openEuler

SJTU

# openEuler Summit 2023

# The Gap between Serverless Research and Real-world Systems


杜东

IPADS Lab，上海交通大学

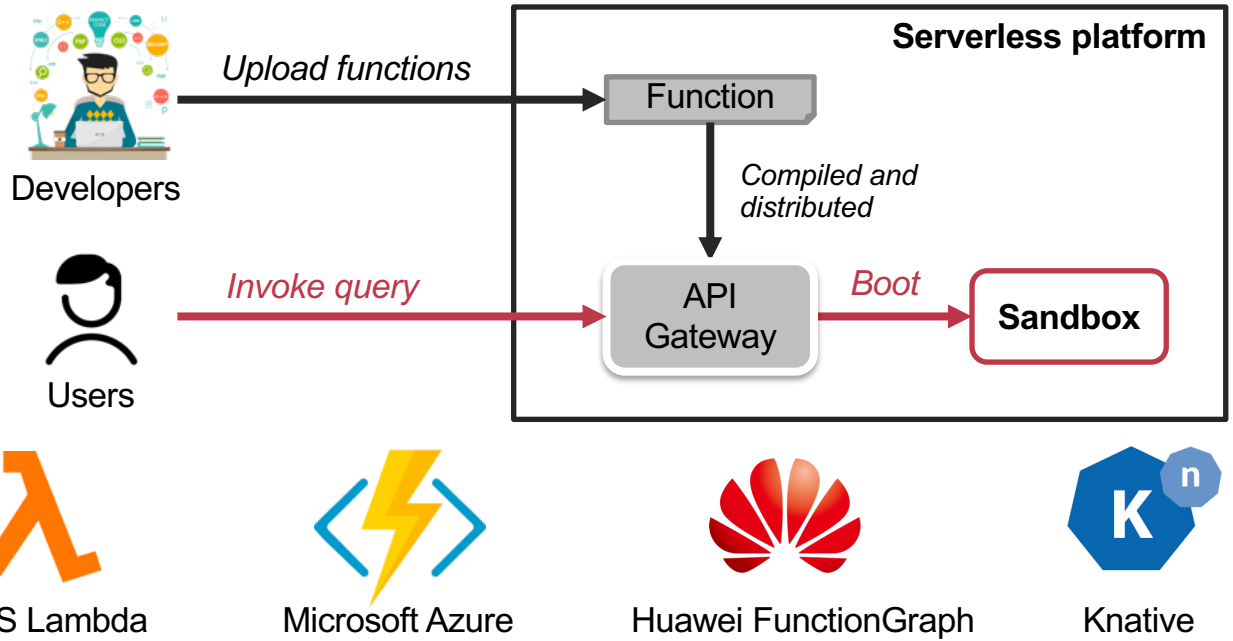

IPADS INSTITUTE OF PARALLEL AND DISTRIBUTED SYSTEMS

Huawei Cloud FunctionGraph


合作者：柳清源、夏虞斌、张平、冯嘉、陈海波等

# 服务器无感知计算（Serverless Computing）

- **什么是Serverless计算:**
  - Write code; upload code; invoke; deploy an instance and execute; destroy the instance



