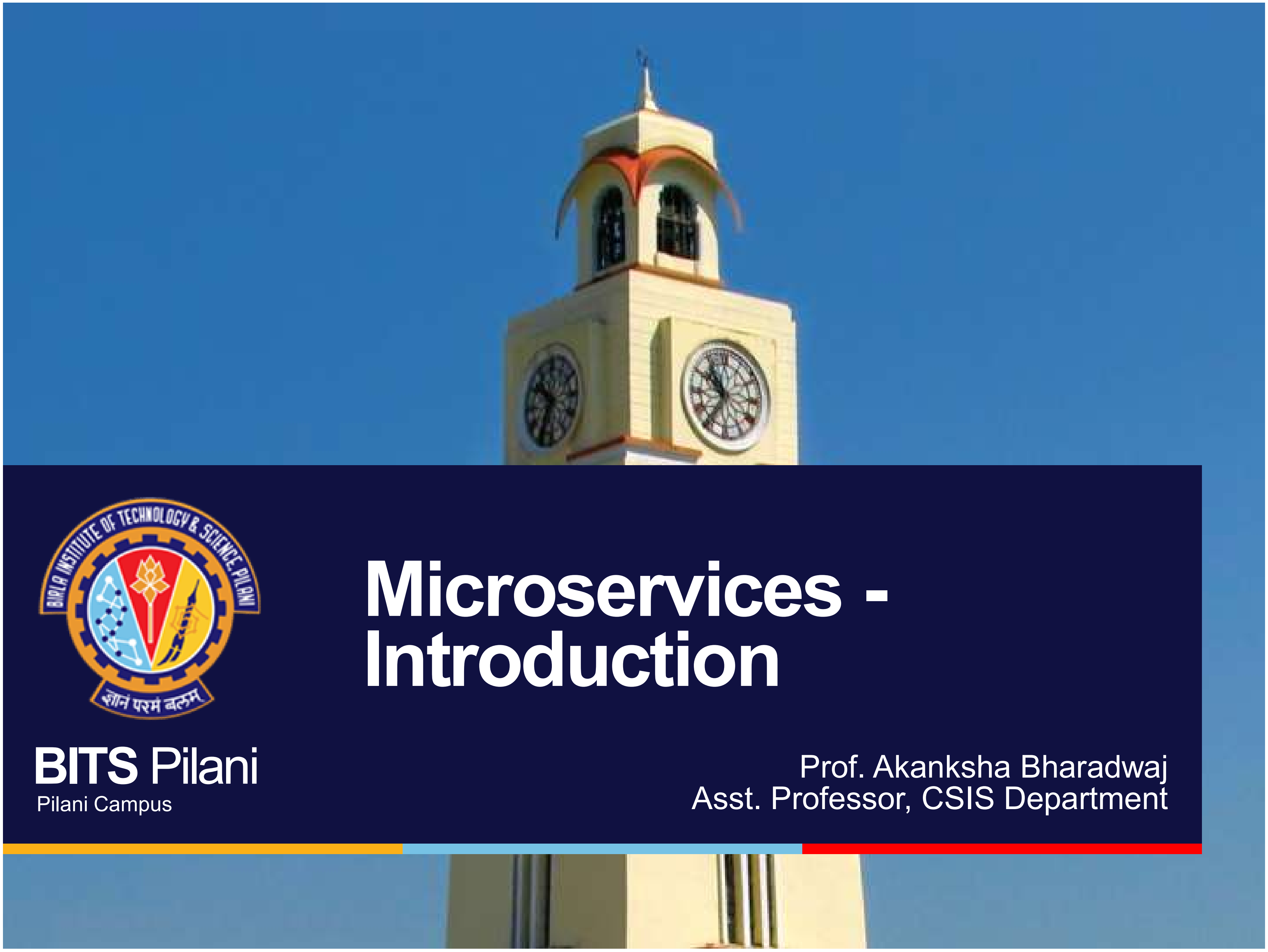


Microservices - Introduction

BITS Pilani
Pilani Campus

Prof. Akanksha Bharadwaj
Asst. Professor, CSIS Department

innovate achieve lead

BITS Pilani
Pilani Campus

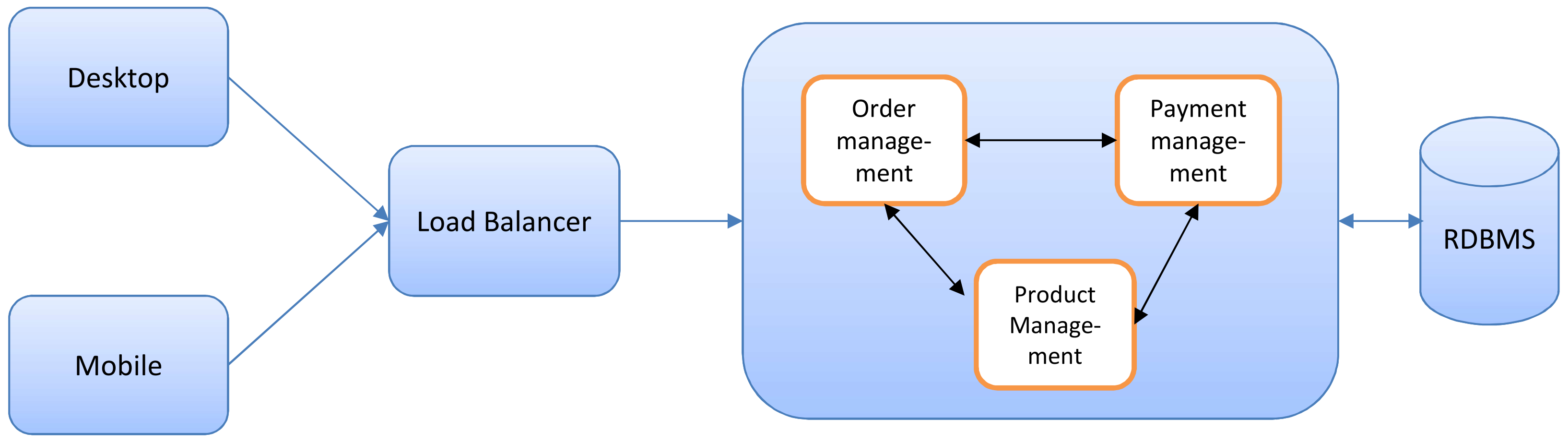
SE ZG583, Scalable Services Lecture No. 4

What is Monolithic Architecture?

- Monolith means composed all in one piece.
- They’re typically complex applications that encompass several tightly coupled functions.
- When all functionality in a system had to be deployed together, we consider it a **monolith**.

