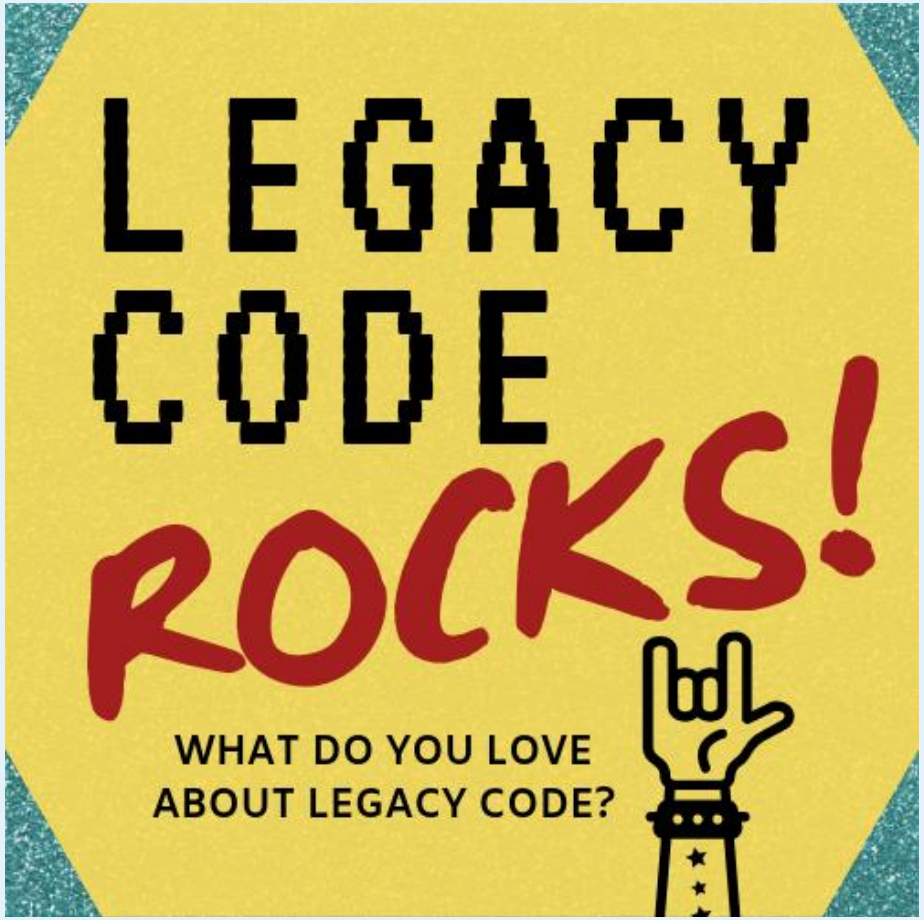
# From **Legacy** Monolith to Microservices via EventStorming

NimblePros

*Better software development. Delivered.*

# Thank you, MenderCon 2024 Organizers!



# Thank You, MenderCon 2024 Sponsors!

CORGIBYTES

old code ⚙ new tricks

DEEP ROOTS

Refactoring with Depth

PLANET ARGON

If you are doing
**simple** CRUD apps
or **simple** business systems,
these solutions are overkill.

# Key Terms

- **Legacy** Monolith
- Microservices
- Modulith
- EventStorming

eSHOP OnWeb

demouser@microsoft.com

0

BRAND
All

TYPE
All

Previous

Showing 10 of 12 products - Page 1 - 2

Next

[ ADD TO BASKET ]

ROSLYN RED SHEET

$ 8.50

[ ADD TO BASKET ]

.NET FOUNDATION SWEATSHIRT

$ 12.00

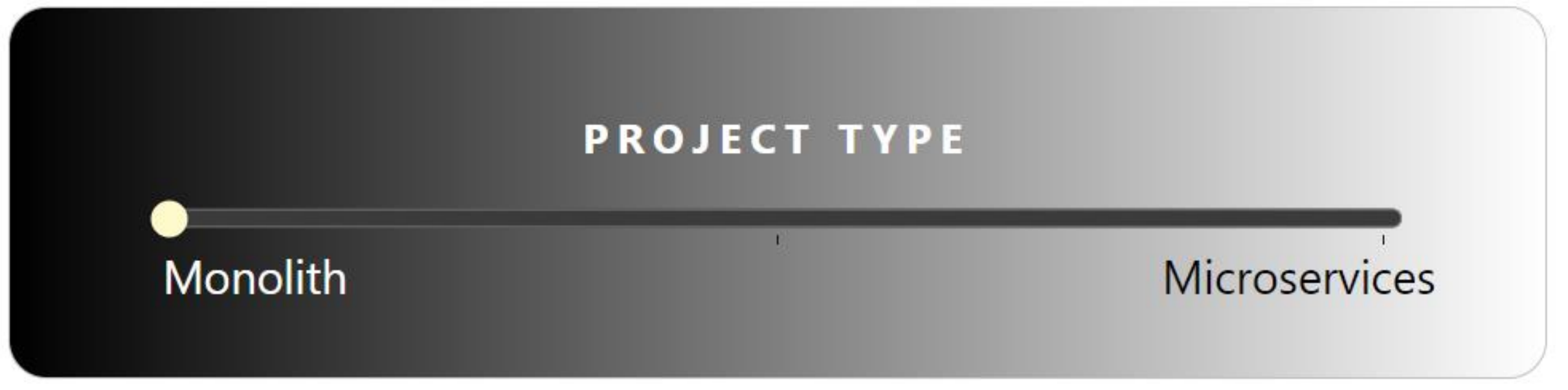
[ ADD TO BASKET ]

PRISM WHITE T-SHIRT

$ 12.00

# Common Characteristics of a **Legacy** Monolith

- Monorepo
- Tightly coupled code
- Scalability is iffy
- Slow release cycles
- Difficult to update
- Big Ball of Mud
- Spaghetti Code


PROJECT TYPE
Monolith
Microservices

MICROSERVICES! MICROSERVICES!
MICROSERVICES!

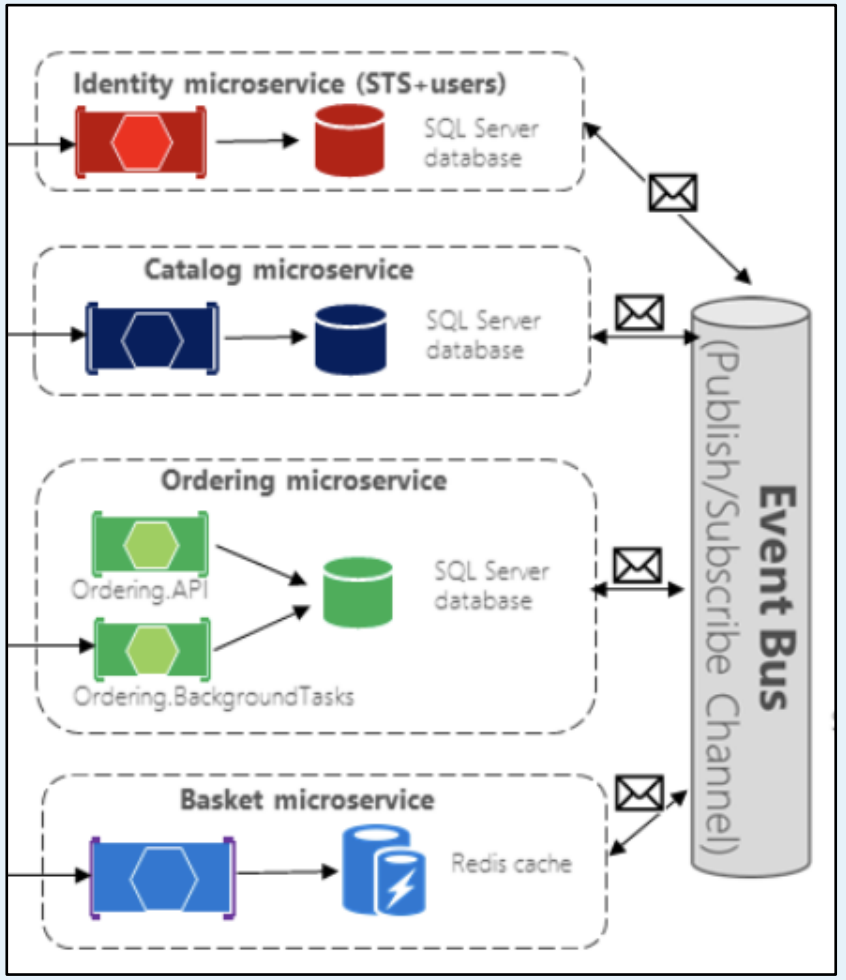
# Characteristics of Microservices

In short, the microservice architectural style 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. These services are **built around business capabilities** and **independently deployable** by fully automated deployment machinery. There is a **bare minimum of centralized management** of these services, which **may be written in different programming languages** and use **different data storage technologies**.

