



aRChiTeCtuRe fOr cONTiNuoUs DeLiVeRy



ThoughtWorks®

NEAL FORD

Director / Software Architect / Meme Wrangler



@neal4d

nealford.com

aRChiTeCtuRe
fOr
CONTiNuOUs DeLiVeRy

Continuous Delivery

integration as an engineering practice over time

integration as an engineering practice over time

1980

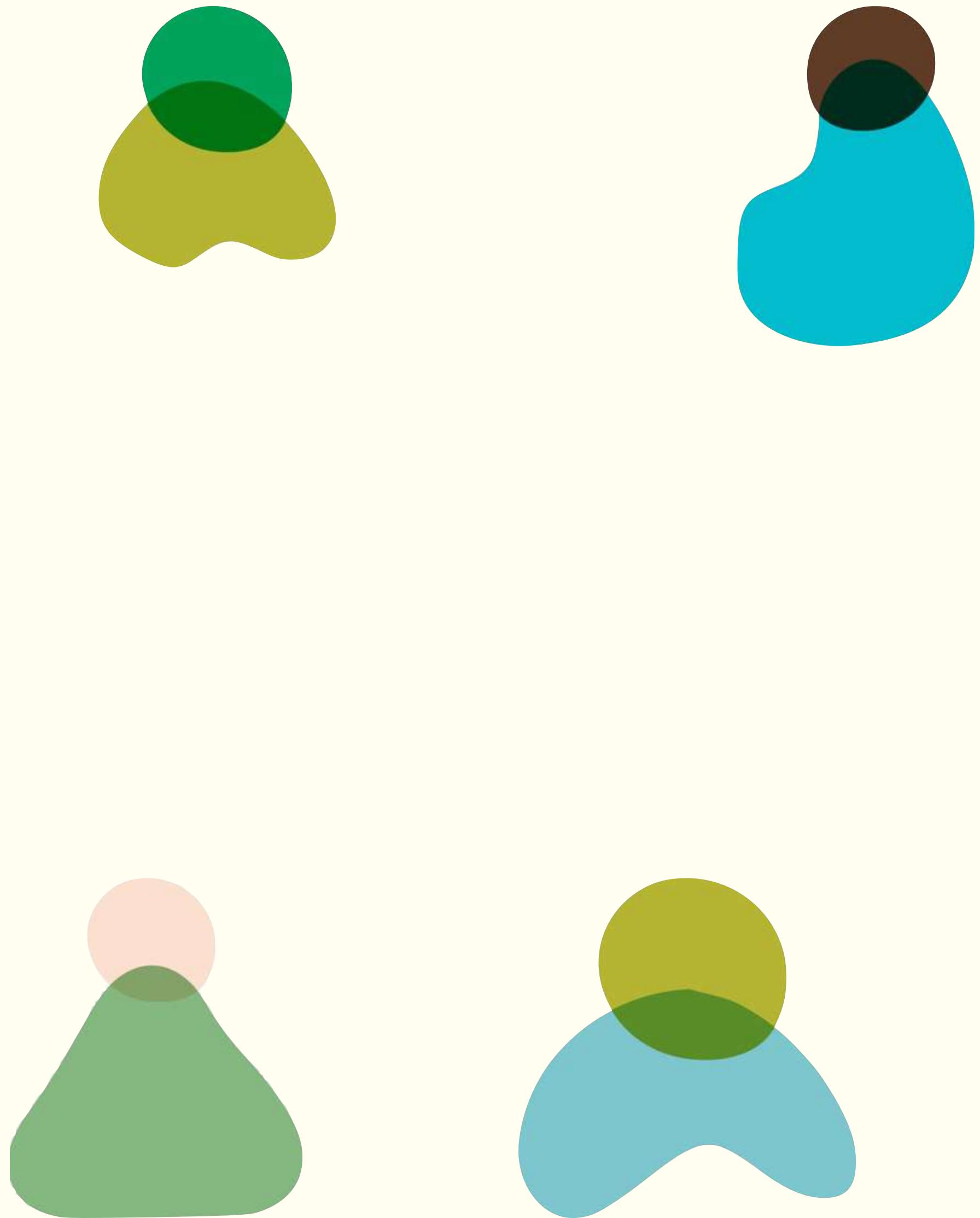
1990

2000

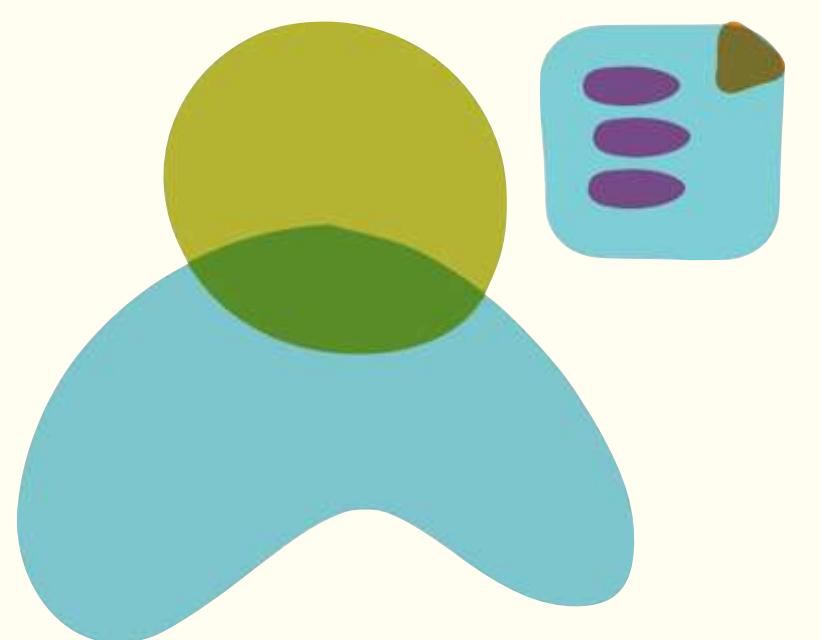
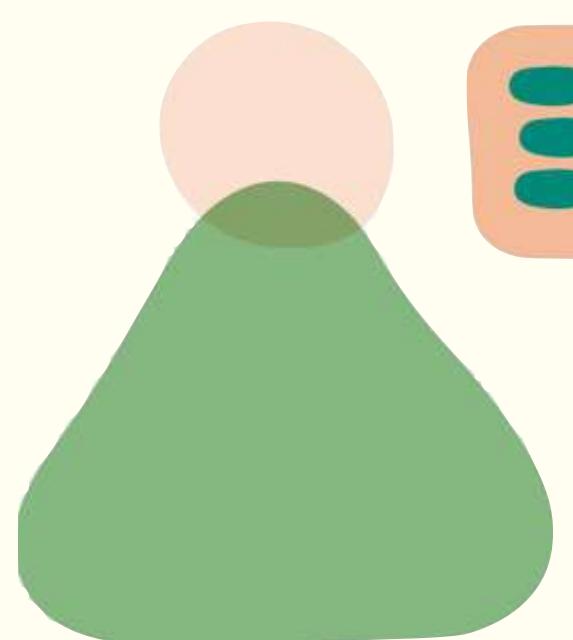
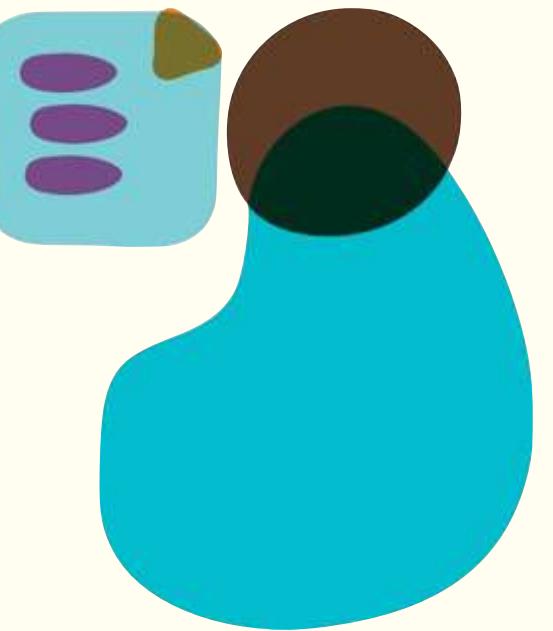
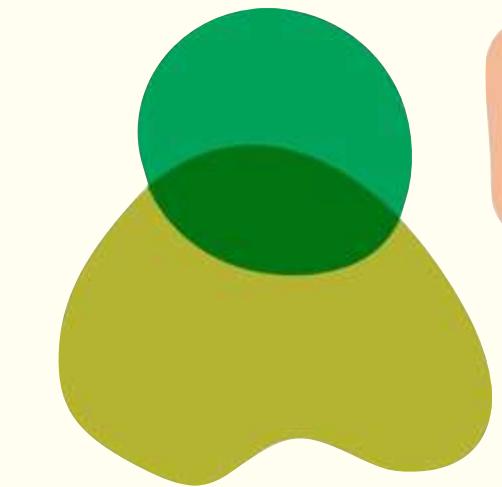
2010

current
day

1980

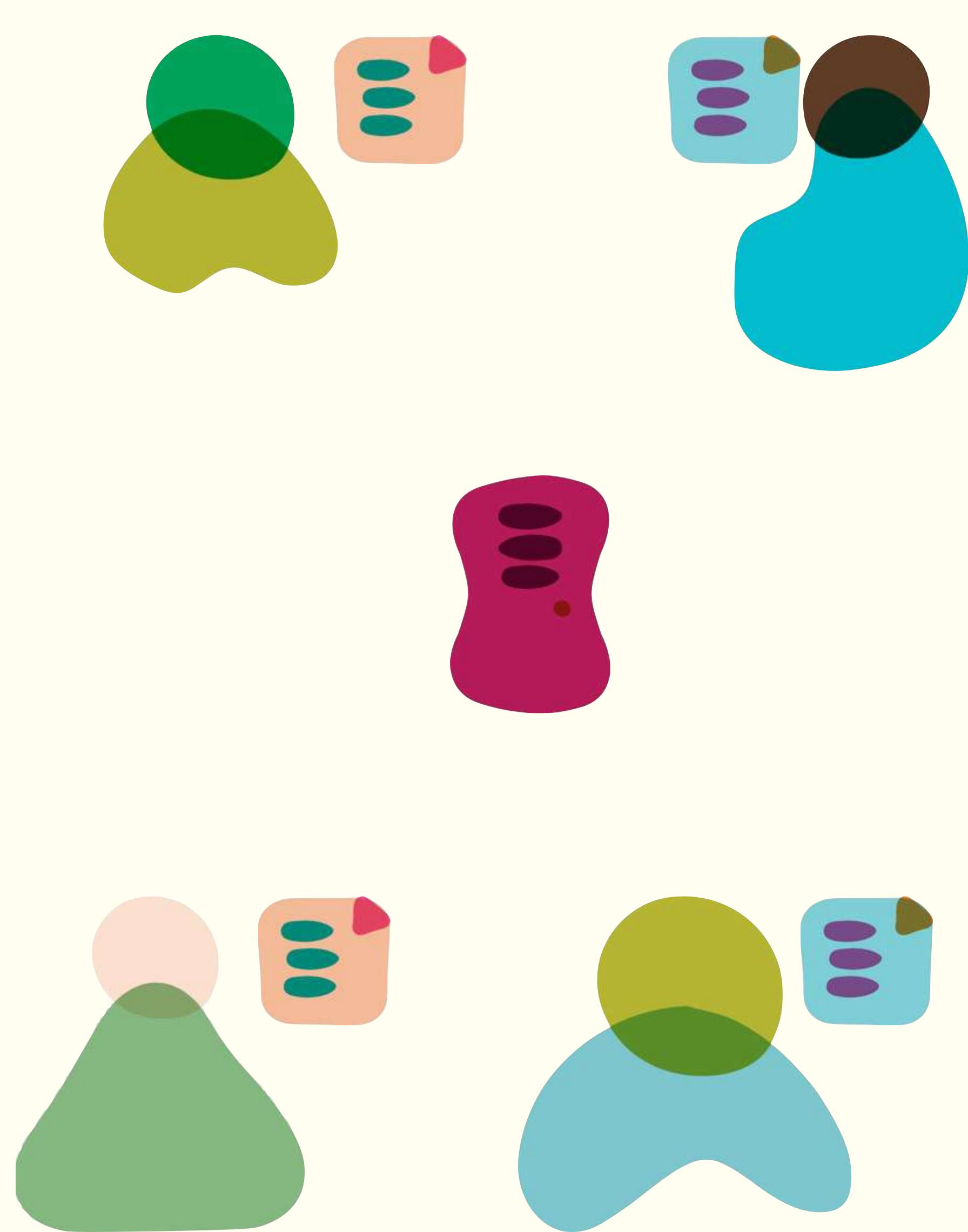


1980

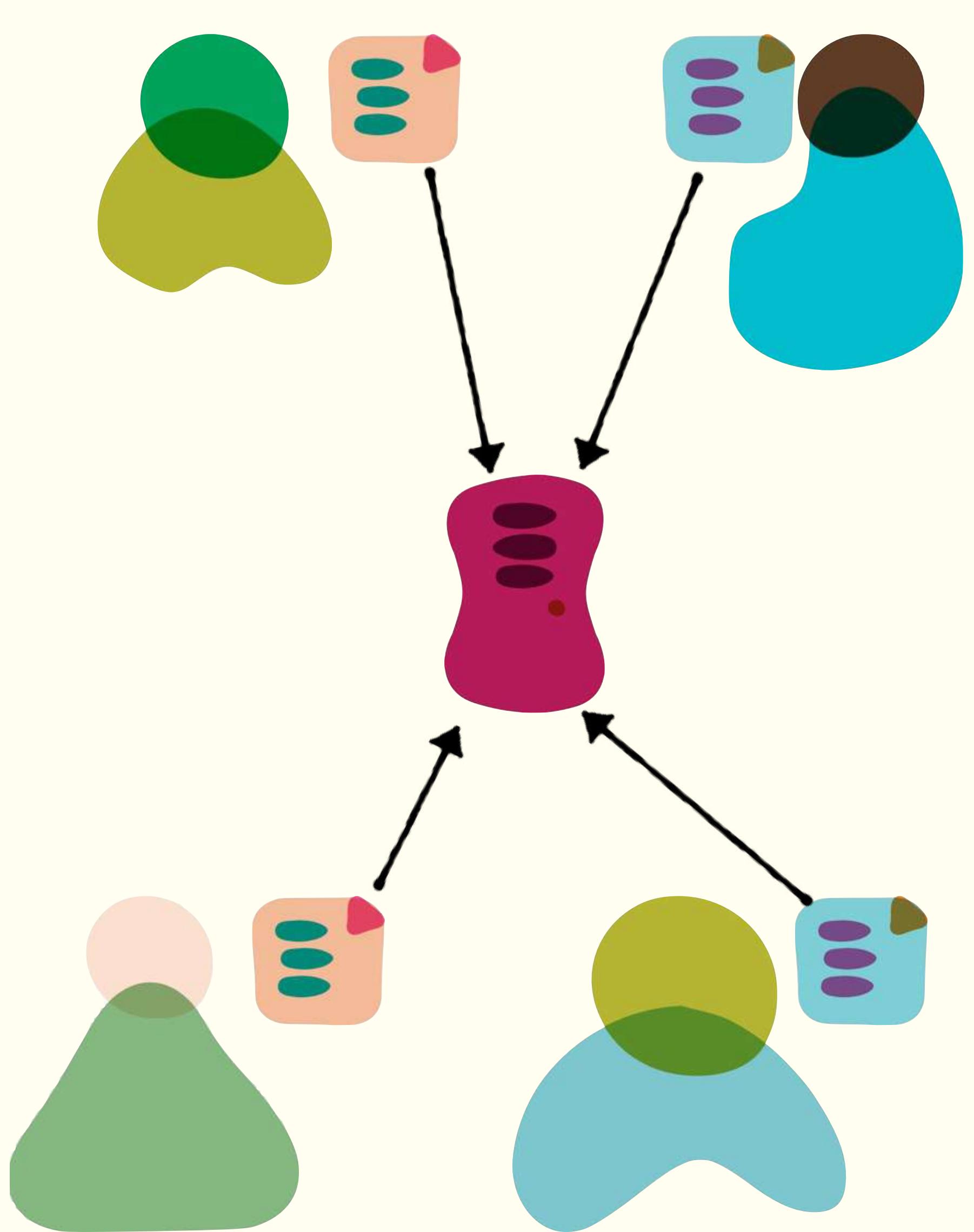


1980

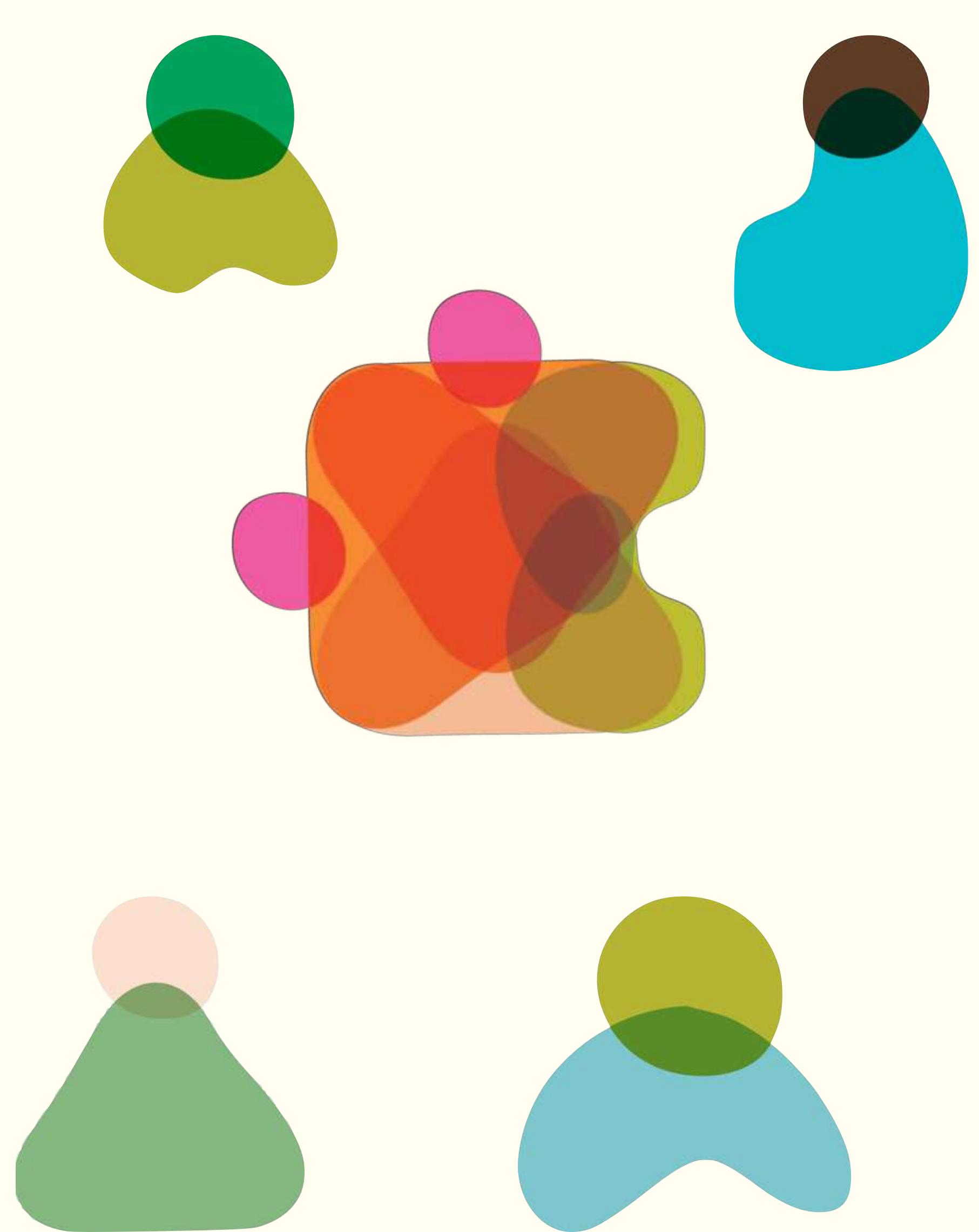
integration
phase



1980

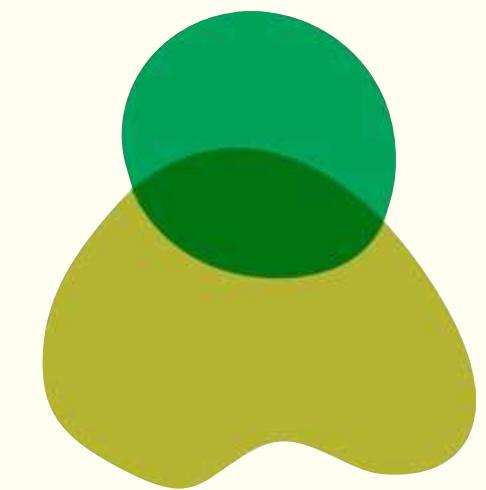


1980

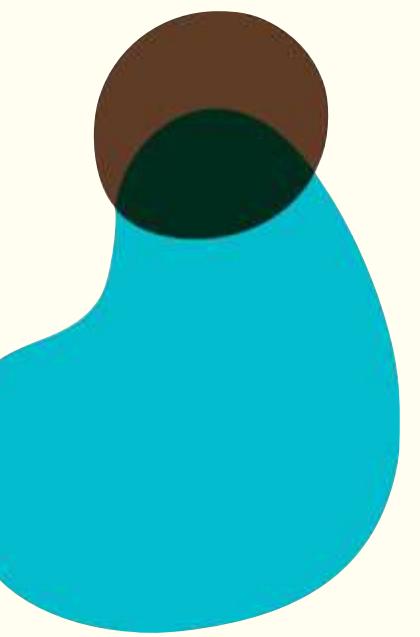


1980

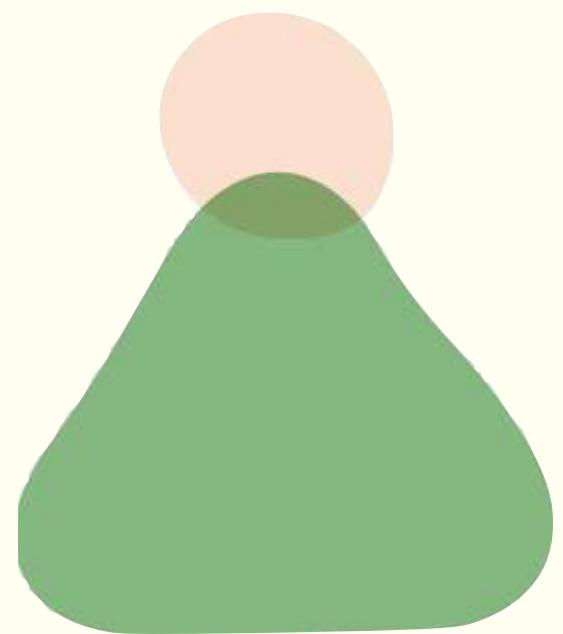
!!?!



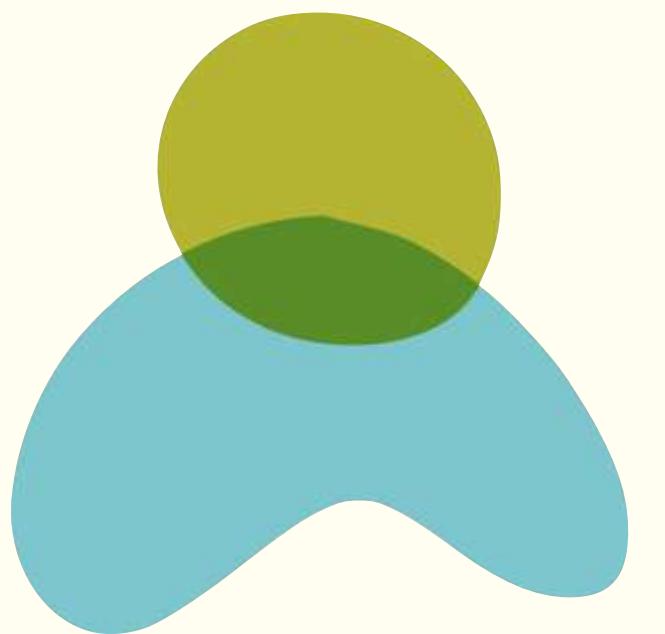
!??!?



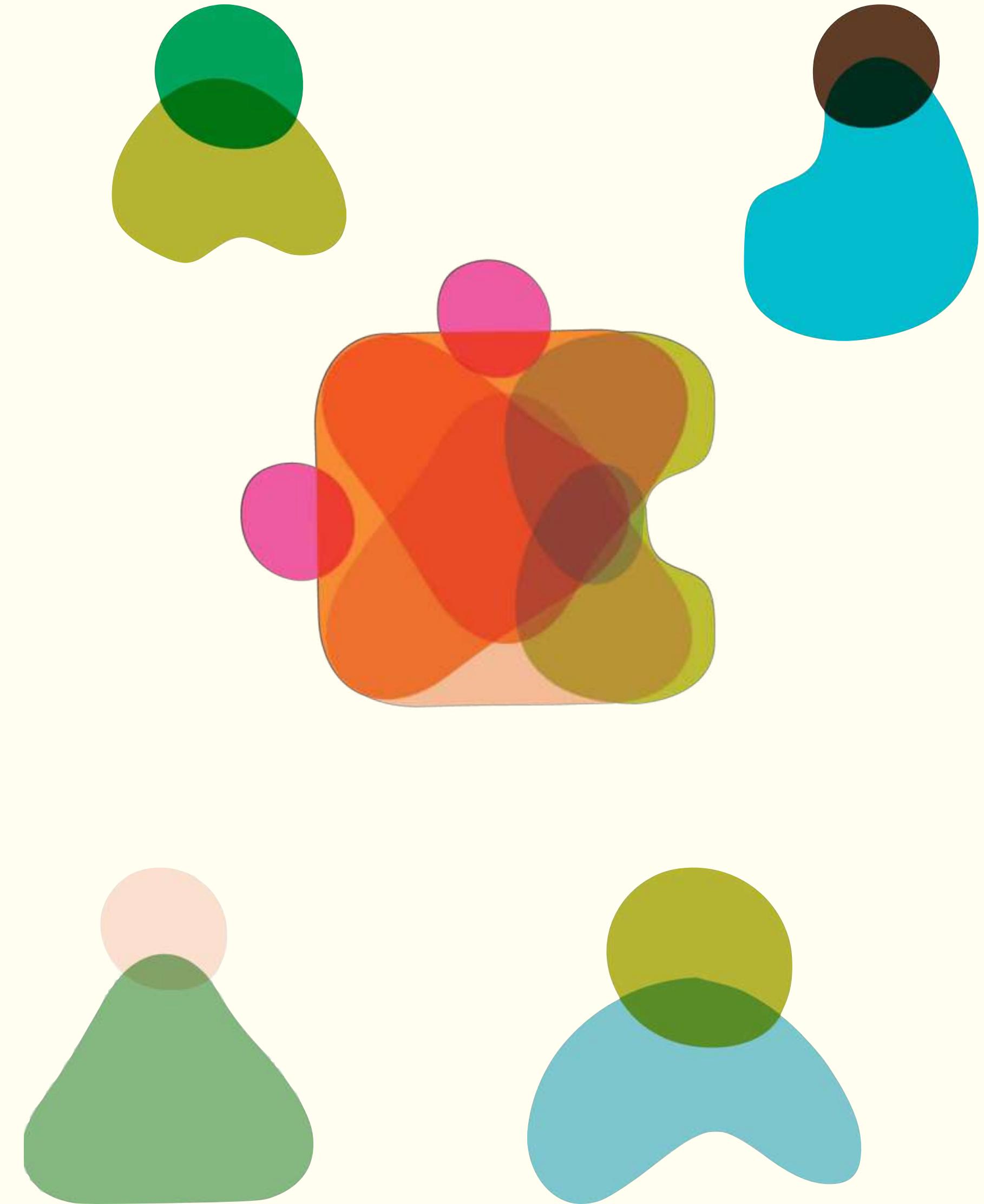
?!?



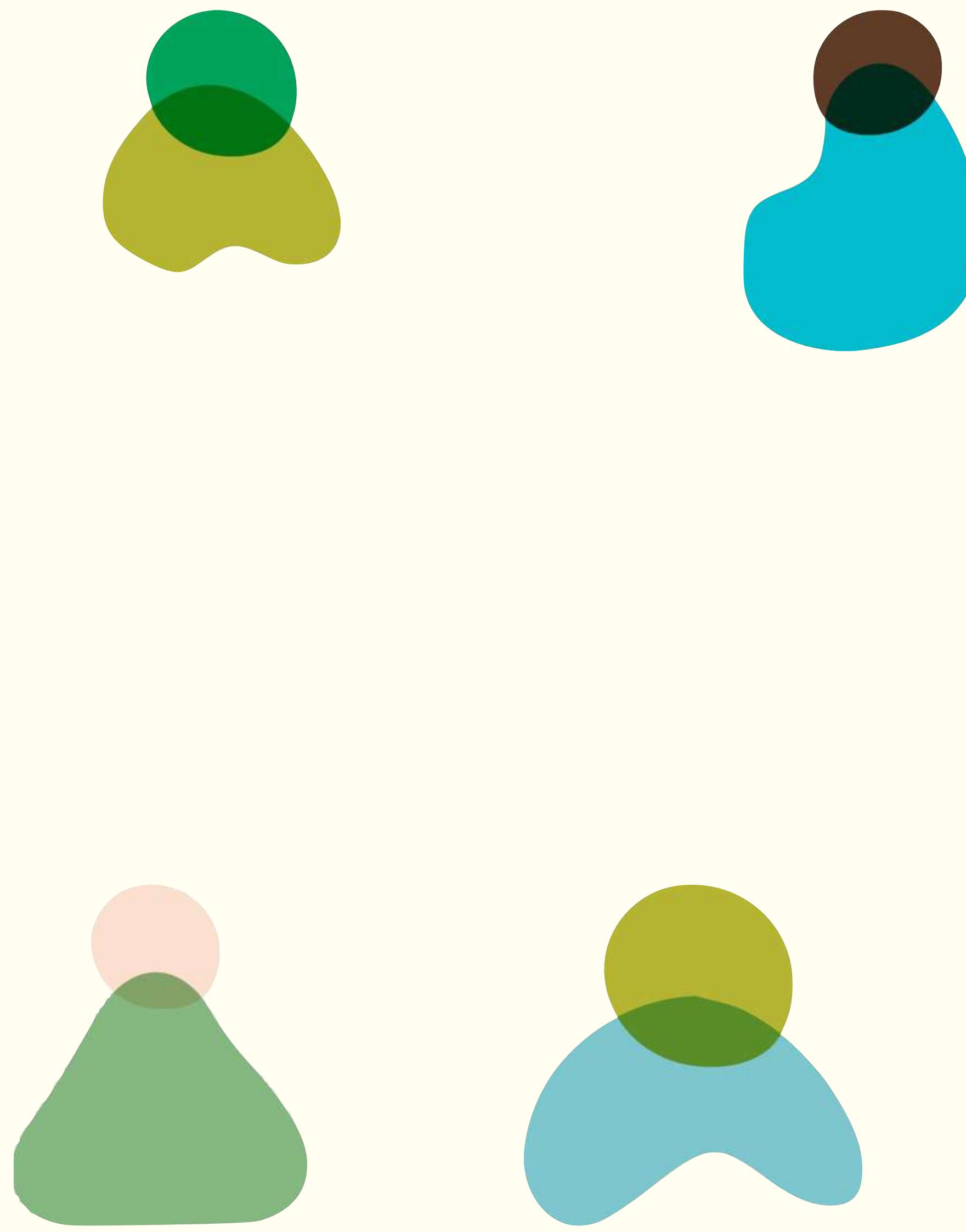
?!?



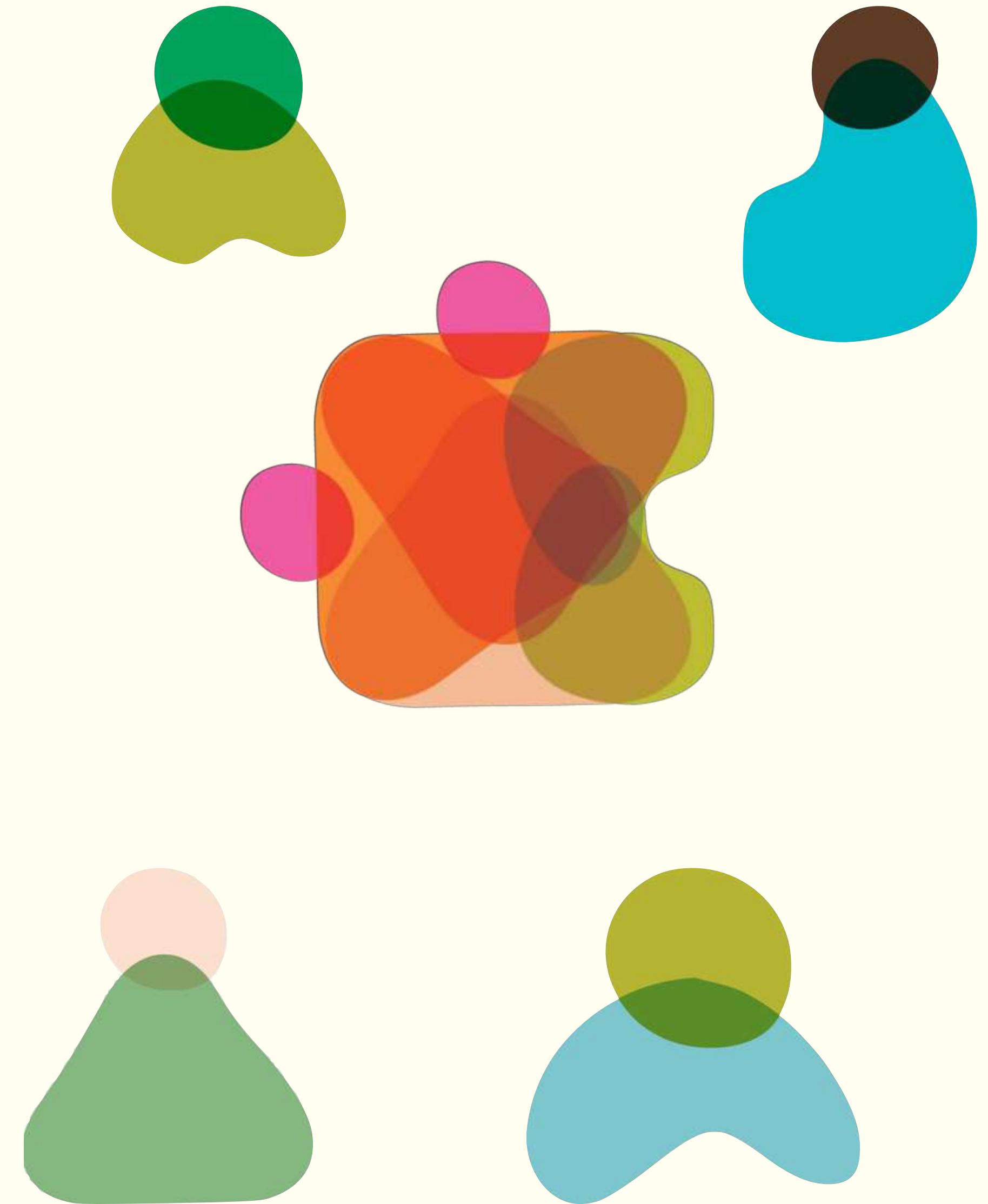
1980



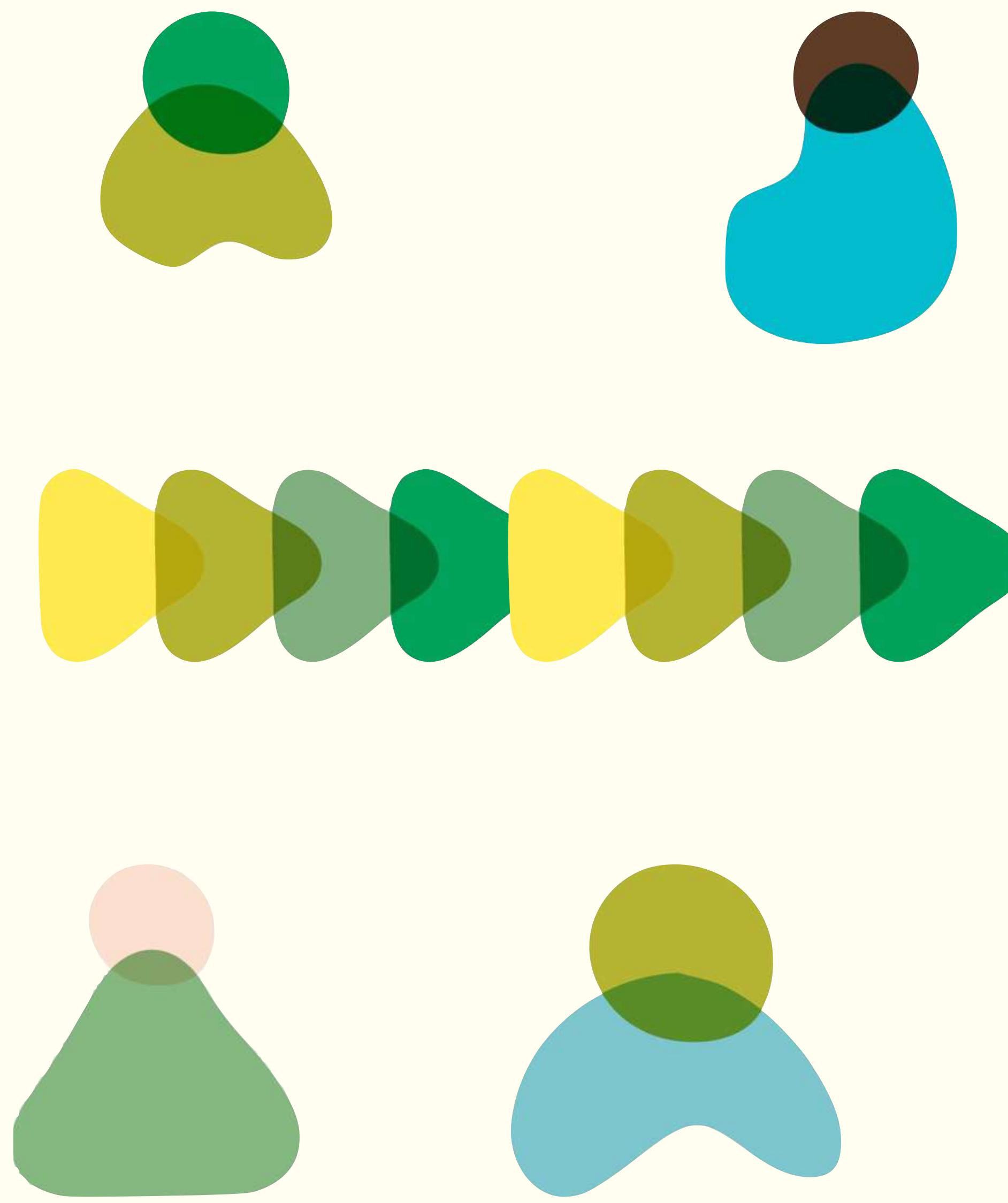
1990



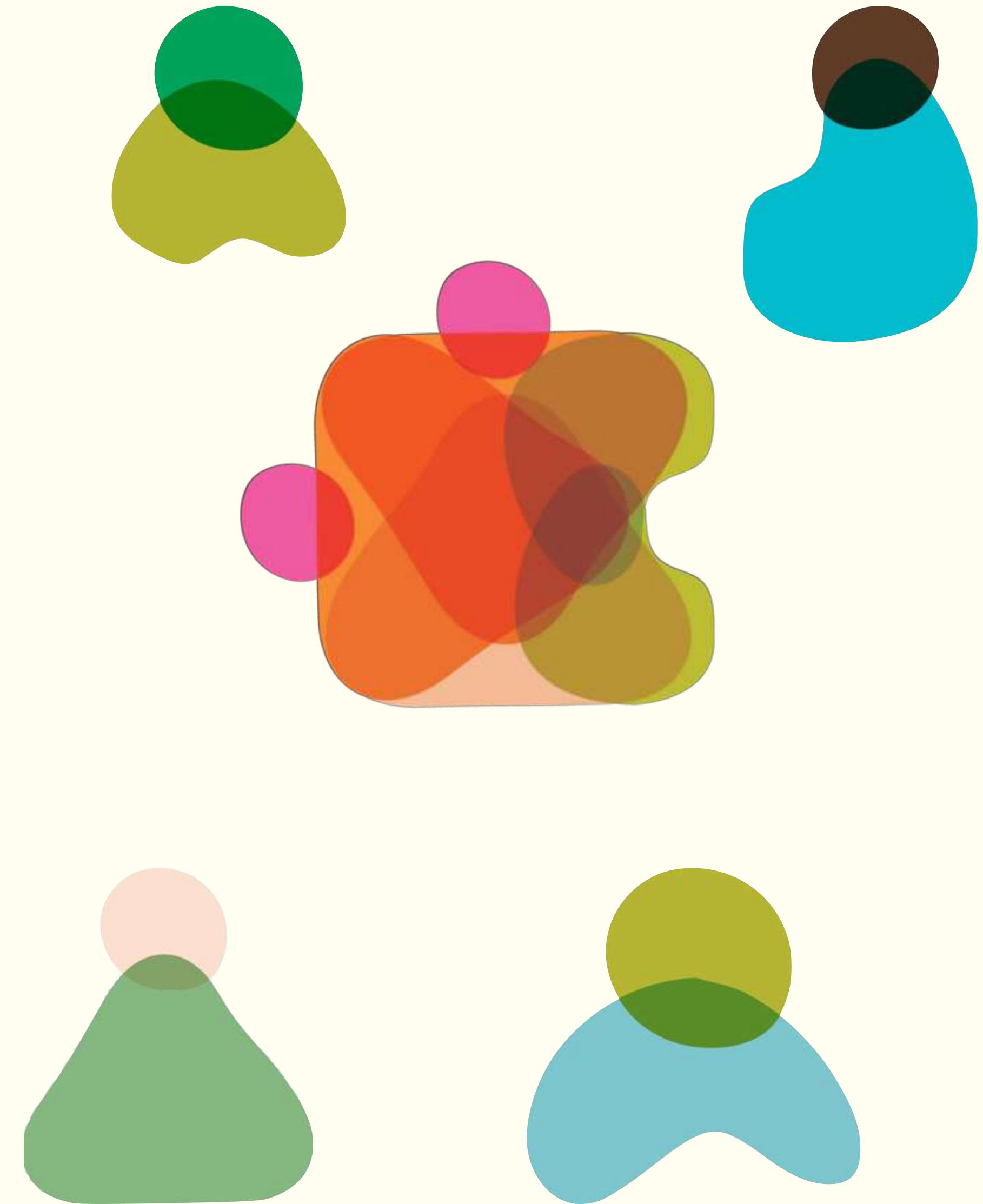
1980



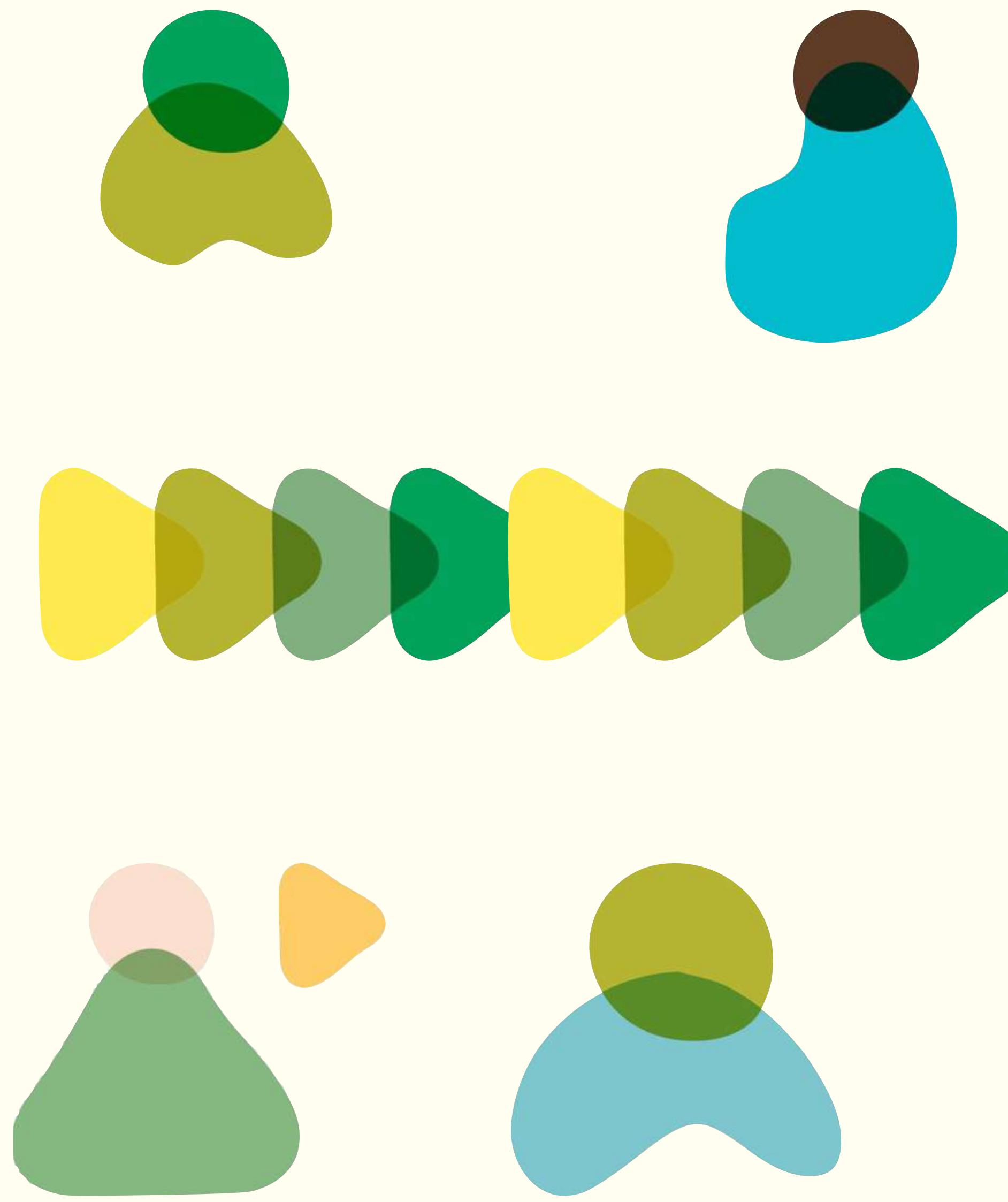
1990



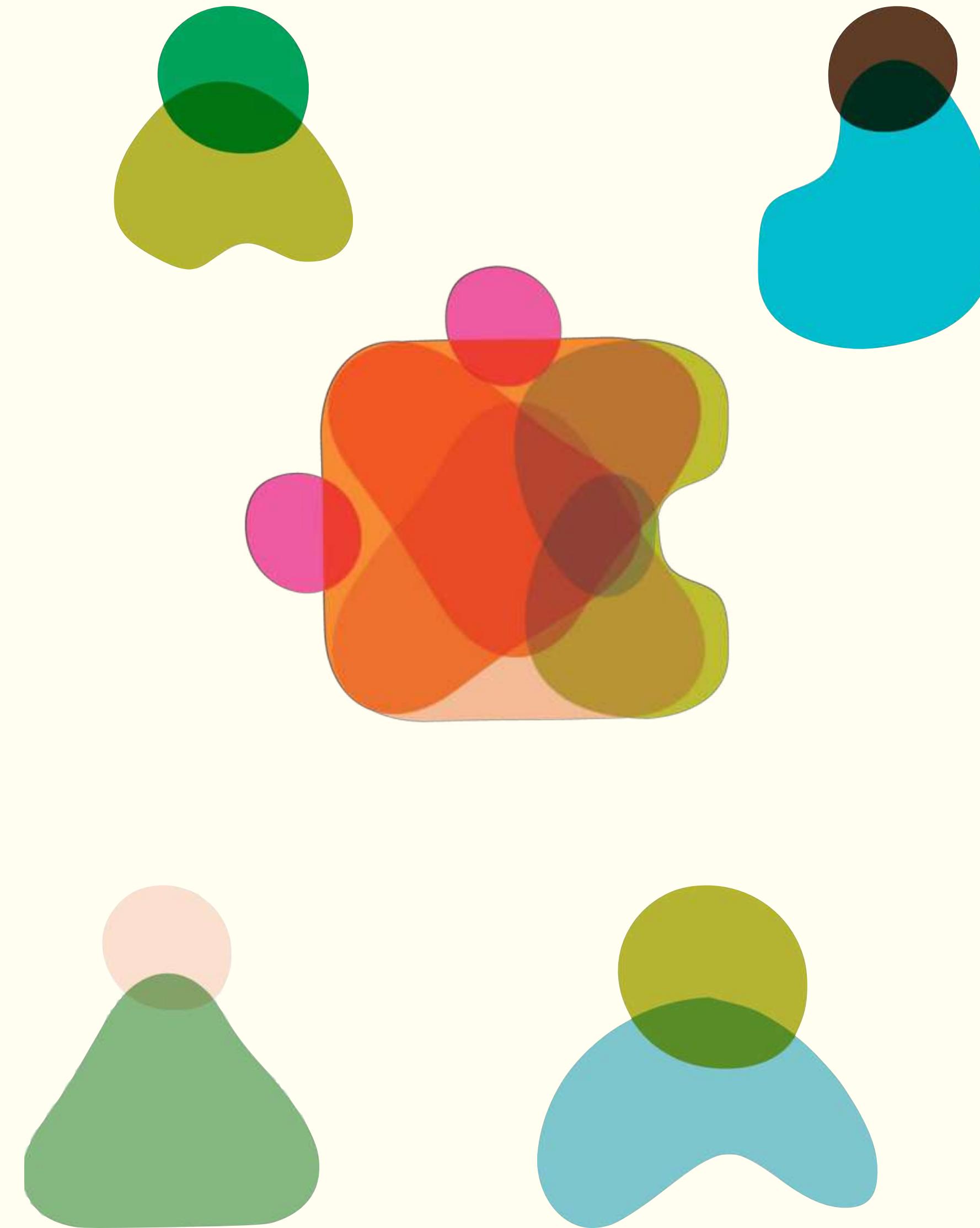
1980



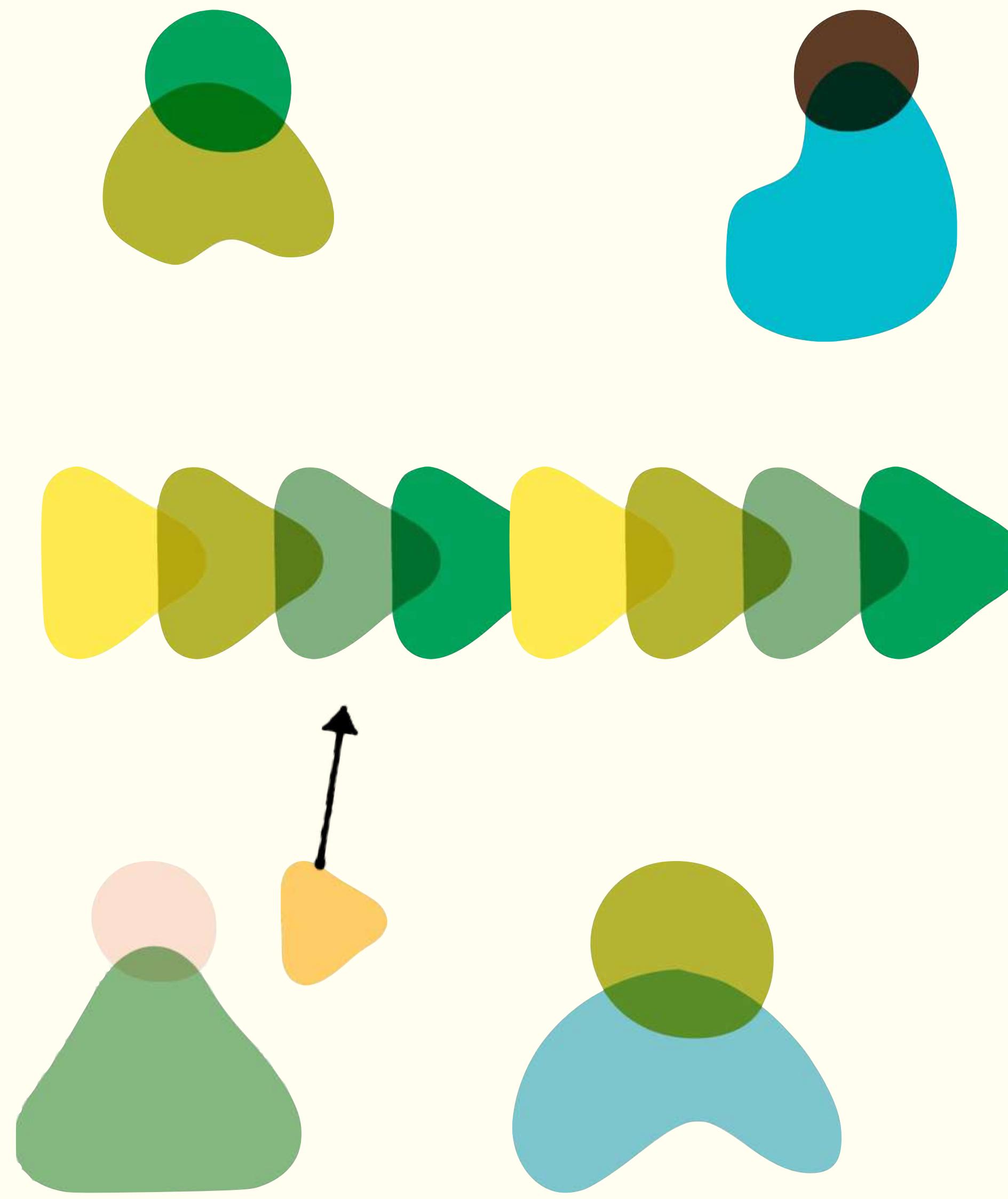
1990



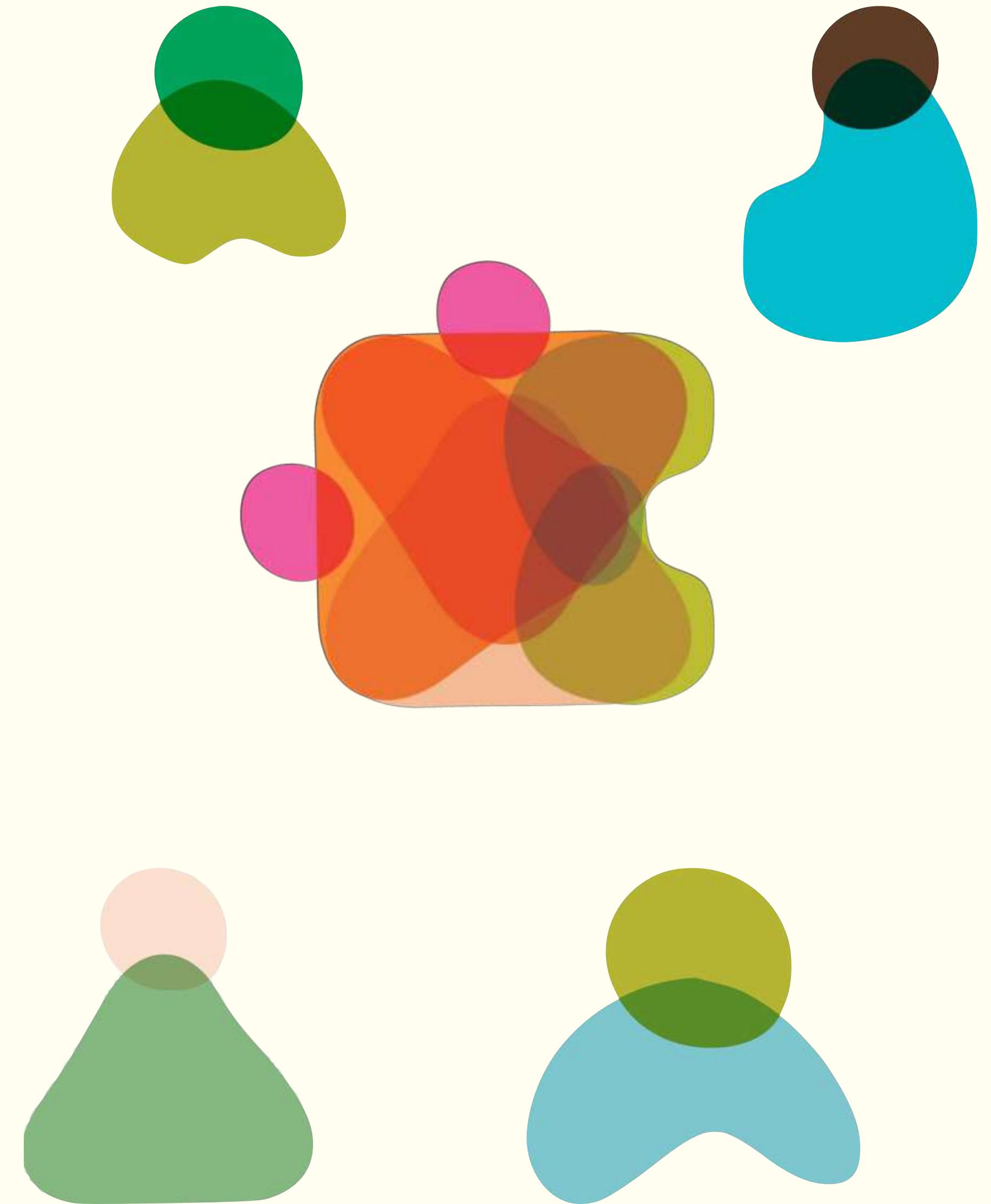
1980



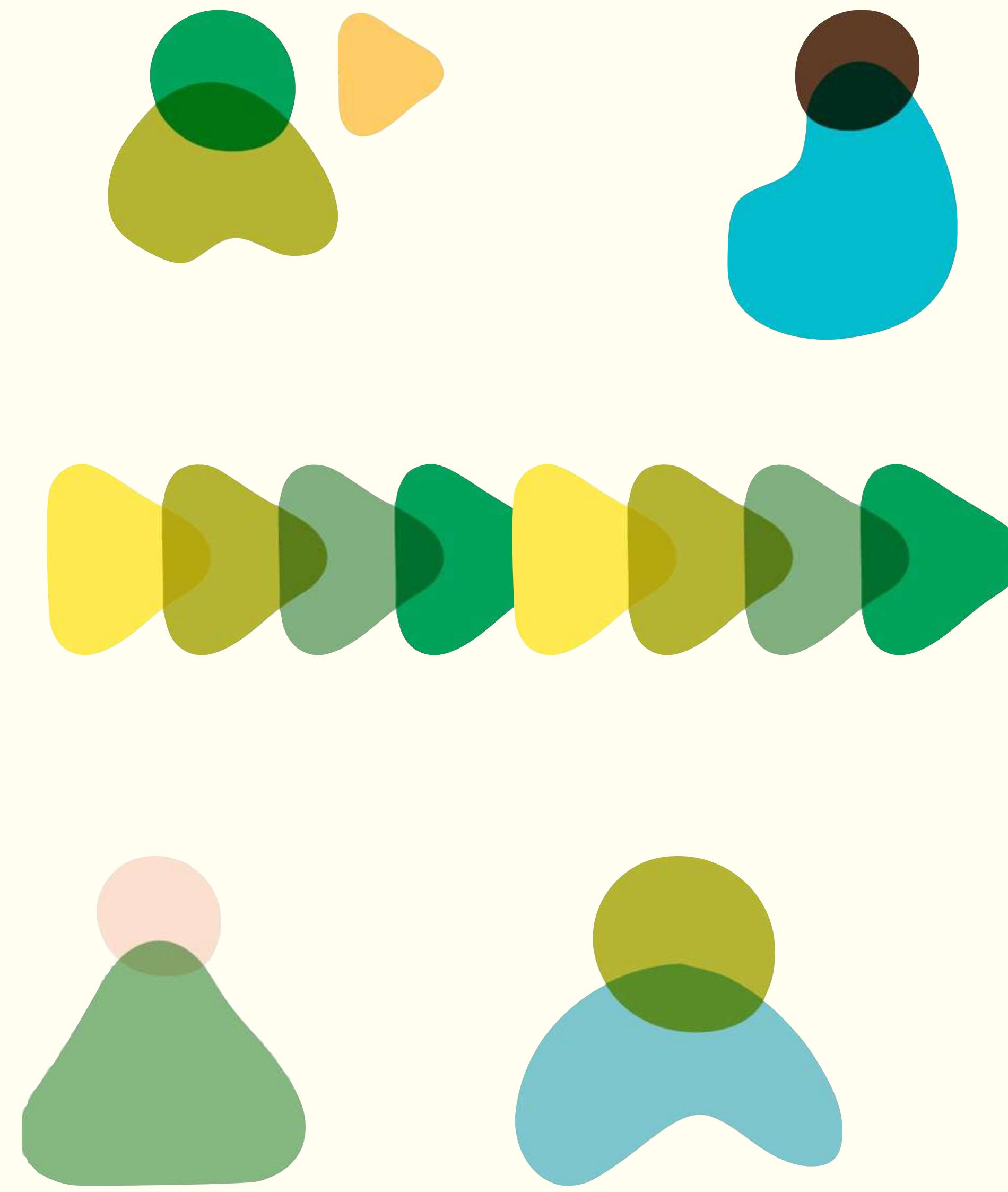
1990



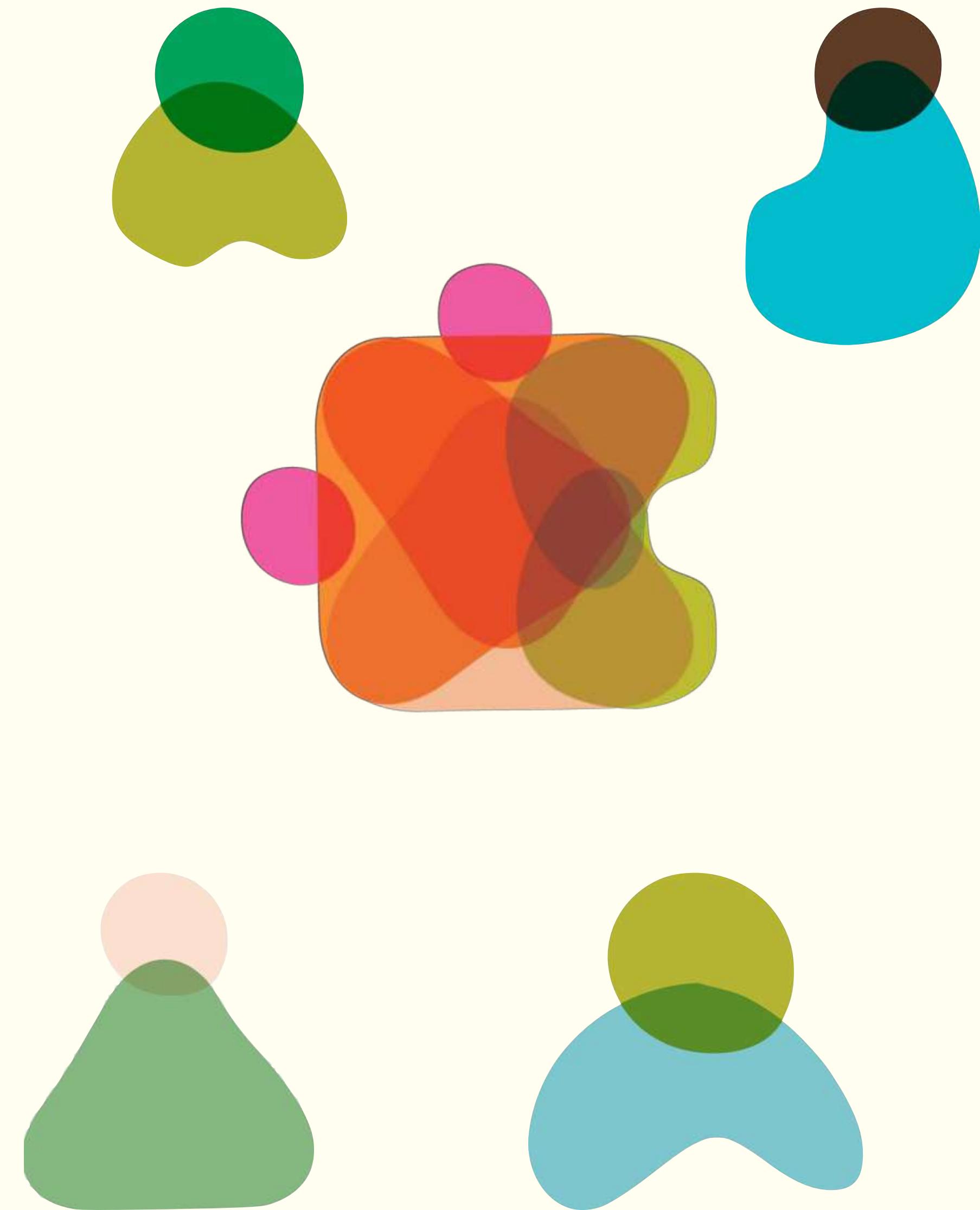
1980



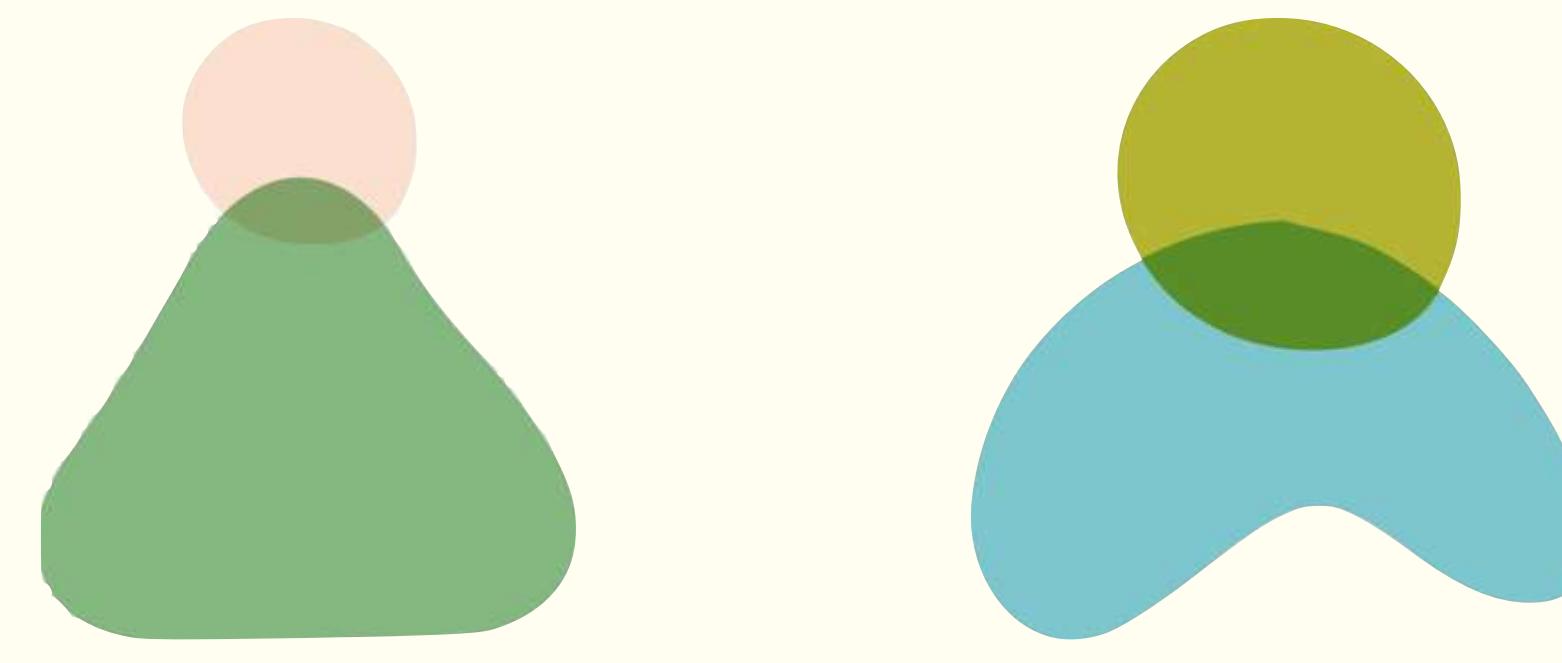
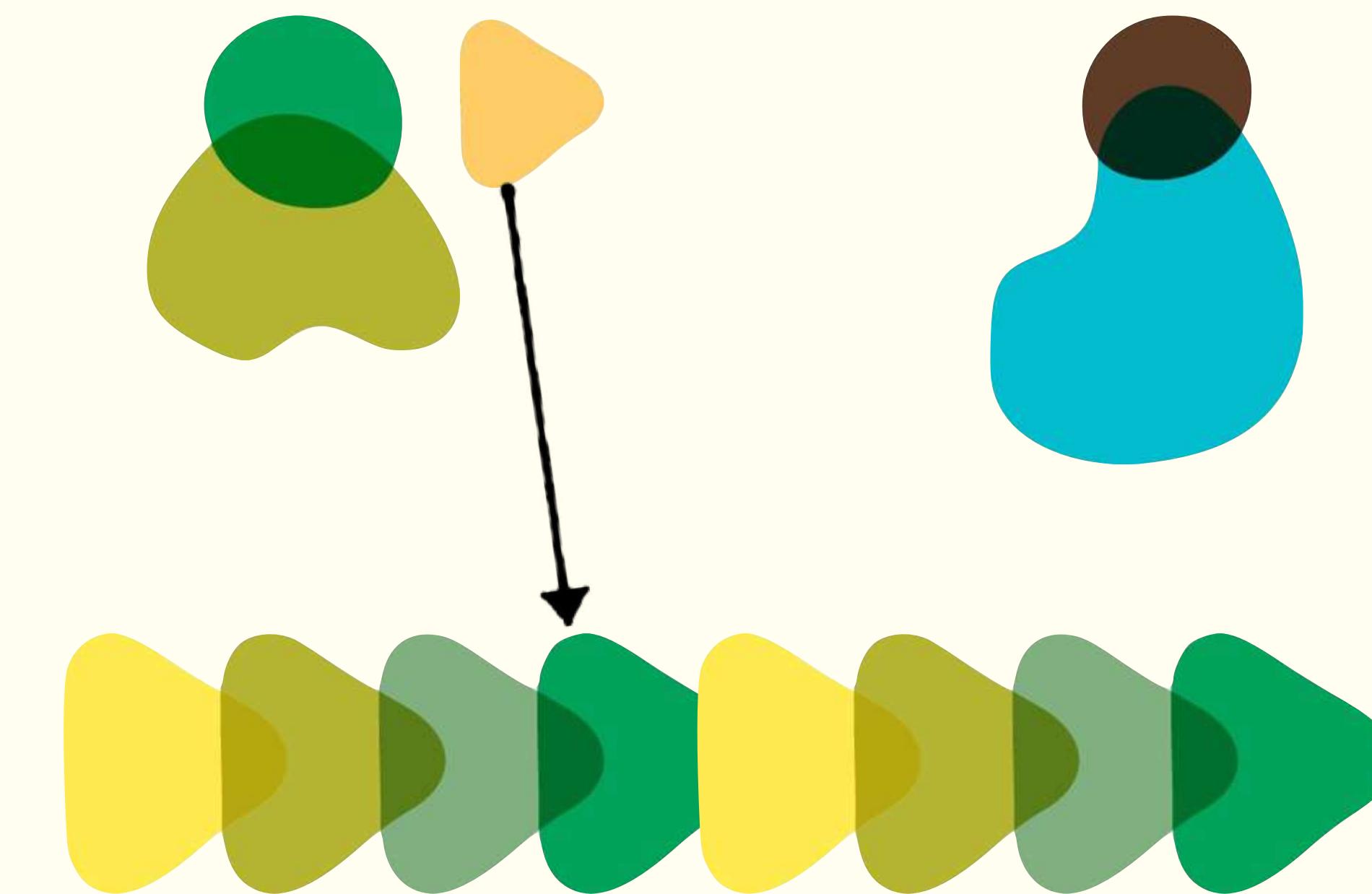
1990



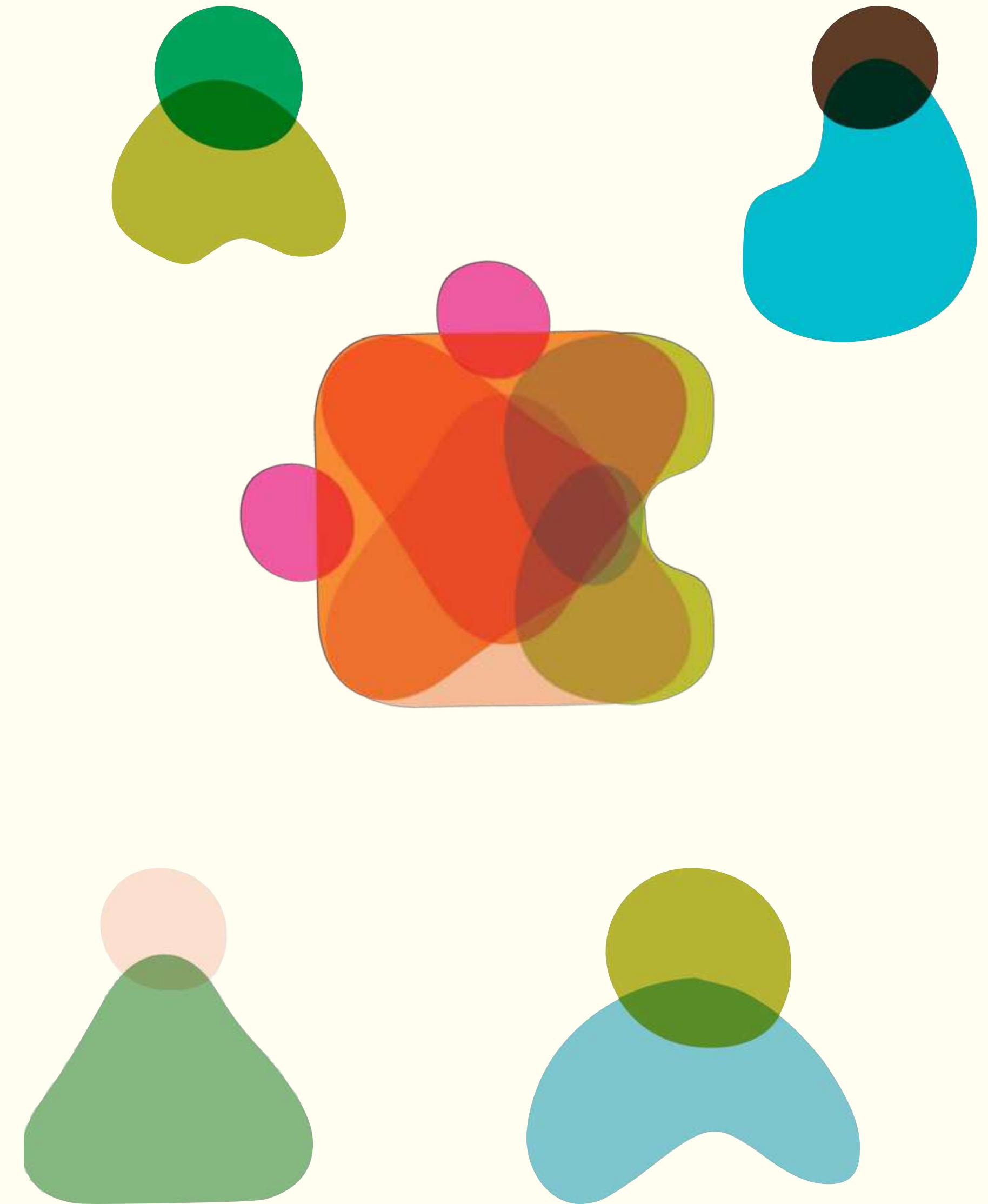
1980



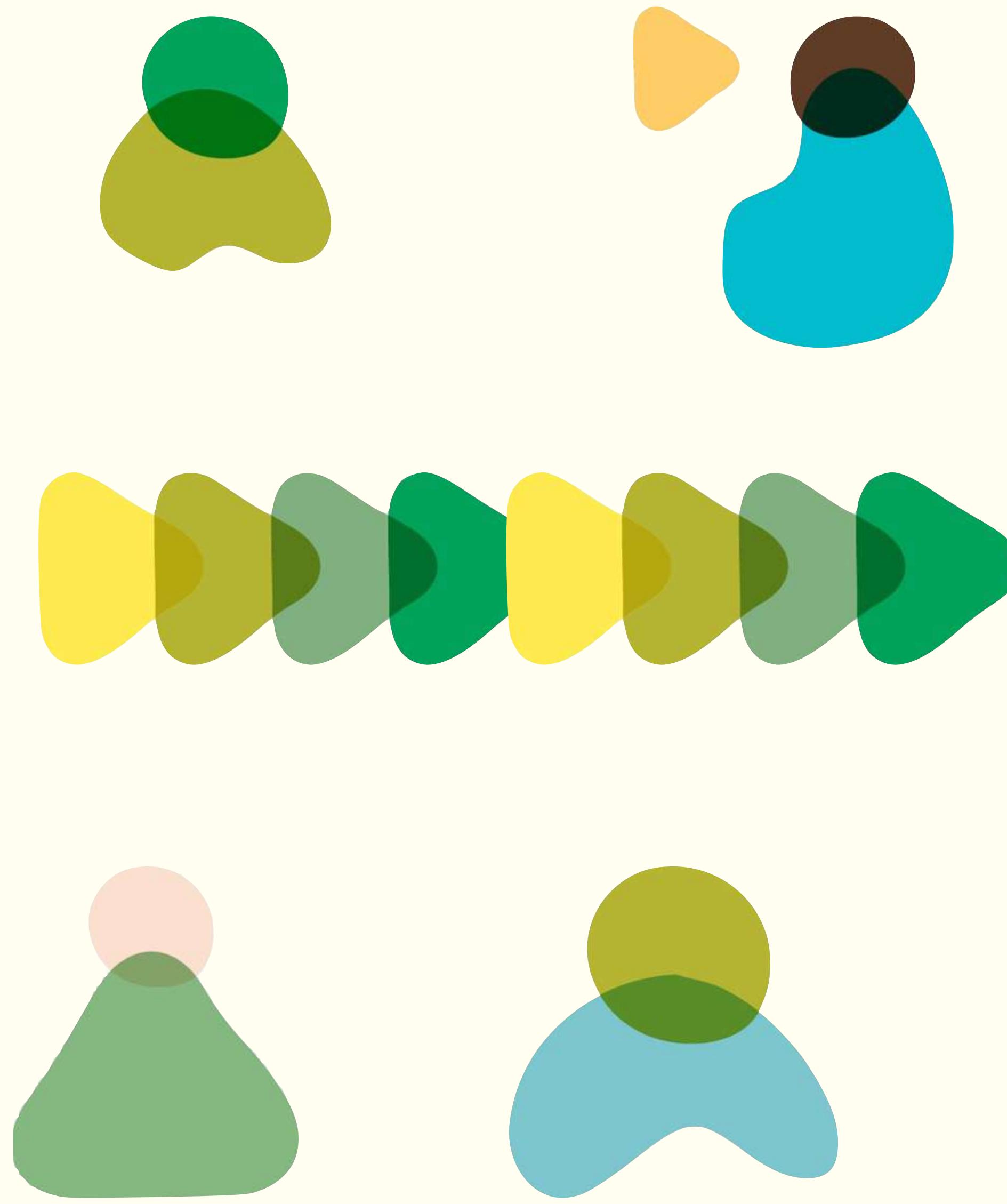
1990



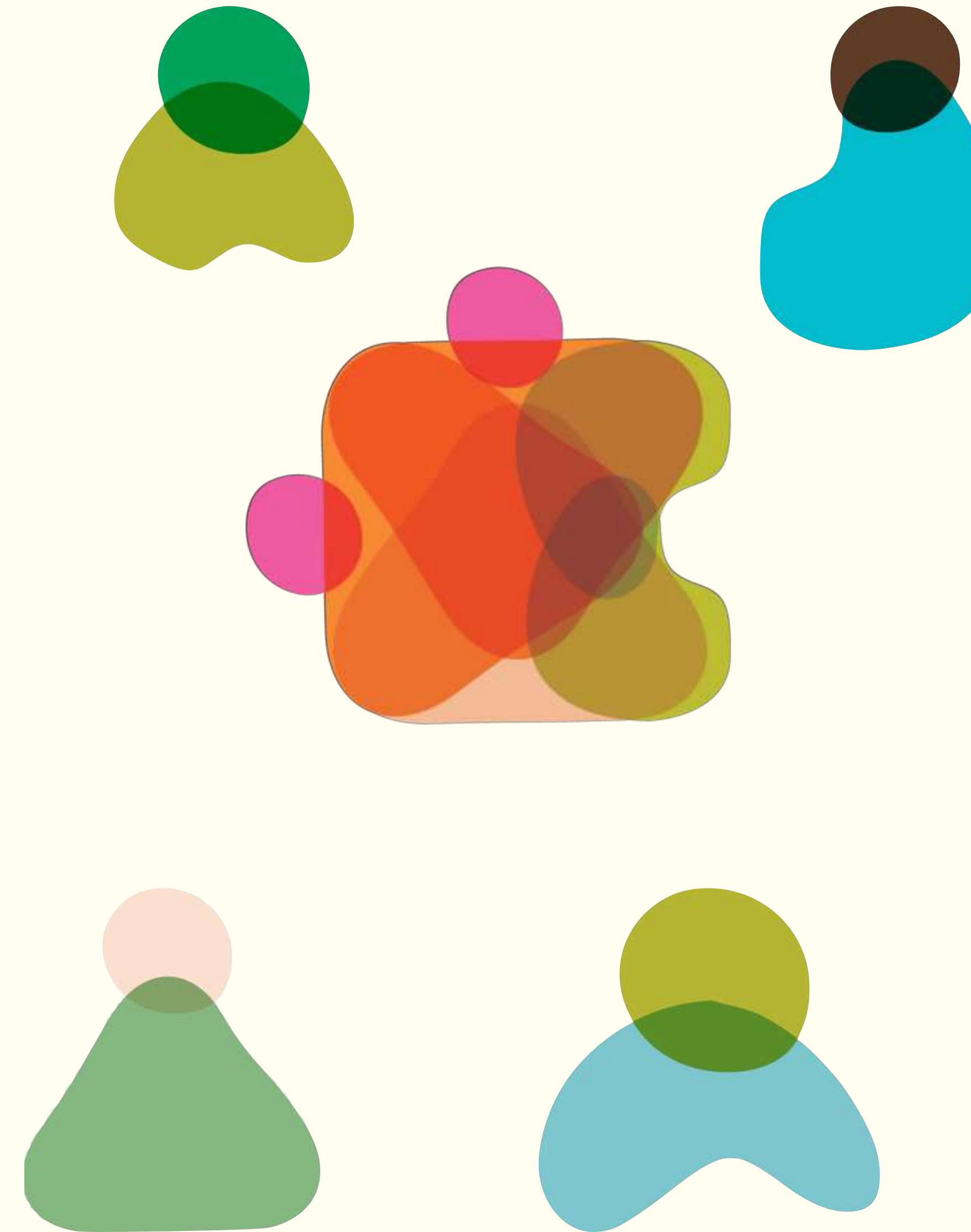
1980



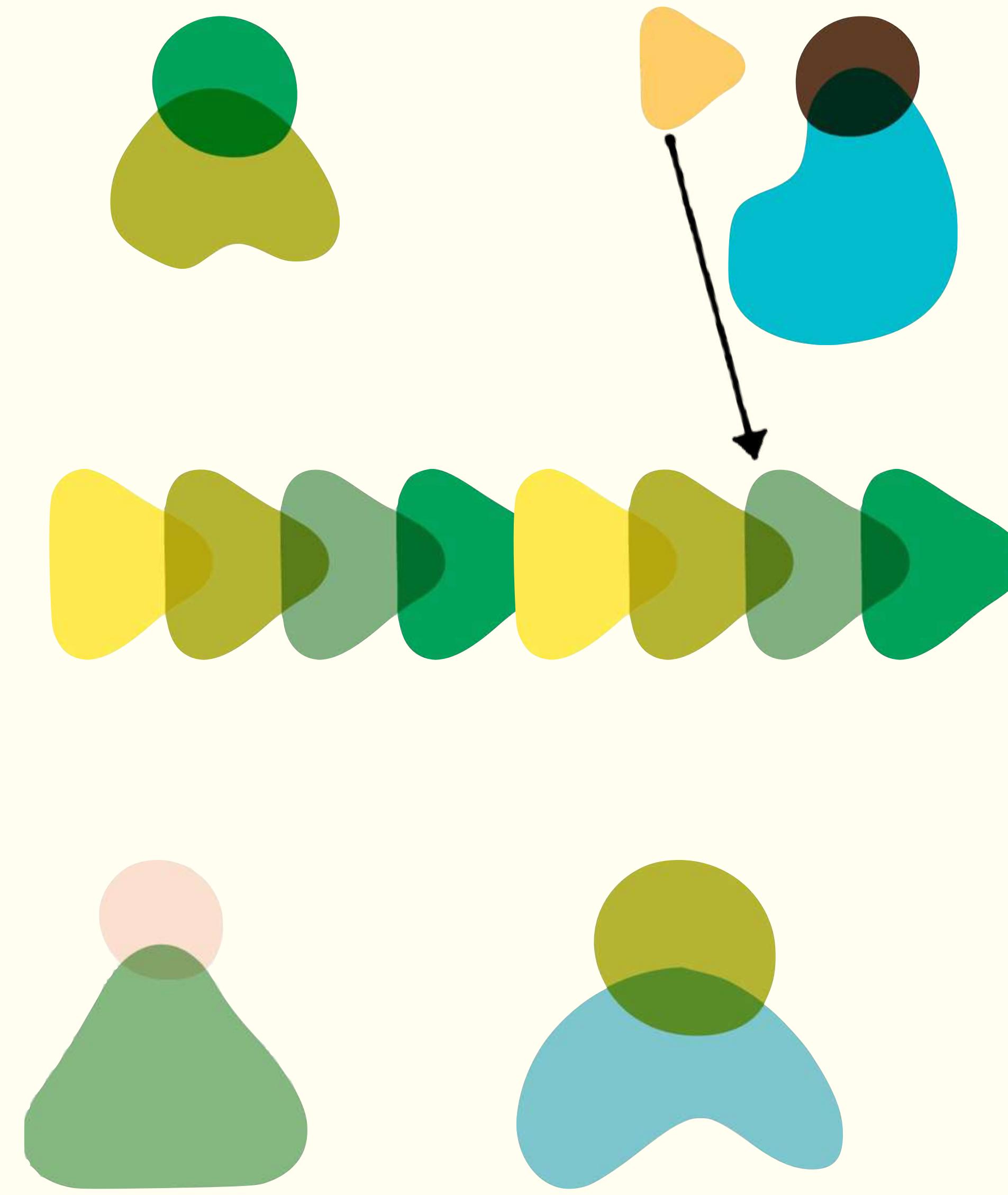
1990



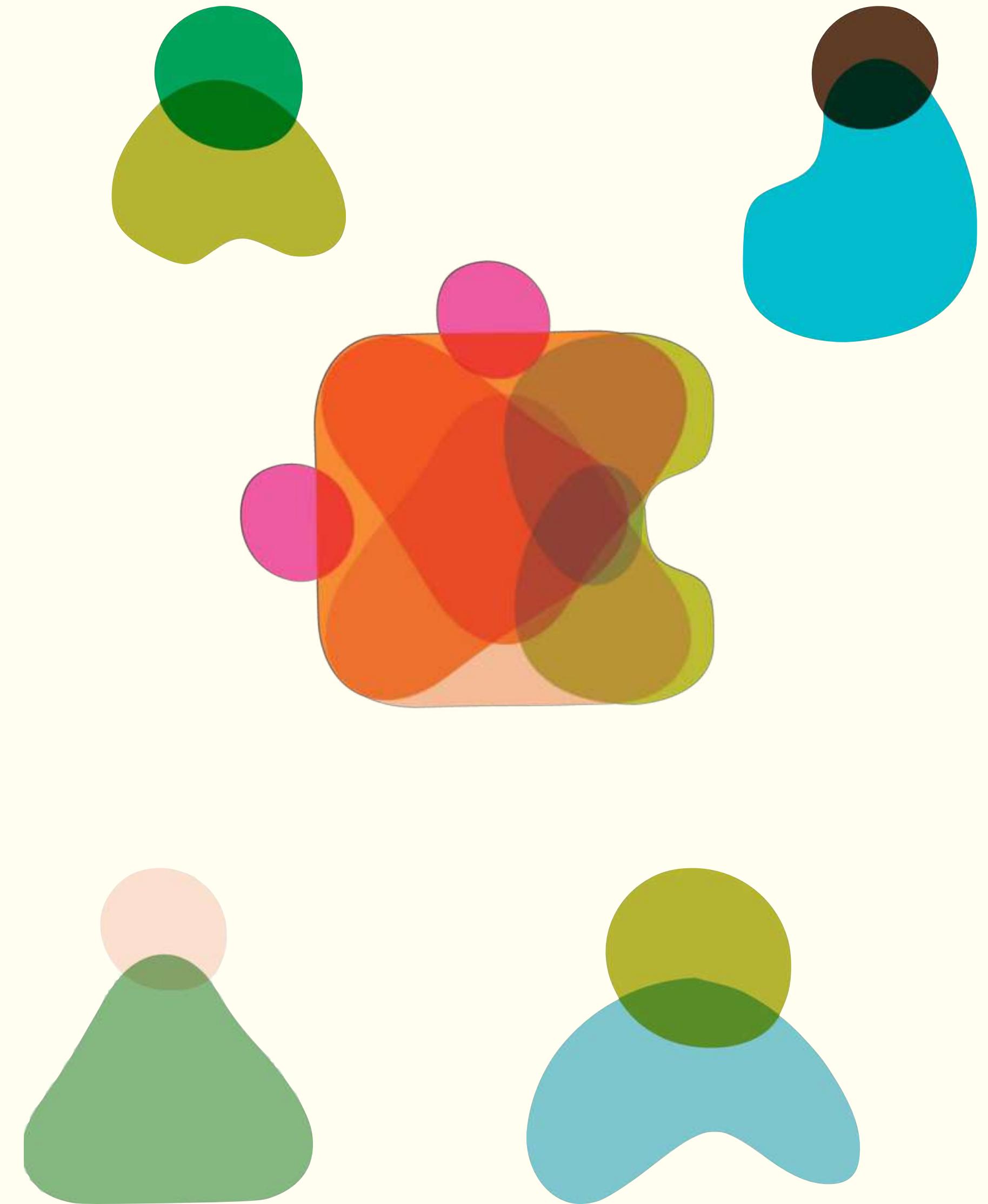
1980



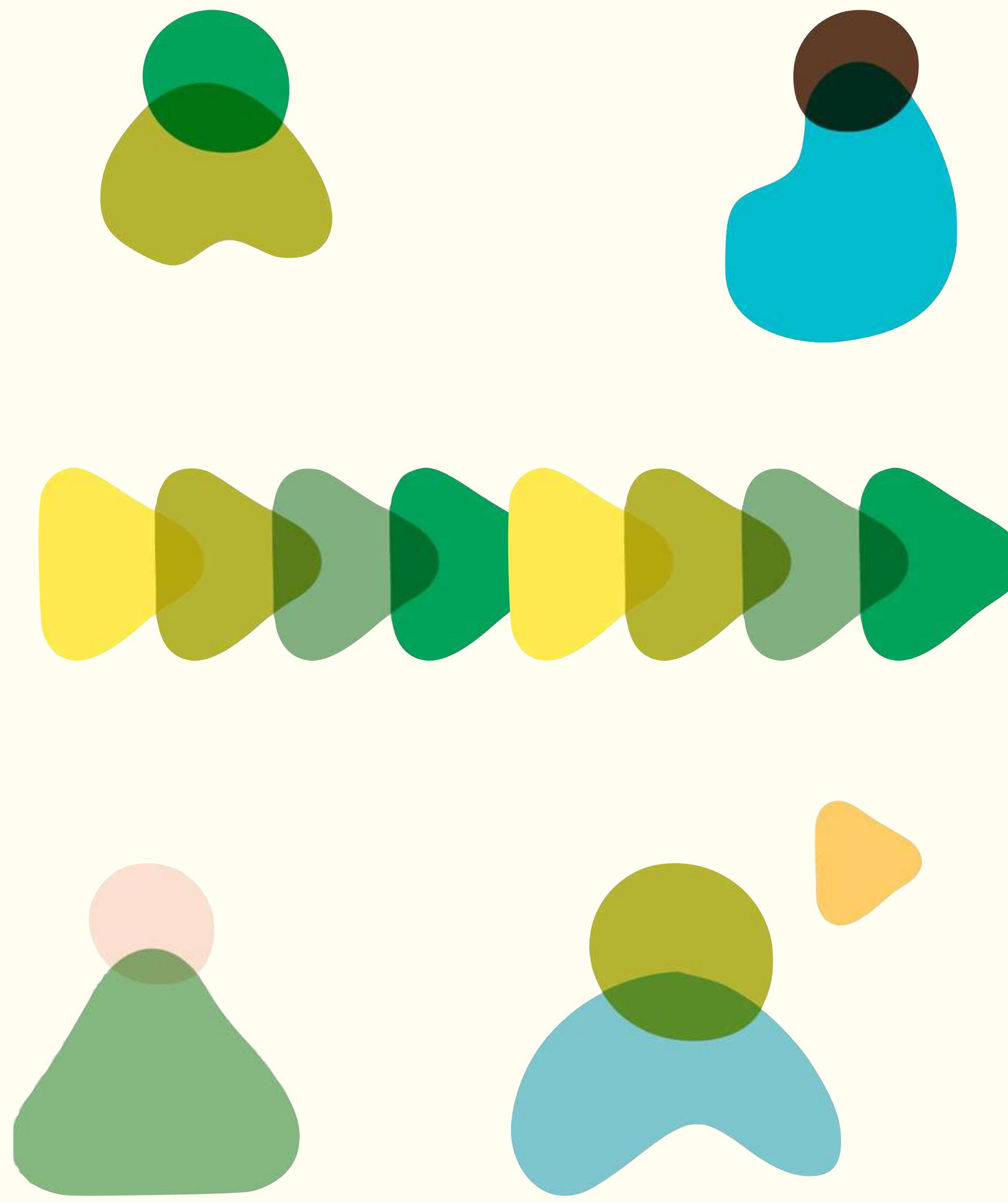
1990



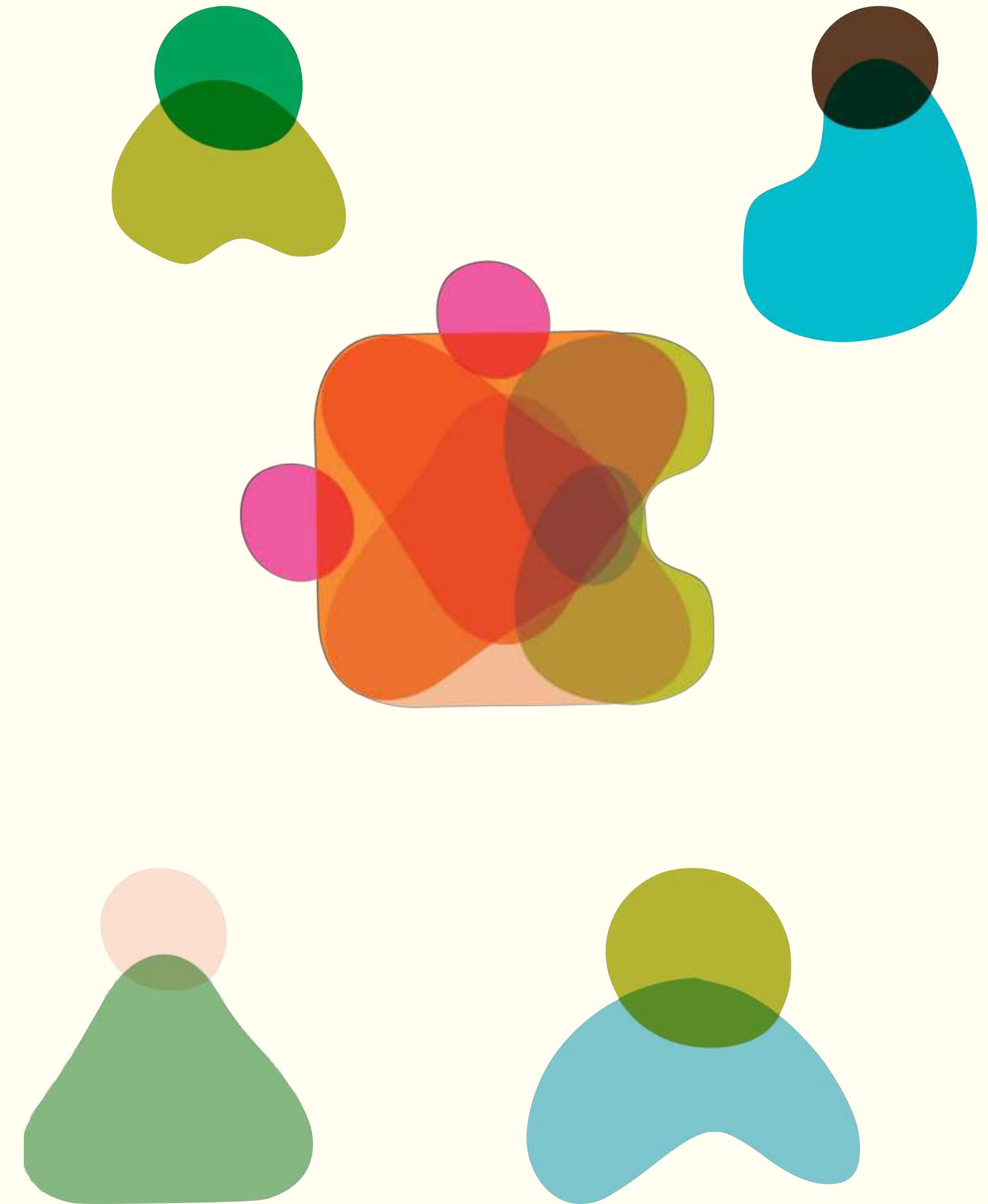
1980



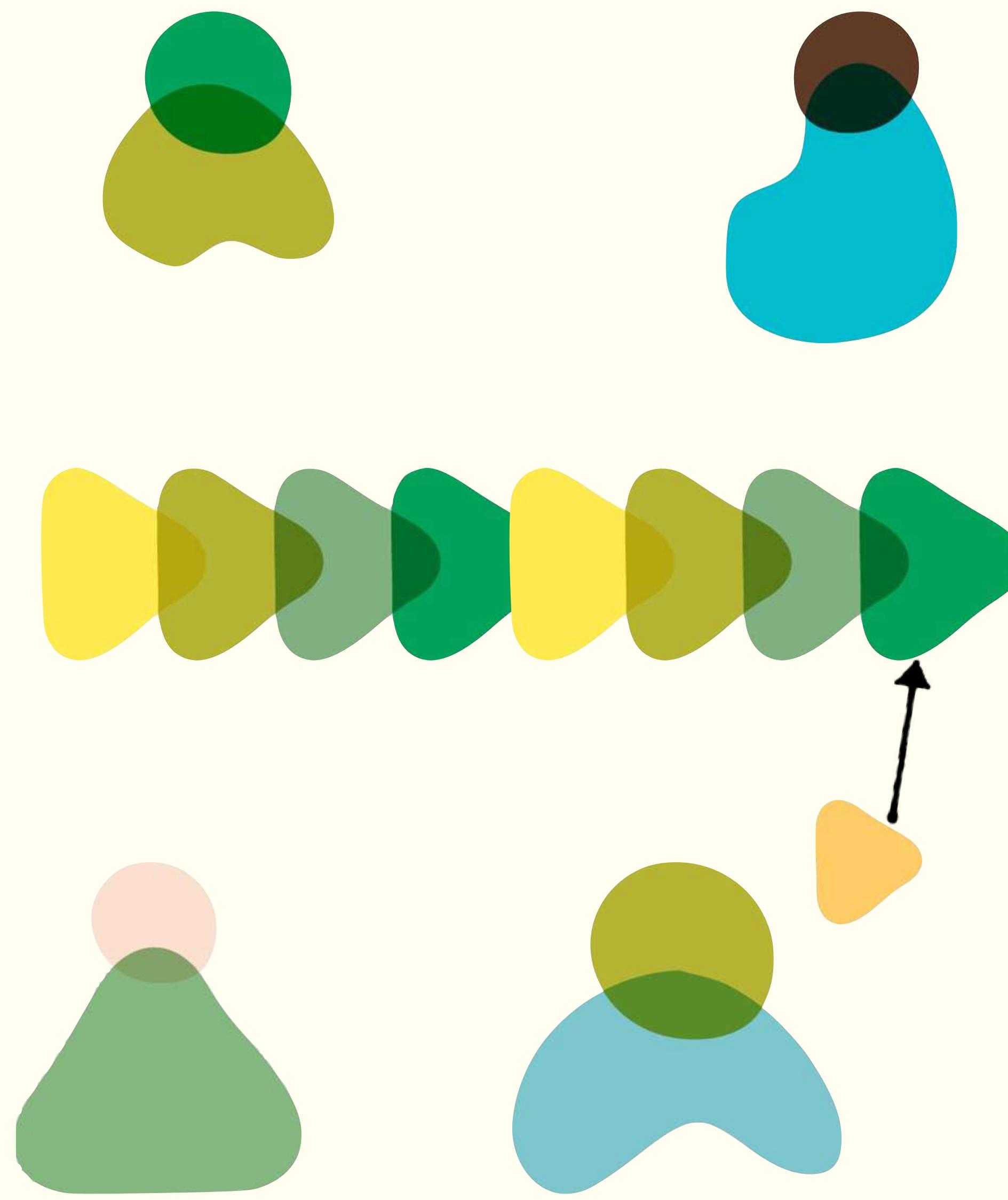
1990

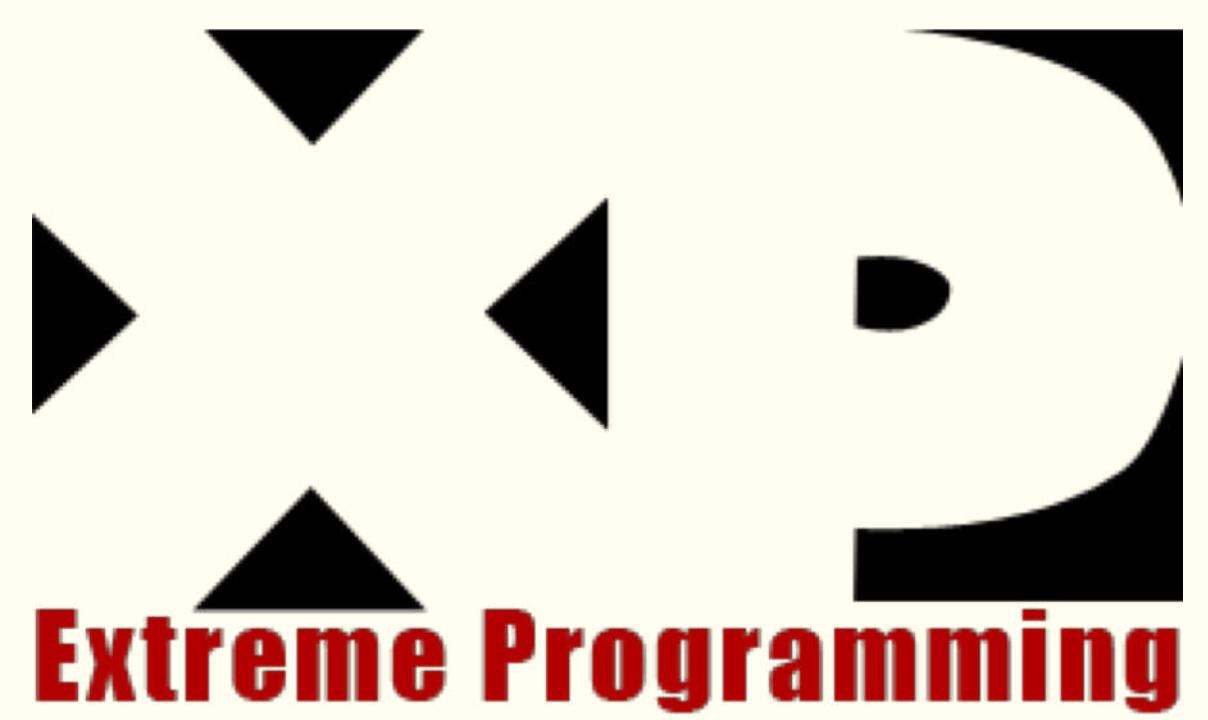


1980

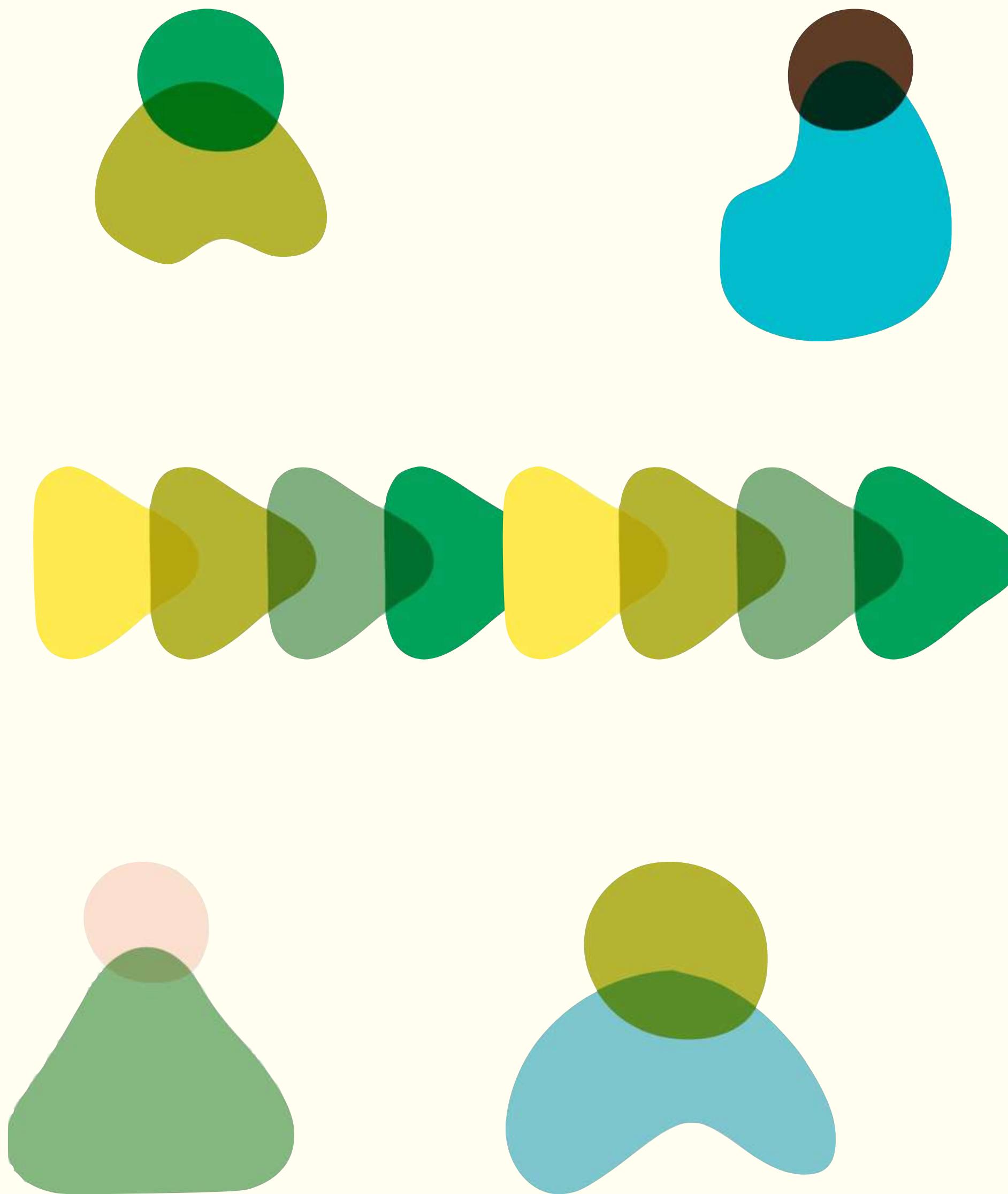


1990



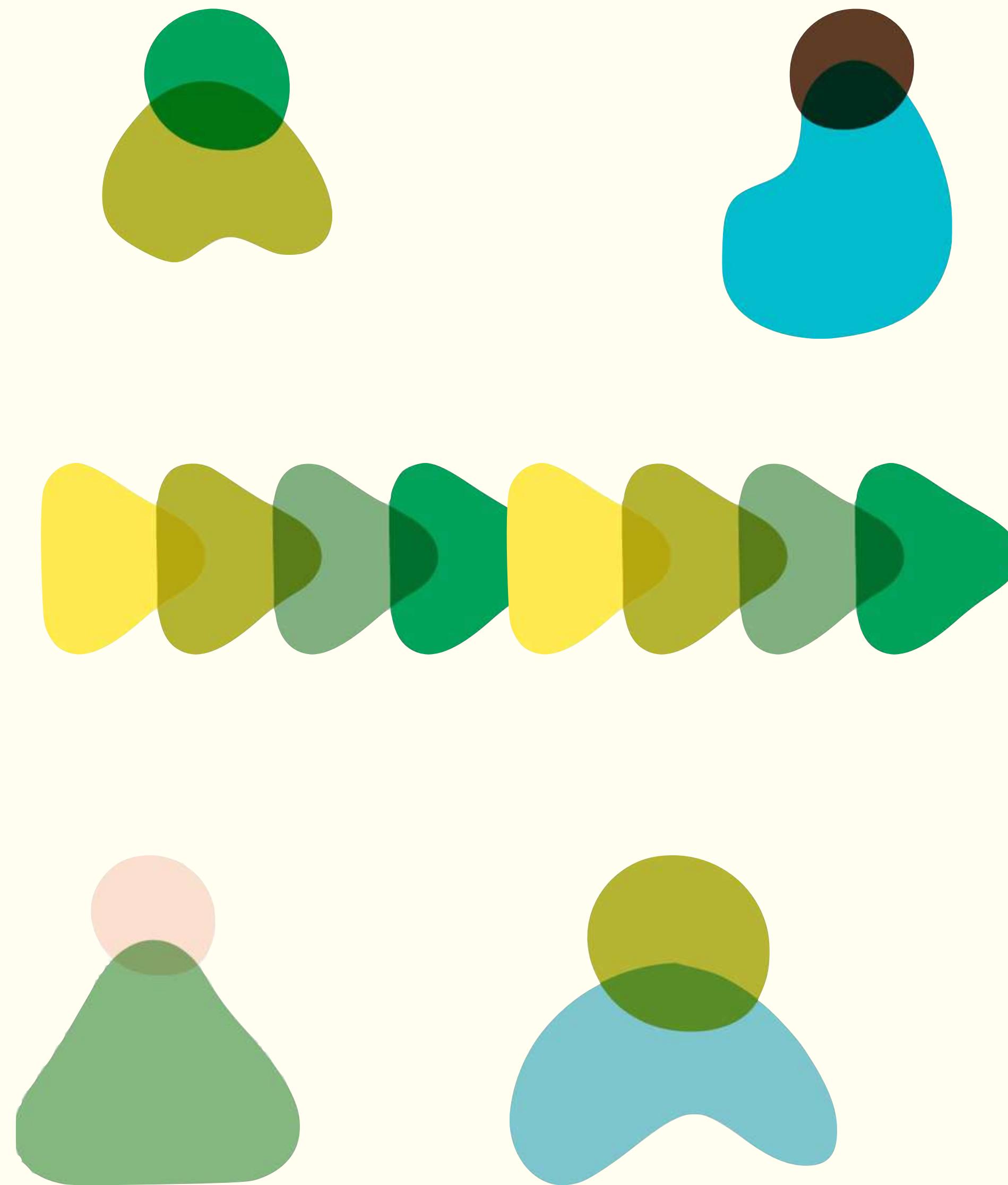


1990



continuous integration:

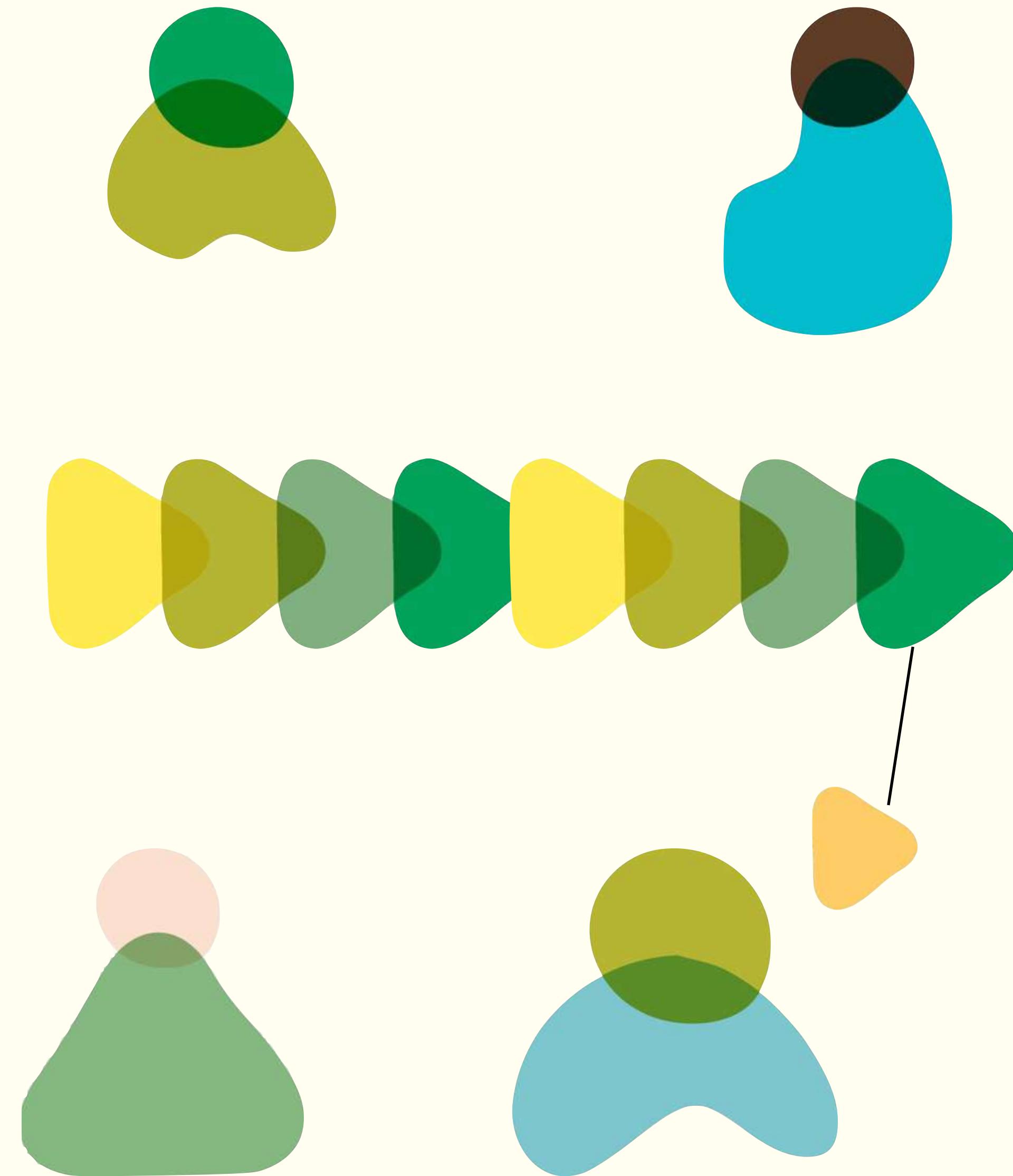
1990



continuous integration:

everyone commits to
trunk at least once per day

1990

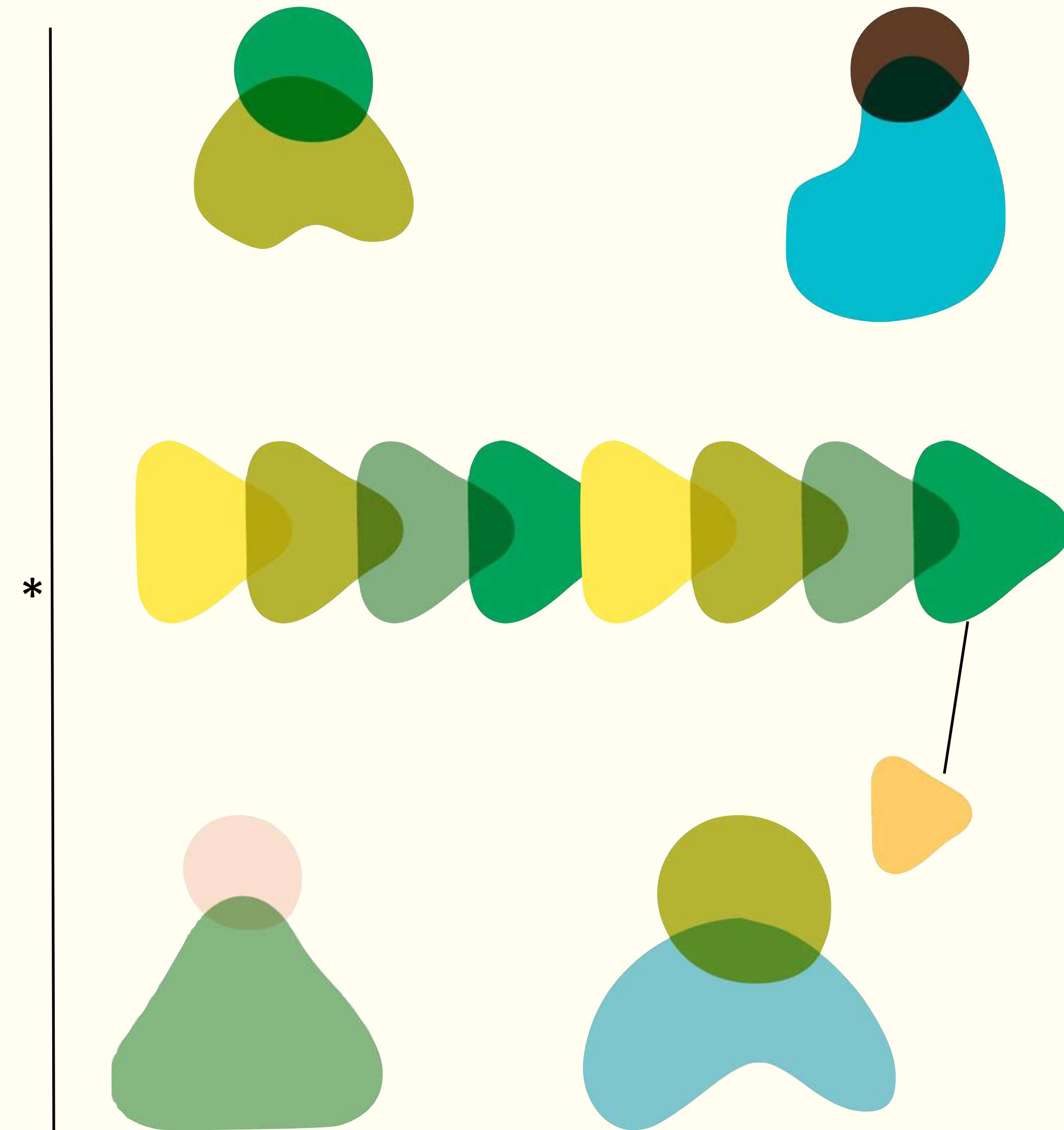


continuous integration:

everyone commits to
trunk at least once per day

*if you do feature branches, you
aren't doing continuous integration

1990



integration as an engineering practice over time

1980

1990

2000

2010

current
day

integration as an engineering practice over time

*continuous
integration*

1980

1990

2000

2010

current
day

integration as an engineering practice over time

*continuous
integration*

1980

1990

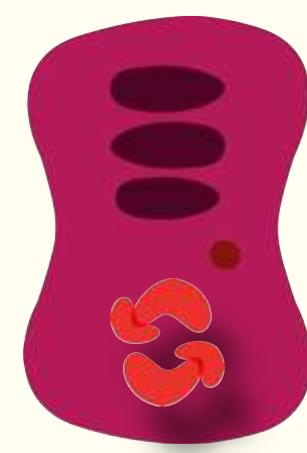
2000

2010

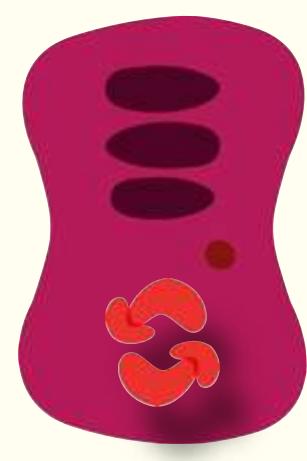
current
day



*continuous
integration*



*continuous
integration*



canonical integration point

*continuous
integration*



canonical integration point

useful platform to do other *stuff*

*continuous
integration*



canonical integration point

useful platform to do other *stuff*

unit testing

*continuous
integration*



canonical integration point

useful platform to do other *stuff*

unit testing

functional testing

*continuous
integration*



canonical integration point

useful platform to do other *stuff*

unit testing

functional testing

integration testing

*continuous
integration*



canonical integration point

useful platform to do other *stuff*

unit testing

functional testing

integration testing

automated machine provisioning

*continuous
integration*



canonical integration point

useful platform to do other *stuff*

unit testing

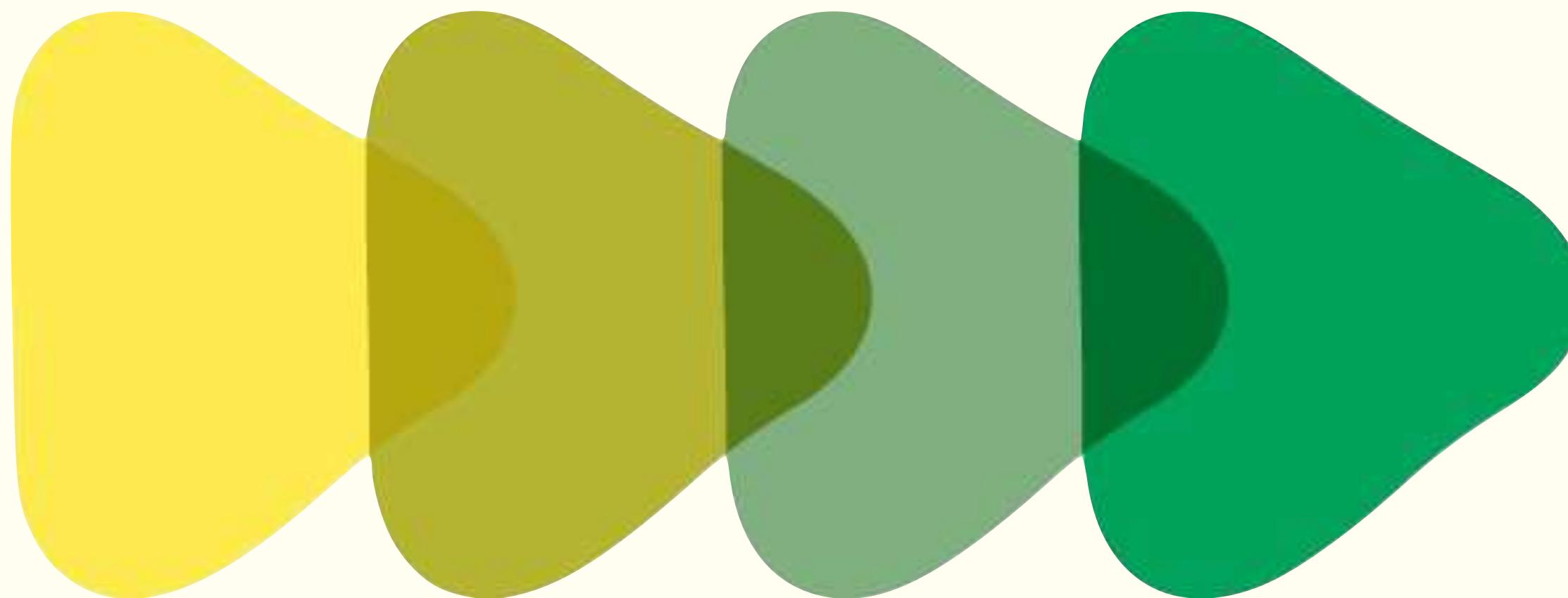
functional testing

integration testing

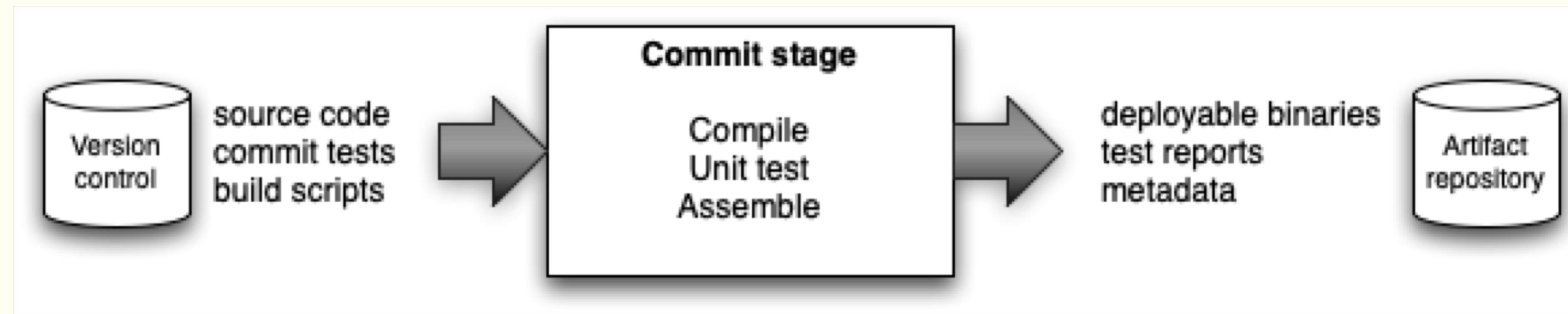
automated machine provisioning

deployment to production?