Example Architecture

Online shopping



Advantages of Monolithic

- Simplicity
- Easier Performance Optimization
- Easy Data Management

Disadvantages of Monolith

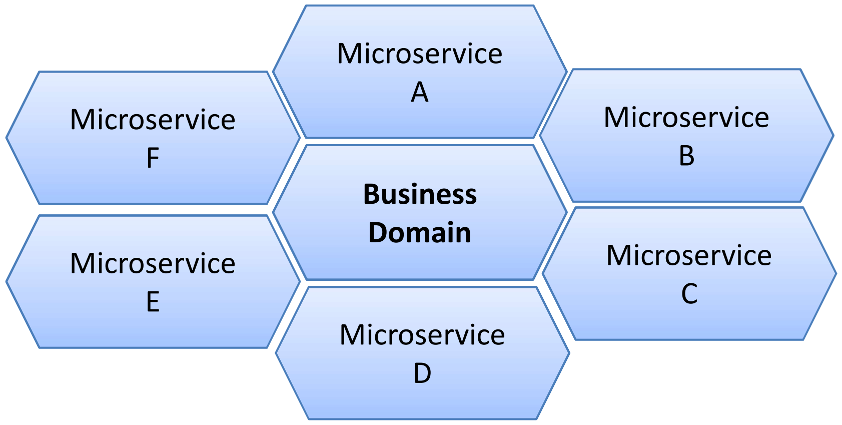
- Scalability Challenges
- Limited Technology Flexibility
- Development Bottlenecks
- Slow & Risky Updates
- Harder Maintenance in Large Systems

Need for Microservices

- Why is there a need to convert a fully functional monolithic application to Microservices ?
- Is the conversion worth the pain and effort?
- Should I be converting all my applications to Microservices?

What is Microservices?

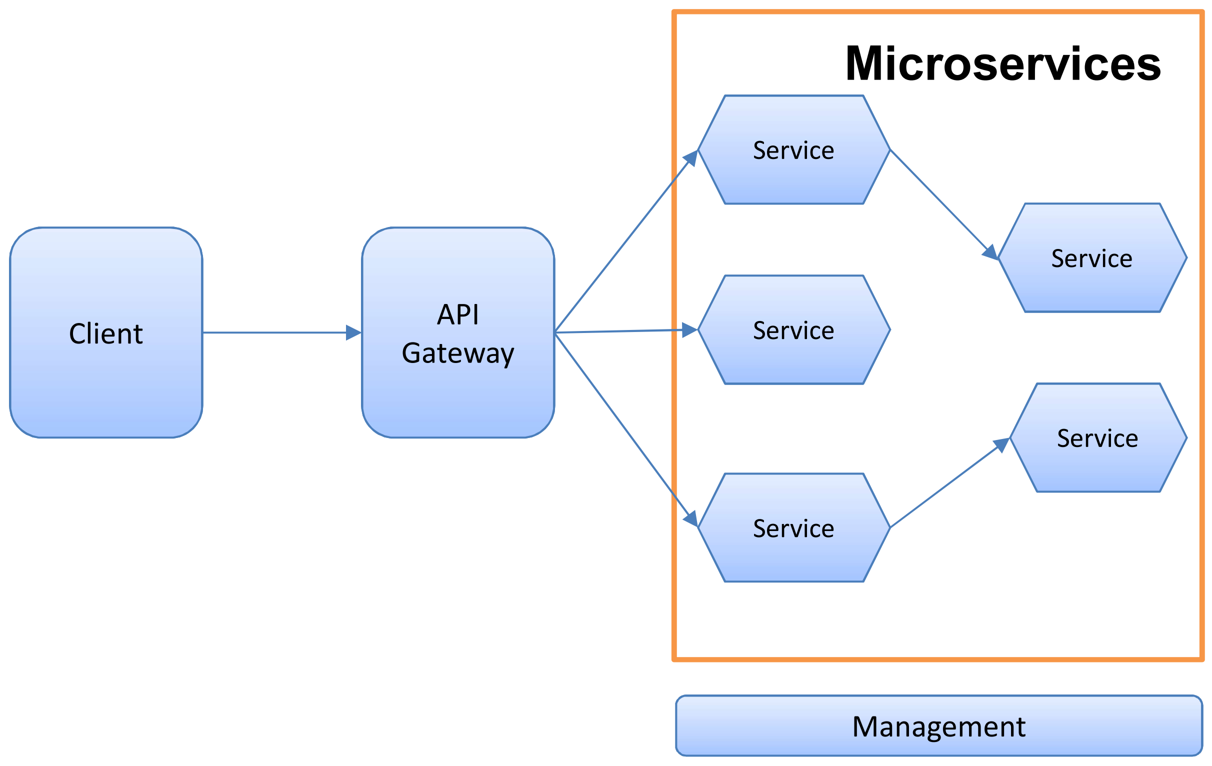
- Microservices are independently deployable services modeled around a business domain.
- They communicate with each other via networks,
- Each microservice can focus on a single business capability



Main characteristics of microservices

- Independent Deployment
- Decentralized Data Management
- Scalability
- Technology Diversity
- Fault Isolation
- Lightweight Communication

Example Architecture



SOA

- SOA, or service-oriented architecture, defines a way to make software components reusable via service interfaces.
- These interfaces utilize common communication standards in such a way that they can be rapidly incorporated into new applications without having to perform deep integration each time.

SOA Vs Microservices

Aspect	SOA (Service-Oriented Architecture)	Microservices
Granularity	Services are **larger, coarse-grained**; often handle multiple related functions.	Services are **smaller, fine-grained**; focused on a single business capability.
Communication	Typically uses **Enterprise Service Bus (ESB)** or other centralized middleware; often **SOAP** or XML-based.	Uses **lightweight protocols** like REST, gRPC, or messaging queues; JSON or binary formats.
Data Storage	Often a **shared database** across services.	Each service **owns its own database**; decentralized data management.
Data Duplication	Less duplication due to shared DB, but can lead to tight coupling.	Possible duplication because each service has its own DB, but improves independence.
Deployment	Services are often **deployed together** in larger applications.	Fully **independent deployment** per service.
Technology Stack	Usually uses a **uniform technology stack** across services.	Each service can use **different tech stacks** (polyglot).
Scalability	Scales at the service level, but larger service units make fine-grained scaling harder.	**Fine-grained scaling** possible — scale only the service that needs it.
Governance	Centralized governance, strong emphasis on **standards and contracts**.	Decentralized governance, teams decide standards per service.
Performance	Can have **higher latency** due to ESB mediation.	Lower latency — direct service-to-service communication.
Fault Isolation	Failures in one service can impact others due to tight coupling.	Better fault isolation — failure in one service is less likely to bring down others.
Best Fit	Large enterprise systems needing **integration of many existing applications**.	Cloud-native, agile environments needing **rapid, independent delivery**.