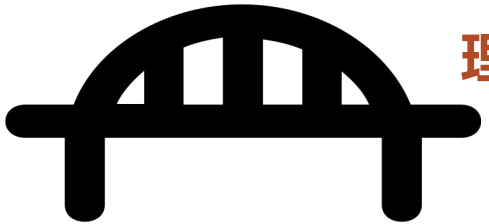
# Serverless研究已经取得大量的成果和进展

- **优化冷启动时延: 从2018到2023**
  - **>10**的顶会论文对函数实例的启动时延进行优化，将指标从数百毫秒降低到最优亚毫秒以内


- **其他工作：优化通信时延、优化调度策略（提升资源利用率）**

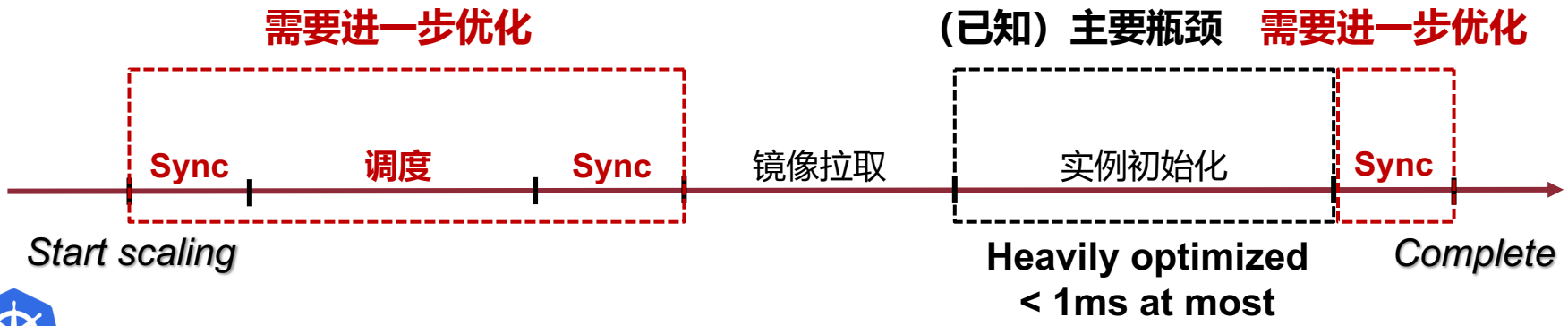
**依然存在差距**

**当前Serverless工作**



**理想的、实际的高性能Serverless系统**

# 案例：冷启动开销



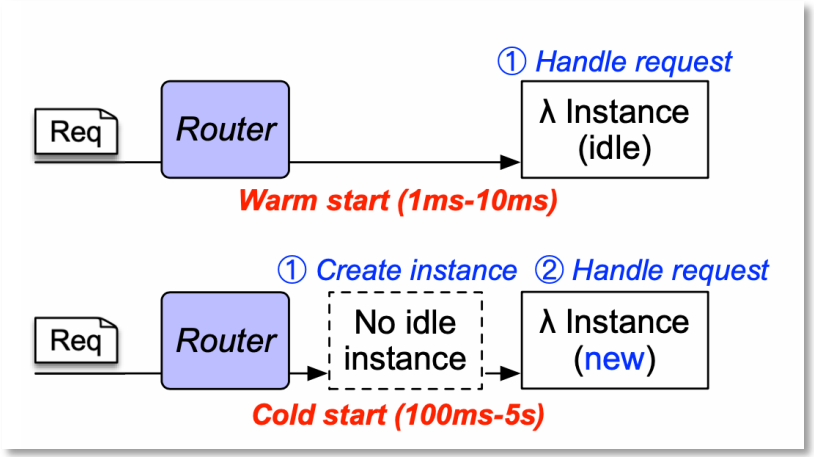

- **实例初始化:**
  - 已经优化至亚毫秒级别
- **忽略的部分:**
  - 数十到数百毫秒时延
  - **成为新的瓶颈!**

# 内容总览

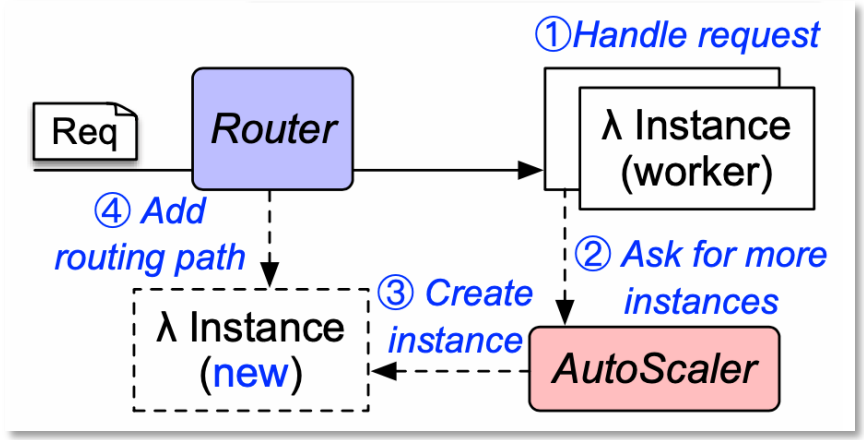
- **方法论:**
  - 从学术界和工业界（华为云合作）协同的视角，分析并给出当前Serverless领域的关键挑战和对应的洞察
- **5个Serverless领域在下一阶段的关键挑战**

