ThoughtWorks®

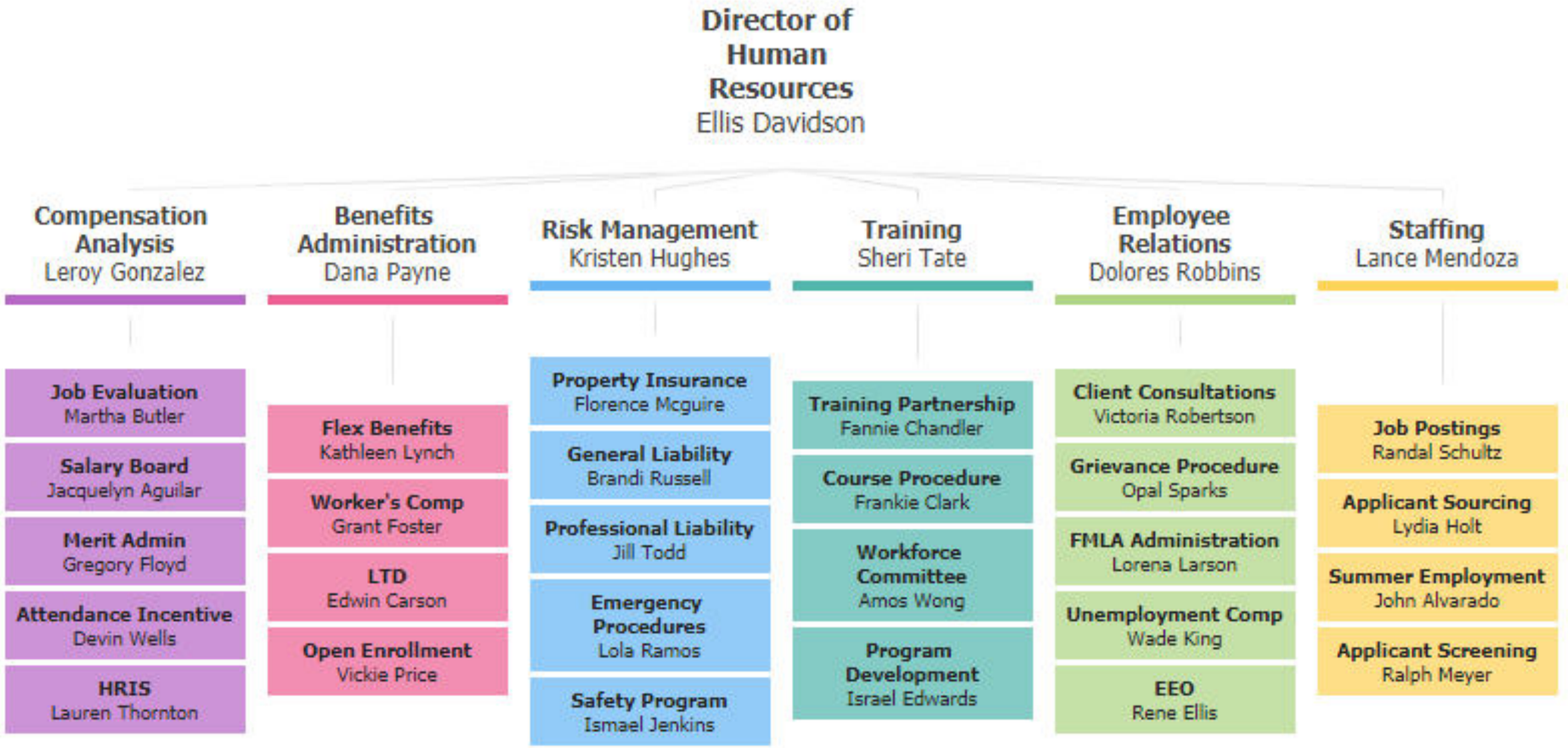
构建可以发挥平台效能的组织

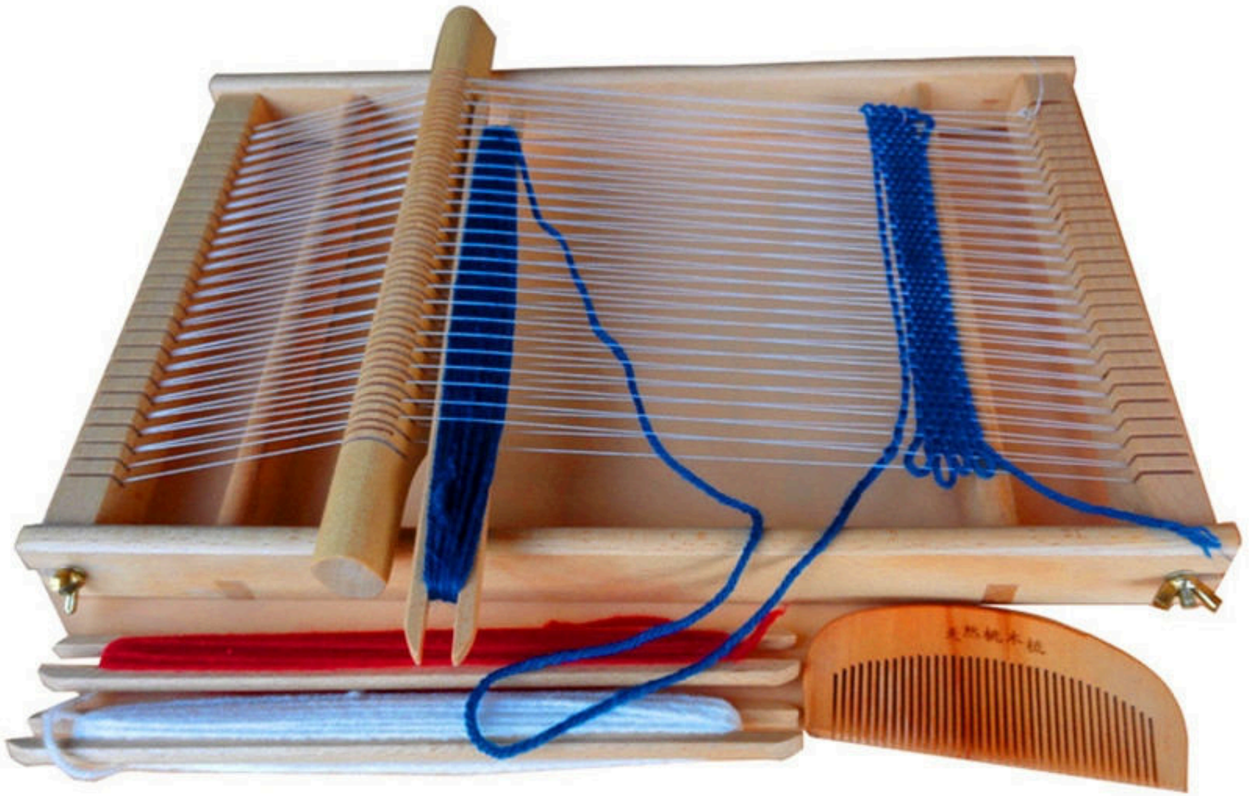
徐昊　ThoughtWorks 中国区CTO

01

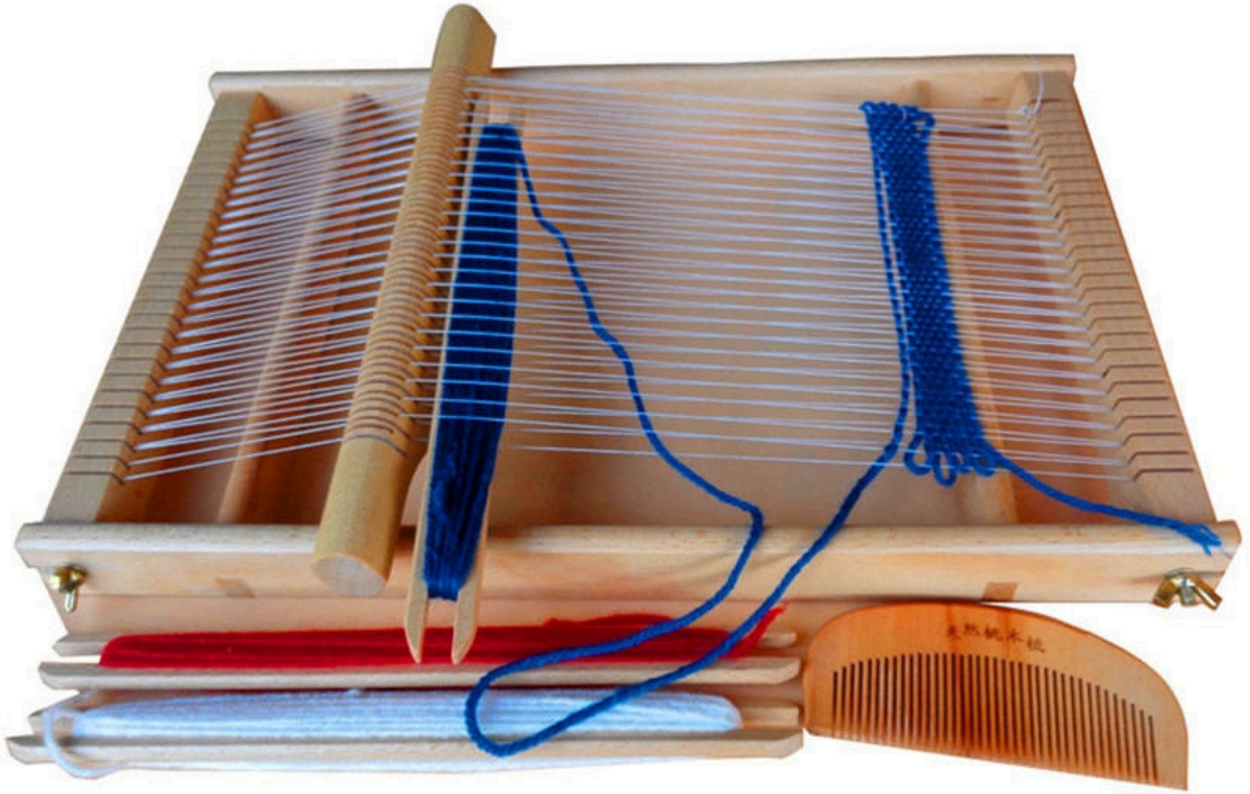
什么是组织?

