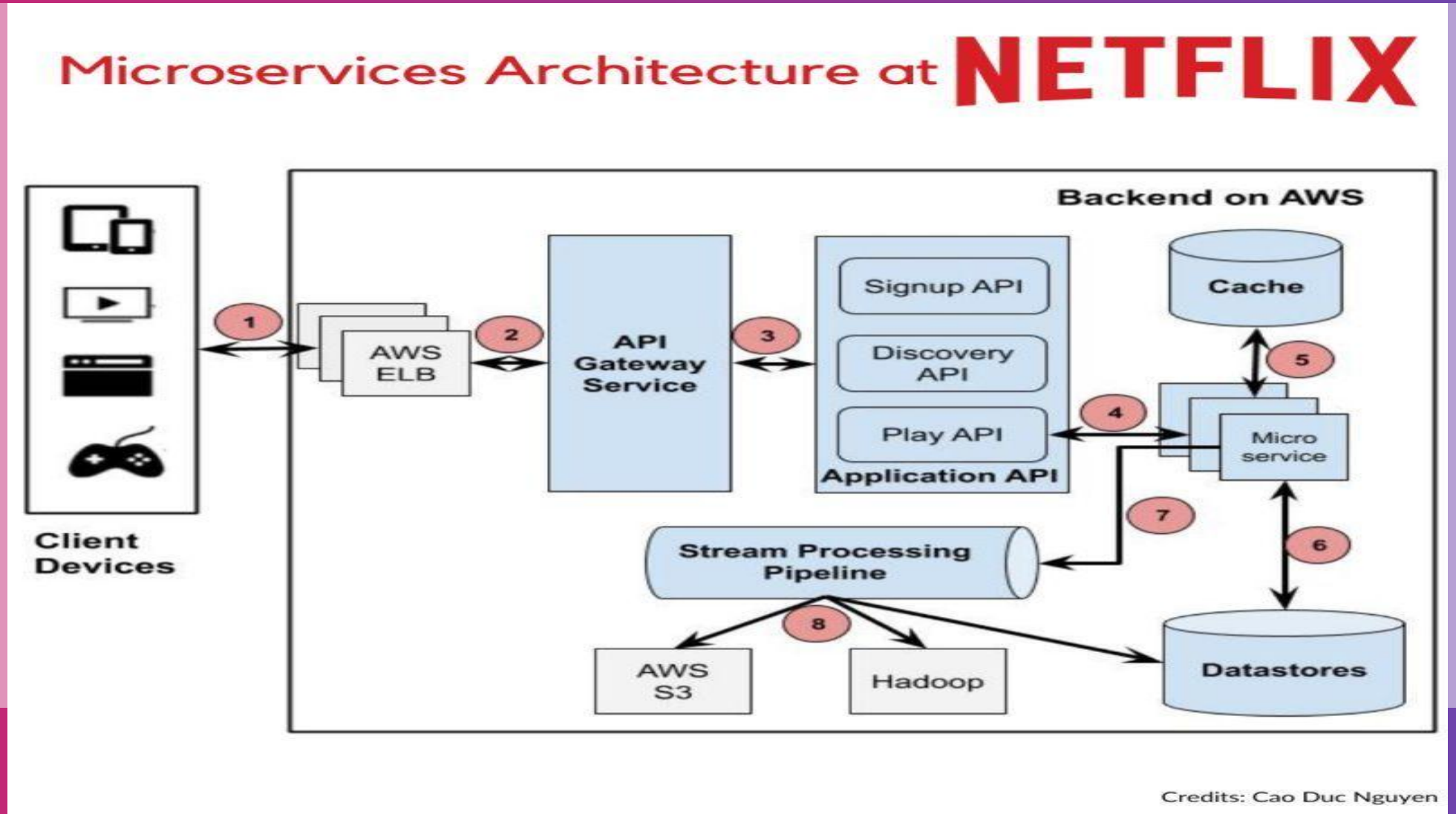
MICROSERVICES ARCHITECTURE



MONOLITHIC APPLICATIONS

Classic 3-Tier Monolithic Web Application

Just as the name suggests, ‘monolithic’ applications are applications where all of the parts — e.g., code base, business logic, database layer etc — are in one system. Legacy monolithic applications are usually server-side applications, i.e., hosted on-premises.

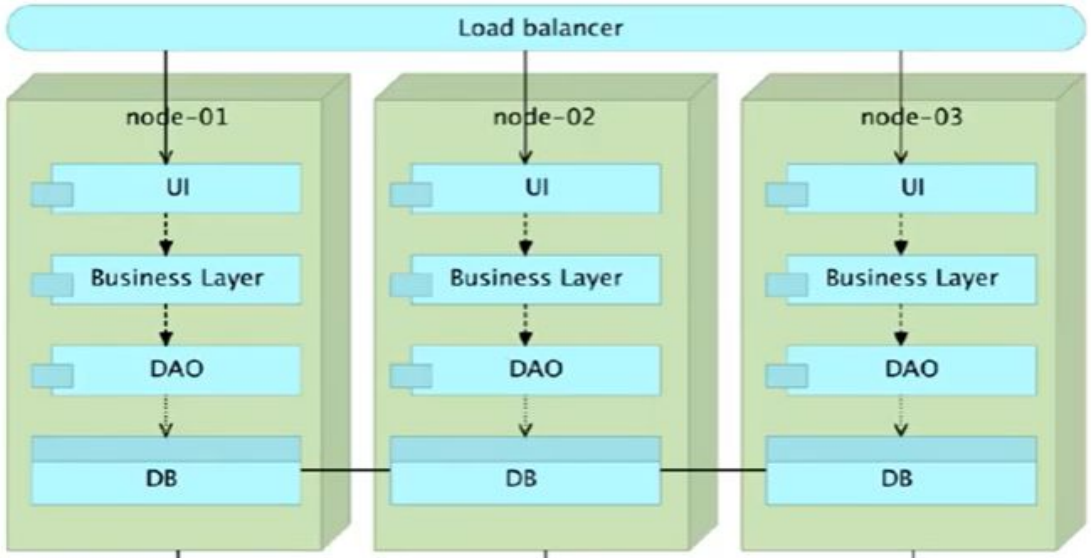
When the application requirements are small or minimal, some companies might opt to develop a monolithic application as it’s easier and more feasible. In the beginning, supporting it would’ve been comparatively easier because there weren’t many components in play.

However, as the years passed and more functionality was added, the monolithic application became exponentially bigger and more complicated. As a result, supporting it became more time consuming and expensive (the choice to go monolithic first resulted in technical debt).

For example, a bug in just one component or service could bring down the entire system. So if you had a faulty update, that update will cause a crash, which could result in dissatisfied users and a loss of revenue for your business. For some businesses, monoliths aren’t future proofed.

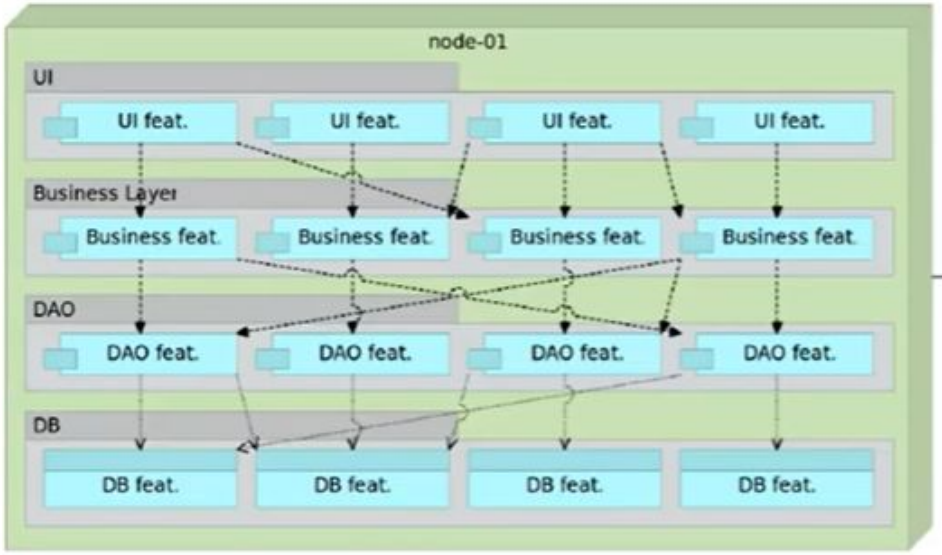
Scaling and increased Features

Scaling Monolithic Application



Scaling monolithic applications is very resource inefficient since everything needs to be duplicated on multiple nodes. There is no option to detect bottlenecks and scale or separate them from the rest of the application.