BITS Pilani

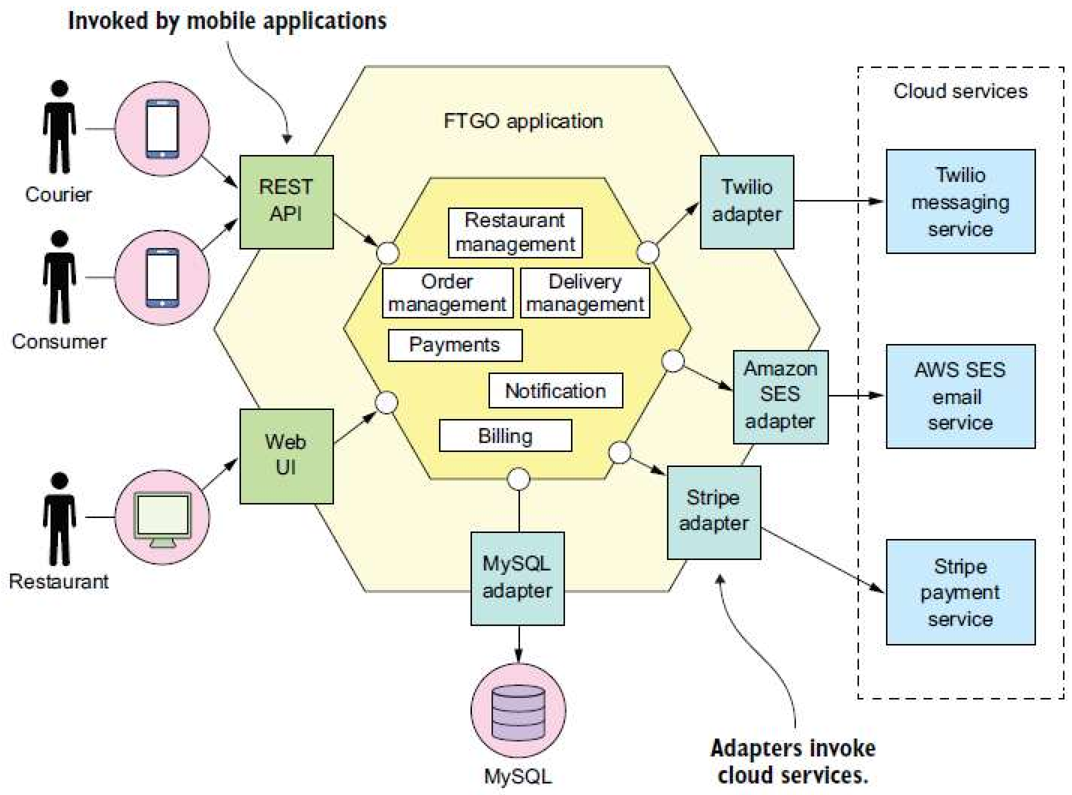
Pilani Campus

Case Study

FTGO Case Study

- Since its launch in late 2005, Food to Go, Inc. (FTGO) had grown by leaps and bounds. Today, it’s one of the leading online food delivery companies in the United States.
- The business even plans to expand overseas, although those plans are in jeopardy because of delays in implementing the necessary features.
- At its core, the FTGO application is quite simple. Consumers use the FTGO website or mobile application to place food orders at local restaurants.
- FTGO coordinates a network of couriers who deliver the orders. It’s also responsible for paying couriers and restaurants. Restaurants use the FTGO website to edit their menus and manage orders.
- The application uses various web services, including Stripe for payments, Twilio for messaging, and Amazon Simple Email Service (SES) for email.

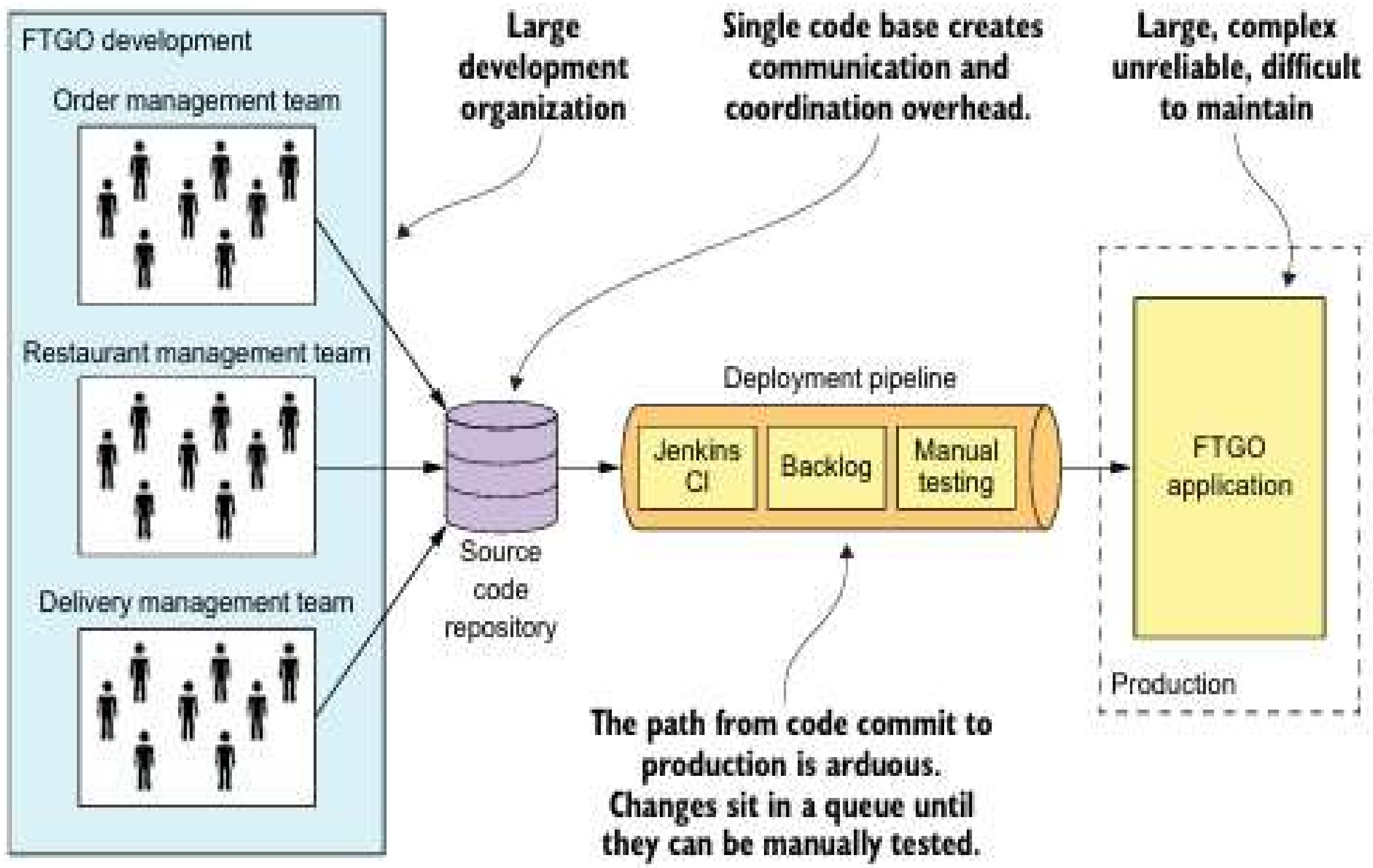
Old Architecture of the FTGO application



