第二届中国eBPF开发者大会

WWW.ebpftravel.com

汽车软件性能提升方法的工程化落地

中国·西安

张旭海

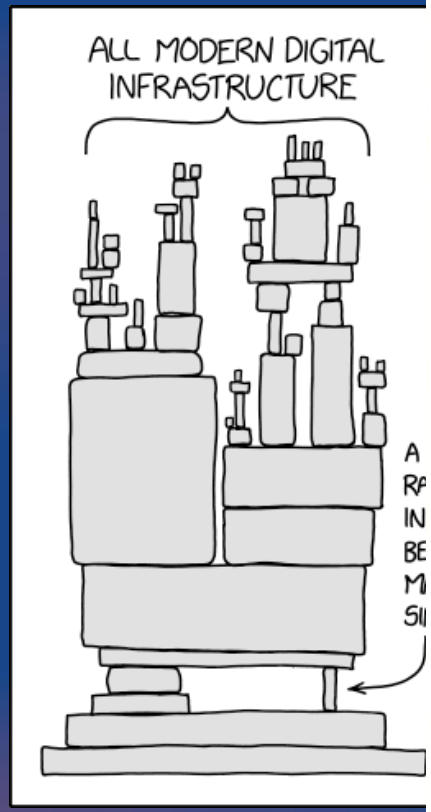
Thoughtworks 安全与系统研发事业部

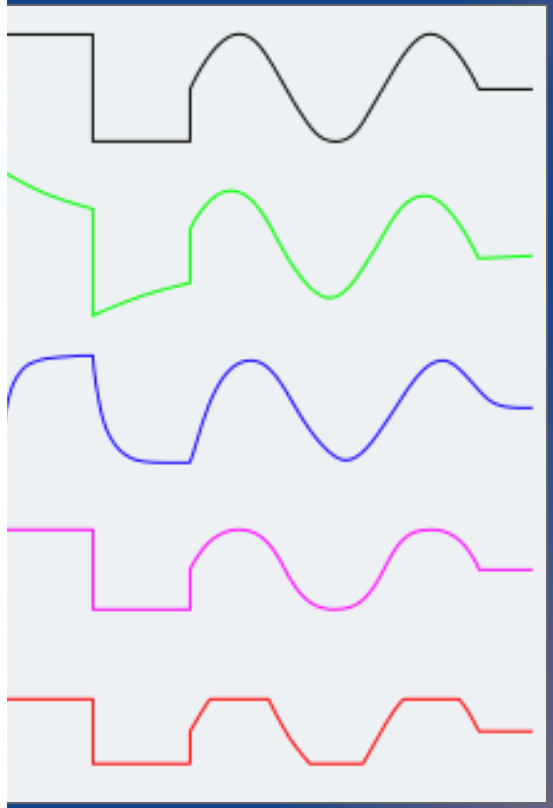
架构师 & 专家咨询顾问

在性能工程、软件架构设计、云原生基础设施、企业数字化等领域拥有8年咨询和交付经验。关注并聚焦于云原生、分布式以及系统研发等技术领域，是活跃的技术博客写作者，技术文章翻译者和开源软件贡献者。

