



# aRChiTeCtuRe fOr cONTiNuoUs DeLiVeRy



ThoughtWorks®

**NEAL FORD**

*Director / Software Architect / Meme Wrangler*



@neal4d  
[nealford.com](http://nealford.com)

aRChiTeCtuRe  
fOr  
cONTINUOUS DeLiVeRy

# Continuous Delivery

integration as an engineering practice over time

# integration as an engineering practice over time

---

1980

1990

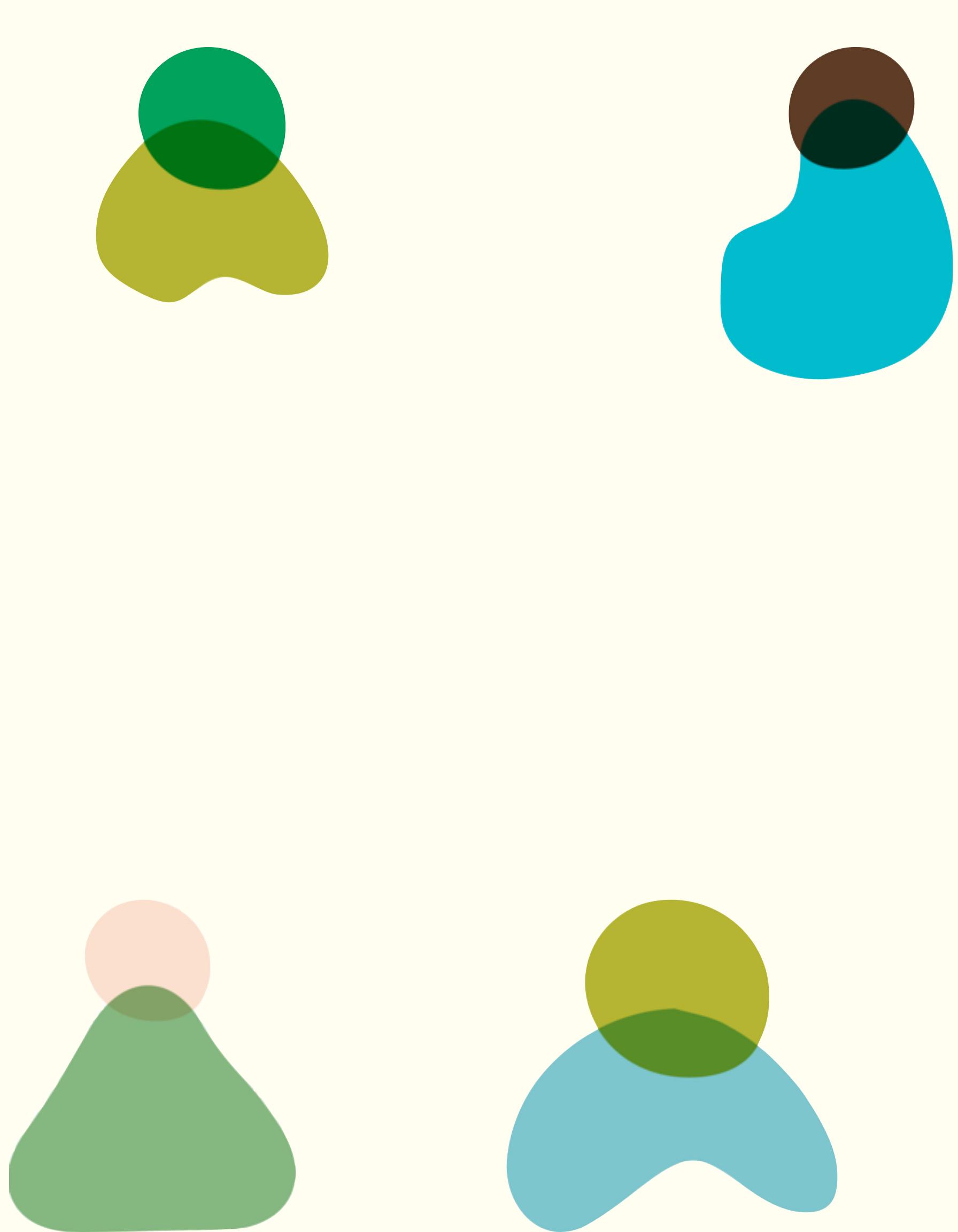
2000

2010

current  
day

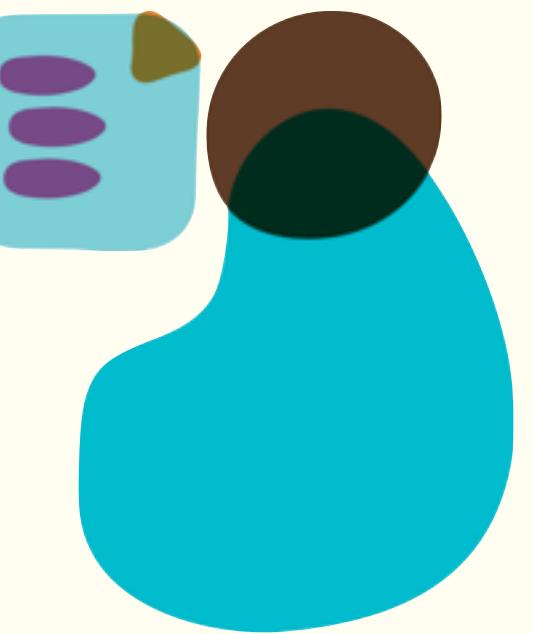
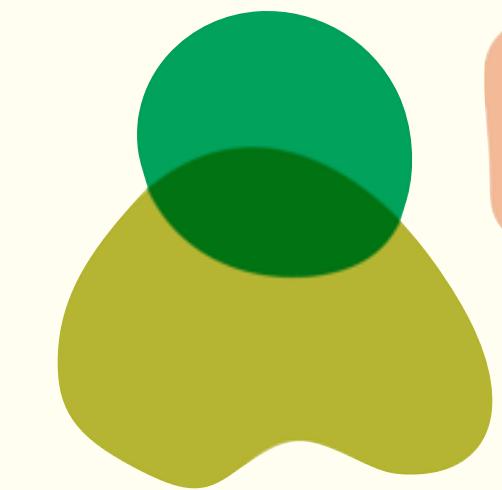
---

1980

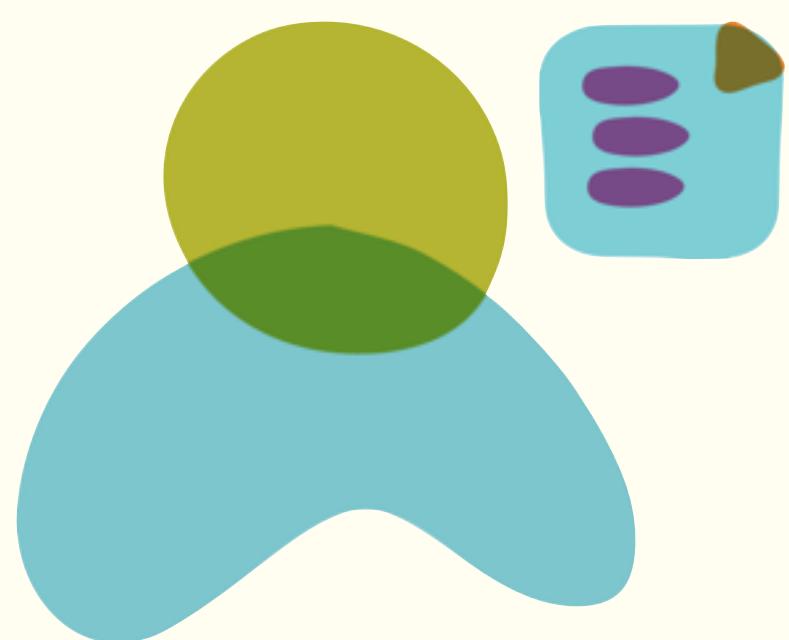
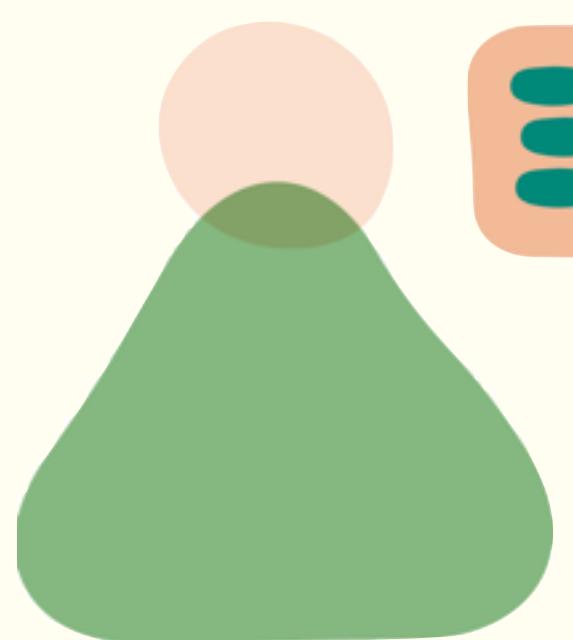


---

1980



|

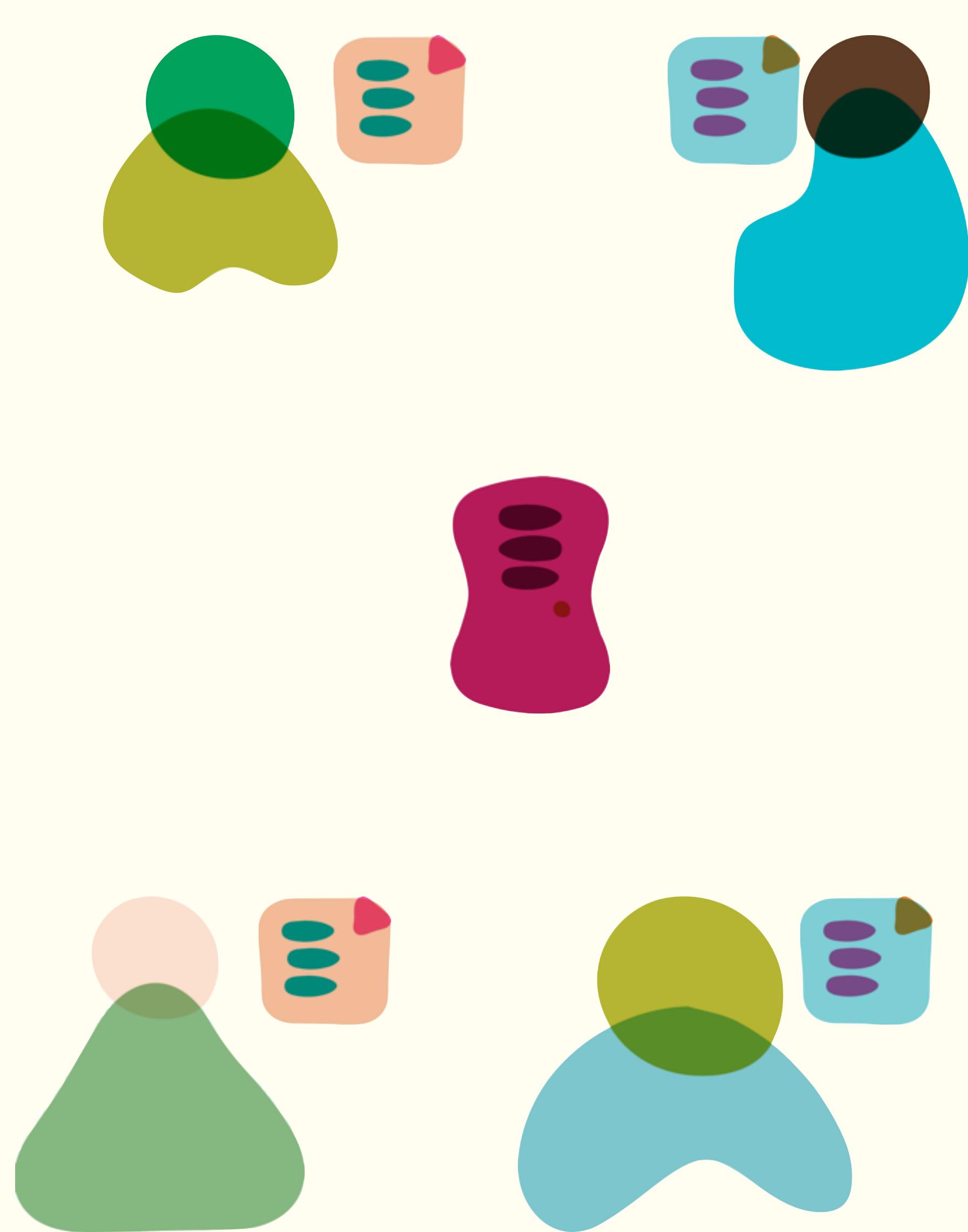


|

---

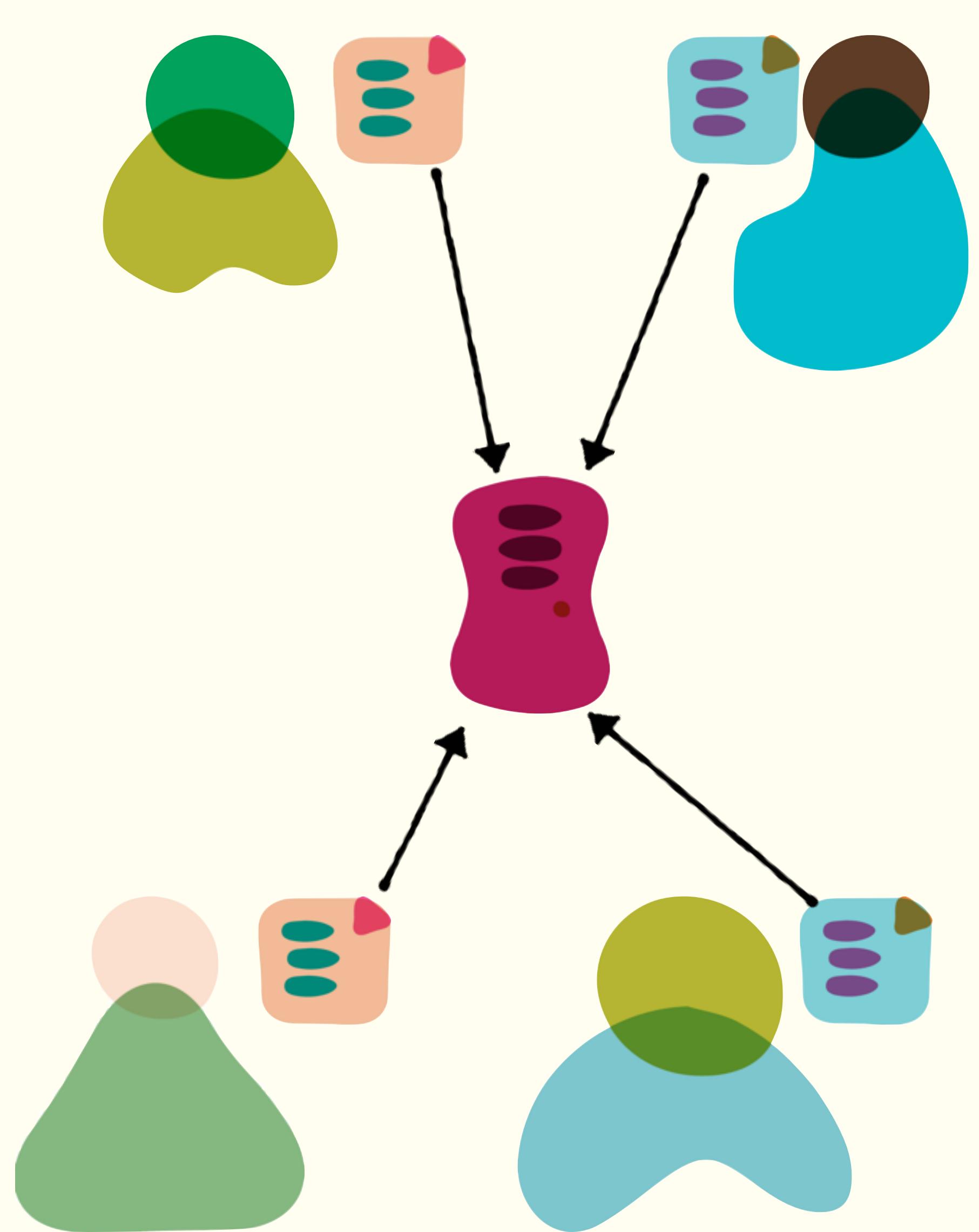
1980

integration  
phase



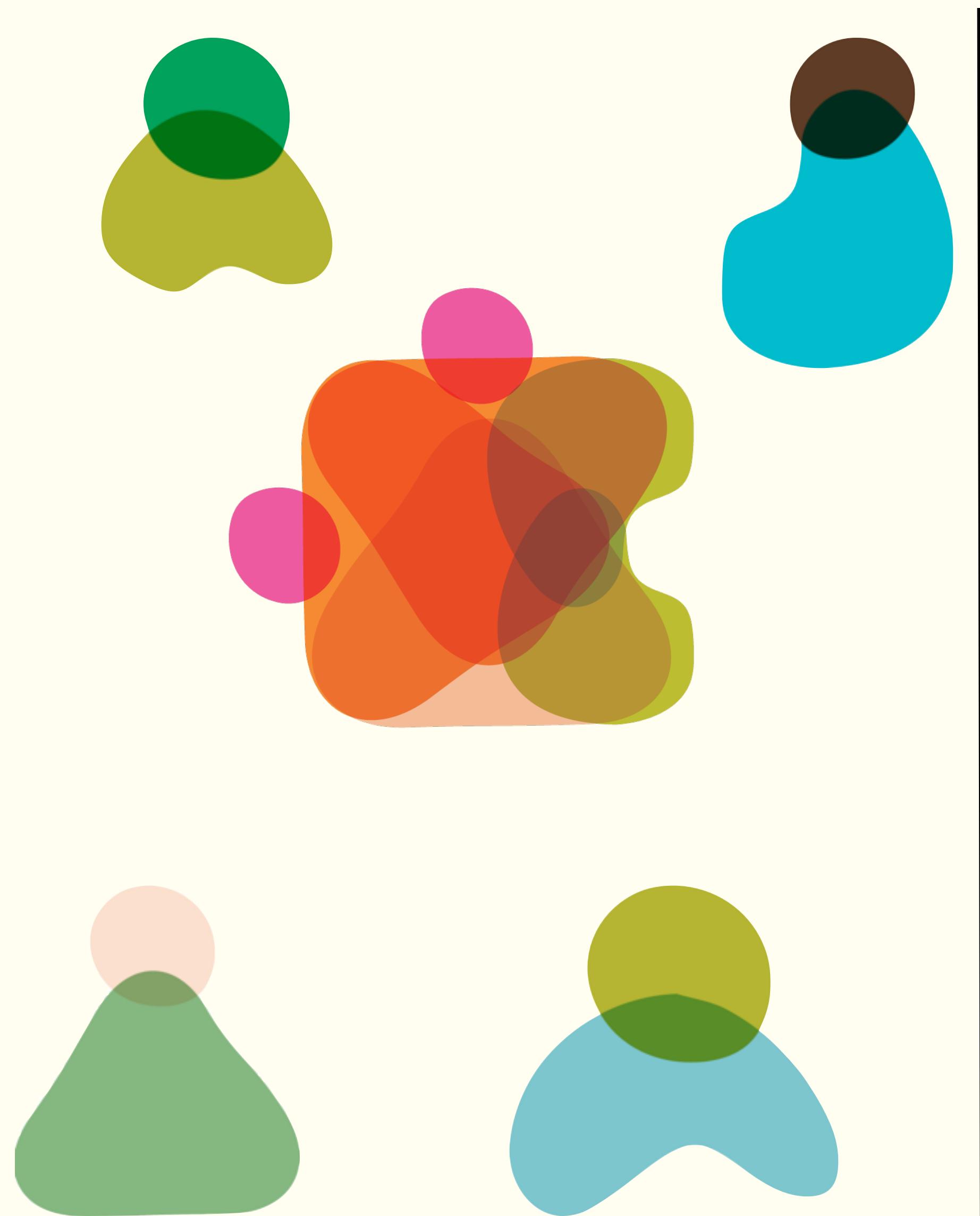
---

1980



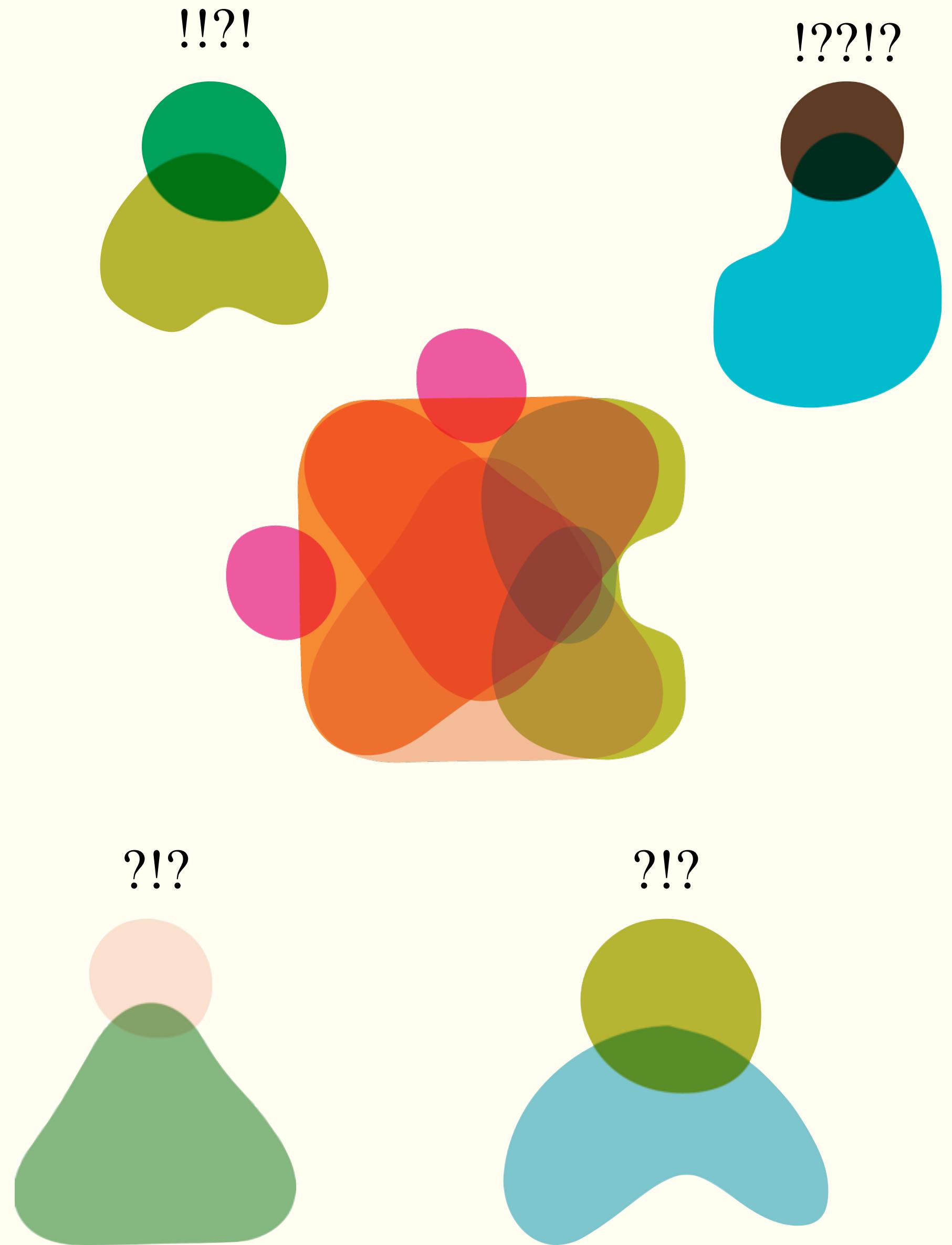
---

1980



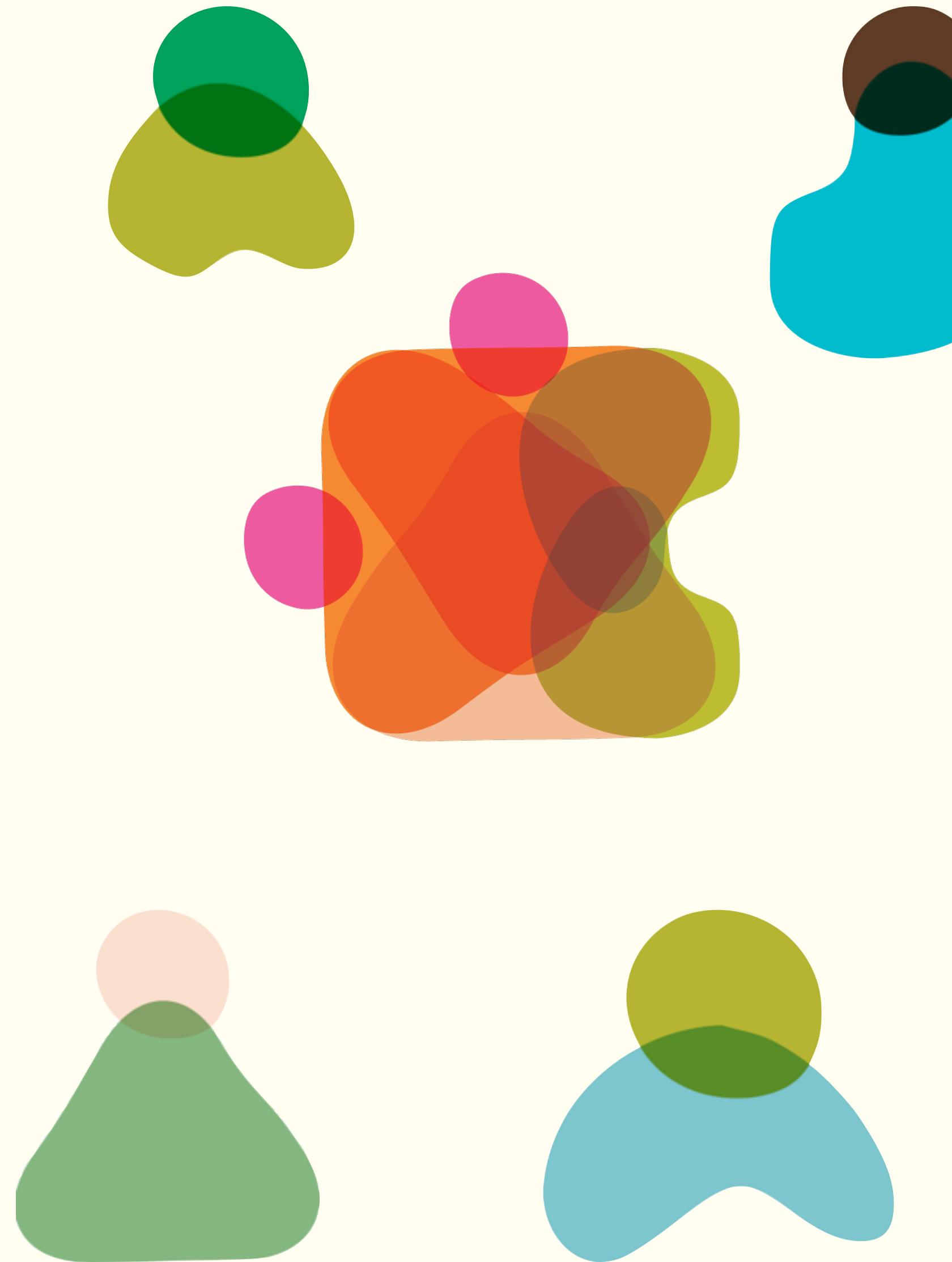
---

1980

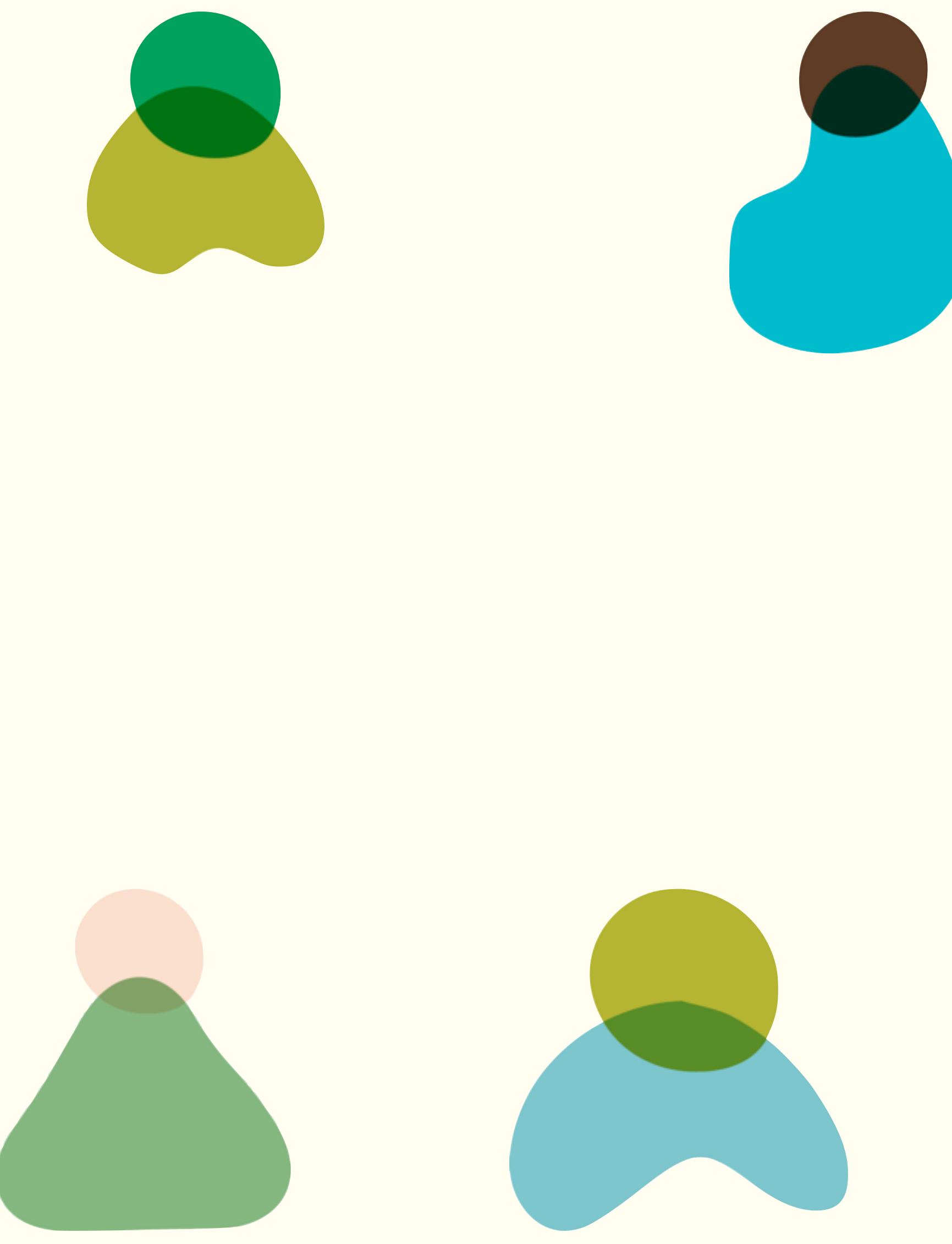


---

1980

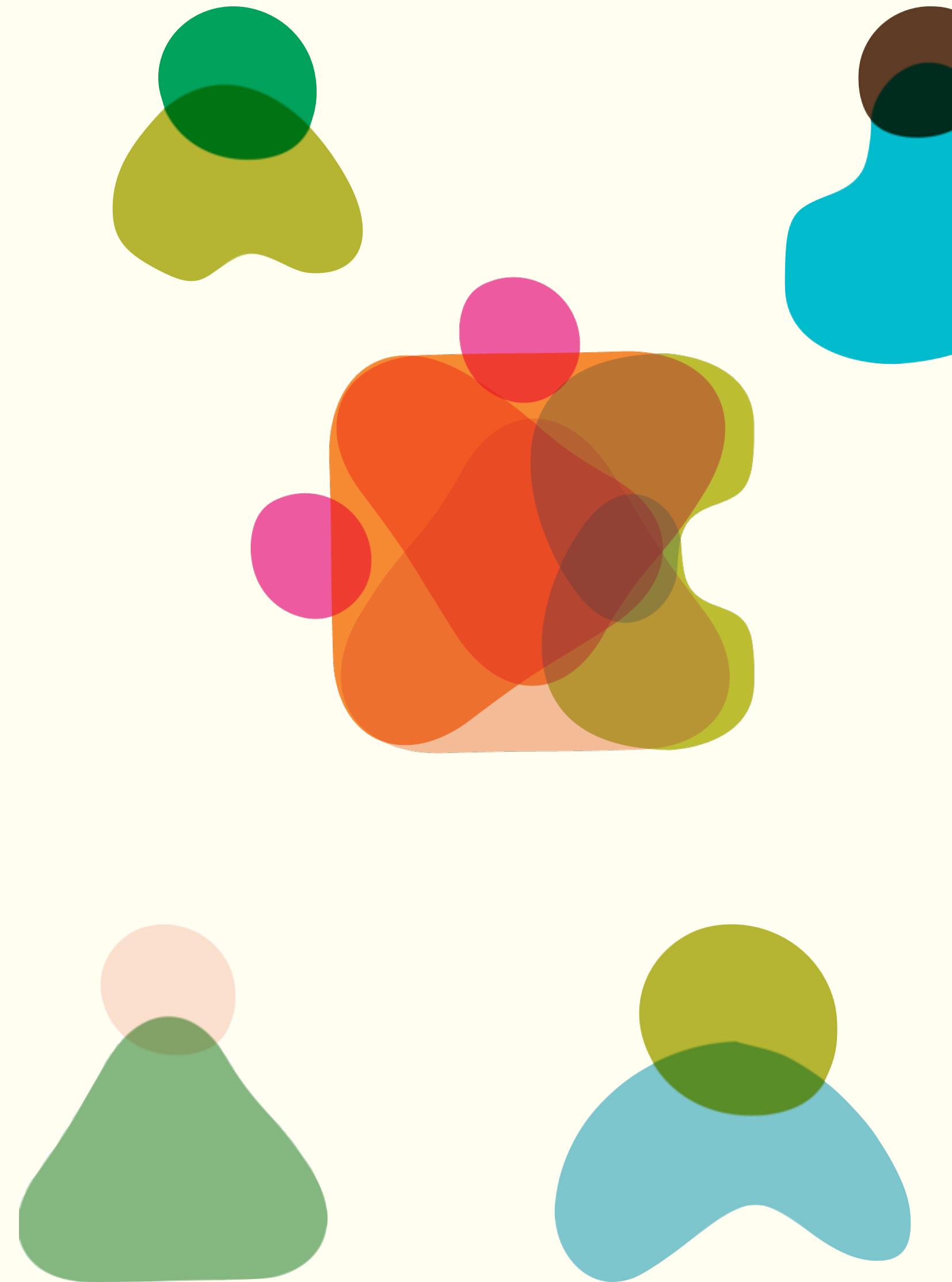


1990

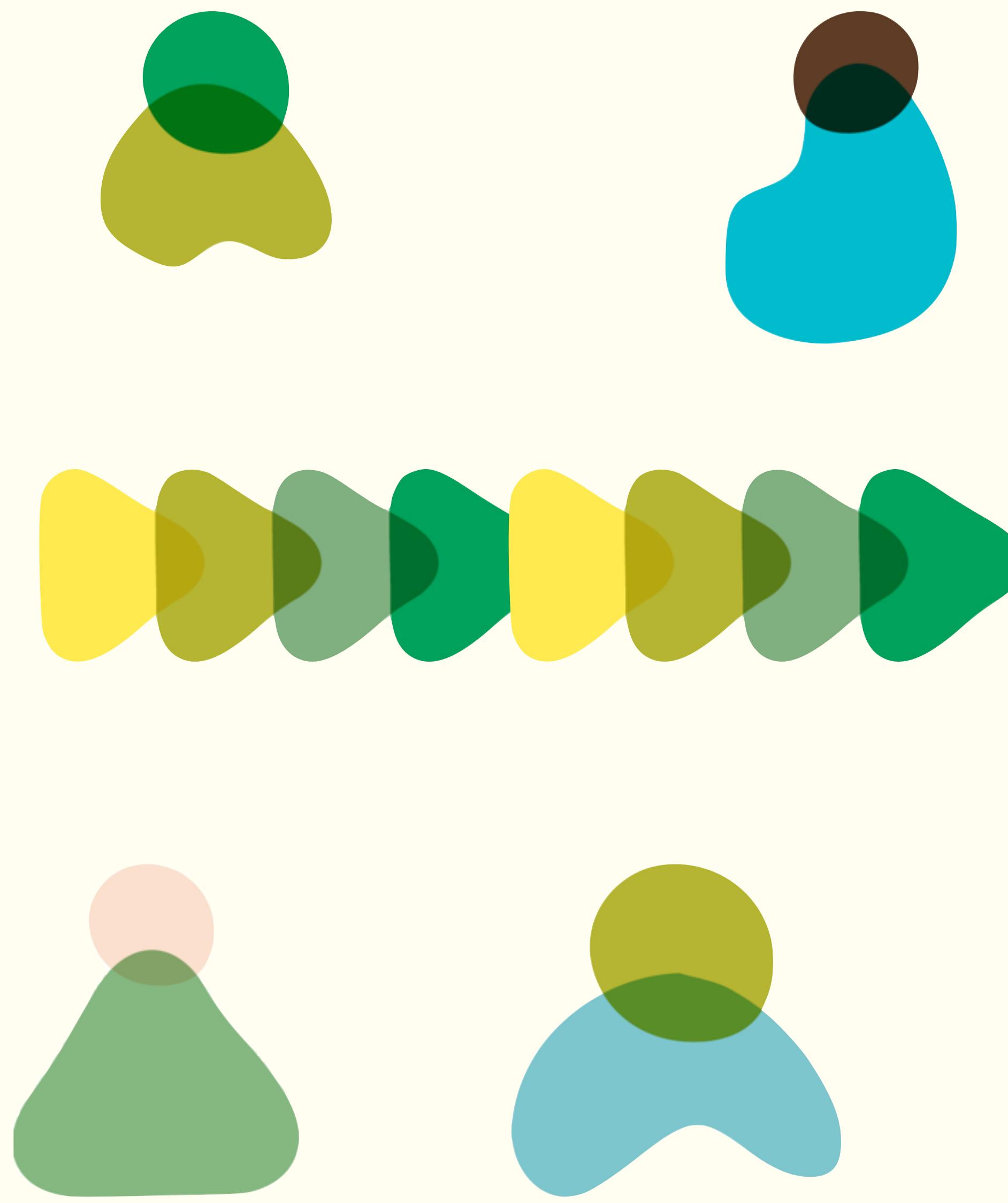


---

1980

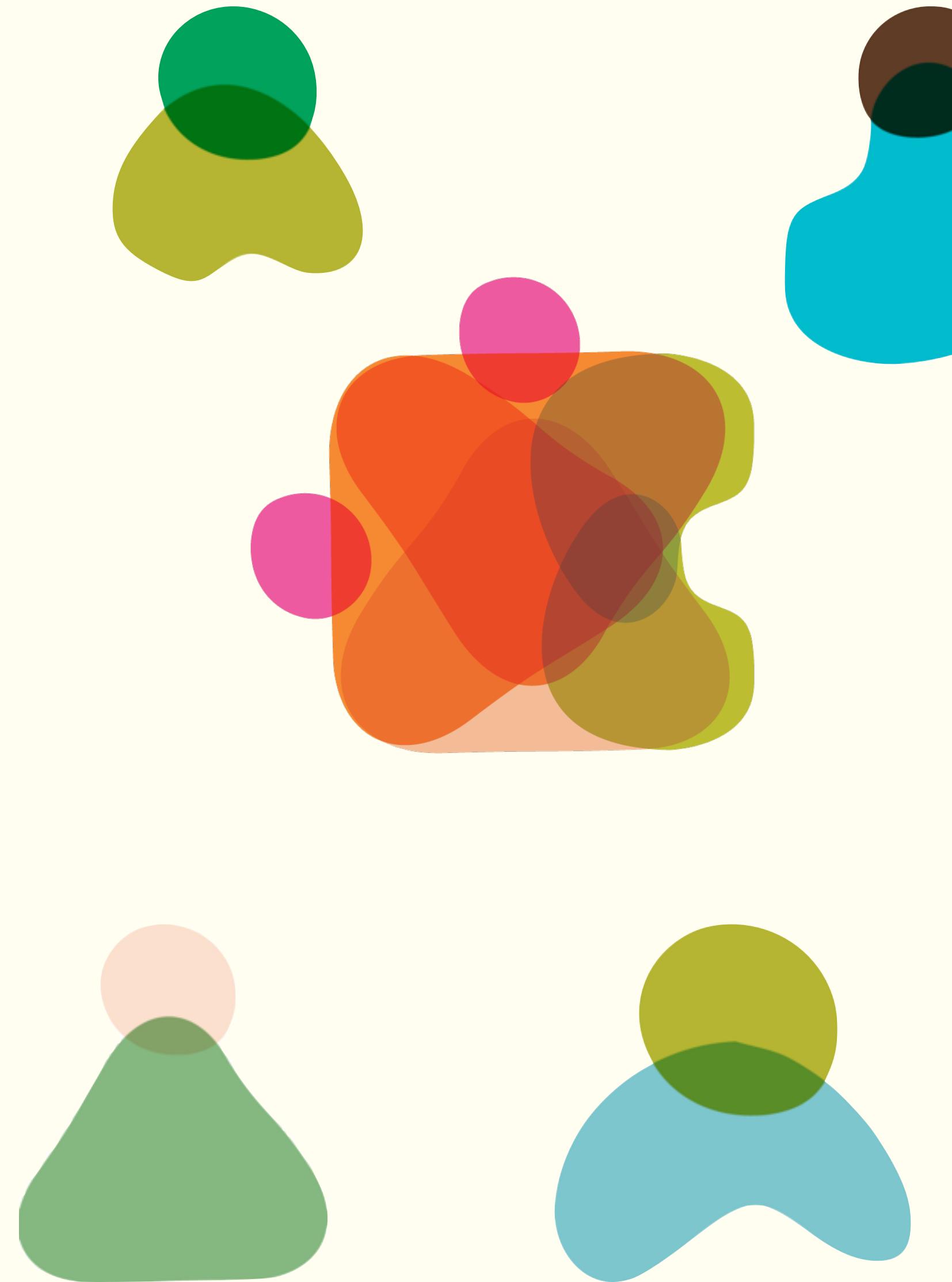


1990

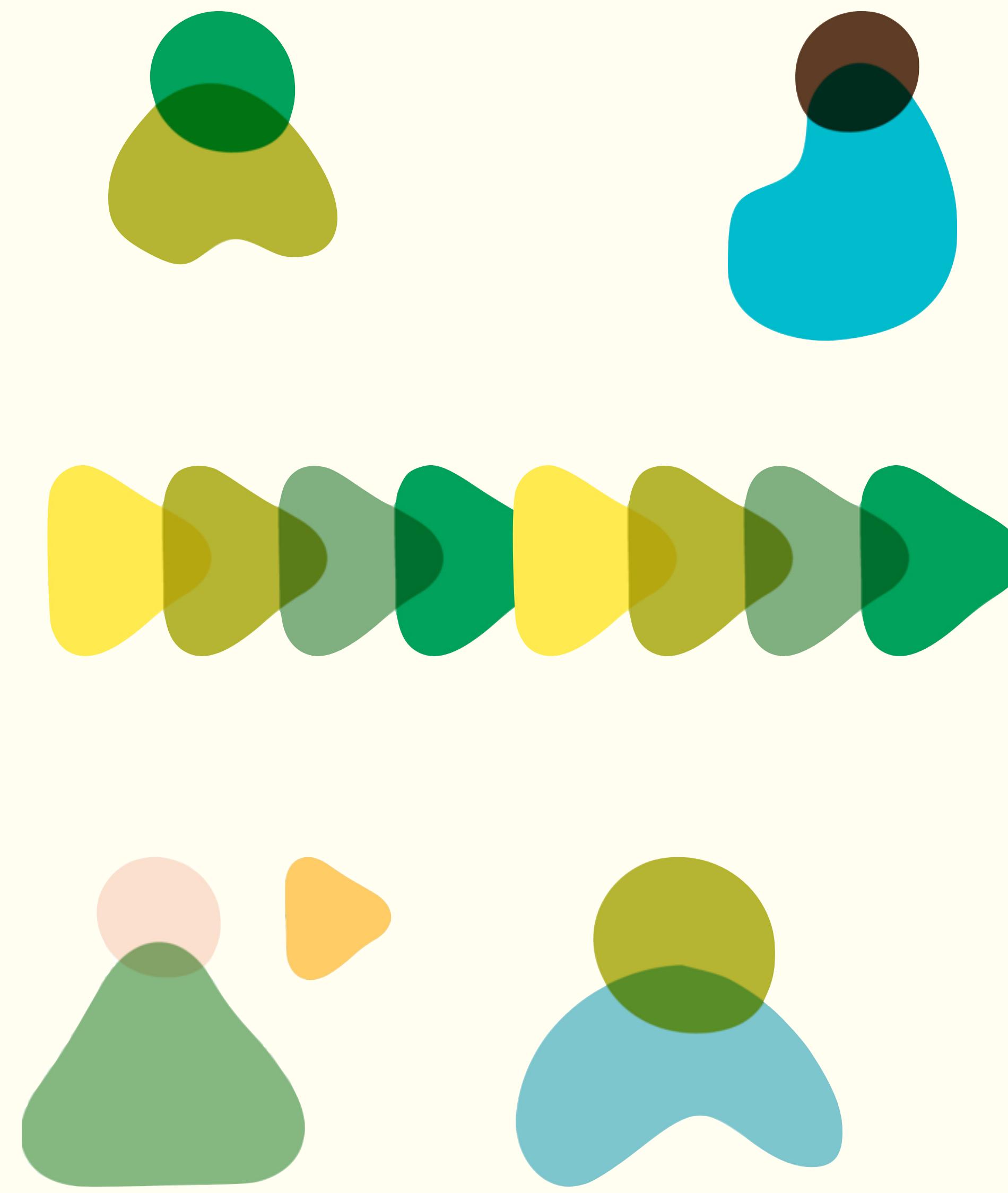


---

1980

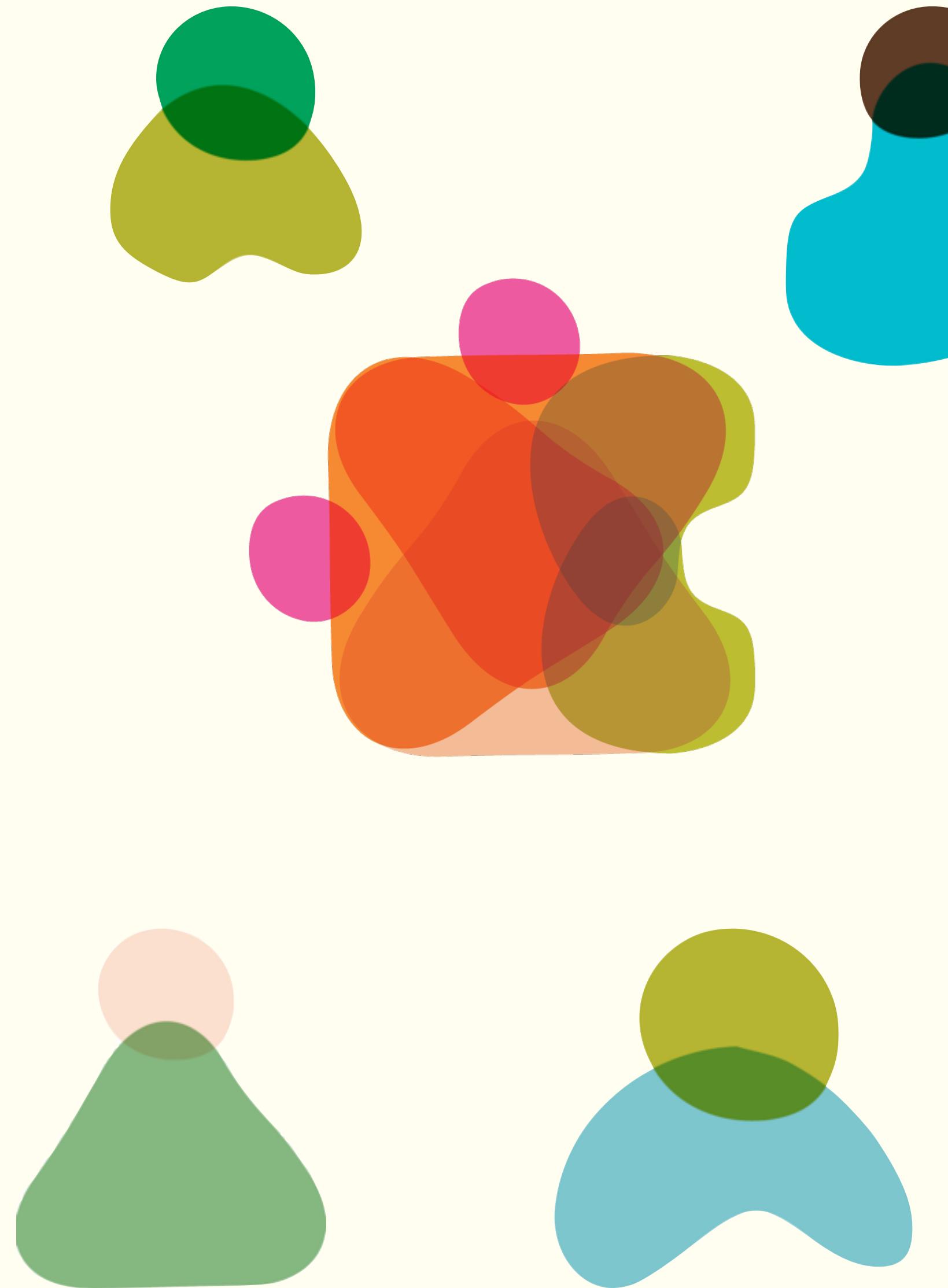


1990

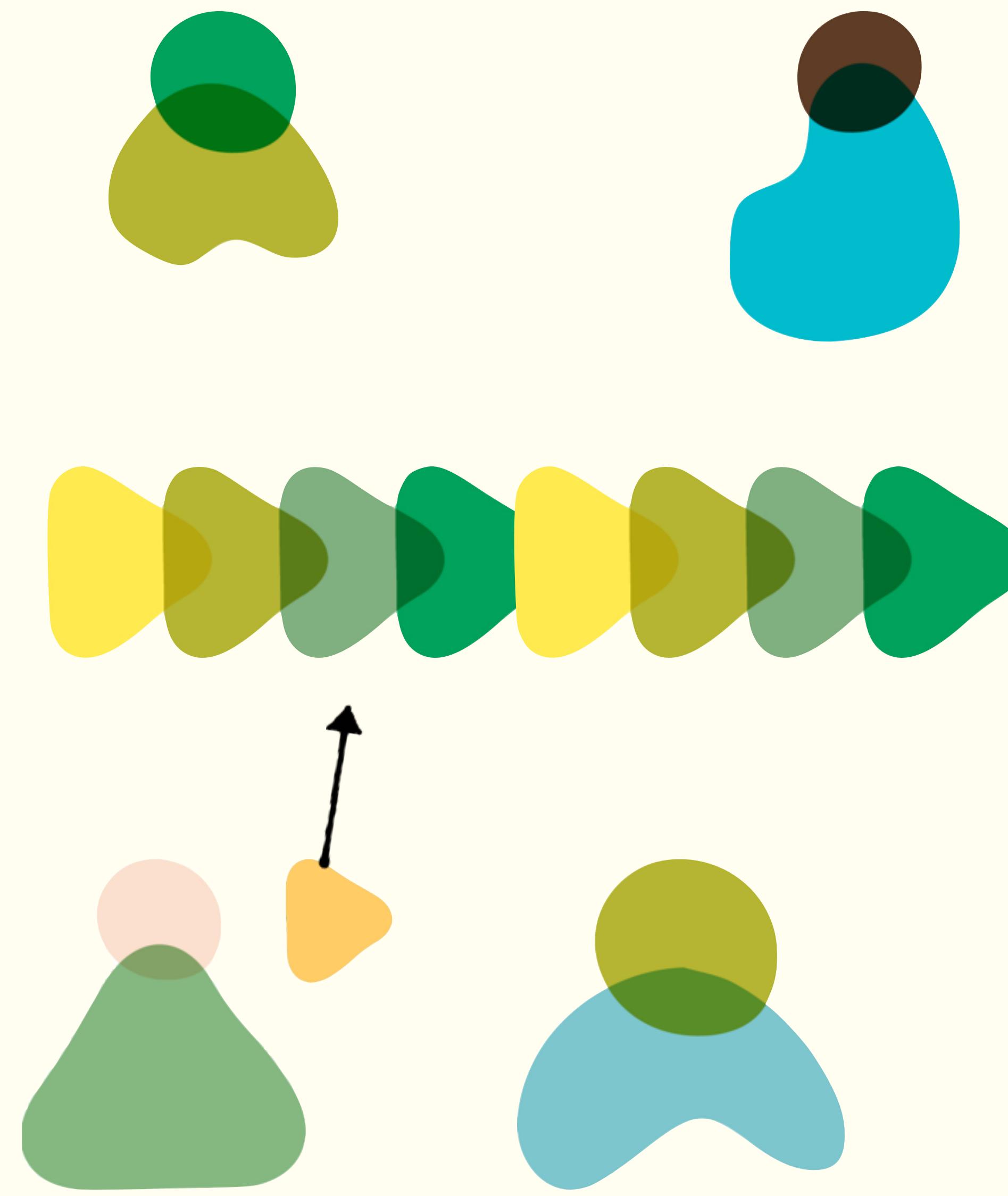


---

1980

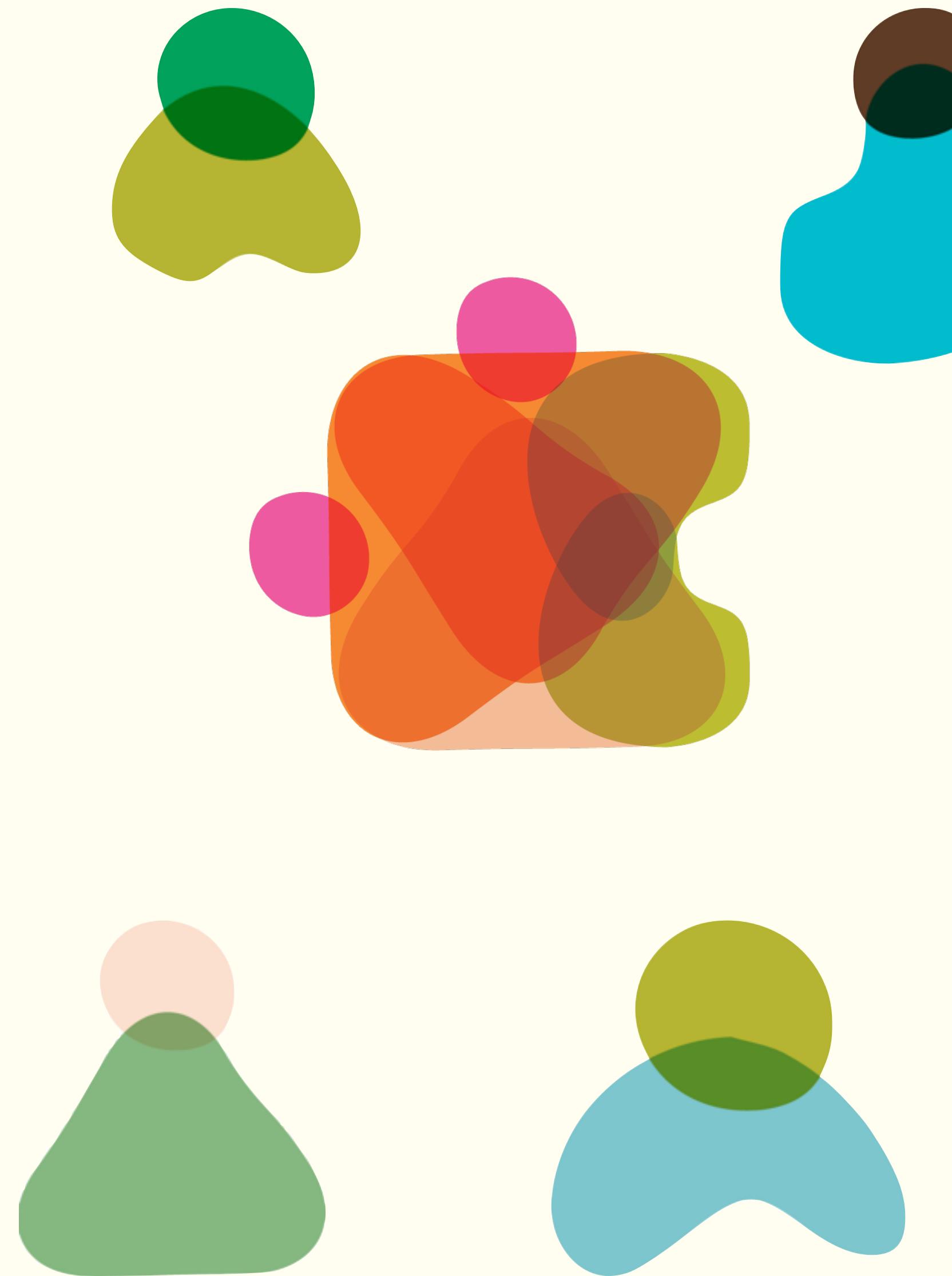


1990

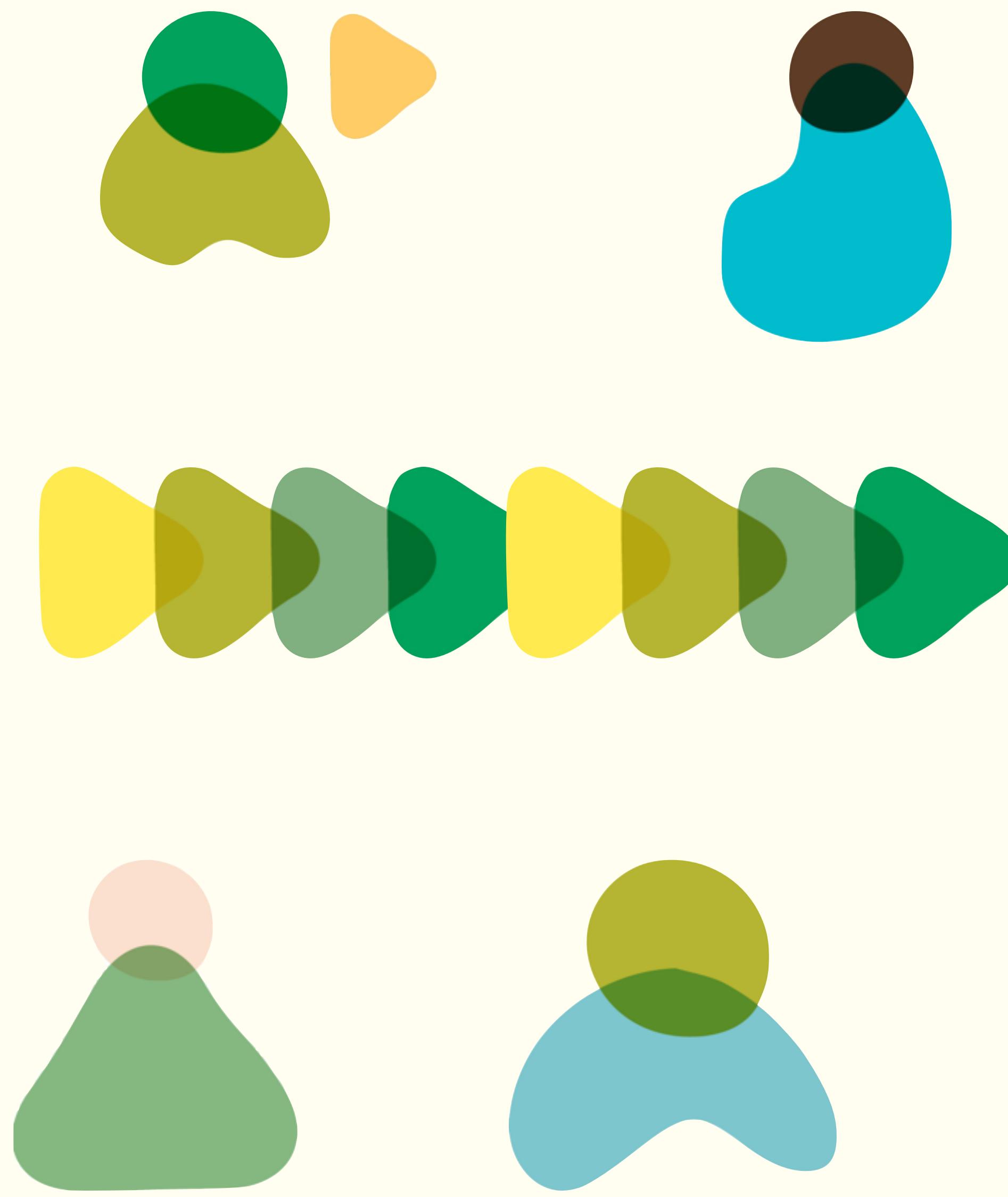


---

1980

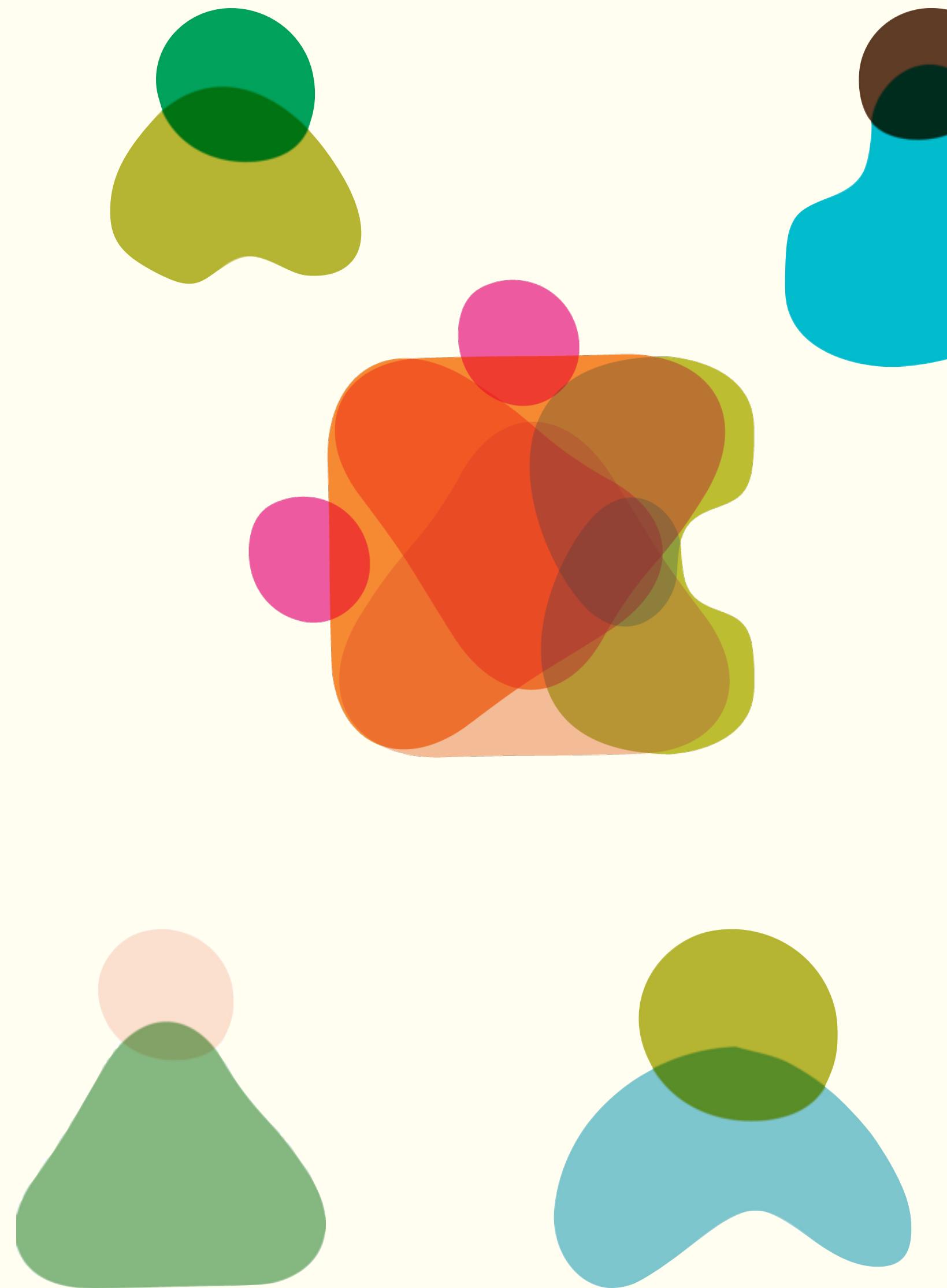


1990

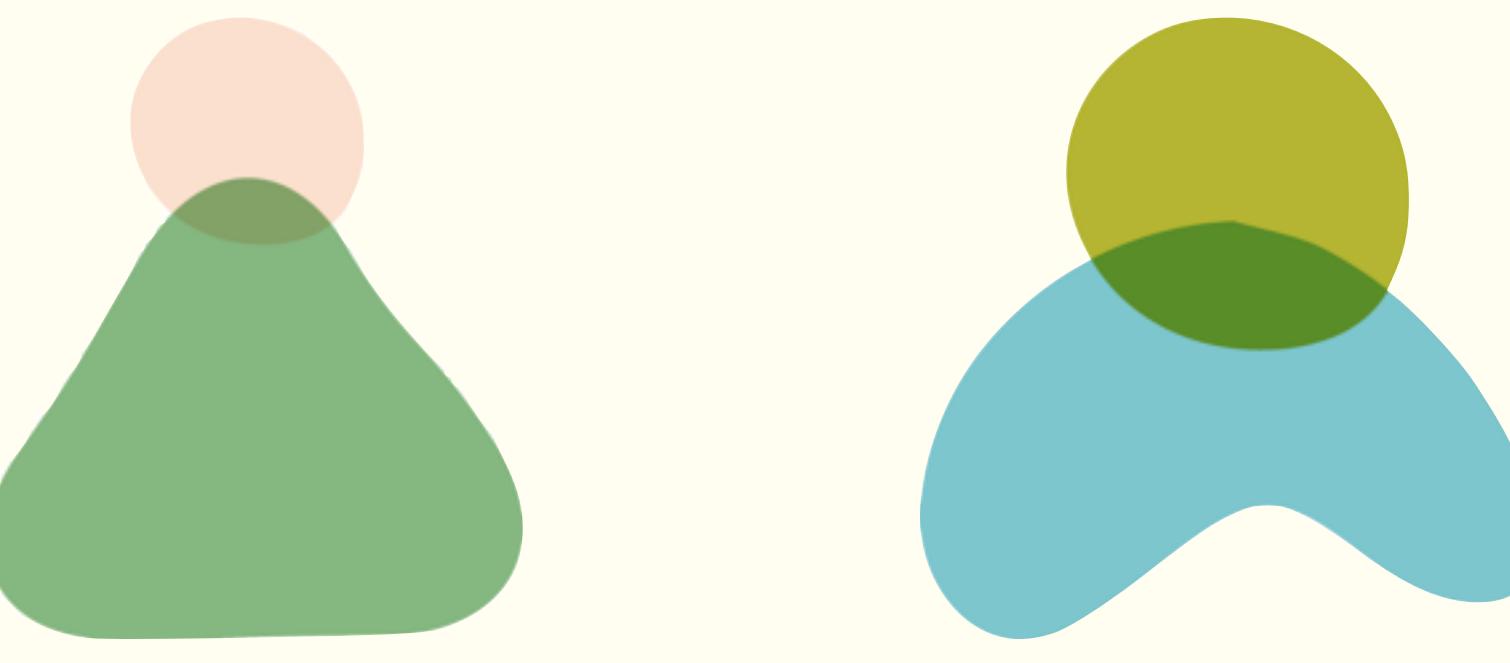
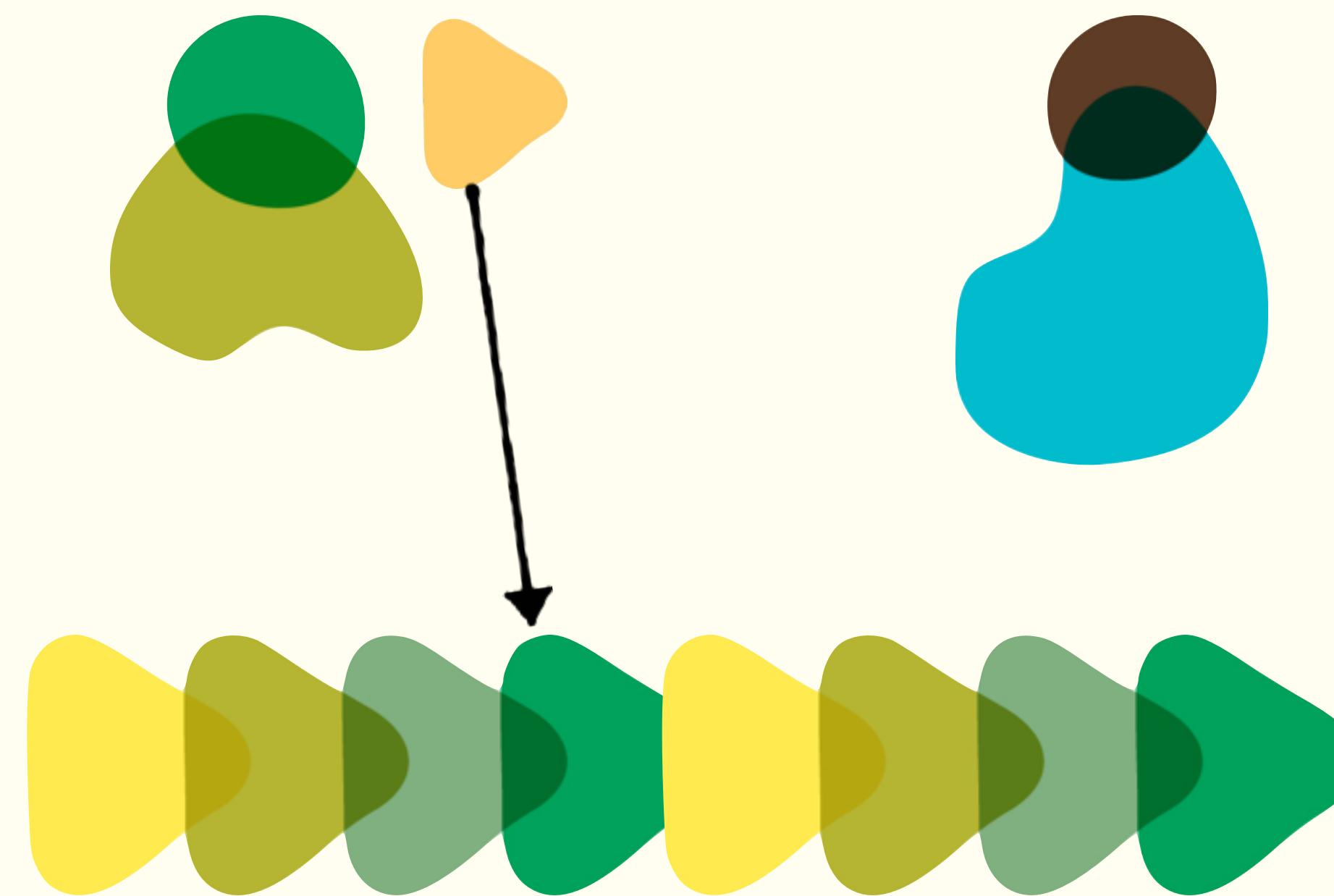


---

1980

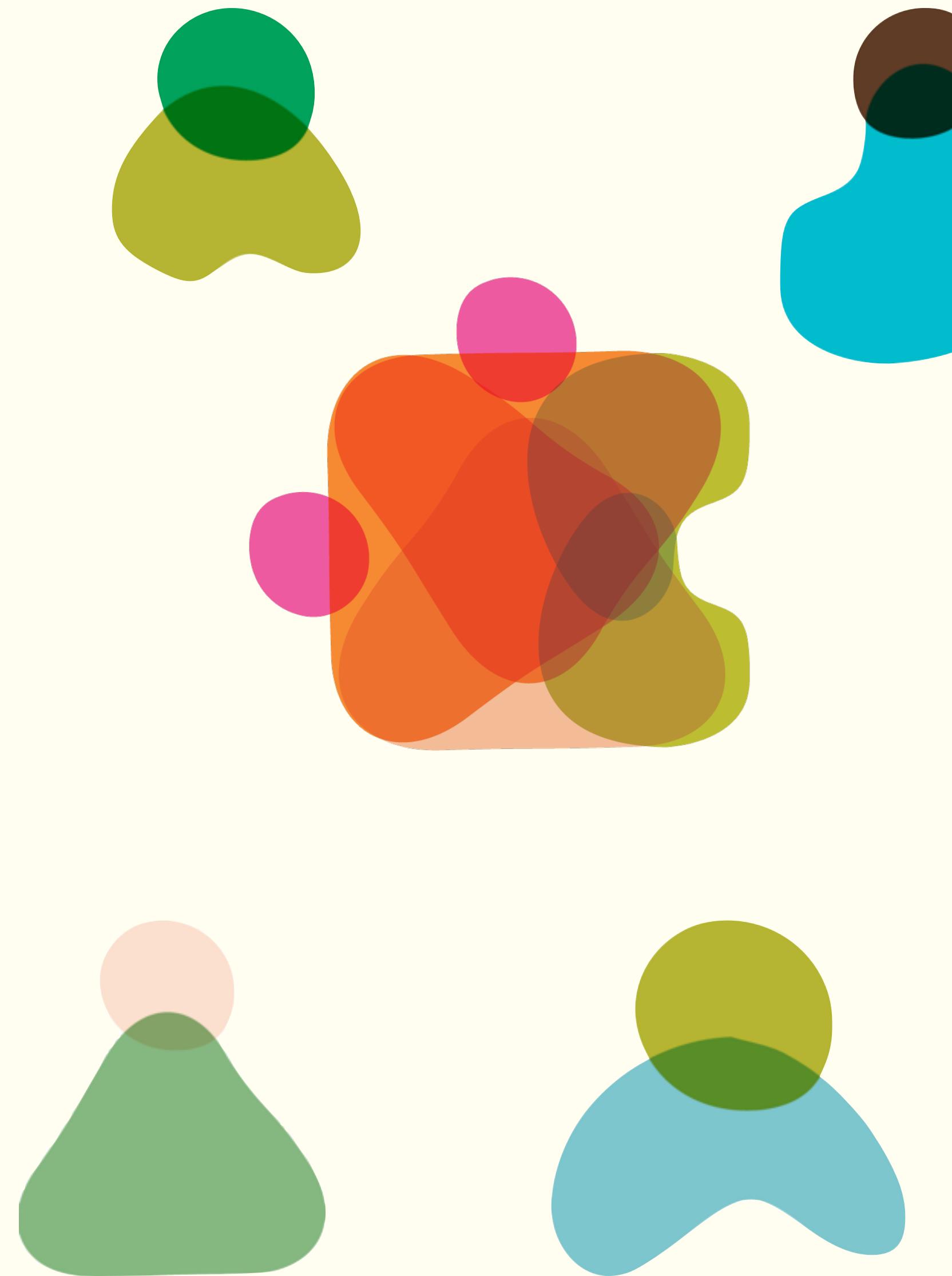


1990

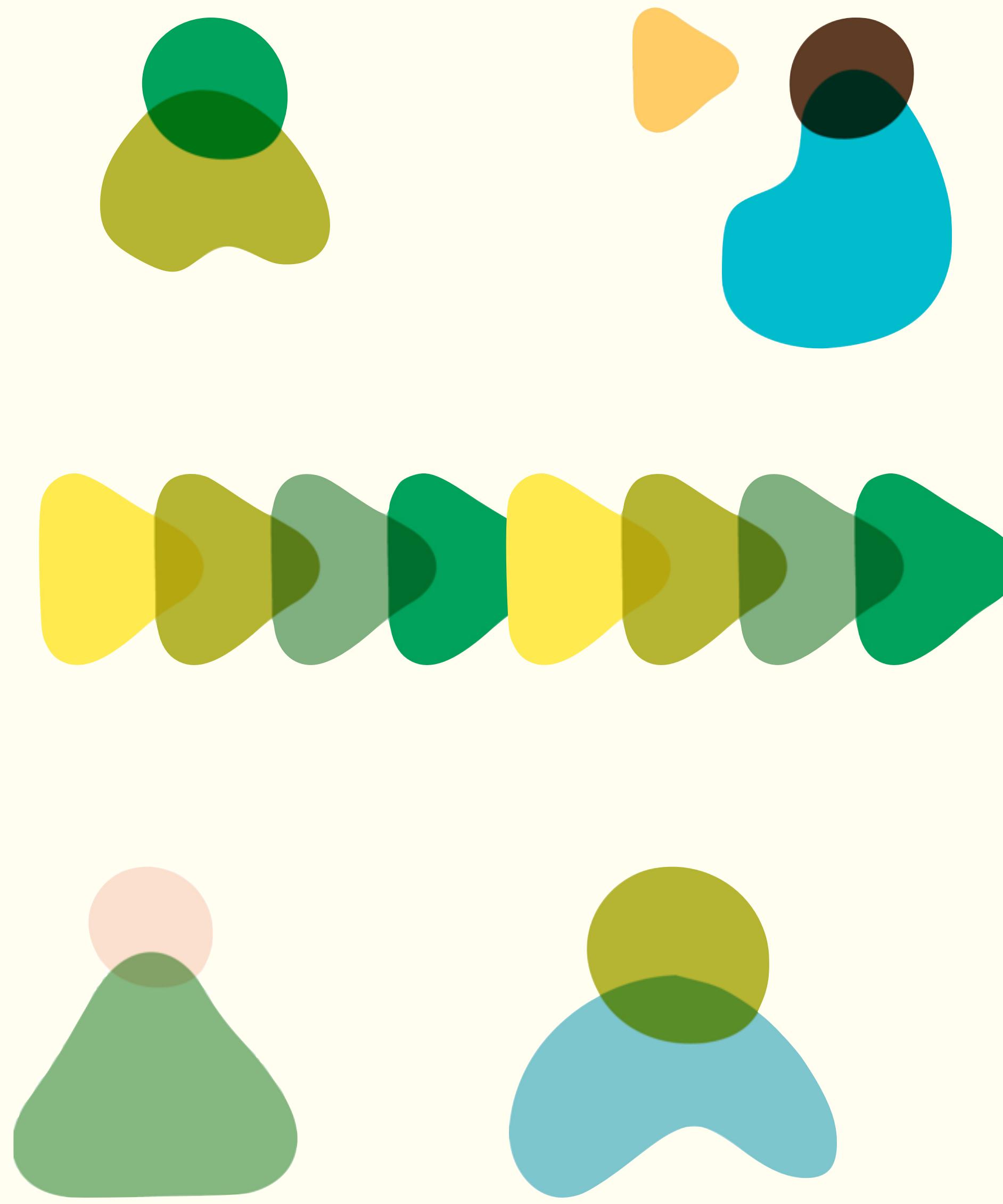


---

1980

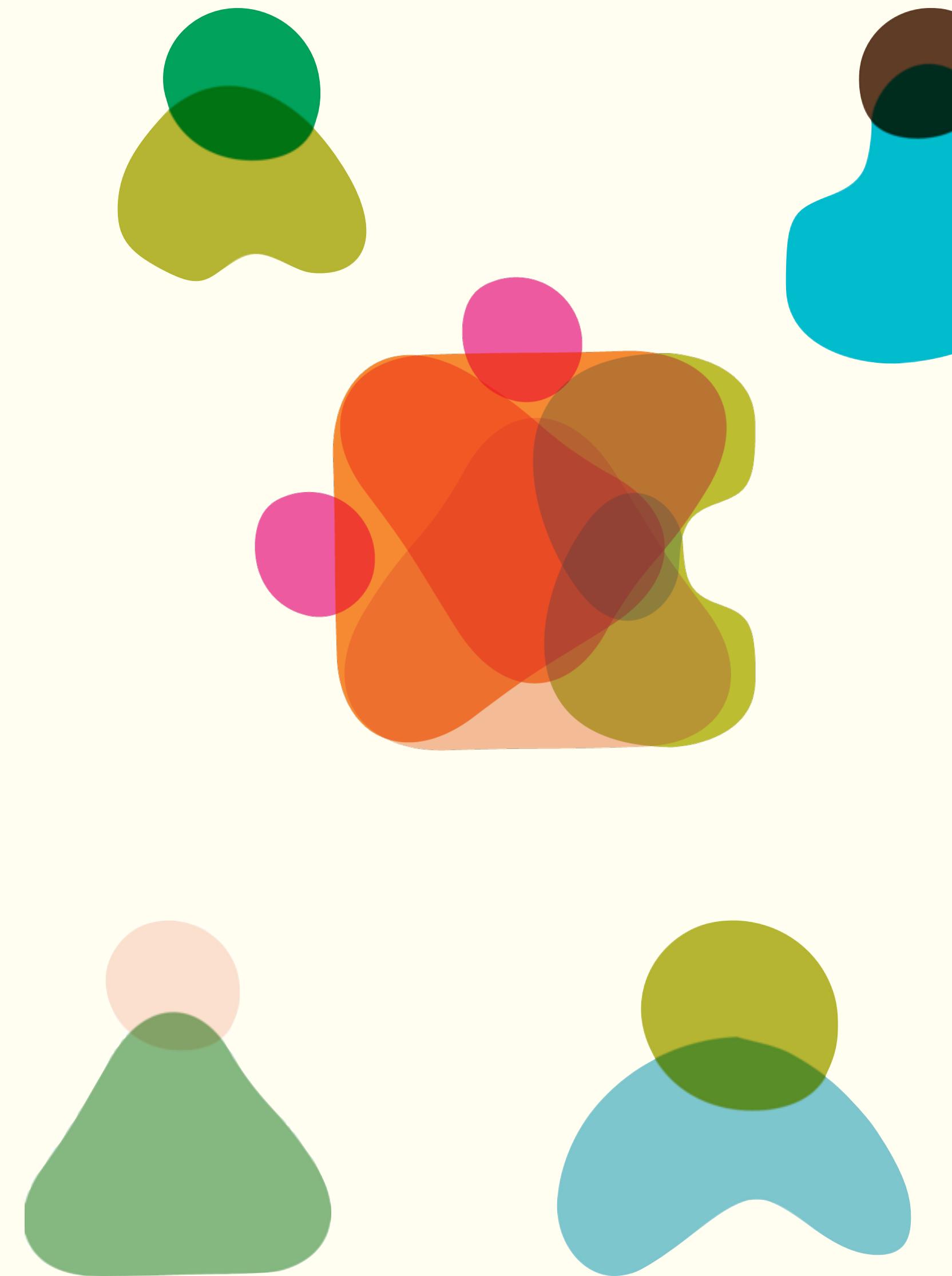


1990

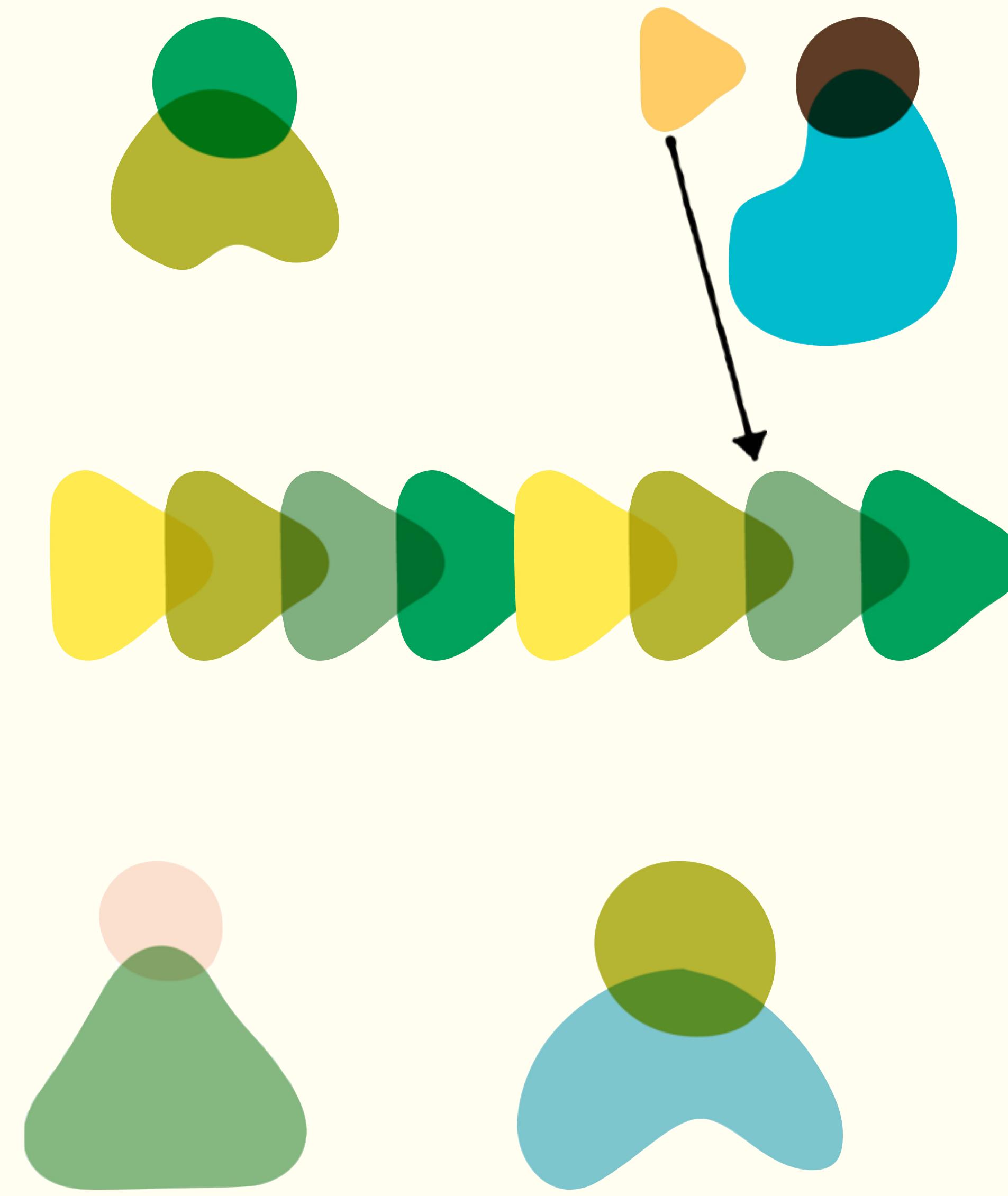


---

1980

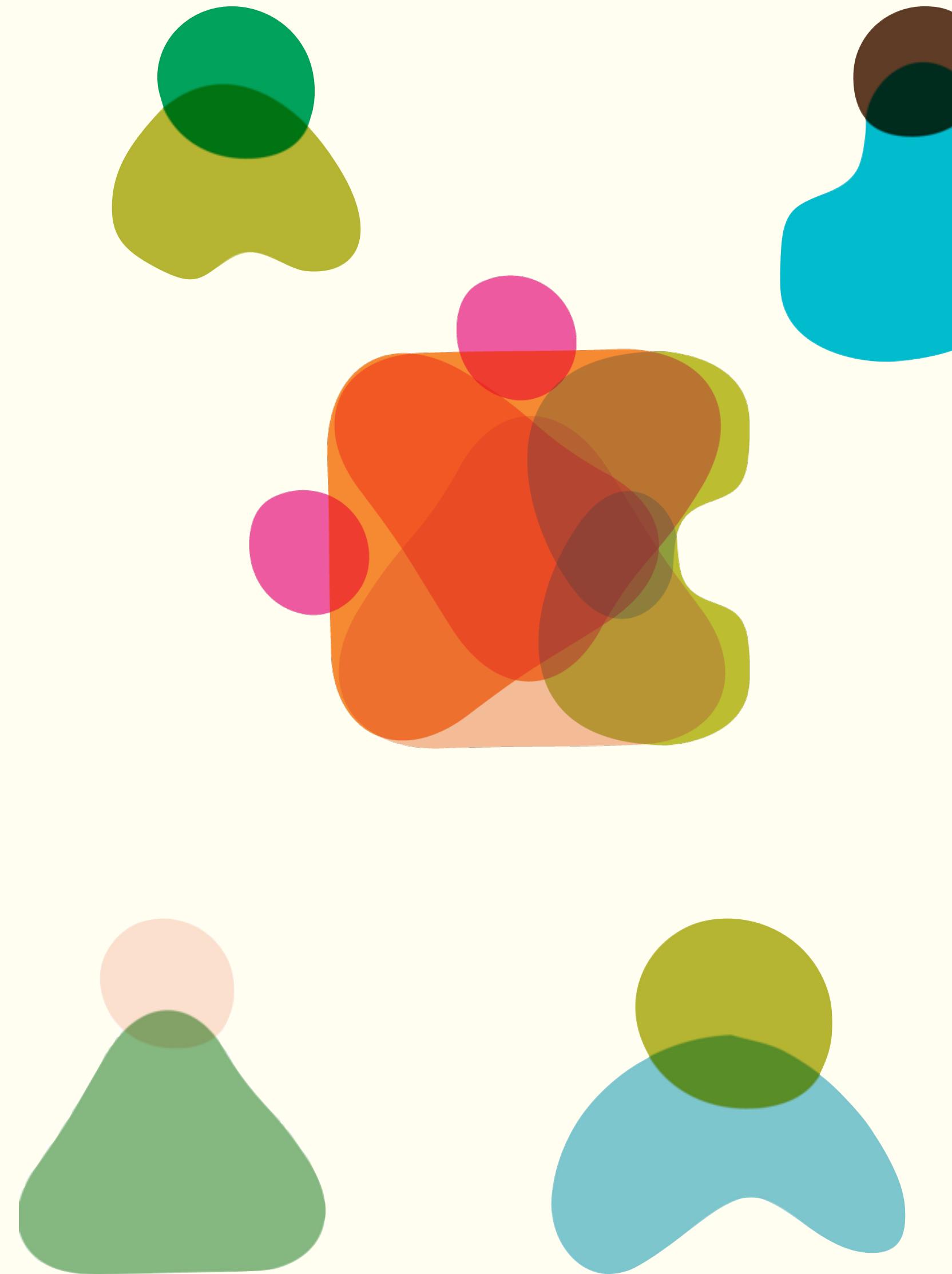


1990

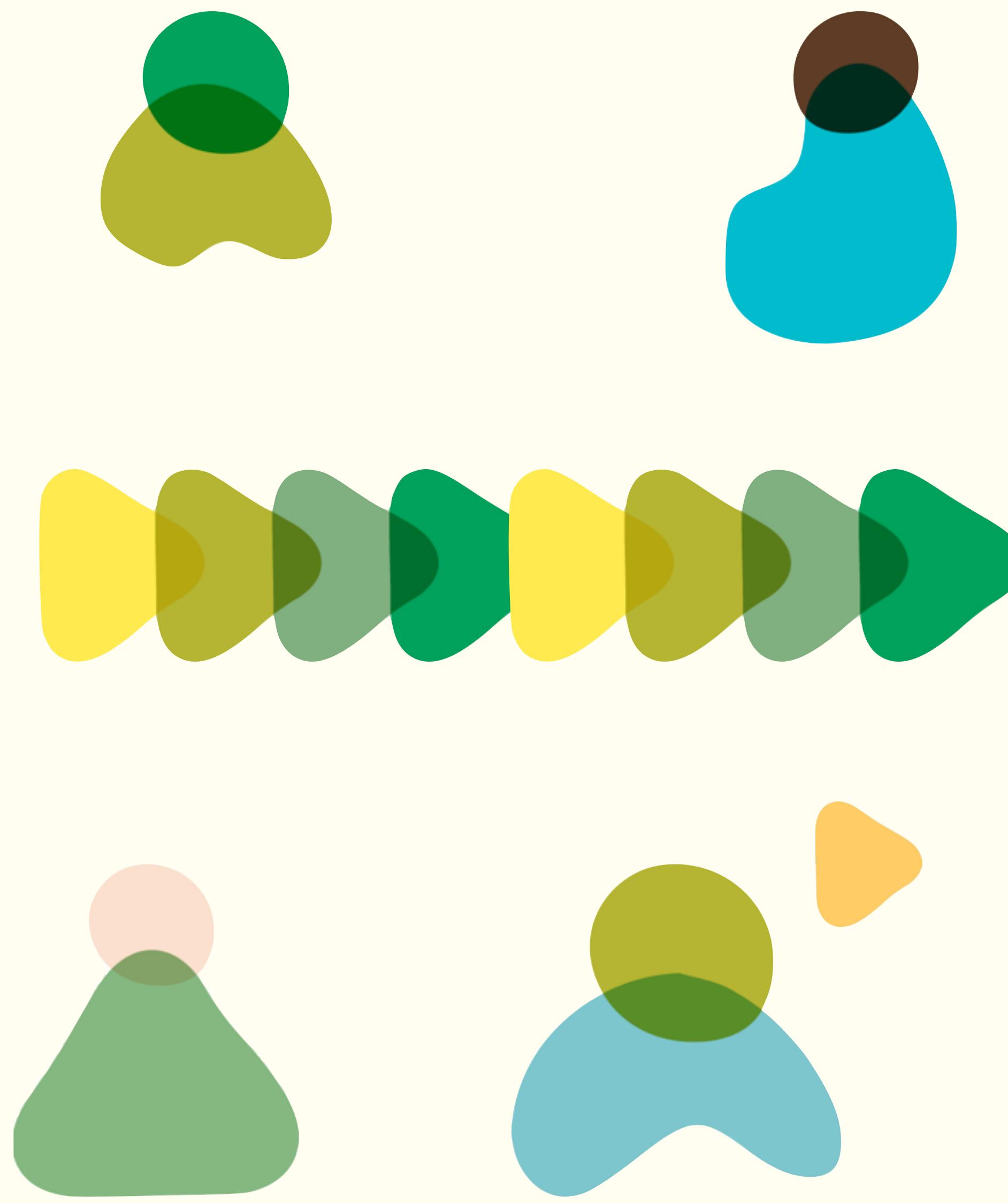


---

1980

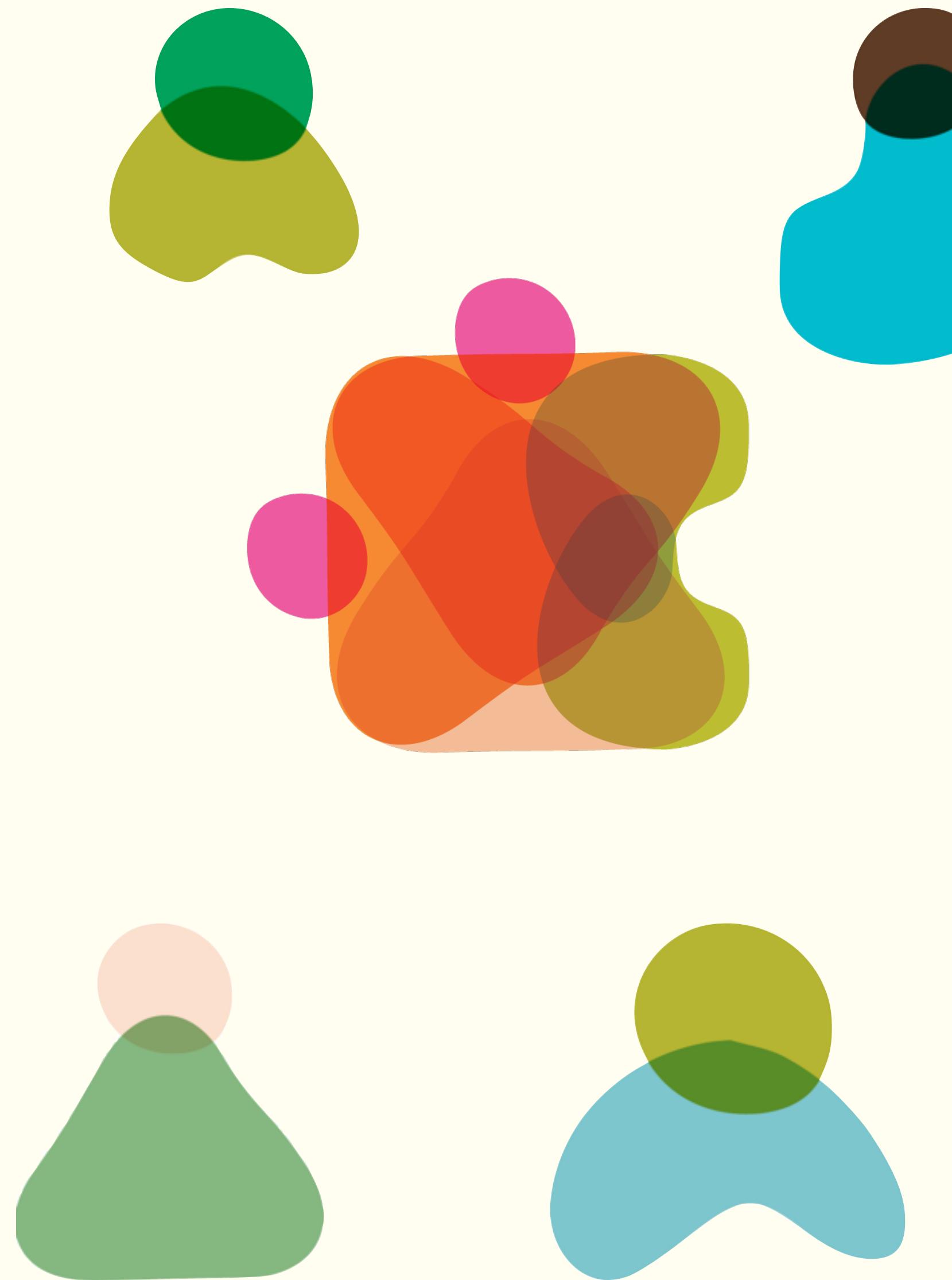


1990

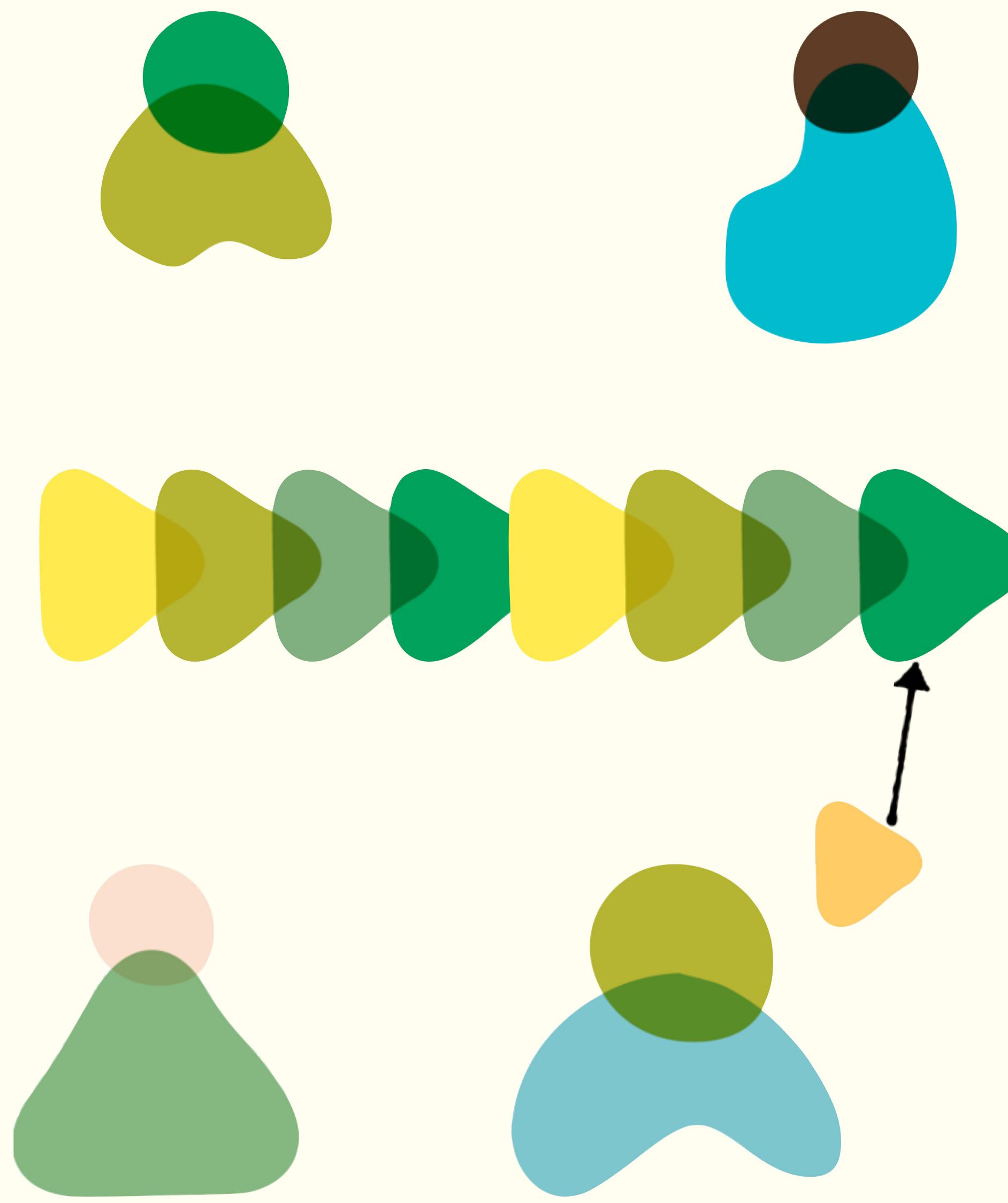


---

1980

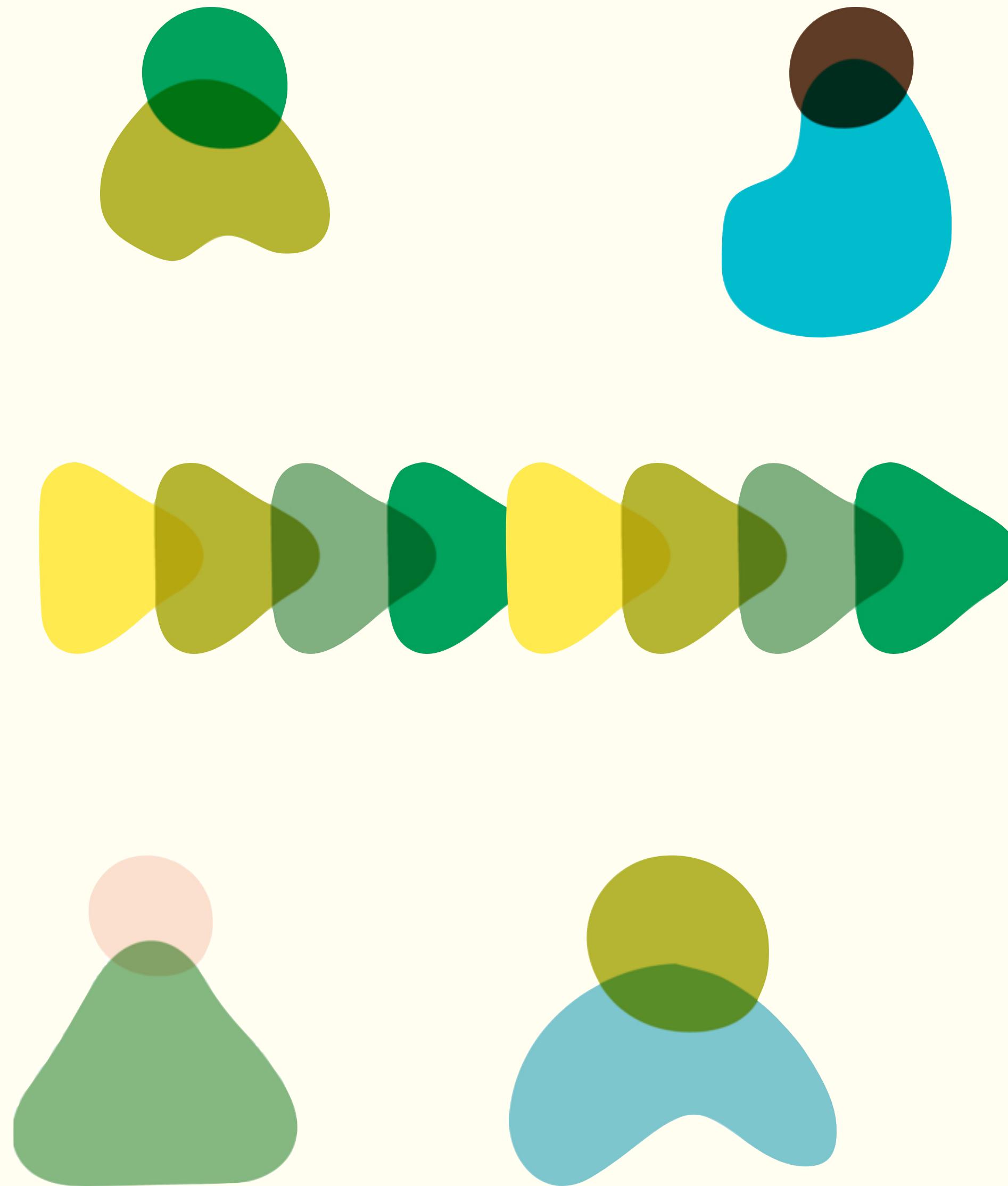
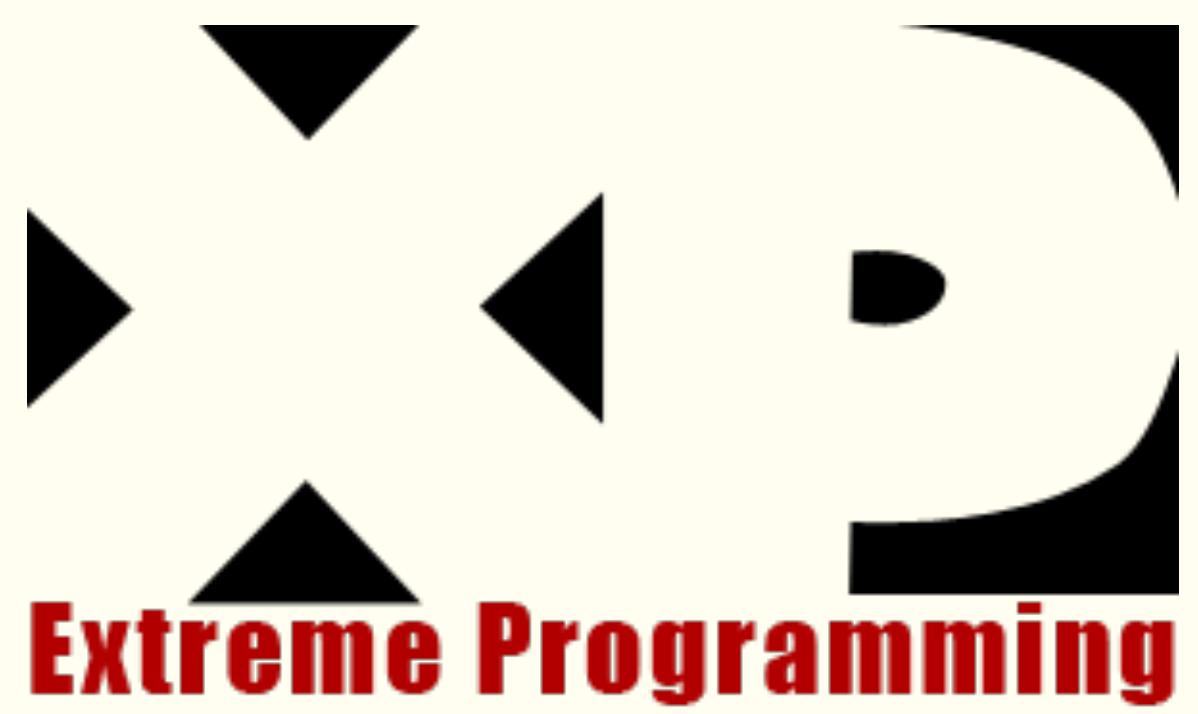


