PREVIOUSLY ON ASD…

- Structure modelling:
    - Static behaviour
    - Assign responsibilities
- Behaviour modelling
    - Runtime behaviour
    - Interaction between objects
- GRASP principles
    - Good design at a fine-grain granularity

ADVANCED SOFTWARE DESIGN
LECTURE 6
SOFTWARE ARCHITECTURE

Kiko Fernandez

OVERVIEW

What **software architecture** is and why it is interesting

Who are the **stakeholders**

What **software qualities** does software architecture concern

UML diagrams expressing aspects of software architecture

Architectural styles or software architectural design patterns

SOFTWARE ARCHITECTURE

What is the purpose of software architecture?

Why not think about system (hardware) architecture?

SOFTWARE ARCHITECTURE

A Software Architecture defines:

- the components of the software system
- how the components use each other's functionality and data
- how control is managed between the components

The highest level of design – large-scale structure of solution

SOFTWARE ARCHITECTURE

A Software Architecture defines:

- constraints in the implementation
- inhibits / enables software quality attributes

SOFTWARE ARCHITECTURE

A Software Architecture defines:

- constraints in the implementation
- inhibits / enables software quality attributes

Functional vs non-functional requirements

SOFTWARE QUALITIES

What are the various software qualities (= non-functional requirements) that software architecture is concerned with?

KEY SOFTWARE QUALITIES

correctness

usability

integrity

reliability

efficiency

security

safety

maintainability

flexibility

extensibility

scalability

testability

modularity

interoperability

reusability

portability

https://en.wikipedia.org/wiki/List_of_system_quality_attributes

KEY SOFTWARE QUALITIES

correctness

usability

integrity

reliability

efficiency

security

safety

maintainability

flexibility

extensibility

scalability

testability

modularity

interoperability

reusability

portability

What does it mean?

https://en.wikipedia.org/wiki/List_of_system_quality_attributes

KEY SOFTWARE QUALITIES

correctness

usability

integrity

reliability

efficiency

security

safety

extensibility

scalability

testability

reusability

portability

MatchCare
Can we guarantee efficiency in the sense of:
"there is a response in X minutes?"

What does it mean?

https://en.wikipedia.org/wiki/List_of_system_quality_attributes

SOFTWARE QUALITIES

accessibility

accountability

adaptability

administrability

affordability

agility

auditability

autonomy

availability

compatibility

composability

correctness

debugability, testability

degradability

dependability

deployability

discoverability

distributability

durability

effectiveness

efficiency

evolvability

extensibility

failure transparency

fault-tolerance, resilience

flexibility

inspectability

installability

integrity

interchangeability

interoperability

learnability

maintainability

manageability

mobility

modifiability

modularity

operability

orthogonality

portability

predictability

recoverability

reliability

reproducibility

responsiveness

reusability

robustness

safety

scalability

seamlessness

self-sustainability

serviceability

securability

simplicity

stability

standards compliance

survivability, sustainability

timeliness, relevance

traceability

ubiquity

understandability

upgradability

usability

https://en.wikipedia.org/wiki/List_of_system_quality_attributes

STAKEHOLDERS

Which stakeholders have an interest in a software development effort?

SOFTWARE QUALITIES

Which software qualities are of interest to which stakeholders?

MATCHCARE STAKEHOLDERS

MATCHCARE STAKEHOLDERS

Users (elder people):

Client (pays the system):

Care Takers:

MATCHCARE STAKEHOLDERS

Users (elder people):

- Reliable service
- Usability
- Effective service (good care taker)

Client (pays the system):

Care Takers:

MATCHCARE STAKEHOLDERS

Users (elder people):

- Reliable service
- Usability
- Effective service (good care taker)

Client (pays the system):

- Schedule and budget
- Reliable
- Secure payment / encryption

Care Takers:

MATCHCARE STAKEHOLDERS

Users (elder people):

- Reliable service
- Usability
- Effective service (good care taker)

Client (pays the system):

- Schedule and budget
- Reliable
- Secure payment / encryption

Care Takers:

- Reliable
- Privacy
- Usability

CONFLICTING QUALITIES

Which software qualities are conflicting?

Why?

How can these conflicts be resolved?

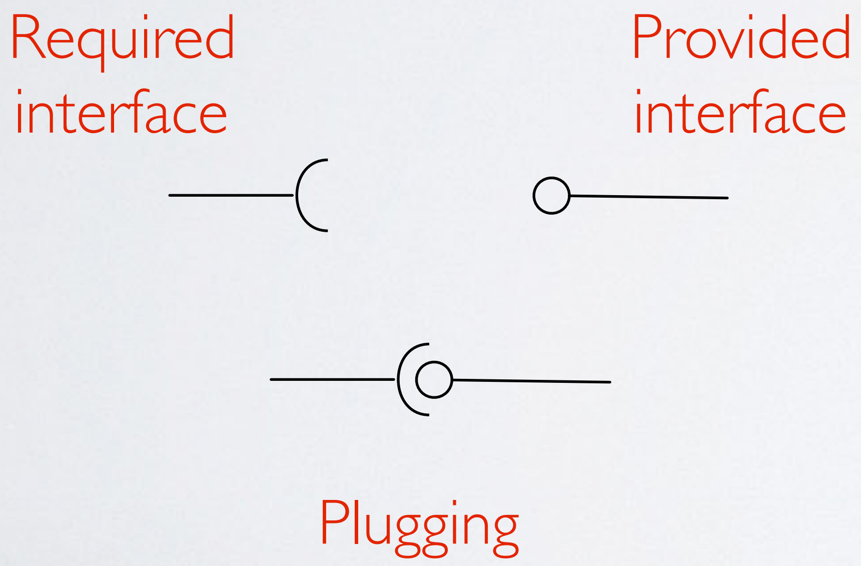
UML DIAGRAMS

- Component diagrams
- Package diagrams
- Deployment diagrams

COMPONENT DIAGRAMS

Component – a set of related operations that share a common purpose

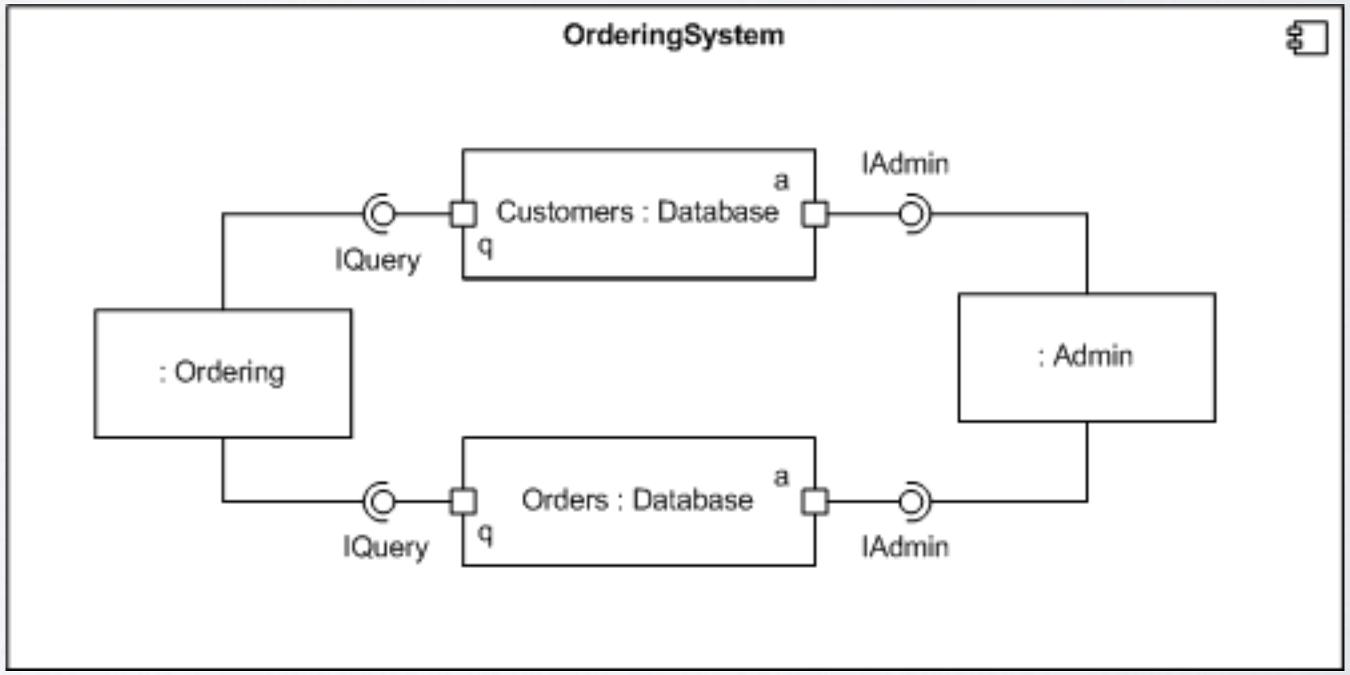
Interface – the set of operations available to other sub-systems



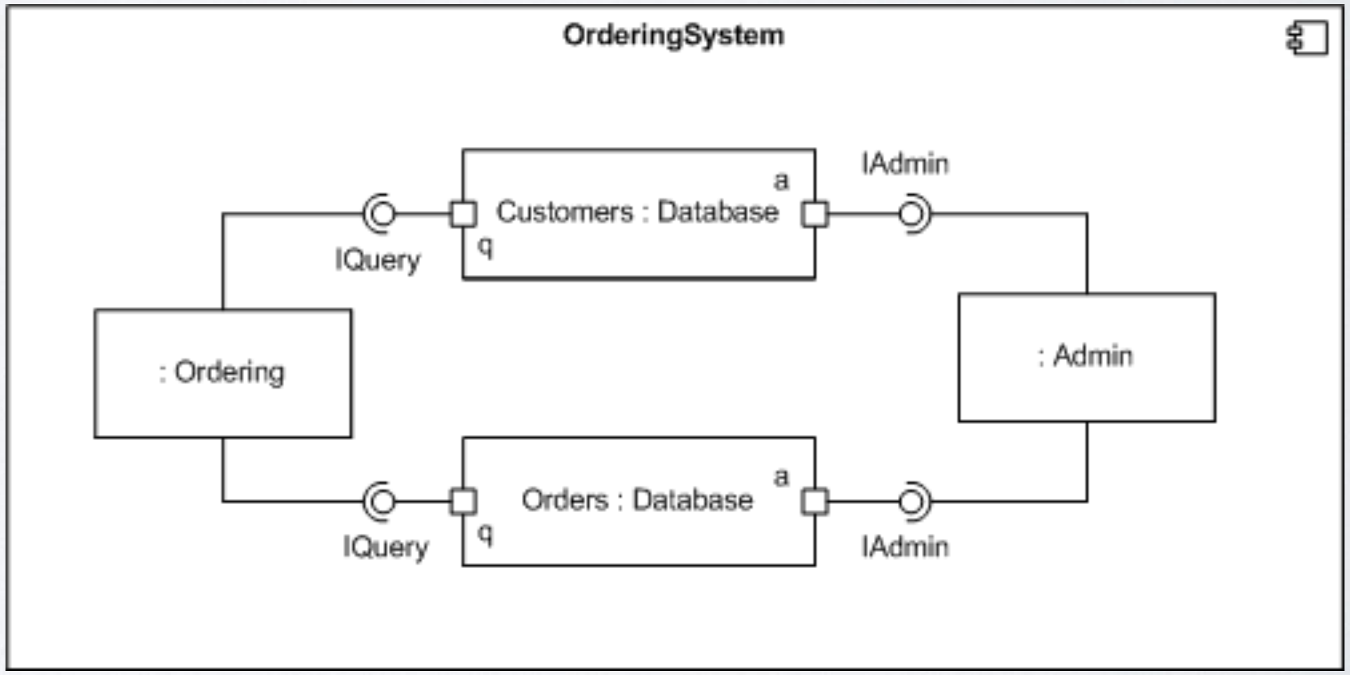
Required interface

Provided interface

Plugging

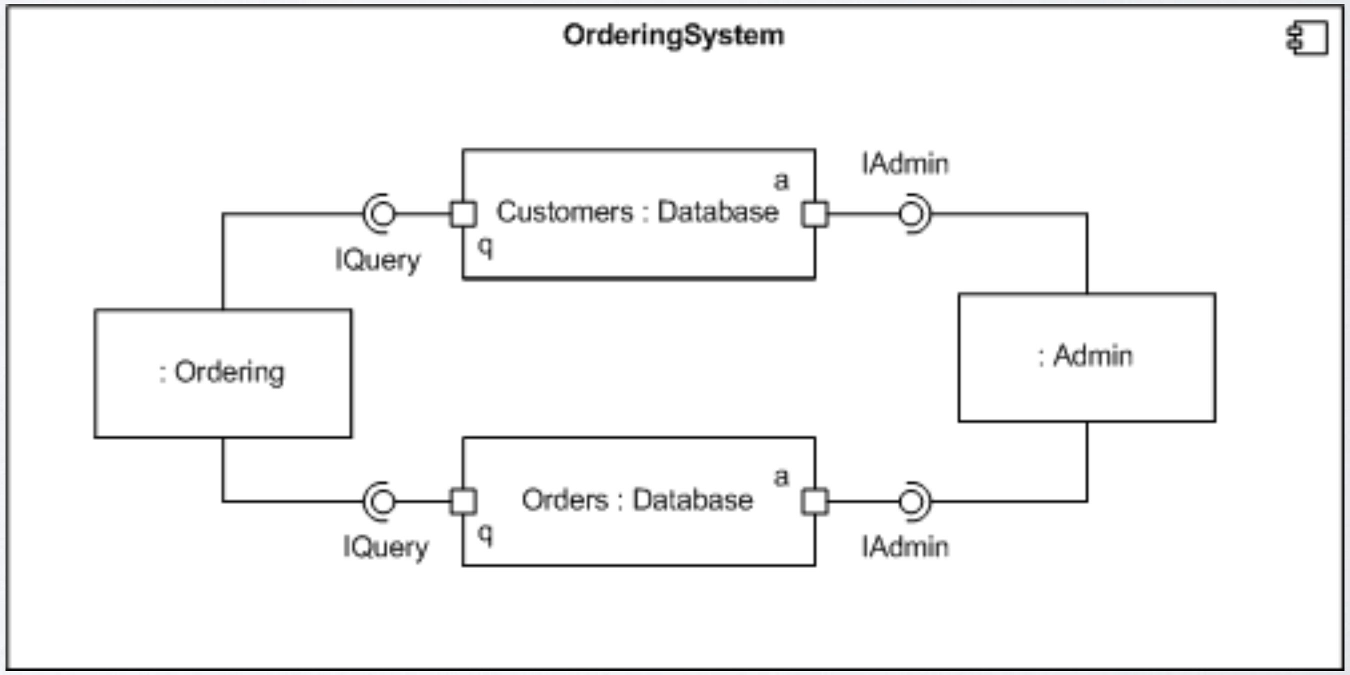


COMPONENT DIAGRAMS



COMPONENT DIAGRAMS

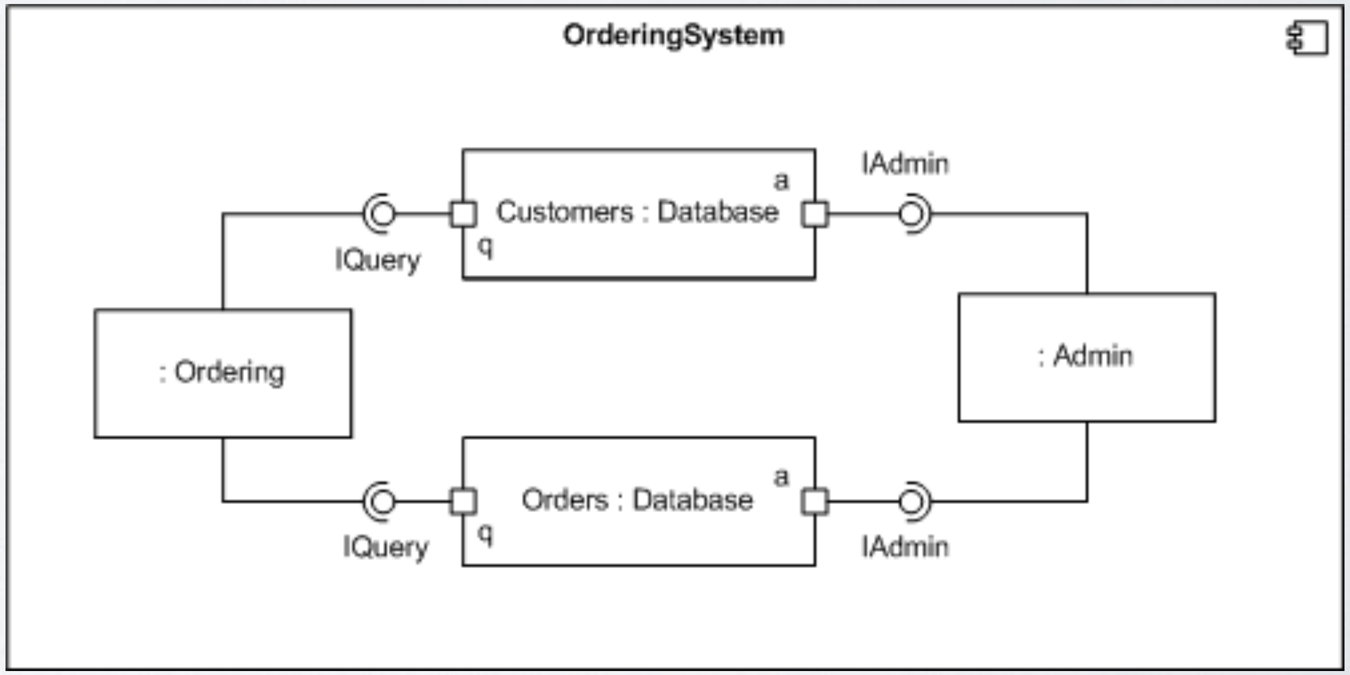
```
interface IQuery {
  void create(Object o);
  void read(int id);
  void update(int id, Object o);
  void destroy(Object o);
}
```



COMPONENT DIAGRAMS

```
interface IQuery {
  void create(Object o);
  void read(int id);
  void update(int id, Object o);
  void destroy(Object o);
}
```

```
class Customers implements IQuery {
  void create(Object o){ … }
  void read(int id){ … }
  void update(int id, Object o){ … }
  void destroy(Object o){ … }
  // other methods
  …
}
```

