Architectural Styles & Patterns

CS6.401 Software Engineering

Dr. Karthik Vaidhyanthan

karthik.vaidhyanathan@iiit.ac.in

https://karthikvaidhyanathan.com

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY HYDERABAD

Software Engineering Research Centre

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. Software Architecture in Practice, Len Bass, 2nd, 3rd edition
2. Various sources from the web that has been duly credited in the respective slide

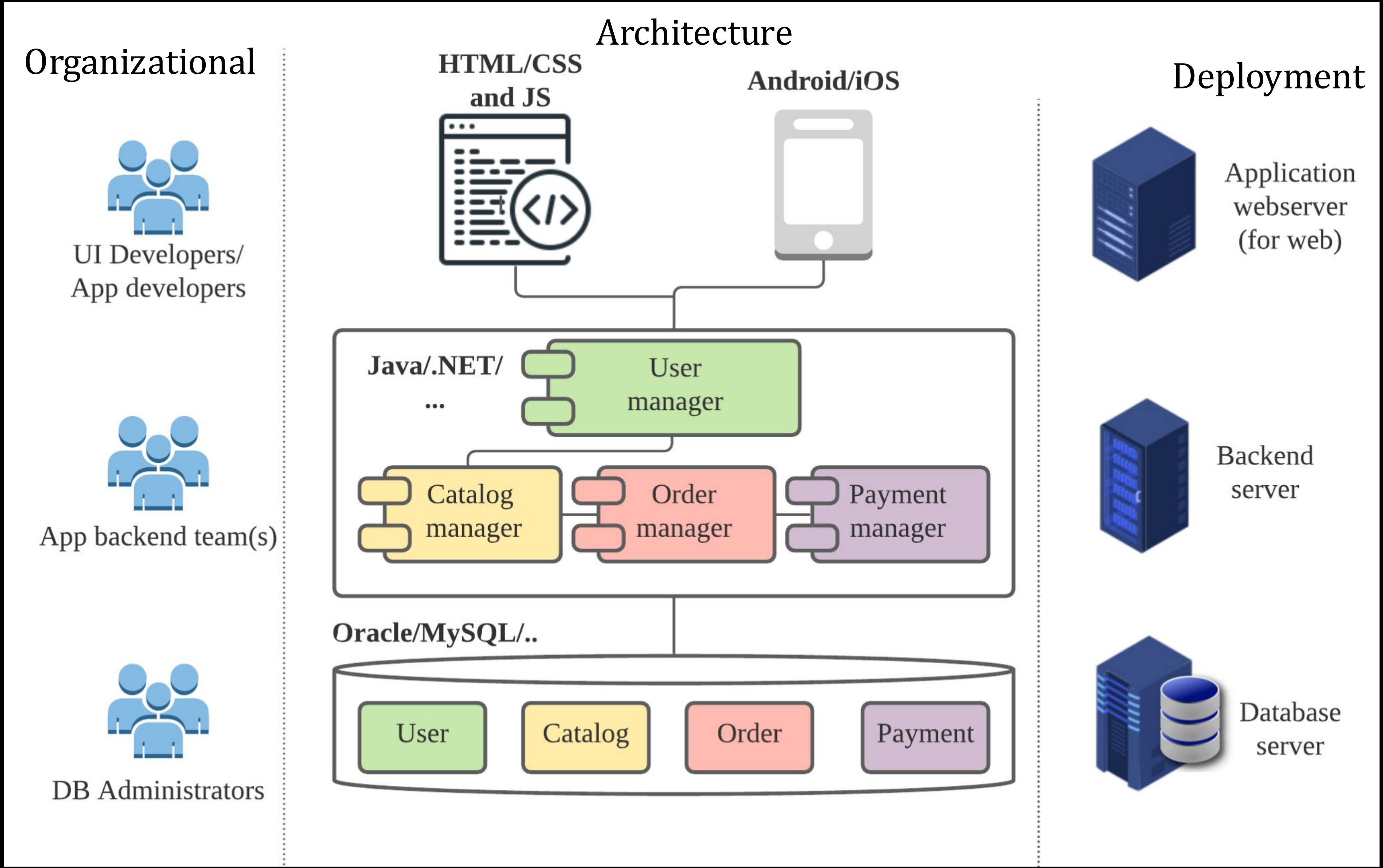
Monolith?

Monolith of Utah, USA

Menhir (monolith), France

Ponce Monolith, Bolivia

Monolithic Approach to E-commerce



Monolithic Approach – What are some pitfalls?

- High degree of coupling - everyone needs to know everything !!!
- Change cycle and bug fix can take weeks - Modifiability and time to market
- Adding new feature can be challenging - Extensibility
- Separation of concerns via components with inherent coupling - Modularity
- Scaling system implies scaling the whole stack - Scalability
- Limited by the language of choice - eg: add recommendation feature to e-commerce (Java or Python ?)
- Database is centralized - addition or modification is a costly process

Monolith has its own advantages too!

Service-Oriented

The Service-Oriented Pattern

Context

A number of services offered by service providers and consumed by service consumers. Service consumer should be able to use services without knowing detailed implementation

Problem

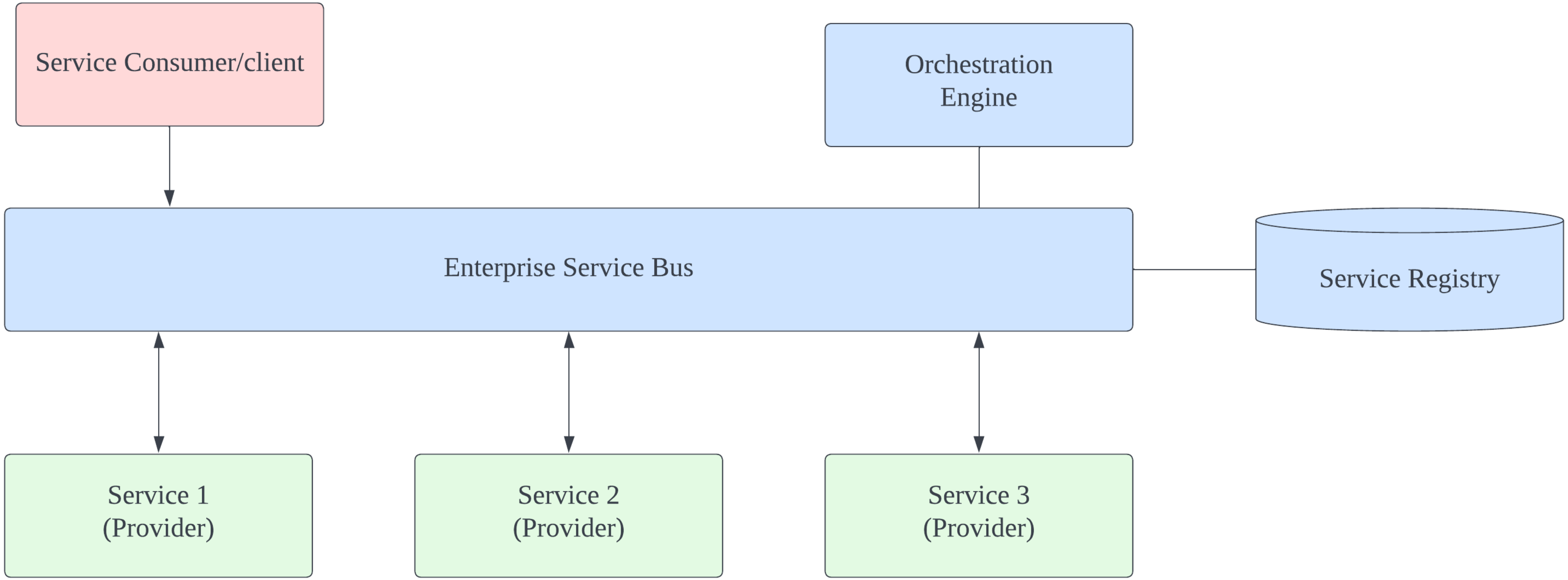
How to provide support for interoperability among different components running in different platforms implemented in different languages

availability, performance, security

Collection of loosely coupled services with clearly defined interfaces. Can be implemented in different languages. Supports commmunication between and to/from services

