Beyond "Hello World"

handling complexity and organization with large systems

Sebastian Verheughe

Chief Enterprise Architect @ FINN.no

FINN fagkveld - 7 March 2017

What to Expect

- You will learn about **real challenges** we observe at FINN
- You will learn how we **try to solve** these challenges
- I will provide you with **some recommendations…**

Did you really think
FINN has **everything** solved?

Chapter I

Partitioning of FINN

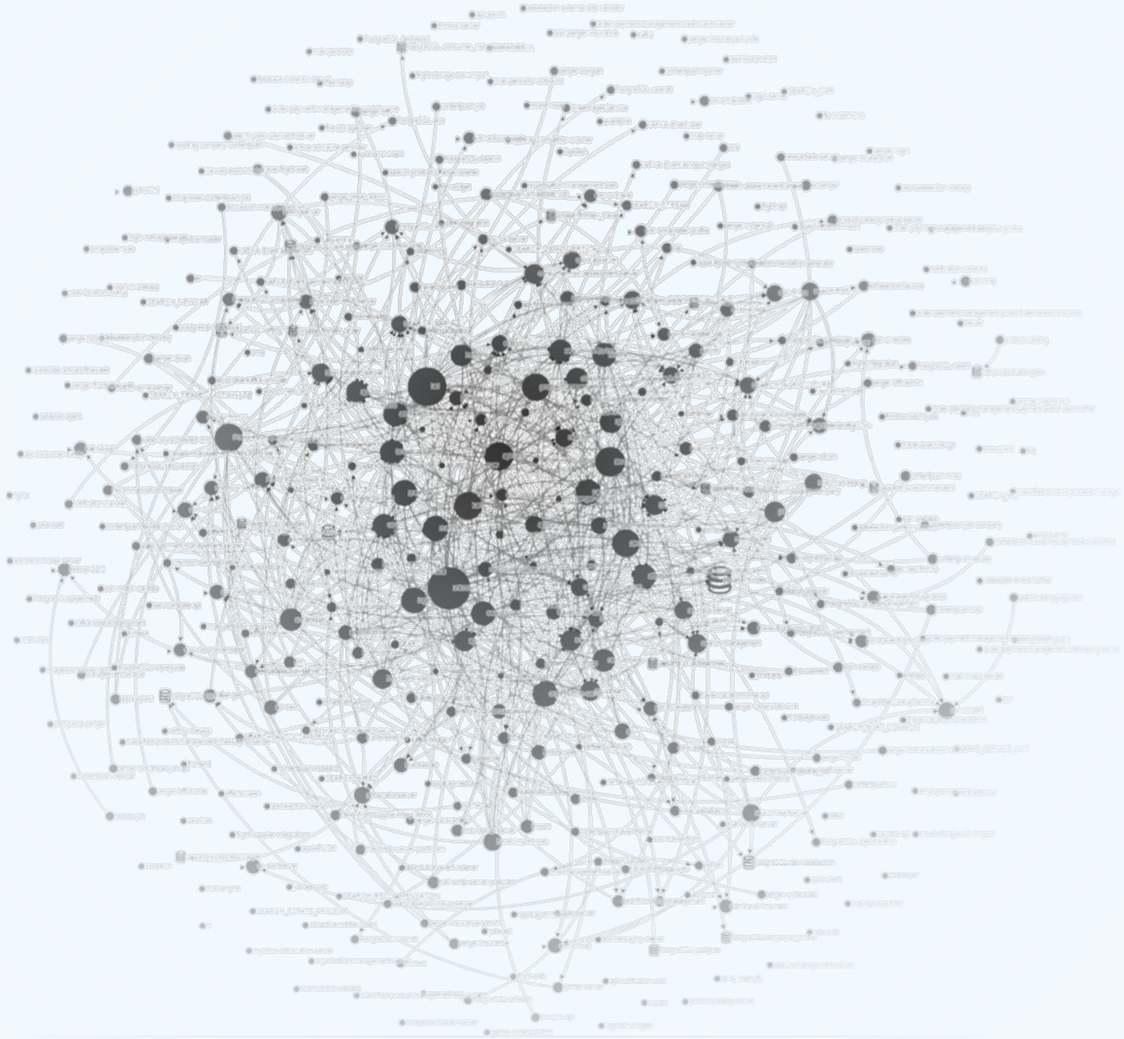
The Value of **Domain Driven Design**

"Getting service boundaries wrong can be expensive. It can lead to a larger number of cross-service changes, overly coupled components, and in general could be worse than just having a single monolithic system"

***Sam Newman**, Thoughtworks*

Extract from the article Microservices for greenfields?

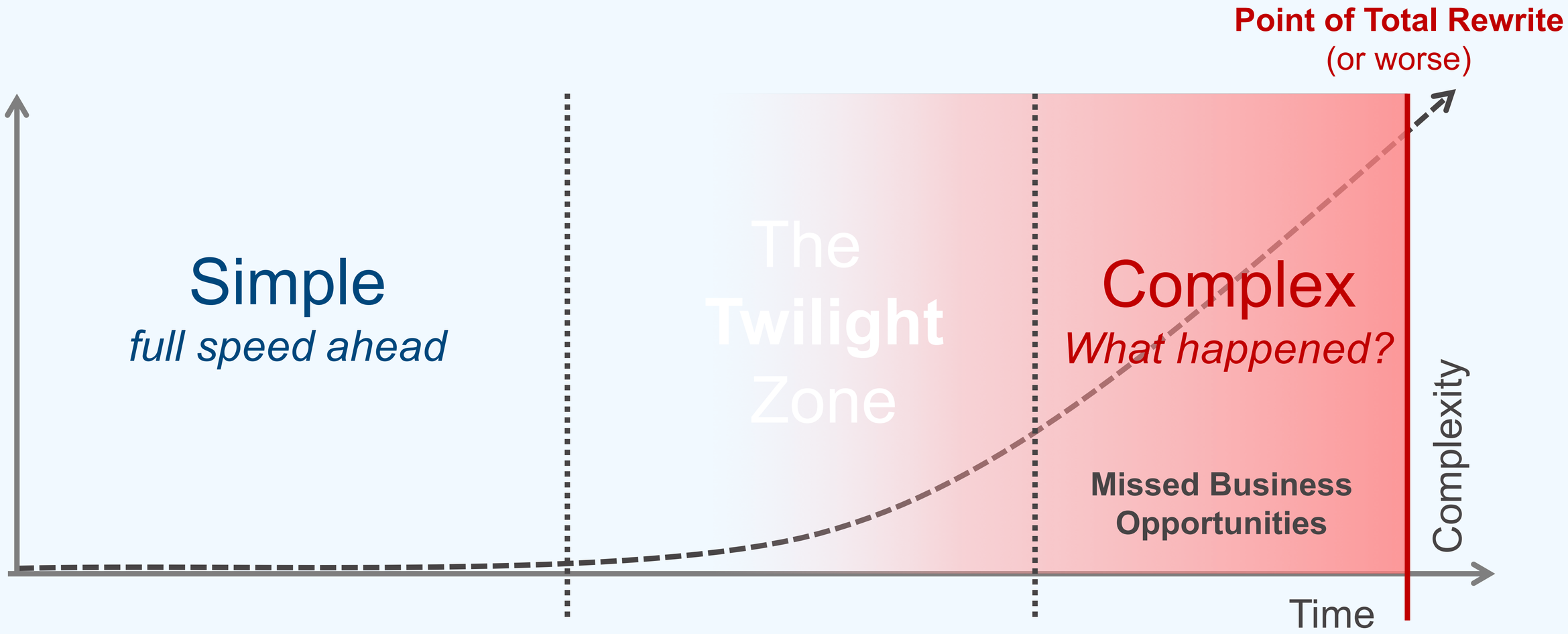
Dependency Graph of FINN



With microservices, **each service is simpler** but the **distributed environment is more complex**

