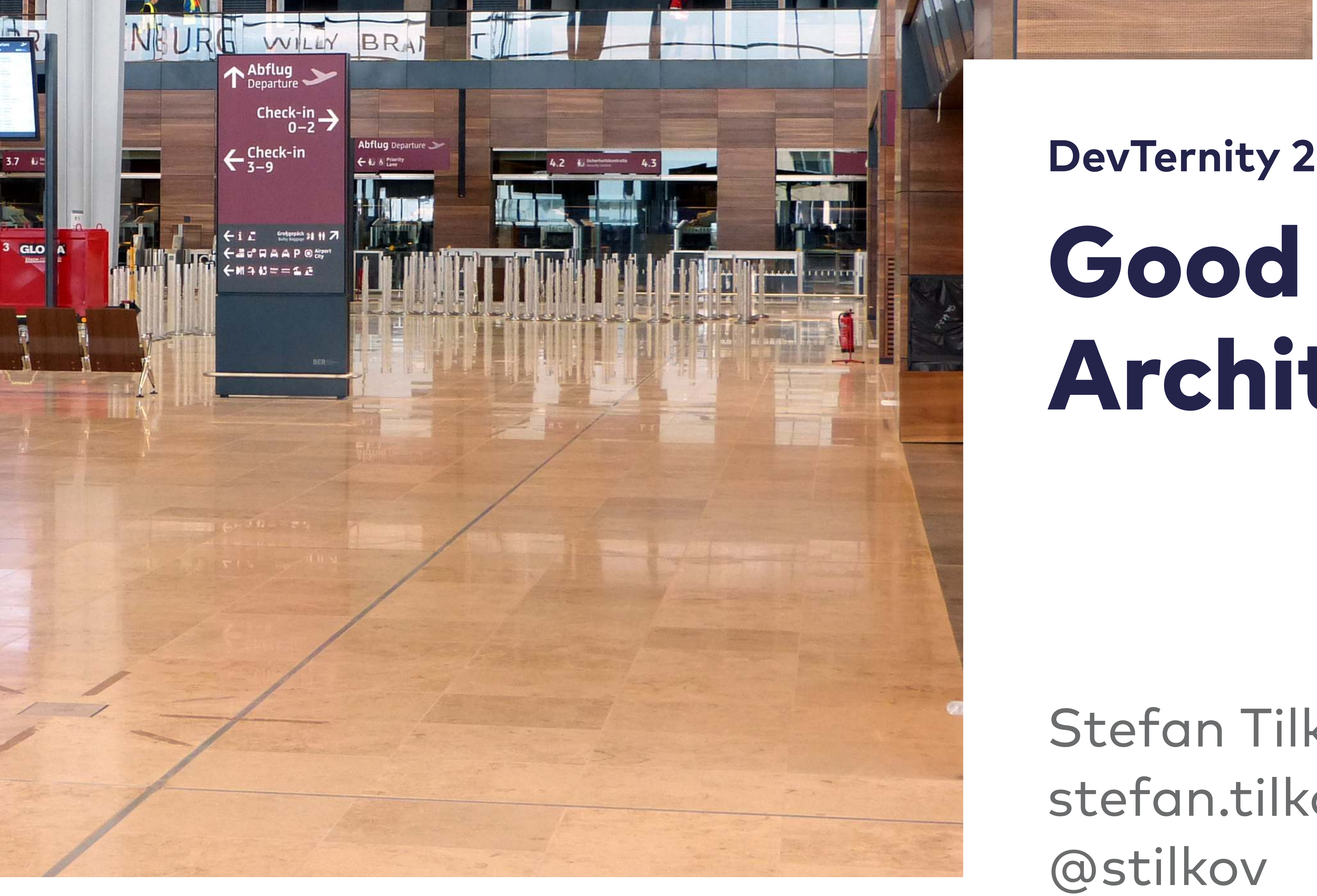


DevTernity 2021

Good Enough Architecture

Stefan Tilkov
stefan.tilkov@innoq.com
@stilkov

INNOQ https://www.innoq.com/

Flughafen Berlin-Brandenburg, Blick in die Abflughalle; https://en.wikipedia.org/wiki/Berlin_Brandenburg_Airport#/media/File:120513-BER-innen.JPG; CC BY 3.0

DevTernity 2021

Architectures you never wanted to know about

Stefan Tilkov
stefan.tilkov@innoq.com
@stilkov

INNOQ https://www.innoq.com/

Flughafen Berlin-Brandenburg, Blick in die Abflughalle; https://en.wikipedia.org/wiki/Berlin_Brandenburg_Airport#/media/File:120513-BER-innen.JPG; CC BY 3.0

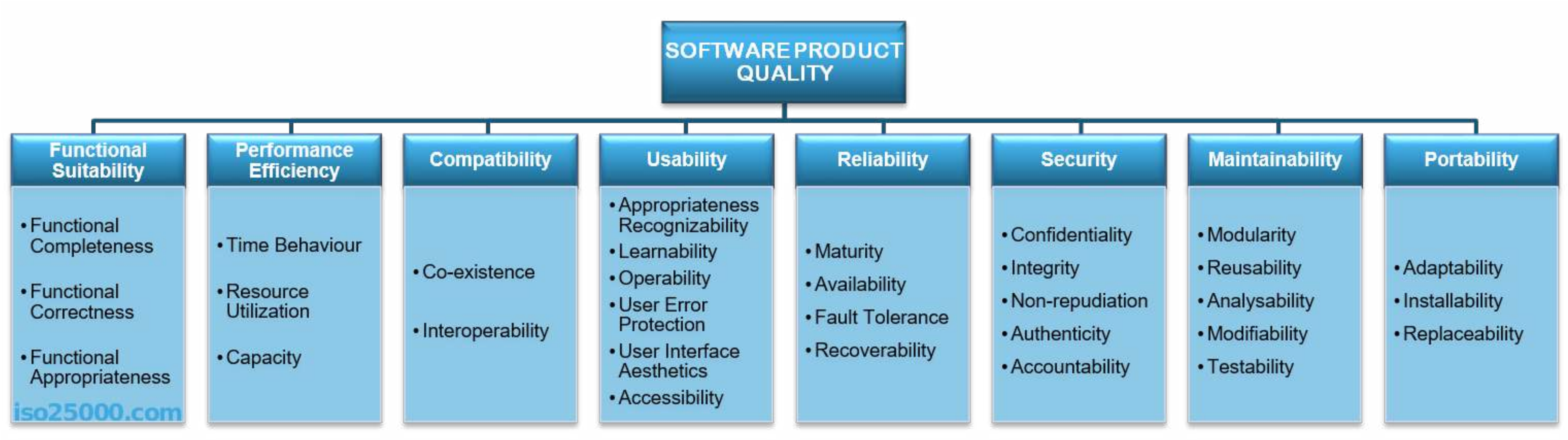
Some generic truths about software architecture

Architecture is not an upfront activity performed by somebody in charge of telling everyone else what to do

Architecture is a property of a system, not a description of its intended design

Pick the best car:

Quality



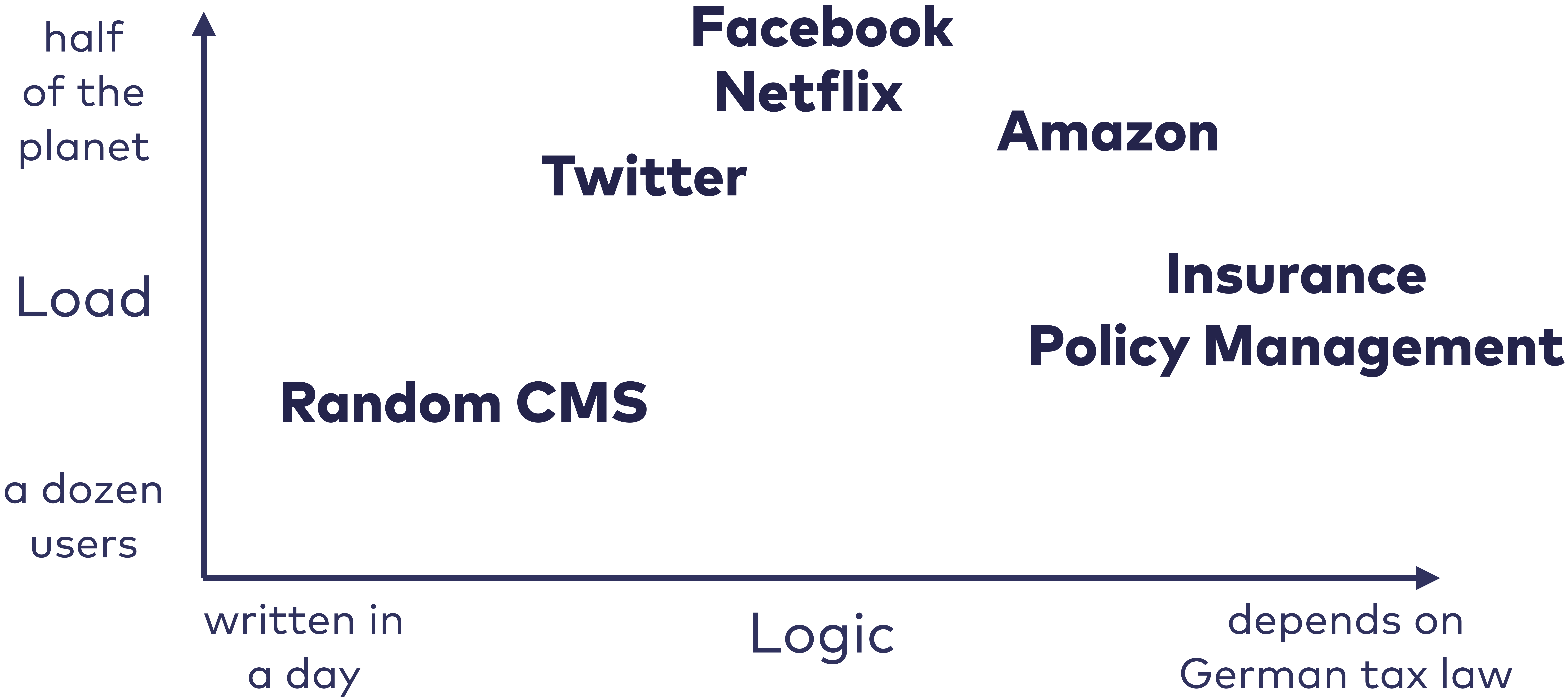
SOFTWARE PRODUCT QUALITY

- Functional Suitability
  - Functional Completeness
  - Functional Correctness
  - Functional Appropriateness
- Performance Efficiency
  - Time Behaviour
  - Resource Utilization
  - Capacity
- Compatibility
  - Co-existence
  - Interoperability
- Usability
  - Appropriateness Recognizability
  - Learnability
  - Operability
  - User Error Protection
  - User Interface Aesthetics
  - Accessibility
- Reliability
  - Maturity
  - Availability
  - Fault Tolerance
  - Recoverability
- Security
  - Confidentiality
  - Integrity
  - Non-repudiation
  - Authenticity
  - Accountability
- Maintainability
  - Modularity
  - Reusability
  - Analysability
  - Modifiability
  - Testability
- Portability
  - Adaptability
  - Installability
  - Replaceability

iso25000.com

https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Scaling Dimensions



half of the planet

Load

a dozen users

written in a day

Logic

depends on German tax law

Facebook

Netflix

Amazon

Twitter

Insurance Policy Management

Random CMS

There is no “good” or “bad” architecture without context; architecture needs to take specific quality attributes into account

Cases

Context:

- …

Observation(s):