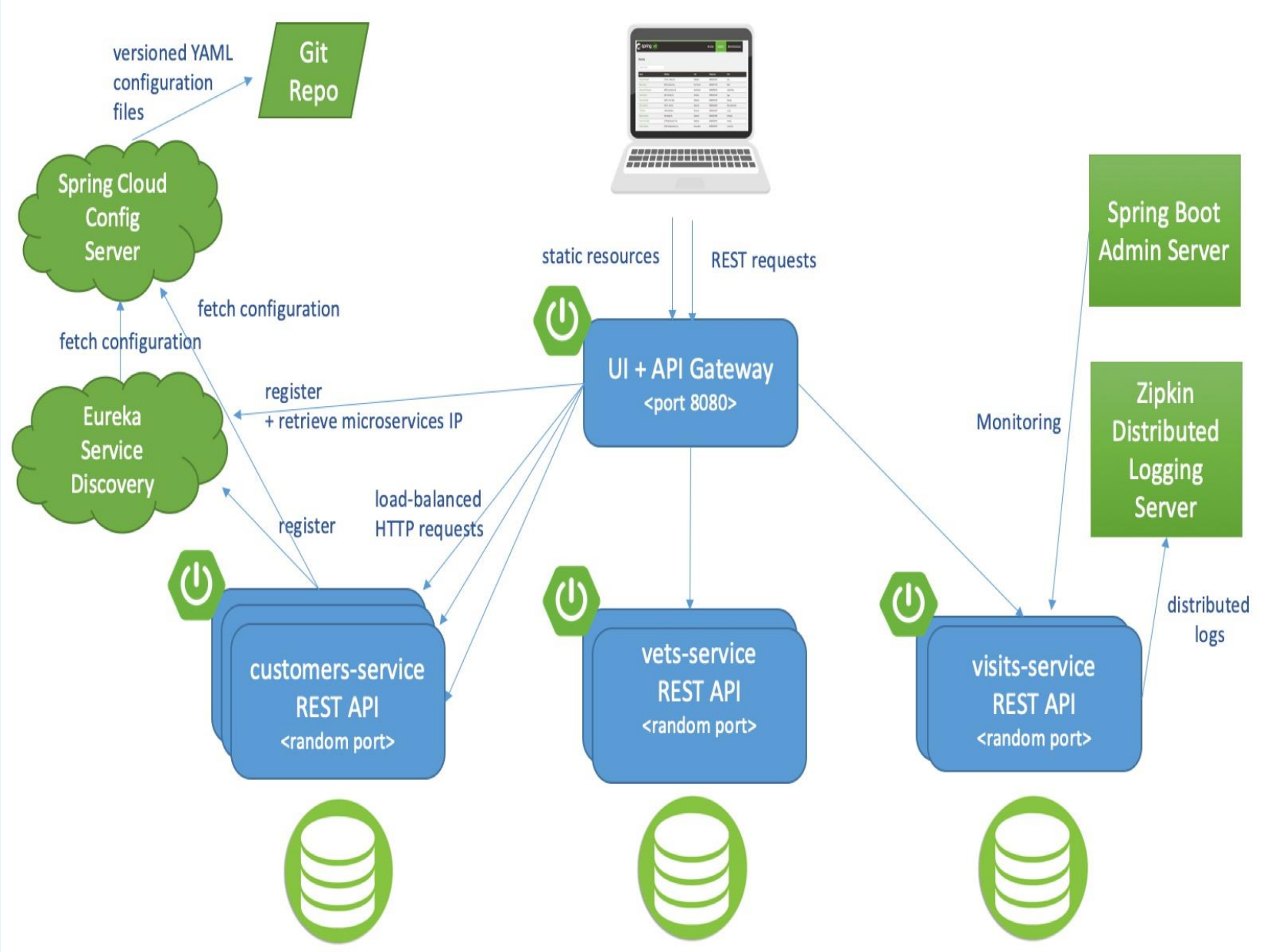
- James Lewis and Martin Fowler, *Microservices* (2014)

# Characteristics of Microservices

- Small, independent, loosely coupled services
- Autonomous services
- Each service manages its own data
- Polyglot friendly
- Scalable
- Resilient – one failure doesn't bring them all down
- DevOps friendly




Microservices from [eShopOnContainers]



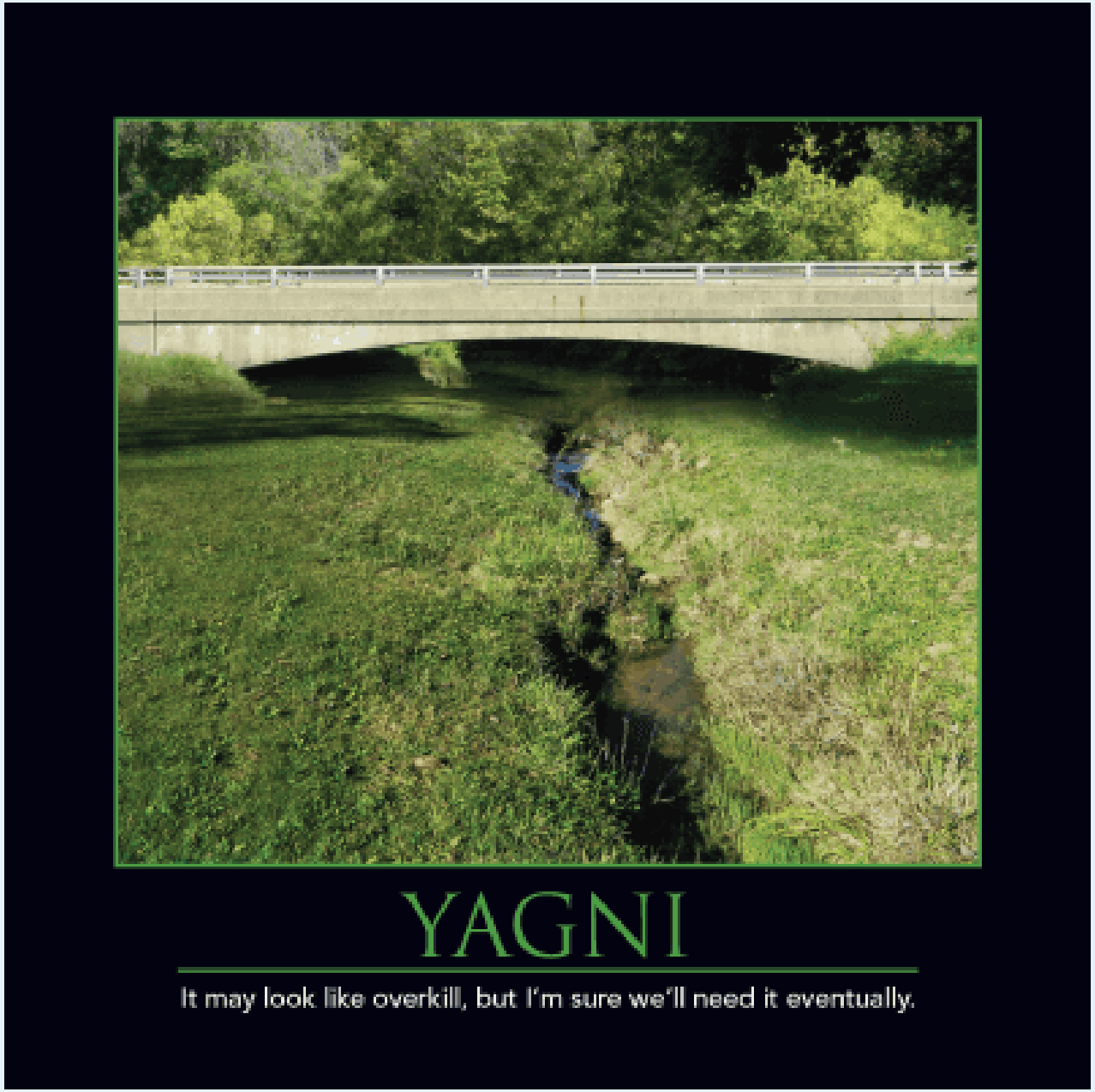

[Spring Petclinic - Microservices]

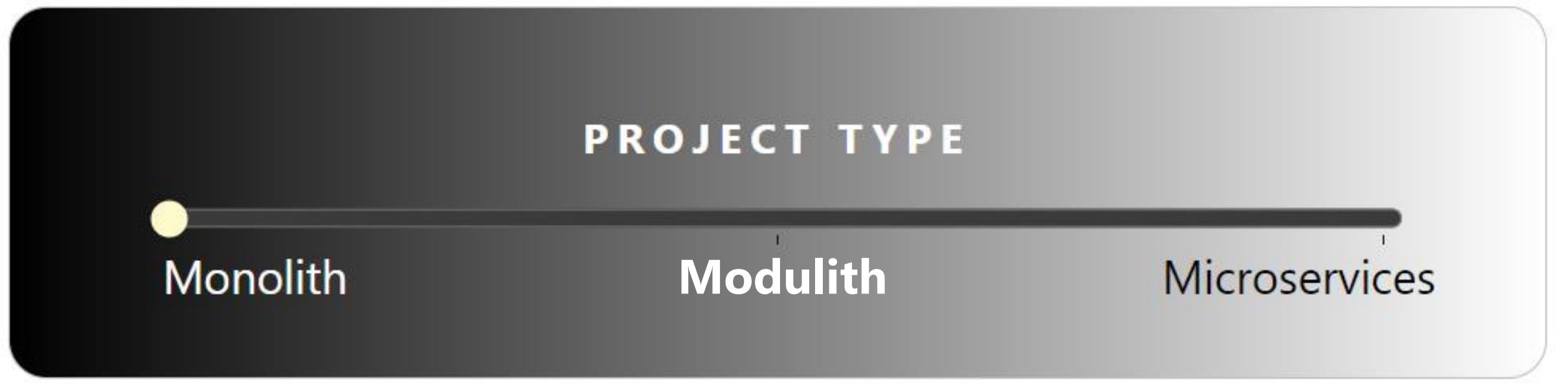
# Monolith || Microservices

COURAGE
Don't refactor complex systems without it.

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.

Martin Fowler,
*Monolith First (2015)*


YAGNI
It may look like overkill, but I'm sure we'll need it eventually.