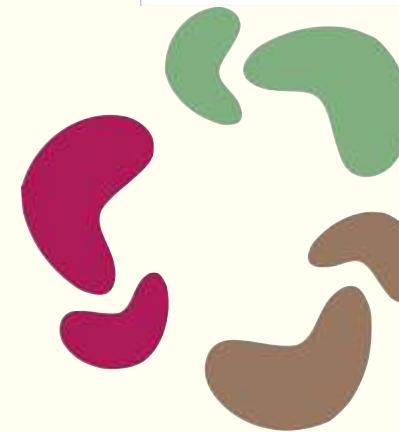
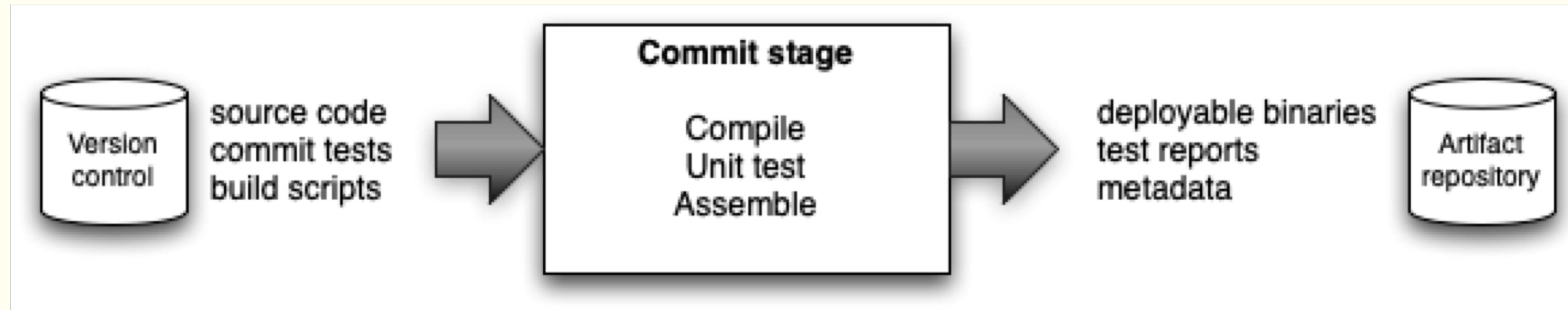
Deployment Pipelines



commit Stage

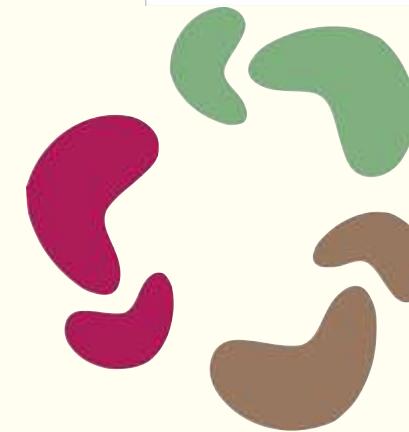
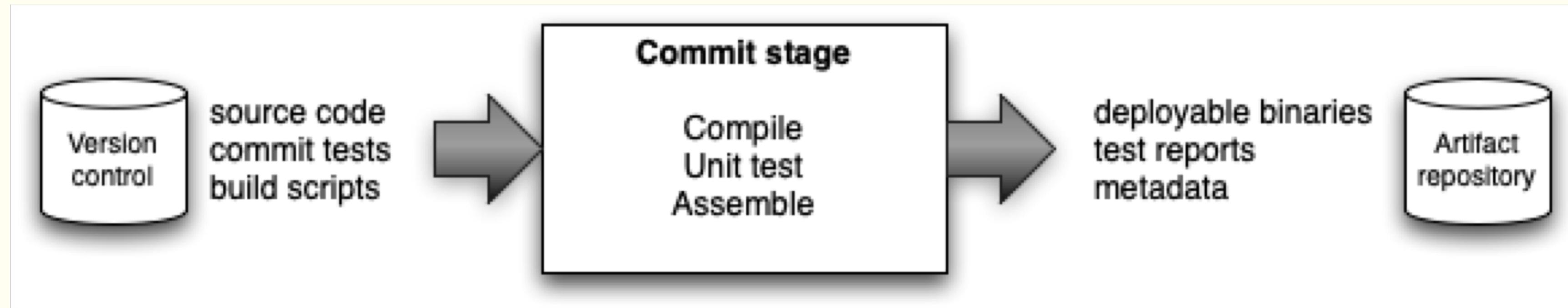


commit Stage



Run against each check-in

commit Stage

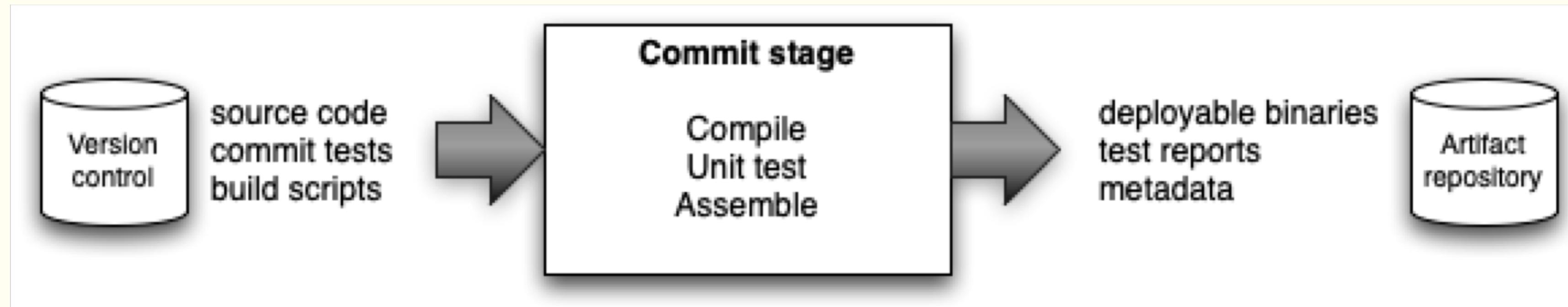


Run against each check-in



Starts building a release candidate

commit Stage



Run against each check-in

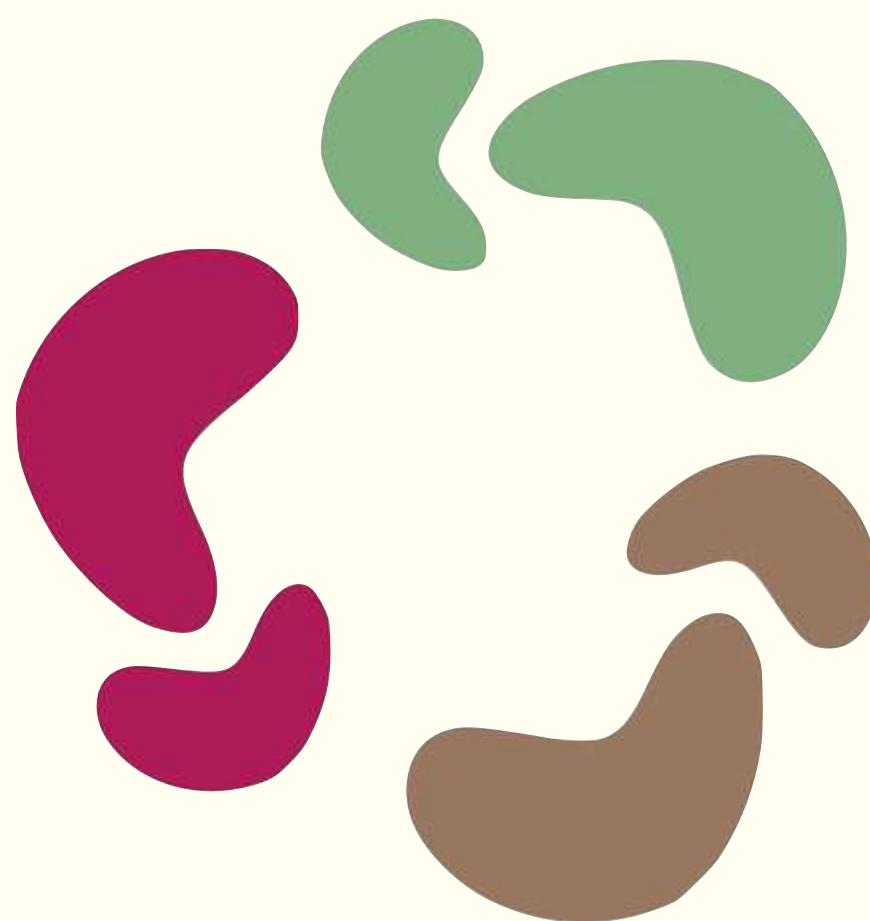
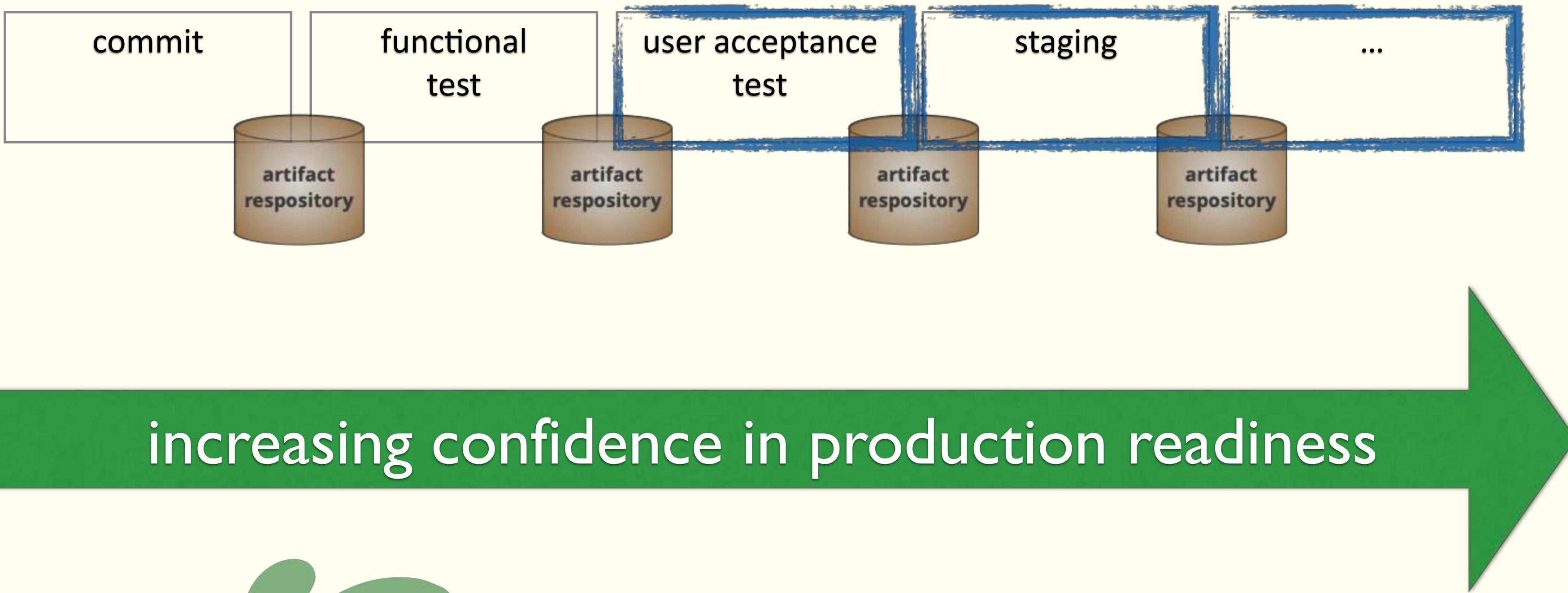


Starts building a release candidate

If it fails, fix it immediately

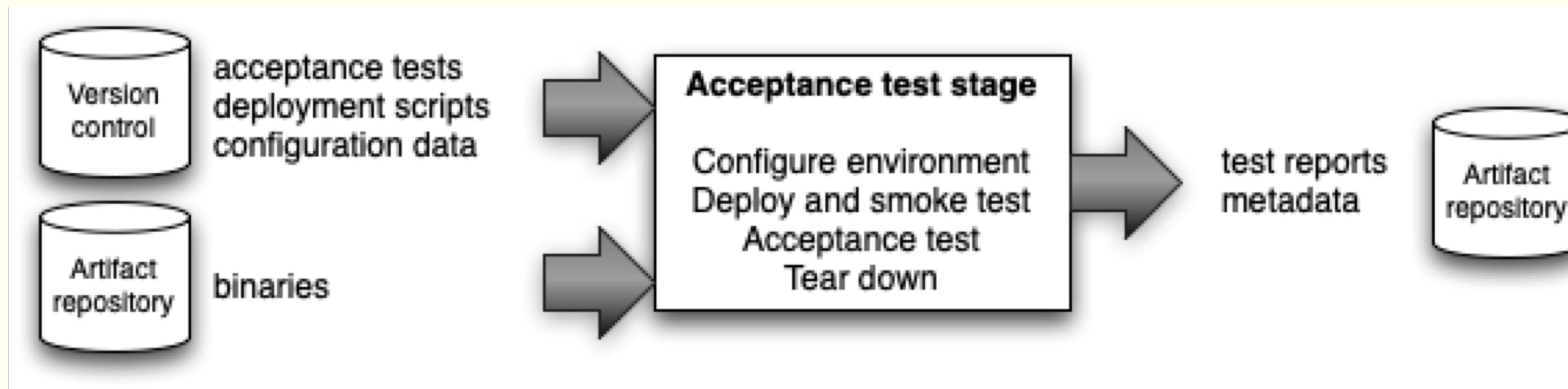


Pipeline Construction

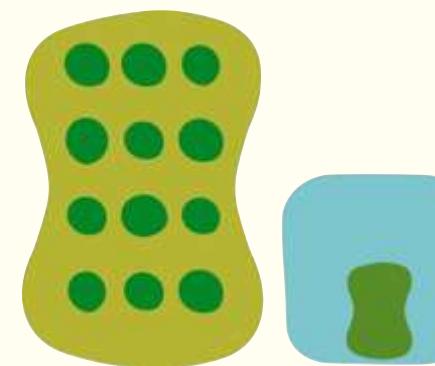
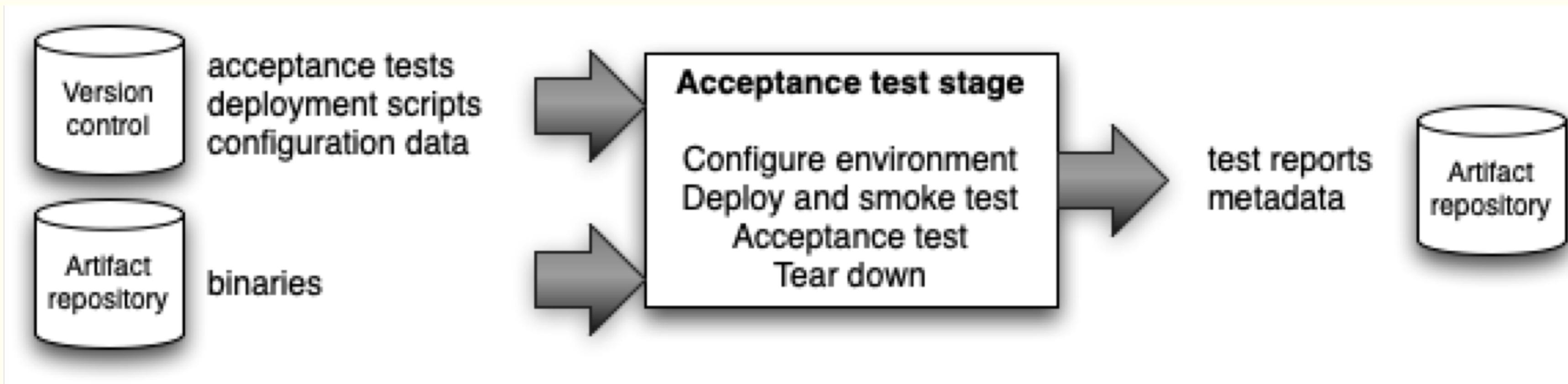


Pipeline stages = feedback opportunities

UAT Stage

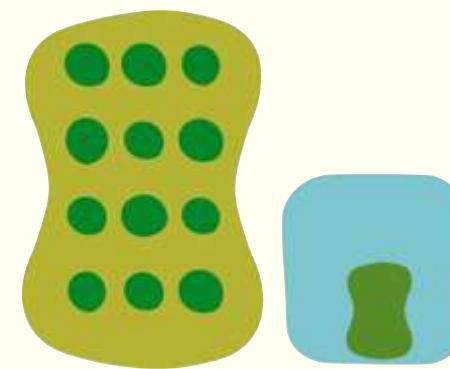
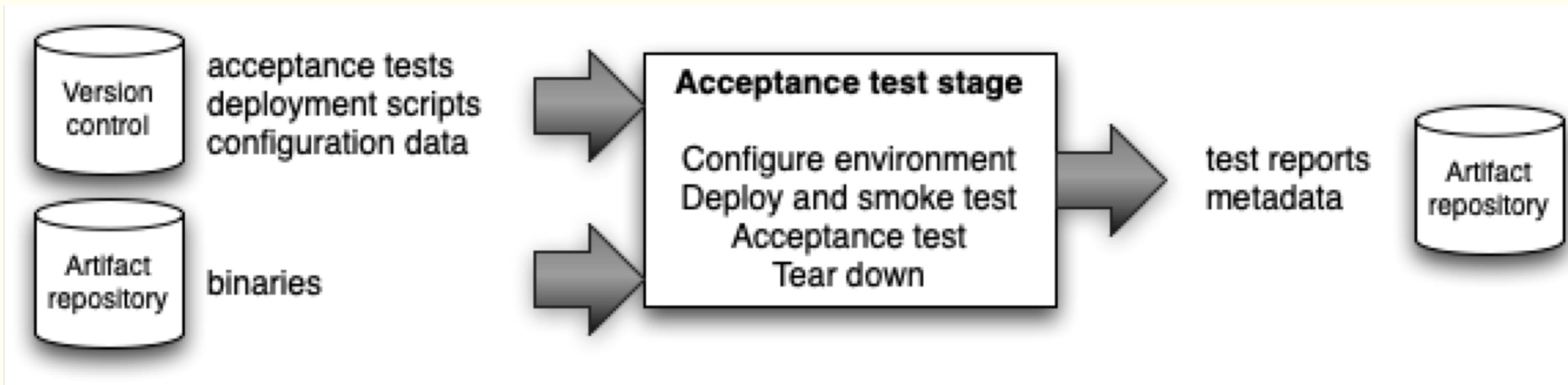


UAT Stage

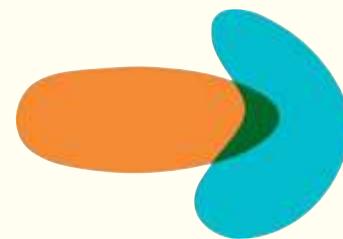


End-to-end tests in production-like environment

UAT Stage

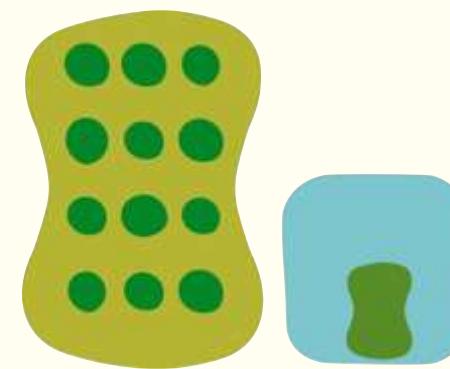
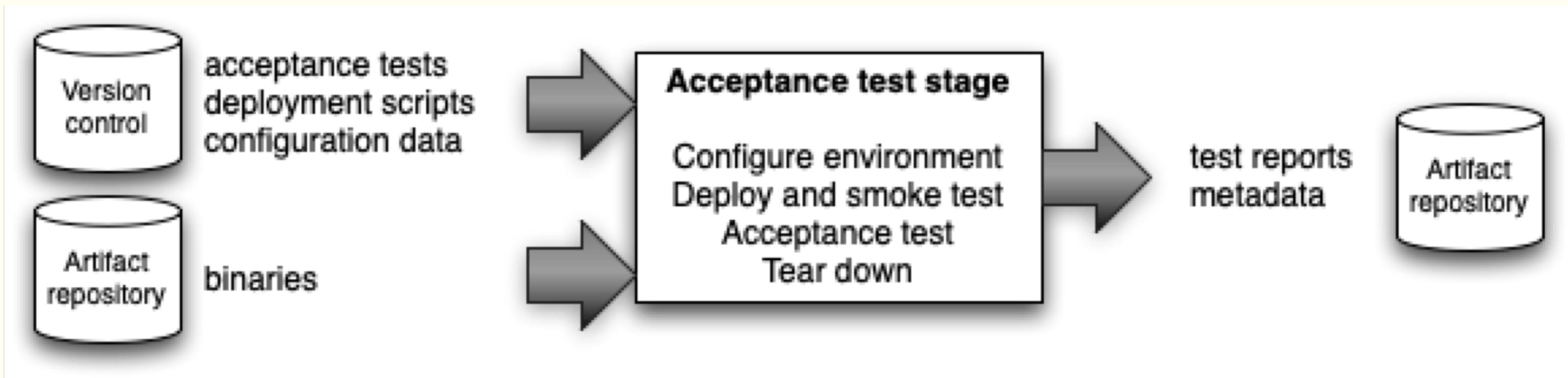


End-to-end tests in production-like environment

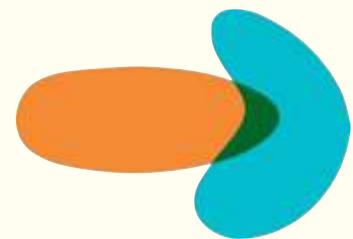


Triggered when upstream stage passes

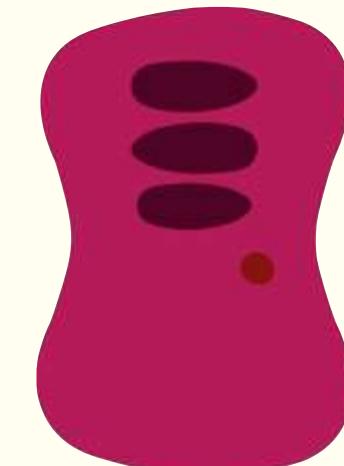
UAT Stage



End-to-end tests in production-like environment

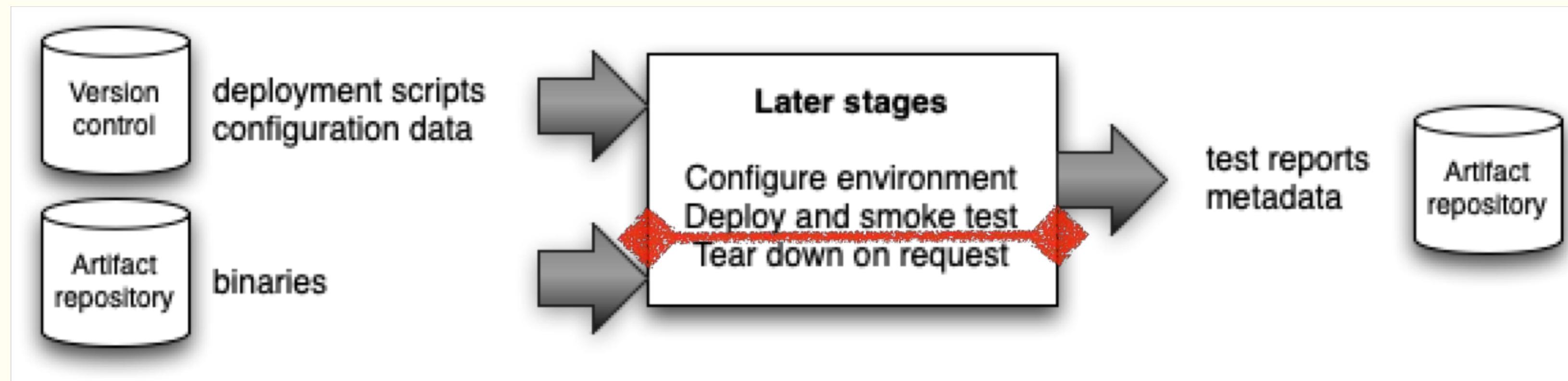


Triggered when upstream stage passes

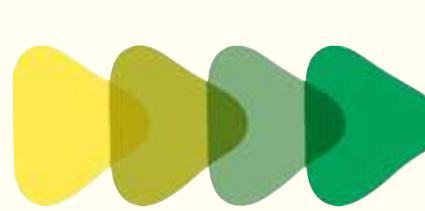
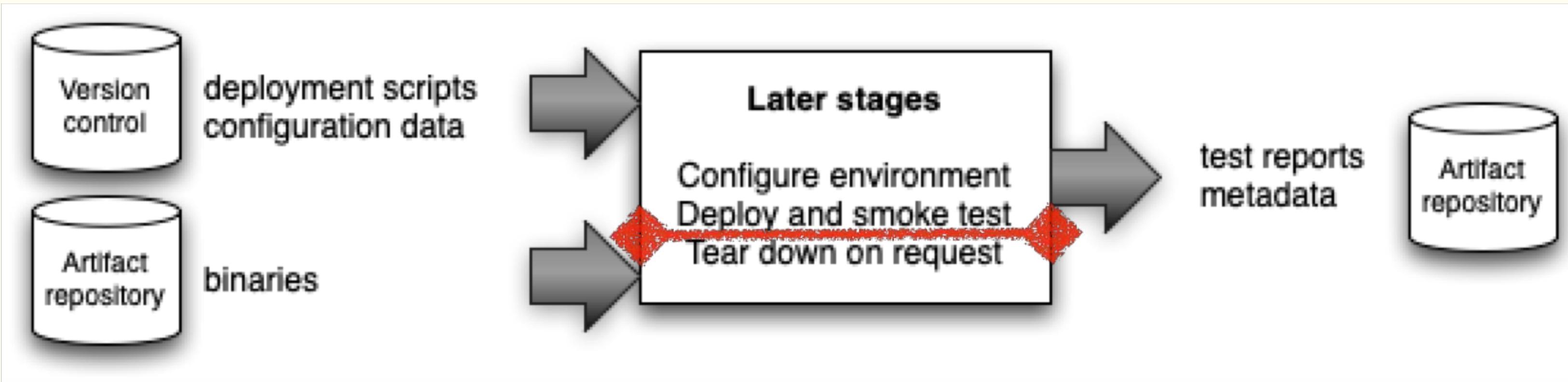


First DevOps-centric build

Manual Stage

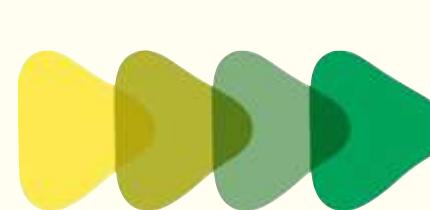
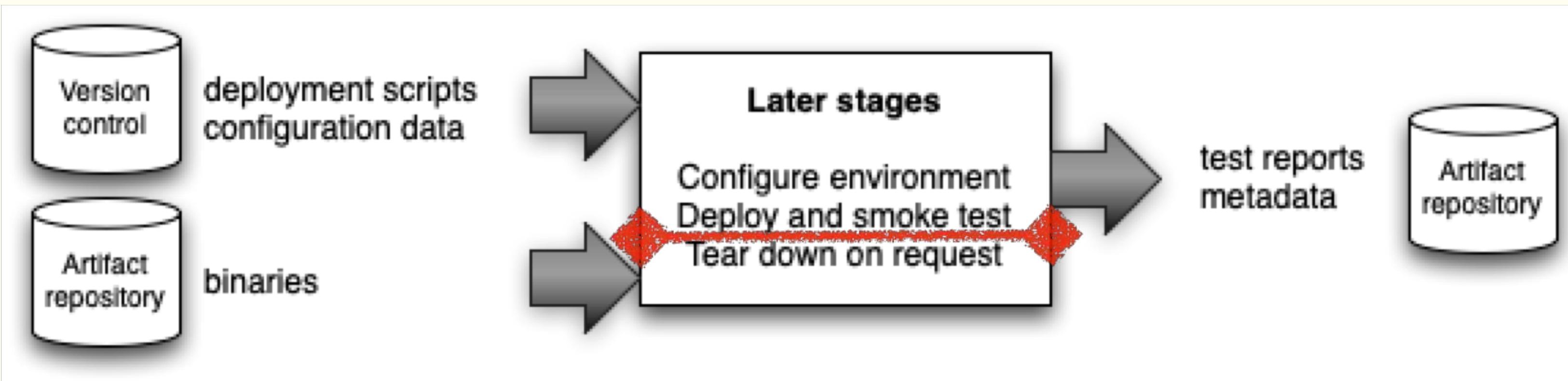


Manual Stage



UAT, staging, integration, production, ...

Manual Stage



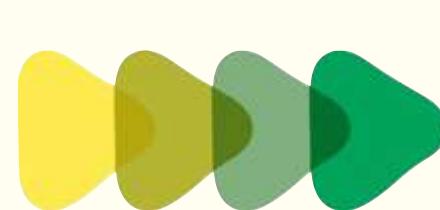
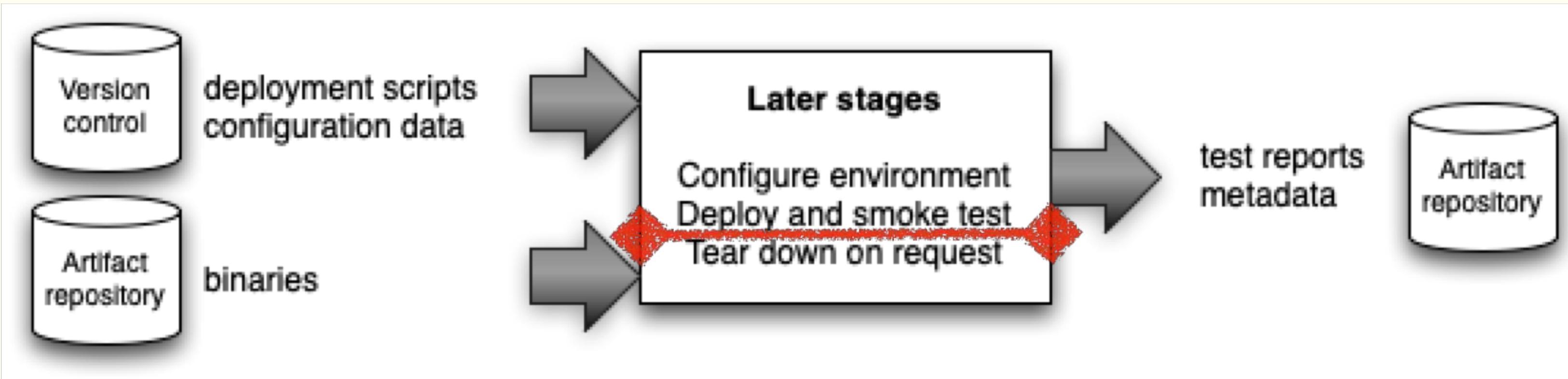
UAT, staging, integration, production, ...



Push versus Pull model



Manual Stage



UAT, staging, integration, production, ...

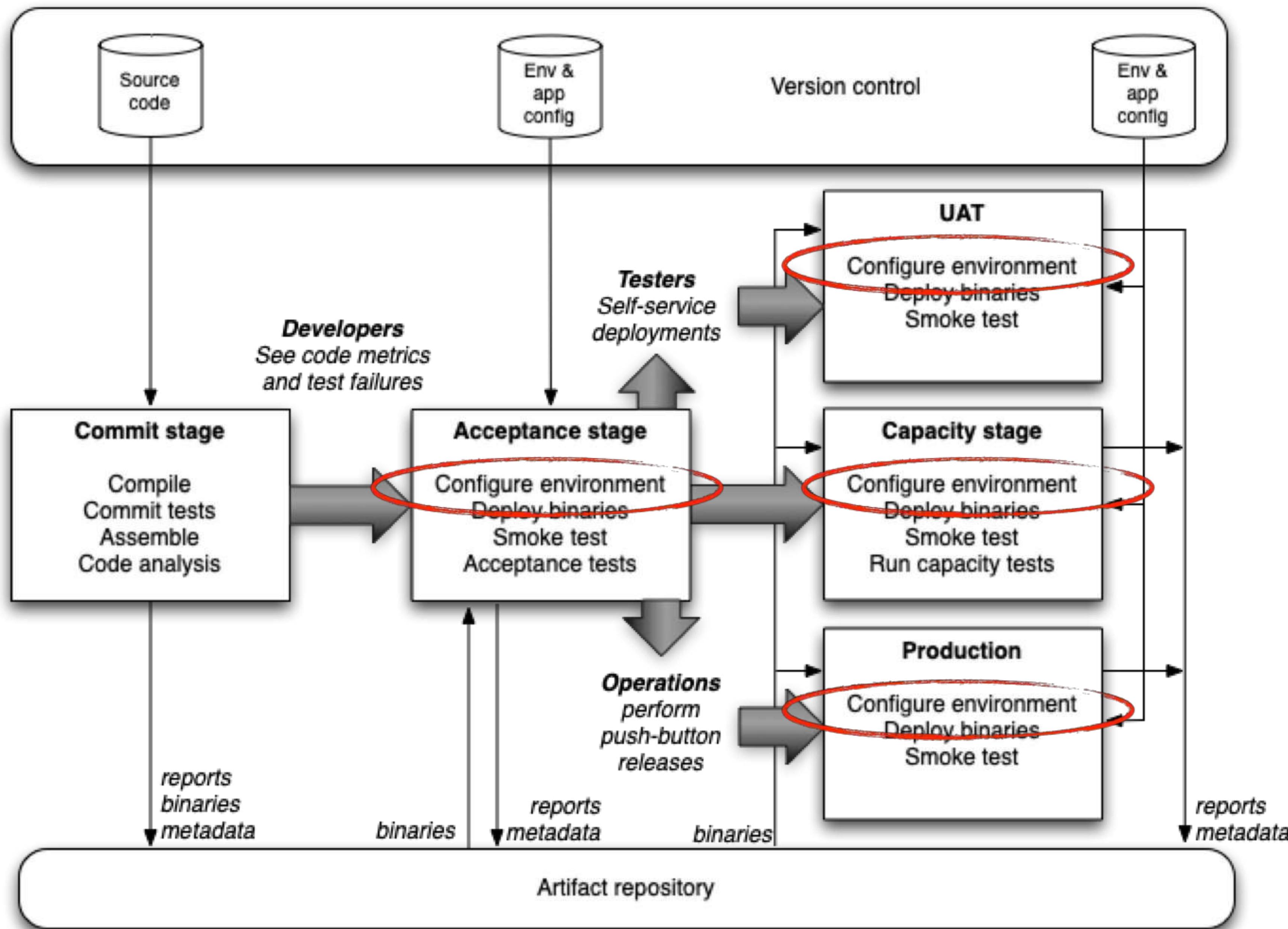


Push versus Pull model



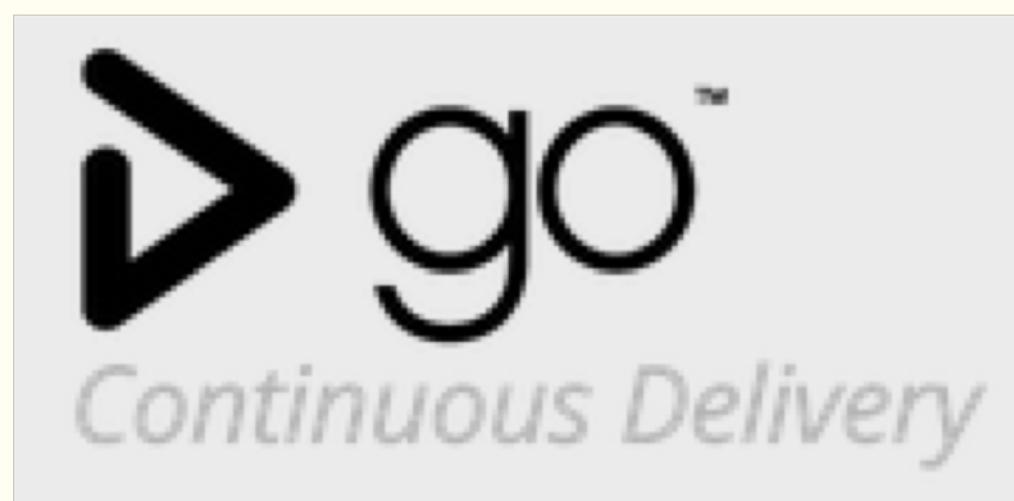
Deployments self-serviced through push-button process



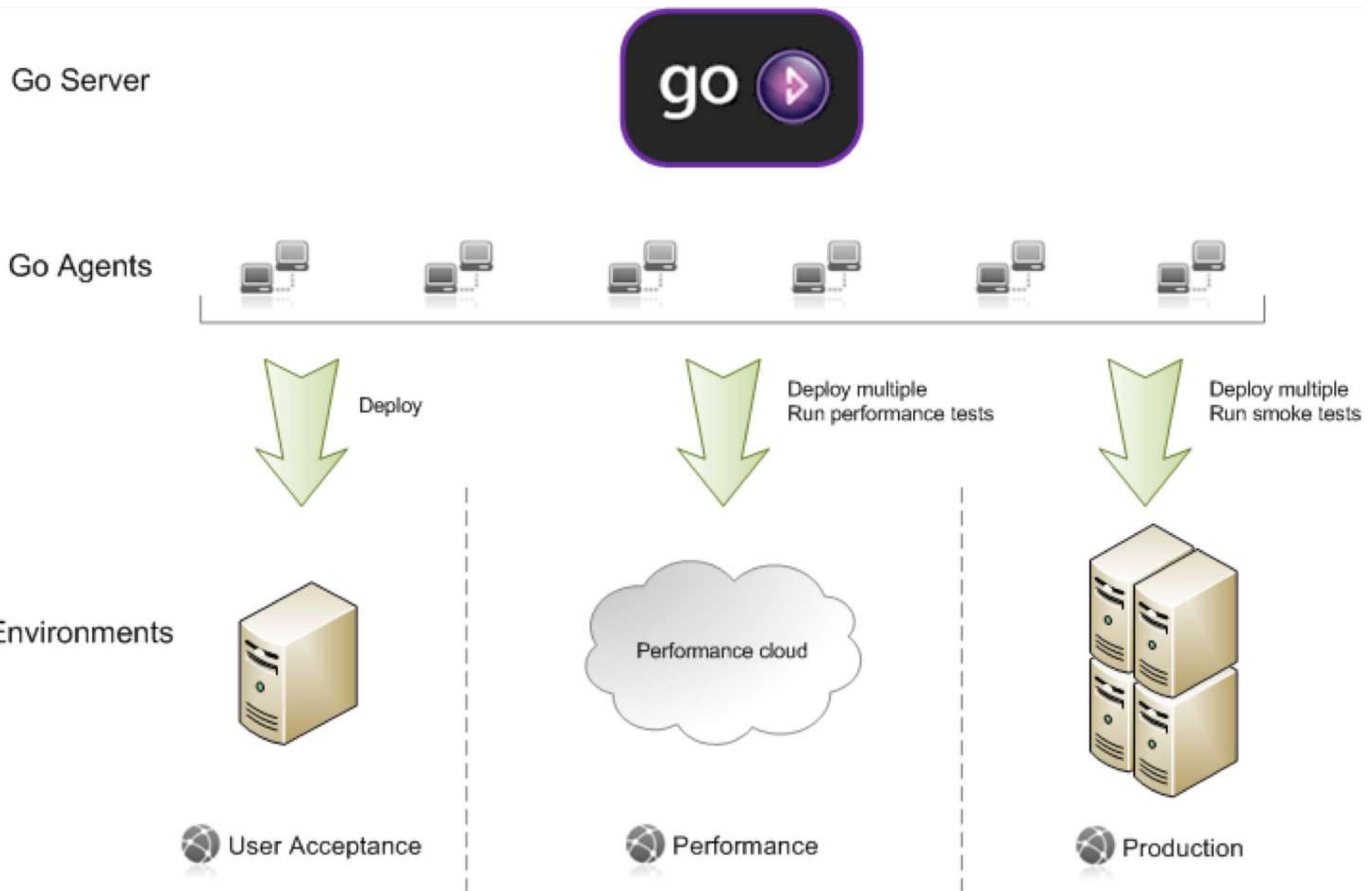


Machinery

continuous integration ++

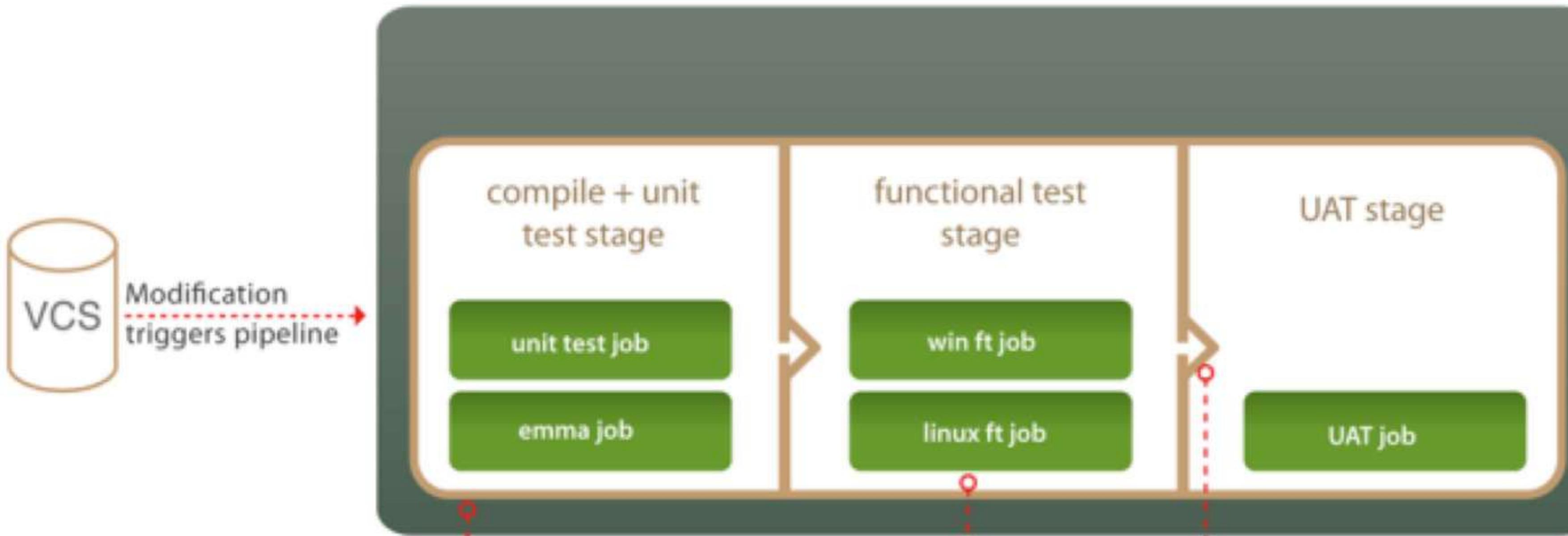


<https://www.gocd.org>



Code moves from check-in tests into UAT

Pipeline



Stage

- Stages run consecutively
- Each stage is triggered automatically by the successful completion of the previous stage
- Can also be triggered manually

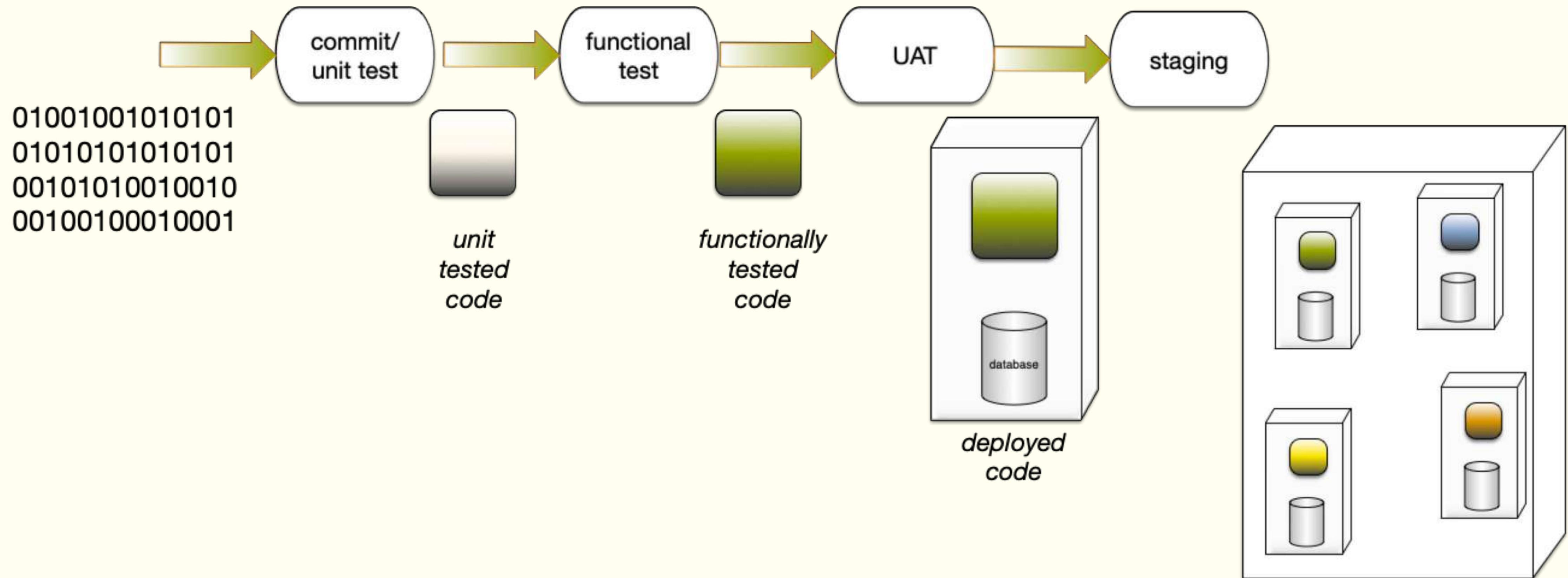
Job

- Jobs run concurrently within a stage
- If a job fails, the stage it is in fails
- Each job plan runs one or more targets (ant, nant, or eexec)
- Jobs can run in Parallel within a stage if you have multiple agents

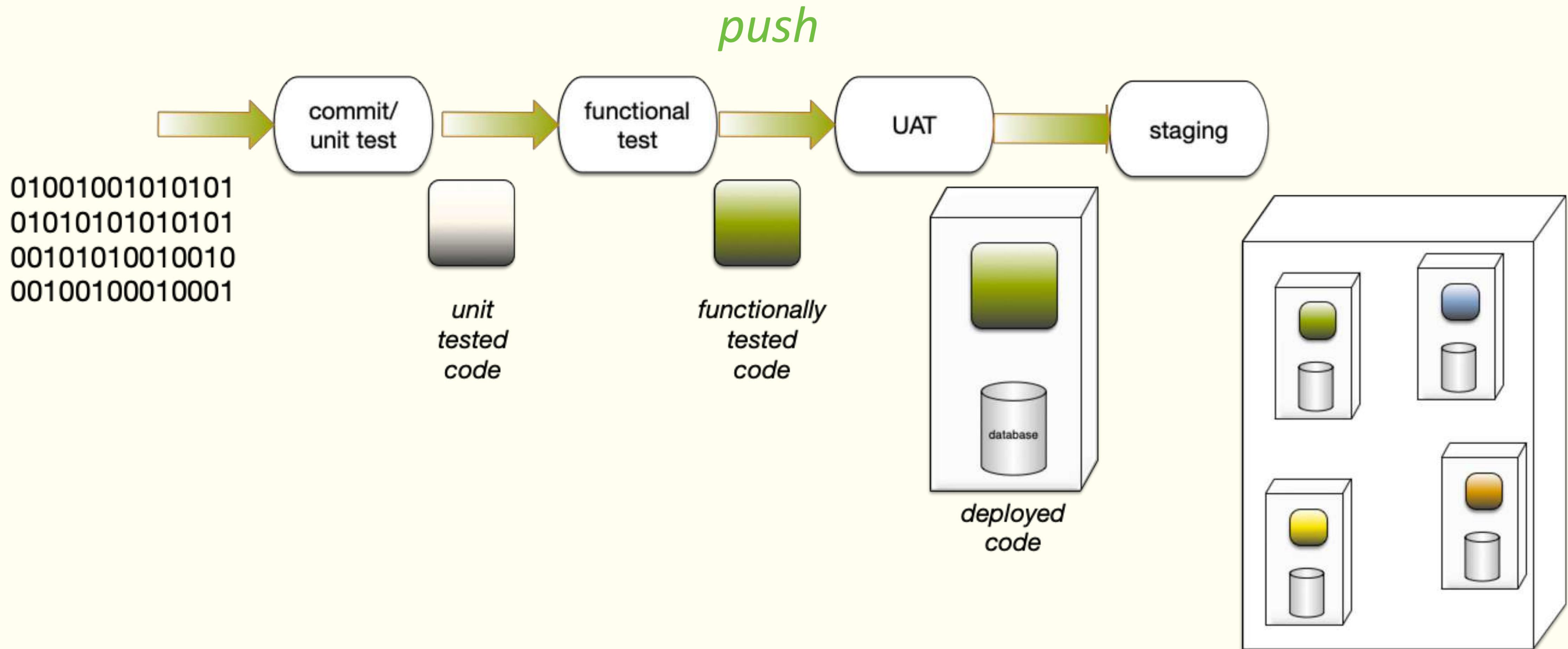
Approval

- Can be automatic or manual

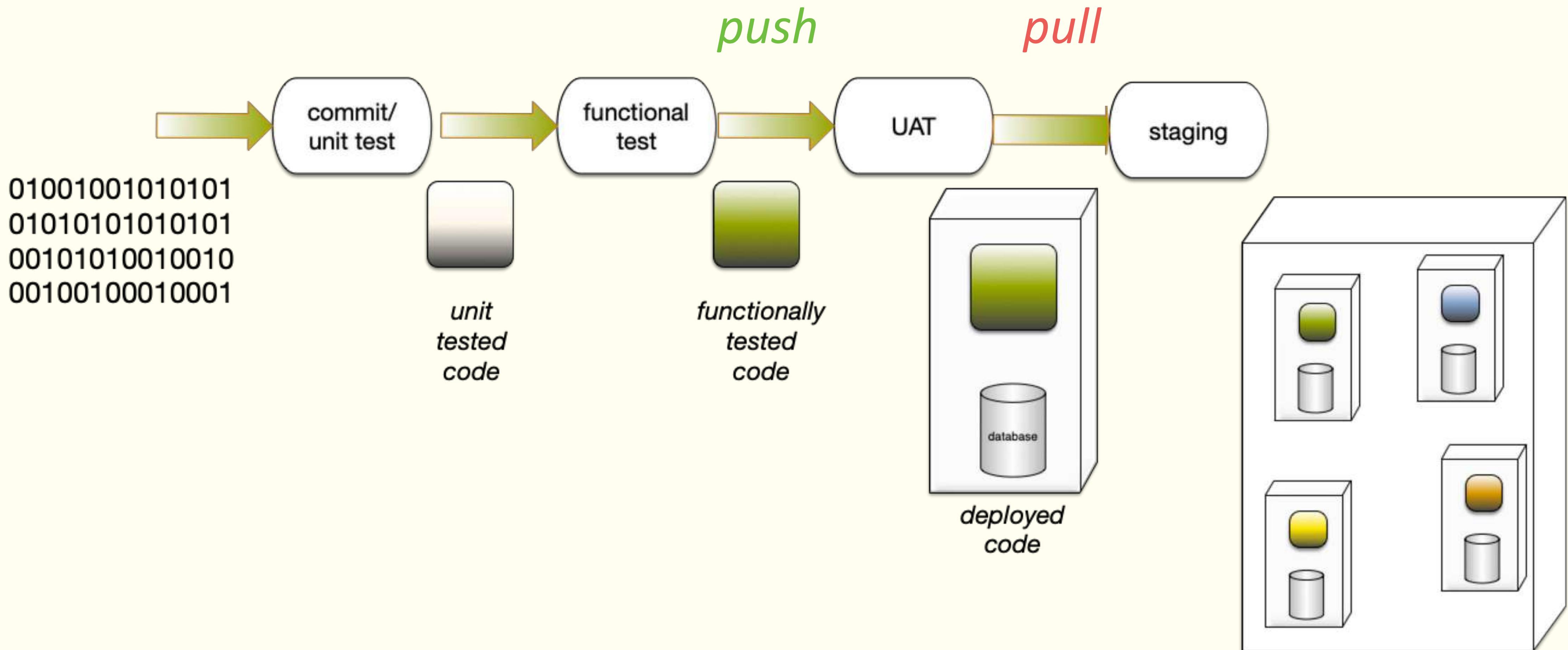
Deployment Pipeline

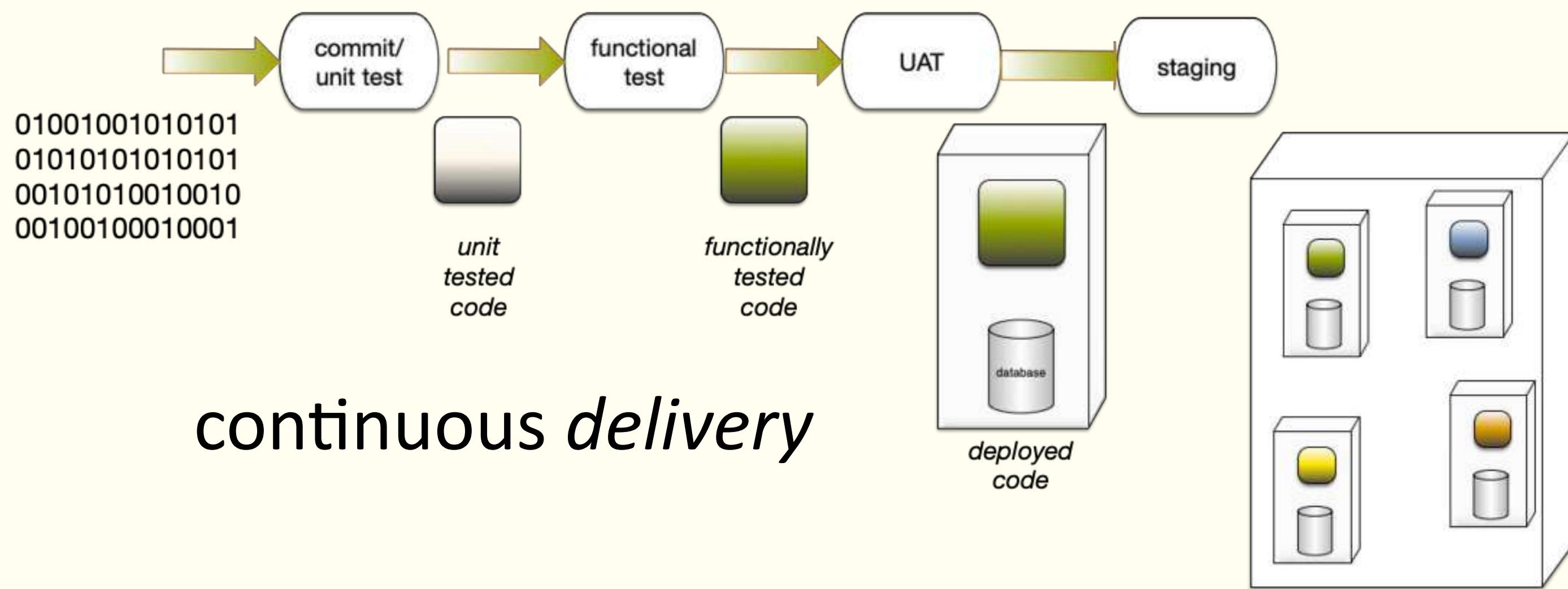


Deployment vs Delivery?

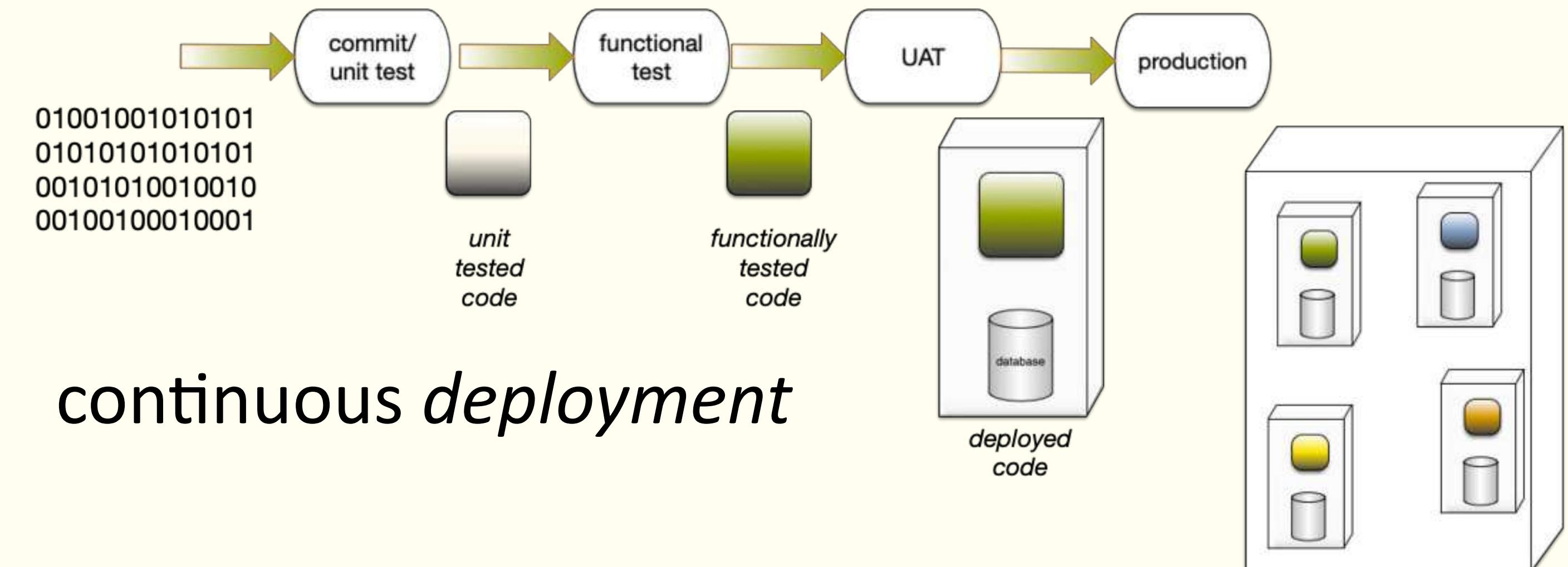


Deployment vs Delivery?

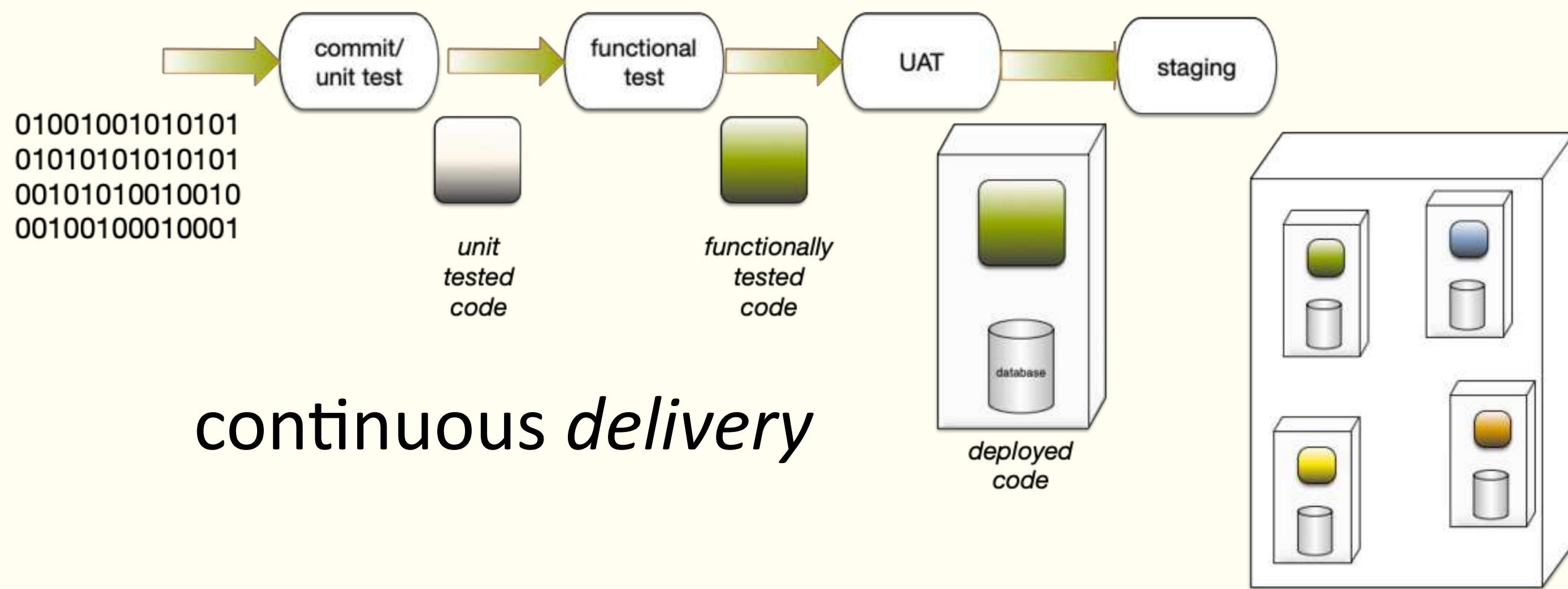




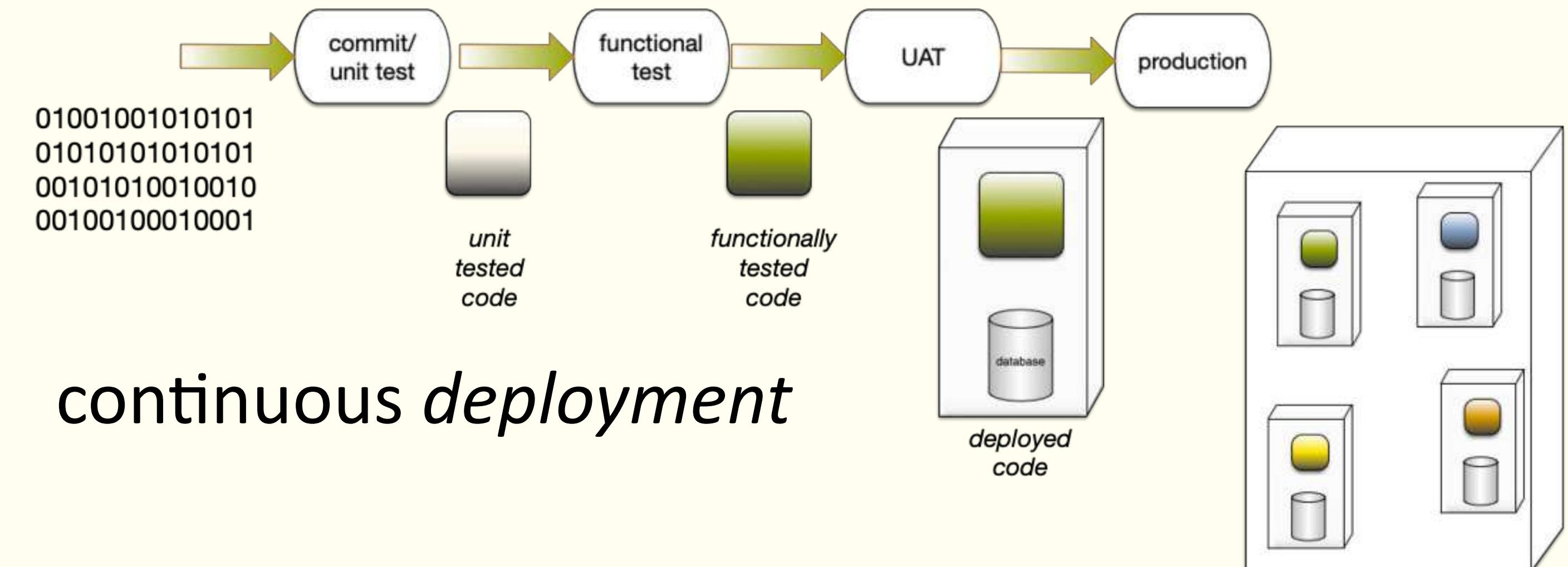
continuous delivery



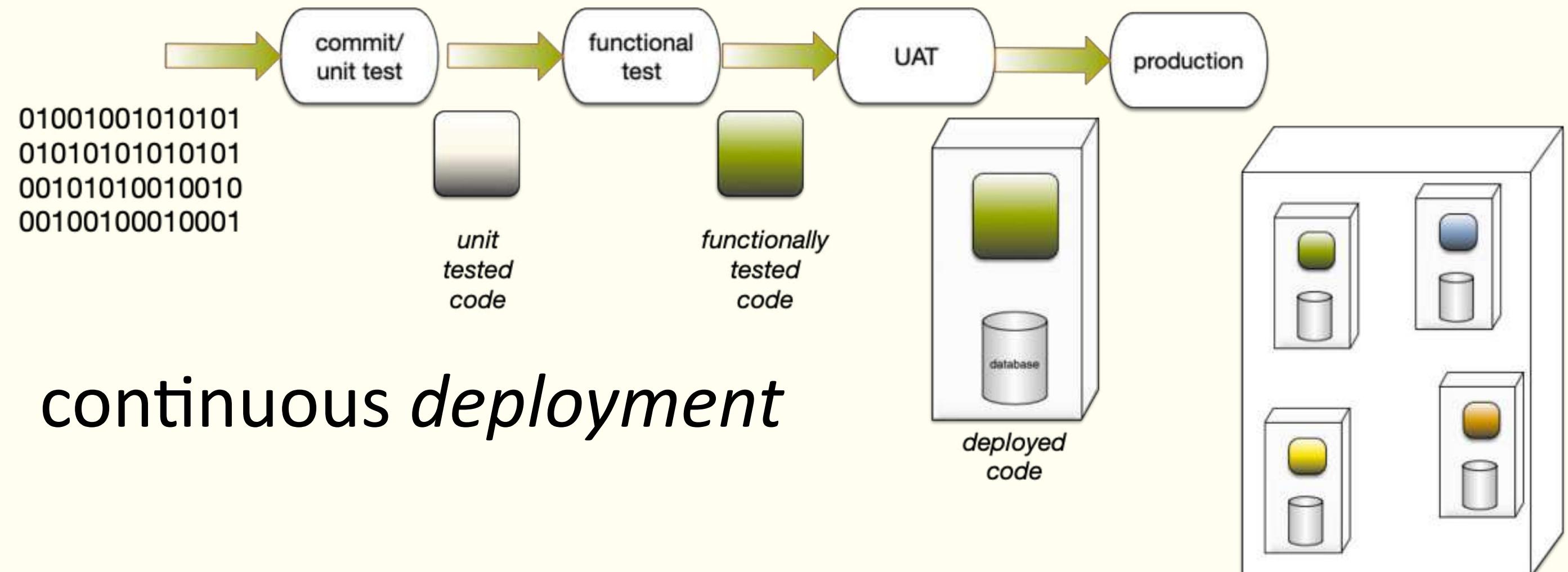
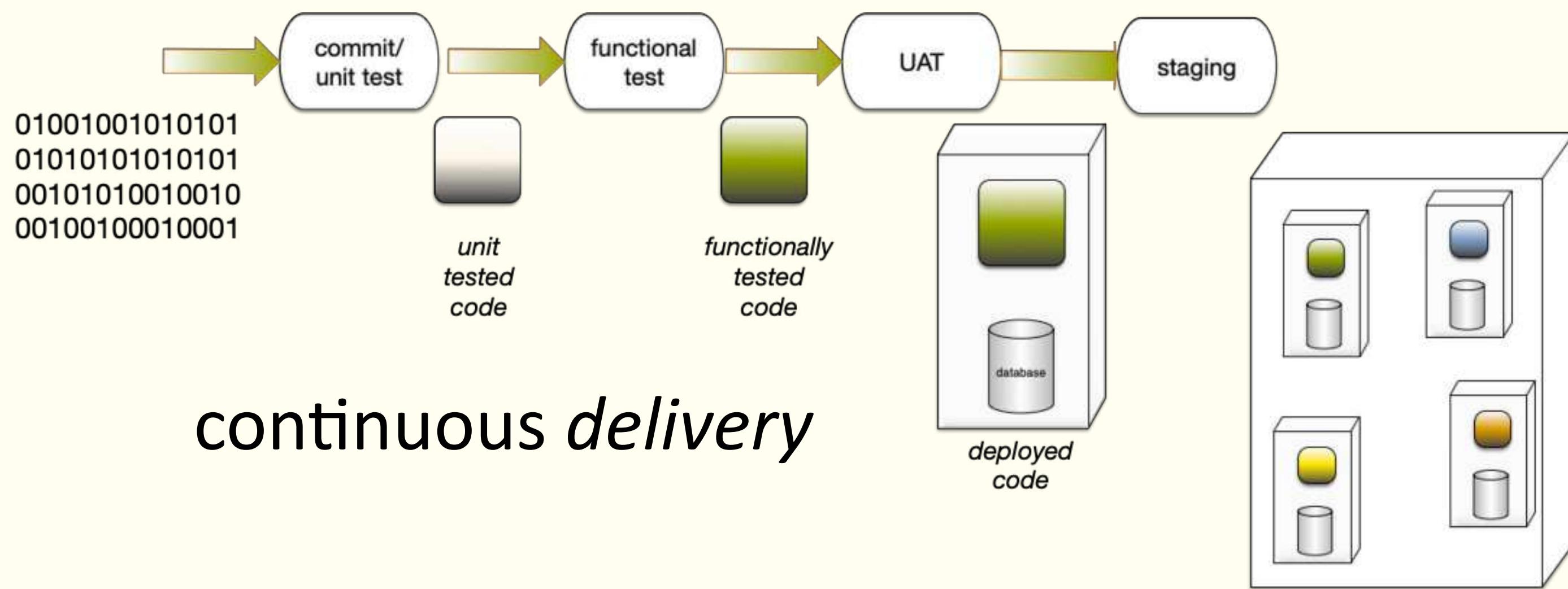
continuous deployment



continuous delivery

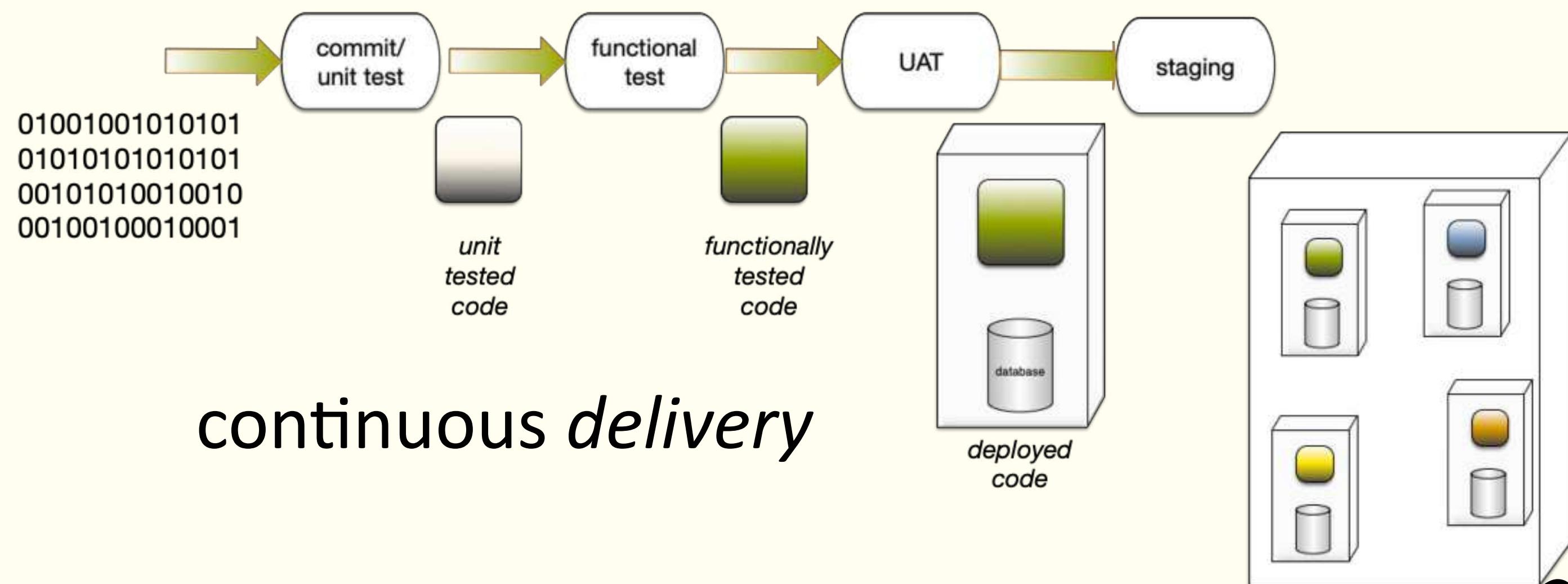


Continuous deployment: all *pushes*



Continuous deployment: all *push* stages

Continuous delivery: at least one *stage*



continuous delivery

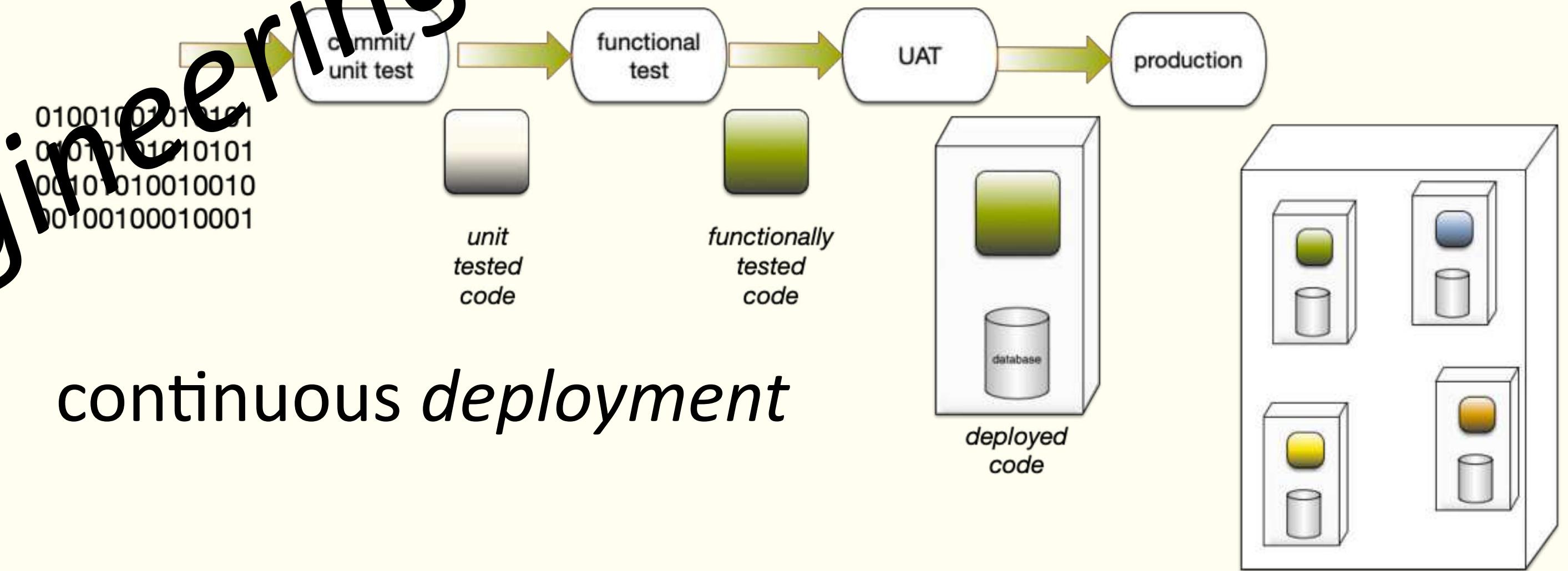
Continuous deployment: all

pushes

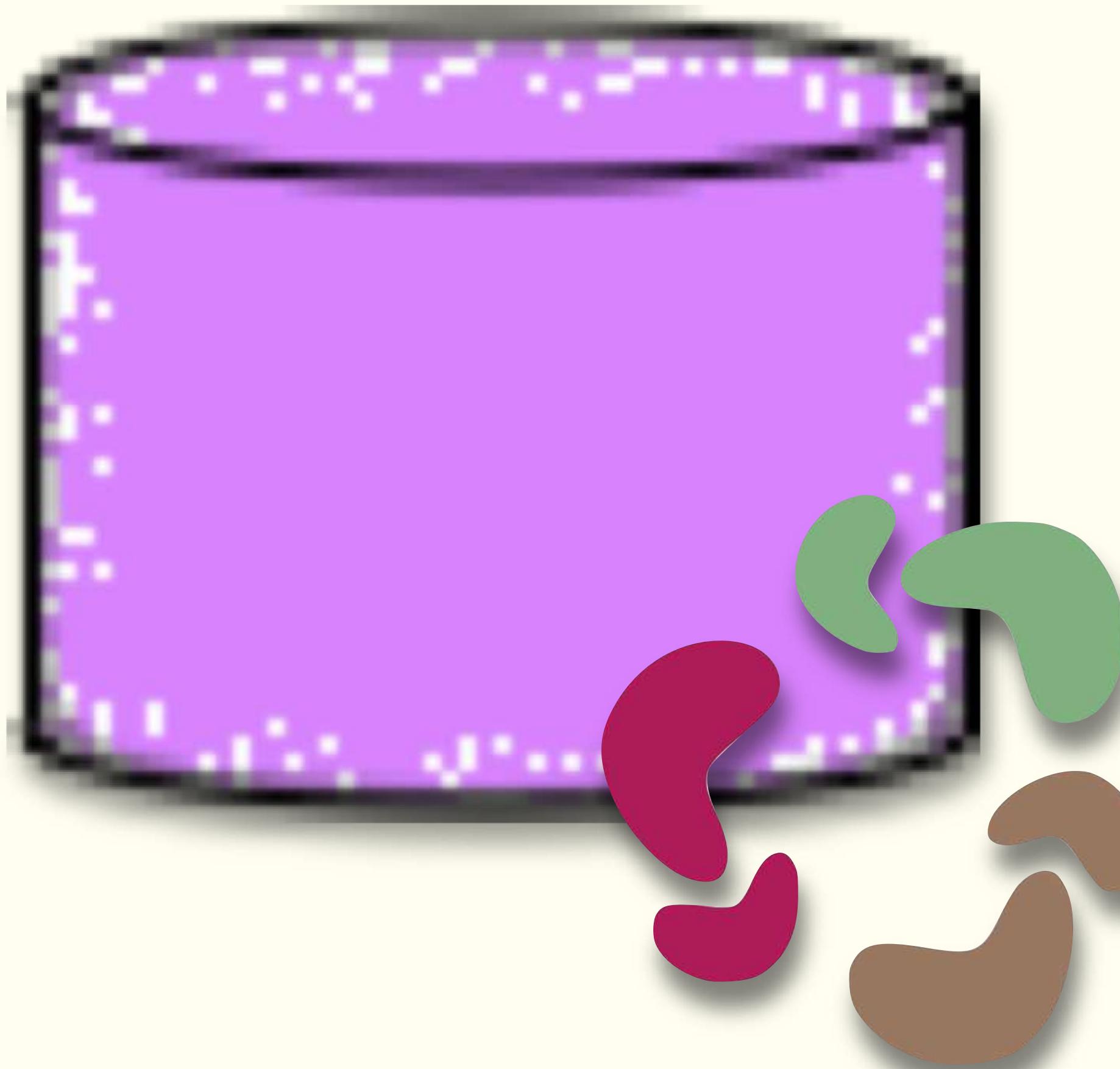
Continuous delivery: at least one

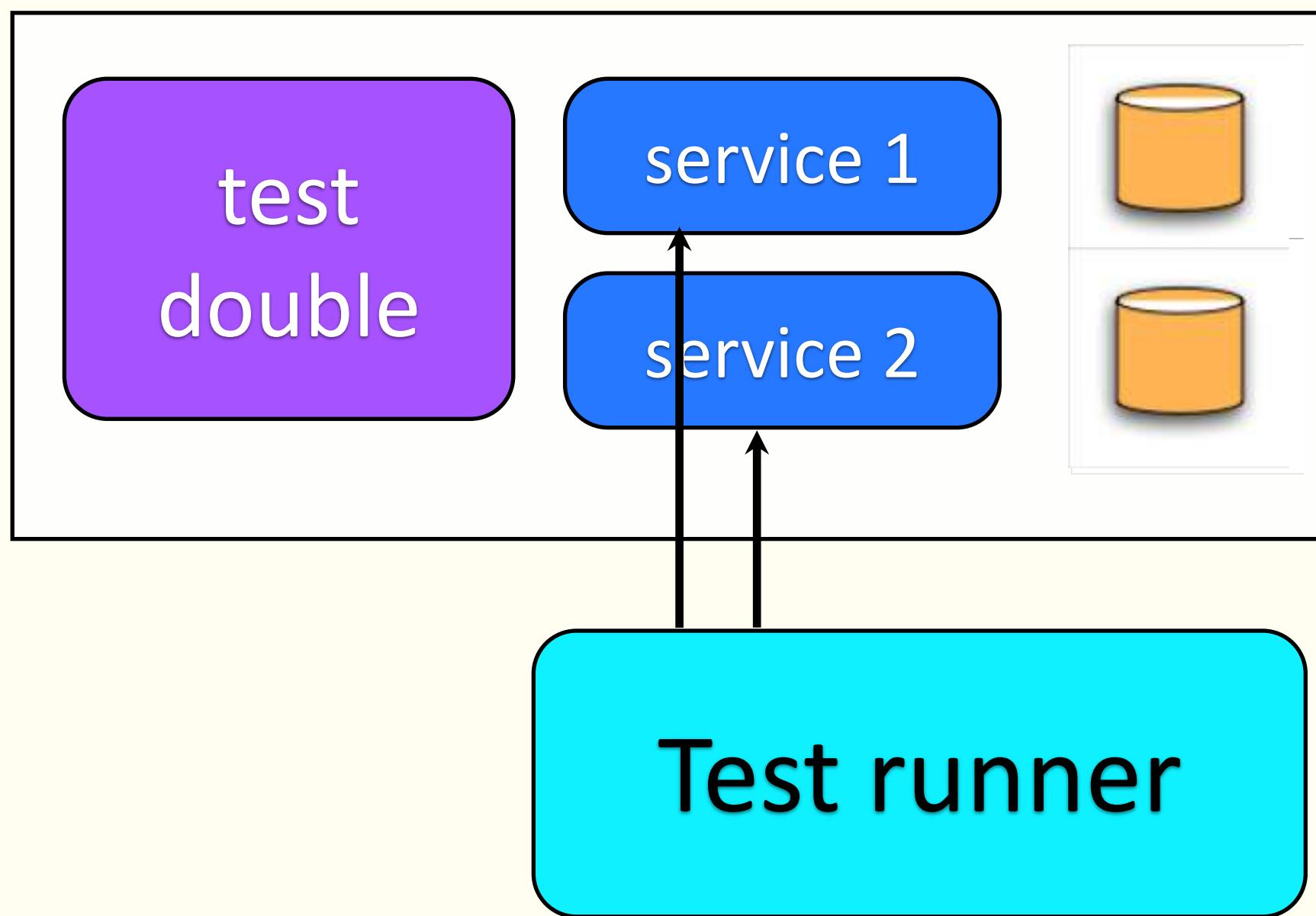
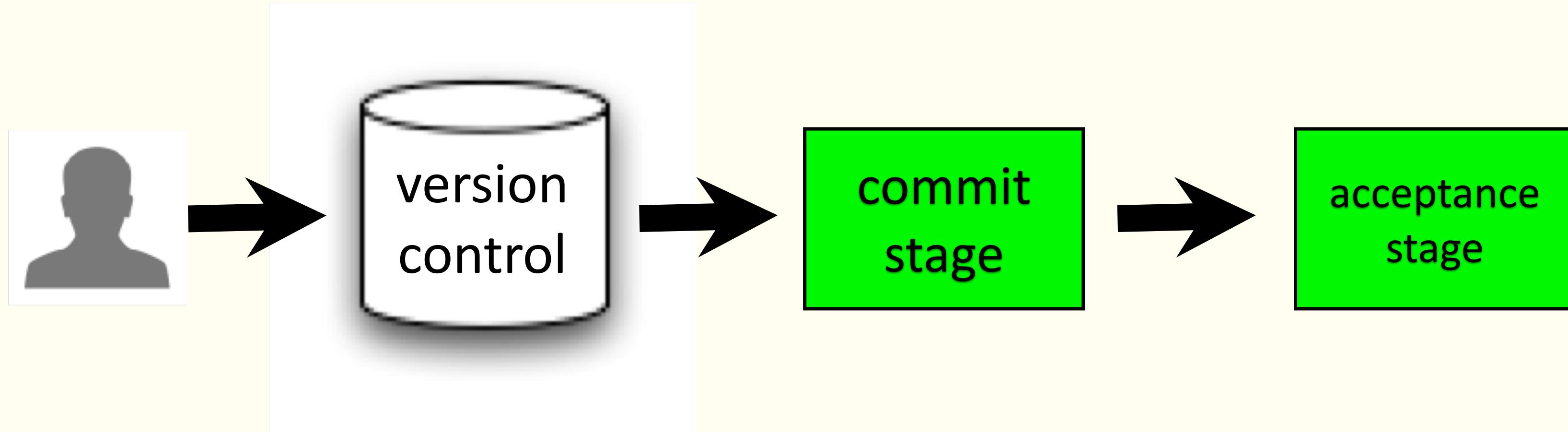
stage

same engineering practices



Continuous Integration for Databases





Prepare environment

Deploy app

Create dbs, apply schema

Add app reference data

Run acceptance tests

For DB CI We Need To:

start with a clean database

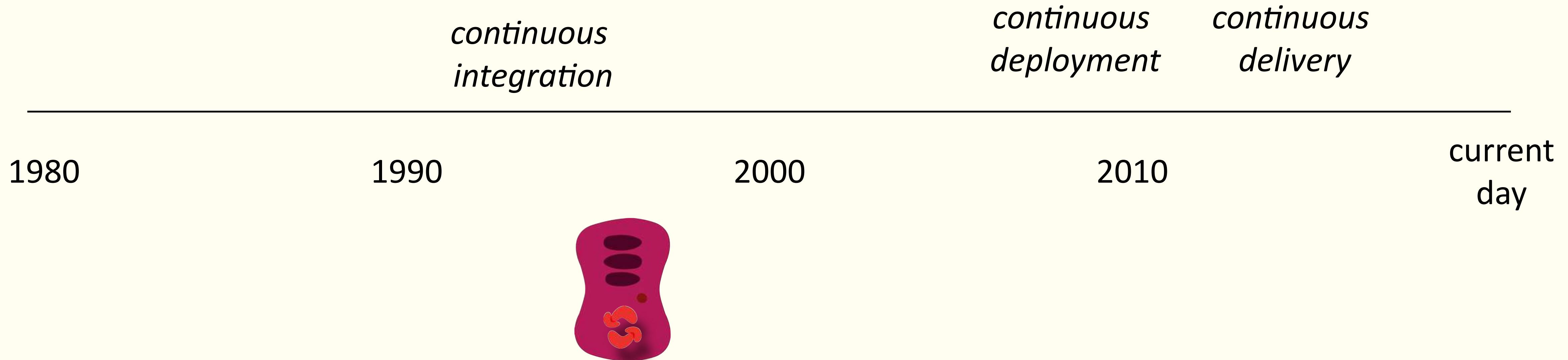


apply changes incrementally

use the same process everywhere

be comprehensive in change management

integration as an engineering practice over time



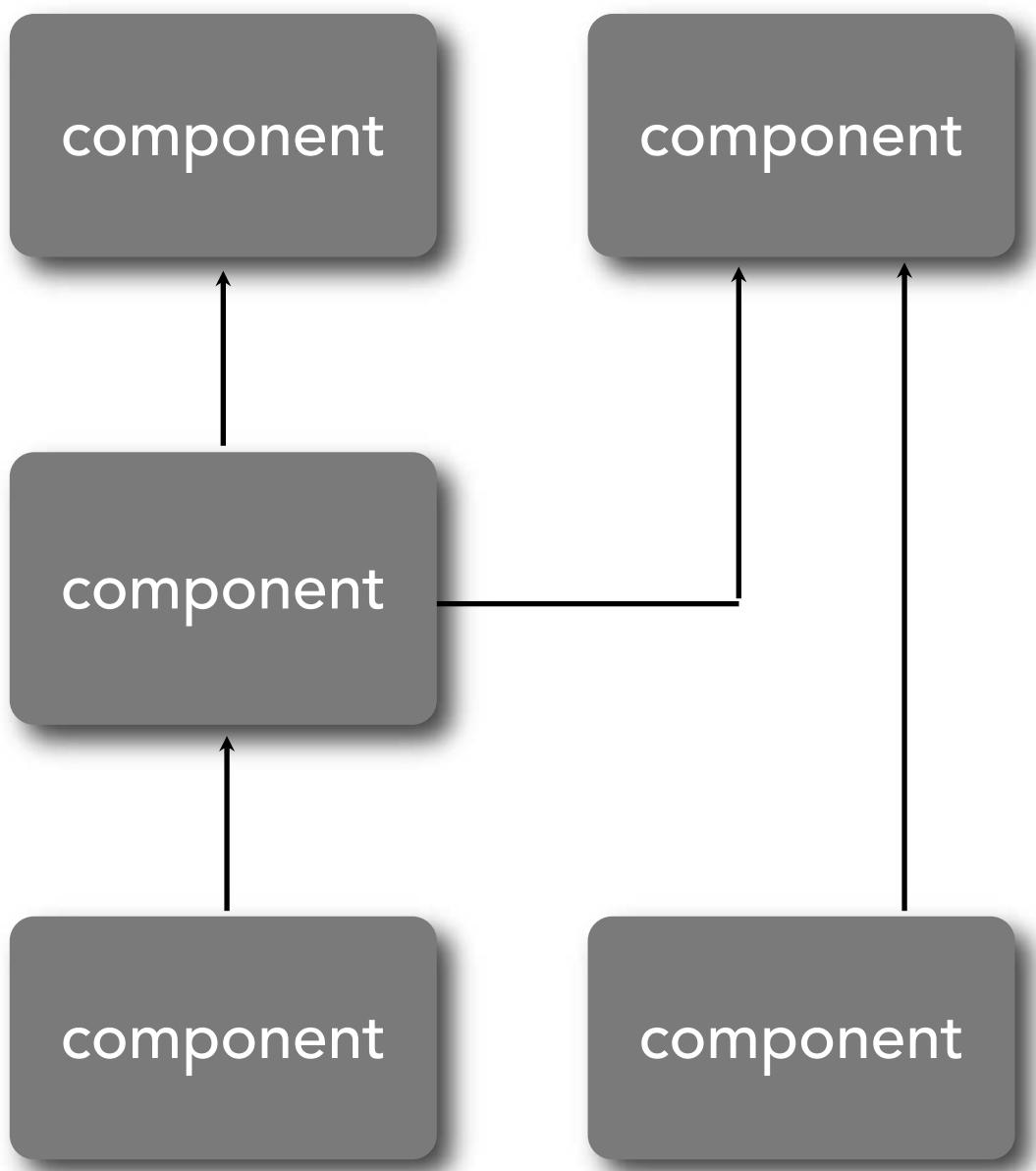
aRChiTeCtuRe
fOr
cONTiNuoUs DeLiVeRy

Question:

What characteristics of architecture make continuous delivery/deployment easier?

component identification

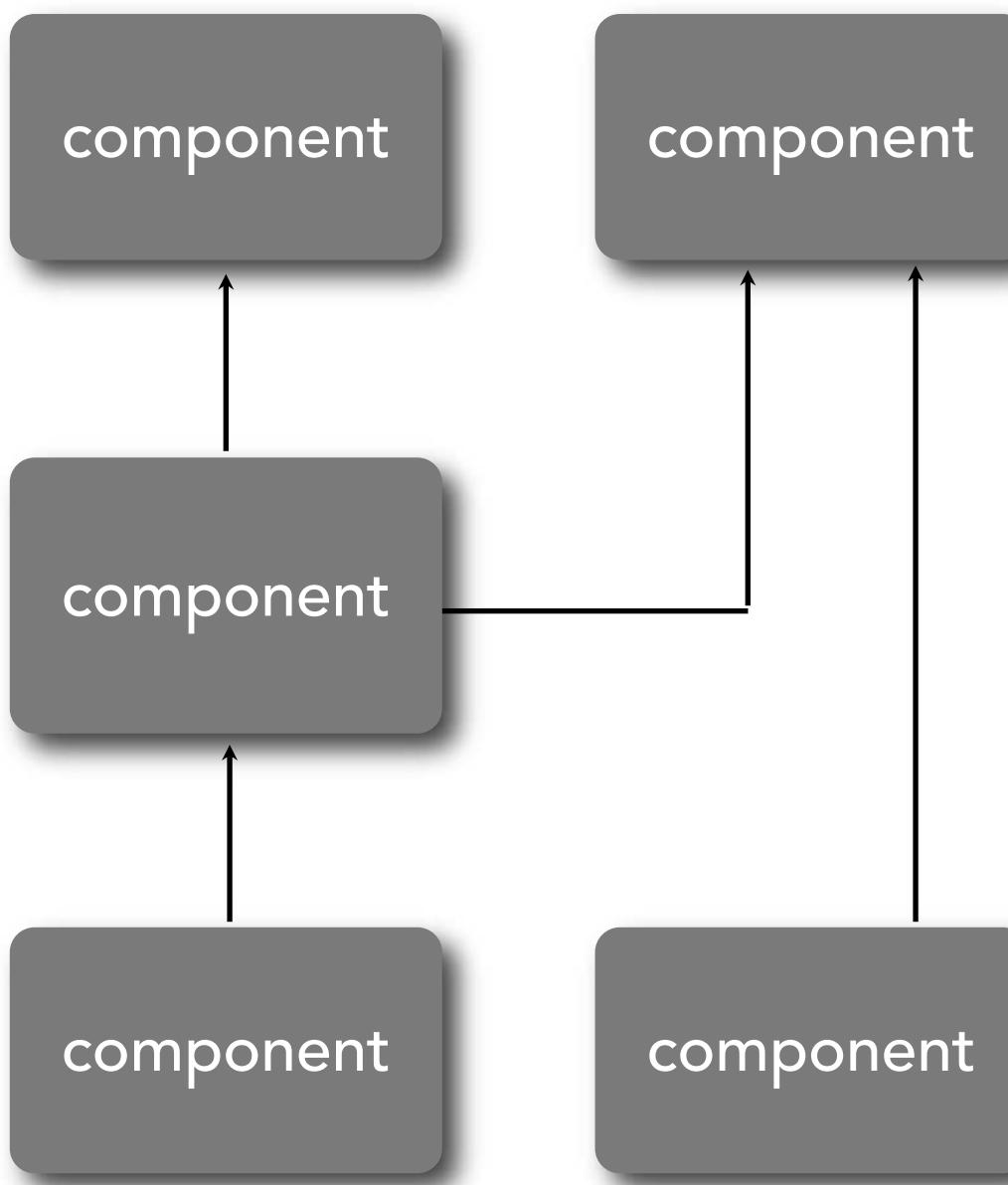
as an architect, you should think about the artifacts
within the architecture in terms of *components*



component:

component identification

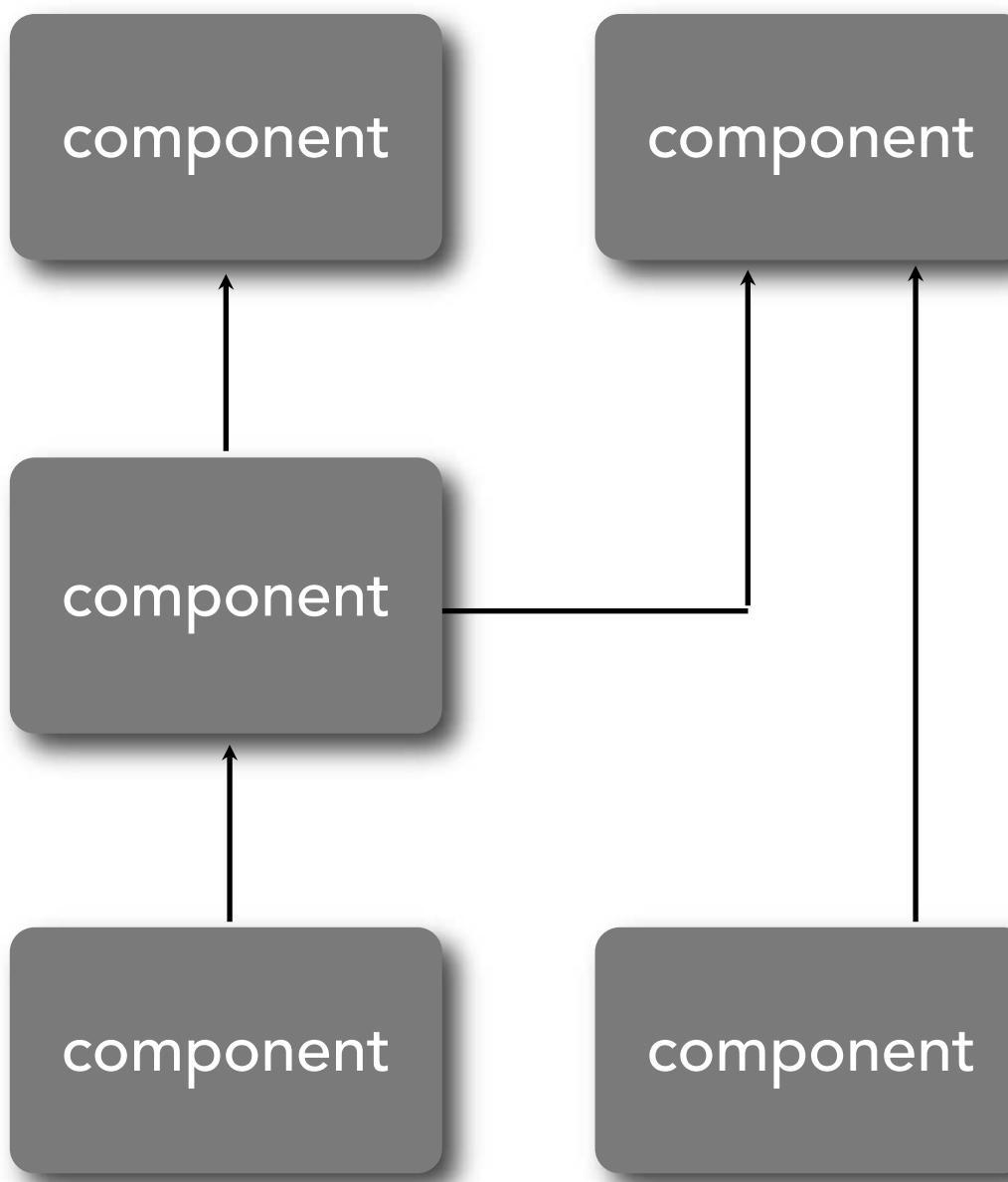
as an architect, you should think about the artifacts
within the architecture in terms of *components*



component:
building block of the application

component identification

as an architect, you should think about the artifacts within the architecture in terms of *components*

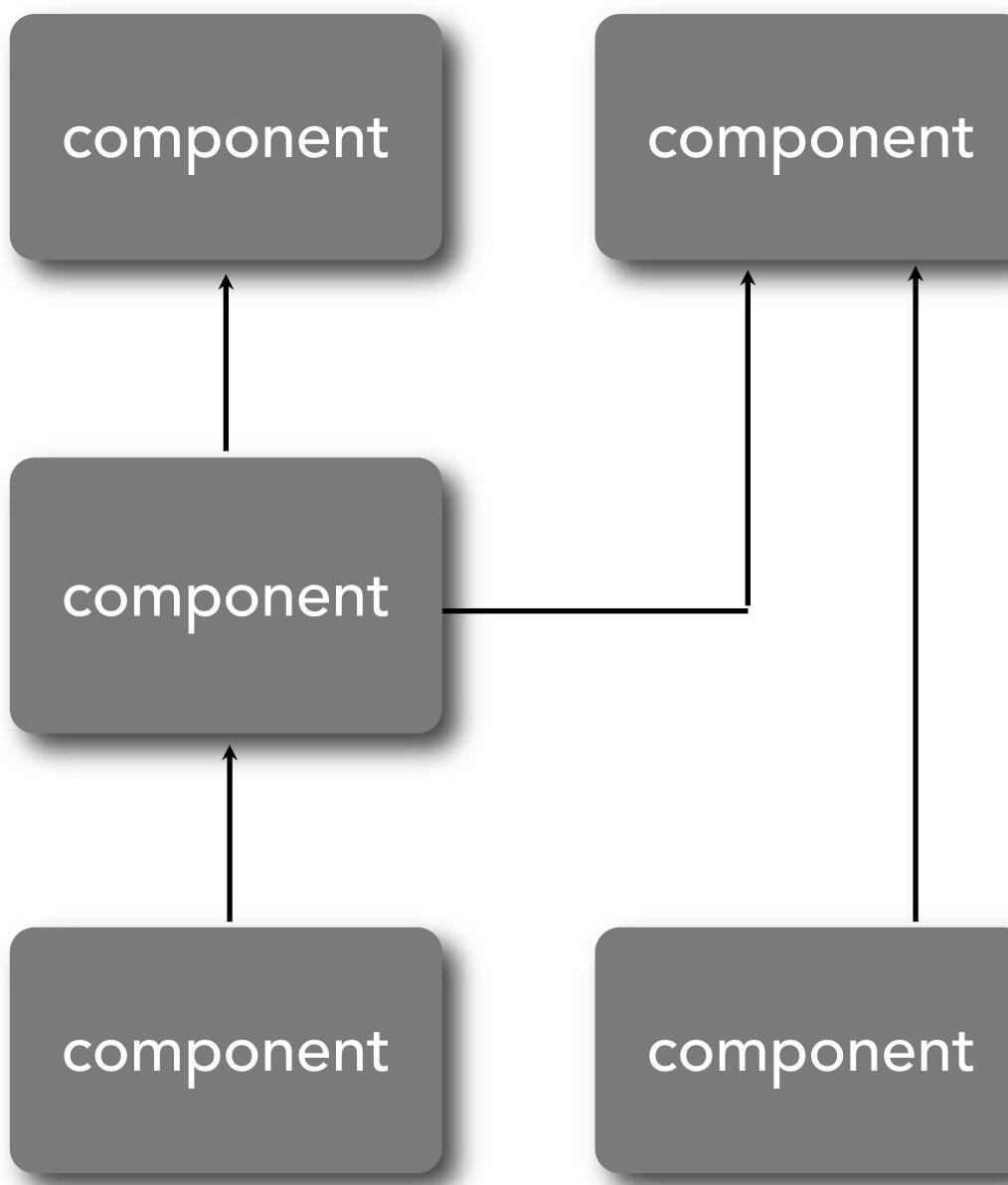


component:

building block of the application
well defined set of operations

component identification

as an architect, you should think about the artifacts within the architecture in terms of *components*



component:

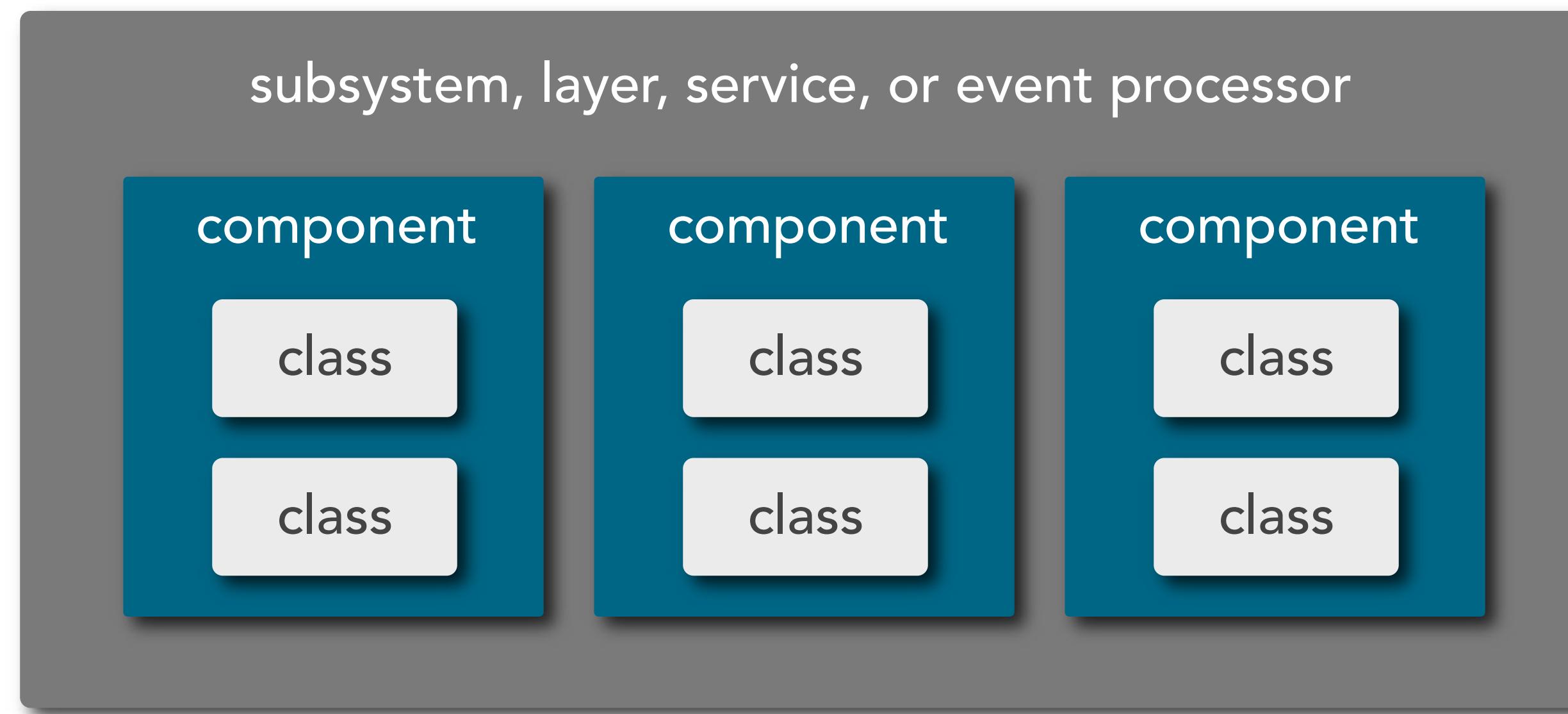
building block of the application

well defined set of operations

well defined role and responsibility

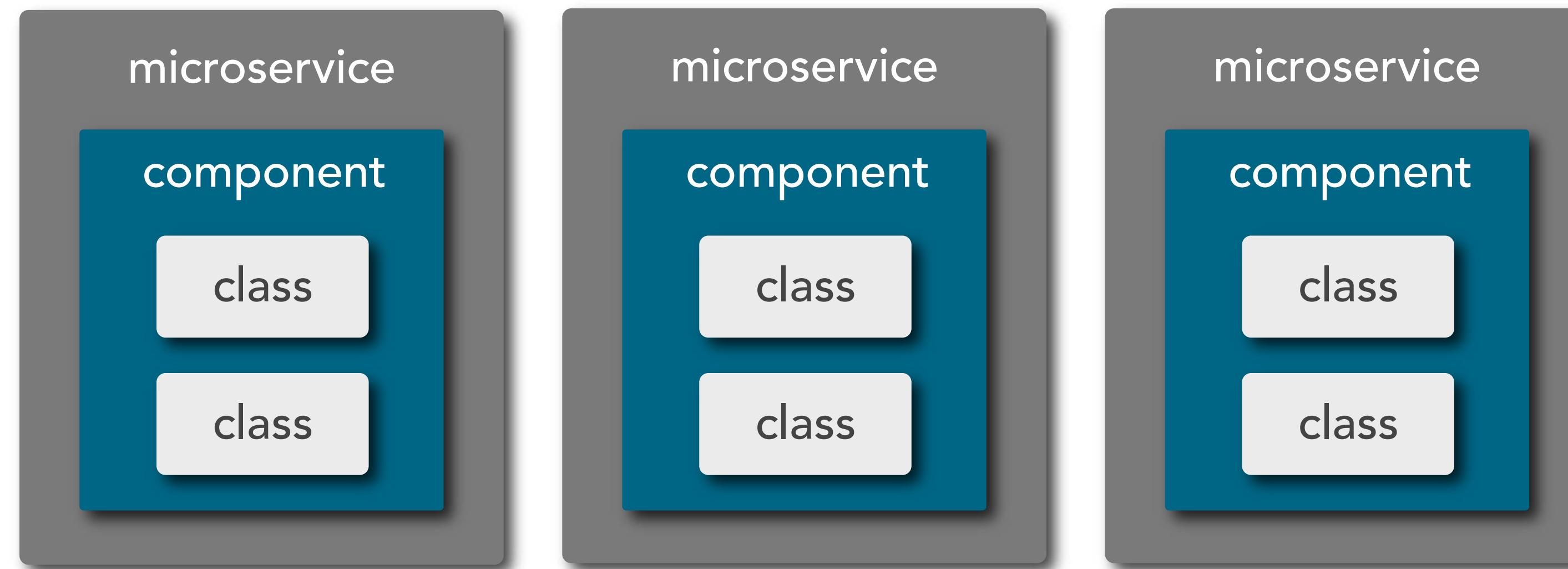
component identification

component scope



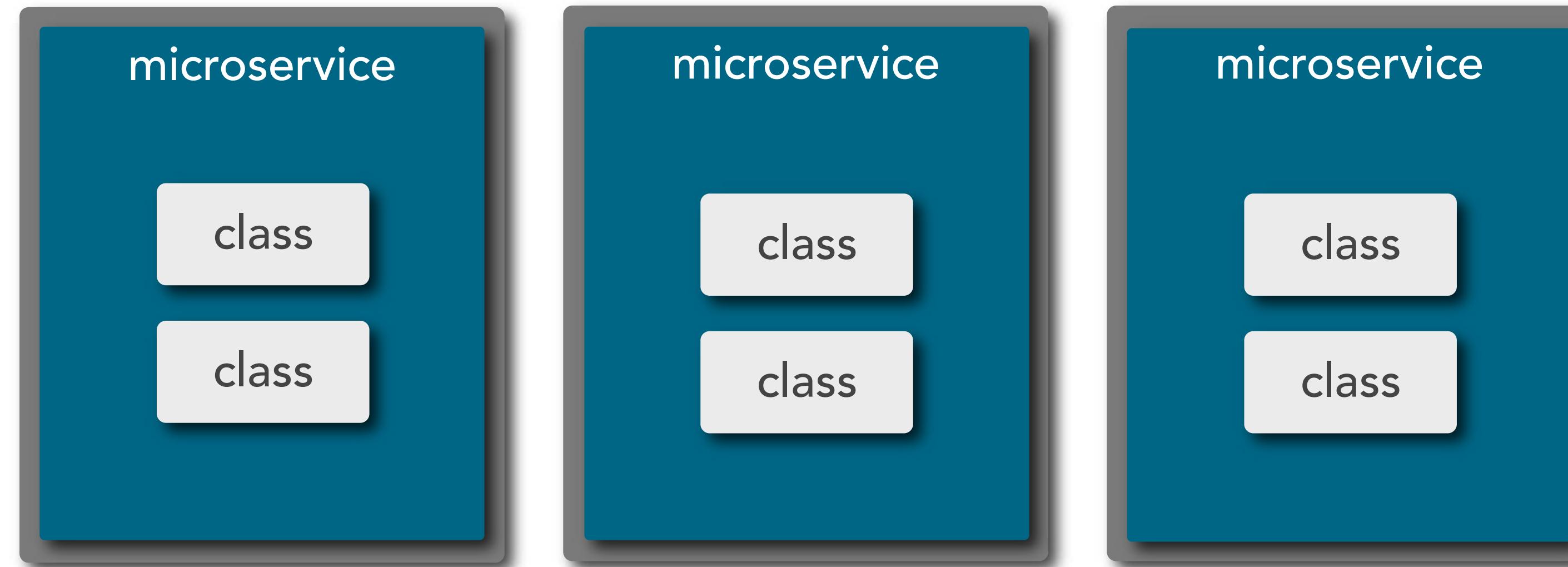
component identification

component scope



component identification

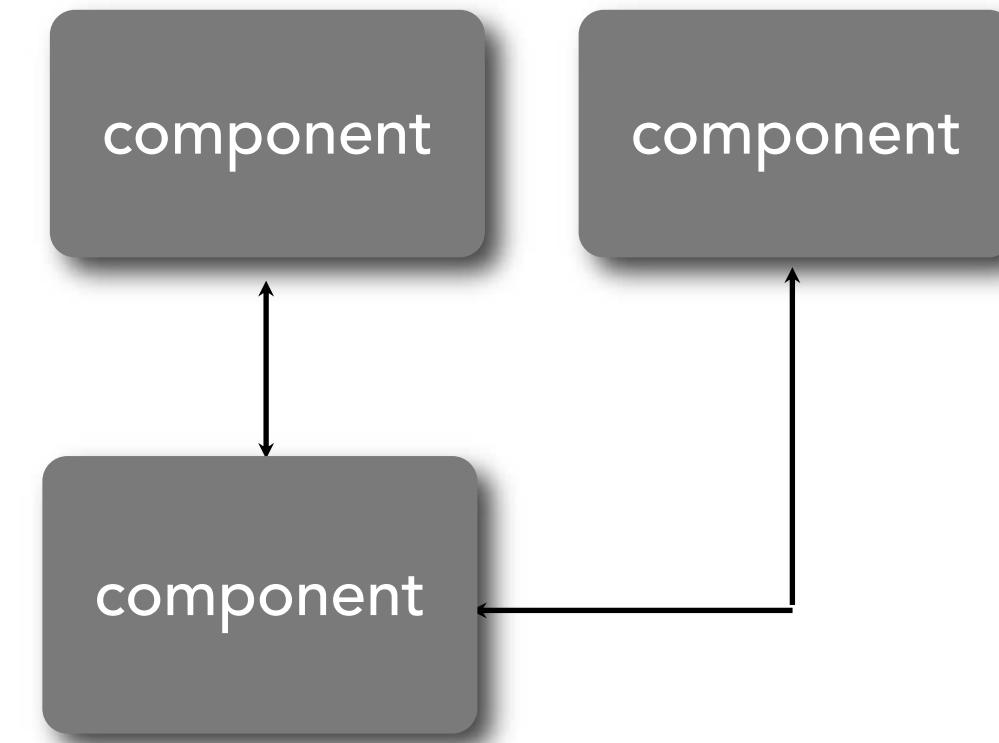
component scope



component coupling

component coupling

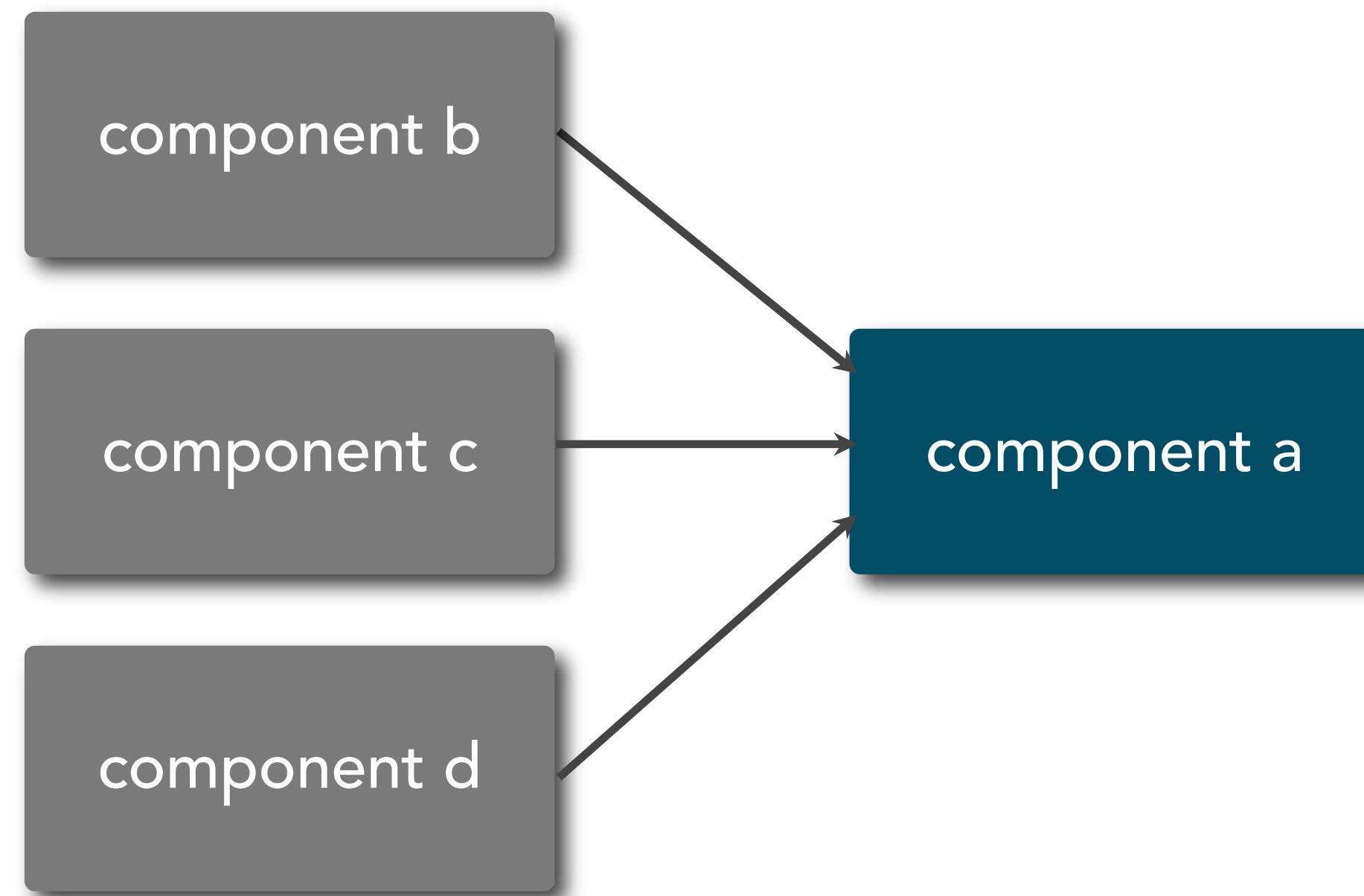
the extent to which components know
about each other



component coupling

afferent coupling

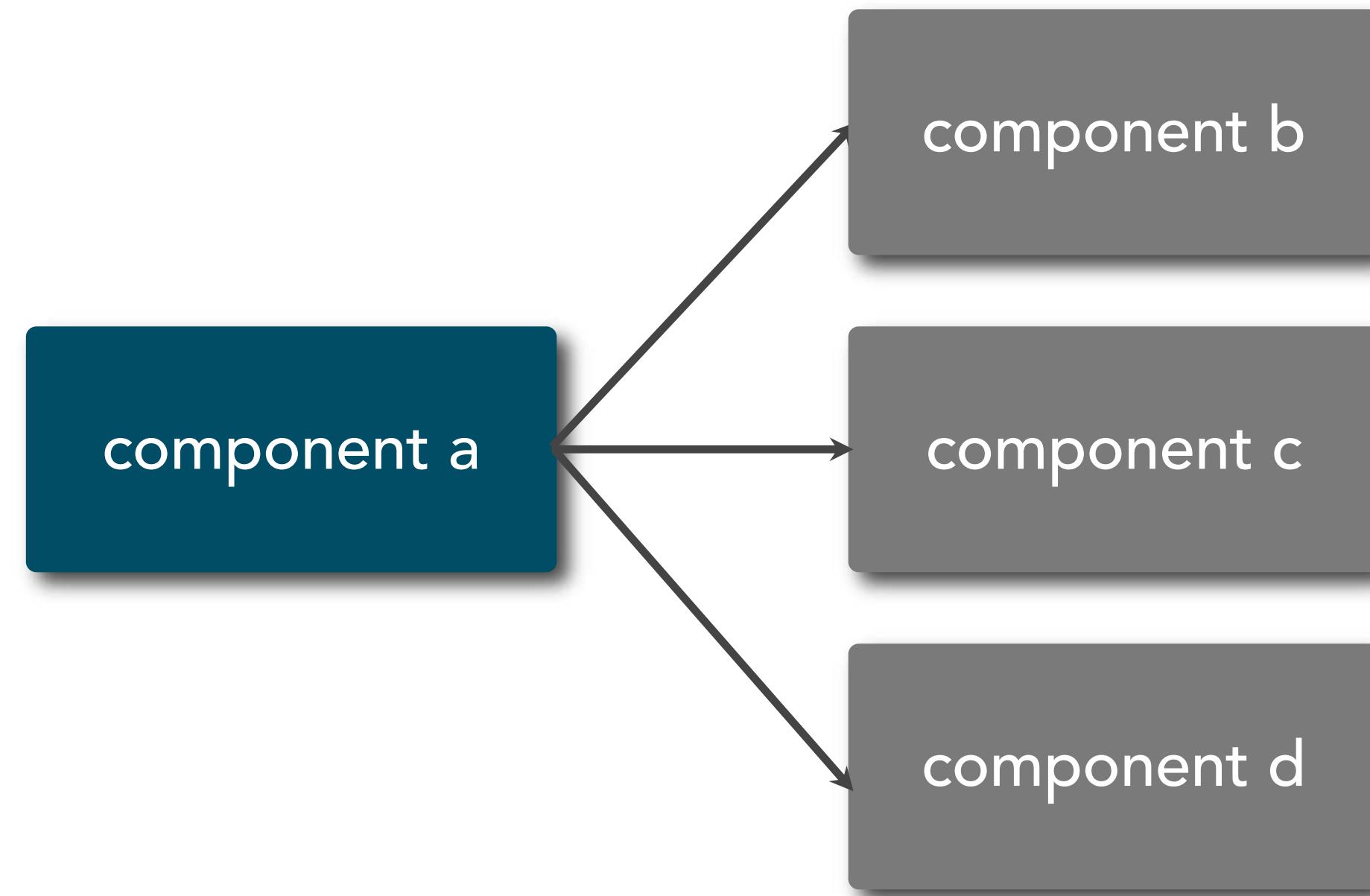
the degree to which other components are dependent on the target component



component coupling

efferent coupling

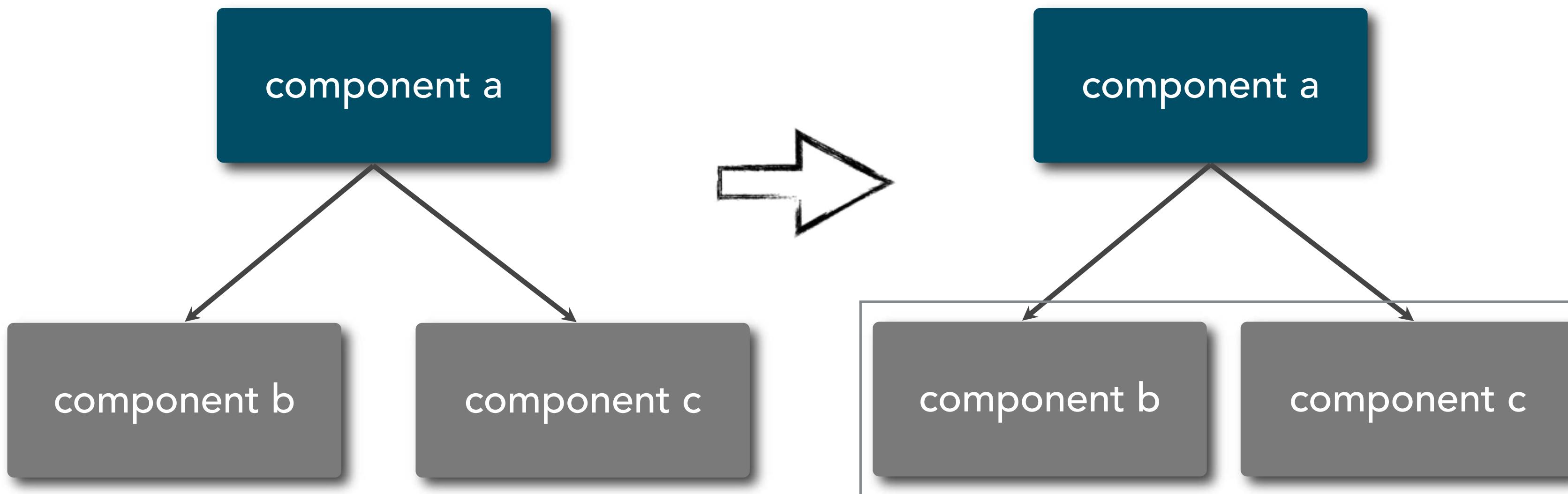
the degree to which the target component
is dependent on other components



component coupling

temporal coupling

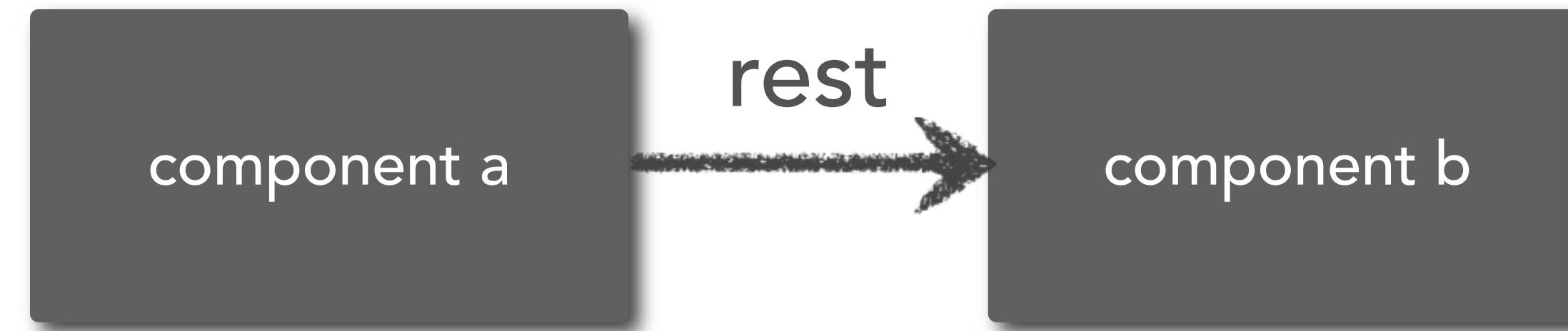
components are coupled due to non-static or timing dependencies



component coupling

external coupling

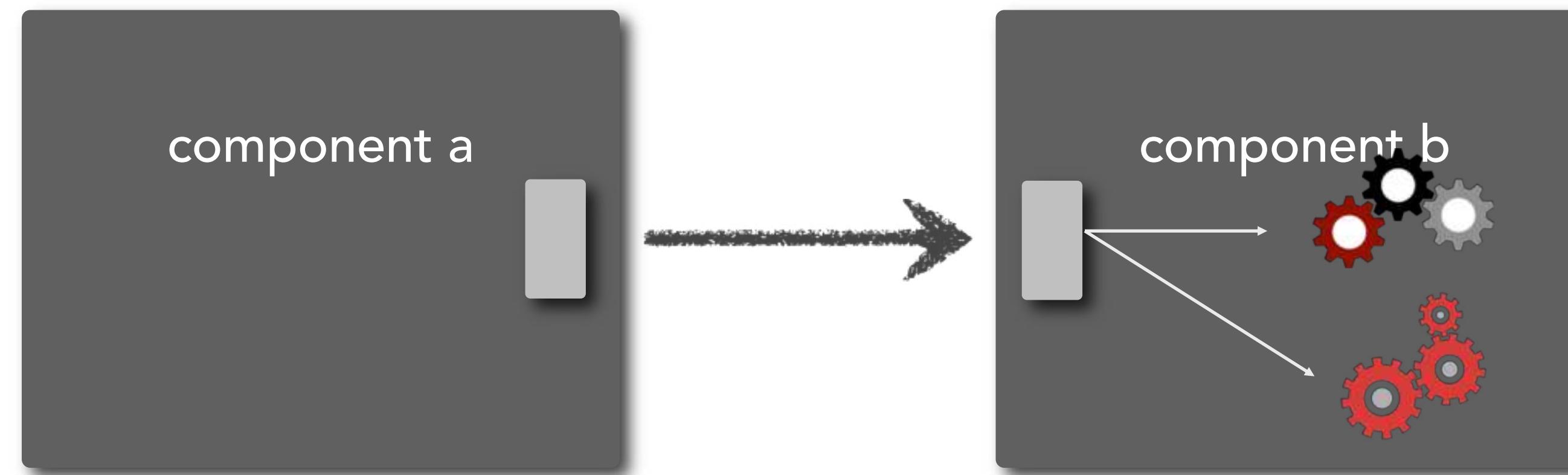
multiple components share an externally imposed protocol or data format



component coupling

control coupling

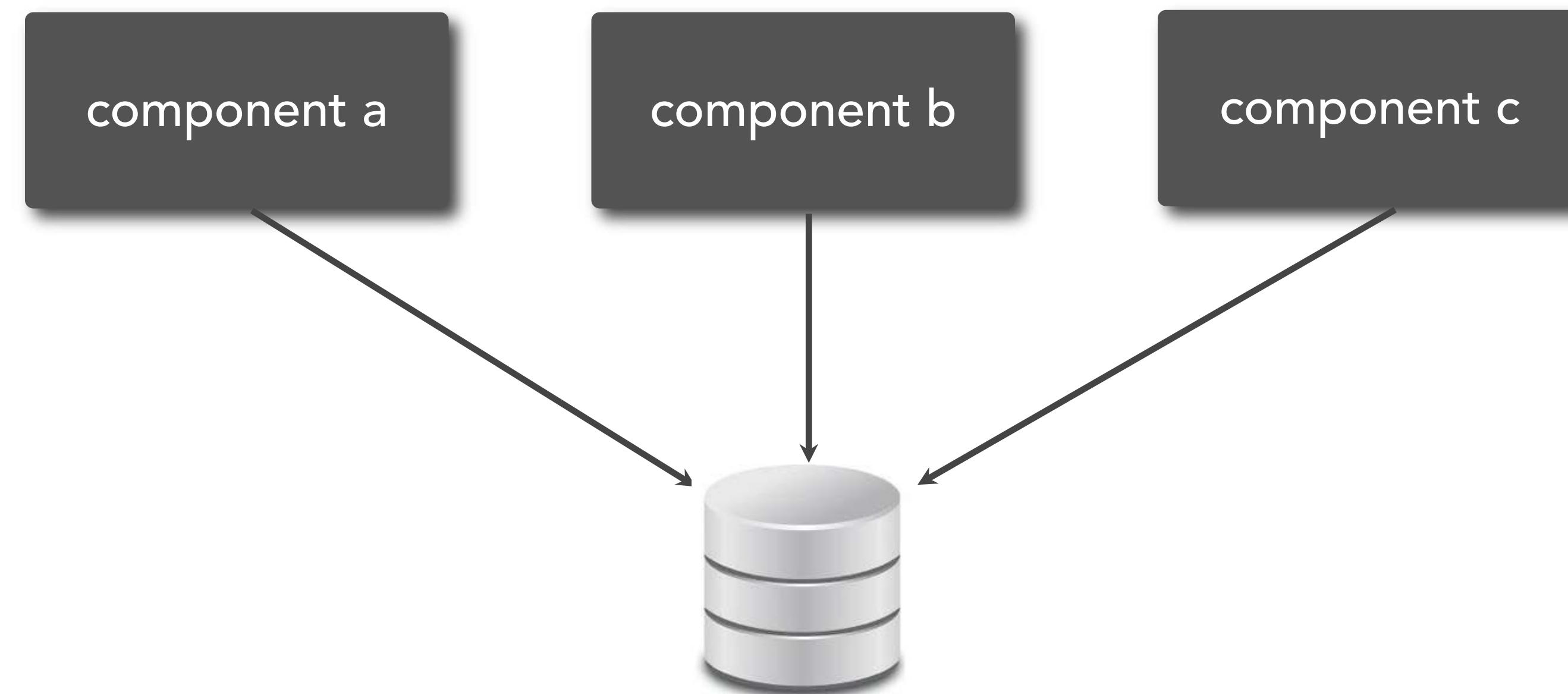
one component passes information to another component on what to do



component coupling

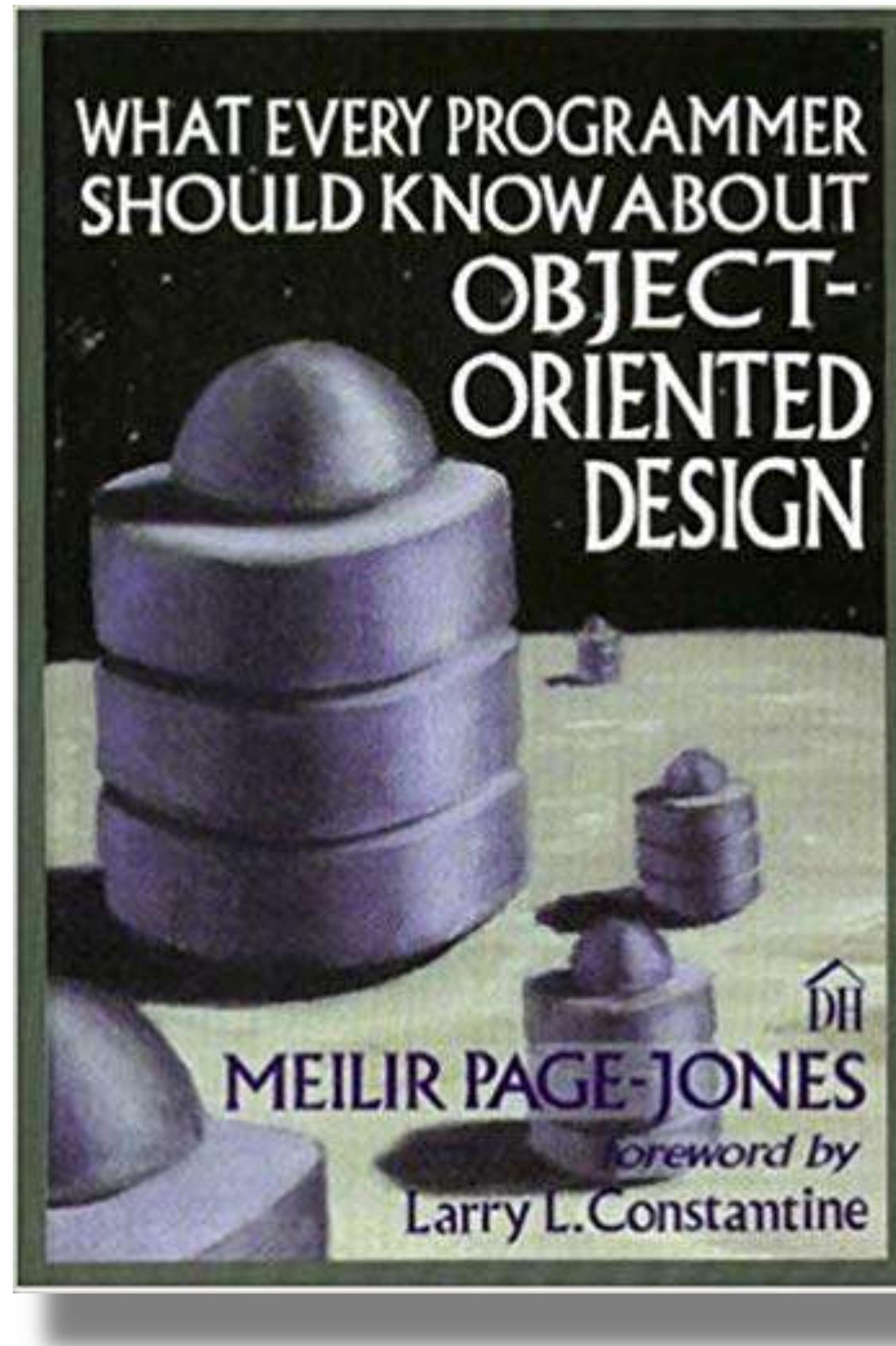
data coupling

the degree to which components are bound to a shared data context



component coupling

connescence



Two components are connascent if a change in one would require the other to be modified in order to maintain the overall correctness of the system.