Solution

The Service Oriented Pattern



SOA Pattern - Architectural Elements (Components)

1. Service Providers: Components that provide 1 or more services through defined interfaces

2. Service Consumers: Invoke services directly or through intermediary

3. ESB: Intermediary component that can route and transform messages

4. Service Registry: Providers can register services, consumers can discvoer services

5. Orchestration Server: Coordinates interaction between consumers and providers based on languages

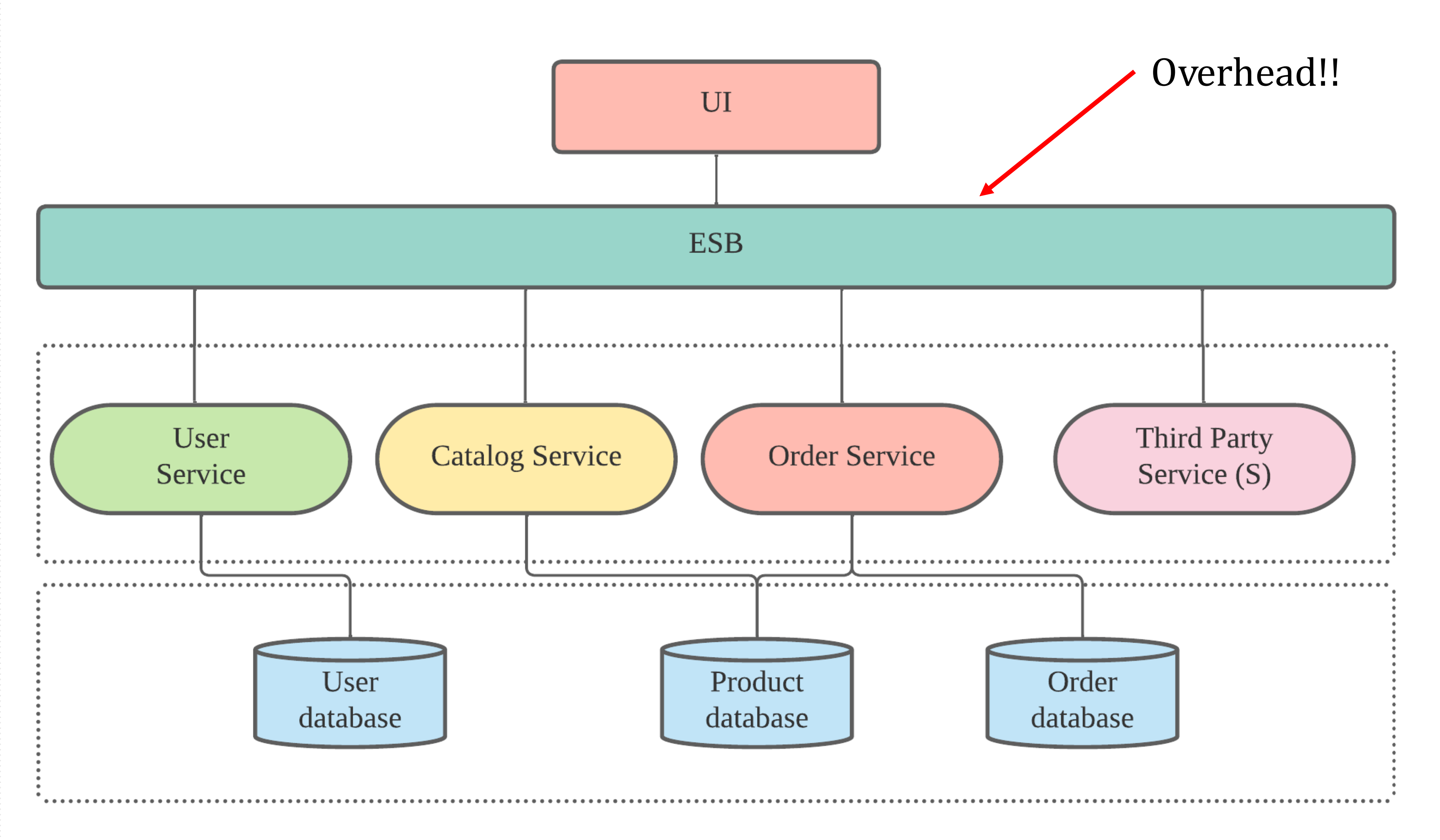
SOA Pattern - Architectural Elements (Connectors)

1. SOAP Connector: SOAP Protocol for synchronous communication over HTTP

2. REST Connector: Relies on request/response operations over HTTP

3. Asynchronous messaging connector: For point-to-point asynchronous message exchanges or pub-sub exchanges

SOA Applied to E-Commerce



SOA Pattern

Relations

Attachments of different components to available connectors

Constraints

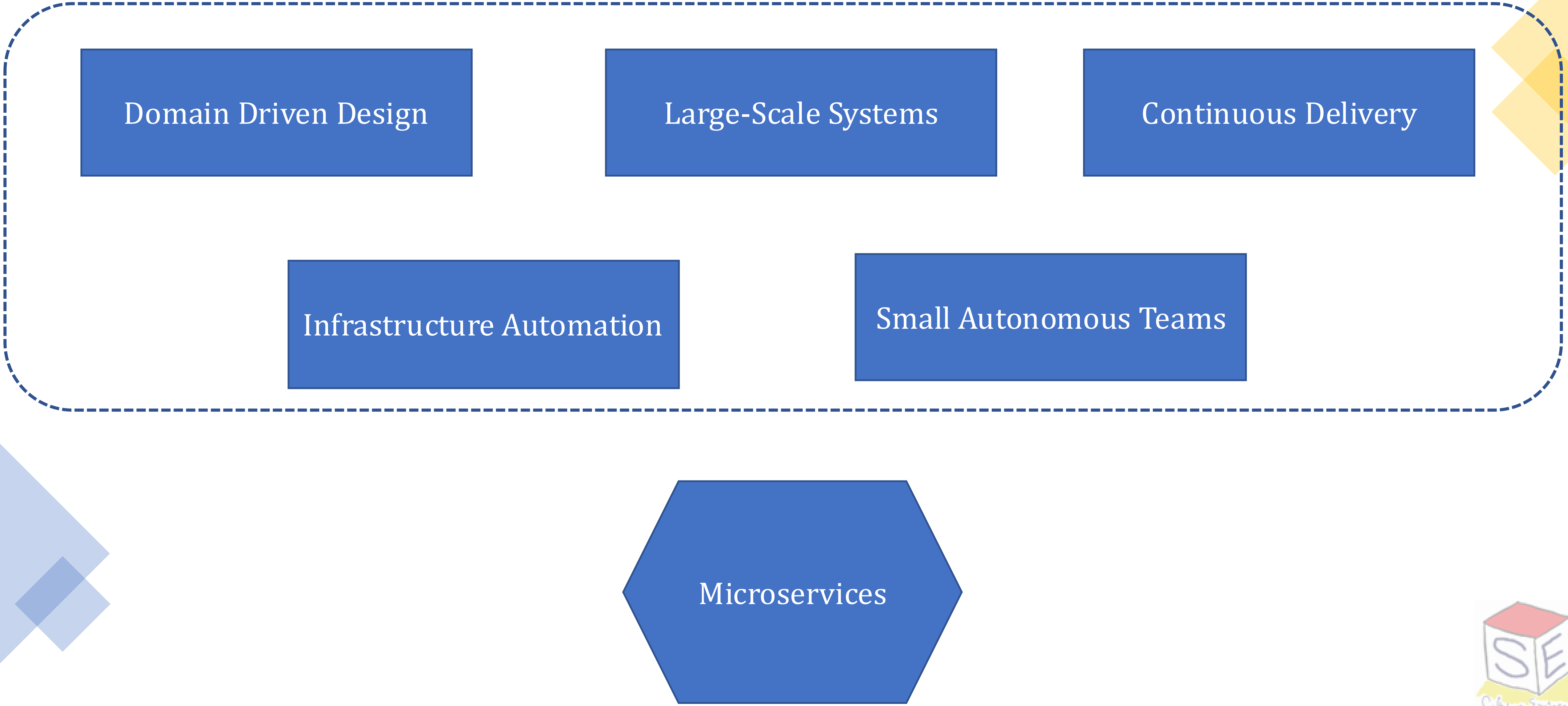
1. Service consumers are connected to providers (ESBs or other intermediary component may be used)