- …

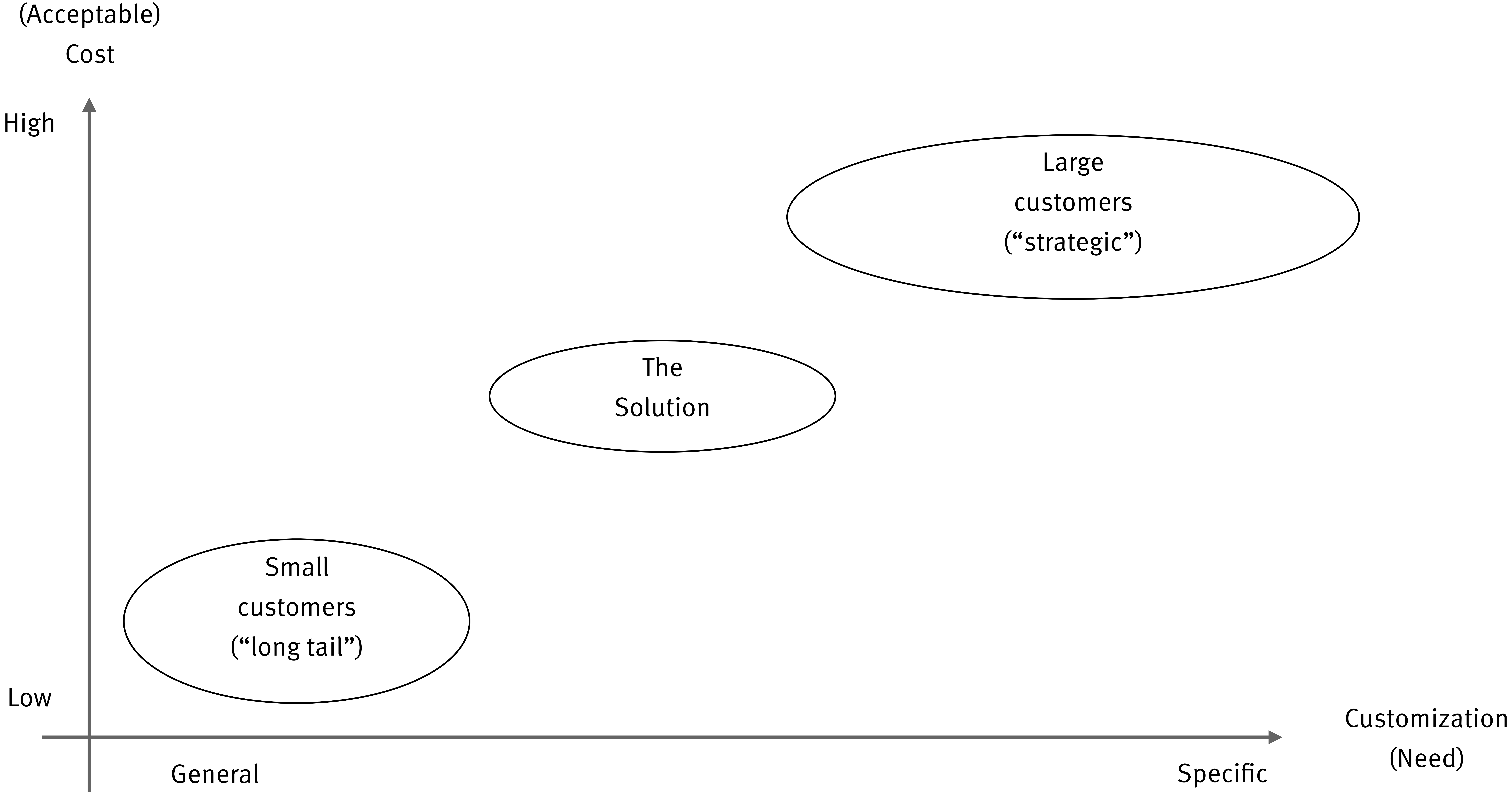
Lesson(s) learned:

- …

#1: Non-extensible Extensibility

Context

- E-Commerce (retail) provider
- Global customer base
- Catalog/CMS/Shop/Fulfillment
- Multi-tenant
- Highly customizable


(Acceptable)
Cost
High
Low
Large customers ("strategic")
The Solution
Small customers ("long tail")
General
Specific
Customization (Need)

@stilkov

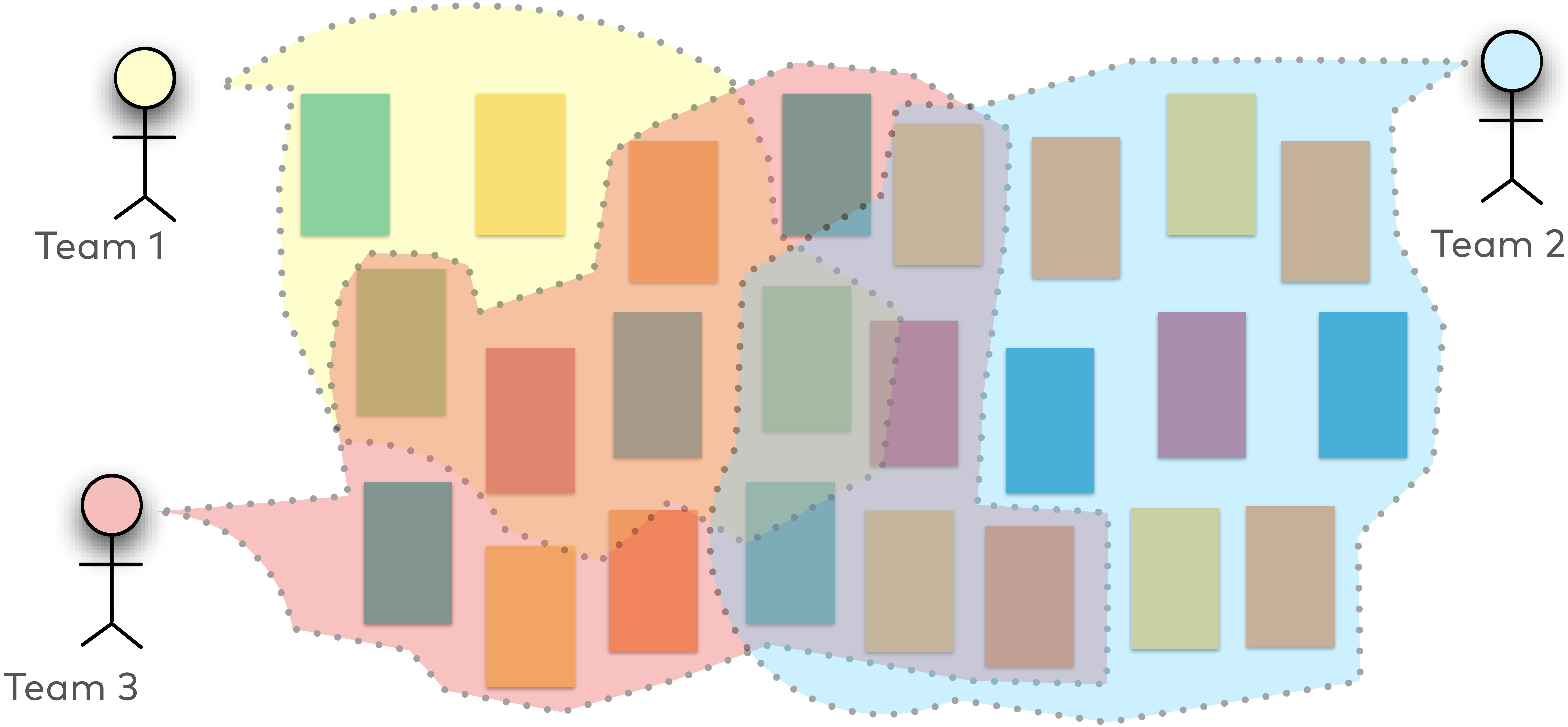
If your design attempts to satisfy everyone, you'll likely end up satisfying no one

Highly specific code is often preferable to sophisticated configuration

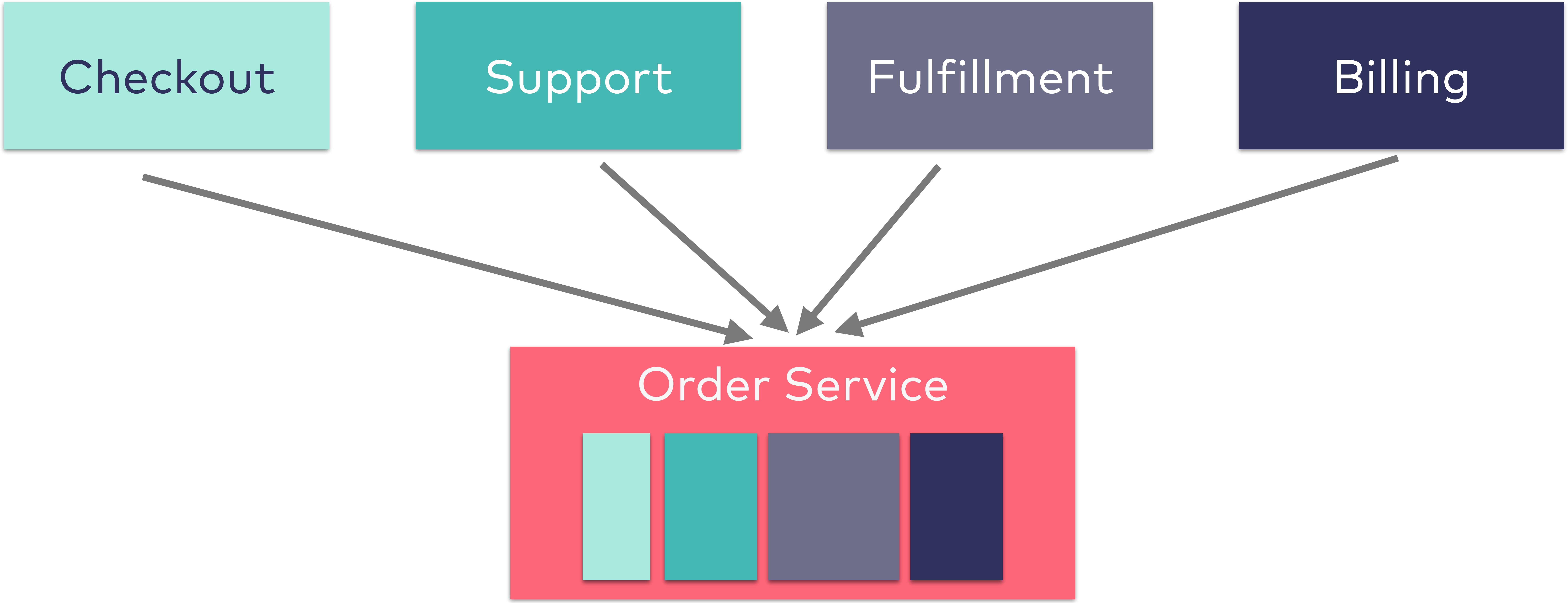
#2: Perilously Fine Granularity

Context

- Large-scale B2B food retailer
- New company-wide shop and logistics system
- >200 developers



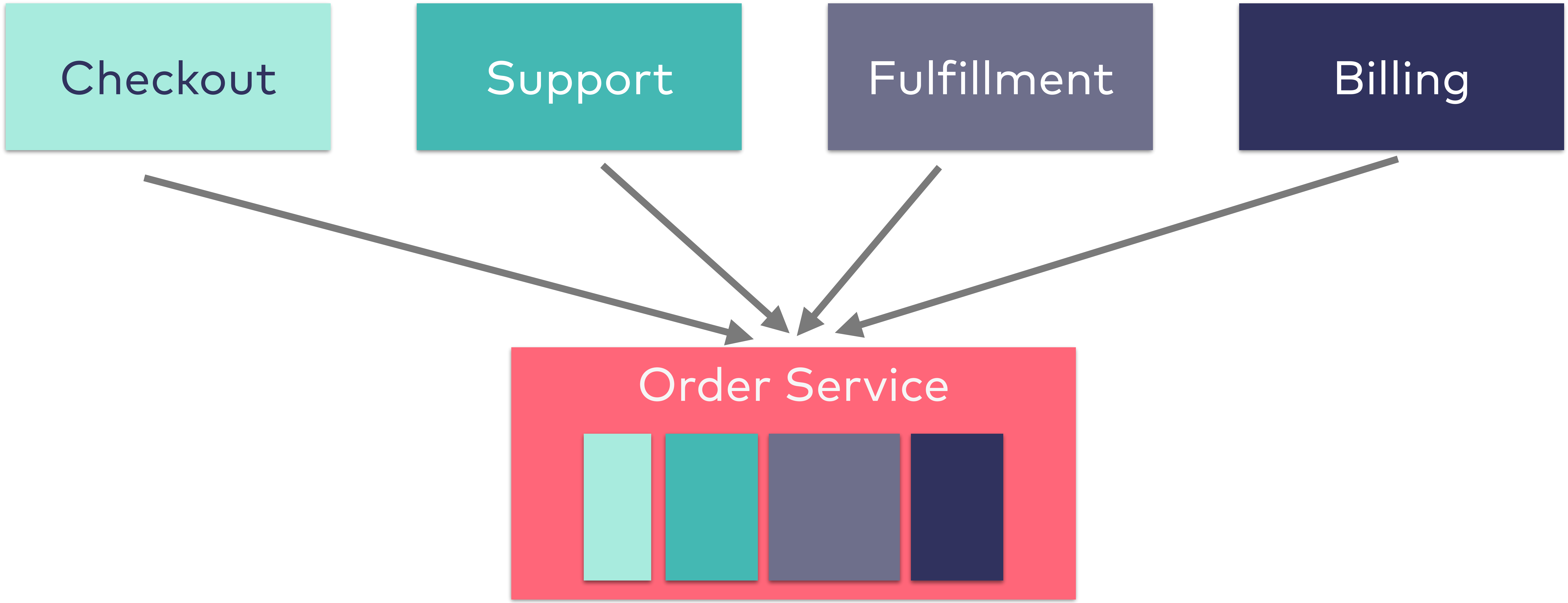
@stilkov



@stilkov

Why would you cut up your system into tiny, distributed, hard-to-manage fragments?