软件的本质复杂度

本质复杂度是指待解决问题本身特

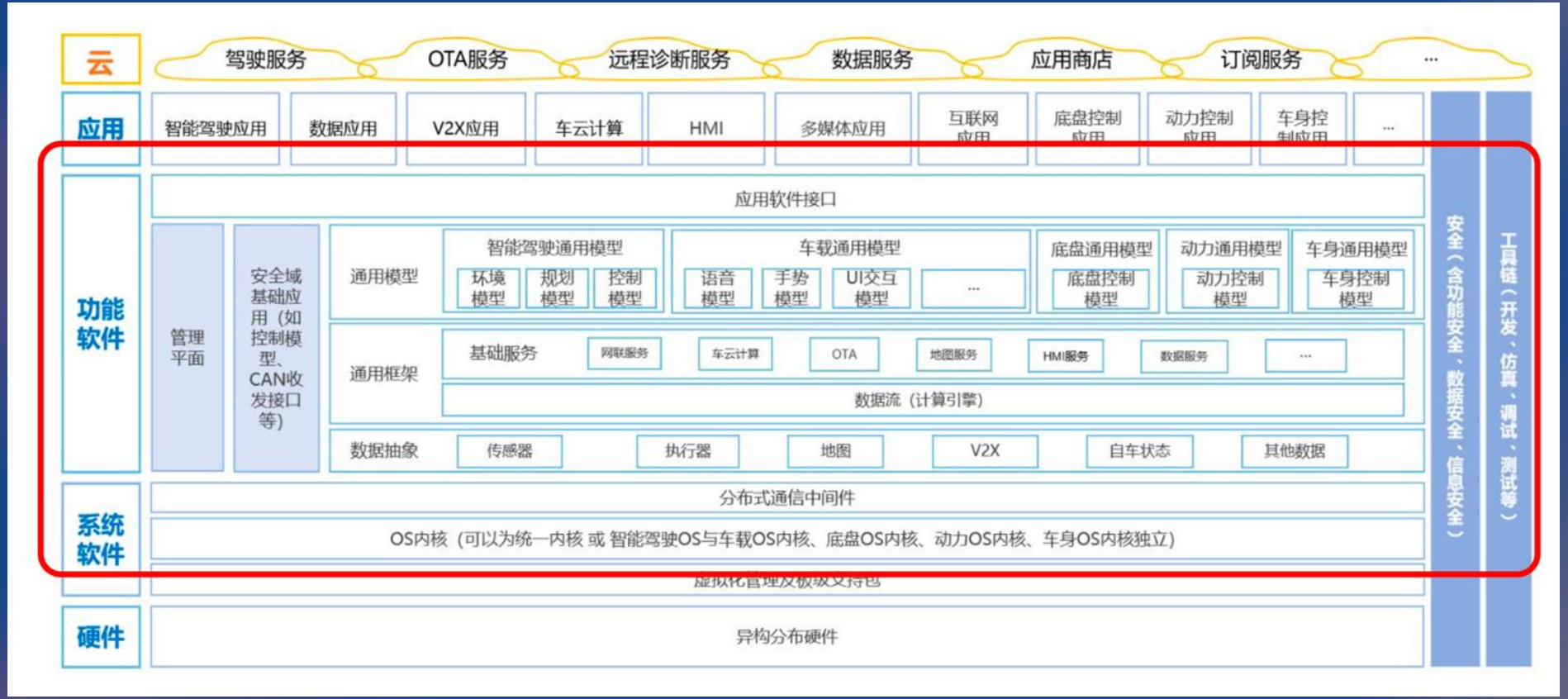




依赖性：牵一发

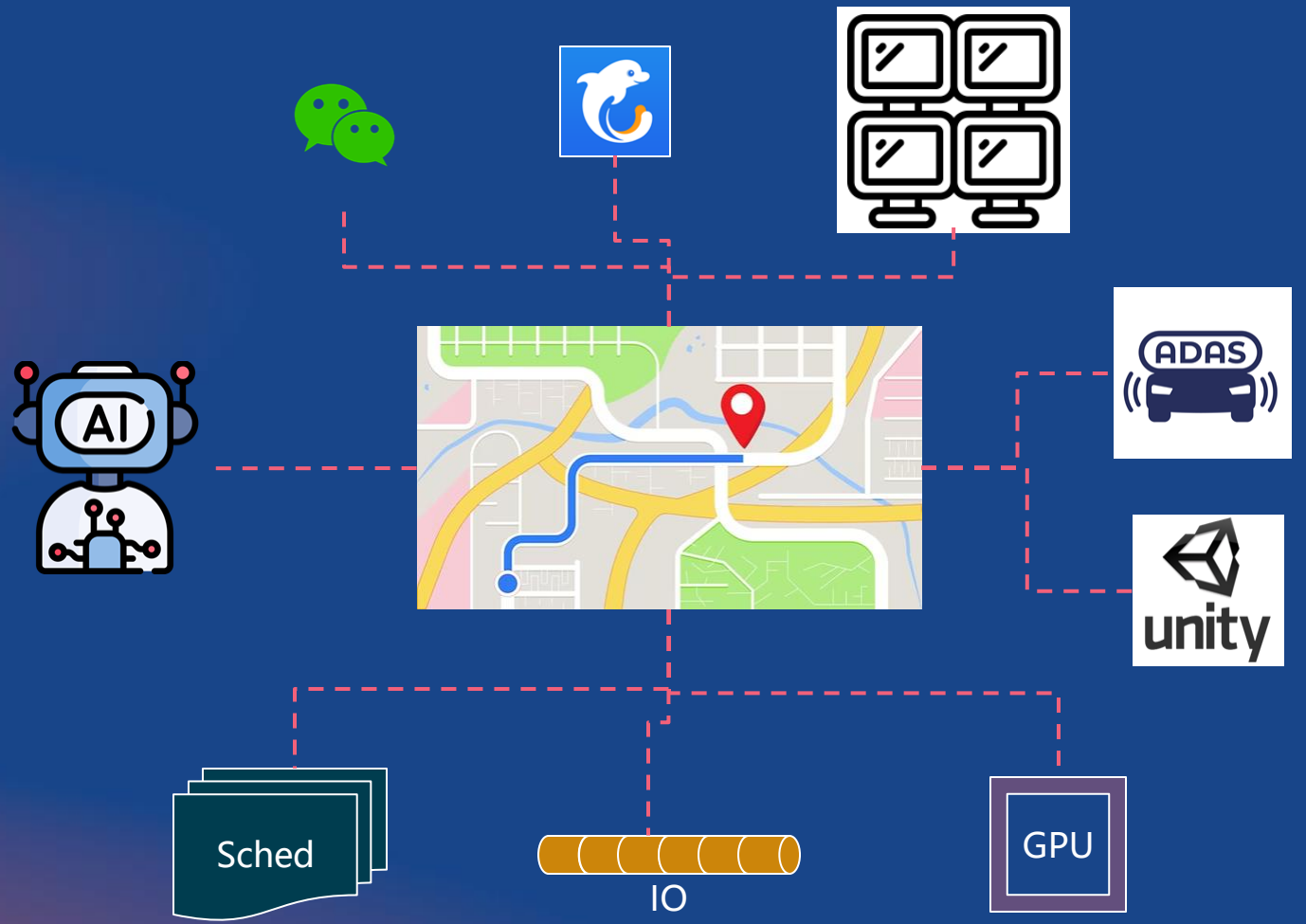
知识传递失真

软件定义汽车的领域复杂性

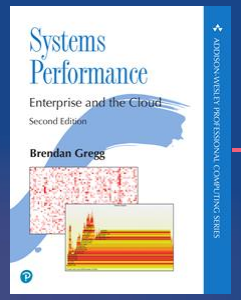


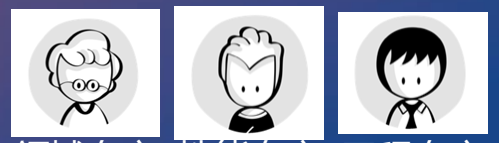
