

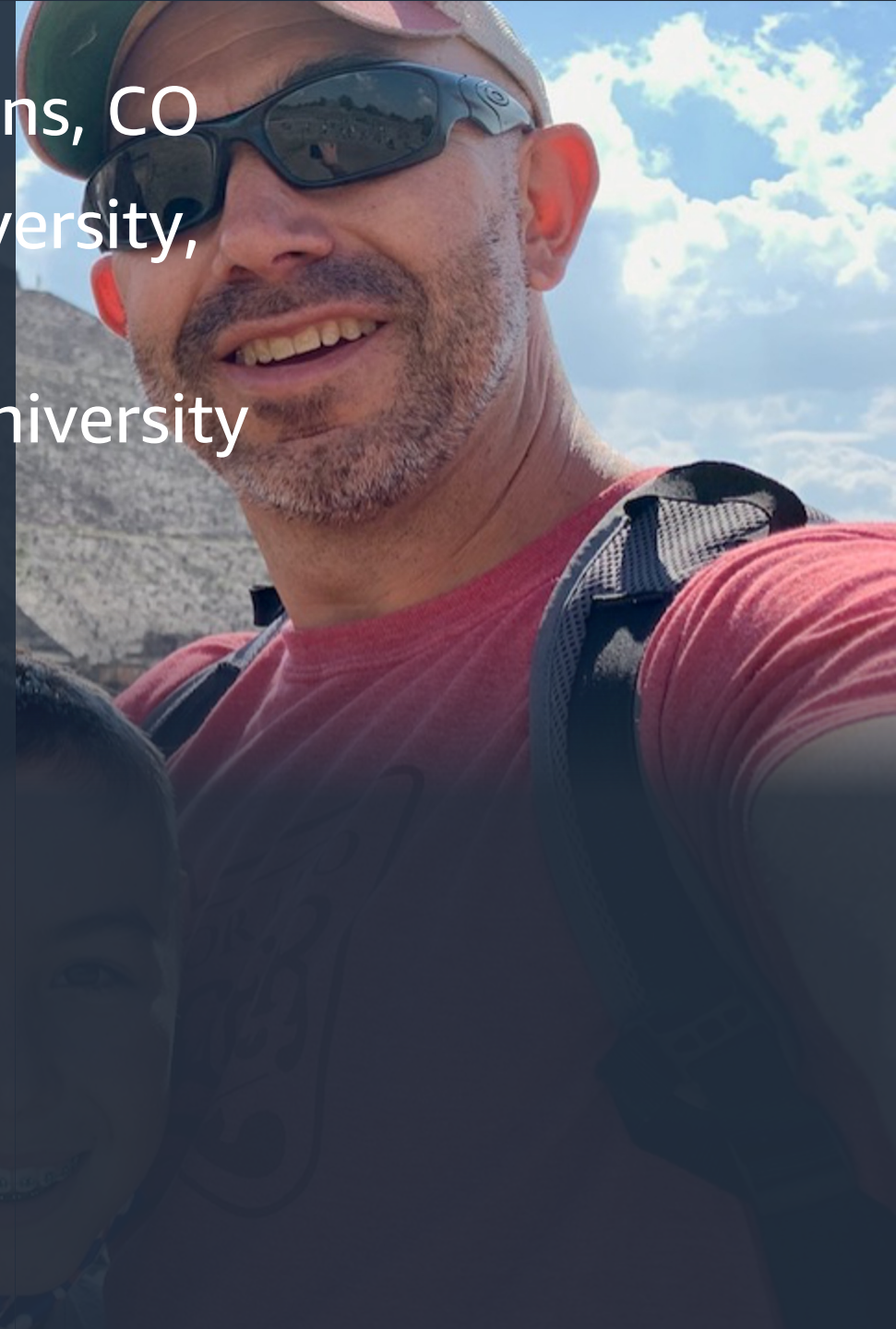


Accelerating Event-driven Architecture with domain-driven design

Brian Zambrano

Sr. Specialist Solutions Architect, Serverless
Amazon Web Services

- Grew up in Sunnyvale, CA; live in Fort Collins, CO
- Materials Engineering - Cal Poly State University, San Luis Obispo, CA
- Computer Science – San Francisco State University
- 20 years as a developer
 - AWS customer since 2010
 - Python, JS, a few others.
- AWS tenure
 - 1.3 years as AWS SDE
 - 2+ years as an account SA
 - 1.4 as Serverless Specialist



Honest question:

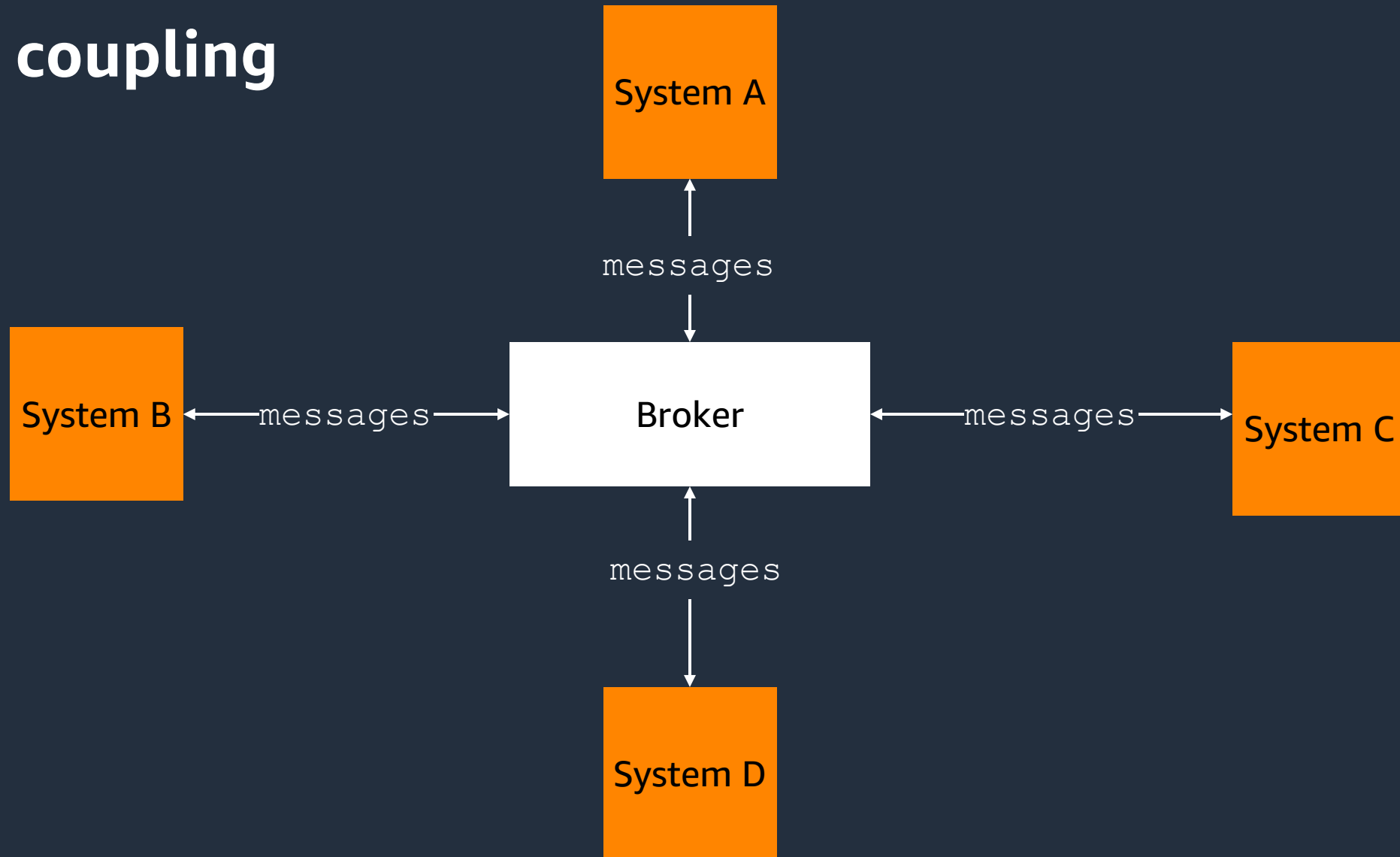
What is EDA?



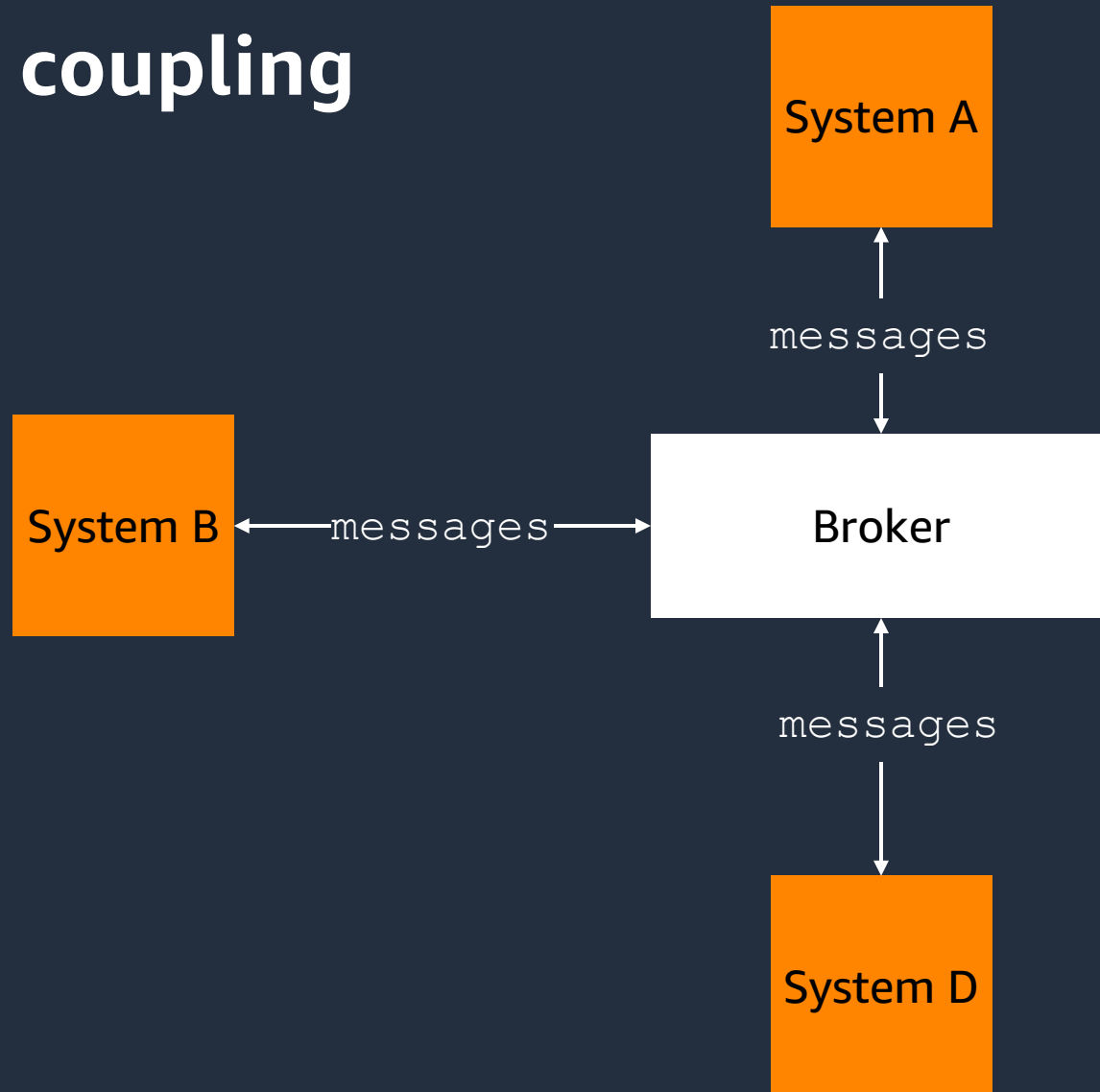
Event-driven architectures

An architectural style of building loosely-coupled software systems that work together by emitting and responding to events.

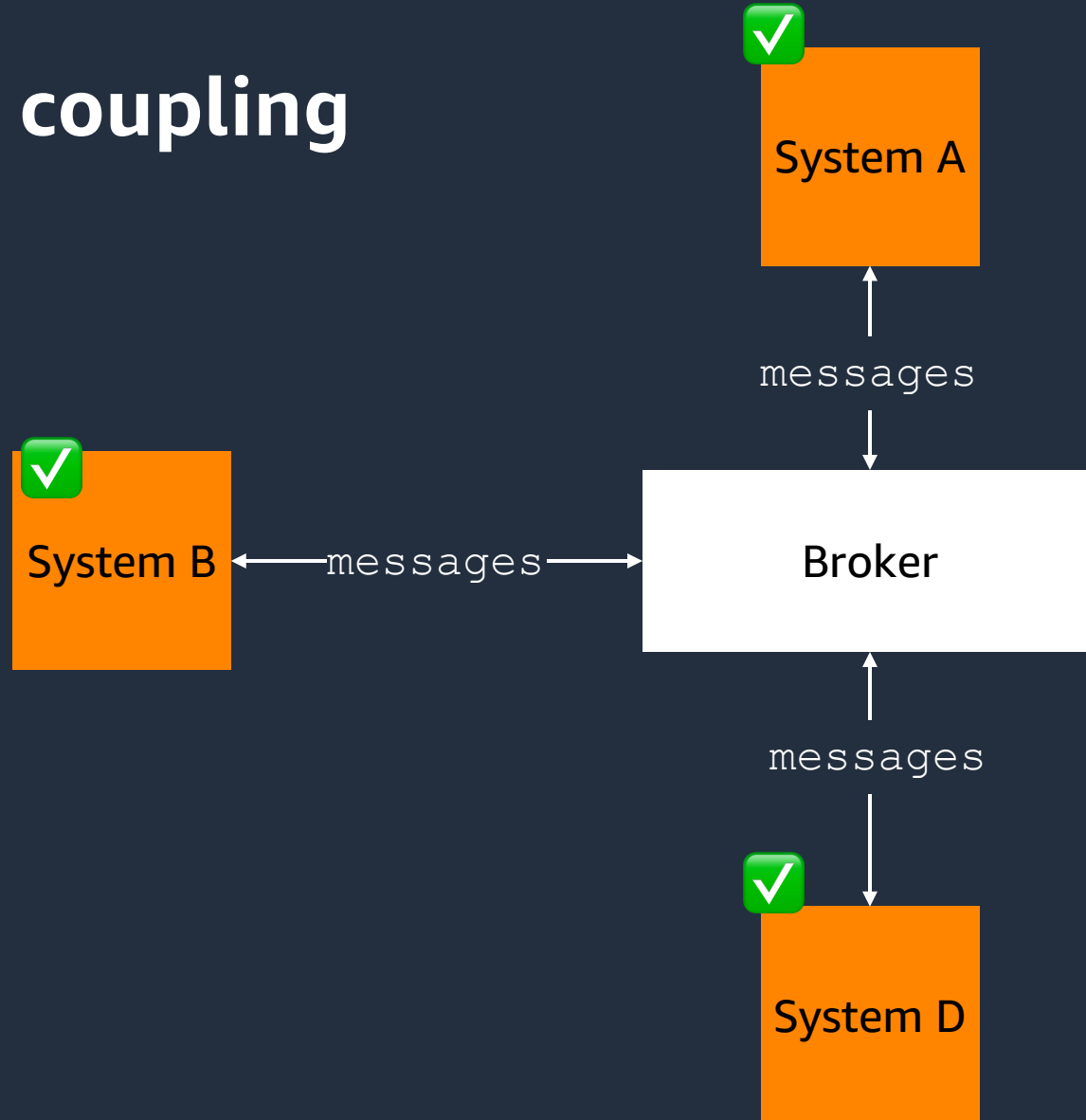
Loose coupling



Loose coupling



Loose coupling



Distributed Systems

Event-driven architectures are inherently distributed.

