

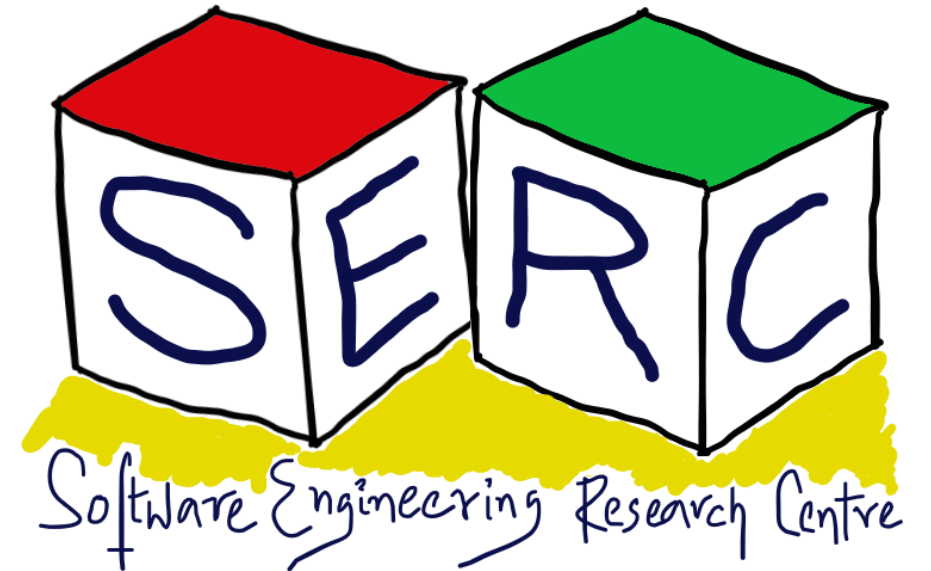
Designing Microservices

CS6.401 Software Engineering

Dr. Karthik Vaidhyanathan

karthik.vaidhyanathan@iiit.ac.in

<https://karthikvaidhyanathan.com>



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

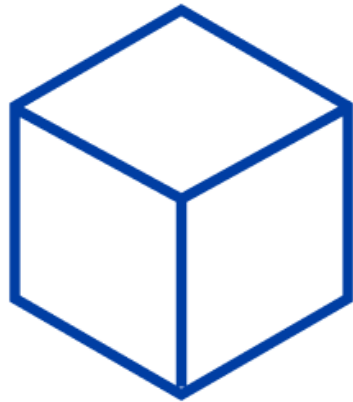
Sources:

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

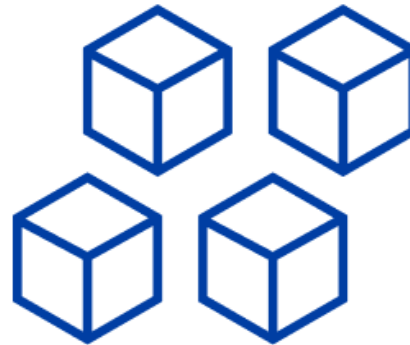
The slide features a light gray background with a large white diamond shape in the center. The diamond is outlined with a thin white border. In the four corners, there are overlapping geometric shapes: yellow diamonds in the top-left and bottom-right, and blue diamonds in the top-right and bottom-left. The text "Microservices: Quick Recap" is centered within the white diamond.