1990



---

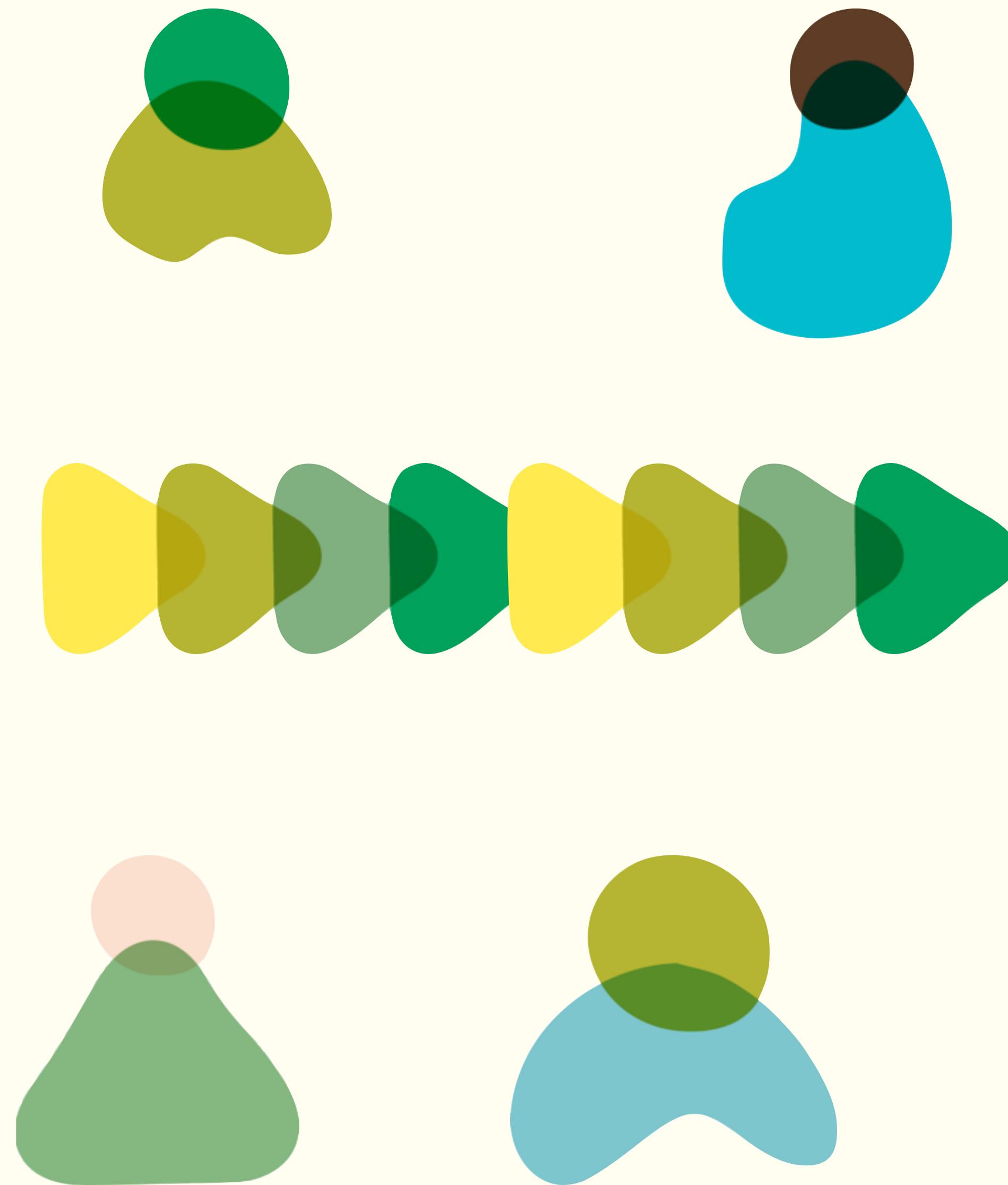
1990



---

continuous integration:

1990

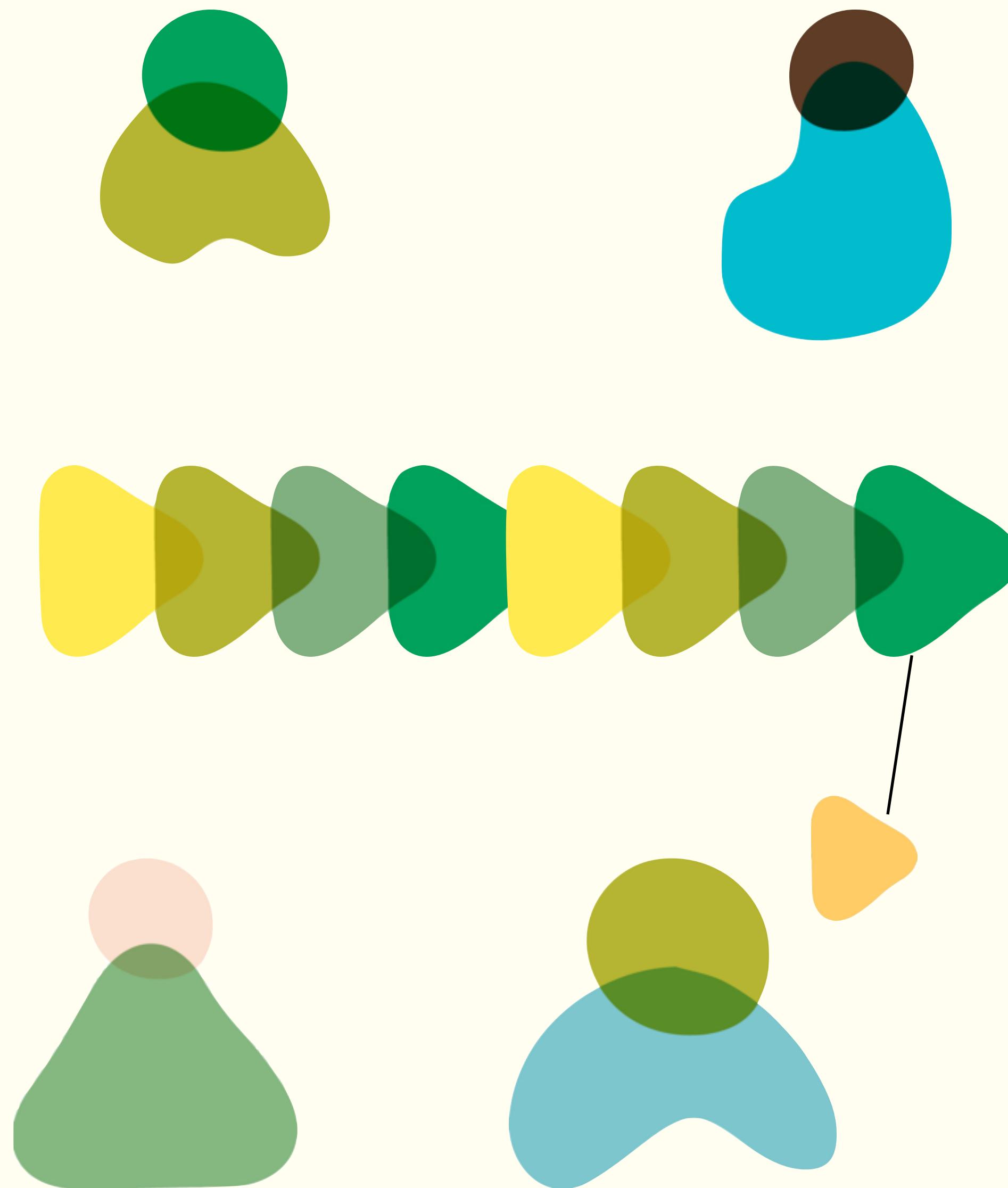


---

continuous integration:

everyone commits to  
trunk at least once per day

1990



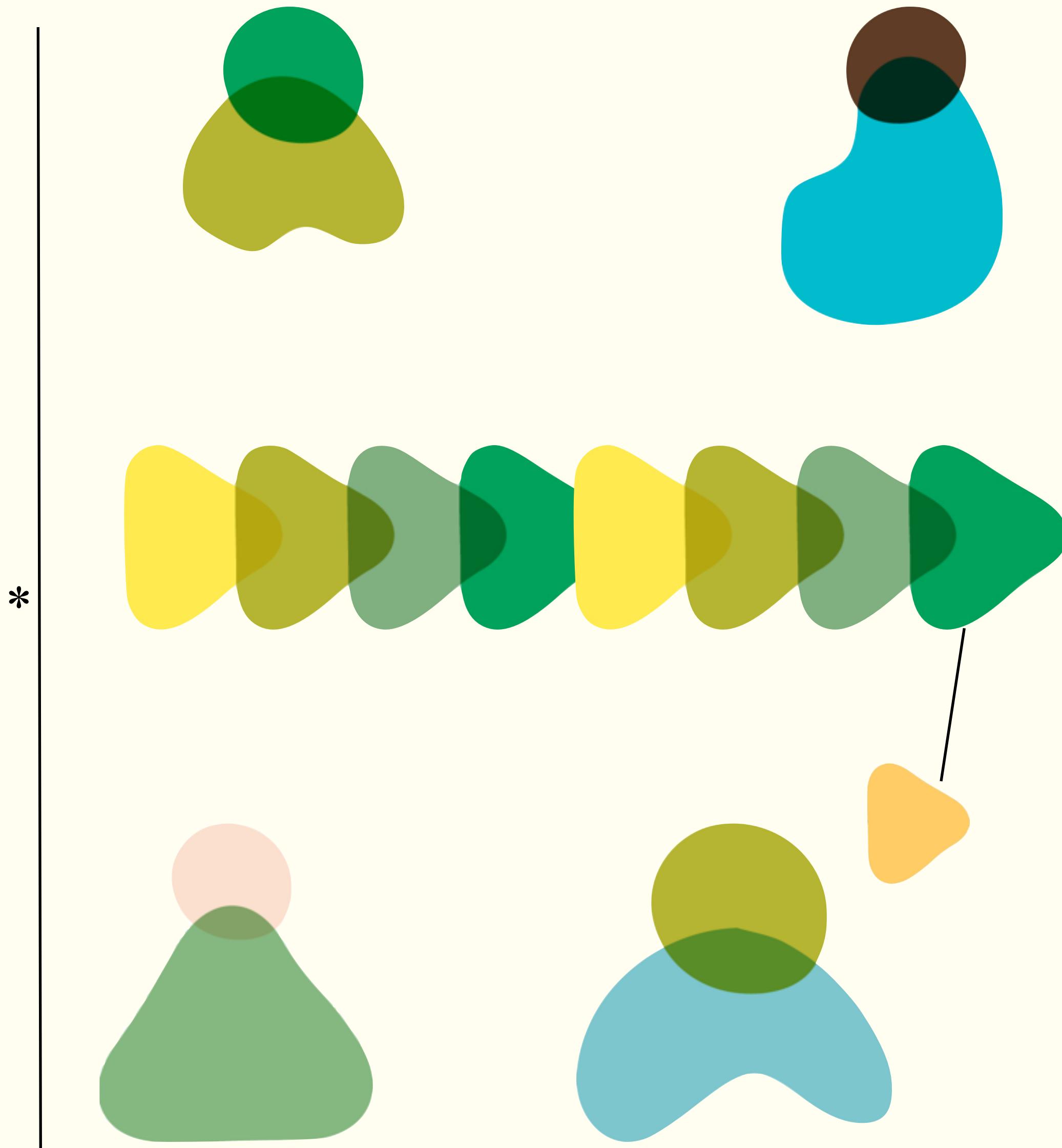
---

continuous integration:

everyone commits to  
trunk at least once per day

\*teams using feature branching  
aren't doing continuous integration\*\*

1990



---

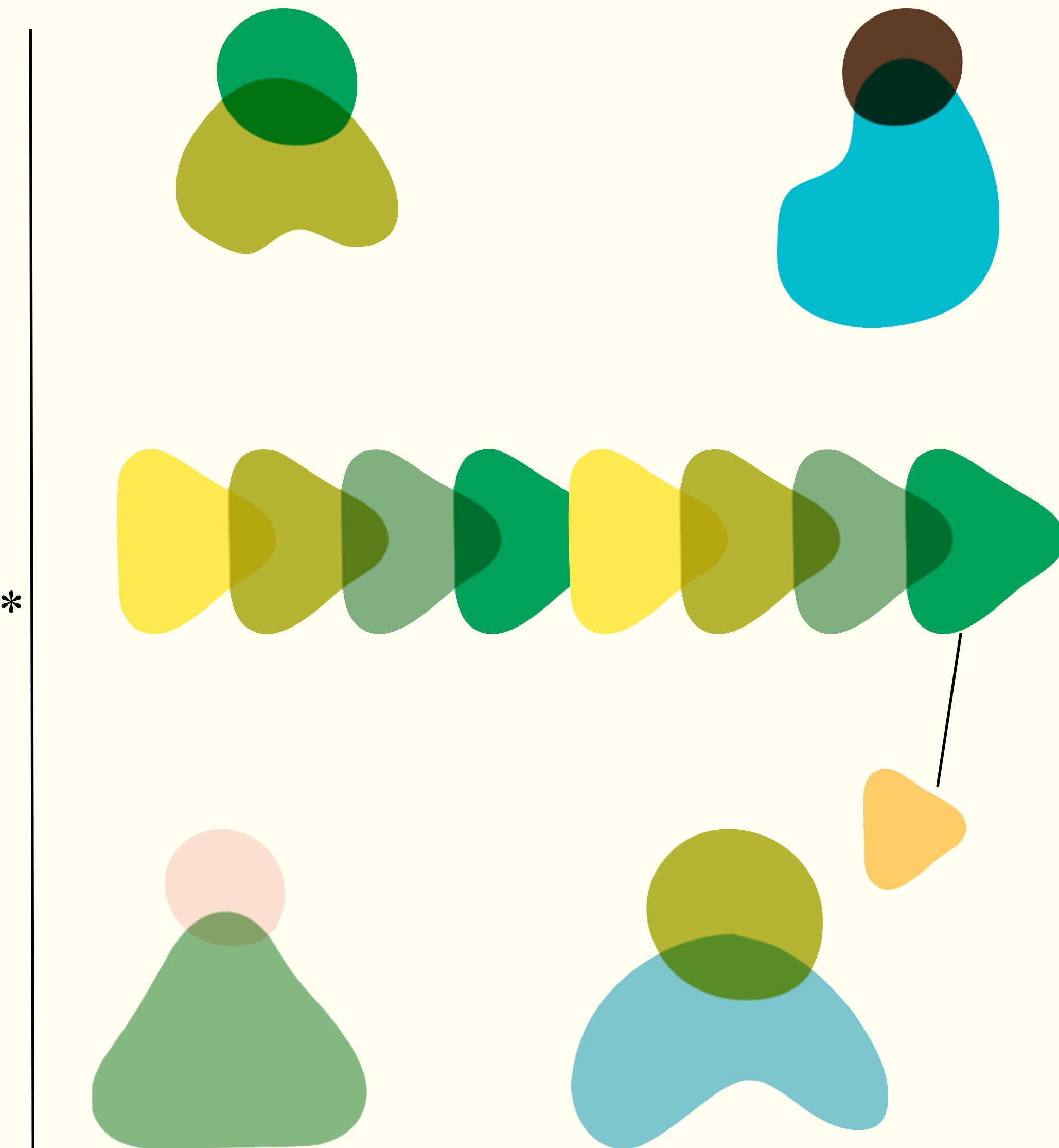
continuous integration:

everyone commits to  
trunk at least once per day

\*teams using feature branching  
aren't doing continuous integration\*\*

\*\*unless all branches live < 1 day

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*

---

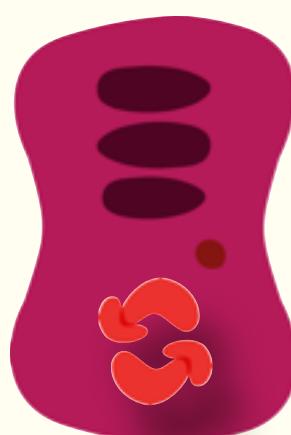
1980

1990

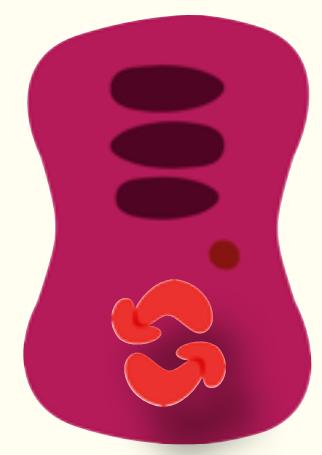
2000

2010

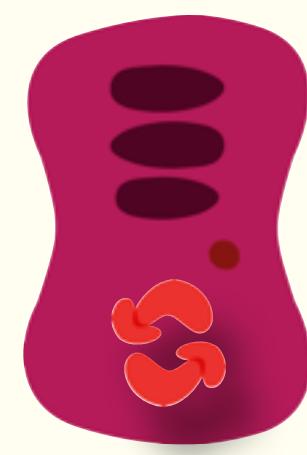
current  
day



*continuous  
integration*

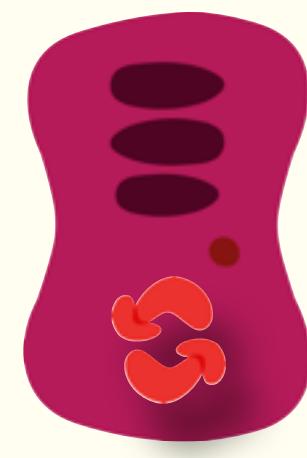


*continuous  
integration*



canonical integration point

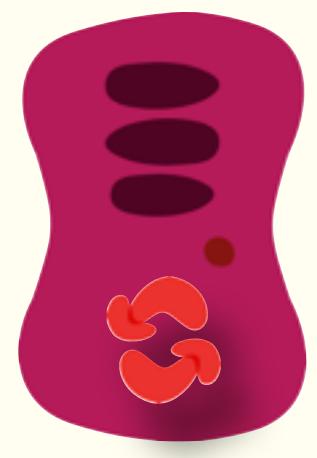
*continuous  
integration*



canonical integration point

useful platform to do other *stuff*

*continuous  
integration*



canonical integration point

useful platform to do other *stuff*

unit testing

*continuous  
integration*



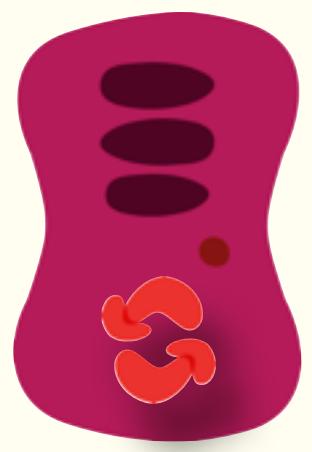
canonical integration point

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*

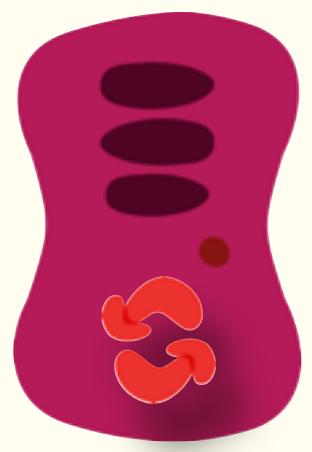
unit testing

functional testing

integration testing

automated machine provisioning

*continuous  
integration*



canonical integration point

useful platform to do other *stuff*

unit testing

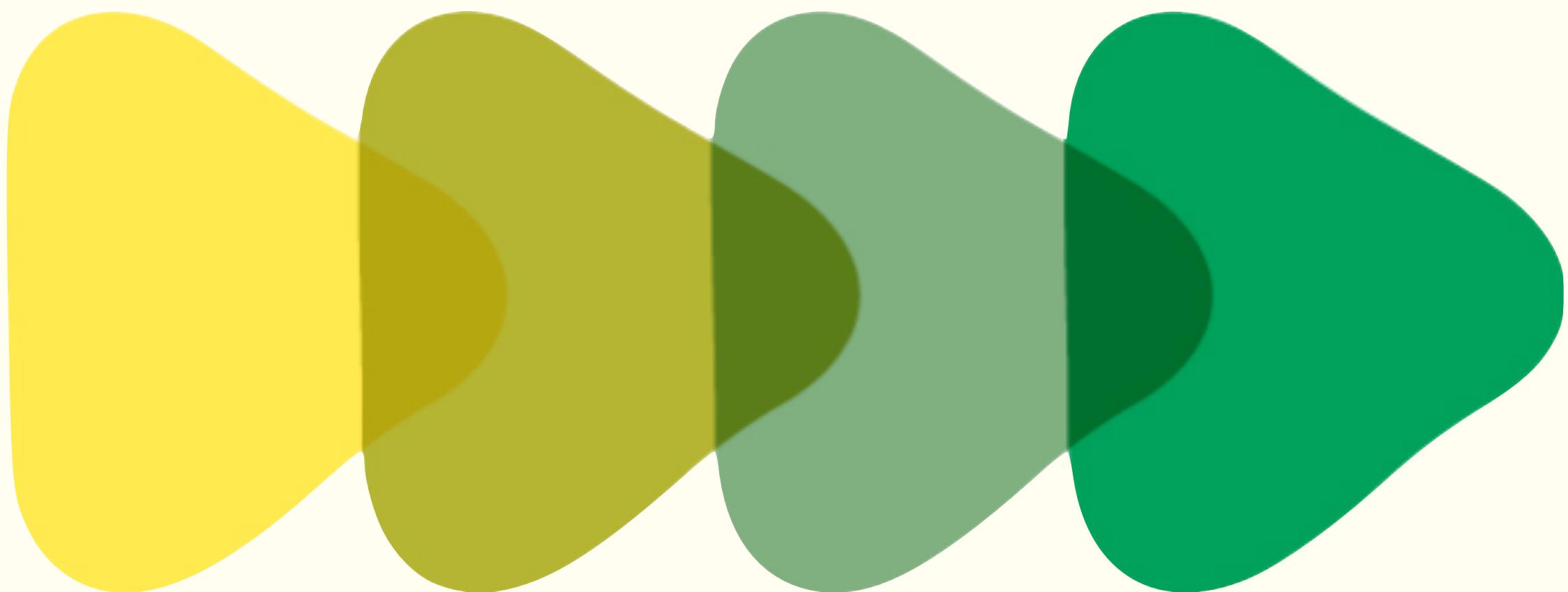
functional testing

integration testing

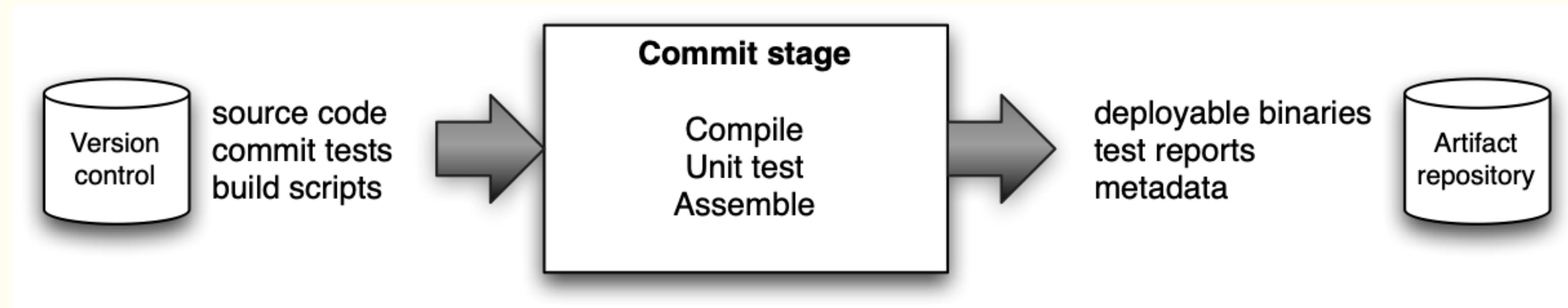
automated machine provisioning

deployment to production?

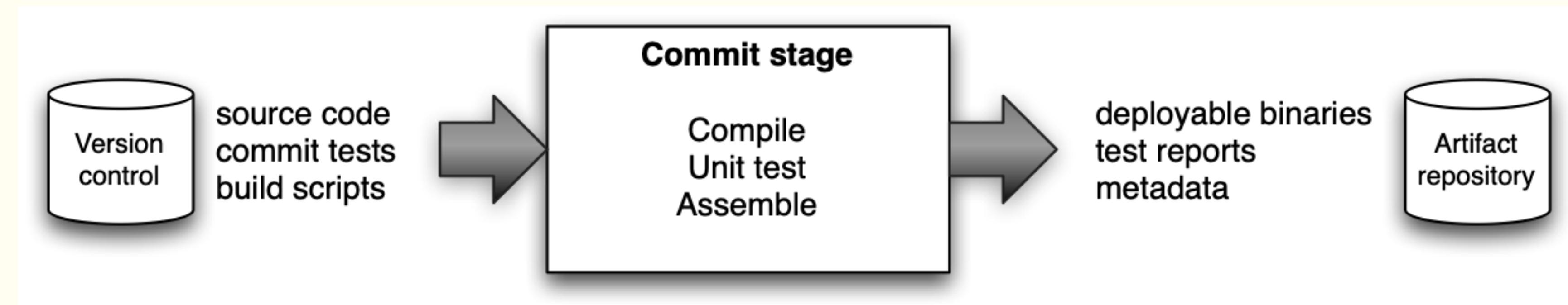
# Deployment Pipelines



# *commit* Stage

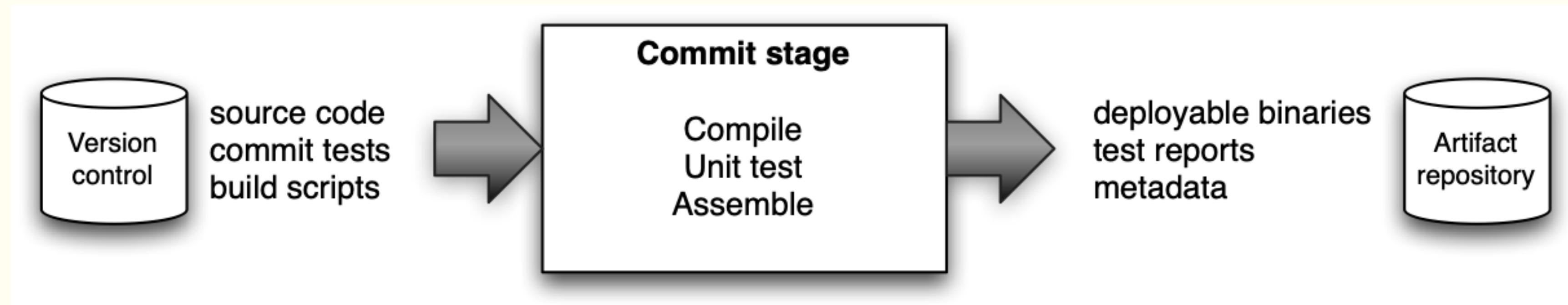


# *commit* Stage

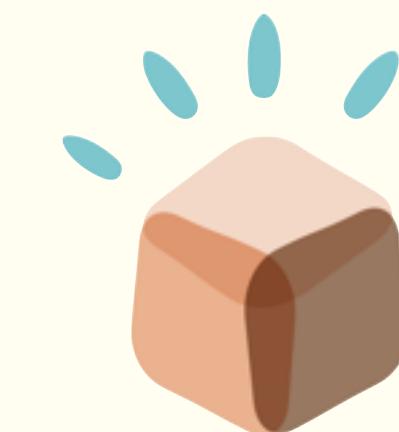


Run against each check-in

# *commit* Stage

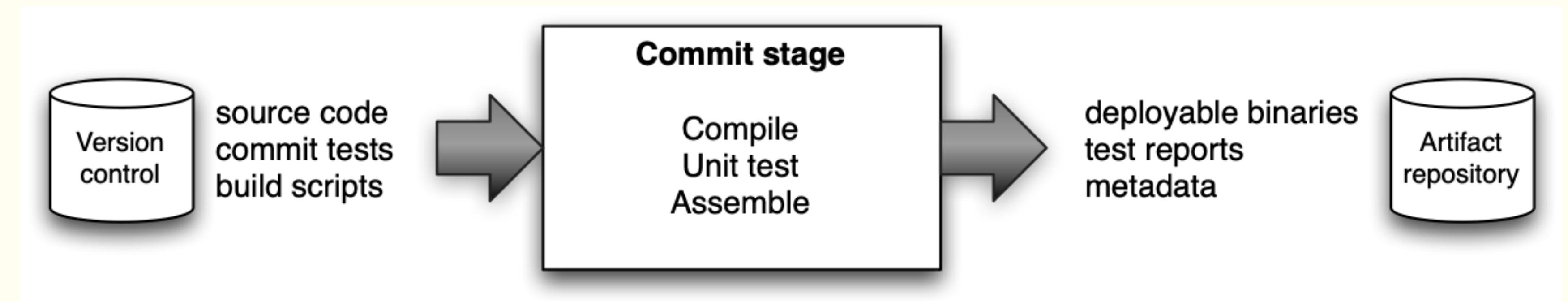


Run against each check-in



Starts building a release candidate

# *commit* Stage



Run against each check-in

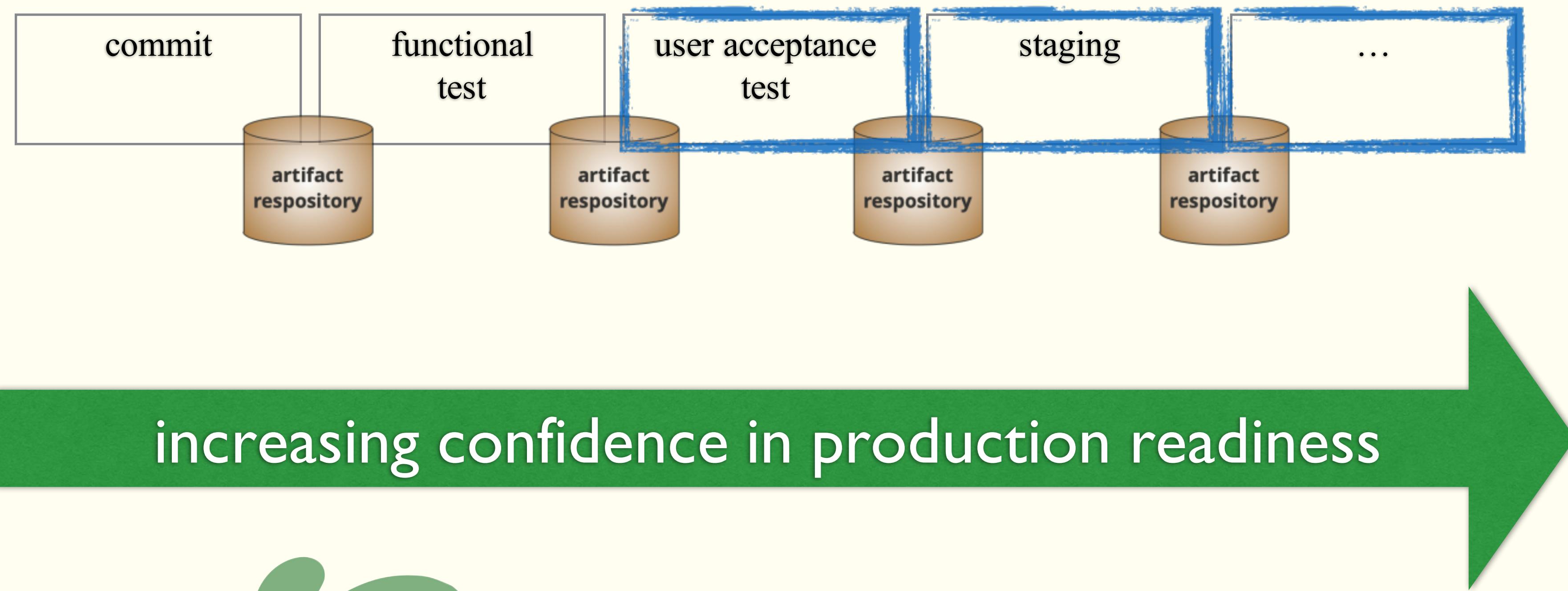


Starts building a release candidate

If it fails, fix it immediately

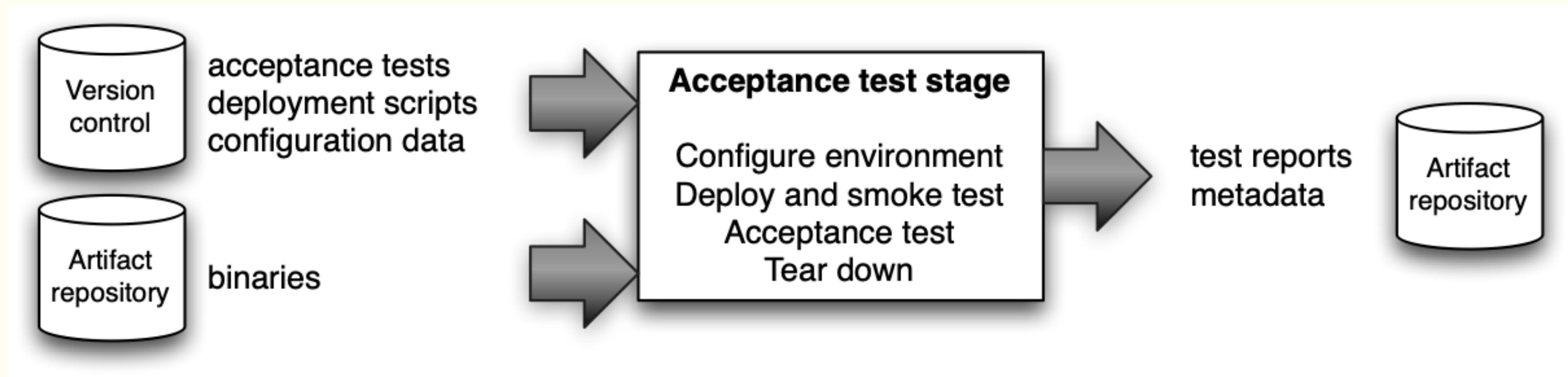


# Pipeline Construction

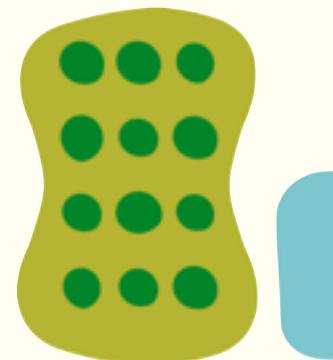
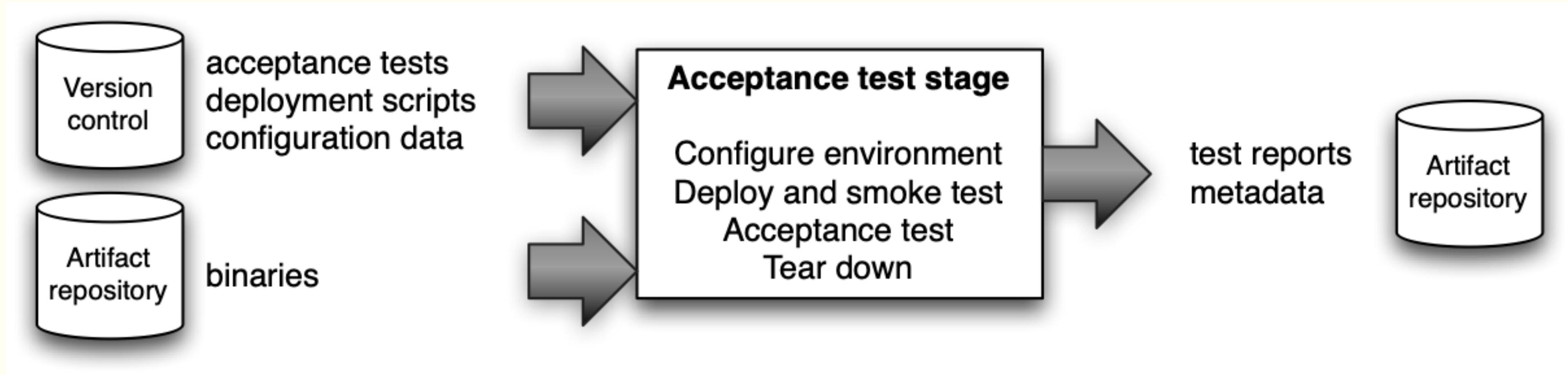


Pipeline stages = feedback opportunities

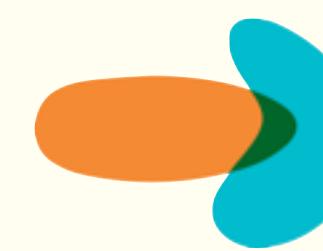
# *UAT Stage*



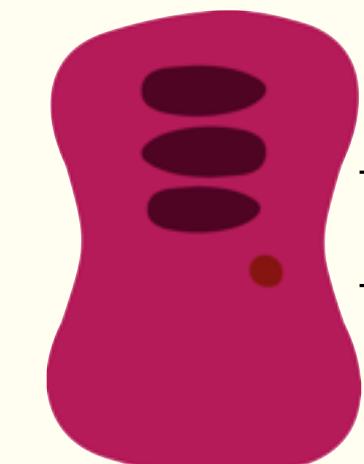
# UAT Stage



End-to-end tests in production-like environment

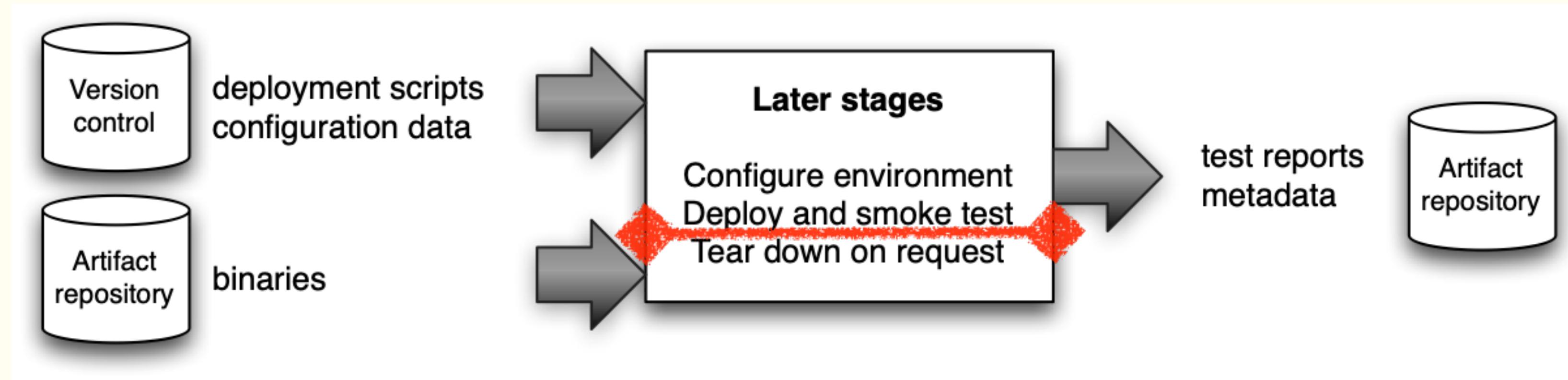


Triggered when upstream stage passes

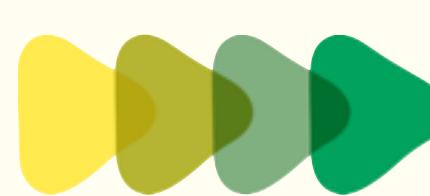
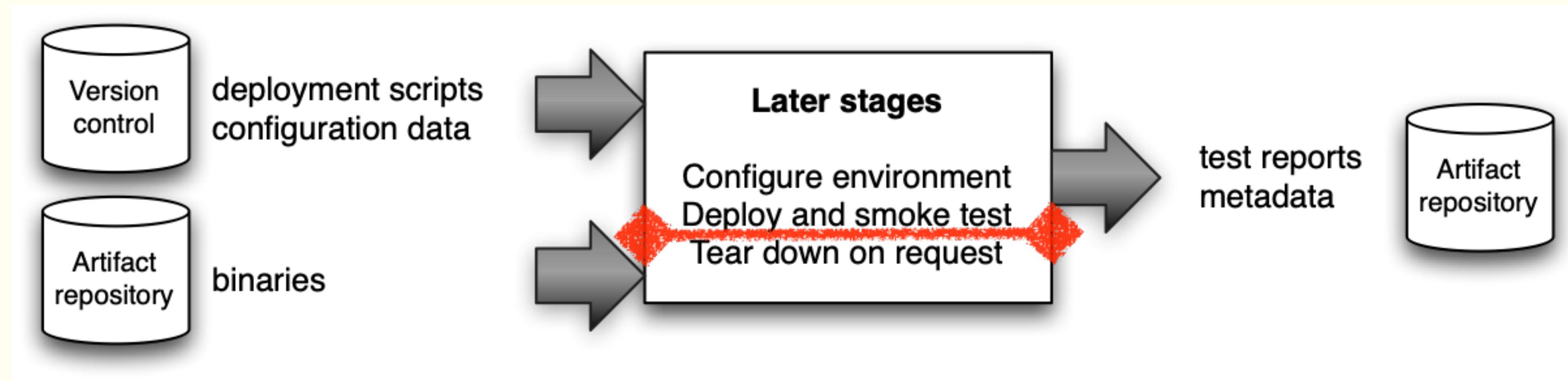


First DevOps-centric build

# *Manual Stage*

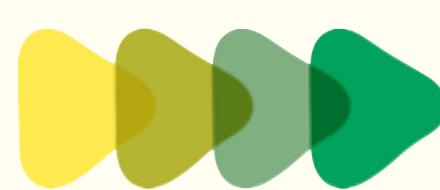
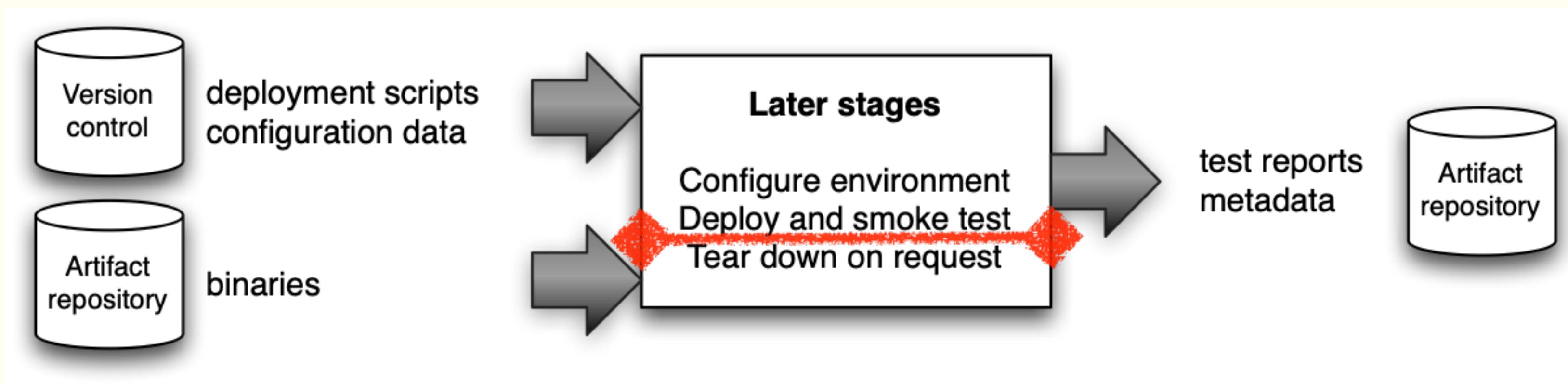


# *Manual Stage*



UAT, staging, integration, production, ...

# *Manual Stage*



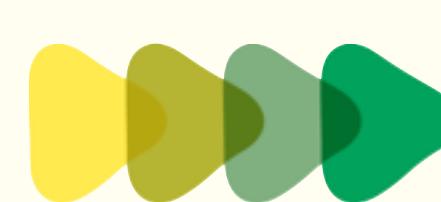
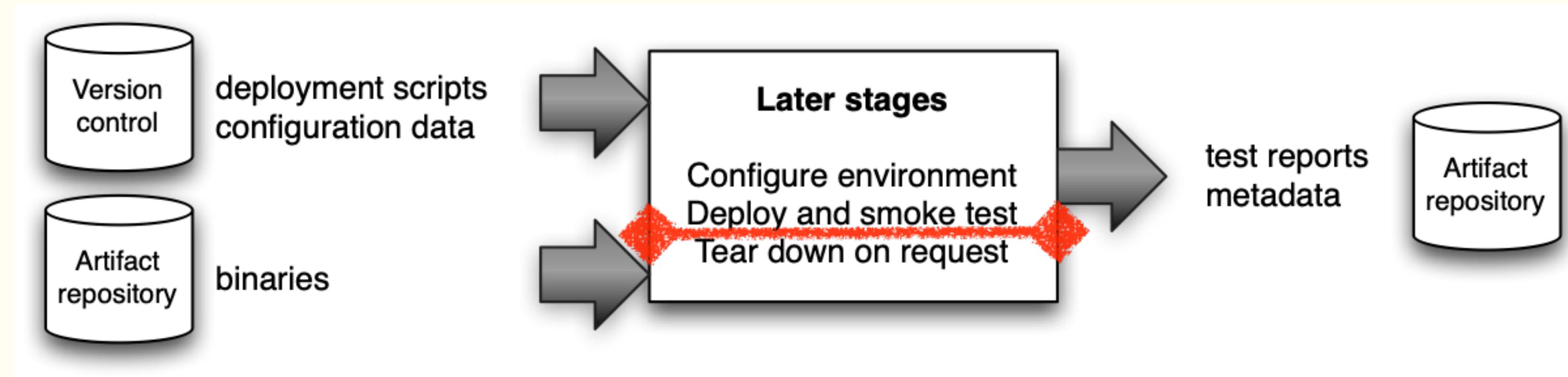
UAT, staging, integration, production, ...



Push versus Pull model



# *Manual Stage*



UAT, staging, integration, production, ...

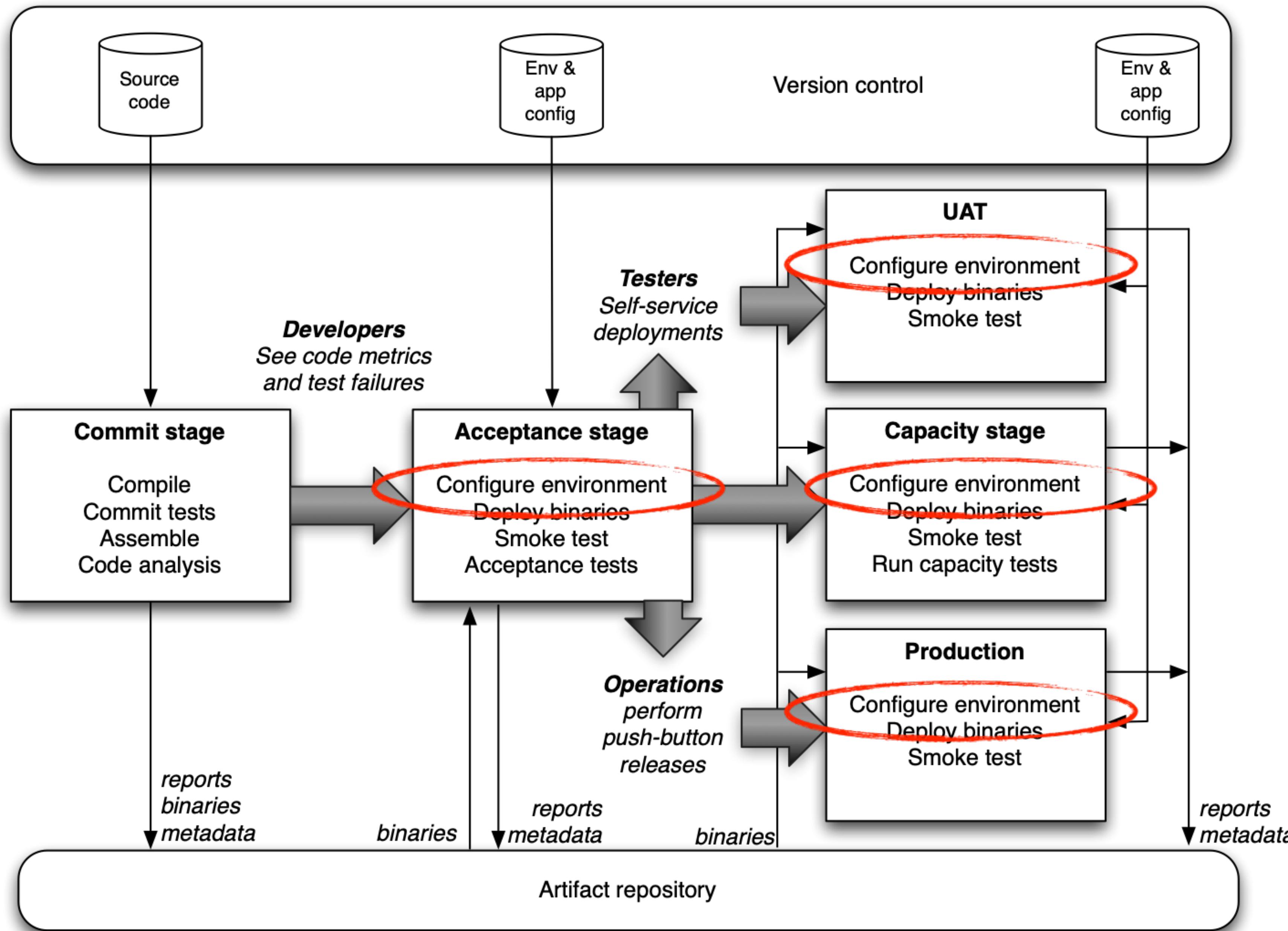


Push versus Pull model



Deployments self-serviced through push-button process





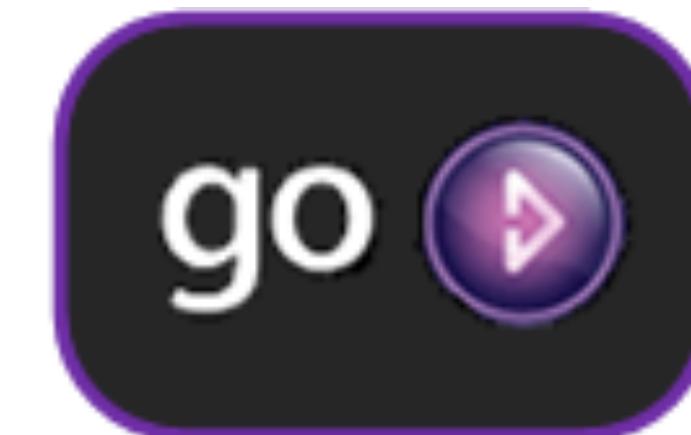
# Machinery

continuous integration ++



<https://www.gocd.org>

Go Server



Go Agents



Deploy

Environments



User Acceptance



Performance

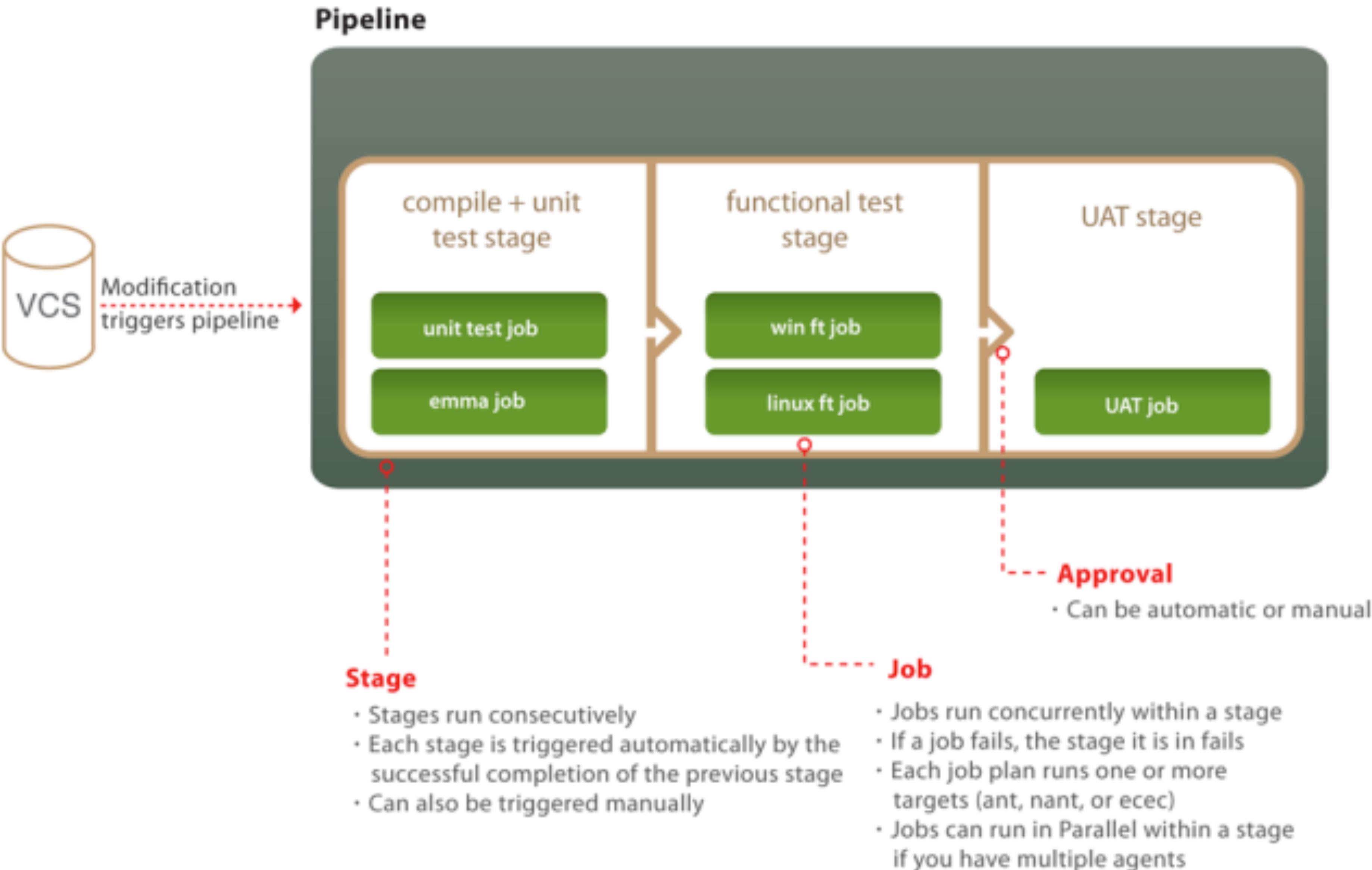


Deploy multiple  
Run smoke tests

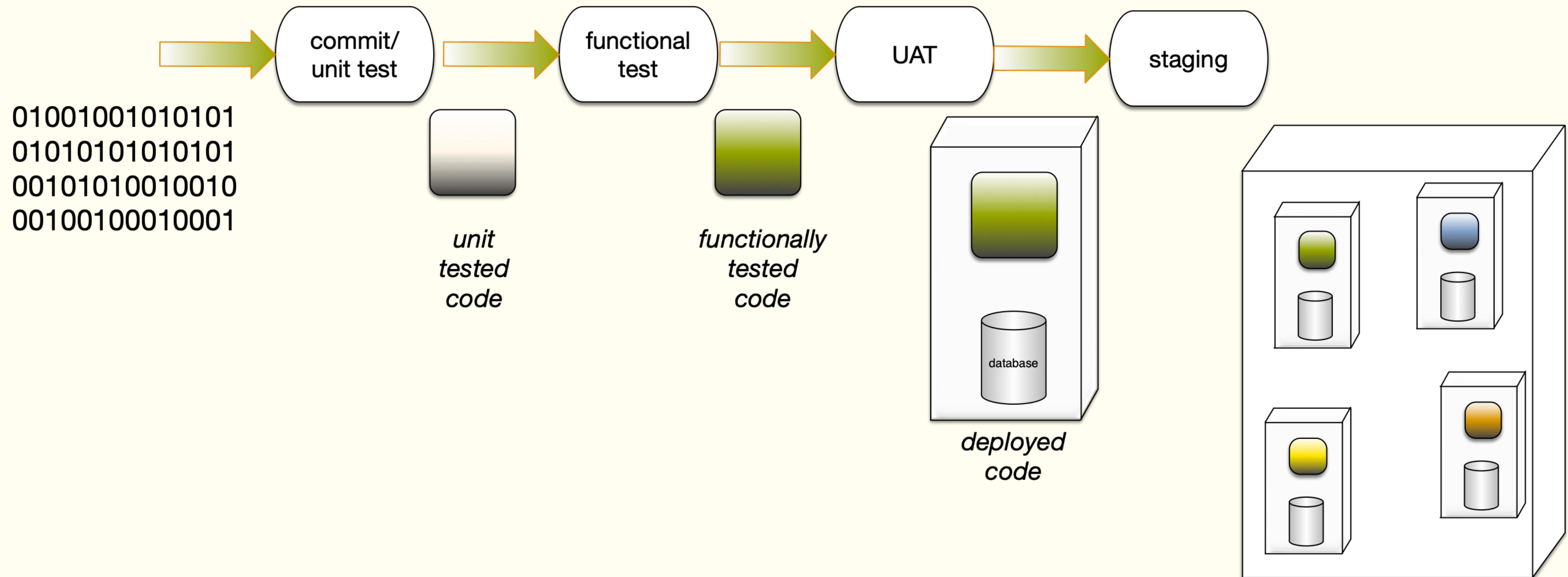


Production

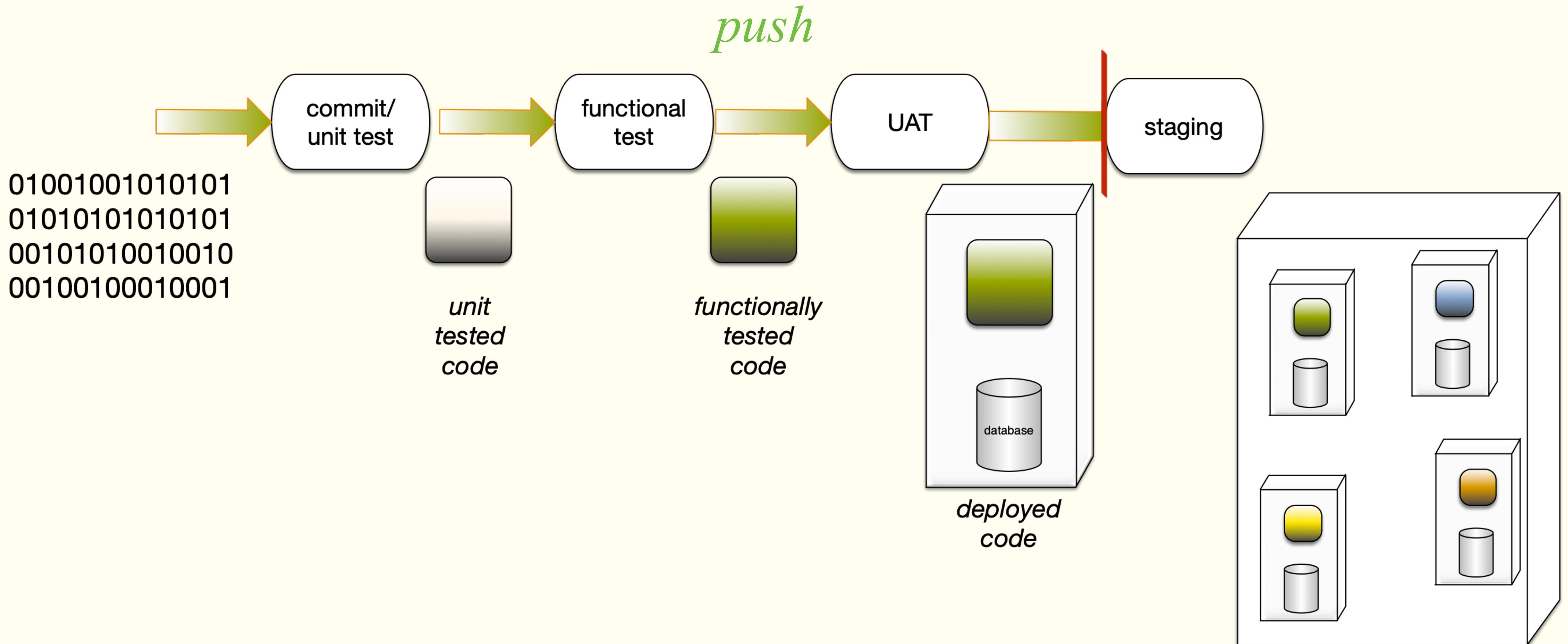
Code moves from check-in tests into UAT



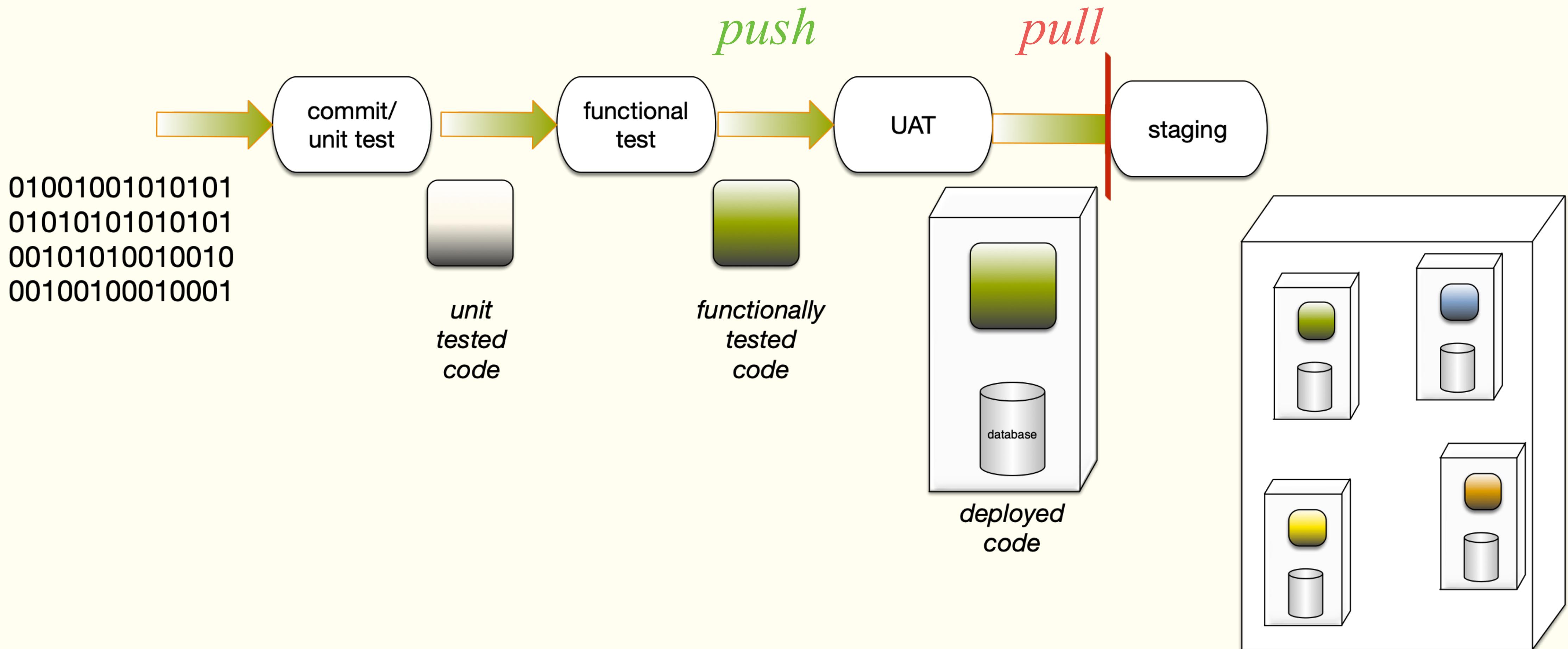
# Deployment Pipeline

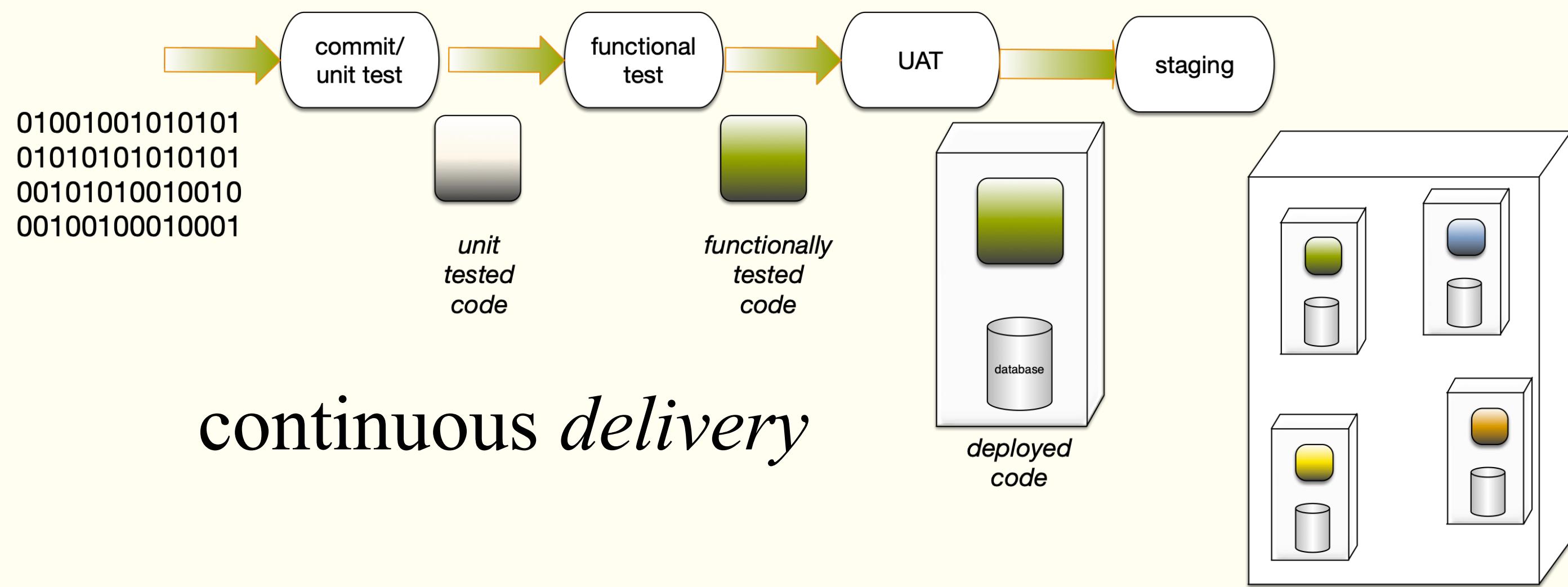


# Deployment vs Delivery?



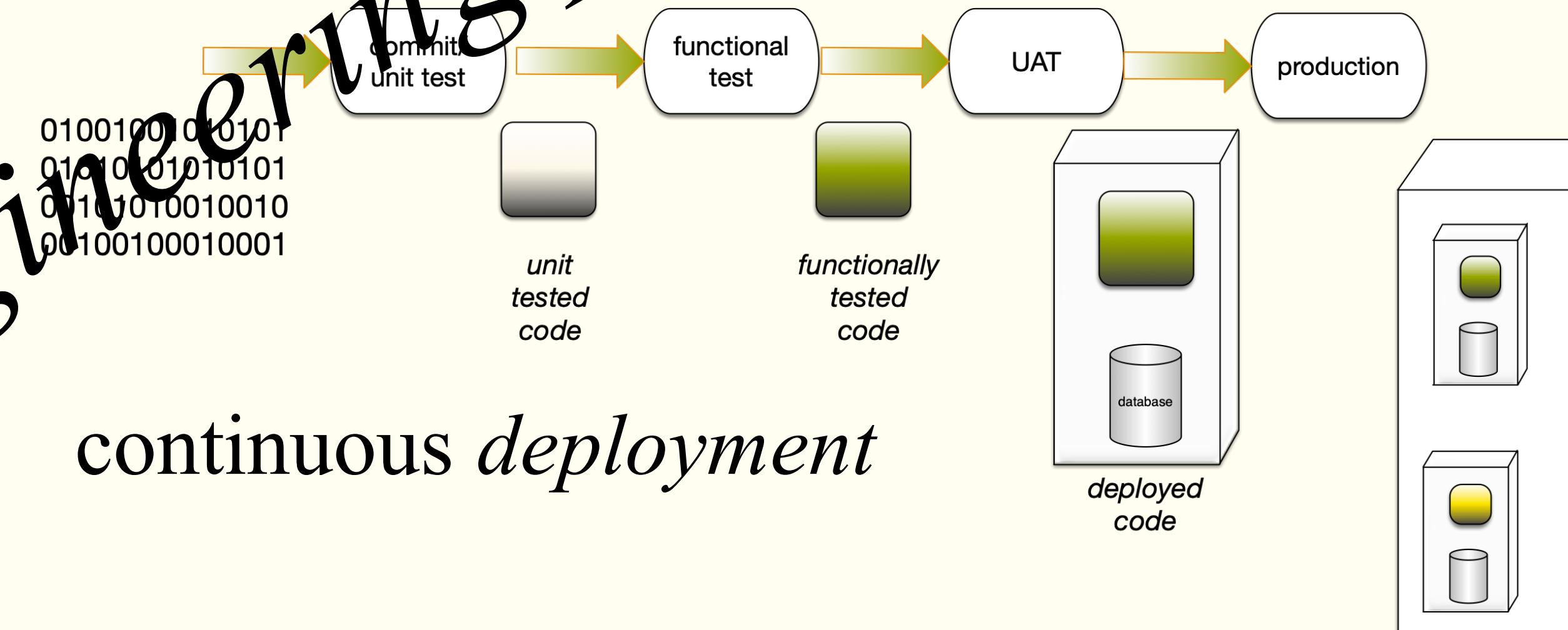
# Deployment vs Delivery?





*continuous delivery*

*engineering practices*



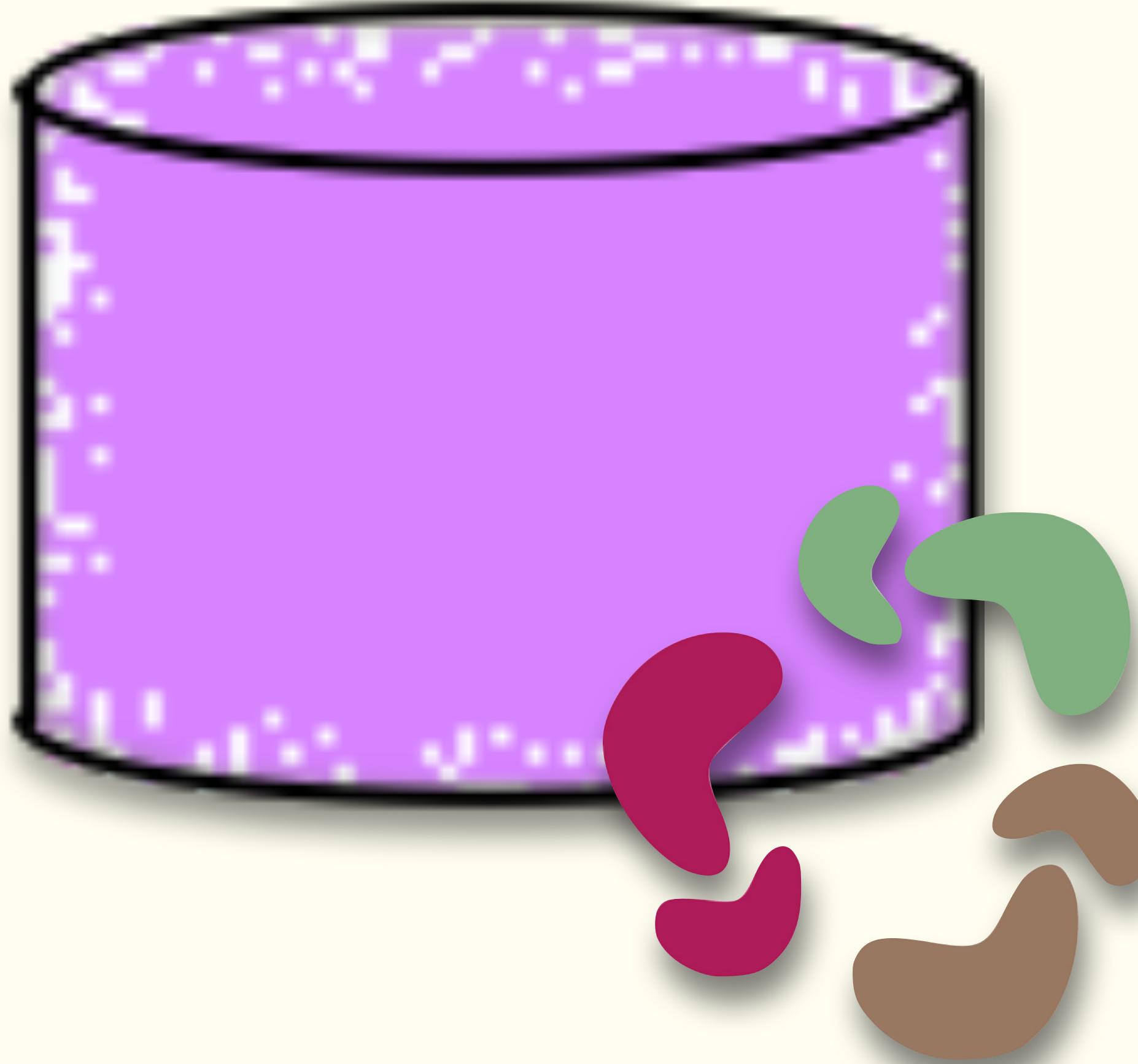
*Continuous deployment: all*

*push stages*

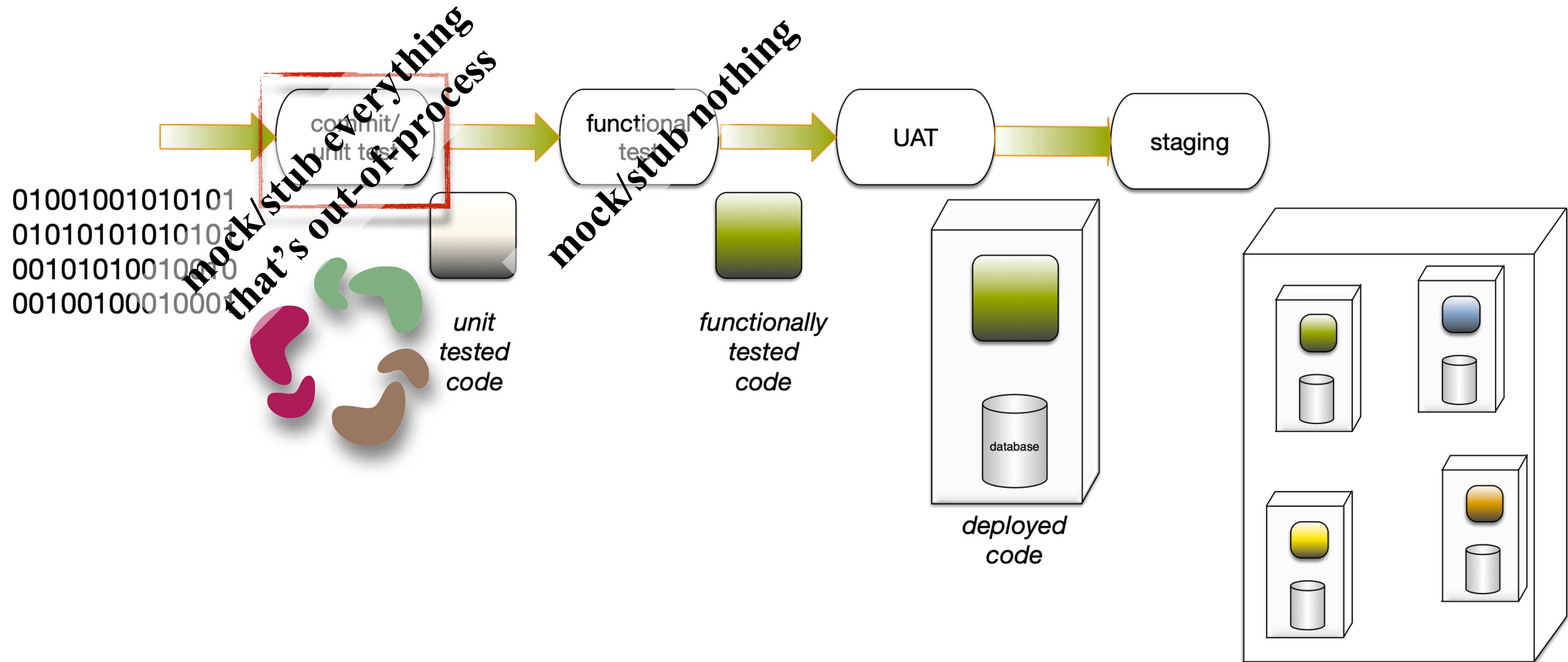
*Continuous delivery: at least one*

*push stage*

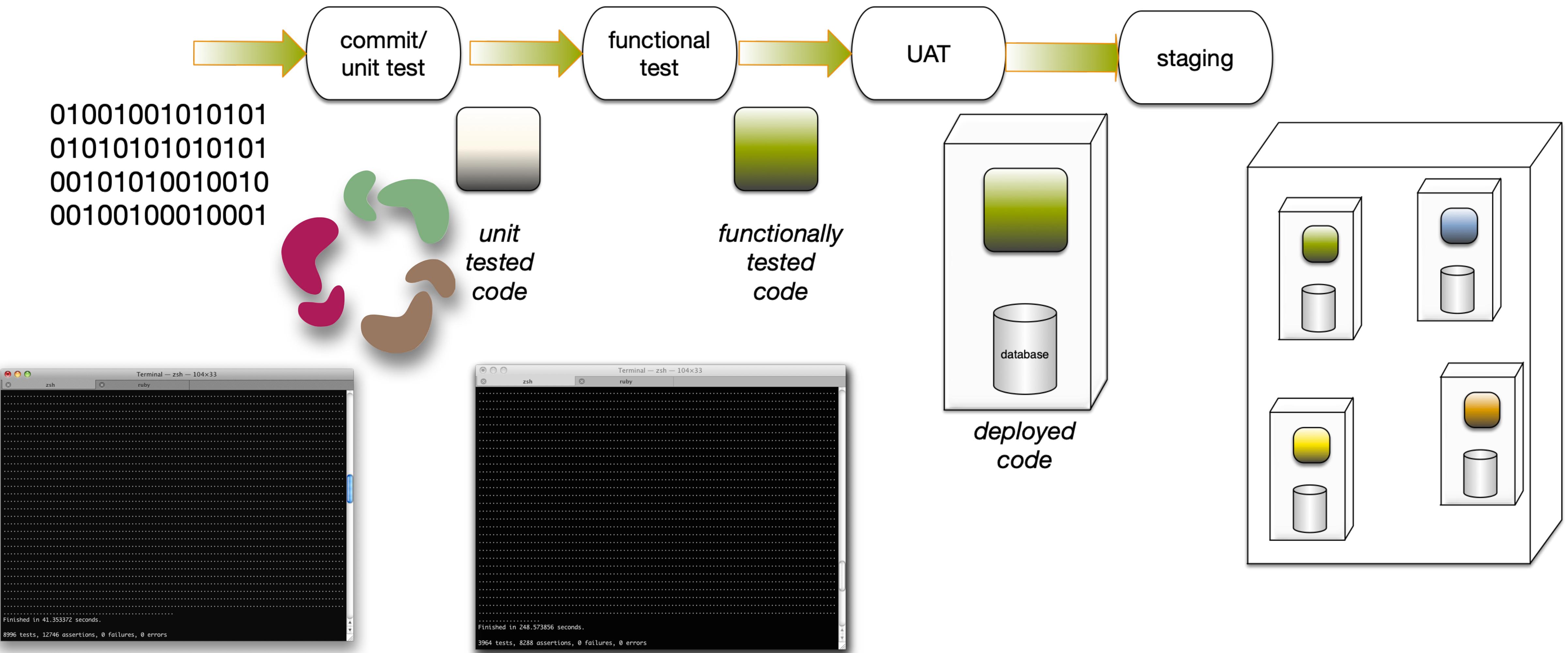
# Continuous Integration for Databases



# Fast/Slow Tests



# Fast/Slow Tests



commit/  
unit test

```
01001001010101  
01010101010101  
00101010010010  
00100100010001
```

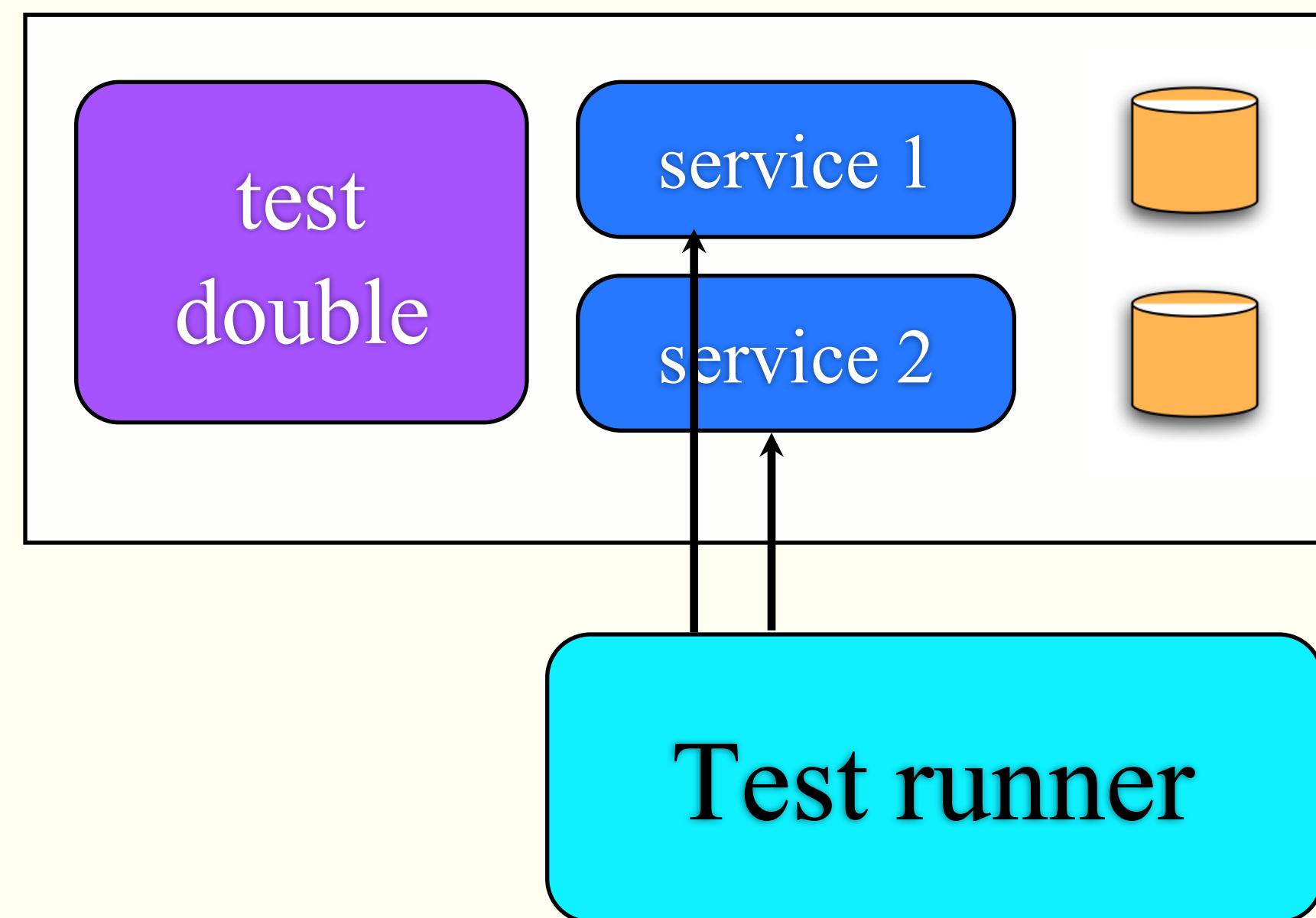
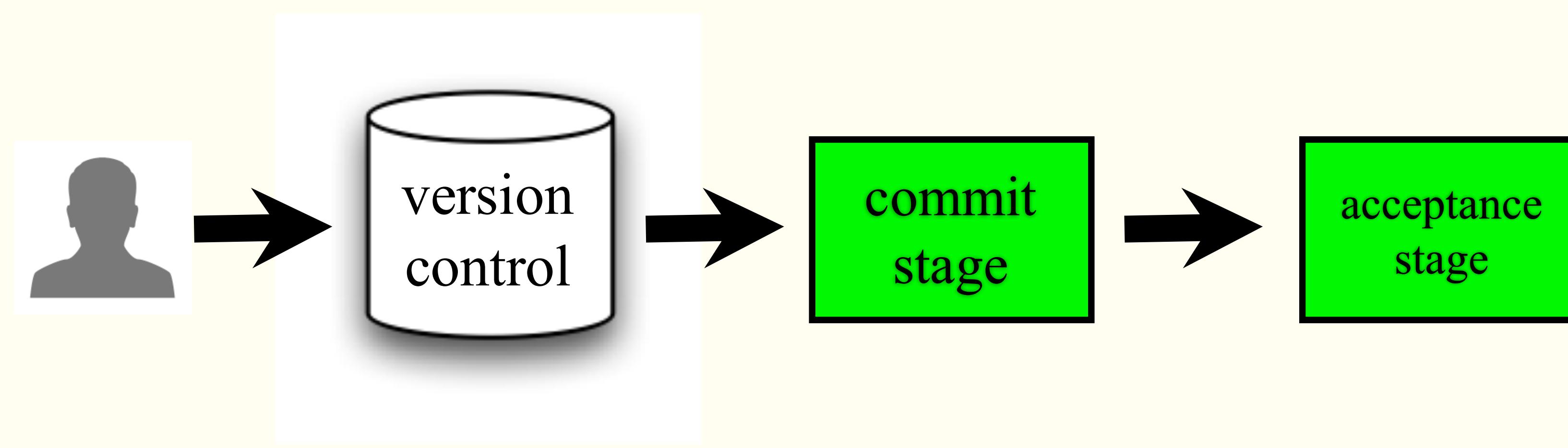
uites  
co

.....  
Finished in 248.573856 seconds.

..... 3964 tests, 8288 assertions, 0 failures, 0 errors

.....  
Finished in 41.353372 seconds.

..... 8996 tests, 12746 assertions, 0 failures, 0 errors



Prepare environment

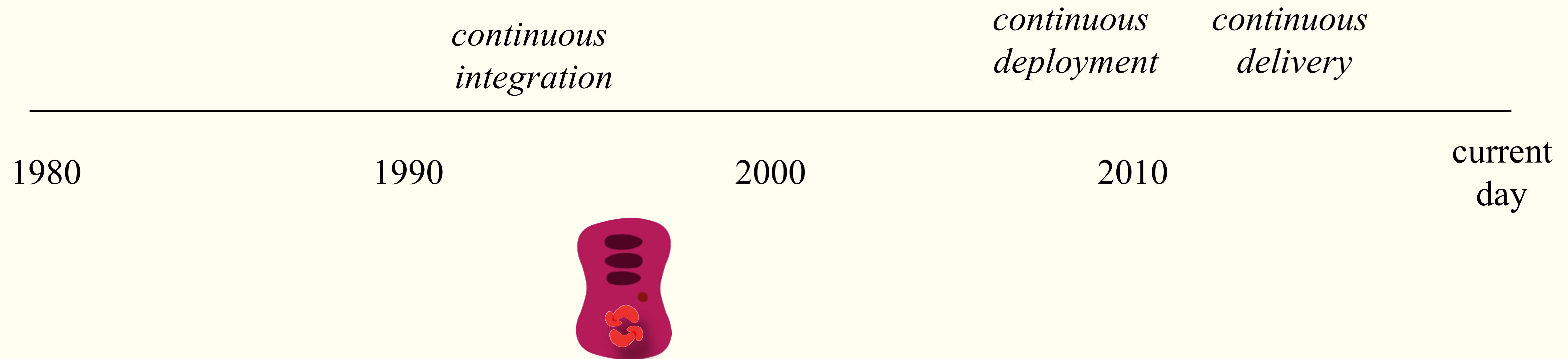
Deploy app

Create dbs, apply schema

Add app reference data

Run acceptance tests

# integration as an engineering practice over time



aRChiTeCtuRe  
fOr  
cONTinuoUs DeLiVeRy

# Question:

What characteristics of architecture make continuous delivery/deployment easier?

# Question:

What does *agility* mean as an architecture characteristic?