PROJECT TYPE
Monolith
Modulith
Microservices

# Modality

# Characteristics of a Modular Monolith

- Single code base
- Module-based organization
- Separation of concerns
- Shared services
- Scalable
- Flexible
- Interaction between modules happens via public APIs or messaging
- Intermediate step between Monolith and Microservices

# Trade-offs for Modular Interactions in Monolith

| Synchronous | Asynchronous |
| --- | --- |
| • Communicate via APIs<br>• Pro: Easy to implement<br>• Pro: Reduced operational complexity<br>• Con: Strong coupling | • Communicate via messaging<br>• Temporally decoupled but…<br>  • Not guaranteed to be loosely coupled<br>• Con: More complex implementation |

# But wait!

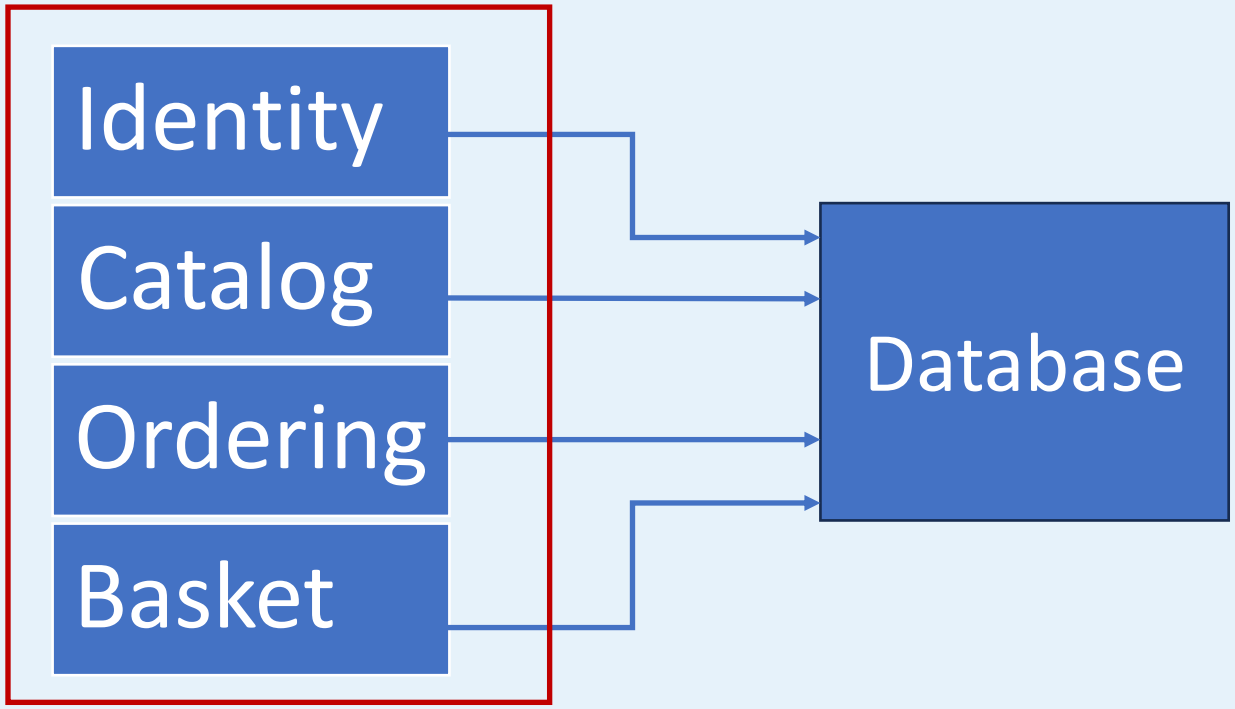
Isn’t this still just a monolith?

Won’t we have the problems mentioned earlier?

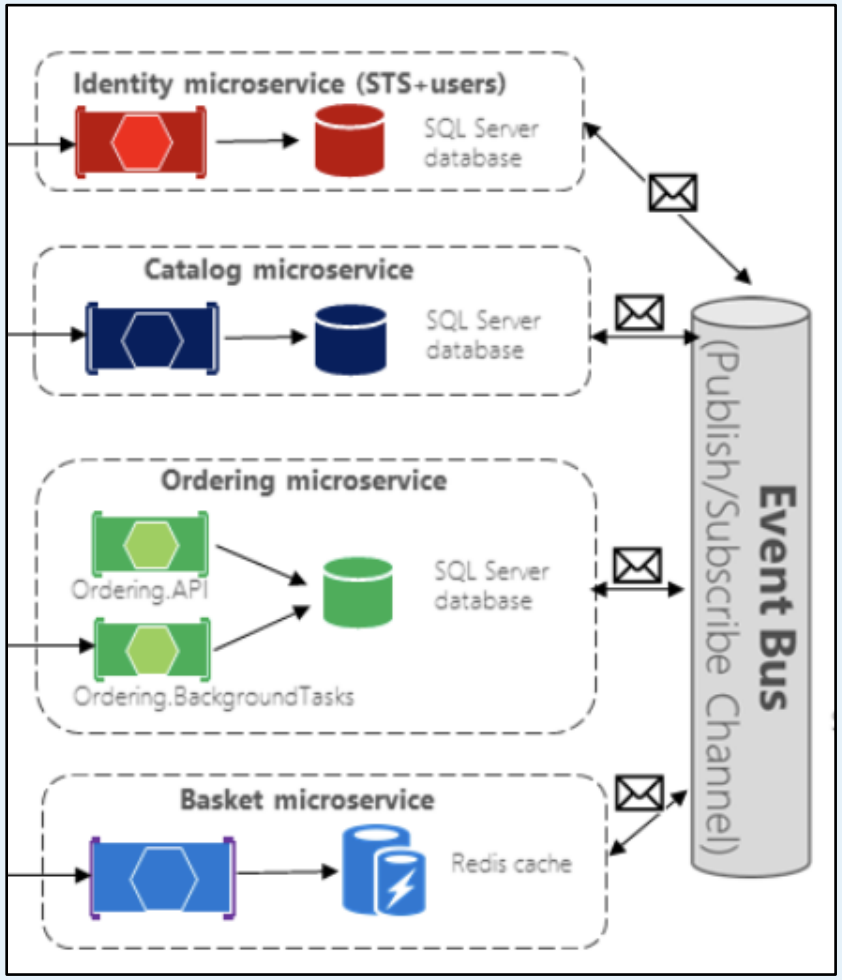
# When to Use Modular Monoliths vs Microservices

| Modular Monoliths | Microservices |
| --- | --- |
| • Greenfield, early stage<br>• Small to medium sized apps<br>• Low to medium scaling<br>• Low complexity business solutions<br>• Team expertise in a tech stack | • Large, complex business systems<br>• Scalability<br>• Polyglot support<br>• Fault isolation<br>• Multiple independent development teams |

# Legacy Monolith



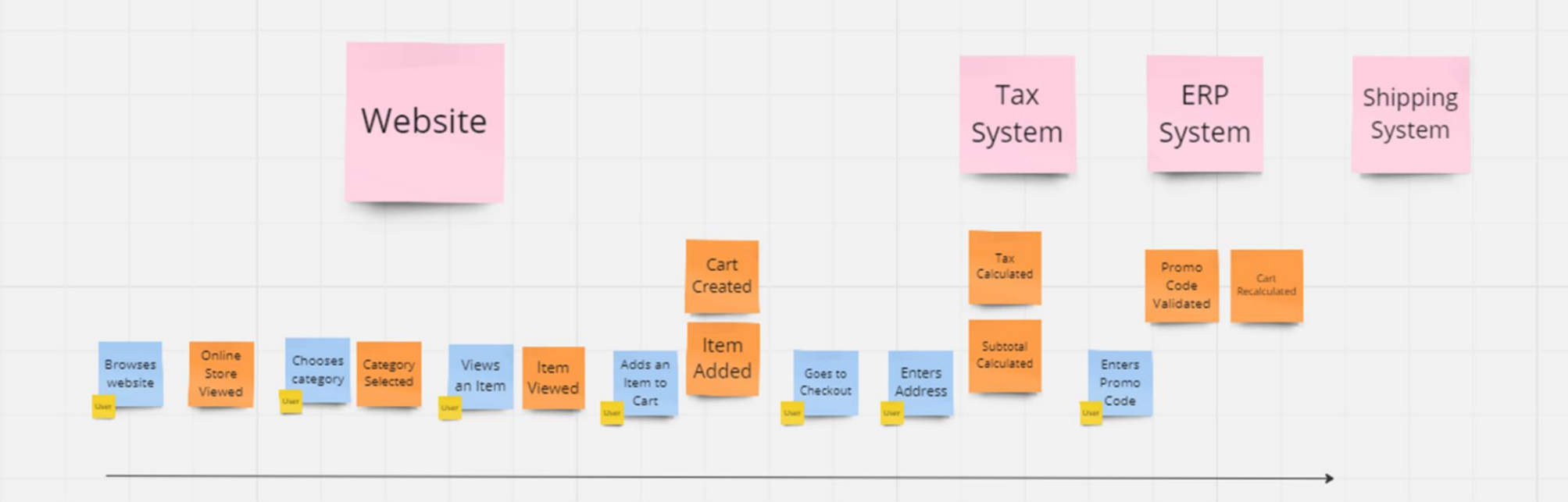
How do we grow and improve from this legacy monolith?


Identity microservice (STS+users)
SQL Server database
Catalog microservice
SQL Server database
Ordering microservice
Ordering.API
Ordering.BackgroundTasks
SQL Server database
Basket microservice
Redis cache
Event Bus
(Publish/Subscribe Channel)

But it’s a mess now! How do we go from disaster to better?