ThoughtWorks®

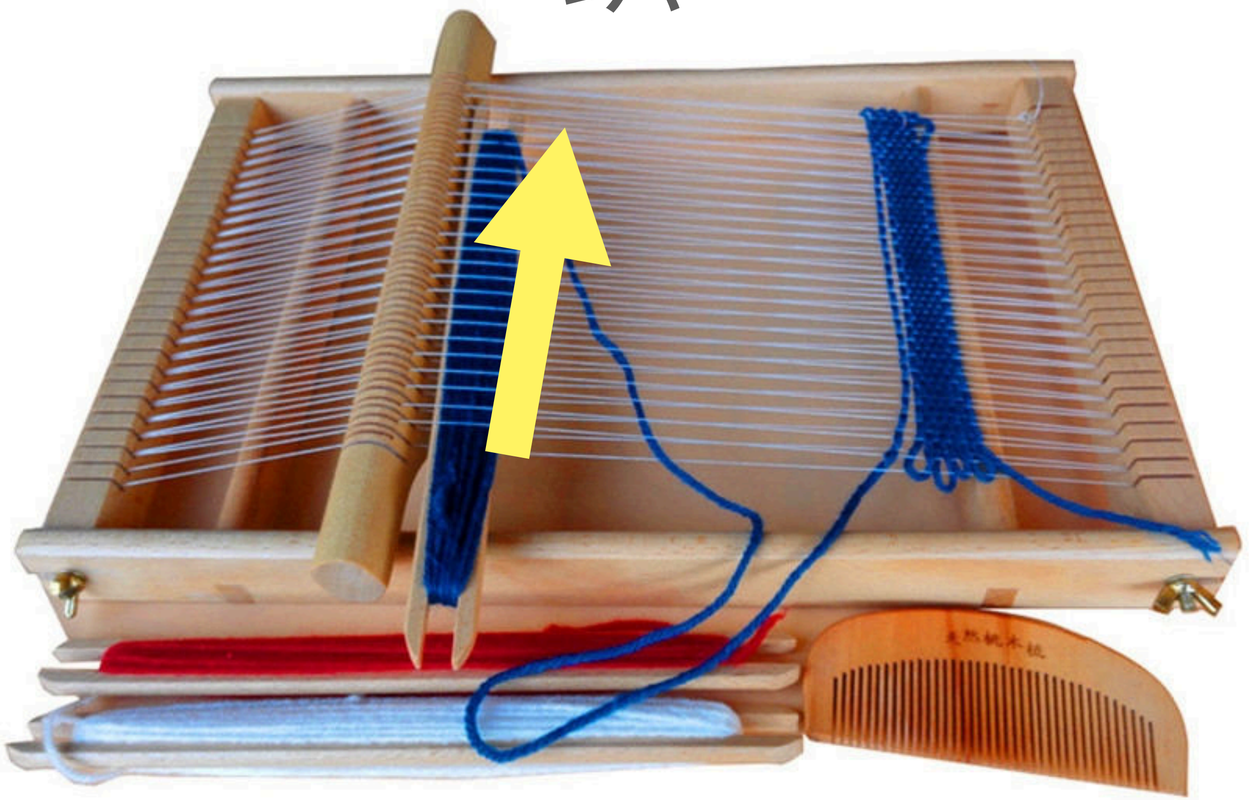


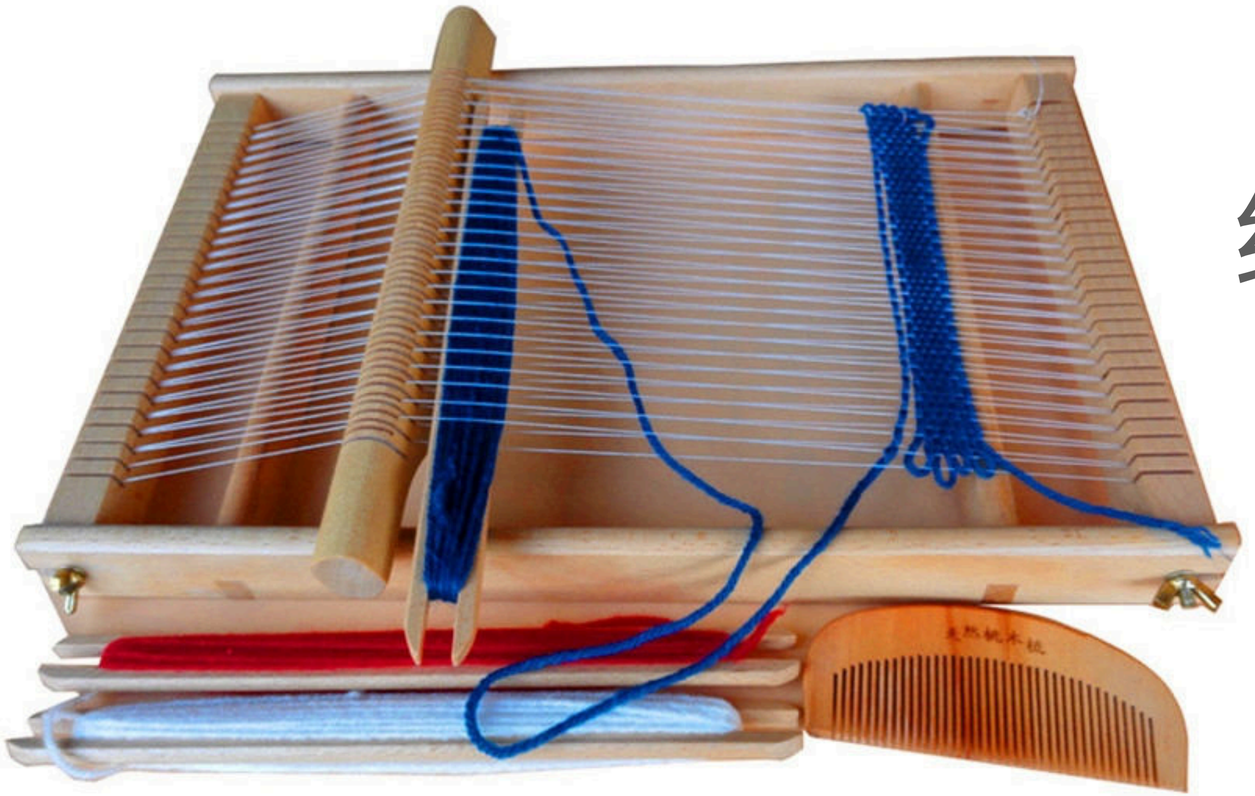


组



织

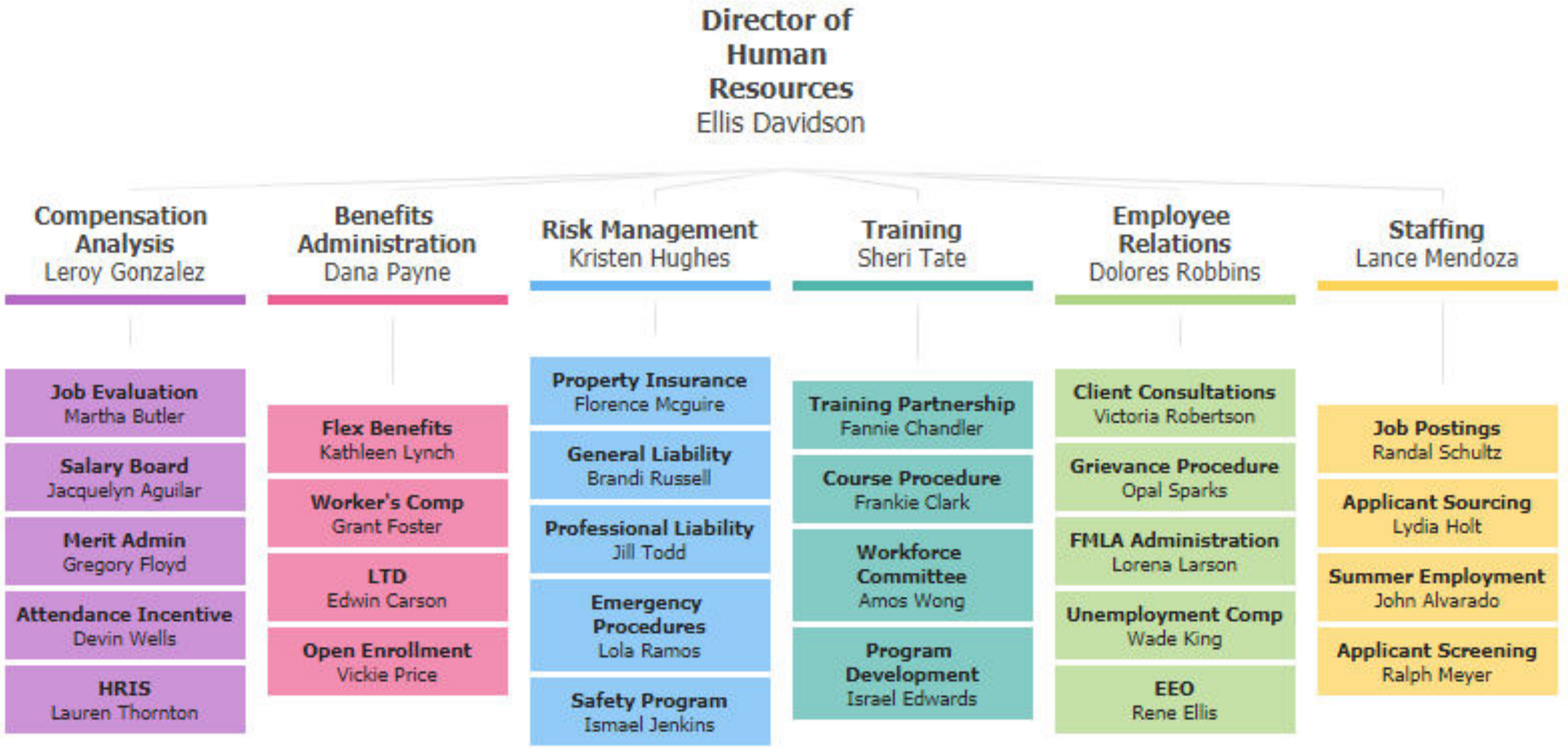




统

组织是围绕着系统的交互

ThoughtWorks®



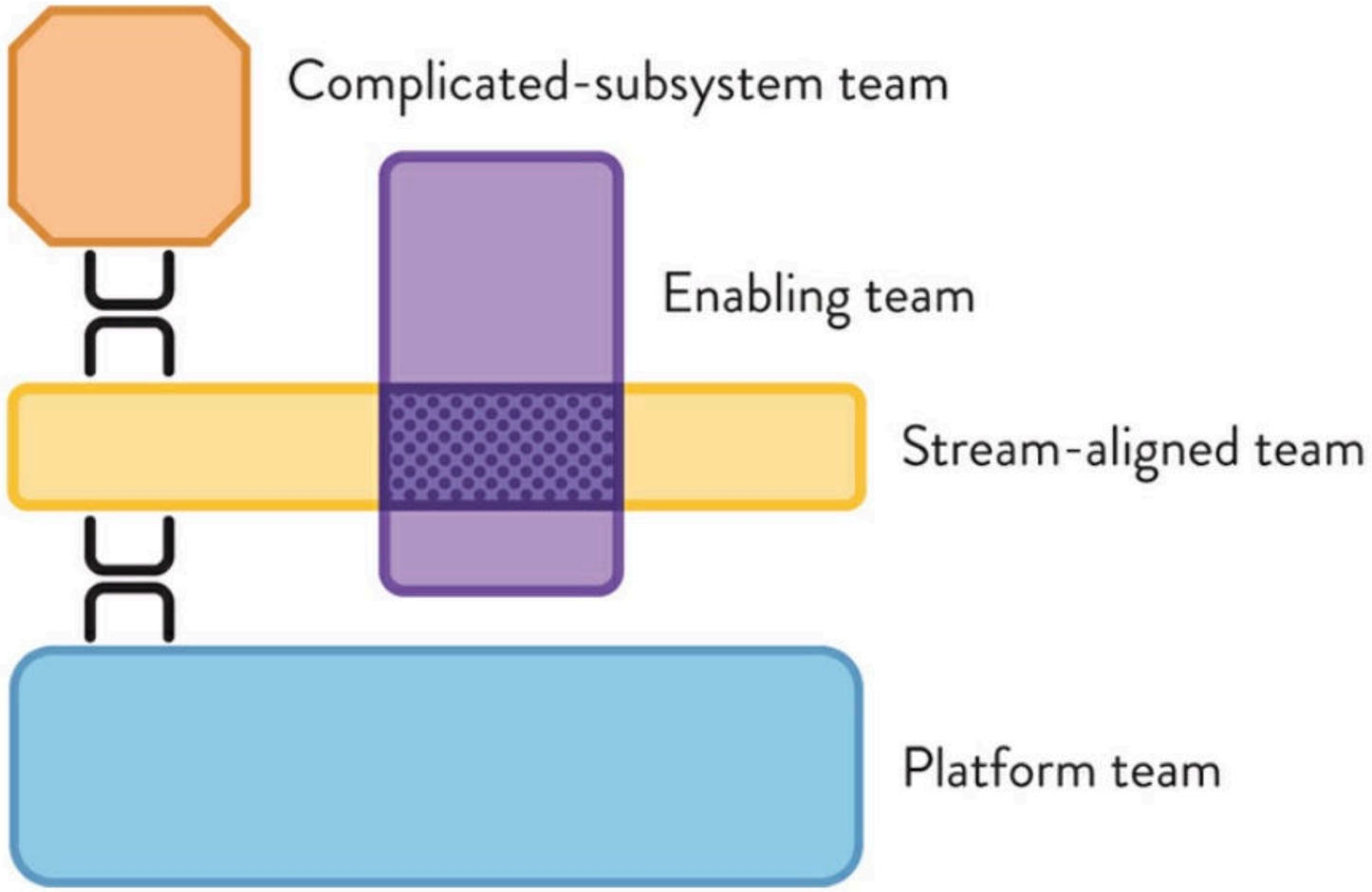
当我们谈及组织，通常会关注组（结构）
而不是织（交互），且忽略系统

ThoughtWorks®

02

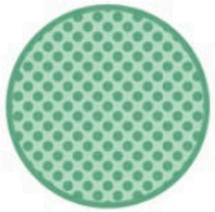
如何关注系统与交互的谈论组织?

ThoughtWorks®


Complicated-subsystem team
Enabling team
Stream-aligned team
Platform team

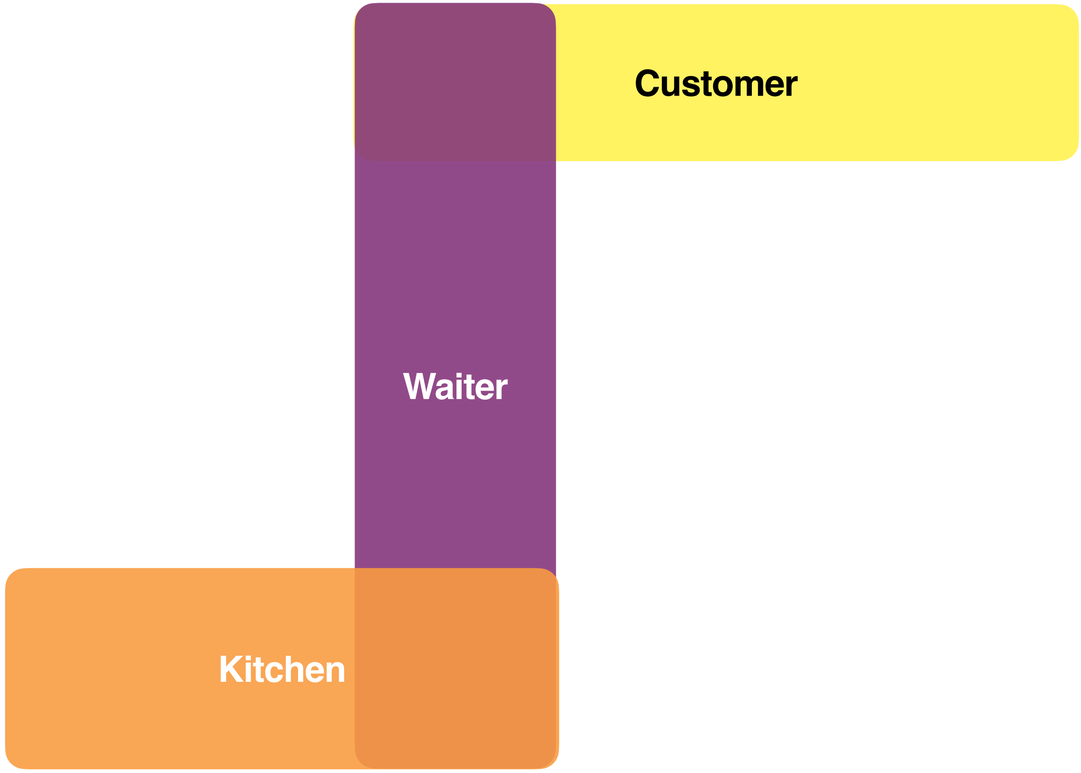
Collaboration

X-as-a-Service

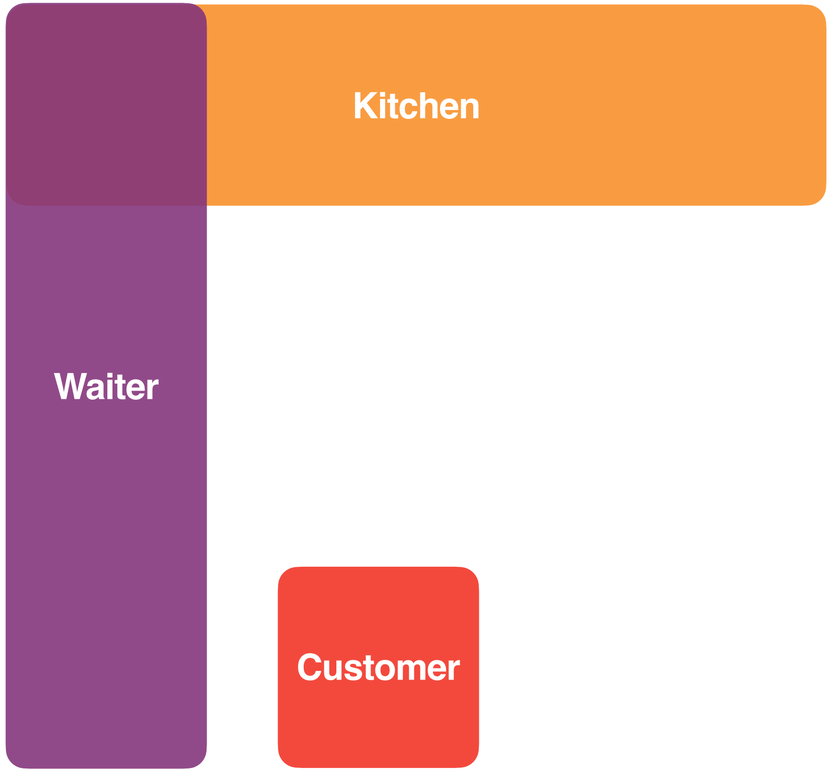


Facilitating

ThoughtWorks®



ThoughtWorks®



03

什么样的组织可以发挥平台效能?

ThoughtWorks®

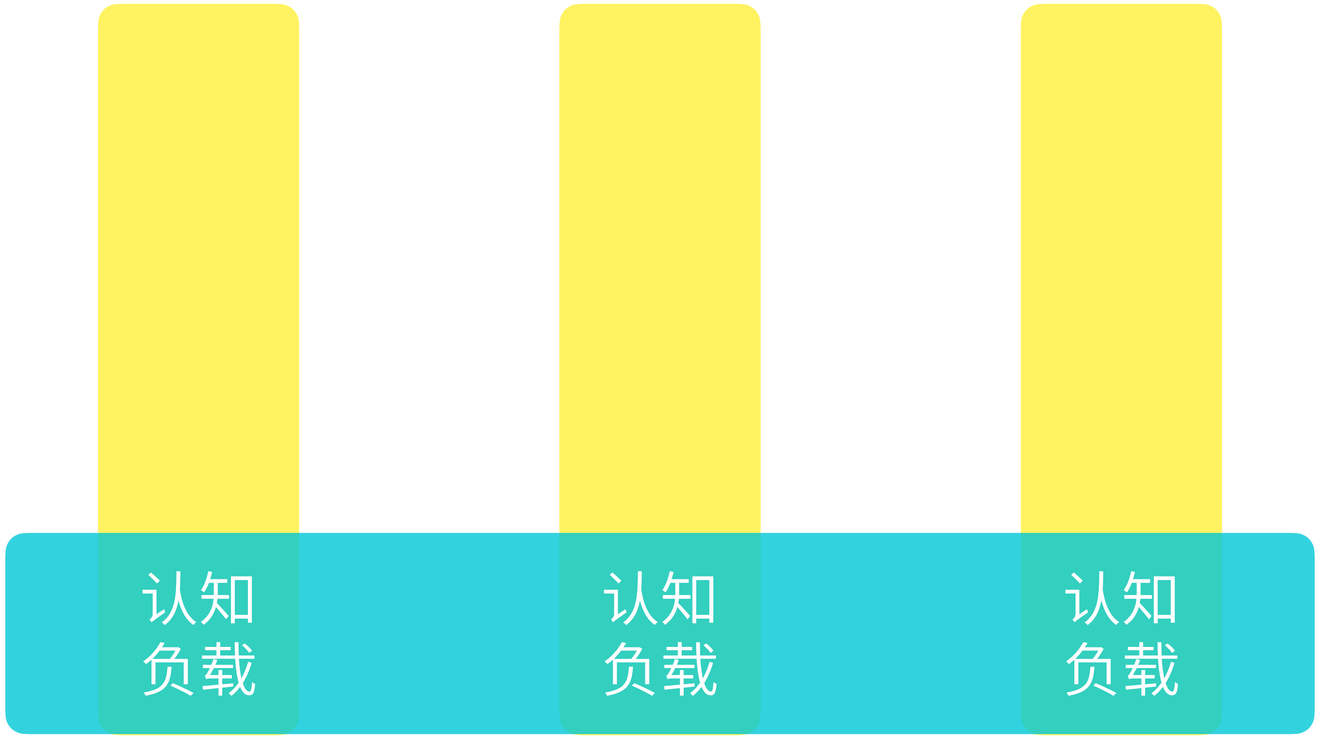

As a service

As a service

ThoughtWorks®

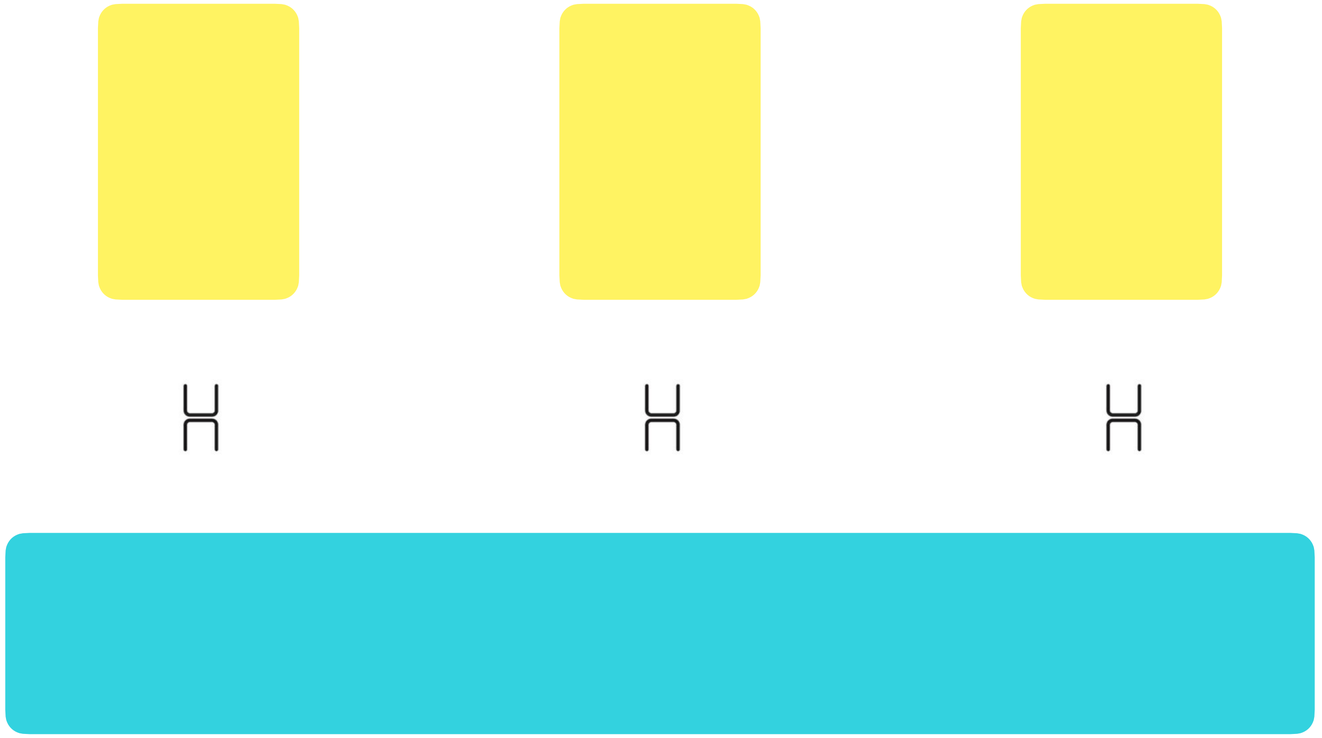
As a service

ThoughtWorks®



As a service

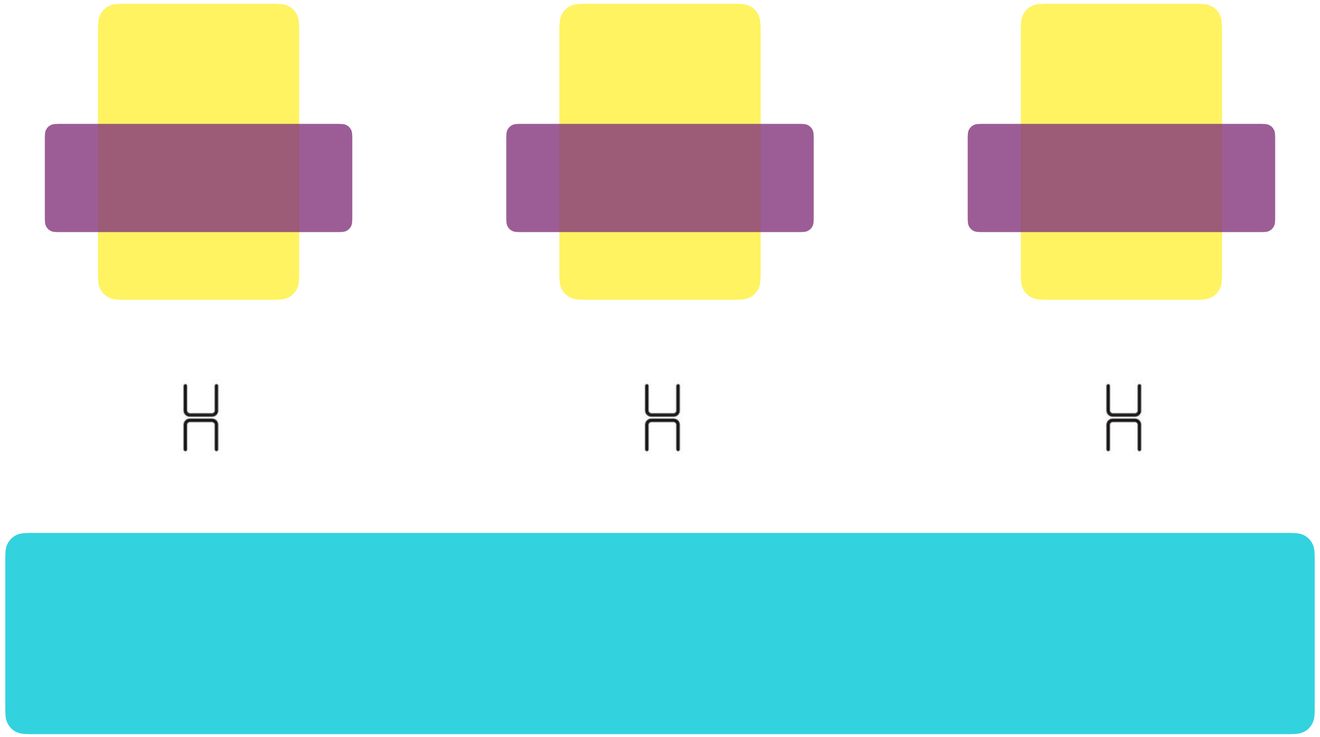
ThoughtWorks®




Enabling Team

As a service

ThoughtWorks®



As a service

ThoughtWorks®

THANK YOU

ThoughtWorks®

ThoughtWorks®

Good Coupling Bad Coupling

刘尚奇

ThoughtWorks 中国区区块链业务负责人

ThoughtWorks®

Good Coupling Bad Coupling

刘尚奇

~~ThoughtWorks 中国区区块链业务负责人~~
软件架构师，致力于去中心化一切

ThoughtWorks®

识别架构耦合上下文

What the Radar says

在软件架构中，如何在微服务、组件、API网关、集成中心、前端等等之间确定一个适当的耦合级别，是几乎每次会议都会讨论的话题。随处可见的情况是，当两个软件需要连接在一起时，架构师和开发人员都在努力地寻找正确的耦合级别。我们需要花时间和精力去理解这些因素，然后因地制宜地做出这些决定，而不是寄希望于找到一个通用却并不恰当的解决方案。

Why does it matter to you

“耦合”是软件系统的一部分与另一部分进行通信的需要。 取决于所使用的体系结构，耦合可以是松散的或紧密的。 耦合影响可能是非常关键的元素，团队可能会陷入分析瘫痪。企业需要与技术团队合作，以避免“完美是善的敌人”。

当架构师在讨论架构时，我们在讨论什么

ThoughtWorks®

功能分析

(画框框)

