That's my monkey

A Functional, Reactive, Domain Driven Design, and Common Sense Approach To Architecture:

This is my circus, and those are my monkeys!

Rob Martin / Version2beta

LambdaDays 2017, Kraków, Poland

What is a Senior?

"Everyone can be taught to sculpt. Michelangelo would have had to be taught how not to."

- Alan Perlis

"Your first 10,000 photographs are your worst."

- Henri Cartier-Bresson

What is a 10x developer?

KSLOC?

Function points?

Cyclomatic complexity?

Constructive cost modeling?

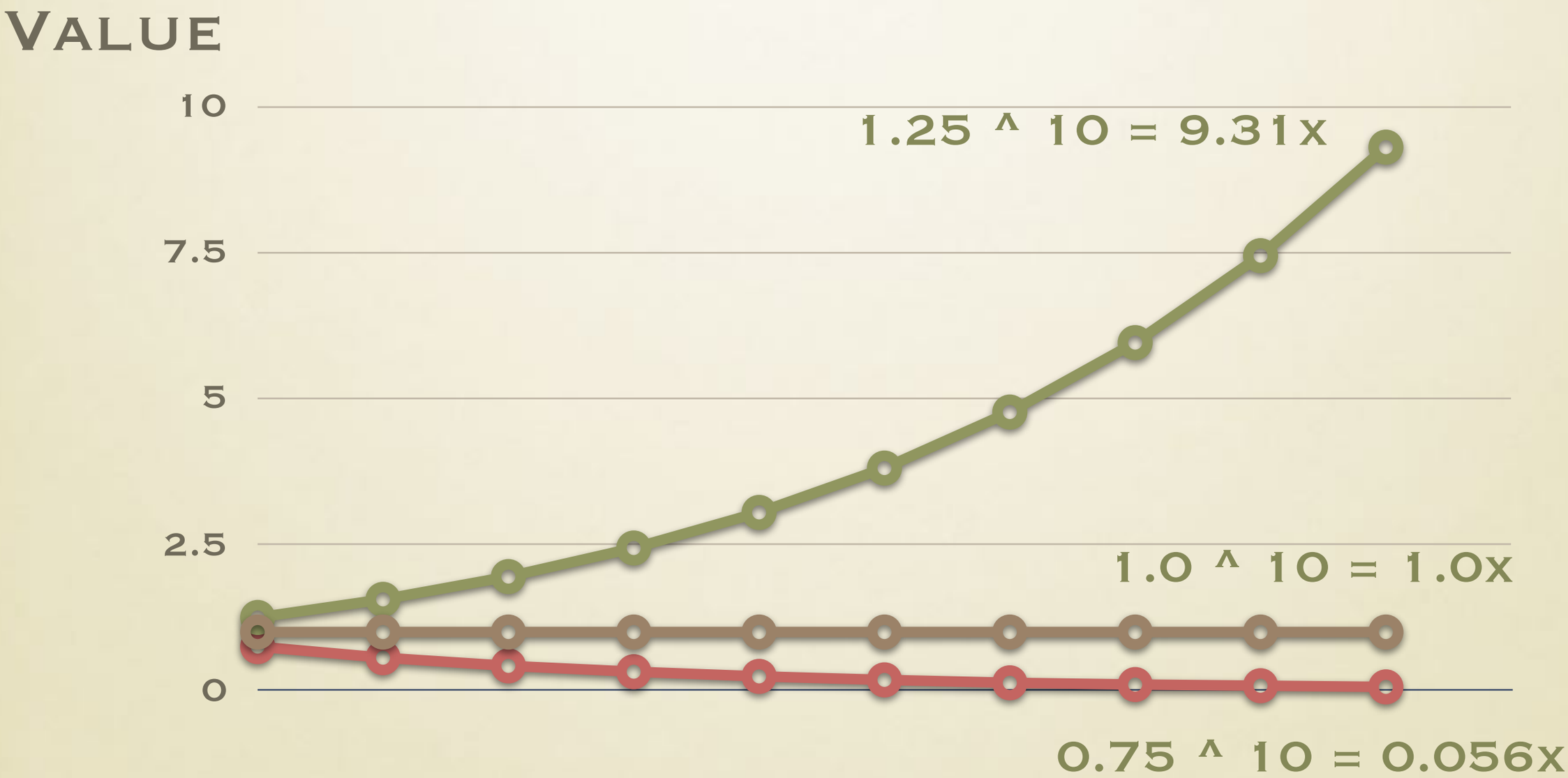
Value.

What is value?

Build tools for others to build richer lives

Quick and frequent delivery with prompt feedback

The 10x developer



Creating 10x developers

Workshops

Internships

Master - Apprentice

Mob programming

Code reviews

Deliver Value Quickly (or, Can we survive our success?)

