金融级系统海量流量下的高可用架构实践

开　　篇

分享的视角

01 黄金链路金融系统

电商黄金链路 、全链路 、全场景、海量流量

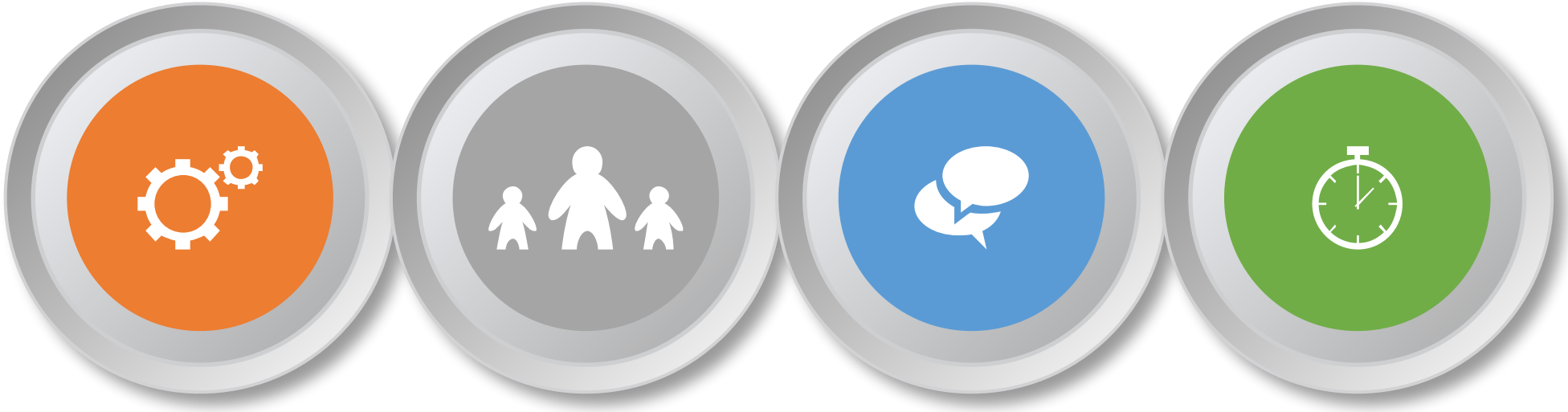
02 关注度

03 影响范围

分享的原因

02 变化

04 why or how



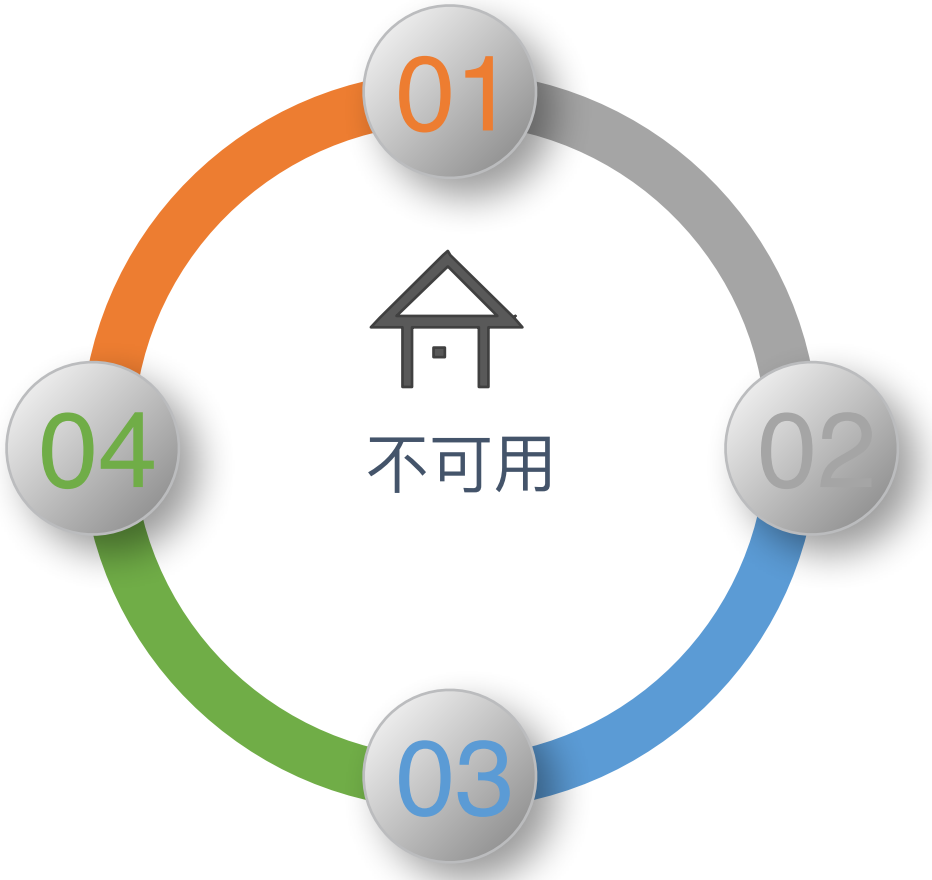
01 成长

03 银弹

定　　义　　篇

不可用

服务器，可控机房故障



光纤挖断，不可控外部事件

9.11 黑天鹅事件

上线故障，内部人为事故

金融业务不可用表现

信息泄露

- 信息盗取：木马病毒、假冒网站、数据安全事故
- 信息泄露：身份信息、账号密码泄露

信贷欺诈

- 身份冒用/非本人申请
- 身份美化
- 恶意逾期/恶意失联

交易欺诈

- 盗刷
- 套现
- 盗帐号交易
- 线上业务交易欺诈

