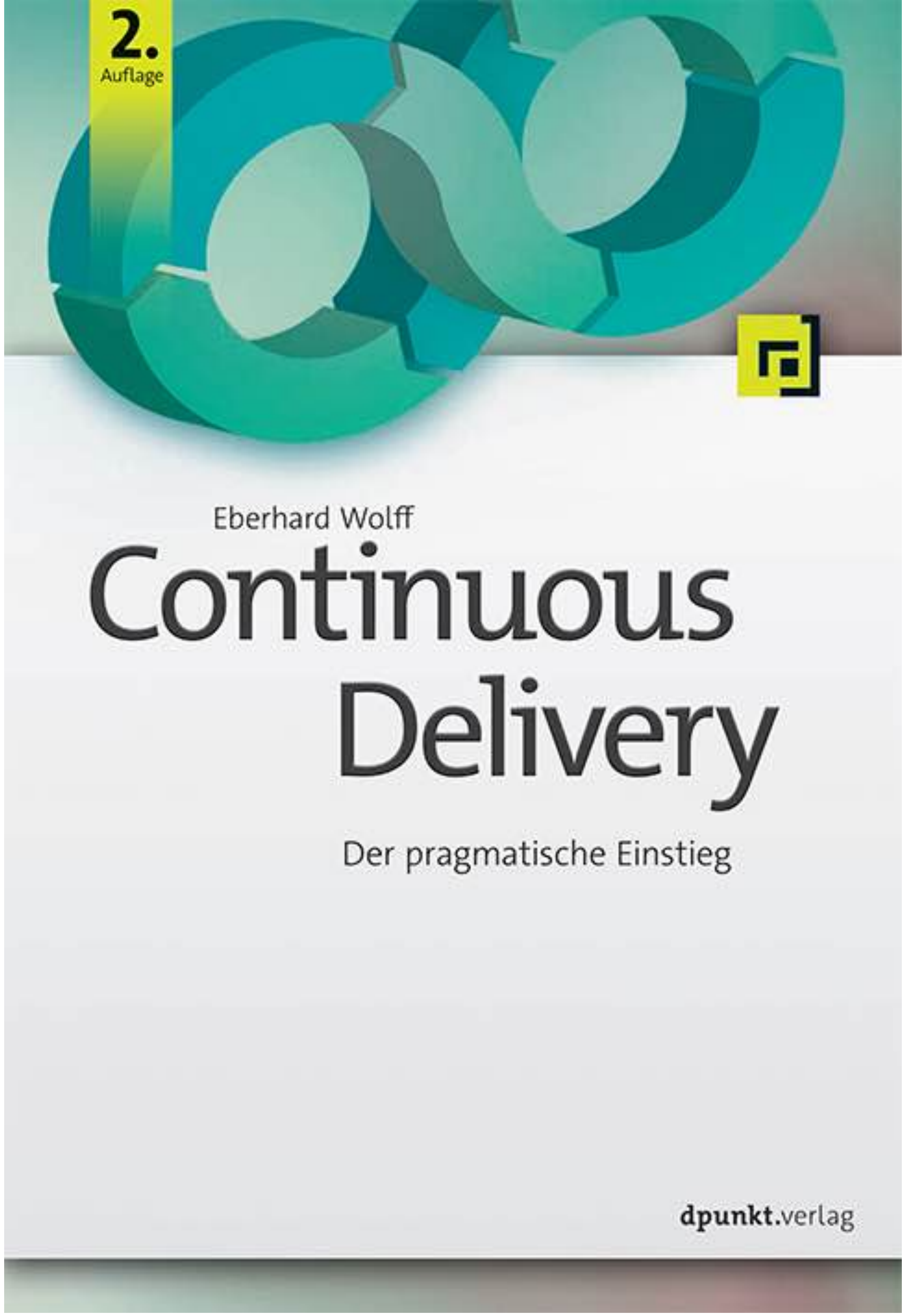
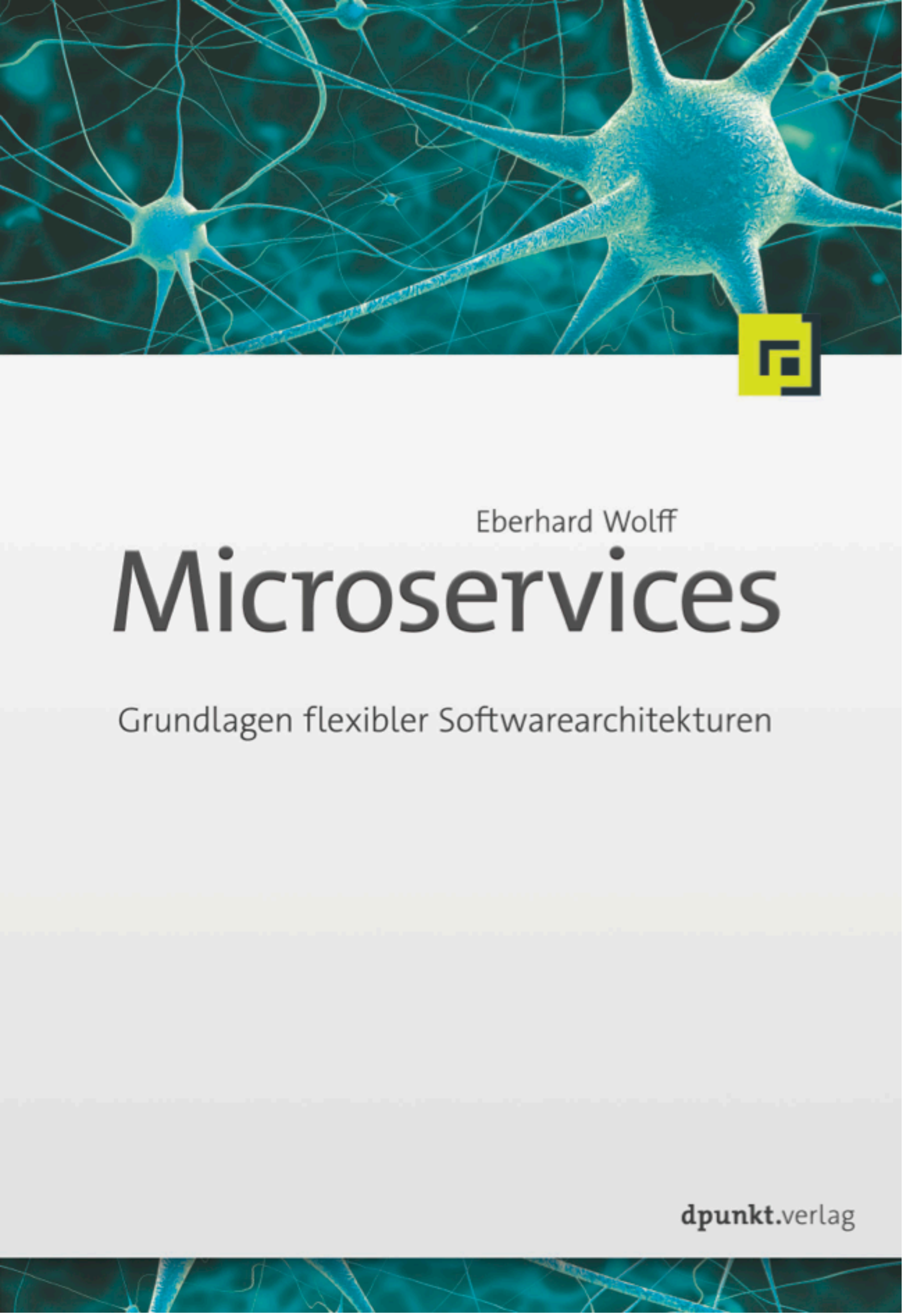
Microservices: Redundancy = Maintainability!

Eberhard Wolff
@ewolff
Fellow
innoQ



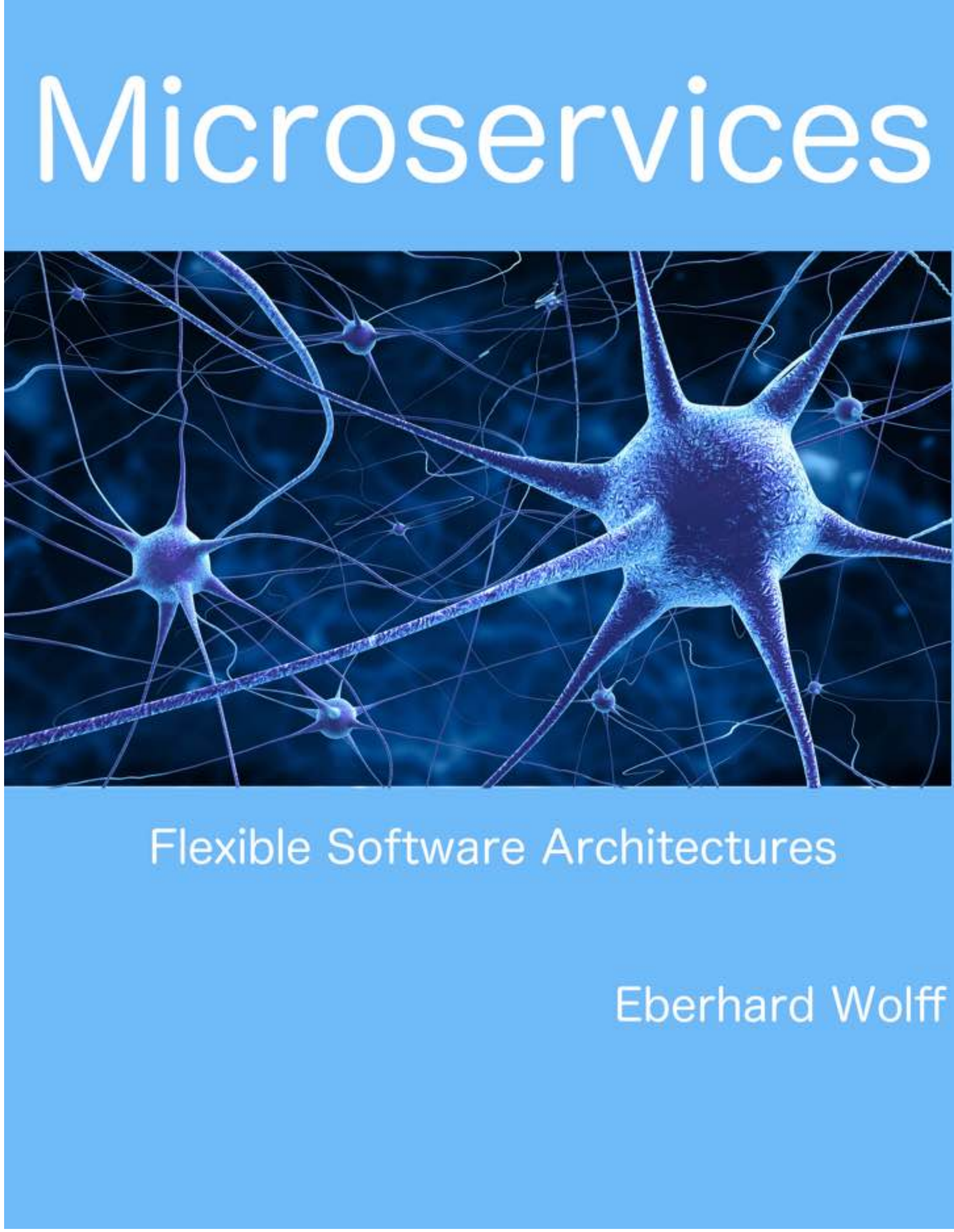
http://continuous-delivery-buch.de/


Eberhard Wolff

Microservices

Grundlagen flexibler Softwarearchitekturen

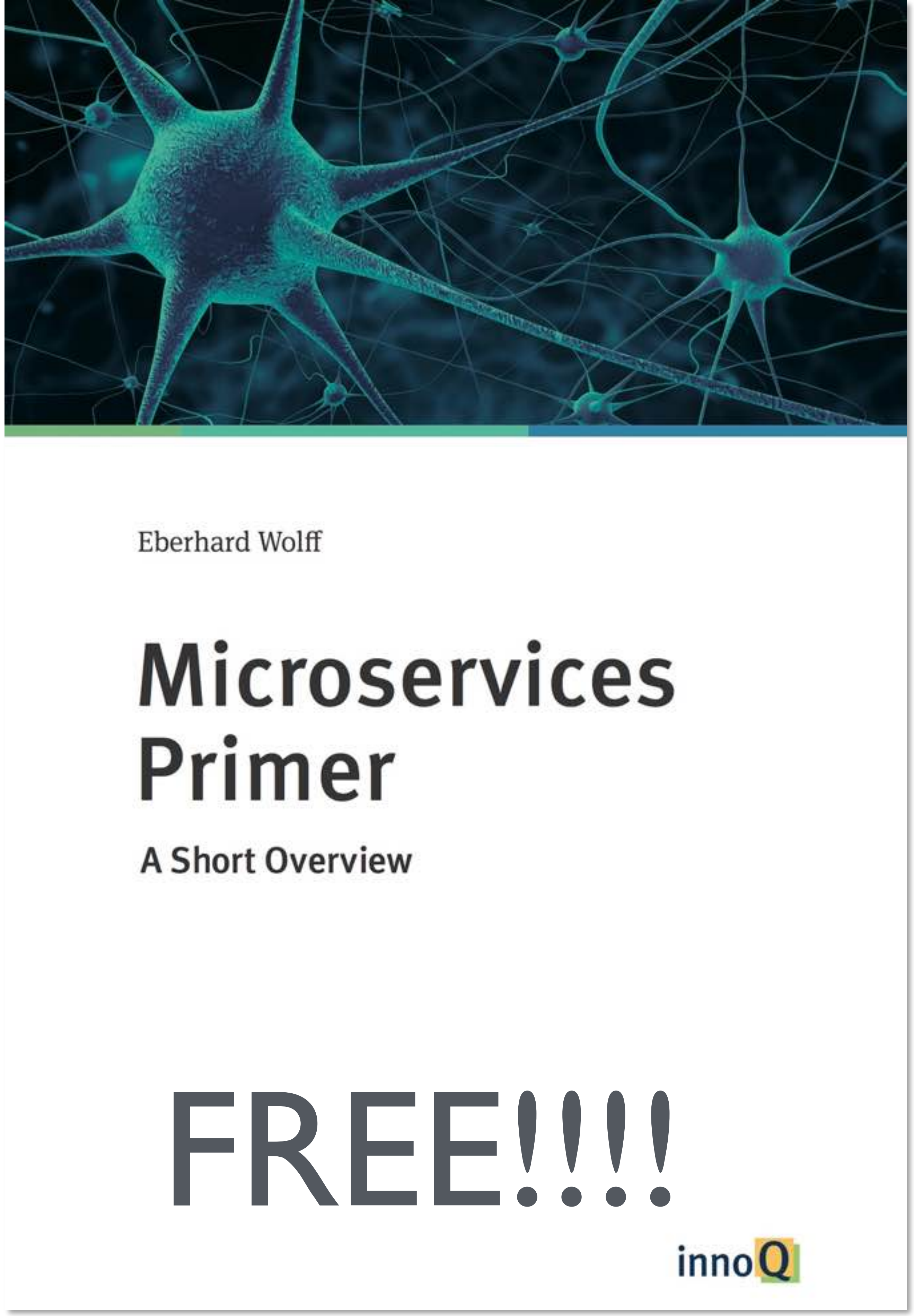
dpunkt.verlag


Microservices

Flexible Software Architectures

Eberhard Wolff

http://microservices-buch.de/ http://microservices-book.com/



Eberhard Wolff

Microservices Primer

A Short Overview

FREE!!!!

innoQ

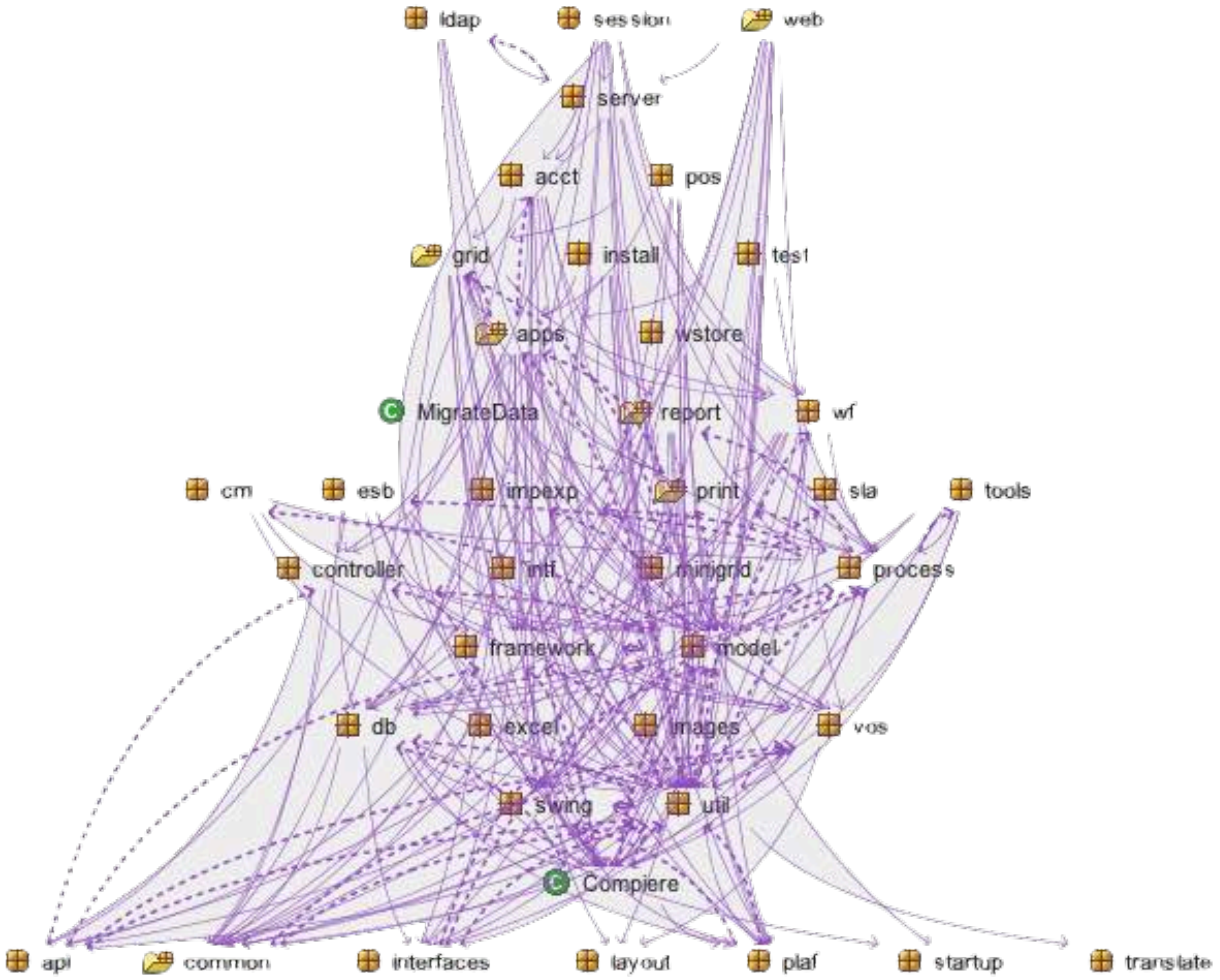
http://microservices-book.com/primer.html

