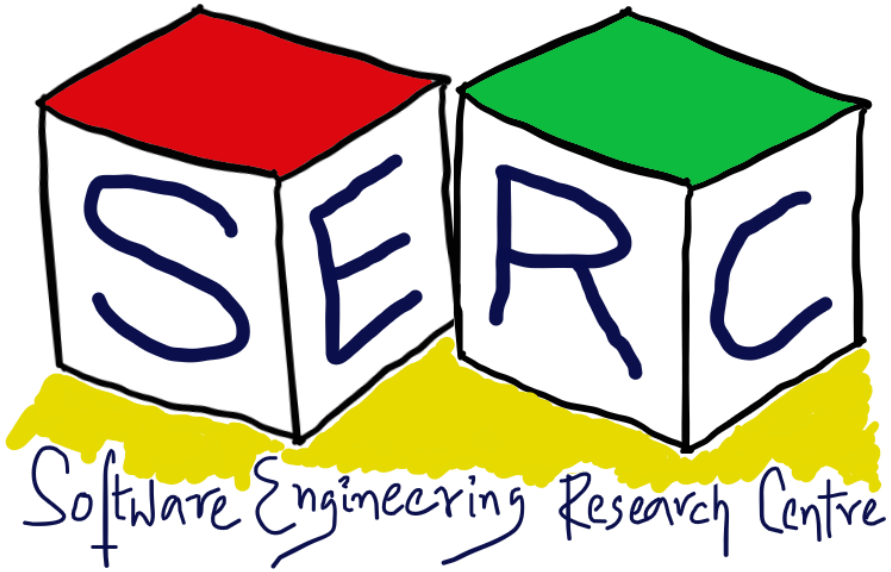
Microservices and EDA

CS6.401 Software Engineering



Dr. Karthik Vaidhyanthan

karthik.vaidhyanathan@iiit.ac.in

https://karthikvaidhyanathan.com

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

Acknowledgements

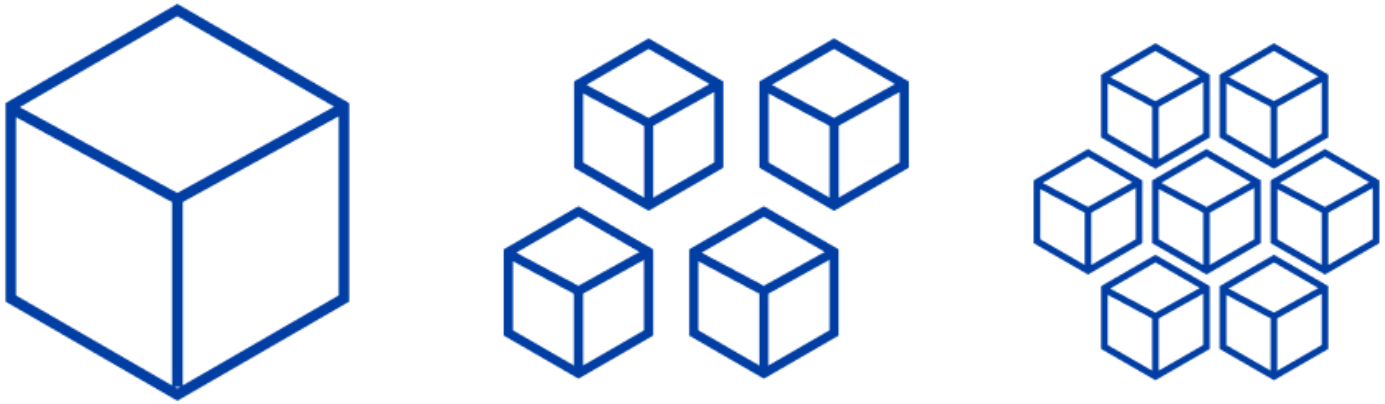
The materials used in this presentation have been gathered/adapted/generate from various sources as well as based on my own experiences and knowledge -- Karthik Vaidhyanathan

Sources:

1. Building Microservices, Sam Newman, 2nd edition
2. Various sources from the web that has been duly credited in the respective slide

Microservices: Quick Recap

Moving Towards Microservices



MONOLITHIC
Single unit

SOA
Coarse-grained

MICROSERVICES
Fine-grained

1990 2000 2010

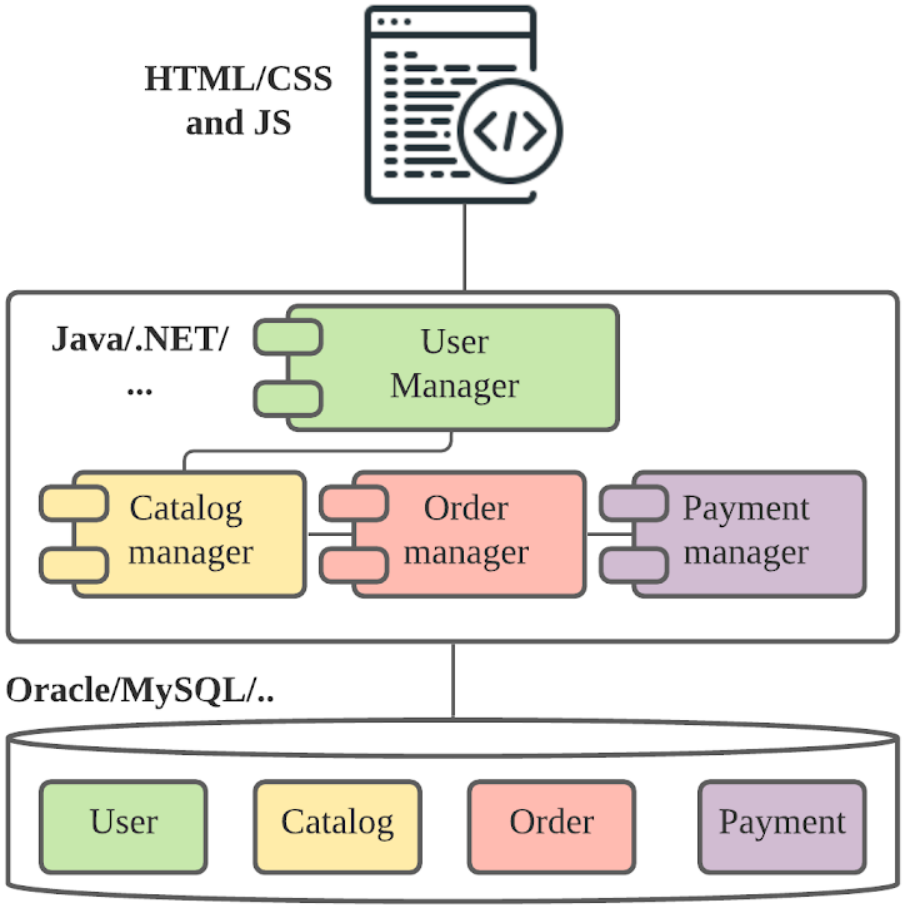
Microservices: What does it Mean?