账号盗用

- 账号接管：登录、修改及管理身份、绑卡

不可用是什么

已知

未知

时间

人

不可用是指潜在风险在一定条件触发下的结果呈现

业内定义


SLA 服务水平协议
MTTR 平均故障时间
04
03
02
01
SLI 衡量服务质量的指标
MTBF 平均故障间隔时间

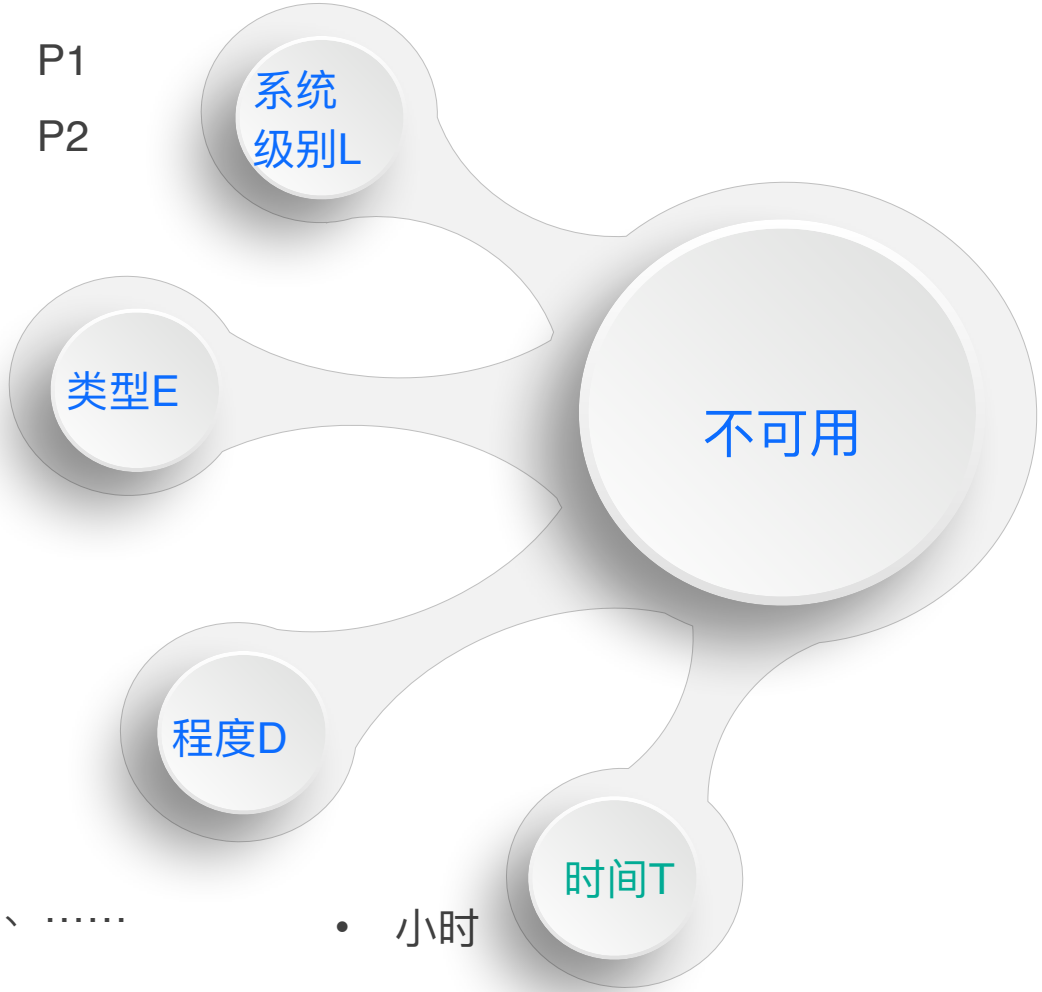
业务视角不可用

不可用 =

L1.E1.D1.T1 + ….. + Ln.En.Dn.Tn

- P0
- P1
- P2

- 业务资质
- 资损
- 舆情
- 客诉
- 用户体验
- 业务收入、预期
  - 核心业务
  - 全部接口
  - 100% 、80% 、……



- 小时
- 分钟
- 秒

不可用的特性

趋势性

不可用的等级有随时间及其他外部因素逐步提升的趋势

隐秘性

风险的触发条件比较隐晦