# EVENT STORMING!

# What is EventStorming?



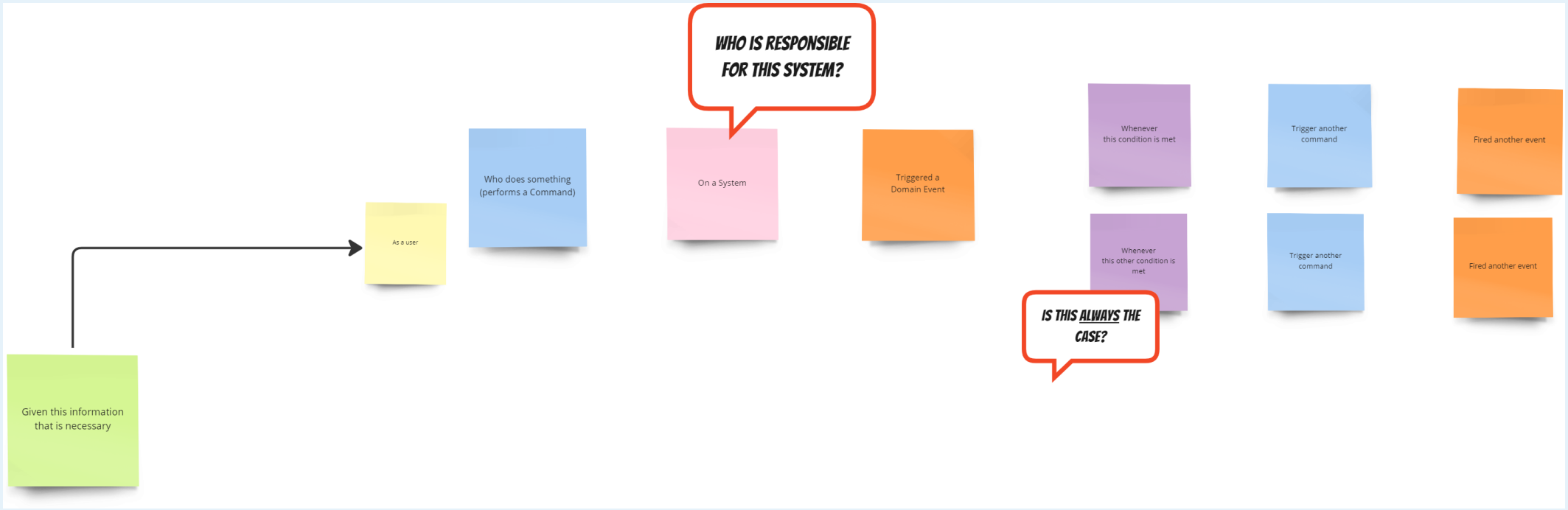

Collaborative workshop-based experience to gain a shared understanding of a **complex business system**

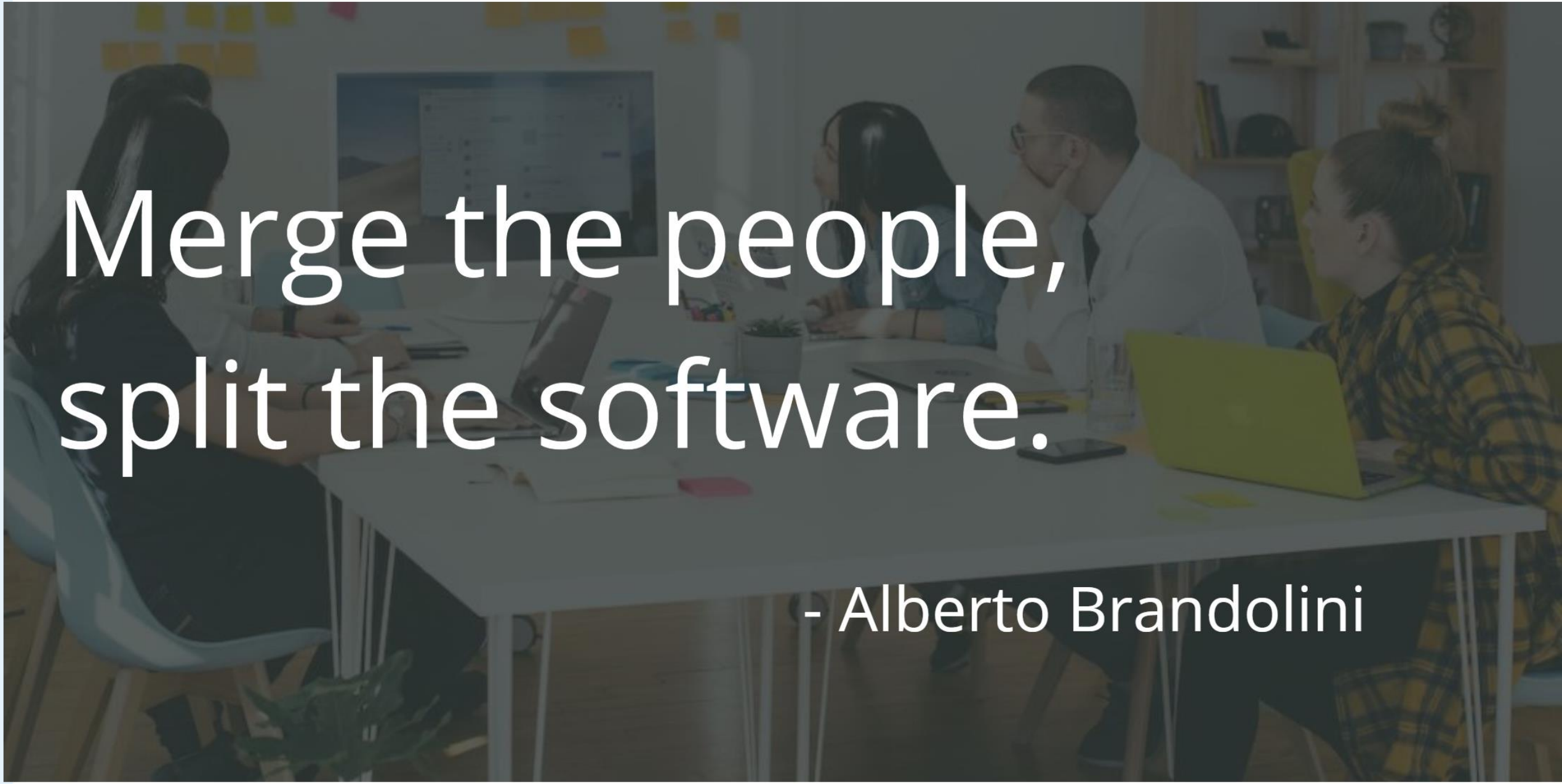
# Key People in EventStorming

- Facilitator
- People with ***questions***
- People with ***answers***
  - Local experts, masters of their silo

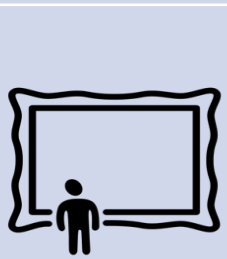
# EventStorming Grammar

- **<u>Conversations</u>** with **sticky notes**
- Eventually structured to follow a particular **grammar**



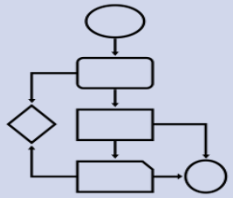

Merge the people,
split the software.
- Alberto Brandolini

# Types of EventStorming



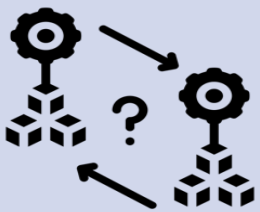
## Big Picture EventStorming

- Goal: Generate shared understanding
- Used for discovery



## EventStorming for Process Modeling

- Goal: Address the Hotspots
- Used for exploring a process
- Typically limited to a single end-to-end process



## EventStorming for System Design

- Goal: Evaluate a system and propose a solution
- Design a solution
- Be aware of alternatives
- Hide unnecessary complexity from the users

## EventStorming for People Experience

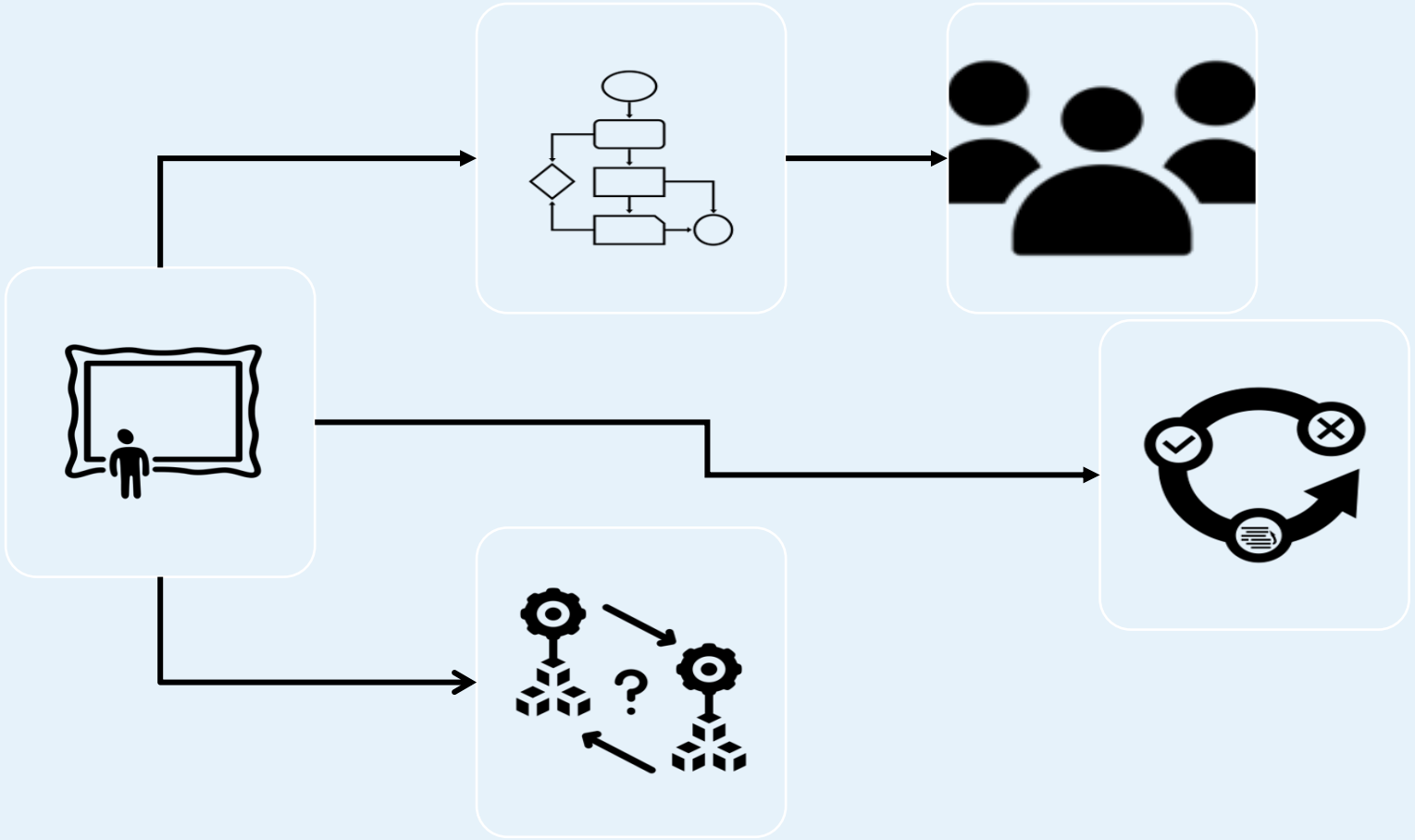
- Goal: Understand customer/user/ persona interactions and experiences

## EventStorming for Refactoring

- Given an existing solution, use EventStorming to identify what is available and potential refactor points

# Common Flow of EventStorming

- How do the EventStorming sessions flow into each other?

