DevOops

Nikhil Barthwal
Product Manager (Serverless),
Google Cloud Platform

nikhilbarthwal@yahoo.com

www.nikhilbarthwal.com

Implementing Microservices Architecture as Cloud Run Application

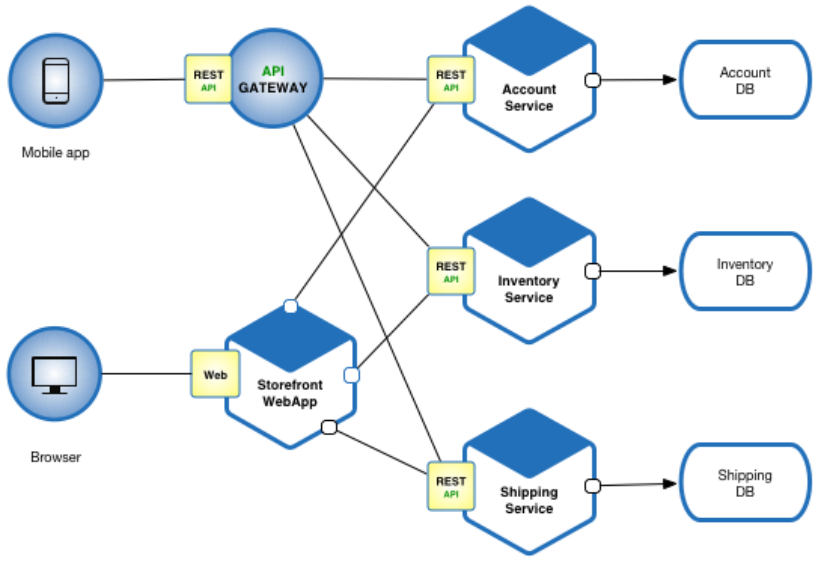
Agenda

- Problems with Microservices
- Why use Serverless to implement Microservices?
- Introducing Cloud Run
- Patterns & Practices for Implementation
- Closing notes …

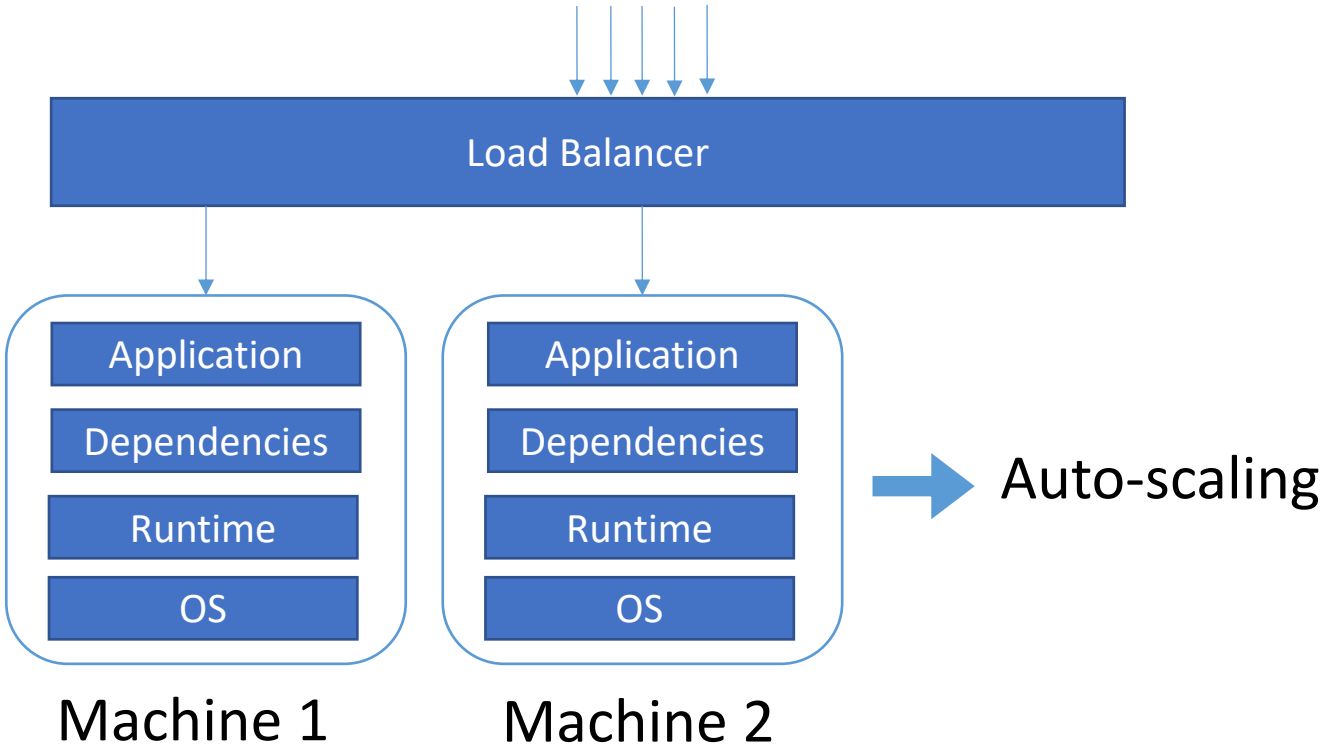
Microservices Architecture

Structures an application as a collection of loosely coupled fine-grained services that communicate with lightweight protocols

- Independent Releasability
- Resilience
- Ease of Migration
- Faster testing & deployment

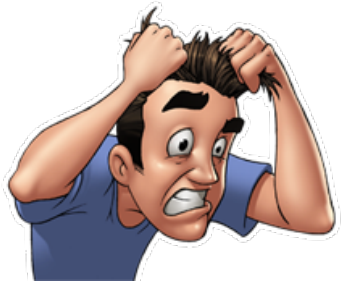


Microservices: The Problem!



- Load Balancing
- Scaling up & down
- Service discovery

Management Overhead!



Serverless Computing

Code execution model where server-side logic is run in stateless, event-triggered, ephemeral compute containers that are fully managed by a third-party.

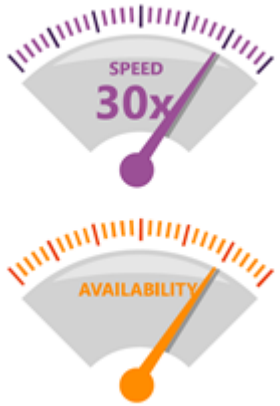
AWS Lambda

Azure Functions

Google Cloud Run

Characteristics of Serverless Applications

Infrastructure Abstraction



Auto-Scaling

Pay as you go

Less Management Overhead!