"Small autonomous services that work together" -- Sam Newman

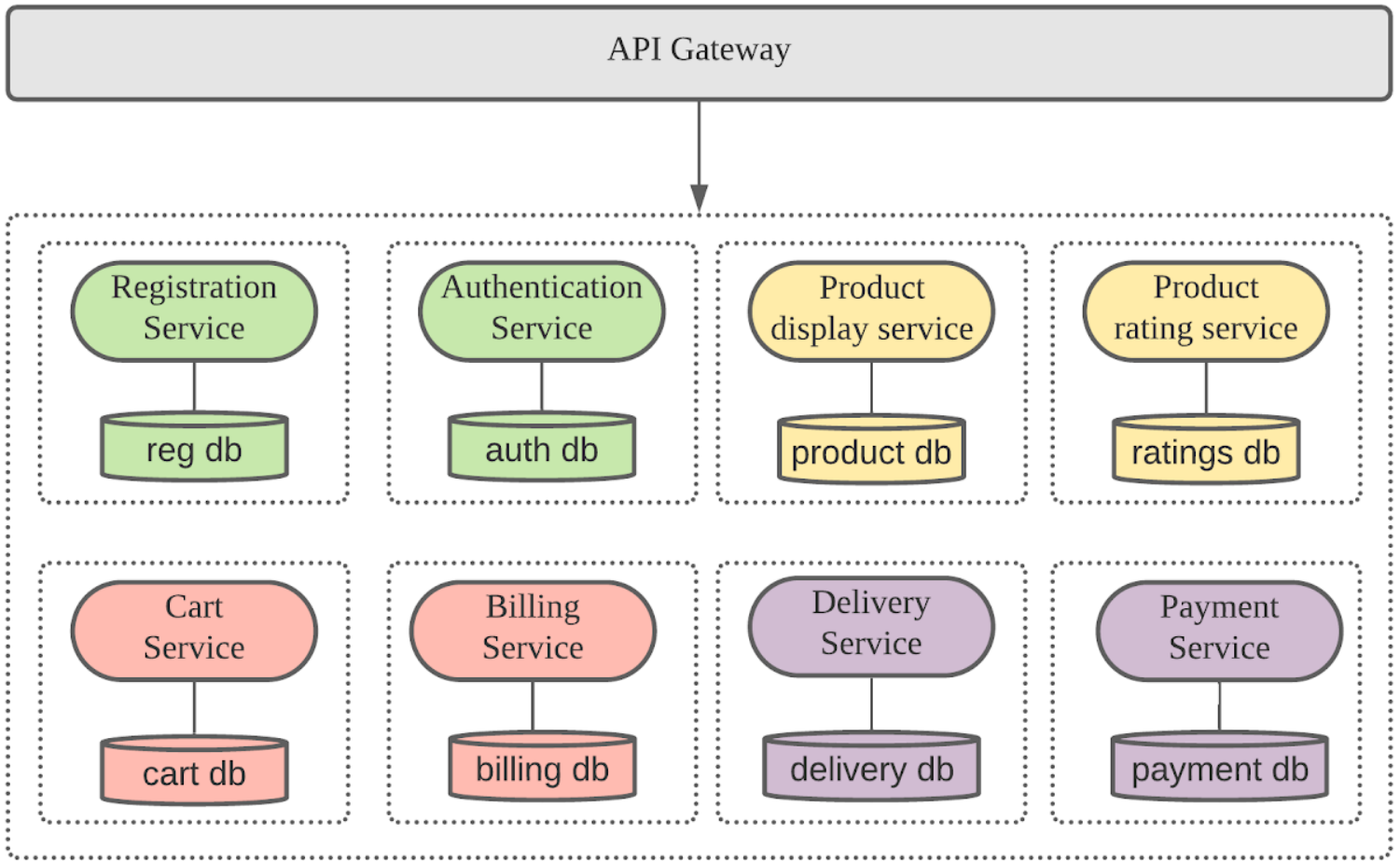
"It is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API" -- Martin Fowler

Microservices: What does it Mean?

Monolithic Version



Microservices Version



How to design?

Follow the principle of bounded contexts

- Identify different contexts inside the main domain [organizational boundary]
- Only share what is important rest remains within context

Ensure loose coupling

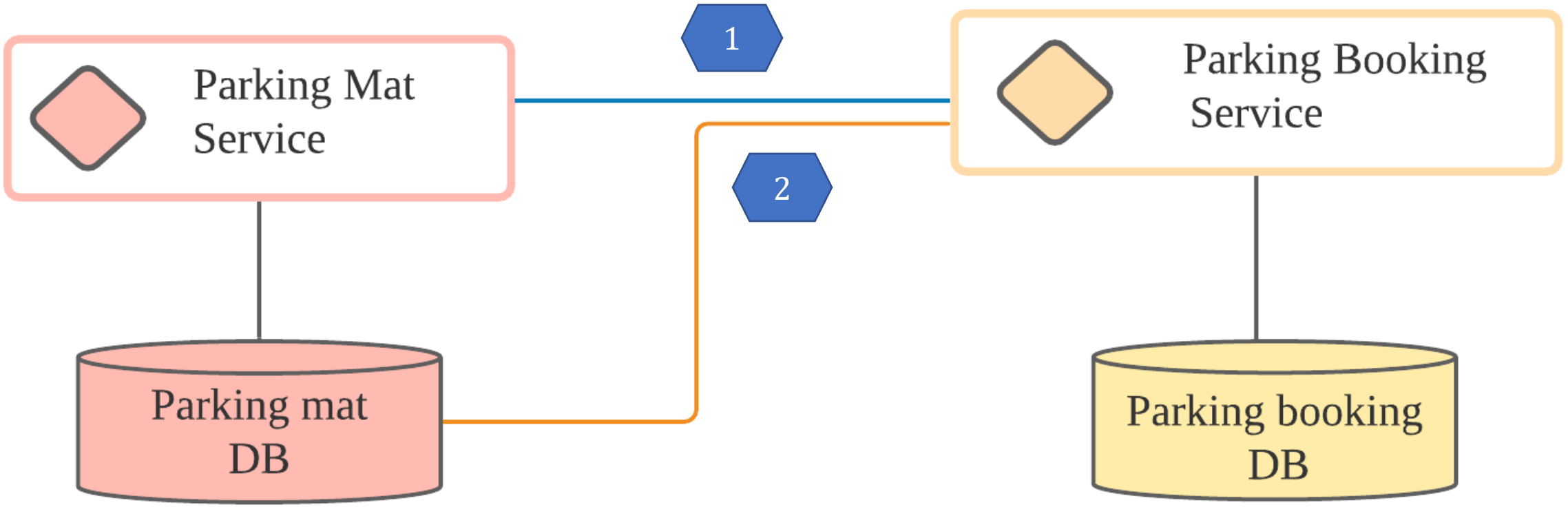
- Minimize coupling between microservices
- Should be easy to change and deploy one without affecting others
- Each microservice needs to know as little as possible about others

Maintain high cohesion

- Bundle one end to end feature or complete part of it inside one microservice
- Promotes robustness and reliability
- One change should never require change in 10 different places

Microservices Integration: Overview

Integration with Shared DB?



