Modular Monoliths: The Goldilocks Architecture?

Steve “ardalis” Smith

@ardalis

steve@nimblepros.com | NimblePros.com

NimblePros

Better software development. Delivered.

Courses!
DomeTrain.com

https://bit.ly/3T1pC17

Software Architecture

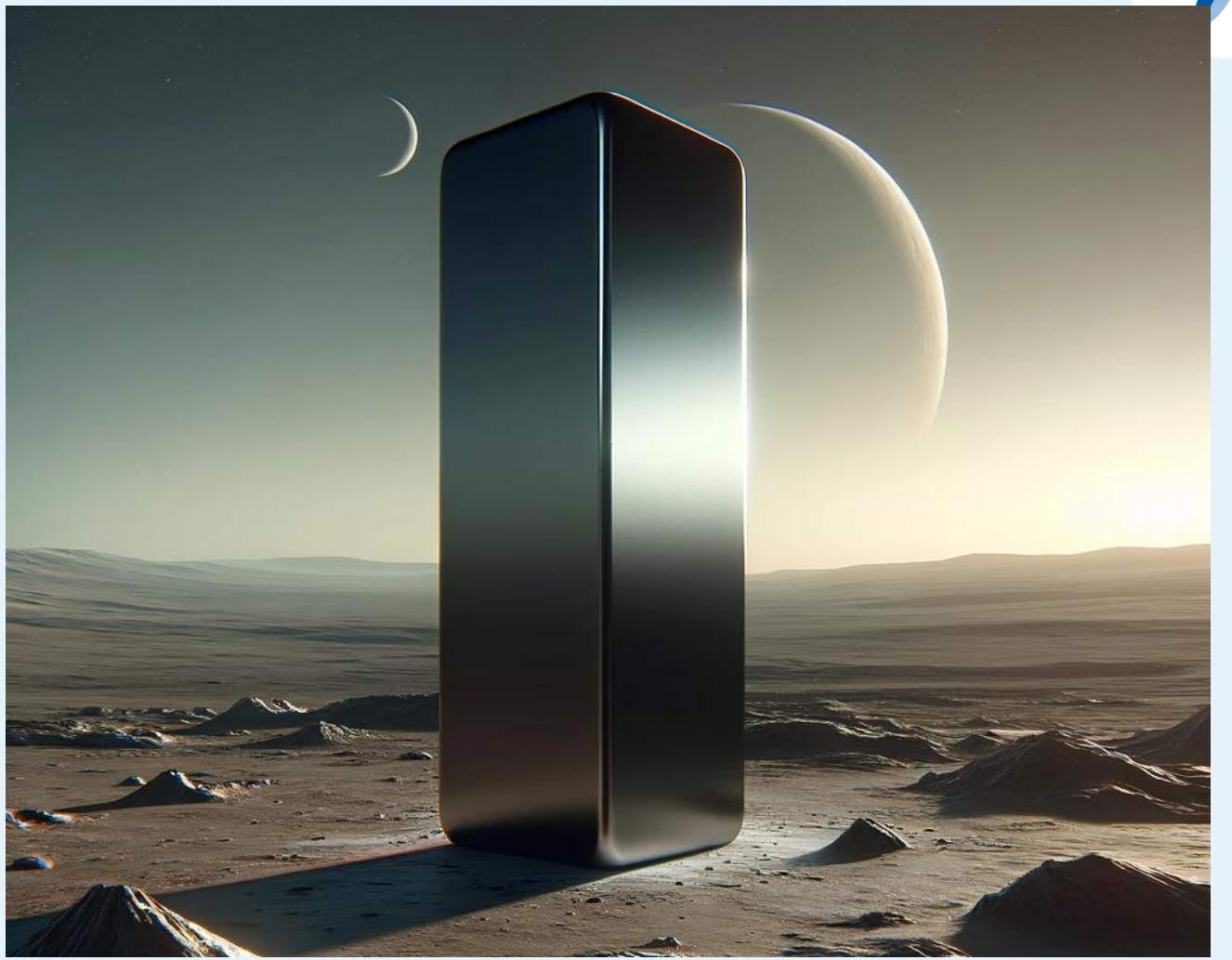
Everything is a trade off

Monolithic Architecture

Simply describes how an application is organized and deployed