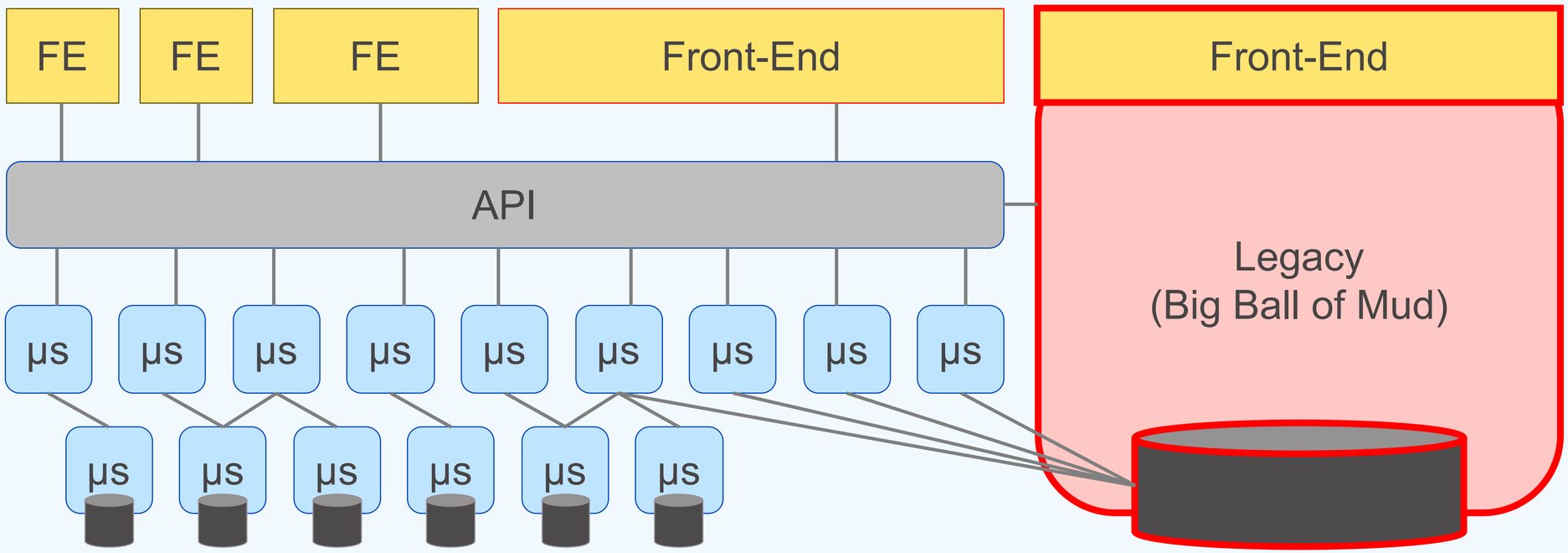
Risk of **Unhandled** Complexity

Key areas with unhandled **complexity**, may **damage** your business



Conceptual Architecture of FINN

Not only microservices, monolithic **front-end** and legacy **system/database**



The highest **complexity** seems to be related to a few **central structures,** and especially legacy **database integration** makes it hard to split.

Architecture **Challenges** @ FINN

- A few **central structures with high complexity** used in many contexts proves very hard to modify and can cause unintended behavior elsewhere.
- High coupling between services and **long request call chains** affects performance and availability negatively (8 fallacies of distributed computing).
- Clients aggregating services become **monolithic front-ends,** bottleneck, single-points-of-failures, and require / desire standardization
- Sharing **data** in a good way **without lots of dependencies**.

Customer Service

Agreement Service

User Service

Store Service

Partitioning **Strategy**

Should we **partition** by **business entities**, **business objectives** or **both**?