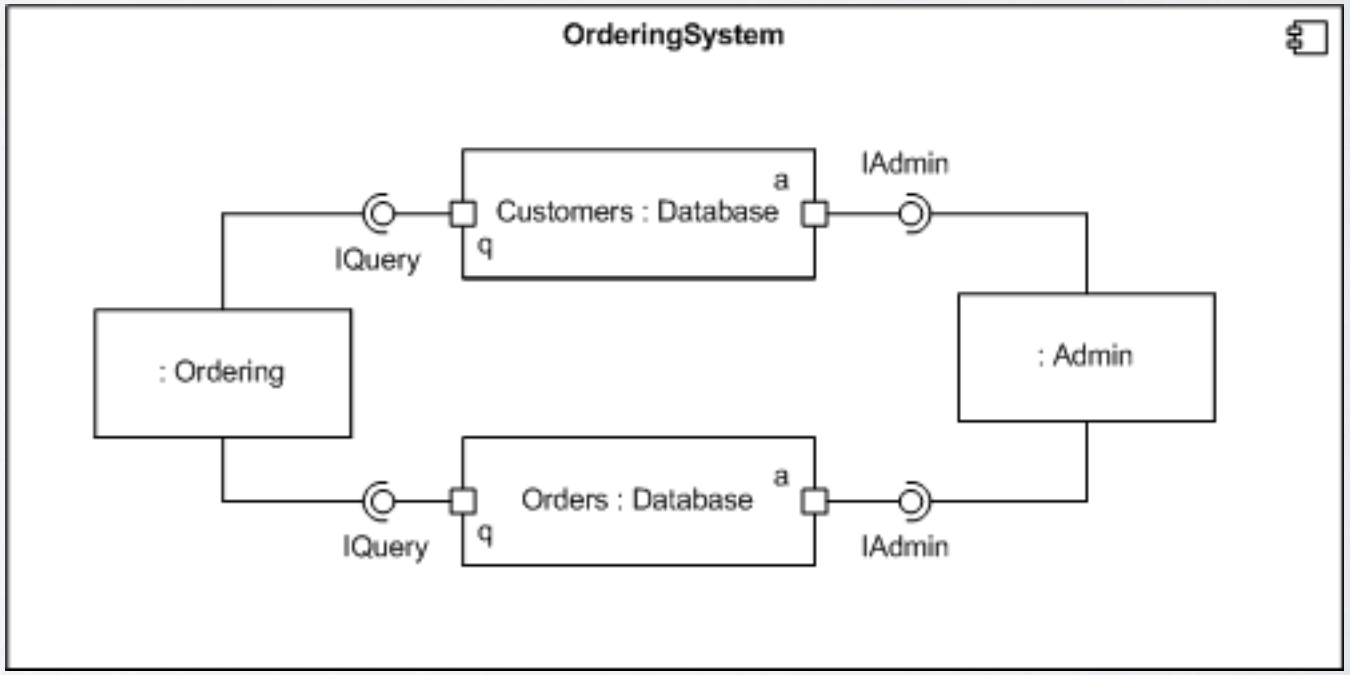


COMPONENT DIAGRAMS

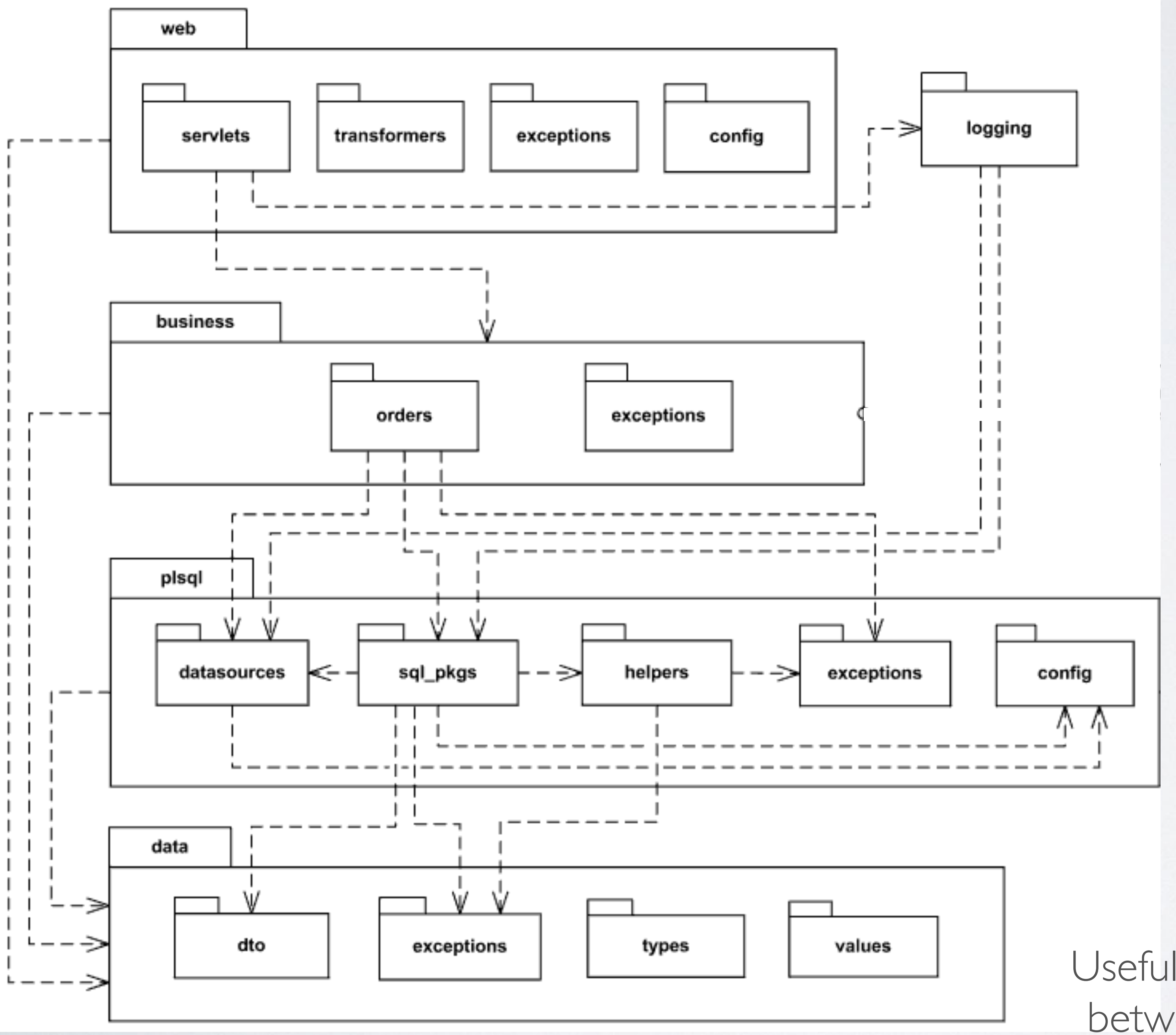
```
interface IQuery {
  void create(Object o);
  void read(int id);
  void update(int id, Object o);
  void destroy(Object o);
}
```

```
class Ordering {
  // attributes
  customers: IQuery
  …
  // methods
  // methods that use the IQuery interface
}
```

```
class Customers implements IQuery {
  void create(Object o){ … }
  void read(int id){ … }
  void update(int id, Object o){ … }
  void destroy(Object o){ … }
  // other methods
  …
}
```



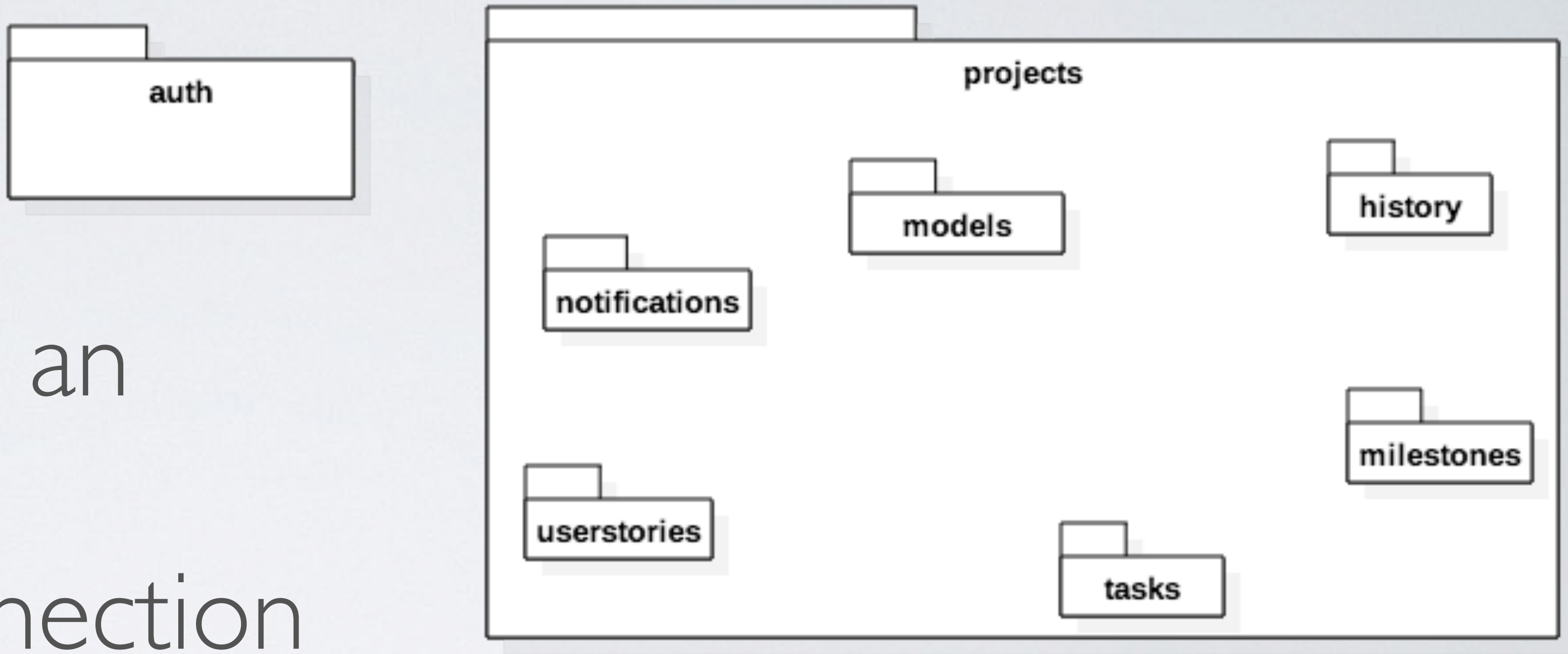
PACKAGE DIAGRAMS



Packages
Hierarchy
Dependency

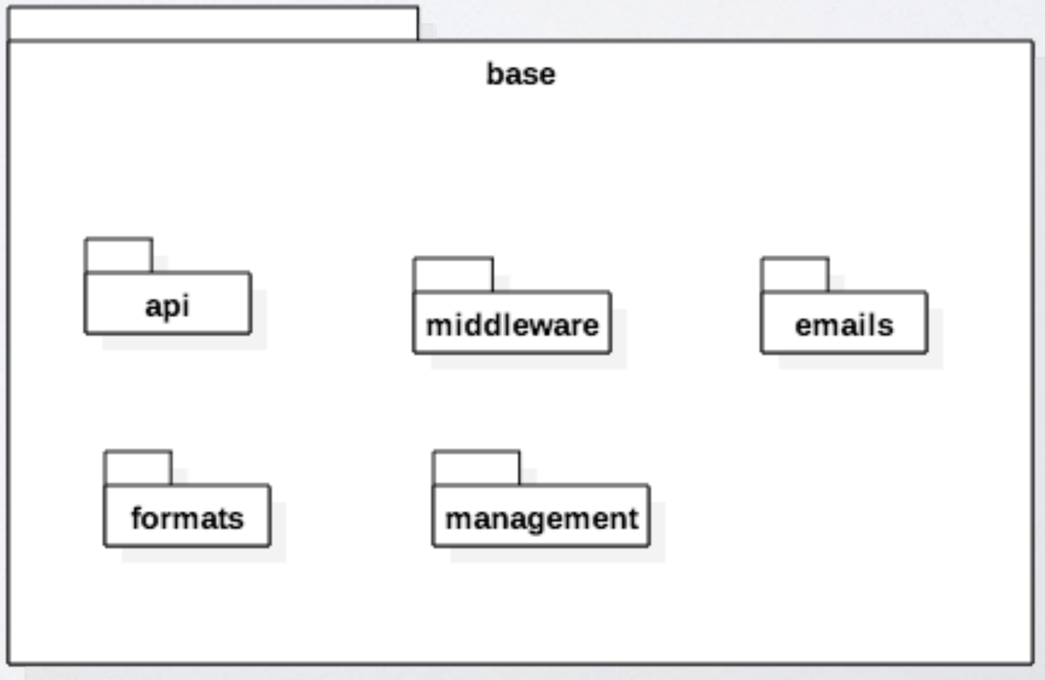
Useful for expressing the dependencies between major elements of a system.

EXAMPLE



Note:

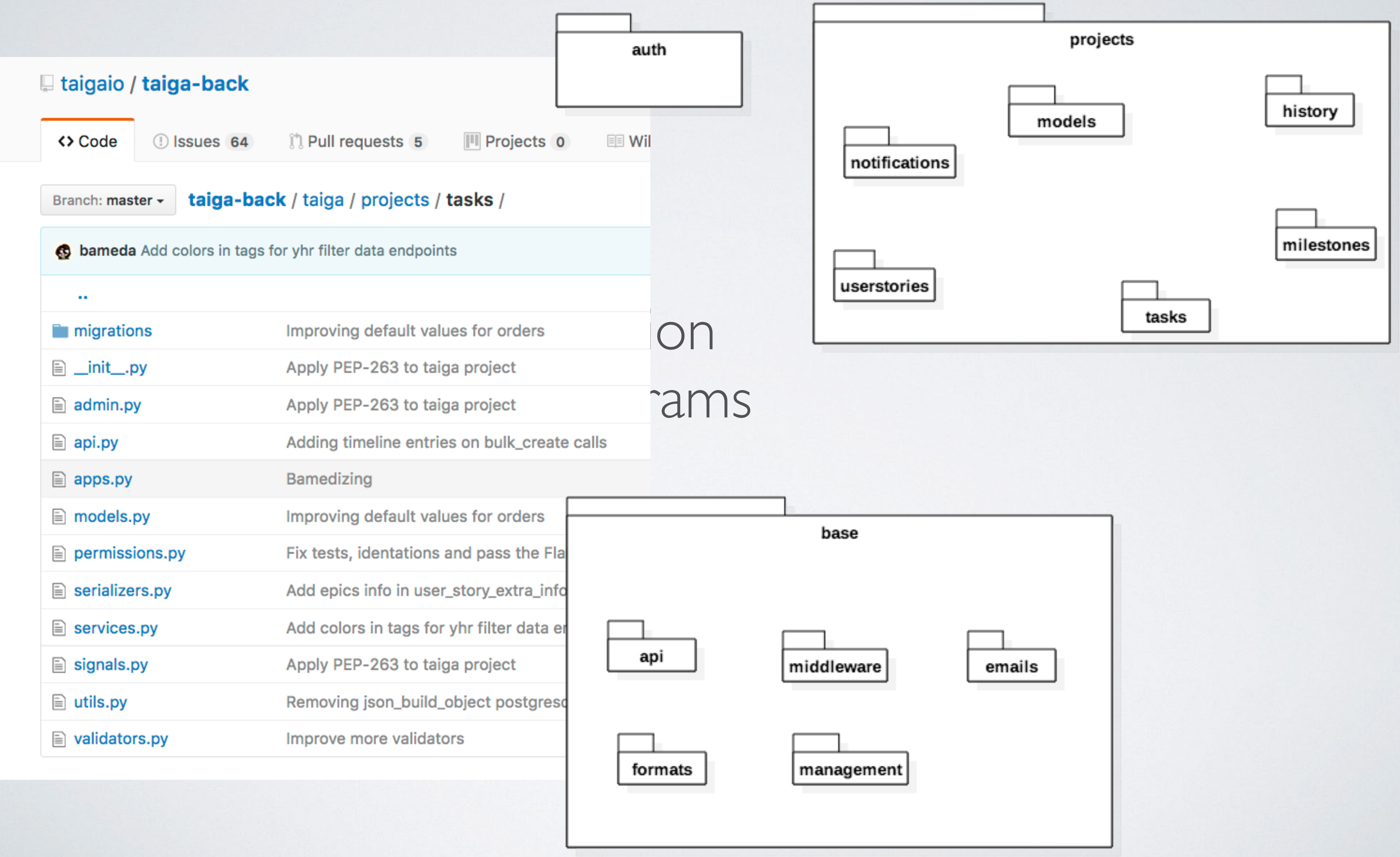
- This is by no means an exhaustive example
- This shows the connection between code and diagrams