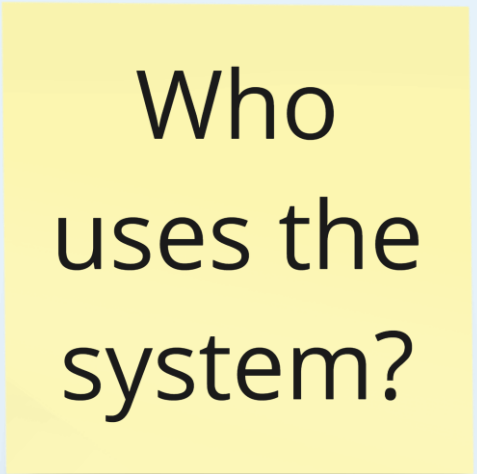


# What if all you have is a Legacy Monolith?

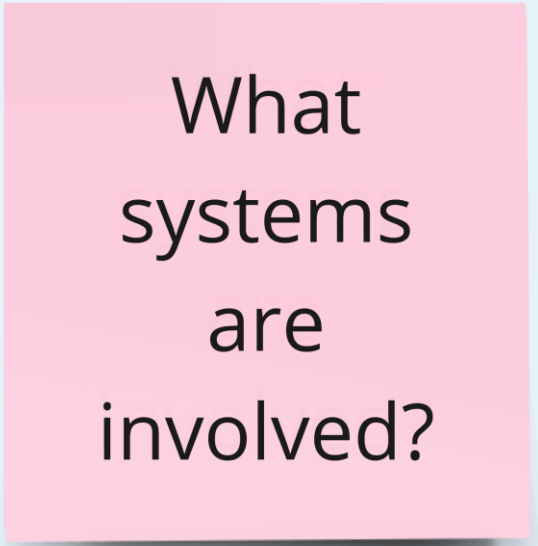
- While you can build upon discussions in other EventStorming sessions, they are not prerequisites for refactoring.
- **No prior EventStorming session needed** for refactoring discussions

# EventStorming for Refactoring – Basic Sticky Notes

Domain Events

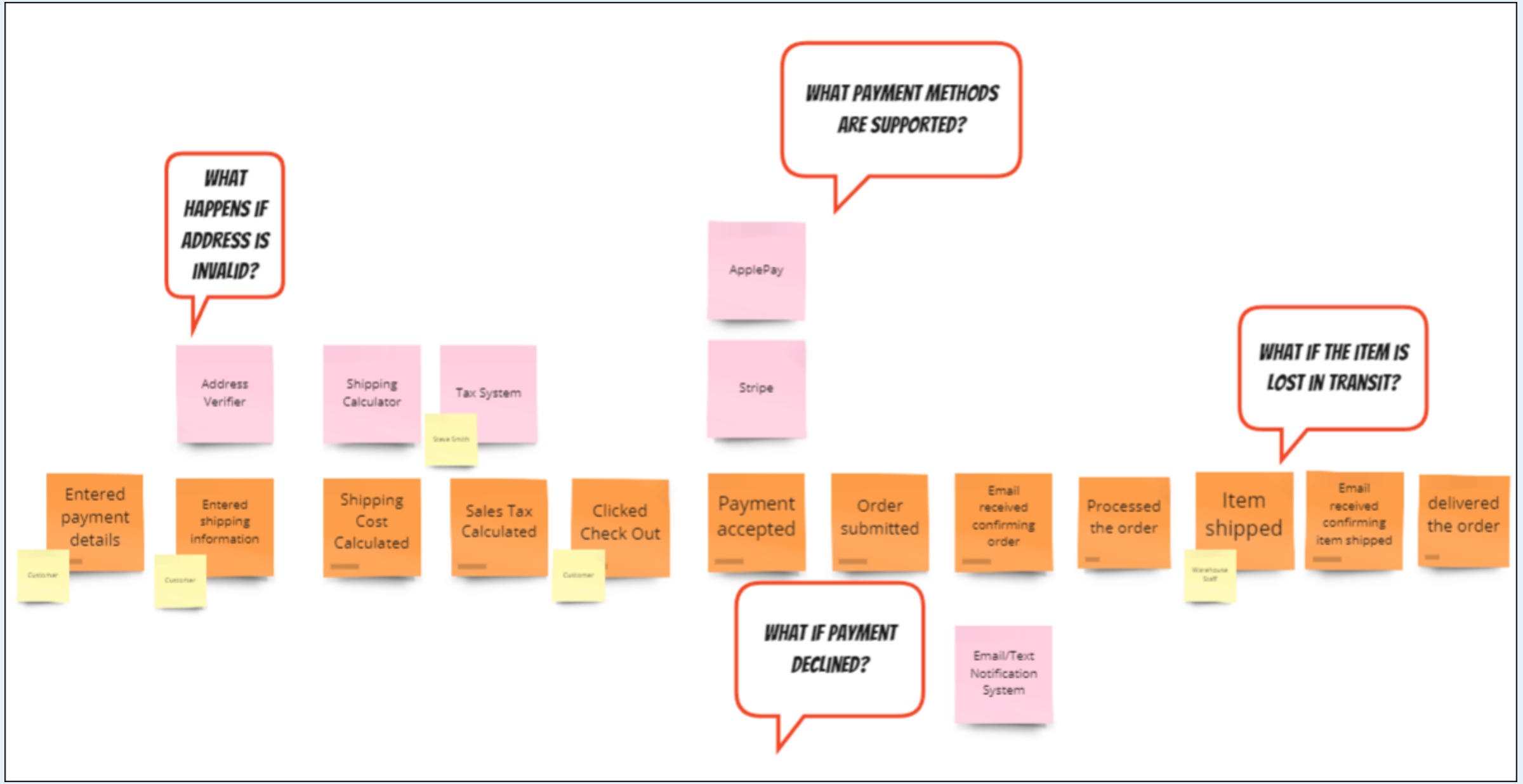



Users/Roles/Personas




Systems

# Identify Hotspots


WHAT PAYMENT METHODS ARE SUPPORTED?
WHAT HAPPENS IF ADDRESS IS INVALID?
WHAT IF THE ITEM IS LOST IN TRANSIT?
WHAT IF PAYMENT DECLINED?
ApplePay
Stripe
Address Verifier
Shipping Calculator
Tax System
Steve Smith
Entered payment details
Customer
Entered shipping information
Customer
Shipping Cost Calculated
Sales Tax Calculated
Clicked Check Out
Customer
Payment accepted
Order submitted
Email received confirming order
Email/Text Notification System
Processed the order
Item shipped
Warehouse Staff
Email received confirming item shipped
delivered the order

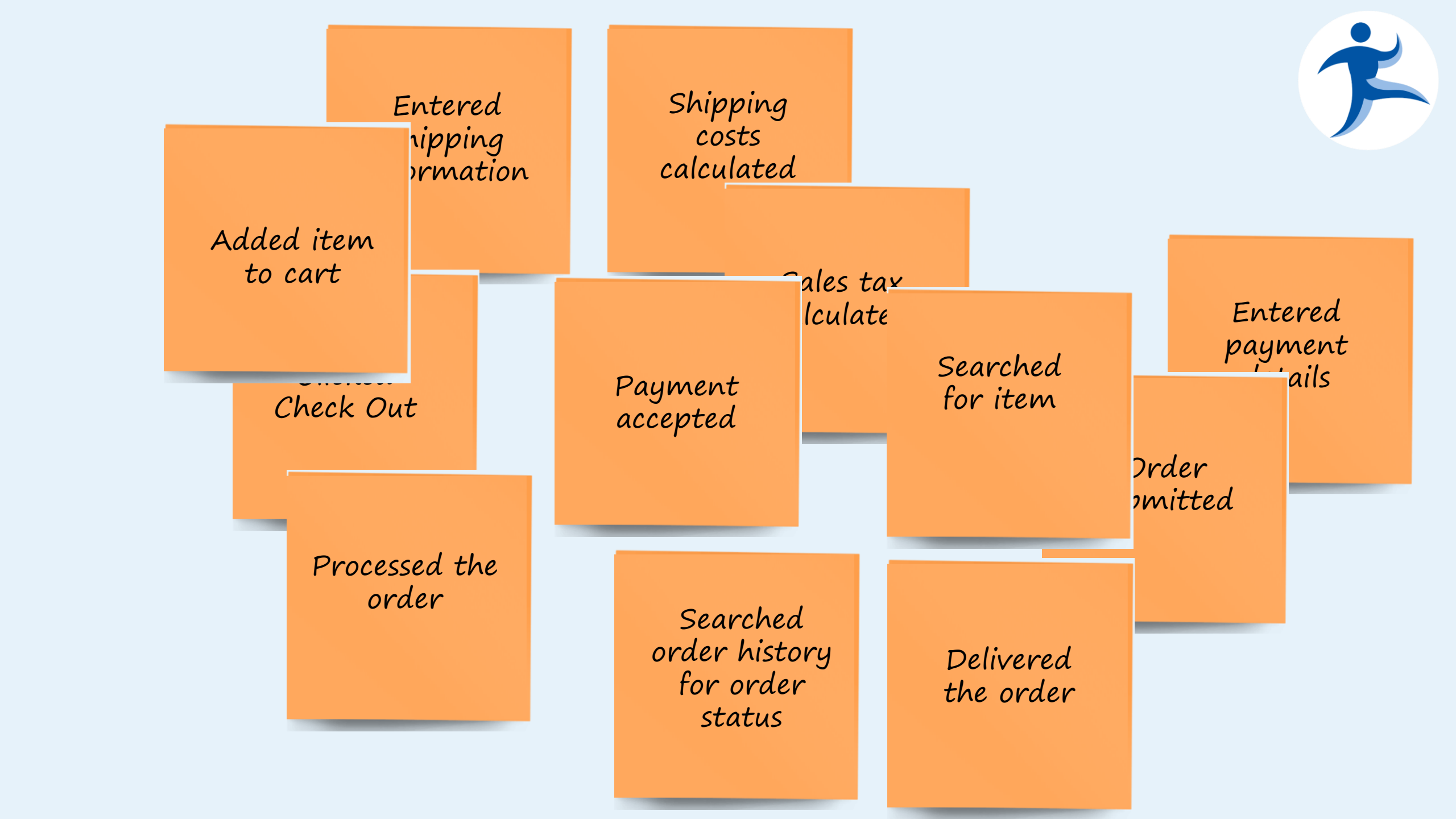
# EventStorming for Refactoring – Additional Sticky Notes

