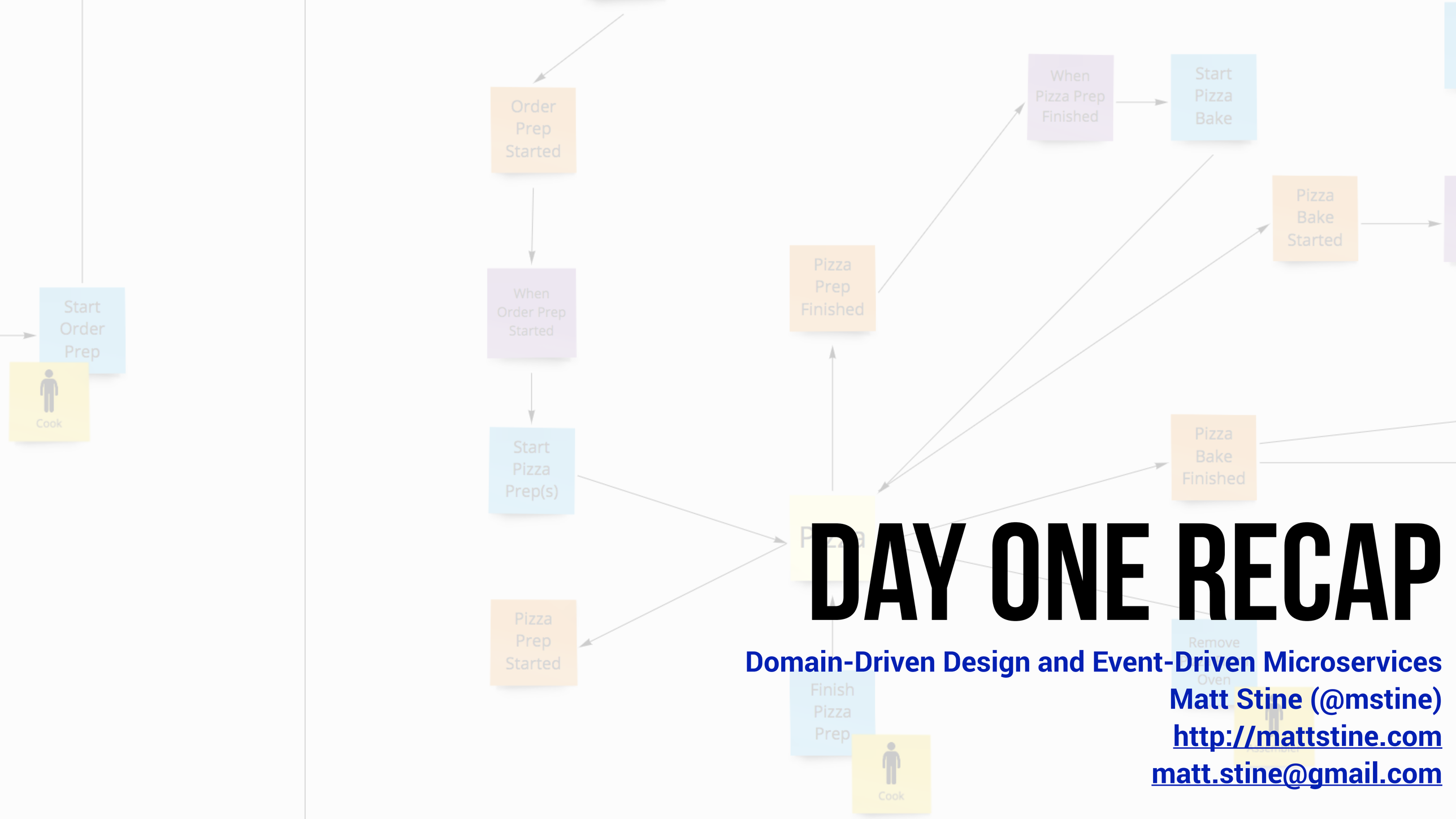


DAY ONE RECAP

Domain-Driven Design and Event-Driven Microservices

Matt Stine (@mstine)
http://mattstine.com
matt.stine@gmail.com

WHAT IS THE STATE OF MICROSERVICES IN 2022?

