WSO2CON 2024

Not Just Microservices: Rightsize Your Services!



Vidura Gamini Abhaya, PhD

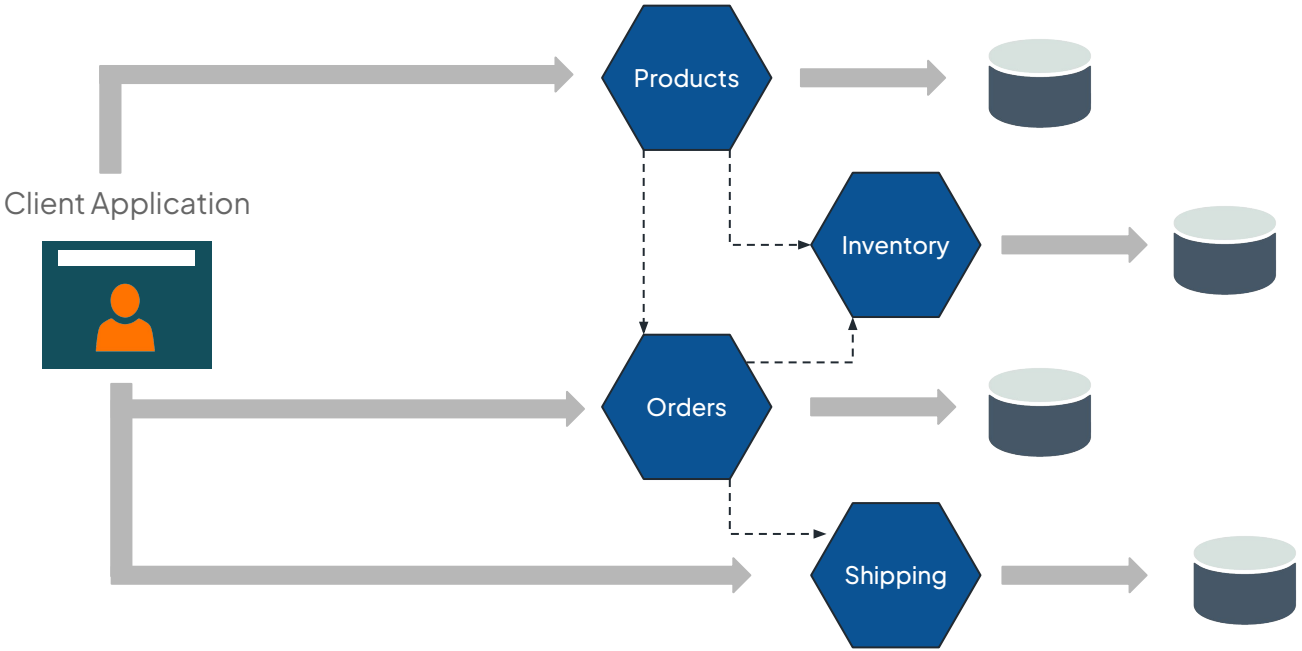
Head of Solution Architecture - Integration, Field CTO