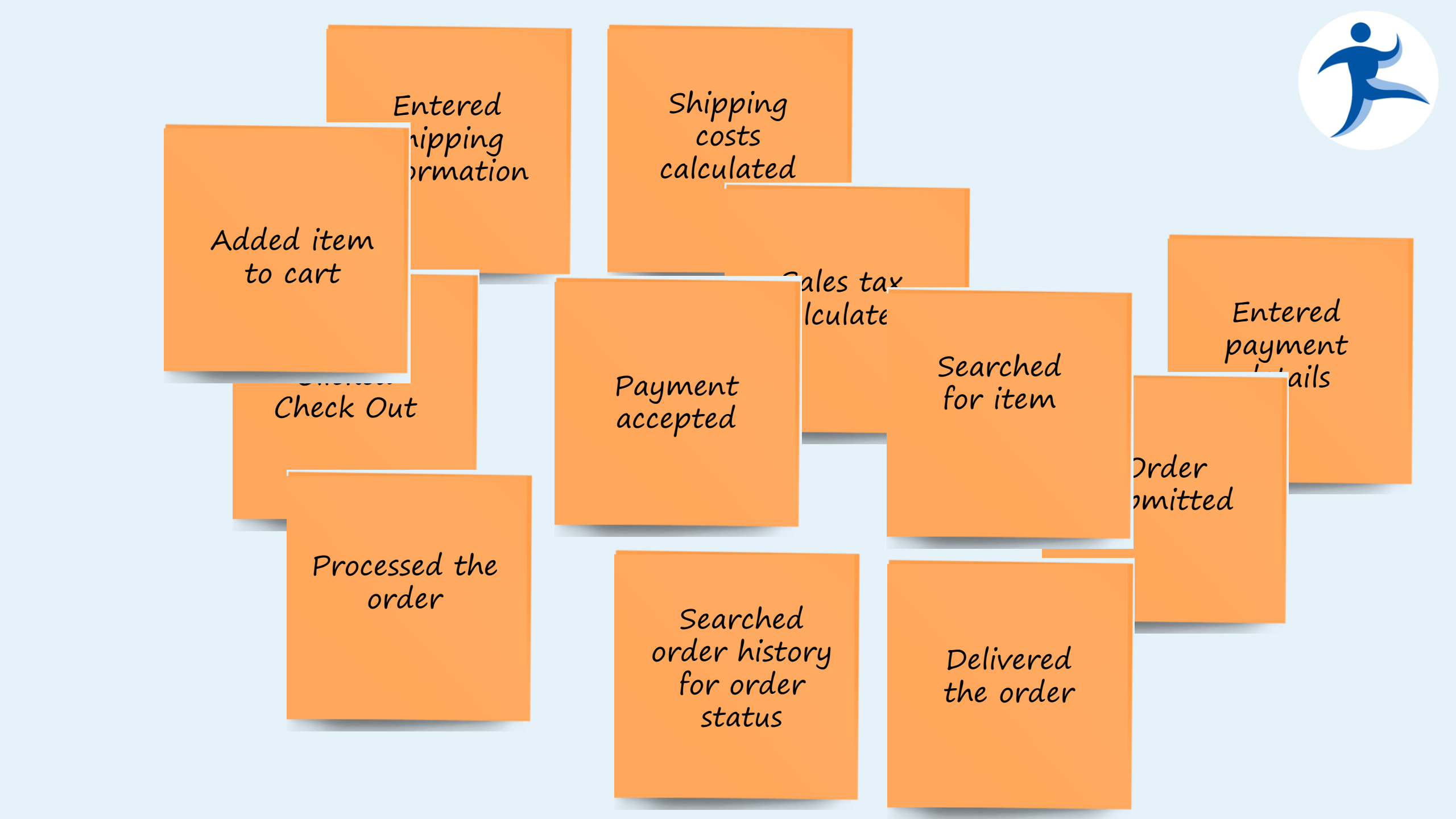
Command

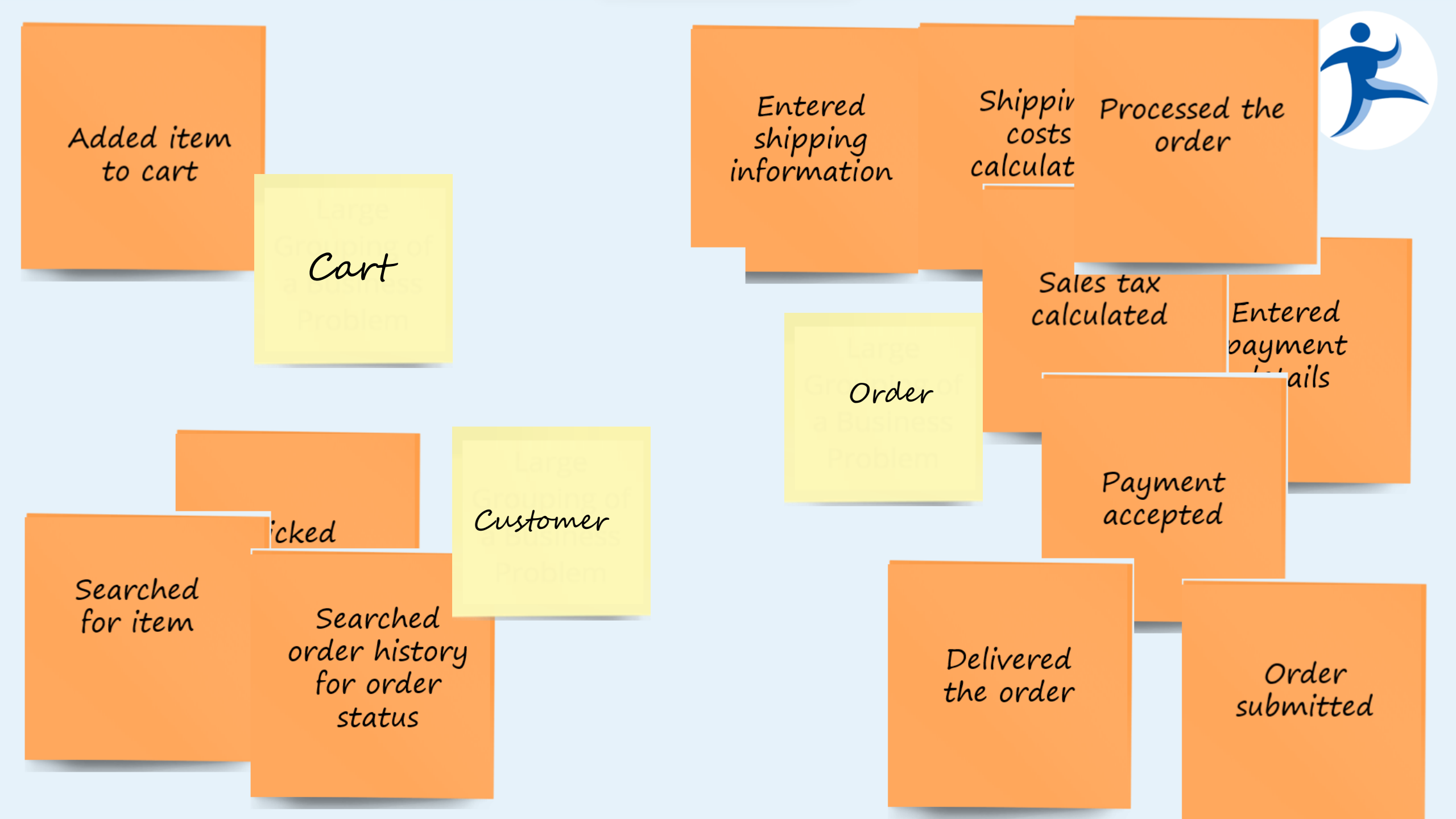
Aggregate

# Bounded Contexts

- Once **aggregates** are identified, **boundaries** are drawn
- Usually helps identify **microservices**
- Examples:
  - Product
  - Order
  - Customer
  - Delivery
  - Payment Processing
  - Tax Systems


Entered
ipping
rmation
Shipping
costs
calculated
Added item
to cart
ales tax
lculate
Entered
payment
ails
Check Out
Payment
accepted
Searched
for item
Order
mitted
Processed the
order
Searched
order history
for order
status
Delivered
the order


Entered
ipping
rmation
Shipping
costs
calculated
Added item
to cart
ales tax
lculate
Entered
payment
ails
Check Out
Payment
accepted
Searched
for item
Order
mitted
Processed the
order
Searched
order history
for order
status
Delivered
the order


Added item to cart
Cart
Entered shipping information
Shippi
costs
calculat
Processed the order
Sales tax calculated
Entered
payment
ails
Order
Payment accepted
Customer
Searched for item
Searched order history for order status
Delivered the order
Order submitted

# Analyze EventStorming Results

- Once **events** are laid out, identify entities related to those events. Is there one that makes sense as an **aggregate**?
- Create **aggregates** with their functions.
- Draw boundaries.
- Use **bounded contexts** as guides for microservices… or modules.