Comparison: Microservices & Serverless

Microservices: Assembly of fine-grained services to provide functionality

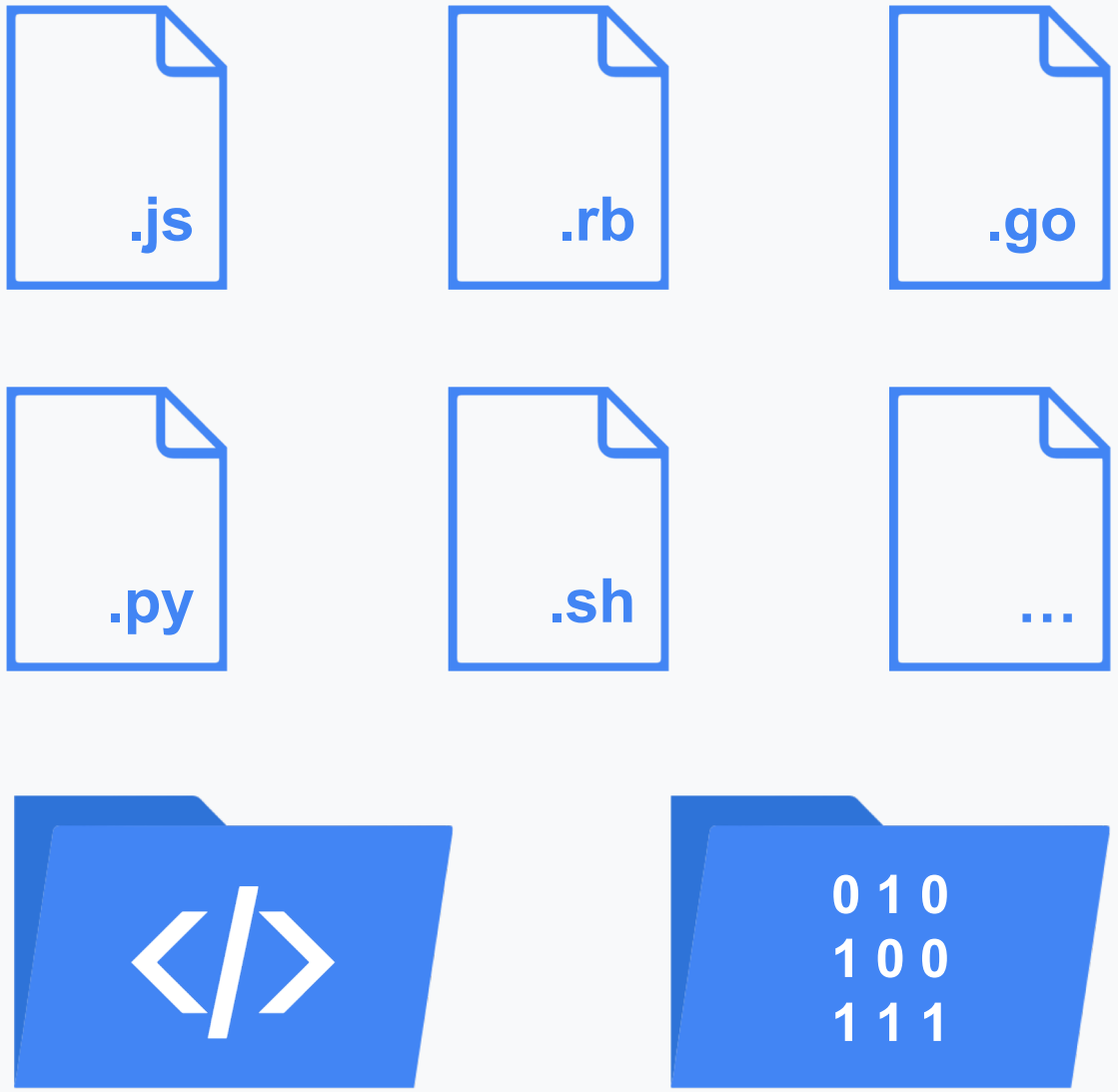
Serverless: Logic distributed in stateless, event-triggered computer container

Application composed of loosely coupled components

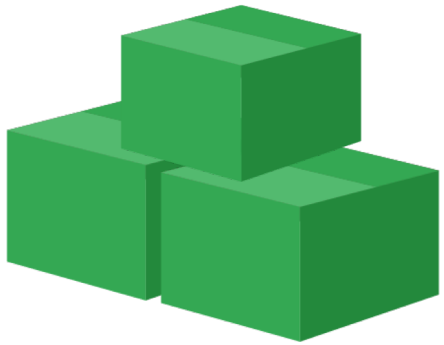
Microservices can be implemented as Serverless Application without management overhead

Containers

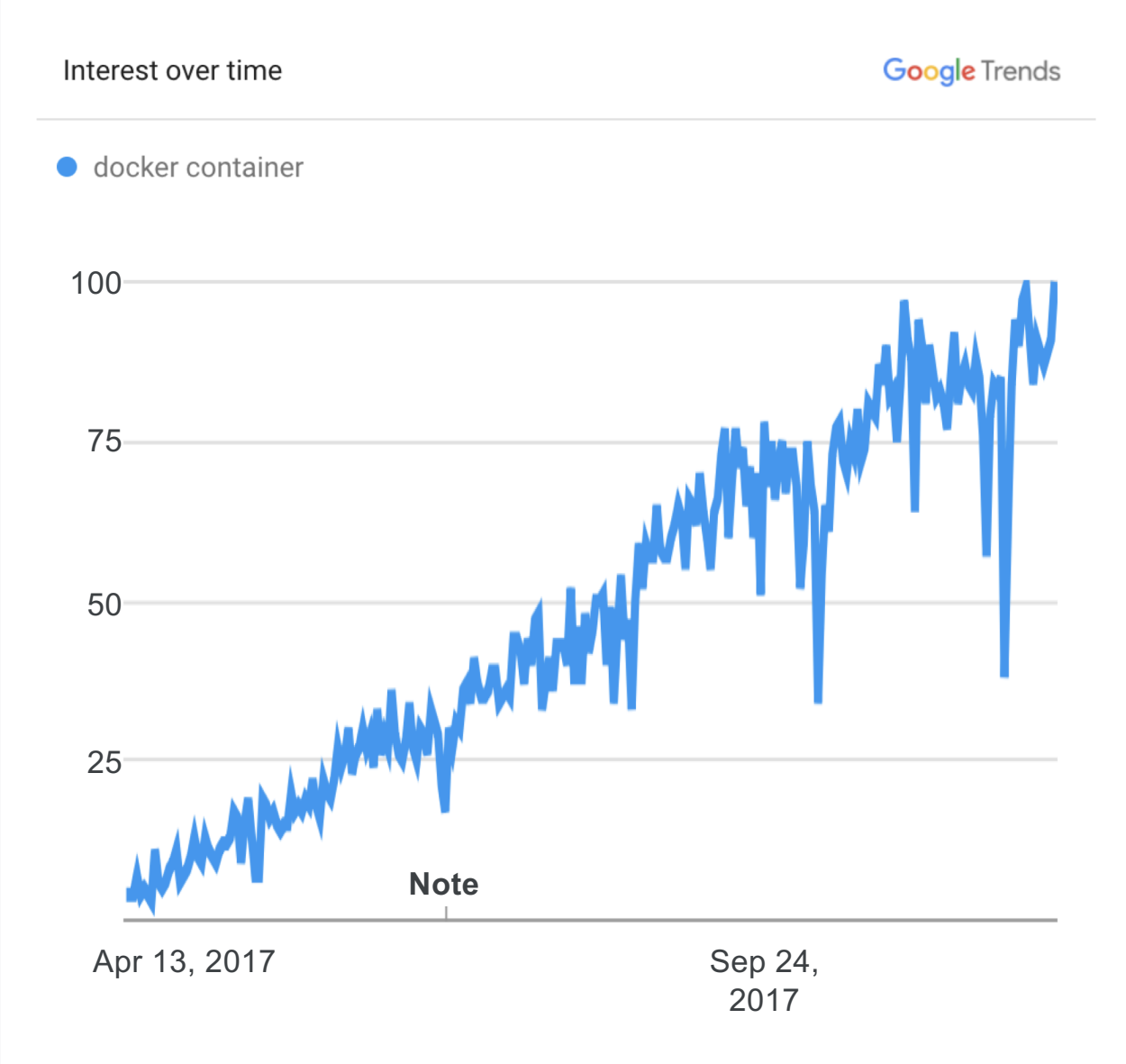
- Any Language
- Any Library
- Any Binary
- Ecosystem of base images



Containers:
An Industry standard



Popular way to package Microservices



Introducing

Cloud Run

Bringing serverless to containers

https://cloud.google.com/run

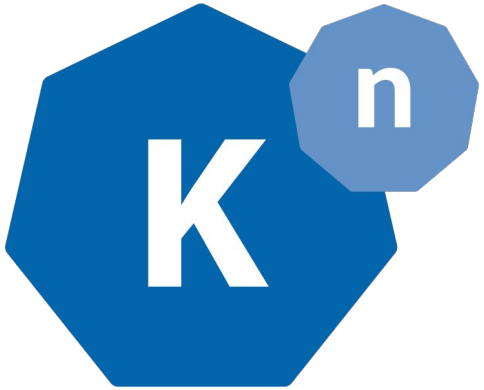
Serverless on Google Cloud

Cloud Run

Fully managed, deploy your workloads and don't see the cluster.