Microservices: Quick Recap

Moving Towards Microservices



MONOLITHIC
Single unit



SOA
Coarse-grained



MICROSERVICES
Fine-grained

1990

2000

2010

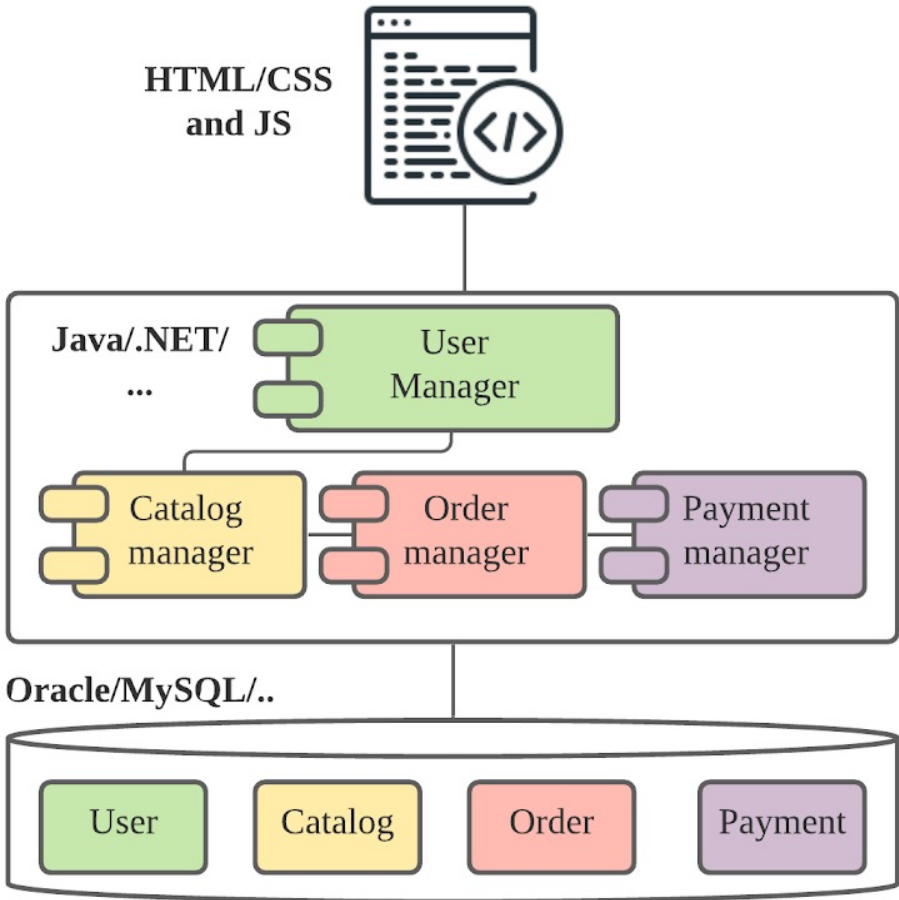
Microservices: What does it Mean?

“Small autonomous services that work together” -- Sam Newman

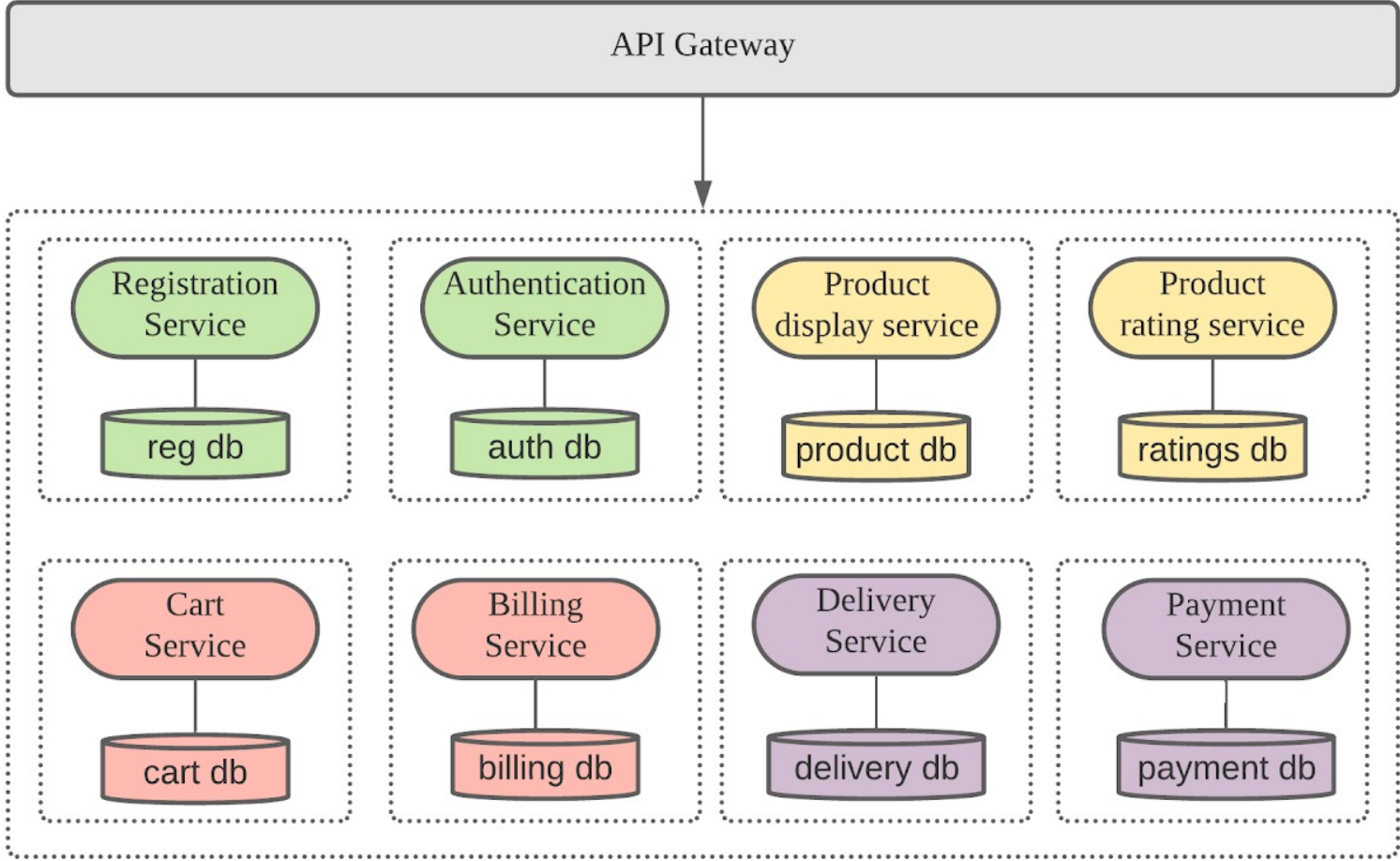
“It is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API” -- Martin Fowler