EXAMPLE

on

rams



taigaio / taiga-back

<> Code | Issues 64 | Pull requests 5 | Projects 0 | Wi

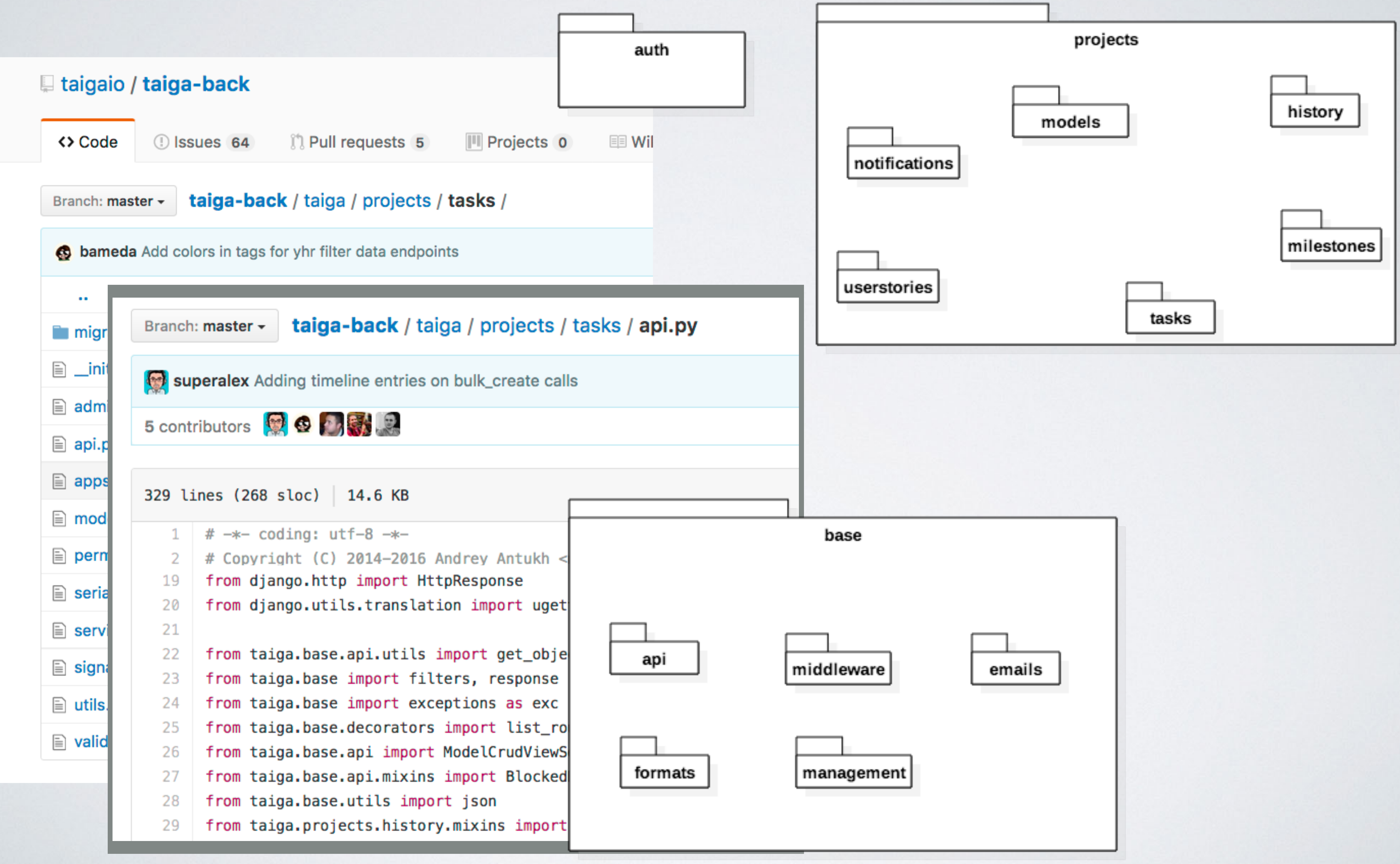
Branch: master ▾ taiga-back / taiga / projects / tasks /

bameda Add colors in tags for yhr filter data endpoints

..

migrations	Improving default values for orders
__init__.py	Apply PEP-263 to taiga project
admin.py	Apply PEP-263 to taiga project
api.py	Adding timeline entries on bulk_create calls
apps.py	Bamedizing
models.py	Improving default values for orders
permissions.py	Fix tests, identations and pass the Fla
serializers.py	Add epics info in user_story_extra_info
services.py	Add colors in tags for yhr filter data e
signals.py	Apply PEP-263 to taiga project
utils.py	Removing json_build_object postgres
validators.py	Improve more validators

EXAMPLE



taigaio / taiga-back

<> Code | Issues 64 | Pull requests 5 | Projects 0 | Wi

Branch: master ▾ taiga-back / taiga / projects / tasks /

bameda Add colors in tags for yhr filter data endpoints

..

migr

__init

admi

api.p

apps

mod

perm

seria

servi

signa

utils.

valid

Branch: master ▾ taiga-back / taiga / projects / tasks / api.py

superalex Adding timeline entries on bulk_create calls

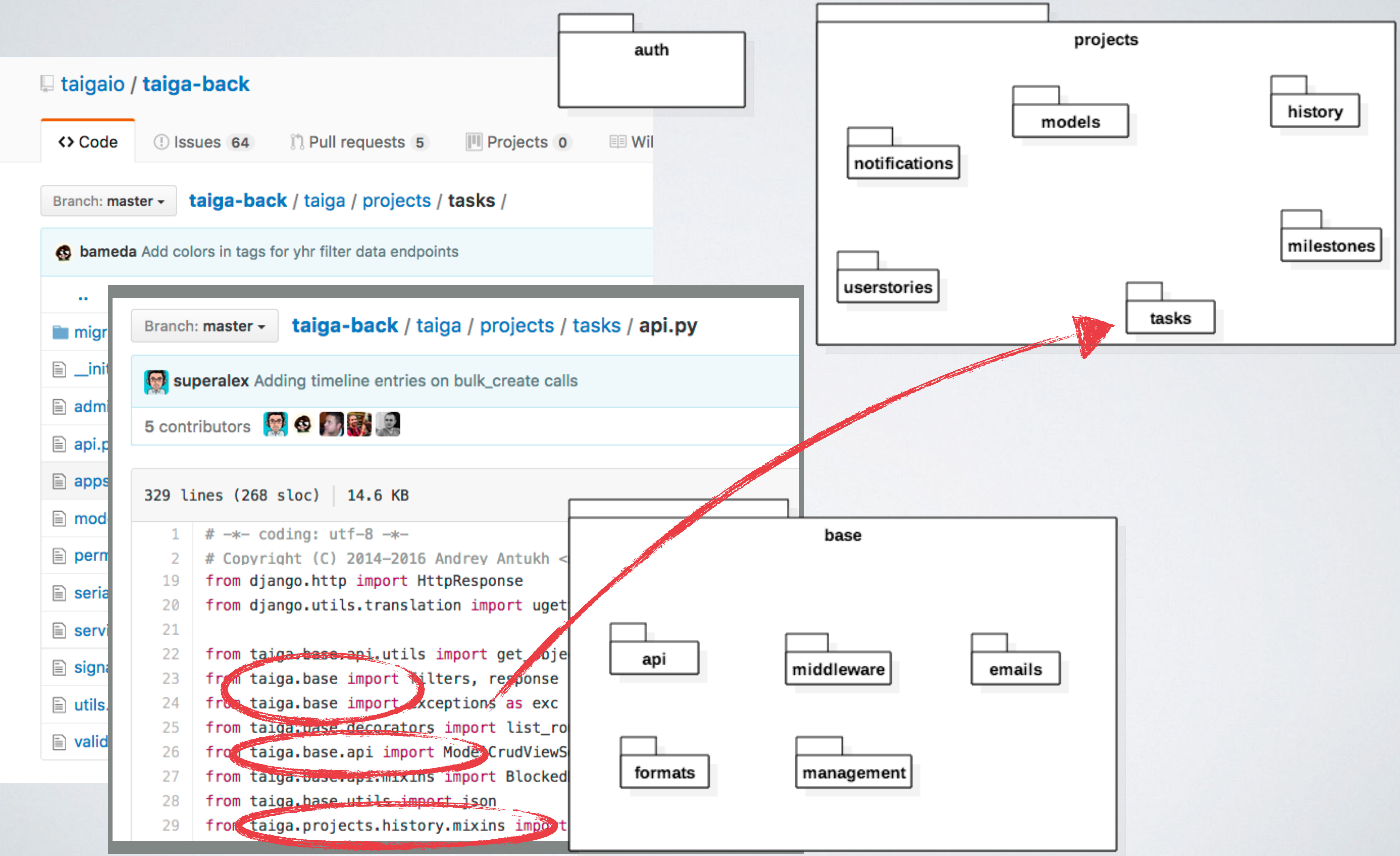
5 contributors

329 lines (268 sloc) | 14.6 KB

```
# -*- coding: utf-8 -*-
# Copyright (C) 2014-2016 Andrey Antukh <
from django.http import HttpResponse
from django.utils.translation import uget

from taiga.base.api.utils import get_obje
from taiga.base import filters, response
from taiga.base import exceptions as exc
from taiga.base.decorators import list_ro
from taiga.base.api import ModelCrudViewS
from taiga.base.api.mixins import Blocked
from taiga.base.utils import json
from taiga.projects.history.mixins import
```

EXAMPLE



taigaio / taiga-back

<> Code | Issues 64 | Pull requests 5 | Projects 0 | Wi

Branch: master ▾ taiga-back / taiga / projects / tasks /

bameda Add colors in tags for yhr filter data endpoints

..
migr
__init
admi
api.p
apps
mod
perm
seria
servi
signa
utils.
valid

Branch: master ▾ taiga-back / taiga / projects / tasks / api.py

superalex Adding timeline entries on bulk_create calls

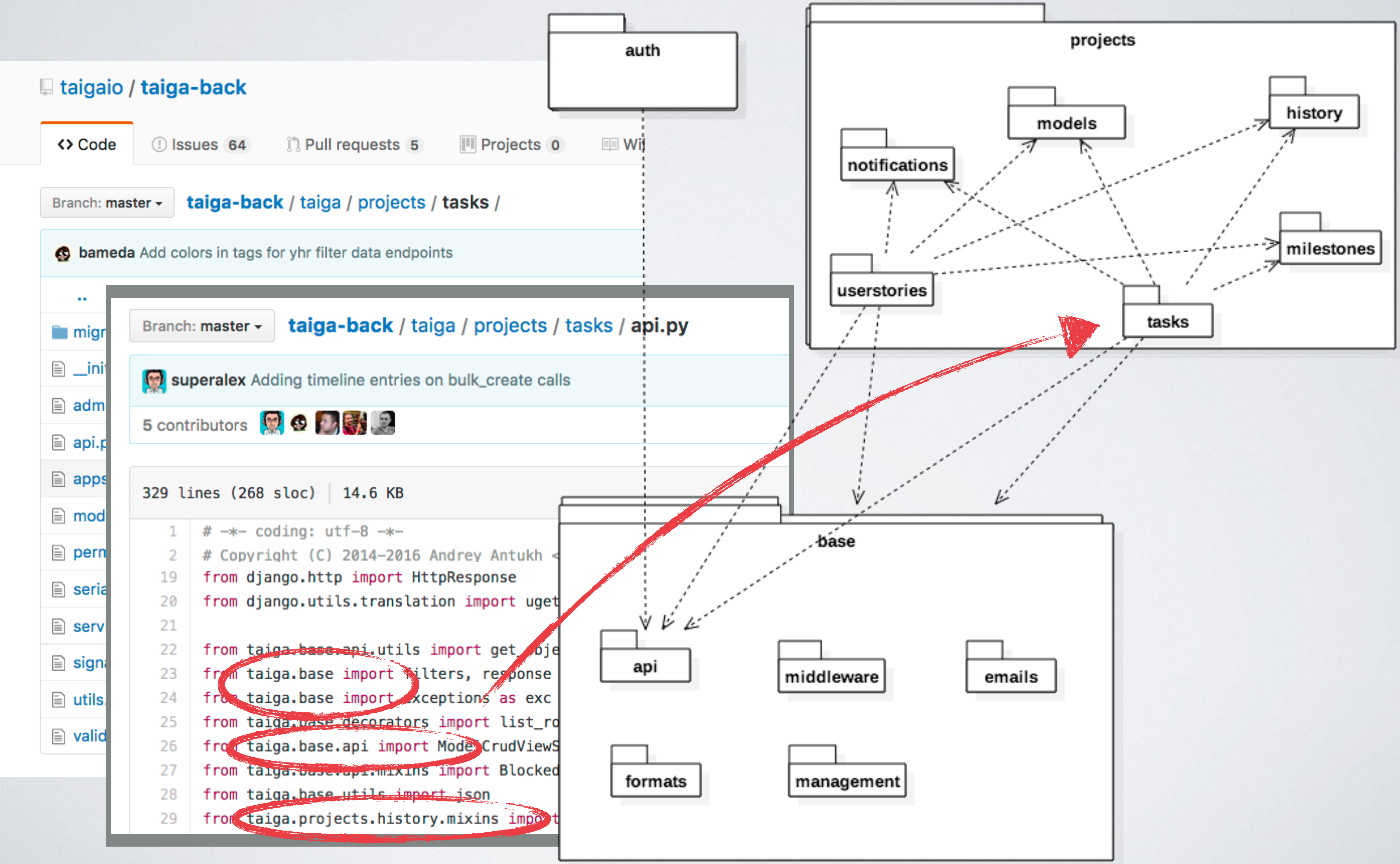
5 contributors

329 lines (268 sloc) | 14.6 KB

```
# -*- coding: utf-8 -*-
# Copyright (C) 2014-2016 Andrey Antukh <
from django.http import HttpResponse
from django.utils.translation import uget

from taiga.base.api.utils import get_obje
from taiga.base import filters, response
from taiga.base import exceptions as exc
from taiga.base.decorators import list_ro
from taiga.base.api import ModelCrudViewS
from taiga.base.api.mixins import Blocked
from taiga.base.utils import json
from taiga.projects.history.mixins import
```

EXAMPLE



taigaio / taiga-back

<> Code | Issues 64 | Pull requests 5 | Projects 0 | Wi

Branch: master ▾ taiga-back / taiga / projects / tasks /

bameda Add colors in tags for yhr filter data endpoints

- ..
- migr
- __init
- admi
- api.p
- apps
- mod
- perm
- seria
- servi
- signa
- utils.
- valid

Branch: master ▾ taiga-back / taiga / projects / tasks / api.py

superalex Adding timeline entries on bulk_create calls

5 contributors

329 lines (268 sloc) | 14.6 KB

```
# -*- coding: utf-8 -*-
# Copyright (C) 2014-2016 Andrey Antukh <
from django.http import HttpResponse
from django.utils.translation import uget

from taiga.base.api.utils import get_obje
from taiga.base import filters, response
from taiga.base import exceptions as exc
from taiga.base.decorators import list_ro
from taiga.base.api import ModelCrudViewS
from taiga.base.api.mixins import Blocked
from taiga.base.utils import json
from taiga.projects.history.mixins import
```

DEPLOYMENT DIAGRAMS

Express the physical deployment of software artefacts to hardware nodes – static view of run-time configuration

Use when application spans several machines.

Nodes correspond to

- devices (e.g., servers, mobile devices)
- specific execution environments (application servers, rule engines, operating system, virtual machines, database engines, web browser).

Nodes connected by communication paths (middleware, protocol)

DEPLOYMENT DIAGRAMS

Express the physical deployment of software artefacts to hardware nodes – static view of run-time configuration

Use when application spans several machines.

Nodes correspond to

- devices (e.g., servers, mobile devices)
- specific execution environments (application servers, rule engines, operating system, virtual machines, database engines, web browser).

Nodes connected by communication paths (middleware, protocol)

DEPLOYMENT DIAGRAMS

Express the physical deployment of software artefacts to hardware nodes – static view of run-time configuration

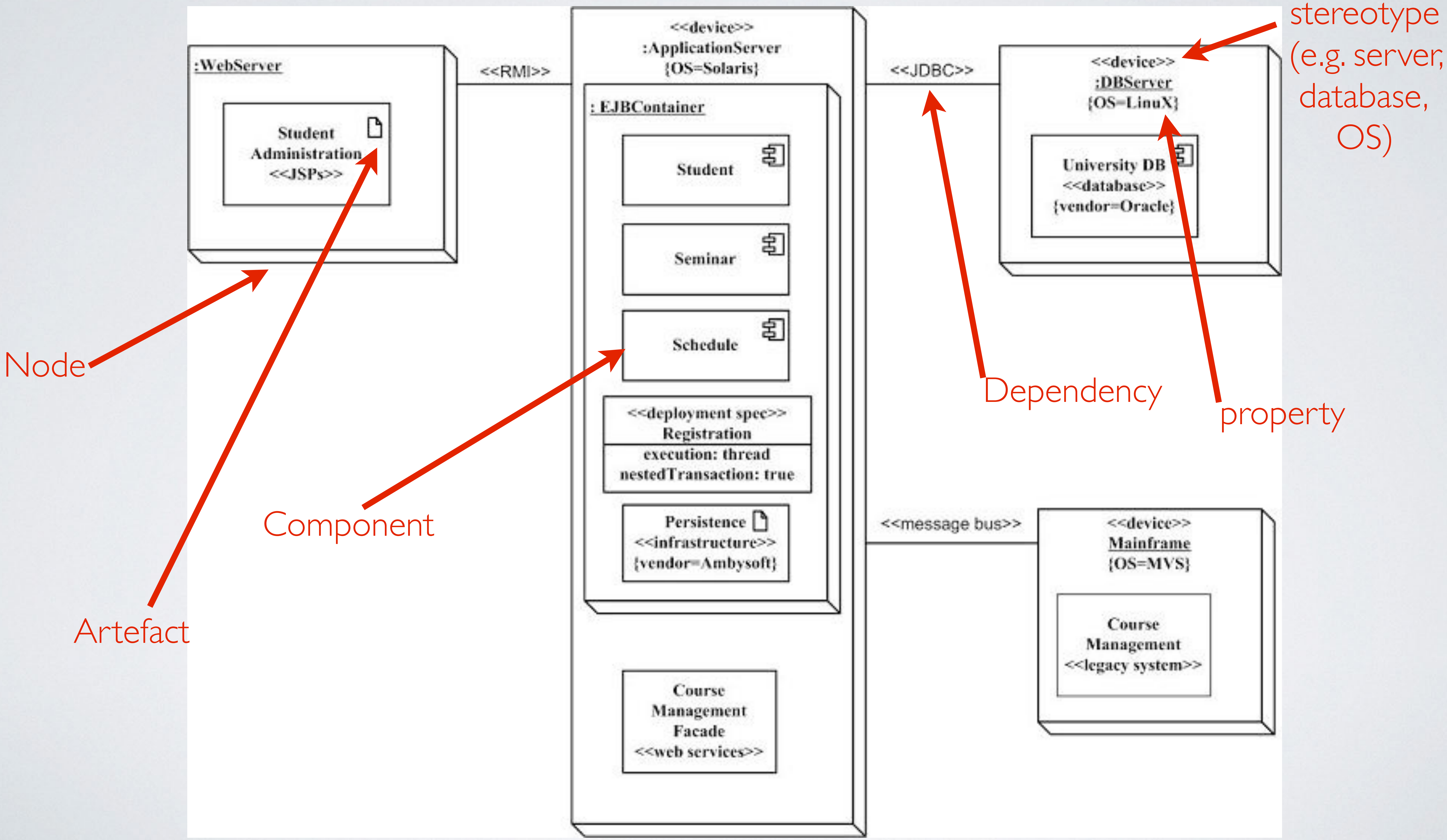
Use when application spans several machines.

Nodes correspond to

- devices (e.g., servers, mobile devices)
- specific execution environments (application servers, rule engines, operating system, virtual machines, database engines, web browser).