*static
dynamic*

static connescence

Connascence of Name (CoN)

multiple components must agree on the
name of something

methodName

static connescence

Connascence of Type (CoT)

multiple components must agree on the type
of something

helloWorld(**String** greeting)

static connescence

Connascence of Type (CoT)

multiple components must agree on the type
of something

helloWorld(**String** greeting)

static connescence

Connascence of Meaning (CoM)

multiple components must agree on the
meaning of particular values

```
int TRUE = 1
```

```
int FALSE = 0
```

static connescence

Connascence of Position (CoP)

multiple components must agree on the
order of values

placeOrder(name, address, amount, discount)

static connescence

Connascence of Algorithm (CoA)

multiple components must agree on a
particular algorithm

authenticate_password()

dynamic connescence

Connascence of Execution (CoE)

the order of execution of multiple
components is important

dynamic connescence

Connascence of Timing (CoT)

when the timing of the execution of multiple components is important.

dynamic connescence

Connascence of Values (CoV) several values must change together

dynamic connescence

Connascence of Identity (CoI)

multiple components must reference
something

Dynamic:

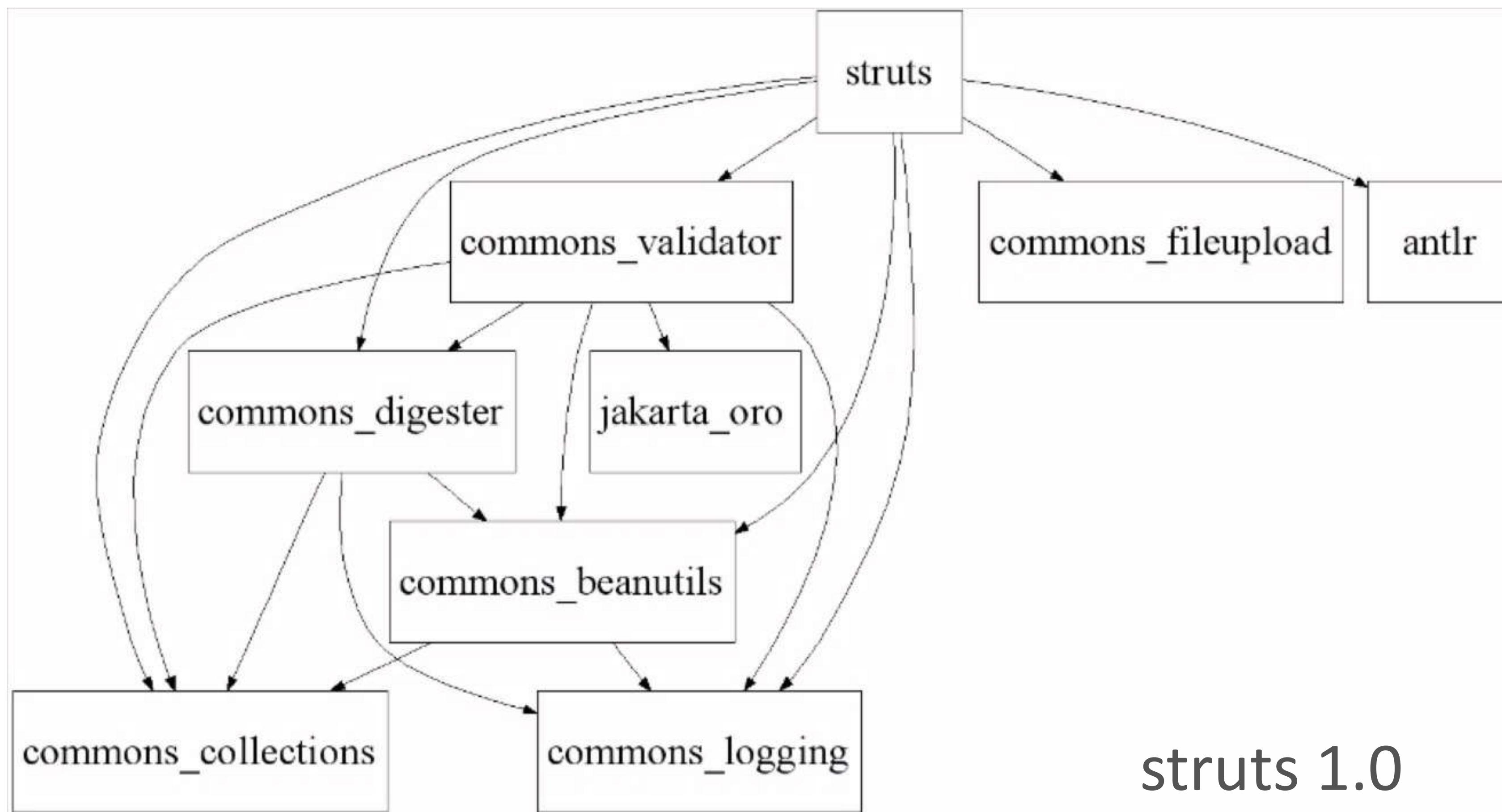


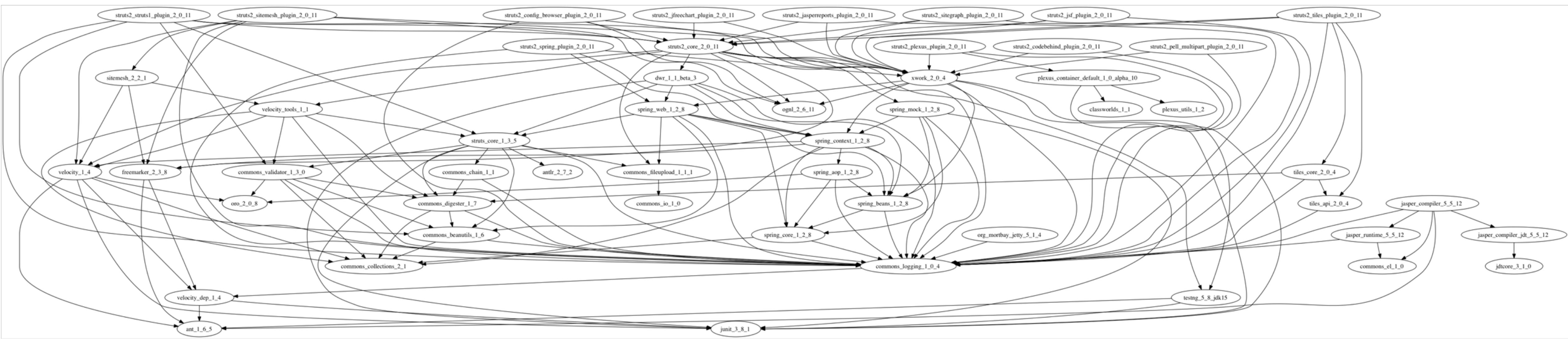
Static:



component coupling

consequences of ignoring...





component cohesion

component cohesion

the degree and manner to which the operations of a component are related to one another

component cohesion

the degree and manner to which the operations of a component are related to one another



component cohesion

the degree and manner to which the operations of a component are related to one another

customer
maintenance

- add customer
- update customer
- get customer
- notify customer
- get customer orders
- cancel customer orders

component cohesion

the degree and manner to which the operations of a component are related to one another

customer
maintenance

- add customer
- update customer
- get customer
- notify customer

order
maintenance

- get customer orders
- cancel customer orders

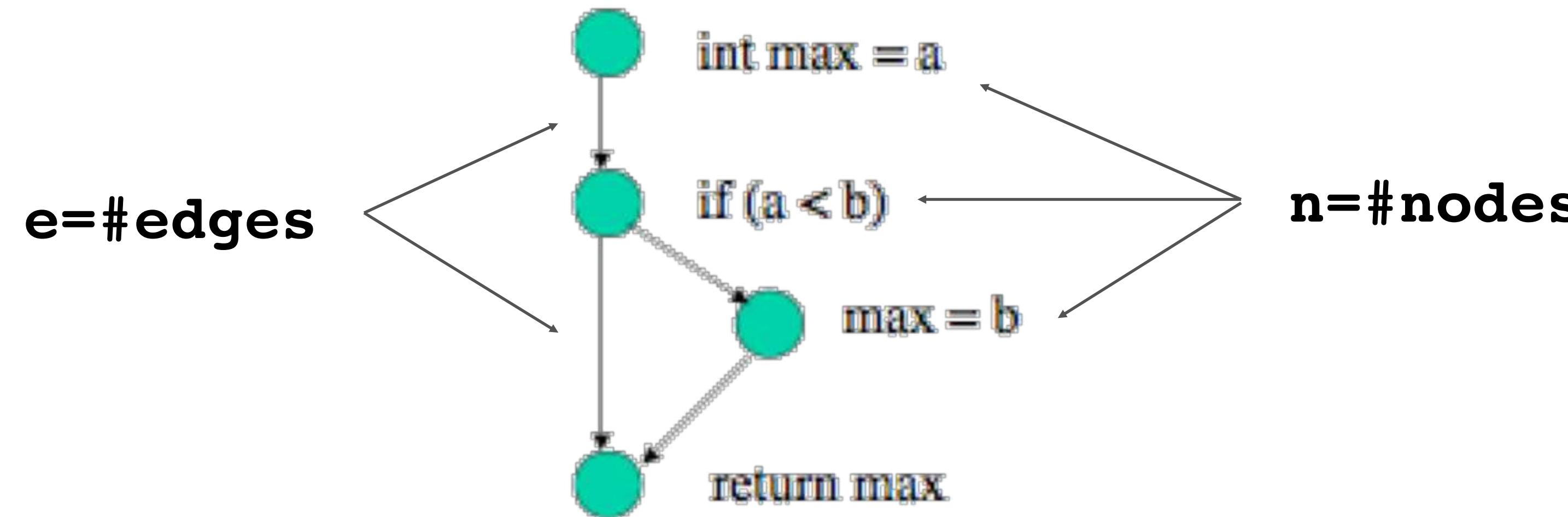


metrics

metrics and structural decay

cyclomatic complexity

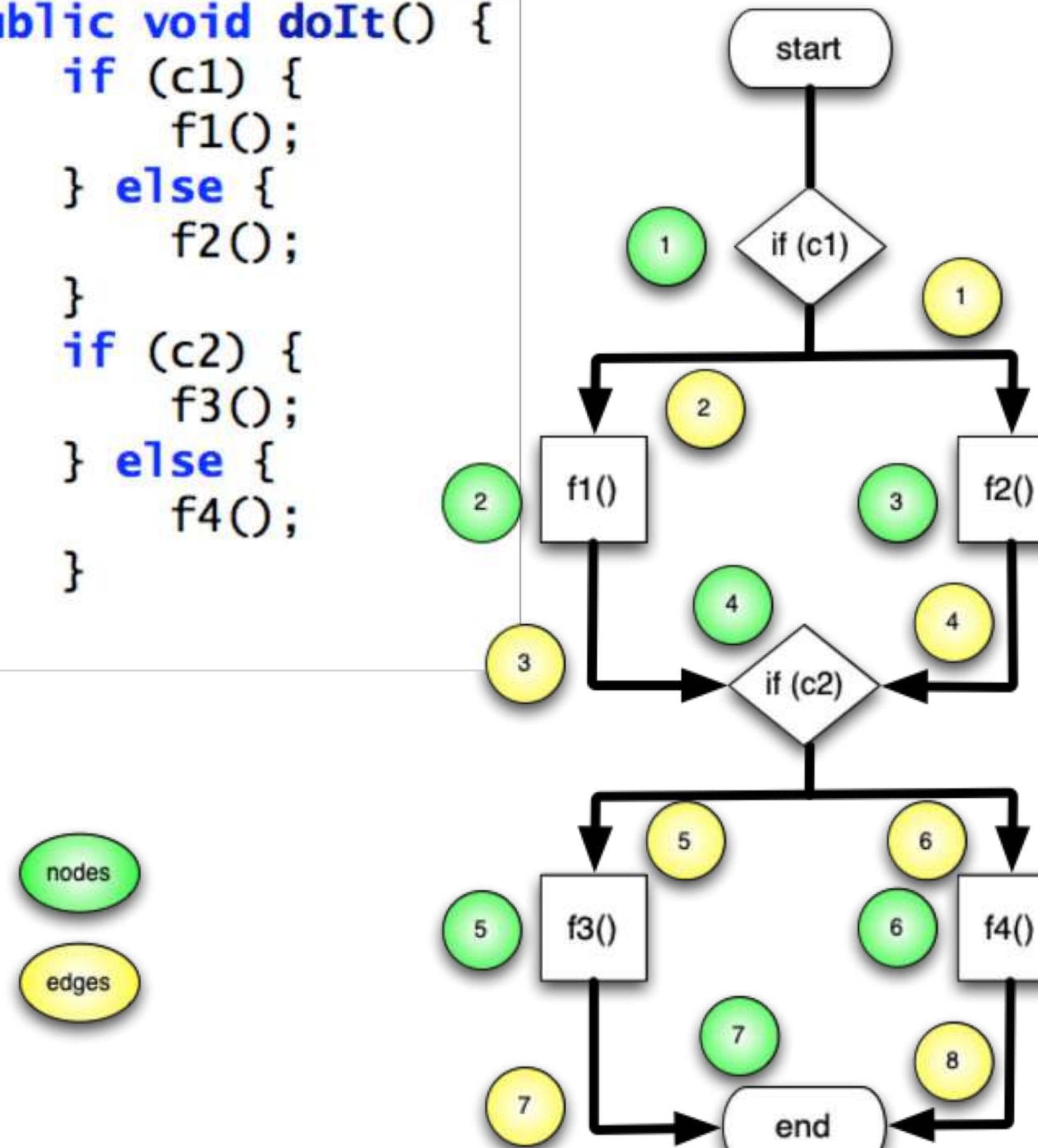
$$V(G) = e - n + 2$$



metrics and structural decay

cyclomatic complexity

```
public void doIt() {  
    if (c1) {  
        f1();  
    } else {  
        f2();  
    }  
    if (c2) {  
        f3();  
    } else {  
        f4();  
    }  
}
```



$$V(G) = e - n + 2$$

$$V(G) = 8 - 7 + 2 = 3$$

metrics and structural decay

core metrics

- ✓ number of classes per package
- ✓ number of lines of source code per package
- ✓ percent comments (range: 8-20)
- ✓ max complexity ($1 + \text{num_paths}$ thru method; range: 2-8)
- ✓ average complexity (range: 2.0 - 4.0)

metrics and structural decay

Chidamber & Kemerer Metrics

- ✓ DIT (depth of inheritance tree)
- ✓ WMC (weighted methods/class; sum of CC)
- ✓ CE (efferent coupling count)
- ✓ CA (afferent coupling count)

metrics and structural decay

Chidamber & Kemerer Metrics

✓ LCOM (Lack of Cohesion in Methods)

$$LCOM96b = \frac{1}{a} \sum_{j=1}^a m - \mu(Aj)$$

metrics and structural decay

Chidamber & Kemerer Metrics

✓ LCOM (Lack of Cohesion in Methods)

The LCOM is a count of the number of method pairs whose similarity is 0 (i.e. $\sigma()$ is a null set) minus the count of method pairs whose similarity is not zero.

metrics and structural decay

Chidamber & Kemerer Metrics

✓ LCOM (Lack of Cohesion in Methods)

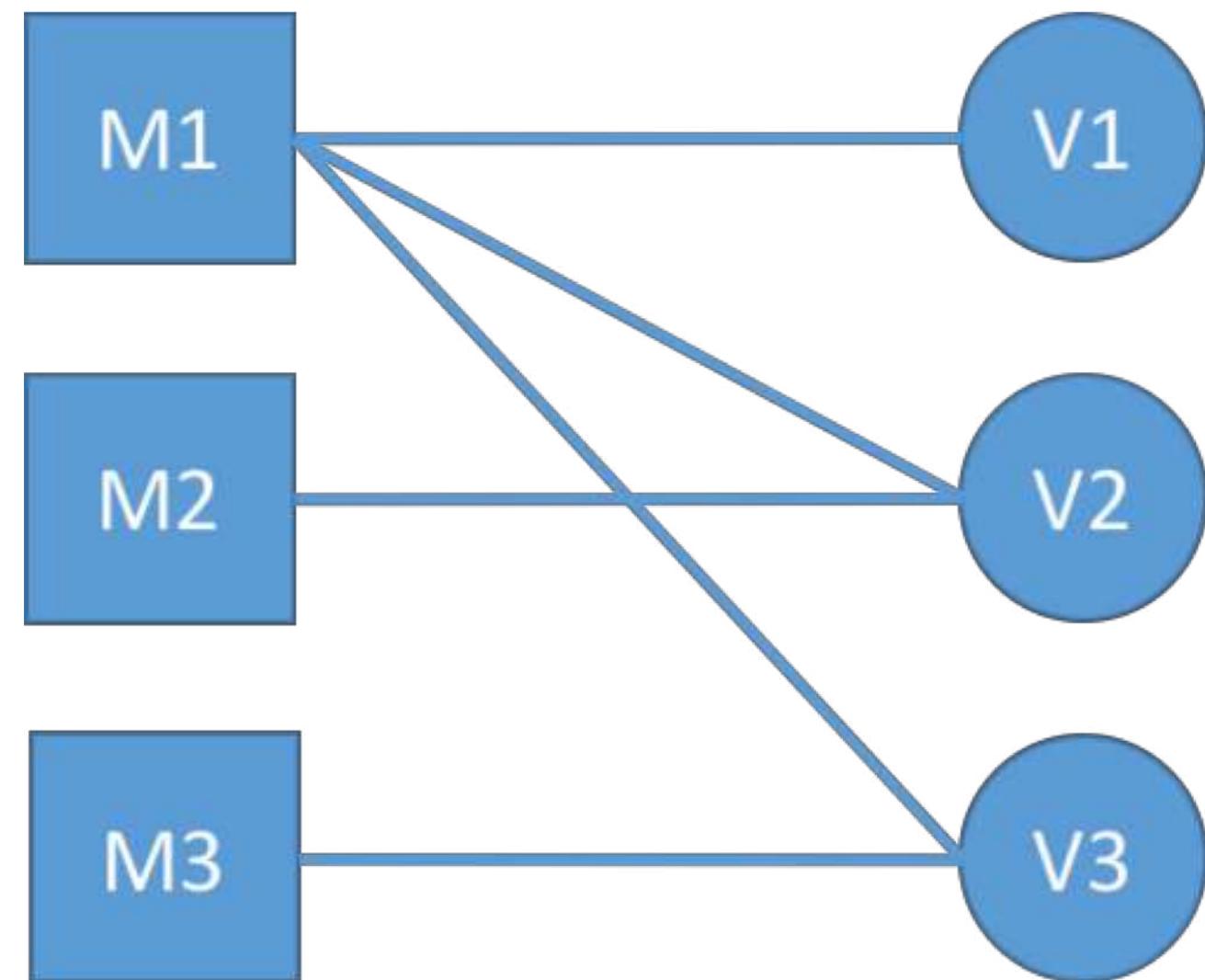
The LCOM is a count of the number of method pairs whose similarity is 0 (i.e. $\sigma()$ is a null set) minus the count of method pairs whose similarity is not zero.

The sum of sets of methods not shared via sharing fields.

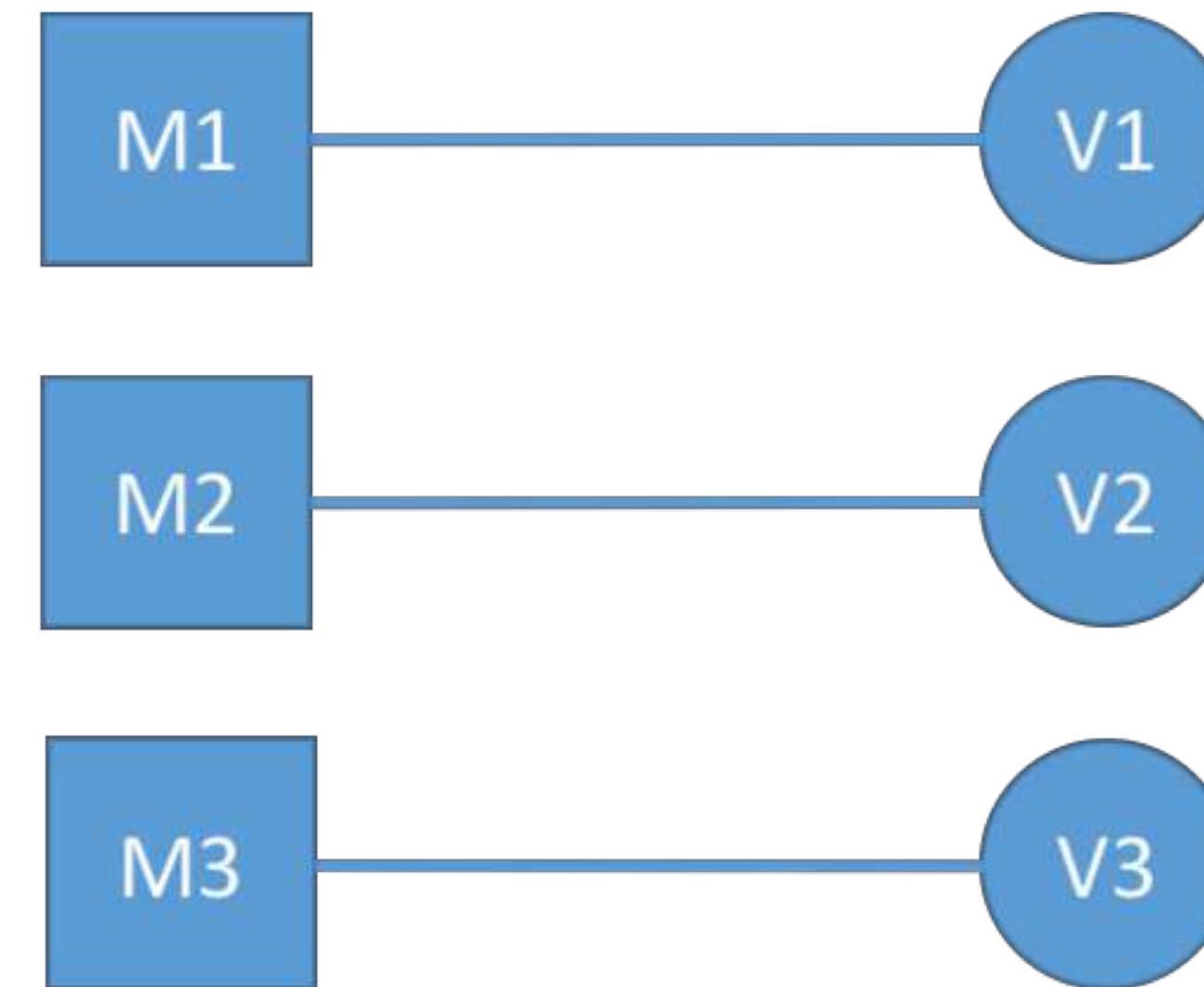
metrics and structural decay

Chidamber & Kemerer Metrics

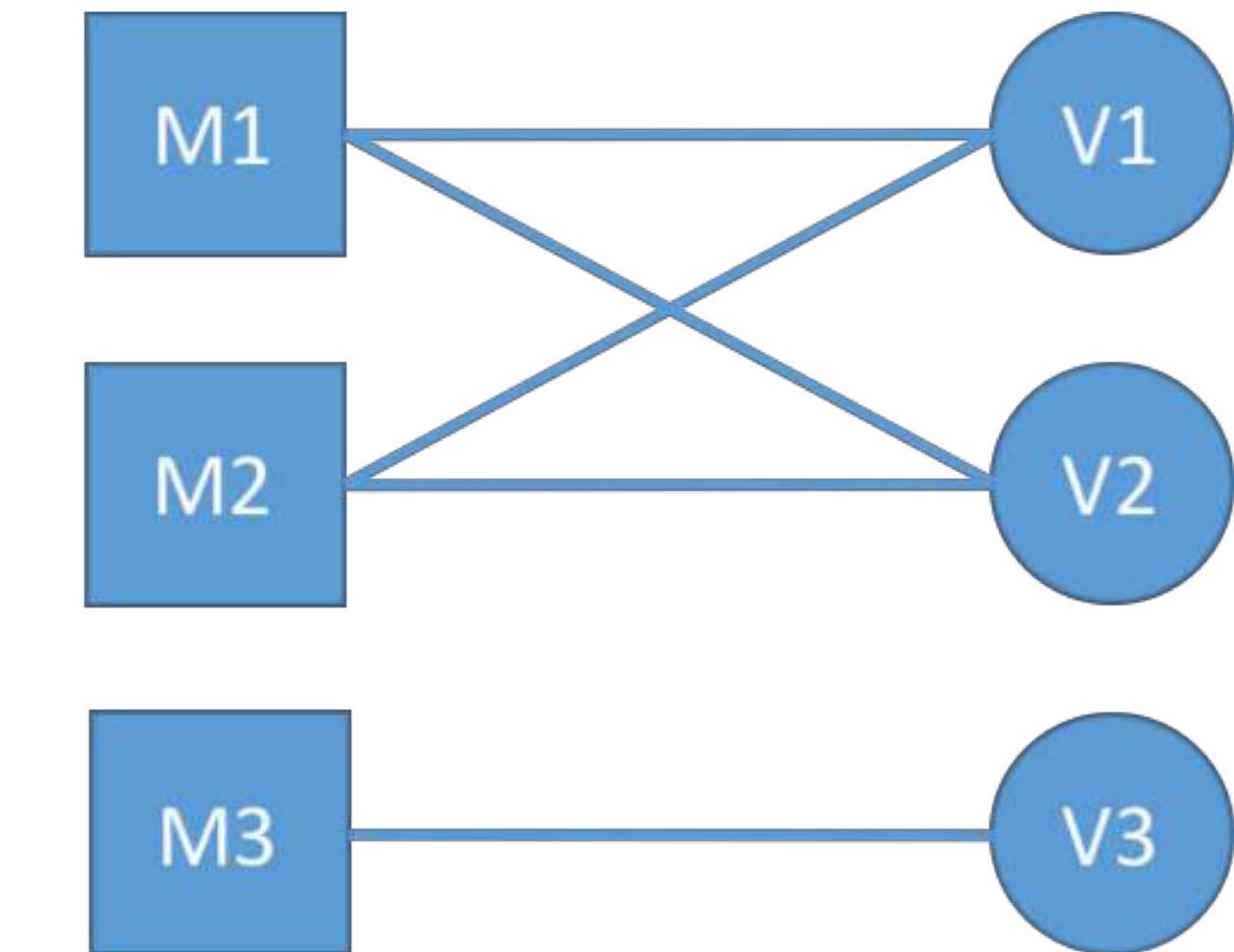
✓ LCOM (Lack of Cohesion in Methods)



(A)



(B)

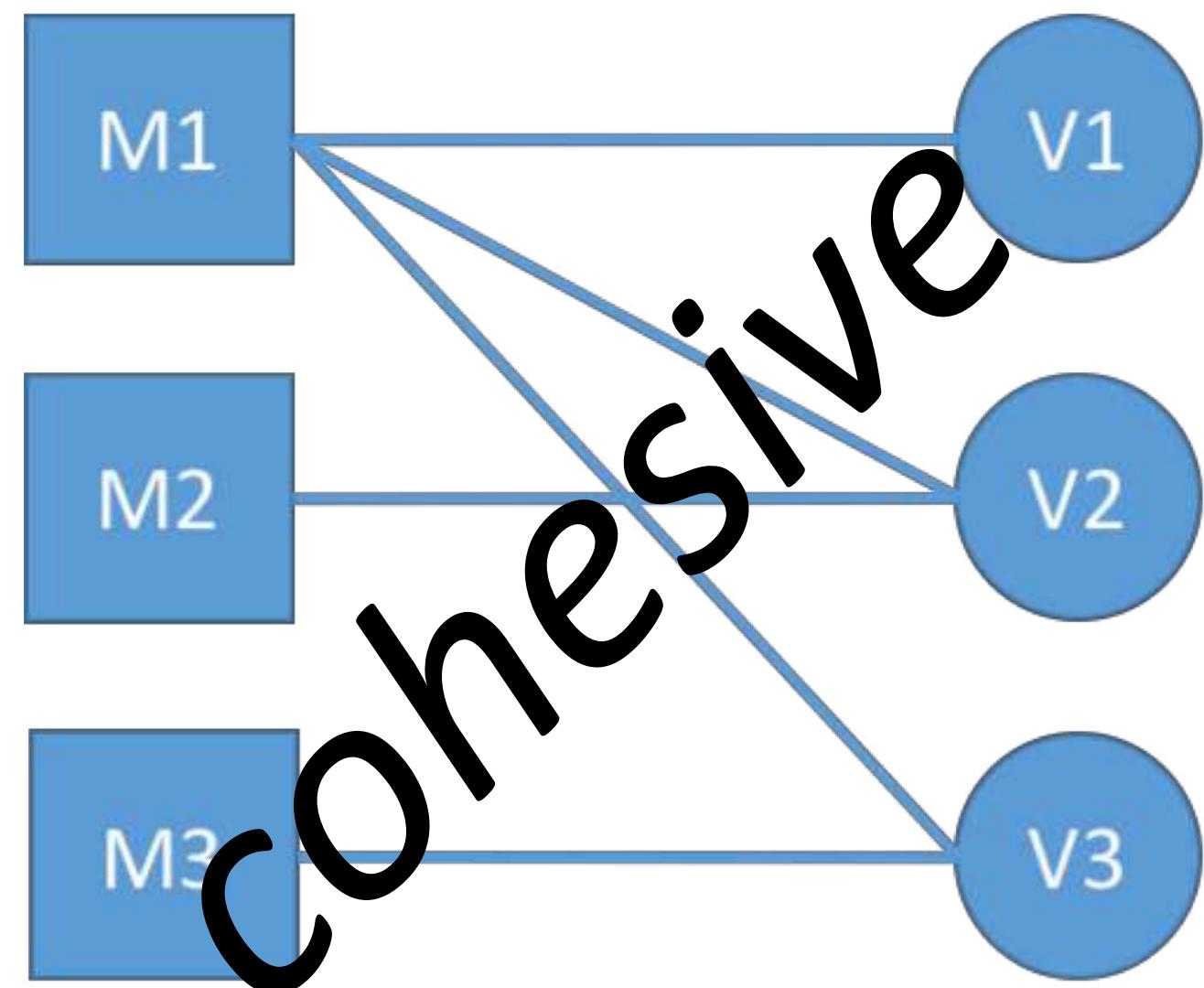


(C)

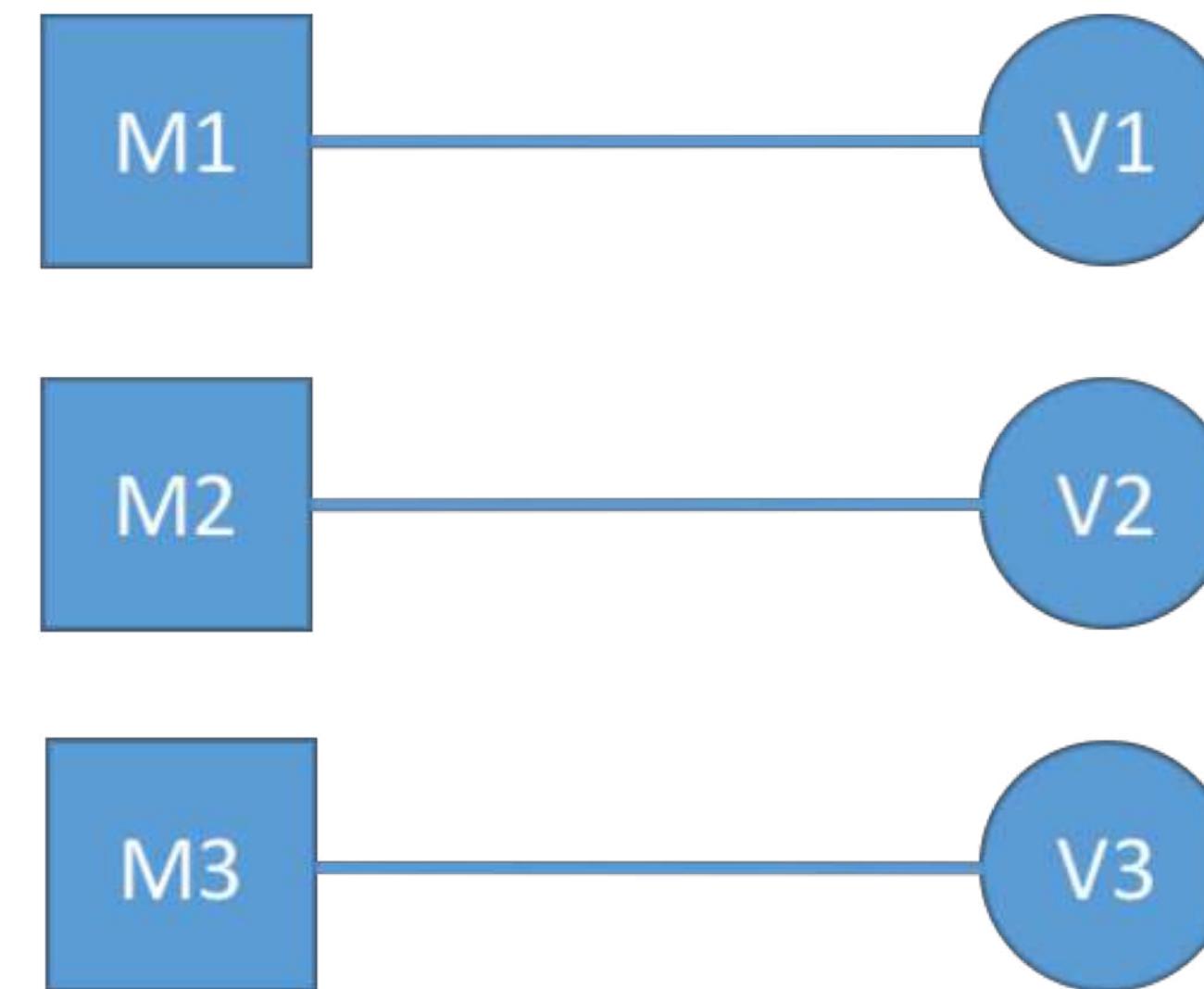
metrics and structural decay

Chidamber & Kemerer Metrics

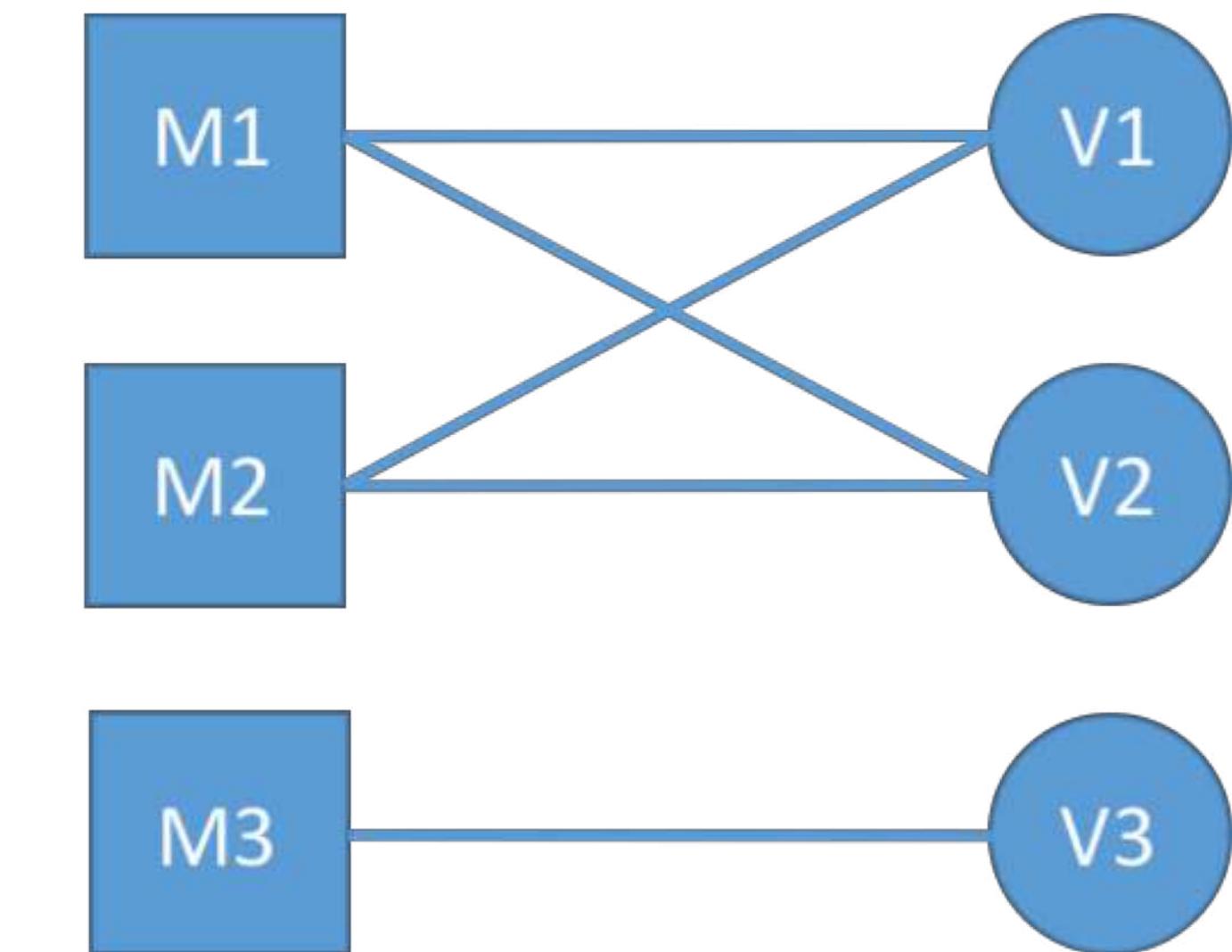
✓ LCOM (Lack of Cohesion in Methods)



(A)



(B)

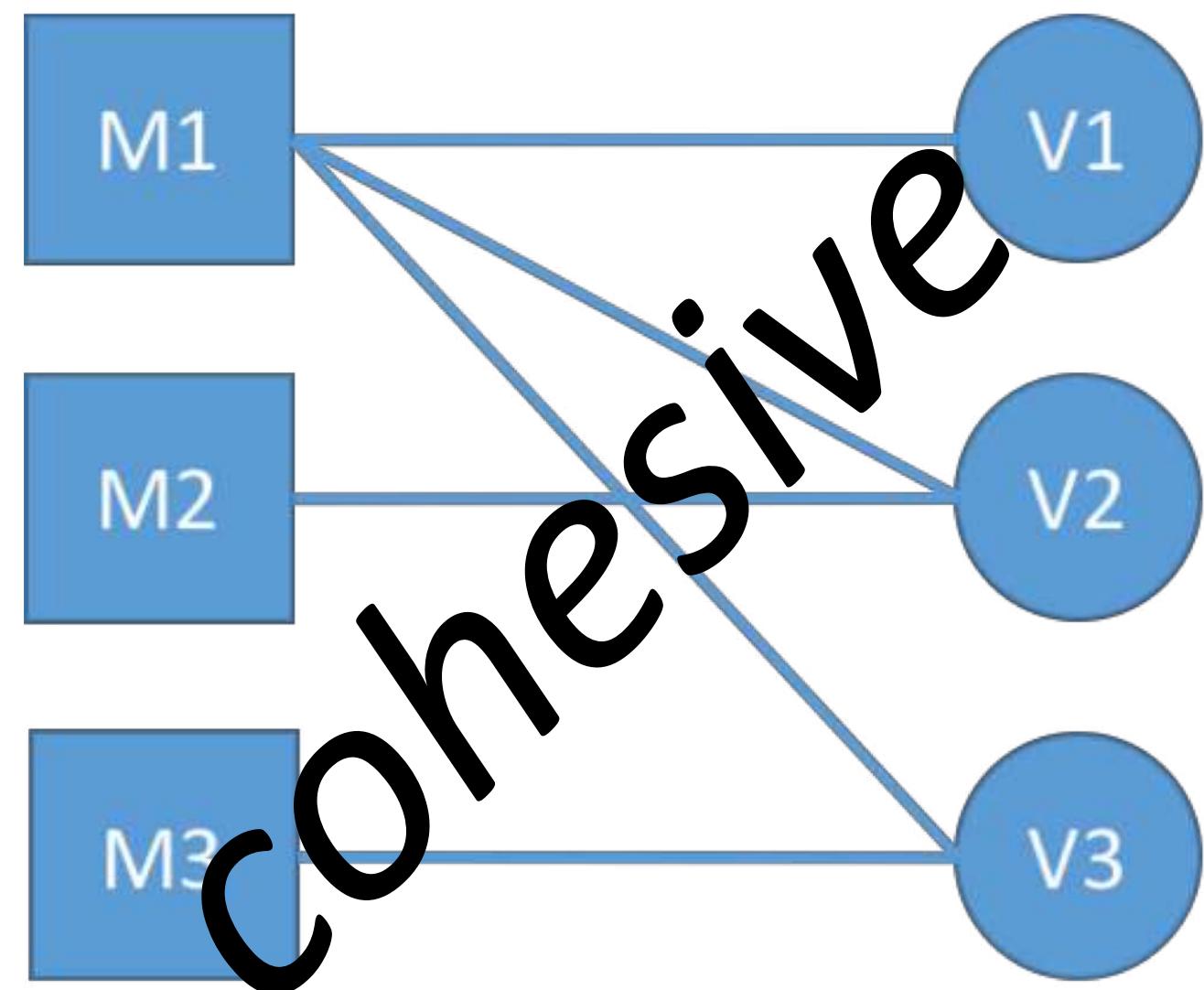


(C)

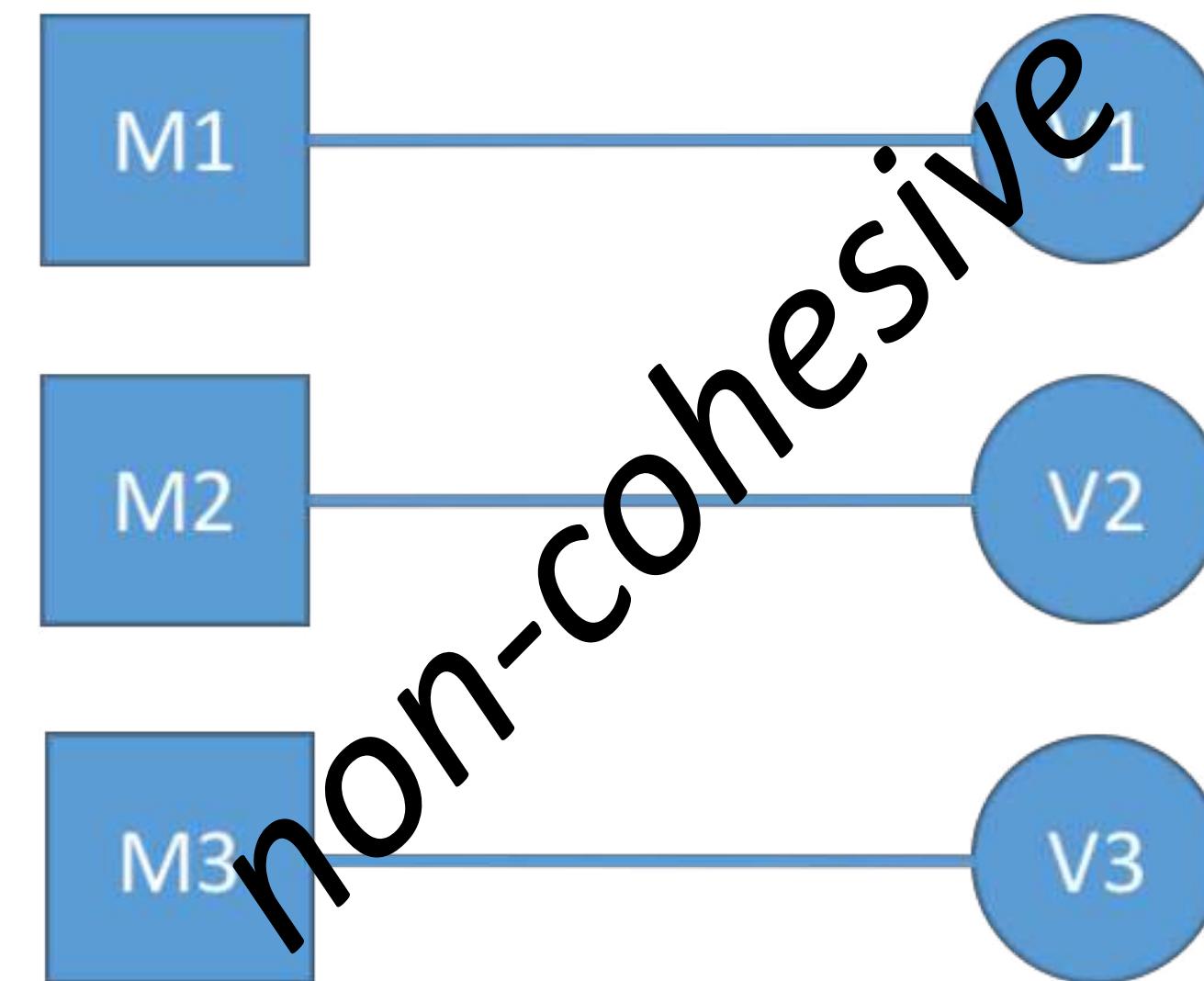
metrics and structural decay

Chidamber & Kemerer Metrics

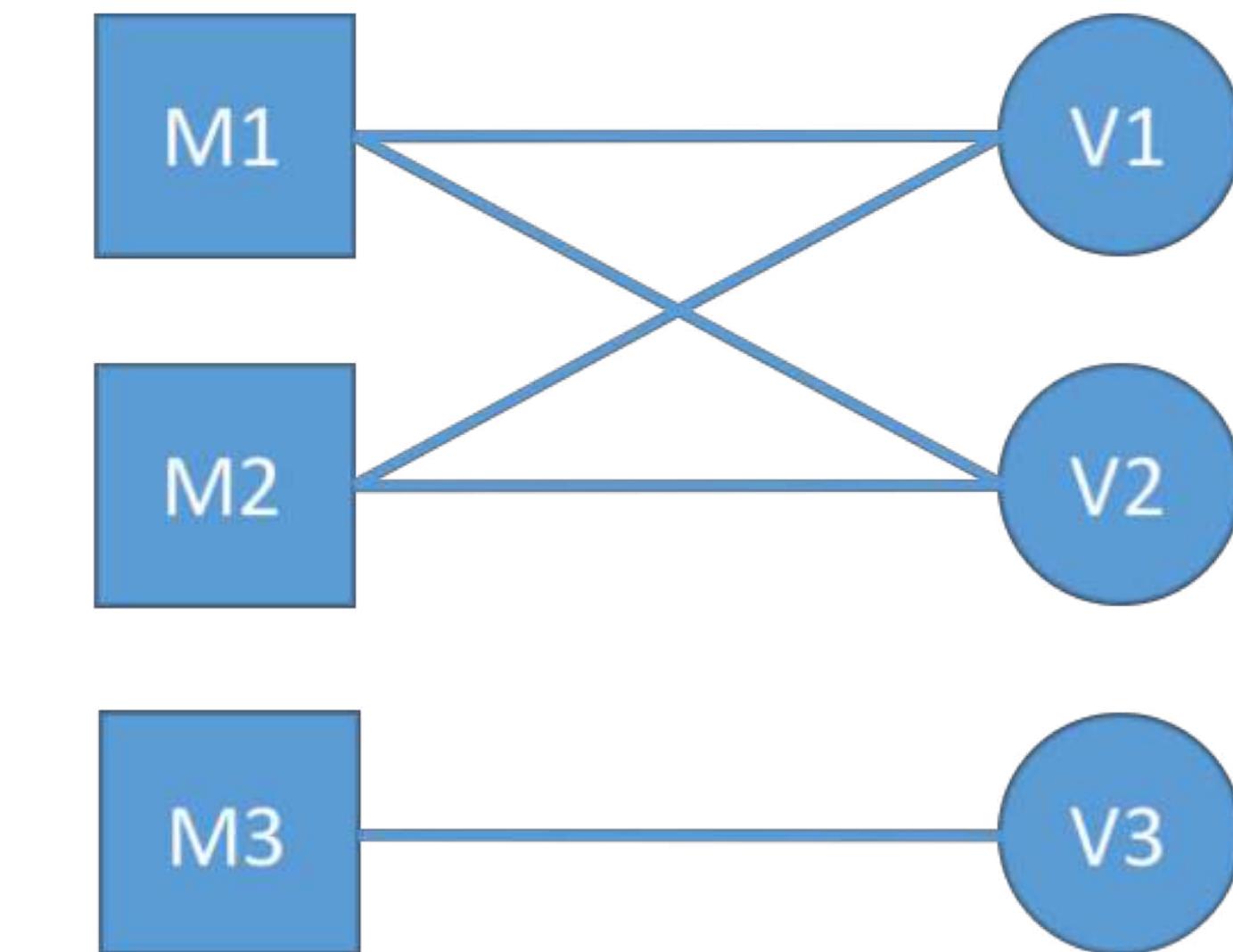
✓ LCOM (Lack of Cohesion in Methods)



(A)



(B)

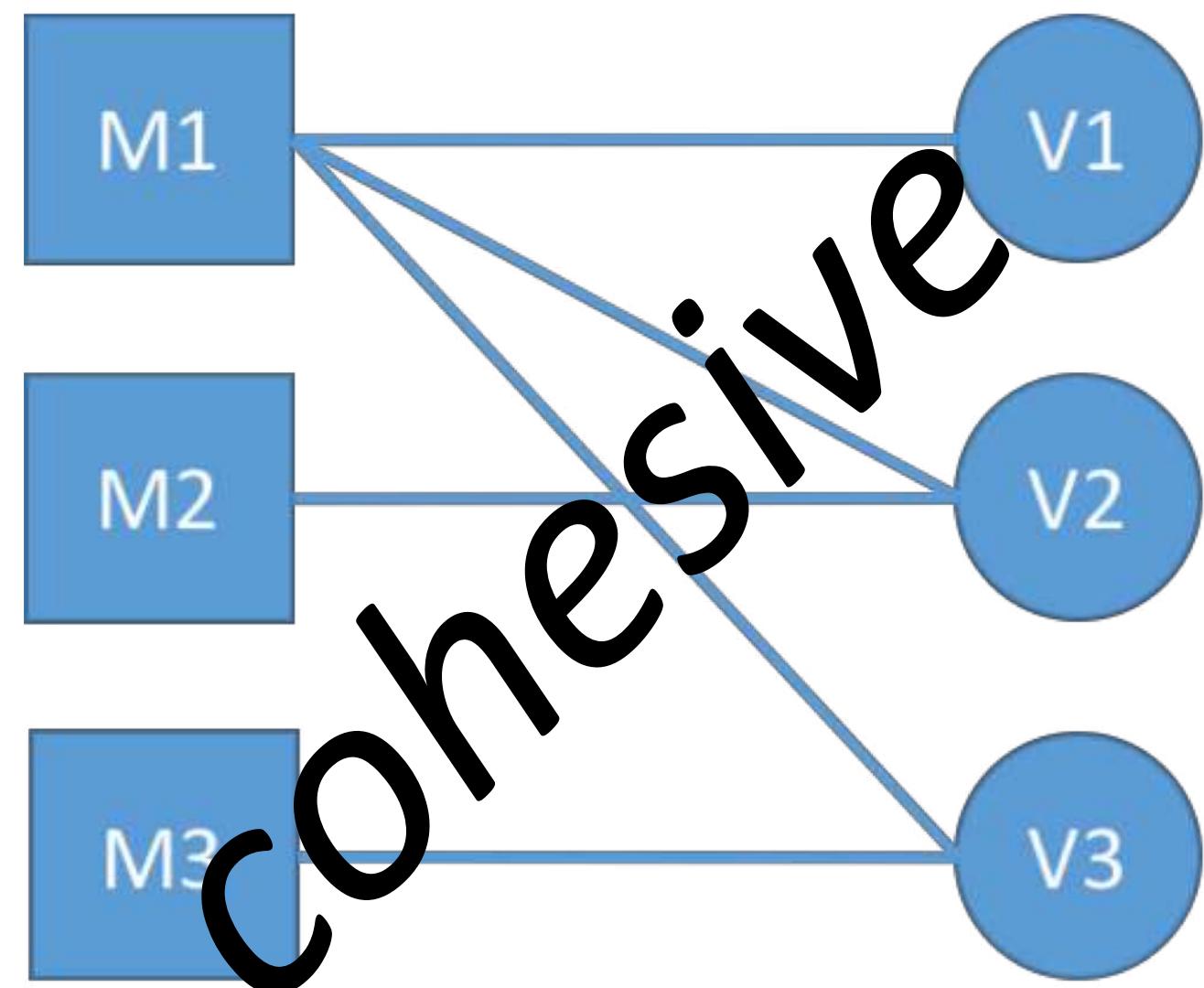


(C)

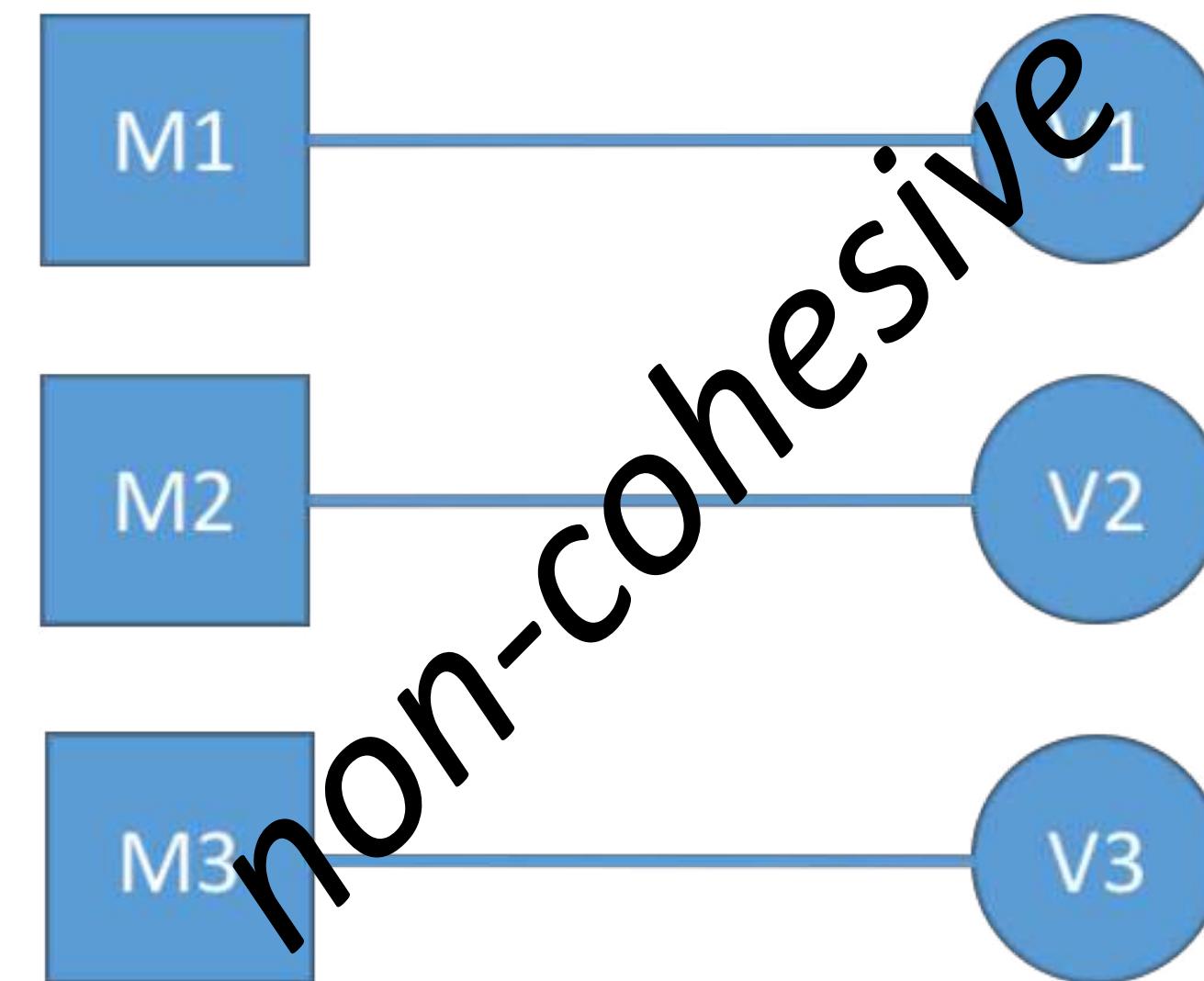
metrics and structural decay

Chidamber & Kemerer Metrics

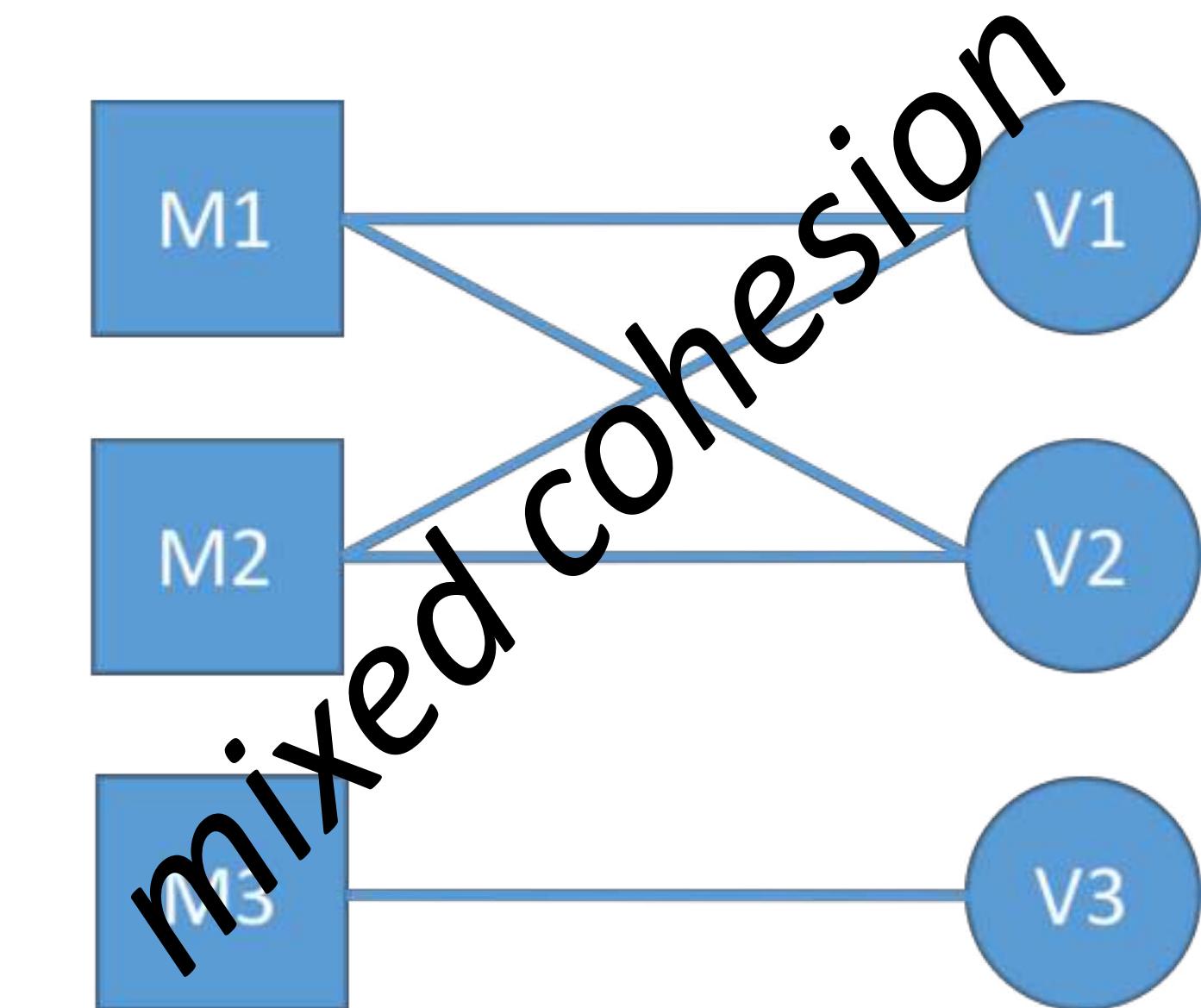
✓ LCOM (Lack of Cohesion in Methods)



(A)



(B)



(C)

cohesive

non-cohesive

mixed cohesion

metrics ∩ architecture characteristics

core metrics

- ✓ number of classes per package
- ✓ number of lines of source code per package
- ✓ percent comments (range: 8-20)
- ✓ max complexity ($1 + \text{num_paths}$ thru method; range: 2-8)
- ✓ average complexity (range: 2.0 - 4.0)

metrics \cap architecture characteristics

Chidamber & Kemerer Metrics

✓ DIT (depth of inheritance tree)

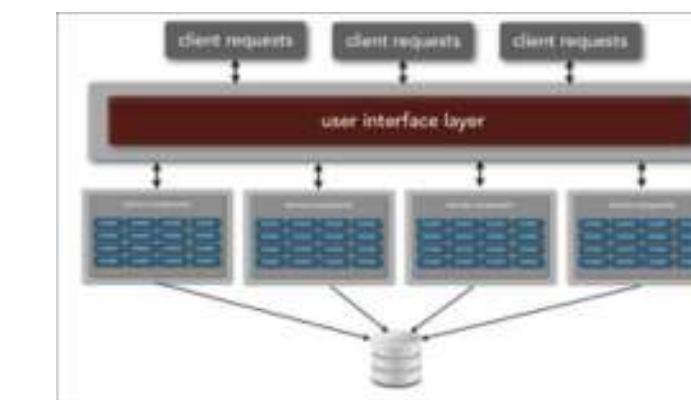
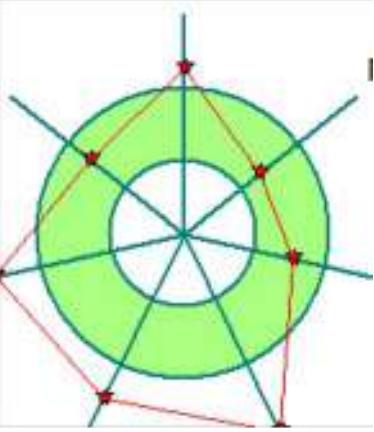
✓ WMC (weighted methods/class; sum of CC)

✓ CE (efferent coupling count)

✓ CA (afferent coupling count)

metrics ∩ architecture characteristics

architecture characteristics mapping



component size

complexity / WMC

coupling (CE, CA, CT)

inheritance depth (DIT)

percent comments

modularity

maintainability

testability

availability

deployment

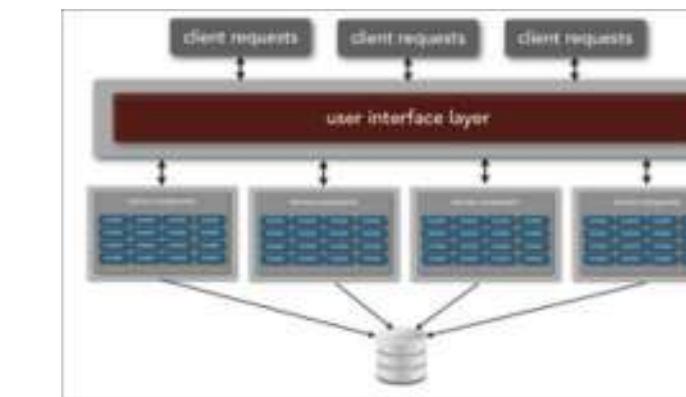
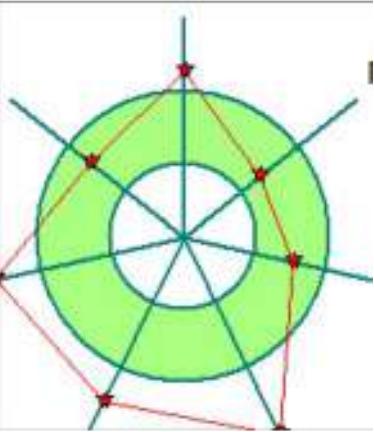
reliability

scalability

evolutionary / migration

metrics ∩ architecture characteristics

architecture characteristics mapping



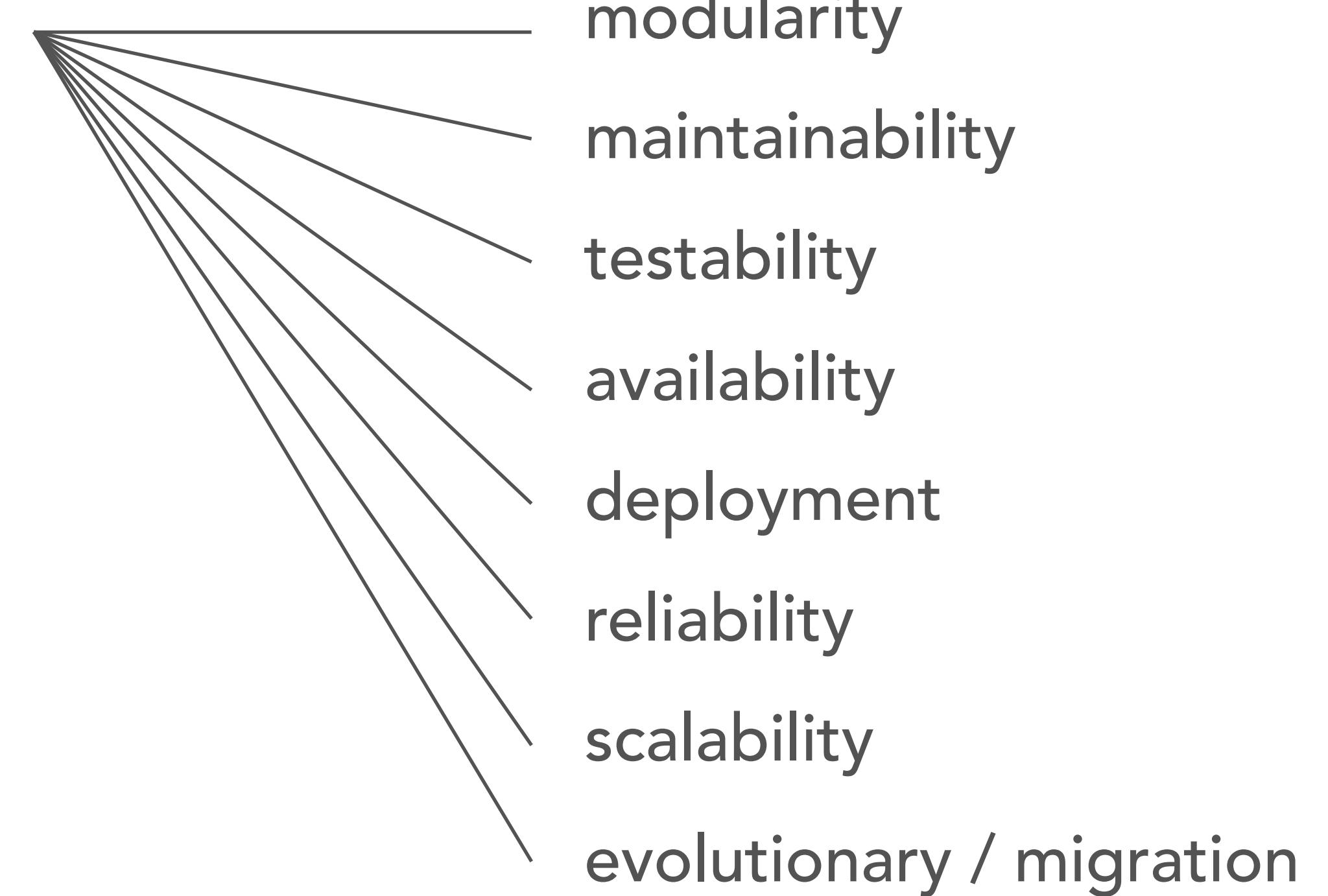
component size

complexity / WMC

coupling (CE, CA, CT)

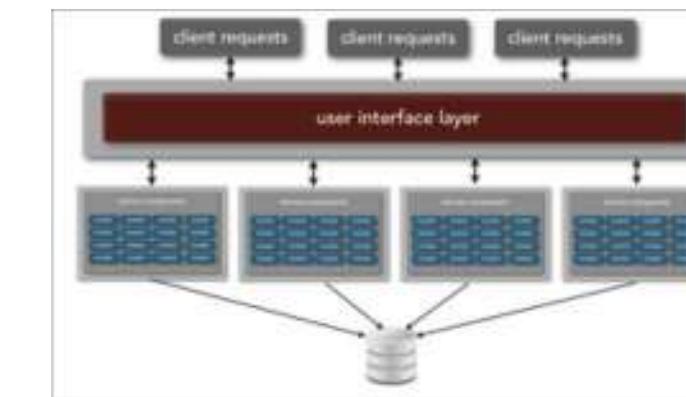
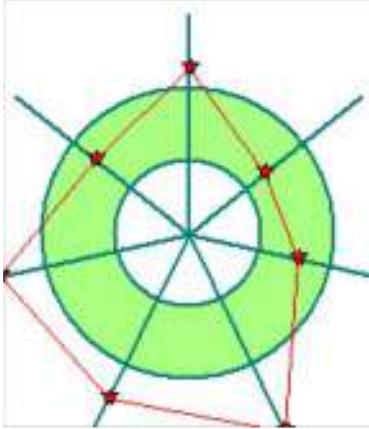
inheritance depth (DIT)

percent comments



metrics \cap architecture characteristics

architecture characteristics mapping



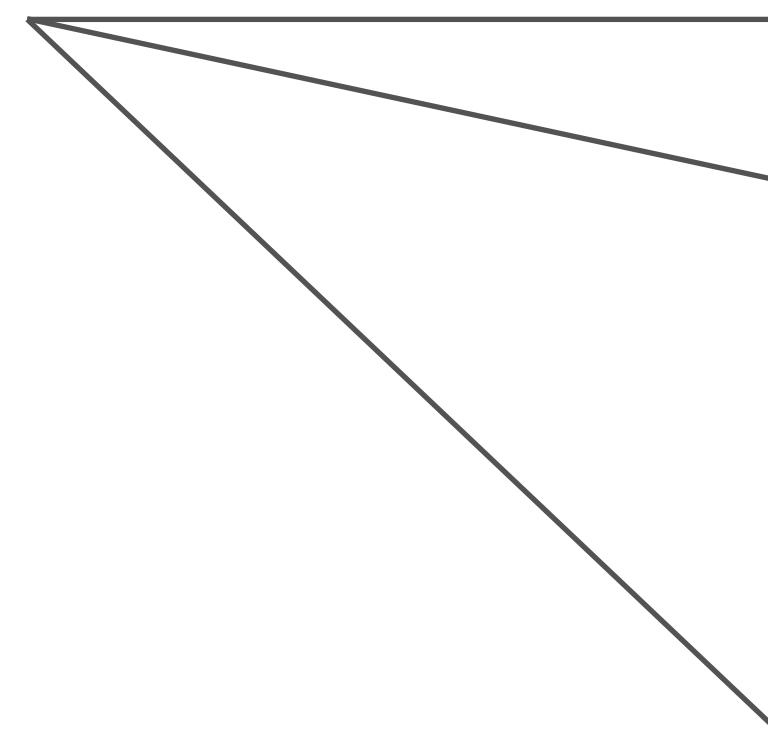
component size

complexity / WMC

coupling (CE, CA, CT)

inheritance depth (DIT)

percent comments



modularity

maintainability

testability

availability

deployment

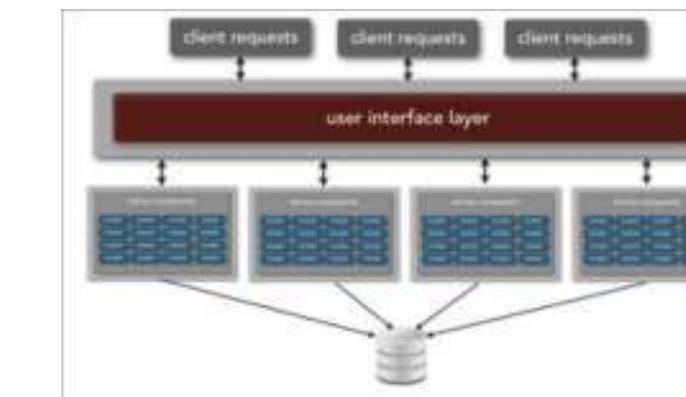
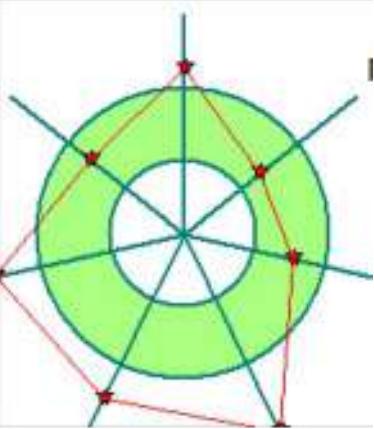
reliability

scalability

evolutionary / migration

metrics ∩ architecture characteristics

architecture characteristics mapping



component size

complexity / WMC

coupling (CE, CA, CT)

inheritance depth (DIT)

percent comments

modularity

maintainability

testability

availability

deployment

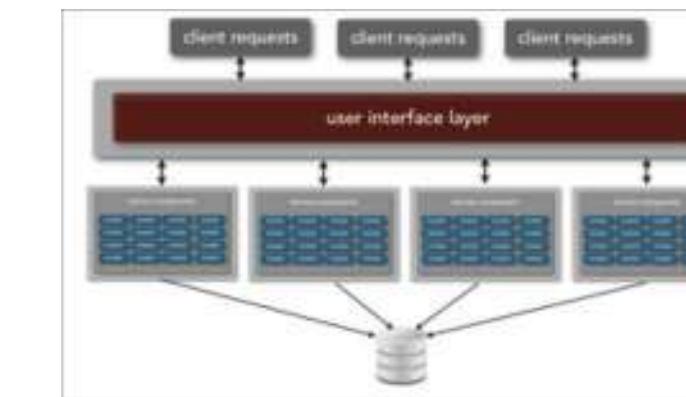
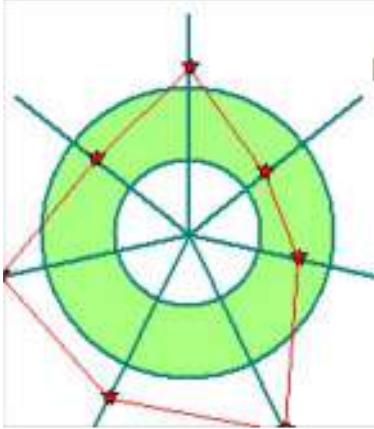
reliability

scalability

evolutionary / migration

metrics ∩ architecture characteristics

architecture characteristics mapping



component size

complexity / WMC

coupling (CE, CA, CT)

inheritance depth (DIT)

percent comments

modularity

maintainability

testability

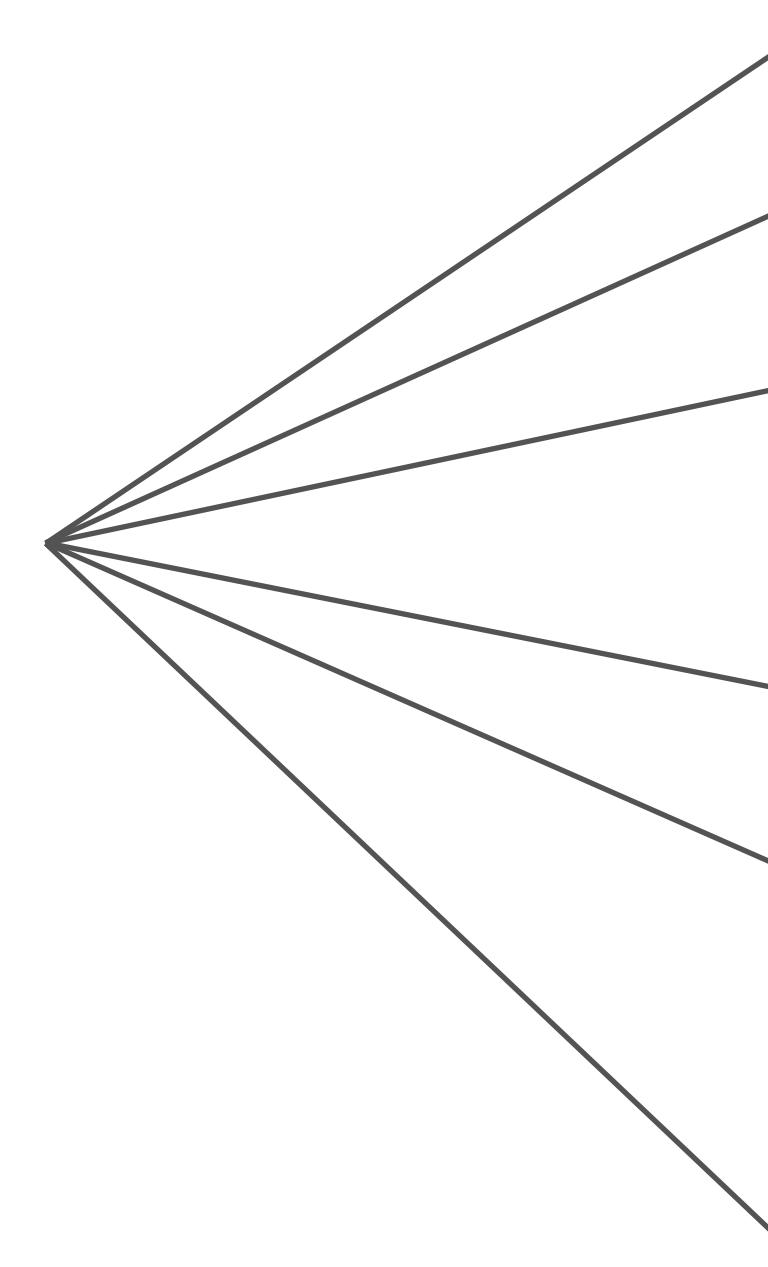
availability

deployment

reliability

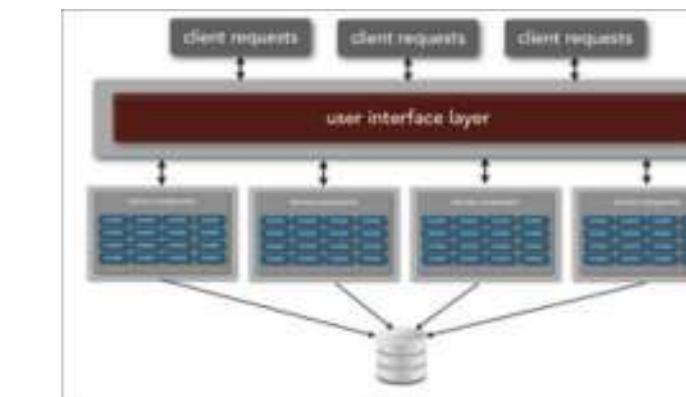
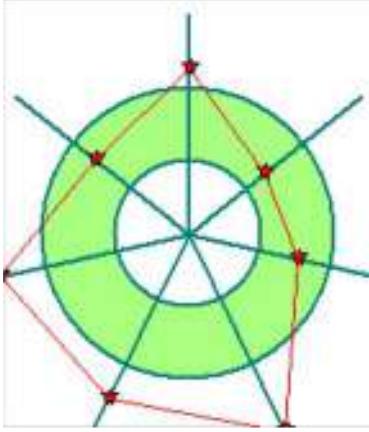
scalability

evolutionary / migration



metrics \cap architecture characteristics

architecture characteristics mapping



component size

complexity / WMC

coupling (CE, CA, CT)

inheritance depth (DIT)

percent comments

modularity

maintainability

testability

availability

deployment

reliability

scalability

evolutionary / migration

continuous delivery ∩ architecture

good engineering practices help preserve important architectural characteristics

continuous delivery ∩ architecture

good engineering practices help preserve important architectural characteristics

awareness of engineering informs architectural decisions

continuous delivery ∩ architecture

good engineering practices help preserve important architectural characteristics

awareness of engineering informs architectural decisions

better component isolation allows for faster cycle time

Question:

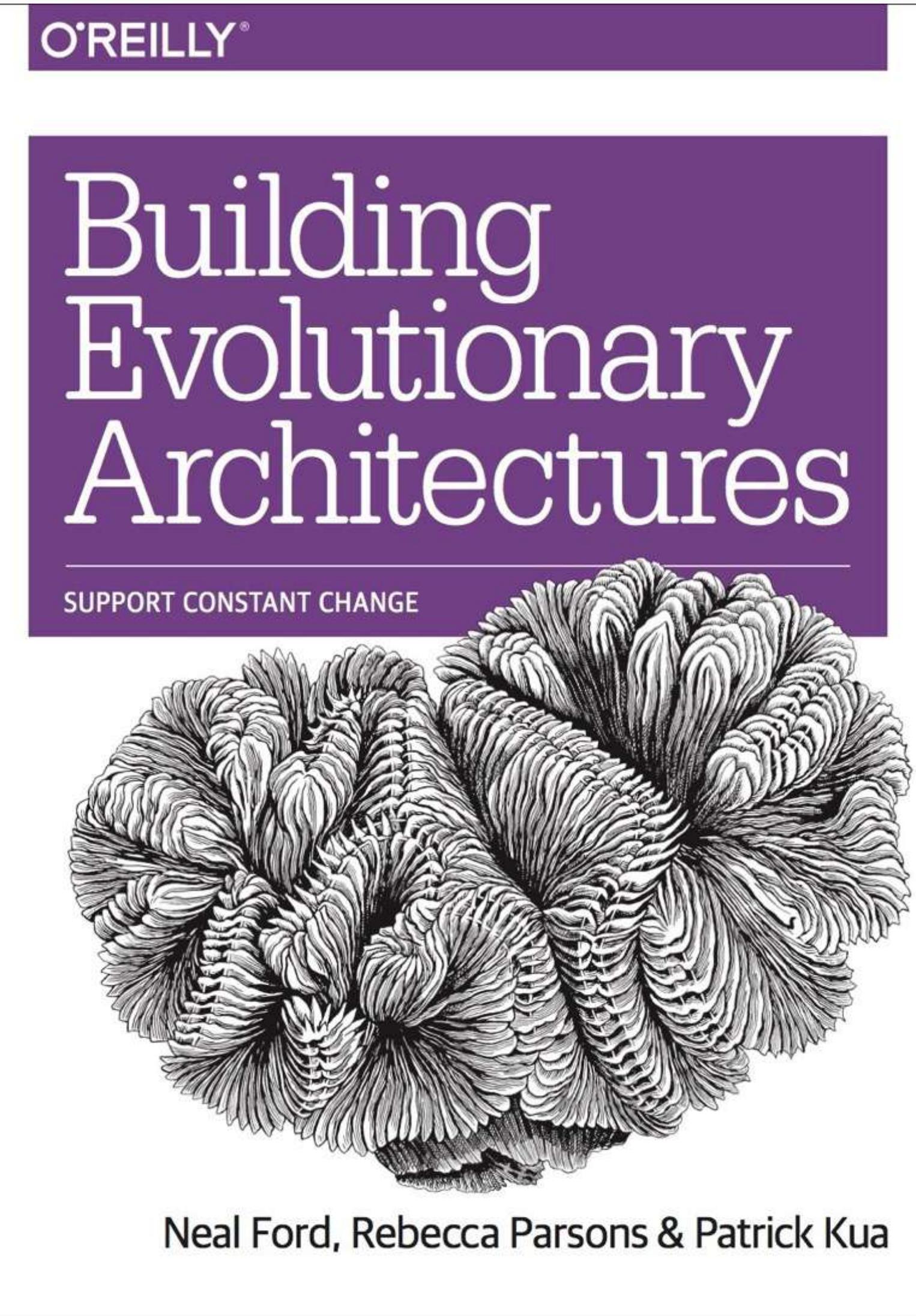
How can we combine things like metrics and other architecture checks with continuous delivery/deployment?

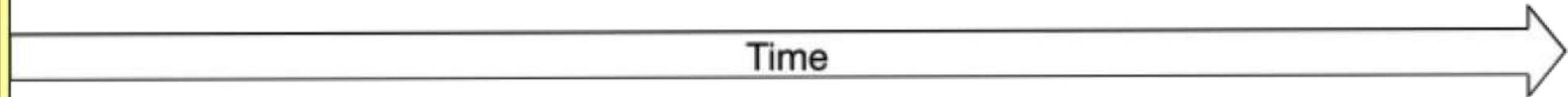
Question:

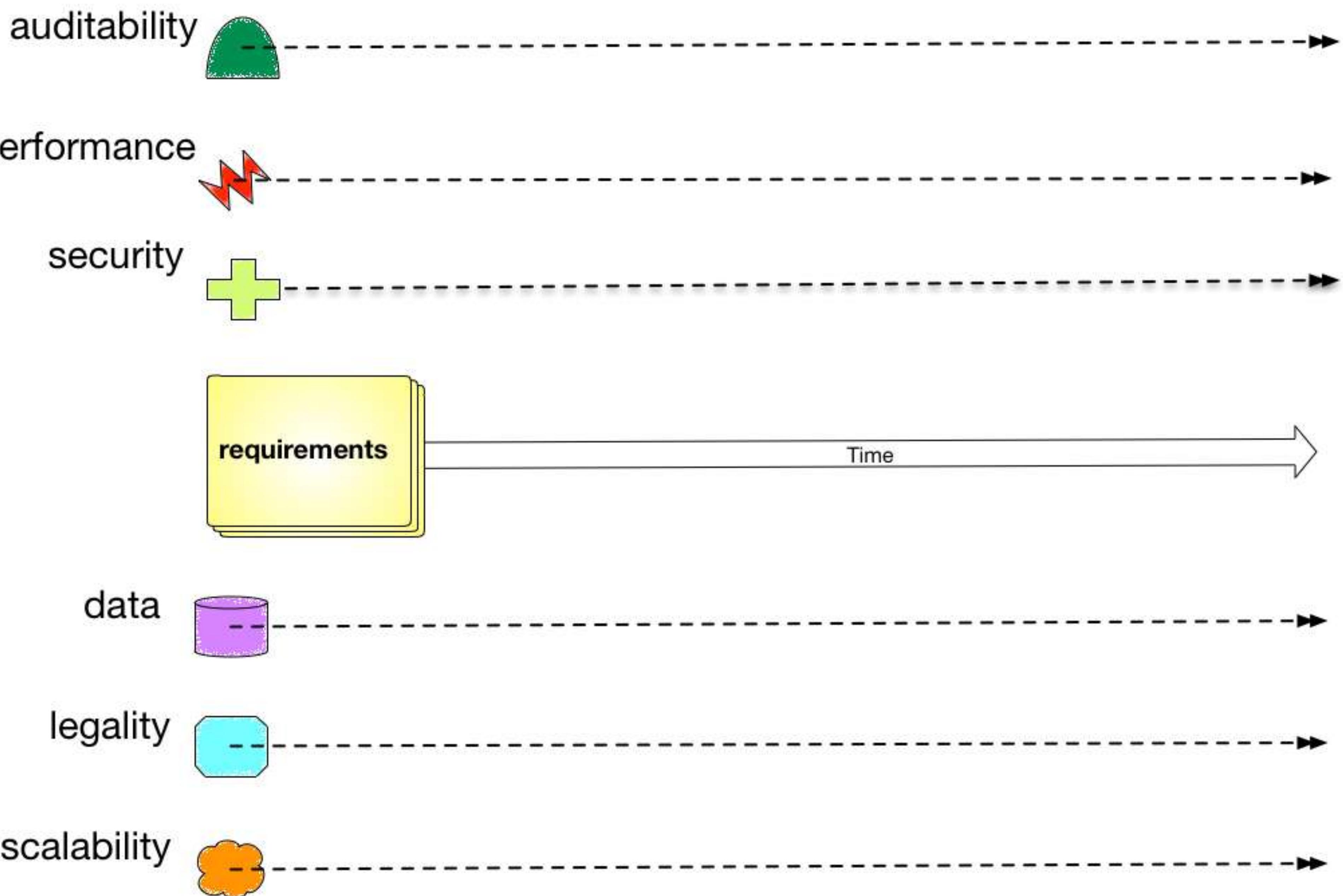
What is the most difficult part of enabling continuous
deployment at your organization?

— fitness functions

— incremental change







accessibility
accountability

accuracy
adaptability
administrability
affordability
agility

auditability
autonomy

availability
compatibility

composability
configurability

correctness
credibility

customizability
debugability

degradability
determinability

demonstrability
dependability

deployability
discoverability

distributability
durability

effectiveness
efficiency

reliability
extensibility

failure transparency
fault-tolerance

fidelity
flexibility
inspectability
installability
integrity

interchangeability
interoperability

learnability
maintainability
manageability
mobility

modifiability
modularity
operability
orthogonality
portability
precision
predictability

process capabilities
productivity
provability
recoverability
relevance

repeatability
reproducibility

resilience
responsiveness
reusability
business
safety
scrability
seamlessness
self-sustainability

serviceability
supportability
securability
simplicity
stability

standards compliance
survivability
sustainability
tailorability
testability
timeliness
traceability
transparency
ubiquity

understandability
upgradability
usability

Sample Spinning

accessibility
accountability
accuracy
adaptability
administrability
affordability
agility
auditability

reliability
extensibility
failure transparency
fault-tolerance
fidelity
flexibility
inspectability
installability

repeatability
reproducibility
resilience
responsiveness
reusability
robustness
safety
scalability

evolvability

debugability
degradability
determinability
demonstrability
dependability
deployability
discoverability
distributability
durability
effectiveness
efficiency

modularity
operability
orthogonality
portability
precision
predictability
process capabilities
productivity
provability
recoverability
relevance

survivability
sustainability
tailorability
testability
timeliness
traceability
transparency
ubiquity
understandability
upgradability
usability

accessibility

accountability

accuracy

adaptability

administrability

affordability

agility

auditability

reliability

extensibility

failure transparency

fault-tolerance

fidelity

flexibility

inspectability

installability

repeatability
reproducibility

resilience

responsiveness

reusability

robustness

safety

scalability

evolvability

debugability

degradability

determinability

demonstrability

dependability

deployability

discoverability

distributability

durability

effectiveness

efficiency

modularity

operability

orthogonality

portability

precision

predictability

process capabilities

productivity

provability

recoverability

relevance

survivability
sustainability

tailorability

testability

timeliness

traceability

transparency

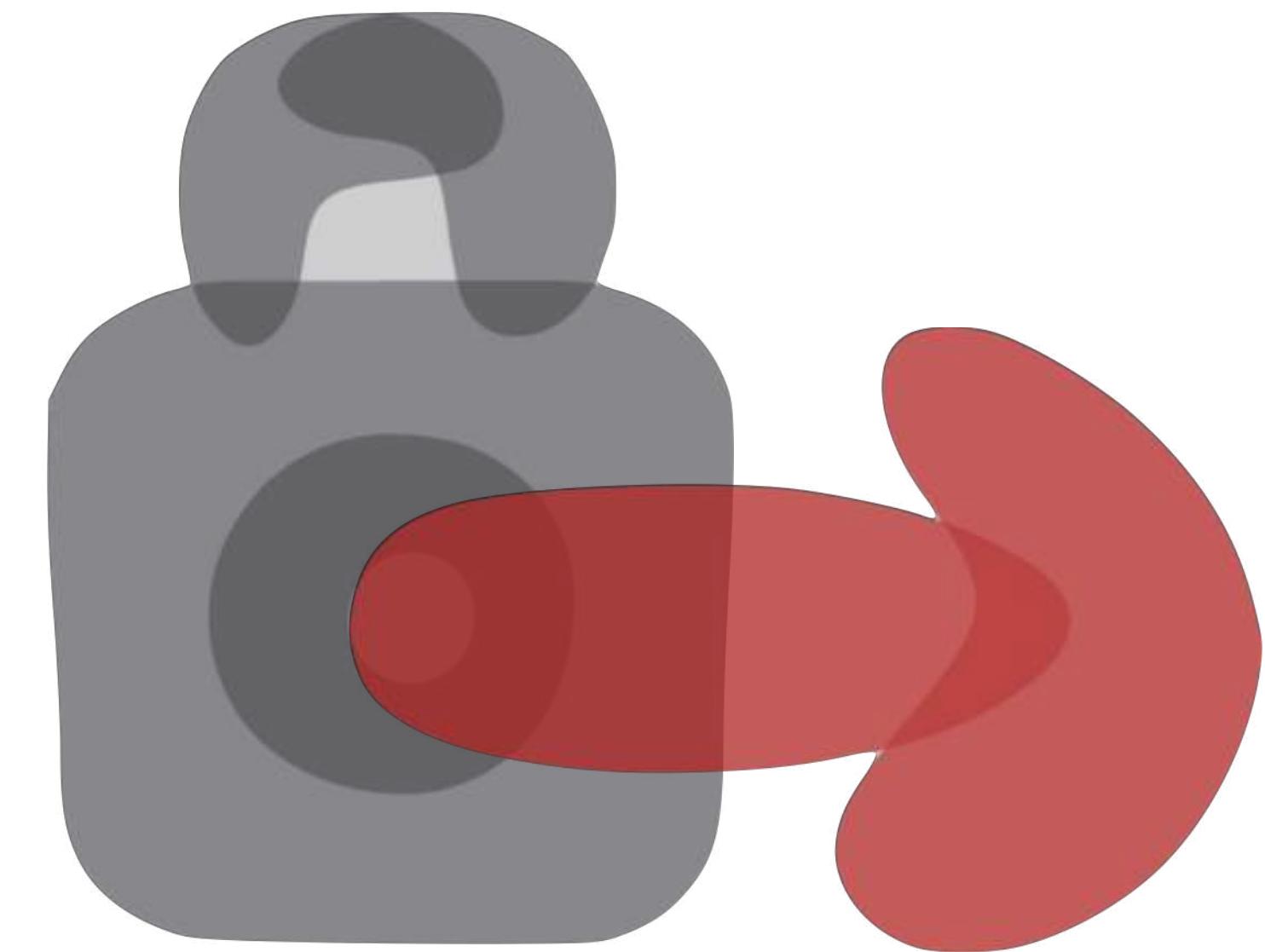
ubiquity

understandability

upgradeability

usability

governance



Evolutionary Architecture

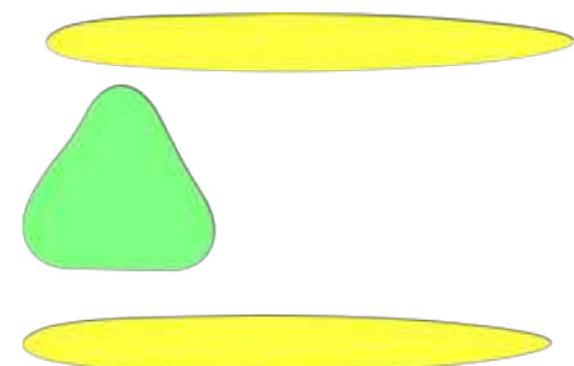
An evolutionary architecture supports
guided,
incremental change
across multiple dimensions.



Evolutionary Architecture

An evolutionary architecture supports
guided
incremental change
across multiple dimensions.



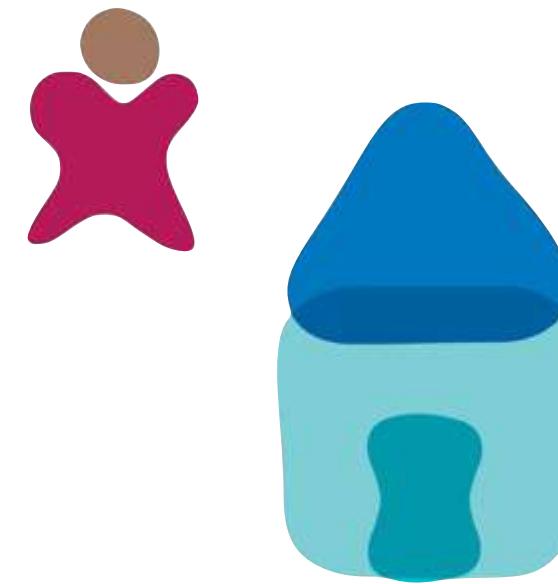
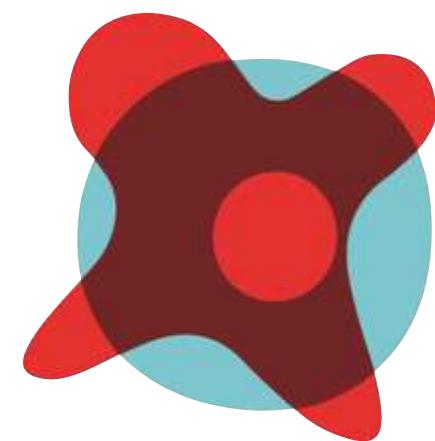
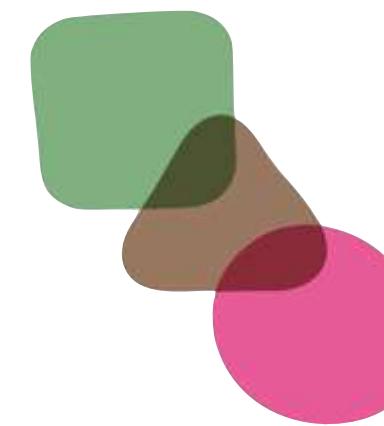
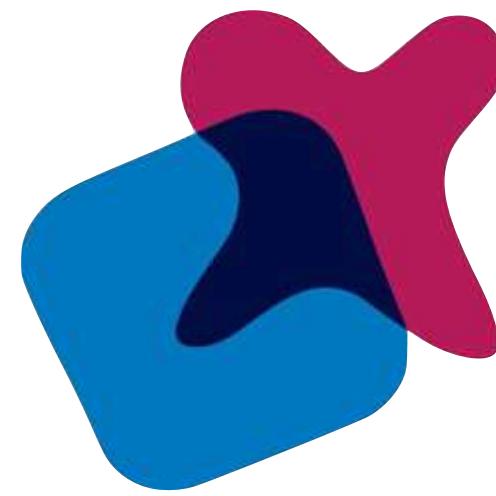
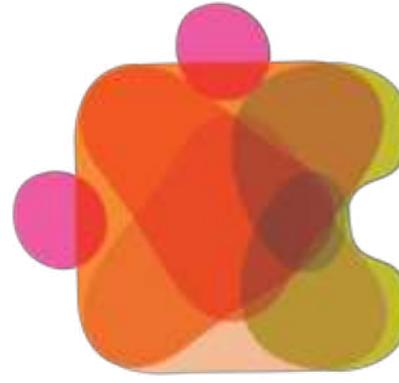


guided

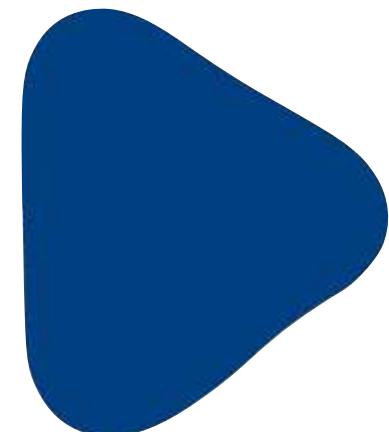
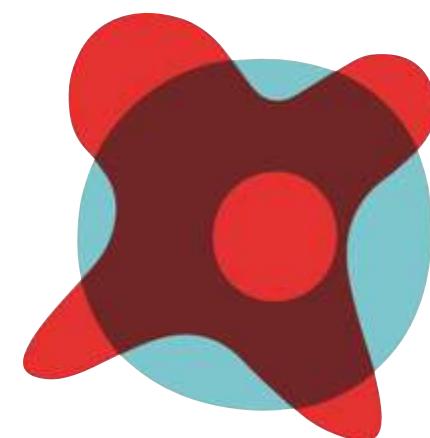
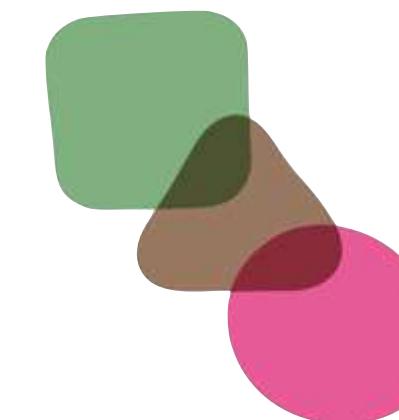
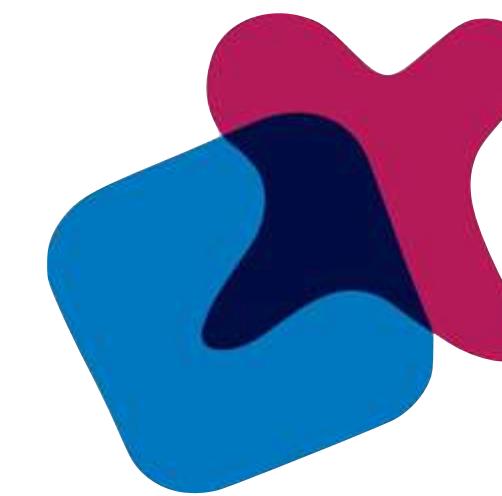
evolutionary computing fitness function:

a particular type of objective function that is used to summarize...how close a given design solution is to achieving the set aims.

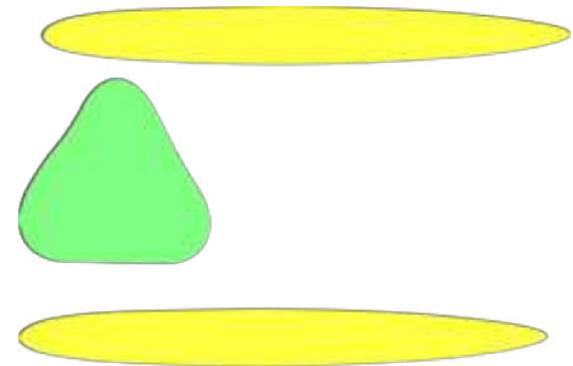
Traveling Salesman Problem



Traveling Salesman Problem



fitness function = length of route



guided

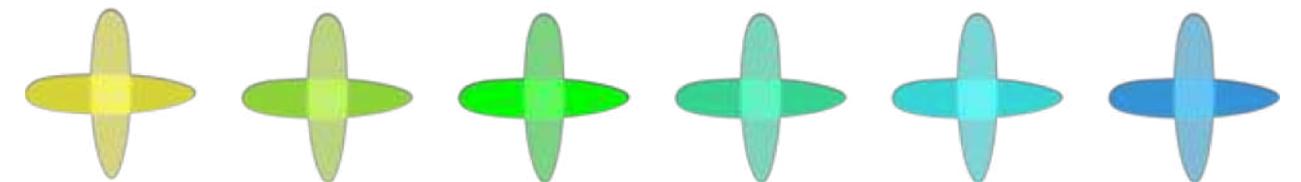
architectural fitness function:

An architectural fitness function provides an objective integrity assessment of some architectural characteristic(s).

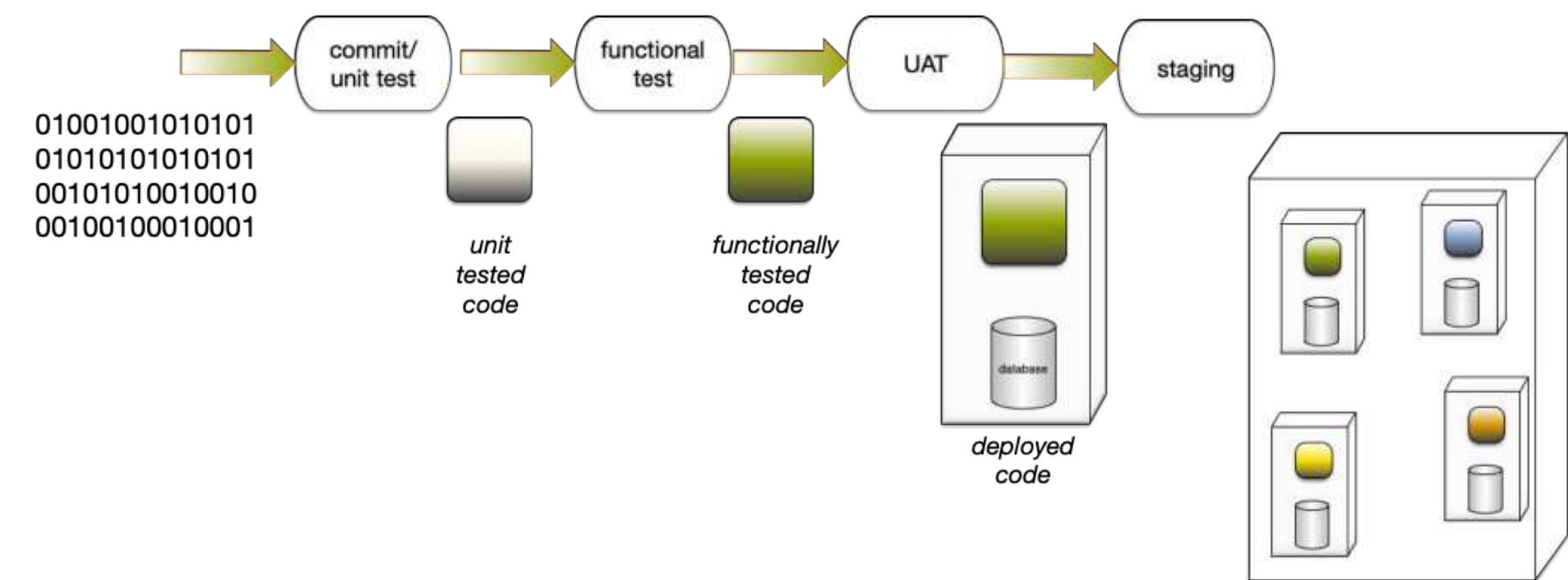
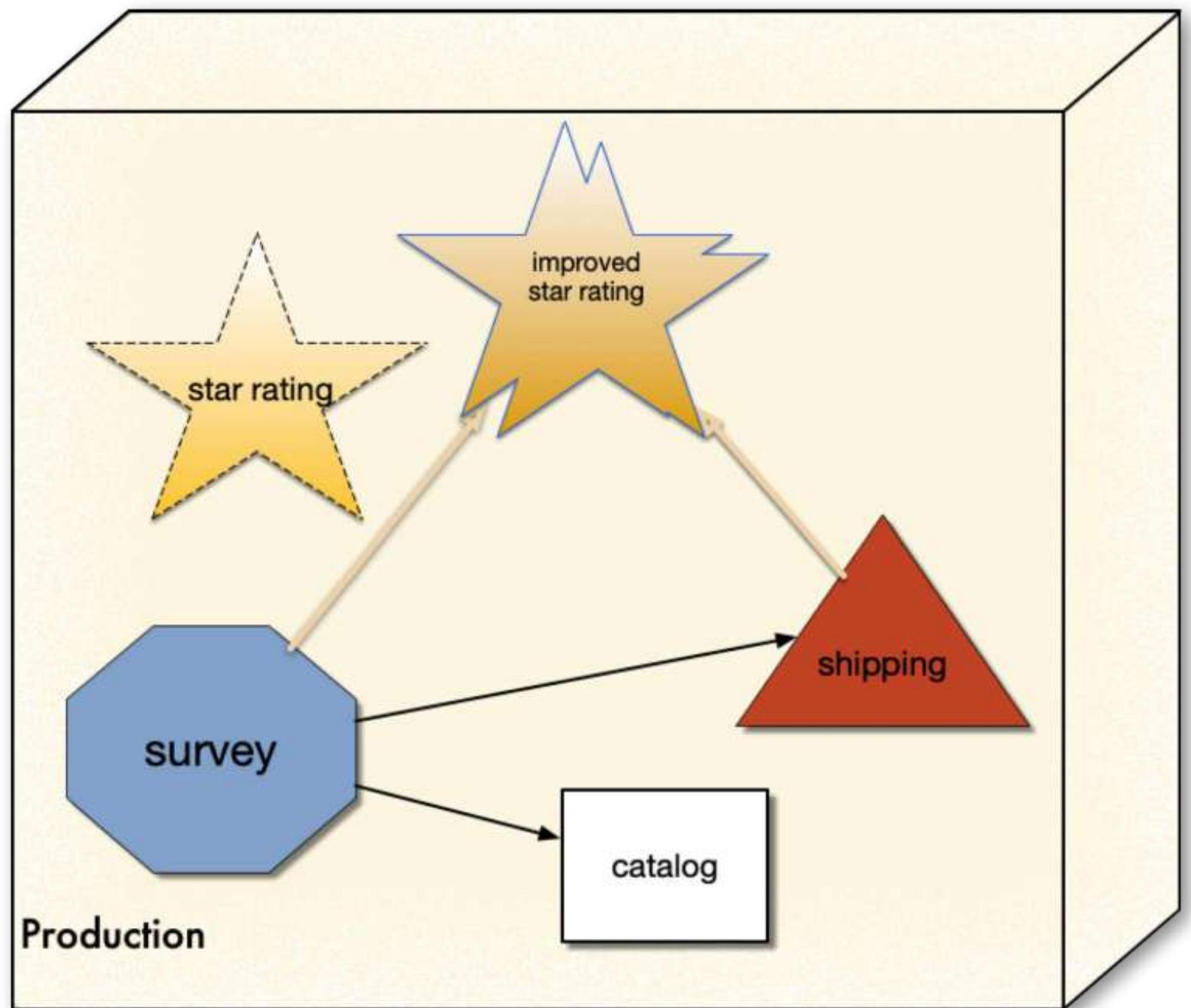
Evolutionary Architecture

An evolutionary architecture supports
guided,
incremental change
across multiple dimensions.



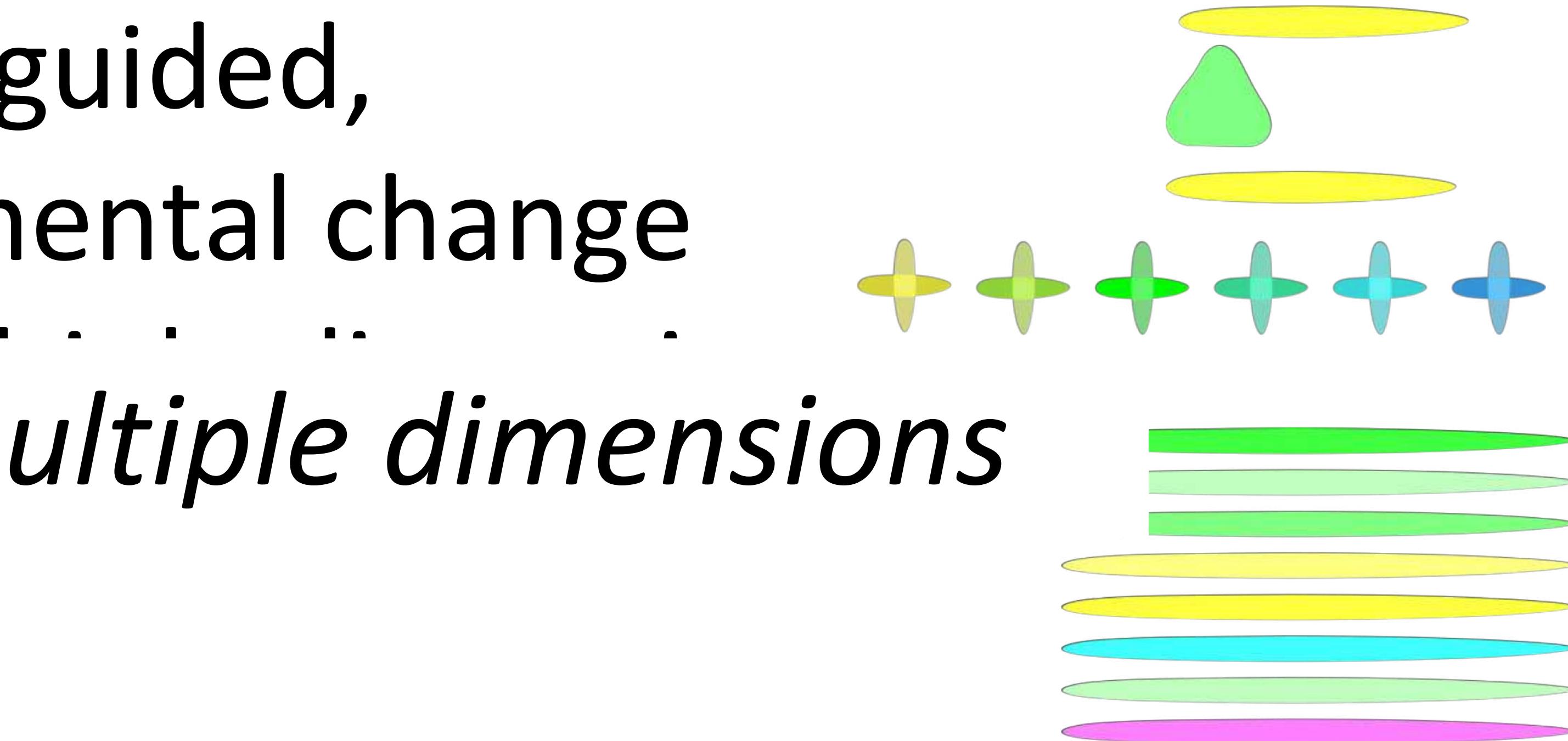


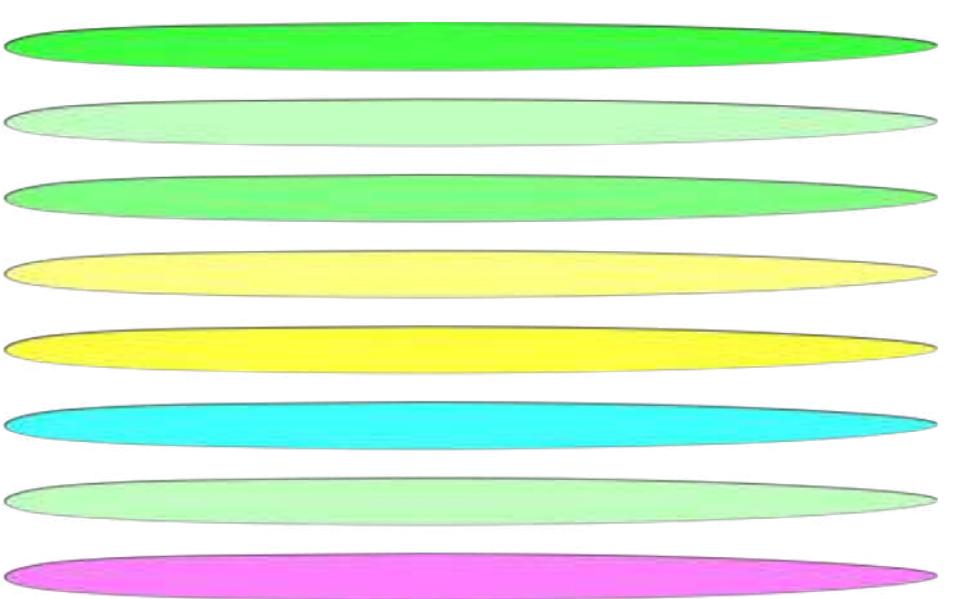
incremental



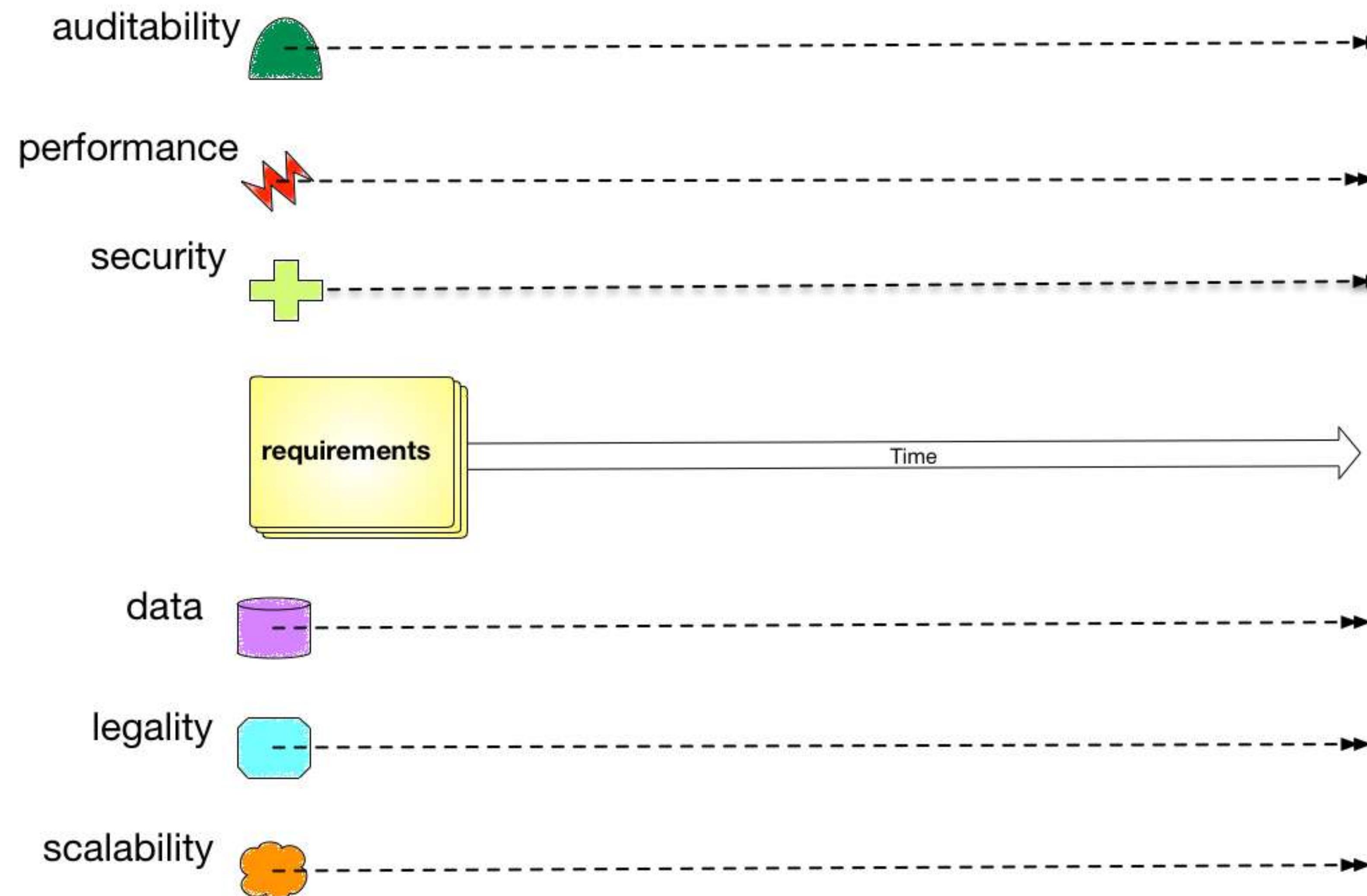
Evolutionary Architecture

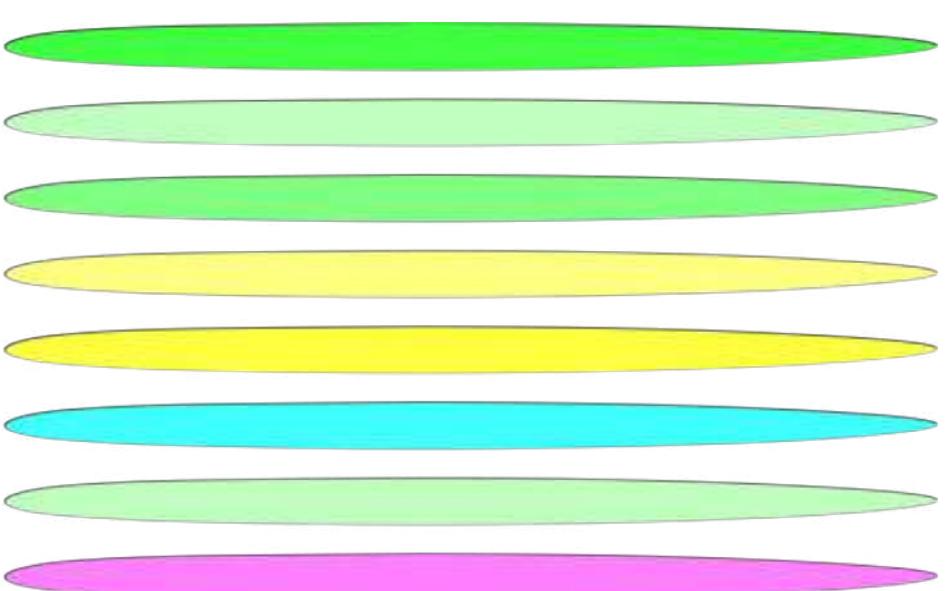
An evolutionary architecture supports
guided,
incremental change
across: *multiple dimensions*



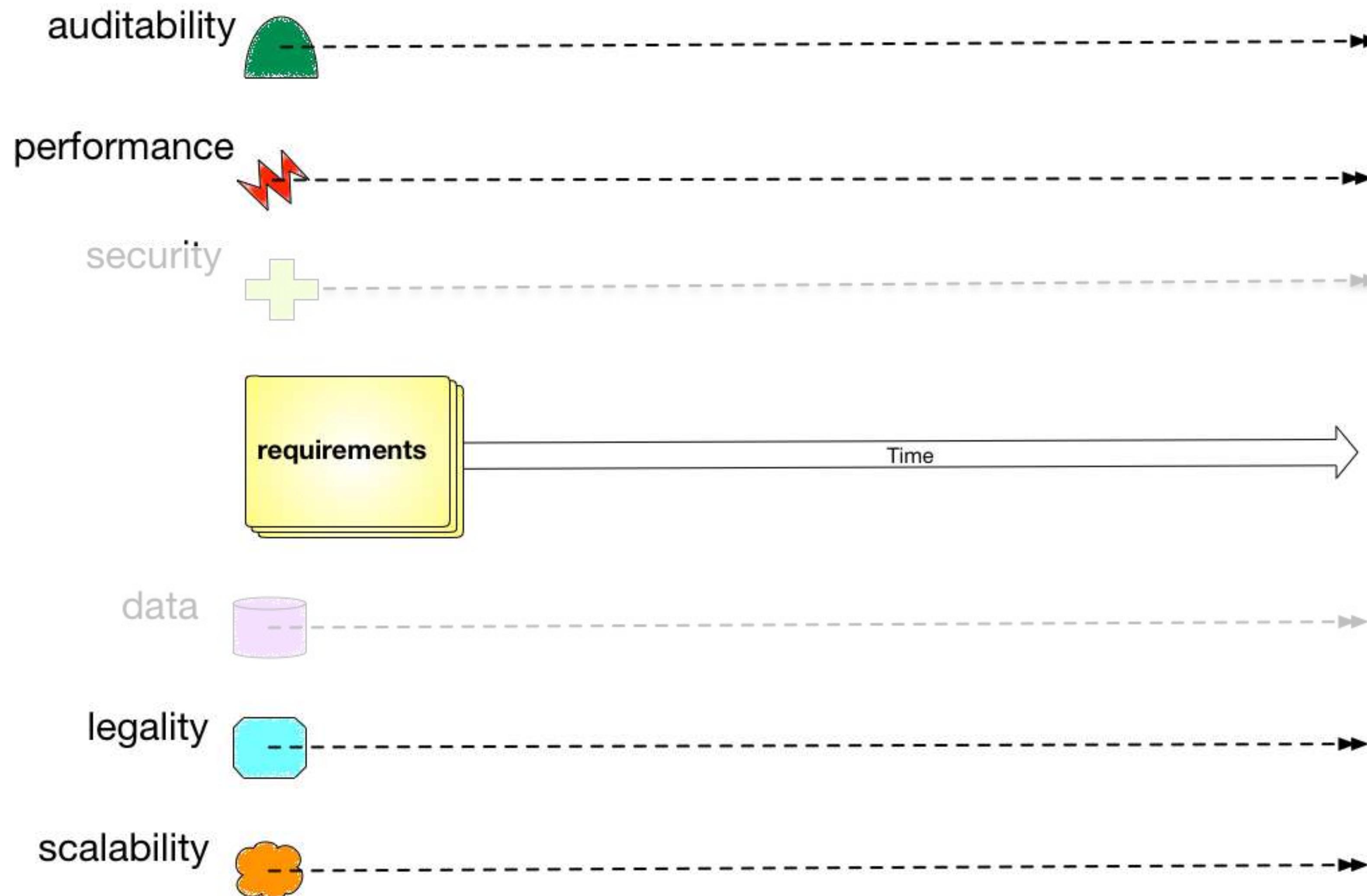


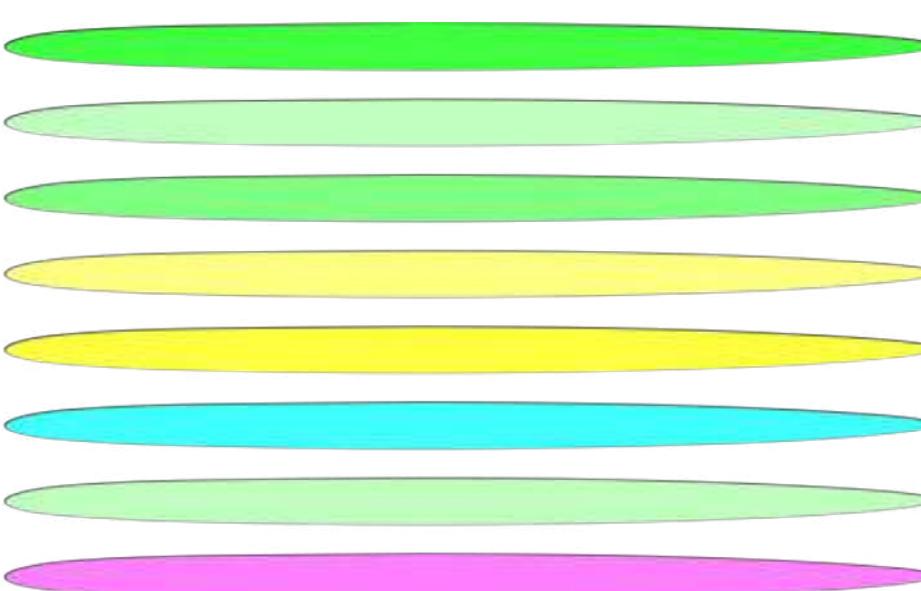
multiple dimensions



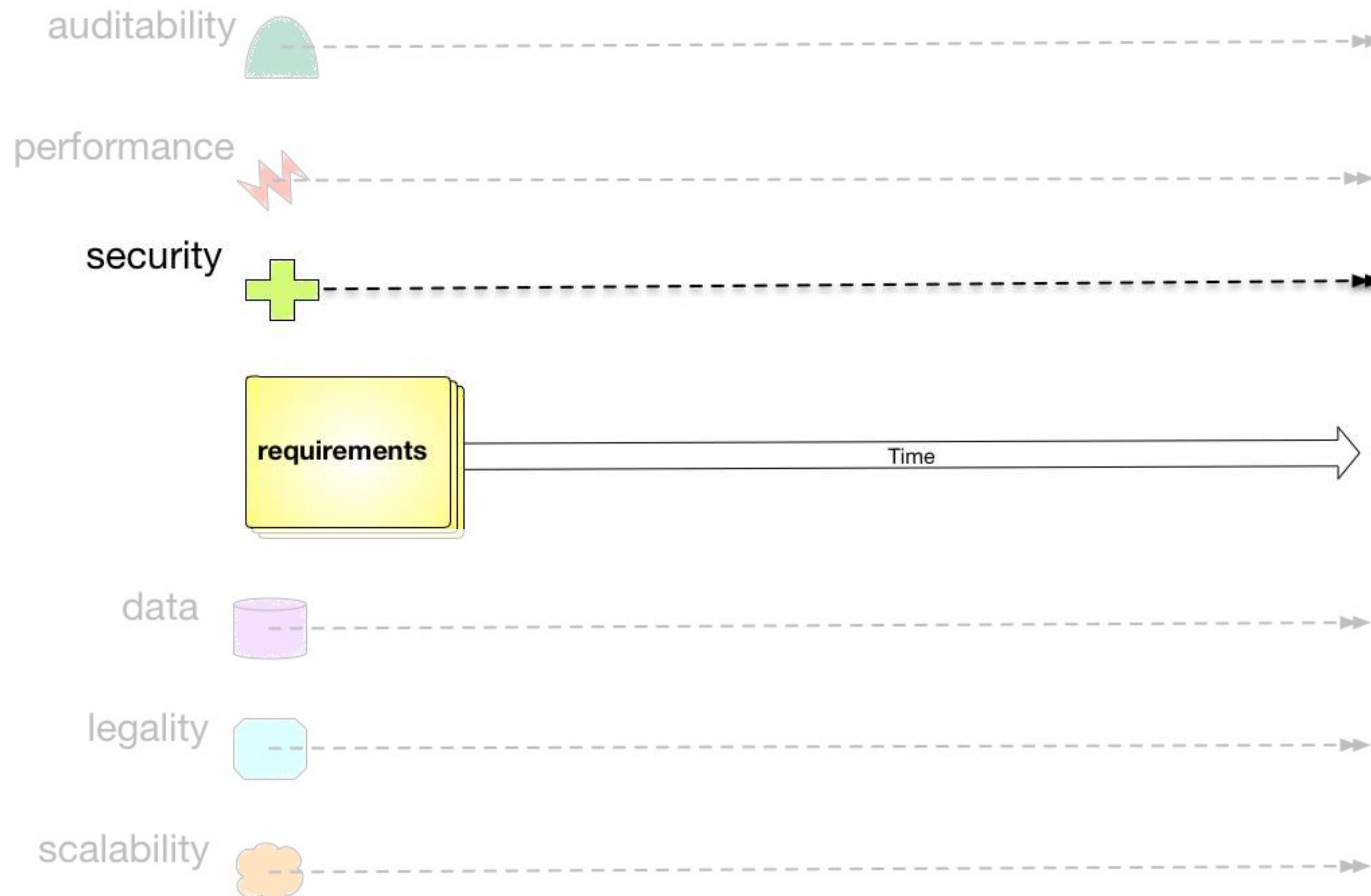


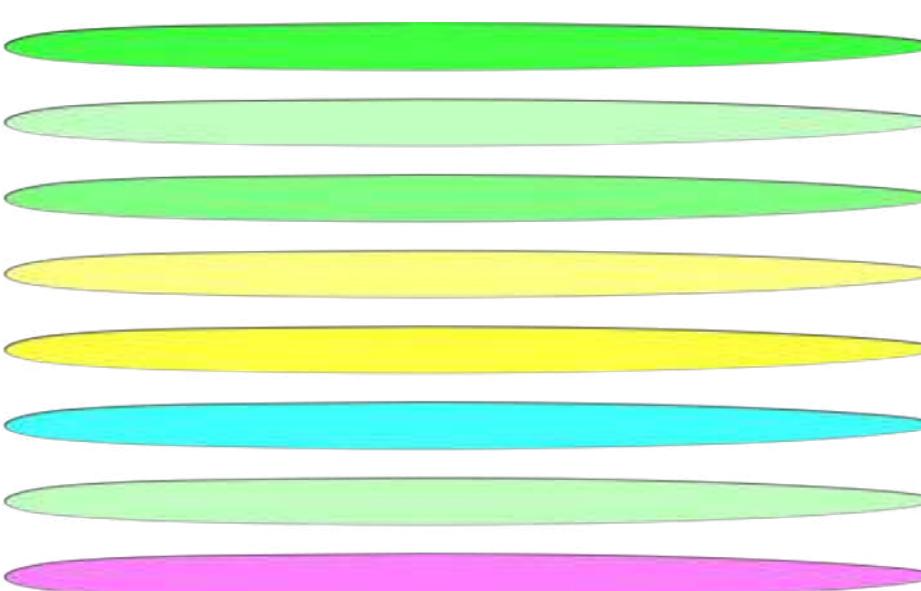
multiple dimensions



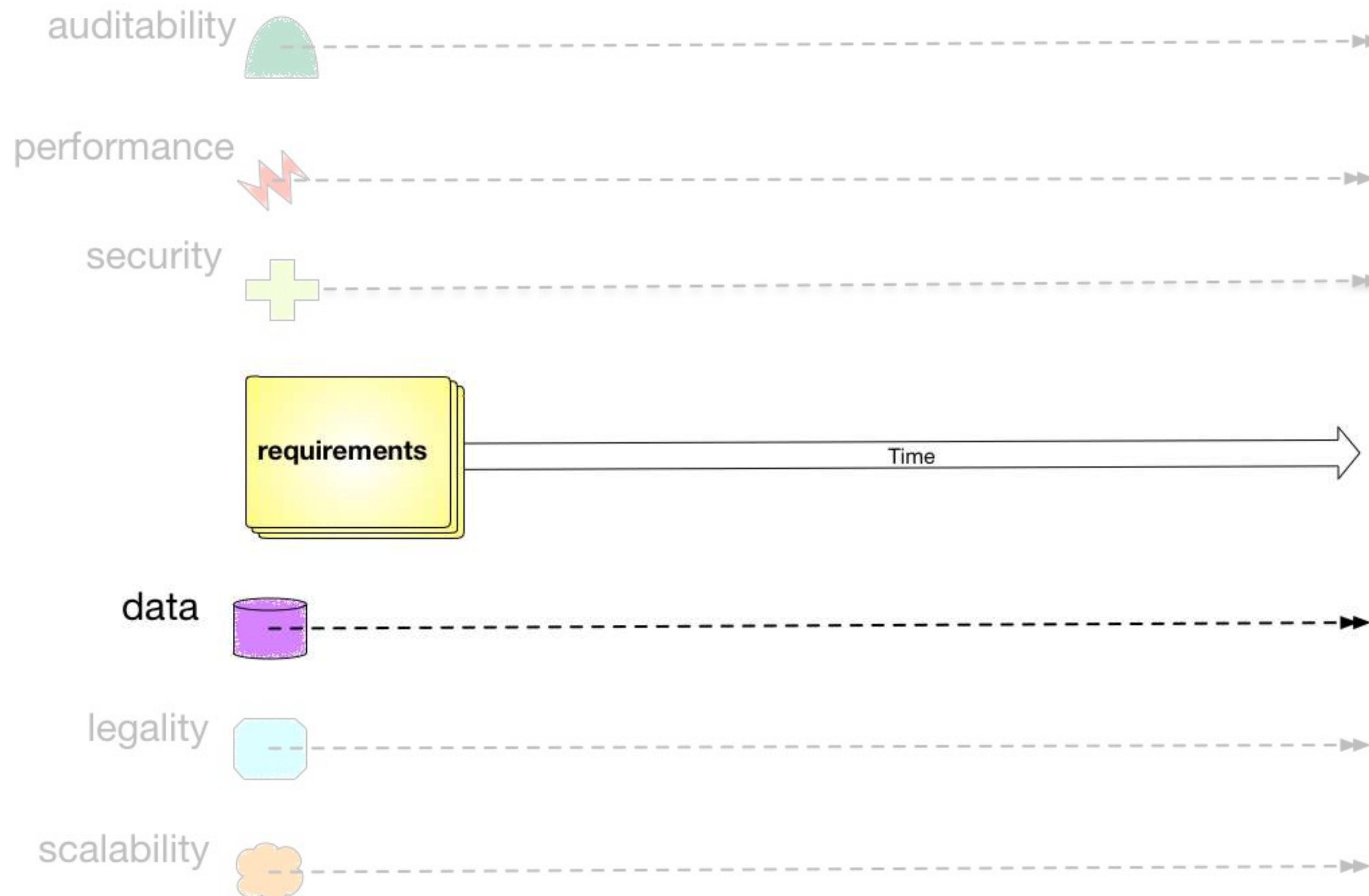


multiple dimensions





multiple dimensions



Evolutionary Architecture

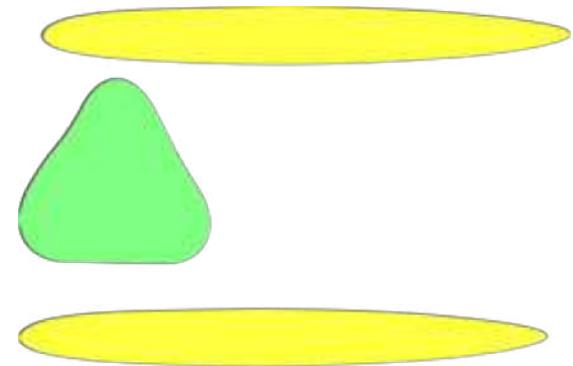
An evolutionary architecture supports
guided,
incremental change
across multiple dimensions.