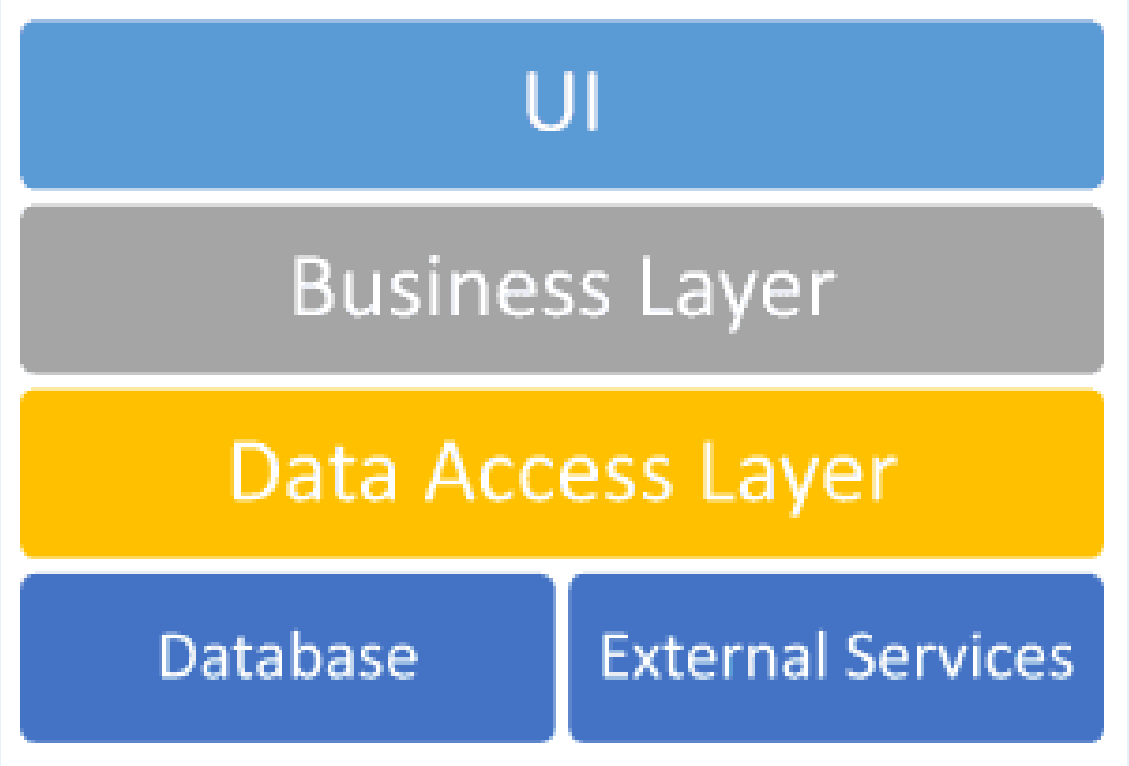
A single distributable – a monolith

With no additional organization, can devolve into a Big Ball of Mud

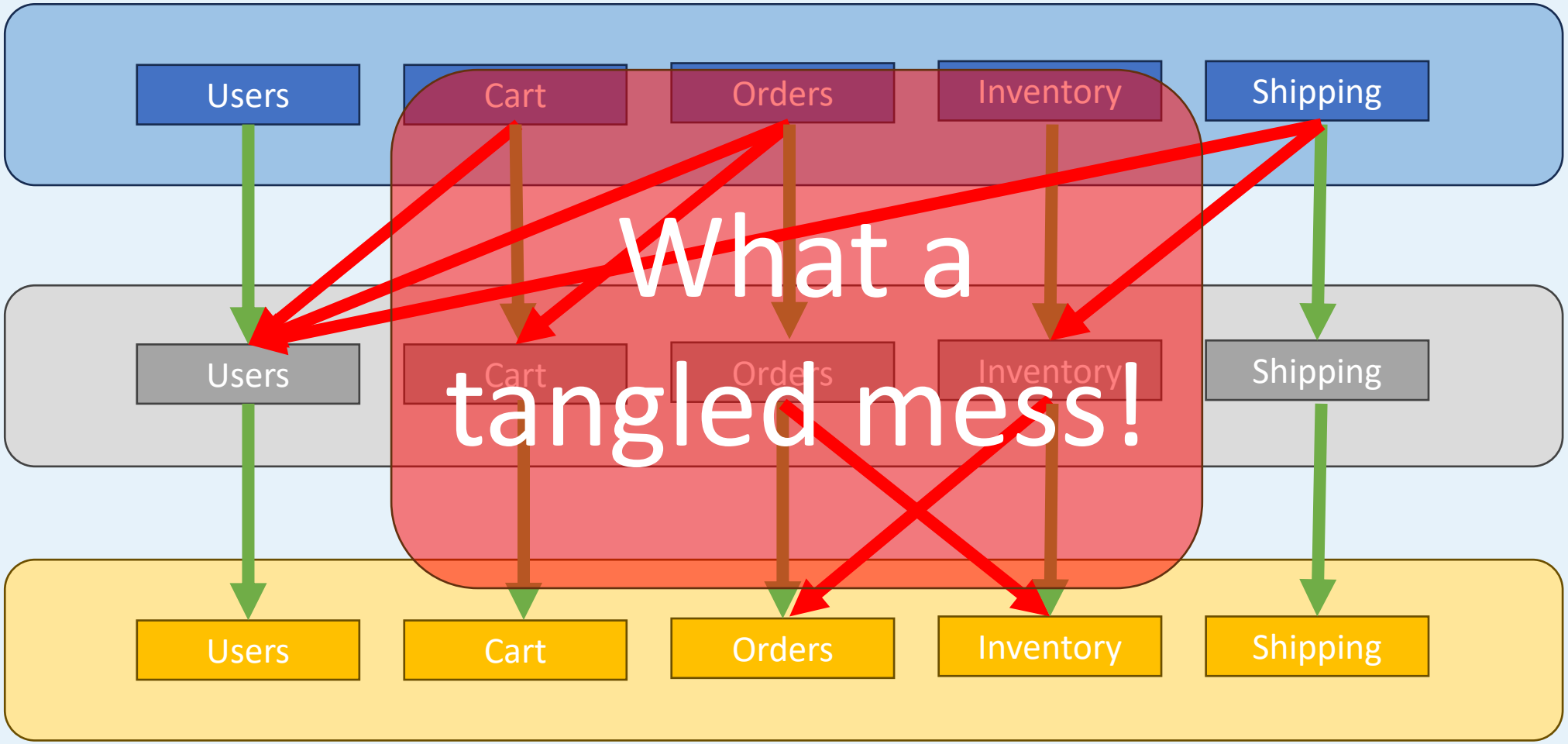


Layered Architecture

- Often added to improve design of monoliths
- Separates code by technical concerns
- Typically separated into different projects/assemblies