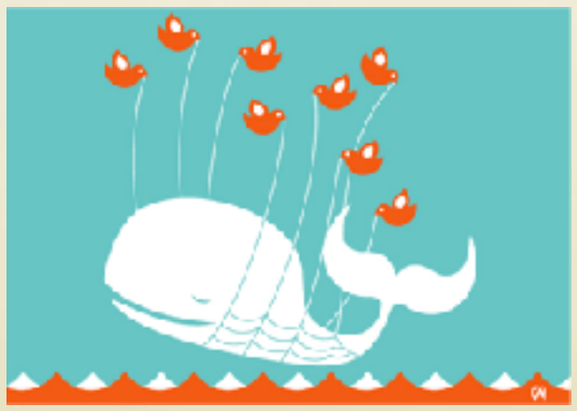
Rounded up, all startups fail

Expediencies: scripting languages, monolithic frameworks, popular libraries

Expediency *means* Shortcuts *that cause* Problems

Difficult to maintain, difficult to scale

Our users lose confidence. We lose confidence.



What is anti-value?

Monolithic

Complex

Rigid

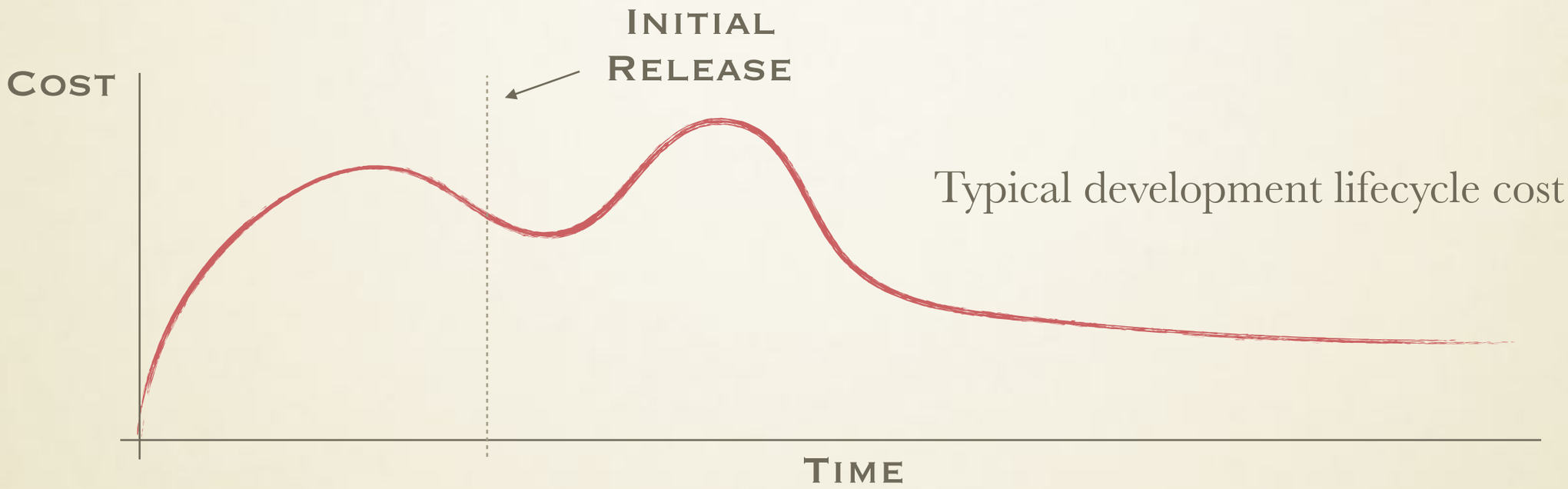
Tightly coupled

Fragile

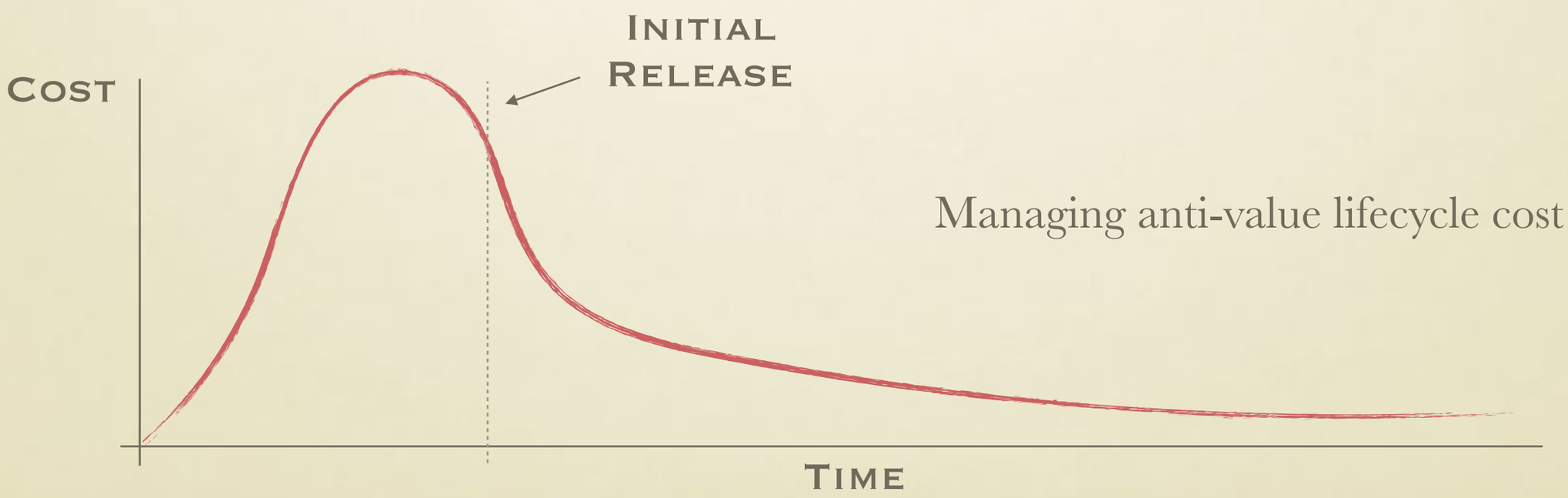
Untested

Laden with technical debt

Value and Anti-value



Typical development lifecycle cost



Managing anti-value lifecycle cost

Simple, demonstrably correct code

Simple.

Demonstrably correct.

What is complexity?

… Any code that can be decomposed into smaller components.

… Interrelationships and dependencies.

… The enemy of reliability.

… The part of our code that isn't beautiful.

… Code.

Why complexity is bad

We can't reason about our code.

We can't demonstrate our code is correct.

We can't trust our code.

What is simplicity?

We can reason about our code.

We can demonstrate our code is correct.

We can trust our code.

The simplicity test

Can I reason about this code?

Can I demonstrate that this code is correct?

Can I trust this code?

Code reviews

Do the tests cover the feature/issue fully?

Is the code under test easy to reason about?

Do I trust this code?

Evaluating tools

Can I reason about this code?

Can I demonstrate that this code is correct?

Can I trust this code?

PURE FUNCTIONS

Referential transparency

Immutability

Function composition

Laziness

Distribution

Functions *are* primitives