Evolutionary Architecture

An evolutionary architecture supports
guided
incremental change
across multiple dimensions.



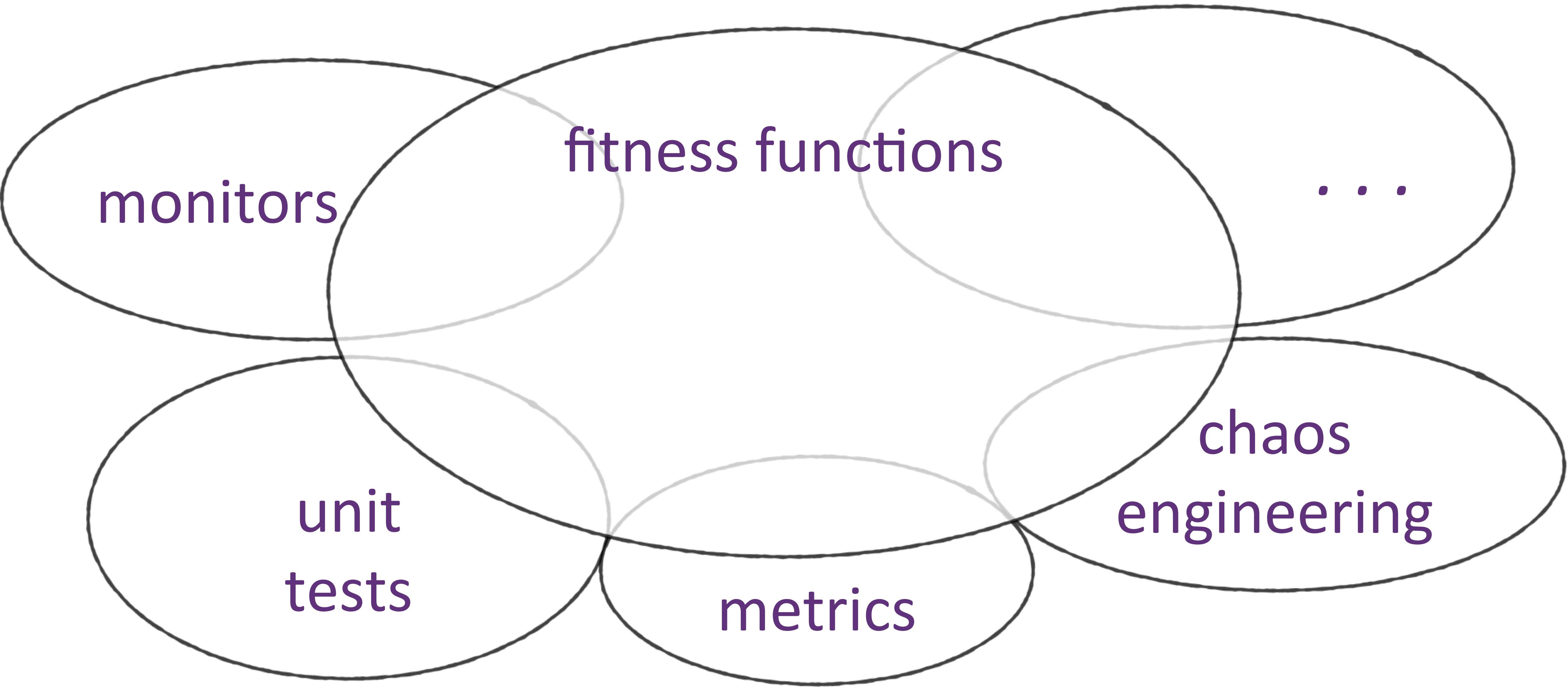


guided

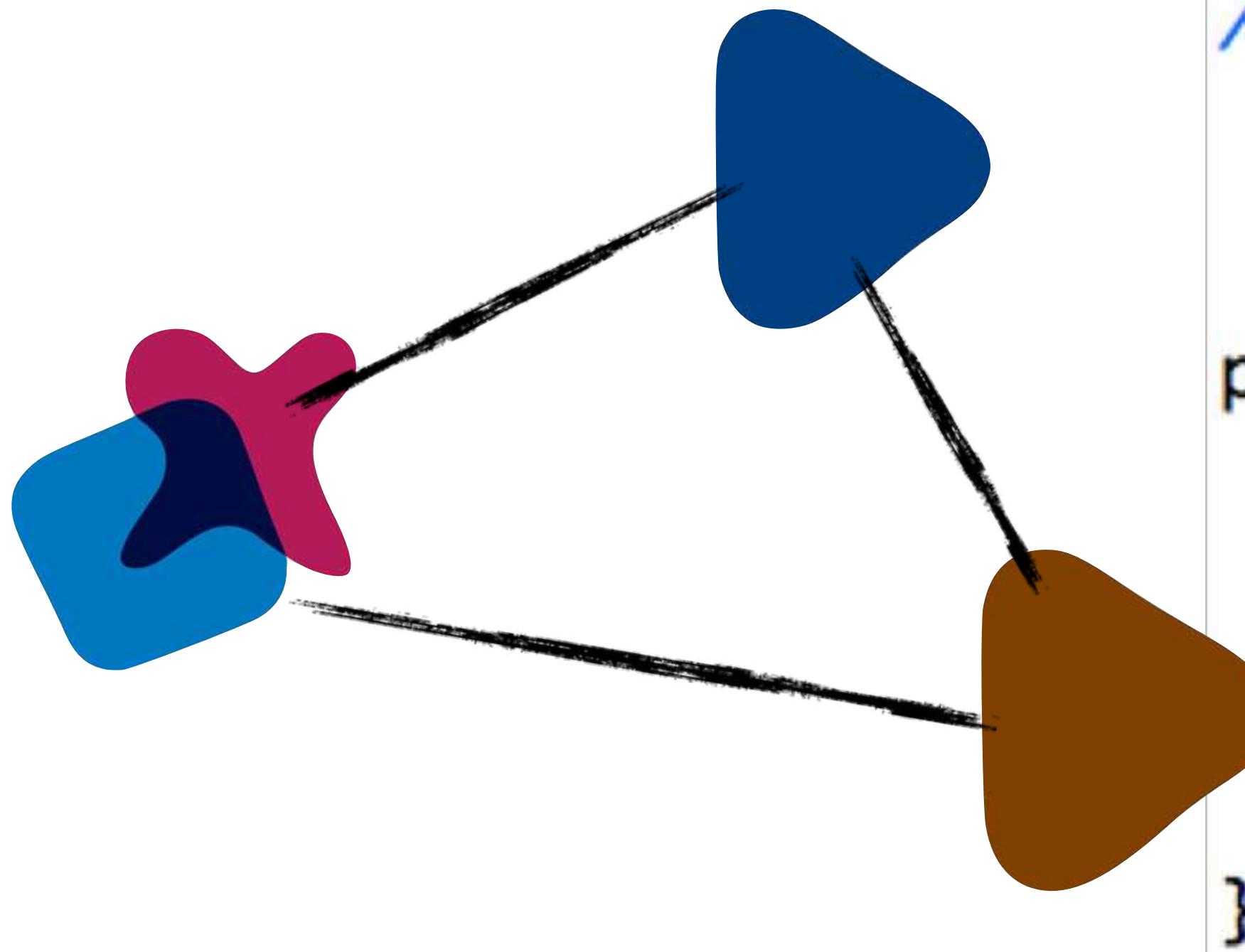
architectural fitness function:

An architectural fitness function provides an objective integrity assessment of some architectural characteristic(s).

Fitness Functions

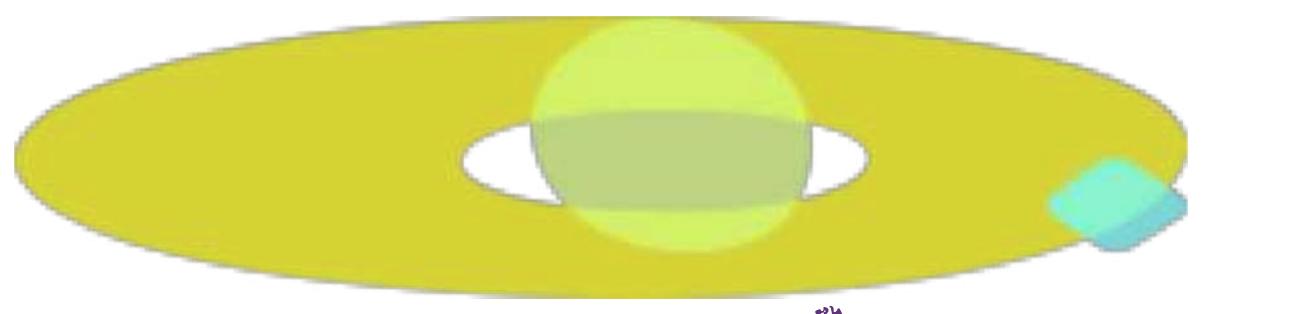


Cyclic Dependency Function

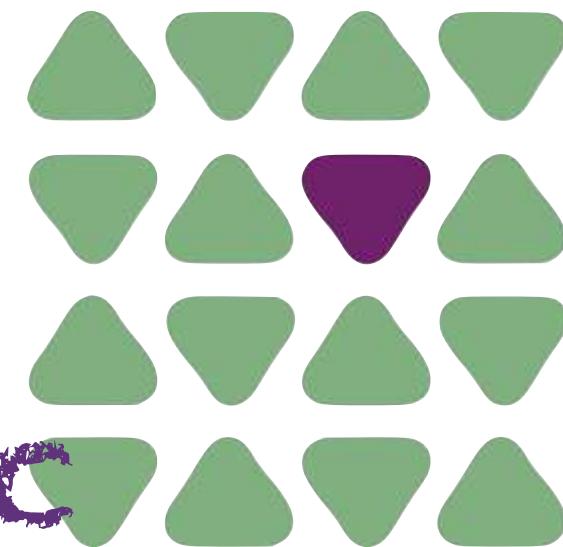
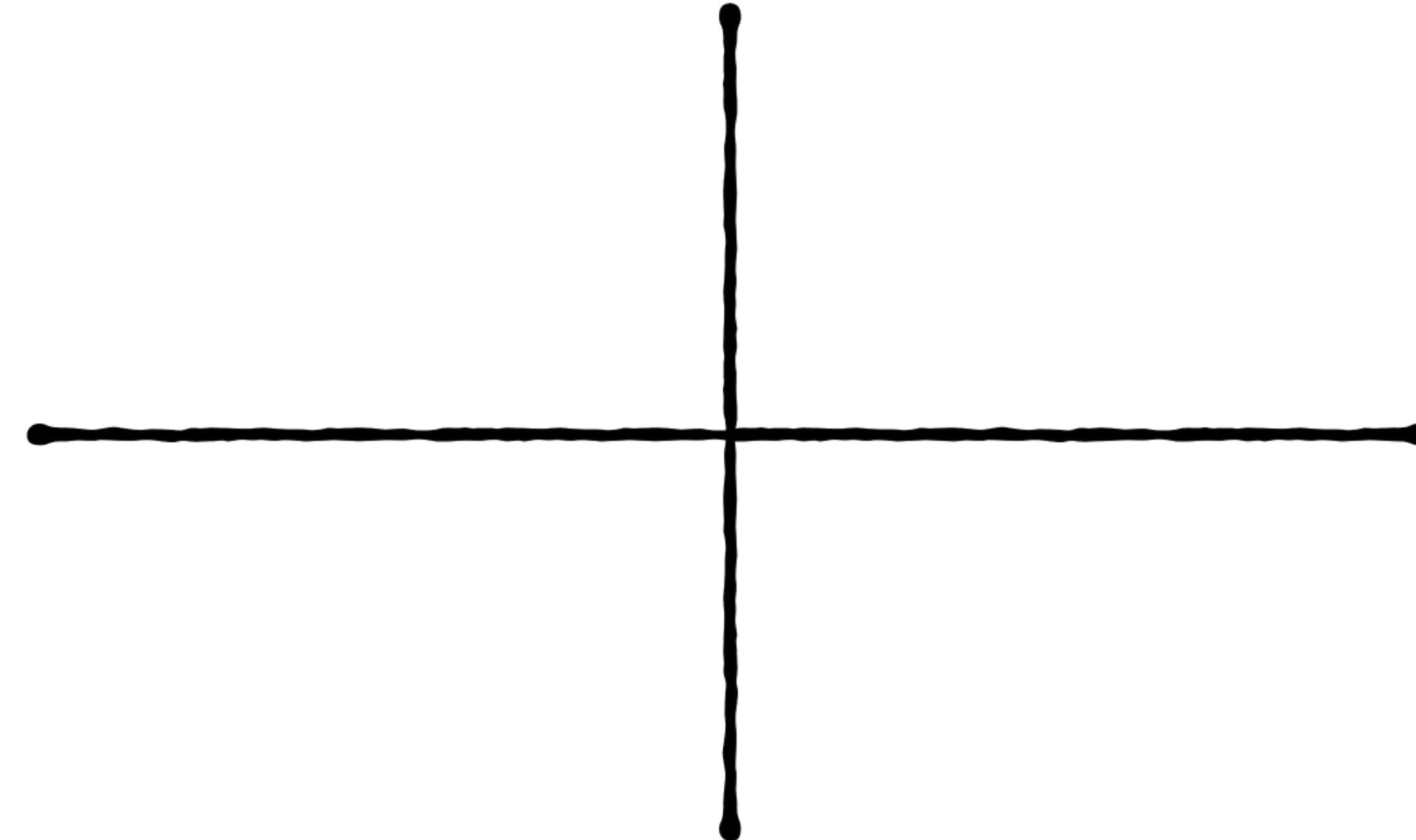


```
/**  
 * Tests that a package dependency cycle does not  
 * exist for any of the analyzed packages.  
 */  
public void testAllPackages() {  
  
    Collection packages = jdepend.analyze();  
  
    assertEquals("Cycles exist",  
                false, jdepend.containsCycles());  
}
```

Fitness Function



atomic



holistic

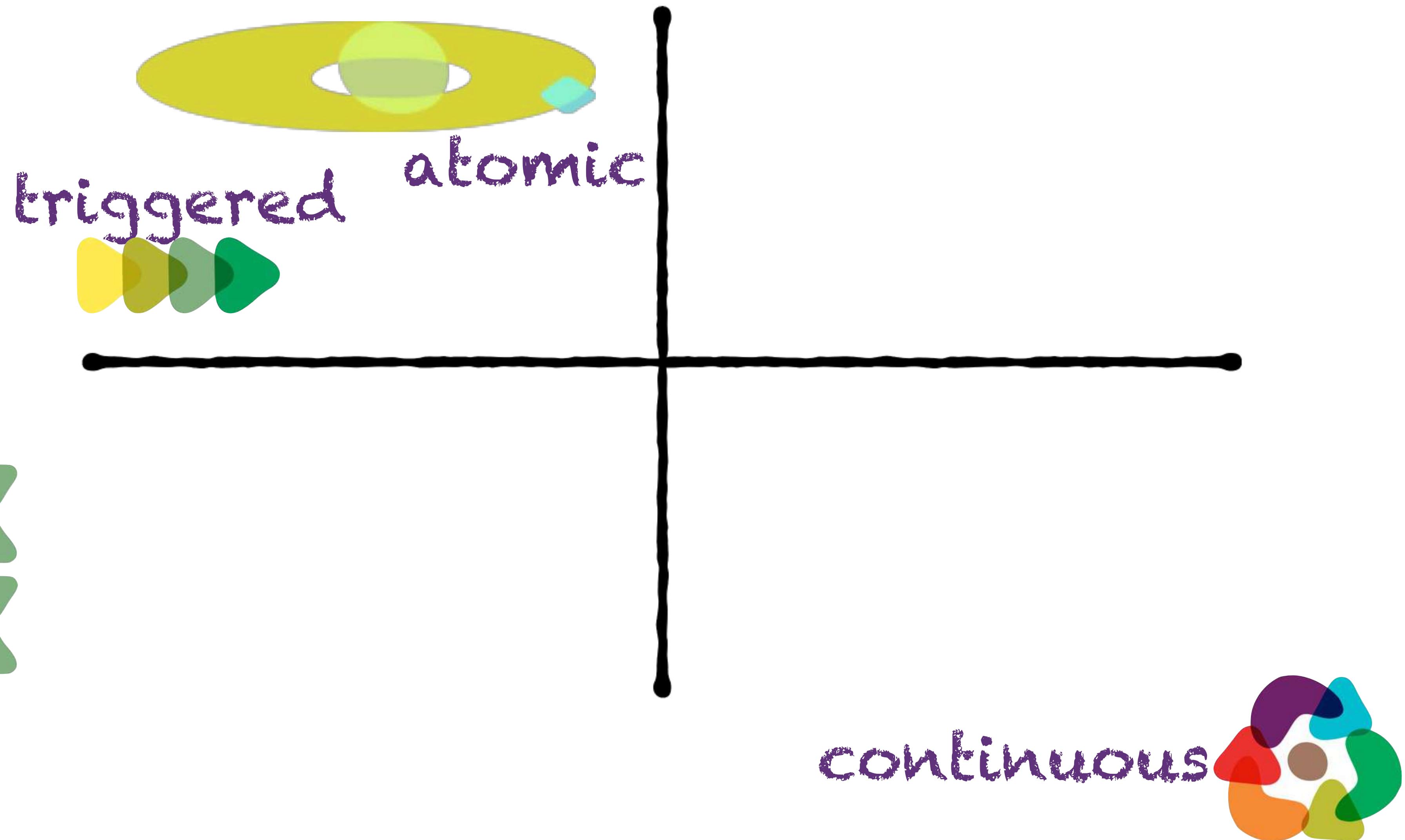


triggered

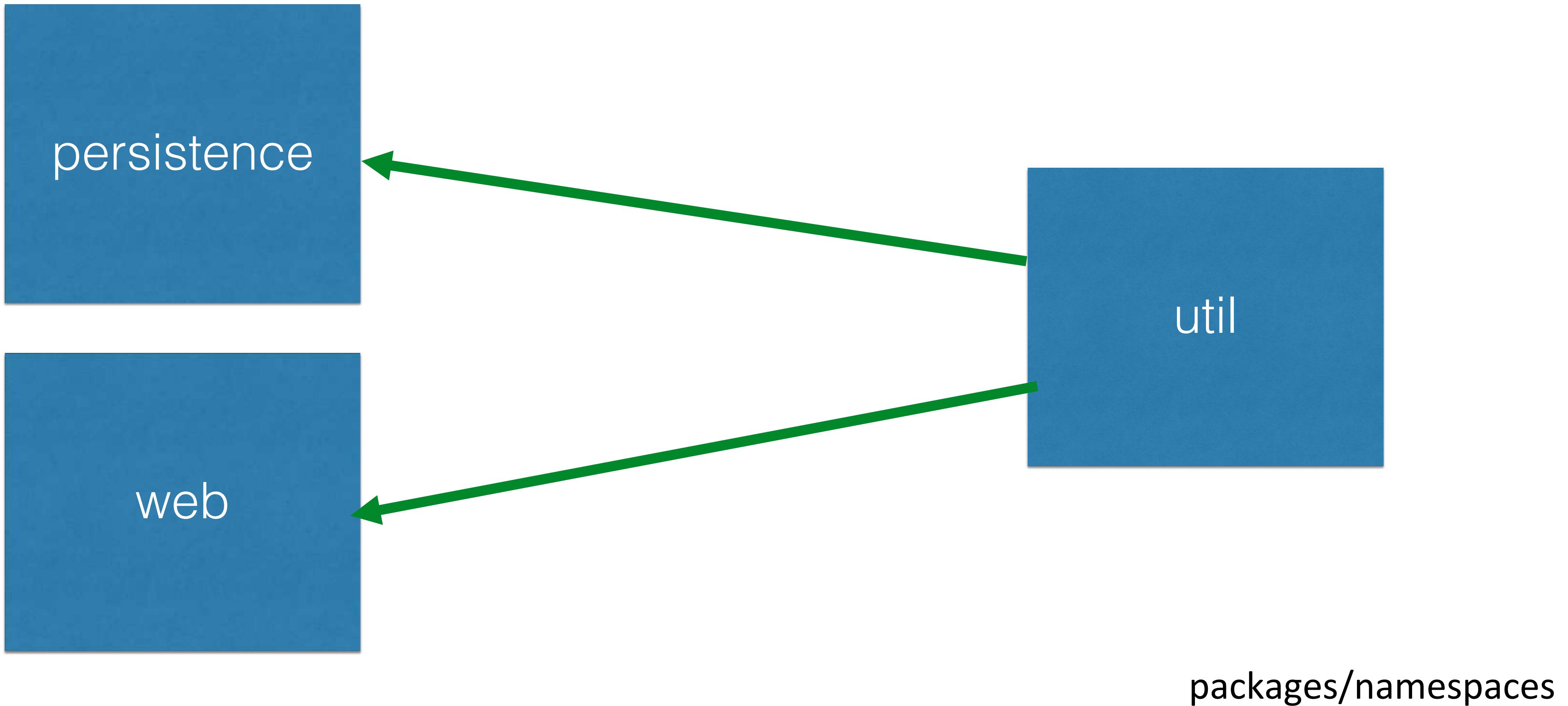


continuous

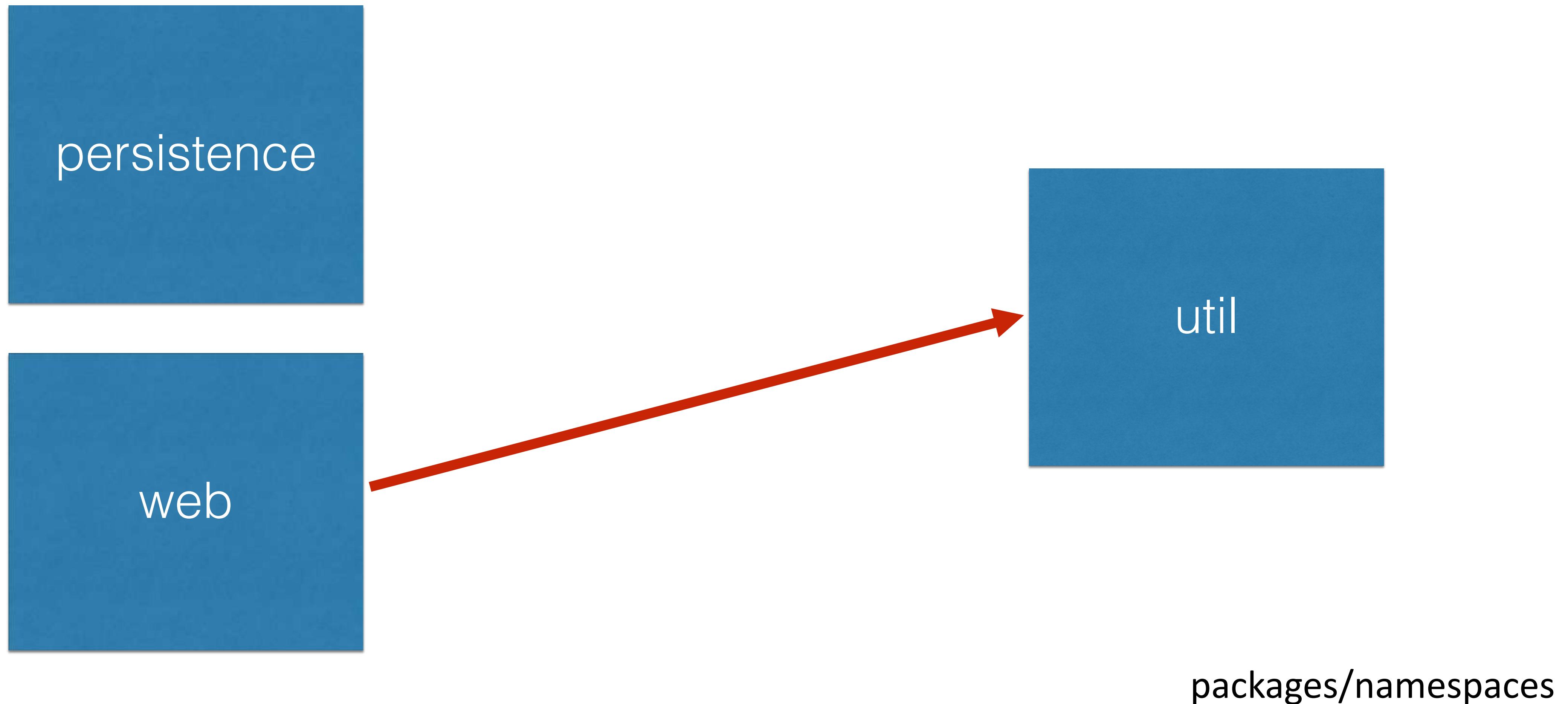
Fitness Function



Directionality of Imports



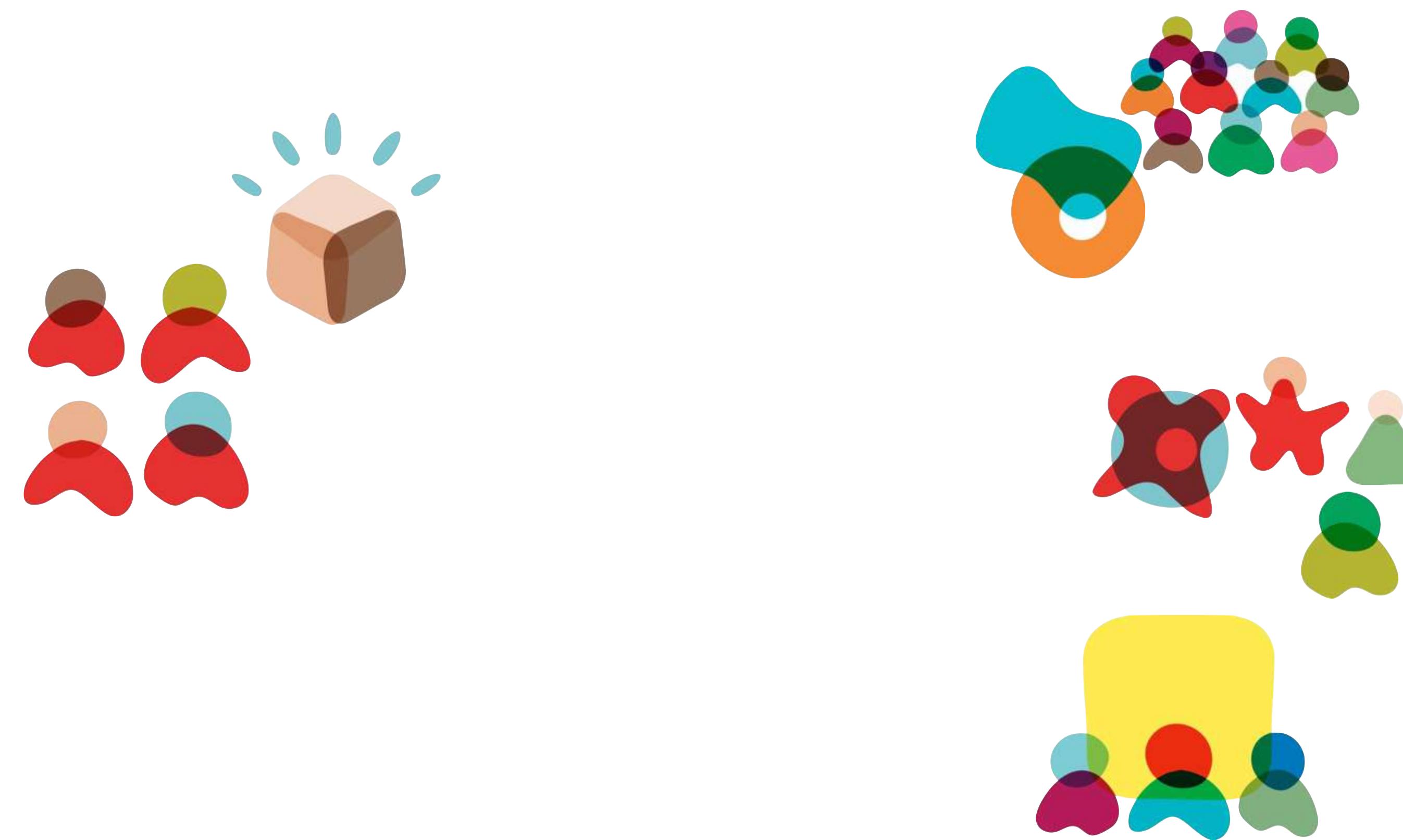
Directionality of Imports



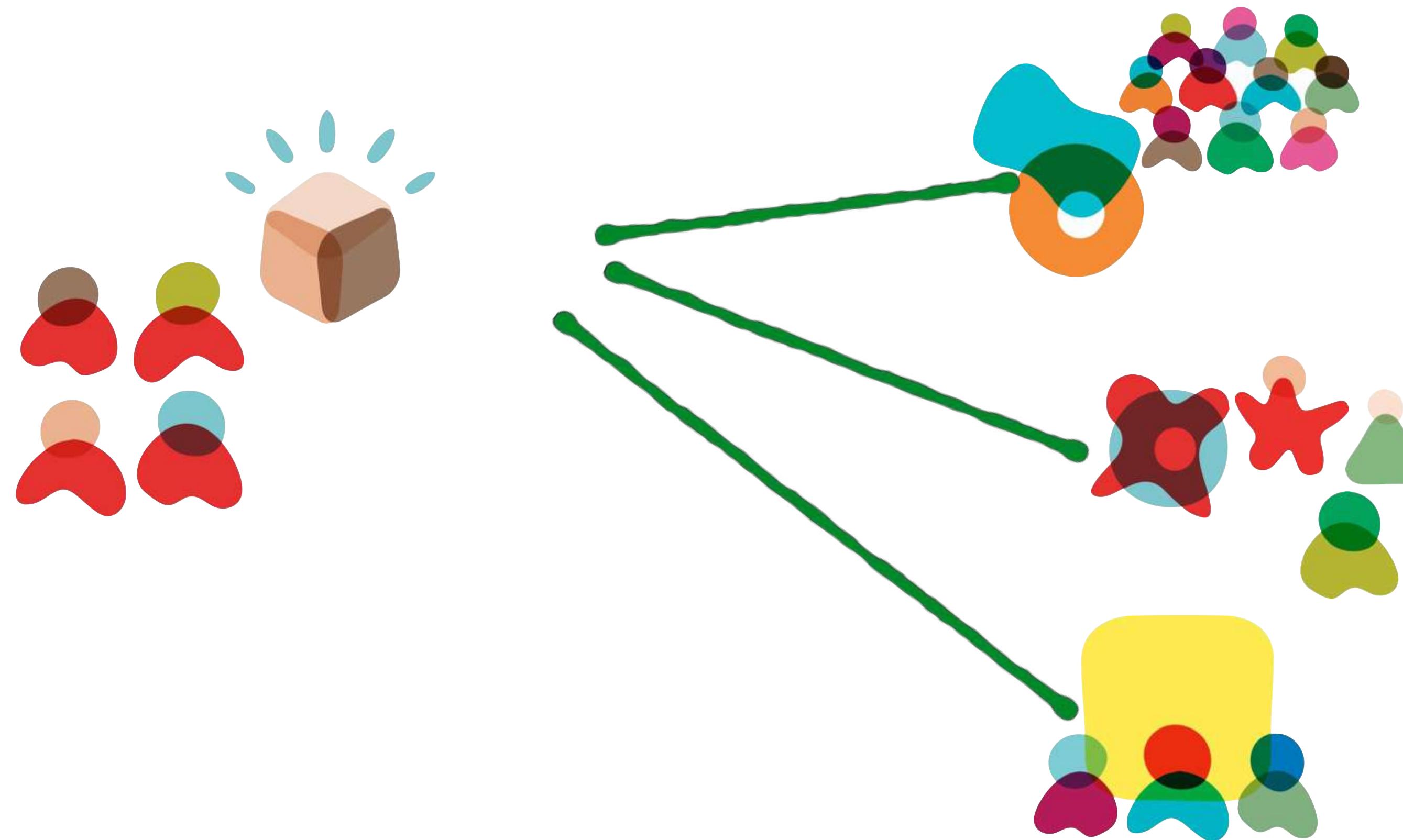
Coupling Fitness Function

```
public void testMatch() {  
    DependencyConstraint constraint = new DependencyConstraint();  
  
    JavaPackage persistence = constraint.addPackage("com.xyz.persistence");  
    JavaPackage web = constraint.addPackage("com.xyz.web");  
    JavaPackage util = constraint.addPackage("com.xyz.util");  
  
    persistence.dependsUpon(util);  
    web.dependsUpon(util);  
  
    jdepend.analyze();  
  
    assertEquals("Dependency mismatch",  
                true, jdepend.dependencyMatch(constraint));  
}
```

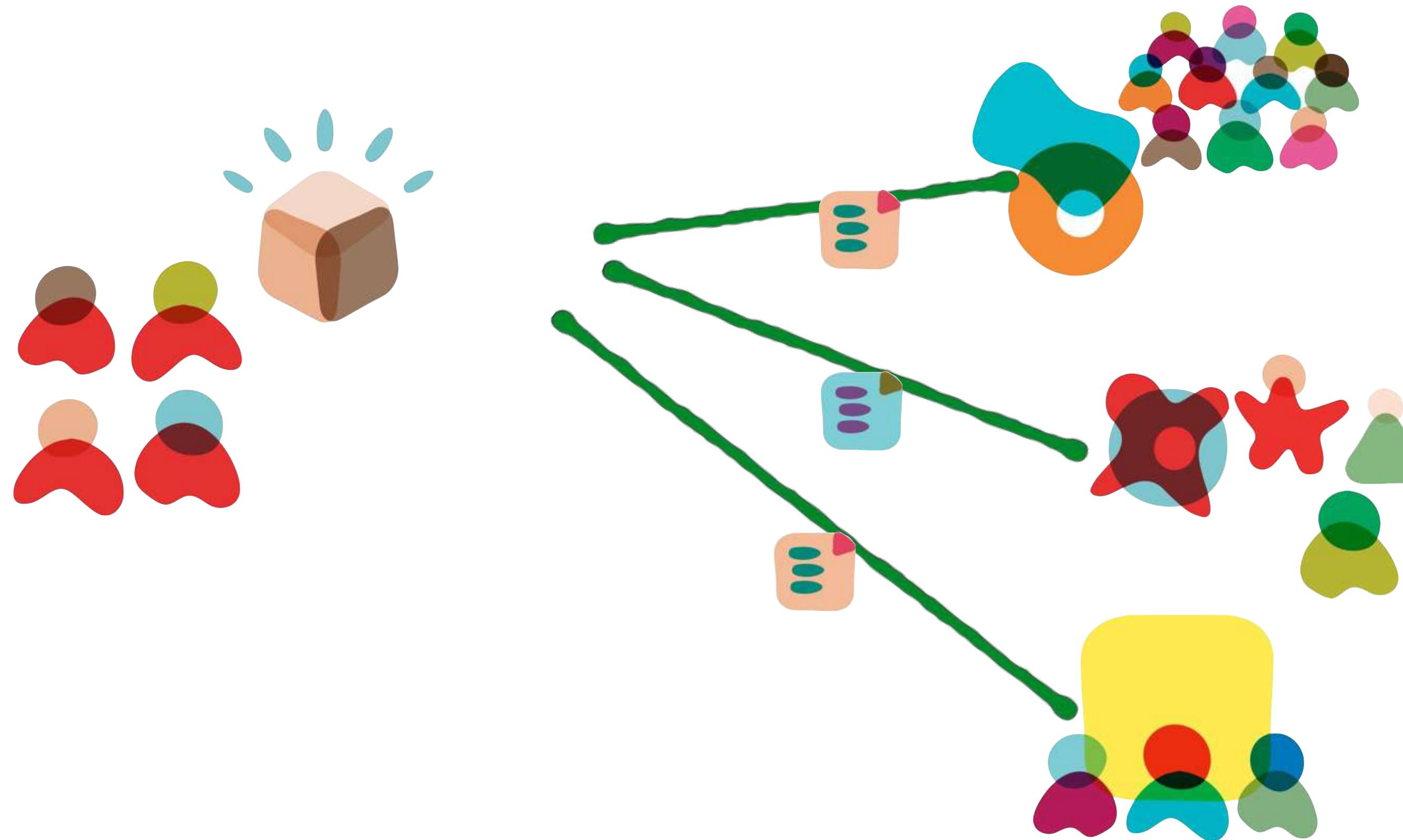
Consumer Driven Contracts



Consumer Driven Contracts

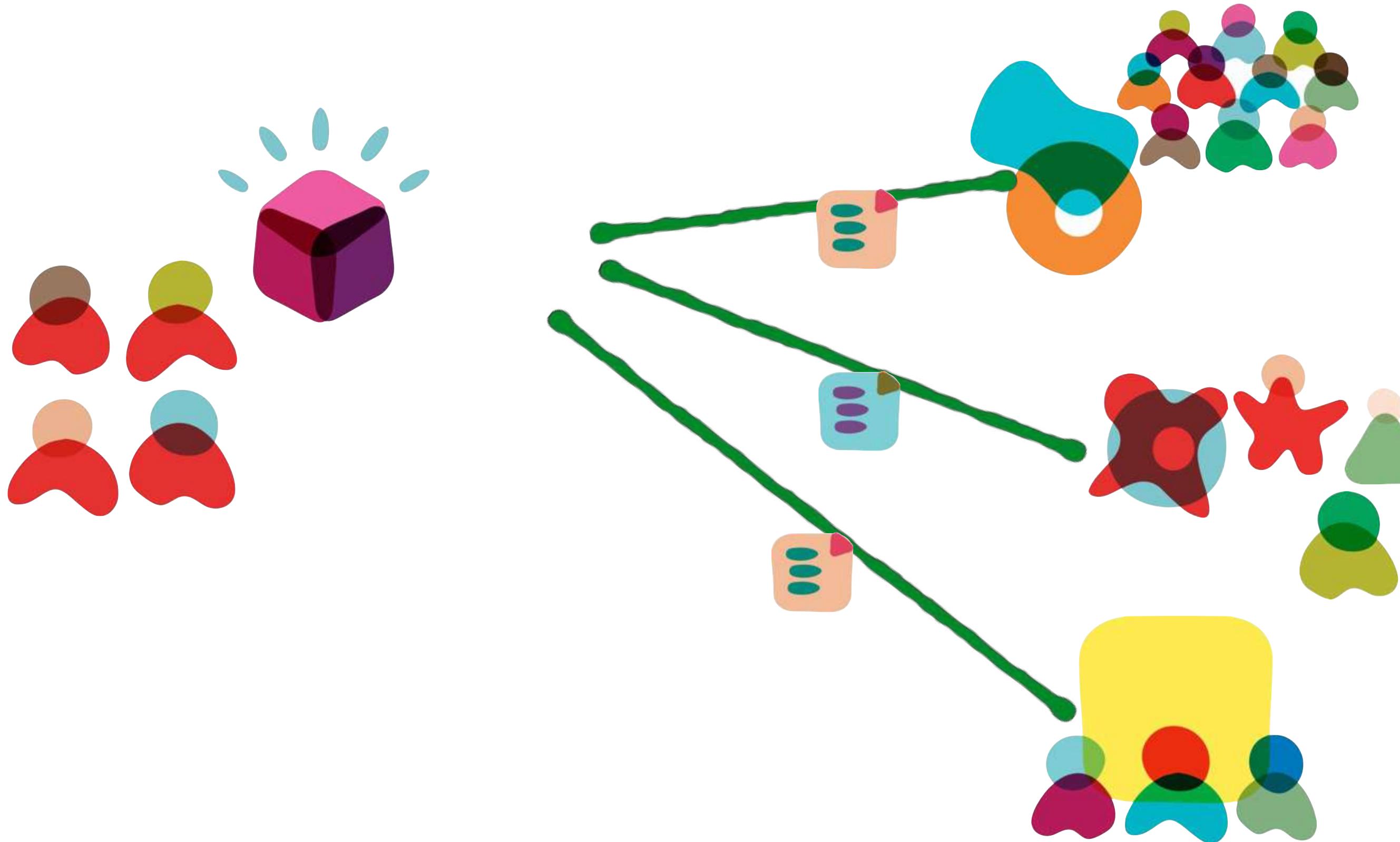


Consumer Driven Contracts



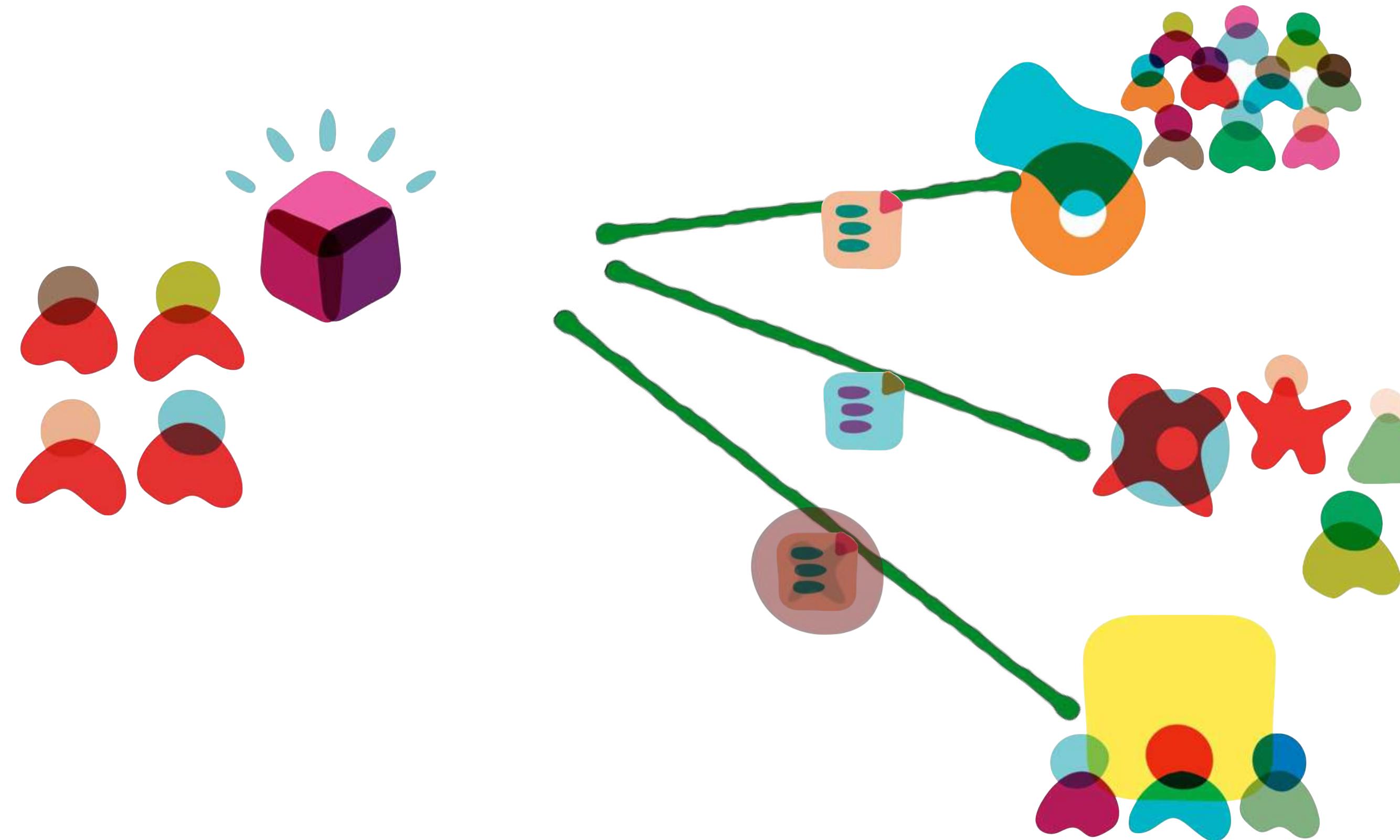
martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts



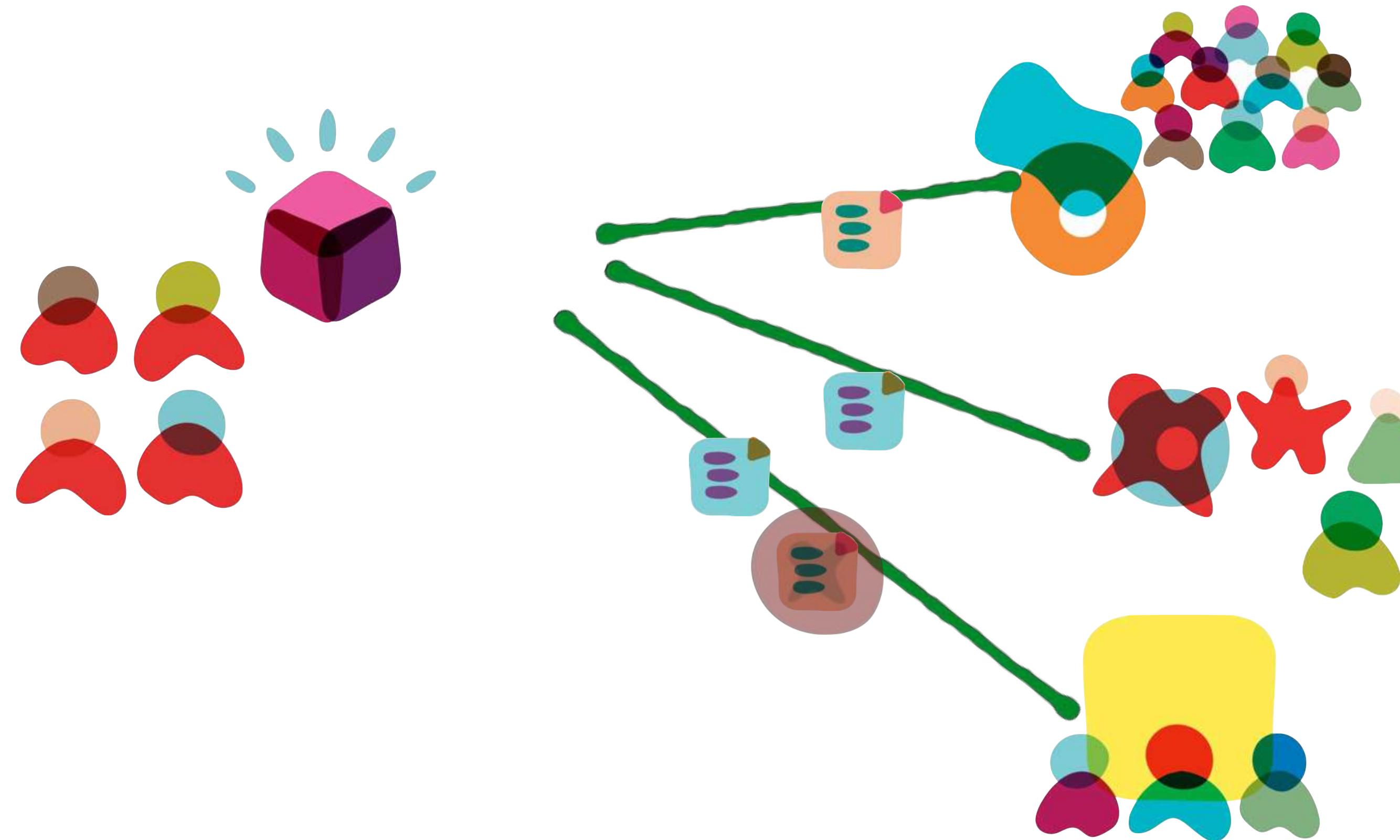
martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts



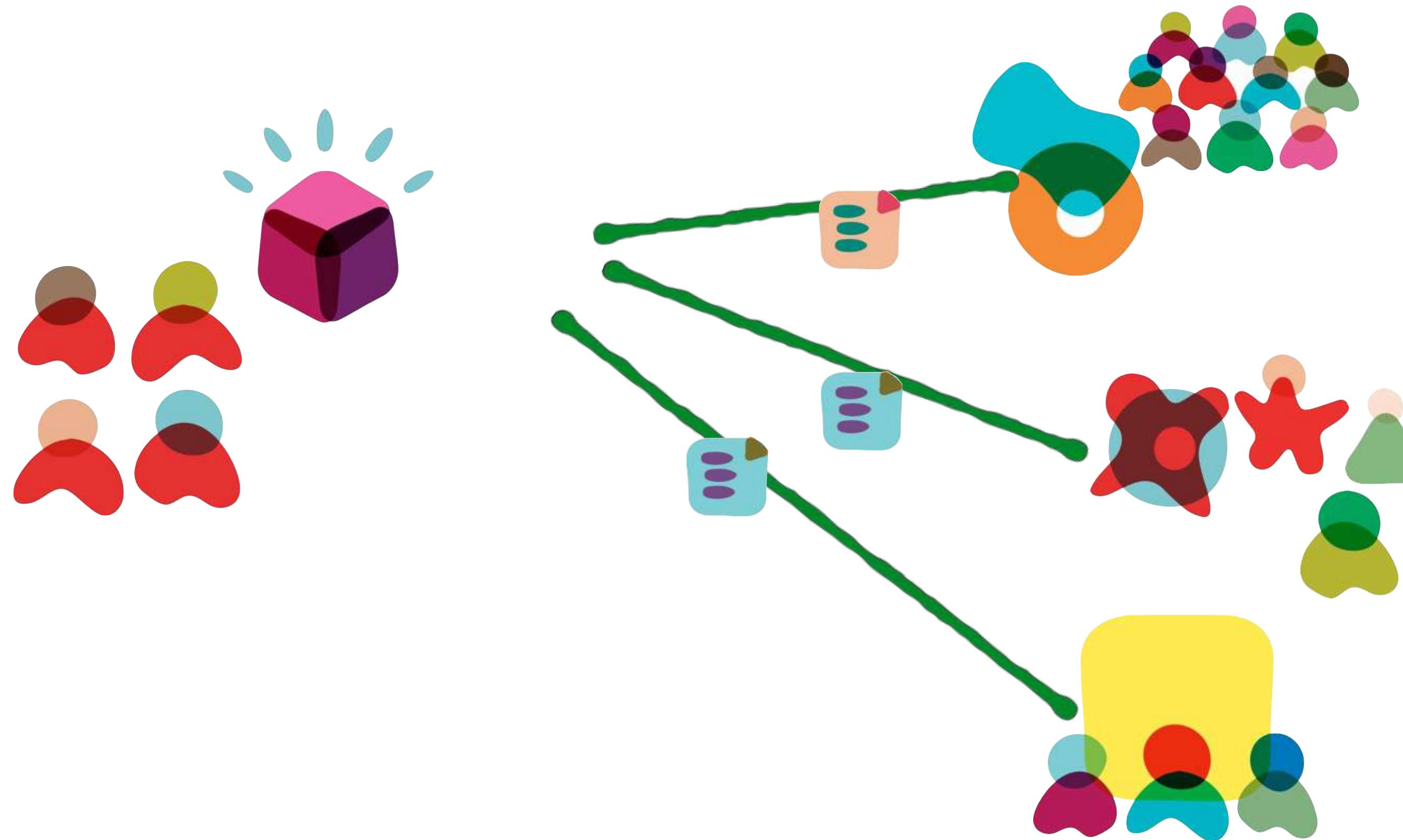
martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts



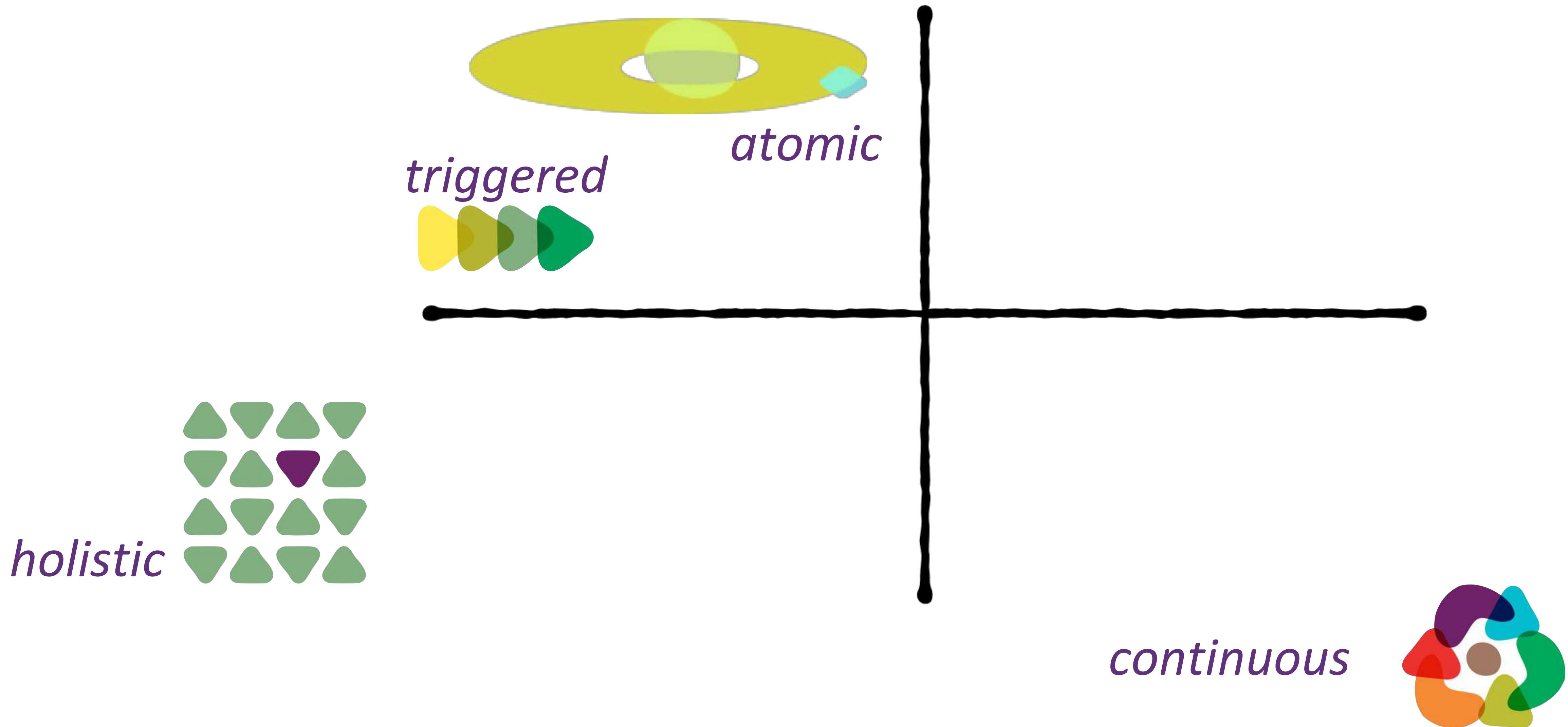
martinfowler.com/articles/consumerDrivenContracts.html

Consumer Driven Contracts

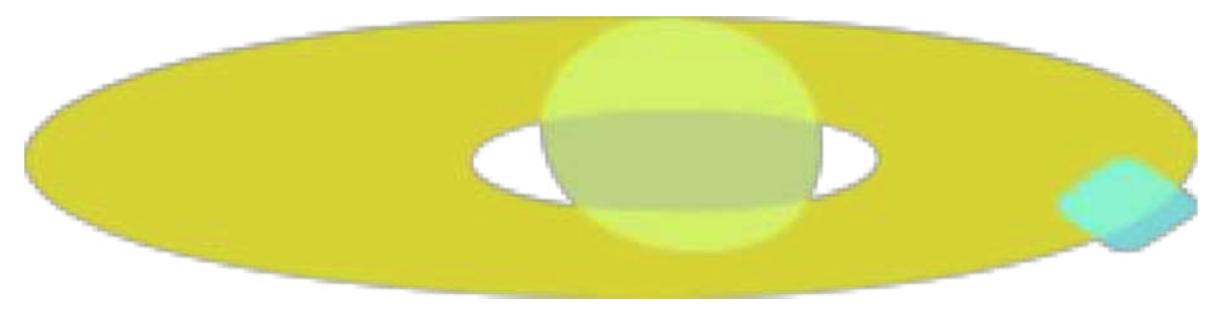


martinfowler.com/articles/consumerDrivenContracts.html

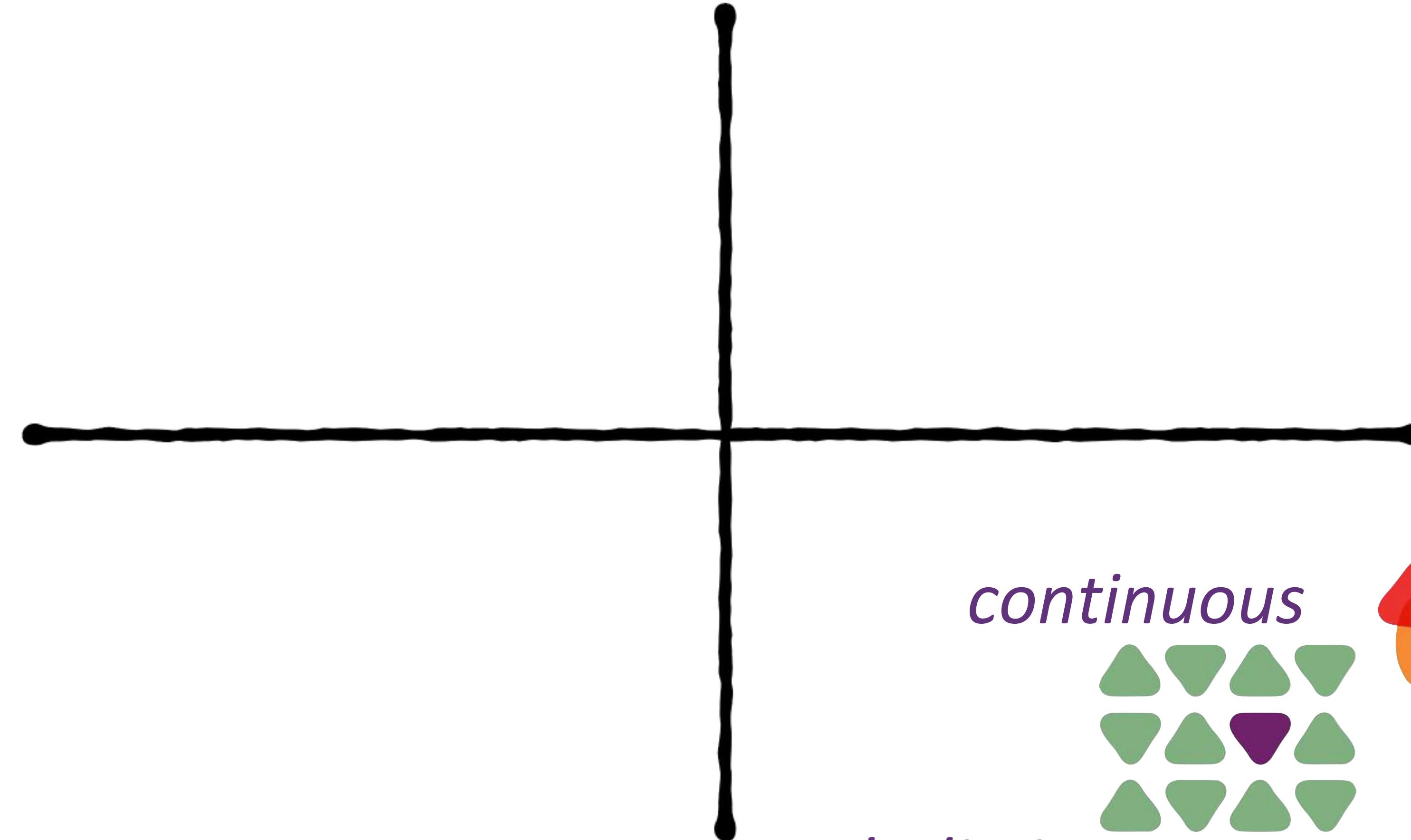
Fitness Function



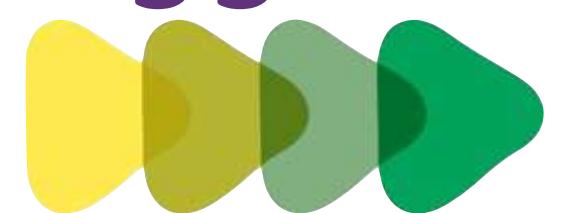
Fitness Function



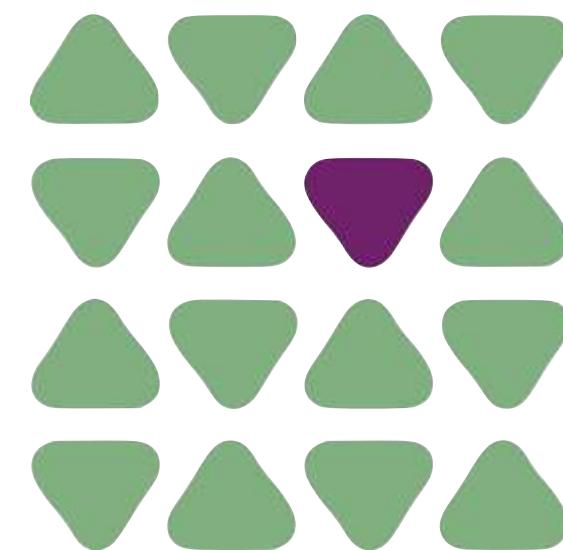
atomic



triggered



continuous



holistic









chaos monkey

SIMIAN
ARMY



chaos
gorilla

SIMIAN
ARMY



Latency
monkey



doctor
monkey

SIMIAN
ARMY



janitor
monkey

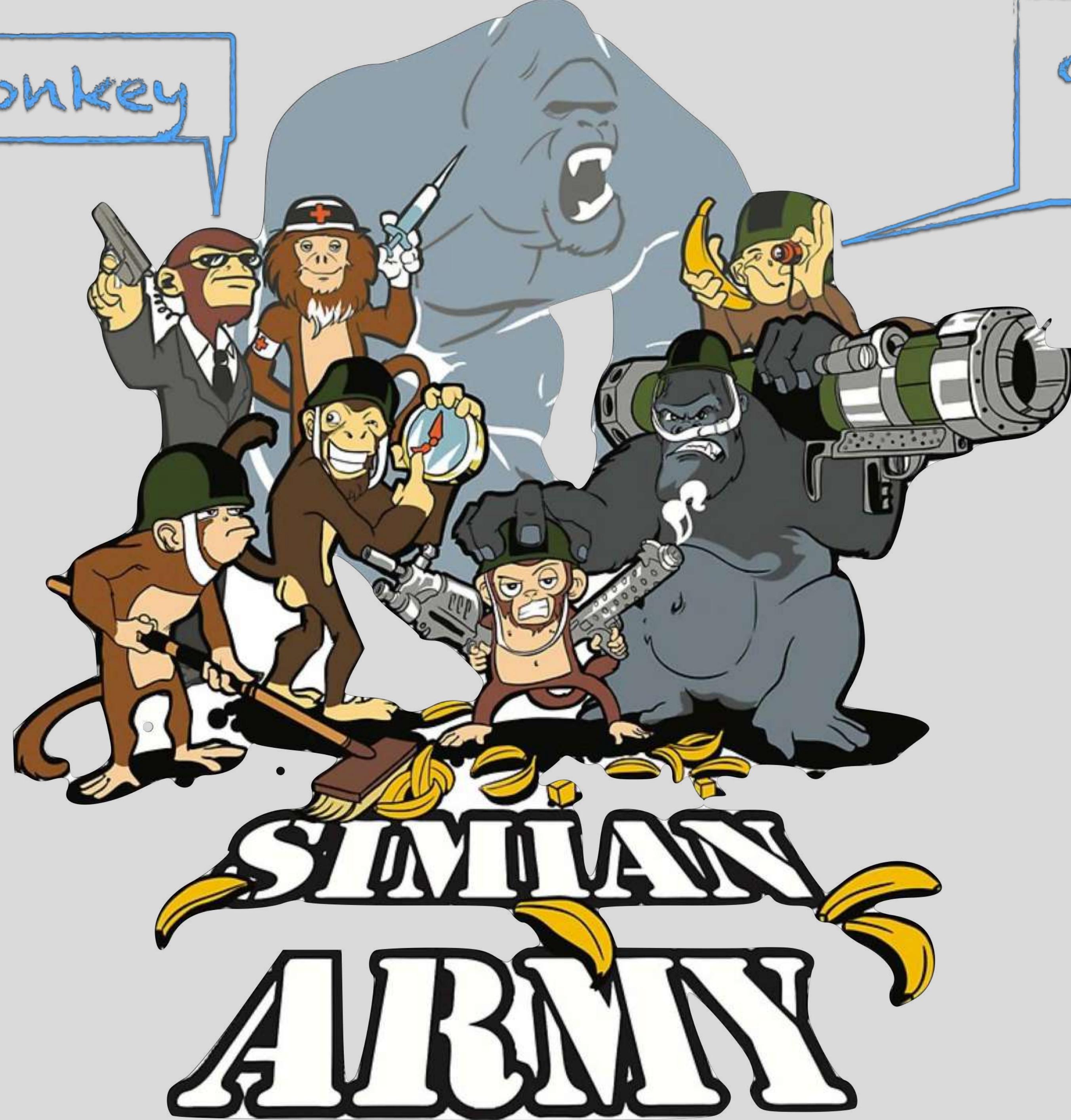
SIMIAN ARMY

conformity
monkey



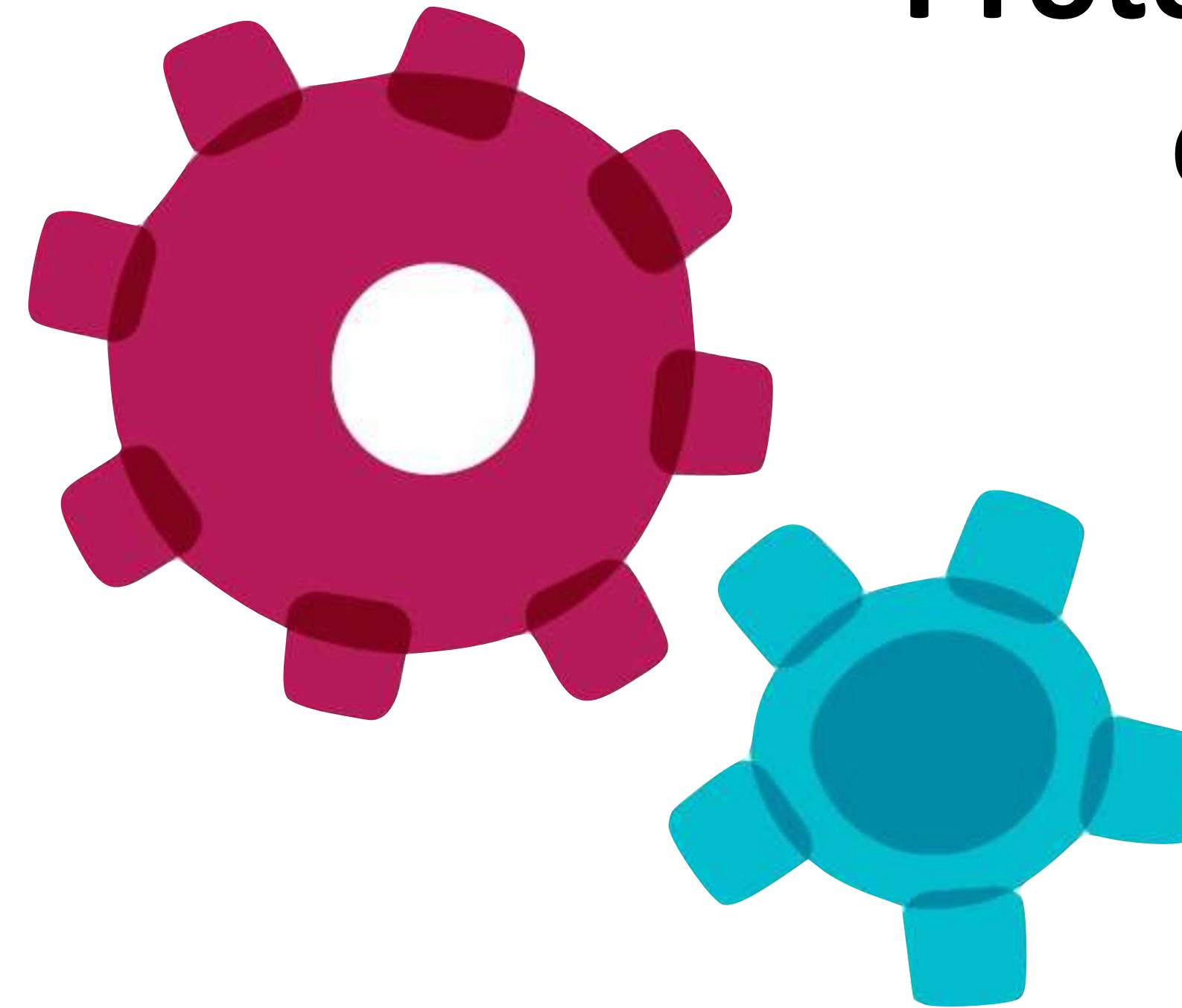
security monkey

conformity
monkey



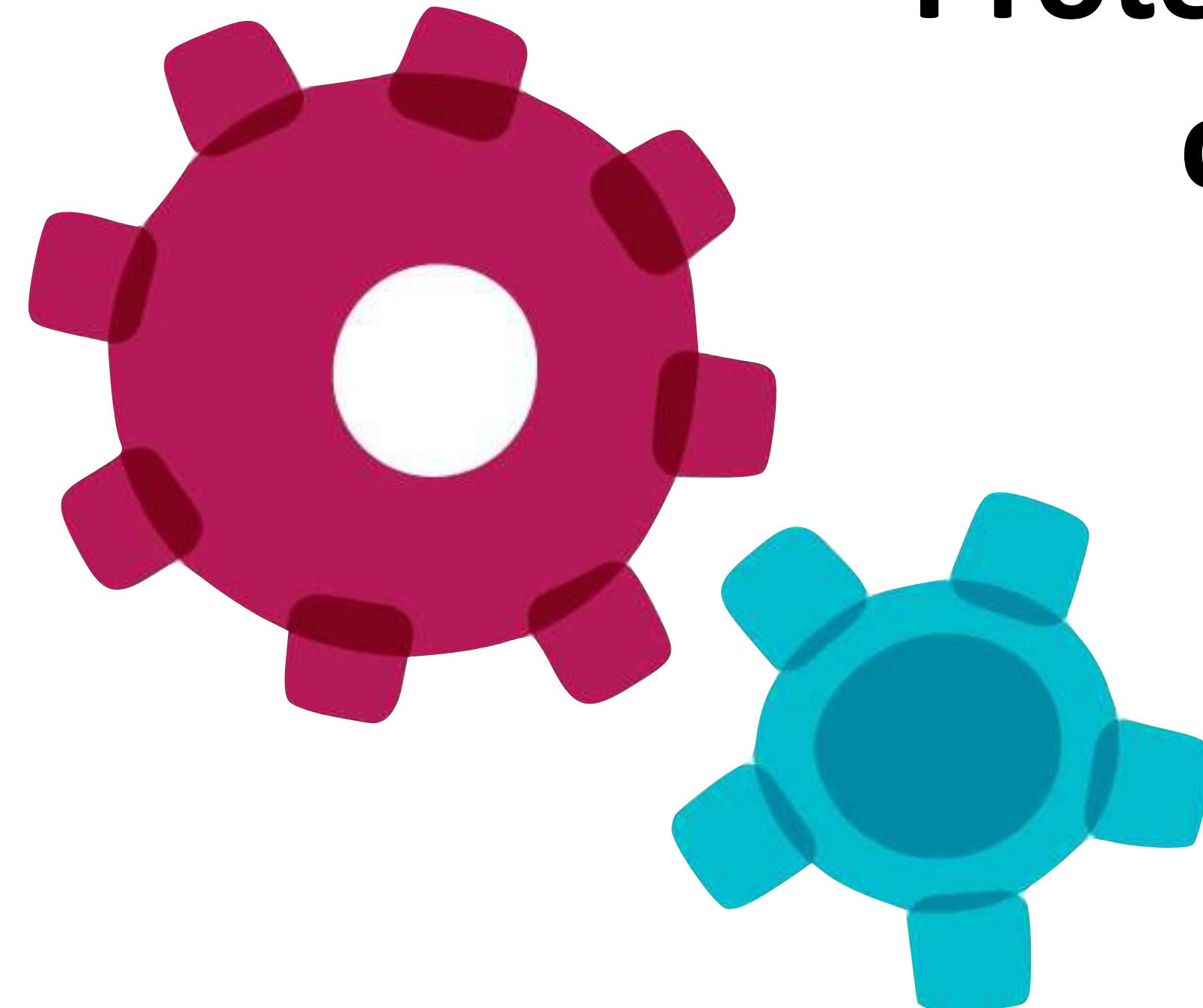
Implementing Fitness Functions

**Protecting architectural
characteristics**

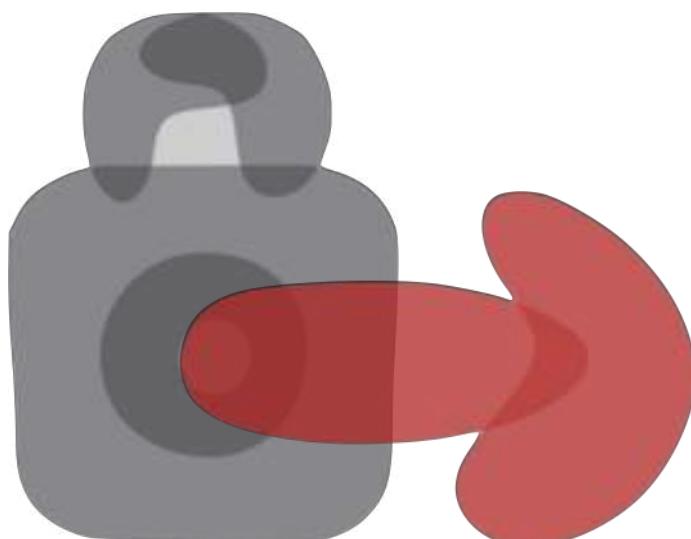


Implementing Fitness Functions

Automating governance



Protecting architectural
characteristics



maintainable?

Cyclomatic complexity < 50 for all projects

Naming conventions

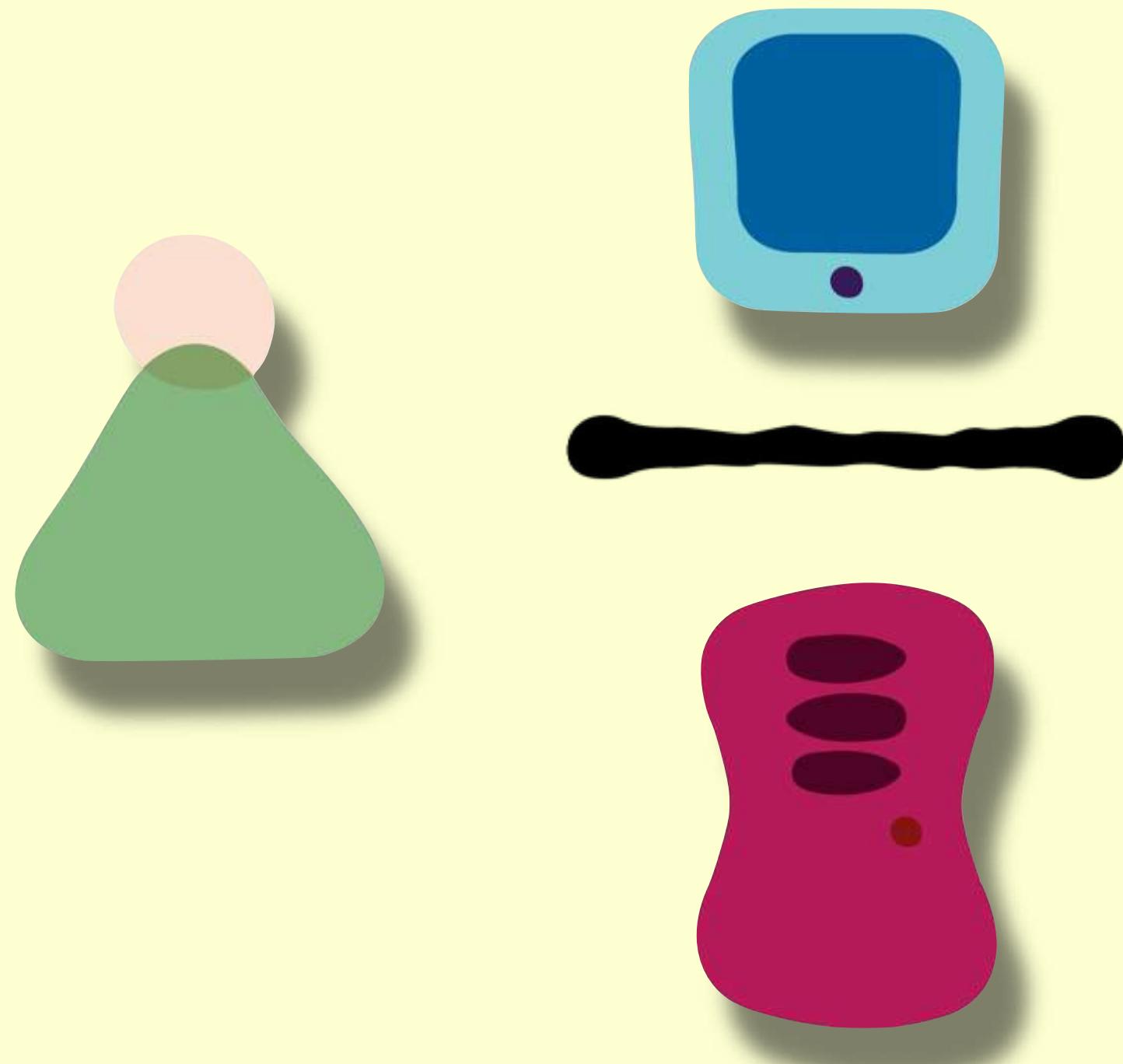
immutability

maintainable?

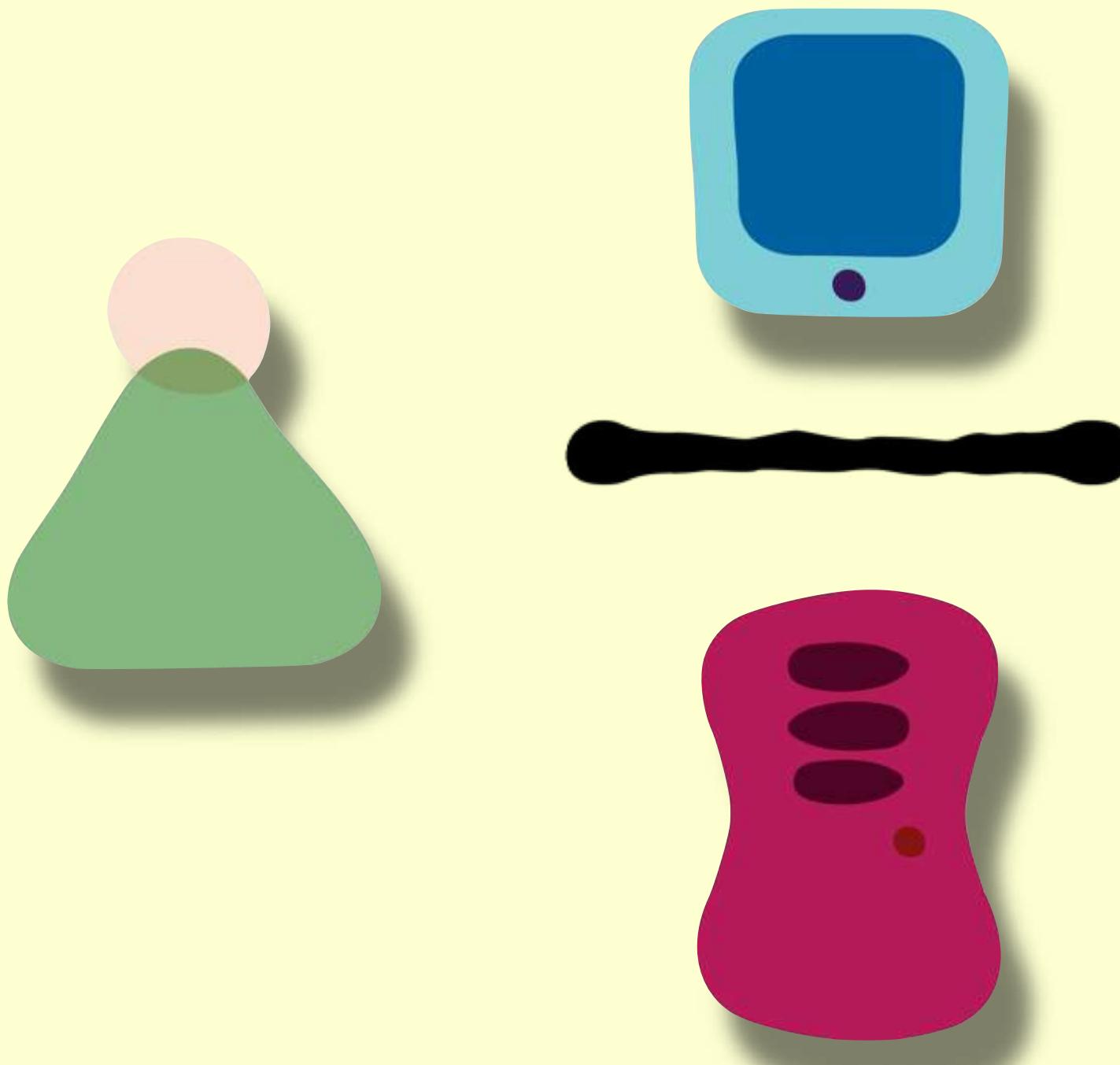
(incoming/outgoing)

Controlled afferent/efferent coupling

Governing Code Quality

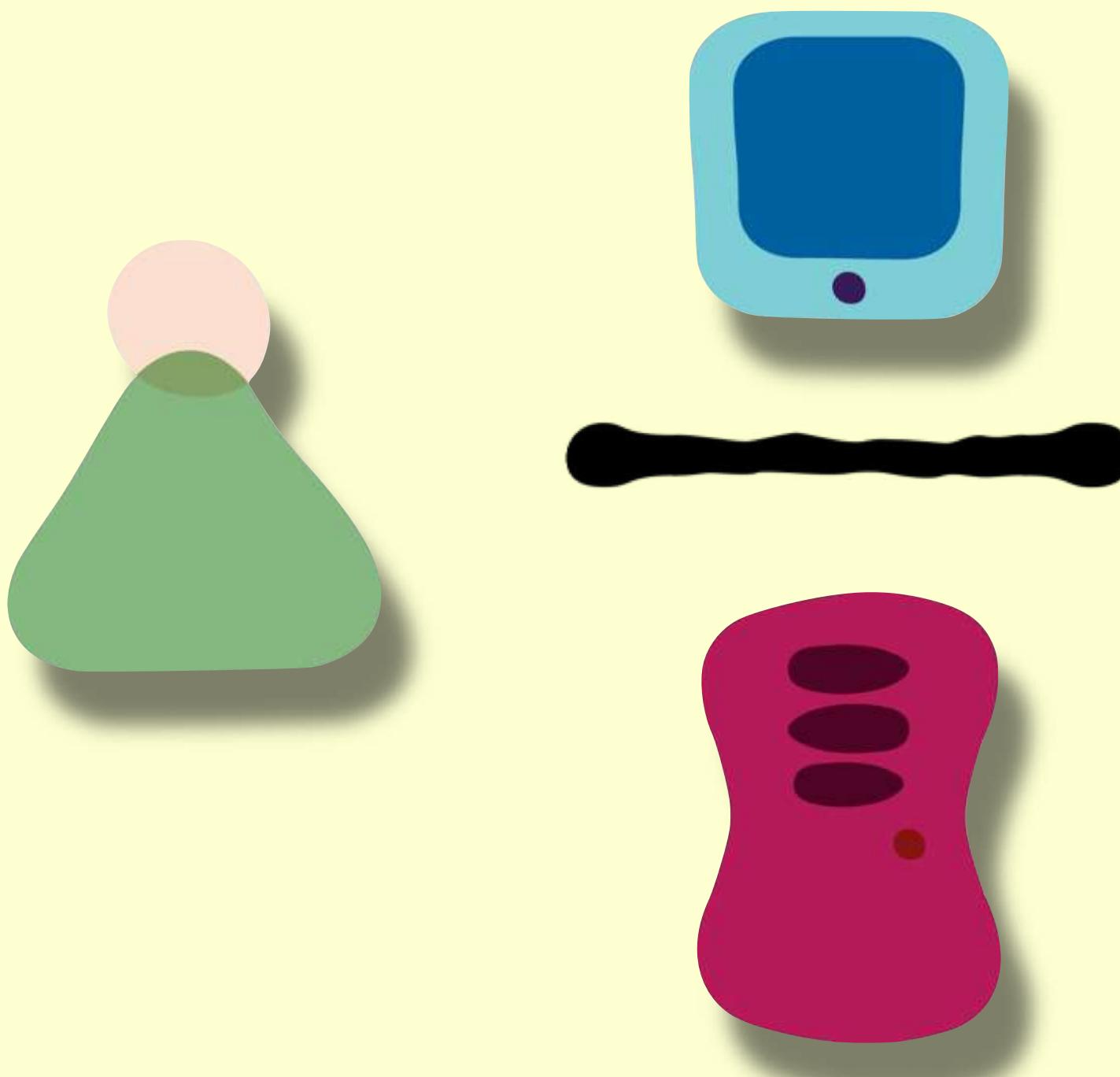


Governing Code Quality

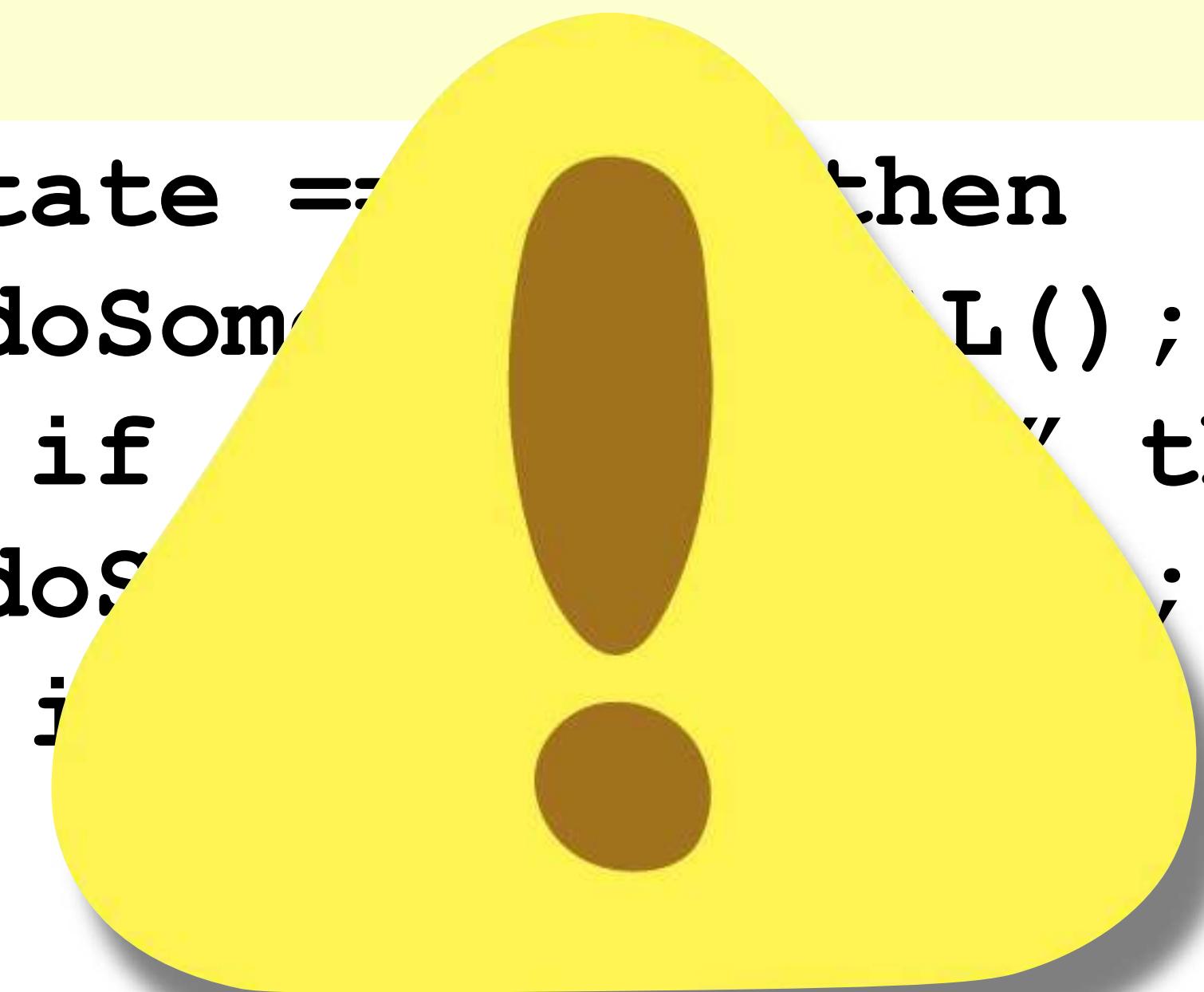


```
if state == "AL" then
    doSomethingForAL();
else if state == "GA" then
    doSomethingForGA();
else if ...
```

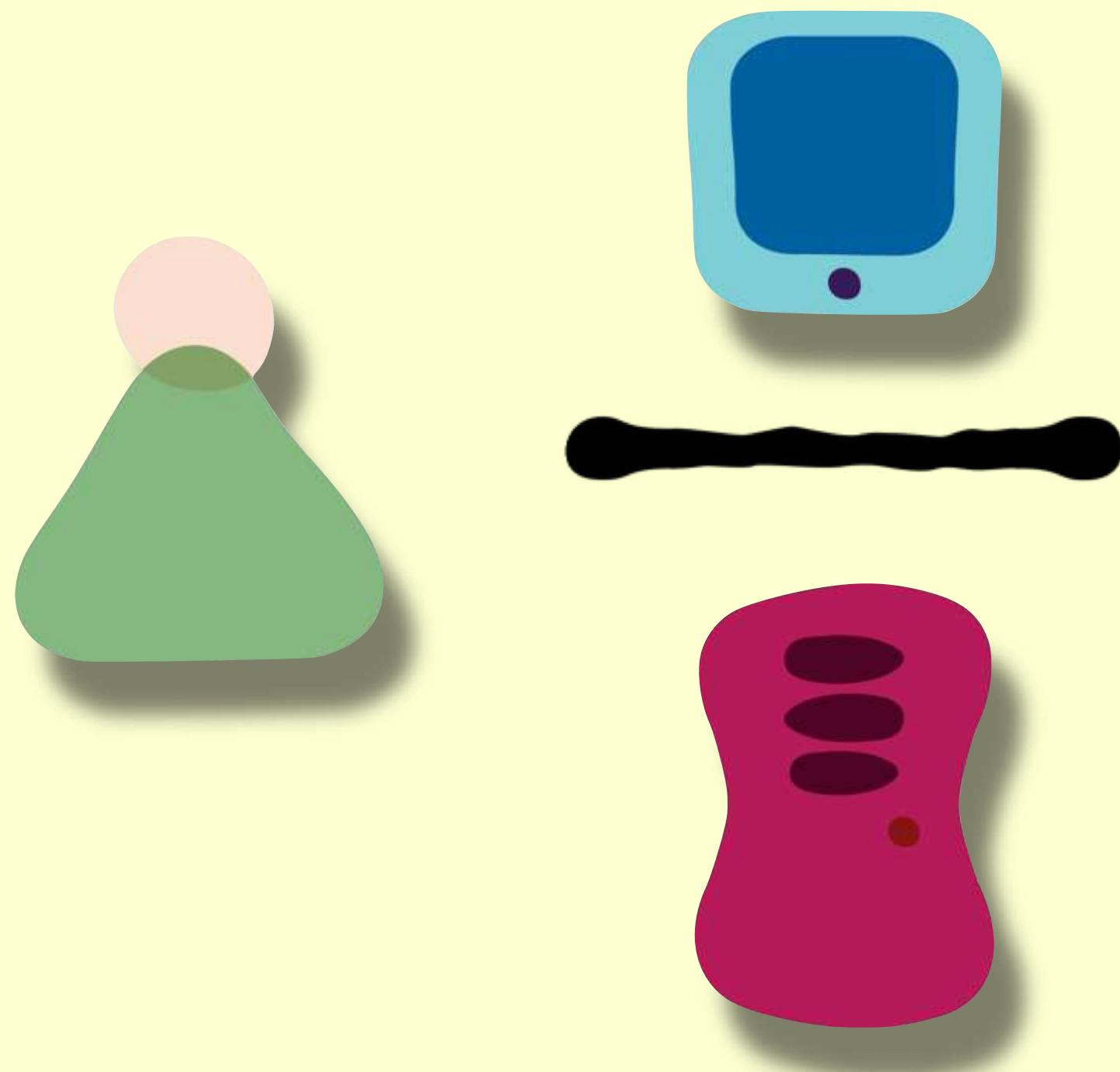
Governing Code Quality



```
if state == 'A' then
    doSomethingA();
else if state == 'B' then
    doSomethingB();
else if state == 'C' then
    doSomethingC();
```

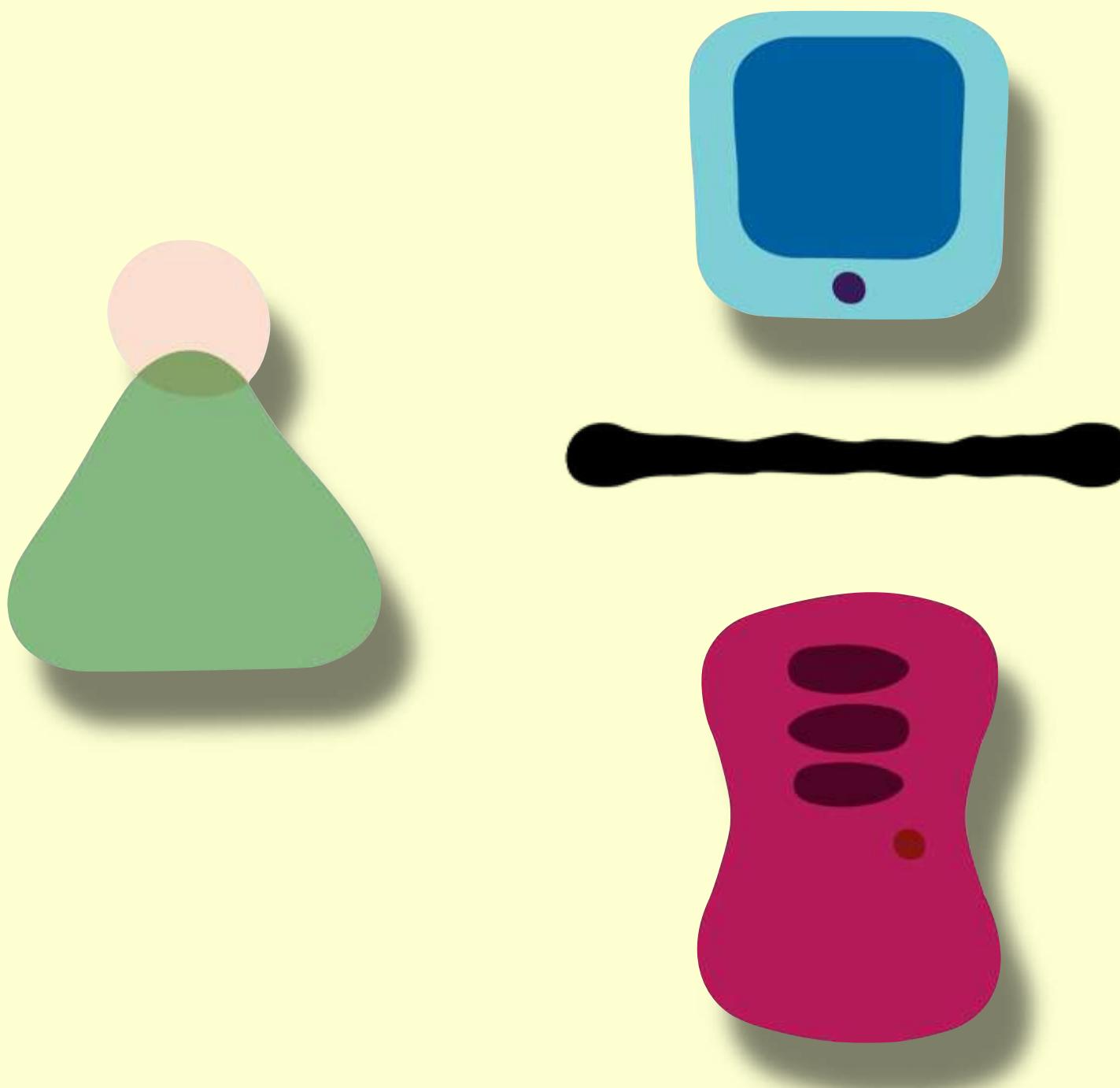


Governing Code Quality

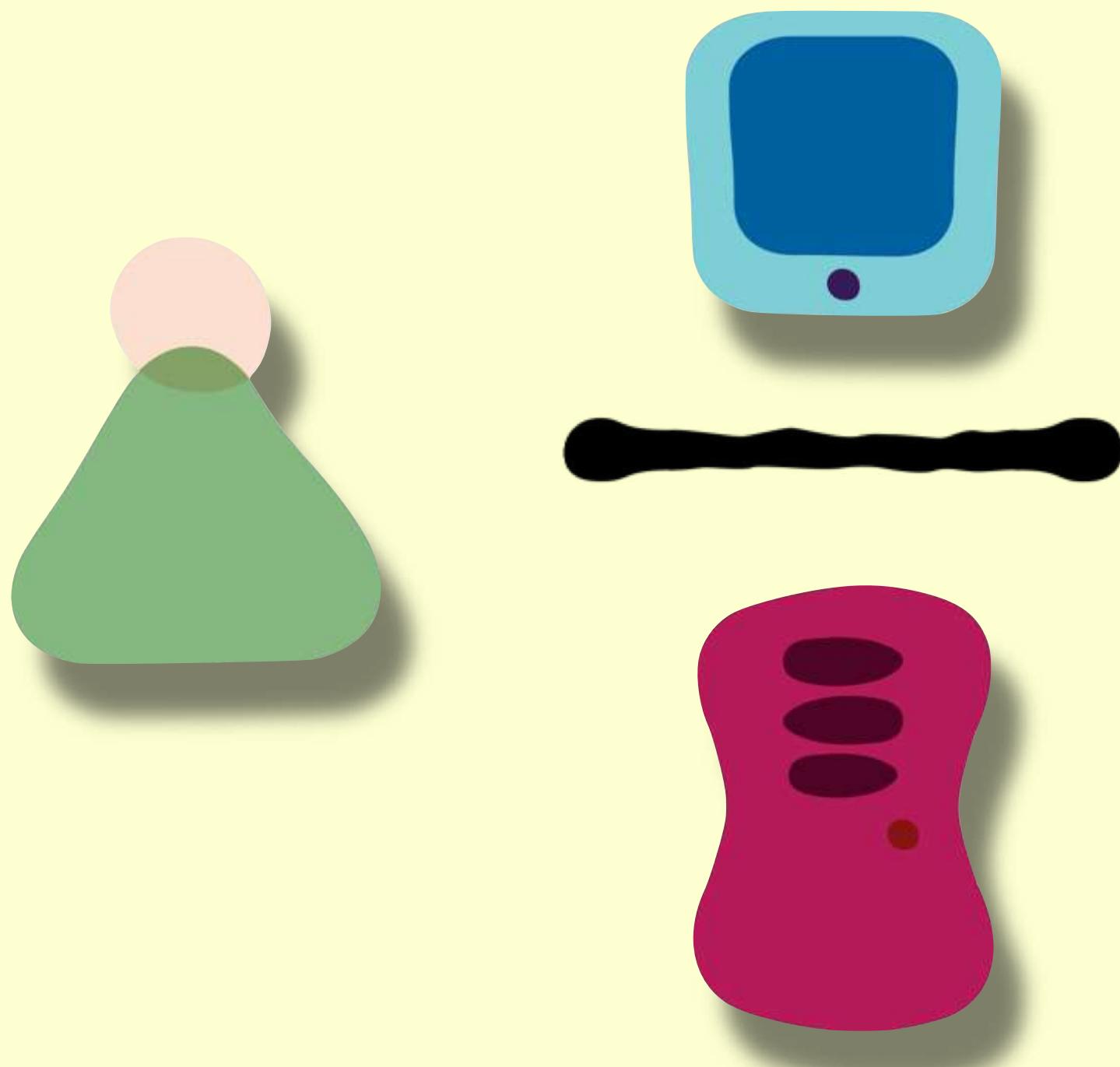


Governing Code Quality

```
if state == "AL" then
    doSomethingForAL();
else if state == "GA" then
    doSomethingForGA();
else if ...
```

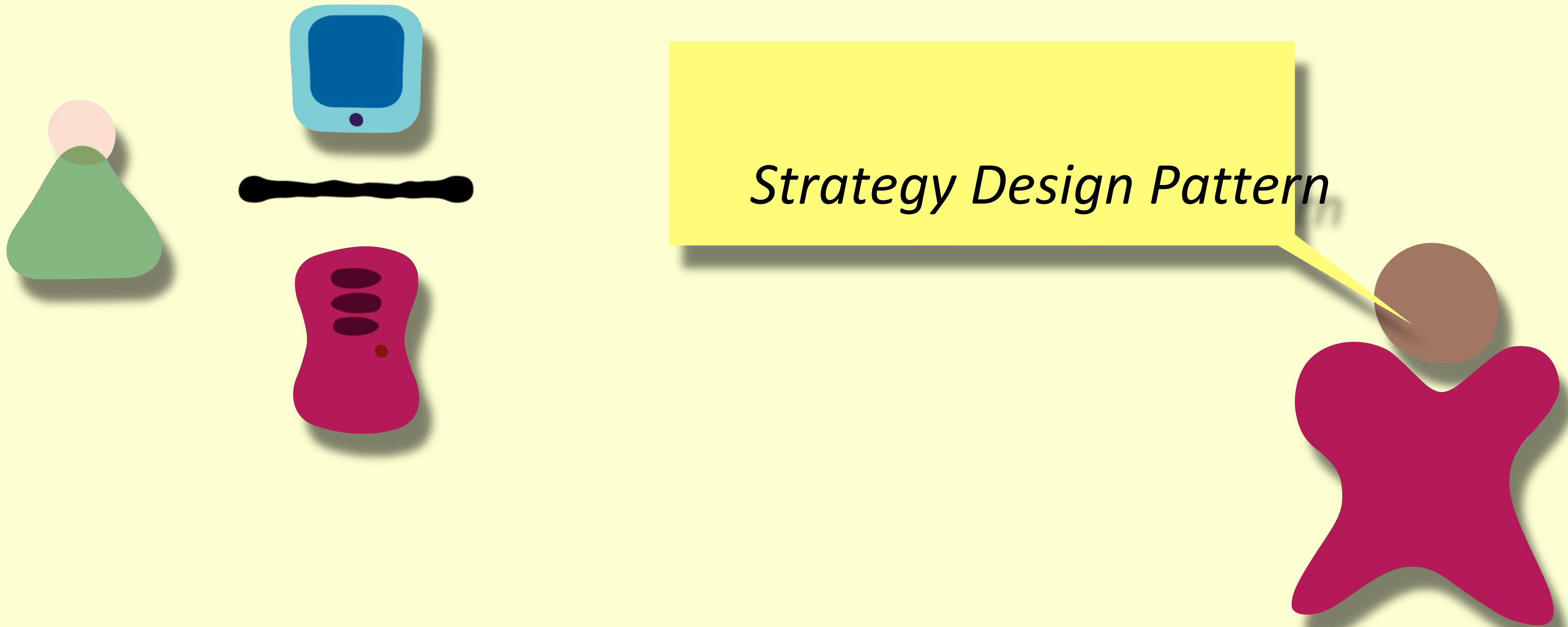


Governing Code Quality



```
if state == "A" then
    doSomethingA();
else if state == "B" then
    doSomethingB();
else if state == "C" then
    doSomethingC();
```

Governing Code Quality



ArchUnit

<https://www.archunit.org/>

The screenshot shows a web browser window displaying the ArchUnit website at https://www.archunit.org/. The page has a blue header bar with the ArchUnit logo and navigation links for Getting Started, Motivation, News, User Guide, API, and About. Below the header is a large blue banner with the text "Unit test your Java architecture" and a subtext "Start enforcing your architecture within 30 minutes using the test setup you already have." A "Start Now" button is visible. The main content area below the banner contains a detailed description of what ArchUnit is and how it works, mentioning its use of plain Java unit test frameworks and bytecode analysis. There is also a "News" section at the bottom.

Unit test your Java architecture

Start enforcing your architecture within 30 minutes using the test setup you already have.

Start Now

ArchUnit is a free, simple and extensible library for checking the architecture of your Java code using any plain Java unit test framework. That is, ArchUnit can check dependencies between packages and classes, layers and slices, check for cyclic dependencies and more. It does so by analyzing given Java bytecode, importing all classes into a Java code structure. You can find examples for the current release at [ArchUnit Examples](#) and the sources on [GitHub](#).

News

ArchUnit

<https://www.archunit.org/>

coding rules

```
import static com.tngtech.archunit.lang.syntax.ArchRuleDefinition.noClasses;
import static com.tngtech.archunit.library.GeneralCodingRules.ACCESS_STANDARD_STREAMS;
import static com.tngtech.archunit.library.GeneralCodingRules.NO_CLASSES_SHOULD_ACCESS_STANDARD_STREAMS;
import static com.tngtech.archunit.library.GeneralCodingRules.NO_CLASSES_SHOULD_THROW_GENERIC_EXCEPTIONS;
import static com.tngtech.archunit.library.GeneralCodingRules.NO_CLASSES_SHOULD_USE_JAVA_UTIL_LOGGING;

public class CodingRulesTest {
    private JavaClasses classes;

    @Before
    public void setUp() throws Exception {
        classes = new ClassFileImporter().importPackagesOf(ClassViolatingCodingRules.class);
    }

    @Test
    public void classes_should_not_access_standard_streams_defined_by_hand() {
        noClasses().should(ACCESS_STANDARD_STREAMS).check(classes);
    }

    @Test
    public void classes_should_not_access_standard_streams_from_library() {
        NO_CLASSES_SHOULD_ACCESS_STANDARD_STREAMS.check(classes);
    }

    @Test
    public void classes_should_not_throw_generic_exceptions() {
        NO_CLASSES_SHOULD_THROW_GENERIC_EXCEPTIONS.check(classes);
    }

    @Test
    public void classes_should_not_use_java_util_logging() {
        NO_CLASSES_SHOULD_USE_JAVA_UTIL_LOGGING.check(classes);
    }
}
```

ArchUnit

<https://www.archunit.org/>

```
@Test  
public void classes_should_not_access_standard_streams_from_library() {  
    NO_CLASSES_SHOULD_ACCESS_STANDARD_STREAMS.check(classes);  
}
```

```
@Test  
public void classes_should_not_throw_generic_exceptions() {  
    NO_CLASSES_SHOULD_THROW_GENERIC_EXCEPTIONS.check(classes);  
}
```

```
@Test  
public void classes_should_not_use_java_util_logging() {  
    NO_CLASSES_SHOULD_USE_JAVA_UTIL_LOGGING.check(classes);  
}
```

coding rules

ArchUnit

<https://www.archunit.org/>

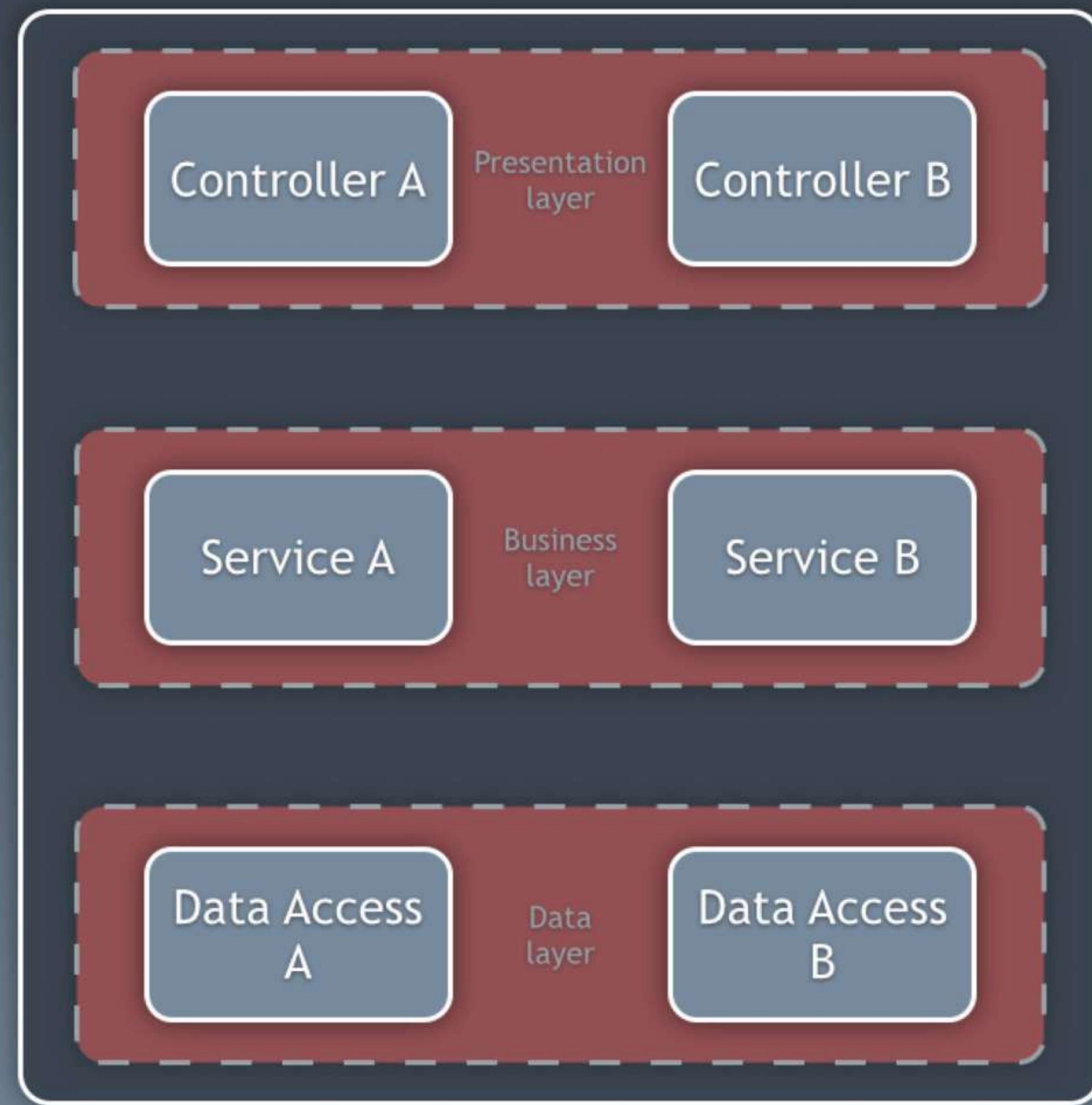
interface rules

```
public class InterfaceRules {  
  
    @Test  
    public void interfaces_should_not_have_names_ending_with_the_word_interface() {  
        JavaClasses classes = new ClassFileImporter().importClasses(  
            SomeBusinessInterface.class,  
            SomeDao.class  
        );  
  
        noClasses().that().areInterfaces().should().haveNameMatching(".*Interface").check(classes);  
    }  
  
    @Test  
    public void interfaces_should_not_have_simple_class_names_ending_with_the_word_interface() {  
        JavaClasses classes = new ClassFileImporter().importClasses(  
            SomeBusinessInterface.class,  
            SomeDao.class  
        );  
  
        noClasses().that().areInterfaces().should().haveSimpleNameContaining("Interface").check(classes);  
    }  
  
    @Test  
    public void interfaces_must_not_be_placed_in_implementation_packages() {  
        JavaClasses classes = new ClassFileImporter().importPackagesOf(SomeInterfacePlacedInTheWrongPackage.class);  
  
        noClasses().that().resideInAPackage("..impl..").should().beInterfaces().check(classes);  
    }  
}
```

ArchUnit

<https://www.archunit.org/>

```
public class InterfaceRules {  
  
    @Test  
    public void interfaces_should_not_have_names_ending_with_the_word_interface() {  
        JavaClasses classes = new ClassFileImporter().importClasses(  
            SomeBusinessInterface.class,  
            SomeDao.class  
        );  
  
        noClasses().that().areInterfaces().should().haveNameMatching(".*Interface")  
    }  
  
    @Test  
    public void interfaces_should_not_have_simple_class_names_ending_with_the_word
```



Package by layer (horizontal slicing)

ArchUnit

<https://www.archunit.org/>

```
public class LayerDependencyRulesTest {  
    private JavaClasses classes;  
  
    @Before  
    public void setUp() throws Exception {  
        classes = new ClassFileImporter().importPackagesOf(ClassViolatingCodingRules.class);  
    }  
  
    @Test  
    public void services_should_not_access_controllers() {  
        noClasses().that().resideInAPackage("..service..")  
            .should().accessClassesThat().resideInAPackage("..controller..").check(classes);  
    }  
  
    @Test  
    public void persistence_should_not_access_services() {  
        noClasses().that().resideInAPackage("..persistence..")  
            .should().accessClassesThat().resideInAPackage("..service..").check(classes);  
    }  
  
    @Test  
    public void services_should_only_be_accessed_by_controllers_or_other_services() {  
        classes().that().resideInAPackage("..service..")  
            .should().onlyBeAccessed().byAnyPackage("..controller..", "..service..").check(classes);  
    }  
}
```

layer dependency

```
private JavaClasses classes;

@Before
public void setUp() throws Exception {
    classes = new ClassFileImporter().importPackagesOf(ClassViolatingCodingRules.class);
}

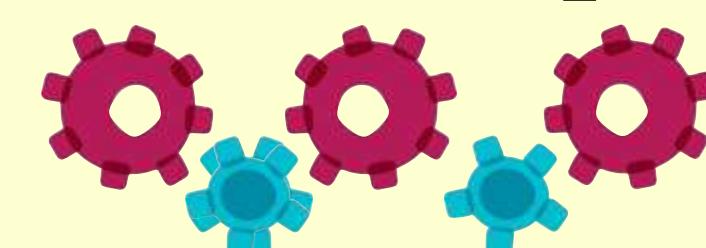
@Test
public void services_should_not_access_controllers() {
    noClasses().that().resideInAPackage(..service..)
        .should().accessClassesThat().resideInAPackage(..controller..).check(classes);
}

@Test
public void persistence_should_not_access_services() {
    noClasses().that().resideInAPackage(..persistence..)
        .should().accessClassesThat().resideInAPackage(..service..).check(classes);
}
```

Legality of Open Source Libraries



Penultima ↑ e



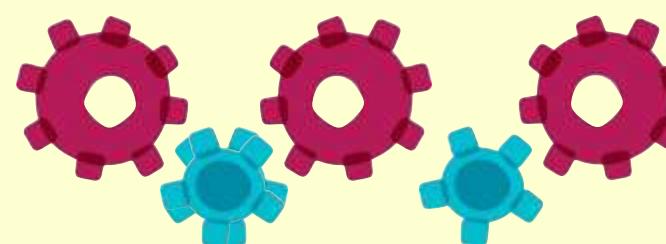
Legality of Open Source Libraries



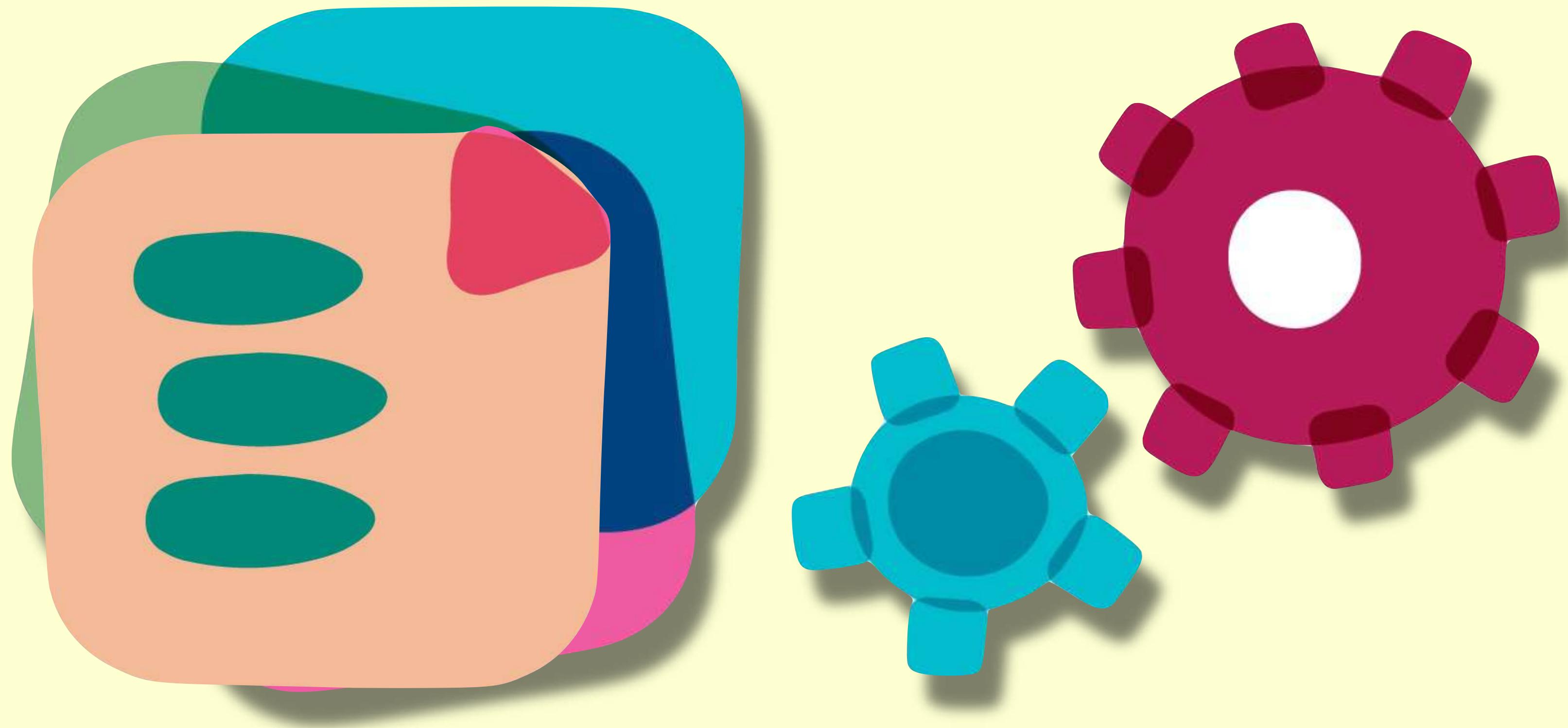
Penultima ↑ e

Legality of Open Source Libraries



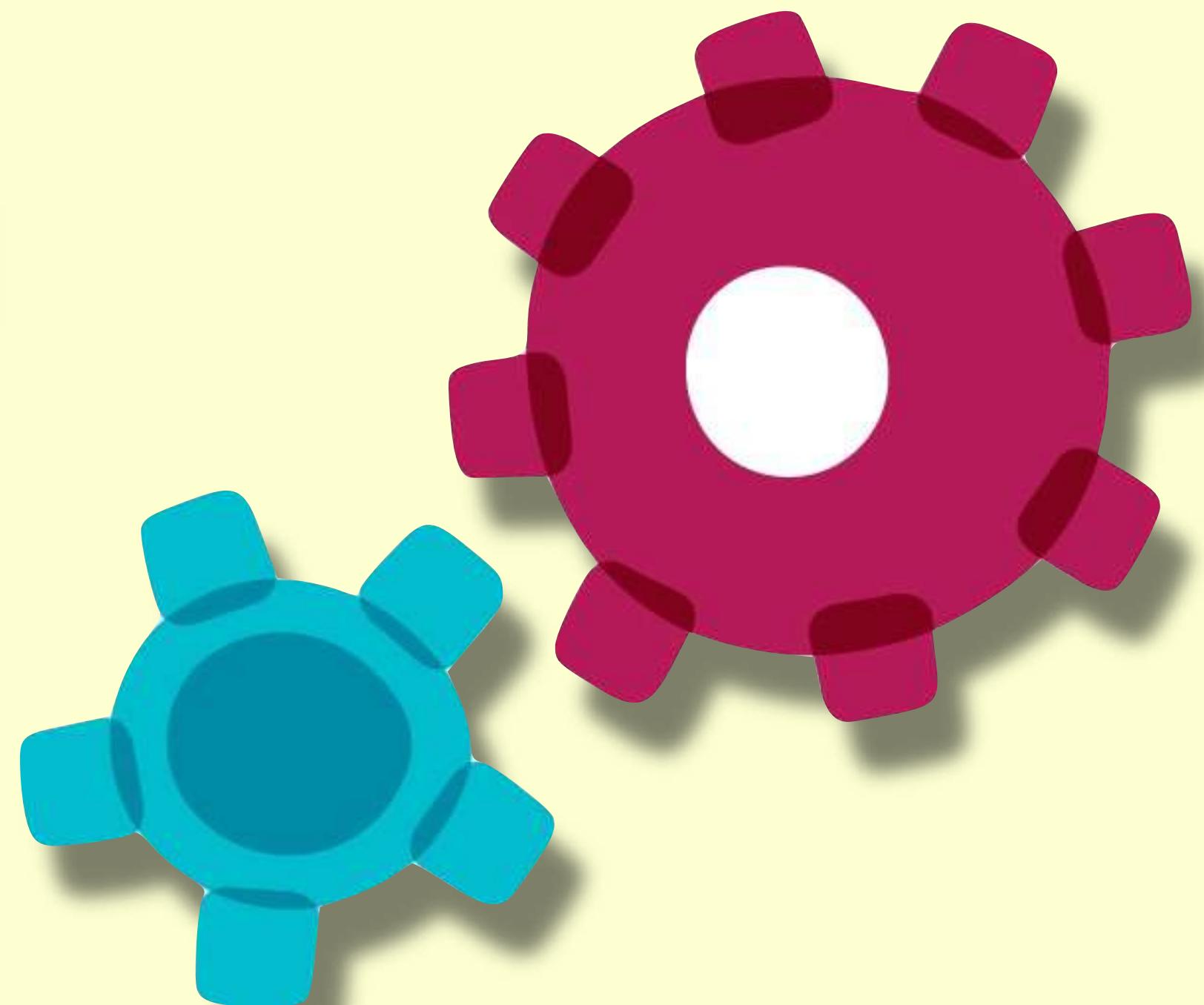
Penultima ↑ e
↑


Legality of Open Source Libraries

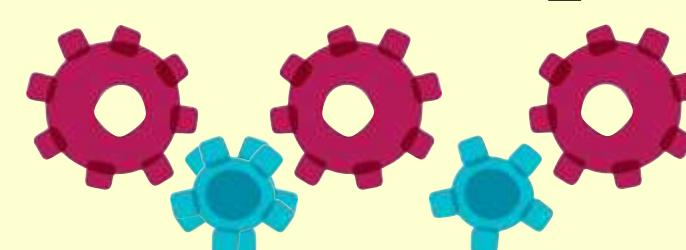


Penultima ↑ e
A row of three small gears, alternating in color between blue and pink, with an upward-pointing arrow above the text "Penultima ↑ e".

