Why DDD Matters Now

@ziobrando

About me

- Coding since 1982
- Into Domain-Driven Design since 2005
- Running www.avanscoperta.it
- Modelling (almost) everything with sticky notes, markers and a paper roll.
- Calling this stuff EVENT STORMING

What is software development?

Idea Nº 1

"Writing software is like building a house"

#PROTIP: THEY'RE NOT *REALLY* BUILDING A HOUSE

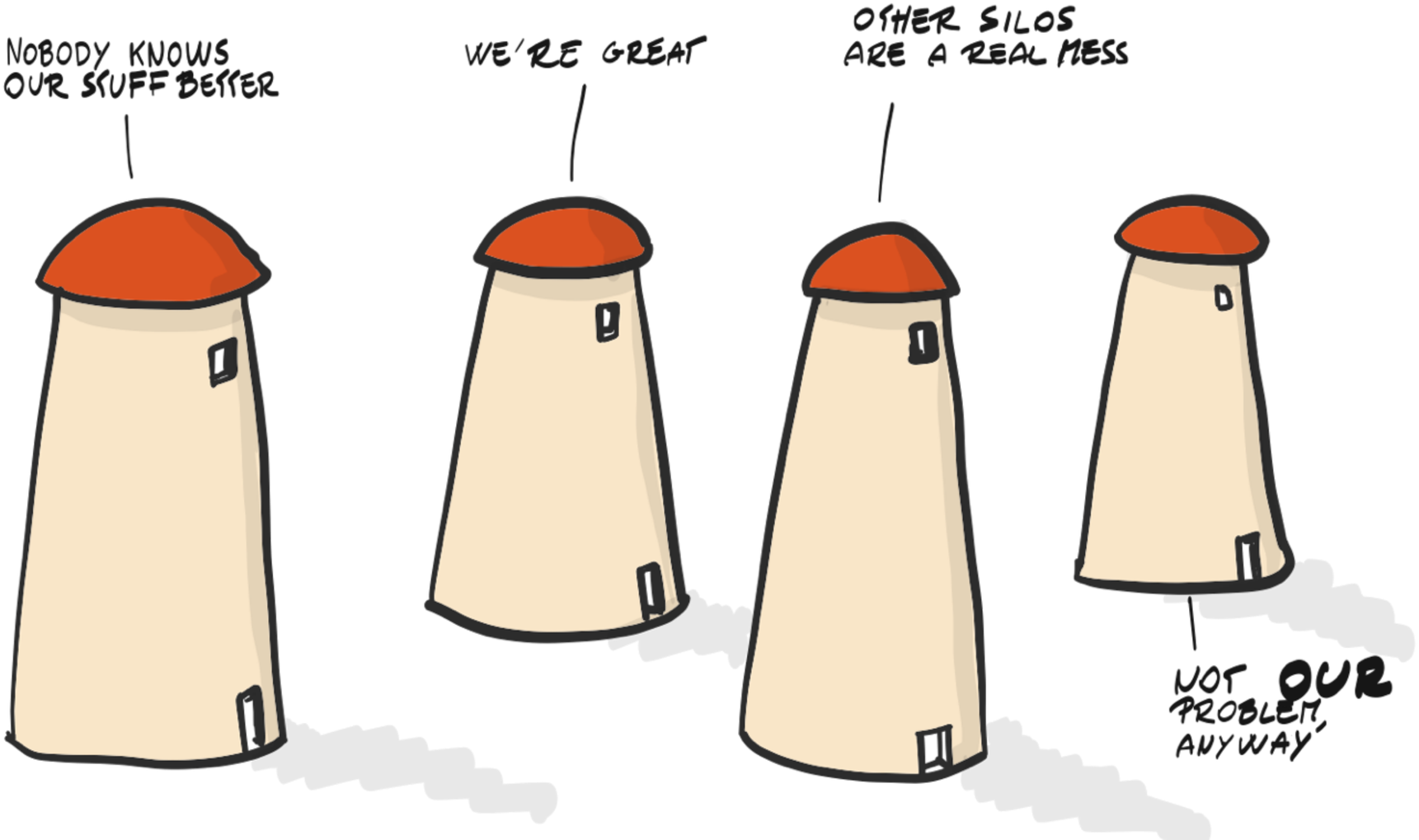
WHEN WAS THE LAST TIME
YOU UPGRADED BRICKS?
imgflip.com

"We are translating requirements into working software"

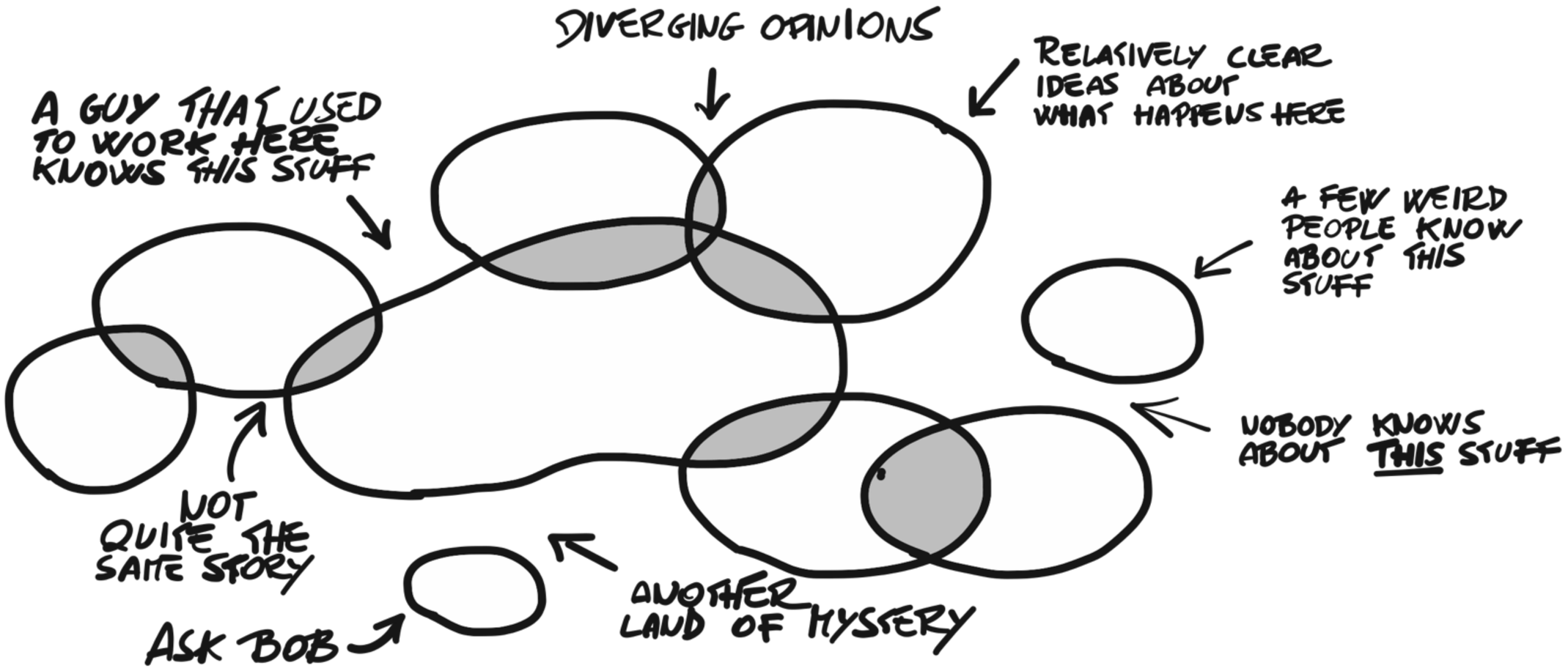
Our eternal Gratitude
The benevolent wisdom of the domain expert