Microservices: What does it Mean?

Monolithic Version



Microservices Version



Microservices: Key Advantages

Scaling is Easy

- Scale only the required microservices
- Adding a new feature can be just adding one another microservice

Heterogeneity

- Each microservice can be developed in different technologies
- Experimenting with new technology is easy

Resilience

- Only specific microservices goes down
- Grouping microservices as critical and non-critical can be done to add more resilience

Microservices: Key Advantages

Organizational Alignment

- Easily distribute teams around microservices - eg: Amazon 2 pizza rule
- Minimize people working on one less codebase

Composability


- Easily compose microservices to get new functionality

Replaceability

- Cost of replacement is small - should not take more than 2 weeks
- Imagine replacing a 25 year old legacy system !!

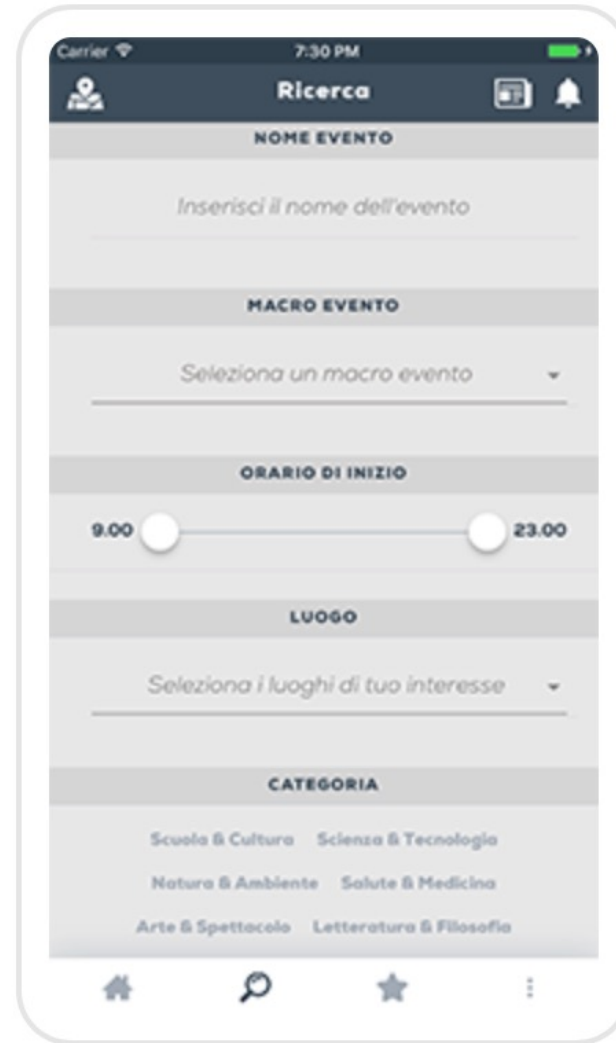
Ease of Deployment

- Check and rollback easily
- Continuous integration and deployment is easier - DevOps!!!



How to identify
Microservices? – Lets go
back to NdR Case

NdR Case Study



NdR Case Study

Goal: Develop a microservice based AI-powered event management system for NdR

Features: User registration, book venues, book parking lots, provide venue and parking lot recommendation, priority booking based on small payment, check weather

Data Sources:

- Parking mats at entrances and exits of parking lot to get count of cars
- Handheld RFID readers to capture the count of people entering venue
- Cameras at different locations to provide real-time video feed
- People counter at venue exits to count people exiting venue



Microservices – How to Design?

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



What are the contexts in
NdR?