*agility*

deployability

*agility*

testability

deployability

modularity

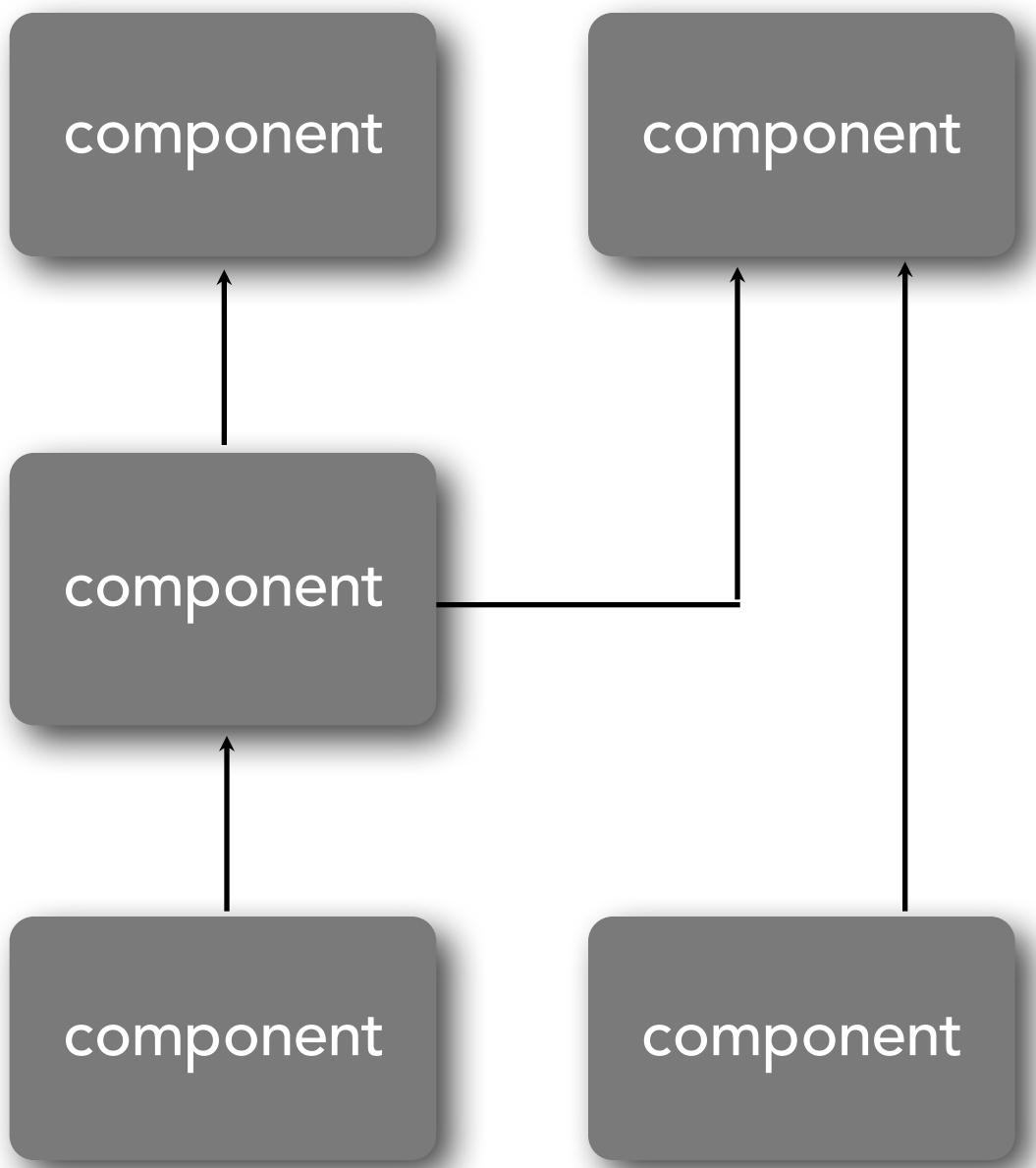
*agility*

controlled coupling

testability

# component identification

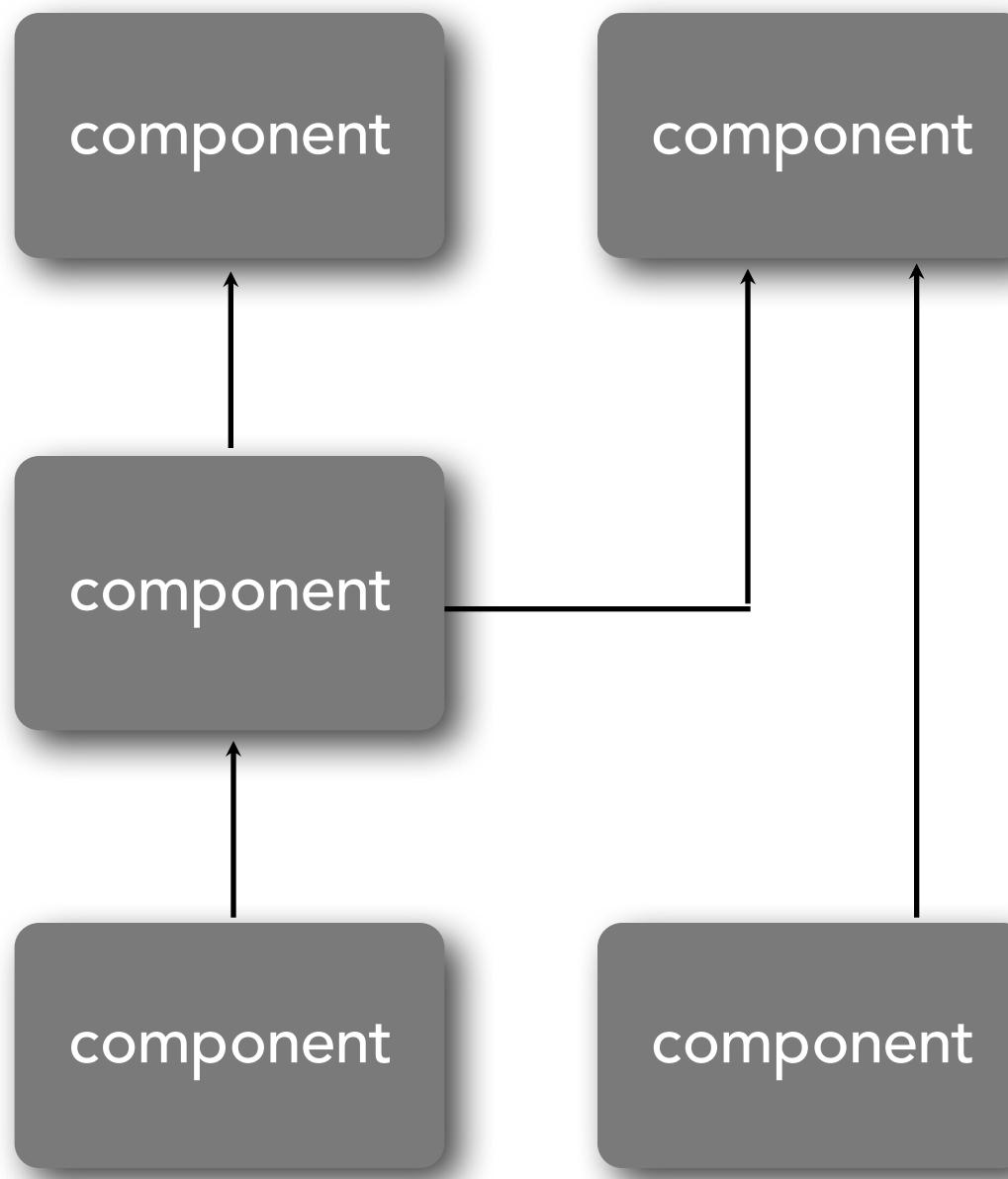
as an architect, you should think about the artifacts  
within the architecture in terms of *components*



**component:**

# component identification

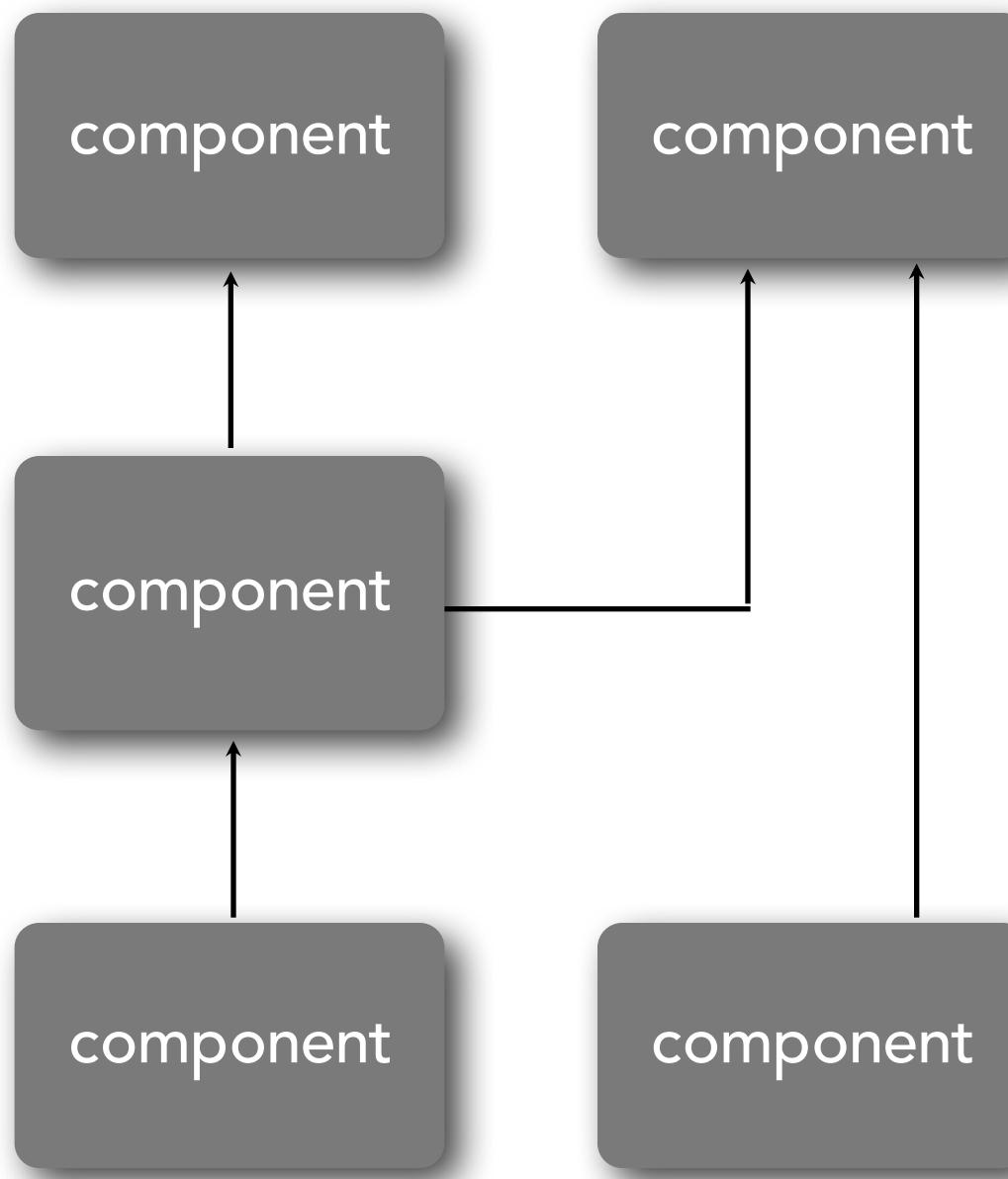
as an architect, you should think about the artifacts within the architecture in terms of *components*



**component:**  
building block of the application

# component identification

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

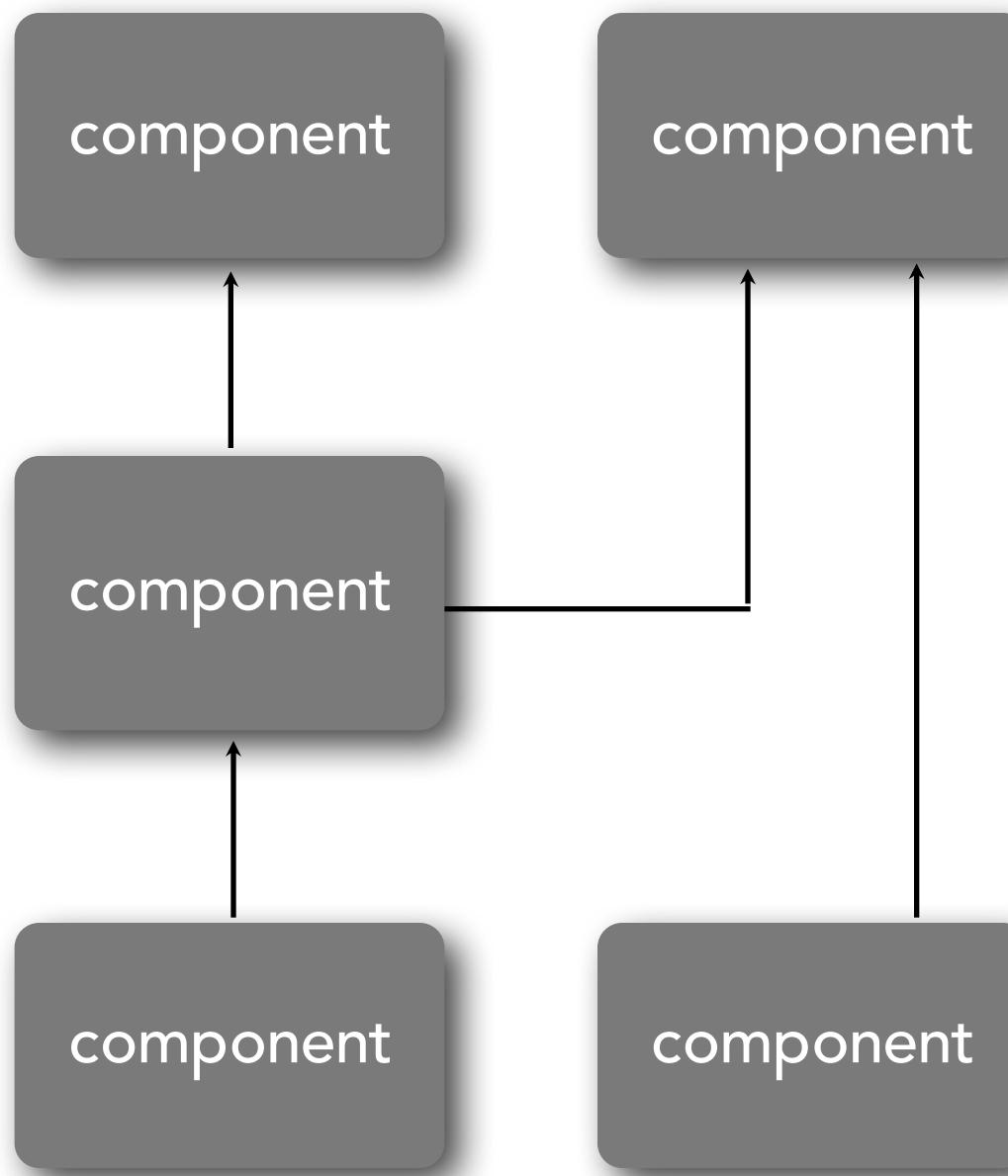


**component:**

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*

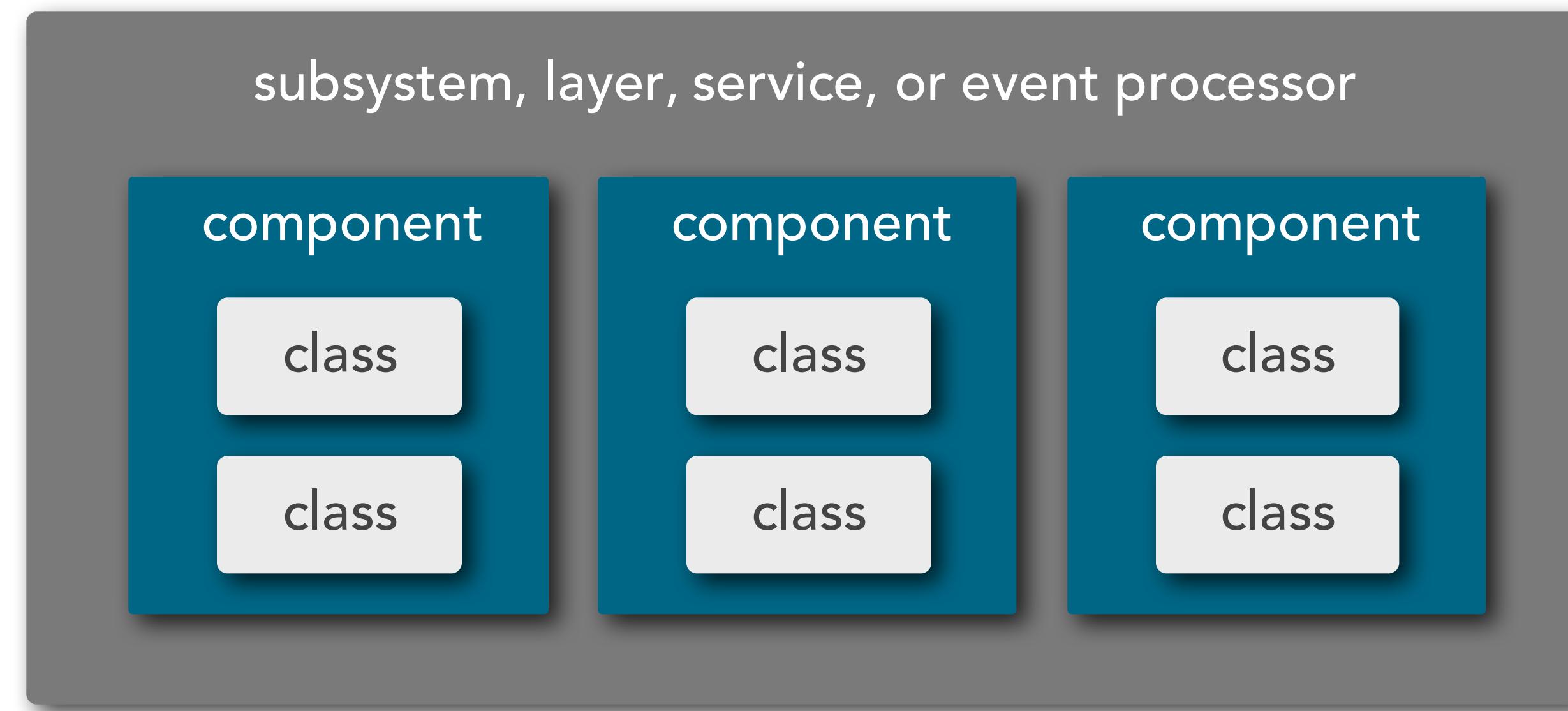


## **component:**

building block of the application  
well defined set of operations  
well defined role and responsibility

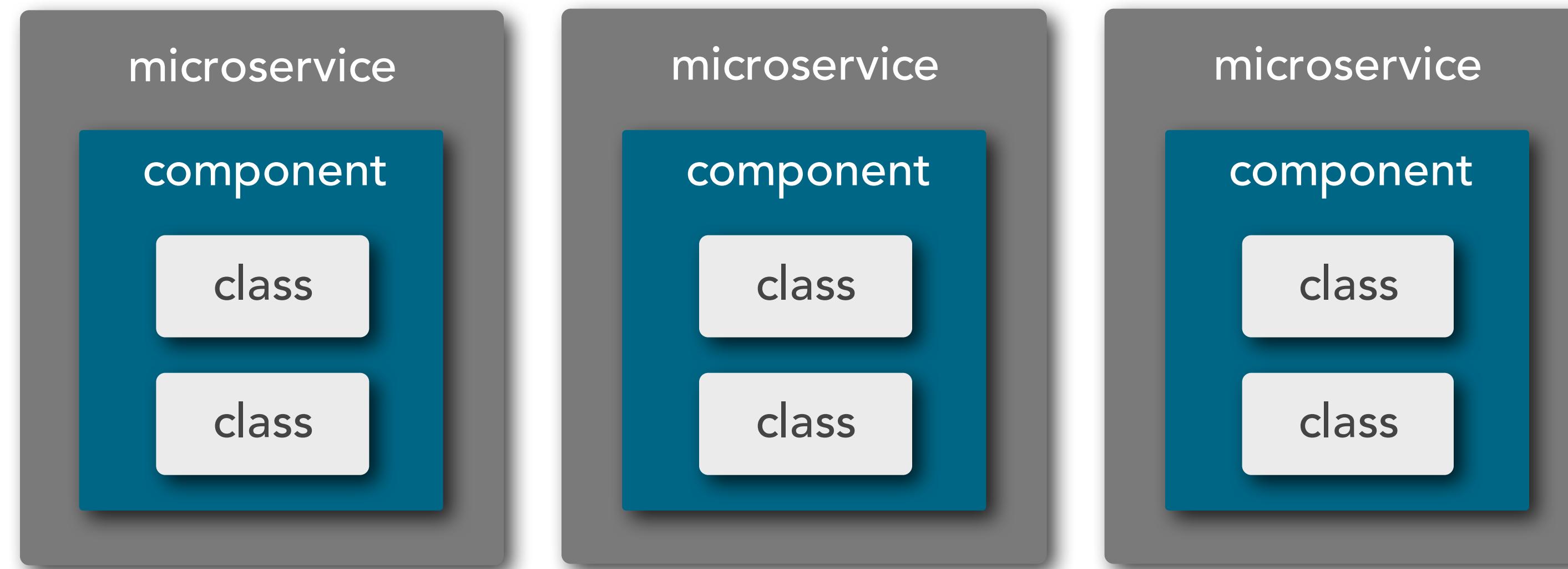
# component identification

## component scope



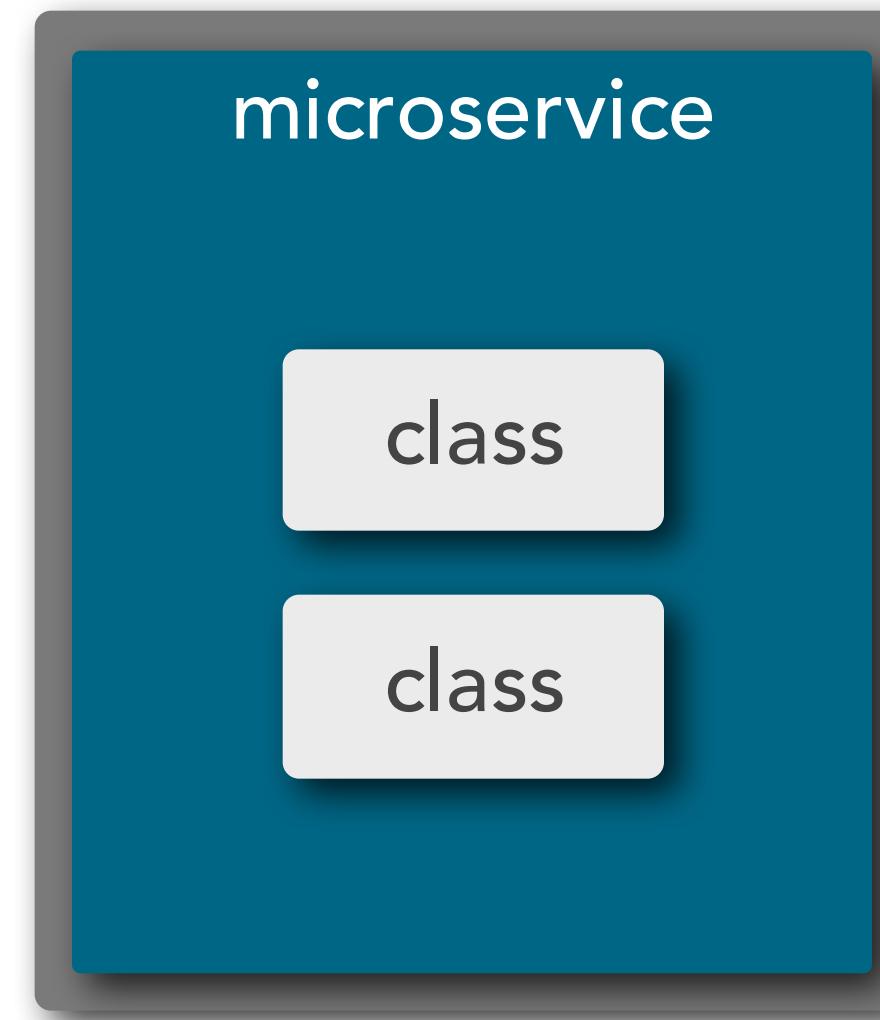
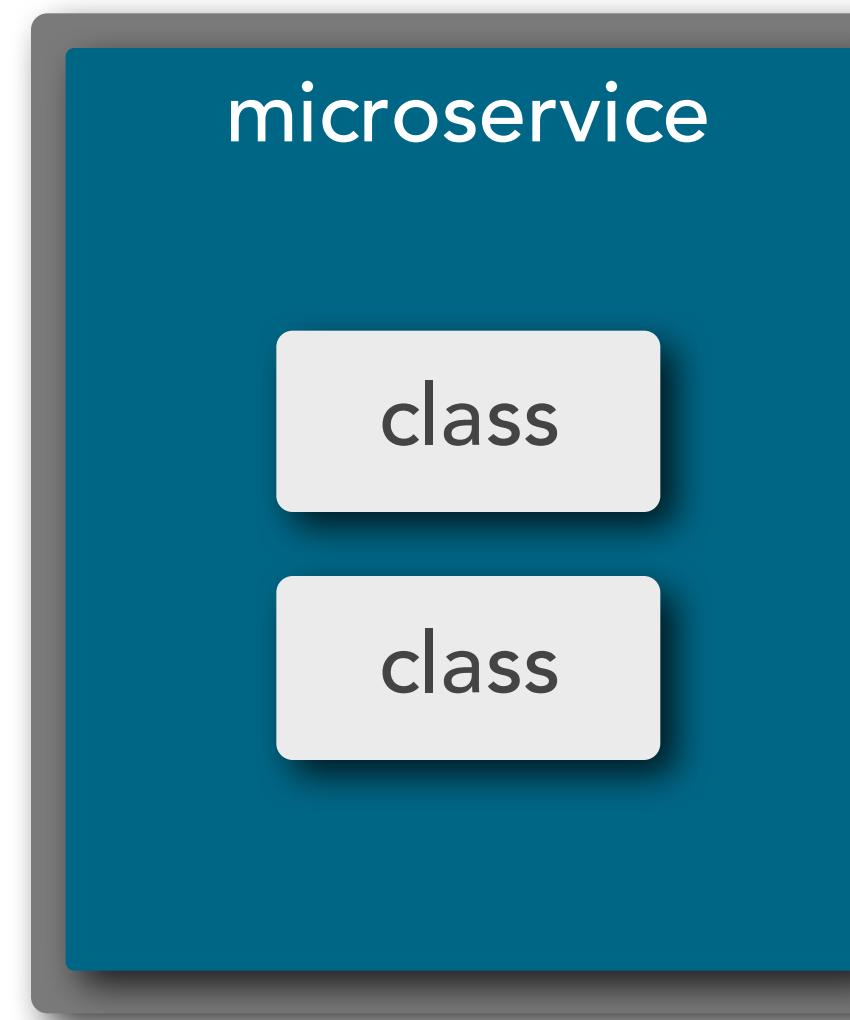
# component identification

## component scope



# component identification

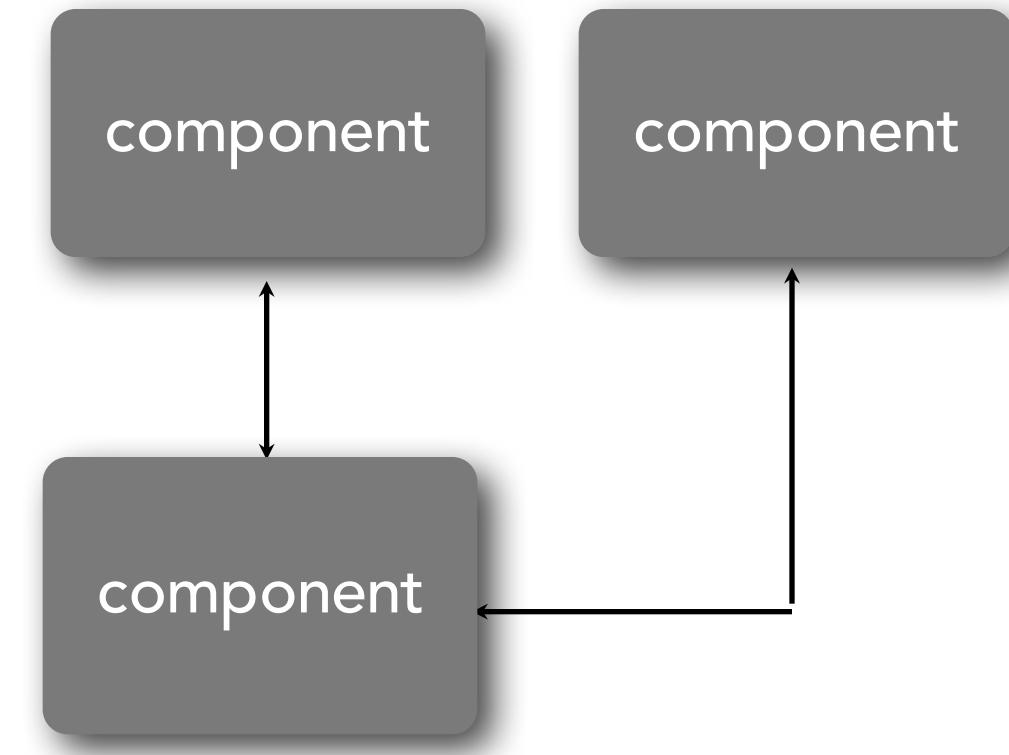
## component scope



component coupling

# component coupling

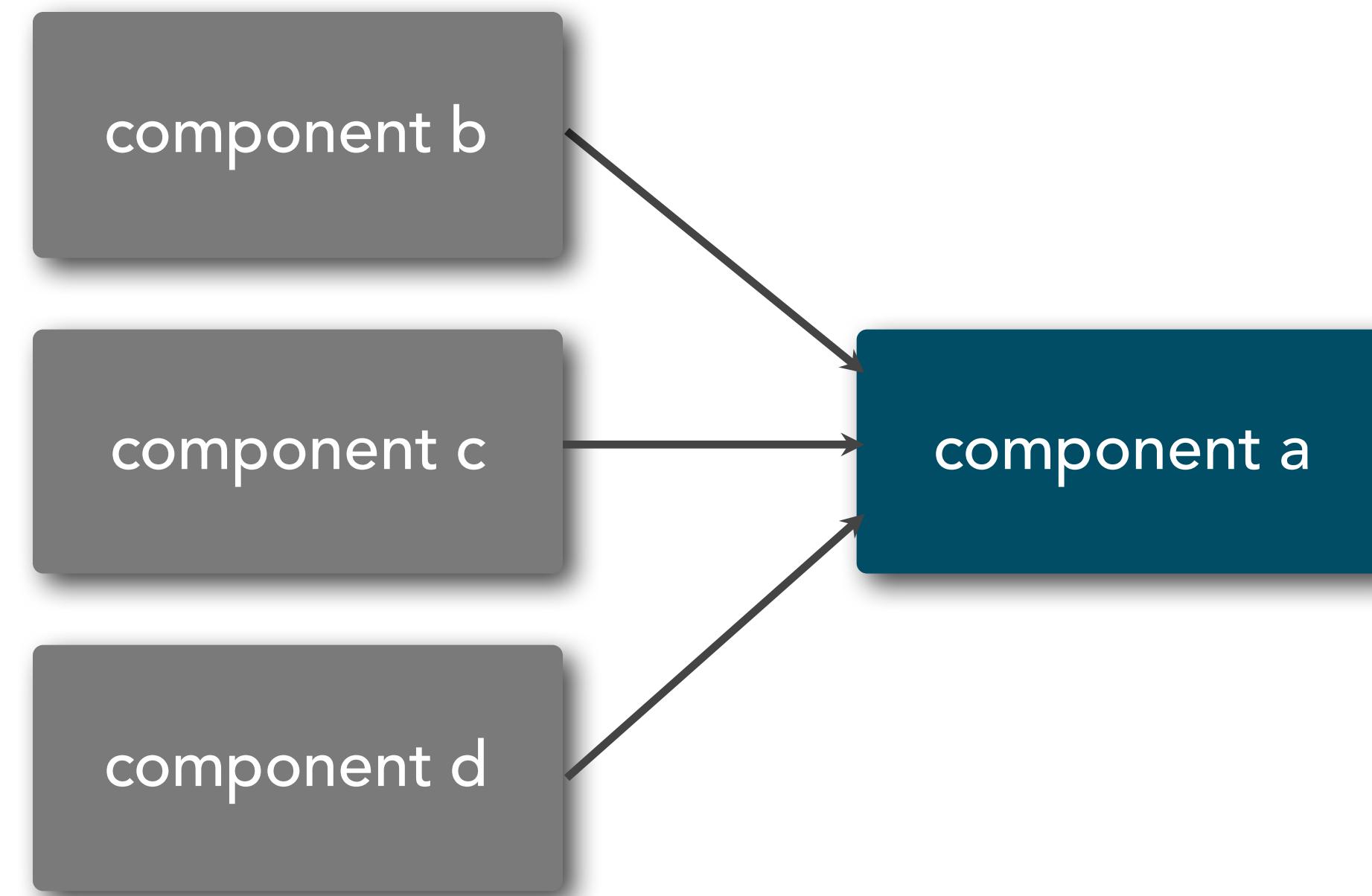
the extent to which components know  
about each other



# component coupling

## afferent coupling

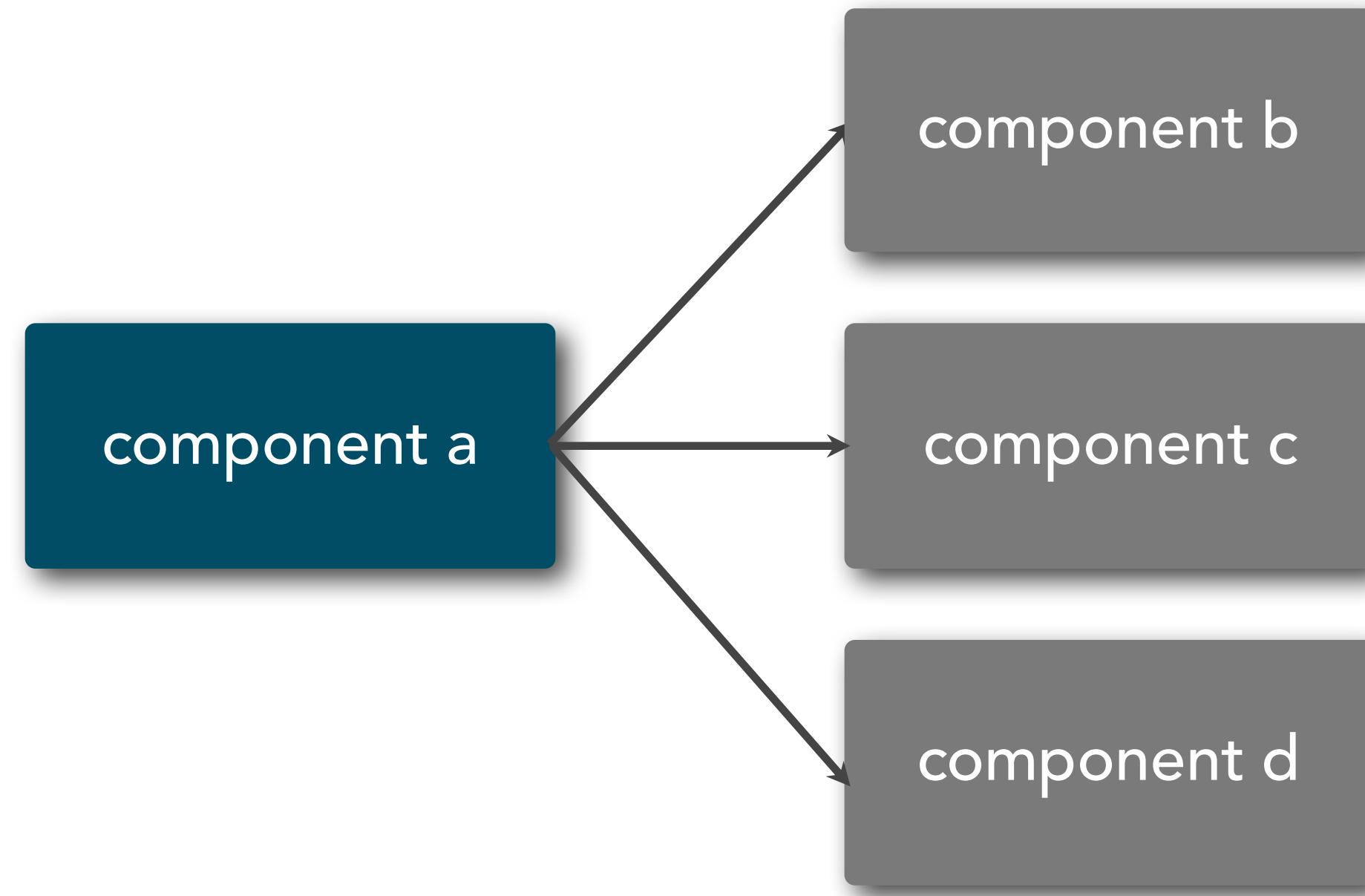
the degree to which other components are dependent on the target component



# component coupling

## efferent coupling

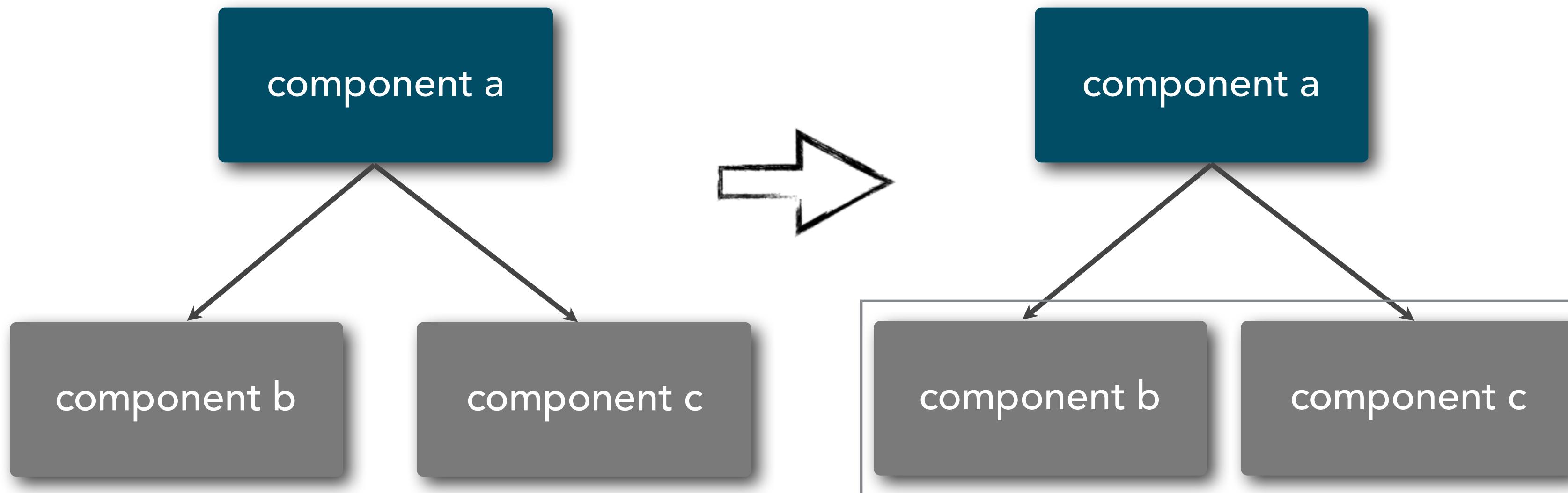
the degree to which the target component  
is dependent on other components



# component coupling

## temporal coupling

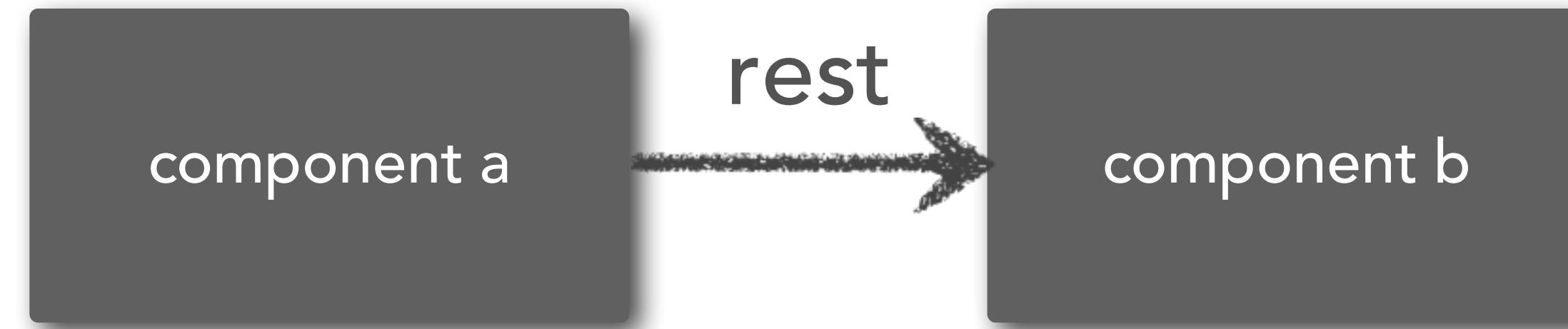
components are coupled due to non-static or timing dependencies



# component coupling

## external coupling

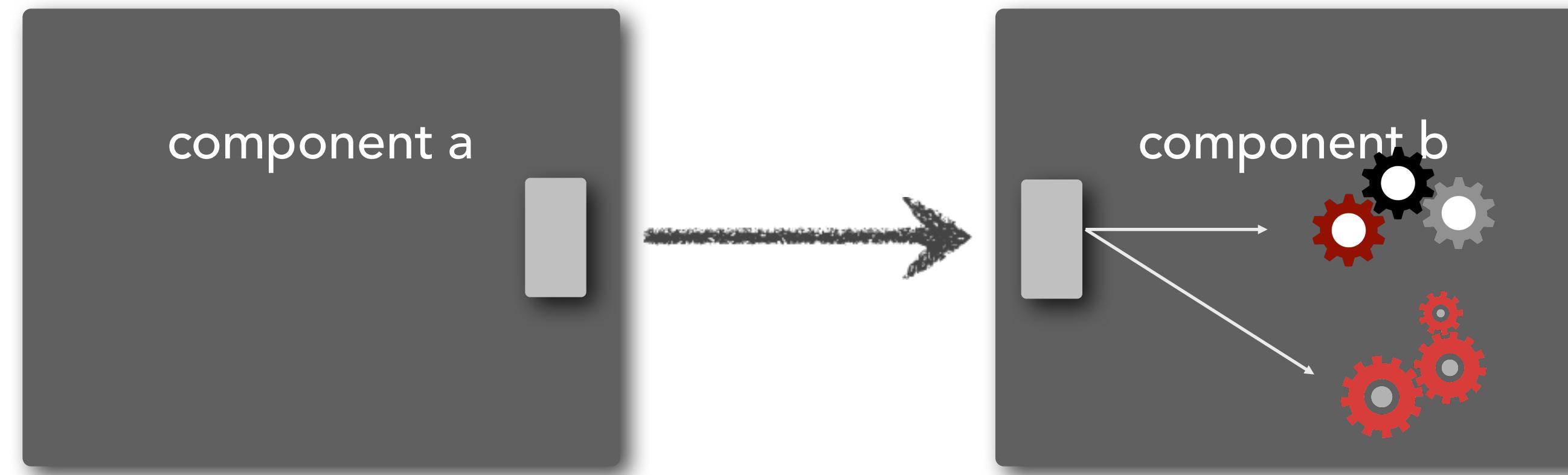
multiple components share an externally imposed protocol or data format



# component coupling

## control coupling

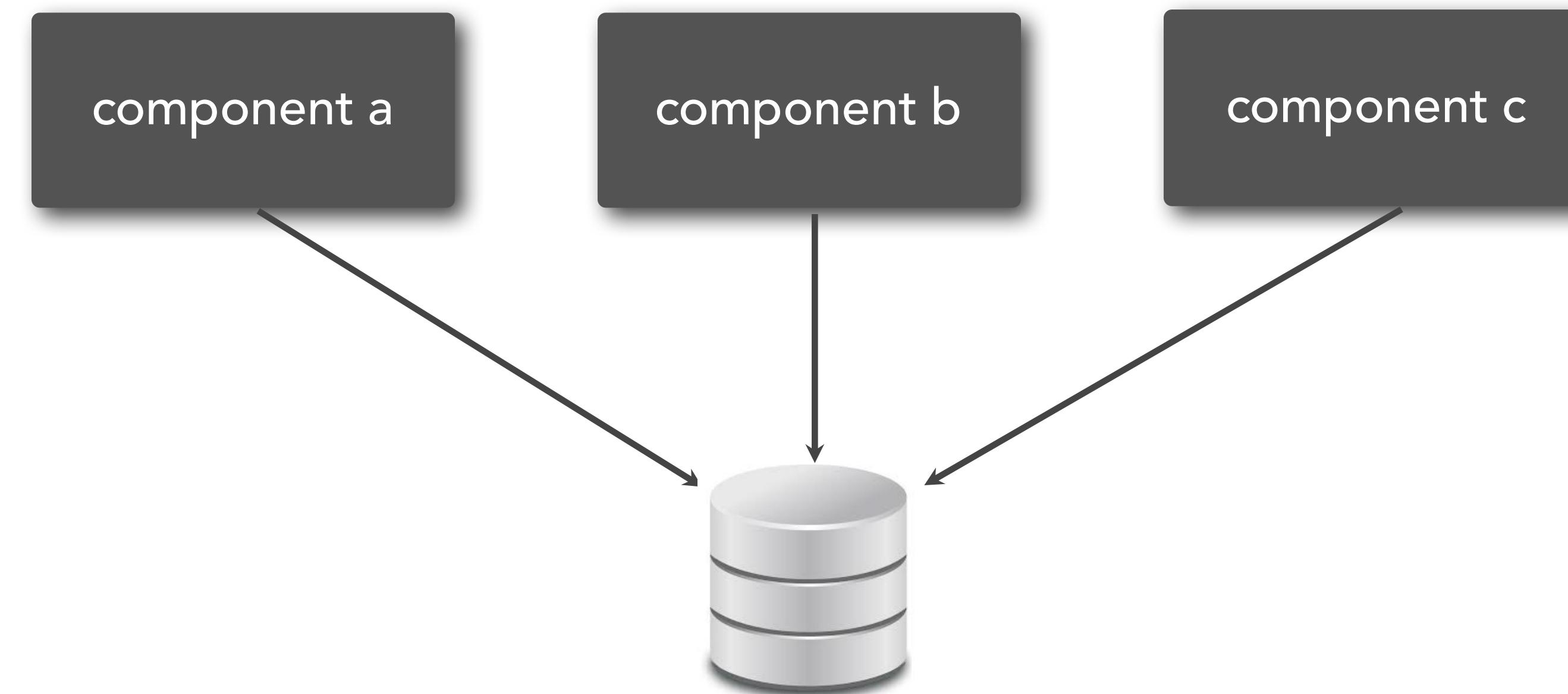
one component passes information to another component on what to do



# component coupling

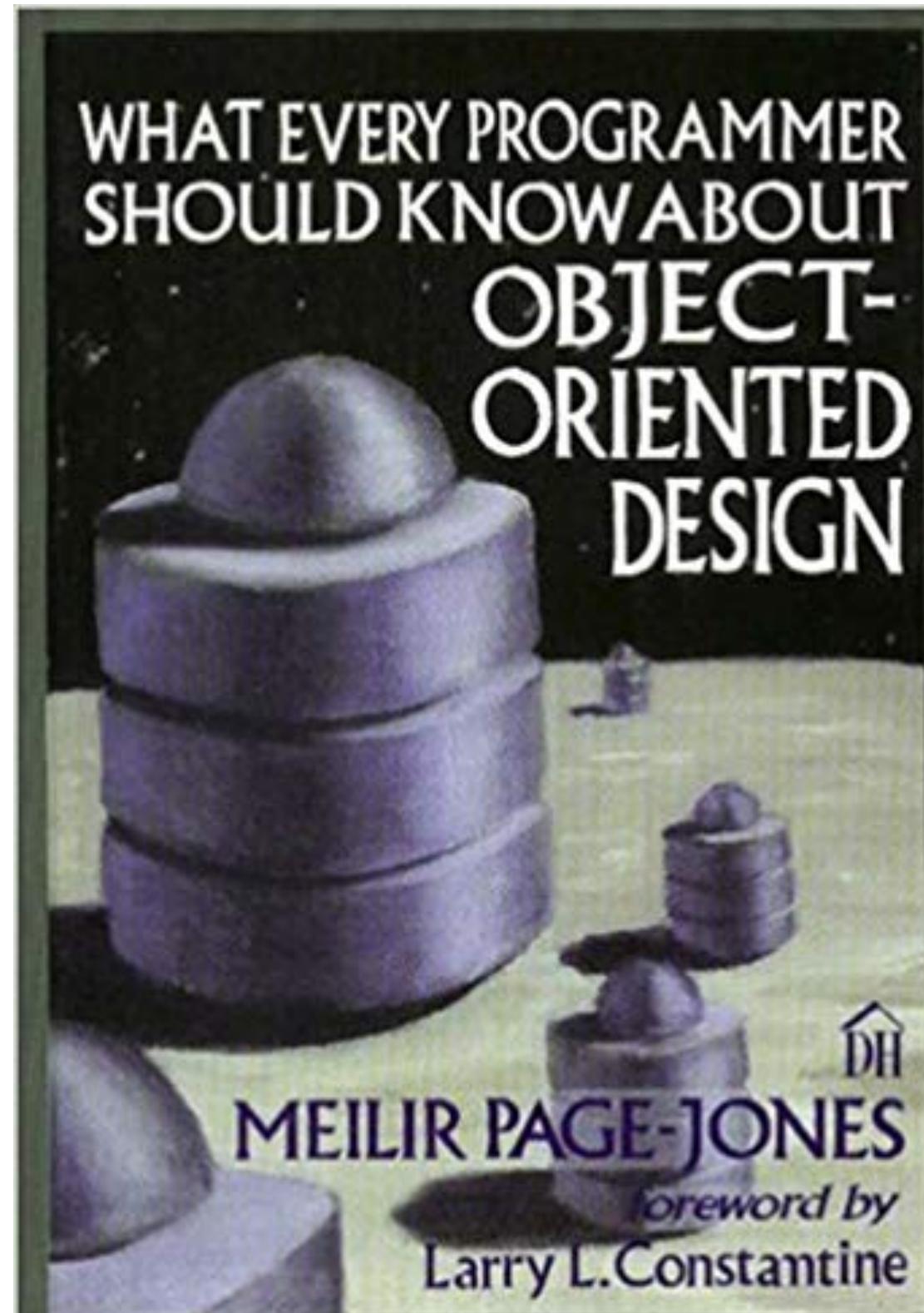
## data coupling

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

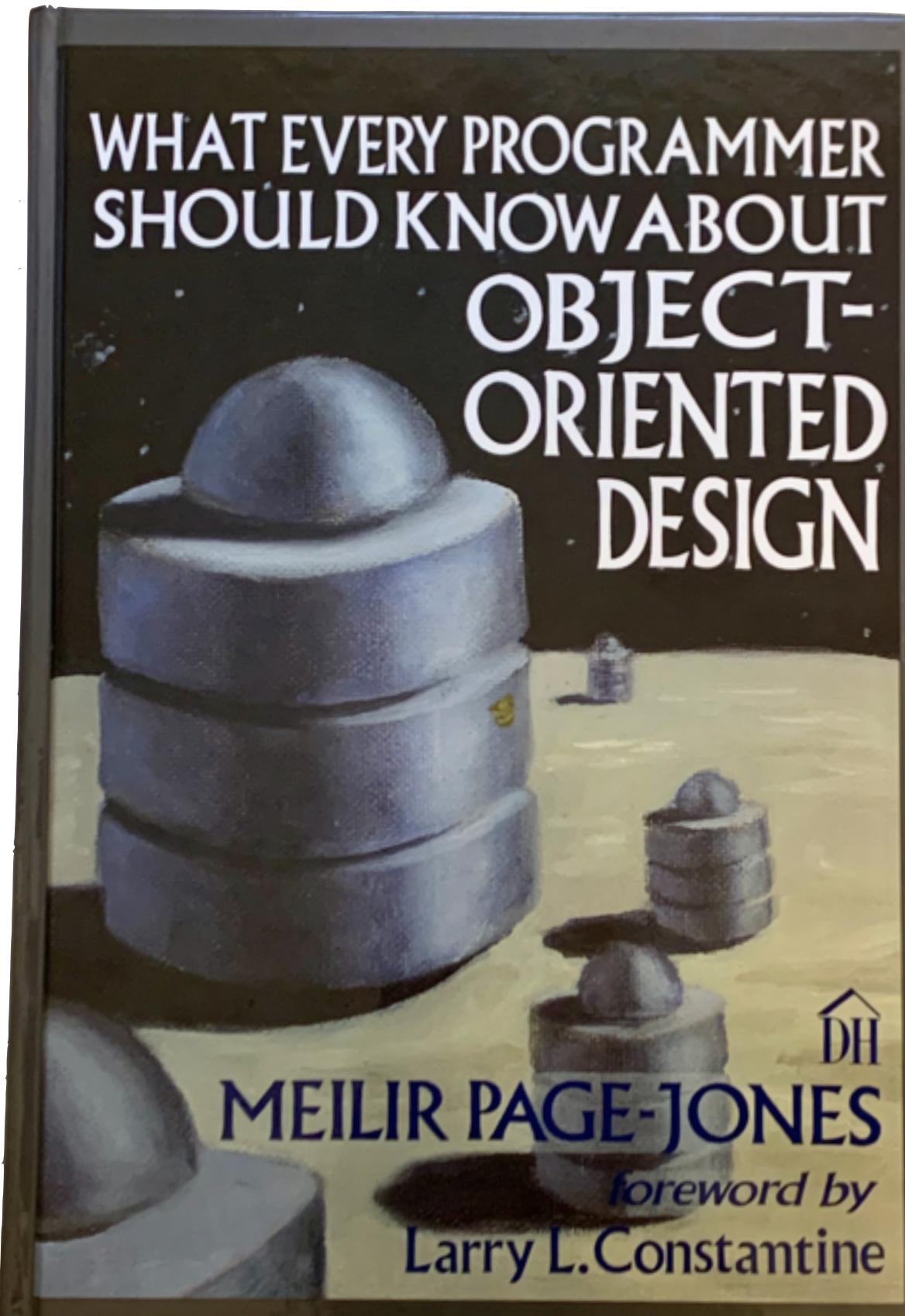
*dynamic connescence*

# Connascence of Identity

(CoI)

multiple components must reference  
something

# Contranscience



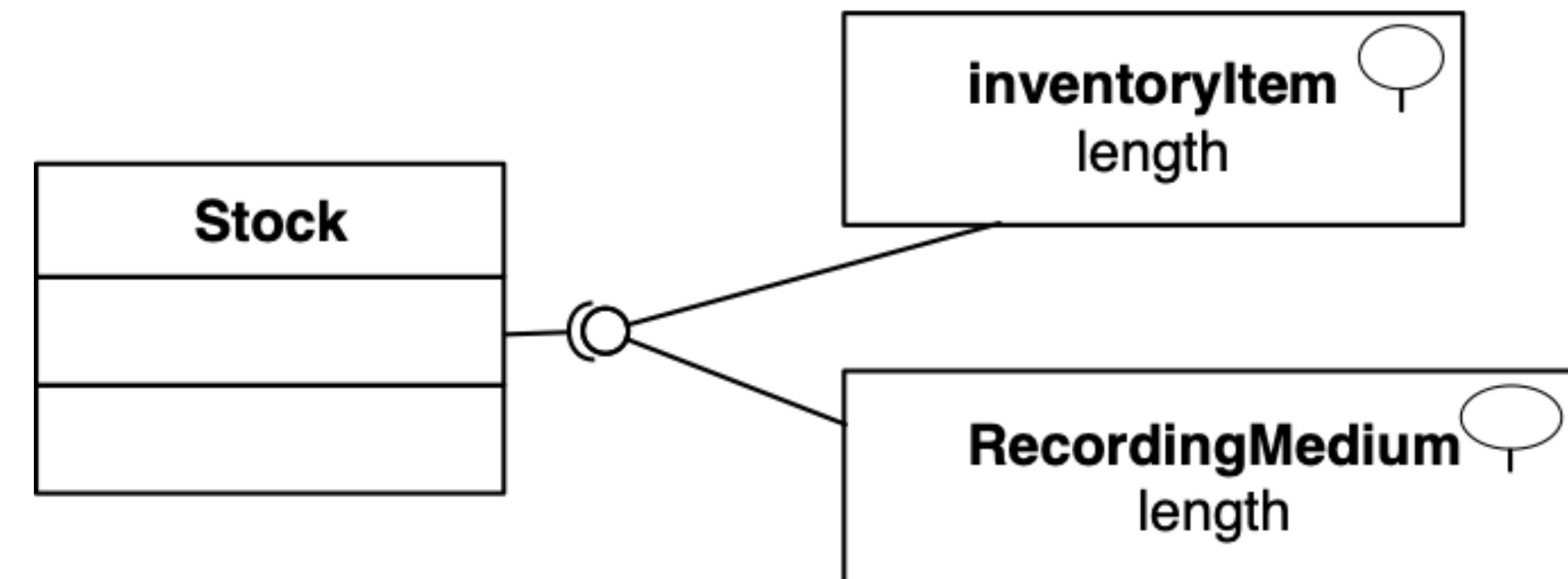
connascence of difference  
negative connascence

*difference rather than equality must  
be preserved.*

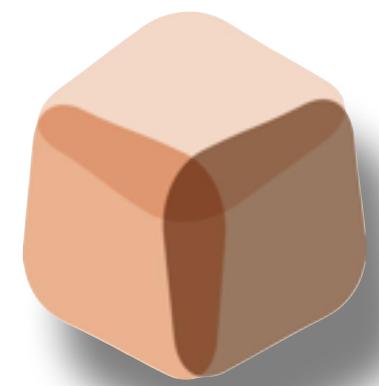
```
int i;  
int j;
```

# Contranscience

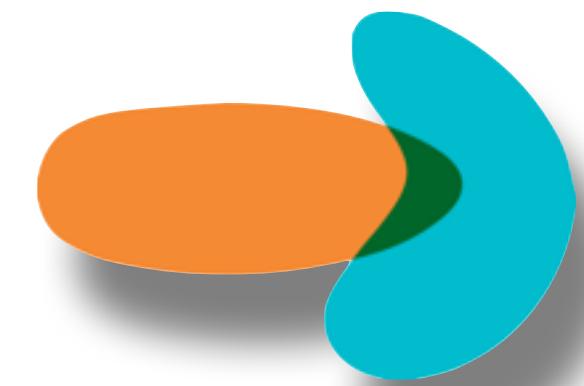
*difference rather than equality must  
be preserved.*



# Connascence Properties



static



dynamic

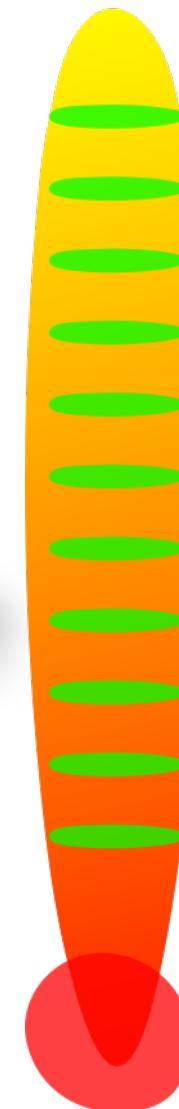
Strength



Locality

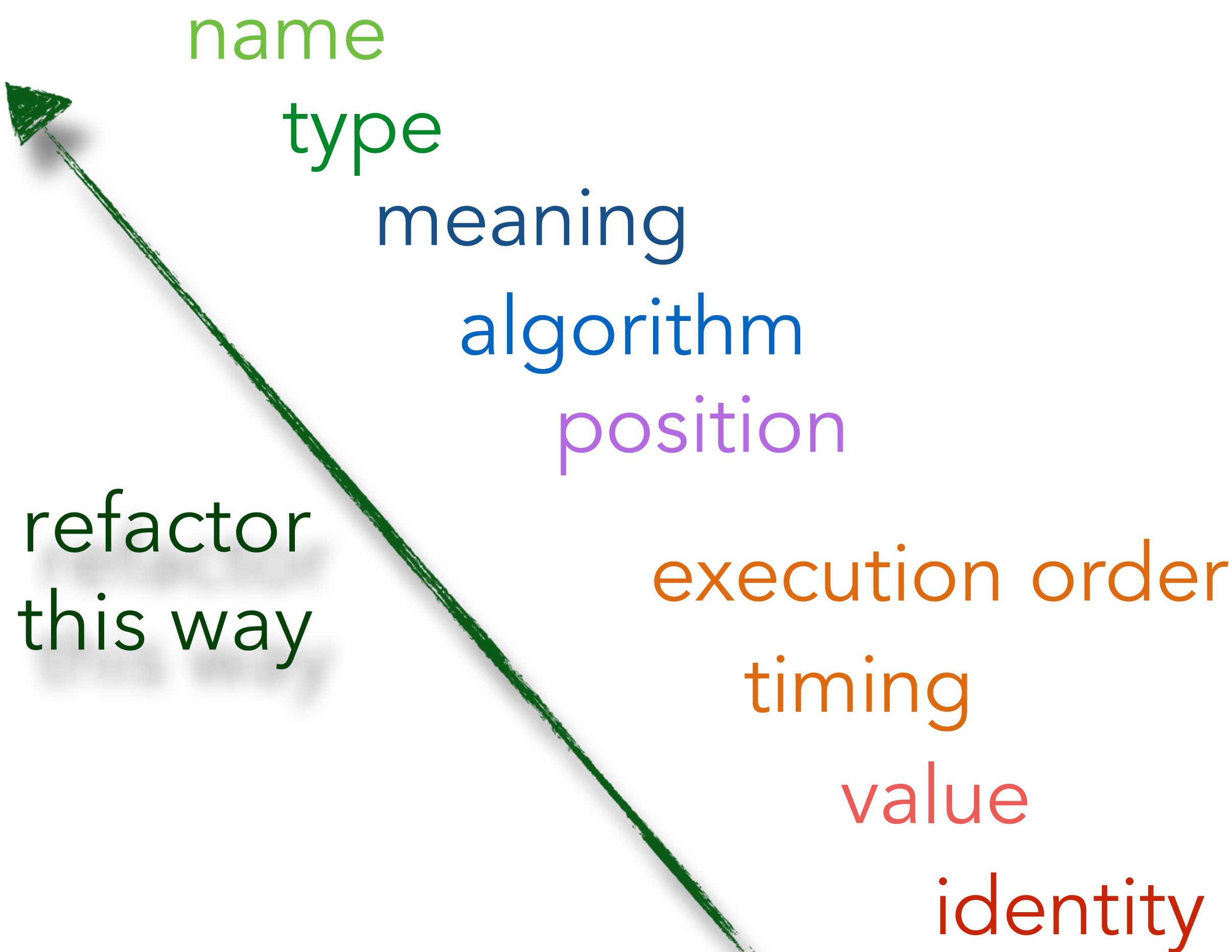


Degree

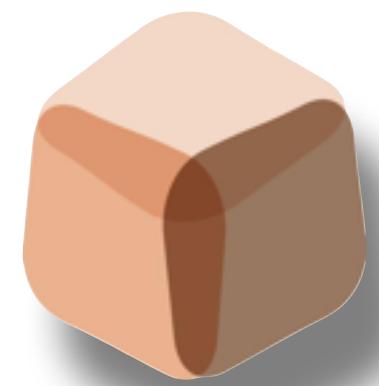


# Connascence Properties

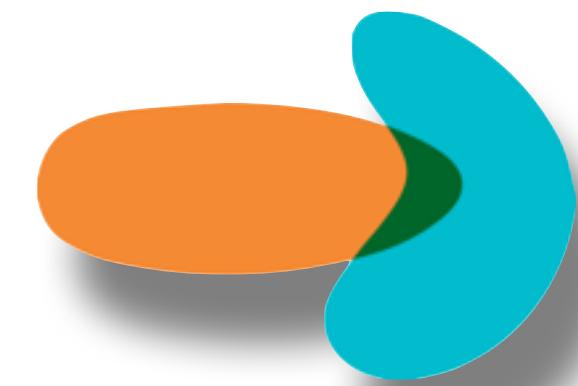
Strength



# Connascence Properties



static



dynamic

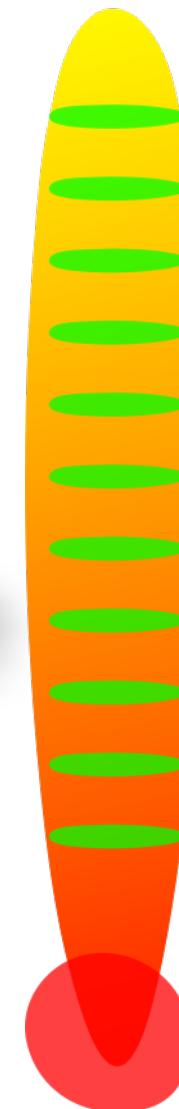
Strength



Locality

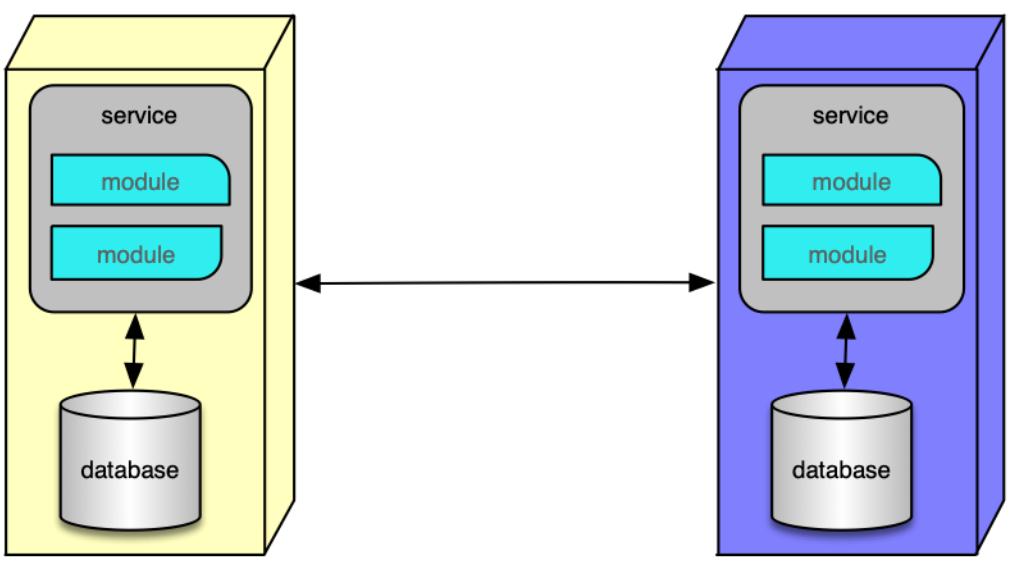
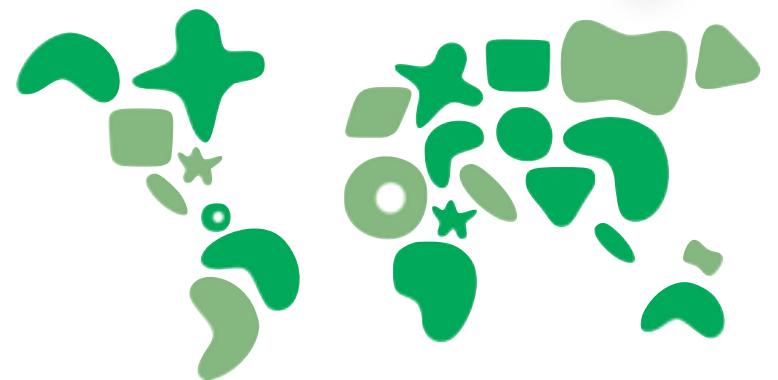


Degree



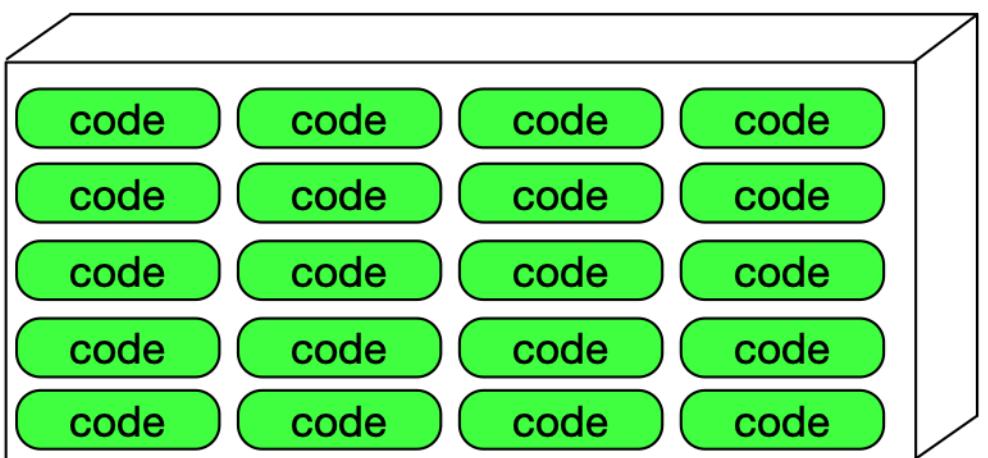
# Connascence Properties

## Locality

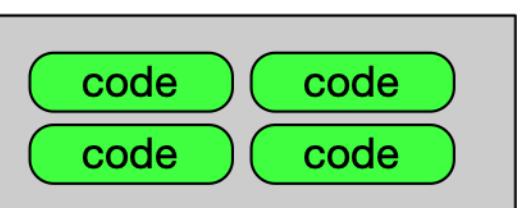


static

Proximal code (in the same module, class, or function) should have more & higher forms of connascence than less proximal code (in separate modules, or even codebases).

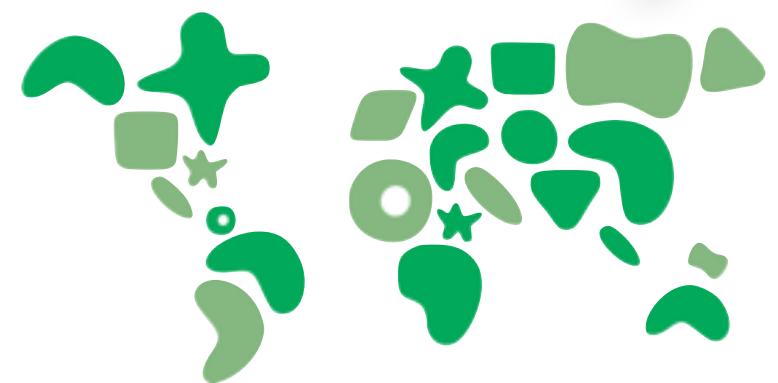


dynamic

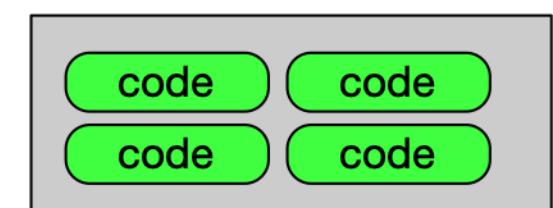
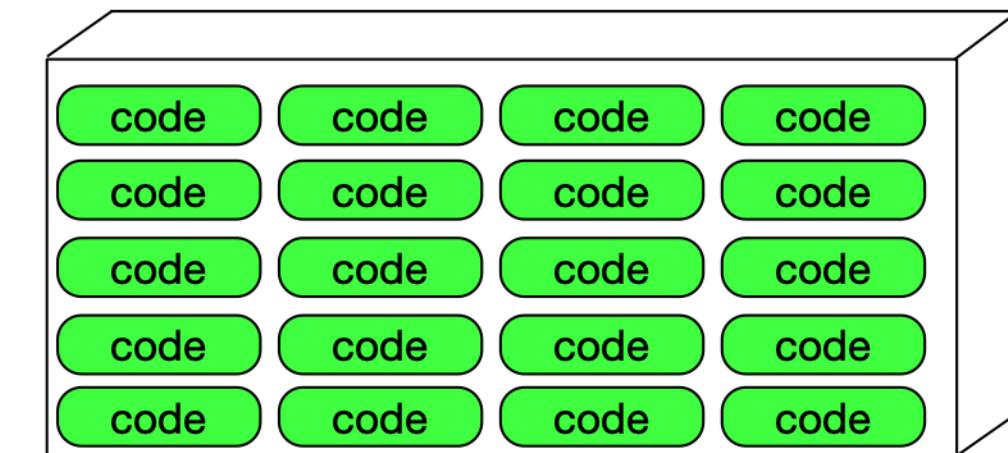
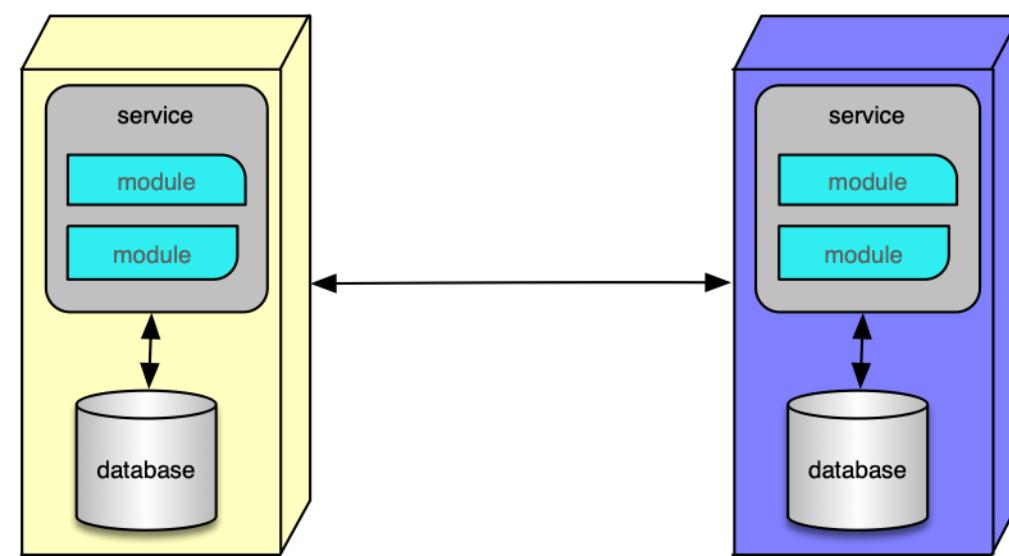


# Connascence Properties

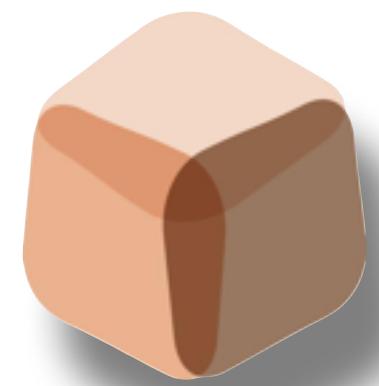
Locality



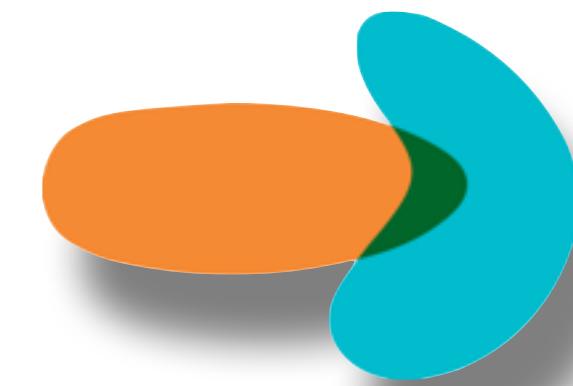
locality matters!



# Connascence Properties



static



dynamic

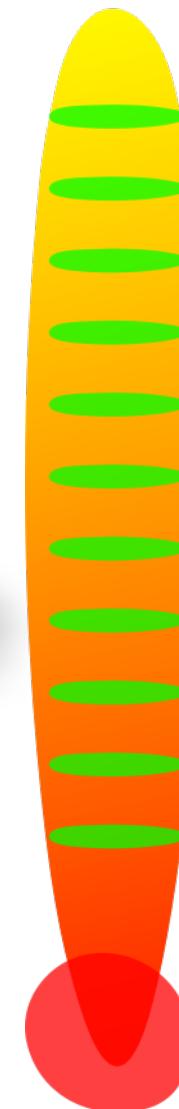
Strength



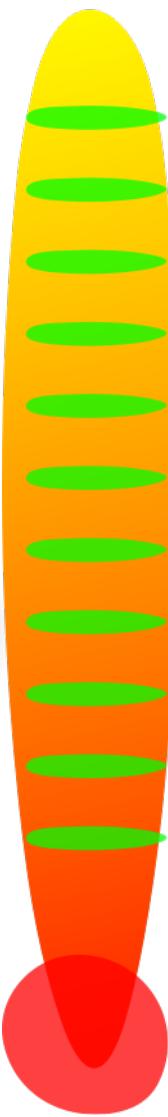
Locality



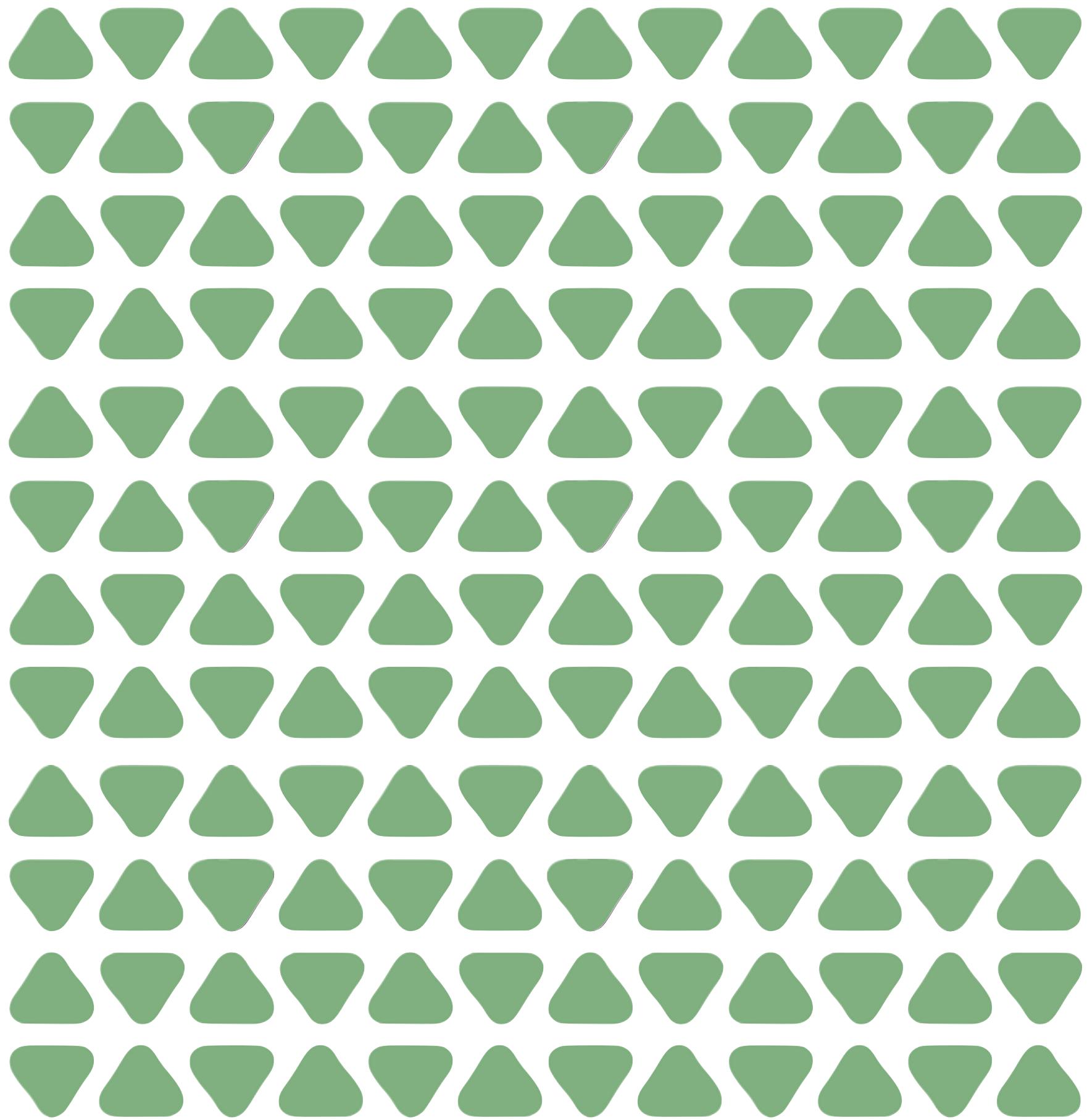
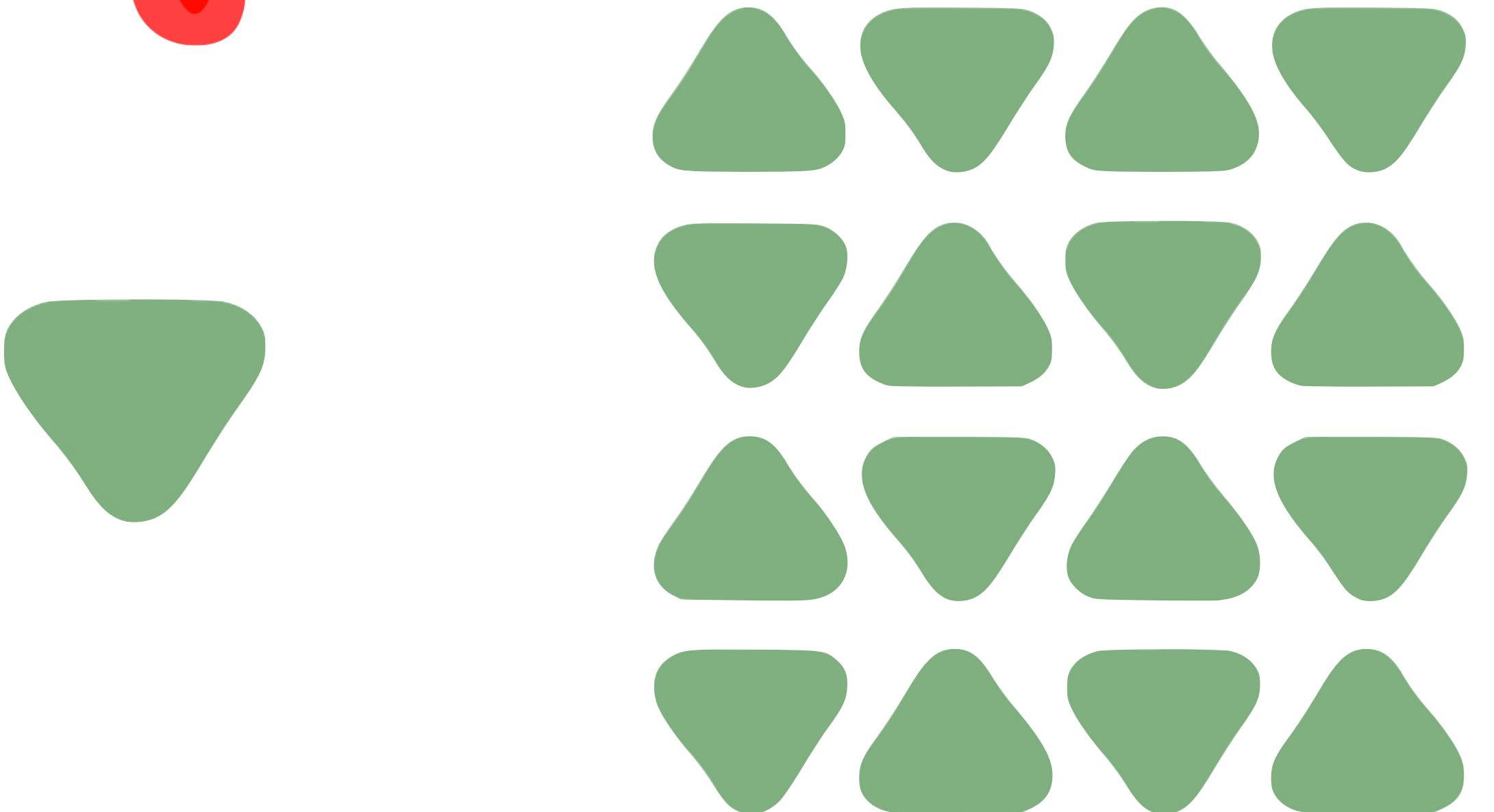
Degree



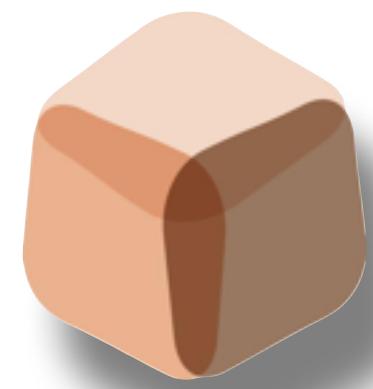
Degree



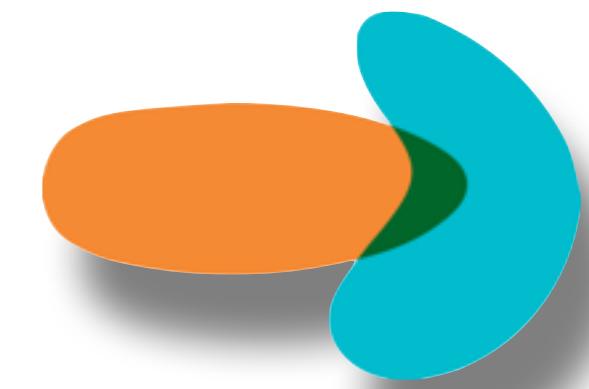
# Connascence Properties



# Connascence Properties



static



dynamic

Strength



Locality



Degree



component cohesion

# component cohesion

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

# component cohesion

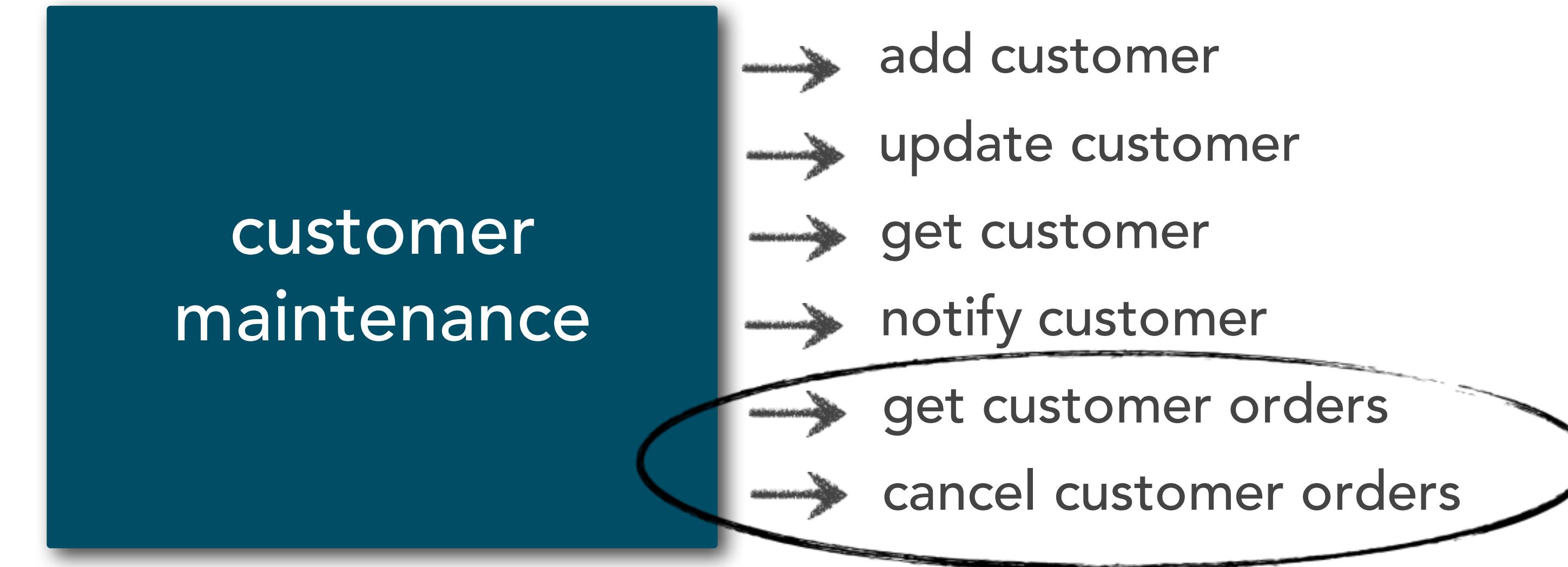
the degree and manner to which the operations of a component are related to one another

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



# 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



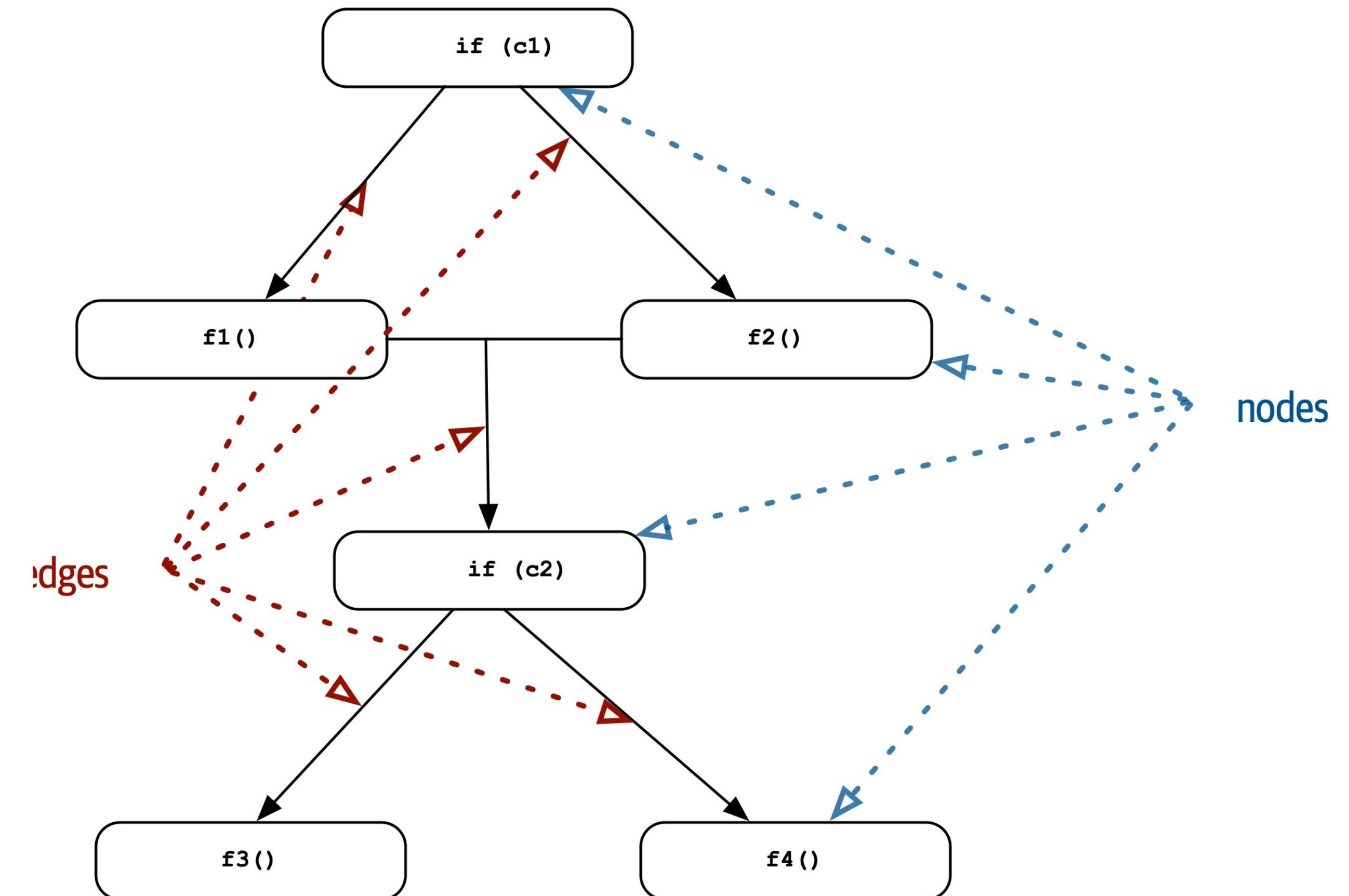
metrics

# metrics

## cyclomatic complexity

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

```
public void decision(int c1, int c2) {  
    if (c1 < 100)  
        return 0;  
    else if (c1 + c2 > 500)  
        return 1;  
    else  
        return -1;  
}
```



# metrics

## Chidamber & Kemerer Metrics

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

# metrics

## Chidamber & Kemerer Metrics

✓ LCOM (Lack of Cohesion in Methods)

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

# metrics

## Chidamber & Kemerer Metrics

### ✓ LCOM (Lack of Cohesion in Methods)

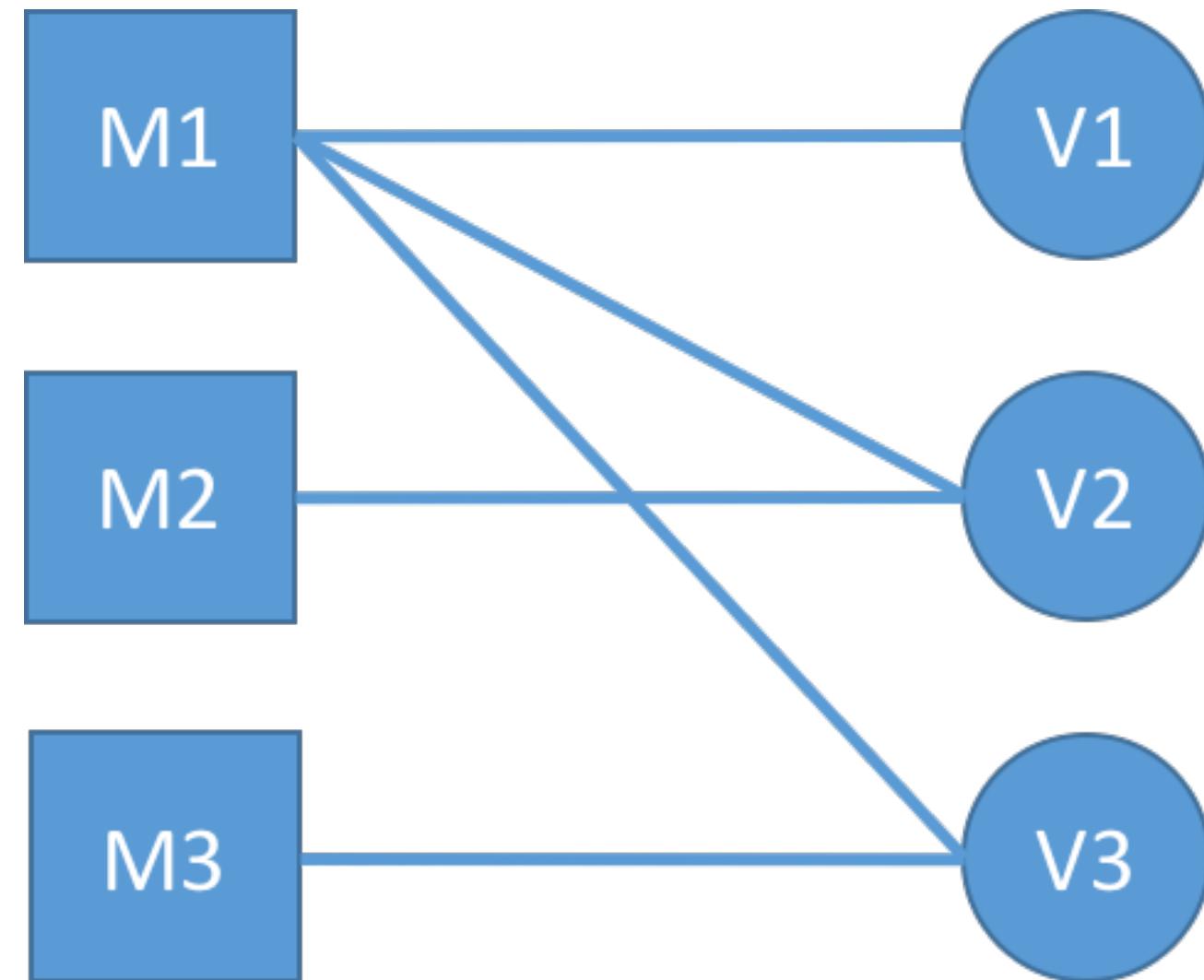
The LCOM is a count of the number of method pairs whose similarity is 0 (i.e.  $\sigma()$  is a null set) minus the count of method pairs whose similarity is not zero.