Bad news:

Distributed computing is **HARD!**



A Note on Distributed Computing

Jim Waldo, Cliff Wang, Martin Fowler, Sam Kendall

1994

Sun Microsystems Laboratories
Research Triangle Park
Mountain View, CA 94043

1 Introduction

Much of the current work in distributed, object-oriented systems is based on the assumption that objects form a single ontological class. This class consists of all entities that can be fully described by the specification of the set of interfaces supported by the object and the semantics of the operations in those interfaces. The class includes objects that share a single address space, objects that are in separate address spaces on the same machine, and objects that are in separate address spaces on different machines (with, perhaps, different architectures). On the view that all objects are essentially the same kind of entity, these differences in relative location are merely an aspect of the implementation of the object. Indeed, the location of an object may change over time, as an object migrates from one machine to another or the implementation of the object changes.

It is the thesis of this note that this unified view of objects is mistaken. There are fundamental differences between the interactions of distributed objects and the interactions of non-distributed objects. Further, work in distributed object-oriented systems that is based on a model that ignores or denies these differences is doomed to failure, and could easily lead to an industry-wide rejection of the notion of distributed object-based systems.

1.1 Terminology

In what follows, we will talk about local and distributed computing. By *local computing* (local object invocation, etc.), we mean programs that are confined to a single address space. In contrast, we will use the term *distributed computing* (remote object invocation, etc.) to refer to programs that make calls to other address spaces, possibly on another machine. In the case of distributed computing, nothing is known about the recipient of the call (other than that it supports a particular interface). For example, the client of such a distributed object does not know the hardware architecture on which the recipient of the call is running, or the language in which the recipient was implemented.

Given the above characterizations of “local” and “distributed” computing, the categories are not exhaustive. There is a middle ground, in which calls are made from one address space to another but in which some characteristics of the called object are known. An important class of this sort consists of calls from one address space to another on the same machine; we will discuss these later in the paper.

EDAs == distributed systems
distributed systems == HARD
 $\therefore$ EDAs == HARD



Considerations with Event-driven Architectures

Finding domain boundaries

Eventual consistency

At-least-once delivery

Patterns

- Event Sourcing

- CQRS

- Saga pattern

Choosing an event broker

Idempotency

Ordering

Failure scenarios

Schema validation

Versioning

Etc...



[This Photo](#) by Unknown Author is licensed under [CC BY-NC-ND](#)

[This Photo](#) by Unknown Author is licensed under [CC BY-ND](#)

Considerations with Event-driven Architectures

Finding domain boundaries

Choosing an event broker

Eventual consistency

At-least-once

Patterns

Event Sourcing

CQRS

Saga pattern

Etc...

Good news!!!

Considerations with Event-driven Architectures

Finding domain boundaries

Choosing an event broker

Eventual consistency

At-least-once

Patterns

Event Sourcing

CQRS

Saga pattern

Etc...

Many complexities are solvable
with technology and well-known
patterns...

Considerations with Event-driven Architectures

Finding domain boundaries

Choosing an event broker

Eventual consistency

At-least-once

Patterns

Event Sourcing

CQRS

Saga pattern

Etc...

Except one...

Considerations with Event-driven Architectures

Finding domain boundaries

Eventual consistency

At-least-once delivery

Patterns

- Event Sourcing

- CQRS

- Saga pattern

Choosing an event broker

Idempotency

Ordering

Failure scenarios

Schema validation

Broker topologies

Etc...

Considerations with Event-driven Architectures

Finding domain boundaries

Eventual consistency

At-least-once delivery

Patterns

Event Sourcing

CQRS

Saga pattern

Choosing and event broker

Idempotency

Ordering

Failure scenarios

Schema validation

Broker topologies

Etc...



A problem domain is the area you and your company are working.

You solve problems in this domain to provide value to your customers and to make money or exist as a business entity.

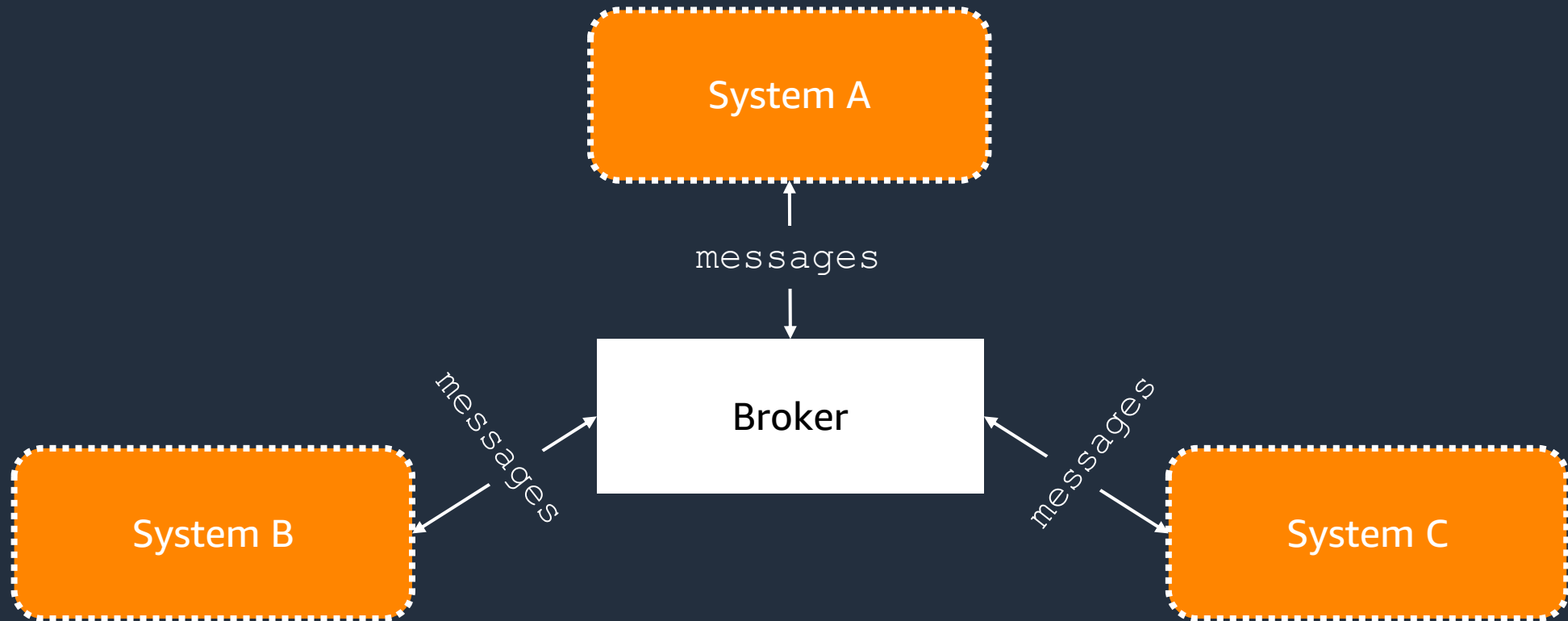
Problem domain

Subdomain

Subdomain

Subdomain

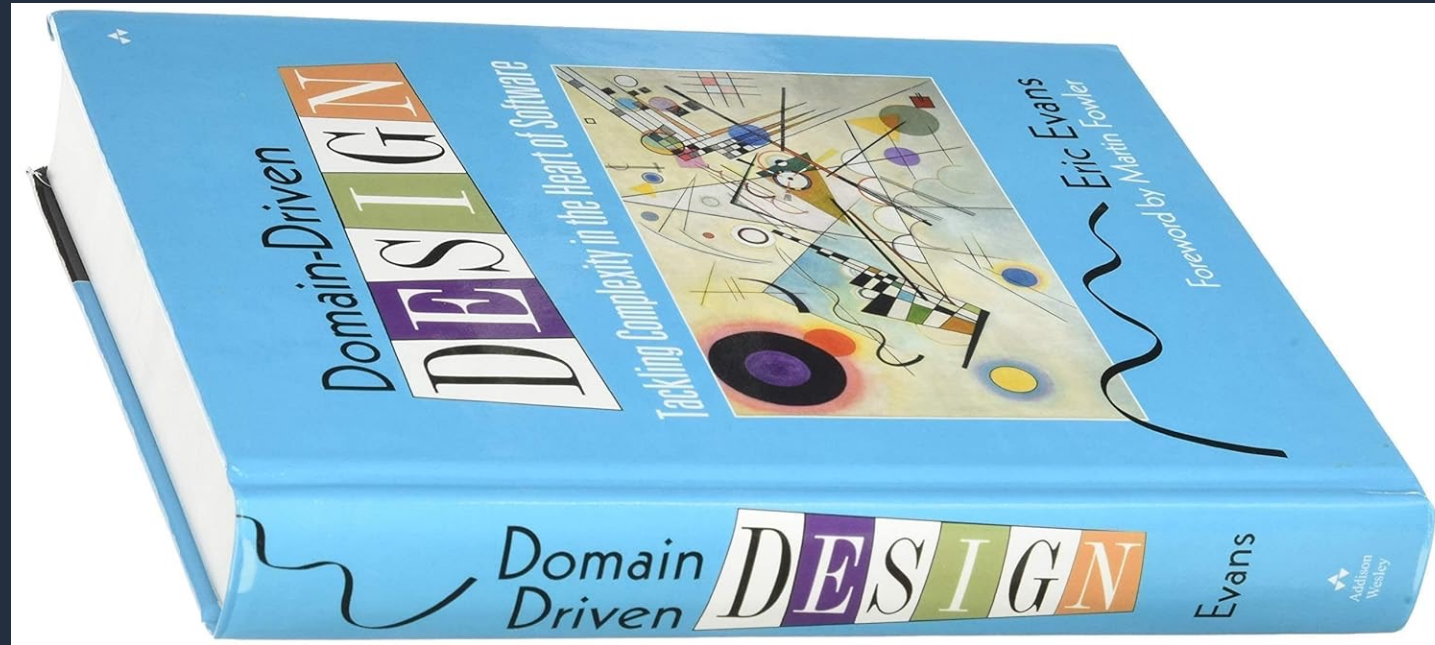
Problem domain



Problem domain

**How do you find
these boundaries?**

Domain-driven Design (DDD)



Domain-driven Design (DDD)

“Domain-driven design is both a way of thinking and a set of priorities, aimed at accelerating software projects that have to deal with complicated domains.”

- *Eric Evans, Domain-Driven Design (2003)*

30000
 1000000
 10000000

Domain-driven design focuses
on *understanding* a domain,
and its sub-domains.

DDD helps you to find a shared language spanning business and engineering.

**“It’s developers’ (mis)understanding,
not domain experts’ knowledge, that
gets released in production.”**

Alberto Brandolini

EventStorming creator



Developers ready to
code



How do we bring order and understanding to a non-trivial domain?

If developers misunderstand the domain...

Turn developers into domain experts.

Turn your teams into domain experts.