Maintainablity

Redundant data

Redudant code

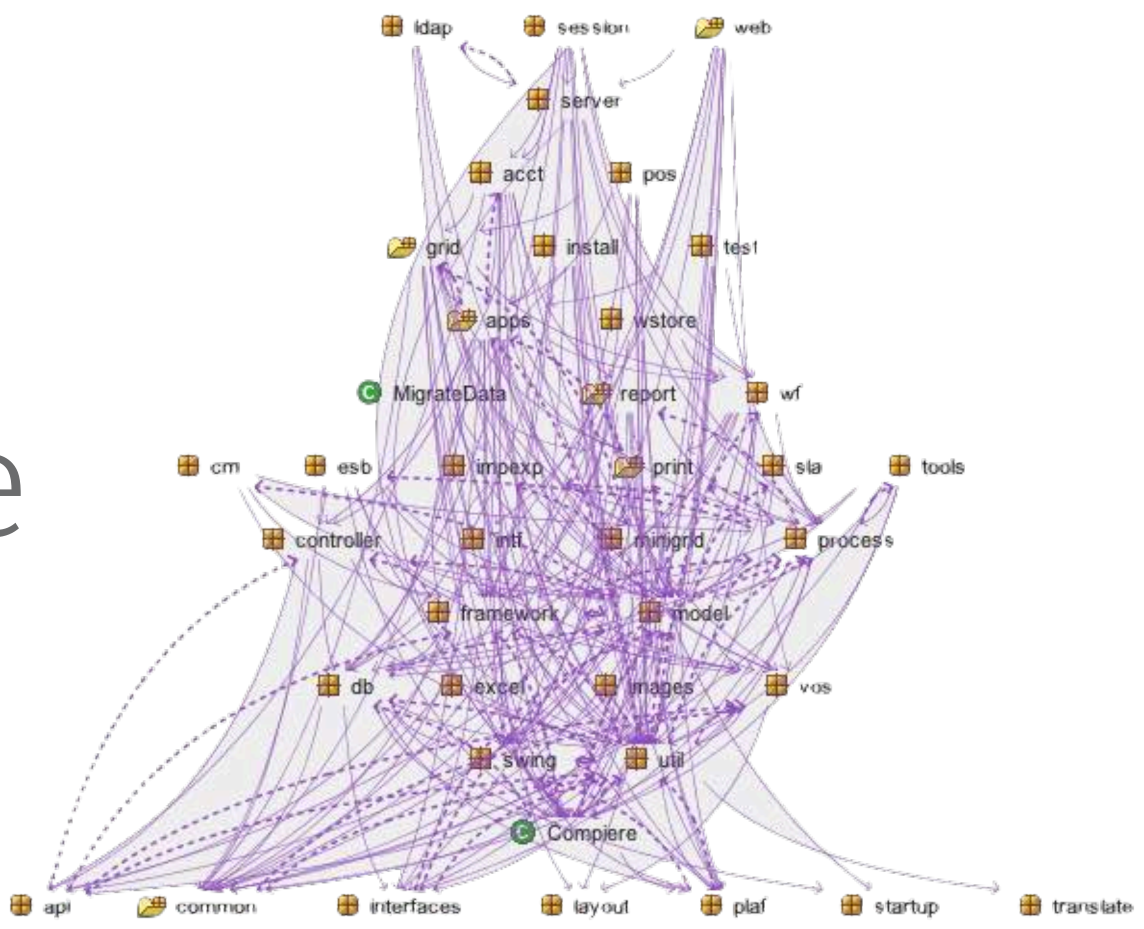
Legacy System


ldap
session
web
server
acct
pos
grid
install
test
apps
wstore
MigrateData
report
wf
cm
esb
impexp
print
sla
tools
controller
intf
minigrid
process
framework
model
db
excel
images
vos
swing
util
Compiere
api
common
interfaces
layout
plaf
startup
translate

Too many dependencies

Cyclic dependencies (dotted lines)

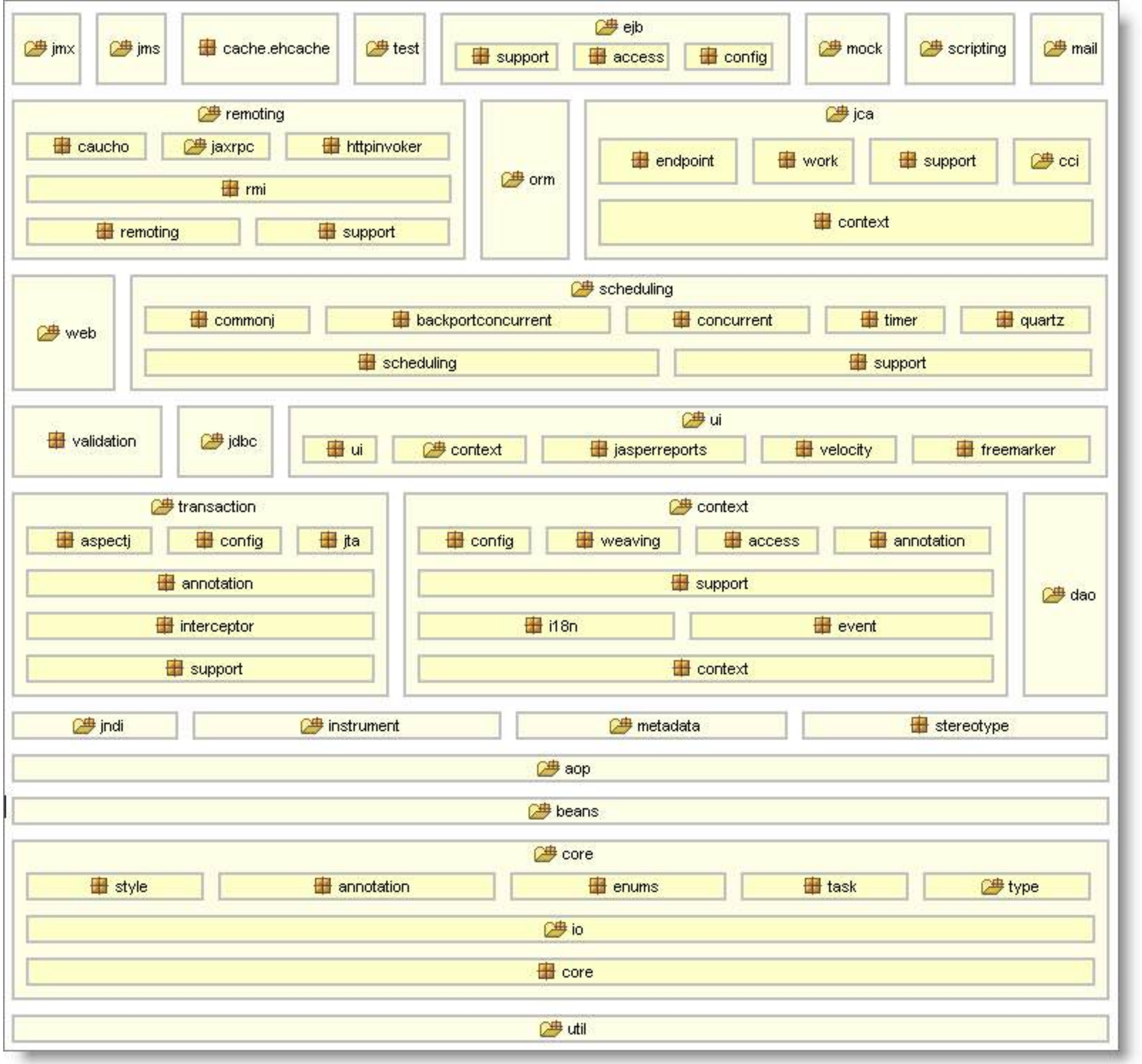
- COBOL, Assembler
- Not maintainable
- Not replaceable



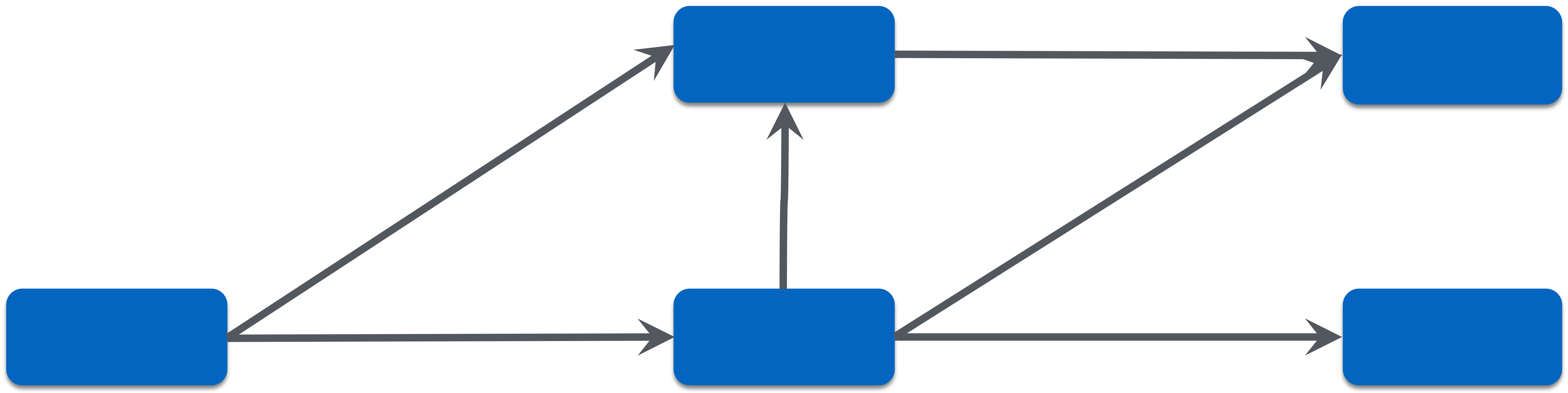
- We will replace it!
- We will make it maintainable!
- It will be beautiful!

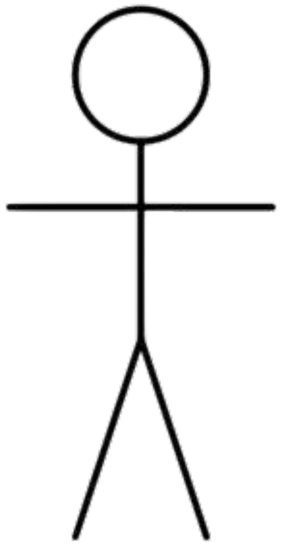
We will take good care of the code!

Clean Like Spring

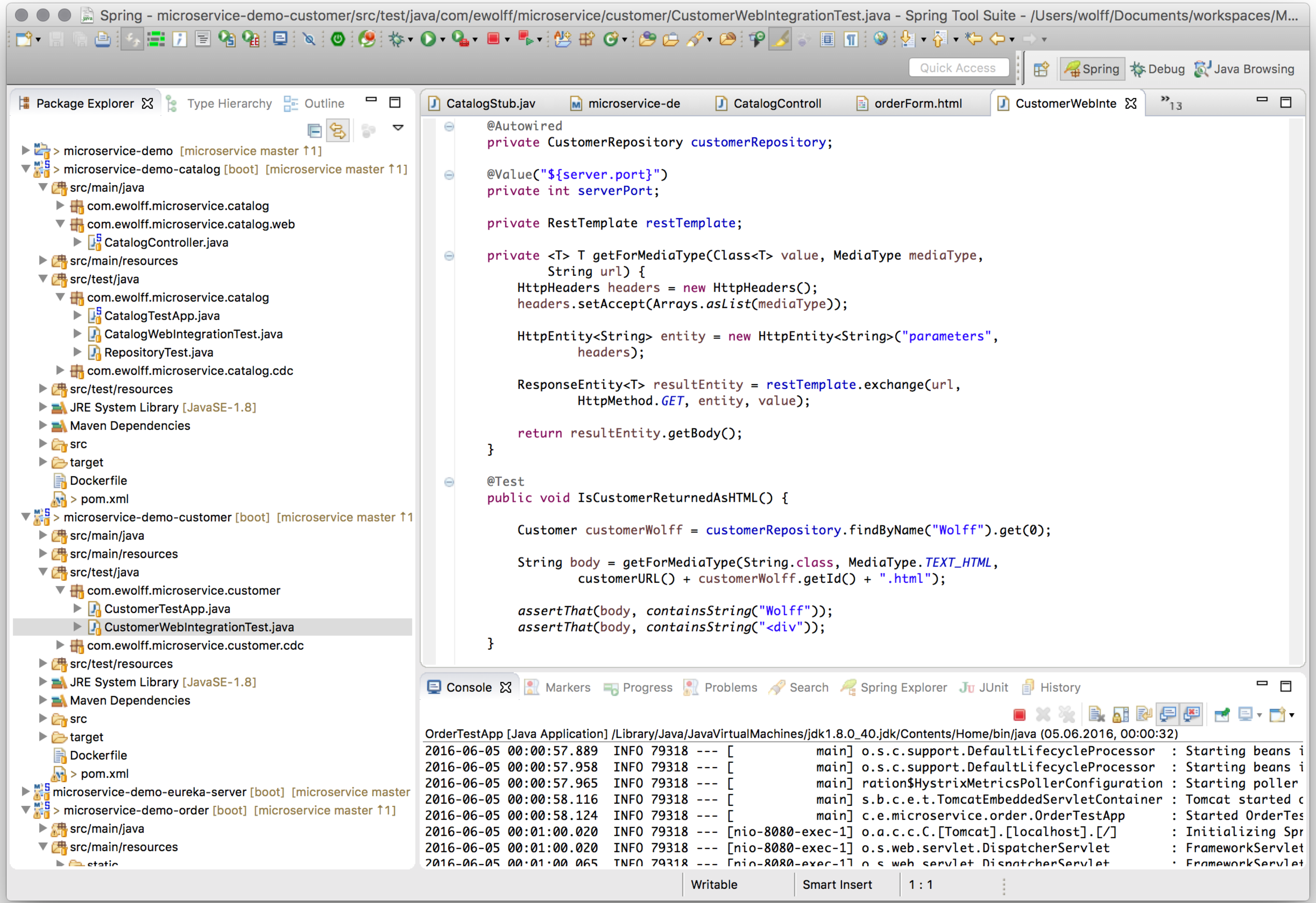


Clean Architecture





Developer

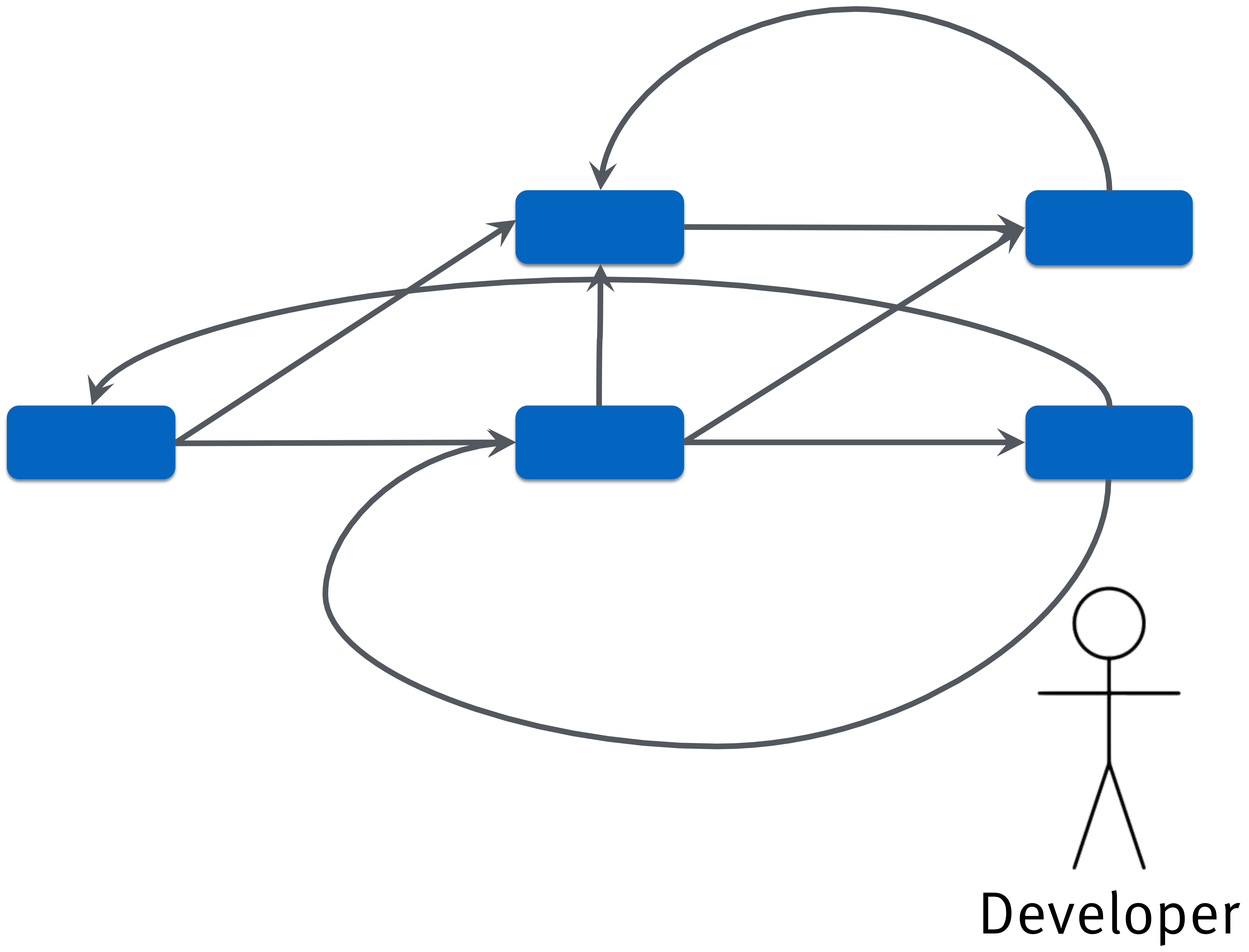


Spring - microservice-demo-customer/src/test/java/com/ewolff/microservice/customer/CustomerWebIntegrationTest.java - Spring Tool Suite - /Users/wolff/Documents/workspaces/M...

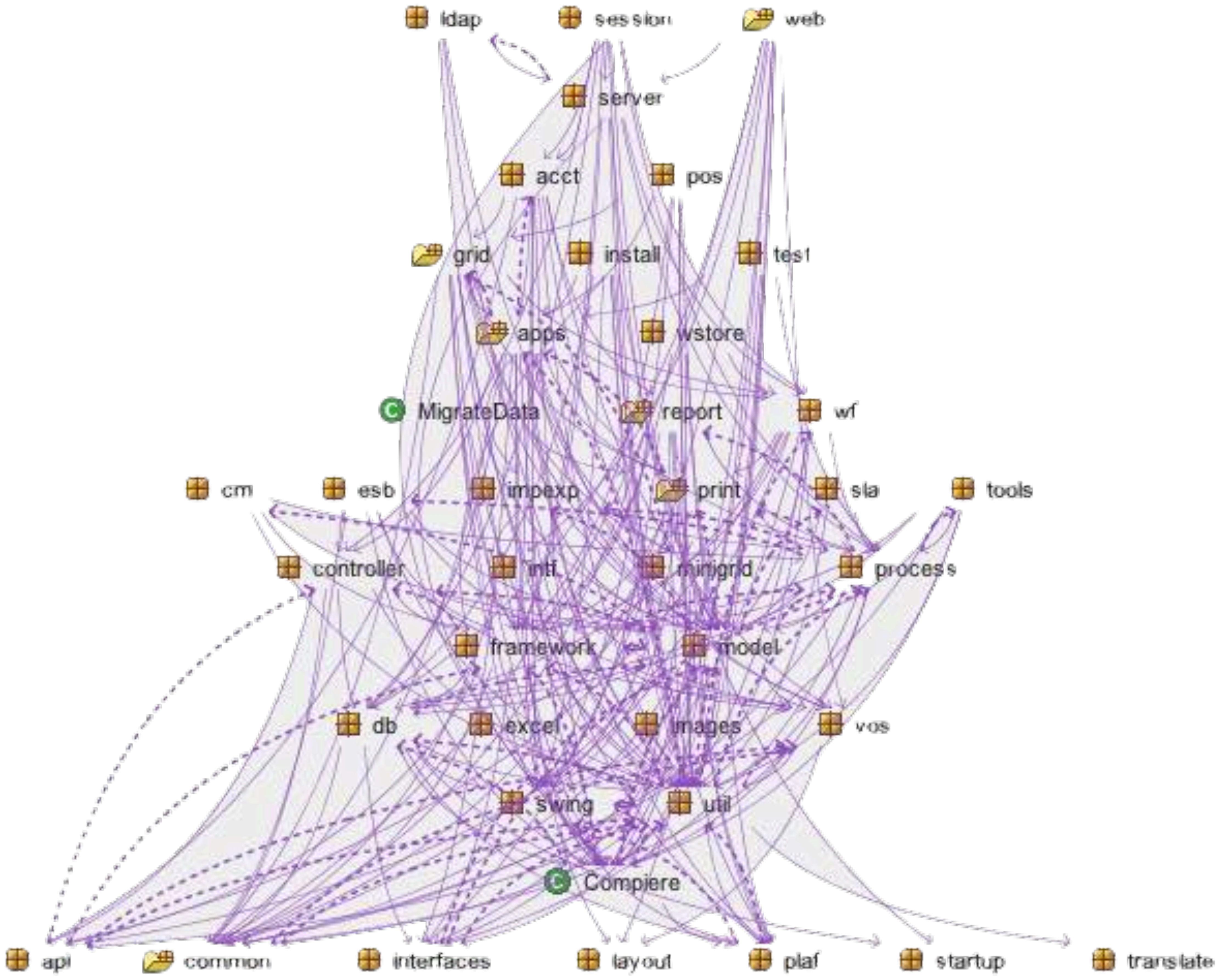
```
> microservice-demo  [microservice master ↑1]
> microservice-demo-catalog [boot] [microservice master ↑1]
    src/main/java
        com.ewolff.microservice.catalog
        com.ewolff.microservice.catalog.web
            CatalogController.java
    src/main/resources
    src/test/java
        com.ewolff.microservice.catalog
            CatalogTestApp.java
            CatalogWebIntegrationTest.java
            RepositoryTest.java
        com.ewolff.microservice.catalog.cdc
    src/test/resources
    JRE System Library [JavaSE-1.8]
    Maven Dependencies
    src
    target
    Dockerfile
    > pom.xml
> microservice-demo-customer [boot] [microservice master ↑1]
    src/main/java
    src/main/resources
    src/test/java
        com.ewolff.microservice.customer
            CustomerTestApp.java
            CustomerWebIntegrationTest.java
        com.ewolff.microservice.customer.cdc
    src/test/resources
    JRE System Library [JavaSE-1.8]
    Maven Dependencies
    src
    target
    Dockerfile
    > pom.xml
microservice-demo-eureka-server [boot] [microservice master
> microservice-demo-order [boot] [microservice master ↑1]
    src/main/java
    src/main/resources
        static
```

```java
    @Autowired
    private CustomerRepository customerRepository;

    @Value("${server.port}")
    private int serverPort;

    private RestTemplate restTemplate;

    private <T> T getForMediaType(Class<T> value, MediaType mediaType,
            String url) {
        HttpHeaders headers = new HttpHeaders();
        headers.setAccept(Arrays.asList(mediaType));

        HttpEntity<String> entity = new HttpEntity<String>("parameters",
                headers);

        ResponseEntity<T> resultEntity = restTemplate.exchange(url,
                HttpMethod.GET, entity, value);

        return resultEntity.getBody();
    }

    @Test
    public void IsCustomerReturnedAsHTML() {

        Customer customerWolff = customerRepository.findByName("Wolff").get(0);

        String body = getForMediaType(String.class, MediaType.TEXT_HTML,
                customerURL() + customerWolff.getId() + ".html");

        assertThat(body, containsString("Wolff"));
        assertThat(body, containsString("<div"));
    }
```

