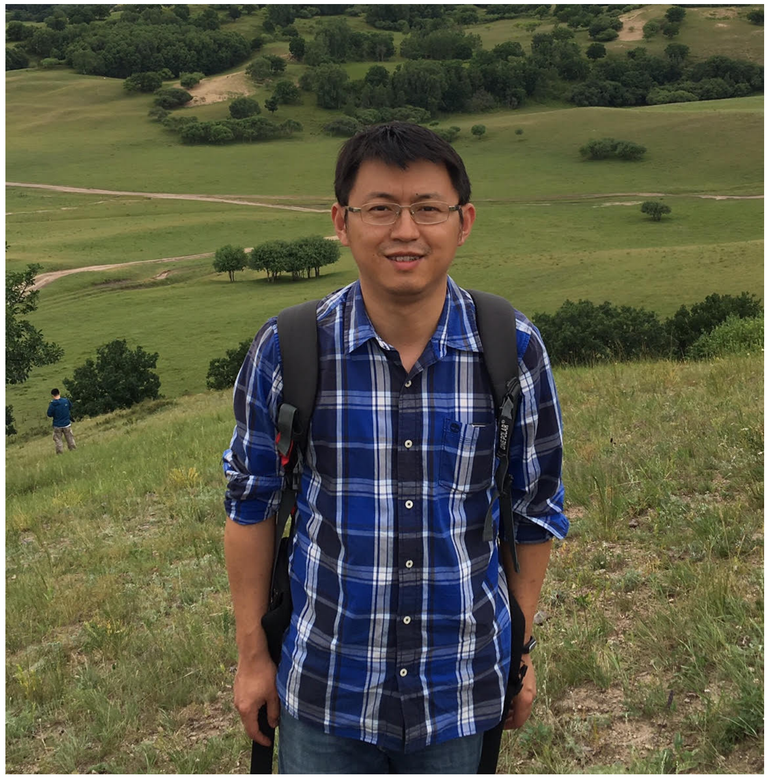
《Saga分布式事务解决方案与实践》

演讲者 / 姜宁

关于我

- 华为开源能力中心
- ServiceComb项目负责人
- Apache Member , IPMC, 多个Apache项目
- RedHat, IONA, Travelsky