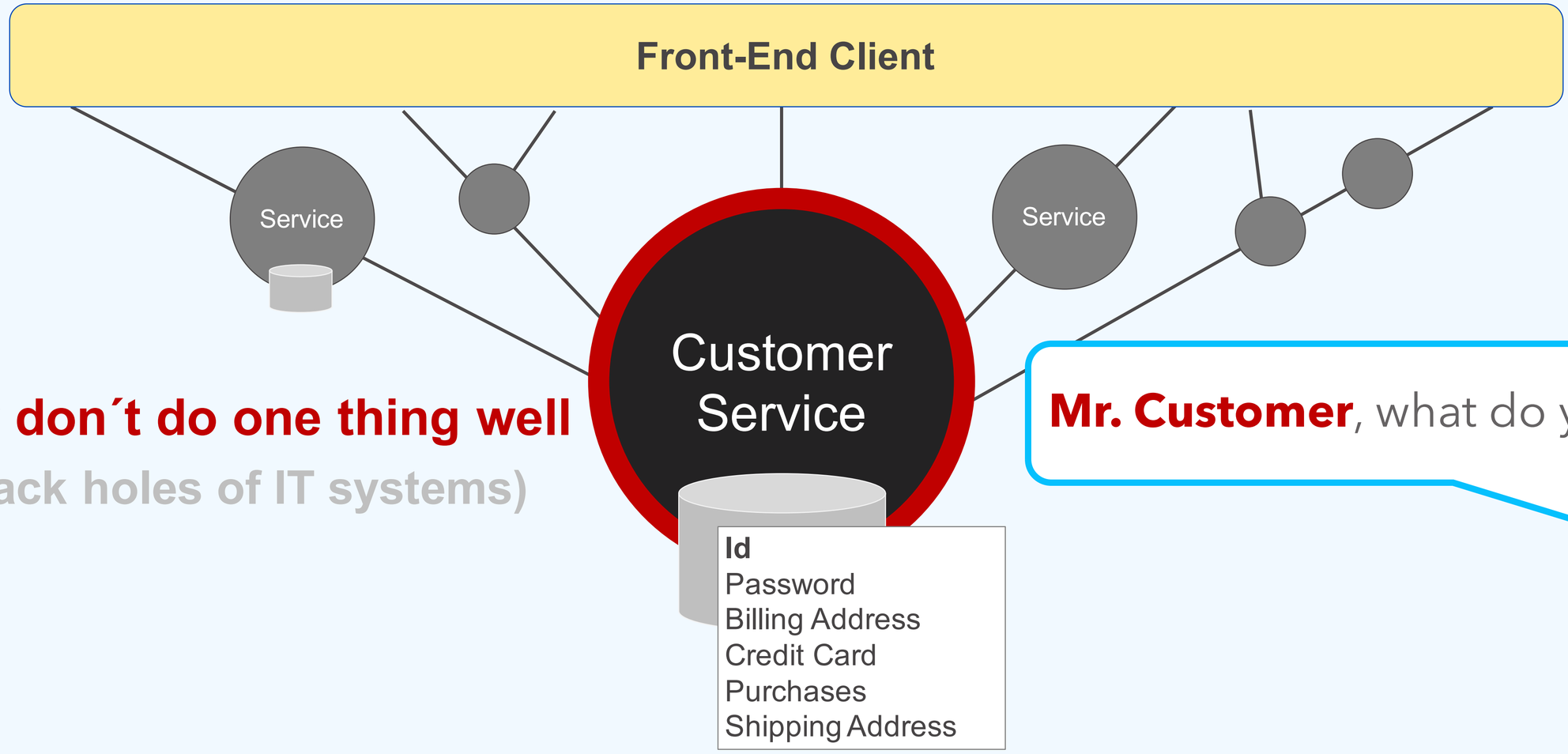
Login Service

Authentication Service

Enrollment Service

Review Service

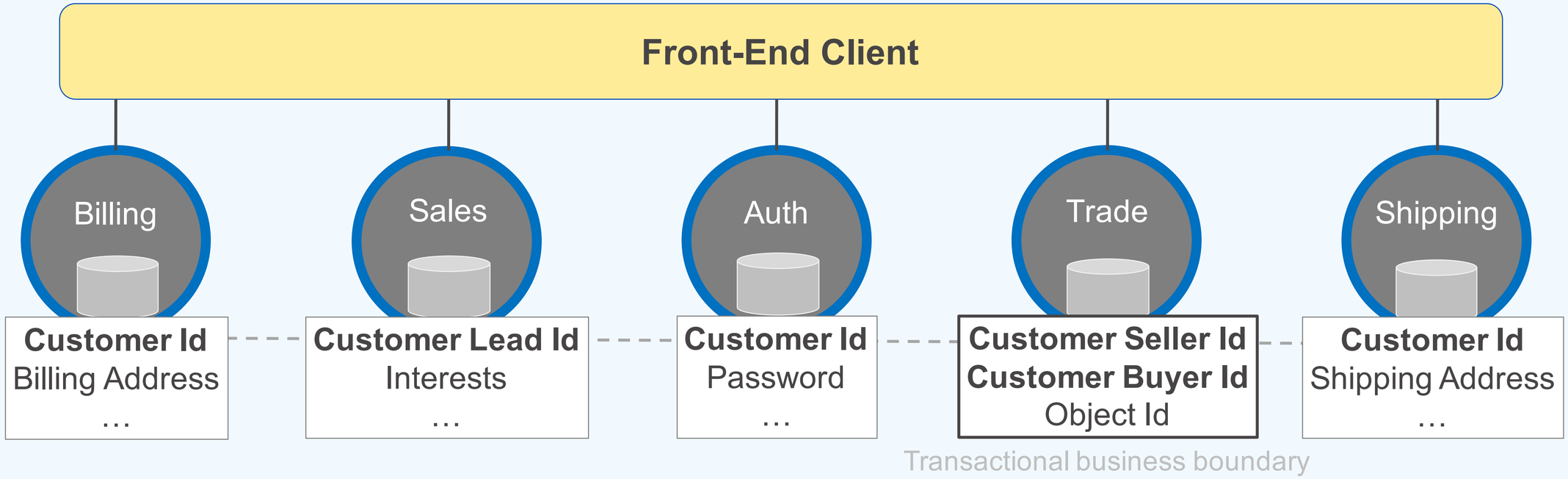
Challenges with Services Based on Entities



They don´t do one thing well

(black holes of IT systems)

Benefits with Services Based on Objectives



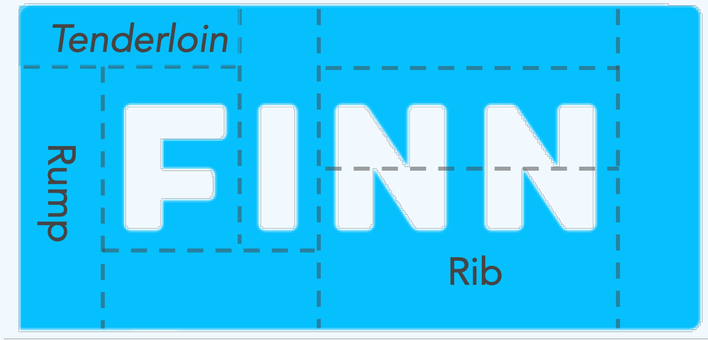
They do one thing well!

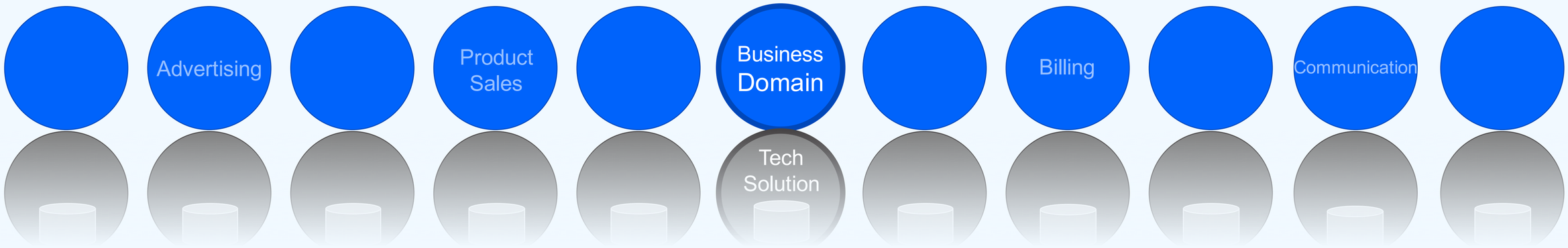
(and use multiple models of entities, e.g. Customer to do so)

Well, we all **know** why we need **Mr. Billing** around…

Aligning the Architecture with the Business

Technical solutions **align** with **business capabilities** (objectives / domains)



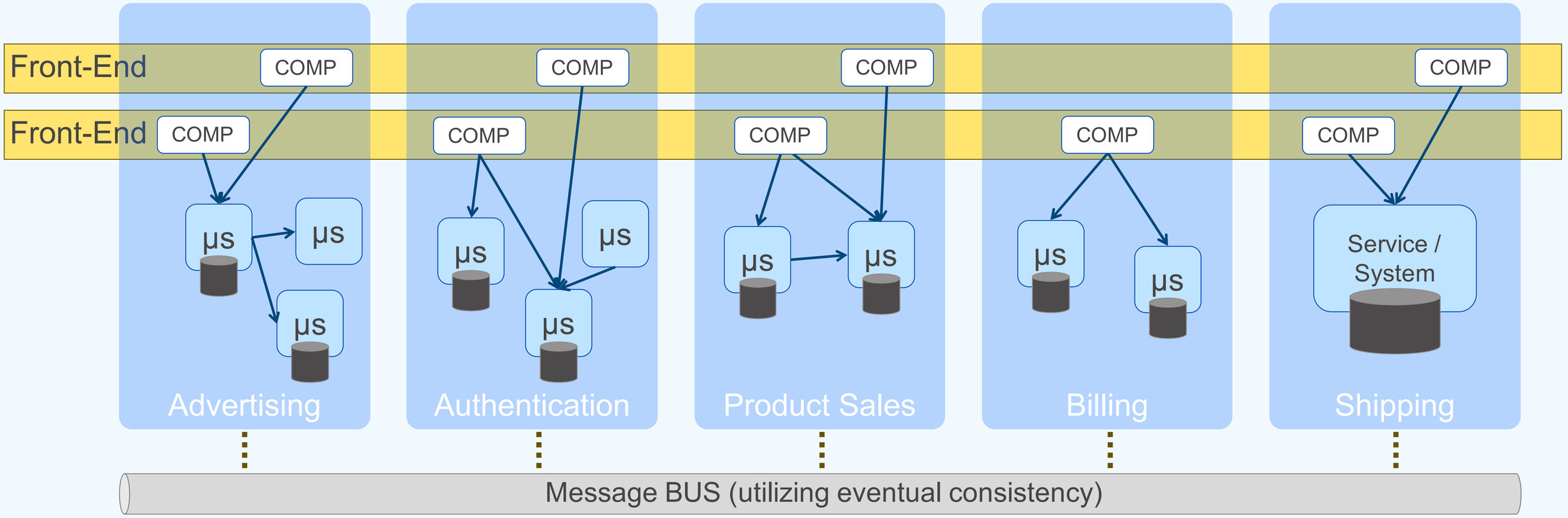


They will be **stable**, with a **minimum of dependencies** between them,

and form **transactional** boundaries from a user / business perspective

Target Architecture (we call it **direction** in FINN)

The front-ends **aggregate** responses from services **clustered** by domain



Partitioning applies to both **front-end** (components) and **back-end**

Capability / domain (problem space) & services (solution space)

Architecture Recommendations

Invest heavily in building competence around distributed systems design - NOW!

