ticket Ping-Pong.

Ship more value.

Go Full-stack!



The Psychology of Computer Programming 1971

Software quality is determined more by human psychology and team behavior than by programming languages or tools.

- **Code for people**: read >> write ⇒ clarity >> cleverness
- **Ego-less** programming ⇒ review the code, not its author
- **Debugging**: confirmation bias, attachment, premature theories
- **Team communication** matters more than individual brilliance
- **Dev Productivity** is hard to measure
- **Learning requires feedback**: review, discussion, experiments

US Navy SEALs

Who makes it through the selection?

I'll tell you who don't, - said a SEAL.

An elite warrior always helps the others.

⬇️ Get a Job

- **Junior Programmer**: responsible to **implement** a task

⬇️ Study Hard

- **Senior Engineer**: responsible to **solve** a problem

⬇️ Get "Promoted" 🎁

- **Team Lead**: responsible for **people** solving the problem

Team Lead Stuff

No matter what a problem seems to be at first,

it's always a people problem.

– First Law Of Consulting

Leading is Hard

People are hard.

Interruptions are draining 😩.

Must take responsibility of failures, 🤕

But give away the credit to the team 🤗.

Everyone can become a lead! (or a parent)

But some won't.
And some shouldn't.

Lead with Empathy

You are not in charge of the people,

You are responsible for the people!

Lead with Empathy

A Team Lead should build

A Healthy Environment

Asking Questions is Hard

Asking Questions is Hard

- What would they say (of me)? 🙈
- Who am I to disturb Sauron? 👴TODO

eye

- Hard to articulate the question... 🤯

I DON'T KNOW!

NEITHER DO I.

NEITHER DO I.

BUT LET'S FIND OUT TOGETHER!

BUT LET'S FIND OUT TOGETHER!

How to ~~Google~~, AI-drive my learning, evaluate, compare

Your Senior / Lead / Architect / CTO:

CAN I HELP YOU?

A new colleague
joins your team...

⇒ Pair Program
for ≤ 3 months

Make it safe to ask for help!

Fear of Failure

For any non-trivial problem

the first solution

is always wrong.

Afraid to fail ⇒

Analysis Paralysis

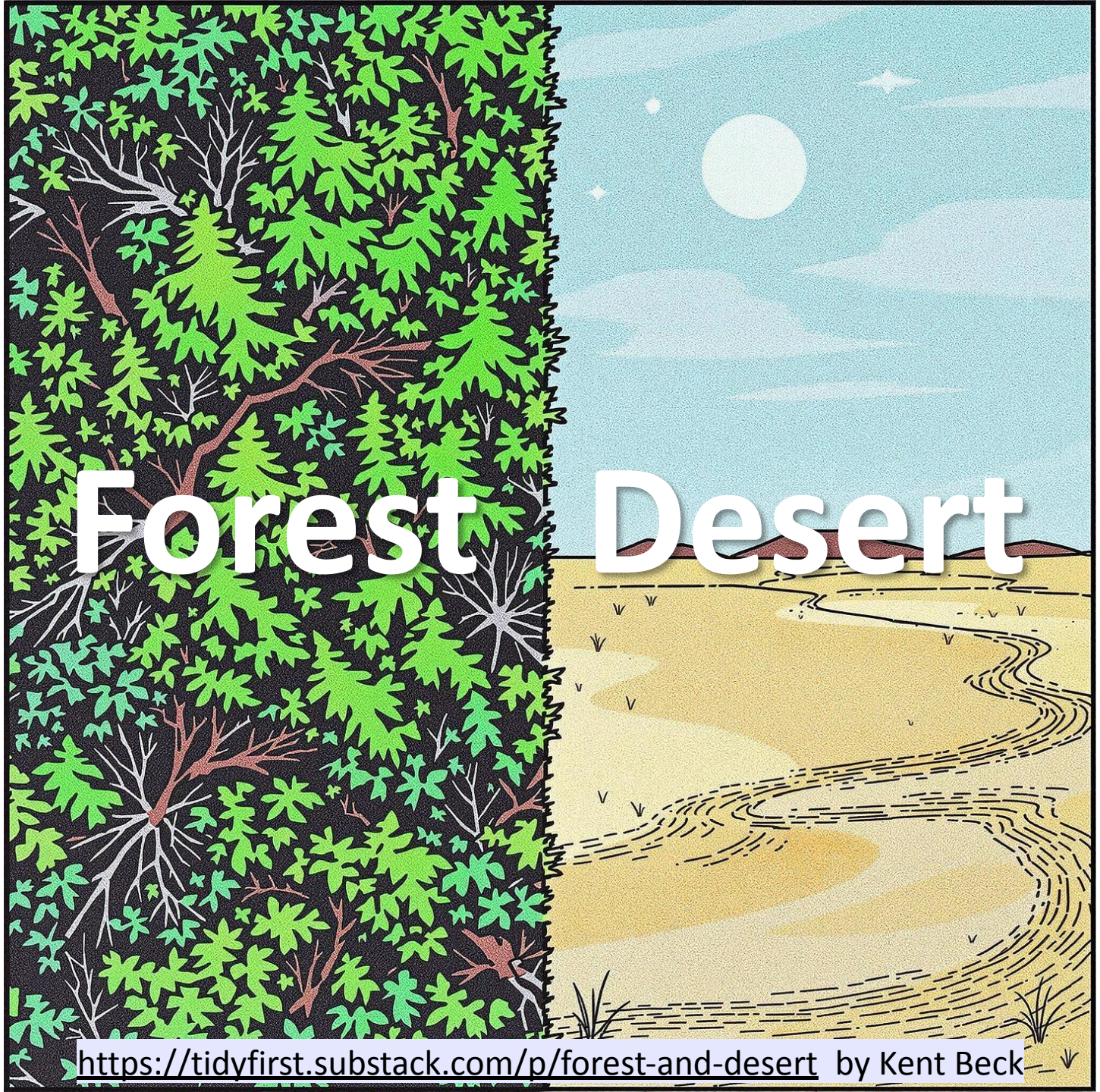
Go Agile: MVP + Iterate ↺

Make it Safe to Fail!

More experiments

Faster innovation⭐

Are you **DevOps**✊ enough?



https://tidyfirst.substack.com/p/forest-and-desert by Kent Beck

A Healthy Environment

> **Safe to Ask**

> **Safe to Fail**

Recently, we conducted a survey to understand your feelings about stress at work. Many of you shared your concerns, which we deeply value

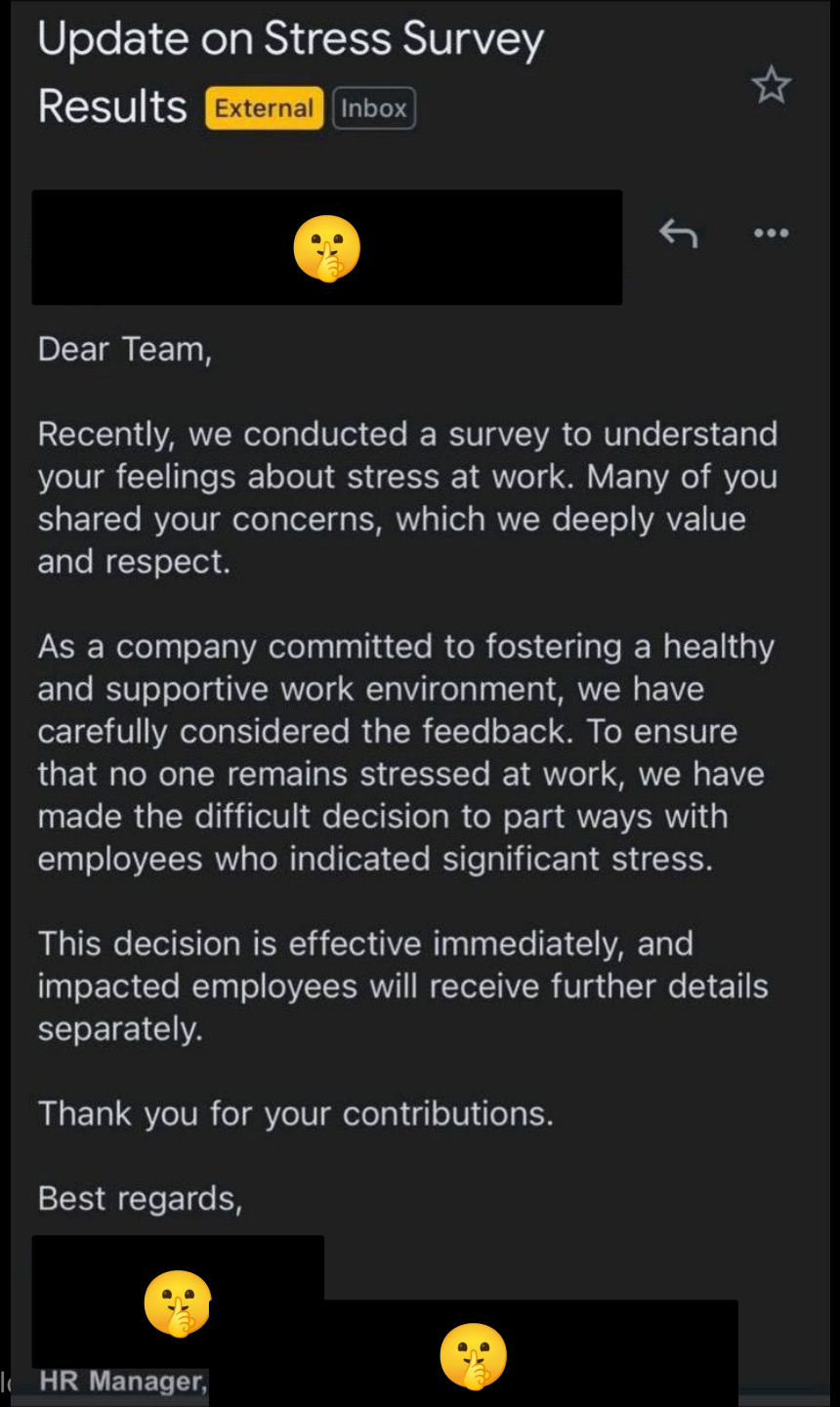
In the meantime....

Feeling Stressed?

As a company committed to fostering a healthy and supportive work environment, we have carefully considered the feedback. To ensure that no one remains stressed at work, we have made the difficult decision to part ways with employees who indicated significant stress.

You're fired!

This decision is effective immediately, and



Some Just Don't Care

Feeling Stressed?

You're fired!


Legacy Code
- You Inherited +
- You Wrote

Meet Spartacus,

The Refactoring Hero

✓ No Bugs
✓ Insanely Fast
✓ IDE Guru
✓ Highest code quality standards

Focal Refactoring

Spartacus was here

Team:

😵

👍 Consistent Code Style

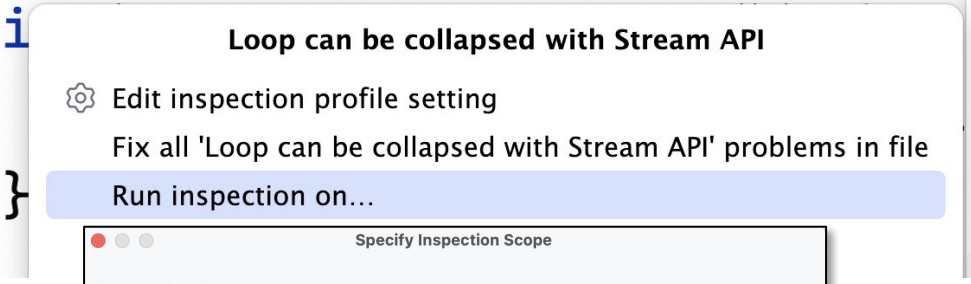
(Tiny) Refactoring Everywhere:

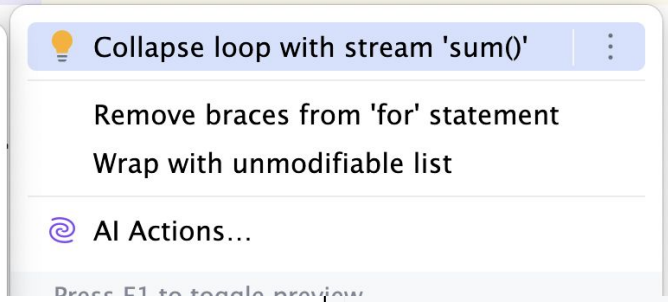
- IntelliJ Inspections & Structural Replace
- OpenRewrite ± Custom Refaster Rules
- Refactoring War Room (all hands 🙌)
- Crowd-Source it ⇒

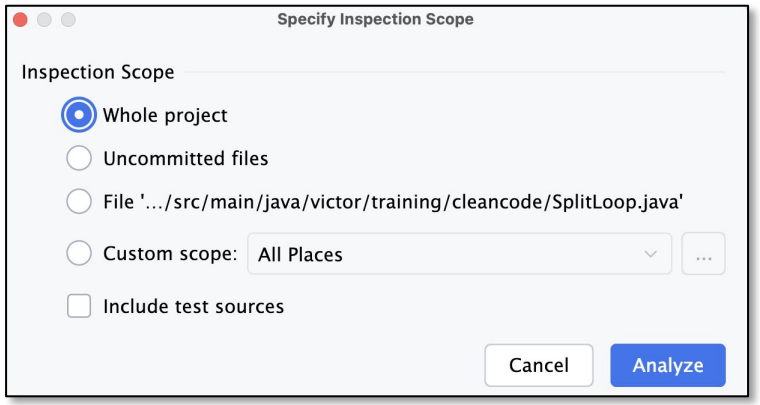
⭐ Team Morale

1) IntelliJ Inspections applied on the entire project

```
double totalConsultantSalary = 0;
for (Employee employee : employees) {
    i
}
```







Prefer Tiny Changes Applied Consistently Everywhere

2)

Productive Teams

A team will always outperforms the sum of us alone,
no matter how great/10x-dev you are = "Synergy"

I'll just throw more people in that team!

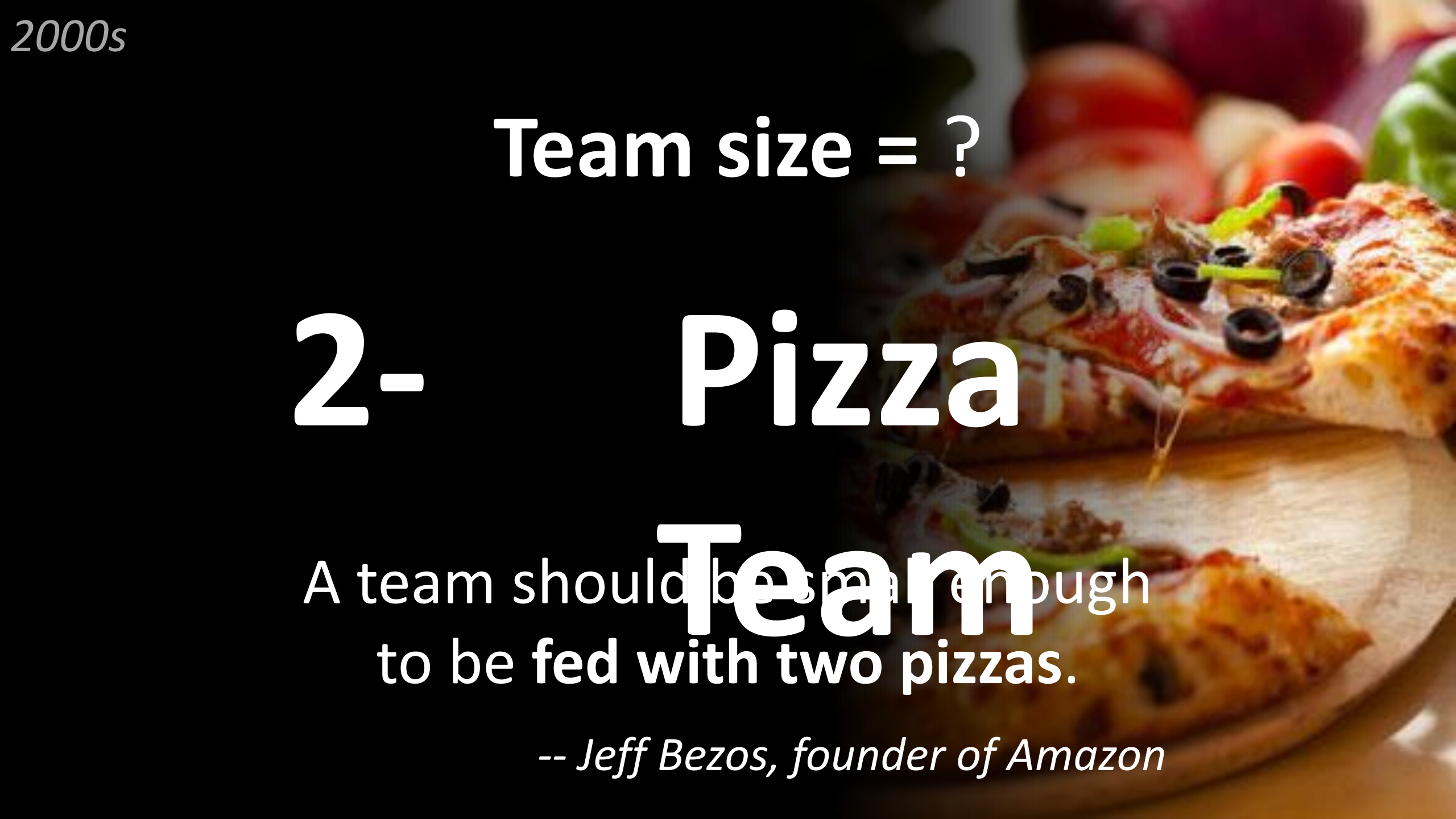
2000s

Team size = ?