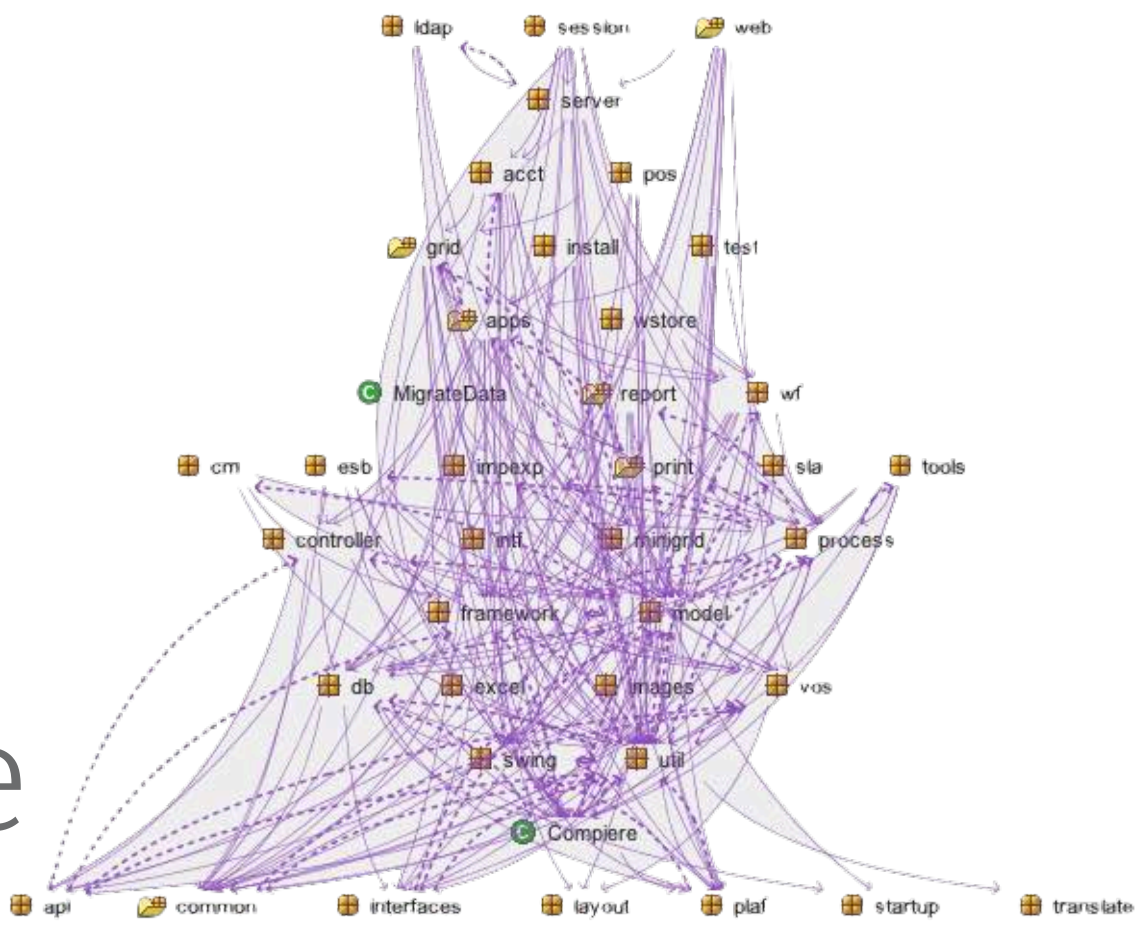
```
OrderTestApp [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/Contents/Home/bin/java (05.06.2016, 00:00:32)
2016-06-05 00:00:57.889  INFO 79318 --- [           main] o.s.c.support.DefaultLifecycleProcessor  : Starting beans i
2016-06-05 00:00:57.958  INFO 79318 --- [           main] o.s.c.support.DefaultLifecycleProcessor  : Starting beans i
2016-06-05 00:00:57.965  INFO 79318 --- [           main] ration$HystrixMetricsPollerConfiguration : Starting poller
2016-06-05 00:00:58.116  INFO 79318 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started c
2016-06-05 00:00:58.124  INFO 79318 --- [           main] c.e.microservice.order.OrderTestApp      : Started OrderTes
2016-06-05 00:01:00.020  INFO 79318 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spr
2016-06-05 00:01:00.020  INFO 79318 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : FrameworkServlet
2016-06-05 00:01:00.065  INFO 79318 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : FrameworkServlet
```


Developer

Result?


ldap
session
web
server
acct
pos
grid
install
test
apps
wstore
MigrateData
report
wf
cm
esb
impexp
print
sla
tools
controller
intf
minigrid
process
framework
model
db
excel
images
vos
swing
util
Compiere
api
common
interfaces
layout
plaf
startup
translate

- Legacy System
- Java
- Not maintainable
- Not replaceable



- We didn’t try hard enough!
- We will replace it!
- We will make it maintainable!
- It will be beautiful!

☹

I need a new job.

While there are still developers:

Replace the legacy system.

Repeat

Insanity:
Doing the same thing over and over again and expecting different results.

Albert Einstein

We can achieve maintainability with clean architecture + clean code.

We can achieve maintainability with clean architecture + clean code.

WRONG

Clean approach tried often.

Results?

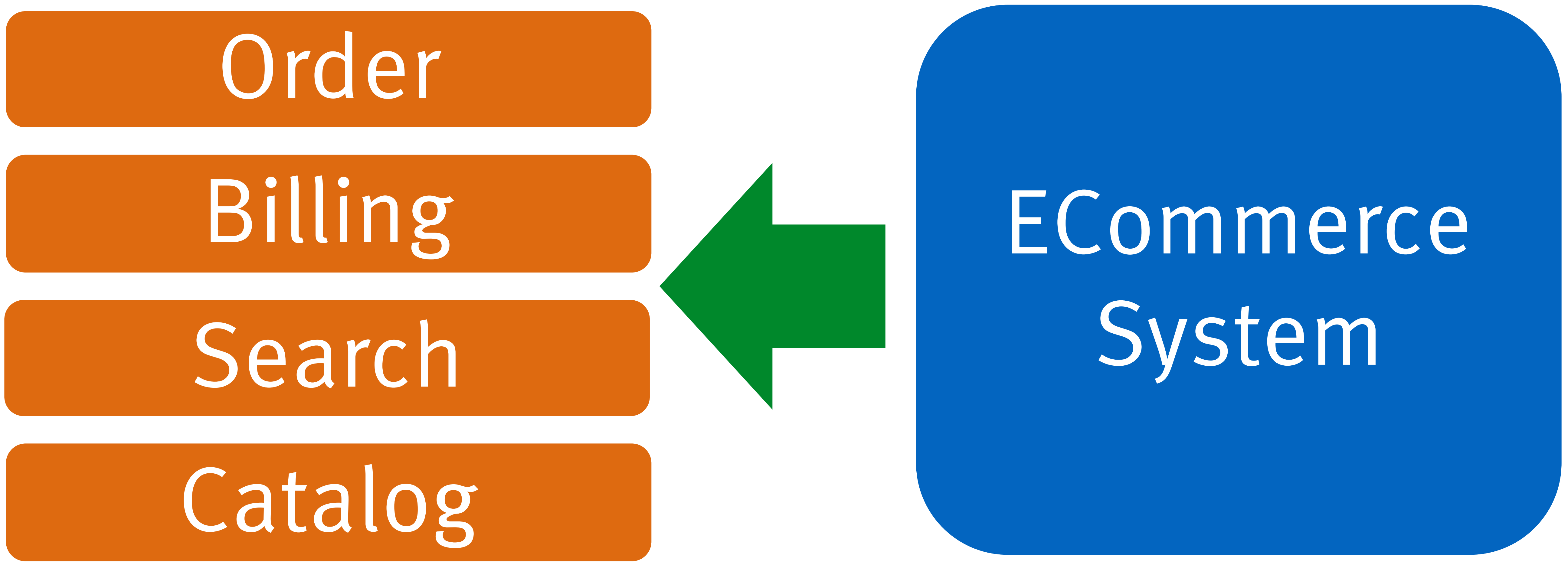
Lots of Legacy Code

...and secure jobs.

We need a different approach!

Parnas 1972

Modules

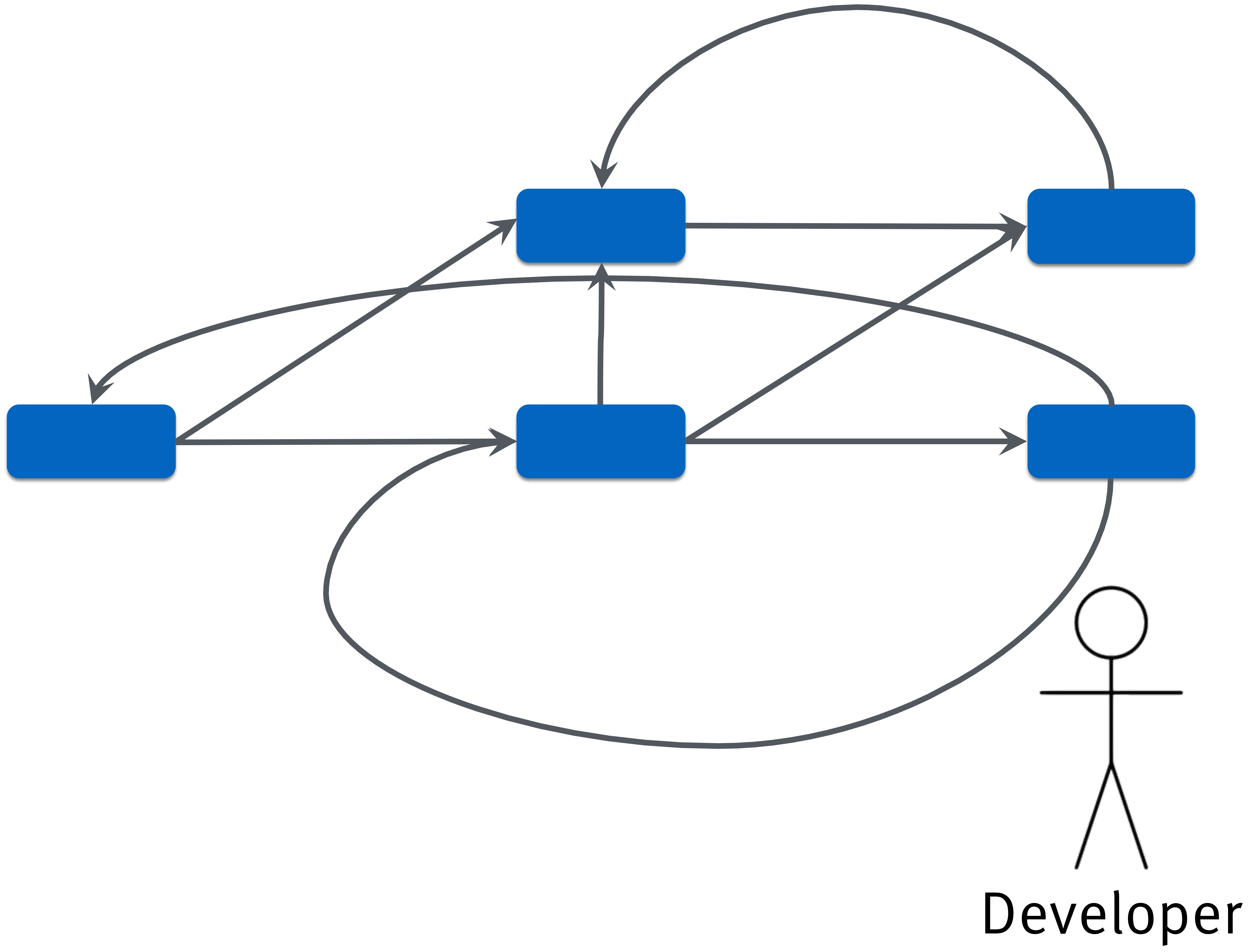

Order
Billing
Search
Catalog
ECommerce System

Modules by Domain

- Each domain problem solved in one module.
- New features easy to add

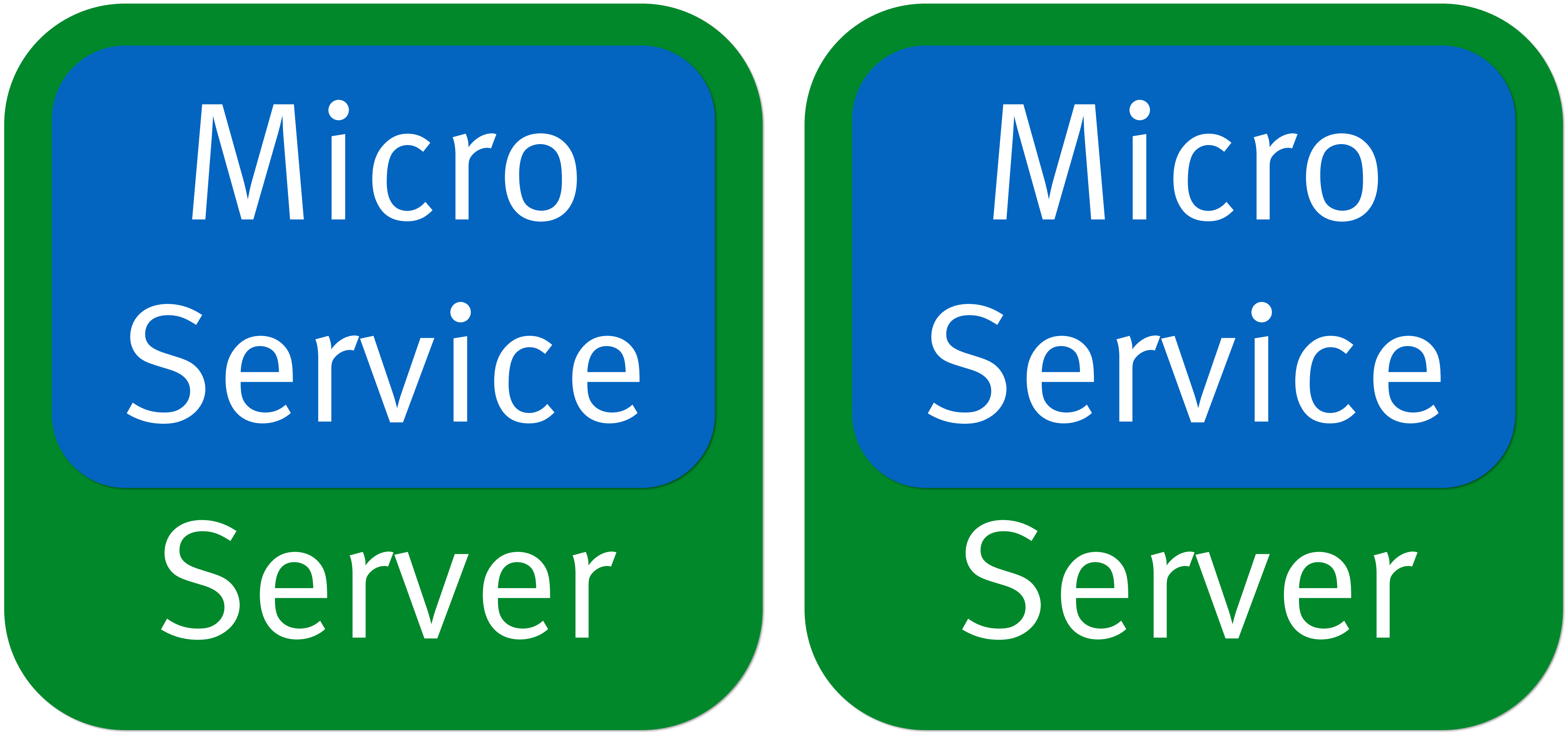
Modules

- Programming language feature
- Class, package, library …
- Rather weak modules

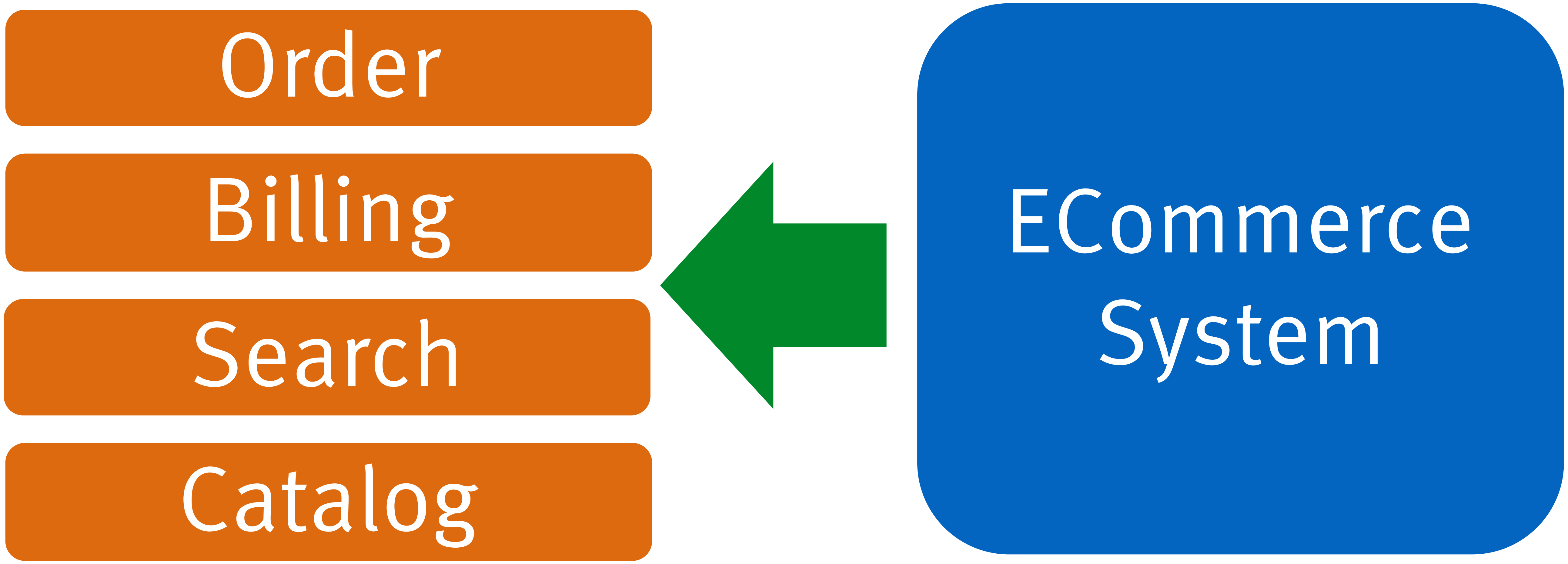

Developer

Microservices

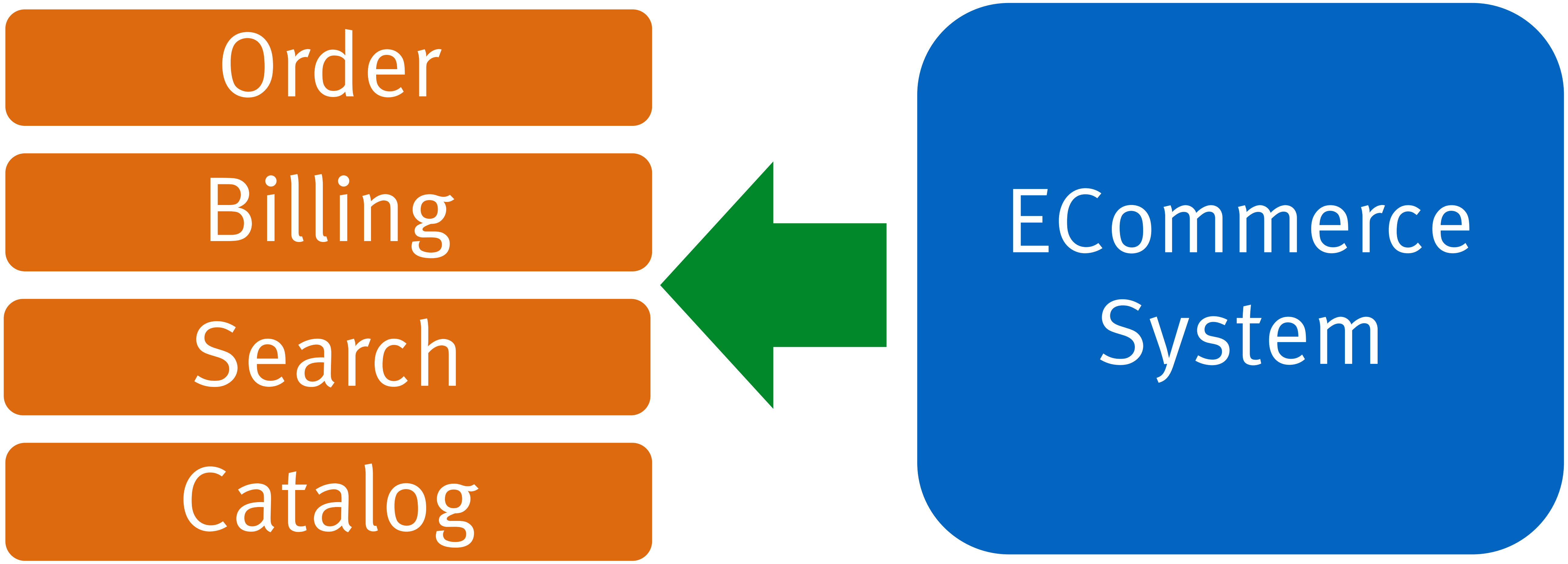
- Modules
- Separate deployment units
- Separate VM / process