Weakness

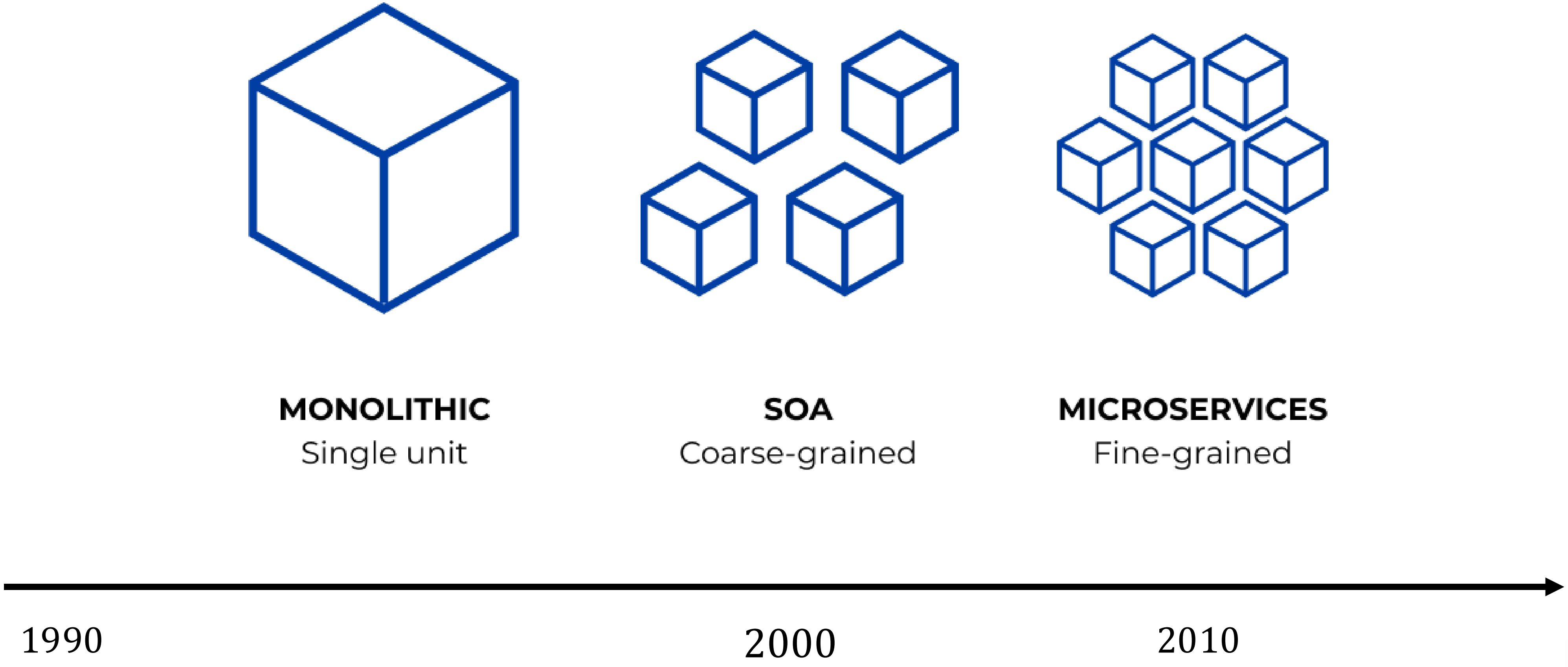
1. Complex to build
2. Performance bottlenecks due to middleware
3. Performance gurantees are usually not met

Time to Evolve: Microservices

SOA Pattern - Architectural Elements (Connectors)



Moving Towards Microservices



MONOLITHIC
Single unit

SOA
Coarse-grained

MICROSERVICES
Fine-grained

1990

2000

2010

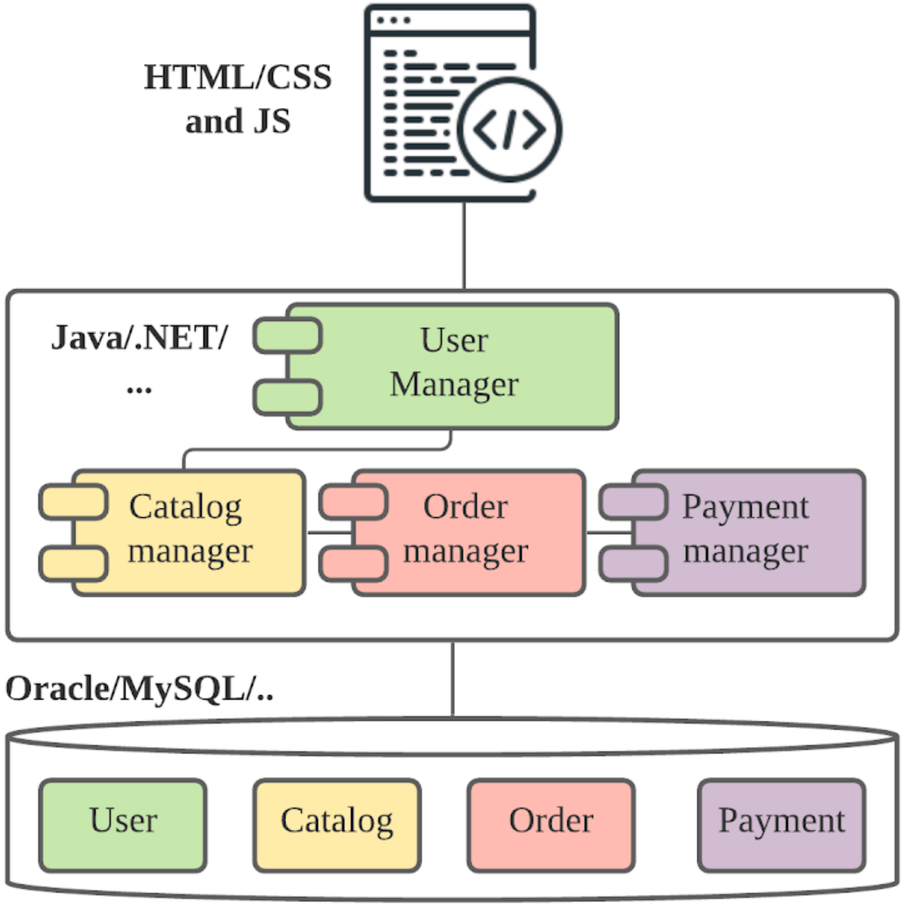
Microservices: What does it Mean?