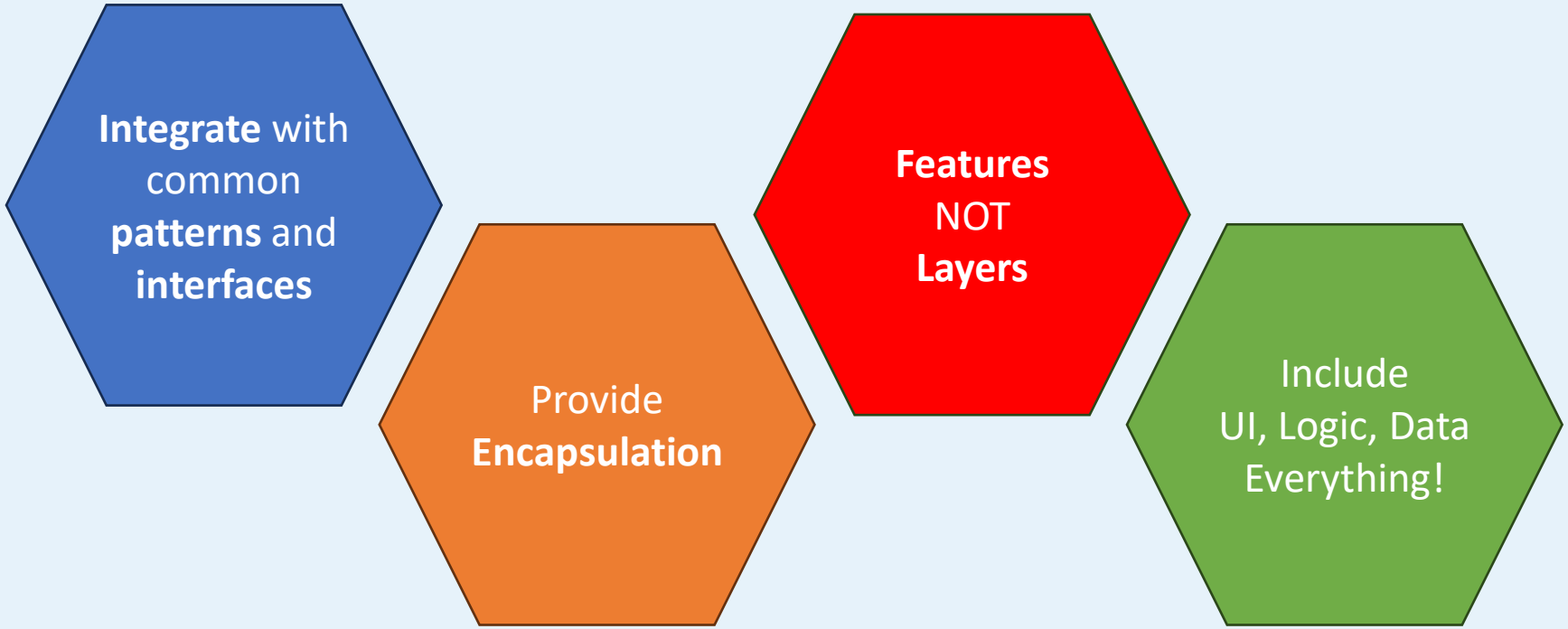


The Problem with Layered Architecture

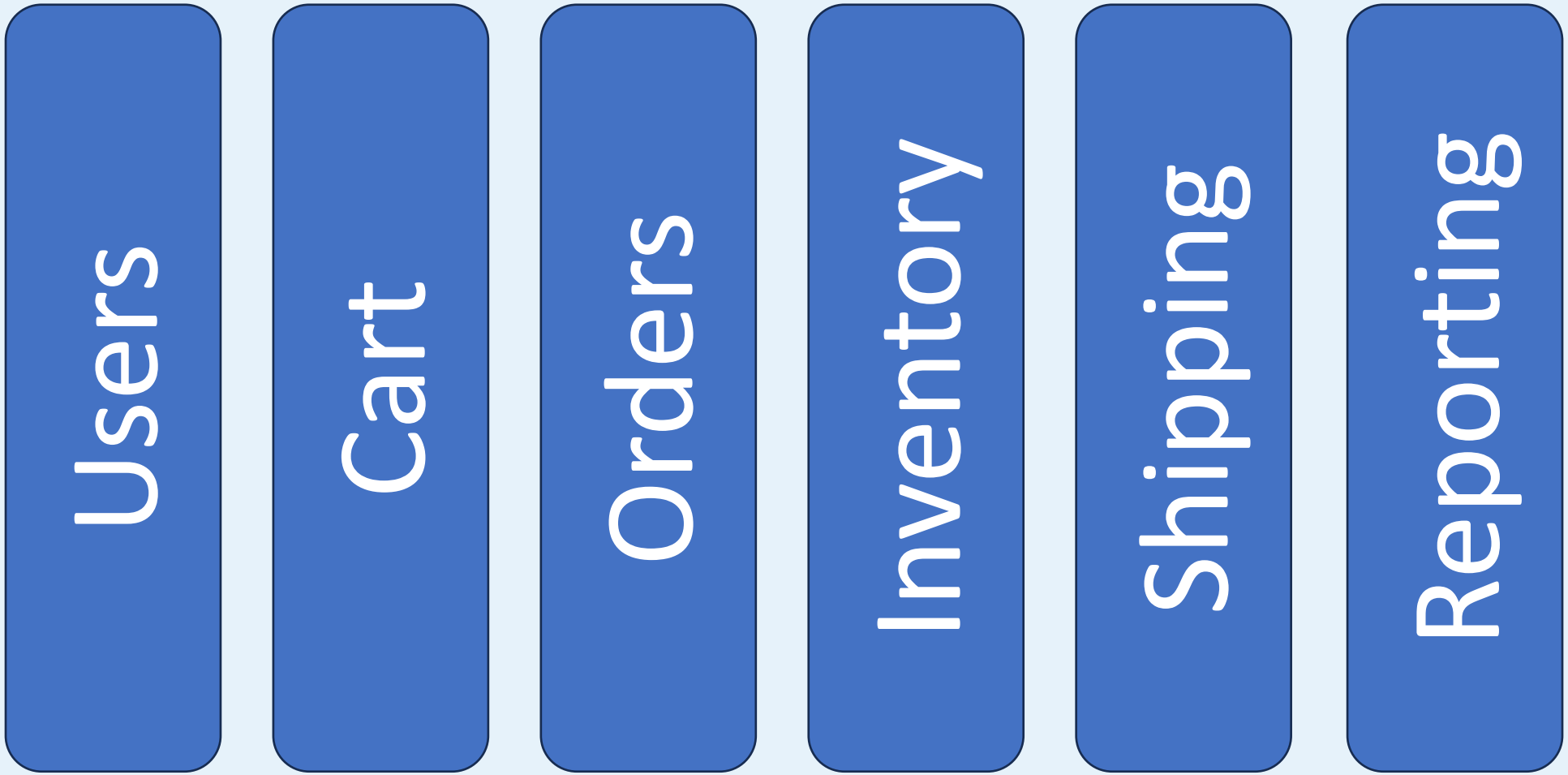


Spaghetti Code! (or at best lasagna)

(Real) Modularity to the Rescue!



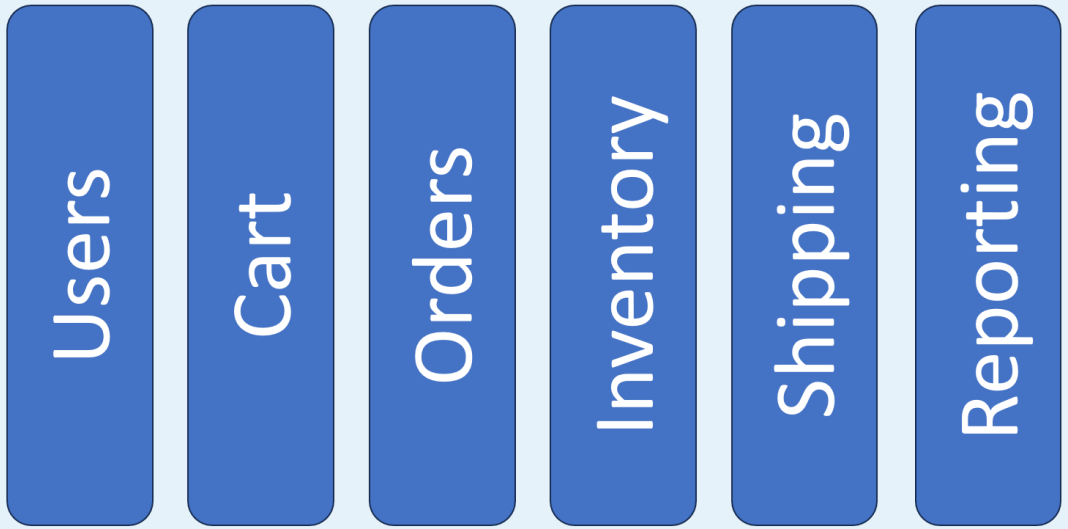
A Solution: Modules



So, Microservices?