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

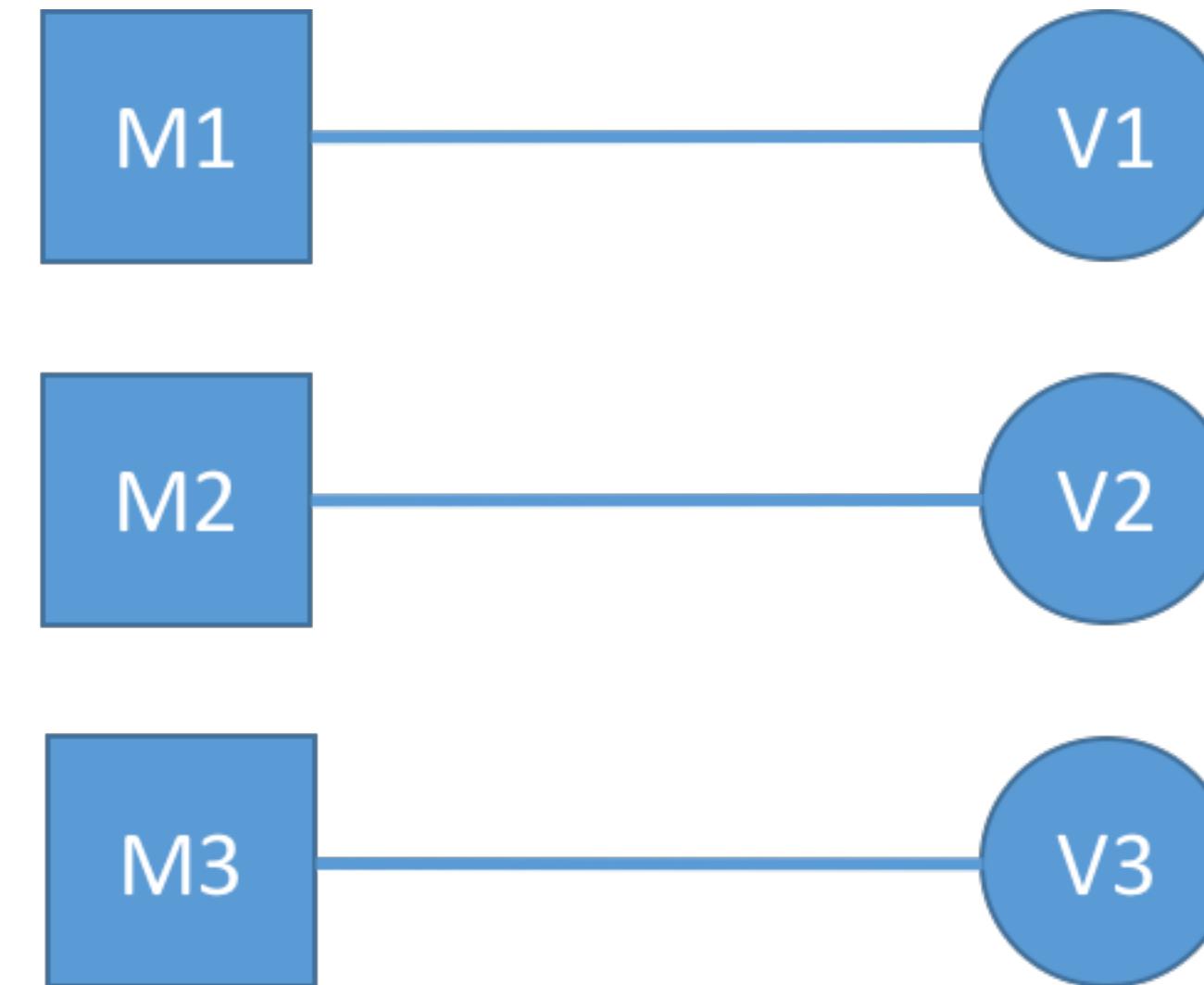
# metrics

## Chidamber & Kemerer Metrics

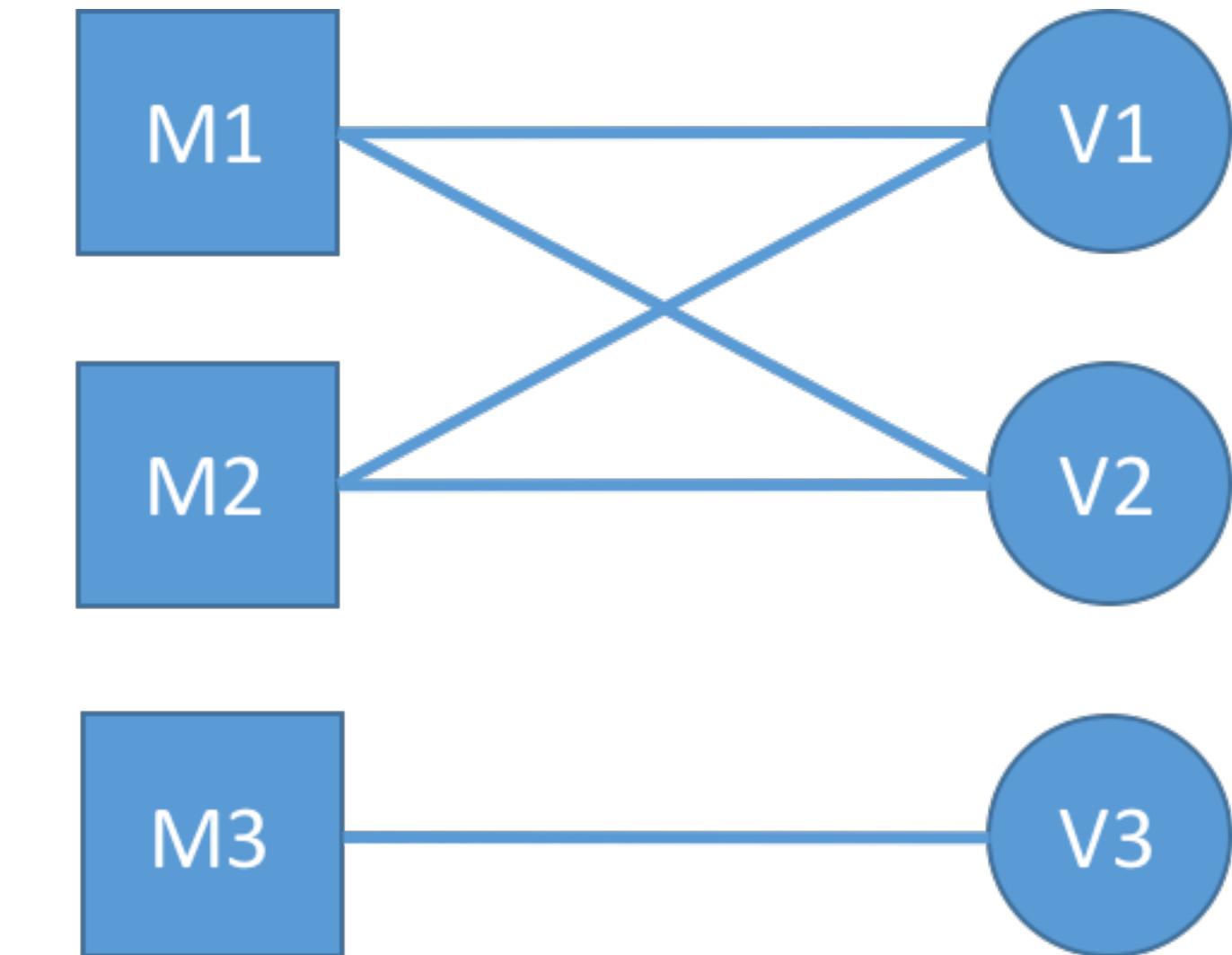
✓ LCOM (Lack of Cohesion in Methods)



(A)



(B)

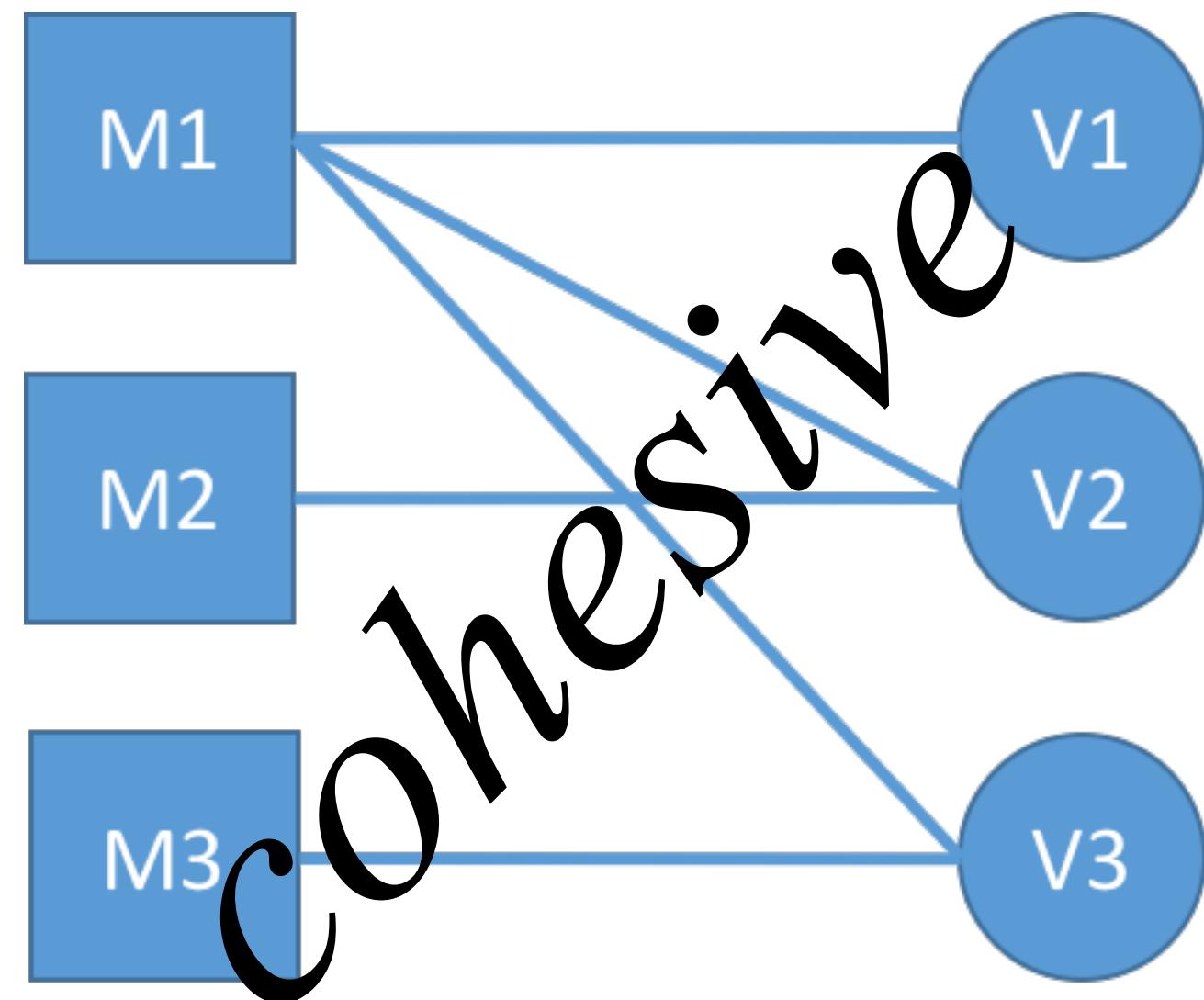


(C)

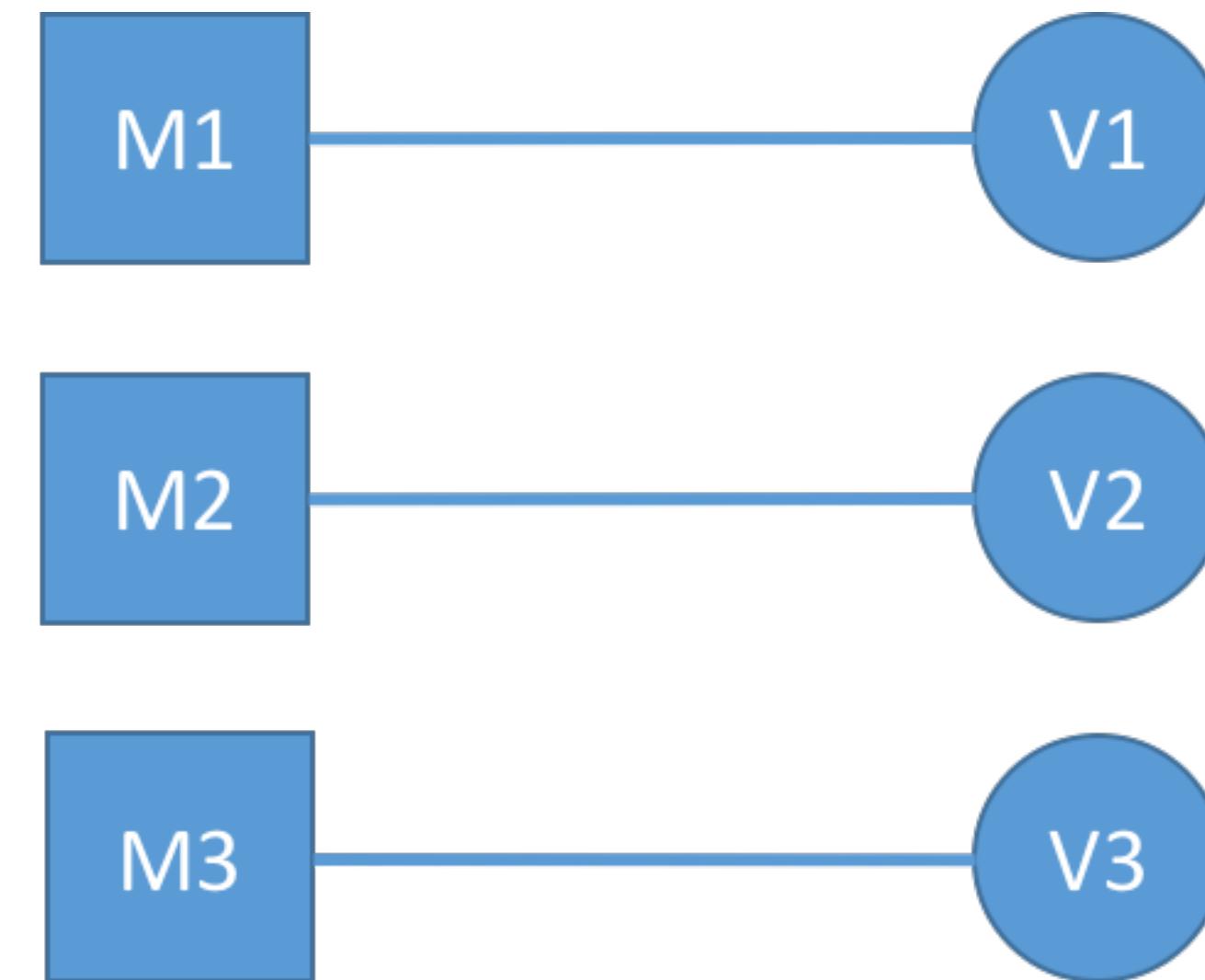
# metrics

## Chidamber & Kemerer Metrics

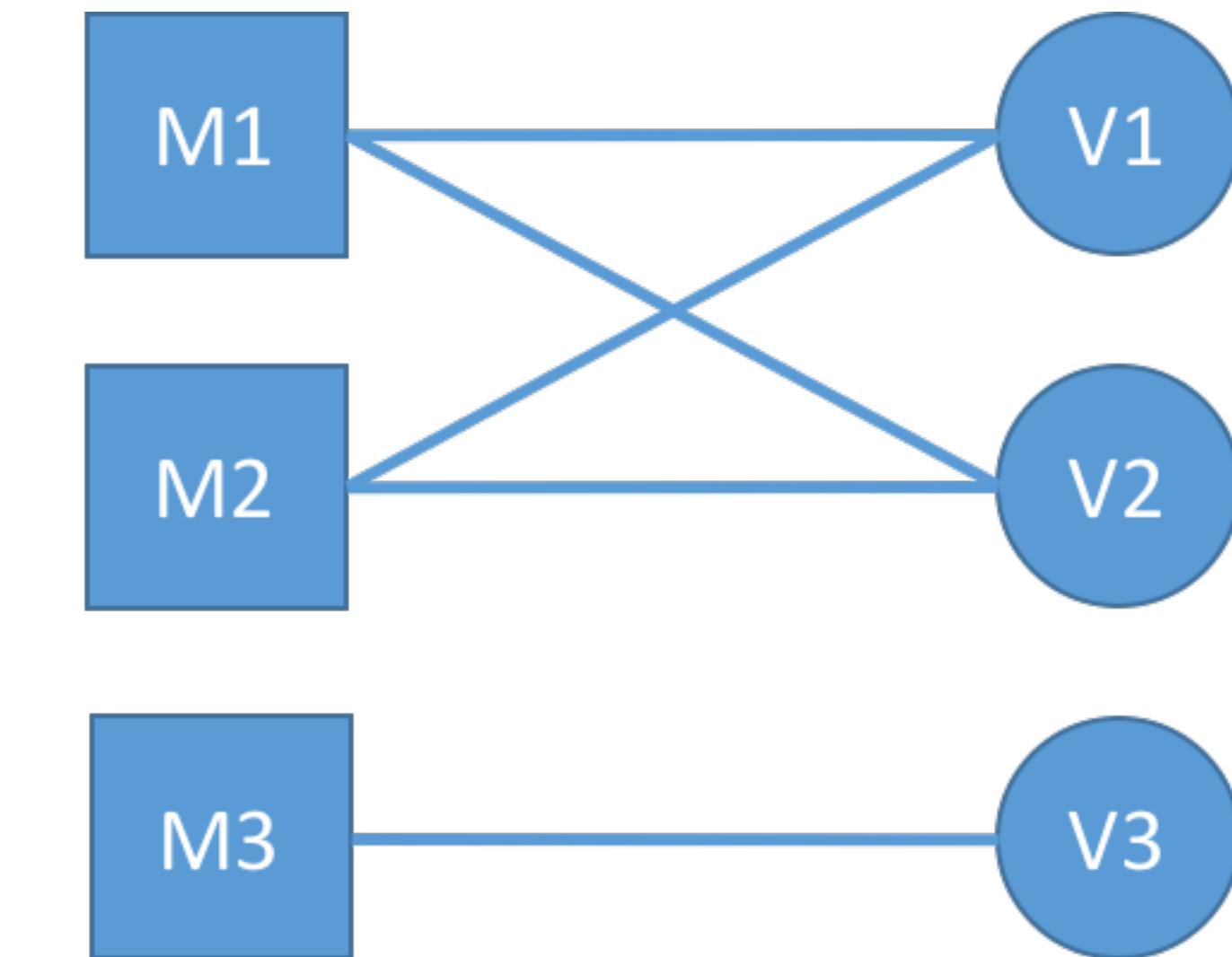
✓ LCOM (Lack of Cohesion in Methods)



(A)



(B)

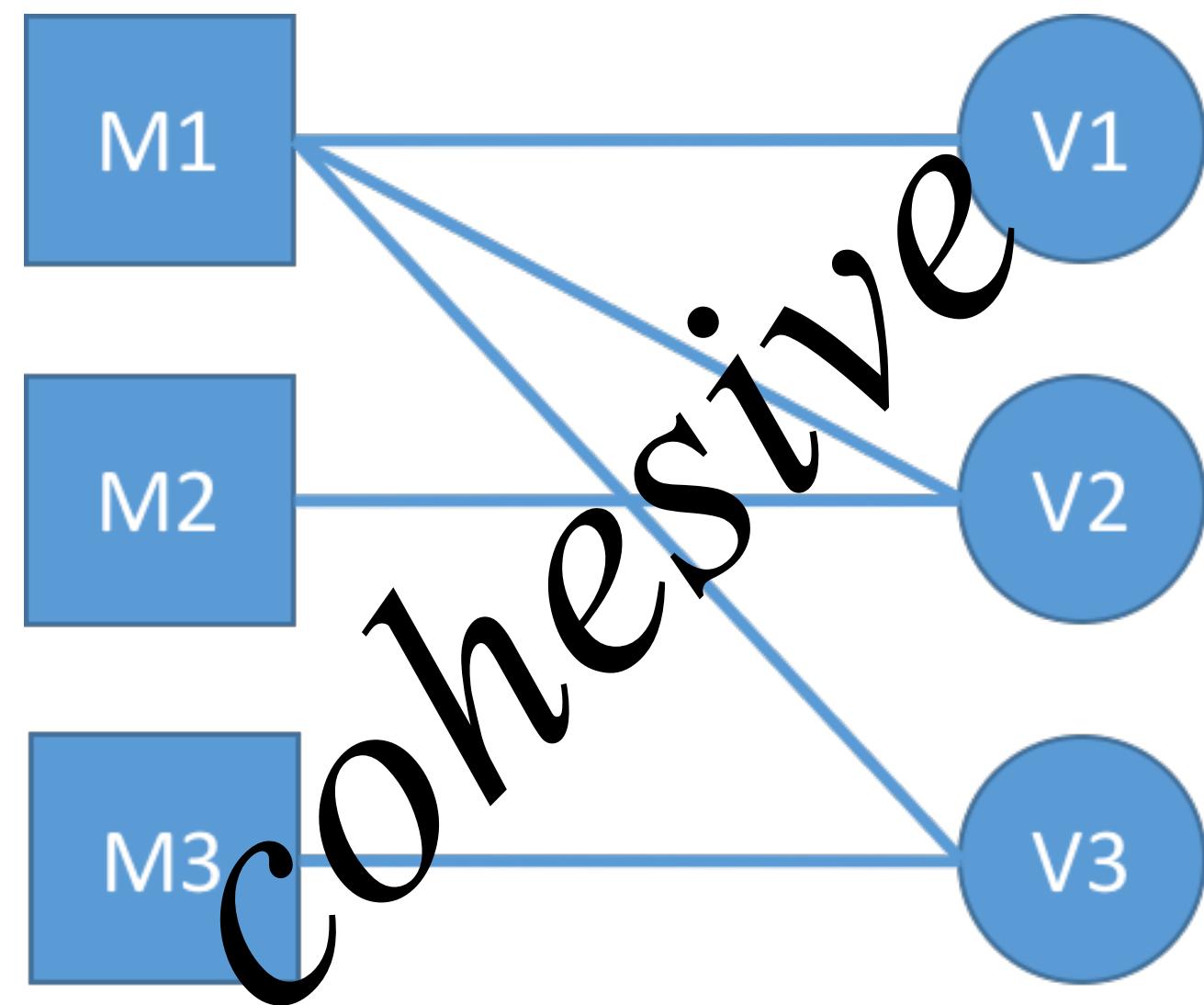


(C)

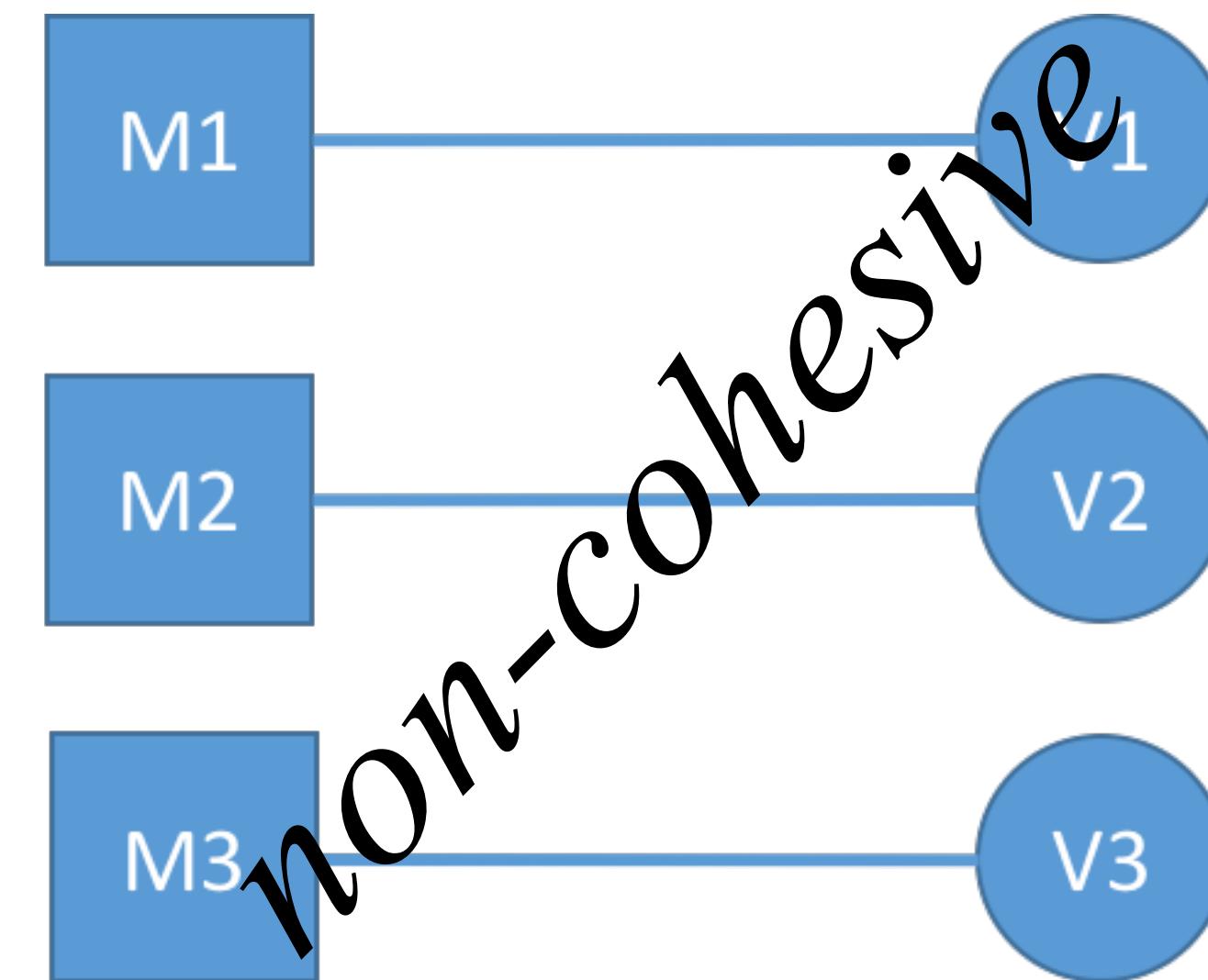
# metrics

## Chidamber & Kemerer Metrics

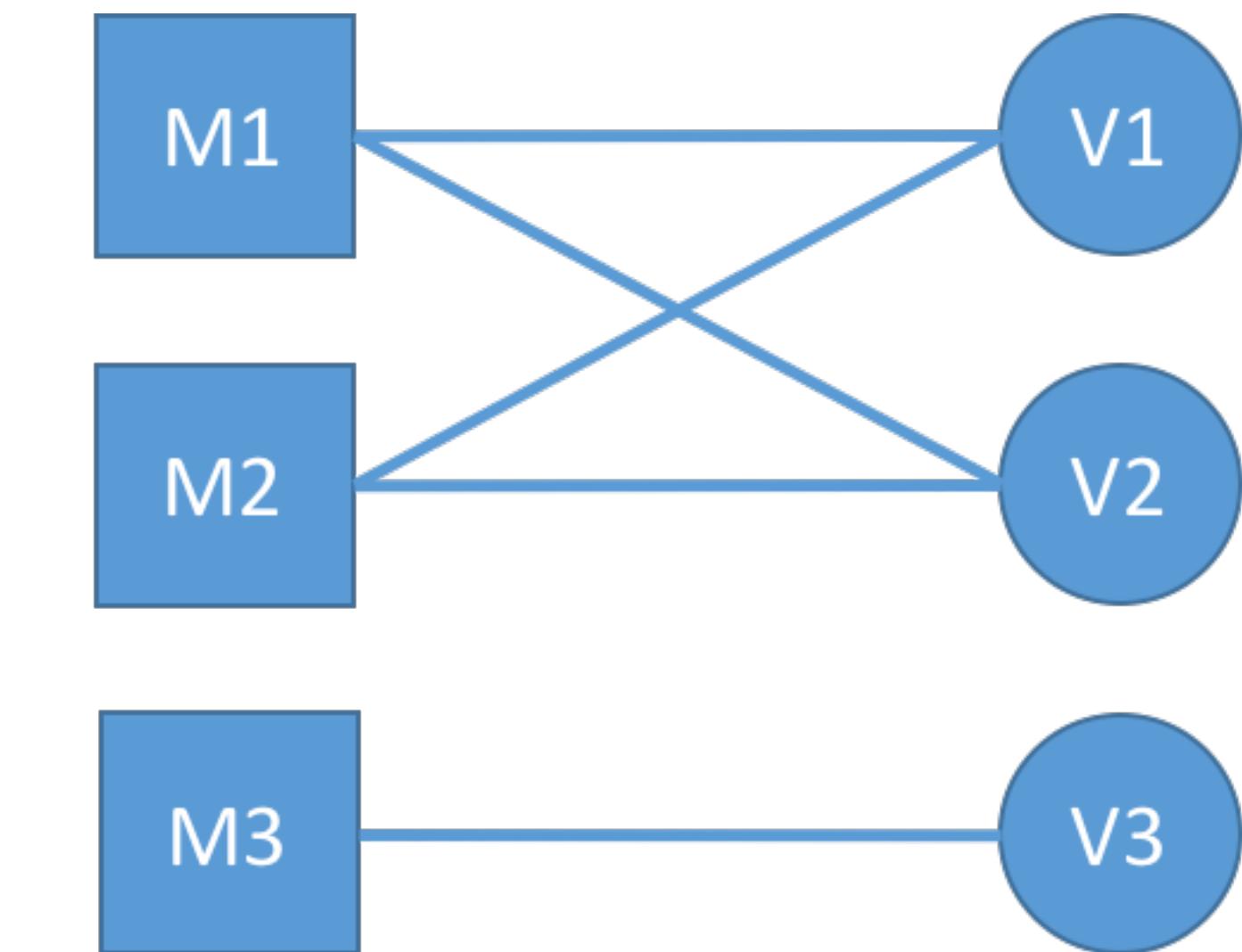
✓ LCOM (Lack of Cohesion in Methods)



(A)



(B)

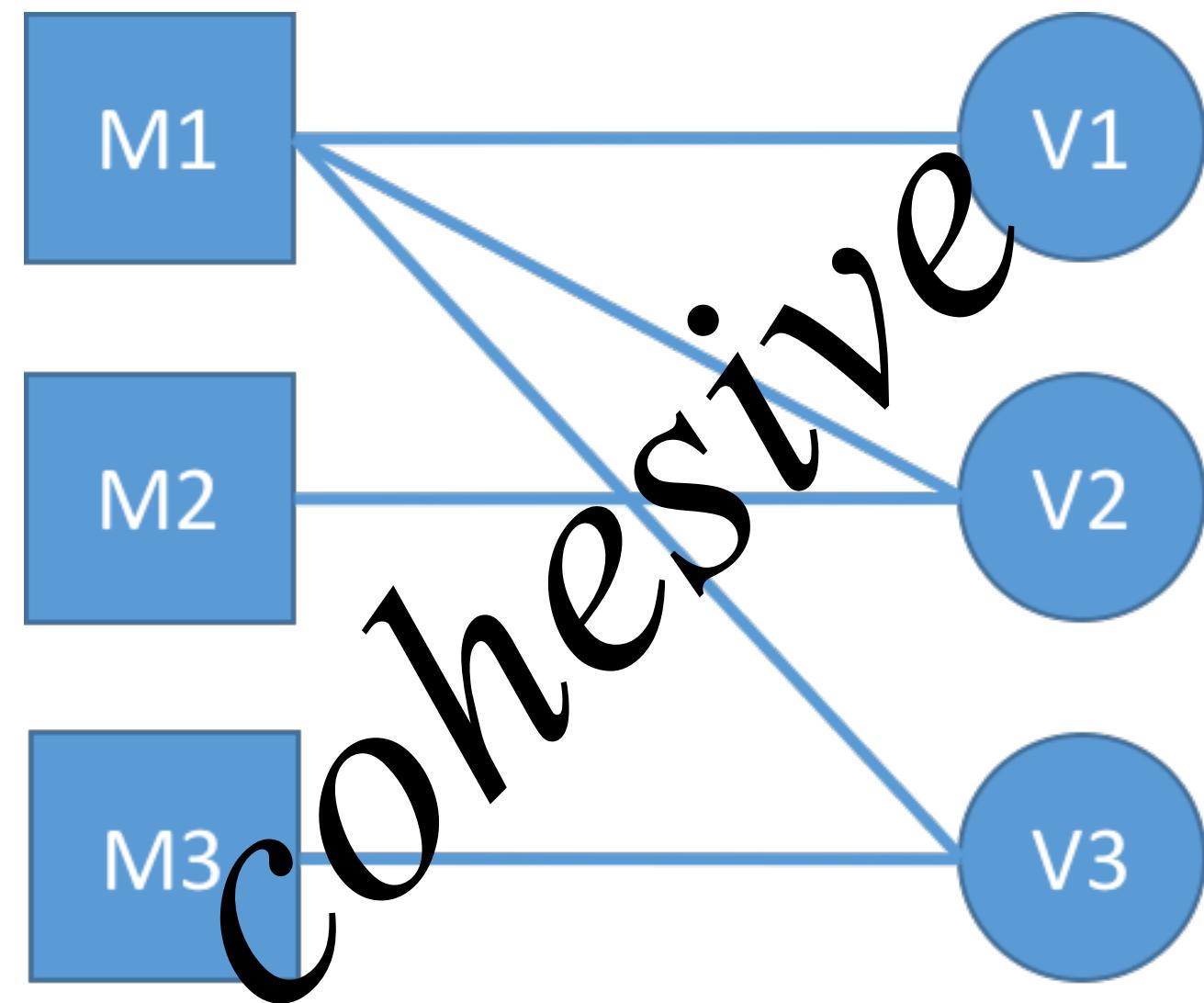


(C)

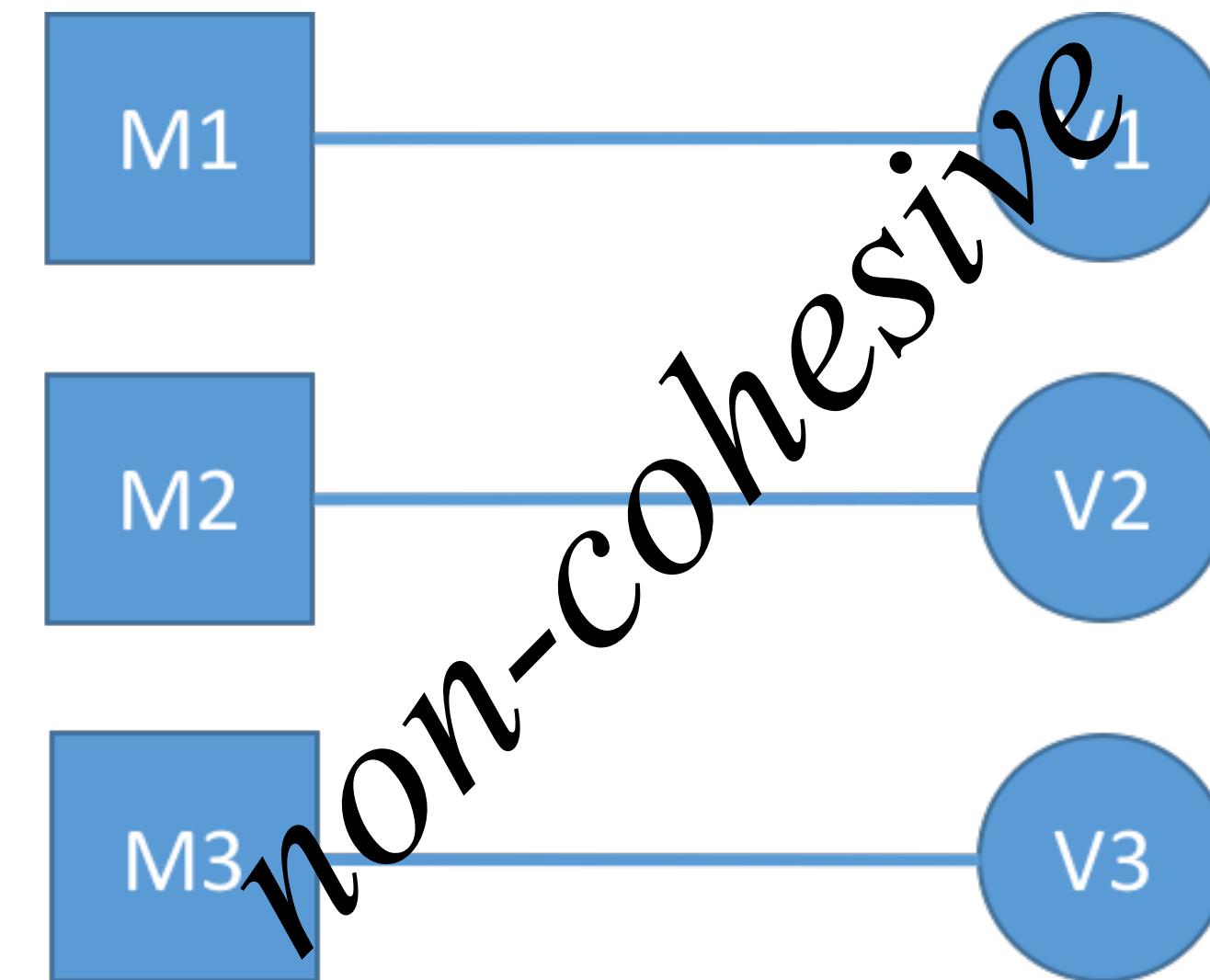
# metrics

## Chidamber & Kemerer Metrics

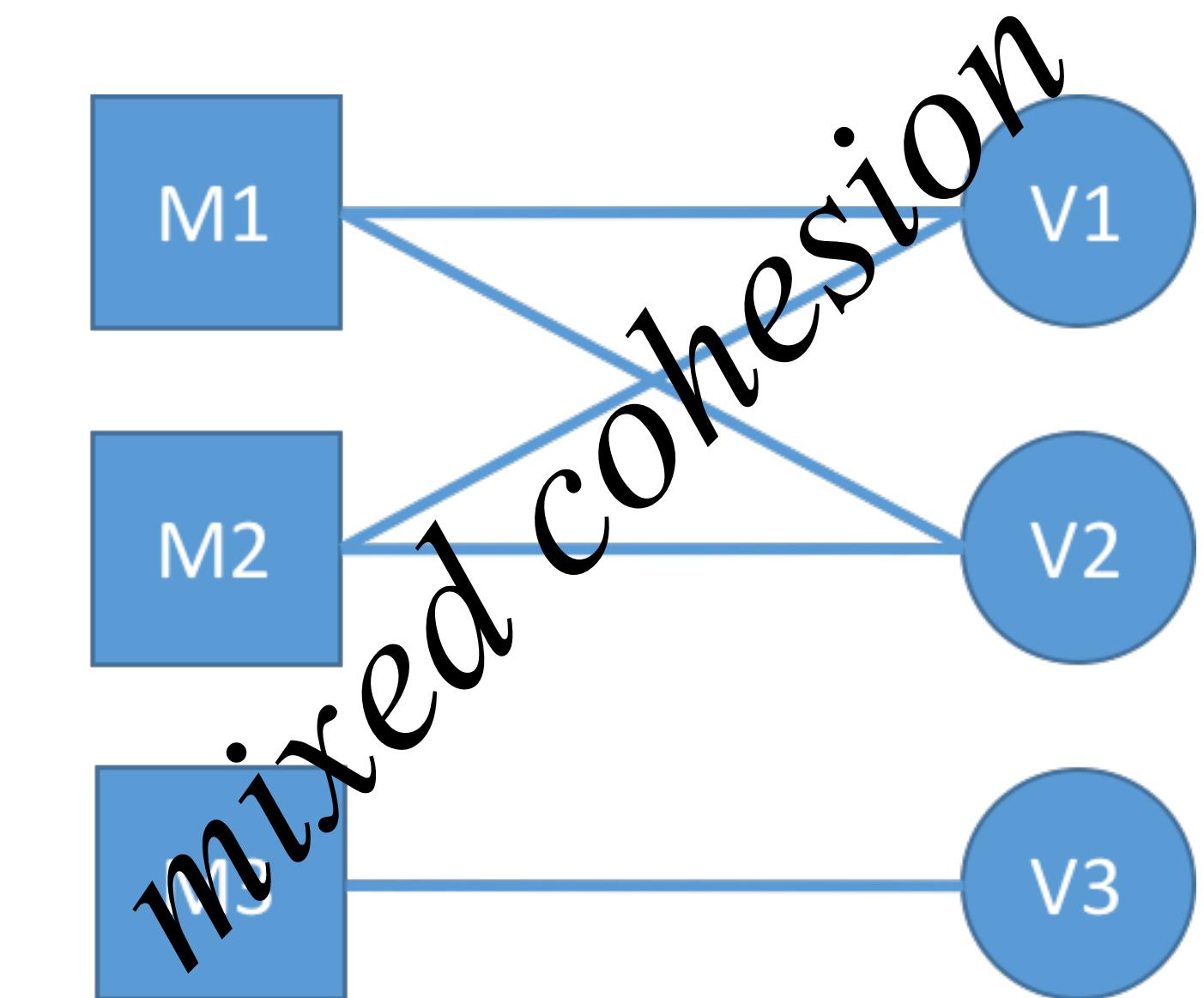
✓ LCOM (Lack of Cohesion in Methods)



(A)



(B)



(C)

# derived coupling metrics

abstractness

$$A = \frac{\Sigma ABSTRACT}{\Sigma CONCRETE}$$

# derived coupling metrics

instability

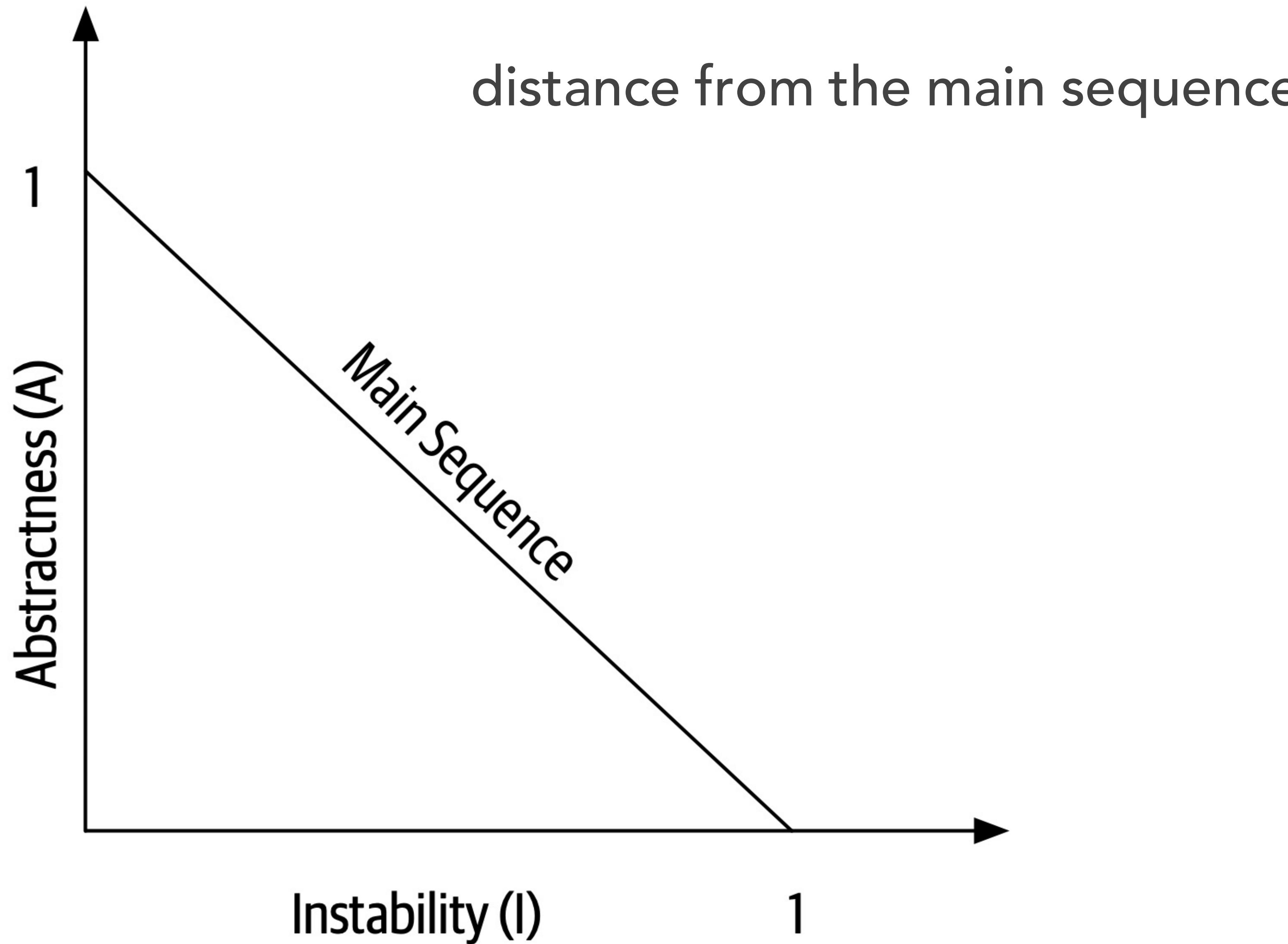
$$I = \frac{C^e}{C^e + C^a}$$

# derived coupling metrics

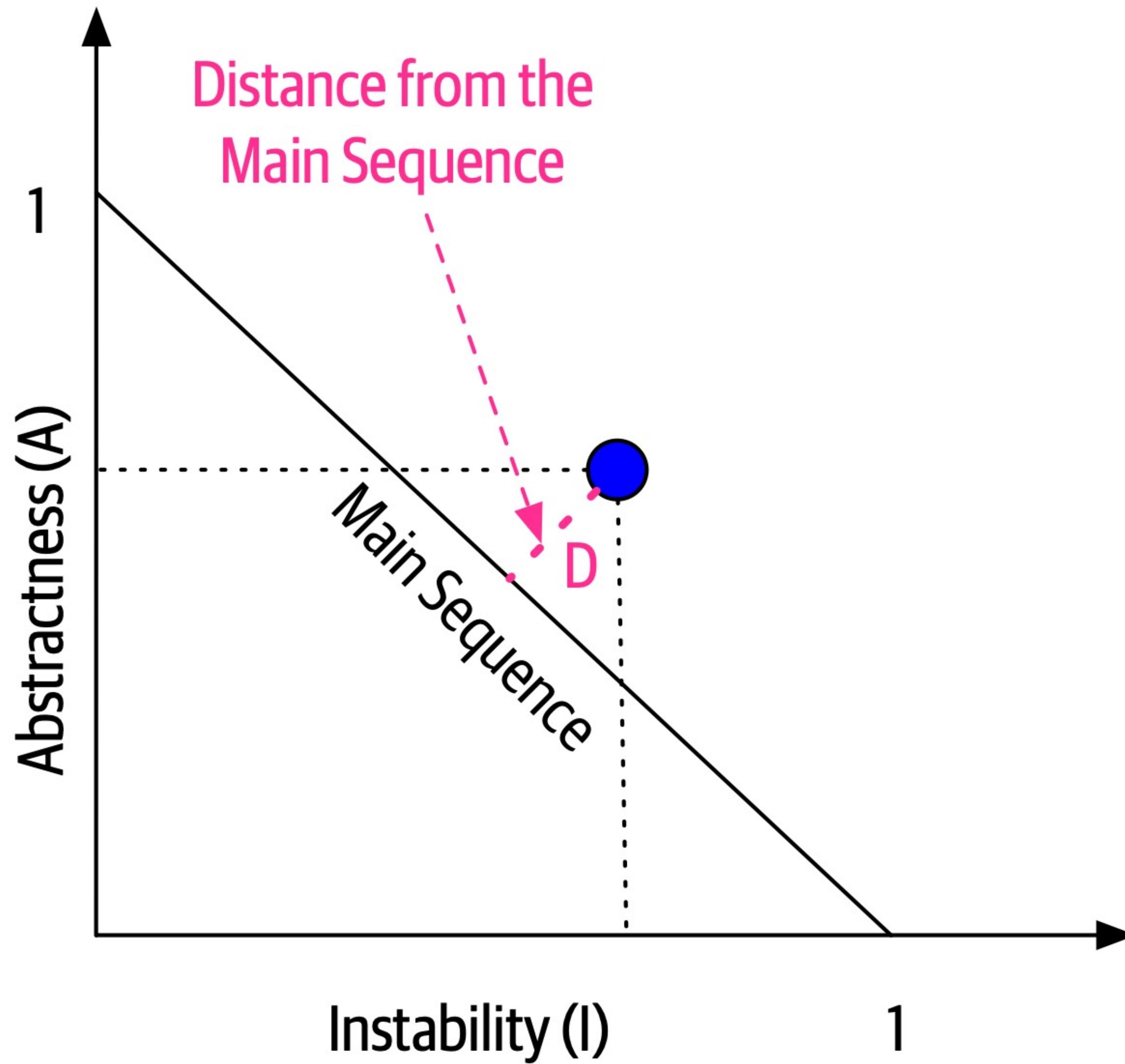
distance from the main sequence

$$D = |A + I - 1|$$

# derived coupling metrics



# derived coupling metrics



# metrics $\cap$ architecture characteristics

## Chidamber & Kemerer Metrics

✓ DIT (depth of inheritance tree)

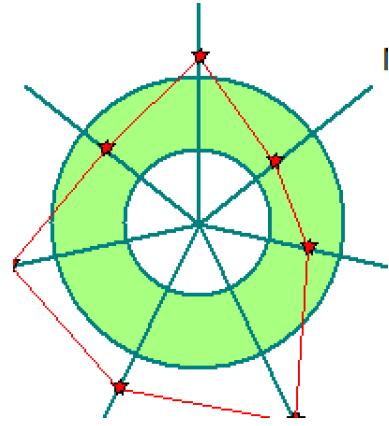
✓ WMC (weighted methods/class; sum of CC)

✓ CE (efferent coupling count)

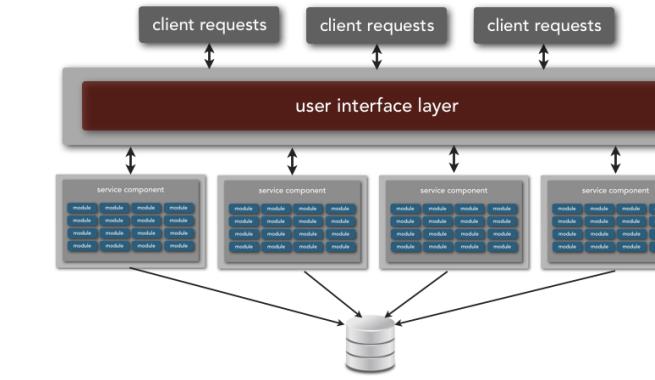
✓ CA (afferent coupling count)

# metrics $\cap$ architecture characteristics

## architecture characteristics mapping



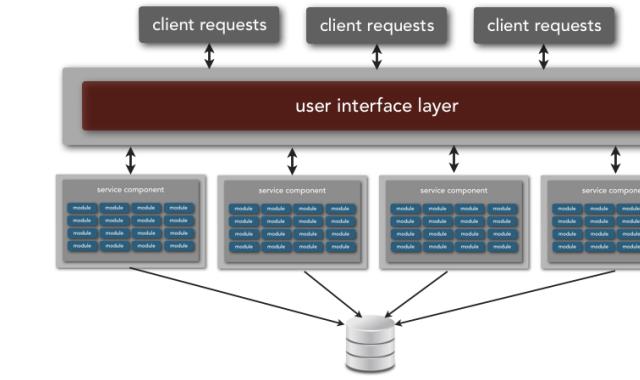
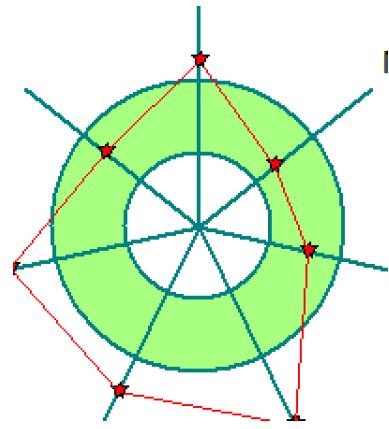
component size  
complexity / WMC  
coupling (CE, CA, CT)  
inheritance depth (DIT)  
percent comments



modularity  
maintainability  
testability  
availability  
deployment  
reliability  
scalability  
evolutionary / migration

# metrics $\cap$ architecture characteristics

## architecture characteristics mapping



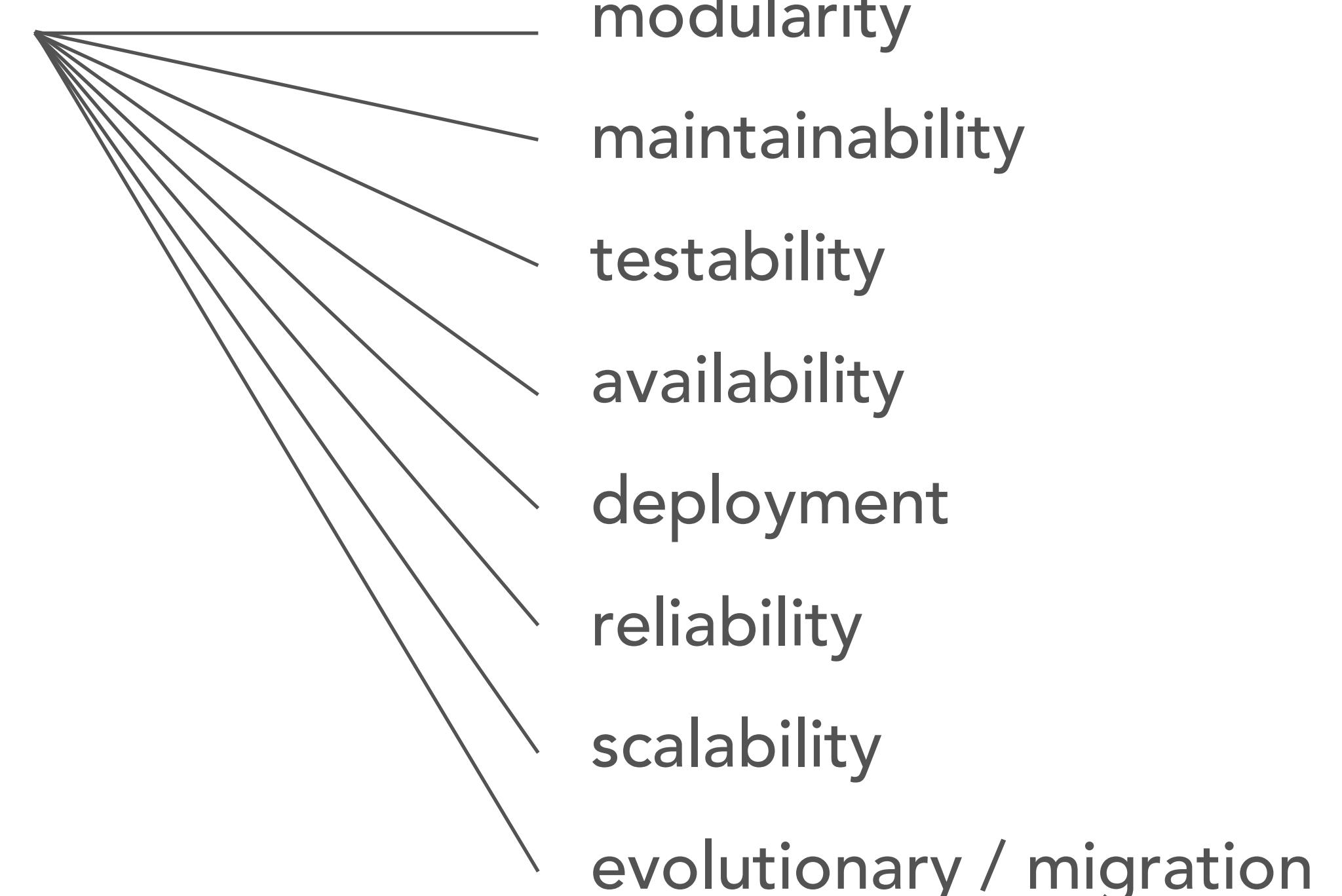
component size

complexity / WMC

coupling (CE, CA, CT)

inheritance depth (DIT)

percent comments



modularity

maintainability

testability

availability

deployment

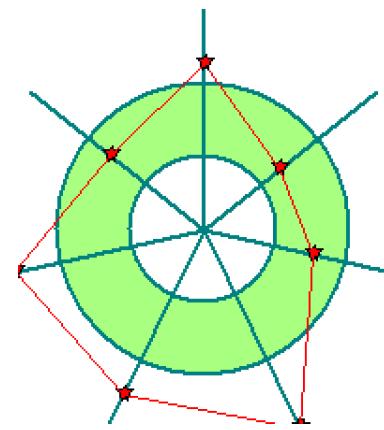
reliability

scalability

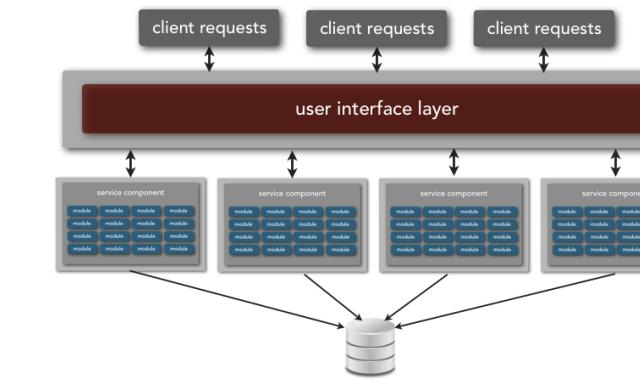
evolutionary / migration

# metrics $\cap$ architecture characteristics

## architecture characteristics mapping



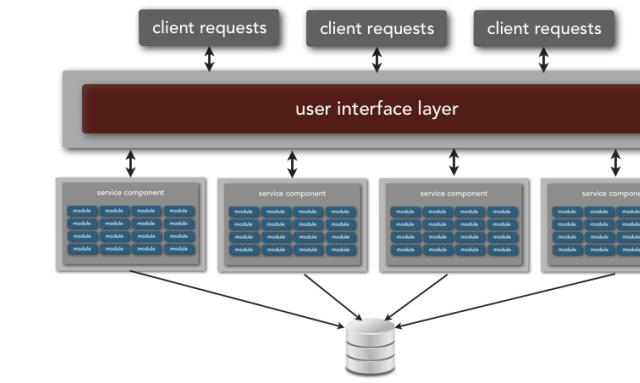
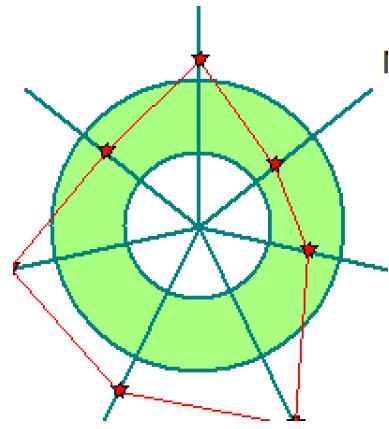
component size  
complexity / WMC  
coupling (CE, CA, CT)  
inheritance depth (DIT)  
percent comments



modularity  
maintainability  
testability  
availability  
deployment  
reliability  
scalability  
evolutionary / migration

# metrics $\cap$ architecture characteristics

## architecture characteristics mapping

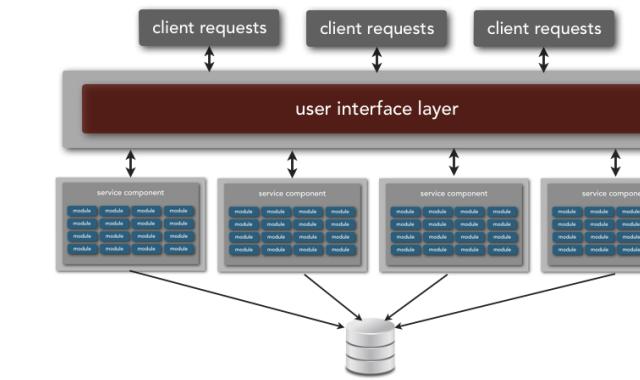
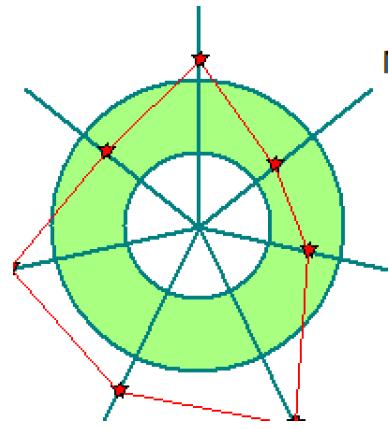


component size  
complexity / WMC  
coupling (CE, CA, CT)  
inheritance depth (DIT)  
percent comments

modularity  
maintainability  
testability  
availability  
deployment  
reliability  
scalability  
evolutionary / migration

# metrics $\cap$ architecture characteristics

## architecture characteristics mapping

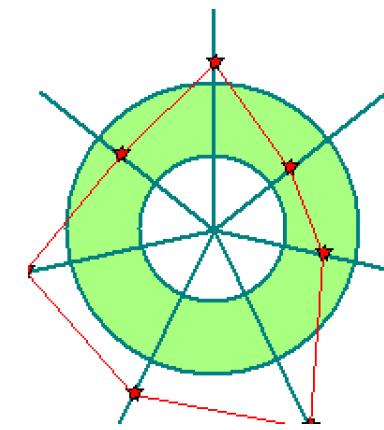


component size  
complexity / WMC  
coupling (CE, CA, CT)  
inheritance depth (DIT)  
percent comments

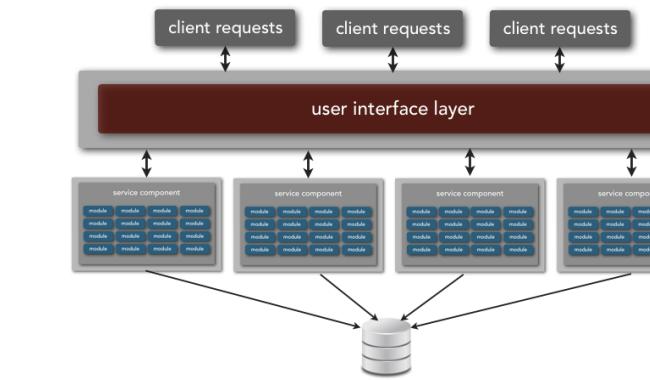
modularity  
maintainability  
testability  
availability  
deployment  
reliability  
scalability  
evolutionary / migration

# metrics $\cap$ 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

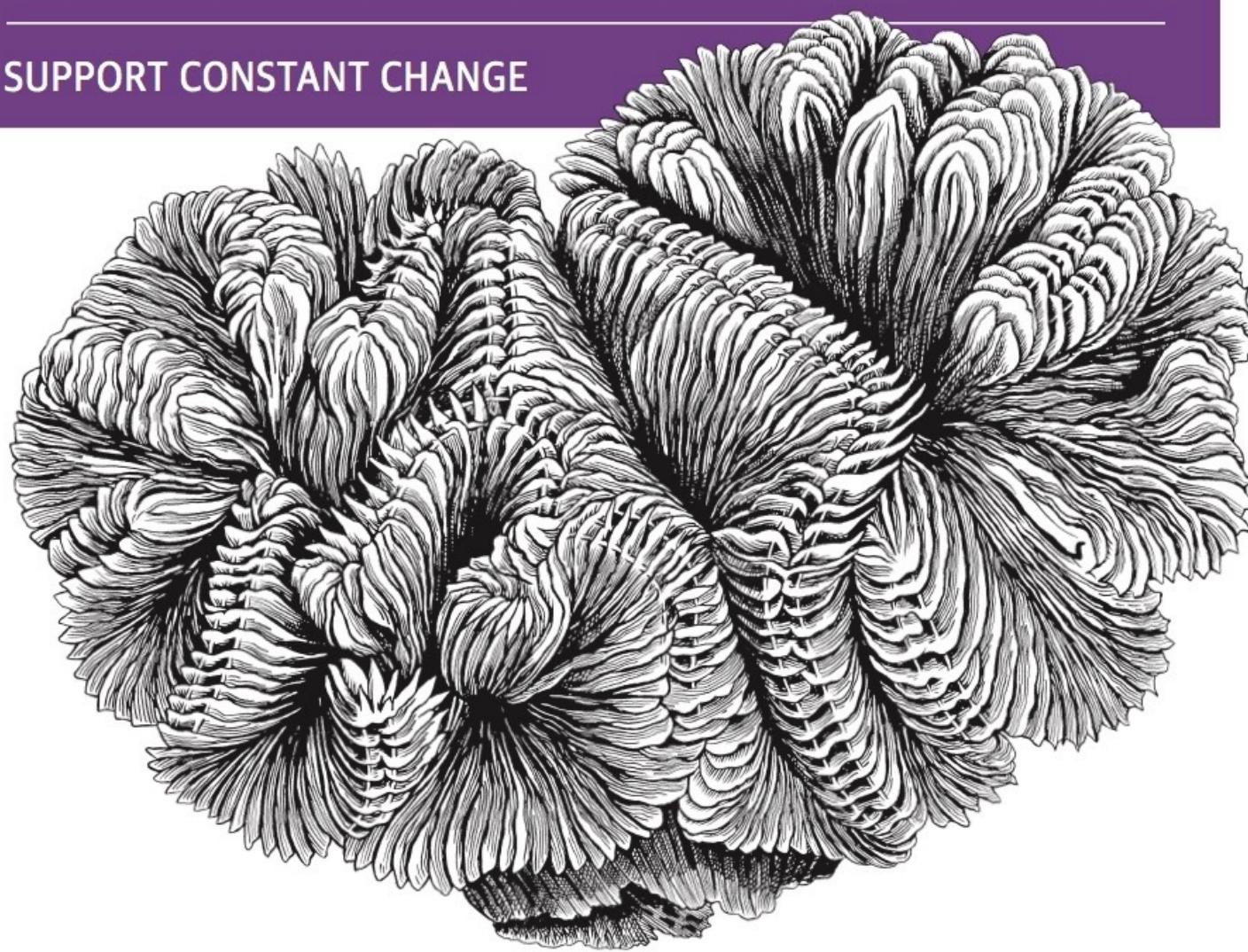
# Question:

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

O'REILLY®

# Building Evolutionary Architectures

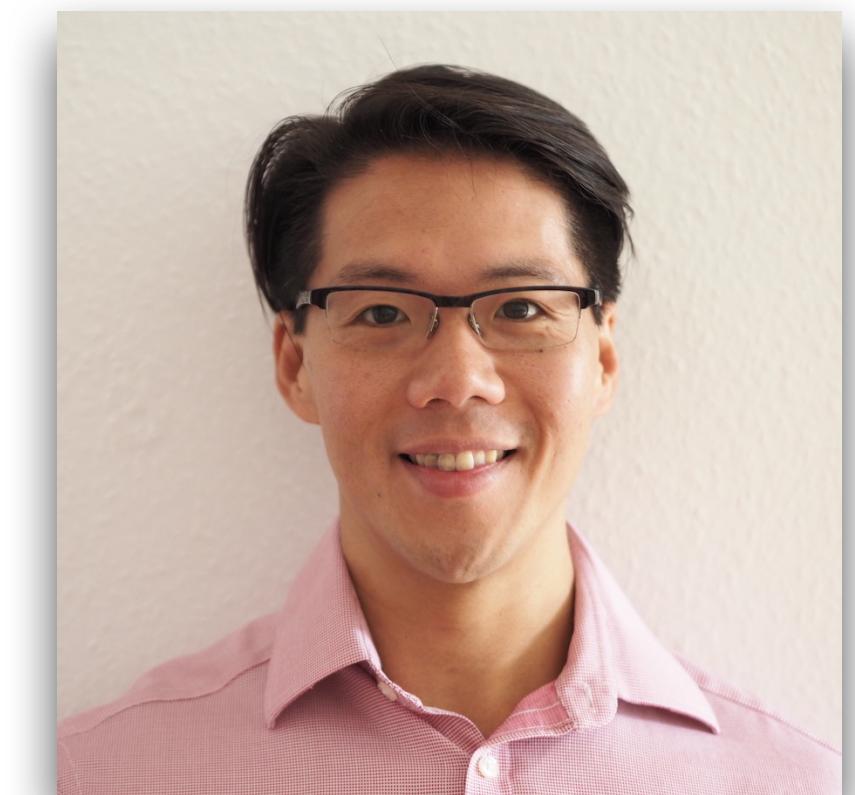
SUPPORT CONSTANT CHANGE

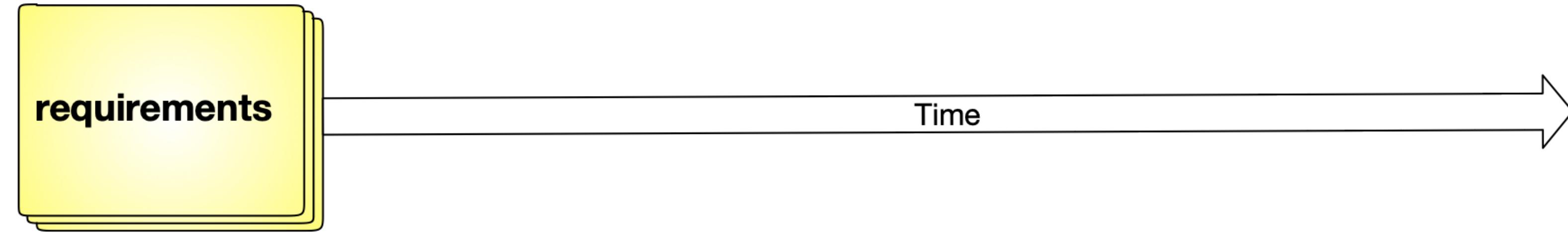


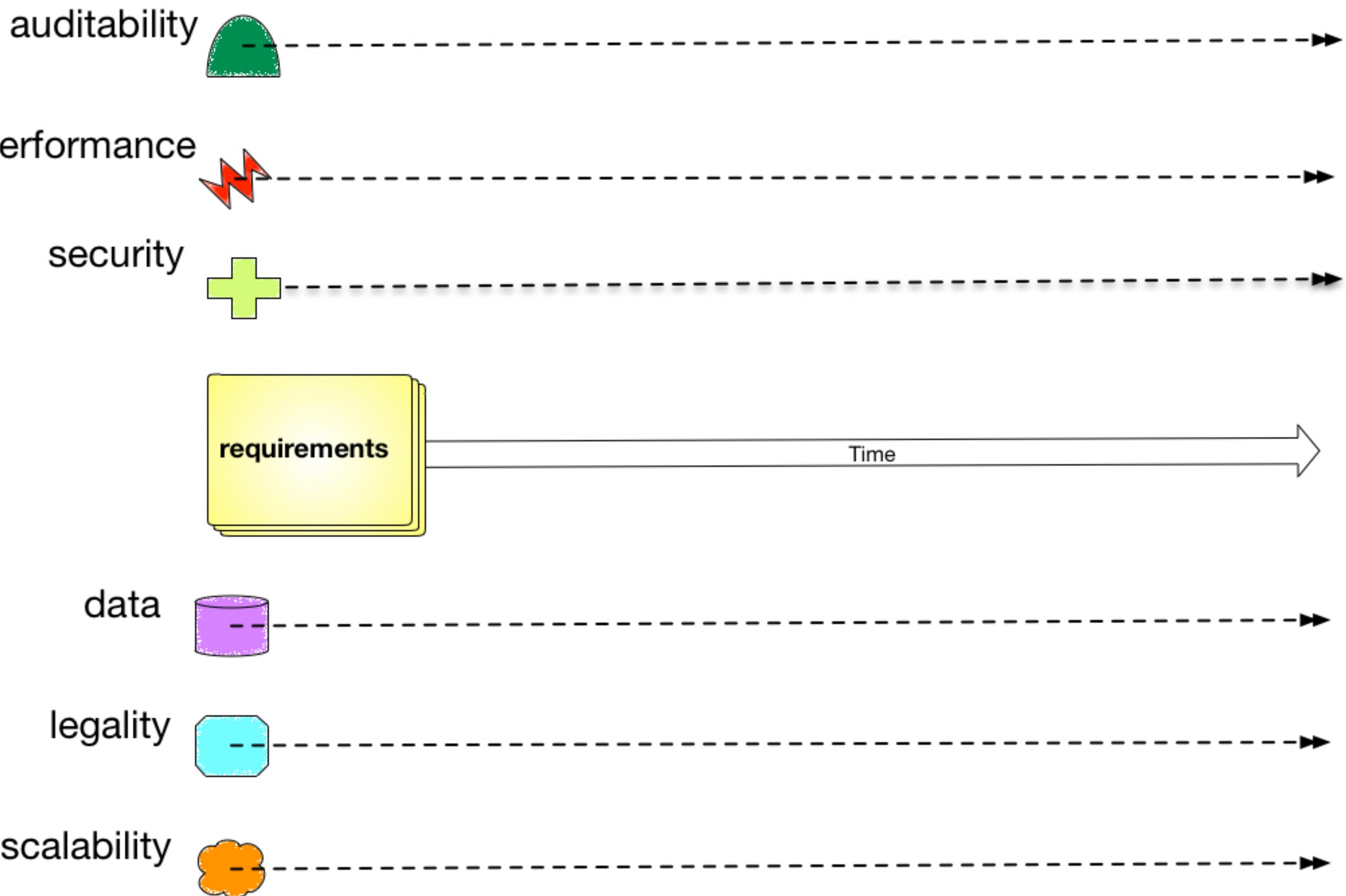
Neal Ford, Rebecca Parsons & Patrick Kua

— fitness functions

— incremental change







*Samenvatting*

reliability	extensibility	repeatability
accessibility	failure transparency	reproducibility
accountability	fault-tolerance	resilience
accuracy	fidelity	responsiveness
adaptability	flexibility	reusability
administrability	inspectability	robustness
affordability	installability	scalability
agility	integrity	seamlessness
auditability	interchangeability	self-sustainability
autonomy	interoperability	serviceability
availability	learnability	supportability
compatibility	maintainability	securability
composability	maneuverability	simplicity
configurability	mobility	stability
correctness	modifiability	standards compliance
credibility	modularity	survivability
customizability	operability	sustainability
debugability	orthogonality	tailorability
degradability	portability	testability
determinability	precision	timeliness
demonstrability	predictability	traceability
dependability	process capabilities	transparency
deployability	productivity	ubiquity
discoverability	provability	understandability
distributability	recoverability	upgradability
durability	relevance	usability
effectiveness		
efficiency		

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

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

# evolvability

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

reconfigurability

relevance

survivability  
sustainability

tailorability

testability

timeliness

traceability

transparency

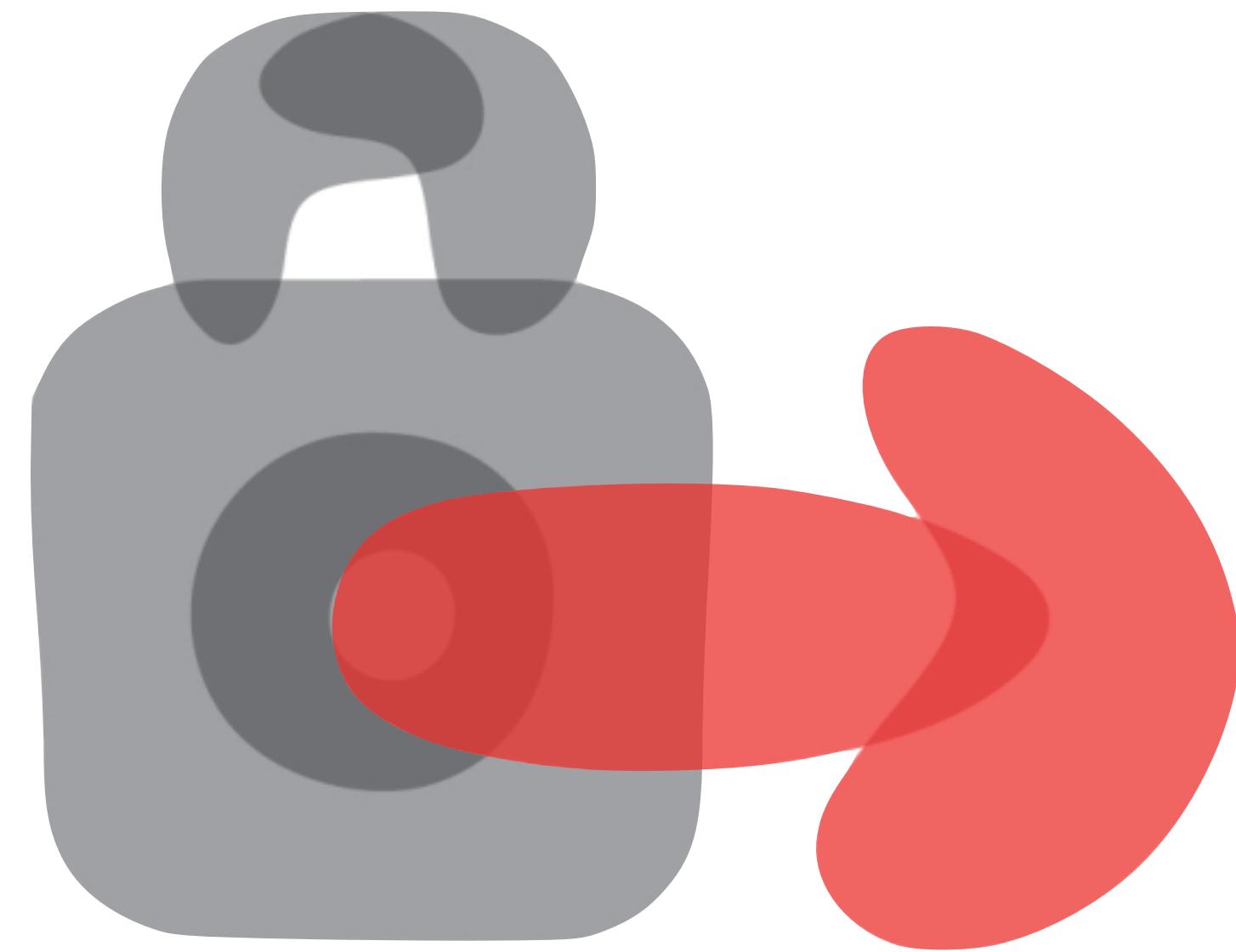
ubiquity

understandability

upgradability

usability

*governance*



# Evolutionary Architecture

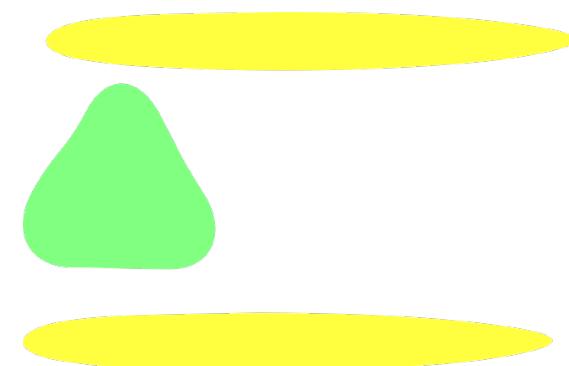
An evolutionary architecture supports  
guided,  
incremental change  
across multiple dimensions.



# Evolutionary Architecture

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



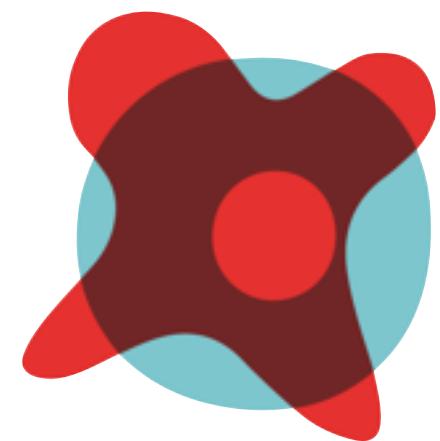
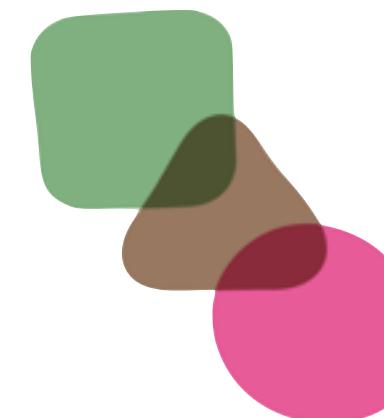
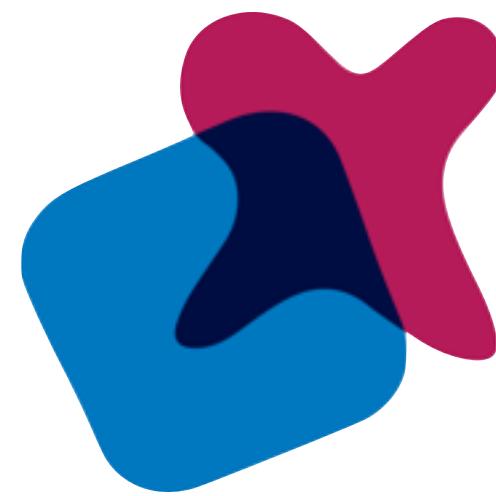
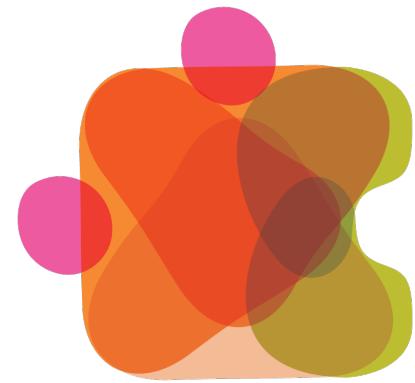


*guided*

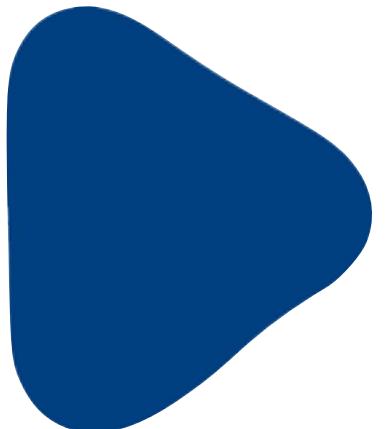
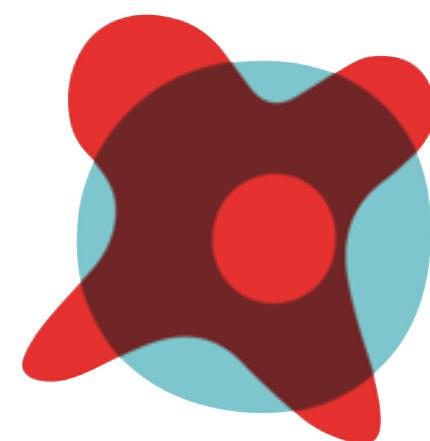
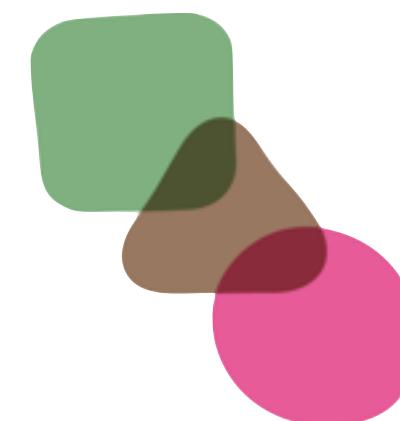
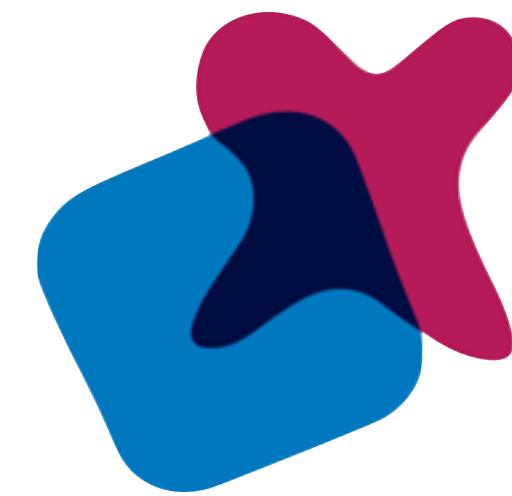
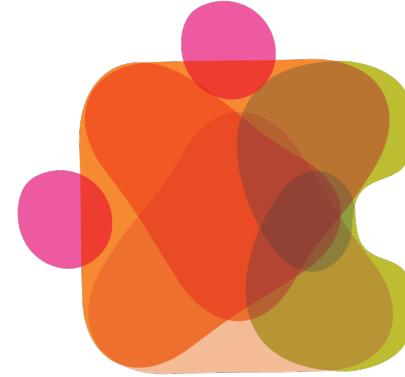
*evolutionary computing fitness function:*

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

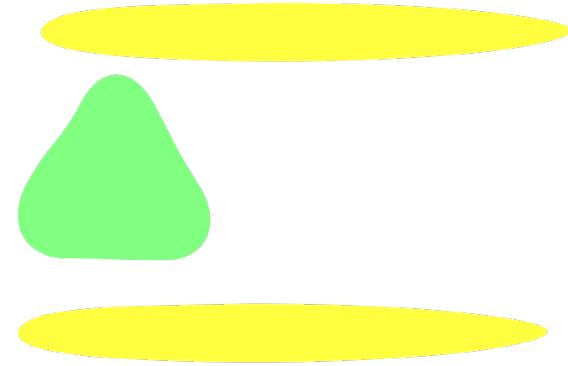
# Traveling Salesman Problem



# Traveling Salesman Problem



fitness function = length of route



*guided*

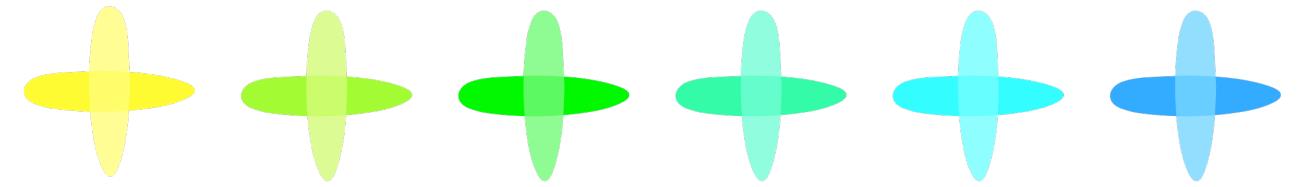
*architectural fitness function:*

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

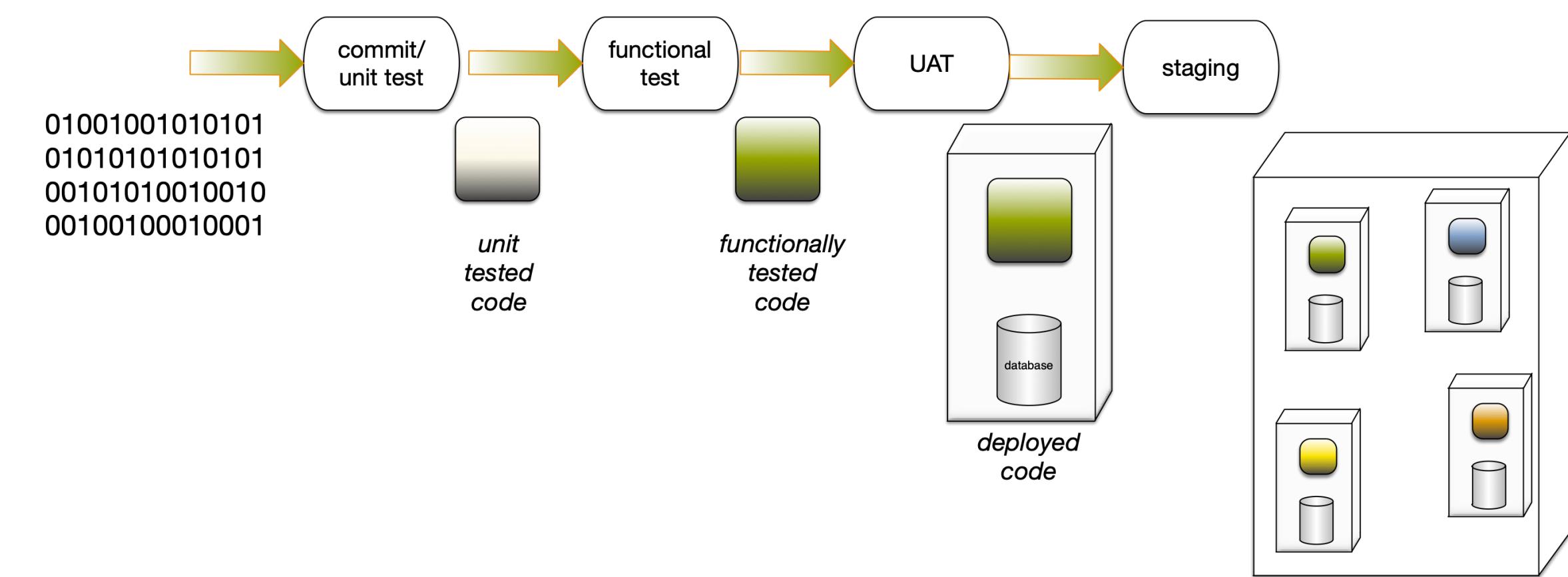
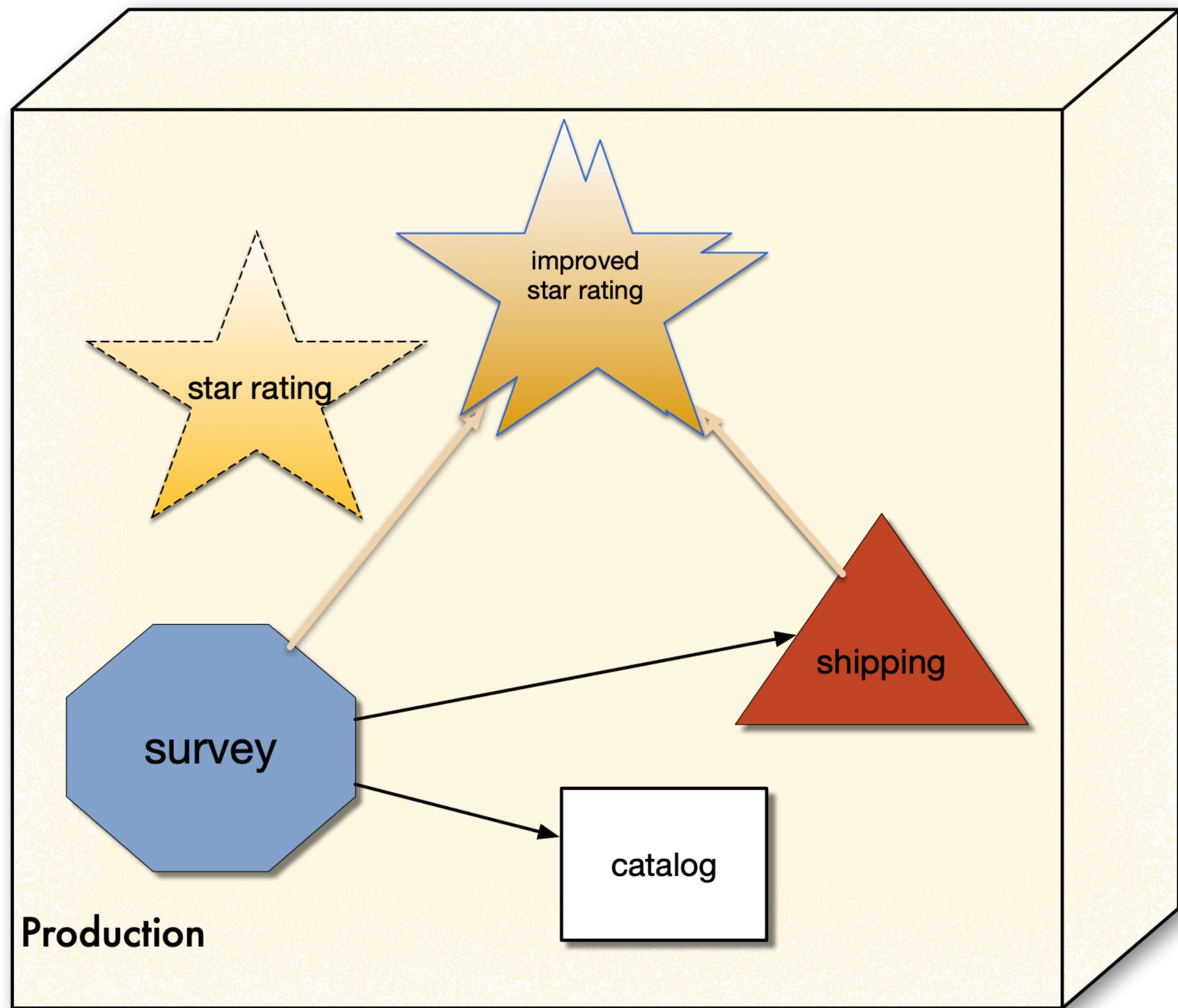
# Evolutionary Architecture

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





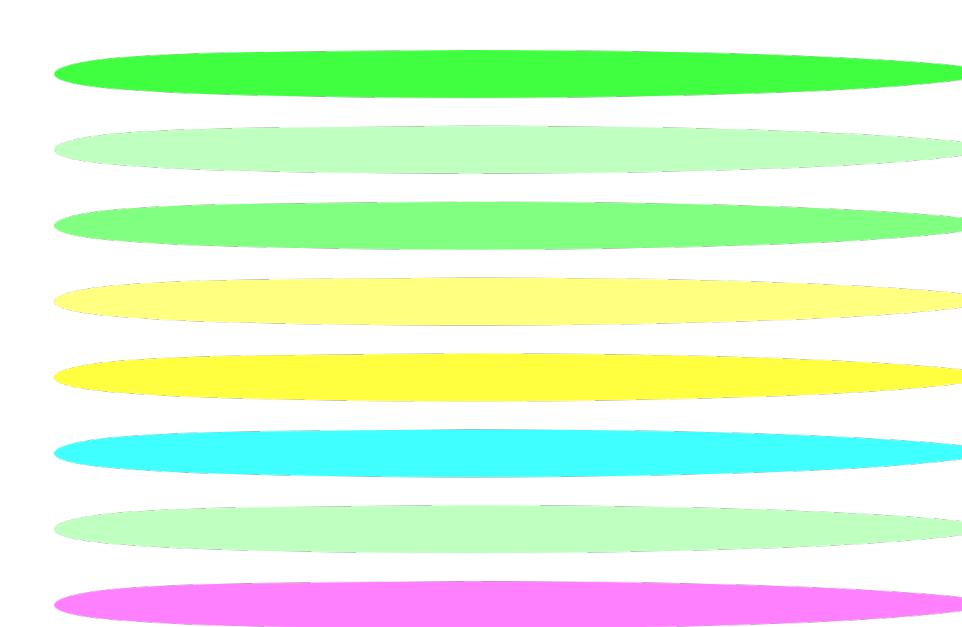
# incremental



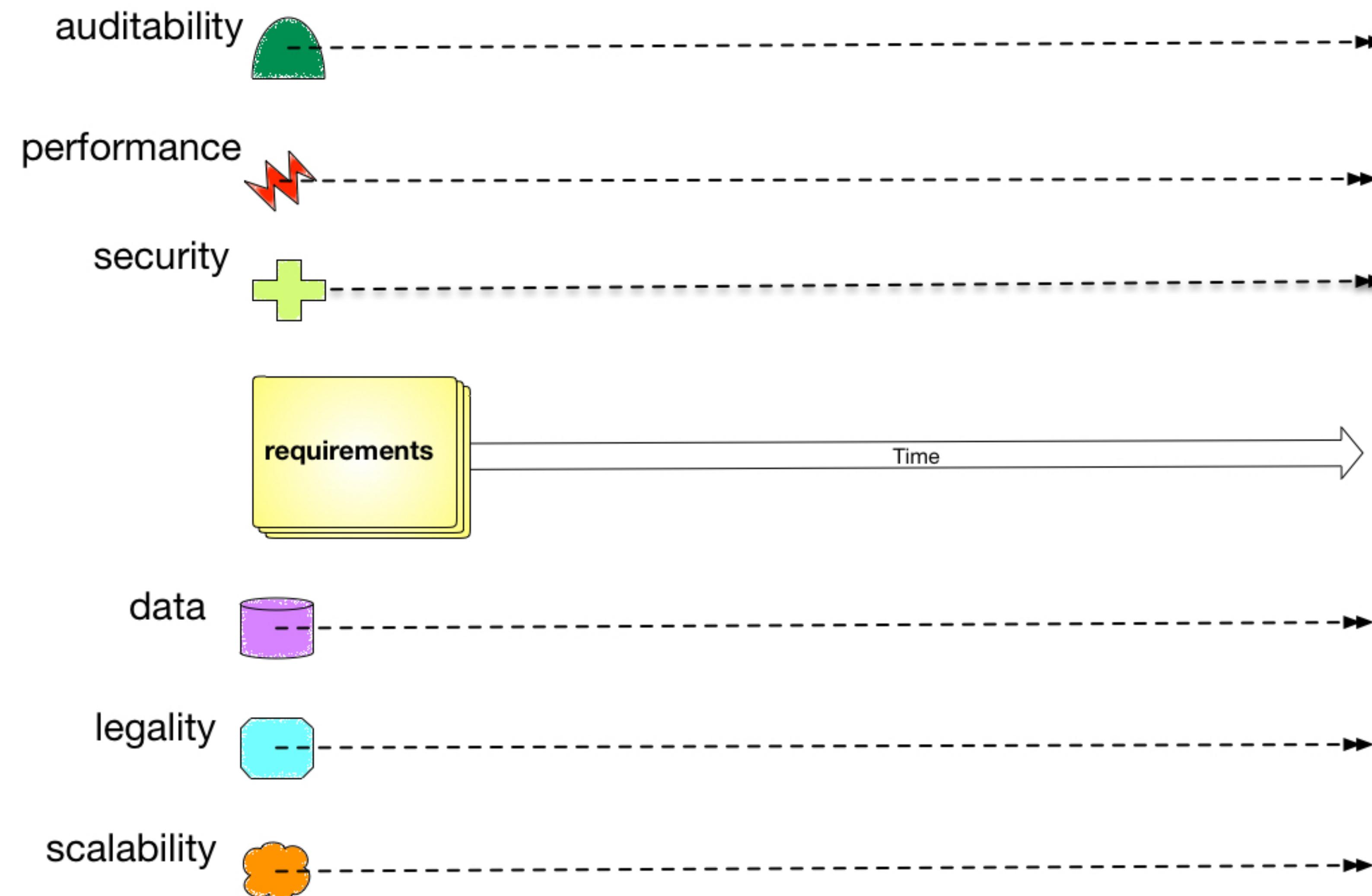
# Evolutionary Architecture

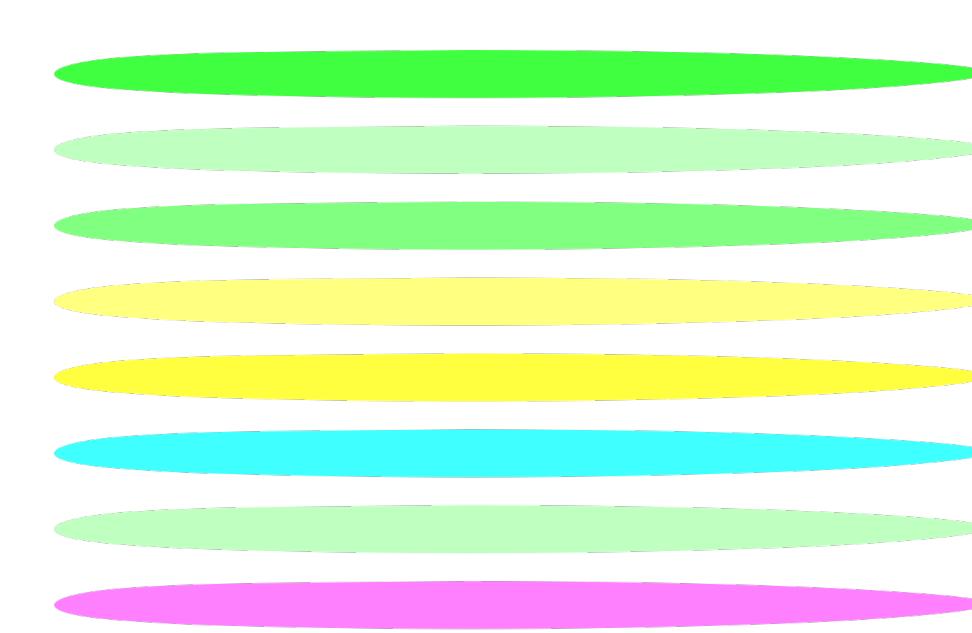
An evolutionary architecture supports  
guided,  
incremental change  
across *multiple dimensions*



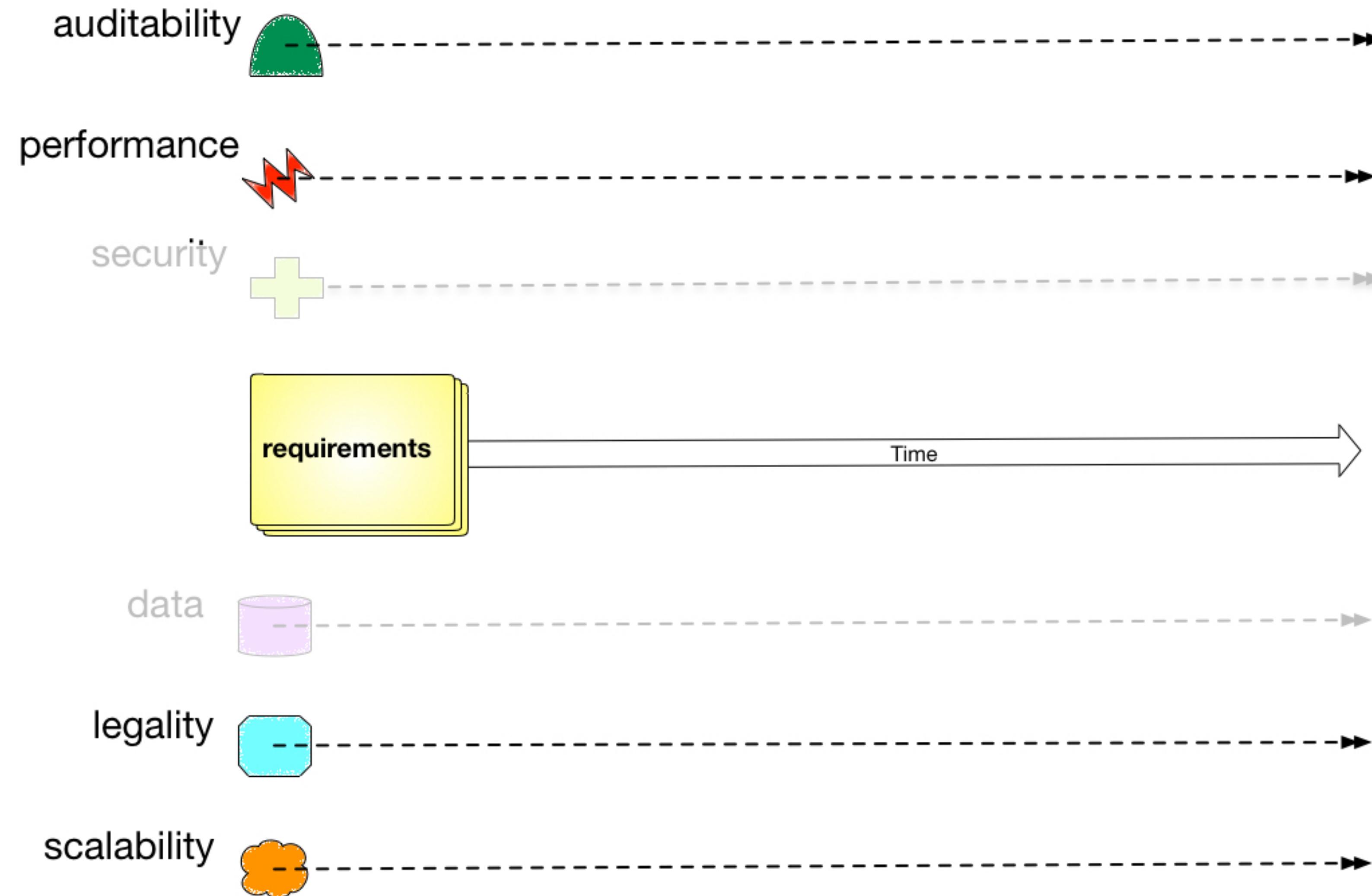


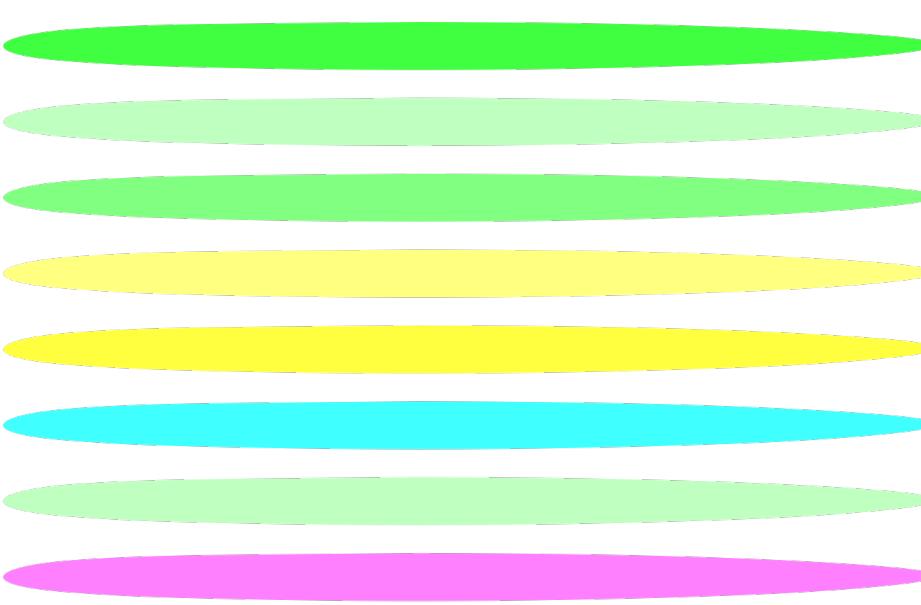
# *multiple dimensions*



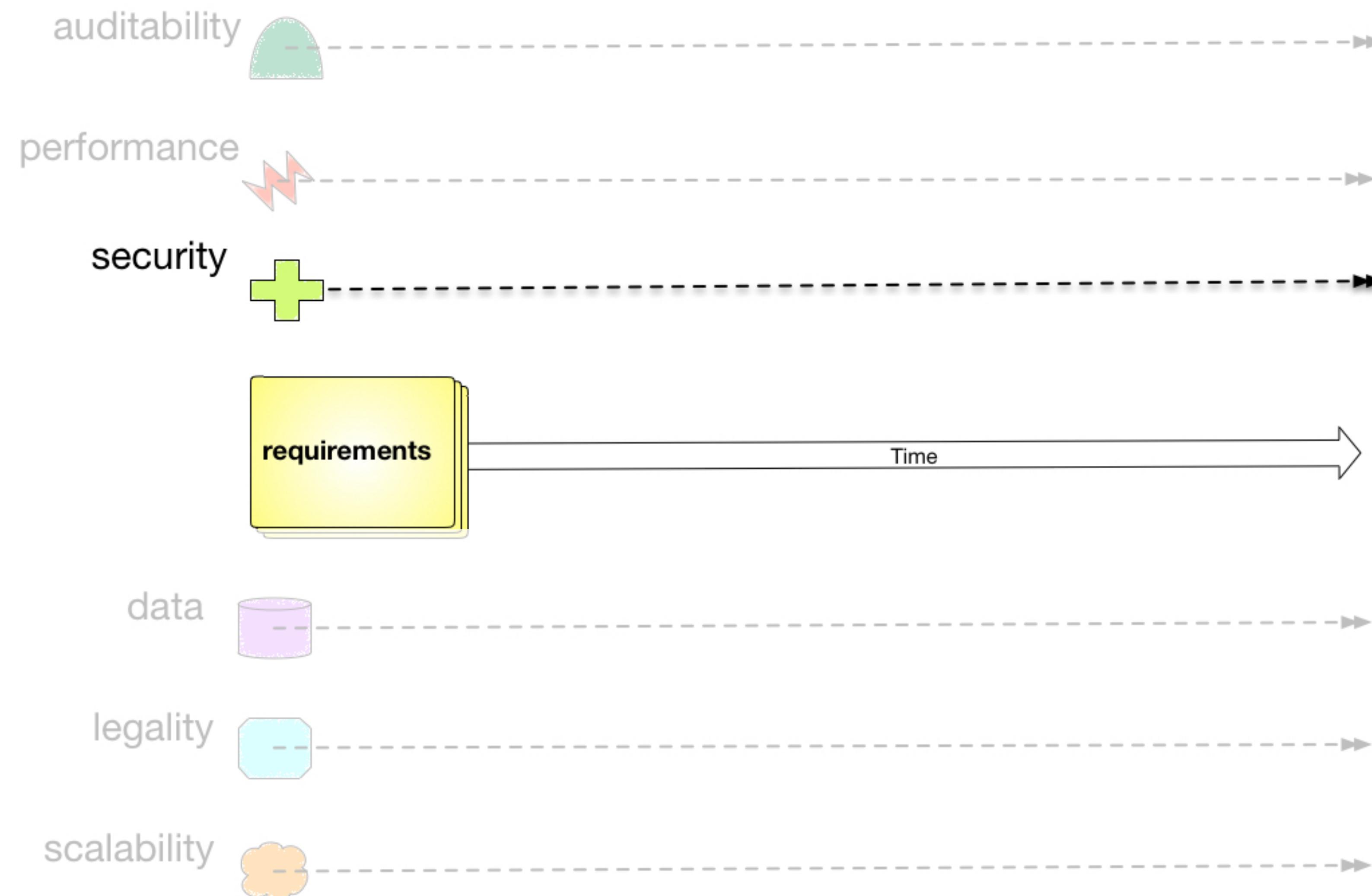


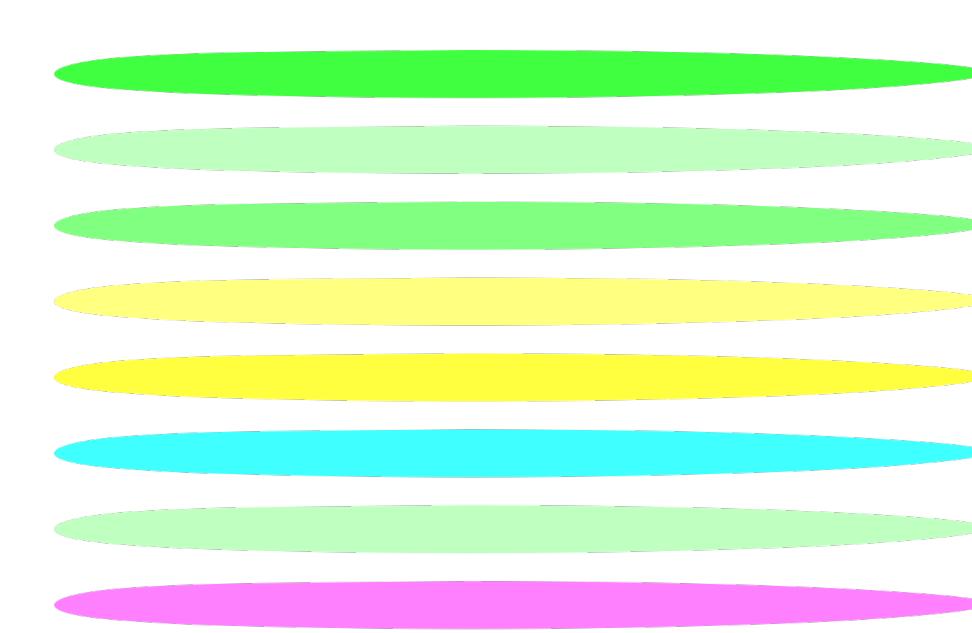
# *multiple dimensions*



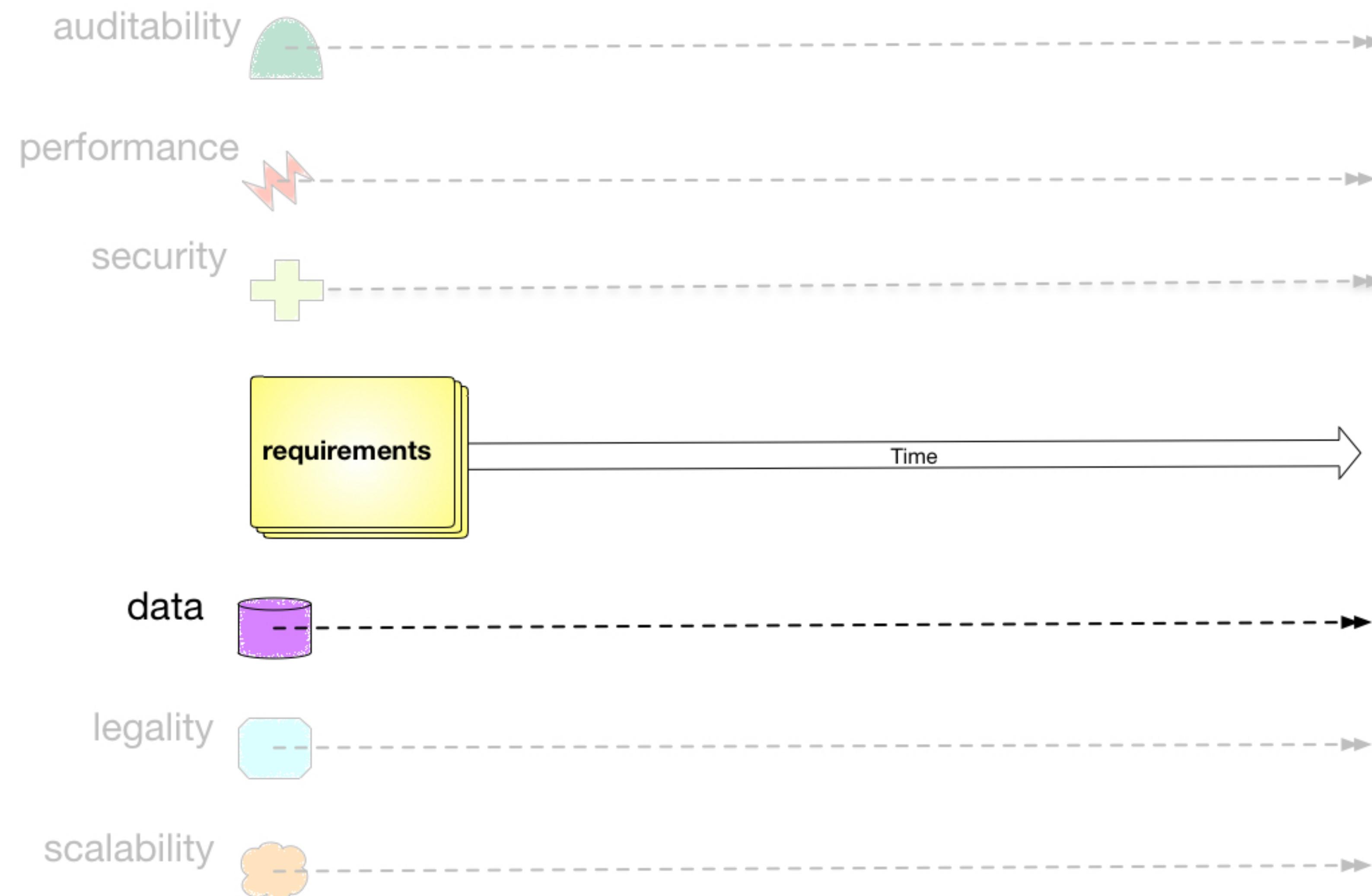


# *multiple dimensions*





# *multiple dimensions*



# Evolutionary Architecture

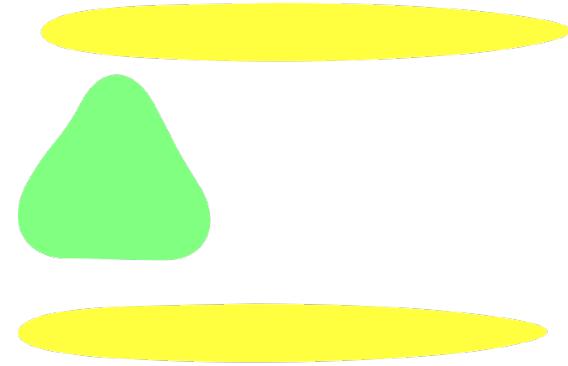
An evolutionary architecture supports  
guided,  
incremental change  
across multiple dimensions.