WSO2

Microservices

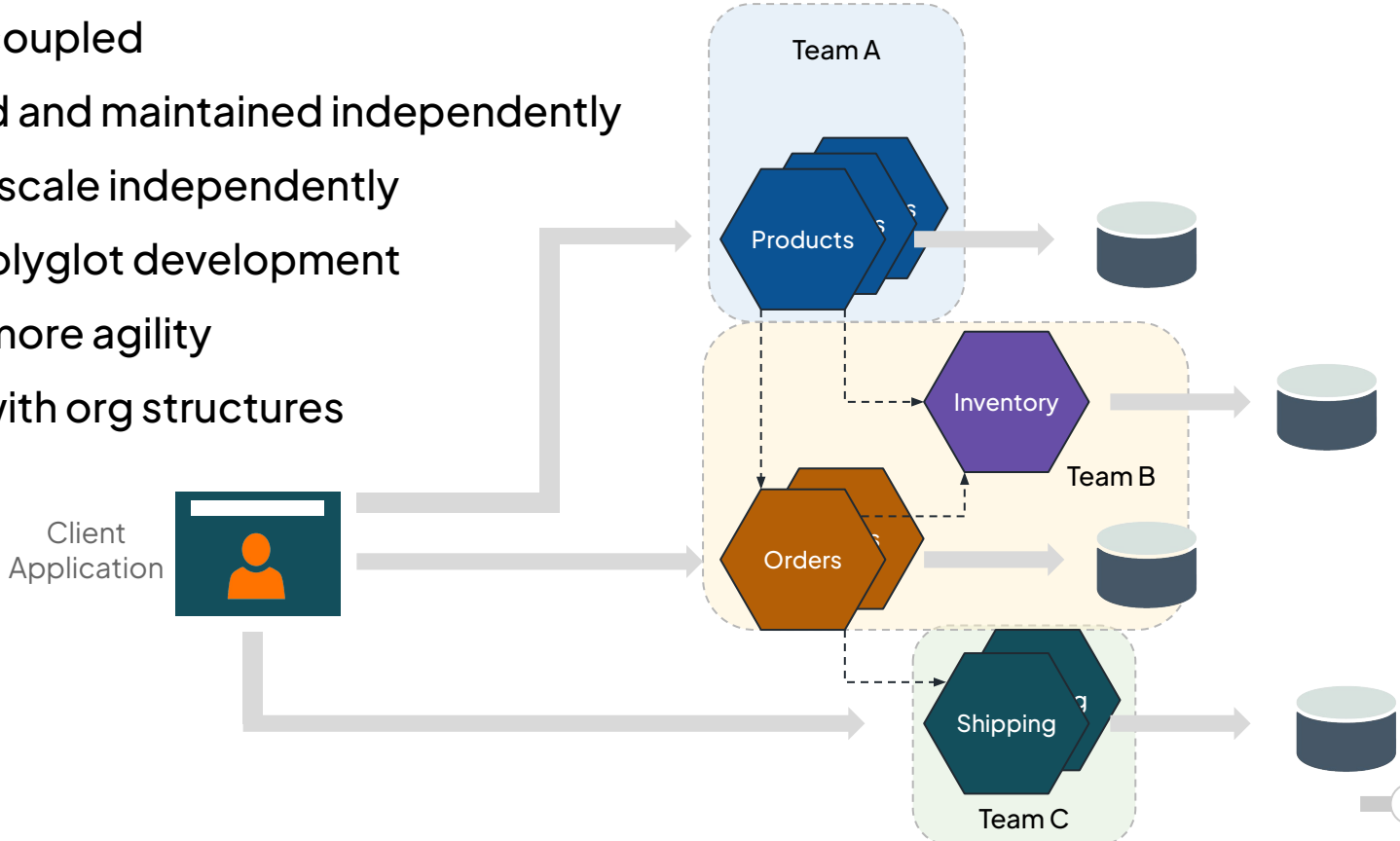
Microservices (Architecture)

An architectural style where an application is structured as a collection of independent services that are loosely coupled



They are popular because…

- Loosely coupled
- Deployed and maintained independently
- Ability to scale independently
- Enable polyglot development
- Achieve more agility
- Aligned with org structures



They come with a few catches…

The good, the bad and the ugly…

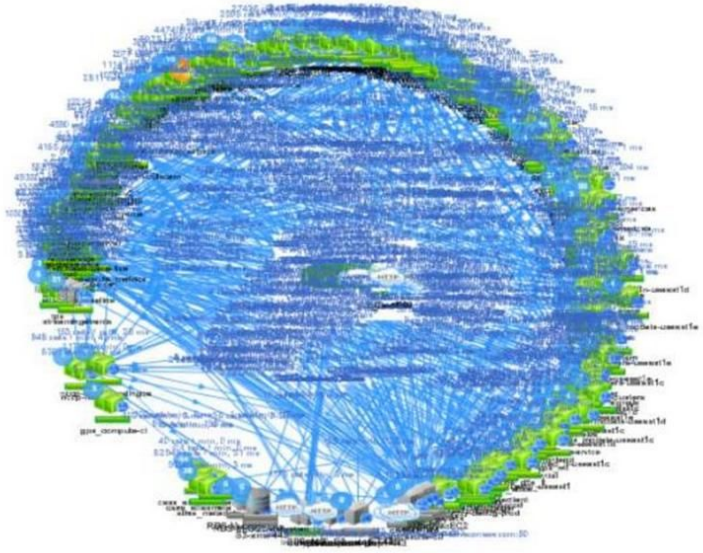
As the usage grows and the number of services increase…

