ThoughtWorks®

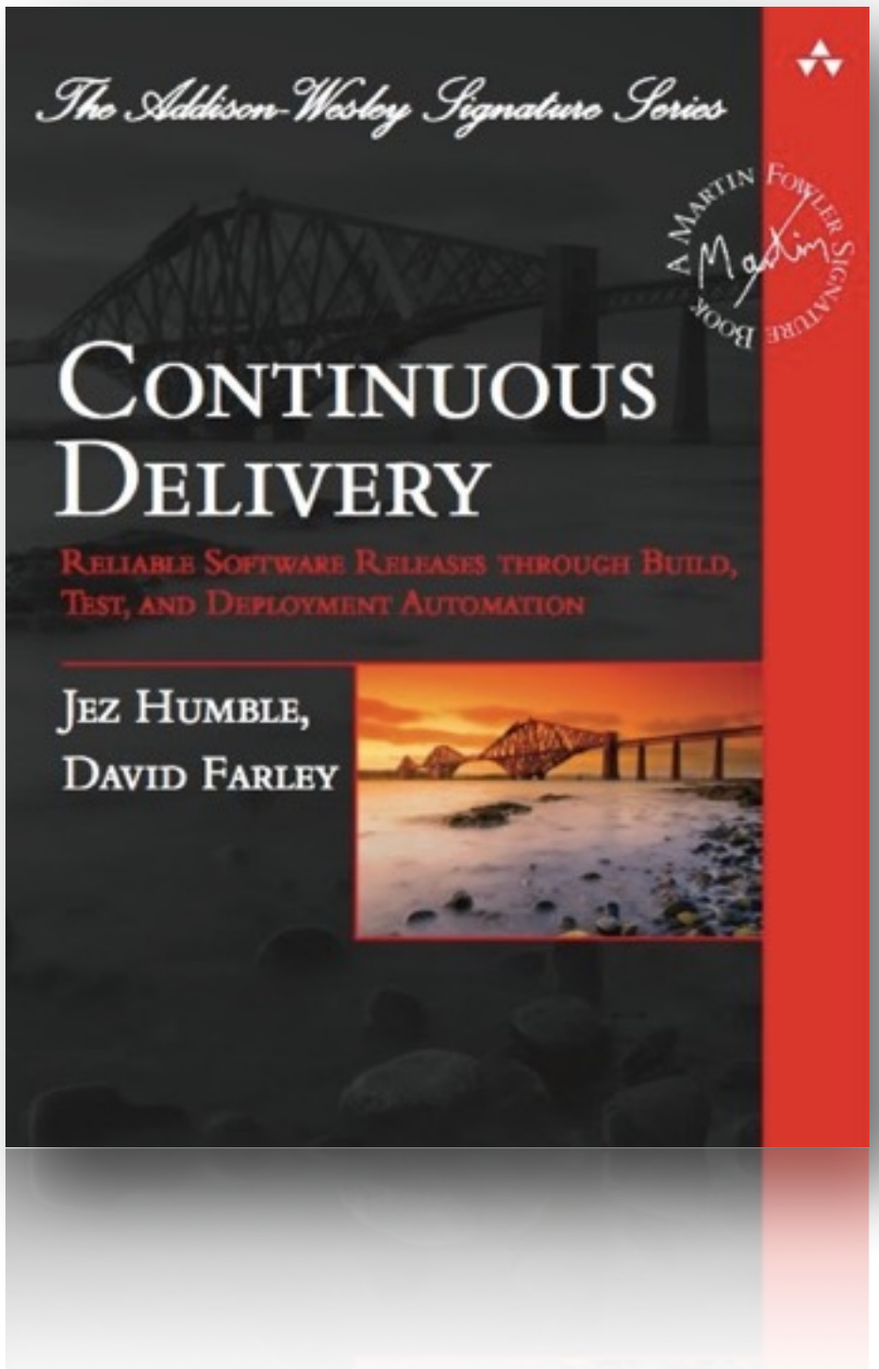
Implementing Continous Delivery

ADJUSTING YOUR ARCHITECTURE

Rachel Laycock
@rachellaycock

ThoughtWorks®

continuous delivery



Our highest priority is to satisfy the customer through early and **continuous delivery** of valuable software

*"You **can't** have*

Continuous Delivery"

“Once upon a time I was a happy developer..."

“I thought I knew what Continuous Delivery was”

“I was doing CD on my projects”

“Then one day..."

“…a client wanted Continuous Delivery”

…and we said “sure”

"...but 3 months later they were still asking..."

“Are we there yet?”

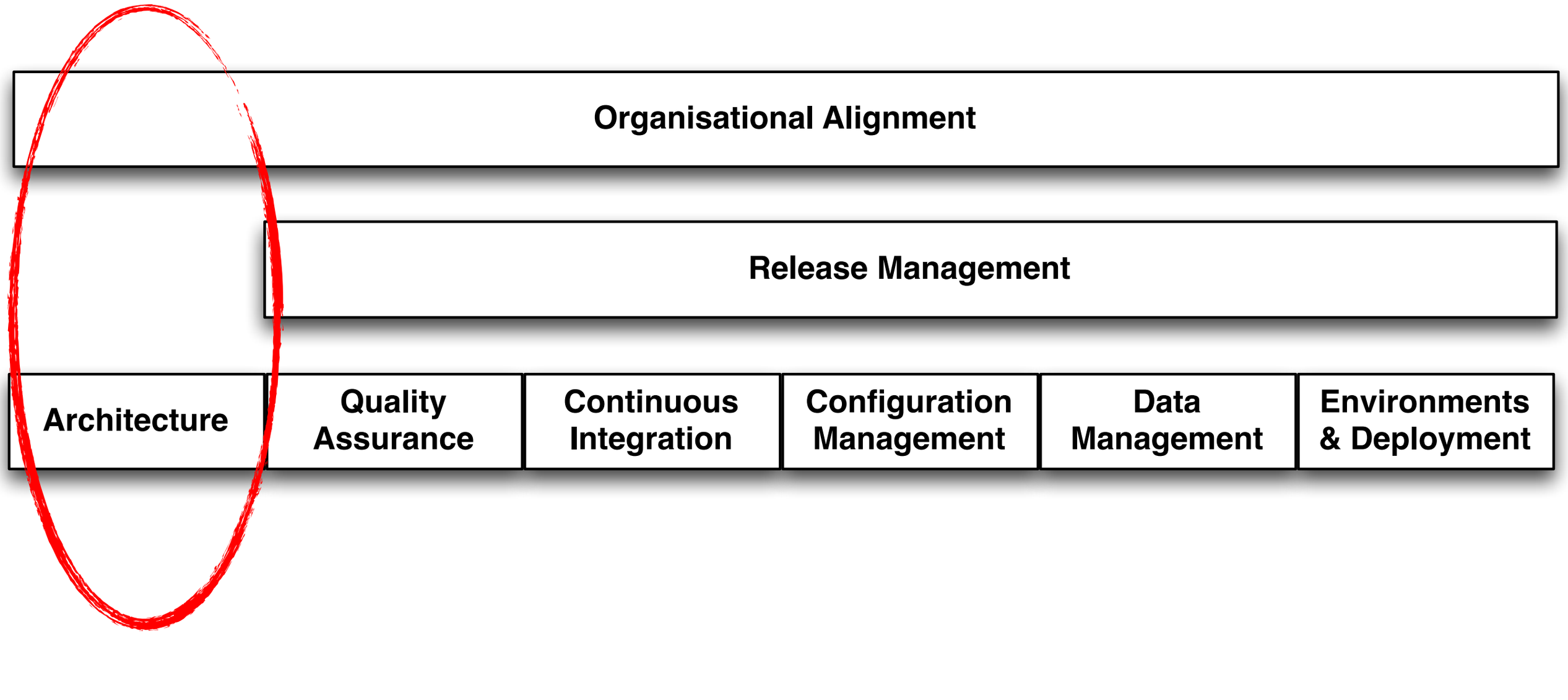
Their code base was **huge** and **complex**

1. Conway’s Law is THE LAW

2. Keep things small

3. Evolve your architecture

continuous delivery is big



stuff that is

hard to change

?

civil architecture

town planning

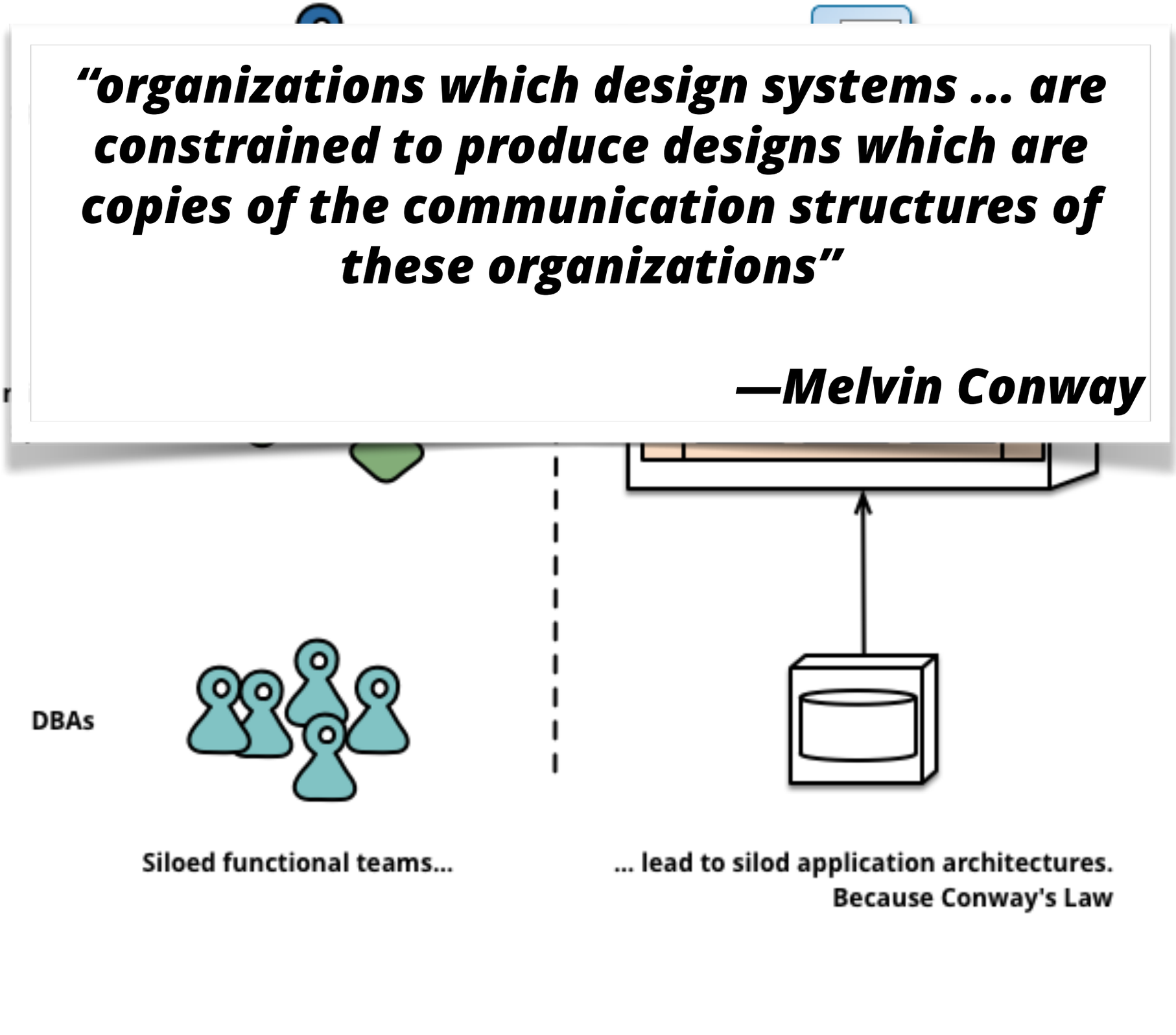
ThoughtWorks®

1. CONWAY'S LAW IS THE LAW

conway's law

"organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations"

—Melvin Conway

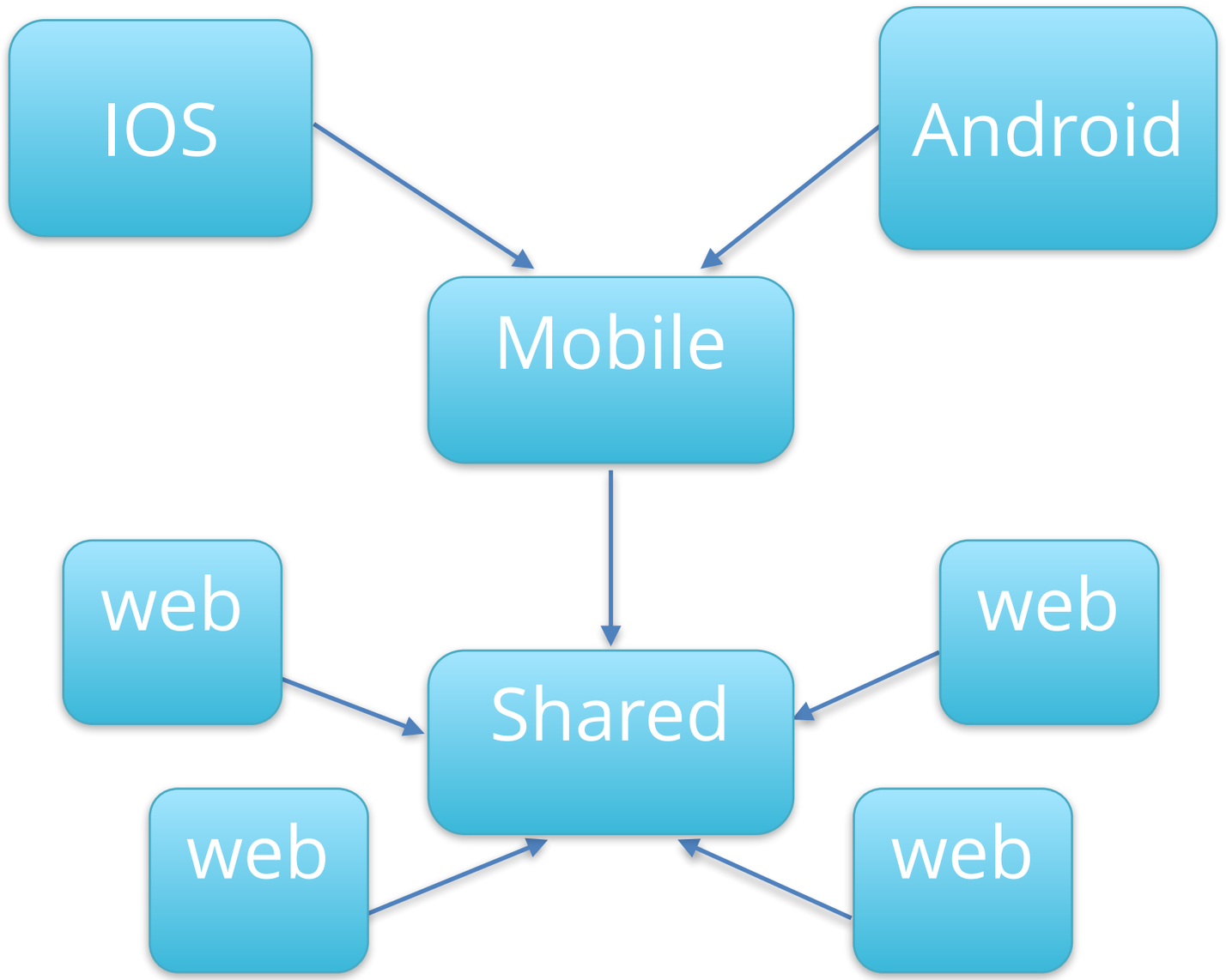


Melvin Conway: "Because ***the design that occurs first is almost never the best possible****, the prevailing system concept may need to change. Therefore, flexibility of organisation is important to effective design."*