Cloud Run on GKE

Deploy into your GKE cluster, run serverless side-by-side with your existing workloads.

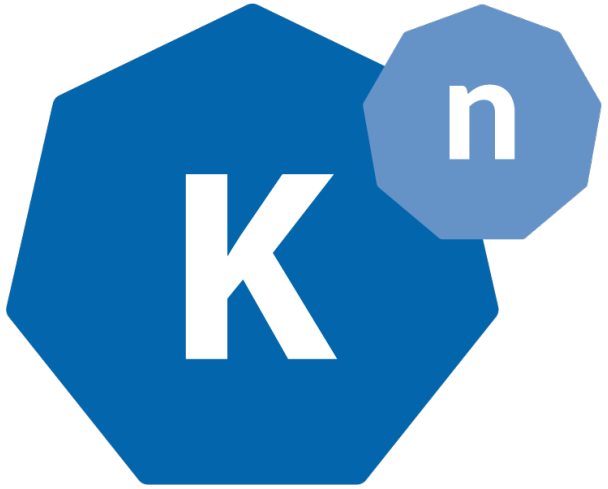


Knative Everywhere

Use the same APIs and tooling anywhere you run Kubernetes with Knative.

Portability of tooling, and workloads - you can even run serverless on-prem

Knative project



- Set of components (serving, eventing, ~~build~~)
- Ingredients for Serverless
- Solves for modern development patterns
- Implements learnings from Google, partners

https://knative.dev

Pivotal

SAP

redhat

IBM

Cloud Run Serverless Model

Operational Model

No Infra Management

Managed Security

Pay only for usage

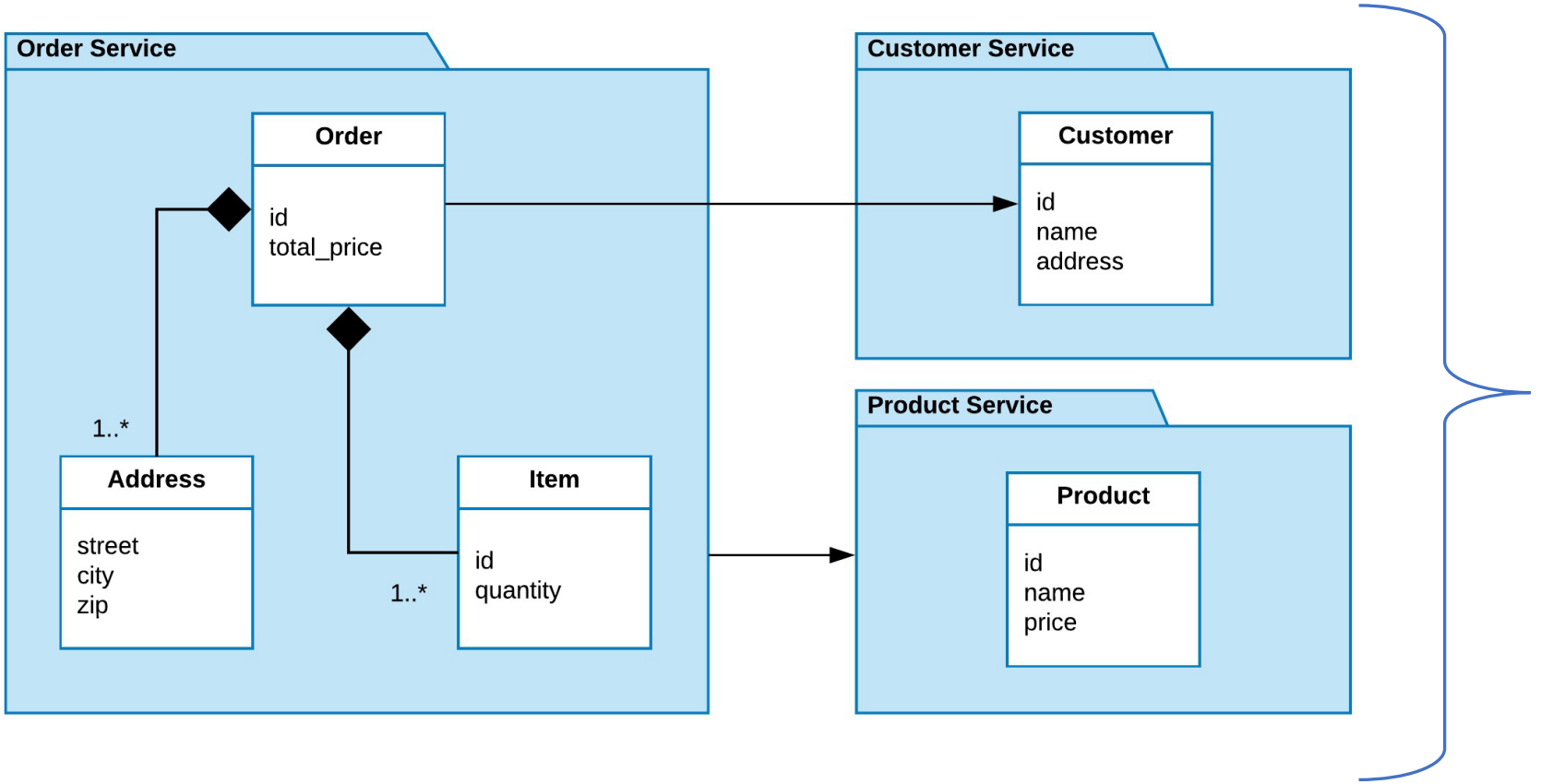
Programming Model

Service-based

Event-driven

Open

Data Partition Strategies: Use DDD

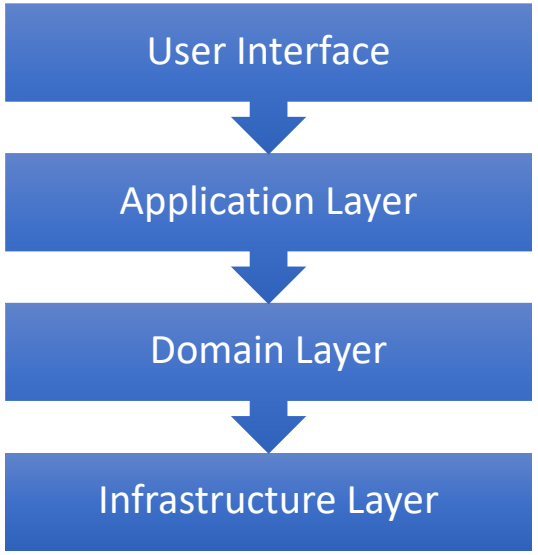


Aggregates provide boundaries for partition

Domain Driven Design (DDD)

An approach to software development for complex needs by connecting the implementation to an evolving model.