1 or 2 ?

Shared DB Integration?

Avoid integration with shared db as much as possible:

- Changing DB schema based on one microservice need affects others
- Affects evolution of system eg: changing from relational to non-relational
- Choice of DB might constrain the choice of language for implementing microservice eg: Java might have more db driver available for MySQL
- Goodbye high cohesion and loose coupling !!!

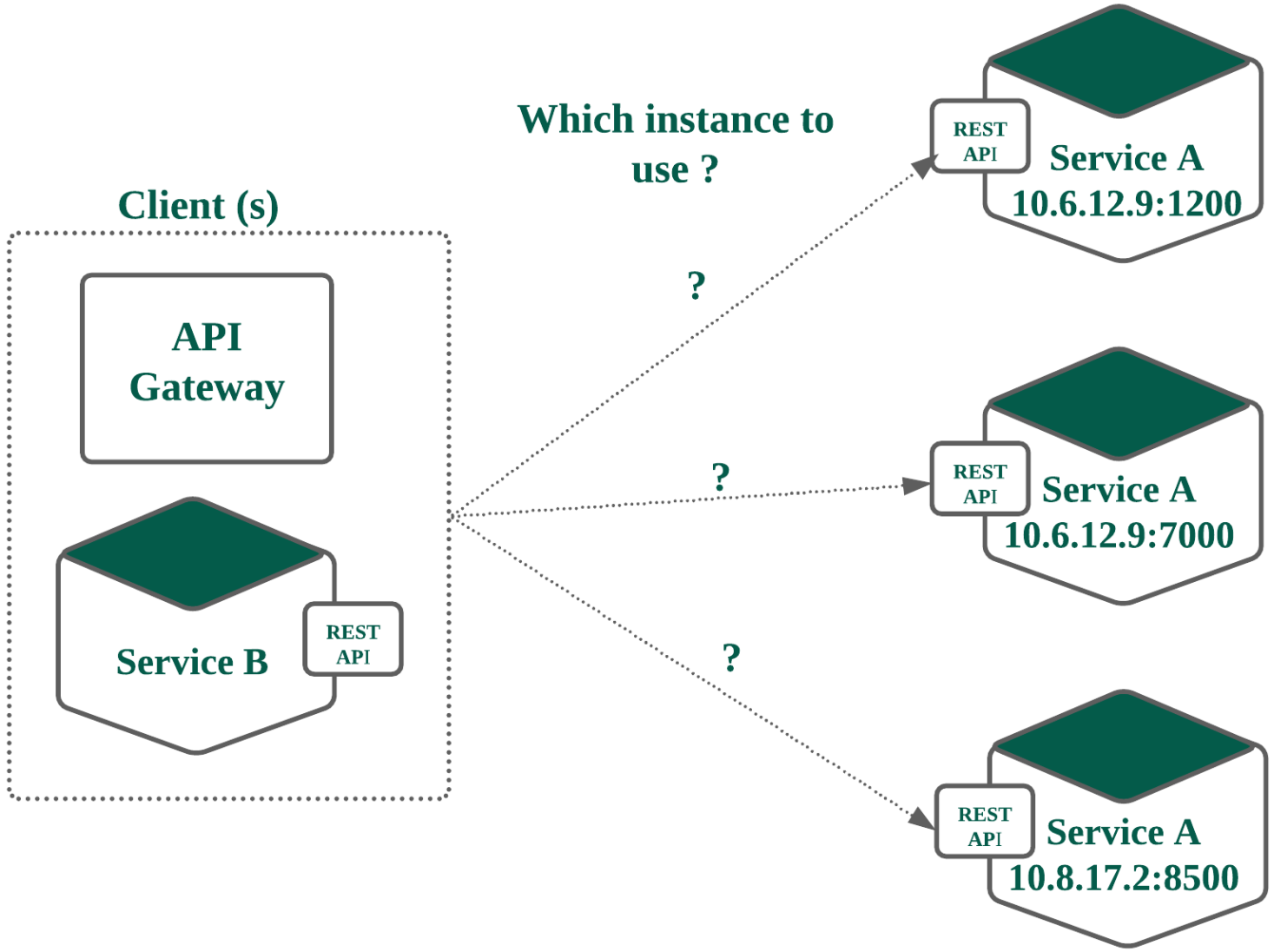
Microservices Communication

Many things to Consider

- Synchronous Vs Asynchronous
- Orchestration v Choreography
- REST vs GraphQL
  - JSON vs XML vs Protobuf
- Communication Patterns exist

How do services discover other service instances?