Problems faced in old FTGO architecture

- Complexity
- Slow development and deployment cycle
- Scaling is difficult
- Delivering a reliable monolith is challenging

Monolithic Hell



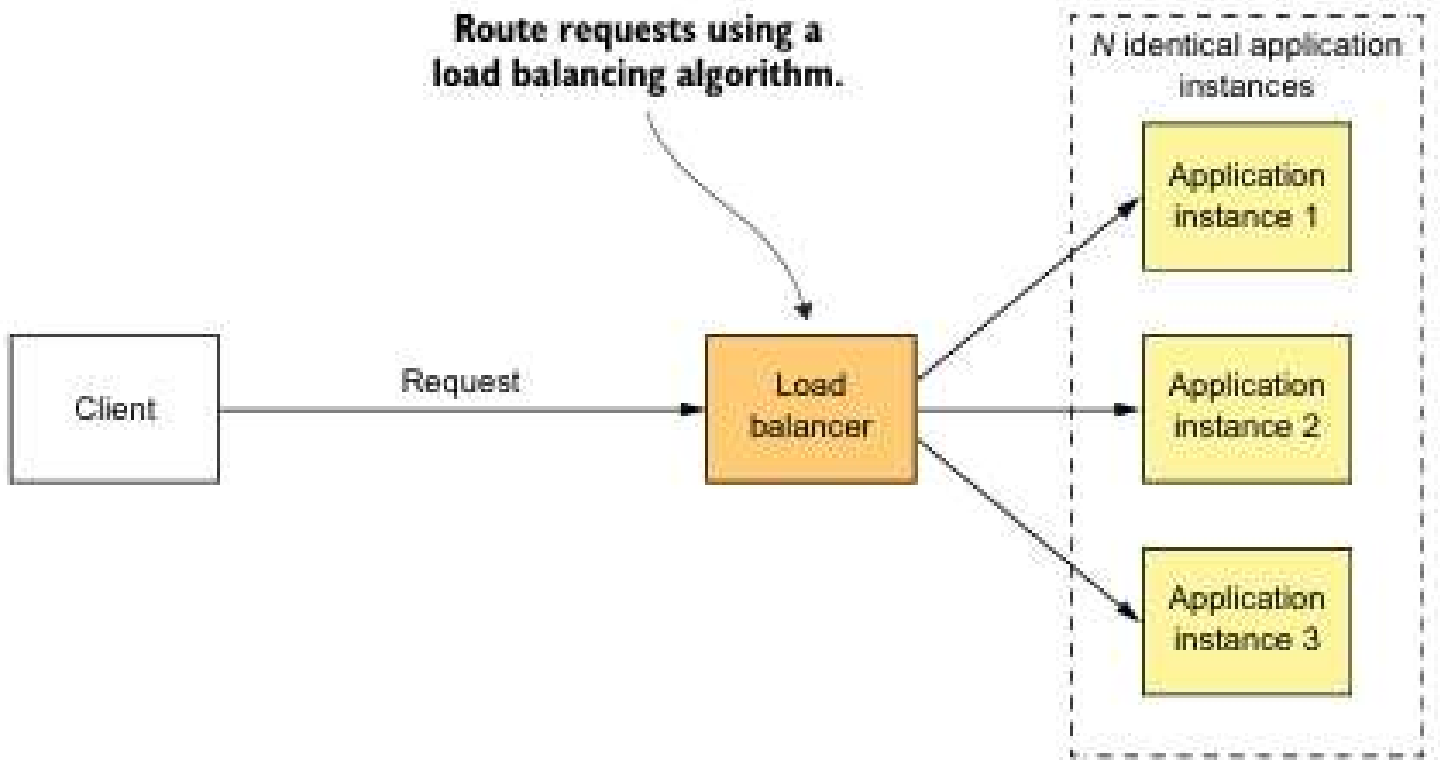
FTGO development
Order management team
Restaurant management team
Delivery management team
Large development organization
Single code base creates communication and coordination overhead.
Large, complex unreliable, difficult to maintain
Deployment pipeline
Jenkins CI
Backlog
Manual testing
Source code repository
FTGO application
Production
The path from code commit to production is arduous. Changes sit in a queue until they can be manually tested.