Everybody wants to be Netflix, but nobody is



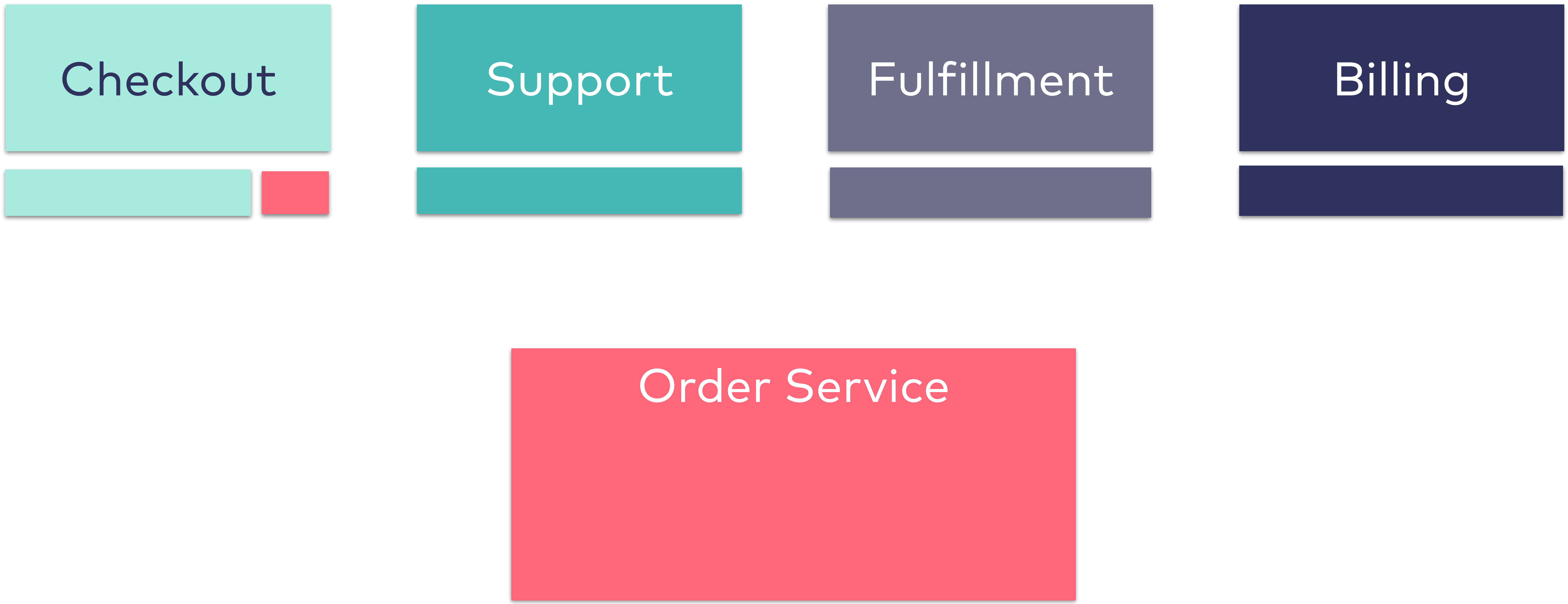
Checkout

Support

Fulfillment

Billing

Order Service



Checkout

Support

Fulfillment

Billing

Order Service

Small is not always beautiful

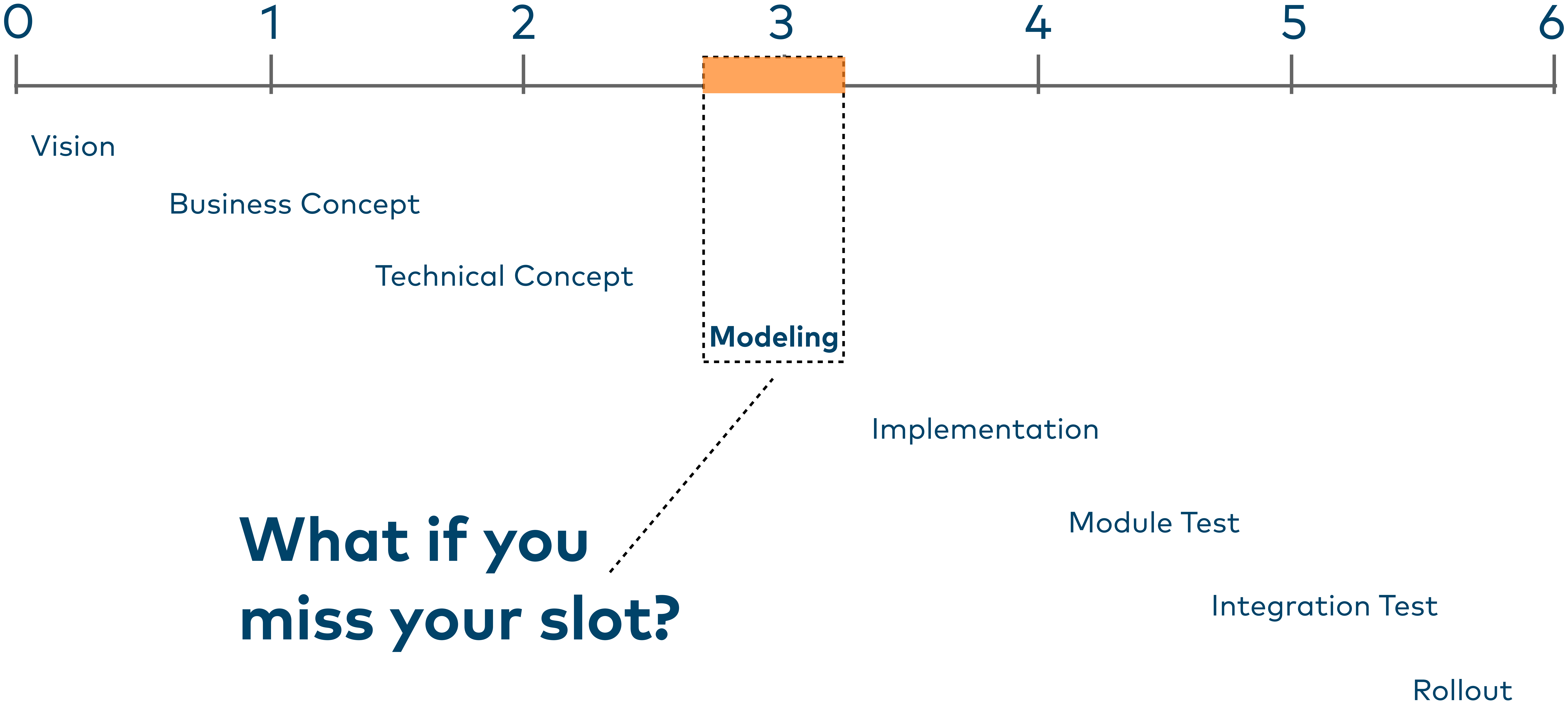
Refactoring within team boundaries much easier than globally

Ignore organizational parameters at your own risk

#3: Your system WILL be dynamic

Context

- Large-scale insurance system
- Model-driven development
- > 100 developers
- 2 Releases/year



Vision

Business Concept

Technical Concept

Modeling

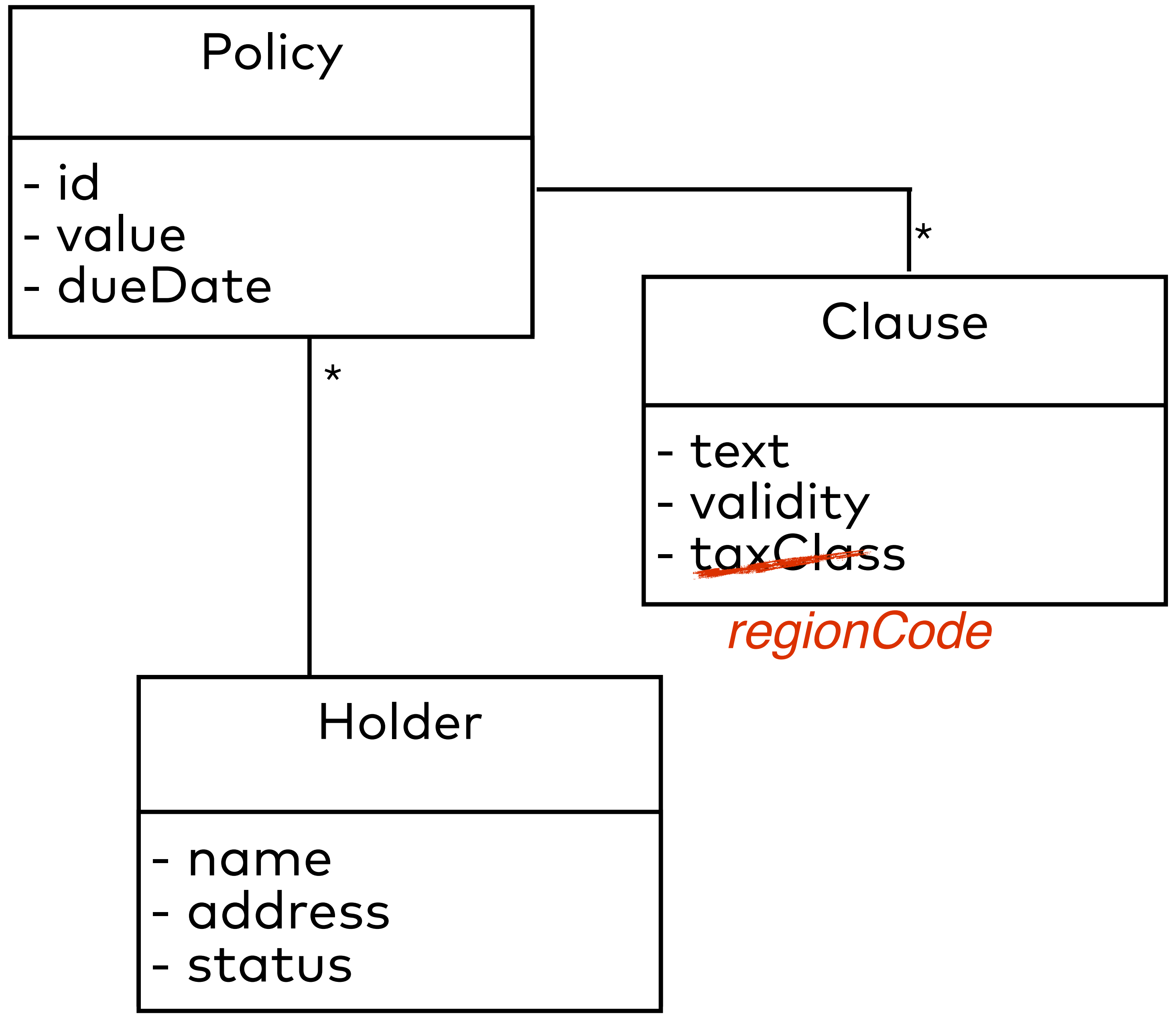
Implementation

Module Test

Integration Test

Rollout

What if you miss your slot?