# Evolutionary Architecture

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



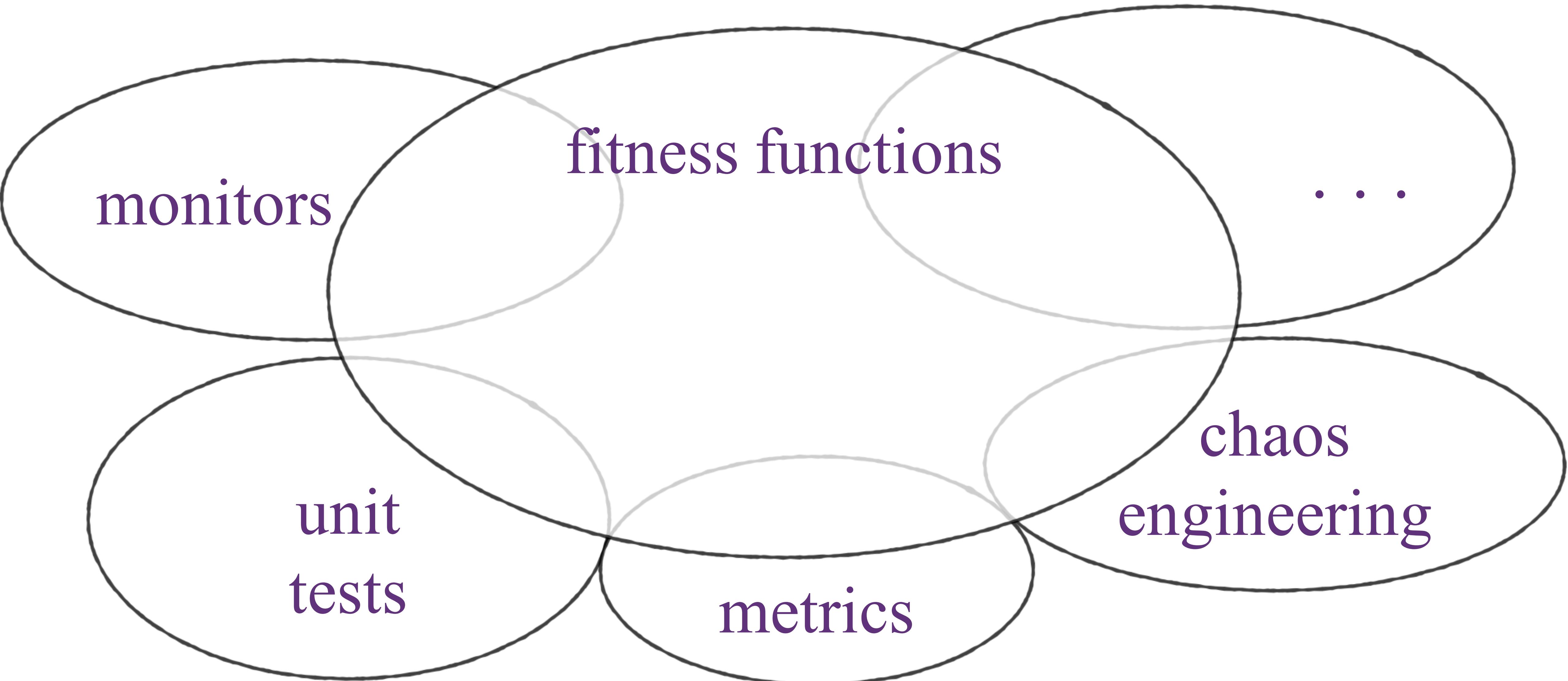


*guided*

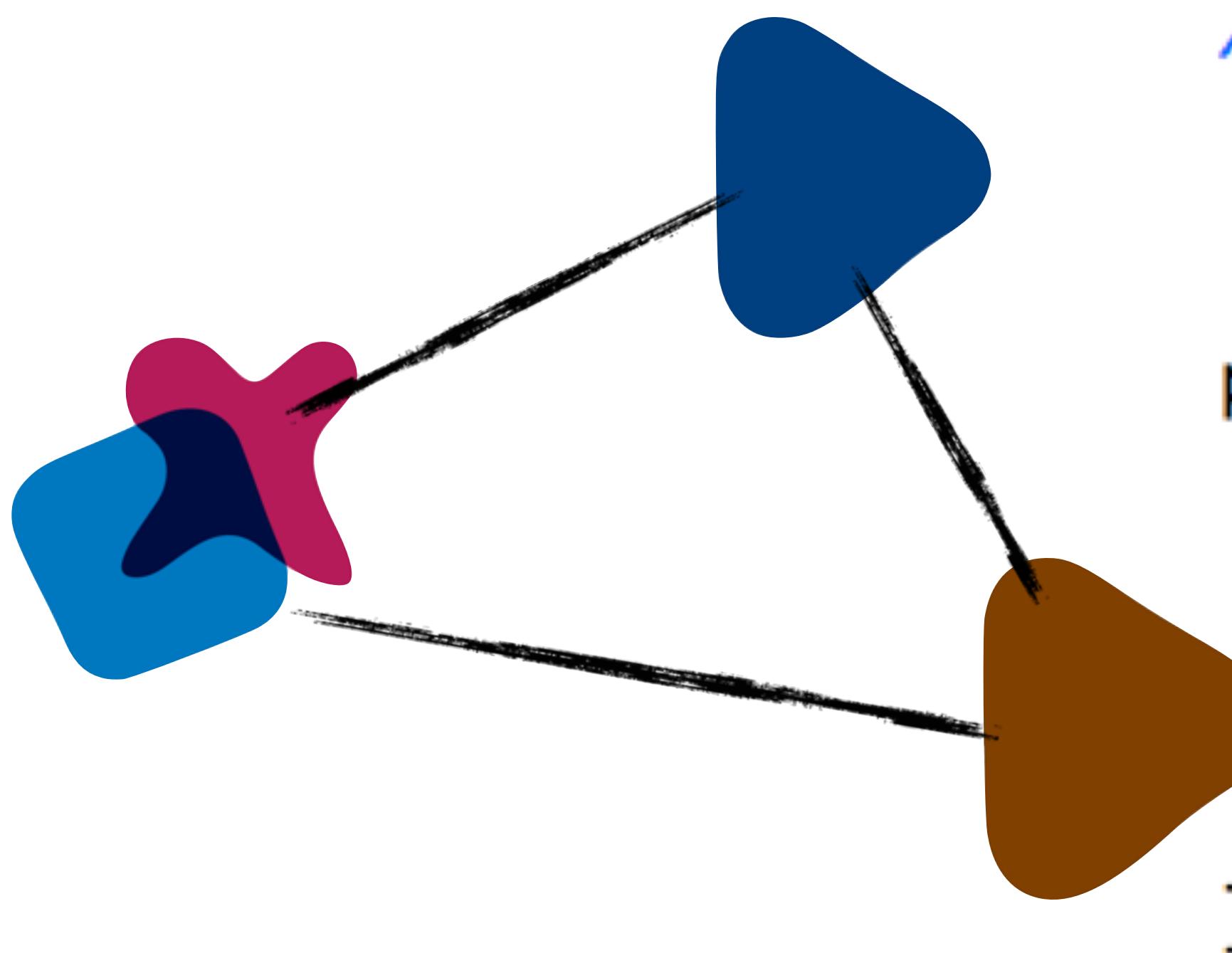
## *architectural fitness function:*

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

# Fitness Functions

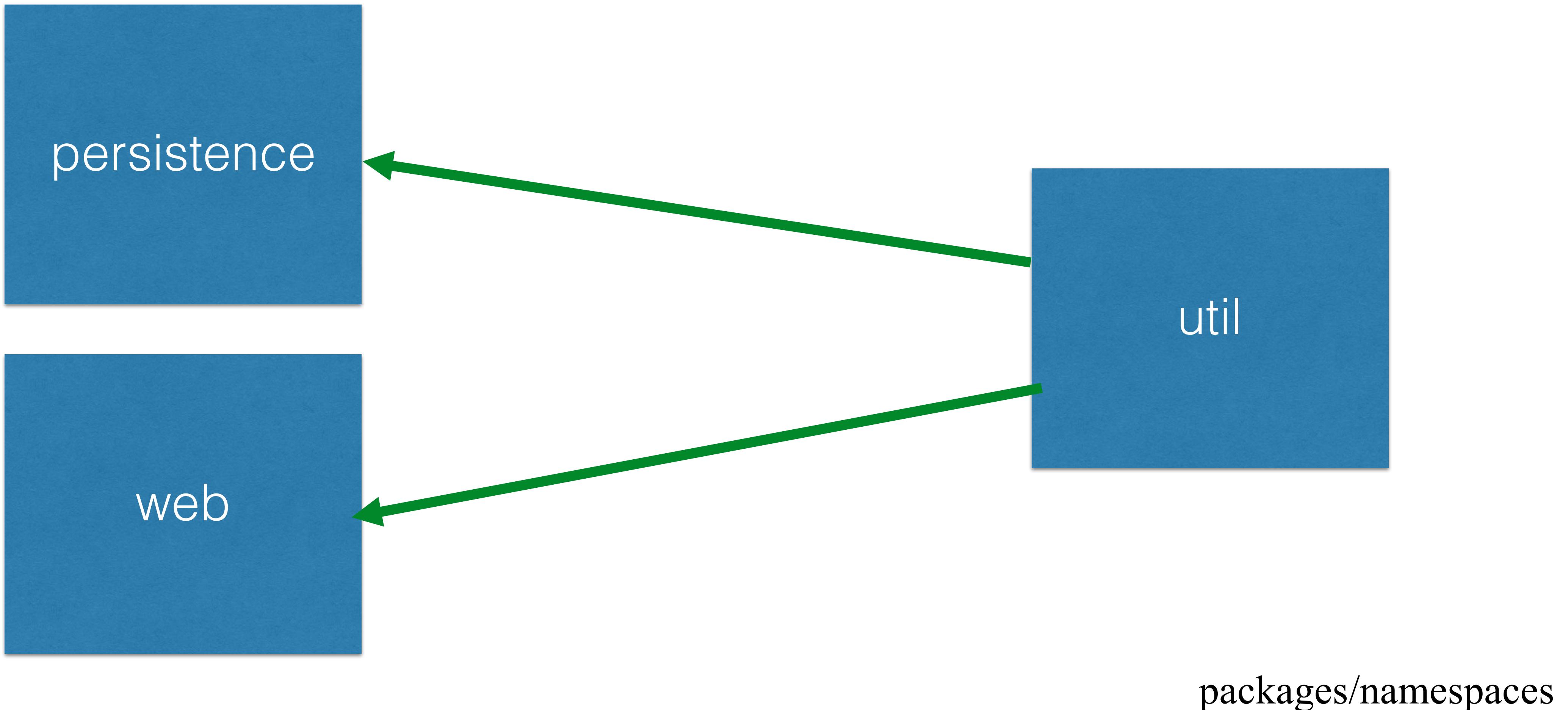


# Cyclic Dependency Function

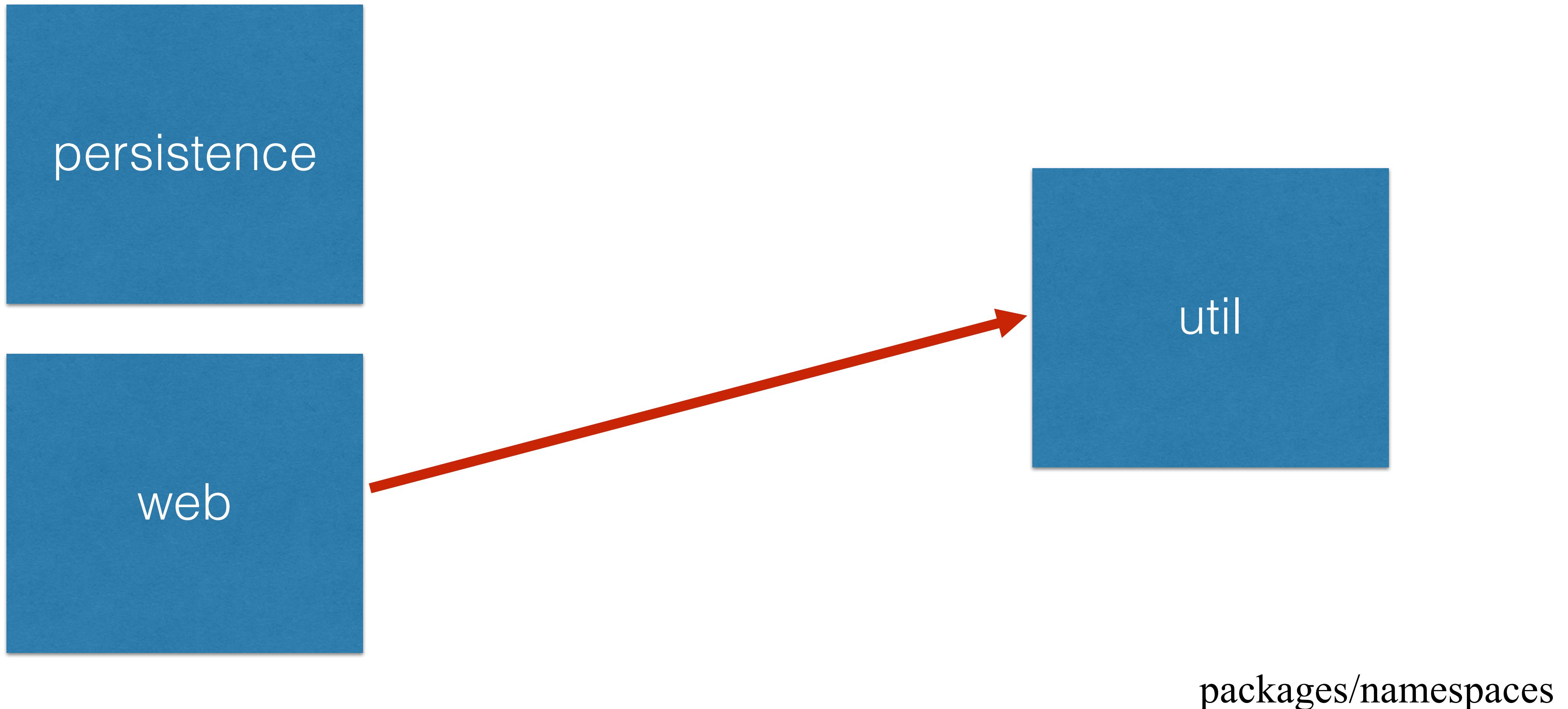


```
/**  
 * Tests that a package dependency cycle does not  
 * exist for any of the analyzed packages.  
 */  
public void testAllPackages() {  
  
    Collection packages = jdepend.analyze();  
  
    assertEquals("Cycles exist",  
                false, jdepend.containsCycles());  
}
```

# Directionality of Imports



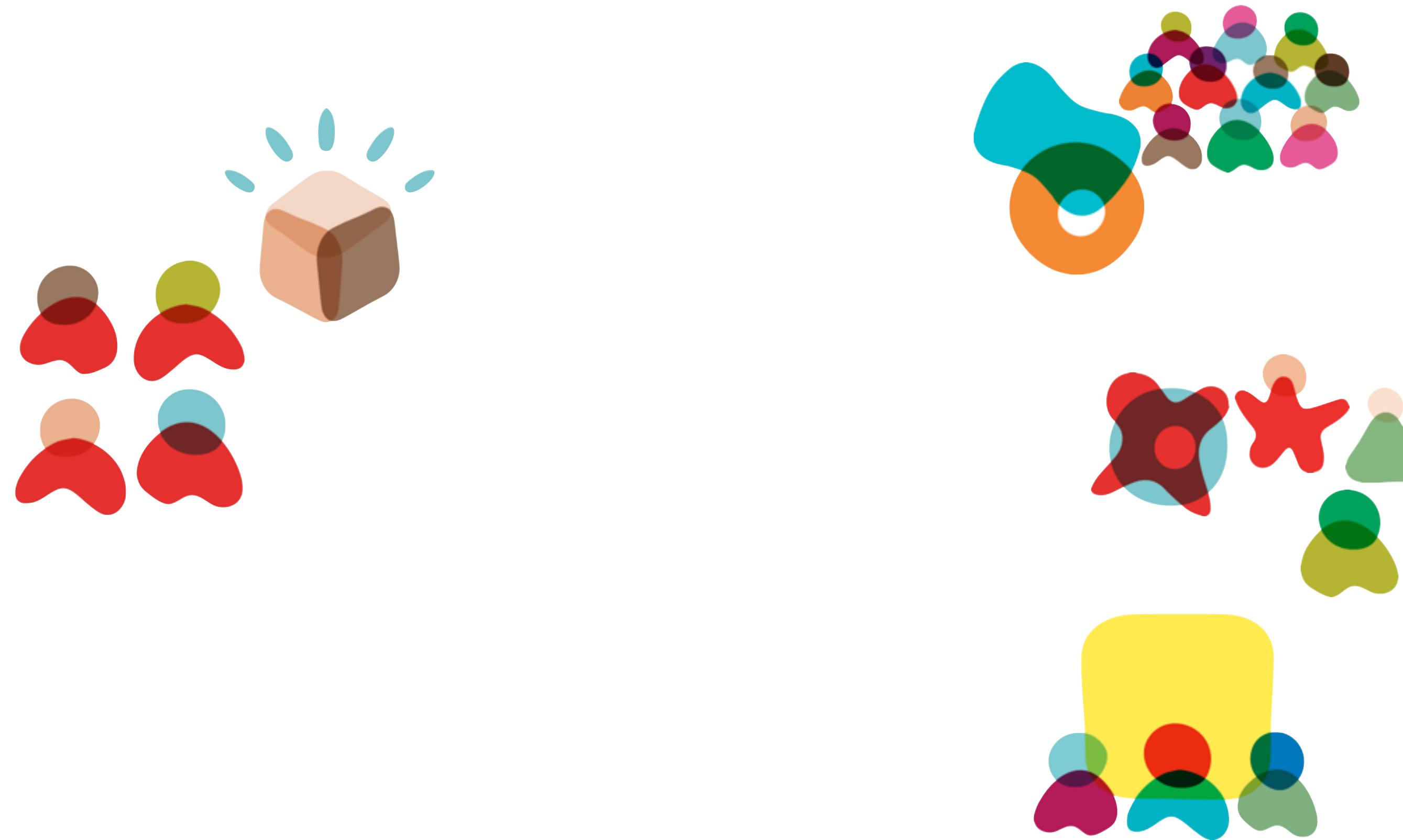
# Directionality of Imports



# Coupling Fitness Function

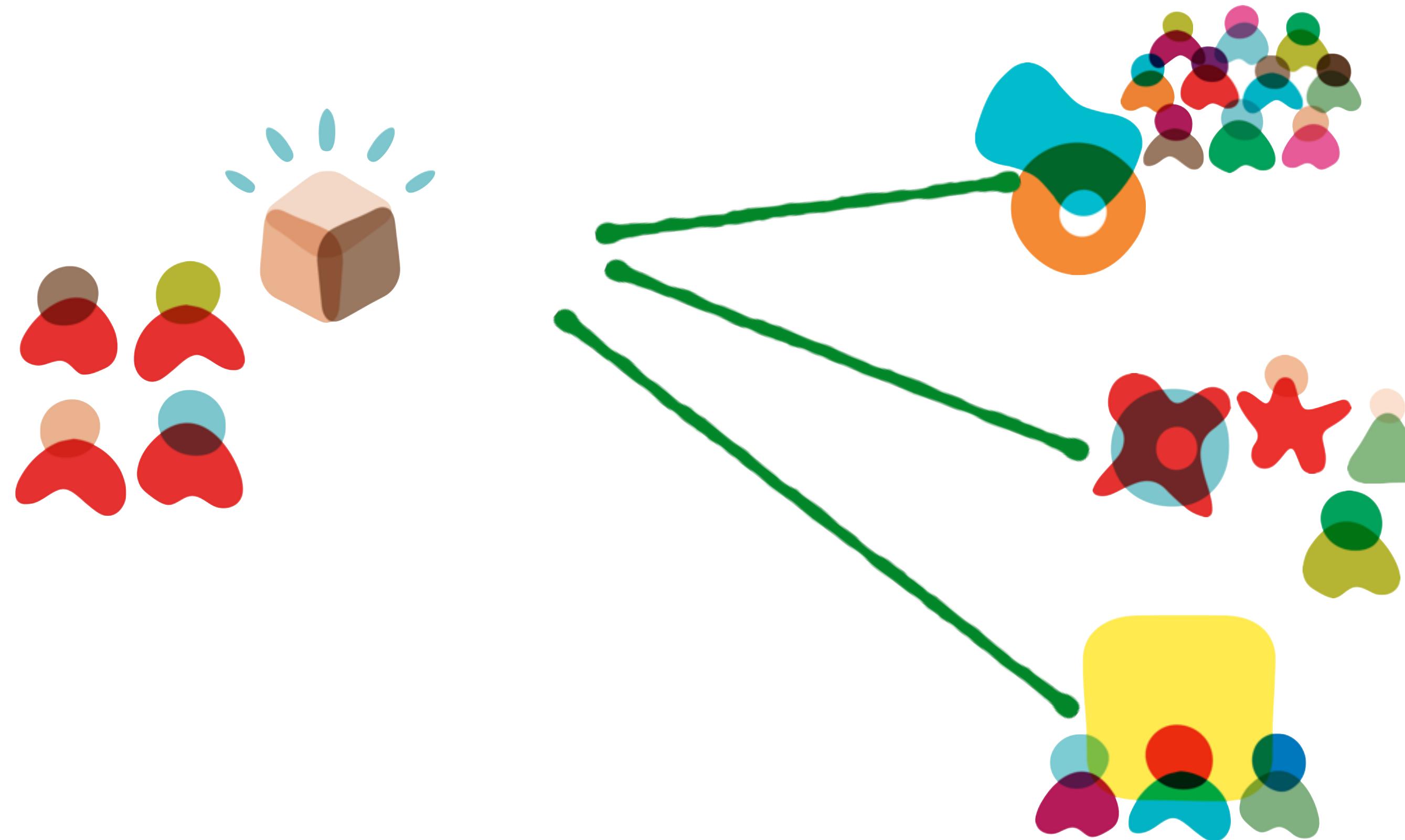
```
public void testMatch() {  
    DependencyConstraint constraint = new DependencyConstraint();  
  
    JavaPackage persistence = constraint.addPackage("com.xyz.persistence");  
    JavaPackage web = constraint.addPackage("com.xyz.web");  
    JavaPackage util = constraint.addPackage("com.xyz.util");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

# Consumer Driven Contracts



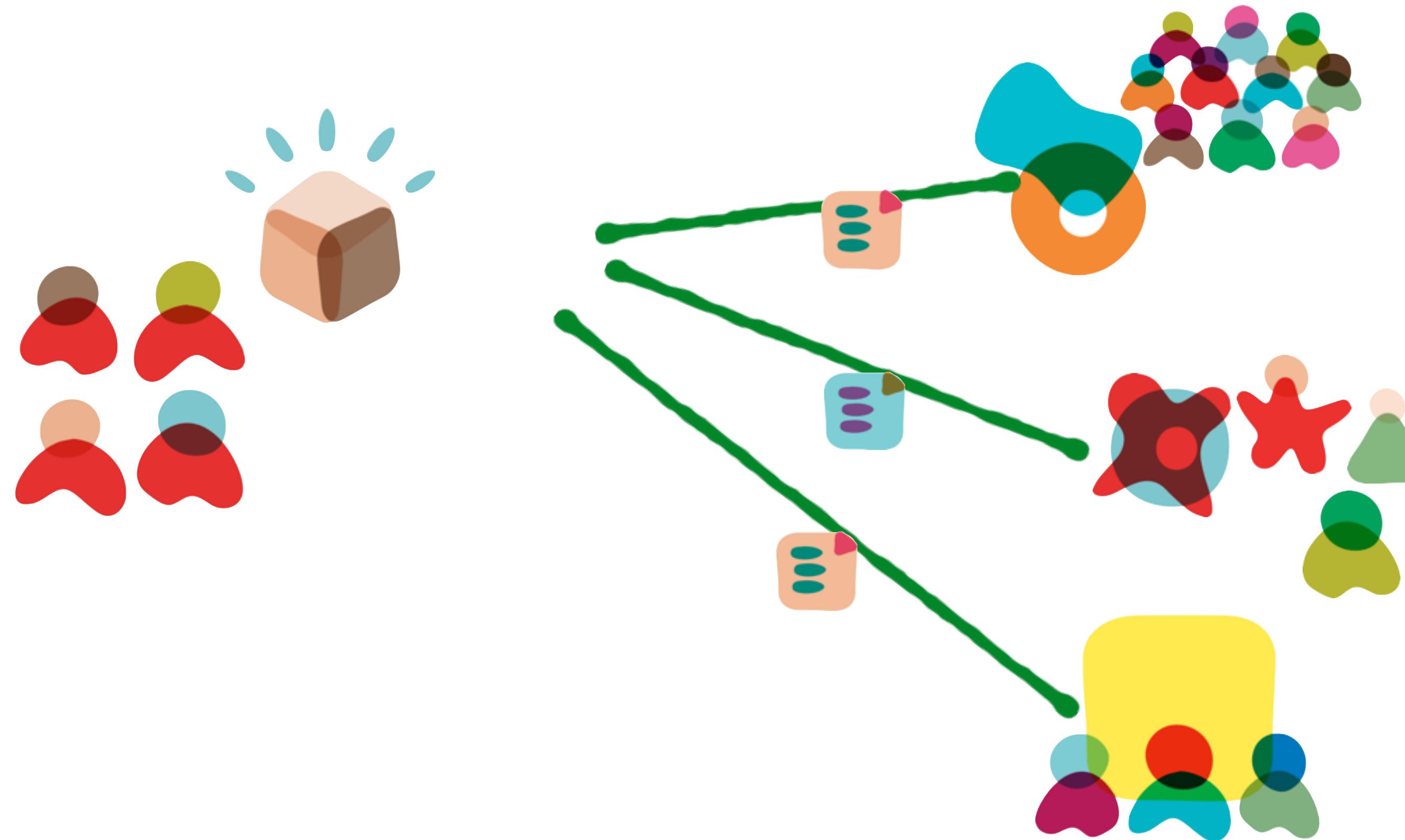
[martinfowler.com/articles/consumerDrivenContracts.html](http://martinfowler.com/articles/consumerDrivenContracts.html)

# Consumer Driven Contracts



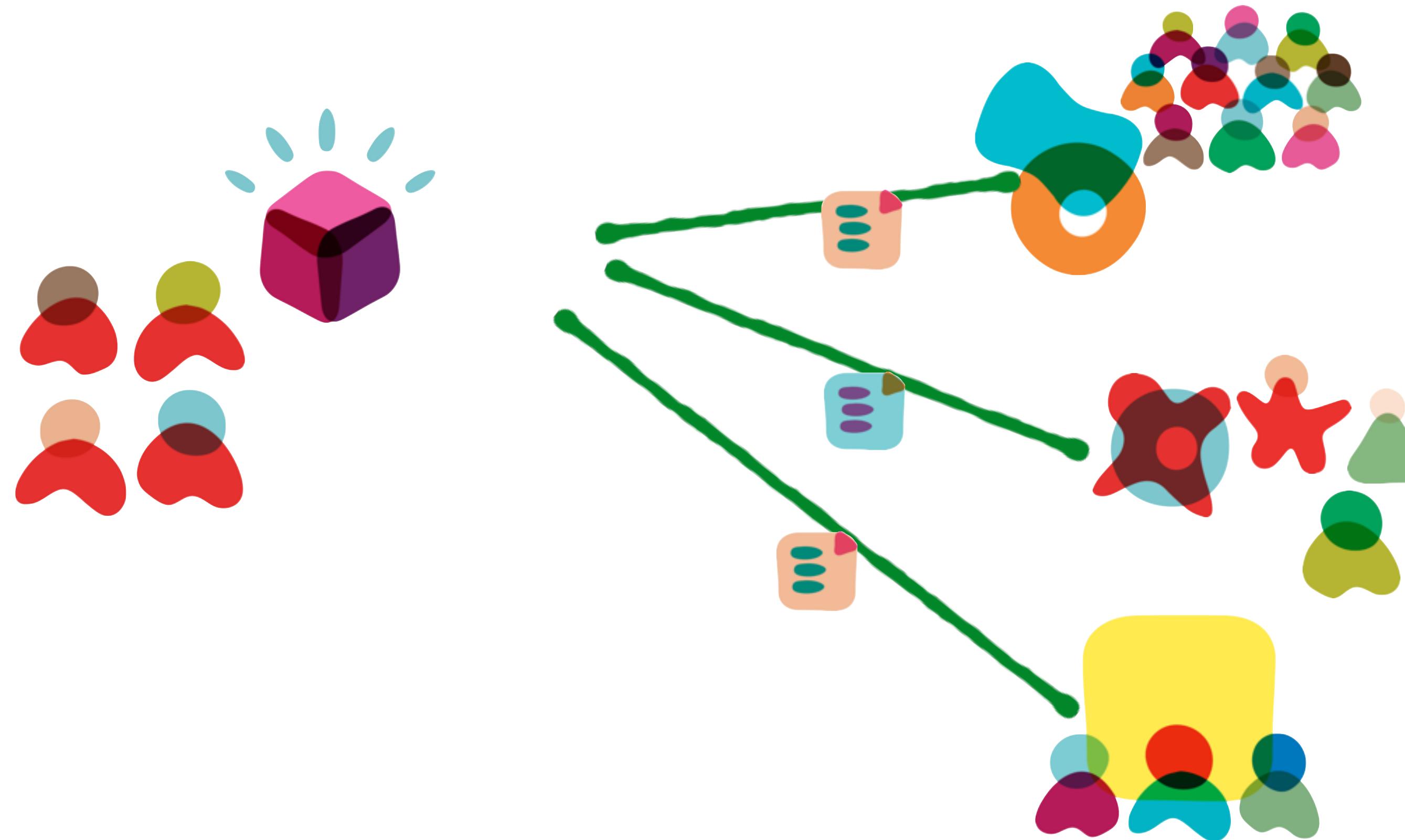
[martinfowler.com/articles/consumerDrivenContracts.html](http://martinfowler.com/articles/consumerDrivenContracts.html)

# Consumer Driven Contracts



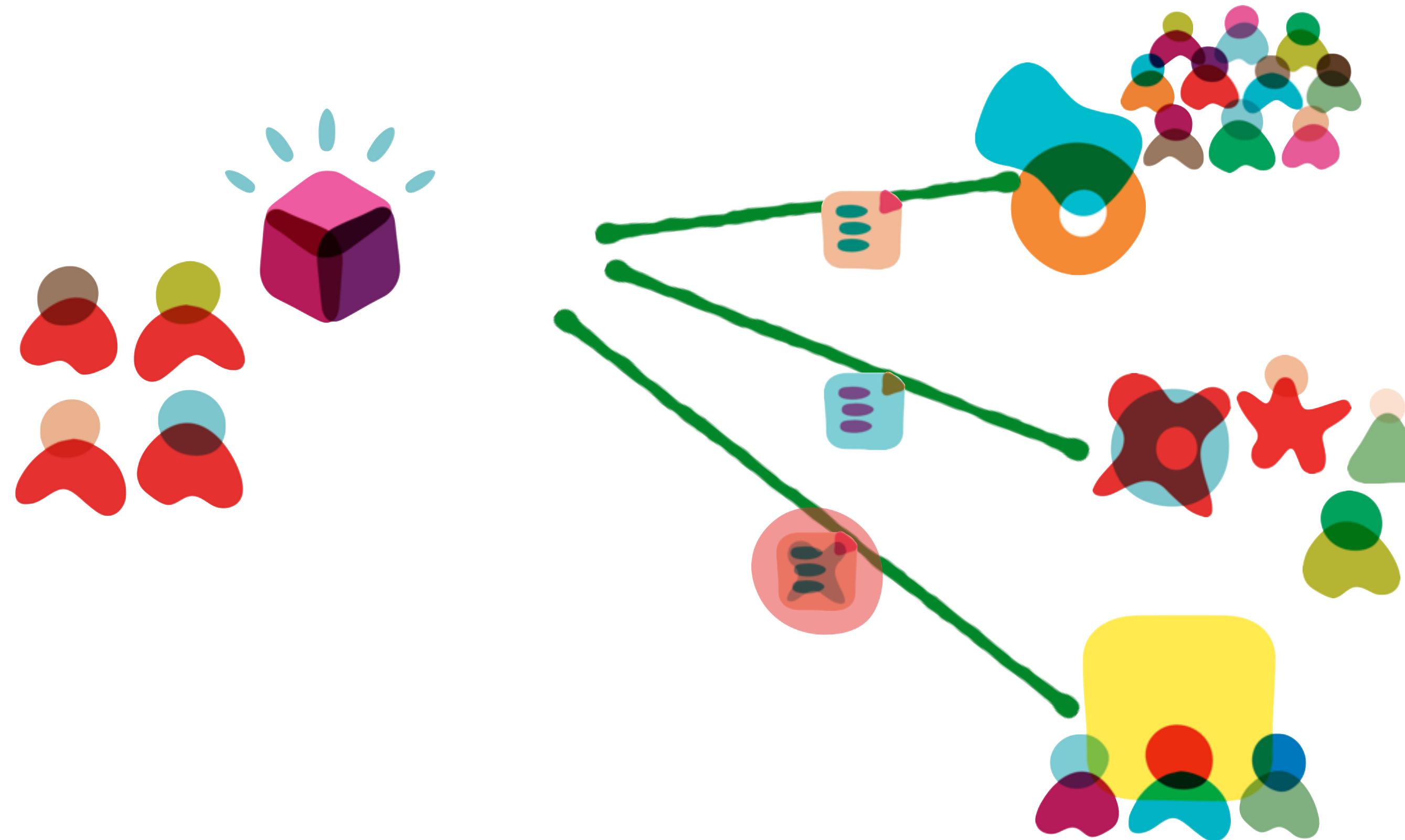
[martinfowler.com/articles/consumerDrivenContracts.html](http://martinfowler.com/articles/consumerDrivenContracts.html)

# Consumer Driven Contracts



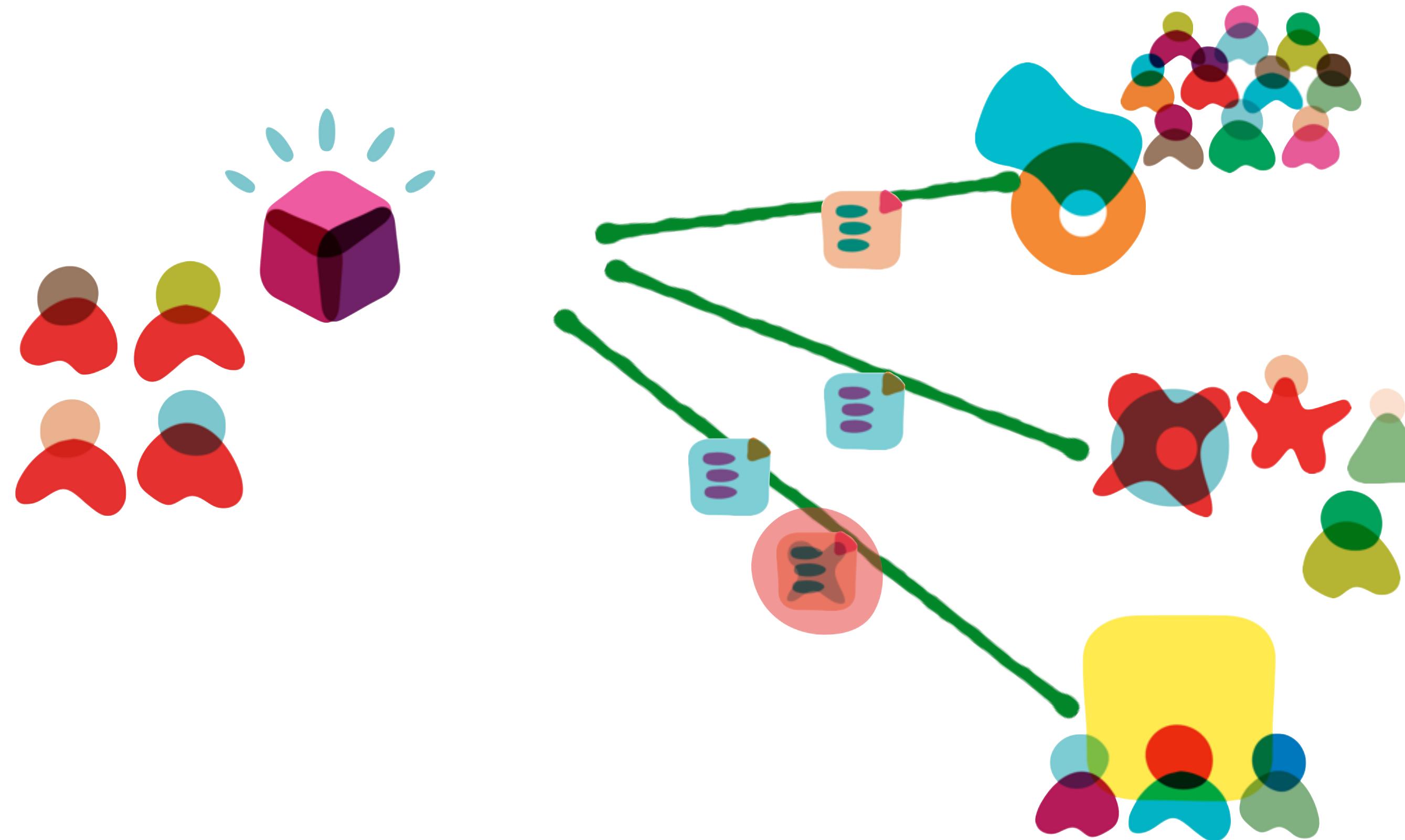
[martinfowler.com/articles/consumerDrivenContracts.html](http://martinfowler.com/articles/consumerDrivenContracts.html)

# Consumer Driven Contracts



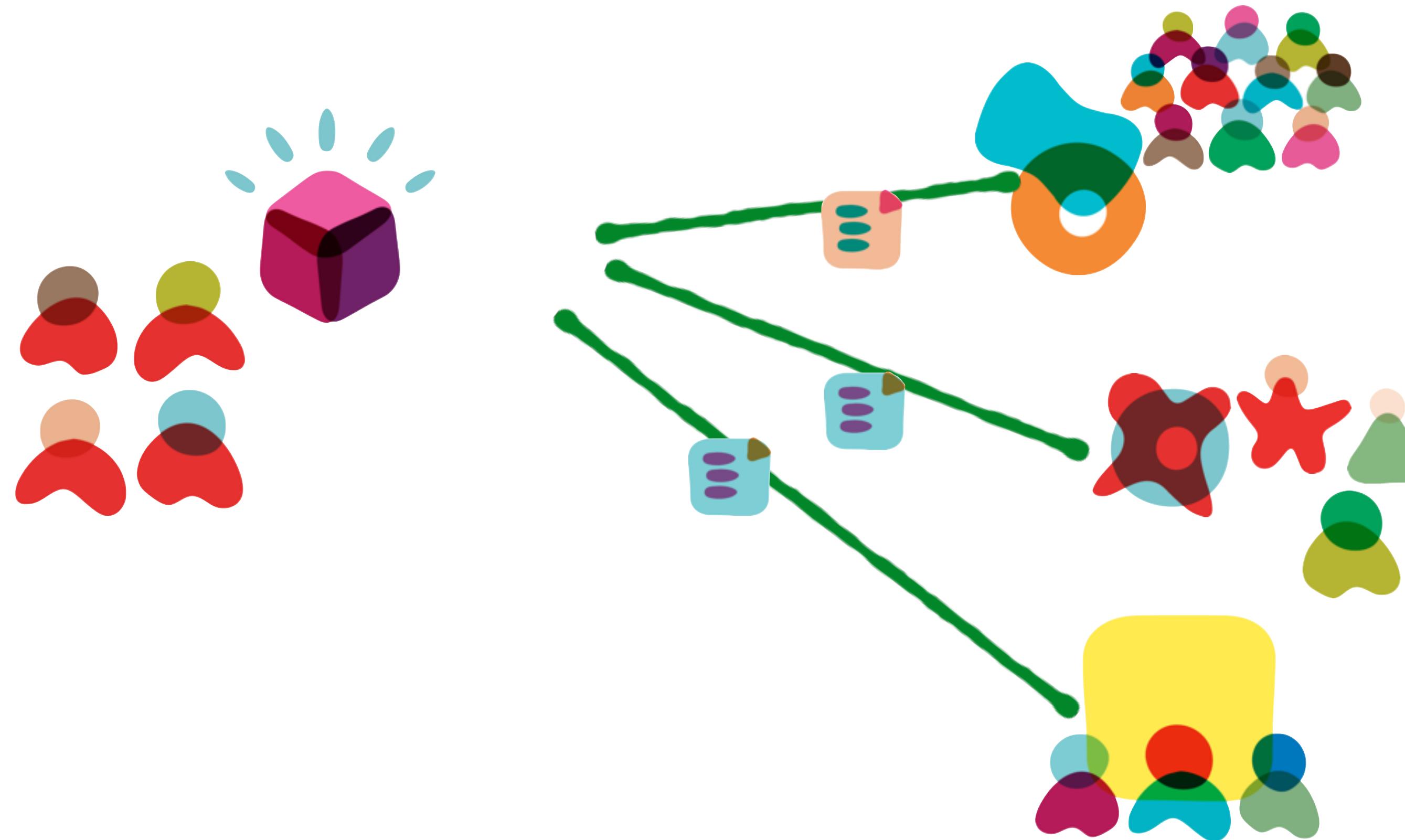
[martinfowler.com/articles/consumerDrivenContracts.html](http://martinfowler.com/articles/consumerDrivenContracts.html)

# Consumer Driven Contracts



[martinfowler.com/articles/consumerDrivenContracts.html](http://martinfowler.com/articles/consumerDrivenContracts.html)

# Consumer Driven Contracts



[martinfowler.com/articles/consumerDrivenContracts.html](http://martinfowler.com/articles/consumerDrivenContracts.html)







chaos monkey



chaos  
gorilla



latency  
monkey

doctor  
monkey





janitor  
monkey

conformity  
monkey

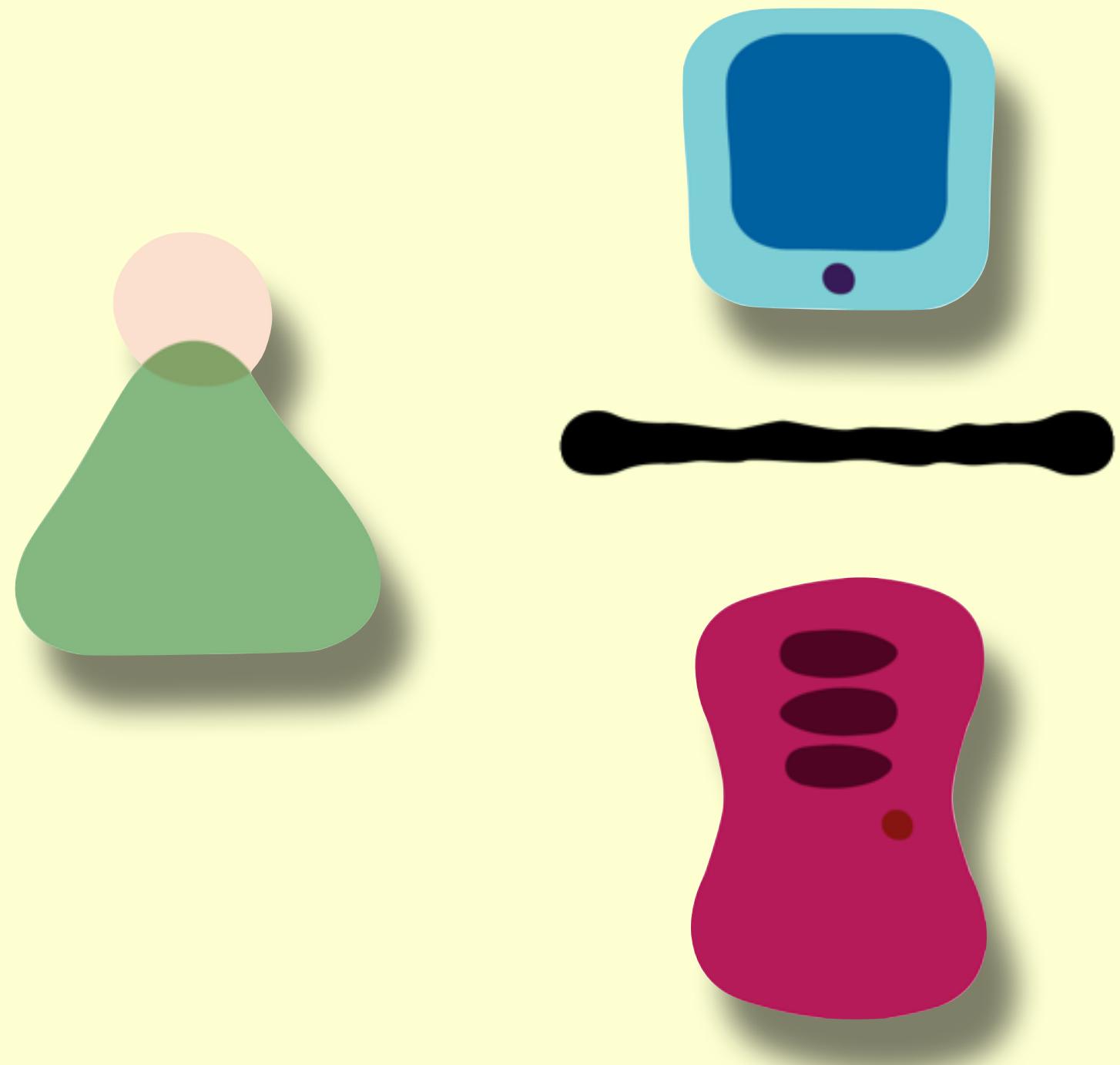


security monkey

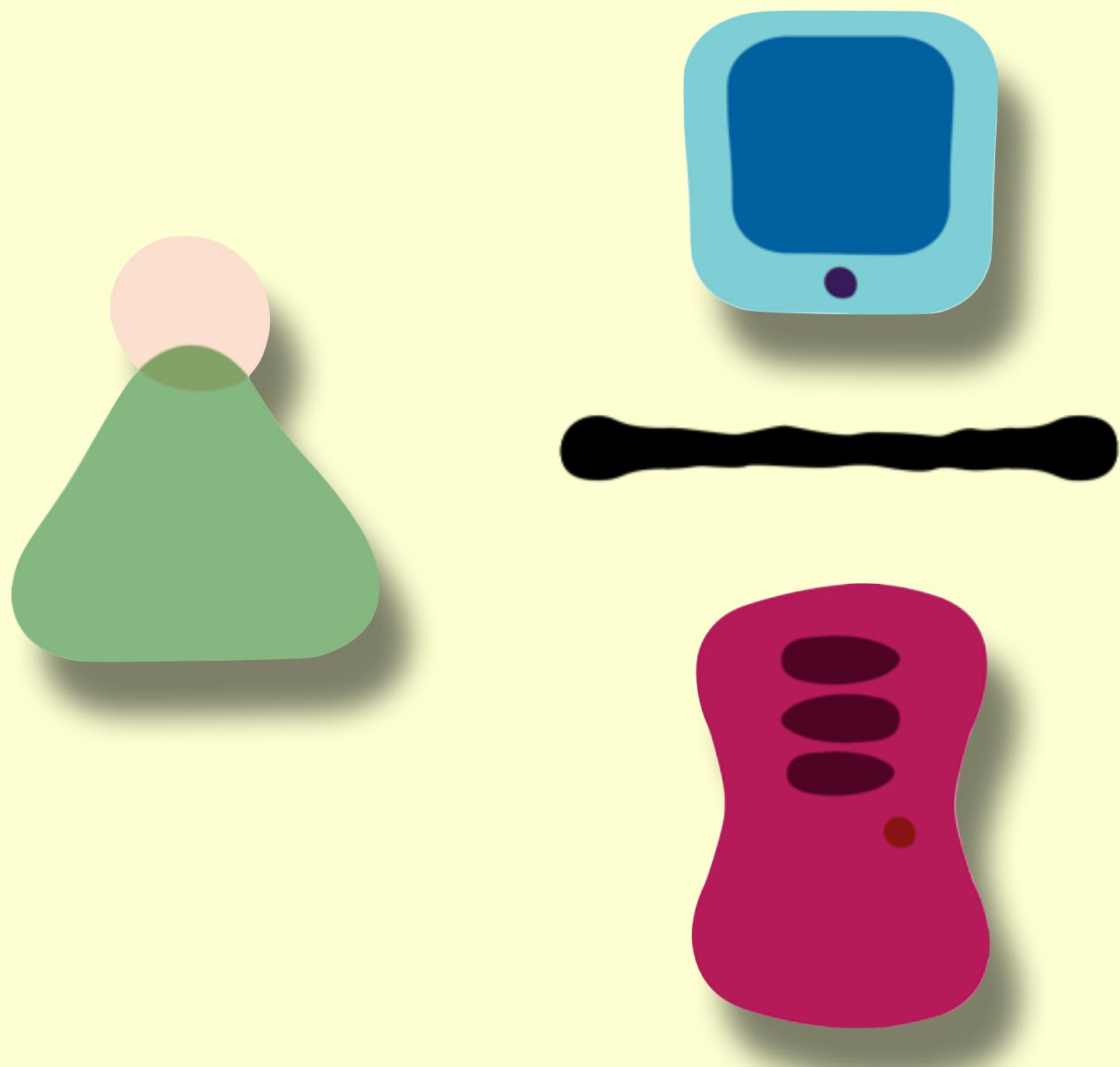
conformity  
monkey



# Governing Code Quality

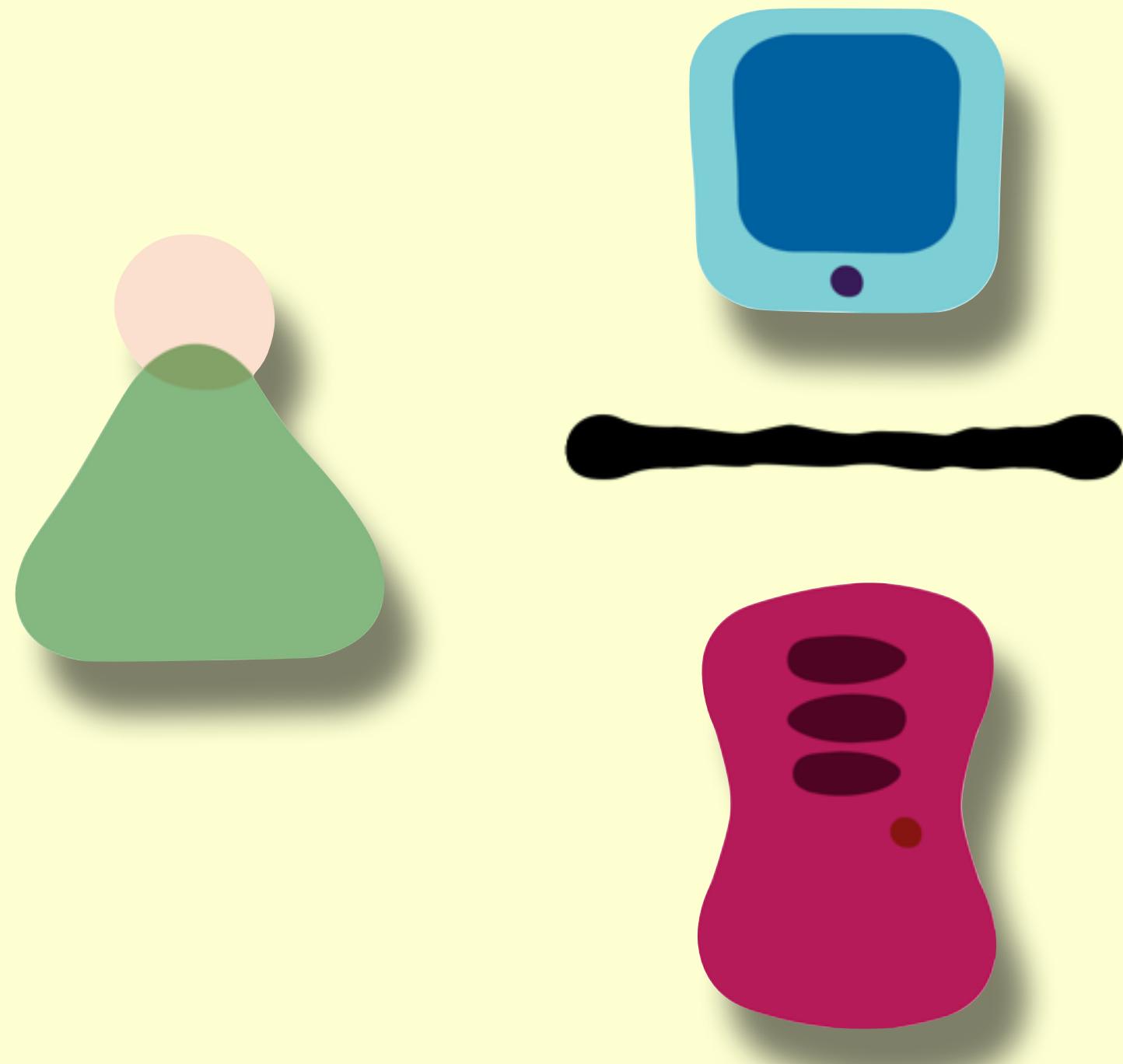


# Governing Code Quality



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

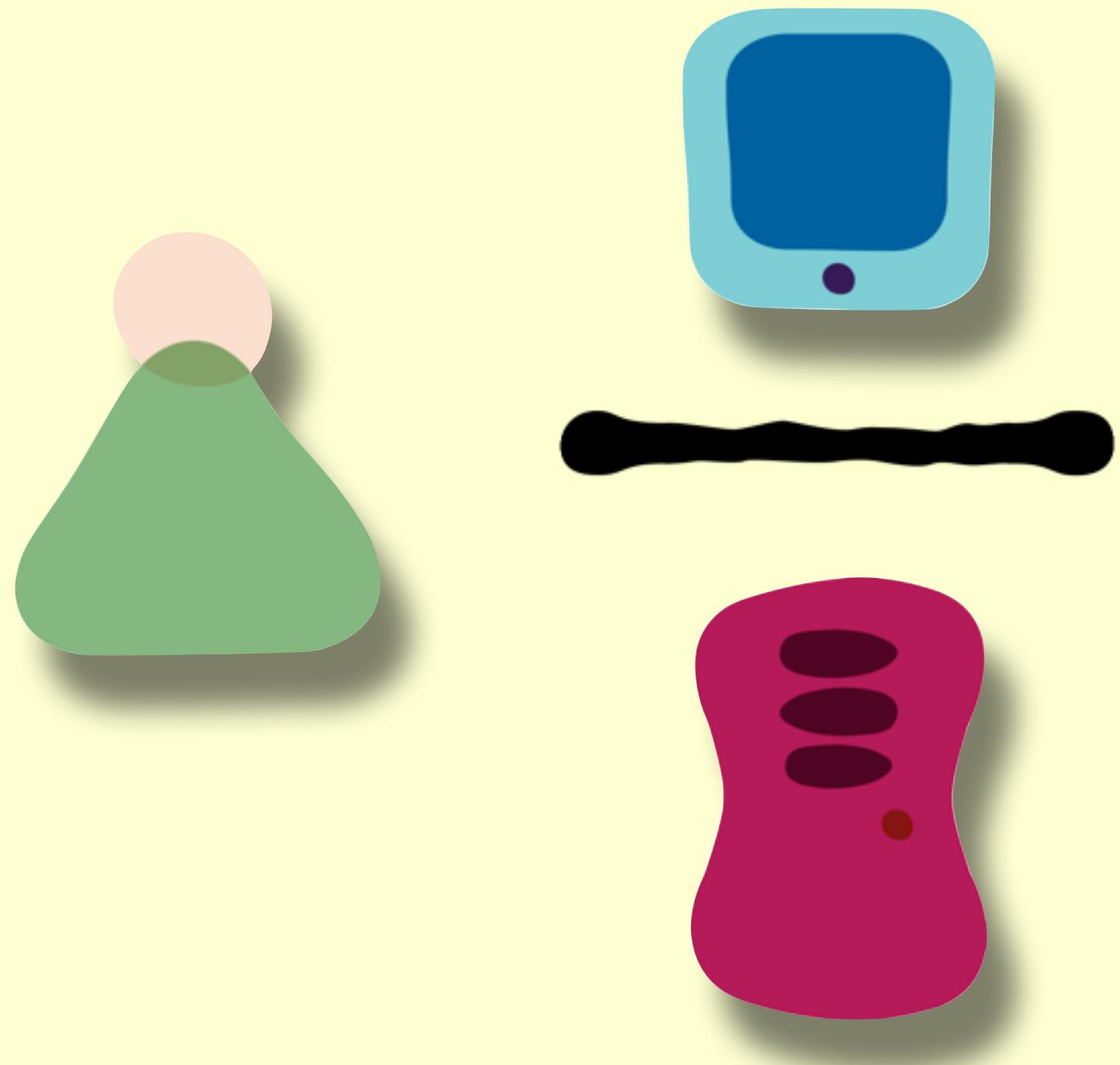
# Governing Code Quality



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

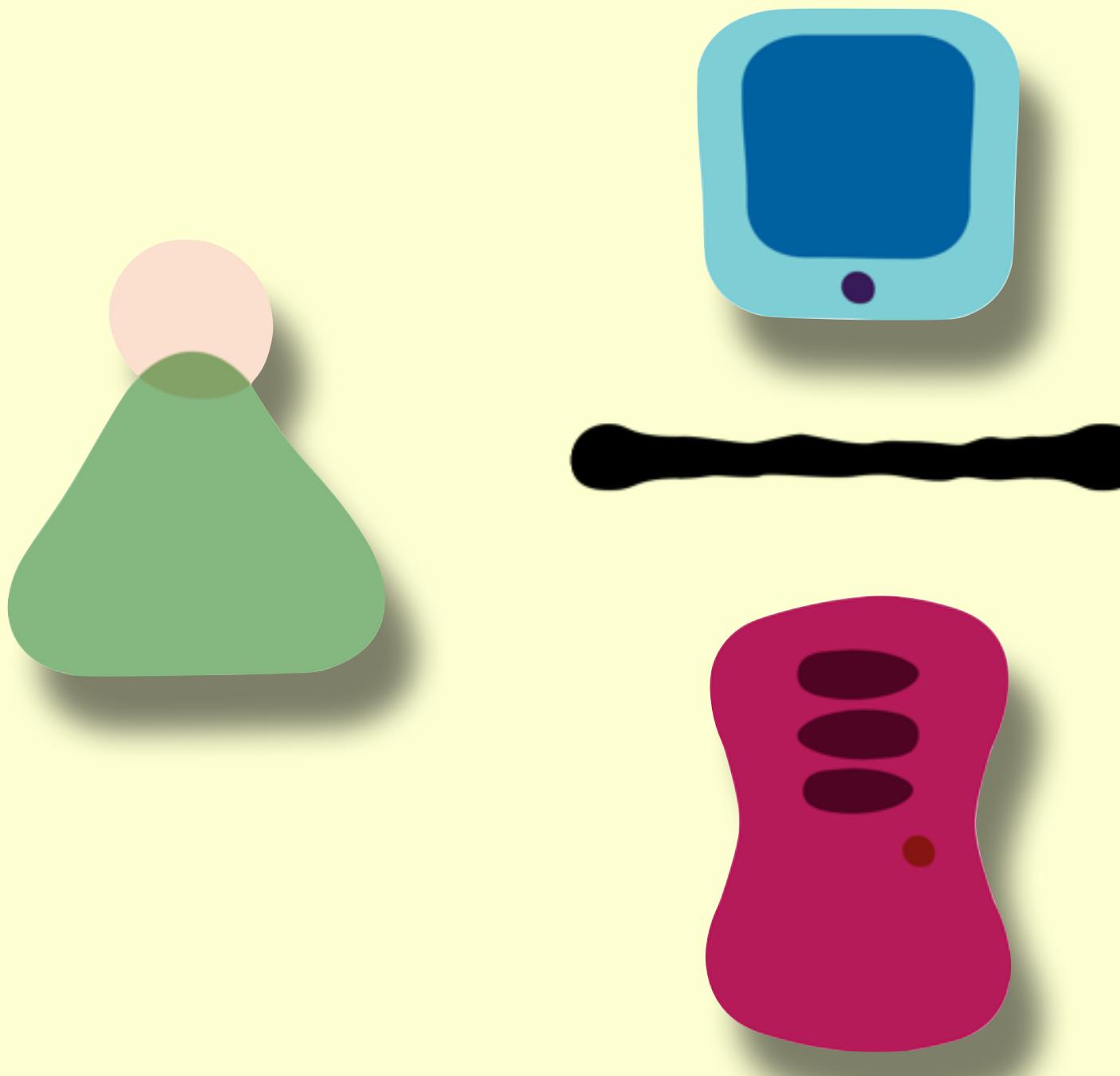


# Governing Code Quality

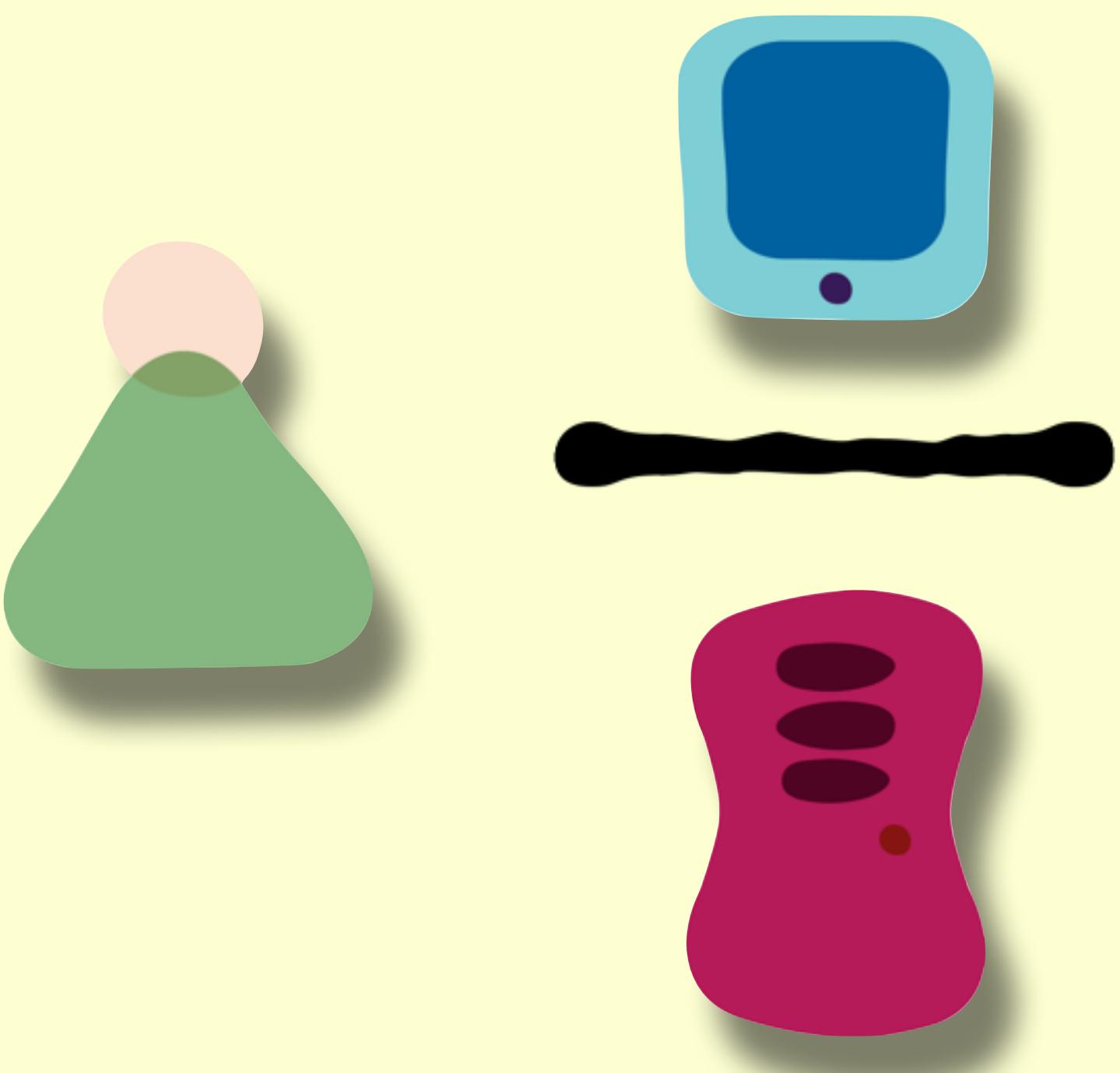


# Governing Code Quality

```
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();  
else if state == "D" then  
    doSomethingD();  
else if state == "E" then  
    doSomethingE();  
else if state == "F" then  
    doSomethingF();  
else if state == "G" then  
    doSomethingG();  
else if state == "H" then  
    doSomethingH();  
else if state == "I" then  
    doSomethingI();  
else if state == "J" then  
    doSomethingJ();  
else if state == "K" then  
    doSomethingK();  
else if state == "L" then  
    doSomethingL();  
else if state == "M" then  
    doSomethingM();  
else if state == "N" then  
    doSomethingN();  
else if state == "O" then  
    doSomethingO();  
else if state == "P" then  
    doSomethingP();  
else if state == "Q" then  
    doSomethingQ();  
else if state == "R" then  
    doSomethingR();  
else if state == "S" then  
    doSomethingS();  
else if state == "T" then  
    doSomethingT();  
else if state == "U" then  
    doSomethingU();  
else if state == "V" then  
    doSomethingV();  
else if state == "W" then  
    doSomethingW();  
else if state == "X" then  
    doSomethingX();  
else if state == "Y" then  
    doSomethingY();  
else if state == "Z" then  
    doSomethingZ();  
end if
```

# 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 contains a brief description of ArchUnit's purpose and how it works, along with links to examples and GitHub. At the bottom, there is a "News" section.

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/>

```
@AnalyzeClasses(packages = "com.tngtech.archunit.example")
public class CodingRulesTest {

    @ArchTest
    private final ArchRule no_access_to_standard_streams = NO_CLASSES_SHOULD_ACCESS_STANDARD_STREAMS;

    @ArchTest
    private void no_access_to_standard_streams_as_method(JavaClasses classes) {
        noClasses().should(ACCESS_STANDARD_STREAMS).check(classes);
    }

    @ArchTest
    private final ArchRule no_generic_exceptions = NO_CLASSES_SHOULD_THROW_GENERIC_EXCEPTIONS;

    @ArchTest
    private final ArchRule no_java_util_logging = NO_CLASSES_SHOULD_USE_JAVA_UTIL_LOGGING;

    @ArchTest
    private final ArchRule no_jodatime = NO_CLASSES_SHOULD_USE_JODATIME;

    @ArchTest
    static final ArchRule no_classes_should_access_standard_streams_or_throw_generic_exceptions =
        CompositeArchRule.of(NO_CLASSES_SHOULD_ACCESS_STANDARD_STREAMS)
            .and(NO_CLASSES_SHOULD_THROW_GENERIC_EXCEPTIONS);

}
```

coding rules

# ArchUnit

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

```
@AnalyzeClasses(packages = "com.tngtech.archunit.example")
public class ControllerRulesTest {

    @ArchTest
    static final ArchRule controllers_should_only_call_secured_methods =
        classes().that().resideInAPackage("..controller..")
            .should().onlyCallMethodsThat(areDeclaredInController().or(areannotatedWith(Secured.class)));

    @ArchTest
    static final ArchRule controllers_should_only_call_secured_constructors =
        classes()
            .that().resideInAPackage("..controller..")
            .should().onlyCallConstructorsThat(areDeclaredInController().or(areannotatedWith(Secured.class)));

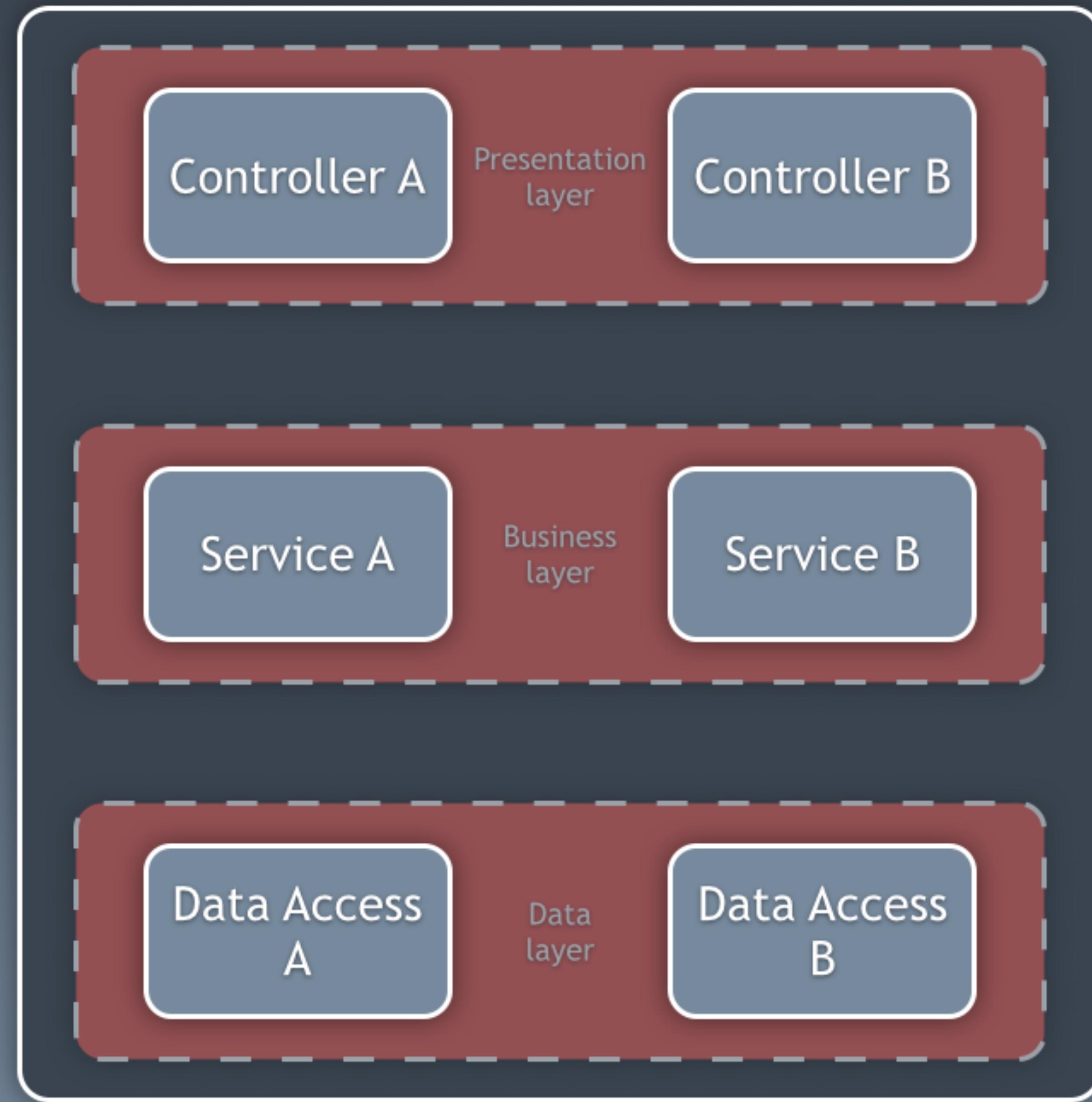
    @ArchTest
    static final ArchRule controllers_should_only_call_secured_code_units =
        classes()
            .that().resideInAPackage("..controller..")
            .should().onlyCallCodeUnitsThat(areDeclaredInController().or(areannotatedWith(Secured.class)));

    @ArchTest
    static final ArchRule controllers_should_only_access_secured_fields =
        classes()
            .that().resideInAPackage("..controller..")
            .should().onlyAccessFieldsThat(areDeclaredInController().or(areannotatedWith(Secured.class)));

    @ArchTest
    static final ArchRule controllers_should_only_access_secured_members =
        classes()
            .that().resideInAPackage("..controller..")
            .should().onlyAccessMembersThat(areDeclaredInController().or(areannotatedWith(Secured.class)));

    private static DescribedPredicate<JavaMember> areDeclaredInController() {
        DescribedPredicate<JavaClass> aPackageController = GET_PACKAGE_NAME.is(PackageMatchers.of("..controller..", "java.."))
            .as("a package '..controller..'");
        return areDeclaredIn(aPackageController);
    }
}
```

coding rules



Package by layer (horizontal slicing)

# ArchUnit

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

```
@AnalyzeClasses(packages = "com.tngtech.archunit.example")
public class LayeredArchitectureTest {
    @ArchTest
    static final ArchRule layer_dependencies_are_respected = layeredArchitecture()

        .layer("Controllers").definedBy("com.tngtech.archunit.example.controller..")
        .layer("Services").definedBy("com.tngtech.archunit.example.service..")
        .layer("Persistence").definedBy("com.tngtech.archunit.example.persistence..")

        .whereLayer("Controllers").mayNotBeAccessedByAnyLayer()
        .whereLayer("Services").mayOnlyBeAccessedByLayers("Controllers")
        .whereLayer("Persistence").mayOnlyBeAccessedByLayers("Services");

    @ArchTest
    static final ArchRule layer_dependencies_are_respected_with_exception = layeredArchitecture()

        .layer("Controllers").definedBy("com.tngtech.archunit.example.controller..")
        .layer("Services").definedBy("com.tngtech.archunit.example.service..")
        .layer("Persistence").definedBy("com.tngtech.archunit.example.persistence..")

        .whereLayer("Controllers").mayNotBeAccessedByAnyLayer()
        .whereLayer("Services").mayOnlyBeAccessedByLayers("Controllers")
        .whereLayer("Persistence").mayOnlyBeAccessedByLayers("Services")

        .ignoreDependency(SomeMediator.class, ServiceViolatingLayerRules.class);
}
```

layer dependency

# ArchUnit

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

```
@ArchTest
static final ArchRule services_should_not_depend_on_controllers =
    noClasses().that().resideInAPackage("..service..")
        .should().dependOnClassesThat().resideInAPackage("..controller..");
```

```
@ArchTest
static final ArchRule persistence_should_not_depend_on_services =
    noClasses().that().resideInAPackage("..persistence..")
        .should().dependOnClassesThat().resideInAPackage("..service..");
```

```
@ArchTest
static final ArchRule services_should_only_be depended_on_by_controllers_or_other_services =
    classes().that().resideInAPackage("..service..")
        .should().onlyHaveDependentClassesThat().resideInAnyPackage("..controller..", "..service..");
```

```
@ArchTest
static final ArchRule services_should_only_depend_on_persistence_or_other_services =
    classes().that().resideInAPackage("..service..")
        .should().onlyDependOnClassesThat().resideInAnyPackage("..service..", "..persistence..", "java..");
```

## layer dependency

```
@ArchTest
static final ArchRule services_should_not_access_controllers =
    noClasses().that().resideInAPackage("..service..")
        .should().accessClassesThat().resideInAPackage("..controller..");

@ArchTest
static final ArchRule persistence_should_not_access_services =
    noClasses().that().resideInAPackage("..persistence..")
        .should().accessClassesThat().resideInAPackage("..service..");

@ArchTest
static final ArchRule services_should_only_be accessed_by_controllers_or_other_services =
    classes().that().resideInAPackage("..service..")
        .should().onlyBeAccessed().byAnyPackage("..controller..", "..service..");

@ArchTest
static final ArchRule services_should_only_access_persistence_or_other_services =
    classes().that().resideInAPackage("..service..")
        .should().onlyAccessClassesThat().resideInAnyPackage("..service..", "..persistence..", "java..");

@ArchTest
static final ArchRule services_should_only_be depended_on_by_controllers_or_other_services =
    classes().that().resideInAPackage("..service..")
        .should().onlyHaveDependentClassesThat().resideInAnyPackage("..controller..", "..service..");

@ArchTest
static final ArchRule services_should_only_depend_on_persistence_or_other_services =
    classes().that().resideInAPackage("..service..")
        .should().onlyDependOnClassesThat().resideInAnyPackage("..service..", "..persistence..", "java..");
```

# NetArchTest

<https://github.com/BenMorris/NetArchTest/>

```
// Controllers should not directly reference repositories
var result = Types.InCurrentDomain()
    .That()
    .ResideInNamespace("NetArchTest.SampleLibrary.Presentation")
    .ShouldNot()
    .HaveDependencyOn("NetArchTest.SampleLibrary.Data")
    .GetResult().IsSuccessful;
```

# NetArchTest

<https://github.com/BenMorris/NetArchTest/>

```
// Only classes in the data namespace can have a dependency on System.Data
result = Types.InCurrentDomain()
    .That().HaveDependencyOn("System.Data")
    .And().ResideInNamespace(("ArchTest"))
    .Should().ResideInNamespace(("NetArchTest.SampleLibrary.Data"))
    .GetResult().IsSuccessful;
```

# NetArchTest

<https://github.com/BenMorris/NetArchTest/>

```
// All the classes in the data namespace should implement IRepository  
result = Types.InCurrentDomain()  
    .That().ResideInNamespace("NetArchTest.SampleLibrary.Data")  
    .And().AreClasses()  
    .Should().ImplementInterface(typeof(IRepository))  
    .GetResult().IsSuccessful;  
  
// Classes that implement IRepository should have the suffix "Repository"  
result = Types.InCurrentDomain()  
    .That().ResideInNamespace("NetArchTest.SampleLibrary.Data")  
    .And().AreClasses()  
    .Should().HaveNameEndingWith("Repository")  
    .GetResult().IsSuccessful;  
  
// Classes that implement IRepository must reside in the Data namespace  
result = Types.InCurrentDomain()  
    .That().ImplementInterface(typeof(IRepository))  
    .Should().ResideInNamespace("NetArchTest.SampleLibrary.Data")  
    .GetResult().IsSuccessful;  
  