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

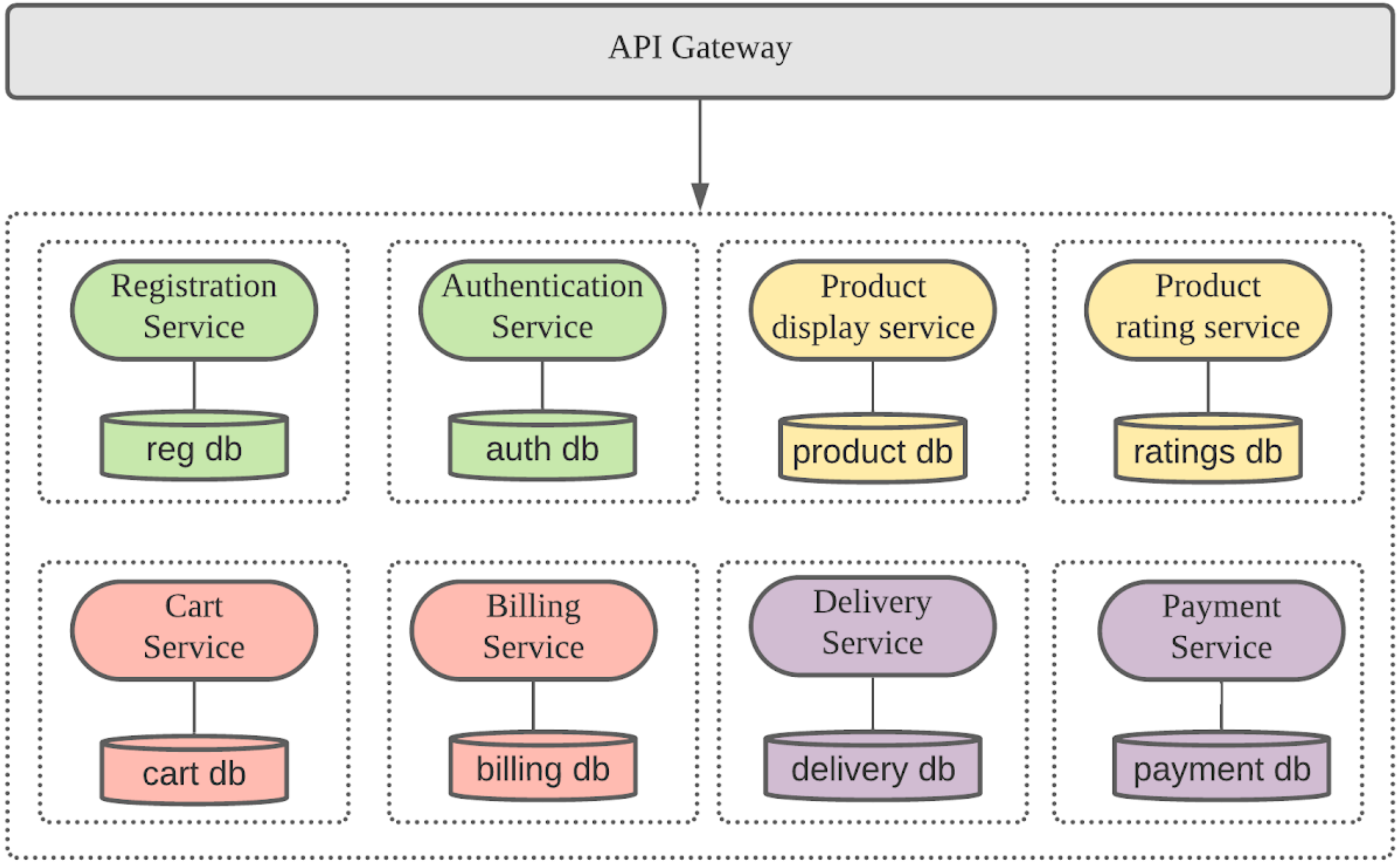
“It is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API” -- Martin Fowler

Microservices: What does it Mean?

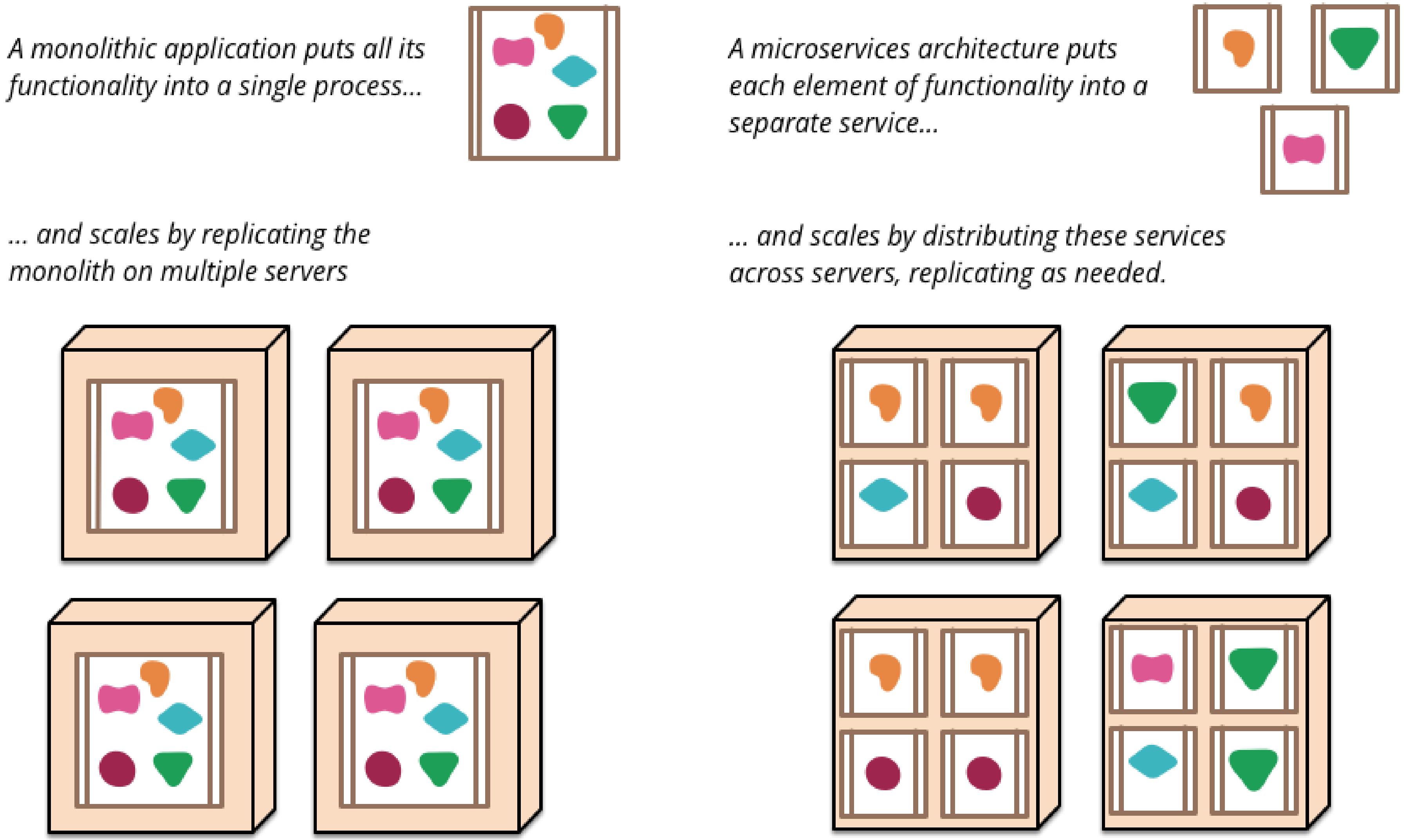
Monolithic Version



Microservices Version



Microservices: What does it Mean?



A monolithic application puts all its functionality into a single process...

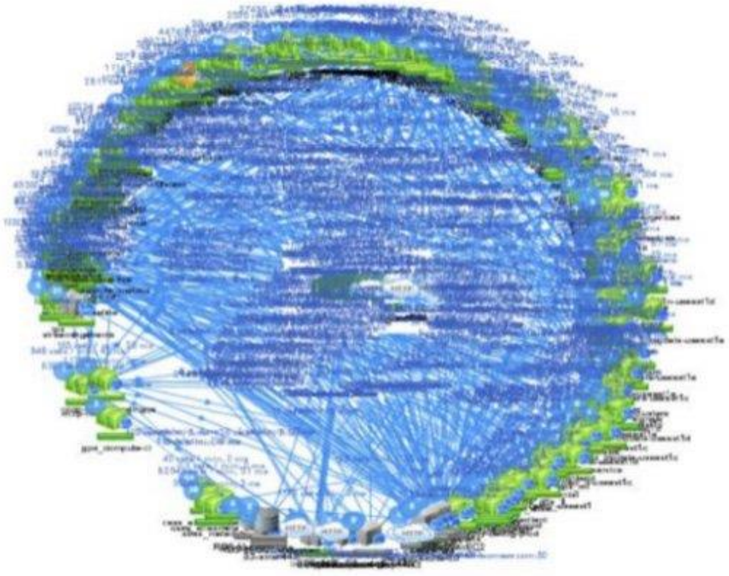
... and scales by replicating the monolith on multiple servers

A microservices architecture puts each element of functionality into a separate service...