Writing a **single microservices is easy**, testing and running **hundreds together is hard**.

Tomorrow: *Use a circuit breaker (Hystrix), use pub-sub between two services to communicate, mock dependencies between two services for testing, read “8 fallacies of distributed systems”.*

Design you services around business objectives, with several entity models

Tomorrow: *select a* ***complex entity service*** *(or table) and divide it by different known contexts*

Chapter II

Organization Strategy

Organizational Observations @ FINN

- Many features required **several hand-offs between teams** in order to be solved.
- Teams where **not able deliver a complete user / business capability** by themselves.
- Layered teams sometimes became **overly focused about technical perfection**.
- Static teams made it **difficult to move resources** to **changing business** challenges.

Conway's Law

“Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations”

Melvin E. Conway

Extract from the paper How do Committees Invent?


"Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations"
Melvin E. Conway

Corollary to Conway´s Law

“If you design a system, but you didn´t design the organization structure, you´re not the system´s designer.”

Mathias Verraes

Extract from Twitter

How to Partition Organizations in General

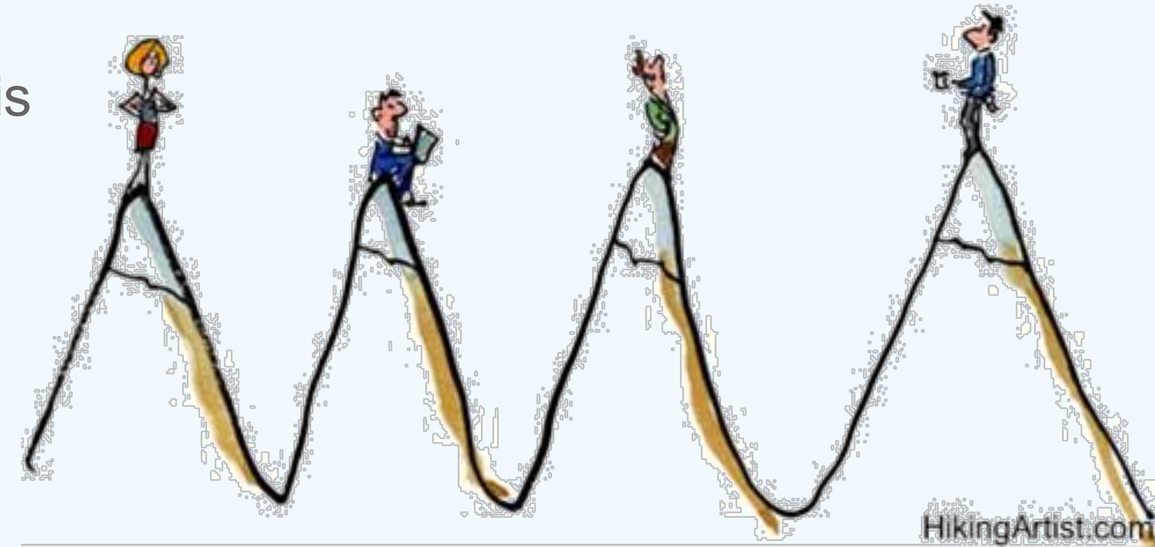
"…Organizations can come in two extreme forms:

in totally **mission-oriented** form or in totally **functional** form…

In the real world, of course, we look for a compromise between the two extremes"

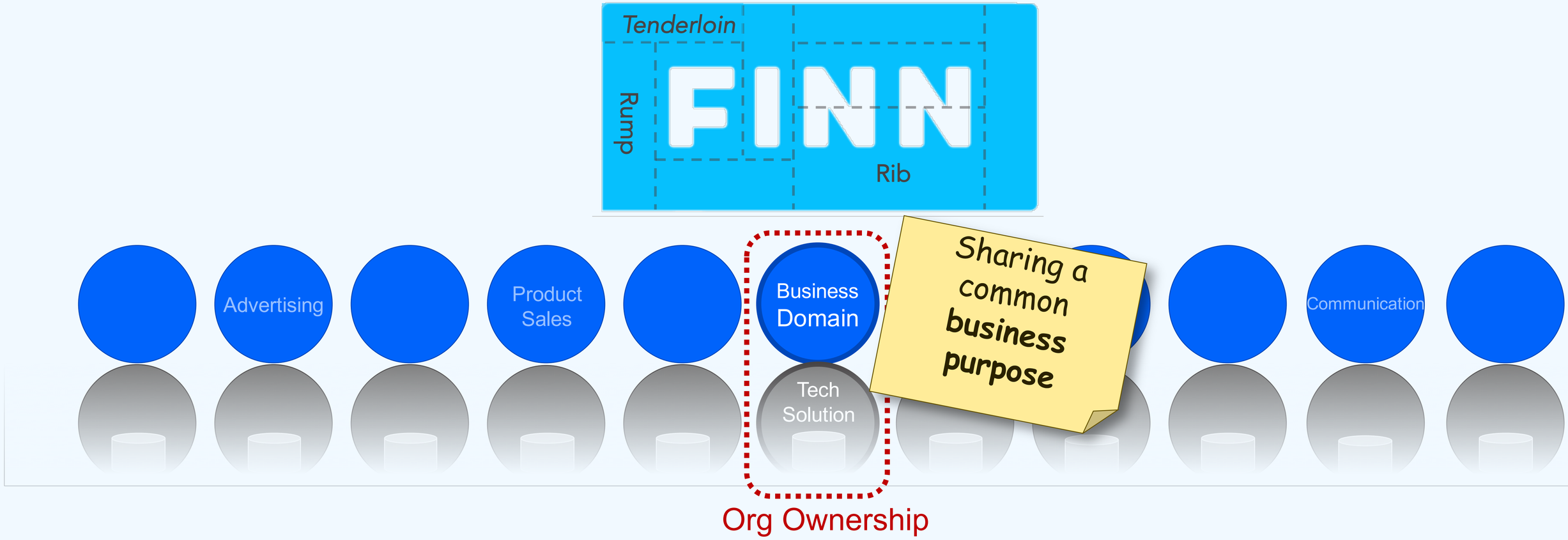
- Andy Groove (book: High Output Management)

However, with every partitioning of the organization, it is extremely important to **compensate** in the orthogonal direction in order to avoid silo organizations, sub optimizations, and failing to deliver on challenges across the organization.