Monolithic Application with Increased Number of Features



When an application becomes bigger and the number of features increase, initial design based on horizontal layers becomes less efficient. Tight coupling between separate features, longer paths for potentially simple solutions, increased complexity, increased development and testing time, and so on.

Problems with monolithic

- Huge and intimidating code base for developers.
- Development tools get overburdened
  - *refactorings take minutes*
  - *builds take hours*
  - *testing in continuous integration takes days*
- Scaling is limited
  - *Running a copy of the whole system is resource intense*
  - *It doesn't scale with the data volume out of the box*
- Deployment frequency is limited
  - *Redeploying means halting the whole system*
  - *Redeployments will fail and increase the perceived risk of deployment*

Service Oriented Architecture(SOA)

If you to buy something online, how to access the service for buying?

- **Lookup for service provider**
- **Find the service offered according to the need**
- **Access the service.**

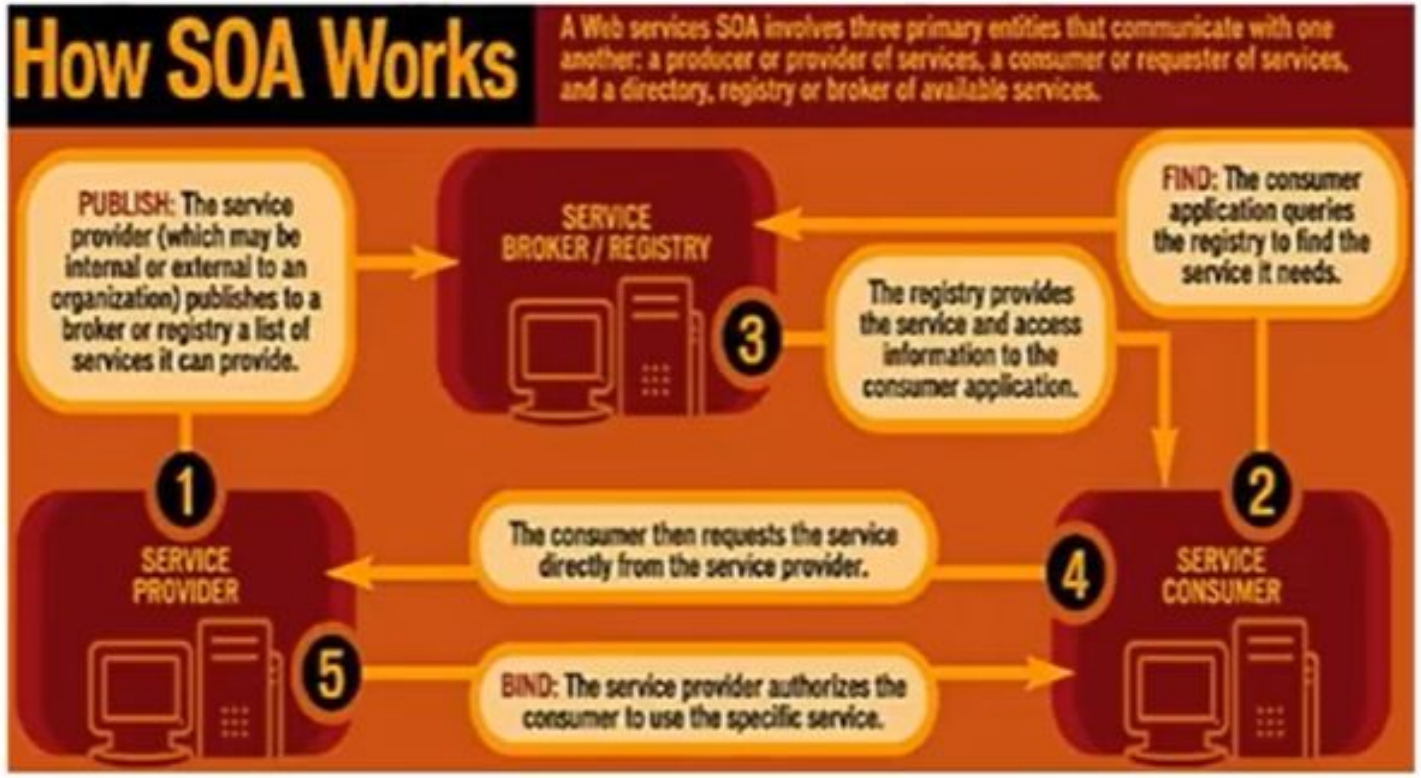
The policies, practices, frameworks that enable application functionality to be provided and consumed as sets of services published at a granularity relevant to the service customer.