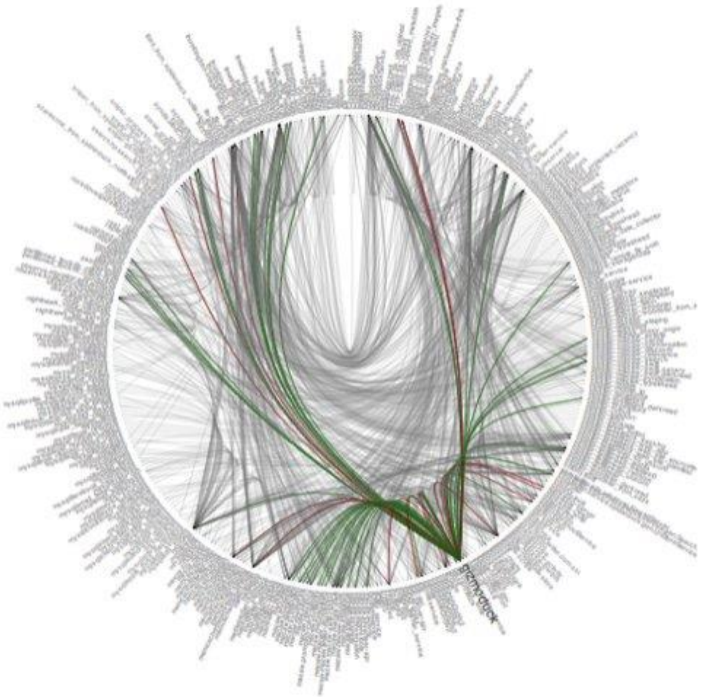
... and scales by distributing these services across servers, replicating as needed.

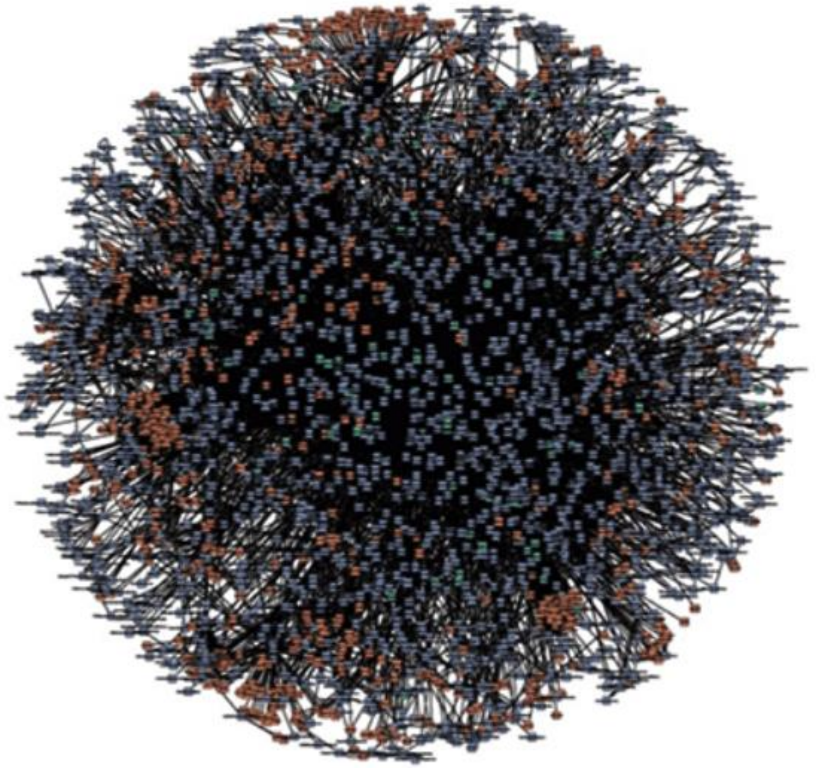
Source: https://martinfowler.com/articles/microservices.html

Microservices: Who Uses Them?



NETFLIX



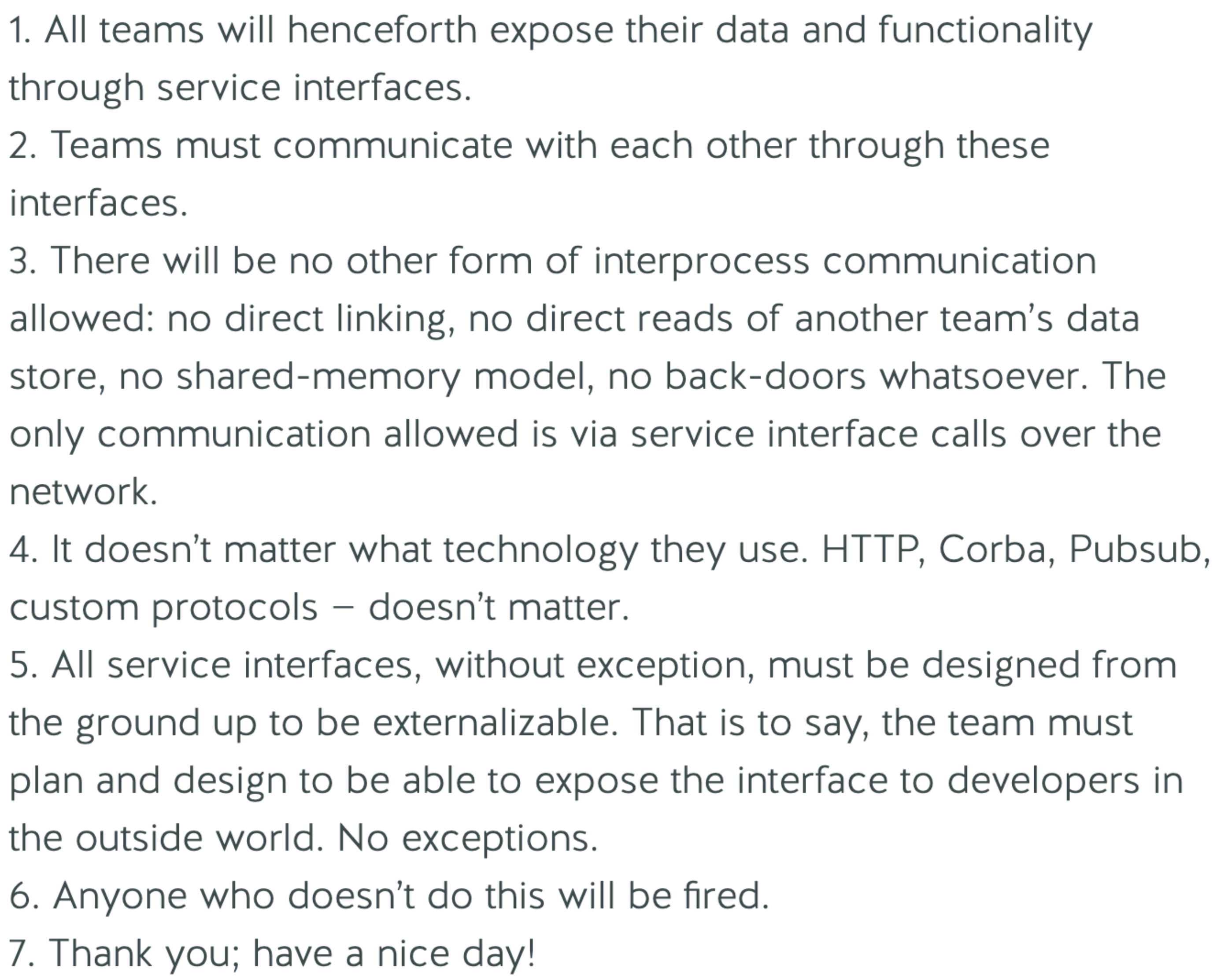


amazon.com

Source: http://tinyurl.com/3kswbtak, Google images, Twitter

Amazon’s API Mandate

Jeff Bezos,
Founder and President, Amazon



1. All teams will henceforth expose their data and functionality through service interfaces.
2. Teams must communicate with each other through these interfaces.
3. There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team’s data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
4. It doesn’t matter what technology they use. HTTP, Corba, Pubsub, custom protocols – doesn’t matter.
5. All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
6. Anyone who doesn’t do this will be fired.
7. Thank you; have a nice day!