Service Discovery

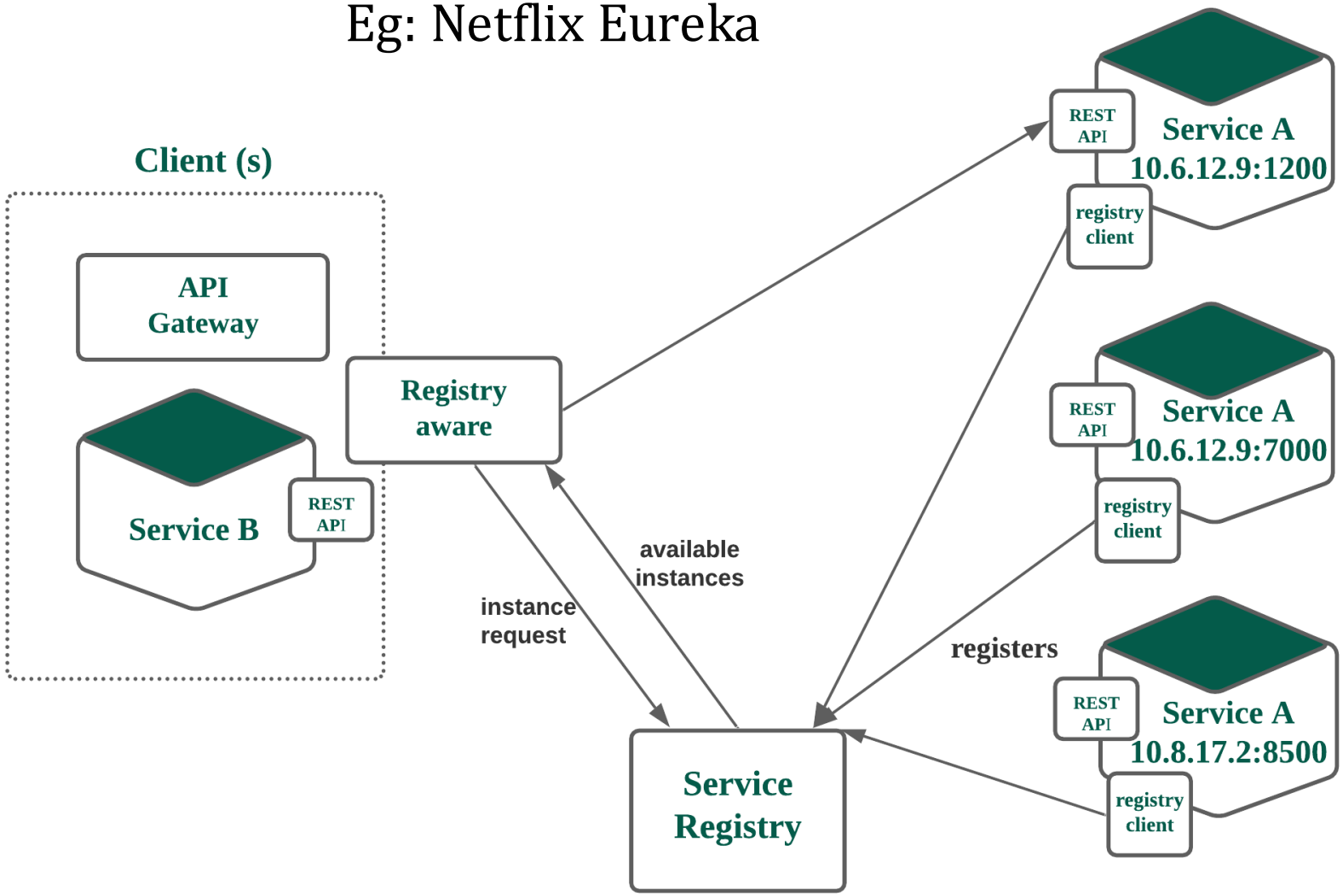


Use Service Registry!!

Client-side Service Discovery

- Each microservice registers itself to service registry (as and when they are available)
- Service registry responds with the instance of the requested service to client
- Fewer network calls (just query service registry)
- Coupling between client and service registry

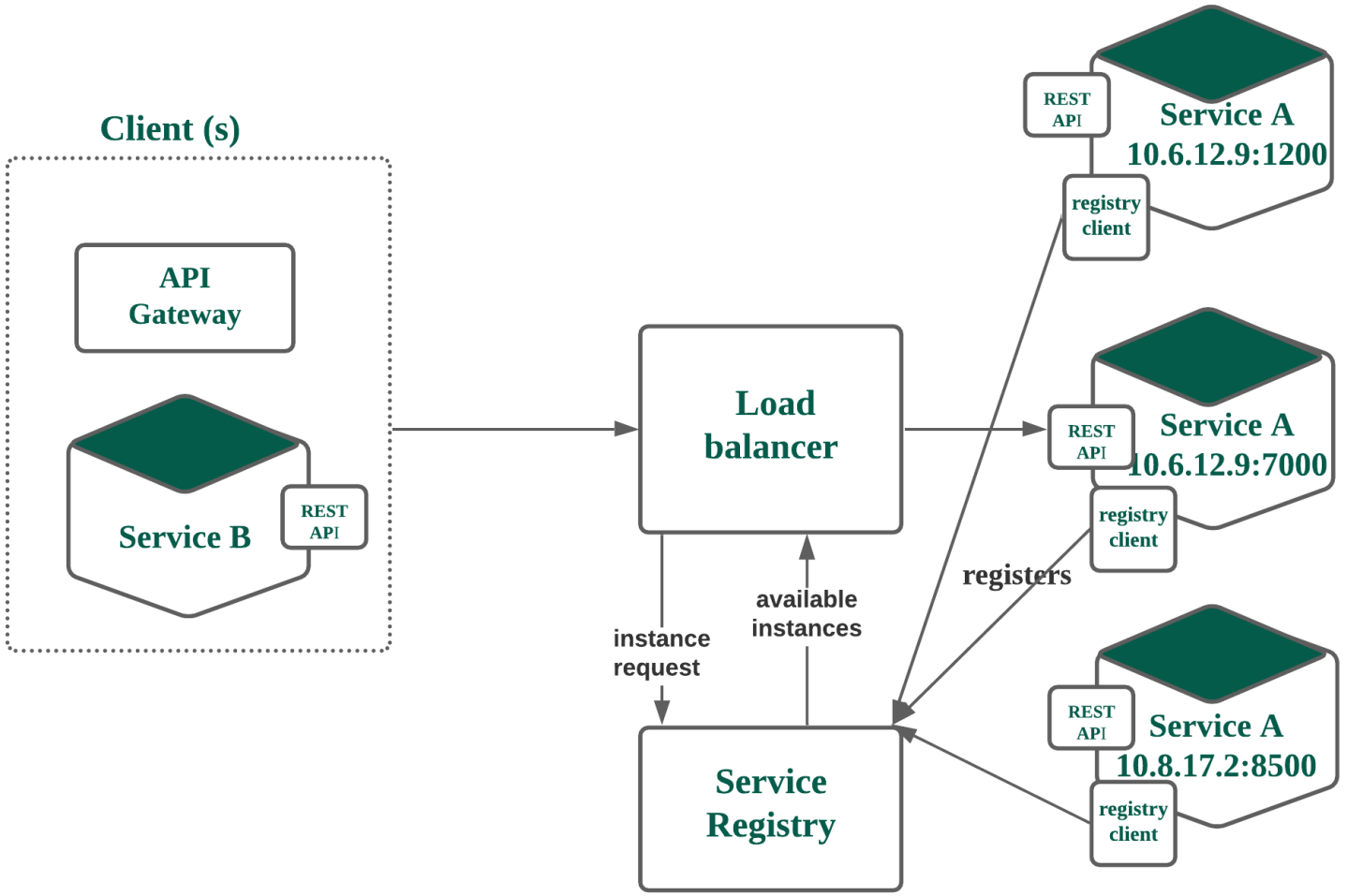
Eg: Netflix Eureka



Server-side Service Discovery

- Client (s) sends request to API gateway or load balancer
- The load balancer or API gateway uses Service registry to discover services
- Separation of logic from client
- Load balancer needs to be managed and replicated
- Additional network hop

Eg: Amazon ELB, Zookeeper



Is Microservice the holy grail?

Some Funny Quotes but makes sense

Honest Status Page @honest_update · Oct 8, 2015

We replaced our monolith with micro services so that every outage could be more like a murder mystery.