Micro Service
Server
Micro Service
Server

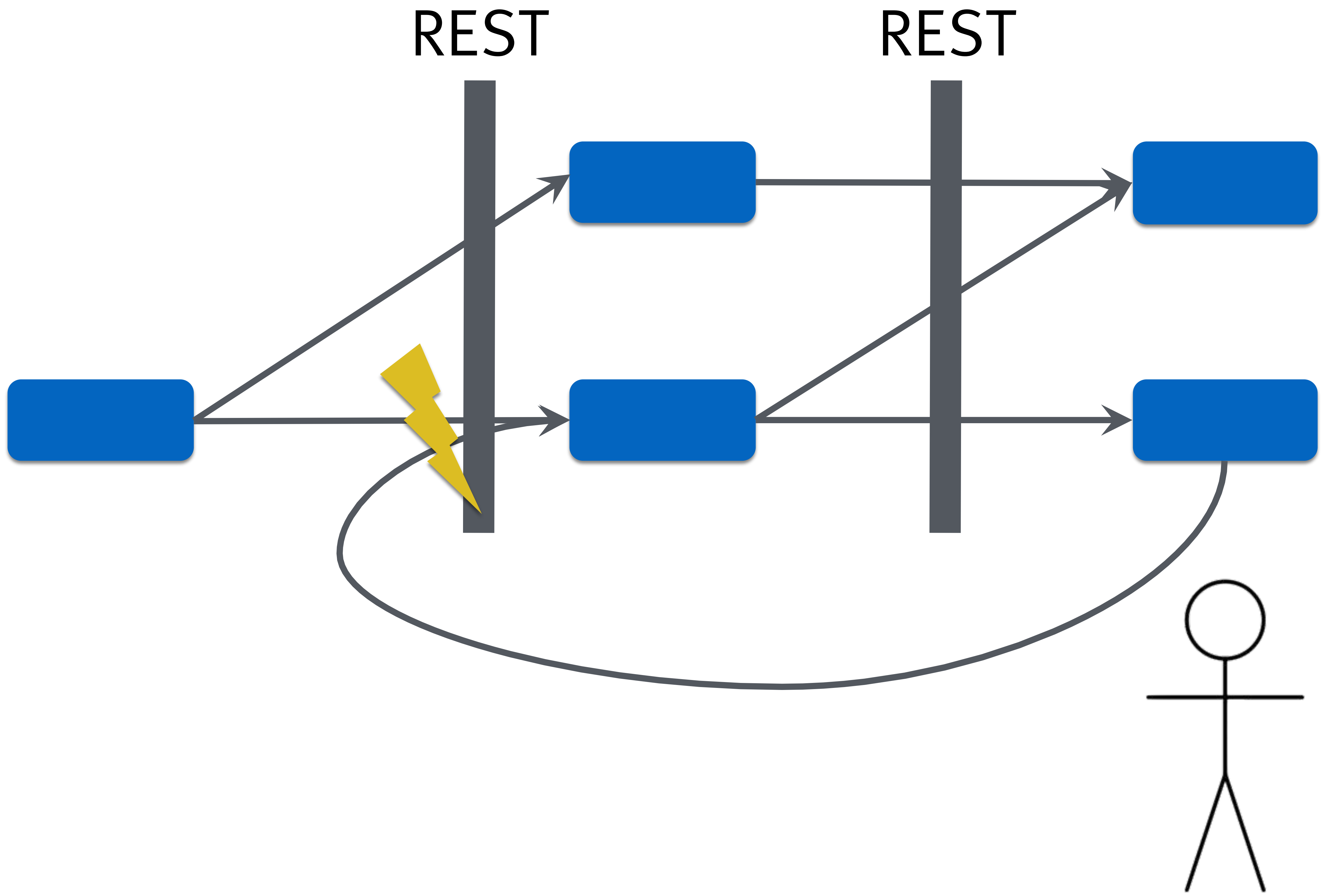
Module = separate deployment units!



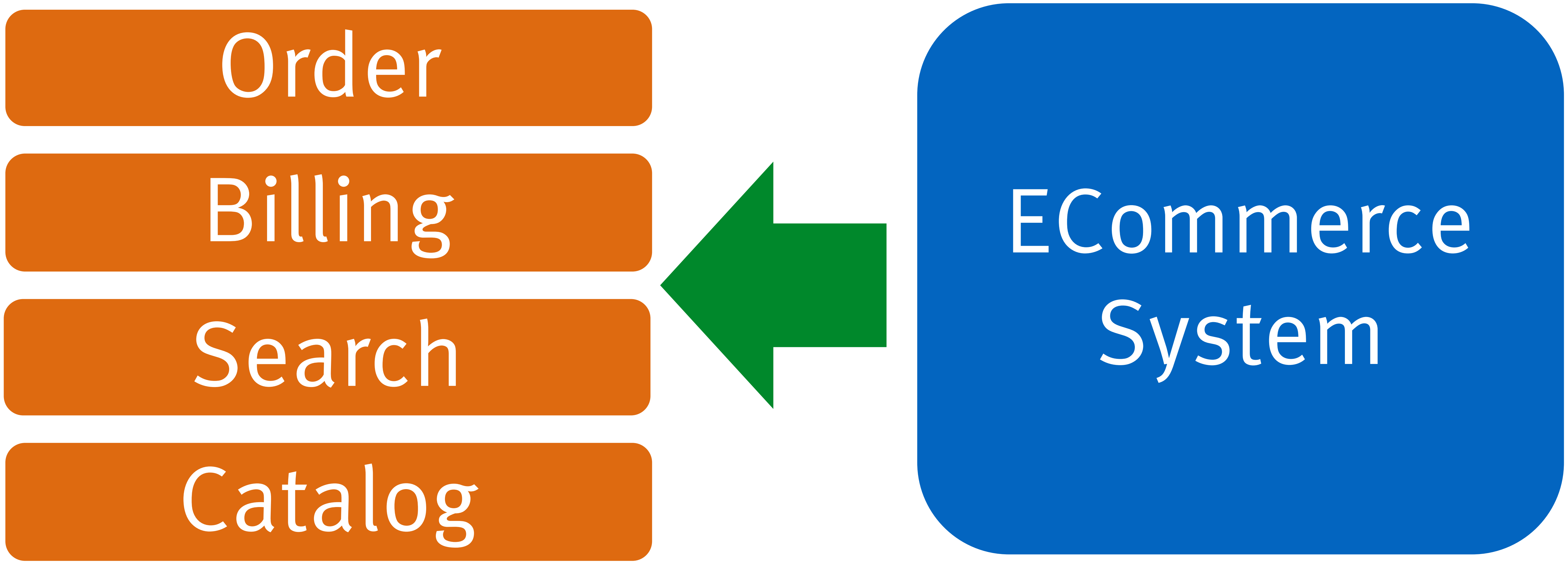
Module = separate deployment units!



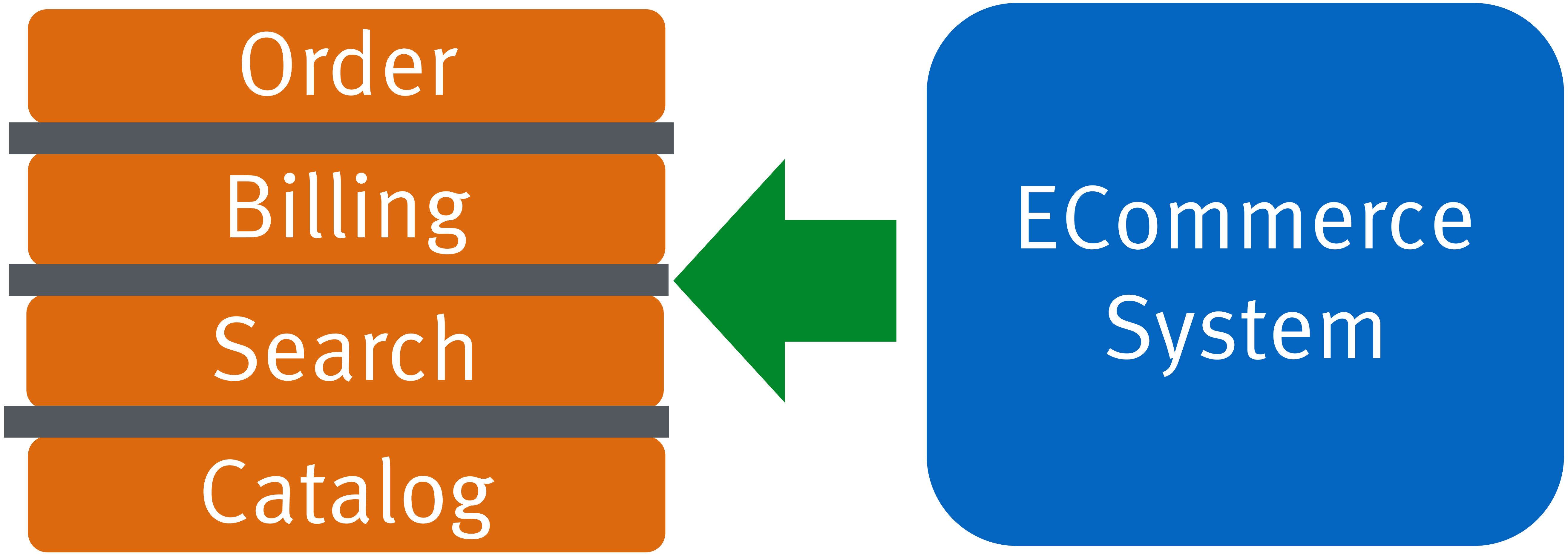
Communication e.g. REST


REST
REST

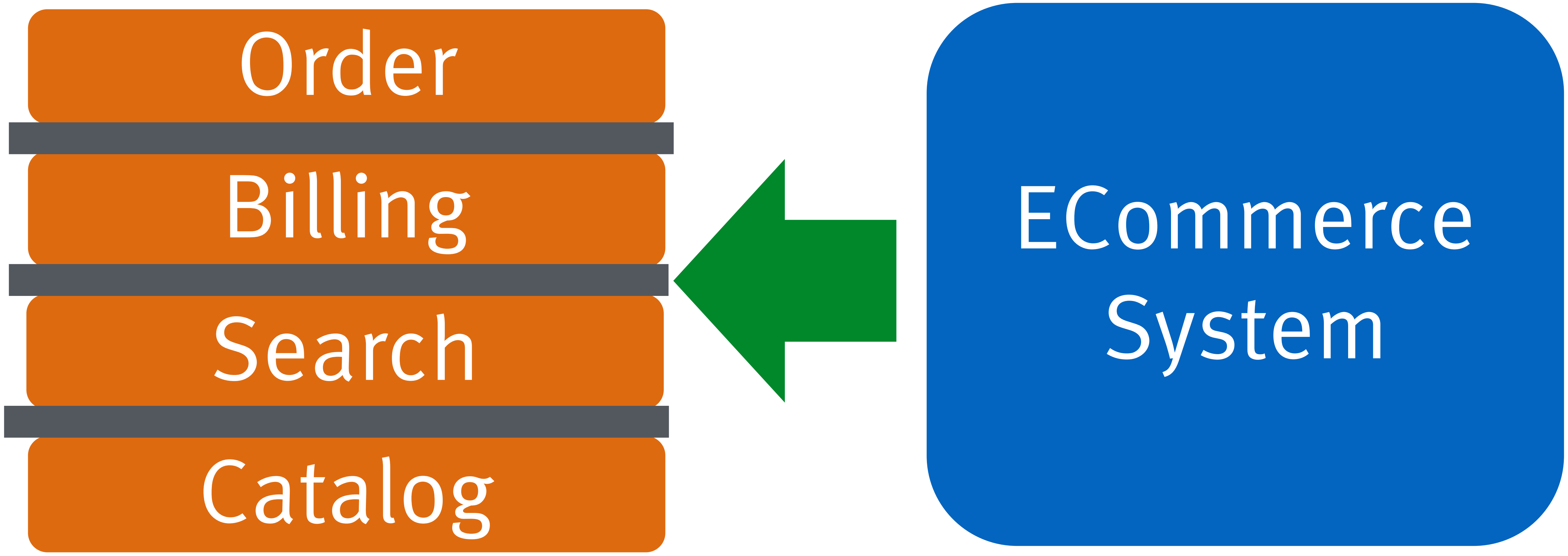
Dependencies between systems cannot sneak in



Dependencies between systems cannot sneak in



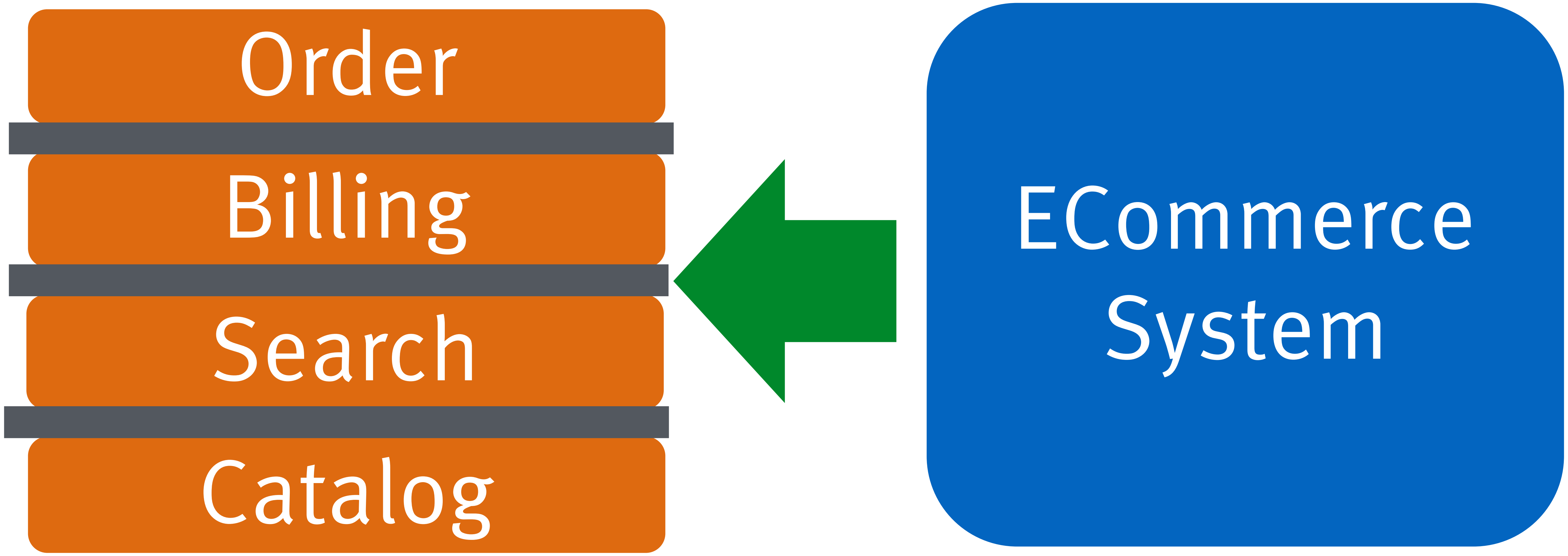
Dependencies between systems cannot sneak in



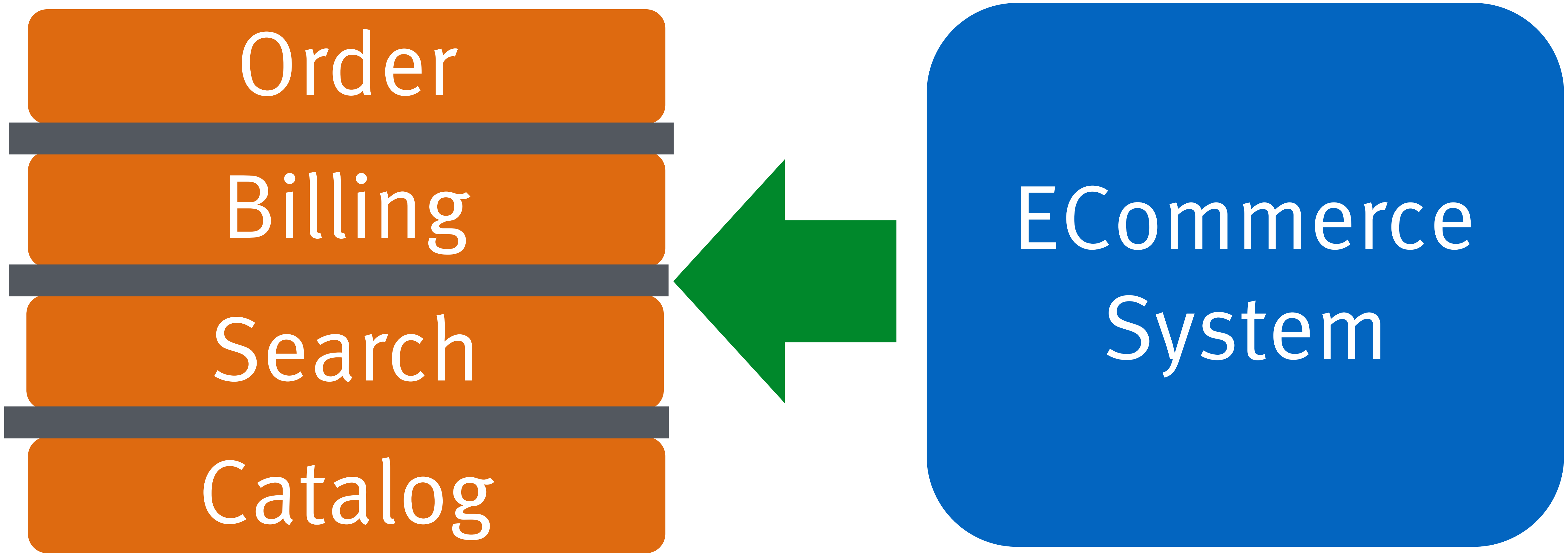
“Architecture Firewalls”