- Each of these could be a microservice!

- Microservices are expensive
  - To build correctly (dependencies!)
  - To deploy (sooo many scripts)
  - To host (sooo many instances)
  - To monitor/diagnose (sooo many…)

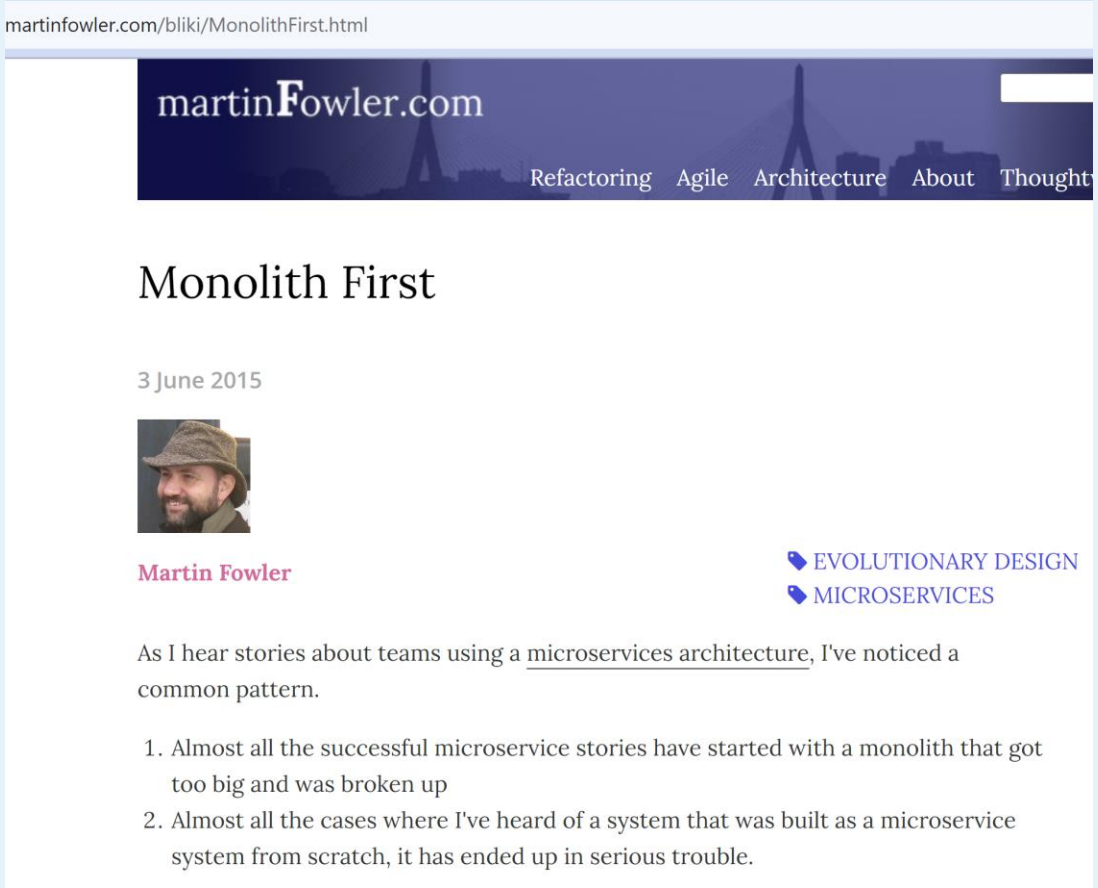


https://twitter.com/ardalis/status/1252777012932026372
(from 2020)

Don’t Listen to Me…

- Martin Fowler, 2015
- https://martinfowler.com/bliki/MonolithFirst.html

- “Almost all successful microservice stories started with a monolith that got too big”
- “Almost all the cases… a system … built as a microservice first … ended up in serious trouble.”