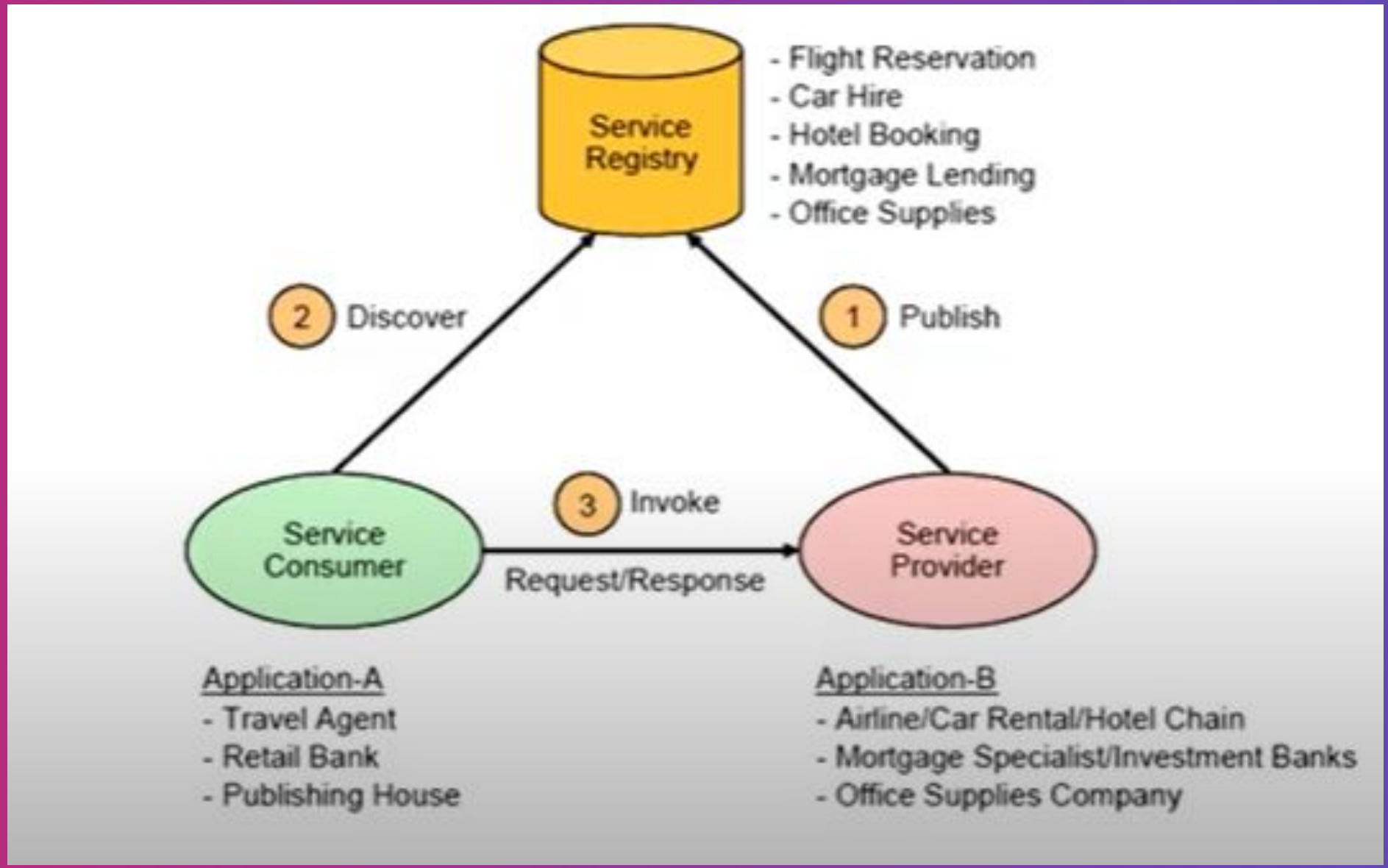
Service can be

Invoked – published and - discovered.

Services are abstracted from the implementation using a single, standard-based form of interface.(eg. Bank web service for payment gateway)

How SOA works?




Service Registry
- Flight Reservation
- Car Hire
- Hotel Booking
- Mortgage Lending
- Office Supplies
2 Discover
1 Publish
3 Invoke
Service Consumer
Request/Response
Service Provider
Application-A
- Travel Agent
- Retail Bank
- Publishing House
Application-B
- Airline/Car Rental/Hotel Chain
- Mortgage Specialist/Investment Banks
- Office Supplies Company

PROS and CONS of SOA

PROS

- Reusability of services

  Due to the self-contained and loosely coupled nature of functional components in service-oriented applications, these components can be reused in multiple applications without influencing other services.

- Better maintainability

  Since each software service is an independent unit, it's easy to update and maintain it without hurting other services. For example, large enterprise apps can be managed easier when broken into services.

- Higher reliability

  Services are easier to debug and test than are huge chunks of code like in the monolithic approach. This, in turn, makes SOA-based products more reliable.

- Parallel development

  As a service-oriented architecture consists of layers, it advocates parallelism in the development process. Independent services can be developed in parallel and completed at the same time

CONS

- Complex Management

  The main drawback of a service-oriented architecture is its complexity. Each service has to ensure that messages are delivered in time. The number of these messages can be over a million at a time, making it a big challenge to manage all services.

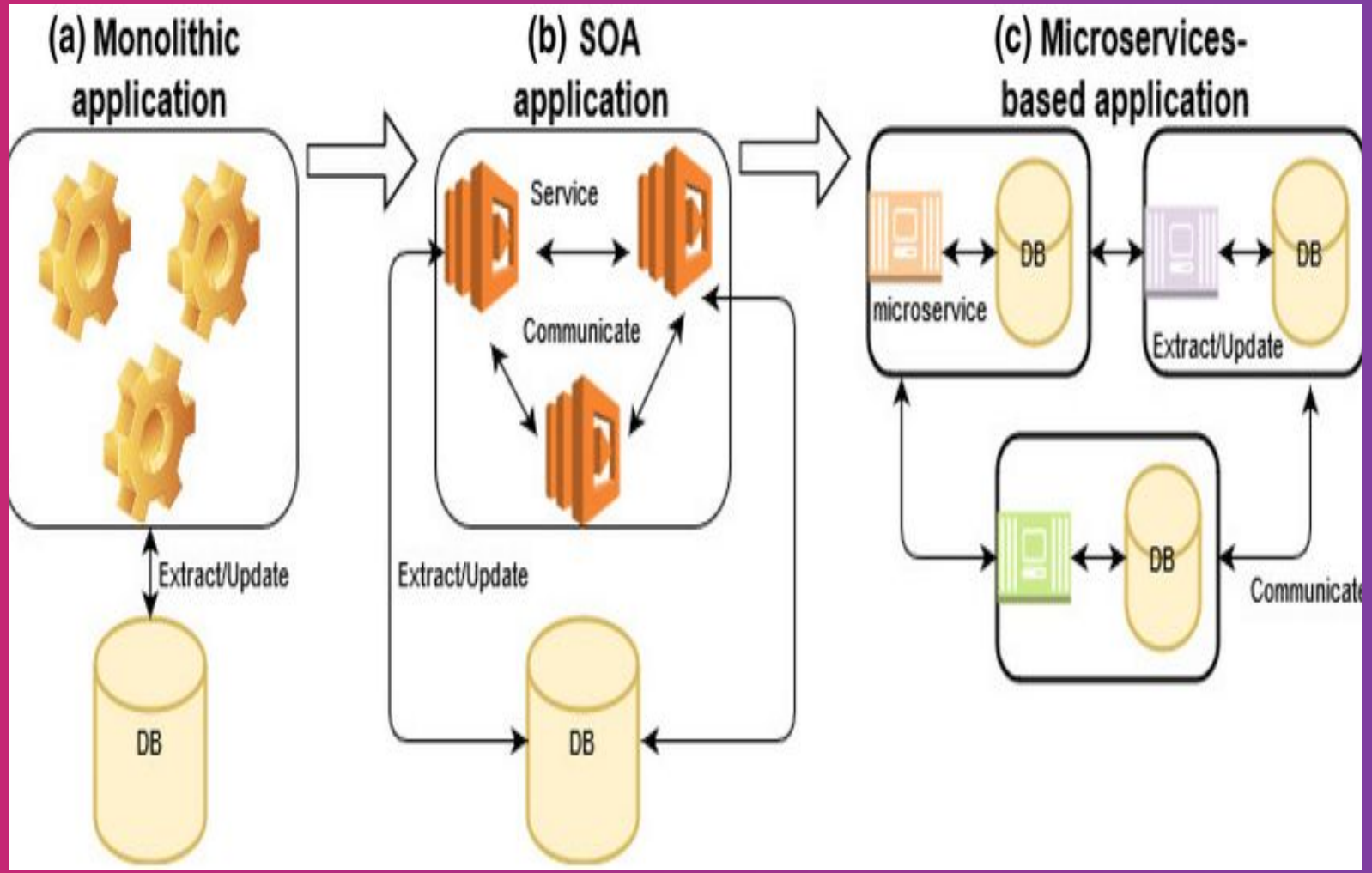
- High investment costs

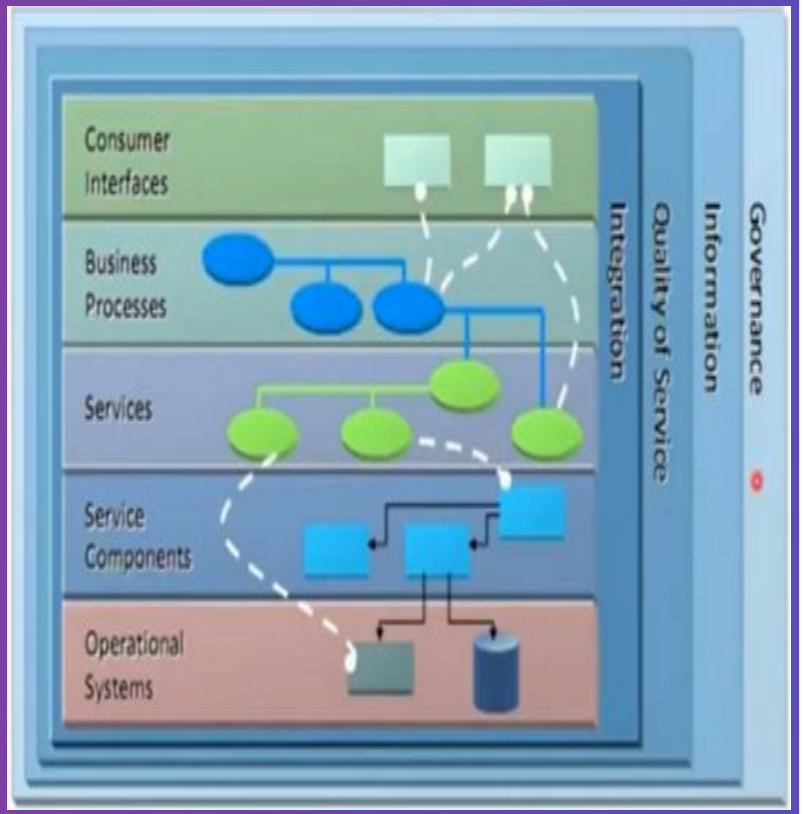
  SOA development requires a great upfront investment of human resources, technology, and development.