EVENT STORMING

Alberto Brandolini

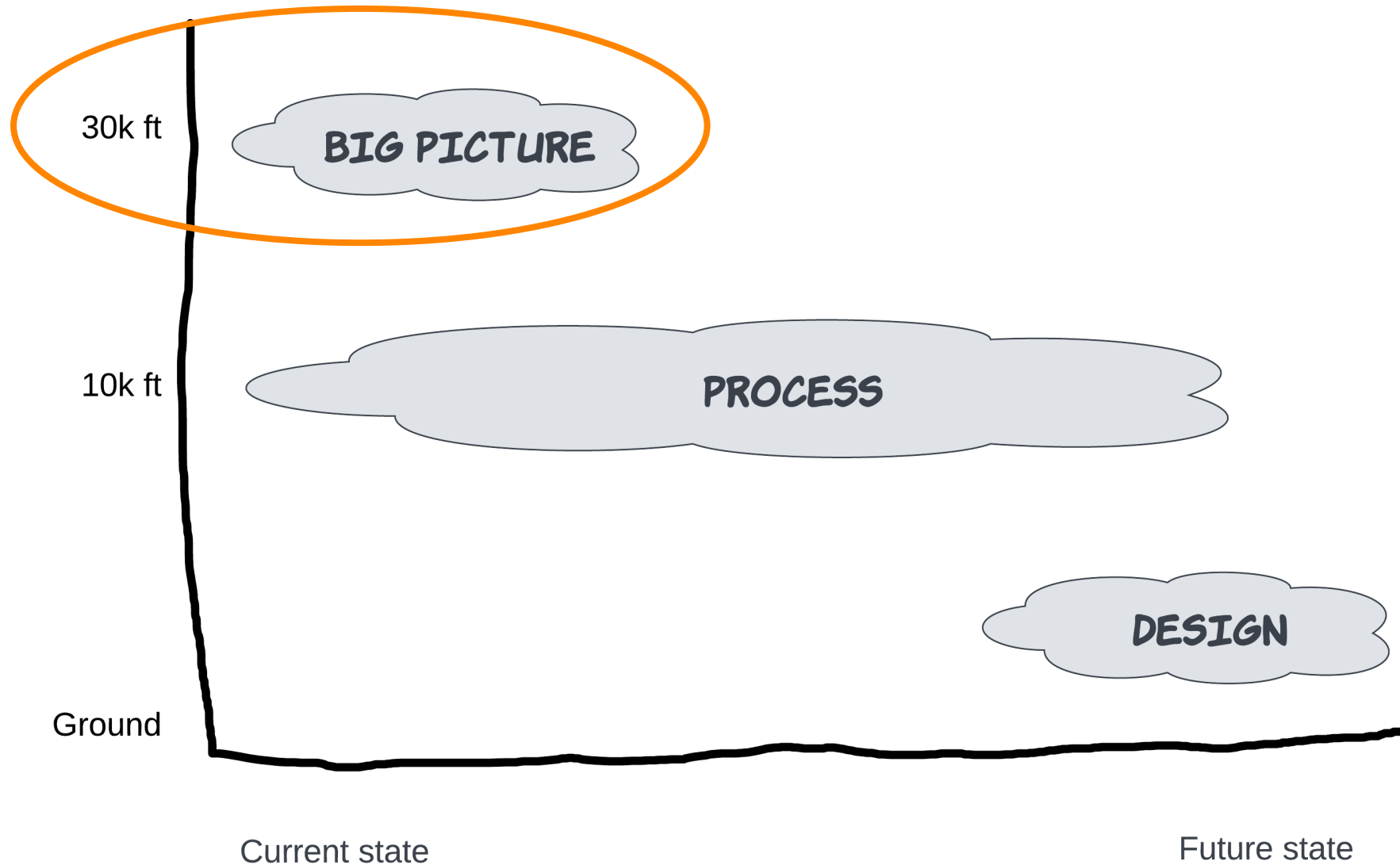


1. Identify events, systems, actors, etc.
2. Cluster-related concepts
3. Define bounded contexts and sub-domains

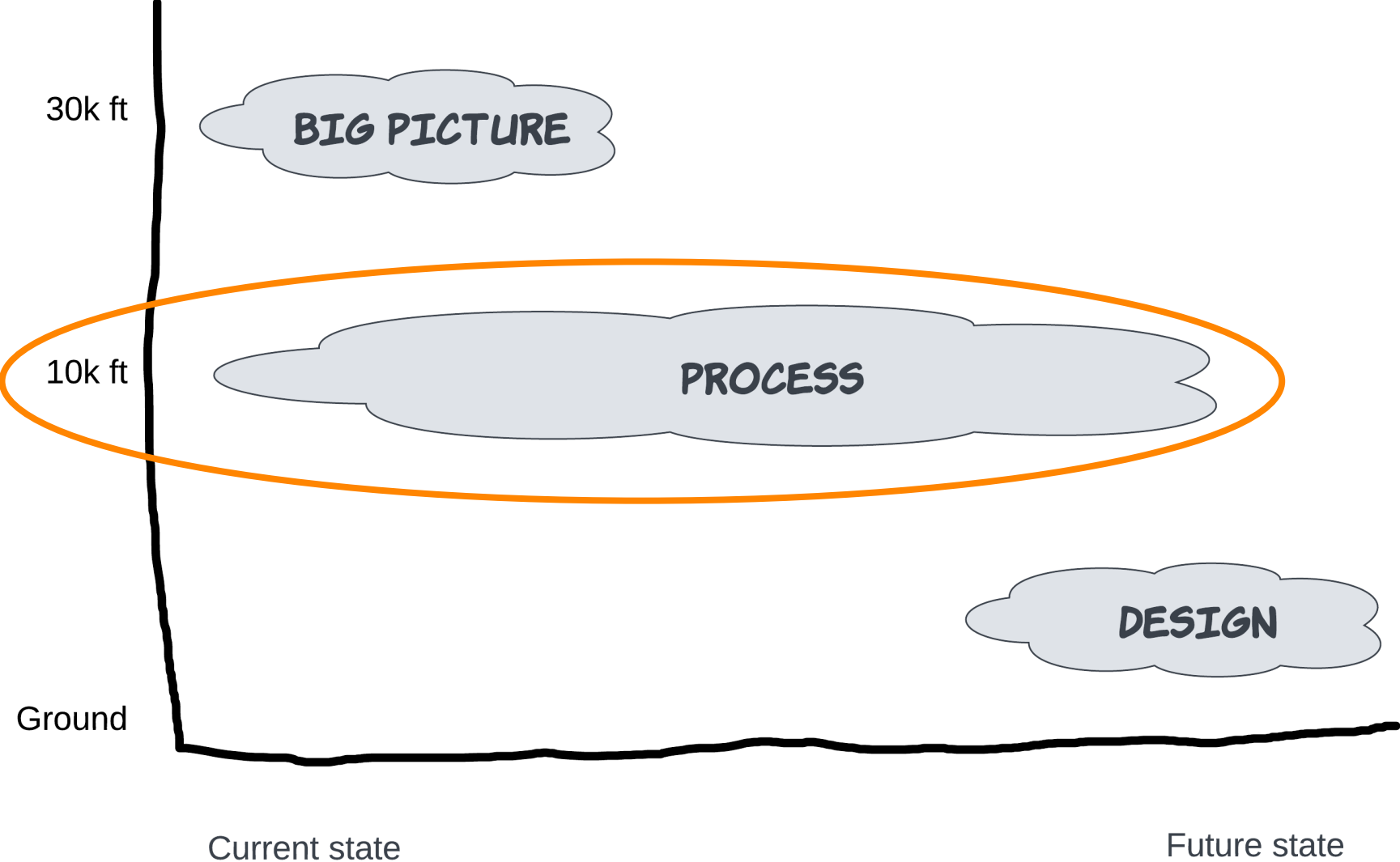
What is EventStorming?

A workshop-based approach to breaking down a non-trivial domain with the goal of coming to a shared understanding.

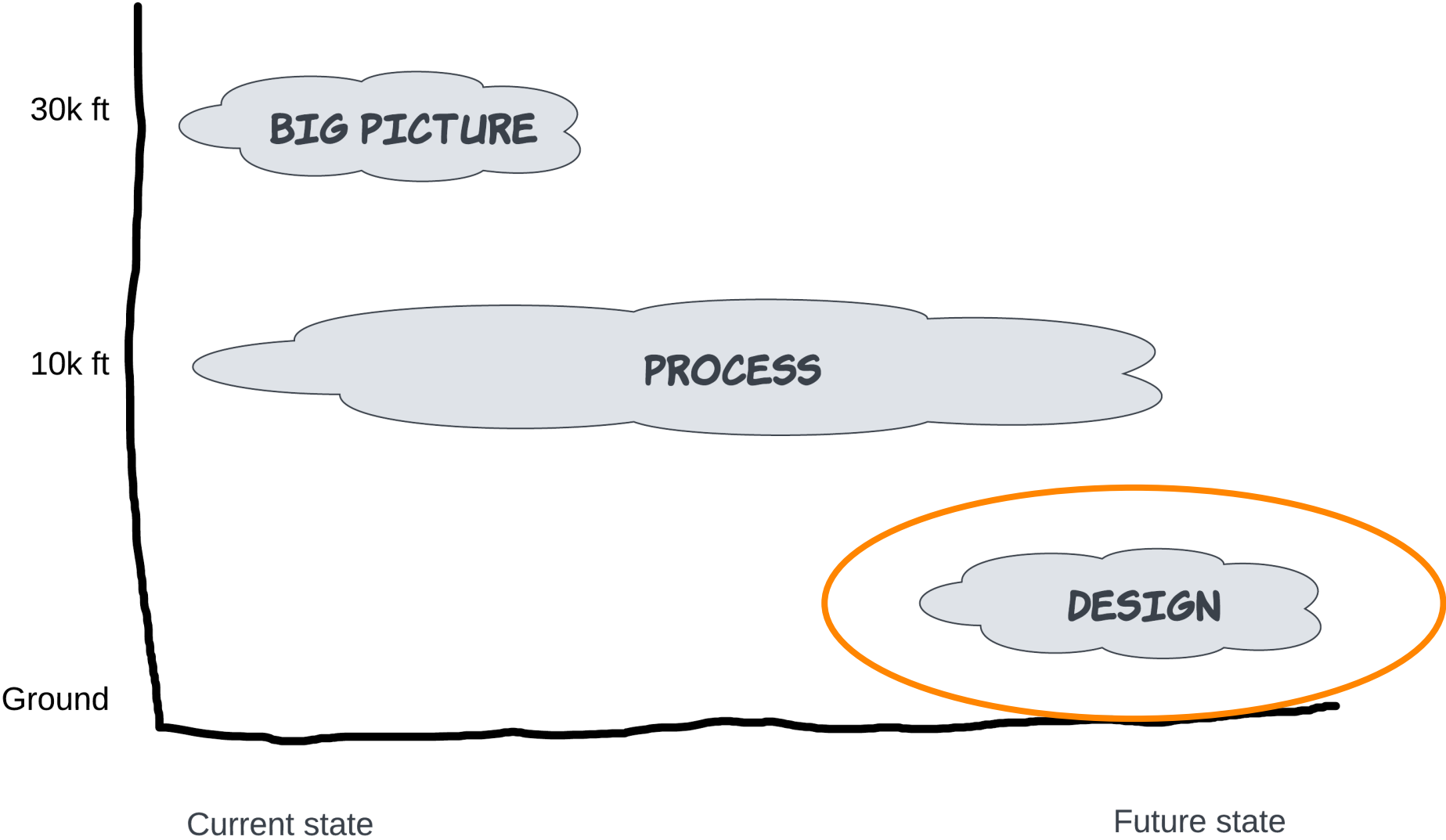
EventStorming flavors



EventStorming flavors

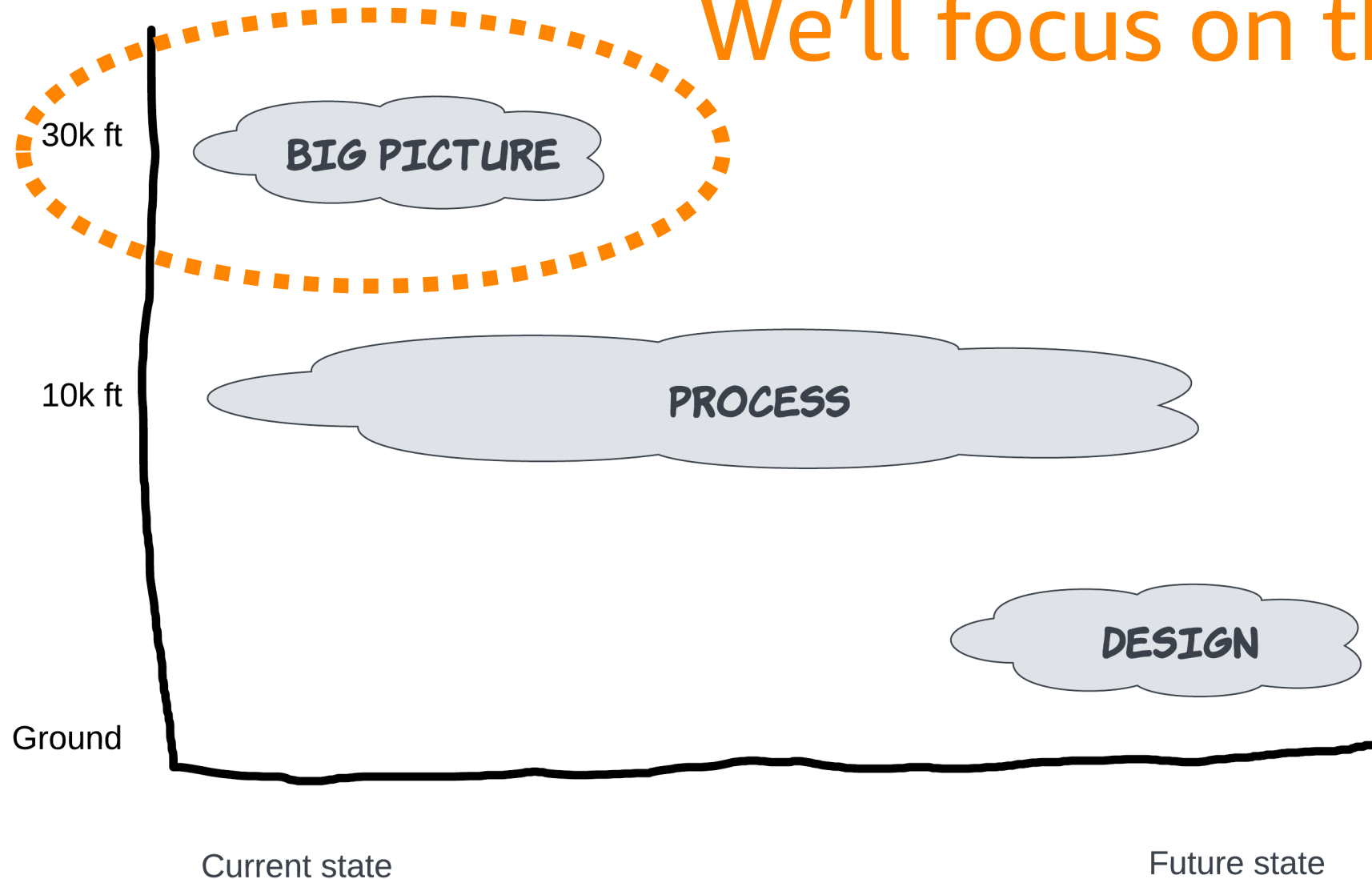


EventStorming flavors



EventStorming flavors

We'll focus on this





event

Something that a domain expert cares about. Immutable facts that have occurred in the past.

Step 1: Audience

Invite the right people!