- Entities
- Value Objects
- Bounded Context
- Aggregates



DDD: Aggregates

- Cluster of domain objects that can be treated as a single unit
- One of its component is root
- All outside reference would only go to the aggregate root

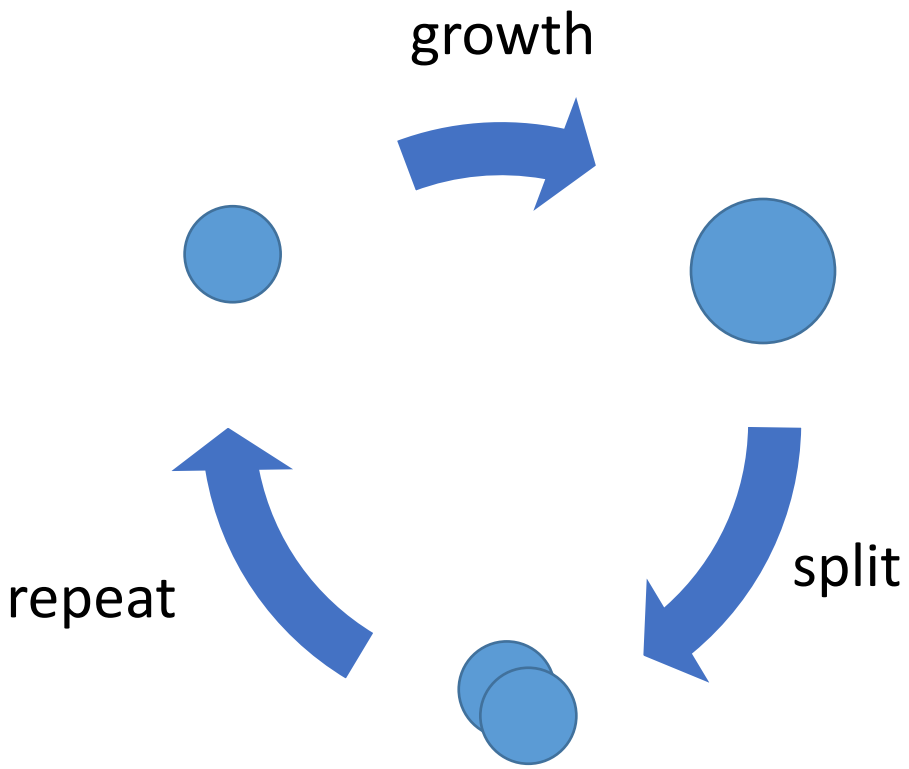
Domain Model

=

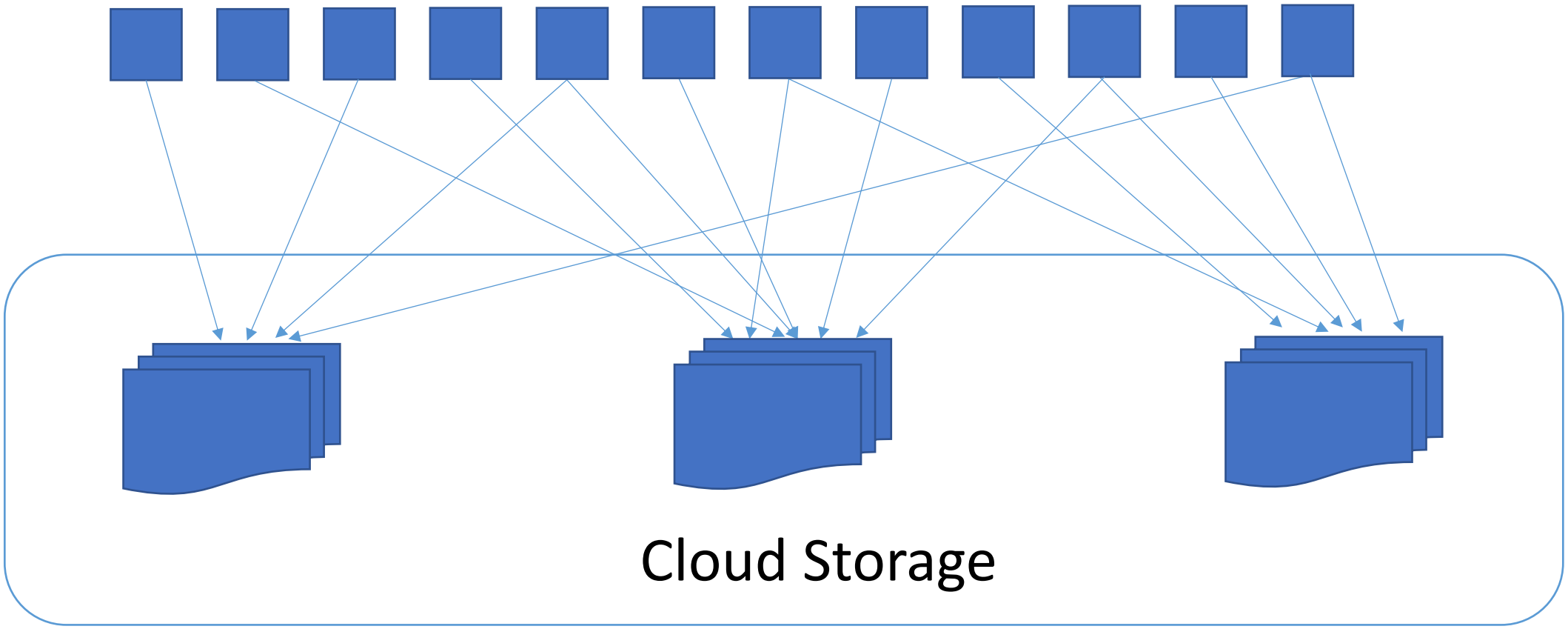
Collection of Aggregates

Pattern: Fine Grained Functionality

- Services have resource limits
- Distribute Functionality as small as possible
- Continuous refactoring needed