- Extra overload

  In SOA, all inputs are validated before one service interacts with another service. When using multiple services, this increases response time and decreases overall performance.

1980s 2000s 2010

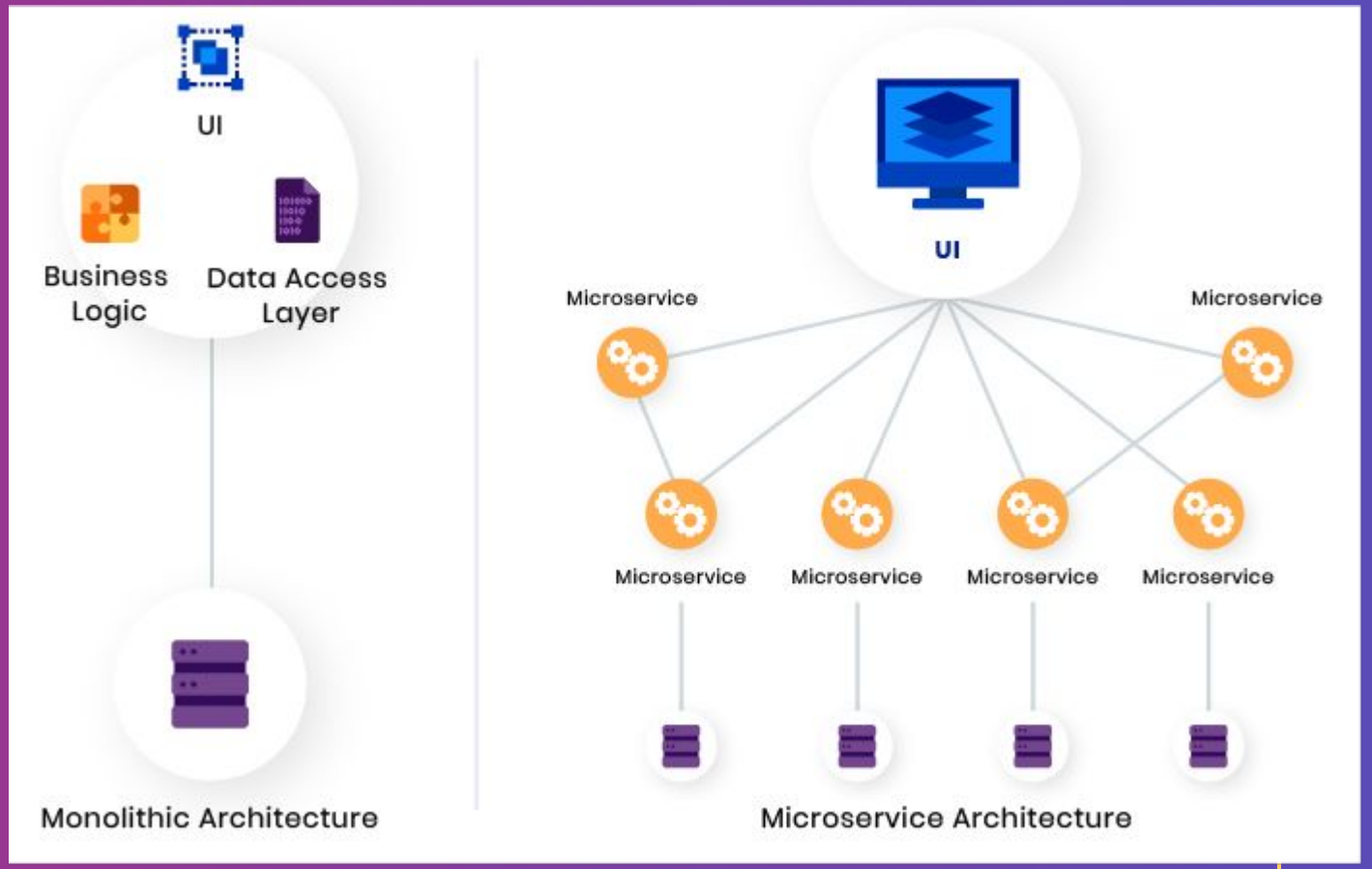




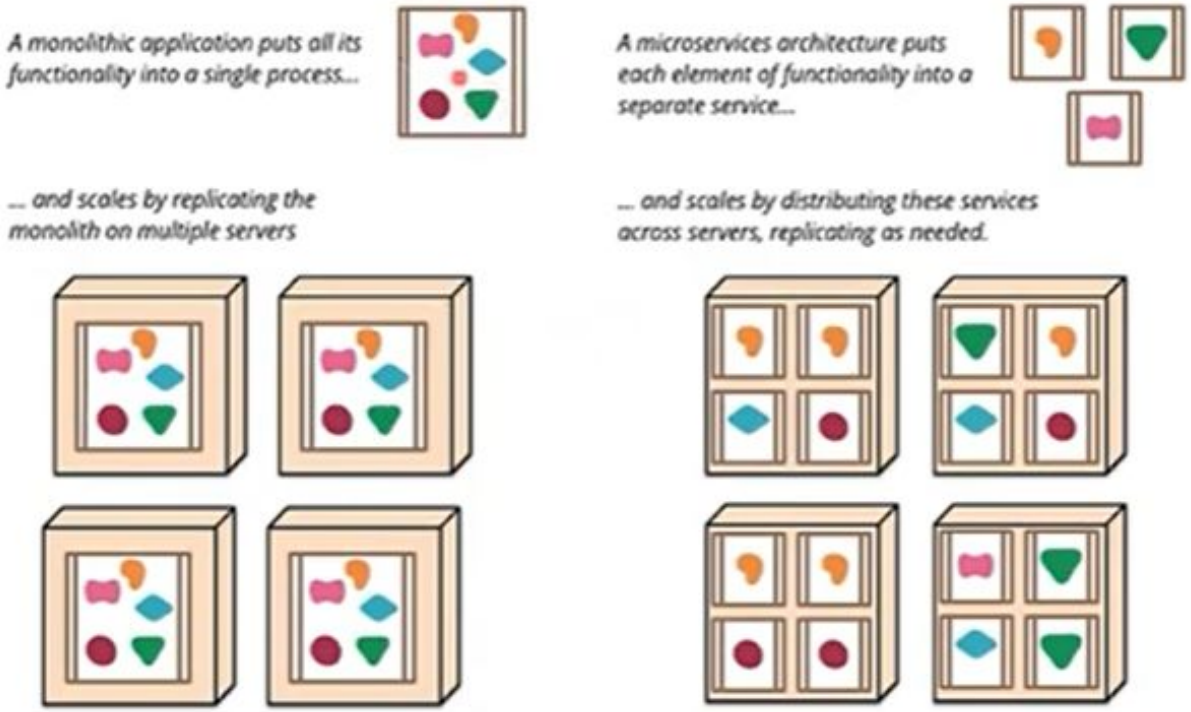
Microservice

The microservices approach focuses mainly on business priorities and capabilities, whereas the monolithic approach is organized around technology layers, UIs, and databases. The microservices approach has become a trend in recent years as more and more enterprises become agile and move toward DevOps.