Nodes connected by communication paths (middleware, protocol)

DEPLOYMENT DIAGRAMS



stereotype (e.g. server, database, OS)

Node

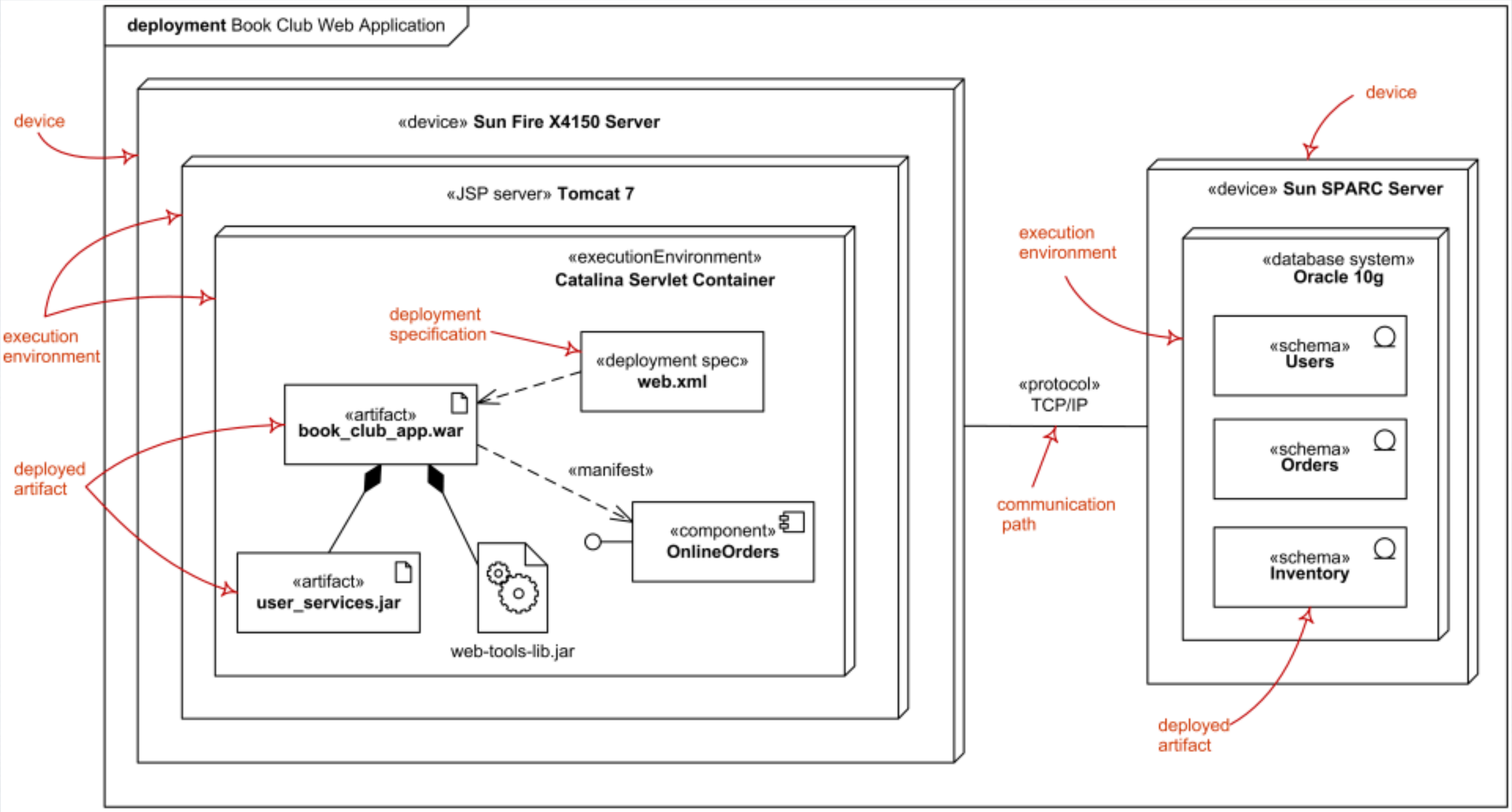
Component

Artefact

Dependency

property

DEPLOYMENT DIAGRAMS



source: https://www.uml-diagrams.org/deployment-diagrams-overview.html

A LINE-AND-BOX DIAGRAM IS NOT AN ARCHITECTURE

Its purpose is to be able to **assess** the level at which the system fulfils **the non-functional requirements**

- How well does it scale?
- How suitable is it for a real-time system?

A LINE-AND-BOX DIAGRAM IS NOT AN ARCHITECTURE

It should follow the standards

- Easier to talk about with other people
- Easier to compare designs
- Easier to understand **why a design was chosen**

A LINE-AND-BOX DIAGRAM IS NOT AN ARCHITECTURE

It should help in dictating **organisational structure** (as the basis for a work-breakdown-structure)

- Forming teams for separate development
- Units of planning, scheduling and budget

ARCHITECTURAL STYLES (= SA DESIGN PATTERNS)

ARCHITECTURAL STYLES

Object-oriented

Client server; object broker; peer to peer

Pipe and filter

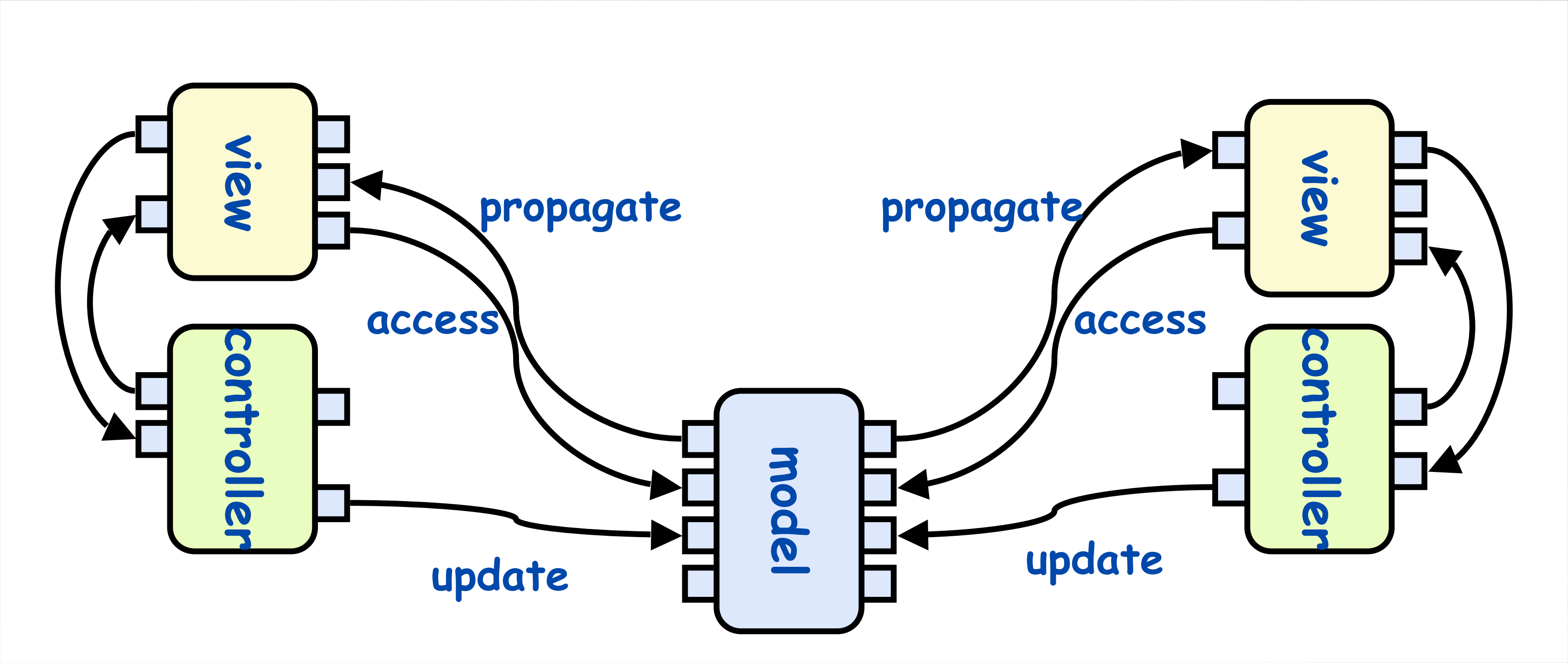
Layered – Three-tier, Four-tier

Repositories: blackboard, Model/View/Controller (MVC)

Microservices

The diagrams that follow capture only high level of abstraction — still need to place components.

MODEL-VIEW-CONTROLLER



Properties

- One central model, many views (viewers)
- Each view has an associated controller
- The controller handles updates from the user of the view
- Changes to the model are propagated to all the views

MODEL-VIEW-CONTROLLER (MVC)

Model contains domain knowledge

Views only display data

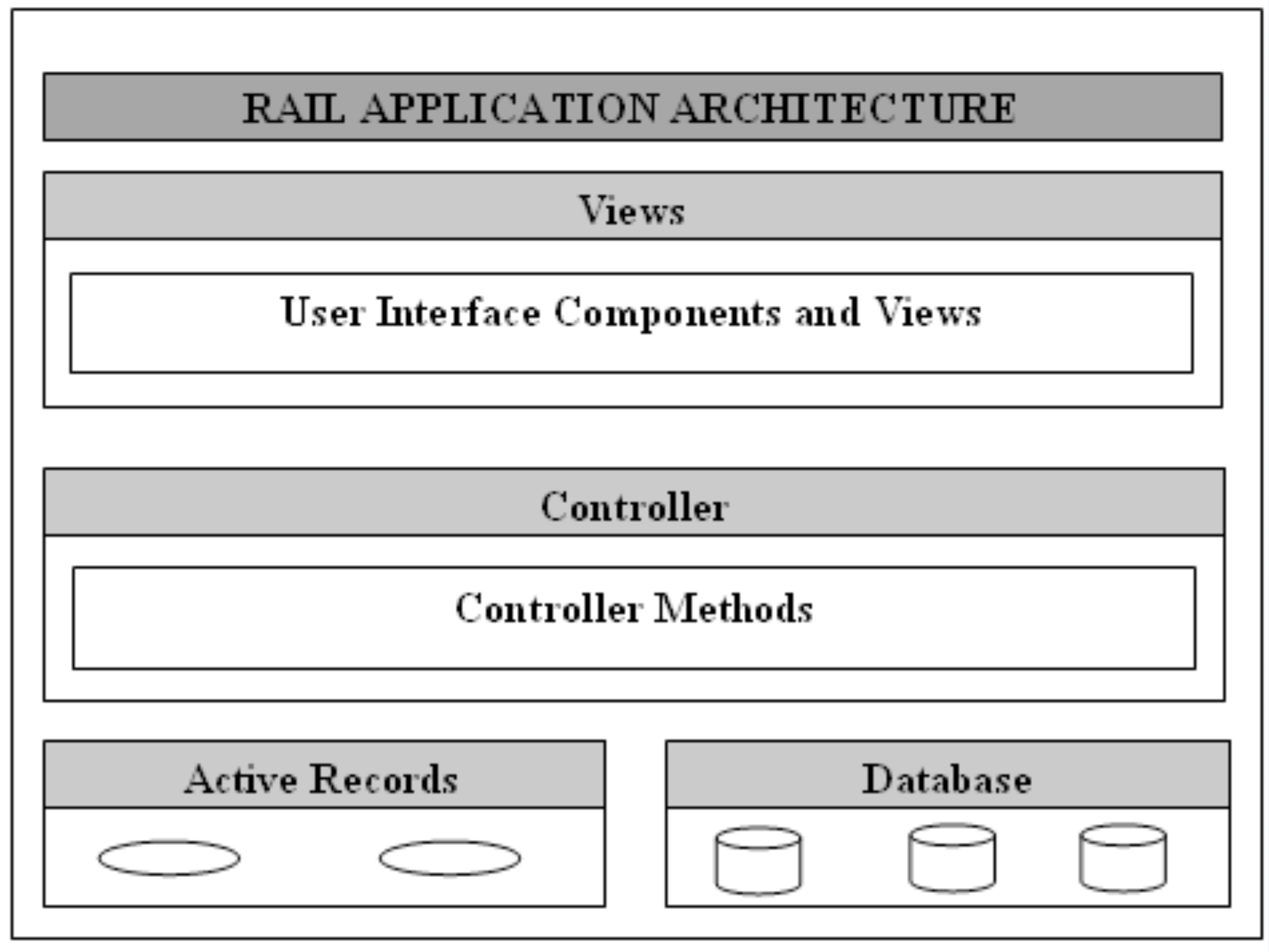
Controllers only manage interaction sequences

Model does not depend on views or controllers

Subscribe / notify mechanism

EXAMPLE MODEL-VIEW-CONTROLLER

Ruby on Rails architecture (Web Framework)



RAIL APPLICATION ARCHITECTURE

Views

User Interface Components and Views

Controller

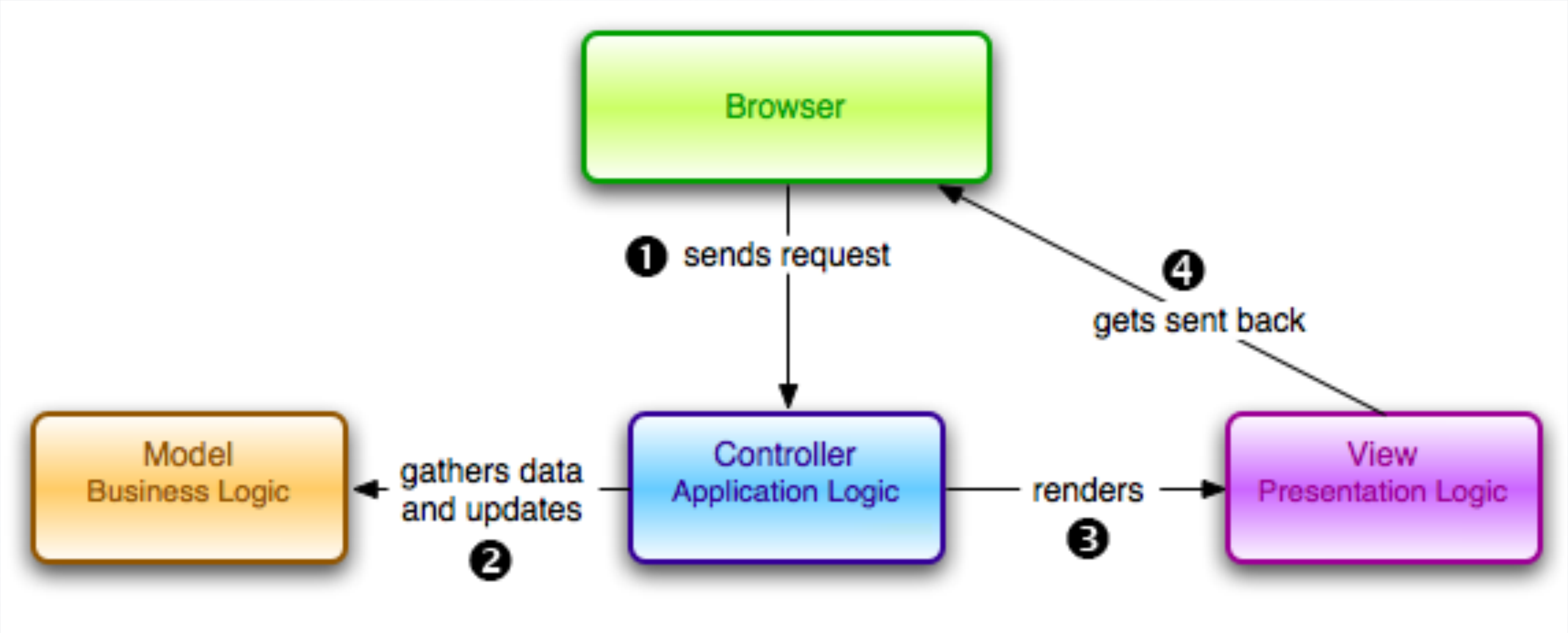
Controller Methods

Active Records

Database

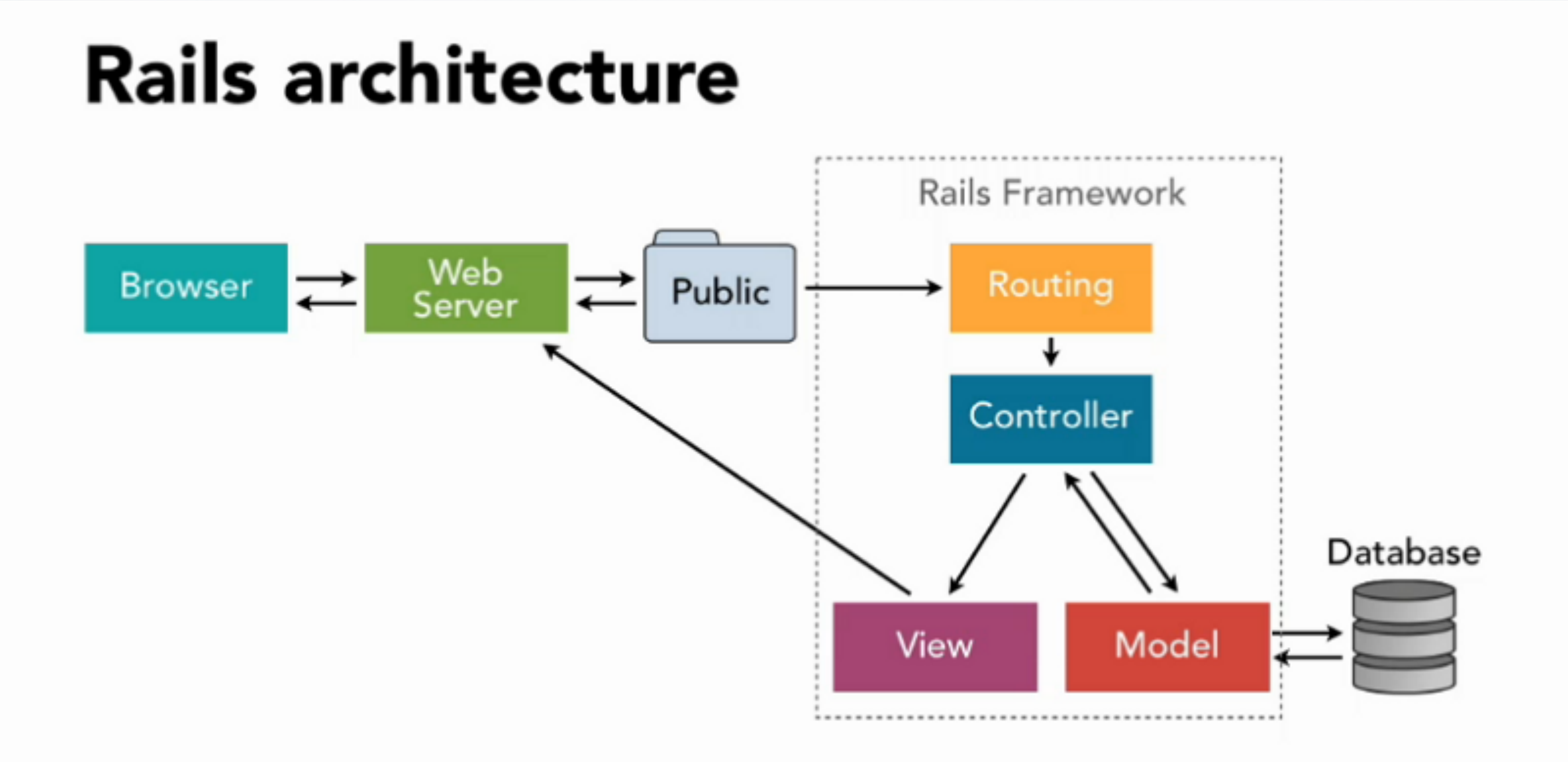
EXAMPLE MODEL-VIEW-CONTROLLER

Ruby on Rails architecture (Web Framework)