Aligning the Organization with the Business

The **organization** should **align** with the **business** (and the technical solution)

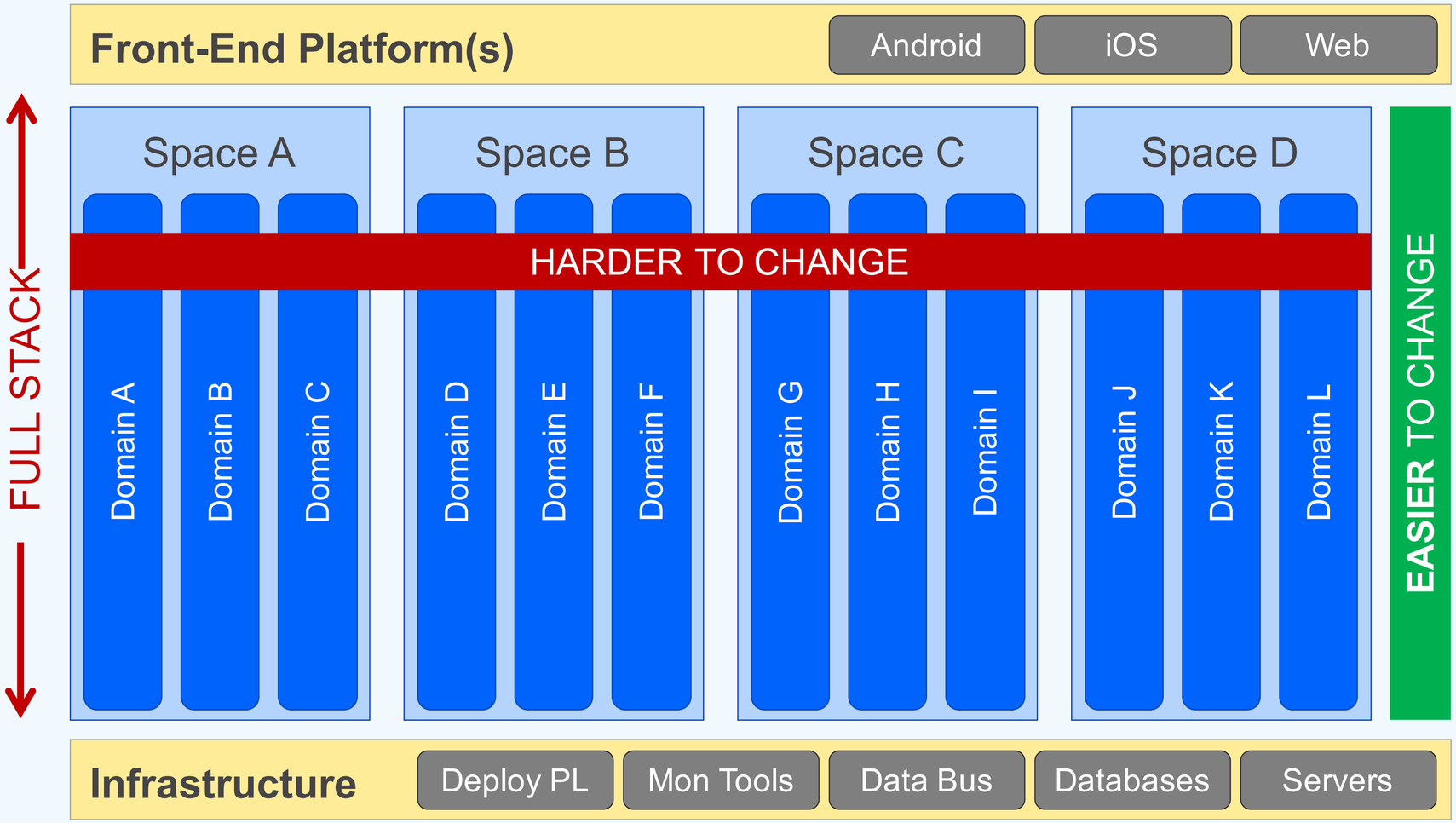


The goal was an organization that more effectively was able to deliver on the business challenges

Inverse Conway Maneuver: Organize to promote your desired architecture (end state)

FINN Technology Organization

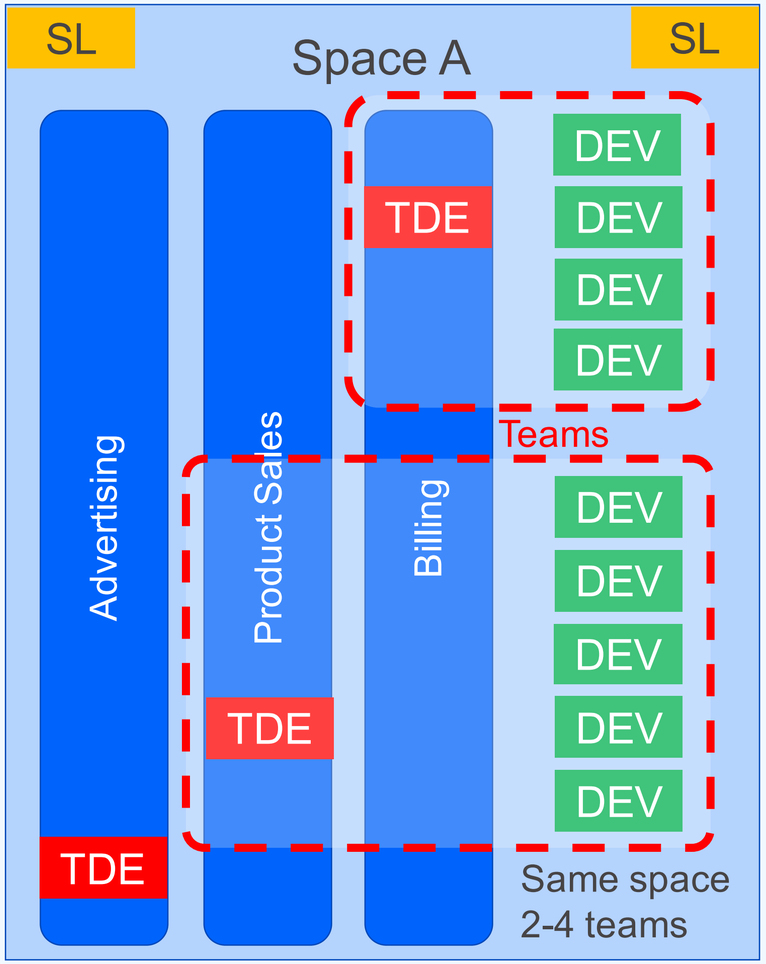
Organized **primarily** by **business domains**, secondarily by cross-cutting concerns



All cross-cutting teams are structured as **enabling** teams, meaning they should deliver **self-service** solutions **upfront** to the vertical functional teams.

Spaces allows for **Flexible** Developer Allocation

This is how we are able to move developers around based on **prioritization**



Space Leads, making it all work / flow.

Developers, available for work that is prioritized.

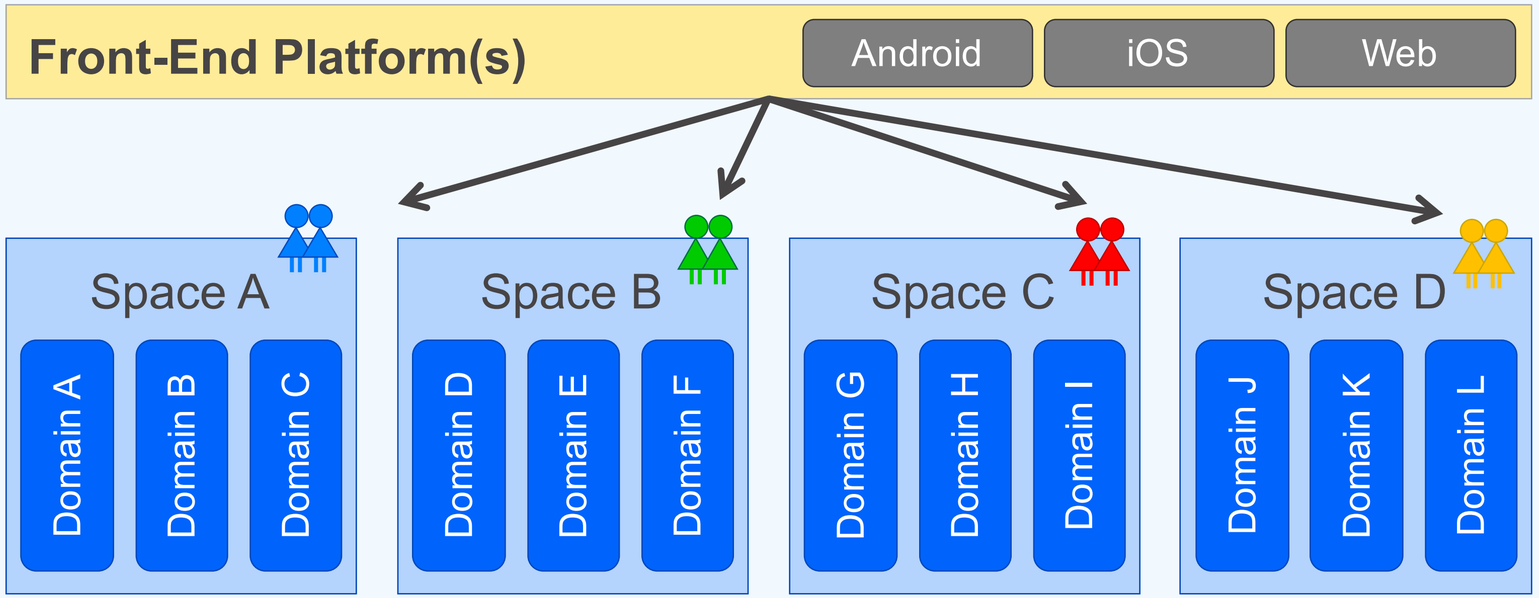
Technical Domain Experts (tech leads**),** responsible for the long-term development of technical solutions supporting a domain, and the Quality of Service of them.