Anti-Pattern: Common Data Ownership



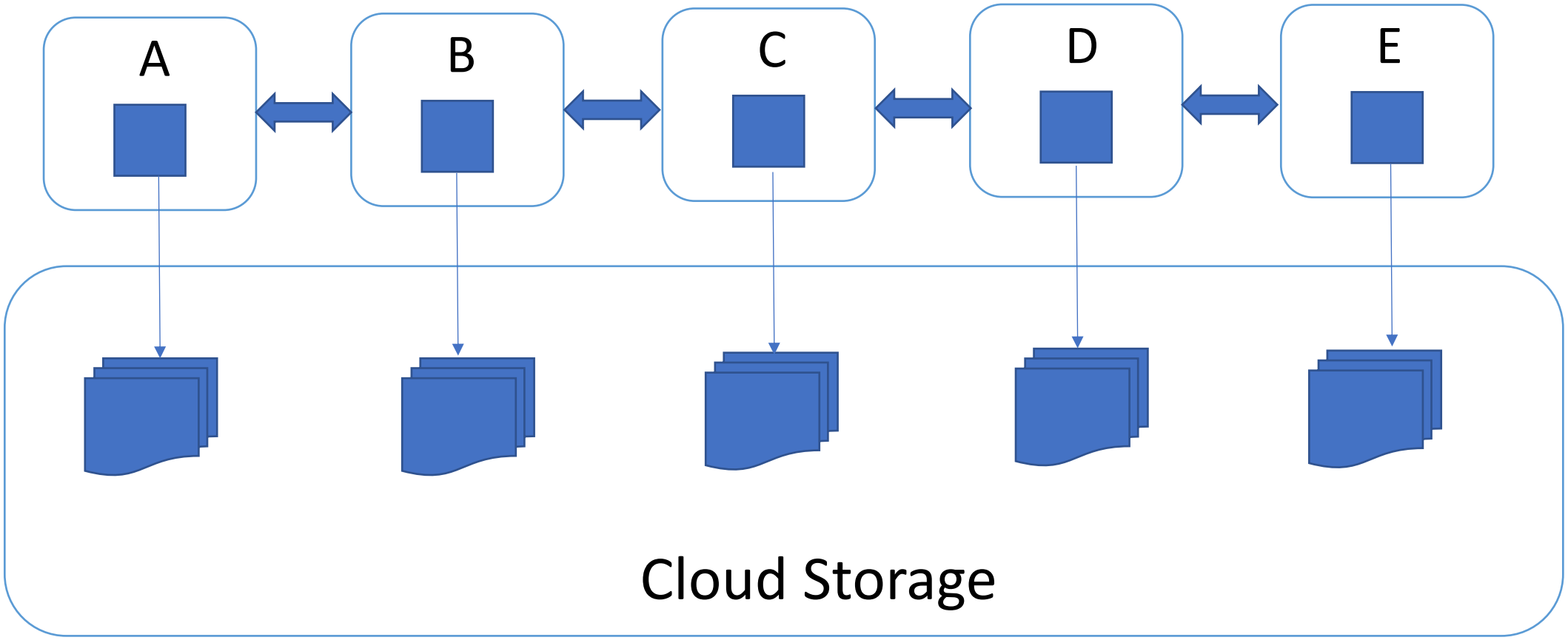
Loose Coupling = Faster Innovation

More loosely coupled execution units

Reduces team interdependencies

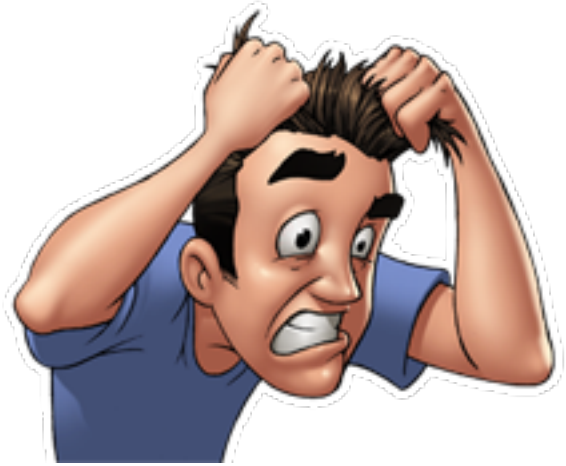
Faster Innovation!

Pattern: Schema Isolation Across Services



Problems with Distributed Data

- How do we query scattered data?
- How do we keep data consistent?



Cannot use ACID Transactions

BEGIN TRANSACTION

SELECT ADDRESS **FROM** CUSTOMERS **WHERE** CUSTOMER_ID = XXX

SELECT PRICE **FROM** PRODUCTS **WHERE** CUSTOMER_ID = YYY

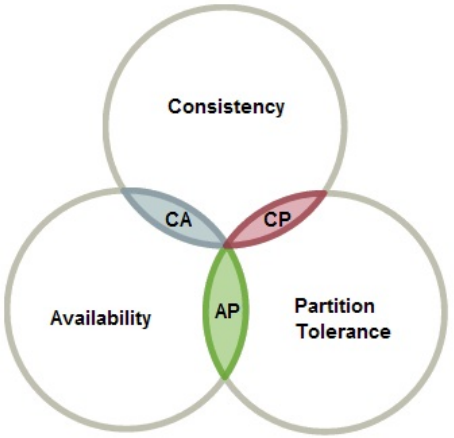
INSERT INTO ORDERS ...

COMMIT TRANSACTION

Private to Customer service

Private to Product service

Eventual Consistency



~~Consistency~~

Availability

Partition (Network)

Eventual Consistency

Use Event Driven Microservice Architecture!

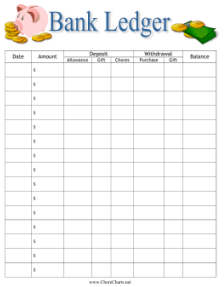
Event Driven Architecture: Introduction

- Event occurs when a change happens in system
- All listeners get notified of the event, may take action
- Highly distributed/loosely coupled architecture
- Often used for asynchronous flows of information

Event Sourcing: Introduction

- Modeling state changes as sequence of events
- Storing the event that could trigger the state change
- Enables rolling back to particular time in history

Examples:



Event Sourcing: Benefits & Drawbacks

Benefits:

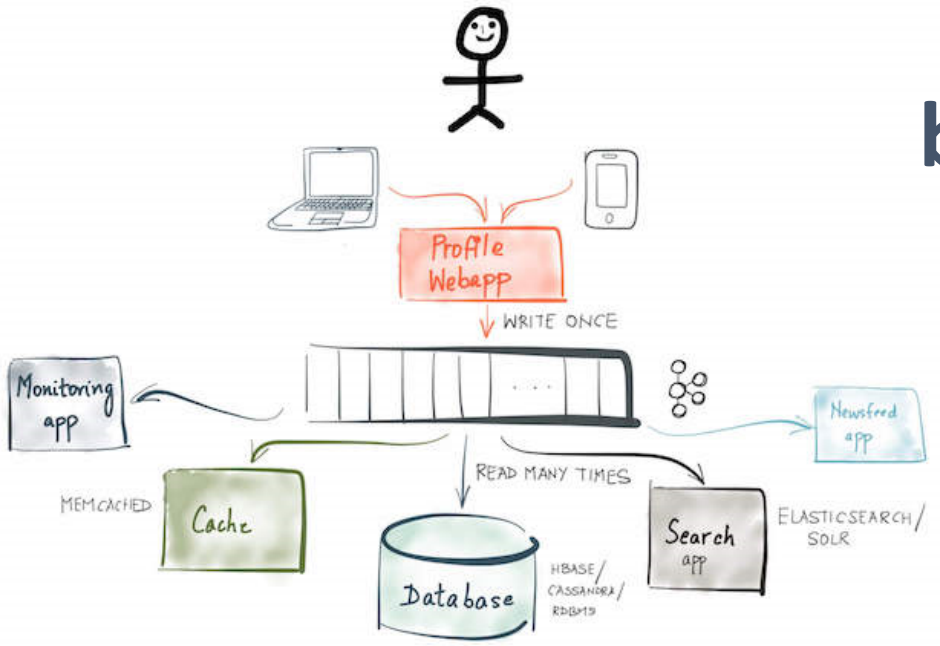
- 100% accurate audit logging
- Easy temporal queries
- Process same events but create views

Drawbacks:

- Adds Complexity
- No Strict Consistency
- Longer bootup times (Snapshots can help)