“Architecture Firewall”
like REST
enforce the architecture

Components small

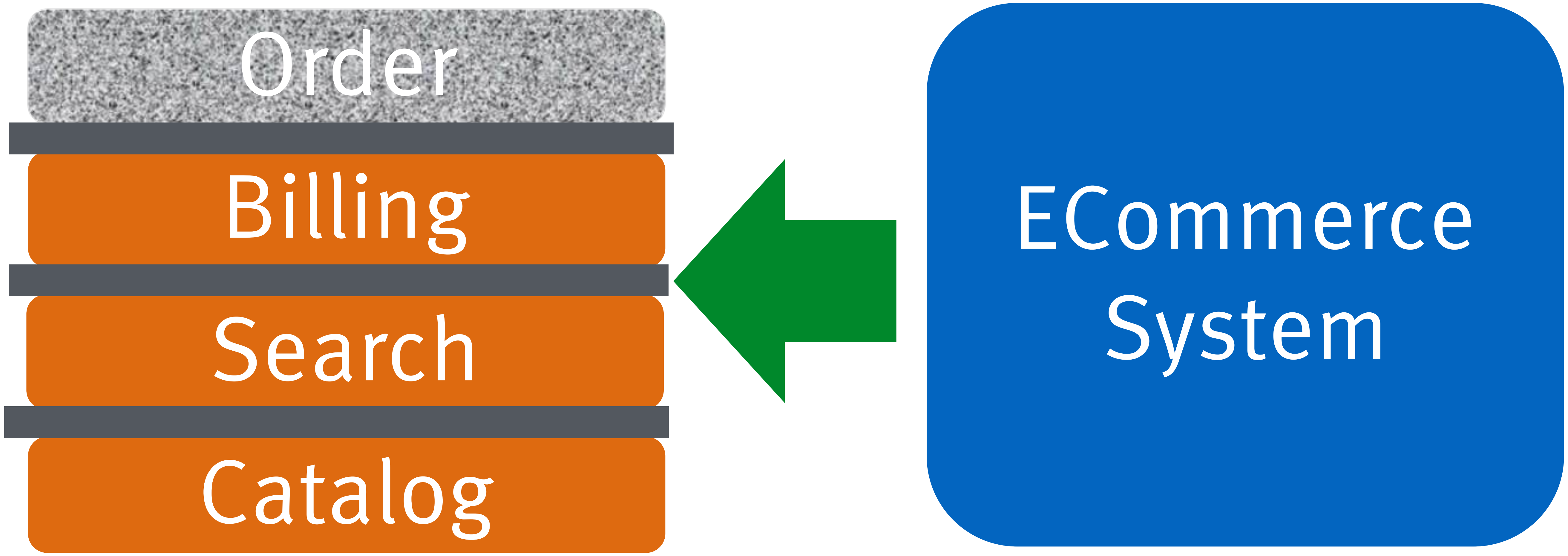


Components small



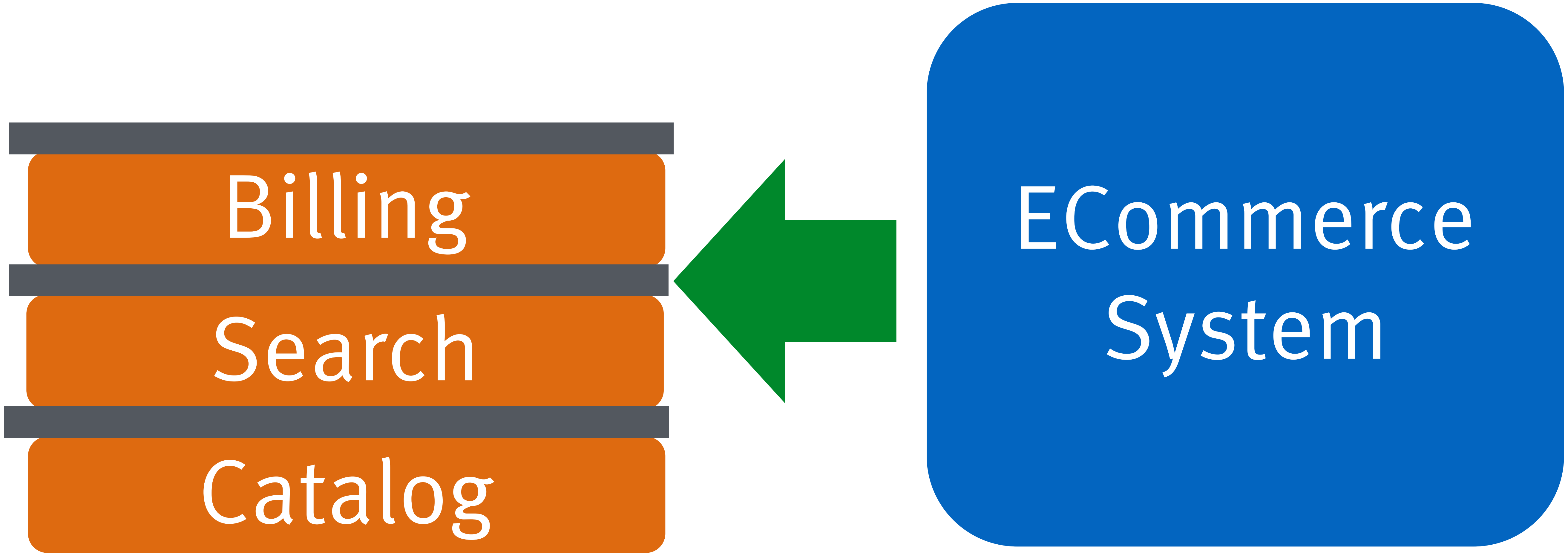
Hard to mess up

Components small



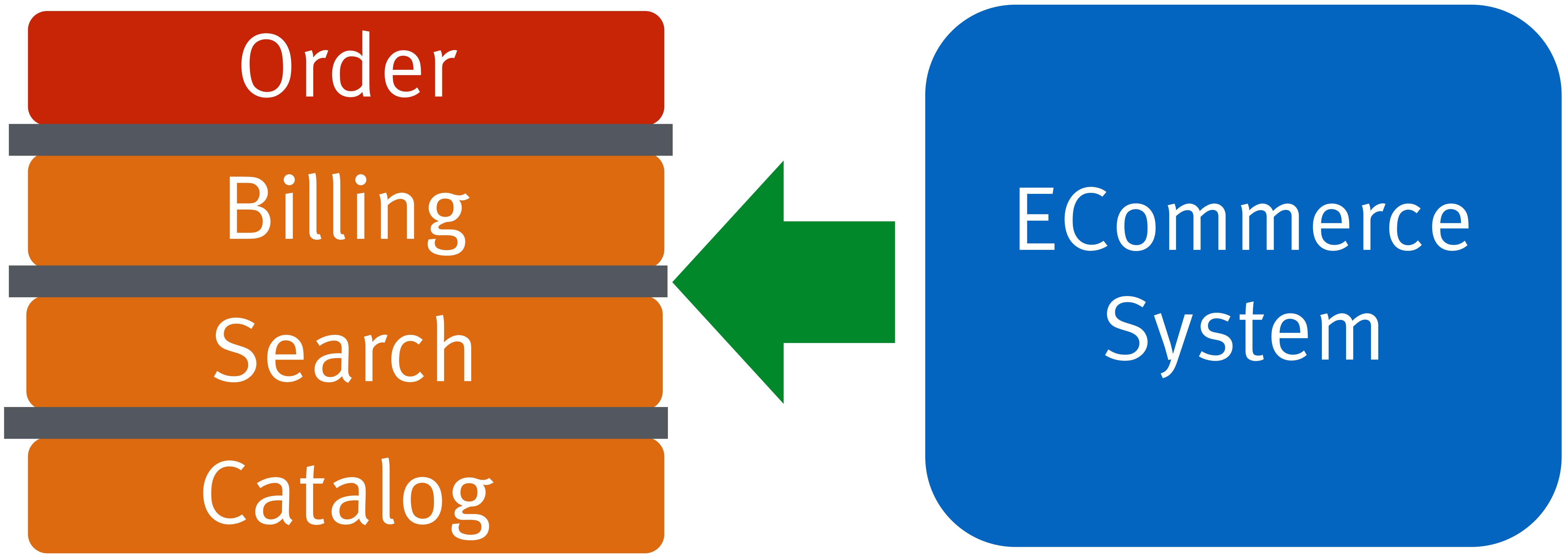
Hard to mess up

Components small



Hard to mess up

Components small



Hard to mess up

Replace if messed up.

Small,
independent deployable
modules
are recyclable.

Recycle your software!

How many people
are trying
to replace legacy
systems?

Replaceability
is usually no goal
for a software project.

Why??

We can achieve maintainability with clean architecture + clean code

WRONG

We can achieve maintainability with architecture firewalls + recyclable modules

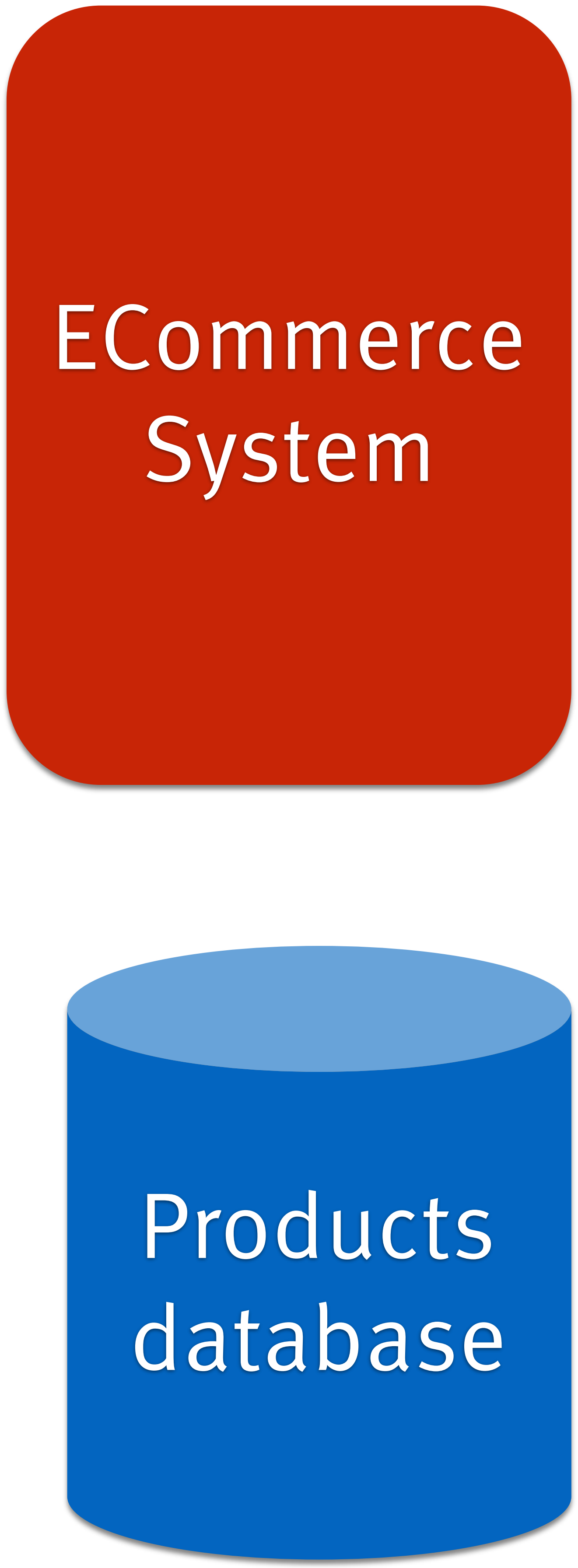
Maintainability✓

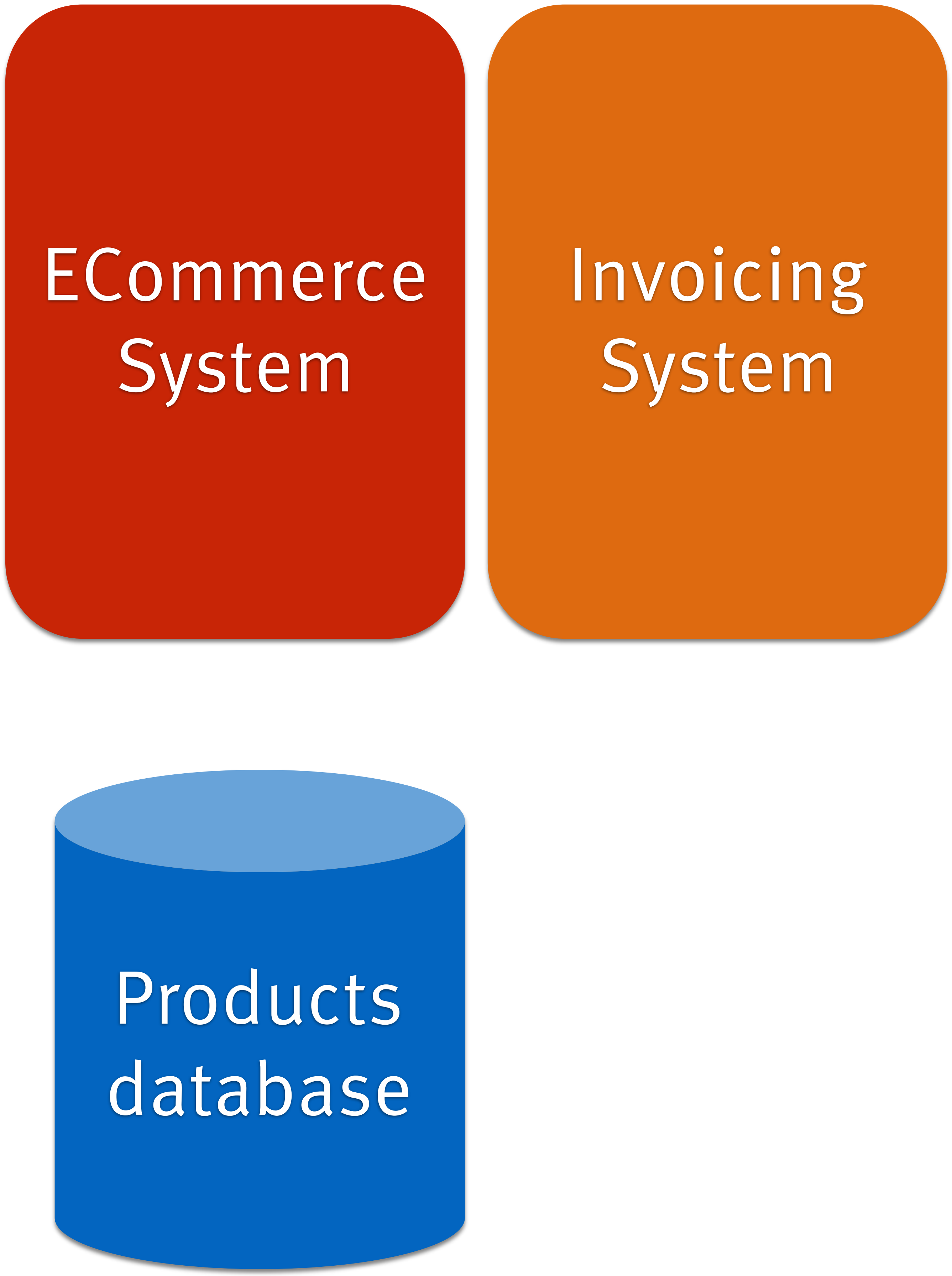
Redundancy

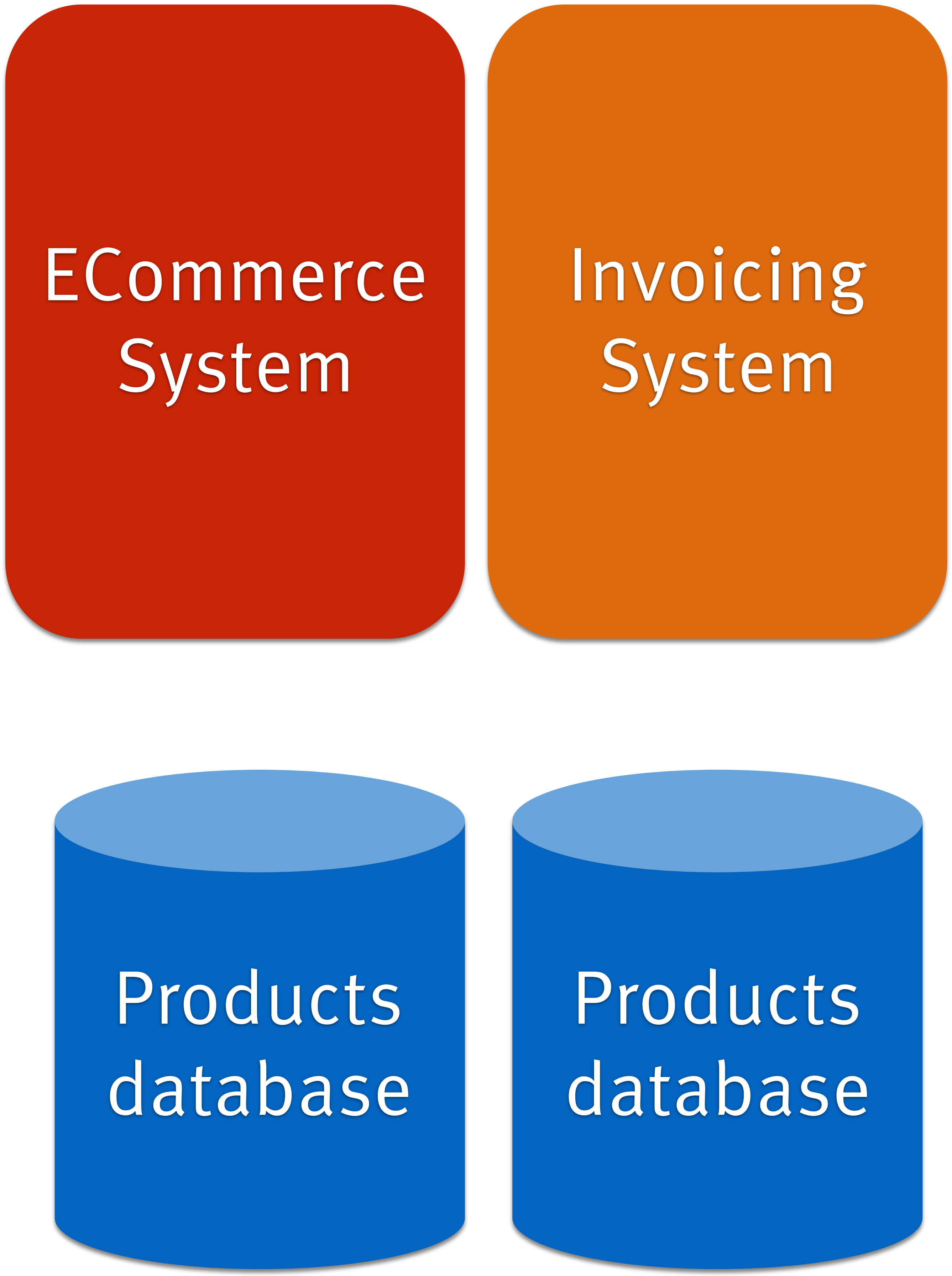
Redundancy
Redundant data

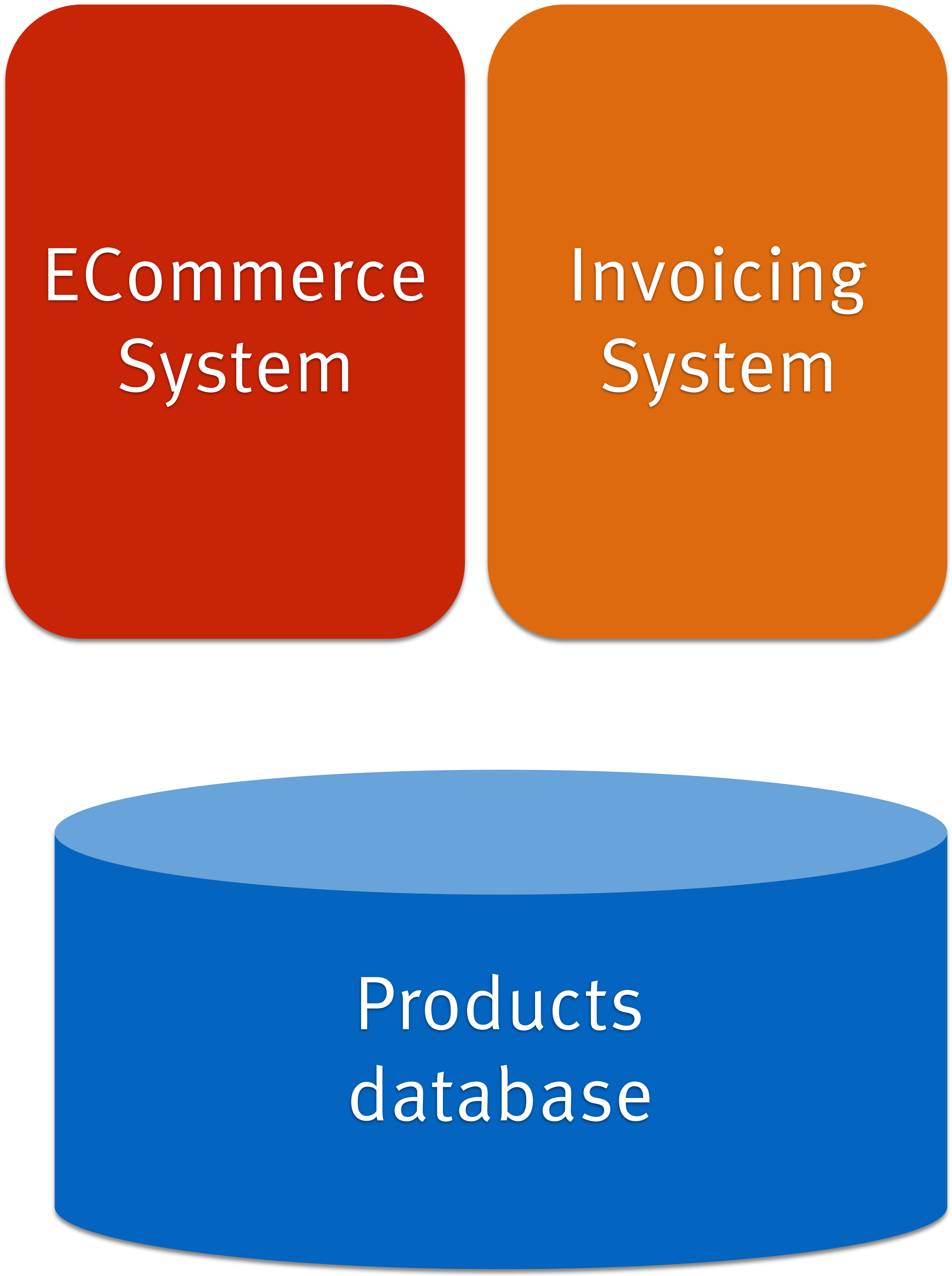
Every information should be stored and updated in one place.