2- Pizza Team

A team should be small enough to be **fed with two pizzas**.

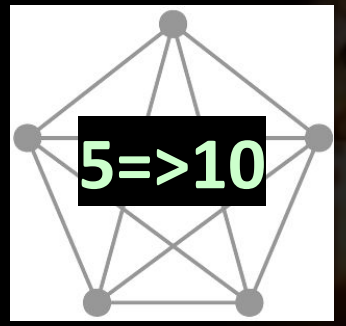
-- Jeff Bezos, founder of Amazon

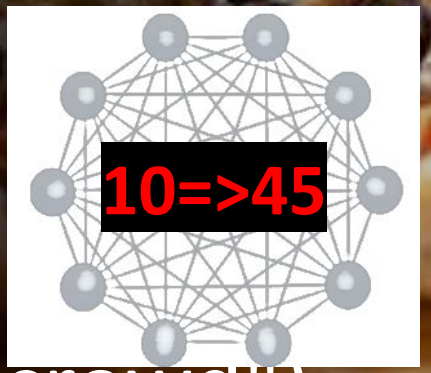


2000s

2-US-Pizza Team

=5..7 people - #onboarding/2



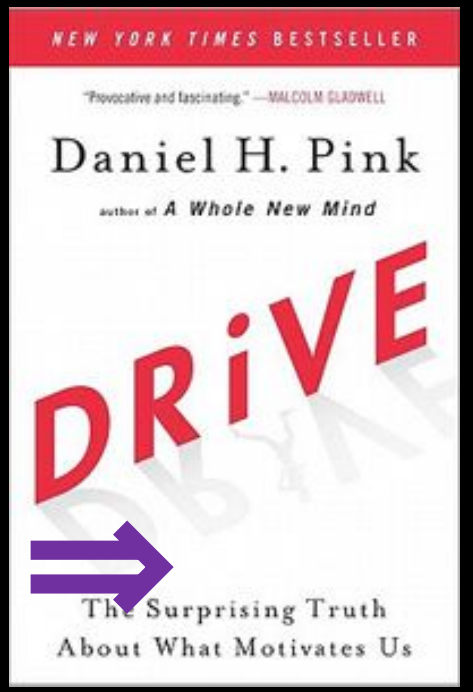


- **Less Communication Paths**
- **Personal Ownership** (can't "hide in the crowd")
- **Trust** to speak-up, innovate, and challenge ideas

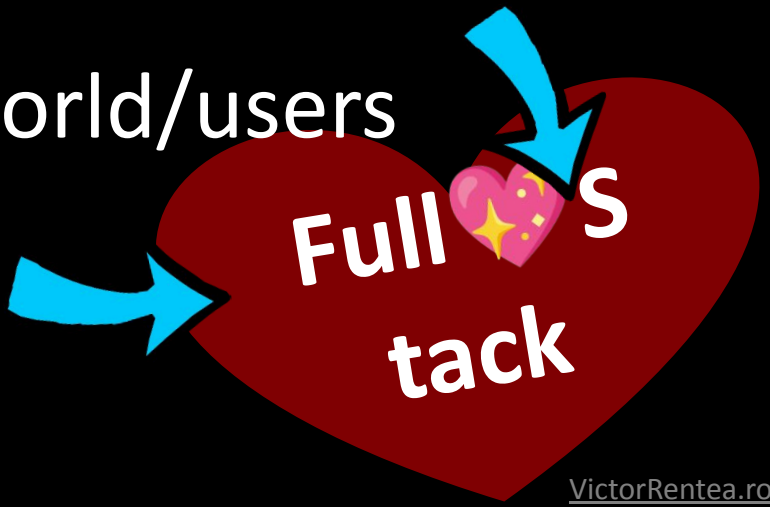
https://martinfowler.com/bliki/TwoPizzaTeam.html

2011

Motivation Drivers



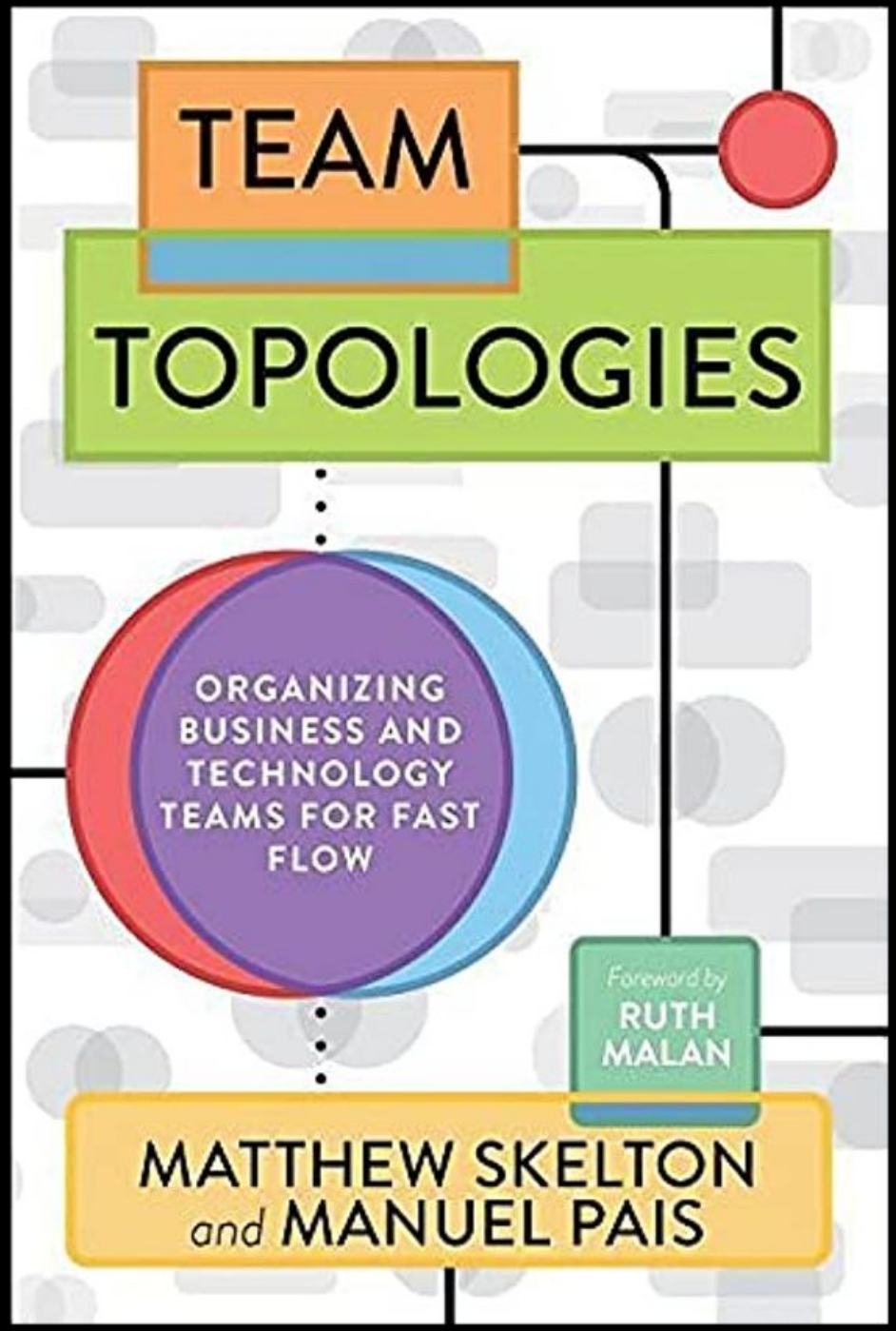
- ✔ **Mastery** ⇒ Limit cognitive load per team
- ✔ **Autonomy** ⇒ Limit communication between teams
- ✔ **Purpose** = See your impact on the world/users



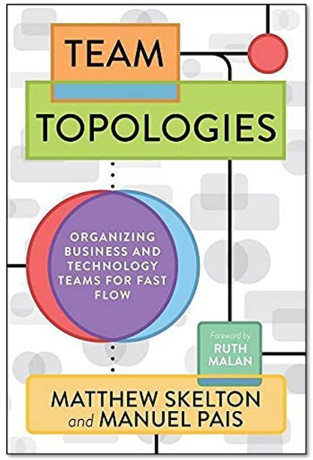
Software that fits in team's head.

- a talk by Dan North

2019

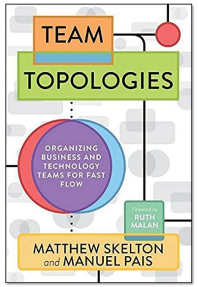

TEAM
TOPOLOGIES
ORGANIZING BUSINESS AND TECHNOLOGY TEAMS FOR FAST FLOW
Foreword by
RUTH MALAN
MATTHEW SKELTON
and MANUEL PAIS

🧠 Cognitive Load for a Developer



- **Core Skills** ← **Upskill** by an 👩‍💻**Enabling Team**
  - *How to write classes? ...use my framework? ... write tests? ... deploy?*
- **Accidental** ← **Support** by a 🤓**Platform Team**
  - *Where to see the log of all instances? ...provision an environment?*
- **Problem to Solve** ← What the team **should focus on**
  - *Refund with partial voucher?*

2019



Team Topologies

Stream-Aligned Team

fast feedback

Customer

Enabling

Trainers & Architects

Complicated Subsystem Team

legacy / deep domain

Stream-Aligned Team

Full-stack DevSecOps team shipping end-to-end features fast

User

🤓Platform Team: increases autonomy of SA

Dev❤️Ex: Support, Wikis, Infra, Ops, k8s, Libraries & Services

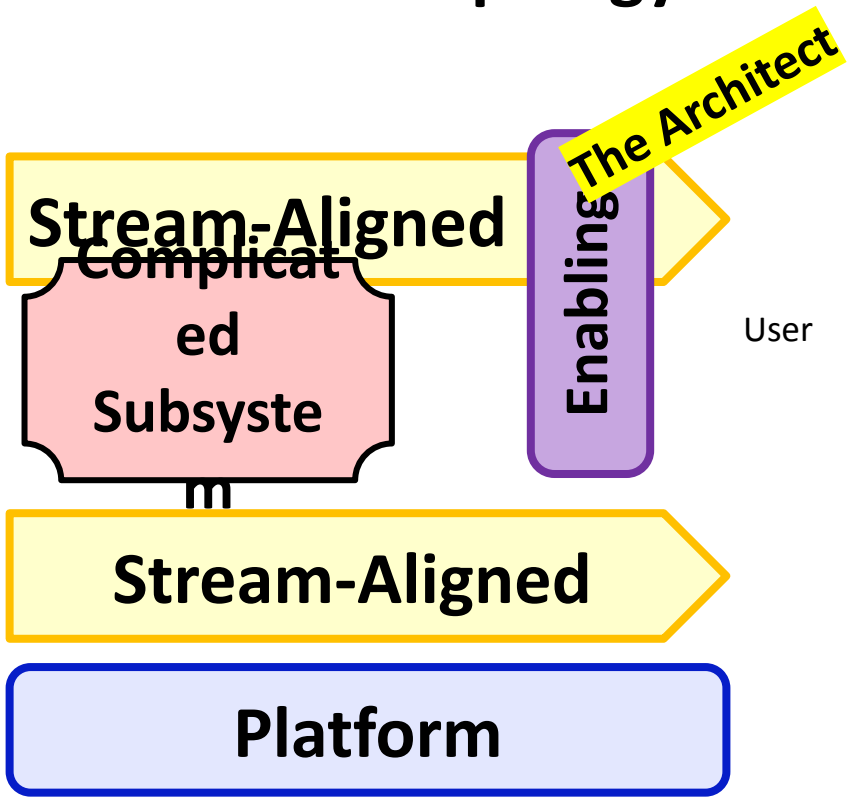
⚔️ Platform Fail: do what..
>no team want to do
>2+ teams want to do

🎫 Golden Pass: fully automated provisioning of a new service up to prod

Summary Article and Intro Video to the Team Topologies Book

Team Structures

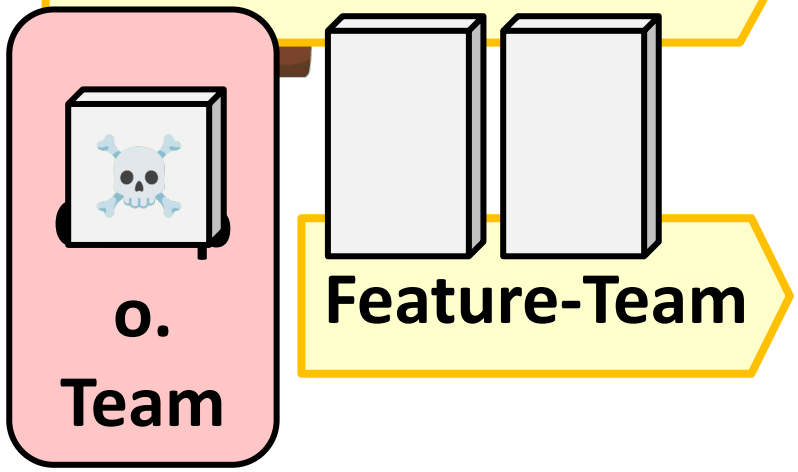
Team Topology



Component vs Feature Team

"Change any Git repo to get it done"

Feature-Team

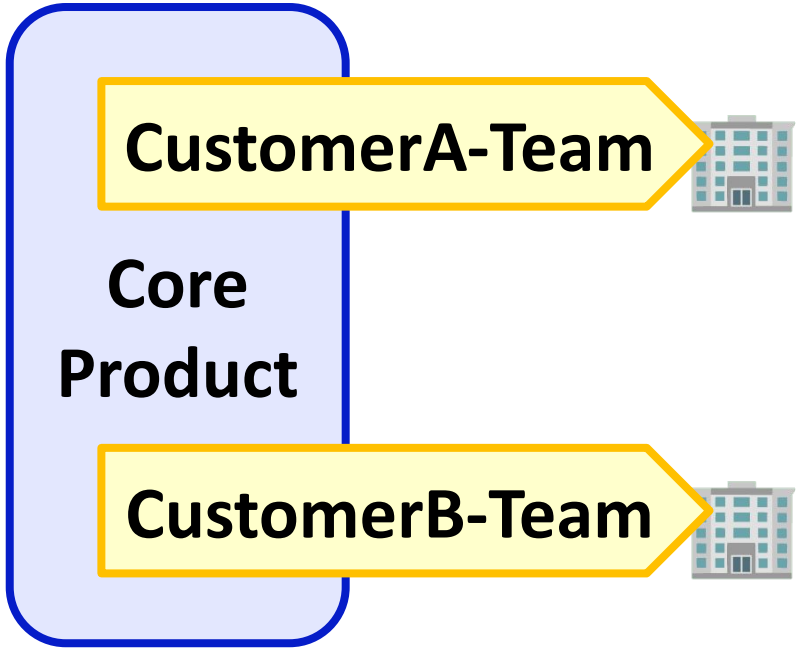


👆 Assembled around a complex/legacy codebase

Light Alternative:

PR approve by SME (via CODEOWNERS)

Product vs Customer



Role of "Platform Team"

⭐ **Reduce the cognitive load** of developers (microservice ecosystem)

- **Manage infrastructure**
  - Kafka cluster, DBs, KeyCloak, Metrics & Tracing
- **Write wikis**
  - how to setup my dev environment, tutorials how to create a topic ..
- **Develop libraries** to standardize:
  - Authentication, TraceID, Integration Testing with Testcontainers
- **Enforce best practices**:
  - pairing, code scanning tools (Sonar, Refaster), dependabot/refaster
- **Develop the core backbone product**
  - Customer-specific teams configure and customize it

Feature- vs Stream-Aligned Teams

	Stream-Aligned Team	Feature Team
Definition	Delivers an end-to-end **continuous value stream.**	Delivers **specific features**.
Autonomy	High: delivers and operates independently.	Limited: relies on other teams.
Duration	Permanent, aligned to a stable domain.	Temporary, feature-based.
DevOps Responsibility	Responsible also for operation.	May not operate what it builds.
Example	Team managing the entire payment process: transactions to reconciliation.	Team developing a checkout module but relying on a payment team.