Event Sourcing: Multiple views

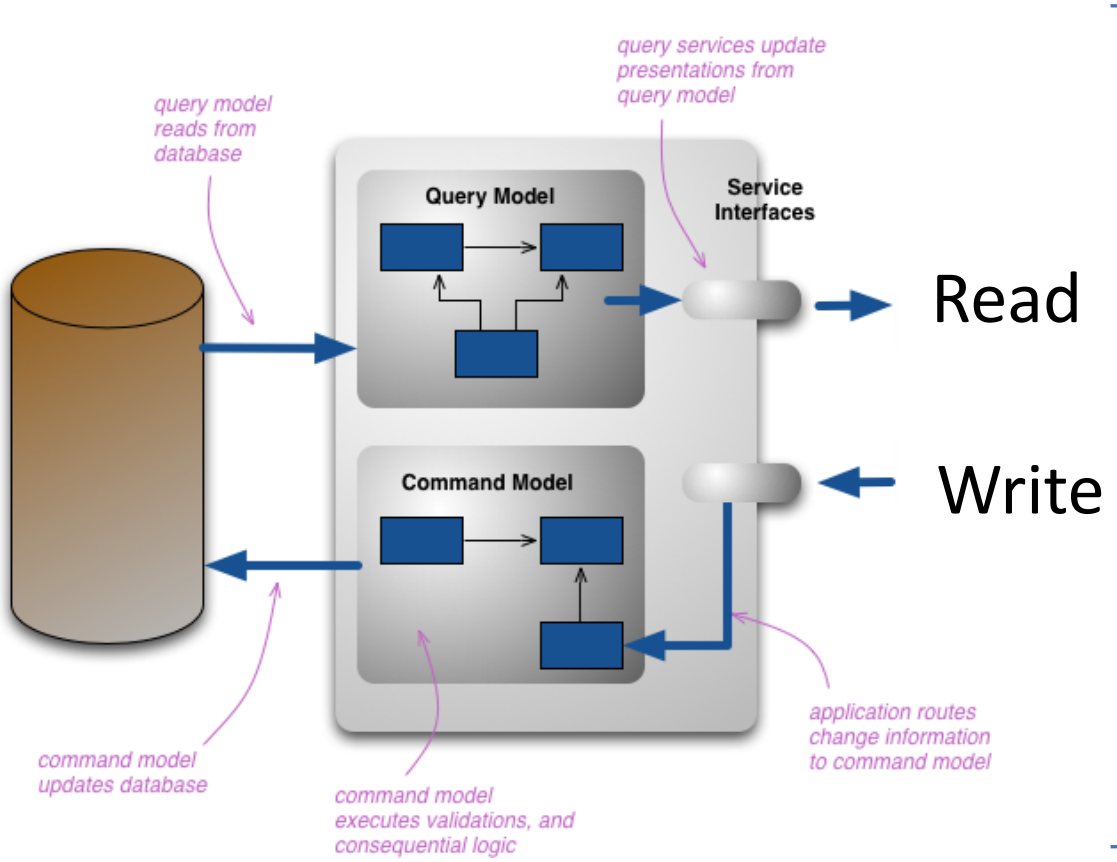
Adding applications that process event …



but create a different view!

Image Source: https://www.confluent.io/wp-content/uploads/2016/09/Event-sourced-based-architecture.jpeg

Command Query Responsibility Segregation



CQRS Pattern

Must for Event Sourcing!

Image Source: https://martinfowler.com/bliki/CQRS.html

CQRS: Benefits & Drawbacks

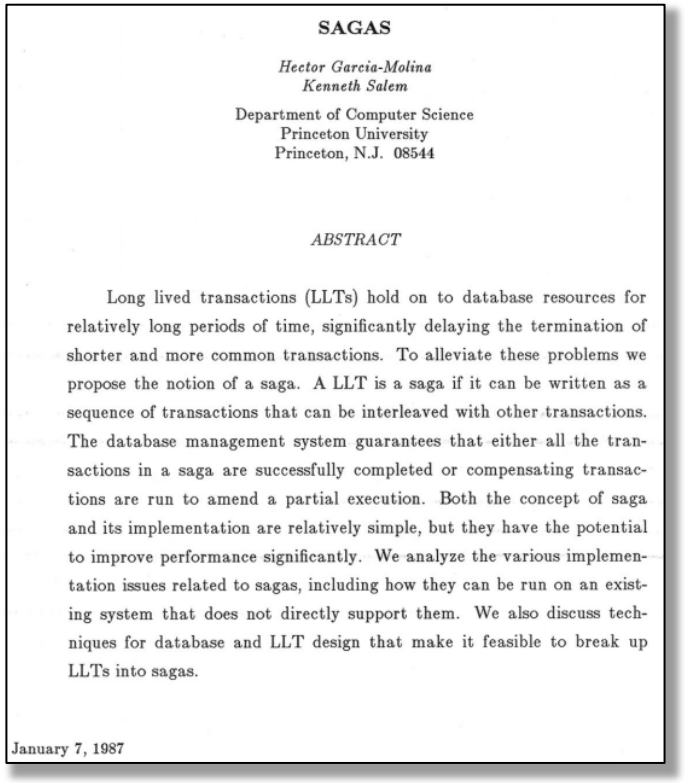
Benefits:

- Needed for Event Sourcing
- Improved separation of concerns
- Supports scalable multiple denormalized views

Drawbacks:

- Increased complexity
- Potential code duplication
- Replication lag as No Strict Consistency

Sagas: Introduction



SAGAS

Hector Garcia-Molina
Kenneth Salem

Department of Computer Science
Princeton University
Princeton, N.J. 08544

ABSTRACT

Long lived transactions (LLTs) hold on to database resources for relatively long periods of time, significantly delaying the termination of shorter and more common transactions. To alleviate these problems we propose the notion of a saga. A LLT is a saga if it can be written as a sequence of transactions that can be interleaved with other transactions. The database management system guarantees that either all the transactions in a saga are successfully completed or compensating transactions are run to amend a partial execution. Both the concept of saga and its implementation are relatively simple, but they have the potential to improve performance significantly. We analyze the various implementation issues related to sagas, including how they can be run on an existing system that does not directly support them. We also discuss techniques for database and LLT design that make it feasible to break up LLTs into sagas.

January 7, 1987

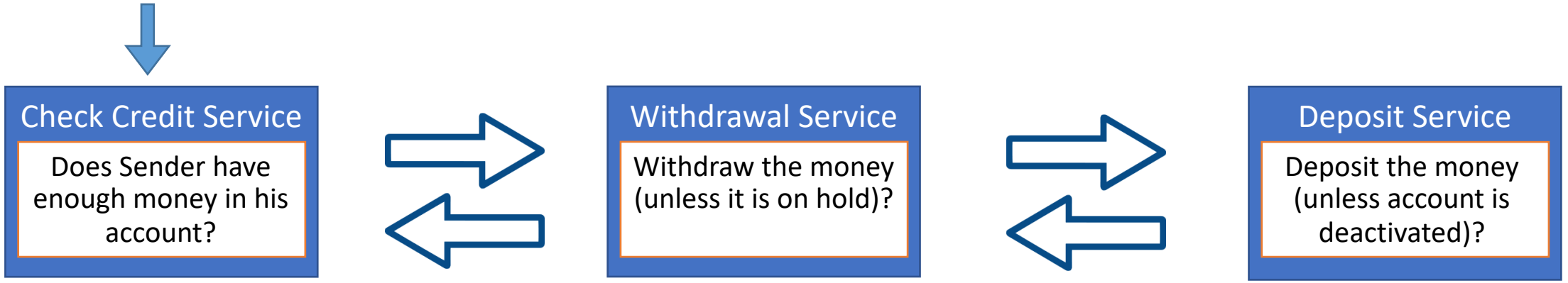
- Based on a 1987 paper
- Initially for a single database running on one node
- Now adapted for distributed systems with asynchrony and partial failure

Introducing Sagas

Long running transactions …

use compensating actions to handle failures!