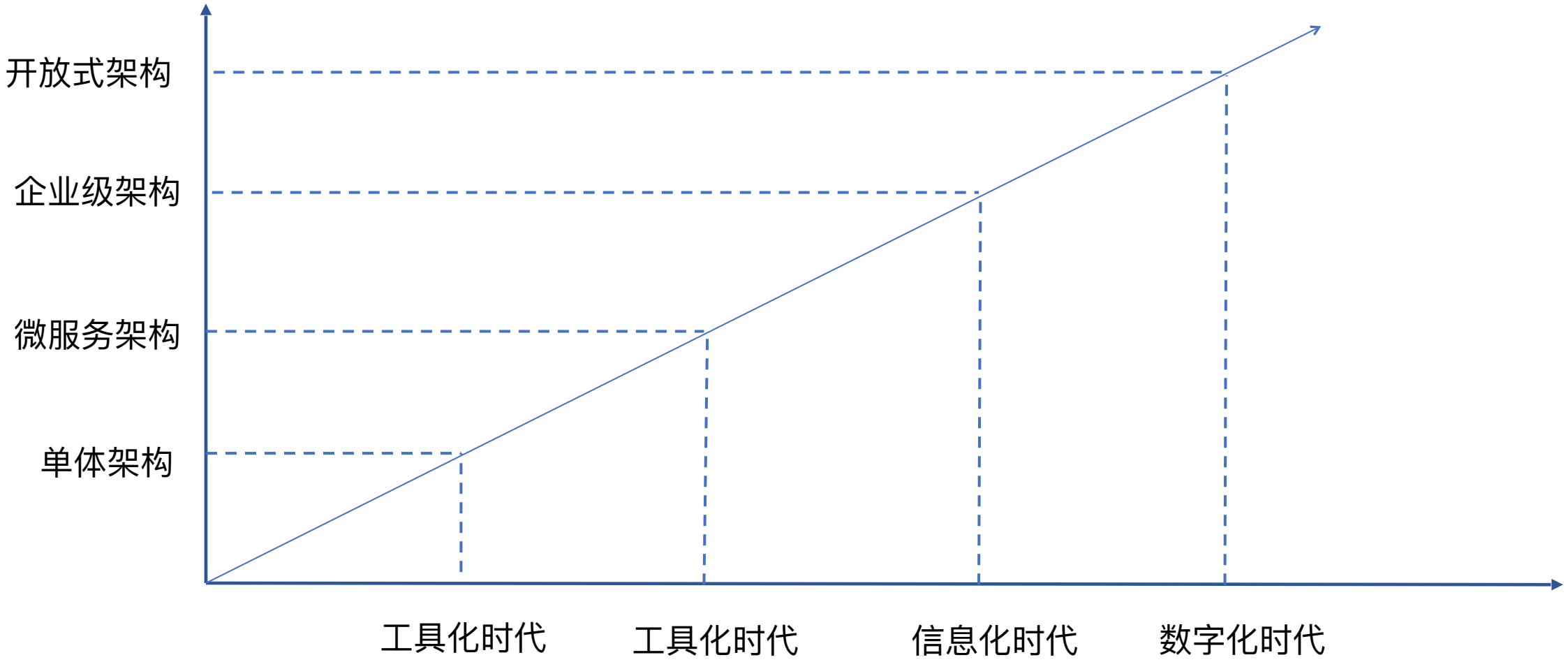
爆发性

- 交换机宕机
- 主机上其他应用有定时任务

不可用原因 – 时代 – 业务

VUCA乌卡（易变性、不确定性、复杂性、模糊性）时代

业务的现状是我们现在已经从单一的做业务 ， 到 **业务生态** ， 我们处在一个快速裂变的时代 **VUCA**



不可用原因 – 时代 – 基础设施

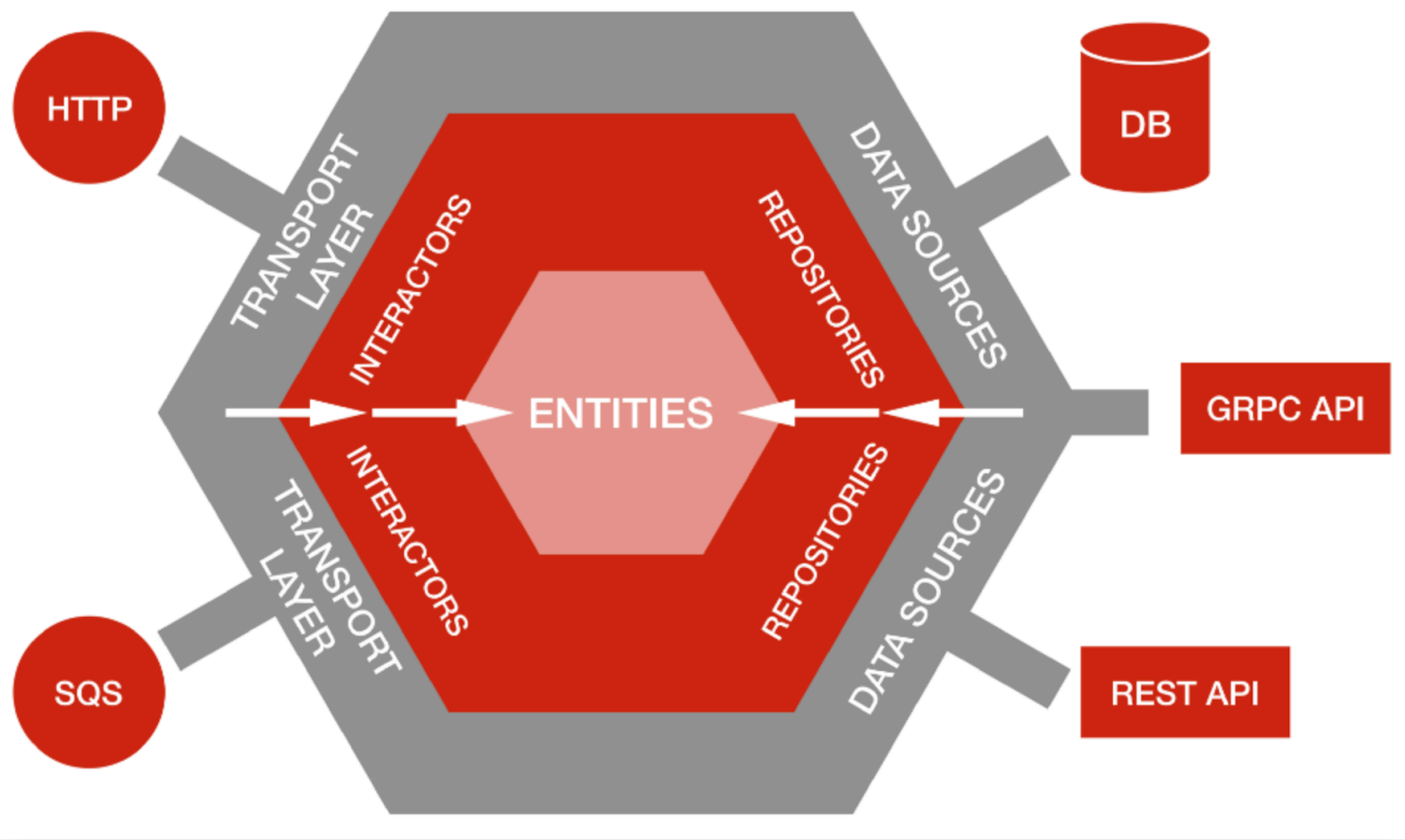
2009

《 Above the Clouds : A Berkeley View of Cloud Computing 》

2019

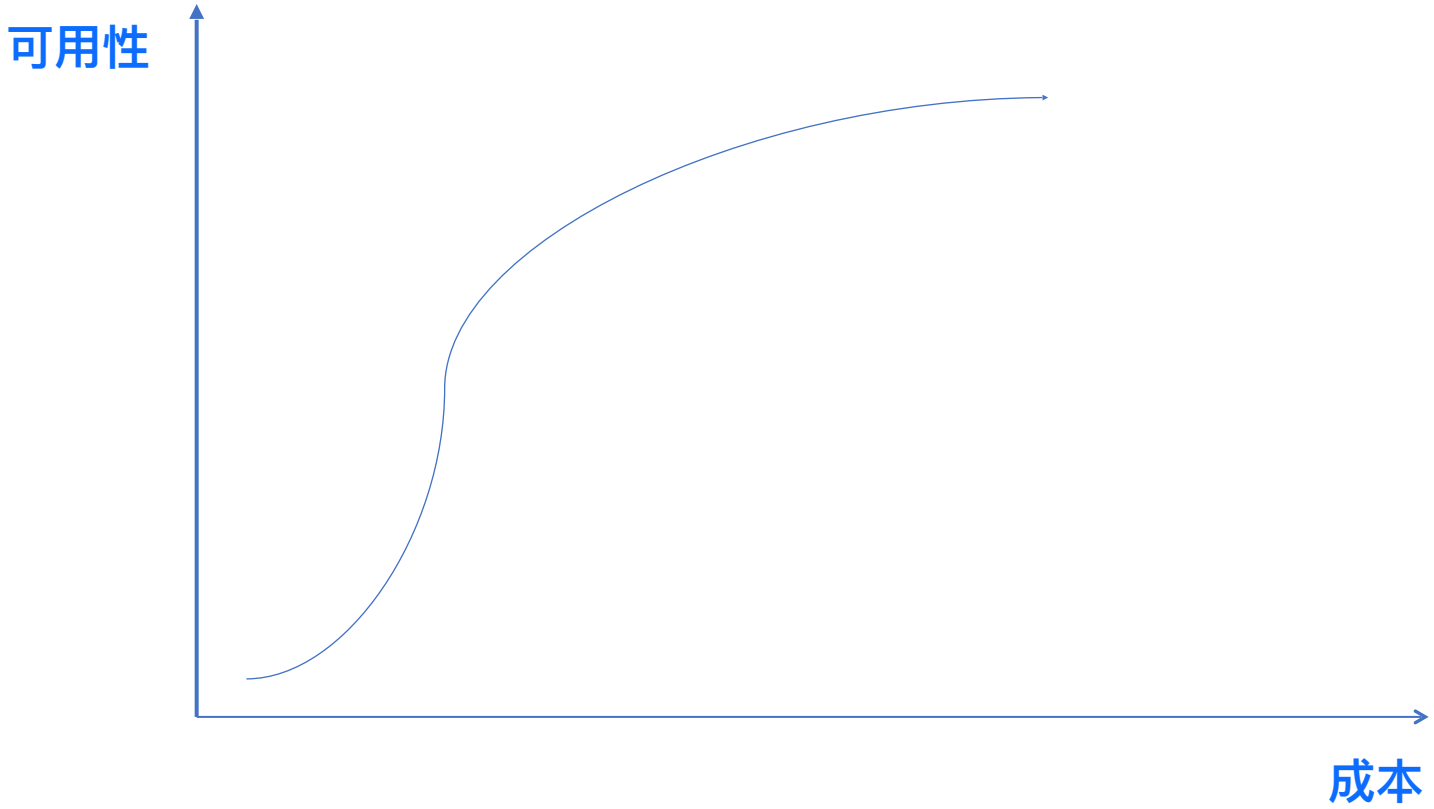
《Cloud Programming Simplified: A Berkeley View on Serverless Computing》