innovate achieve lead

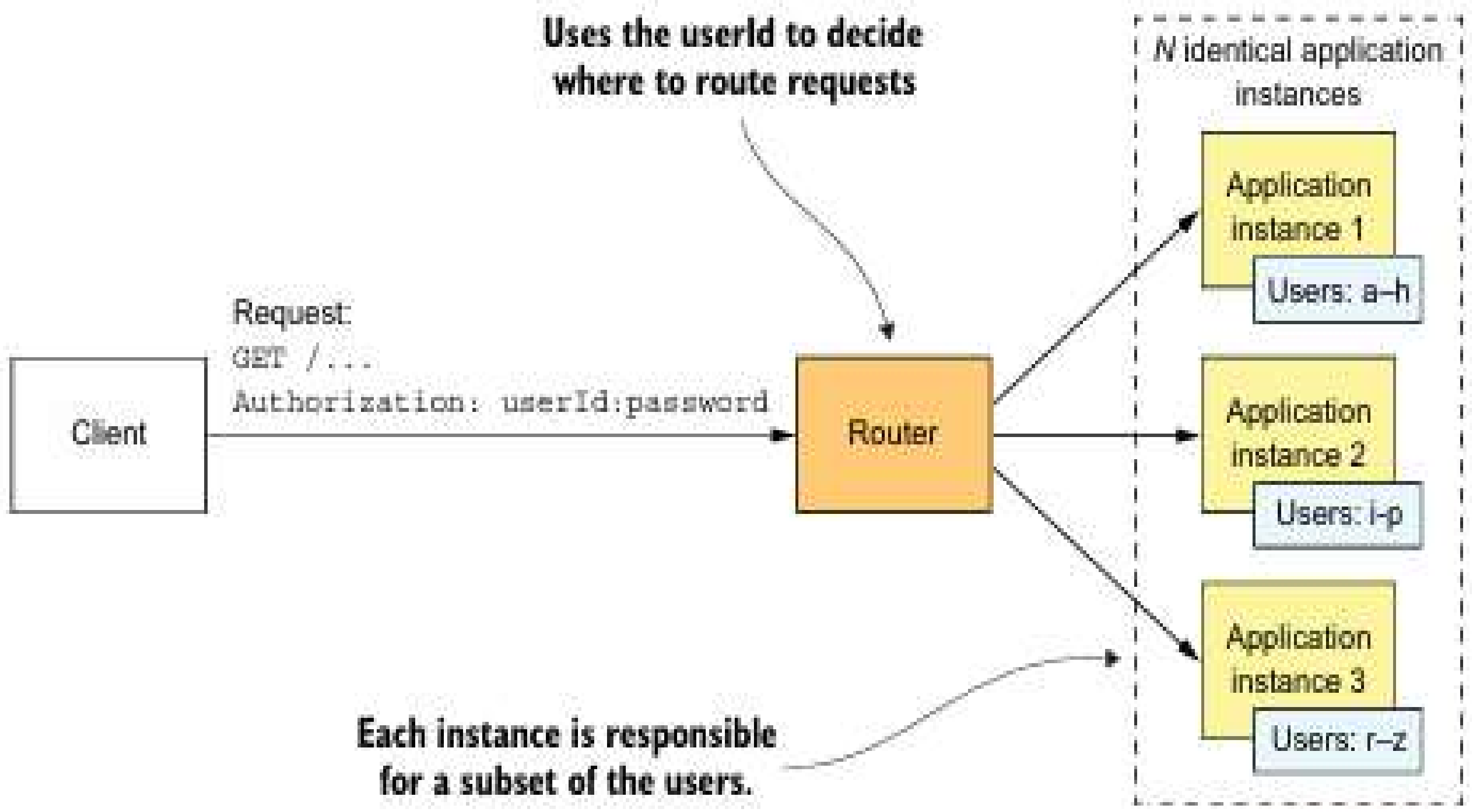
BITS Pilani
Pilani Campus

Possible solution

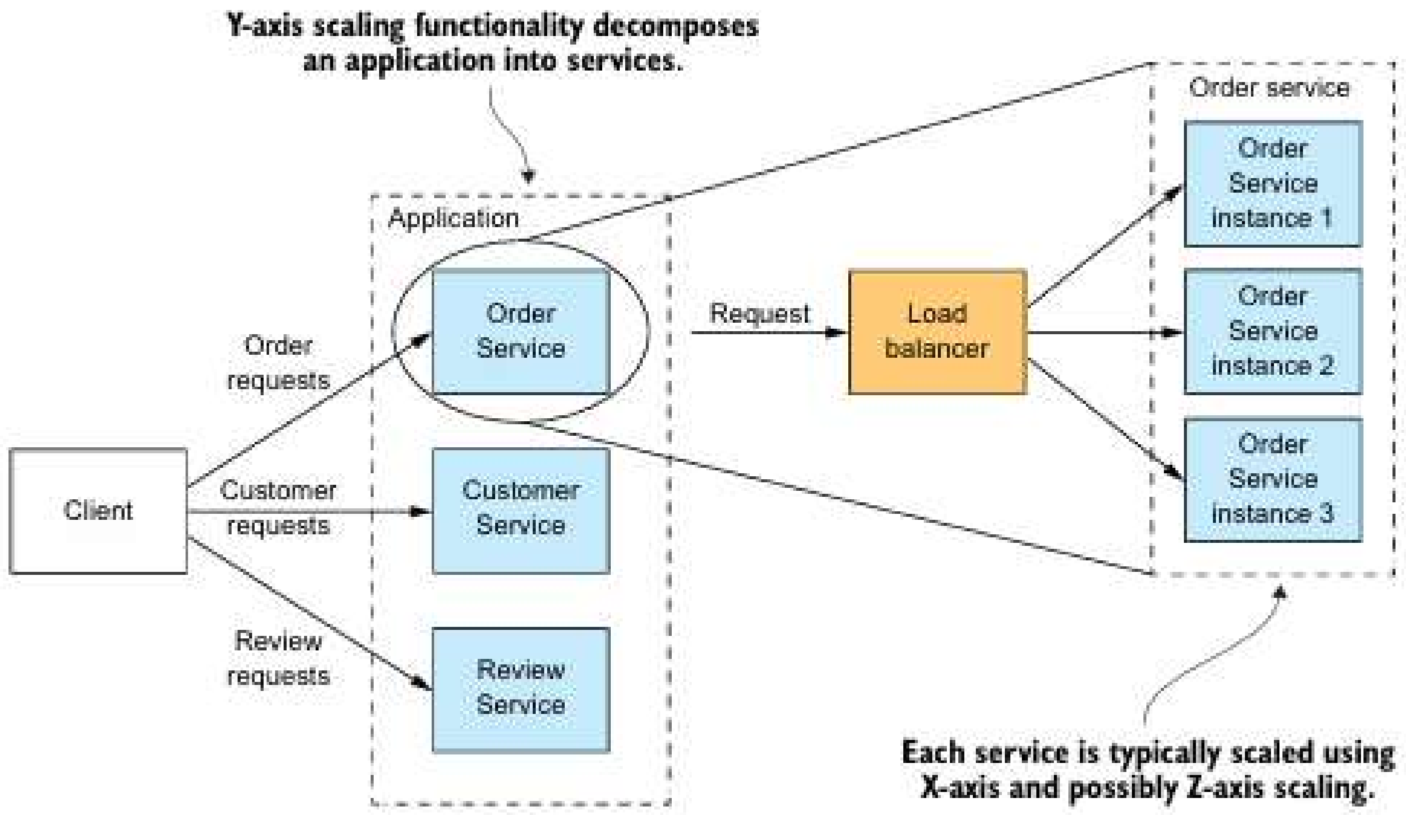
X- axis scaling



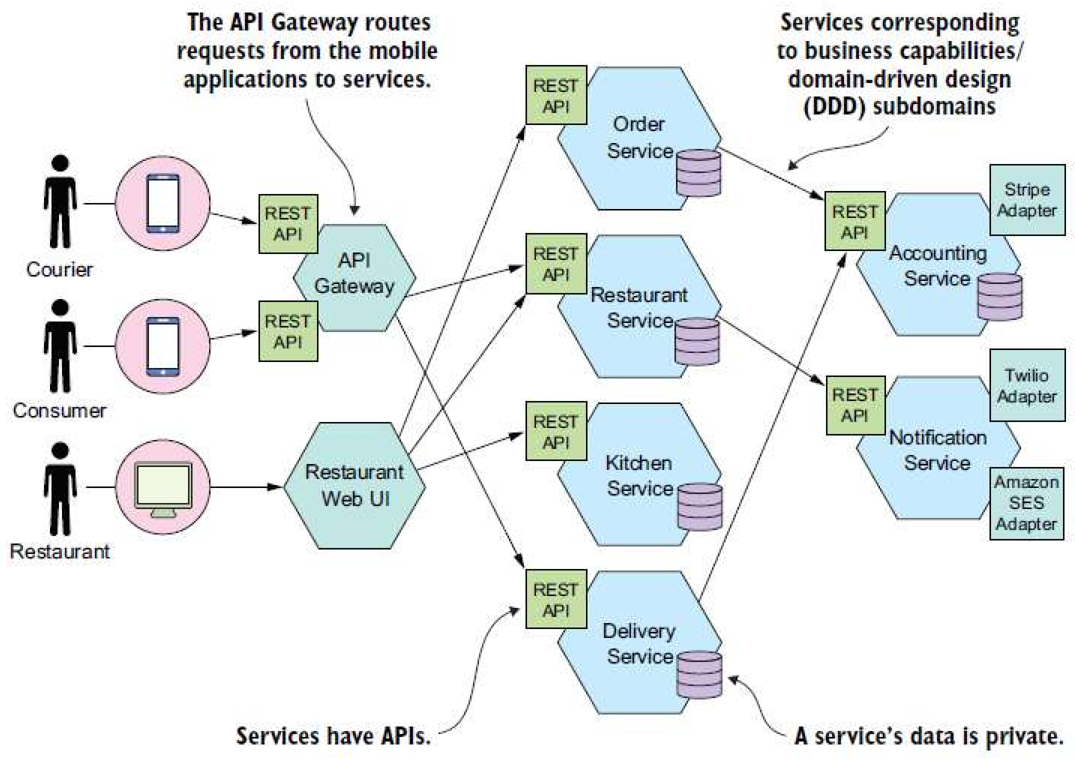
Z-axis scaling


Uses the userId to decide where to route requests
N identical application instances
Request:
GET /...
Authorization: userId:password
Client
Router
Application instance 1
Users: a–h
Application instance 2
Users: i–p
Application instance 3