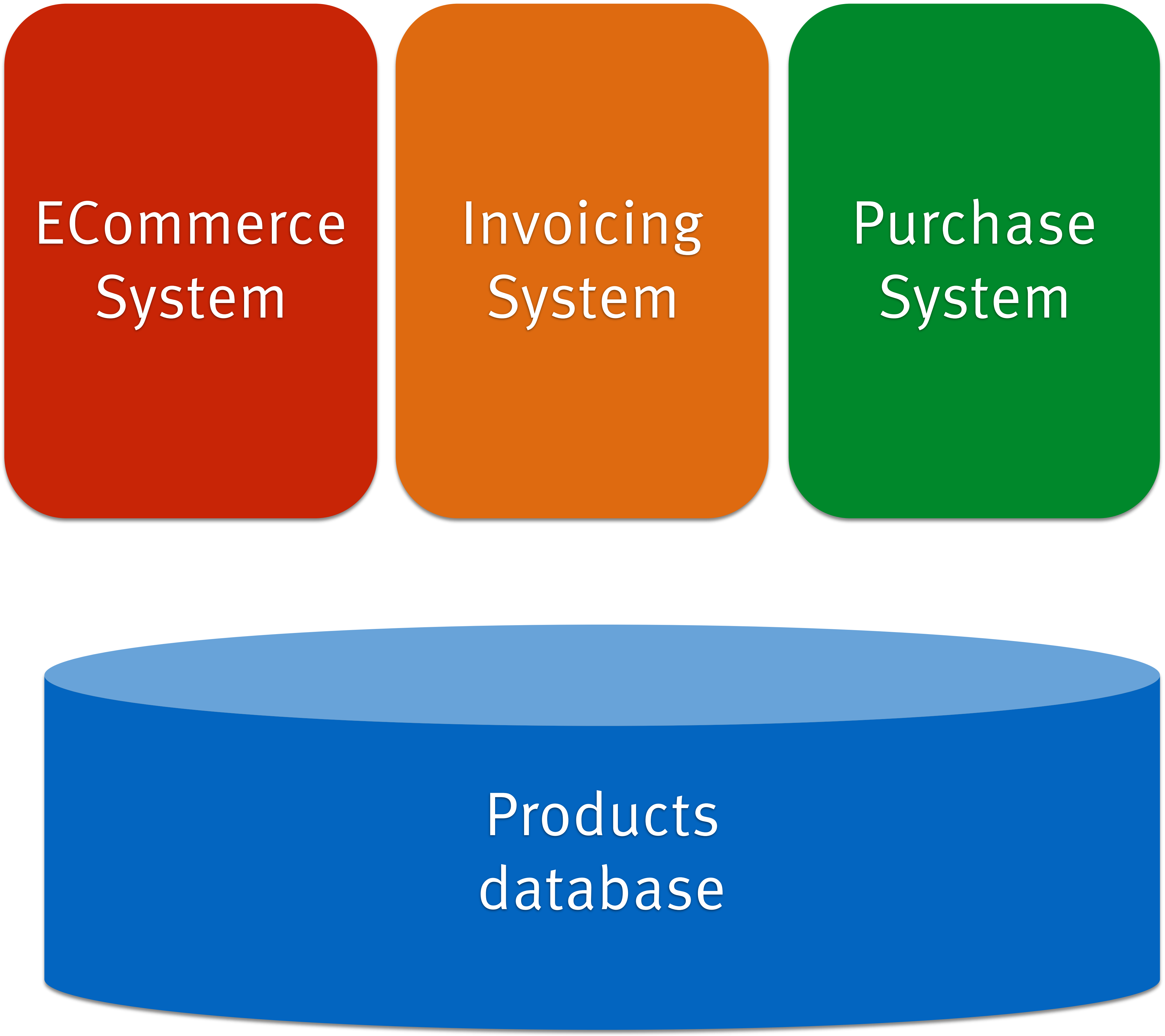
No redundancy for our product data!

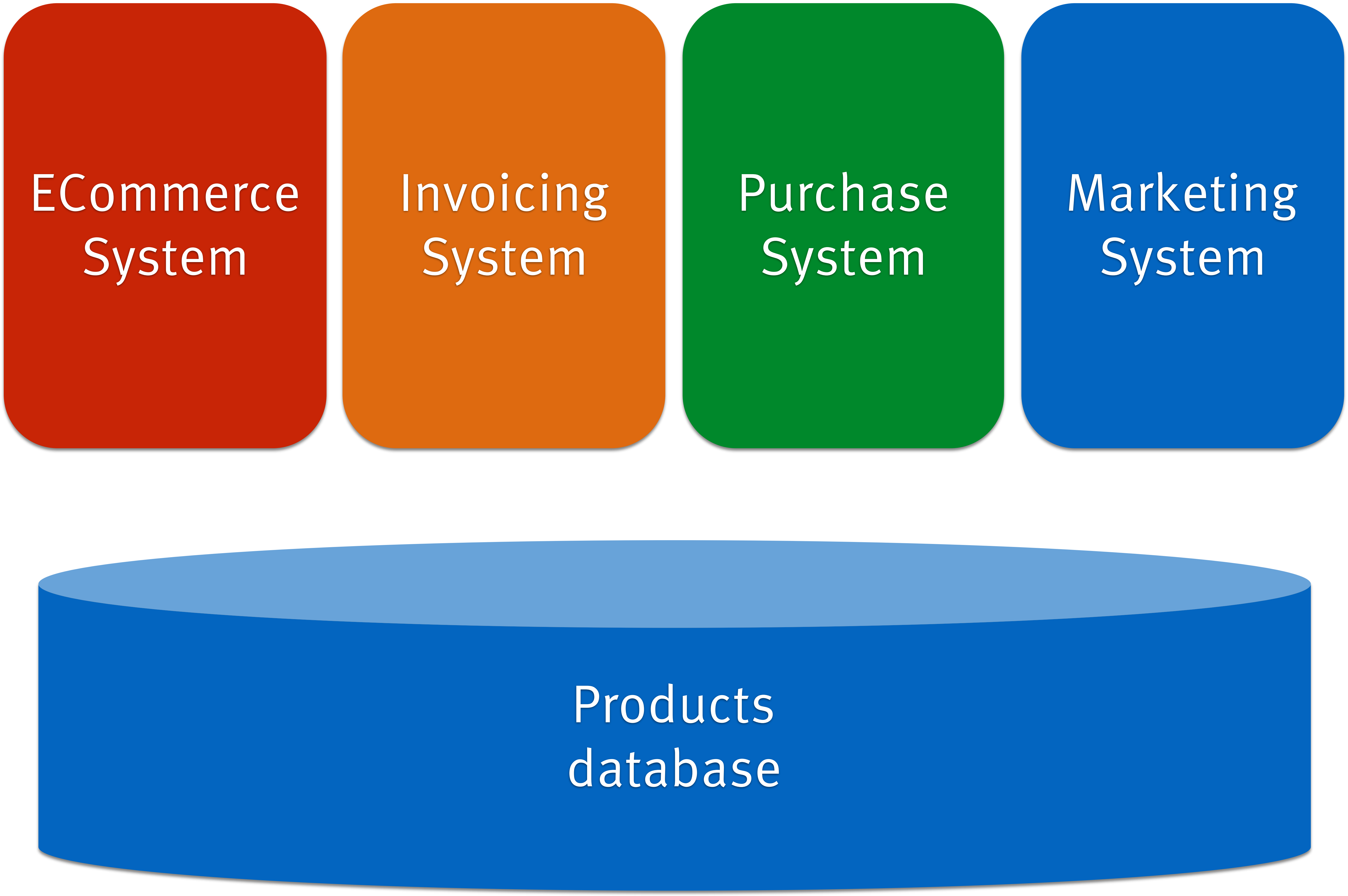

ECommerce System
Products database


ECommerce System
Invoicing System
Products database

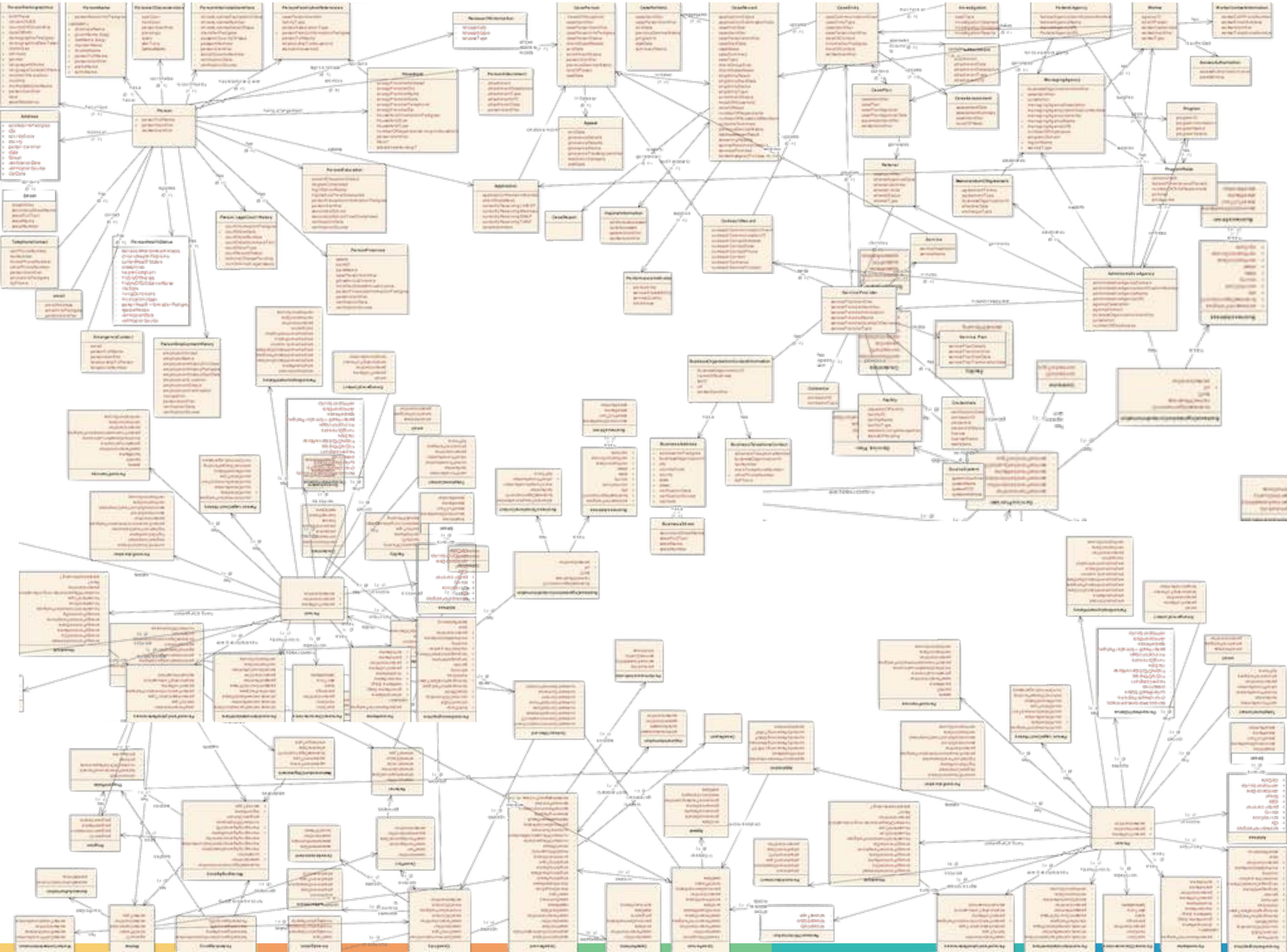

ECommerce System
Invoicing System
Products database
Products database


ECommerce System
Invoicing System
Products database


ECommerce System
Invoicing System
Purchase System
Products database


ECommerce System
Invoicing System
Purchase System
Marketing System
Products database

Products data model?



Person
Address
TelephoneContact
EmergencyContact
PersonHealthStatus
PersonEmploymentHistory
PersonEducation
PersonFinances
Application
CaseReport
CaseRecord
CaseEntity
Investigation
FederalAgency
Worker
ManagingAgency
AdministrativeAgency
ServiceProvider
Program
ProgramRules
Referral
Service
Facility
Contractor
BusinessAddress
BusinessTelephoneContact
Appeal
ContactRecord
CasePlan
Service Plan

No redundancies

High complexity

Hard to change

A central,
redundancy-free data model
is the optimum.

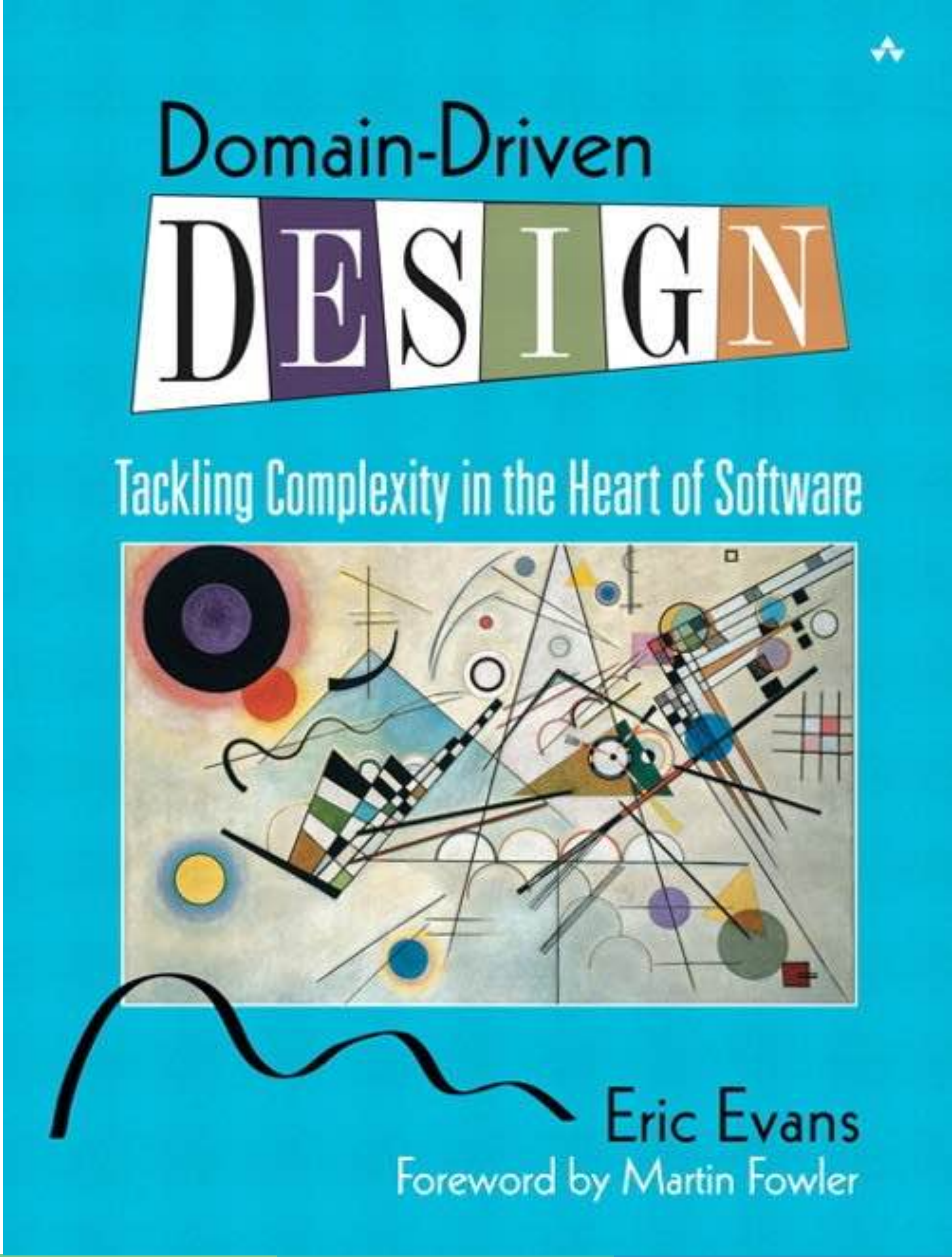
A central, redundancy free data model is the optimum.

WRONG

UBIQUITOUS LANGUAGE

VALUE OBJECT

ENTITY


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

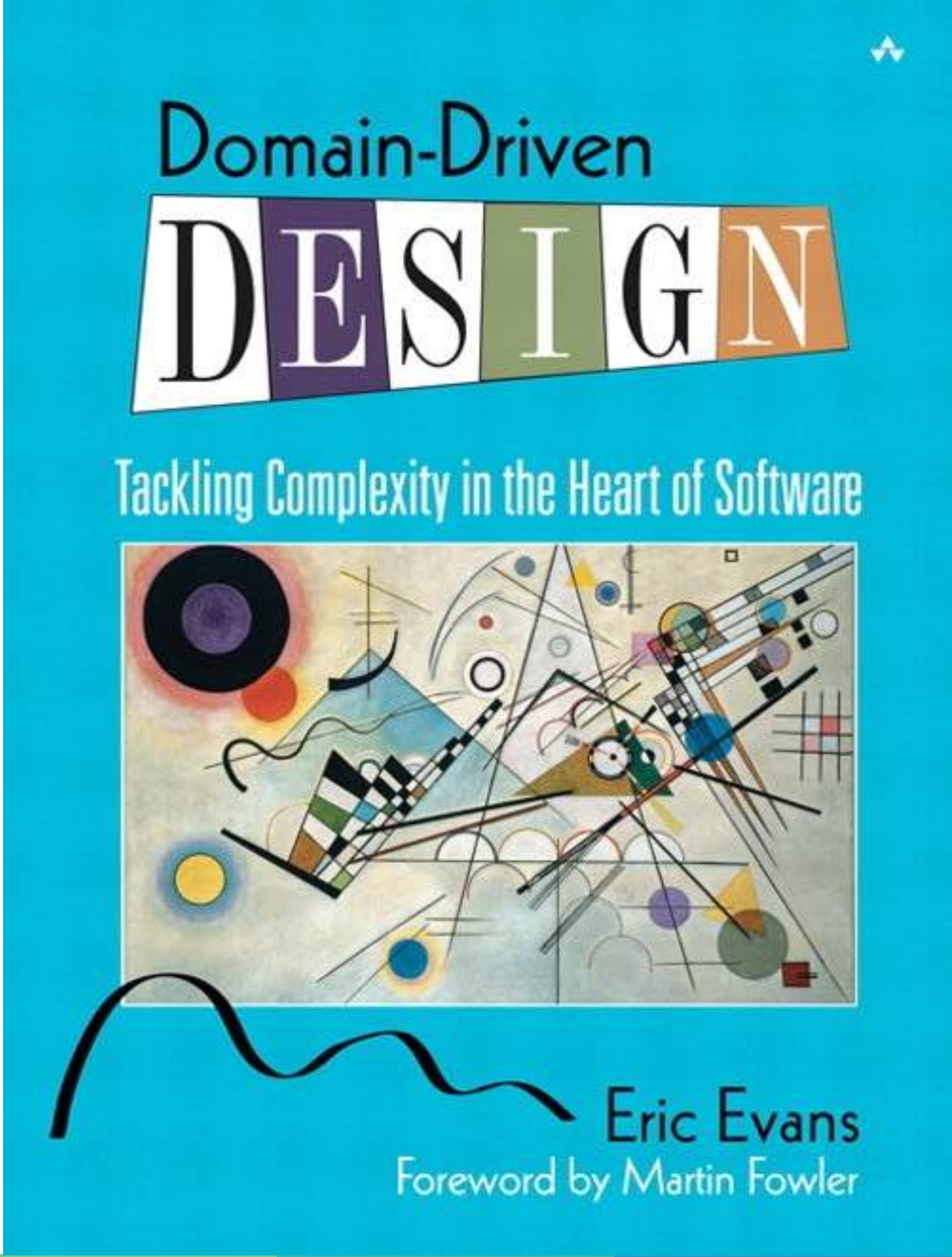
Address

VALUE OBJECT

or

ENTITY

529 pages
Part IV
Chapter 14


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

A domain model
is only useful
in a Bounded Context.

There is no
universal data model
in a large system.

Let me repeat:

There is no
universal data model
in a large system.