A SINGLE PERSON WILL NEVER TELL THE ENTIRE STORY

The shape of the organization



THE KNOWLEDGE DISTRIBUTION



*“It's a basic truth of the human condition that **everybody lies**. The only variable is about what.”*

Dr. Gregory House

...OR WE MIGHT DISCOVER THAT CULTURE MATTERS AND COMMUNICATION IS NOT AS RELIABLE AS WE THINK

Perfect Metaphor in Europe

...or maybe just a cliche'

Completely useless here

Enterprise software development is discovery, challenge, and investigation

"SOFTWARE DEVELOPMENT IS A LEARNING PROCESS

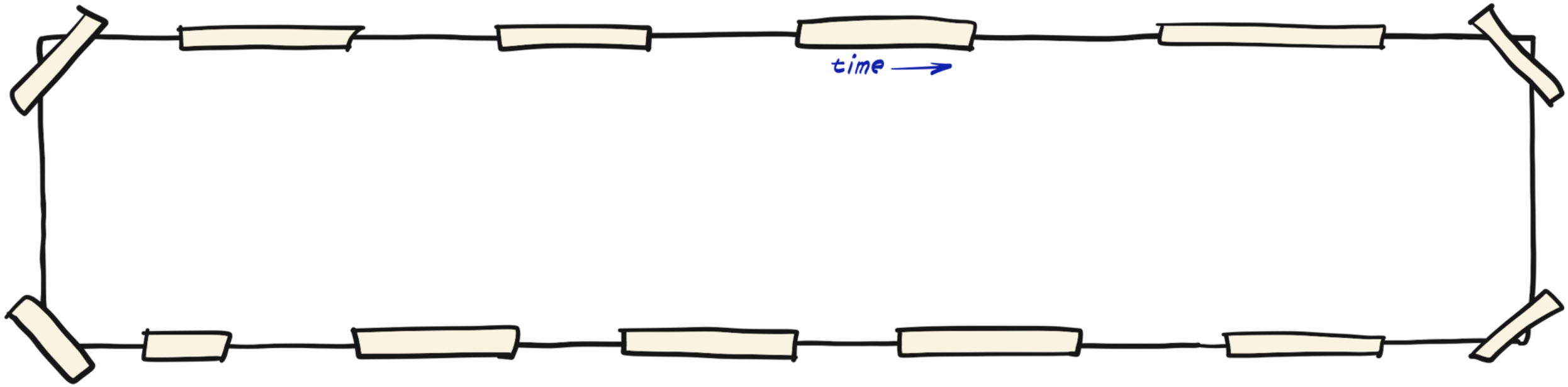
WORKING CODE IS A SIDE EFFECT"

This is where
EventStorming
Fills in...

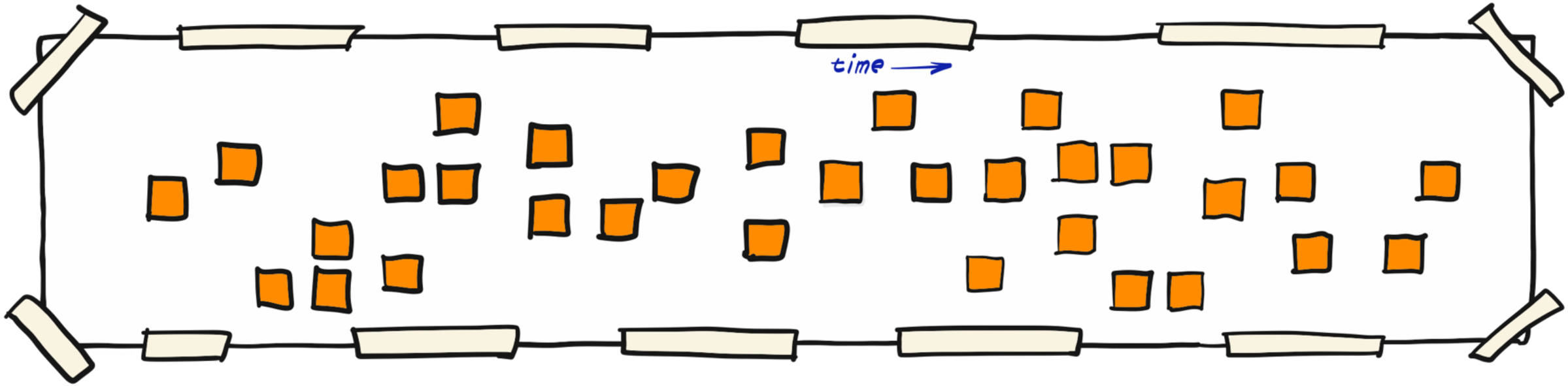
Big Picture Workshop

- Invite the right people -> Business, IT, UX
- Provide unlimited modelling space
  - Surface, Markers, stickies
- Model an entire line of business with Domain Events