Platform Engineering Challenges

- Set up feedback loops from start
  - "50% of devs would know about a problem before *you* find out"
  - Have regular interviews with users, 1:1s not surveys (questions bias the answers)
  - Show up at the team's standups.
  - Embark devs to platform team only after going through product teams = "customer shadowing", to find out where the pain and time loss is.
  - Categorize users (developers) in personas
- **Learn to $ell the value of the Platform** (hard for many developers):
  - "you can do this in 1 hour instead of 1 week" regarding testing, deployment, observability;
  - Agregated across multiple teams, requires a holistic view of many teams
  - Should be supported by a higher-management vision (eg VP).
- **Only add to platform stuff needed by 2-3 projects**. Otherwise, build a library.
- **Target small projects first, not the big fishes**.
- **You don't want to build a platform,** but you want to establish a "platform mindset".
  - Start small, eg with a google doc "how to start a new service?"
  - Consider buying/adopting external toolsa

https://www.infoq.com/presentations/art-platform-engineering/

An Effective Team

- Kick-off = BA hands over 1:1 (explains) the ticket to implementing DEV + QA
  - ~~Grooming can marginalize introverts~~
- *They* banned 1:1 chat (can't DM anyone) made safe all on #general, stored forever
  - Mult mai multa grija la ce scrii. doar in RO e vina si rusine post-comunista
- Mandatory Documentation updated in Confluence ± OpenAPI ∈ "Done"
- KT between DEVs after implementing a complex feat or bug – 1/week
- On-call support in pairs sr + jr/mr
- Official **Pair Programming** (rotated)
- "3 amigos" = anyone can call an on-demand brainstorm session +2 others
- Demo 100% feat to BA (in DoD) usually release right after demo (even daily=CD)
- Using Kotlin 💖

Shared Library / Microservice Chassis

- **Problem:** implement common tasks in services without copy-paste
- **Solution:** create and manage a **shared library** helping with technical, non-domain specific tasks.
- **Debates**🤔
  - Specialized libs (-commons-security-v5') or a single large one ('commons-v12')
  - Should clients use version=LATEST? Dependabot auto-upgrade?
  - What's the deprecation policy?
  - Who is paid to maintain the lib? Who's its GateKeeper?
- Why turn a lib into a service?
  - a) better support high rate of change
  - b) reuse across languages
  - c) clearer boundary and ownership

Feature vs Component Teams

A **cross-functional team,** responsible for delivering **end-to-end features** of a software product

A specialized group that is responsible for a specific **function** or **domain** of the software

- **Faster** Delivery & **Progress** Tracking: **FT** (less dependencies)
- **Holistic Vision**: **FT** understands better clients' scenarios
- **Planning/Budget**: **FT** easily map to budget; CT devs may be idle
- **Domain Knowledge**: **CT** can specialize in the tech/subdomain
- **Quality**: **CT** build better architectures but ping-pong bugs
- **Scale: CT split** = domain split; too many FT => collisions

https://medium.com/mercury-business-services/scaling-development-teams-feature-vs-component-teams-a195c99dab21

The Architect


ANTI-SOCIAL
and stupid (often)!
I'M AN
ARCHITECT.
TO SAVE TIME LET'S JUST
ASSUME I'M ALWAYS RIGHT.


AM I THE ONLY ONE
"THE ENFORCER"
WRONG
WHO CARES ABOUT PROPER SOFTWARE ARCHITECTURE
memegenerator.net


"THE ENABLER"
SOFTWARE ARCHITECTURE
IS ABOUT COMMUNICATION

The Ivory Tower Architect

Who stopped writing code 5 years ago.

But wants to decide today
the libraries and design patterns
we should use in code.

Decisions Must Come with Accountability

= you must experience their consequences

Questions for Finding Design Hotspots

Quarterly questions to ask a dev team:

- Longest Class
- Highest Cyclomatic Complexity method/class
- Largest package
- Total number of TAB(indents) in a class
- File with highest number of bugs
- File with highest number of commits / month
- Class that scares you the most
- What is your ugliest test

THE COBRA EFFECT

A WELL-INTENTIONED MEASURE CAN OFTEN BACKFIRE AND HAVE THE OPPOSITE EFFECT TO INTENDED

RELEASED

TOTAL > INITIAL

😱

INTENTION

REDUCE COBRA POPULATION

ACTION

A BOUNTY FOR DEAD COBRAS!

EFFECT

PEOPLE START COBRA FARMING

Testing for Coverage

- GodService.java ..... 1200 lines 😬
  - 87% Test Coverage 🤩 written by a contractor
- GodServiceTest.java .... 2000 lines
  - 67 mocks
  - 200 x *when*(dataStruct.getB()).thenReturn(mock2);
  - 0 x "assert" ☠️
  - 0 x "verify" ☠️

getter mocked ⛔

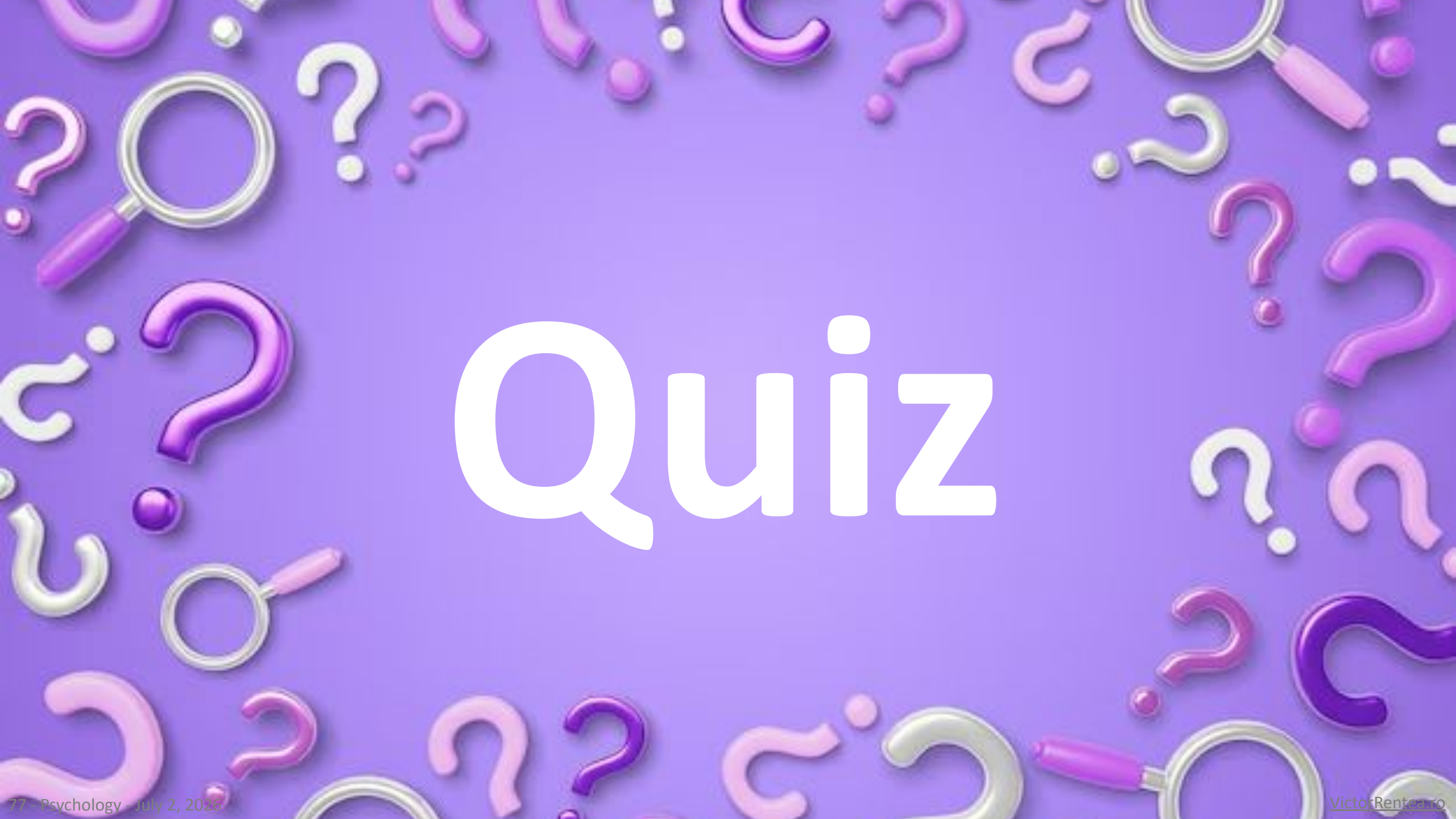
mocks returning mocks ⛔

87% Pass ✅? 🤔 Deploy 🚚 Prod 🤞

Mental Biases

= When your 🧠 already decided.
⇒ Systematic cognitive errors

- **Buzzword-Driven**: "Everyone is going serverless – so should we"
- **Survivorship Bias**: Netflix uses microservices (ignoring invisible failures)
- **Confirmation Bias**: we over-value evidence supporting our existing beliefs
- **Anchoring Bias**: we over-value the first option we hear (-20% off, 50K€)
- **Golden Hammer**: we want to keep using the familiar tool: $, PLSQL, Redis..
- **Sunk-Cost Effect**: can't abandon, long after it's obvious we should
- **Not Invented Here**: let's build an in-house framework instead of using OSS
- **Planning Fallacy**: we always under-estimate effort (= "optimism")
- **Dunning-Kruger**: amateurs feel over-confident due to unknown unknowns
- **Authority Bias**: over-value opinions of seniors even outside their expertise
- **Post-Traumatic System Design**: instinctual, fear-driven decisions


Quiz

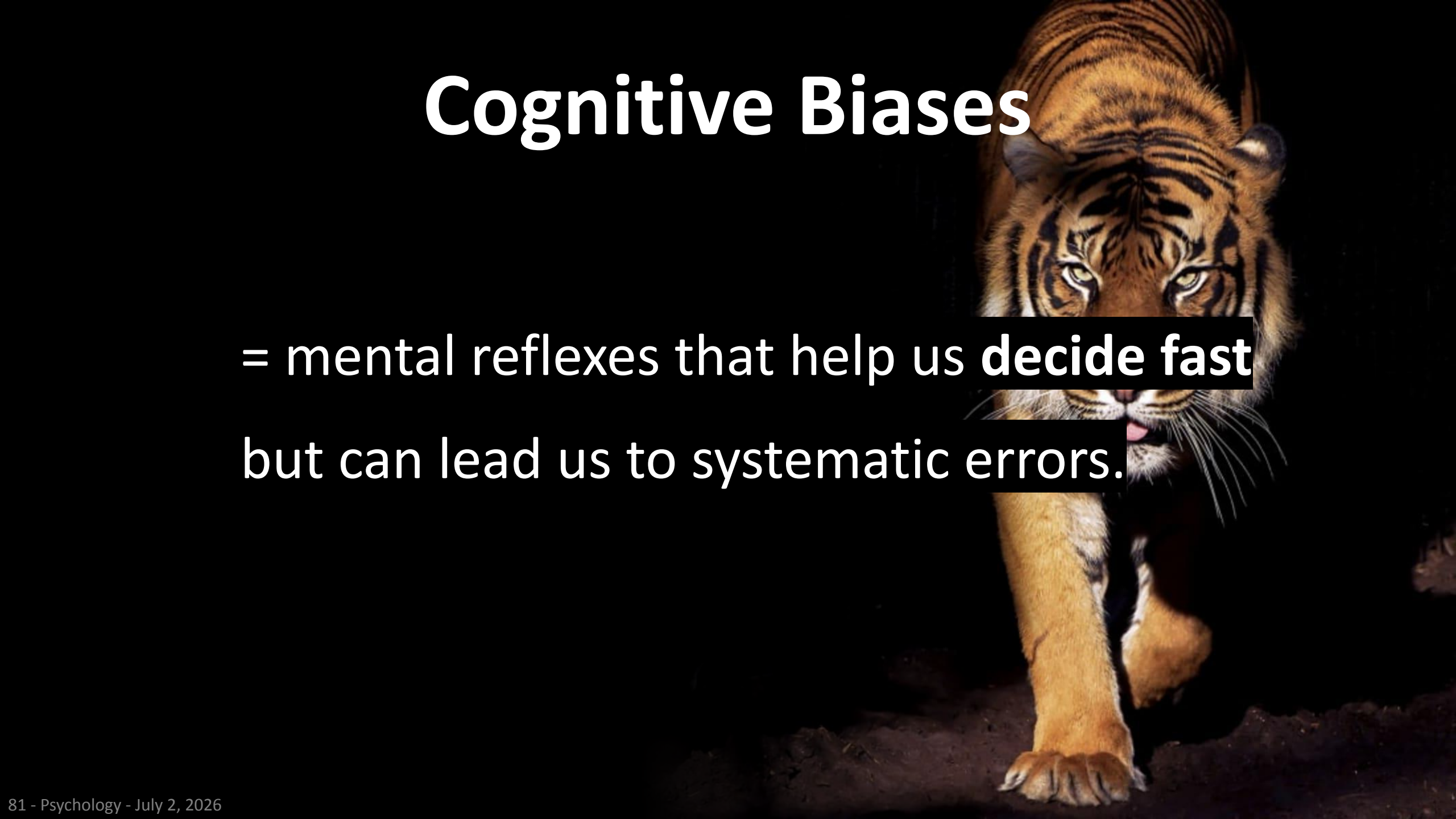
- I feel unqualified for this job (but you 🤘 rock) = **Imposter Syndrome**
- I Can't stop, I have invested so much in this! = **Irrational Artifact Attachment**
- What was the faster lumberjack doing 1 hour each day? **Sharpening his 🪓**
- Endless debates on low-impact topics = 🚲 **Bike-Shedding**
- Debating 2 weeks on the best of 5 design options: **Analysis Paralysis**
- Best way to onboard a new colleague = **Pair Programming**
- Key skill for a good leader = **Empathy**
- 3 Human Motivation Drivers = **Mastery, Autonomy & Purpose**
- How big should a team be? **≤ Two-Pizza**
- 4 Team Topologies = **Stream-Aligned**, **Enabling**, **Platform**, **Complicated compo**
- Takes design decisions without accountability = 🦄 **Ivory-Tower Architect**
- Following the letter, rather than the spirit of the law = 🐍 **Cobra Effect**
- Early adopting the latest & coolest trends = **Buzz-word Driven**
- Solving the problem with that old tool you know so well = **Golden 🔨 Hammer**


Slides: http://victorrentea.ro/psychology >>
victorrentea.ro
Be the change
you want to see in the world!
- Mahatma Gandhi
🙏
Thank you!

Cognitive Biases

When your brain decides before you even realize it.