Address for a customer

Value Object

or

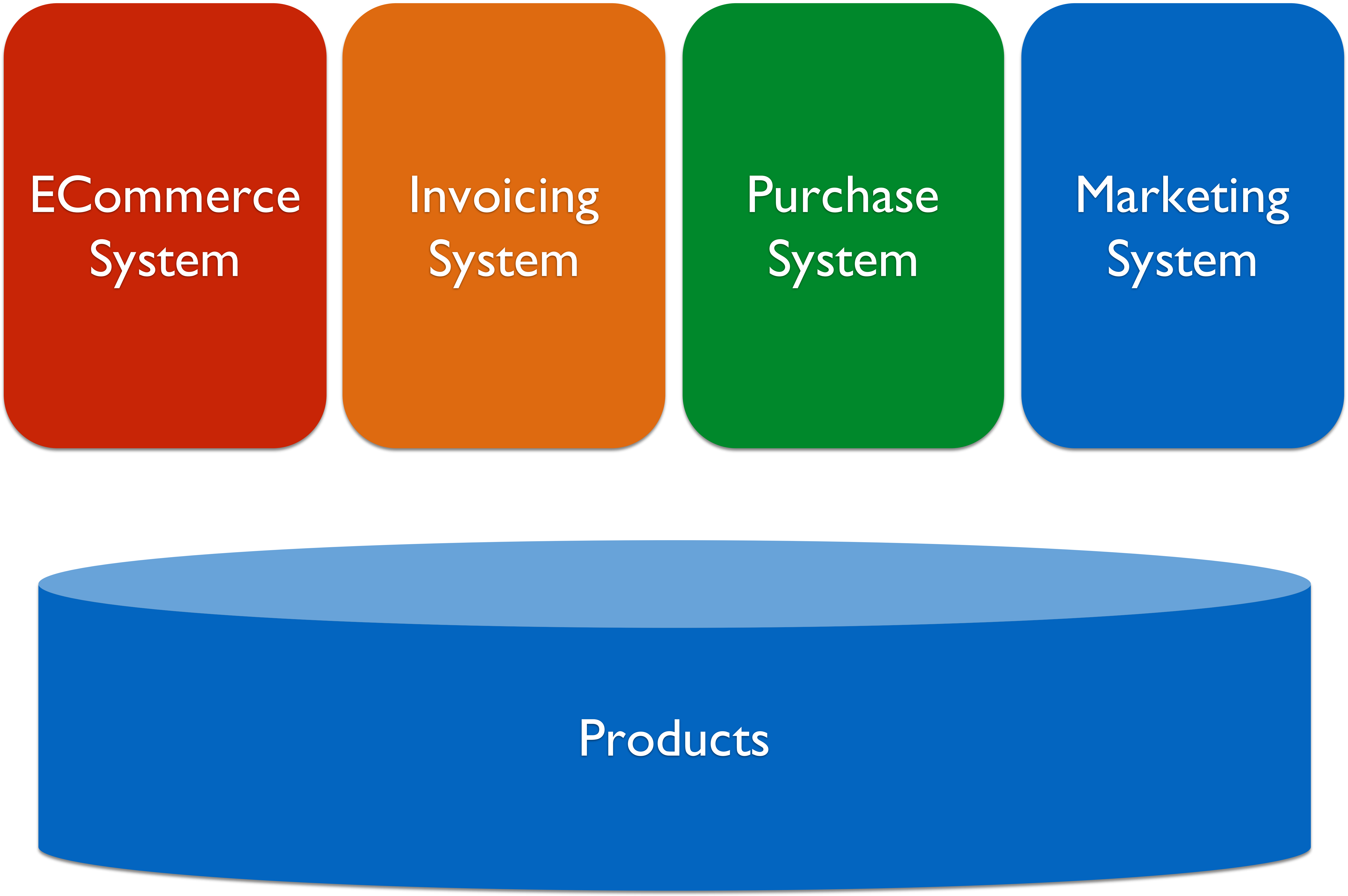
Entity

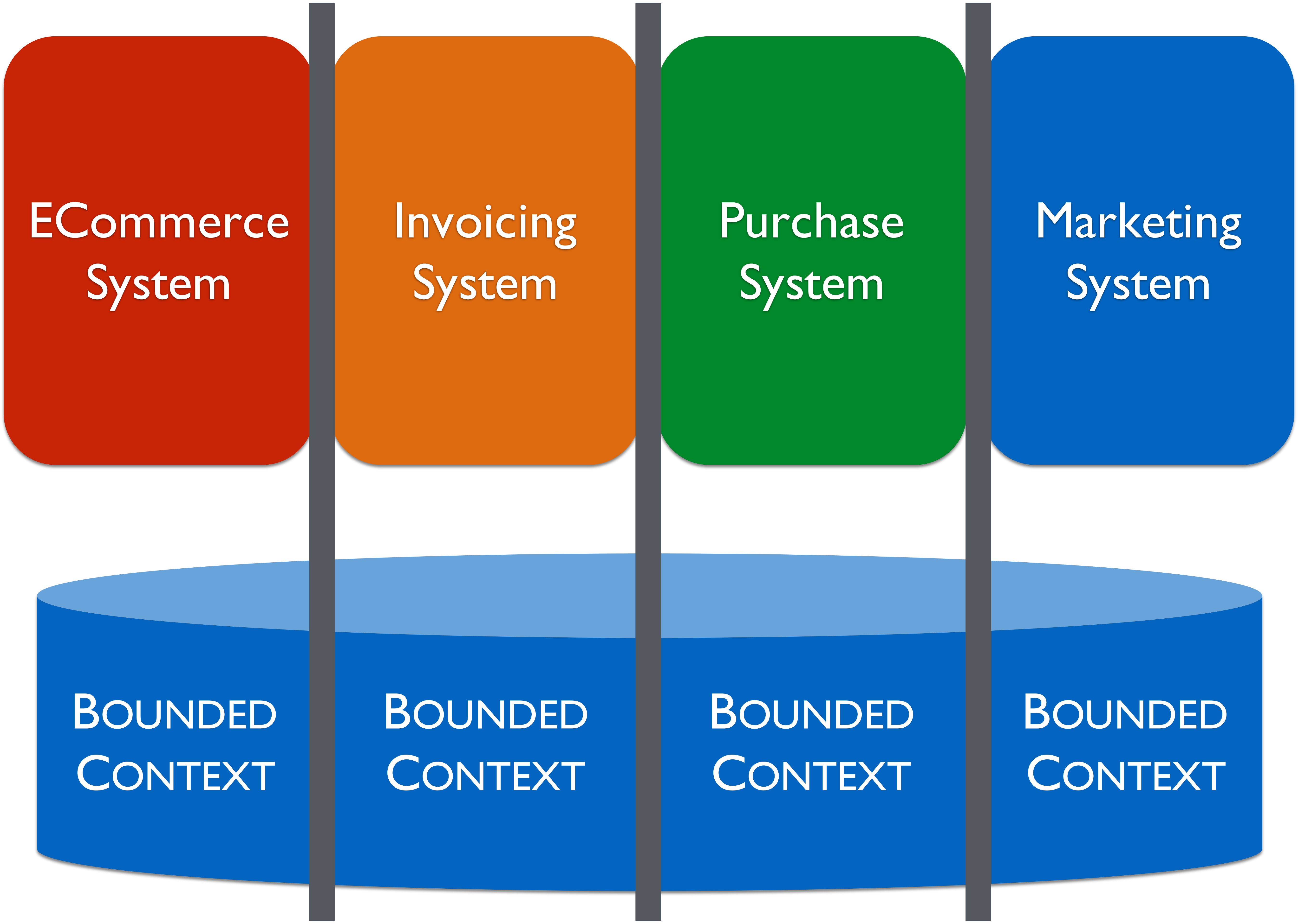
Address for calculating the drones’ routes

VALUE OBJECT

or

ENTITY


ECommerce System
Invoicing System
Purchase System
Marketing System
Products


ECommerce System
Invoicing System
Purchase System
Marketing System
Bounded Context
Bounded Context
Bounded Context
Bounded Context

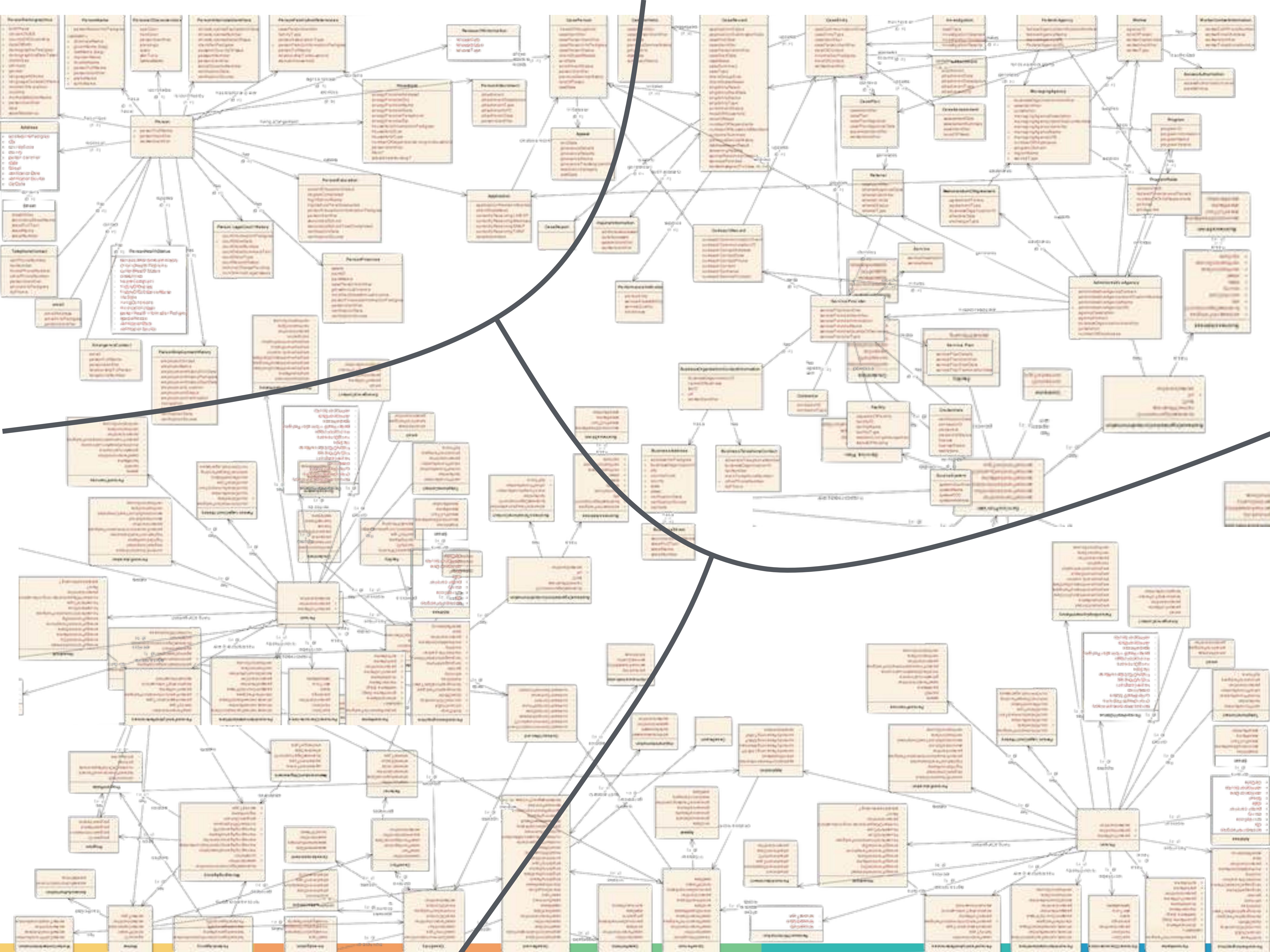
Create a model
for each BOUNDED CONTEXT.

Each Bounded Context
can be a Microservice
with its own database schema

Low complexity

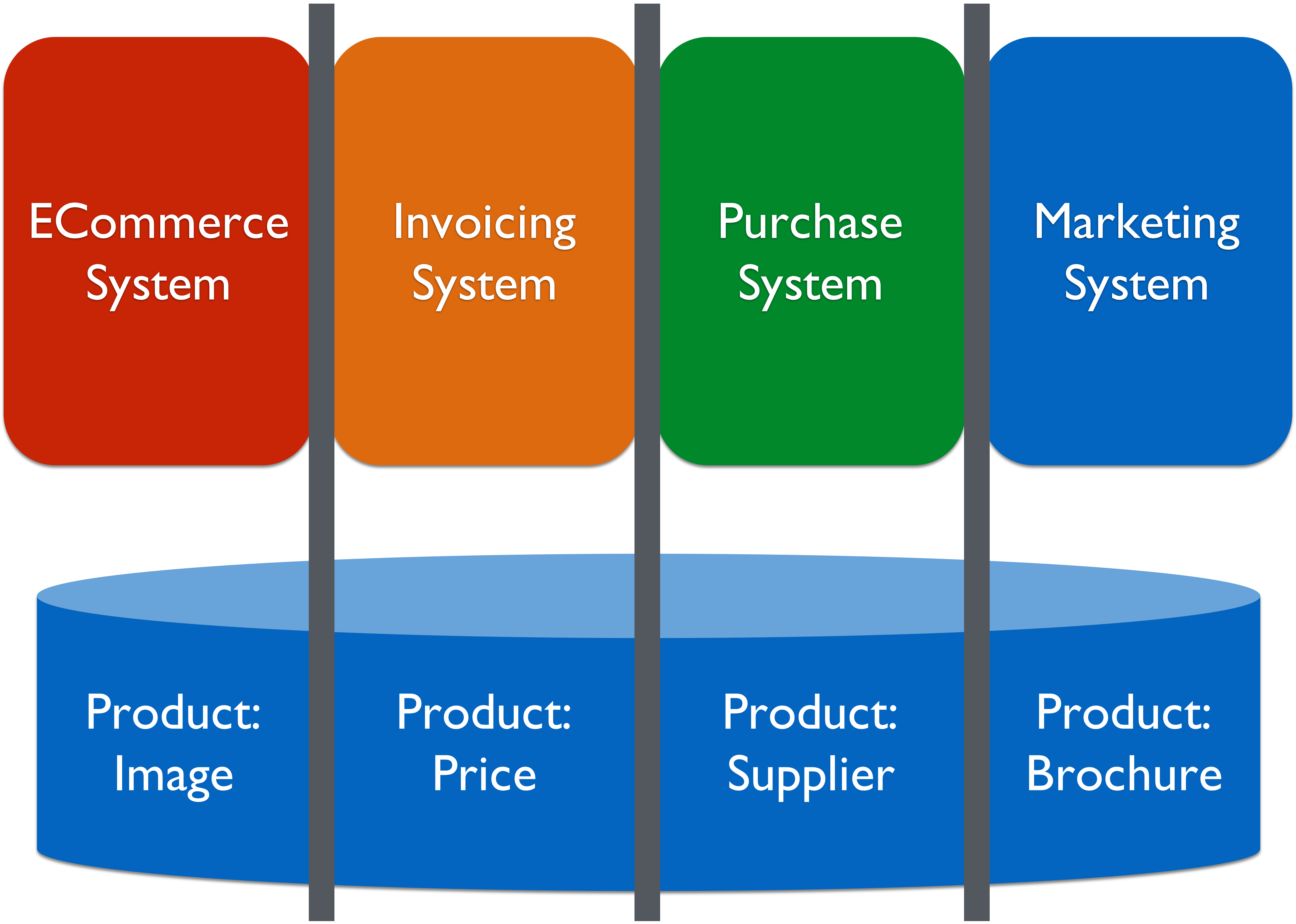
Easy to change

i.e. easy to maintain



Few redundancies

Separate facets


ECommerce System
Invoicing System
Purchase System
Marketing System
Product: Image
Product: Price
Product: Supplier
Product: Brochure

A central, redundancy free data model is the optimum.

WRONG

A central,
“redundancy-free”
data model
is often hard to maintain
and wrong.

Redundancy
Redundant data ✓

Redundancy
Redundant code

Redundant code: The ultimate sin

- Fix bug in many different place
- Decisions implemented in many places
- ...and hard to change

DRY

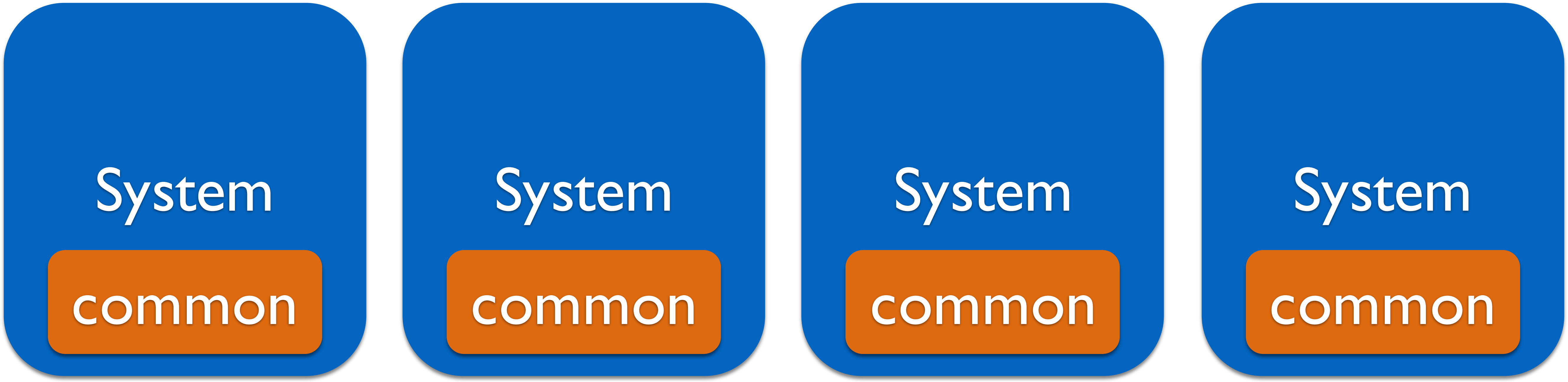
Don’t

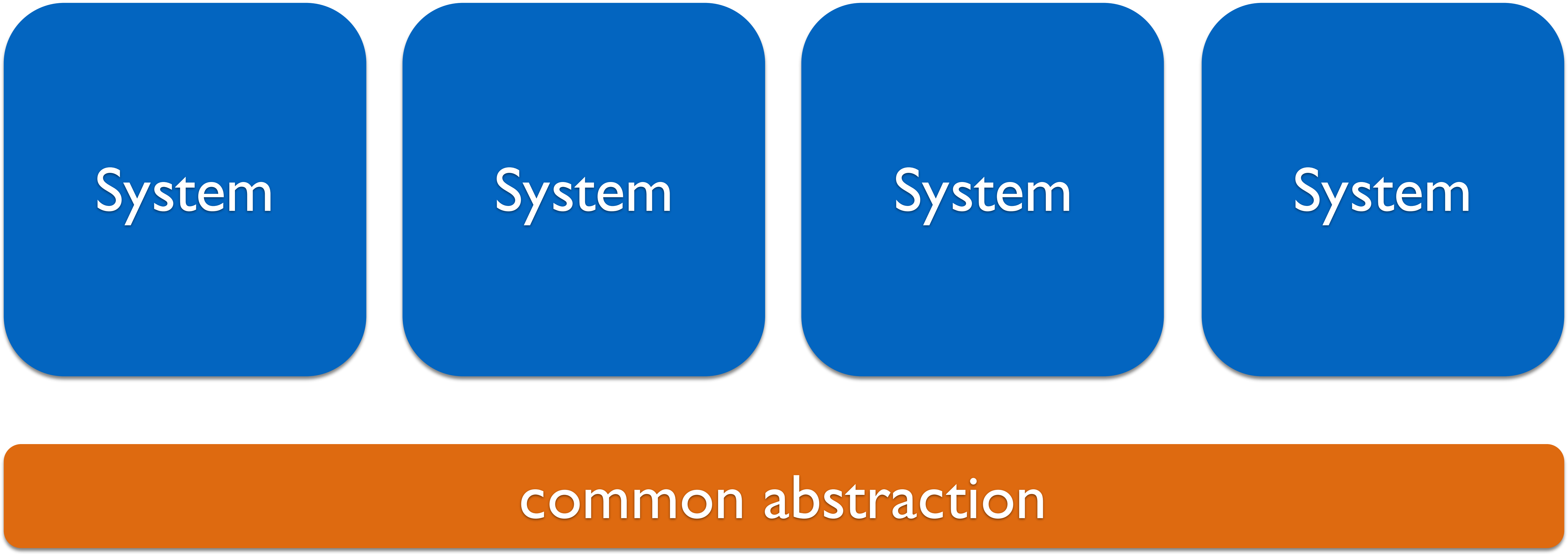
Repeat

Yourself

DRY Systems?
Great!

DRY between systems?
DRY is a trade-off


System
common
System
common
System
common
System
common


System
System
System
System
common abstraction

Reuse:
The Holy Grail
of the nineties

So where are all the
reusable internal
frameworks?