Establish a timeline



Explore with Domain Events

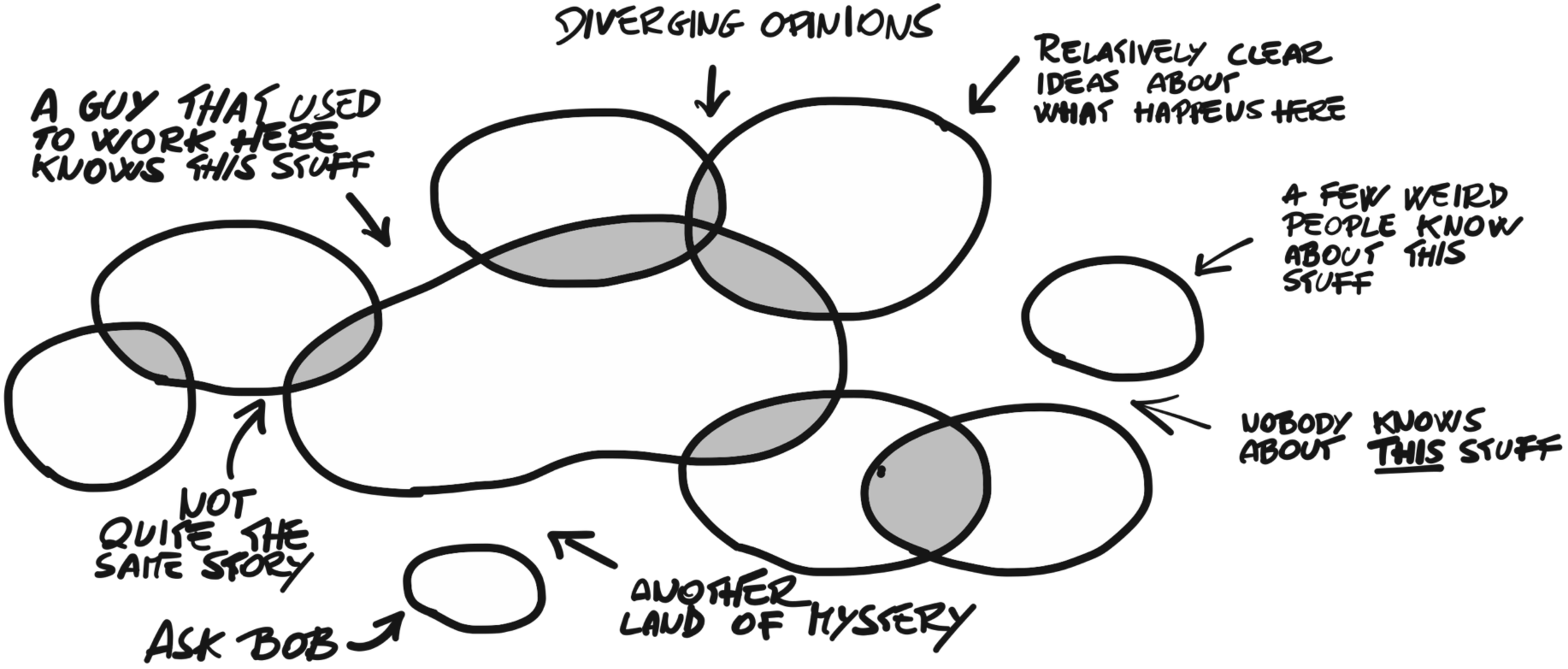


ITEM ADDED TO CART

This is a **Domain Event**

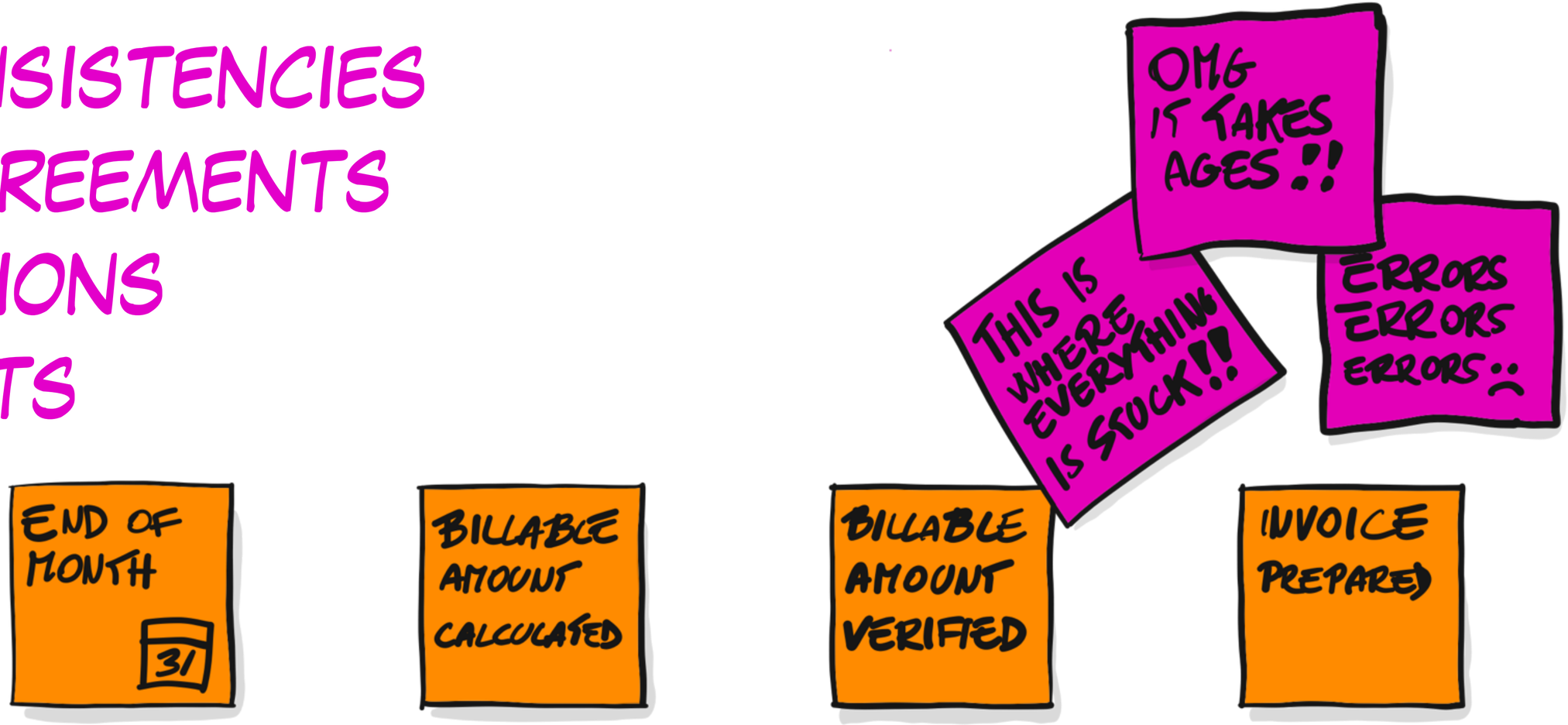
- **Orange** sticky note
- Verb at **past tense**
- **Relevant** for domain experts

SINCE YOU HAVE THIS...



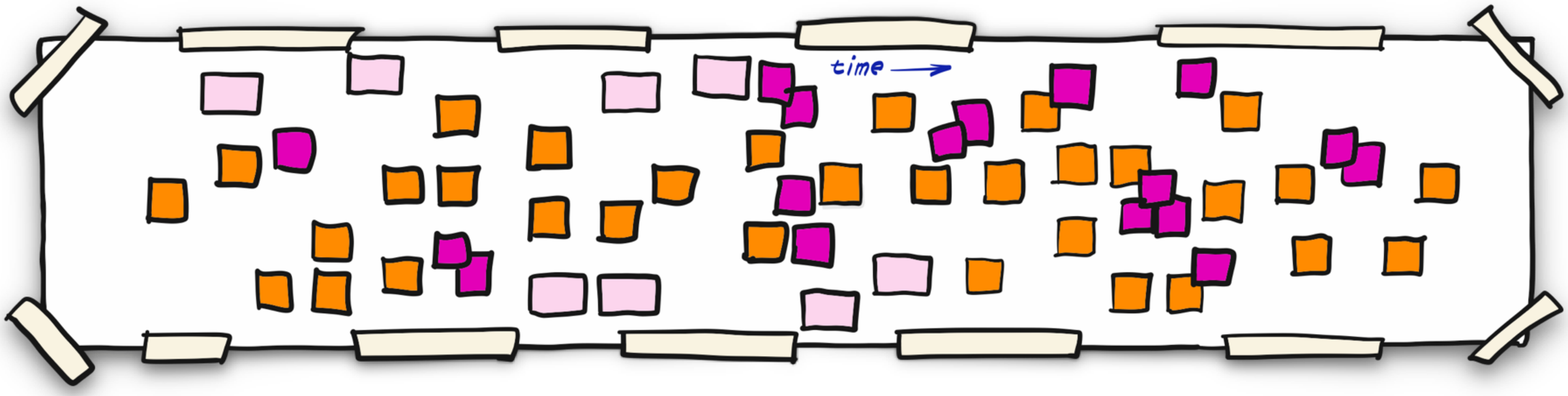
...YOU'LL HAVE THAT!

- INCONSISTENCIES
- DISAGREEMENTS
- FRICTIONS
- DOUBTS



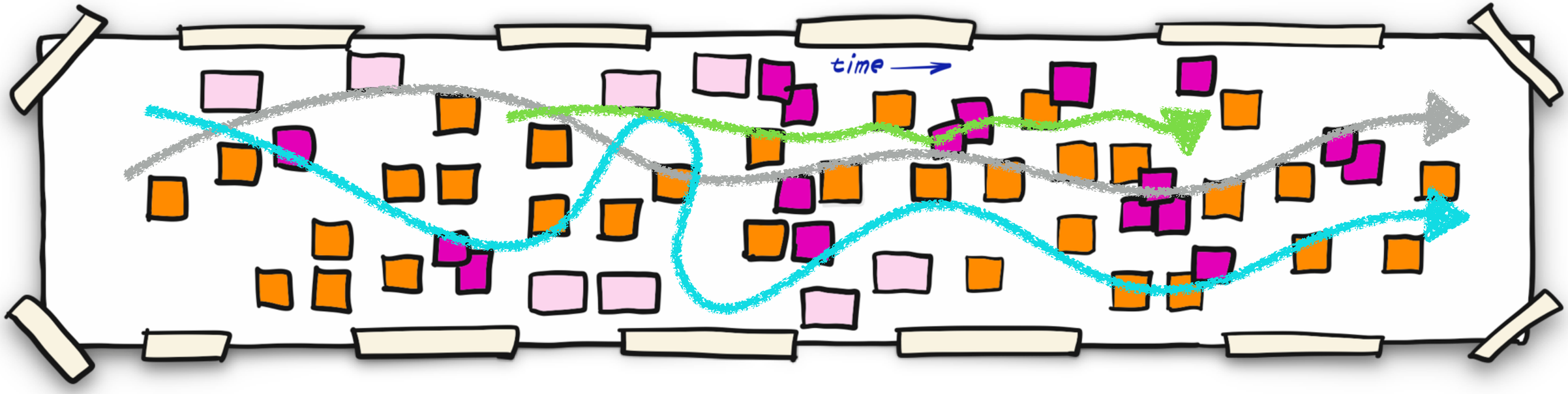
And that's great!!!