- Multiple code bases to manage
- Large number of dependencies
- Complexity of the deployments
- Monitoring overhead
- Managing data consistency across services
- Communications overhead

The good, the bad and the ugly…

As the usage grows and the number of services increase…

- Multiple code bases to manage
- Large number of dependencies
- Complexity of the deployments
- Monitoring overhead
- Managing data consistency across services
- Communications overhead

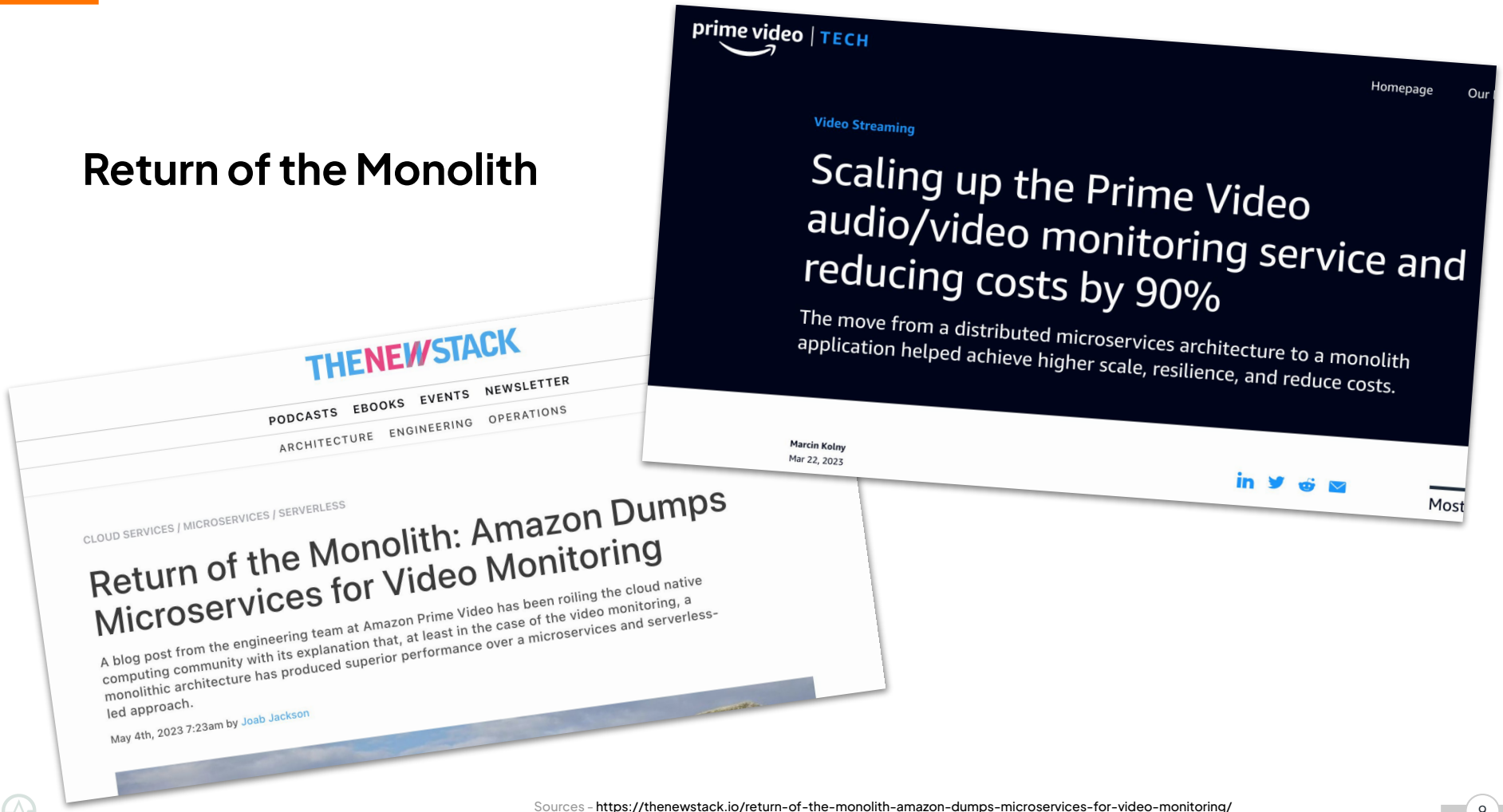


NETFLIX

Image source - https://walkthechat.com/wp-content/uploads/2019/05/netflix-microservices-diagram-bruce-wong.jpg

Back to basics…