技术选型

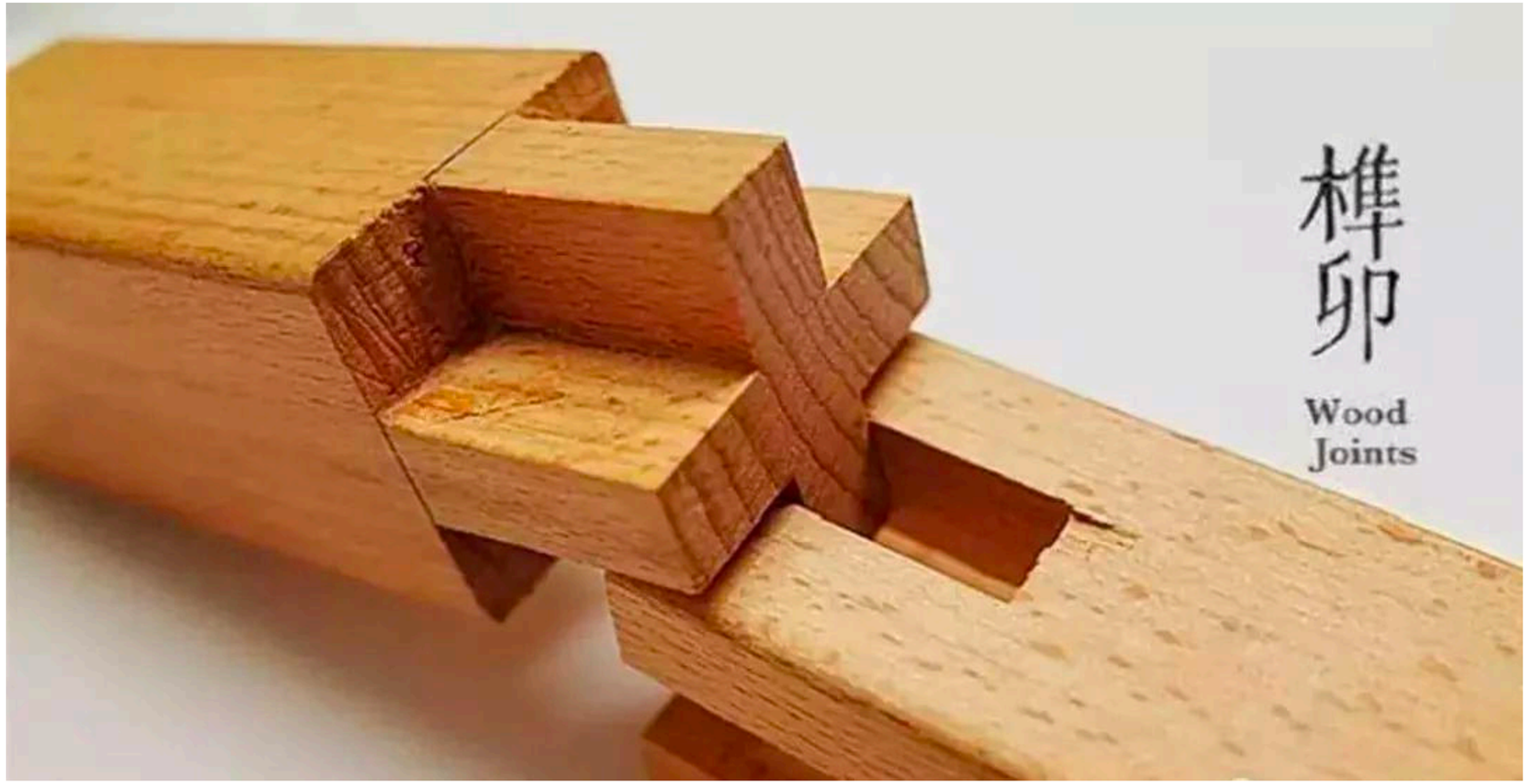
(挑工具)

架构设计

(高内聚/低耦合)

架构耦合是必要的…

ThoughtWorks®

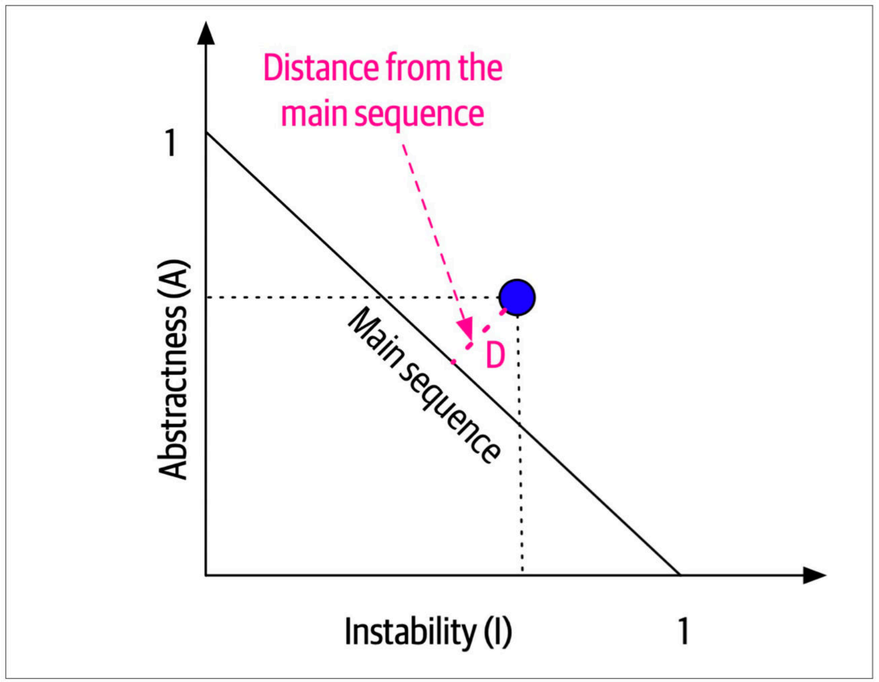


架构耦合是必要的…恶

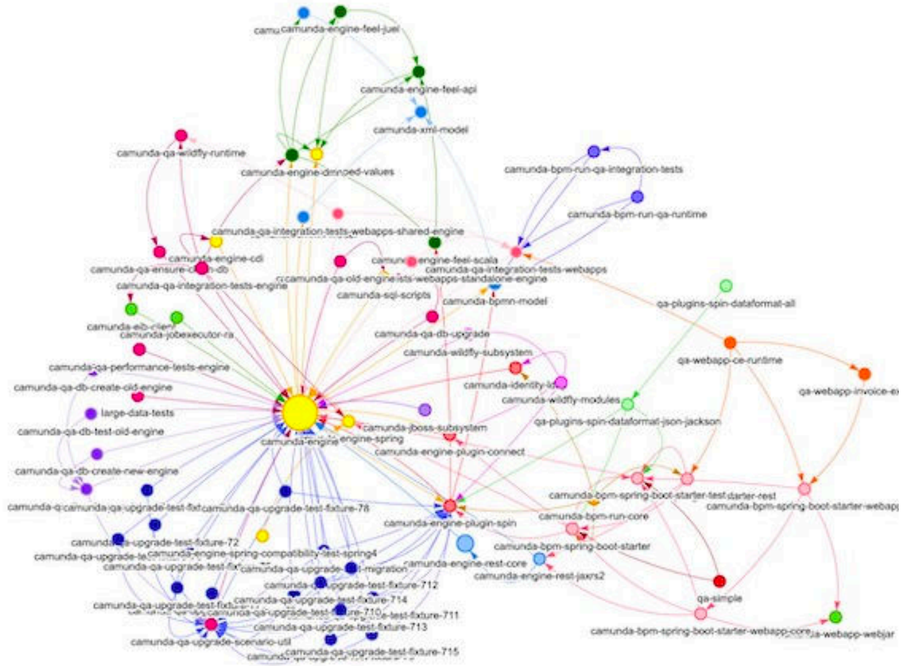
ThoughtWorks®

架构耦合可以被定量分析…但更多的是艺术

ThoughtWorks®



抽象程度: 接口数量/组件数量
不稳定性： 软件的变化速度



架构聚类分析建模

解耦最佳实践(1):

“我们总是可以通过微服务拆分，
从而获得松耦合架构”

ThoughtWorks®

服务拆分是行业趋势…

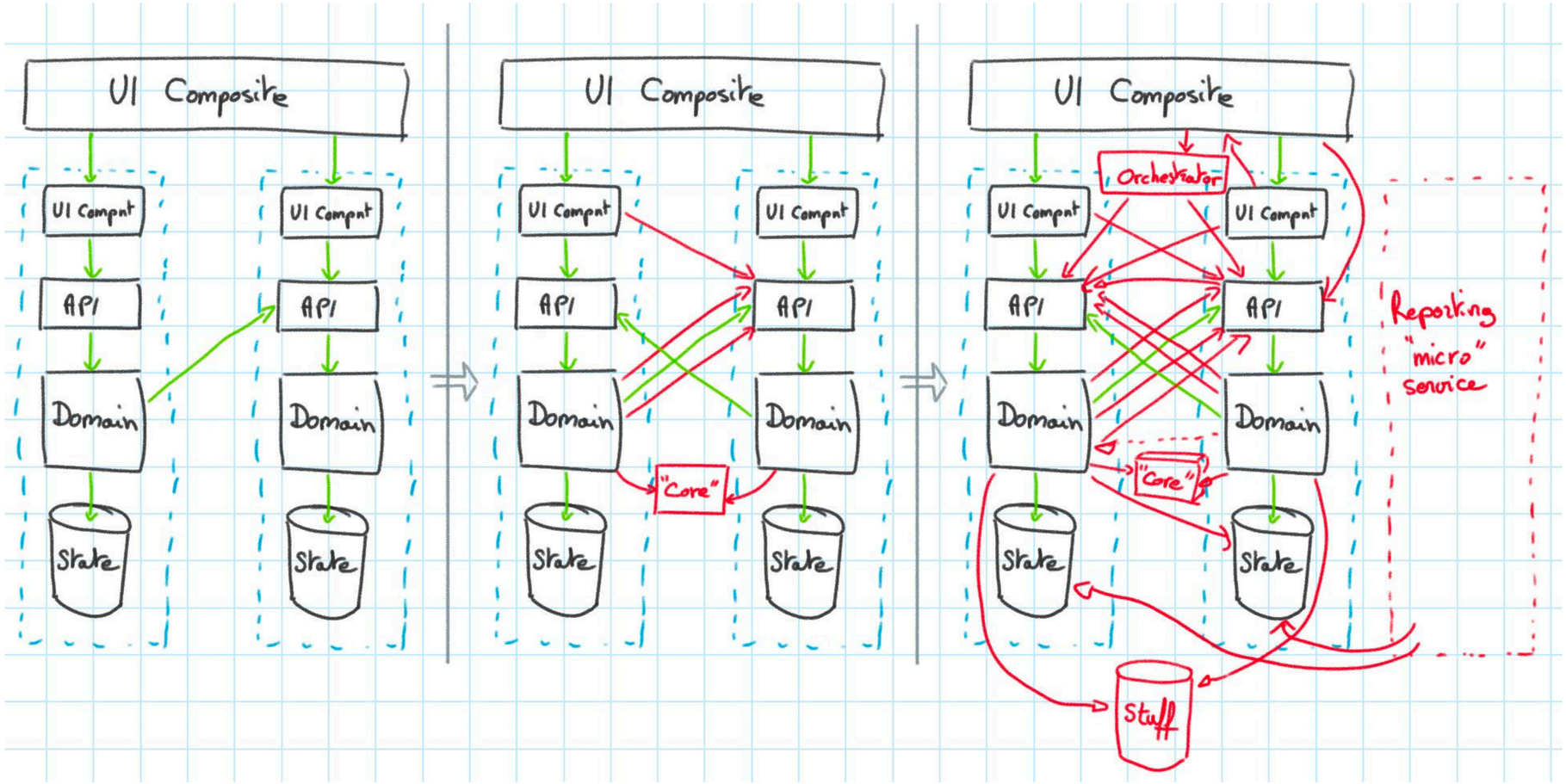
ThoughtWorks®

1990s and earlier	2000s	2010s
Coupling		
Pre-SOA (monolithic) Tight coupling	Traditional SOA Looser coupling	Microservices Decoupled

http://www.pwc.com/us/en/technology-forecast/2014/cloud-computing/features/microservices.html

服务拆分是行业趋势…也催生了分布式单体的悲剧

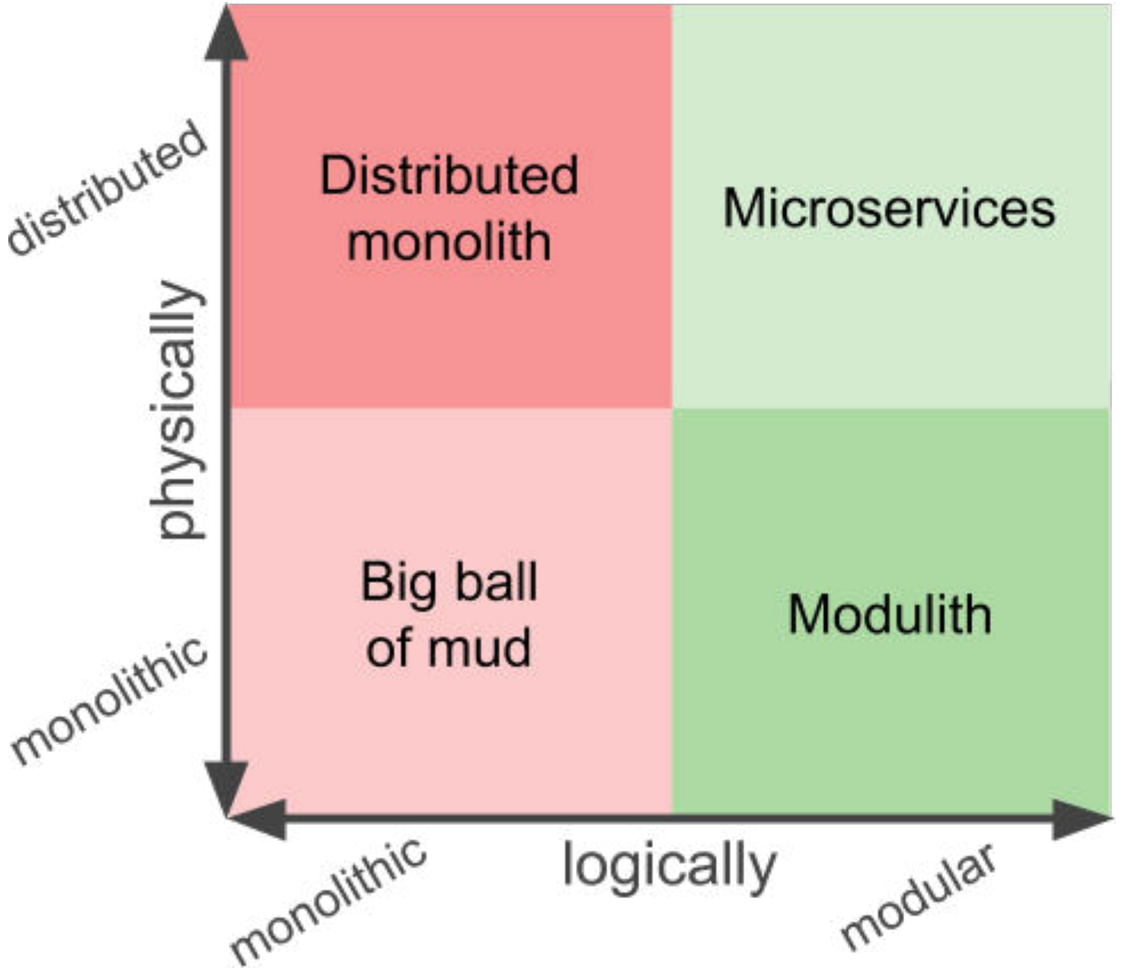
ThoughtWorks®



Charles Feval on Twitter: when not to use #microservices - avoid the distributed monolith

Good Coupling vs Bad Coupling

ThoughtWorks®



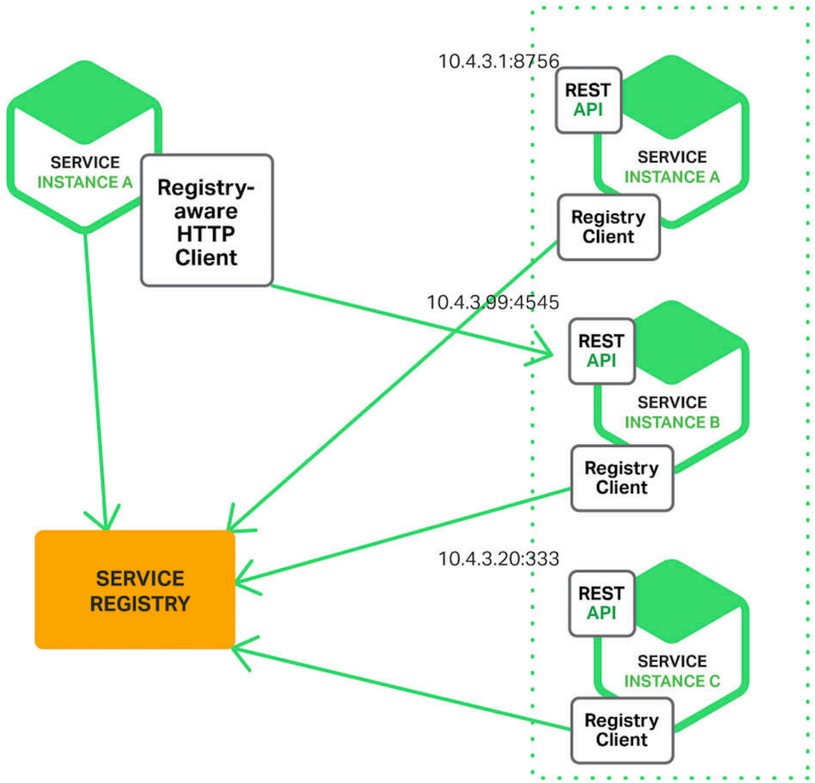
解耦最佳实践(2):

“我们总是可以构造一个中间层，
从而消除组件间的耦合”

ThoughtWorks®

引入中间层，可以消除服务间对ip地址的依赖…

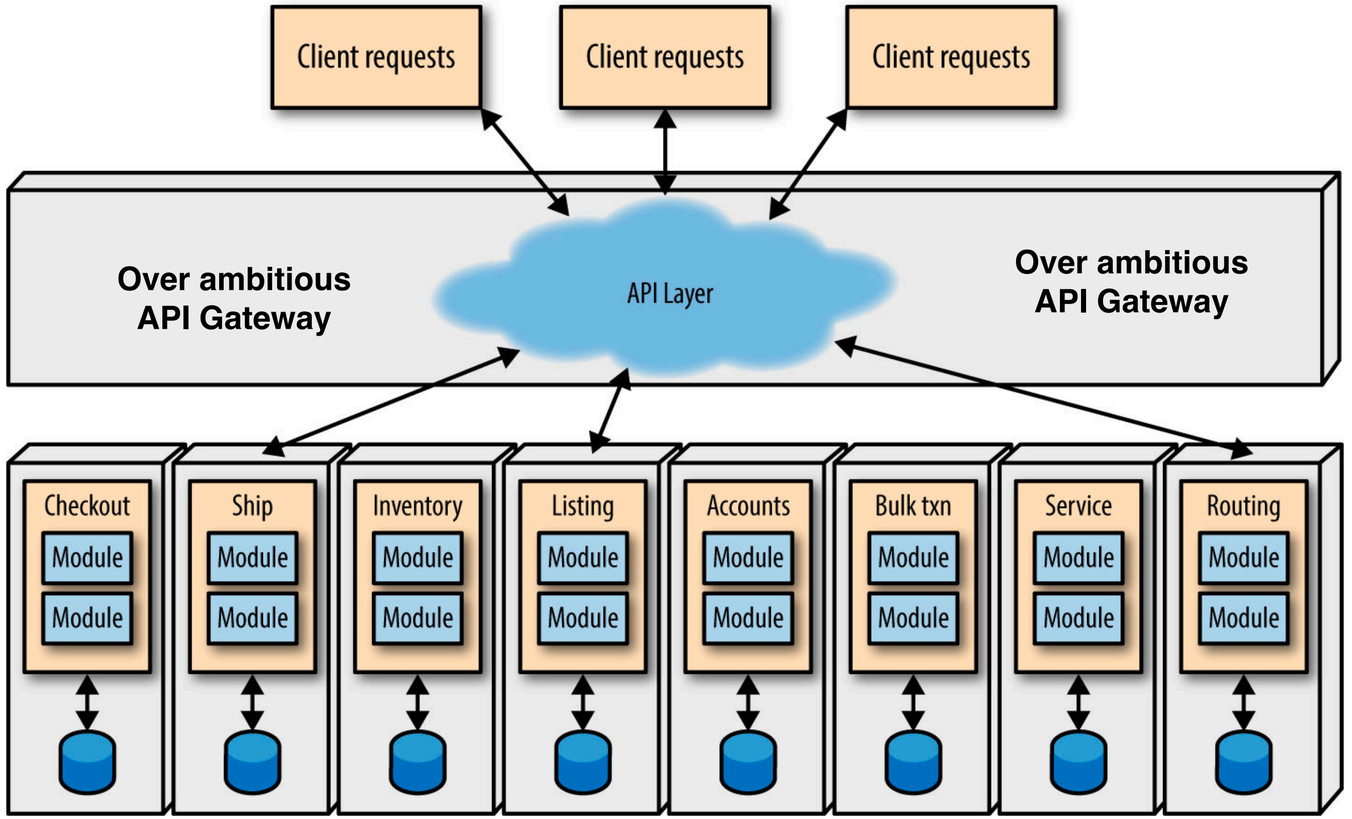
ThoughtWorks®



Service Discovery

…也可以连续霸榜技术雷达，成为最不欢迎技术

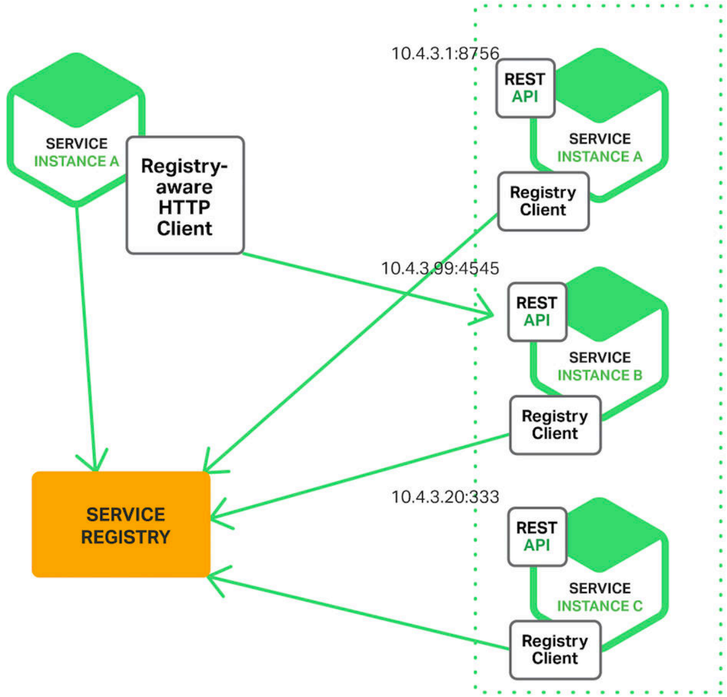
ThoughtWorks®



Over ambitious
API Gateway

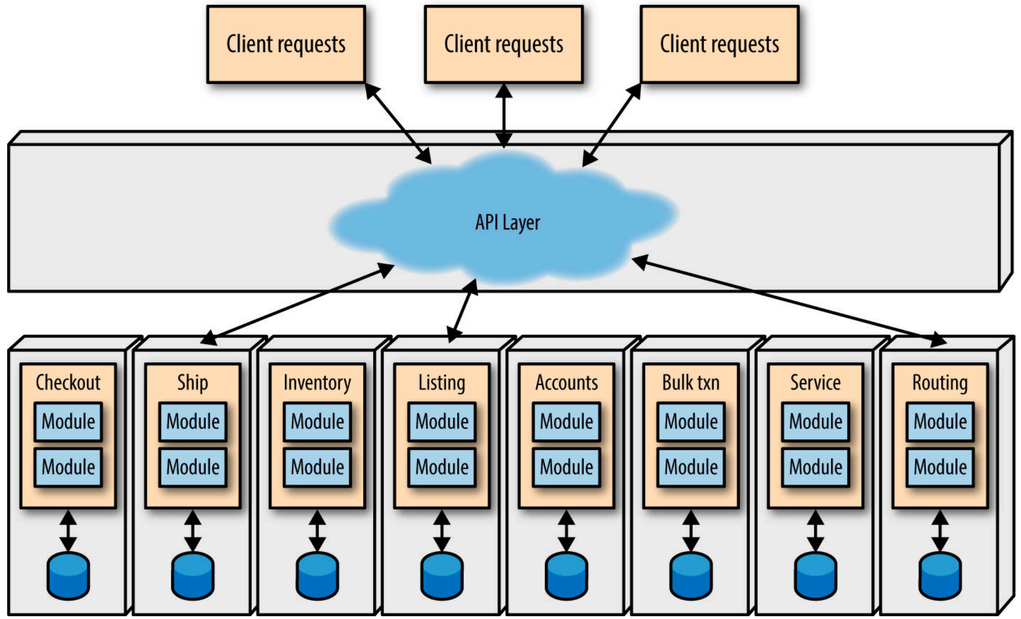
Good Coupling vs Bad Coupling

ThoughtWorks®



Service Discovery

VS



Over ambitious API Gateway

解耦最佳实践(3):