| Challenges | Research works | Real-world systems |
|---|---|---|
| Cold start | Synchronous | Asynchronous |
| Declarative tax | Hardly consider | Non-trivial cost |
| Scheduling cost | <100ms (<120 nodes) | >10s (2K nodes) |
| Scheduling policy | Single policy | ≥20 policies |
| Sidecar | Hardly consider | Non-trivial cost |

# 技术挑战1：从同步启动到异步启动




- 同步启动（Synchronous Start）




- 异步启动（Asynchronous start）

# 技术挑战1：从同步启动到异步启动

- **同步启动会直接影响尾时延（tail latency）**
  - 一个常见的，不一定正确的假设（基于同步启动）：

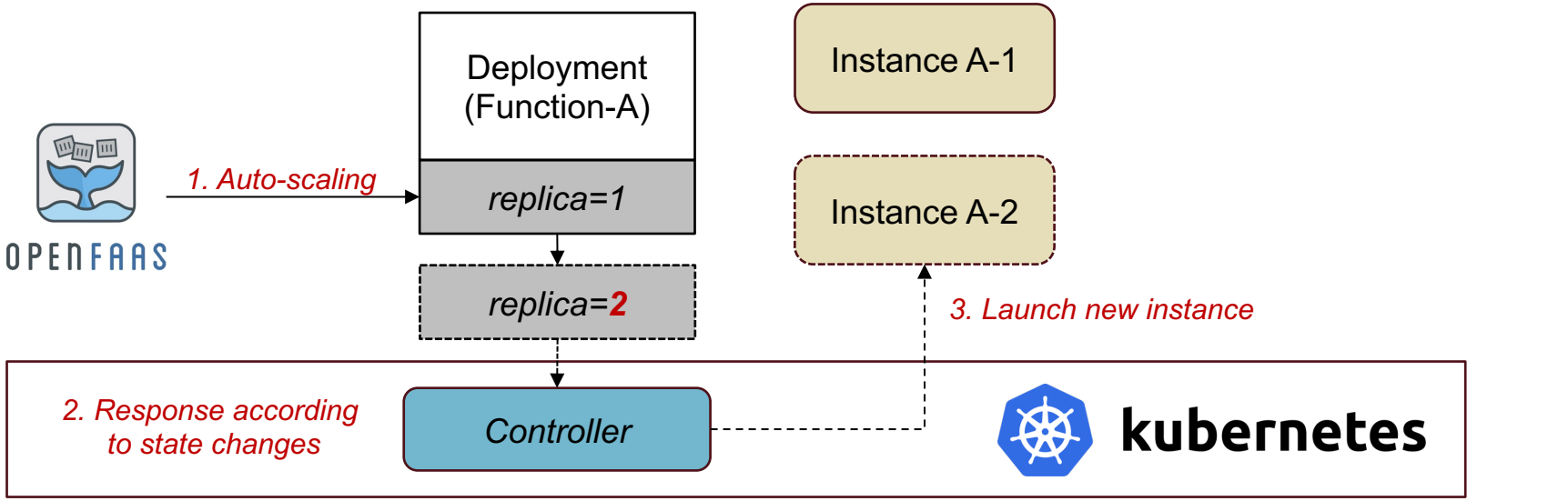
    ***End-to-end latency = cold start latency+ execution latency***