Architects (4 in FINN), responsible for everything that affects multiple domains…

Distributing the (central) Apps Team

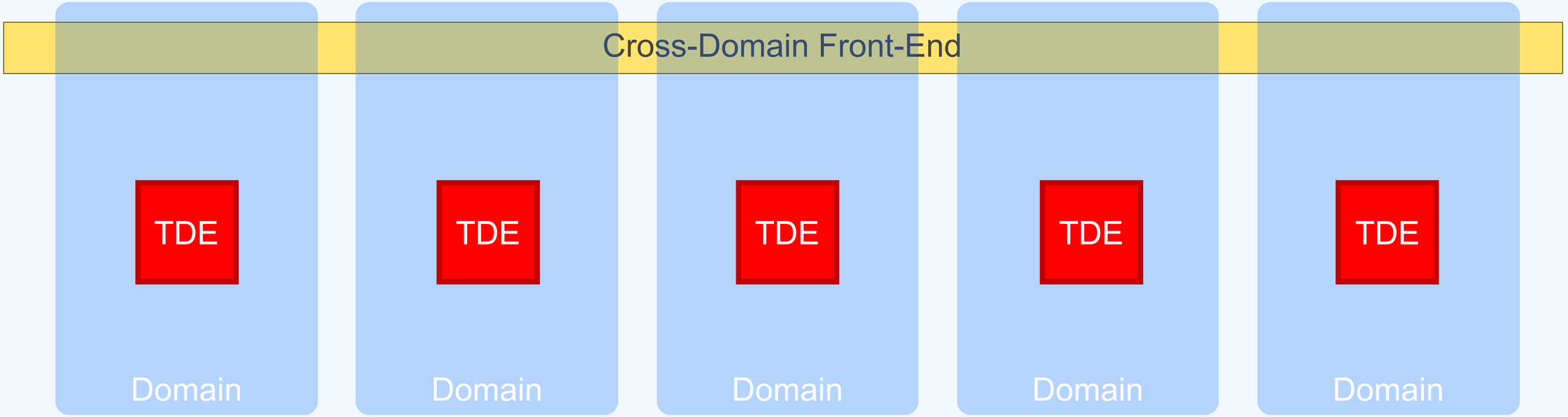
About 25% of visits are from native apps and growing, but we had **one team**.

Since we believe that our **main biz challenges** are **not related to native apps** development, it makes sense to focus on **full-stack** functional and more **autonomous** teams instead.



Decentralized Authority (Framing)

The TDEs are **free** to choose solutions within their domain, no sign-offs needed



The architects only govern **integration** and **boundaries** between the **domains** and the **front-end,** in addition to help **succeeding** with the implementation of the **strategy**.

Reorganization Observations (one year in)

- **A single team has proven that they can improve a full-stack capability autonomously**, and it seems like faster (we have no god way of measuring)
- **Tech leads are starting to take full-stack responsibility for the capability they deliver**.
- **The teams can freely choose how they within a domain solve each business challenge**, there are less technical “religious” discussions. And no exploitation of new technologies**.**
- **One year in, the systems does not automagically partition themselves by org. structure**, but needs to be driven in each case. The inverse Conway maneuver only removed barriers.
- **The organization structure (silos) works as barriers when cross domain work is needed**, and developers rather wait than contribute if changes are needed in someone else domain. It may be a smaller problem now, but we need to change the culture and expectations.

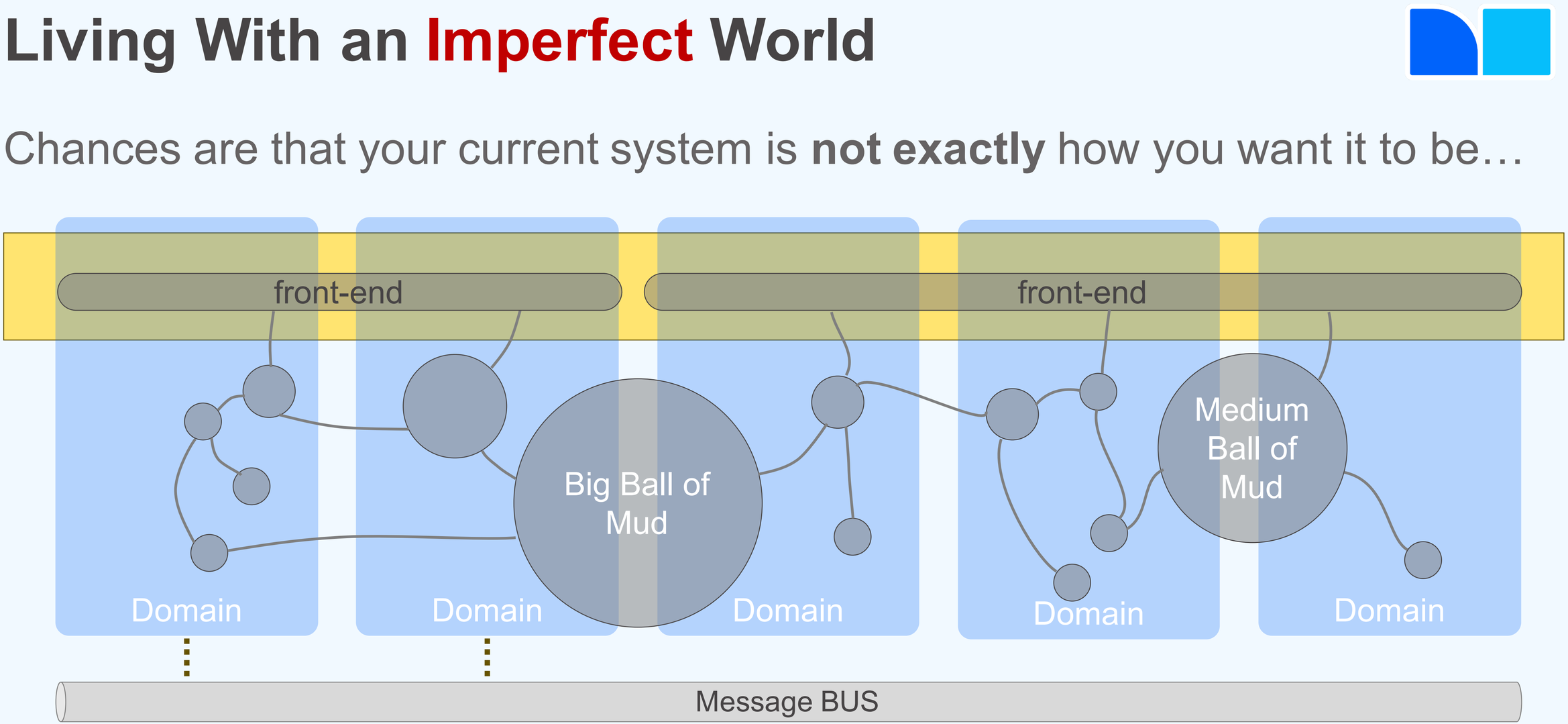
Organizational Recommendations

- Primarily **align your teams** with your **business domains**, not tech layers
- **Decentralize decision making** whenever applicable
- Structure **cross-cutting teams** as **enabling teams**, not bottlenecks or guards
- Make sure your organization culture **encourages cross-organizational collaboration**