MICROSERVICES ANTIPATTERNS!


WHAT WE'VE GOT
HERE IS FAILURE TO
DESIGN

ON THE CRITERIA TO BE USED IN DECOMPOSING SYSTEMS INTO MODULES

D. L. Parnas

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pa.

August, 1971

FUNCTIONAL DECOMPOSITION

Changes

Struggles with knowledge spread across all modules, so often each module has to change in response to a desired functional change.

Independent Development

Dependent on shared data formats and schema. Must be jointly defined and agreed upon across multiple groups.

Comprehensibility

You need to know something about how all of the modules work to understand the whole system.

CAPABILITY DECOMPOSITION

Changes

Usually isolates a change to a single module.

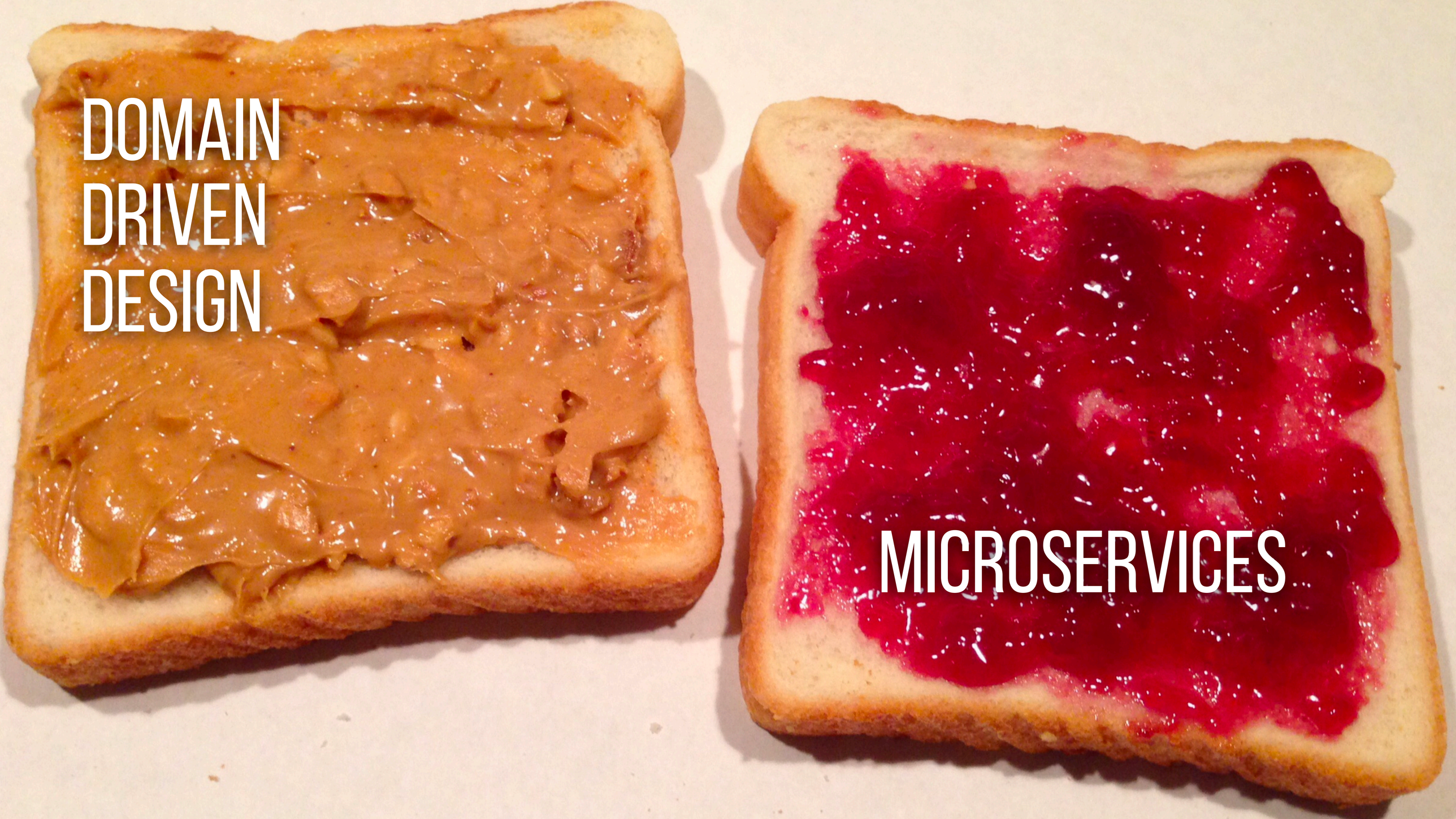
Independent Development

Has abstract interfaces that encapsulate the work to be done..