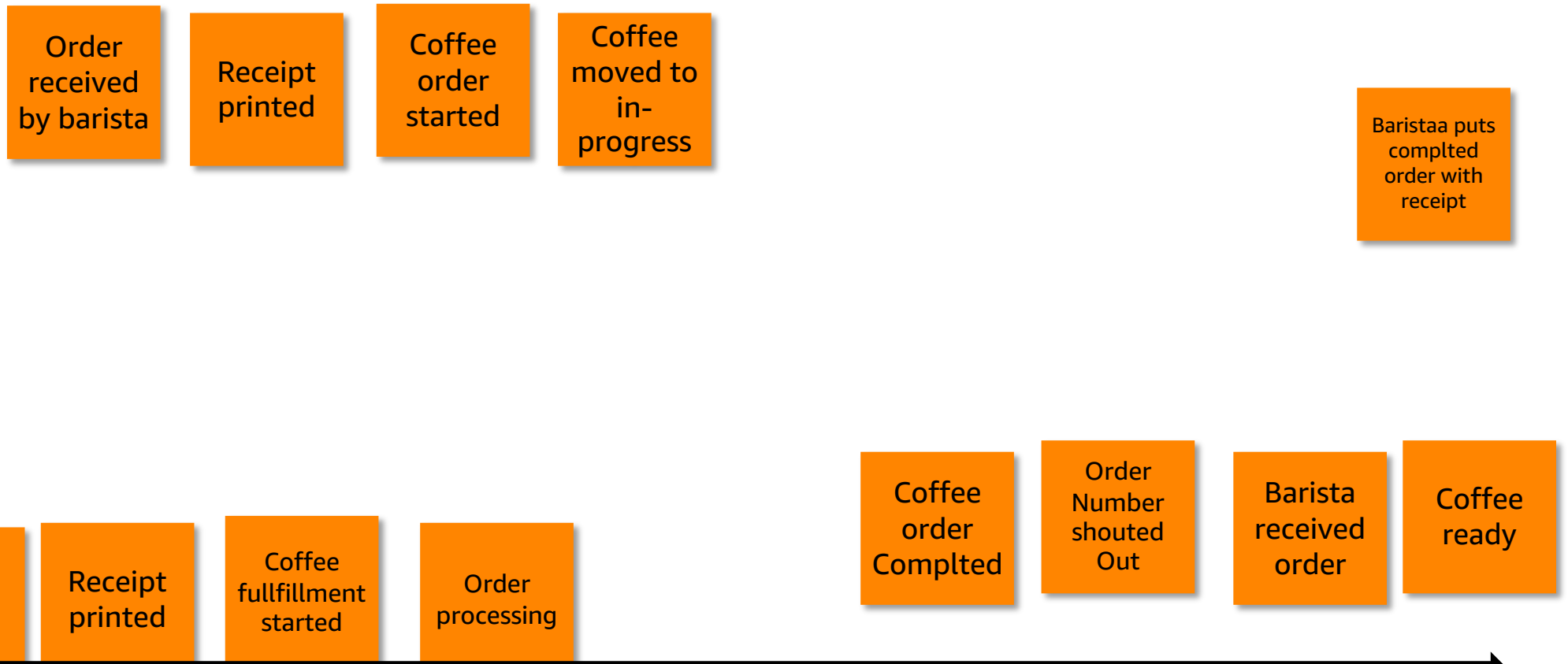
Step 2: Events

Start by crowdsourcing all of the events that occur within a particular journey.

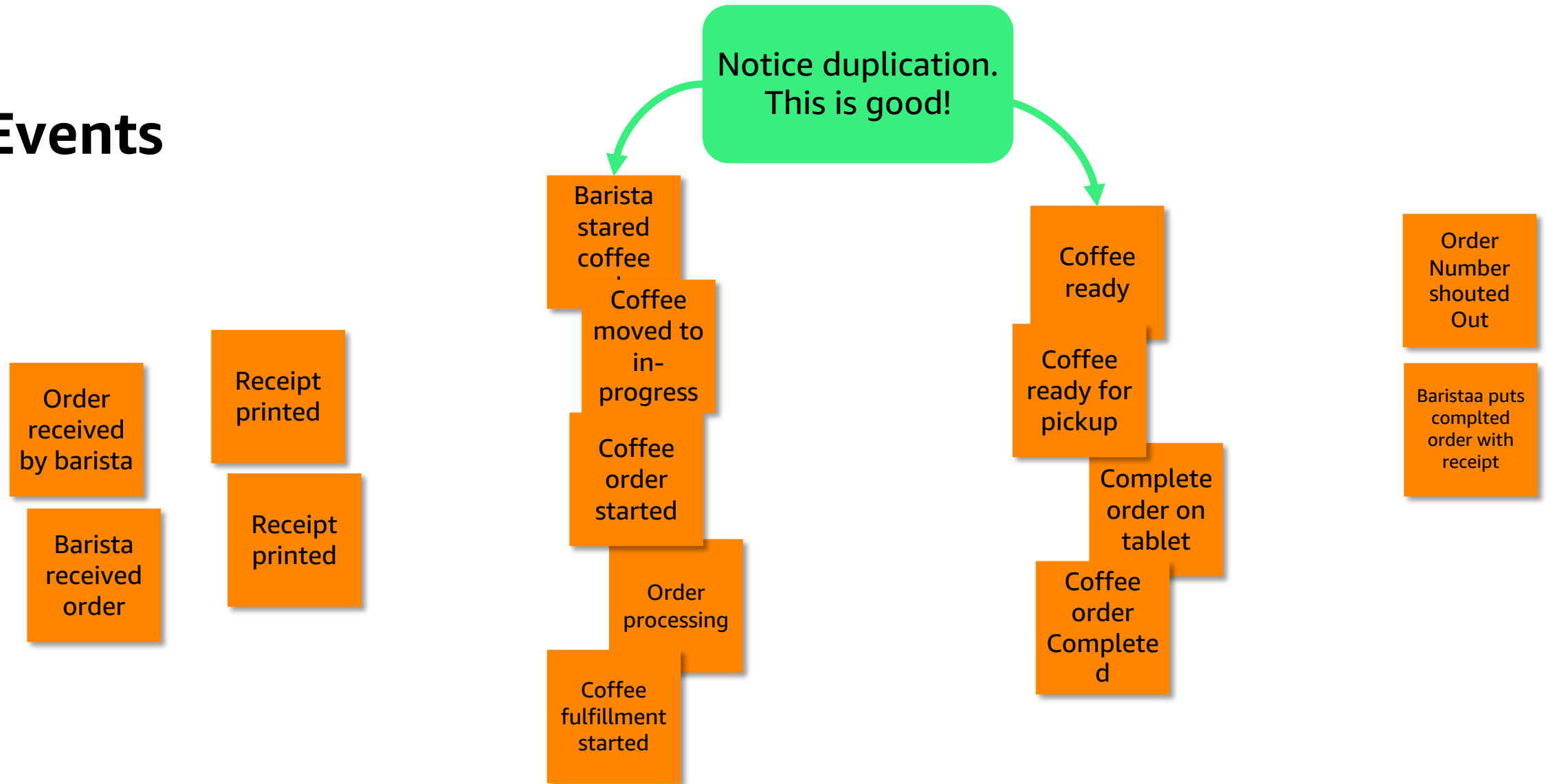
Step 2: Events

Everyone writes individually.
Don't worry about details.