不可用原因 – 内因



- 流量
- 变化 （时间 ...）
- 不可靠
- 耦合
- 系统架构
- 上线

不可用原因 – 成本



成本 VS 风险

不可用原因 – 认知

所有降级措施是有效的

低优先级系统不需要备战

上游已经替我做了保护

下游是稳定的

流量洪峰后系统能快速恢复

所有应急预案是可执行的

历史的经验是可靠的

系统重试能解决问题

不可用原因

- 高可用 涉及 所有 业务场景 、 系统各层级 、 软件开发的全生命周期 、 组织架构 、团队分工 ，贯穿整个 软件开发体系

- 高可用的难点在于对软件的全景认知 ， 对于 （新）变化 的感知 ， 对于 (历史) 风险点 的识别 ，在于事前预估与预案 ， 事中有效处理 ， 事后复盘总结 ， 形成闭环

- 即使付出了99%的努力 ， 也会因为 1%的疏忽 ， 带来意想不到的结果 ， 所以高可用应该是一套需要持续完善和迭代的体系和方法论 、 需要持续的关注和投入

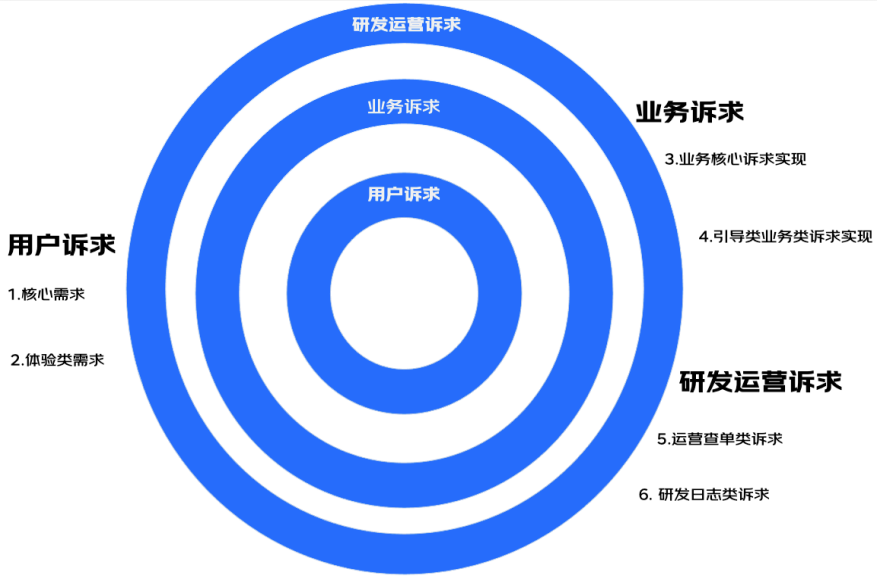
结　　构　　篇

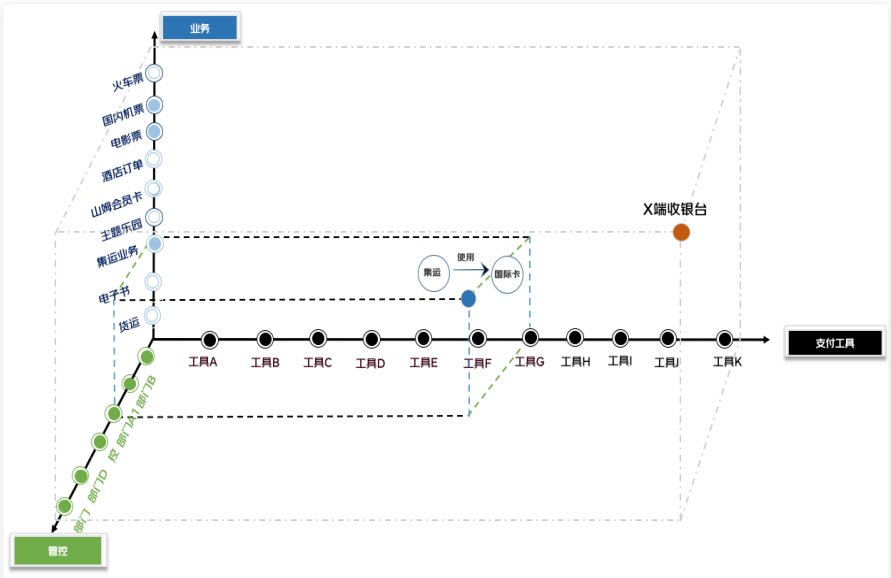
结构

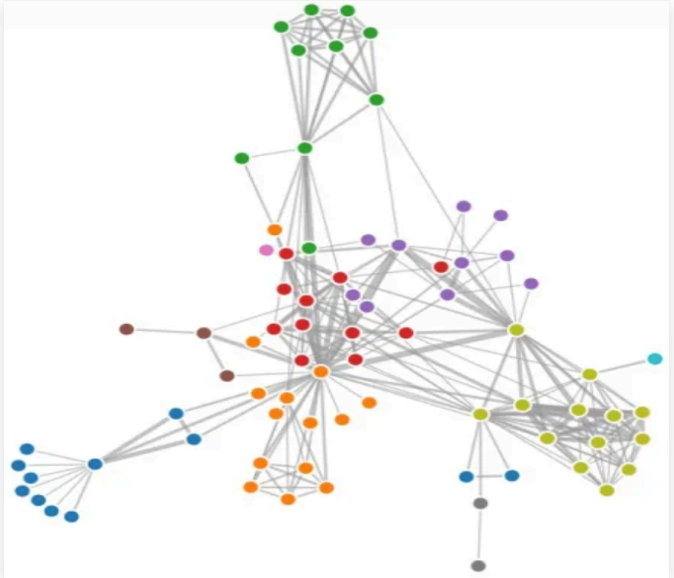
单一结构

复合结构

复杂结构串联







体　　系　　篇

高可用指导思想

01

系统 -> 业务

- 追求业务连续性
- 以不可用等级为评判标准
- 追求业务收益与线上风险之间的平衡

02

风险 -> 事故

- 减少风险点
- 控制风险转化为事故
- 有风险必有监控，有监控必有预案，有预案必经验证

03

架构升级

- 通过架构升级，沉淀标准的业务模型 、技术解决方案 ，降低人为错误几率 ，提升系统的可用性

04

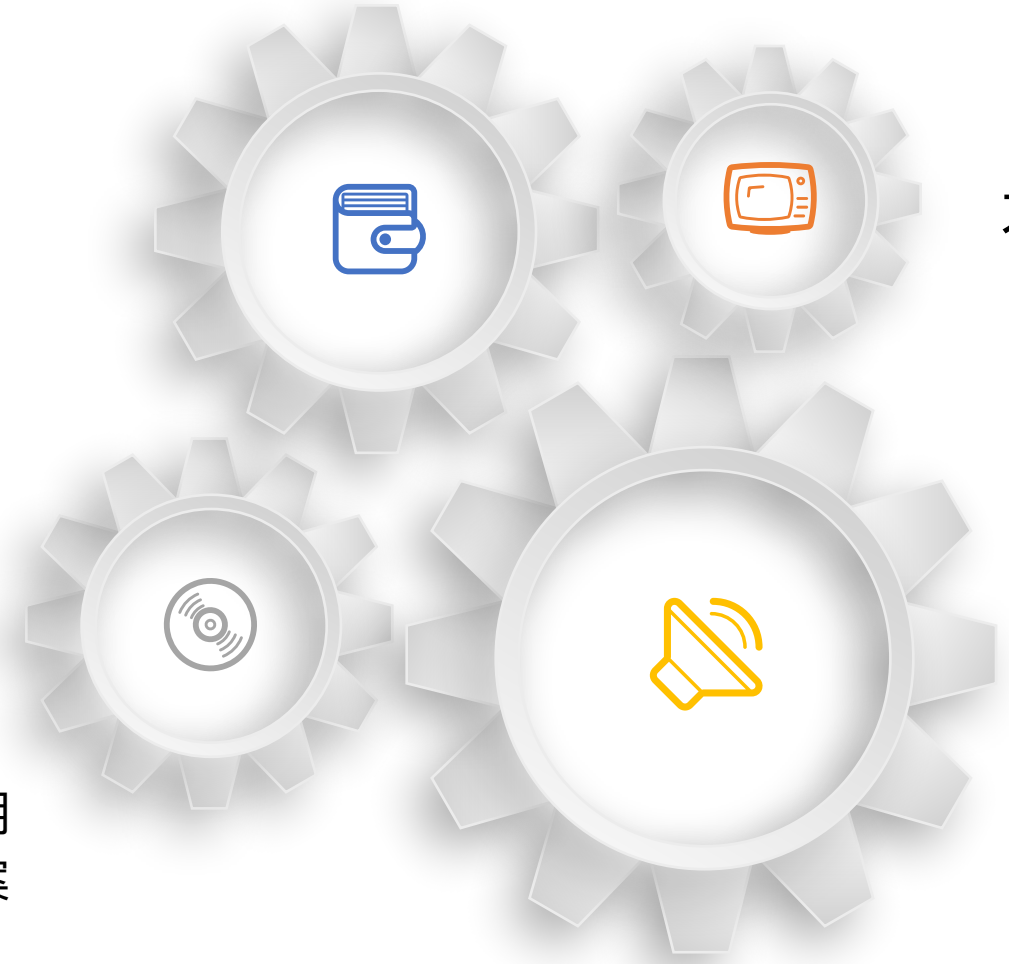