“把接口发布成IDL是邪恶的，
我们应该尽可能使用异构REST API”

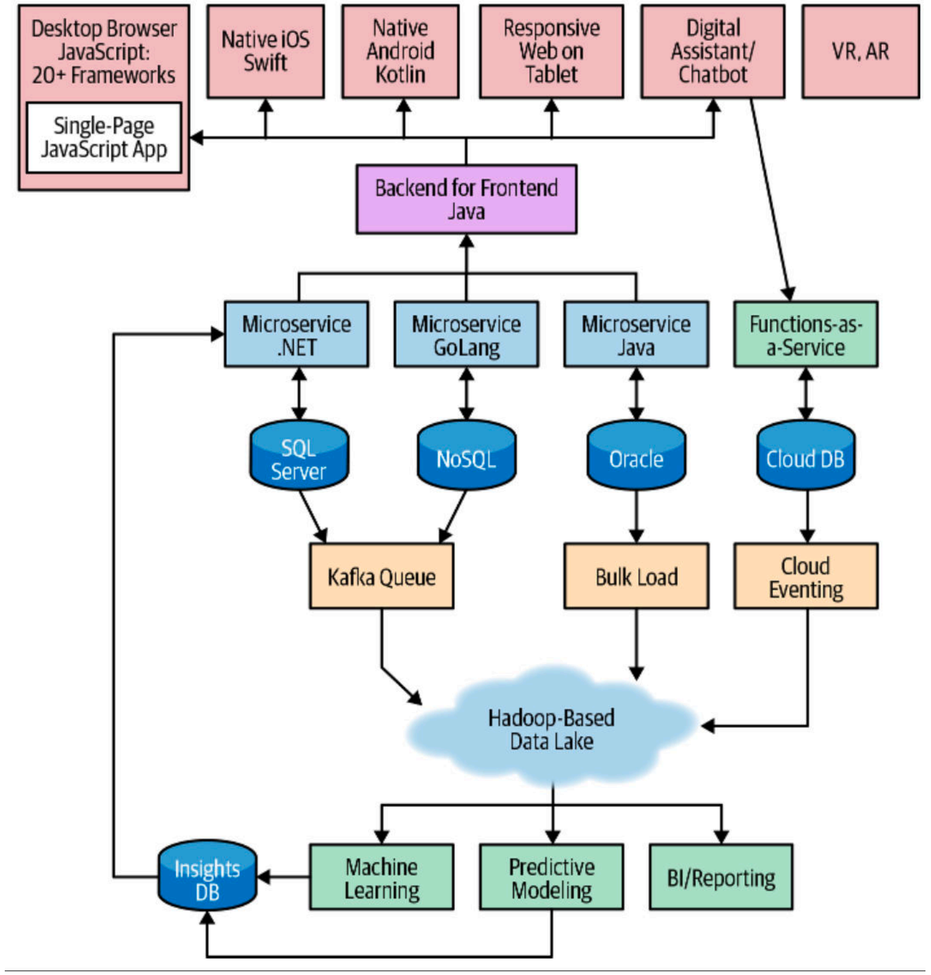
ThoughtWorks®

我们在IDL吃了太多亏…最好通过多语言编程来实现多样性

ThoughtWorks®

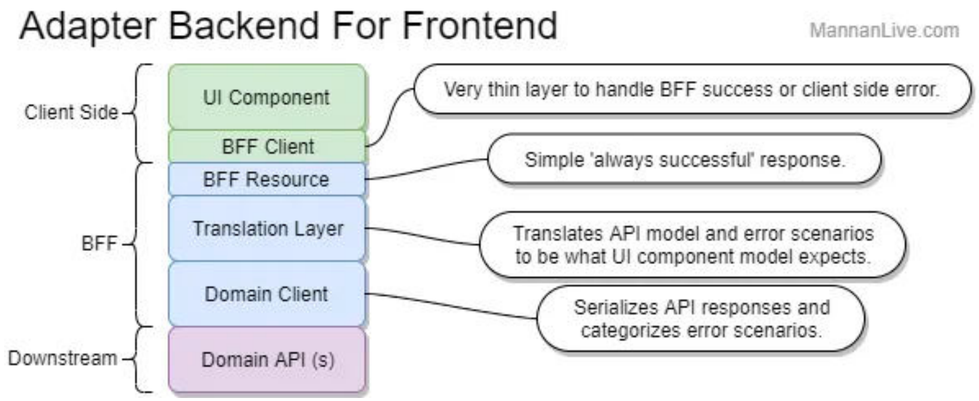
```
<?xml version="1.0" encoding=
<definitions name="AktienKurs
  targetNamespace="http://loc
  xmlns:xsd="http://schemas.xmlsoap.or
  xmlns="http://schemas.xmlsoap.org/wsd
  <service name="AktienKurs">
    <port name="AktienSoapPort" binding
      <soap:address location="http://loc
    </port>
    <message name="Aktie.HoleWert">
      <part name="body" element="xsd:Tra
    </message>
    …
  </service>
</definitions>
```

WSDL



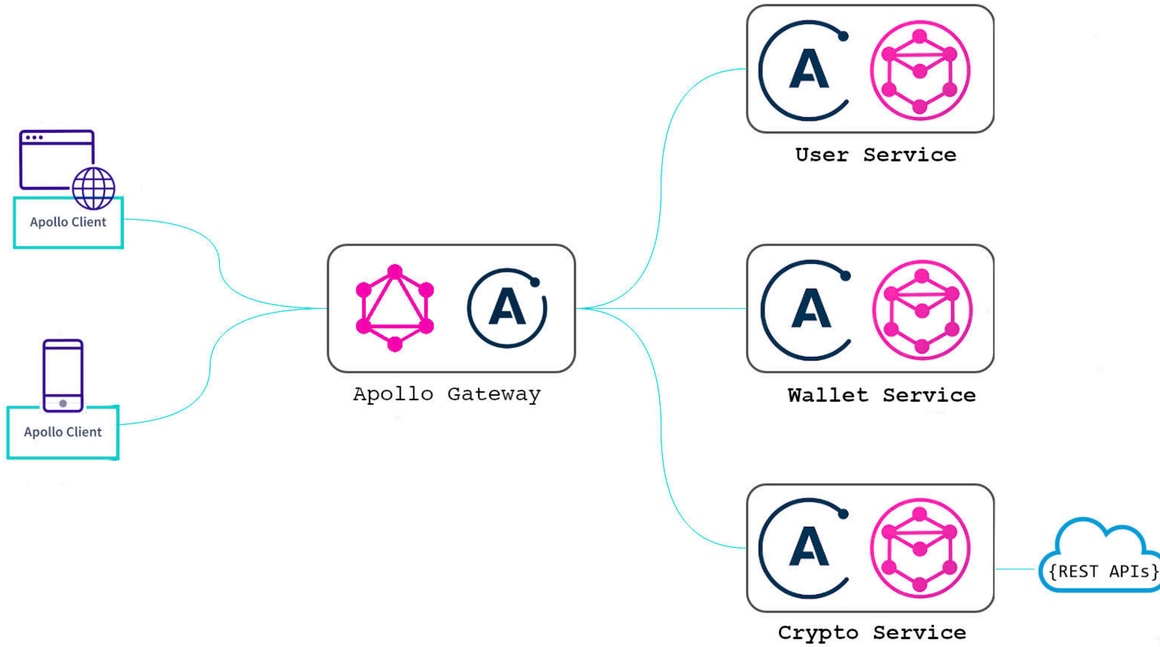
Good Coupling vs Bad Coupling

ThoughtWorks®



UI/BFF shared types

VS



Apollo Federation

没有最佳实践！

ThoughtWorks®

~~没有最佳实践！~~

但经验和原则是有用的

ThoughtWorks®

什么时候寻求架构解耦?

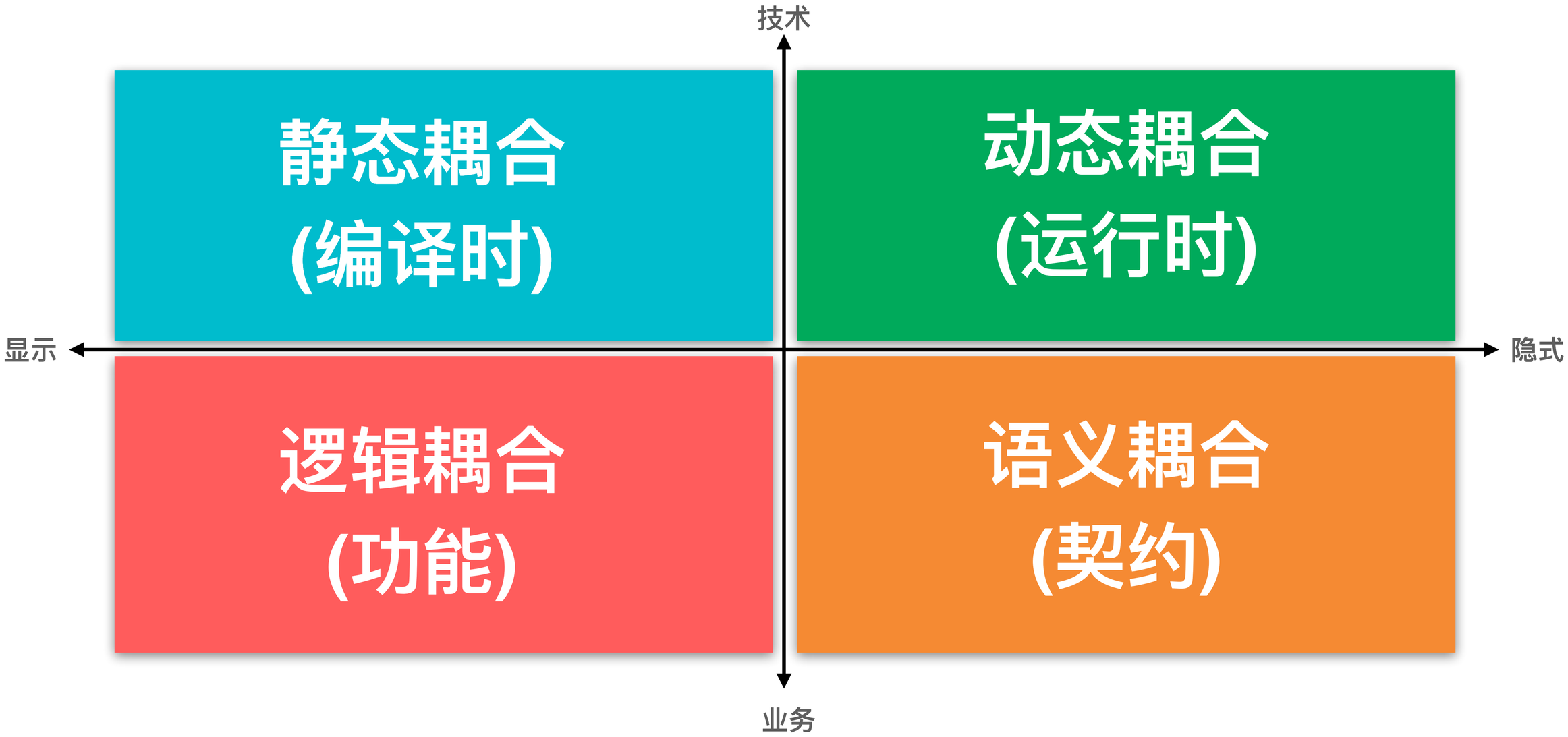
ThoughtWorks®

什么时候耦合是必要的?

ThoughtWorks®

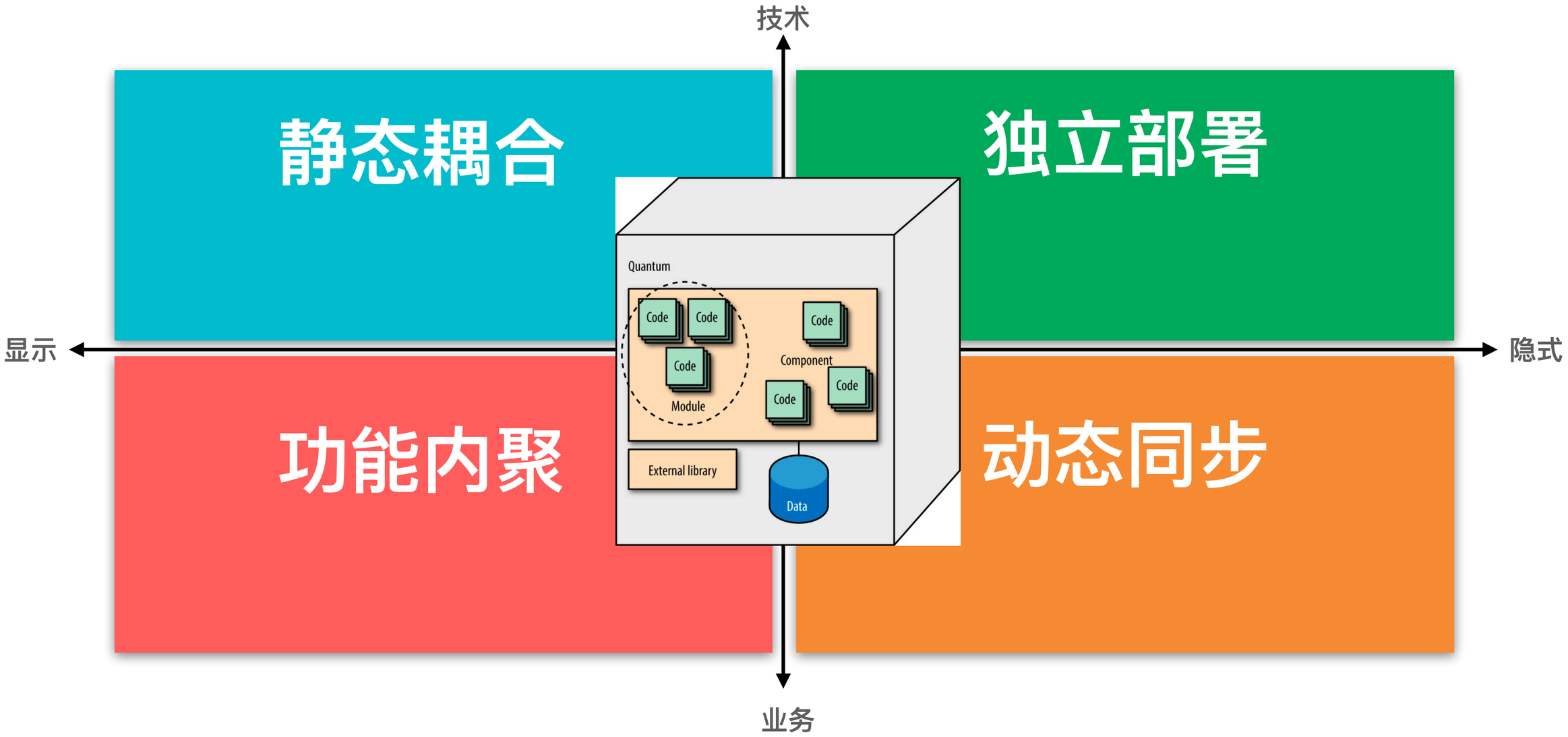
识别耦合的边界上下文

ThoughtWorks®



识别耦合的边界上下文：架构量子作为最小组合单元

ThoughtWorks®



ThoughtWorks®

Good Coupling
Bad Coupling

THANK YOU

欲了解详情，请发送邮件至：
sqliu@thoughtworks.com

ThoughtWorks®

ThoughtWorks®

重新激活你的遗留资产

梅雪松

ThoughtWorks 首席咨询师

太复杂以至于无法进入雷达

What the Radar says

在技术雷达的命名法则里，对许多复杂主题的讨论，最终状态都会是“TCTB—太复杂以至于无法进入雷达(too complex to blip) ”，这意味着那些条目无法被分类，因为它们呈现出太多正反两面的特性，这让我们无法用短短几句话总结它们。通常来说，这些主题会变成文章, 播客, 以及其他非雷达形式的内容。

许多主题经历了一次又一次会议，最终进入了TCTB 的状态，包括Monorepos、分布式架构的编排指南以及分支模型等等。如果你好奇为什么这些重要的主题没有进入雷达,可以确信的是，这并不是因为我们缺乏意识或者意愿。就像软件开发中的许多主题一样，那里存在太多权衡，难以提供清晰明确的建议。我们有时也会发现，可以对一些大的主题中的小部分提供建议，使其进入雷达，但这些大的主题对雷达来说，仍然过于微妙难以确认。

Why does it matter to you

对于遗留系统现代化这样一个主题，同样有些话题呈现正反两面的特性。例如在谈到遗留系统改造时，微服务似乎是一个不错的选项。但是我们也发现在一些金融行业的核心系统改造时，微服务并不是一个高优先级的选项。降低团队对业务和系统的认知负担、建立领域模型、模块解耦和DB解耦反而是更加迫切的事情。