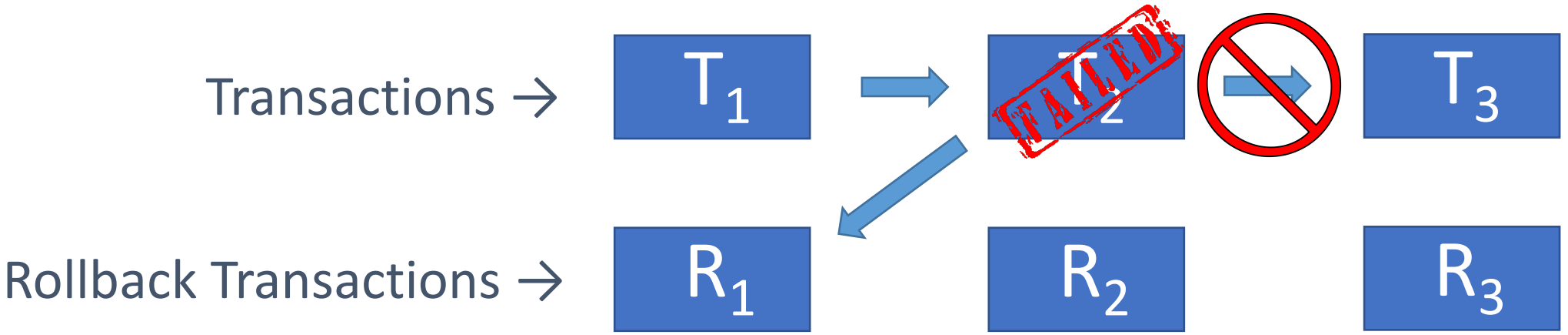
Deposit Check ← This action initiates the saga



Example Source: https://www.youtube.com/watch?v=YPbGW3Fnmbc

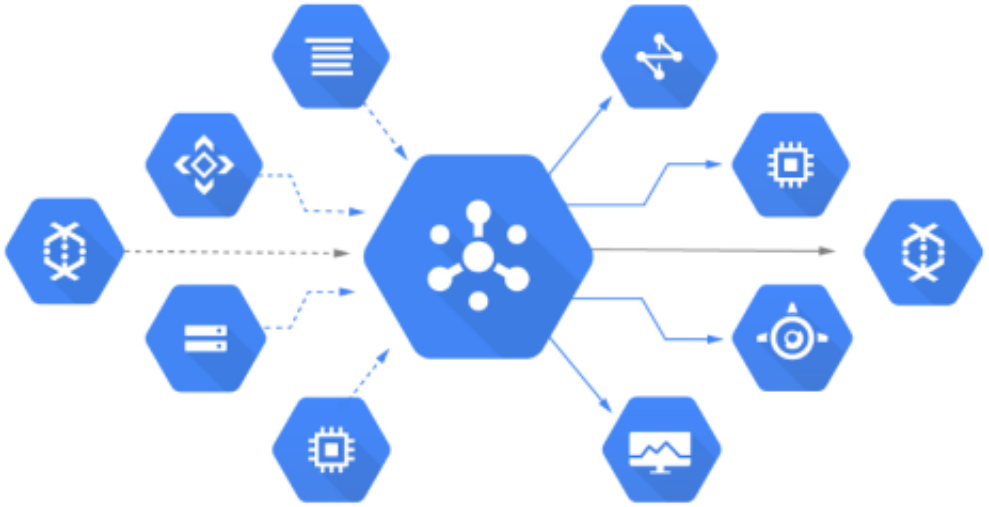
Transaction & Rollback Transaction

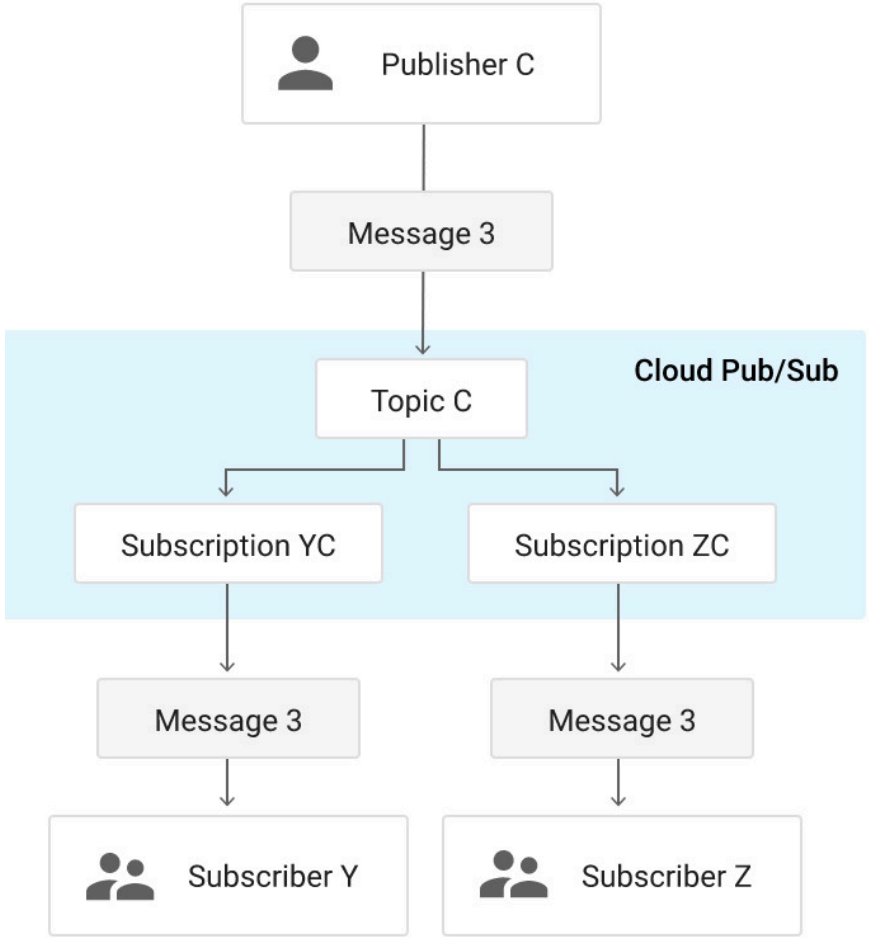
- Every Transaction has a Rollback transaction
- This logic must be included in the service



GCP Pub/Sub: Event bus

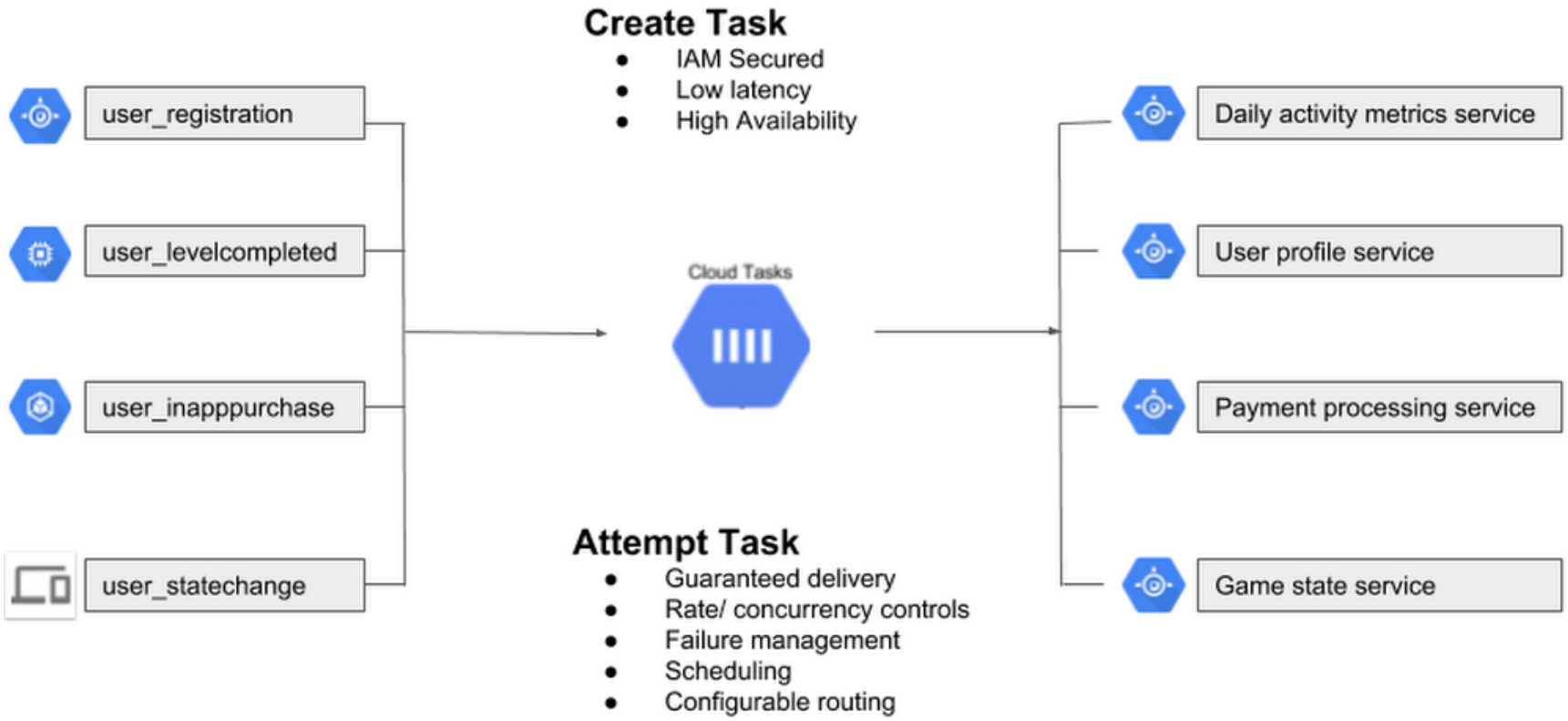
Flexible & Reliable Enterprise grade message bus