- Loosely coupled or decoupled services.
- Separate but shared database.
- Independently managed.
- Examples of companies that have evolved from a monolithic approach to microservices. Among the most prominent are Netflix, Amazon, Twitter, eBay, and PayPal.

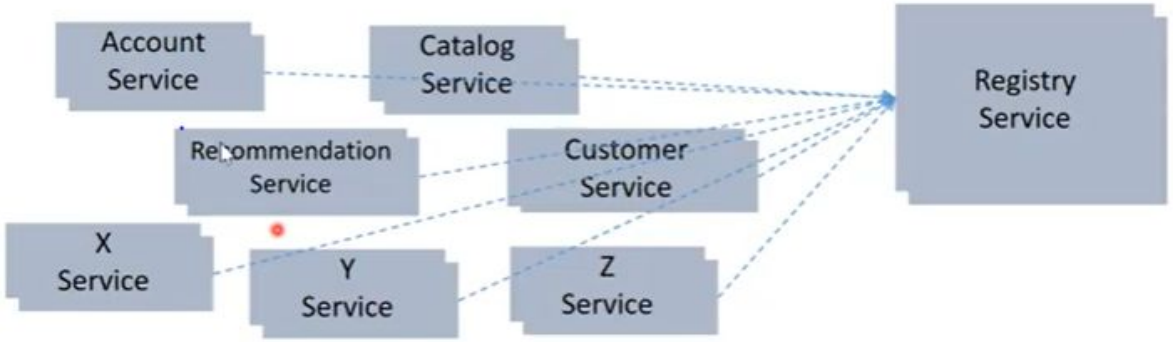


Microservices. Scalability



Service Discovery

- 100s of MicroServices
  - Need a Service Metadata Registry (Discovery Service)



PROS and CONS of Microservice

PROS

- Easy to develop, test, and deploy

  Since a deployment unit is small, it facilitates and speeds up development and release. Besides, the release of one unit isn't limited by the release of another unit that isn't finished. And the last plus here is that the risks of deployment are reduced as developers deploy parts of the software, not the whole app.

- Ability to scale horizontally

  Vertical scaling (running the same software but on bigger machines) can be limited by the capacity of each service. But horizontal scaling (creating more services in the same pool) isn't limited and can run dynamically with microservices. Furthermore, horizontal scaling can be completely automated.

- Increased agility

  Each individual part of an application can be built independently due to the decoupling of microservice components.

CONS

- Complexity

  This type of architecture requires careful planning, enormous effort, team resources, and skills. The reasons for high complexity are the following:

  - ✔ Increased demand for automation, as every service should be tested and monitored
  - ✔ Available tools don't work with service dependencies
  - ✔ Data consistency and transaction management becomes harder as each service has a database

- Security concerns

  In a microservices application, each functionality that communicates externally via an API increases the chance of attacks. These attacks can happen only if proper security measurements aren't implemented when building an app

- Different programming languages

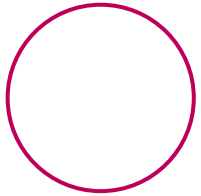
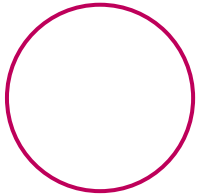
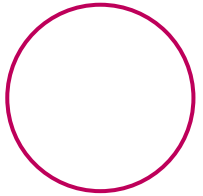
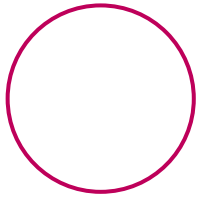
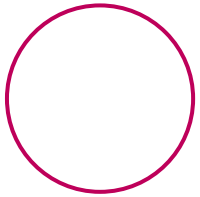
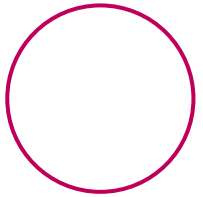
  The ability to choose different programming languages is two sides of the same coin. Using different languages make deployment more difficult. In addition, it's harder to switch programmers between development phases when each service is written in a different language.

Bottom line

The monolithic model isn't outdated, and it still works great in some cases. Some giant companies like Etsy stay monolithic despite today's popularity of microservices. Monolithic software architecture can be beneficial if your team is at the founding stage, you're building an unproven product, and you have no experience with microservices. Monolithic is perfect for startups that need to get a product up and running as soon as possible. However, certain issues mentioned above come with the monolithic package.

The SOA approach is best suited for complex enterprise systems such as those for banks. A banking system is extremely hard to break into microservices. But a monolithic approach also isn't good for a banking system as one part could hurt the whole app. The best solution is to use the SOA approach and organize complex apps into isolated independent services.

Microservices are good, but not for all types of apps. This pattern works great for evolving applications and complex systems. Consider choosing a microservices architecture when you have multiple experienced teams and when the app is complex enough to break it into services. When an application is large and needs to be flexible and scalable, microservices are beneficial

Monolithic, Service Oriented Architecture(SOA) and Microservice Architecture

Prepared by Asst. Prof. Aditi Prajapati,
Department of Information Technology, RJ College

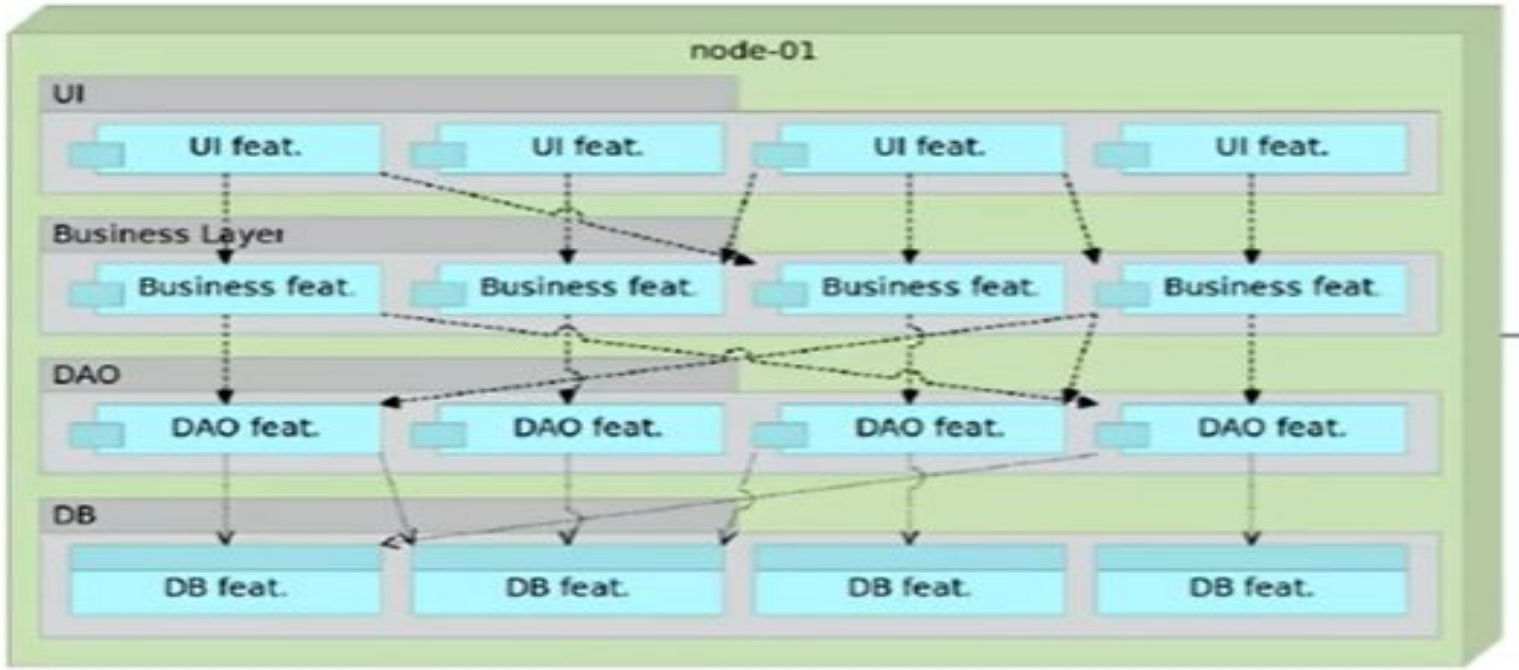
Contents Overview

- Monolithic Architecture
- Service Oriented Architecture(SOA)
- Microservice Architecture
- Difference between them