数据驱动

- 通过业务数据形成业务系统风险数据模型
- 借助数据模型+AiOps强化风险预警与提前部署预案能力

高可用考虑因素

成本

- 研发成本
- 管理成本
- 资源成本



不可用等级

系统

- 软件生命周期
- 现有解决方案
- 容错性
- 体系完善度（监控覆盖度、上线SOP、预案）

人

- 认知、能力要求
- 组织架构

高可用面临的挑战

识别

如何**尽早识别**未知风险

预防

如何**全面预防**事故发生

感知

如何**尽快感知**事故

定位

如何**精准定位**问题

解决

如何快速、有效解决问题

高可用体系


识别
指标体系
混沌工程
报警体系
01
预防
02
研发规范
安全漏洞修复
容量保障
代码质量提升
报警收敛
容灾
复盘
06
自动化
平台化
体系
打造
03
感知
监控大屏
上下游沟通机制
值班机制
解决
05
日常/大促预案
硬件异常预案
业务异常预案
大促预案
异常执行规范
04
定位
硬件监控平台
链路监控平台
业务监控平台

识别 – 指标体系

场景层		用户访问次数及时长统计 场景流量
应用程序层	交互	QPS TPS
	内部	消息积压
	质量	错误码统计 报警数量
基础设施层		CPU Memory 负载 线程数 I/O Statistics Network

识别 – 混沌工程

MYSQL故障

REDIS故障

RPC故障

CPU负载故障

磁盘故障

网络延迟故障

预防 – 研发生命周期

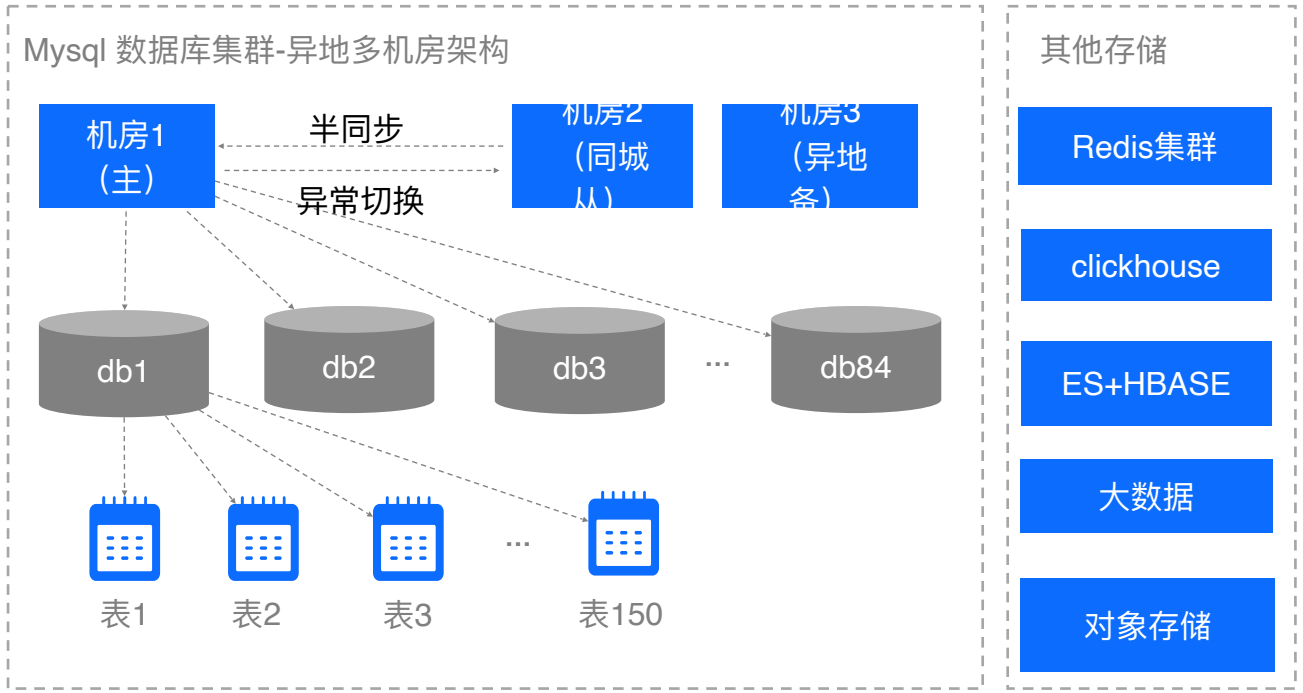

需求确认
业务本质确认
结构影响评估
承载系统确认
设计
架构设计
组件设计
技术方案设计
开发
代码质量提升
报警收敛
研发生命周期
测试
单侧覆盖
流量回放
上线
上线SOP
灰度上线
白天上线
Serverless
运营
指标体系
监控大屏
预案
备战