martinfowler.com/bliki/MonolithFirst.html

martinFowler.com

Refactoring Agile Architecture About Thought

Monolith First

3 June 2015

Martin Fowler

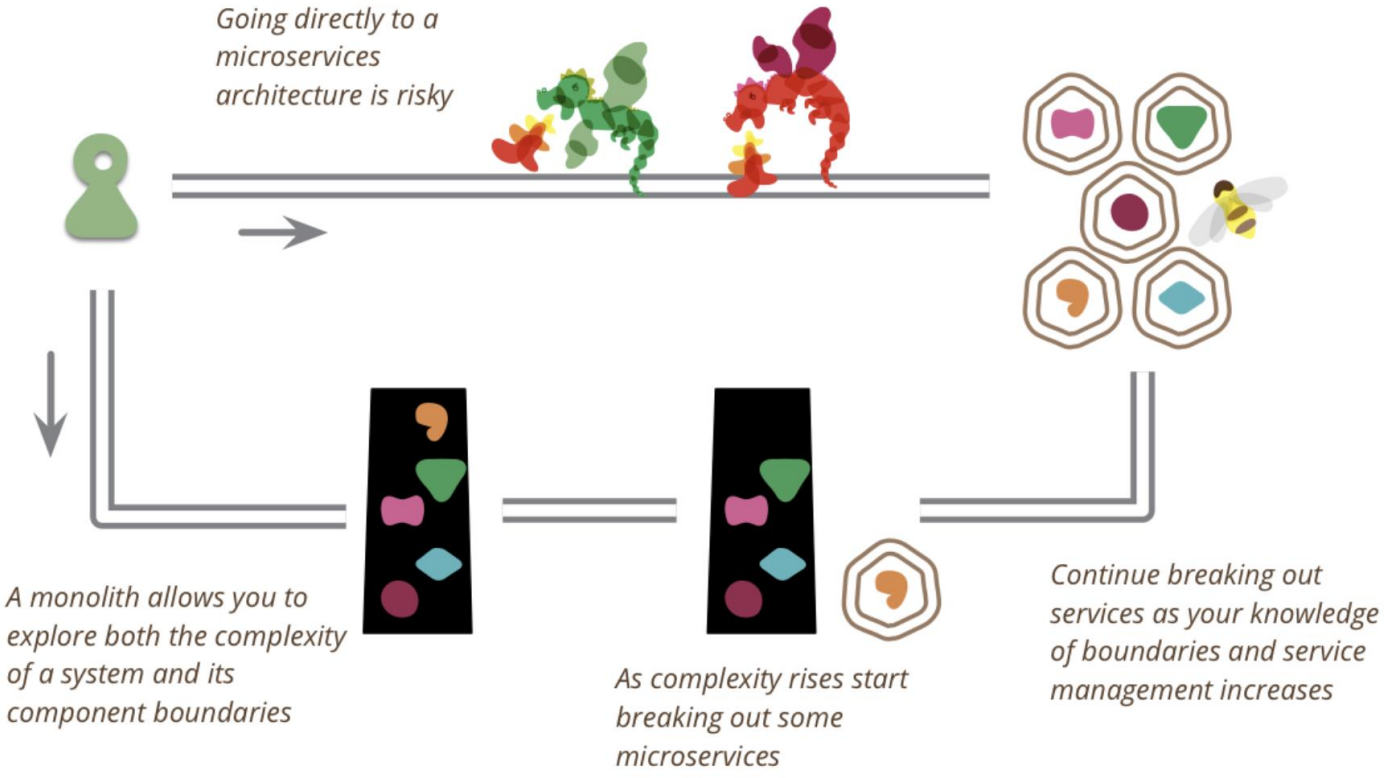
EVOLUTIONARY DESIGN
MICROSERVICES

As I hear stories about teams using a microservices architecture, I've noticed a common pattern.

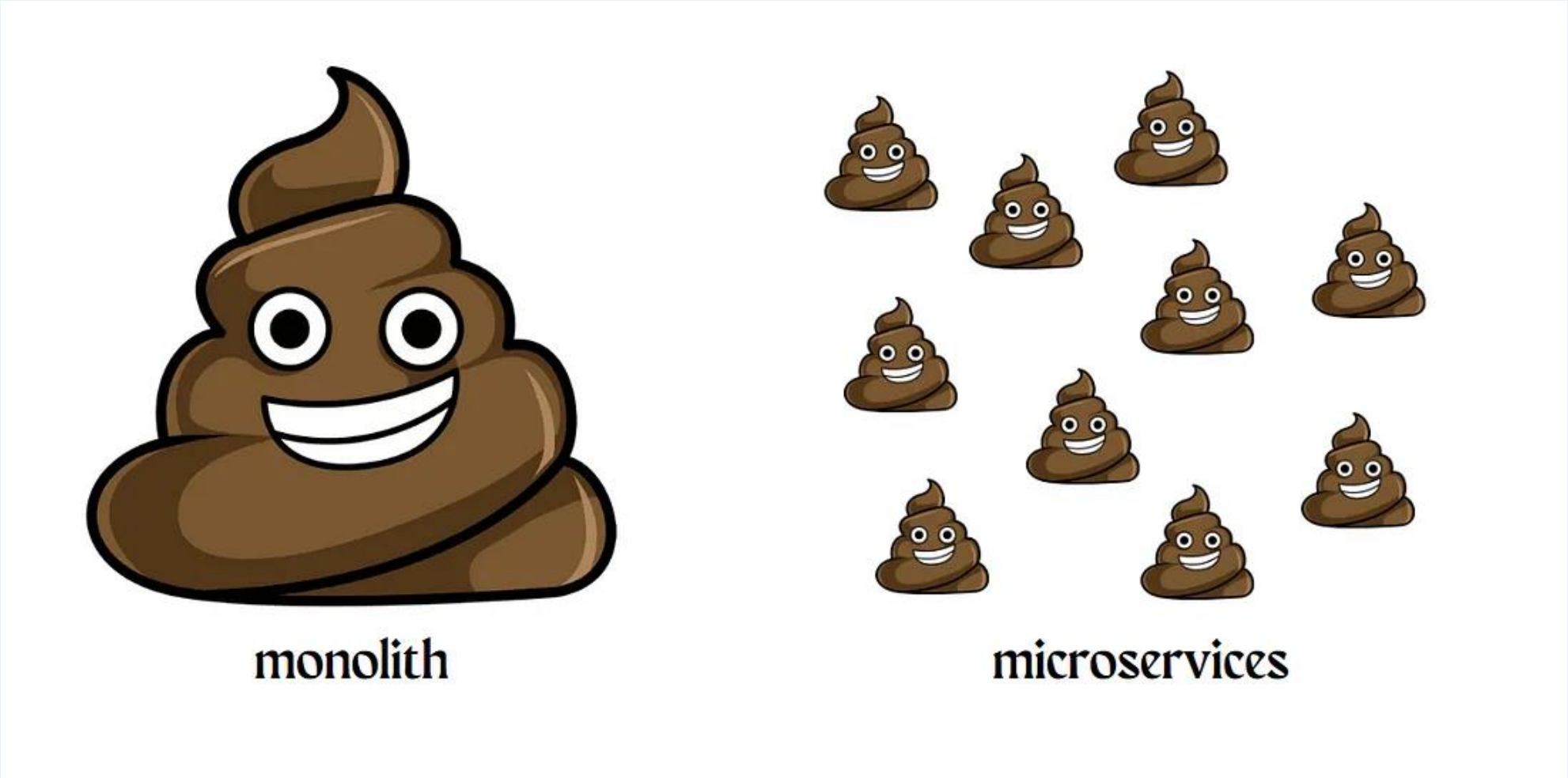
1. Almost all the successful microservice stories have started with a monolith that got too big and was broken up
2. Almost all the cases where I've heard of a system that was built as a microservice system from scratch, it has ended up in serious trouble.