图源：智能网联汽车电子电气架构产业技术路线图

持续性能提升的跨领域挑战



持续性能提升的工程化方法





系统方法论

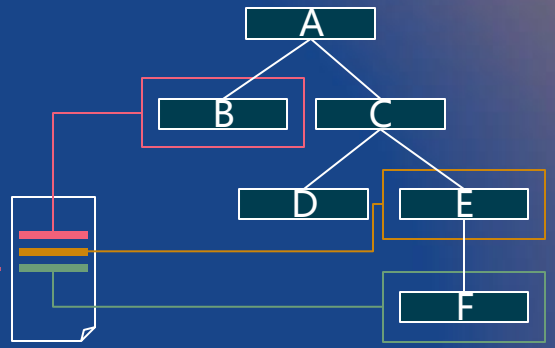
标准化流程和规范

成熟的技术支撑

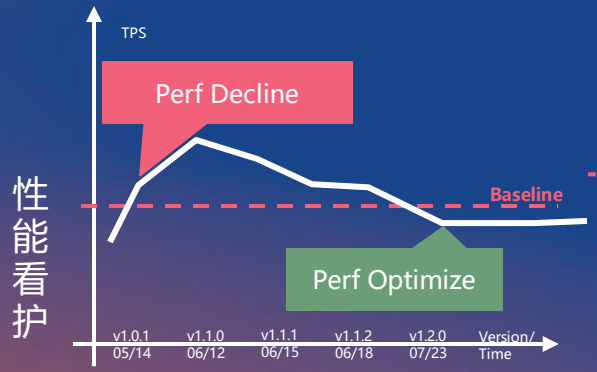
全生命周期管理

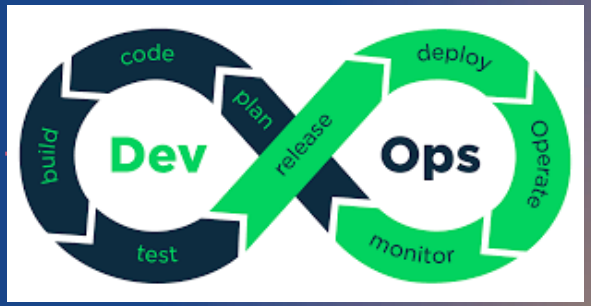
持续改进

工程化方法