21 3K 2.6K

Gert de Pagter @BackEndTea · Jan 7

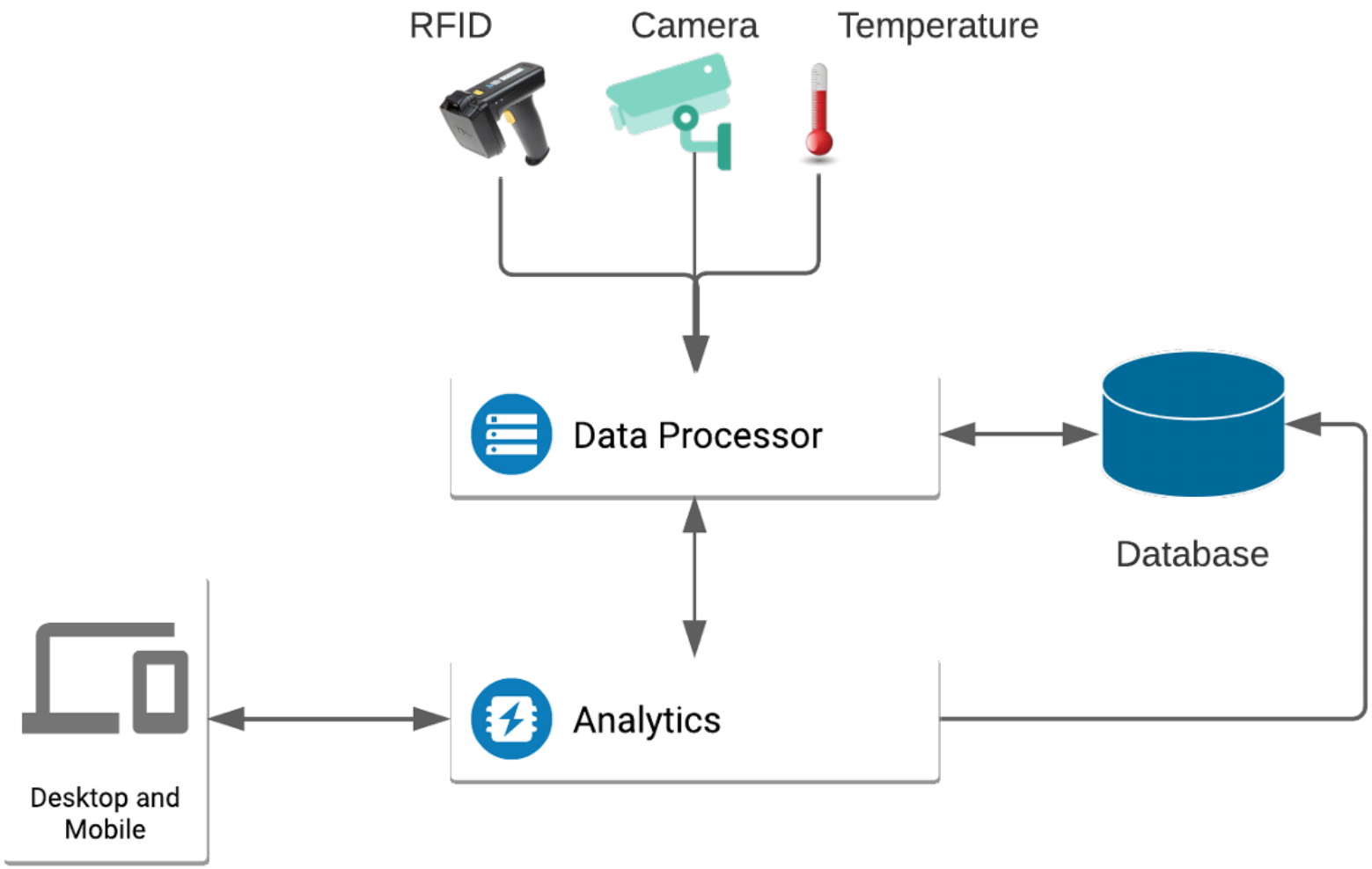
Thanks to **microservices**, our JOINS are now over HTTP.

39 345 1.4K

Monolith -> microservice but then we need docker, kubernetes, monitoring and what not !!!!

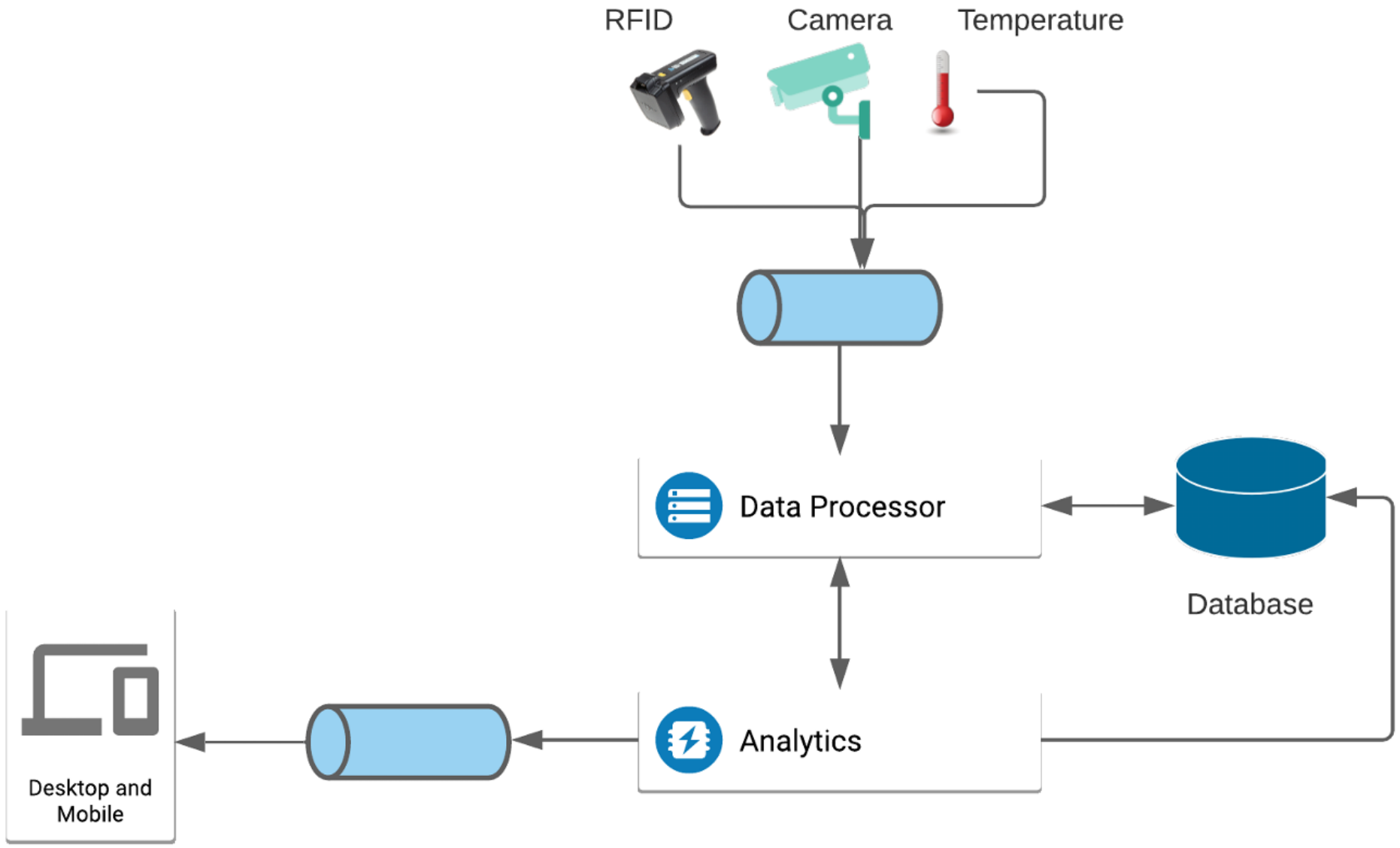
image source: twitter

EDA: An Intuition



What can be the issues with the above design?

Enters Event Driven Architecture



Event Driven Architecture: An Overview

- Independent components asynchronously emit and receive events communicated over event buses
- Produce, detect and consume events
- Highly decoupled components – Minimal amount of coupling (topics, queue names, etc.)