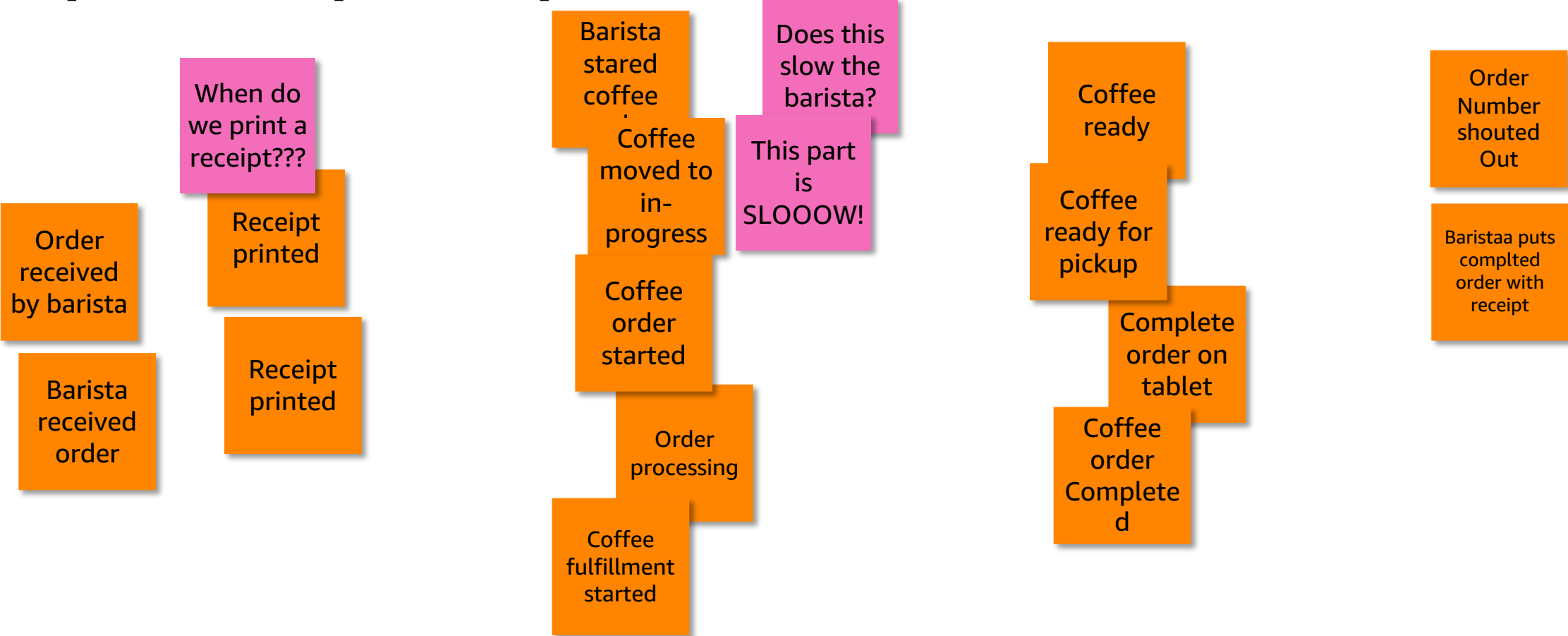
Events



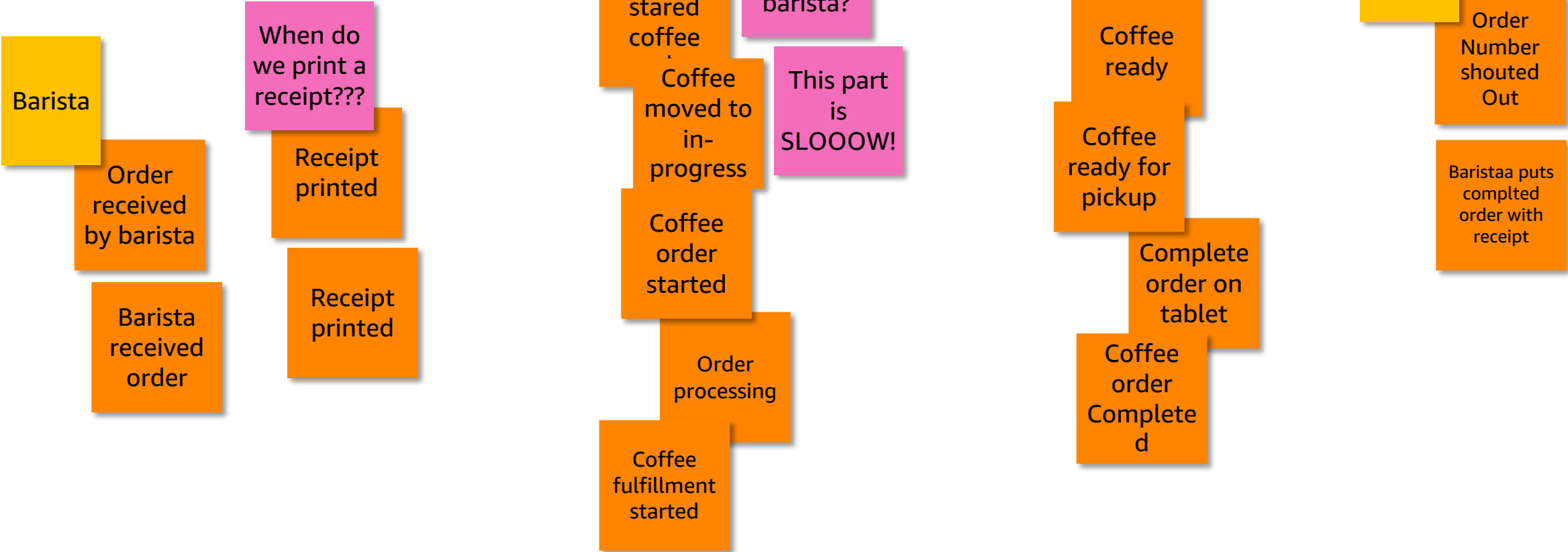
Events



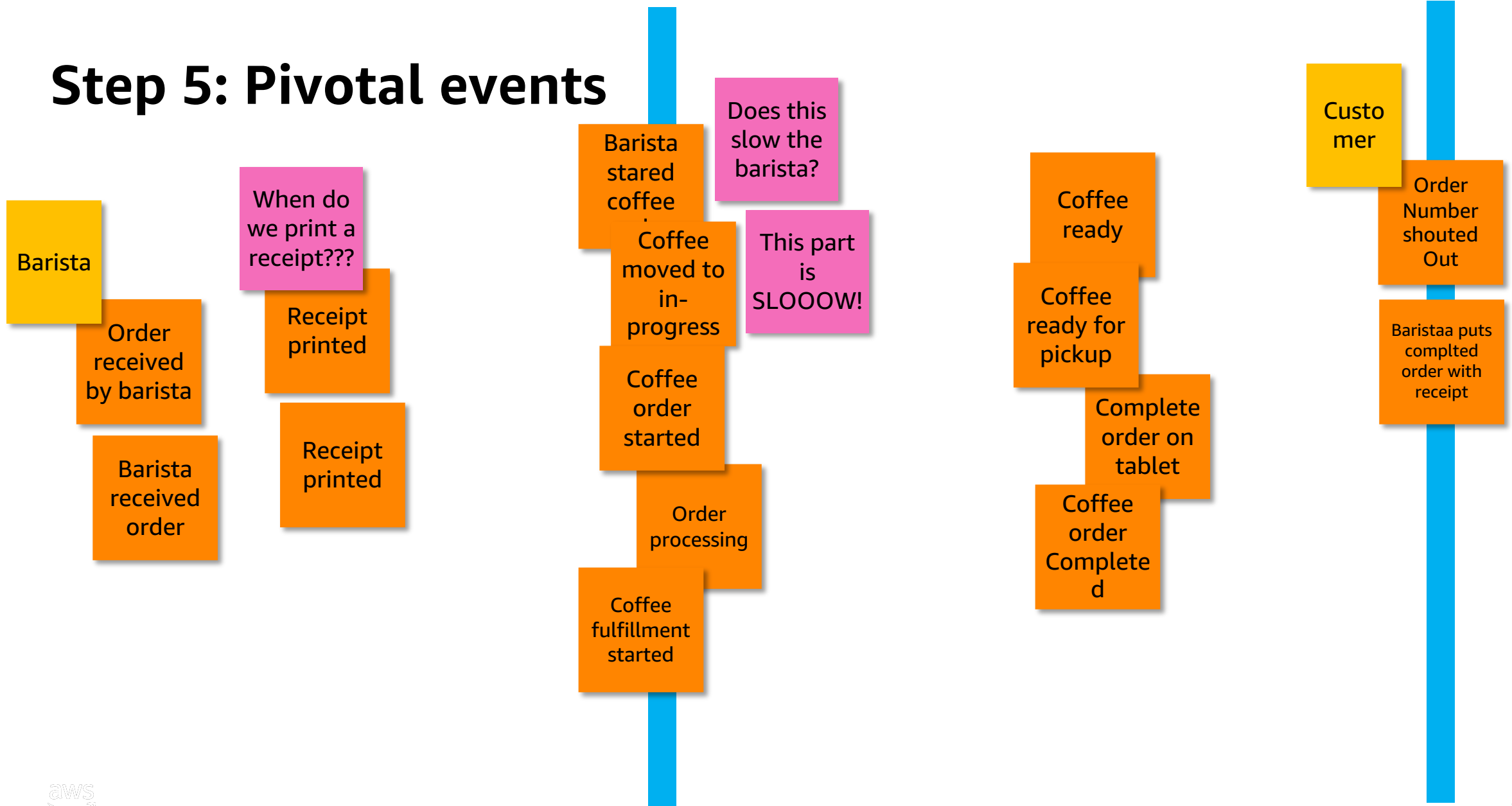
Step 3: Hot spots / questions



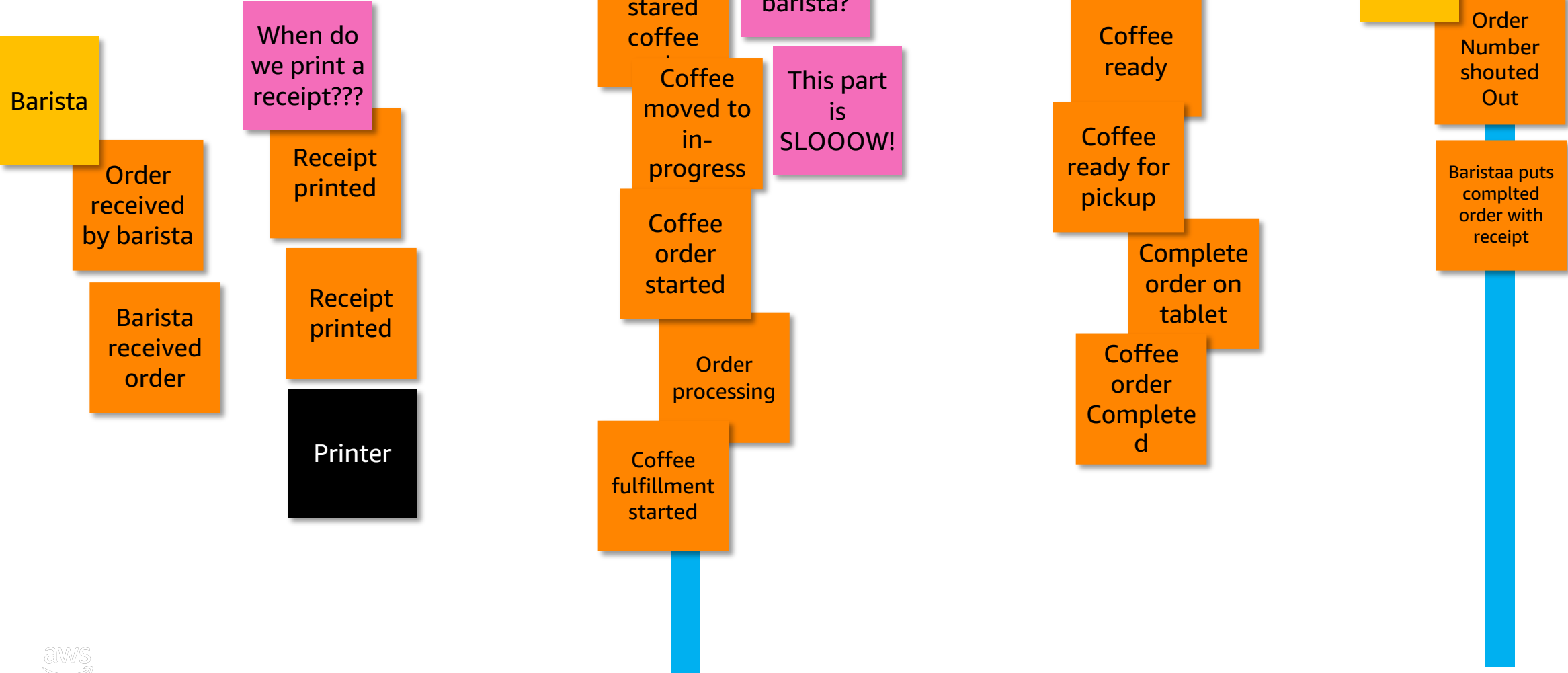
Step 4: People



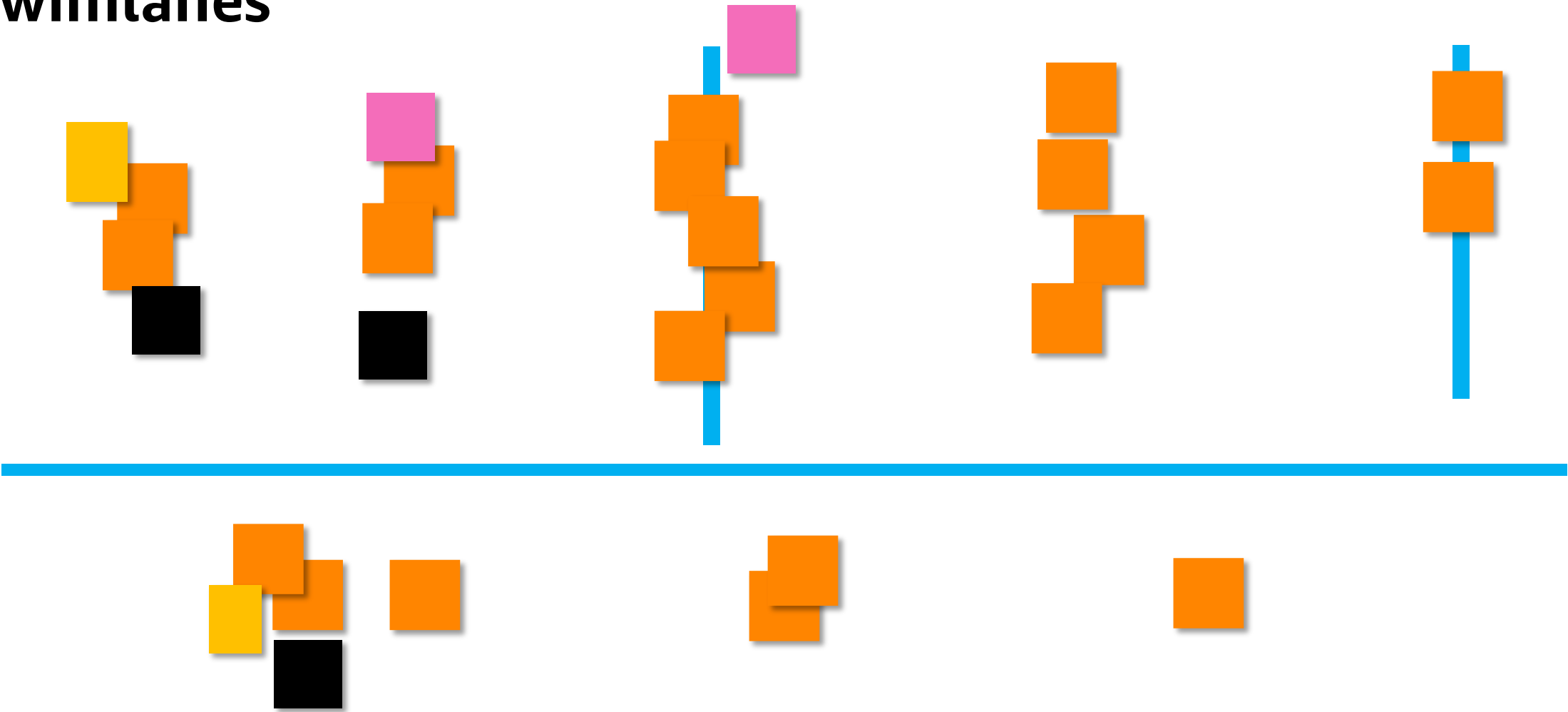
Step 5: Pivotal events



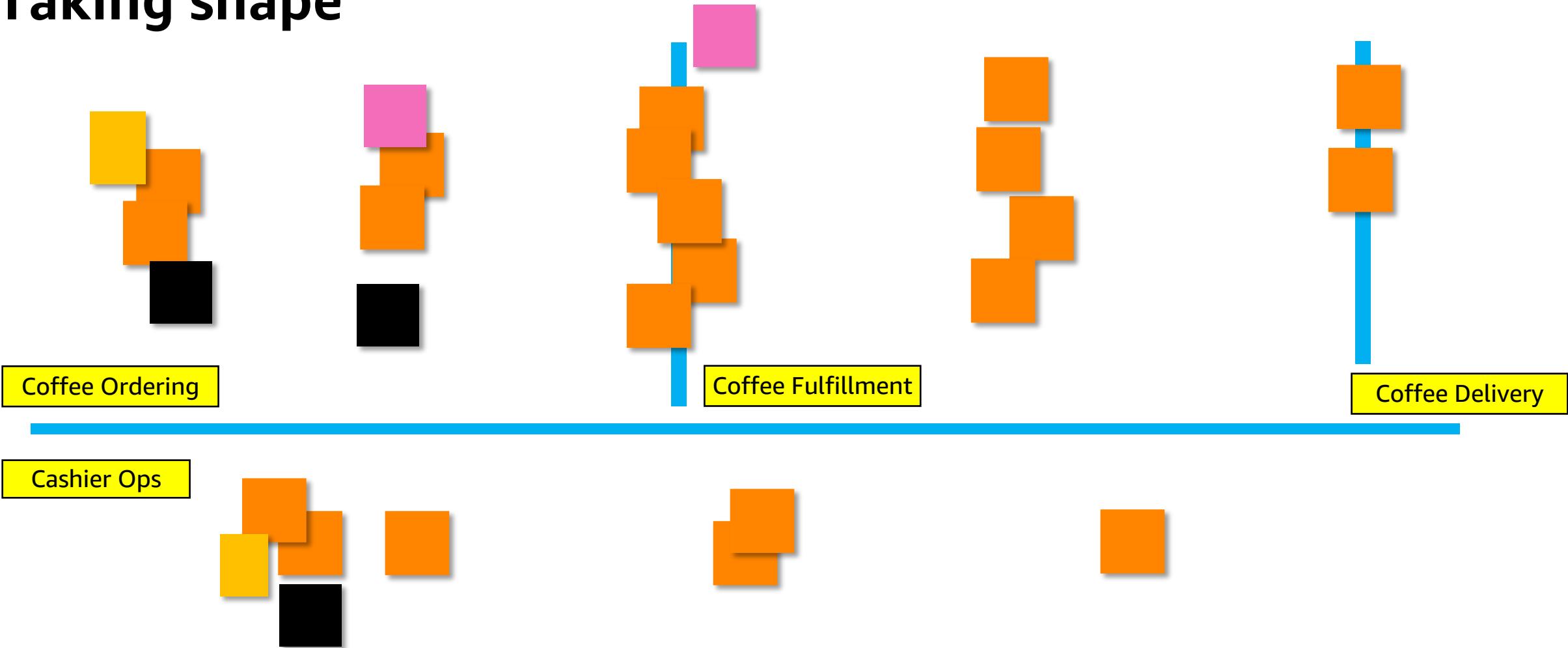
Step 6: External systems



Swimlanes

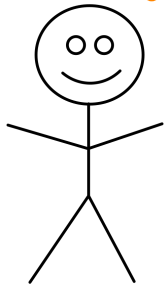


Taking shape

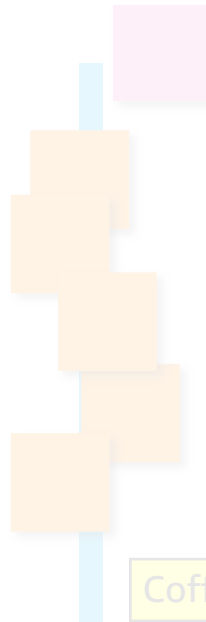


Walking through the story

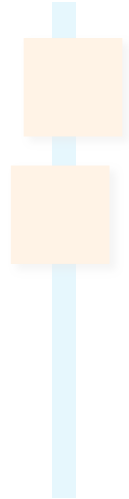
A user scans the
QR code and logs
in



Coffee Ordering



Coffee Fulfillment



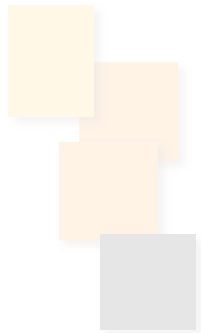
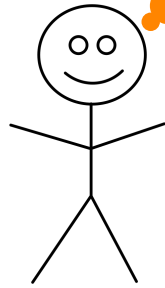
Coffee Delivery

Walking through the story

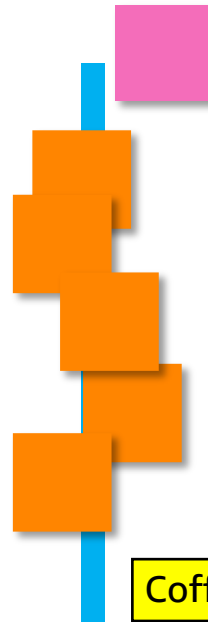


Walking through the story

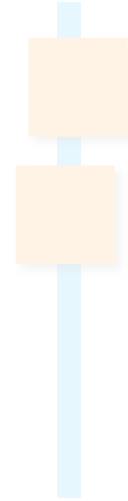
The barista
receives the order.