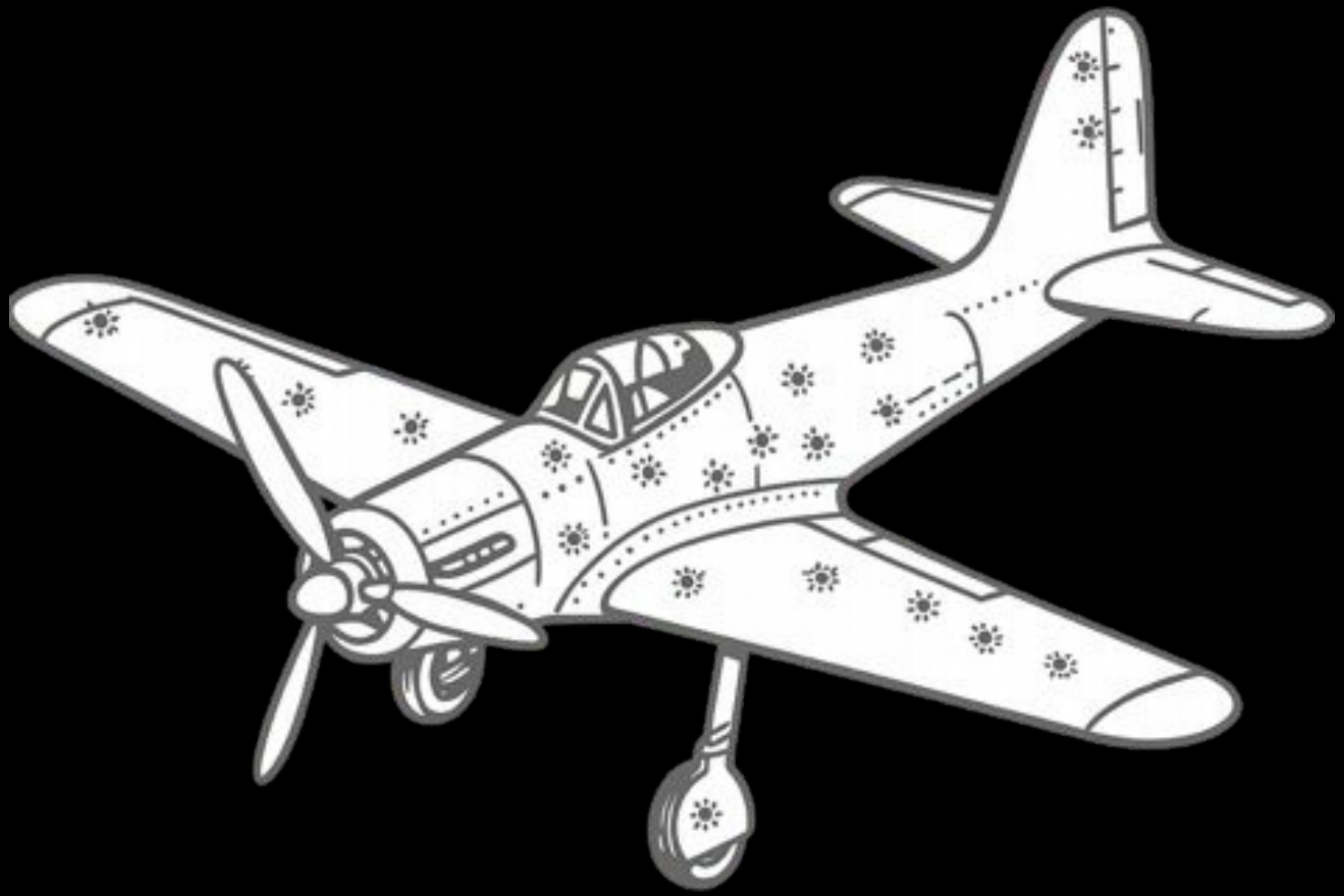
Cognitive Biases



= mental reflexes that help us **decide fast**

but can lead us to systematic errors.

Survivorship Bias



Survivorship Bias

Copying Netflix because it’s visible,

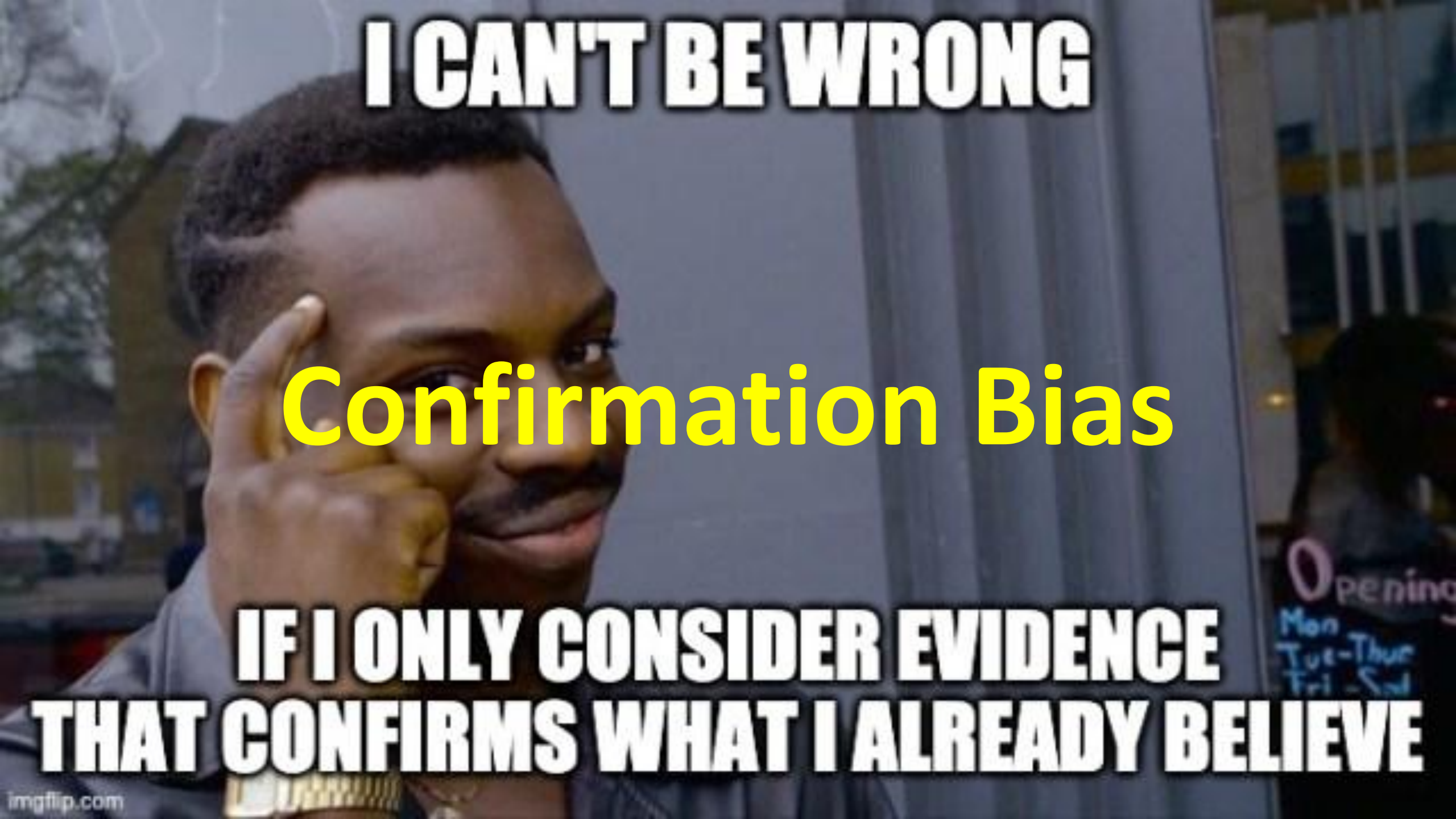
ignoring hundreds of failed “microservice rewrites”

We learn from survivors we observe.
Dead tribes didn’t leave stories behind.

NEXT- NETFLIX SYNDROMME

Overengineer today for potential high load tomorrow.

More Data => Better Judgement



Confirmation Bias

Confirmation Bias

- "Kafka is the right tool"

You only search for articles showing Kafka outperforming RabbitMQ

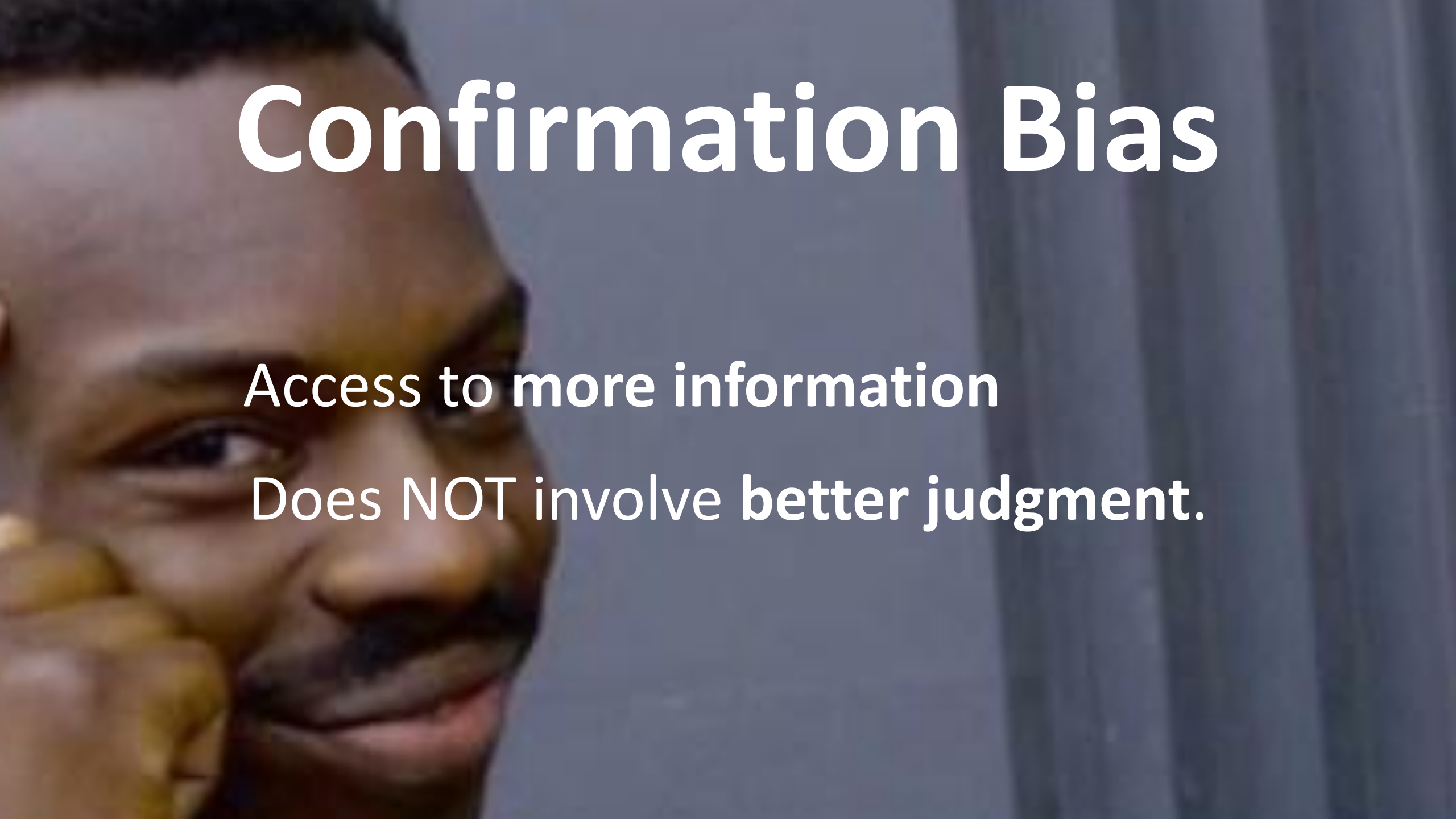
- "Microservice scale better"

When latency rises: “we need more pieces,” not “maybe we over-split.”

- "We should rewrite the legacy monolith to microservices"

Every bug proves it, though modularization could be cheaper/safer.

Doubting the shared 'truth' => social exclusion.
Our brain rewards evidence for what our peers believe.



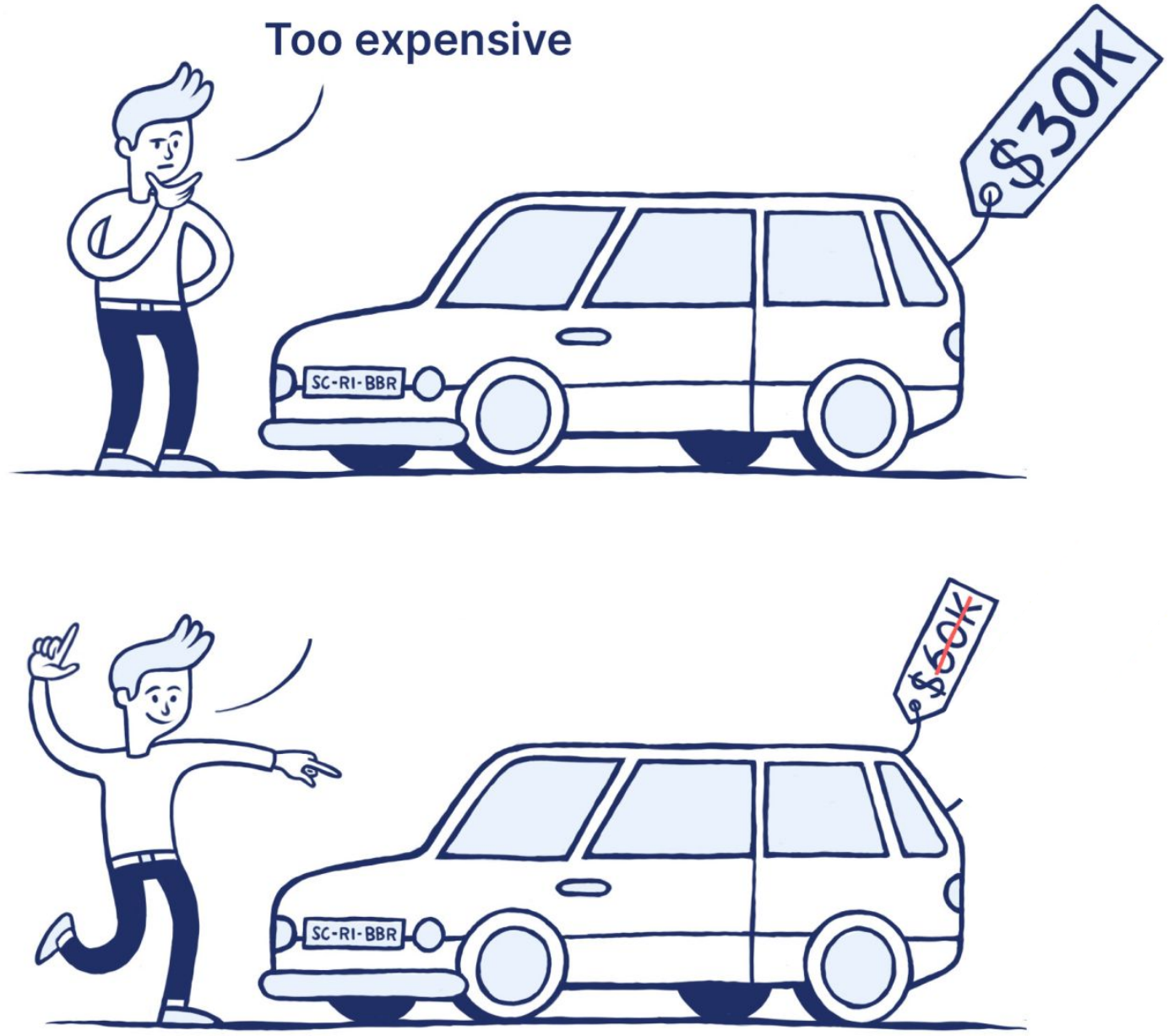
Confirmation Bias

Access to **more information**

Does NOT involve **better judgment**.