Contexts within NdR

IoT

User

Booking

Weather

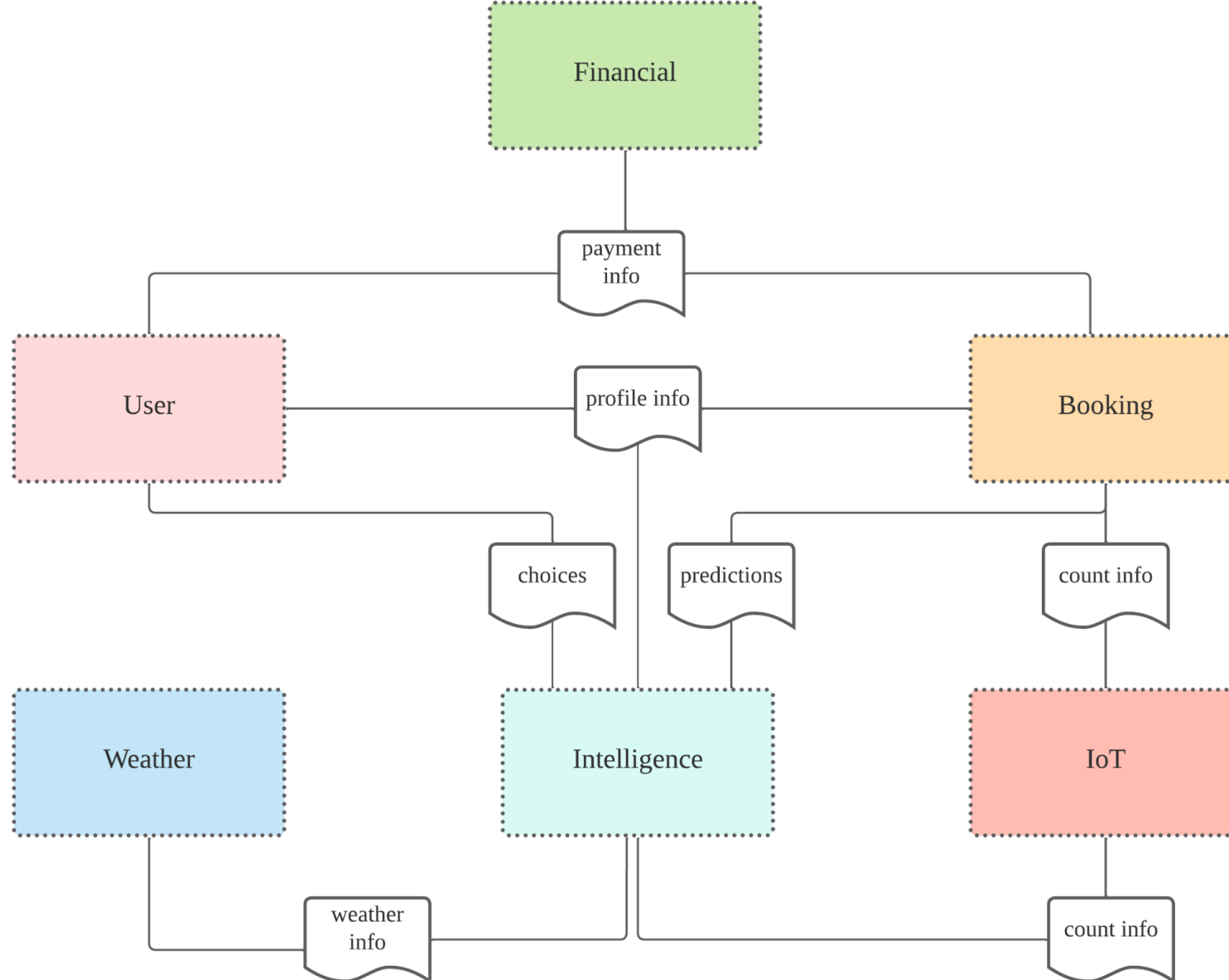
Intelligence

Financial



Hidden and Shared Models

Hidden and Shared Models



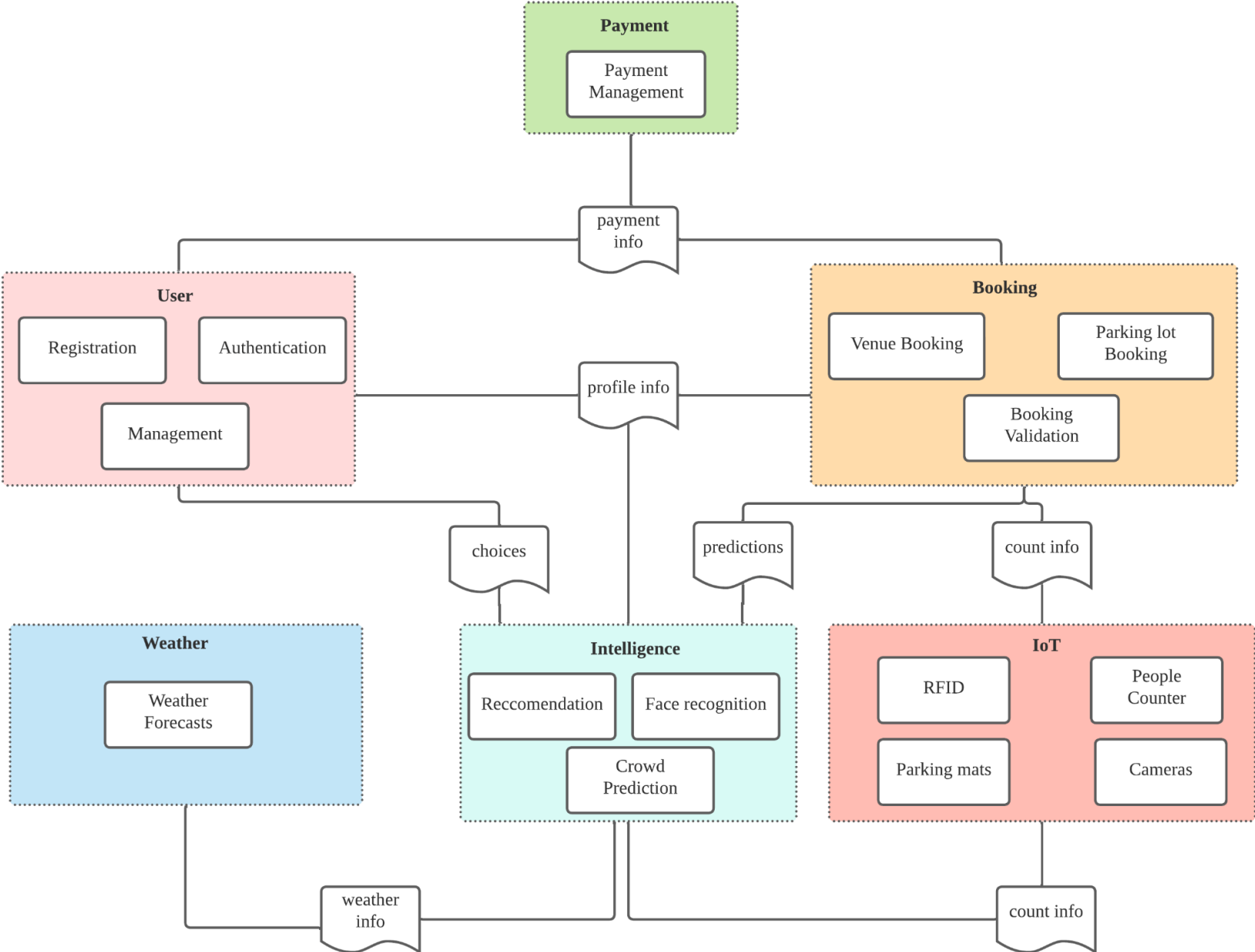
Shared and Hidden Models

- Identify what needs to be shared
 - Eg: Sharing of information on people and car count to booking context
- Same things may have different meaning in different contexts
 - Eg: Sensor data in IoT context and booking context
- This process will facilitate avoiding of high coupling (**Pitfall !!**)
- Microservices should never be chatty!
 - Adds to performance issues
 - Lack of cohesion
 - Eg: too many back and forth communication between two microservices