目录

ThoughtWorks®

01

如何启动

ThoughtWorks®

如何启动

ThoughtWorks®

1

为什么要对遗留系统进行现代化改造

IT遗留系统的隐秘角落

自2010年以来，全世界的公司和政府在IT产品和服务上的支出估计为35万亿美元。其中，约四分之三用于运营和维护现有的IT系统。 至少有2.5万亿美元用于尝试替换旧的IT系统，其中约有7,200亿美元被浪费在失败的替换工作上。

https://spectrum.ieee.org/computing/it/inside-hidden-world-legacy-it-systems

Software is Never Done

与硬件的“硬”不同，软件的“软”体现在它可以持续地演进，增加功能、改进性能、调整结构等等。因此，软件在上线的那一刻只是诞生，从未真正完成，直到它的生命周期结束。

Tech@Core

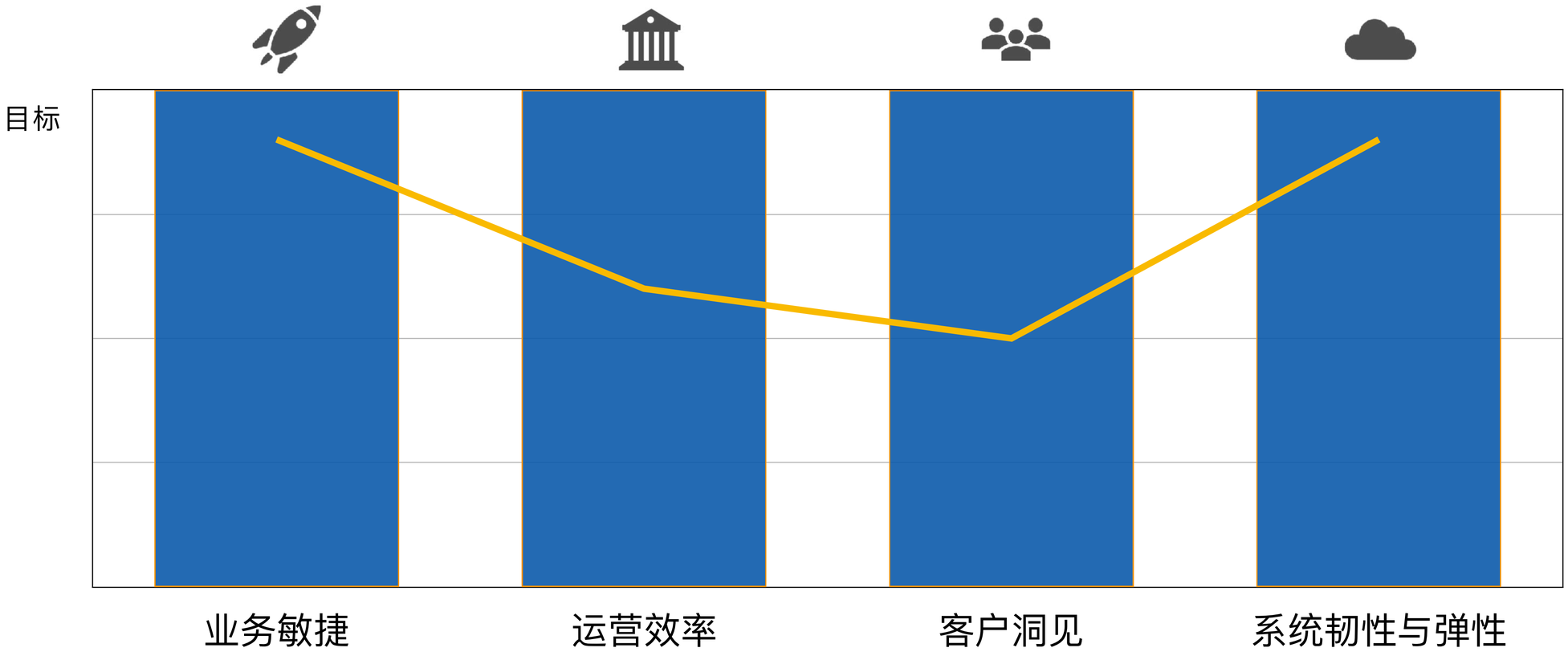
在数字化时代，IT不再被认为是成本中心，因为技术正在重塑创造商业机会的方法，创造新的商业模式，从而创新数字化收益。今天每家企业从根本上说都是一种以技术为核心的新型数字企业。

如何启动

ThoughtWorks®

2

明确系统改造的目标



假设驱动的遗留系统改造（试验）

如何启动

ThoughtWorks®

3

制定度量指标，可视化度量结果

02

如何降低认知复杂度

ThoughtWorks®

如何降低认知复杂度

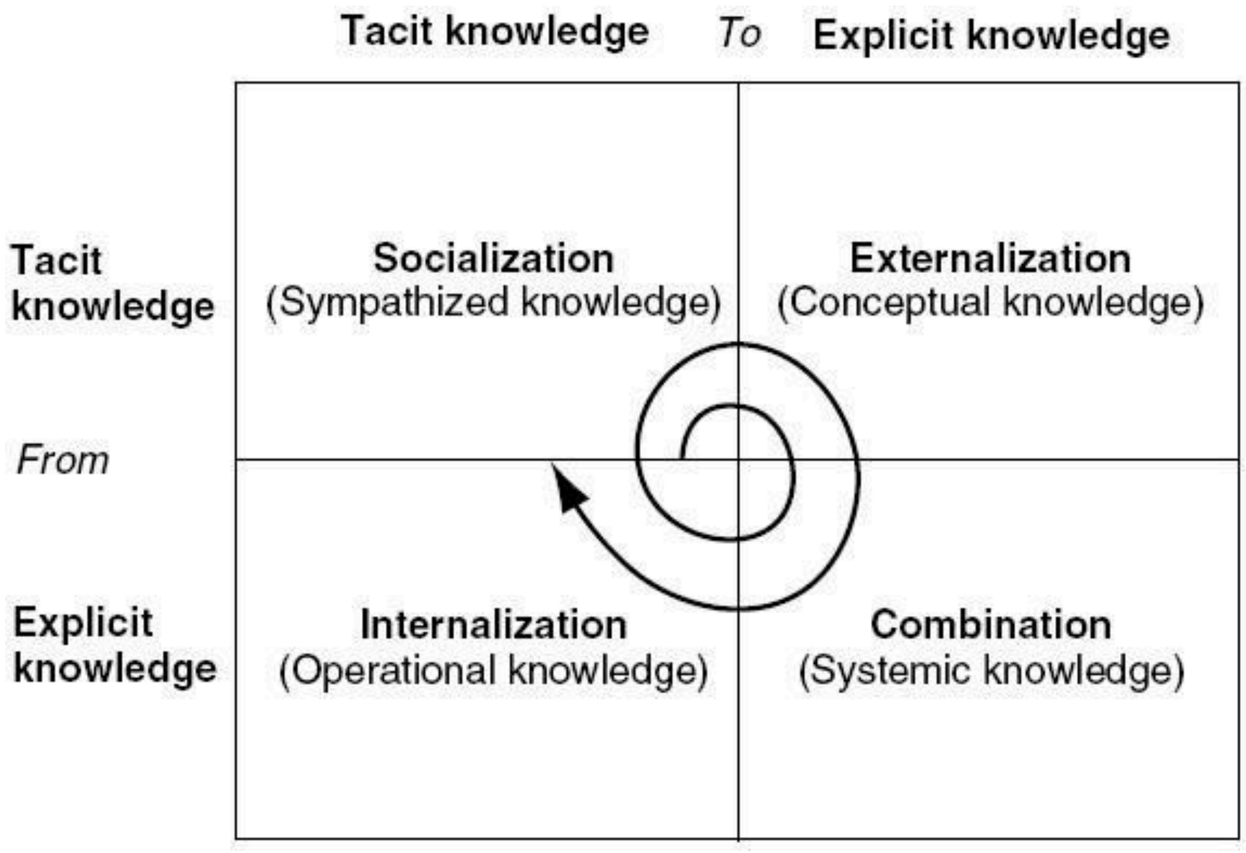
ThoughtWorks®

1 业务认知与系统认知复杂度交织一起，是系统现代化改造的最大阻碍之一

团队的认知负担（试验）

知识螺旋

ThoughtWorks®



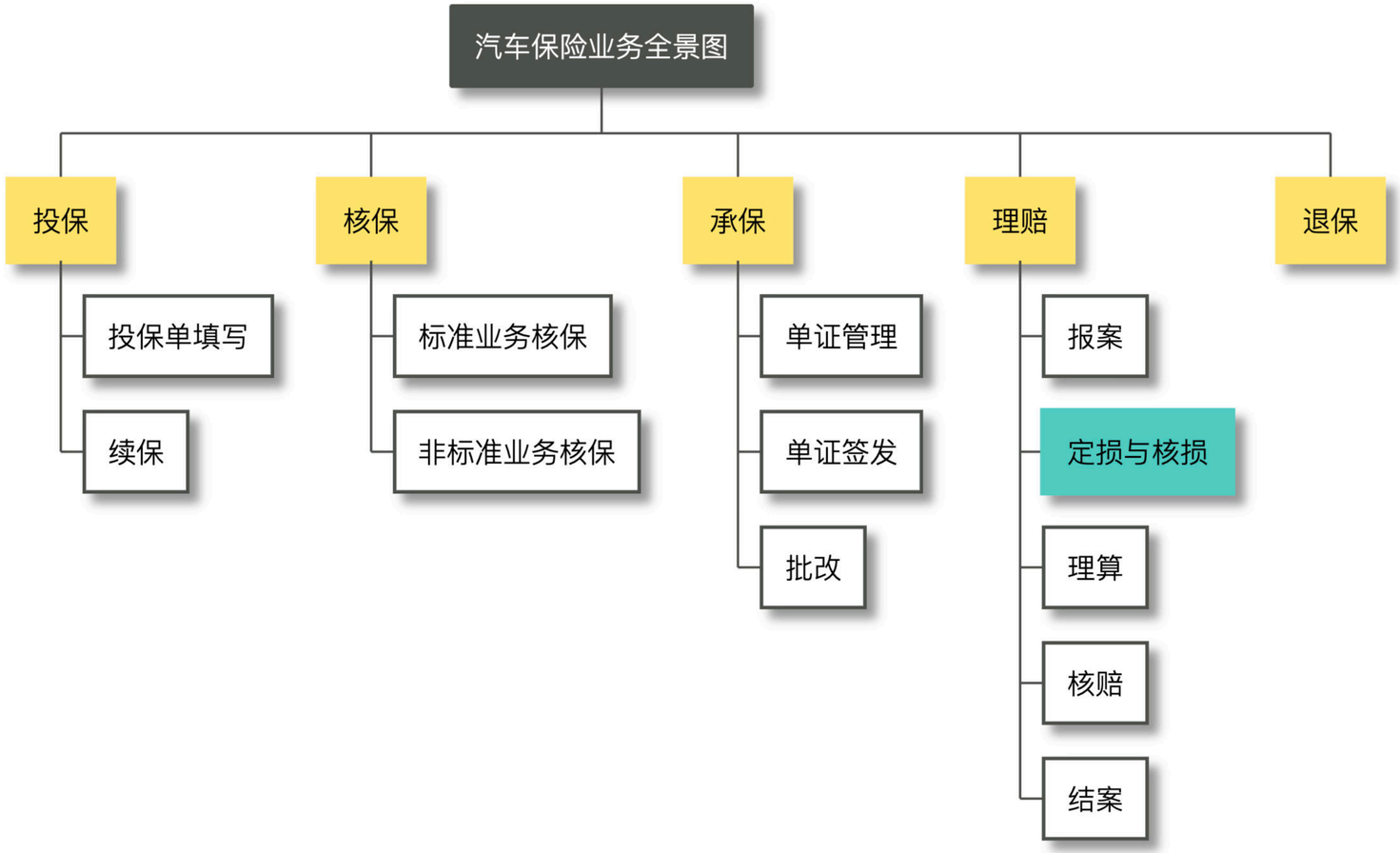
Source: The knowledge spiral, source: Nonaka and Takeuchi (1995, p.62)

如何降低认知复杂度

ThoughtWorks®

2

利用活文档降低业务认知复杂度

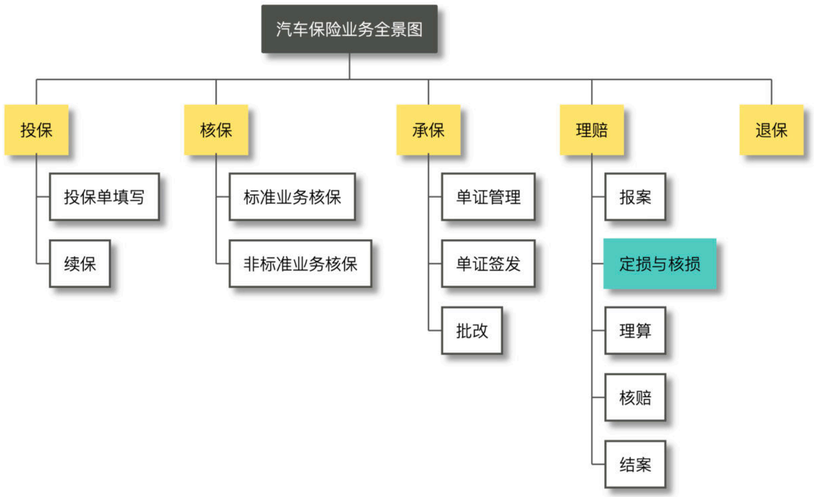


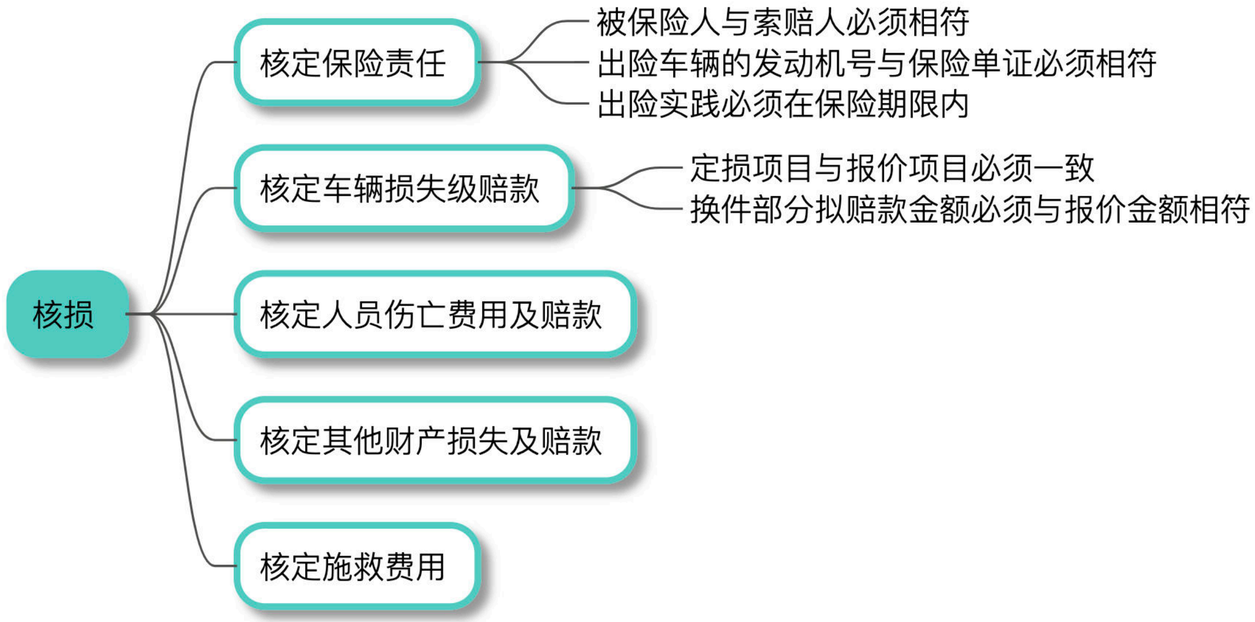
如何降低认知复杂度

ThoughtWorks®

2

利用活文档降低业务认知复杂度



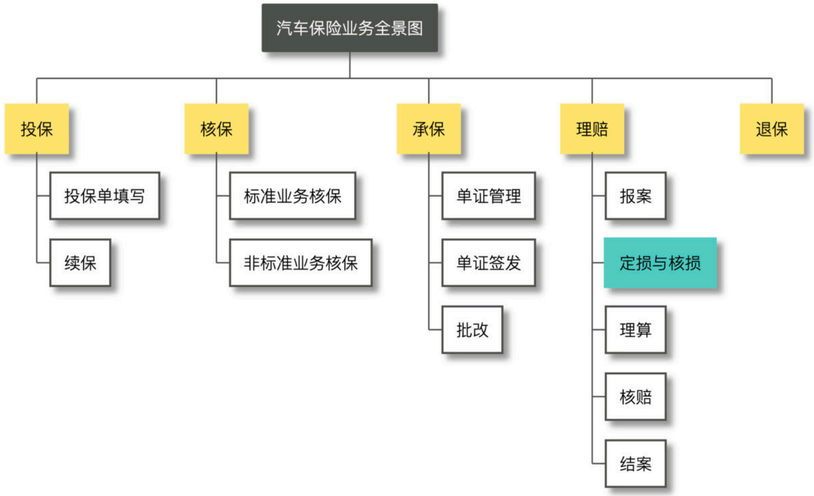


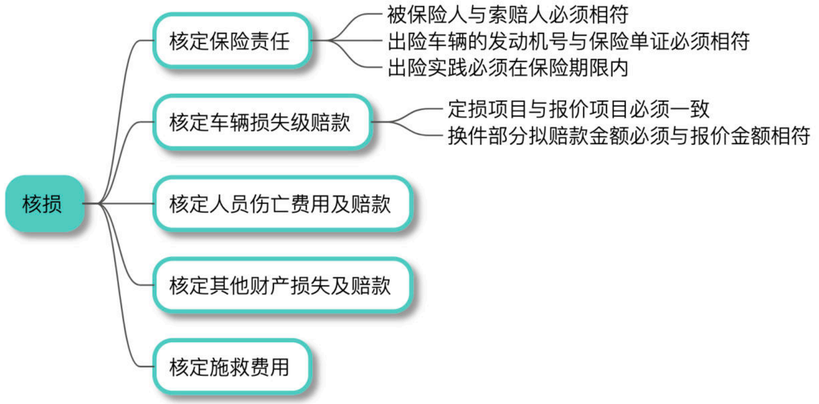
如何降低认知复杂度

ThoughtWorks®

2

利用活文档降低业务认知复杂度





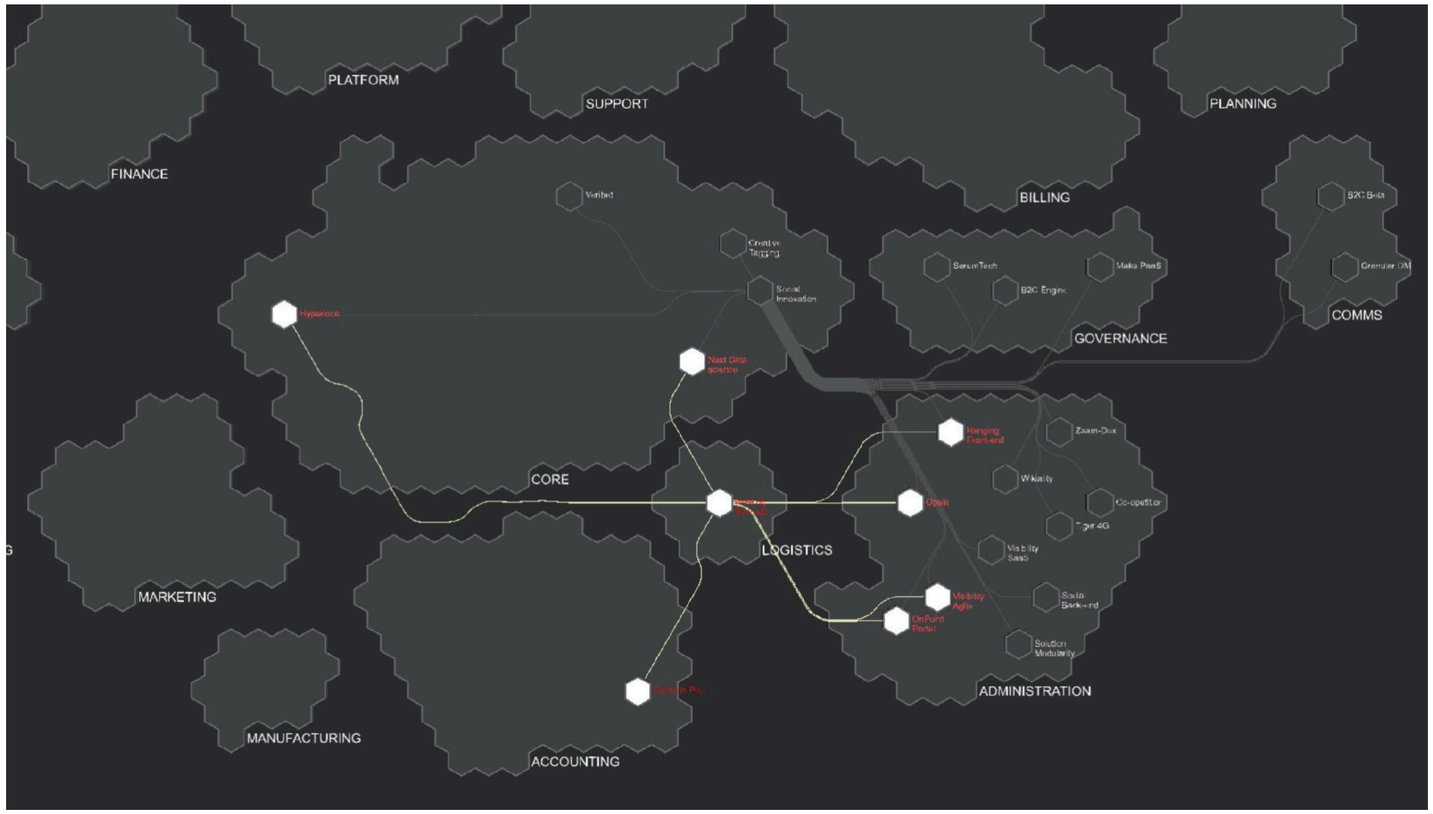

```
@Doc("核定保险责任")
public void checkLiability(Policy policy, ClaimCase claimCase) {
    if (! policy.getInsured().equals(claimCase.getClaimDeclarer())) {
        throw new InvalidLiabilityException("被保险人与索赔人必须相符");
    }

    //...
}
```