建模：评估模型
自动化：适应度函数





构建研发团队自主驱动的性能工程反馈闭环

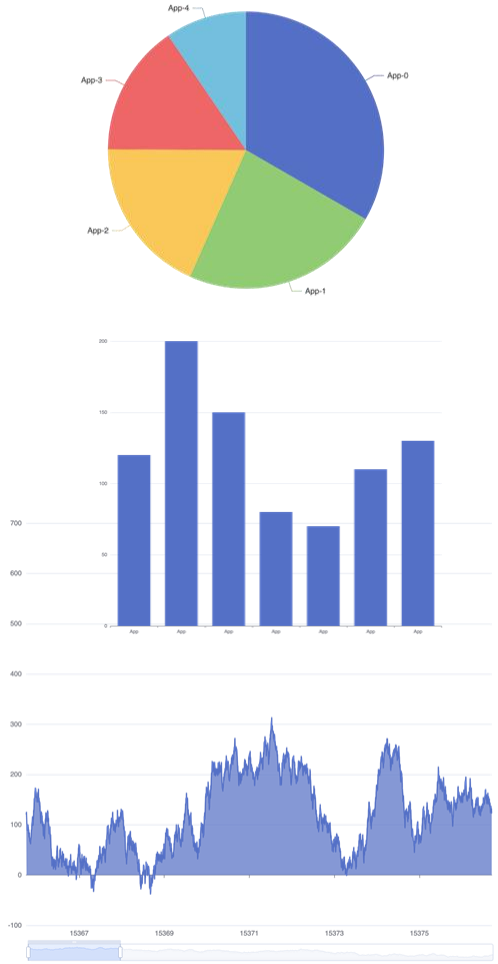
性能工程实践

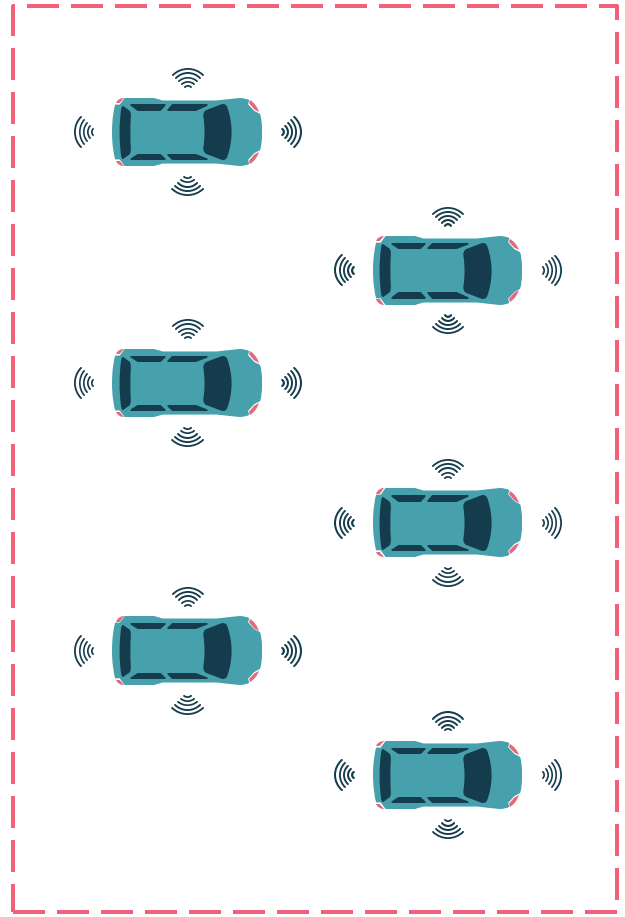
工程实践 1：持续性能观测

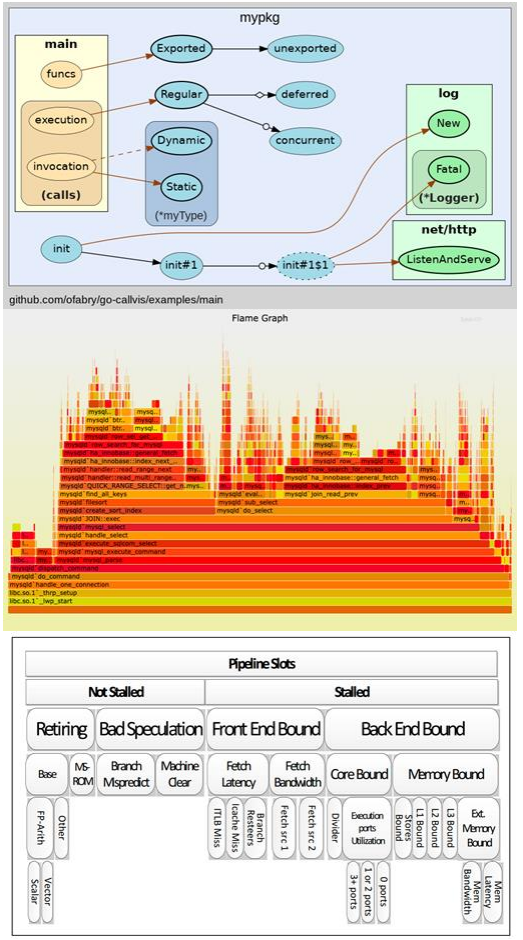
工程实践 1：持续性能观测

OpenTelemetry

Matrix