- **Gap: 异步启动下，时延的影响及优化方向是当前研究考虑较少的**
  - 异步启动下，存在多个因素共同影响时延表现：实例启动、请求队列设计方式和大小、request rate等等


**一个由于per-instance队列导致的负载不均衡的案例**

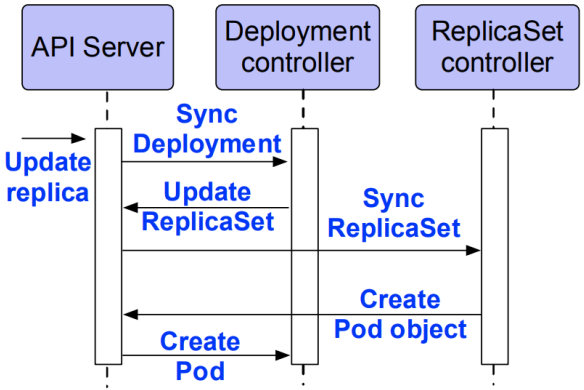
# 技术挑战2：声明式方法带来的开销（Declarative tax）

- 基于状态同步（state synchronization）声明式管理



# 技术挑战2：声明式方法带来的开销（Declarative tax）

- **声明式方法由于涉及大量的状态同步，会导致**
  - 时延开销：直接影响端到端时延表现
  - 时延不确定性：性能表现抖动较大
  - **当前研究与优化工作，较少考虑声明式方法开销！**
- **机会、建议，以及思考**
  - 当前系统（如基于K8s）仍存在大量的可优化空间 (APIServer, etcd, controllers)
  - 是否存在快速路径?
  - **思考：**k8s是当前云原生平台广泛使用的系统，但是在面向毫秒级的Serverless计算场景时，是否是最终方案?