如何降低认知复杂度

ThoughtWorks®

3

可视化依赖分析降低系统认知复杂度



Backstage（试验） **Aplas（评估）** **Honeycom（评估）** **Systems（评估）**

03

如何进行改造

ThoughtWorks®

如何改造遗留系统

ThoughtWorks®

1

遗留系统现代化改造原则

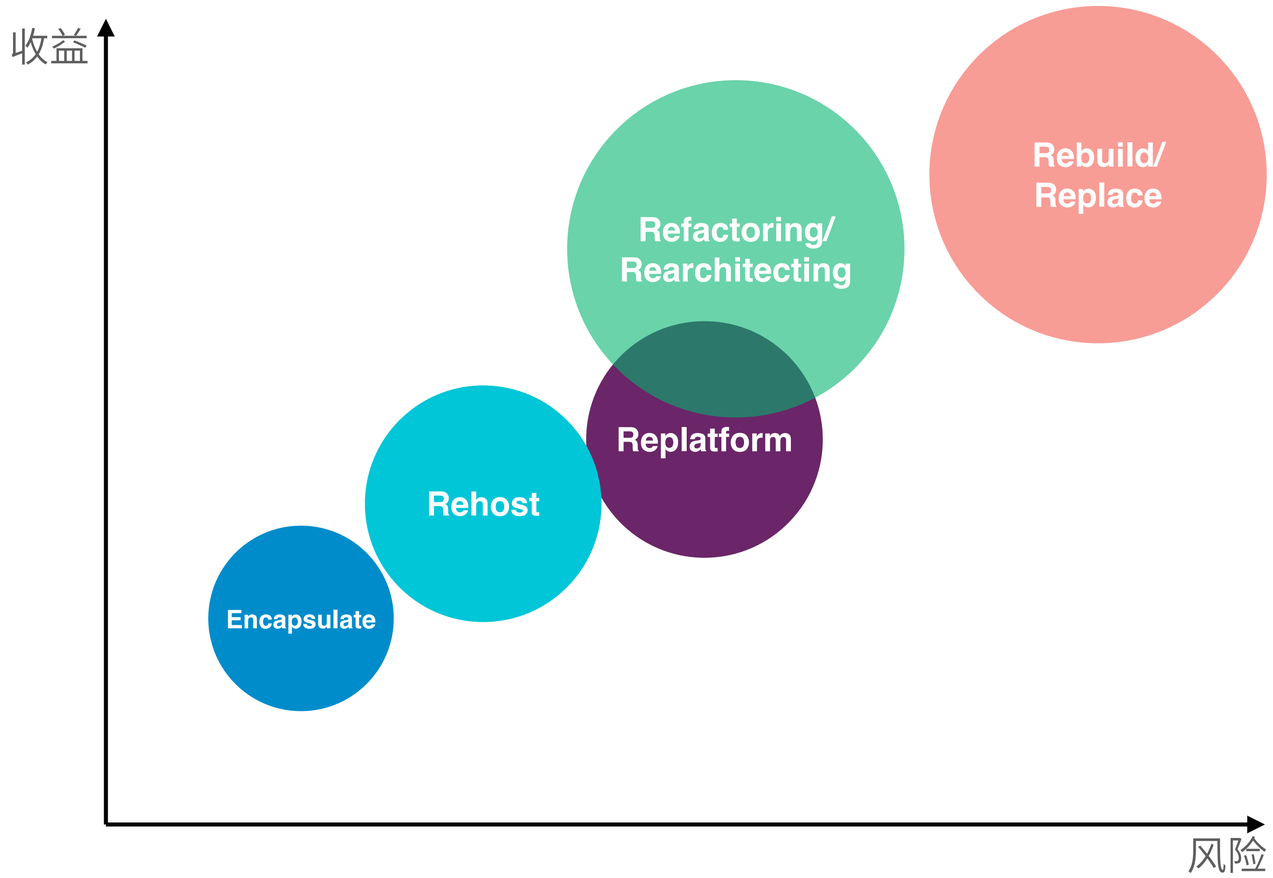
如何改造遗留系统

ThoughtWorks®

2

遗留系统现代化改造策略

- **Encapsulate**
  封装数据或者功能，以API形式提供服务。
- **Replatform**
  迁移到新的运行时平台，最小化代码修改。例如将Weblogic替换成Tomcat。
- **Rehost**
  将应用或组件部署到新的环境，例如云迁移或者容器化改造。
- **Refactoring/Rearchitecting**
  重构代码和架构，不改变业务行为，优化内部结构以提升系统健康度。
- **Rebuild/Replace**
  重建/替换系统或组件。



改造策略的收益、风险和成本（面积）

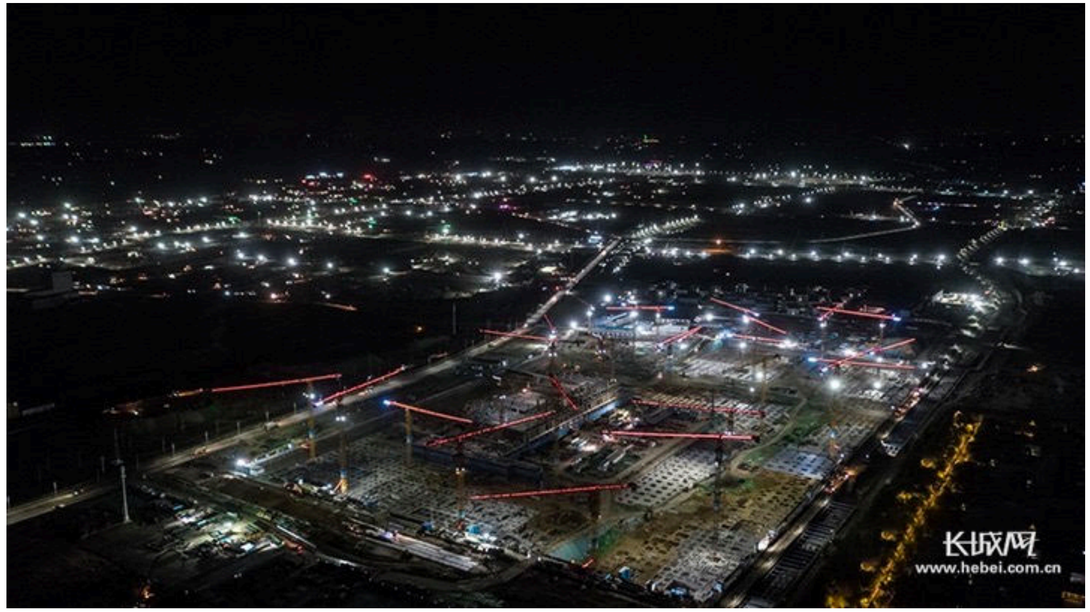
如何改造遗留系统

ThoughtWorks®

3

遗留系统现代化改造模式分类

建设新区



建设中的雄安新区

改造老城区

Photo by Laura Siegal on Unsplash

如何改造遗留系统

ThoughtWorks®

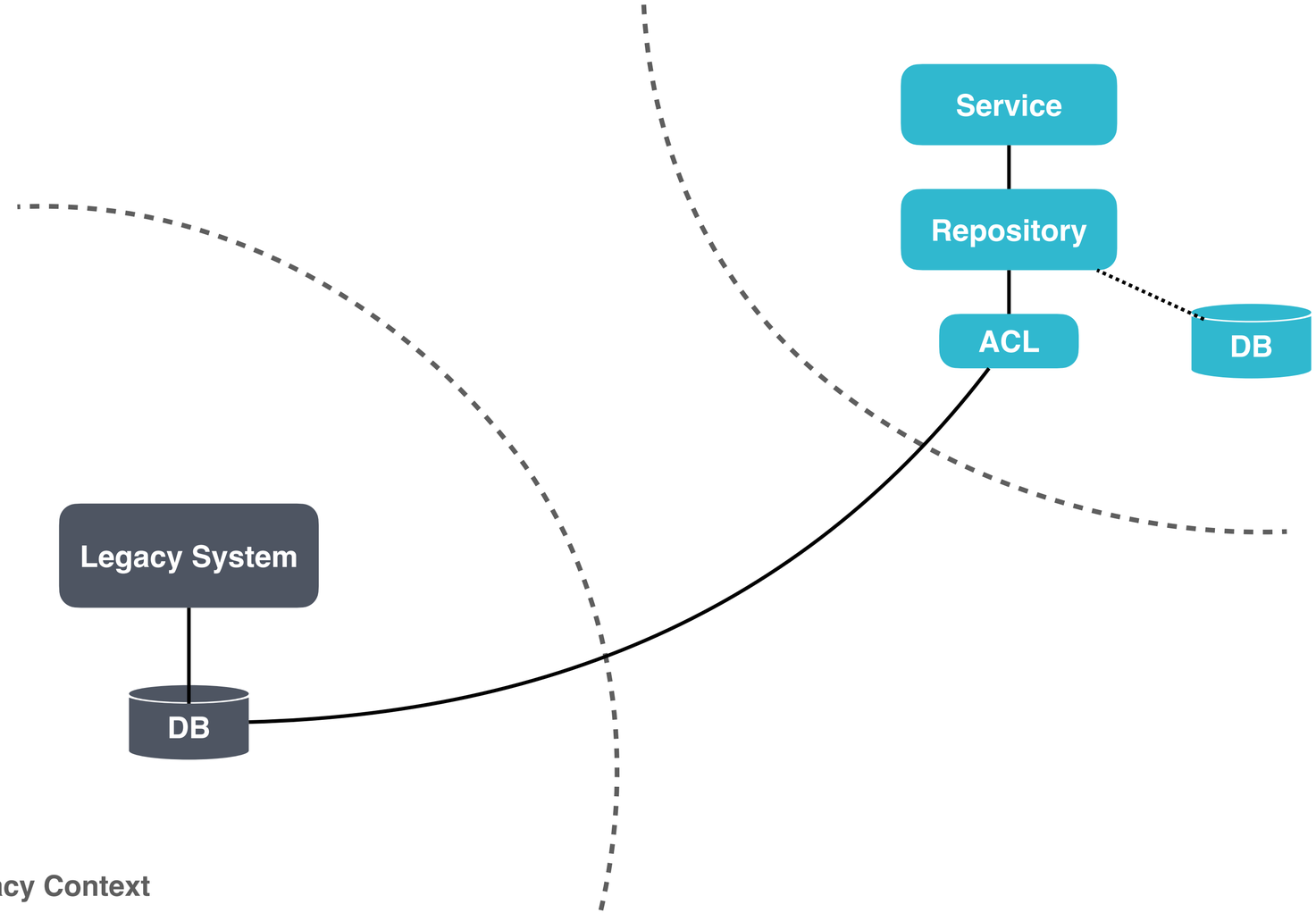
3

遗留系统现代化改造模式

建设新区

主要模式：

- 冒泡上下文
- 遗留系统封装API
- 事件拦截 (试验)
- DB封装API
- 变动数据捕获(CDC)



如何改造遗留系统

ThoughtWorks®

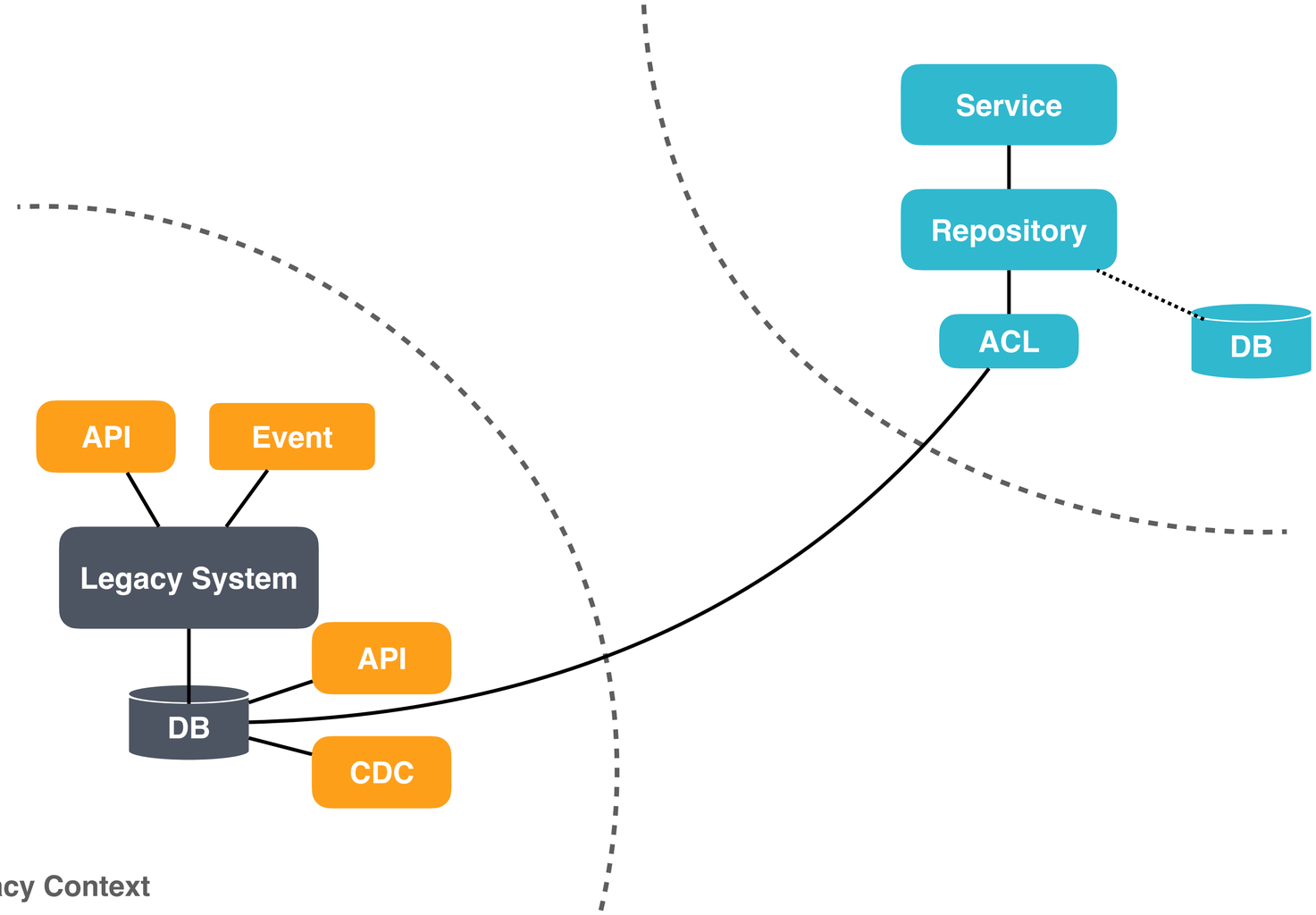
3

遗留系统现代化改造模式

建设新区

主要模式：

- 冒泡上下文
- 遗留系统封装API
- 事件拦截 (试验)
- DB封装API
- 变动数据捕获(CDC)



如何改造遗留系统

ThoughtWorks®

3

遗留系统现代化改造模式

改造老城区

主要模式：

- 单页应用 (SPA) 注入 (试验)
- 微前端（采纳）
- 抽象分支
- 扩张与收缩 (采纳)
- 变更数据所有权
- 数据库作为契约 (暂缓)

单页应用(SPA)注入 （试验）

单页应用注入允许你逐渐迭代掉旧的单页应用，直到它被新应用完全替代掉。与“由外而内”的绞杀模式不同，这是一种“从内到外“的方式来替换遗留系统。

抽象分支

"Branch by Abstraction”是一种对软件系统进行大规模修改的技术，它以渐进的方式让你在修改仍然可以进行时定期发布。

变更数据所有权

遗留系统的数据通常耦合在一起，通过识别并区分数据所有权，将其进行解耦，改为采用API的方式进行访问。

微前端 （采纳）

旧的、大型的、单一的前端被过去的技术栈所束缚，为了避免全面重写的危险，我们更愿意把旧的应用程序一块一块地扼杀掉，同时继续提供新的功能而不被单体所拖累。

扩张与收缩 （采纳）

也称为并行修改，经常与数据库或代码一起使用。当通过简单的扩张再收缩就可以满足需求的情况下，建议API也采用此方法，而不是用复杂的版本控制和破坏性的修改来实现。

数据库作为契约（暂缓）

在许多遗留系统中，都可以看到基于数据库的集成。当外部系统无法修改时，可以在内部优化DB结构，并将数据同步到新建的DB或采用视图作为契约。

如何改造遗留系统

ThoughtWorks®

4

改造后的验证

04

One More Thing

ThoughtWorks®

One More Thing

ThoughtWorks®

为什么别人能继承家族产业，我们只能继承遗留系统?