Comprehensibility

You can understand modules independently.

THE FUNDAMENTAL WAYS IN WHICH WE DESIGN MODULES EFFECTIVELY HAS NOT CHANGED IN THE LAST 50 YEARS.


DOMAIN
DRIVEN
DESIGN
MICROSERVICES

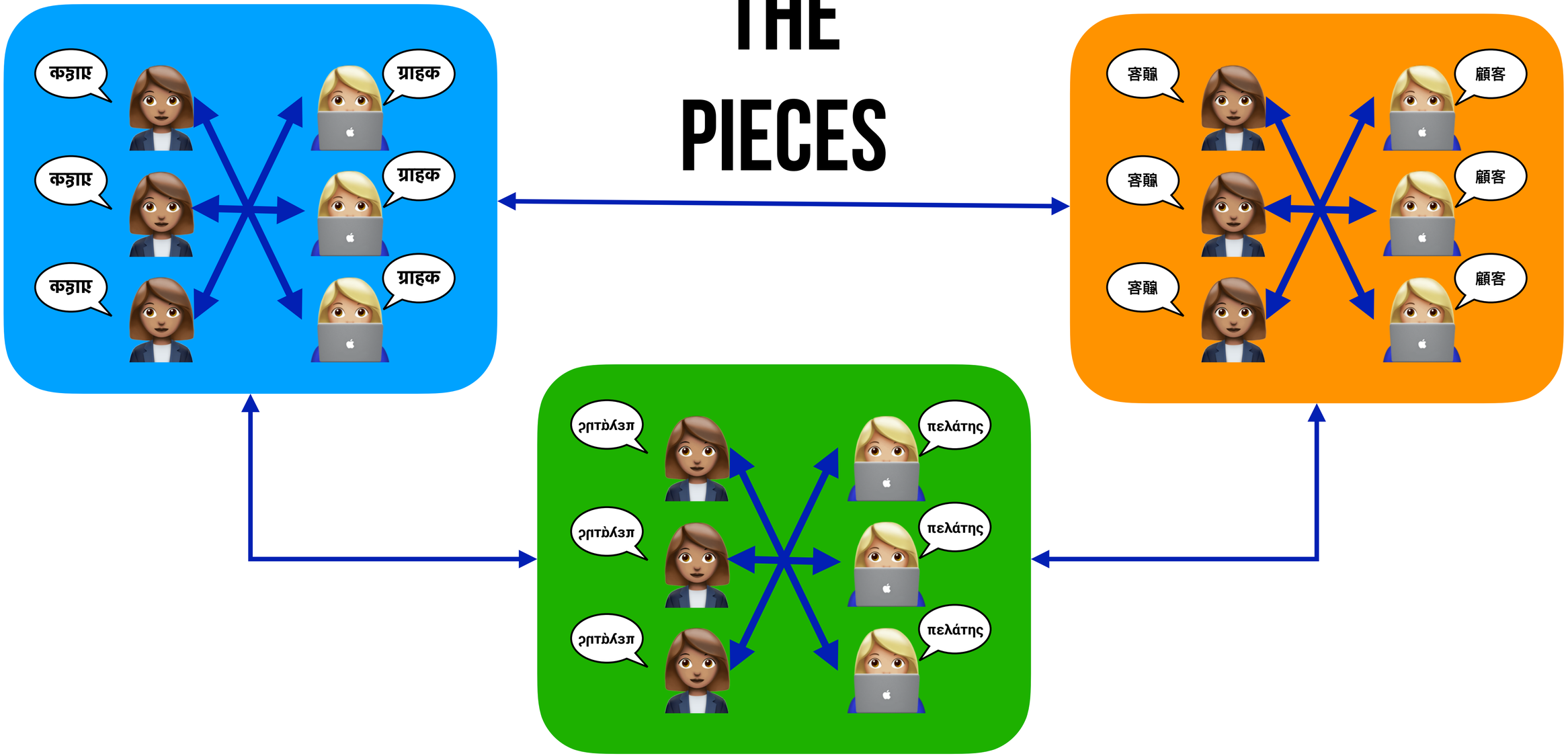
THE UBIQUITOUS LANGUAGE

When discussing the domain, and I use a term, and you use exactly the same term, we mean precisely the same thing.