eBPF

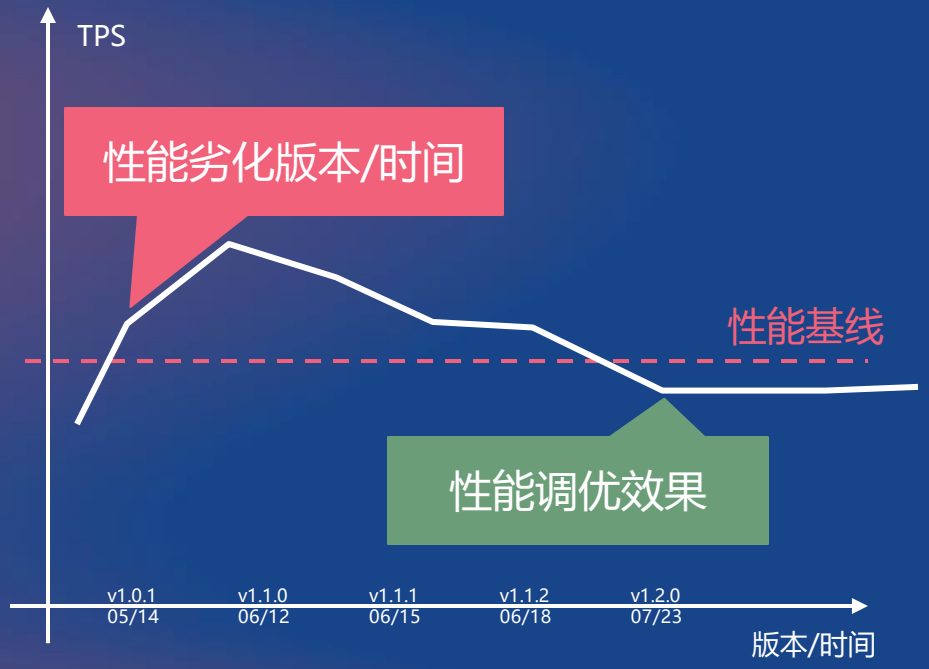
Tracing

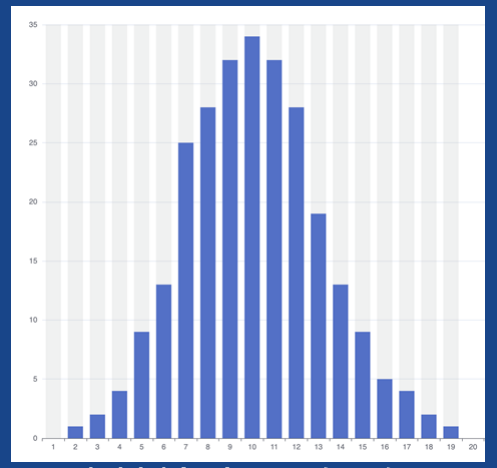
perfetto

For Evaluation

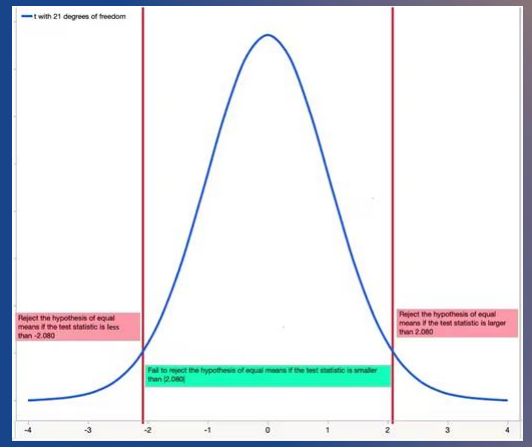
For Problem Solving

工程实践 1：持续性能观测

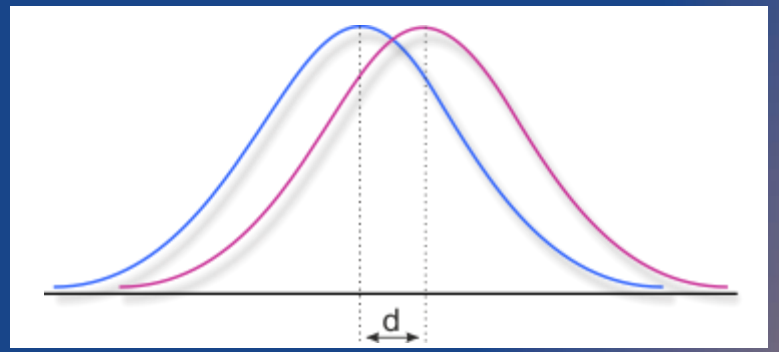