Policy

- id
- value
- dueDate

*

*

Clause

- text
- validity
- ~~taxClass~~

regionCode

Holder

- name
- address
- status

Model Name	New Name (Meaning)	Description	Release Introduced
taxClass	regionCode	…	10.3
…			

Centralized responsibility hurts and creates bottlenecks

Faced with too much rigidity, developers will find a way around the rules

Just because you're used to it doesn't mean it's acceptable

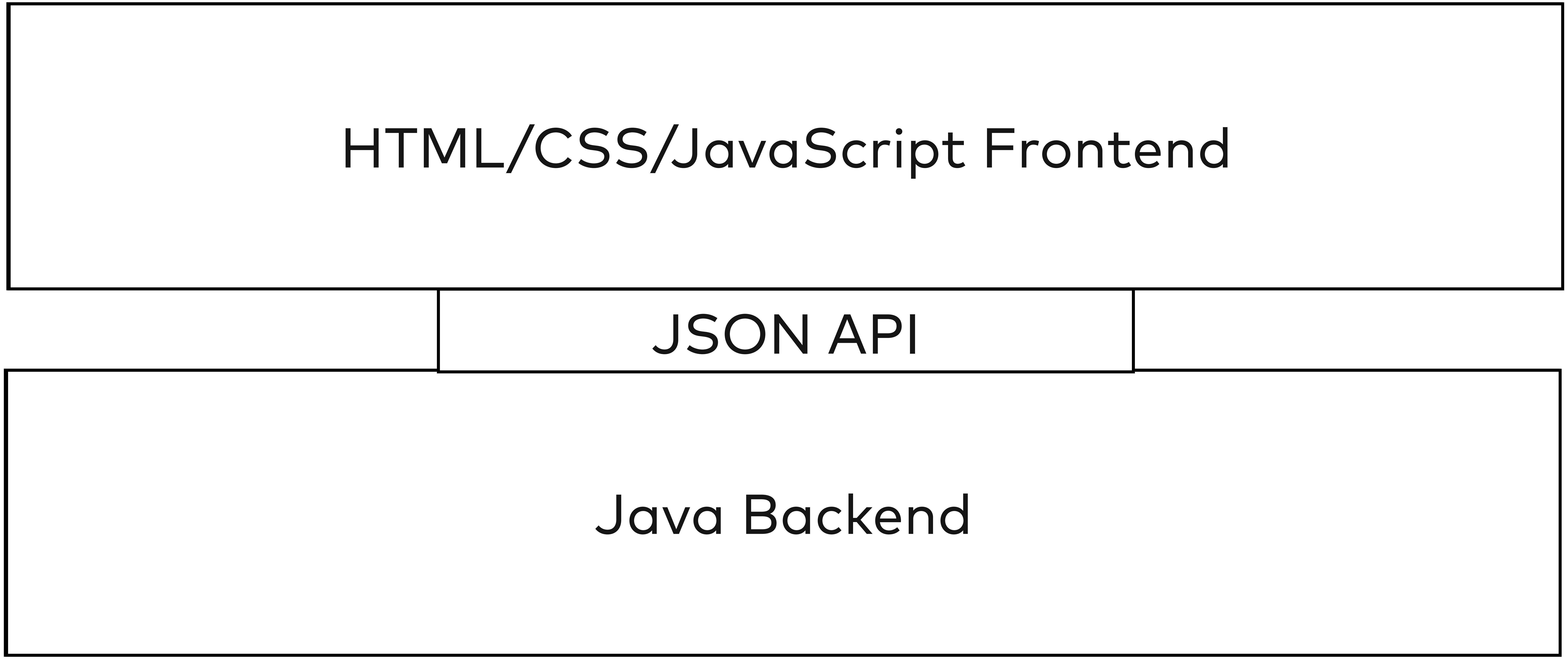
#4: Horizontal Conway Split

Context

- Unified communication platform
- Straight-forward business logic
- High scaling requirement
- 1-2 teams/10-20 developers

Group 1
Motto: »Java is a legacy programming language used in the last century«

Group 2
Motto: »Obviously, you can’t build correct programs with JavaScript«



HTML/CSS/JavaScript Frontend

JSON API

Java Backend

When faced with unresponsive backend teams, frontend teams quickly become full-stack teams

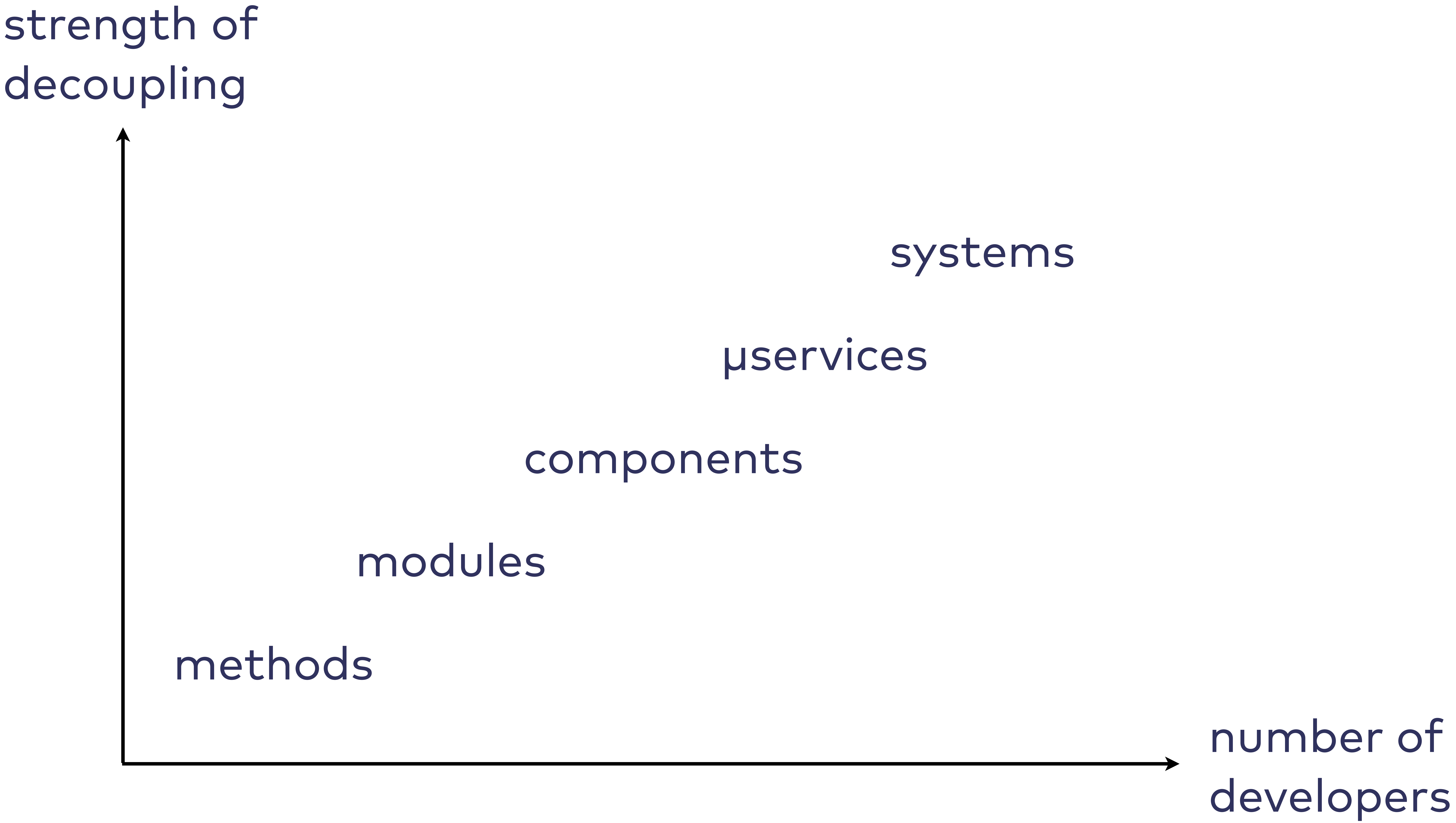
Every meaningful feature development requires frontend and backend work

There are no limits to architectural complexity people will accept to stay among their own

#5: System of Systems

Context

- E-Commerce/Online shop (Retail)
- 100-120 developers
- ~10 self-contained teams