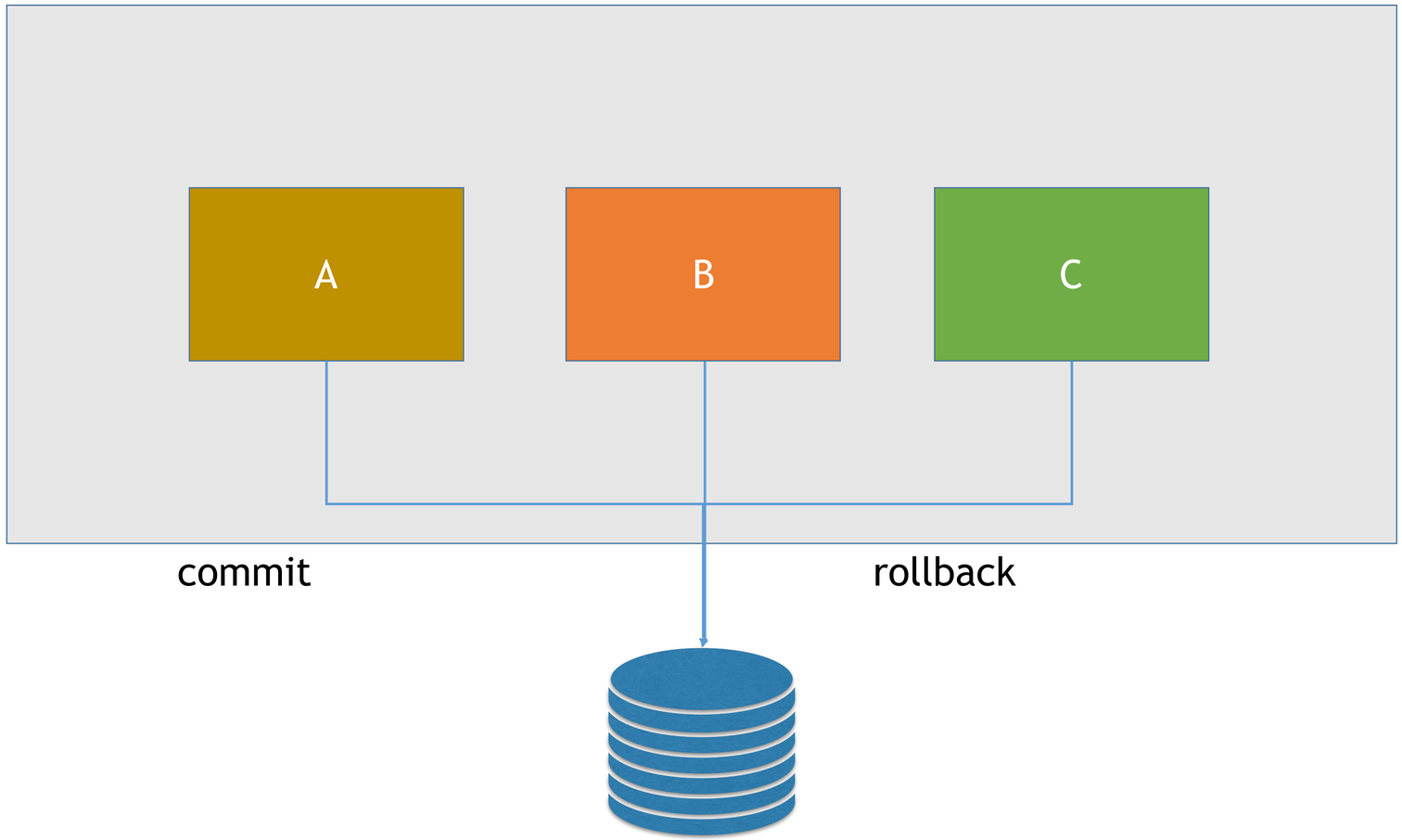
议题

- 微服务事务一致性问题?
- 业界Saga的解决方案
- ServiceComb Saga的演进
- 后续的开发计划

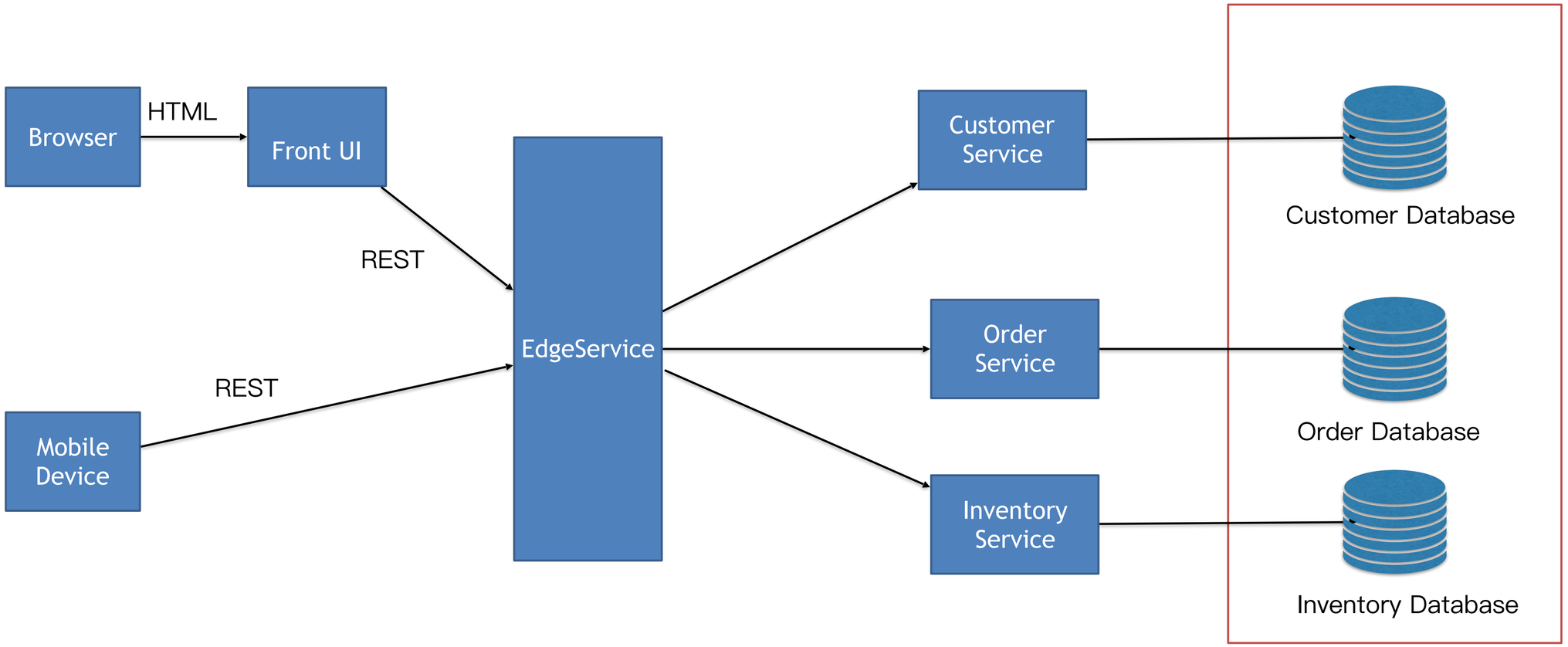
微服务架构

- 微服务架构将一个应用分成多个相互独立的服务。
- 好处是各个服务能够持续独立的开发和部署。
- 难题是服务的数据需要采用什么样的方式来进行存储?

多个微服务使用同一数据库



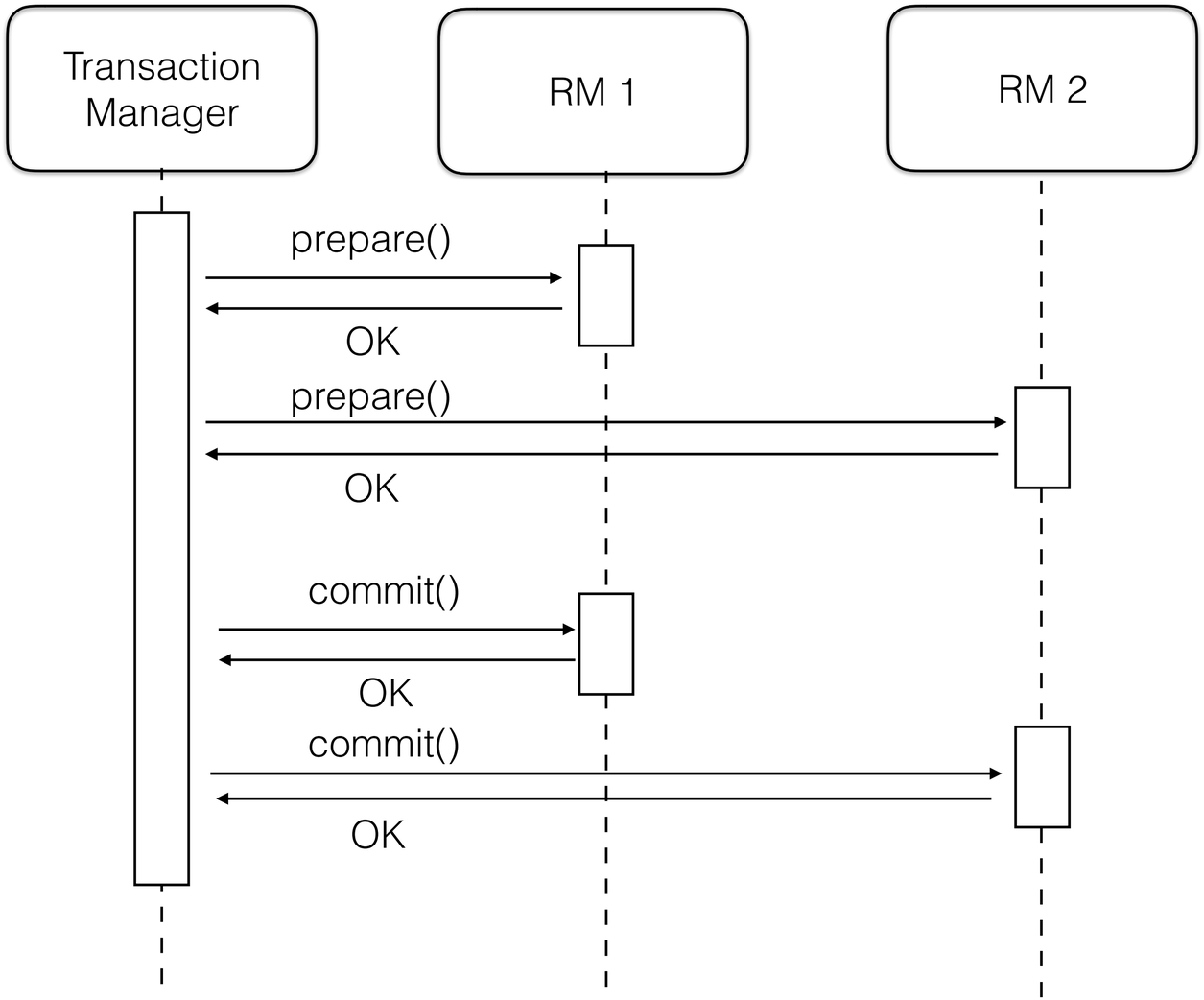
一个典型的微服务架构应用



http://microservices.io/patterns/data/database-per-service.html

http://blog.christianposta.com/microservices/the-hardest-part-about-microservices-data/

两阶段提交 2PC



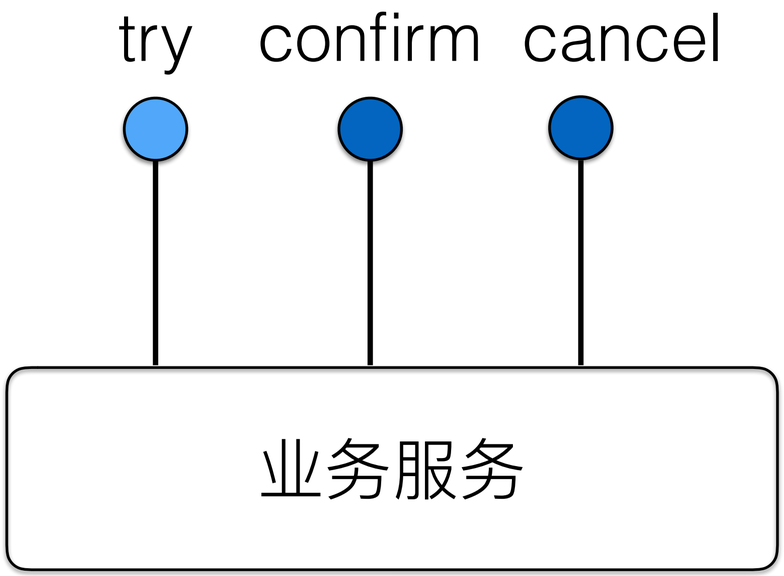
- 提供强一致保障
  - 准备阶段完成资源操作
  - 如果准备过程中出现问题，可以回滚
  - 提交阶段不允许出错
  - 资源层面提供保障业务侵入性低
  - 协议成本高，并且存在全局锁的问题

ACID 与 BASE

- ACID （刚性事务）
  - 原子性 （Atomicity）
  - 一致性 （Consistency）
  - 隔离性 （ Isolation）
  - 持久性 （Durability）
- BASE （柔性事务）
  - 基本可用（Basically Available）
  - 柔性状态 （Soft state）
  - 最终一致性 （Eventually Consistent）

https://queue.acm.org/detail.cfm?id=1394128

TCC



- Try: 尝试执行业务
  - 完成所有业务检查，预留必须的业务资源
- Confirm：确认执行业务
  - 真正执行业务，不做业务检查
- Cancel：取消执行业务
  - 释放Try阶段预留的业务资源