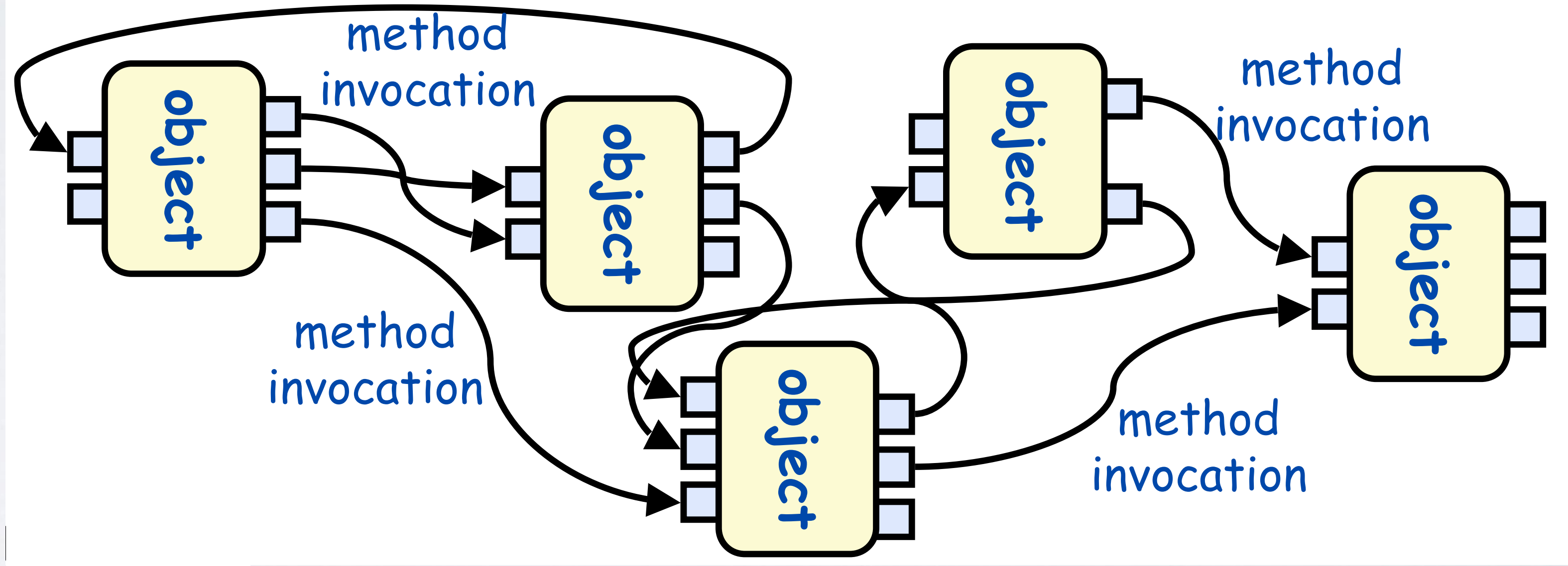
EXAMPLE MODEL-VIEW-CONTROLLER

Ruby on Rails architecture (Web Framework)



https://www.freecodecamp.org/news/understanding-the-basics-of-ruby-on-rails-http-mvc-and-routes-359b8d809c7a/

OBJECT-ORIENTED ARCHITECTURES



Example

- Abstract data types (modules)

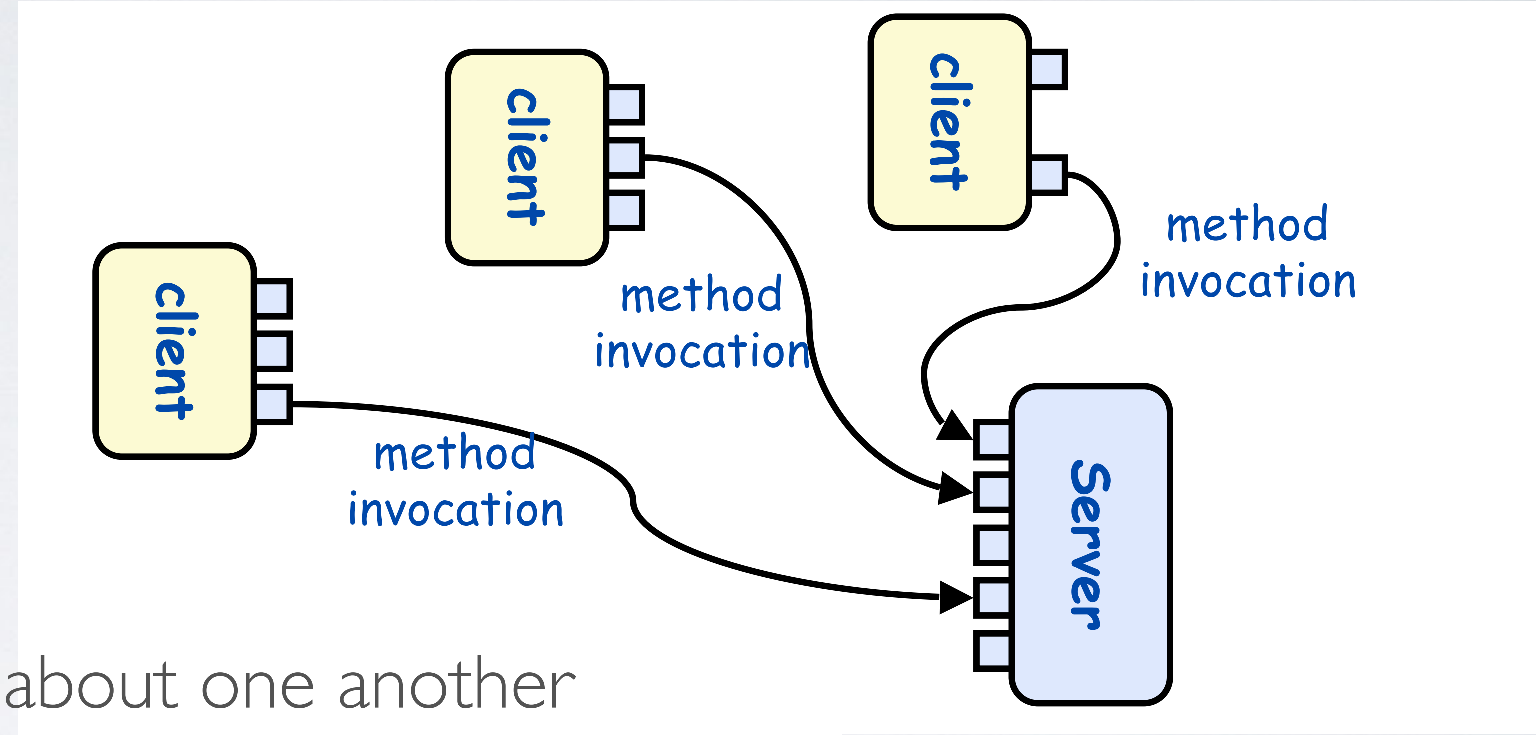
Interesting properties

- data hiding (internal data representations are not visible to clients)
- can decompose problems into sets of interacting agents
- can be multi-threaded or single thread

Disadvantages

- objects must know the identity of objects they wish to interact with

CLIENT-SERVER



Interesting properties

- Clients do not need to know about one another
- Breaks the system into manageable components
- Independent flow of control
- Server generally responsible for persistence and consistency of data

Disadvantages

- Client objects must know the identity of the server

CLIENT-SERVER

Client/Server communication via remote procedure call or common object broker (e.g. CORBA, Java RMI, or HTTP)

Variants

– **thick** clients have their own services

– **thin** ones get everything from servers

CLIENT-SERVER

Traditional model

CLIENT-SERVER

Traditional model

Each request is handled by a separate thread or process

CLIENT-SERVER

Traditional model

Each request is handled by a separate thread or process

If the process is waiting for the I/O, whole thread is blocked

CLIENT-SERVER

Traditional model

Each request is handled by a separate thread or process

If the process is waiting for the I/O, whole thread is blocked

If an unexpected problem occurs while processing a request, only that particular thread will crash leaving rest of the requests and threads intact

CLIENT-SERVER

Traditional model

Each request is handled by a separate thread or process

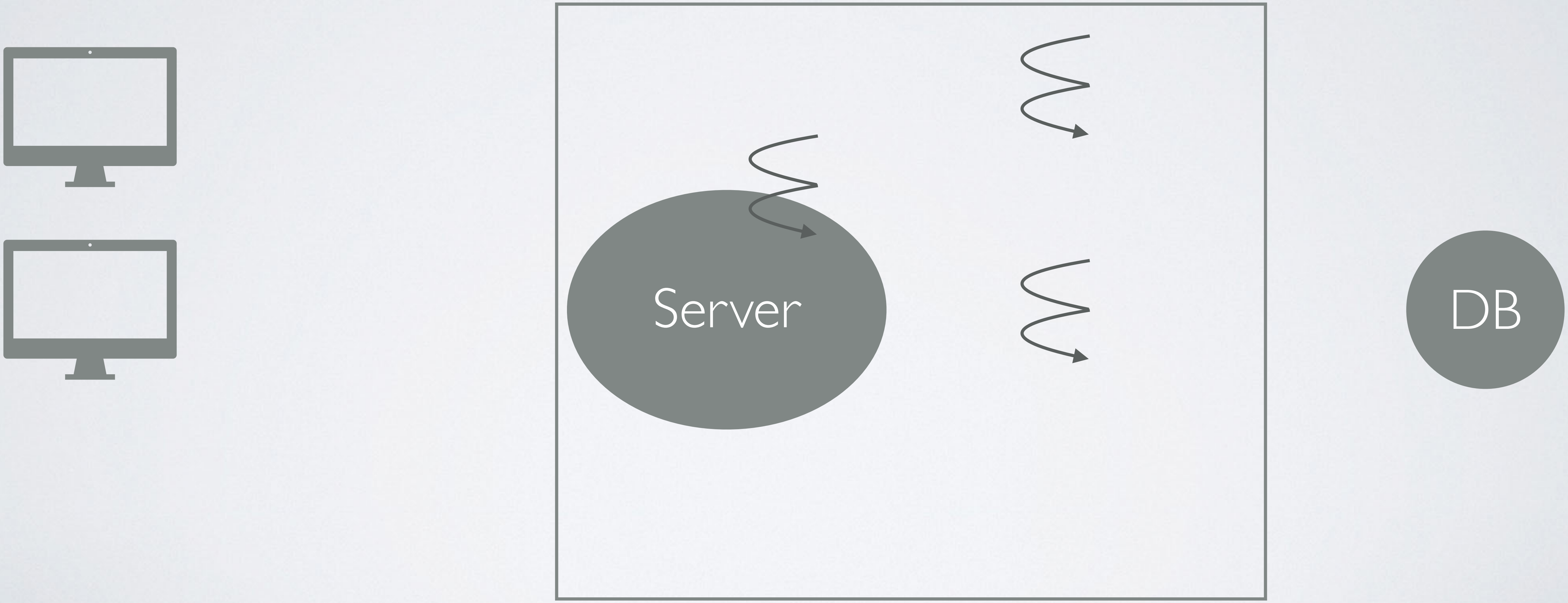
If the process is waiting for the I/O, whole thread is blocked

If an unexpected problem occurs while processing a request, only that particular thread will crash leaving rest of the requests and threads intact

Apache gets a request which is CPU intensive, other request do not get blocked because of the context switching between the threads