Modules and Services

Modules and Services in NdR



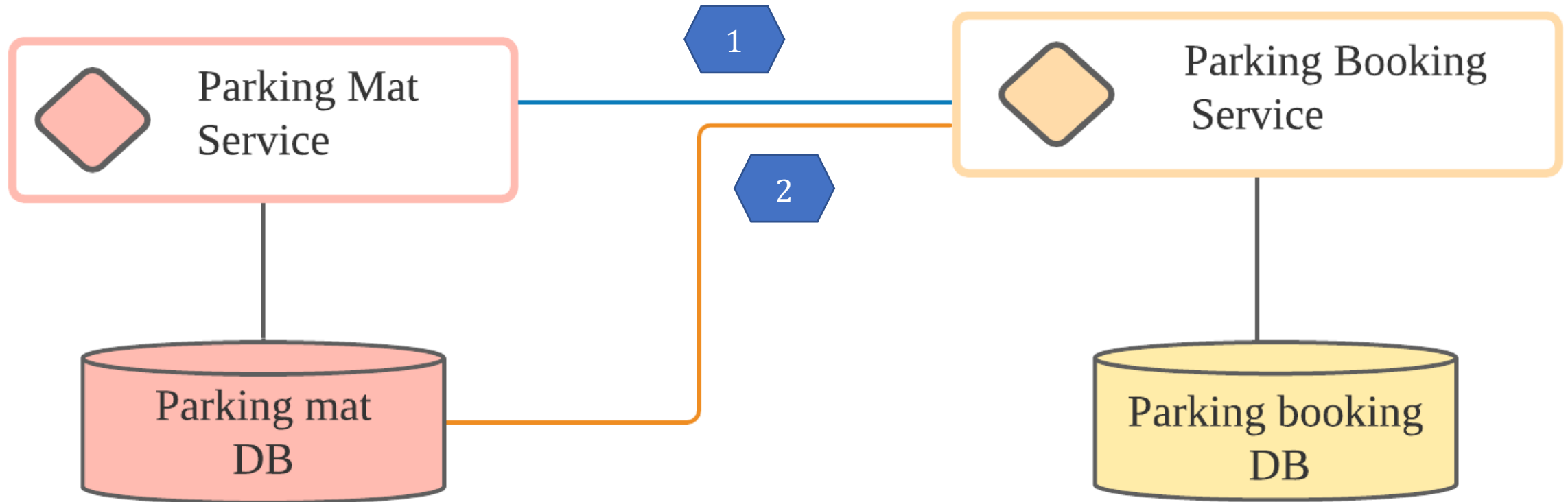
Shared and Hidden Models

- Separate the contexts into modules
 - Eg: Recommendation and prediction inside intelligence
- Use the help of hidden and shared models
 - Shared becomes the bridge and hidden becomes the separation points
- The modules becomes candidates for microservices
 - High Cohesion - Everything stays within context and modules are independant
 - Loose Coupling - Only what is needed is shared
- **Avoid** premature decomposition
 - Early decisions can be costly (eg: entire IoT as one module)
 - Re-decomposition may take time, effort and expenditure

The background of the slide features a central white diamond shape with a thin white border, set against a light gray background. The four corners of the slide are decorated with overlapping geometric shapes in yellow and blue. The top-left and bottom-right corners have yellow shapes, while the top-right and bottom-left corners have blue shapes. The text is centered within the white diamond.

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 !!!

The slide features a central white diamond shape on a light gray background. The diamond is outlined by a white border. In the four corners of the slide, there are overlapping geometric shapes in yellow and blue. The top-left and bottom-right corners have yellow shapes, while the top-right and bottom-left corners have blue shapes. The text "Microservices Communication" is centered within the white diamond.