Monolithic Architecture

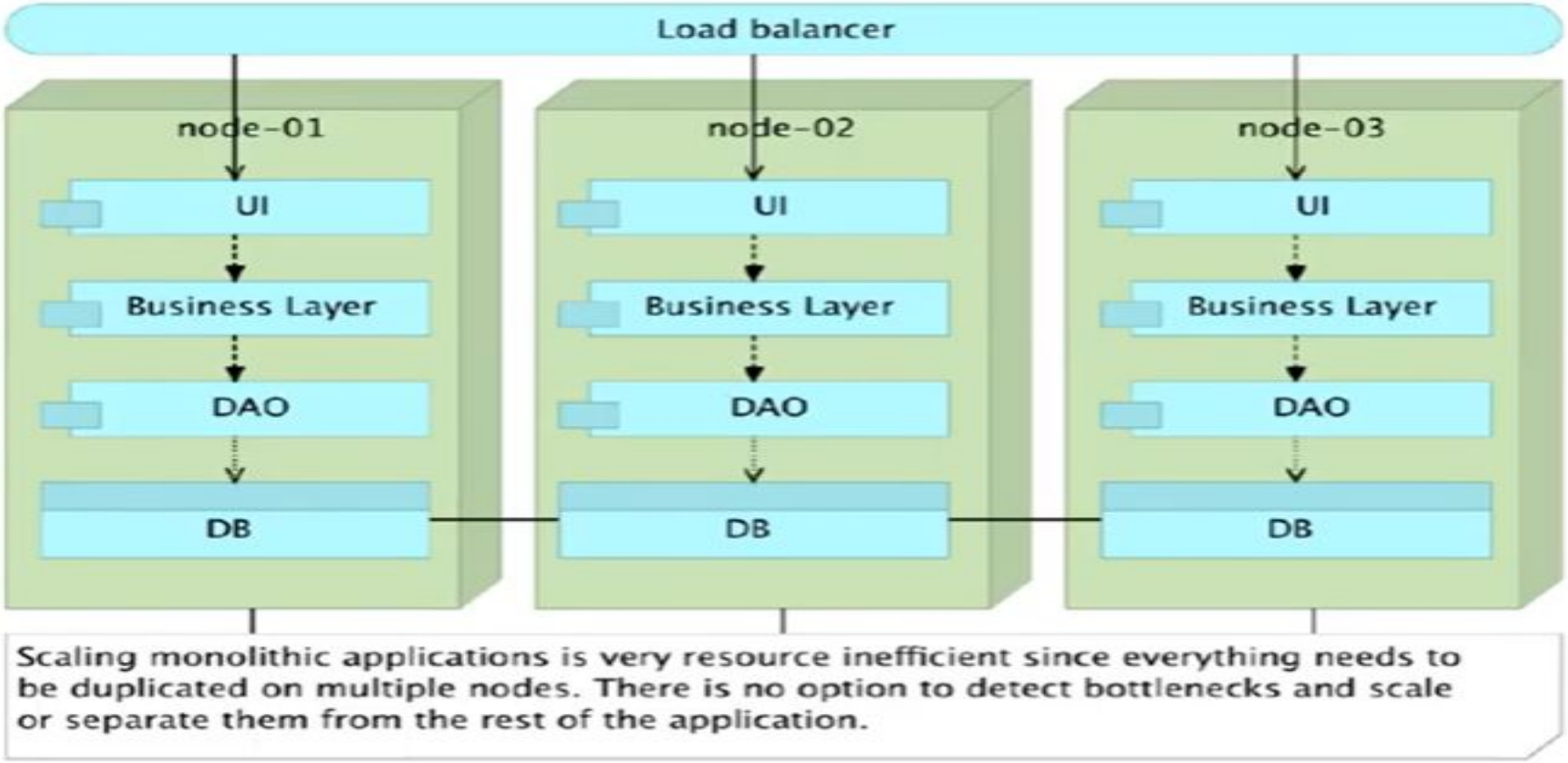
‘Monolithic’ applications are applications where all of the parts — e.g., code base, business logic, database layer etc — are in one system.

Monolithic Application with Increased Number of Features



When an application becomes bigger and the number of features increase, initial design based on horizontal layers becomes less efficient. Tight coupling between separate features, longer paths for potentially simple solutions, increased complexity, increased development and testing time, and so on.

Scaling Monolithic Application



Service Oriented Architecture(SOA)

If you to buy something online, how to access the service for buying?

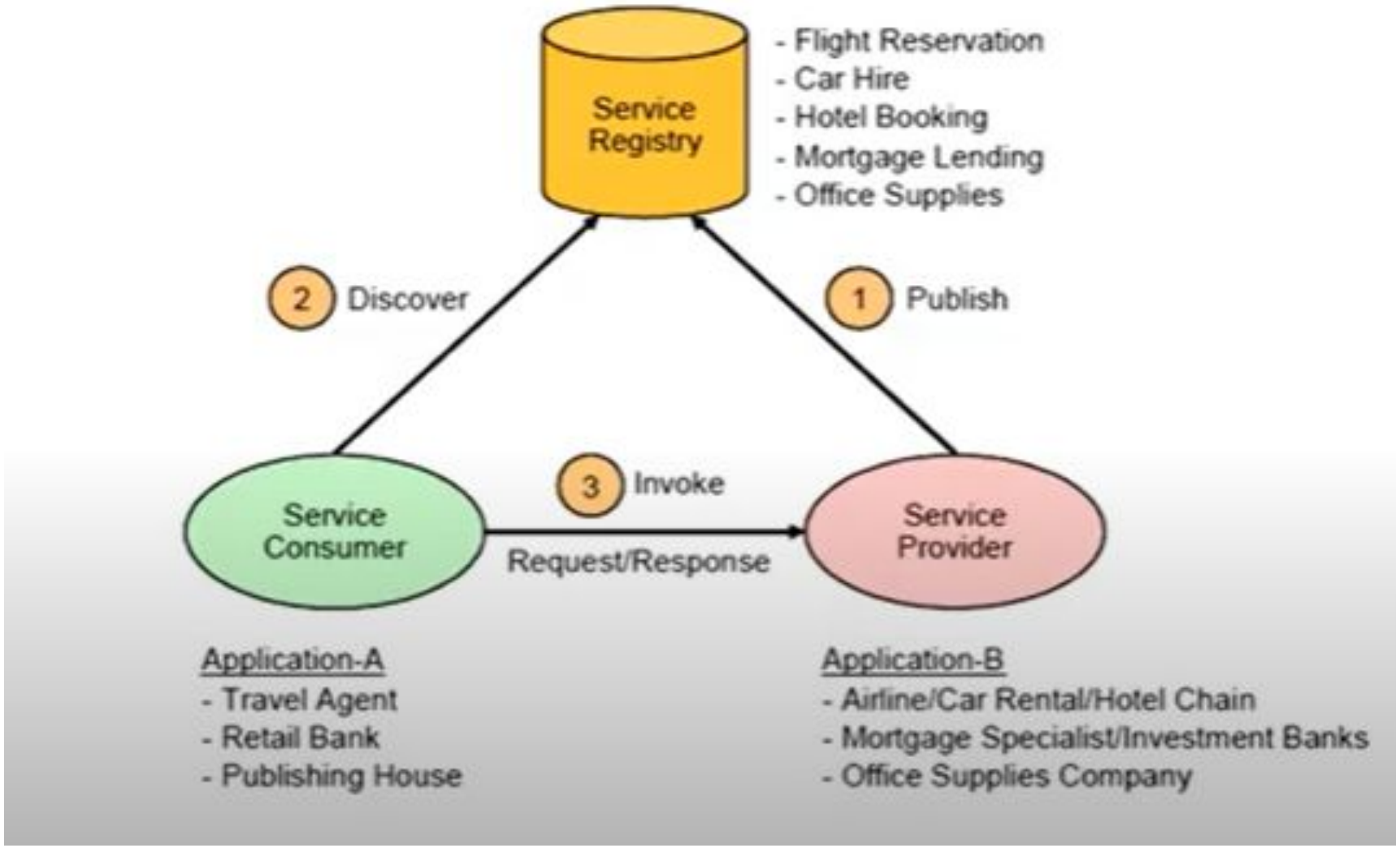
- **Lookup for service provider**
- **Find the service offered according to the need**
- **Access the service.**