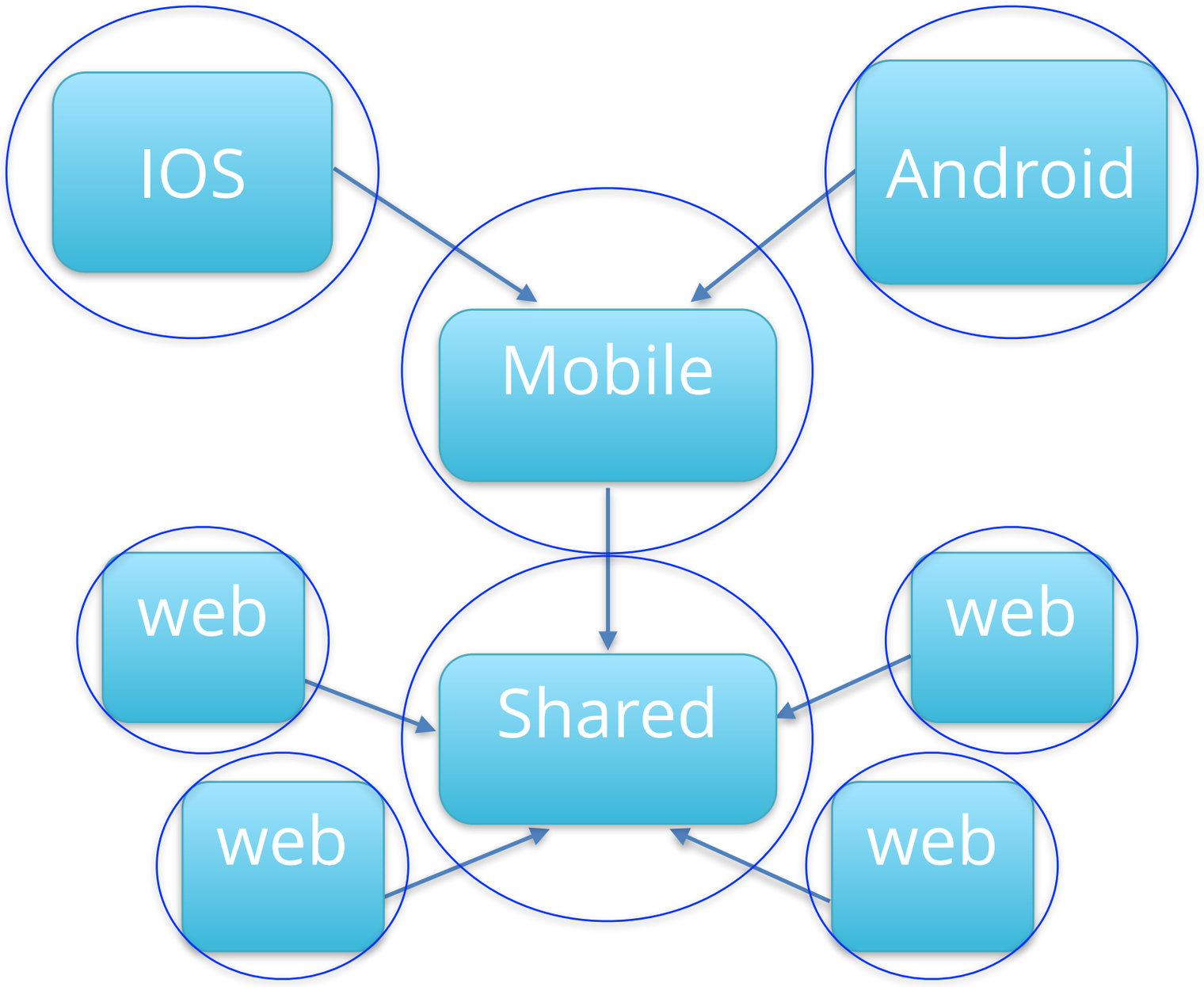
the ***wrong*** side of the law

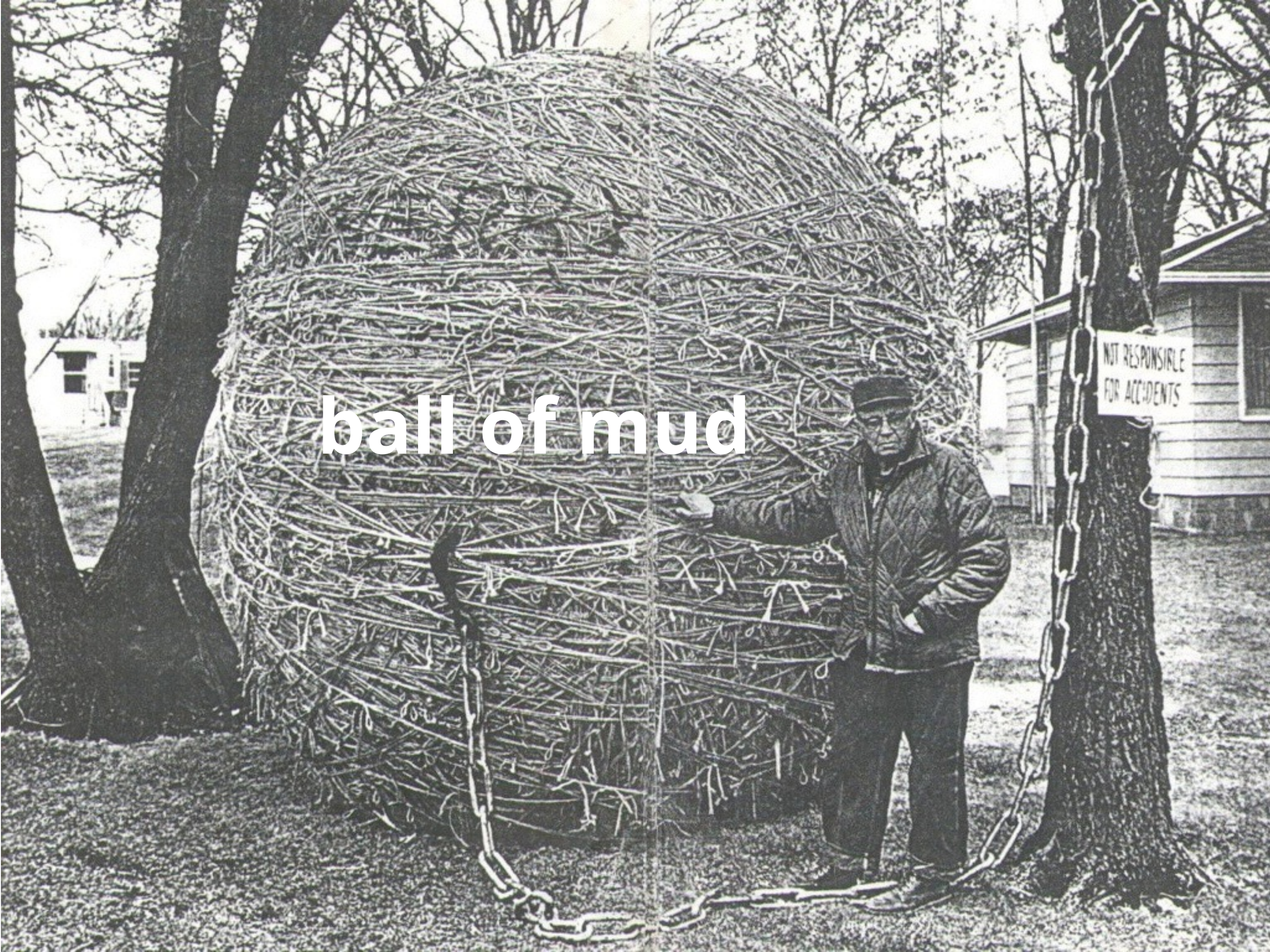
layered / tiered architecture

current state

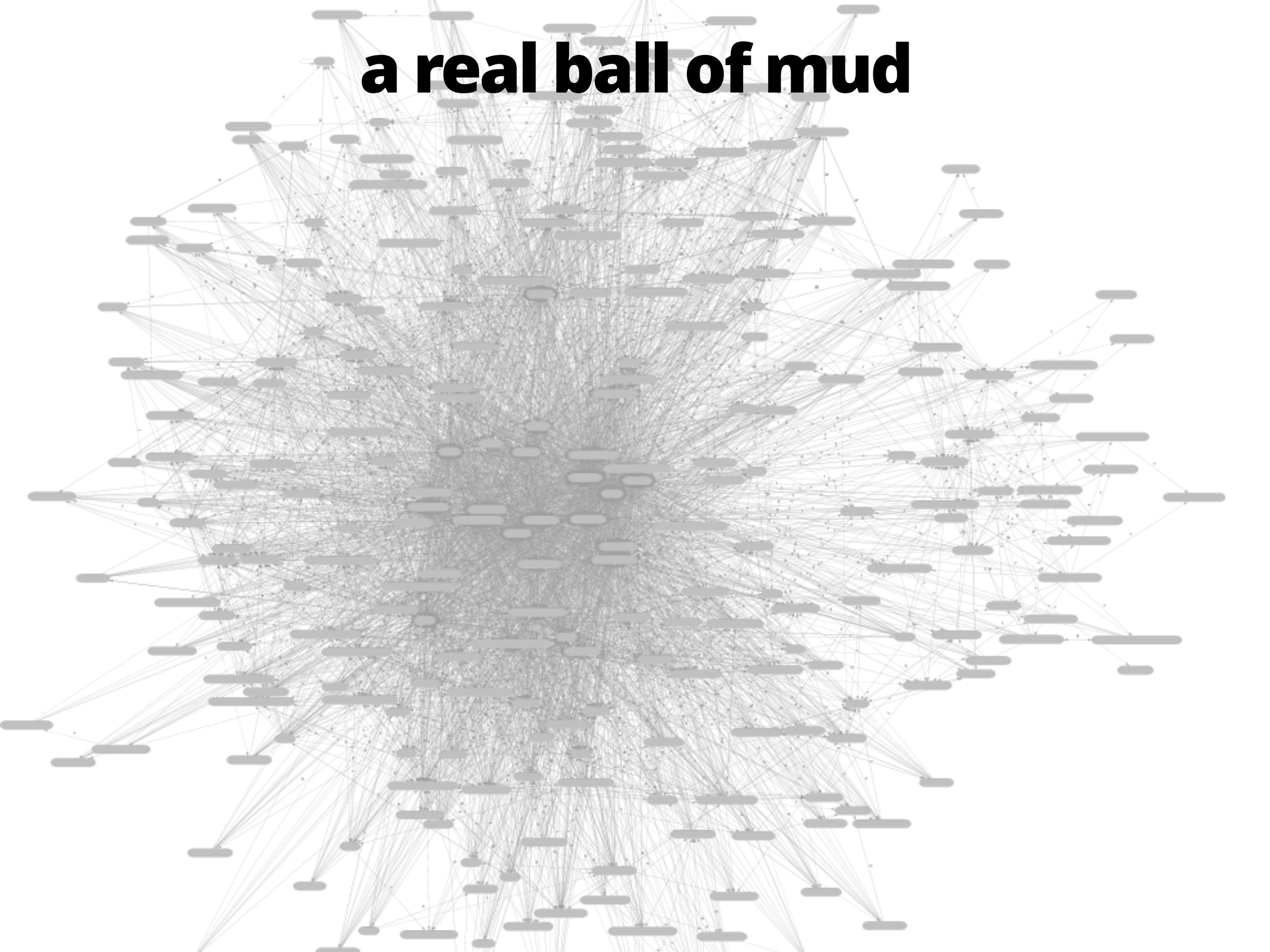


problems?




ball of mud
NOT RESPONSIBLE
FOR ACCIDENTS

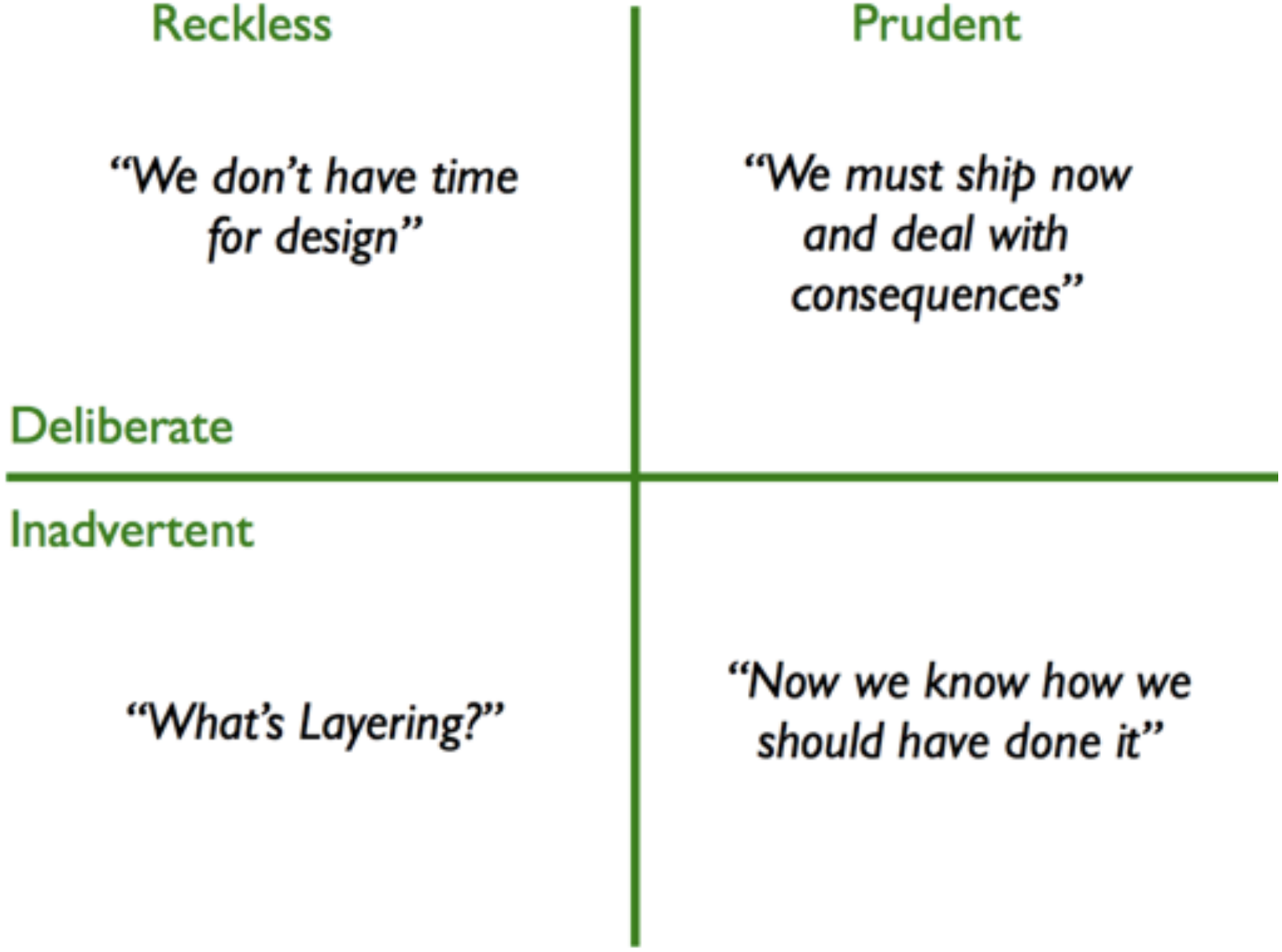
a real ball of mud



“expediency over design”