Chapter III

Being Strategic about Complexity

Living With an Imperfect World



Chances are that your current system is **not exactly** how you want it to be…

What to **do** about it…

The Total Rewrite Disease

Identifying **What** Strategic Capabilities to Improve

Strategy: **fix the basics**, **focus on advantages**, leave the rest when possible…

Ignore **how** for now…

← Why customers **choose** you…

← Avoid **feature** driven…

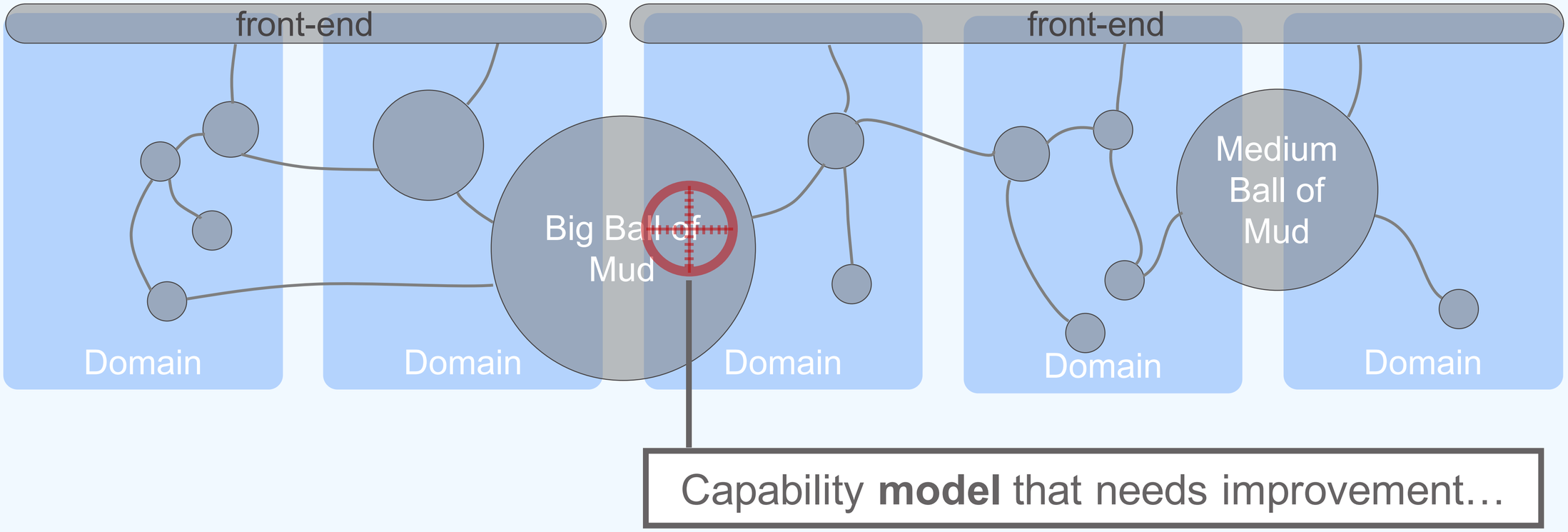
← Customers **expect / require**…

Fastest Minimum Viable Capability

Identifying the **smallest step** needed to deliver a modified or new capability

Understanding How & Where to Deliver

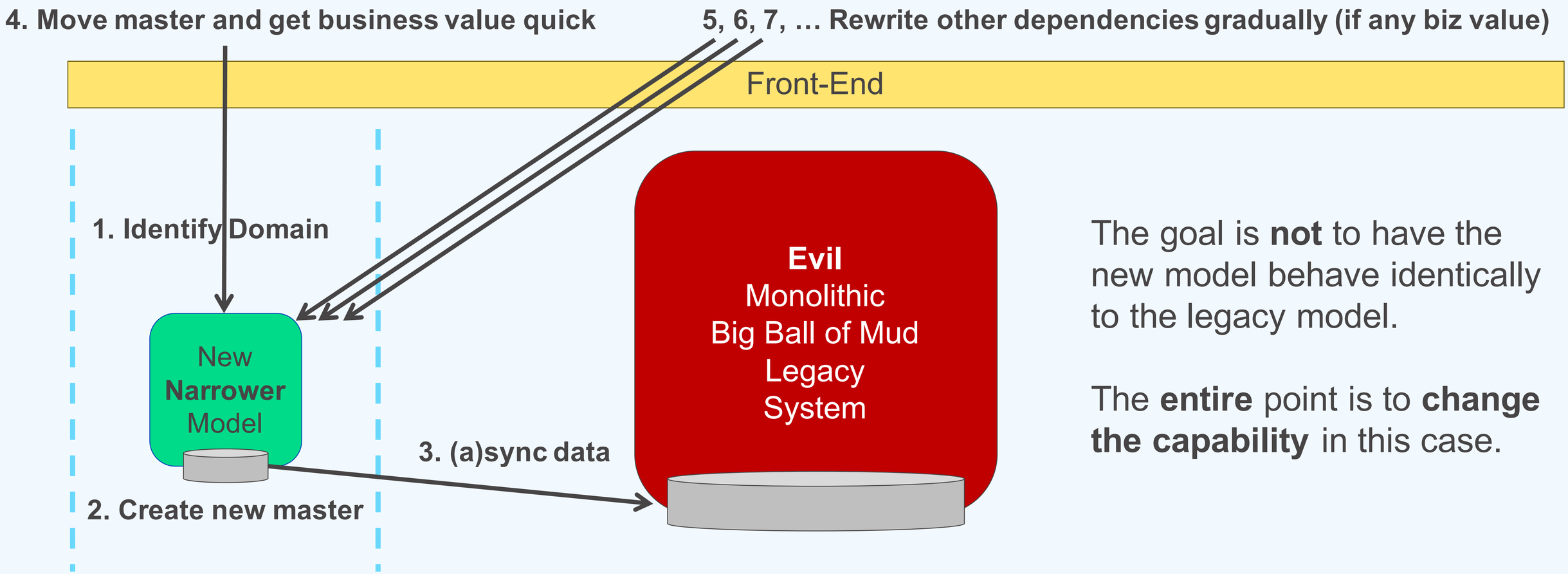
When you know **what**, identify **where** it should go and **how** to deliver it.



About this time, it might be helpful with an architecture strategy…

Slicing a Part of a the Legacy Model @ FINN

Illustration of **how we modified** parts of a legacy model **without needing to rewrite all dependencies** at the same time / at all. (Strangler Application)



The goal is **not** to have the new model behave identically to the legacy model.

The **entire** point is to **change the capability** in this case.