The policies, practices, frameworks that enable application functionality to be provided and consumed as sets of services published at a granularity relevant to the service customer.

Service can be

Invoked – published and - discovered.

Services are abstracted from the implementation using a single, standard-based form of interface.(eg. Bank web service for payment gateway)


Service Registry
- Flight Reservation
- Car Hire
- Hotel Booking
- Mortgage Lending
- Office Supplies
2 Discover
1 Publish
3 Invoke
Request/Response
Service Consumer
Service Provider
Application-A
- Travel Agent
- Retail Bank
- Publishing House
Application-B
- Airline/Car Rental/Hotel Chain
- Mortgage Specialist/Investment Banks
- Office Supplies Company

Microservice Architecture

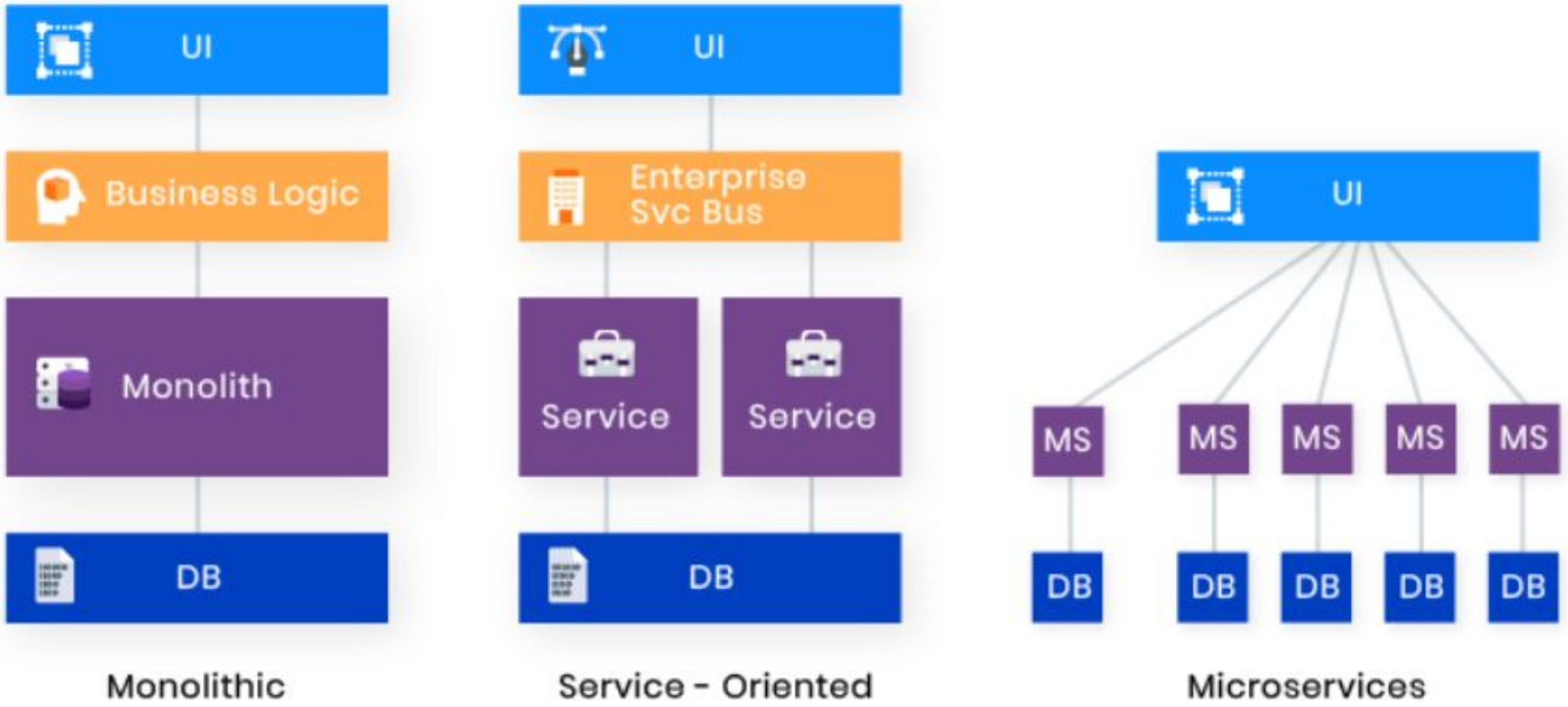
The microservices approach focuses mainly on business priorities and capabilities, whereas the monolithic approach is organized around technology layers, UIs, and databases. The microservices approach has become a trend in recent years as more and more enterprises become agile and move toward DevOps.