Return of the Monolith



Sources – https://thenewstack.io/return-of-the-monolith-amazon-dumps-microservices-for-video-monitoring/
https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90

"A monolithic architecture is a choice, and a valid one at that. It may not be the right choice in all circumstances, any more than microservices are—but it's a choice nonetheless."

- Sam Newman -

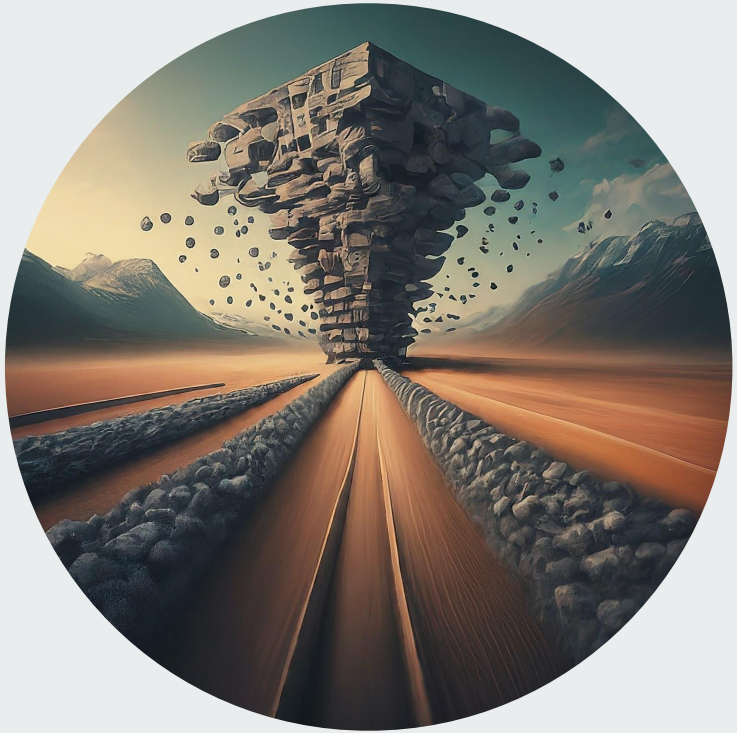


Simplicity of the Monolith

It is favoured by many due to...

- A single code base to manage
- Deployment simplicity
- More performant as no network latencies
- Ease of management
- Easier to Test
- Lower initial cost