STRATEGIC DESIGN - THE VIEW FROM 35,000 FEET

FINDING THE SEAMS

INTEGRATING THE PIECES



TACTICAL DESIGN?

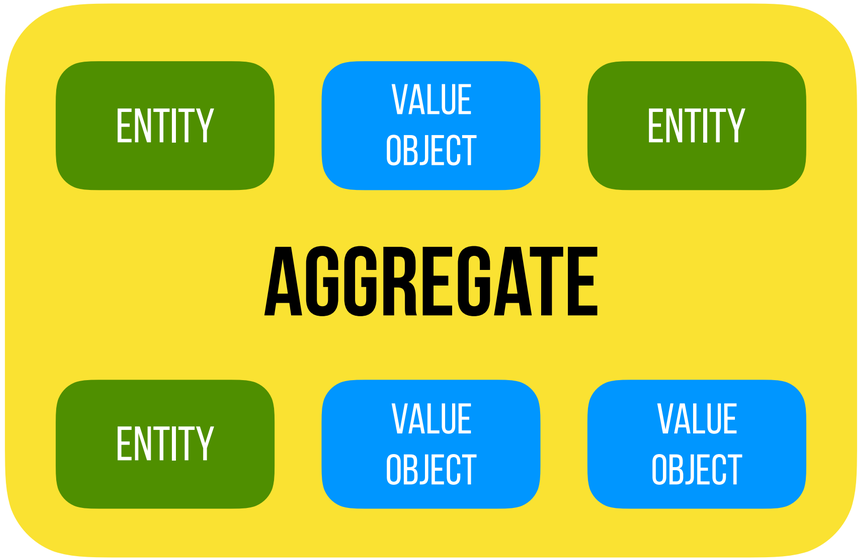
Now it's time to start speaking the Ubiquitous Language within each Bounded Context.

- Business Invariants
- Policies
- Transactions
- State
- Persistence

THE CENTRAL CONCEPT

A cluster of objects treated as a single unit.

Often modeled as a state machine.