Beware: Dragons

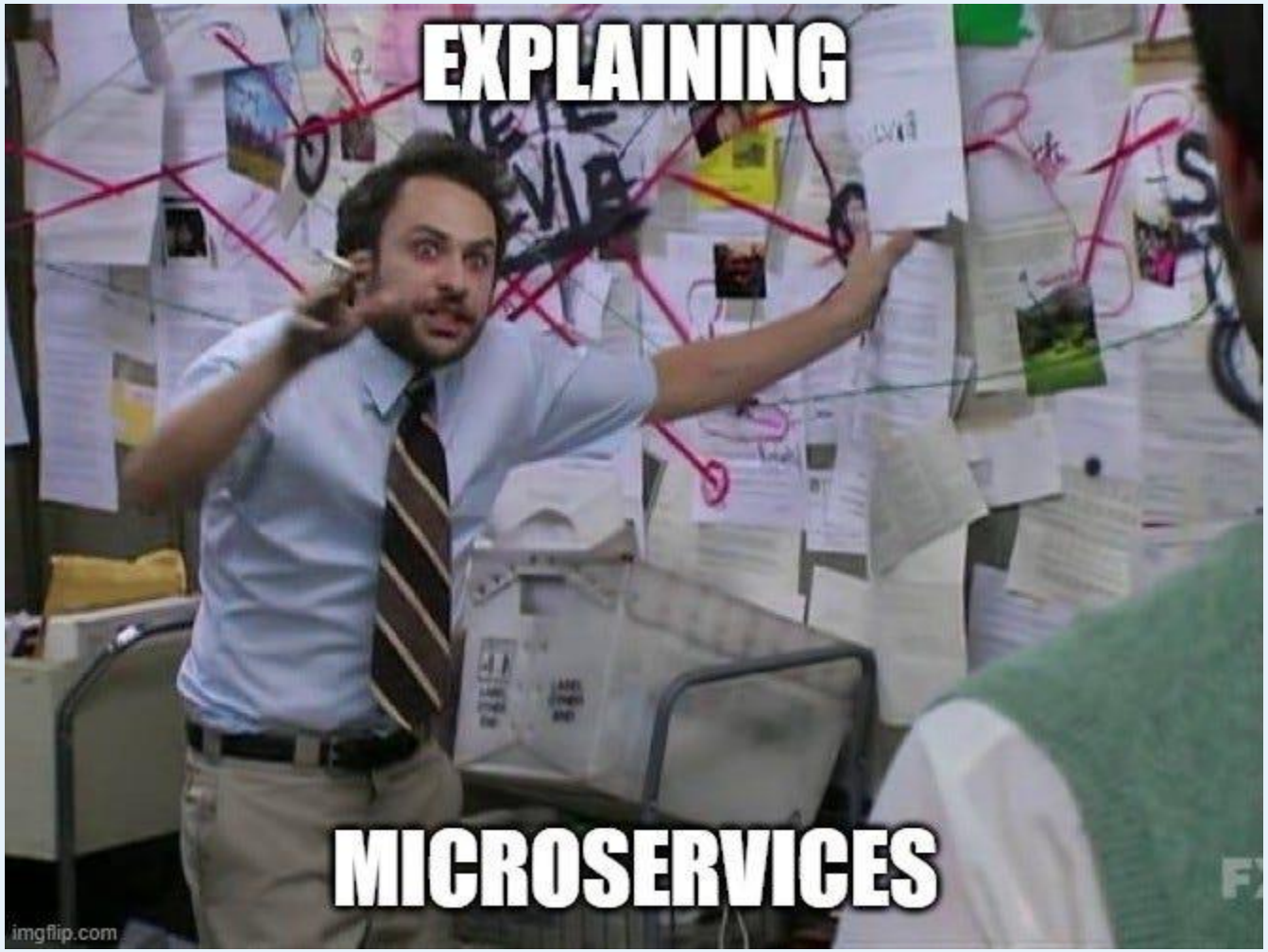
This pattern has led many of my colleagues to argue that **you shouldn't start a new project with microservices, even if you're sure your application will be big enough to make it worthwhile.** .



Beware: More Isn’t Always Better

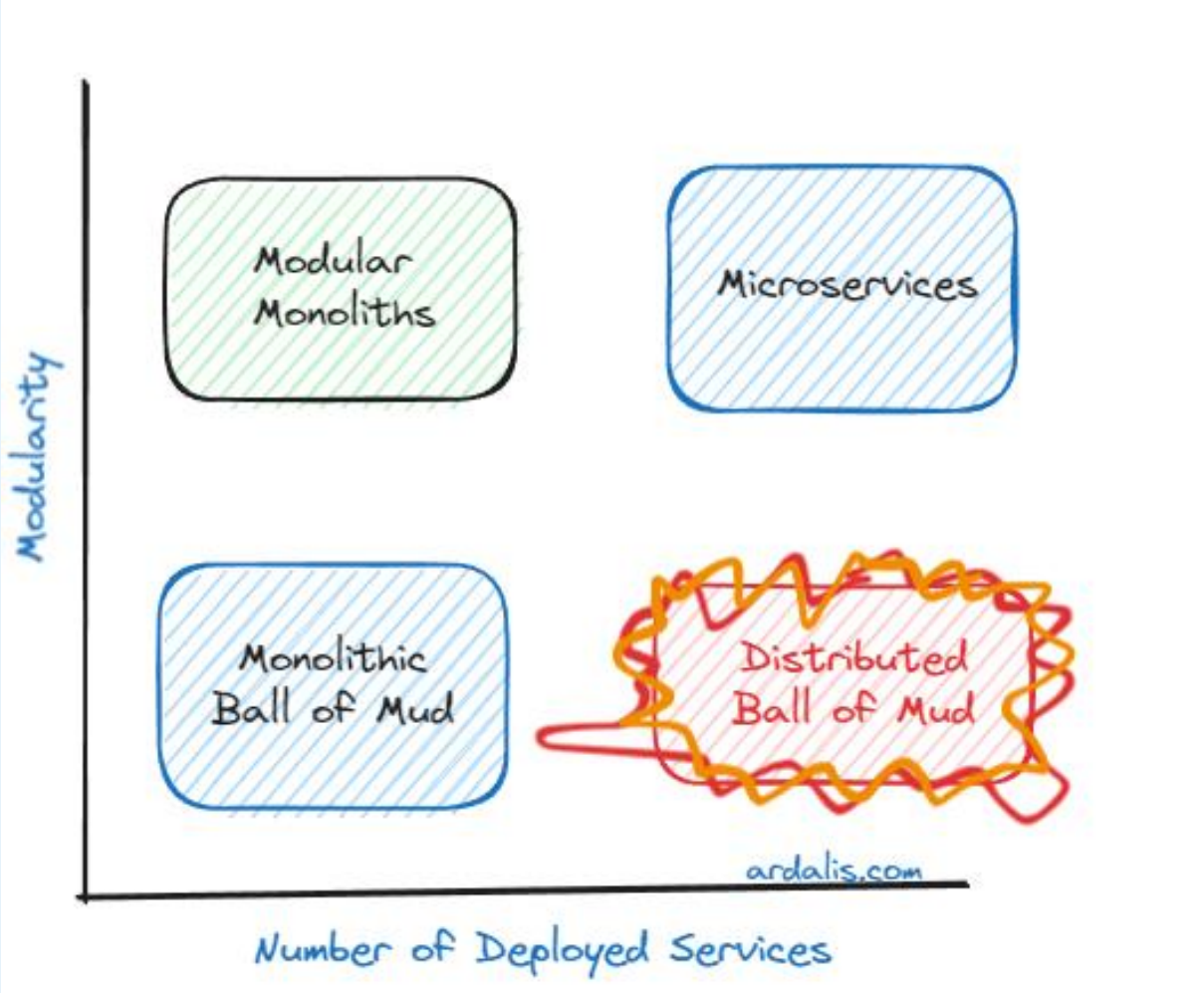


Beware: Complexity



Beware: Distributed Ball of Mud

(aka Distributed Monolith)