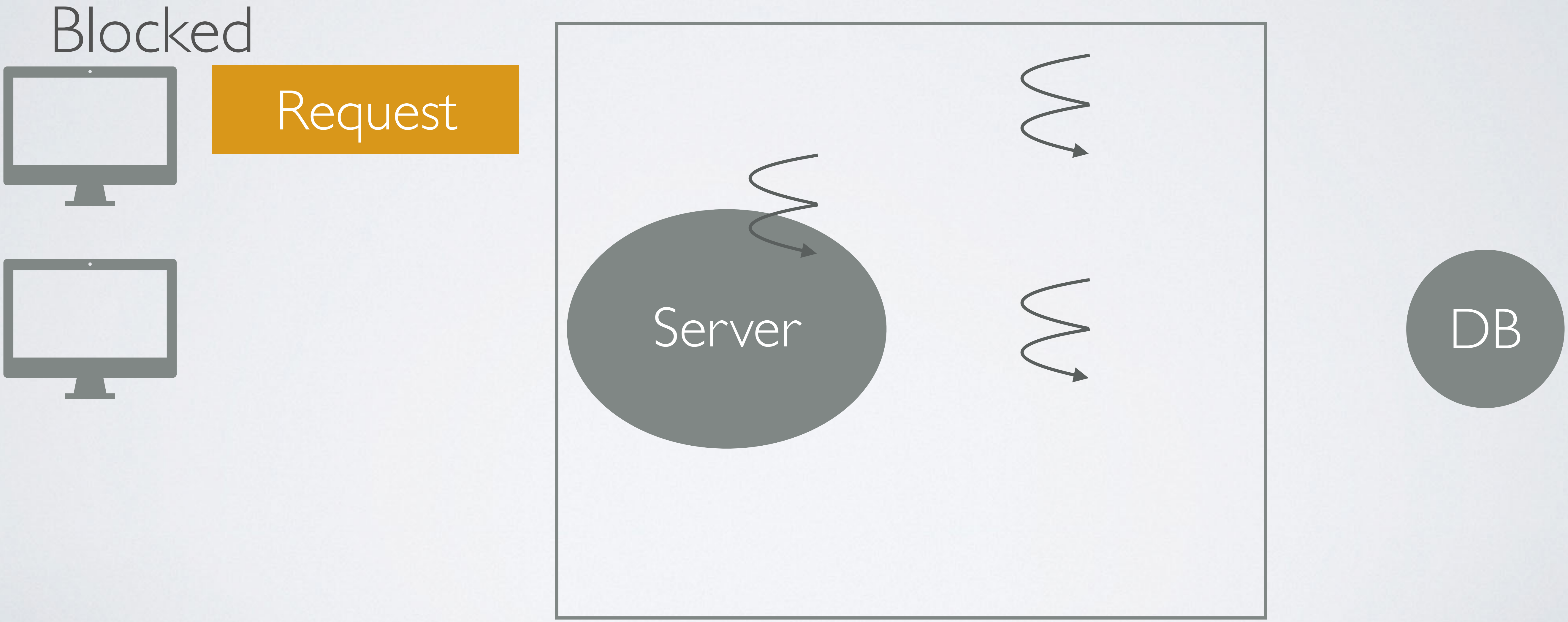
CLIENT-SERVER

Traditional model



CLIENT-SERVER

Traditional model



Blocked

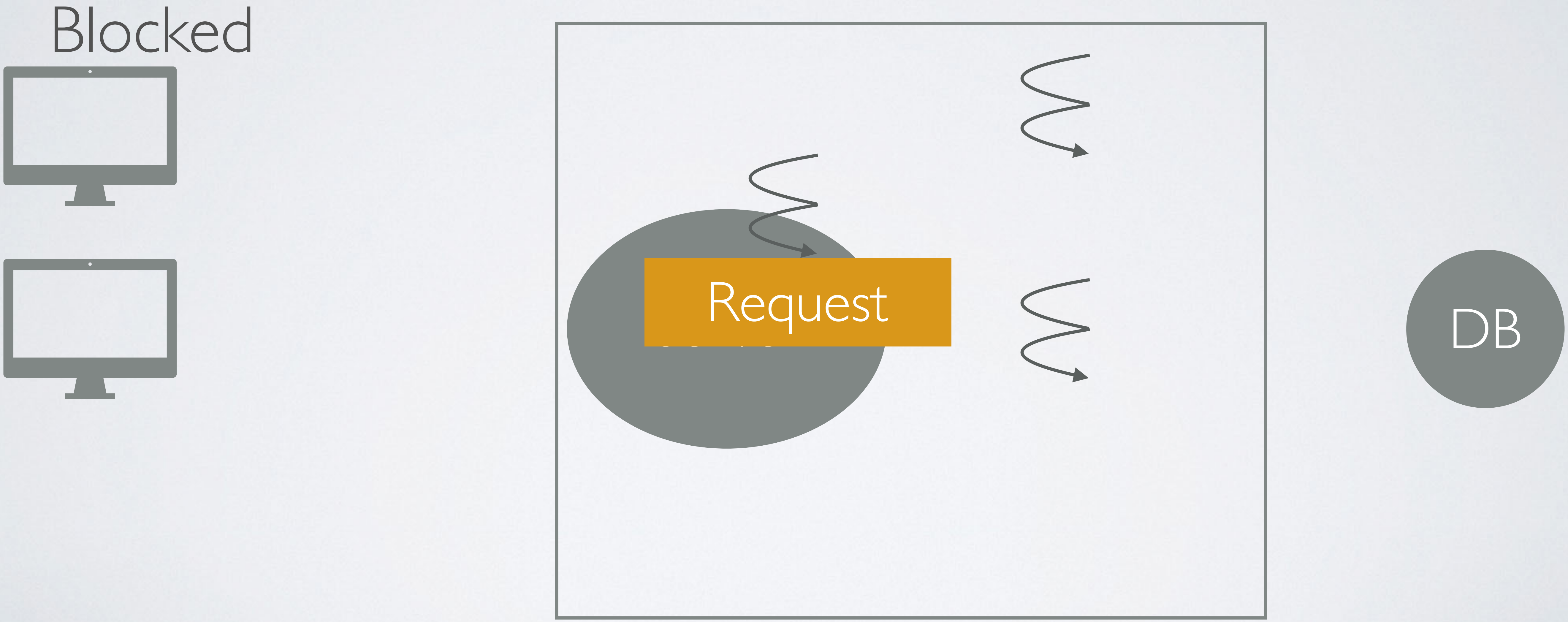
Request

Server

DB

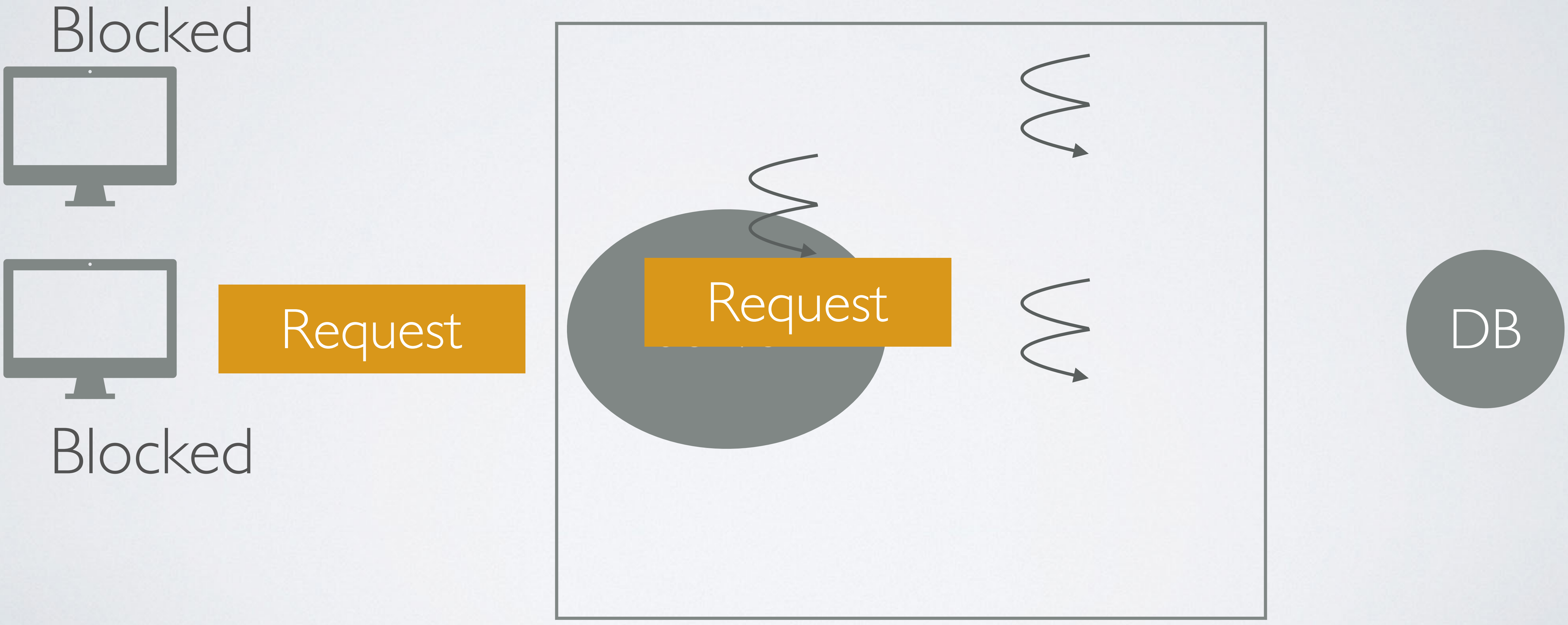
CLIENT-SERVER

Traditional model



CLIENT-SERVER

Traditional model



Blocked

Request

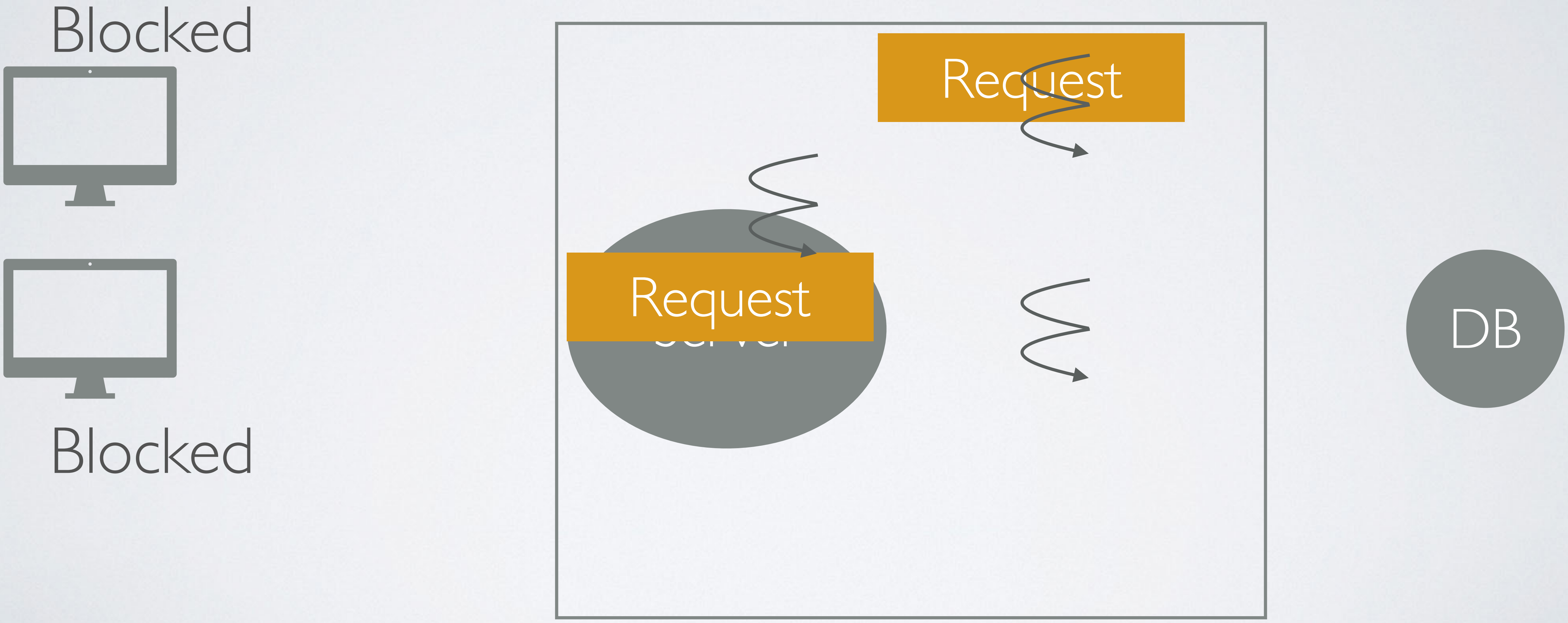
Request

DB

Blocked

CLIENT-SERVER

Traditional model



Blocked

Request

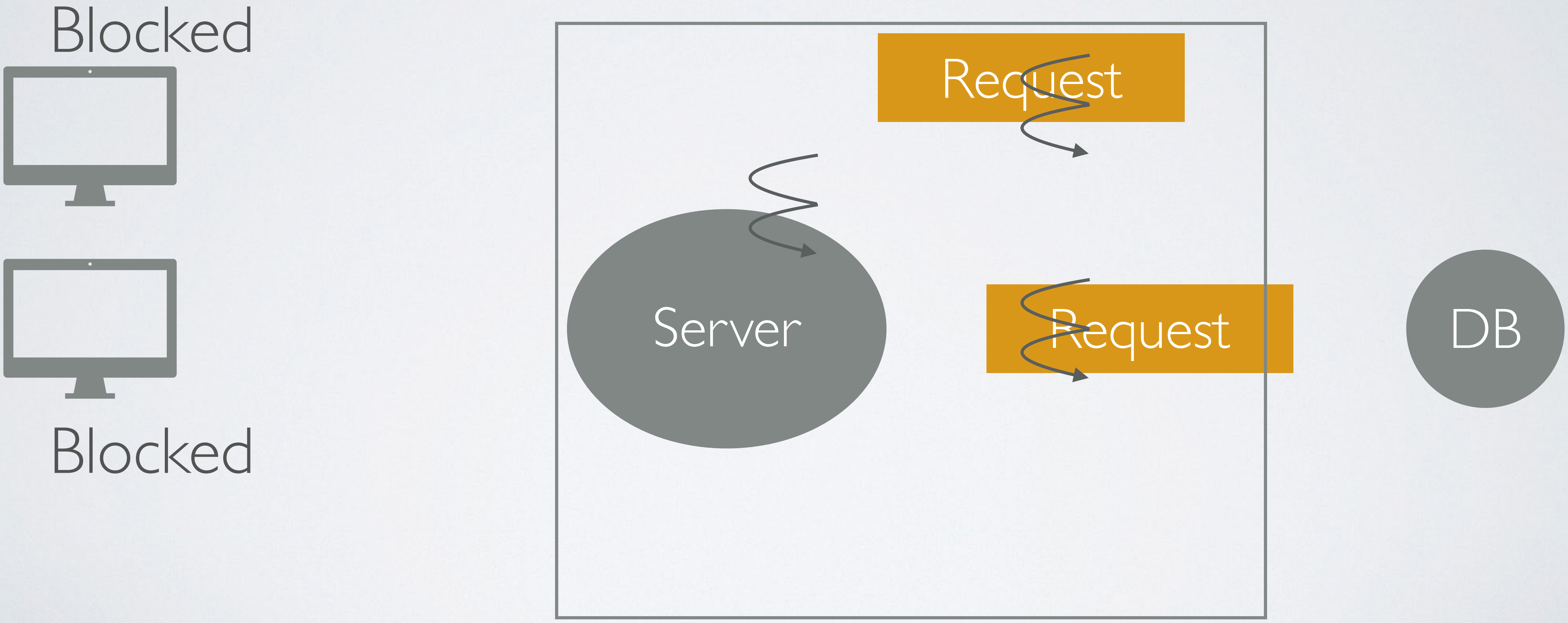
Request

DB

Blocked

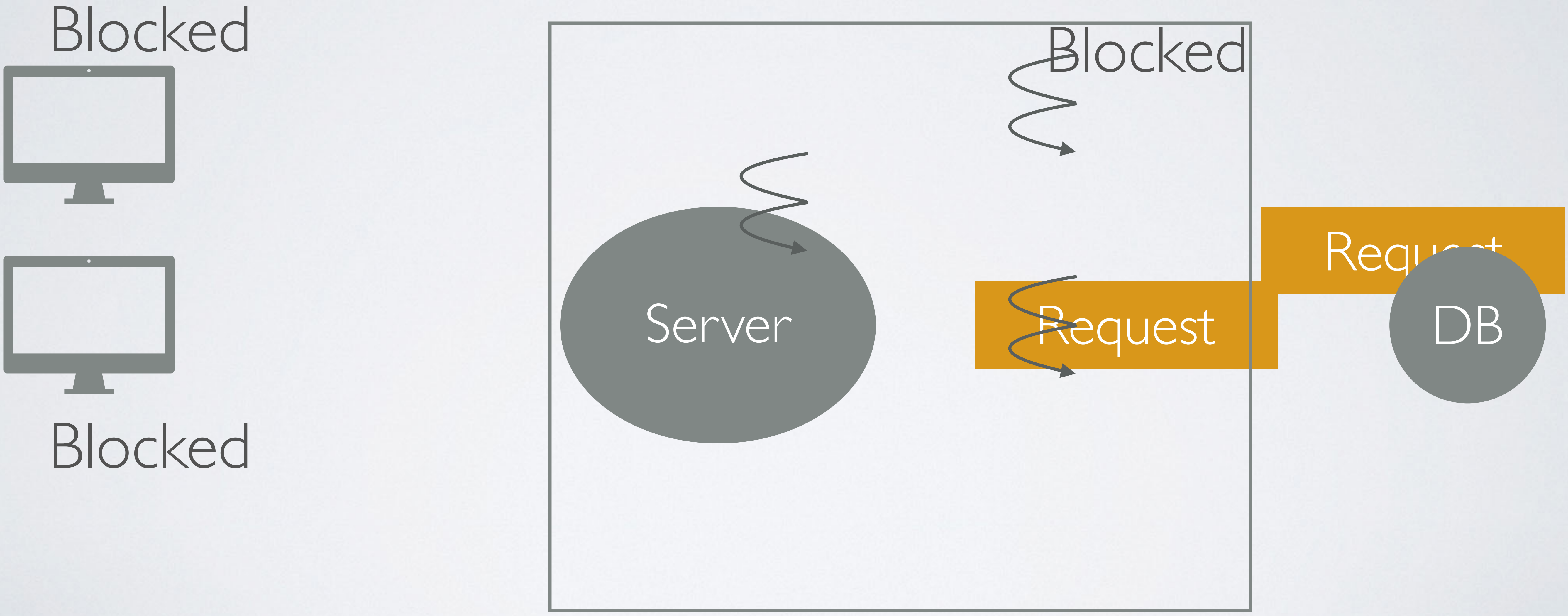
CLIENT-SERVER

Traditional model



CLIENT-SERVER

Traditional model



Blocked

Blocked

Blocked

Server

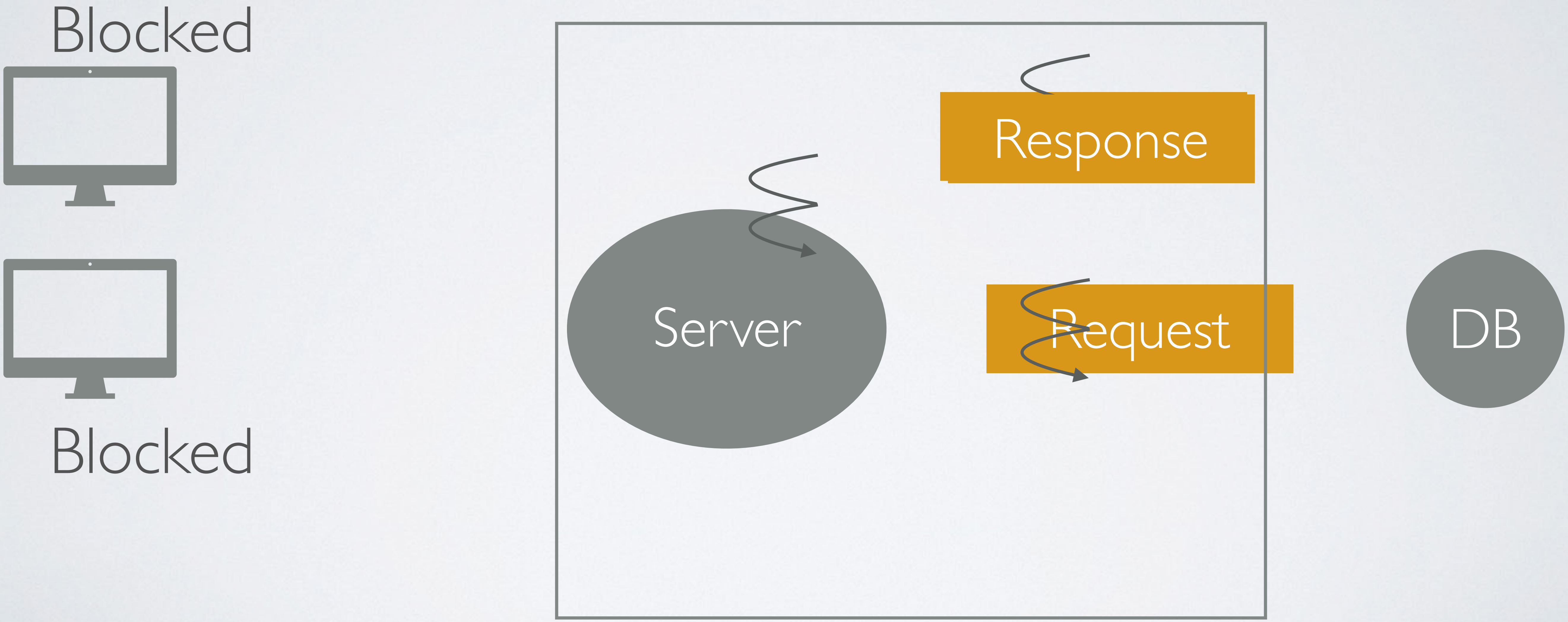
Request

Request

DB

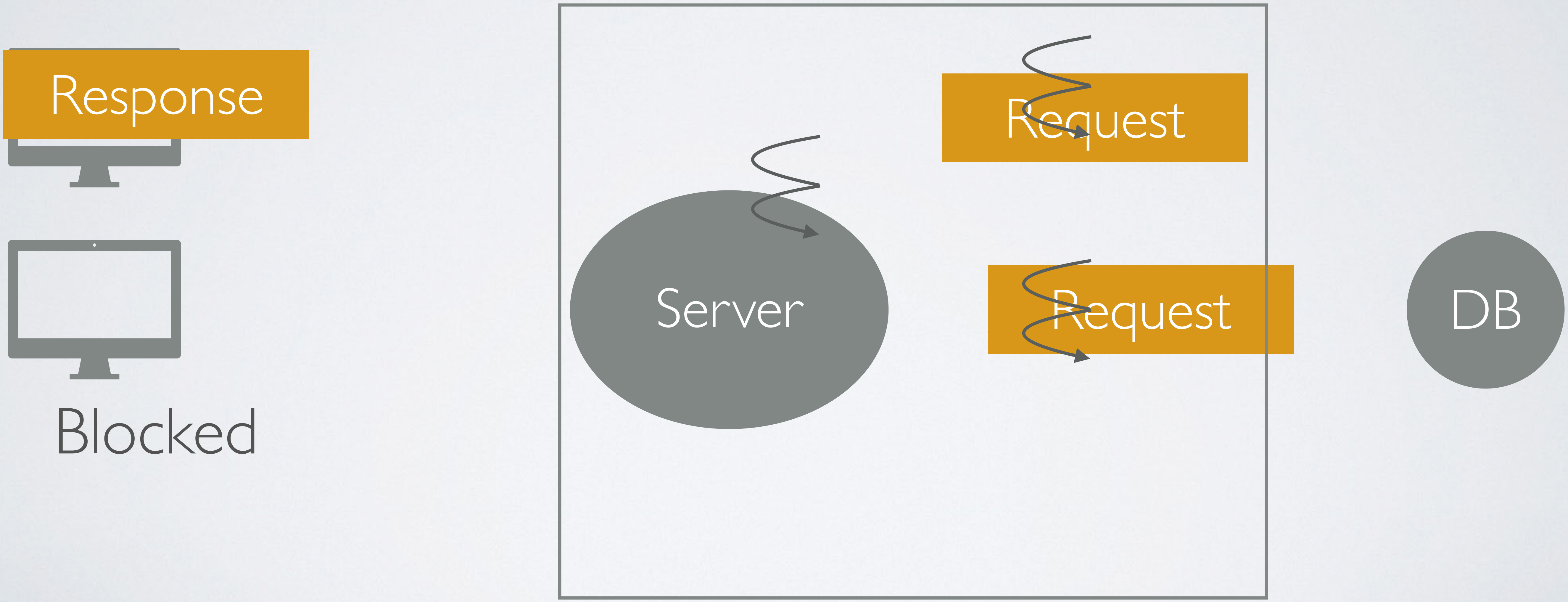
CLIENT-SERVER

Traditional model



CLIENT-SERVER

Traditional model



CLIENT-SERVER

Asynchronous, event driven I/O

CLIENT-SERVER

Asynchronous, event driven I/O

Every NodeJS instance runs in a *single thread* and due to its asynchronous nature, it can handle far more number of concurrent requests as compared to apache