One More Thing

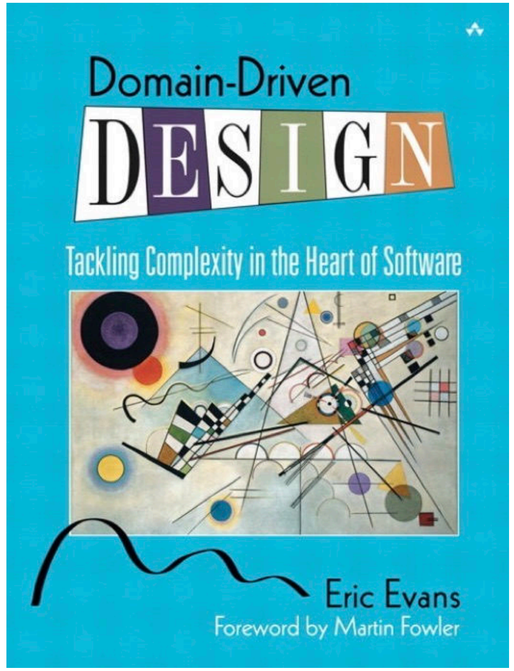
ThoughtWorks®

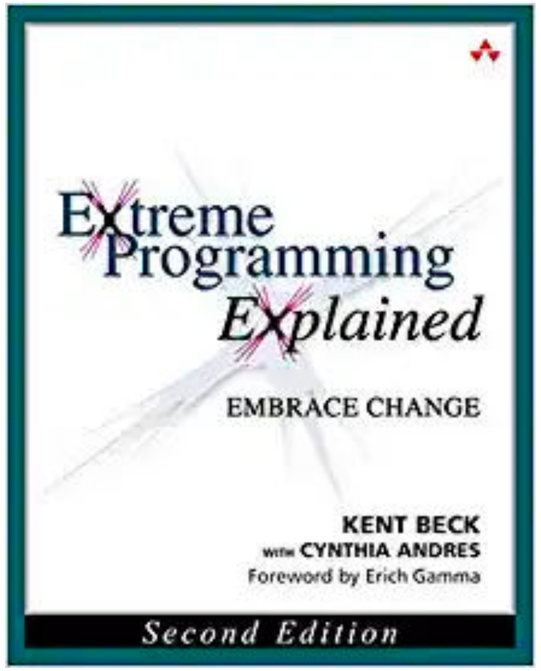
让我们面对现实吧，我们今天所做的一切就是在编写明天的遗留系统。

——Martin Fowler

One More Thing

ThoughtWorks®





THANK YOU

欲了解详情，请发送邮件至：
xsmei@thoughtworks.com

ThoughtWorks®

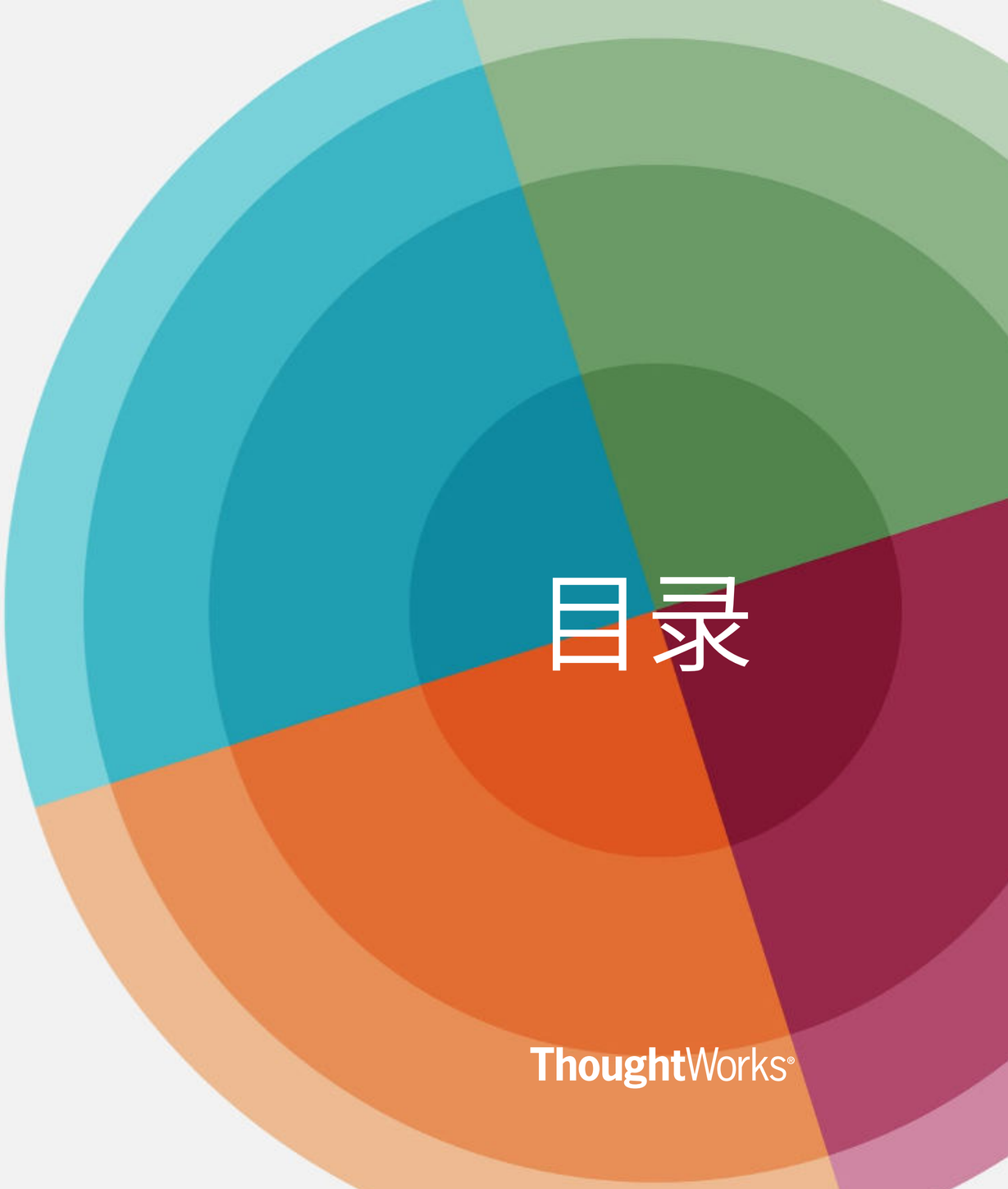
ThoughtWorks®

Data Mesh数据架构助力数据变现

白发川

ThoughtWorks 智能服务技术总监

目录



ThoughtWorks®

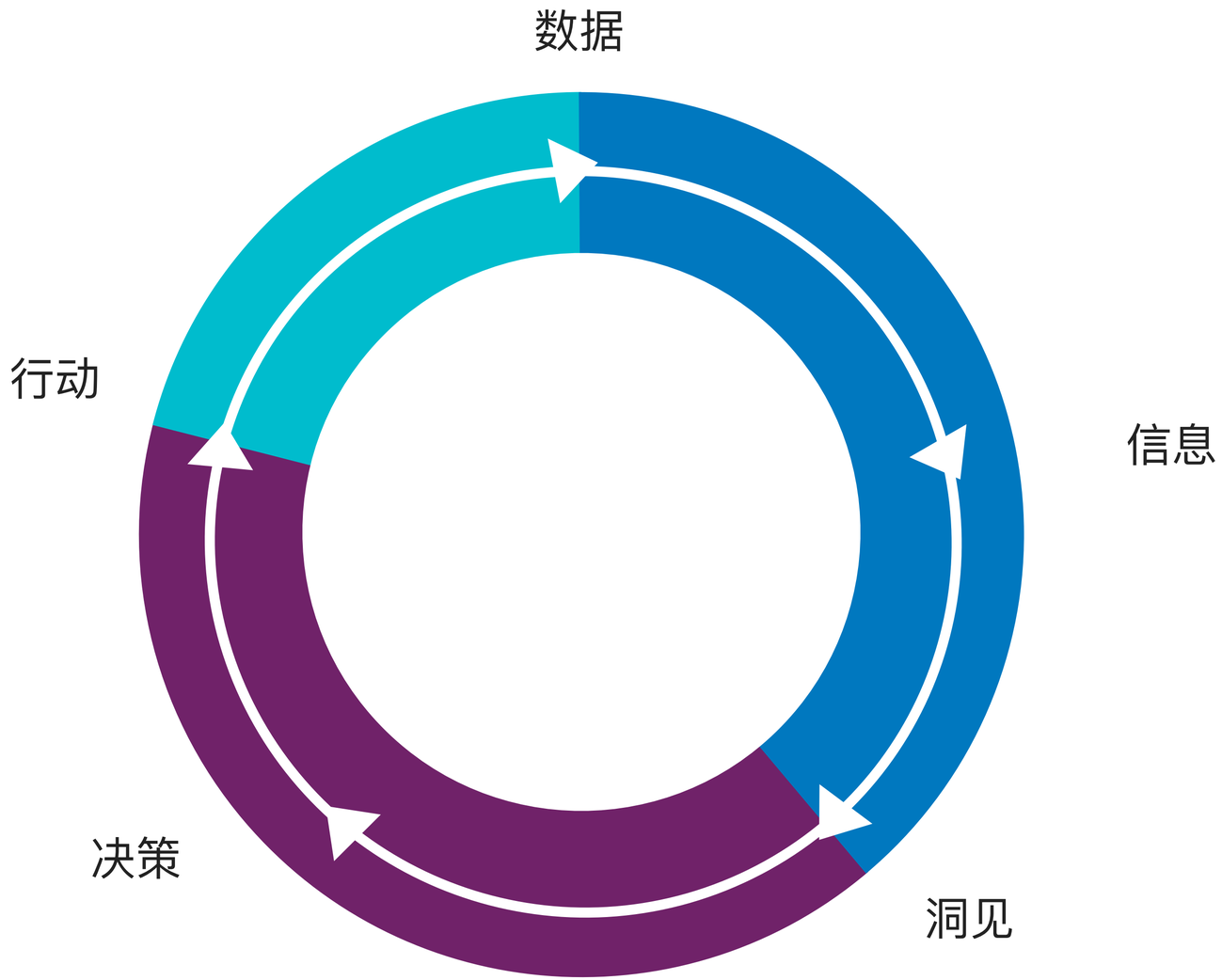
01

数据架构的挑战

ThoughtWorks®

智能驱动的决策

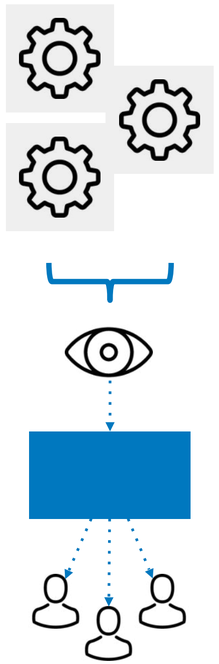
ThoughtWorks®



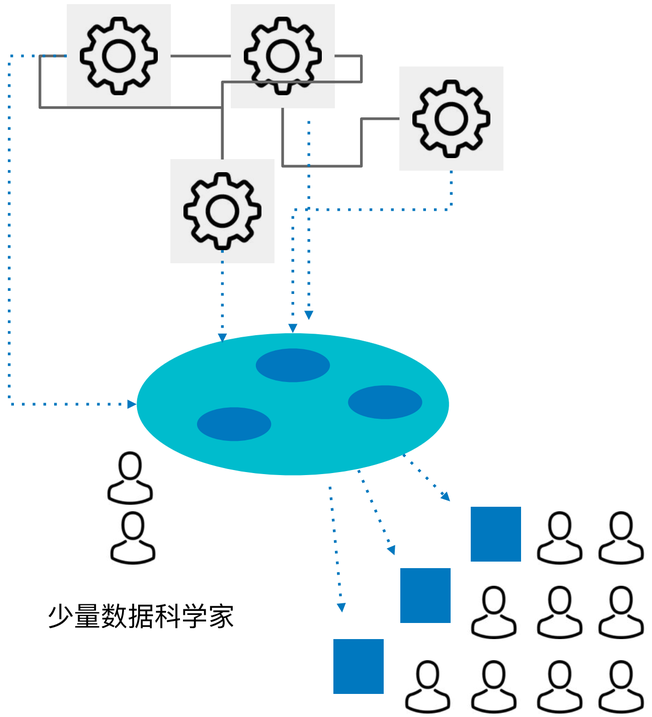
不同类型的数据架构

ThoughtWorks®

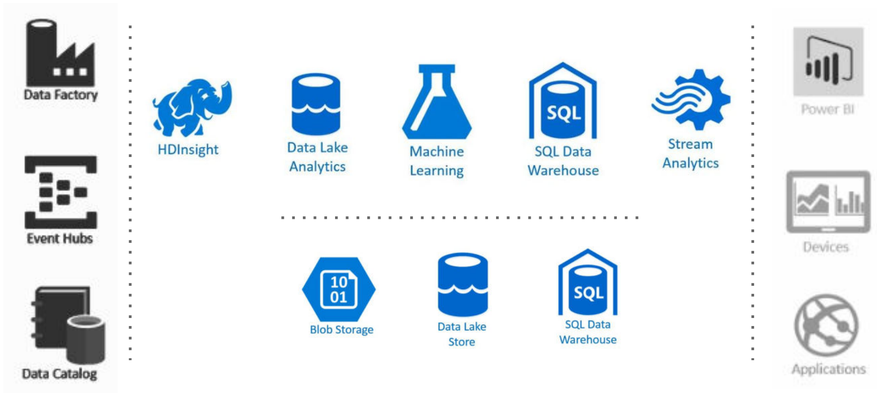
数据仓库



数据湖架构



云上数据平台



数据系统的挑战

ThoughtWorks®

难以启动

缺少用例支持，无法获得业务支持；

长时间的数据湖设计与技术评估；

需要统一组织内多个业务或技术部门；

数据源难以规模化

缺少手段对错综复杂的源数据系统进行疏浚与管理；

难以跟上不断增长的数据源系统规模；

数据消费难以规模化

数据平台项目跟不上企业创新要求；

用例过窄，难以满足规模化需求；

平台能力跟不上错综复杂的用例需求；

数据难以商业化

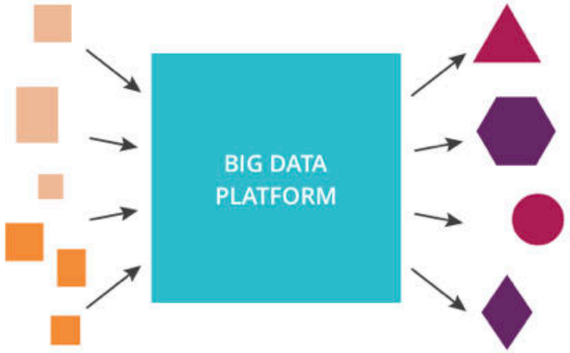
极高的开发和运营成本；

难以将数据平台真正转化为商业竞争力；

难以形成创新文化。

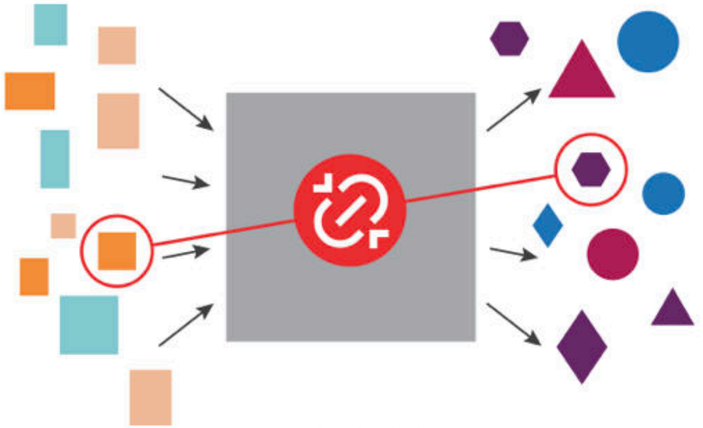
影响数据架构价值的根本原因

ThoughtWorks®



集中式架构

数据业务分离



高成本协作

基础设施无法响应业务弹性需求

单体数据架构下，基础设施资源所有业务共享，进行集中式的管理和维护，无法基于业务需求灵活进行资源调整。

数据处理流水线复用成本高

每一个数据流水线卫独立的数据工作空间上下文，跨流水线的数据结果或者中间结果需要进行复用时成本较高，难度较大。

数据商业化成本高

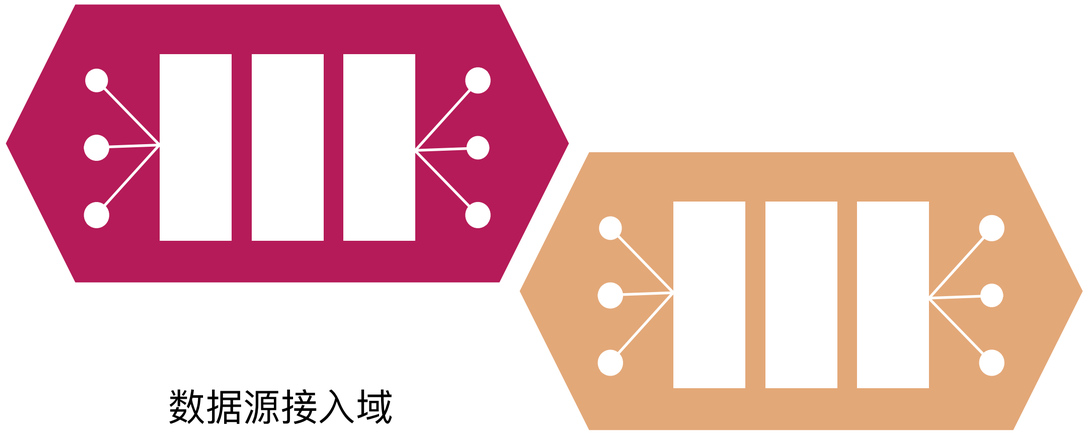
加工数据以非产品思路对数据进行处理，因此大部分数据处理结果集无法以商品的形式度量其业务价值。

数据处理成本较高

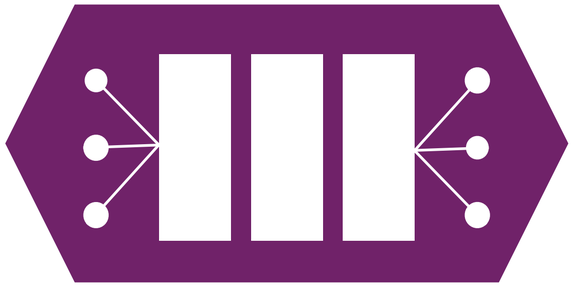
单体数据架构模式下，大部分的数据处理工作进行集中统一管理，在涉及更多业务场景支持，更多团队协作下，数据处理的成本较高。