通过领域建模来解决


DOMAIN-DRIVEN DESIGN
TACKLING COMPLEXITY
IN THE HEART OF SOFTWARE
Pearson
领域驱动设计
软件核心复杂性应对之道


DDD Community
PEARSON
实现
领域驱动设计
IMPLEMENTING
DOMAIN-DRIVEN DESIGN
Vaughn Vernon 著

微服务事务一致性建议

- 内刚
  - 微服务内：聚合通过数据库
- 外柔
  - 微服务间：最终一致

Saga简介

- 1987年Hector & Kenneth 发表论文 Sagas
- Saga = Long Live Transaction (LLT)
- LLT = T1 + T2 + T3 + ... + Tn
- 每个本地事务Tx 有对应的补偿 Cx

SAGAS

Hector Garcia-Molina
Kenneth Salem

Department of Computer Science
Princeton University
Princeton, N J 08544

T1 T2 T3 ... Tn
C1 C2 C3 ... Cn

T1 T2 T3 ... Tn

正常情况

T1 T2 T3 C3 C2 C1

异常情况

https://www.cs.cornell.edu/andru/cs711/2002fa/reading/sagas.pdf

业界Saga的研究应用情况

Caitie McCaffrey

Distributed Sagas

https://github.com/aphyr/dist-sagas/blob/master/sagas.pdf

Chris Richardson

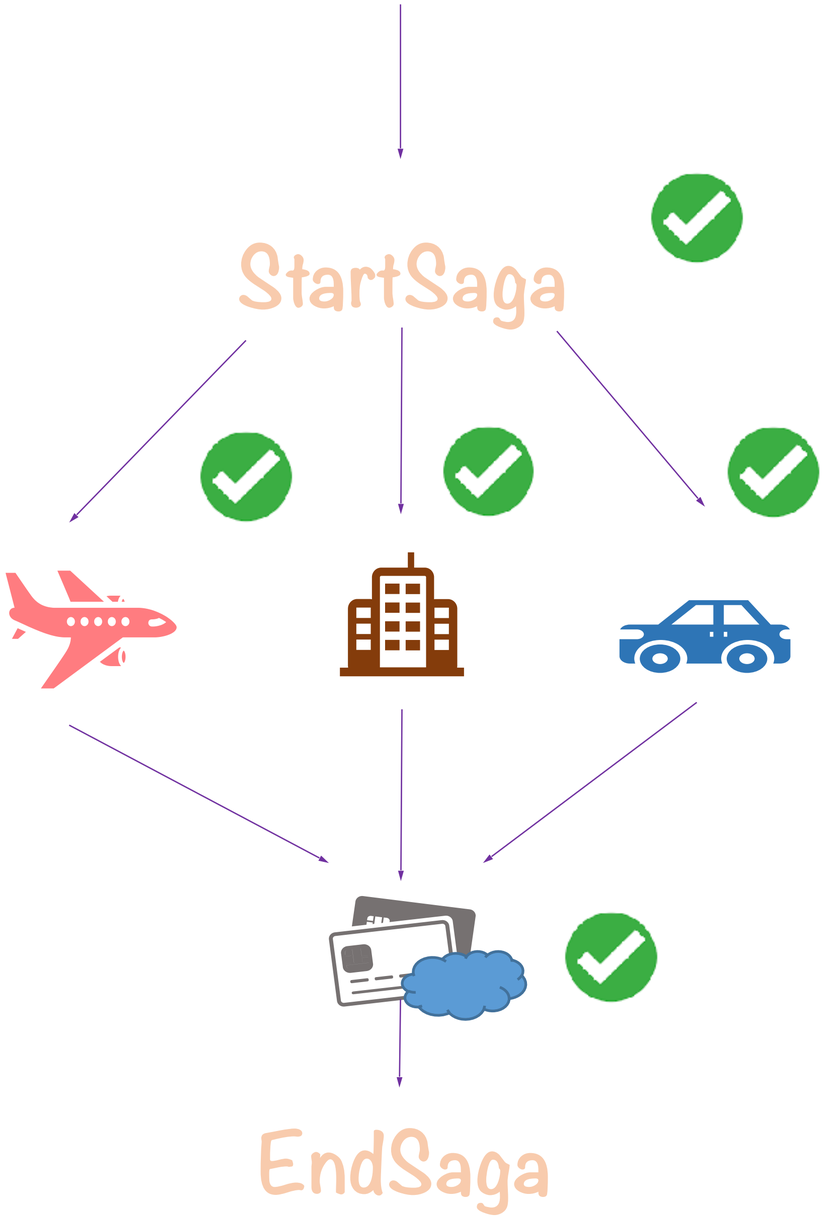
Microservice saga pattern

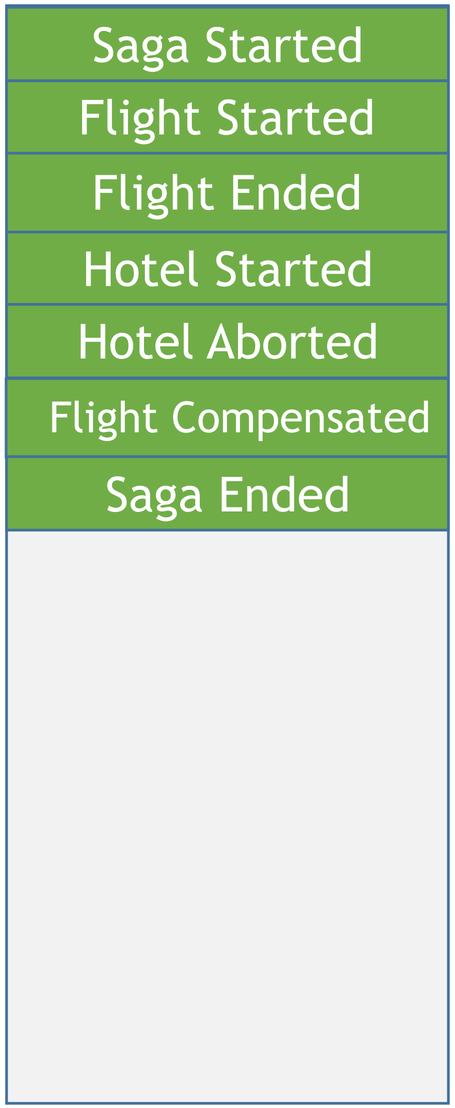
http://microservices.io/patterns/data/saga.html

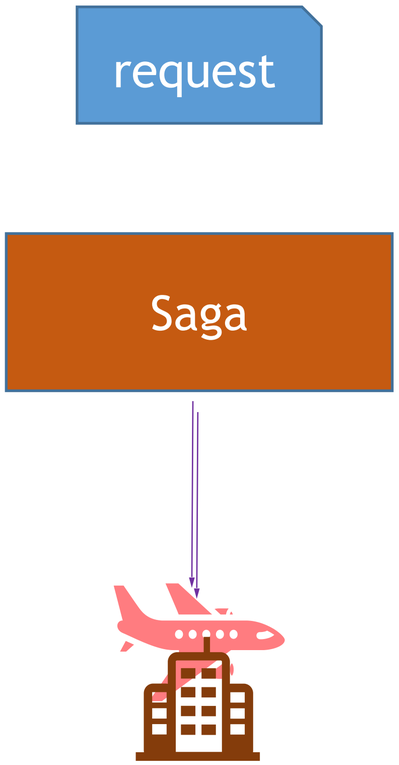
Saga Started
Flight Started
Flight Ended
Hotel Started
Hotel Ended
Car Started
Car Ended
Payment Started
Payment Ended
Saga Ended