CLIENT-SERVER

Asynchronous, event driven I/O

Every NodeJS instance runs in a *single thread* and due to its asynchronous nature, it can handle far more number of concurrent requests as compared to apache

If a problem occurs the whole NodeJS instance will crash along with any global data that was stored in javascript variables or arrays

CLIENT-SERVER

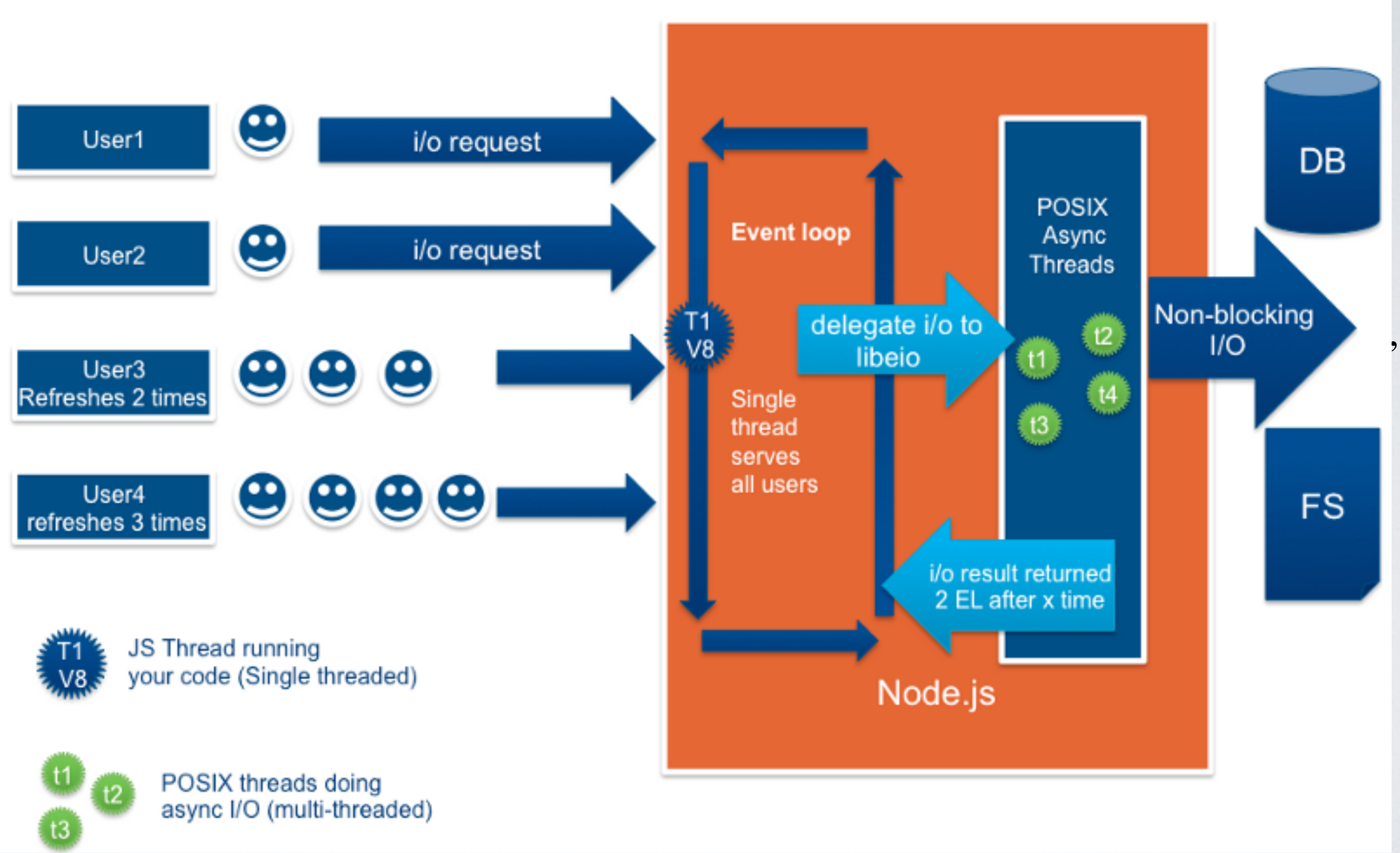
Asynchronous, event driven I/O

Every NodeJS instance runs in a *single thread* and due to its asynchronous nature, it can handle far more number of concurrent requests as compared to apache

If a problem occurs the whole NodeJS instance will crash along with any global data that was stored in javascript variables or arrays

NodeJs gets a CPU intensive request, all the other requests get blocked till this CPU intensive request stops for an I/O

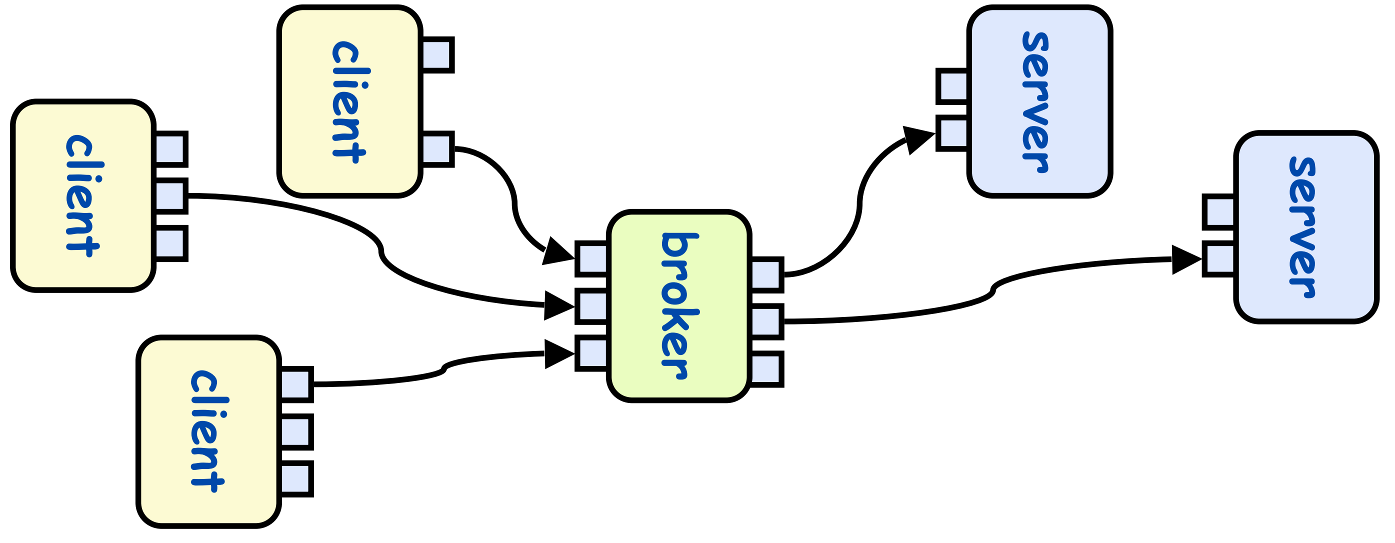
CLIENT-SERVER



,

https://codeburst.io/how-node-js-single-thread-mechanism-work-understanding-event-loop-in-nodejs-230f7440b0ea

OBJECT BROKER



Interesting Properties

- Adds a broker between the clients and servers
- Clients no longer need to know which server they are using
- Can have many brokers, many servers.

Disadvantages

- Broker can become a bottleneck
- Degraded performance

When would you mix the following styles?

When would you mix the following styles?

+ Client-Server

When would you mix the following styles?

+ Client-Server
+ Broker

When would you mix the following styles?

- + Client-Server
- + Broker
- + Object-oriented

When would you mix the following styles?

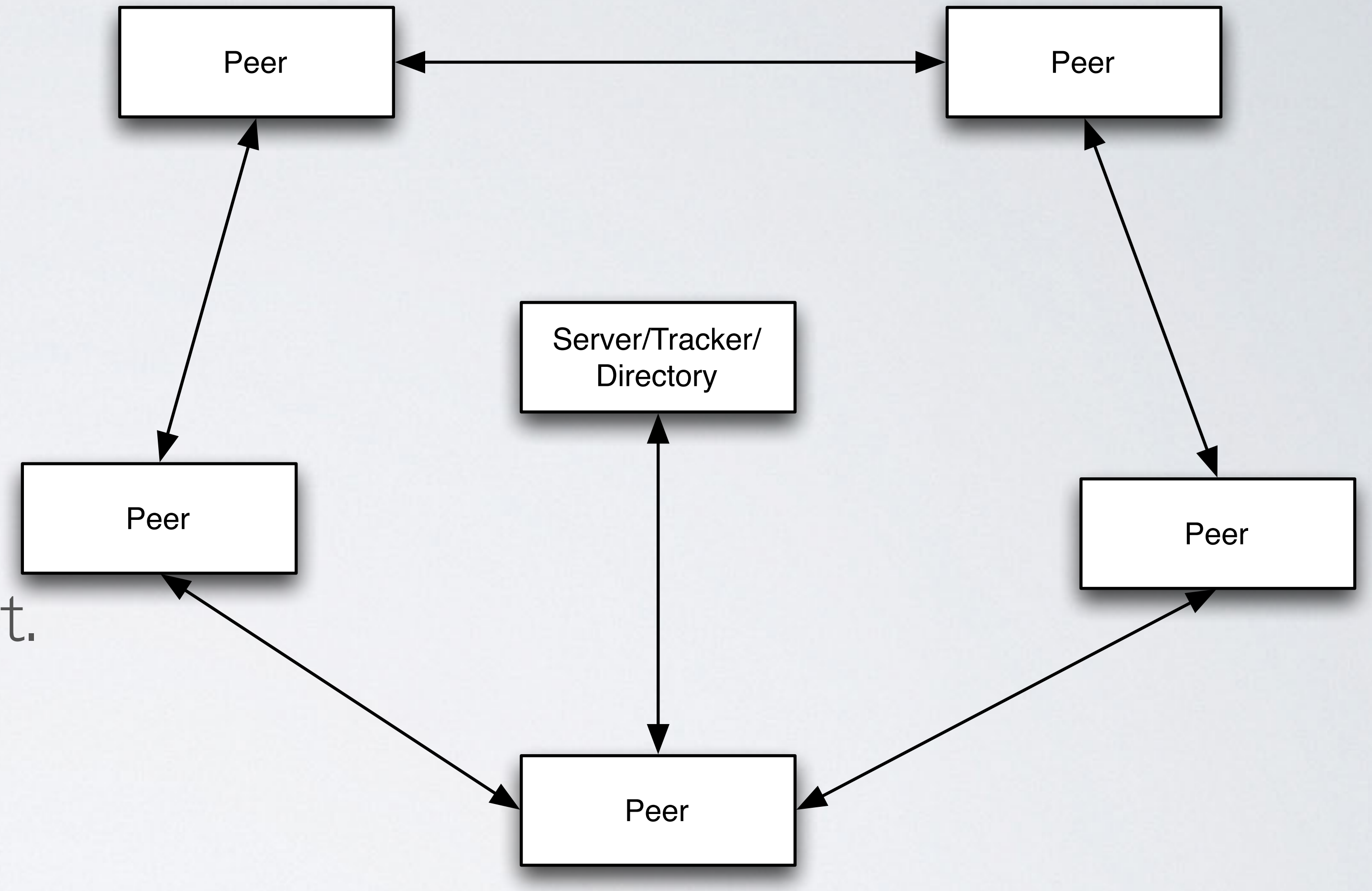
+ Client-Server
+ Broker
+ Object-oriented
+ MVC

When would you mix the following styles?

- + Client-Server
- + Broker
- + Object-oriented
- + MVC

This is standard nowadays in any web app!

PEER-TO-PEER



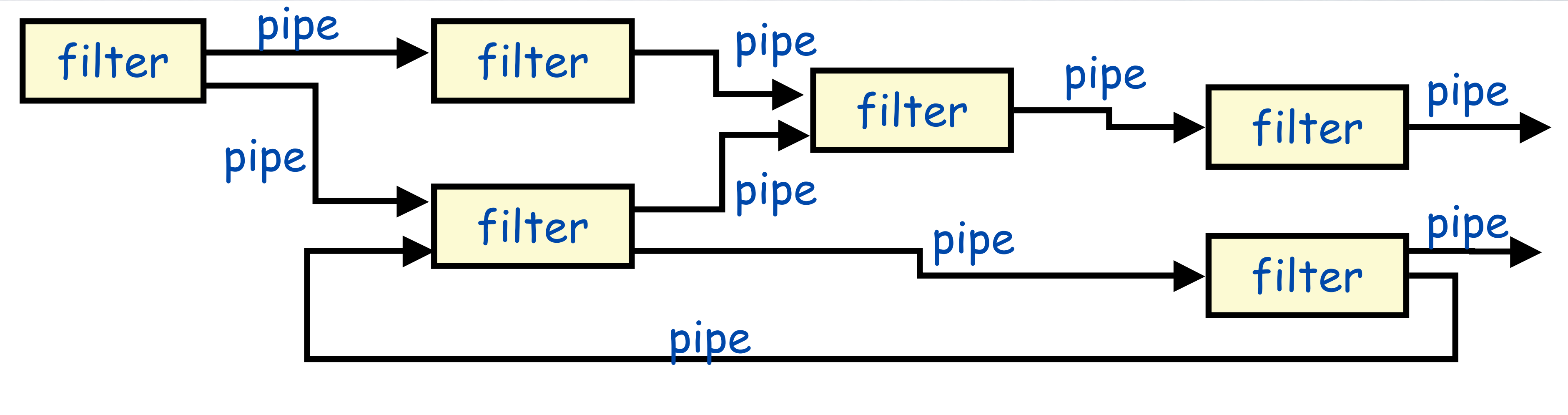
Interesting Properties

- Find peers via server or broadcast.
- Interact subsequently with peers.
- Reduces bottleneck. Robust to peer failure.

Disadvantages

- Server can become a bottleneck
- Peers have only incomplete picture – synchronisation is (virtually) impossible

PIPE AND FILTER



Examples

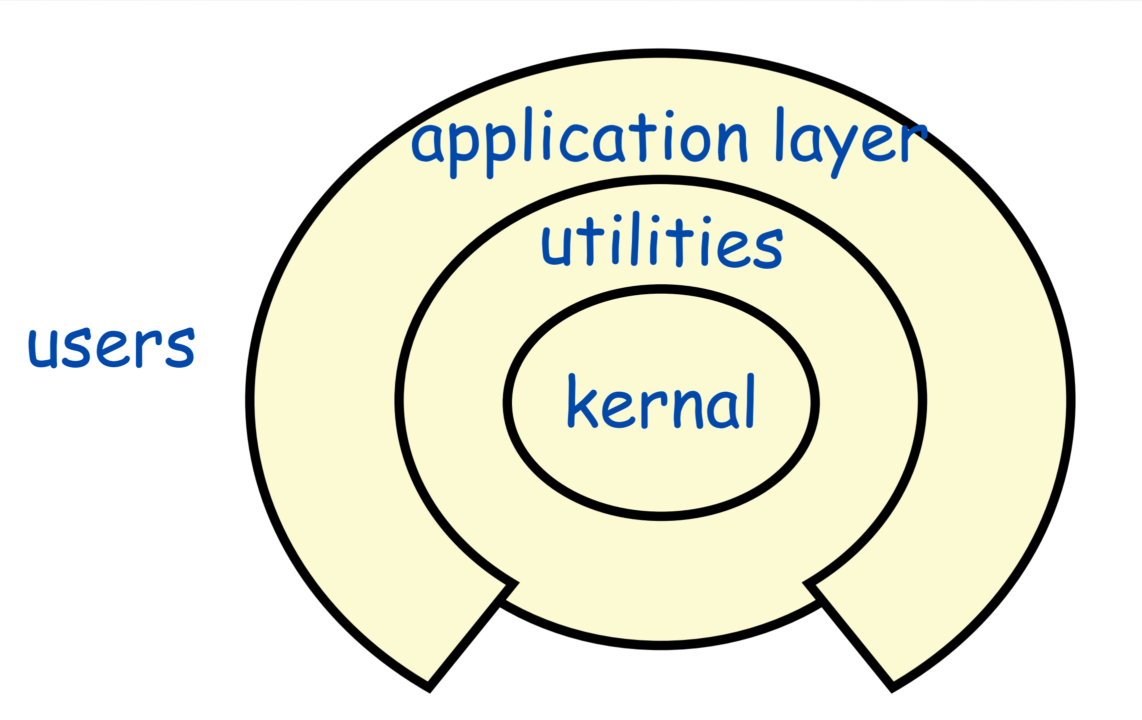
- Unix command shell
- compiler chain: lexical analysis → parsing → semantic analysis → code generation
- signal processing

```
grep gustav < foo.txt | sort | cut -f2-3
```

Interesting properties:

- filters don't need to know anything about what they are connected to
- filters can be implemented in parallel
- behaviour of the system is the composition of behaviour of the filters