Coffee Ordering

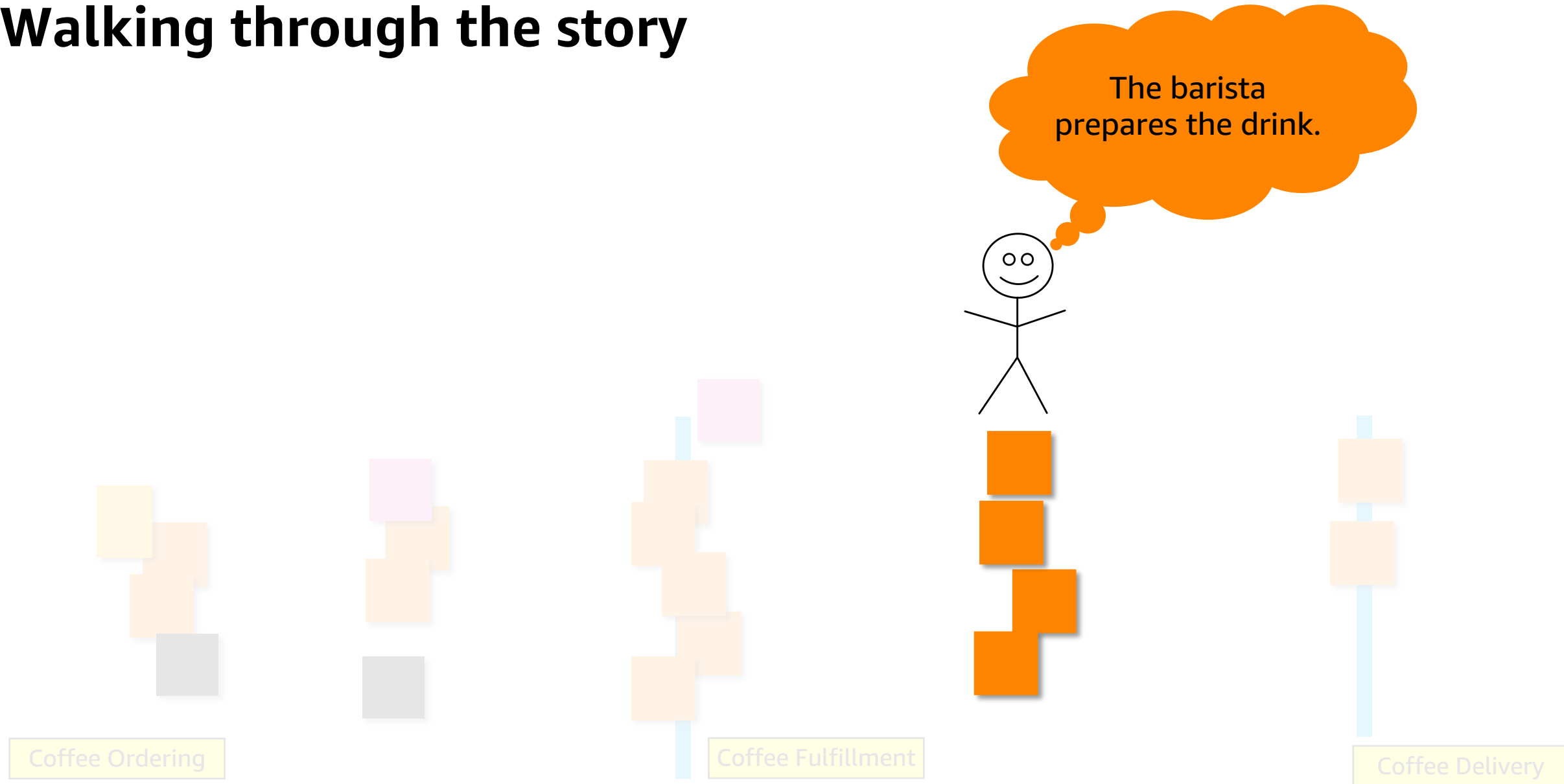


Coffee Fulfillment

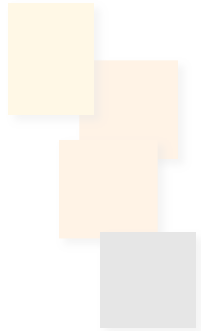


Coffee Delivery

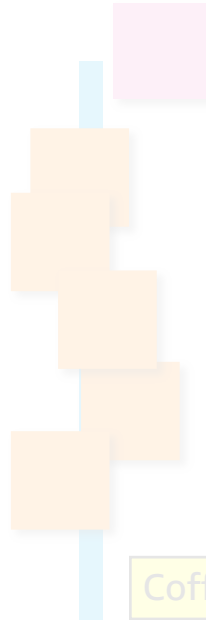
Walking through the story



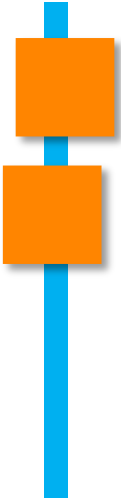
Walking through the story



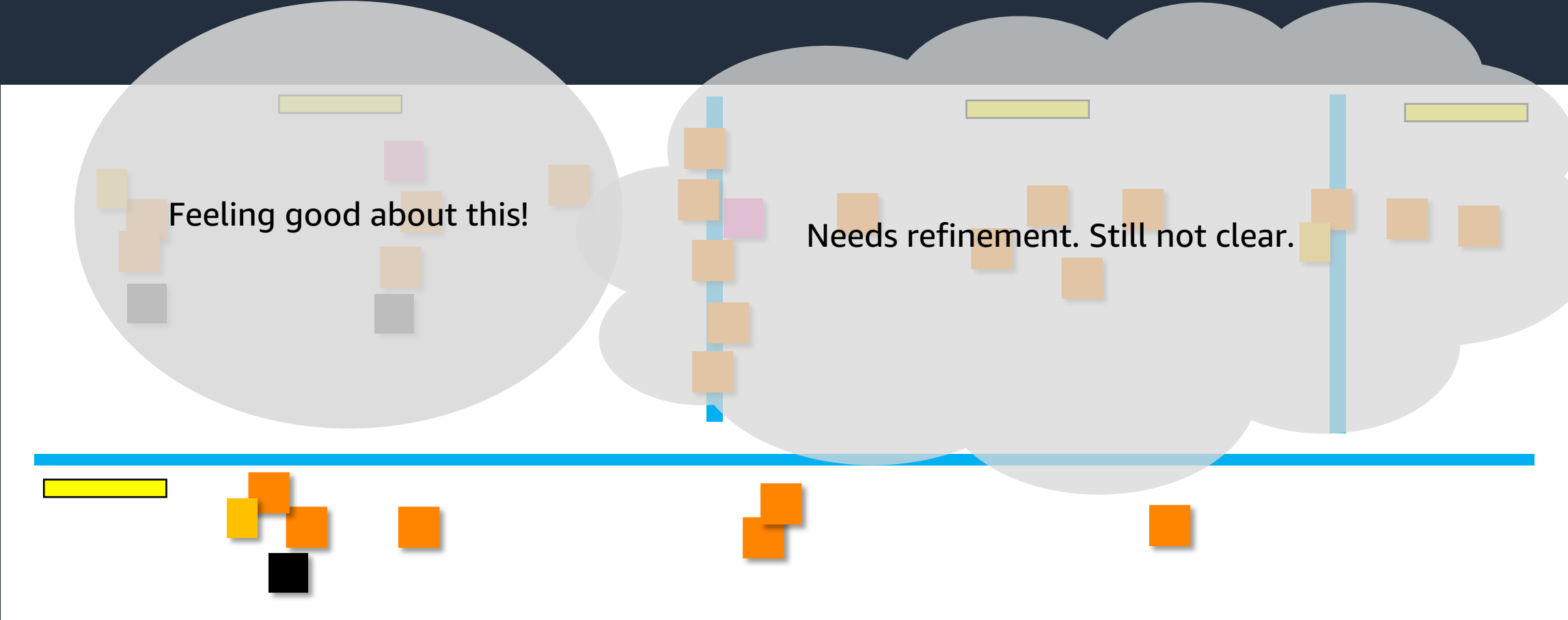
Coffee Ordering



Coffee Fulfillment



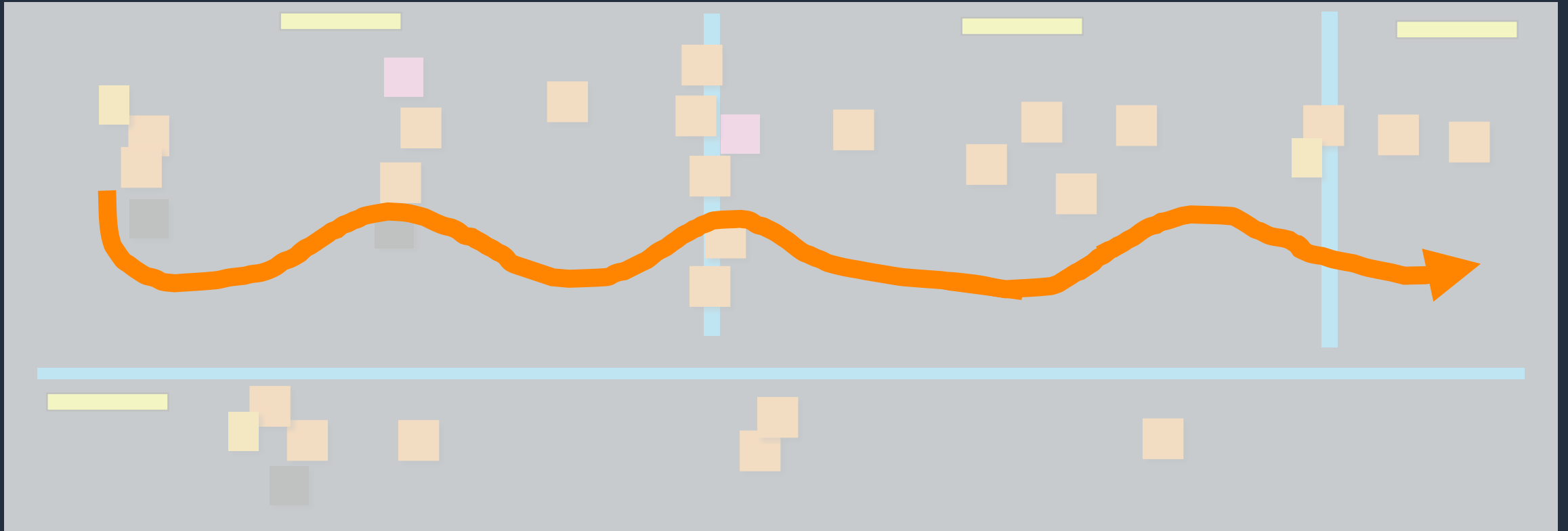
Coffee Delivery



Feeling good about this!

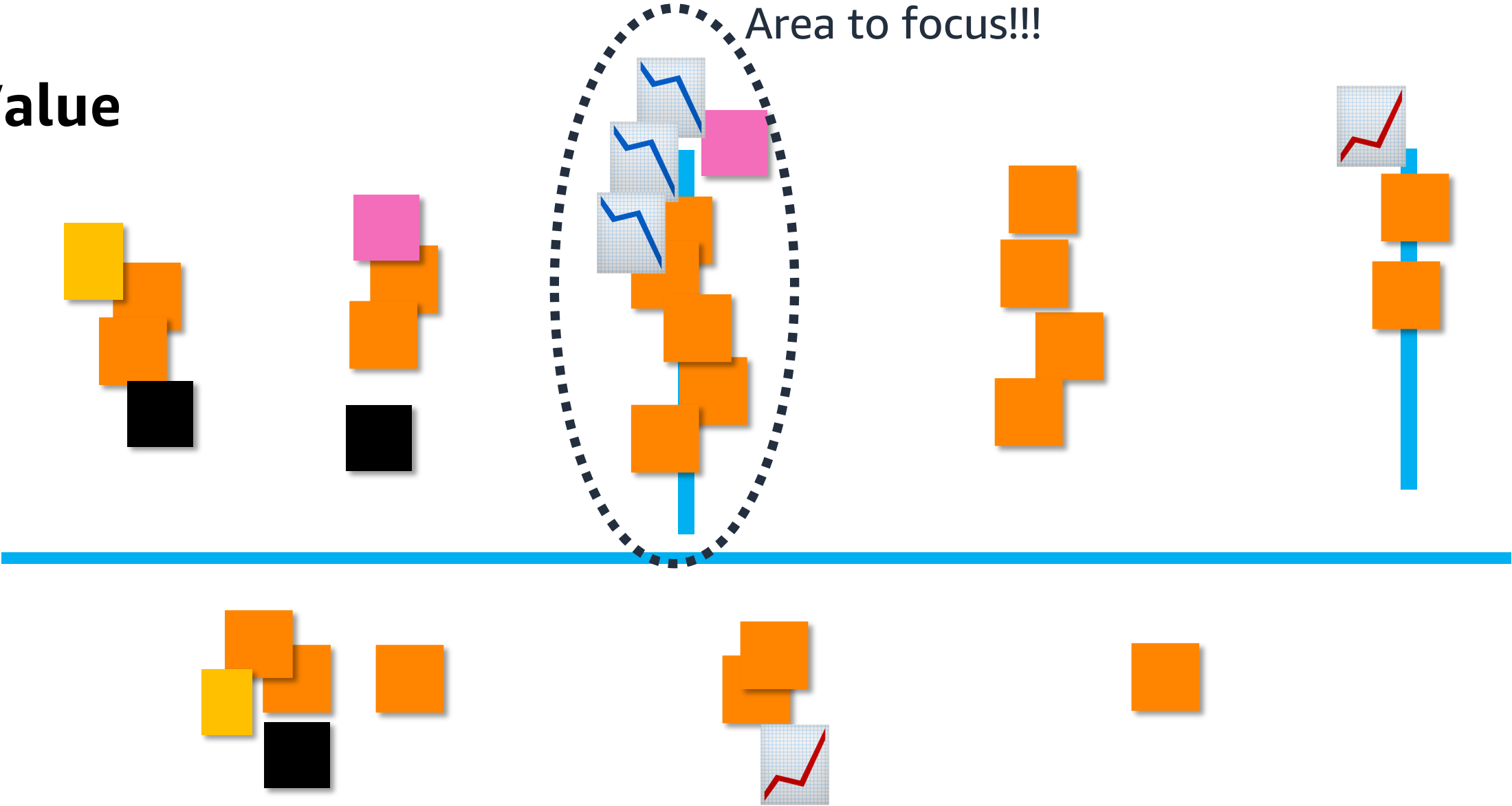
Needs refinement. Still not clear.