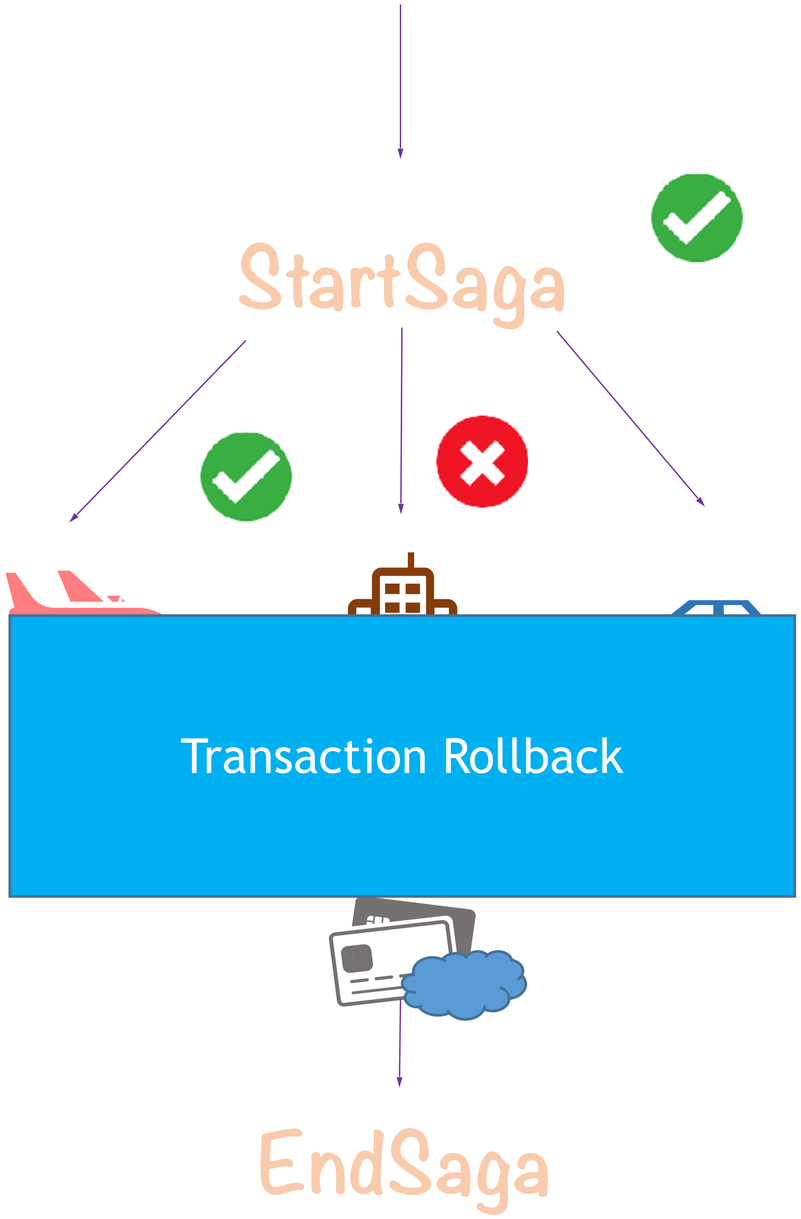
request

Saga



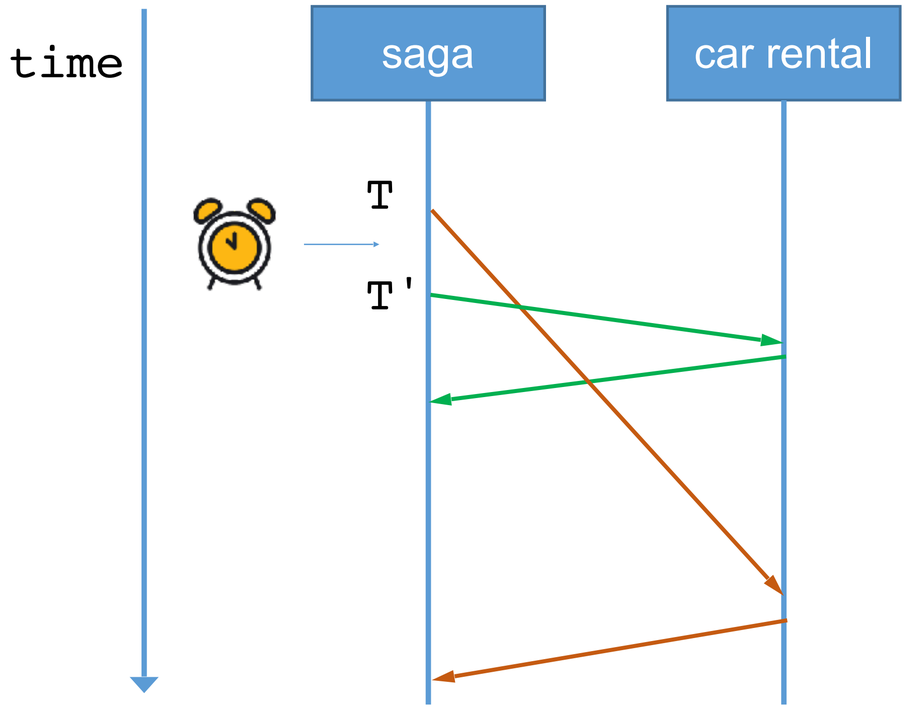

Saga Started
Flight Started
Flight Ended
Hotel Started
Hotel Aborted
Flight Compensated
Saga Ended