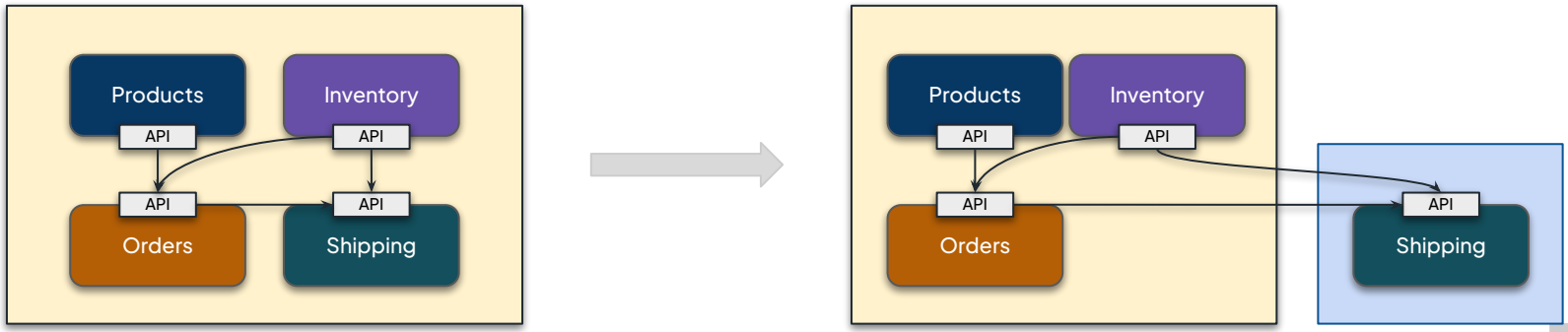


Can we go Hybrid?

An option that has the simplicity of the monolith and the flexibility of Microservices

The Modular Monolith

- Loosely coupled modules
- Modules communicating with each other using APIs
- Single code base to maintain
- Modules can be separated out when needed and deployed independently
- Within the monolith, modules can possibly communicate within a single process -> resulting in lower latency communication



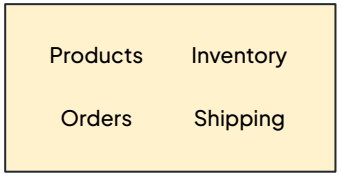
Rightsizing Services...

The Rightsizing Spectrum

Choosing the right service size based on the specific needs of the application

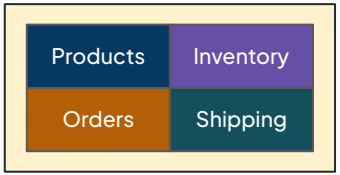
Simplicity ⟷ Granularity

Monolith



- Single code base
- Single deployment
- Management Simplicity

Modular Monolith



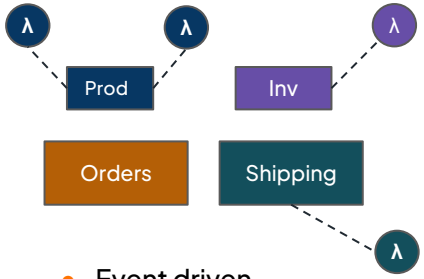
- Best of both worlds
- Single code base and single deployment
- Can act as the perfect bridge between both worlds

Microservices



- Better scalability
- Polyglot development
- Agility in development
- Better fit team structures

Serverless



- Event driven
- Ideal for self-contained routines
- Cost-effective

Factors to consider when Rightsizing

- Business needs : Transaction volumes, Latency requirements, Data privacy regulations
- Technical characteristics: payload sizes, complexity of logic, Fault tolerance requirements
- Team structure and Dev practices: team size, skill sets, deployment and monitoring processes

"You shouldn’t start a new project with microservices, even if you’re sure your application will be big enough to make it worthwhile."

- Martin Fowler -



Rightsizing Techniques and Examples

- Start with a monolith (start simple)
- Design the modules to be loosely coupled (i.e. Modular Monolith)
- Decompose the monolith strategically -> Use Strangler Fig pattern

Strangler Fig Pattern





Image source - https://umtrees.wordpress.com/2015/10/20/strangler-fig/comment-page-1/

Strangler Fig Pattern





Monolith

Time

Image source - https://umtrees.wordpress.com/2015/10/20/strangler-fig/comment-page-1/

Strangler Fig Pattern







Image source - https://umtrees.wordpress.com/2015/10/20/strangler-fig/comment-page-1/