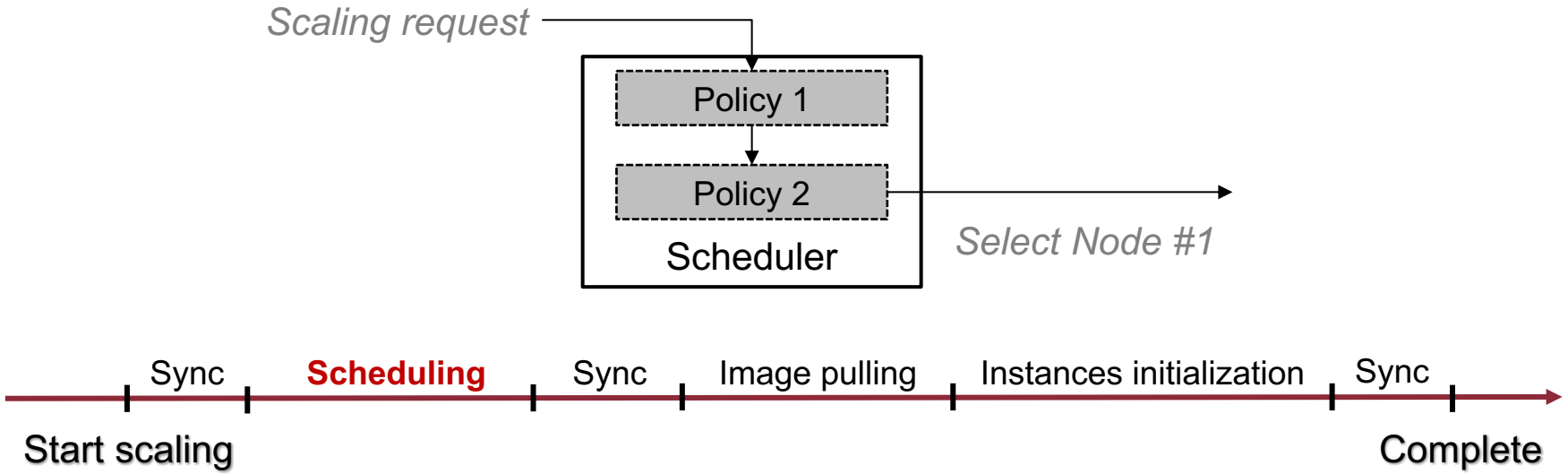



Total cost: **10-100** ms

**Non-negligible!**

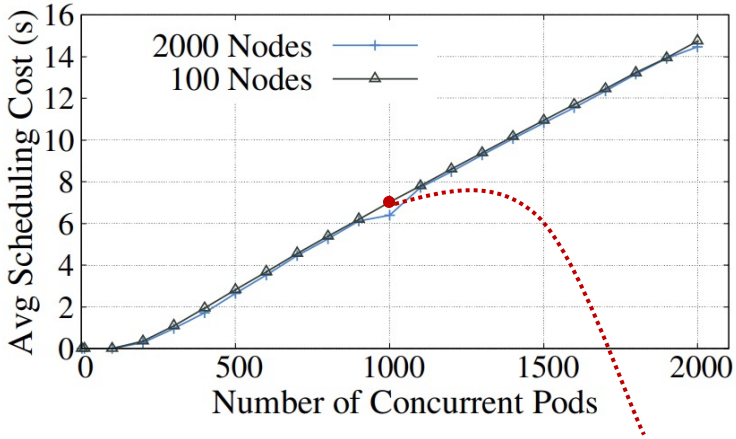
# 技术挑战3：实例调度时延（Scheduling Cost）

- **当前工作:**
  - 较少（几乎没有）对调度时延进行优化和考虑
- **真实场景:**
  - 随着规模增大，调度开销成为端到端性能表现的重要一环



# 技术挑战3：实例调度时延（Scheduling Cost）

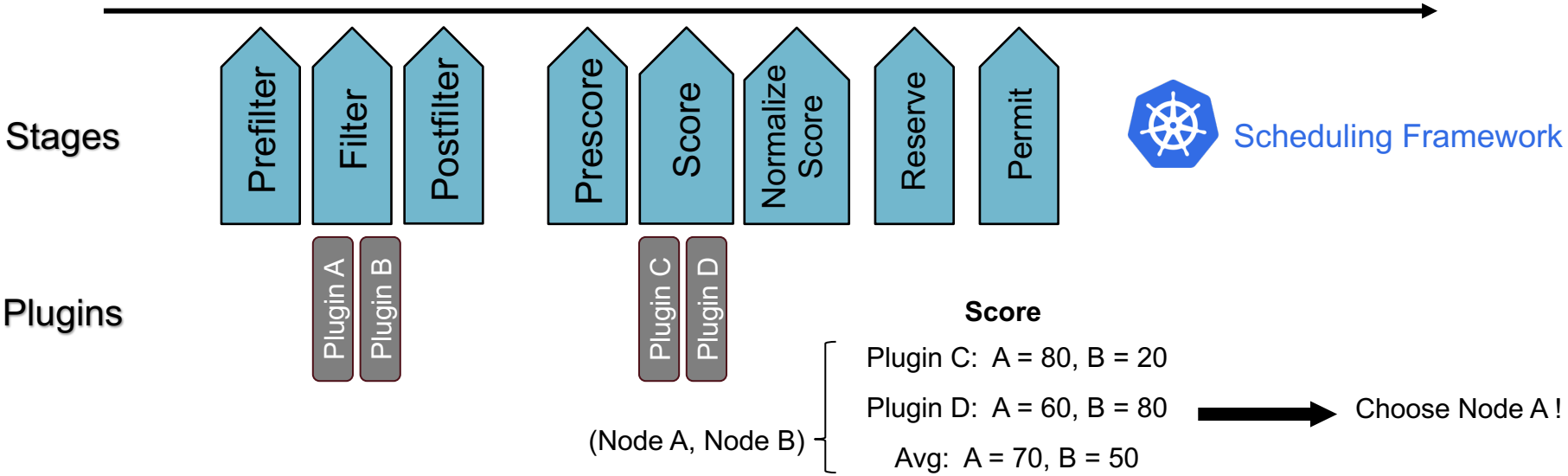
- **云场景下，调度开销能够有多严重?**
  - 时延开销、资源开销
  - 高并发实例下尤其严重（Serverless常态）
- **导致的原因**
  - 当前调度系统可扩展性较差（Unscalable）
    - E.g., *binding* process of Kubernetes
  - 调度策略本身不可扩展
    - E.g., 调度决策依赖之前Pods的状态、调度结果等
- **机会与建议**
  - 可扩展性，应该是serverless调度系统（机制与策略）的重要指标
  - 需要基于大规模的环境进行系统评估和分析（如Kwok等工具）