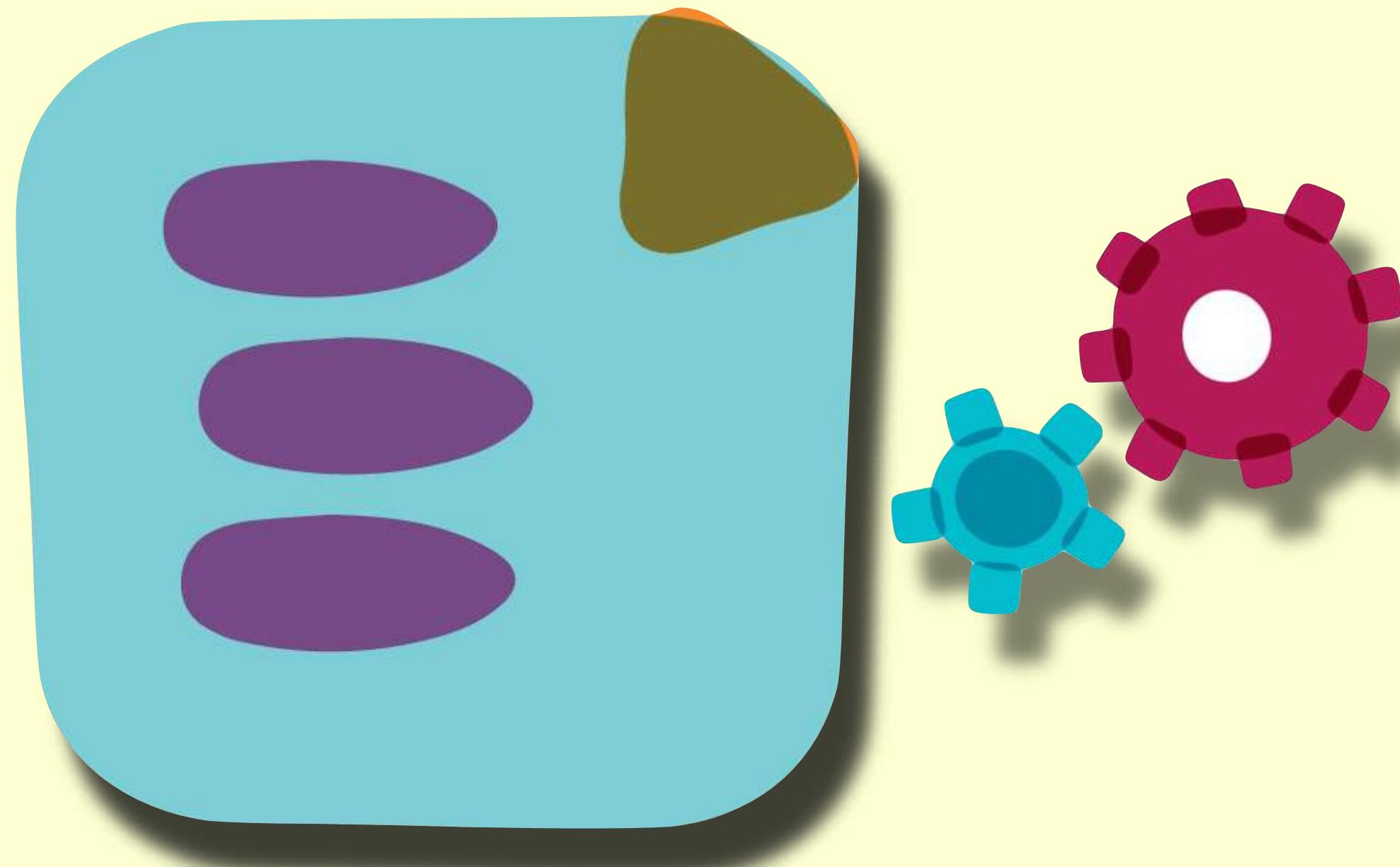
Legality of Open Source Libraries



Penultima ↑ e

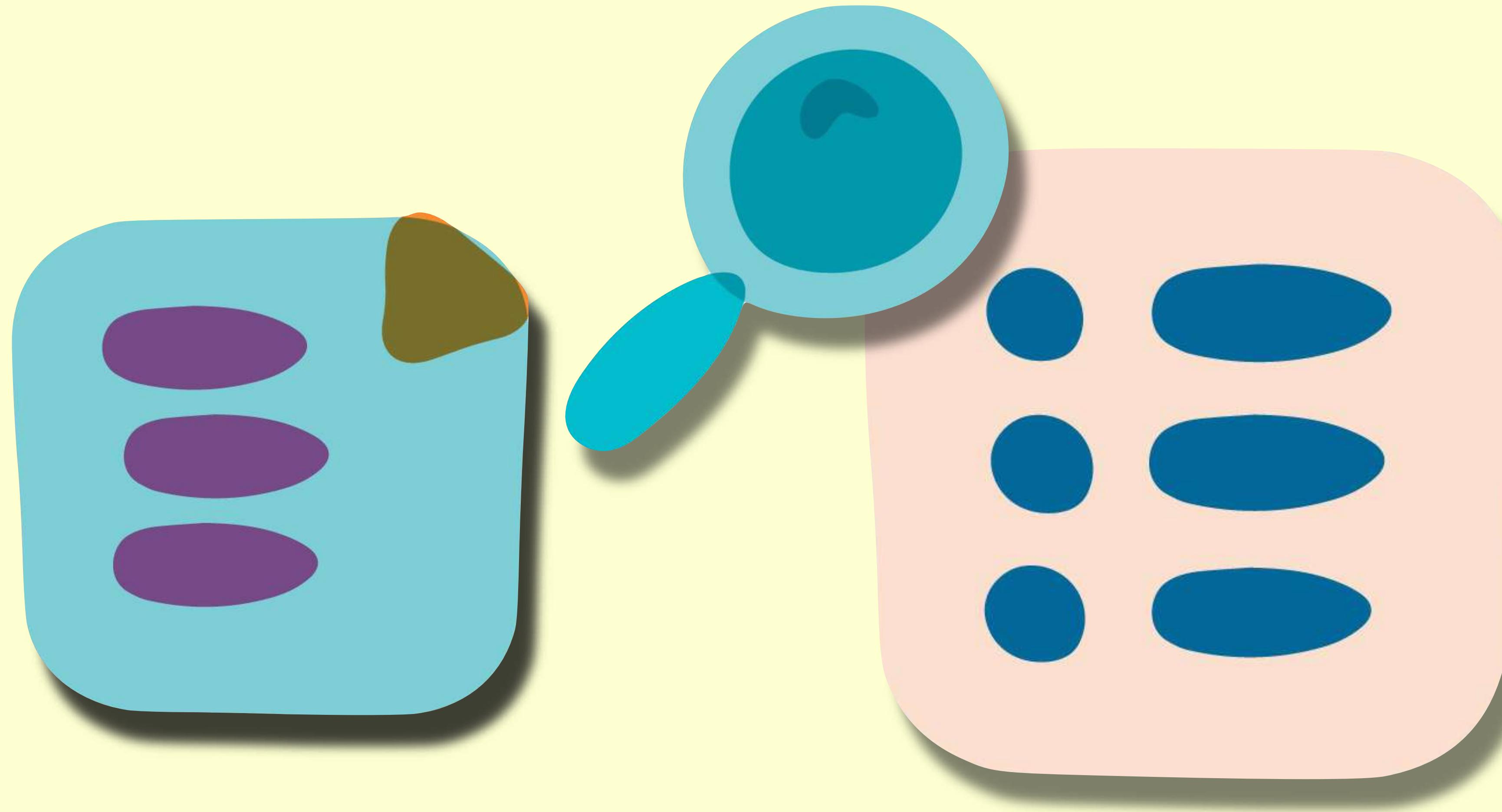


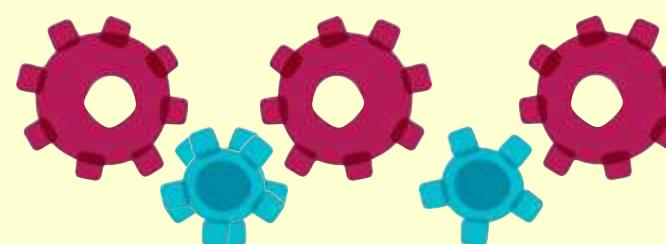
Legality of Open Source Libraries



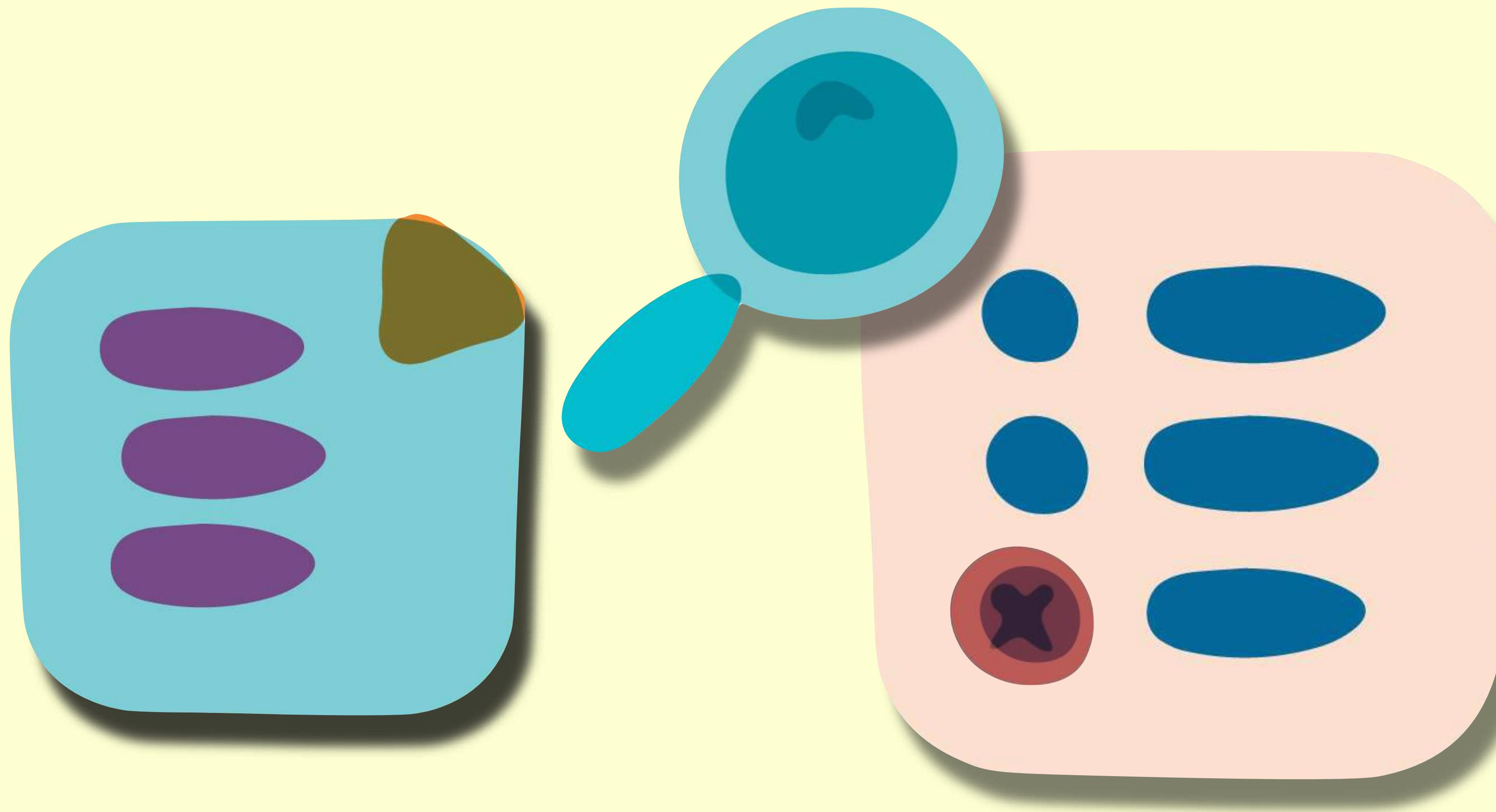
Penultima ↑ e

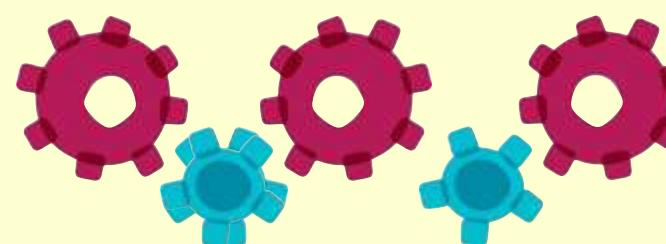

Legality of Open Source Libraries



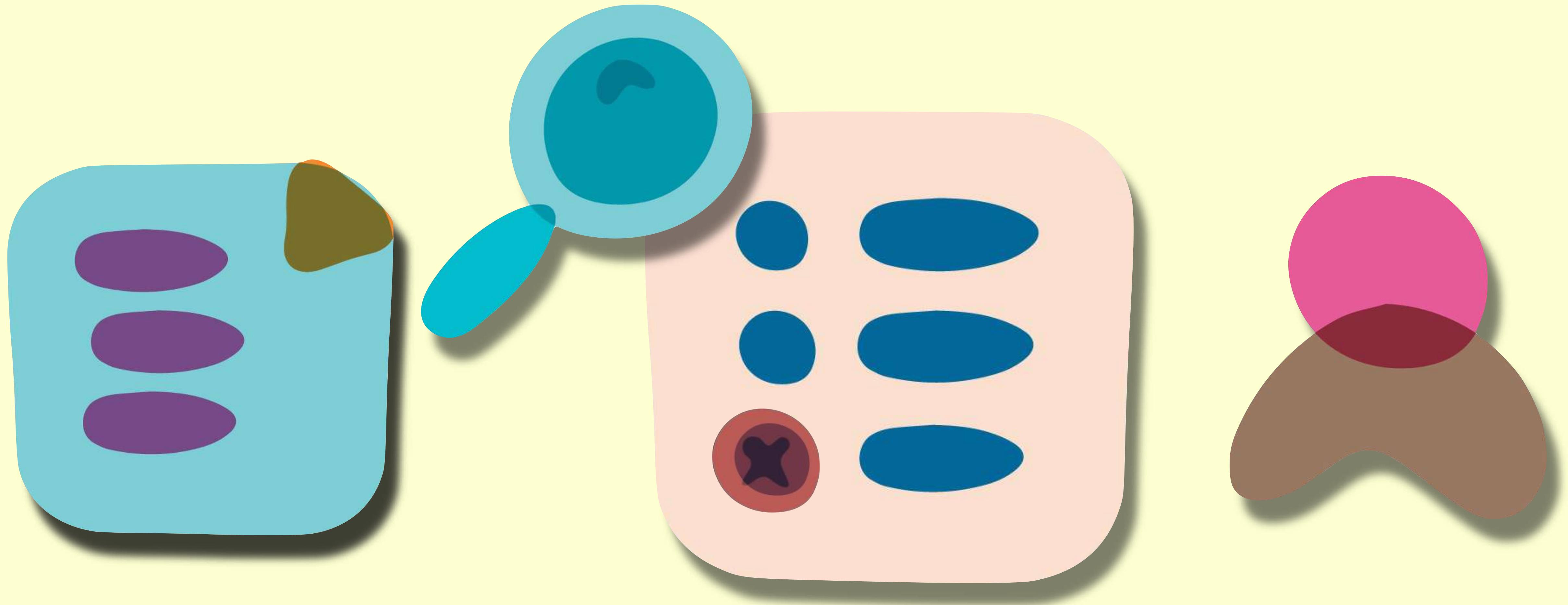
Penultima ↑ e
↑


Legality of Open Source Libraries

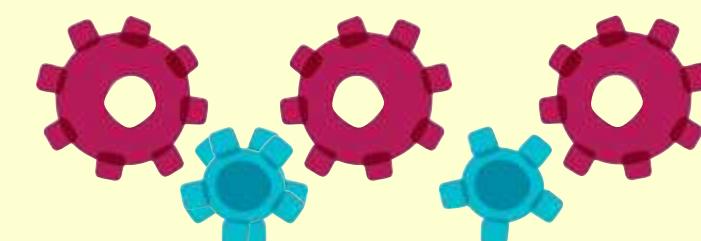


Penultima ↑ e


Legality of Open Source Libraries



Penultima ↑ e



Fitness Function Guidelines

most fitness function are local

Fitness Function Guidelines

most fitness function are local

global only when
universally applied.

Fitness Function Guidelines

most fitness function are local

global only when
universally applied.

No EA

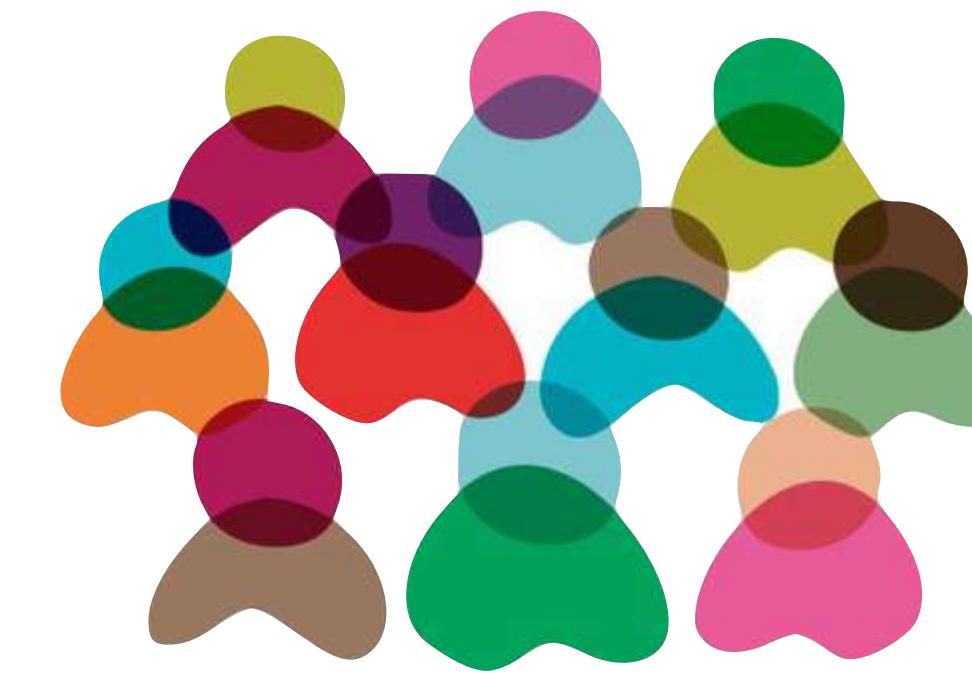
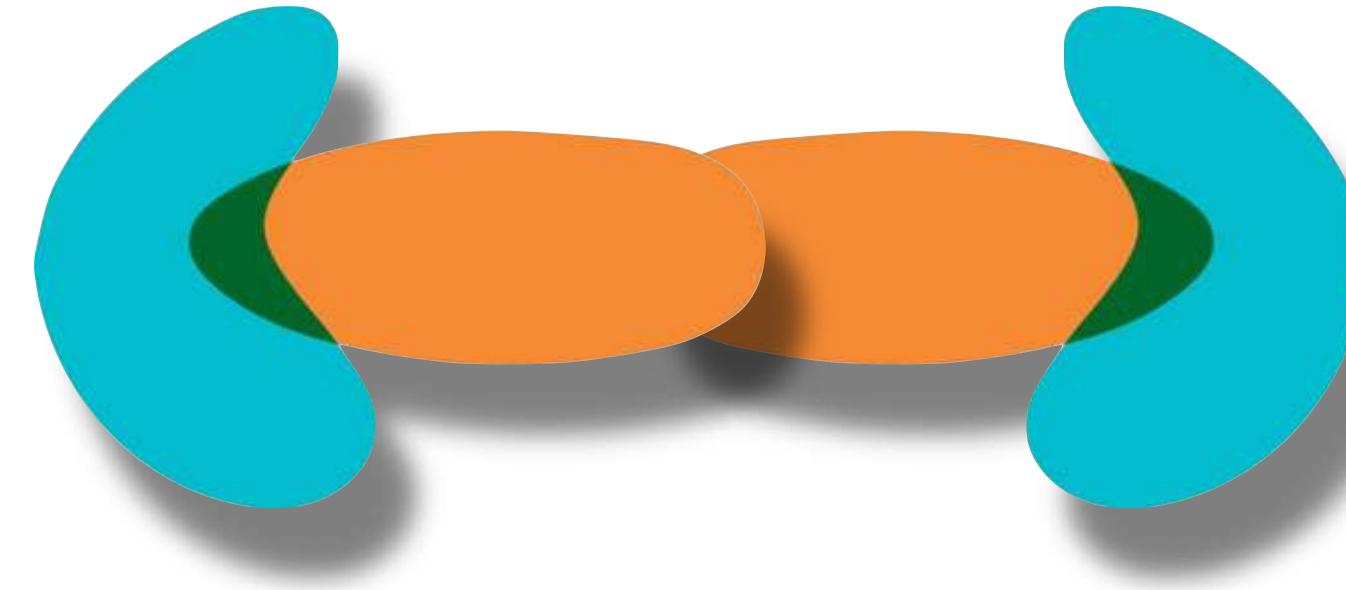
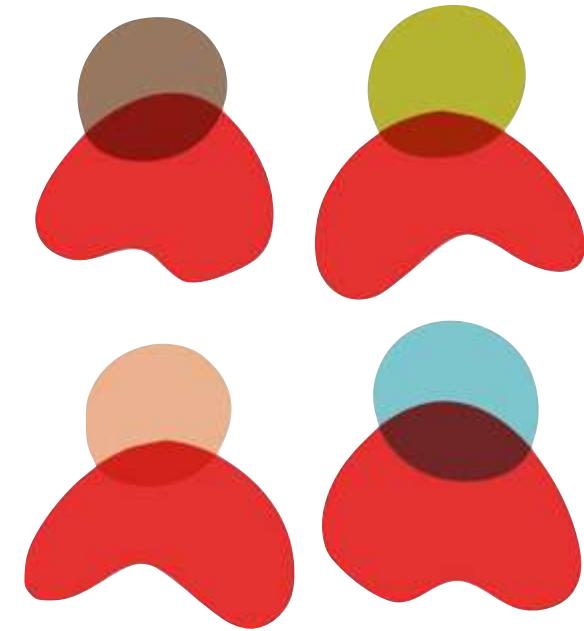


states!

Fitness Function Guidelines

developers must collaborate

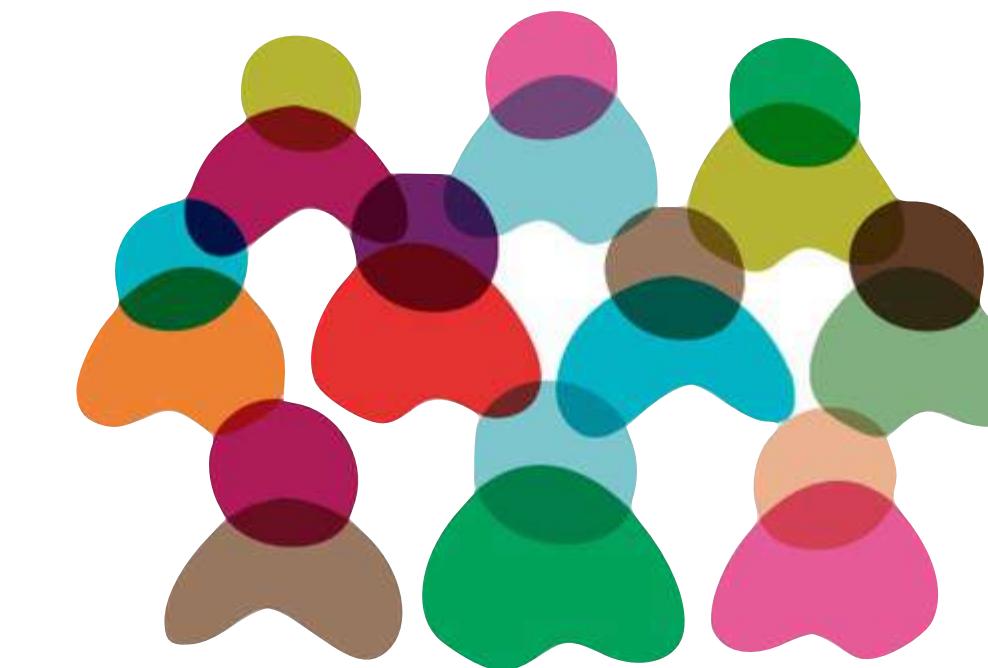
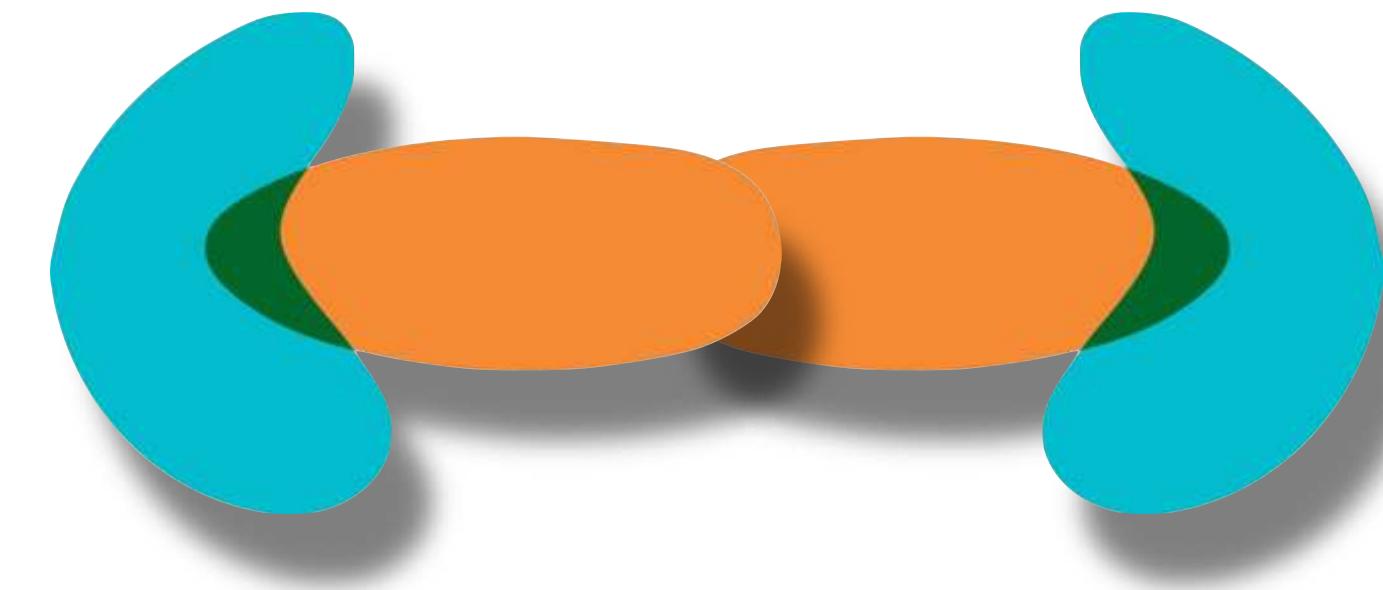
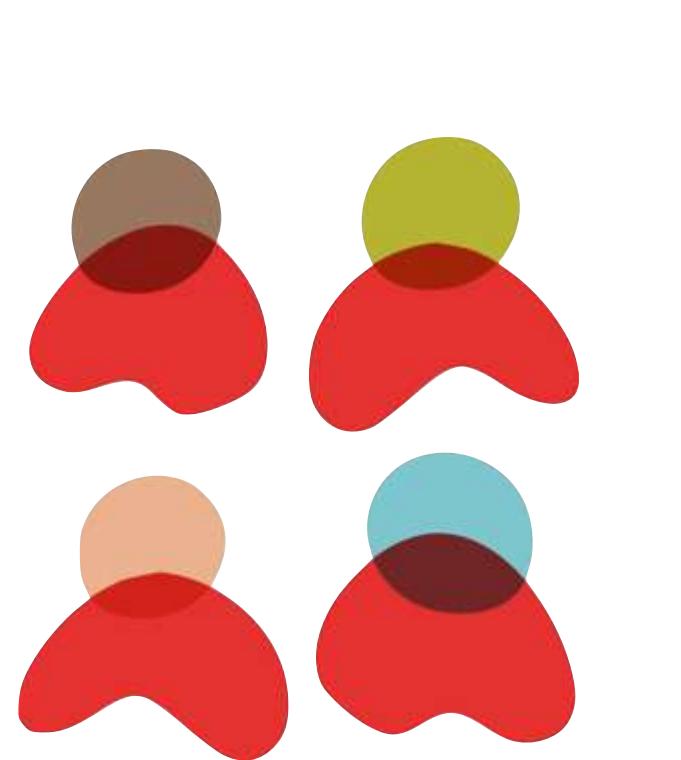
enterprise
architects



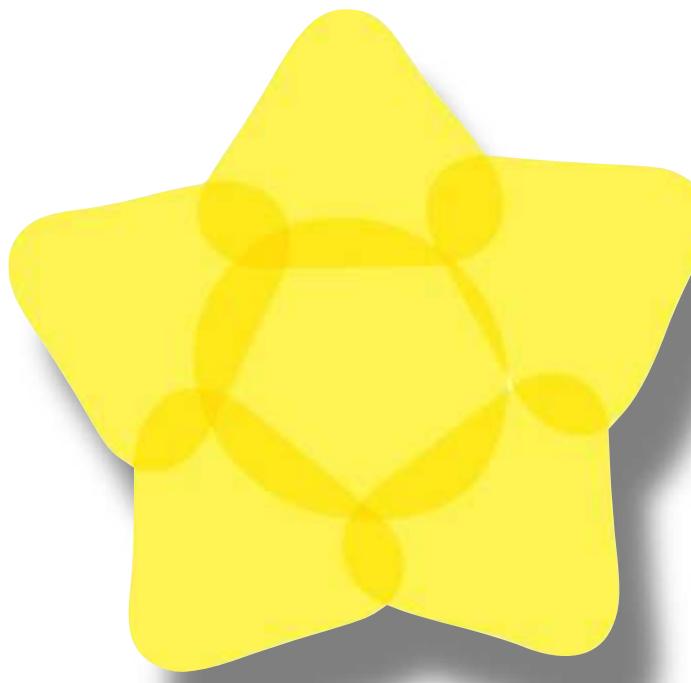
developers

Fitness Function Guidelines

developers must collaborate



No EA states!



Fitness Function Guidelines

prefer automation but
rely on manual when needed

Fitness Function Guidelines

prefer automation but
rely on manual when needed

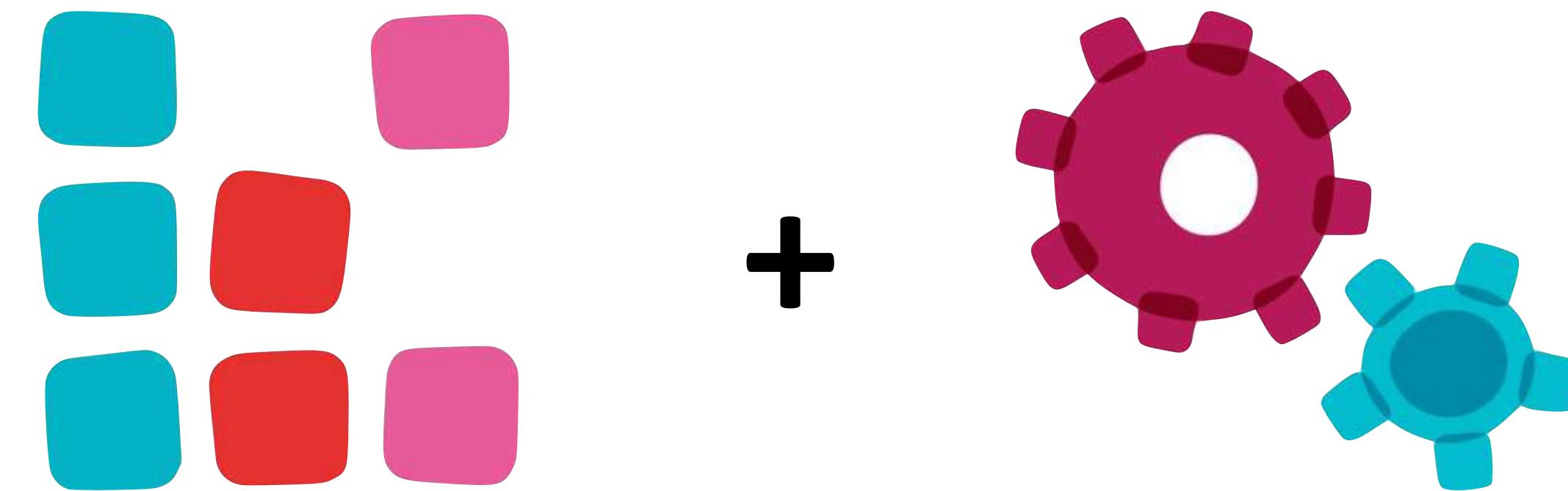
automation =



when needed

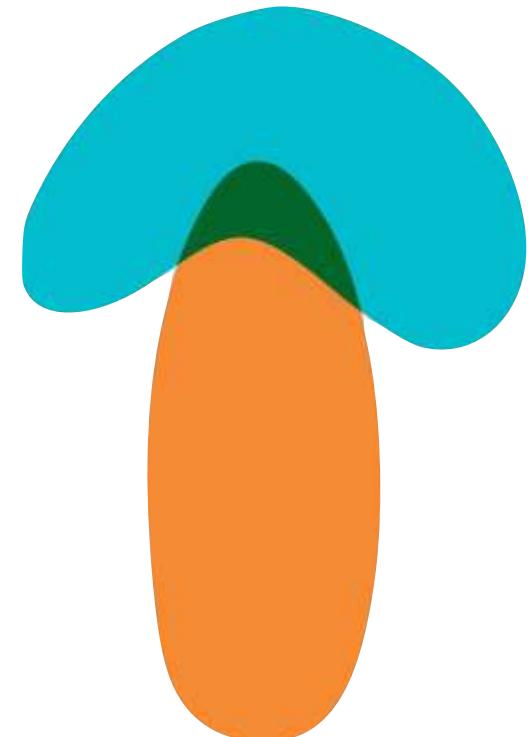
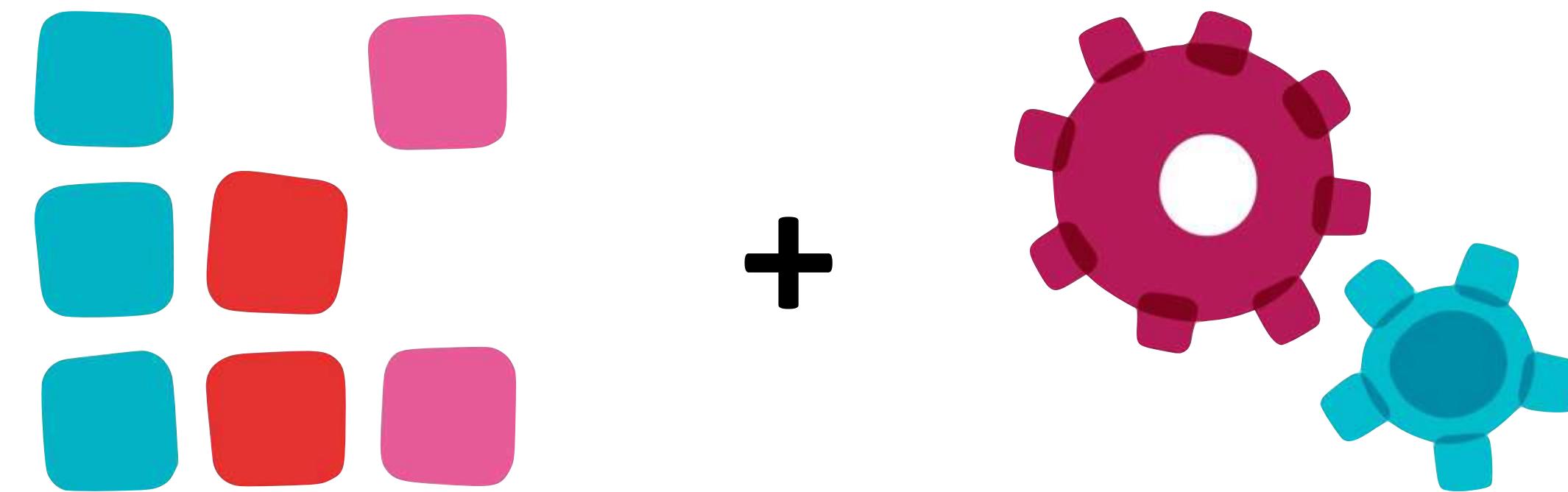
Architecture Defined

architecture =



Architecture Defined

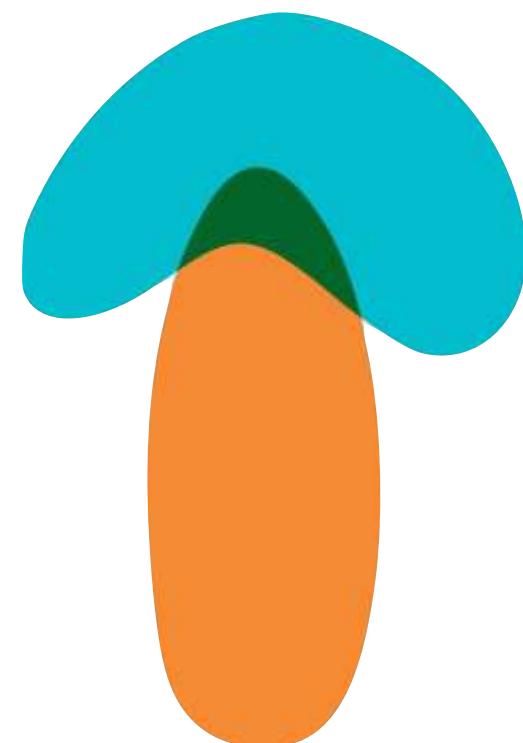
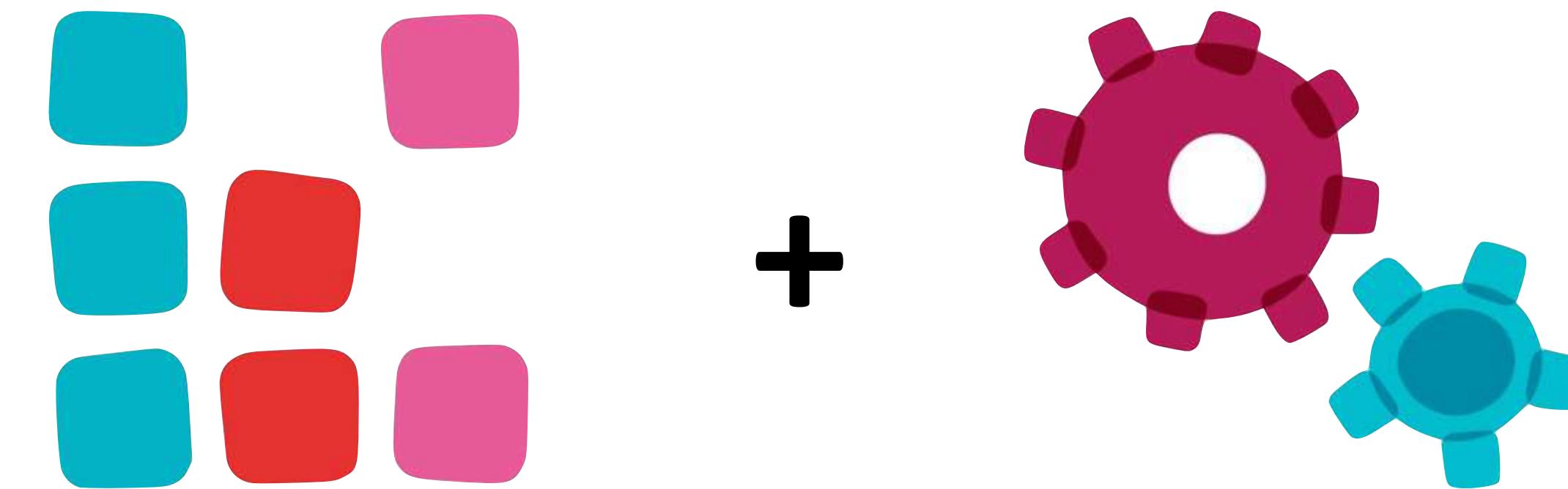
architecture =



prioritization

Architecture Defined

architecture =



prioritization



cost

Question:

How do teams doing continuous deployment handle things like release branches?

Question:

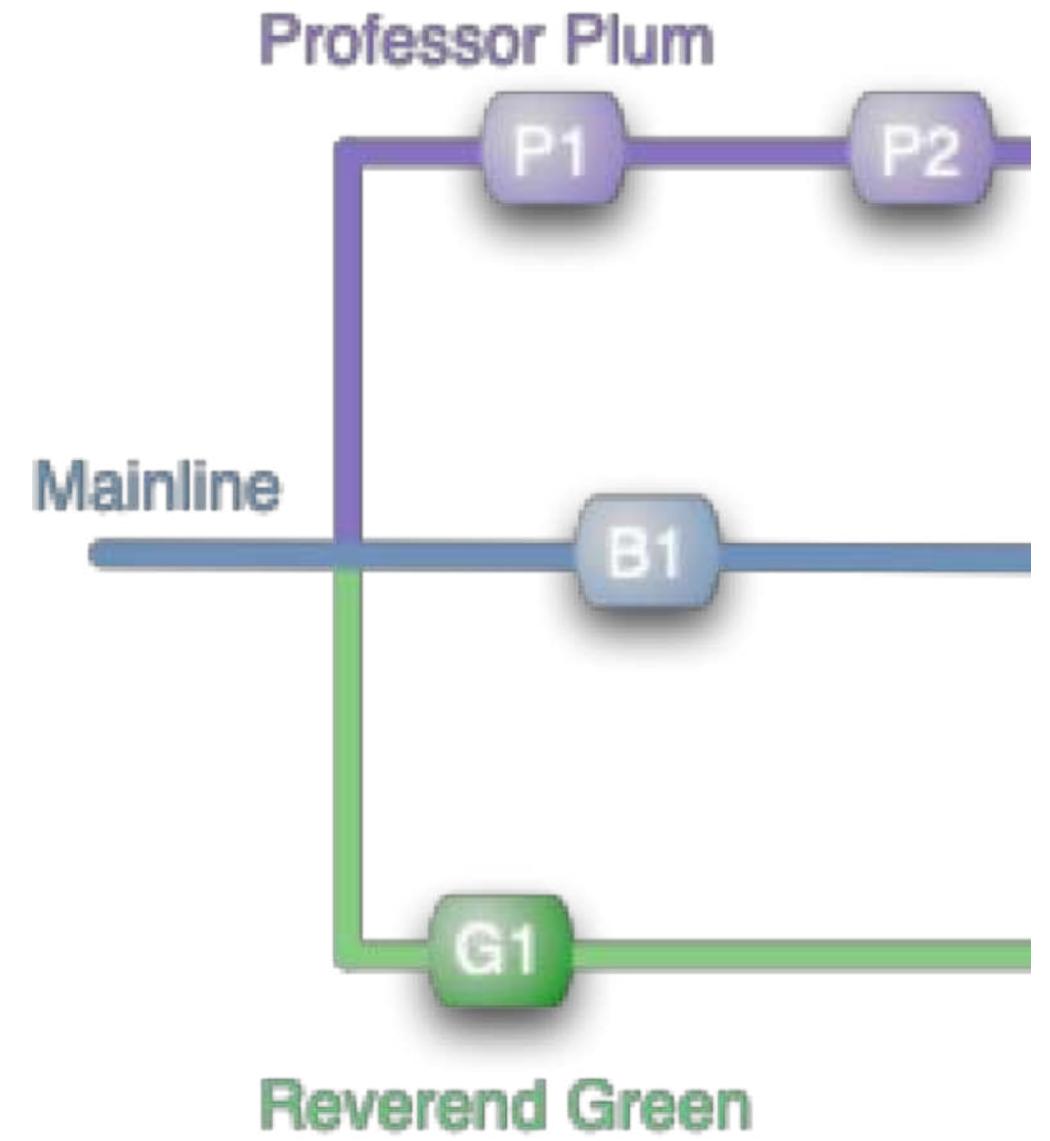
How can teams support advanced architecture tricks like A/B testing while doing continuous delivery?

Question:

How can teams make structural changes to architecture
(continuously!) without breaking other parts by
unexpected side effects?!?

Question:

How do teams reduce risk when doing continuous delivery/deployment?

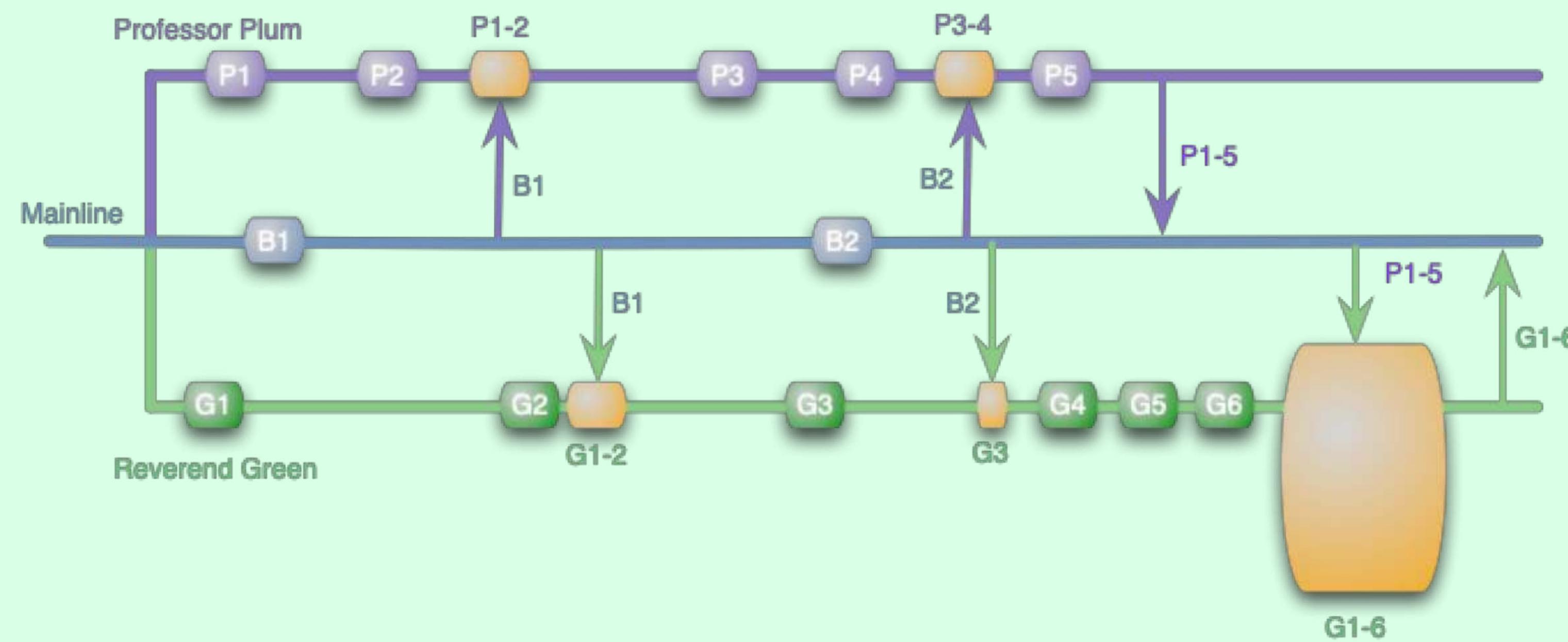


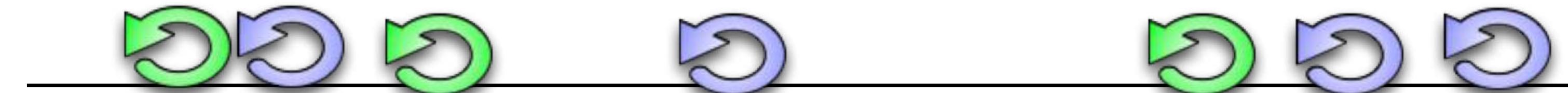
*merge
ambush!*

Feature Branching

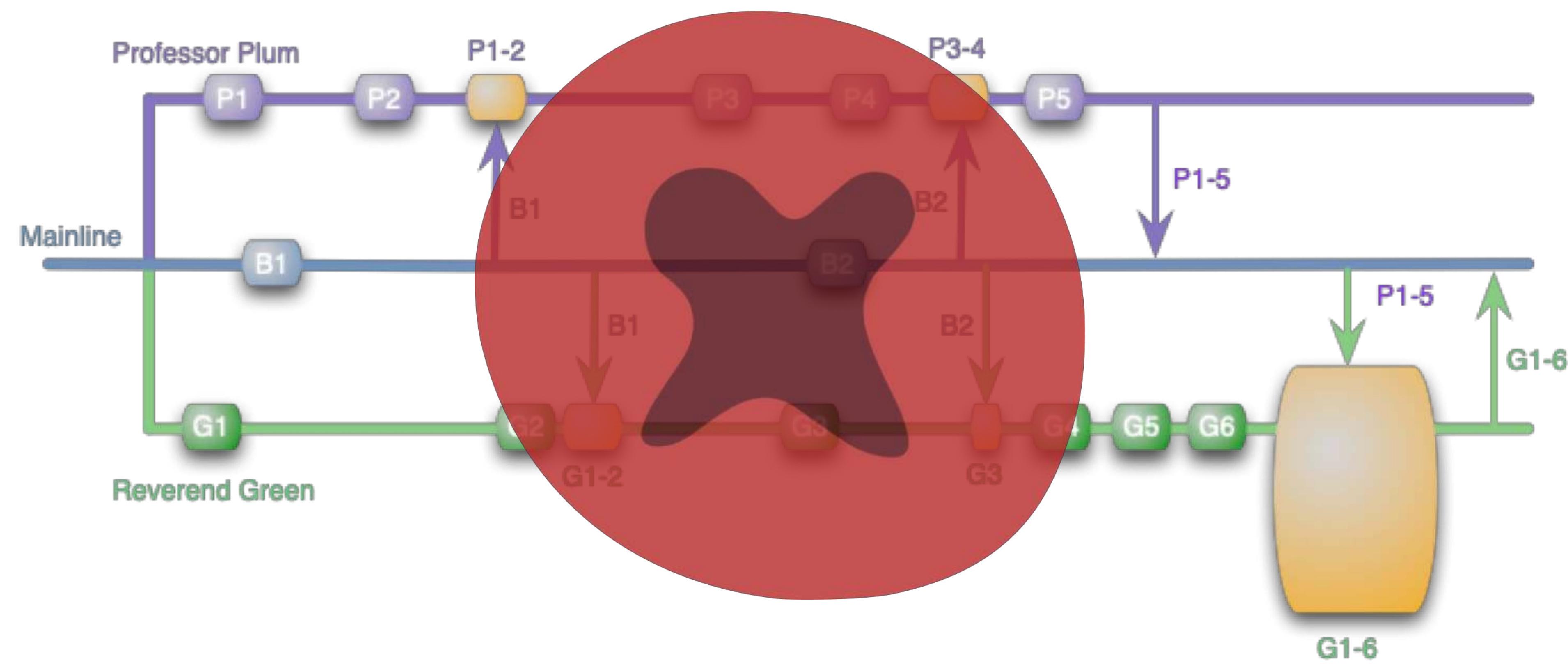


trunk-based development *versus* *feature branching*





trunk-based development



Trunk Based Development: Introduction

The screenshot shows the homepage of the Trunk Based Development website. The header includes the title "Trunk Based Development: Introduction" and the URL "trunkbaseddevelopment.com". The left sidebar contains a navigation menu with sections like "Introduction", "One line summary", "Caveats", "History", "This site", "Context", "Five minute overview", "Deciding factors", "VCS features", "VCS choices", "Feature flags", "Branch by Abstraction", "Branch for release", "Release from trunk", "Continuous Integration", "Continuous Code Review", "Continuous Delivery", "Concurrent development of con...", "Strangulation", "Observed habits", "Doing it wrong", "Alternative branching models", "Monorepos", "Expanding Contracting Monorep...", "Game Changers", and "Publications". A "Site Source on Github" link is also present. The main content area starts with a "One line summary" section, followed by a diagram titled "A shared branch" showing code being developed in a single trunk. Below this is a "Caveats" section with a list of bullet points, and a "History" section with historical context and links to publications.

Introduction

One line summary

A source-control branching model, where developers collaborate on code in a single branch called 'trunk' *, resist any pressure to create other long-lived development branches by employing documented techniques, avoid merge hell, and live happily ever after.

The diagram shows a horizontal grey bar labeled "A shared branch" at the top. Below it, two arrows point down to a blue arrow labeled "The trunk" pointing to the right. The text "The whole team develops here ... not here" is written above the trunk arrow, with "not here" pointing to a small grey arrow pointing away from the trunk.

* 'master', in Git nomenclature

Trunk-Based Development is a key enabler of [Continuous Integration](#), and by extension [Continuous Delivery](#). When individuals on a team are committing their changes to the trunk multiple times a day it becomes easy to satisfy the core requirement of Continuous Integration that all team members commit to trunk at least once every 24 hours. This ensures the codebase is always releasable on demand and helps to make Continuous Delivery a reality.

Caveats

- If you have more than a couple of developers on the project, you are going to need a hook up a [build server](#) to verify their commits
- Depending on the team size, and the rate of commits, **very short-lived** feature/task branches are used for code-review and build checking (CI) to happen before commits land in the trunk for other developers to depend on. Such branches allow developers to engage in [eager and continuous code review](#) of contributions before they land in the trunk.
- Depending on the intended release cadence, there may be [release branches](#) that are cut from trunk on a just-in-time basis, and are 'hardened' before a release (without that being a team activity). Alternatively there may also be no release branches if the team is [releasing from Trunk](#), and choosing a roll forward strategy for bug fixes.
- Teams should become adept with the related [branch by abstraction](#) technique for longer to achieve changes, and use [Feature Flags](#) in day to day development to allow for hedging on the order of releases (and other good things - see [concurrent development of consecutive releases](#))
- Development teams can casually flex up or down in size (in the trunk). Proof? [Google do Trunk-Based Development](#) and have 25000 [developers and QA automators](#) in that trunk.
- People who practice the [GitHub-flow branching model](#) will feel that this is quite similar, but there is one small difference.
- People who practice the [Gitflow branching model](#) will find this **very different**, as will many developers used to the popular ClearCase, Subversion, Perforce, StarTeam, VCS [branching models of the past](#).
- [Many publications](#), including the best-selling 'Continuous Delivery' book promote Trunk Based Development - this is not even controversial any more.

History

Trunk-Based Development is not a new branching model. The word 'trunk' is referent to the concept of a growing tree, where the fattest and longest span is the trunk, not the branches that radiate from it and are of more limited length.

It has been a lesser known branching model of choice since the mid-nineties, and considered tactically since the eighties. The largest of development organizations, like Google (as mentioned) and Facebook practice it at scale.

Over 30 years different [advances to source-control technologies and related tools/techniques](#) have made Trunk-Based Development more (and occasionally less) necessary.

This site

This site attempts to collect all the related facts, rationale and techniques for Trunk-Based Development together in one place, complete with twenty four diagrams to help explain things. All without using TBD as an acronym even once twice.

© 2017 TrunkBasedDevelopment.com's contributors. – Site built with [Hugo](#) using the [Material](#) theme, and auto-built as well as hosted on [Netlify](#).

Next → Context →

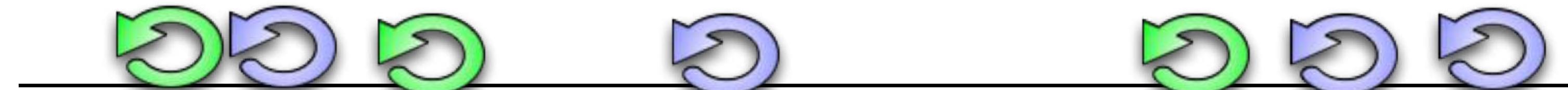


Paul Hammant



trunk-based development

<https://trunkbaseddevelopment.com>



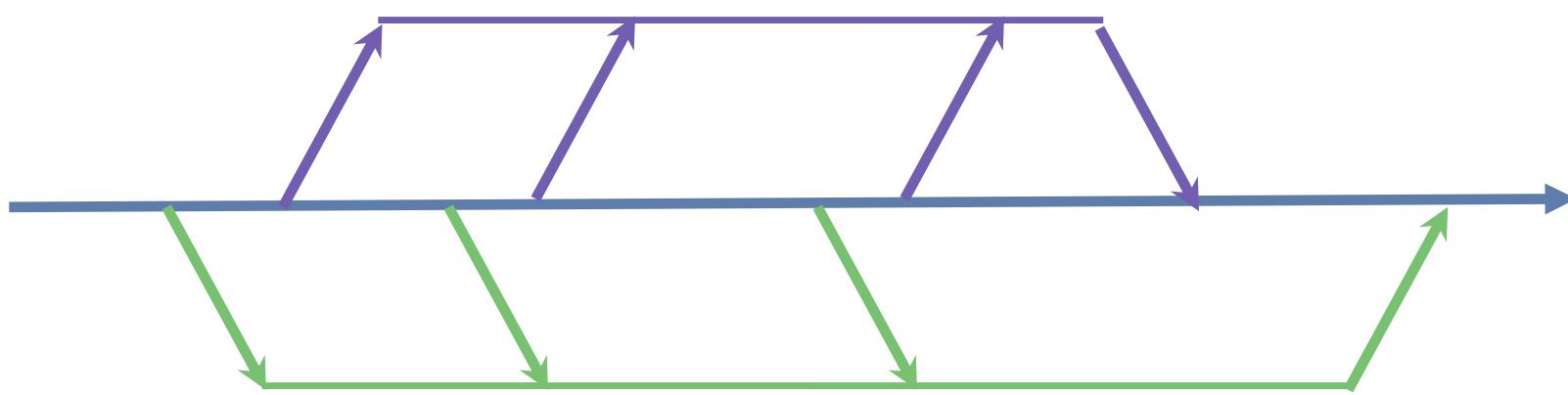
trunk-based development



does it!!

<https://cacm.acm.org/magazines/2016/7/204032-why-google-stores-billions-of-lines-of-code-in-a-single-repository/fulltext>

Feature Branching



1 !

Big Scary Merge

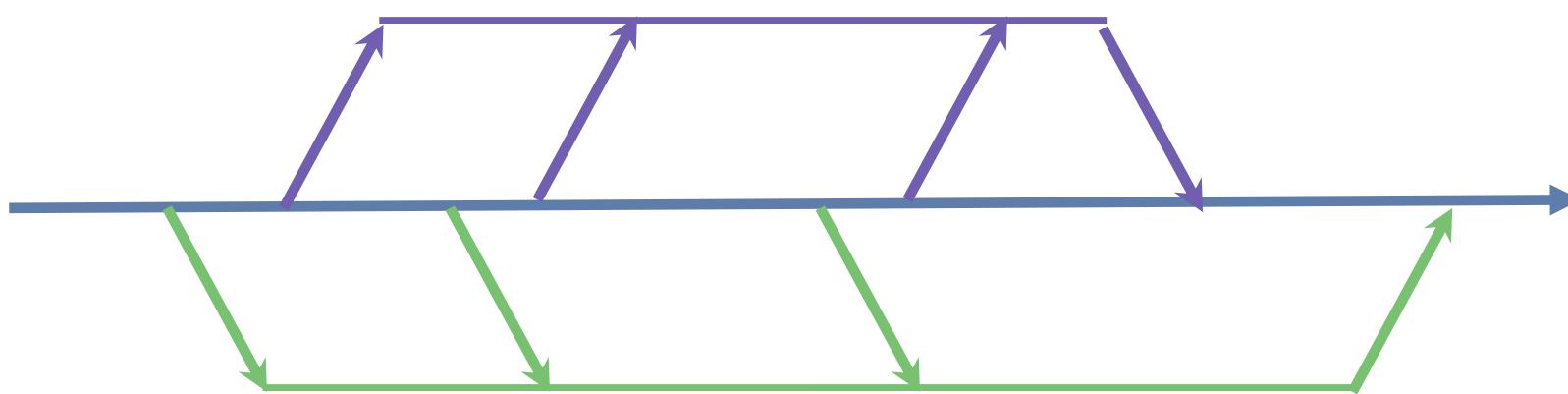
Cherry Picking



Untrusted Contributors

Continuous Integration

Feature Branching



1 !
2 !

Big Scary Merge

Discouraging refactoring

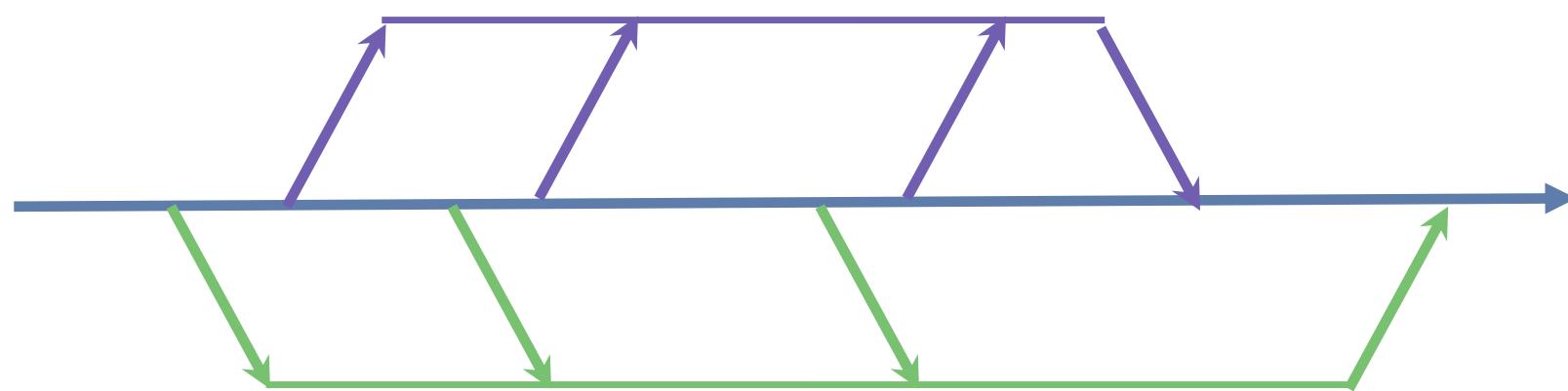
Cherry Picking



Untrusted Contributors

Continuous Integration

Feature Branching



1 !

Big Scary Merge

2 !

Discouraging refactoring

3 !

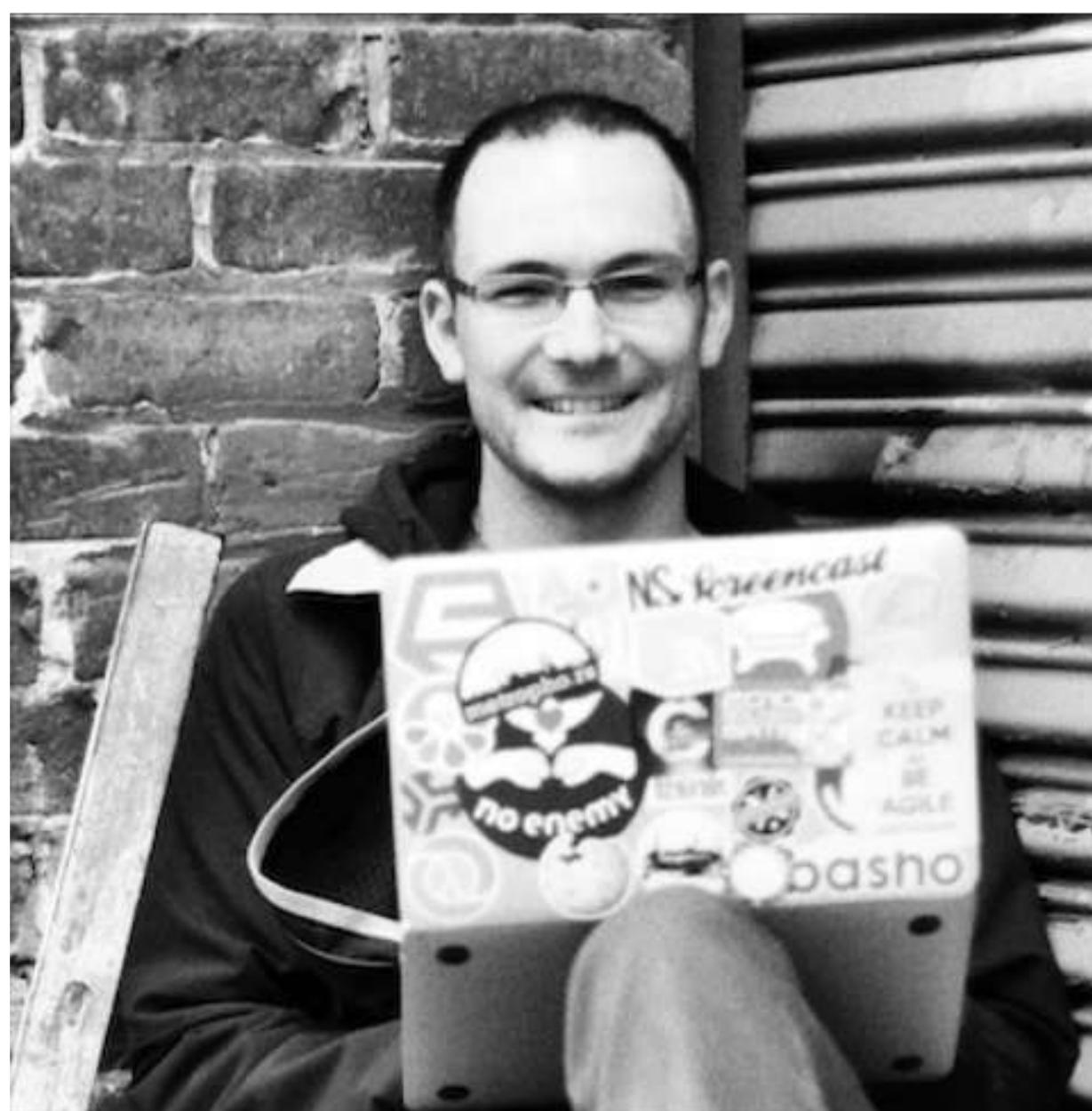
Hard to combine features

Cherry Picking



Untrusted Contributors

Continuous Integration



martinfowler.com

MARTIN FOWLER

Feature Toggles

Feature toggles are a powerful technique, allowing teams to modify system behavior without changing code. They fall into various usage categories, and it's important to take that categorization into account when implementing and managing toggles. Toggles introduce complexity. We can keep that complexity in check by using smart toggle implementation practices and appropriate tools to manage our toggle configuration, but we should also aim to constrain the number of toggles in our system.

08 February 2016

Pete Hodgson

Pete Hodgson is a consultant at ThoughtWorks, where he's spent the last few years helping teams become awesome at sustainable delivery of high-quality software. He's particularly passionate about web-based mobile, ruby, agile, functional approaches to software, and beer.

Find [similar articles](#) to this by looking at these tags: [delivery](#) · [application architecture](#) · [continuous integration](#)

Contents [expand](#)

- A Toggling Tale
- Categories of toggles
- Implementation Techniques
- Toggle Configuration
- Working with feature-toggled systems

"Feature Toggling" is a set of patterns which can help a team to deliver new functionality to users rapidly but safely. In this article on Feature Toggling we'll start off with a short story showing some typical scenarios where Feature Toggles are helpful. Then we'll dig into the details, covering specific patterns and practices which will help a team succeed with Feature Toggles.

A Toggling Tale

Picture the scene. You're on one of several teams working on a sophisticated town planning simulation game. Your team is responsible for the core simulation engine. You have been tasked with increasing the efficiency of the Spline Reticulation algorithm. You know this will require a fairly large overhaul of the implementation which will take several weeks. Meanwhile other members of your team will need to

before

```
function reticulateSplines(){
    // current implementation lives here
}
```

these examples all use JavaScript ES2015

after

```
function reticulateSplines(){
    var useNewAlgorithm = false;
    // useNewAlgorithm = true; // UNCOMMENT IF YOU ARE WORKING ON THE NEW SR ALGORITHM

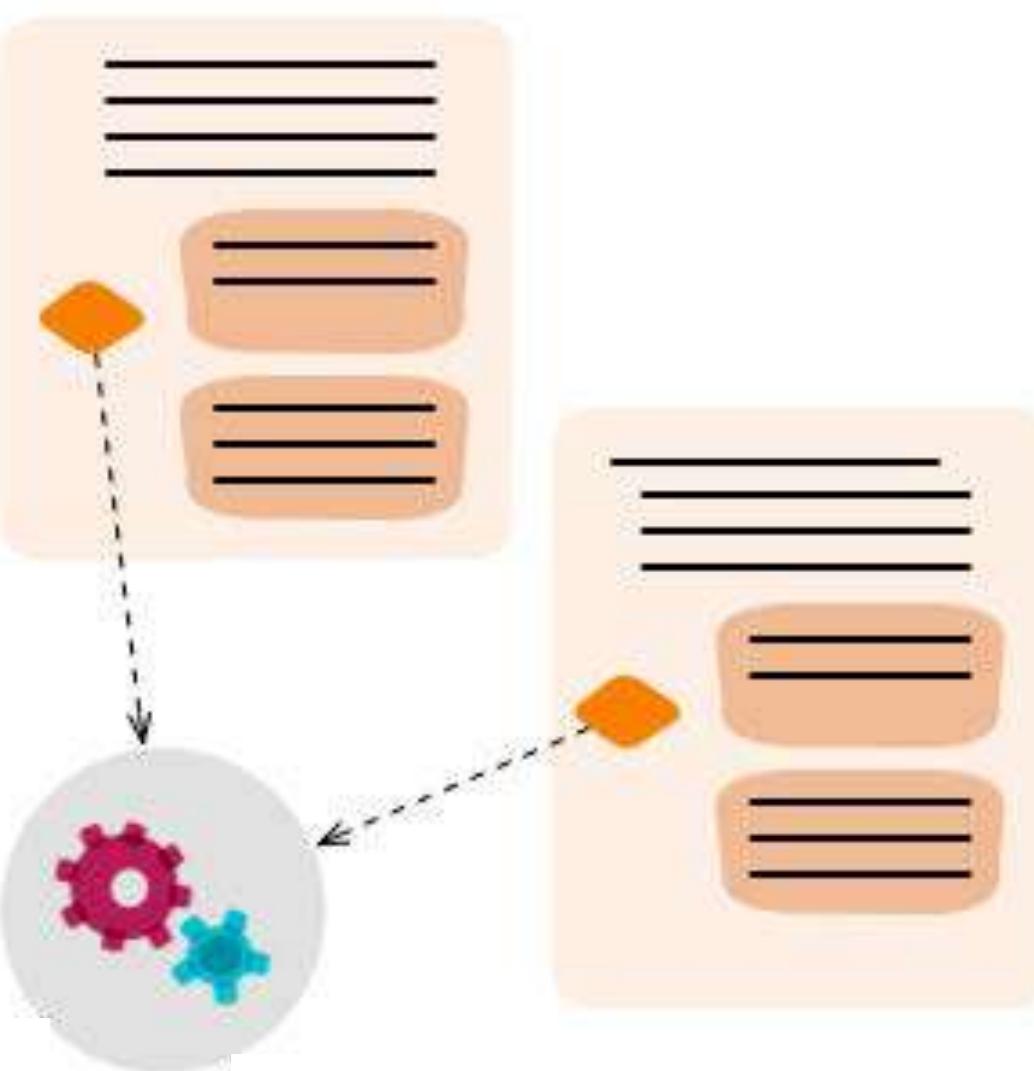
    if( useNewAlgorithm ){
        return enhancedSplineReticulation();
    }else{
        return oldFashionedSplineReticulation();
    }
}

function oldFashionedSplineReticulation(){
    // current implementation lives here
}

function enhancedSplineReticulation(){
    // TODO: implement better SR algorithm
}
```

```
function reticulateSplines(){
  if( featureEnabled("use-new-SR-algorithm") ){
    return enhancedSplineReticulation();
  }else{
    return oldFashionedSplineReticulation();
  }
}
```

make toggle dynamic



```
function createToggleRouter(featureConfig){
  return {
    setFeature(featureName, isEnabled){
      featureConfig[featureName] = isEnabled;
    },
    featureEnabled(featureName){
      return featureConfig[featureName];
    }
  };
}
```

note that we're using ES2015's method shorthand

```
describe( 'spline reticulation', function(){
  let toggleRouter;
  let simulationEngine;

  beforeEach(function(){
    toggleRouter = createToggleRouter();
    simulationEngine = createSimulationEngine({toggleRouter:toggleRouter});
  });

  it('works correctly with old algorithm', function(){
    // Given
    toggleRouter.setFeature("use-new-SR-algorithm",false);

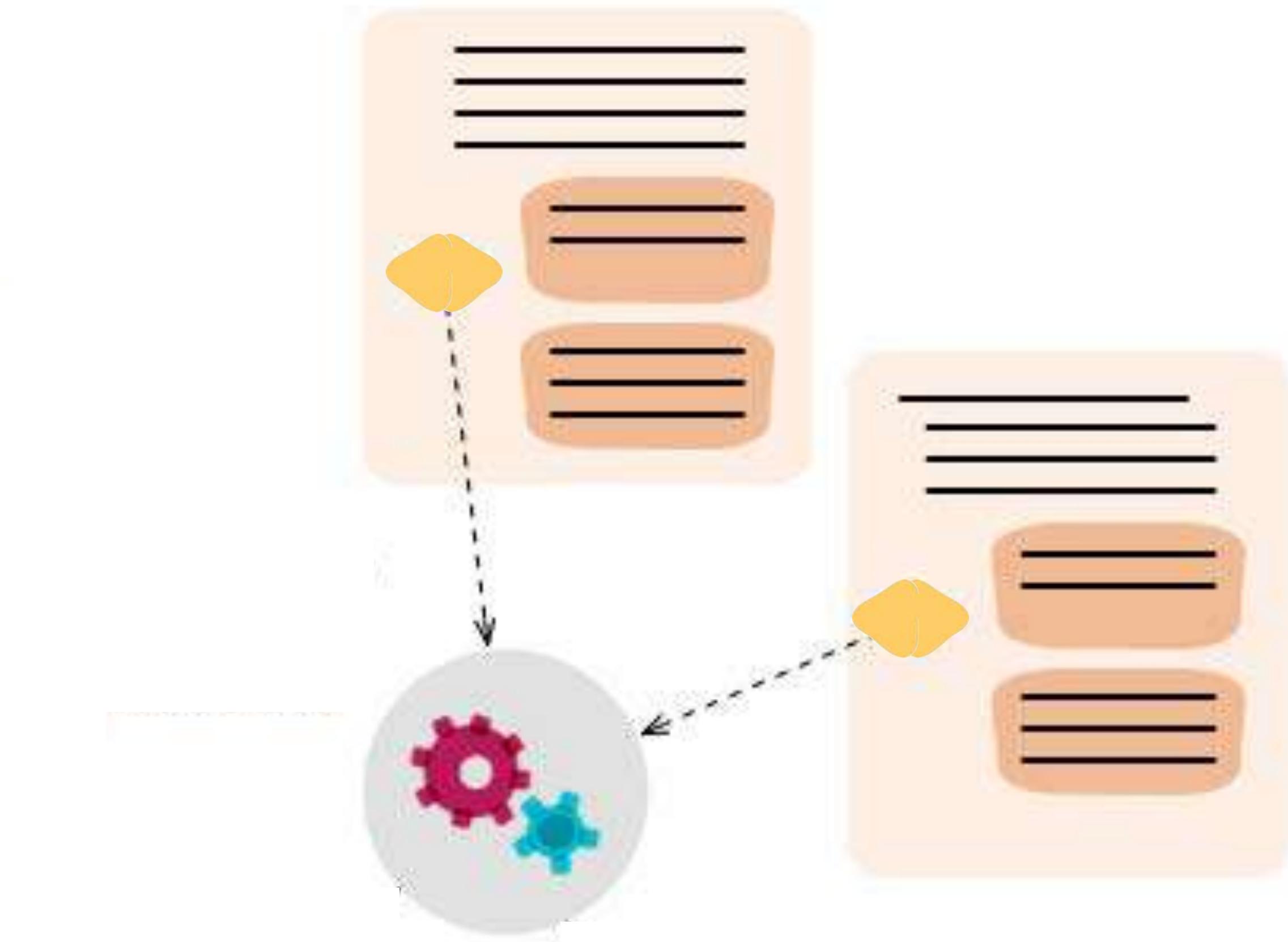
    // When
    const result = simulationEngine.doSomethingWhichInvolvesSplineReticulation();

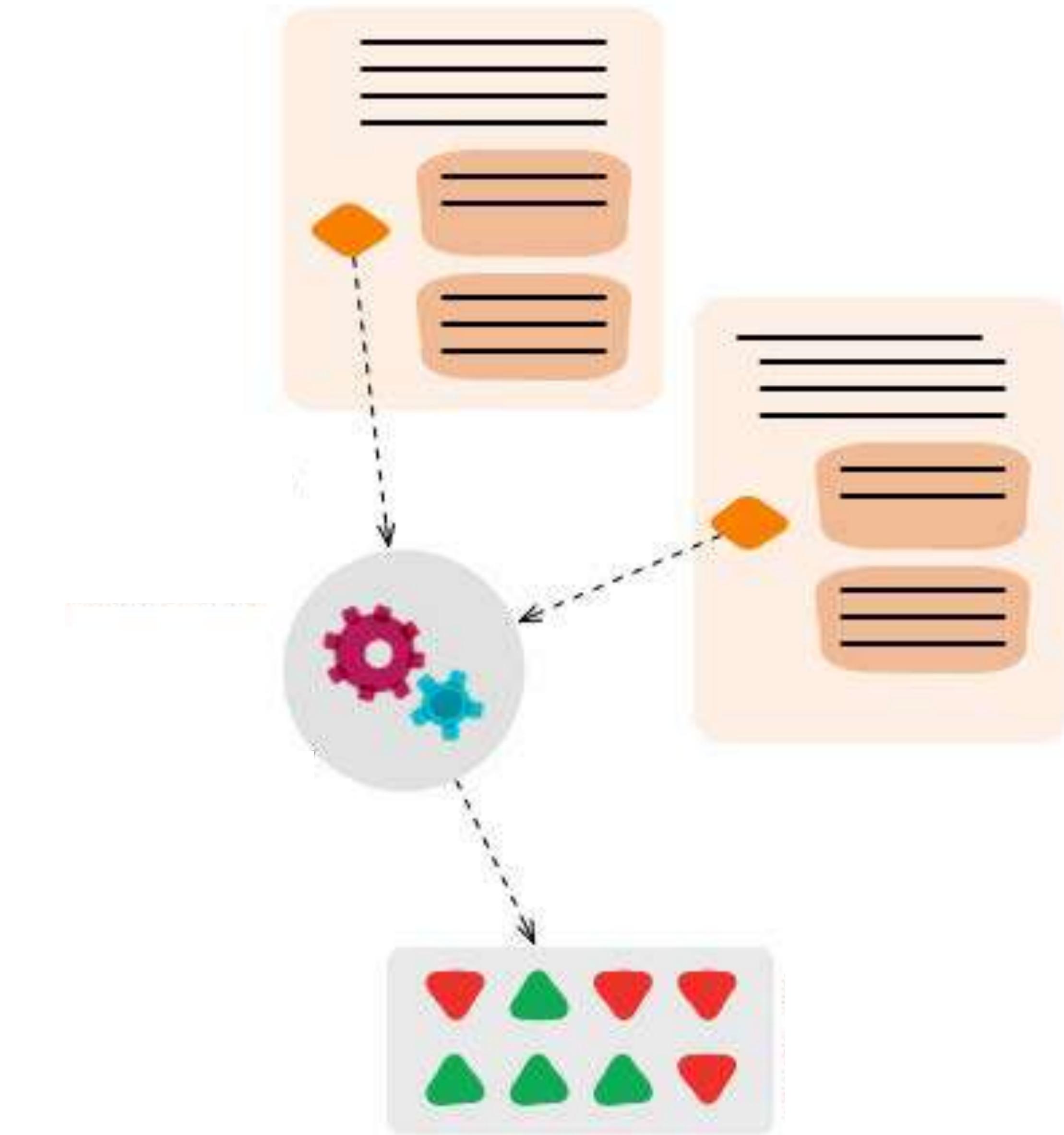
    // Then
    verifySplineReticulation(result);
  });

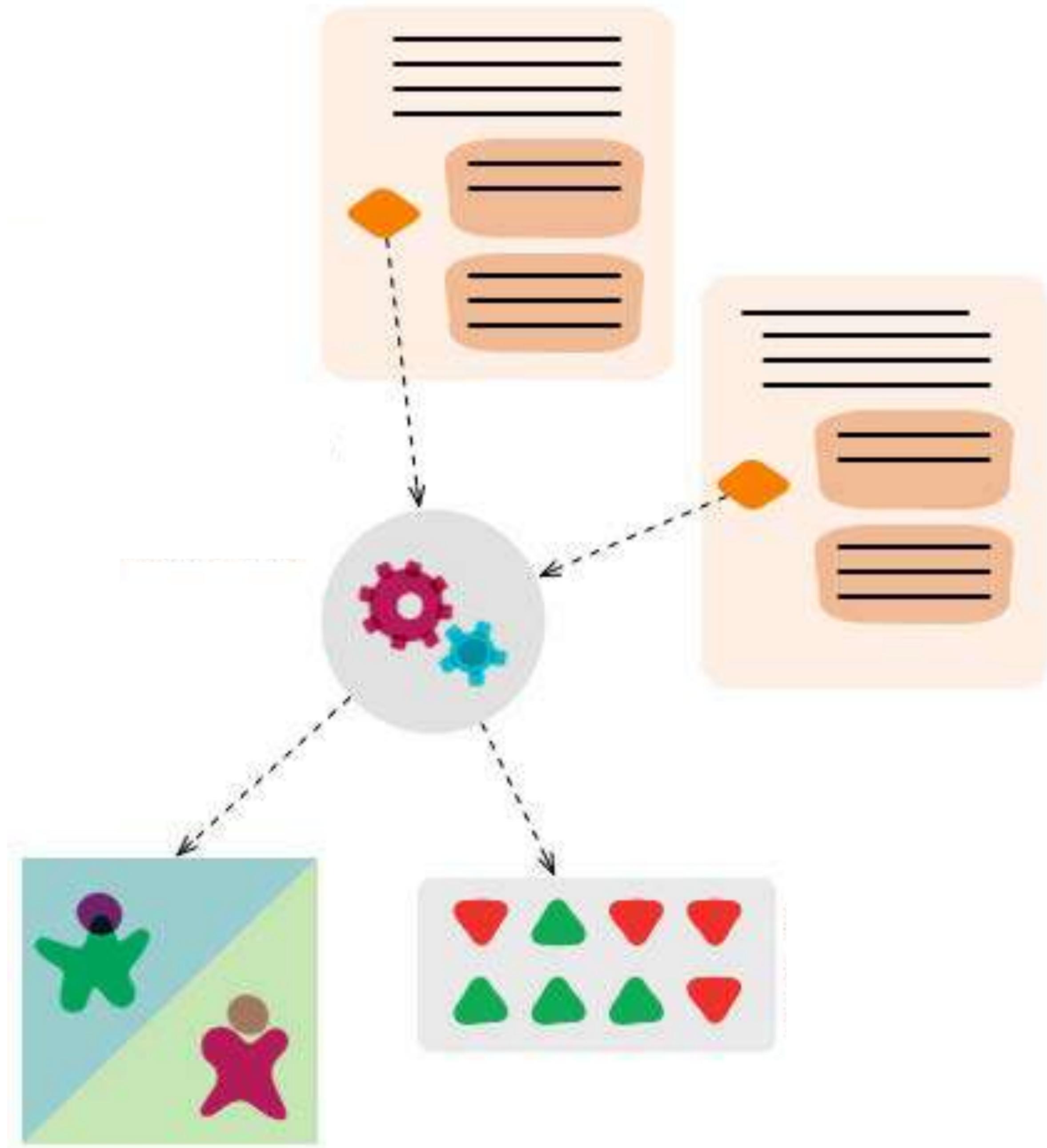
  it('works correctly with new algorithm', function(){
    // Given
    toggleRouter.setFeature("use-new-SR-algorithm",true);

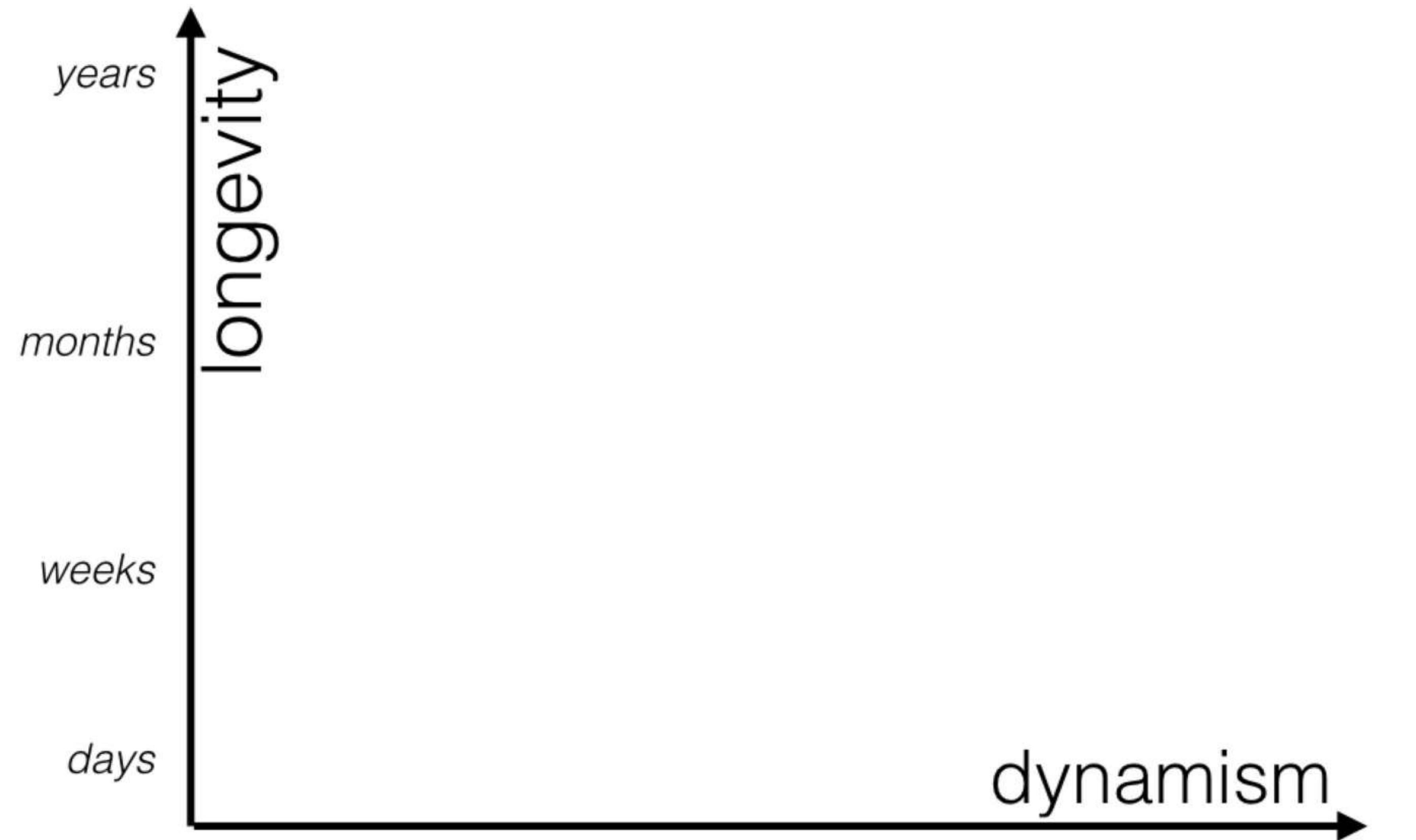
    // When
    const result = simulationEngine.doSomethingWhichInvolvesSplineReticulation();

    // Then
    verifySplineReticulation(result);
  });
});
```





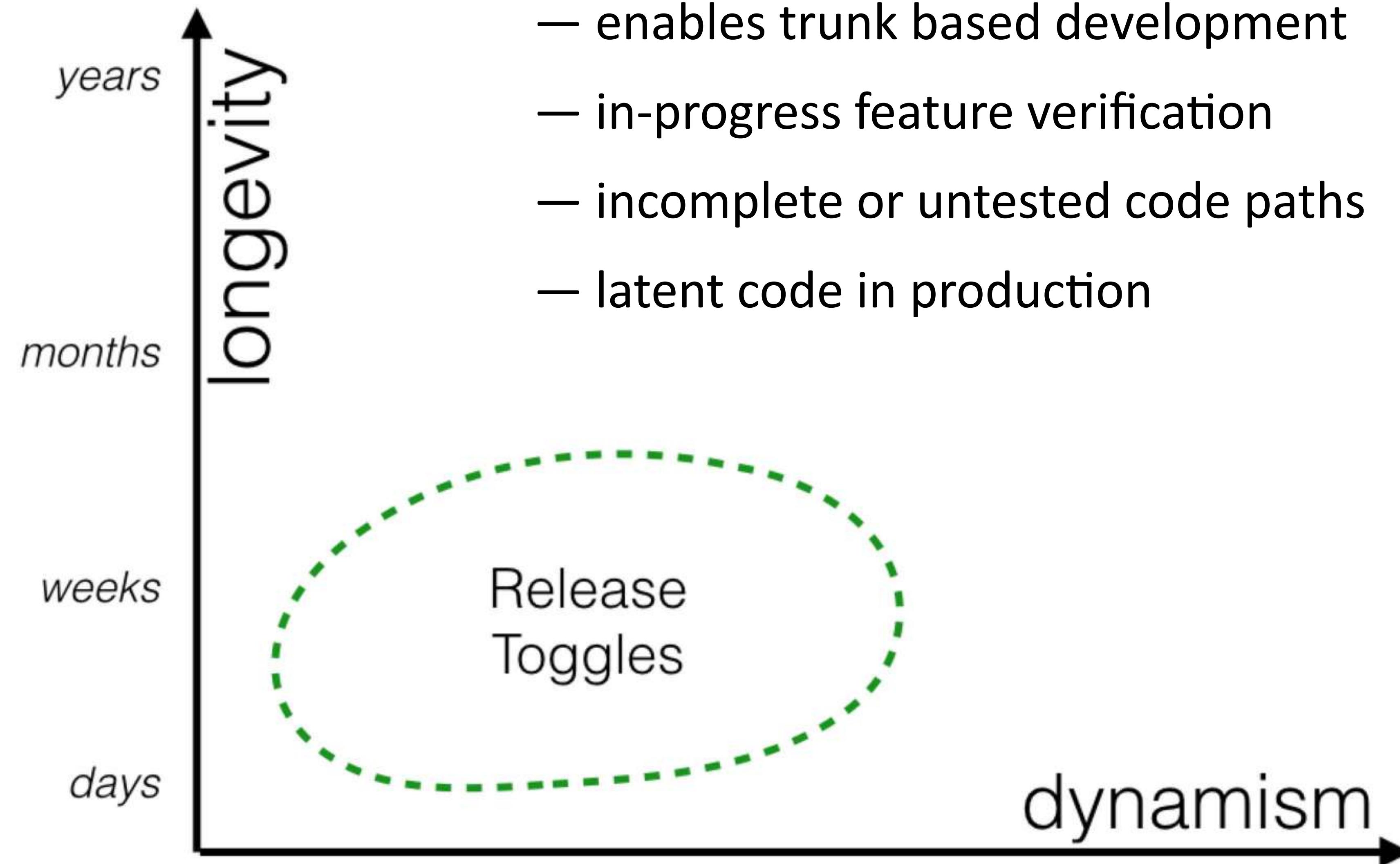




*changes with
a deployment*

*changes with runtime
re-configuration*

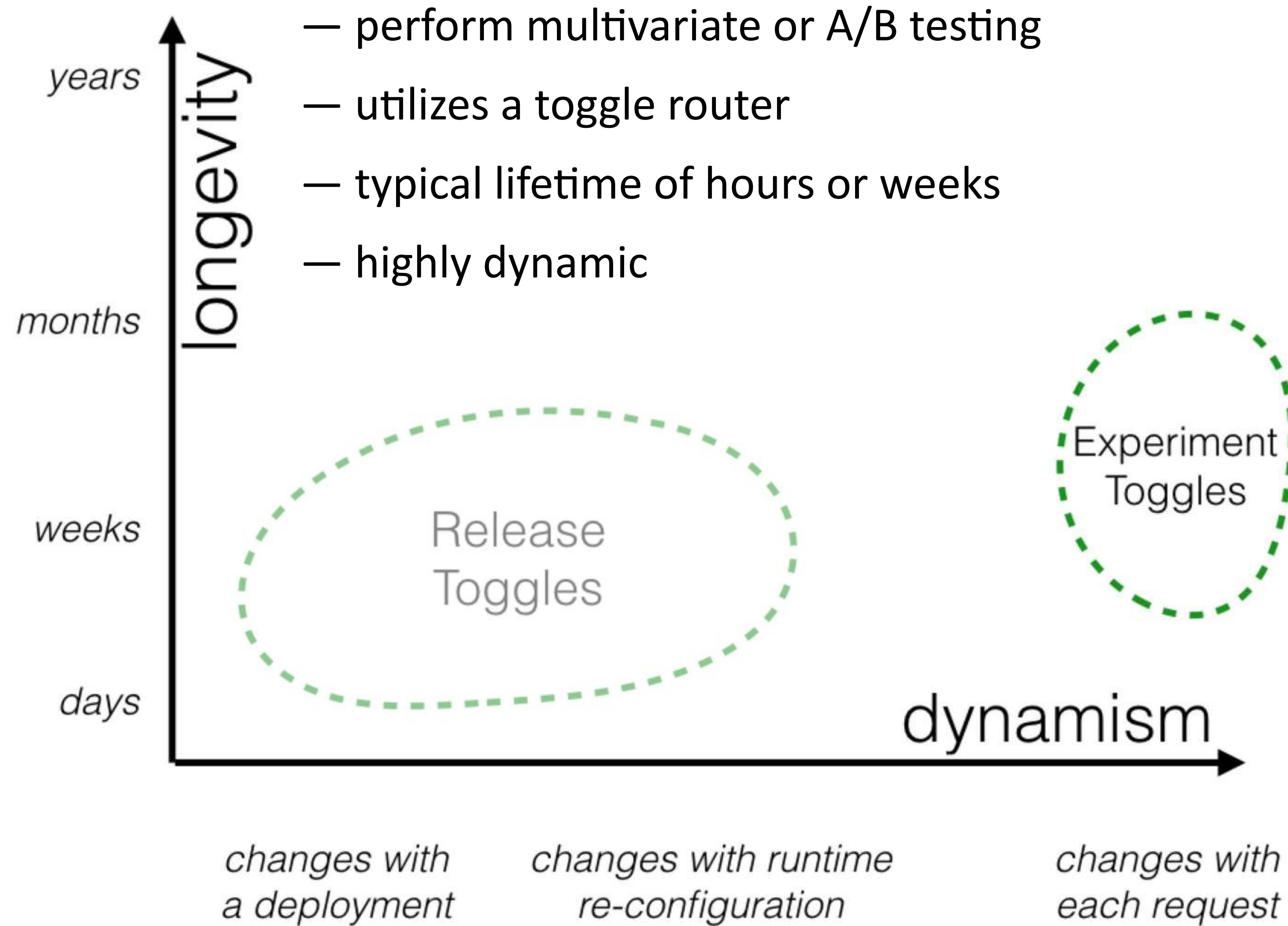
*changes with
each request*

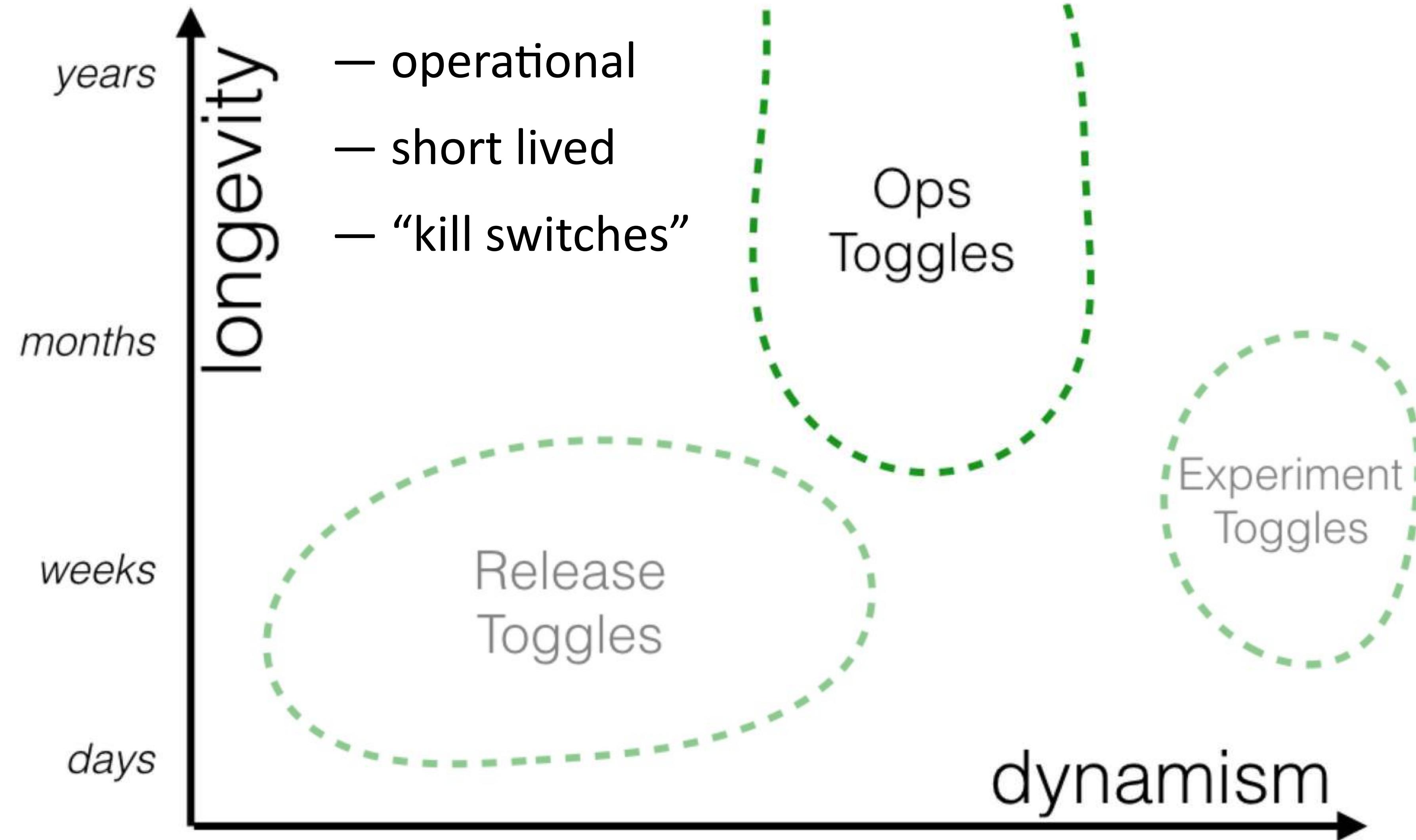


*changes with
a deployment*

*changes with runtime
re-configuration*

*changes with
each request*

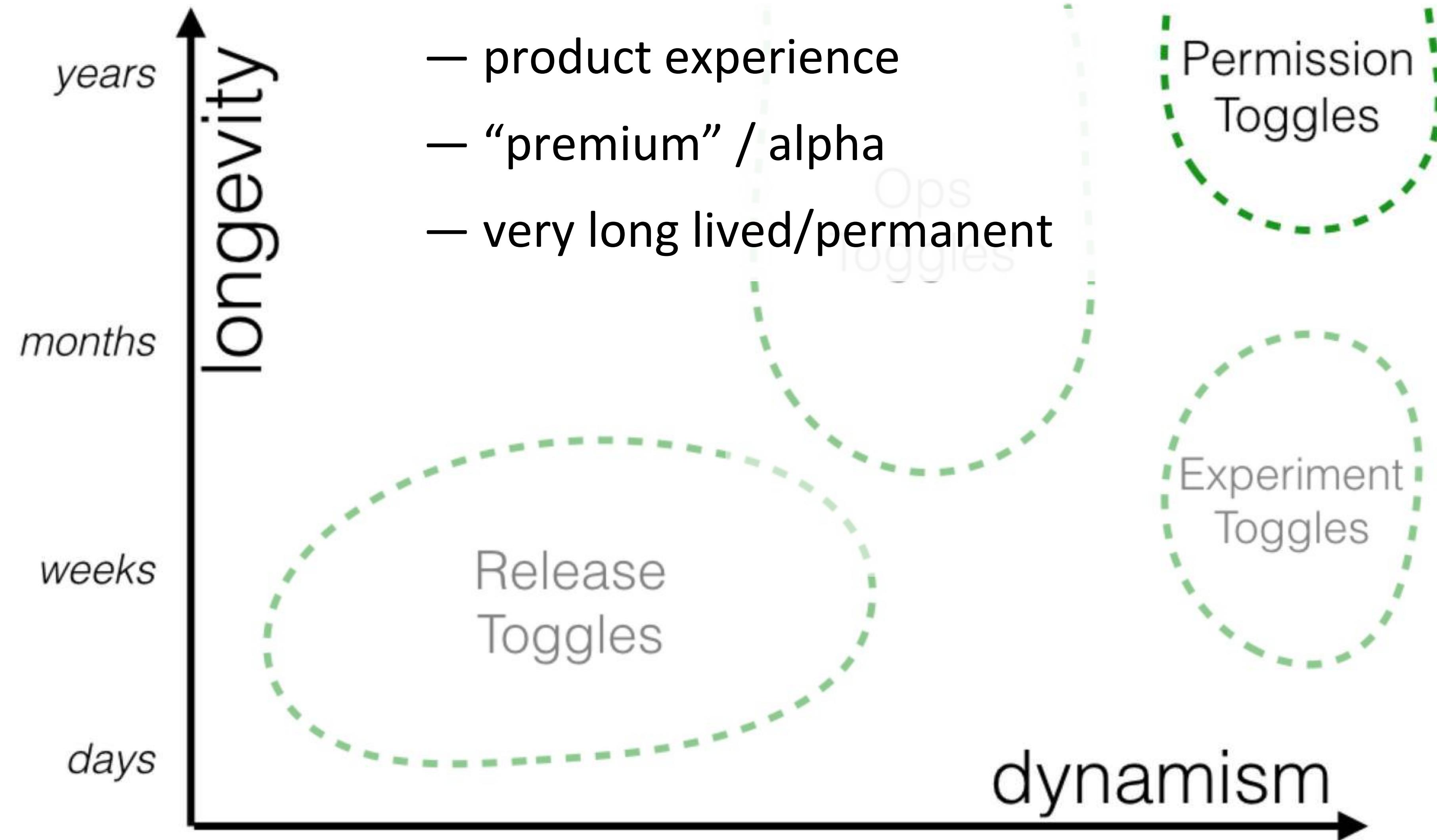




*changes with
a deployment*

*changes with runtime
re-configuration*

*changes with
each request*

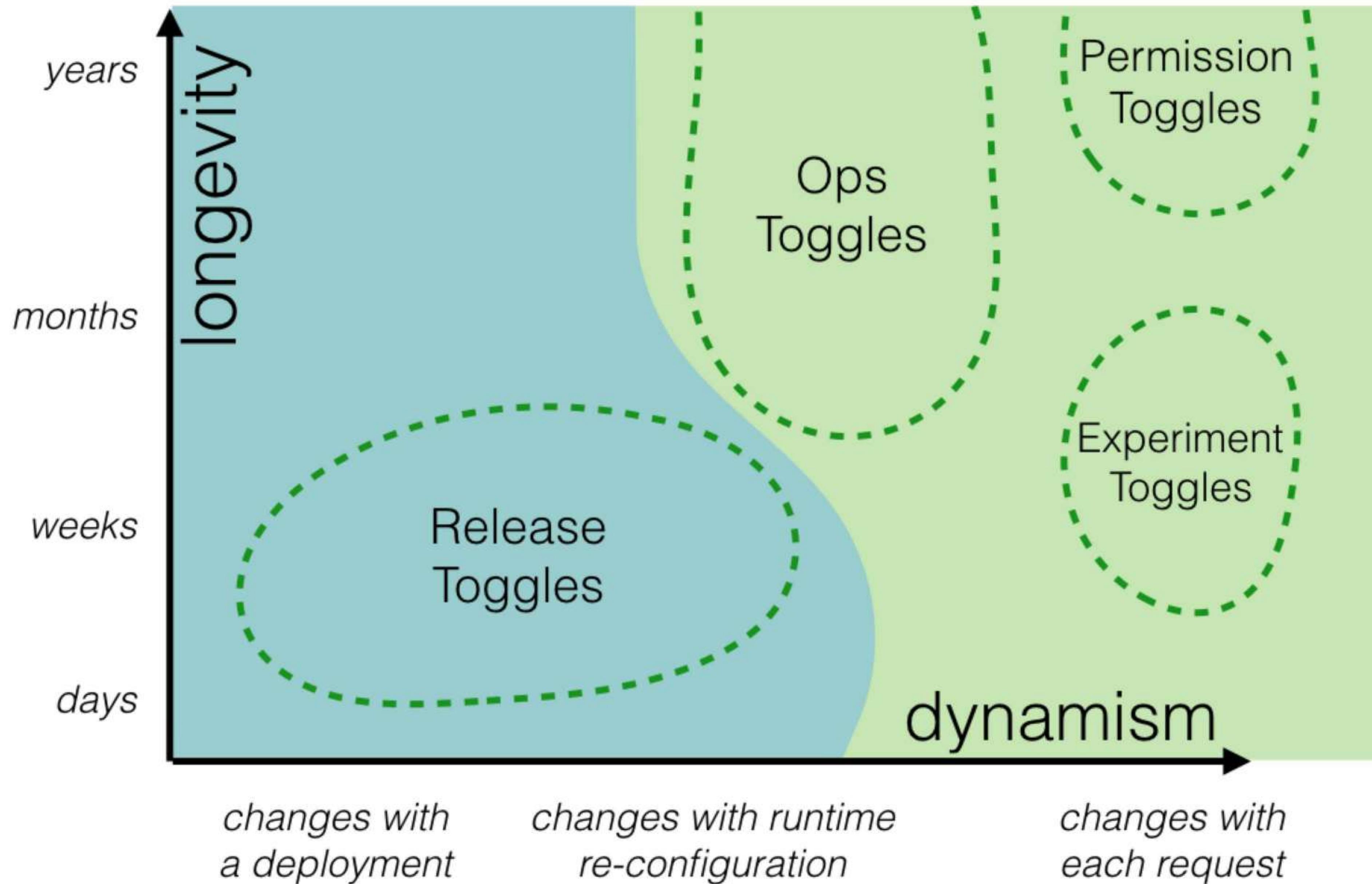


*changes with
a deployment*

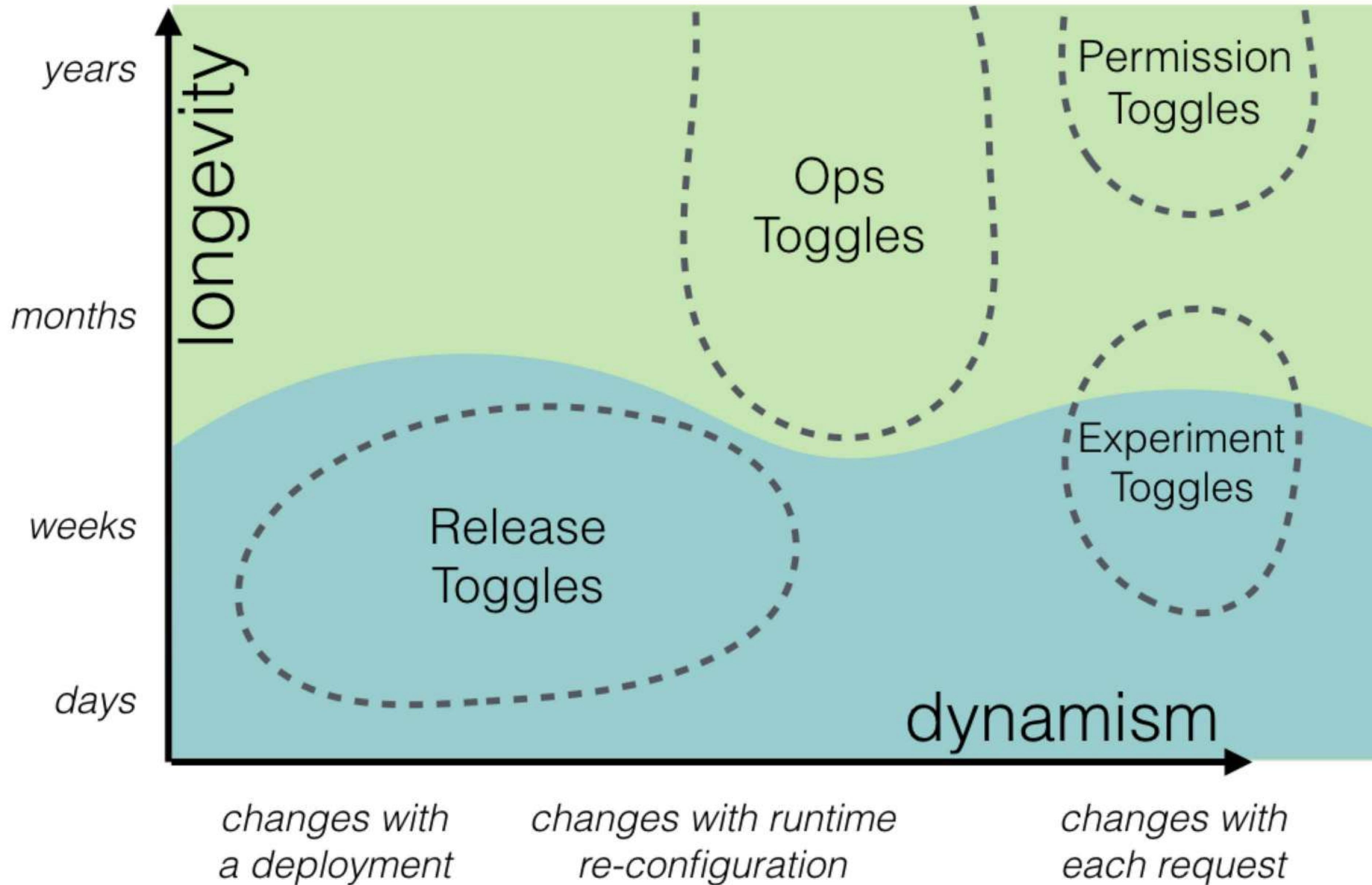
*changes with runtime
re-configuration*

*changes with
each request*

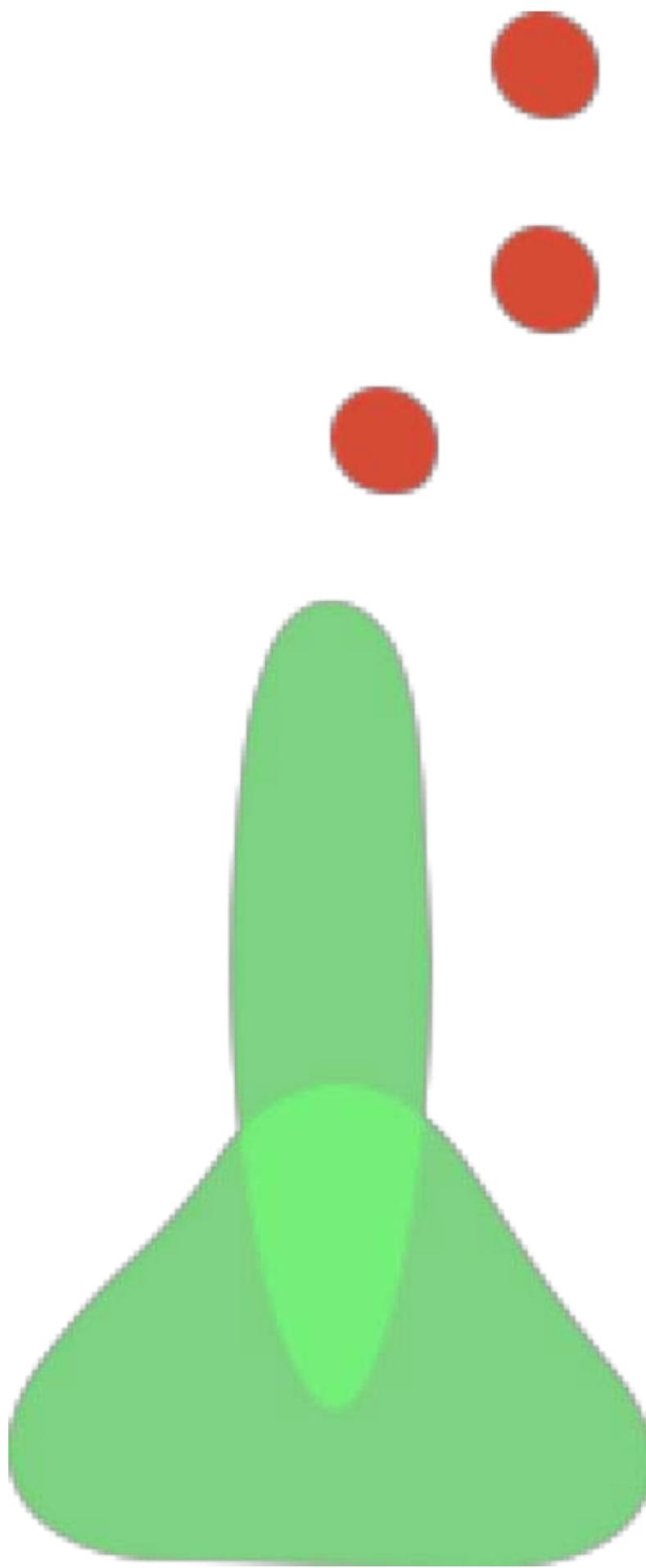
static versus dynamic toggles

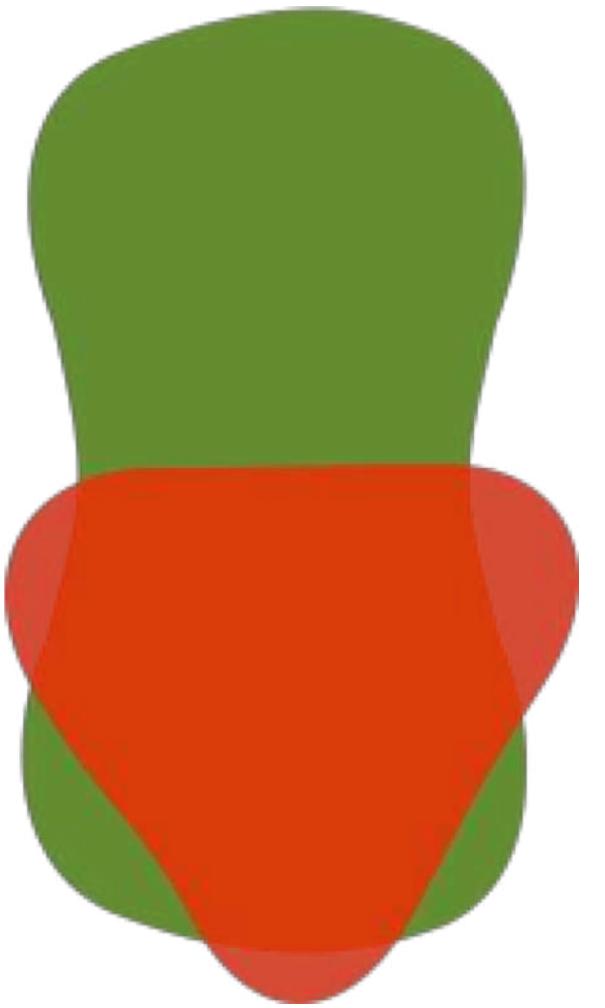


long-lived versus transient toggles



experimental toggles





removed as soon as feature decision is resolved

Feature toggles are purposeful technical debt added to support engineering practices like Continuous Delivery.



DZone / DevOps Zone

REFCARDZ GUIDES ZONES | AGILE BIG DATA CLOUD DATABASE DEVOPS INTEGRATION IOT JAVA MOBILE MORE

Feature Toggles are one of the Worst kinds of Technical Debt

Technical debt is pretty bad, and feature toggles are some of the most horrible examples of technical debt.

by Jim Bird · Aug. 11, 14 · DevOps Zone

Like (0) Comment (0) Save Tweet 9,288 Views

The DevOps Zone is brought to you in partnership with [New Relic](#). Learn more about the common barriers to DevOps adoption so that you can come up with ways to win over the skeptics and [kickstart DevOps](#).

Feature flags or config flags aka feature toggles aka flippers are an important part of Devops practices like dark launching (releasing features immediately and incrementally), A/B testing, and branching in code or branching by abstraction (so that development teams can all work together directly on the code mainline instead of creating separate feature branches).

Feature toggles can be simple Boolean switches or complex decision trees with multiple different paths. [Martin Fowler](#) differentiates between release toggles (which are used by development and ops to temporarily hide incomplete or risky features from all or part of the user base) and business toggles to control what features are available to different users (which may have a longer – even permanent – life). He suggests that these different kinds of flags should be managed separately, in different configuration files for example. But the basic idea is the same, to build conditional branches into mainline code in order to make logic available only to some users or to skip or hide logic at run-time, including code that isn't complete (the case for branching by abstraction).

Let's be friends: [RSS](#) [Twitter](#) [Facebook](#) [Google+](#) [LinkedIn](#)

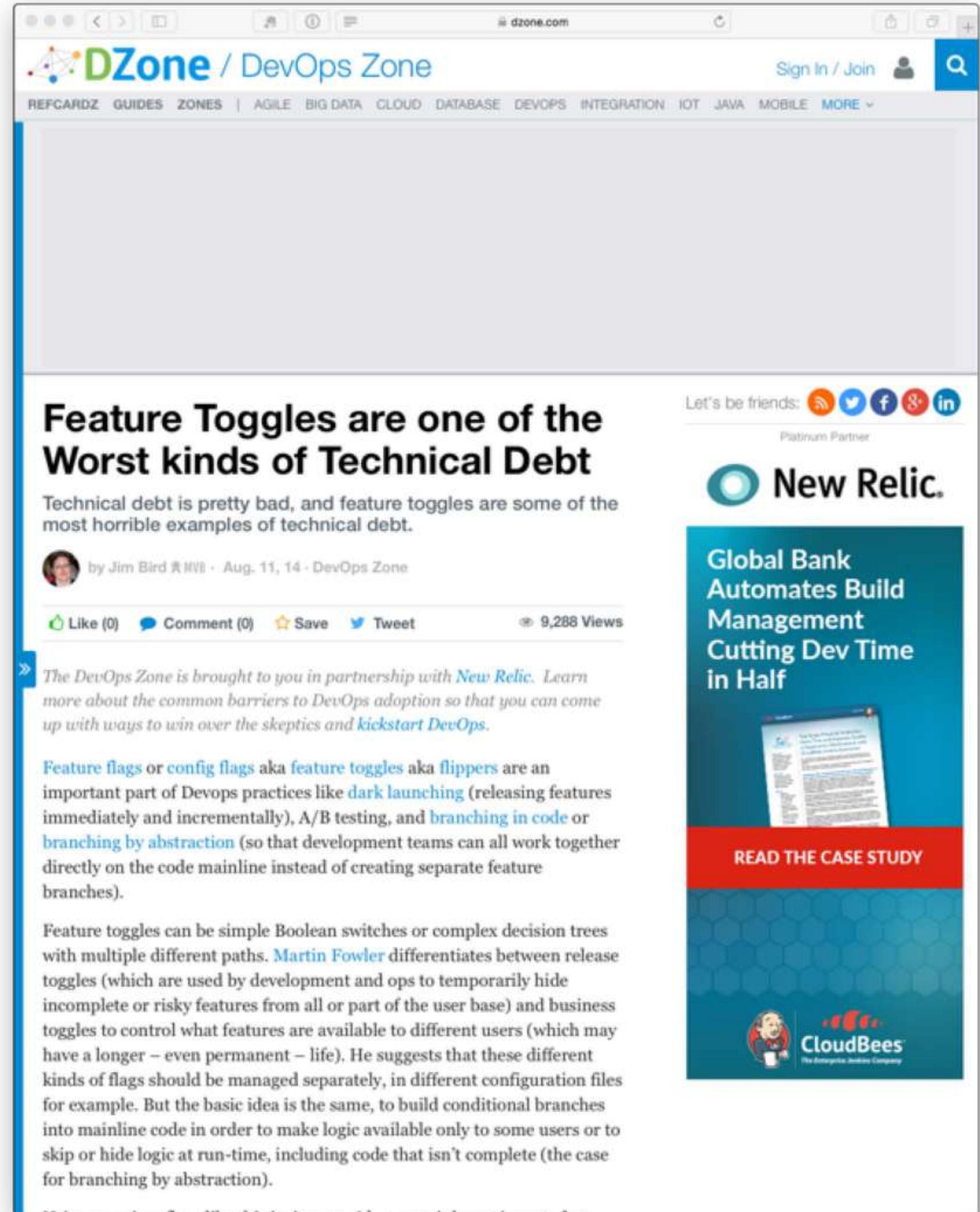
Platinum Partner

New Relic.

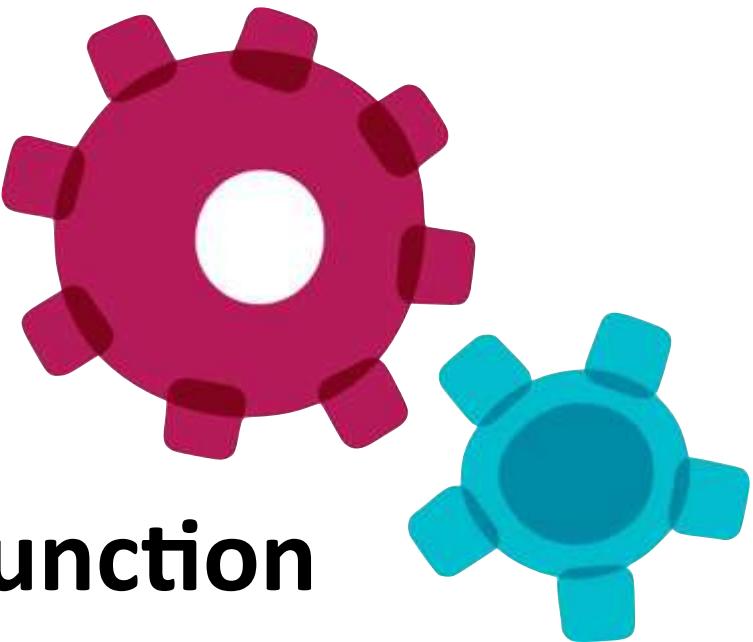
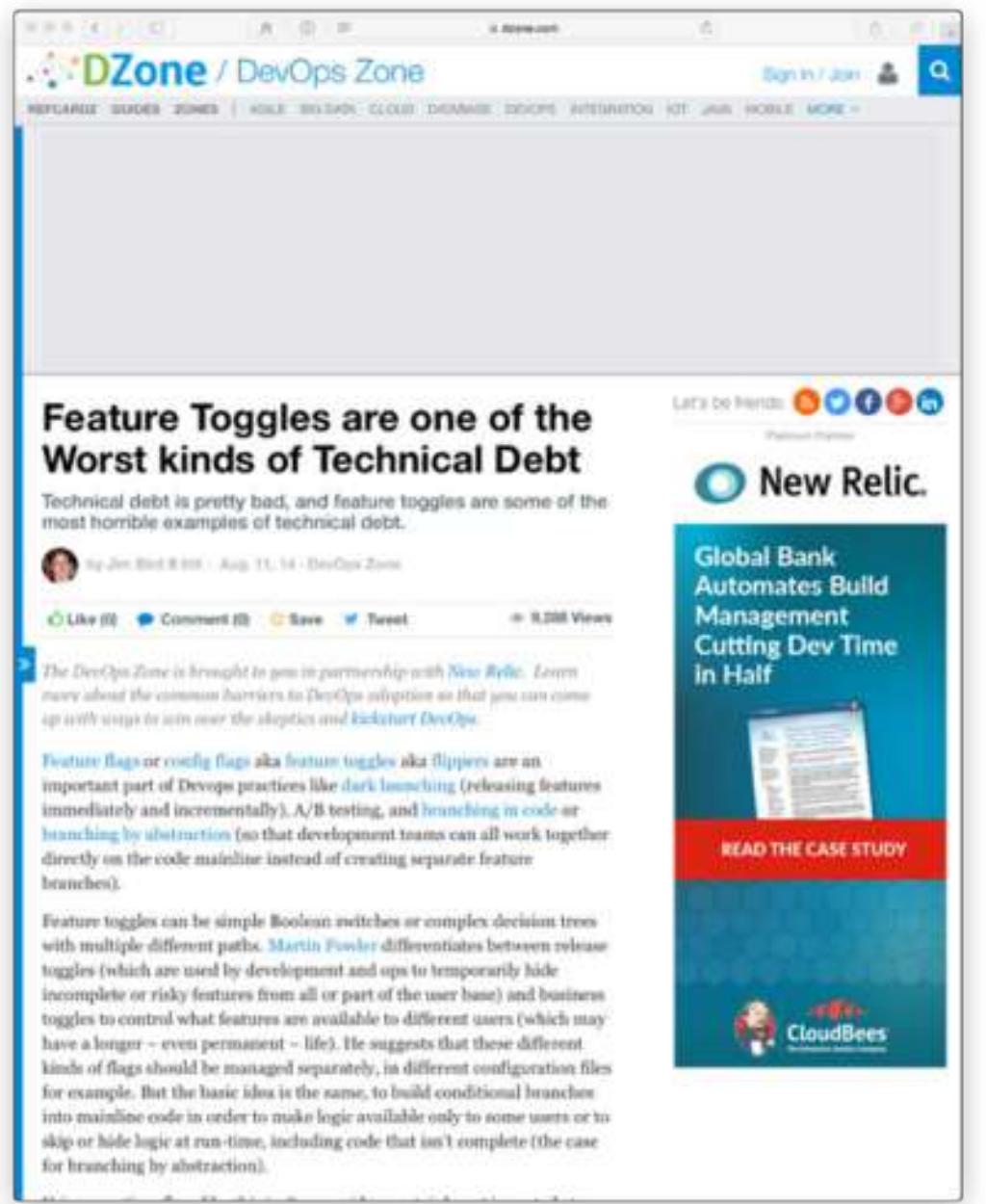
Global Bank Automates Build Management Cutting Dev Time in Half

READ THE CASE STUDY

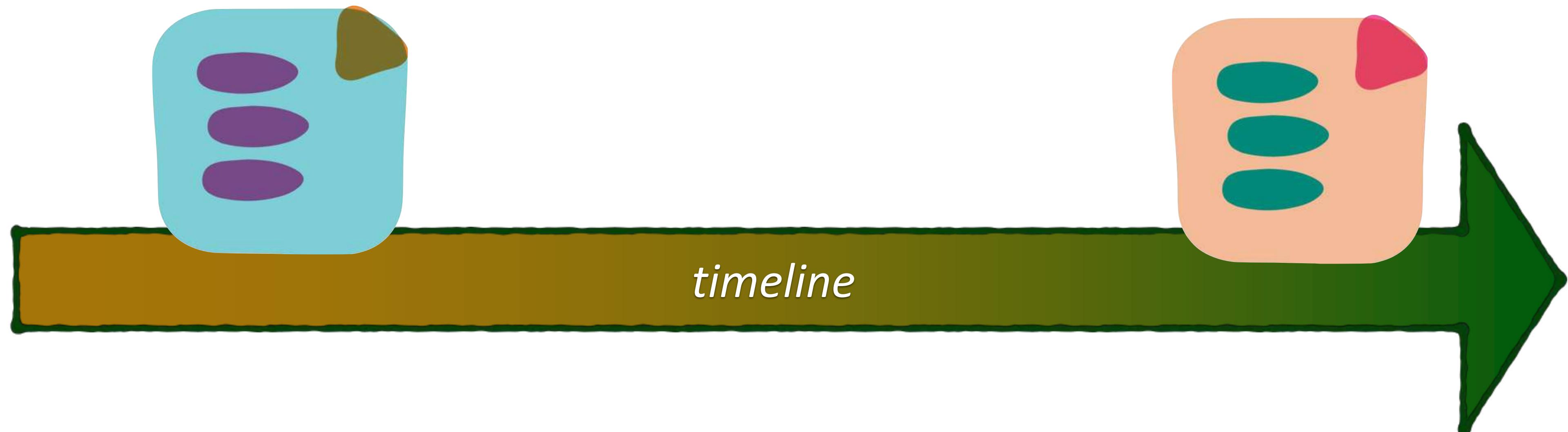
CloudBees The Enterprise Jenkins Company

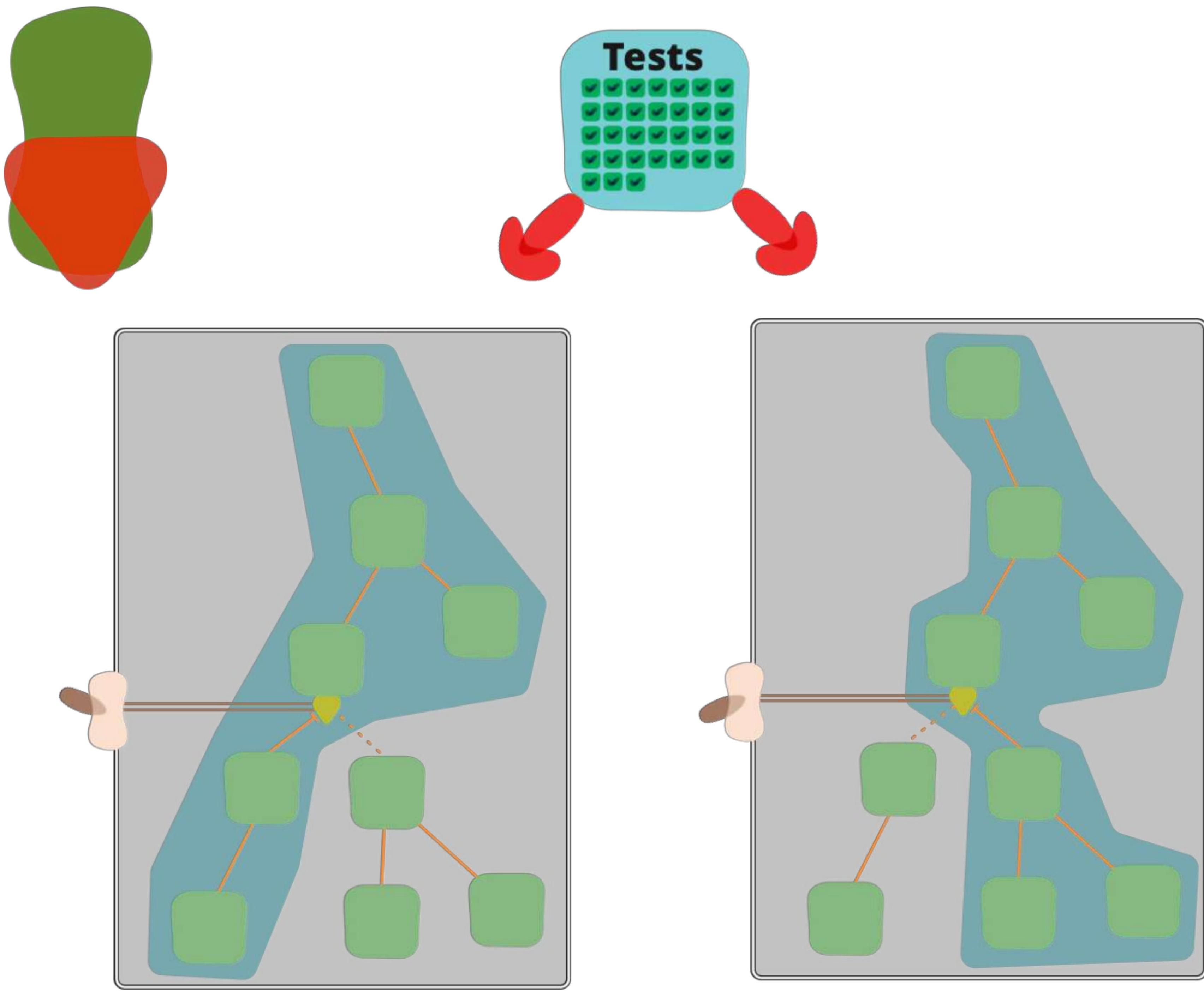


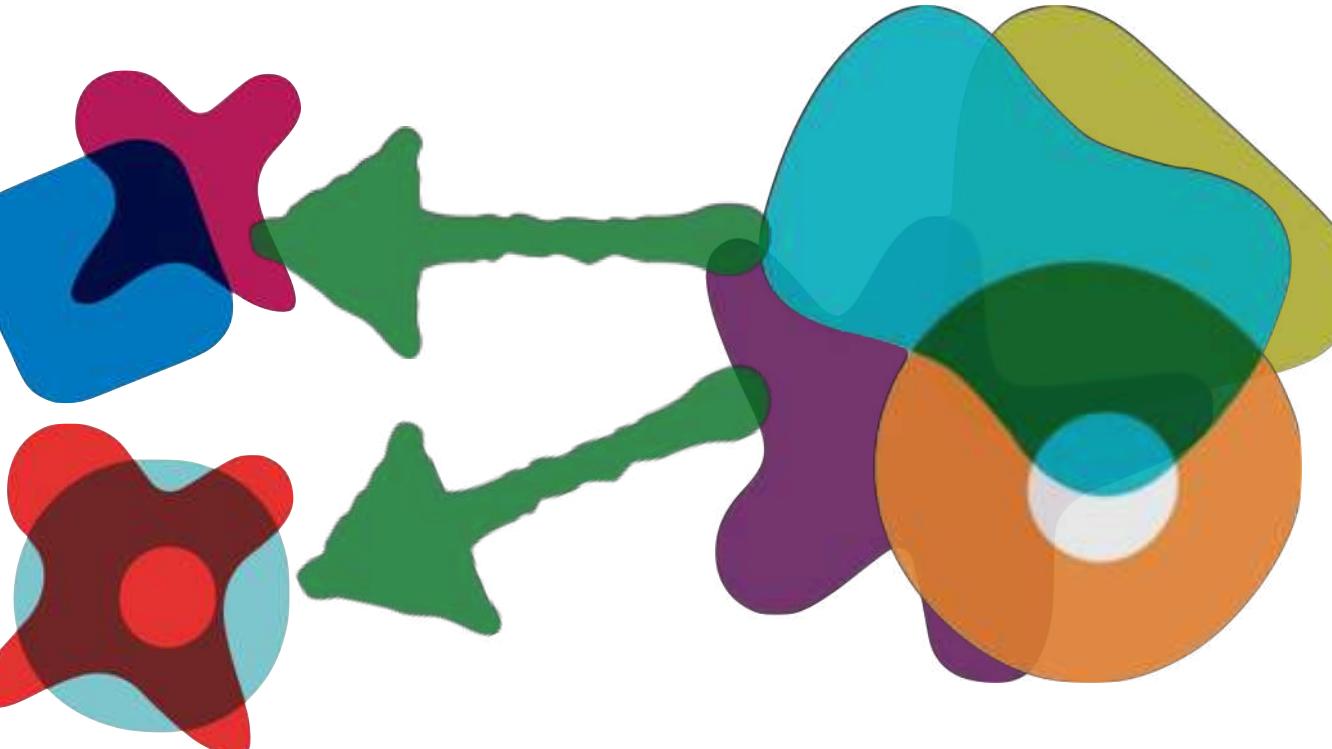
<https://dzone.com/articles/feature-toggles-are-one-worst>



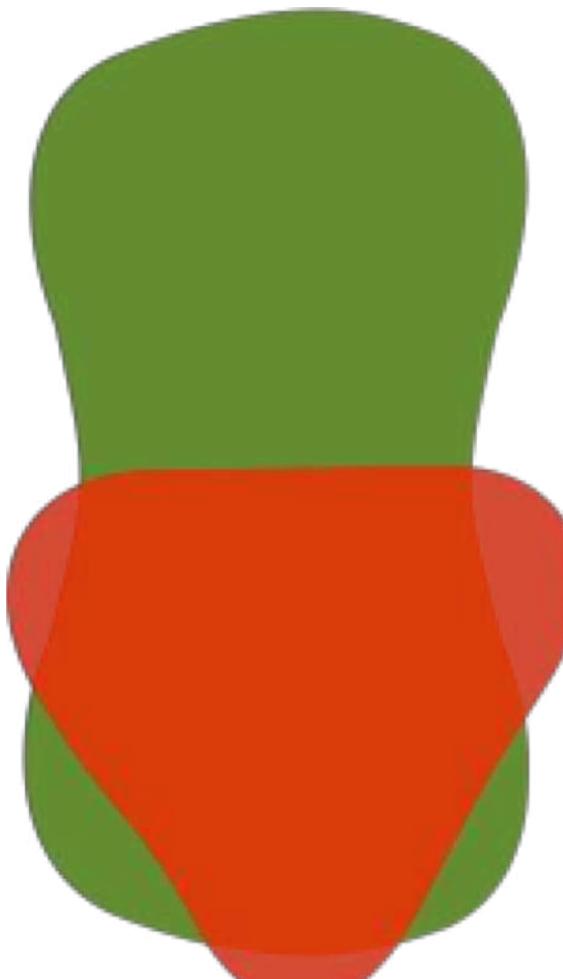
fitness function

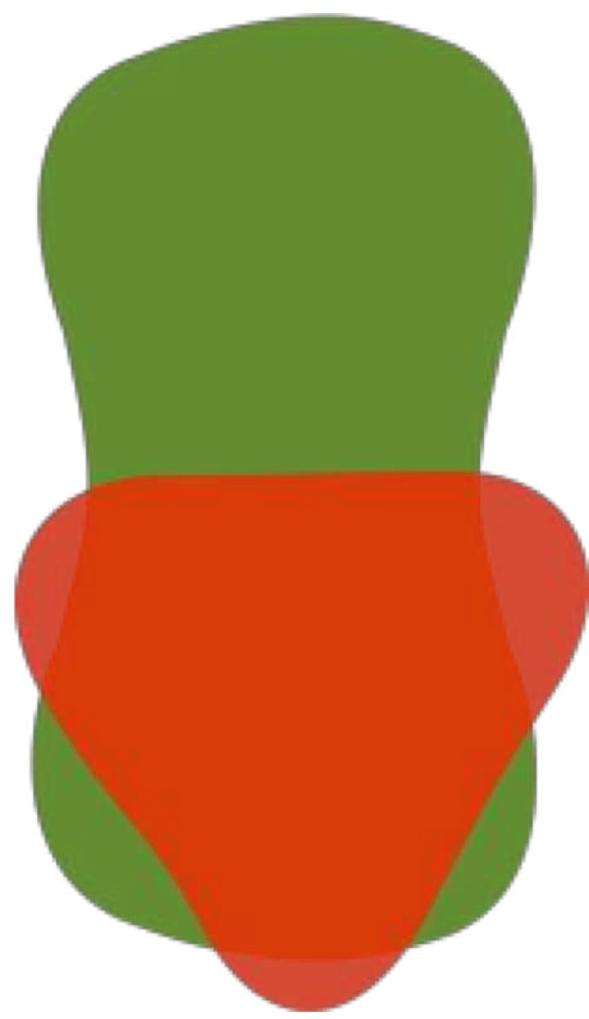
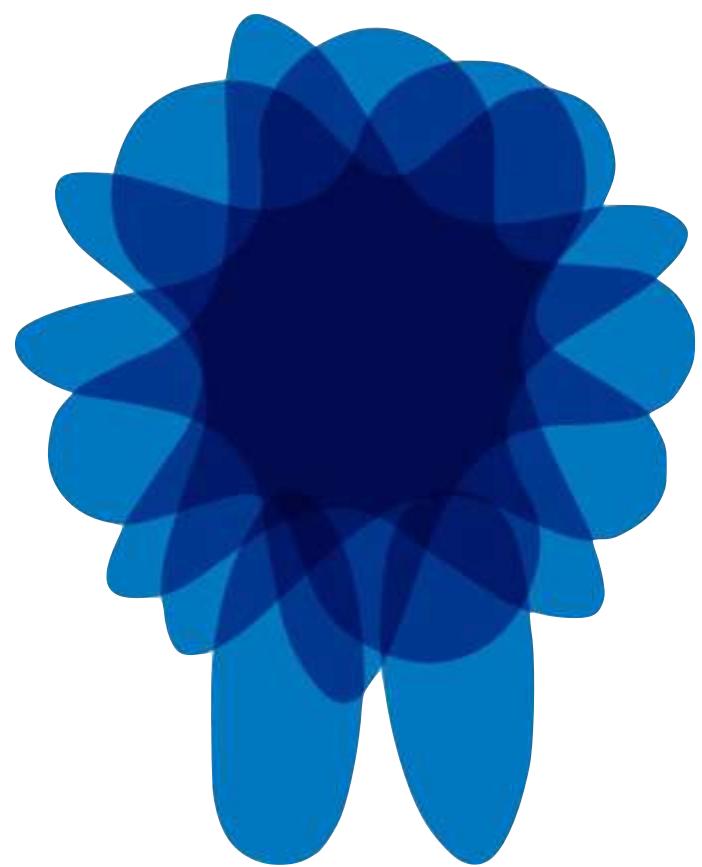






don't create toggles that depend
on other toggles.





works on all platforms &
technology stacks

Togglz - Features flag for Java

www.togglz.org

Reader



togglz Feature Flags for the Java platform

MAIN

- [Home](#)
- [Downloads](#)
- [Source Code](#)
- [Forums](#)
- [Issue Tracker](#)
- [stackoverflow.com](#)
- [Continuous Integration](#)
- [License](#)

REFERENCE

- [What's new?](#)
- [Getting Started](#)
- [Javadocs 2.0.0.Final](#)
- [Javadocs 1.1.0.Final](#)
- [Javadocs 1.0.0.Final](#)
- [Updating Notes](#)

DOCUMENTATION

Togglz

What is it about?