strength of decoupling

systems

µservices

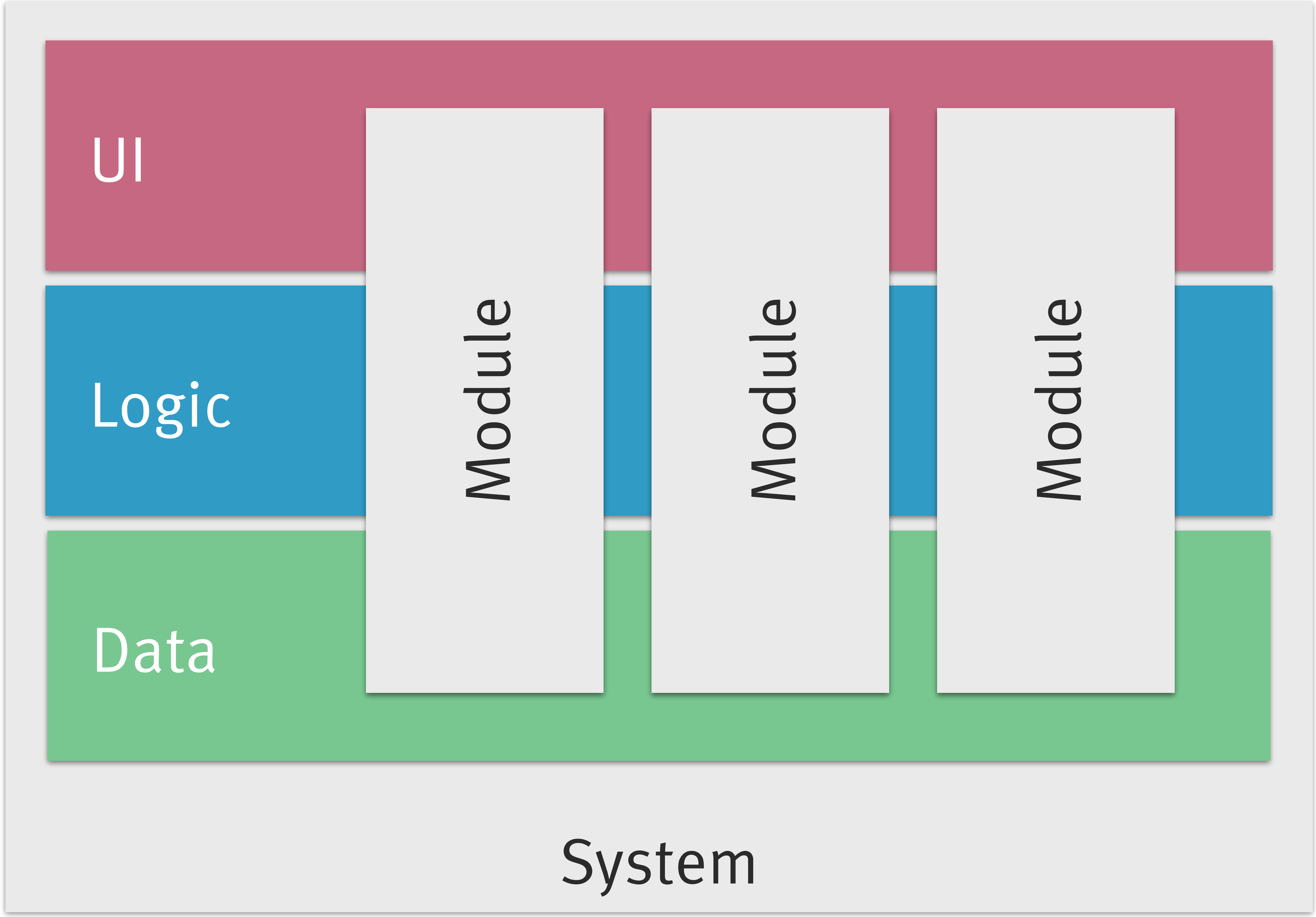
components

modules

methods

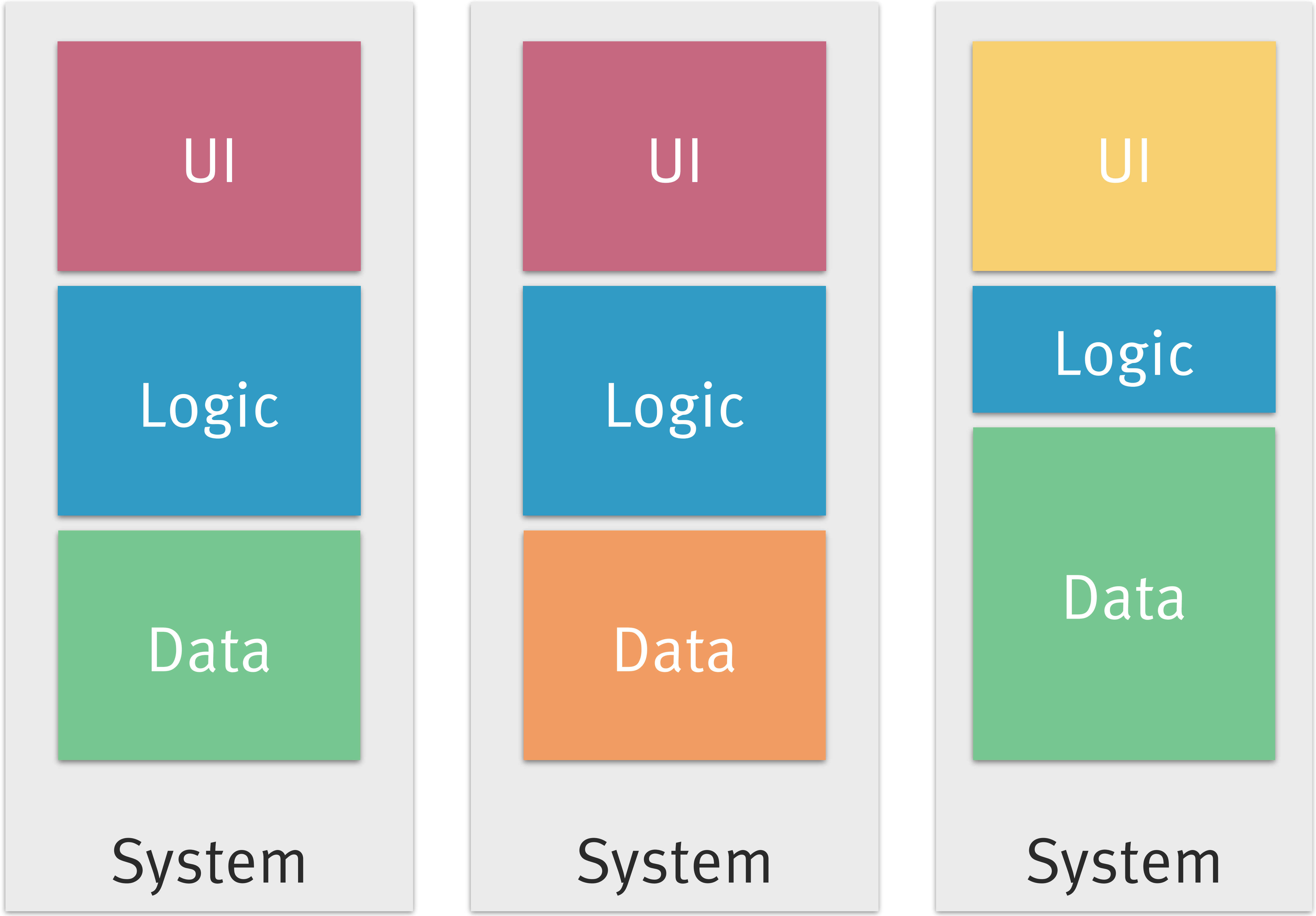
number of developers

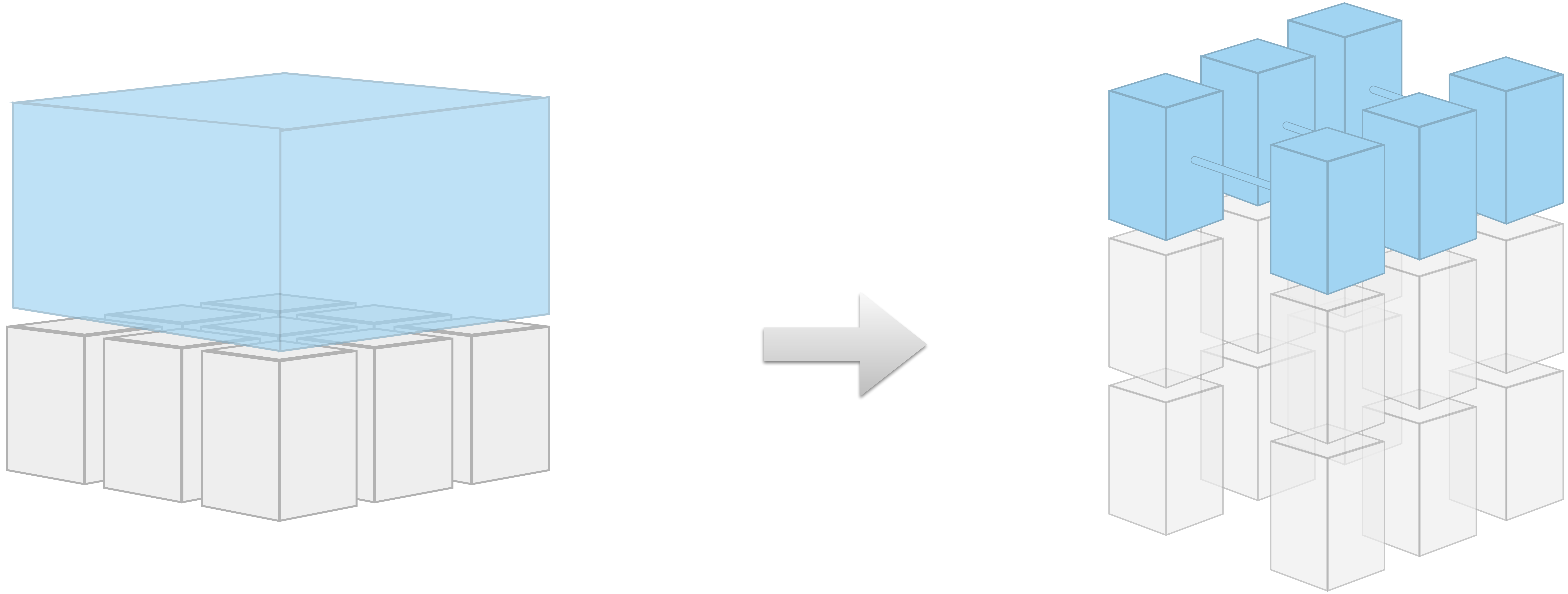
From a layered system …



@stilkov

… to a system of systems





In-page

JavaScript method calls

Shared abstractions & frameworks

Common language runtime

HTML 5 JS platform

Cross-page

Links & redirects

Micro-architecture

HTTP

Standard Browser

System boundaries are possibly the most important architectural design choice

Extremely loose coupling requires few rules, but they need to be enforced strictly

#6: Free-style Architecture

Context

- E-Commerce/Online shop (Retail)
- 100-120 developers
- ~10 self-contained teams

But …

- Lack of standardization led to inefficient UI integration at runtime
- Vast differences in API style, formats, documentation created needless extra work
- Despite no centralised frontend, a central frontend team created a new bottle neck

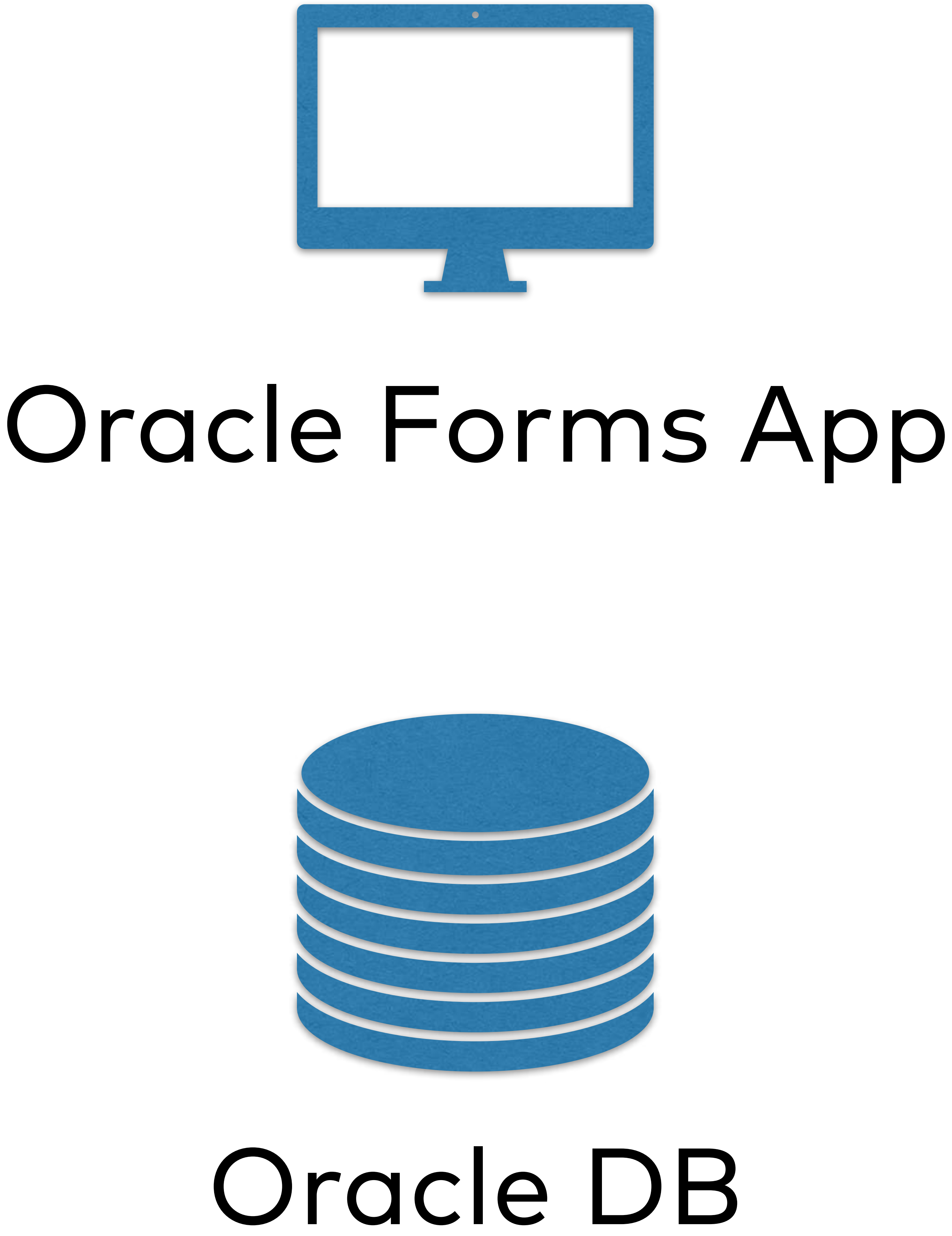
You cannot decide to not have an architecture; if you don’t actively create it, be prepared to deal with the one that emerges

There's a fine line between diversity (that adds value) and chaos (that doesn't)

#7: Cancerous Growth

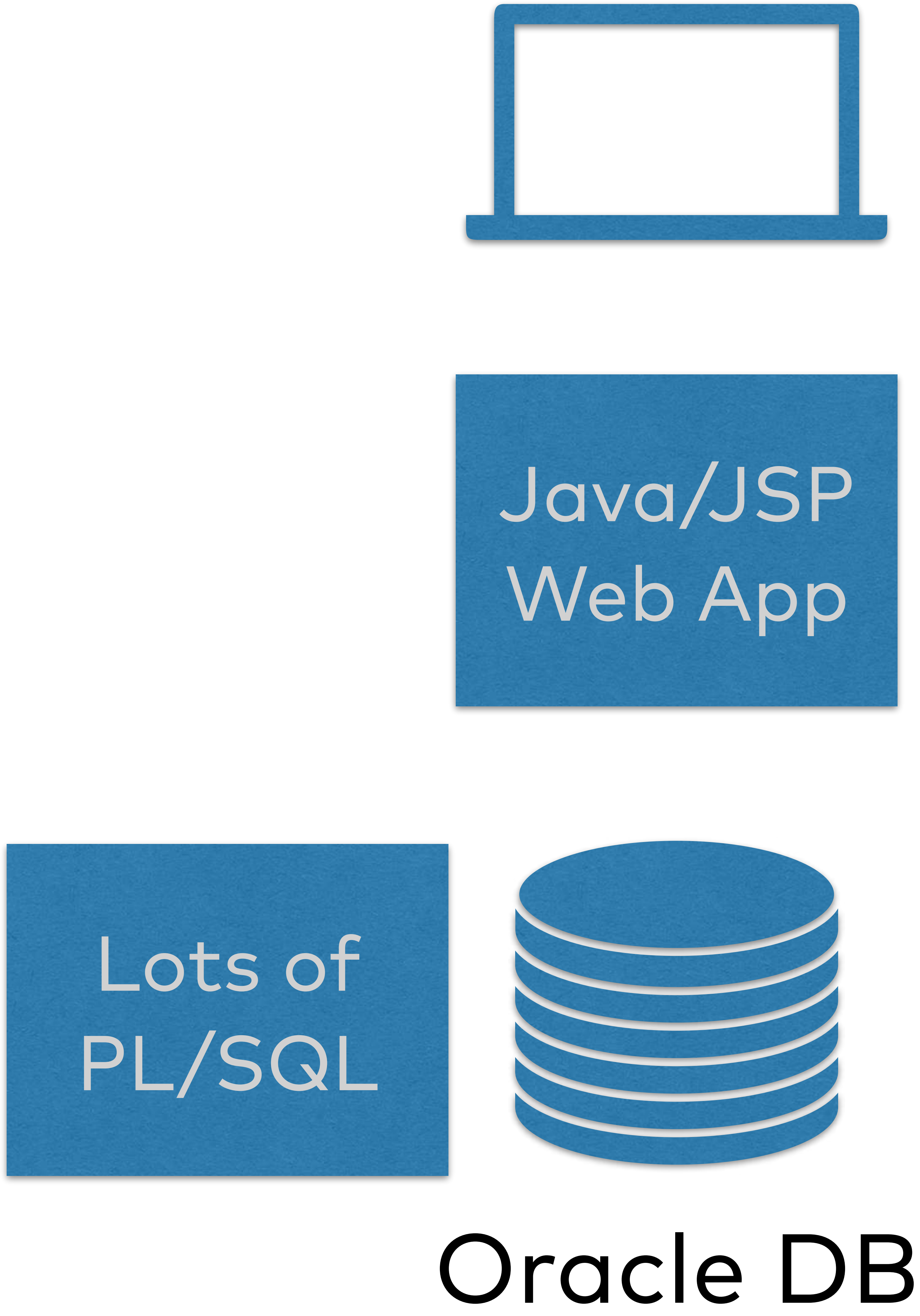
Context

- Financial services provider with independent brokers as clients
- ~30 developers
- 20 years of company history