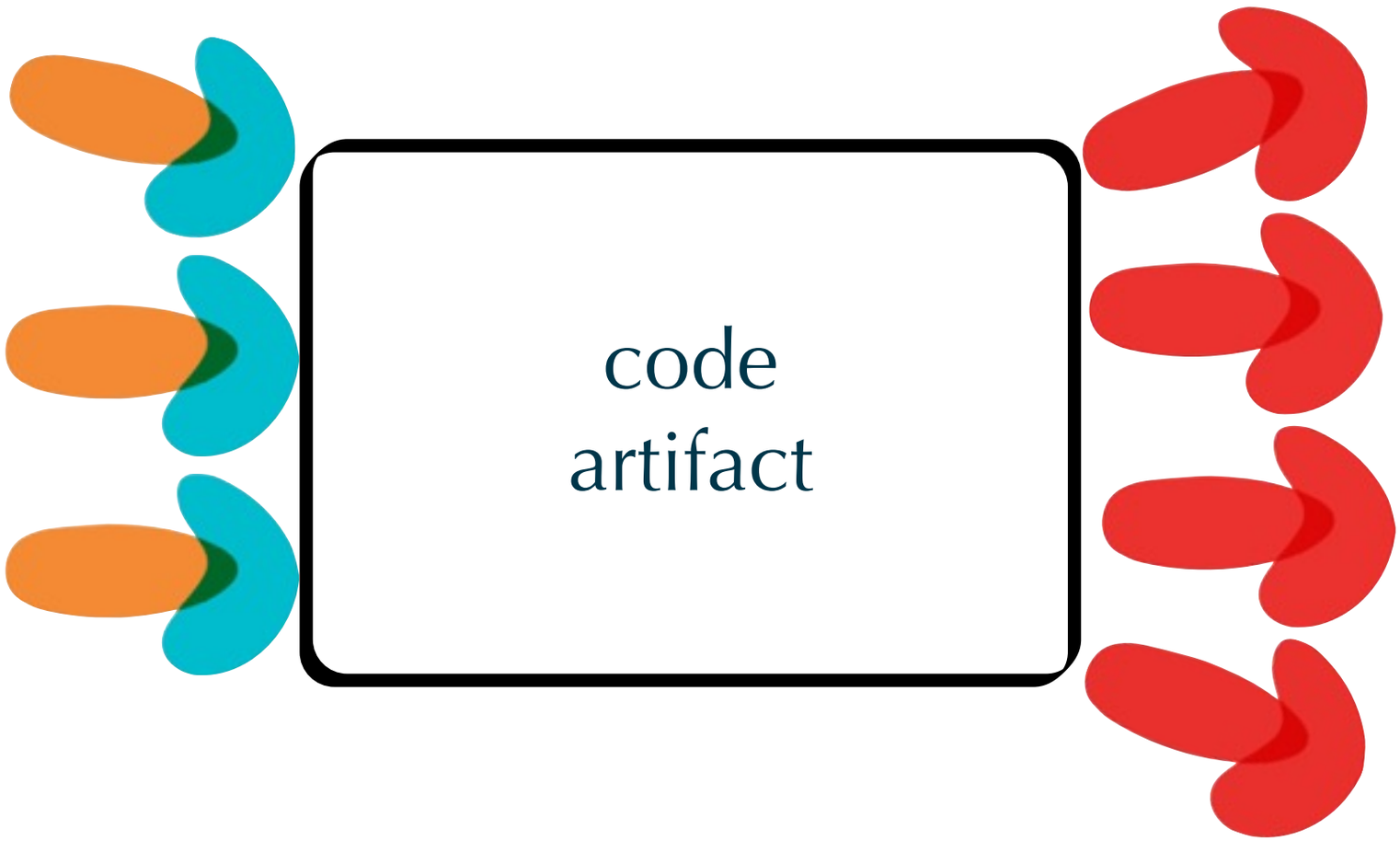
- Brian Foot & Joseph Yoder

technical debt



http://martinfowler.com/bliki/images/techDebtQuadrant.png

coupling problems

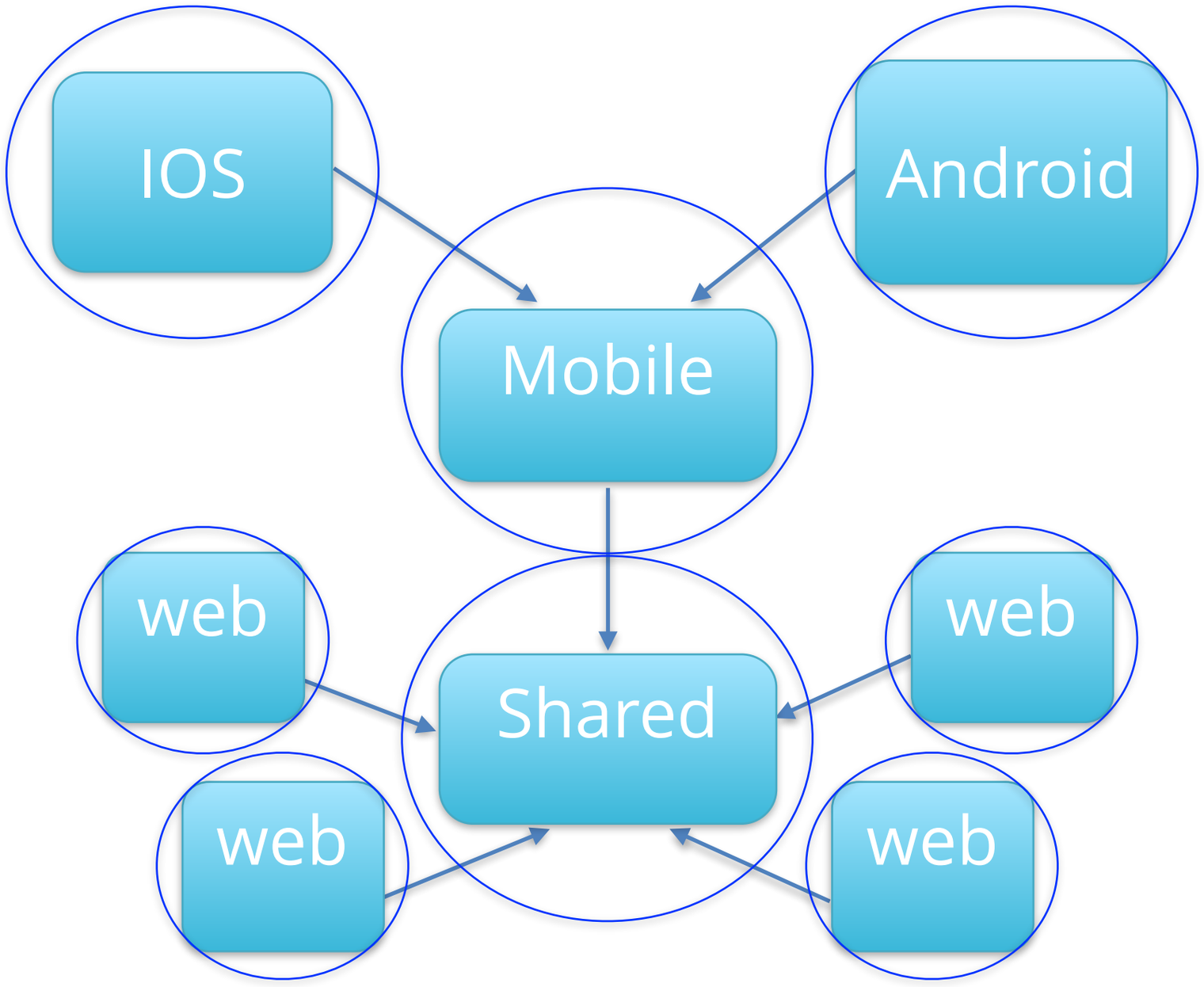


afferent

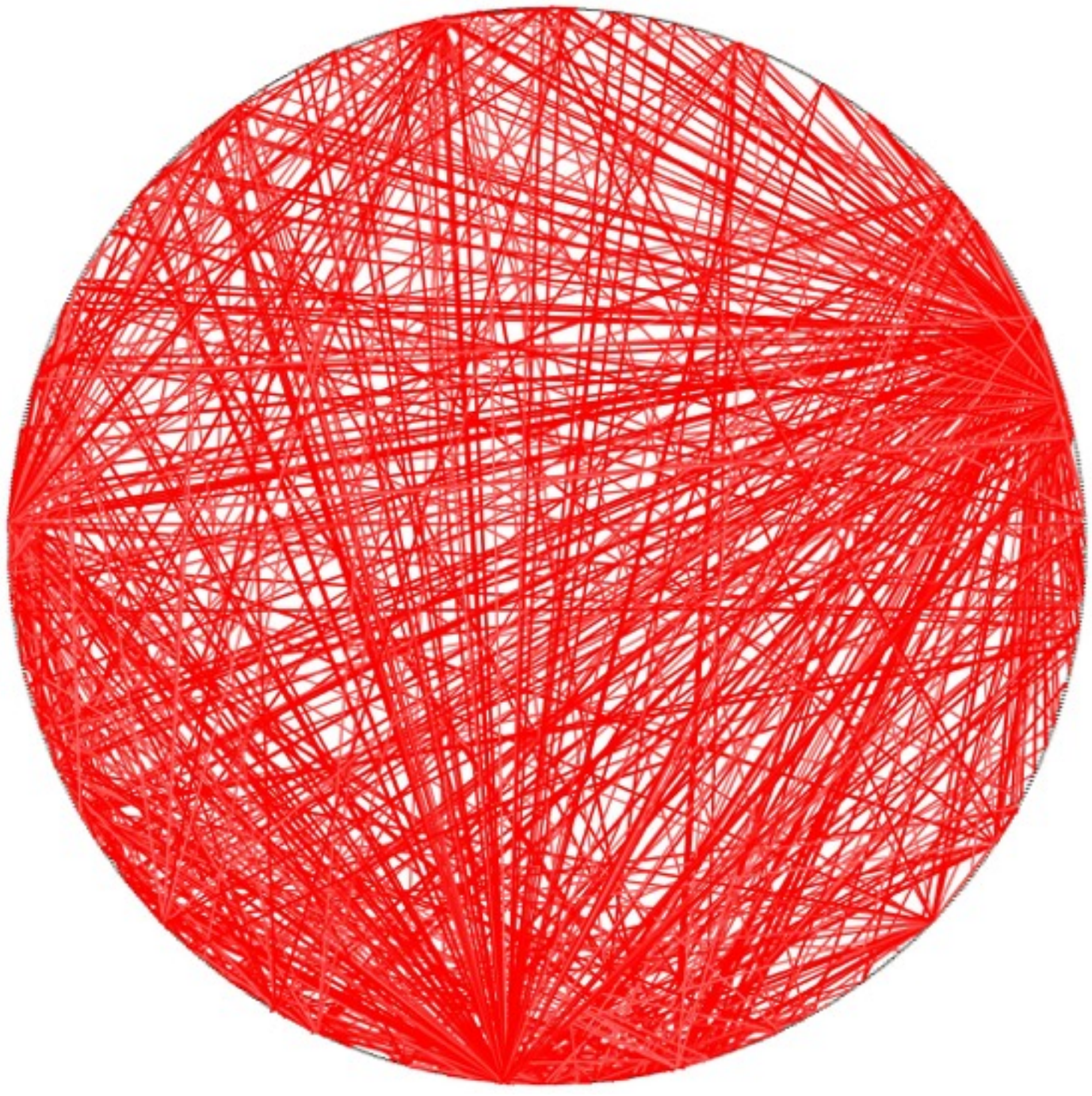
efferent

Software architecture represents the tension between coupling & cohesion.

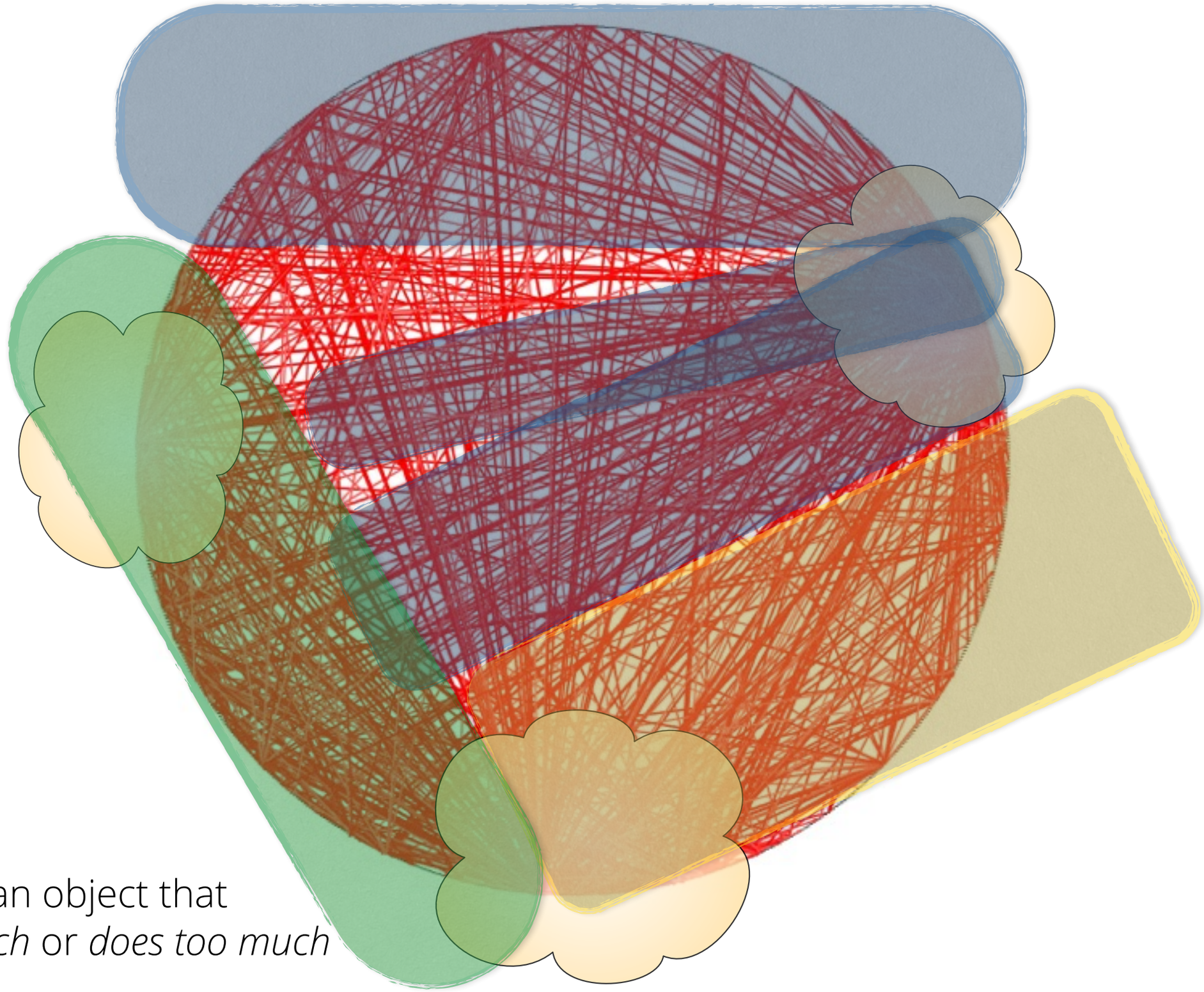
problems?