Togglz is an implementation of the [Feature Toggles](#) pattern for Java. Feature Toggles are a very common agile development practices in the context of continuous deployment and delivery. The basic idea is to associate a toggle with each new feature you are working on. This allows you to enable or disable these features at application runtime, even for individual users.

Want to learn more? Have a look at an [usage example](#) or check the [quickstart guide](#).

News

01-Jul-2013

Togglz 2.0.0.Final released

I'm very happy to announce the release of Togglz 2.0.0.Final. This new version is the result of many months of hard work. Many core concepts of Togglz have been revised to provide much more flexibility.

The most noteworthy change in Togglz 2.0.0.Final is the new extendible feature activation mechanism that allows to implement custom strategies for activating features. Beside that there are many other updates.

Togglz - Features flag for Java

www.togglz.org/documentation/activation-strategies.html

Reader



togg_lz Feature Flags for the Java platform

MAIN

- [Home](#)
- [Downloads](#)
- [Source Code](#)
- [Forums](#)
- [Issue Tracker](#)
- [stackoverflow.com](#)
- [Continuous Integration](#)
- [License](#)

REFERENCE

- [What's new?](#)
- [Getting Started](#)
- [Javadocs 2.0.0.Final](#)
- [Javadocs 1.1.0.Final](#)
- [Javadocs 1.0.0.Final](#)
- [Updating Notes](#)

DOCUMENTATION

- [Overview](#)
- [Installation](#)
- [Configuration](#)
- [Usage](#)
- [Admin Console](#)

Activation Strategies

Togglz defines the concept of *activation strategies*. They are responsible to decide whether an enabled feature is active or not. Activation strategies can for example be used to activate features only for specific users, for specific client IPs or at a specified time.

Togglz ships with the following default strategies:

- Username
- Gradual rollout
- Release date
- Client IP
- Server IP
- ScriptEngine

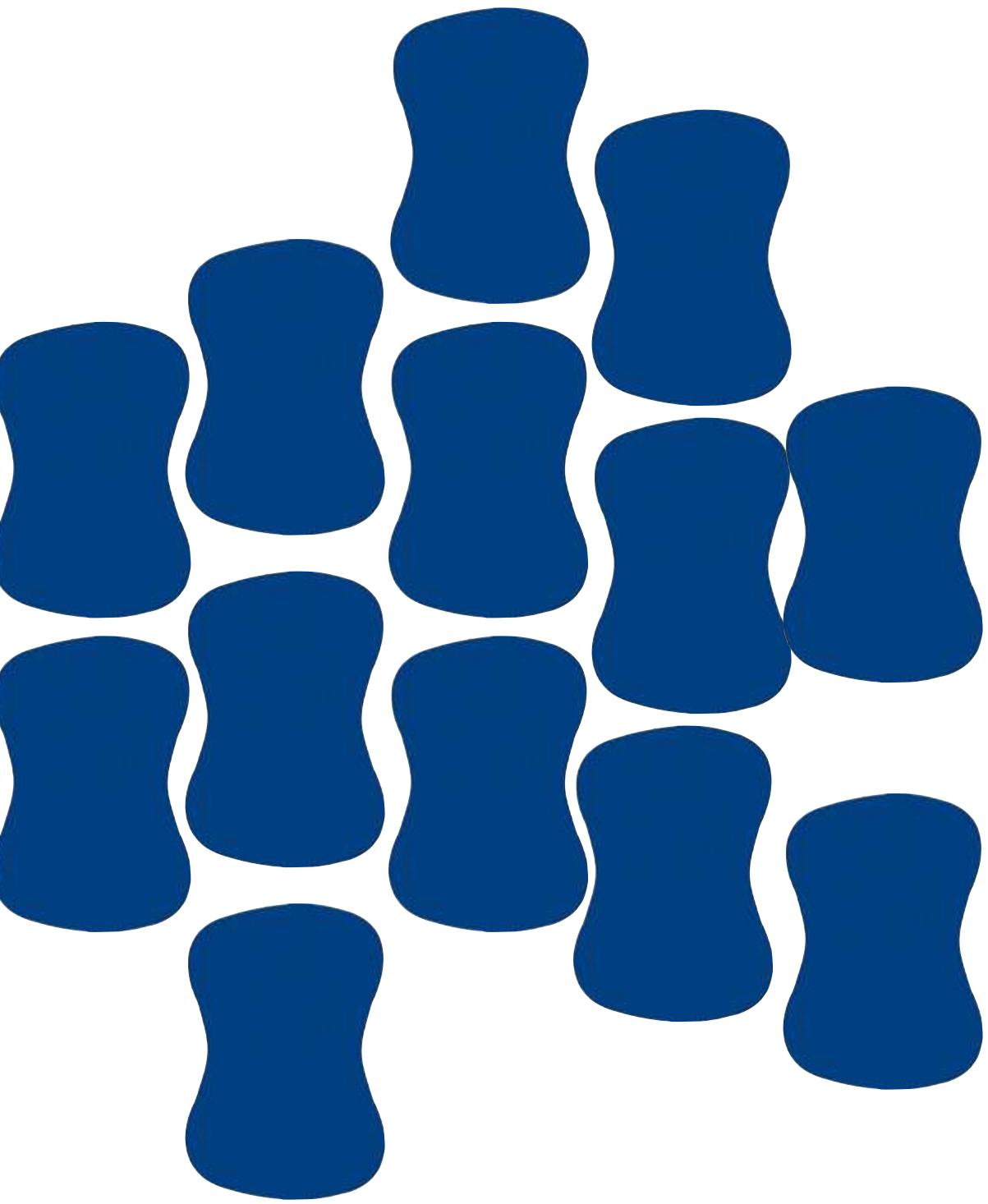
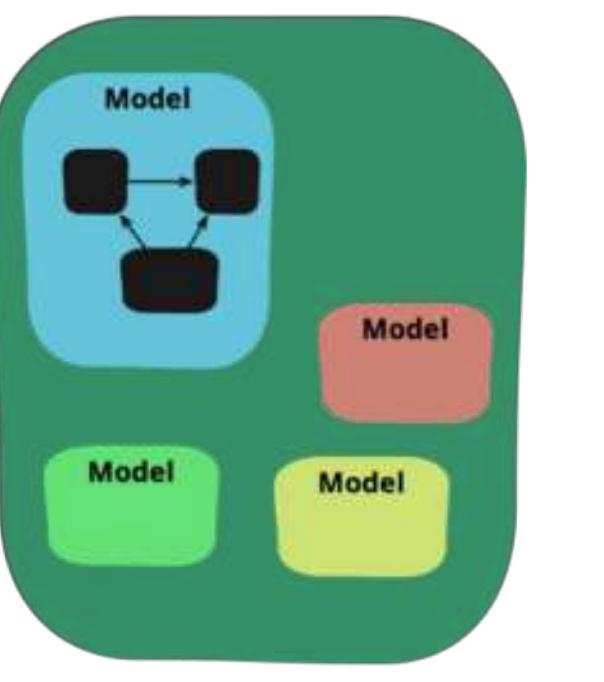
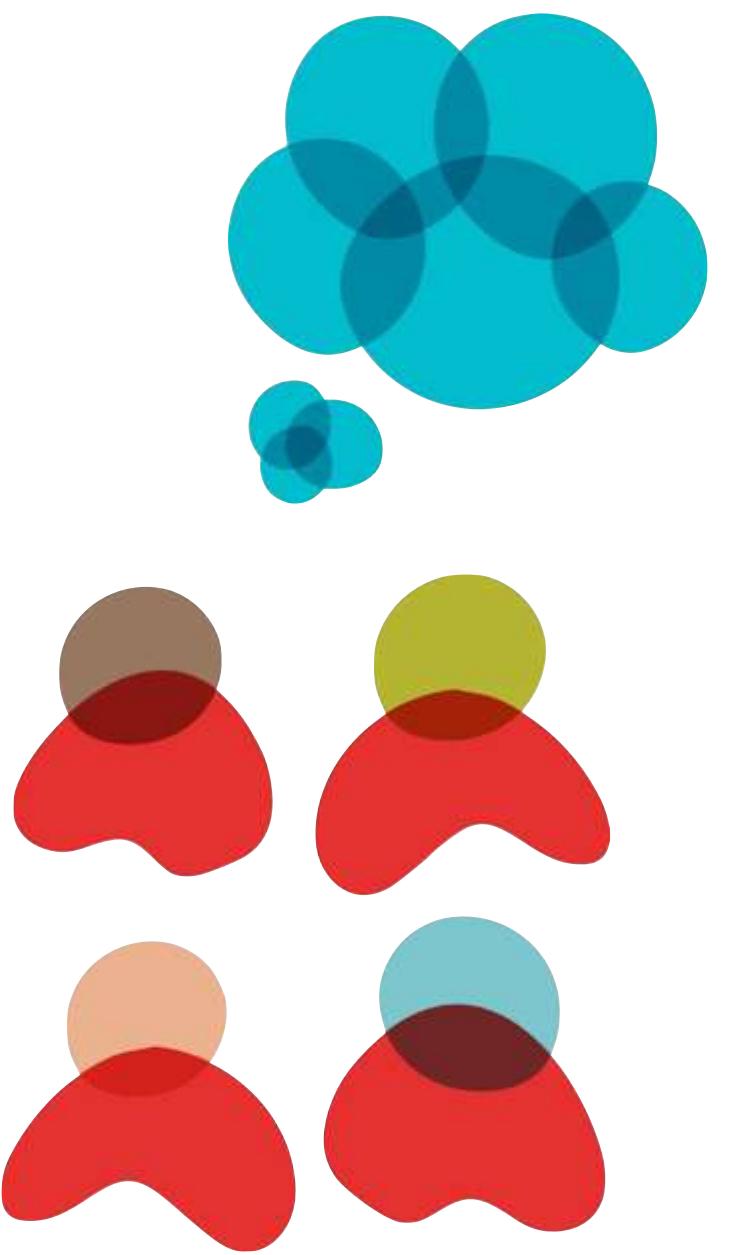
The following sections will describe each strategy in detail. The last section [custom strategies](#) describes how to build your own strategies.

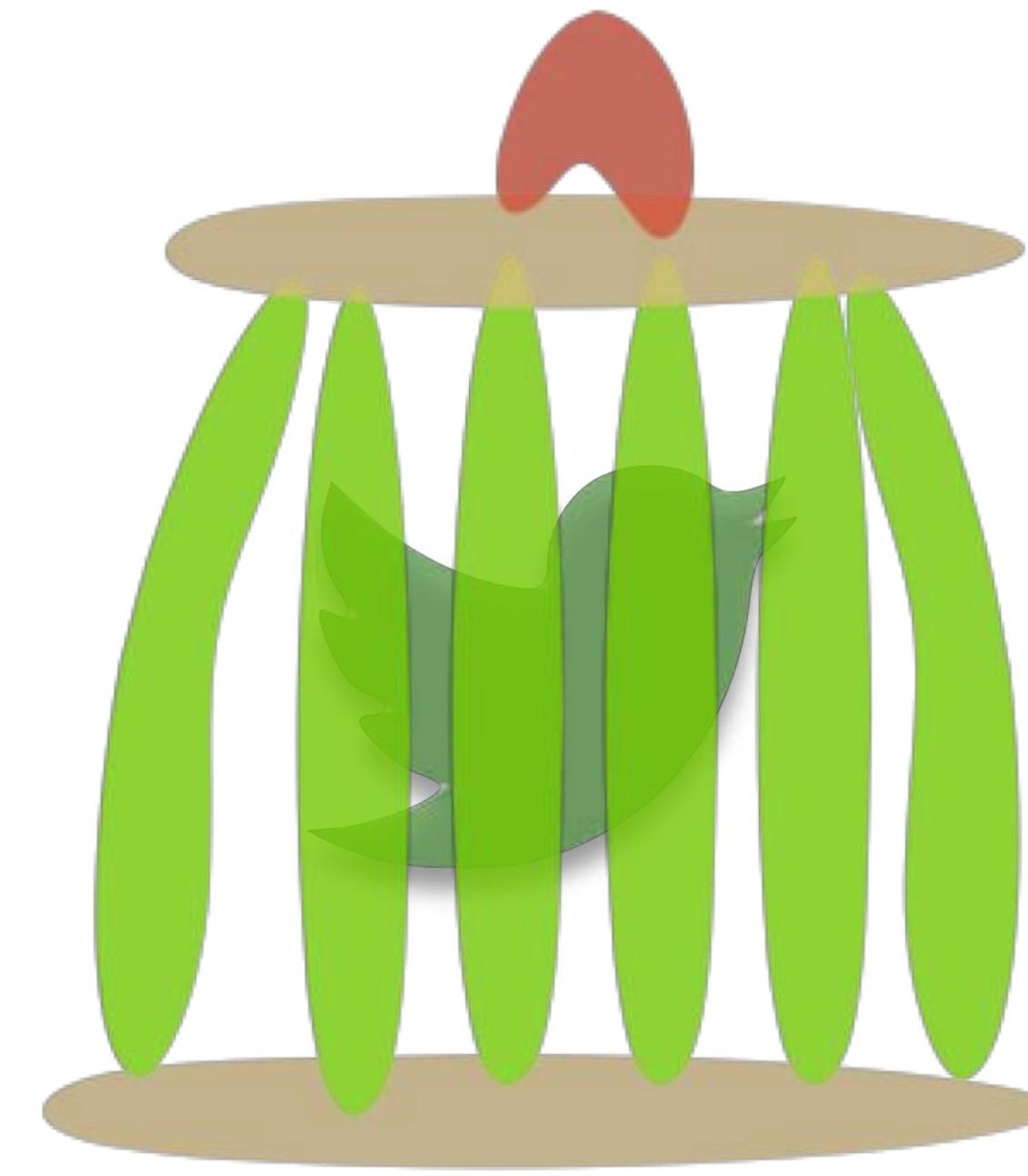
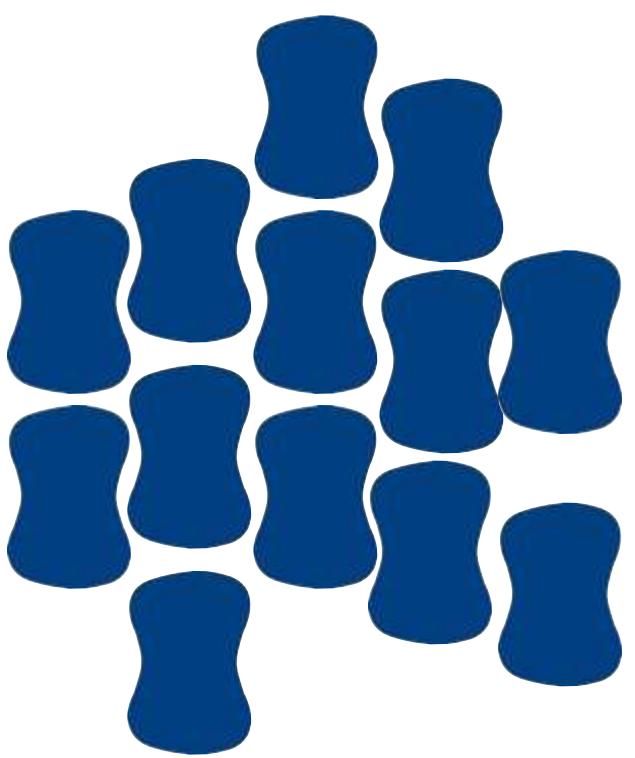
Username

Enabling features for specific users was already supported in very early versions of Togglz, even before the activation strategy concept was introduced in Togglz 2.0.0.

If you select this strategy for a feature, you can specify a comma-separated list of users for which the feature should be active. Togglz will use the `UserProvider` you configured for the FeatureManager to determine the current user and compare it to that list.

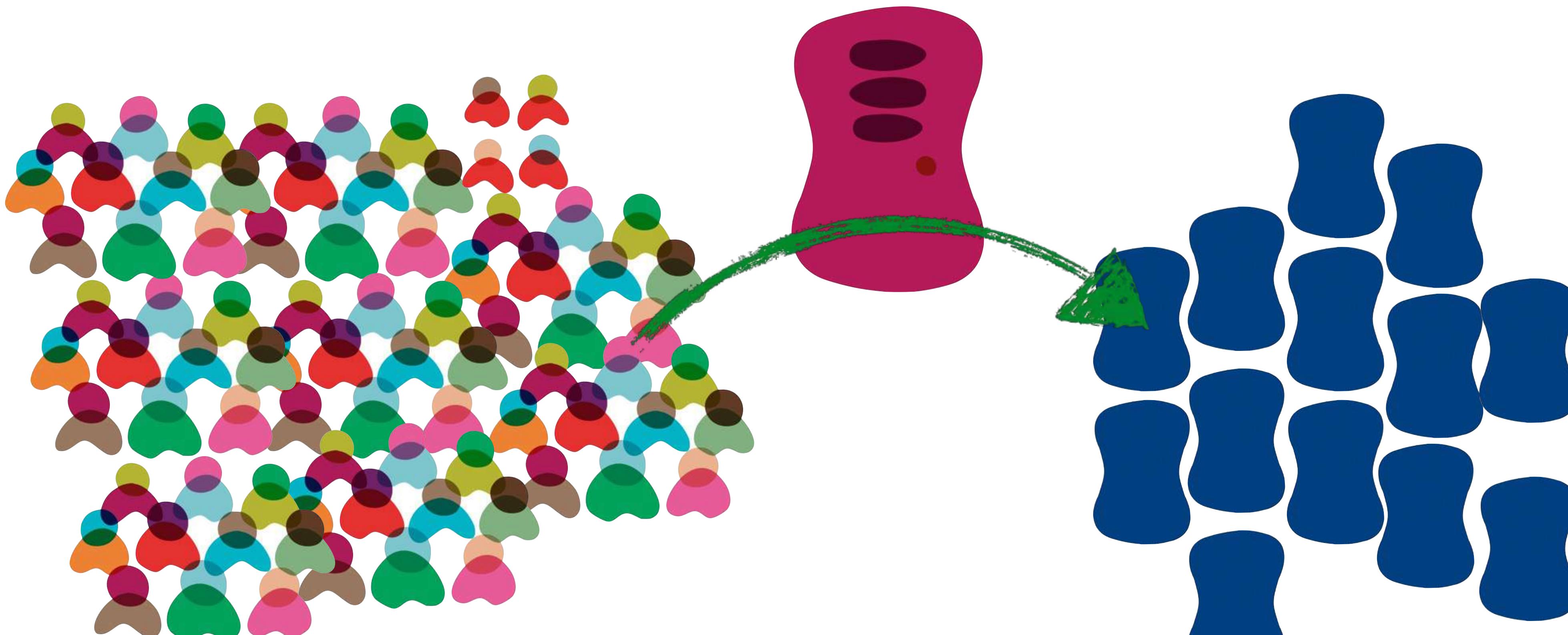
Please note that Togglz will take case into account when comparing the usernames. So the users `admin` and `Admin` are NOT the same.



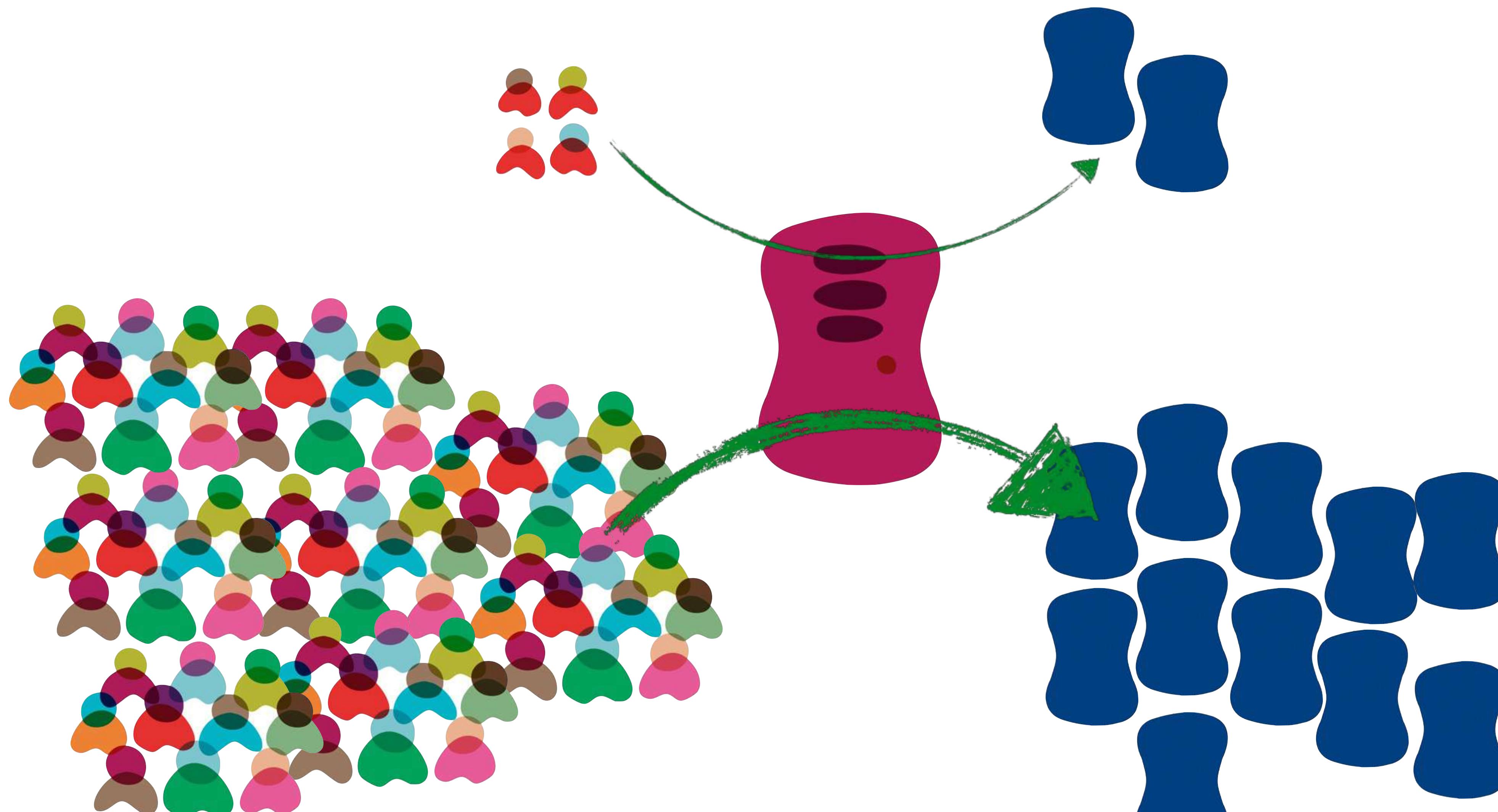


canary releasing

canary releasing



canary releasing



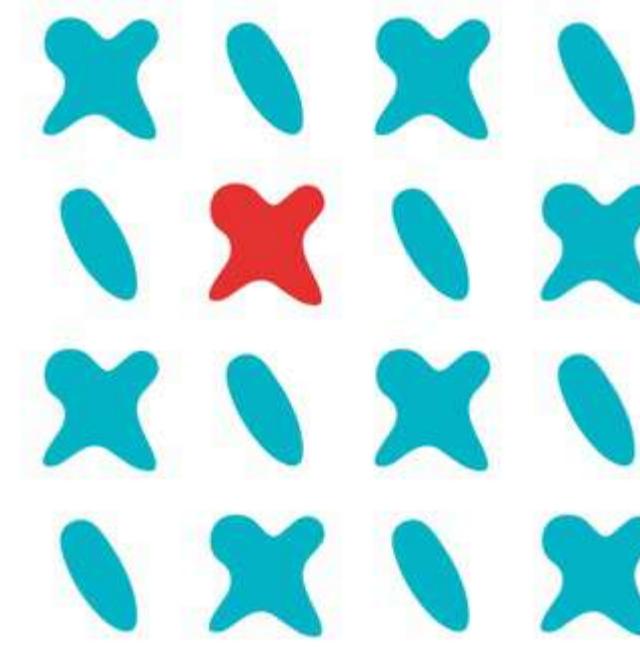
canary releasing

reduce risk of release



canary releasing

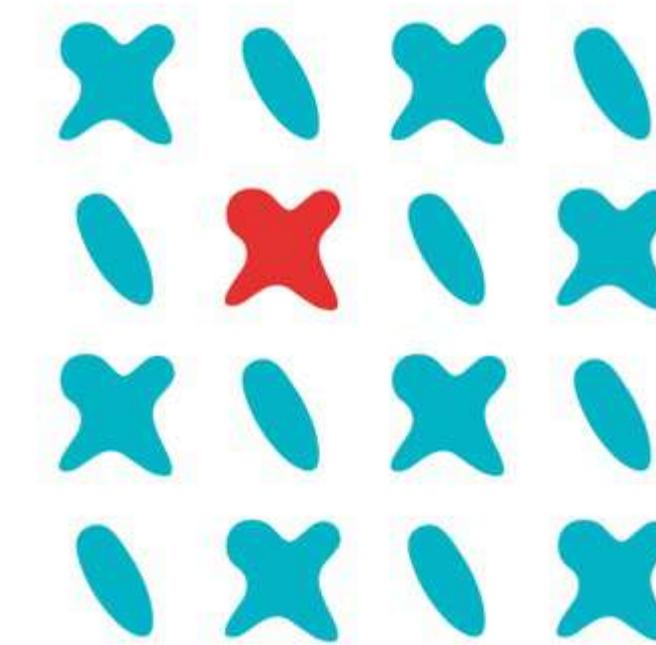
reduce risk of release



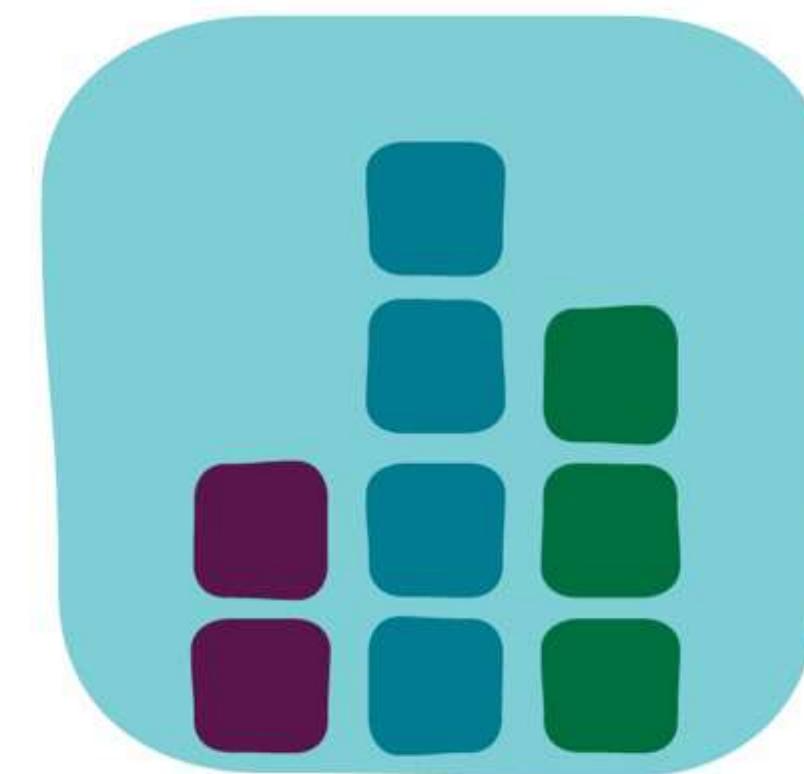
multi-variant testing

canary releasing

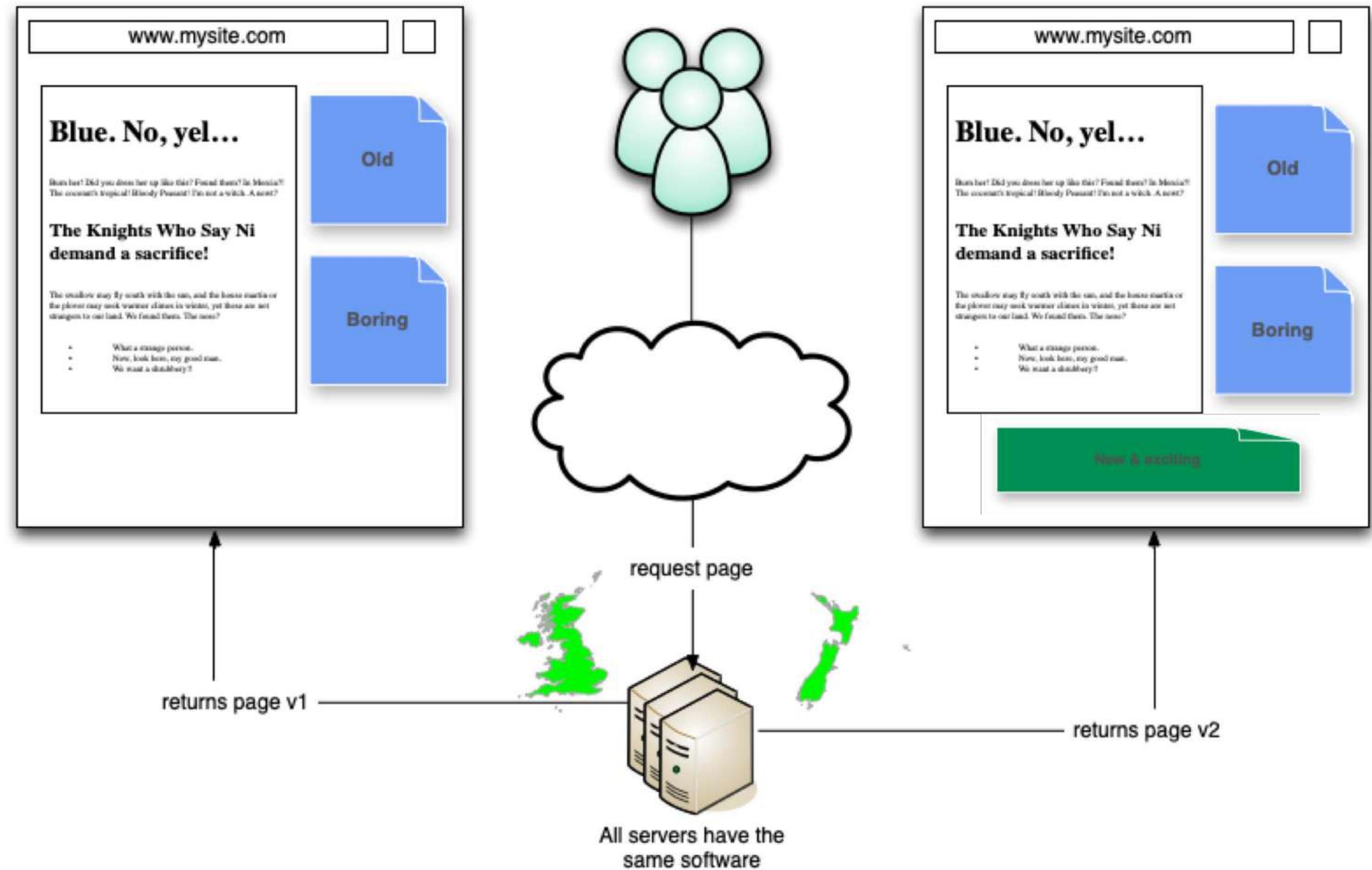
reduce risk of release



multi-variant testing



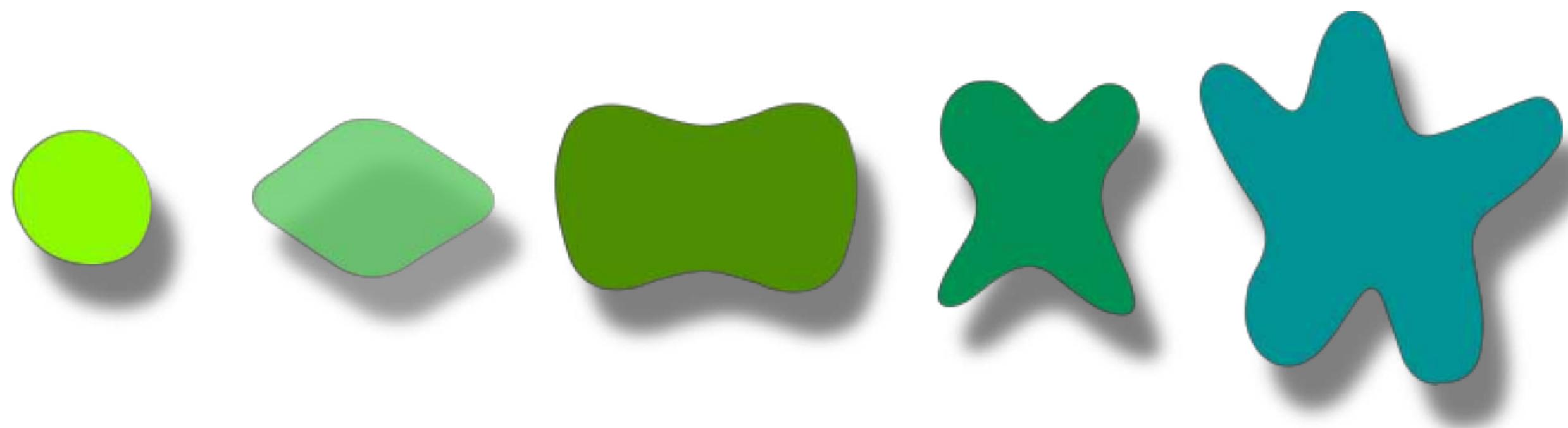
performance testing



Question:

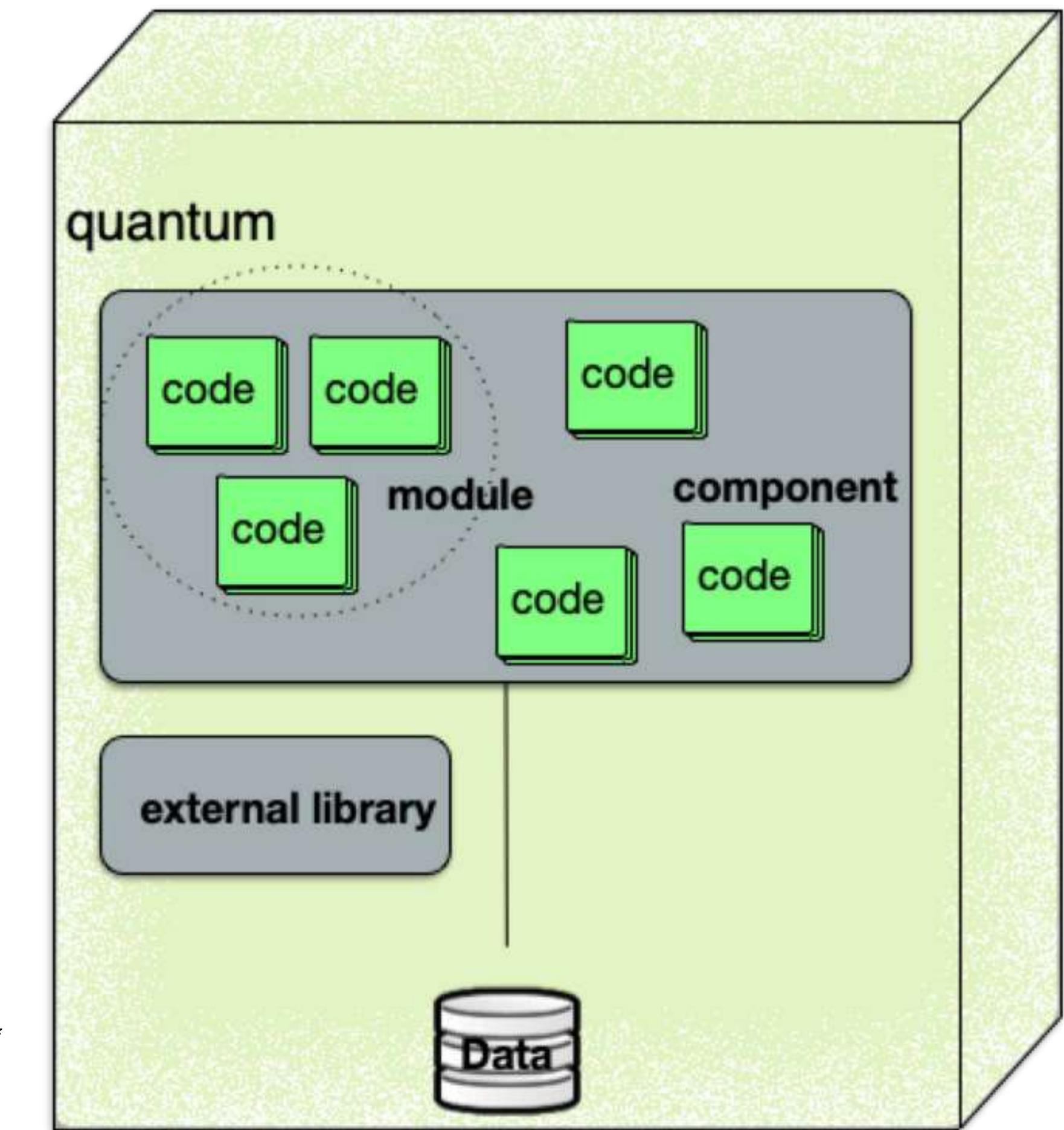
What's the most important characteristics of fitness functions?

two big ideas

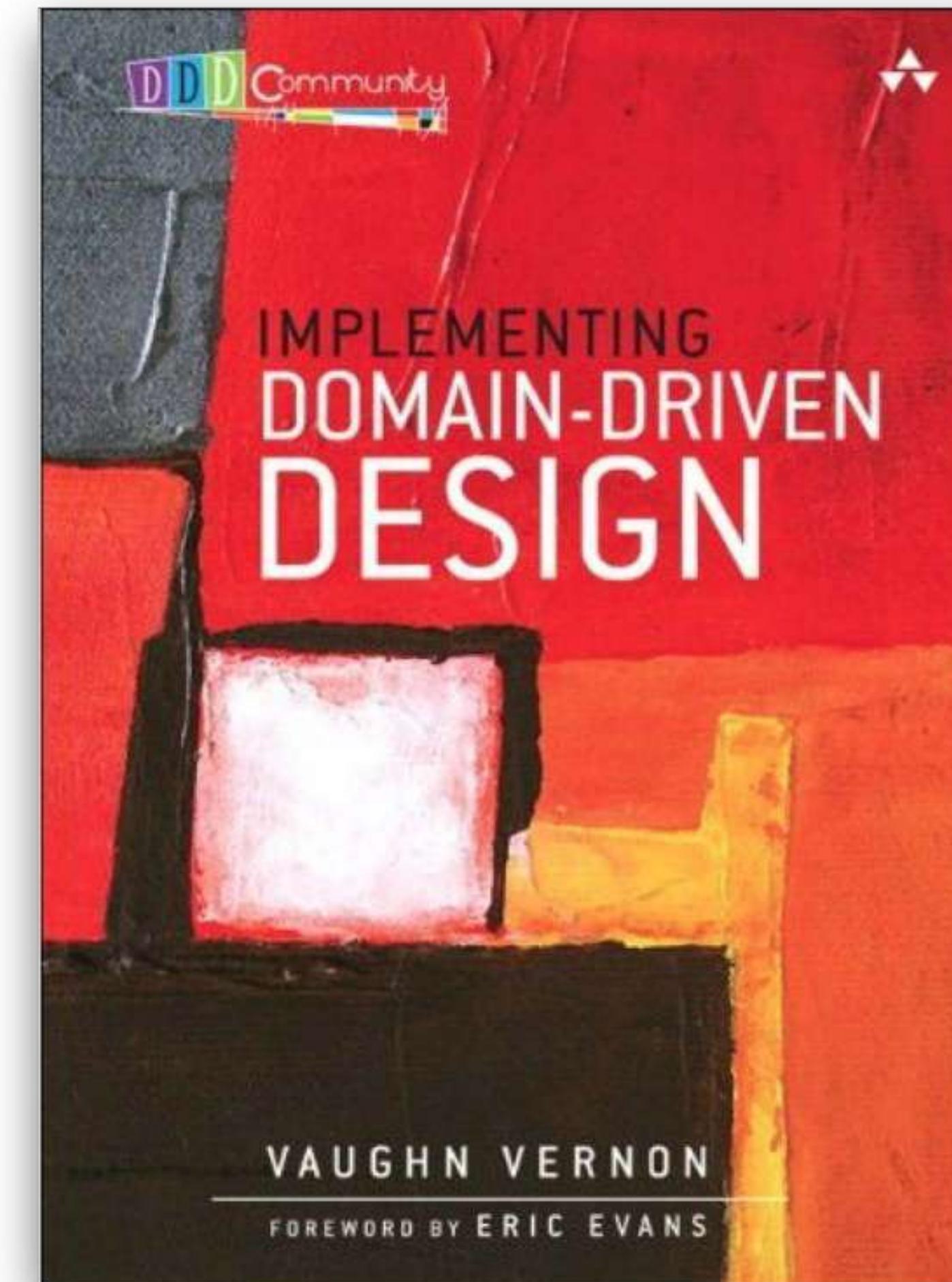
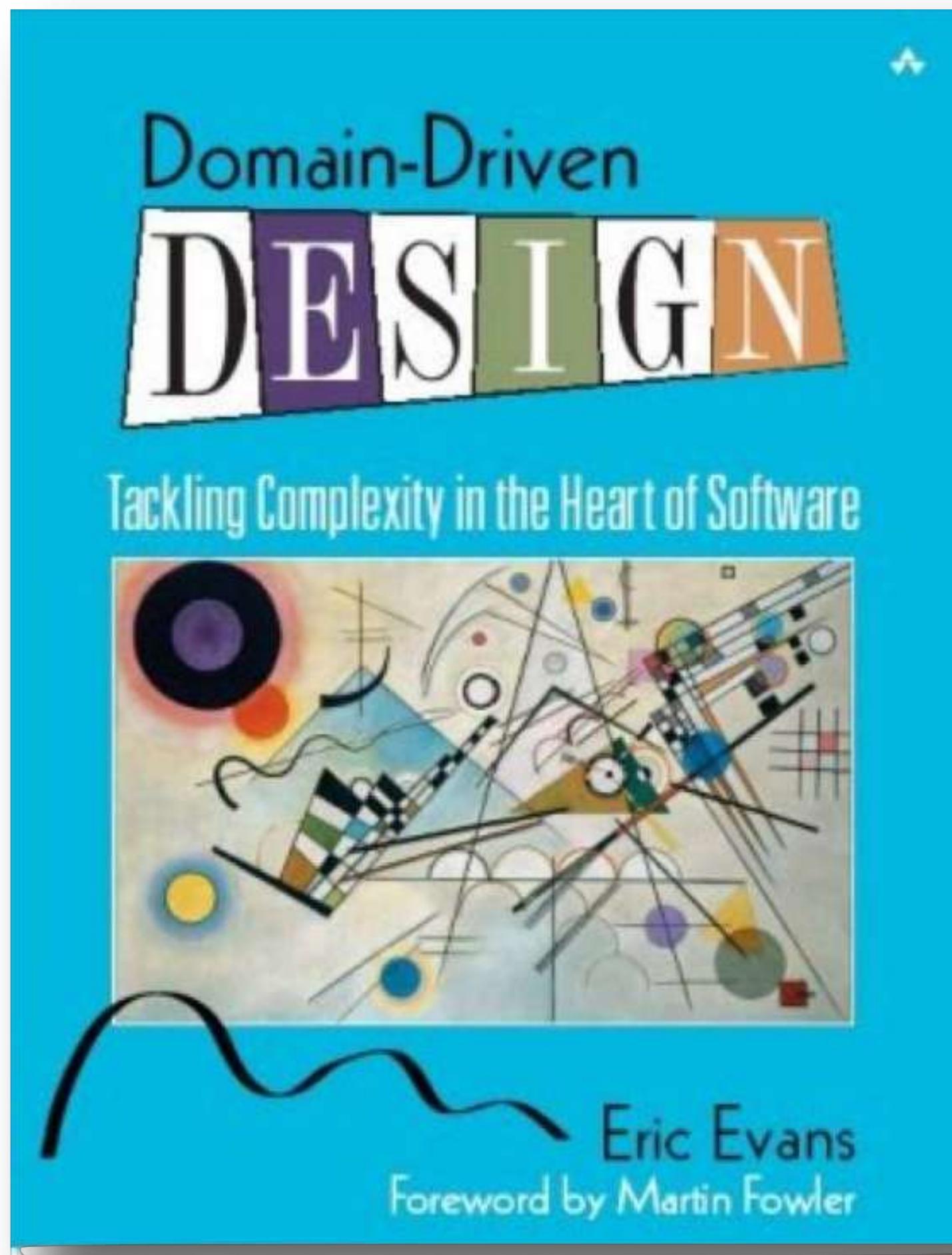


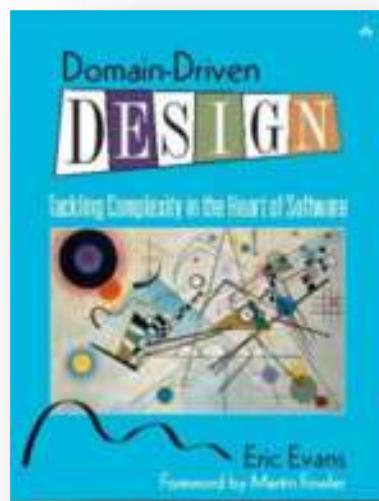
**fitness functions for
evolvability**

**architectural
quantum**



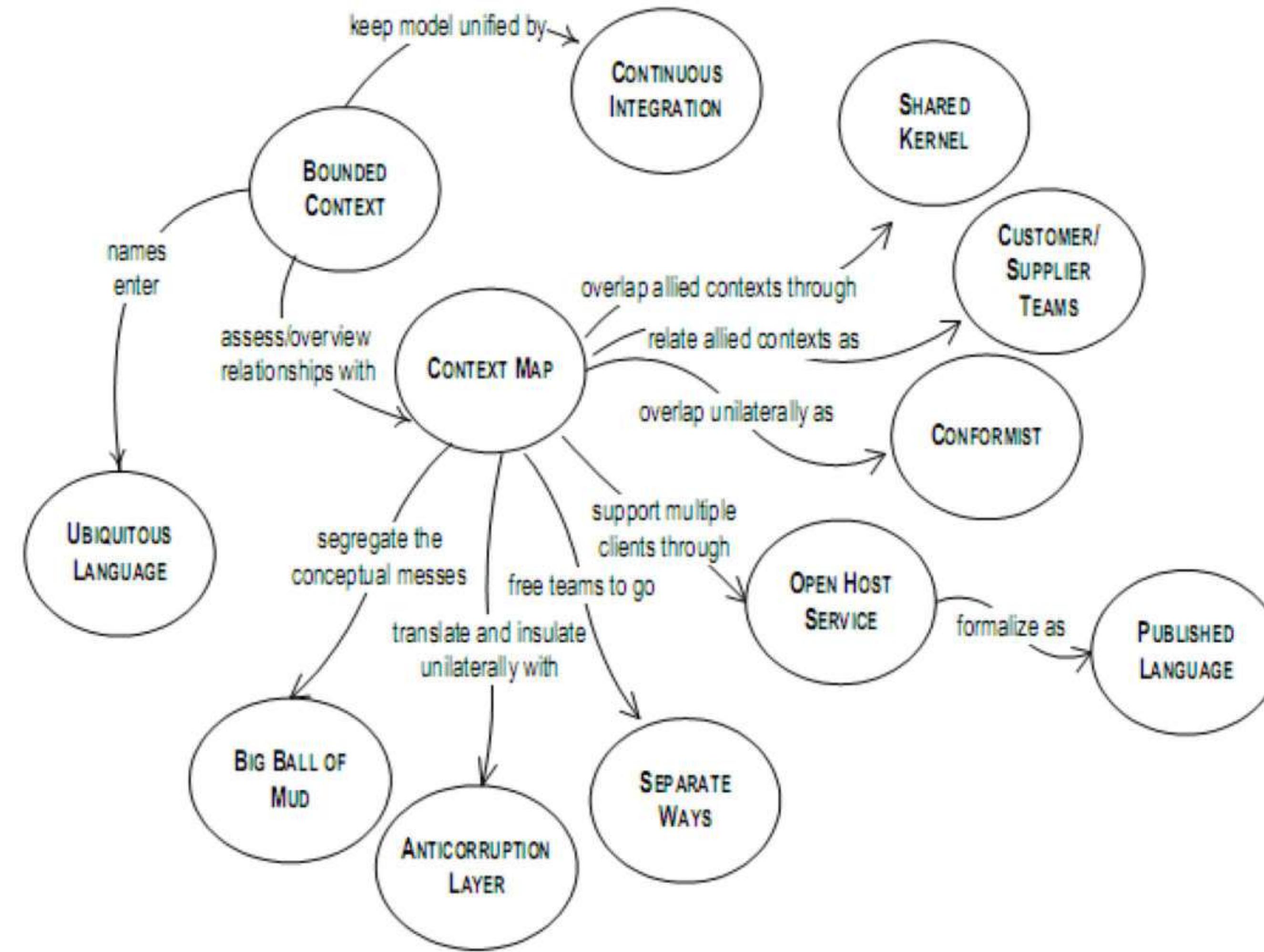
Domain Driven Design





bounded context

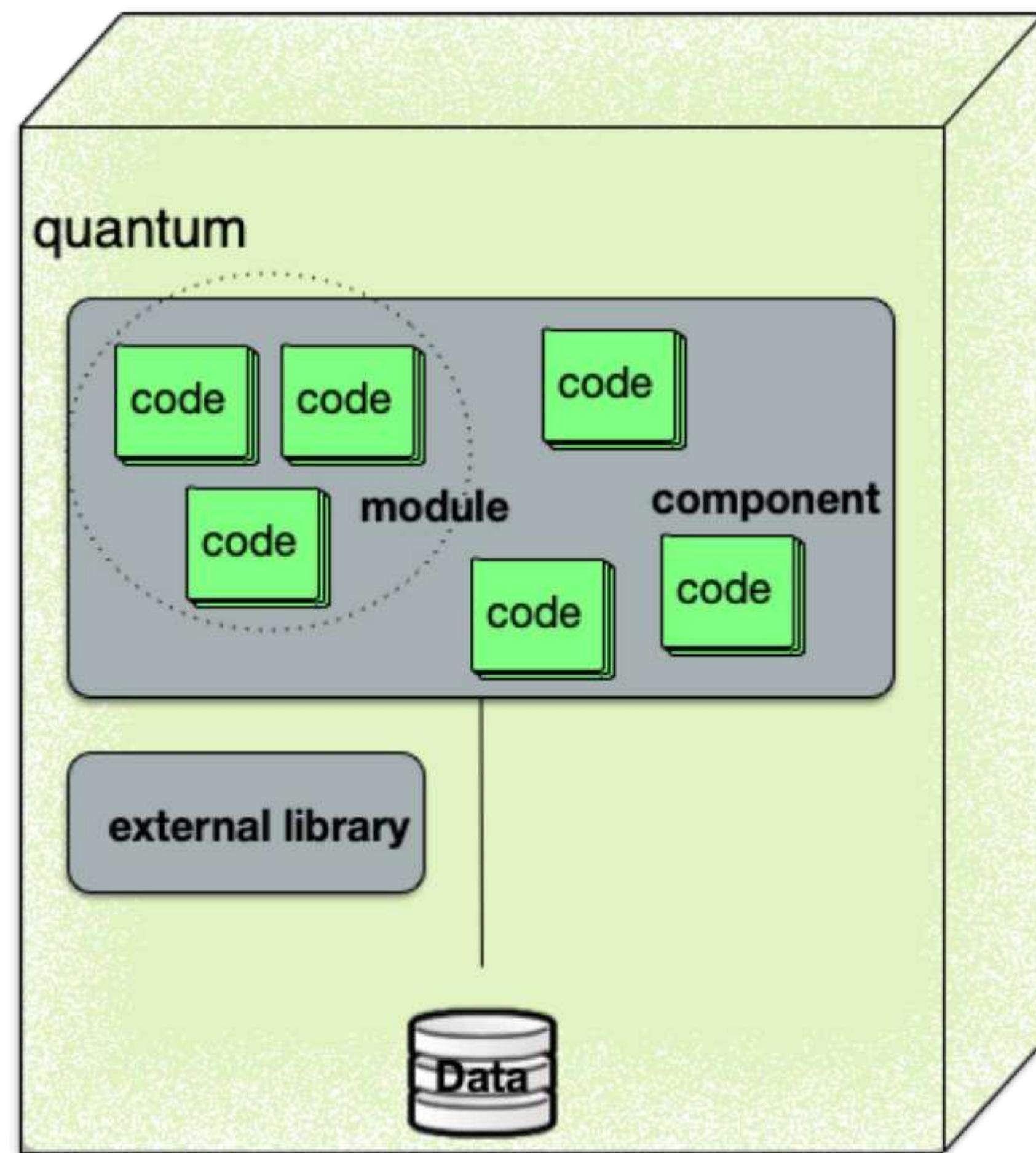
Maintaining Model Integrity



architectural quantum

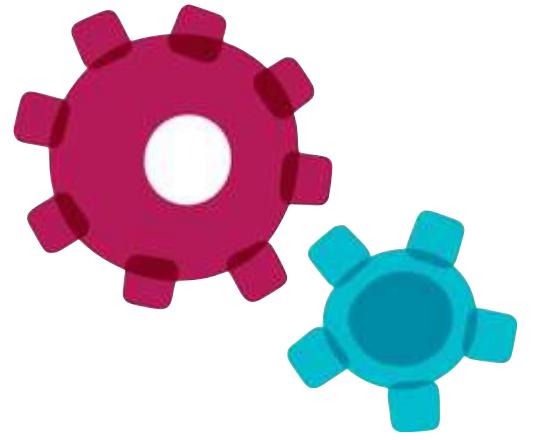
An architectural quantum is an independently deployable component with synchronous cohesion, which includes all the structural elements required for the system to function properly.

architectural quantum



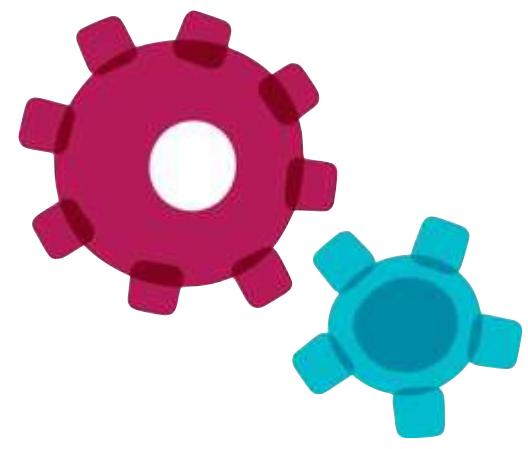
why quantum?

why quantum?

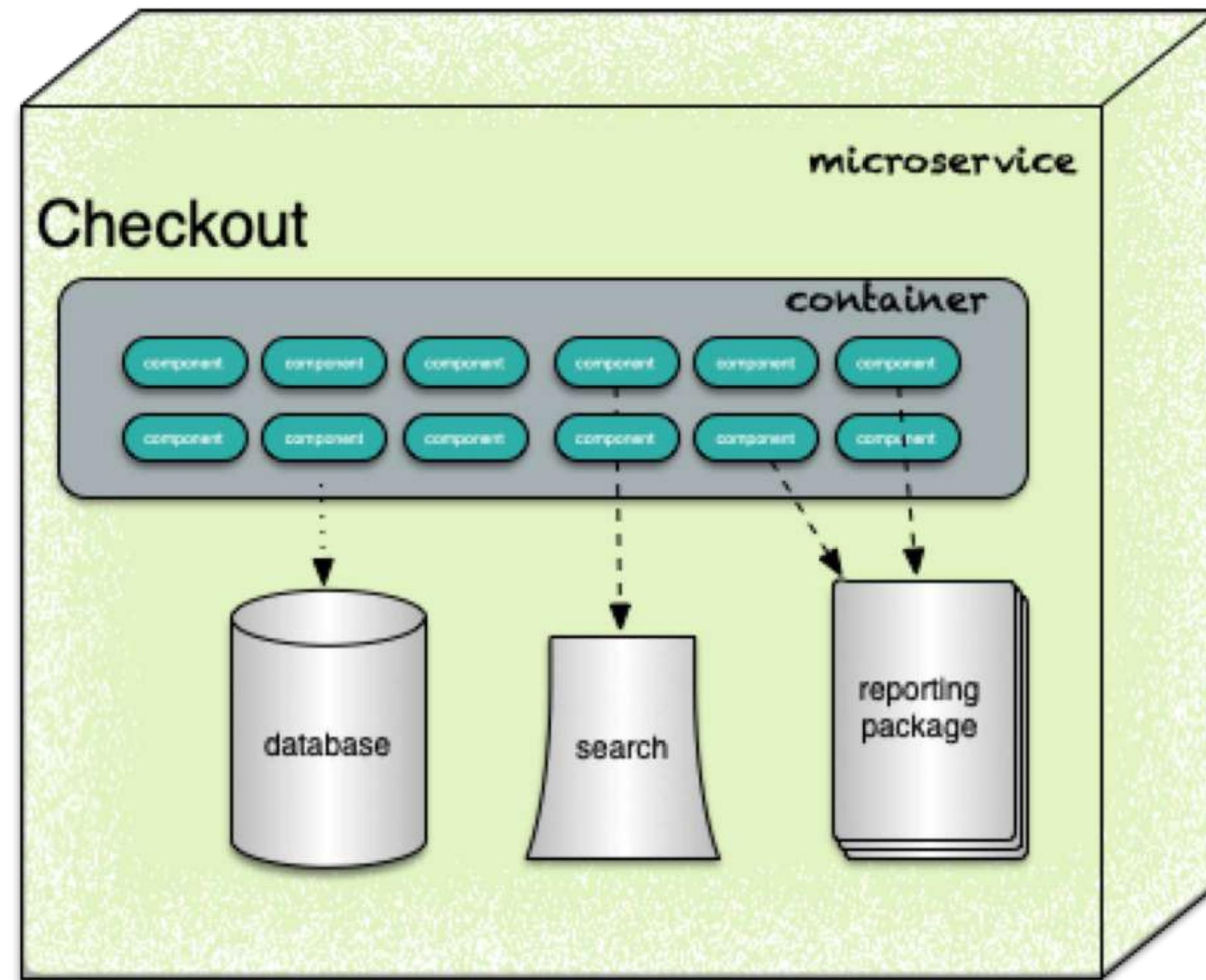


operational view
of architecture

why quantum?



operational view
of architecture



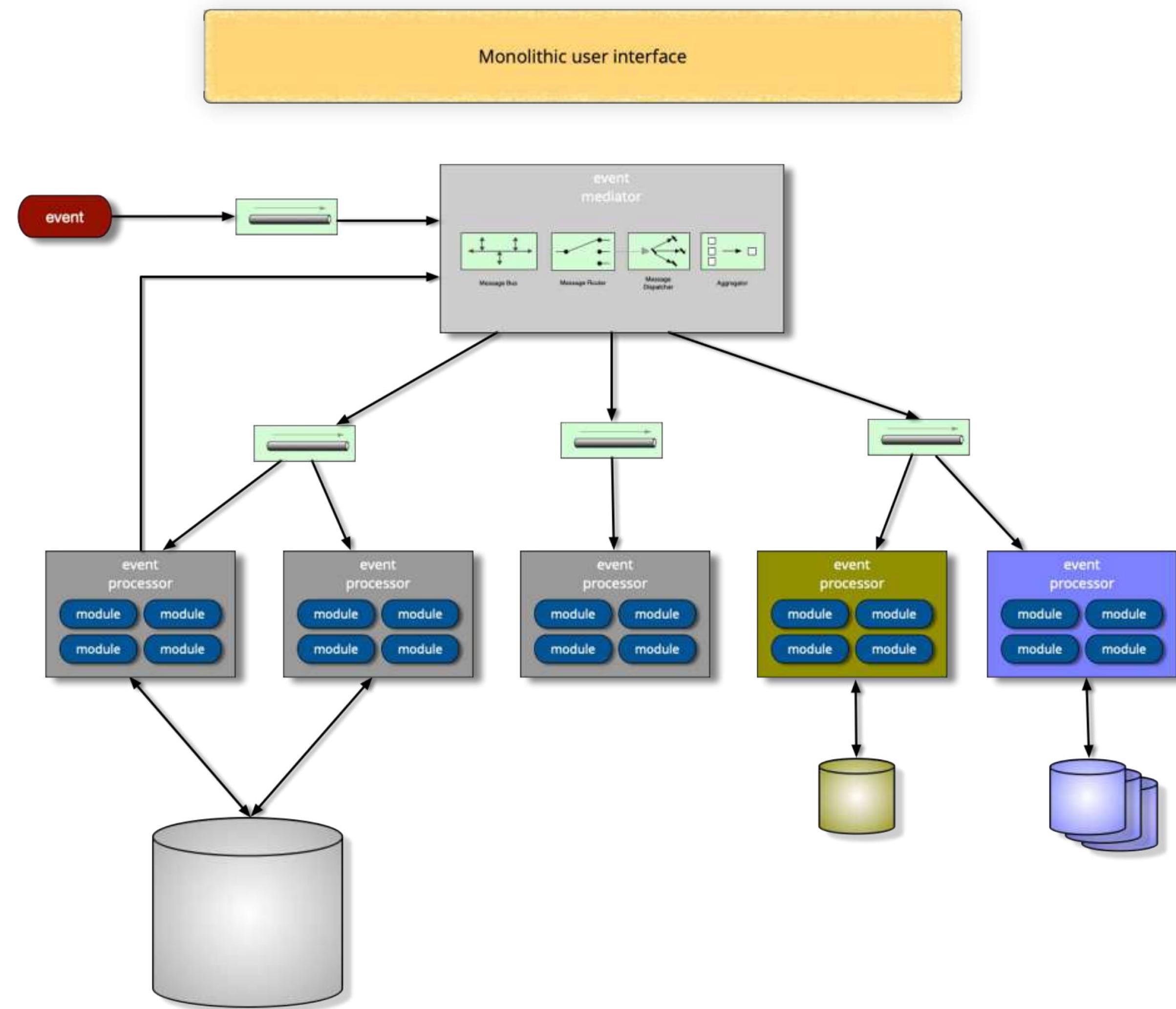
why quantum?

holistic



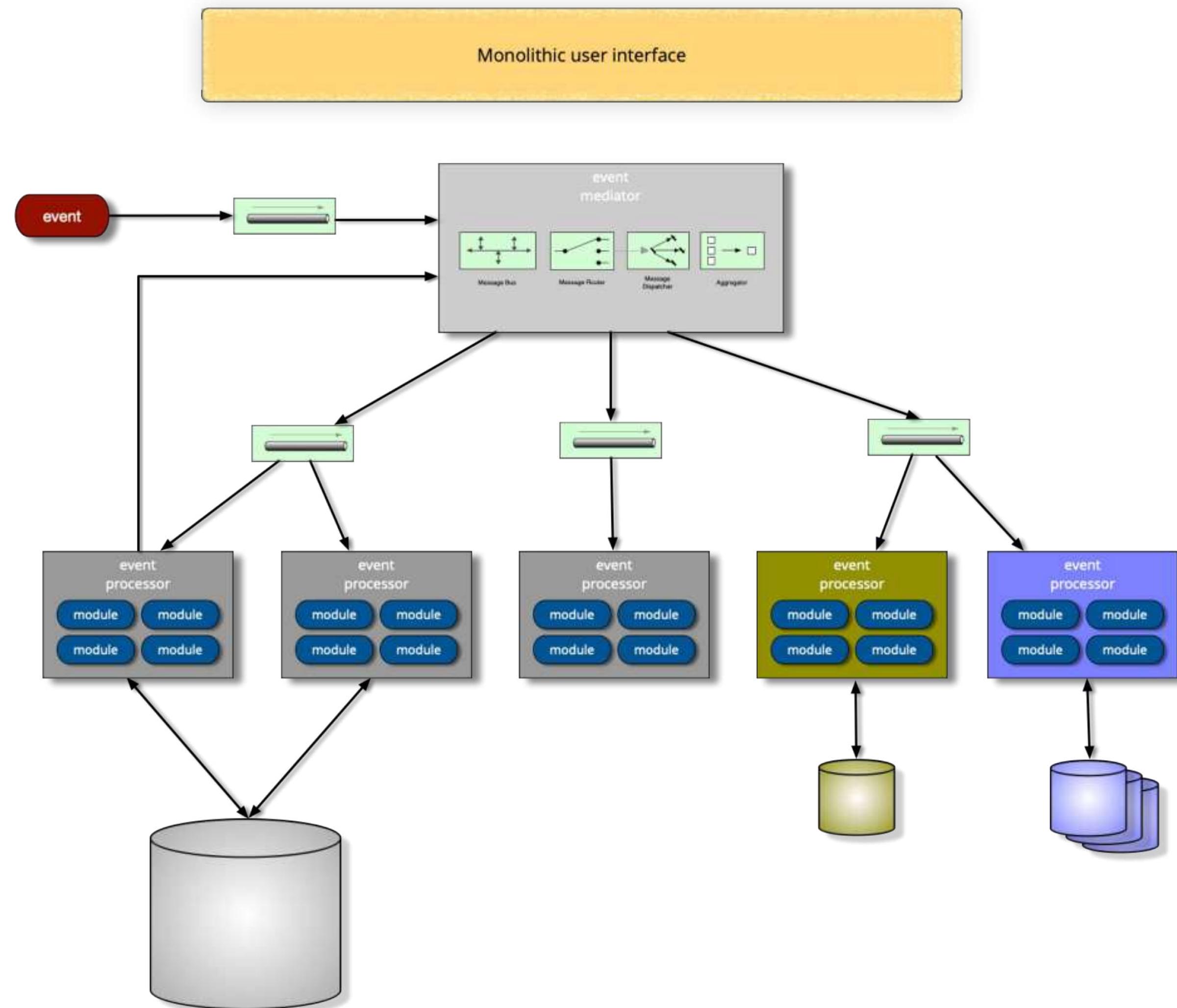
why quantum?

holistic



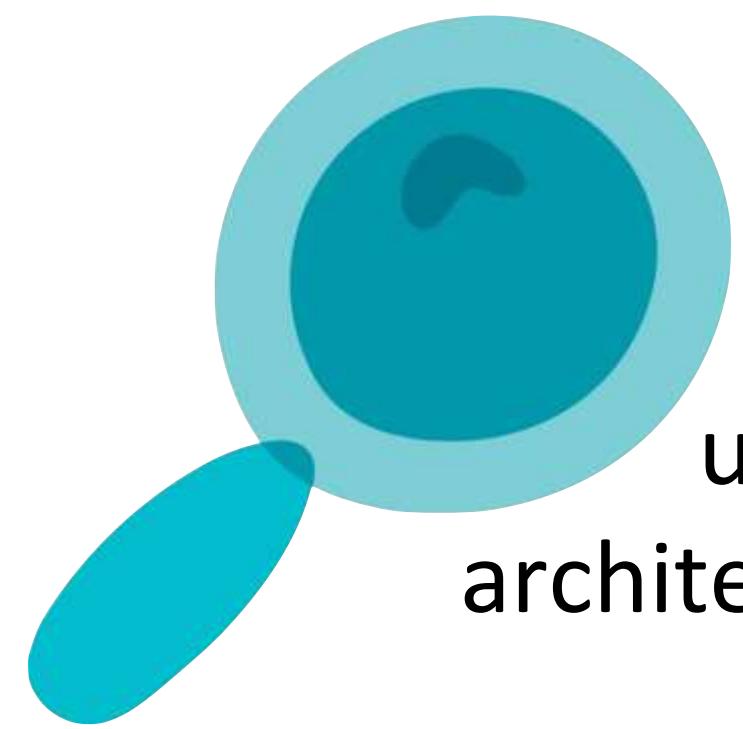
why quantum?

holistic



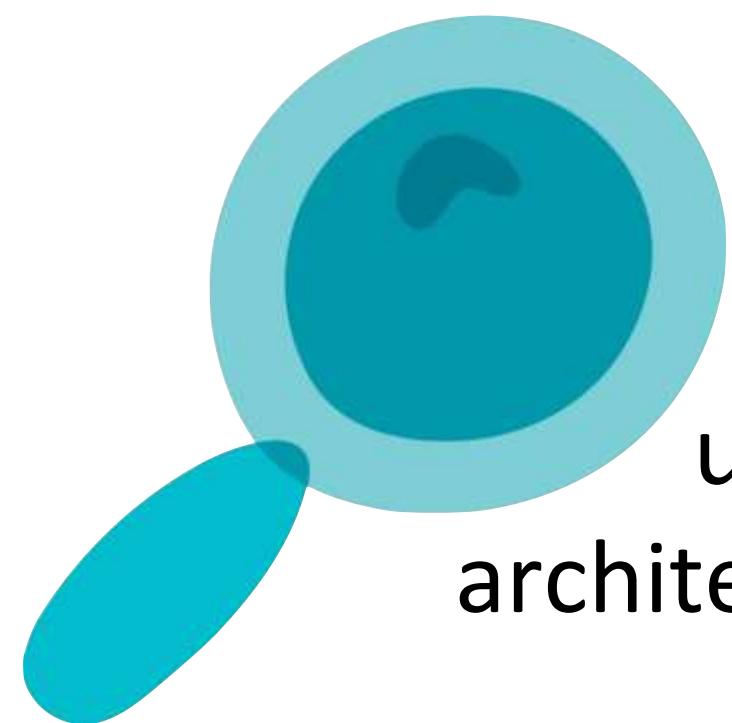
"multiple dimensions"

why quantum?

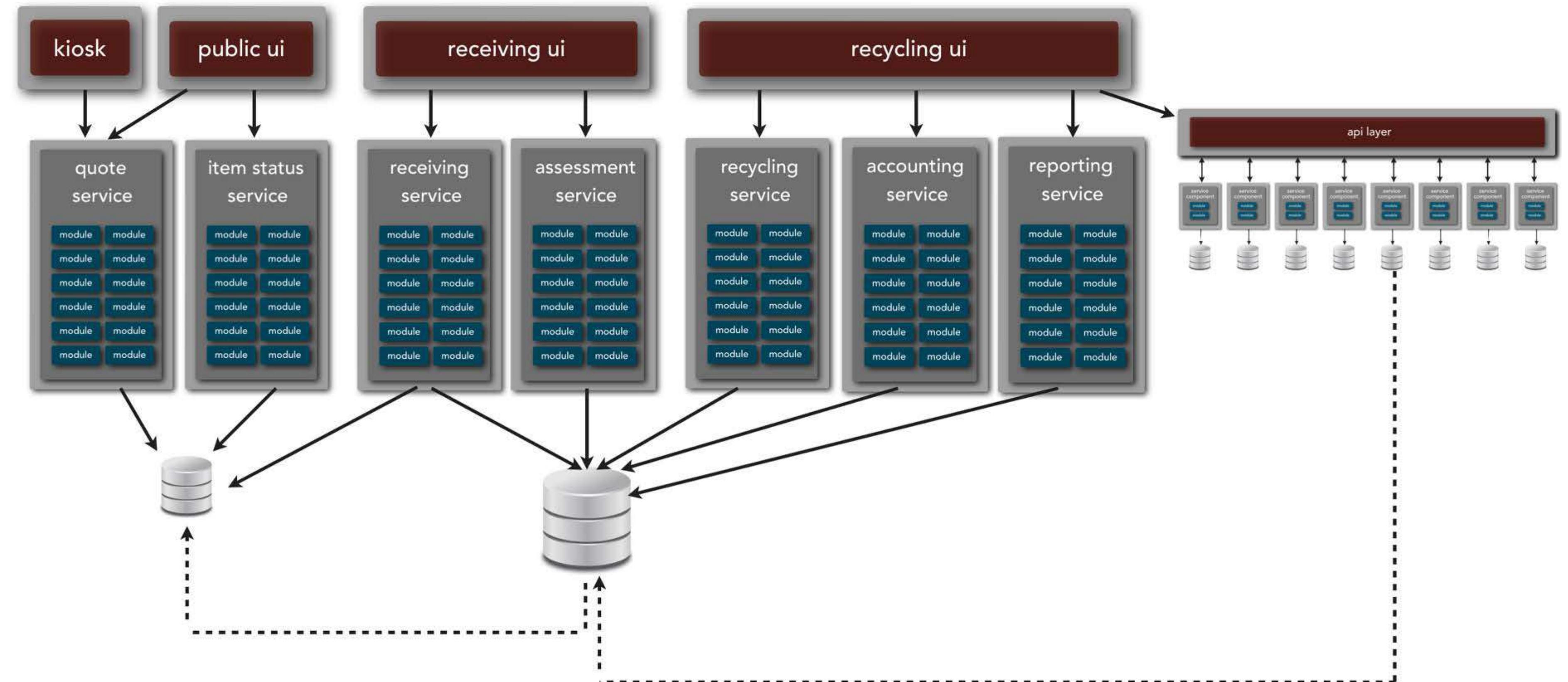


useful for
architectural analysis

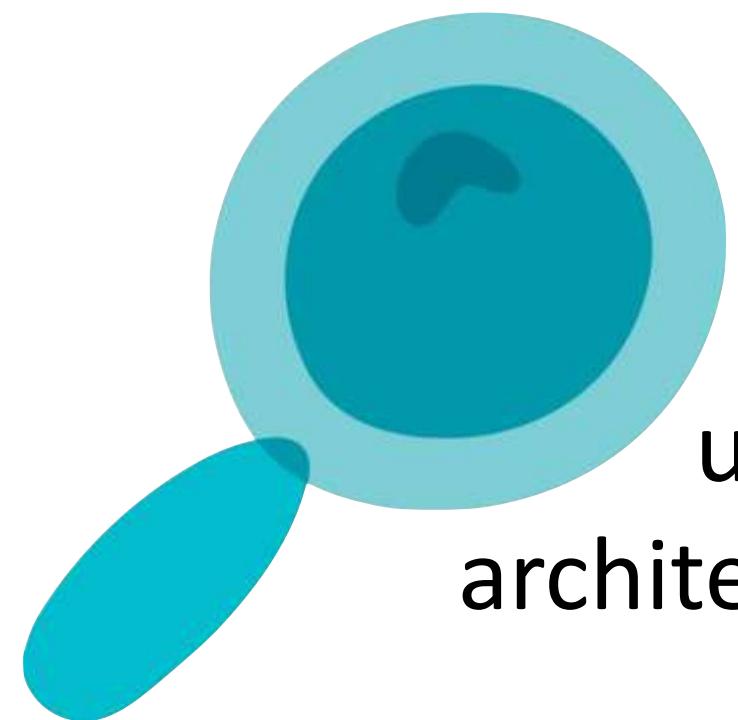
why quantum?



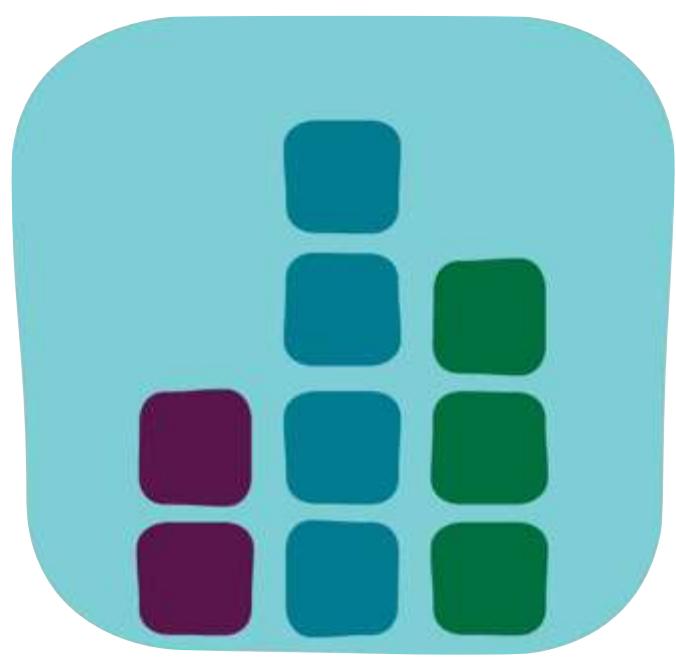
useful for
architectural analysis



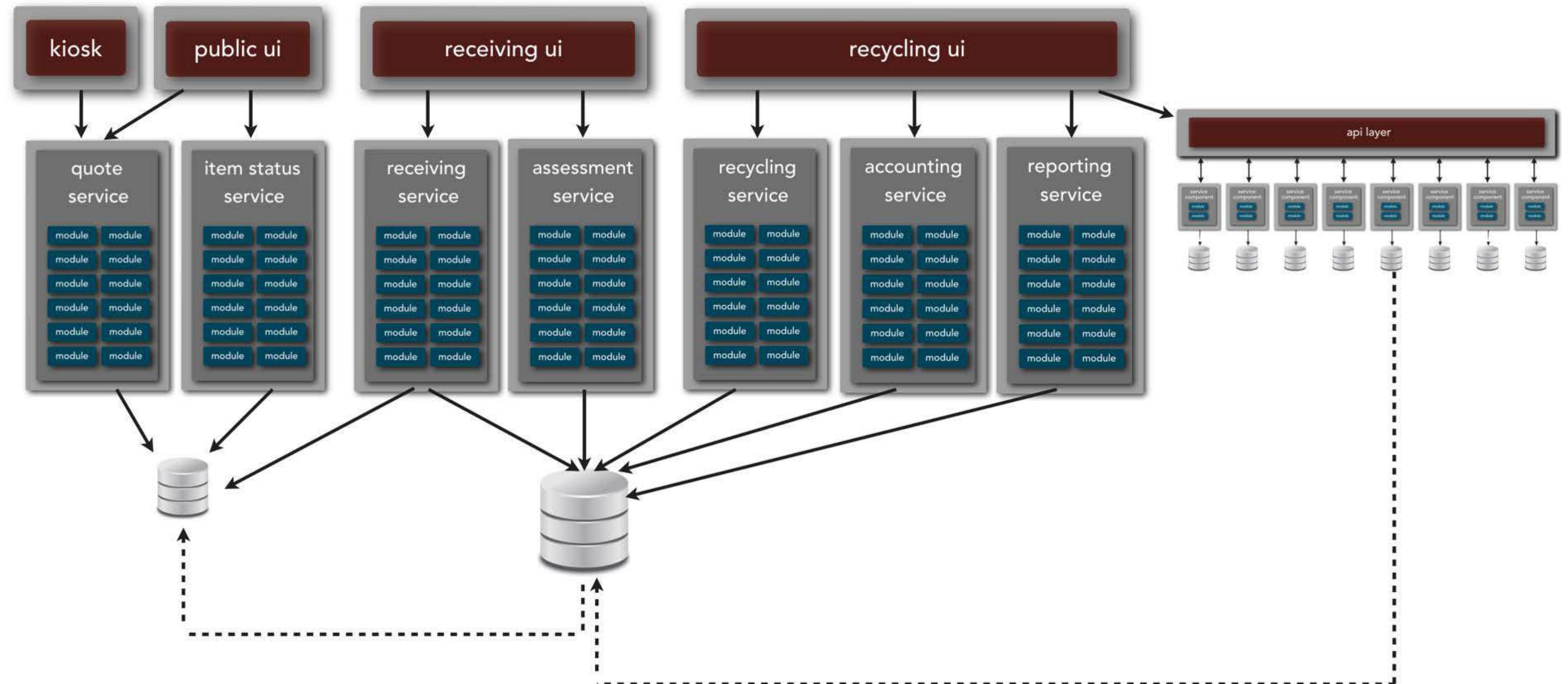
why quantum?



useful for
architectural analysis

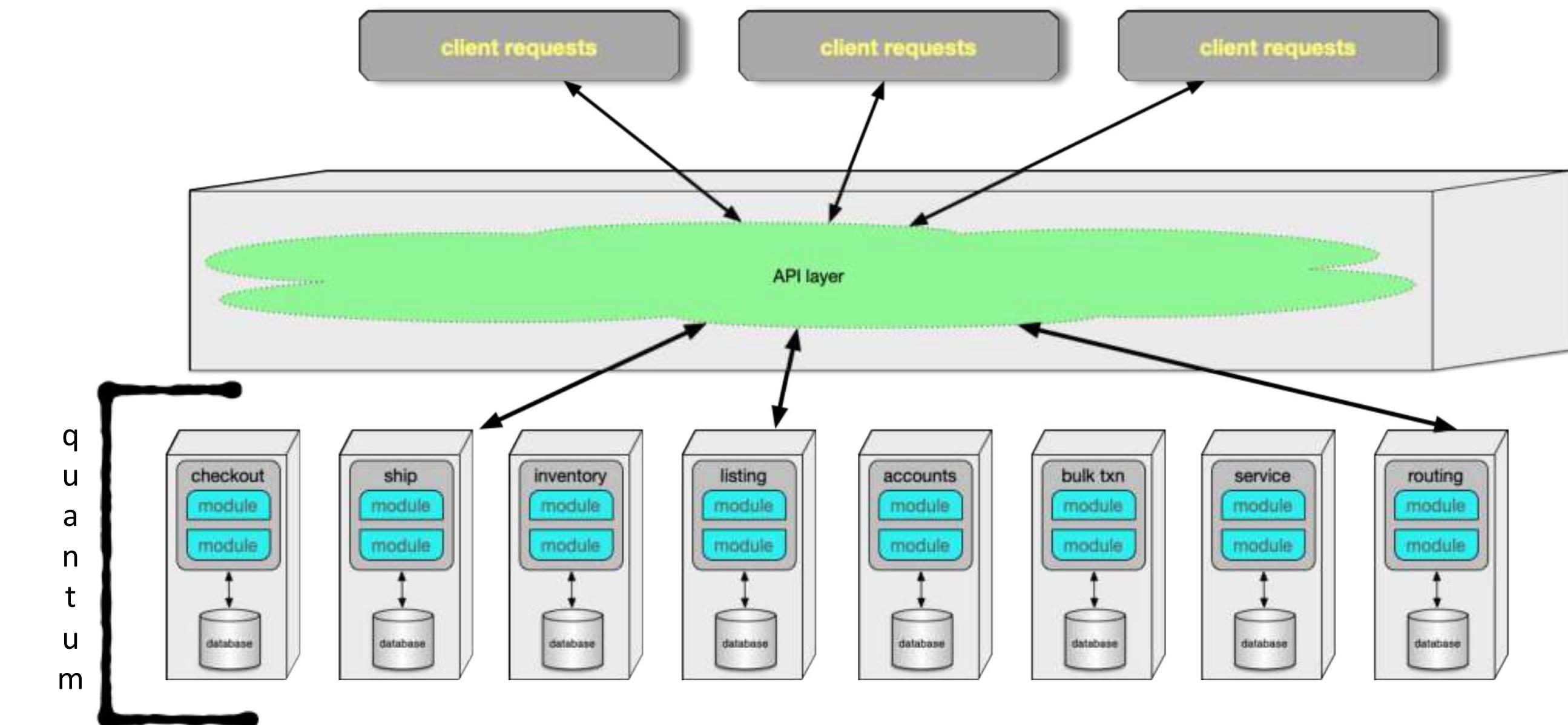
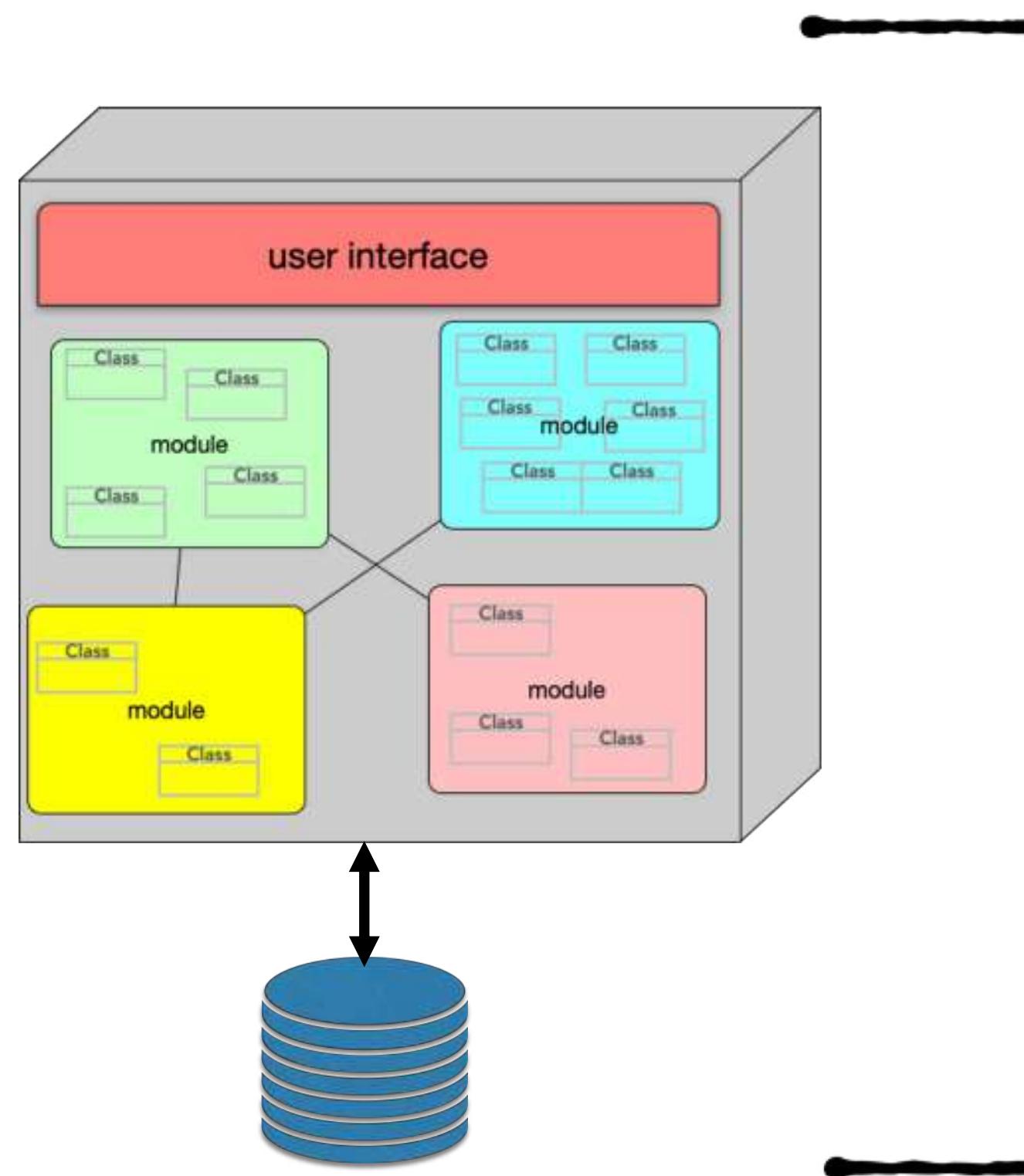


helps analyze
coupling

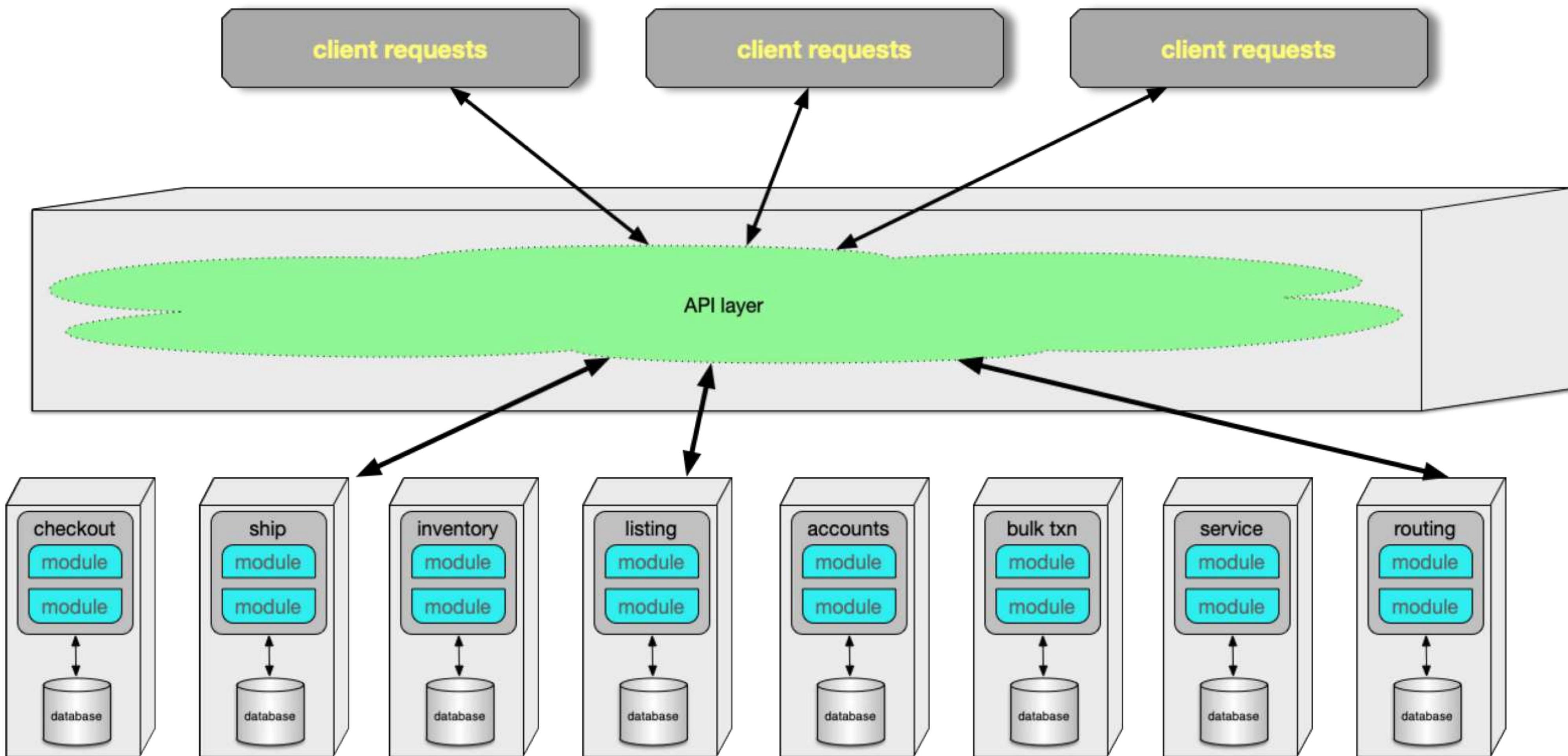


why quantum?

The quantum is where architectural characteristics live.



microservices

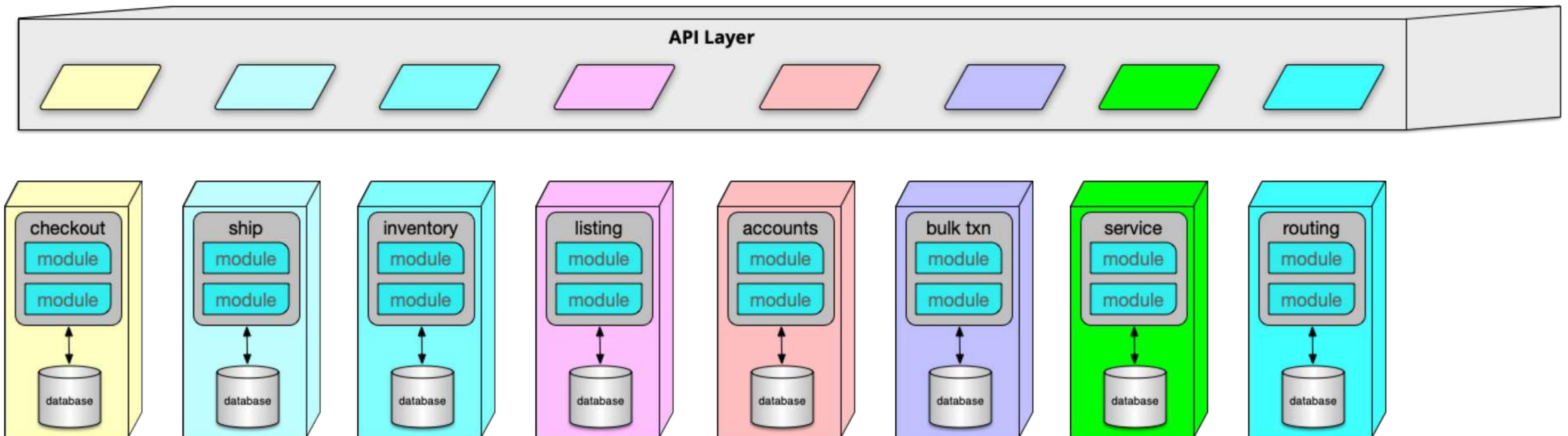


microservices => evolvable

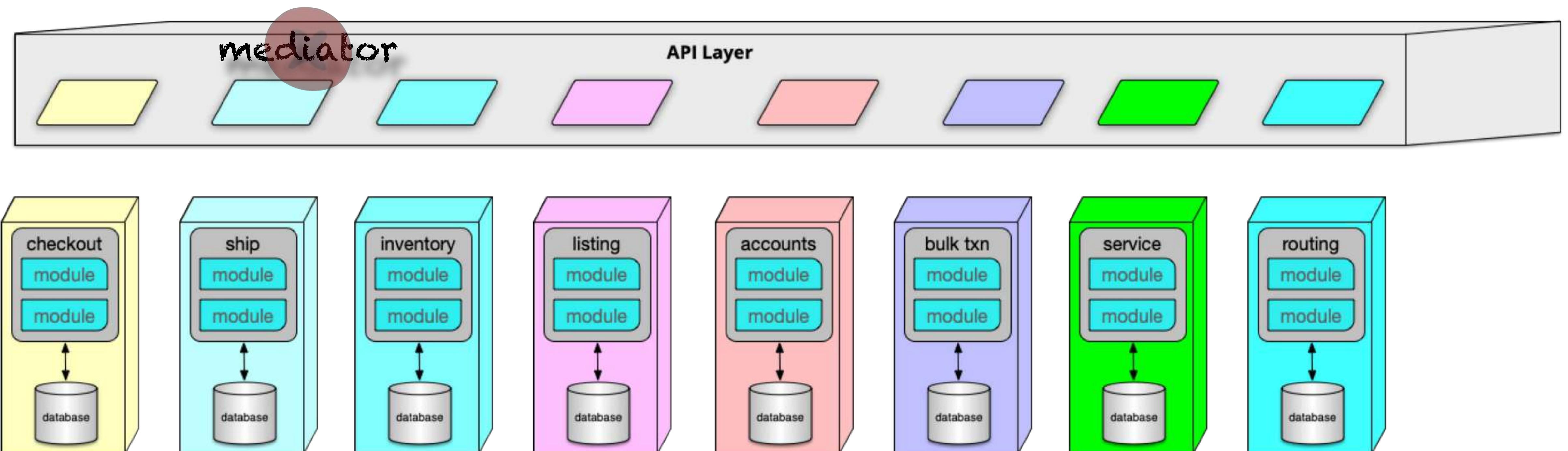
extremely loose coupling



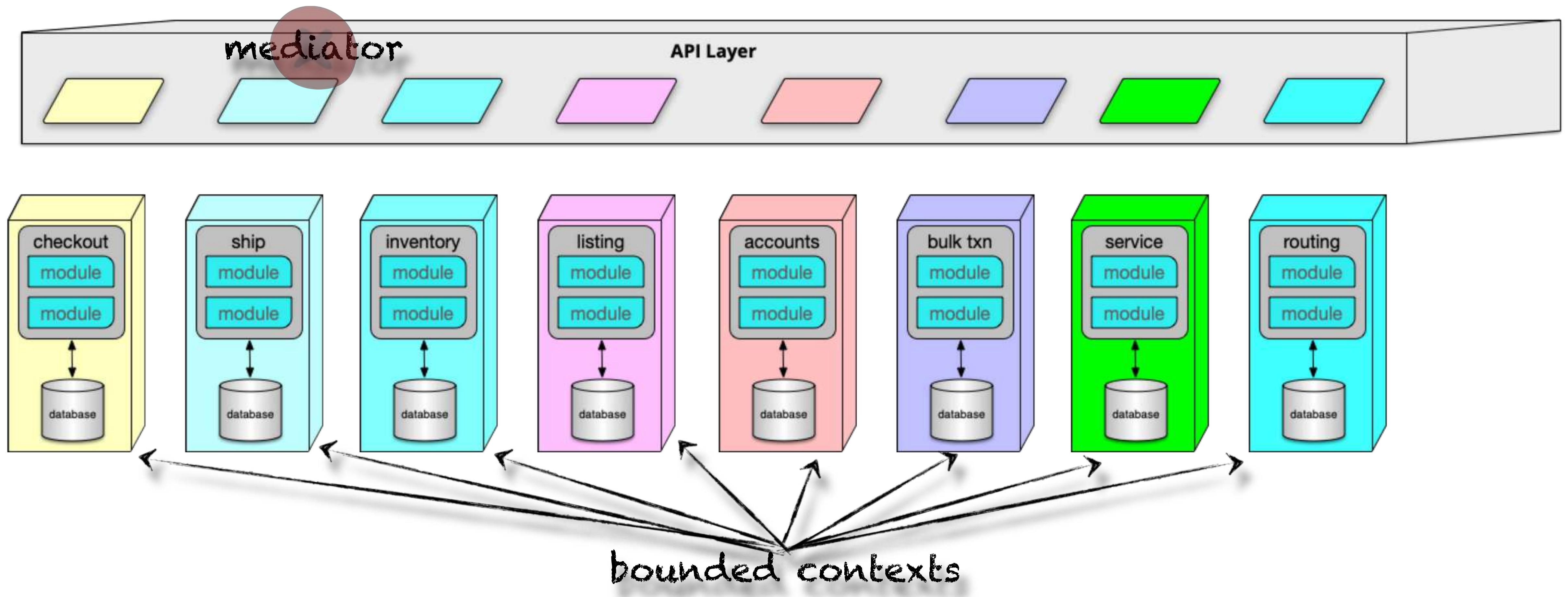
microservices



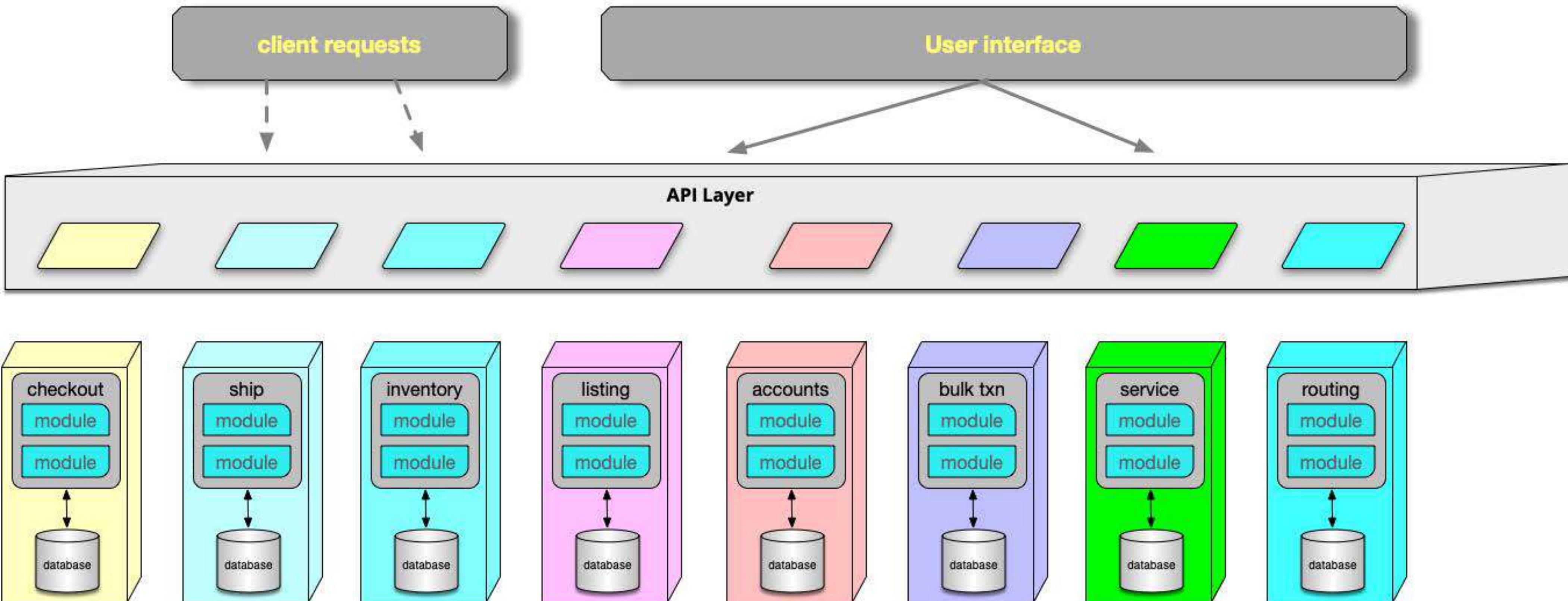
microservices



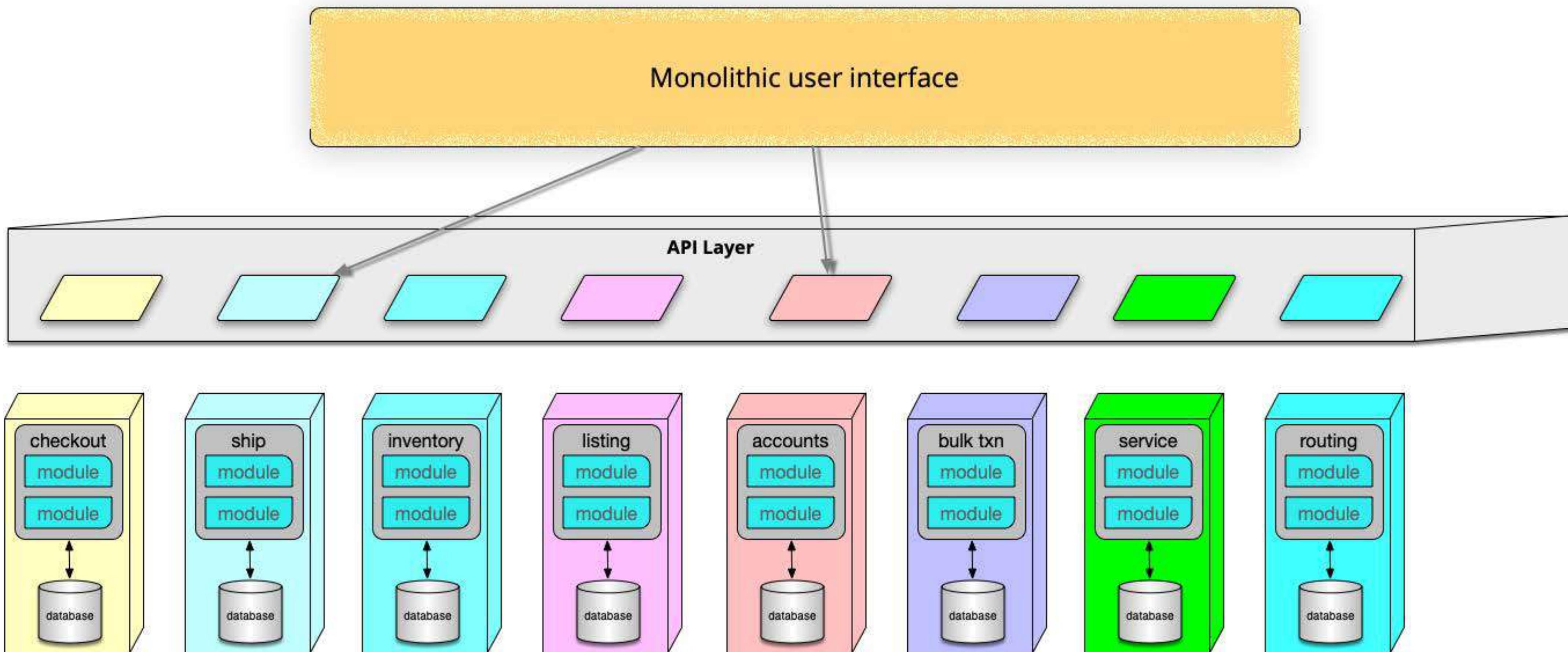
microservices



microservices

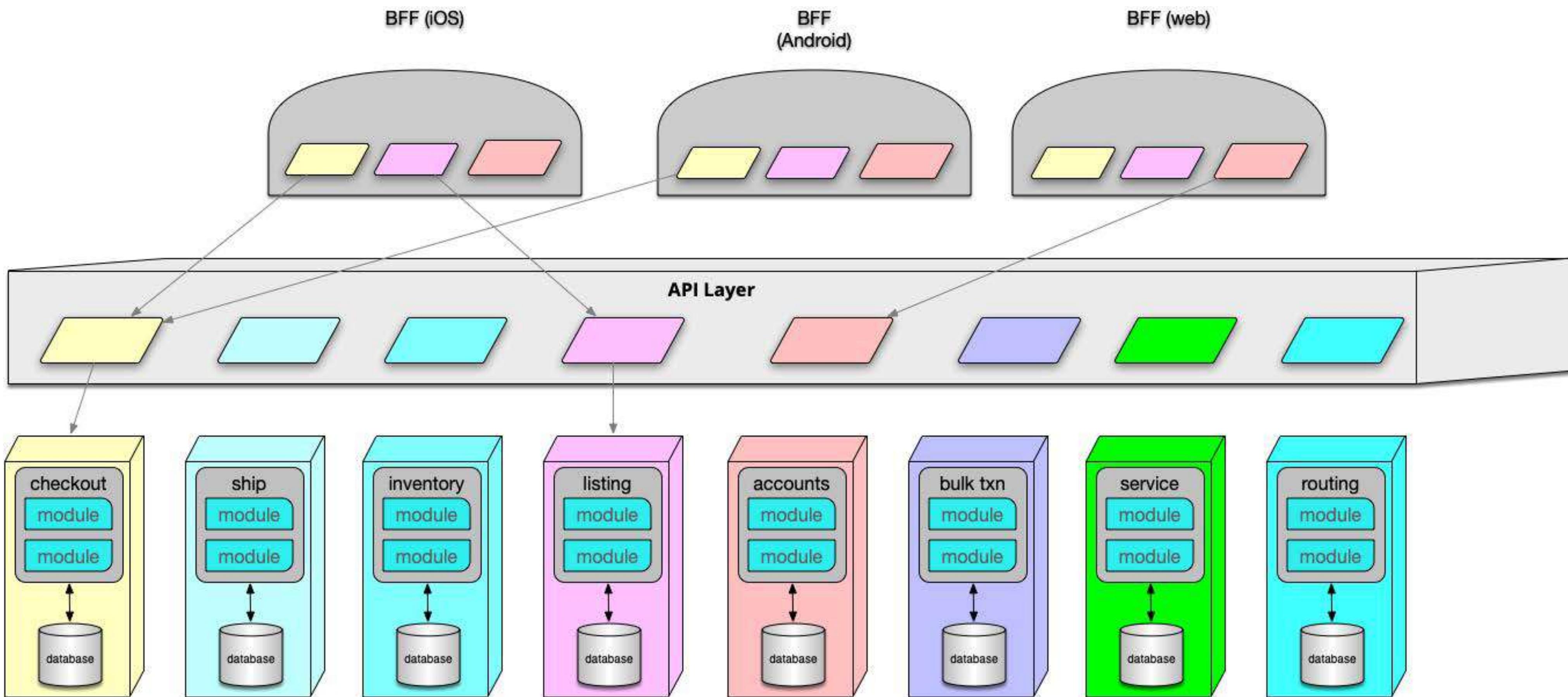


microservices

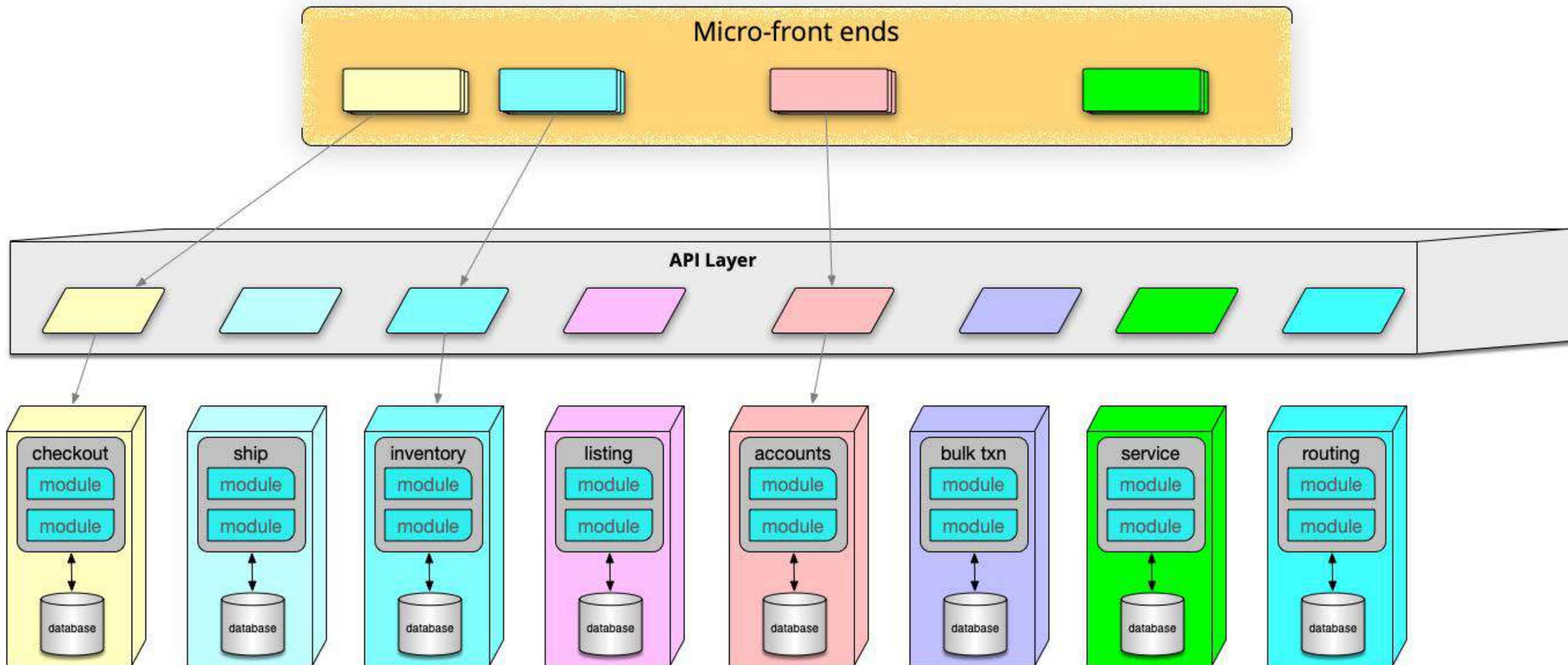


microservices: BFF

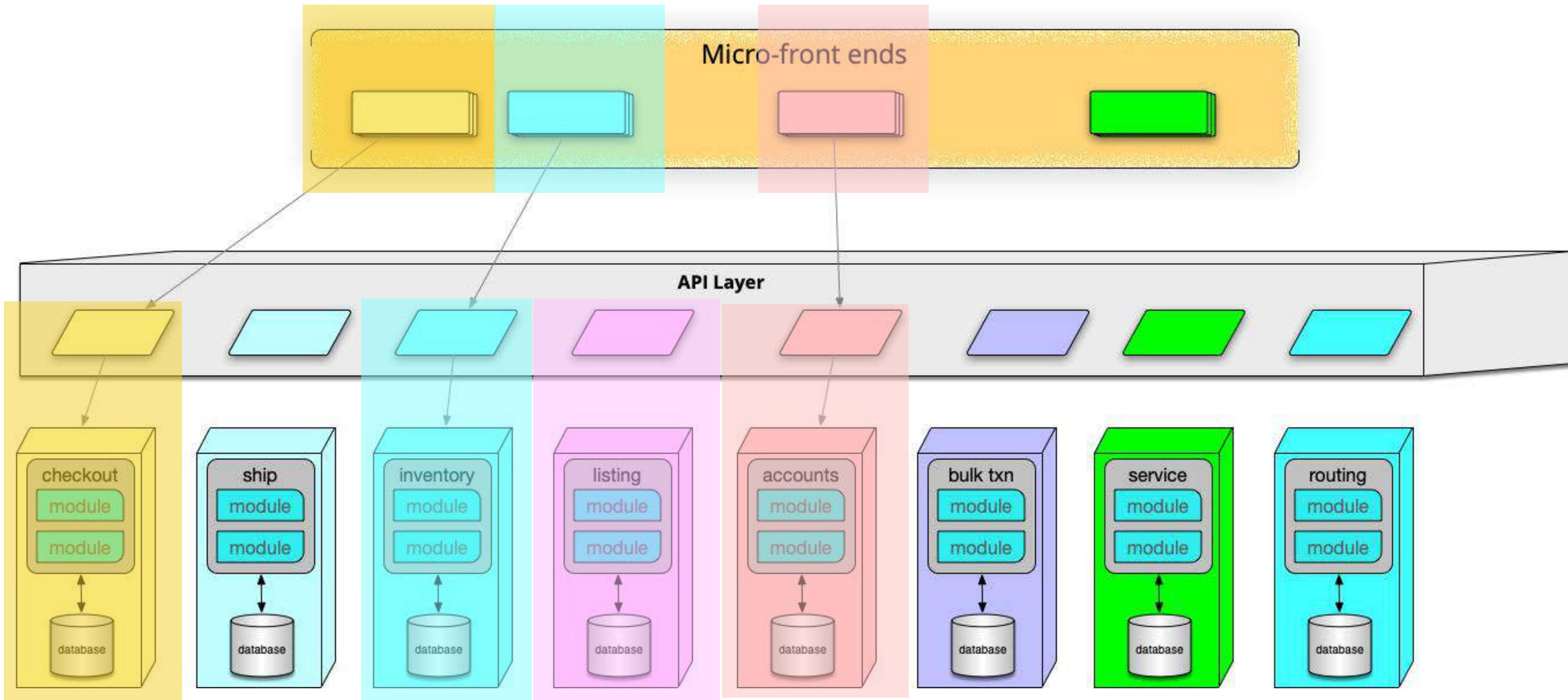
<https://samnewman.io/patterns/architectural/bff/>



microservices

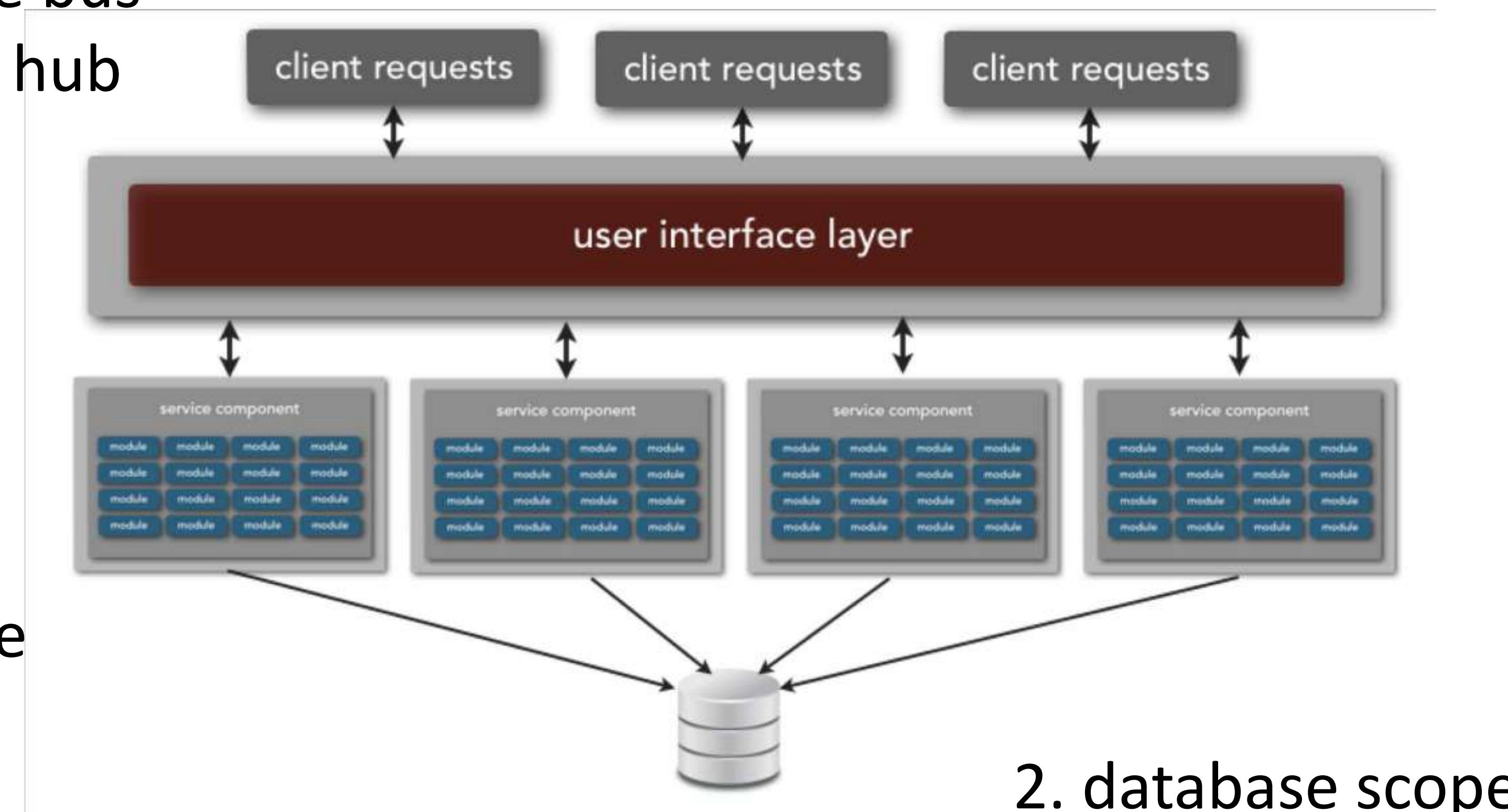


microservices quanta



service-based architectures

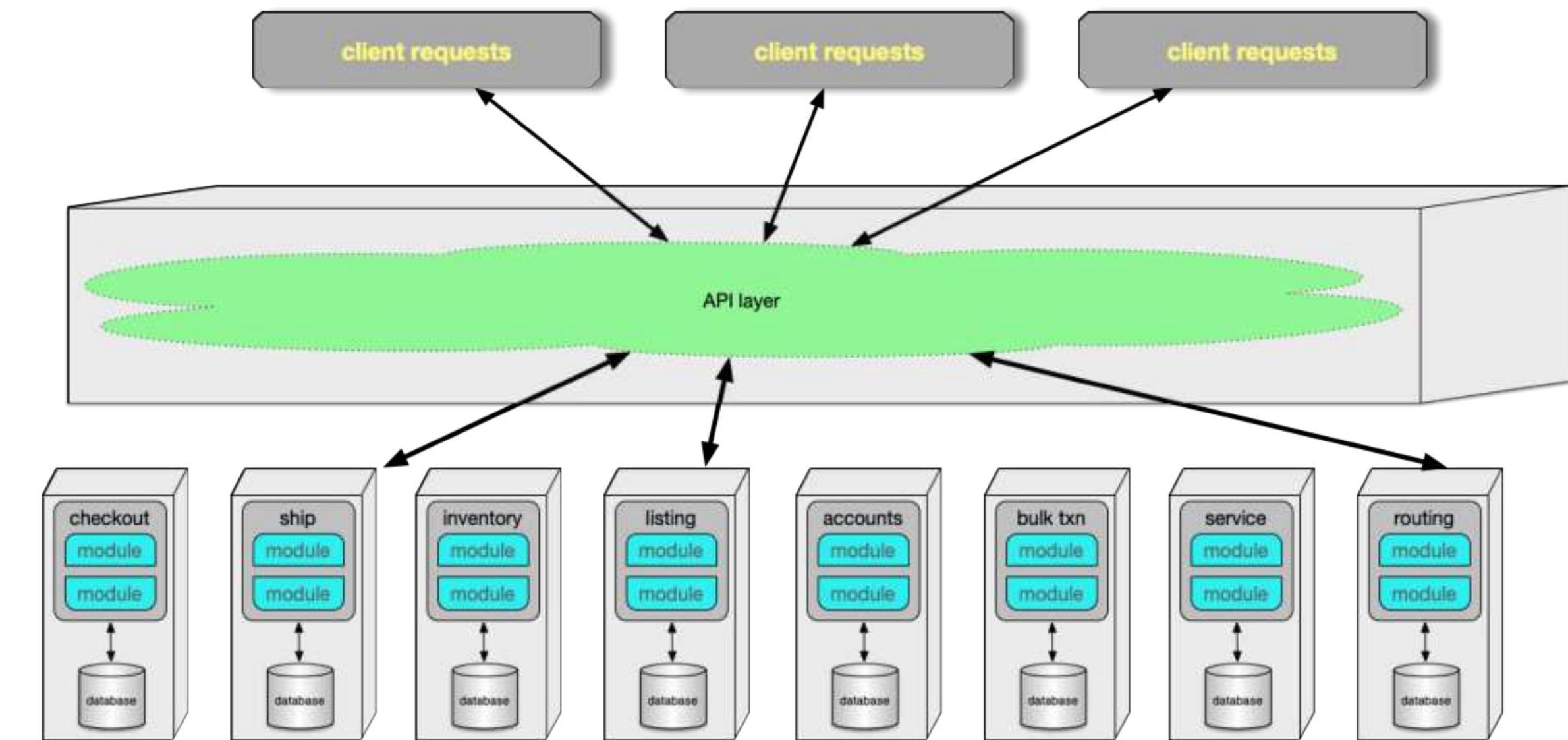
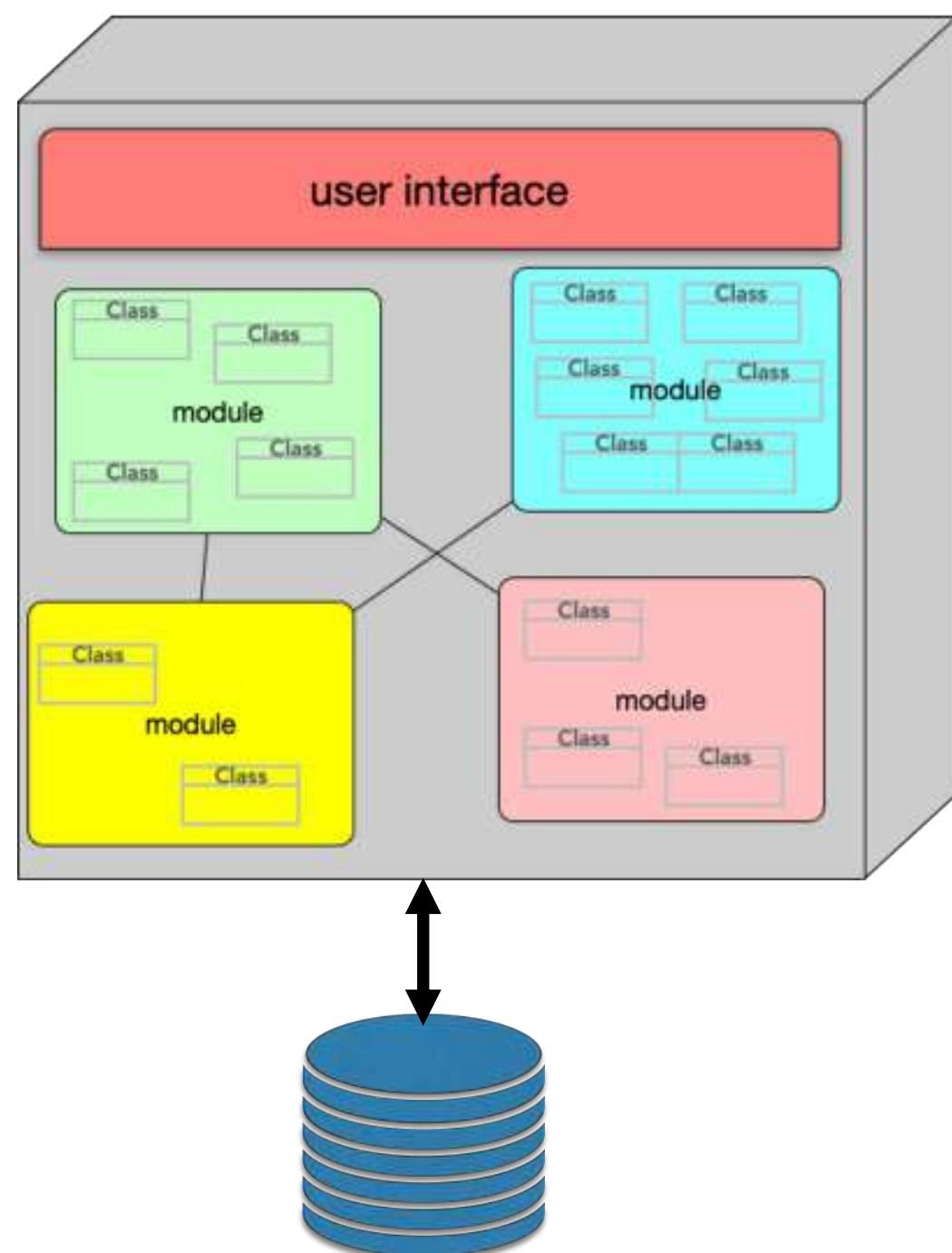
3. use of service bus as integration hub

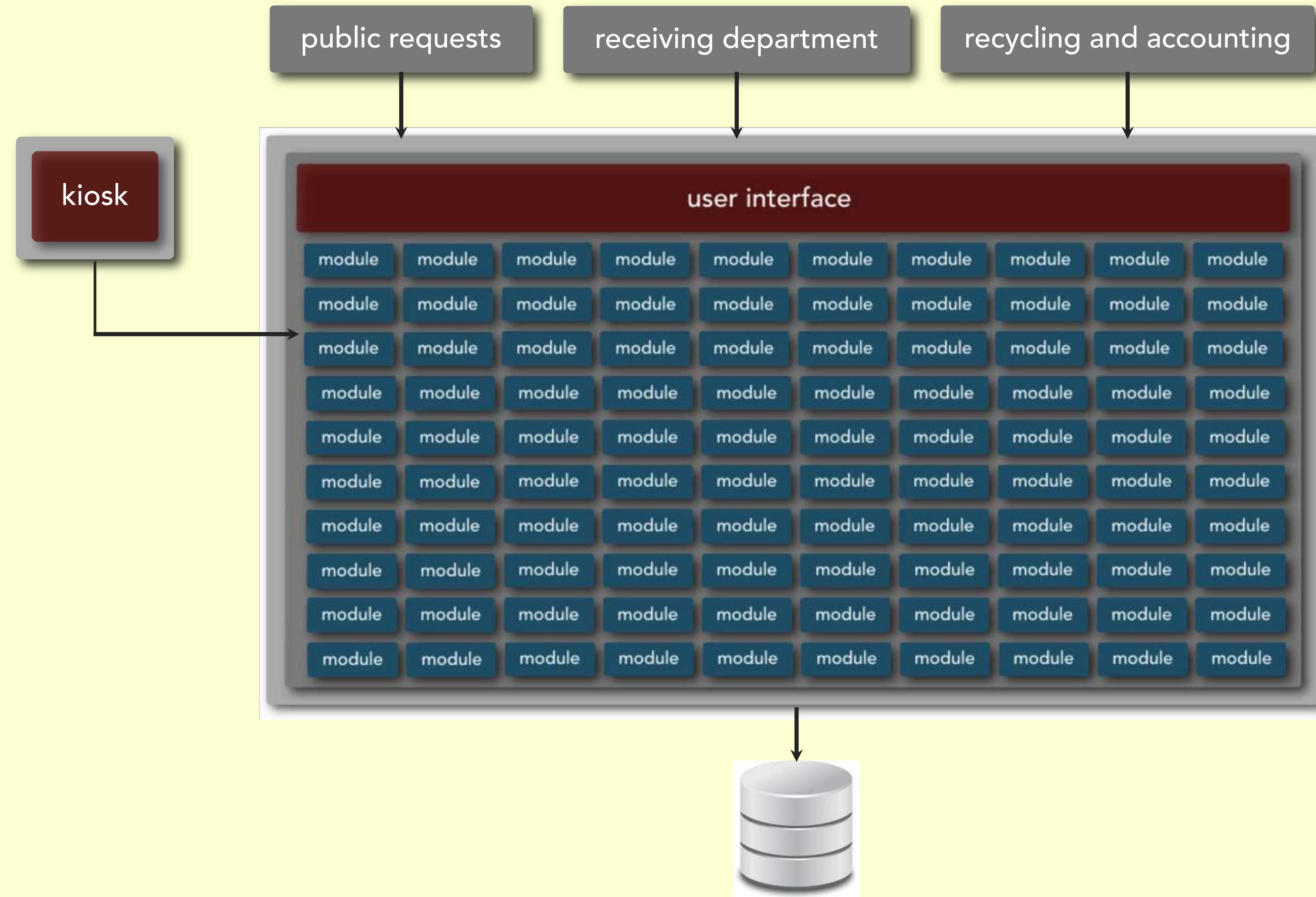


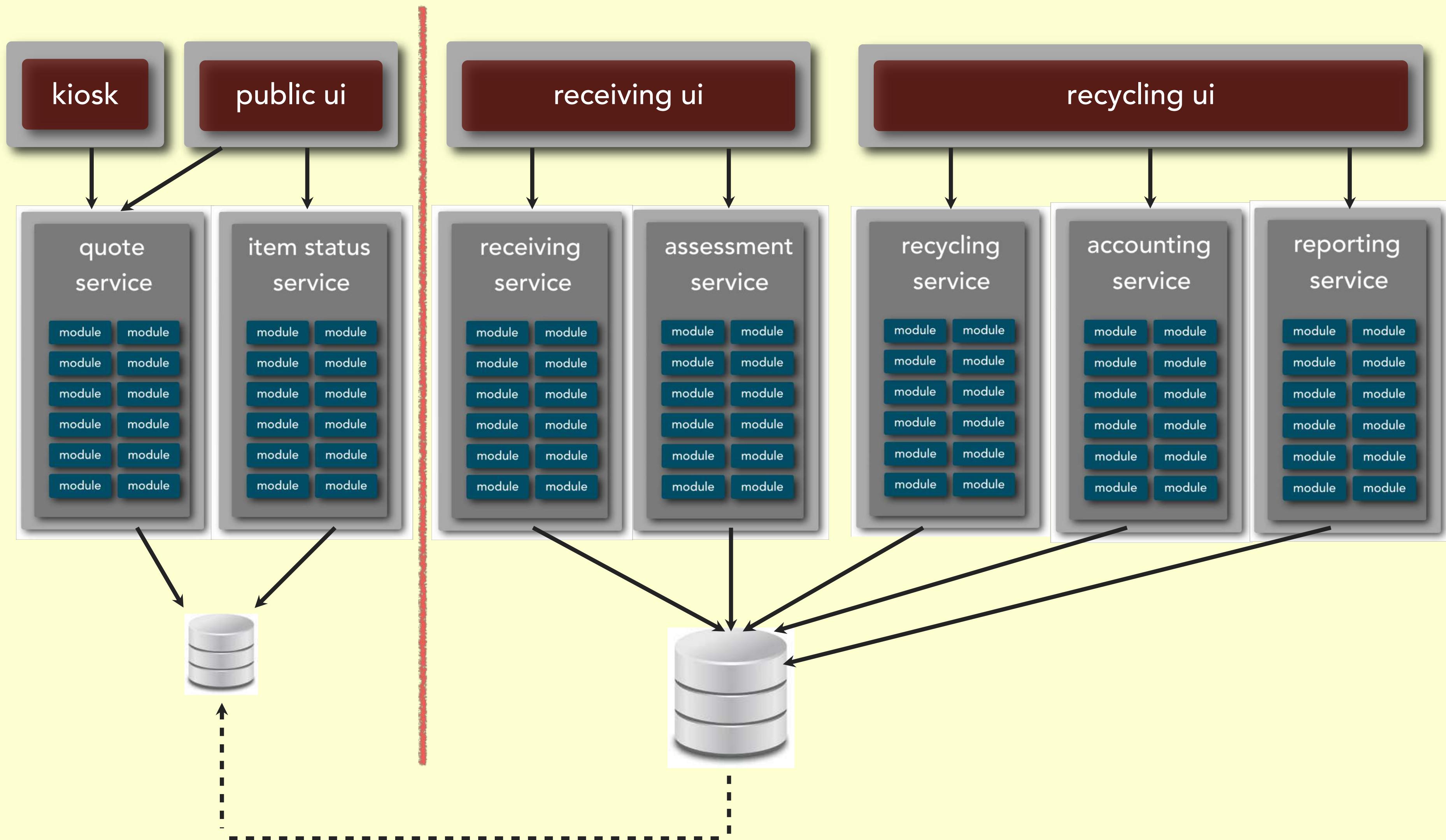
smaller quanta size

\triangleq

evolutionary architecture







Your Architectural Kata is...