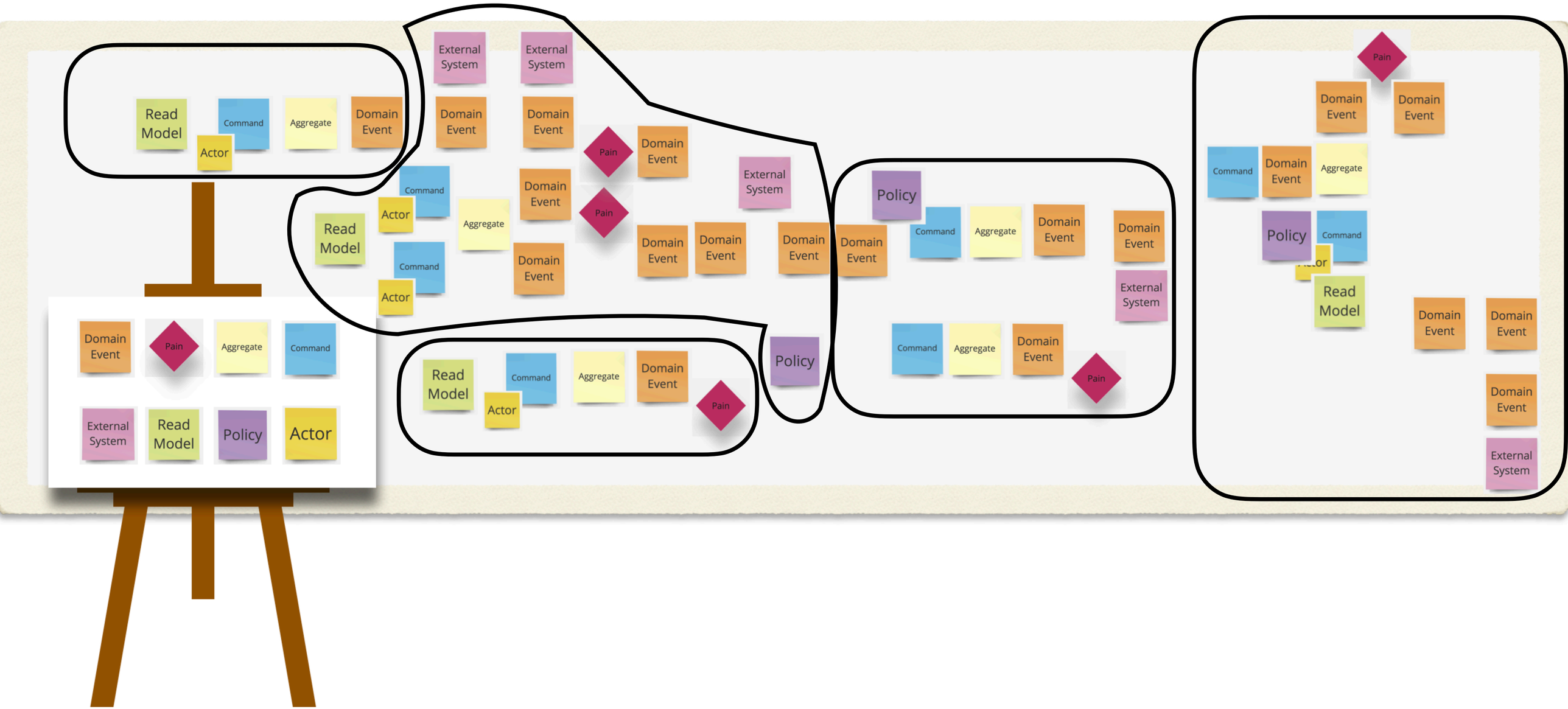


Only accessed through its Root Entity.