Letter from Jeff Bezos in 2002: https://nordicapis.com/the-bezos-api-mandate-amazons-manifesto-for-externalization/

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!!!

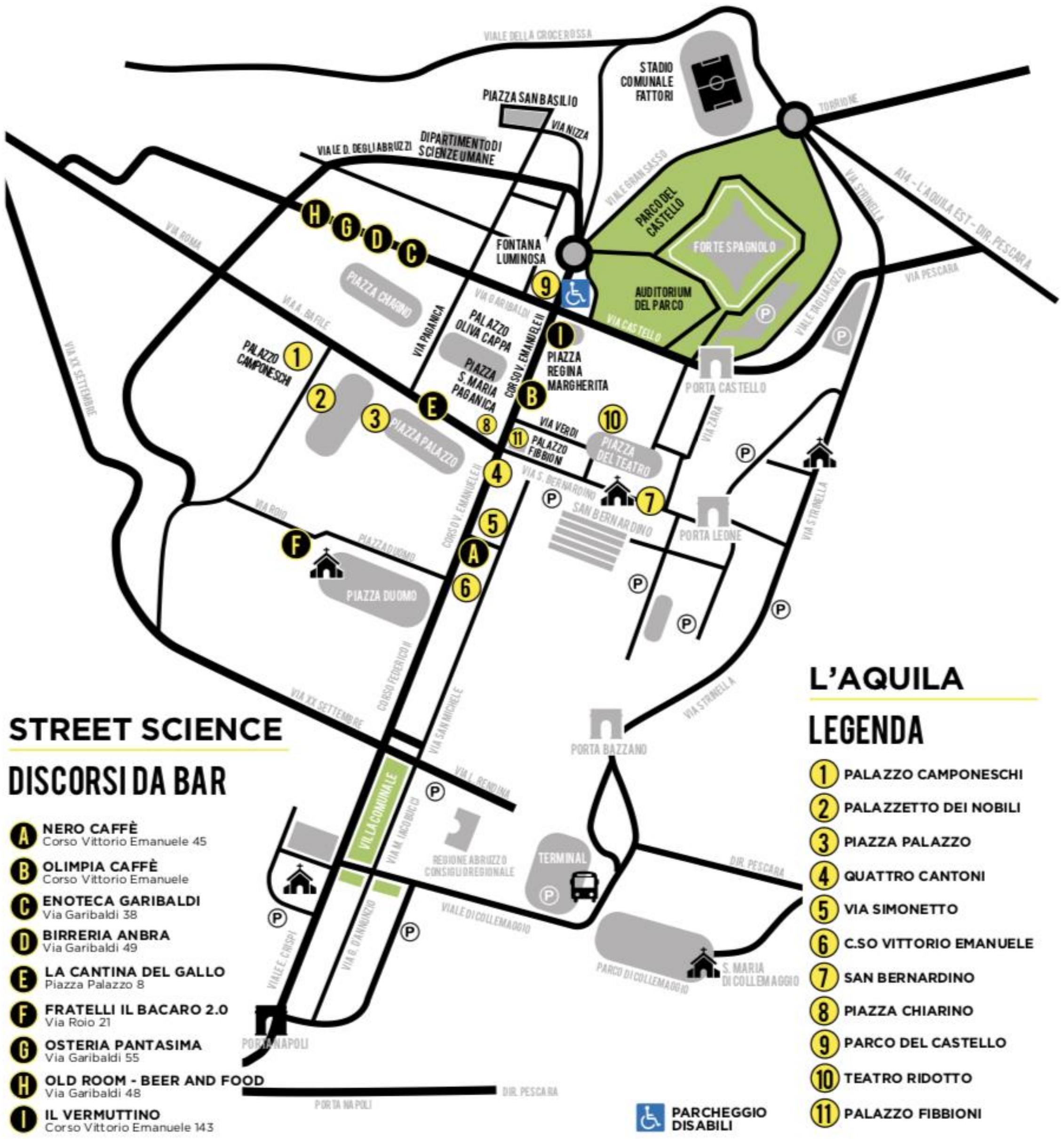
How to identify Microservices?

Main Takeaways

- Architectural Pattern serves as guidelines
- Always be aware of trade-offs
- A complex system can consists of multiple architectura
- Think about an IoT system, e-commerce system or any production system

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

NdR Case Study



STREET SCIENCE

DISCORSI DA BAR

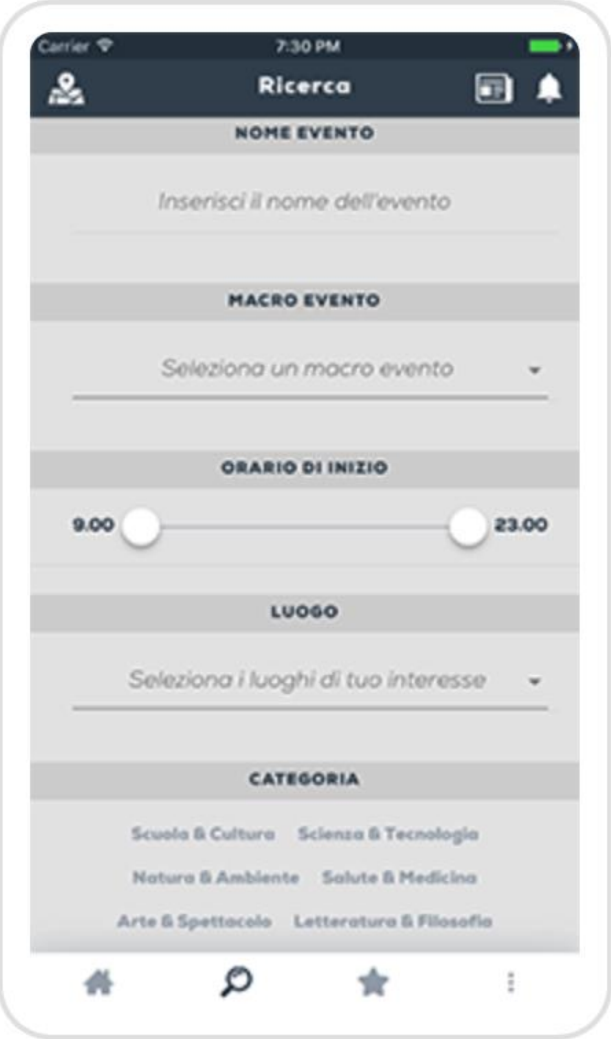
- A NERO CAFFÈ
  Corso Vittorio Emanuele 45
- B OLIMPIA CAFFÈ
  Corso Vittorio Emanuele
- C ENOTECA GARIBALDI
  Via Garibaldi 38
- D BIRRERIA ANBRA
  Via Garibaldi 49
- E LA CANTINA DEL GALLO
  Piazza Palazzo 8
- F FRATELLI IL BACARO 2.0
  Via Roio 21
- G OSTERIA PANTASIMA
  Via Garibaldi 55
- H OLD ROOM - BEER AND FOOD
  Via Garibaldi 48
- I IL VERMUTTINO
  Corso Vittorio Emanuele 143