Users: r–z
Each instance is responsible for a subset of the users.

Y-axis scaling



Microservices Architecture for FTGO

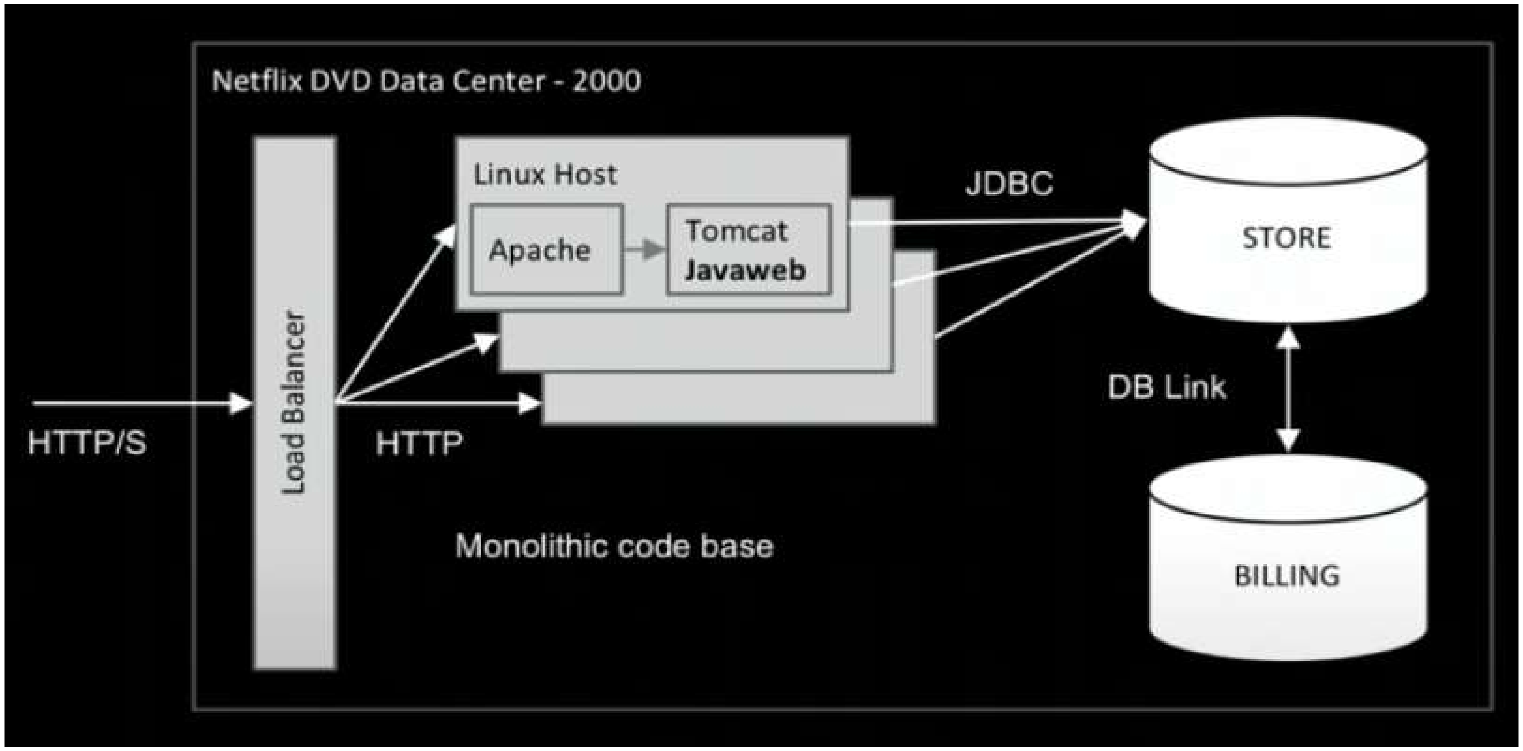


Netflix Case Study

- Netflix launched in 1998. At first they rented DVDs through the US Postal Service. But Netflix saw the future was on-demand streaming video
- In 2007 Netflix introduced their streaming video-on-demand service
- Why are we considering this case study?

How Netflix worked earlier?