# Talk about splitting functionality as a team

- Do the splits make sense?
- Is there a ubiquitous language established? This would be a good point to get everyone on the same terms.

# Modulith to Microservices?

Modulith as a stepping stone

# Strangler Fig Pattern

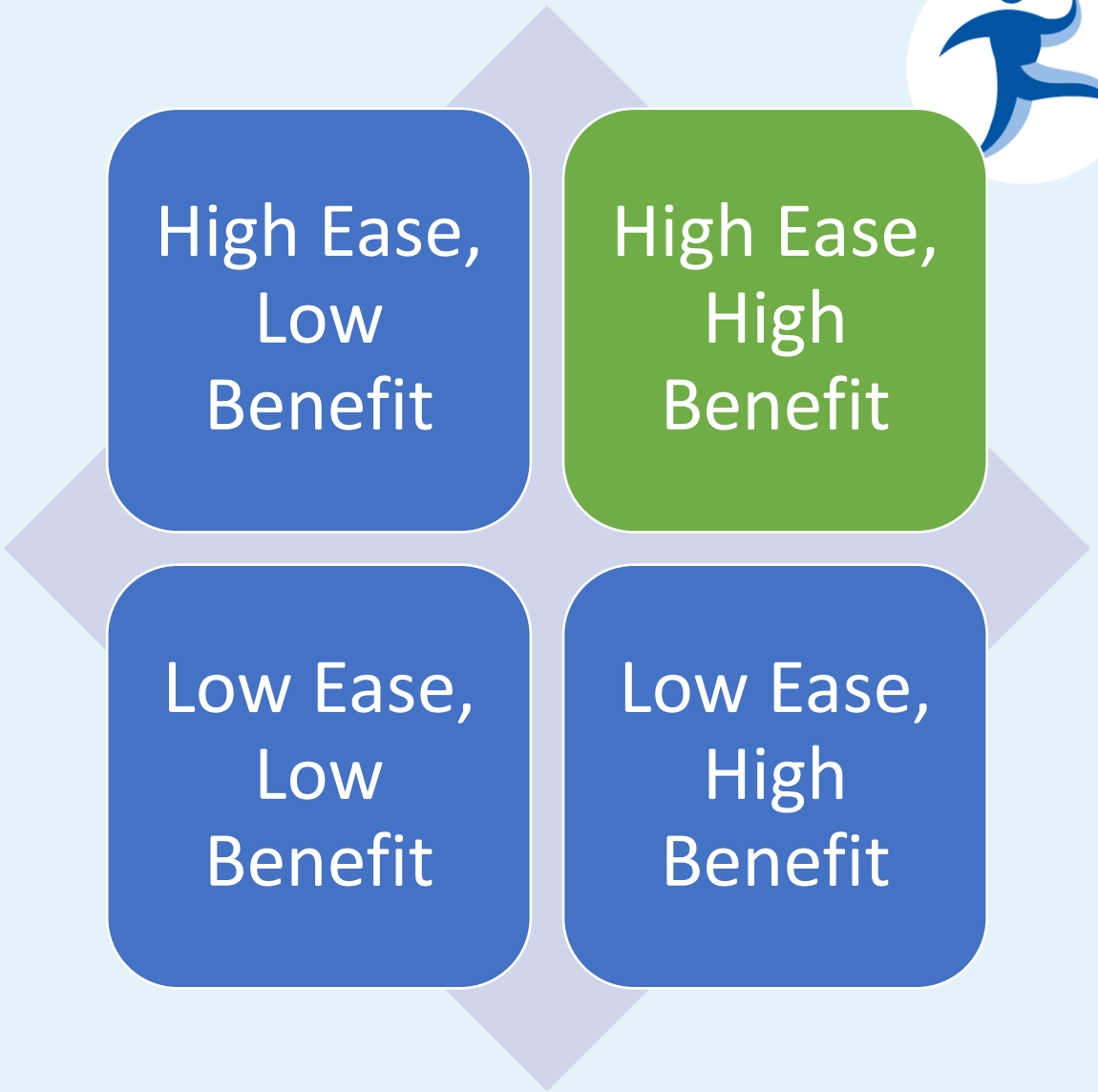
- Create the adapter for the front-end to use
- Migrate one service at a time
  - Focus on the services where microservices will give the best ROI
  - Cost isn't just $$$
  - Maintenance costs
  - Decoupling time
- Monolith shrinks over time
- If going wholly microservices, monolith will disappear

- But sometimes, a mix of the two may be where you are for awhile

Photo by **David Clode** on **Unsplash**

# Low Hanging Fruit

- Break API layer out of monolith
- API calls to monolith and microservices
- Draw out service dependencies
  - Might determine order of migration
- New features follow the microservices pattern
- If specialized development teams, let them figure the priorities for their specialties.




Ease of Separation vs Benefit of Separation

# Case Study: Amazon Prime Video Detector

- Moved from serverless, microservices to modular monolith for defects monitoring tool
- Problems:
  - Infrastructure at high scale was very expensive
  - Some components were hitting hard scaling limits
  - Orchestration with AWS Step Functions was costly – multiple state transitions every second of the stream, billed per state transition
  - Passing video around was expensive in making Tier-1 calls to S3
  - Cost of building the blocks of code was too high for large scale

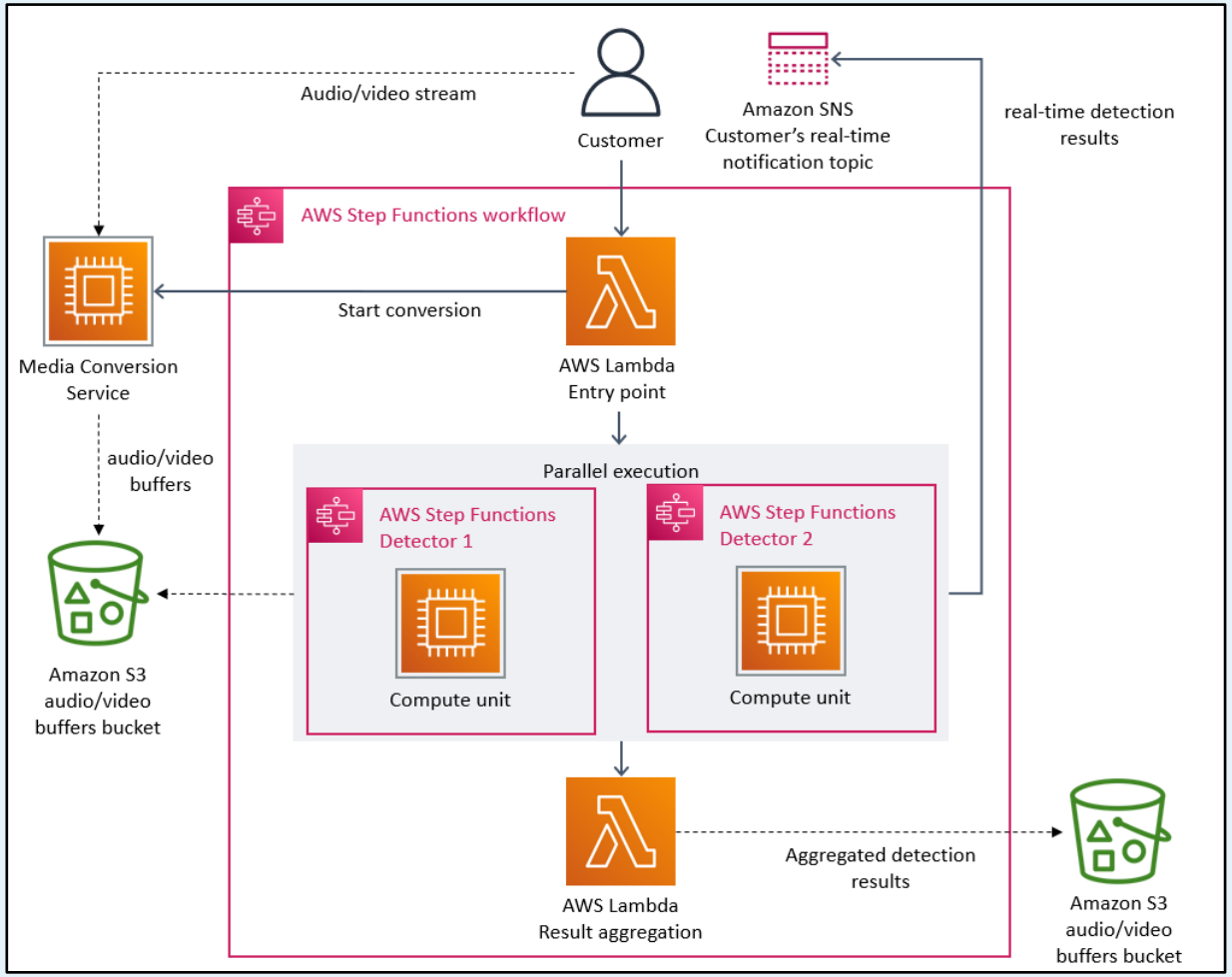



Diagram of Amazon Prime Video Detector with Microservices – Source: Prime Video Tech

# Amazon Prime Video Detector (continued)

- Moved to a monolith with vertical scaling for detectors
  - Detectors run within the same instance
- Benefits of moving to a monolith
  - Reduced costs by 90%
  - Streamlined for development, maintenance, and debugging
  - Minimized latency
- Microservices only when necessary
- Not an all-or-nothing approach – there is grey area!