~7s for ~1k concurrent Pods

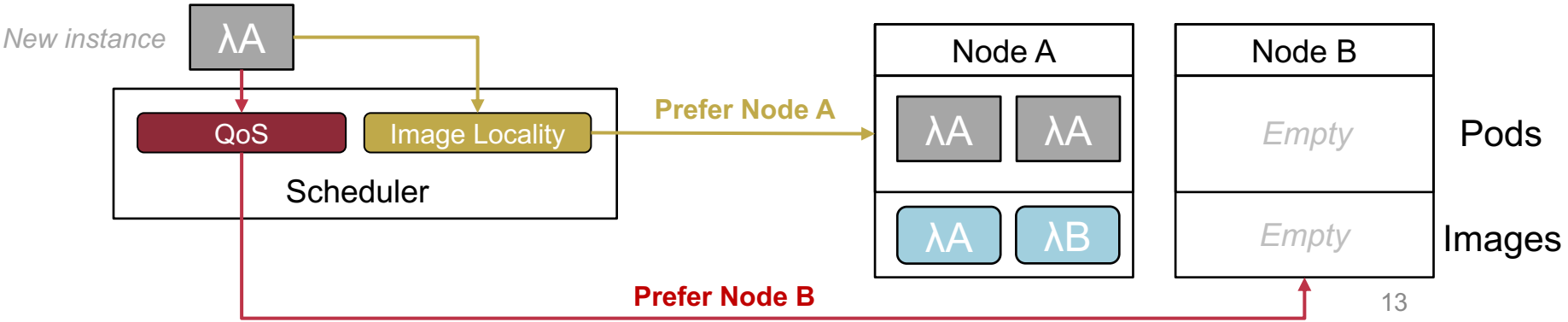
# 技术挑战4：多种调度策略的互相冲突导致次优决策

- **当前工作:**
  - 通常针对具体场景，只考虑单一的策略
- **真实场景和需求:**
  - 通常同时使用多个策略的组合



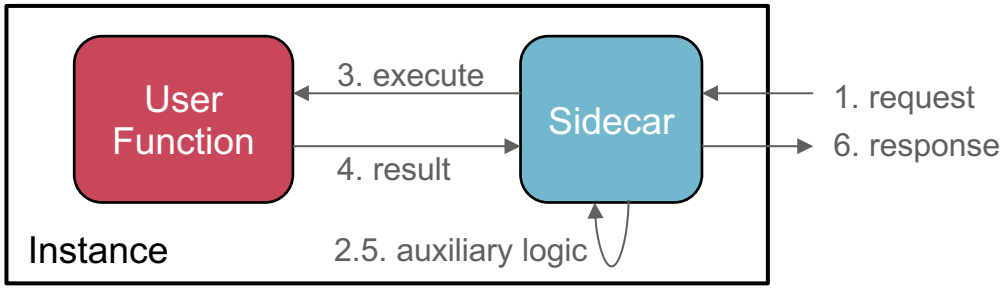
# 技术挑战4：多种调度策略的互相冲突导致次优决策

- **多种调度策略（插件）可能彼此之间会有冲突**
  - 不同任务和负载和具体的策略间的相关性也不同，最终导致次优的决策
- **挑战与机会：当前仍然缺乏有效平衡多种策略的方法**
  - 现有机制通过简单的加权等来平衡多个策略，无法适应复杂情况
  - 机会：结合应用特征，基于更灵活的方式（如ML）来协助调度策略的平衡