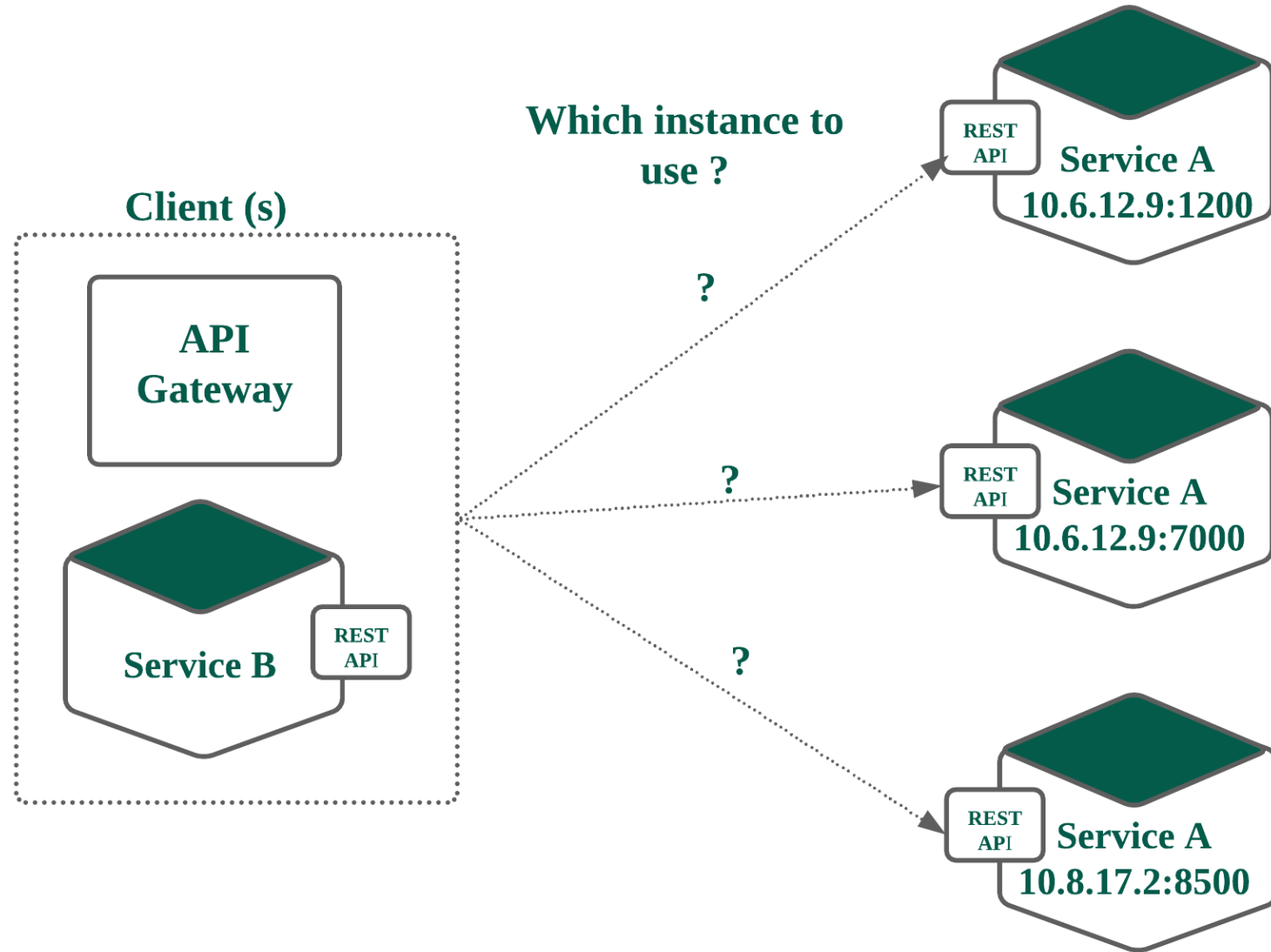
Microservices Communication

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

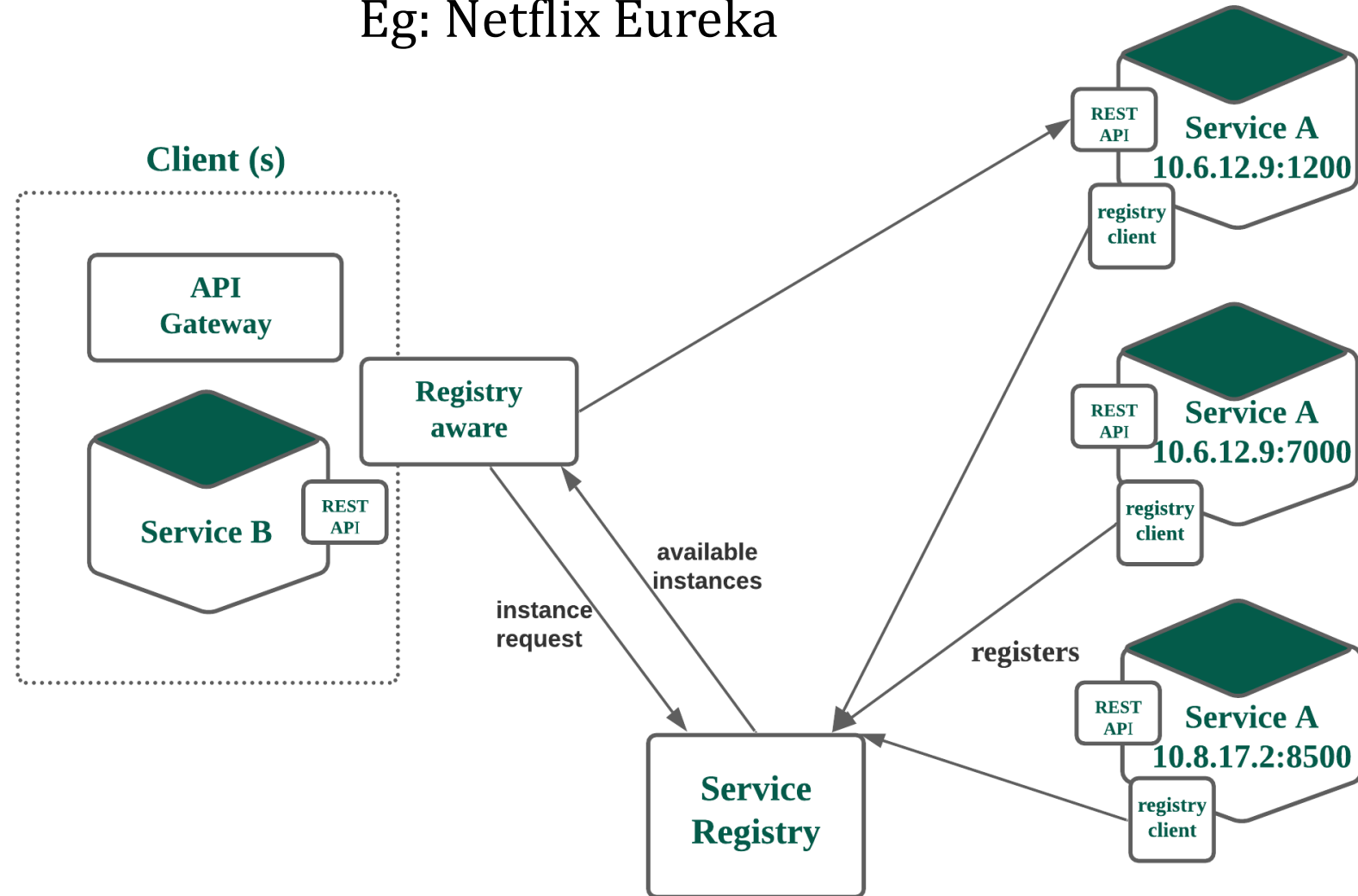


Use Service Registry!!