预防 – 弱依赖

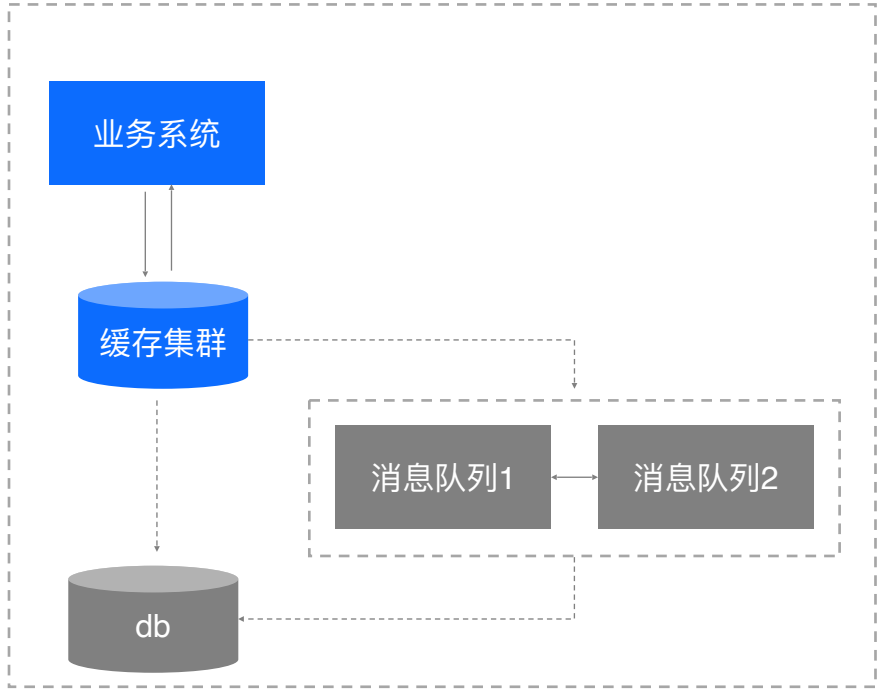
基于 MySQL自身高可用， 以及一定的 容灾设计（如热备、读写分离、冷热分离、内存DB、延时异步入库等），