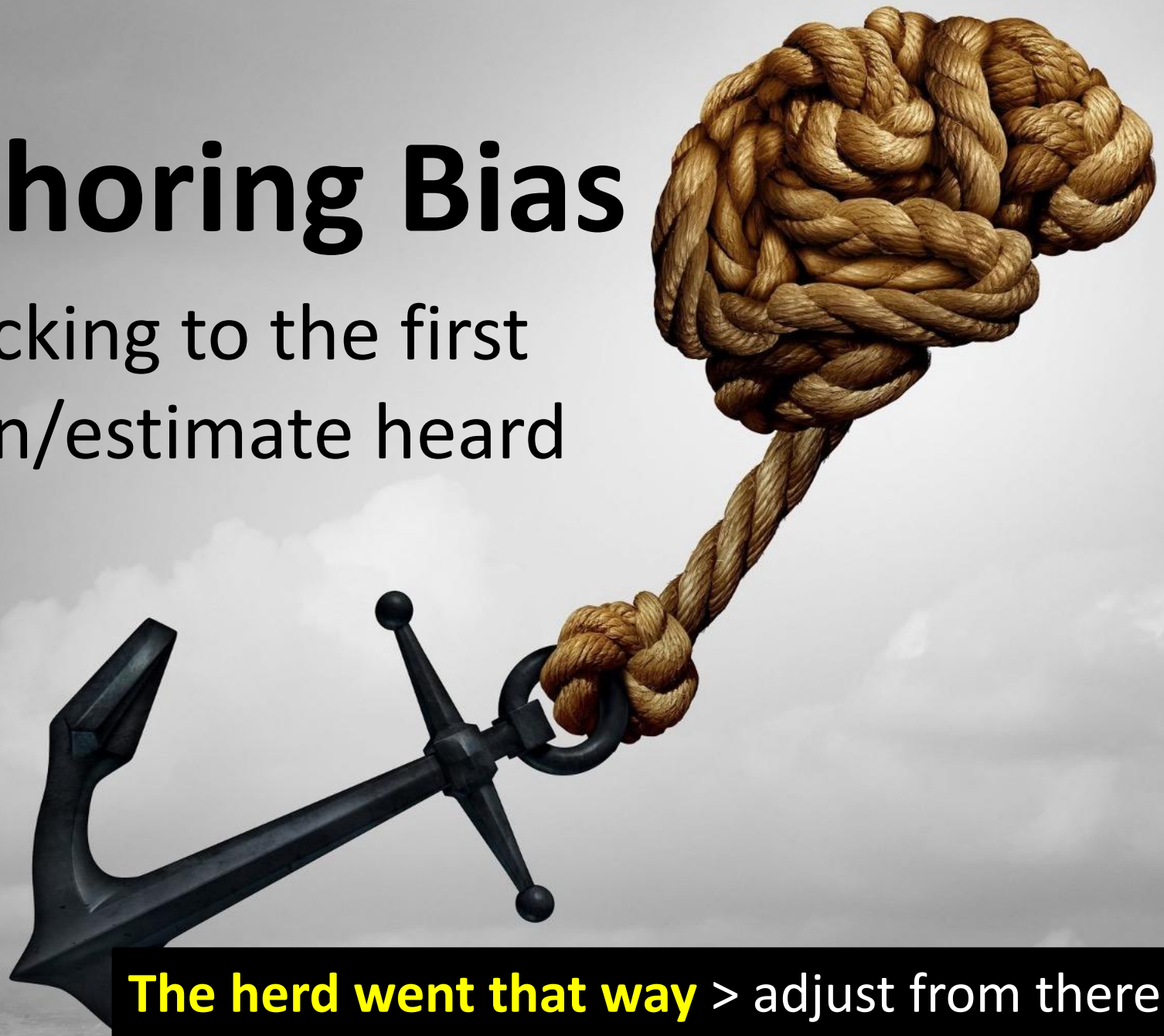

Echo Chambers
Social media, Google.. (*everyone)
feeds you content similar to
what you've previously liked.

Anchoring Bias



Anchoring Bias

= sticking to the first option/estimate heard



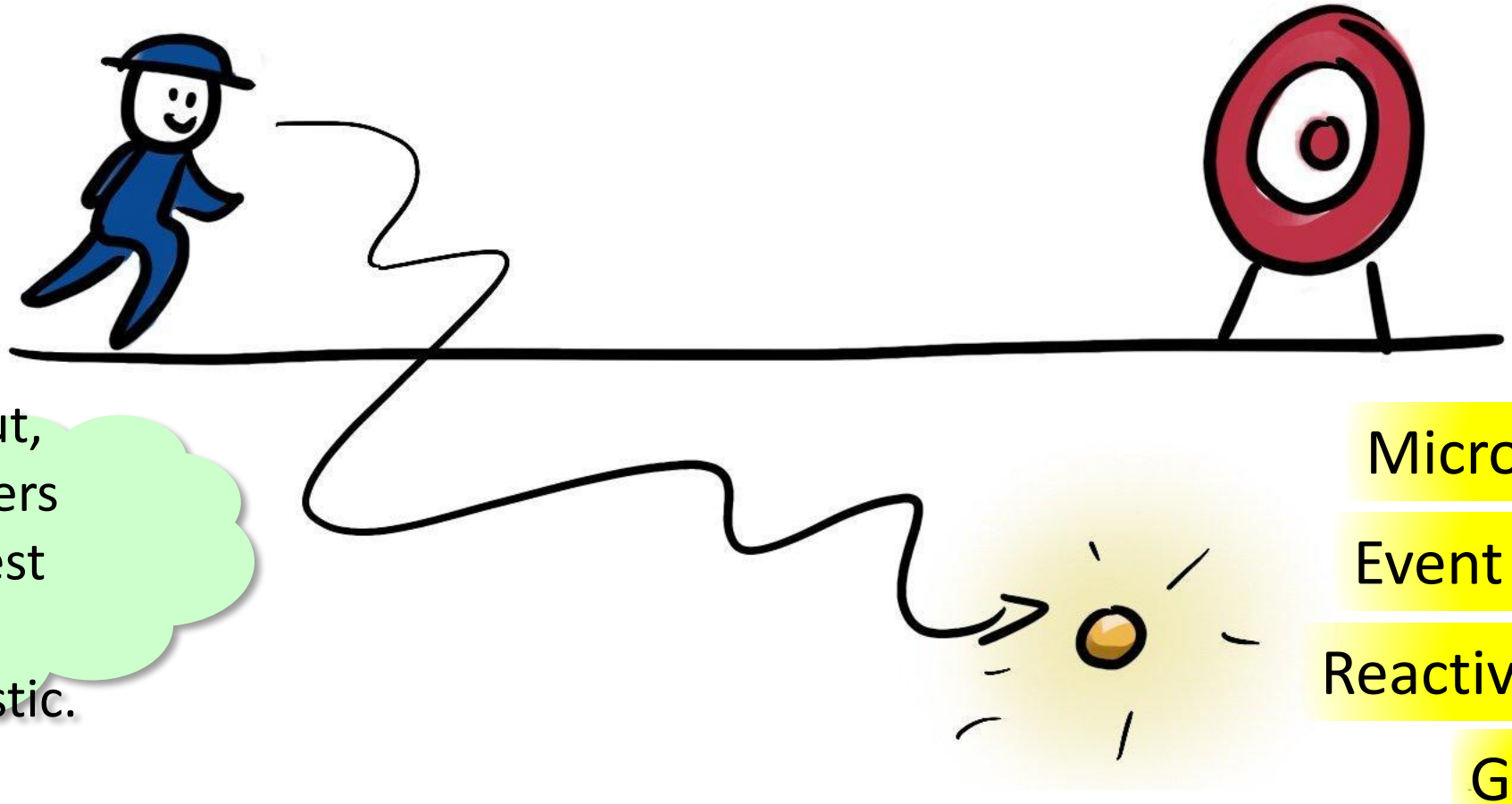
The herd went that way > adjust from there.
It's **faster/easier** than recomputing everything.

Sunk-Cost Fallacy

Abandoning a half-built shelter could mean death.
Persistence increased survival odds

Shinny Object Syndrome

Unethical Variant: Résumé-Driven Development



True, but, developers work best when enthusiastic.

Microservices

Event Sourcing

Reactive Streams

GenAI

Bandwagon Effect

Bandwagon Effect

Everyone is going serverless—so should we.

In tribes, survival meant conformity.
If everyone fled, you ran too—questioning was fatal.

Golden Hammer

To someone with a **hammer**,

everything looks like a **nail**.

- Mark Twain

The **known past was safe**.
Novelty often killed: new foods, new paths.

Not Invented Here Syndrome

We'll build our own framework
instead of that open-source tool (10K⭐/GitHub)

Ancient groups **distrusted outsiders**:
foreign tools could carry real risk (disease, failure).

Planning Fallacy

This migration should take “*just one quarter*”.

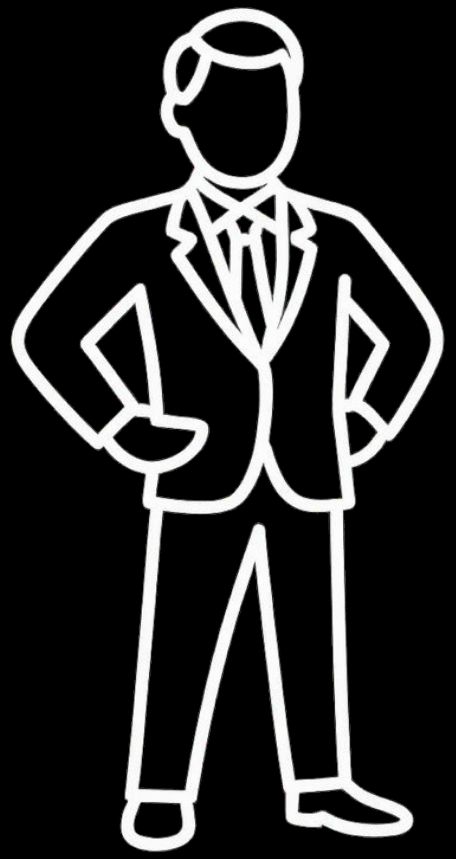
Optimism encouraged exploration and cooperation.
Pessimists stayed home and starved.

Dunning–Kruger

people with low skill
often **overestimate their competence**
because **they don’t know what they don’t know.**

Authority Bias

= overvalue opinions of people with authority
even if they are not experts in the topic.



If the senior said so...
who am I to disagree?

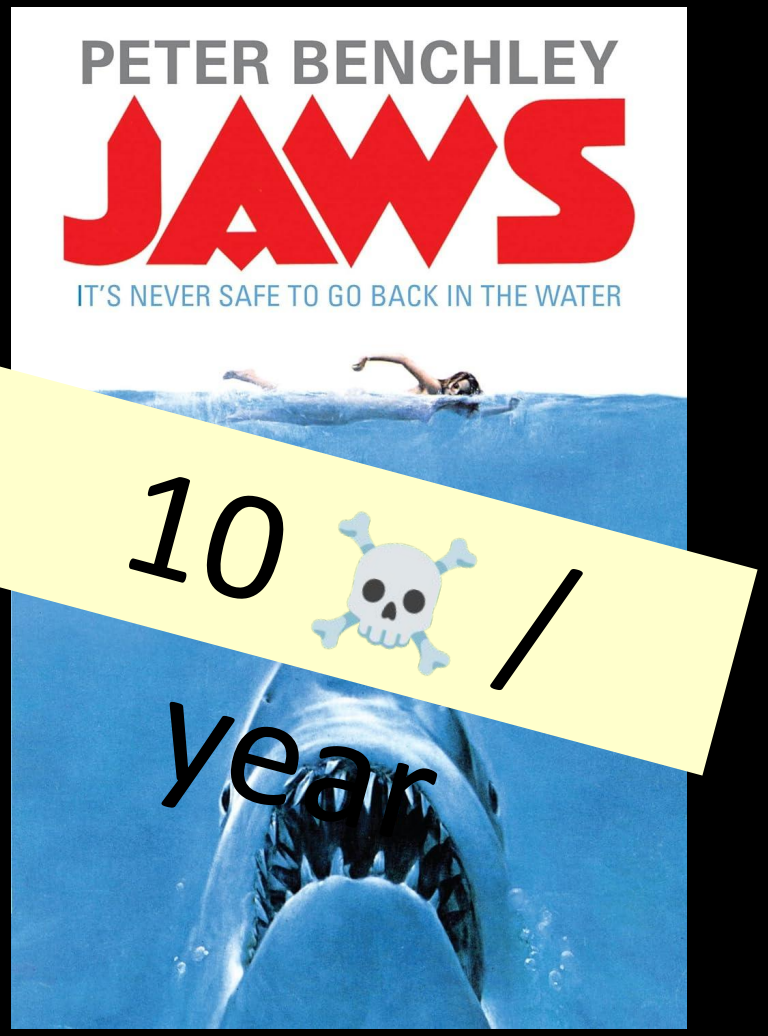
Valuable opinions are silenced.

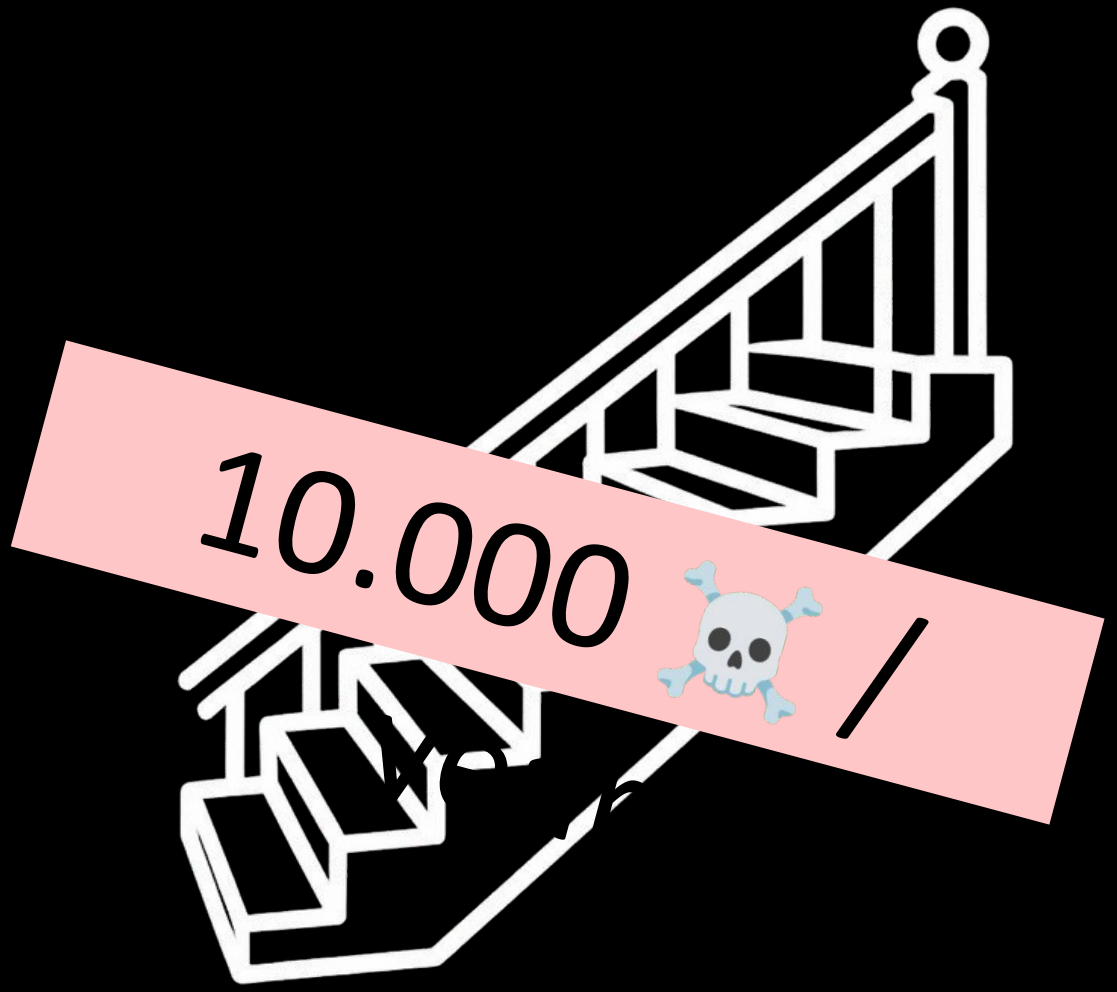
Overconfidence Bias

My feature will work perfectly without any testing.

Alpha confidence improved mating
and leadership status in tribes.