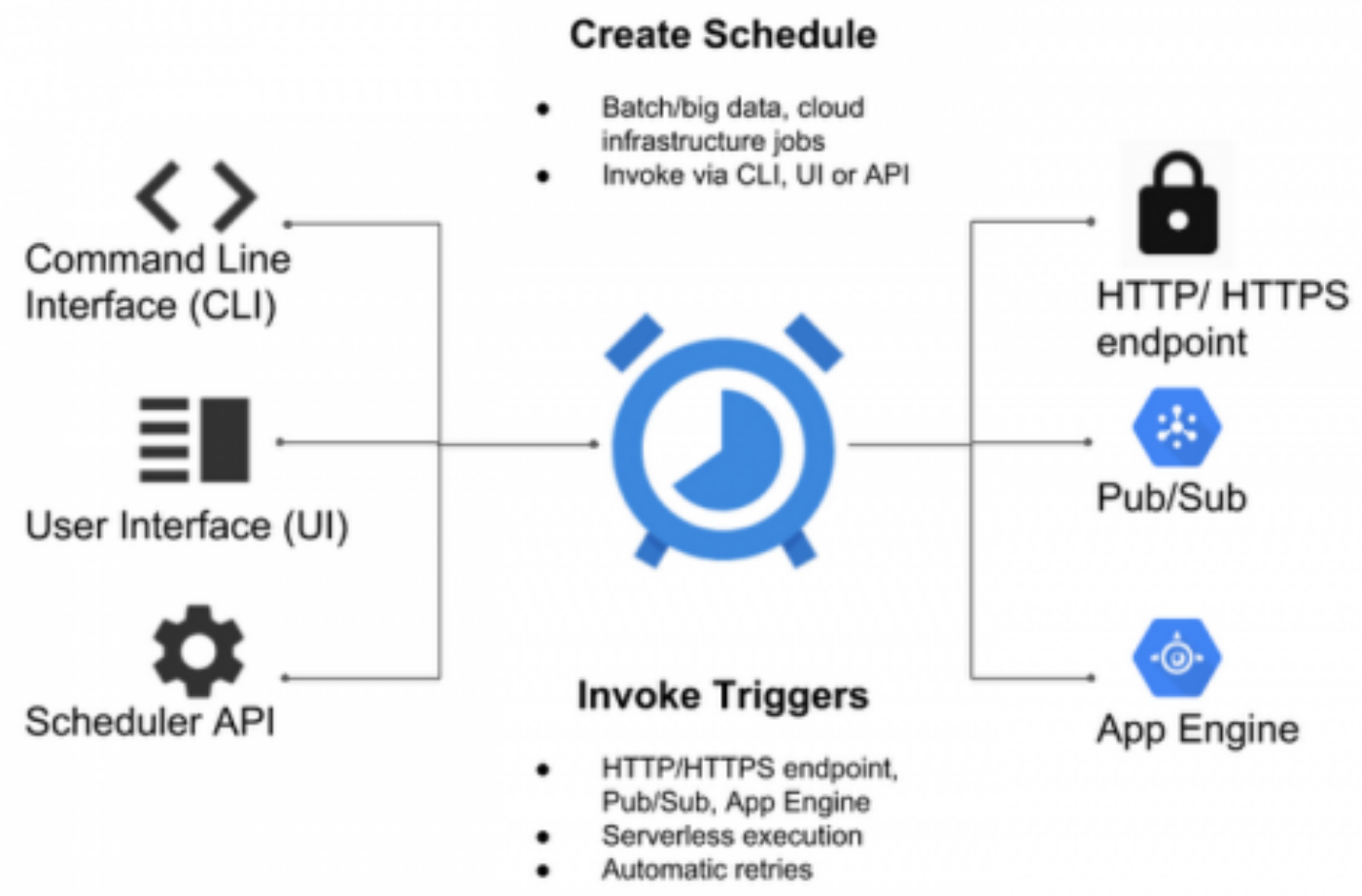
GCP Cloud Tasks

Fully managed distributed Asynchronous task queues



GCP Cloud Scheduler: Managed Cron

Fully managed, enterprise-grade scheduler



Migrating Microservices to Cloud Run

- Decoupling components
- Data first to Cloud Storage
- Message Queues next ones
- “Lift and Shift” code

Advantages of Cloud Run Microservices

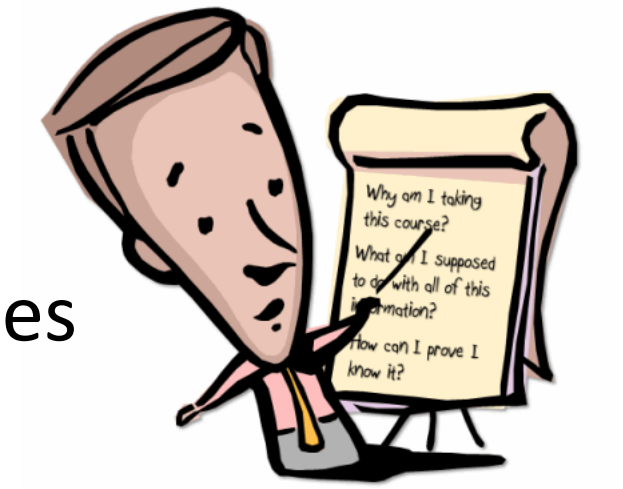
- Focus on application code, underlying Runtime & OS all managed
- Out of the box Auto-Scaling and Load Balancing support
- More cost effective, Pay as you go model

Disadvantages of Cloud Run Microservices

- Cold Start problem
- Dependence of certain technologies (Kubernetes etc.)
- Vendor Lock-in (to some extent)

Summary

- Microservice has high management overhead
- Serverless has much lower overhead
- Containers are popular ways for packing services
- Cloud Run brings containers to serverless



Implementing Microservices as Cloud Run application retains the benefits but avoid the drawback

DevOops

Nikhil Barthwal
Product Manager (Serverless),
Google Cloud Platform

nikhilbarthwal@yahoo.com

www.nikhilbarthwal.com

Questions?