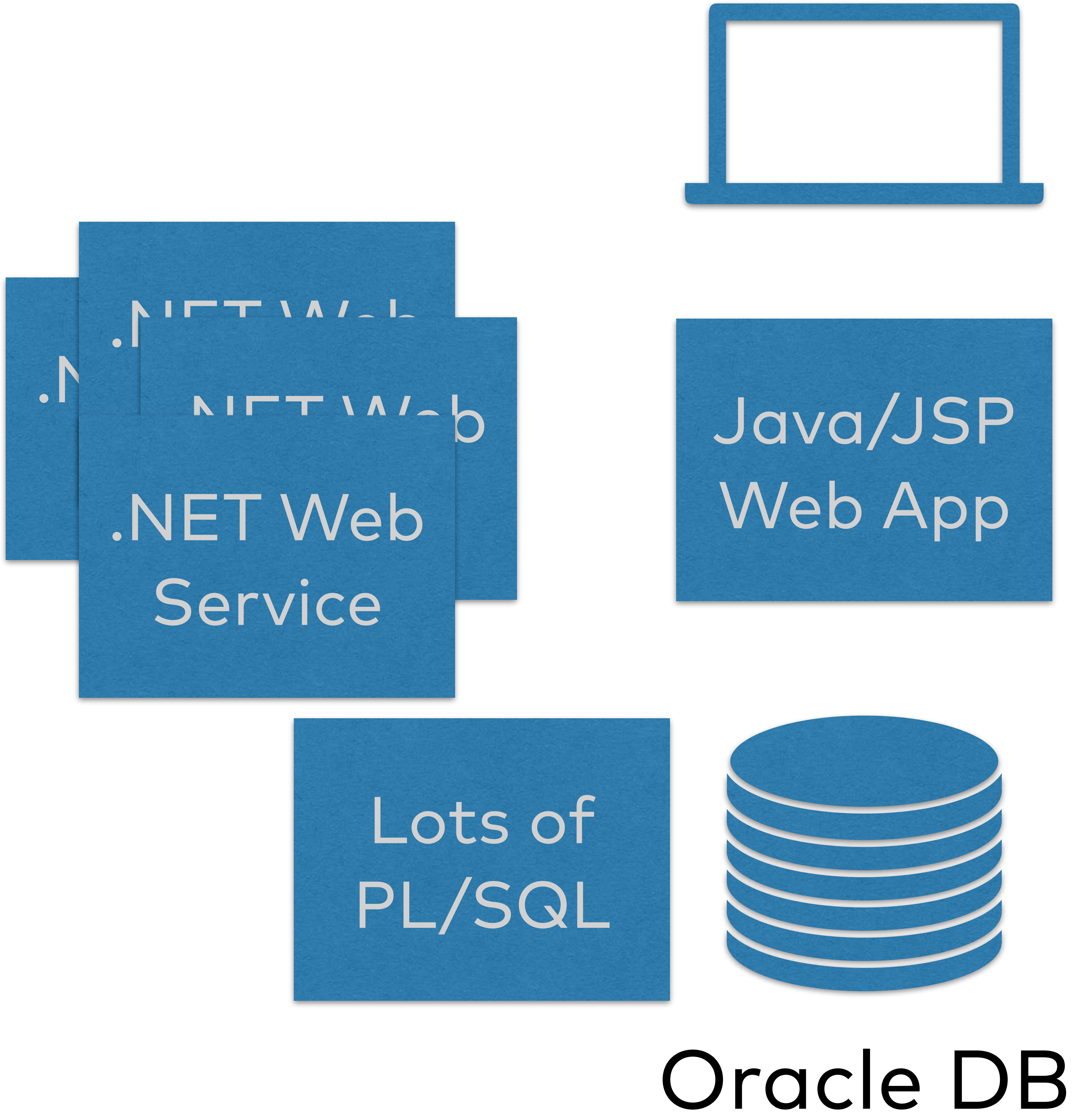


Oracle Forms App

Oracle DB



Java/JSP
Web App
Lots of
PL/SQL
Oracle DB

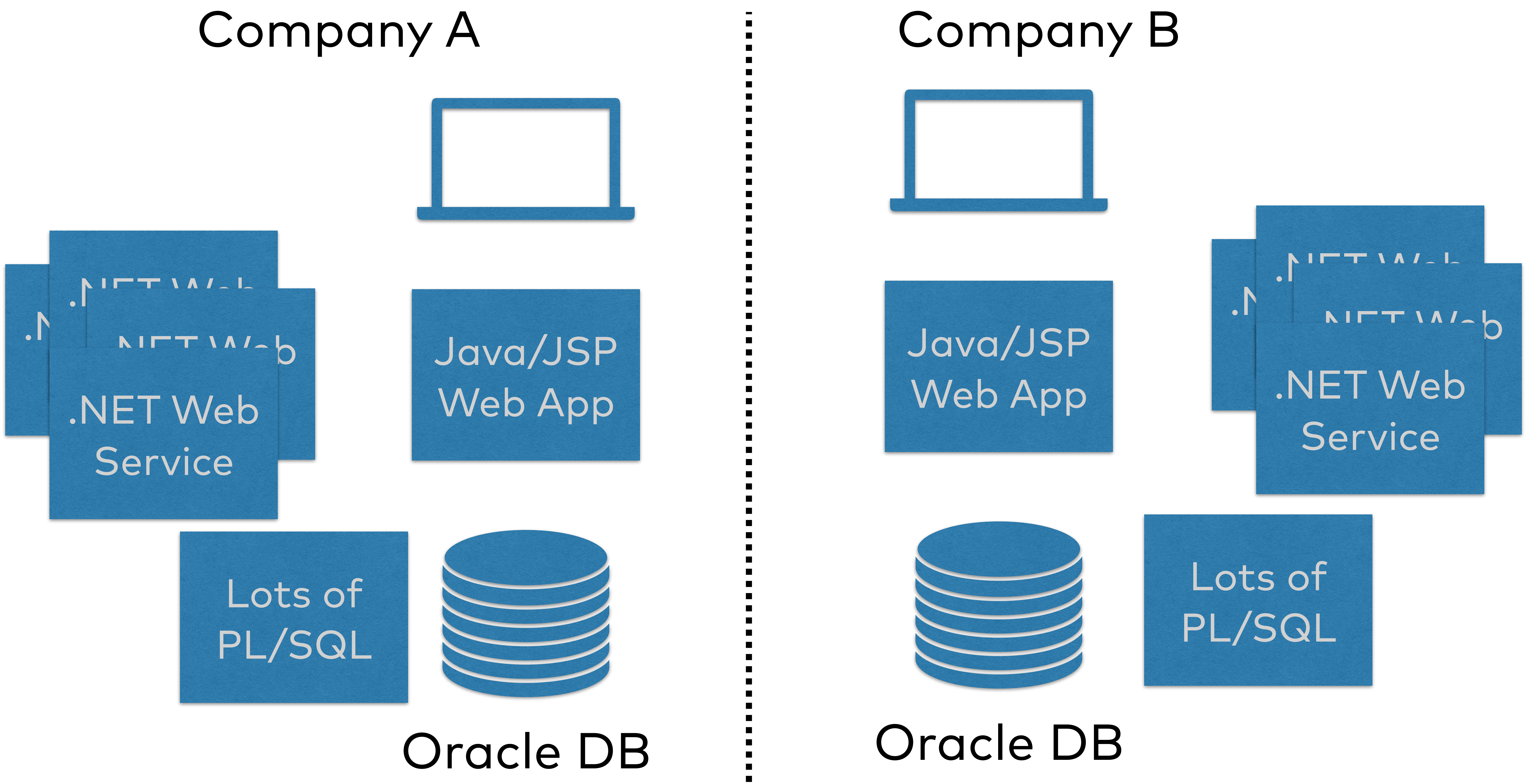


.NET Web Service

Java/JSP Web App

Lots of PL/SQL

Oracle DB



Company A

.NET Web Service

Java/JSP Web App

Lots of PL/SQL

Oracle DB

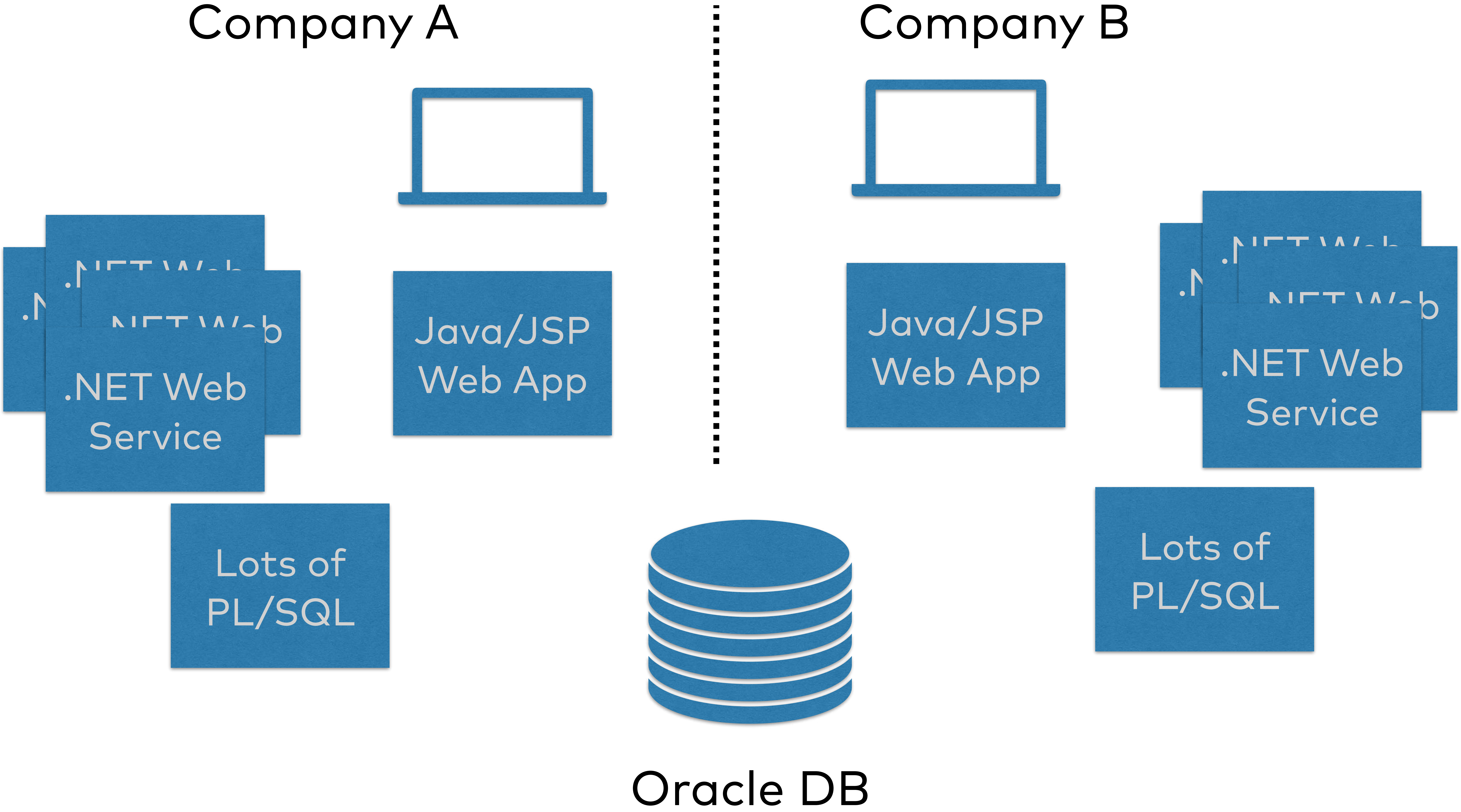
Company B

Java/JSP Web App

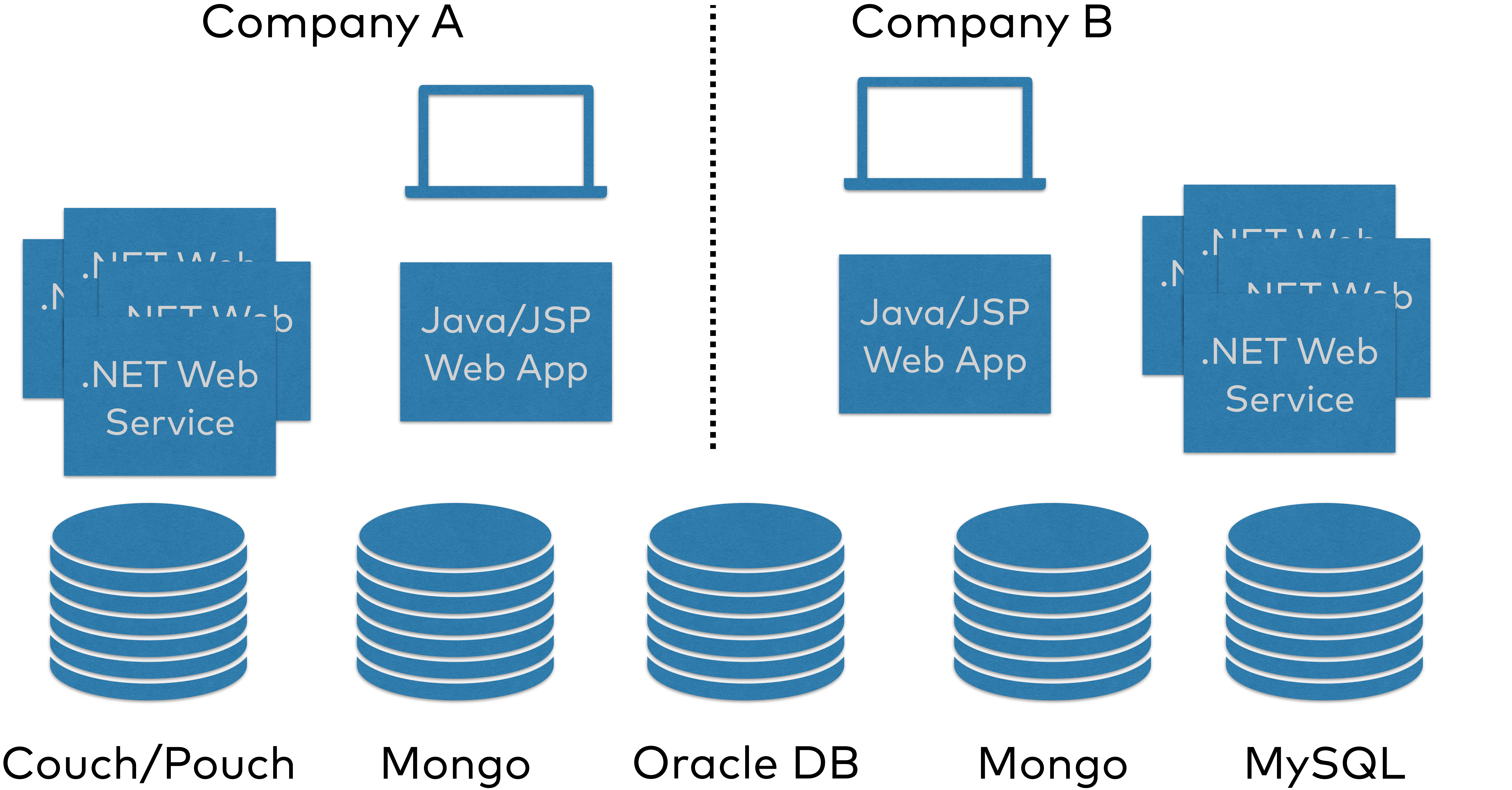
.NET Web Service

Lots of PL/SQL

Oracle DB

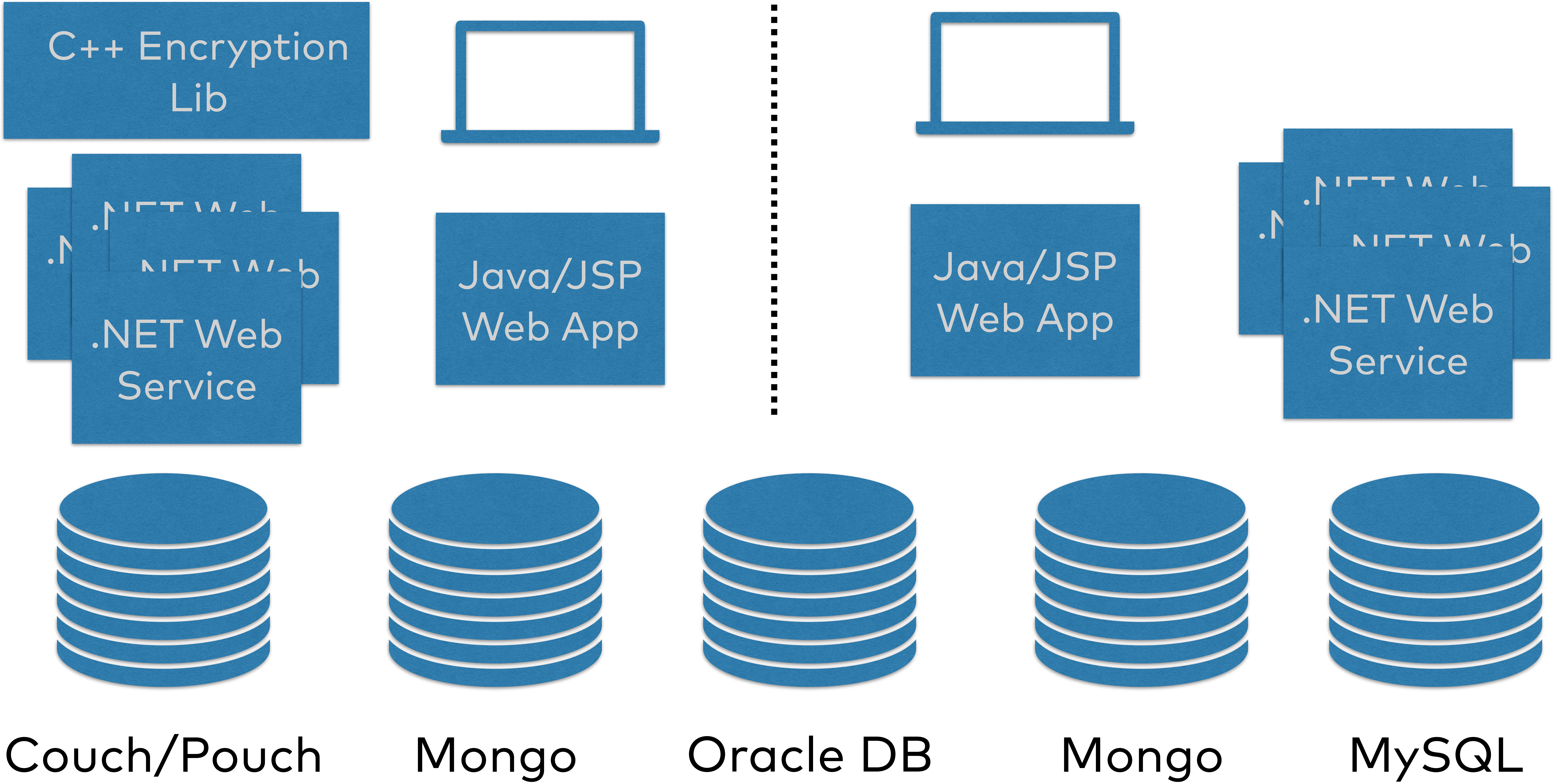

Company A
Company B
.NET Web Service
Java/JSP Web App