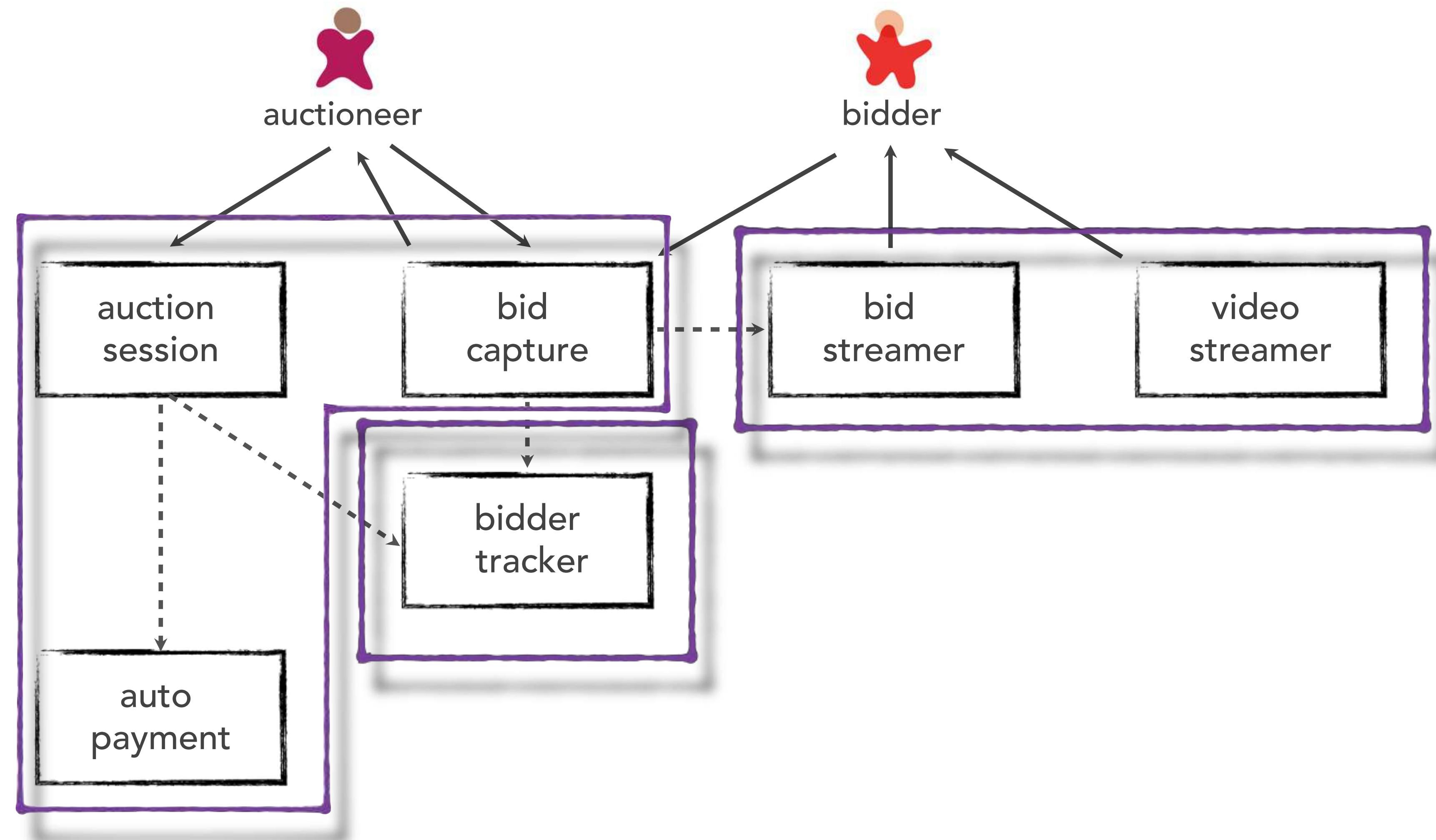
An auction company wants to take their auctions online to a nationwide scale--customers choose the auction to participate in, wait until the auction begins, then bid during the live auction as if they were there in the room, with the auctioneer.

- **Users:** scale up to hundreds of participants (per auction), potentially up to thousands of participants, and as many simultaneous auctions as possible
- **Requirements:**
 - bidders can see a live video stream of the auction and see all bids as they occur
 - auctions must be as real-time as possible
 - both online and live bids must be received in the order in which they are placed
 - bidders register with credit card; system automatically charges card if bidder wins
 - participants must be tracked via a reputation index
- **Additional Context:**
 - auction company is expanding aggressively by merging with smaller competitors
 - if nationwide auction is a success, replicate the model overseas
 - budget is not constrained--this is a strategic direction
 - company just exited a lawsuit where they settled a suit alleging fraud

Your Architectural Kata is...

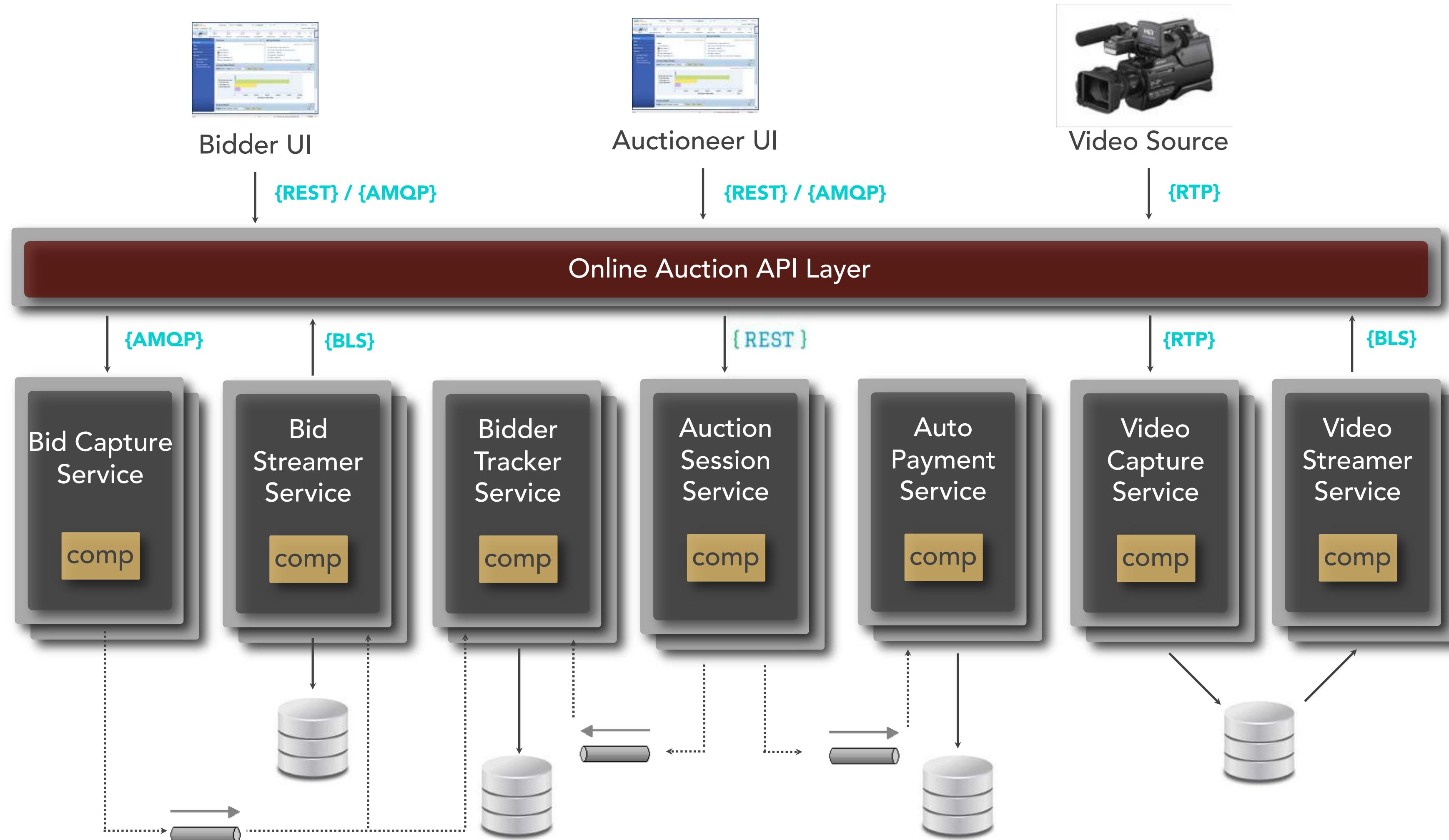
Going Going Gone!

quanta



Your Architectural Kata is...

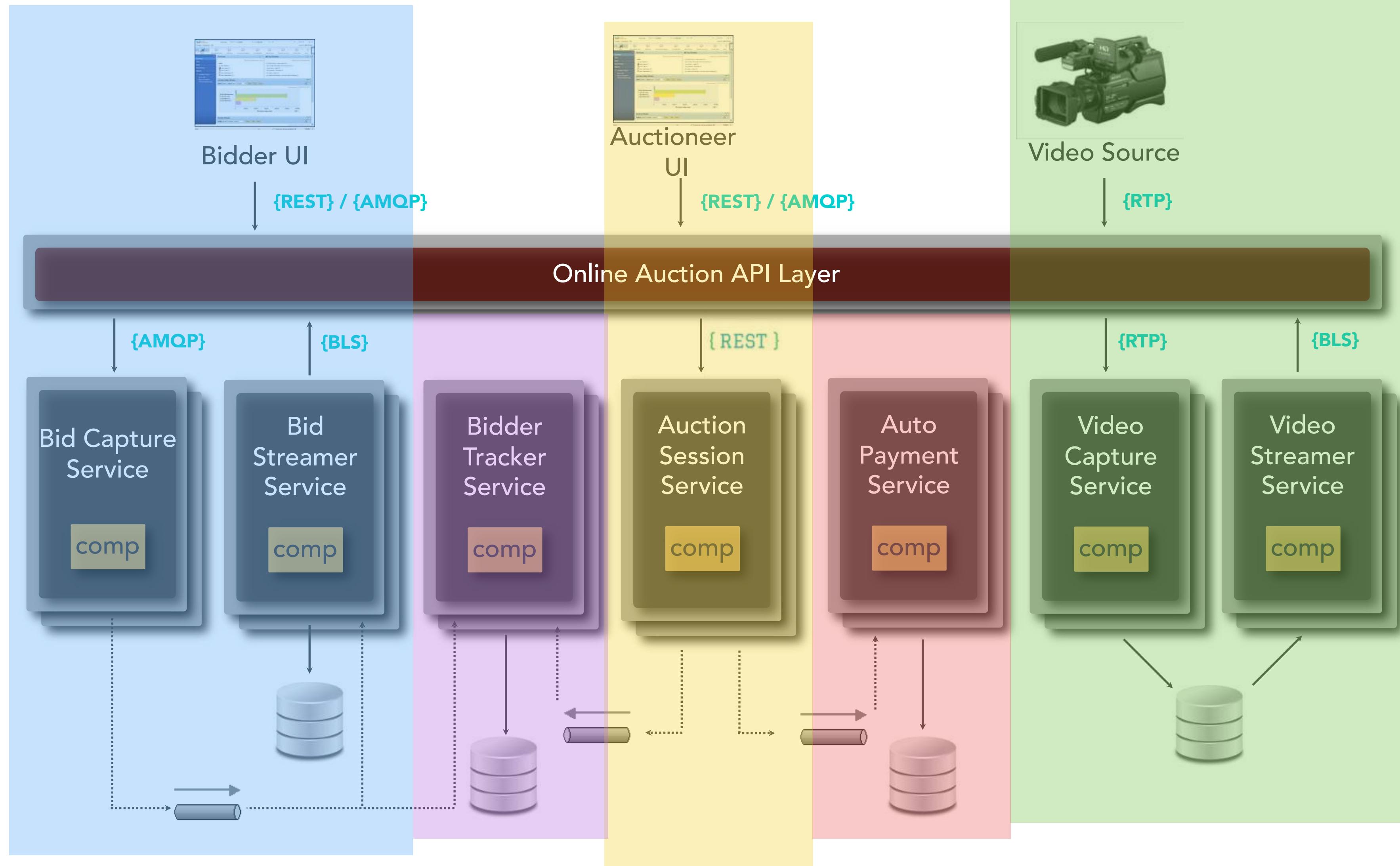
Going Going Gone!



Your Architectural Kata is...

quanta

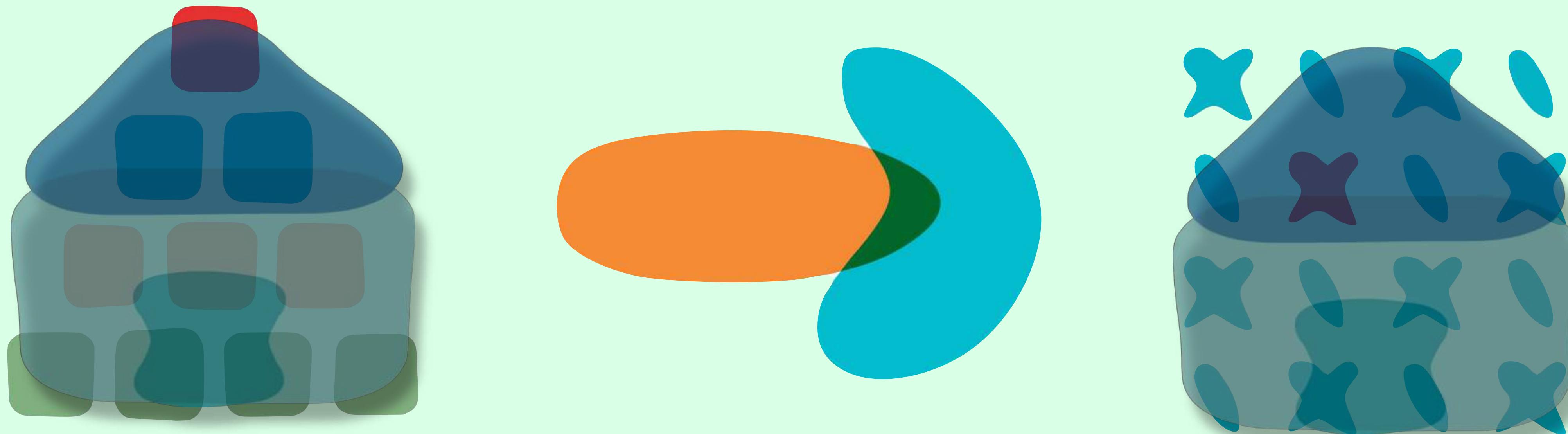
Going Going Gone!



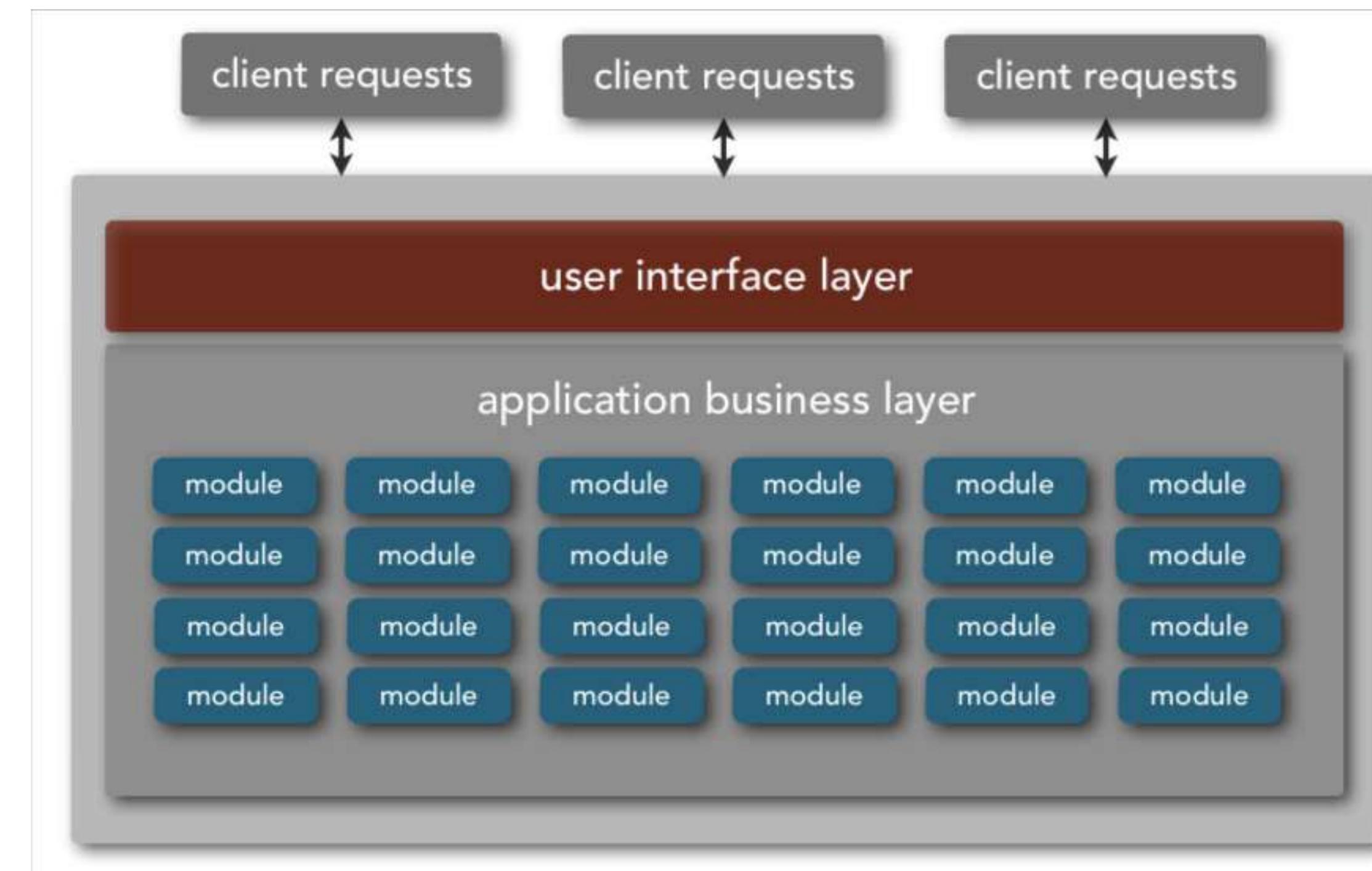
Question:

Outside defining a new architecture, when would the architecture quantum measure come in handy?

Migrating Architectures

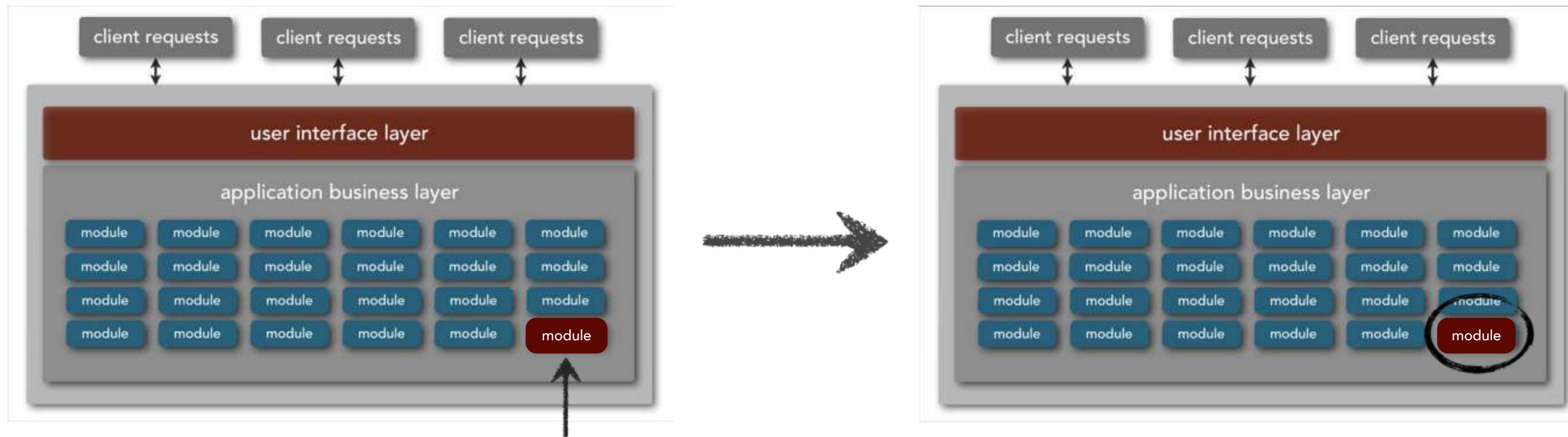


monolithic application issues



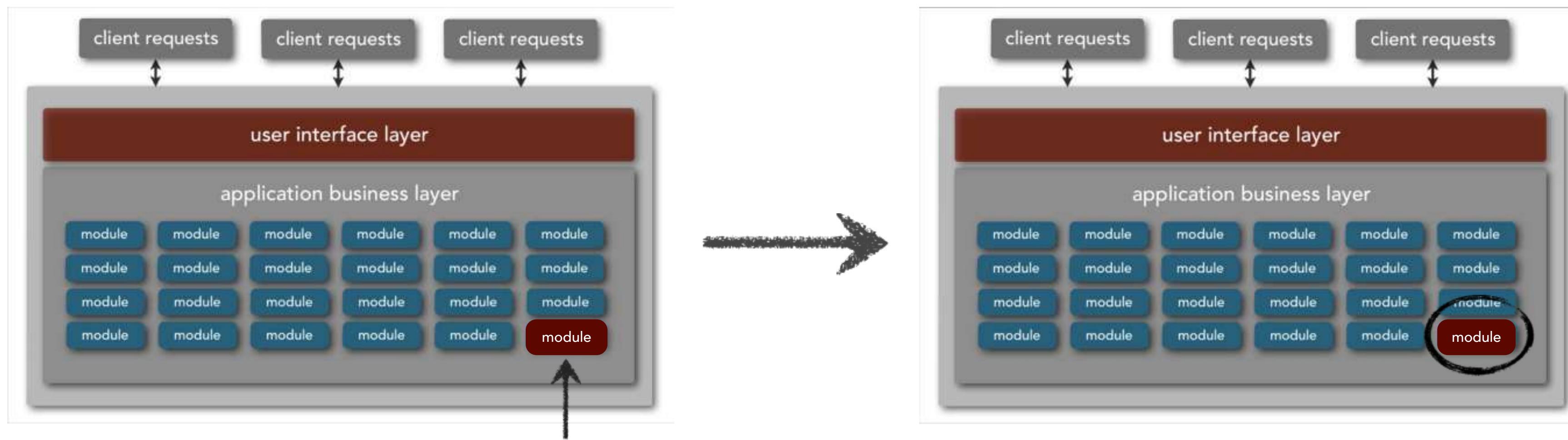
monolithic application issues

deployment



monolithic application issues

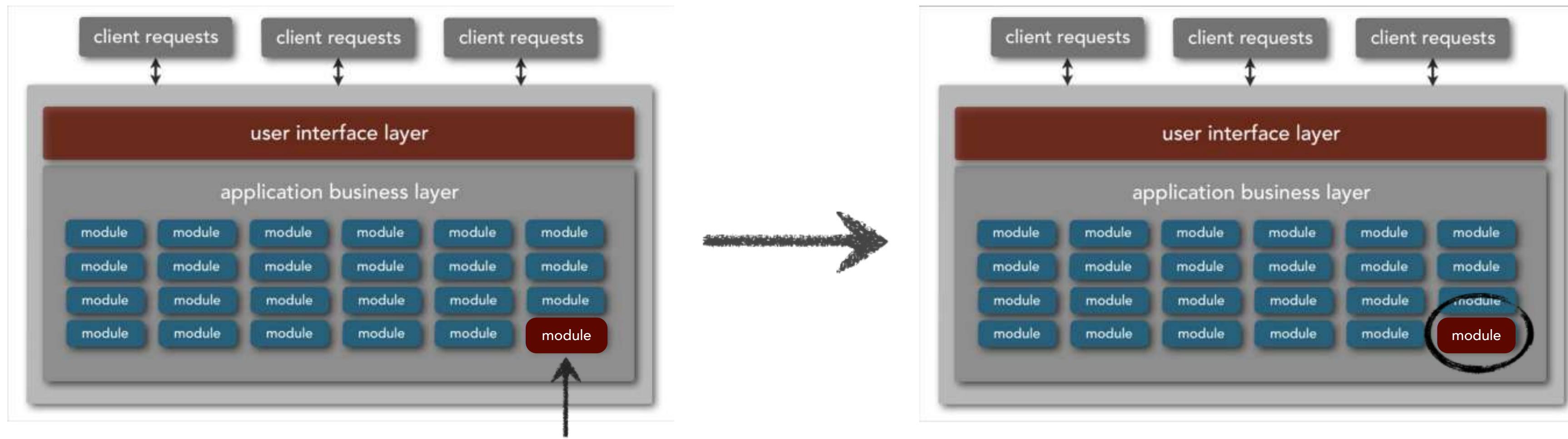
deployment



entire application must be deployed for a small change

monolithic application issues

deployment

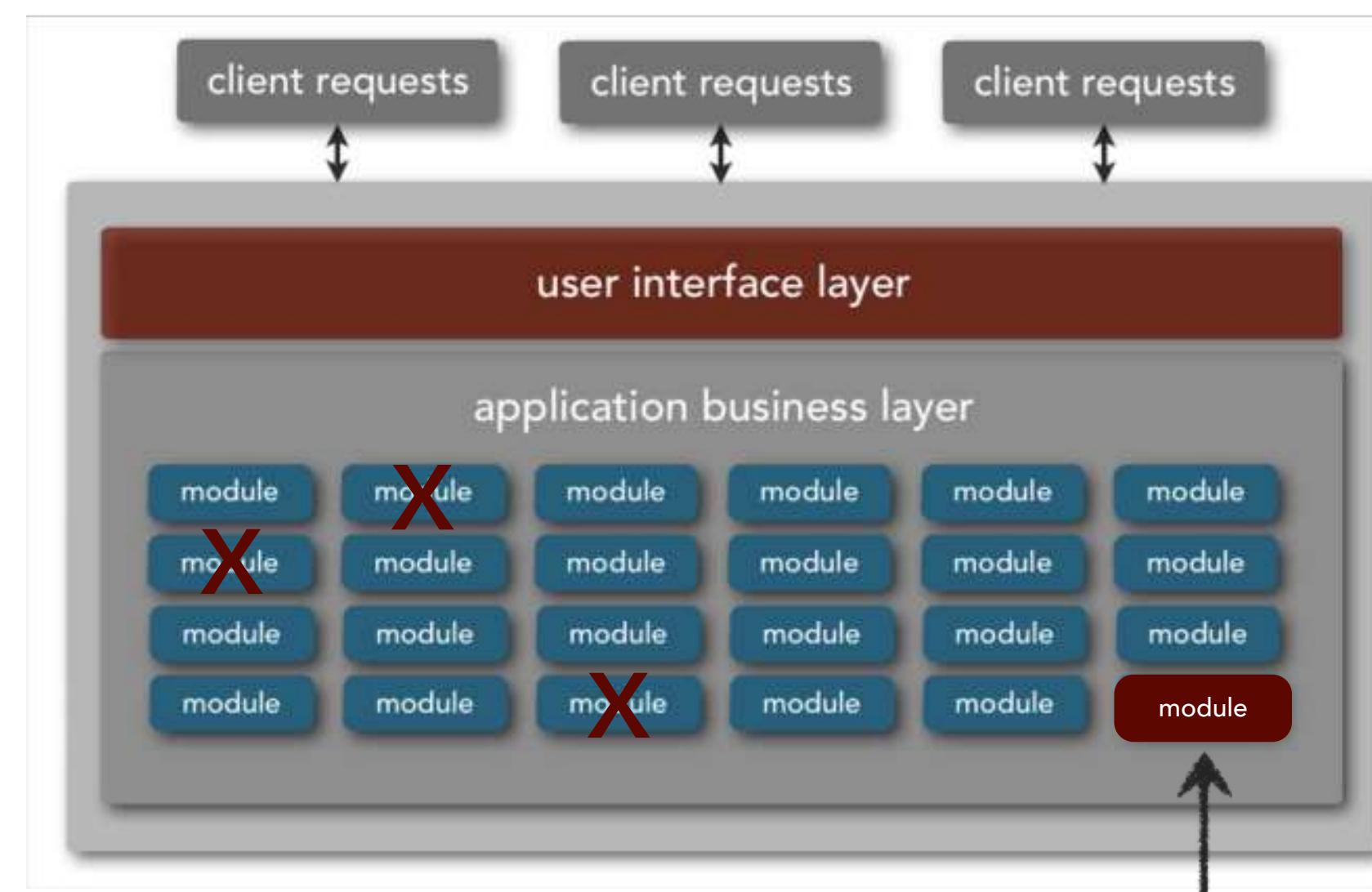


entire application must be deployed for a small change

scheduling and coordination challenges can impact deployment

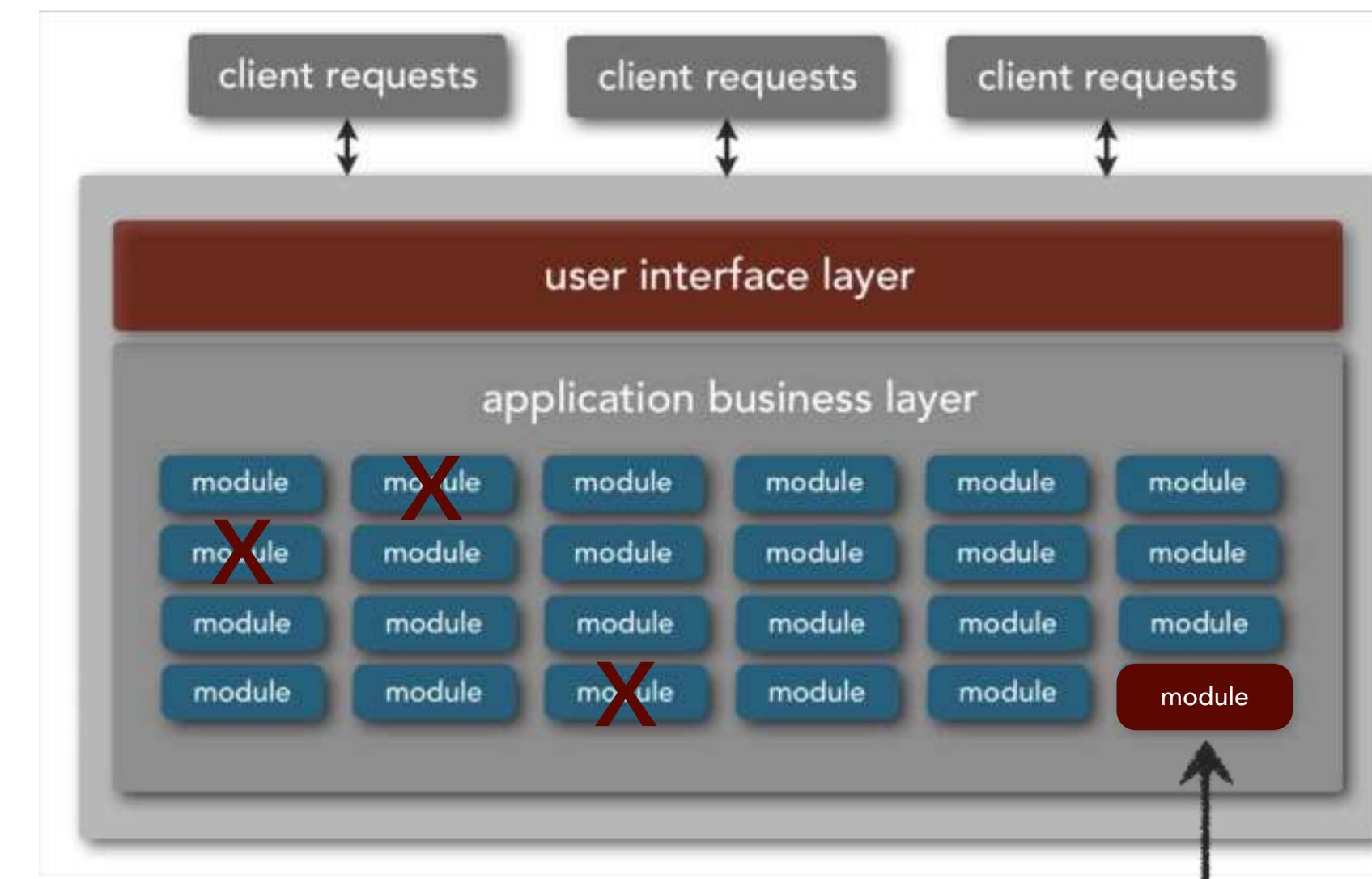
monolithic application issues

reliability and robustness



monolithic application issues

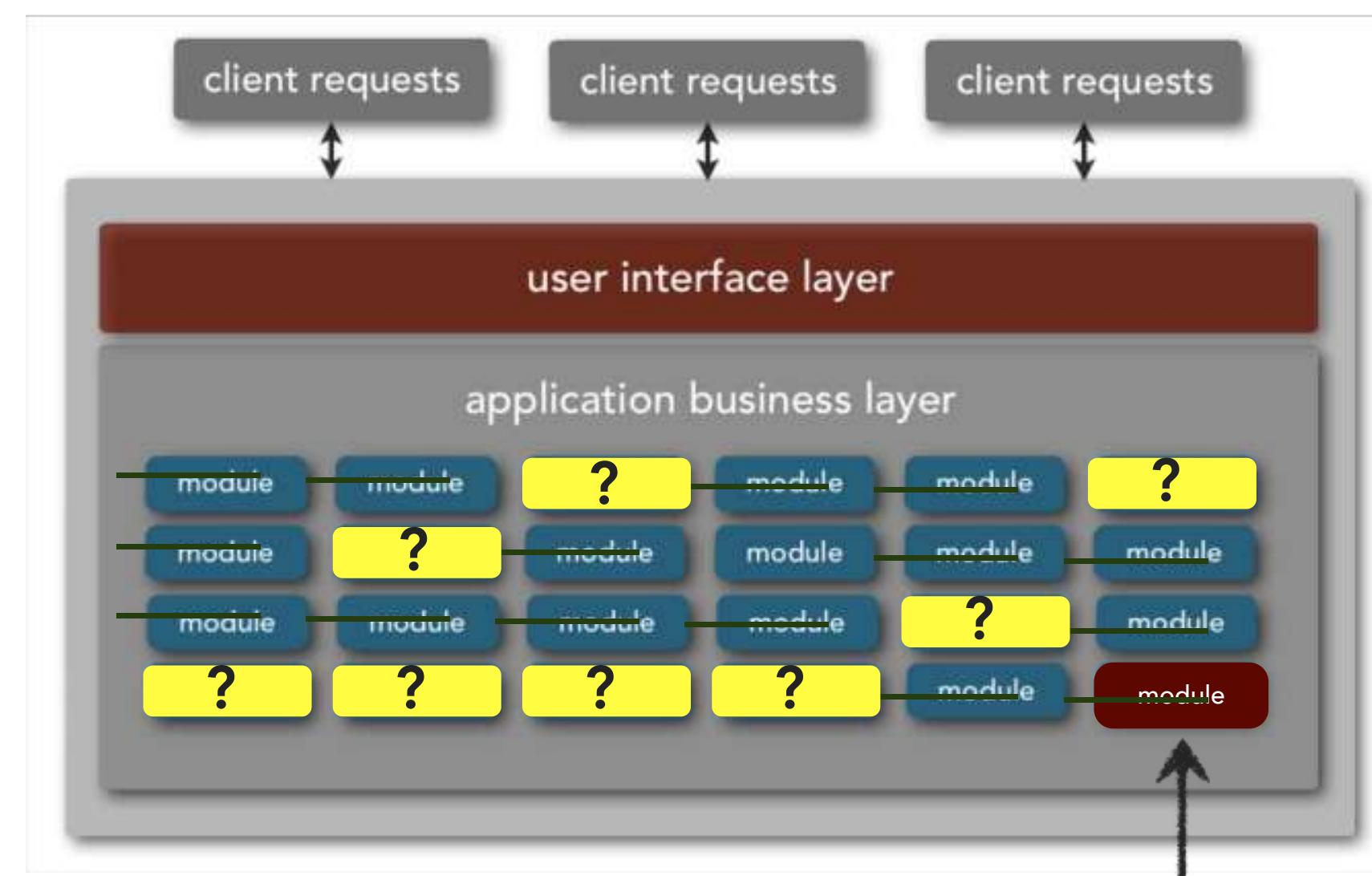
reliability and robustness



due to tight coupling, changes in one area of the application adversely affects other areas

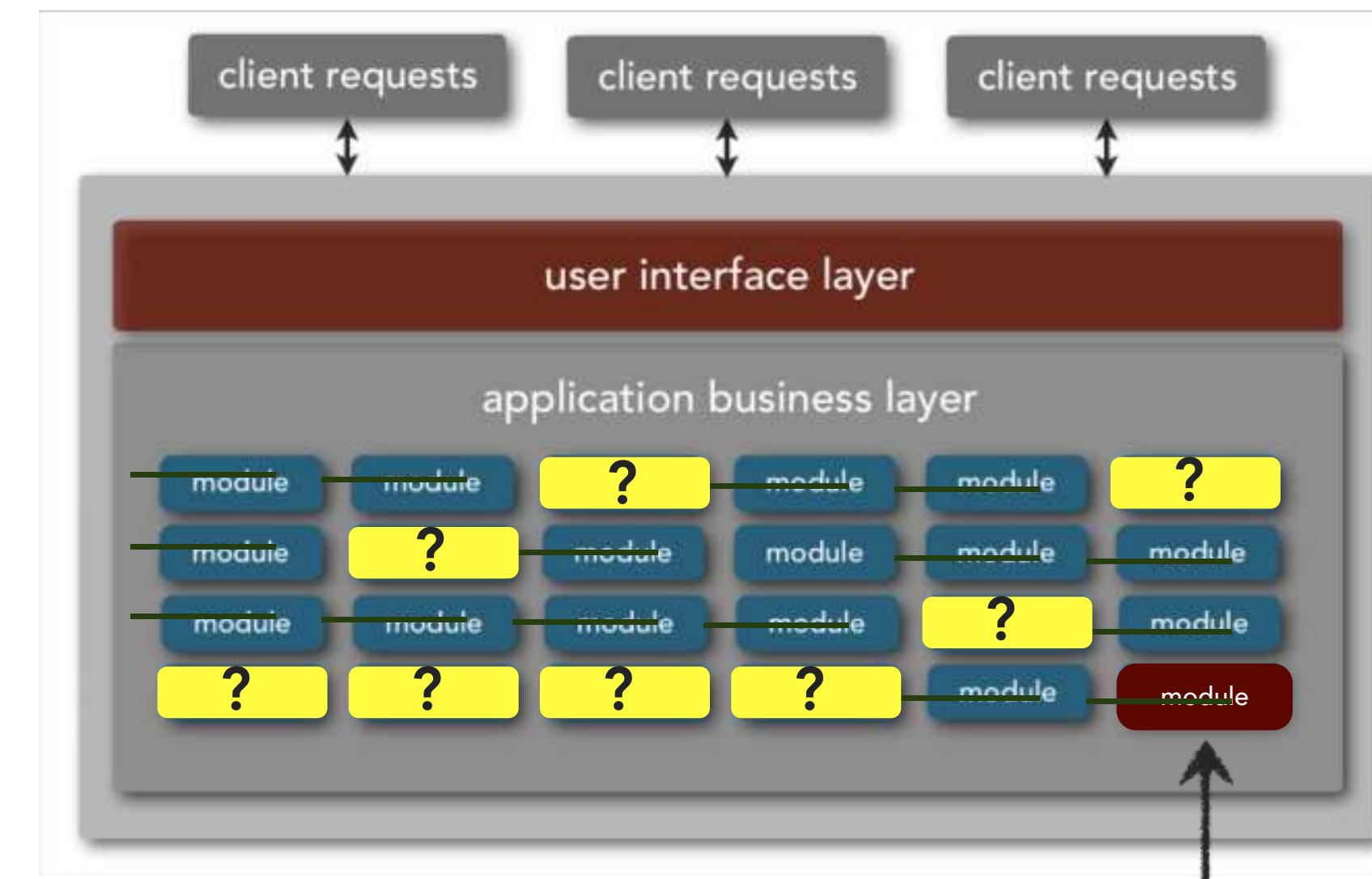
monolithic application issues

testability



monolithic application issues

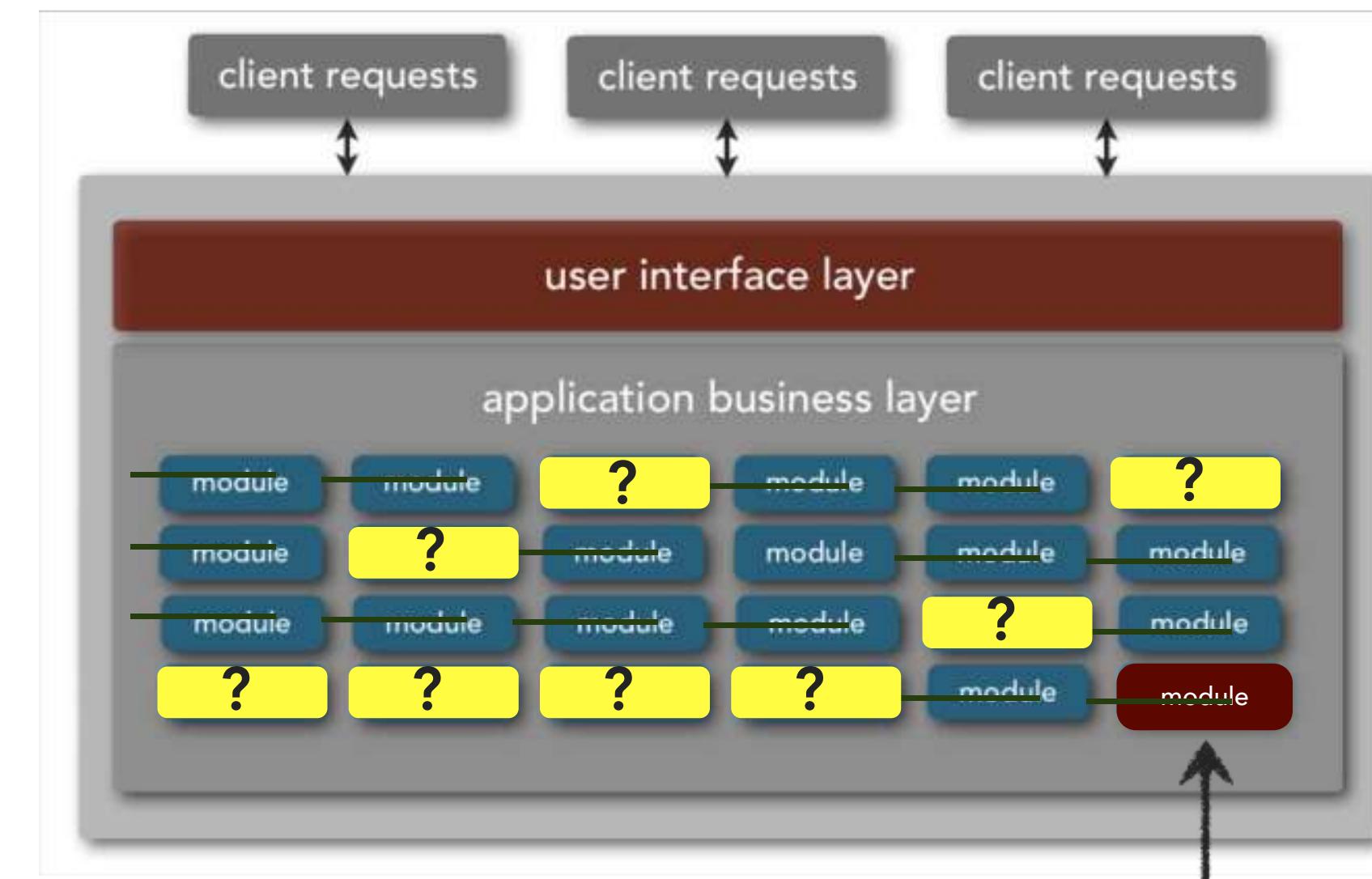
testability



poor test coverage for entire application

monolithic application issues

testability

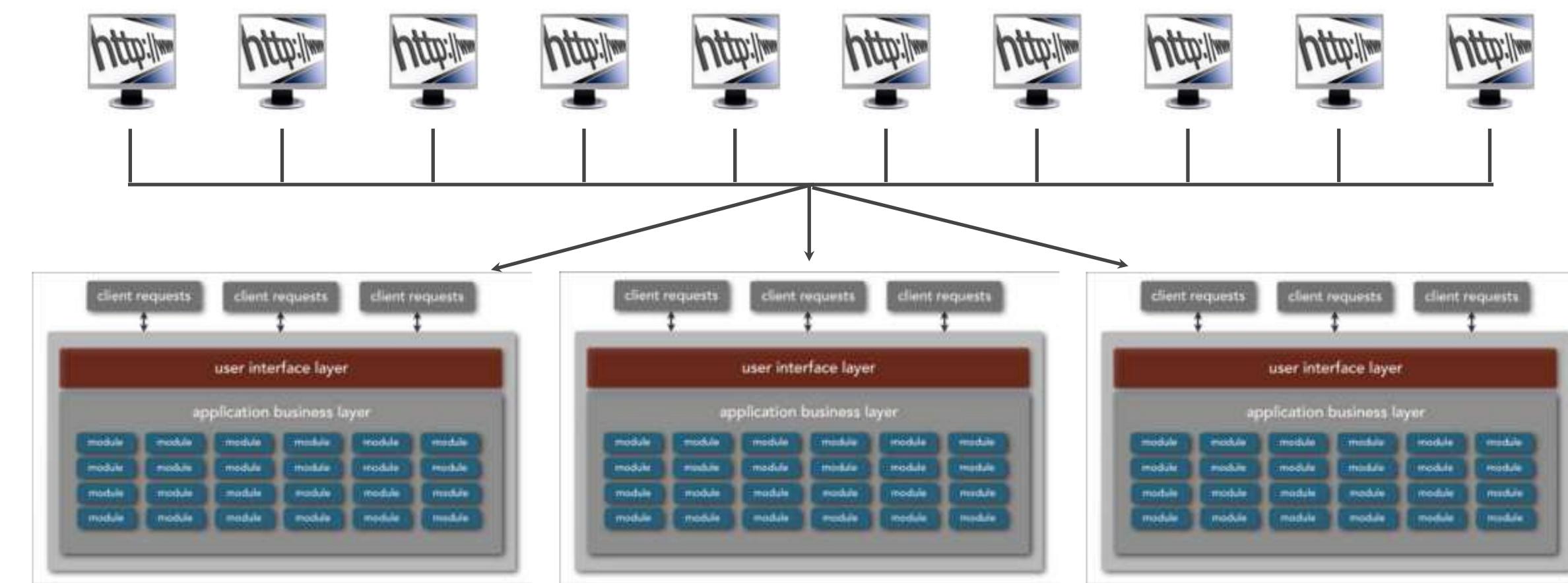


poor test coverage for entire application

low confidence level that nothing else is impacted by the change

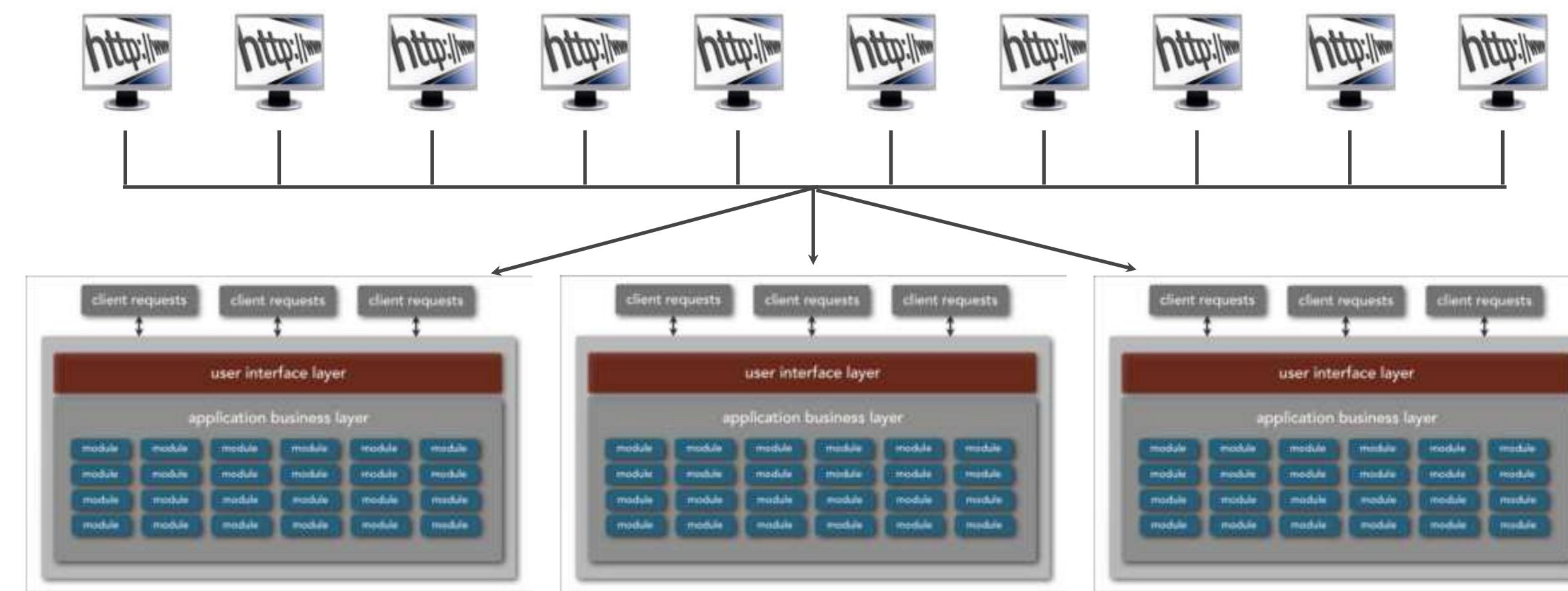
monolithic application issues

scalability



monolithic application issues

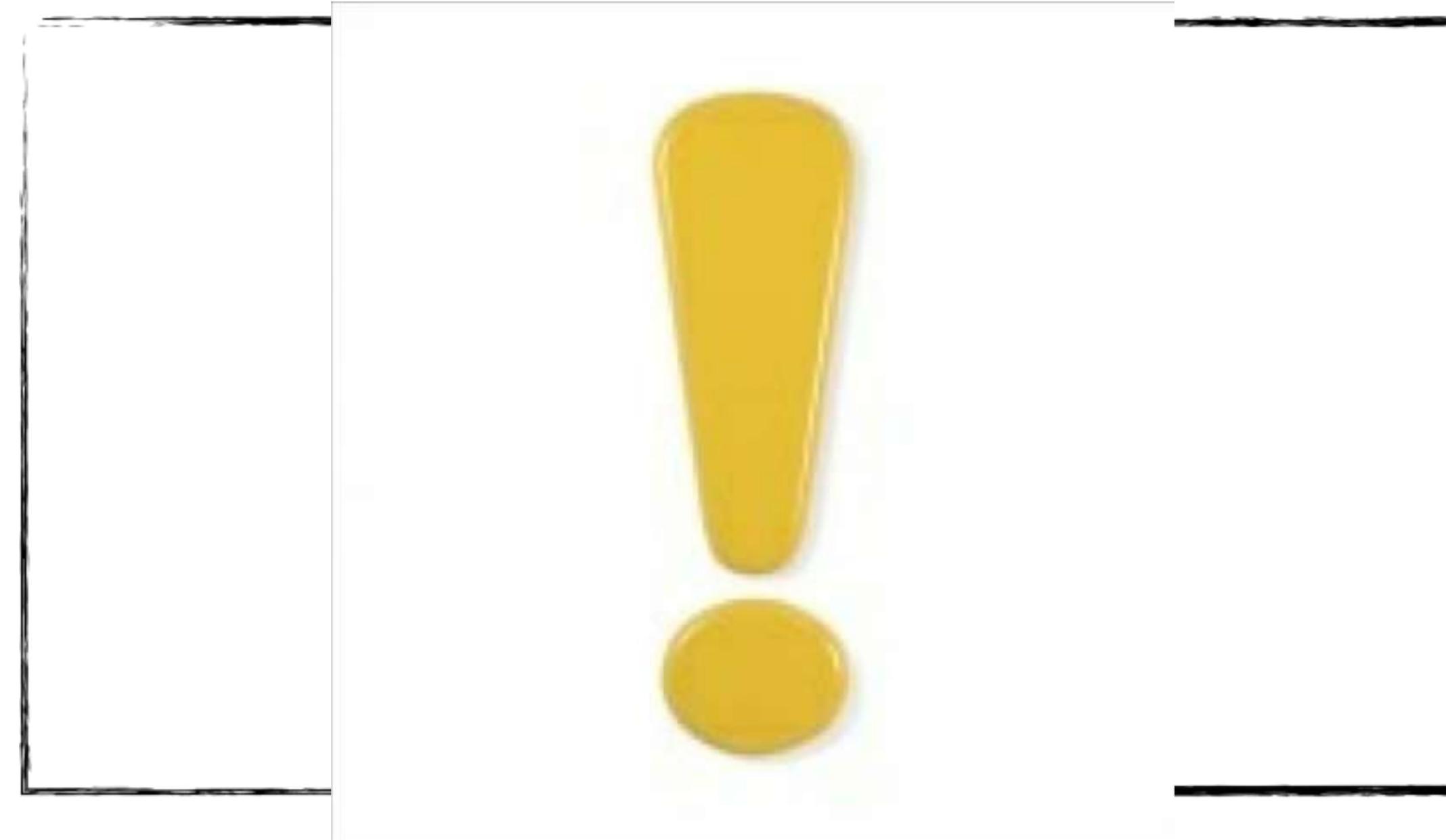
scalability



scaling monolithic applications requires you to scale the entire application, which can be both difficult and costly

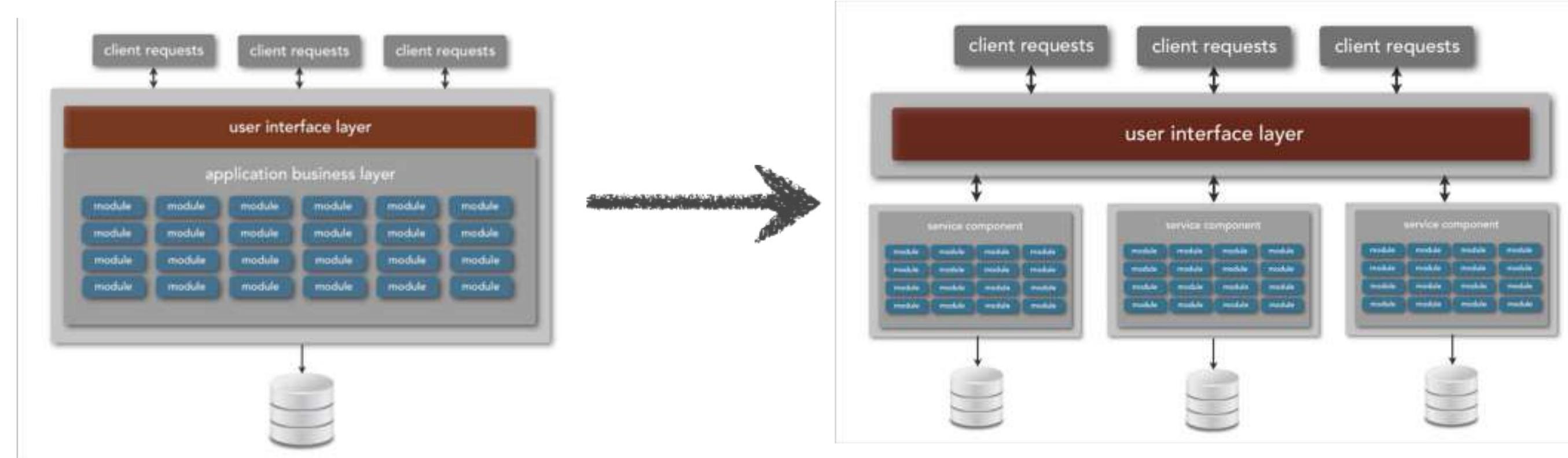
monolithic application issues

application size



monolithic applications can grow to quickly and consume all available resources

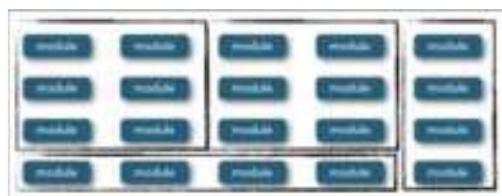
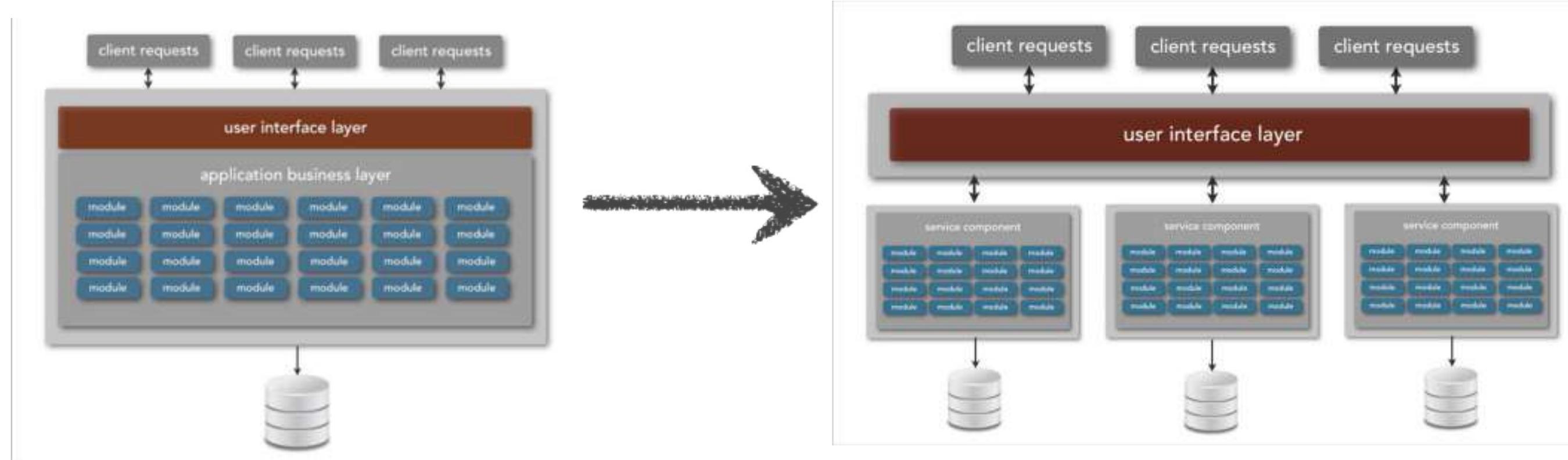
migration challenges



share everything
architecture

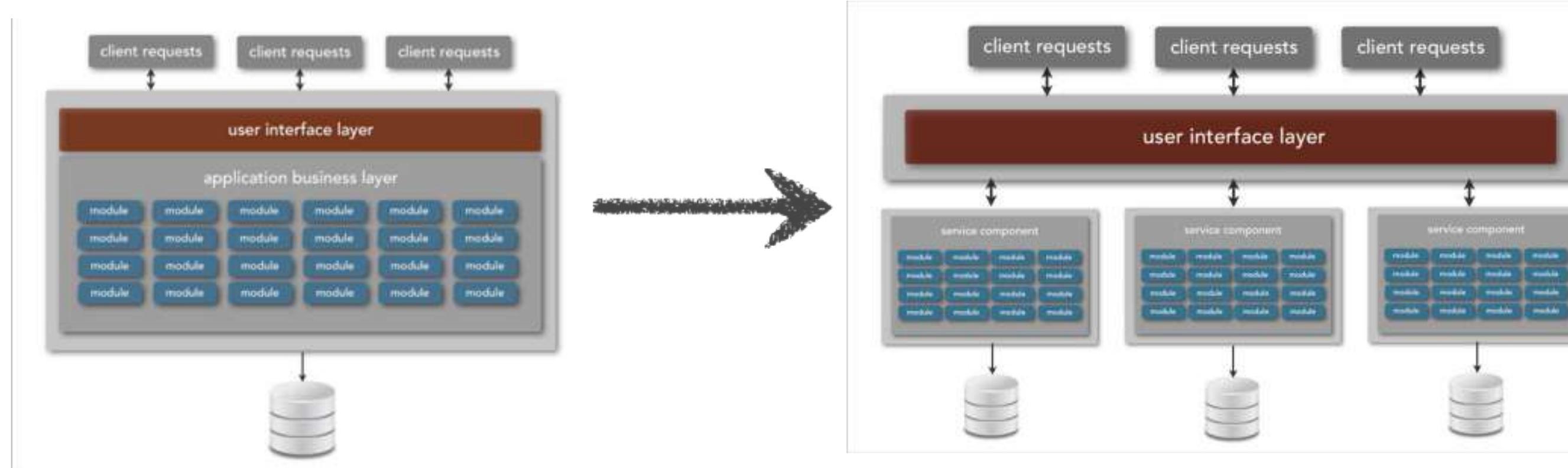
share as little as
possible architecture

migration challenges

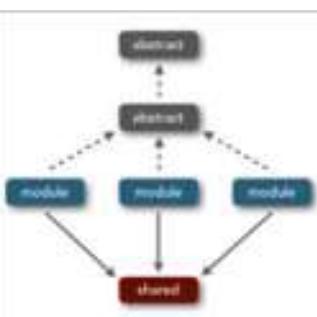


service component granularity and transactional boundaries

migration challenges

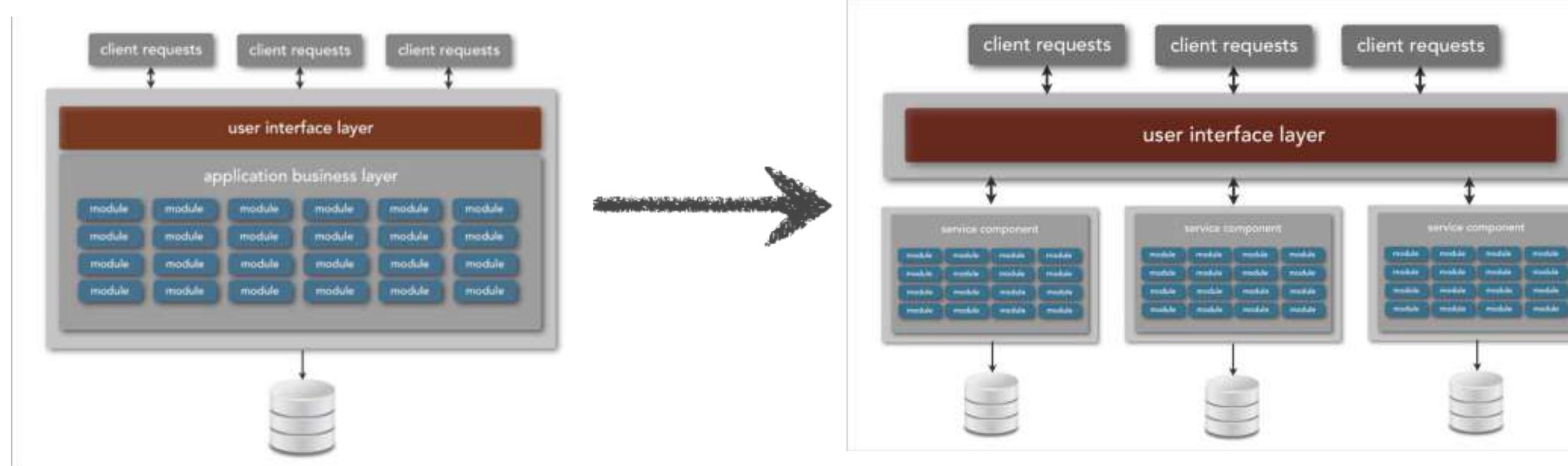


service component granularity and transactional boundaries

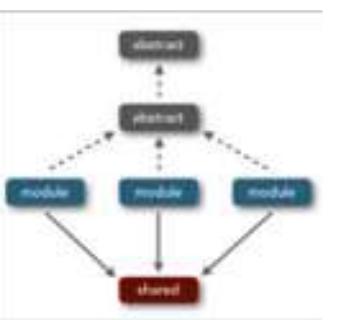


shared services, modules, and object hierarchies

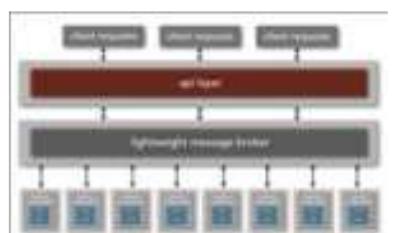
migration challenges



service component granularity and transactional boundaries

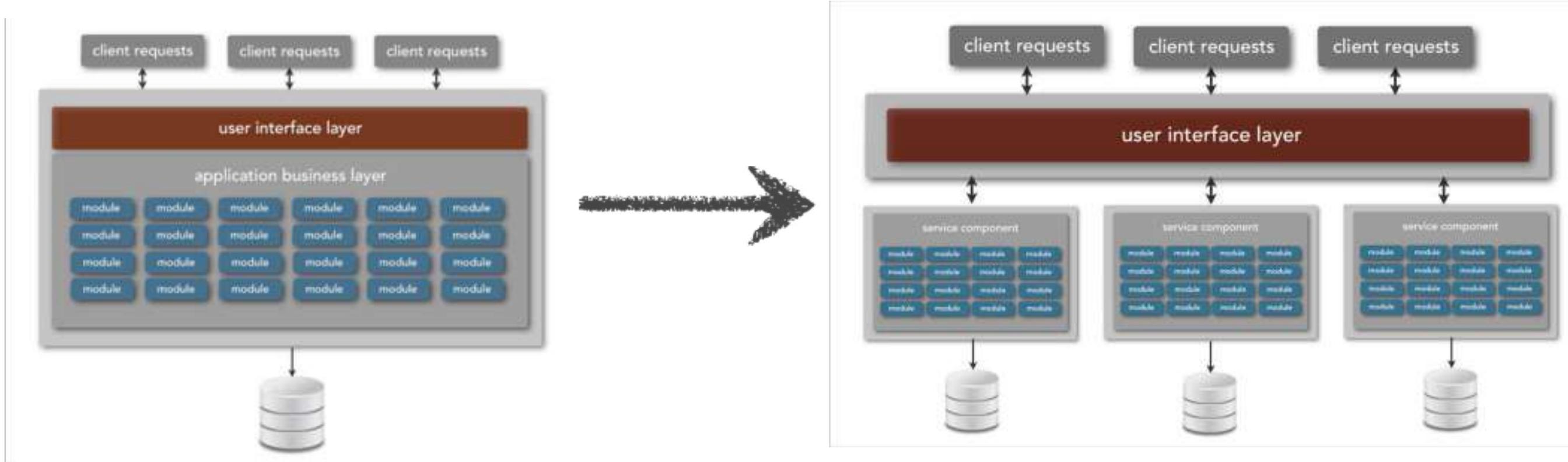


shared services, modules, and object hierarchies

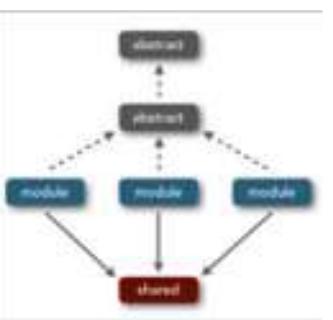


distributed architecture and remote service issues

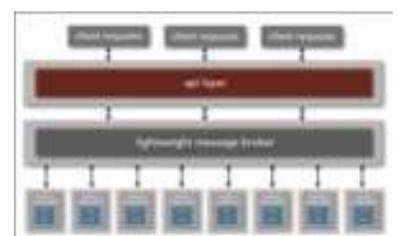
migration challenges



service component granularity and transactional boundaries



shared services, modules, and object hierarchies



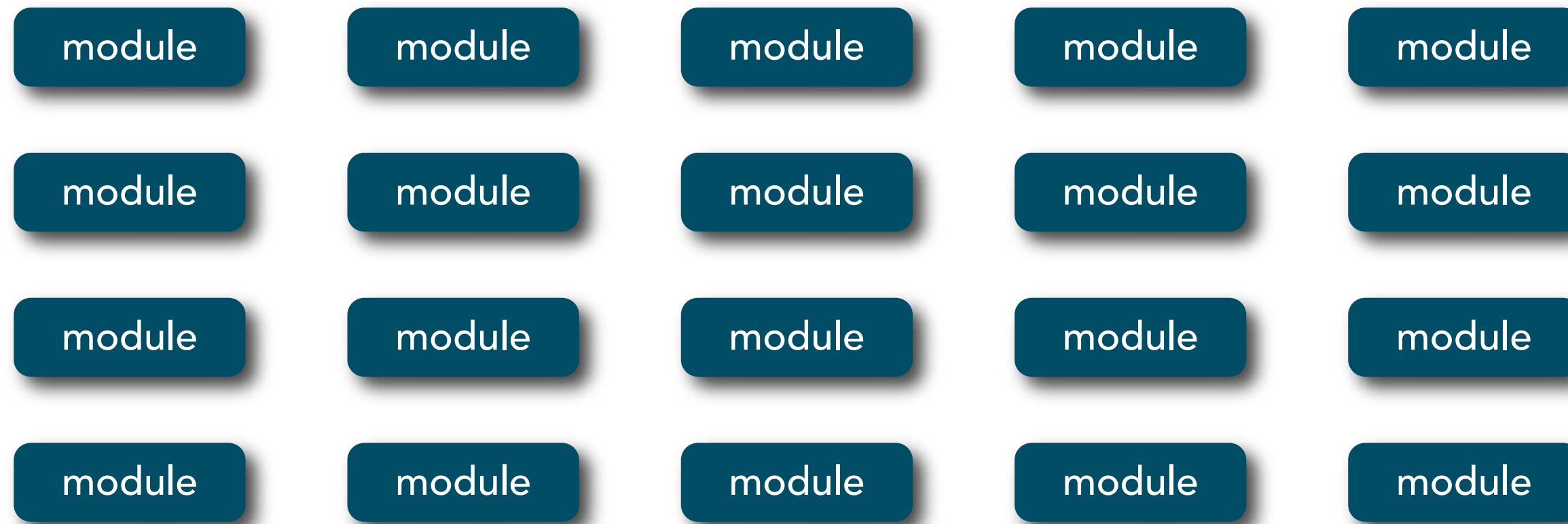
distributed architecture and remote service issues



large shared relational database

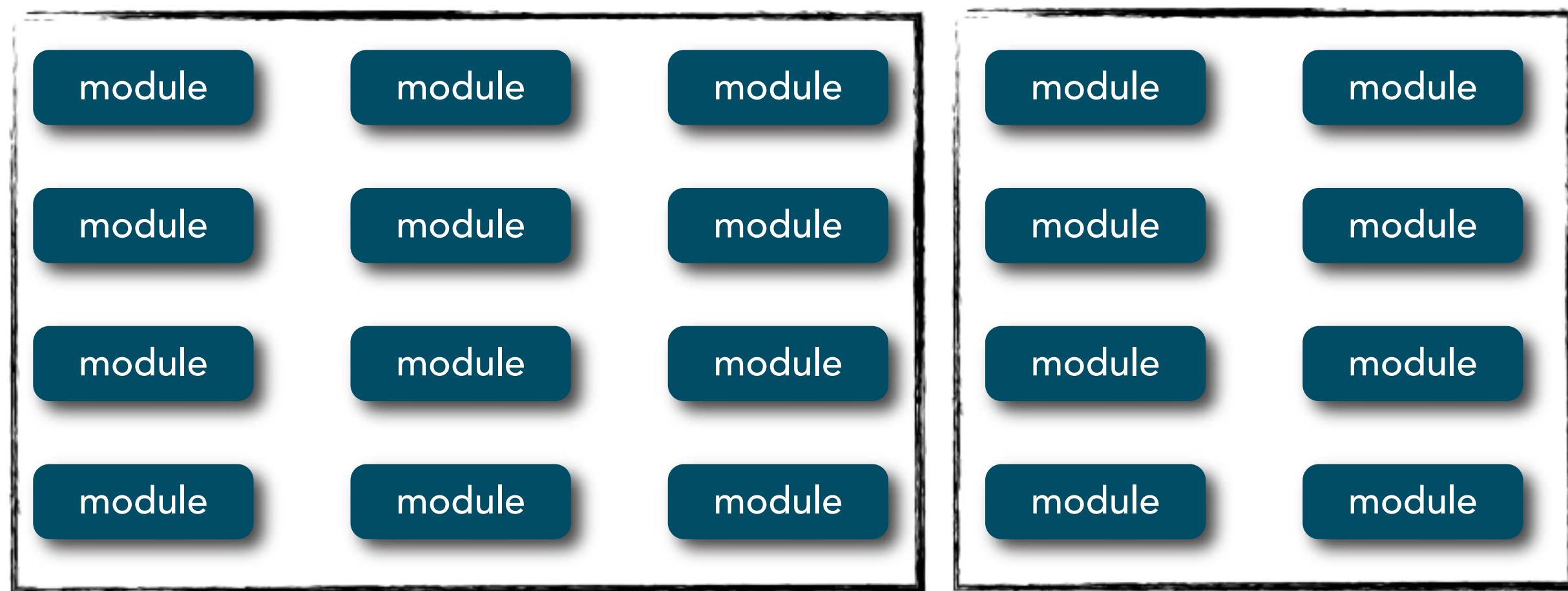
migration challenges

service component granularity



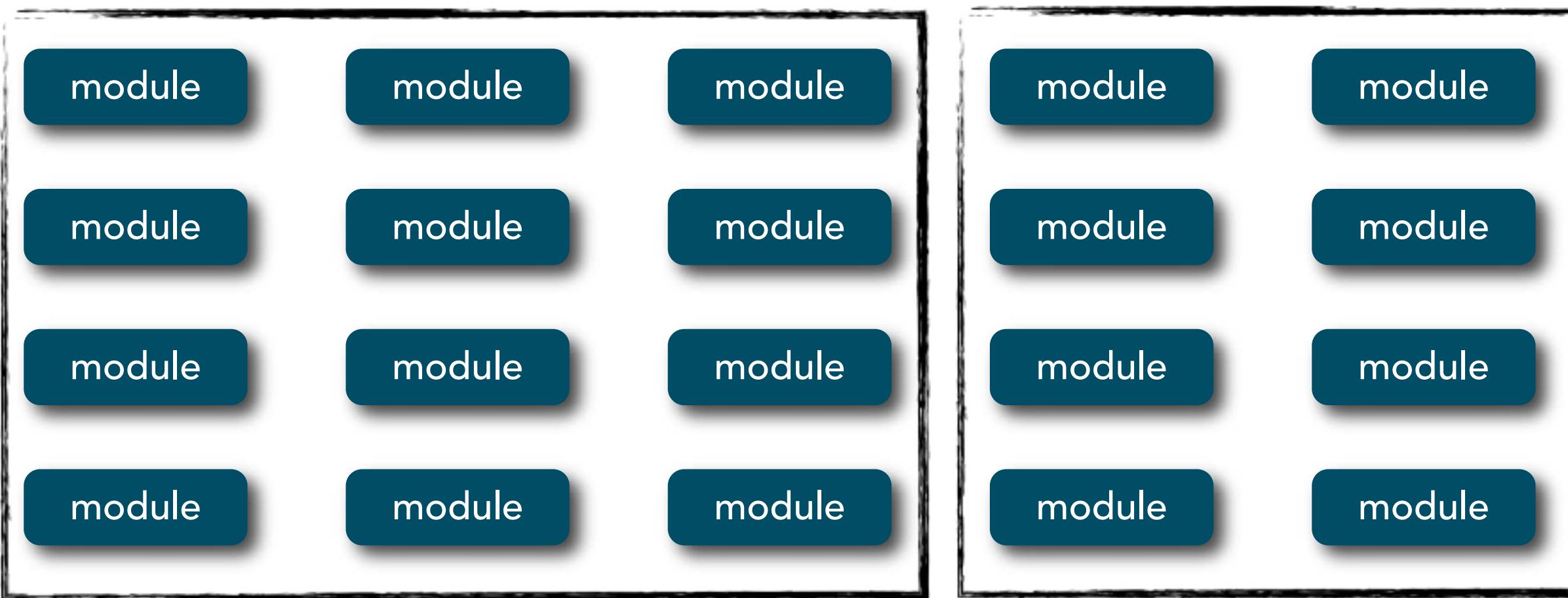
migration challenges

service component granularity



migration challenges

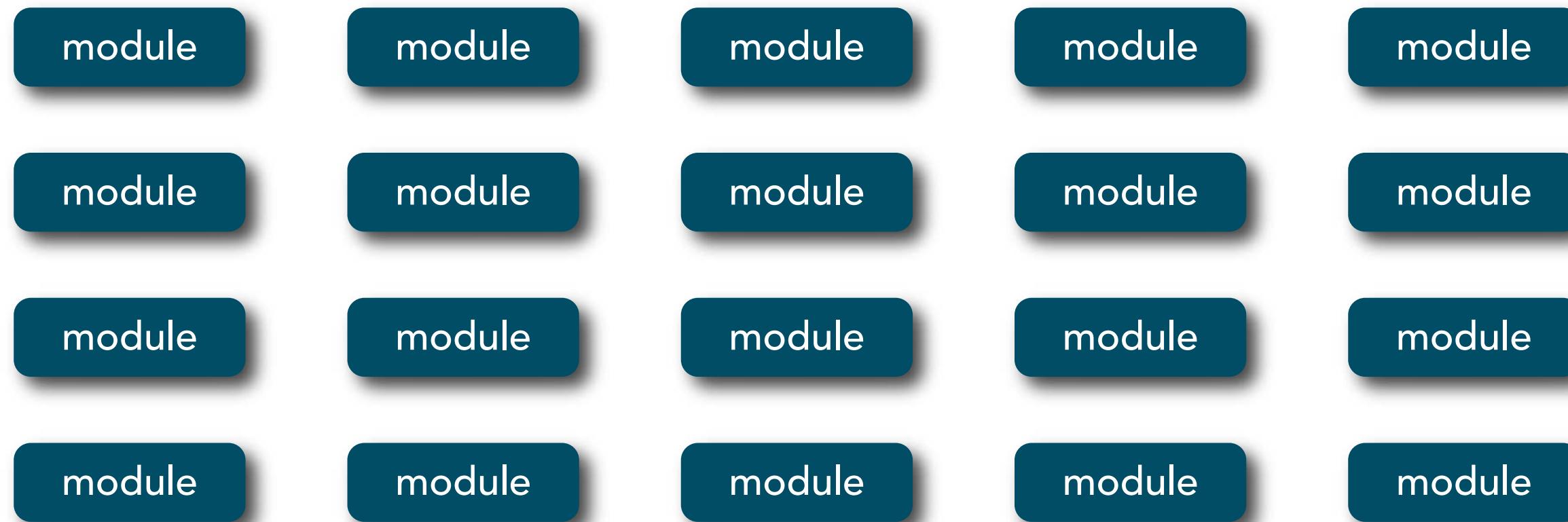
service component granularity



coarse-grained service components address transactional issues but may not achieve your desired goals

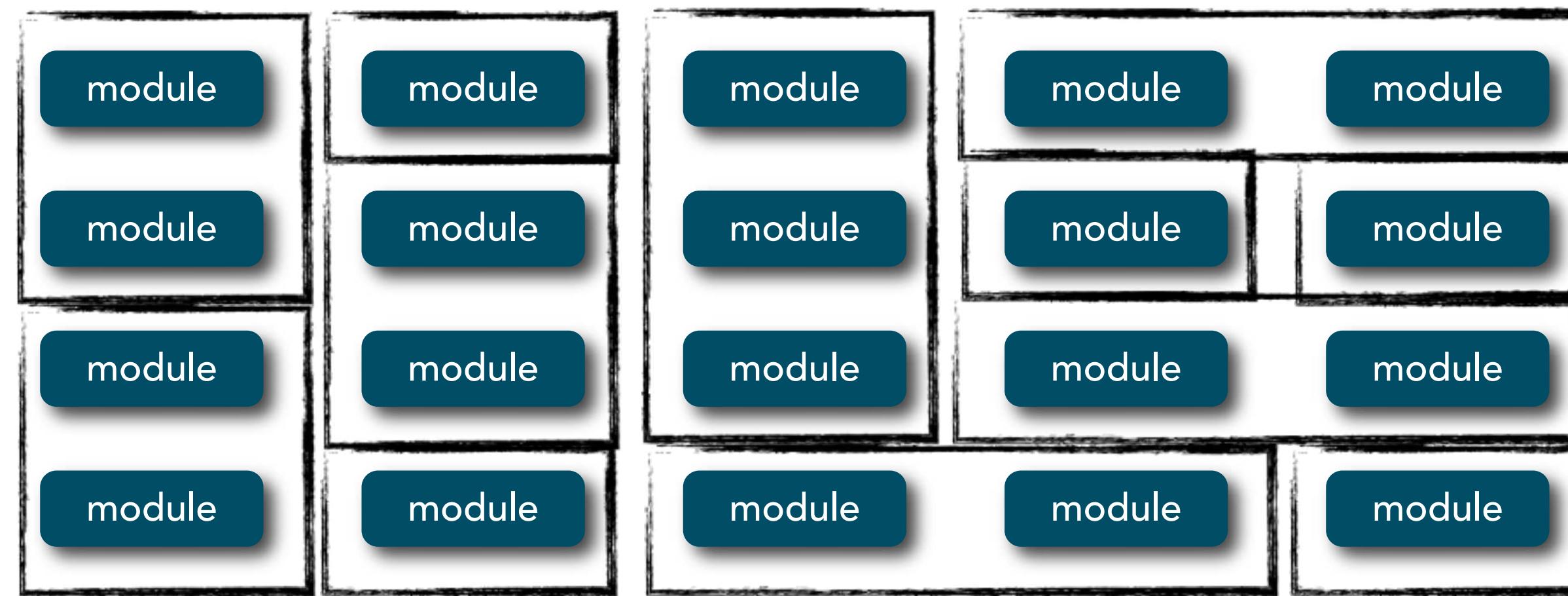
migration challenges

service component granularity



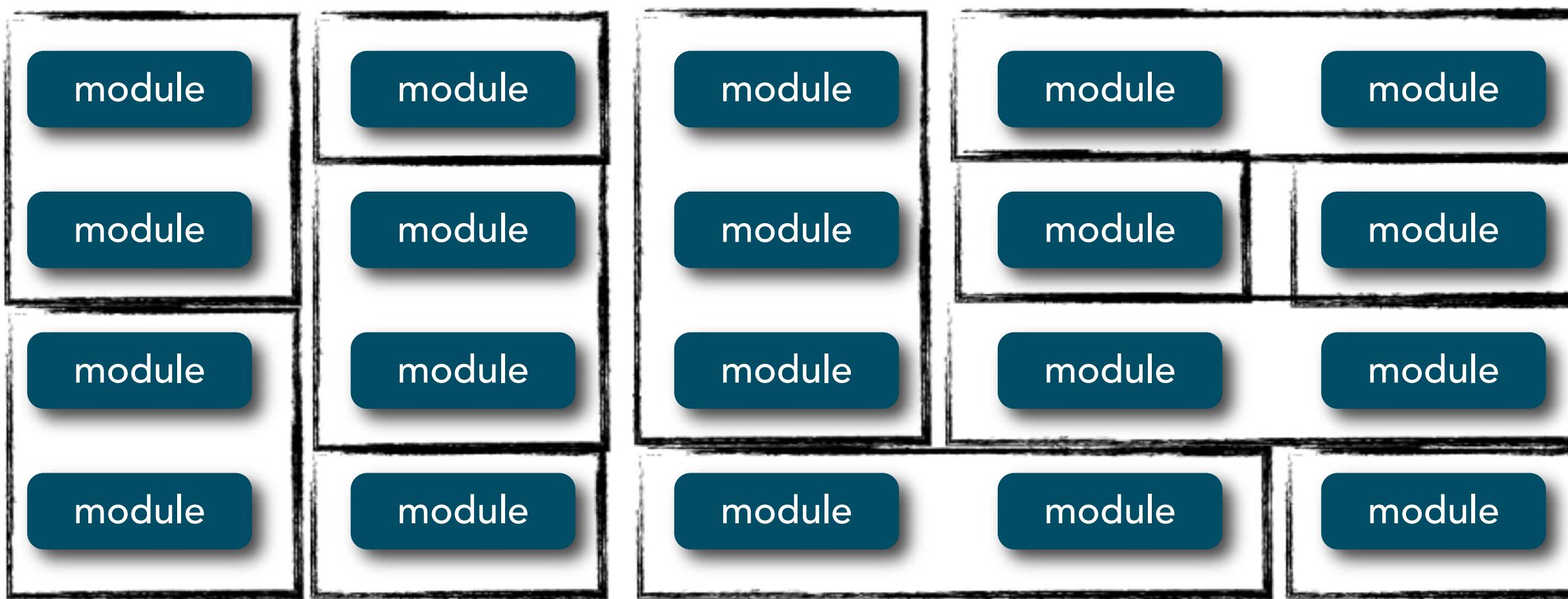
migration challenges

service component granularity



migration challenges

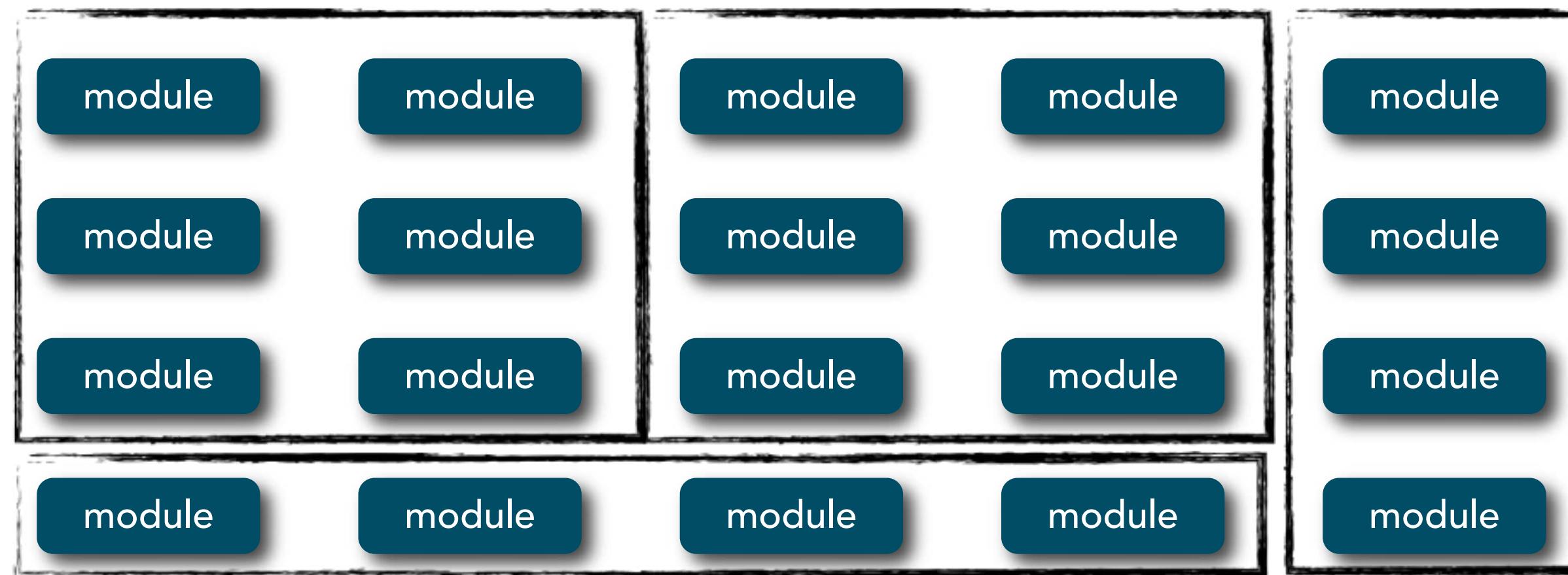
service component granularity



fine-grained service components may lead to too much orchestration and inter-dependency between service components

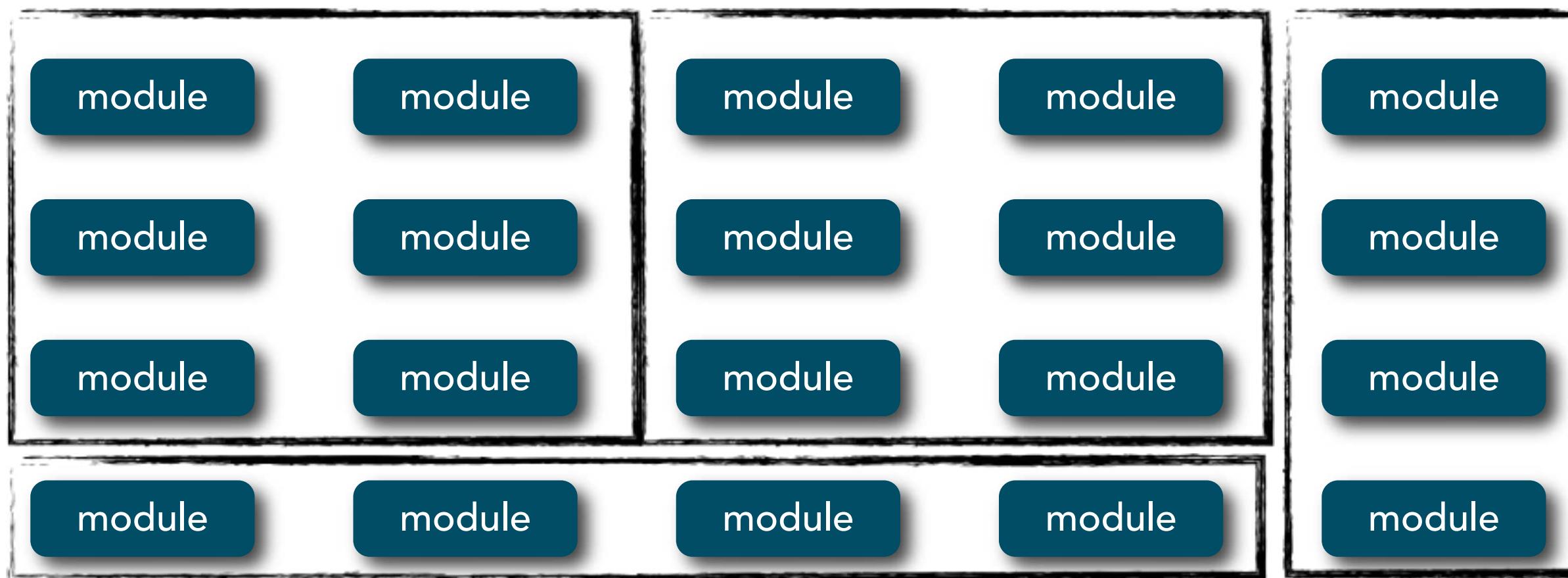
migration challenges

service component granularity



migration challenges

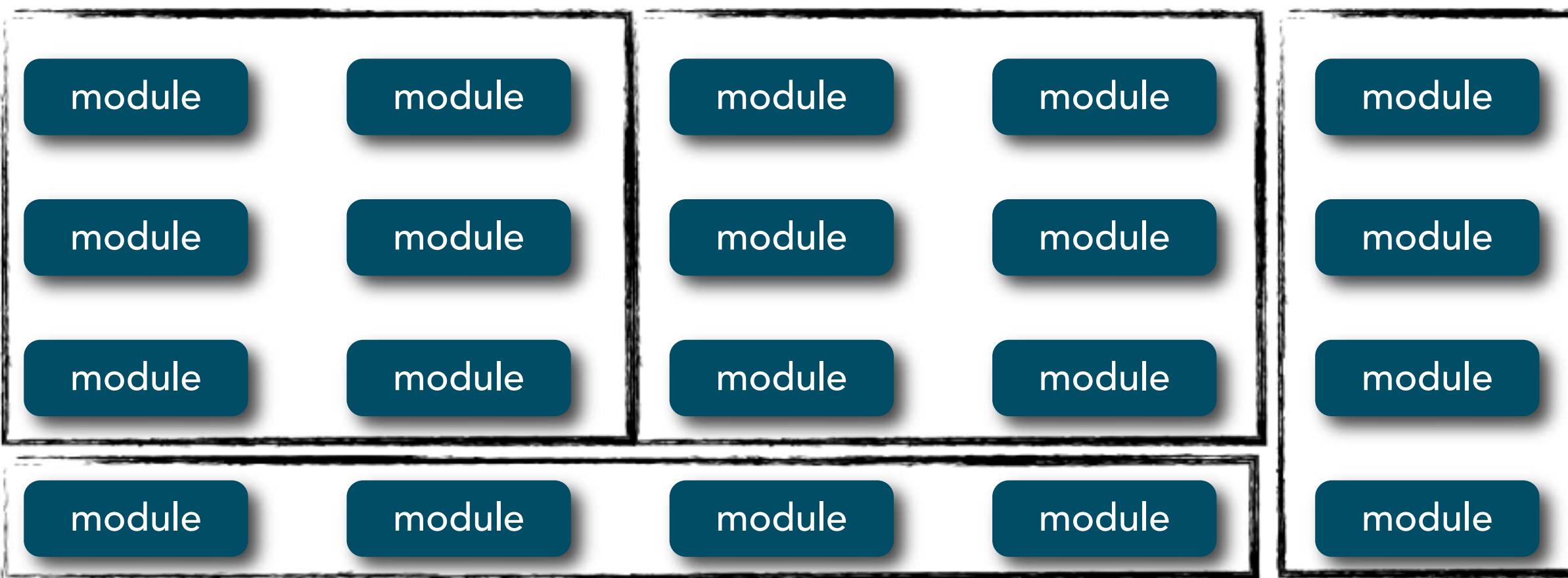
service component granularity



business functionality groupings

migration challenges

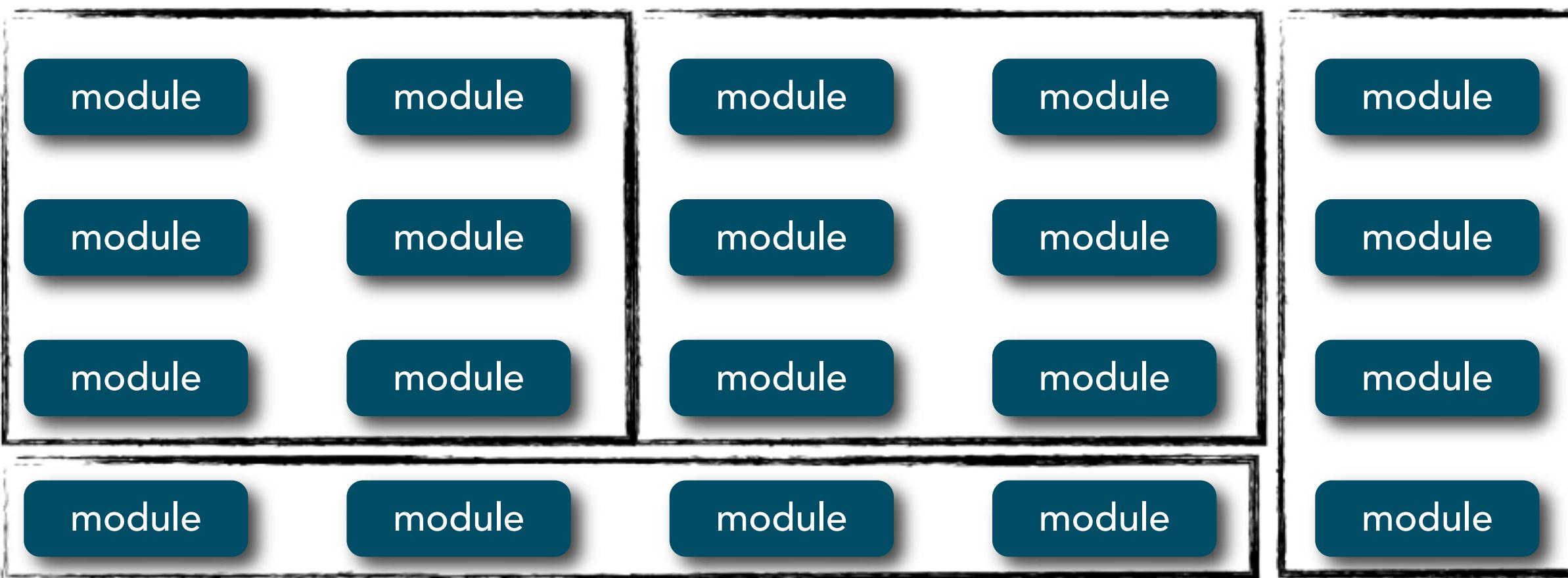
service component granularity



business functionality groupings
transactional boundaries

migration challenges

service component granularity



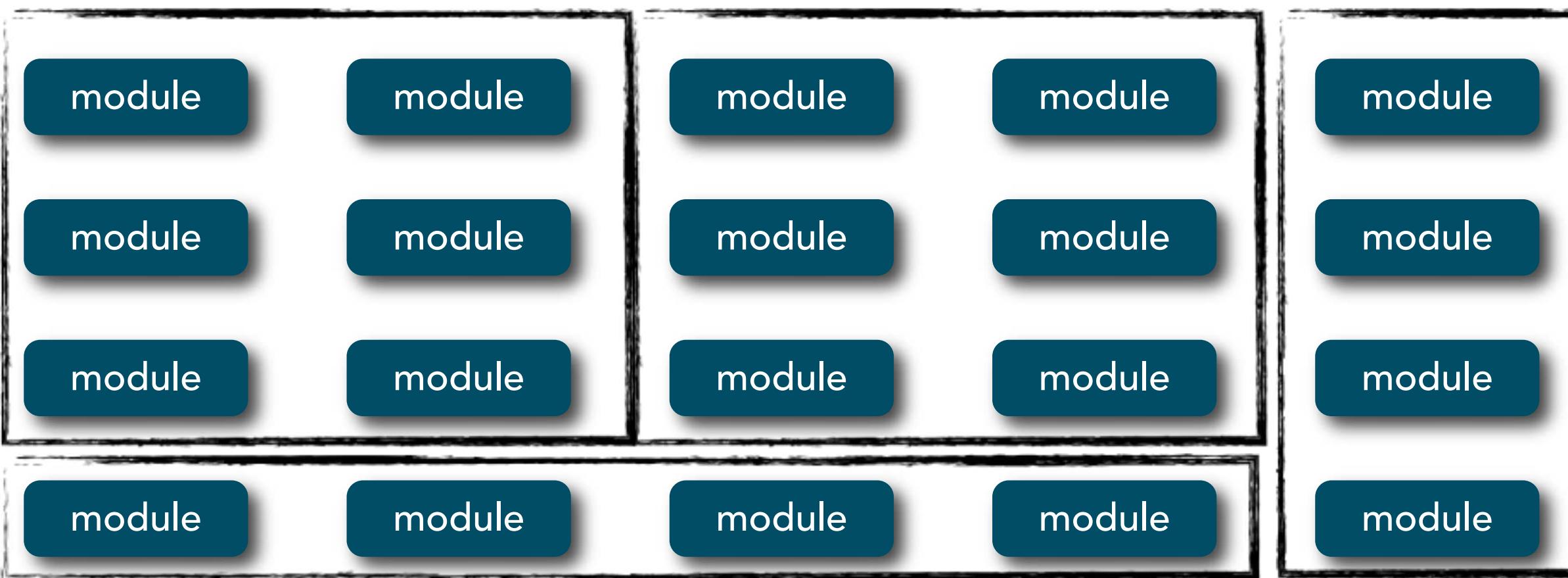
business functionality groupings

transactional boundaries

deployment goals

migration challenges

service component granularity



business functionality groupings

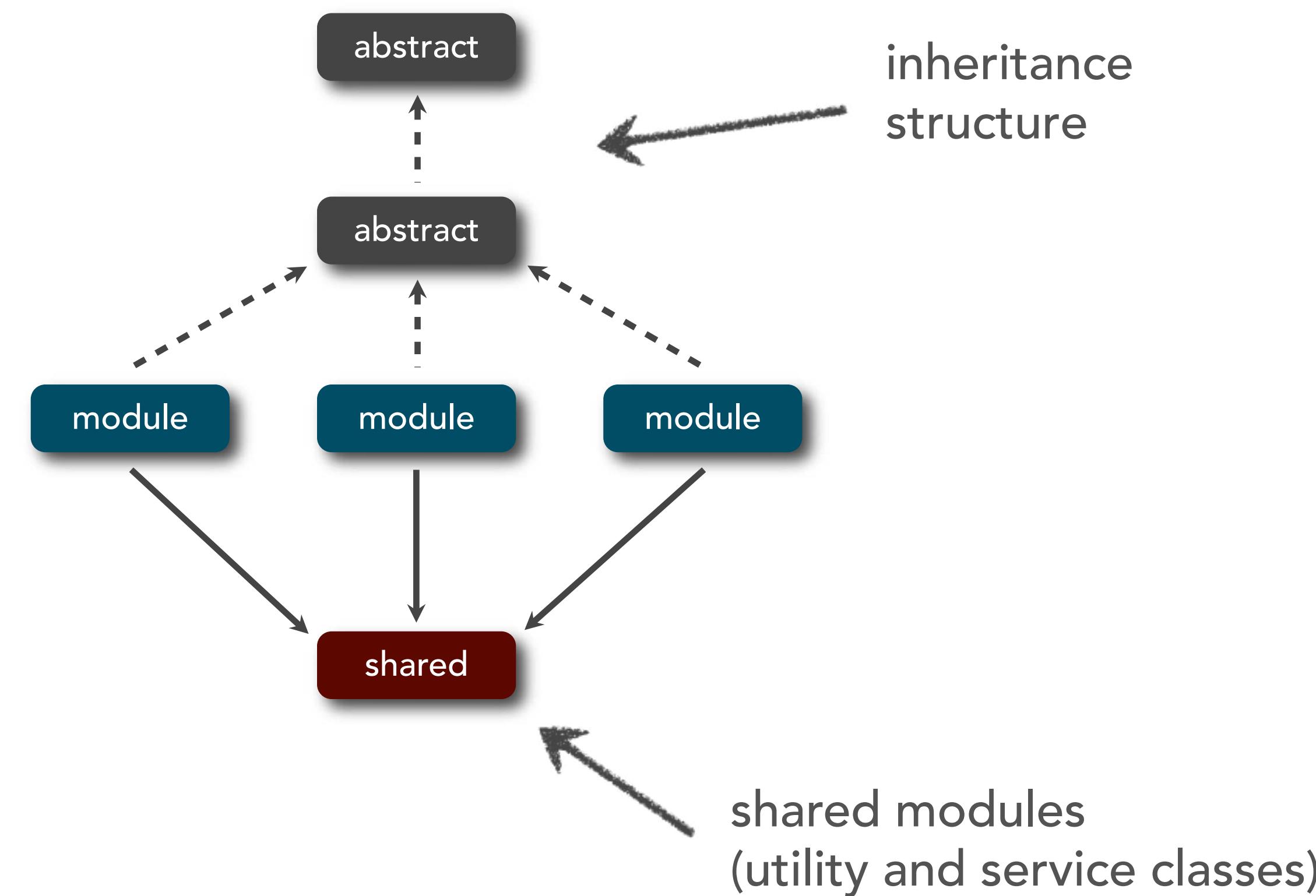
transactional boundaries

deployment goals

scalability needs

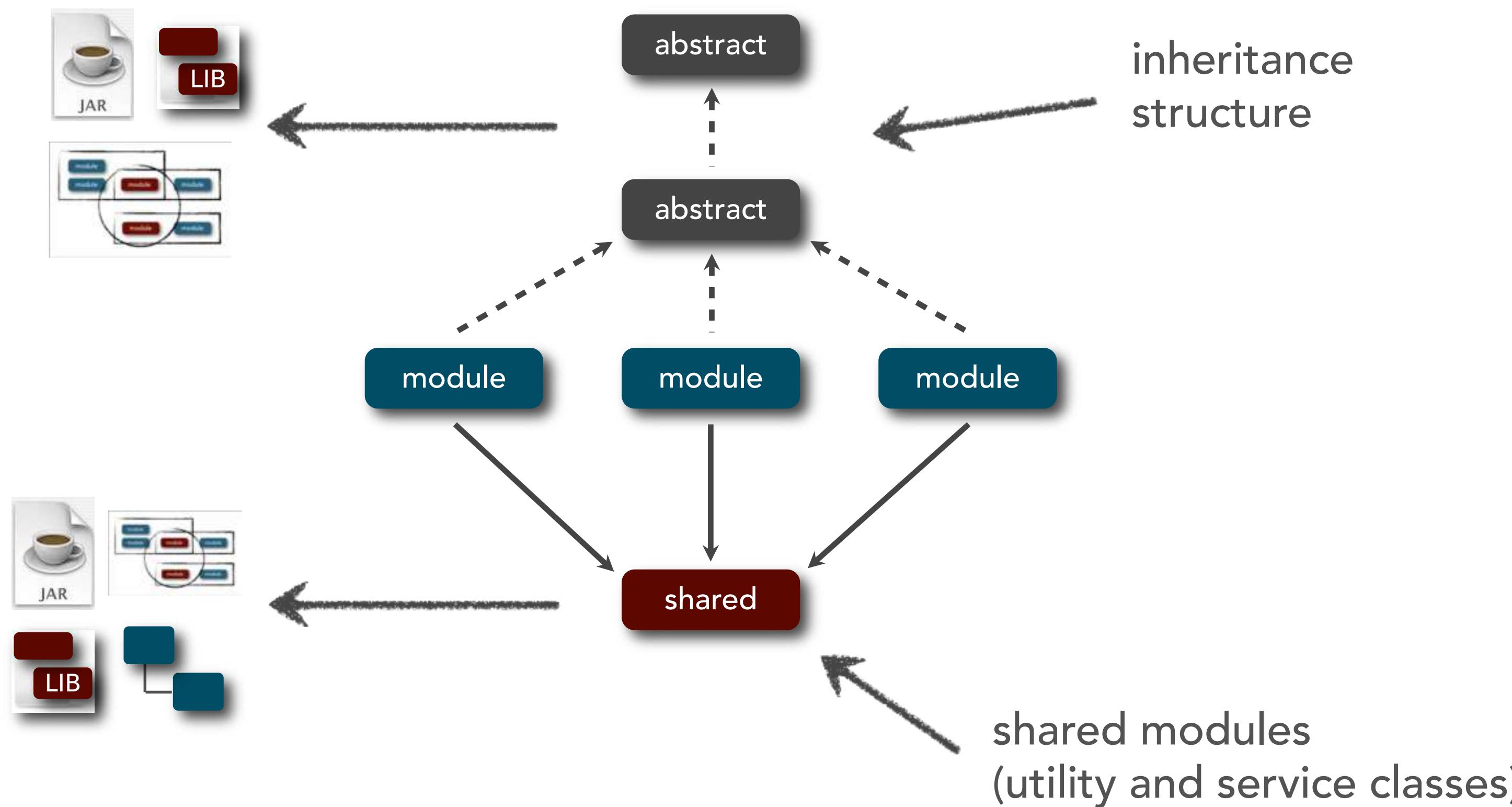
migration challenges

shared components and object hierarchies



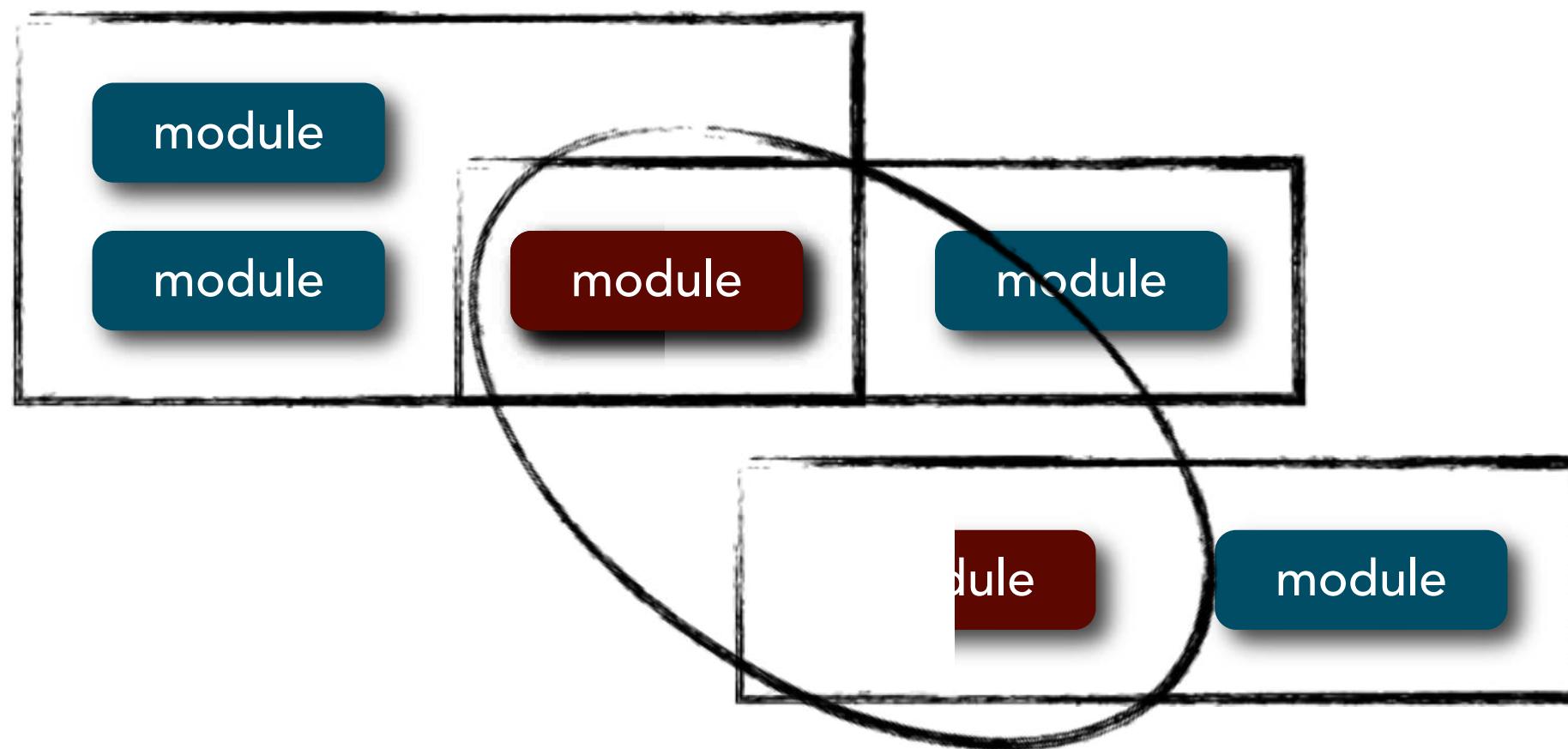
migration challenges

shared components and object hierarchies



migration challenges

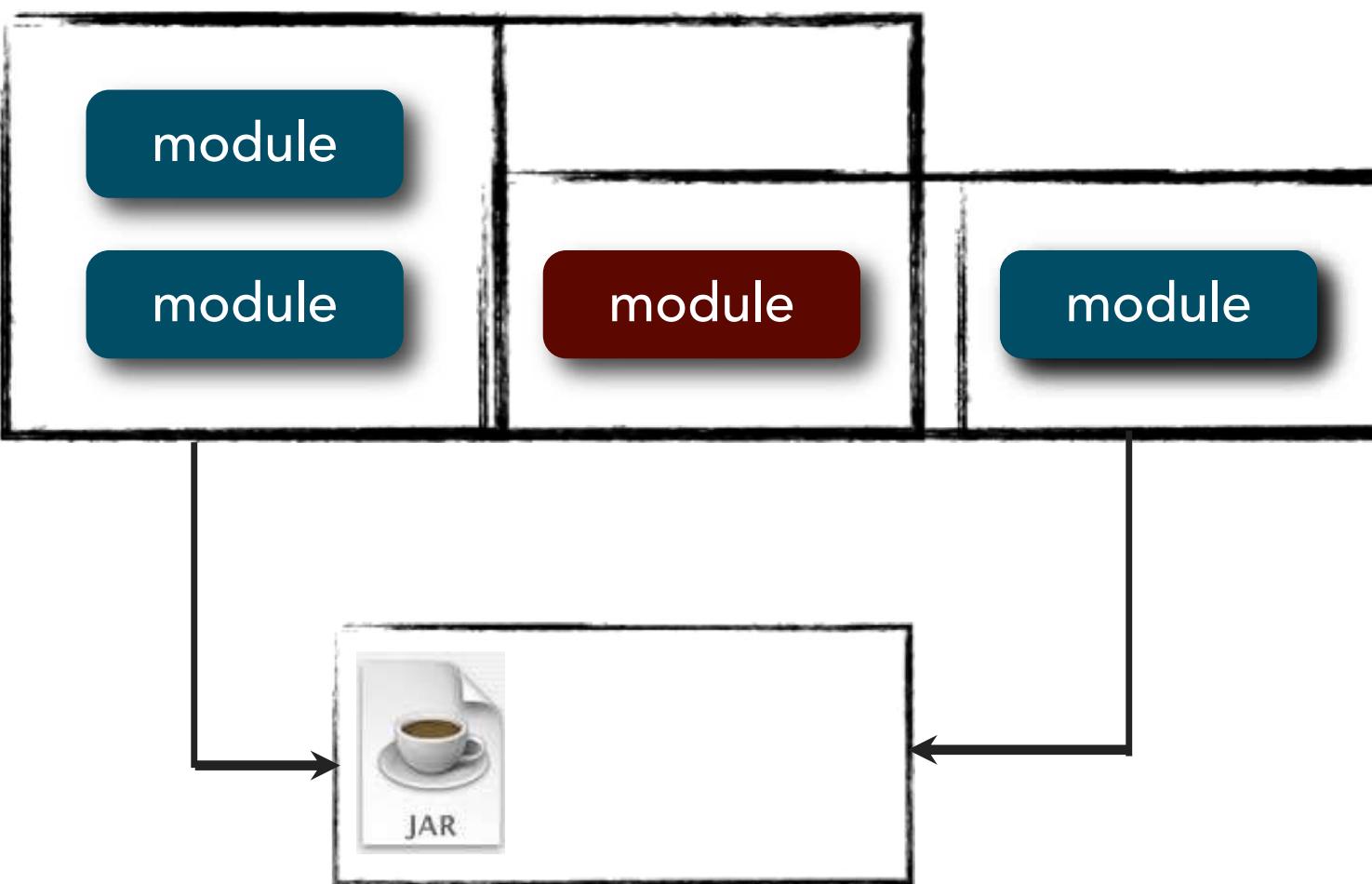
shared components and object hierarchies



modules can be split....

migration challenges

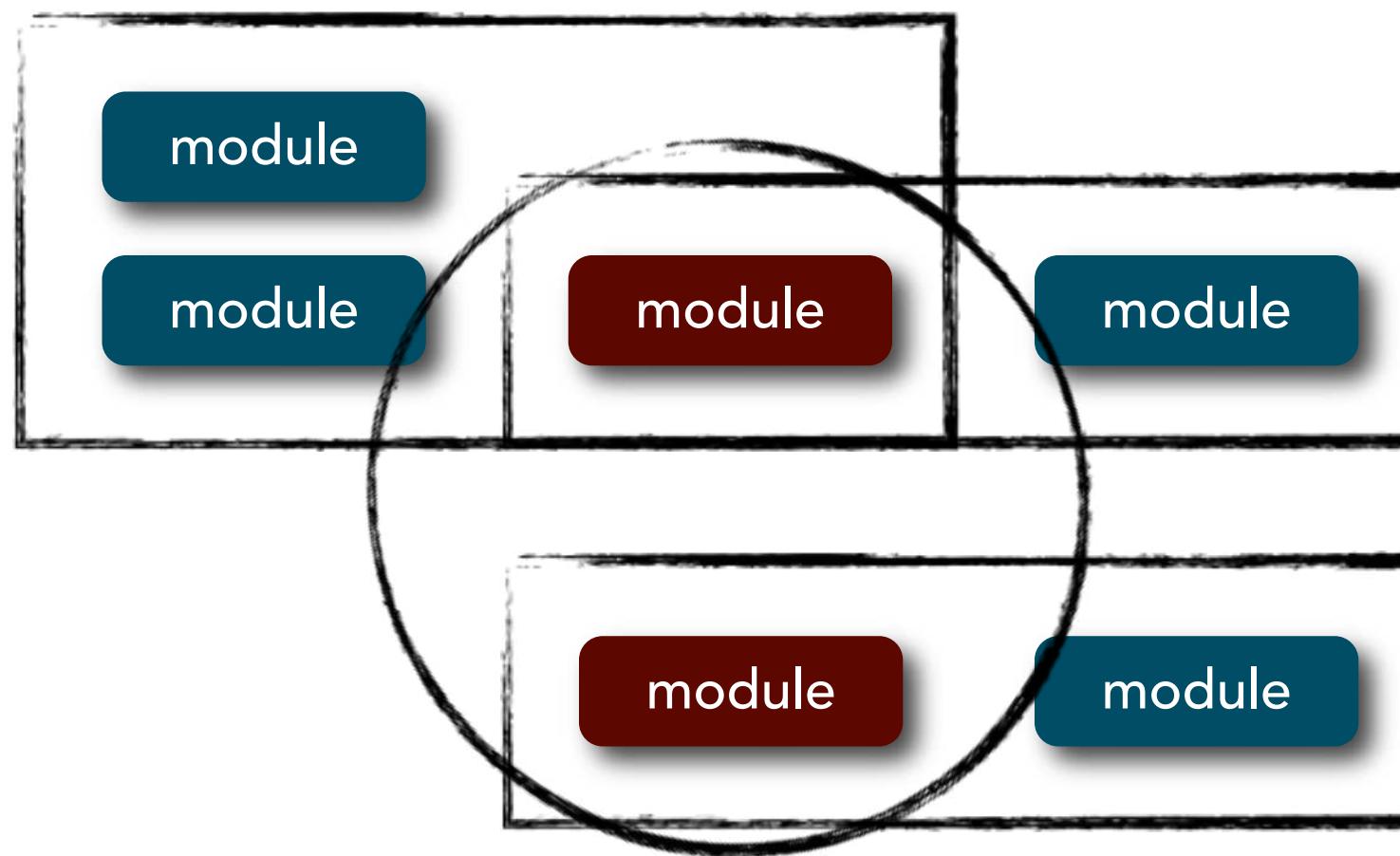
shared components and object hierarchies



or moved into a shared library or jar file...