Availability Bias





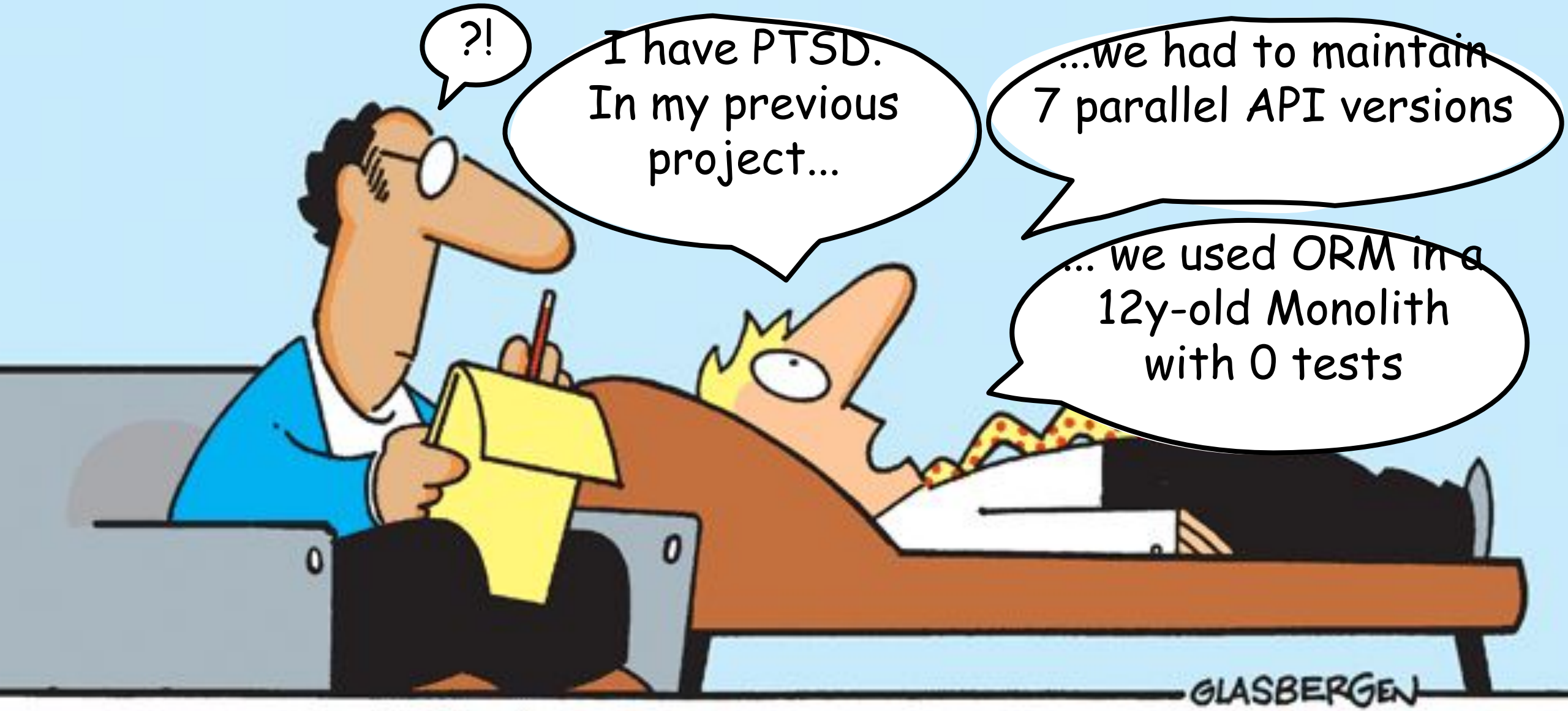
Availability Bias

One recent outage

$\longrightarrow$ overengineering high availability even for non-critical components.

We evolved to overvalue **recent vivid memories/stories** (e.g., a predator attack) to react faster next time.

Post-Traumatic System Design



Biases – What to do?

- Be **aware** of them => Detection
- We can't avoid them, but a **diverse group** can reduce them
- Challenge ideas / **assumptions**, be humble
- Use **data** whenever possible
- Build a culture where the **best idea** wins, not the most comfortable

Mental Biases: Recap

- Be **aware** of them => Detection
- We can't avoid them, but a **diverse group** can reduce them
- Challenge ideas / **assumptions**, be humble
- Use **data** whenever possible
- Build a culture where the **best idea** wins, not the most comfortable

Teams

that Deliver Value Faster

Book List:

Phoenix Project

Release It

Accelerate

SRE Handbook

Team Topologies

DORA Metrics

to measure DevOps success

1. **Frequency of deployments** (eg. every quarter ... every 2h)
   = **confidence** in automated tests (not just %coverage)
   = fast CI => lots of fast feedback to dev team => learn
2. **Lead time for changes**
   = how fast you ship a feature once the team **accepts** the task (in sprint backlog)
3. **How frequently deployments fail %**
   => automated tests smoke, performance, spike, chaos prior to deploy
4. **How fast can you recover from a failure**
   = MTTR => observability + disaster planning🧠

... but, **When a measure becomes a target, it ceases to be a good measure** - *Goodhart's law*

https://www.atlassian.com/devops/frameworks/dora-metrics

Example Developer Survey

Rate how often the following statements apply to you:

	Never	Rarely	Sometimes	Very often	Always	Unsure N/A
When I investigate production issues, it's easy for me to debug.						
For the product areas I work on, production issues and incidents are handled effectively.						
When I need a code review, I receive one in a timely fashion.						
It's easy for me to understand and modify the code that I work with.						
While developing, I can quickly verify that my changes work.						

you can't measure productivity, but you can measure anti-productivity.

With the developer experience index we're trying to measure the waste,
because if we get rid of that, we can go faster – versus telling people to go faster.

Boundary Types

- **Business** subparts of the problem (eg: shipping, invoicing , ...
  - We speak the same **Ubiquitous Language** within our **Bounded Context**
- **Humans** (teams)
  - Align **Dev** Team with **Business** Team (= **Conway's Law**)
- **Technical** (codebase)
  - A dev team should own a manageable piece of code: split the BBoM in Modules
- Your 1st attempt will likely fail □ Prepare to **iterate** 🔁

Context Maps
&
Politics

https://github.com/ddd-crew/context-mapping-quiz

Mutually Dependent

- Two software artifacts or systems in two bounded contexts need to be delivered together to be successful and work.
- There is often a close, reciprocal link between data and functions between the two systems.

Free

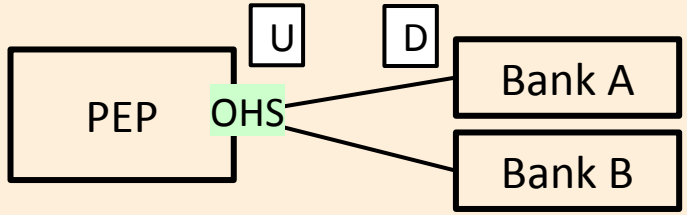
- Changes in one bounded context do not influence success or failure in other bounded contexts.
- There is, therefore, no organizational or technical link of any kind between the teams.

U D Upstream / Downstream

- An upstream context will influence the downstream counterpart while the opposite might not be true.
- This might apply to code but also on less technical factors such as schedule or responsiveness to external requests.

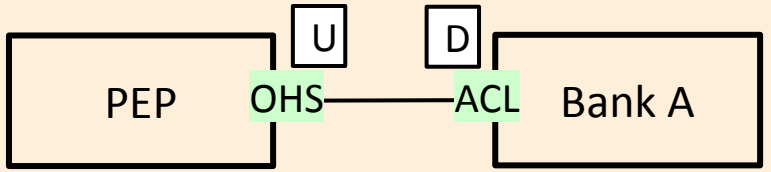
Open-Host Service

1 general purpose API for all consumers
No point-to-point contracts



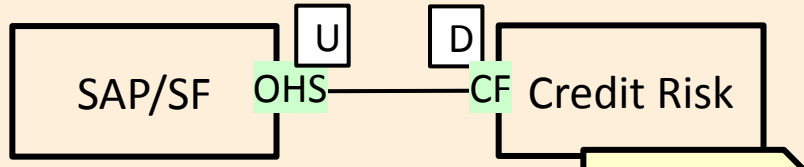
Anti-Corruption Layer

Translates an external ⇔ internal model
Restricts coupling to a single layer



Conformist

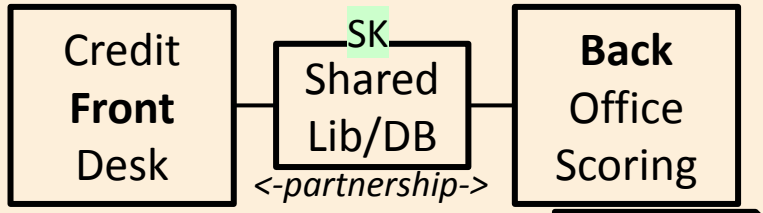
No model ⇔ model transformation
Slavishly adhere to upstream model, for simplicity, time, or delight



Quiz #1

Shared Kernel

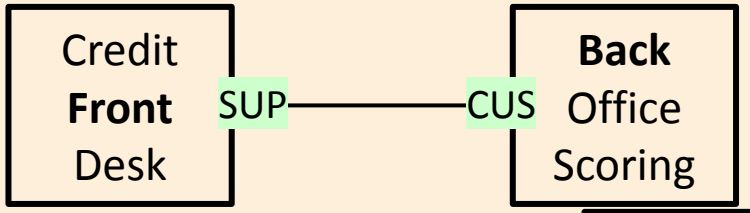
Two teams share a part of their domain
Coupling, requires coordination & coop



Quiz #2

Customer-Supplier

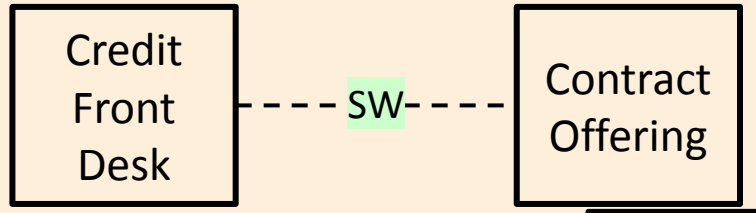
Downstream can influence Upstream
Care when upstream is shared (=OHS?)



Quiz #3

Separate Ways

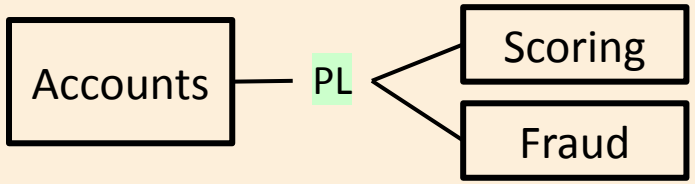
No connection between bounded contxt
Integration is too expensive/long || MVP



Quiz #3

Published Language

Well documented shared language, by a consortium: iCalendar, vCard, HL7, F8r...

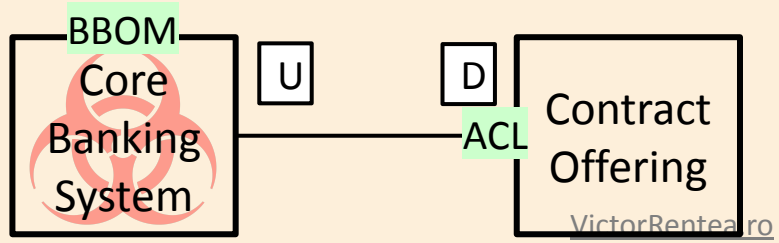


Context Maps
and
Team Politics

Introduction to Context Mapping by Michael Plod

Big Ball of Mud

A mess of mixed models w/o boundaries
Downstream must protects itself via ACL



Which comments apply best to this Context Map?

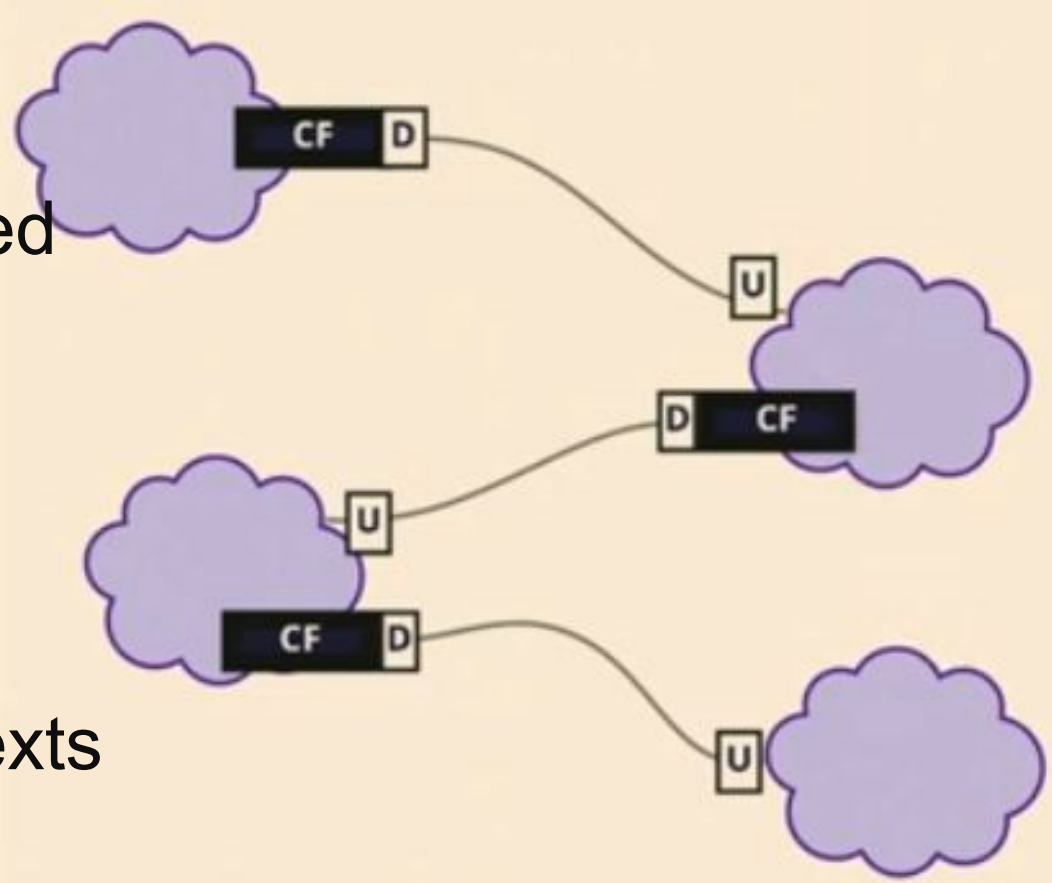
A) High degree of loose coupling

B) Lots of cross-team collaboration needed

C) This is a good scenario

D) Teams are Independent

E) Models propagate through many contexts



Quiz #1

https://github.com/ddd-crew/context-mapping-quiz

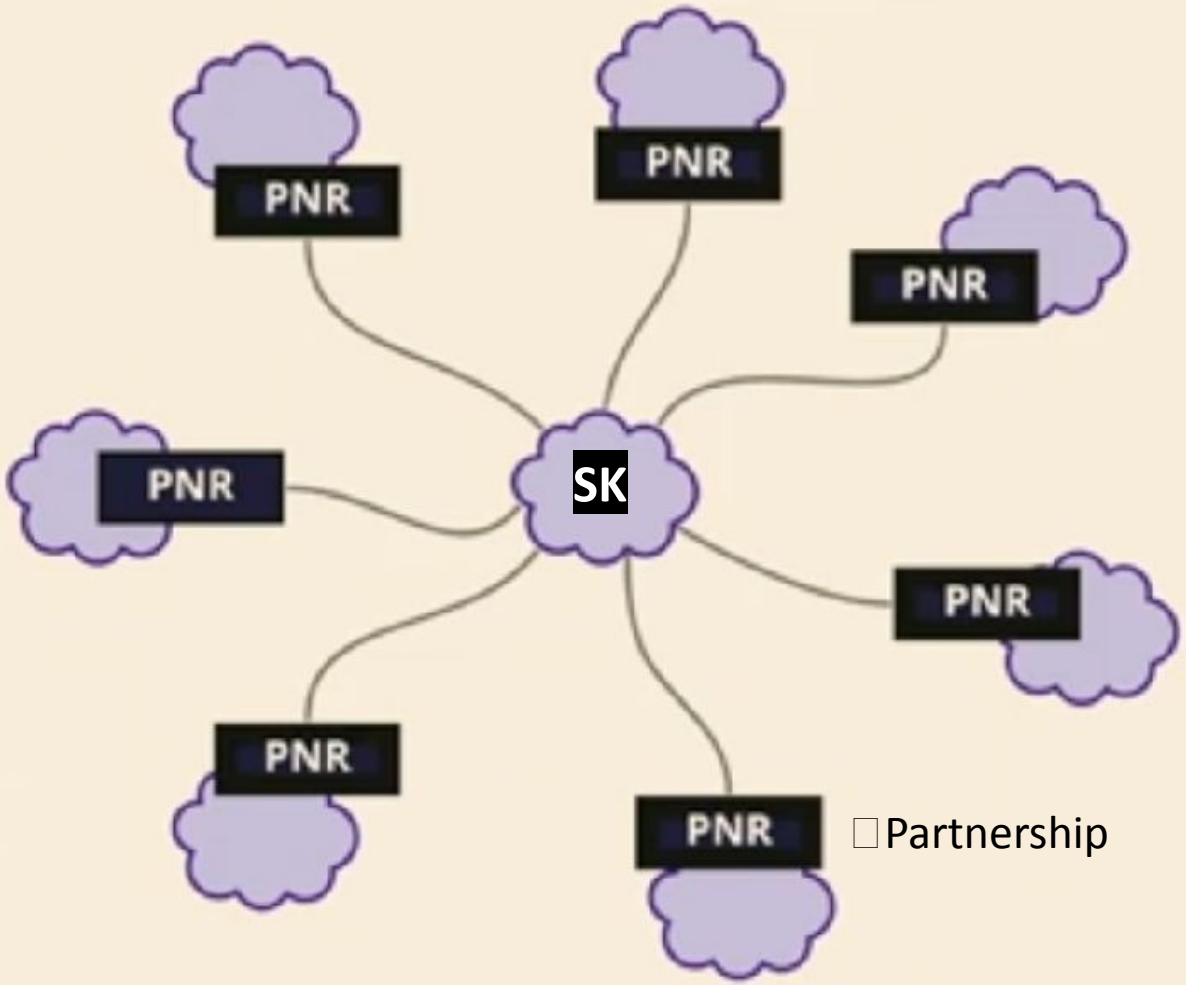
Which comments apply best regarding to the team in the middle