L'AQUILA

LEGENDA

1. PALAZZO CAMPONESCHI
2. PALAZZETTO DEI NOBILI
3. PIAZZA PALAZZO
4. QUATTRO CANTONI
5. VIA SIMONETTO
6. C.SO VITTORIO EMANUELE
7. SAN BERNARDINO
8. PIAZZA CHIARINO
9. PARCO DEL CASTELLO
10. TEATRO RIDOTTO
11. PALAZZO FIBBIONI

PARCHEGGIO DISABILI



NOME EVENTO

Inserisci il nome dell'evento

MACRO EVENTO

Seleziona un macro evento

ORARIO DI INIZIO

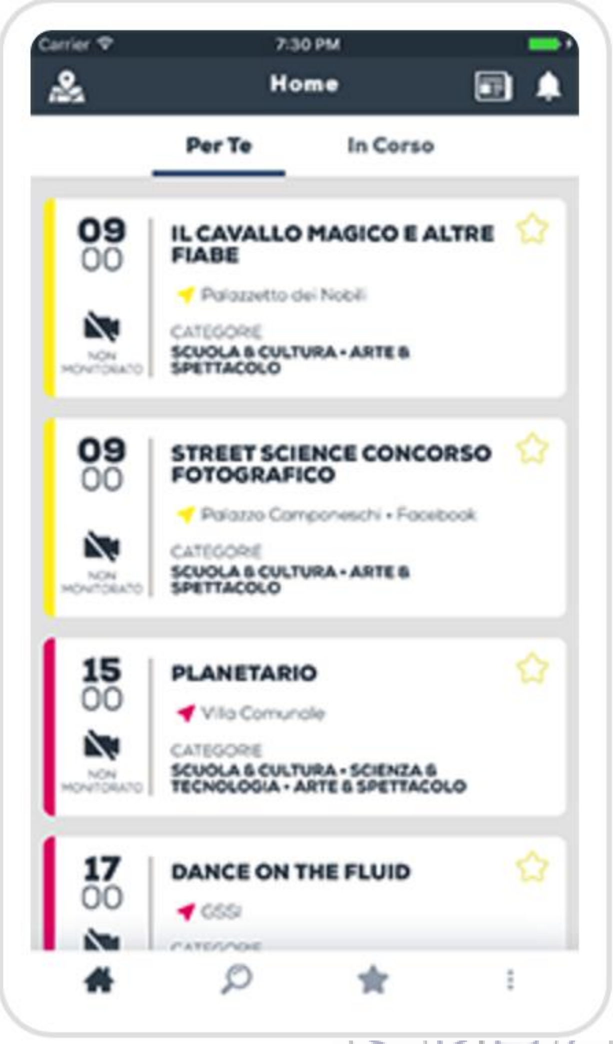
9.00 23.00

LUOGO

Seleziona i luoghi di tuo interesse

CATEGORIA

Scuola & Cultura · Scienza & Tecnologia · Natura & Ambiente · Salute & Medicina · Arte & Spettacolo · Letteratura & Filosofia



09 00 — IL CAVALLO MAGICO E ALTRE FIABE
Palazzetto dei Nobili
NON MONITORATO
CATEGORIE
SCUOLA & CULTURA • ARTE & SPETTACOLO

09 00 — STREET SCIENCE CONCORSO FOTOGRAFICO
Palazzo Camponeschi • Facebook
NON MONITORATO
CATEGORIE
SCUOLA & CULTURA • ARTE & SPETTACOLO

15 00 — PLANETARIO
Villa Comunale
NON MONITORATO
CATEGORIE
SCUOLA & CULTURA • SCIENZA & TECNOLOGIA • ARTE & SPETTACOLO

17 00 — DANCE ON THE FLUID
GSSI

https://www.streetscience.it

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

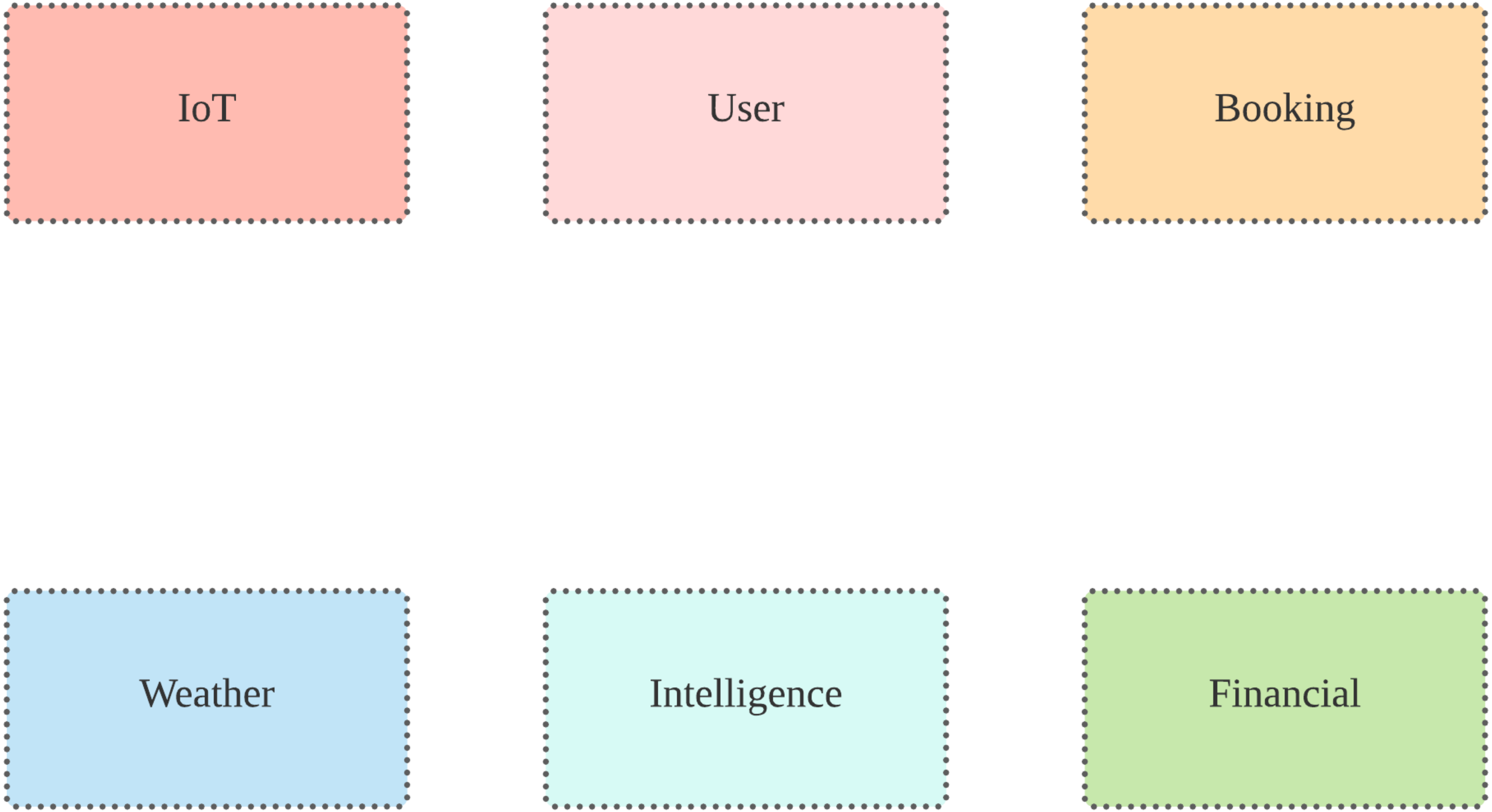
- Minimize coupling between microservices
- Should be easy to change and deploy one without affecting others
- Each microservice needs to know as little as possible about others