request
Saga


StartSaga
Transaction Rollback
EndSaga

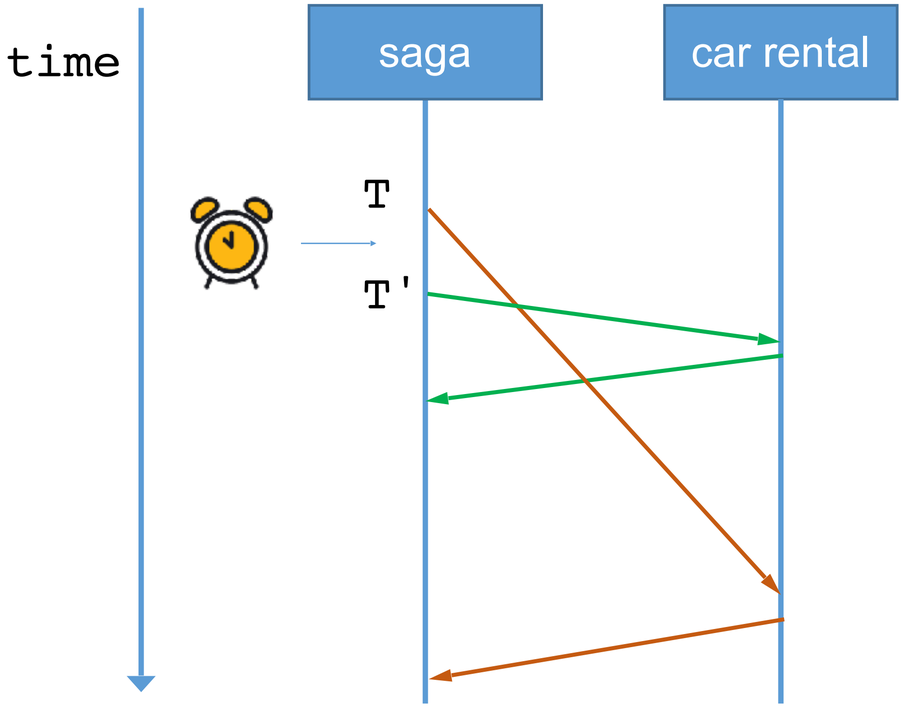
对服务的要求

- 幂等 T = T T … T

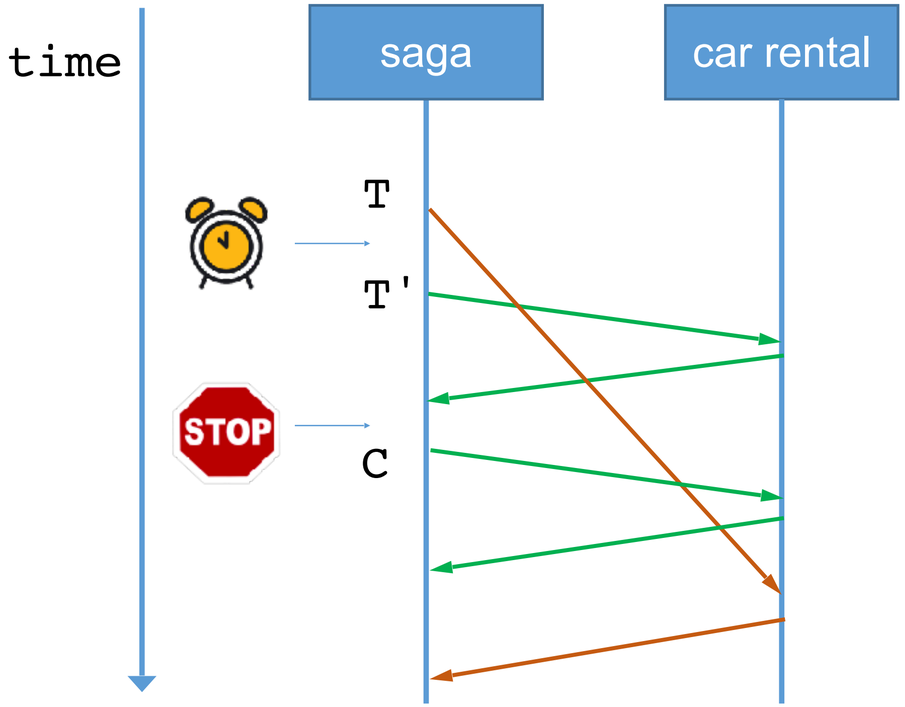


对服务的要求

- 幂等 T = T T … T



- 可交换补偿 T C = T C T



保留所有事务数据！

ACID 与 Saga

- ACID
  - 原子性 （Atomicity）
  - 一致性 （Consistency）
  - 隔离性 （ Isolation）
  - 持久性 （Durability）

- Saga只提供ACD保证
  - 原子性（通过Saga协调器实现）
  - 一致性 （本地事务 + Saga log）
  - 隔离性 （Saga不保证）
  - 持久性 （Saga log）

缺乏隔离性带来的问题

- 两个Saga事务同时操作一个资源会出现数据语义不一致的的情况。
- 两个Saga事务同时操作一个订单，彼此操作会覆盖对方（更新丢失）
- 两个Saga事务同时访问扣款账号，无法看到退款 （脏读取问题）
- 在一个Saga事务内，数据被其他事务修改前后的读取值不一致（模糊读取问题）

http://microservices.io/microservices/general/2018/03/22/microxchg-sagas.html

如何应对隔离性问题

- 隔离的本质是控制并发，防止并发事务操作相同资源而引起结果错乱
  - 在应用层面加入逻辑锁的逻辑。
  - Session层面隔离来保证串行化操作。
  - 业务层面采用预先冻结资金的方式隔离此部分资金。
  - 业务操作过程中通过及时读取当前状态的方式获取更新。

Service Center

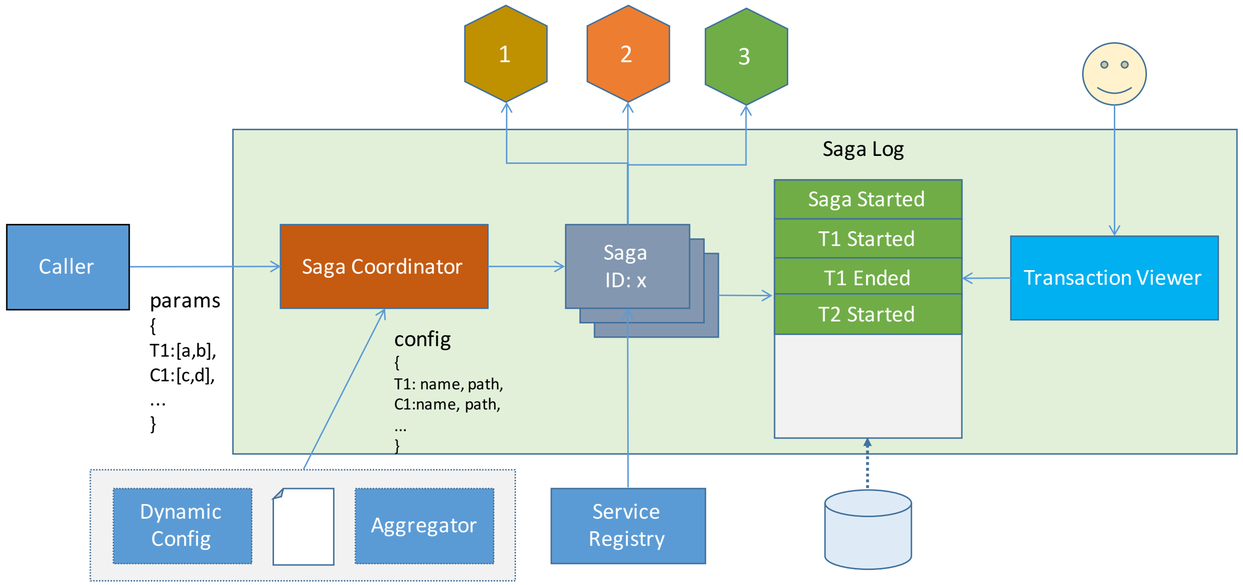
Java Chassis

Saga

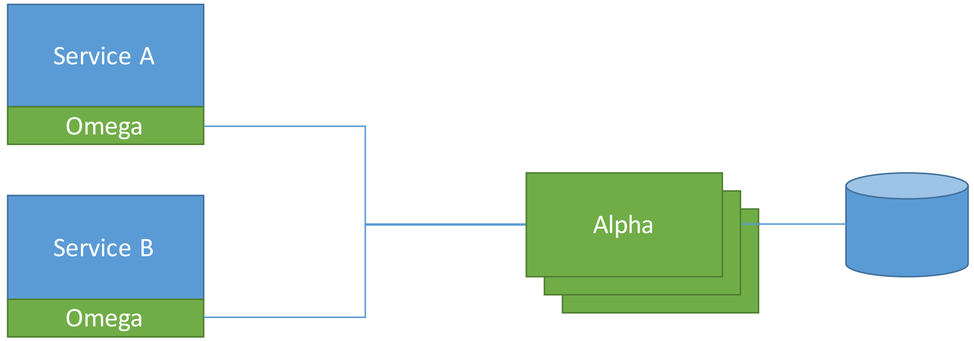
让云原生应用开发更简单

代码： https://github.com/apache?q=incubator-servicecomb
网站：http://servicecomb.incubator.apache.org/
华为云： https://www.huaweicloud.com/product/cse.html