最终达到数据库整体架构的高可用

强依赖数据库架构



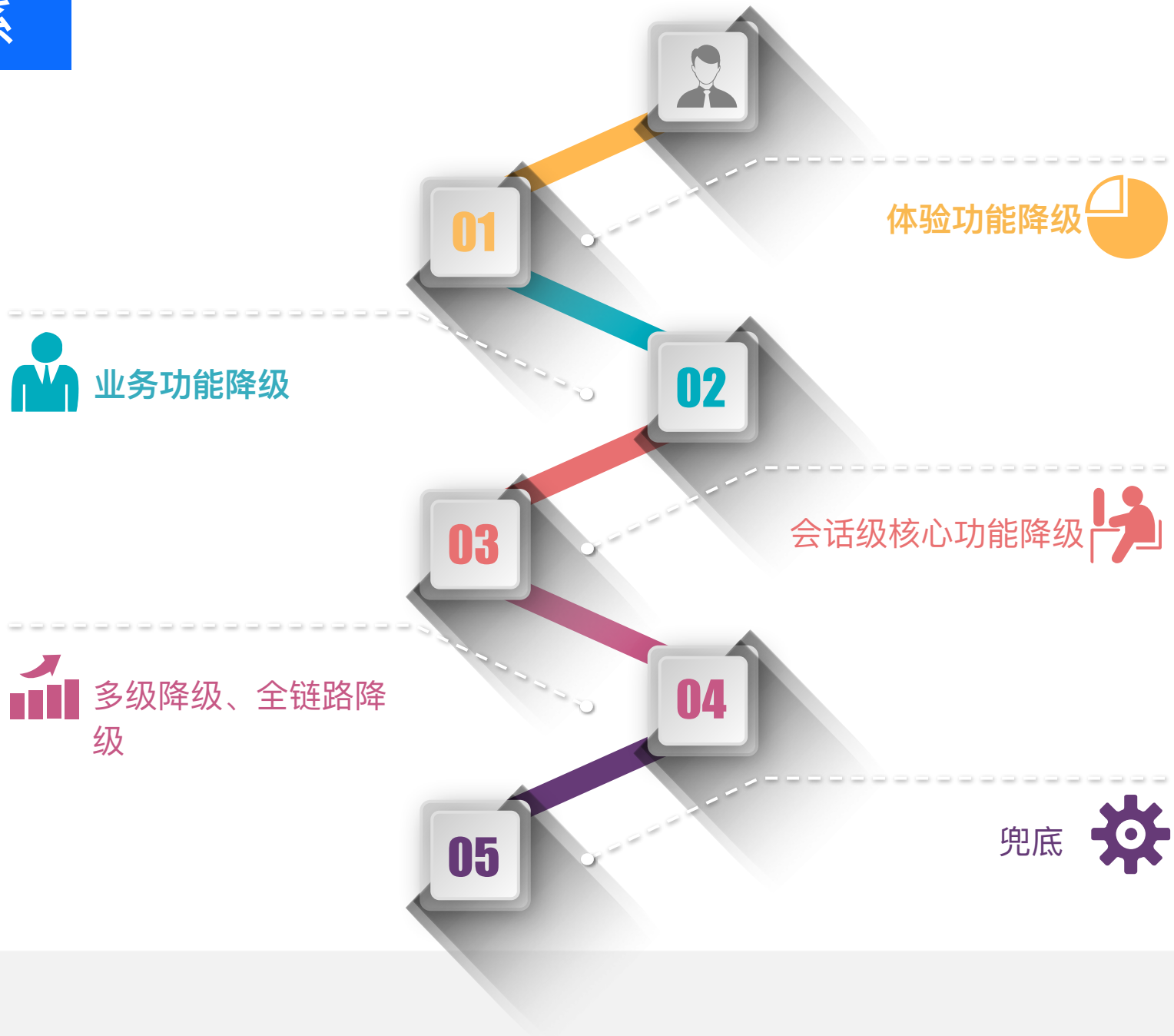
弱依赖数据库架构



定位 – 全域可观测

- 多层级监控体系
- AIOPS

解决 – 降级体系



成败藏在细节里

- https的证书大小是多少
- 已经设置了有效期的缓存如何设置为不过期
- 影响TPS的因素到底有哪些， 影响的程度分别是什么

架　　构　　篇

研发的本质

软件的本质是基于对业务本质理解进行的 业务建模 并打造描述业务模型的 复杂概念结构 ，

结合业务发展持续演进， 使用 语言 表达这些概念结构 ， 在空间和时间限制下将他们映射成

机器 语言

方法论之争

业务 VS 技术

是什么 VS 怎么做

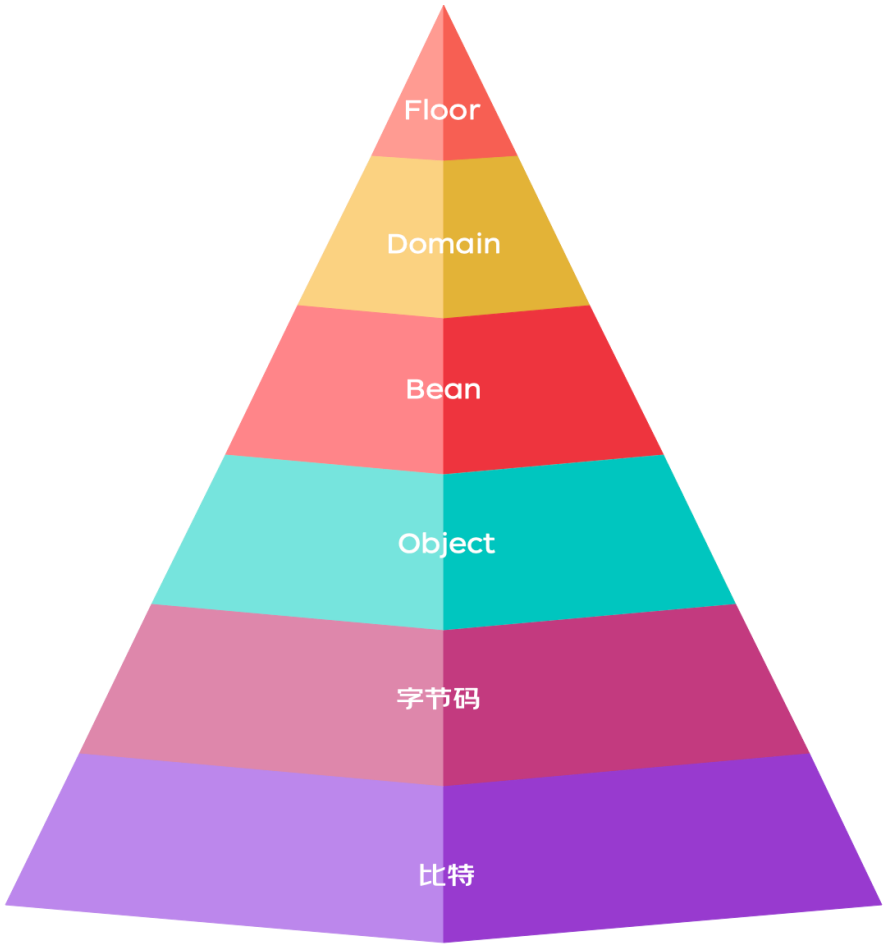
语言 VS 机器

lambda calculus VS 图灵机

语言的演进

计算机以 二进制系统 为基础 ， 并使用 比特 去描述和映射世界， 而 万物皆比特 ， 语言的终点是 **DSL**

- 视图模型语言层 【 Floor 、Structure 】
- 业务模型语言层 【 Domain 、Relation 】
- Spring Framework
- Java
- JVM
- 冯.诺依曼机