正态性检验 + 数据矫正



t 检验

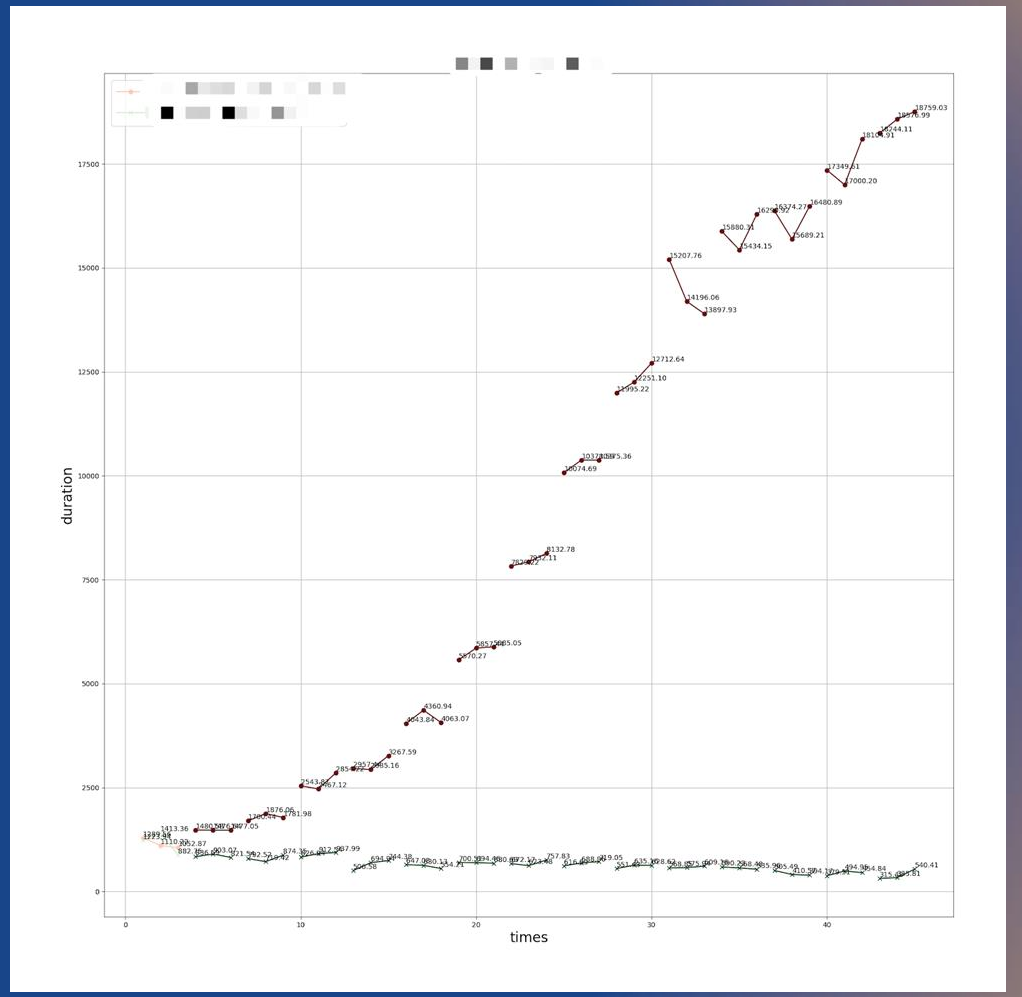


效应量

工程实践 2：持续性能提升

多屏流畅度体验提升

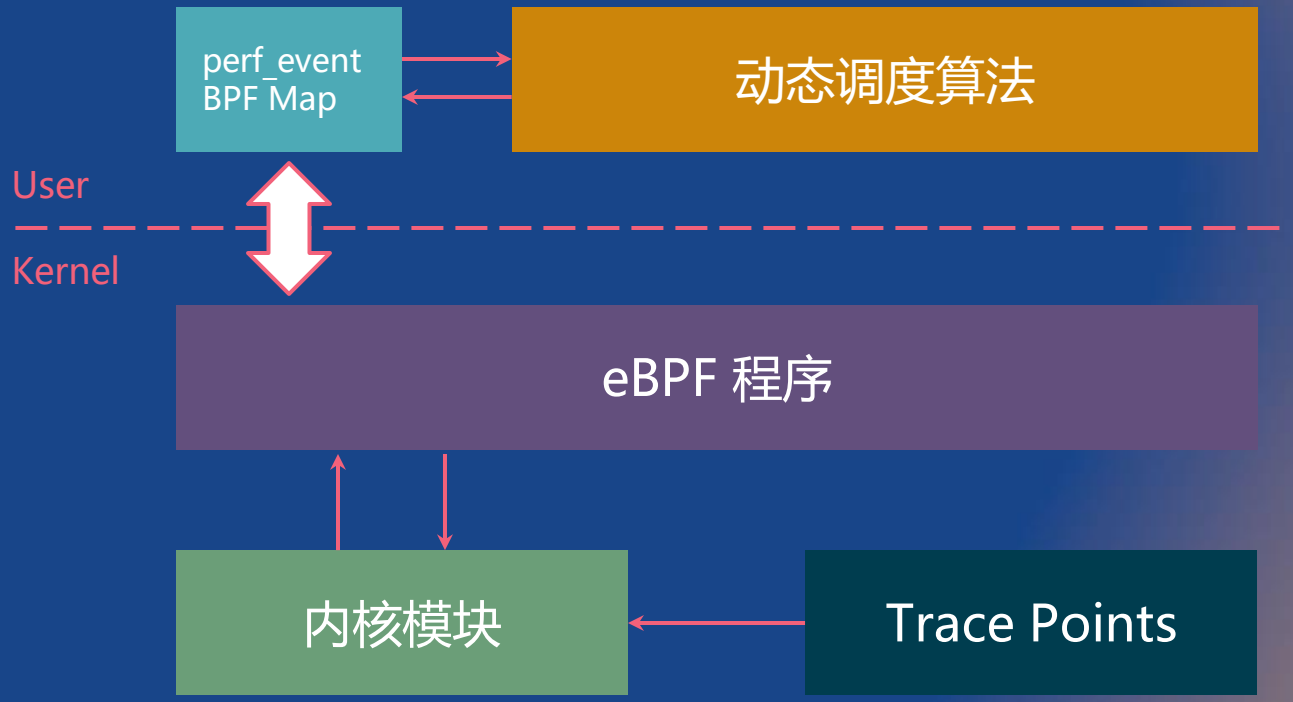
- 控制调度延迟：确保 UI 渲染流畅
- 高优先级组：前台应用拥有更多资源优先级
- 极端压力场景：低内存、高 IO 延迟
- 屏幕优先级：多屏功能区划分