ServiceComb Saga演进

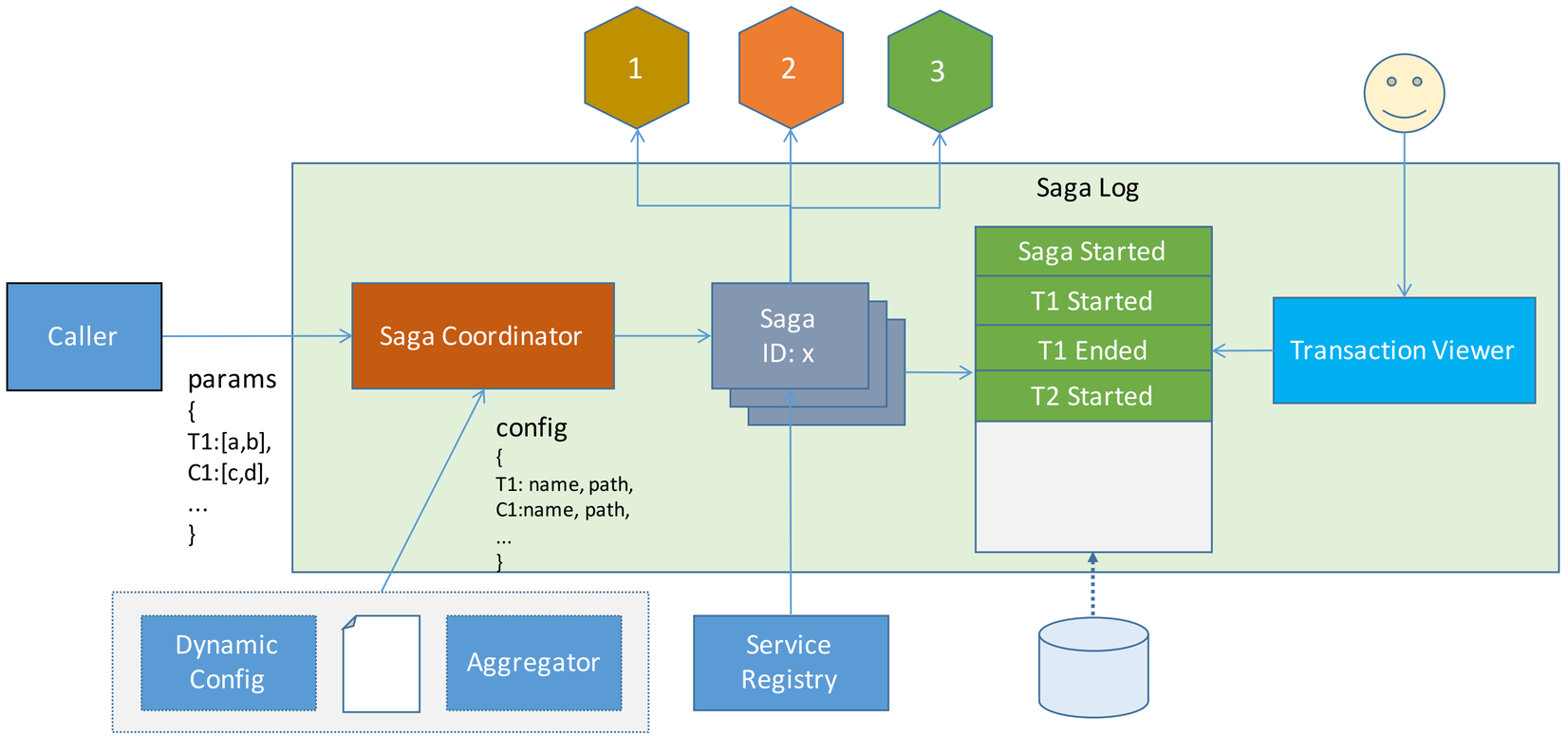


集中式的Saga协调器

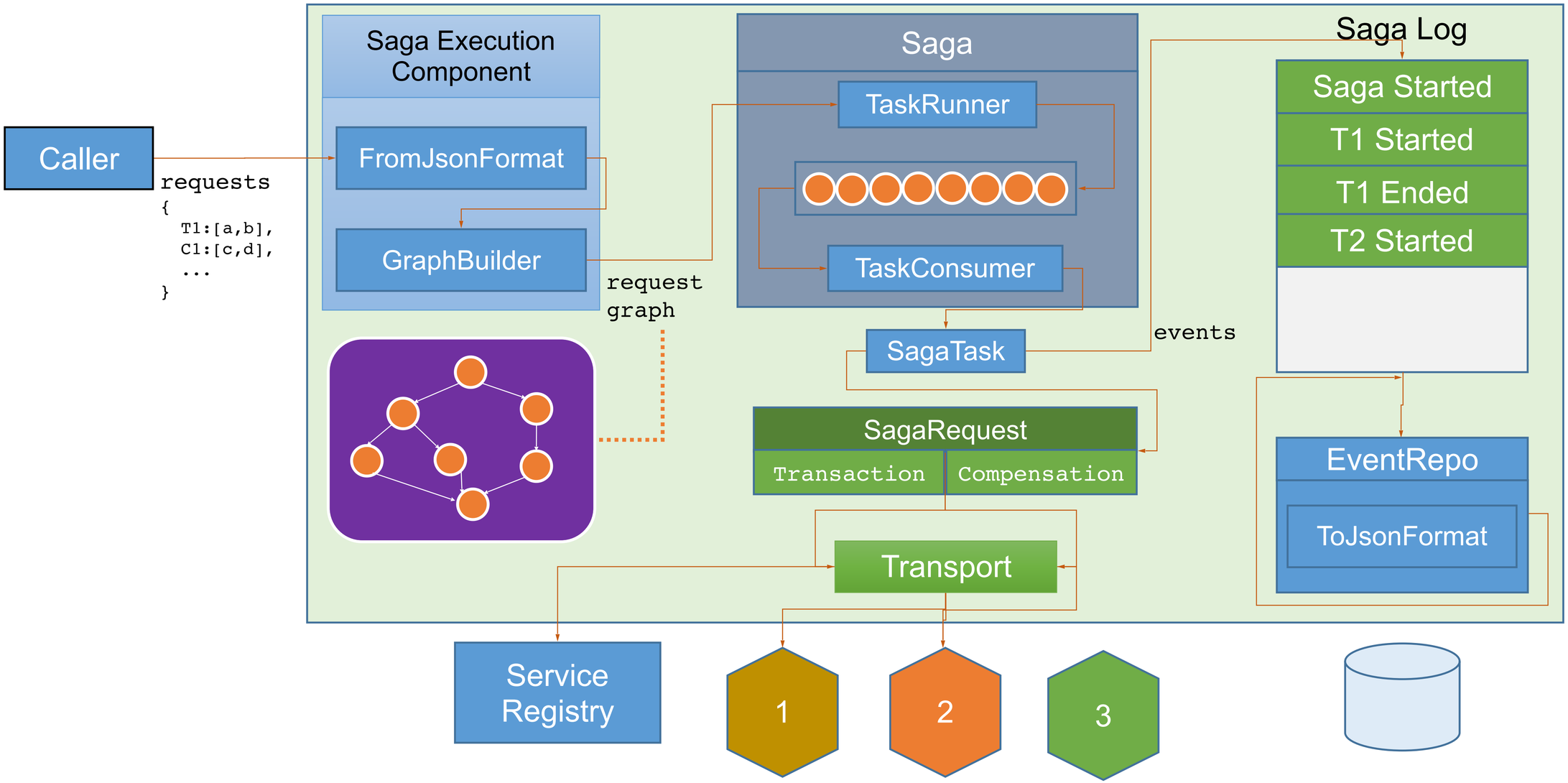


分布式Saga协调器

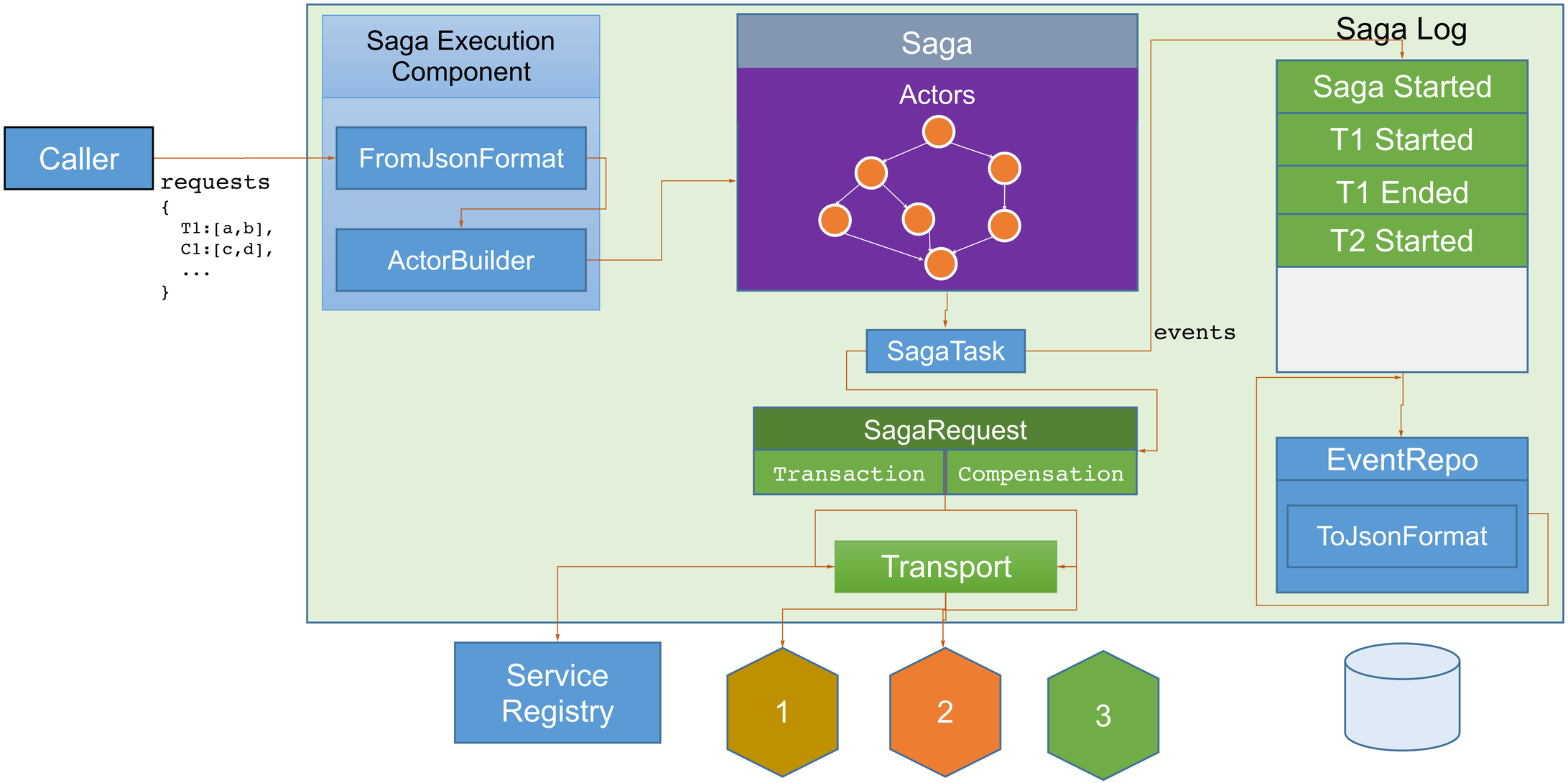
集中式Saga协调器


1
2
3
Saga Log
Caller
Saga Coordinator
Saga
ID: x
Saga Started
T1 Started
T1 Ended
T2 Started
Transaction Viewer
params
{
T1:[a,b],
C1:[c,d],
...
}