Netflix Architecture earlier



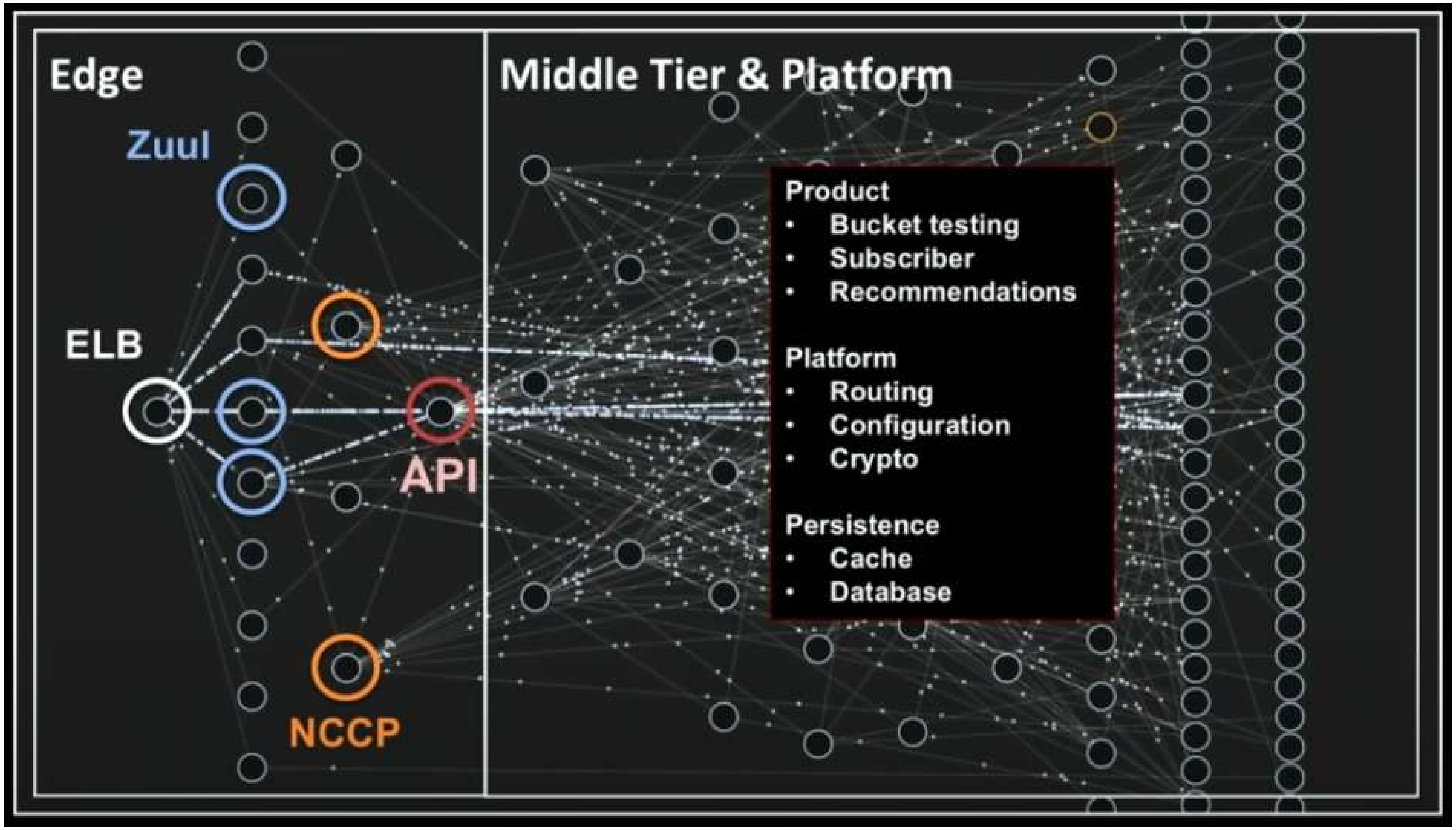
Challenges in previous architecture

- Monolithic Code base
- Monolithic Database
- Tightly coupled Architecture

What they need in new Architecture?

- Modularity and encapsulation
- Scalability
- Virtualization and Elasticity

Updated Netflix architecture



Product
- Bucket testing
- Subscriber
- Recommendations

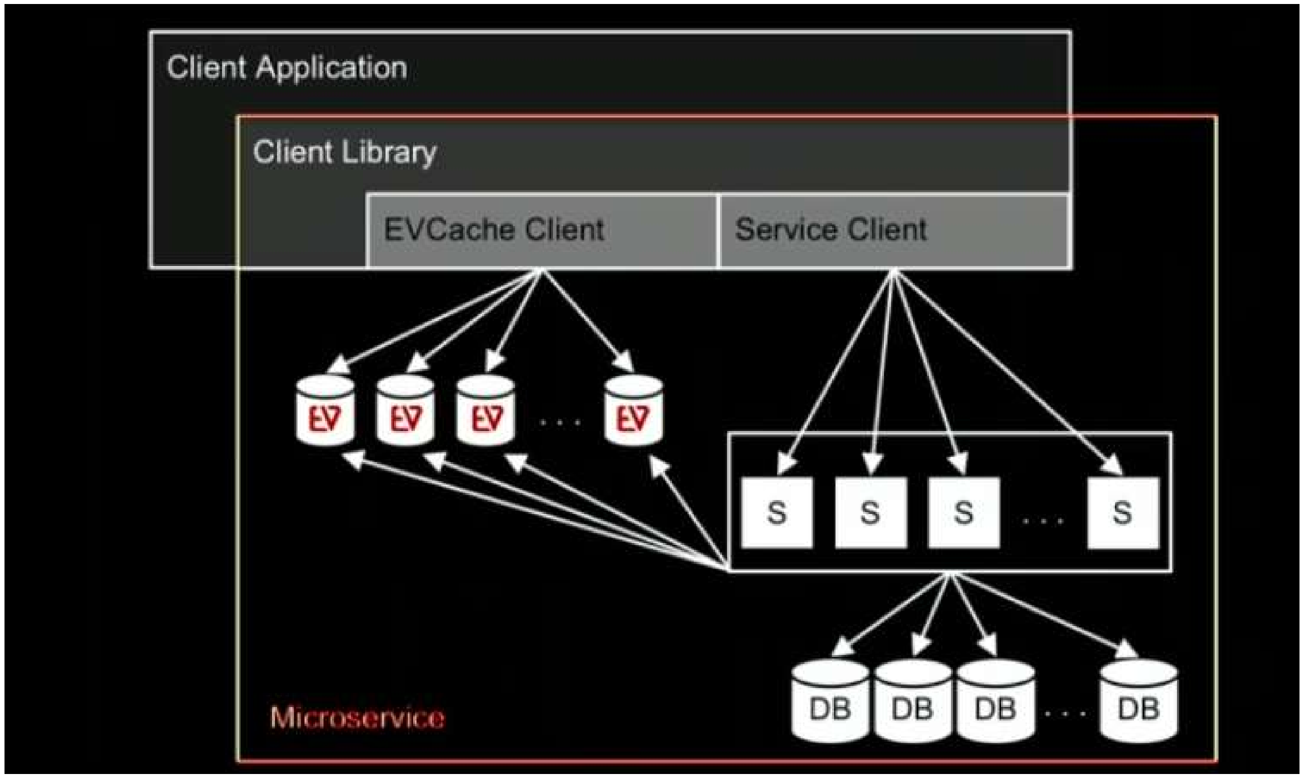
Platform
- Routing
- Configuration
- Crypto