// All the service classes should be sealed  
result = Types.InCurrentDomain()  
    .That().ImplementInterface(typeof(IWidgetService))  
    .Should().BeSealed()  
    .GetResult().IsSuccessful;
```

# Legality of Open Source Libraries

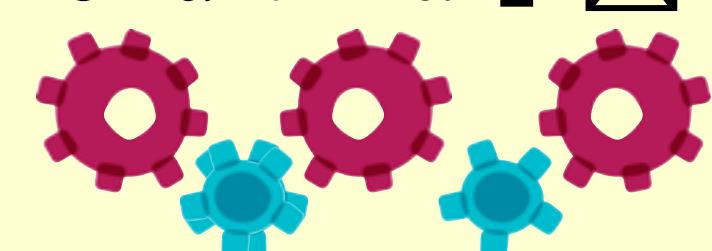


Penultima ↑⊗e  
↑⊗e

# Legality of Open Source Libraries



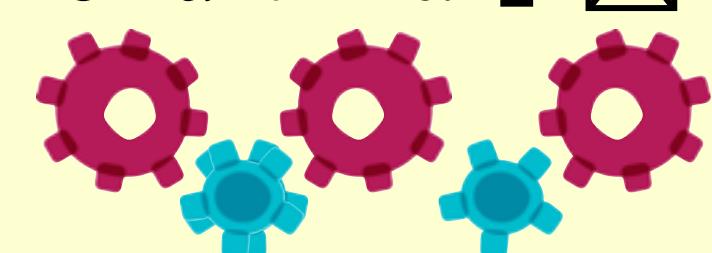
Penultima ↑⊗e



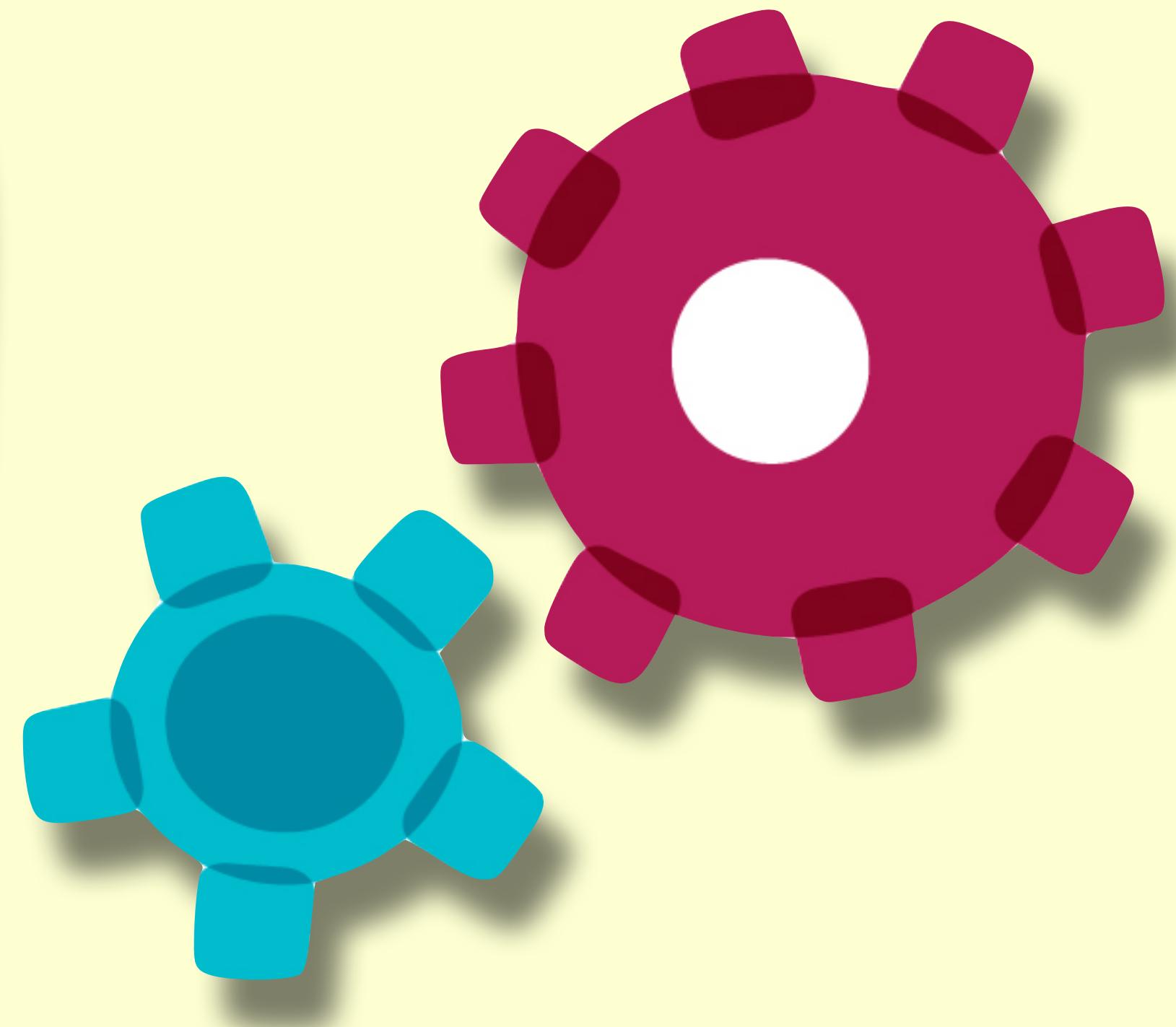
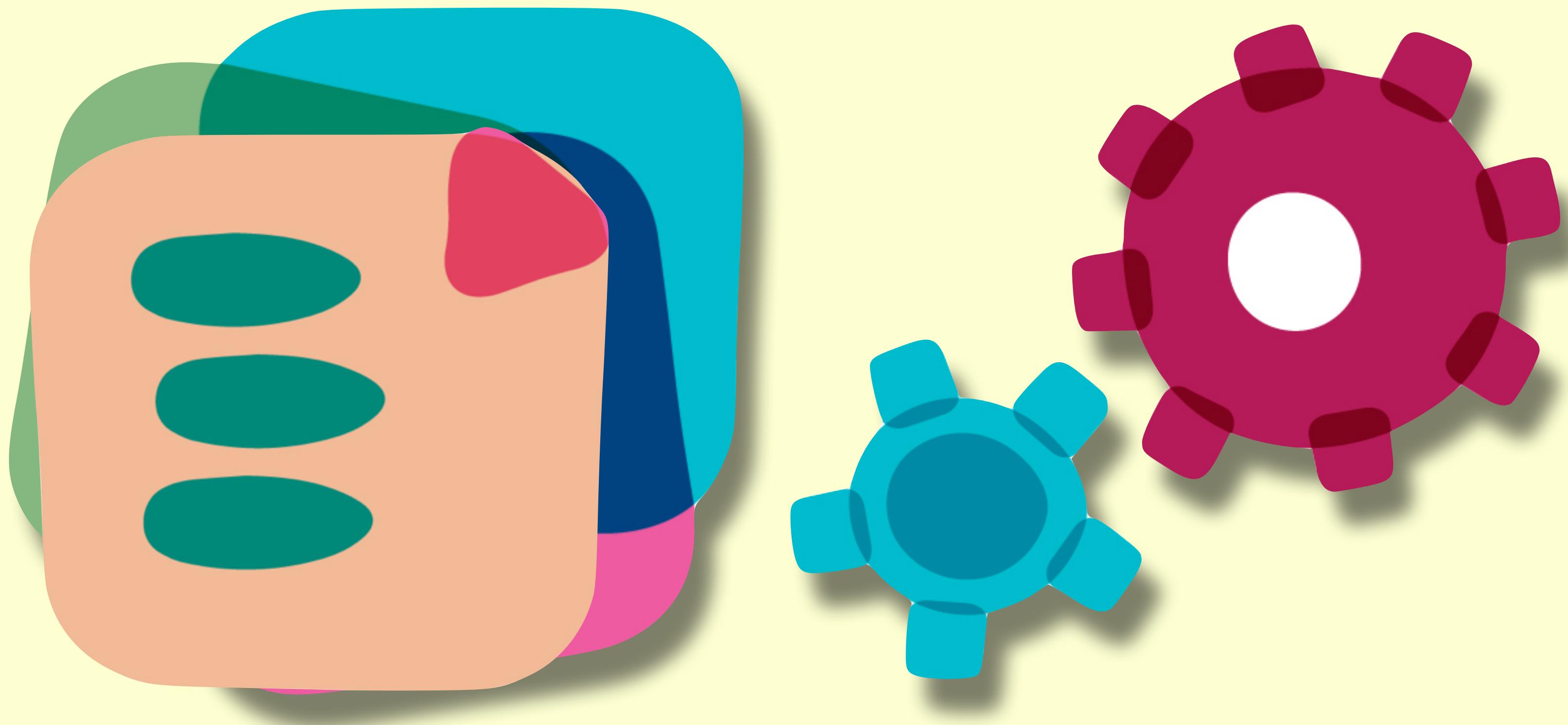
# Legality of Open Source Libraries



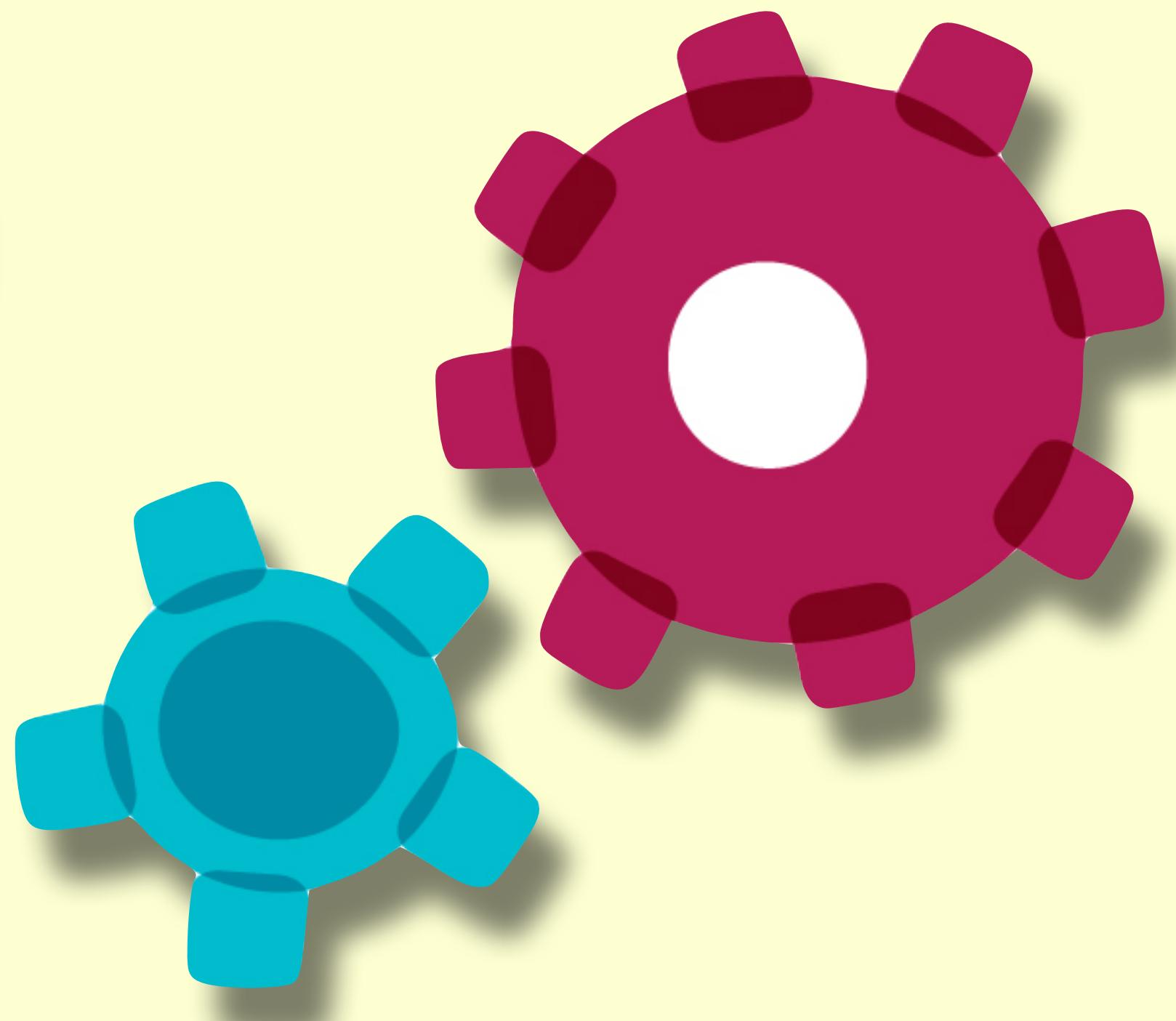
Penultima ↑⊗e



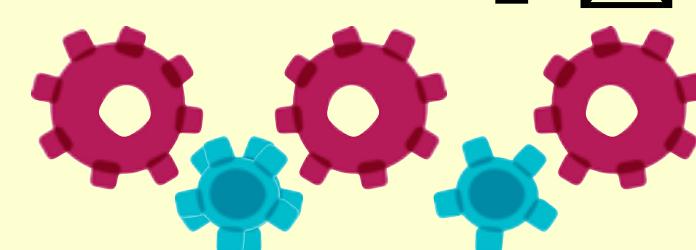
# Legality of Open Source Libraries



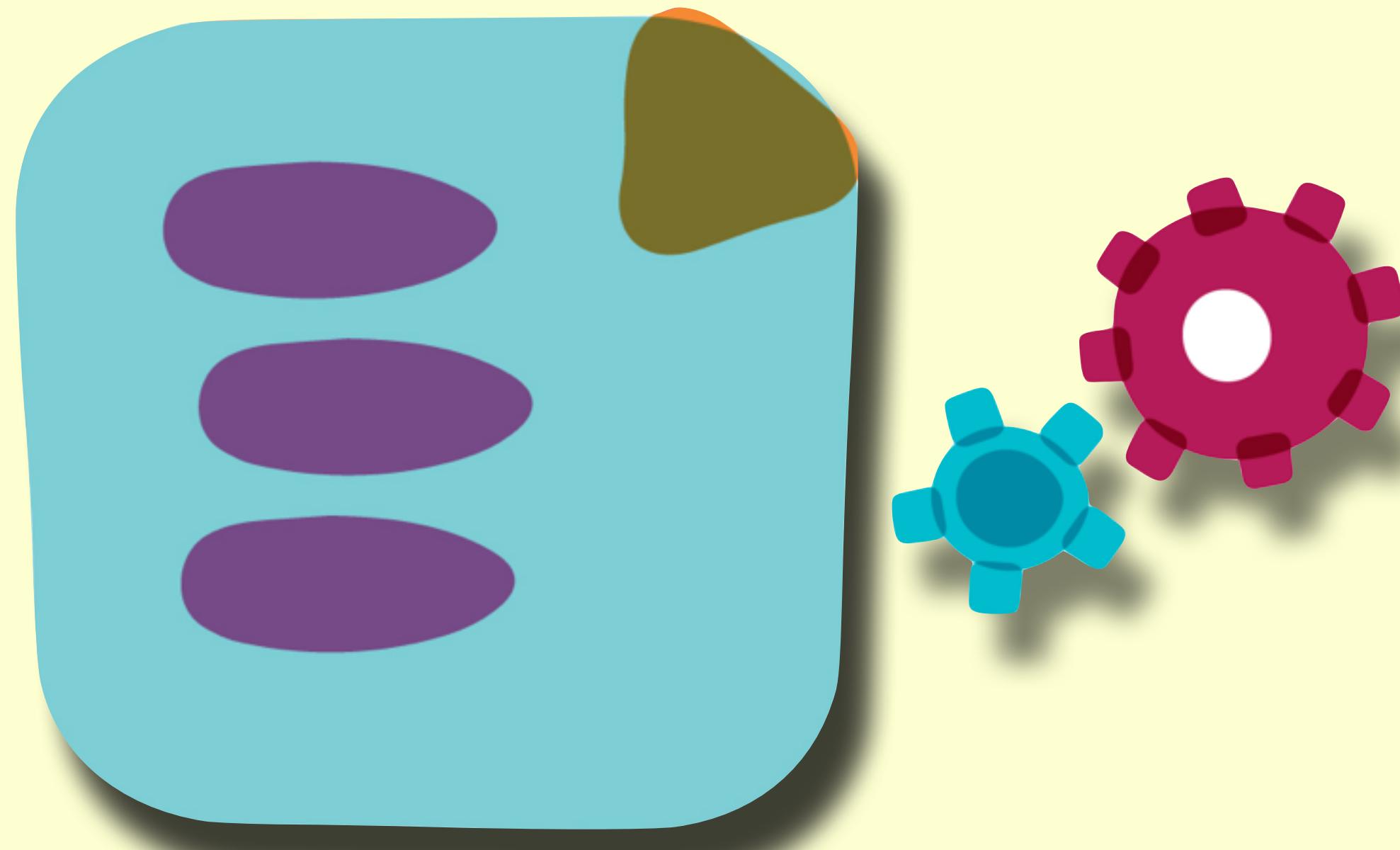
# Legality of Open Source Libraries

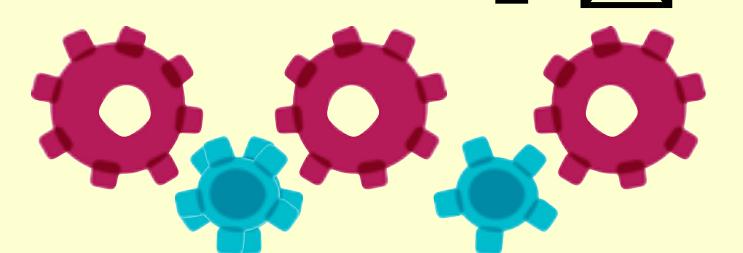


Penultima ↑⊗e

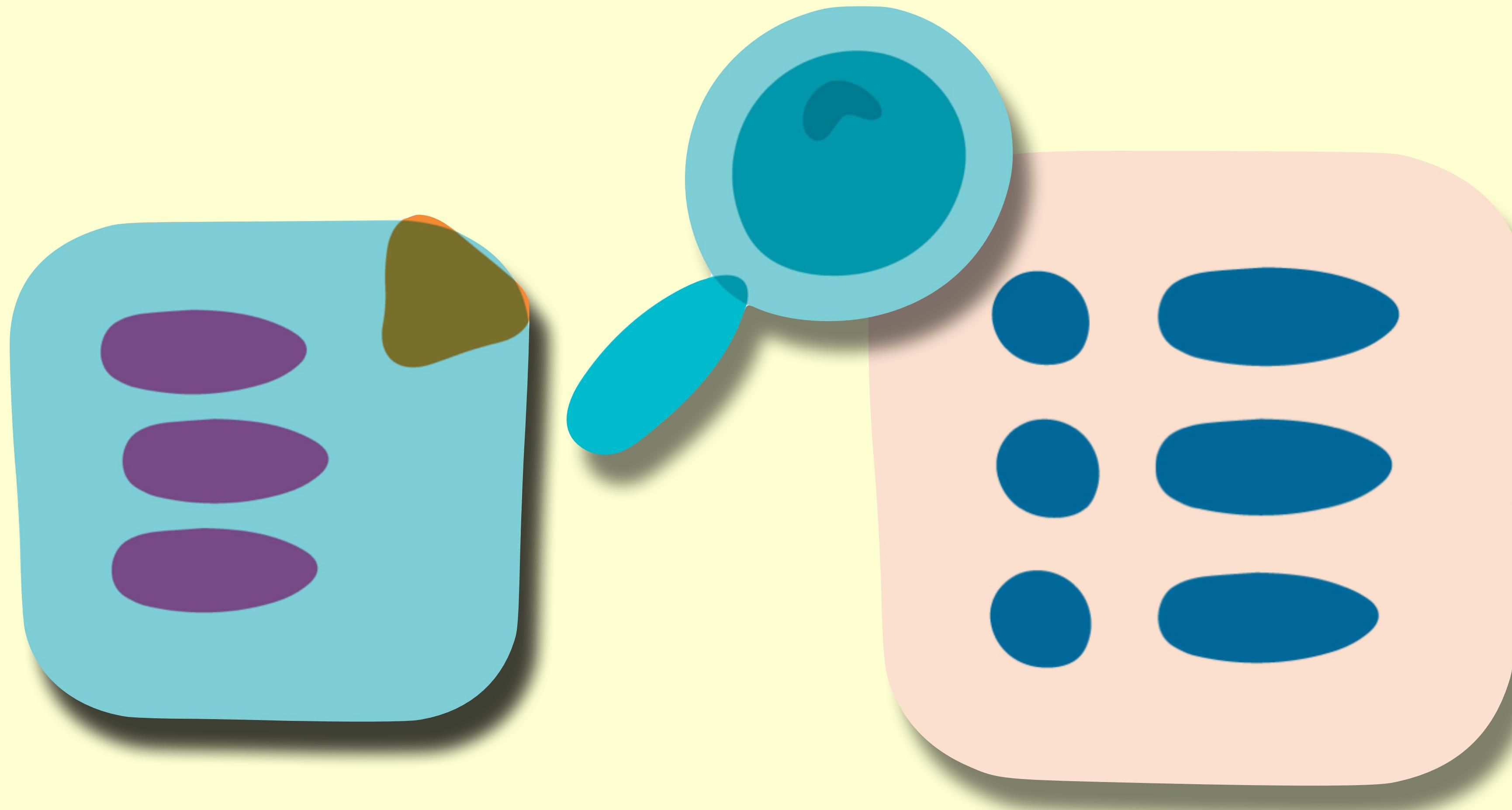


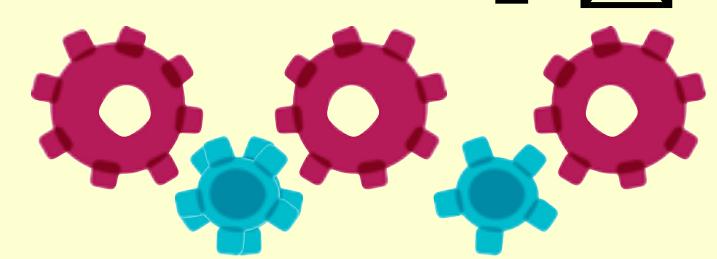
# Legality of Open Source Libraries



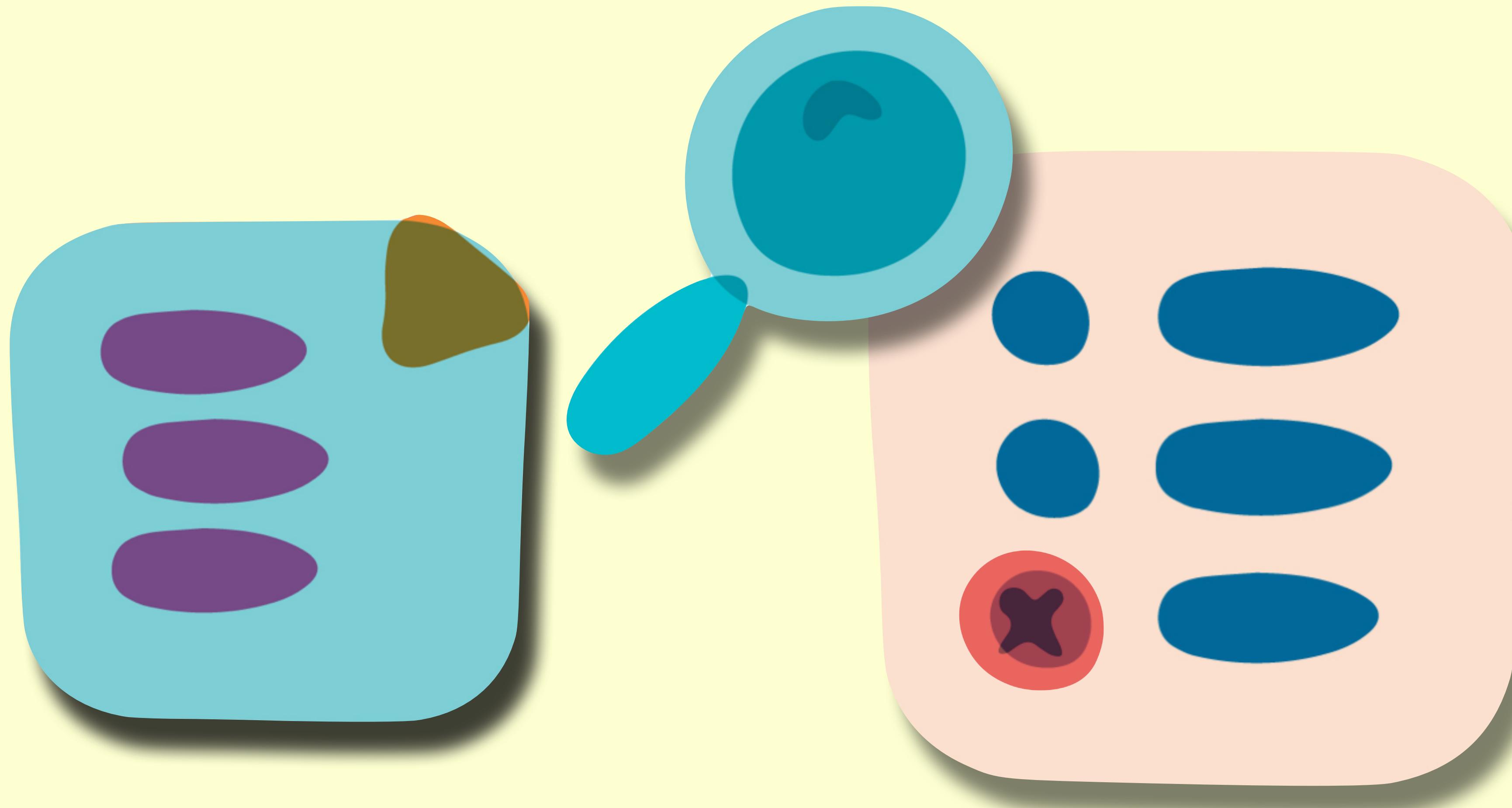
Penultima ↑⊗e  


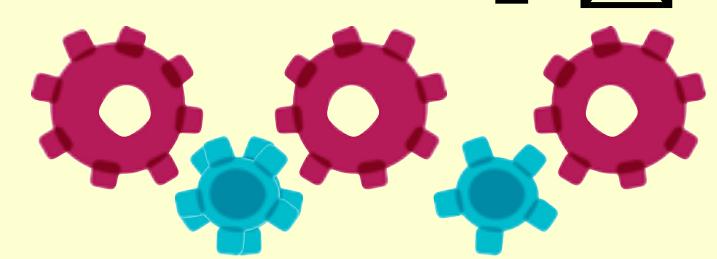
# Legality of Open Source Libraries



Penultima ↑⊗e  


# Legality of Open Source Libraries

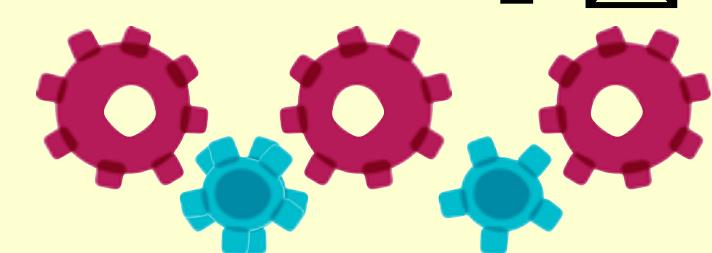


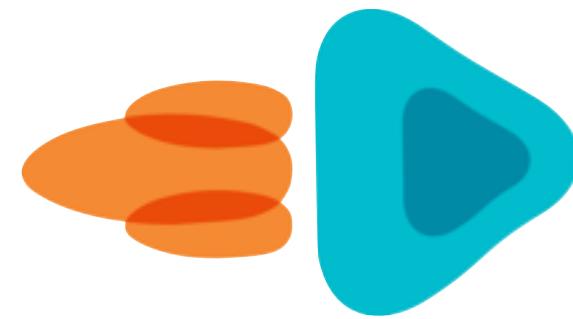
Penultima ↑⊗e  


# Legality of Open Source Libraries

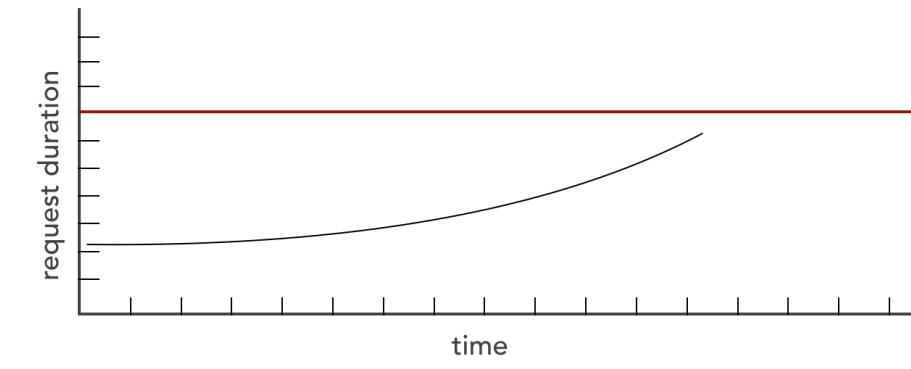
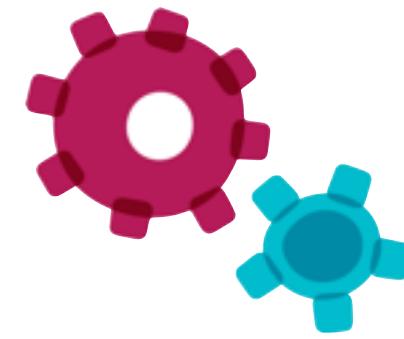


Penultima ↑⊗e

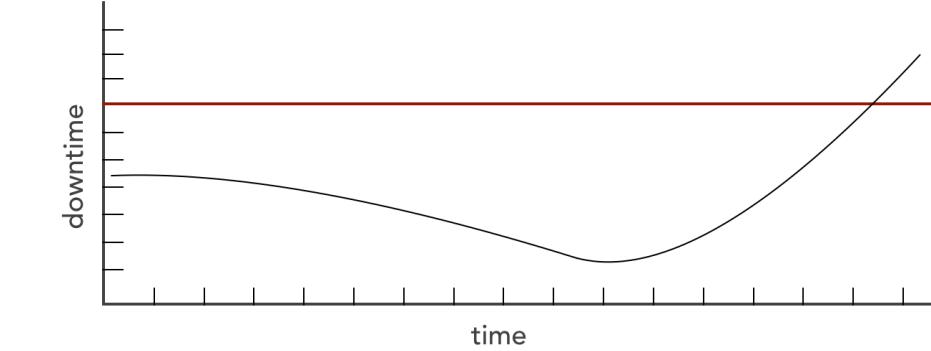
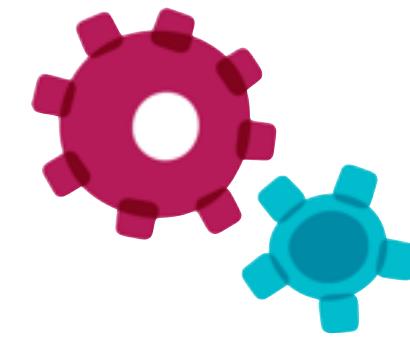




# Performance

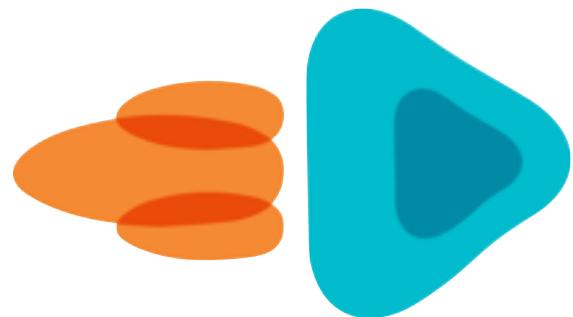


measure and track average response times  
(cumulative)



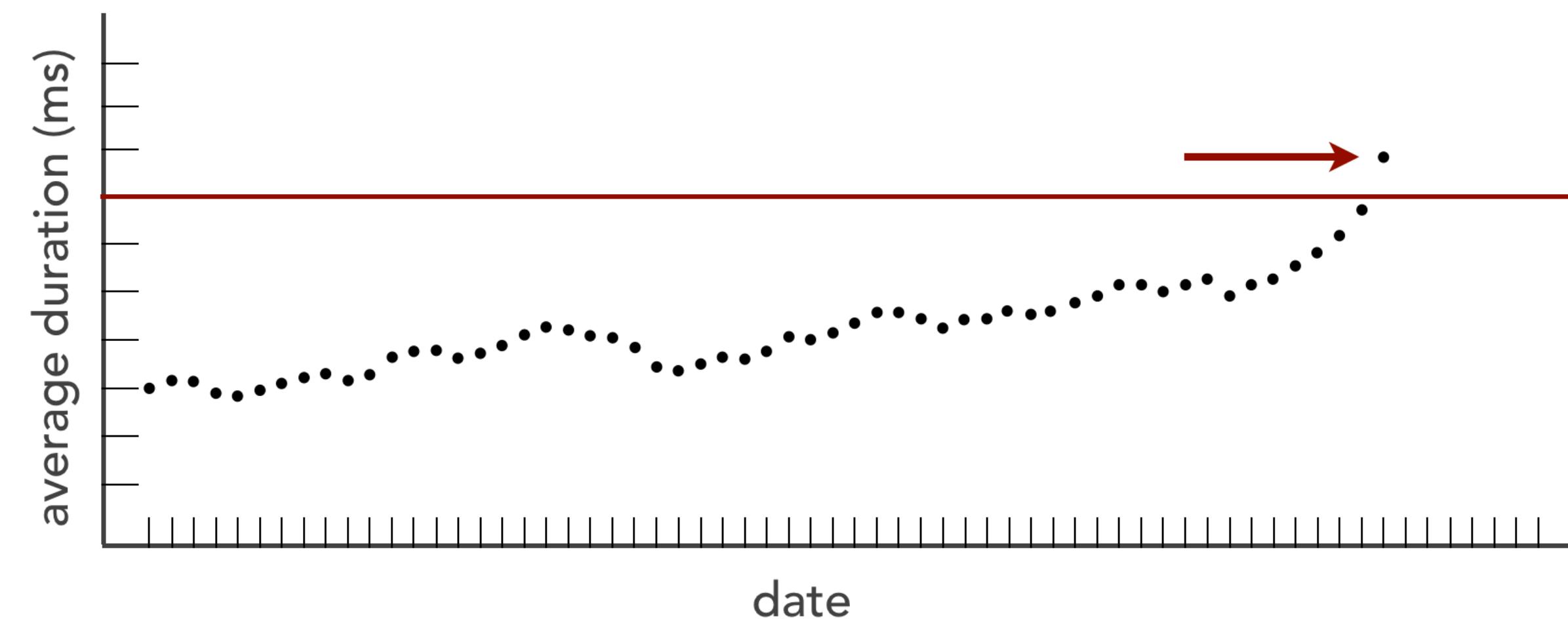
measure and track maximum response times

# Cumulative Average Response Times

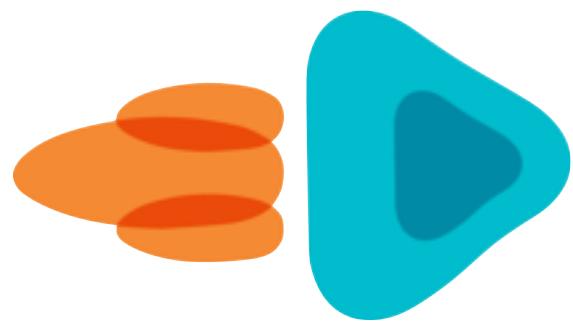


(per request, domain, or application context)

trigger alert when the average response times within a particular context exceeds 1400ms

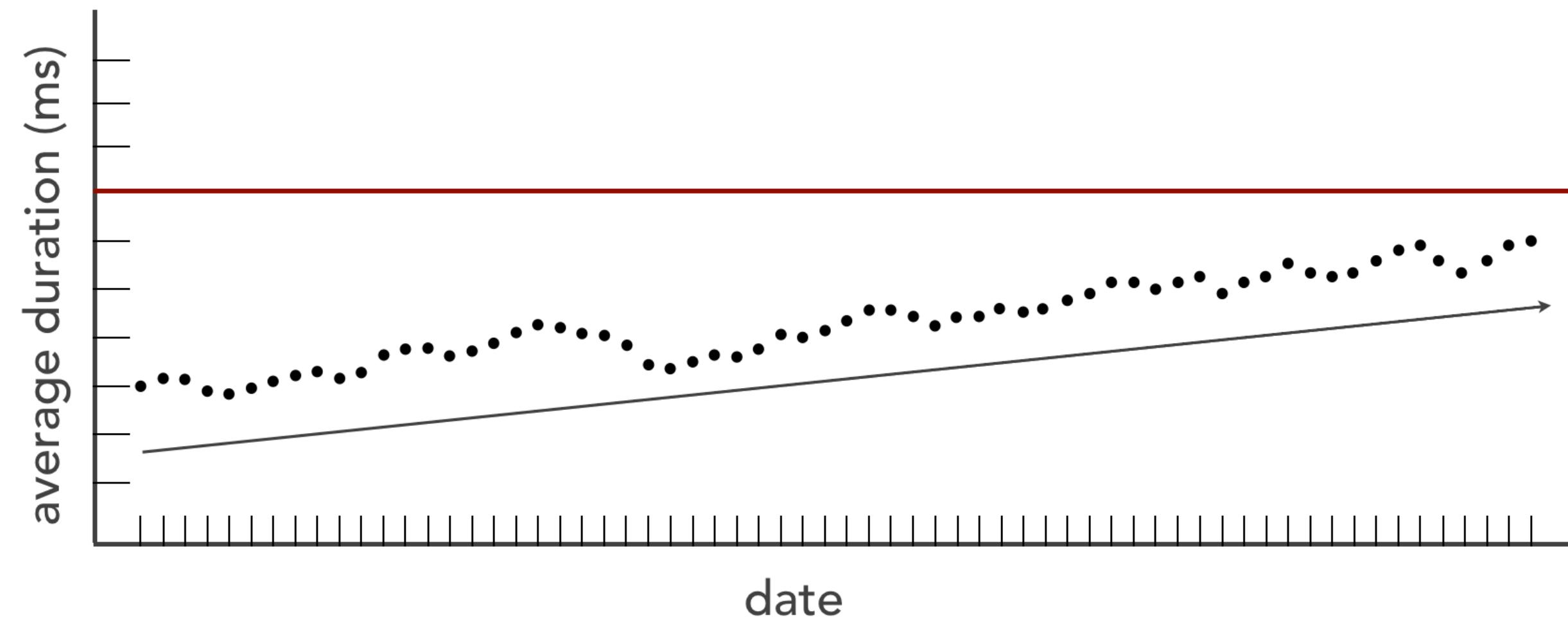


# Cumulative Average Response Times



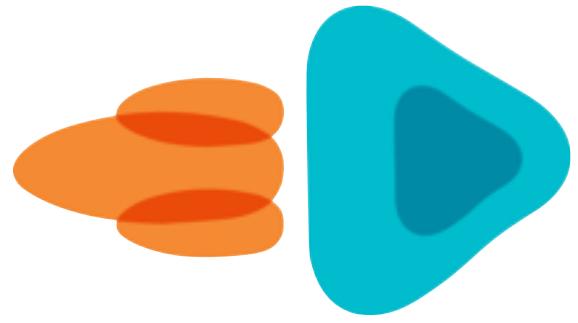
(per request, domain, or application context)

trigger alert when the average response times continues to increase over a 2 month period of time

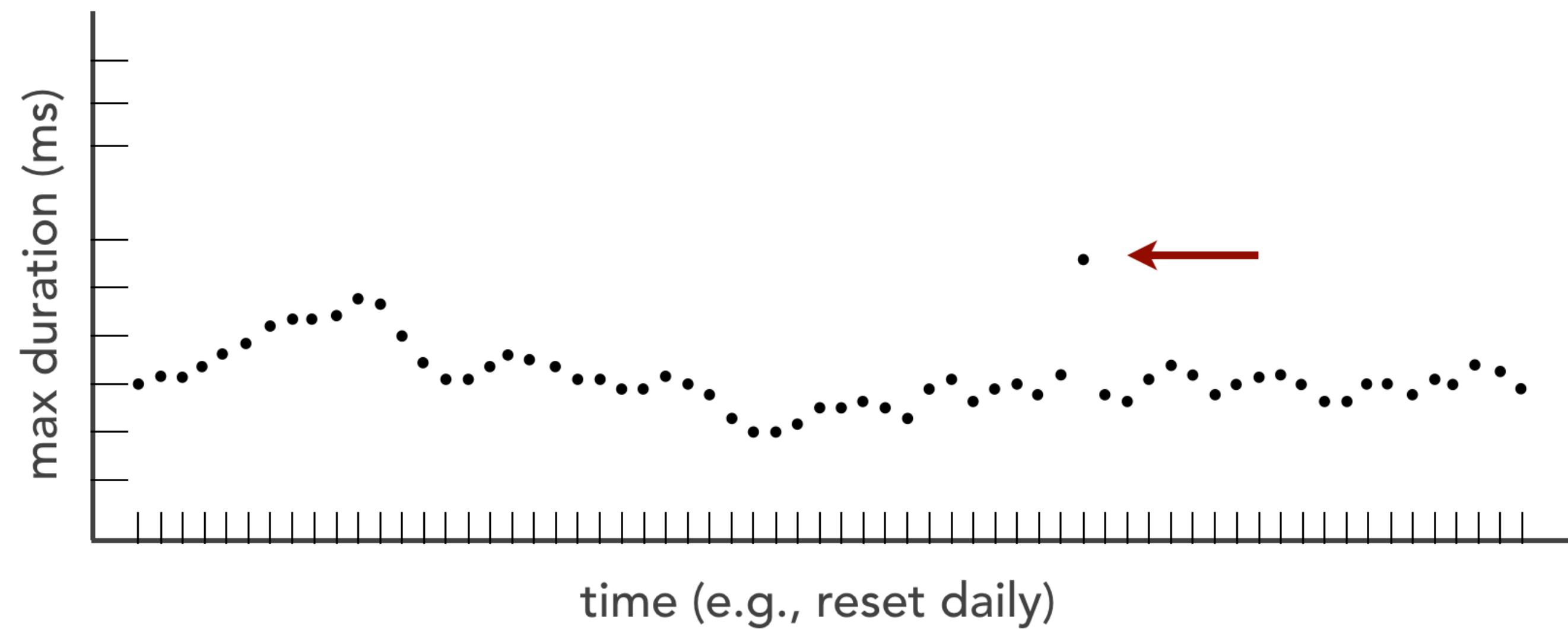


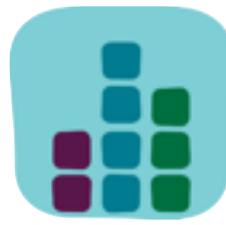
# Maximum Response Times

(per request, domain, or application context)

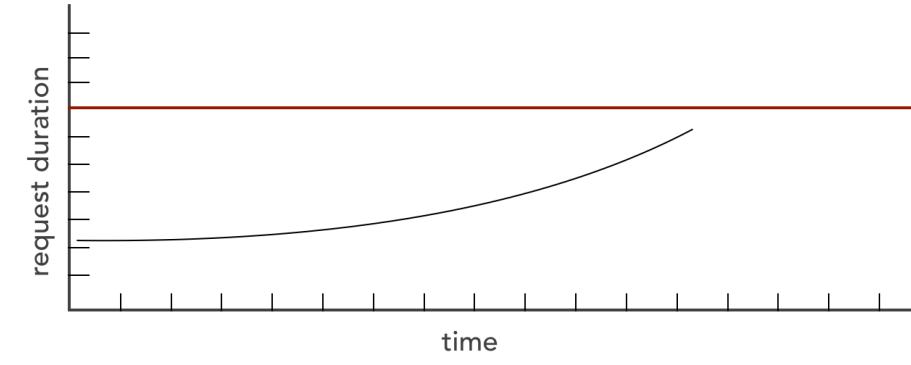


trigger alert anytime the maximum response time of selected requests exceeds 2000ms

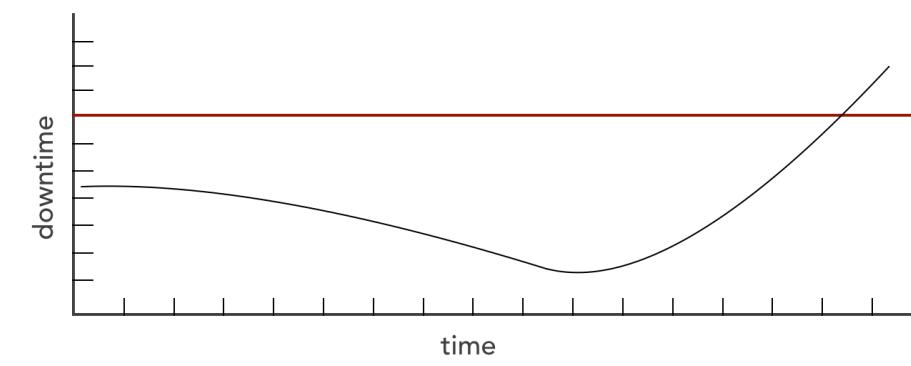




# Scalability



measure and track average response times across number of users or business requests



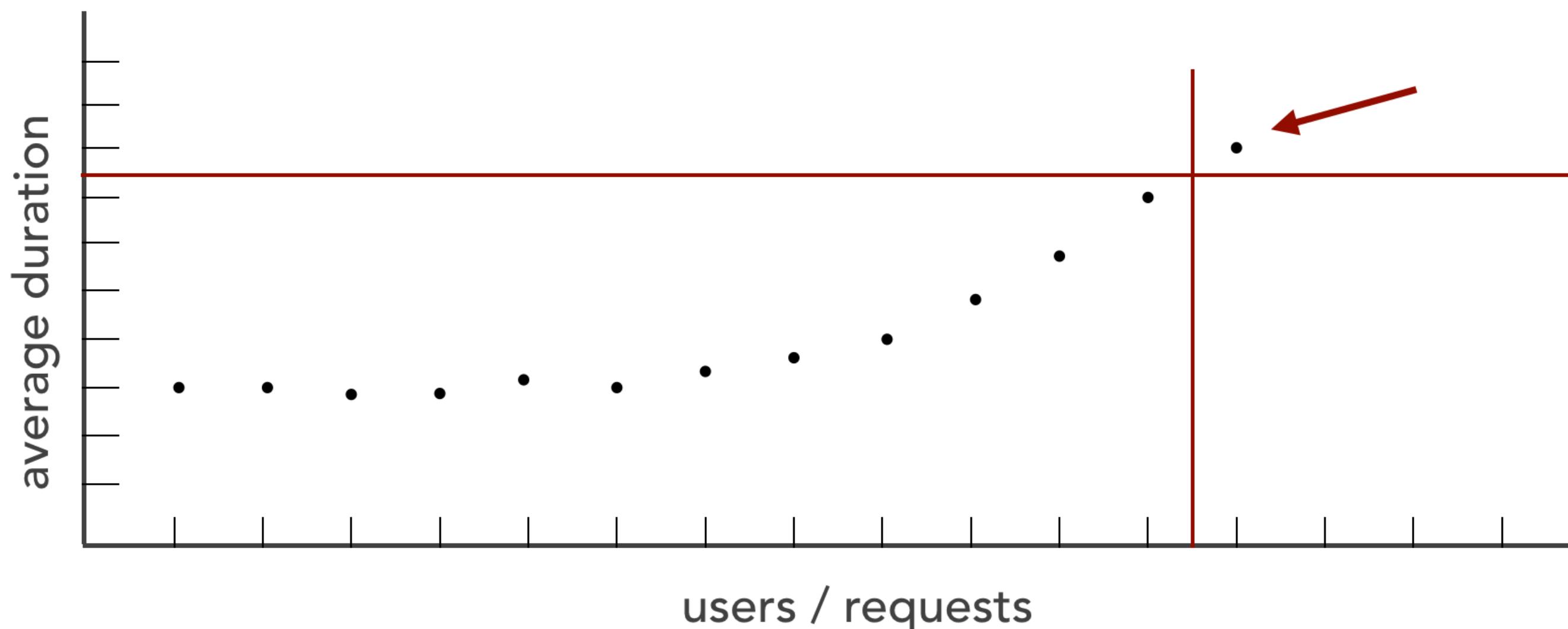
measure and track timeouts across number of users or business requests



# Average Response Times

(per request, domain, or application context)

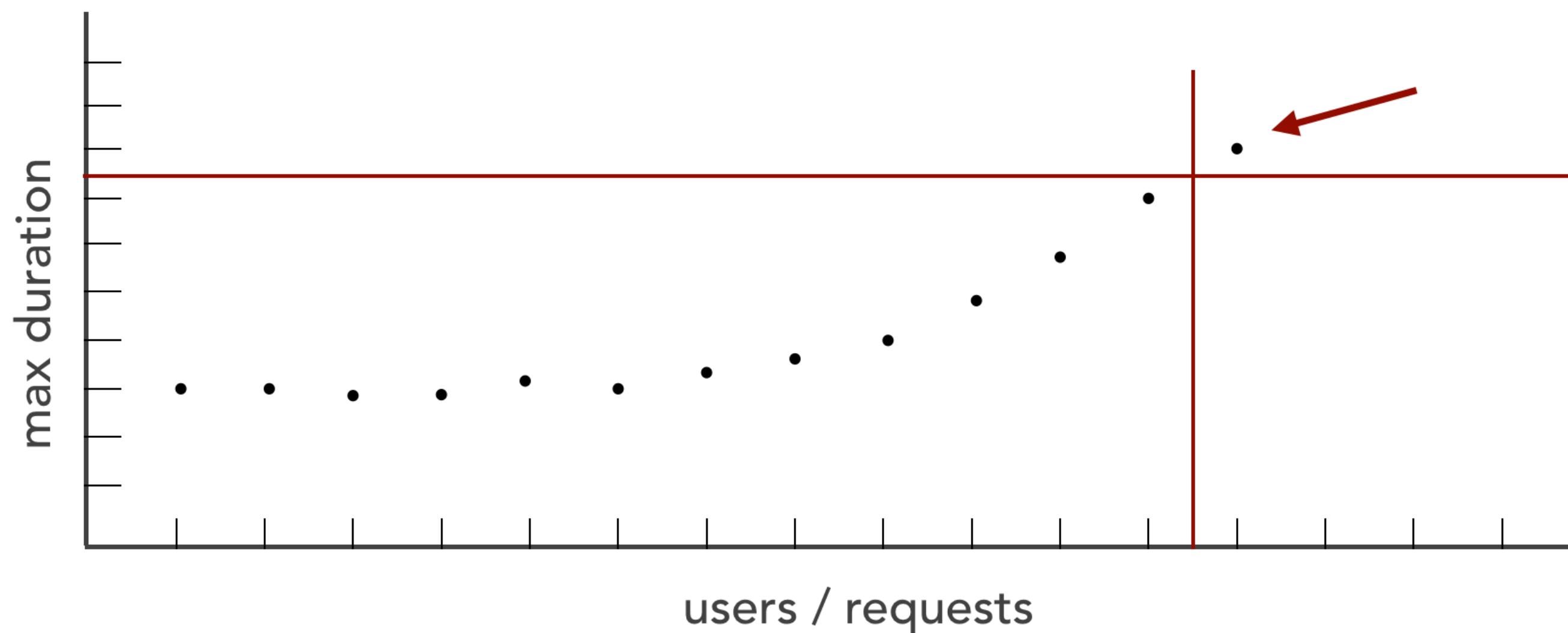
trigger alert if the average response time increases by 10% when the number of users increases over 7%

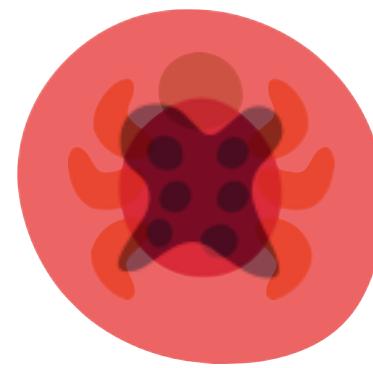


# Maximum Response Times

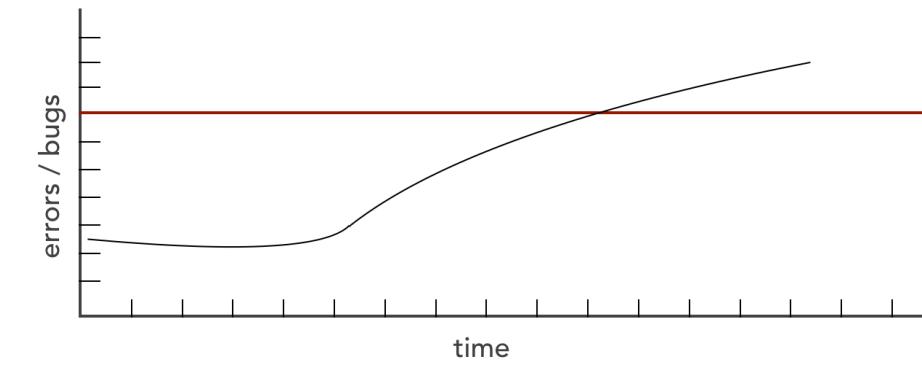
(per request, domain, or application context)

trigger alert if the maximum response time increases over 5% when the number of users increases over 10%

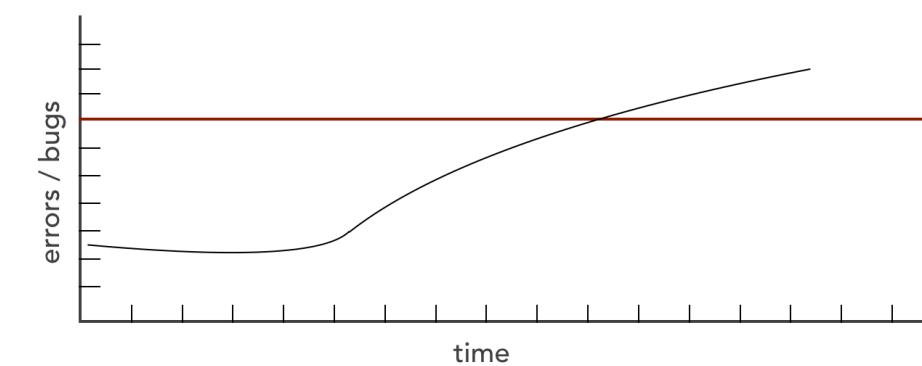




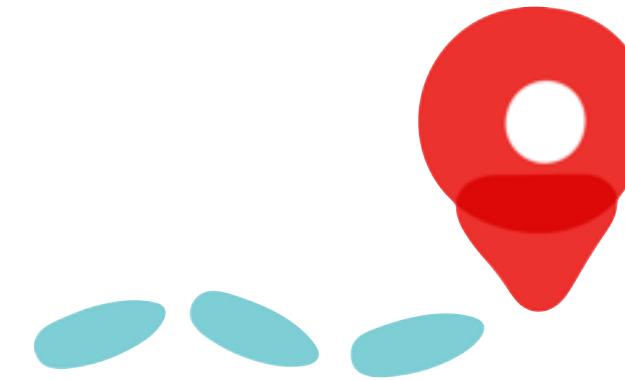
# Fault Tolerance



measure and track number of users impacted by a system or service crash

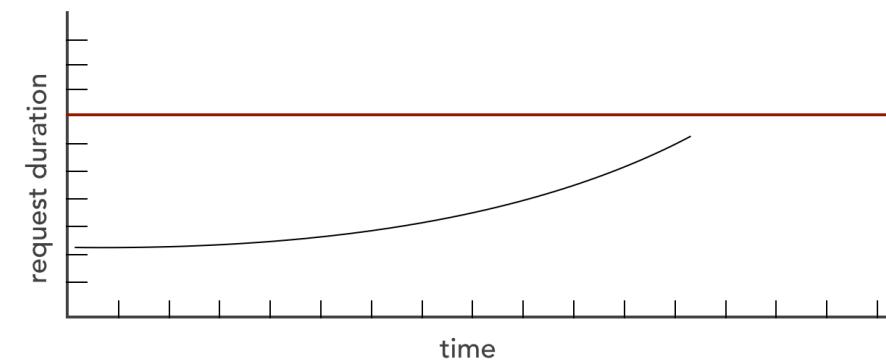


measure and track number of requests impacted by a system or service crash

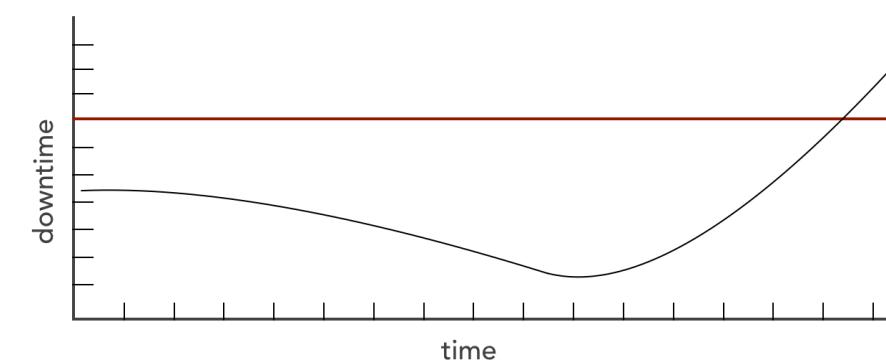


# Deployability

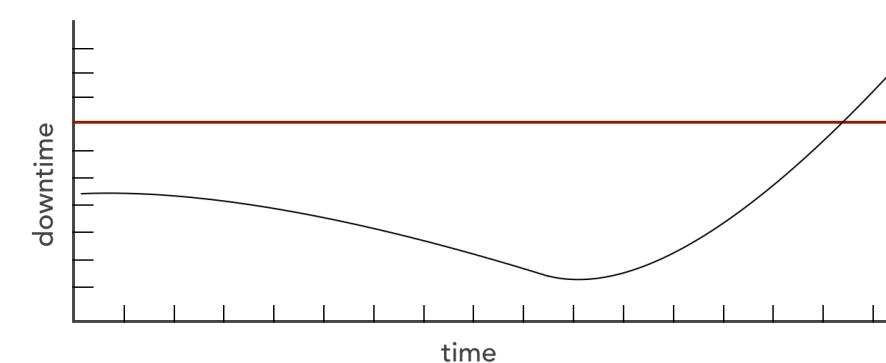
the ease of, risk, and frequency of deployment



measure and track number of actual hours spent to deploy



measure and track failed deployments or errors resulting from deployment

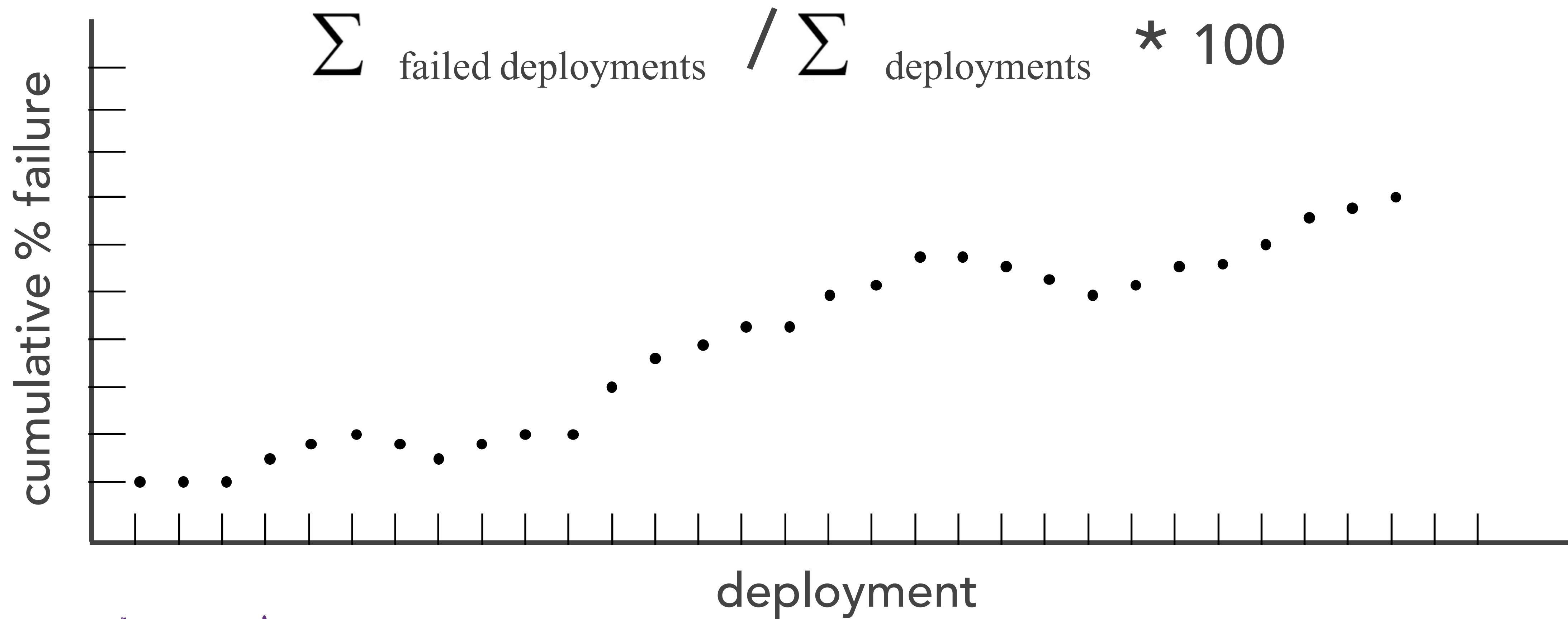


measure and track the frequency of deployment



# Deployment Errors/Failures

(per service, domain, or application context)



triggered



# 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

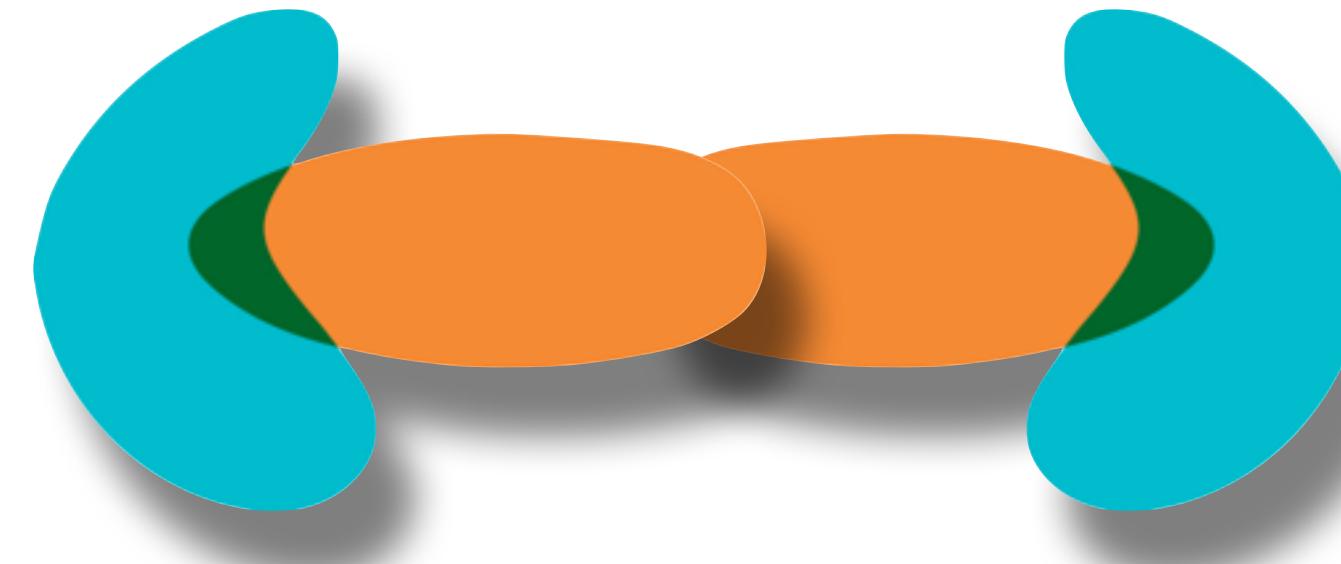


states!

# Fitness Function Guidelines

developers must collaborate

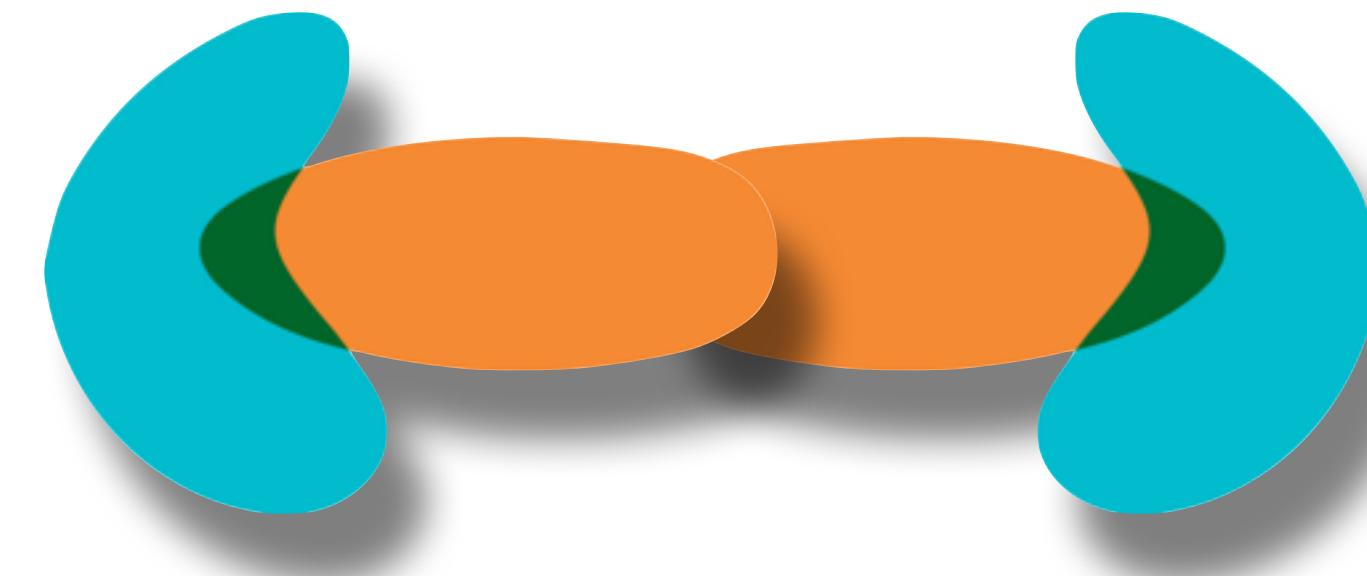
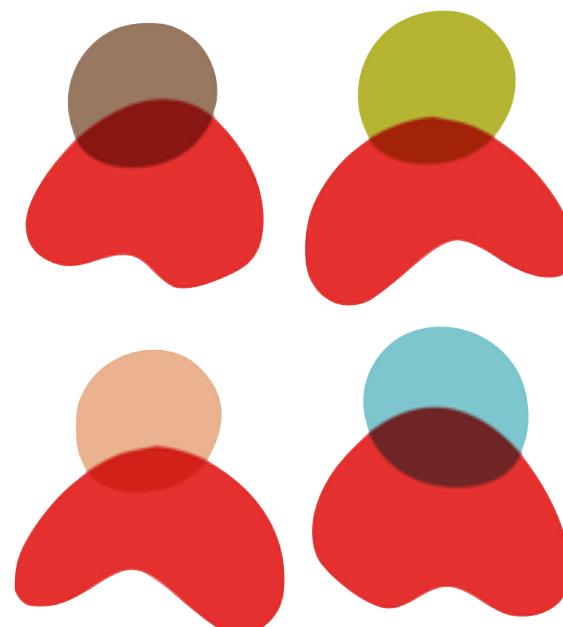
enterprise  
architects



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

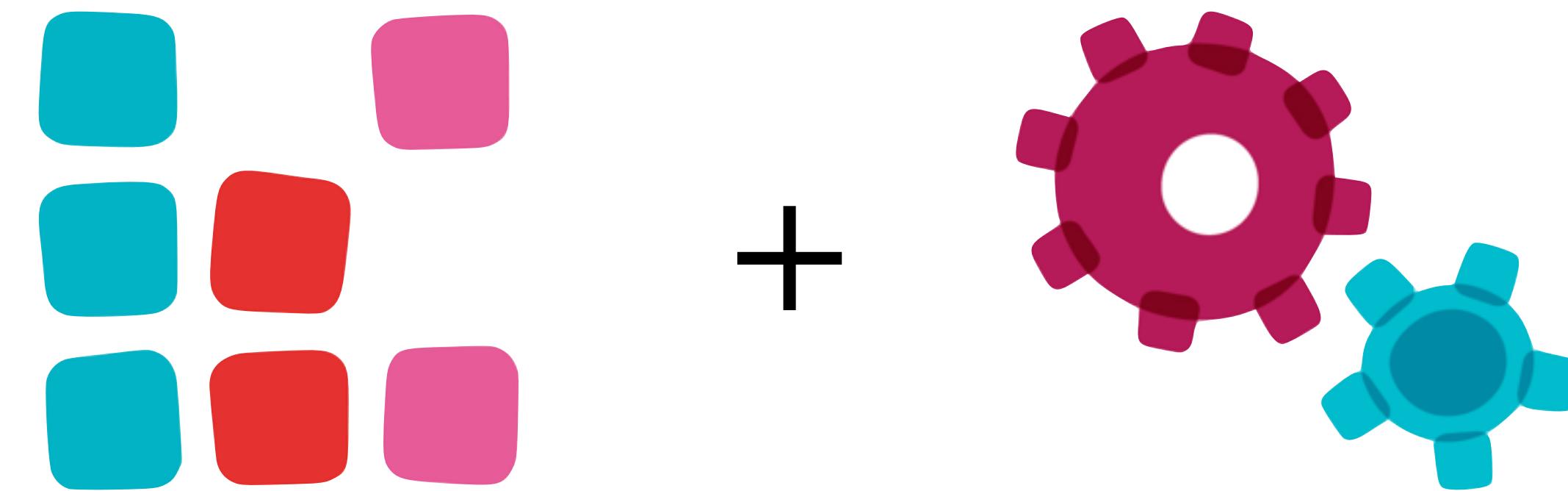
automation =



when needed

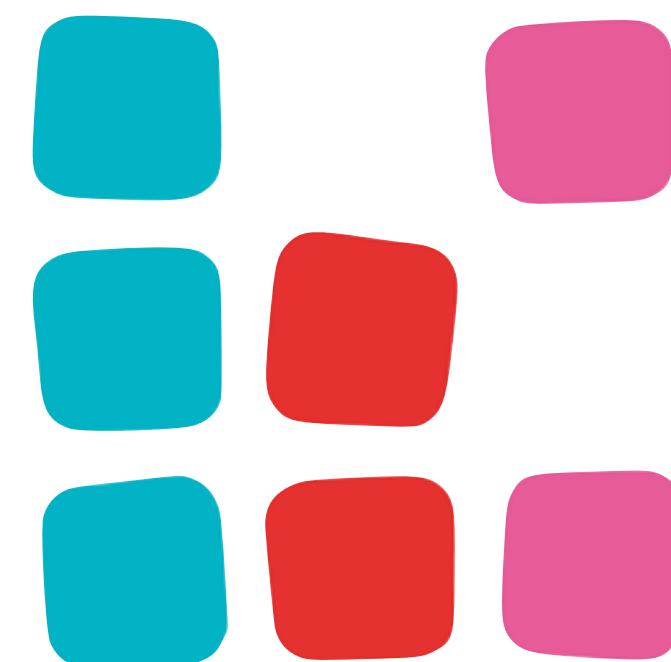
# Architecture Defined

architecture =

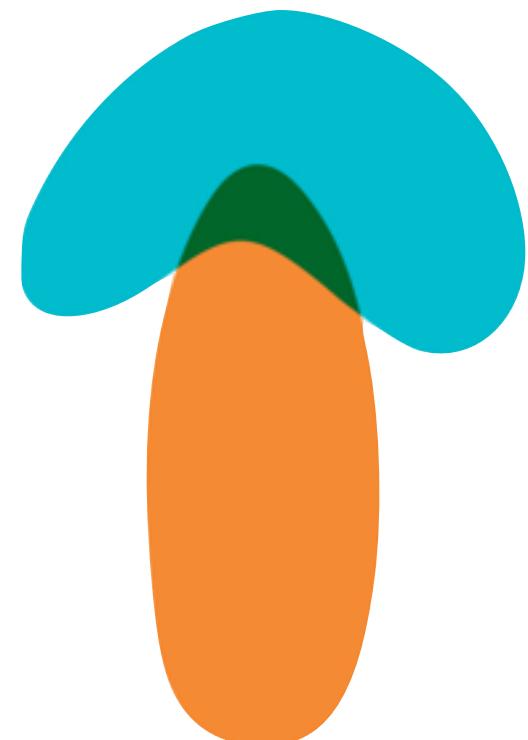
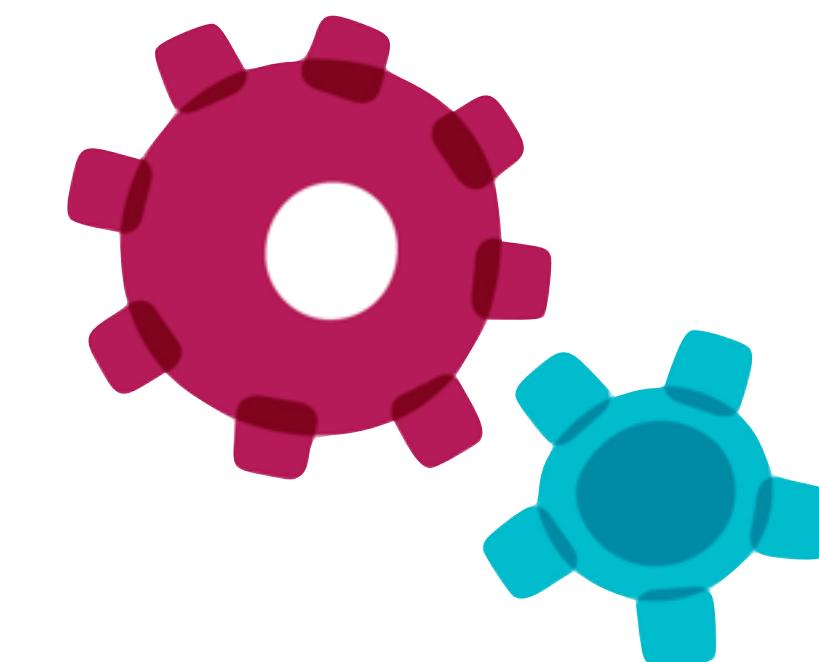


# Architecture Defined

architecture =



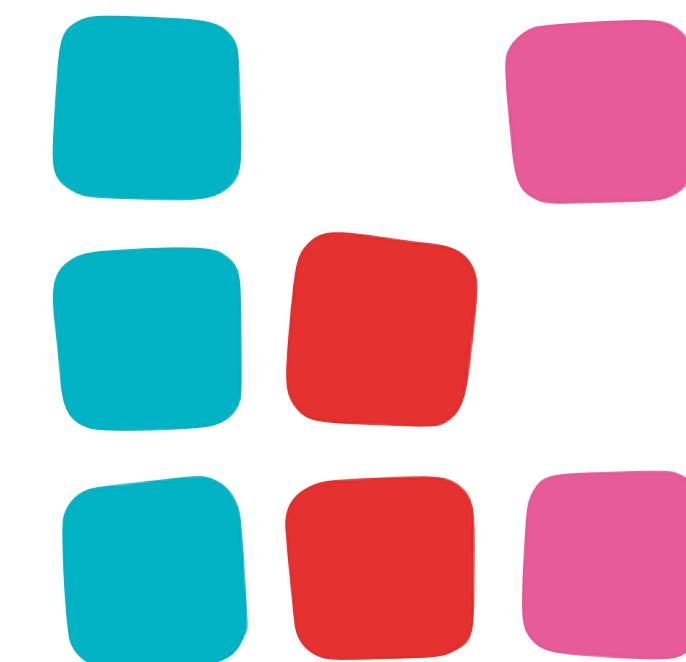
+



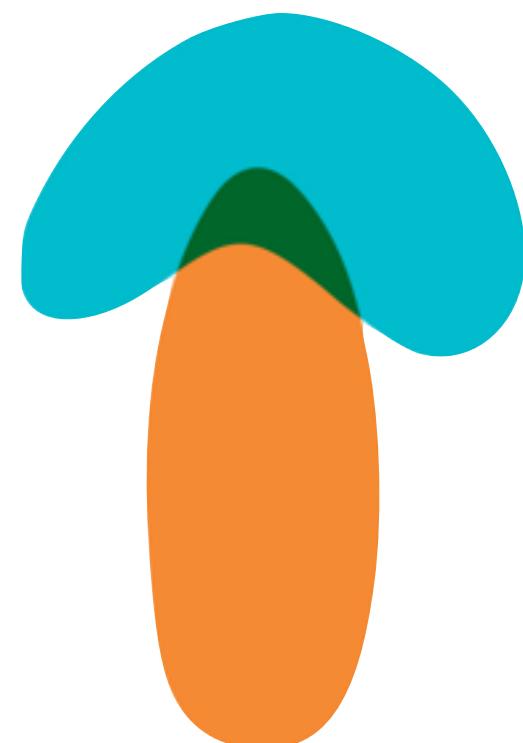
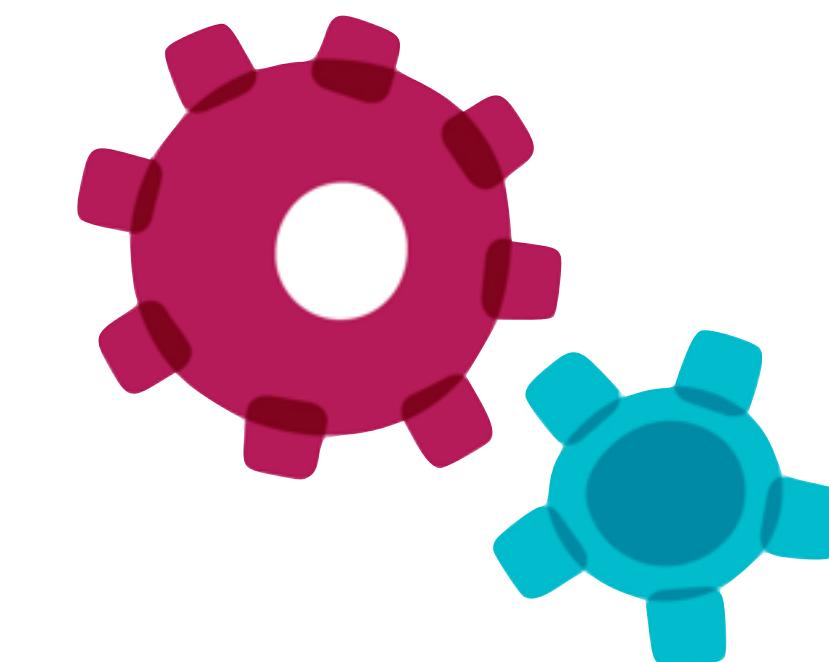
prioritization

# Architecture Defined

architecture =



+



prioritization



cost

# Question:

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

# Question:

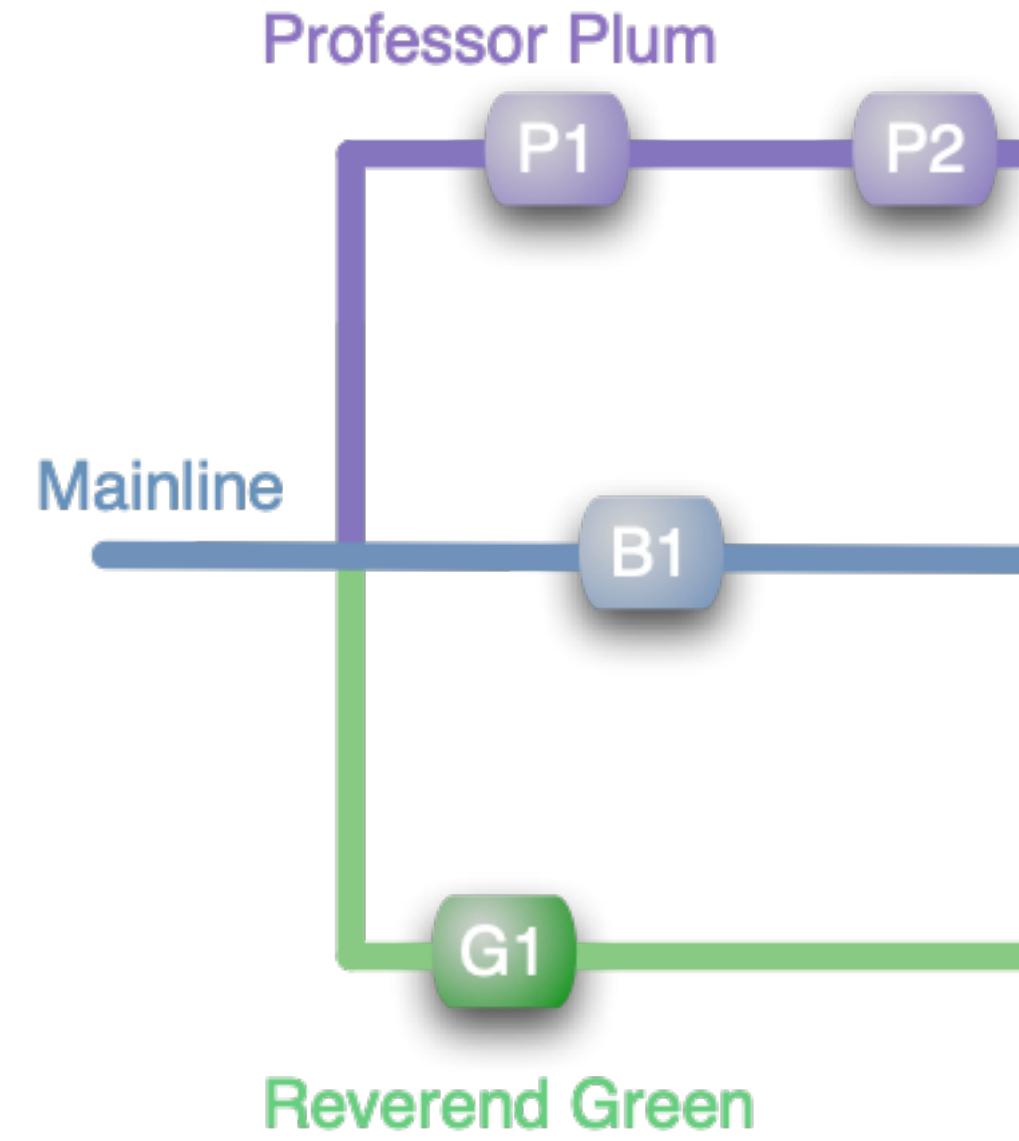
How can teams support advanced architecture tricks like A/B testing while doing continuous delivery?

# Question:

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

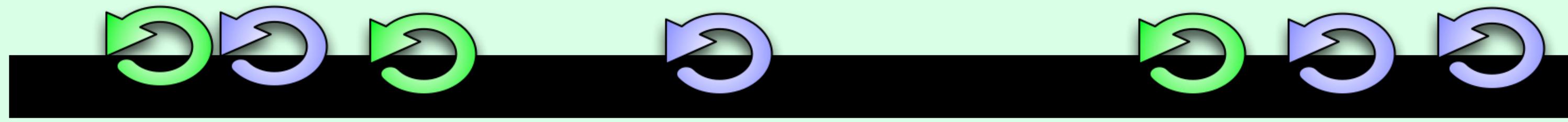
# Question:

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

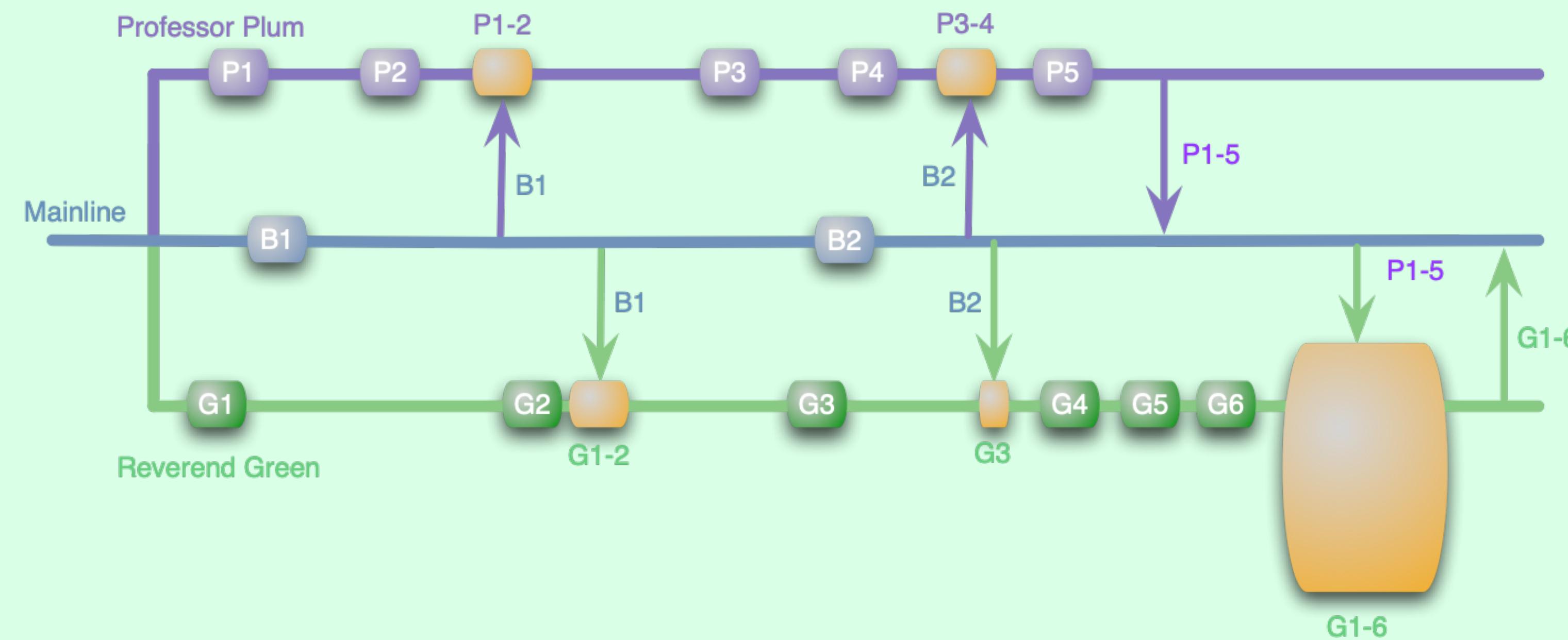


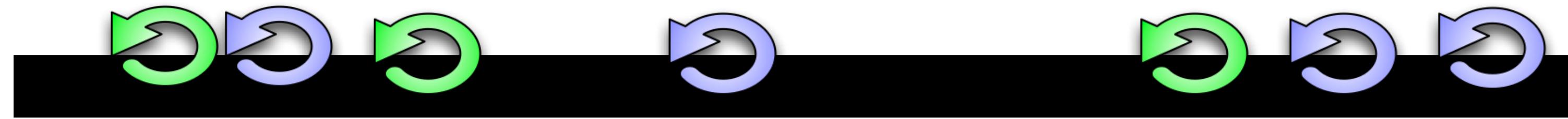
*merge  
ambush!*

Feature Branching

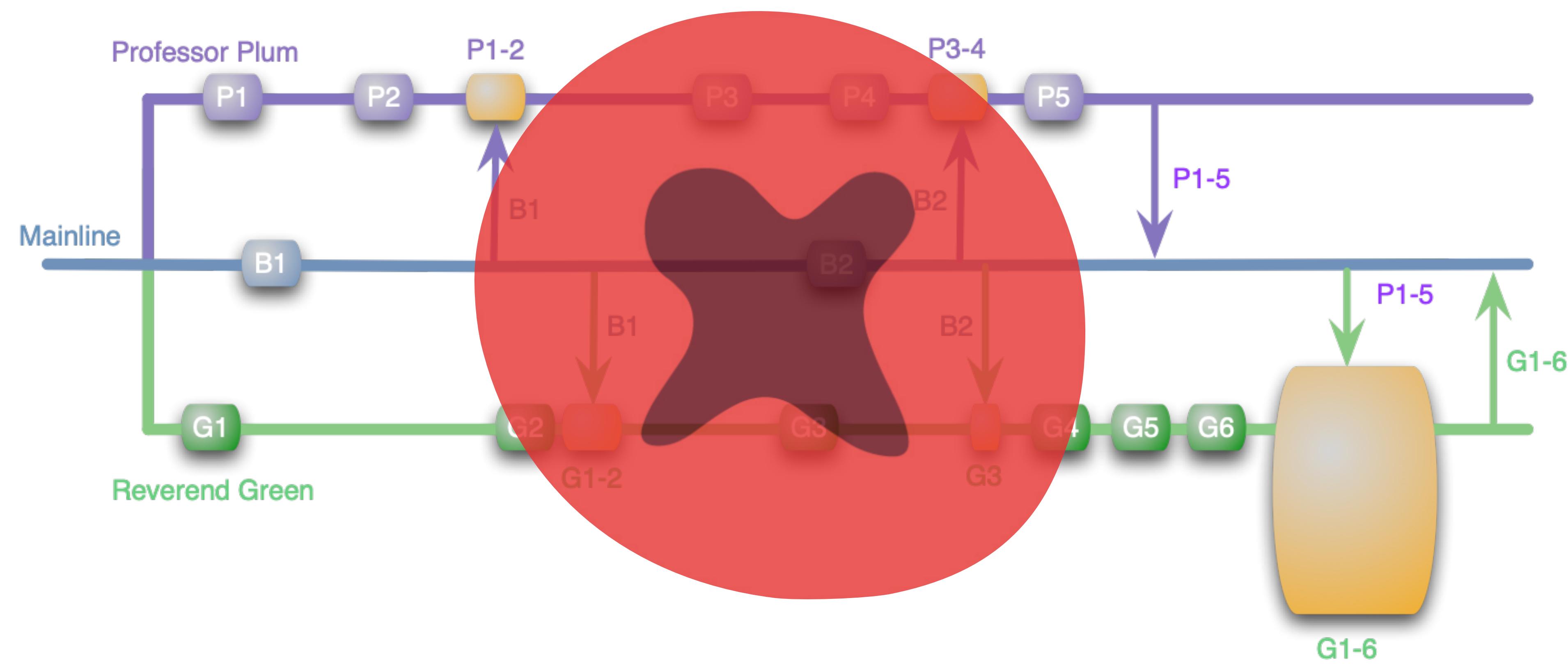


*trunk-based development*  
*versus*  
*feature branching*

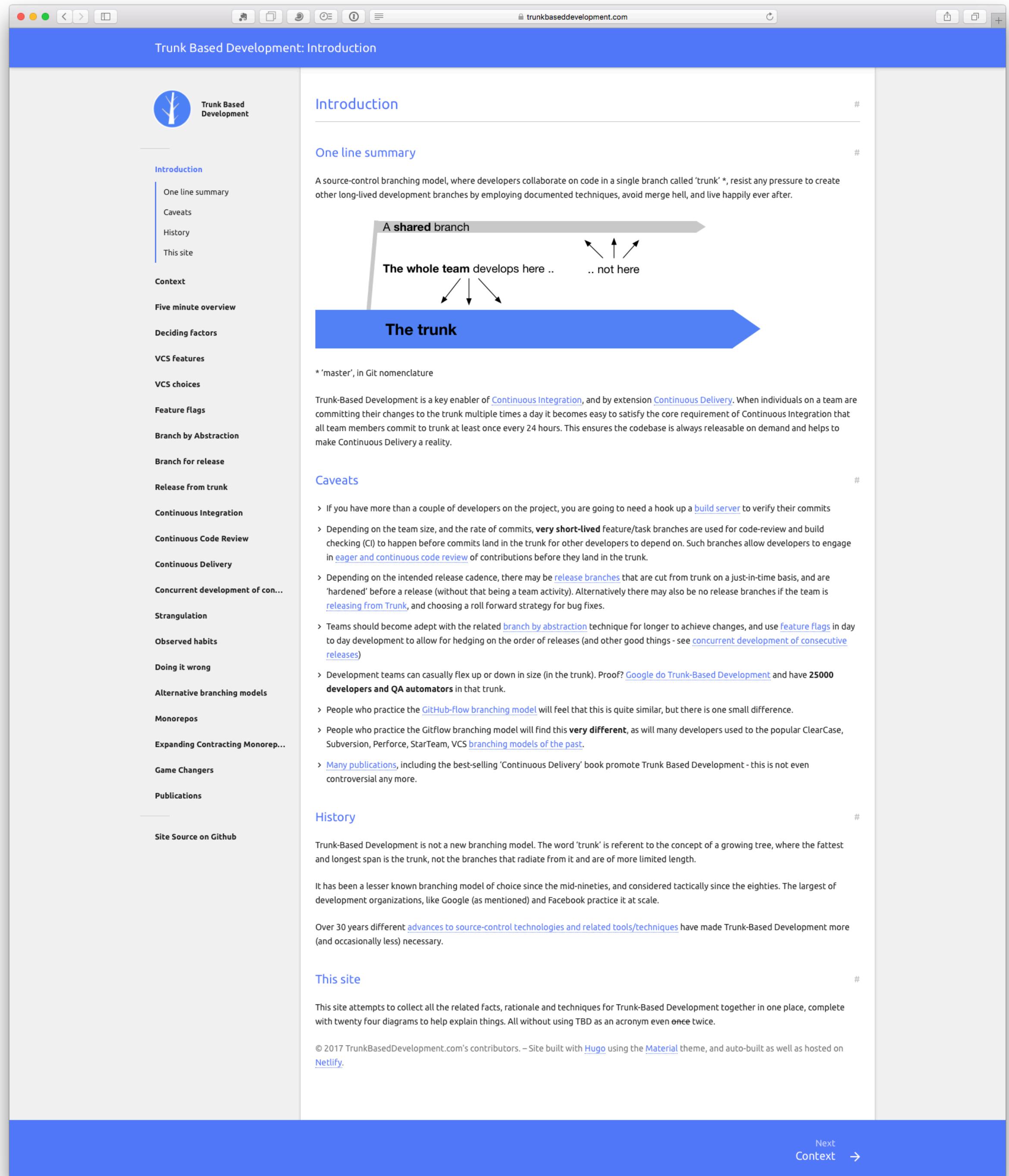




*trunk-based development*



Trunk Based Development: Introduction



The screenshot shows the homepage of the Trunk Based Development website. The header includes the site's name and a navigation bar with links like 'Home', 'About', 'Diagrams', 'Blog', 'Contact', and 'Search'. The main content area has a large title 'Trunk Based Development' with a subtitle 'The trunk-based development model explained'. Below this is a section titled 'Introduction' with a sub-section 'One line summary'. A diagram illustrates the 'trunk-based development model' where a 'shared branch' is at the top, and a 'trunk' is at the bottom. Arrows point from the trunk to the shared branch, indicating that the whole team develops in the trunk. A note says '.. not here' with arrows pointing away from the trunk. The sidebar on the left contains a navigation menu with sections like 'Introduction', '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 Monore...', 'Game Changers', and 'Publications'. At the bottom, there's a link to 'Site Source on Github'.

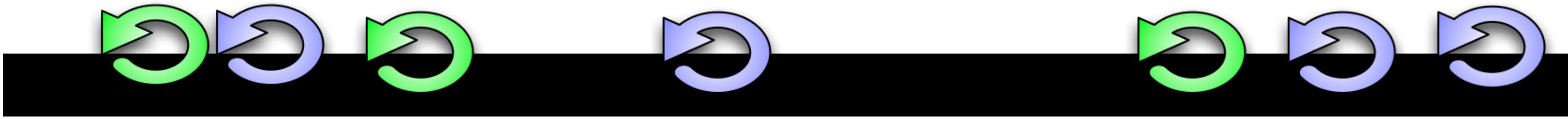


Paul Hammant



*trunk-based development*

<https://trunkbaseddevelopment.com>



*trunk-based development*

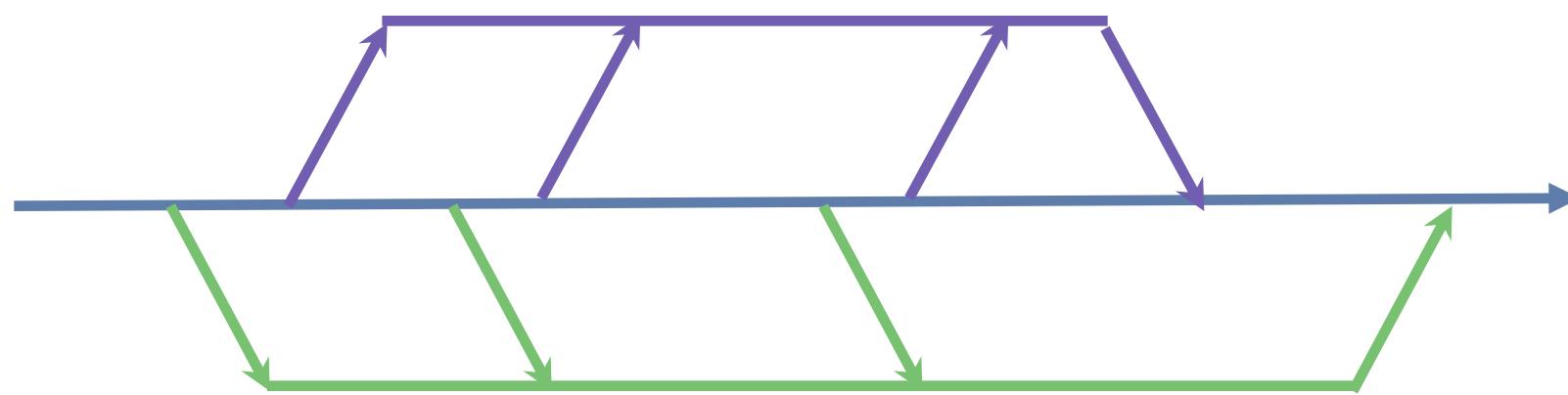


*does it!!*

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

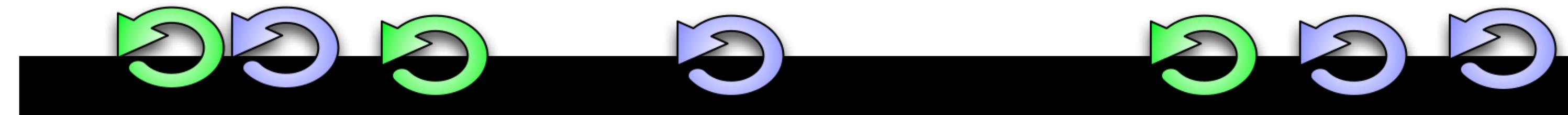
don't panic

Feature Branching



*Big Scary Merge*

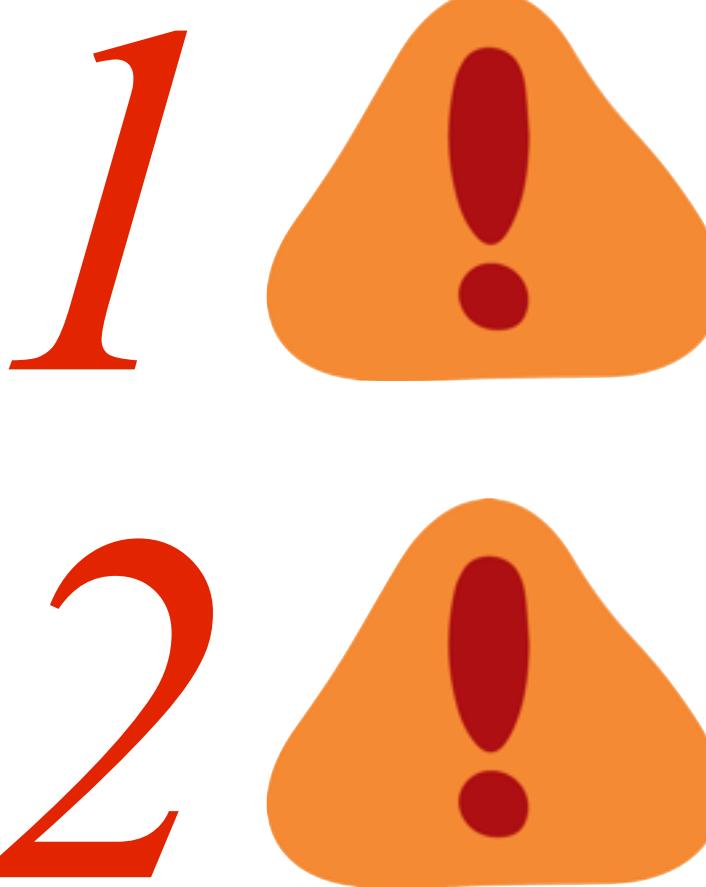
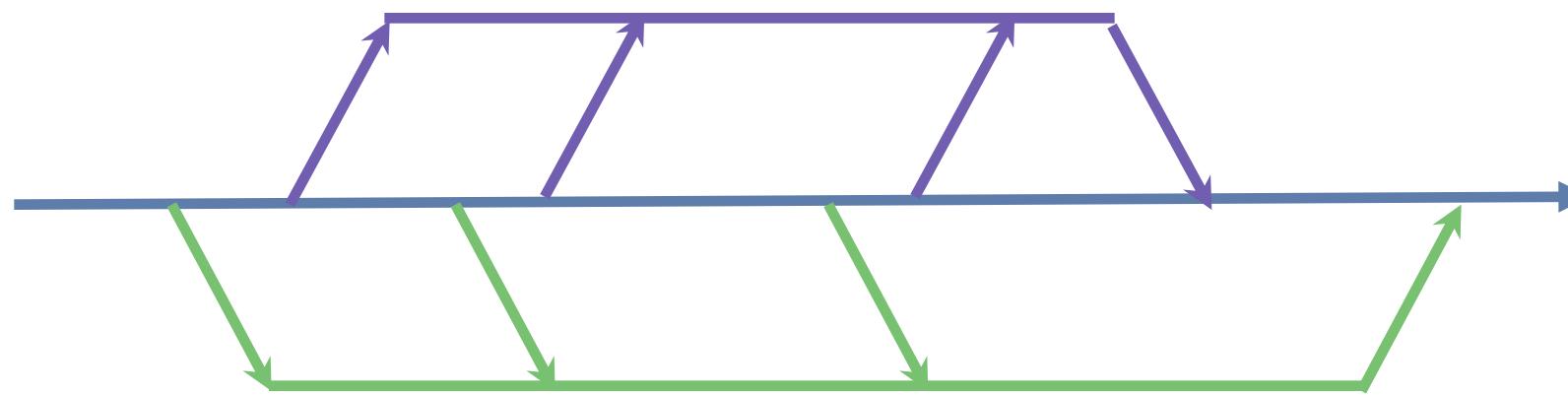
*Cherry Picking*



*Untrusted  
Contributors*

Continuous Integration

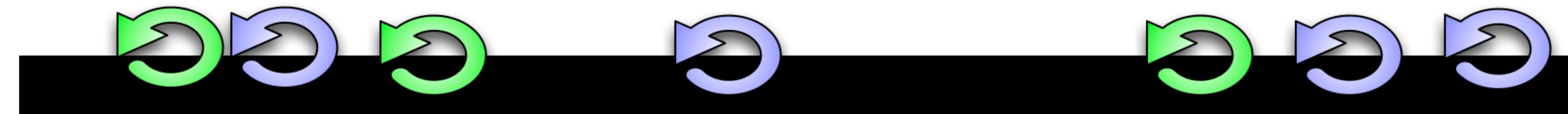
Feature Branching



*Big Scary Merge*

*Discouraging refactoring*

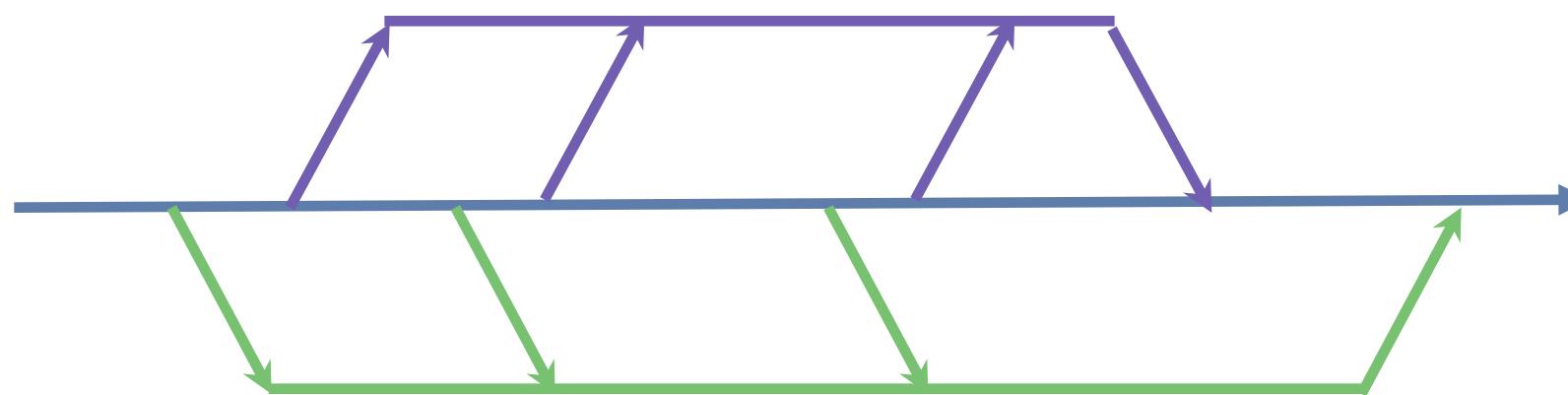
*Cherry Picking*



*Untrusted  
Contributors*

Continuous Integration

## Feature Branching



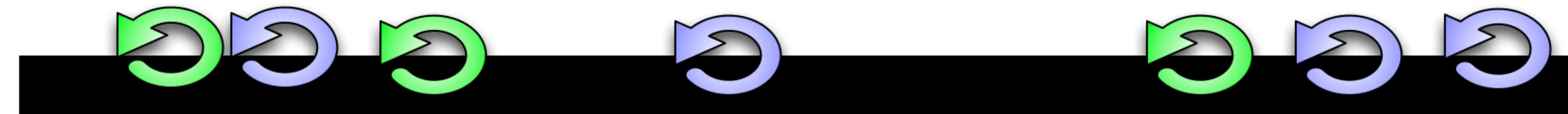
- 1 !
- 2 !
- 3 !

*Big Scary Merge*

*Discouraging refactoring*

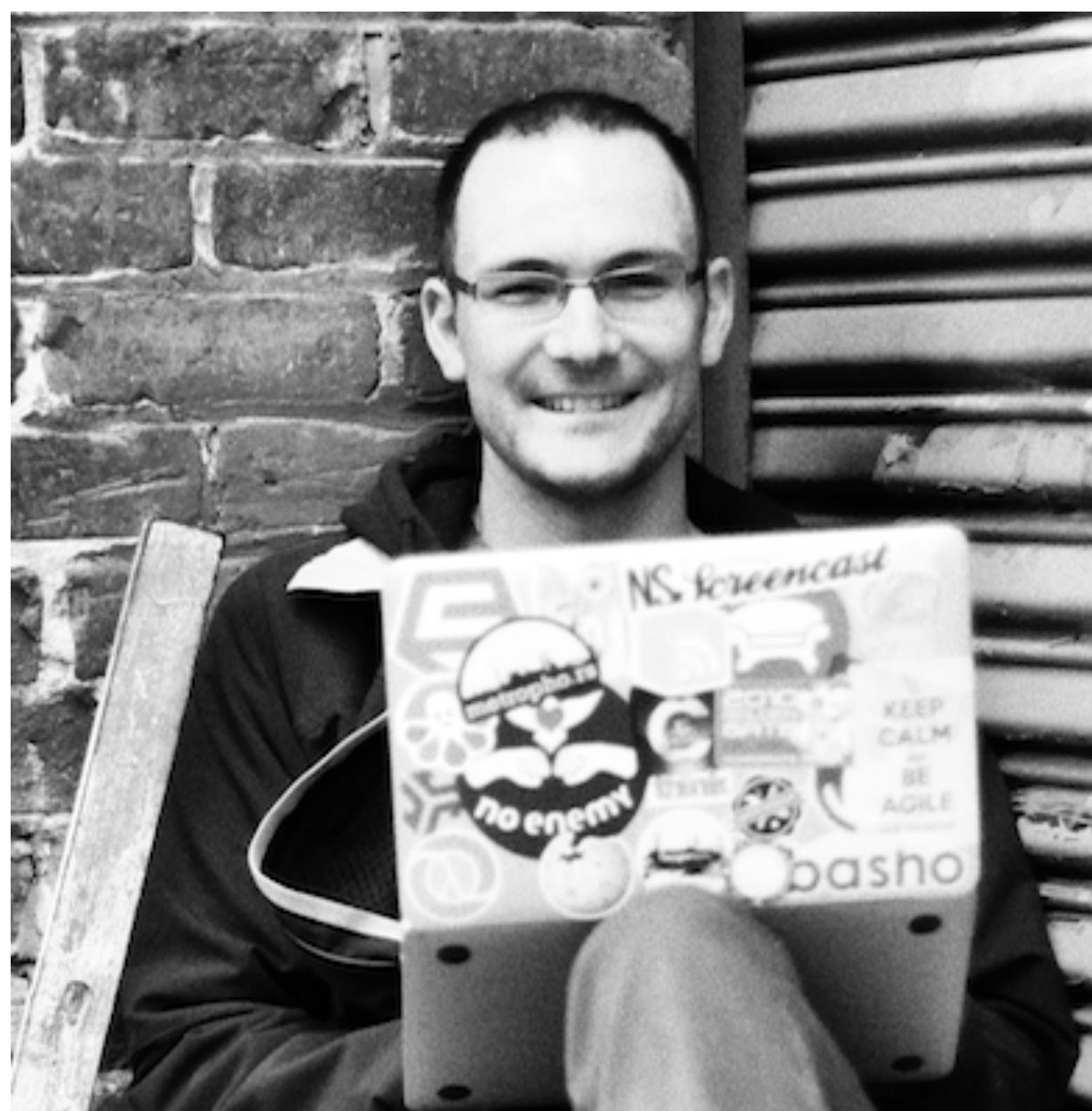
*Hard to combine features*

## Cherry Picking



*Untrusted  
Contributors*

Continuous Integration



<http://martinfowler.com/articles/feature-toggles.html>

The screenshot shows a Mac OS X browser window displaying the Martin Fowler website at [martinfowler.com](http://martinfowler.com). The page title is "Feature Toggles". The main content area contains a block of text about feature toggles, followed by author information for Pete Hodgson, a bio, and a section titled "A Toggling Tale". The right sidebar includes a "Contents" section with links to related articles.

**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](#)

---

"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

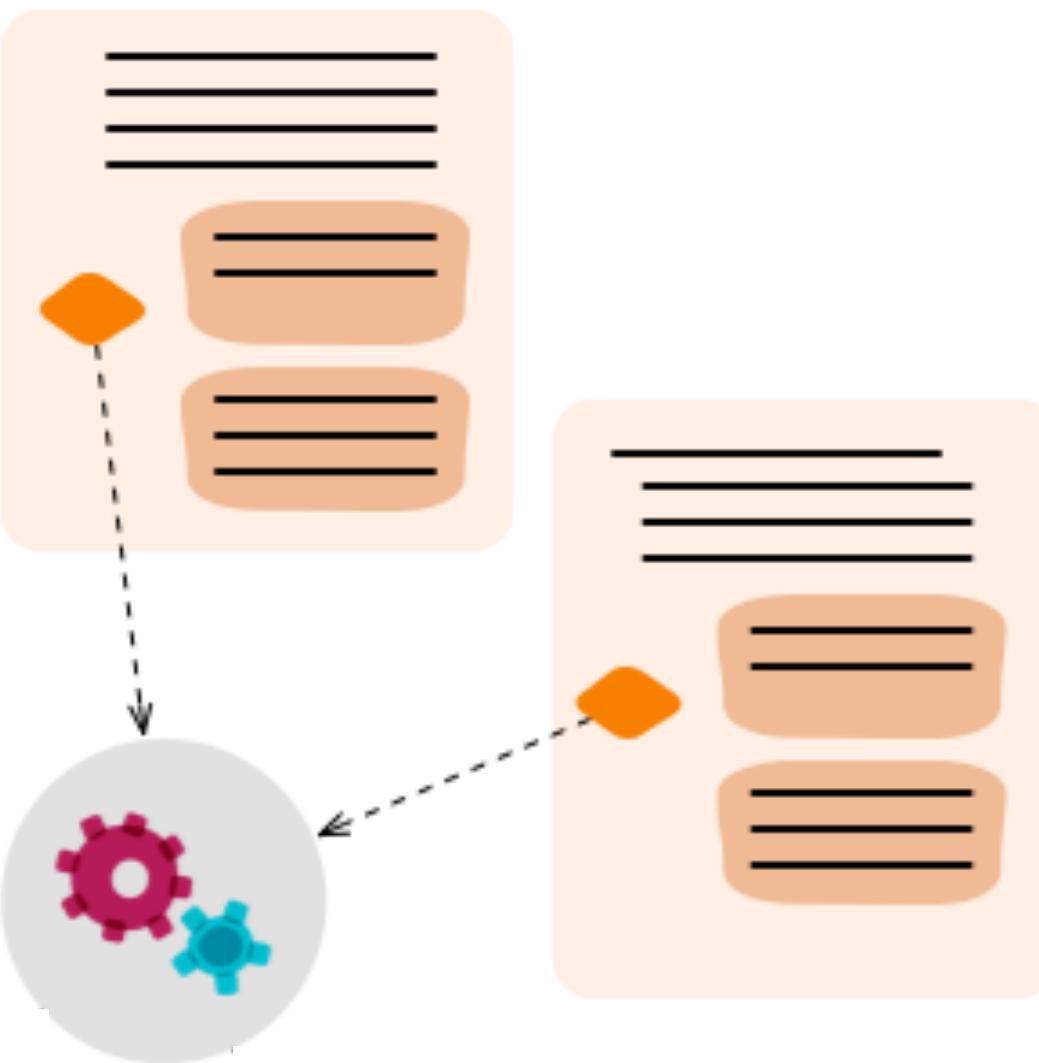
    if( useNewAlgorithm ){
        return enhancedSplineReticulation();
    }else{
        return oldFashionedSplineReticulation();
    }
}

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);

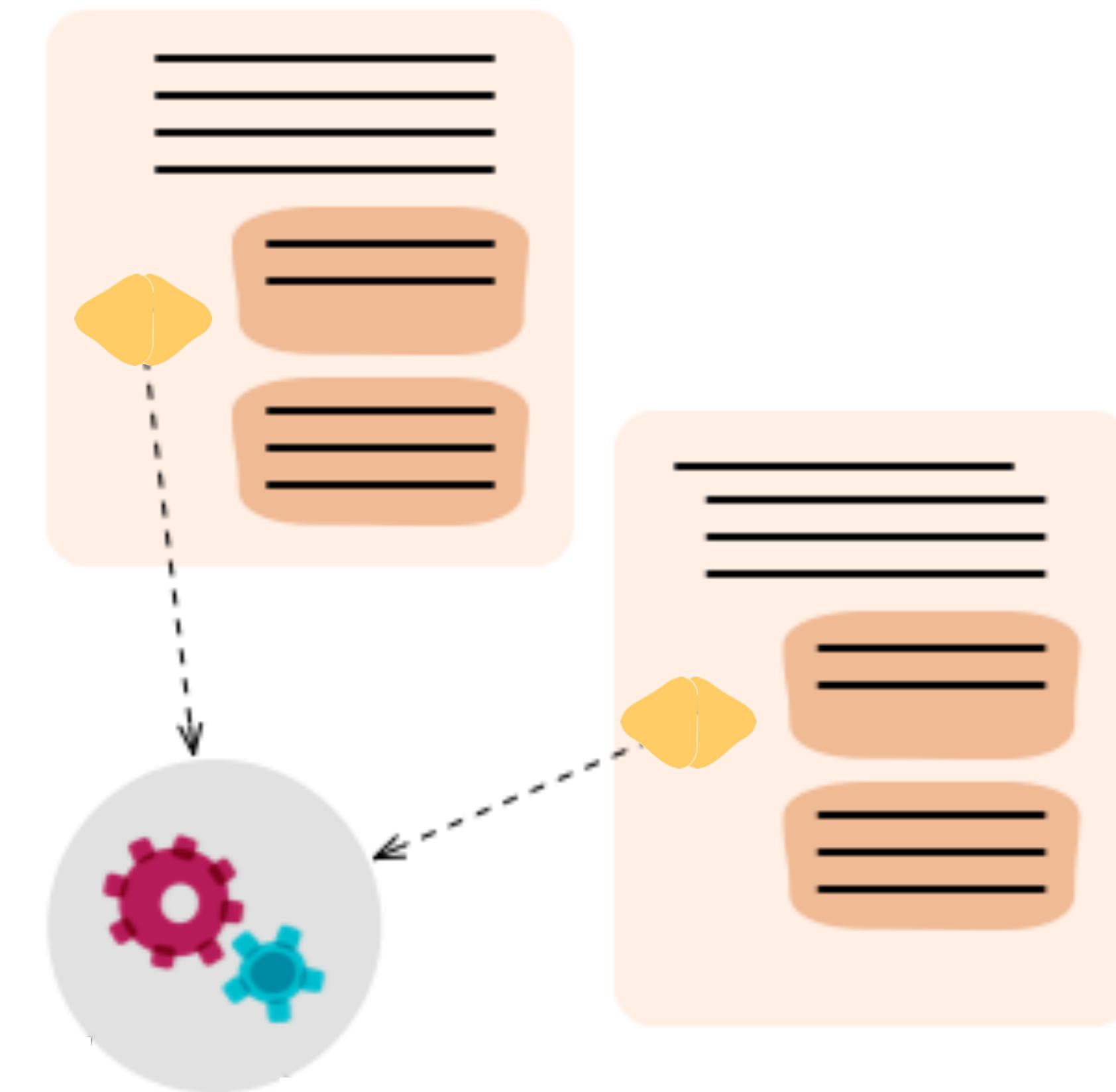
    // When
    const result = simulationEngine.doSomethingWhichInvolvesSplineReticulation();

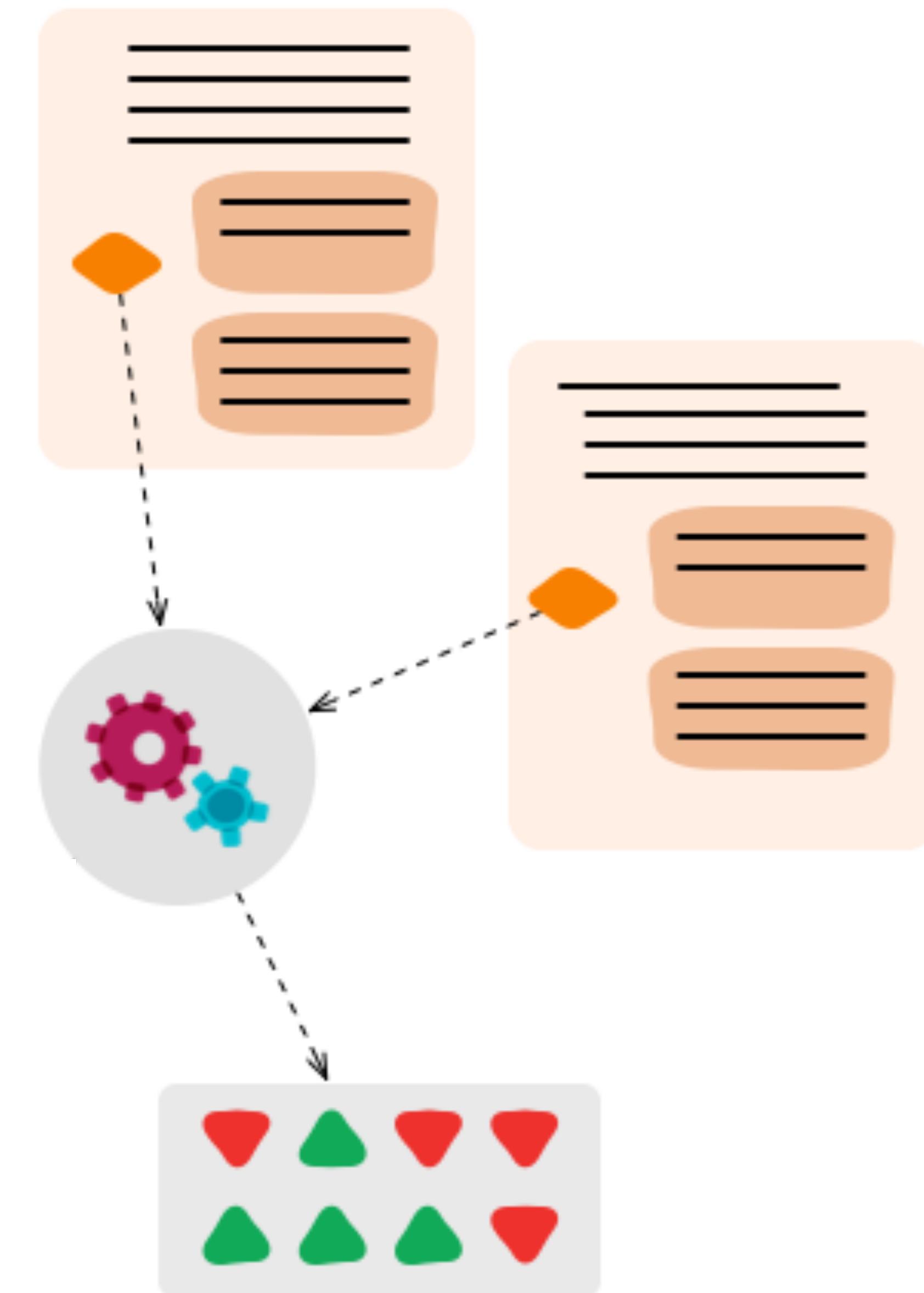
    // Then
    verifySplineReticulation(result);
  });

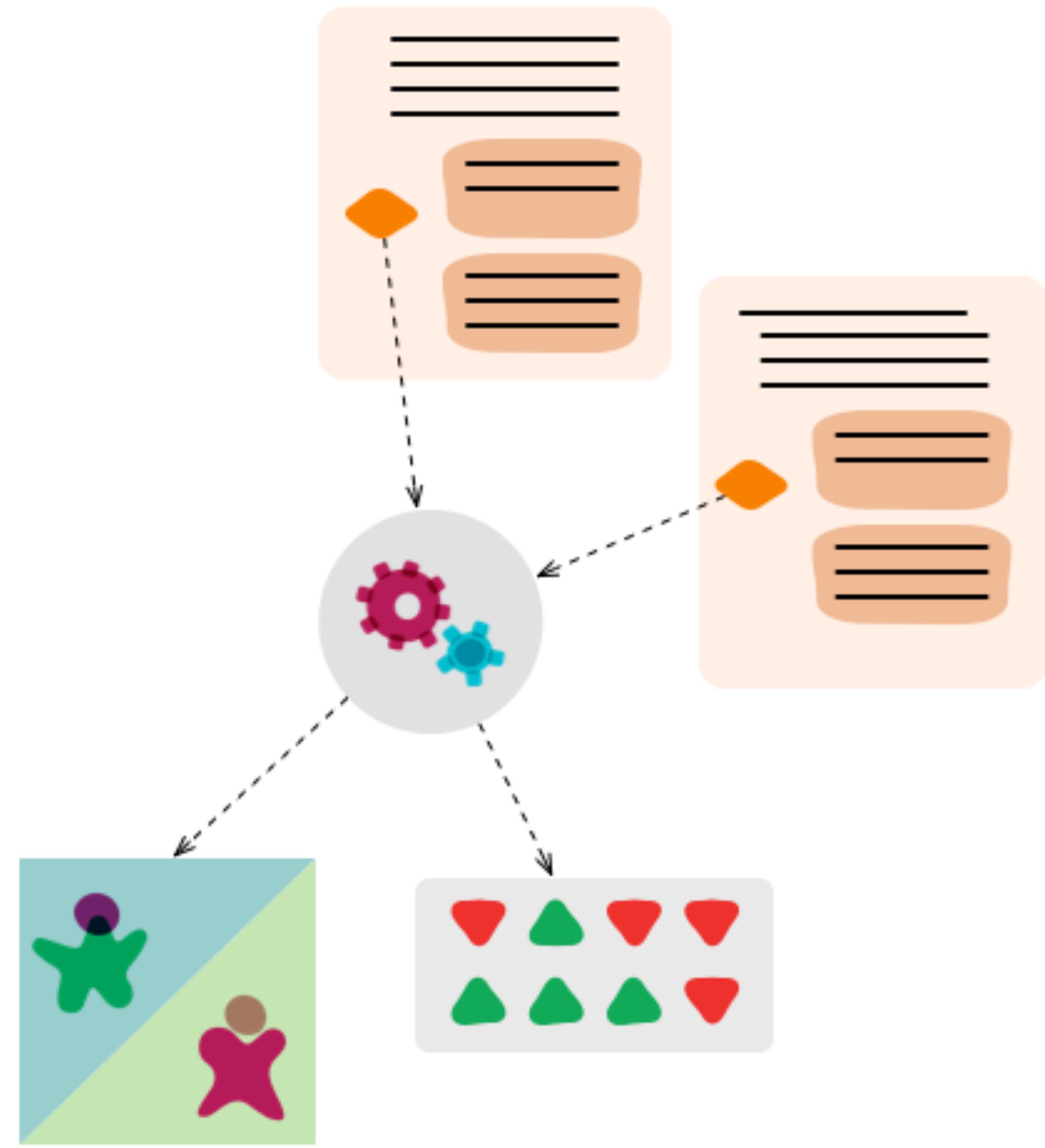
  it('works correctly with new algorithm', function(){
    // Given
    toggleRouter.setFeature("use-new-SR-algorithm",true);

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

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





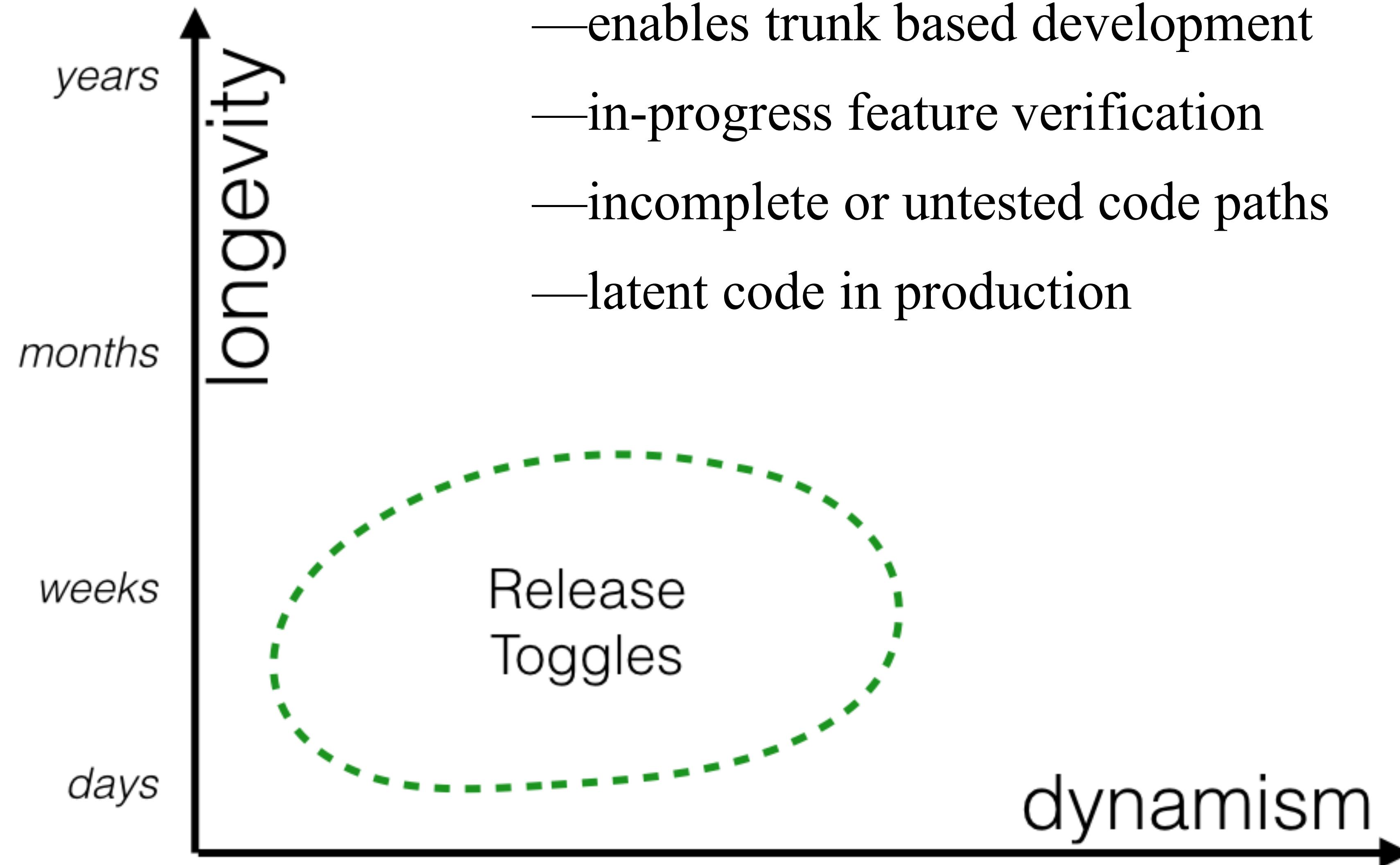




*changes with  
a deployment*

*changes with runtime  
re-configuration*

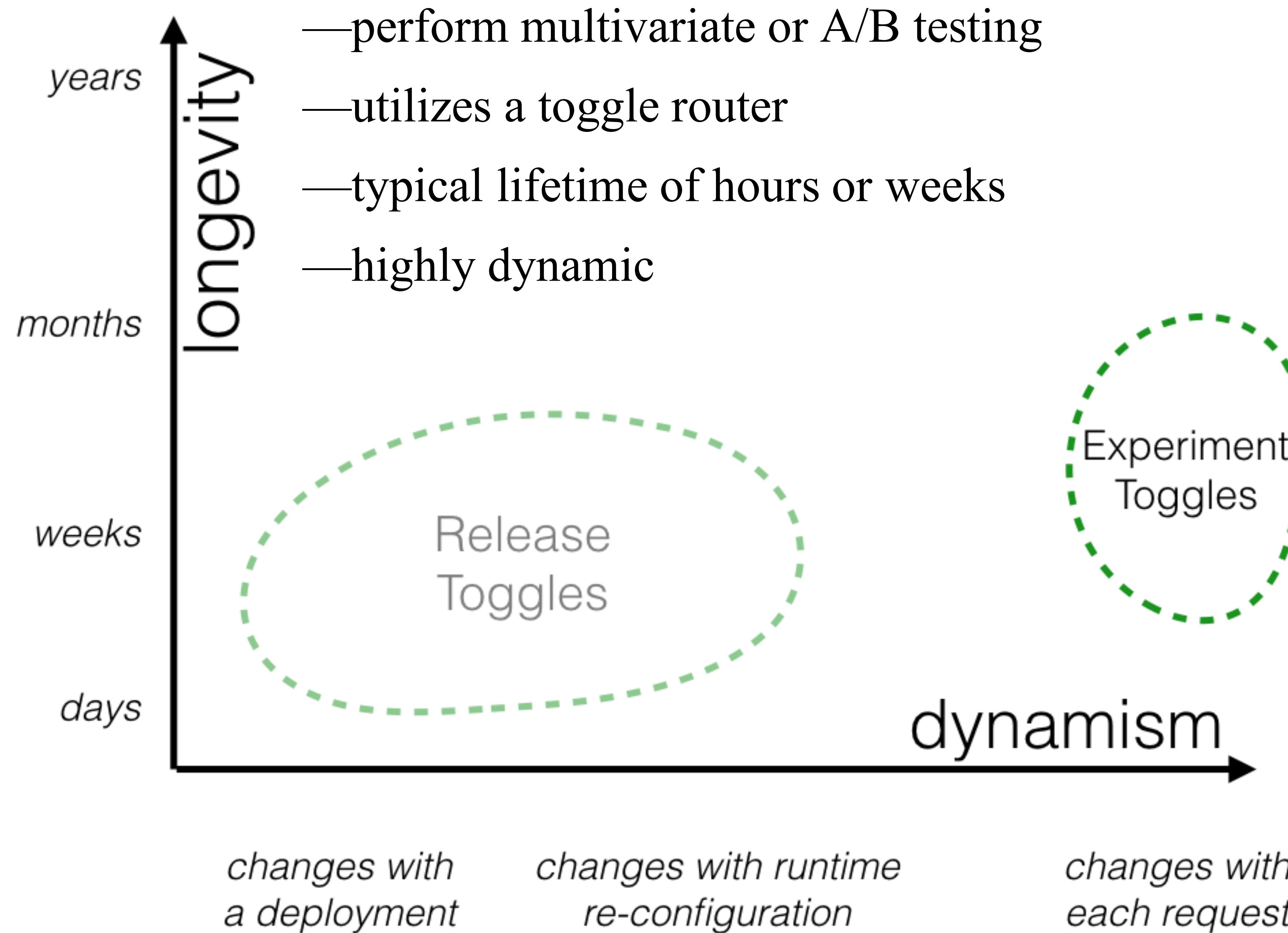
*changes with  
each request*

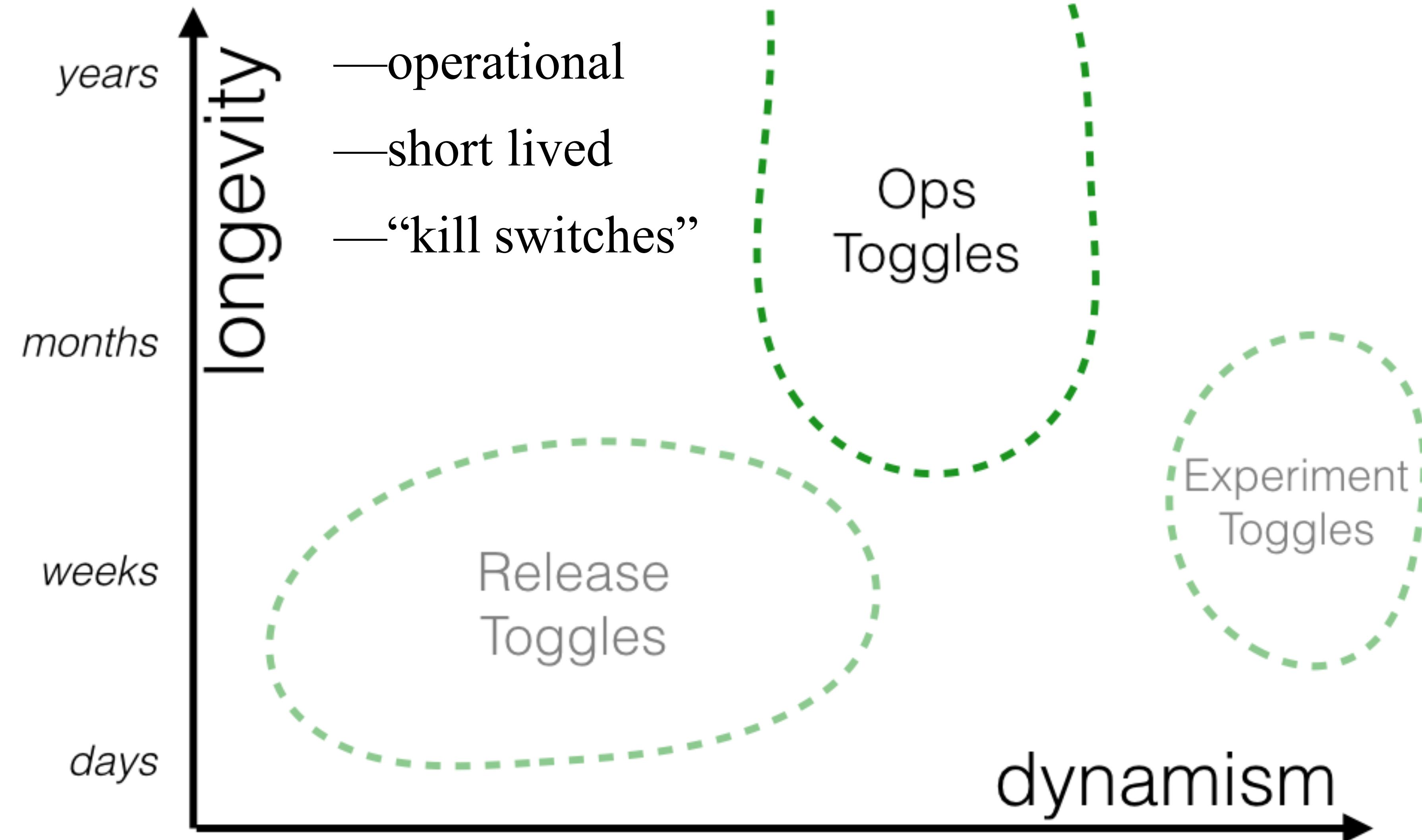


*changes with  
a deployment*

*changes with runtime  
re-configuration*

*changes with  
each request*

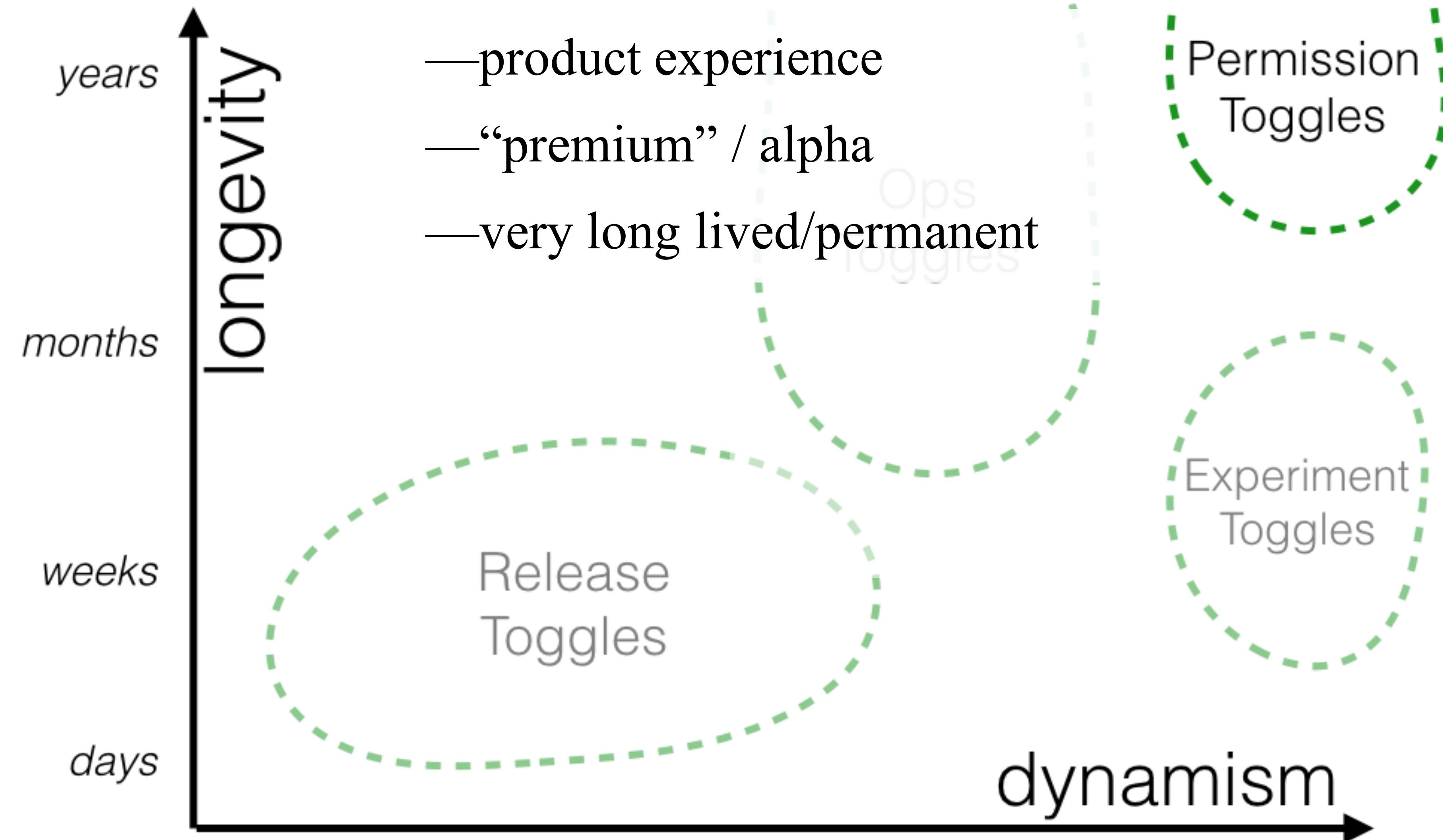




*changes with  
a deployment*

*changes with runtime  
re-configuration*

*changes with  
each request*

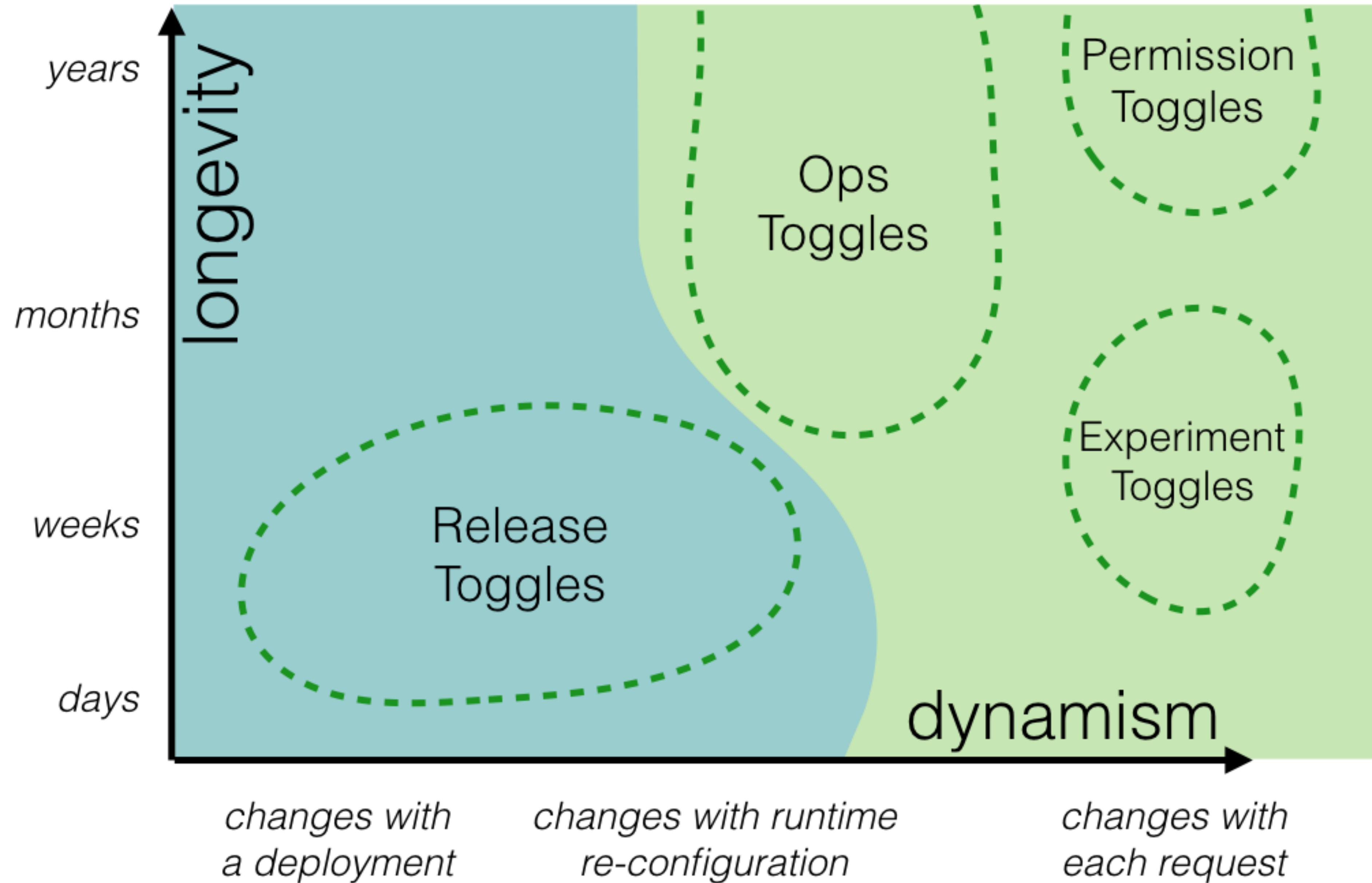


*changes with  
a deployment*

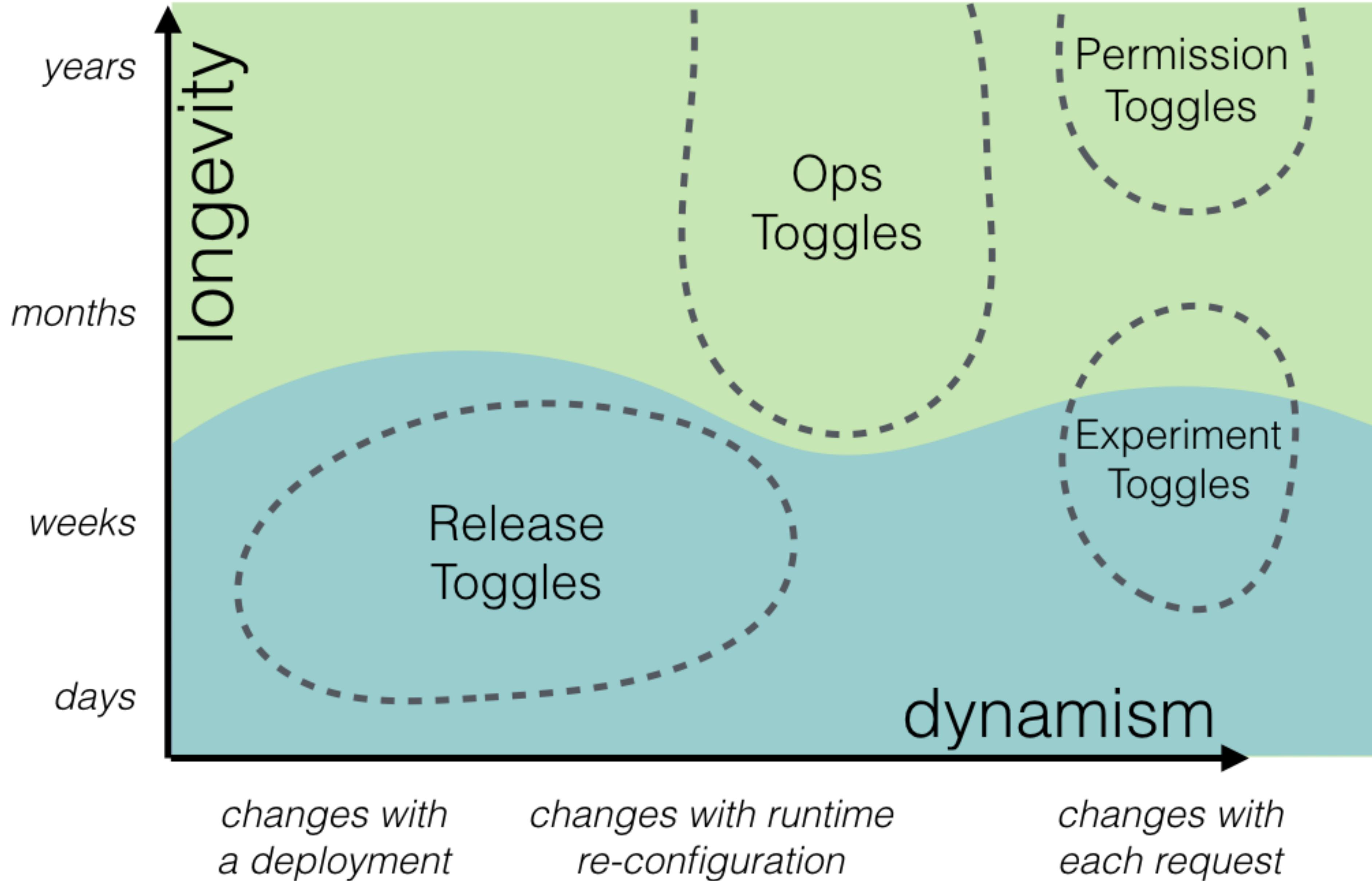
*changes with runtime  
re-configuration*

*changes with  
each request*

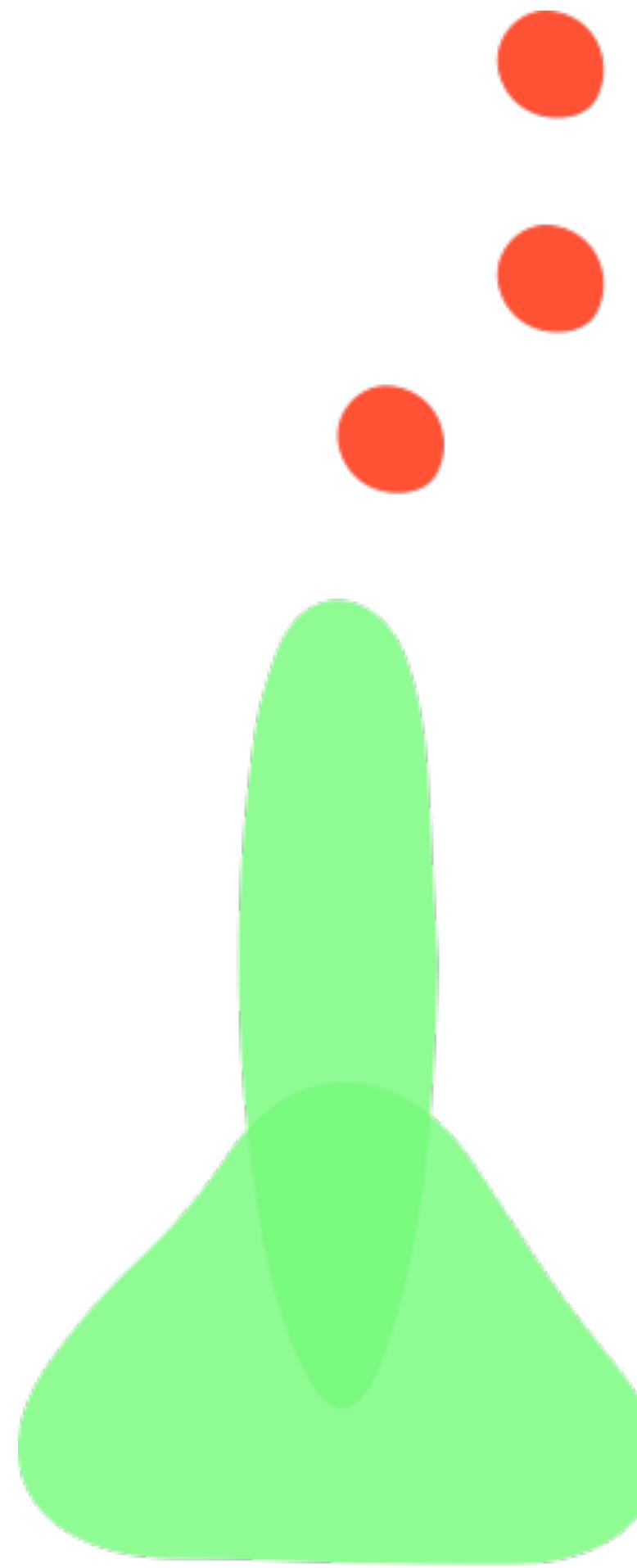
# static versus dynamic toggles

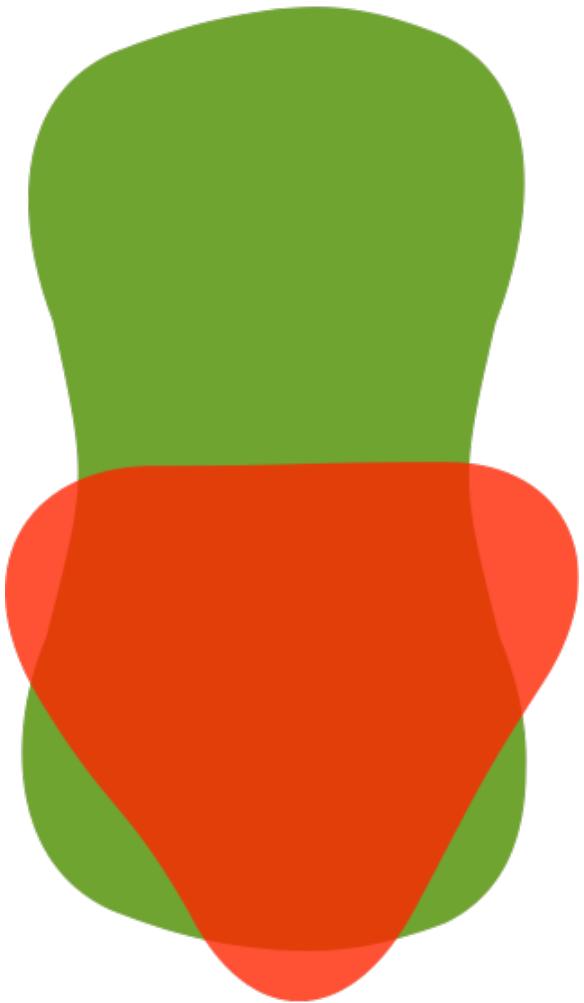


# long-lived versus transient toggles



# 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.**

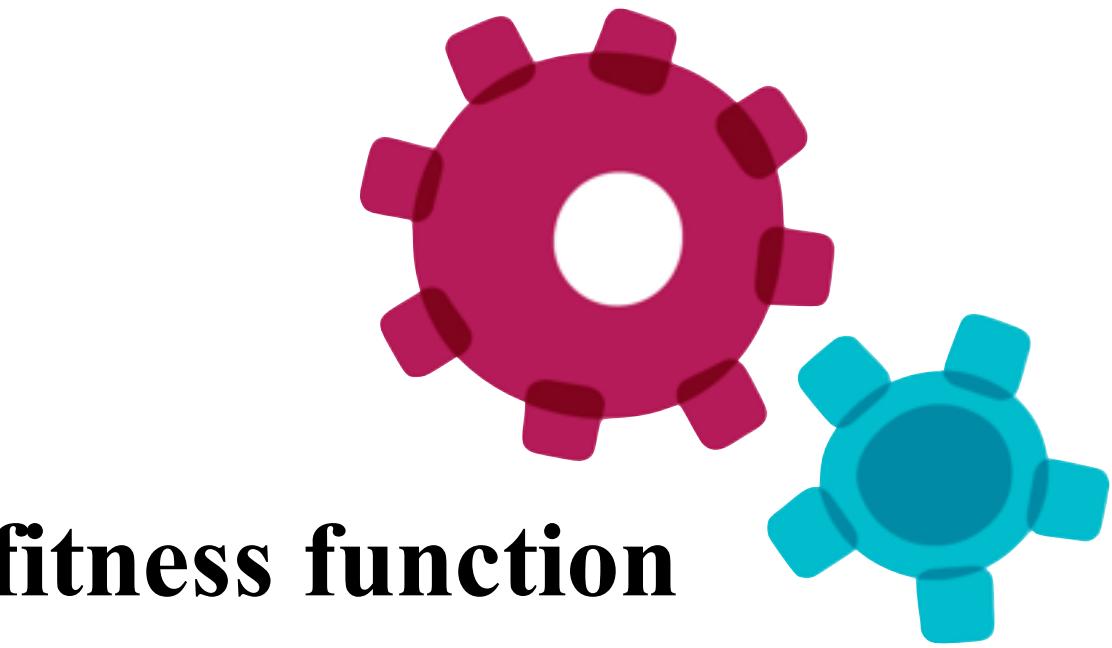
Global Bank Automates Build Management Cutting Dev Time in Half

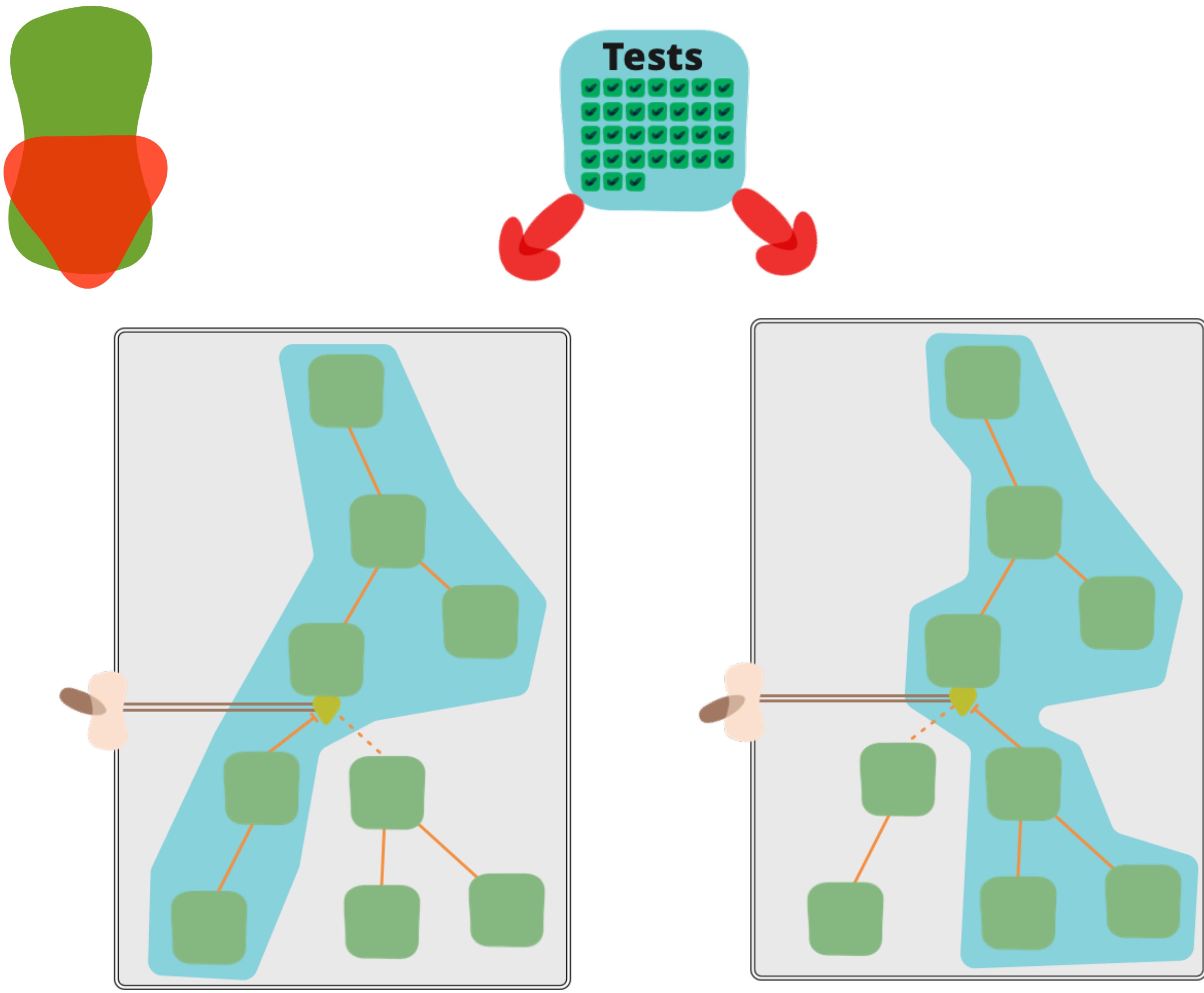
READ THE CASE STUDY

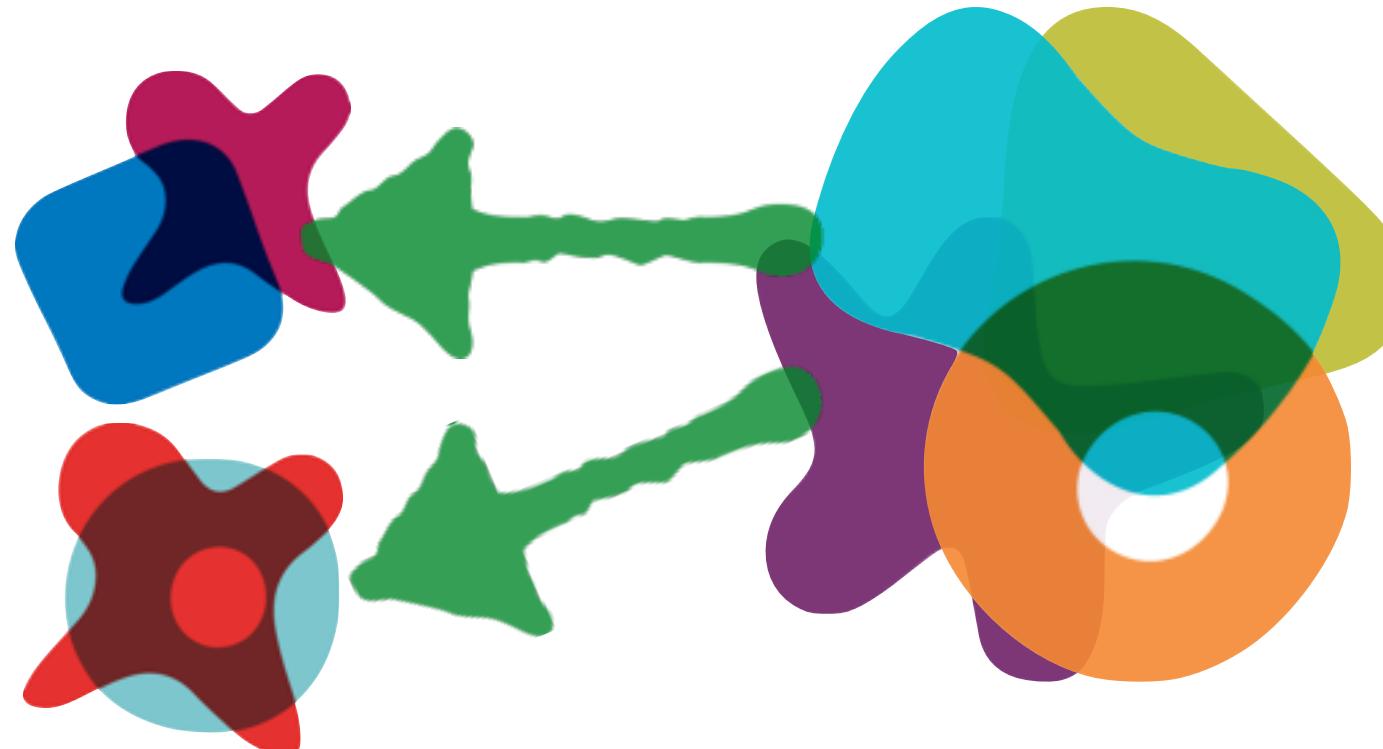
 CloudBees The Enterprise Jenkins Company

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

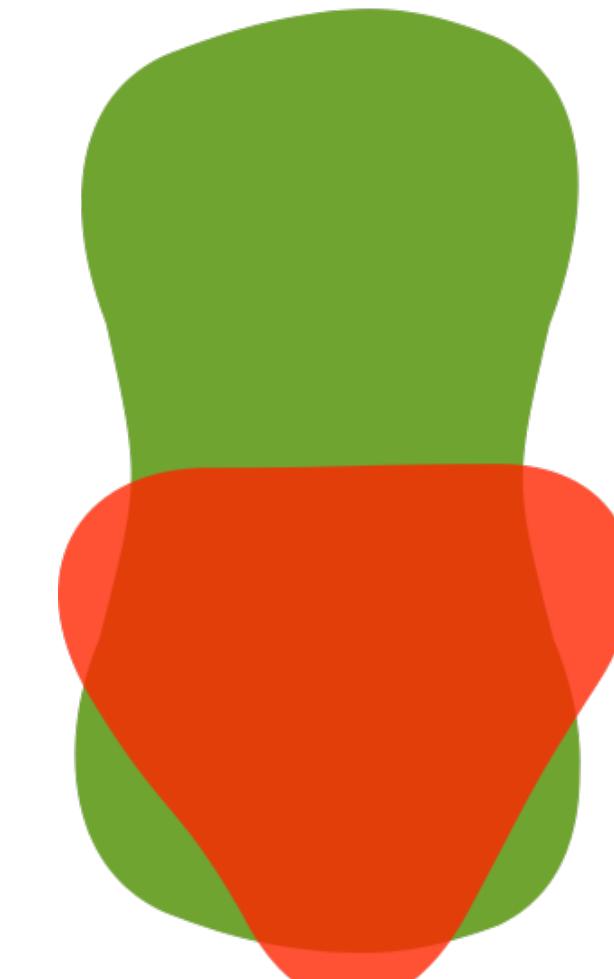
The screenshot shows a web browser displaying a DZone article. The title is "Feature Toggles are one of the Worst kinds of Technical Debt" by Jim Bird. The article discusses technical debt and feature toggles. Below the article, there is a sidebar for New Relic with a case study about a global bank automating build management.

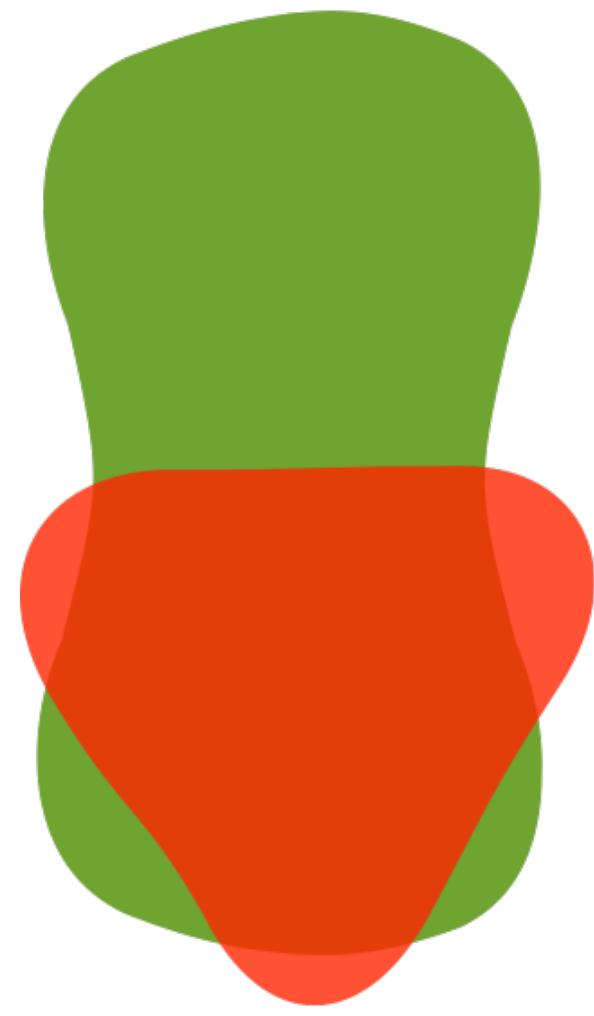
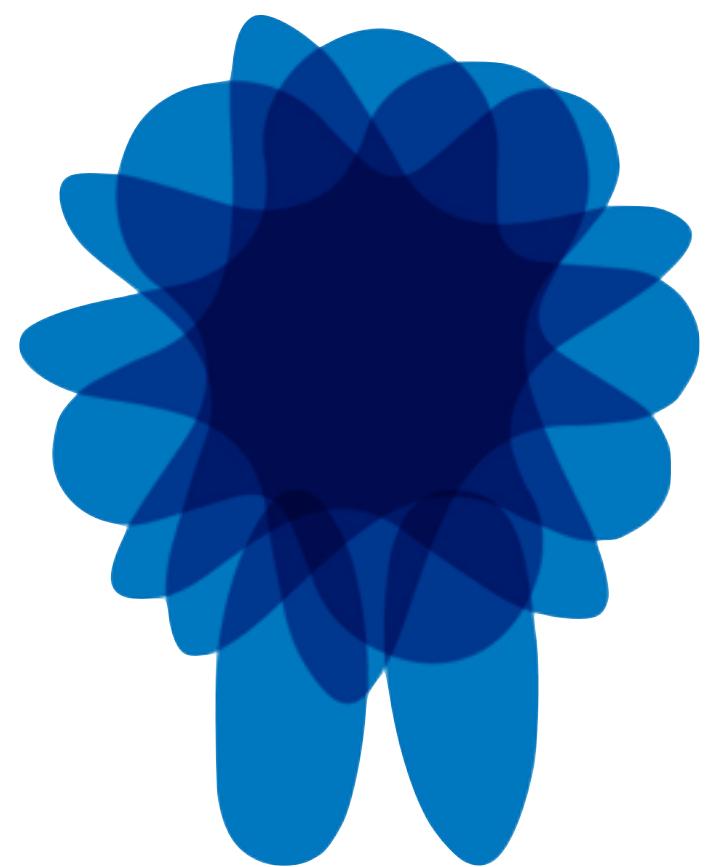






don't create toggles that depend  
on other toggles.





works on all platforms &  
technology stacks

Togglz - Features flag for Java

www.togglz.org

Reader



**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.

The screenshot shows a web browser window with the title "Togglz – Features flag for Java". The URL in the address bar is "www.togglz.org/documentation/activation-strategies.html". The page content is titled "Activation Strategies". It includes a sidebar with links for "MAIN" (Home, Downloads, Source Code, Forums, Issue Tracker, stackoverflow.com, Continuous Integration, License) and "REFERENCE" (What's new?, Getting Started, Javadocs 2.0.0.Final, Javadocs 1.1.0.Final, Javadocs 1.0.0.Final, Updating Notes). The main content area describes activation strategies, lists default strategies (Username, Gradual rollout, Release date, Client IP, Server IP, ScriptEngine), and provides notes on building custom strategies. A section titled "Username" details how features can be activated for specific users.