Strategic Delivery Recommendations

- Always **question large rewrites** without a clear strategic **user / business value**
- Deliver new / modified capabilities **outside of legacy** monoliths
- Always find the **simplest** / smallest solution that **delivers on strategic capabilities**

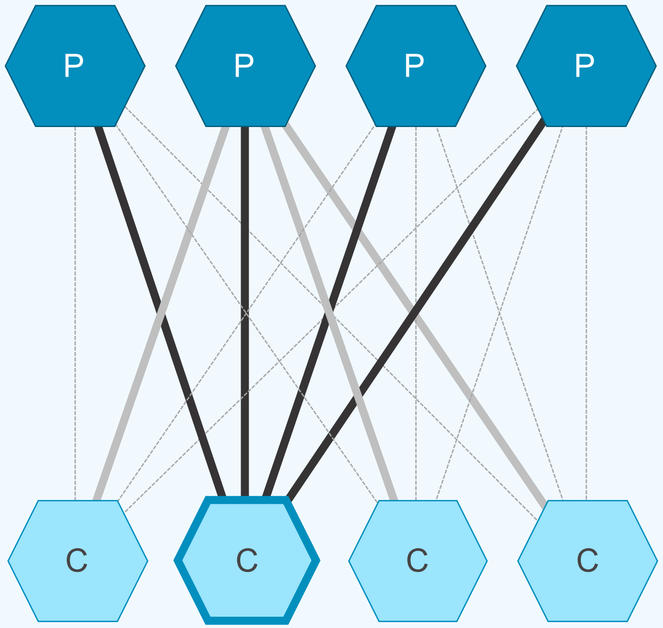
RISK: While more focused and faster, there is a risk of "**unfinished business**" which can have a high cost in the organization (stability, complexity, added maintenance). There is a need for monitoring and following up these issues continuously.

Chapter IV

Data

Accessing Distributed Data

With **smart** products and **data analytics**, how to best **access** distributed data?

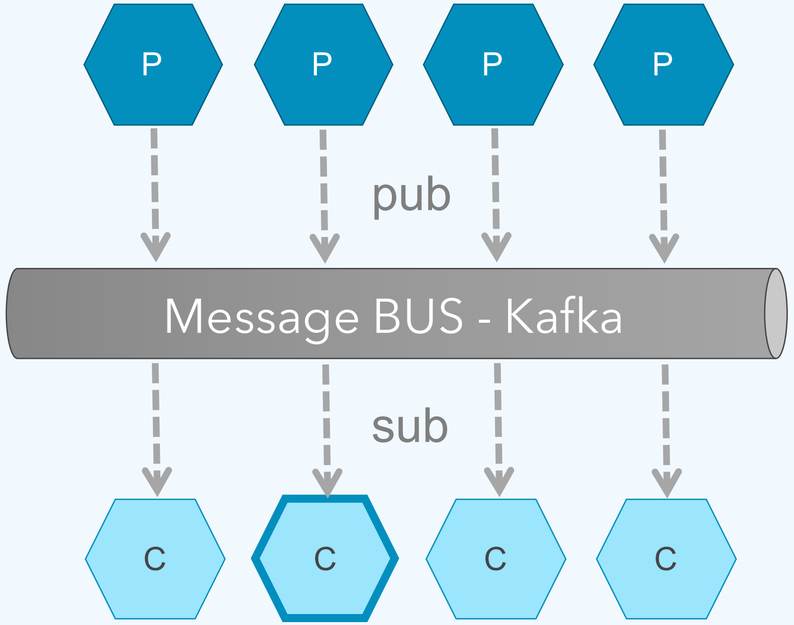


Data & Signals

Share biz truths on "dumb" data pipeline

not "smart" ESB

Publish Domain Events
(not commands)



To avoid **exponential complexity**, **move the complexity to the infrastructure**

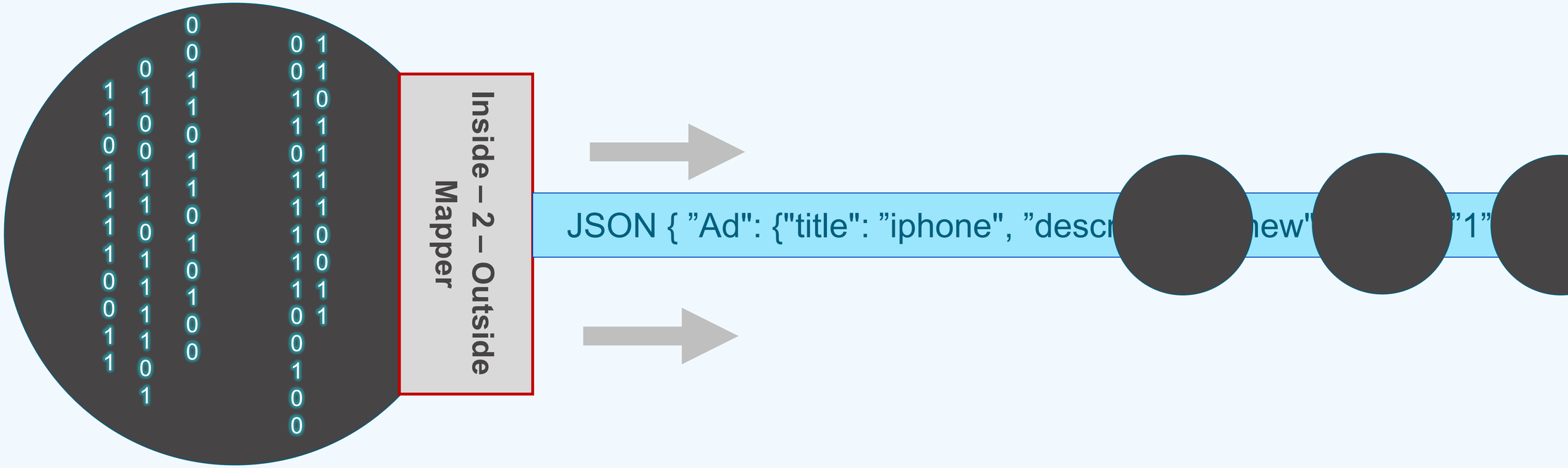
Pipeline Challenges @ FINN

Challenges we experienced at FINN when learning to use a pipeline

- **Reliable messaging** is more than a product feature on the message bus
- Introducing **event based communication is hard**, developers are used to RPC

Data on the Outside versus Data on the Inside

- **Article** written by Pat Helland many years ago…



Continuous development
requires **flexibility** inside

Outside users of data
requires **stable** contracts

Data Integration Recommendations

- Separate **data on the outside** formats from **data on the inside** formats in order to achieve the best of both worlds.
- Start experimenting with a **data pipeline**
- Understand business needs for **transactional boundaries** and when **eventual consistency** works and not.
- **Avoid point-2-point** data integration as key data is likely to be used in many different contexts. Publish once…

Key Takeaways

1. **Partition** by **business objective** **(domains)**, **avoid global entity models**
2. **Organize** by **business objective**, not by competence, technology or process
3. **Take** the **smallest possible step** to reach **strategic** capability, **no total rewrites**
4. **Make your data easily available across the organization** to capitalize on it

But more importantly, it all requires an development organization with **good knowledge** of the **business domain** and a **great culture** for working together **across organizational boundaries**.

The End

Agenda

15.30 –15:45	Velkommen hjem til FINN
15:45 –16:30	Slik jobber vi i FINN
16:30 –17:00	Mat og drikke
17:00 –17:30	Introducing Node.js in an Enterprise
17.30 - 18.00	Beyond "Hello World", handling complexity and organization with large systems
18.00 - 18.15	**Pause**
18:15 - 18.45	Maskinlæring, anbefalingsalgoritmer og datadrevne produkter
18:45 –19:15	Hvordan vi flytter FINN.no ut i skyen
19:15 – 21:00	*Kahoot, drikke og snacks!*