Outcome (big picture):



- The whole process is visible
- Massive learning (crossing silo boundaries)
- consensus around the <u>core problem</u>

Outcome (Big Picture):



- Multiple storytellings
- Incremental Notation #NoUML #NoBPMN
- A language for different tribes #Lean #UX #Agile #SW

More specifically...

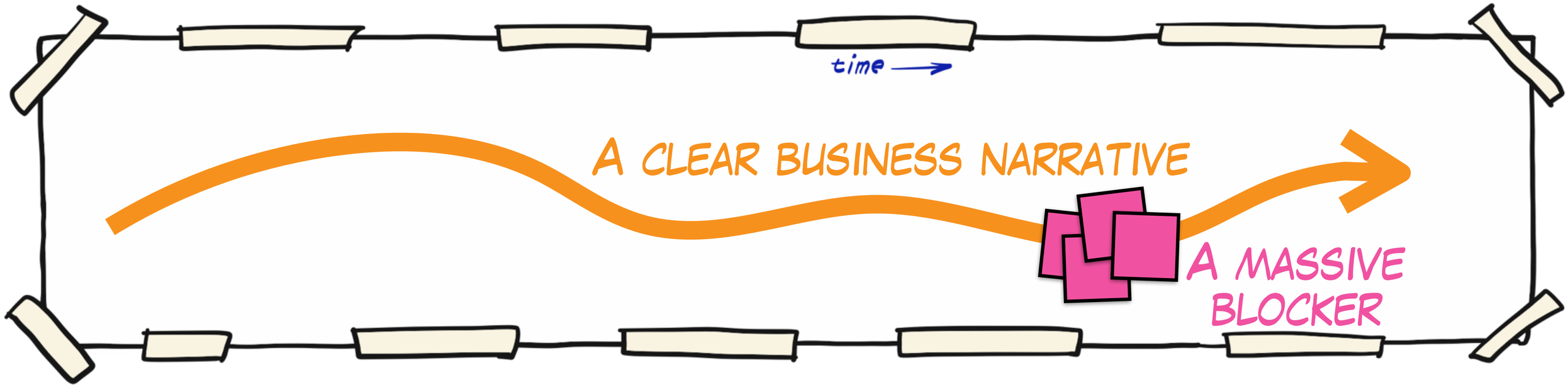
- No scope limitation (paper roll)
- Exploration of boundaries (External Systems & People)

- -> The bottleneck is in the picture.
- -> The core domain is in the picture

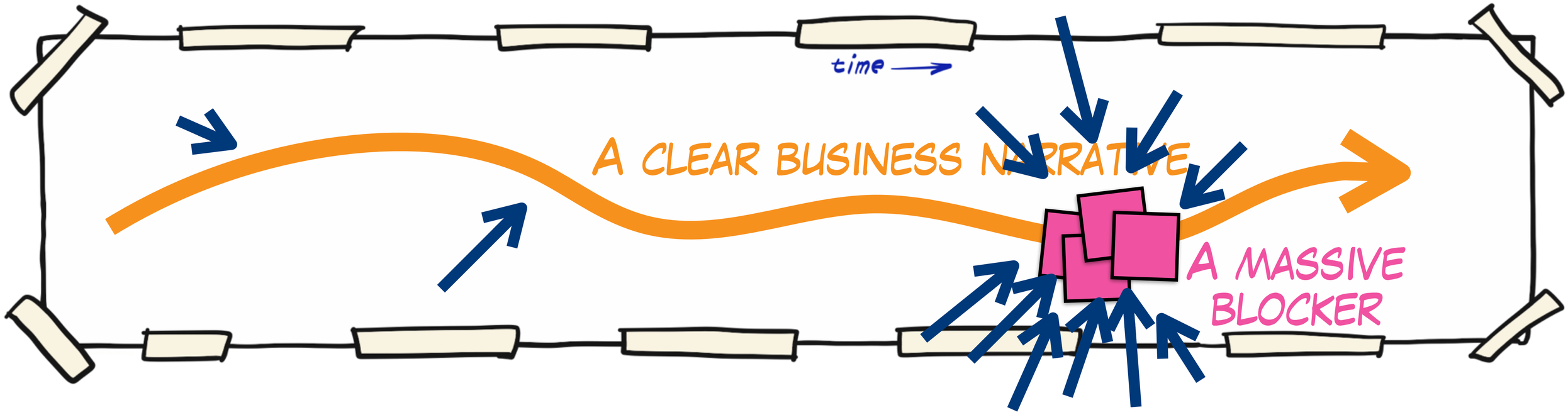
CLARITY

INCLUDING WHAT WE REALLY DON'T KNOW.

The outcome:



Arrow Voting



a swift kick
in the butt
$1.00

Not all software development is the same

"Core domain"

https://www.spreadshirt.de/selbst-gestalten?product=169608967&view=1#?affiliateId=1246955&orgn=CYO&netw=LI

I DELIVER SOFTWARE
Without understanding it

Delivery without understanding

- Pretty dumb idea
- Dominant pattern in enterprise software development

In the Core Domain,
Learning is the key asset

Strategies for non-core

- Traditional on-time / on-budget pseudo-agile thing
- Follow the backlog
- Avoid taking unnecessary risks
- No time for "gold plating"

Standard Approaches Won't deliver in the CORE

Strategies for CORE

- Creative collaboration with domain experts
- Experiments & Iterations
- Continuous refinement of knowledge and implementation glued together by Ubiquitous Language

There is no such thing as "Gold Plating" in the Core Domain

It's "Gold Making"