brownfield

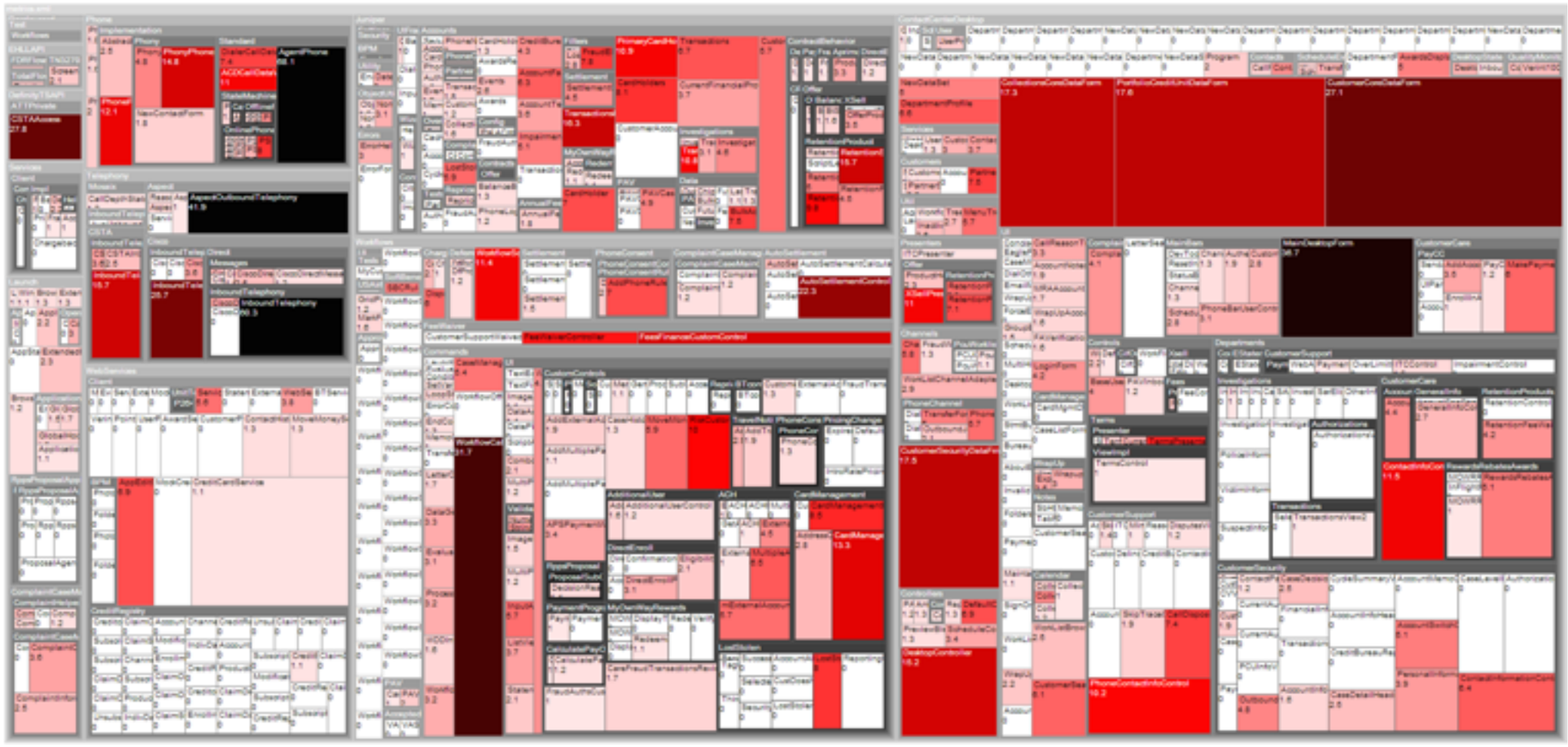


untangling

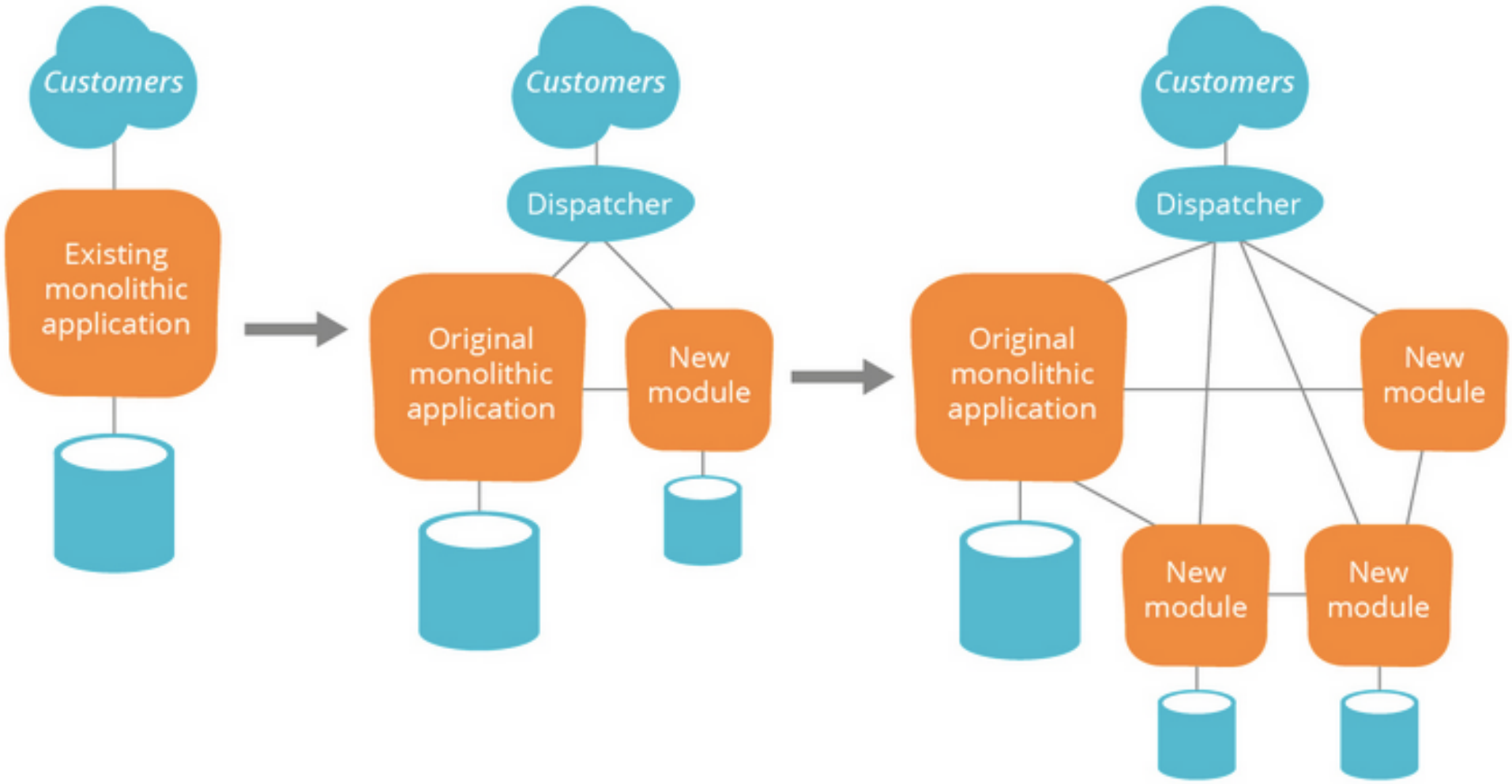


God Object: an object that
knows too much or *does too much*

metrics



strangler pattern

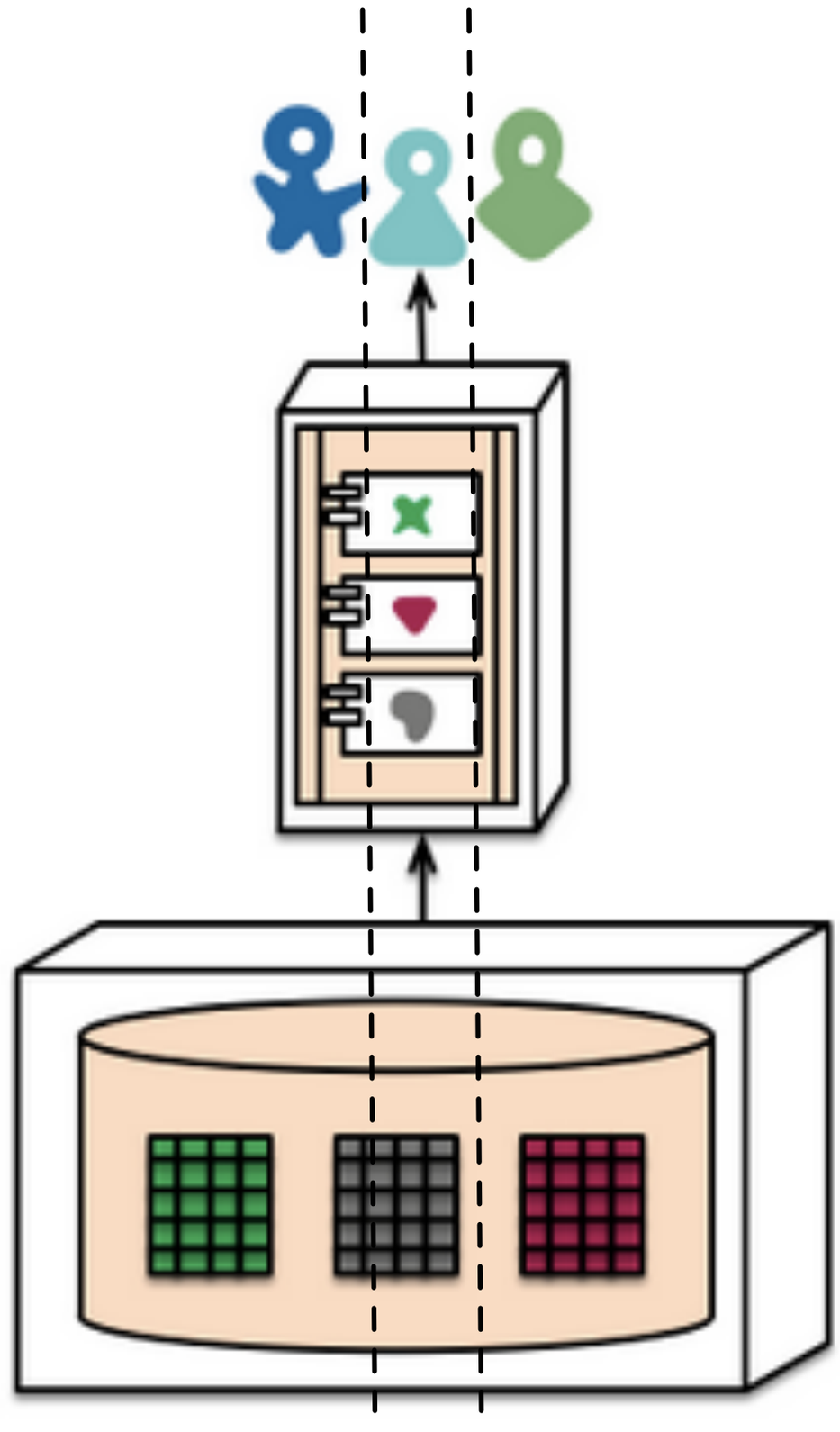


the law on your side

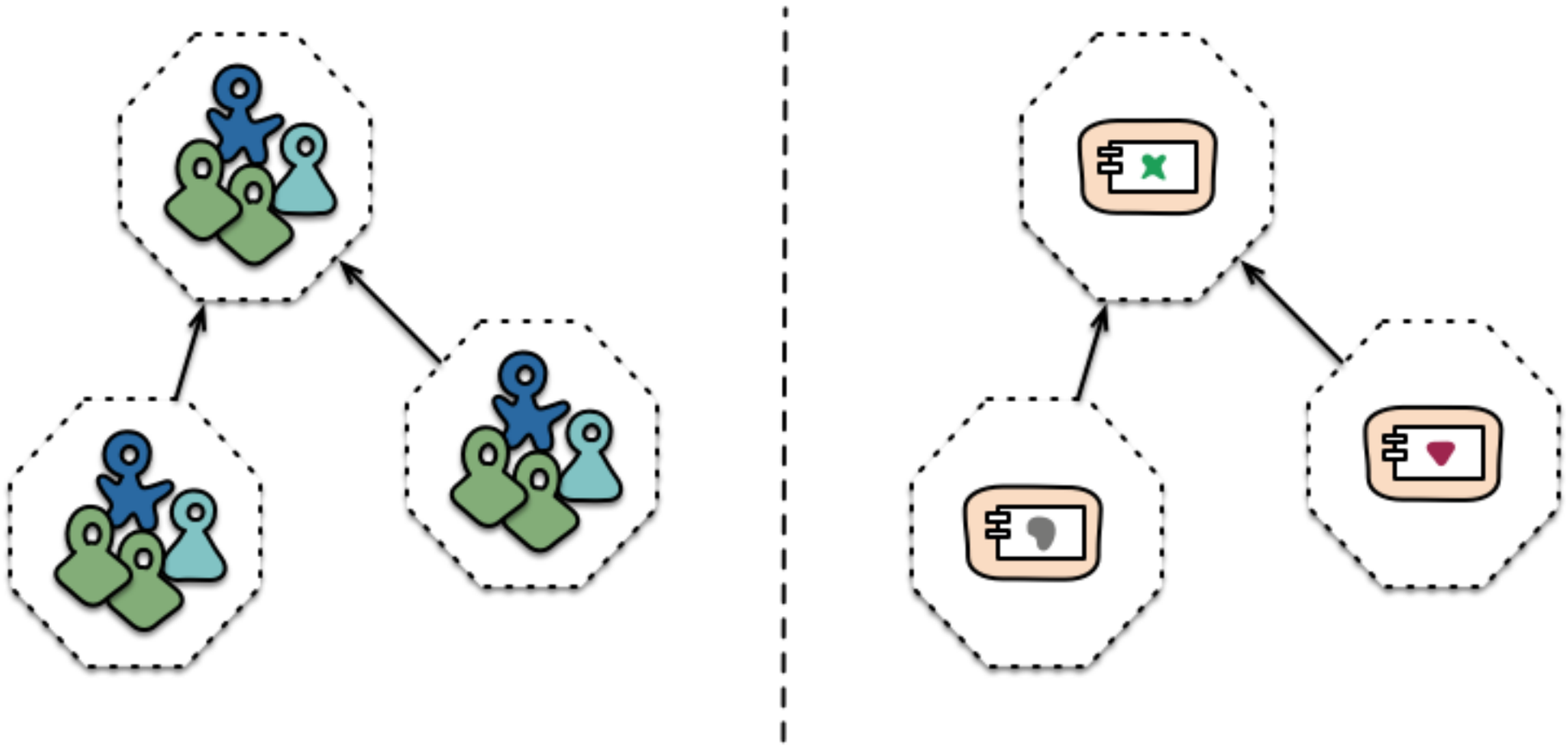
“team designs are the first draft of your architecture”

- Michael Nygard

vertical teams



inverse conway manoeuvre

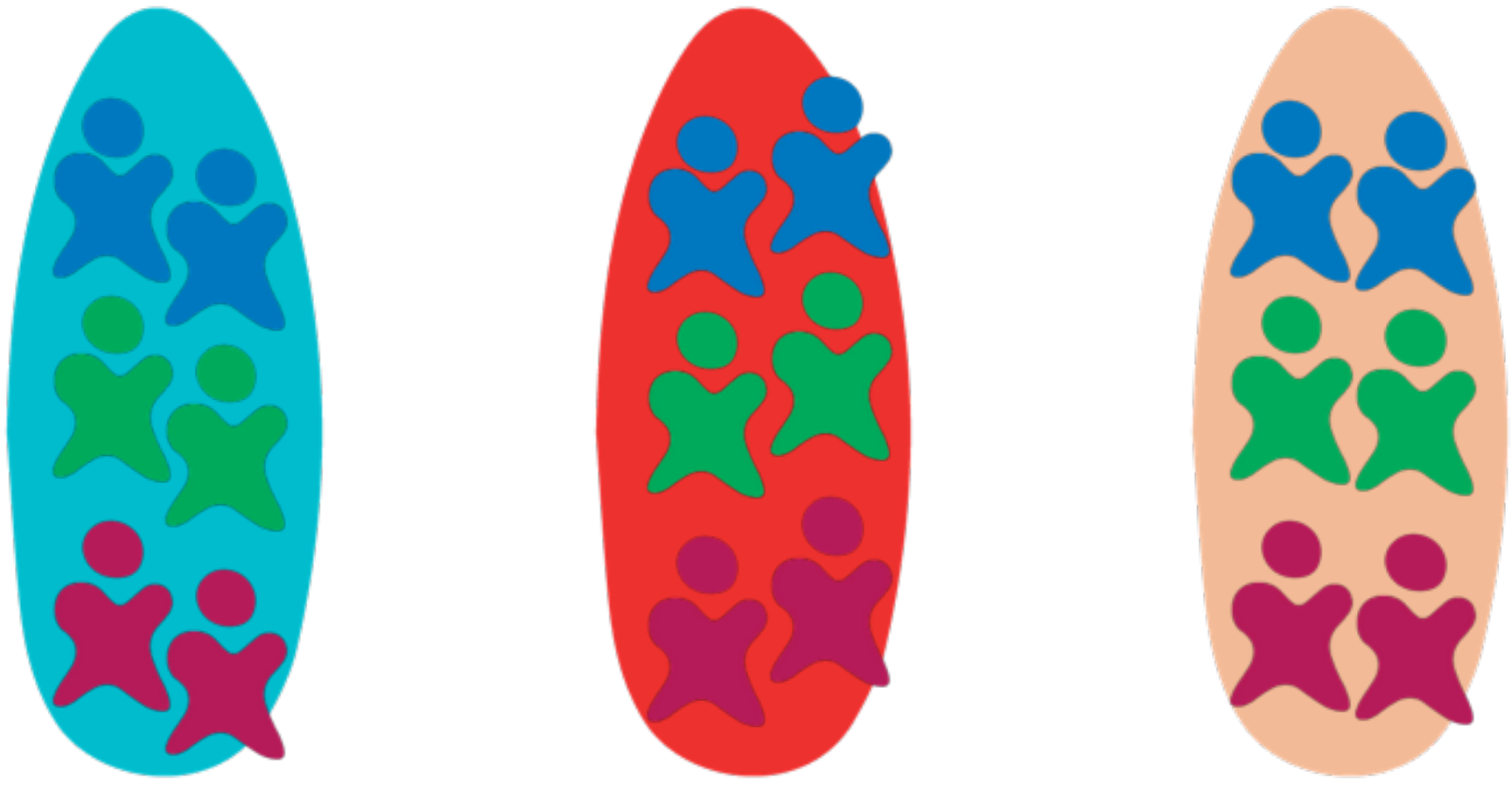


Cross-functional teams...

... organised around capabilities
Because Conway's Law

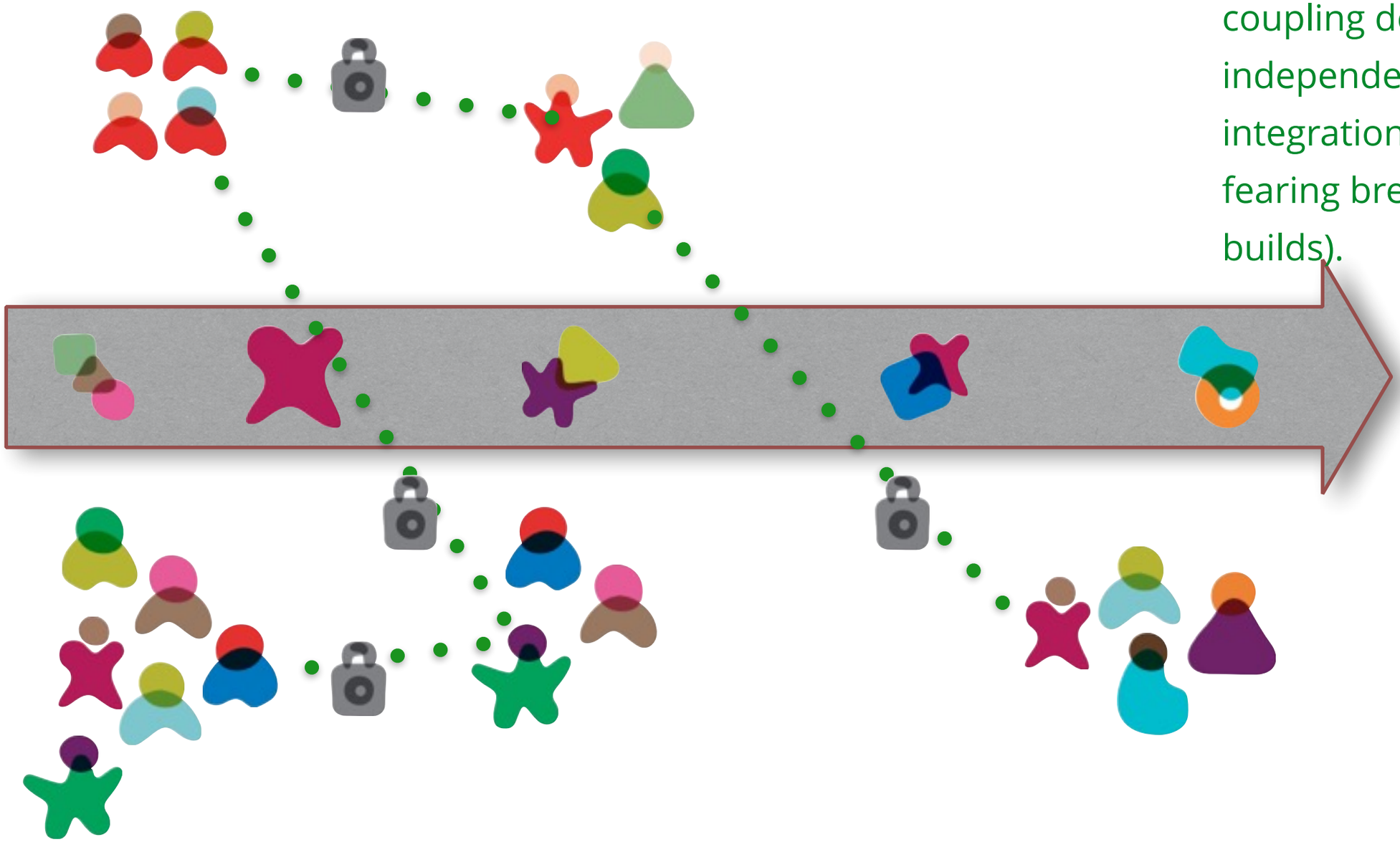
Build teams that look like
the architecture you want
(and it will follow).

the new world



payments statements rewards

continuous delivery



Teams with low efferent coupling deliver relatively independently into a common integration pipeline (without fearing breaking each others builds).