Strangler Fig Pattern





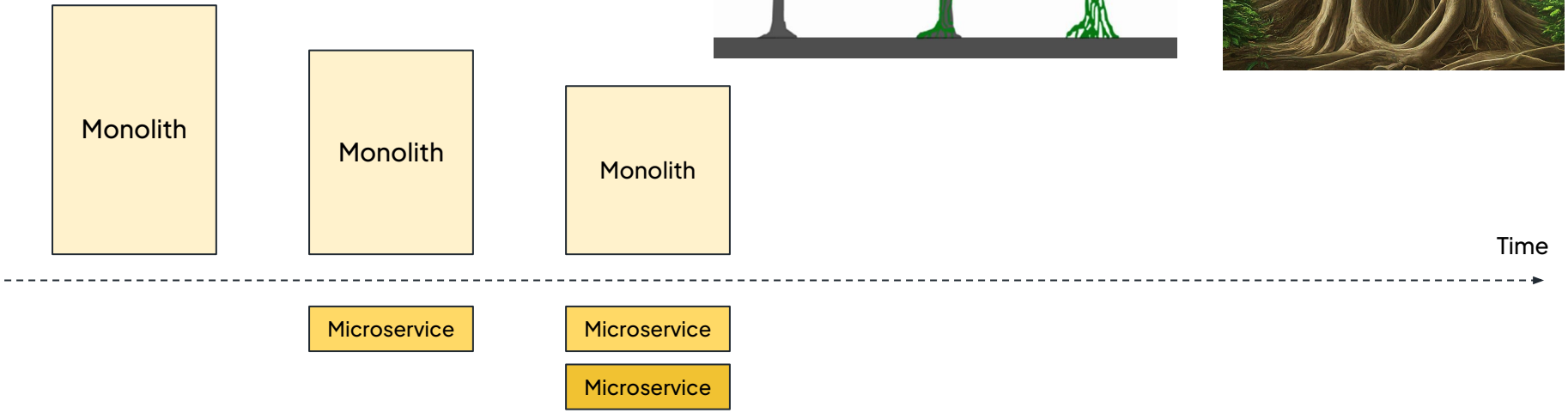


Image source - https://umtrees.wordpress.com/2015/10/20/strangler-fig/comment-page-1/