config
{
T1: name, path,
C1:name, path,
...
}
Dynamic
Config
Aggregator
Service
Registry

集中式的Saga – 基于图形



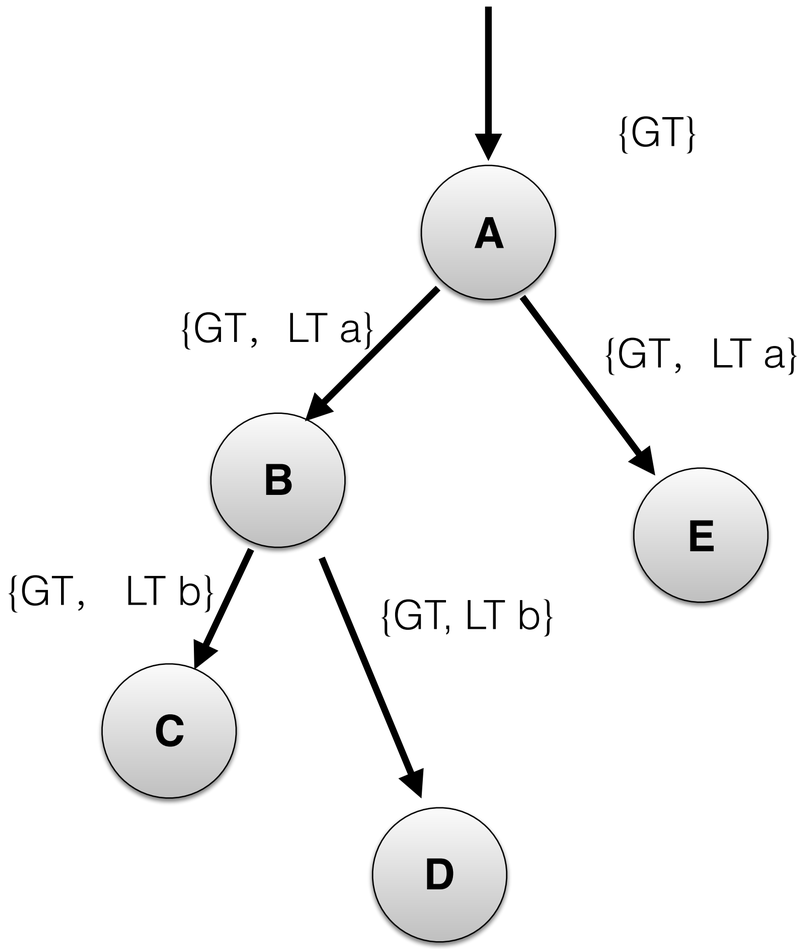
集中式Saga – 基于Actor模型

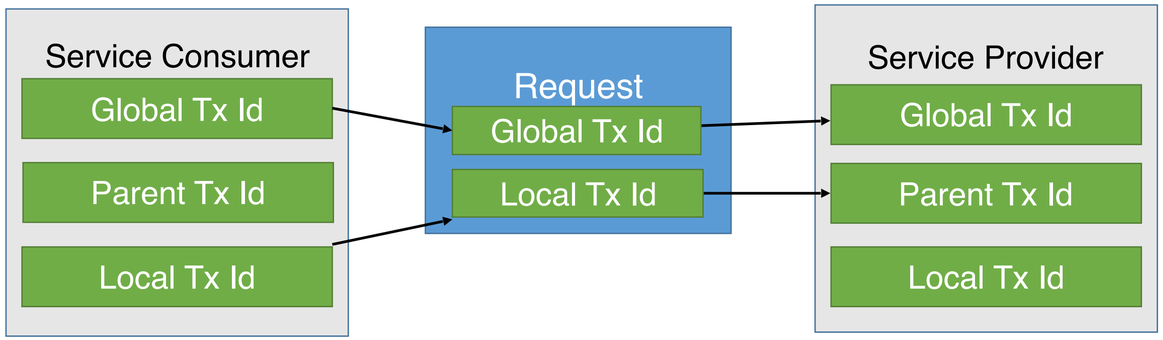


集中式Saga实现过程中问题

- 好处
  - 集中式的服务调用易于监控和协调
- 坏处
  - 通过JSon描述Saga事务执行灵活性不高
  - 业务描述与代码相分离，需要依赖UI工具的帮助
- 如何解决自动获取Saga事务定义的问题?