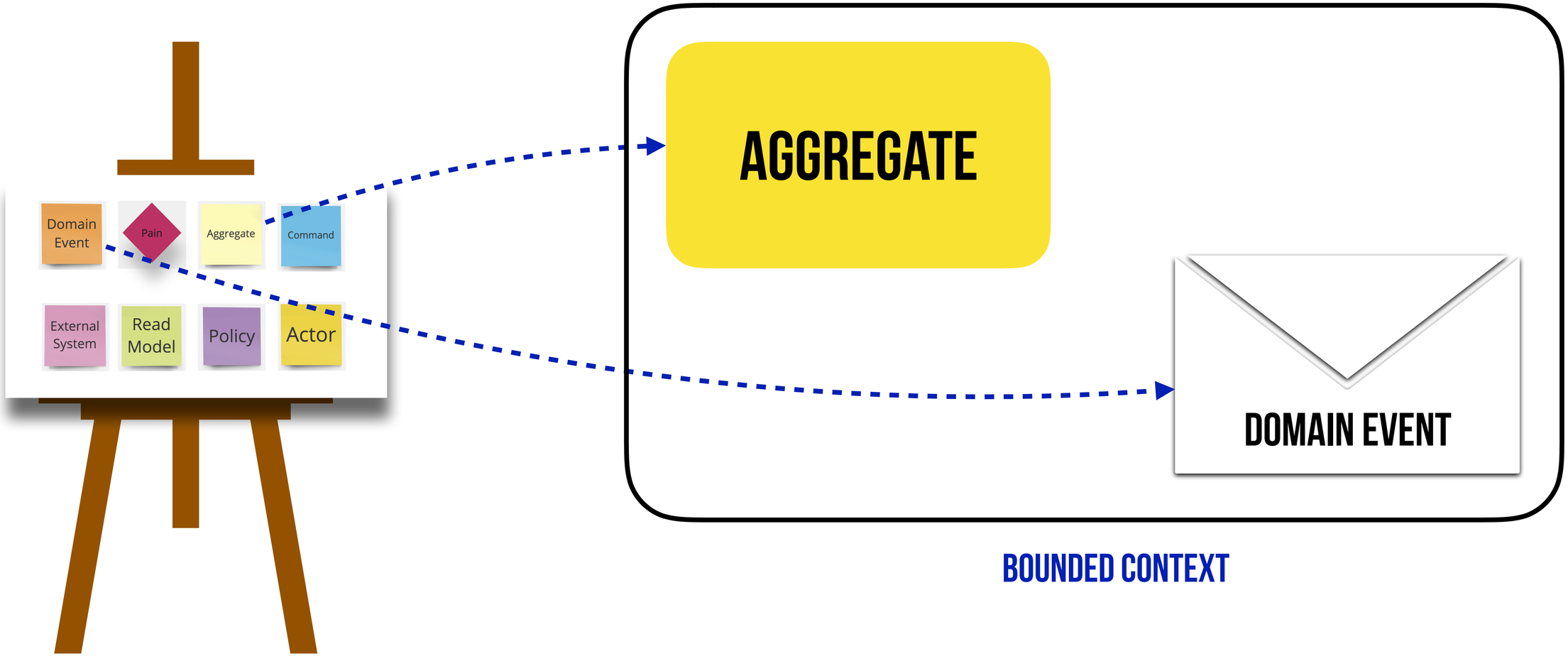
The atomic unit for any transactional behavior.