This is starting to flow and make sense



Value

Area to focus!!!



Discovering your bounded contexts

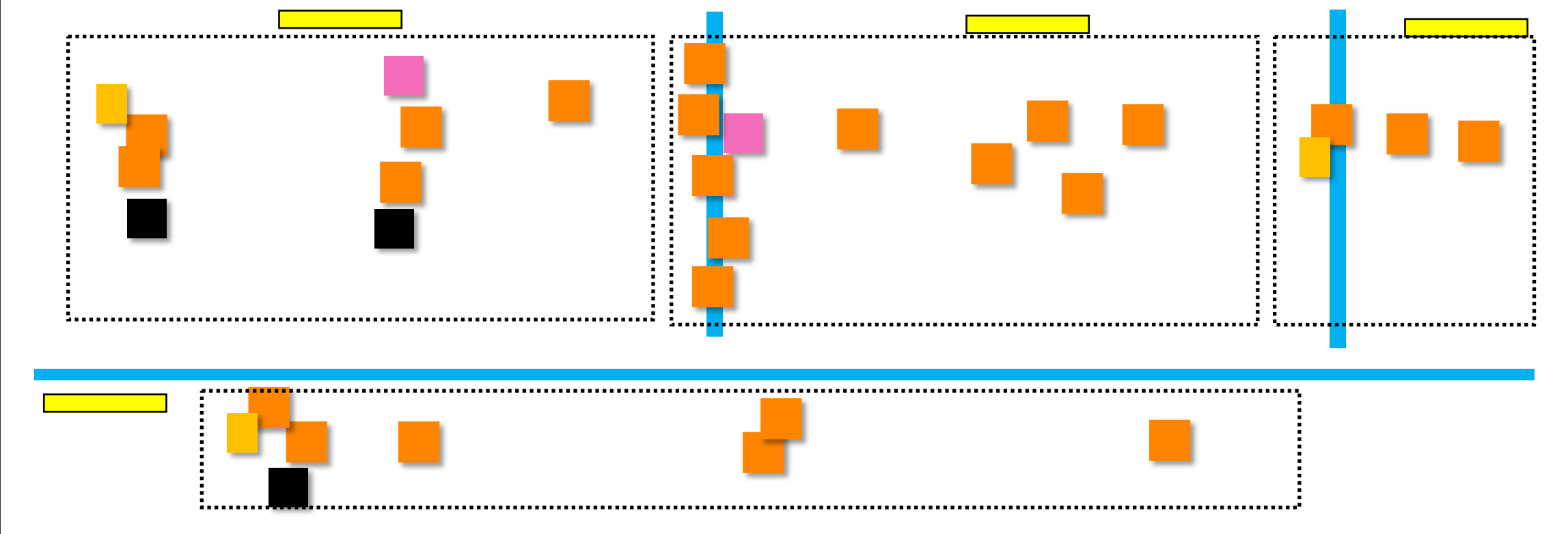
Getting your boundaries right is the most important design decision you will make.

“Ideally, a bounded context should contain a model tailored around a specific *purpose*: the perfectly shaped tool for one specific job, no trade-offs.”

Alberto Brandolini

Introducing EventStorming

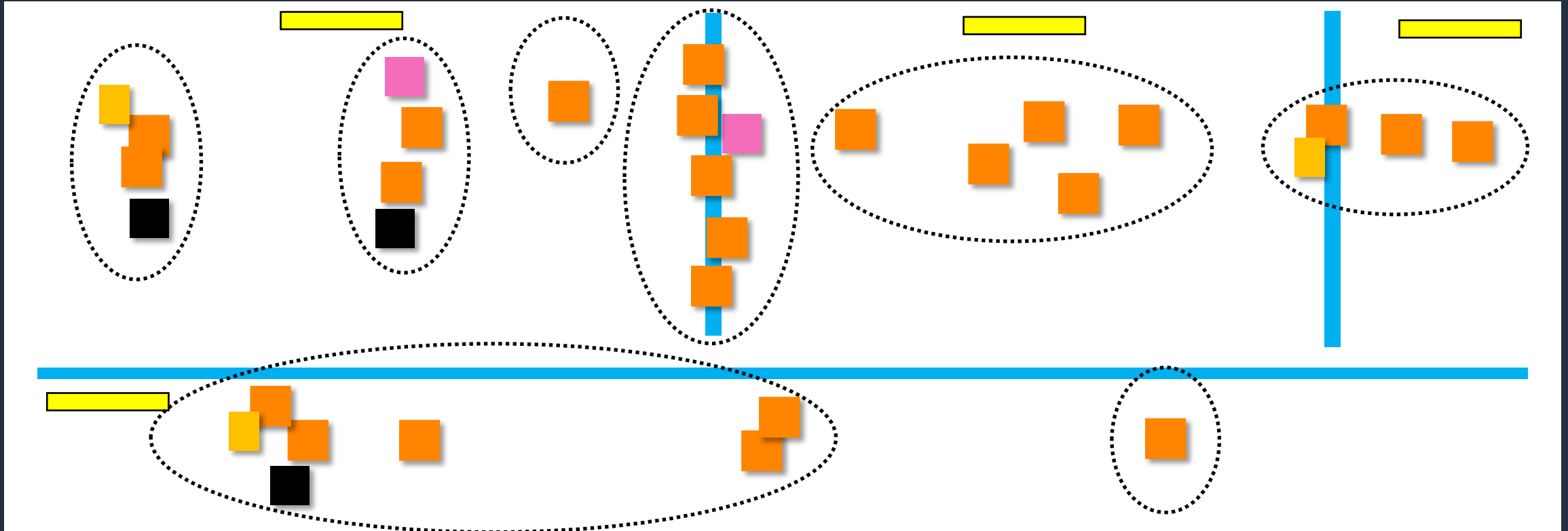
Identifying boundaries



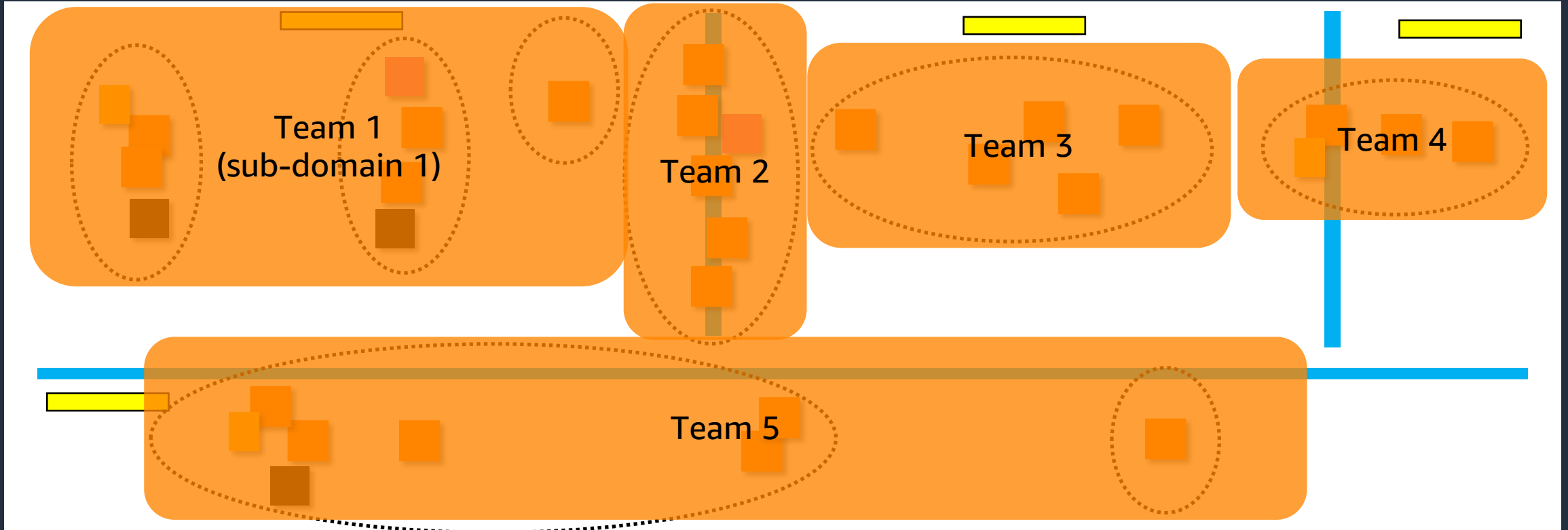
Tricks to identify context

- Look at the pivotal events
- Look at the people on the board
- Look at the language(s) people are using
- Look at the interaction between people