# 技术挑战5：旁车（Sidecar）的开销

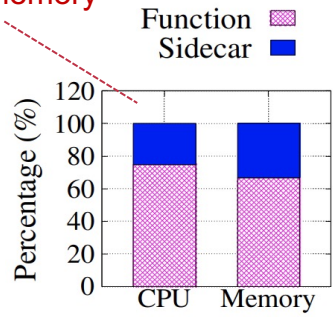
- **当前工作:**
  - 较少在Serverless当中考虑Sidecar
- **真实场景和需求:**
  - Sidecar在真实Serverless系统中被广泛使用
- **为什么旁车（Sidecar）会被用于Serverless?**
  - 提供额外的能力：网络、代理、日志等
  - 更好的模块化（更高的抽象）
  - 将旁车逻辑和函数实例逻辑解耦，能支持二进制镜像的函数
  - 能够实现细粒度的资源配比，旁车和函数实例容器分别配置



# 技术挑战5：旁车（Sidecar）的开销

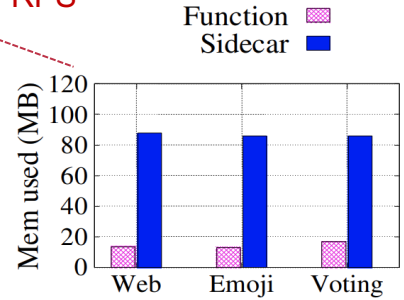
- **旁车仍然缺乏面向Serverless计算的优化，导致显著开销**
  - 资源开销： 占用CPU与memory，函数本身资源不大，导致旁车开销显著
  - 通信时延开销: 导致网络通信引入更多的上下文切换和网络协议栈（openEuler Kmesh等项目是较好的优化方向）
  - 启动时延：启动和初始化更多的实例，同步顺序等开销

25% CPU
33% memory




Allocated resources

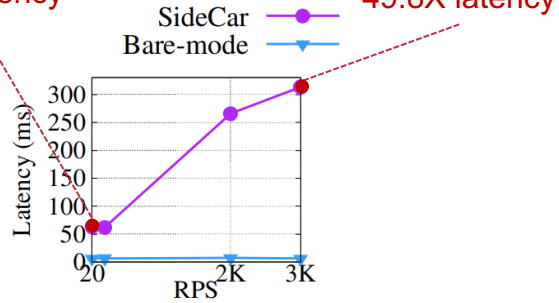
Use more resources
with low RPS




Used resources

9.5X latency

49.8X latency




Latency

# 总结与Takeaway

**Takeaway-1**: 从同步启动到**异步启动**，传统的优化方法已经显得不够，需要面向异步启动重新思考和设计

**Takeaway-2**: **声明式方法**简化管理负担的同时，也引入了开销与不确定性，如何兼顾较好的管理方法（如声明式）和较优的性能表现，是一个重要挑战

**Takeaway-3**: 面向函数粒度的Serverless场景，**调度器的可扩展能力**至关重要，当前系统仍然存在调度性能较差、并发调度能力不足的问题

**Takeaway-4**: 多调度策略并存将是一个常态，如何设计一个统筹方法，能够面向不同的负载和请求情况，**自适应平衡多调度策略**，是实现Serverless资源优化的重要方向

**Takeaway-5**: **旁车架构**为系统带来了大量的好处，但是其开销不可忽视，尤其在极小粒度的Serverless场景下，对其的性能和资源占用的优化对全平台十分重要

# Thanks!

联系我：dd_nirvana@sjtu.edu.cn

ACM SoCC'23 同名论文：
https://dl.acm.org/doi/10.1145/3620678.3624785