Strangler Fig Pattern





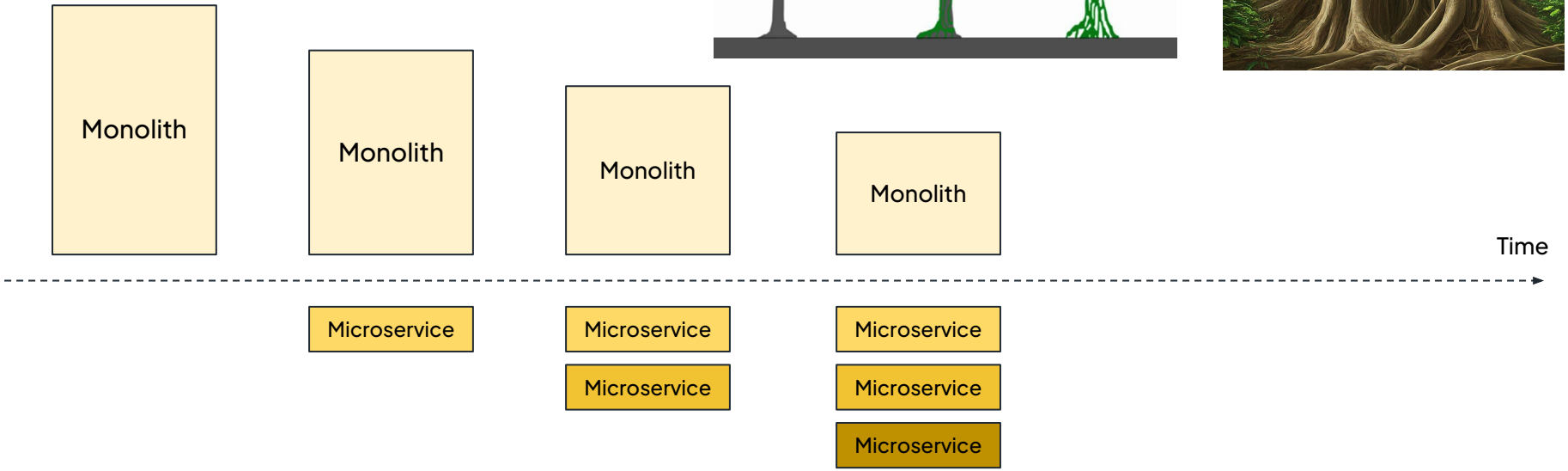


Image source - https://umtrees.wordpress.com/2015/10/20/strangler-fig/comment-page-1/

Strangler Fig Pattern





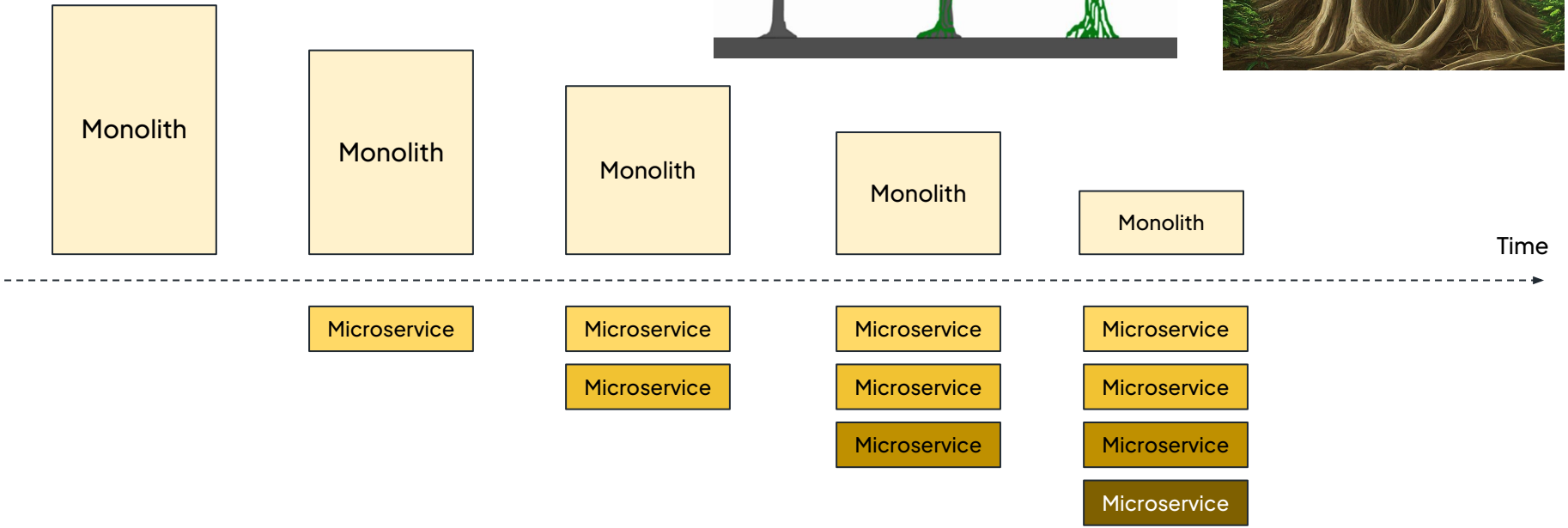


Image source - https://umtrees.wordpress.com/2015/10/20/strangler-fig/comment-page-1/