ThoughtWorks®

2. KEEP THINGS SMALL

`

small, single responsibility

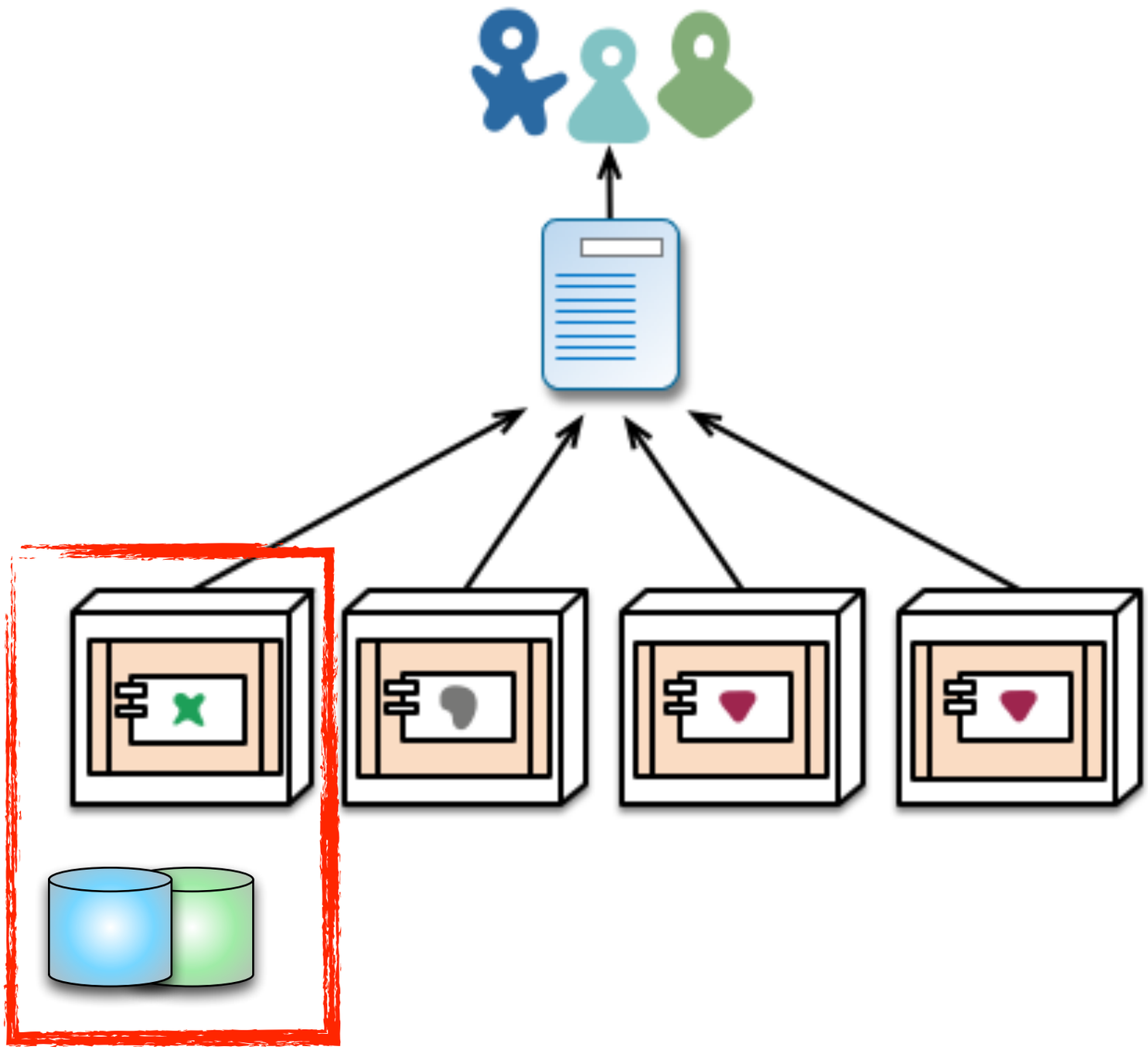
small enough to fit in your head

rewrite over maintain

(10—1000 LOC)-ish / service

single responsibility

decentralised governance



preparing for the unknown

future Rachel *is much smarter*

than ***present Rachel***

“with great power…”

THE LIFE OF A SOFTWARE ENGINEER.

CLEAN SLATE. SOLID FOUNDATIONS. THIS TIME I WILL BUILD THINGS THE RIGHT WAY.

MUCH LATER...

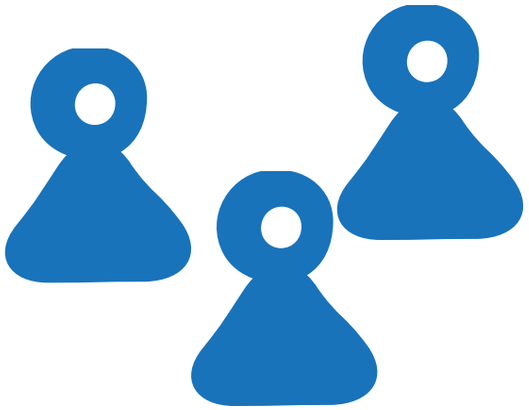
OH MY. I'VE DONE IT AGAIN, HAVEN'T I ?

MANU

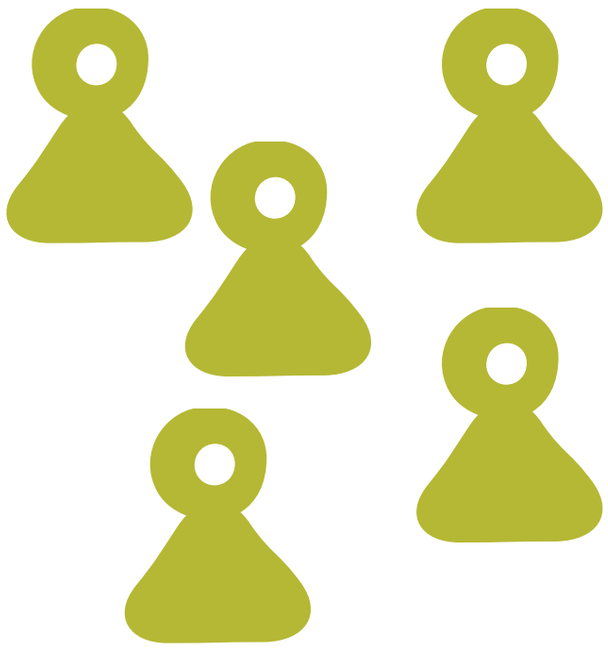
“fine grained SOA…?”

Products

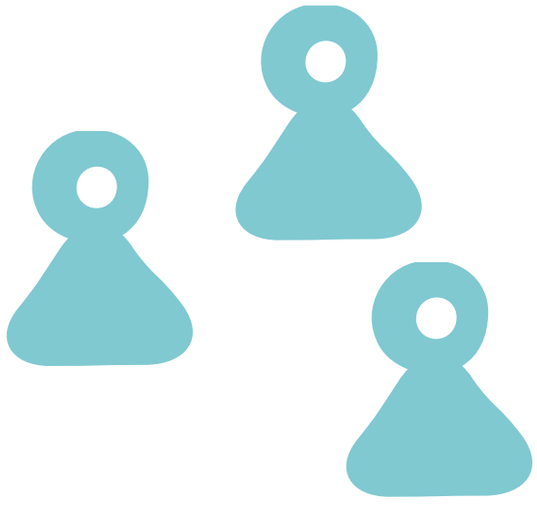
Customers



user interface

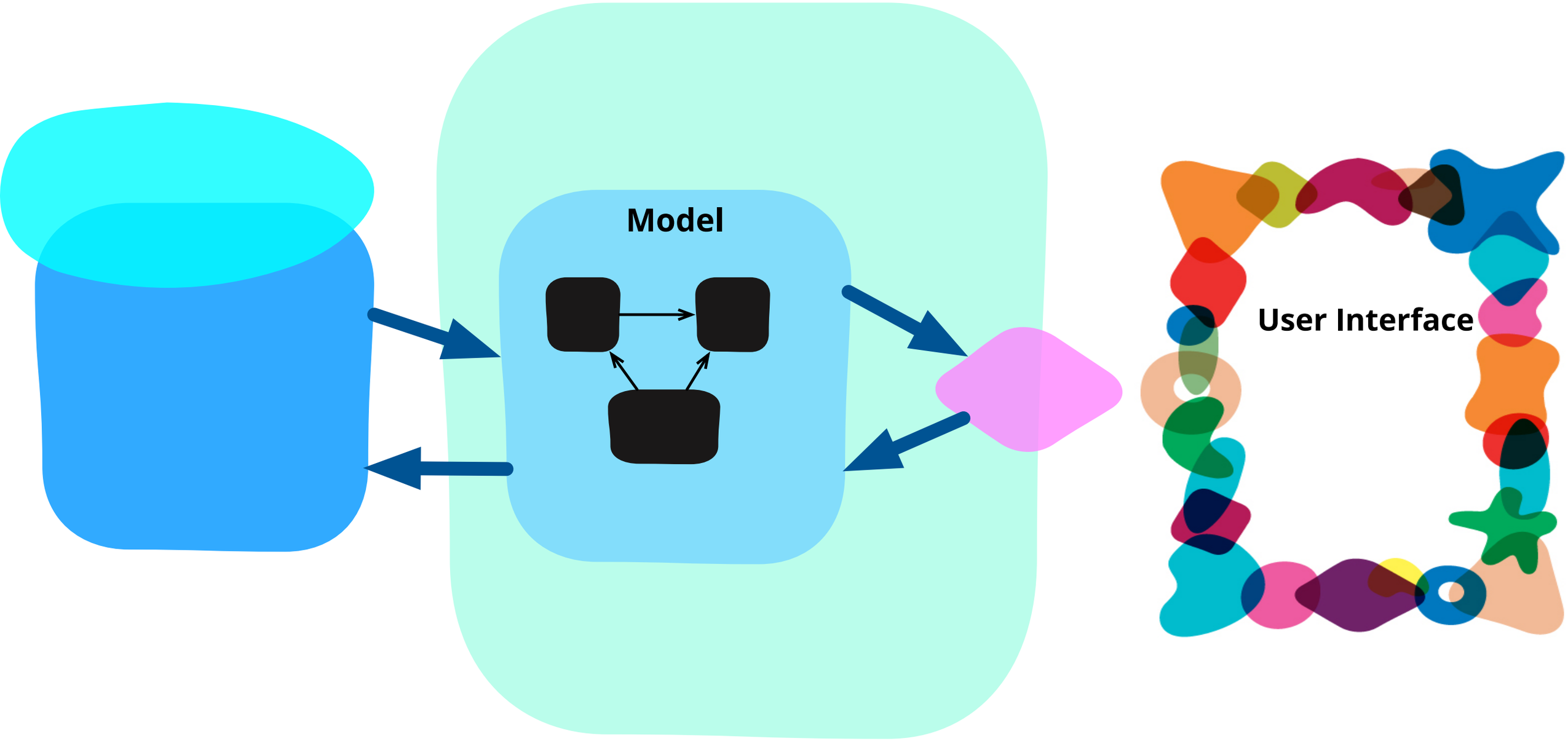


server-side

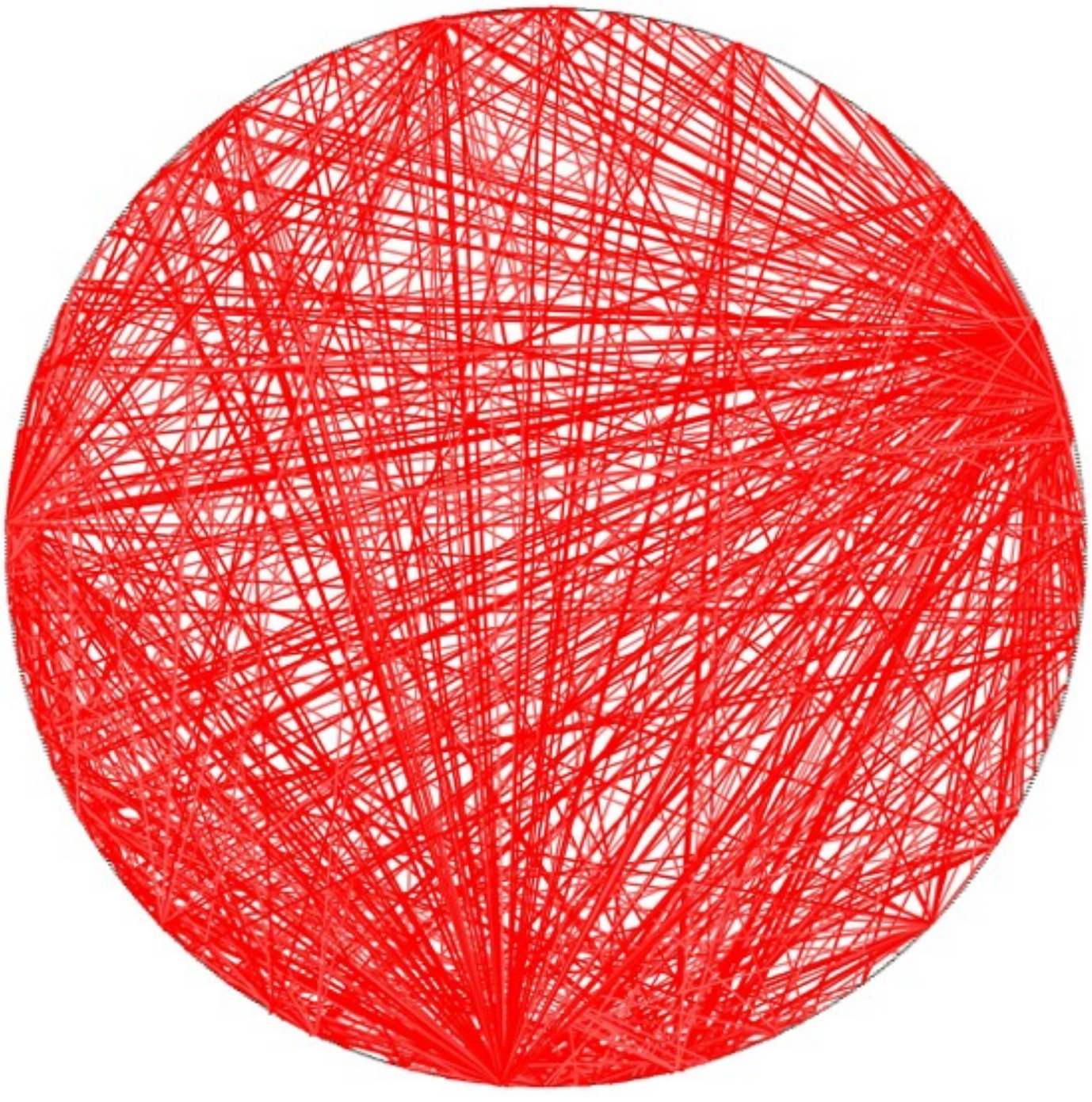


DBA

traditional monolith



ball of mud



monolith drawbacks

Complexity increases

Hard to change

Low reusability

Slow to deploy

Testing takes time

High cognitive load

Reliability and scale is hard

enter SOA

Explicit Boundaries

Shared Contract and Schema, not class

Policy Driven

Autonomous

http://www.infoq.com/articles/tilkov-10-soa-principles

Product

Customer

What "Traditional" SOA Got Right

- Breaking monoliths into services
- Focus on *integration* over *internal* coupling
- Prefer BASE to ACID

What "Traditional" SOA Missed

- Architecture is abstract until operationalized.
- Impact of Conway's Law
- The folly of trying to build "Uber" services (`Customer`)
- Didn't support easy change (ESB pattern)

so microservices?

fallacies of distributed computing

1.The network is reliable.
2.Latency is zero.
3.Bandwidth is infinite.
4.The network is secure.
5.Topology doesn't change.
6.There is one administrator.
7.Transport cost is zero.
8.The network is homogeneous.