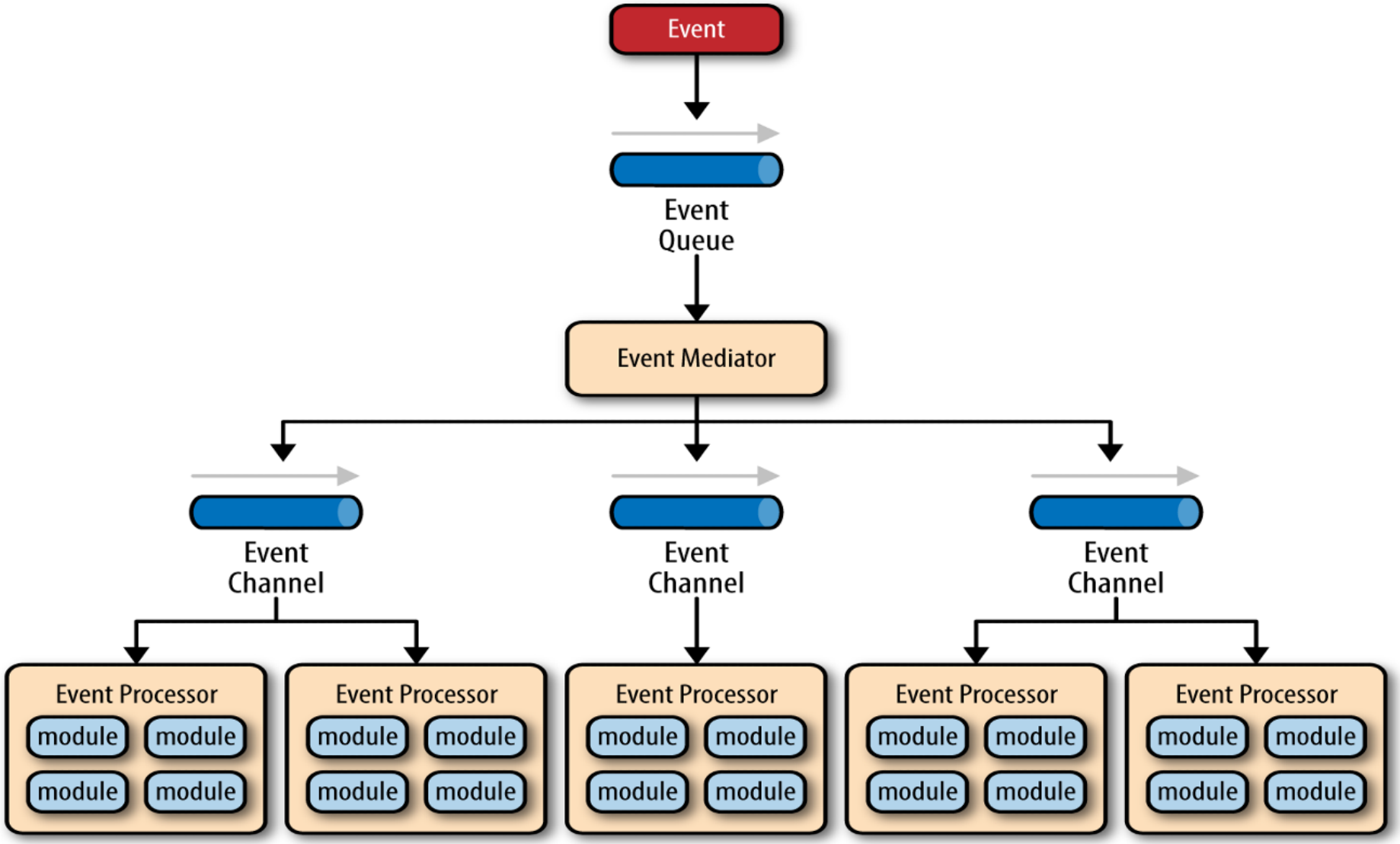
Design elements

- Components: concurrent event generators and event consumers
- Connectors: event bus (may be more than one)
- Data: events

Topology

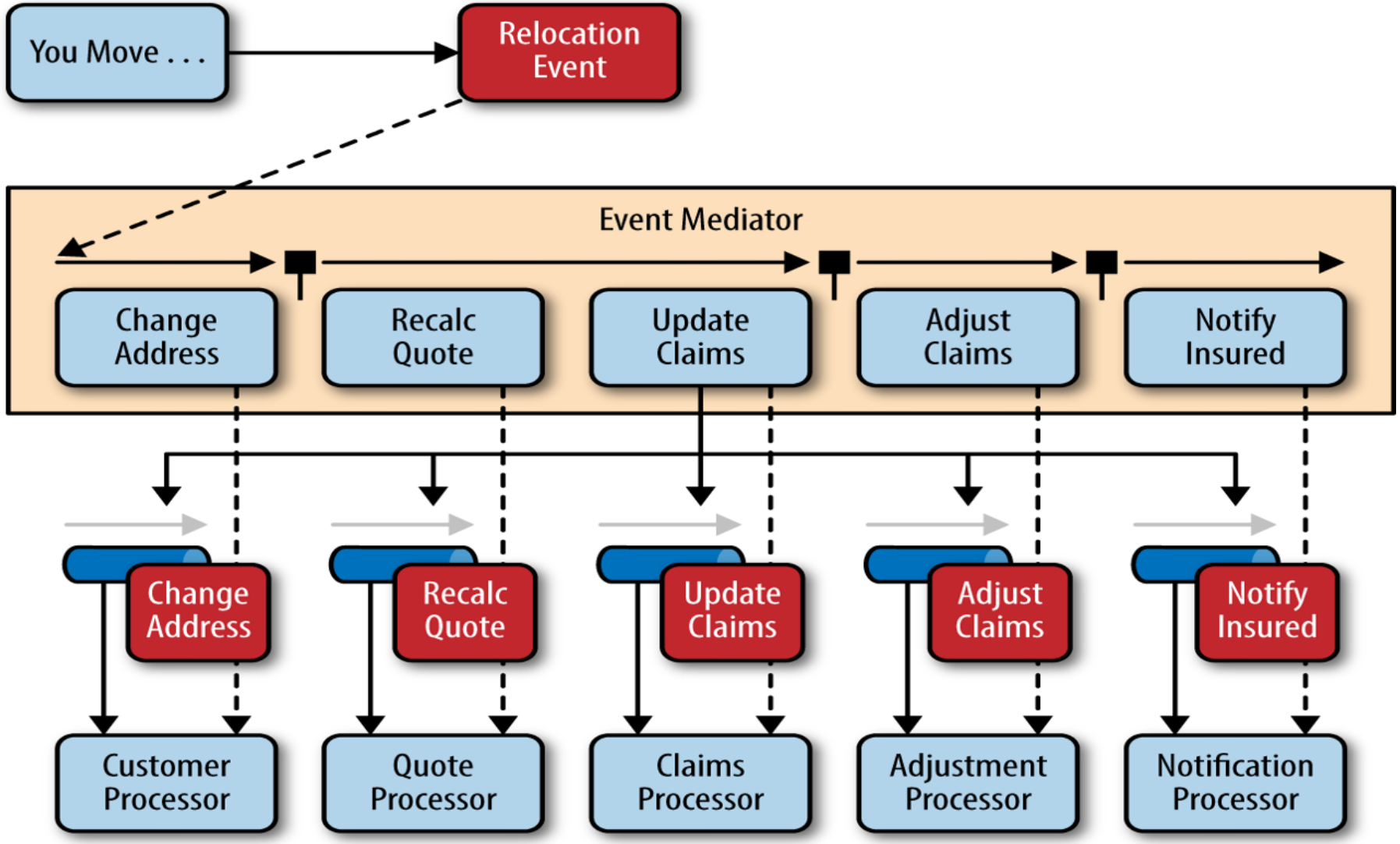
Communication via the event bus or link only (Mediator or Broker)