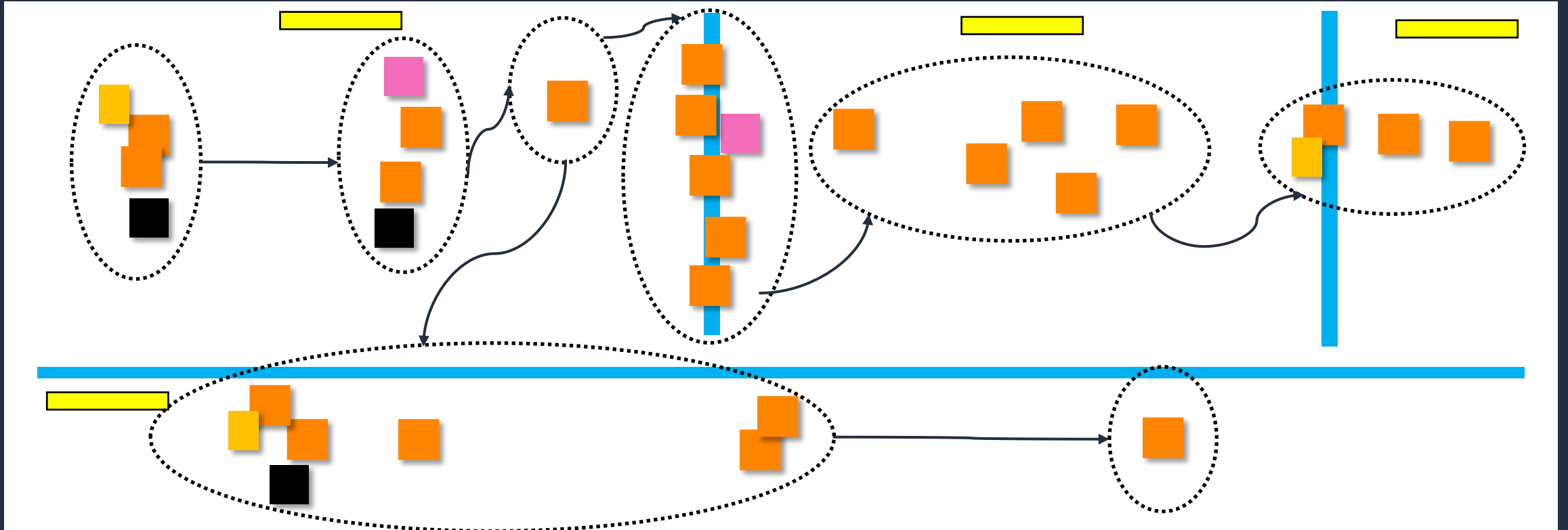
Identifying boundaries



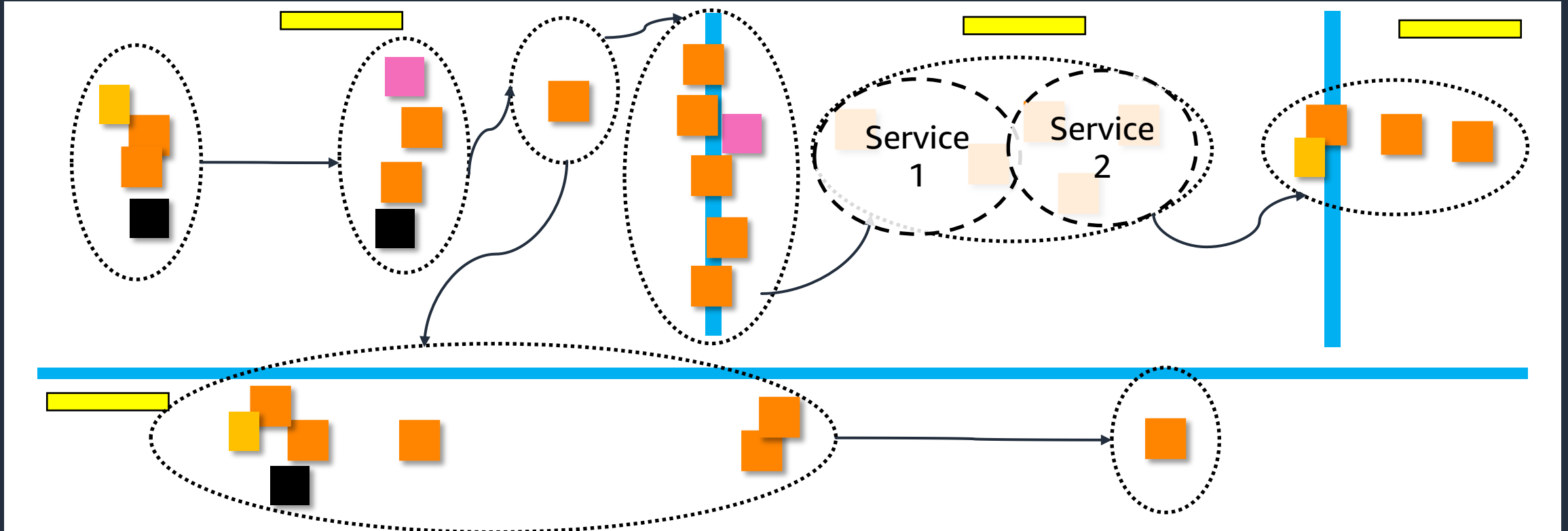
Identifying boundaries



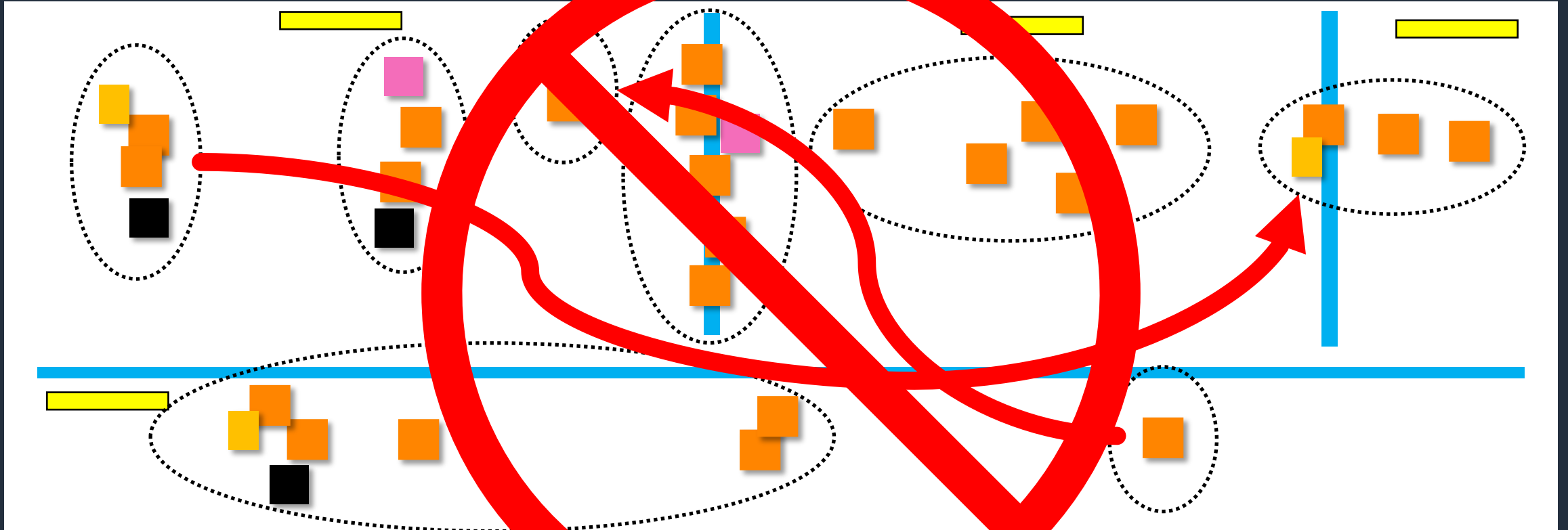
Building with EDA



Building with EDA



Building with EDA



Finding the right scope

Order Delivery

Order Delivery



Customer



[This Photo](#) by Unknown Author is licensed under [CC BY-SA](#)

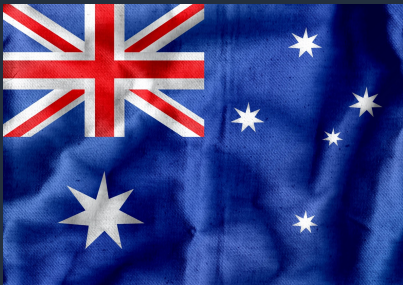
F-35 Delivery



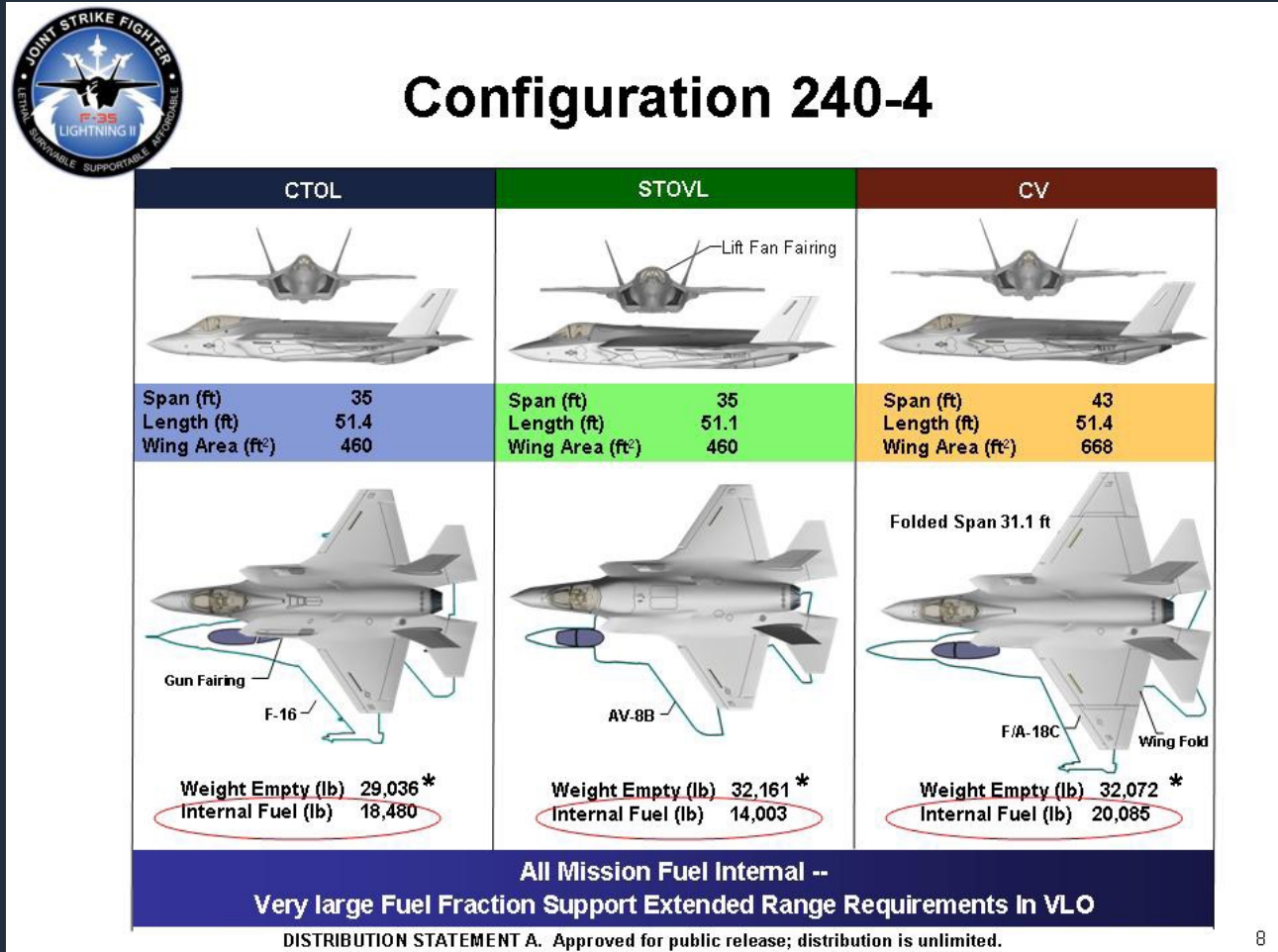
This Photo by Unknown Author is licensed under [CC BY-SA](#)



This Photo by Unknown Author is licensed under [CC BY-SA](#)

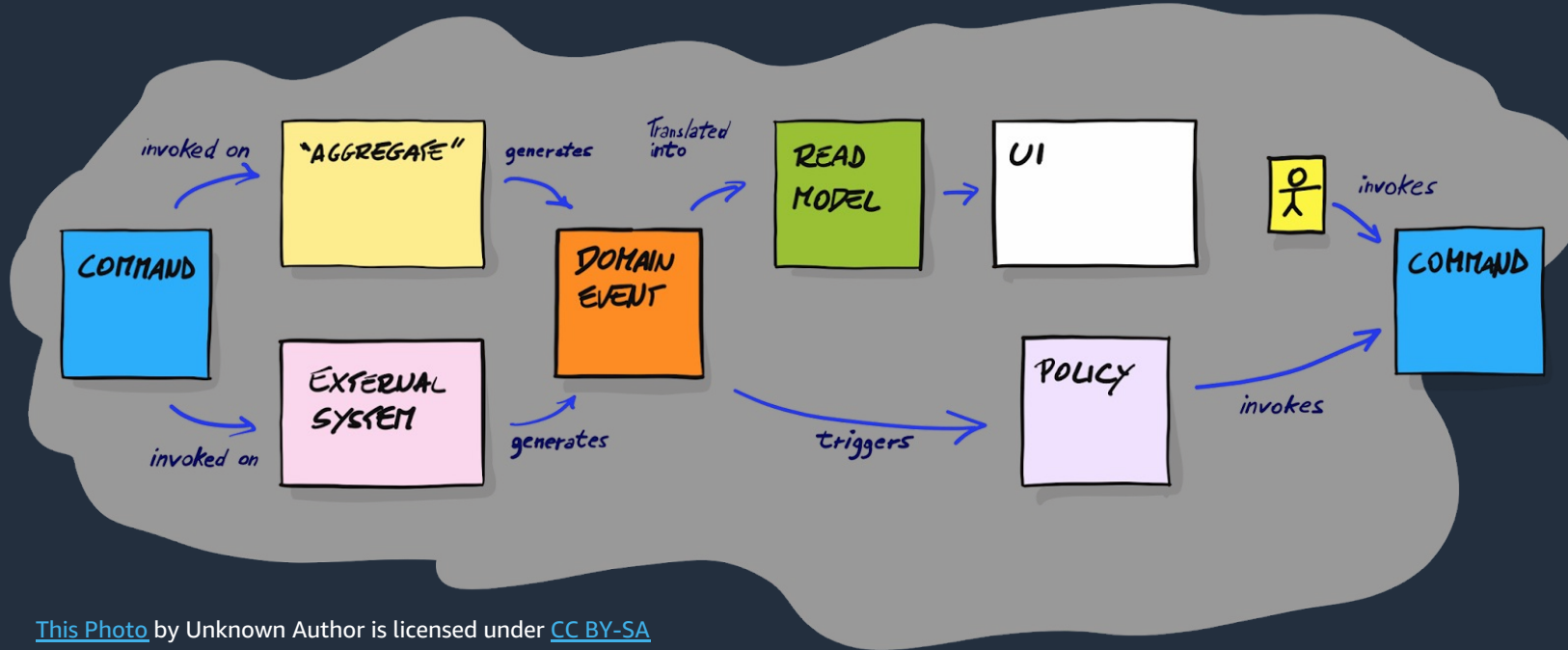


This Photo by Unknown Author is licensed under [CC BY-SA](#)

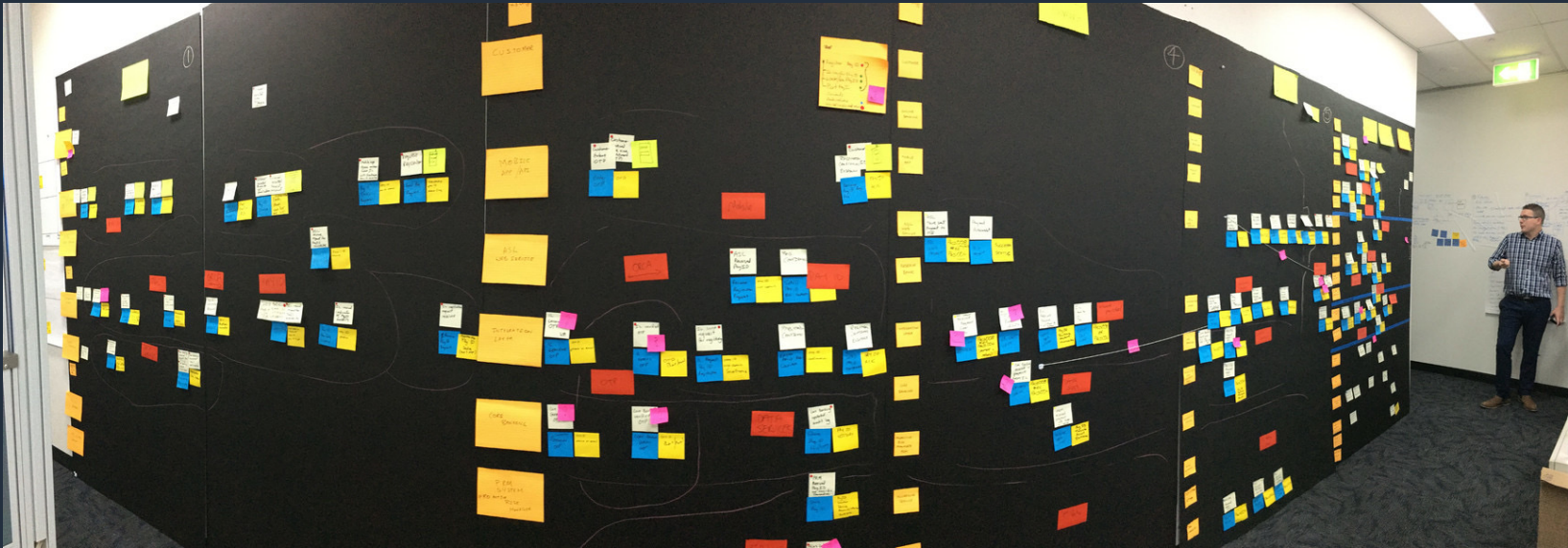


This Photo by Unknown Author is licensed under [CC BY-SA](#)

You probably need to break down
further than you think.



[This Photo](#) by Unknown Author is licensed under [CC BY-SA](#)



[This Photo](#) by Unknown Author is licensed under [CC BY-SA](#)

Introducing EventStorming

by Alberto Brandolini

<https://s12d.com/es>



The EventStorming Handbook

by Paul Rayner

<https://s12d.com/es-handbook>



Thank you!

Brian Zambrano

zambb@amazon.com

@brianzambrano