DDD is for domains that can't be modelled right in one shot

1) Too complex to be understood without experimenting

2) Continuously Evolving

Continuous Rewriting Needs solid Architecture Foundations

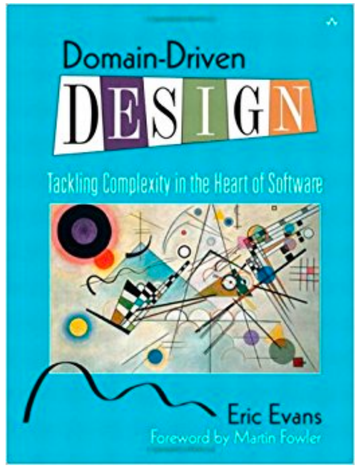
DDD

IS

NOT AN ARCHITECTURE

DDD Needs Good Architecture(s) Badly

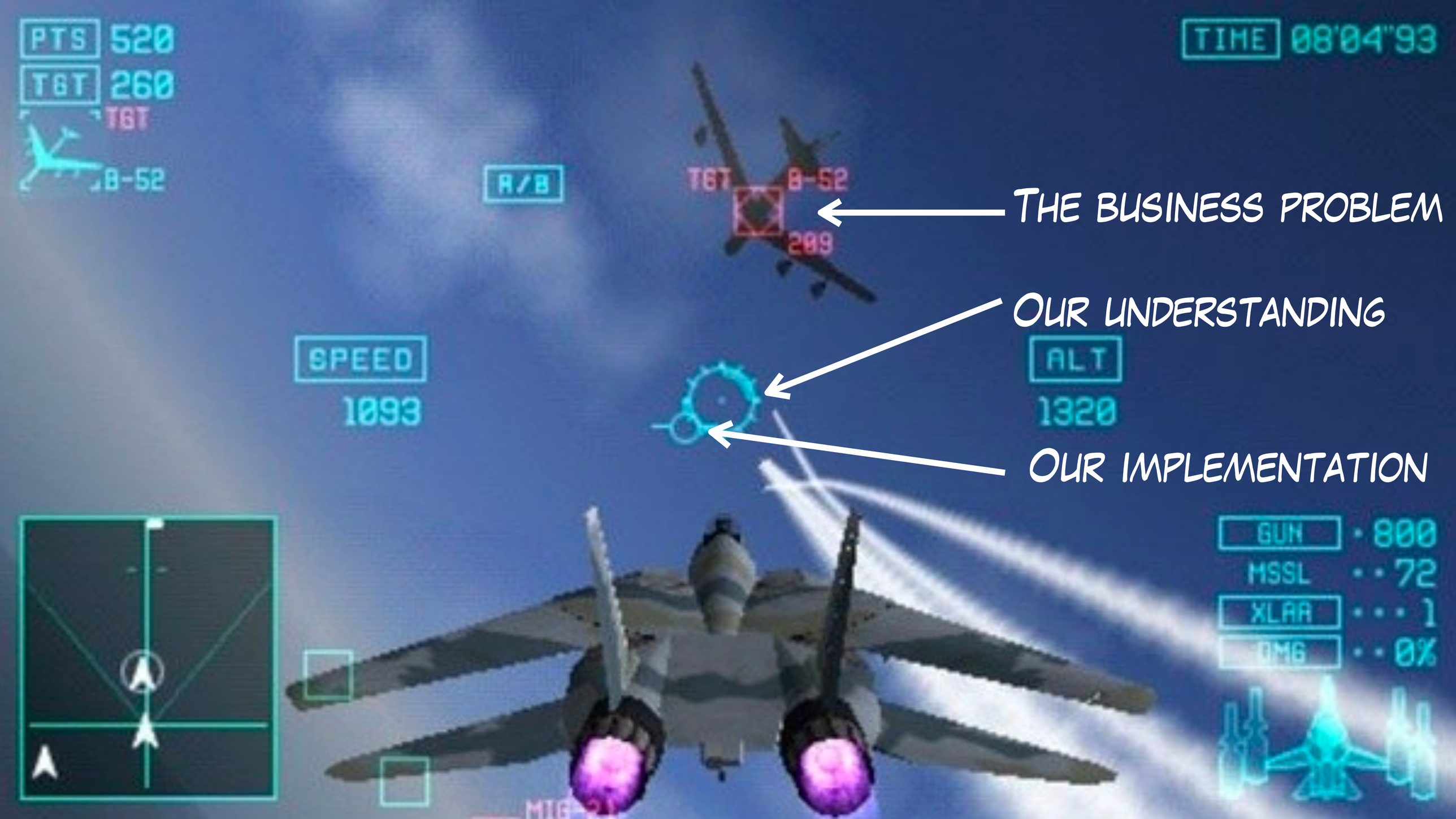
Only One Viable option in 2003



Plenty of alternatives available now:
CQRS, EventSourcing, MicroServices, Actors, Functional, ...

"Chasing the Right Model"

AKA "Creative Collaboration"


PTS 520
TGT 260
TGT
B-52
A/B
TGT B-52
209
TIME 08'04"93
SPEED
1093
ALT
1320
GUN 800
MSSL 72
XLAR 1
DMG 0%
The business problem
Our understanding
Our implementation

If you can keep the three close...

Easy!

1) Understand the problem

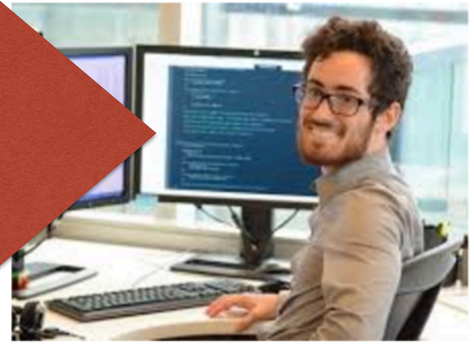
What happens when you google "software developer" for images?

(No idea about what happens here...)

maturity model | agile | gartner | idc | gantt | wbs

GEARS?

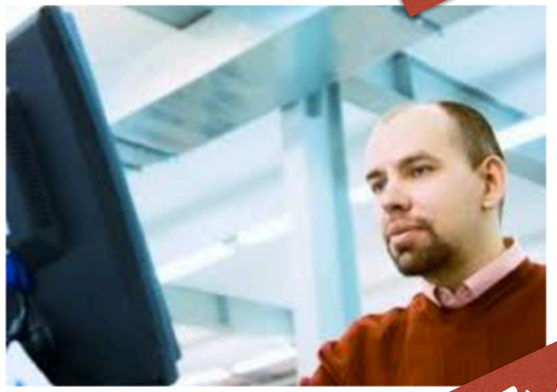
SLACKING OFF



CLUELESS DUDE IN THE MIDDLE

MY MAN!

WATCHING PORN



EXACTLY!!

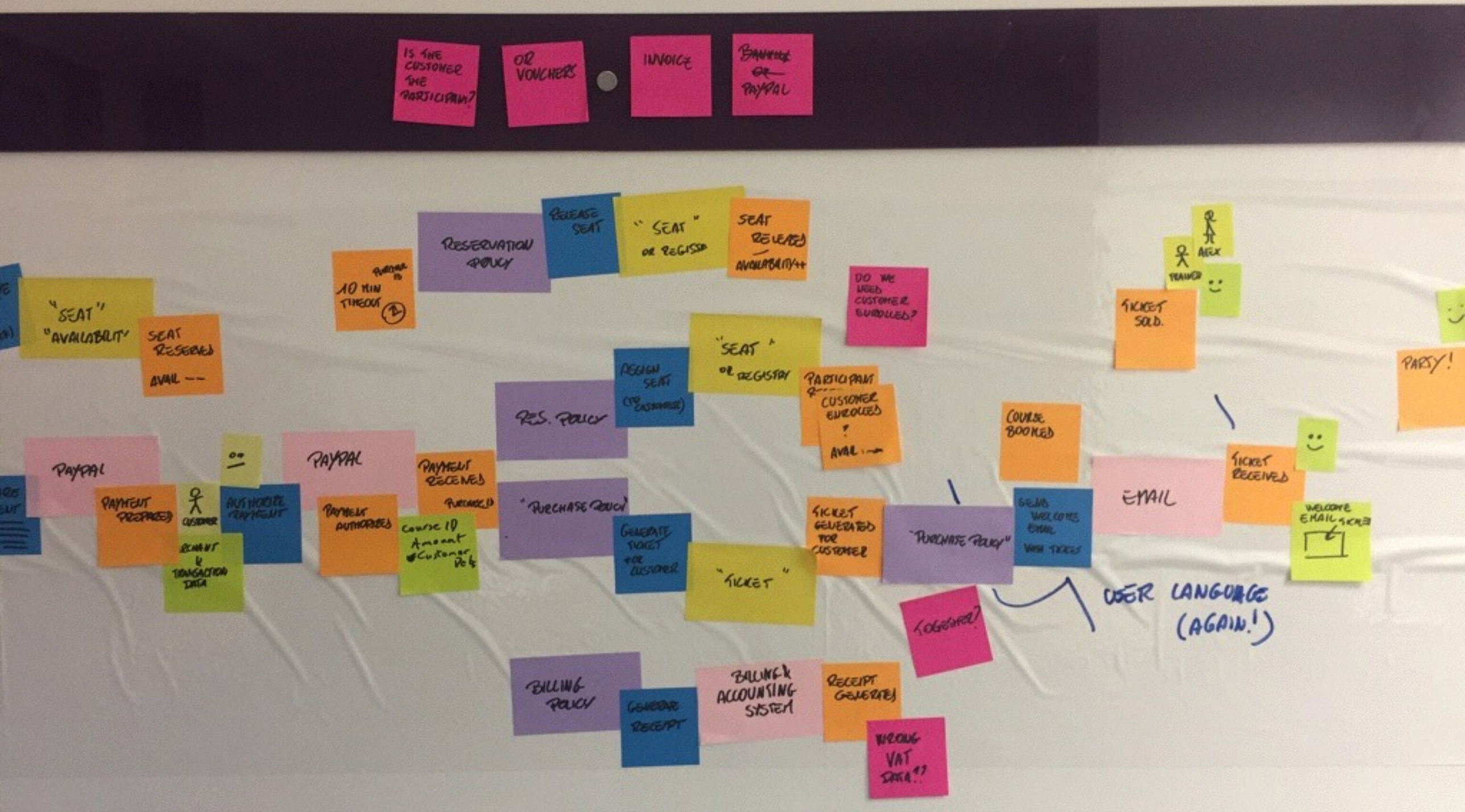
Software development is a learning process