Responsible for maintaining any/all business invariants.

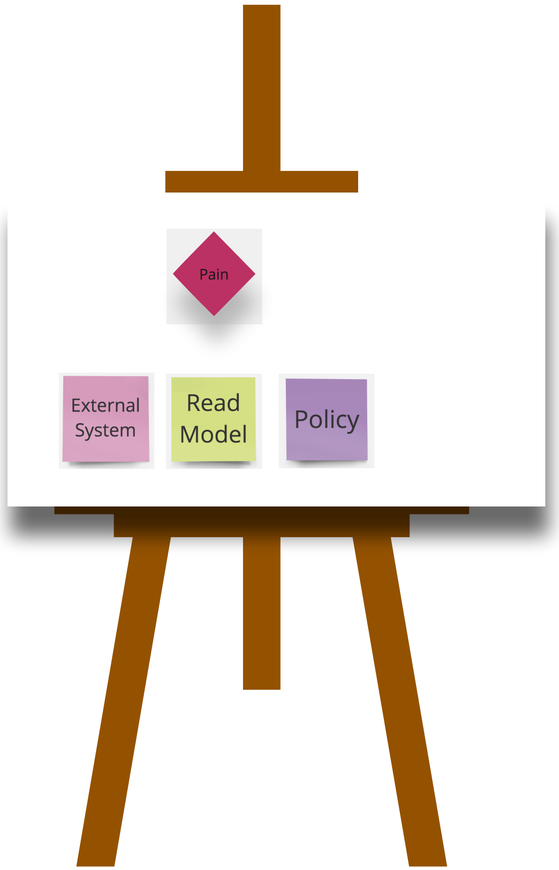
EVENT STORMING



FROM EVENT STORM TO DDD



USER STORIES

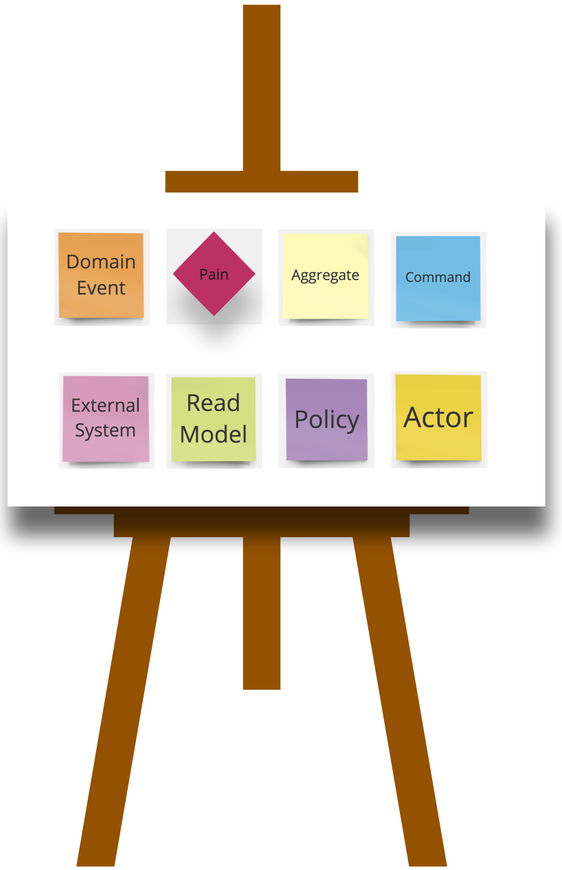


As a ROLE Actor

I want to ACTION Command Aggregate

So that I can GOAL Domain Event

WHAT ABOUT TESTS?

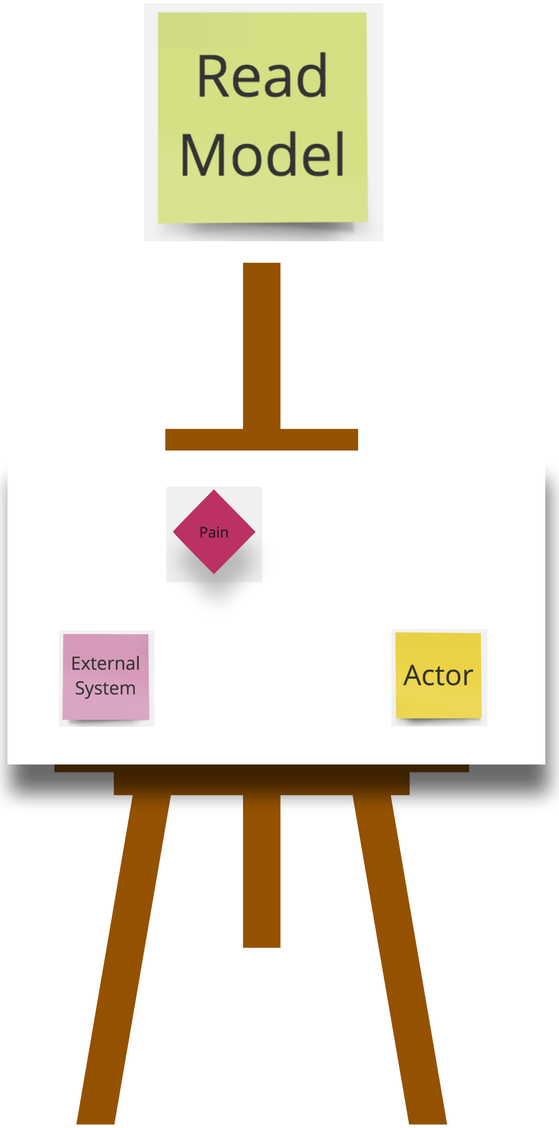


- Aggregate Business Invariants:
  `should_not_allow_placeOrder_without_a_lineItem()`
- Aggregate Domain Events:
  `should_fire_event_when_order_placed()`
- Policies:
  `should_check_inventory_when_order_placed()`

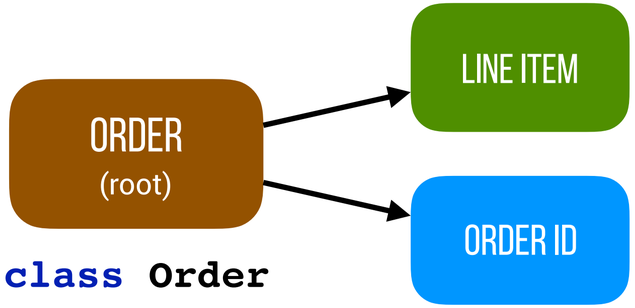
WHAT ABOUT CODE?