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

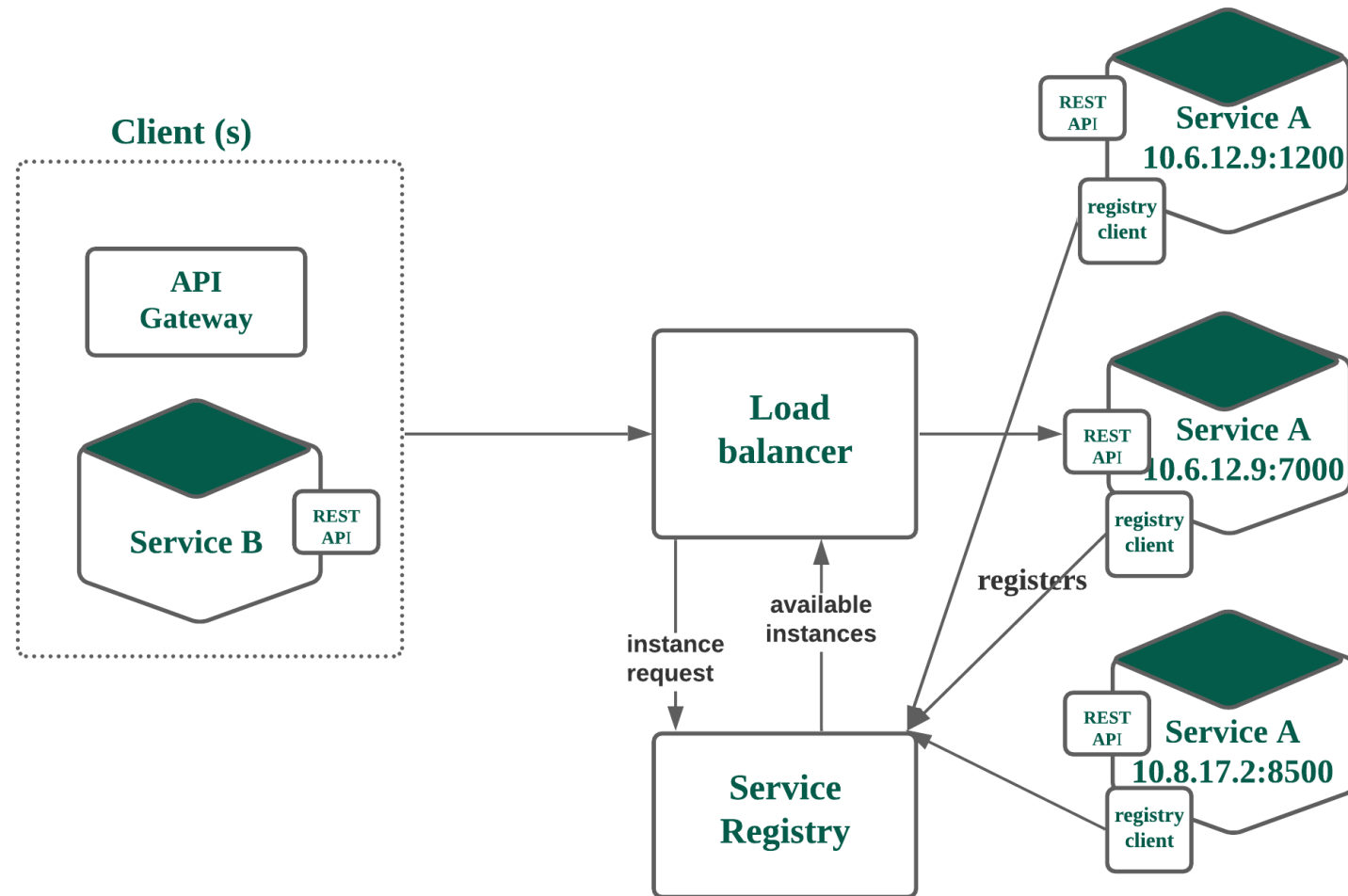
Eg: Netflix Eureka



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 !!!!

Thank You



Course website: karthikv1392.github.io/cs6401_se

Email: karthik.vaidhyanathan@iiit.ac.in

Web: <https://karthikvaidhyanathan.com>

Twitter: @karthi_ishere