BUT WE'RE ALWAYS DOING "SECOND HAND" LEARNING

Learning is too important to leave it to Specs.

2) Keep the implementation as close as possible to our understanding

#ProTip this is what Ubiquitous Language is all about
#ProTip ...and EventStorming too!!


IS THE CUSTOMER THE PARTICIPANT?
OR VOUCHER
INVOICE
BANKING OR PAYPAL
"SEAT" "AVAILABILITY"
SEAT RESERVED
AVAIL --
10 MIN TIMEOUT
RESERVATION POLICY
RELEASE SEAT
"SEAT" OR REGISTER
SEAT RELEASED
AVAILABILITY++
DO WE NEED CUSTOMER ENROLLED?
ASSIGN SEAT (TO CUSTOMER)
"SEAT" OR REGISTRY
PARTICIPANT
CUSTOMER ENROLLED
AVAIL --
RES. POLICY
PAYPAL
PAYMENT PREPARED
CUSTOMER
REMAINS & TRANSACTION DATA
AUTHORIZE PAYMENT
PAYPAL
PAYMENT AUTHORIZED
PAYMENT RECEIVED
PURCHASE ID
Course ID
Amount
Customer Data
"PURCHASE POLICY"
GENERATE TICKET FOR CUSTOMER
"TICKET"
TICKET GENERATED FOR CUSTOMER
"PURCHASE POLICY"
COURSE BOOKED
SEND WELCOME EMAIL WITH TICKETS
EMAIL
TICKET RECEIVED
WELCOME EMAIL
TICKET SOLD.
ALEX
TRAINER
PARTY!
LOGGING?
BILLING POLICY
GENERATE RECEIPT
BILLING & ACCOUNTING SYSTEM
RECEIPT GENERATED
WRONG VAT DATA!?
USER LANGUAGE (AGAIN!)

The Cracks in the Monolith

Making sense of a huge mess

The DRY principle

"Every piece of knowledge must have a single, unambiguous, authoritative representation, within a system"

Andy Hunt & Dave Thomas

The DRY principle

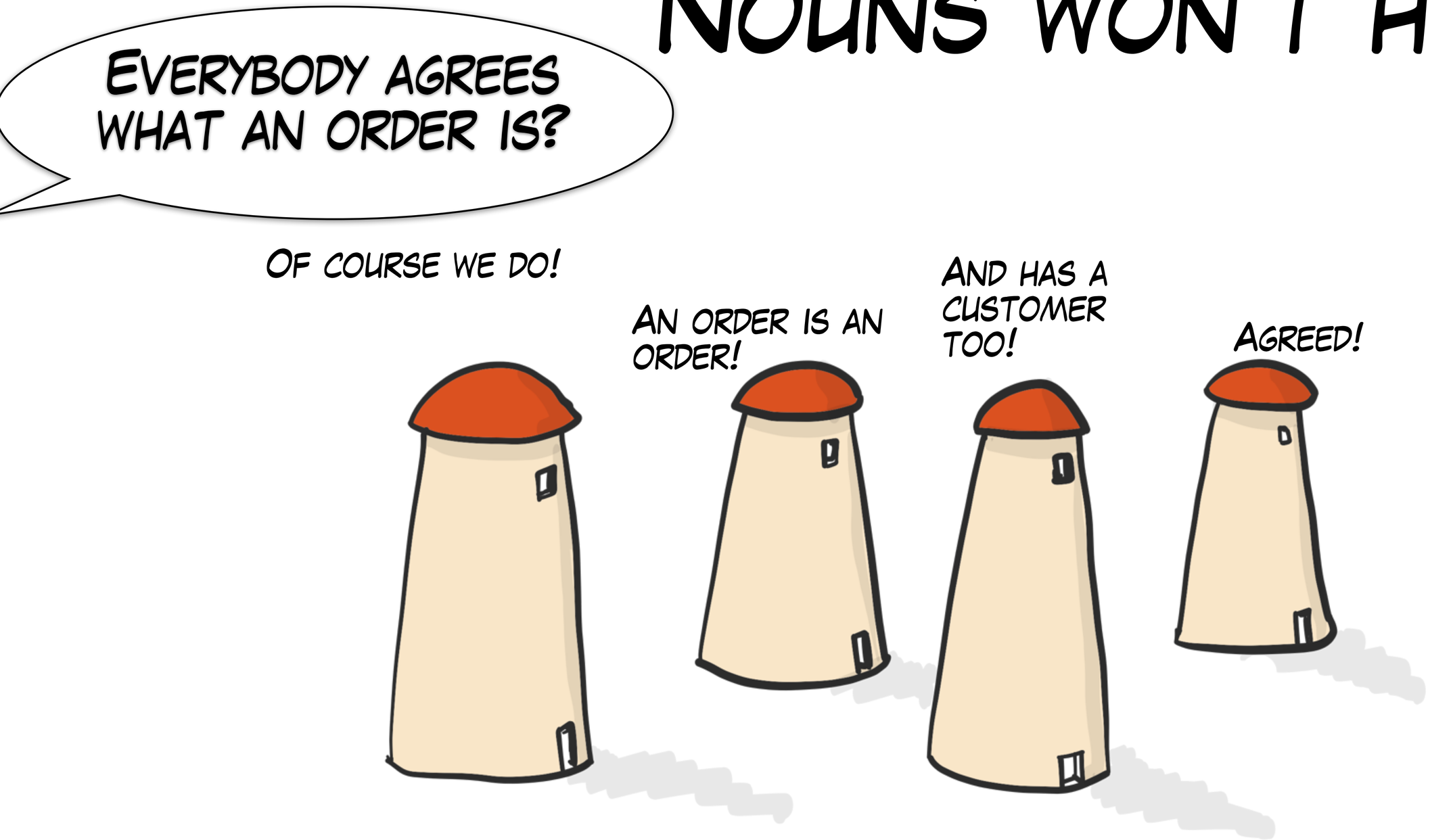
*"Every piece of knowledge must have a single, **unambiguous**, authoritative representation, within a system"*