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

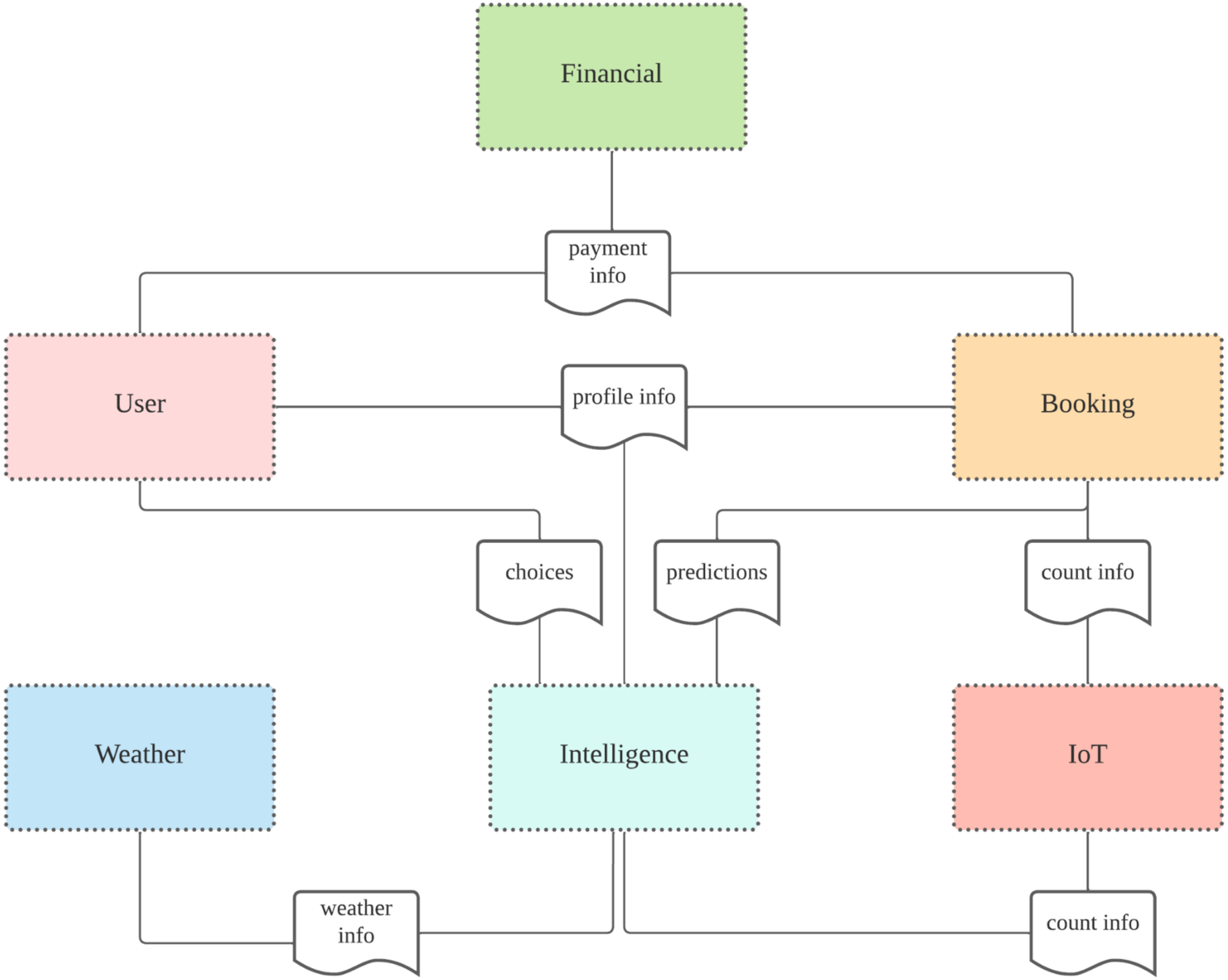
Weather

Intelligence

Financial

Hidden and Shared Models

Hidden and Shared Models



Financial

payment info

User

profile info

Booking

choices

predictions

count info

Weather

Intelligence

IoT

weather info

count info

Shared and Hidden Models

- Identify what needs to be shared
  - Eg: Sharing of information on people and car count to booking context

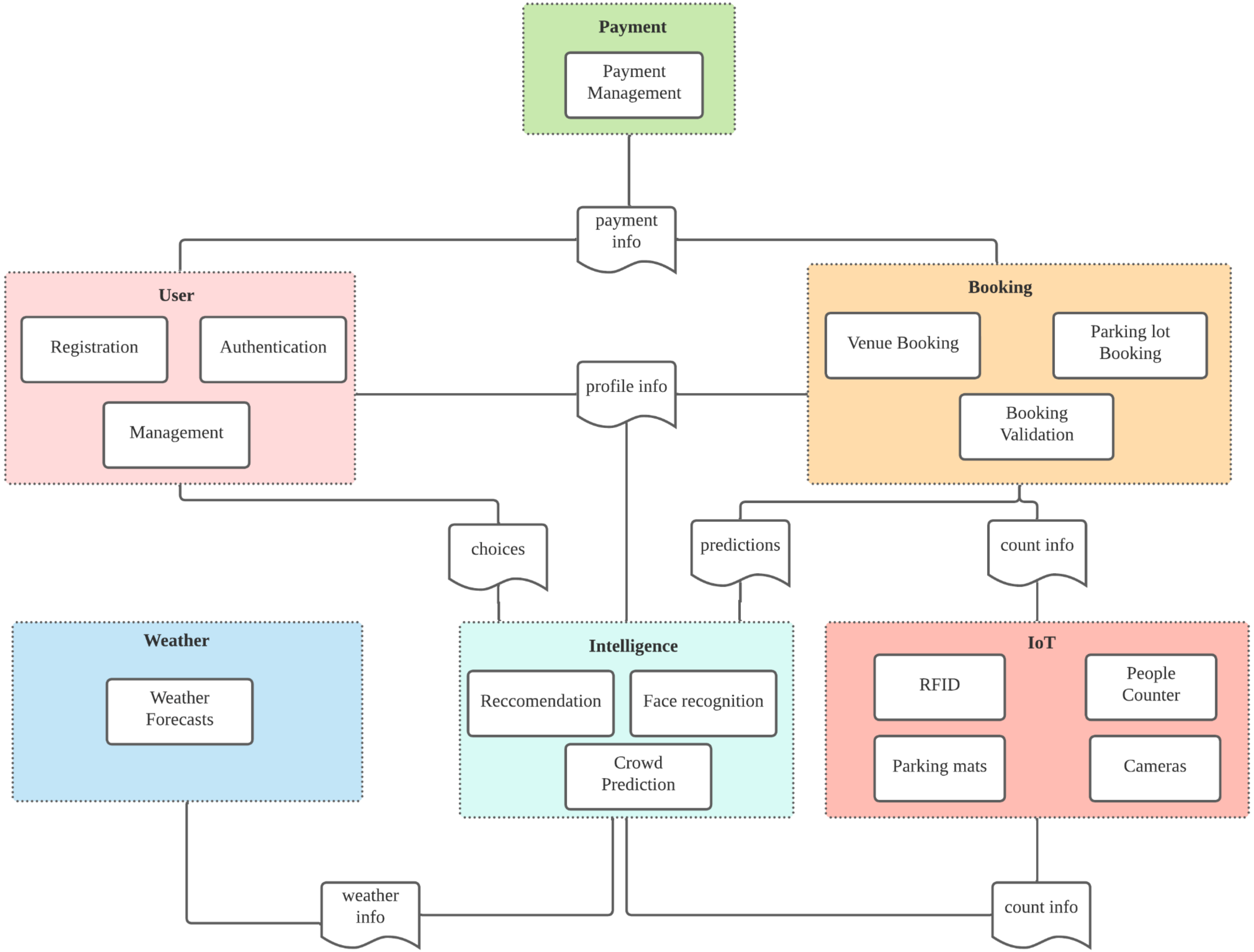
- Same things may have different meaning in different contexts
  - Eg: Sensor data in IoT context and booking context

- 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

- Seperate 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

Thank You

Course website: karthikv1392.github.io/cs6401_se

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

INTERNATIONAL INSTITUTE OF
INFORMATION TECHNOLOGY
HYDERABAD