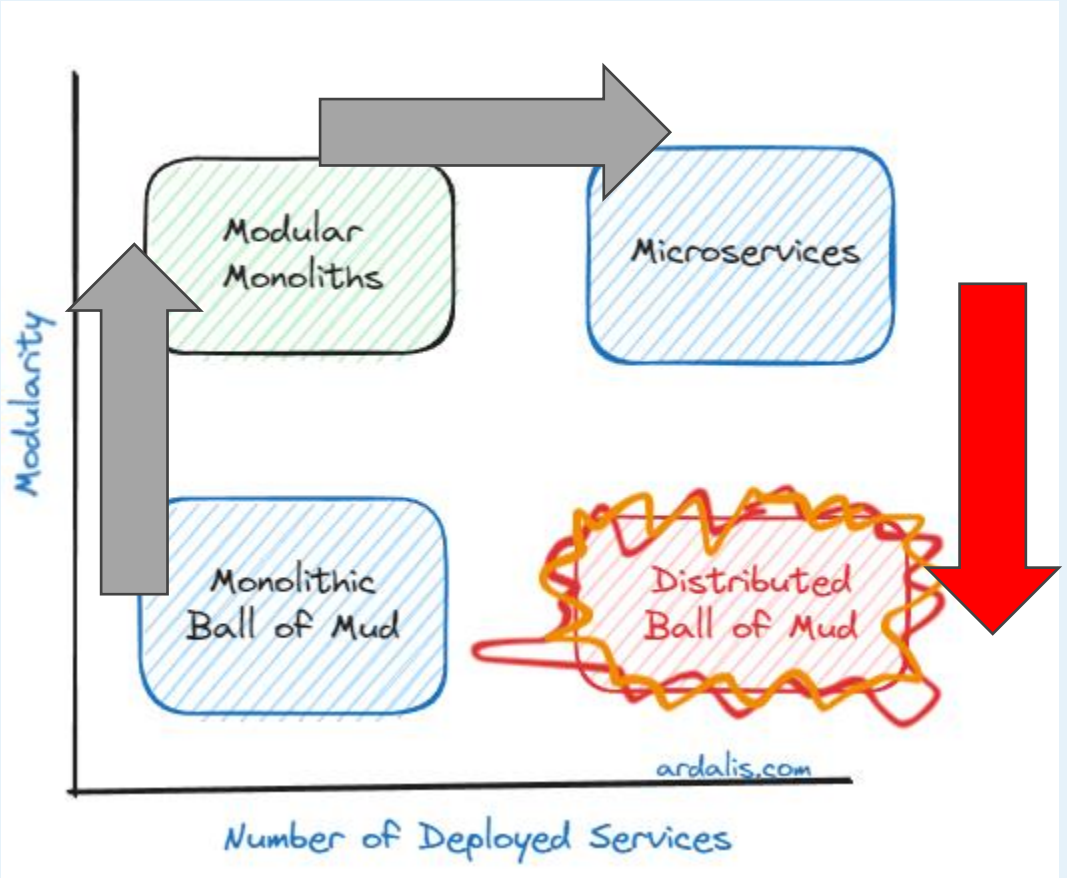


Moving Between Quadrants

1. Move from no modularity to modules (up)
2. Move from a single deployed app to a distributed app (right)

- BIG learning curve involved in BOTH choices

- Don’t Move to bottom right…



Move Along One Axis at a Time

- Move Up
  - Embrace Modularity
  - Learn how to build modular systems
- Move Right
  - Embrace Distributed Architecture
  - Learn how to build distributed systems
- Word of Advice
  - Don’t do both at the same time

Beware: Distributed Systems

- **“You can have a second computer once you’ve shown you know how to use the first one.”** -Paul Barham

- The first rule of distributed systems is don’t distribute your system until you have an observable reason to.

https://bravenewgeek.com/service-disoriented-architecture/

Why Not Build a Distributed System?

Fallacies of Distributed Computing

- The network is reliable
- Latency is zero
- Bandwidth is infinite
- The network is secure
- Topology doesn't change
- There is one administrator
- Transport cost is zero/free
- The network is homogeneous