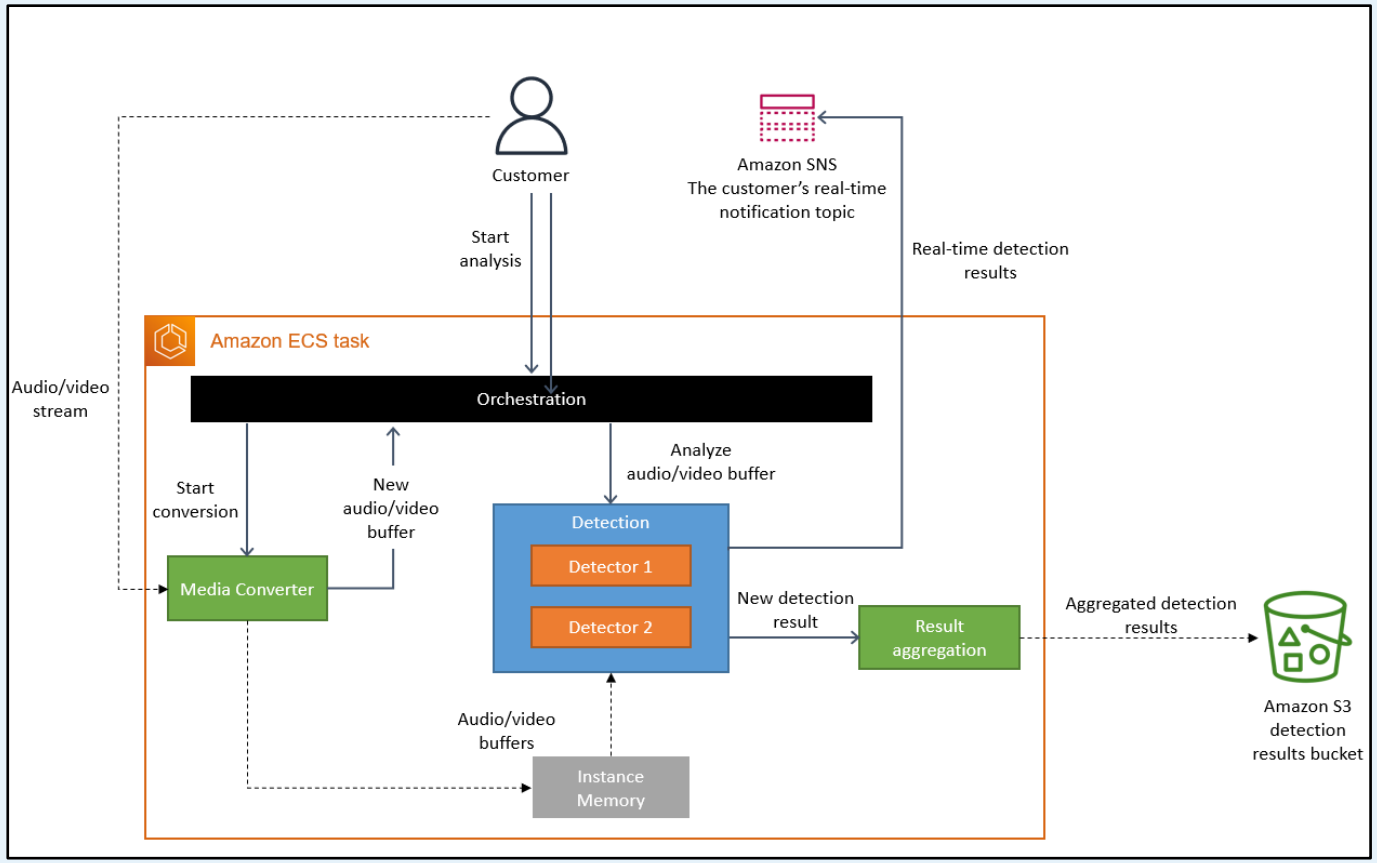
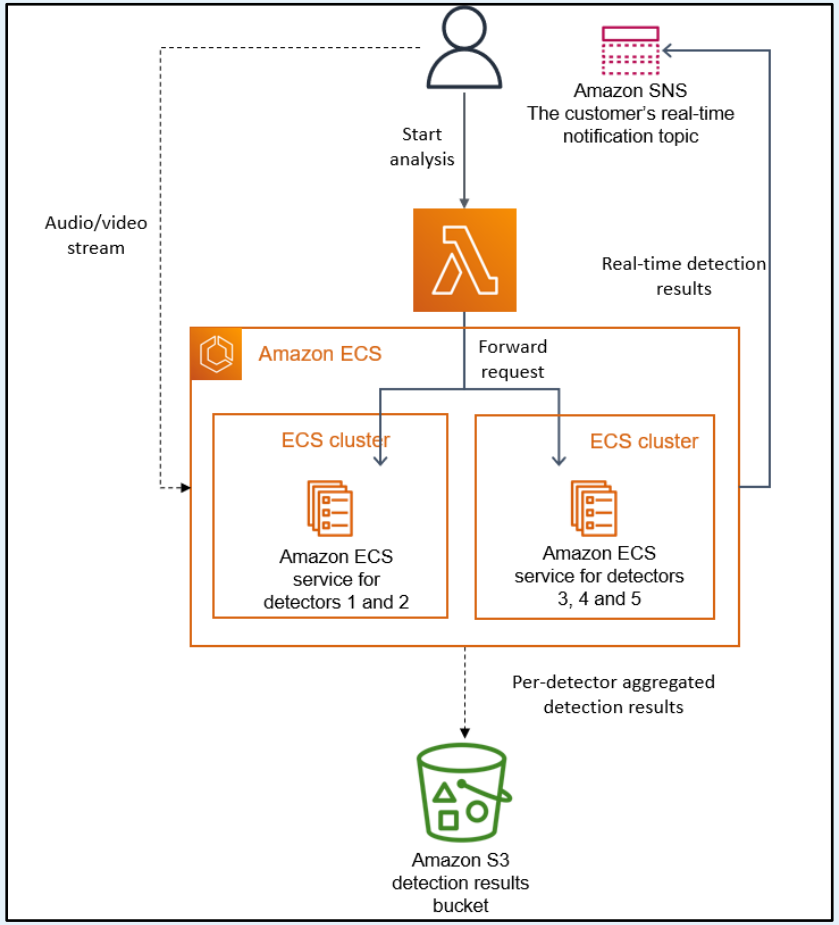



Diagram of Amazon Prime Video Detector with Microservices – Source: Prime Video Tech

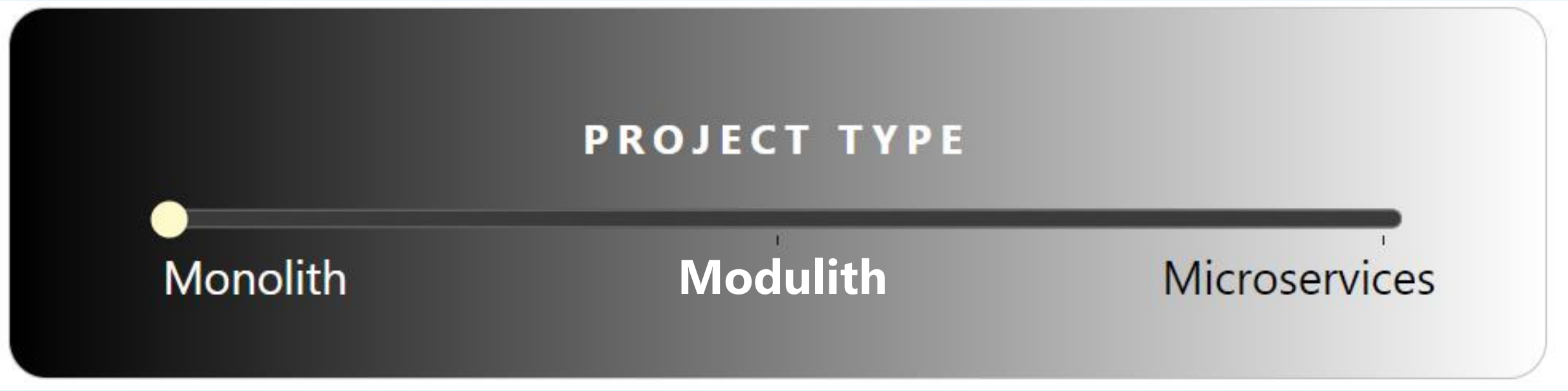
# But wait… capacity limits?

# Amazon Prime Video Detector (continued)



# Recap

# Is it all-or-nothing?
# Monolith or microservices?


PROJECT TYPE
Monolith
Modulith
Microservices

# What are the key points for using Microservices?

# When to Use Microservices

- High business complexity
- High need for fault isolation and resilience
- Specialized teams with polyglot backgrounds
- Independent scalability
- Enhanced maintainability
  - Agility for growth and improvement

# What are the key points for using Moduliths?

# When to Use Moduliths

- Low-to-no business complexity
- Cost-effective
- Tight integration
- Reduced operational complexity
- Team expertise in a tech stack

# Additional Resources

- eShopOnContainers - .NET microservices sample app
- Telerik – Creating Good Monoliths in ASP.NET Core
- Fear of Oblivion – Build the modular monolith first
- Martin Fowler – Monolith First
- Nicholas Frankel (A Java geek) – Chopping the Monolith

# EventStorming Resources

- EventStorming by Alberto Brandolini
- The EventStorming Handbook by Paul Rayner
- EventStorming.com
- 50,000 Stickies Later – Alberto Brandolini (Explore DDD 2017)
- 100,000 Stickies Later – Alberto Brandolini (Øredev 2019)

# Thank you!

/in/sadukie

sarah.dutkiewicz@nimblepros.com

https://bit.ly/contact-np