工程实践 2：持续性能提升

动态调度微调

- 按场景划分：行车/休闲/哨兵
- 按业务划分：关键服务分组
- 按前后台划分：前台应用优先

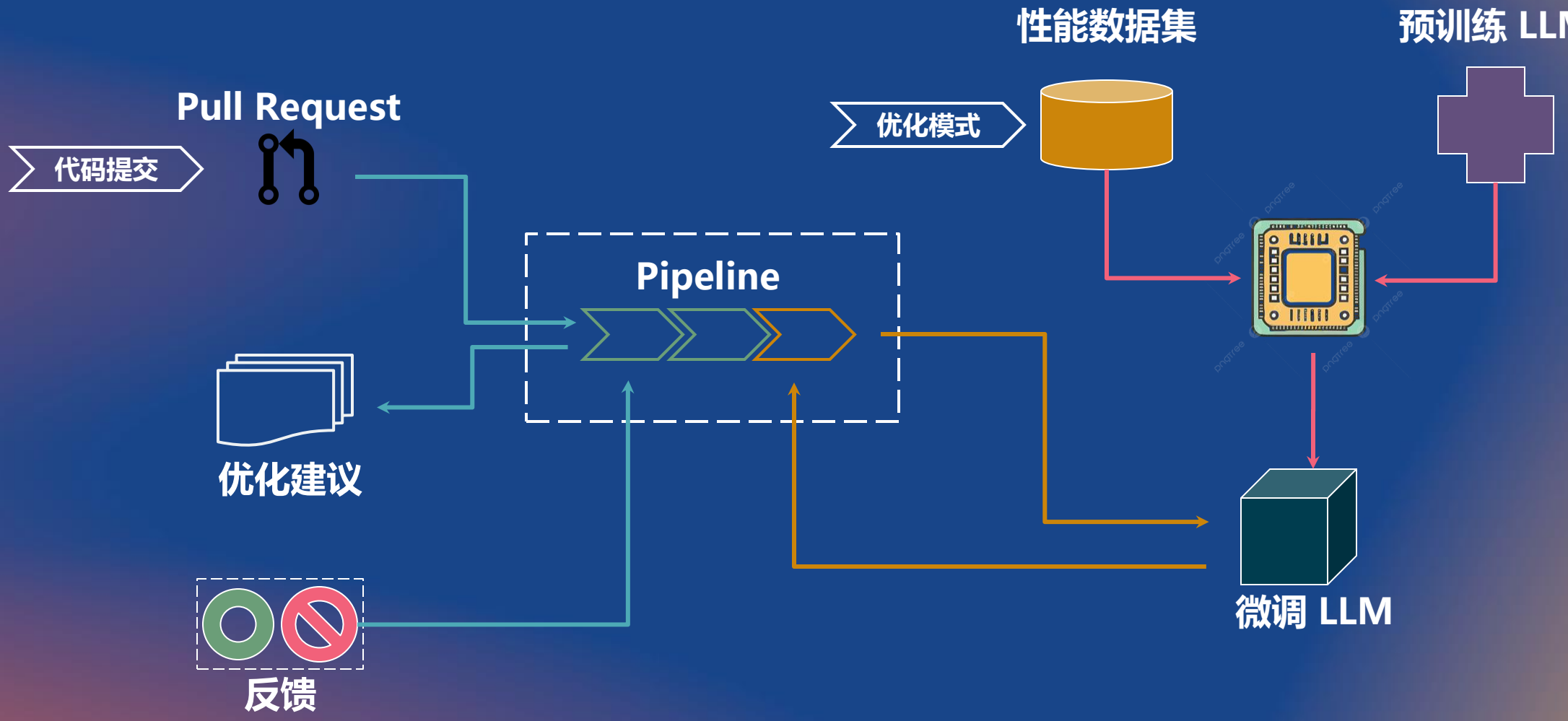


工程实践 3：AI 代码性能优化

Model + Configuration	%Optimized	Agg. Speedup	%Correct
Same Human, Best Edit	100%	3.64	100%
Best of all Humans in the Dataset	**100%**	**4.06**	100%
GPT4 With Instruction Prompts, Best@1	8.6%	1.15	93.45%
GPT3.5 With Chain of Thought Prompts, Best@1	21.78%	1.26	67.01%
Fine-Tuned CodeLlama w/ Perf-Conditioning, Best@1	31.87%	2.95	38.7%
Fine-Tuned Gpt3.5 w/Self-Play, Best@1	**45.62%**	**3.02**	61.71%
Fine-Tuned CodeLlama w/ Perf-Conditioning, Best@8	66.60%	5.65	71.08%
Fine-Tuned Gpt3.5 w/Self-Play, Best@8	**87.68%**	**6.86**	95.11%

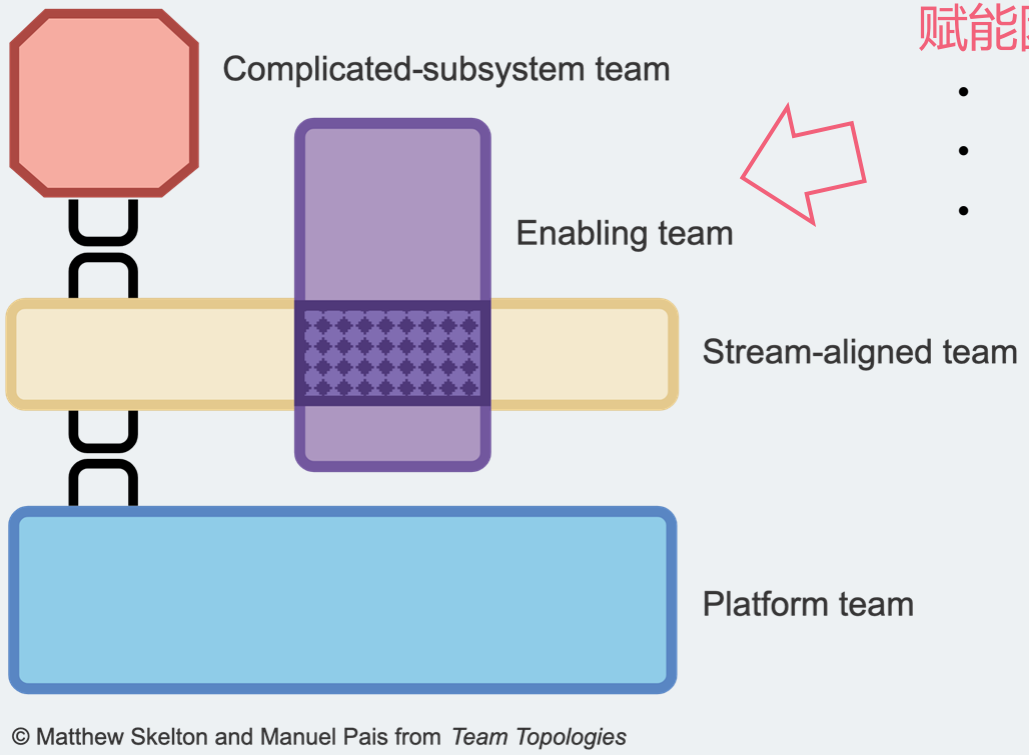
```
Below is a program. Optimize the program and provide a more efficient version.

### Program:
S = eval(input())
T = eval(input())

cnt = len(S)-len(T)+1
res = []

for i in range(cnt):
    res2 = 0
    for j in range(len(T)):
        if S[i+j] != T[j]:
            res2 += 1
    res.append(res2)
print((min(res)))

### Optimized Version:
S = eval(input())
T = eval(input())
res = 0

for i,j in zip(S,T):
    if i != j:
        res += 1
print(res)
```