A) Is in a powerful position

B) Must attend many meetings

C) Gets a lot of own work done

D) Can's drive their own agenda



Quiz #2

https://github.com/ddd-crew/context-mapping-quiz VictorRentea.ro

Which statements are valid for the supplier?

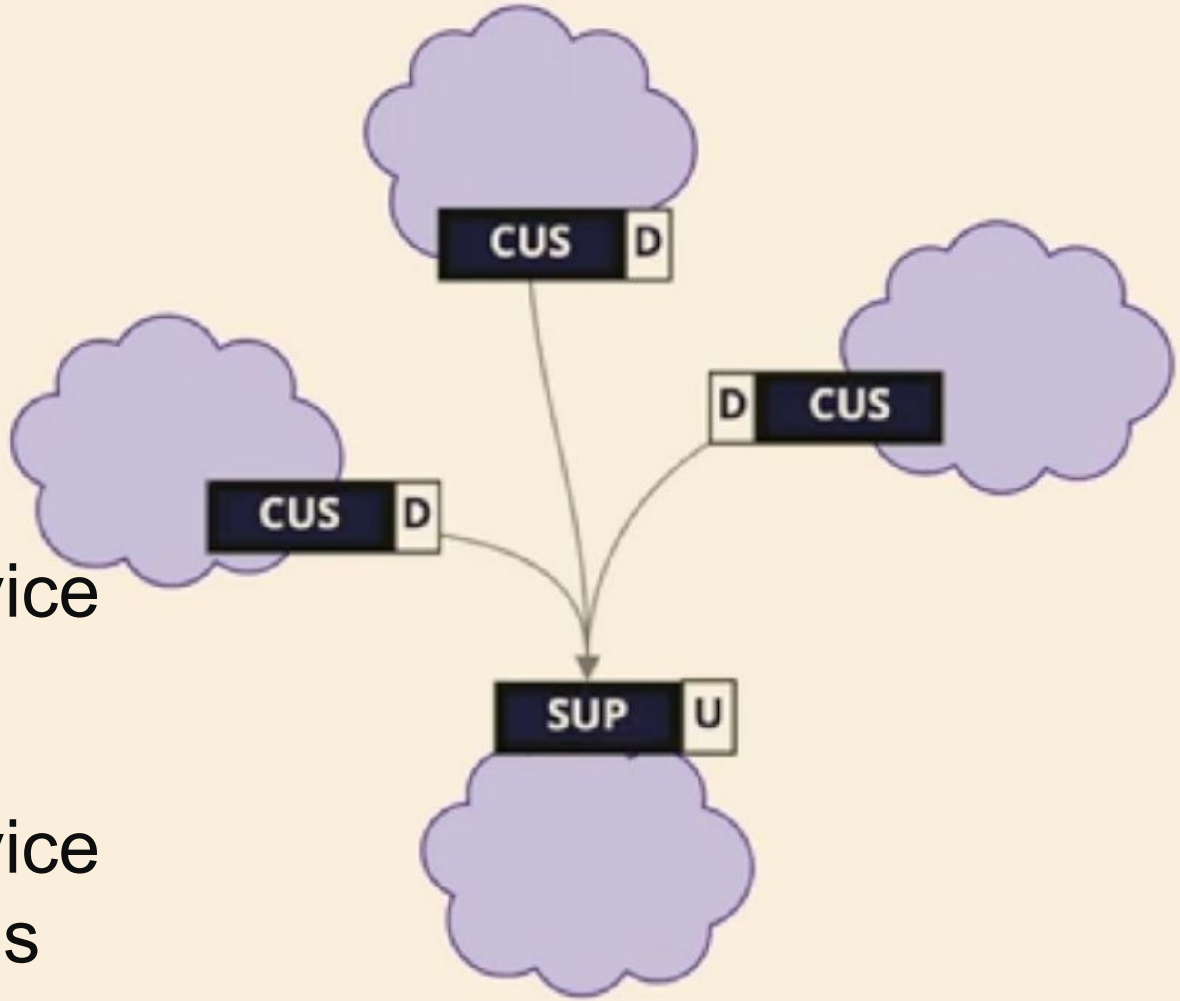
A) Is driven by customers

B) Will spend a lot of effort in prioritization of requirements

C) Has a lot of influence

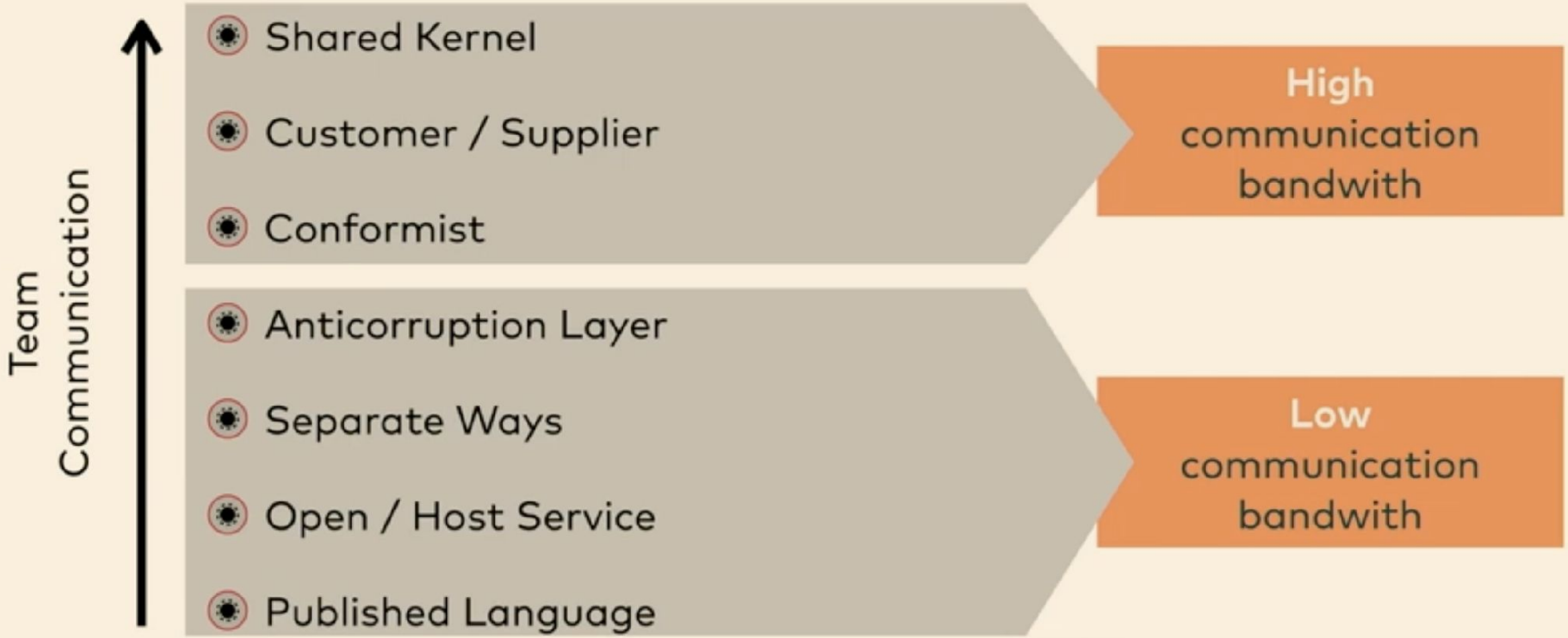
D) Should introduce an Open-host Service in addition to the CUS-SUP …

E) Should introduce an Open-host Service instead of the CUS-SUP relationships

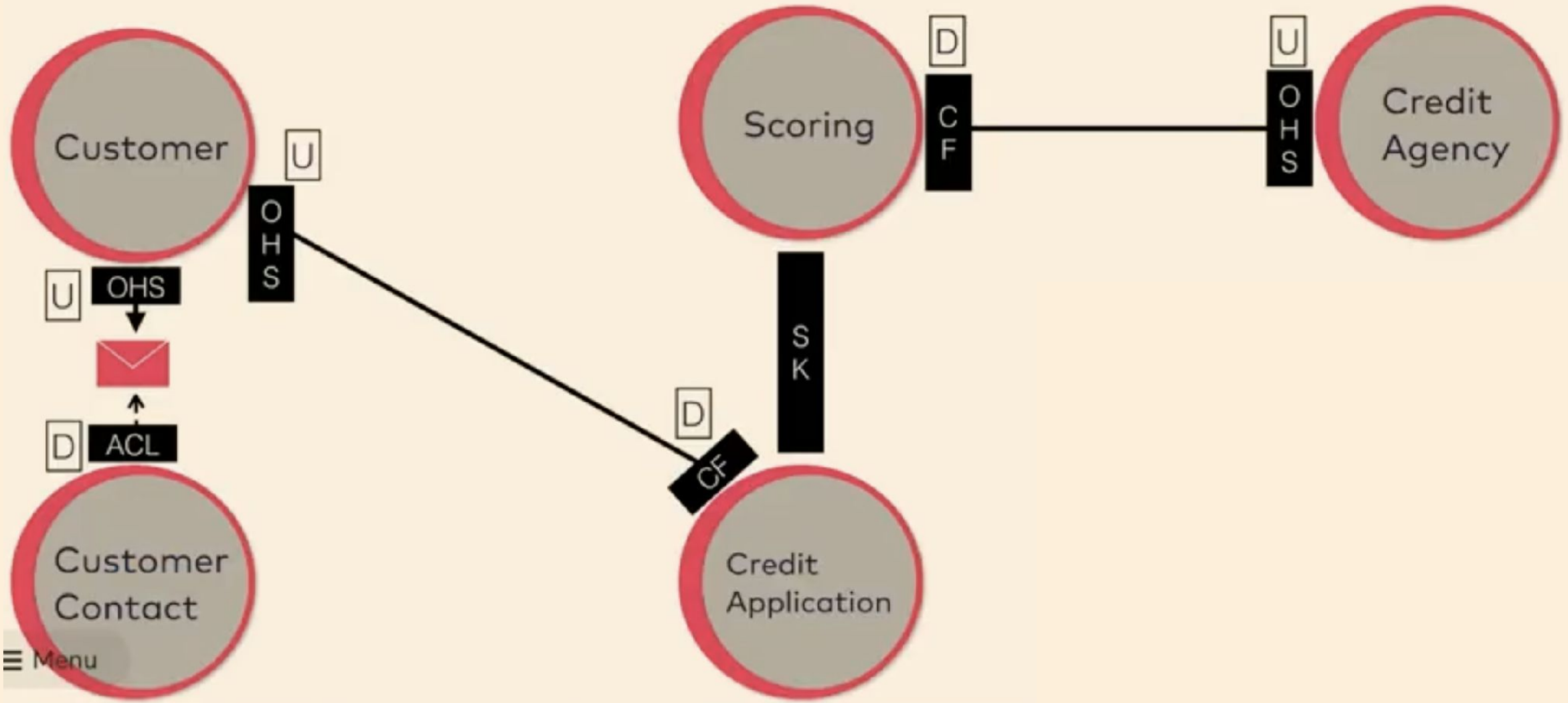


Quiz #3

https://github.com/ddd-crew/context-mapping-quiz


Team Communication
Shared Kernel
Customer / Supplier
Conformist
High communication bandwith
Anticorruption Layer
Separate Ways
Open / Host Service
Published Language
Low communication bandwith

Opinions ?


Customer
U
OHS
U
OHS
D
ACL
Customer Contact
Scoring
D
CF
U
OHS
Credit Agency
SK
D
CF
Credit Application
Menu

Subdomains

⚠️ Bright developers are notoriously attracted to technology ("marketable skills"/resume-building), rather than domain concepts

Core Subdomain 💎

= Differentiator vs competitors

eg: shipping in lockers

- Dev **in-house** by **talented mature problem solvers**
- Excellent working conditions (eg 1 free day/sprint)

Supporting Subdomain

= Vital for core subdomain

eg: invoicing

- Dev in-house / near-shore development
- Buy product + minor customizations

Generic Subdomain 🤭

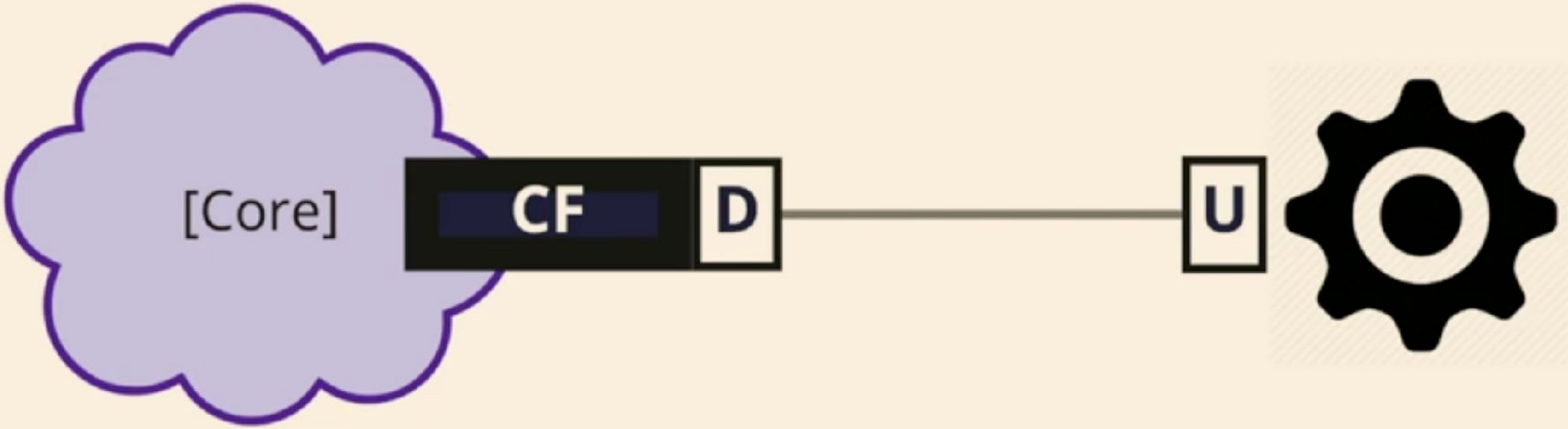
= "We need some solution to problem X"

eg: OAuth AS (KeyCloak)