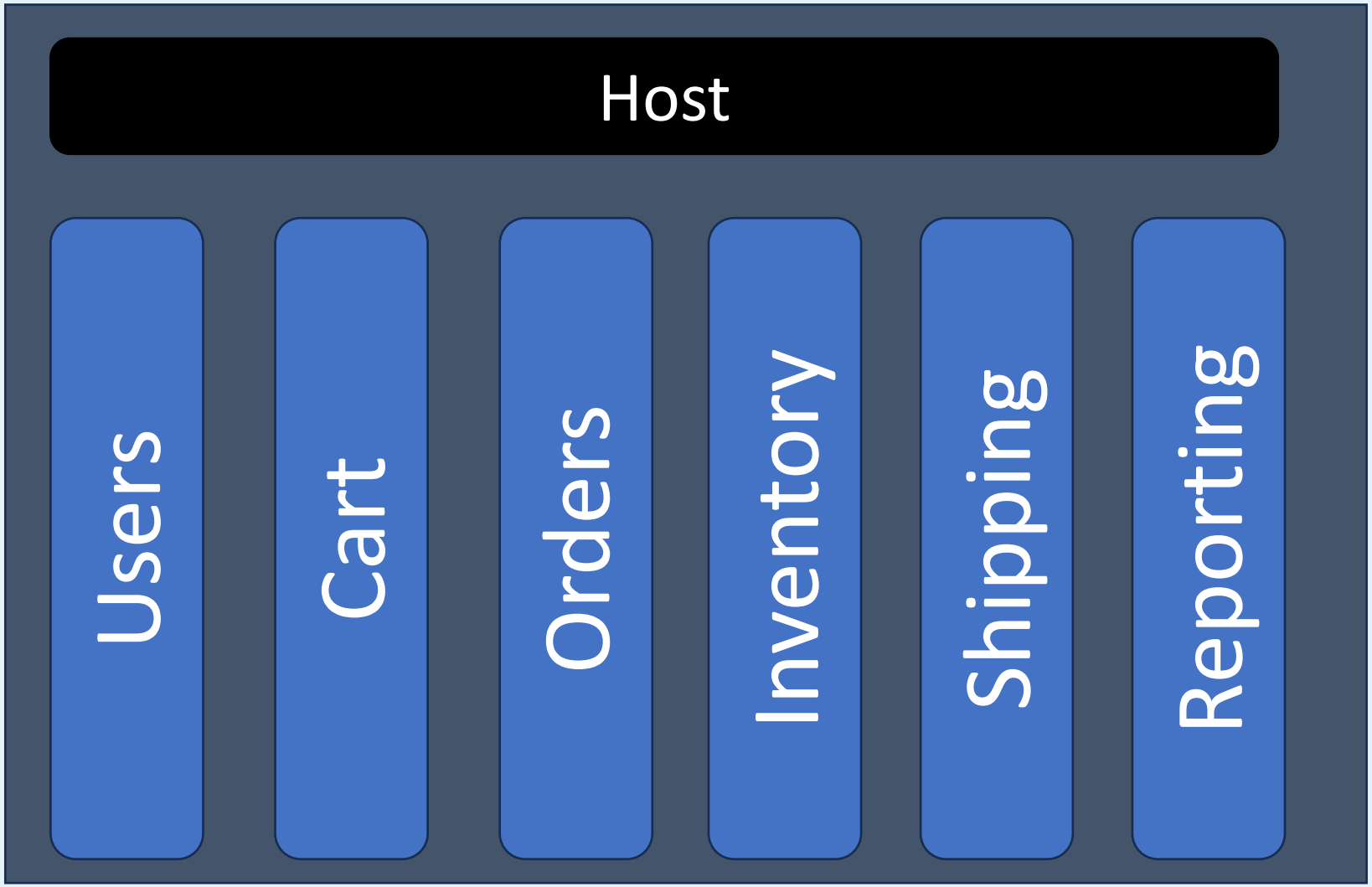
None of these are issues with monoliths!

Let’s not forget…

- Versioning between services
- Message contracts
- Distributed tracing and logging
- Kubernetes!
- Eventual consistency
- Distributed transactions
- Orchestration vs Choreography
- Accidentally creating a Distributed Ball of Mud

What if we could have modules in *one* monolithic application?

The Modular Monolith



Modules

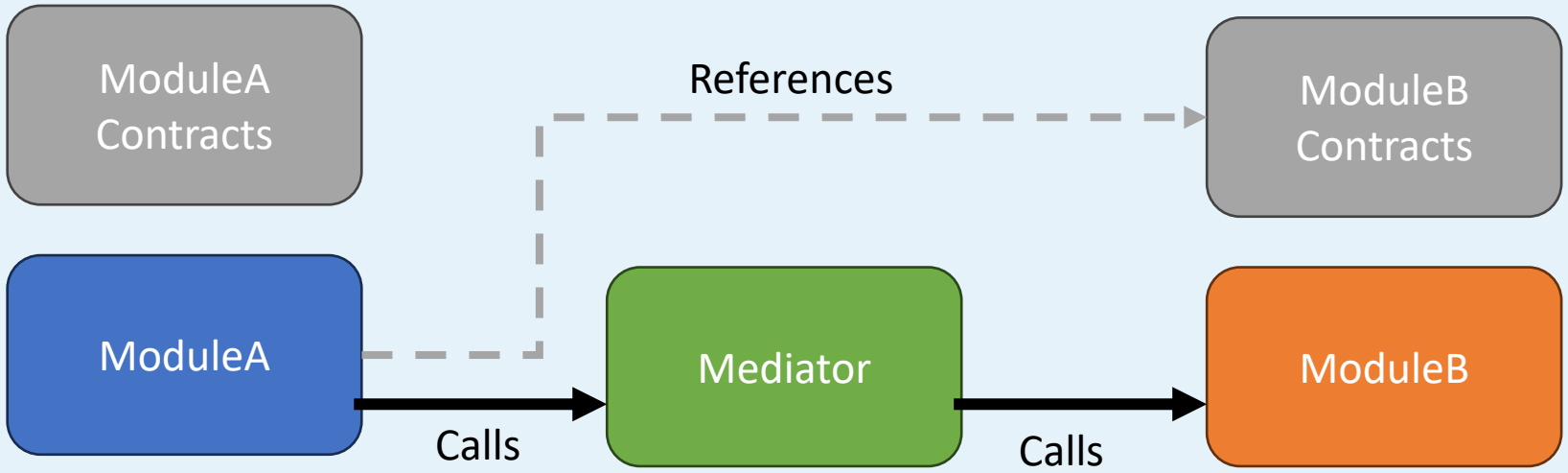
- Are logically separate and independent
- Can be worked on by separate teams
- Are feature-specific
- Represent a Bounded Context (DDD)
- Have their own persistence
- Can be organized as appropriate to their complexity

- Everything microservices should be!

How to Achieve Modularity in a Monolith?

- Internal types and methods
  - Maintain encapsulation
- Modules should own their own data
  - Can use separate schema in same database
- Modules expose public contracts
  - Interfaces
  - Data Types (DTOs) used by Interfaces
- Modules communicate indirectly
  - Via Mediator pattern
  - Via out-of-process message bus

Module Communication



Module Communication

From Cart Checkout to Order Processing

```
var createOrderCommand = new CreateOrderCommand(Guid.Parse(user.Id),
    request.shippingAddressId,
    request.billingAddressId,
    items);

// TODO: Consider replacing with a message-based approach for perf reasons
var result = await _mediator.Send(createOrderCommand); // synchronous

if (!result.IsSuccess)
```

Compare

Modular Monolith

- Logically Independent
- Easier to deploy
- Cheaper to host
- Easier to troubleshoot
- Less scalable
- Less available
- Single platform

Microservices

- Logically/Physically Independent
- Harder to deploy
- More expensive to host
- Harder to troubleshoot
- More scalable
- More available
- Multi-platform option

Demo

Summary

- Monoliths often lack sufficient logical separation
  - Can turn into Big Ball of Mud and Spaghetti Code
- Layers help but often are insufficient
- Microservices are more difficult and more expensive
  - And often devolve into Distributed Balls of Mud

- Modular monolith can offer a “Goldilocks” solution

Learn More

- Modular Monoliths: From Zero to Hero Course (Bundle)
  - https://dometrain.com/bundle/from-zero-to-hero-modular-monoliths-in-dotnet/

- Contact me at NimblePros.com for training and application/architecture consulting services