Functional-first programming

Simple mode

- *First*, code everything you can without side effects.
- *Then*, code your side effects.

Advanced mode

- *First*, code your side effects, and only your side effects.
- *Then*, code your business logic using pure functions.

Functional-first refactoring

1. Reproduce the issue manually.
2. Find the likely problem with the code.
3. Extract the business logic into a function, while leaving the side effects in the method.
4. Pass any arguments the function needs (purify).
5. Write a failing regression test around the function.
6. Write passing characterization tests around the function.
7. Fix the code to pass the regression test.
8. Refactor to reduce complexity and improve maintainability.

There are two types of programming languages*

Some languages feel like they were built in a garage

Some languages feel like they are based on the laws of the universe

* sarcasm provided at no extra charge

Functional programming languages

Referential transparency

Immutability

Function composition

Laziness

Distribution

Functions *as* primitives

Identity

In OOP, Identity is the current state of the object.

In FP, Identity is a collection of states over time.

Databases

Relational databases

(Most) NoSQL databases

Immutable databases

What is a database anyway?

Reactive systems

Responsive

Resilient

Elastic

Event driven

Domain Driven Design

Domain

Model

Ubiquitous language

Entity

Event

Bounded context

The front end

Purpose

- Maximizing value for the user
- Creating the best experience
- Accomplishing the user's goal