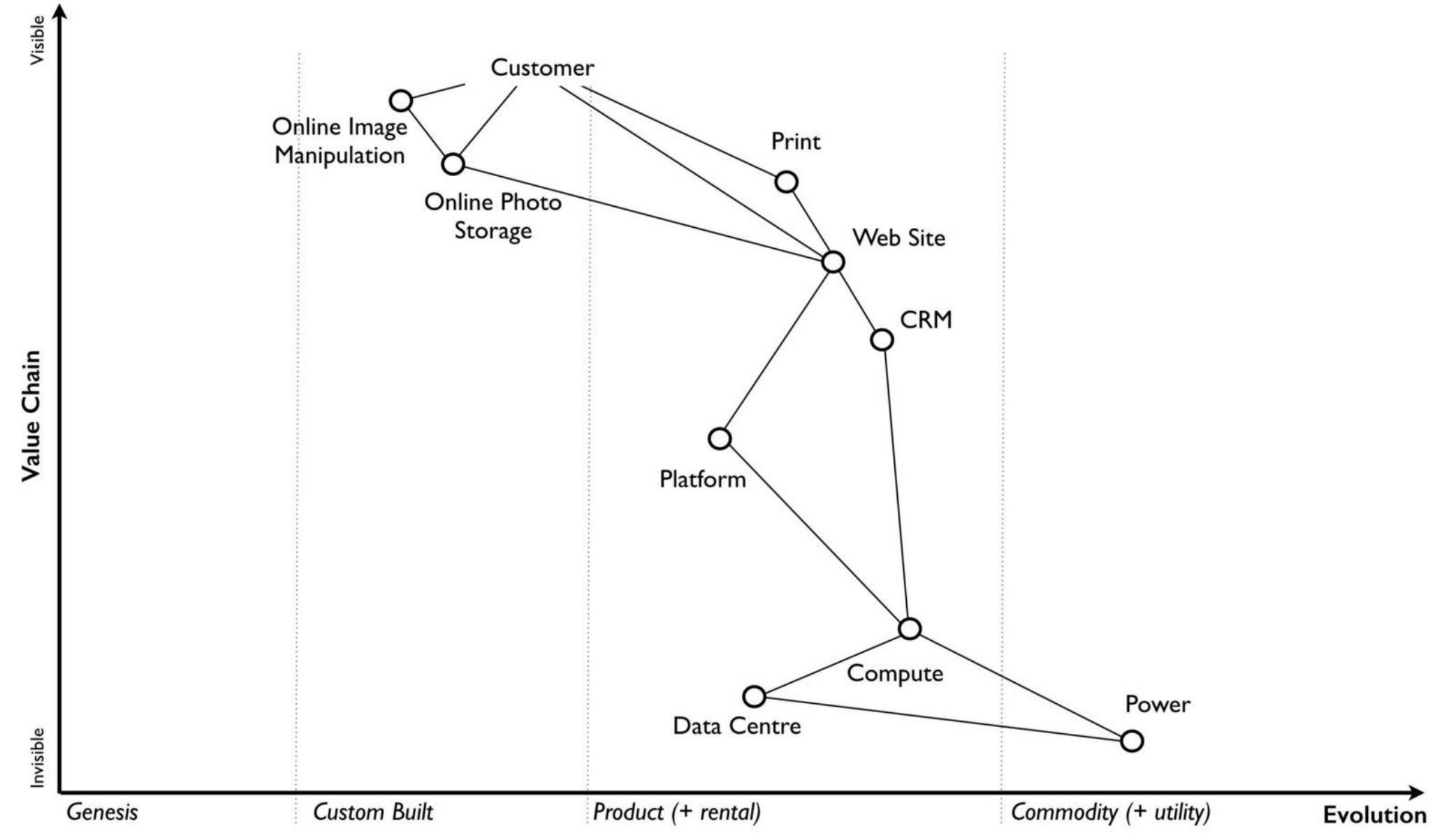
- Buy a product Off-the-shelf / SaaS products
- Cost Saving via "outsourcing"

from DDD book by Eric Evans

What are your thoughts on a core domain conforming to an external system?



Wardley Map


Visible
Invisible
Value Chain
Customer
Online Image Manipulation
Online Photo Storage
Print
Web Site
CRM
Platform
Compute
Data Centre
Power
Genesis
Custom Built
Product (+ rental)
Commodity (+ utility)
Evolution

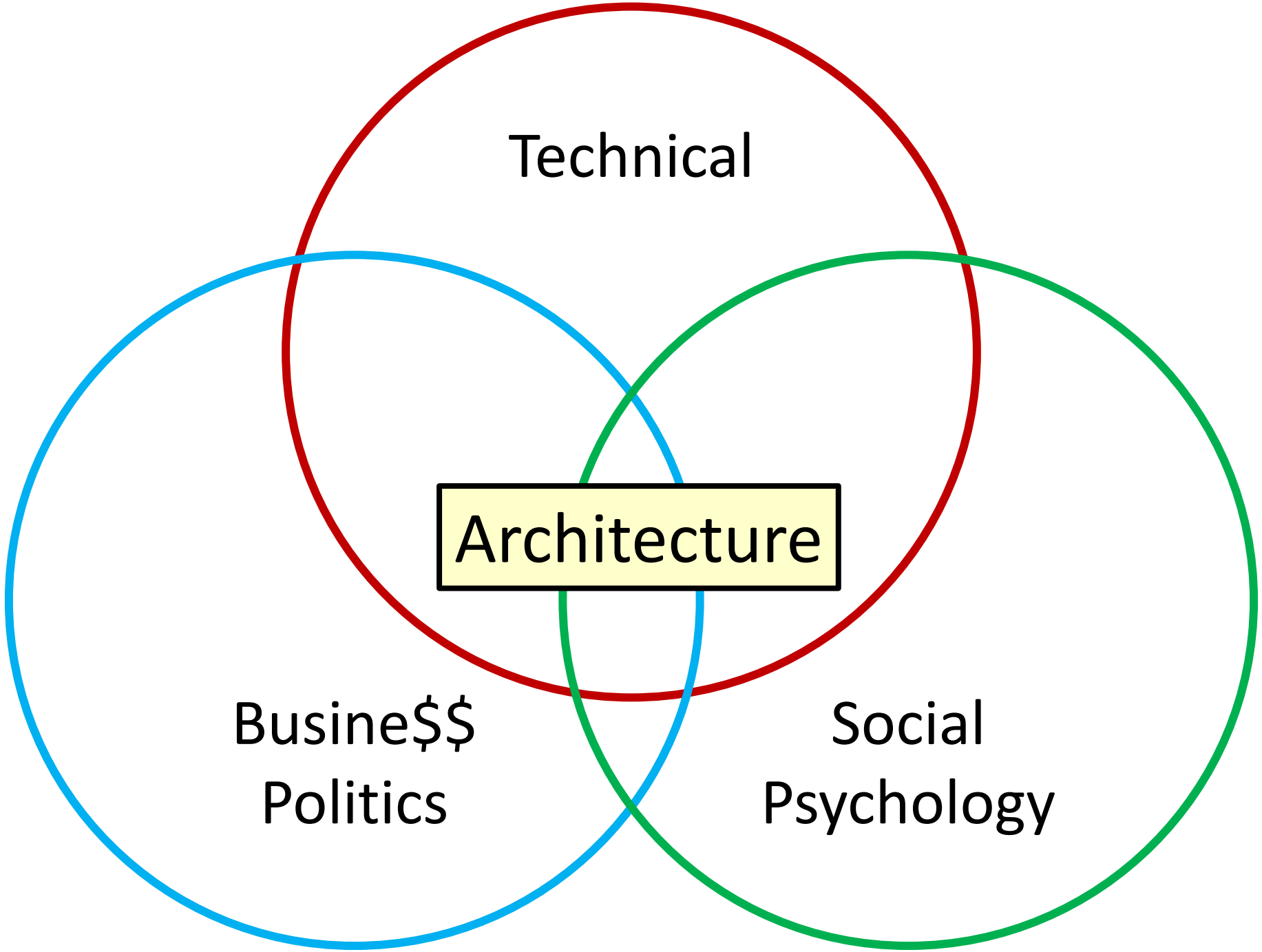
Rules can Hurt Creativity

over-justification

architecturally interested developers 😍

Role of an Architect

https://event-driven.io/en/architect_manifesto


Technical
Architecture
Busine$$
Politics
Social
Psychology

Architect: Support, don't Control

- A **bad architect** wants to **have control** ("control freak")
  - *I'm an archtect, to save time, let's just assume I'm always right.*
  - Tells you what to do, no comments;
  - Imposes cold rules without explanations (or explained in a 20 pages wiki 😵‍💫)
  - Developers don't understand, won't follow => Design disaster ⚠️
  - Architect can become an "approval bottleneck" ⚠️
- A **good architect** **helps you** take the right decisions
  - Explains design options, with pros/cons (from his extensive experience)
  - Does on-demand **Code Reviews** & **Pair Programming**
  - *A good leader makes him/herself useless*

The Coding Guideline Document

- Trying to maintain a homogenous code
- Some in the team know of this document, but no one has ever read it
- It has 100 pages and it was written 10 years ago by people that left

- Automate as many checks you can
- Use & challenge this document to explain/review work of new joiner
- Craft a quiz at the end + some exercises.
- Keep it short! Don't blindly copy from a standard. Express YOUR understanding.
- Every year, burn it and ask each team member to bring 5-6 items to rewrite it

Dogmatic vs Pragmatic

More Rules = easy to obey

✅ **Fast onboarding** (eg mass-hire😕)

✅ **Less decisions**

✅ Easy to **enforce** by lead|architect

✅ Suitable for **mid-junior** teams

Less Rules = needs creativity

✅ **More design** debates/refactoring

✅ **Simplest solution**

✅ **Easier to unit test**

✅ Suitable for **mid-senior** teams

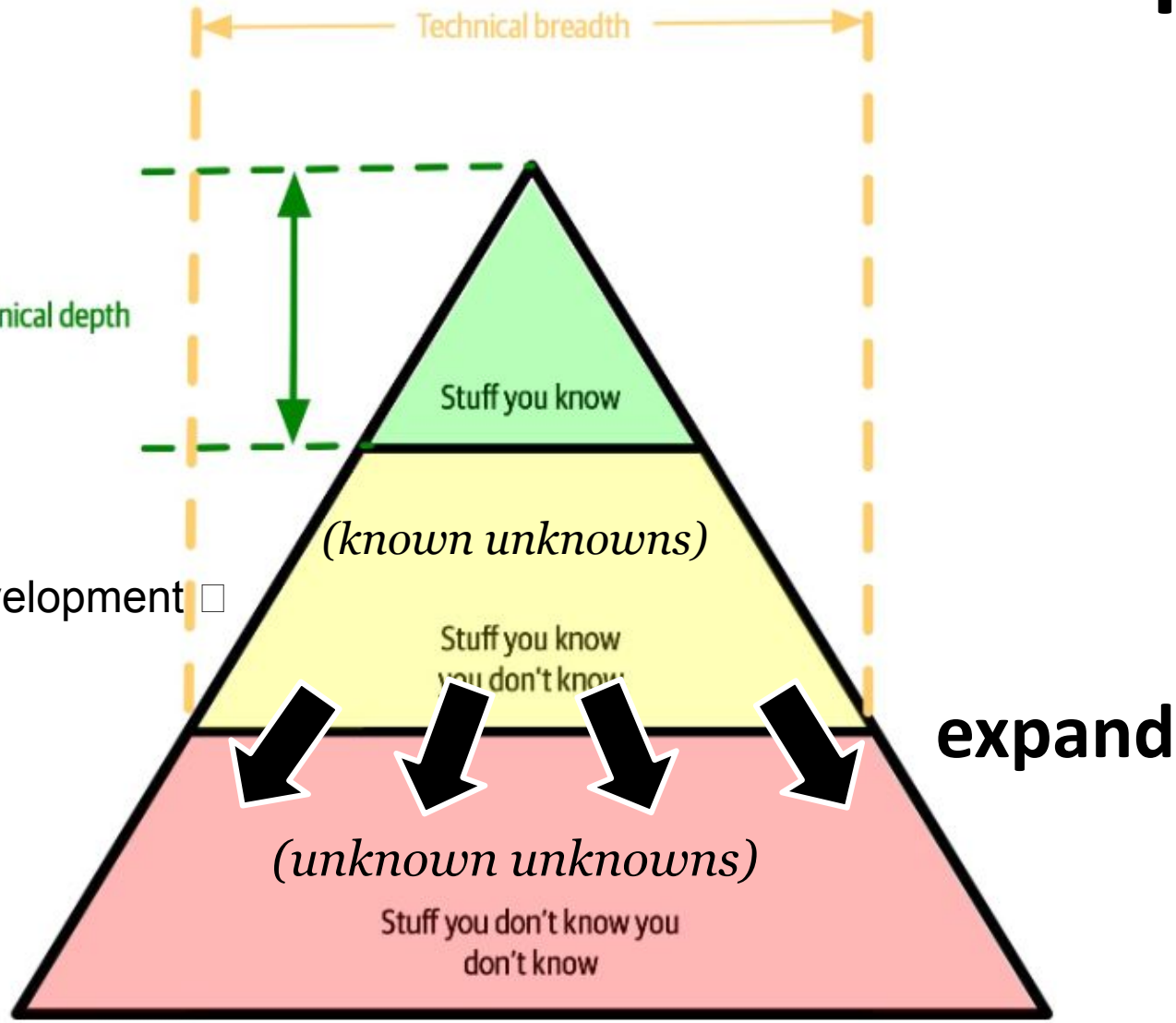
Expectations of an Architect

- **Make Architecture Decisions**: guide not dictate technology
- **Continually Review the Architecture**: is it still valid after 3 years?
- **Ensure Compliance with Decisions**, Blueprints, Design Principles 😡#$!@
- **Keep Current with Latest Trends** to take long-term decisions
- **Diverse Exposure and Experience:** favor breadth over depth
- **Have Business Domain Knowledge**: talk busine$$/user language
- **Exceptional Interpersonal Skills**: leadership, facilitation, teamwork
- **Understand and Navigate Politics** to negotiate, prioritize

Architect vs ...

- **Engineering Practices**
  - Architects should **code 1-3 iterations** to experience their decisions, and other **devs' pains**: procedures, processes, dev environment
  - Arc to write quality POC, refactor tech debt, bugfix and code-review
  - Developers should learn to build *architectural fitness functions*
- **Ops**=>Dev+Ops: avoid the overengineering due of an Ops silo
  - CD made possible via Strangler Pattern and Feature Toggles
- **Process>Agile:** recognizes the importance of unknown unkowns
  - Shorter feedback loop > more experiments with architecture decisions
- **Data**: consistency, volumes, numerous columns

Favor Technical Breadth over Depth



Zone of Proximal Development

expand

developertoarchitect.com/lessons

Species of Architects

- **Code Architects,** Software-, System-
  - Coding daily/weekly, willing to pair/review/refactor your code
  - Usually seniors/tech leads/coaches
  - "Every developer is an architect"
- **Solution Architects**
  - Mixing products (AWS, Kafka, boxes and lines), eats RFP
- **Functional** ("Enterprise") **Architects**
  - Super-smart **Business guy** with a bigger overview

Changing Mindset

- How to fit all this in my head? How to fight my bad[?] habits?
  - **Practice**: redo all the refactoring steps I did! (following the Git history)
  - **Play** on your project (Trick: delete all tests first). Undo after.
- Fight the inertia of mindset & legacy code
  - Identify common pain points
  - Choose your battles!!
  - Start with the smallest change that makes an observable difference
- Often you need a "Kindred Spirit" (a friend) to support you
- Mind consistency of existing codebases!

Changing *Their* Mindset

- Not ethical.
- Instead, make them see where they waste time/money...

- Sell me this pen!
- I'm gonna tell you my 12 card number. Here;s a piece of paper. Do you need a pen?