Event Driven Architecture: Mediator



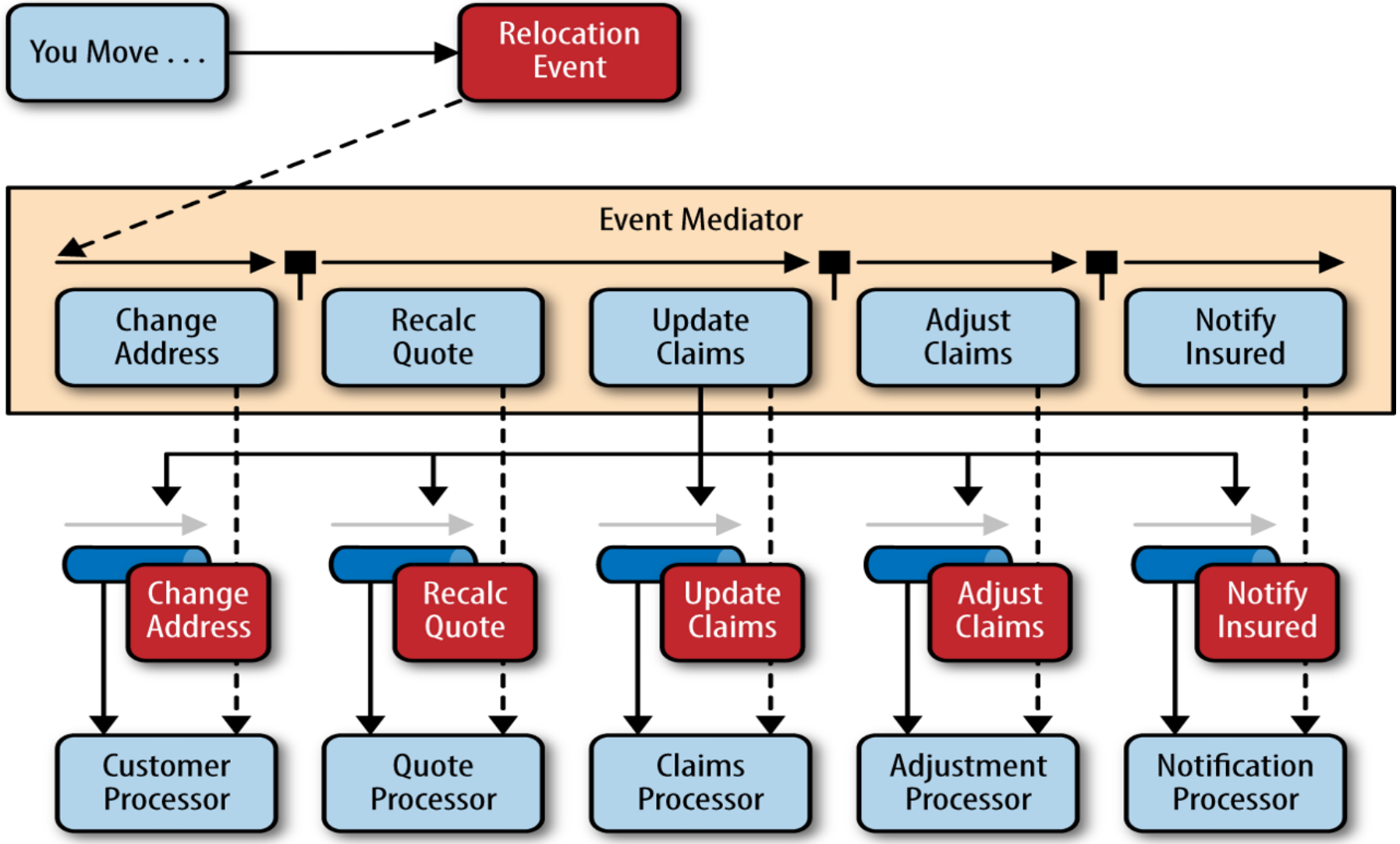
https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch02.html