Premature optimization,
that’s like a sneeze.
Premature abstraction
is like Ebola;
it makes my eyes bleed.

Christer Ericson

The entire history of
software engineering
is that of the
rise in levels of abstraction.

Grady Booch

Using code is hard.

Reusing code is almost impossible.

But we are reusing Open Source all the time!

Create an Open Source project!

Open Source

- Good code quality
- Documentation
- Model to accept contributions

“But high quality Open Source is hard.

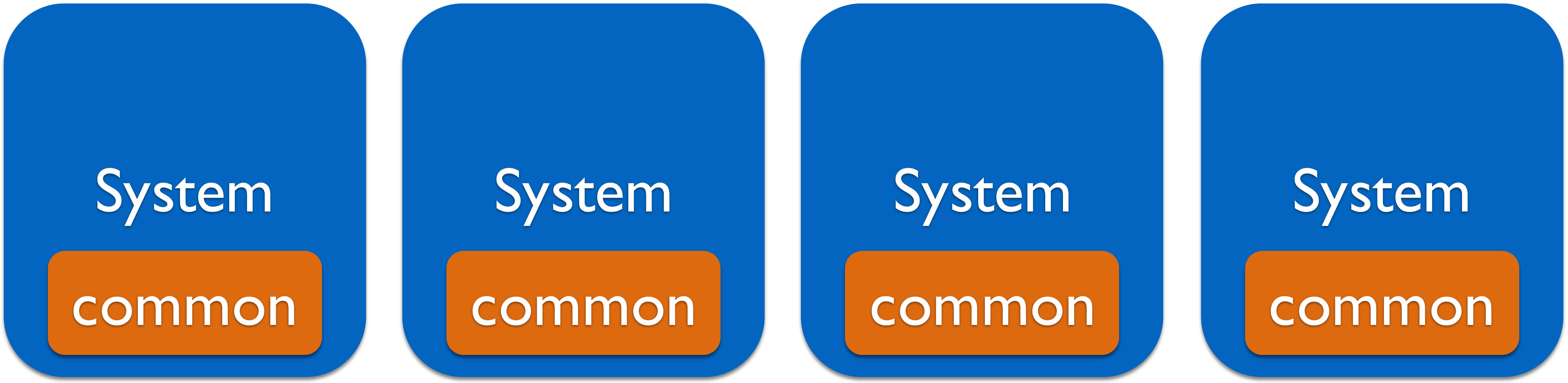
We just share code!”

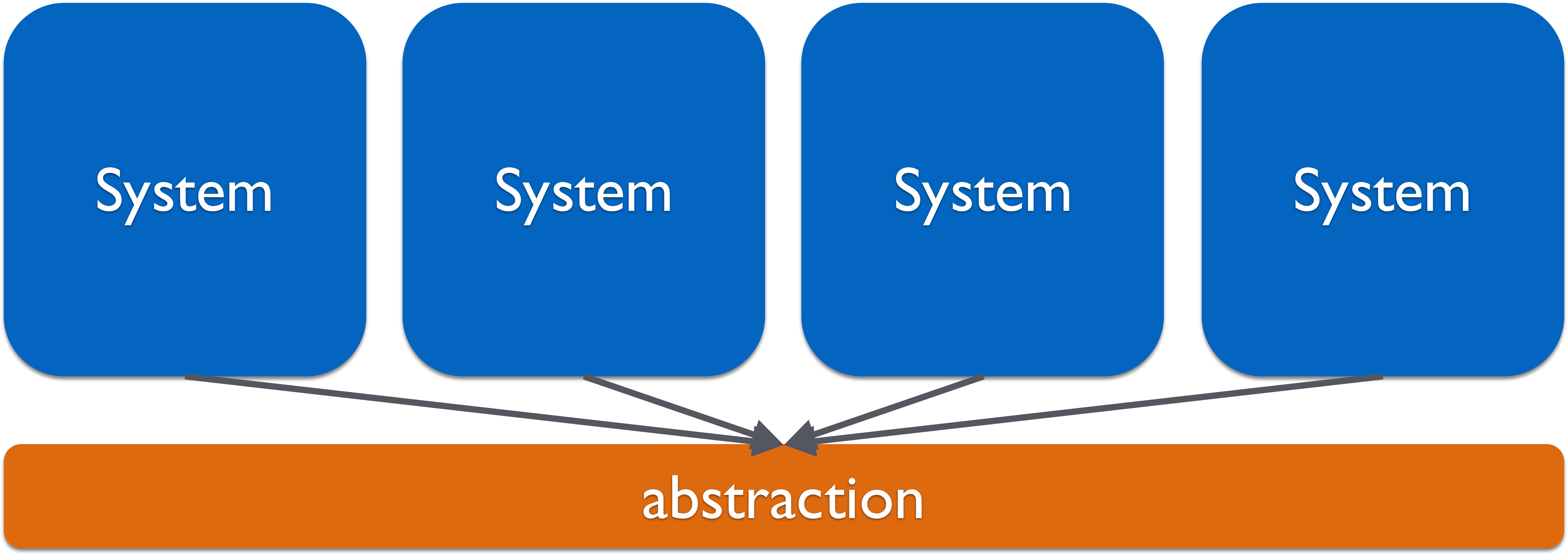
“You only provide high quality as Open Source…

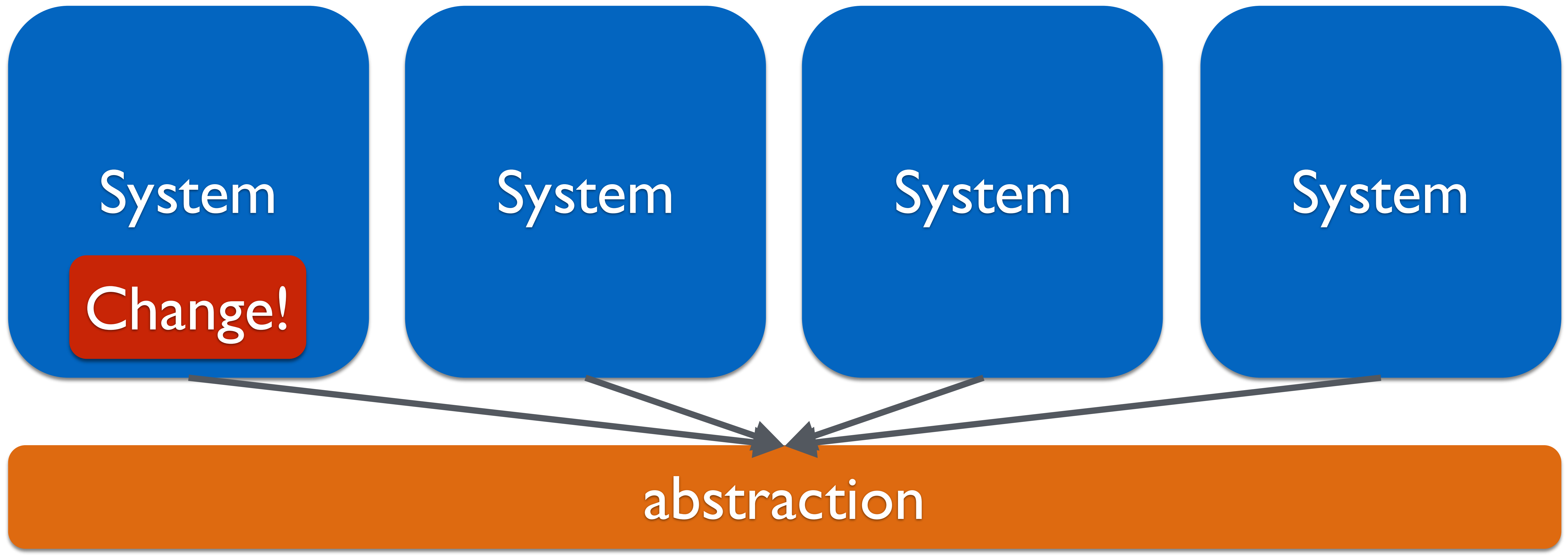
…but for colleagues low quality is OK?”

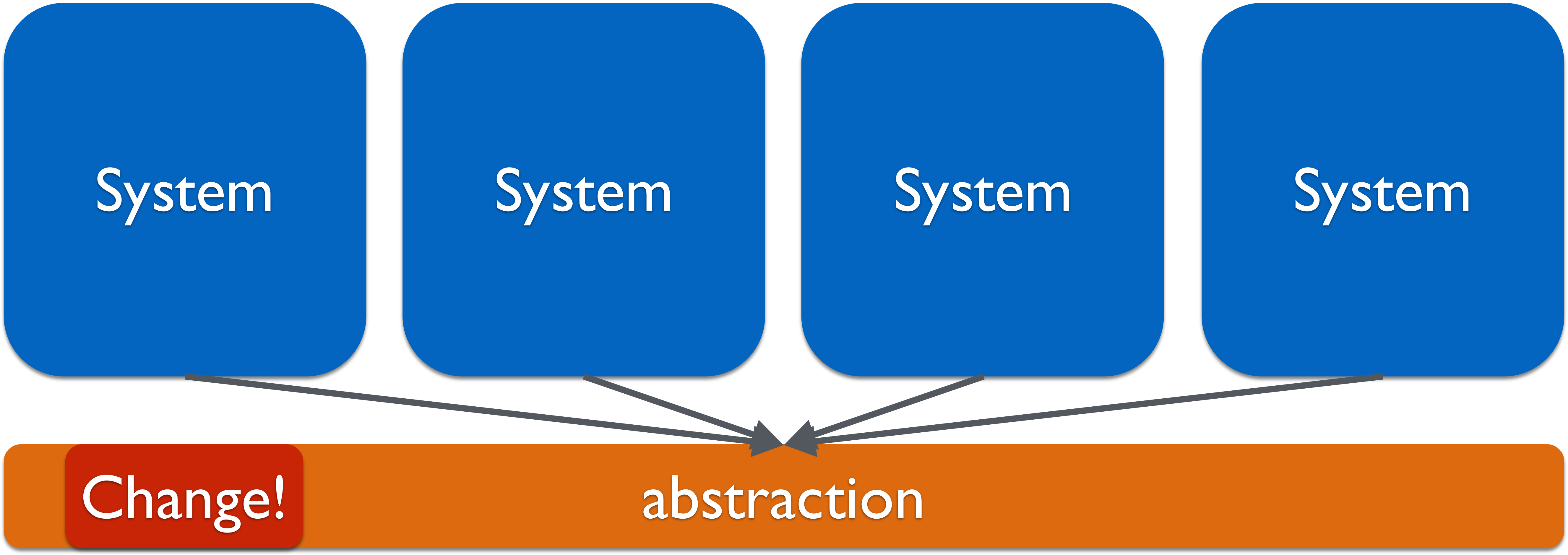
Let’s assume it’s possible to reuse code.

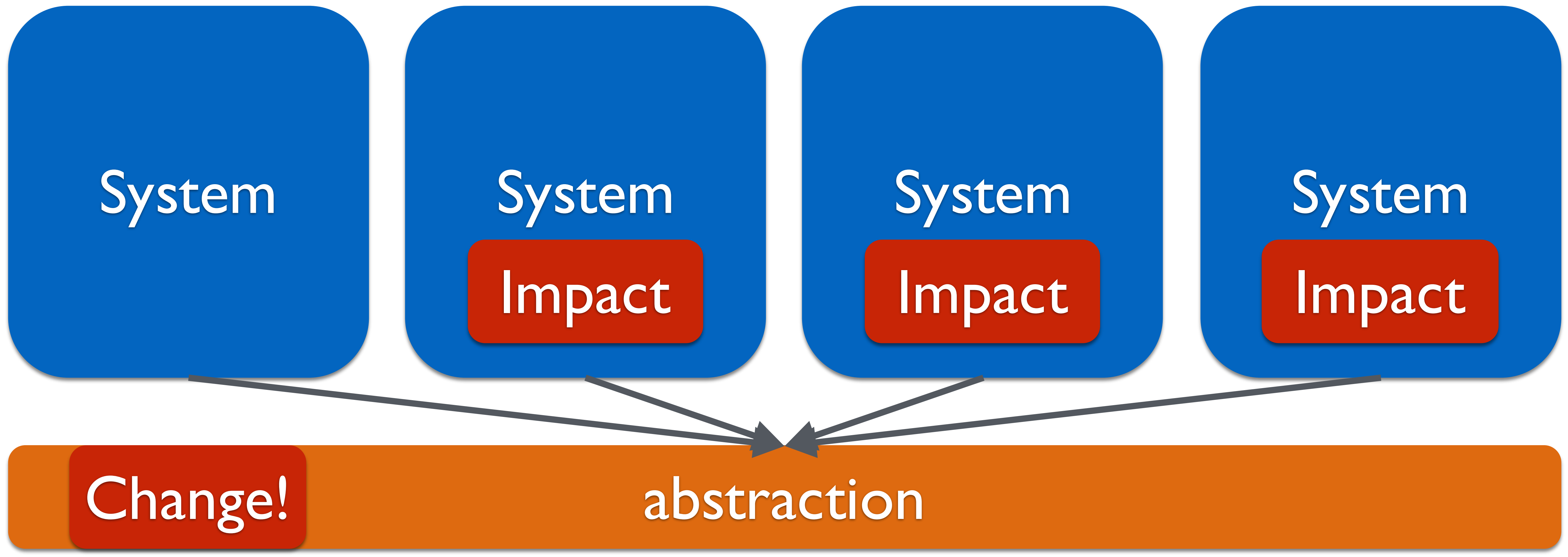
Reuse is still a tradeoff.

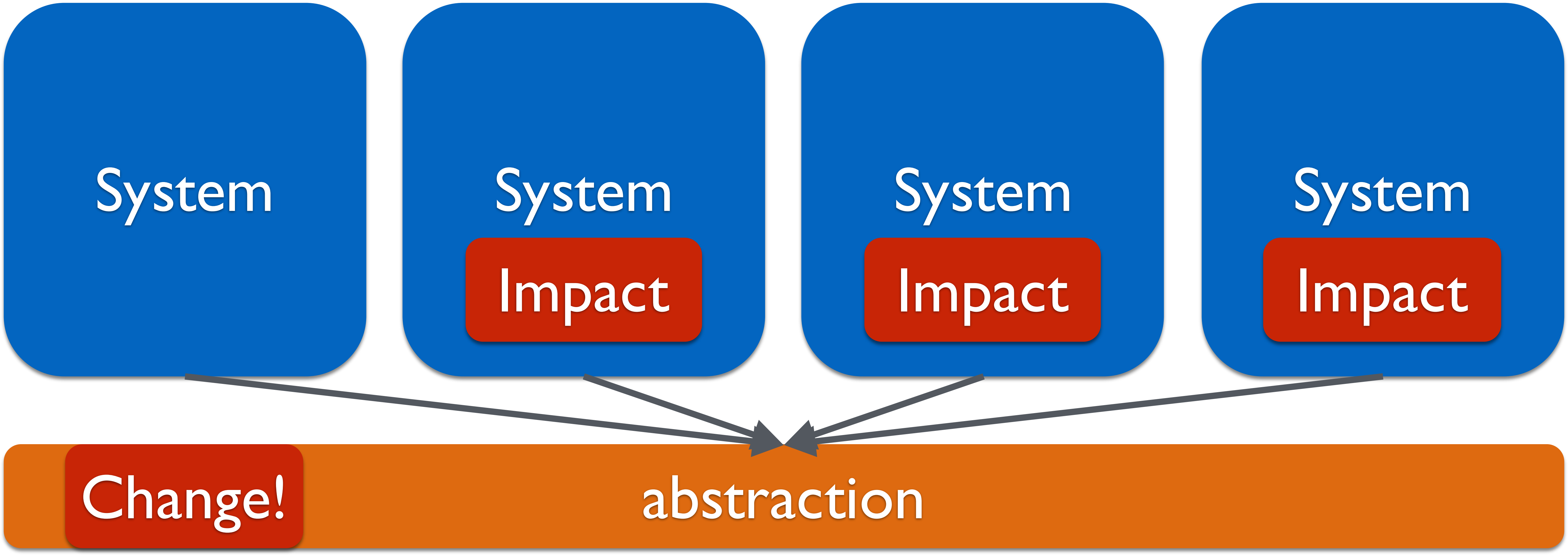

System
common
System
common
System
common
System
common


System
System
System
System
abstraction


System
Change!
System
System
System
abstraction


System
System
System
System
Change!
abstraction

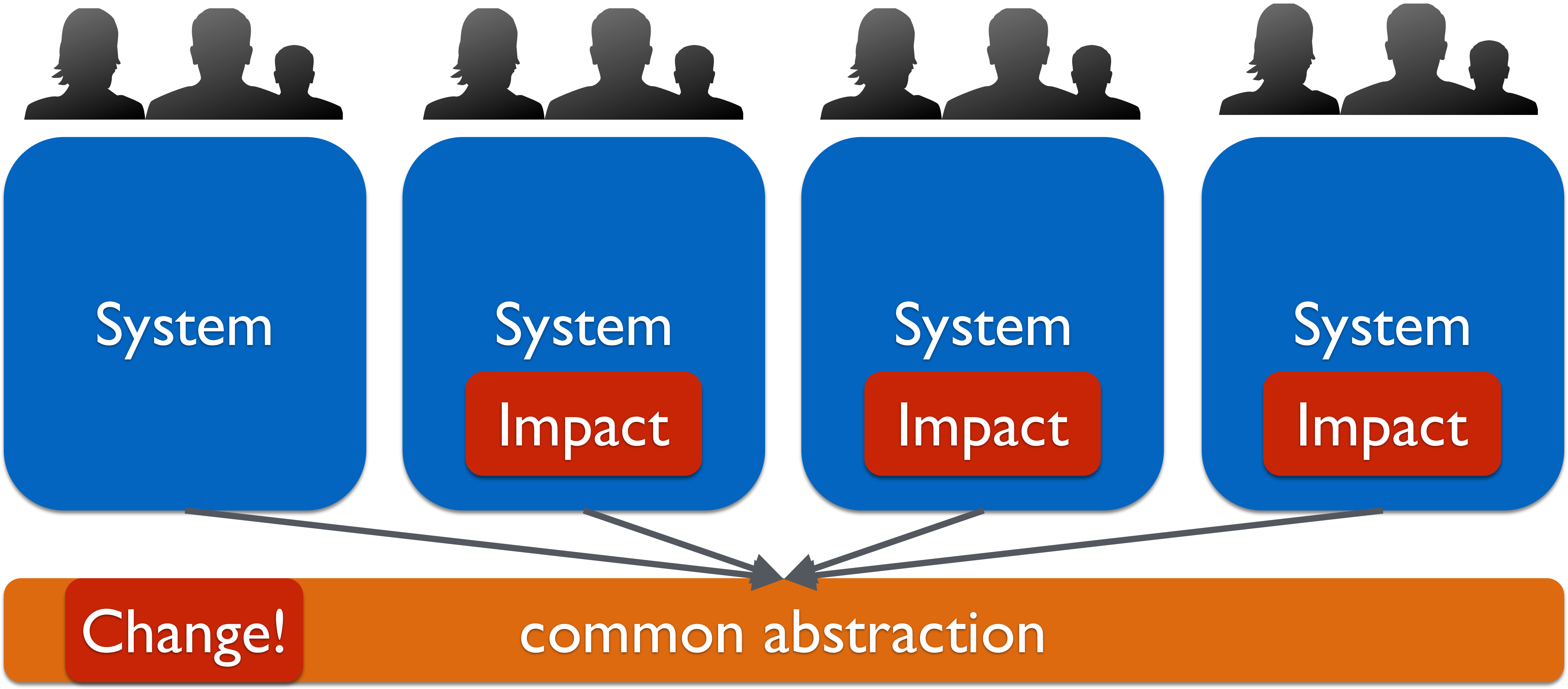

System
System
Impact
System
Impact
System
Impact
Change!
abstraction


System
System
Impact
System
Impact
System
Impact
Change!
abstraction

Now we have reuse

…and a dependency.

Dependency not just in software!


System
System
Impact
System
Impact
System
Impact
Change!
common abstraction

Dependency between teams

Coordination

Meetings

Getting no real work done

Reuse is a tradeoff:

Reuse vs.
Independence

Independence=
Easy to change=
Maintainability

Independence is important for self-organization.

Self-organization = deciding yourself

Not meetings upon meetings

Deciding yourself
is only possible,
if teams and modules
are independent.

Redundancies between systems must be avoided.

Redundancies between systems must be avoided.

WRONG

Reuse is a tradeoff:

Reuse vs.
Independence

Microservices focus on independence

The Microservices Manifesto ;-)

Microservices Manifesto ;-)

We value:
Replaceability over maintainability

Microservices Manifesto ;-)

We value:
BOUNDED CONTEXT over redundancy-free data

Microservices Manifesto ;-)

We value:
Independence over
“Don’t Repeat Yourself!”

Replaceability over maintainability

Bounded Context over redundant-free data

Independence over DRY