migration challenges

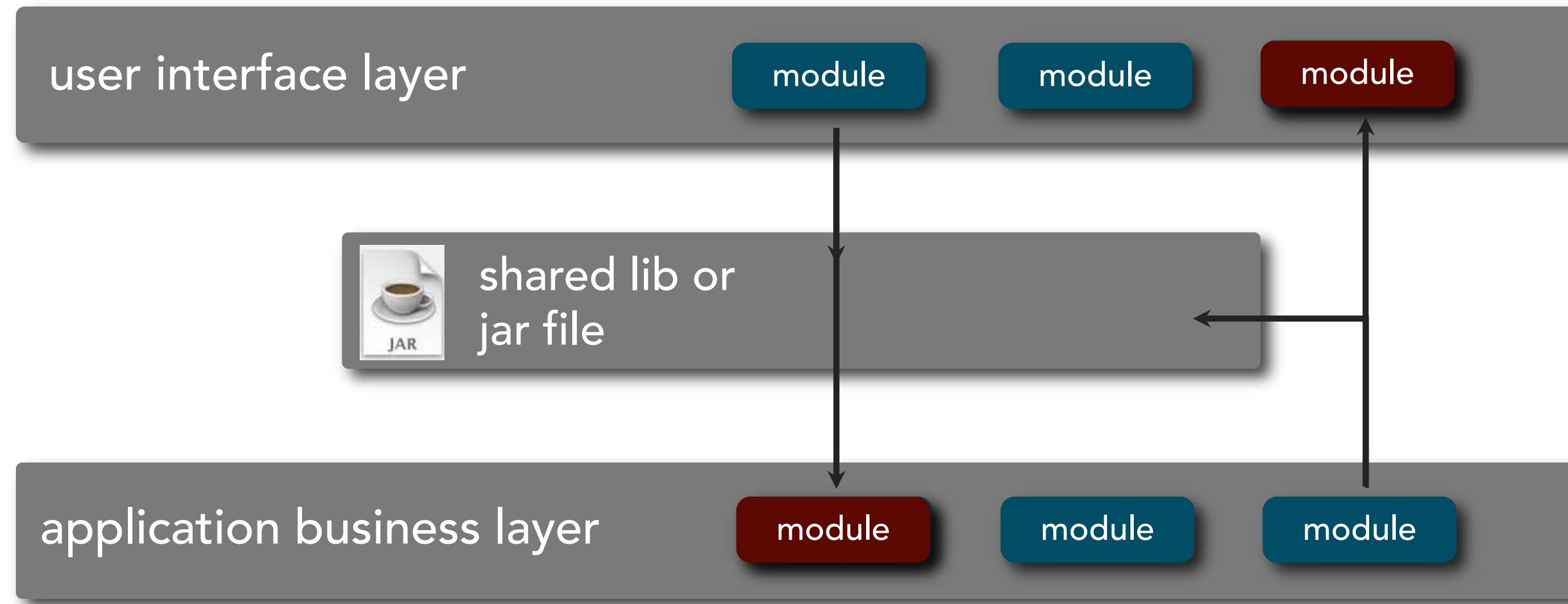
shared components and object hierarchies



or replicated in each service component

migration challenges

shared components and object hierarchies

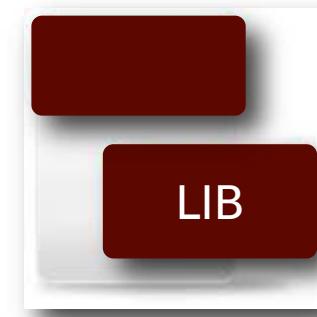


migration challenges

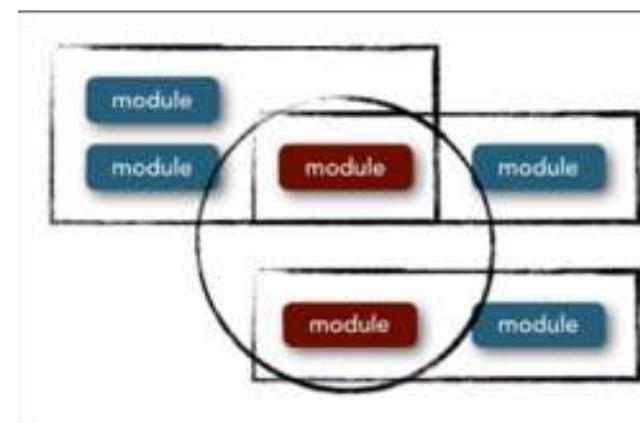
shared component techniques



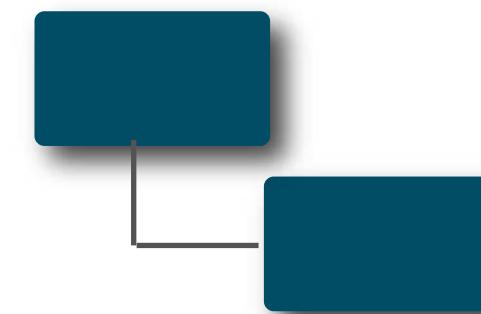
jar /dll
(compile or runtime)



shared library
(compile time)

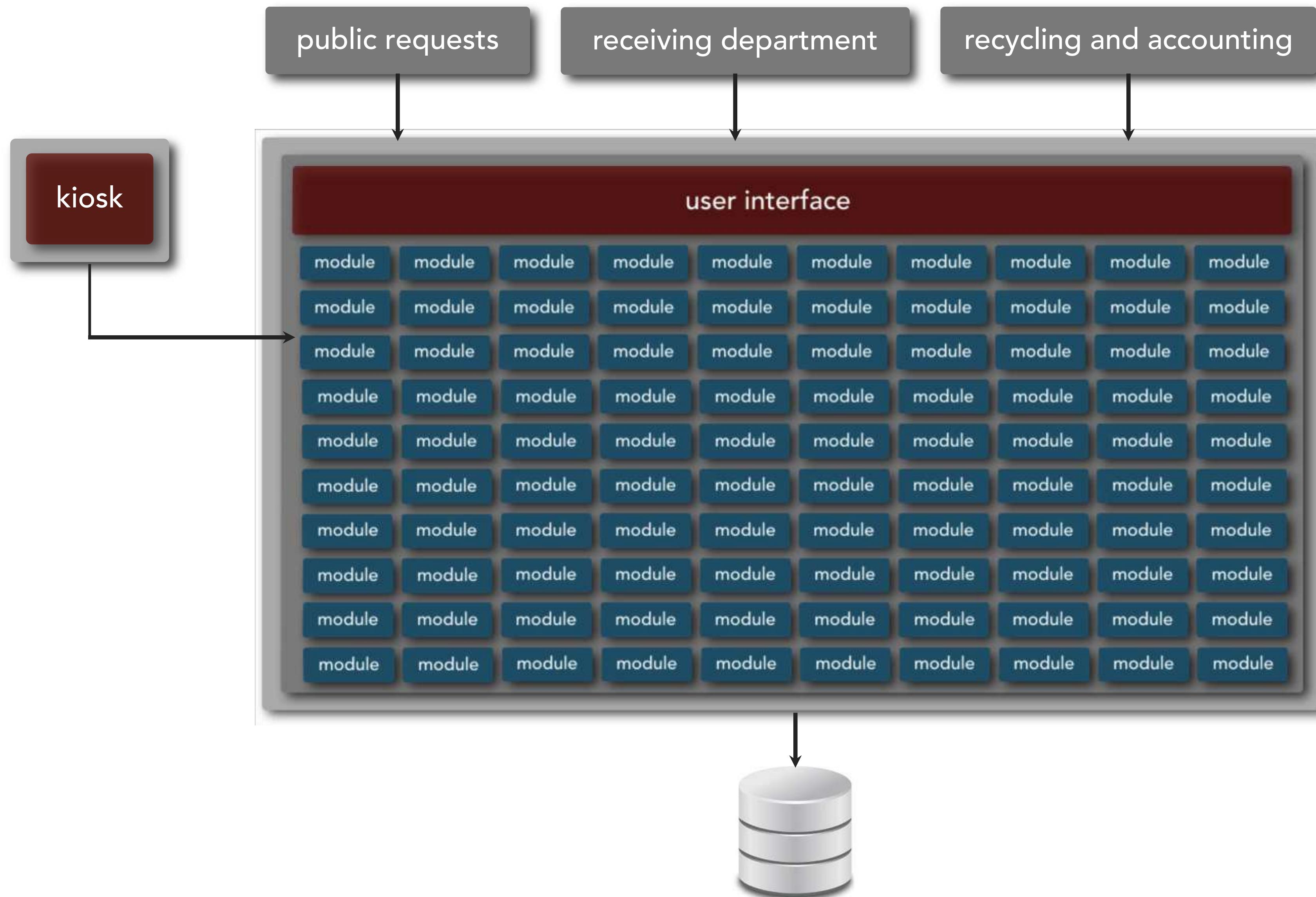


code replication

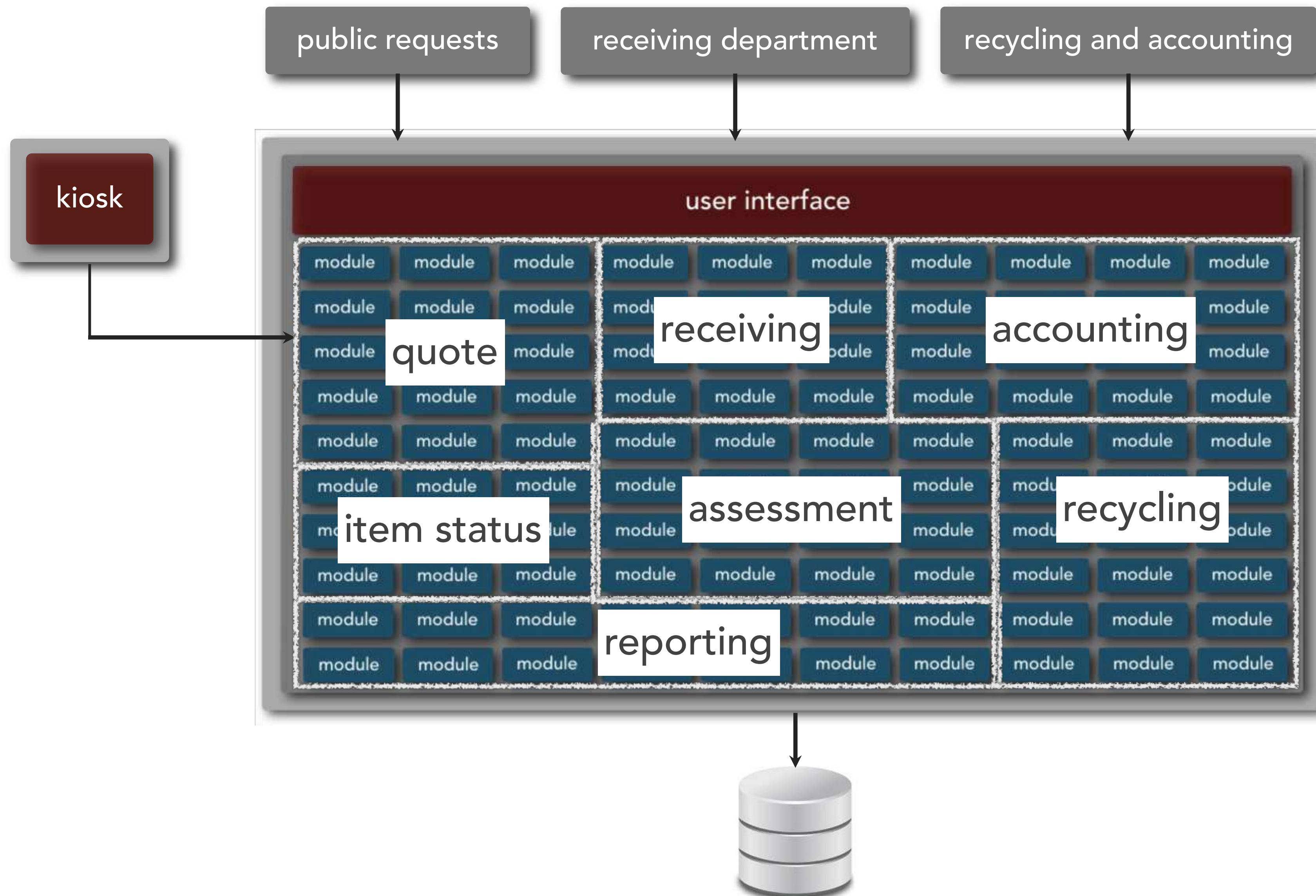


remote services

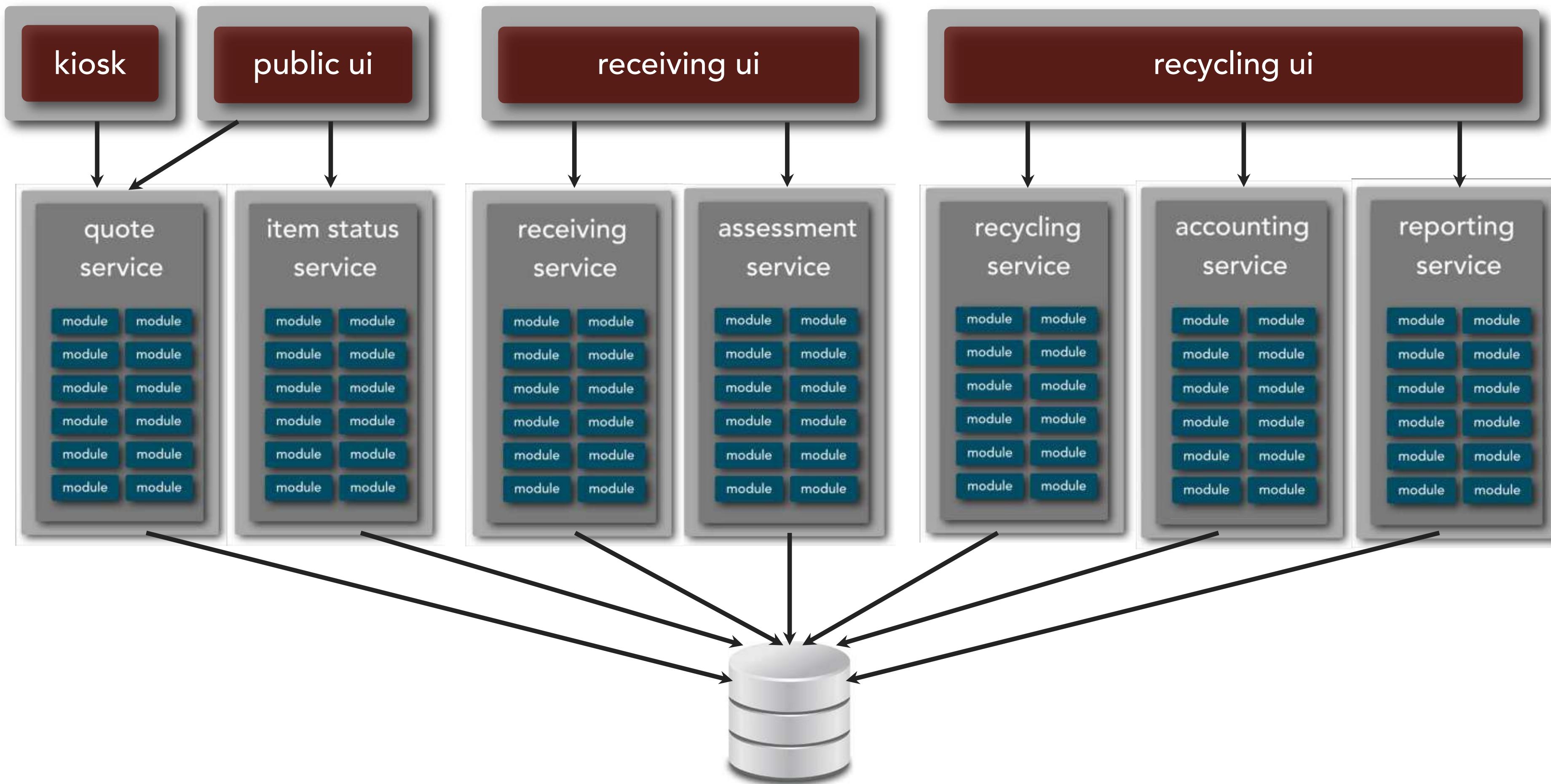
electronics recycling application



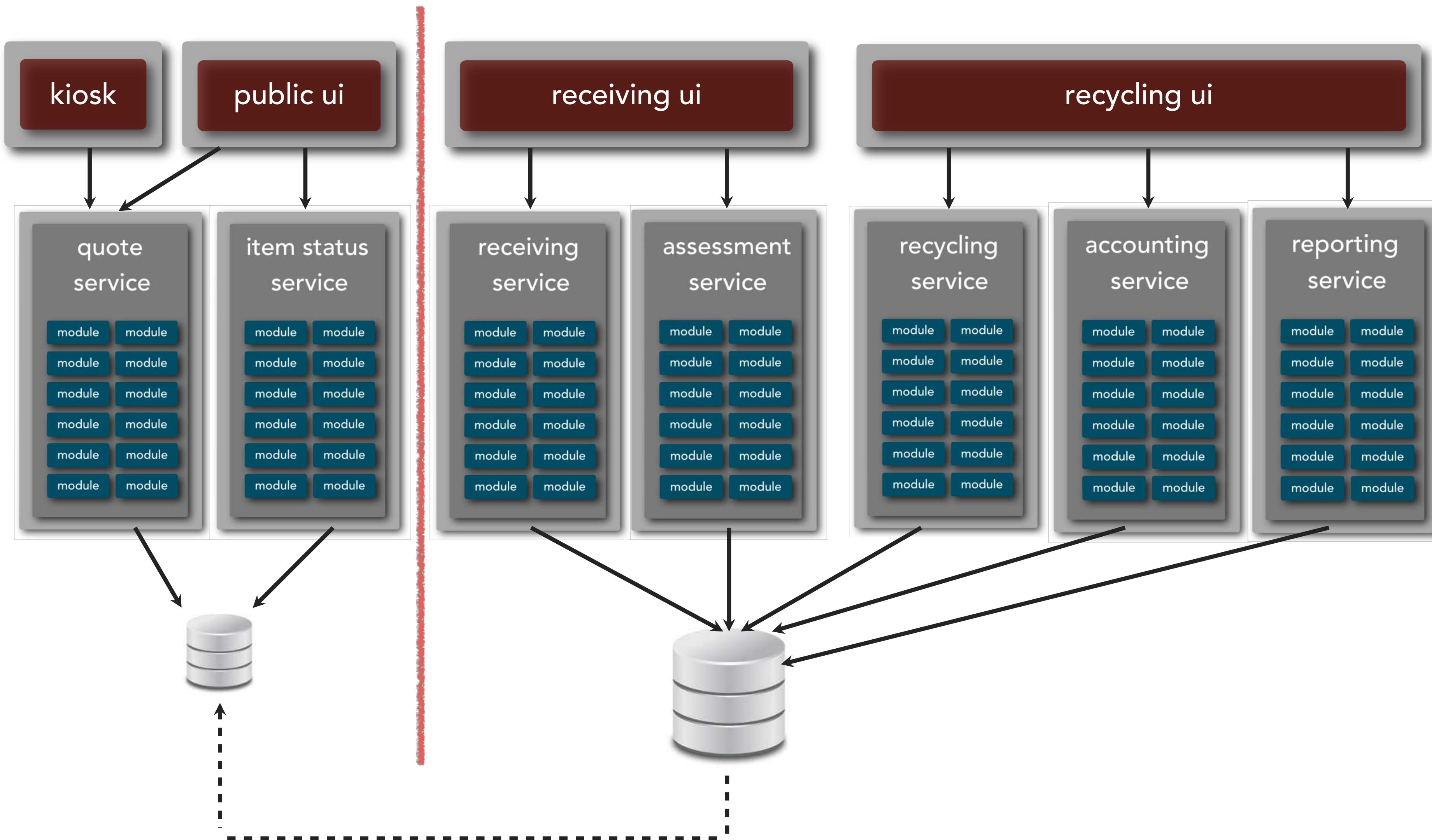
electronics recycling application



electronics recycling application

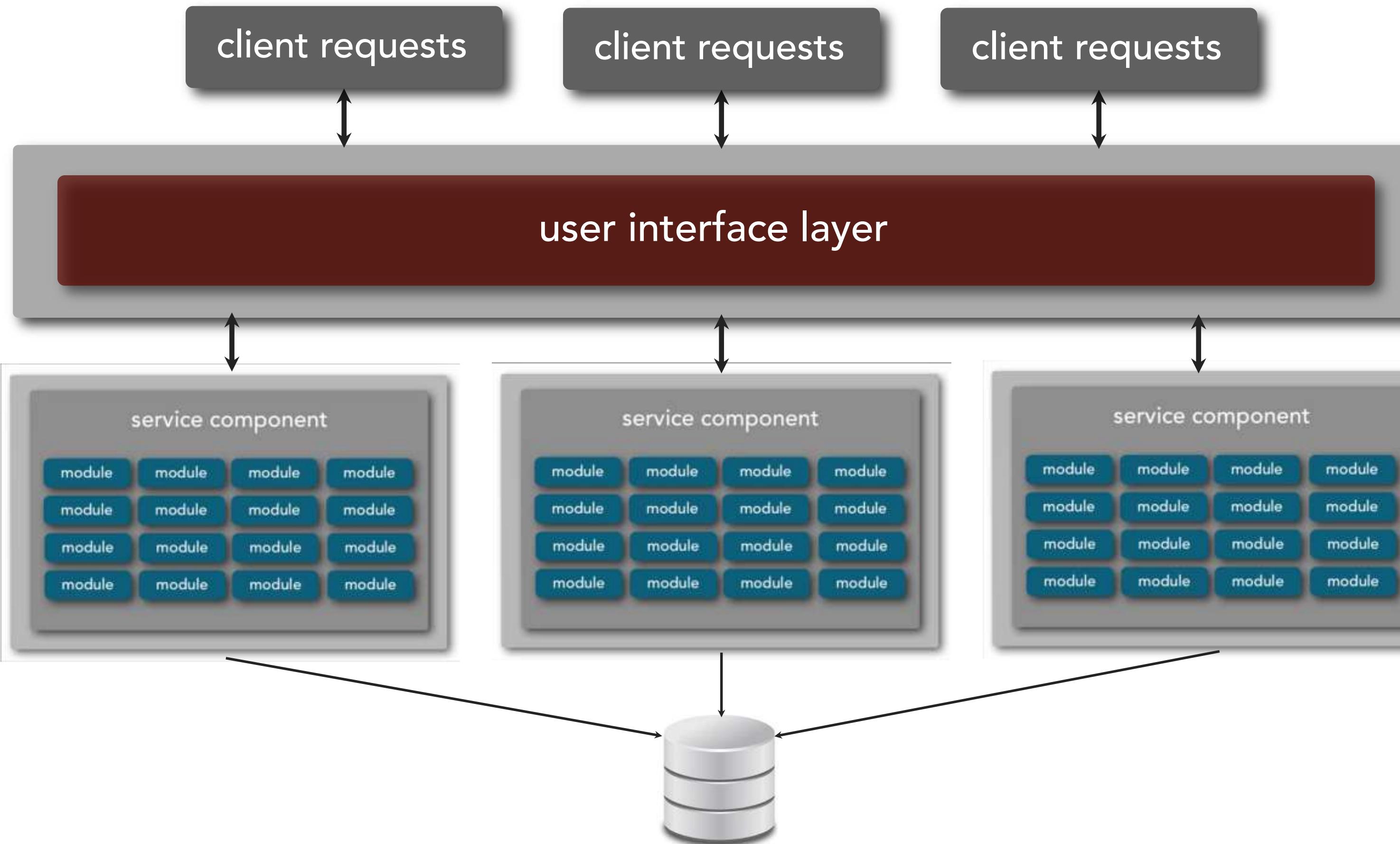


electronics recycling application



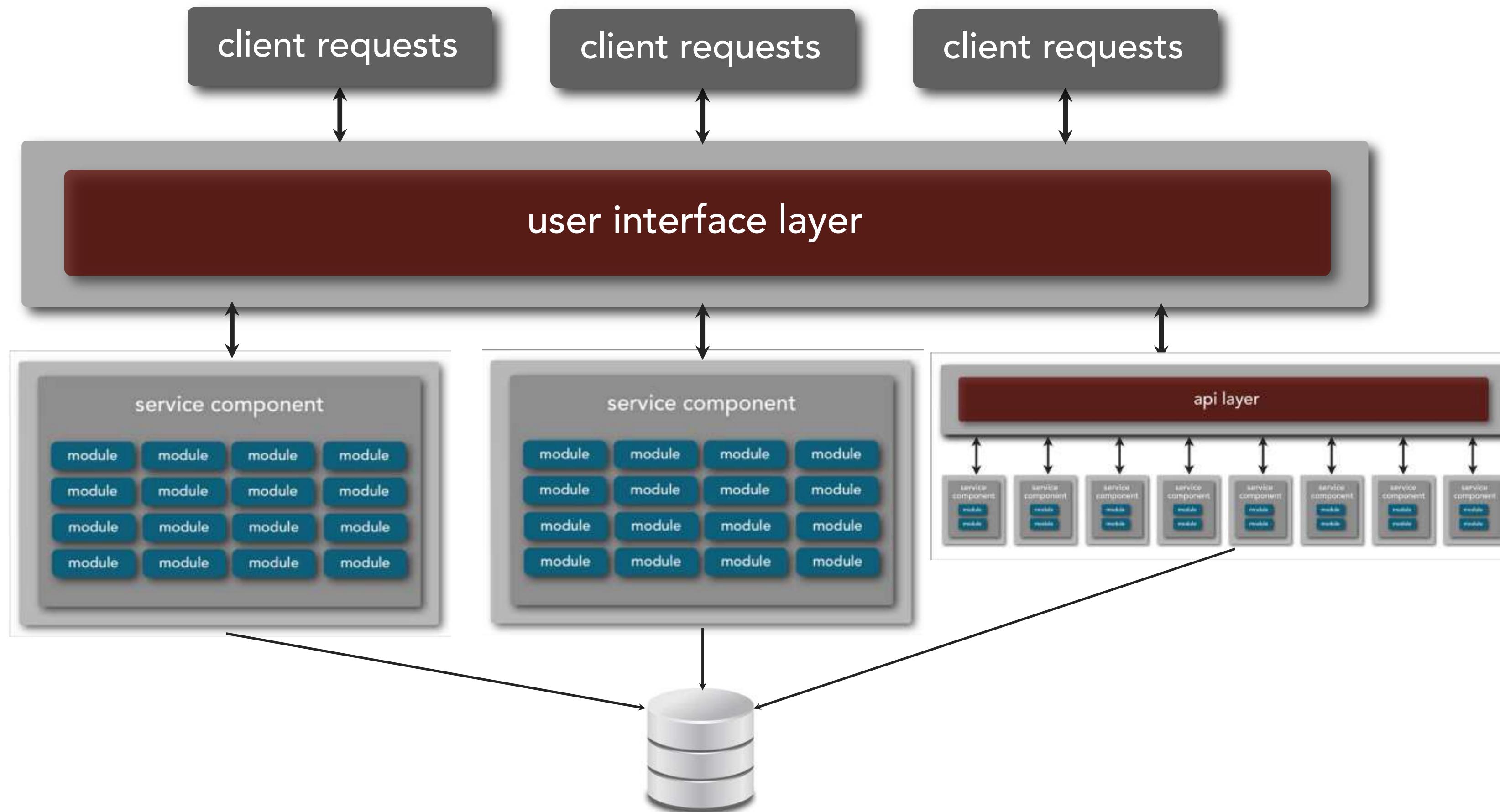
service-based architecture

adding microservices



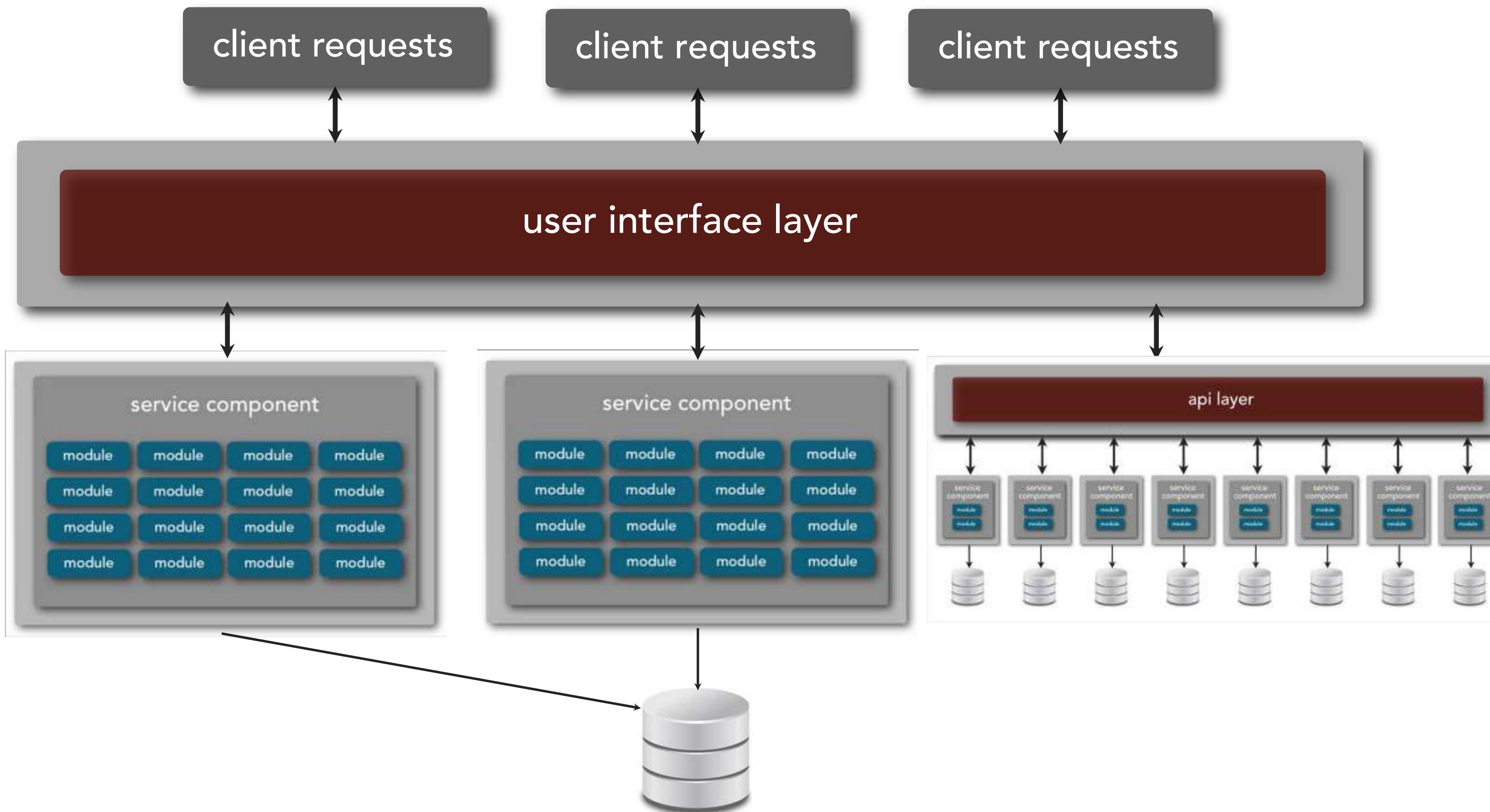
service-based architecture

adding microservices

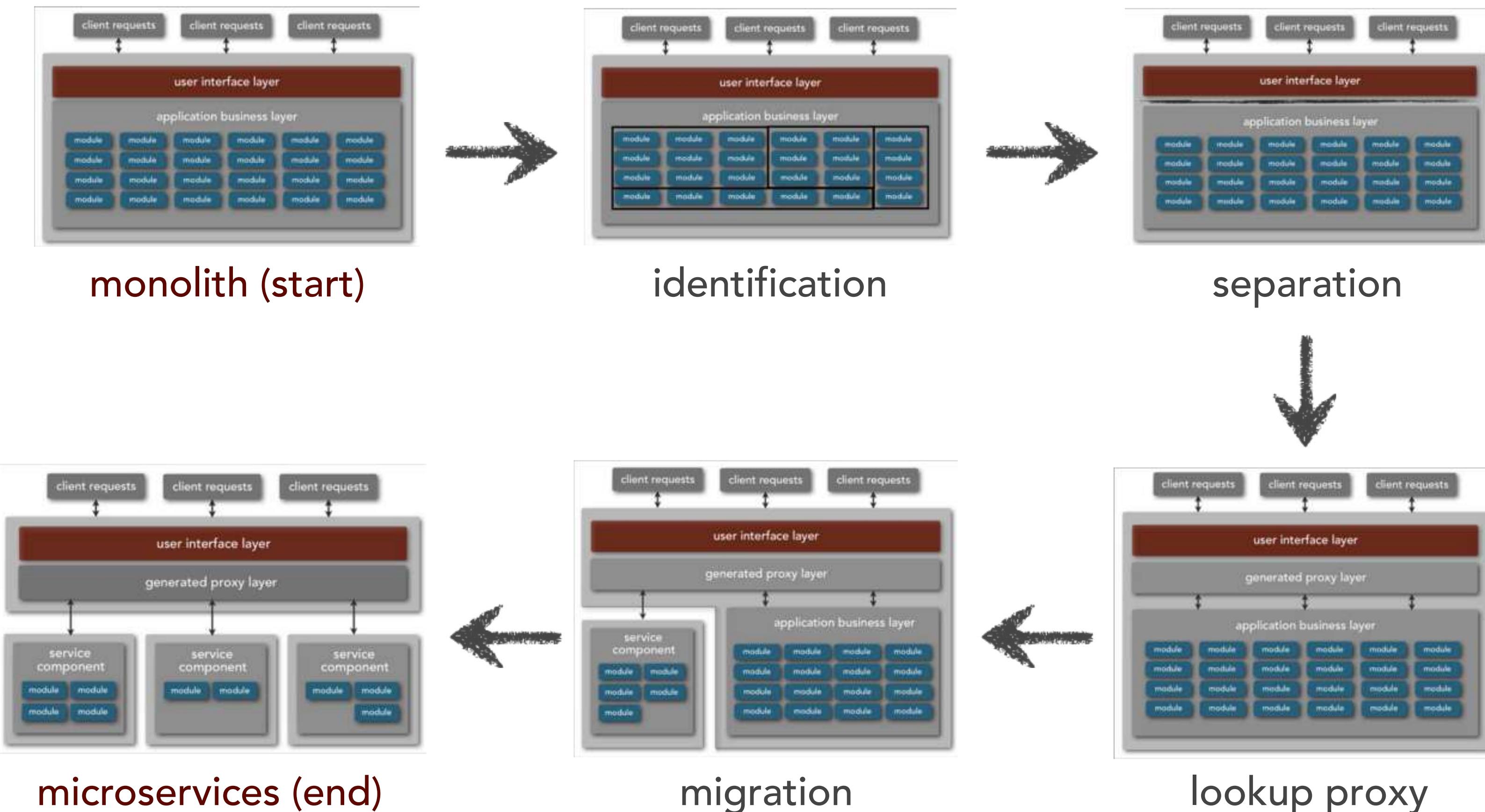


service-based architecture

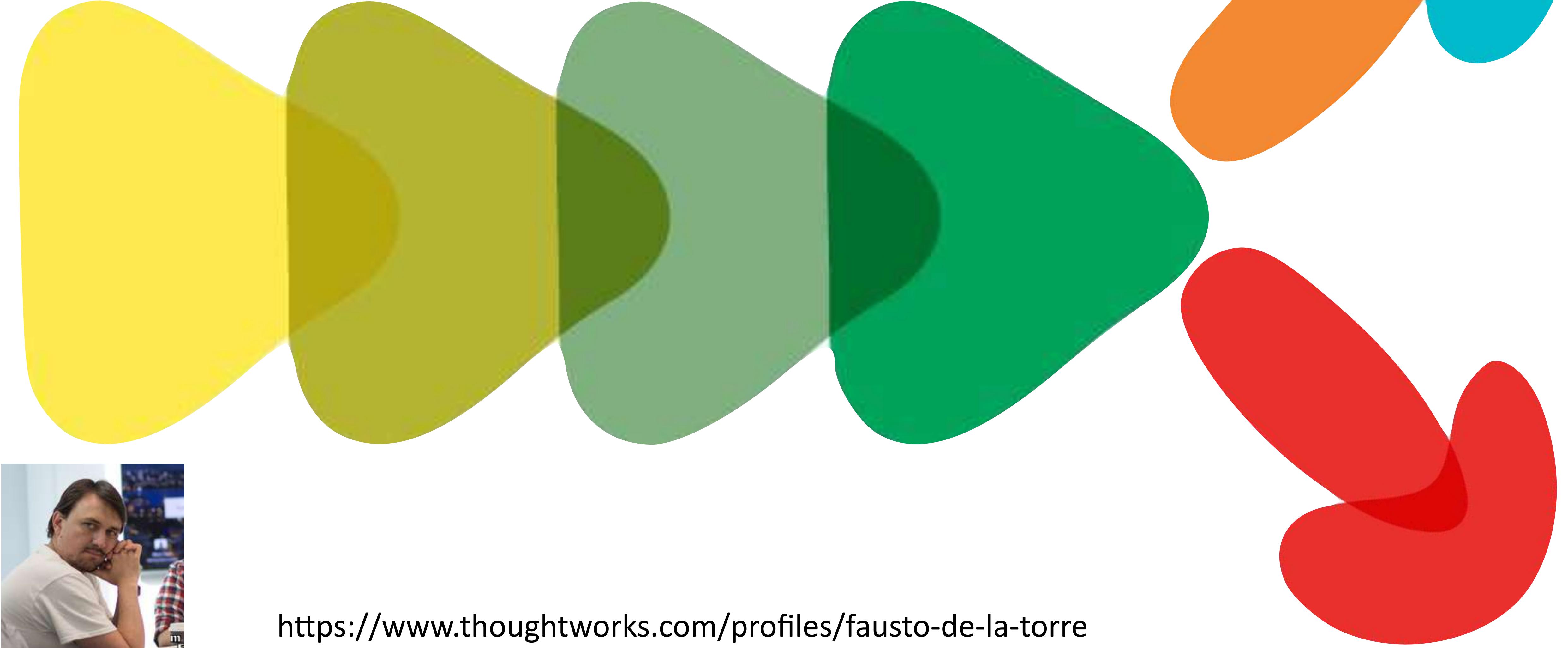
adding microservices



migration steps - refactoring

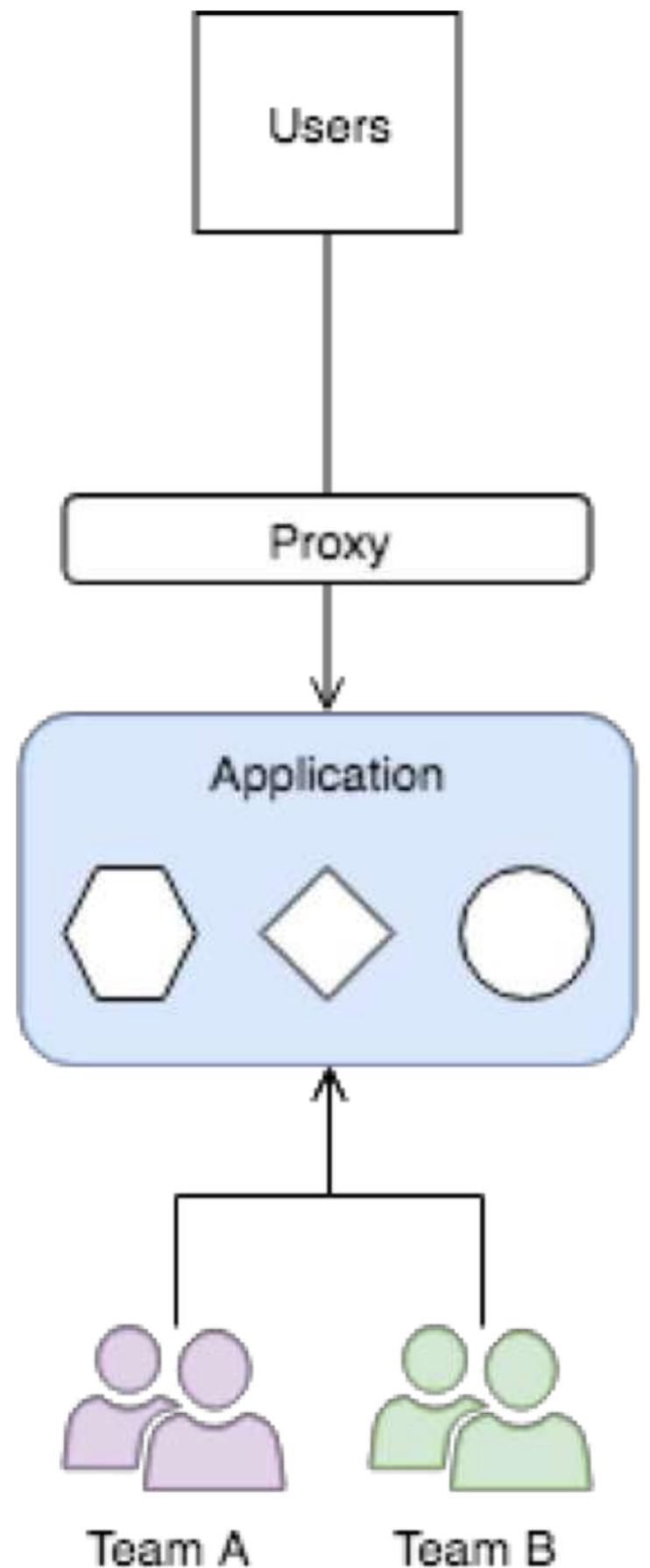


tactical forking



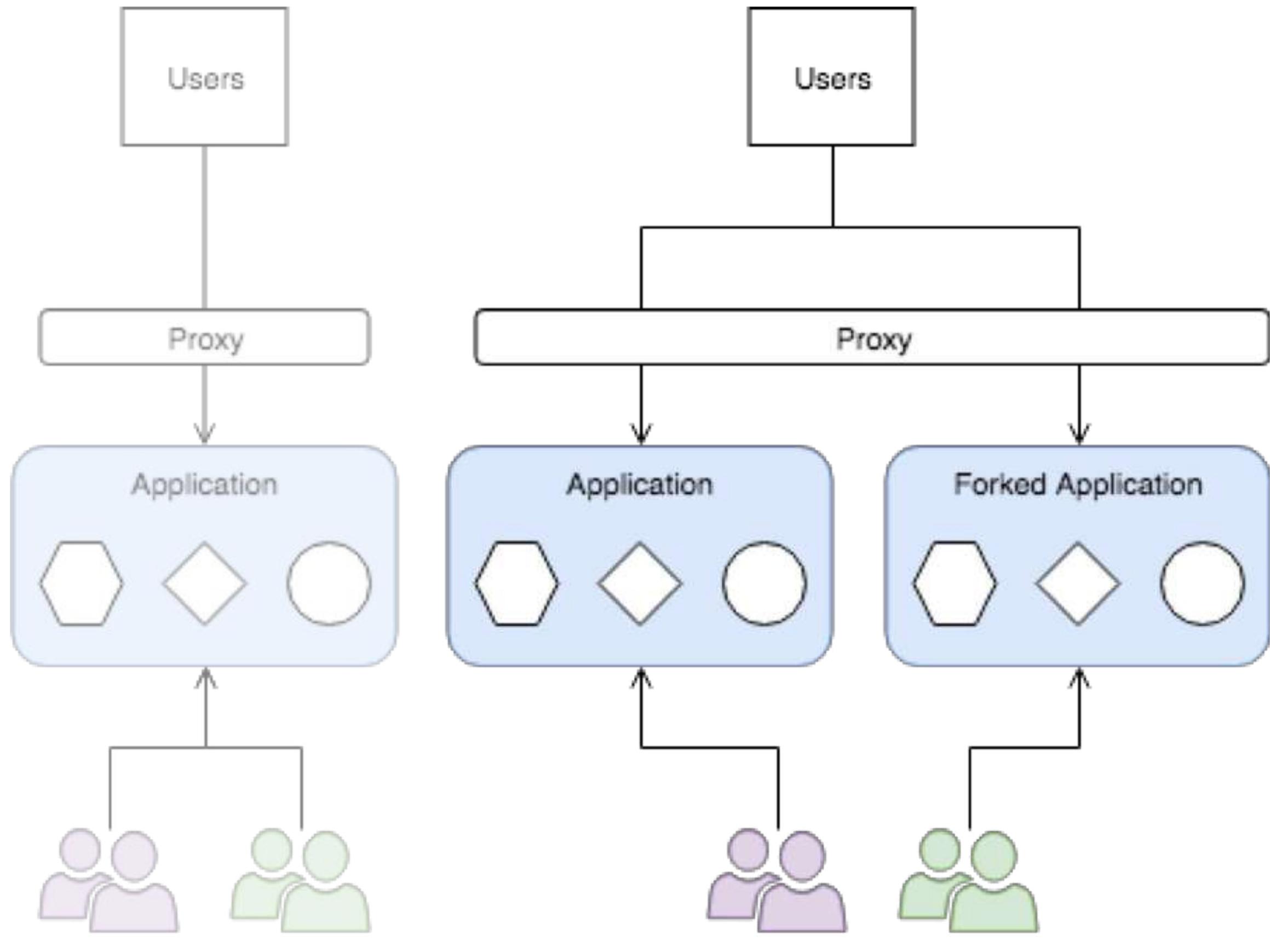
<https://www.thoughtworks.com/profiles/fausto-de-la-torre>

tactical forking



**initial
state**

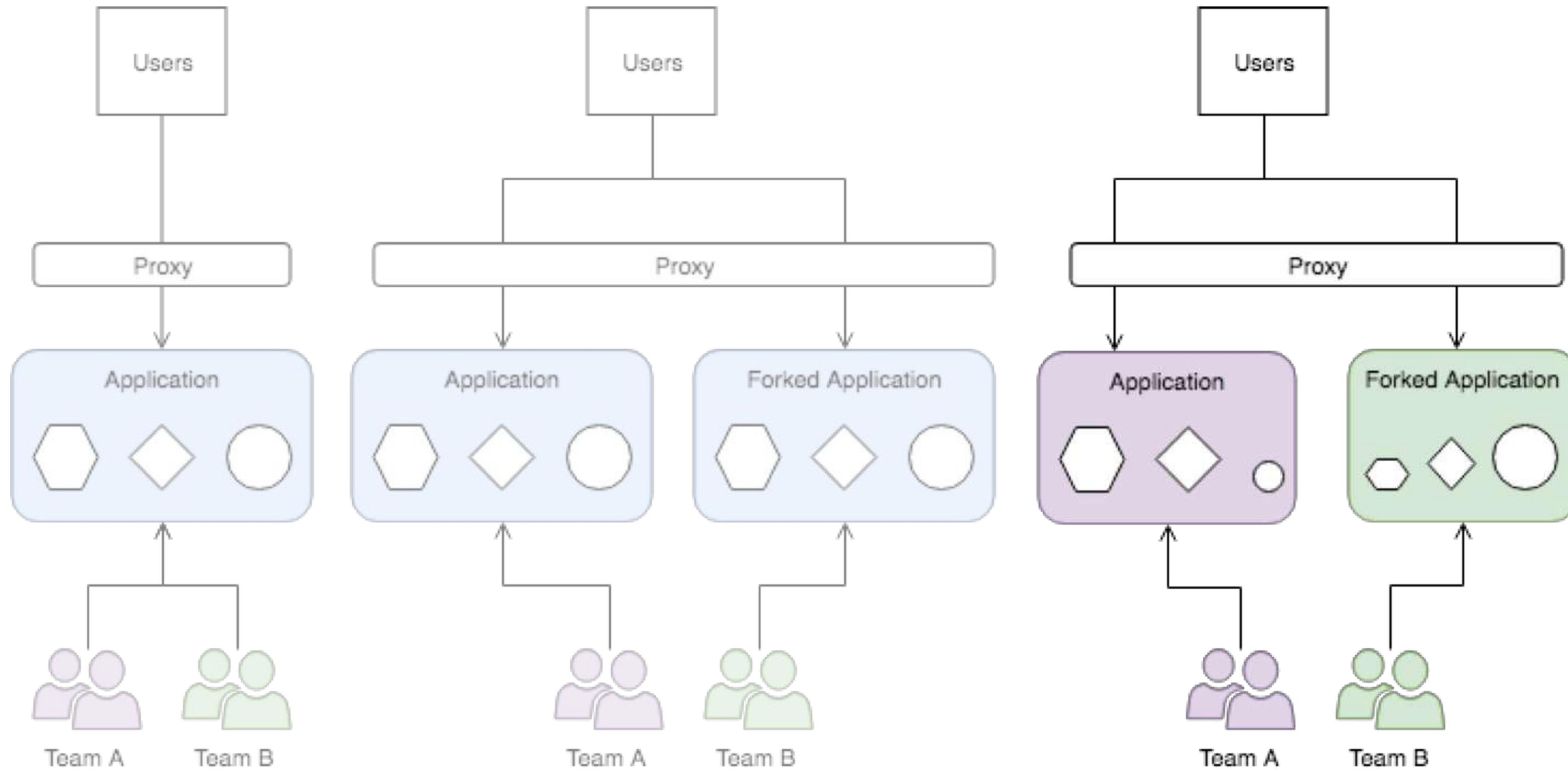
tactical forking



initial
state

**first step
benefit is tangible**

tactical forking

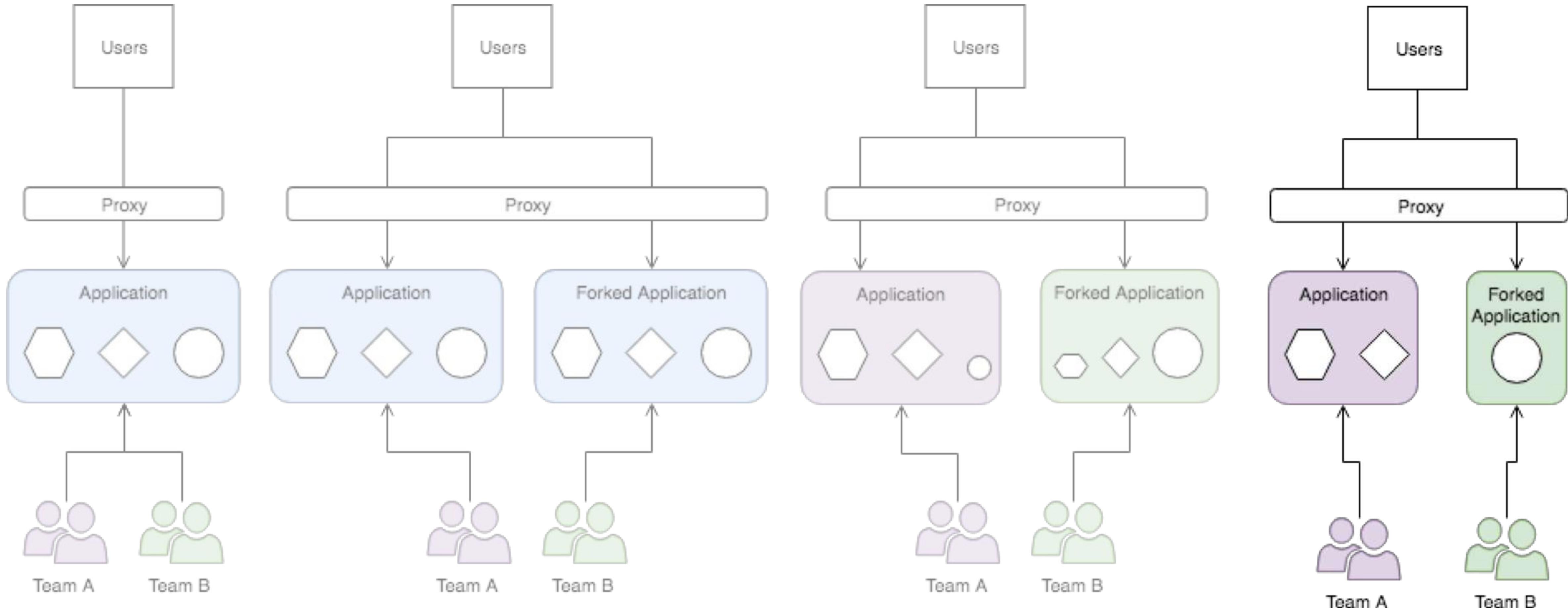


initial
state

first step
benefit is tangible

constant
refactor

tactical forking



initial
state

first step
benefit is tangible

constant
refactor

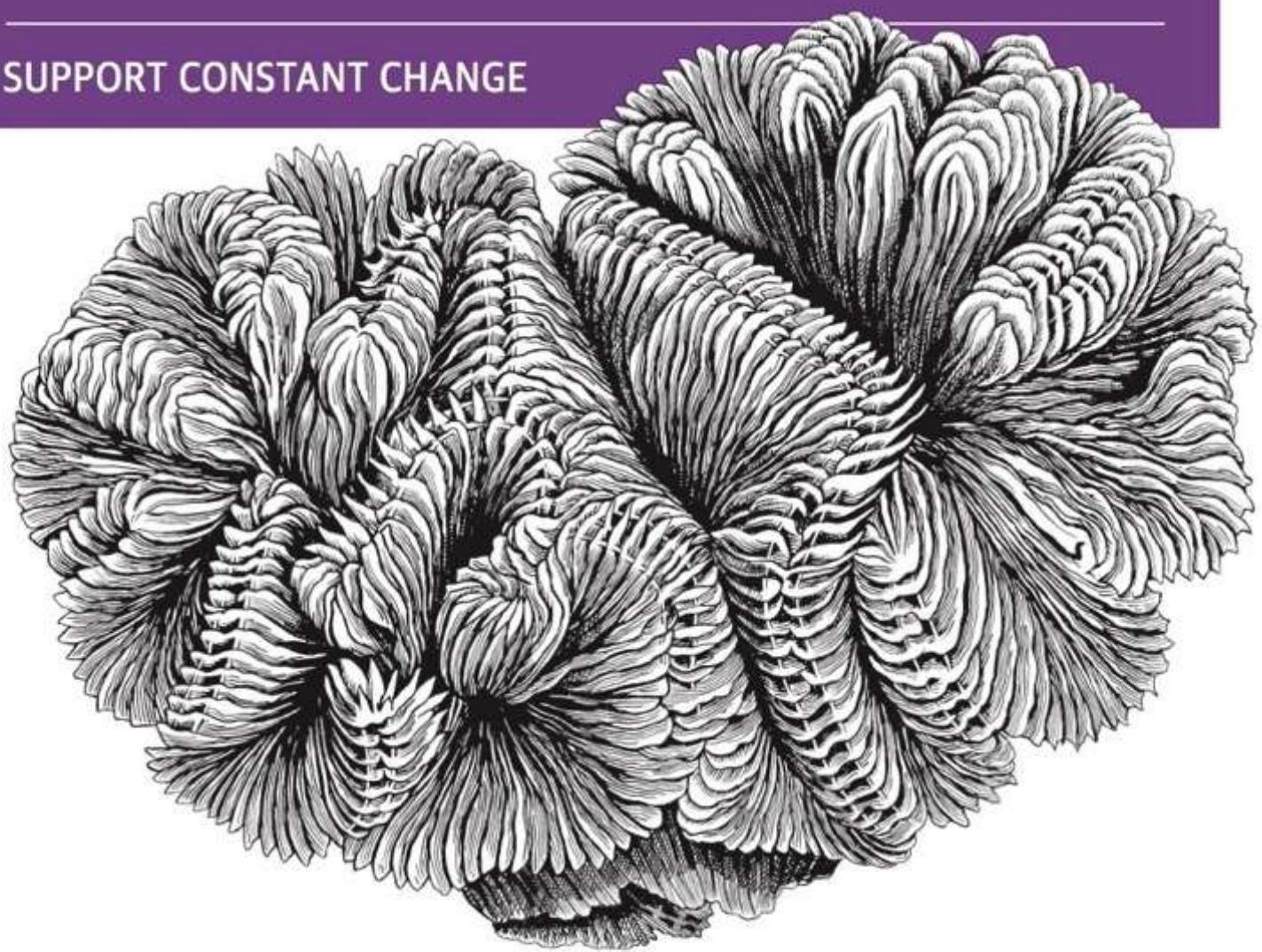
desired state

aRChiTeCtuRe
fOr
cONTiNuOUs DeLiVeRy

O'REILLY®

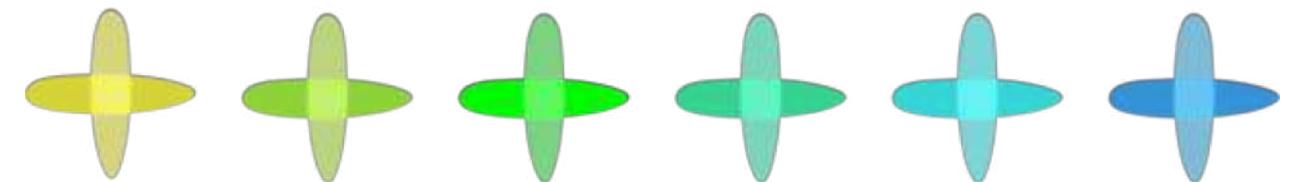
Building Evolutionary Architectures

SUPPORT CONSTANT CHANGE

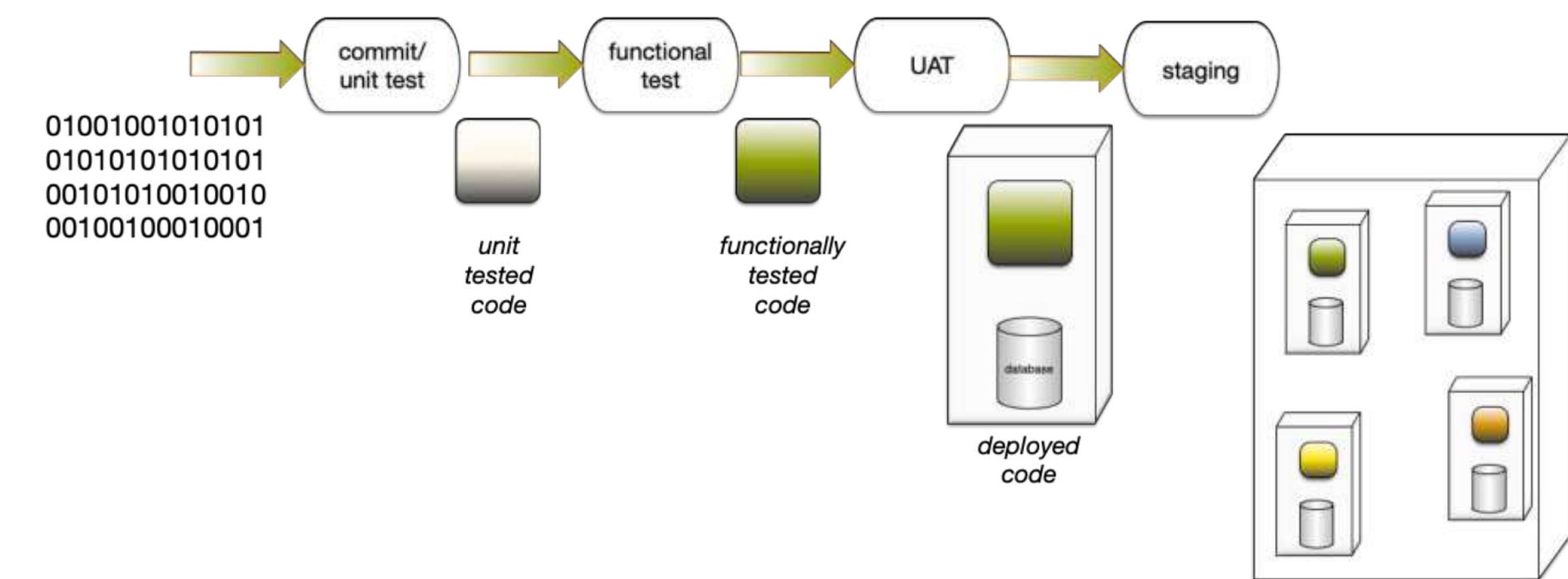
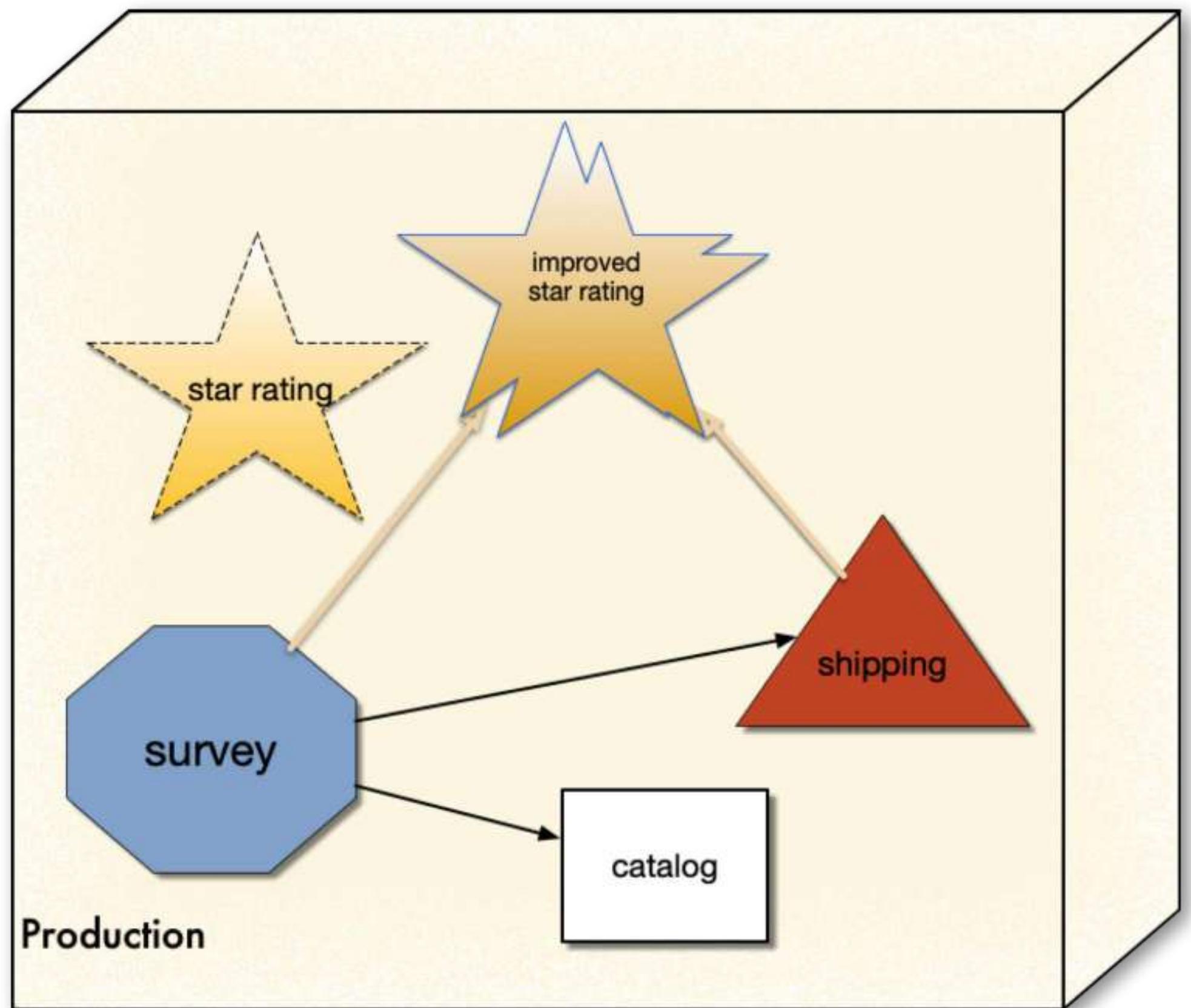


Neal Ford, Rebecca Parsons & Patrick Kua

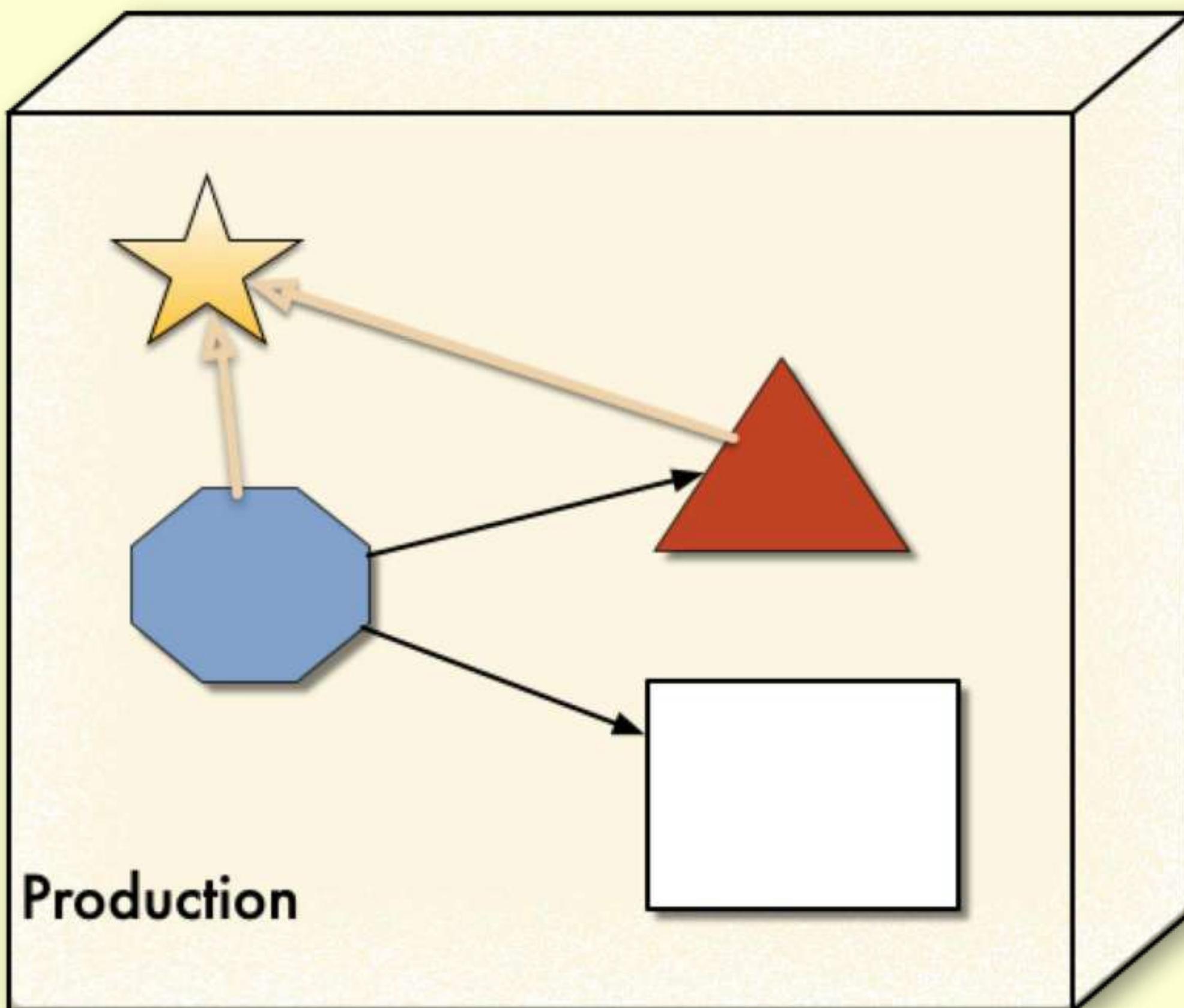
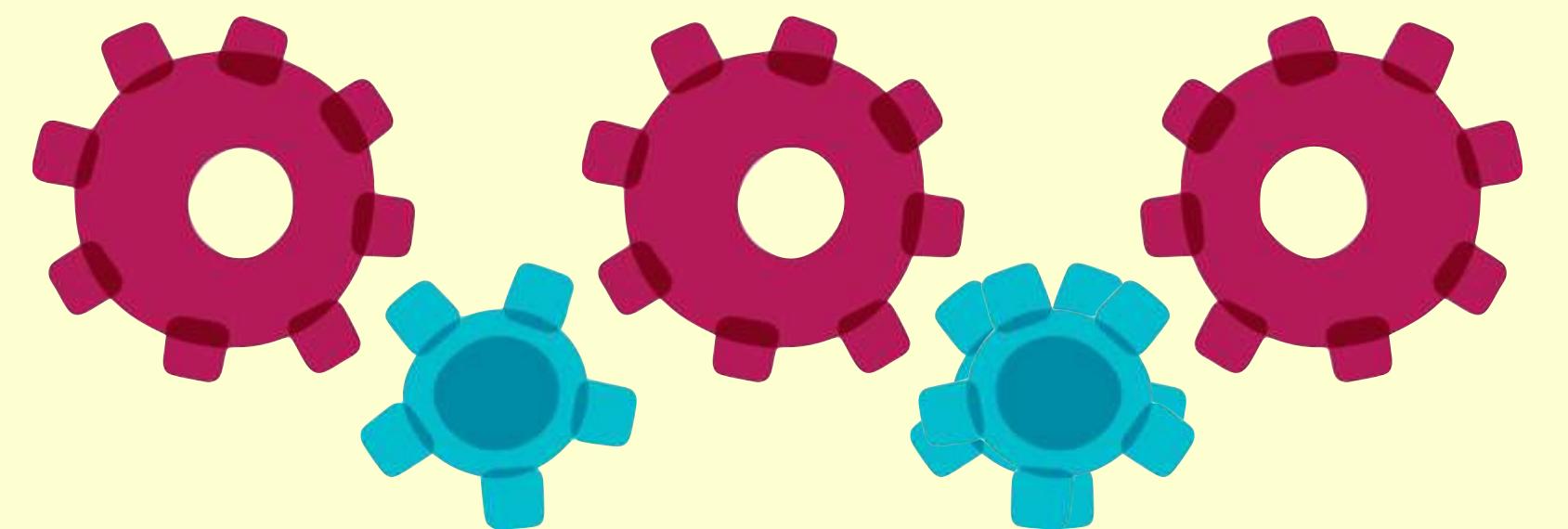




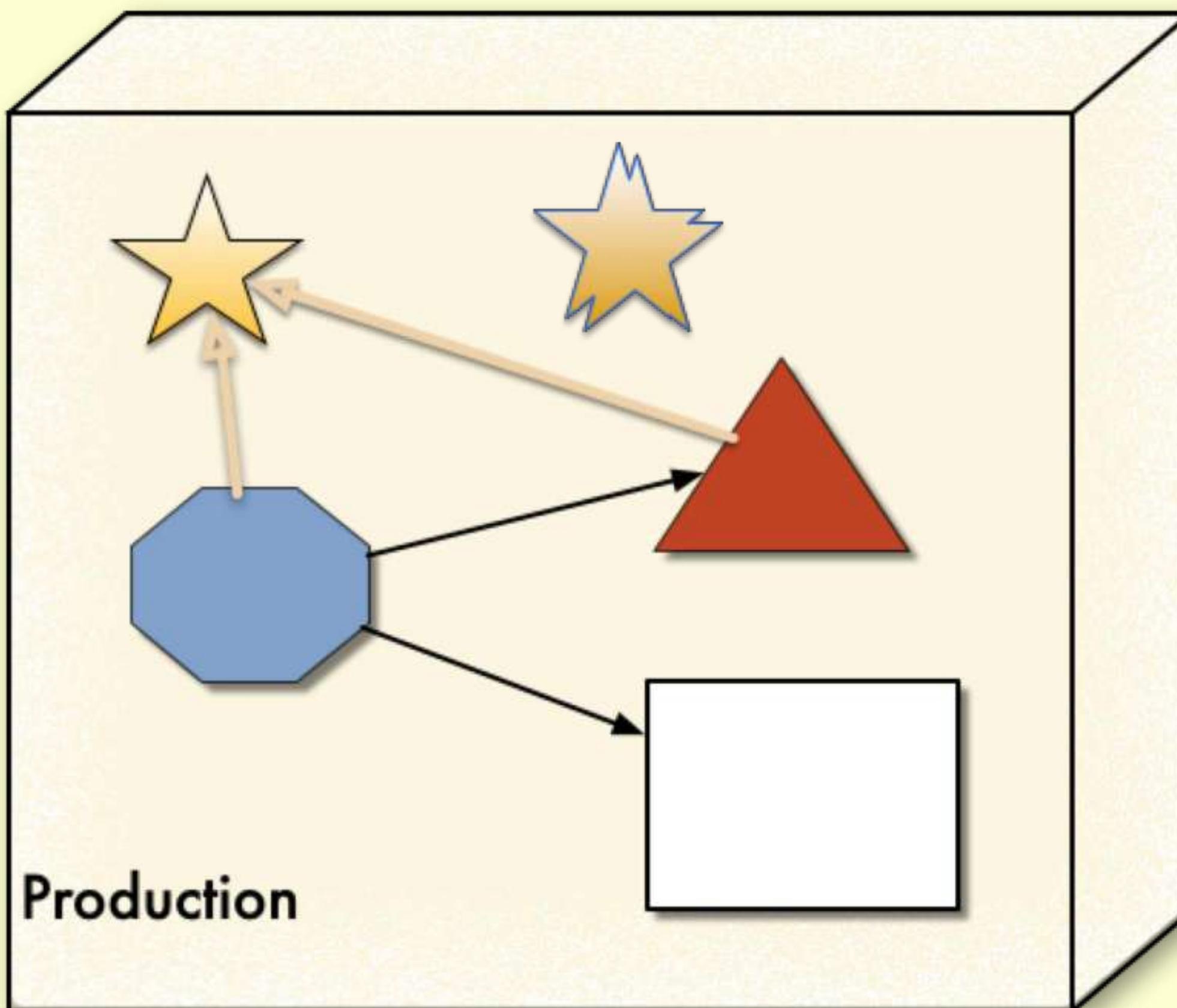
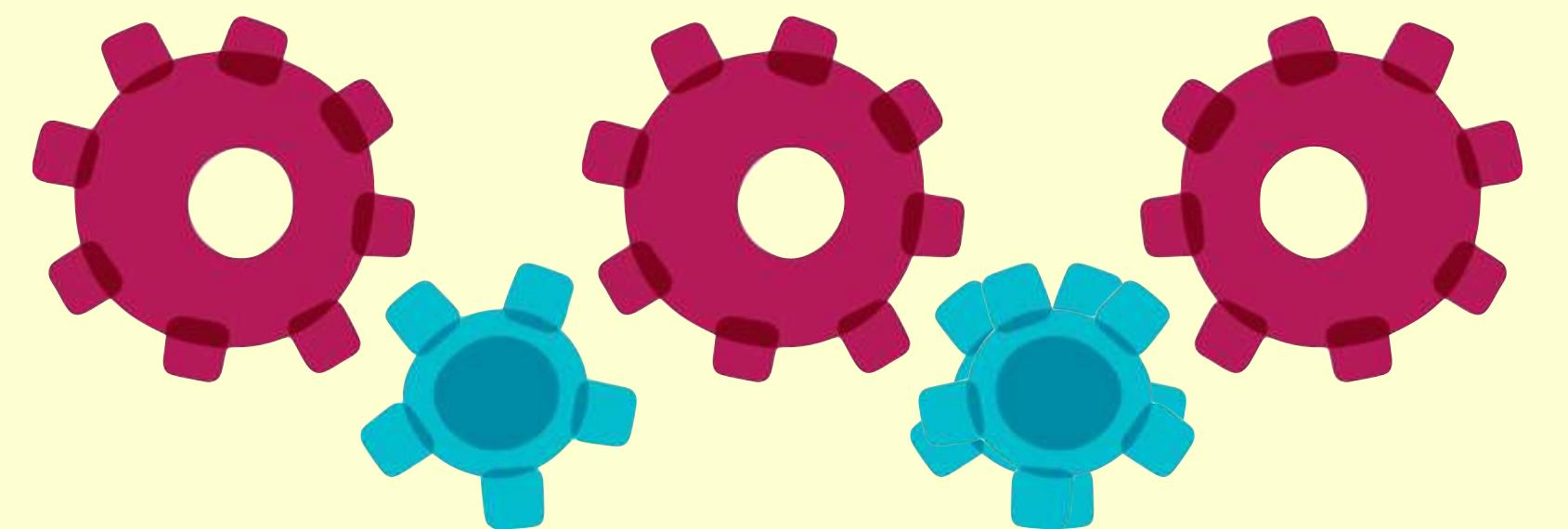
incremental



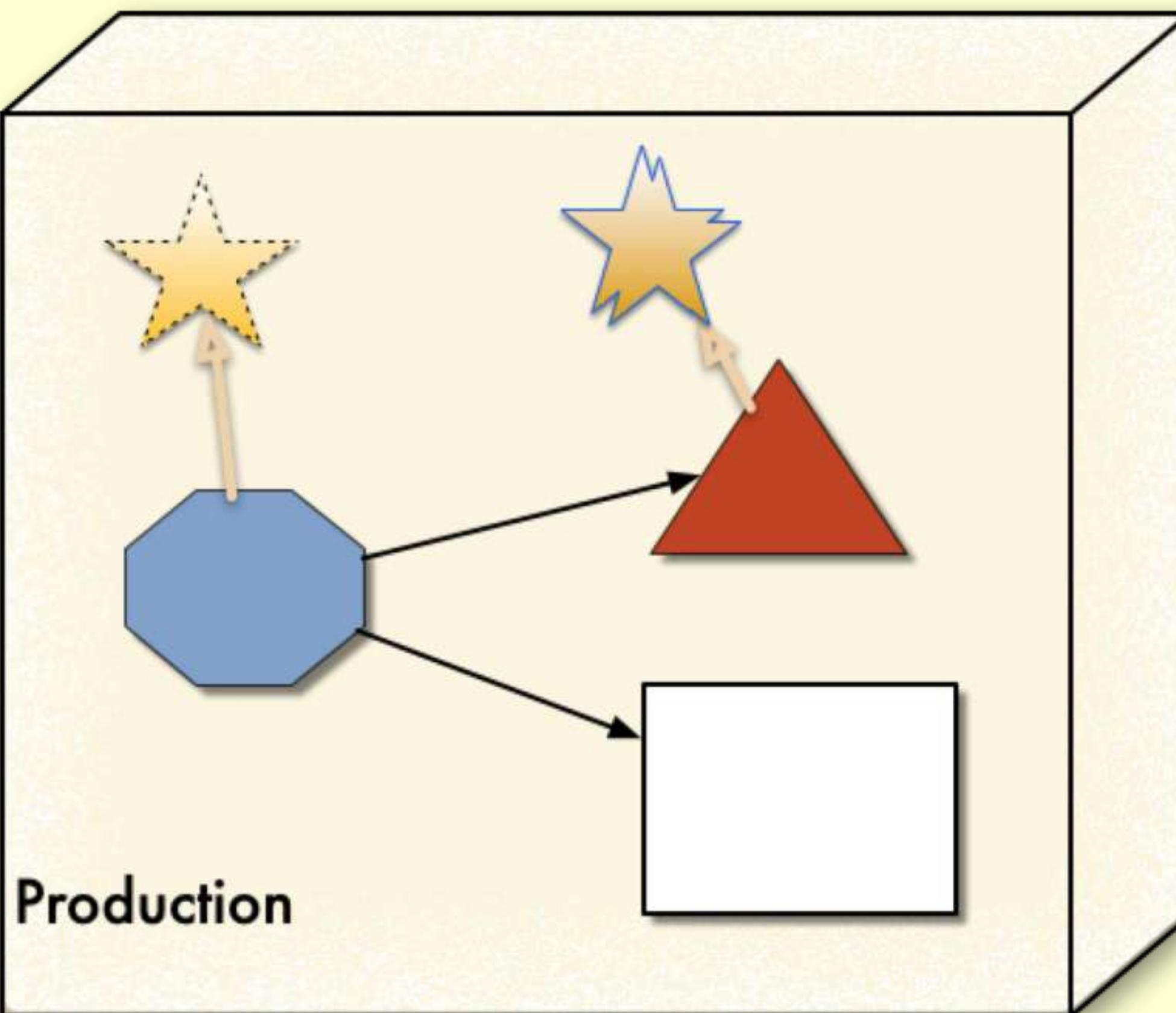
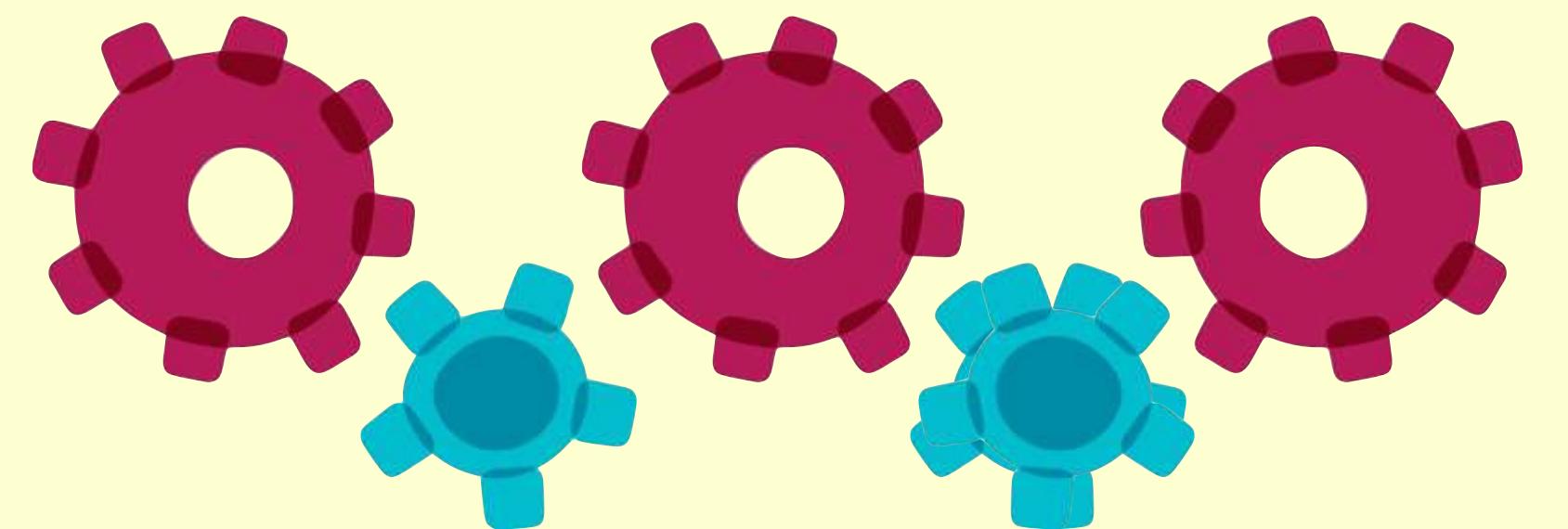
Penultima ↑ e



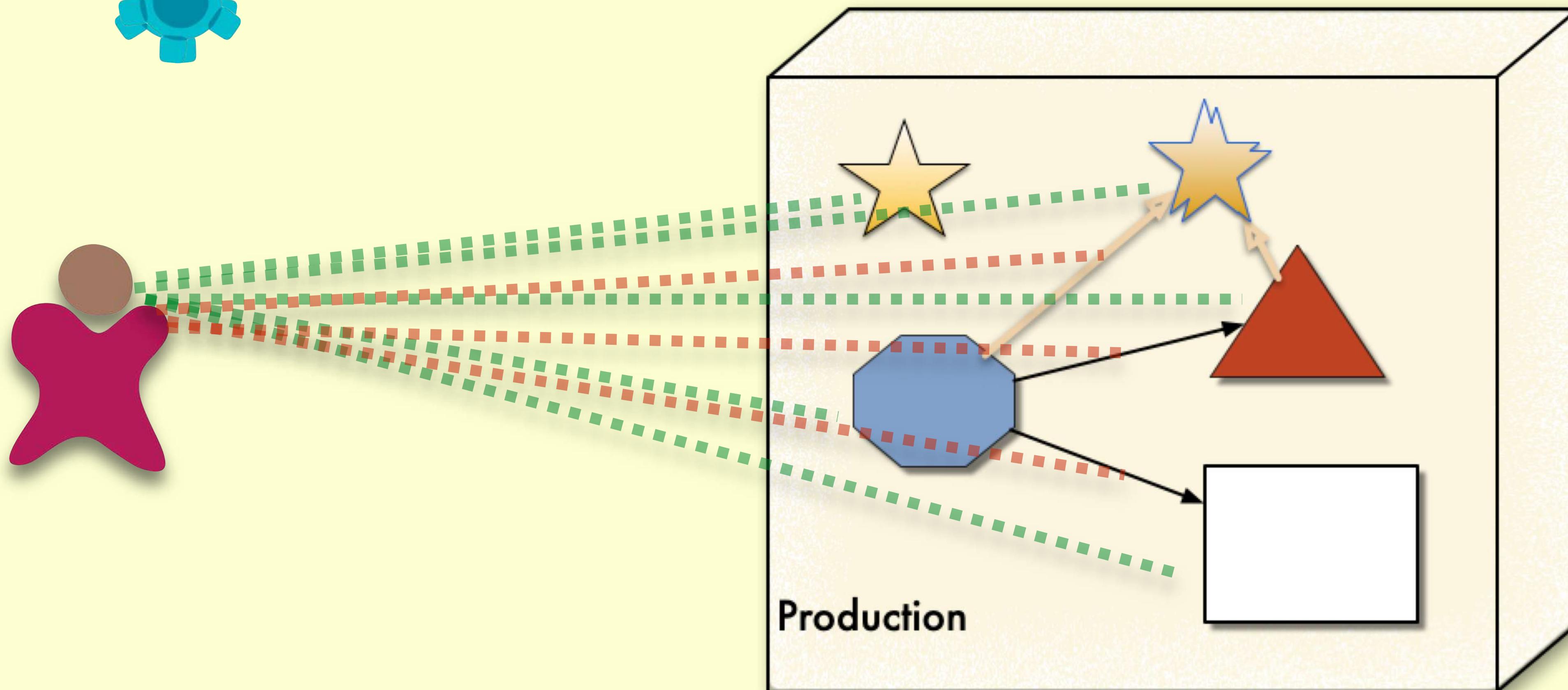
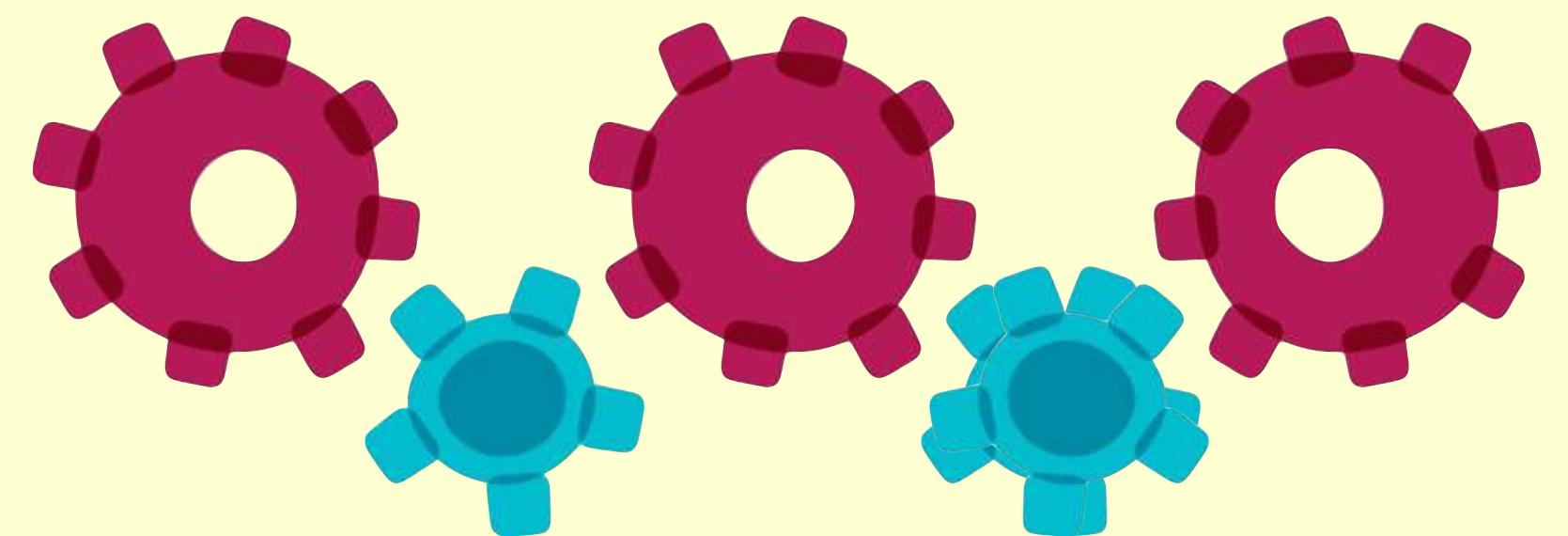
Penultima ↑ e



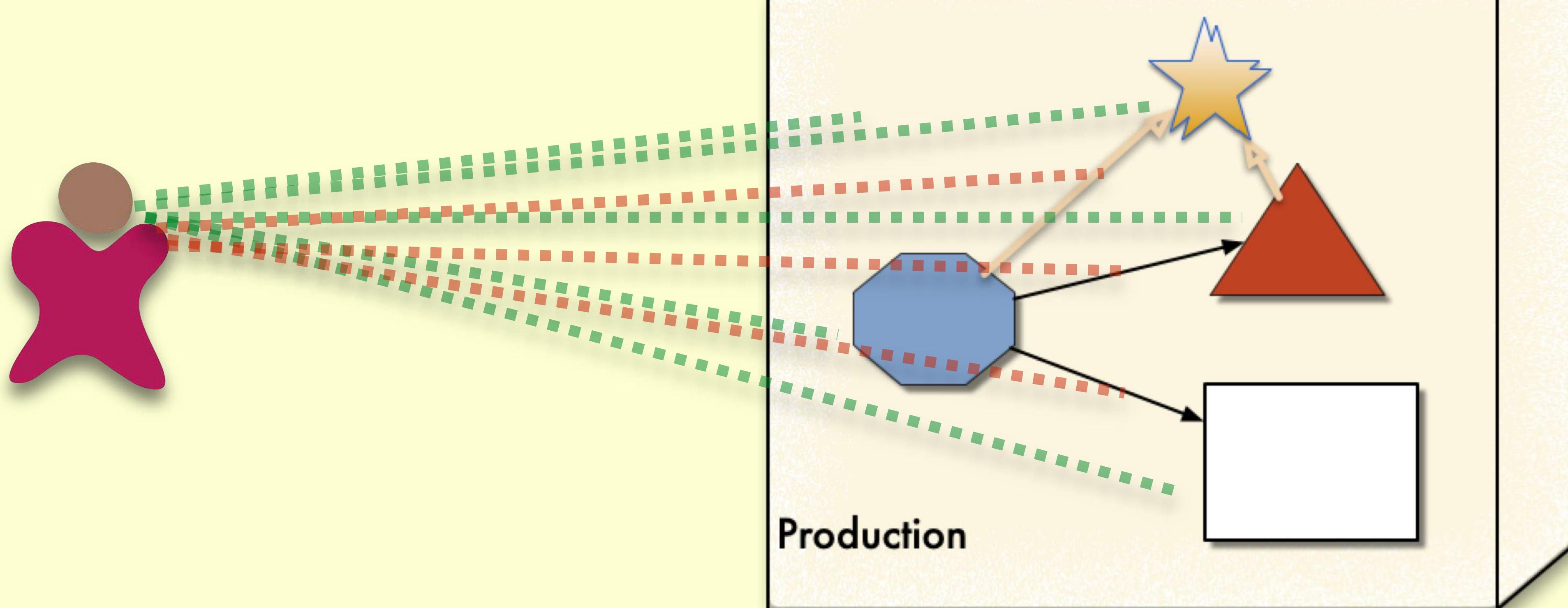
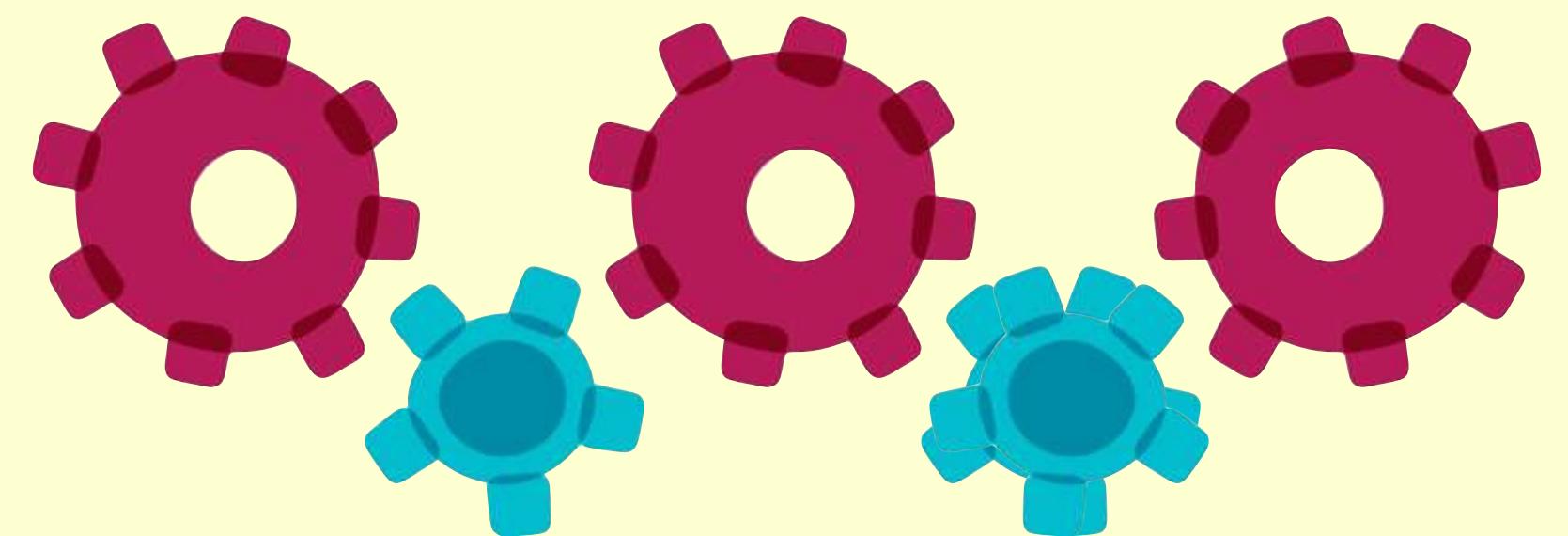
Penultima ↑ e



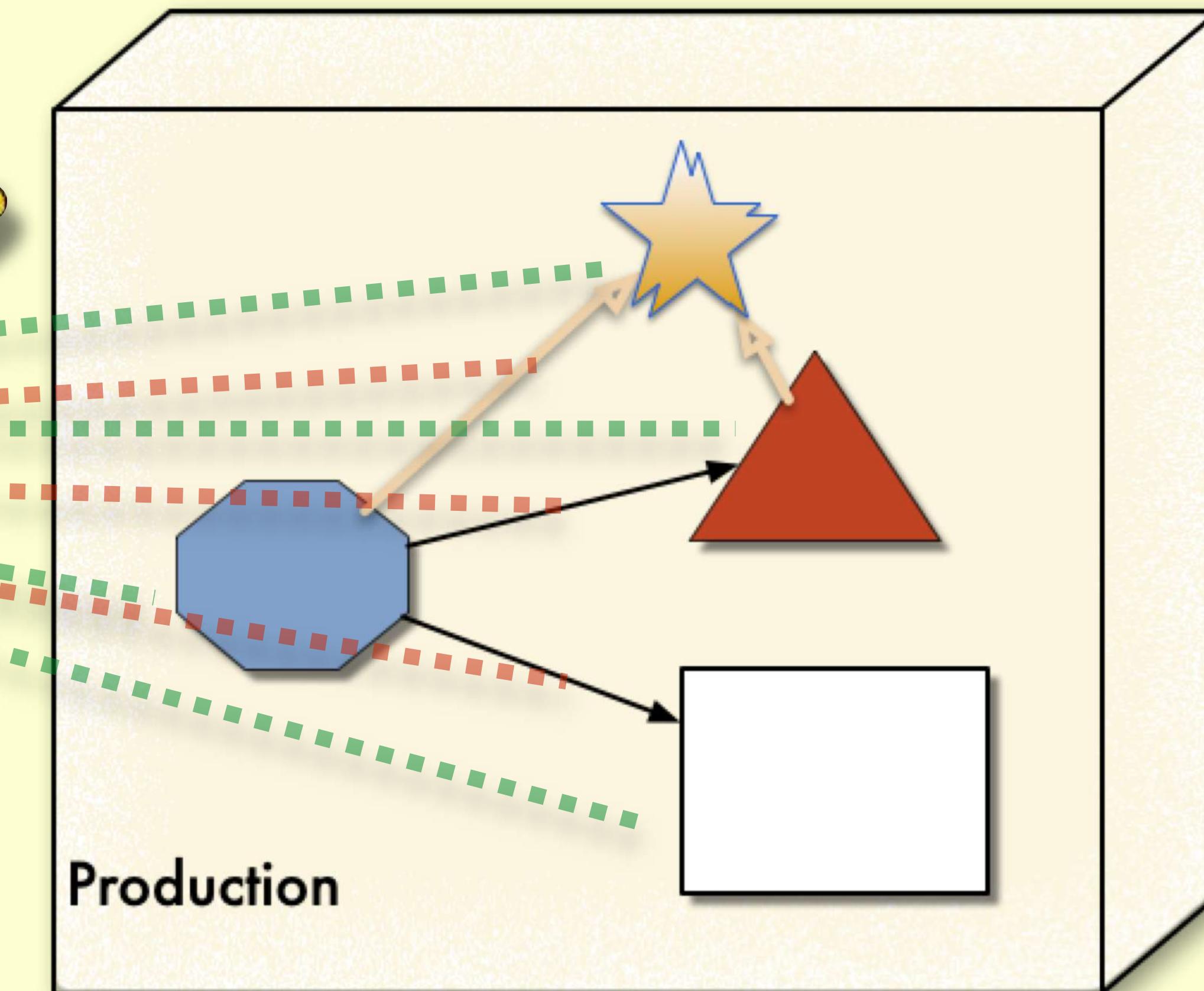
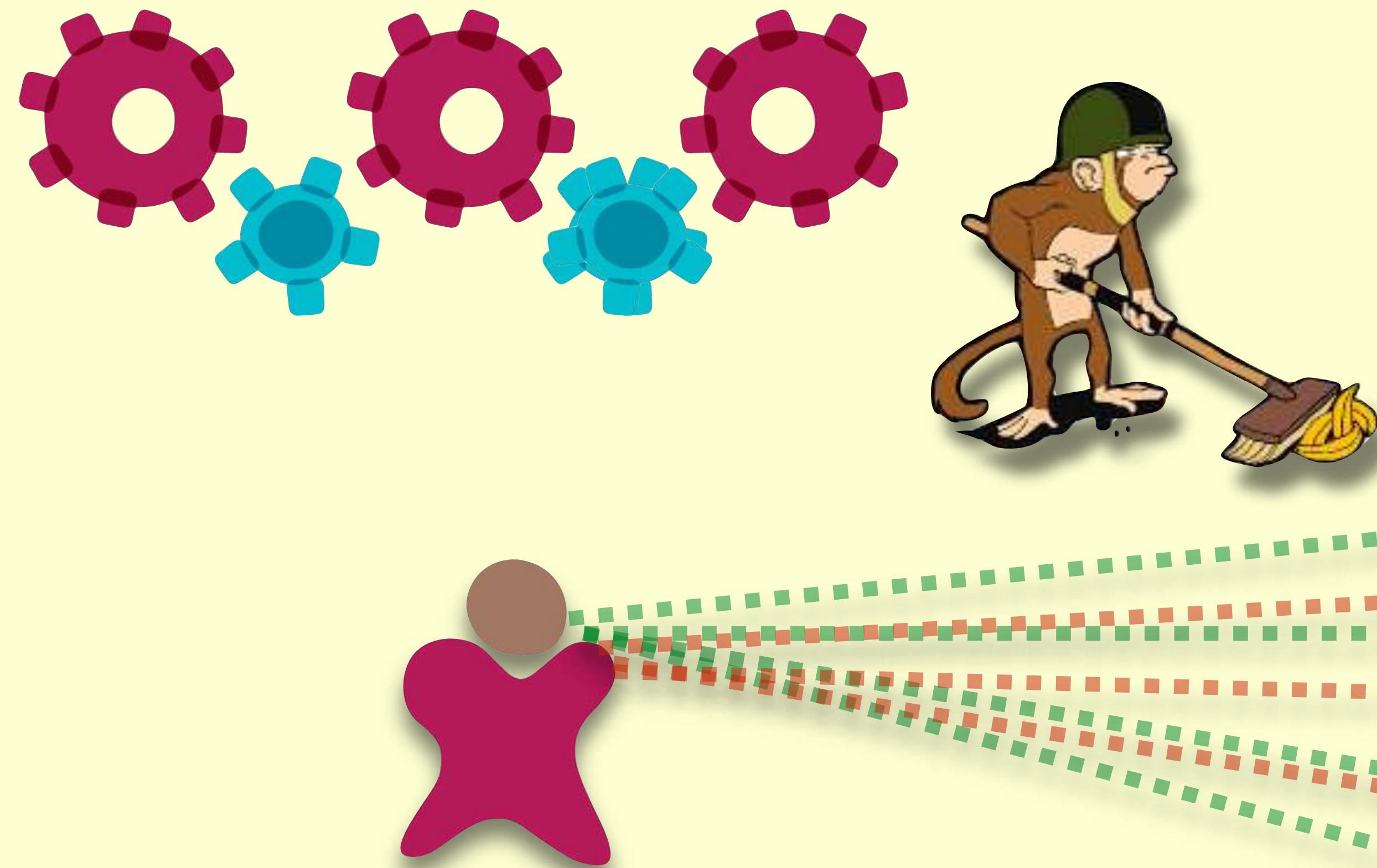
Penultima ↑ e



Penultima ↑ e



Penultima ↑ e



The screenshot shows a web browser window displaying a blog post from GitHub Engineering. The title of the post is "Move Fast and Fix Things". The author is listed as "vmg" and the date is "December 15, 2015". The post discusses technical debt and how GitHub handles it. It includes a section titled "Merges in Git" which explains the storage model of GitHub repositories and the challenges of performing merges in a production environment.

Move Fast and Fix Things

vmg December 15, 2015

Anyone who has worked on a large enough codebase knows that technical debt is an inescapable reality: The more rapidly an application grows in size and complexity, the more technical debt is accrued. With GitHub's growth over the last 7 years, we have found plenty of nooks and crannies in our codebase that are inevitably below our very best engineering standards. But we've also found effective and efficient ways of paying down that technical debt, even in the most active parts of our systems.

At GitHub we try not to brag about the "shortcuts" we've taken over the years to scale our web application to more than 12 million users. In fact, we do quite the opposite: we make a conscious effort to study our codebase looking for systems that can be rewritten to be cleaner, simpler and more efficient, and we develop tools and workflows that allow us to perform these rewrites efficiently and reliably.

As an example, two weeks ago we replaced one of the most critical code paths in our infrastructure: the code that performs merges when you press the Merge Button in a Pull Request. Although we routinely perform these kind of refactorings throughout our web app, the importance of the merge code makes it an interesting story to demonstrate our workflow.

Merges in Git

We've talked at length in the past about the storage model that GitHub uses for repositories in our platform and our Enterprise offerings. There are many implementation details that make this model efficient in both performance and disk usage, but the most relevant one here is the fact that repositories are always stored "bare".

This means that the actual files in the repository (the ones that you would see on your working directory when you clone the repository) are not actually available on disk in our infrastructure: they are compressed and delta'ed inside [packfiles](#).

Because of this, performing a merge in a production environment is a nontrivial endeavour. Git knows several [merge strategies](#), but the recursive merge strategy that you'd get by default when using `git merge` to merge two branches in a local repository assumes the existence of a working tree for the repository, with all the files checked out on it.

The workaround we developed in the early days of GitHub for this limitation is effective, but not particularly elegant: instead of using the default [git-merge-recursive](#) strategy, we wrote our own merge strategy based on the original one that Git used back in the day: [git-merge-resolve](#). With some tweaking, the old strategy can be adapted to not require an actual checkout of the files on disk.

To accomplish this, we wrote a shell script that sets up a [temporary working directory](#), in

“move
fast
&
fix
things”



```
def create_merge_commit(base, head, author, commit_message)
  base = resolve_commit(base)
  head = resolve_commit(head)
  commit_message = Rugged.prettyify_message(commit_message)

  merge_base = rugged.merge_base(base, head)
  return [nil, "already_merged"] if merge_base == head.oid

  ancestor_tree = merge_base && Rugged::Commit.lookup(rugged, merge_base).tree
  merge_options = {
    :fail_on_conflict => true,
    :skip_reuc => true,
    :no_recursive => true,
  }
  index = base.tree.merge(head.tree, ancestor_tree, merge_options)
  return [nil, "merge_conflict"] if (index.nil? || index.conflicts?)

  options = {
    :message => commit_message,
    :committer => author,
    :author => author,
    :parents => [base, head],
    :tree => index.write_tree(rugged)
  }

  [Rugged::Commit.create(rugged, options), nil]
end
```

```
def create_merge_commit(author, base, head, options = {})
  commit_message = options[:commit_message] || "Merge #{head} into #{base}"
  now = Time.current

  science "create_merge_commit" do |e|
    e.context :base => base.to_s, :head => head.to_s, :repo => repository.nwo
    e.use { create_merge_commit_git(author, now, base, head, commit_message) }
    e.try { create_merge_commit_rugged(author, now, base, head, commit_message) }
  end
end
```

A screenshot of the GitHub repository 'github/scientist'. The top part shows the repository overview with 71 commits, 1 branch, 6 releases, and 16 contributors. Below this is a commit list with details like author, date, and message. The bottom part shows the 'README.md' file content. It starts with a heading 'Scientist!', followed by a sub-section 'How do I science?'. It explains how to use the library to handle permissions in a web app. It includes a code snippet showing how to wrap a 'use' block around the original behavior and a 'try' block around the new behavior. A note at the bottom states: 'Wrap a use block around the code's original behavior, and wrap try around the new behavior. experiment.run will always return whatever the use block returns, but it does a bunch of stuff behind the scenes.'

<https://github.com/github/scientist>



```
require "scientist"

class MyWidget
  def allows?(user)
    experiment = Scientist::Default.new "widget-permissions"
    experiment.use { model.check_user?(user).valid? } # old way
    experiment.try { user.can?(:read, model) } # new way

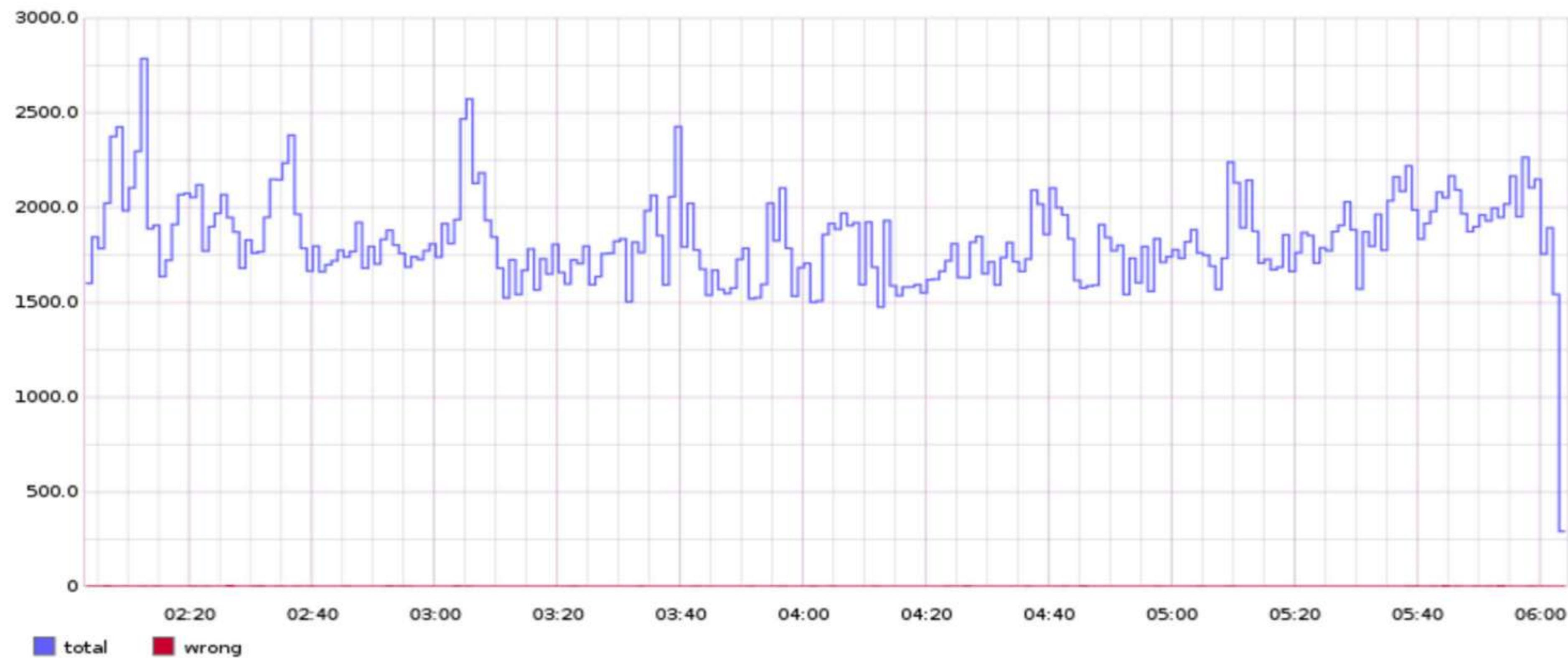
    experiment.run
  end
end
```

- It decides whether or not to run the try block,
- Randomizes the order in which use and try blocks are run,
- Measures the durations of all behaviors,
- Compares the result of try to the result of use,
- Swallows (but records) any exceptions raised in the try block
- Publishes all this information.



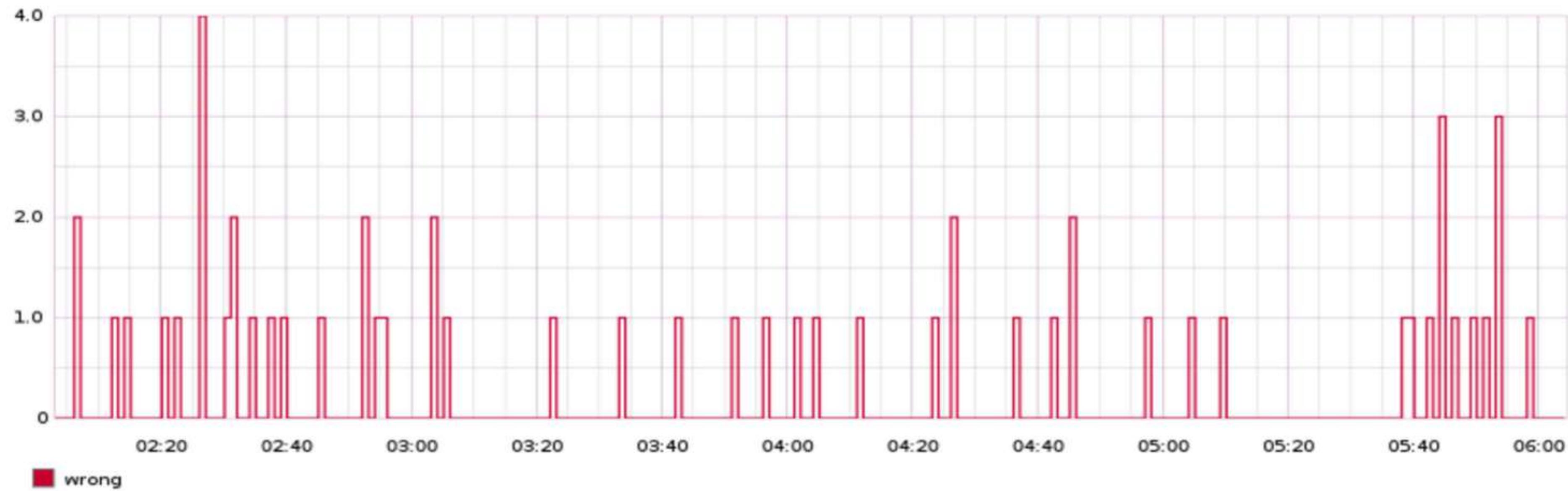
Accuracy

The number of times that the candidate and the control agree or disagree. [View mismatches](#)



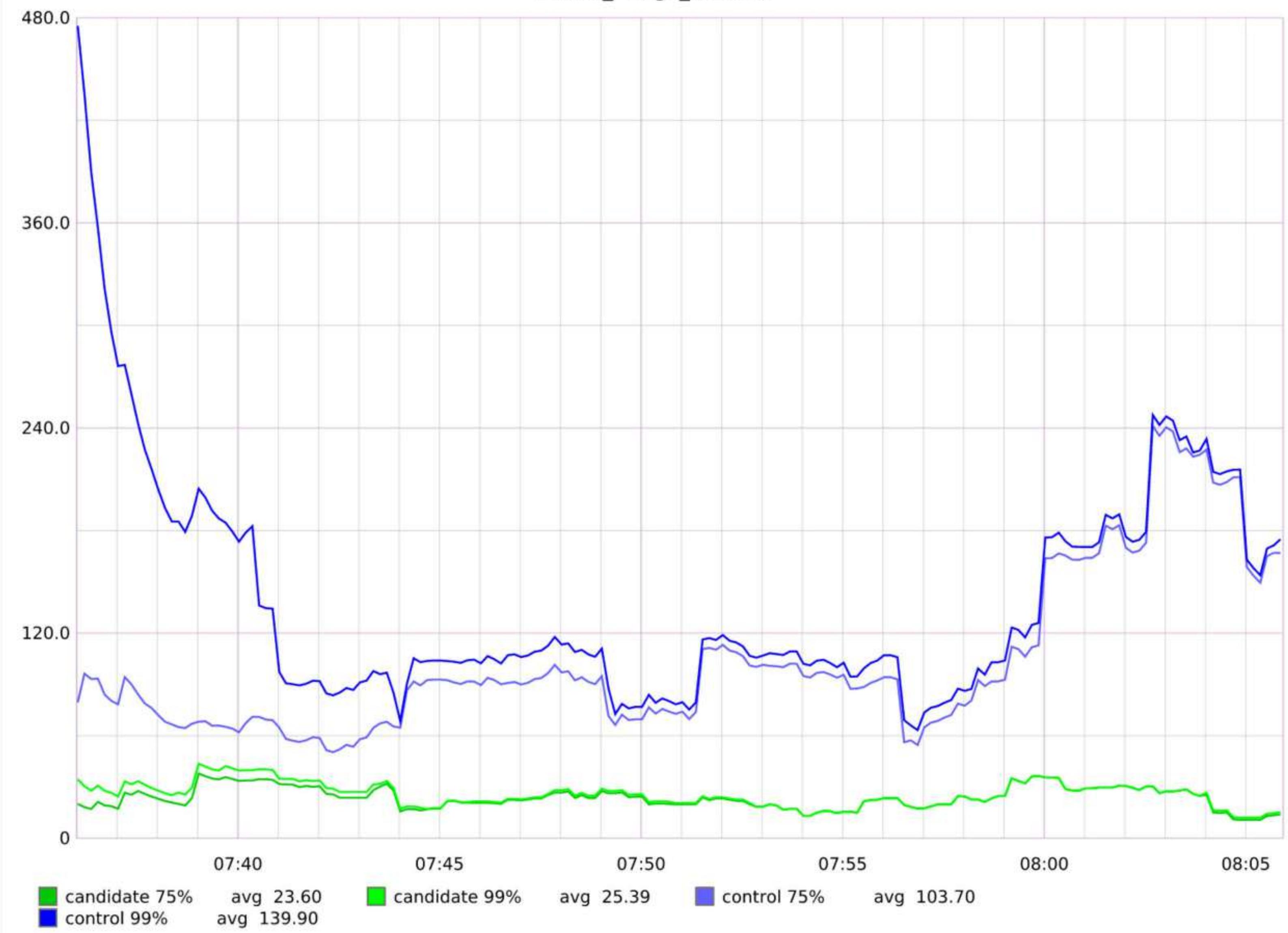


The number of incorrect/ignored only.





create_merge_commit



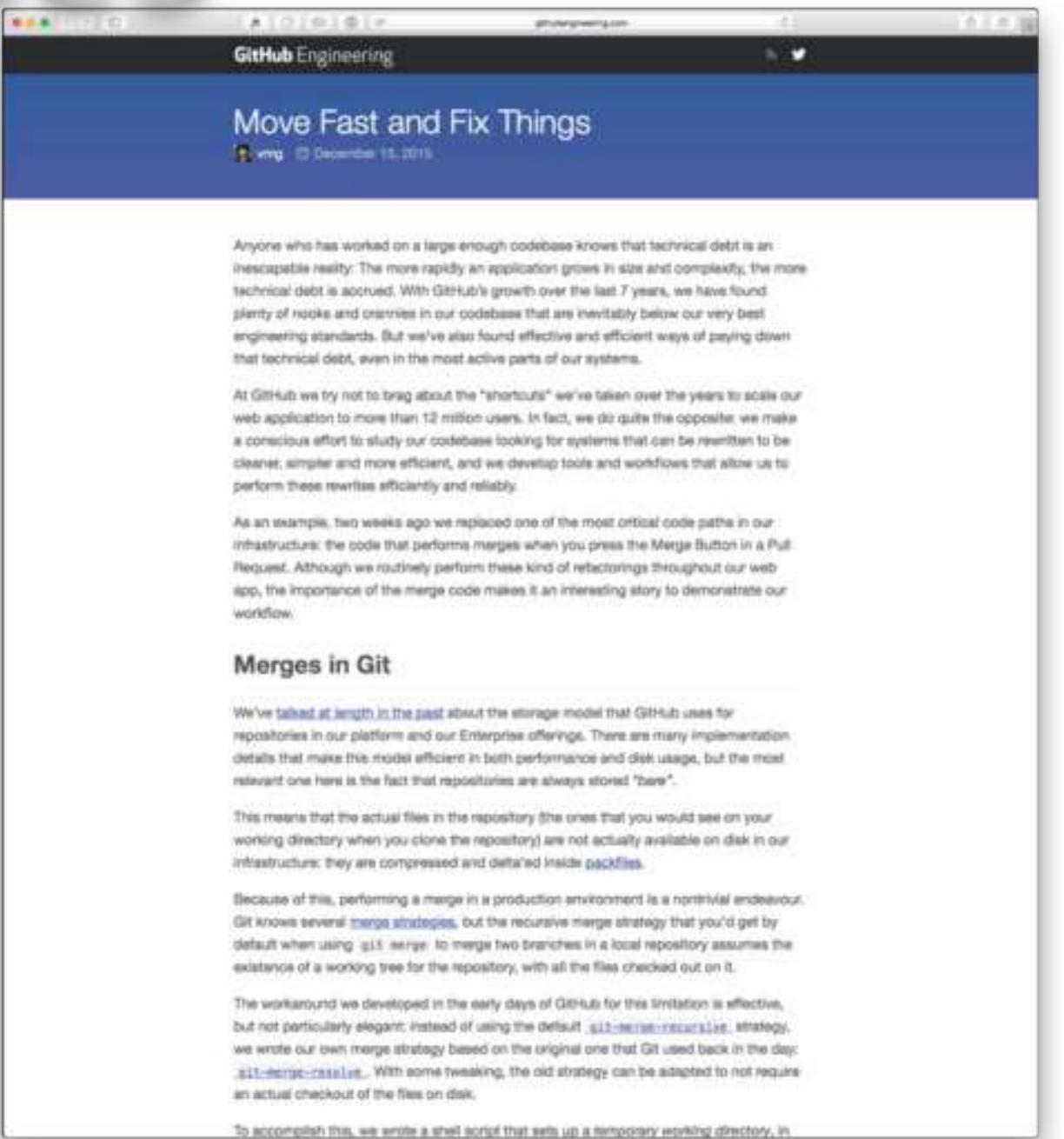
4 days

24 hours/

no mismatches or slow cases

> 10,000,000

comparisons





aRChiTeCtuRe fOr cONTiNuoUs DeLiVeRy



Good engineering practices



aRChiTeCtuRe fOr CONTiNuoUs DeLiVeRy



Appropriate structure.



aRChiTeCtuRe fOr CONTiNuoUs DeLiVeRy



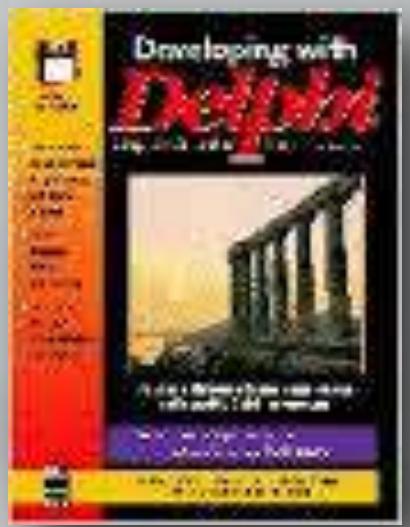
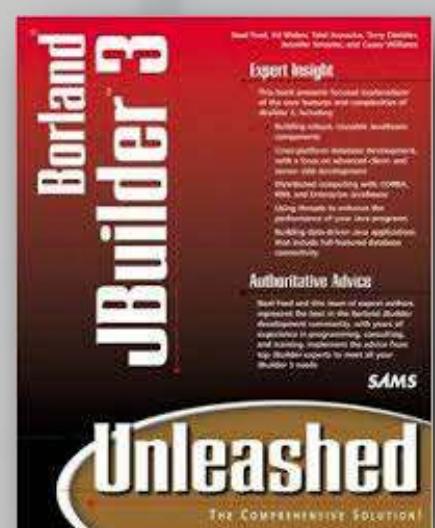
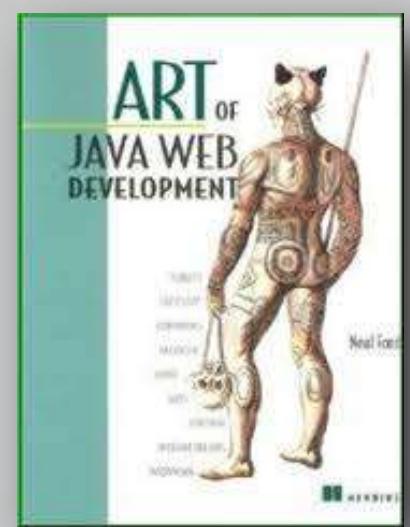
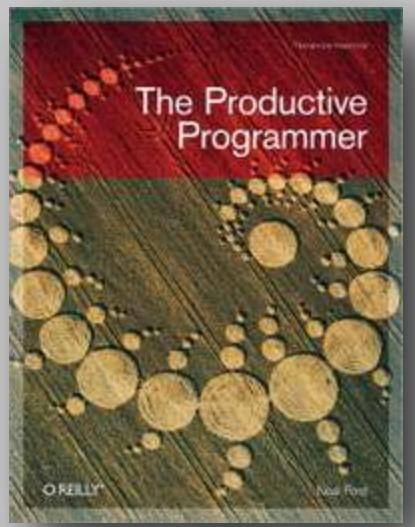
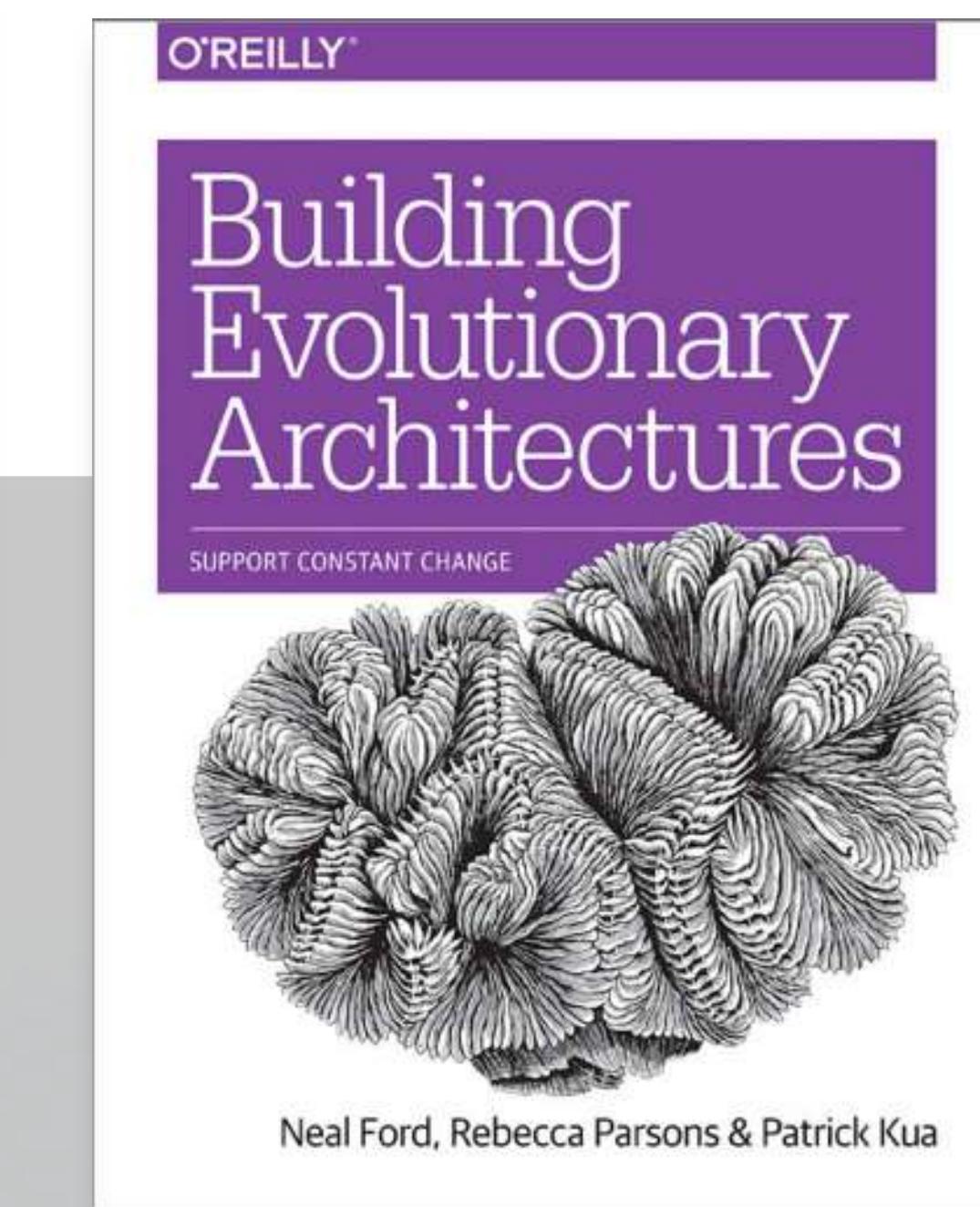
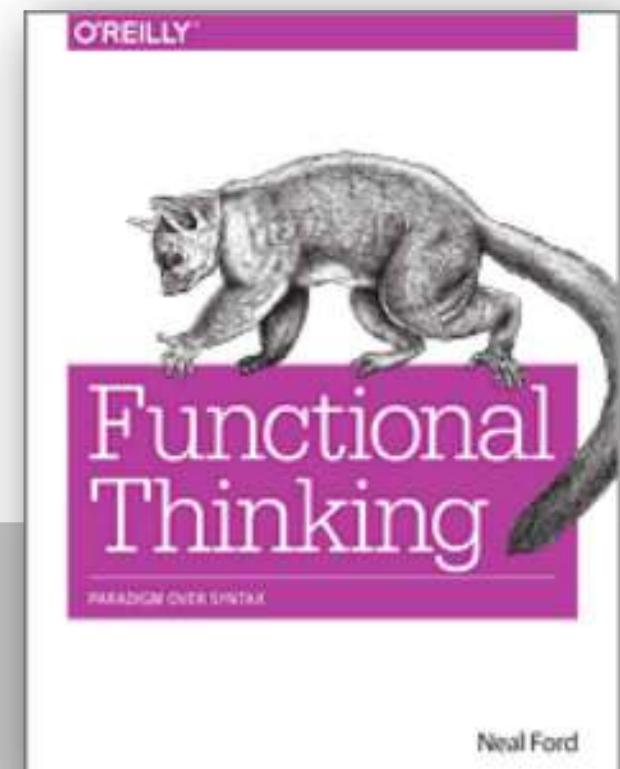
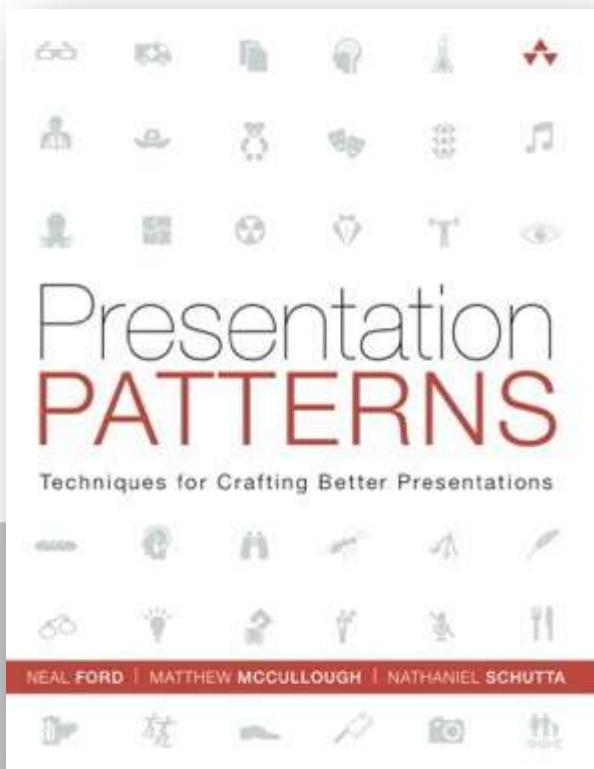
**Apply good engineering practices
to architecture.**



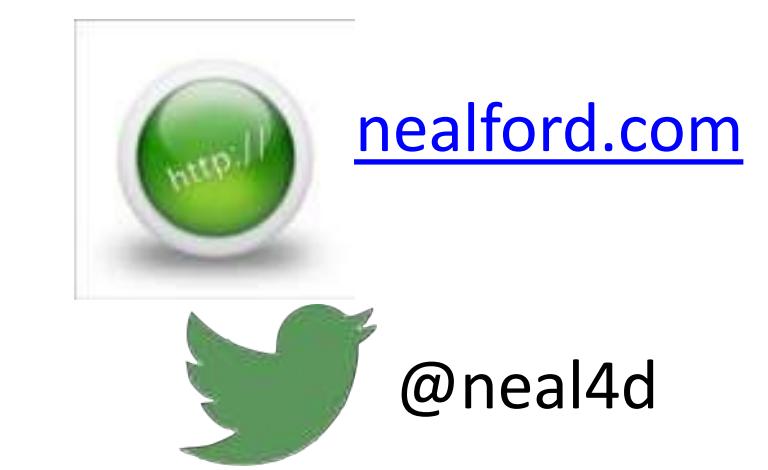
aRChiTeCtuRe fOr CONTiNuoUs DeLiVeRy



**Ability to move fast without
breaking things.**



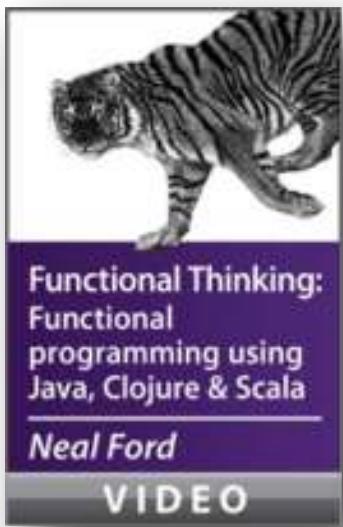
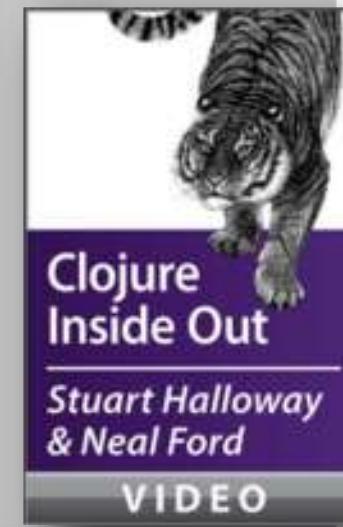
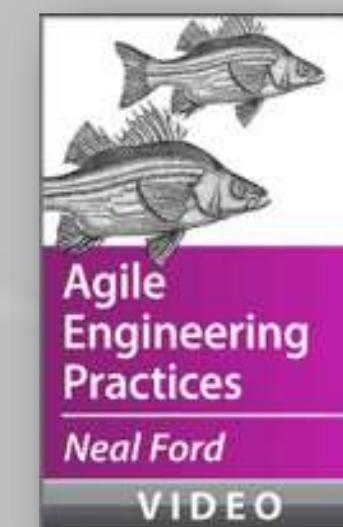
nealford.com/books



ThoughtWorks®
NEAL FORD

Director / Software Architect / Meme Wrangler

nealford.com/videos



www.oreilly.com/software-architecture-video-training-series.html