Strangler Fig Pattern





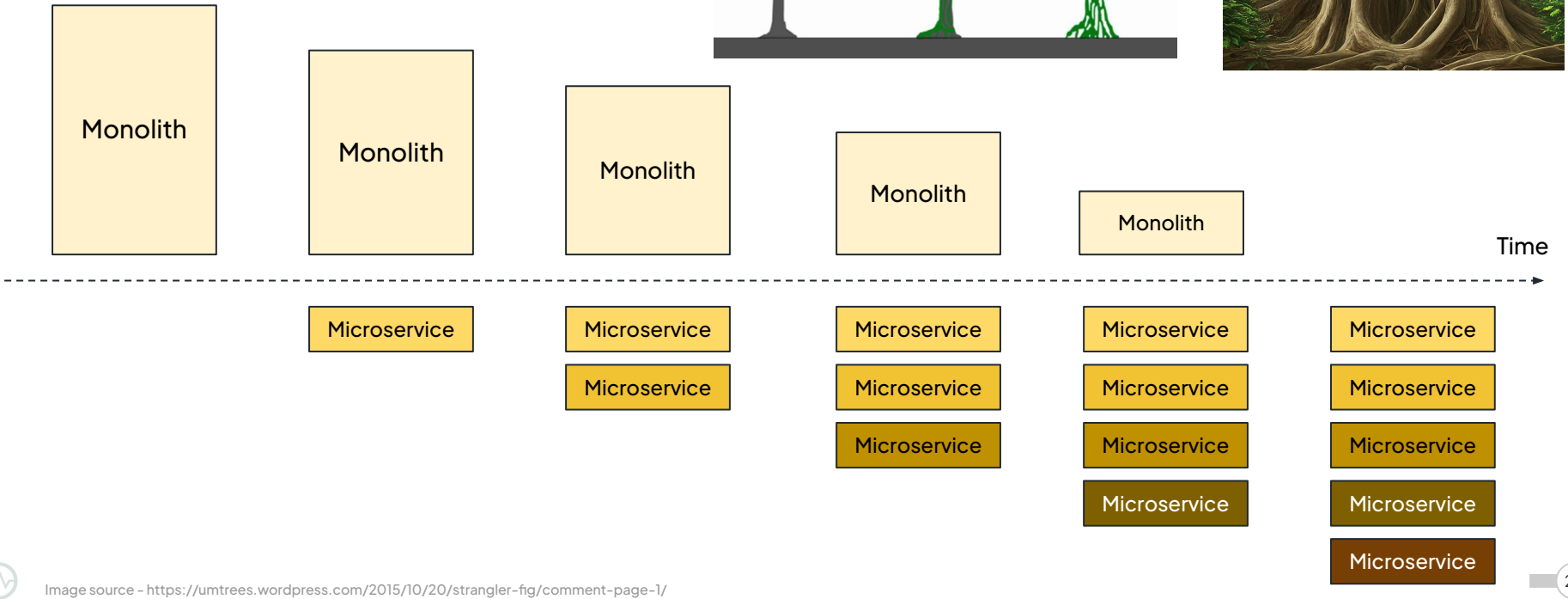


Image source - https://umtrees.wordpress.com/2015/10/20/strangler-fig/comment-page-1/

Rightsizing Techniques and Examples

- Start with a monolith (start simple)
- Design the modules to be loosely coupled (i.e. Modular Monolith)
- Decompose the monolith strategically -> Use Strangler Fig pattern
- Refactor microservices - combine small services
- Explore serverless for specific tasks
- Go hybrid as your architecture evolves

Some well known examples of Hybrid Architectures

- PayPal

  Known for adopting Microservices Architecture alongside Monoliths. This Hybrid approach allowed them to evolve their architecture while relying on the stability and reliability of the Monolith

- SoundCloud

  Evolved their Monolithic architecture to a Hybrid one by adopting Microservices for certain functionalities on their platform.

Challenges and Trade-offs

- Refactoring may be seen as too much maintenance work
- Tooling Complexity
- Monitoring Overhead
- Agility vs Simplicity
- Scalability vs Maintainability
- Management complexity



"Rightsizing services is like trimming a bonsai tree. You remove what's unnecessary to cultivate optimal growth and efficiency"

"Simplicity is the ultimate sophistication"

- Leonardo da Vinci -



Key Takeaways

- Rightsizing services is important
- It is an ongoing process, and not a one-time decision
- Start simple, proceed with necessity
- Consider all factors (business, technical, org structures, etc.)
- There is no one-size-fits-all solution !



Question Time!



Thank You!