工程实践 3：AI 代码性能优化



性能工程团队拓扑

复杂子系统团队：
- 系统侧性能优化
- 性能评估
- 定义参考性能指标



赋能团队：
- DevPerfOps 赋能
- 落地性能实践
- 辅助分析性能问题

平台团队：
- 构建性能工具集
- 实施性能工程平台能力

性能工程体系平台

通过工具平台构建性能分析调优能力，帮助产品线更快识别性能问题，分析性能瓶颈，并完成性能调优验证，通过 6大能力 单元的共建一站式 可编排 智能化的性能工程平台

建模

排队模型 | 分发模型 | 分支模型
缓存模型 | 拷贝模型 | 循环模型

数据

基线设定 | 劣化识别 | 分析推荐 | 性能优化
数据平台 | 知识图谱

仿真

ARM | x86 | Qemu | Docker
仿真环境管理

研发流水线反馈闭环

采集/观测

perf | valgrind
bcc | perfectto
sar | ftrace

分析优化

火焰图 | 函数分析 | 系统性能 | 组件定界
调用图 | 访存分析 | 组件性能 | 函数定界
Top-Down | 指令分析 | 算法性能 | 符号定界

看护

测试用例 | 流水线
看护对象 | 性能门禁
监控指标 | 自动提单

流程端到端

通过看护、定界驱动问题发现，落实责任

基于知识图谱建立性能建模、分析、优化流程

开放API/UI

通过API、UI控件的方式将原子能力通过内源平台开放，各产品业务线可以对原子能力进行编排定制，形成自己的性能工程能力

产品线A

ftrace | 函数调用图
网络模型 | 网络性能测试用例

产品线B

perf | I/O火焰图
缓存模型 | 高速存储性能用例

性能工程成熟度模型

	需求/建模/设计	开发/编码	测试/看护	监控/发现	分析/优化/解决
L3 高效	1. 针对性能指标进行预先的建模、计划和规格选取 2. 可以预测需求变更带来的性能变化影响 3. 针对性能问题有专门的架构演进设计文档记录，并与实现保持一致	1. 开发者有工具可以自主化识别性能问题 2. 开发者有可参考的高性能编码实践 3. 针对性能约束高的场景存在通用解决方案，形成库和框架 4. 知识库管理成体系，方便开发人员参考和推广。	1. 针对设计阶段定义的系统性能指标可做每个版本（包括开发中期版本）各维度的持续看护 2. 可以自动化执行性能测试用例，并及时反馈给开发团队 3. 性能看护有足够的工具集成，可定位系统发生性能劣化的位置 4. 性能劣化的结果有相应组件的责任人可及时进行问题的修复或缓解	1. 可及时动态地对运行中的系统下发指标收集任务 2. 可以及时发现并对潜在的性能指标持续降级进行监控告警 3. 可以及时获取性能异常信息及运行时上下文，以供后续性能问题原因排查 4. 针对性能降级有缓解策略	1. 积累性能问题的优化成功案例，持续赋能团队 2. 通过数据统计与挖掘，进行性能优化模式的识别和抽象提炼 3. 针对领域内的性能反模式与常见优化思路，形成方法 4. 完善的业务模型到组件服务的架构图。
L2 量化	1. 有明确的性能需求基线 2. 针对性能目标有相应的设计考量 3. 可以提前设计量化和度量系统性能指标 4. 可以通过量化的指标，对系统性能进行各维度的用例约束	1. 开发者对可能导致性能问题的编码反模式有相应的知识，并能做出针对性能问题的避免。 2. 开发过程可以对性能要求进行自检，并及时通过特定测试用例及时获取反馈 3. 有一些零散的知识库文档供开发人员从参考。	1. 团队有制定针对不同产品/业务的性能基线测试。 2. 有准备好的数据和环境，但是不能自动化，测试结果归档不统一。 3. 有性能测试的活动，但是没有做到自动化。 4. 性能测试工具散乱。没有针对不同场景的性能工具归类。	1. 有针对系统性能指标的监控，可以形成时序性量化指标的可视化 2. 可以针对跨多组件和链路的性能问题进行问题定界 3. 有一些动态监控方案，但是不足够灵活，不能做到全链路的动态的加载卸载	1. 积累性能优化的知识图谱 2. 可以针对特定的性能问题基于知识图谱找到解决方案与思路 3. 在团队内对性能目标与优化方向达成共识 4. 有业务模型->软件系统架构->组件或服务的整体架构梳理，但不够细粒度。
L1 无	1. 存在大量设计缺陷导致的性能问题 2. 需求阶段无法获得目标性能指标，纳入需求规约 3. 需求阶段未计划解决性能问题的投入	1. 开发者对可能存在性能问题的编码无感知 2. 开发过程无针对性能要求的设计与自检 3. 编码实现存在大量性能设计反模式（如大量IO/重复对象/阻塞等） 4. 没有对应的提升性能的案例和知识库的管理。	1. 没有对应不同产品/业务的Benchmark基线测试。 2. 没有性能测试的CI，不能自动化的预警性能瓶颈。 3. 没有事先准备好的测试数据和测试环境，测试结果没有很好的管理。 4. 没有评估和选择最合适的测试工具。	1. 没有对性能指标的量化进行可视化，以及辅助的观测数据（堆栈/时延/调用链路等）。 2. 没有动态监控或发现性能问题的机制。 3. 缺乏全链路的监控和发现手段。	1. 缺乏性能优化思路 2. 性能优化方法依赖个人经验 3. 针对系统性能优化方向与目标缺乏团队共识 4. 存在过度设计导致的性能问题 5. 没有业务模型->软件系统架构->组件或服务的整体架构梳理。

第二届中国eBPF开发者大会

WWW.ebpftravel.com

谢谢！

张旭海 @Thoughtworks 安全与系统研发事业部

中国·西安