Andy Hunt & Dave Thomas

Modelling Data-First

- Focus on nouns
- Expand the scope in order to accomodate new concepts
- ...
- ...
- Leave the company cursing the Legacy

Nouns won't help



Perfect recipe:

- Talk with many people
- Model Data-First (everybody agrees on nouns)
- Add some "Dogmatic DRY principle"
- Repeat

The Big Ball of Mud

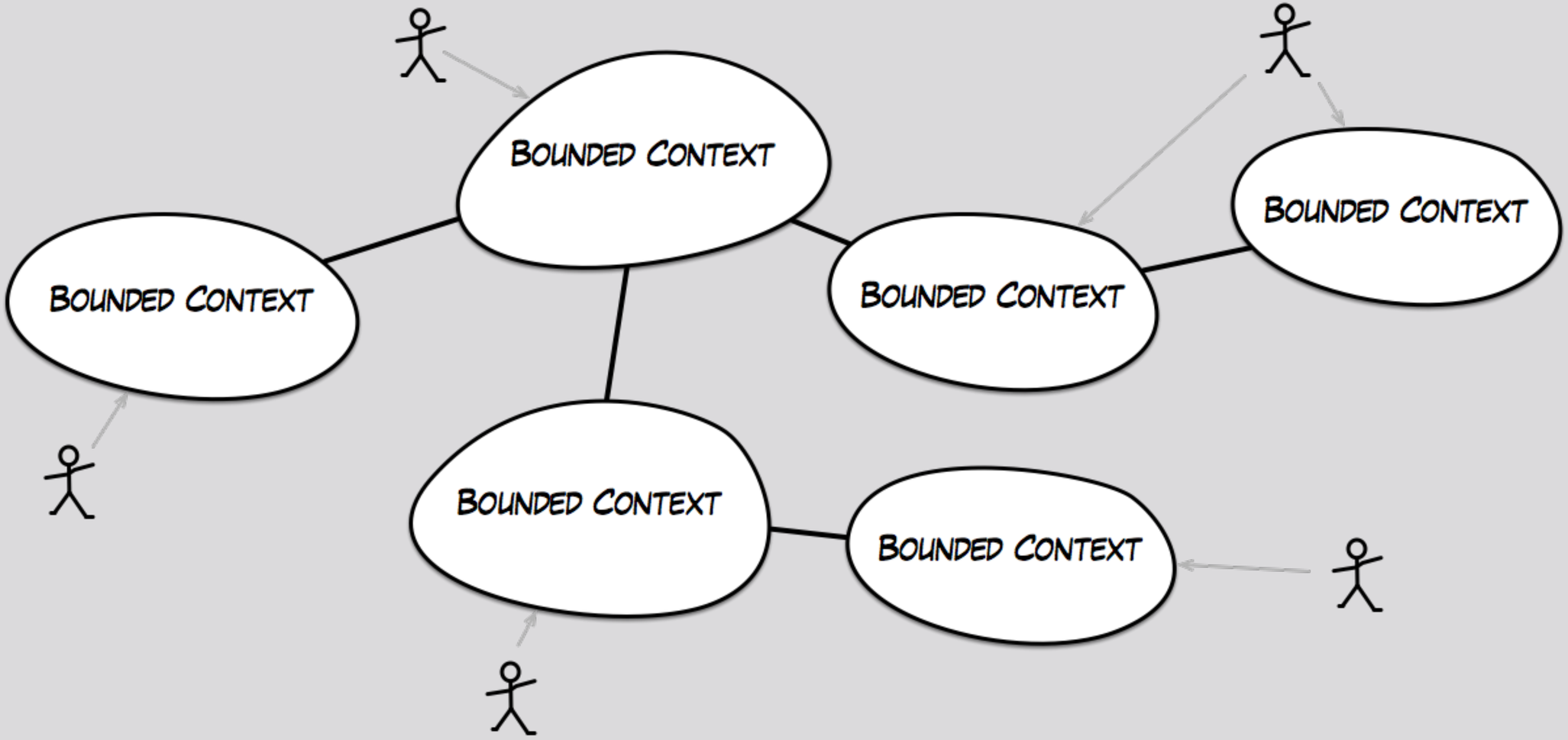
A SINGLE MODEL WON'T SOLVE AN ENTERPRISE PROBLEM, WE'LL NEED MORE!

Enterprise software means "many people, with many different problems"

Many different problems

Many different Models

Conceptual boundaries



Safety

You just won't learn under pressure

Thin red line

- Start a project with a data first approach
- Add misunderstood DRY principle
- Lack of control requires protection & specialisation
- few people become key, others retire or get minionized Business grows on top of old code
- SAFETY is gone. Without it, no experiments are possible.
- More responsibilities: less time to change it, harder to make the call
- Hard to recruit seniors to finish the job
- Move to Microservice in order to refresh the technology landscape

Thin red line

- Postpone evolutions of critical software
- prevent the organisation from getting new feedback
  - -> how long are your iterations?
- Suddenly realise that the organisation is dumbed down and blames IT for everything

They're the same problem

Microservices

Is Envy a driver for architectural choices?

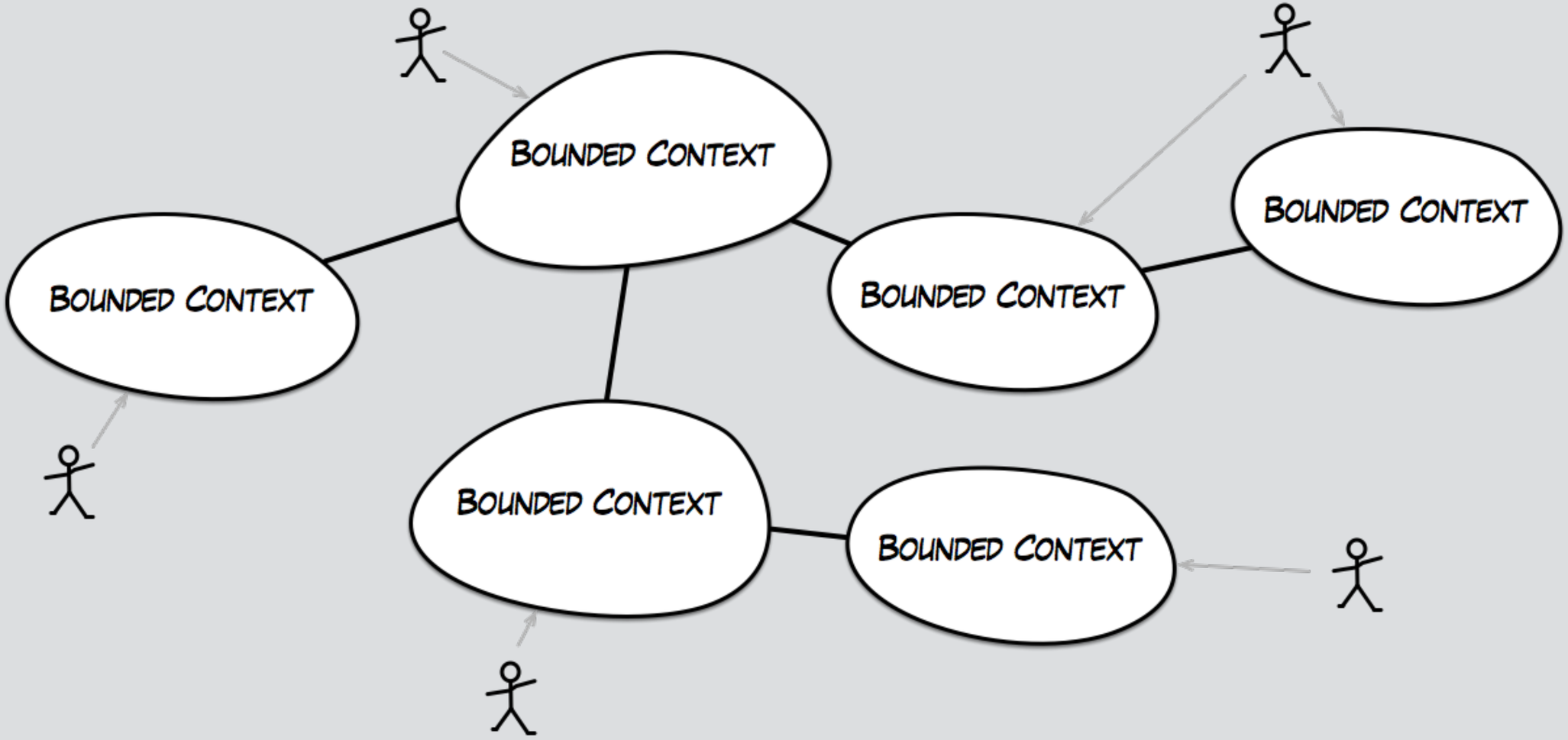
I SPEND MOST OF MY TIME RESCUING TEAMS THAT WENT MICRO SERVICES OVER A MONOLITH...