principles SOA

Explicit Boundaries

Shared Contract and Schema, not class

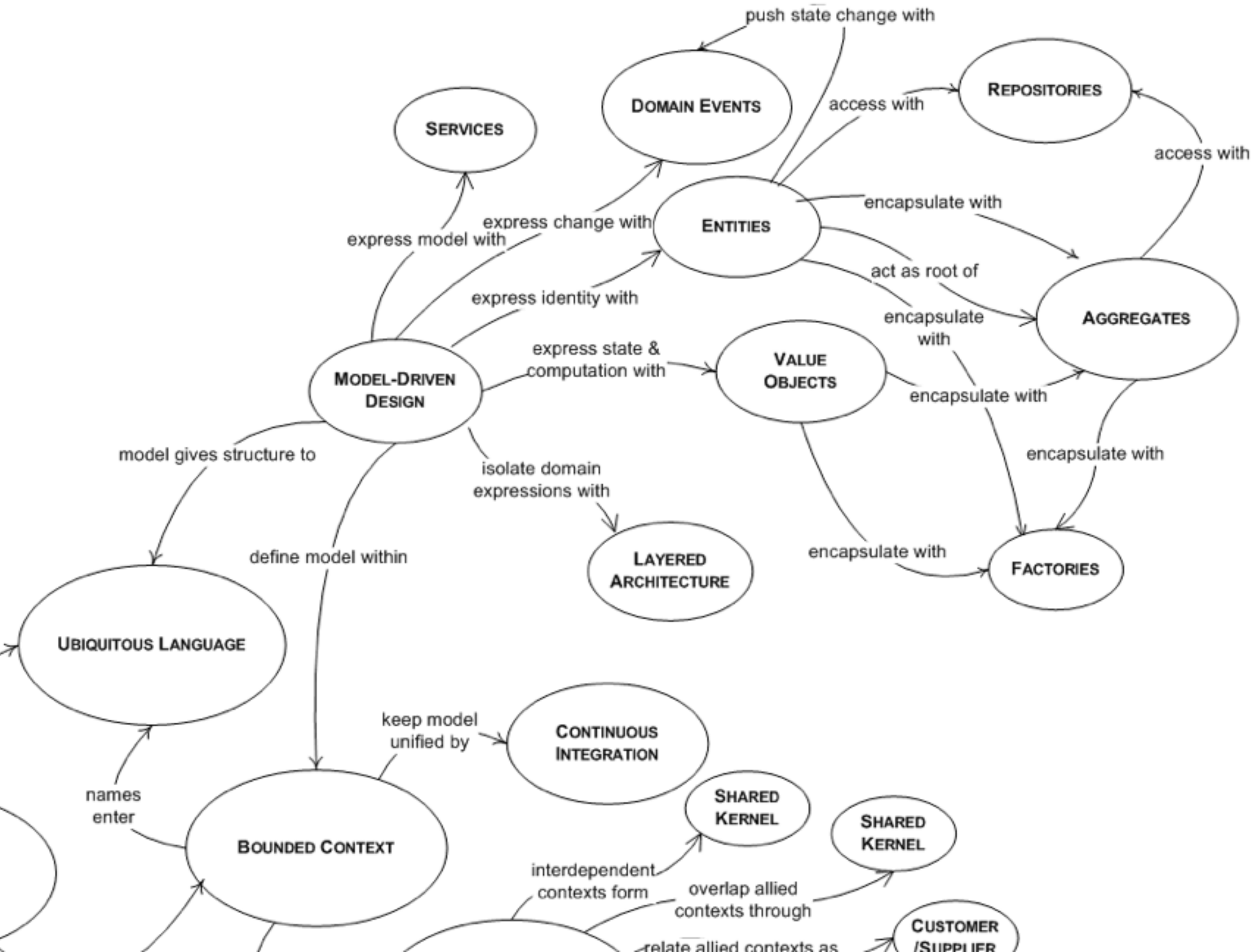
Policy Driven

Autonomous

http://www.infoq.com/articles/tilkov-10-soa-principles

boundaries are explicit

boundaries are explicit



Ordering

Shipping

Billing

Reporting

Catalog

Ordering

Shipping

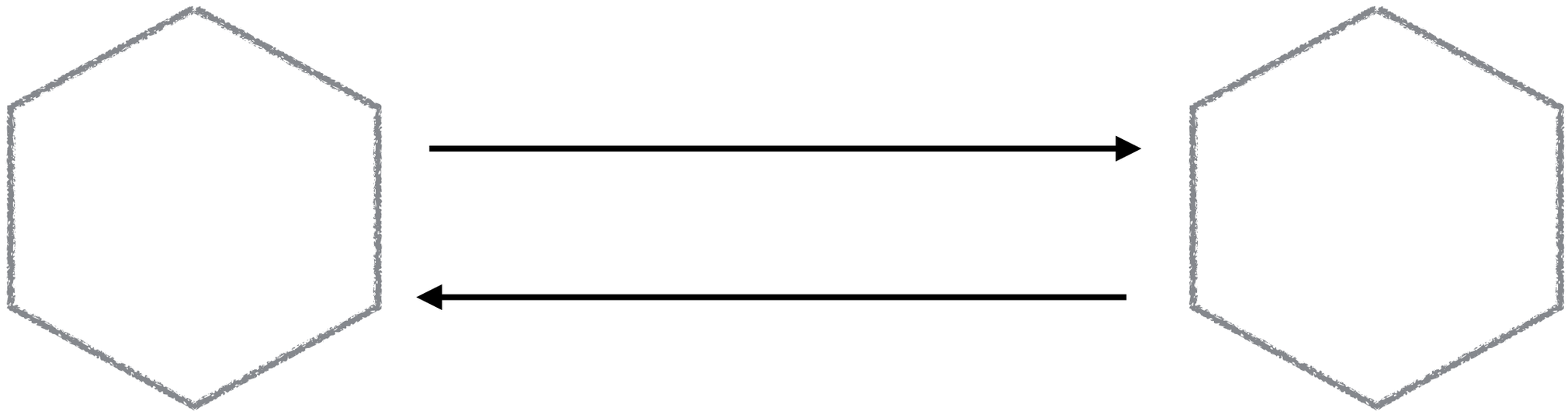
Billing

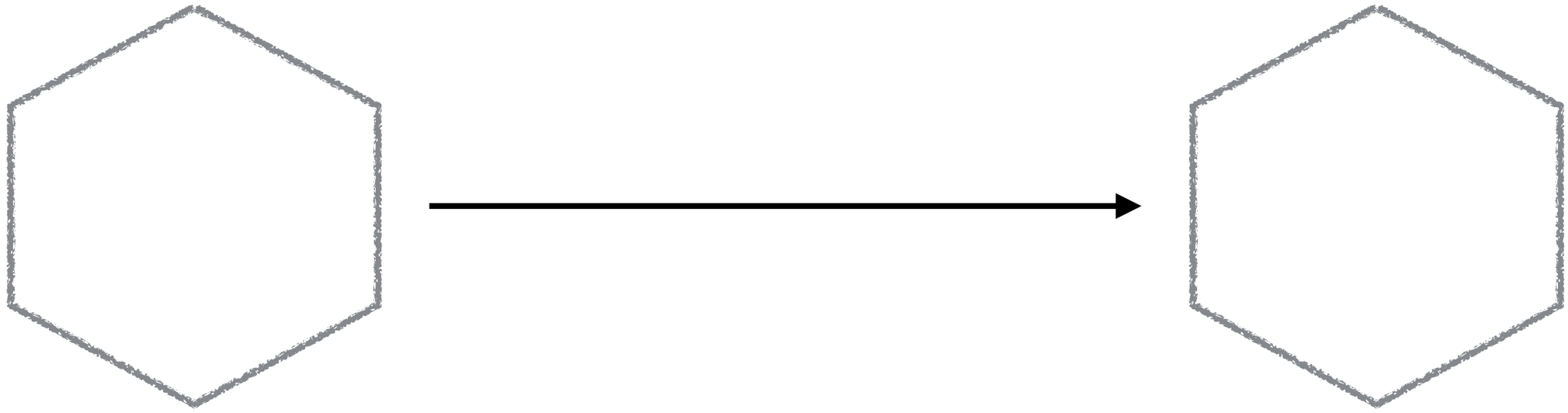
Reporting

Catalog

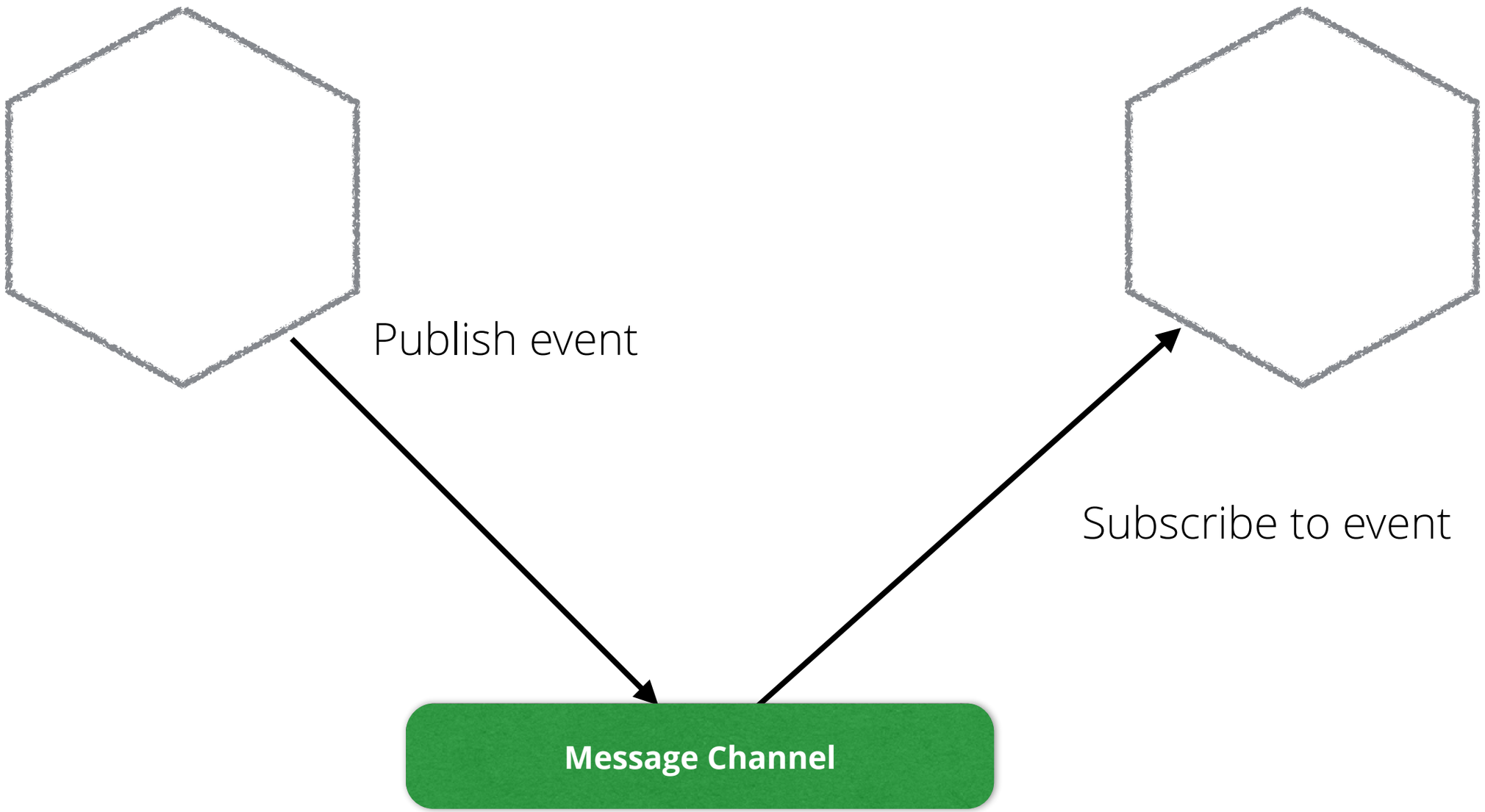
What about to integration?

asynchronous vs synchronous





eventual consistency



ACID

- **A**tomic
- **C**onsistent
- **I**solated
- **D**urable

BASE

- **B**asic **A**vailability
- **S**oft-state
- **E**ventual Consistency

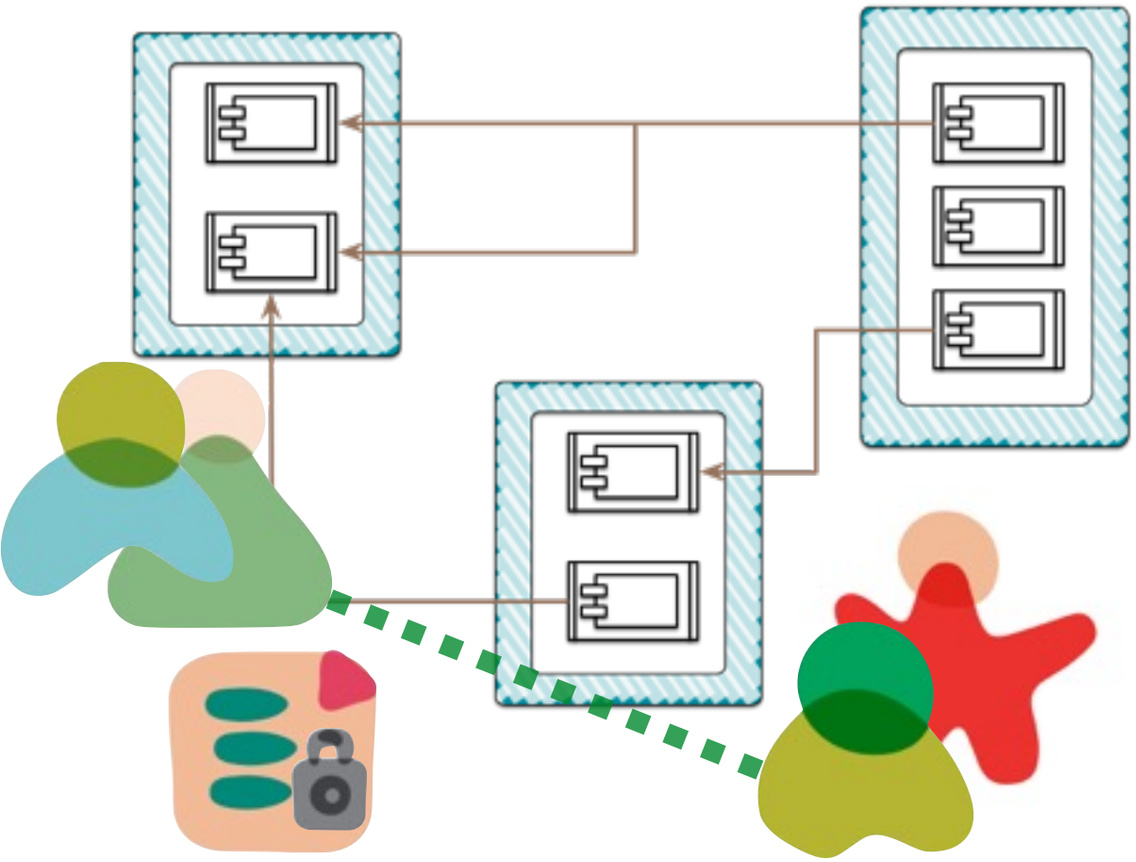
As your system becomes more distributed, prefer BASE to ACID...