Event Driven Architecture: Mediator



https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch02.html

Event Driven Architecture: Mediator



https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch02.html

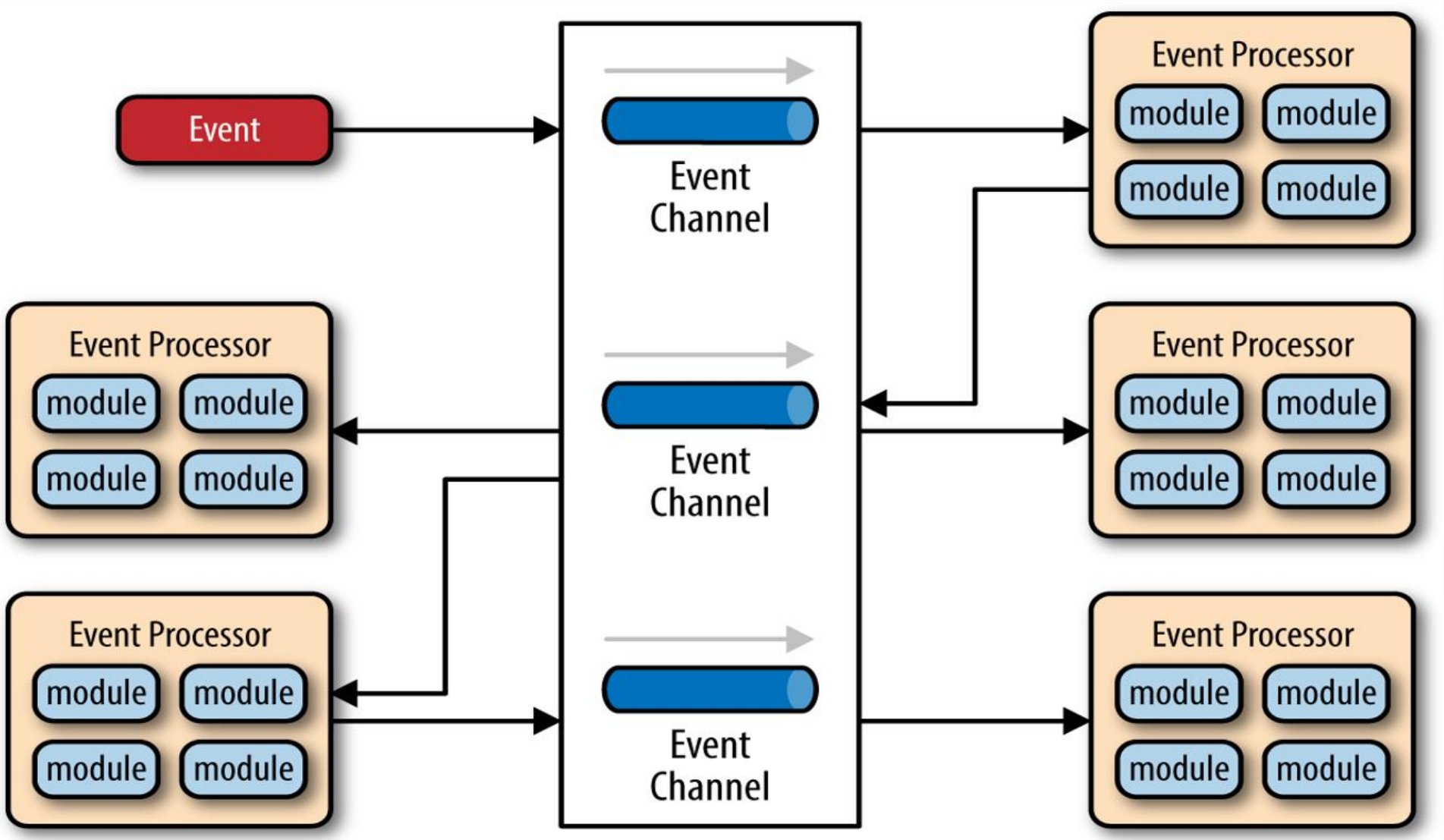
Mediator Topology: An Overview

Similar to the Orchestration in traditional SOA
Two key events – **Initial and Processing event**

Four main types of components:
1. **Event queue –** Responsible to transfer events to event mediator
2. **Event Mediator –** Orchestrates the processing of events to accomplish the overall functionality
3. **Event Channel** - Topics or queues to which events are ingested by mediator (eg: Kafka topic)
4. **Event Processor** - Implements the business logic
    1. Can be fine grained or Coarse grained)
    2. Advice: keep it to one functionality

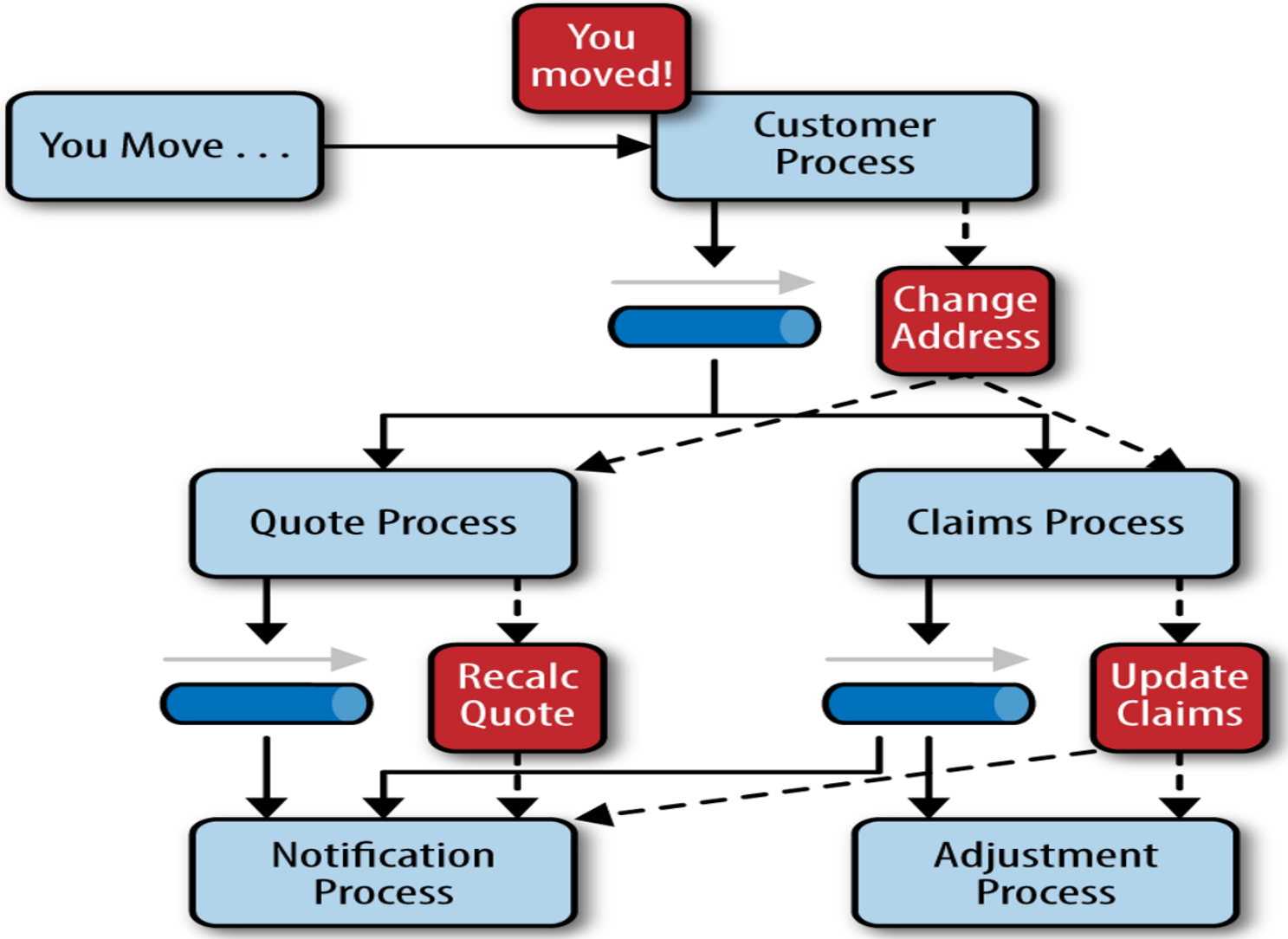
https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch02.html

Event Driven Architecture: Broker



https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch02.html

Event Driven Architecture: Broker



https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch02.html

Broker Topology: An Overview

- Similar to the Choreography in traditional SOA
- Two main types of components:

1. Broker – Consists of all the event channels for event processing. Can be topics or queues
2. Event Processor – Responsible for processing the event and sending a notification to the event channels

https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch02.html

How to Decide?

Advantages

1. High performance
2. High Scalability
3. Ease of Deployment
4. Ease of modifications/Evolved easily

Disadvantages

- Remote process availability – Liveliness of a consumer
- Lack of responsiveness
- Broker or mediator failures
- Testing can be tedious
- Development can be complex

https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch02.html

Thank You

Course website: karthikv1392.github.io/cs6401_se

Email: karthik.vaidhyanathan@iiit.ac.in
Web: https://karthikvaidhyanathan.com
Twitter: @karthi_ishere