Lots of PL/SQL
Java/JSP Web App
.NET Web Service
Lots of PL/SQL
Oracle DB

@stilkov


Company A
Company B
.NET Web Service
Java/JSP Web App
Java/JSP Web App
.NET Web Service
Couch/Pouch
Mongo
Oracle DB
Mongo
MySQL

@stilkov

Company A | Company B



C++ Encryption Lib

.NET Web Service

Java/JSP Web App

Java/JSP Web App

.NET Web Service

Couch/Pouch

Mongo

Oracle DB

Mongo

MySQL

Successful systems often end up with the worst architecture

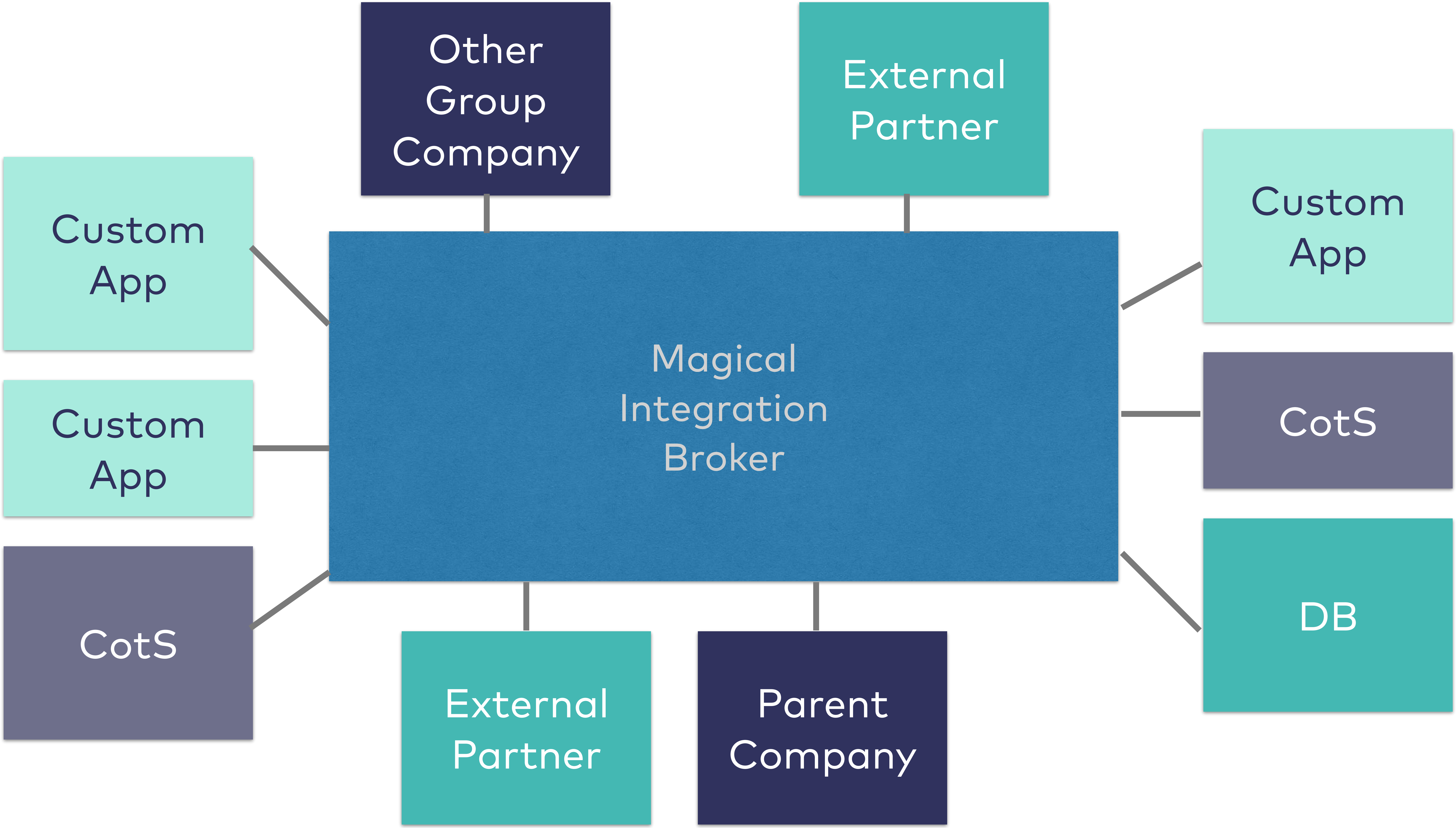
Unmanaged evolution will lead to complete chaos