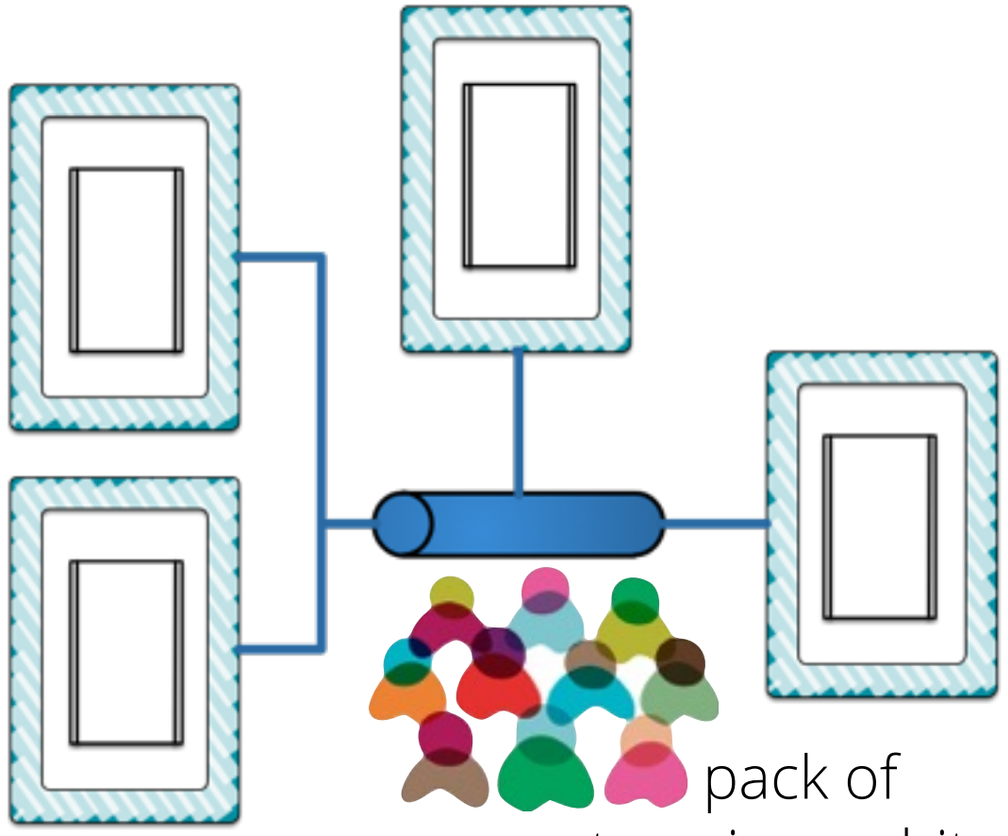
...because CAP Theorem

www.julianbrowne.com/article/viewer/brewers-cap-theorem

Formal proof: http://lpd.epfl.ch/sgilbert/pubs/BrewersConjecture-SigAct.pdf

Prefer *Choreography* to *Orchestration*





pack of
enterprise architects

Because Conway's Law!

traditional SOA /
ESB pattern

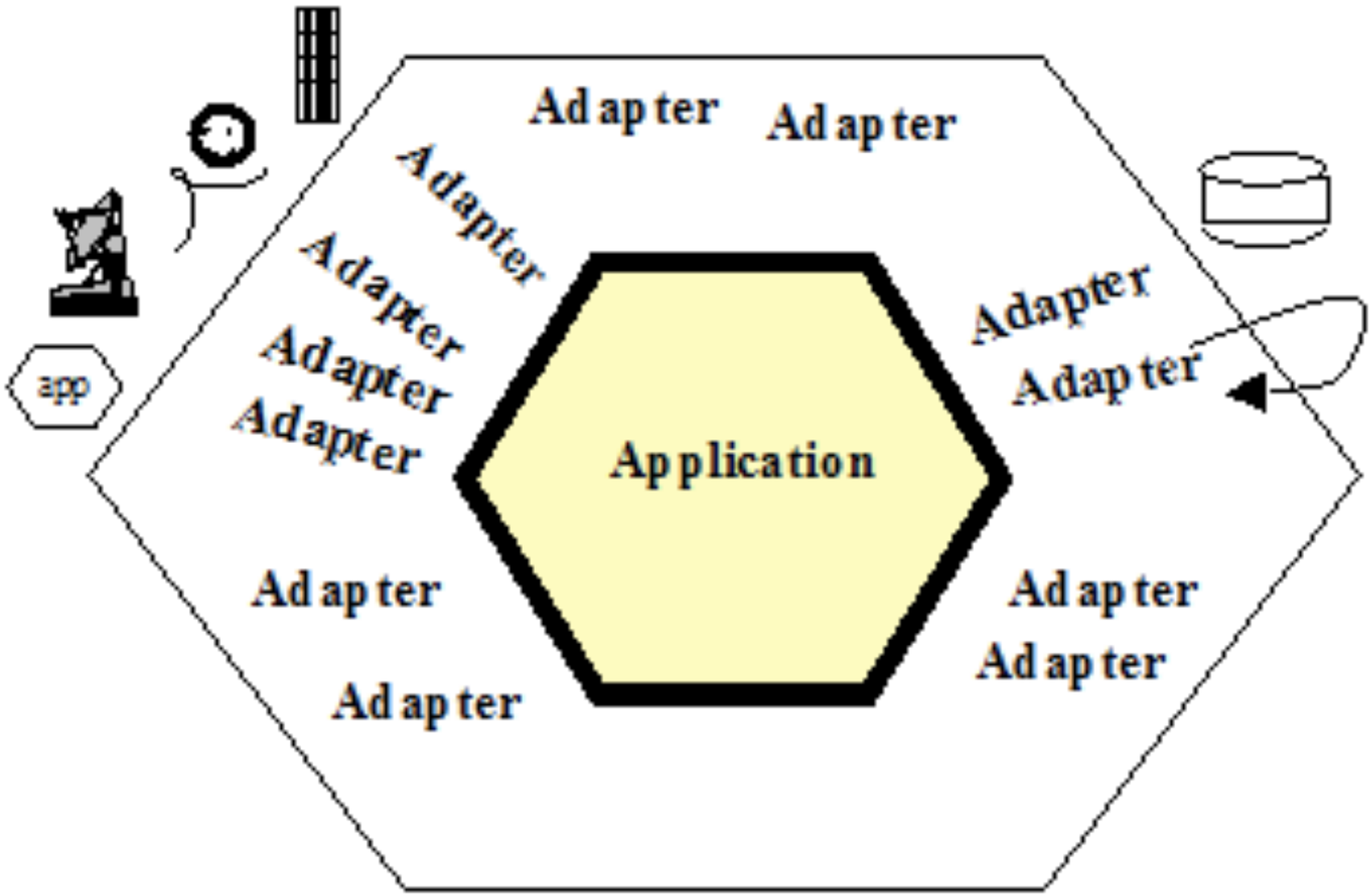
Standardize on integration, not platform

...but dont go crazy

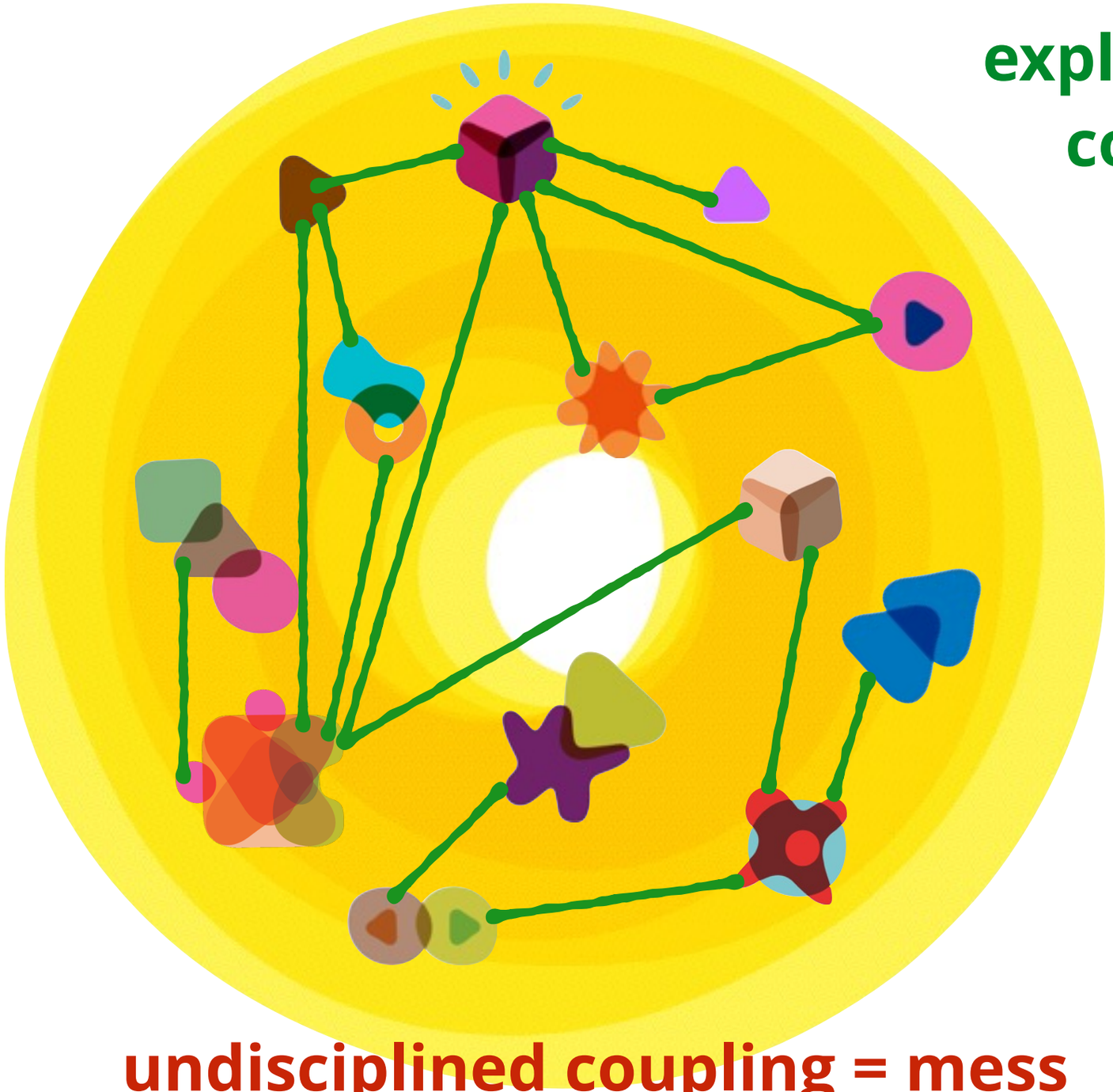
> *Have one, two or maybe three ways of integrating, not 20.*

> *Pick some sensible conventions, and stick with them.*

hexagonal architecture



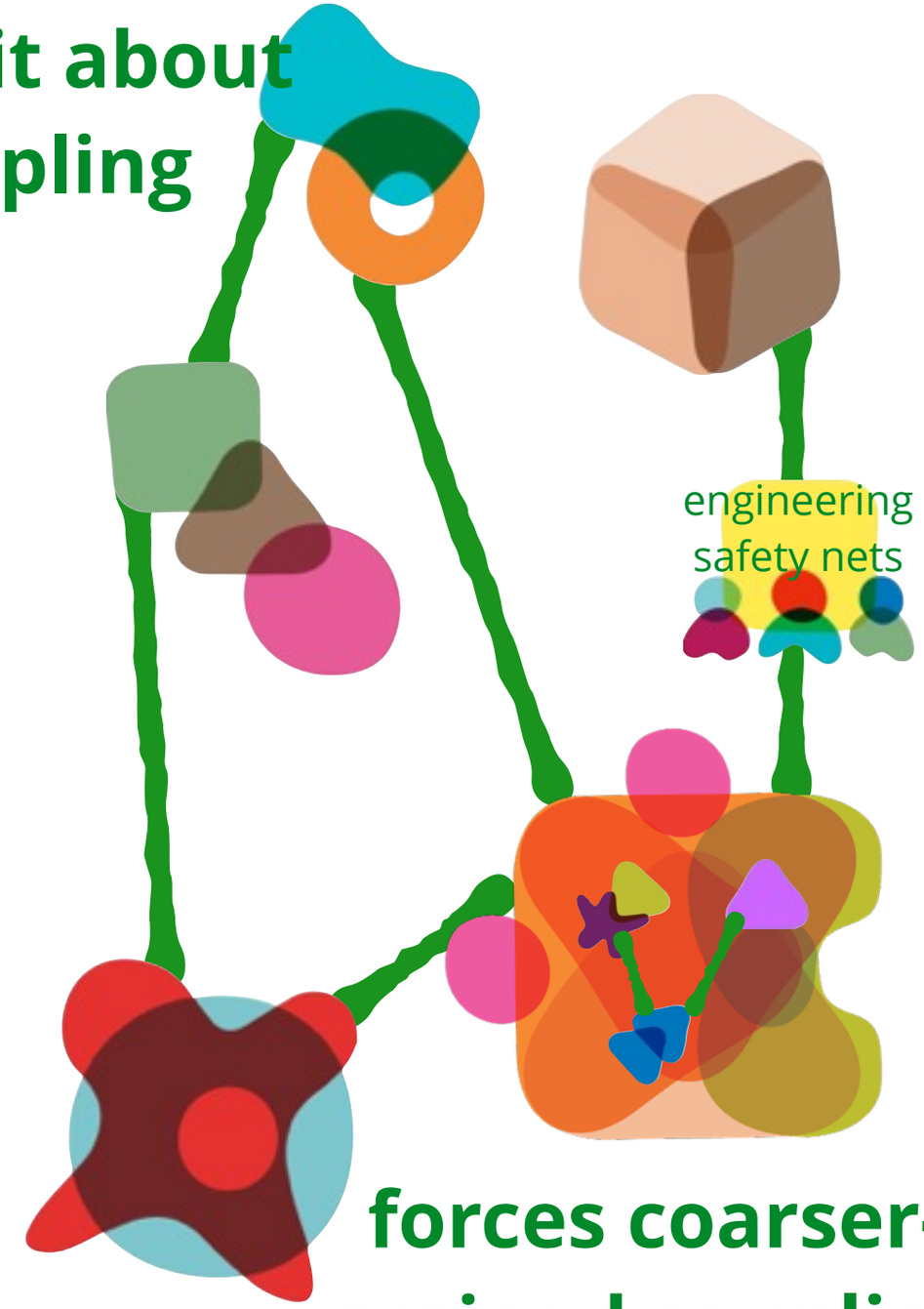
http://alistair.cockburn.us/Hexagonal+architecture



undisciplined coupling = mess

coupling dynamics become integration issues

explicit about coupling



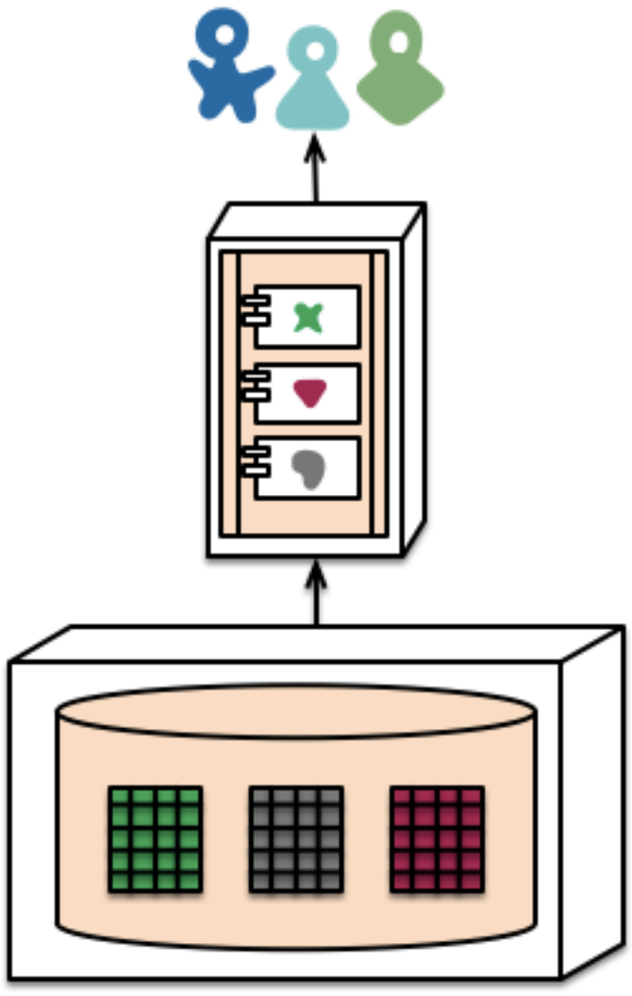
forces coarser-grained coupling

microservice architectures promote coupling from *application* to *integration* architecture.

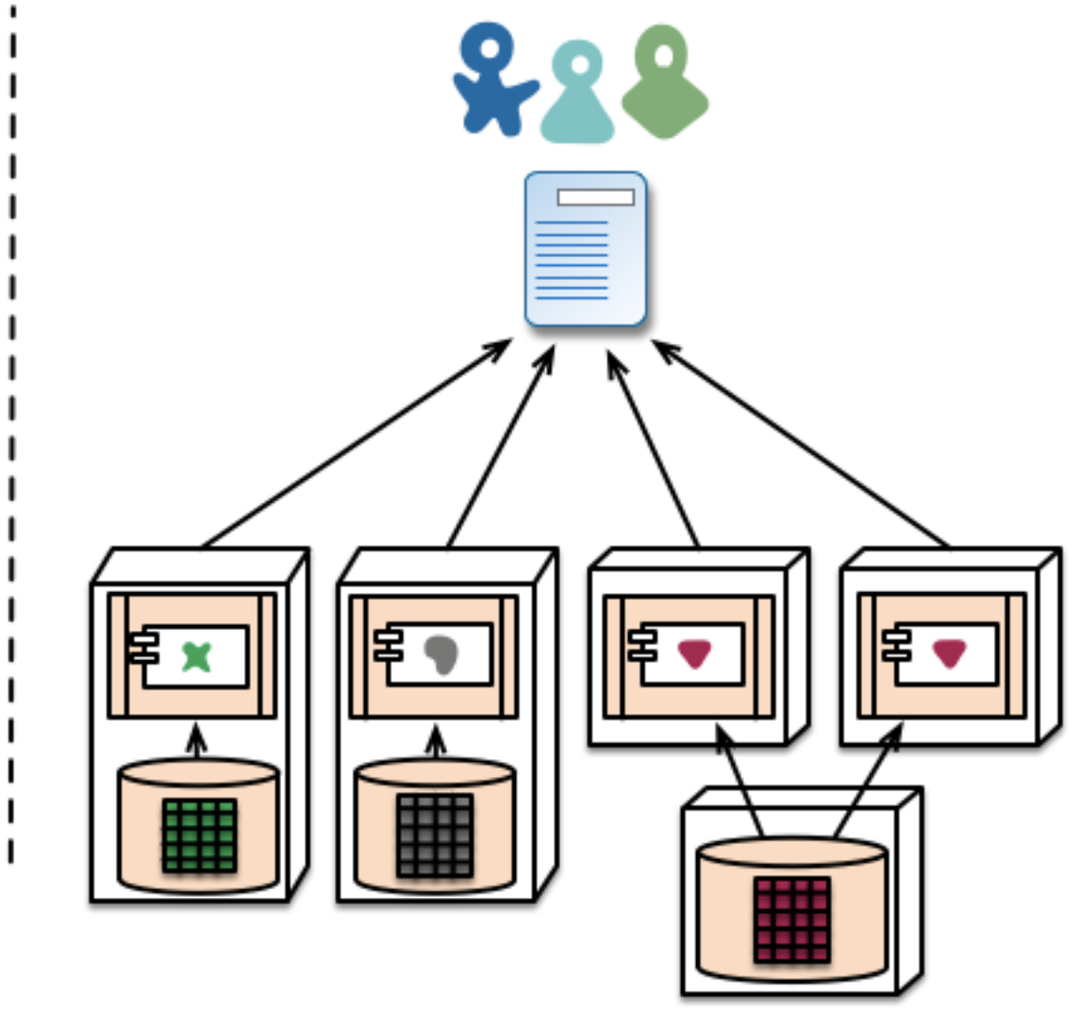
microservice architectures promote coupling from *application* to *integration* architecture.