Persistence
- Cache
- Database

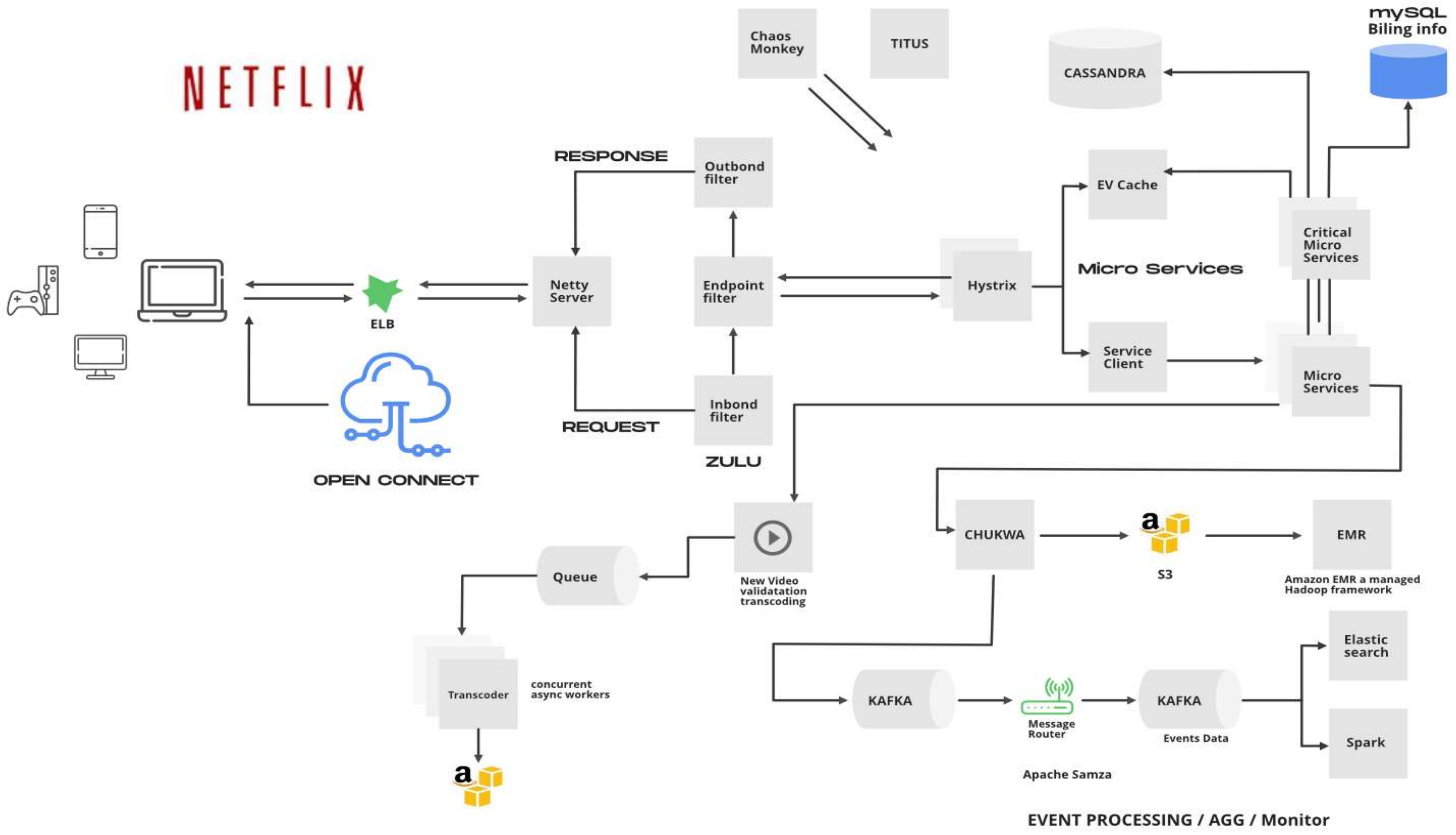
Ref: //www.slideshare.net/adrianco

Netflix Approach

Microservice at Netflix

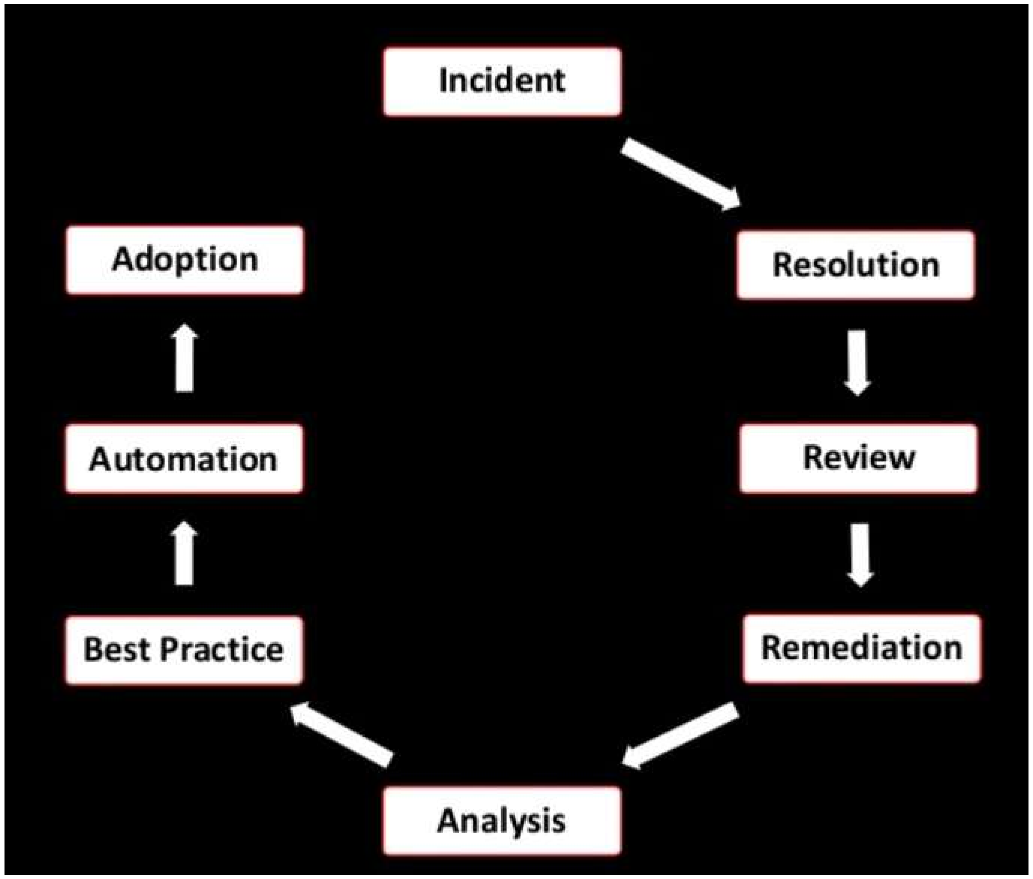


High-Level Design of Netflix System Design



Ref: https://www.geeksforgeeks.org/system-design/system-design-netflix-a-complete-architecture/

Netflix follows continuous learning



Self Study

- Journey from monolithic to SOA to Microservices: https://www.linkedin.com/pulse/evolving-architecture-journey-from-monolithic-soa-avita-katal/

- Uber case study – Read about Domain Oriented Microservices Architecture. Link: https://eng.uber.com/

- Github journey: https://www.infoq.com/articles/github-monolith-microservices/

References

- Research Paper: Challenges When Moving from Monolith to Microservice Architecture, Miika Kalske, Niko M¨akitalo, and Tommi Mikkonen
- Book: Monolith to Microservices by Sam Newman
- Book: Building Microservices by Sam Newman
- Book: Microservices Vs Service Oriented Architecture by Mark Richards
- Book: Microservices Patterns by Chris Richardson
- Link: https://www.ibm.com/cloud/blog/soa-vs-microservices
- Link: https://www.slideshare.net/adrianco
- Talks about Netflix by Josh Evans
- https://www.geeksforgeeks.org/system-design/system-design-netflix-a-complete-architecture/