我们希望的效果

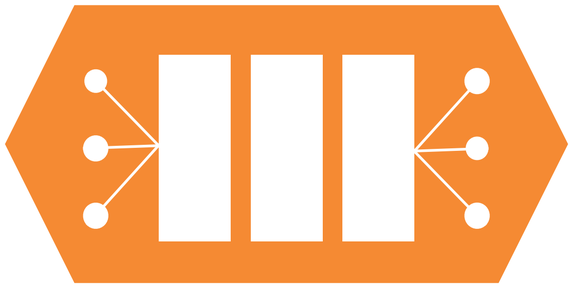
ThoughtWorks®



数据源接入域



共享数据域



数据服务消费域

影响数据架构价值的根本原因

ThoughtWorks®

> 我们现在数据架构面临的挑战，类似于微服务架构之前的单体软件所面临的挑战。
>
> 如果我们的数据架构能够以微服务的架构思想来设计，会带来什么样的改变。

02

什么是Data Mesh

ThoughtWorks®

Data Mesh核心思路

ThoughtWorks®

数据域对齐业务域

民主，可发现，
自助服务

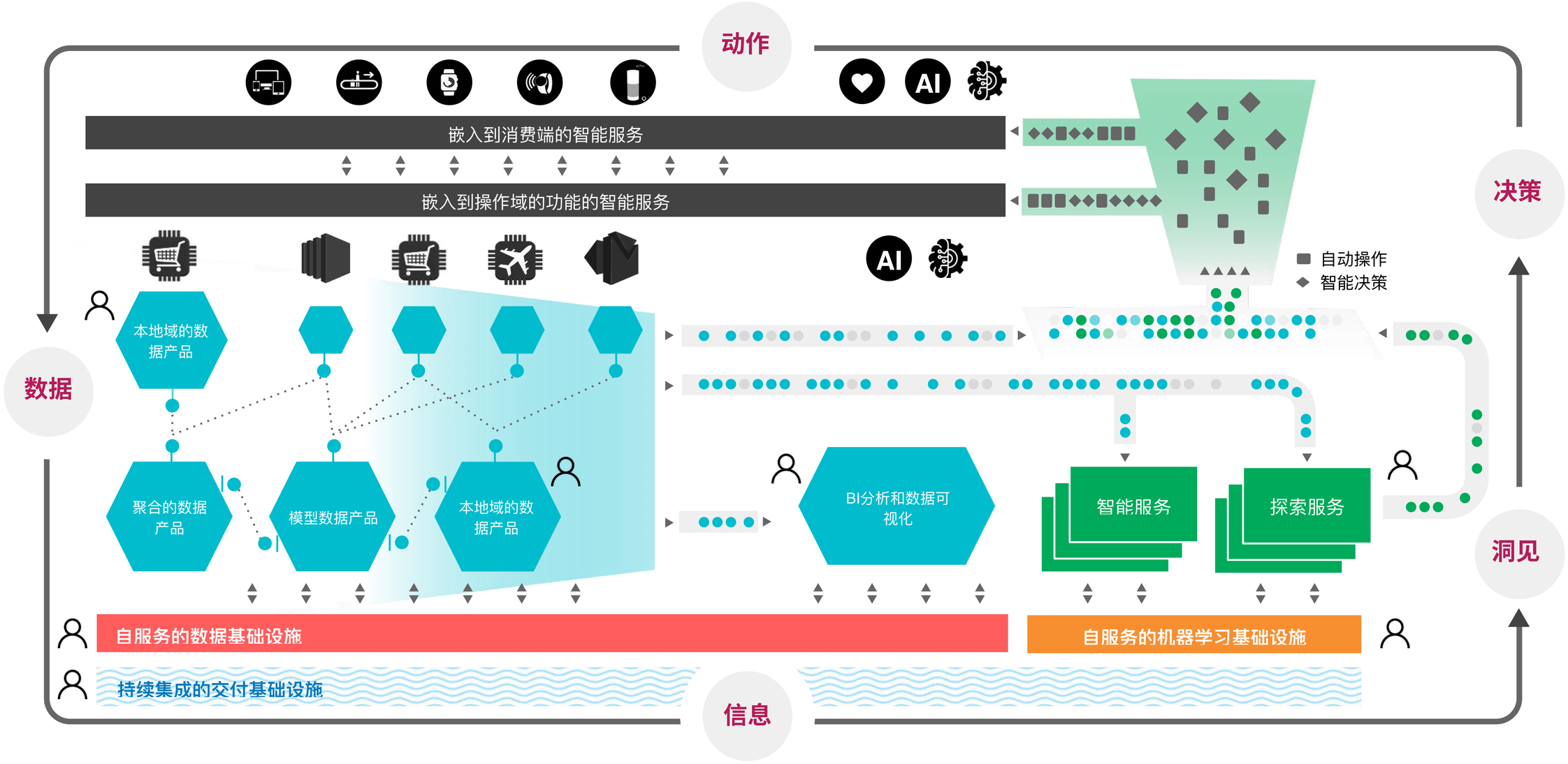
减少摩擦提高效率

协作和可信

创新

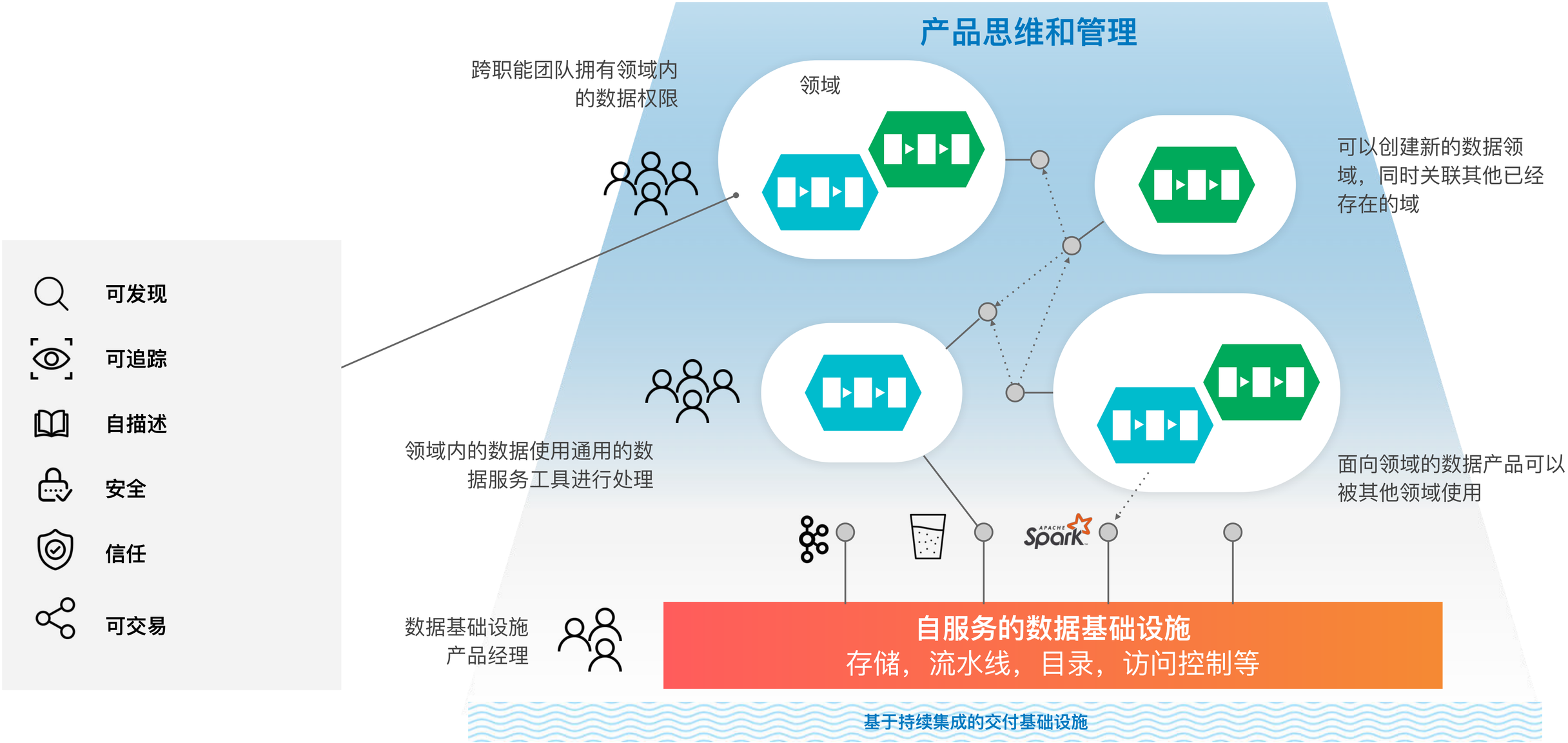
Data Mesh架构逻辑

ThoughtWorks®



Data Mesh架构逻辑

ThoughtWorks®



Data Mesh架构逻辑

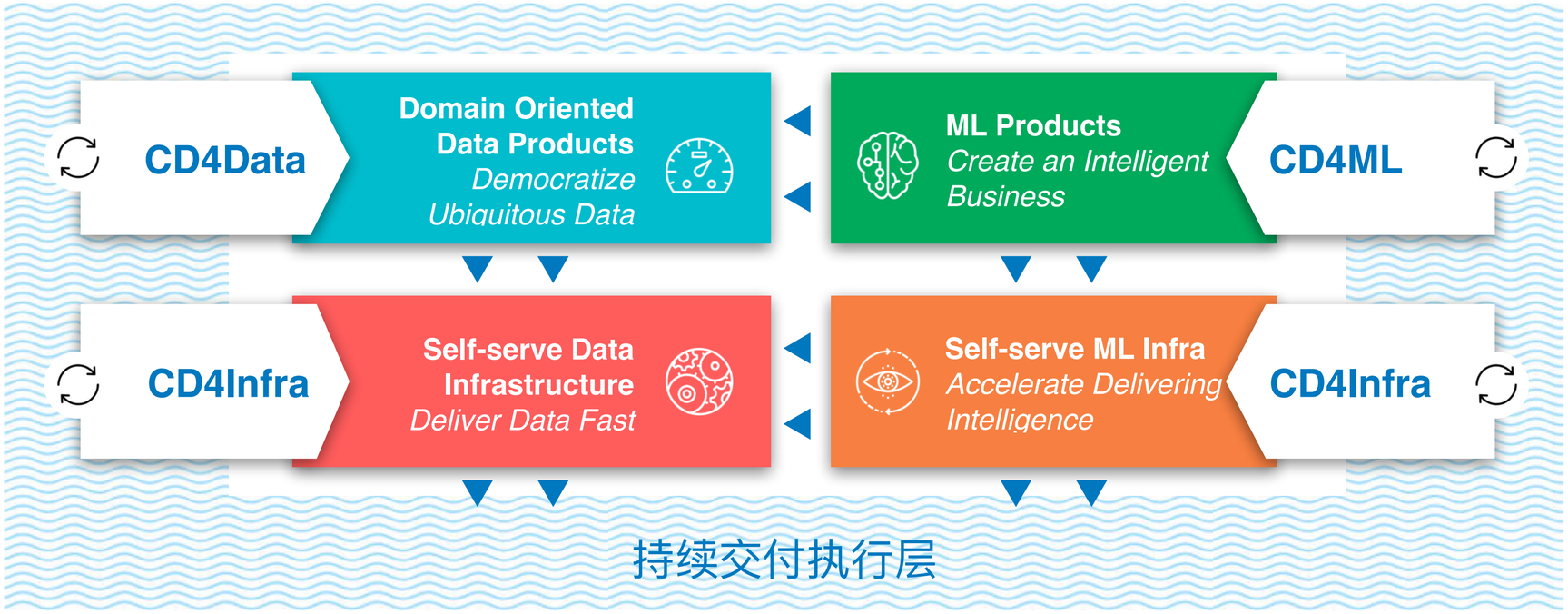
ThoughtWorks®

通过精益数据治理进行顶层设计

战略

基于数据治理模型和产品进行全局的数据治理

技术

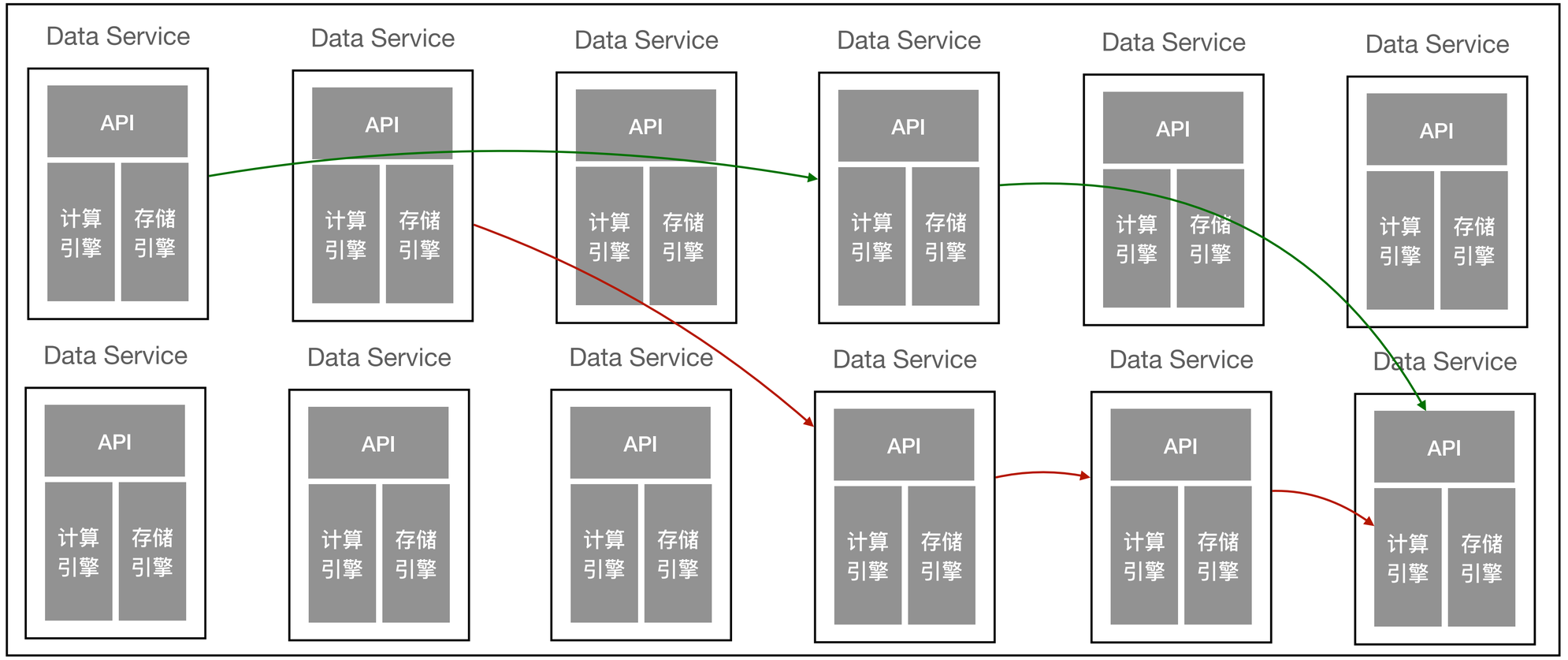


运营

分布式的系统和组织结构进行支撑

Data Mesh架构逻辑

ThoughtWorks®



定价体系

元数据管理

数据服注册与发现

订阅通知

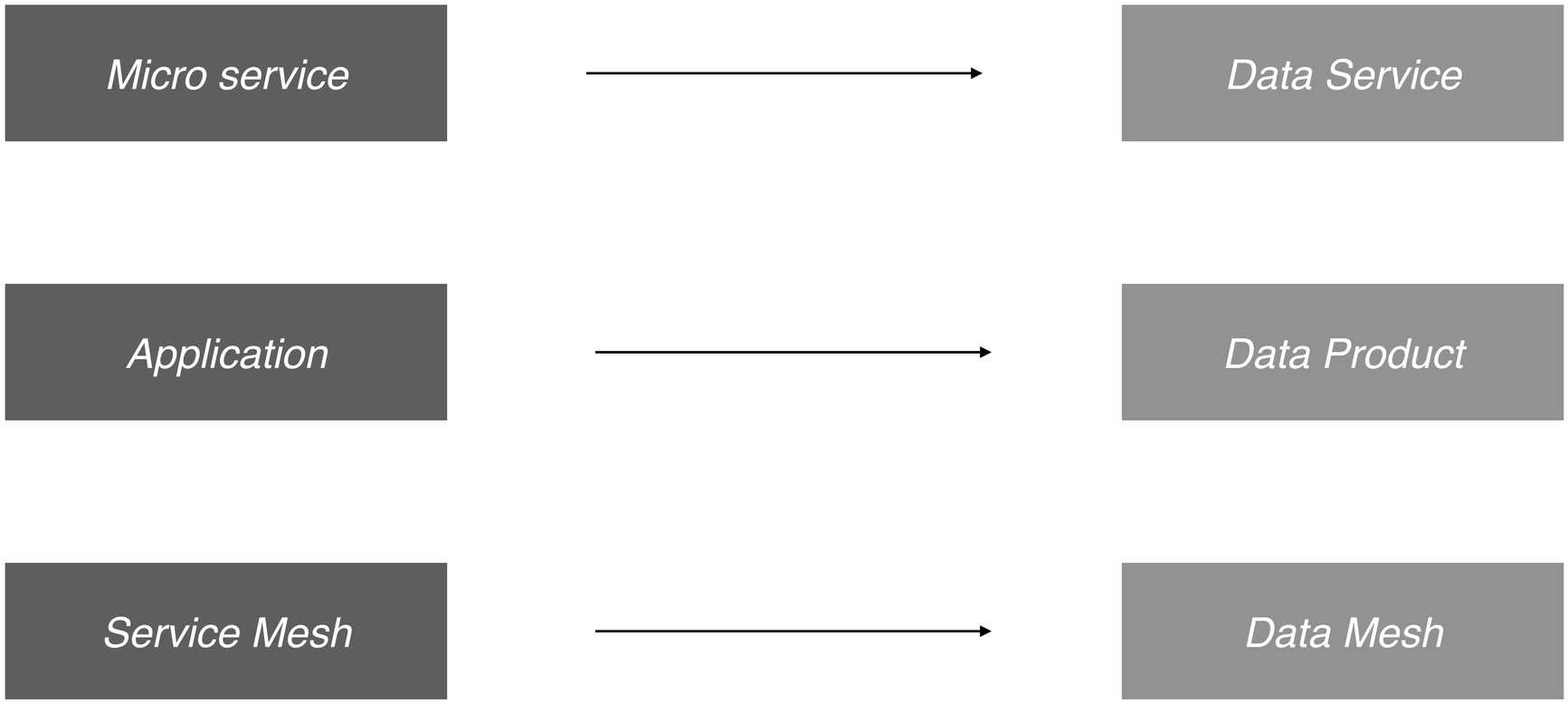
数据版本

API版本

云基础设施

Data Mesh-云原生数据架构

ThoughtWorks®



03

Data Mesh演进路线

ThoughtWorks®

数据架构类型

ThoughtWorks®

- **单体架构（数据仓库）：**
  - 复用业务架构*:* 和业务系统高度耦合，数据架构建立在业务系统之上，和业务系统共用同一个DB。
  - 大泥球架构*:* 整体没有分层，耦合的架构
  - *Layered Data Architecture:* 采用了Kimball, Inmon, 星形模型,雪花模型等分层模型的数据架构，更加容易进行改变了创造适应度函数。
  - *Data Vault:* 整体架构进行了模块化的拆封，更加容易结耦，也更加容易进行模块替换。
- **分布式数据架构（基于数据湖的架构）：**
  - 基于存储系统（*S3*，*HDFS*）*:* 采用分布式存储系统对数据进行存储的数据存储解决方案，数据难以修改。
  - 基于*NoSQL* 数据库 *(HBase, etc.):* 将数据存储在NoSQL的存储介质中，同时承担存储和分析的职责，高度耦合。
  - *SQL on Hadoop:* 基于Hadoop的 SQL解决方案，更加容易演进和分层设计。
  - 实时数据处理架构*:* Lambda, Kappa等架构，很容易修改和演进，但是整体复杂度会很高。

数据架构类型

ThoughtWorks®

- **面向域的分布式数据架构（Data Mesh）：**
  - 数据即服务：提供数据端到端开发所需要的工具和服务，架构既承担着工具提供的职责，也负责维护工具产生的数据服务，更像是一个数据的SAAS平台。
  - 数据平台即服务：提供数据PAAS的能力，架构提供更加灵活的能力，允许租户使用自己的工具，也可以使用平台的工具，架构仅仅是基础设施。

演进式数据架构

ThoughtWorks®

演进式数据架构-增量变更

ThoughtWorks®

CD4Data

Domain Oriented Data Products
Democratize Ubiquitous Data

- Source / native domain data products
- Harmonized & enriched domain data products
- Fit for purpose modelled data products
- Learnt insights, KPIs and visualizations
- BI artifacts

ML Products
Empower an Intelligent Business

- Intelligent domain services
- Domain specific reusable (pre-trained) models
- Intelligence infused business products

CD4ML

CD4Infra

Self-serve Data Infrastructure & Tooling
Deliver Data Fast

- Real-time streaming backbone
- Polyglot storage on-demand
- Data pipeline and democratization tooling
- BI infrastructure
- Data quality monitoring infrastructure
- Unified access control
- Data discovery and catalogue tooling

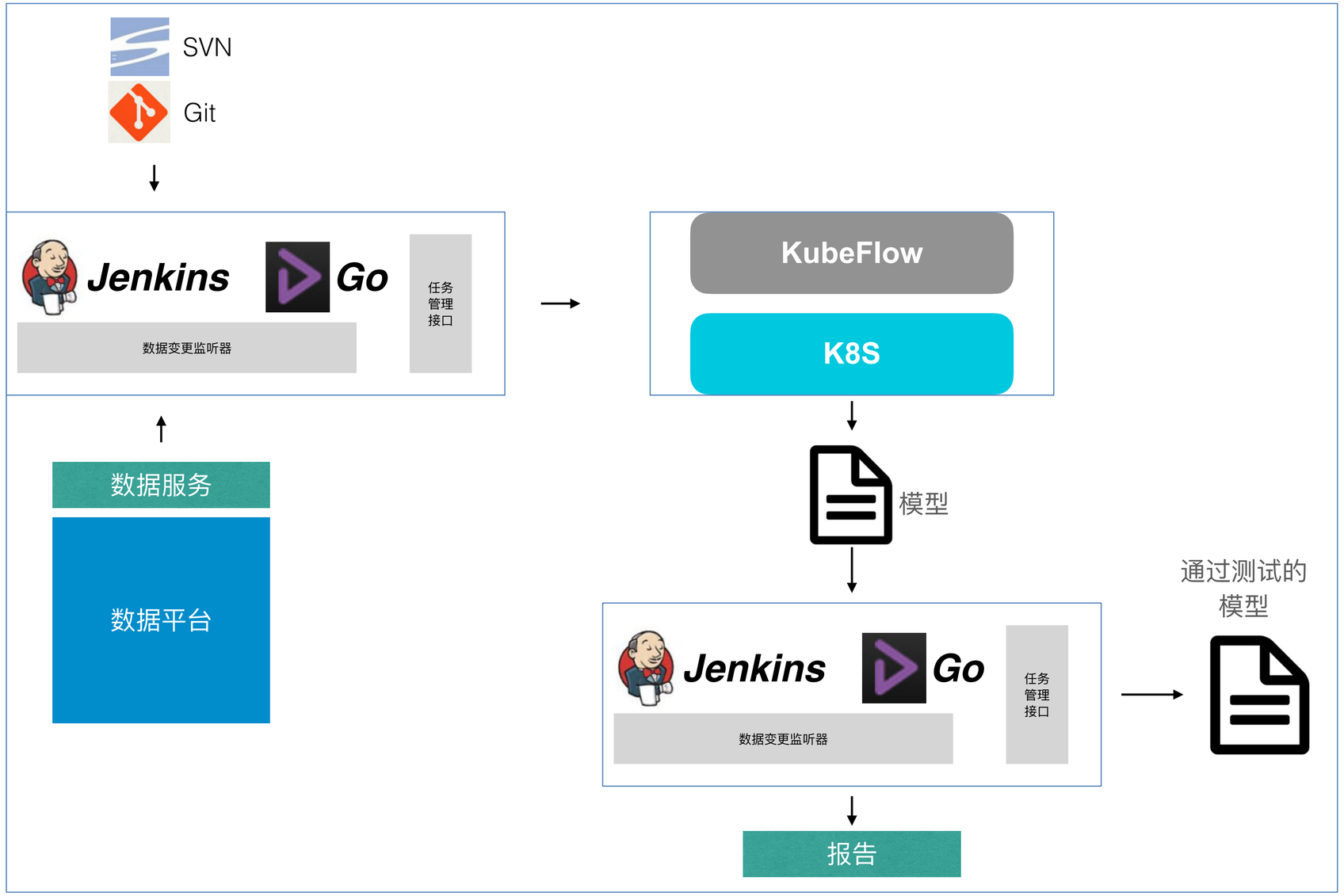
Self-serve ML Infra
Accelerate Delivering Intelligence

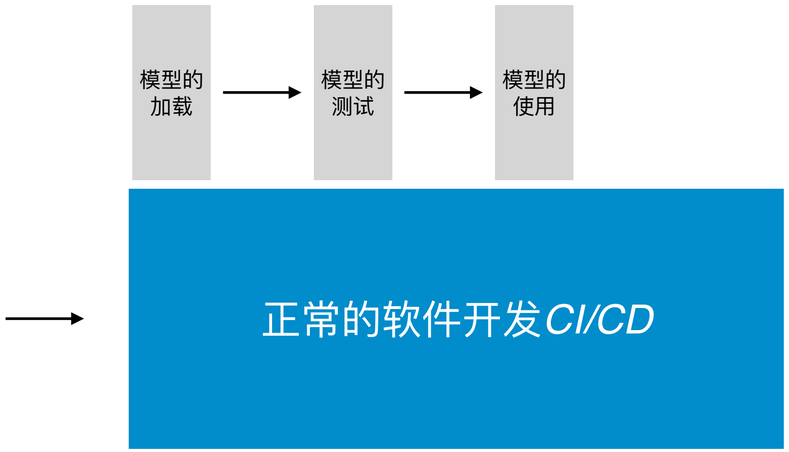
- ML sandbox on-demand
- Training & validation dataset mgmt
- Feature engineering tooling
- Research & experimentation platform
- Model deployment tooling
- ML Frameworks support

CD4Infra

演进式数据架构-增量变更

ThoughtWorks®





演进式数据架构-适应度函数

ThoughtWorks®

	架构可用		架构安全	架构敏捷	架构演进	架构前瞻
	性能	质量	数据/流程安全	易于改变	架构解藕	方向指导性
DATA		数据质量监控			数据血缘	
OPS	- 平均恢复时间 - 恢复失败百分比	- 生产环境失败比例		- 部署频率		

演进式数据架构-适当得耦合

ThoughtWorks®

THANK YOU

欲了解详情，请发送邮件至：
fcbai@thoughtworks.com

扫码了解更多

ThoughtWorks®