- Pros:
  - explicit about coupling dynamics
  - forces coarser-grained coupling points
- Cons:
  - undisciplined coupling becomes a mess
  - transaction boundaries become an architectural issue

the monolith backlash?



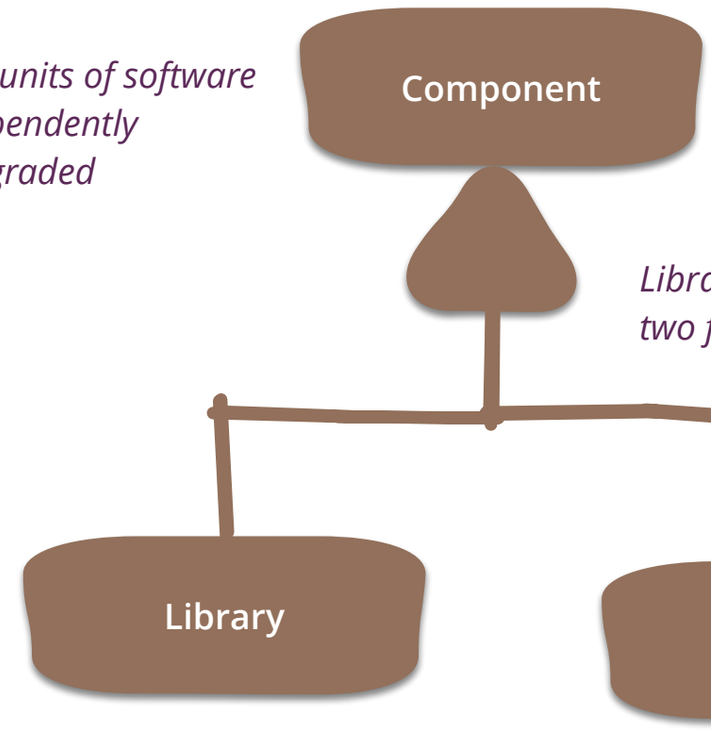
monolith - single database



microservices - application databases

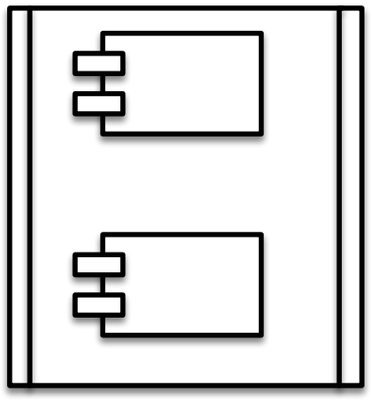
return to the monolith?

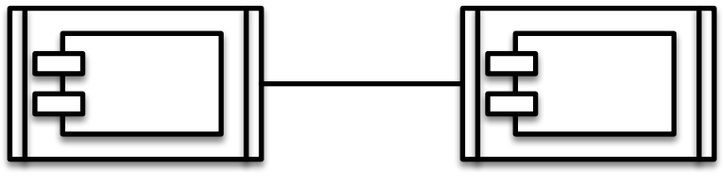
Components are units of software that can be independently replaced and upgraded



Libraries and Services are two forms of component

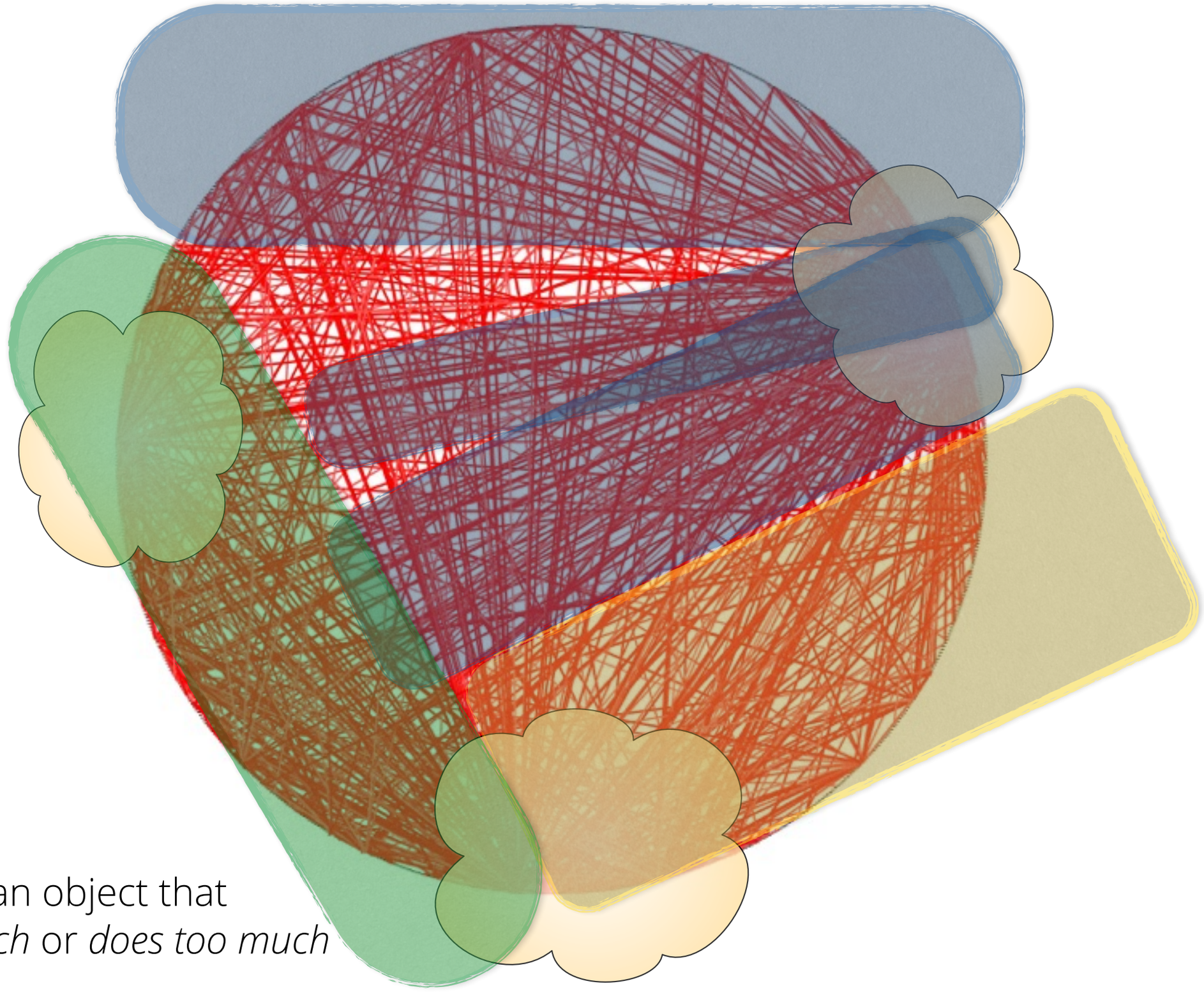
Libraries run within a single process, communicating through language function call mechanisms





Services run in separate processes, communicating with networking mechanisms such as HTTP or TCP/IP

partitioning by existing coupling



God Object: an object that
knows too much or *does too much*

maturity


YOU MUST BE
THIS TALL
TO HIT YOUR HEAD
ON THIS SIGN
Cyanide and Happiness © Explosm.net

Yesterday's best practice is tomorrow's anti-pattern.

We inadvertently build architectures to solve outdated problems.

ThoughtWorks®

3. EVOLVE YOUR ARCHITECTURE

last responsible moment

all the things to consider

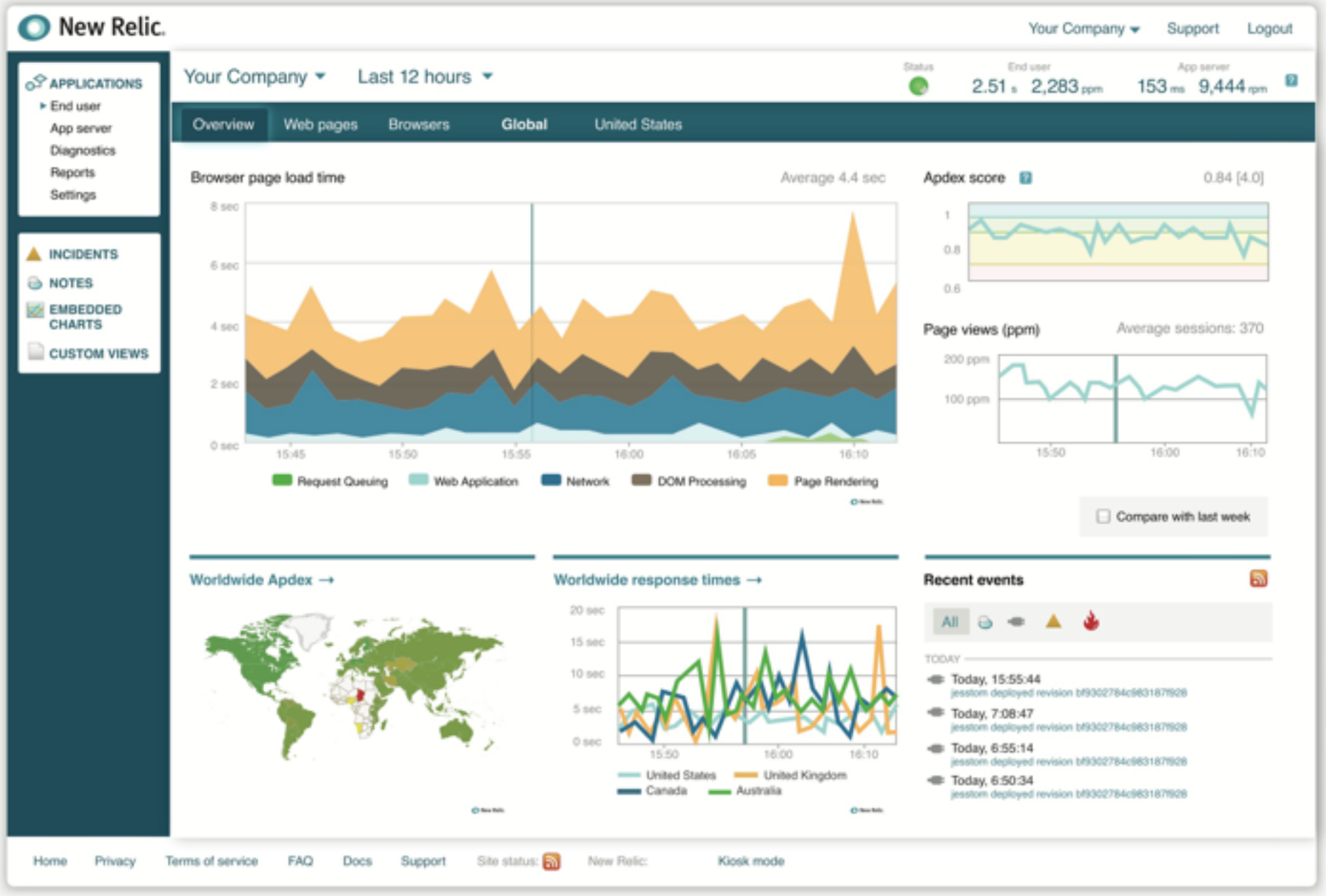
Network

Security

Devices/Hardware

Usability

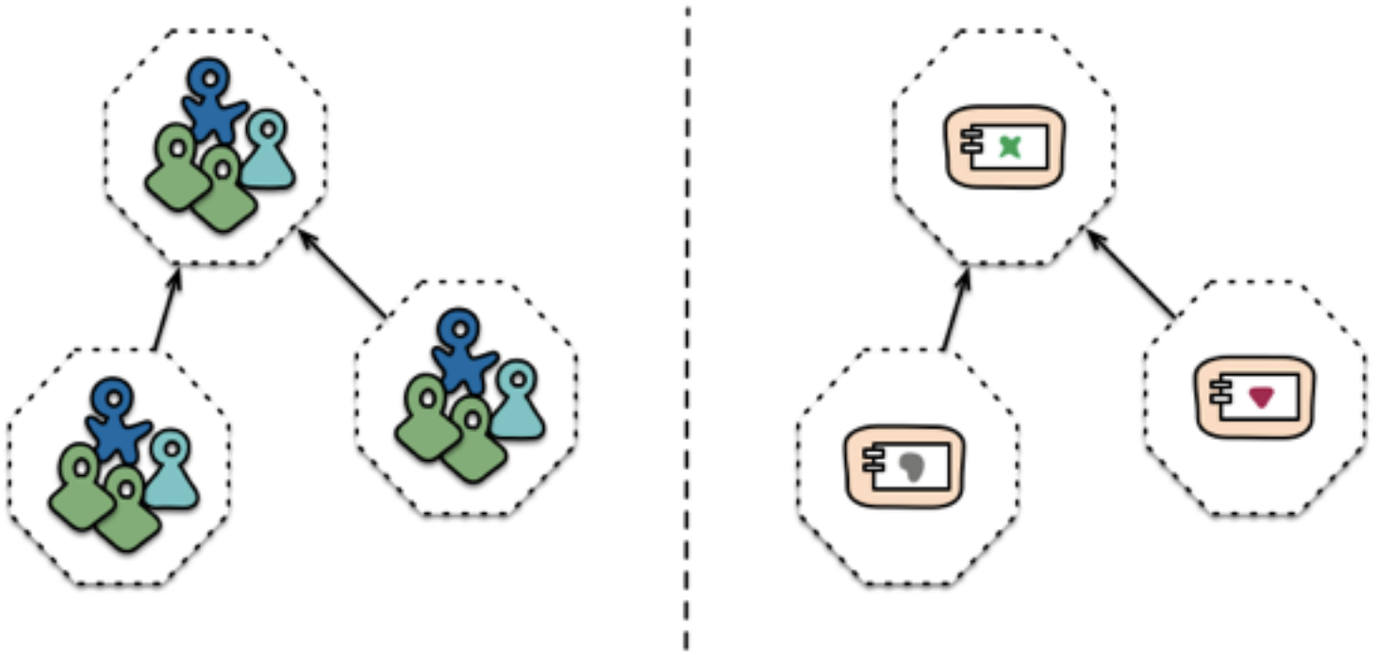
architect for evolvability



http://tutorials.jumpstartlab.com/images/elevate/newrelic_snapshot.jpg

architect for testability

conway's law



Cross-functional teams...

... organised around capabilities
Because Conway's Law

1. Conway's Law is THE LAW

2. Keep things small

3. Evolve your architecture

“hope is not a design method”

- Michael Nygard

if you fail to design for production your life will be filled with "excitement"

- Michael Nygard

THANK YOU!

@rachellaycock

ThoughtWorks®

Resources

Books:

- Continuous Delivery - Jez Humble, Dave Farley
- Working Effectively with Legacy Code - Michael Feathers
- Release It - Michael Nygard
- Domain Driven Design - Eric Evans

Articles/Blogs:

- Branch by Abstraction - Jez Humble: http://continuousdelivery.com/2011/05/make-large-scale-changes-incrementally-with-branch-by-abstraction/
- Branch by Abstraction - Paul Hammant: http://paulhammant.com/blog/branch_by_abstraction.html/
- Feature Toggles - Martin Fowler: http://martinfowler.com/bliki/FeatureToggle.html
- Evolutionary Architecture - Neal Ford: http://www.ibm.com/developerworks/views/java/libraryview.jsp?search_by=evolutionary+architecture+emergent+design:
- Ball of Mud: http://www.laputan.org/mud/
- Demming - http://leanandkanban.wordpress.com/2011/07/15/demings-14-points/
- Coding Horror: http://www.codinghorror.com/blog/2007/11/the-big-ball-of-mud-and-other-architectural-disasters.html
- Who needs an architect: http://martinfowler.com/ieeeSoftware/whoNeedsArchitect.pdf
- Evolutionary Architecture and Emergent Design: http://www.ibm.com/developerworks/java/library/j-eaed1/index.html
- Strangler Application: http://martinfowler.com/bliki/StranglerApplication.html
- Microservices: http://www.infoq.com/presentations/Micro-Services and http://yobriefca.se/blog/2013/04/29/micro-service-architecture/ and http://davidmorgantini.blogspot.co.uk/2013/08/micro-services-what-are-micro-services.html