REPOSITORY

```
eventLog.subscribe(ProductAddedEvent,
e -> {});
```



Aggregate




```
class OrderLineItem
```

```
class Order
```

```
class OrderId
```

Command

```
void placeOrder()
```

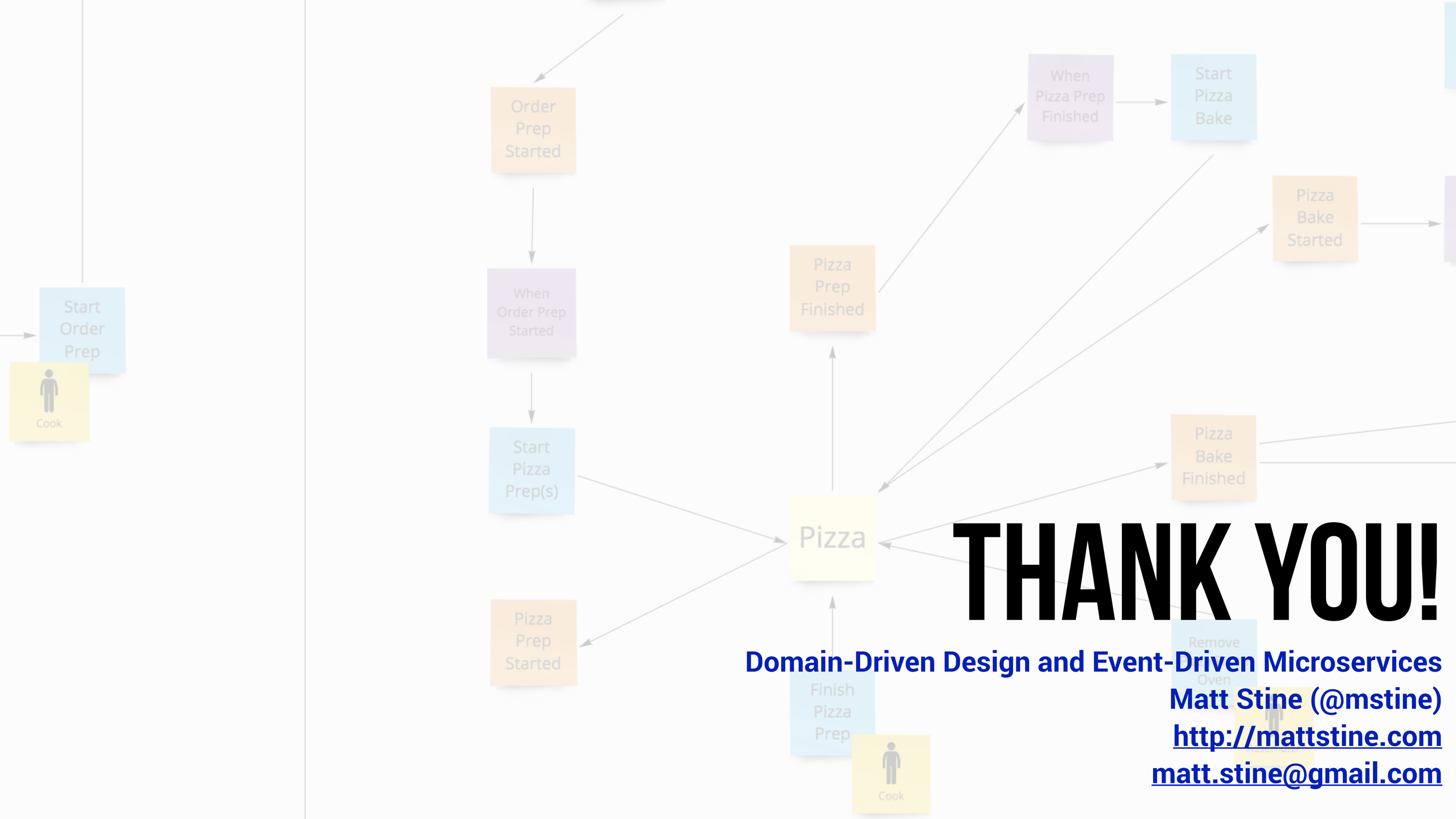
Policy

APPLICATION SERVICE

```
eventLog.subscribe(OrderPlacedEvent,
e -> {});
```

Domain Event


```
class OrderPlacedEvent
```



THANK YOU!

Domain-Driven Design and Event-Driven Microservices

Matt Stine (@mstine)

http://mattstine.com

matt.stine@gmail.com