```
@FeatureGroup
@Label("Performance Improvements")
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Performance {
    // no content
}
```

Togglz – Features flag for Java

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

Reader



## Feature Flags for the Java platform

**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.

MAIN

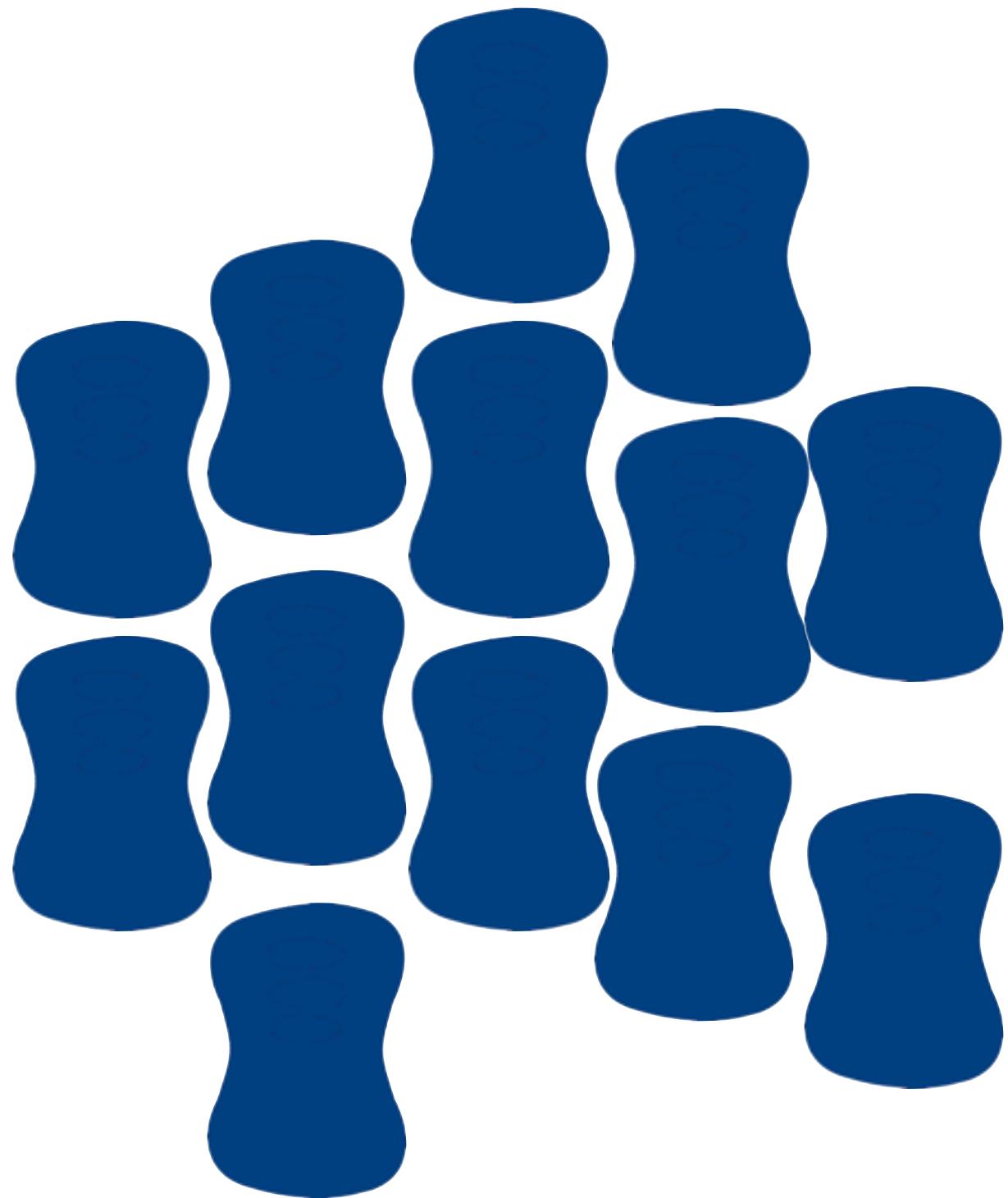
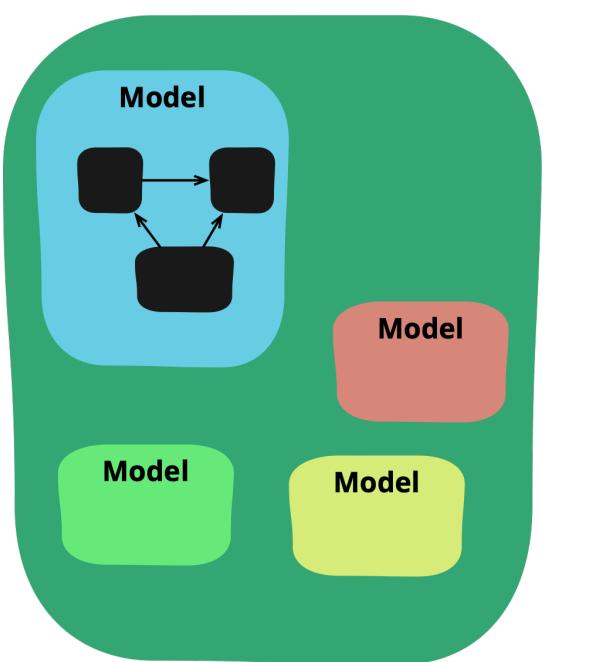
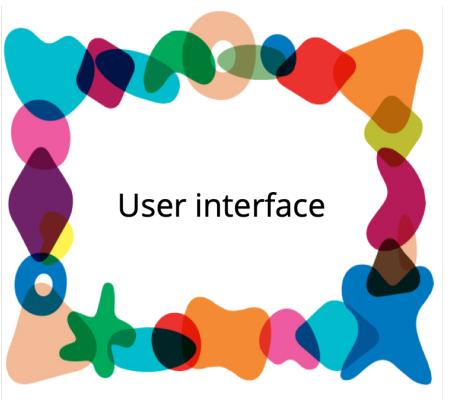
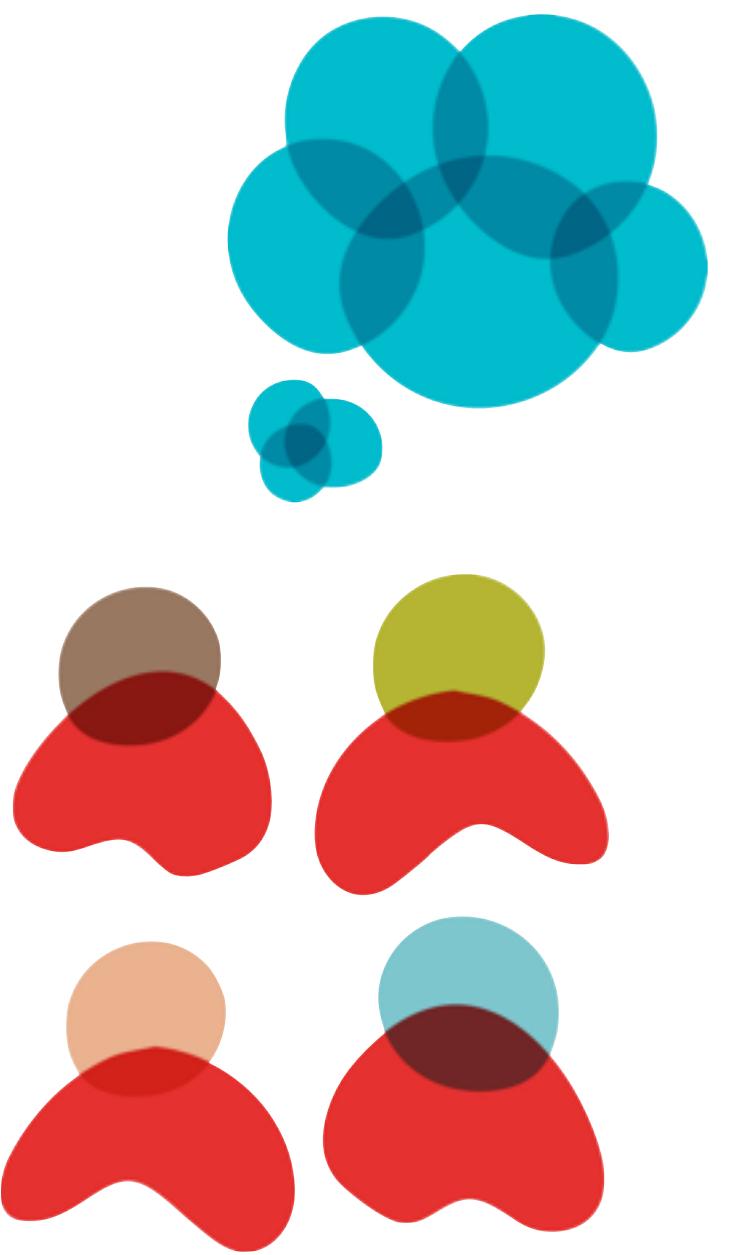
- [Home](#)
- [Downloads](#)
- [Source Code](#)
- [Forums](#)
- [Issue Tracker](#)
- [stackoverflow.com](#)
- [Continuous Integration](#)
- [License](#)

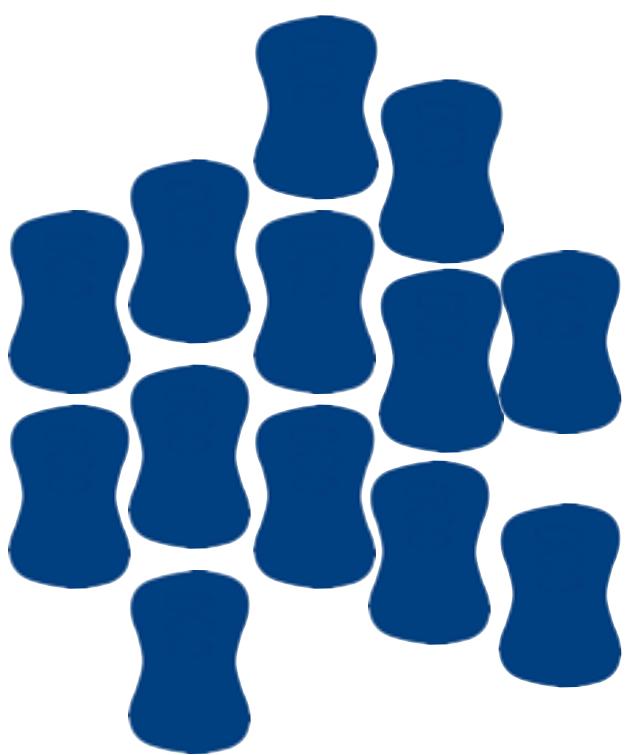
REFERENCE

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

DOCUMENTATION

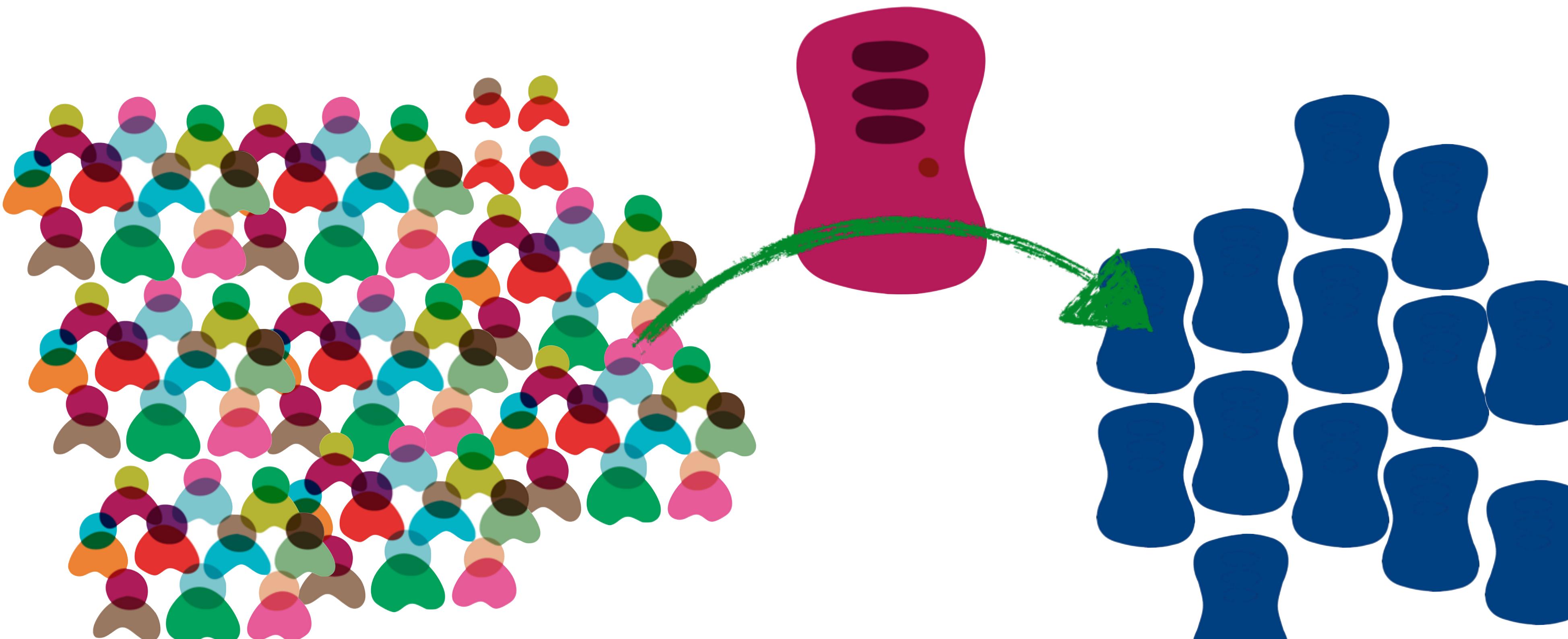
- [Overview](#)
- [Installation](#)
- [Configuration](#)
- [Usage](#)
- [Admin Console](#)



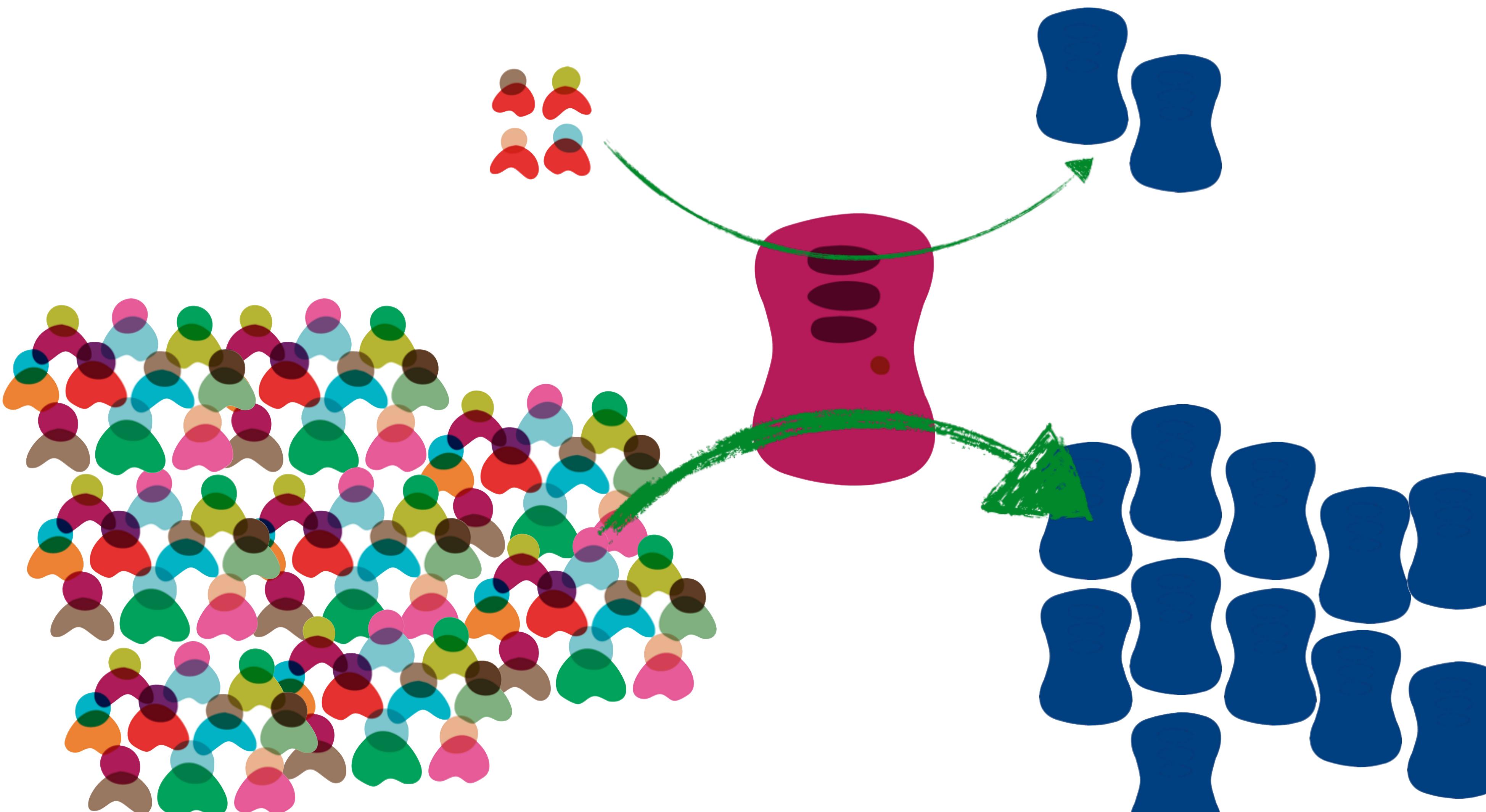


canary releasing

# canary releasing



# canary releasing



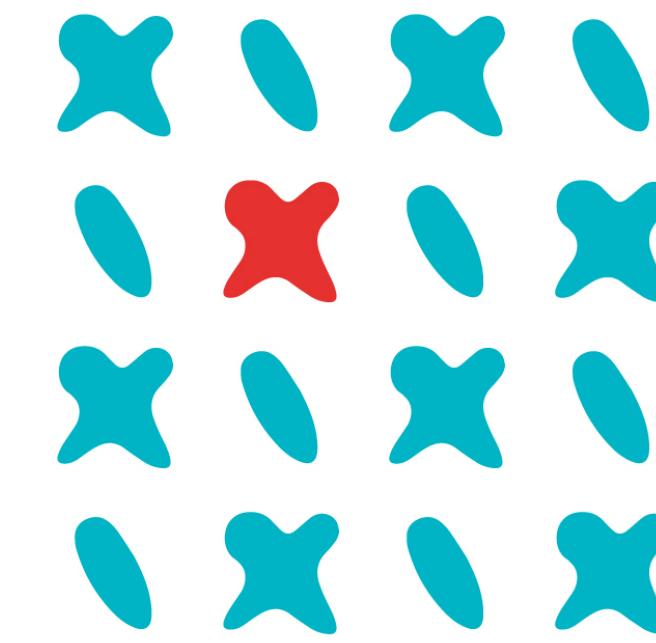
# canary releasing

reduce risk of release



# canary releasing

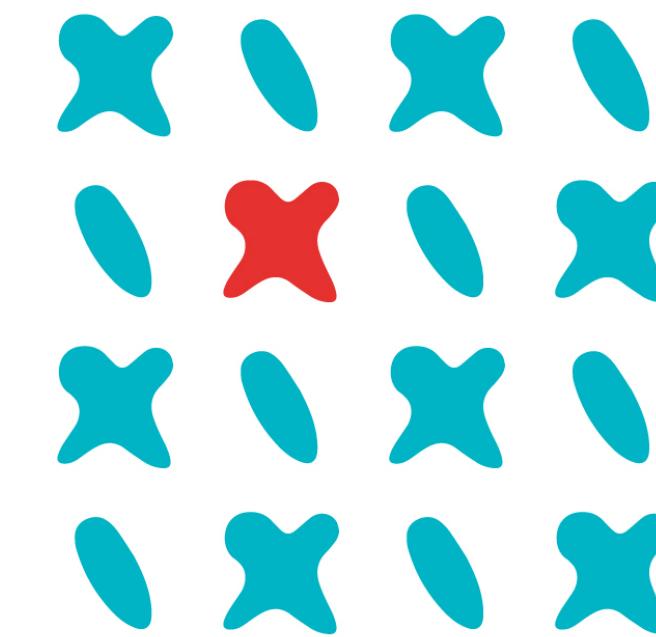
reduce risk of release



multi-variant testing

# canary releasing

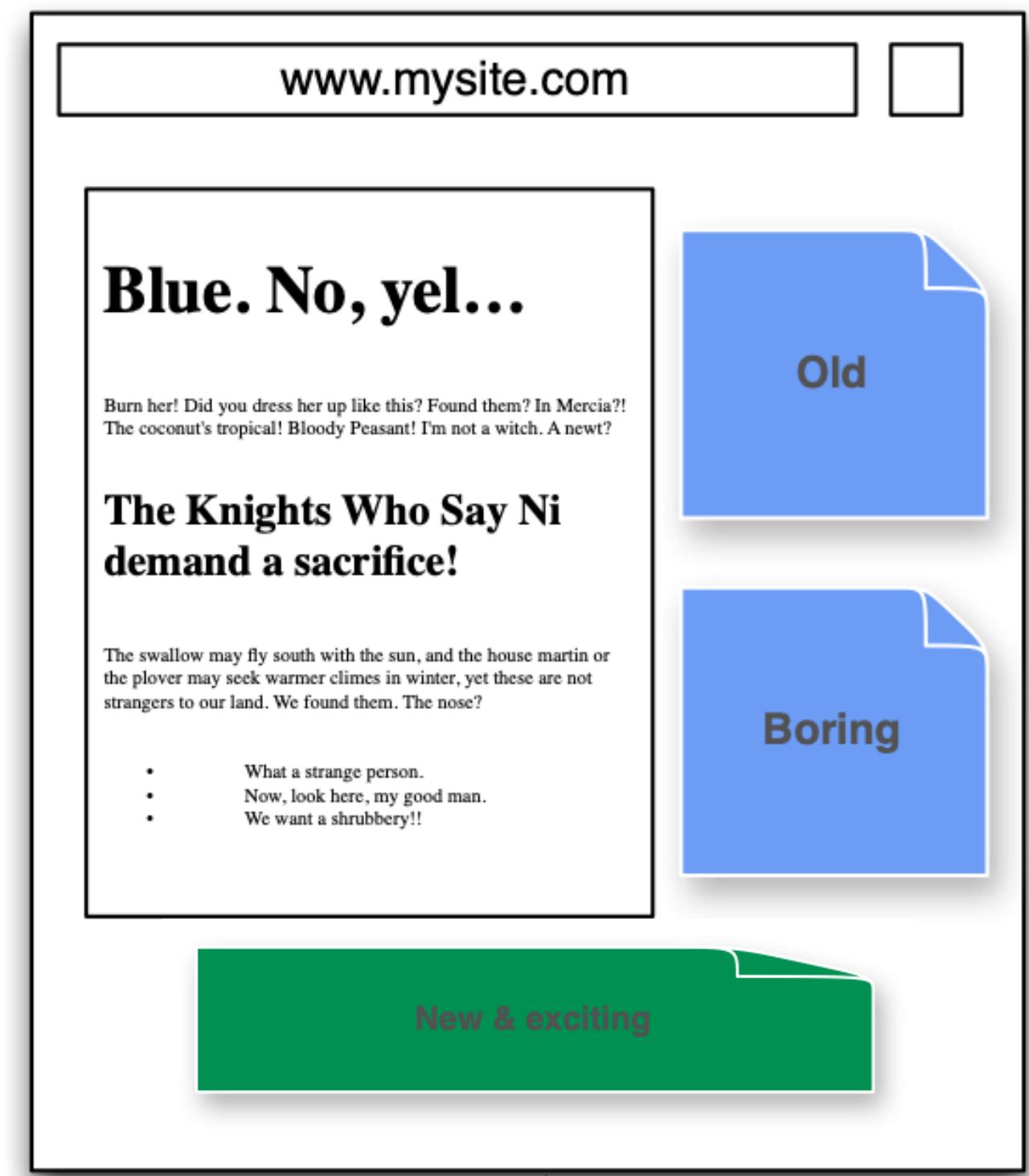
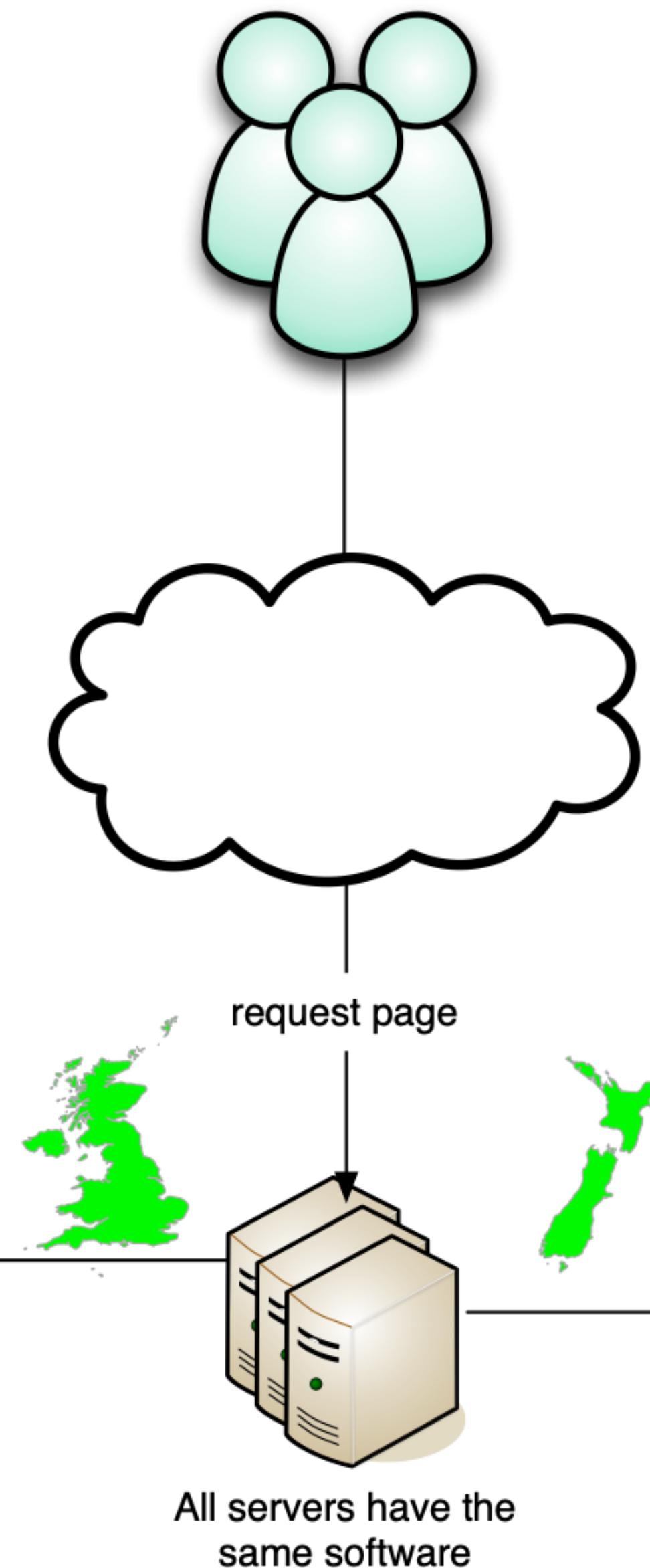
reduce risk of release



multi-variant testing



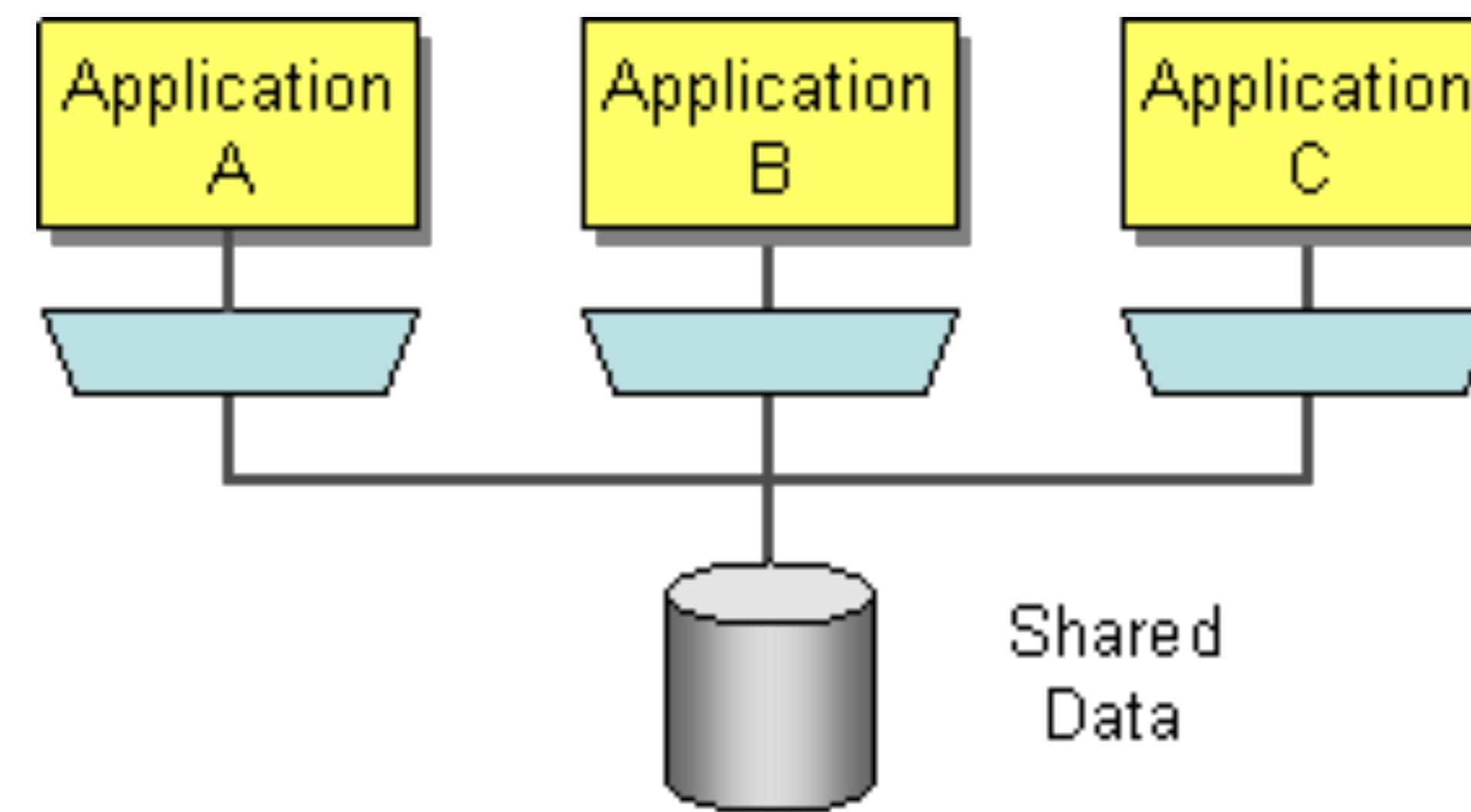
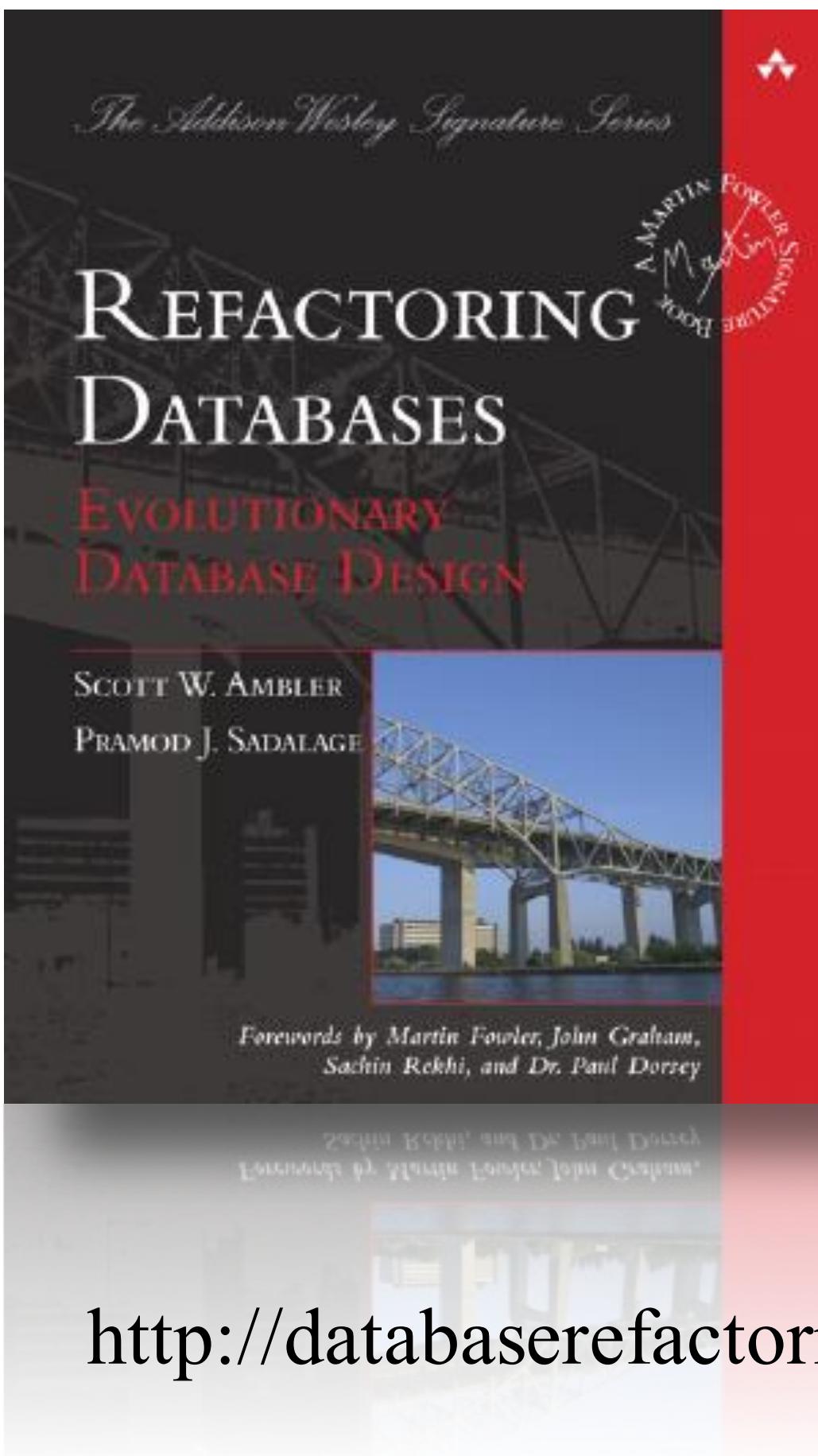
performance testing



# Toggling Databases

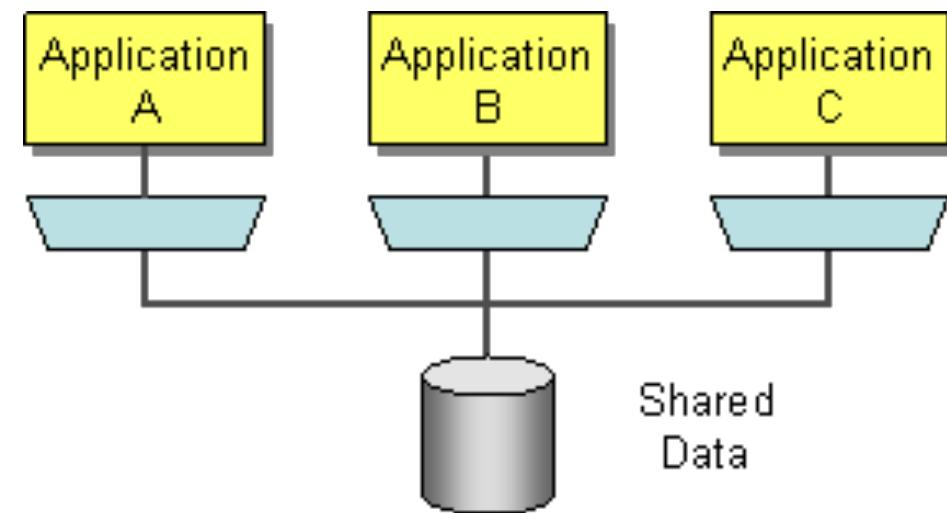
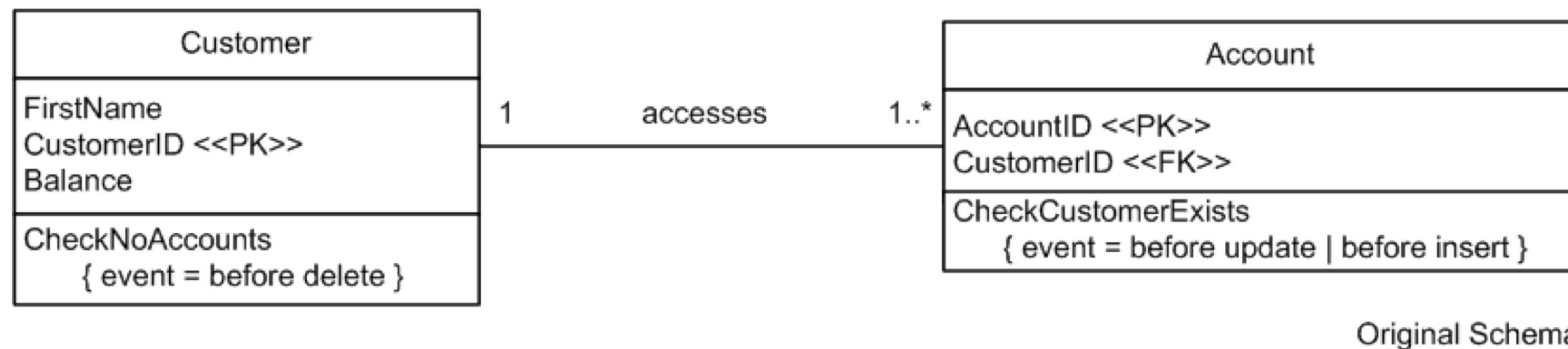


# Evolutionary Database Design

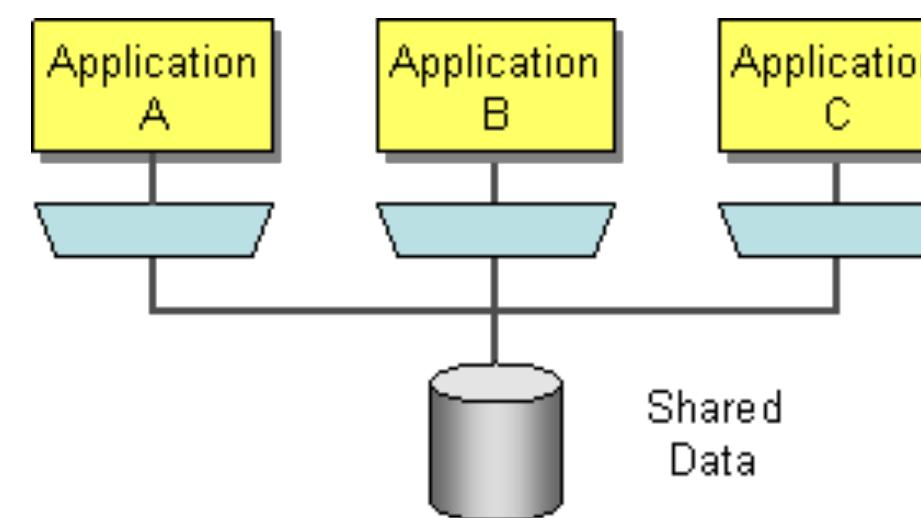
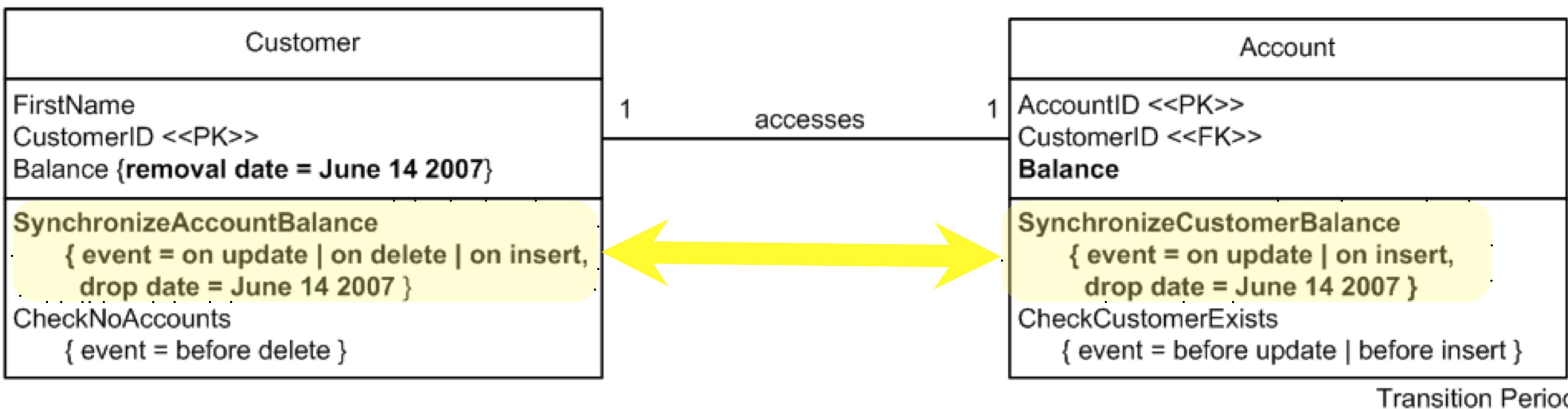


<http://databaserefactoring.com/>

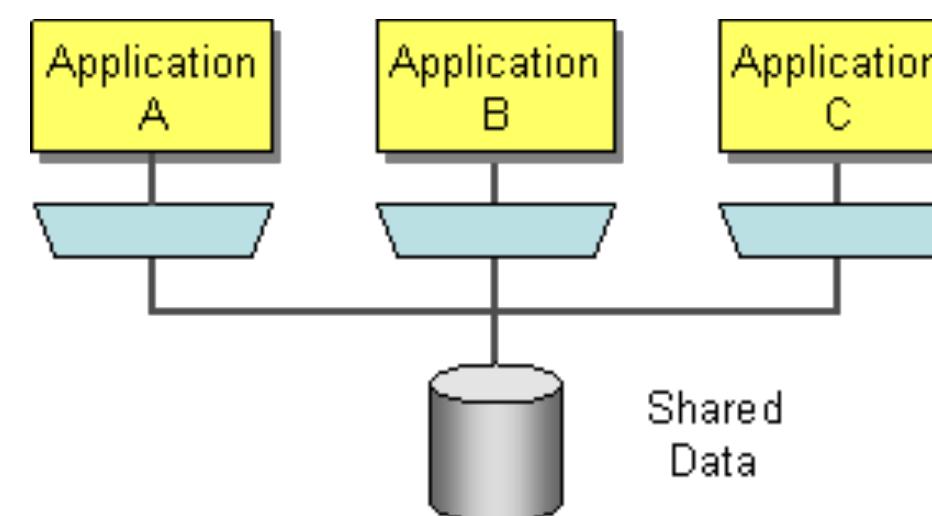
# Initial Phase



# Transition Phase



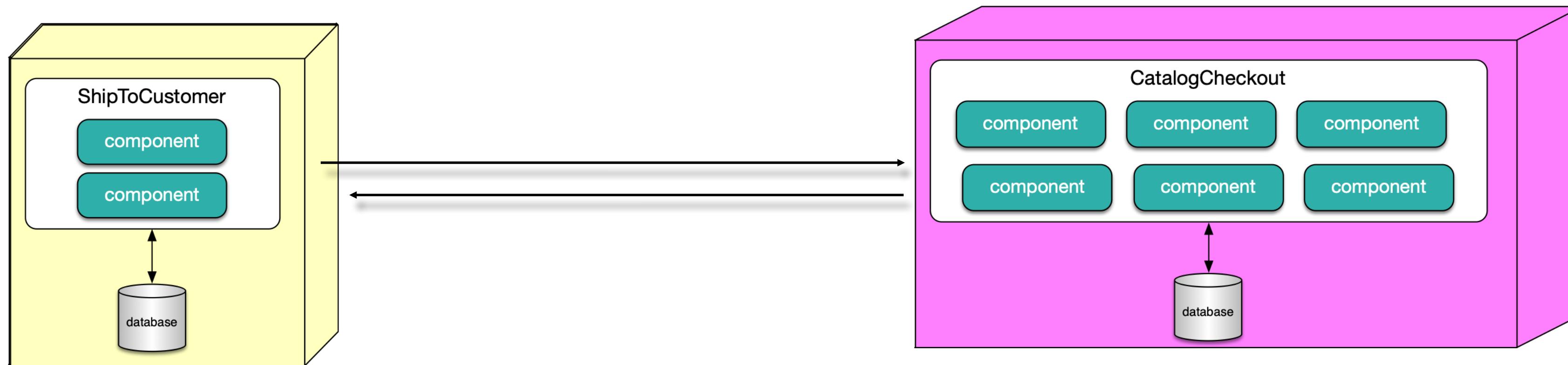
# Ending State



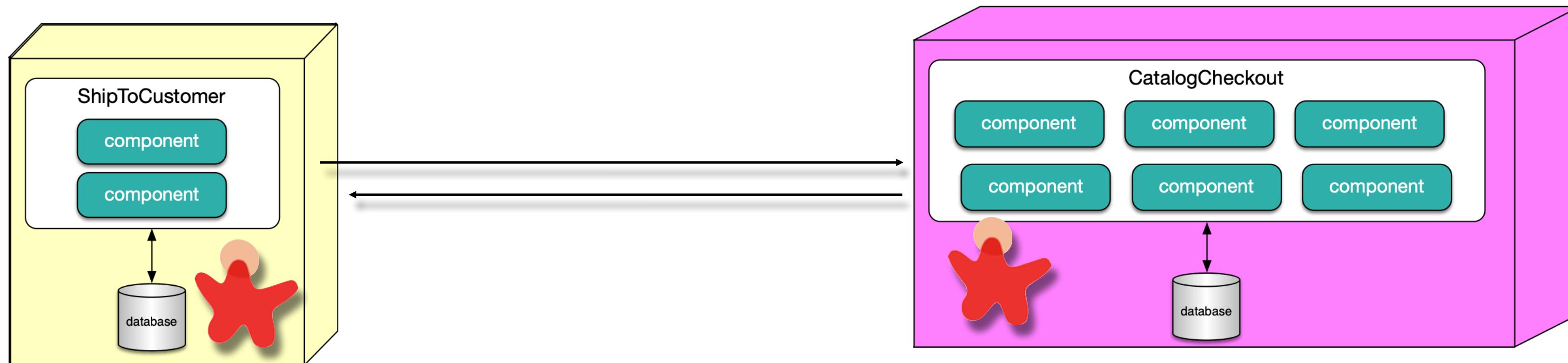
# Question:

How can architects build decoupled systems that allow continuous delivery/deployment and evolution?

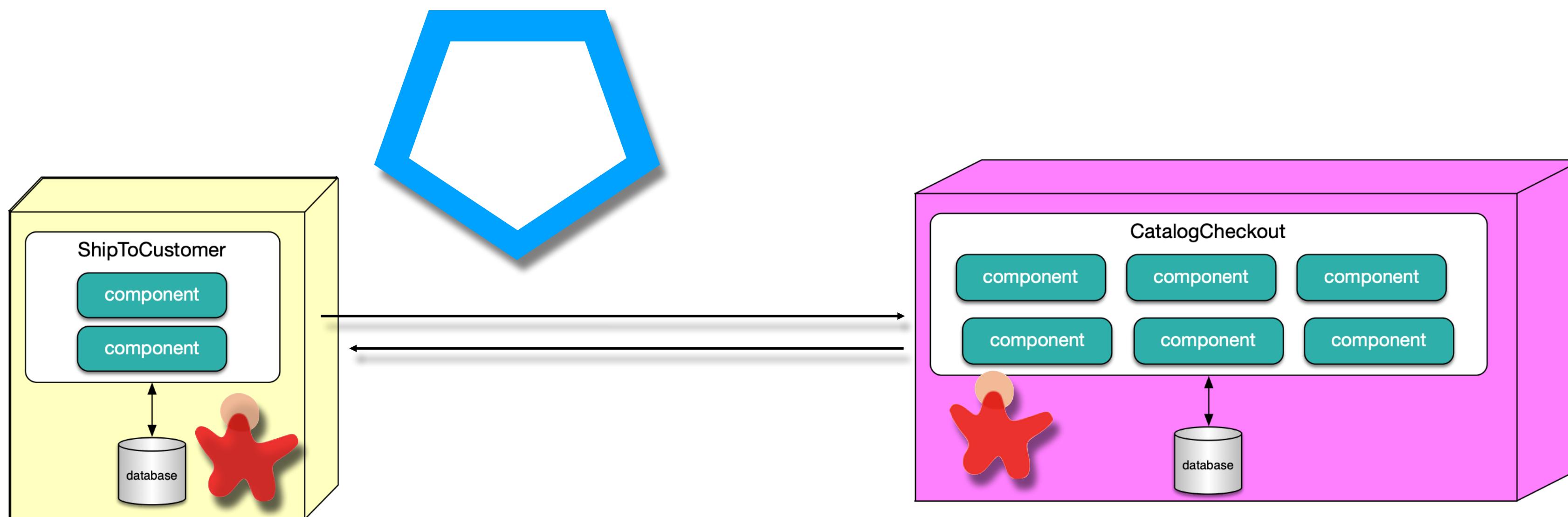
# value passing in microservices



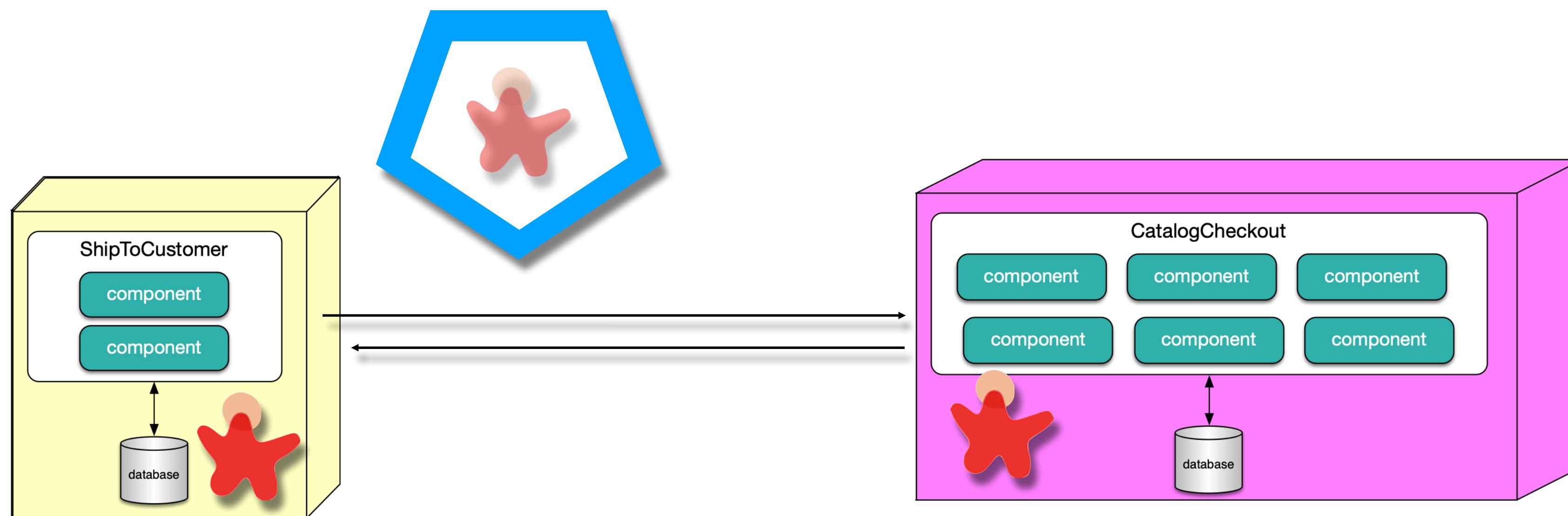
# value passing in microservices



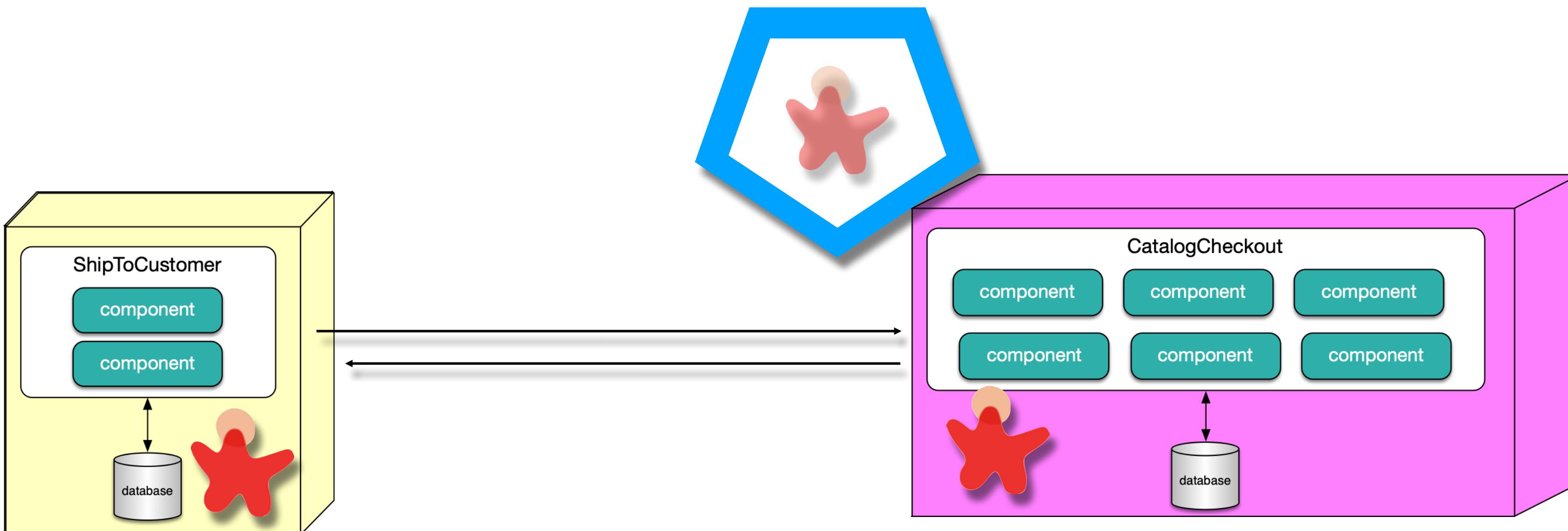
# value passing in microservices



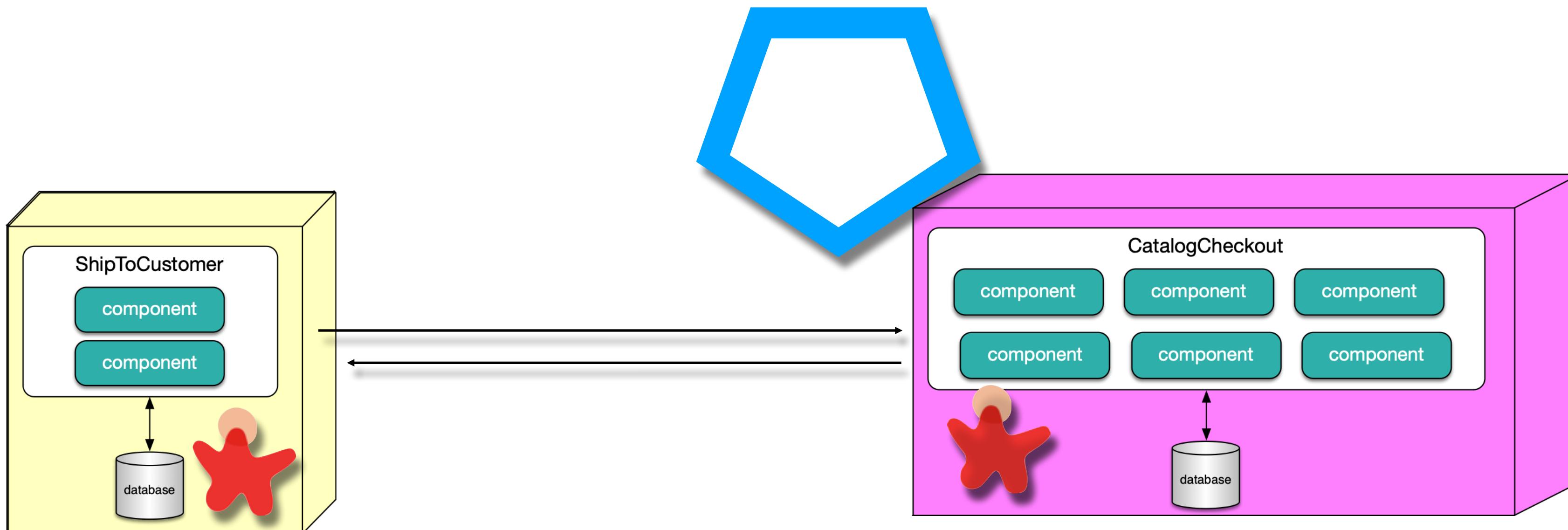
# value passing in microservices



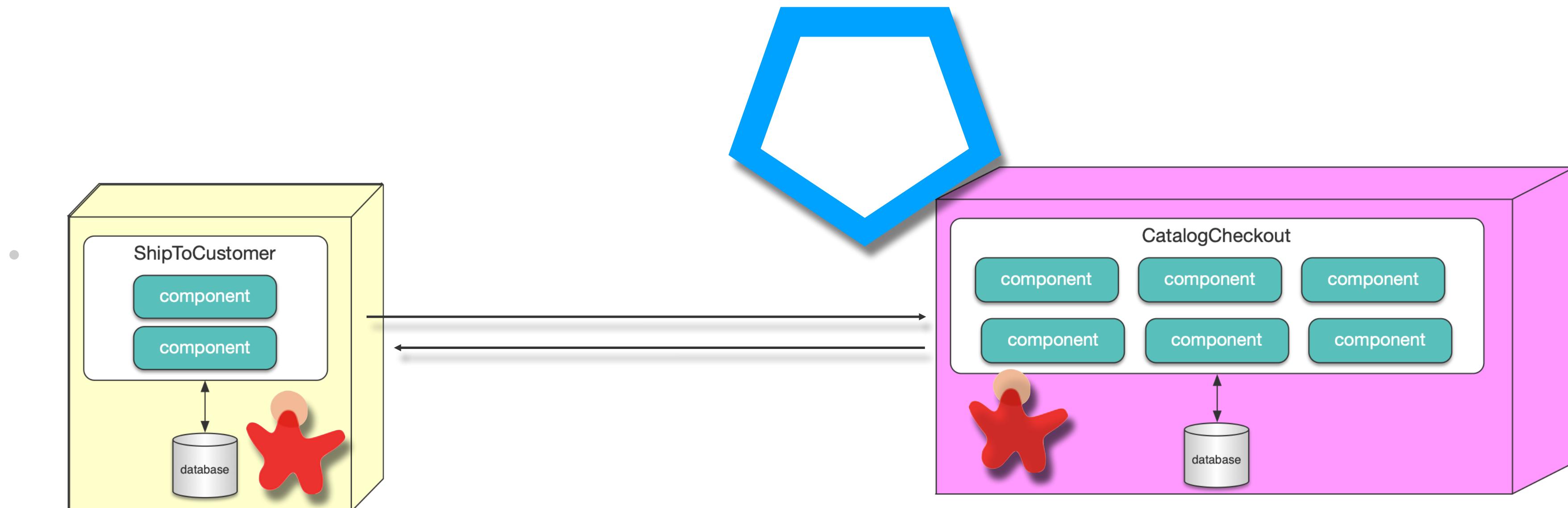
# value passing in microservices



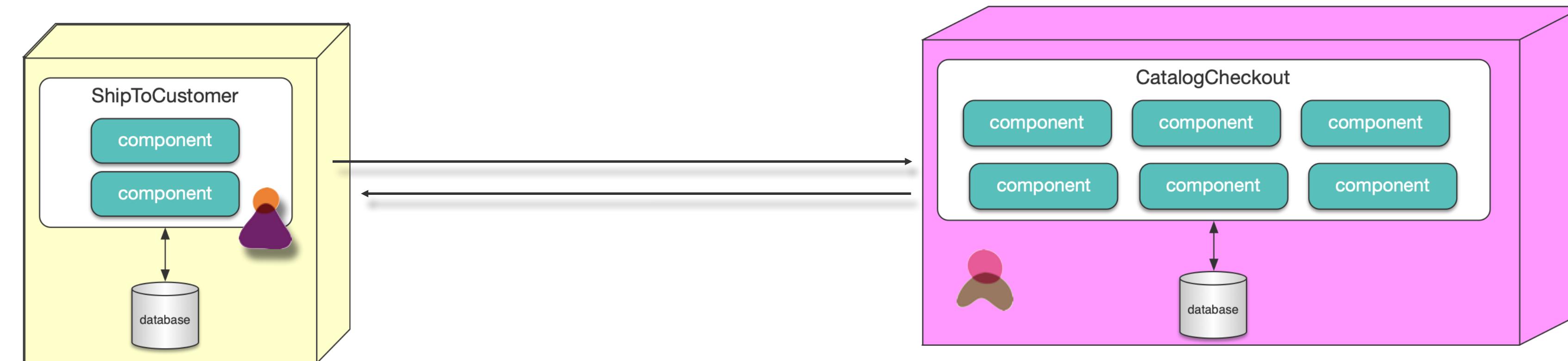
# value passing in microservices



# value passing in microservices



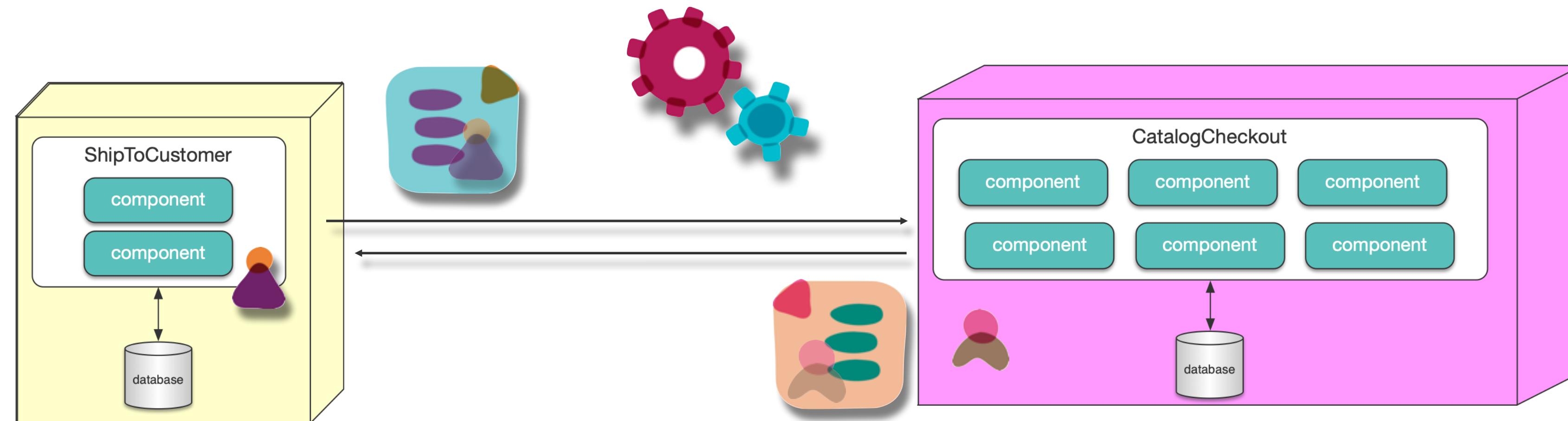
# value passing in microservices



***Transfer values, not types.***

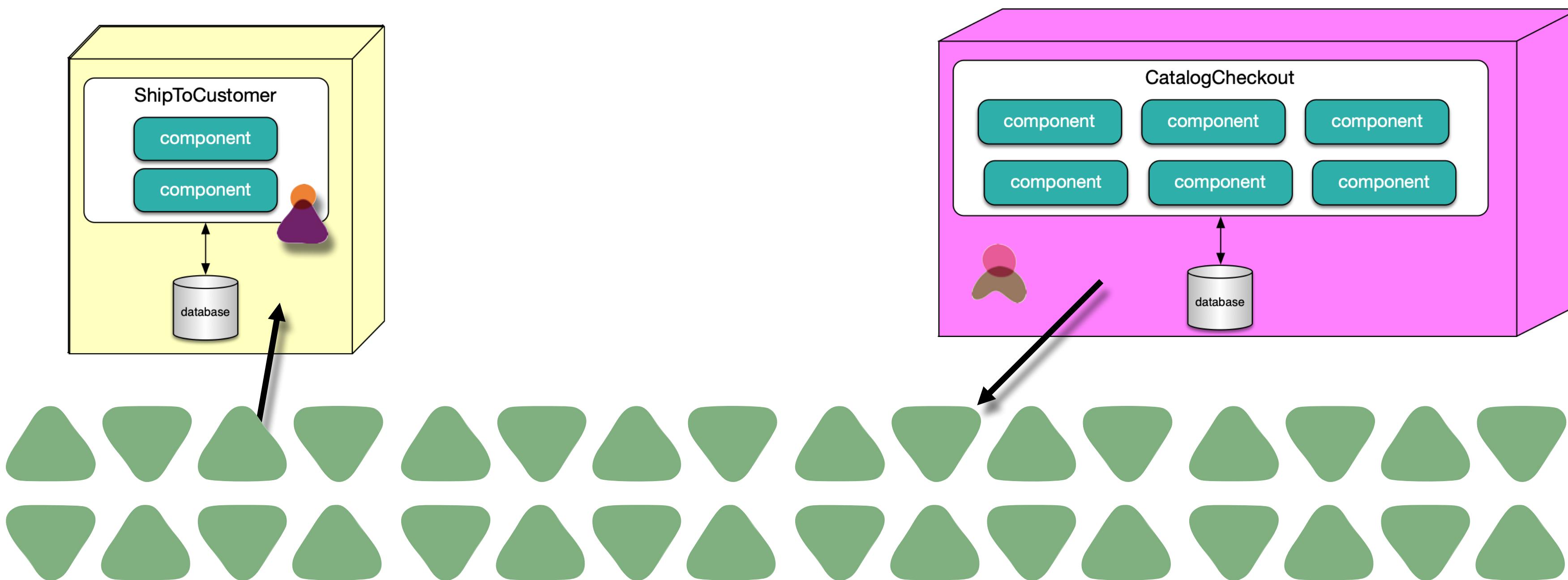
# value passing in microservices

***fitness functions protect contracts***



***Transfer values, not types.***

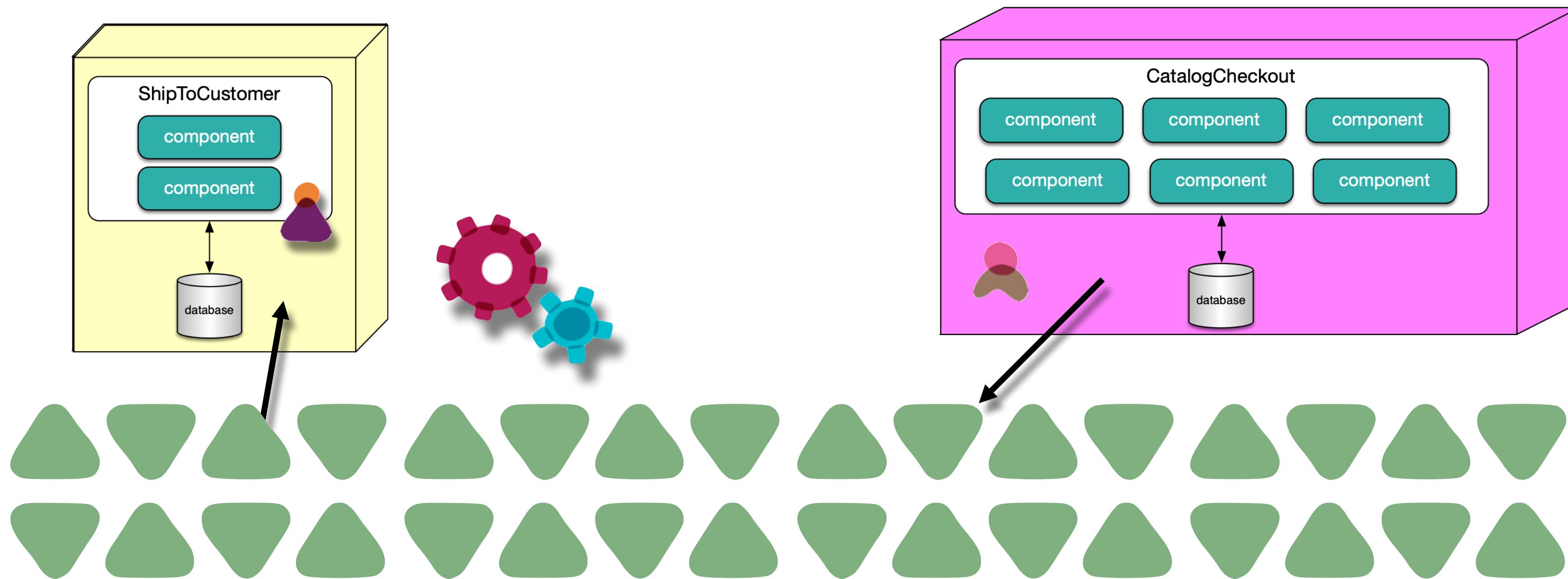
# value passing in microservices



*Transfer values, not types.*

# value passing in microservices

***fitness functions protect contracts***



***Transfer values, not types.***

# Question:

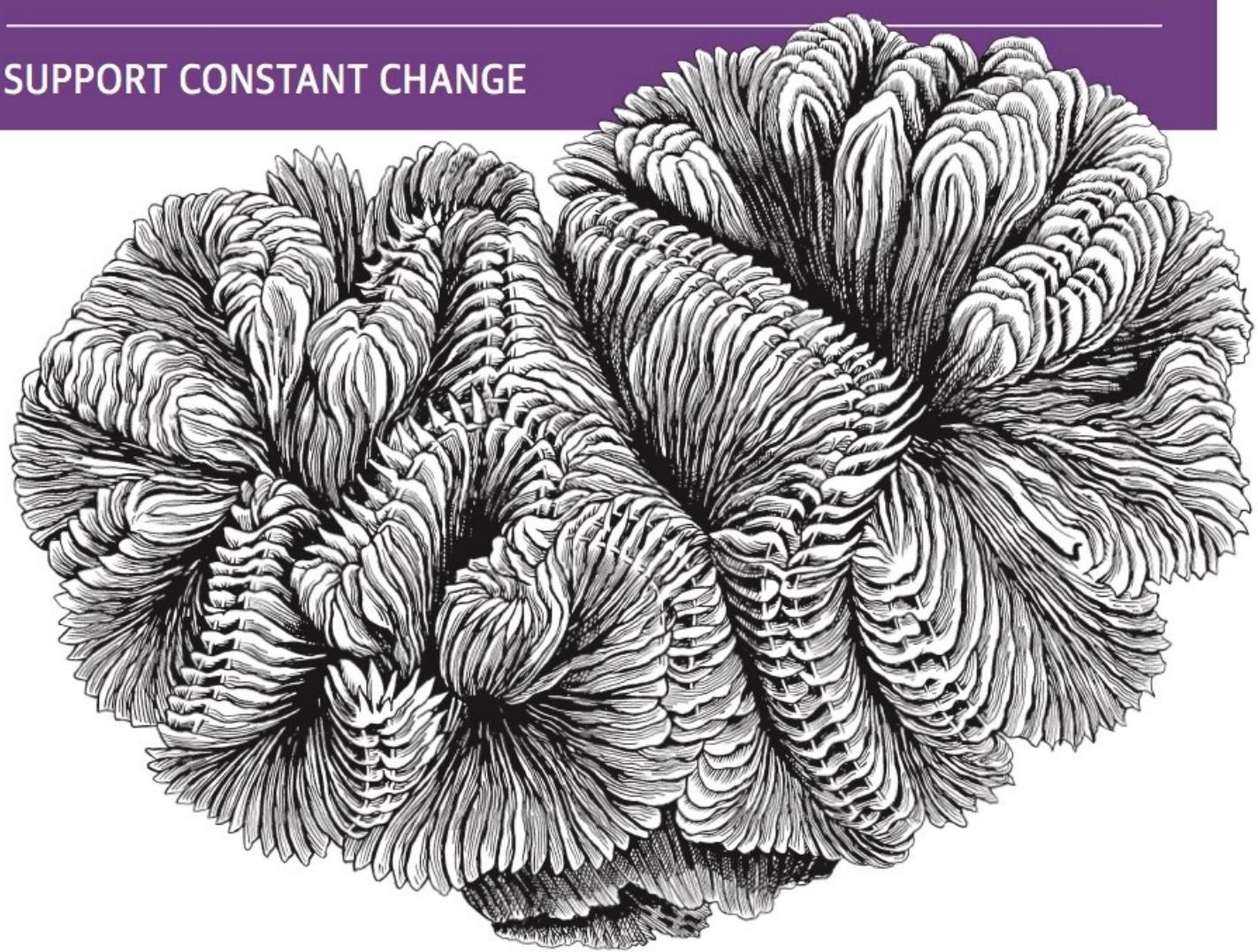
What's the most important characteristics of fitness functions?

aRChiTeCtuRe  
fOr  
cONTiNuoUs DeLiVeRy

O'REILLY®

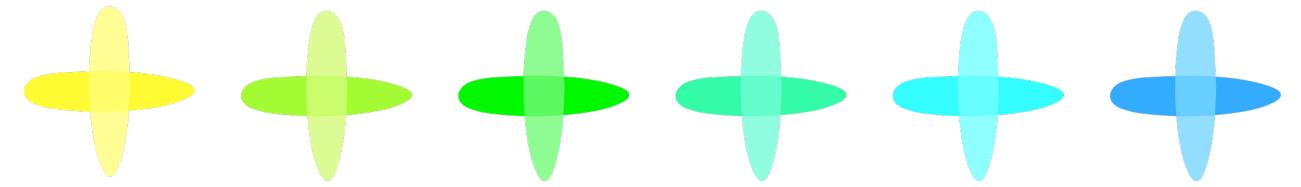
# Building Evolutionary Architectures

SUPPORT CONSTANT CHANGE

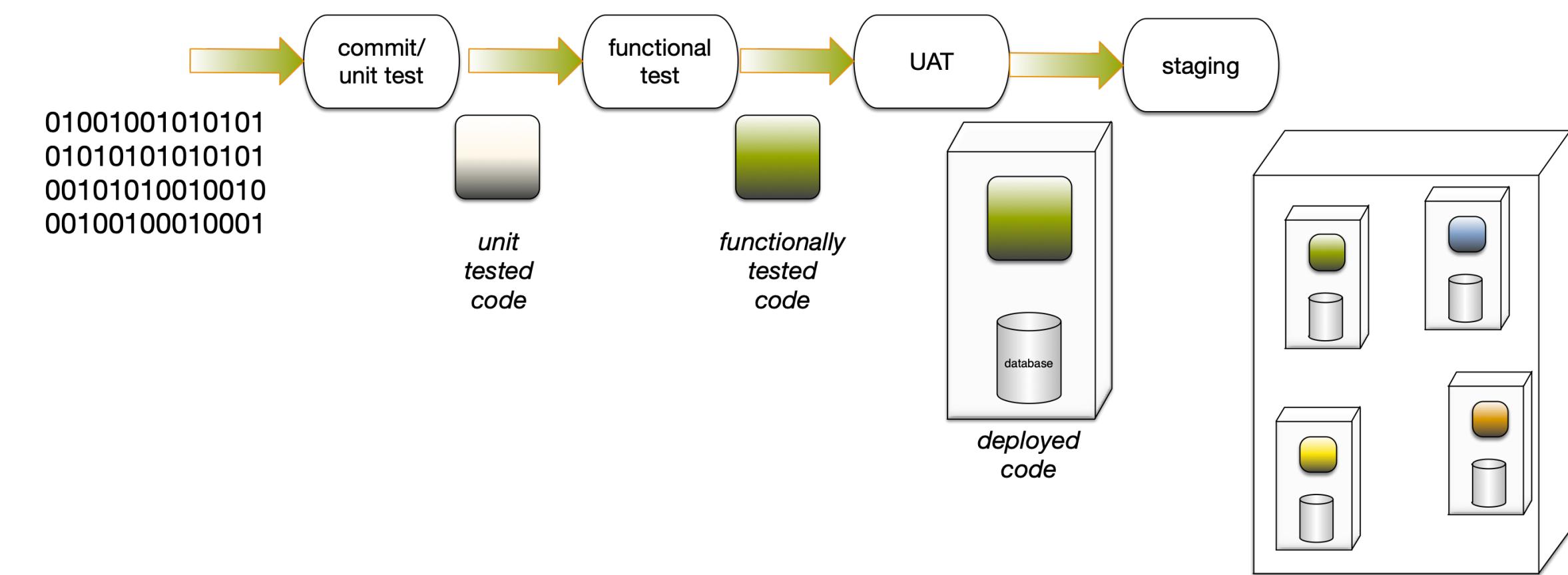
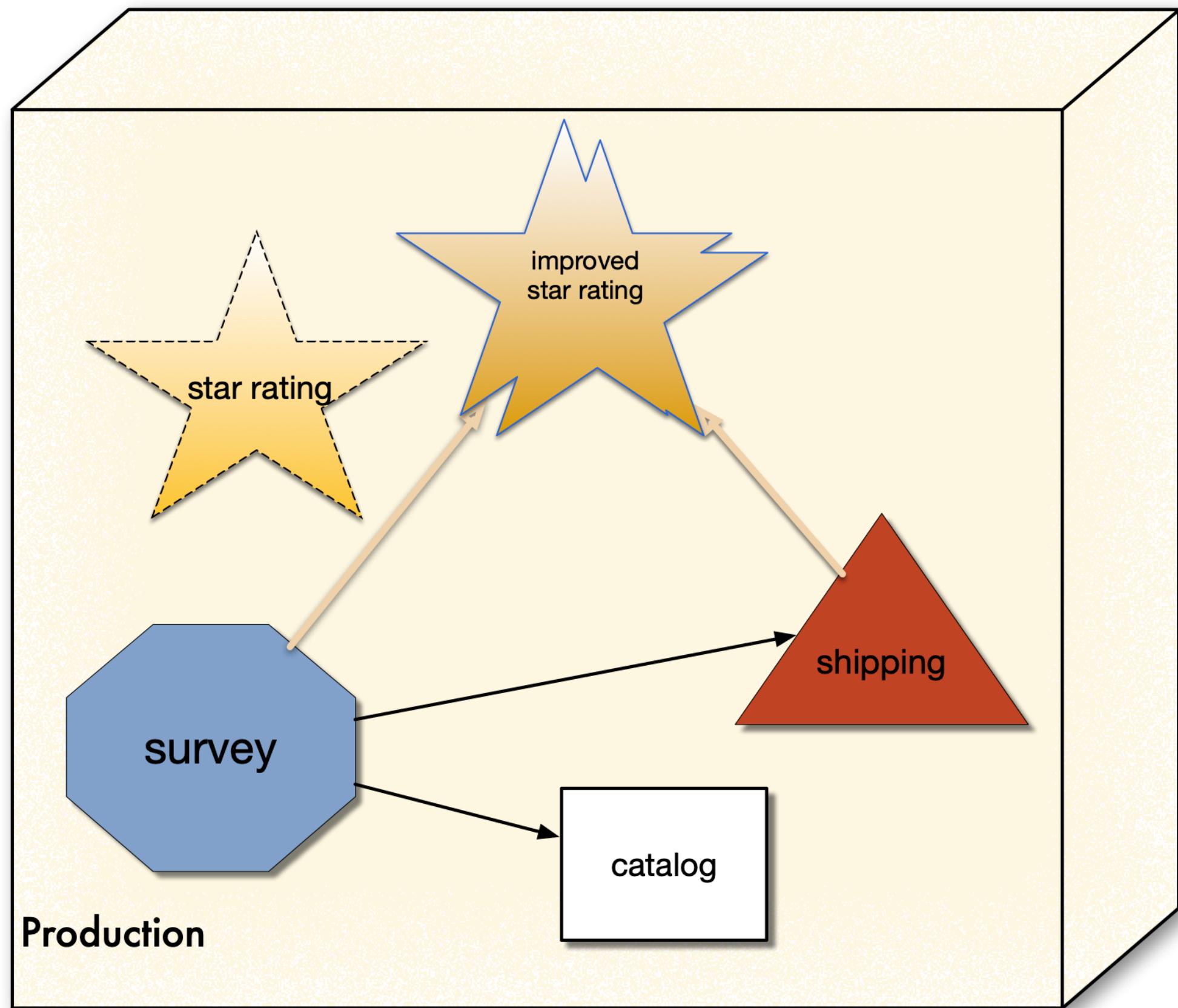


Neal Ford, Rebecca Parsons & Patrick Kua

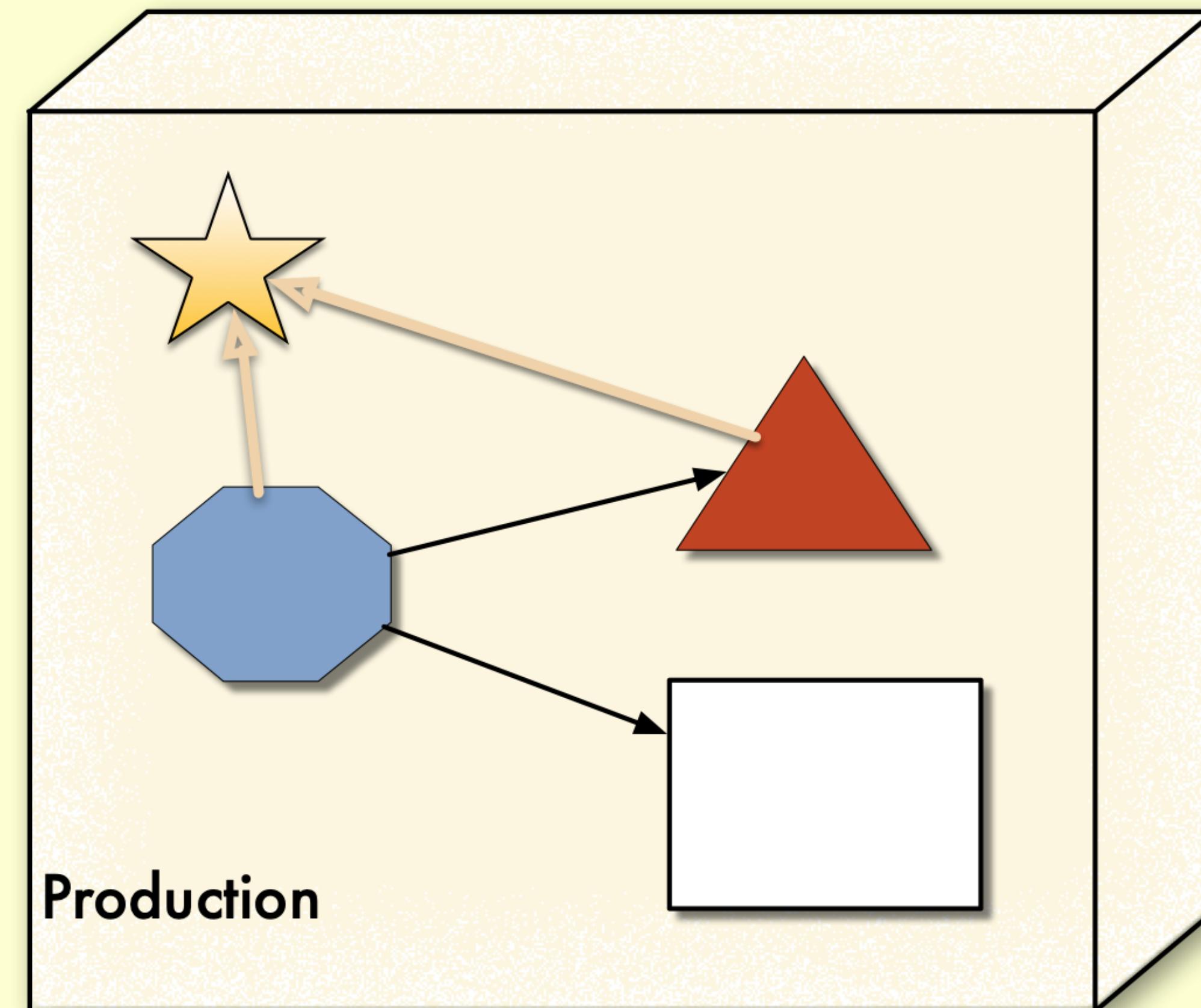
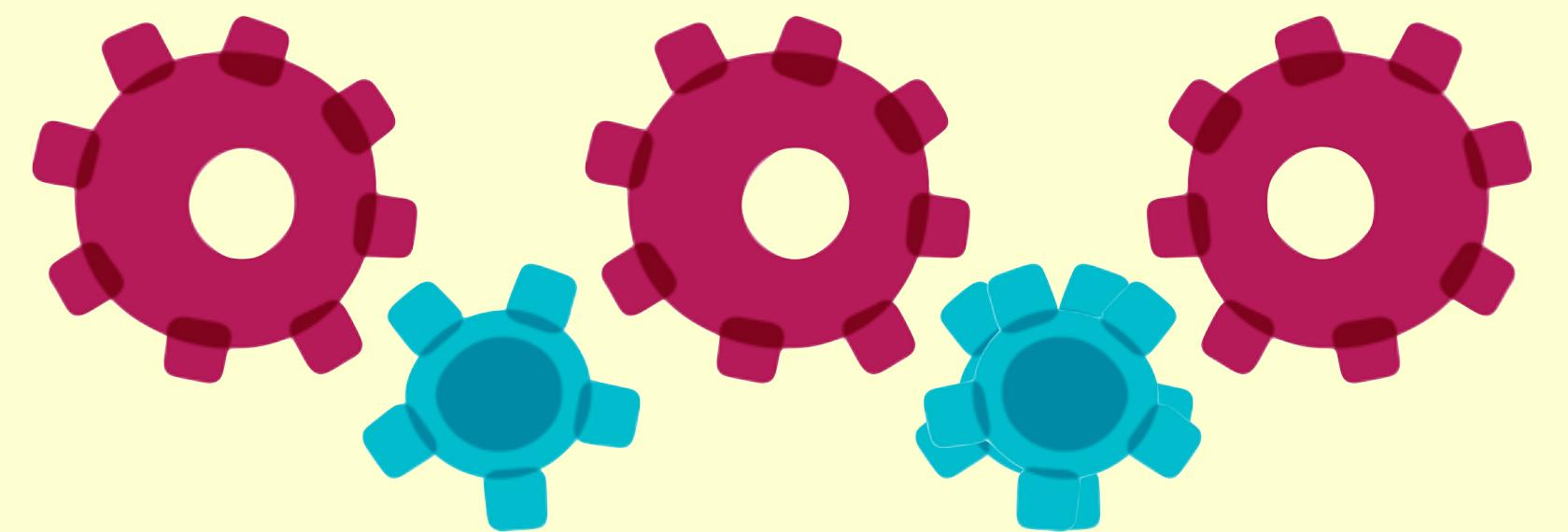




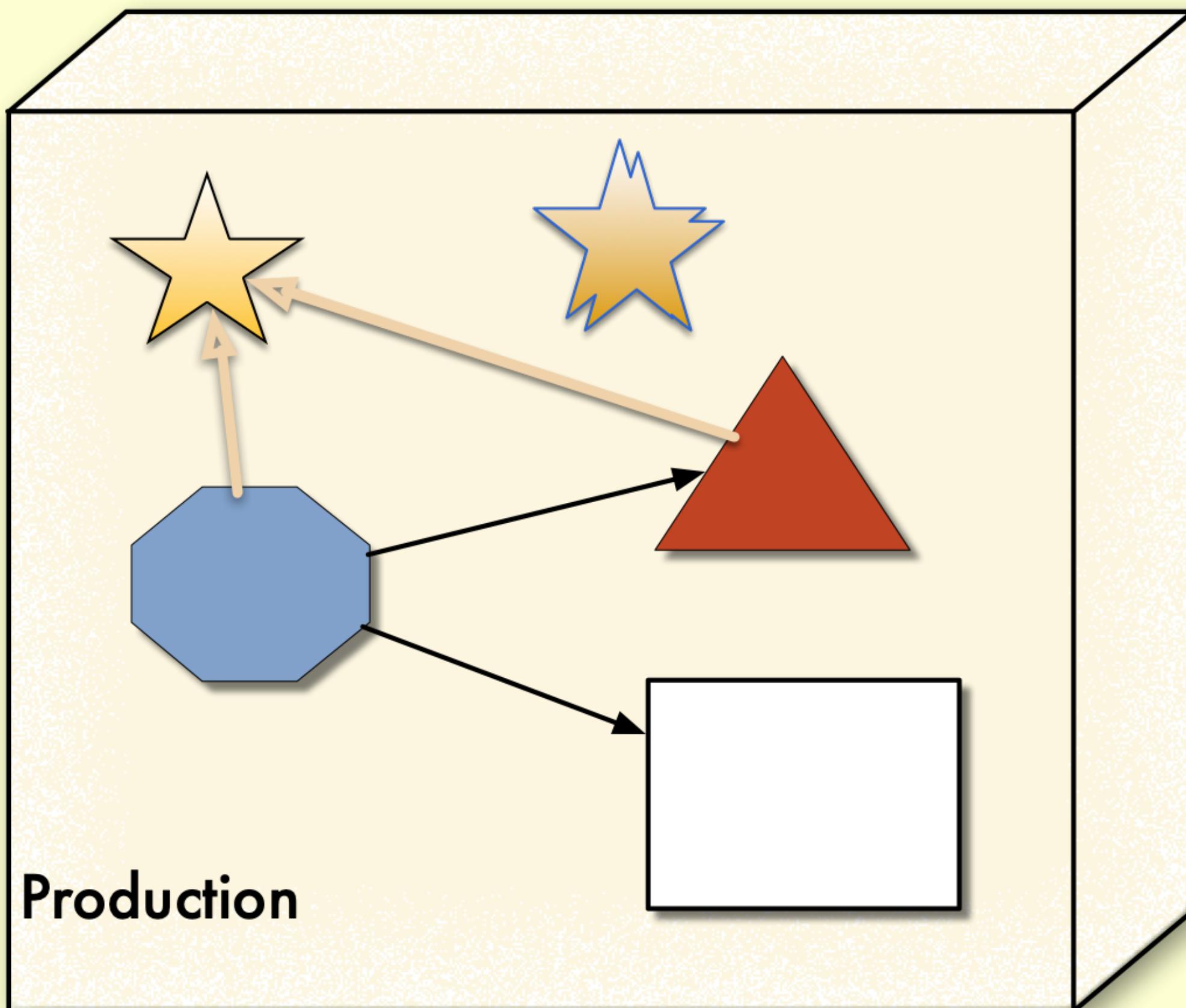
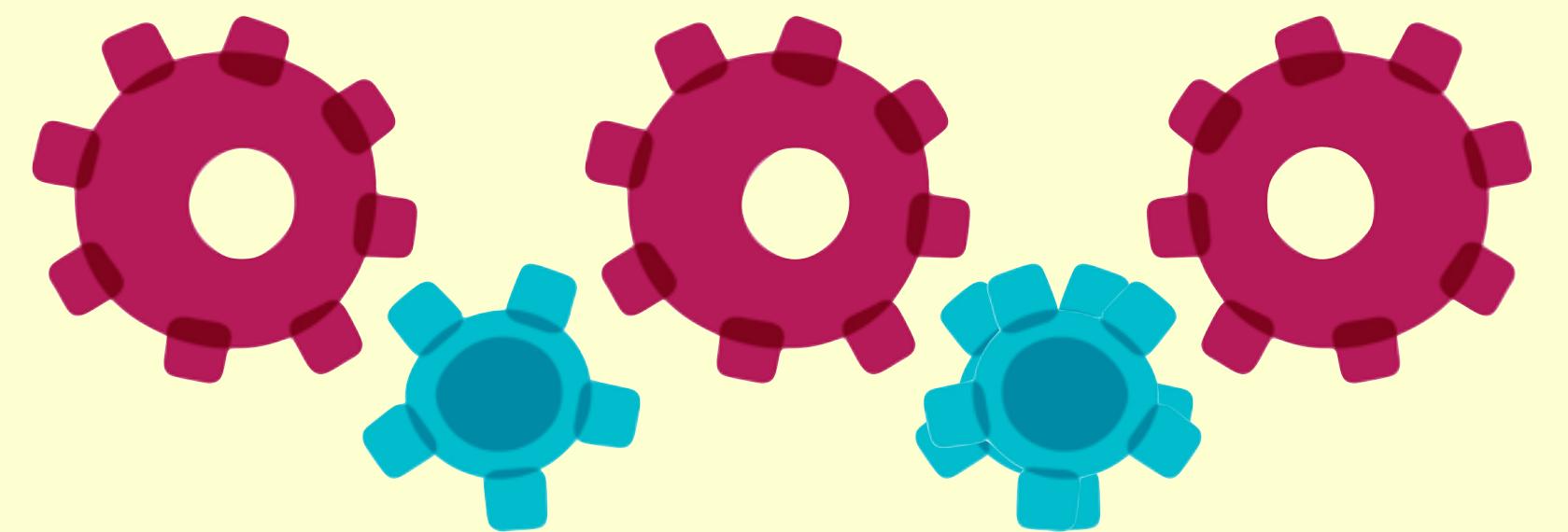
# incremental



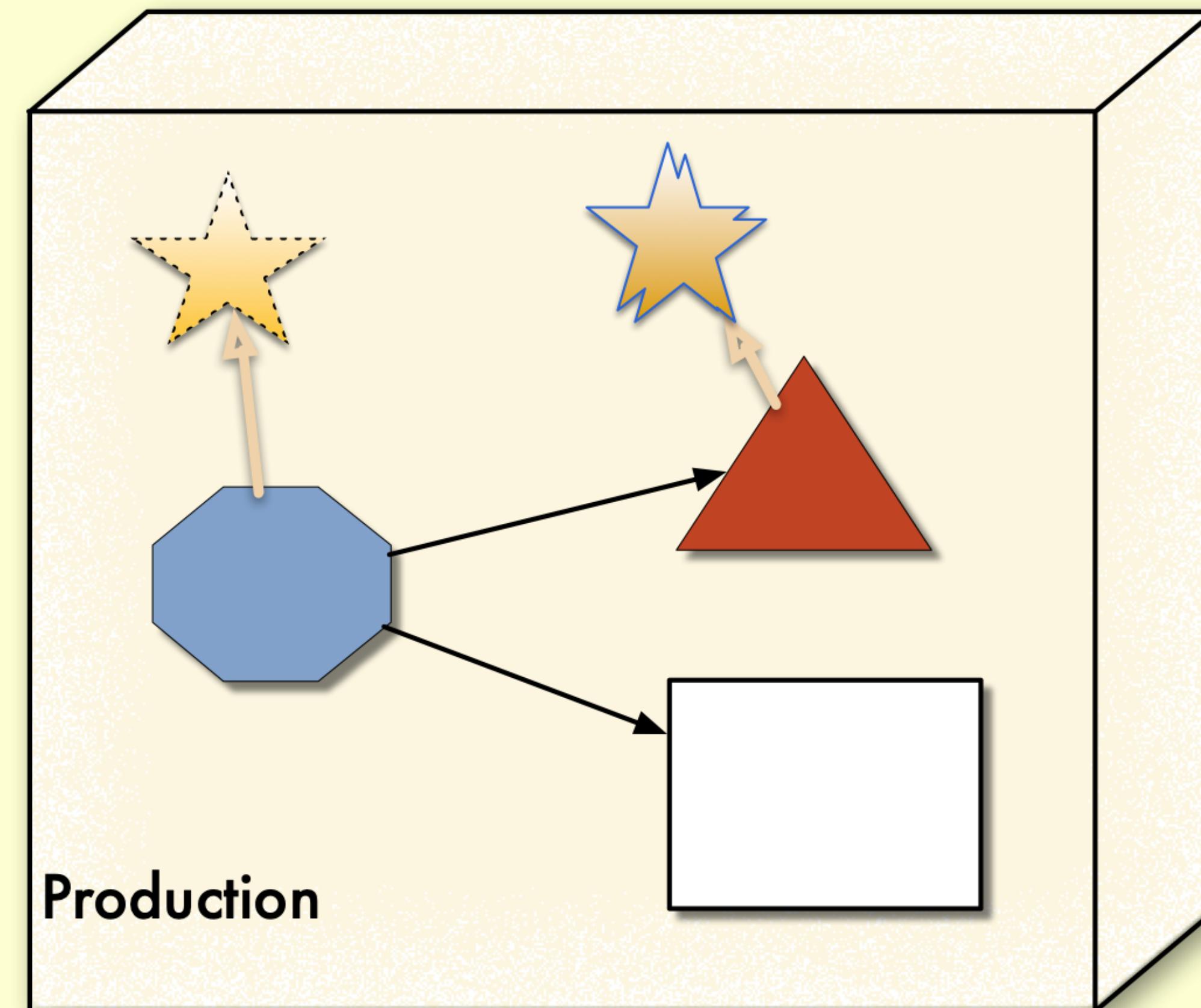
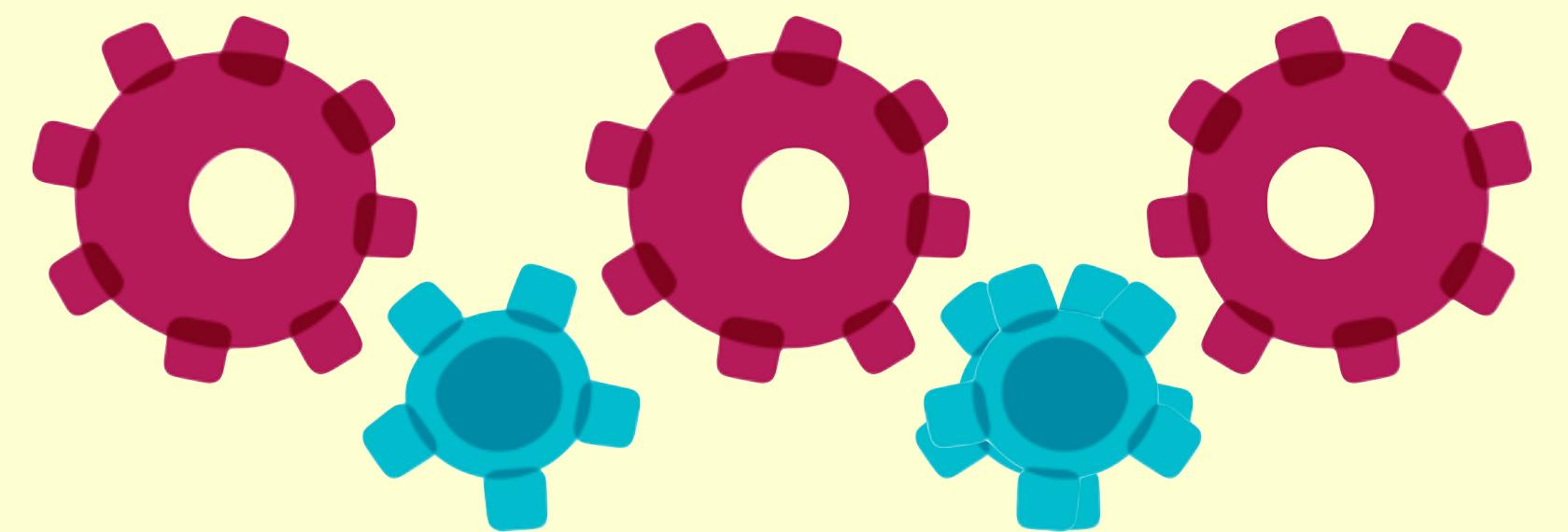
Penultima ↑ ✖ e



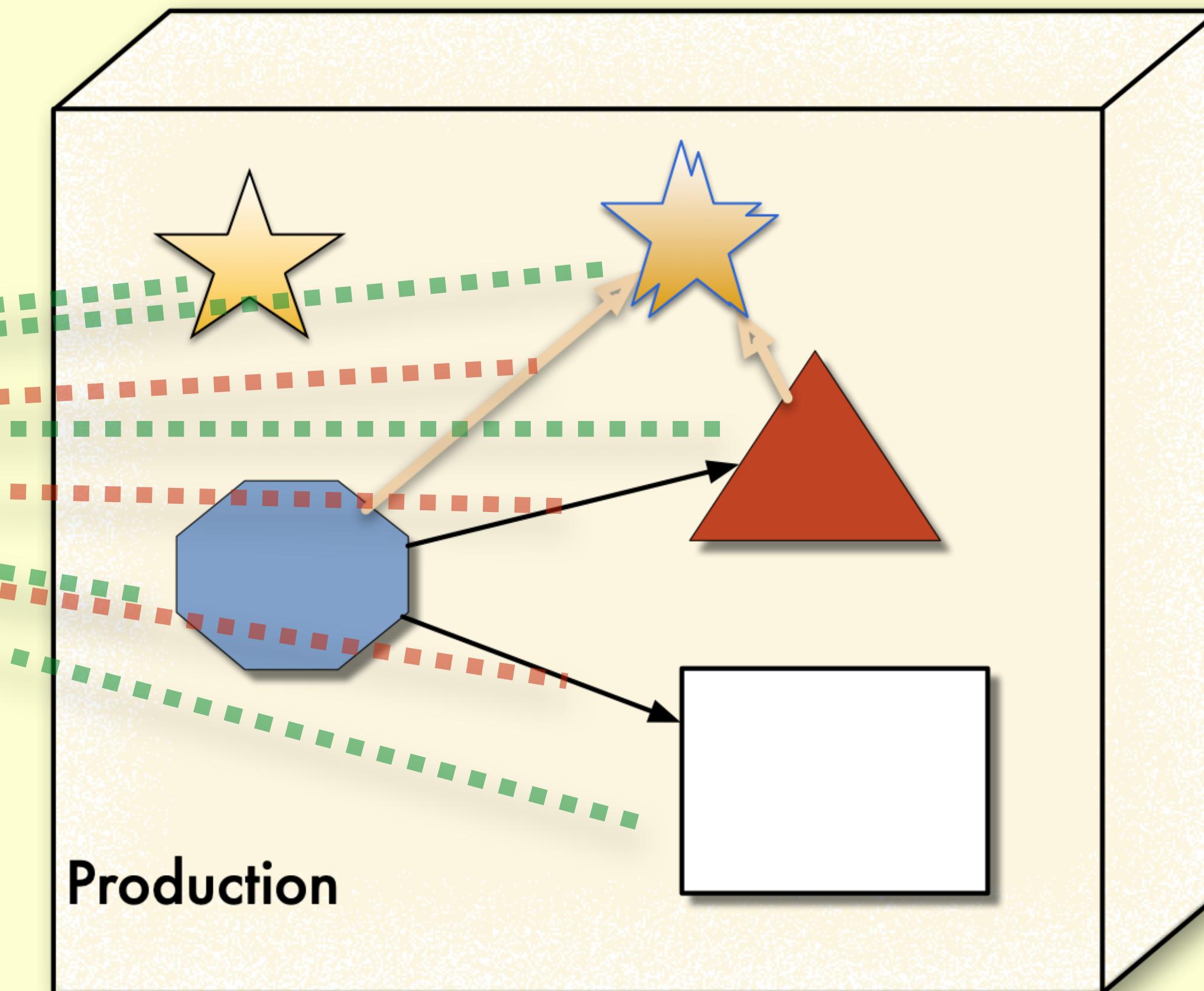
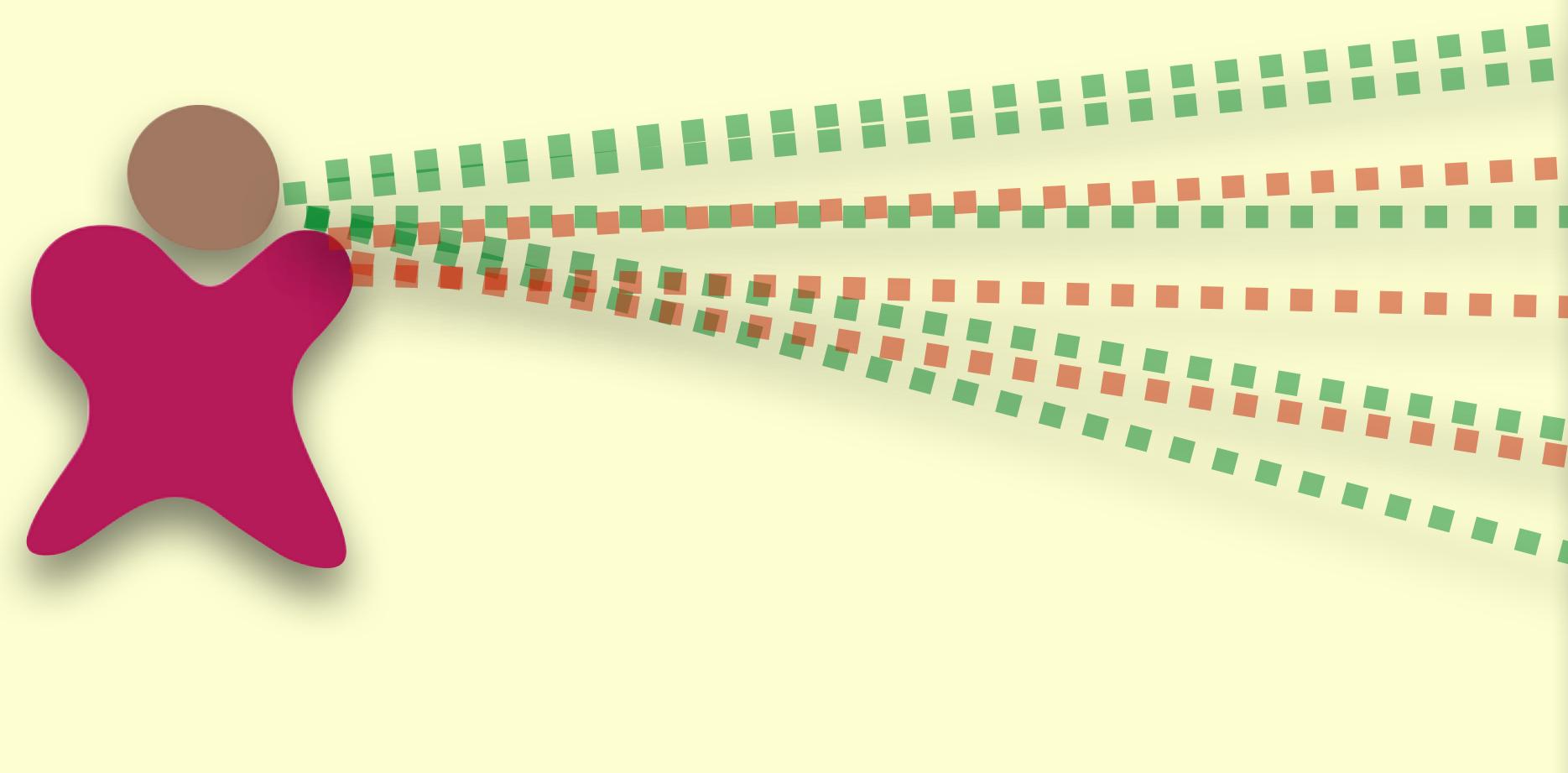
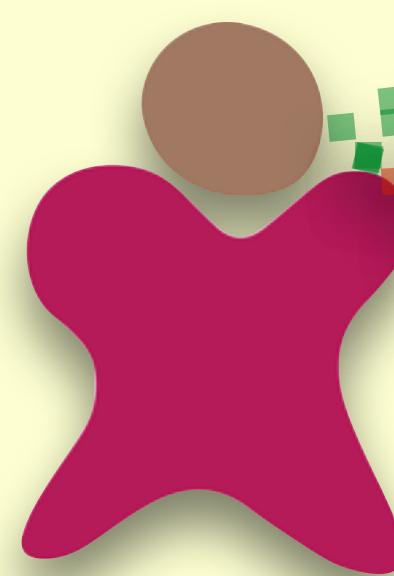
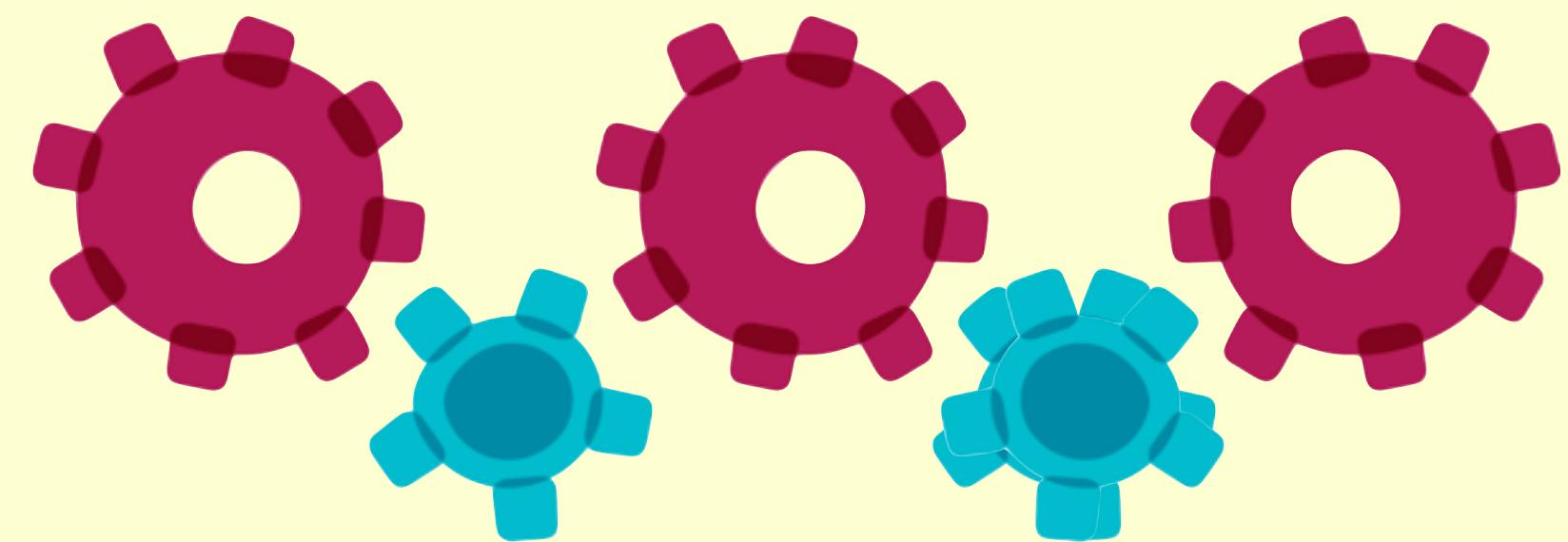
Penultima  $\uparrow \otimes e$



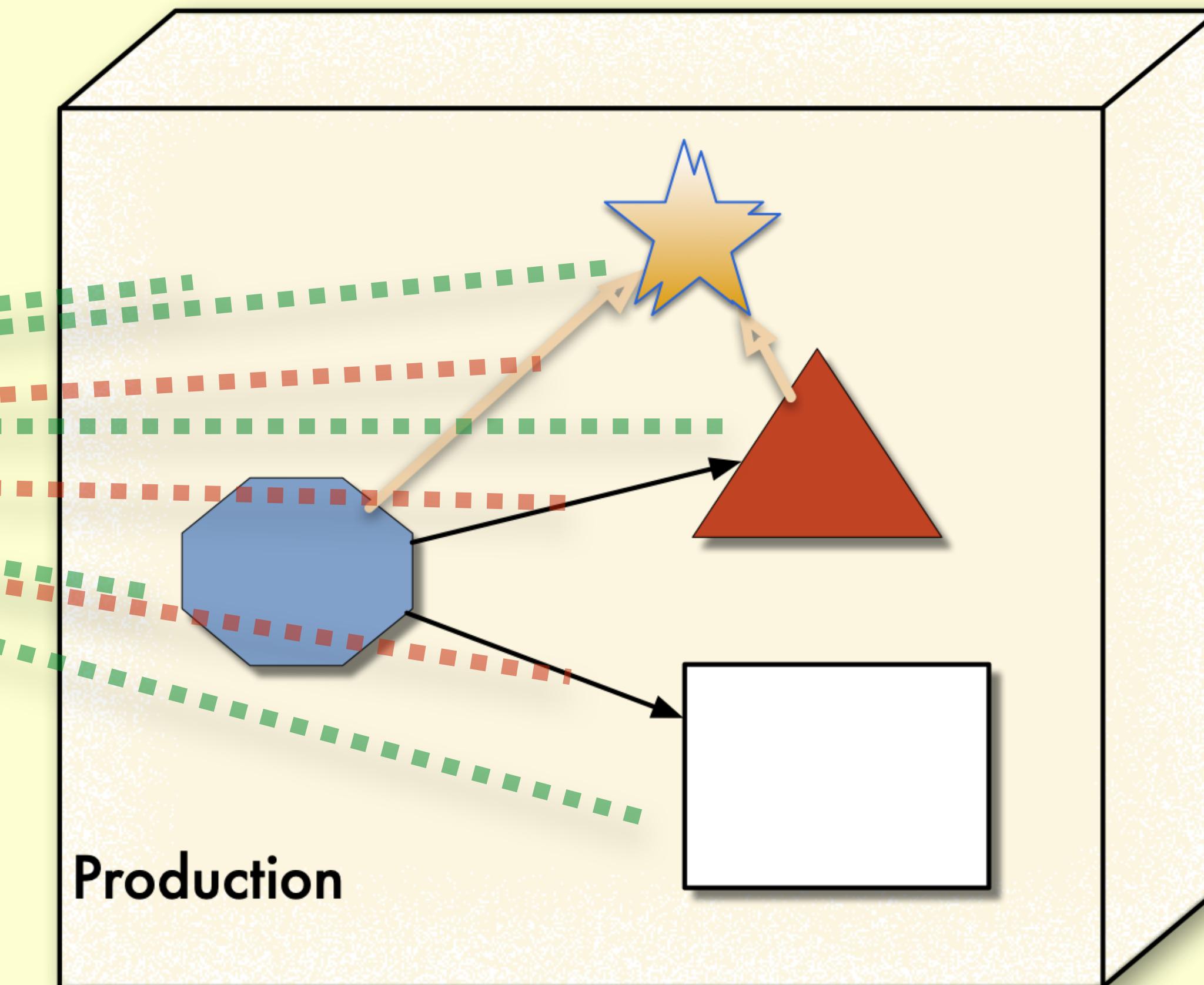
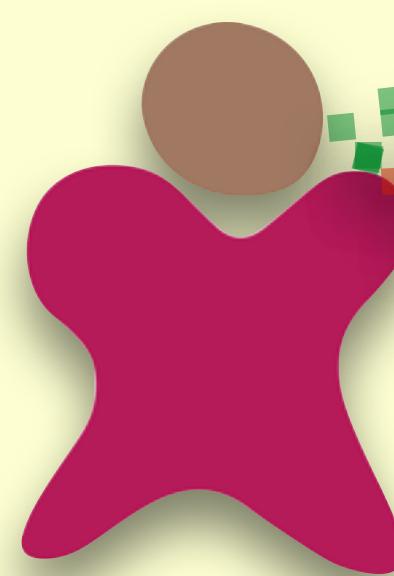
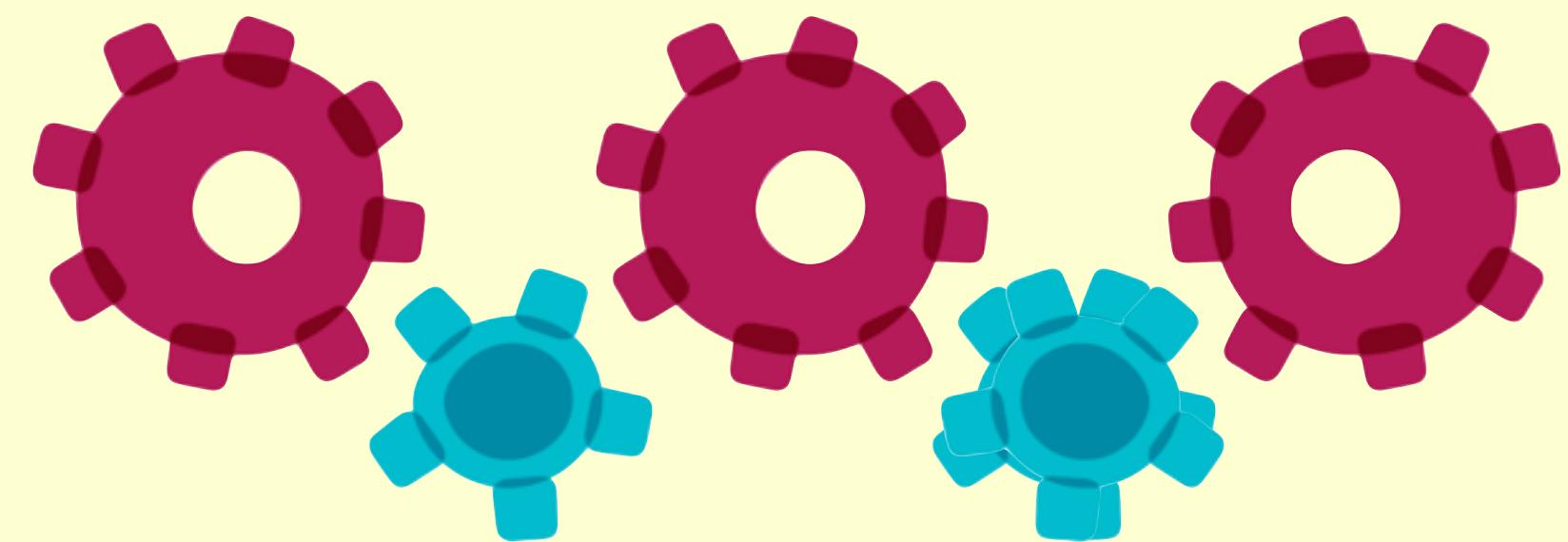
Penultima  $\uparrow \otimes e$



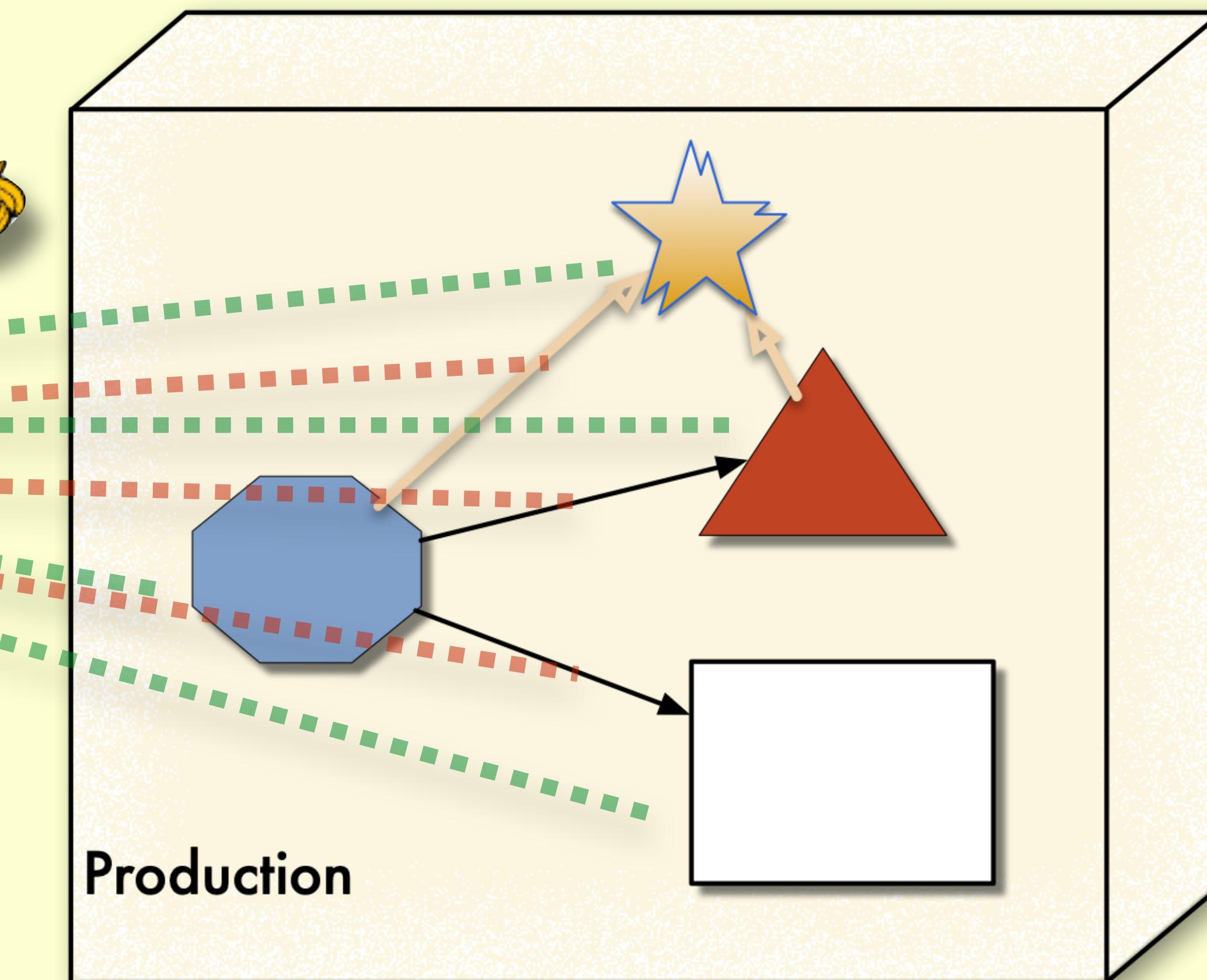
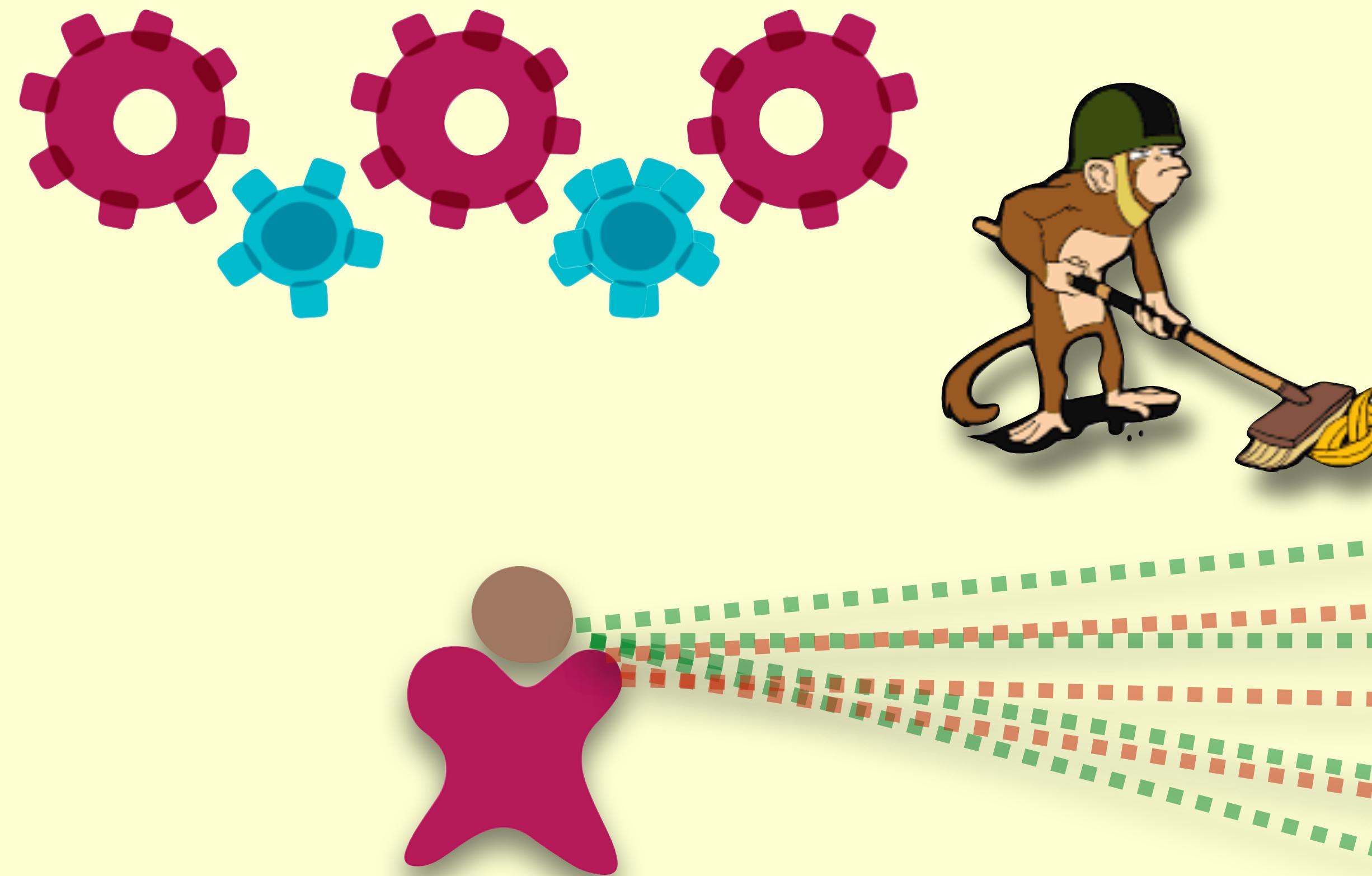
Penultima  $\uparrow \otimes e$



Penultima  $\uparrow \times e$



Penultima  $\uparrow \times e$



The screenshot shows a web browser window with the URL [githubengineering.com](http://githubengineering.com) in the address bar. The page title is "GitHub Engineering". The main heading is "Move Fast and Fix Things". Below it is a sub-heading "vmg December 15, 2015". The content discusses technical debt and GitHub's engineering standards. It includes a section on "Merges in Git" which details how GitHub handles file storage and merge strategies.

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".

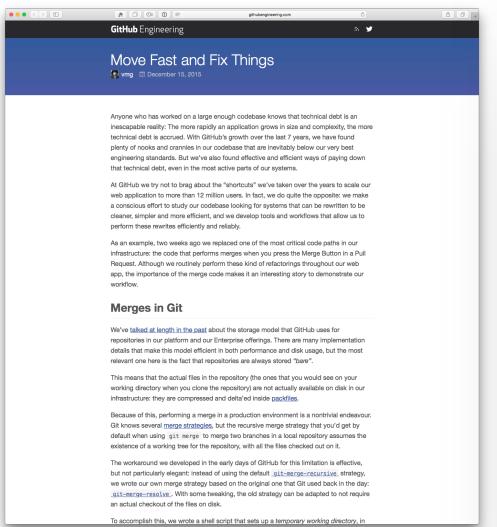
This means that the actual files in the repository (the ones that you would see on your working directory when you clone the repository) are not actually available on disk in our infrastructure: they are compressed and delta'ed inside [packfiles](#).

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.prettify_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