Don't be afraid of some light architectural governance

#8: Improve with Less Intelligence

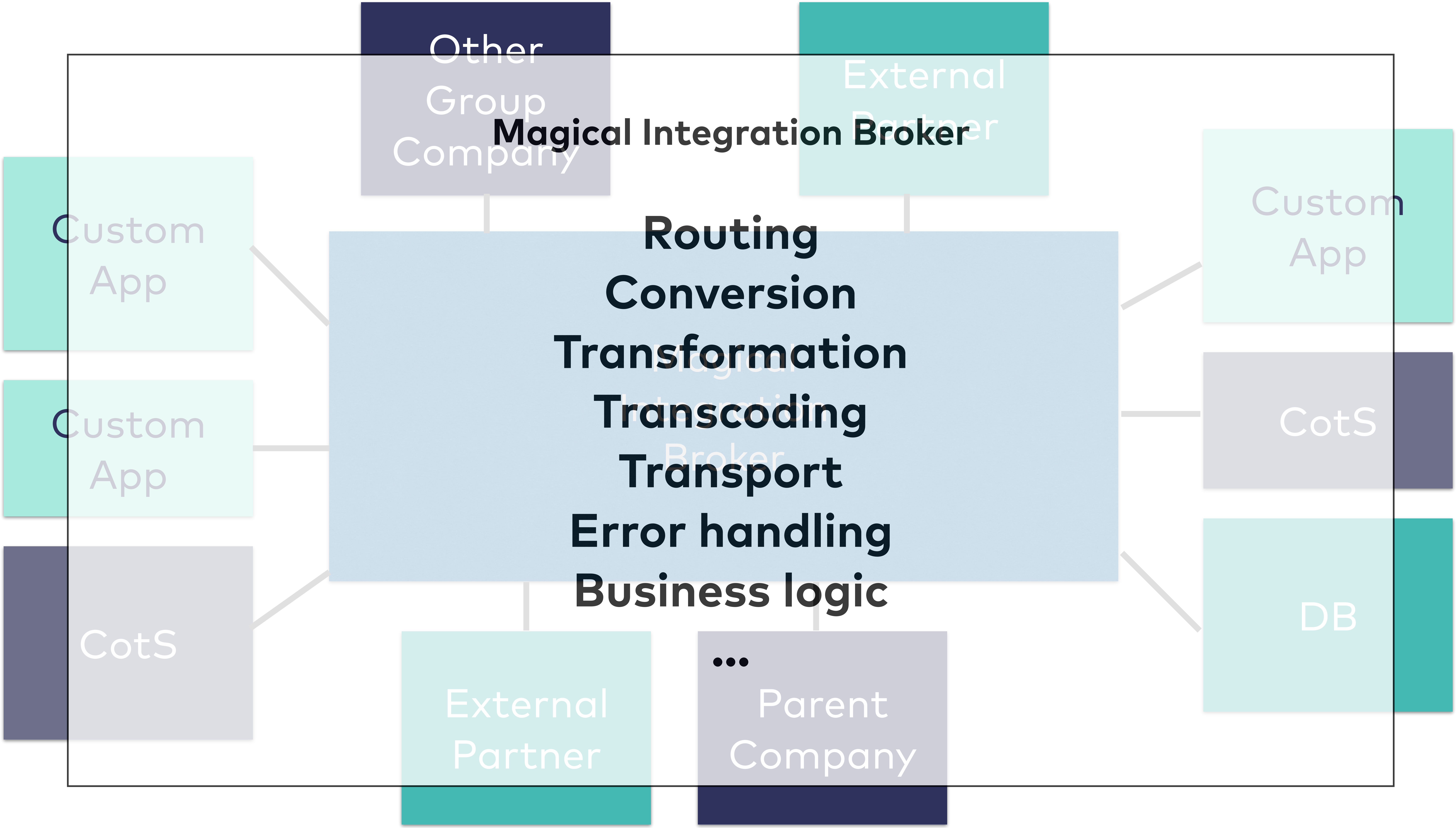
Context

- Bank with multiple CotS systems
- Highly proprietary integration solution phased out by vendor
- Project launched to replace commercial product with open source solution


Other Group Company
External Partner
Custom App
Custom App
Magical Integration Broker
Custom App
CotS
CotS
DB
External Partner
Parent Company

@stilkov

Magical Integration Broker



Other Group Company

External Partner

Custom App

Custom App

CotS

Magical Integration Broker

Custom App

CotS

DB

External Partner

Parent Company

Routing

Conversion

Transformation

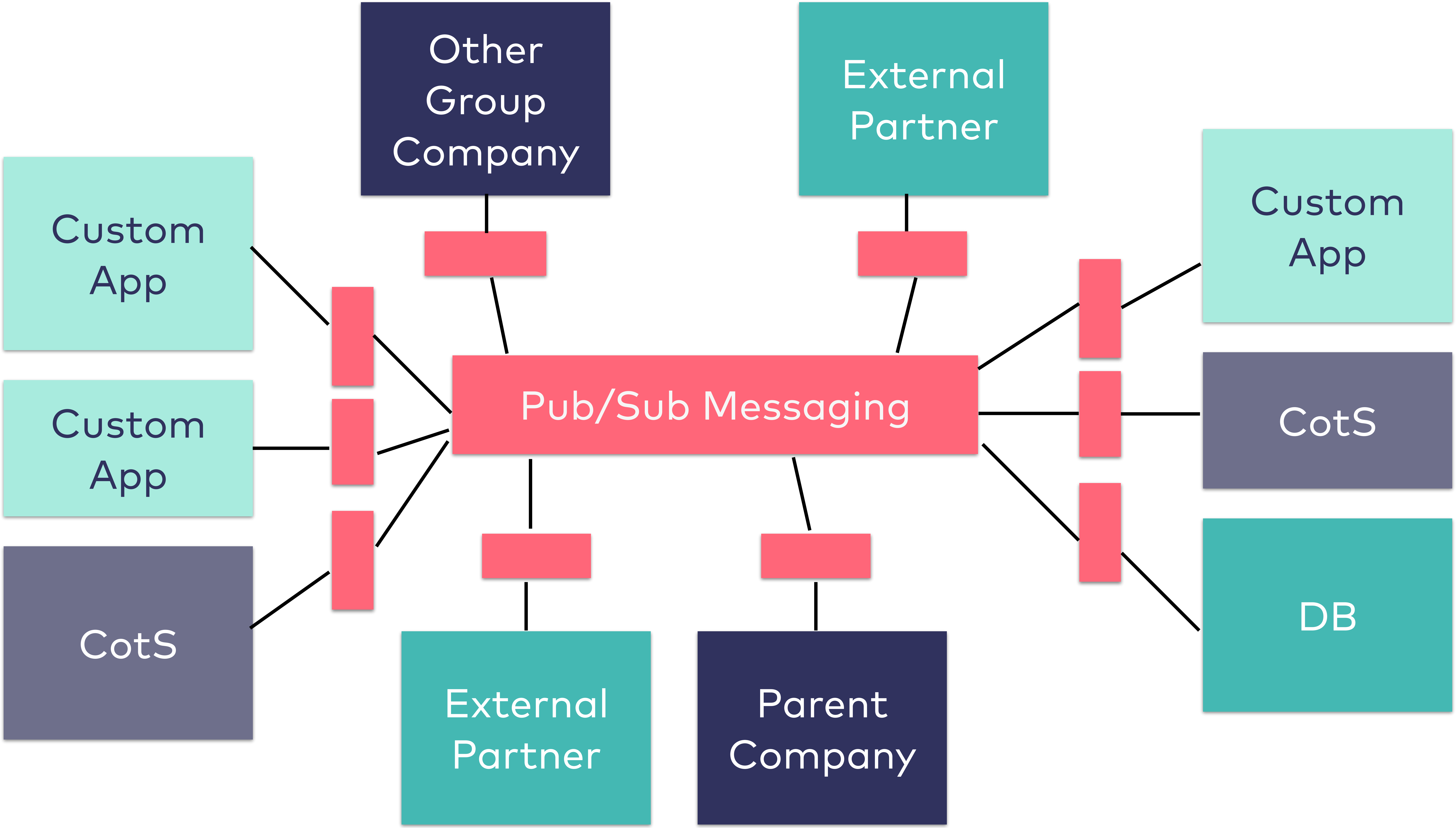
Transcoding

Transport

Error handling

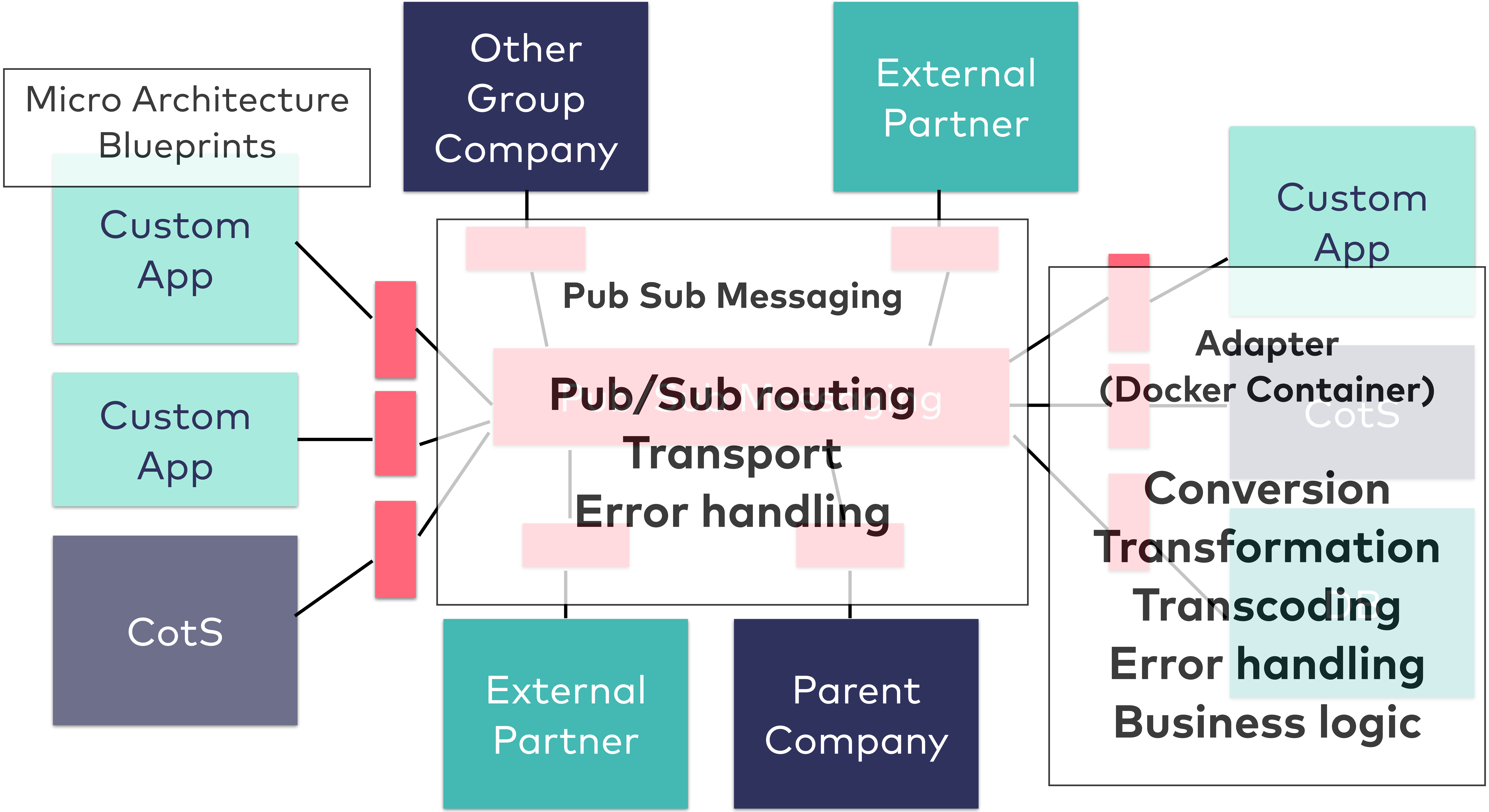
Business logic

…


Other Group Company
External Partner
Custom App
Custom App
Custom App
Pub/Sub Messaging
CotS
CotS
DB
External Partner
Parent Company

@stilkov

Micro Architecture Blueprints



Other Group Company

External Partner

Custom App

Custom App

Custom App

CotS

Pub Sub Messaging

Pub/Sub routing
Transport
Error handling

Adapter
(Docker Container)

CotS

Conversion
Transformation
Transcoding
Error handling
Business logic

External Partner

Parent Company

@stilkov

Lessons learned

- Smart endpoints, dumb pipes (cf. Jim Webber)
- Value of specific over generic solutions
- Micro architecture with blueprints

Prefer smart endpoints and dumb pipes over overly complex, powerful middleware (cf. Jim Webber)

Be brave enough to pick specific solutions and avoid over-generalization

Takeaways

1.
Don’t be afraid of architecture

2.
Choose the simplest thing that will work

@stilkov

3.
Create evolvable structures

4.
Manage your system's architectural evolution

5.
Don’t build road blocks – create value and get out of the way

That's all I have.
Thanks for listening!

Stefan Tilkov
@stilkov
stefan.tilkov@innoq.com
Phone: +49 170 471 2625

INNOQ
www.innoq.com

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim am Rhein
Germany
Phone: +49 2173 3366-0

Ohlauer Straße 43
10999 Berlin
Germany
Phone: +49 2173 3366-0

Ludwigstr. 180E
63067 Offenbach
Germany
Phone: +49 2173 3366-0

Kreuzstraße 16
80331 München
Germany
Phone: +49 2173 3366-0

innoQ Schweiz GmbH

Gewerbestr. 11
CH-6330 Cham
Switzerland
Phone: +41 41 743 0116

@stilkov