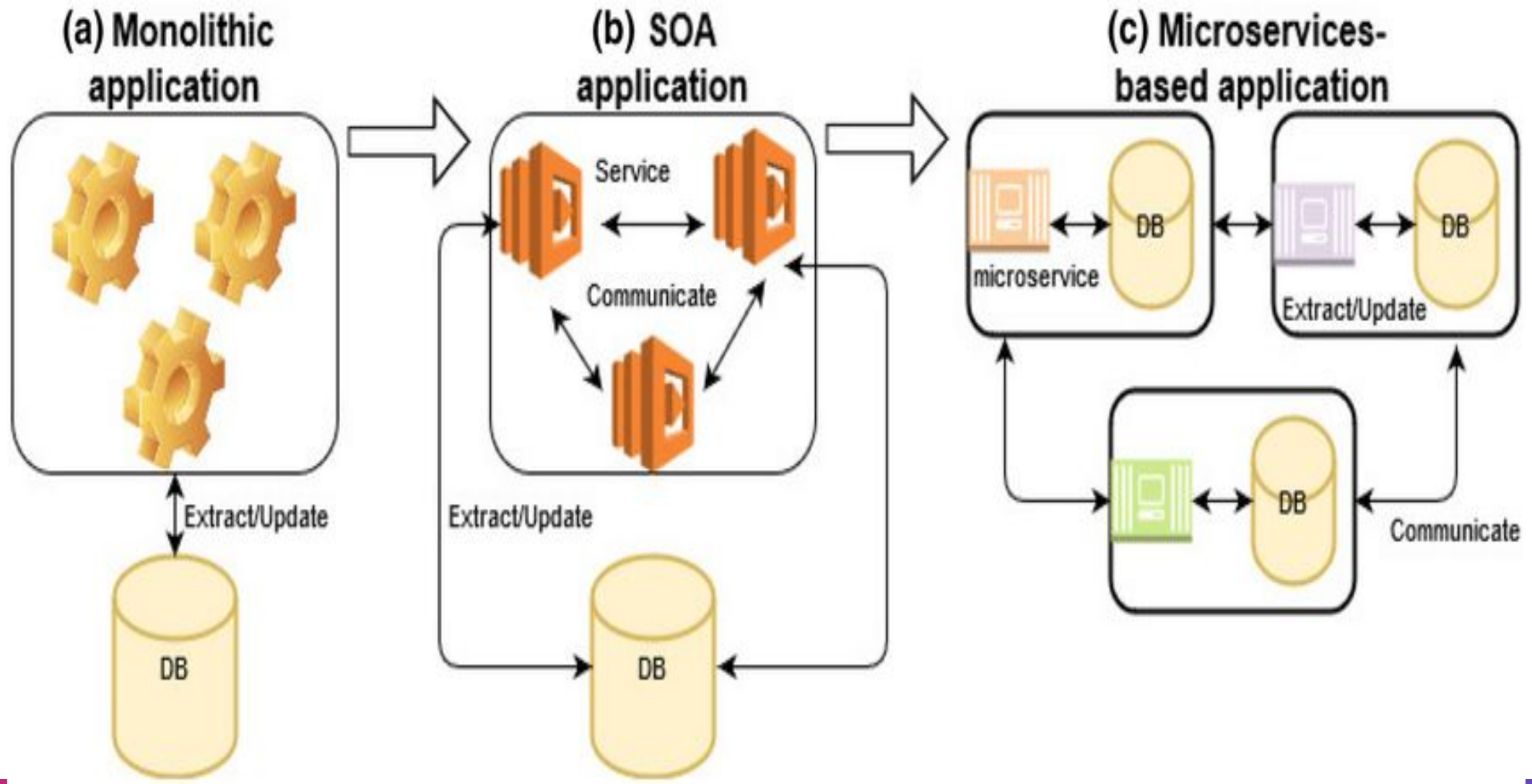
Loosely coupled or decoupled services.

Separate but shared database.

Independently managed.

Examples of companies that have evolved from a monolithic approach to microservices. Among the most prominent are Netflix, Amazon, Twitter, eBay, and PayPal.


UI
Business Logic
Monolith
DB
Monolithic
UI
Enterprise Svc Bus
Service
Service
DB
Service - Oriented
UI
MS
MS
MS
MS
MS
DB
DB
DB
DB
DB
Microservices


(a) Monolithic application
Extract/Update
DB
(b) SOA application
Service
Communicate
Extract/Update
DB
(c) Microservices-based application
microservice
DB
DB
Extract/Update
DB
Communicate

	Microservices	SOA	Monolithic
Design	Services are built in small units and expressed formally with business-oriented APIs.	Services can range in size anywhere from small application services to very large enterprise services including much more business functionality.	Monolithic applications evolve into huge size, a situation where understanding the entirety of the application is difficult.
Usability	Services exposed with a standard protocol, such as a RESTful API, and consumed/reused by other services and applications.	Services exposed with a standard protocol, such as SOAP and consumed/reused by other services – leverage messaging middleware.	Limited re-use is realized across monolithic applications.
Scalability	Services exist as independent deployment artifacts and can be scaled independently of other services.	Dependencies between services and reusable sub-components can introduce scaling challenges.	Scaling monolithic applications can often be a challenge.
Agility	Smaller independent deployable units ease build/release management, thereby high operational agility.	Enhances components sharing that increases dependencies and limits management capabilities.	Difficult to achieve operational agility in the repeated deployment of monolithic application artifacts.
Development	Developing services discretely allows developers to use the appropriate development framework for the task at hand.	Reusable components and standard practices helps developers with implementation.	Monolithic applications are implemented using a single development stack (i.e., JEE or .NET), which can limit the availability of “the right tool for the job”.

Understanding Microservice

Microservices are ideal for big systems.
Microservice architecture is goal-oriented.
Microservices are focused on replaceability.

Companies are more interested in the goal of improving changeability than finding a universal pattern or process. This goal can be achieved with improving the replaceability of components.

Q. What are characteristics of microservice?

Q. What are benefits of microservice?

Speed and safety at scale and in harmony is the microservice way

Deriving Business Value

For each organization, the balance will be a function of its delivery speed, the safety of its systems, and the growth of the organization's functional scope and scale

For example, A media company that aims to reach the widest possible audience for its content may place a much higher value on delivery speed than a retail bank whose compliance requirements mandate specific measures around safety.

Successful companies do not focus on increasing software delivery speed for its own sake. They do it for requirements. Similarly, the level of safety implemented in an organization's software system should be tied to specific business objectives.

Continue…

Microservice architecture benefits that drive delivery *speed* contribute real business value:

- Agility - allow organization to deliver new products, functions, and features more quickly.
- Composability - reduces development time.
- Comprehensibility of the software system - simplifies development planning, increases accuracy, and allows new resources to come up to speed more quickly.
- Independent deployability of components - gets new features into production more quickly
- Organizational alignment of services to teams - build more complex products and features iteratively.
- Polyglotism permits the use of the right tools for the right task. thus accelerating technology introduction and increasing solution options.

Continue…

Digital native consumers expect always-on services and are not shy about changing corporate allegiances. Outages or lost information can cause them to take their business elsewhere. A safe software system is indispensable. Microservice architecture benefits that drive *safety* contribute real business value:

- **Greater efficiency** in the software system **reduces infrastructure costs** and reduces the **risk of capacity-related service outages.**
- **Independent manageability** contributes to improved efficiency, and also r**educes the need for scheduled downtime.**
- **Replaceability** of components reduces t**he technical debt that can lead to aging, unreliable environments.**
- **Stronger resilience and higher availability ensure a good customer experience.**
- Better **runtime scalability** allows the software system t**o grow or shrink with the business.**
- Improved **testability** allows the business to **mitigate implementation risks.**

Defining a Goal-Oriented, Layered Approach

The first companies to use microservice architecture made the switch from monolithic applications once they passed a certain scale threshold. With the benefit of hindsight, and with an analysis of the common goals and benefits of microservice architecture, we can map out a set of layered characteristics to consider when adopting microservice architecture.

Modularized Microservice Architecture

At its most basic level, microservice architecture is about breaking up an application or system into smaller parts.

To help software delivery speed, modularized services are independently deployable. It is also possible to take a polyglot approach to tool and platform selection for individual services, regardless of what the service boundaries are. With respect to safety, services can be managed individually at this layer. Also, the abstracted service interfaces allow for more granular testing.

Continue…

Cohesive Microservice Architecture

In order to have a cohesive microservice architecture, it must already be modularized. Achieving service cohesion comes from defining the right service boundaries and analyzing the semantics of the system.

A cohesive microservice architecture can enable software speed by aligning the system's services with the supporting organization's structure. It can also yield composable services that are permitted to change at the pace the business dictates, rather than through unnecessary dependencies. Reducing the dependencies of a system featuring cohesive services also facilitates replaceability of services. Moreover, service cohesion lessens the need for highly orchestrated message exchanges between components,thereby creating a more efficient system.

Continue…

Systematized Microservice Architecture

After breaking the system into pieces through modularization, and addressing the services' contents through cohesion, it is time to examine the interrelationships between the services.

There are two ways speed of delivery is impacted in a systematized microservice architecture. The overall software system is only comprehensible when the connectivity between services is known. Also, agility is only possible when the impacts of changes on the whole system can be identified and assessed rapidly.

Although individual components may be isolated and made resilient in a modularized or cohesive microservice architecture, the system availability is not assured unless the interdependencies of the components are understood.

A maturity model for microservice architecture goals and benefits