```

**github / scientist**

Code Issues 4 Pull requests 4 Projects 0 Pulse Graphs

A Ruby library for carefully refactoring critical paths.

71 commits 1 branch 6 releases 16 contributors MIT

ibarnette committed on GitHub Merge pull request #58 from dmcinnes/accept-false-results ... Latest commit 48655df on Oct 25

lib accept and clean "false" results a month ago  
script 2 years ago  
test test for returning false boolean results a month ago  
.gitignore 2 years ago  
.travis.yml Bump CI Ruby patchlevels, add Ruby 2.0 11 months ago  
Gemfile 2 years ago  
LICENSE.txt 2 years ago  
README.md Merge pull request #55 from hykw/link2elixir a month ago  
scientist.gemspec Merge pull request #31 from github/one-dot-oh-dot-oh 10 months ago

Find file Clone or download

**README.md**

## Scientist!

A Ruby library for carefully refactoring critical paths. [build passing](#)

### How do I science?

Let's pretend you're changing the way you handle permissions in a large web app. Tests can help guide your refactoring, but you really want to compare the current and refactored behaviors under load.

```

require "scientist"

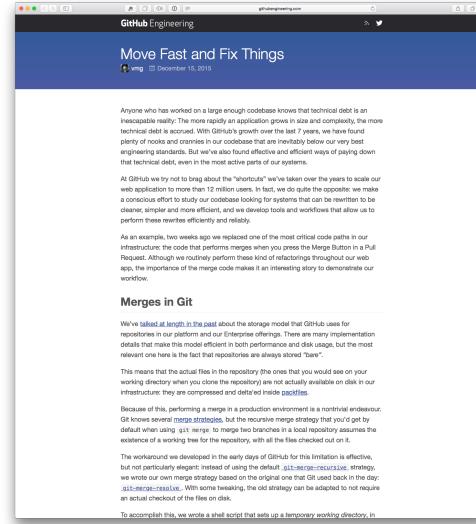
class MyWidget
  def allows?(user)
    experiment = Scientist::Default.new "widget-permissions"
    experiment.use { model.check_user?(user).valid? } # old way
    experiment.try { user.can?(:read, model) } # new way

    experiment.run
  end
end

```

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

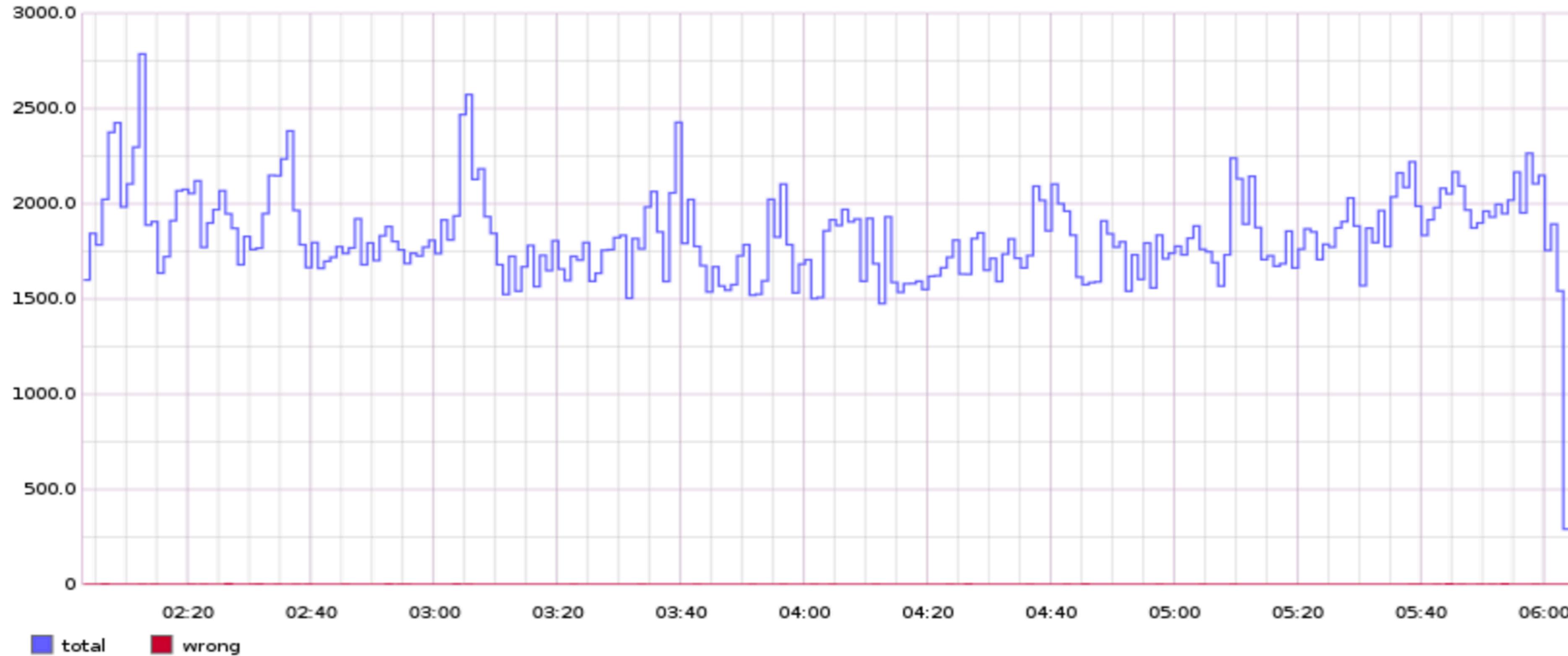
    experiment.run
  end
end
```

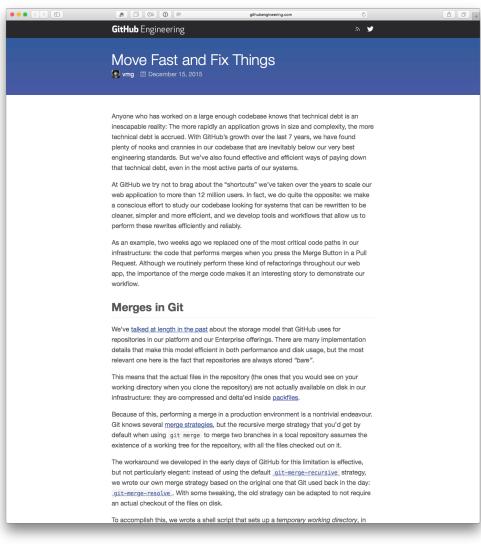
- It decides whether or not to run the try block,
- Randomizes the order in which use and try blocks are run,
- Measures the durations of all behaviors,
- Compares the result of try to the result of use,
- Swallows (but records) any exceptions raised in the try block
- Publishes all this information.



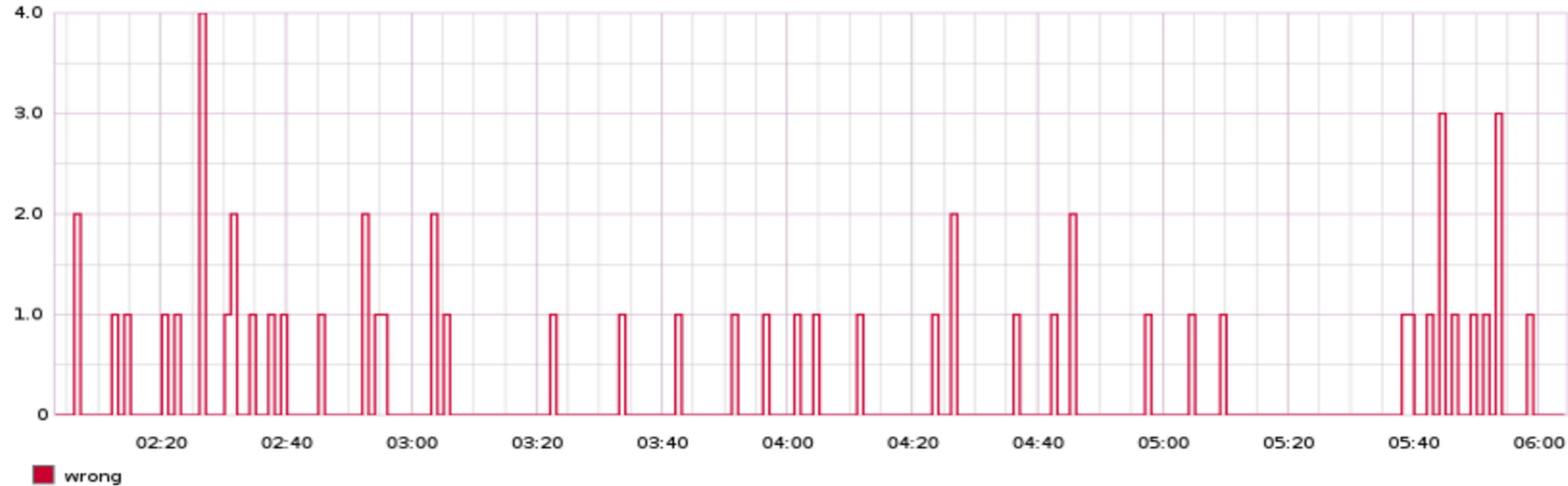
## Accuracy

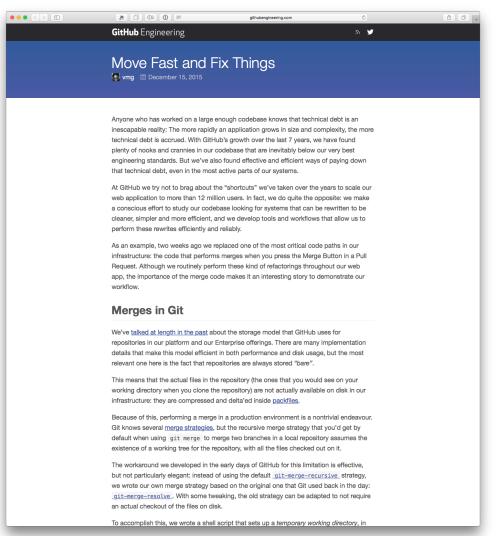
The number of times that the candidate and the control agree or disagree. [View mismatches](#)



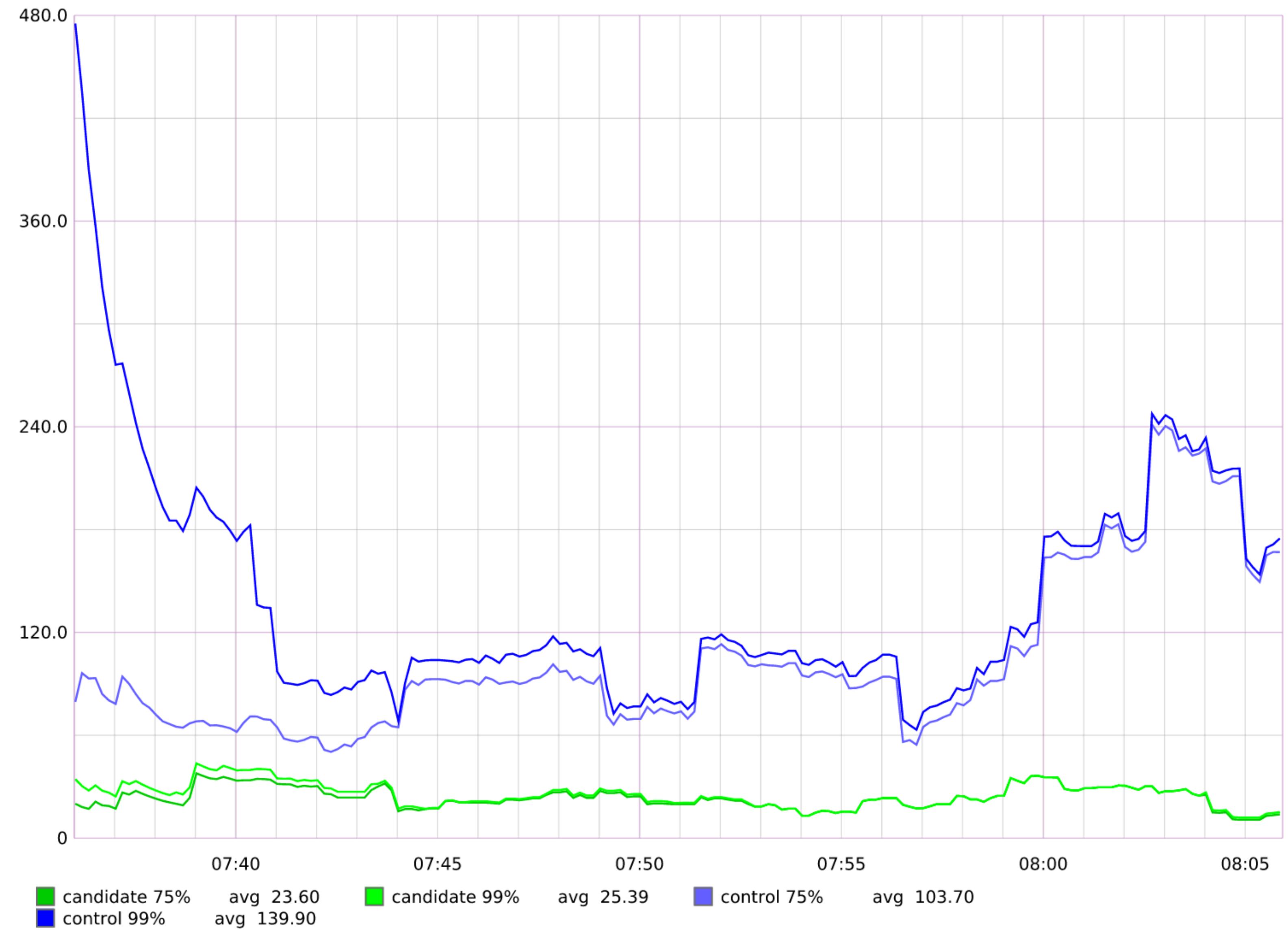


The number of incorrect/ignored only.

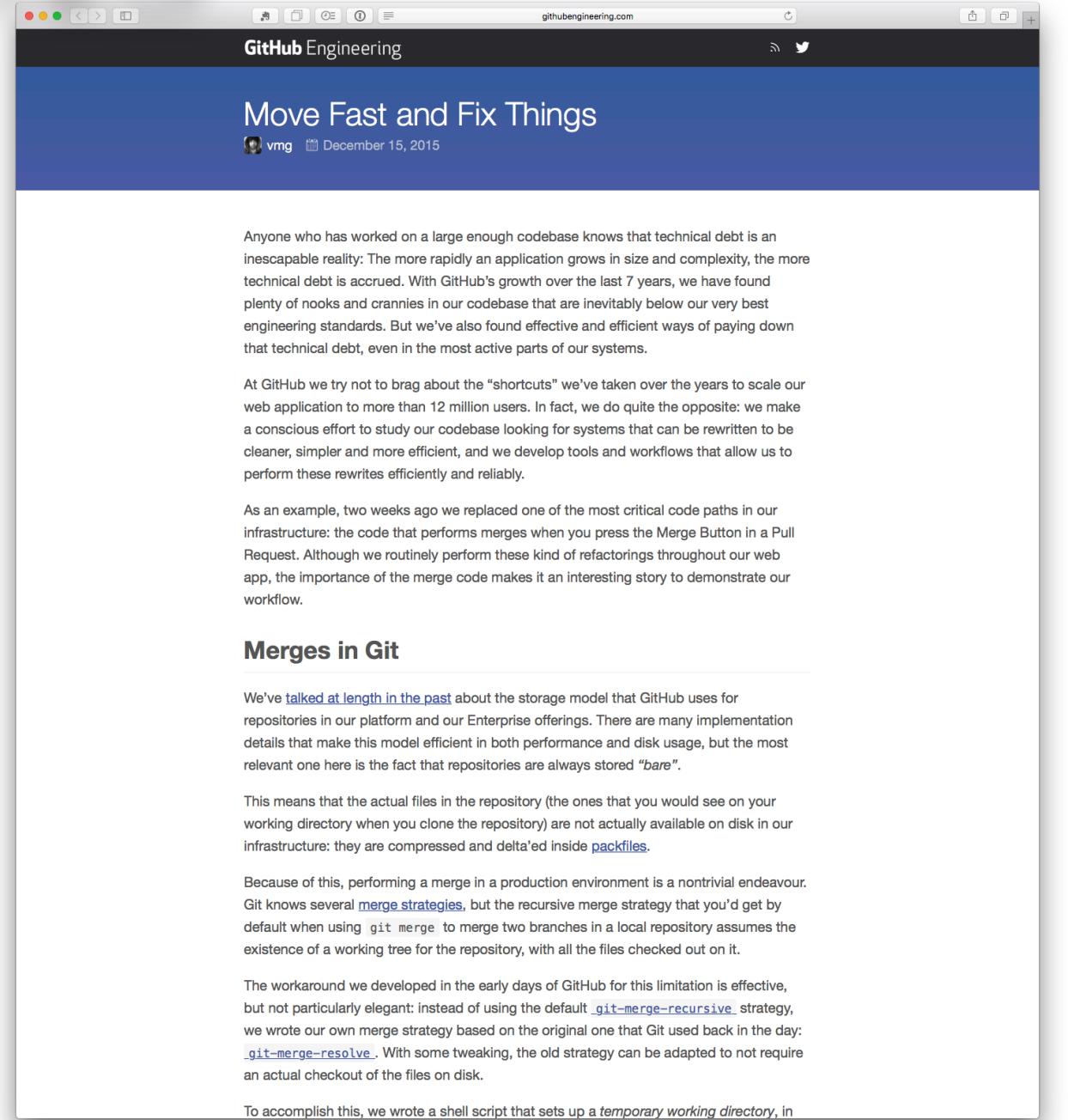




### create\_merge\_commit



*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



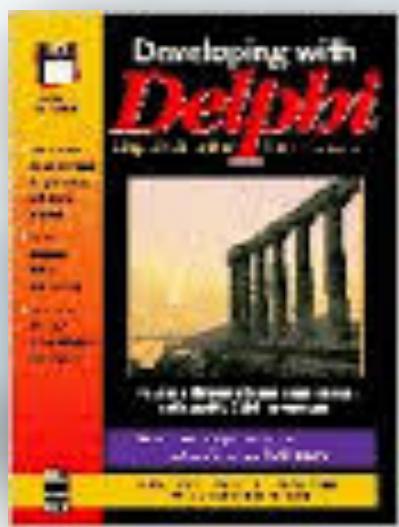
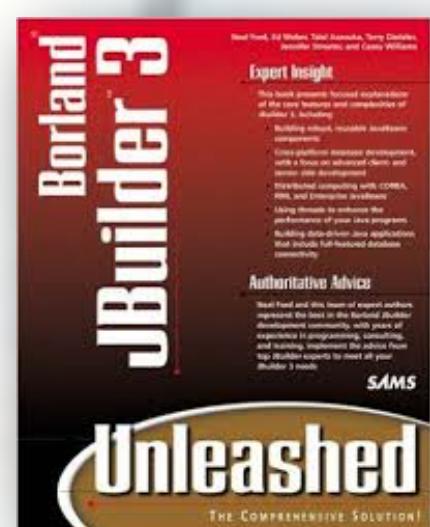
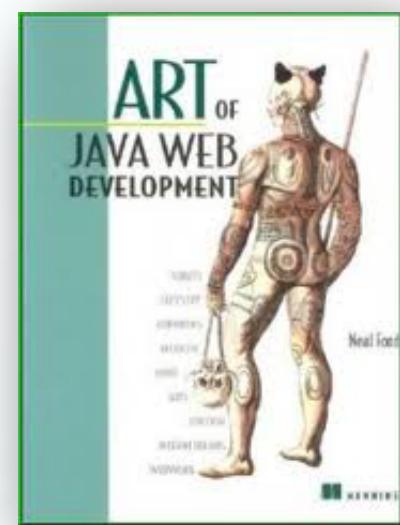
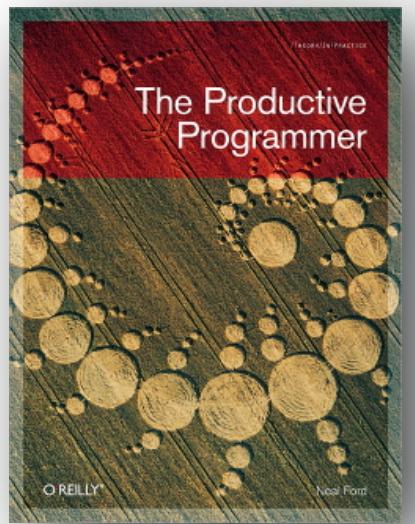
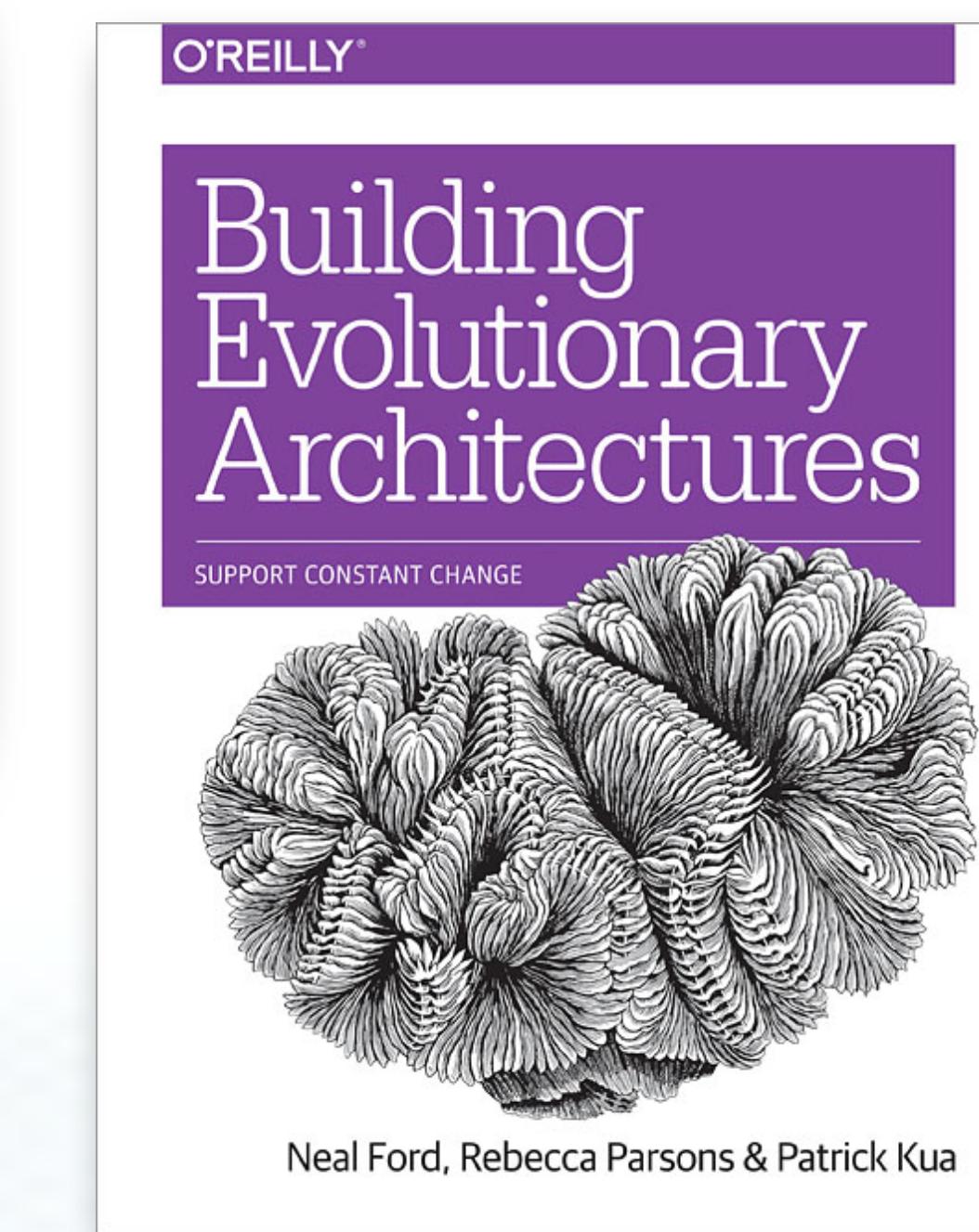
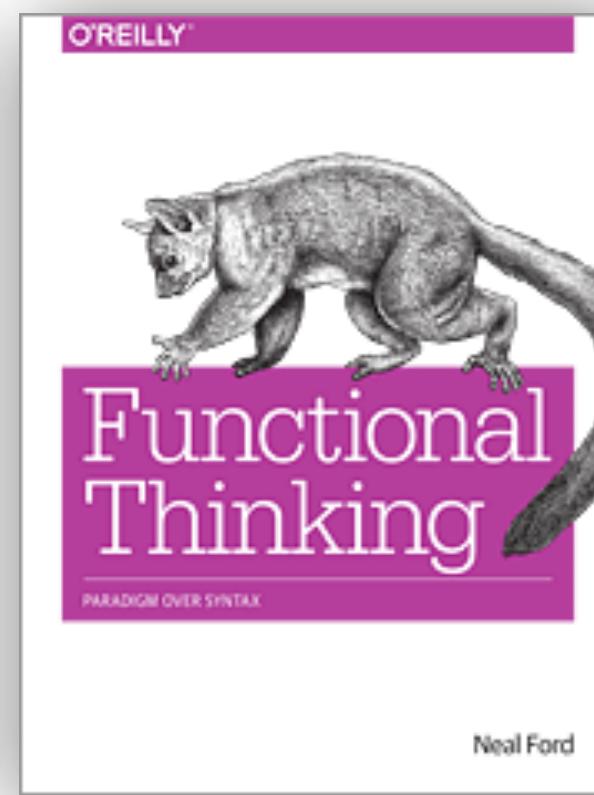
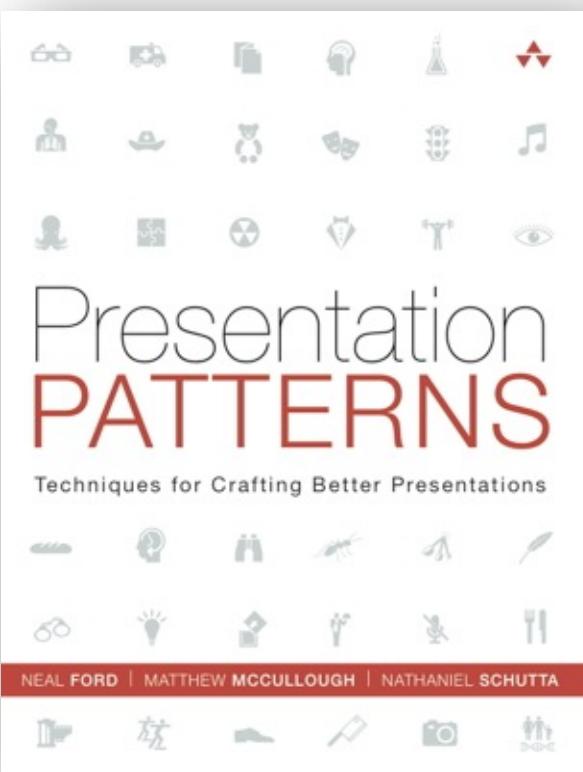
Apply good engineering practices  
to architecture.



# aRChiTeCtuRe fOr cONTiNuoUs DeLiVeRy



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



[nealford.com/books](http://nealford.com/books)

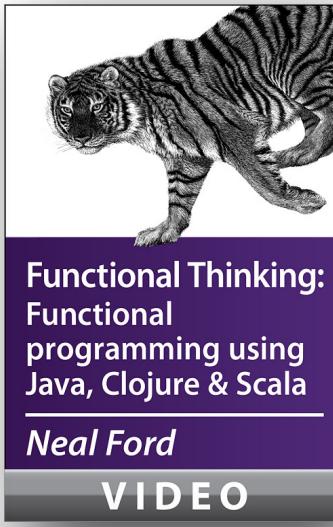
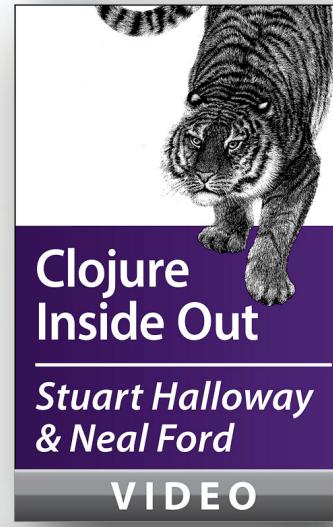
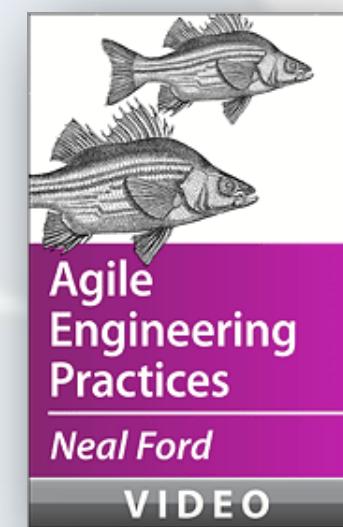


ThoughtWorks®

**NEAL FORD**

Director / Software Architect / Meme Wrangler

[nealford.com/videos](http://nealford.com/videos)



[www.oreilly.com/software-architecture-video-training-series.html](http://www.oreilly.com/software-architecture-video-training-series.html)