- 万物皆楼层 Floor
- 万物皆域 Domain
- 万物皆 Bean
- 万物皆 对象
- 万物皆 字节码
- 万物皆域 01

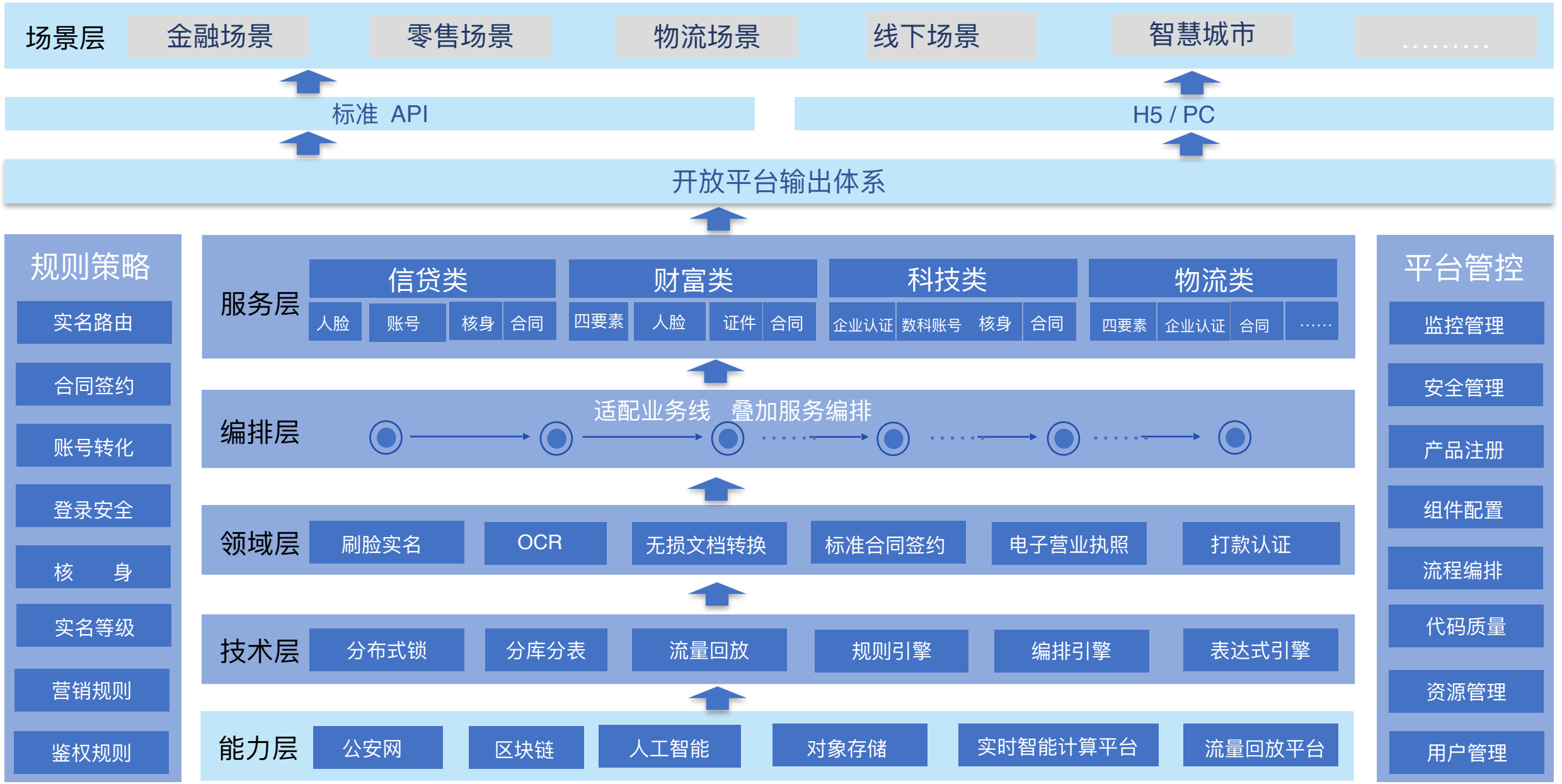
曲高和寡 λ

- 抽象程度 越来越高
- 表达能力 越来越强
- 业务属性 越来越强
- 逻辑性 越来越强

普适 【 图灵机 】

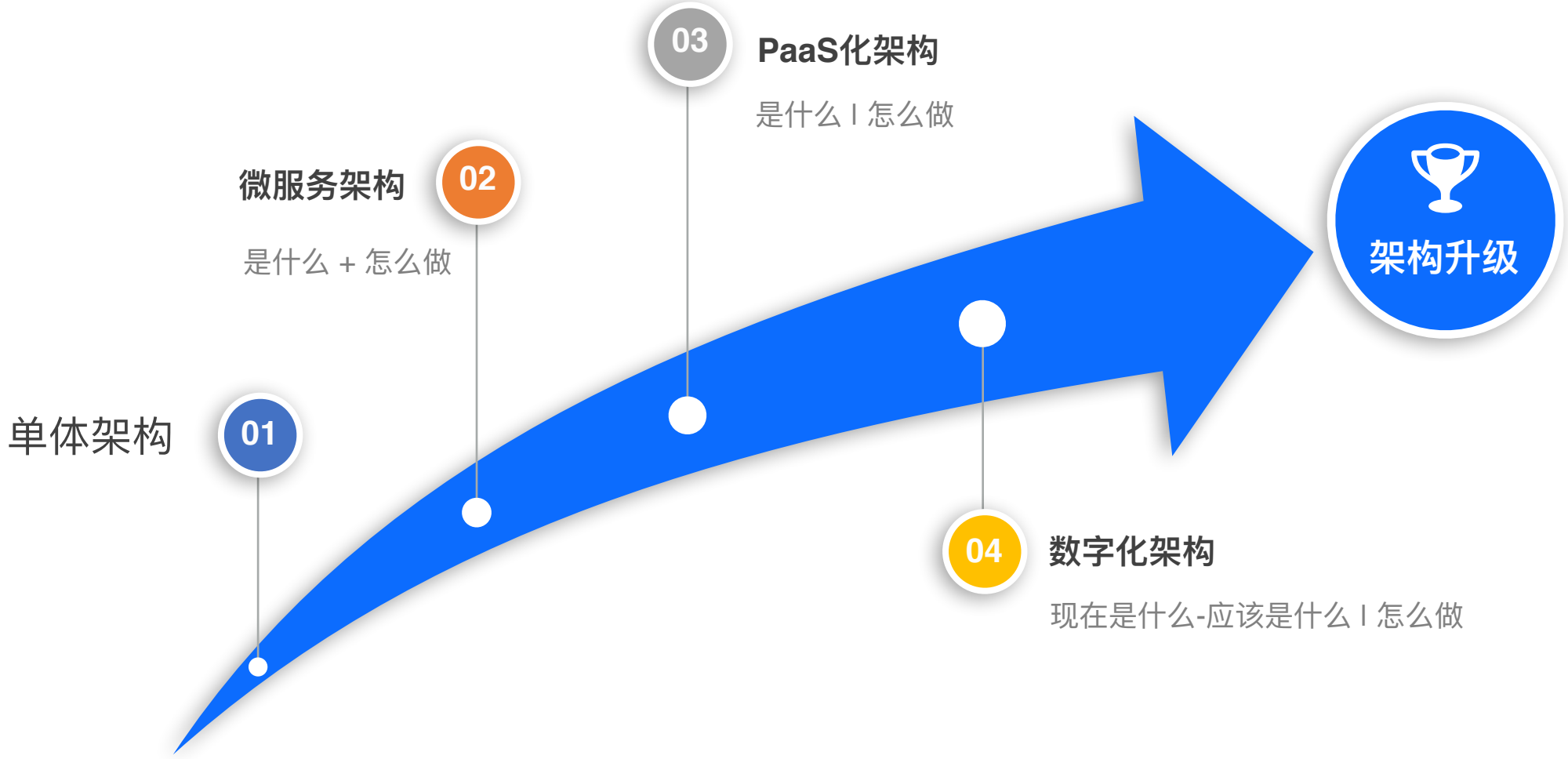
- 表达范围 越来越强
- 硬件属性 越来越强

PaaS化架构


场景层
金融场景
零售场景
物流场景
线下场景
智慧城市
.....
标准 API
H5 / PC
开放平台输出体系
规则策略
实名路由
合同签约
账号转化
登录安全
核 身
实名等级
营销规则
鉴权规则
服务层
信贷类
人脸
账号
核身
合同
财富类
四要素
人脸
证件
合同
科技类
企业认证
数科账号
核身
合同
物流类
四要素
企业认证
合同
.....
编排层
适配业务线 叠加服务编排
领域层
刷脸实名
OCR
无损文档转换
标准合同签约
电子营业执照
打款认证
技术层
分布式锁
分库分表
流量回放
规则引擎
编排引擎
表达式引擎
能力层
公安网
区块链
人工智能
对象存储
实时智能计算平台
流量回放平台
平台管控
监控管理
安全管理
产品注册
组件配置
流程编排
代码质量
资源管理
用户管理

PaaS化架构特点

架构未来演进


03
PaaS化架构
是什么 I 怎么做
微服务架构
02
是什么 + 怎么做
单体架构
01
架构升级
04
数字化架构
现在是什么-应该是什么 I 怎么做

总结

- 高可用是一个零和游戏 ， 只有0和100分 ，成败藏在每一个细节里
- 高可用需要一套持续迭代的 体系 来保障
- 体系是通过企业级 架构 的升级来实现， 提升可用性的同时，降低系统复杂度和研发成本
- 高可用的背后是业务与技术之间的博弈和平衡， 在数字化和云原生的时代， 演进的方向是从矛盾的对立走向统一， 实现业务与技术的双轮驱动