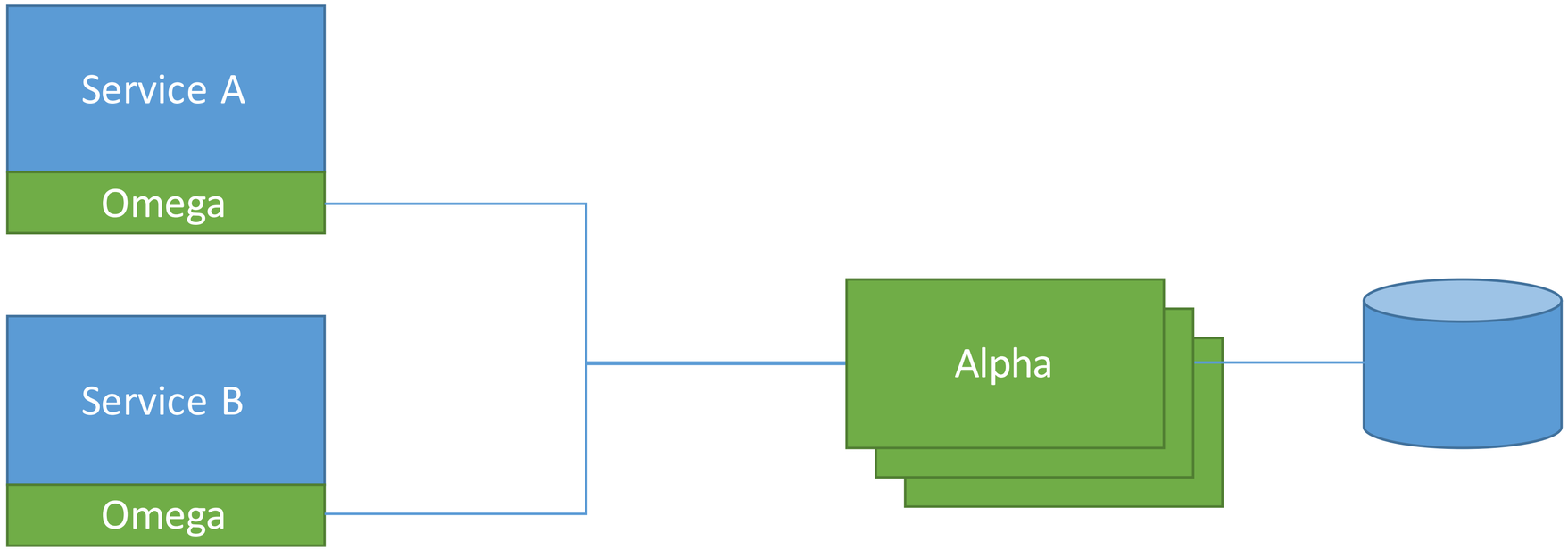
自动构造事务调用信息


{GT}
A
{GT， LT a}
{GT， LT a}
B
E
{GT， LT b}
{GT, LT b}
C
D

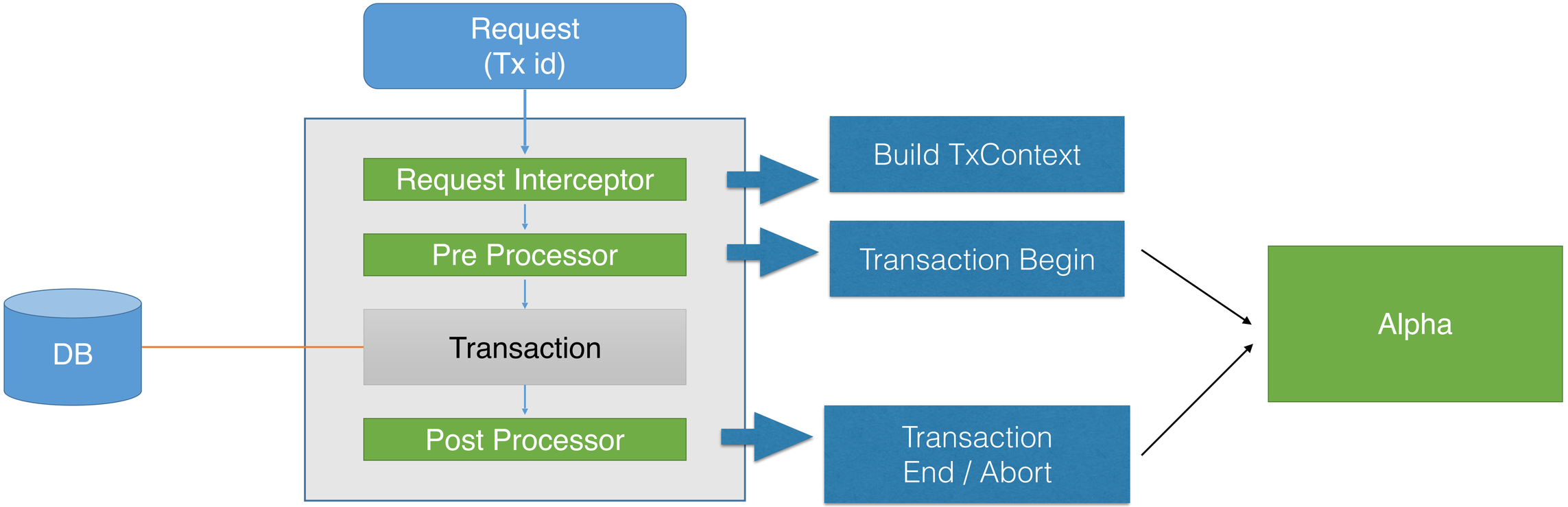

Service Consumer
Global Tx Id
Parent Tx Id
Local Tx Id
Request
Global Tx Id
Local Tx Id
Service Provider
Global Tx Id
Parent Tx Id
Local Tx Id

Saga的狼群架构

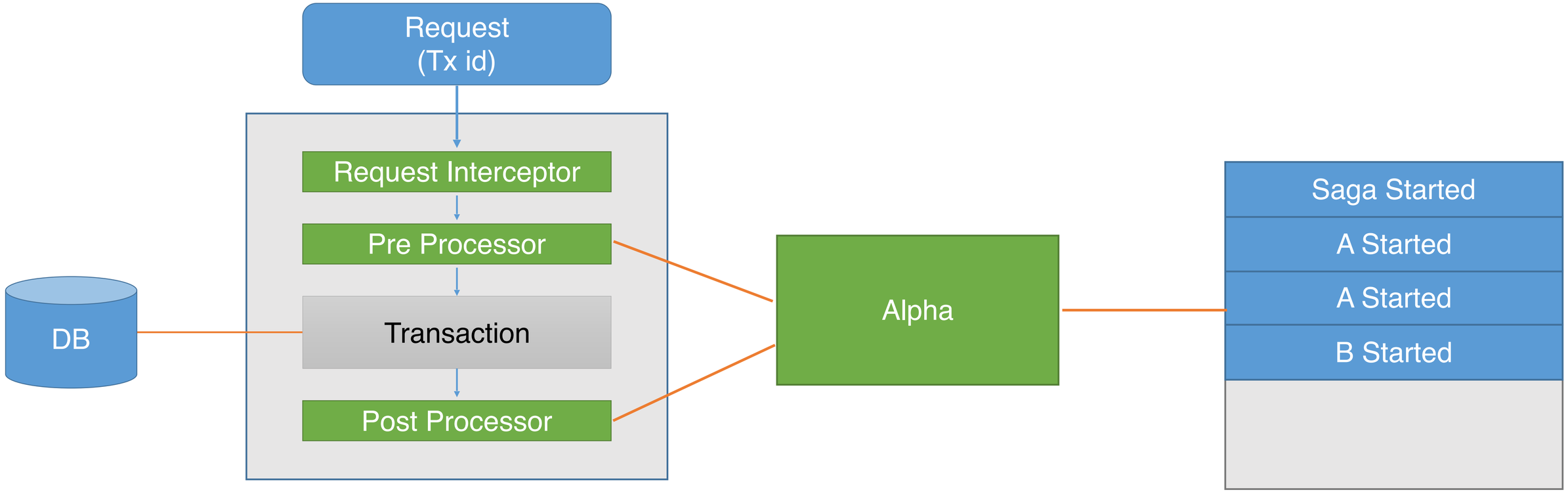
- Alpha是狼群首领，负责协调事务执行情况
- Omega是狼群成员，负责收集事务，向狼群首领上报情况，并执行相关指令



Omega 内部实现


Request
(Tx id)
Request Interceptor
Pre Processor
Transaction
Post Processor
DB
Build TxContext
Transaction Begin
Transaction
End / Abort
Alpha

Omega 与Alpha之间交互



未来的开发计划

- Alpha高可用多租户架构
- 基于消息队列的服务
- 提供TCC的协调控制服务
- 通过Omege提供幂等操作功能
- 可视化的事务拓扑，定位异常最多服务
- Omega进一步解决多线程间共享调用链问题

http://servicecomb.incubator.apache.org/developers/contributing

小结

- 微服务事务一致性问题?
- 业界Saga的解决方案
- ServiceComb Saga的演进
- 后续的开发计划

让云原生应用开发更简单

代码： https://github.com/apache?q=incubator-servicecomb
网站：http://servicecomb.incubator.apache.org/
华为云： https://www.huaweicloud.com/product/cse.html