Business logic

- Purely functional
- Easy to test

Interface

- Event-driven
- Reactive

The back end

Purpose

- Protecting the integrity of the data by applying business logic

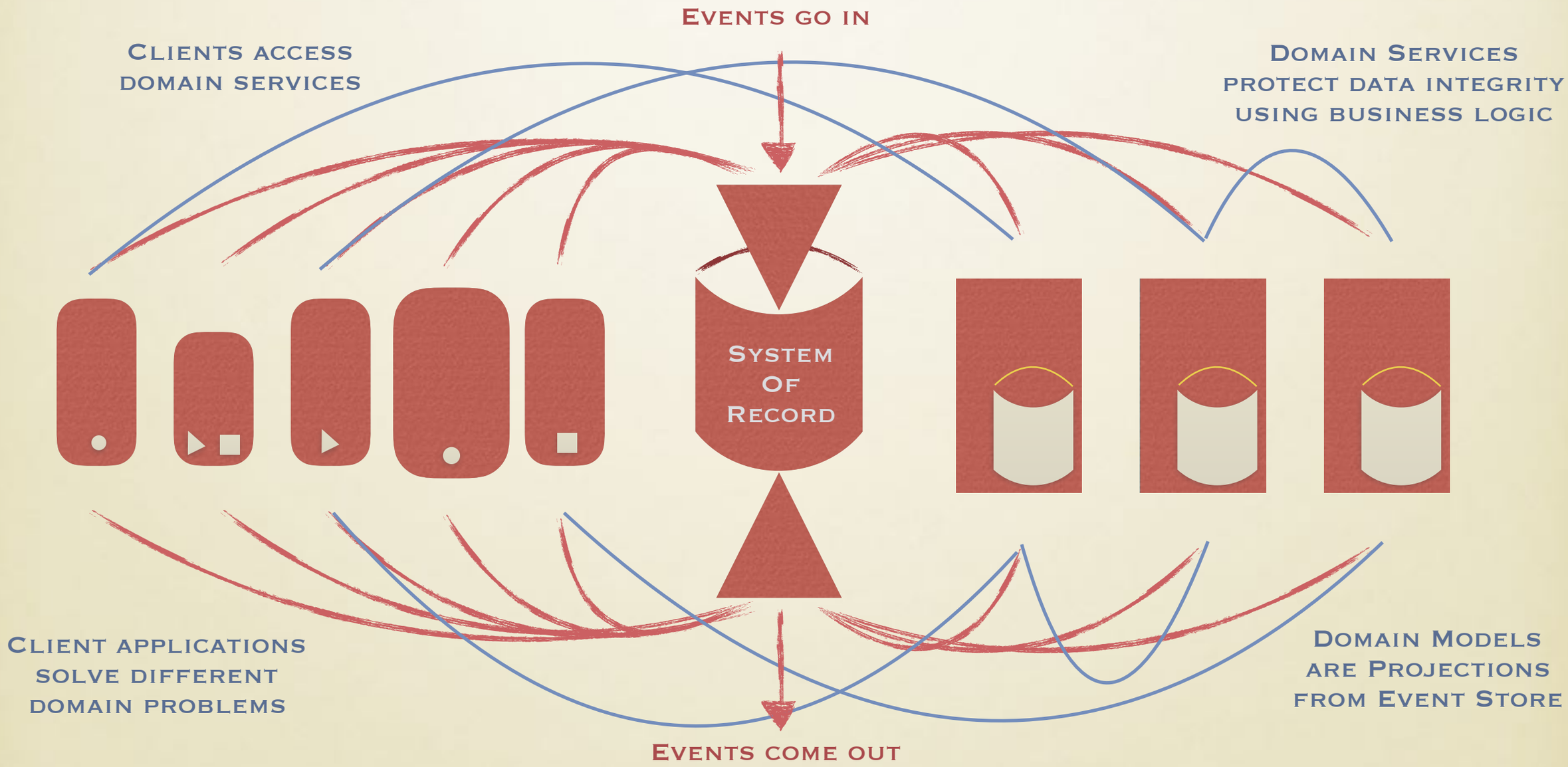
Business logic

- Purely functional
- Horizontal scaling
- Easy to test

Side effects

- Isolated
- Integration testing only
- Hexagonal

Solve for an architecture



Domain events

Message bus or POST { [event data] } TO
http://[eventstore]/[realm]/[domain]/[entity ID]/[event type]

Persist to immutable CASSANDRA event store

Message bus or web sockets or GET { [event data] } FROM
http://[eventstore]/[realm]/[domain]/[*opt. entity ID*]/[*opt. event type*]

Domain services

Map events to functions

Apply the business logic

Maintain the domain model

Answer queries

Domain model

Projections

v.

Reductions

Distributed computing

Not invented here.

Simple, demonstrably correct systems (reactive version)

Focus on response

Reduce blast radius

Scale elastically

Pass messages asynchronously

Simple, demonstrably correct systems (Domain Driven Design version)

Architecture and business processes are isomorphic

We can talk to our business partners

Our business partners can comprehend our code

Domain events and services map to both ES-CQRS and Reactive

Does Eric Evans have an FP secret?

Simple, demonstrably correct systems (functional programming version)

The immutable event store

Events map to pure functions

Domain services are like a function applied to all our data

Eagerly, lazily

Isolated side effects

Pure functions, composed in a pipeline

Easy to reason about our whole system

Simple, demonstrably correct systems (summary version)

Can we reason about our system?

Can we demonstrate that our system is correct?

Can we trust our system?

Rob Martin / version2beta

VP Engineering at BigSquid.com

Blogging at version2beta.com

@version2beta most everywhere online

Eventually, there will be an article version of this talk posted at

version2beta.com/articles/thats_my_monkey/