... ONLY TO DISCOVER THAT THEY'RE TANGLED WITH DEPENDENCIES!!

Data-Driven Thinking Is the problem

How do I untangle this mess?

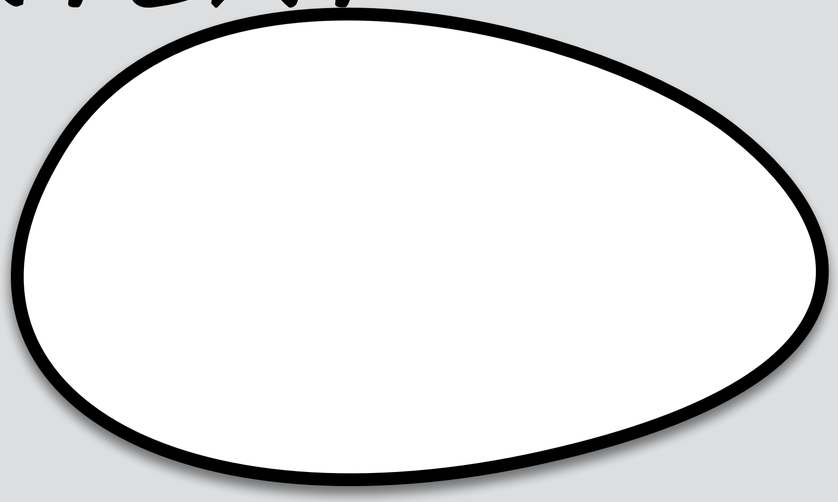
Conceptual boundaries?



Bounded Contexts to the rescue

- Specialized models for a well defined purpose

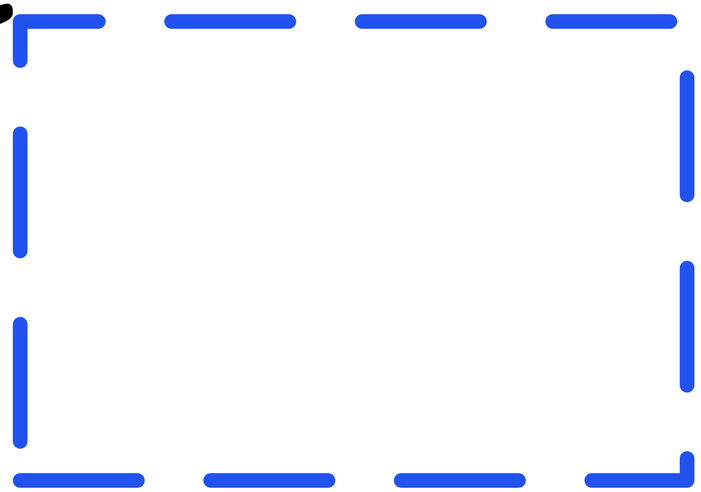
Bounded Context



- Is a <u>language</u> boundary.
- Defines a consistent <u>model</u> around a specific <u>purpose</u>

#protip: "managing" is not a purpose

MicroService

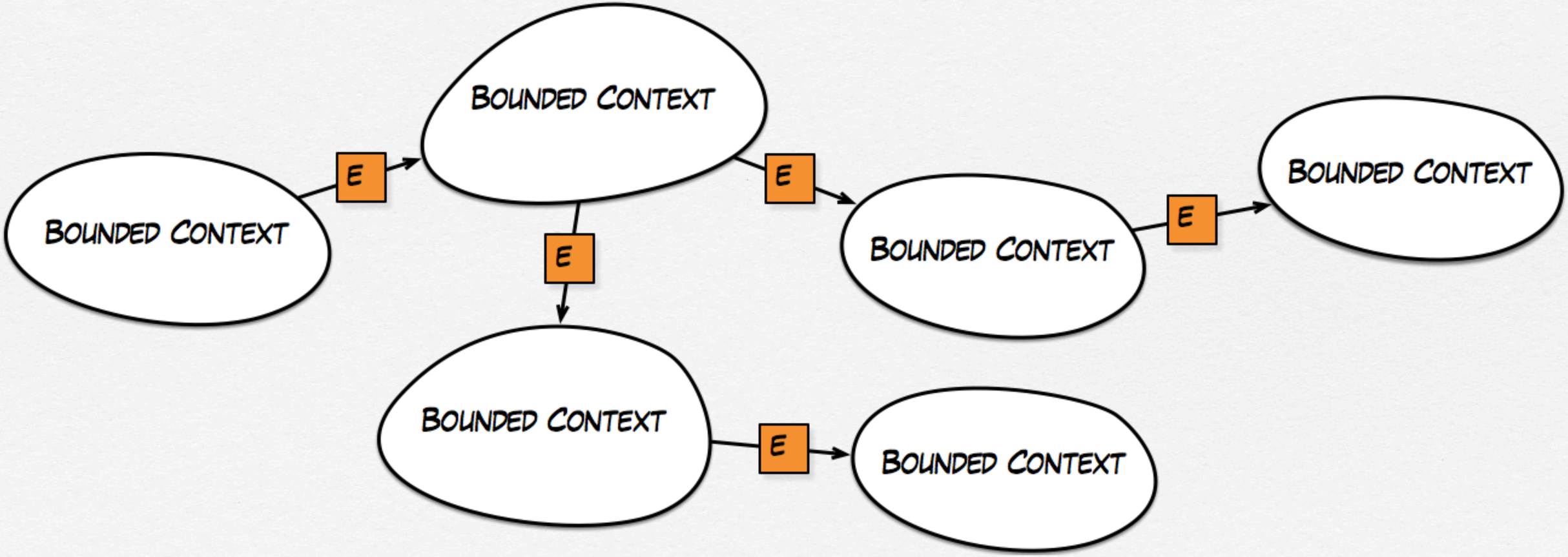


- A deployment boundary:
- Interestingly, a unit of ownership

NOT THE SAME THING

But One thing is clear...

This works just better!



Domain Events Provide the best abstraction for modelling enterprise software

WHY DDD MATTERS NOW

@ZIOBRANDO

High-Value enterprise software needs a special approach

Focus On Behaviour

Instead of data

Focus On Events
Instead of Nouns

Focus On Learning

Instead of just delivering

Life is too short to write meaningless software

Questions?

Thank you!