LAYERED SYSTEMS



Examples

- Operating Systems
- communication protocols

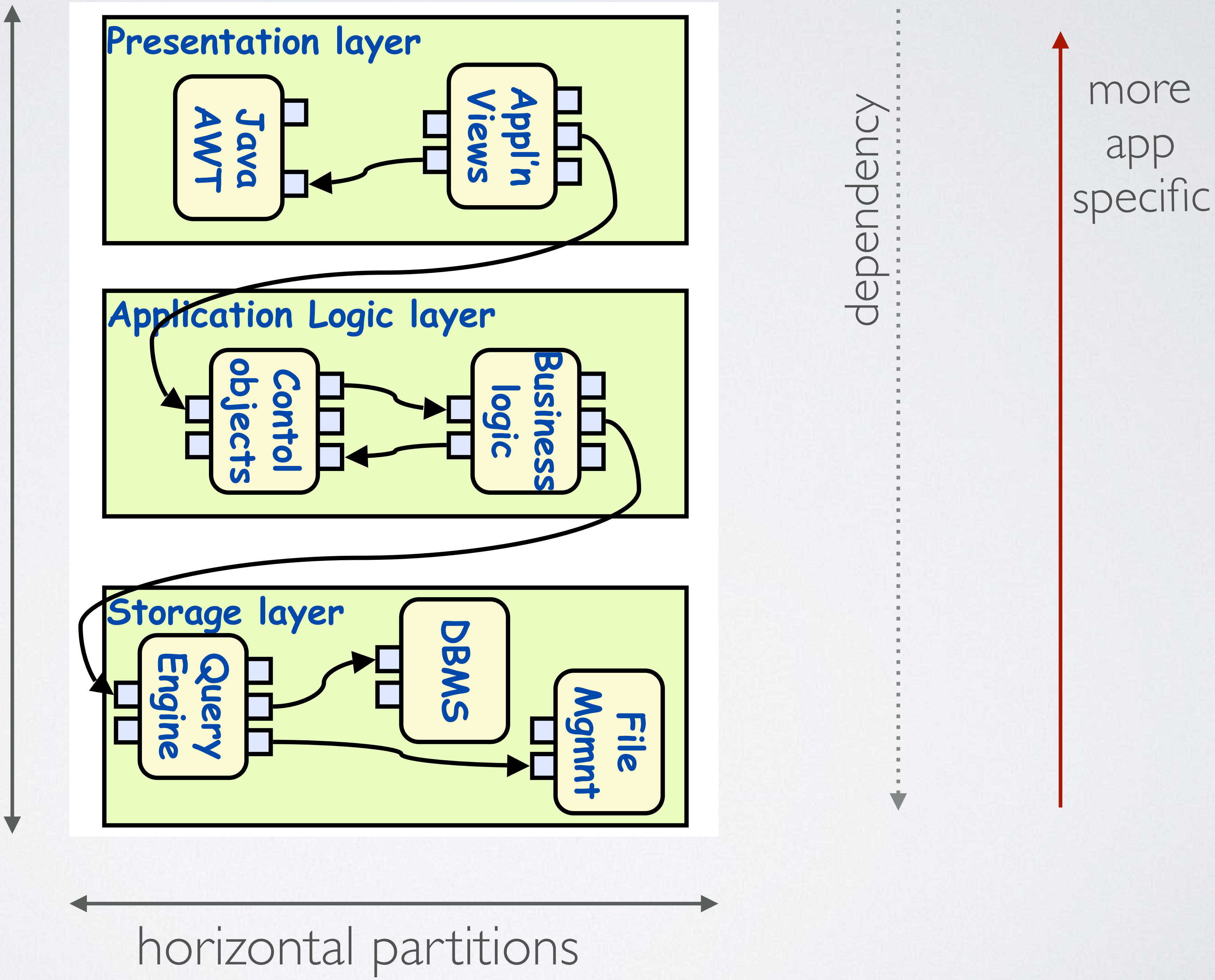
Interesting properties

- Support increasing levels of abstraction during design
- Support enhancement (add functionality) and re-use
- can define standard layer interfaces

Disadvantages

- May not be able to identify (clean) layers

EXAMPLE: 3-LAYER DATA ACCESS



vertical layers

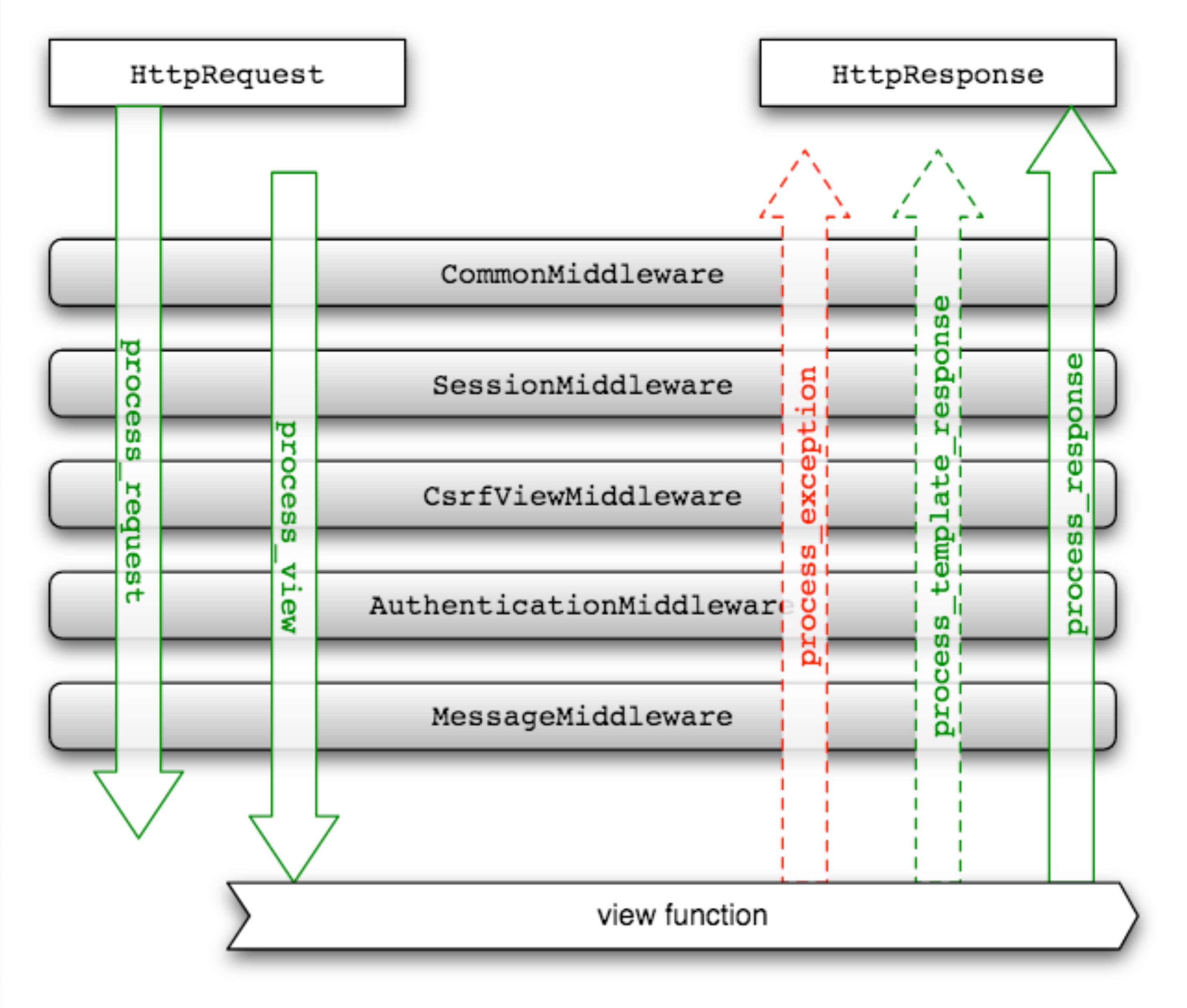
horizontal partitions

dependency

more app specific

EXAMPLE: DJANGO FRAMEWORK

- How does Django (Python Web framework) handle HTTP requests?



EXAMPLE: DJANGO FRAMEWORK

- How does Django (Python Web framework) handle HTTP requests?

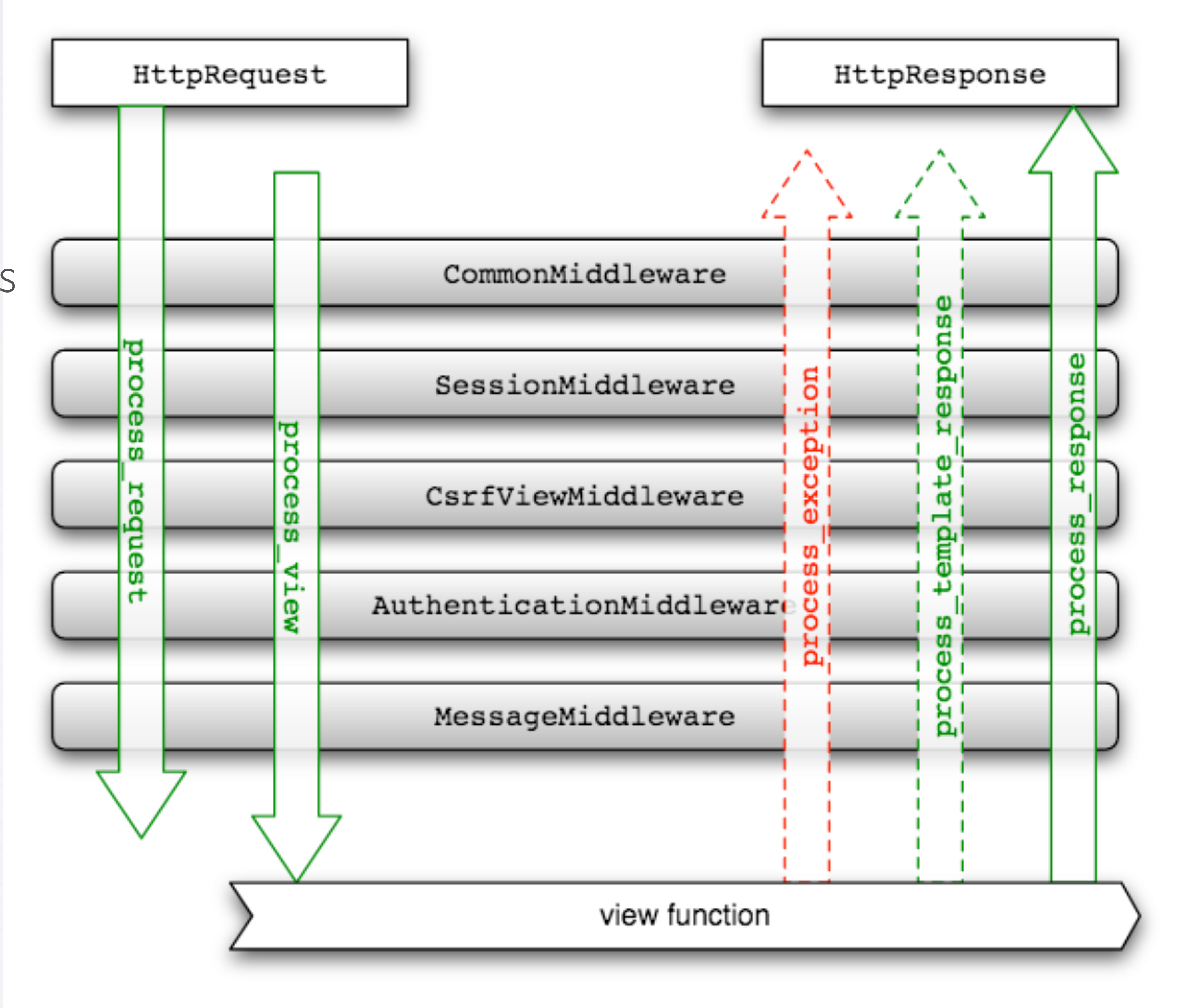
Handles ETag - identifier for caching purposes

Sessions and cookies

Cross Site Request Forgery

Adds user attribute in controller

Cookie- and session-based messages



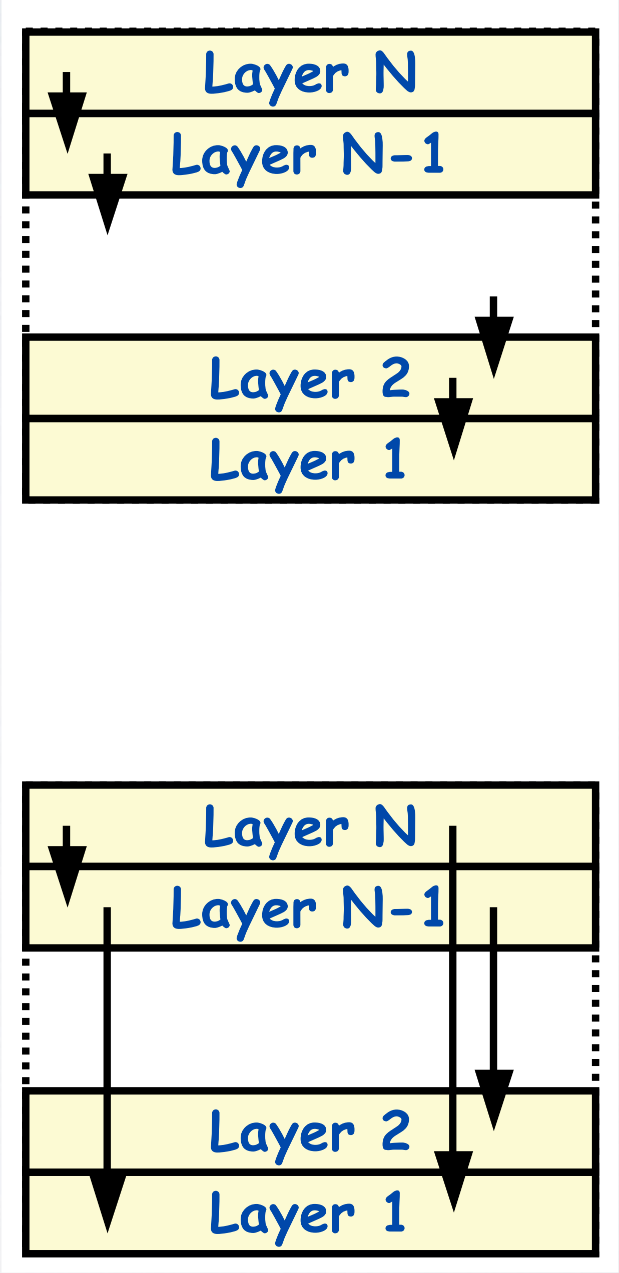
OPEN VS CLOSED ARCHITECTURE

closed architecture

- each layer only uses services of the layer immediately below;
- minimises dependencies between layers and reduces the impact of a change.

open architecture

- a layer can use services from any lower layer.
- more compact code, as the services of lower layers can be accessed directly
- breaks the encapsulation of layers, so increase dependencies between layers



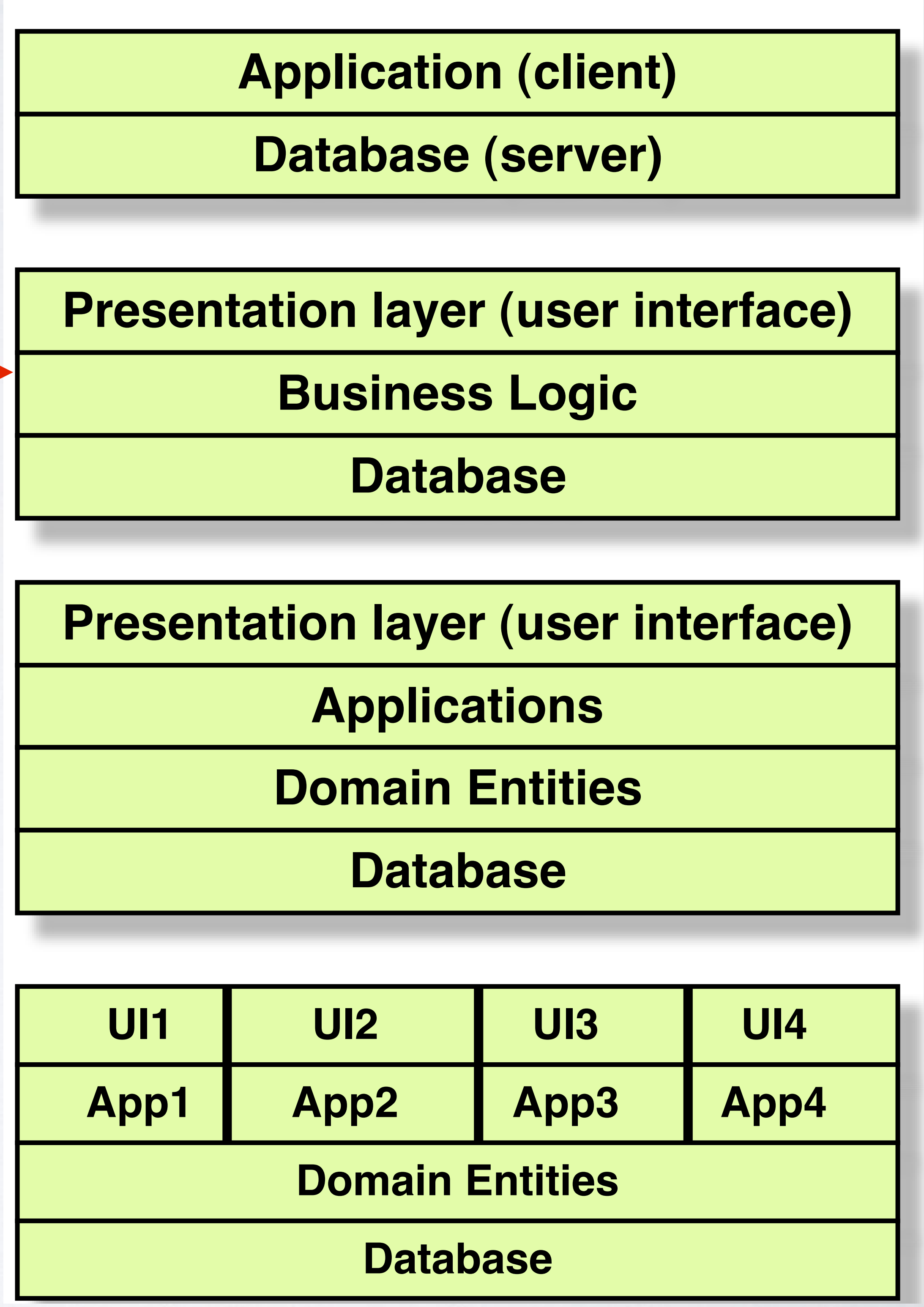
HOW MANY LAYERS?

2 layers:
- application layer
- database layer
- e.g., simple client-server model

3 layers (three tier):
- separate out the business logic
  - helps make both user interface and database layers modifiable

4 layers (four tier):
- separate applications from the domain entities that they use
  - boundary classes in presentation layer
  - control classes in application layer
  - entity classes in domain layer

partitioned 4 layers:
- identify separated applications



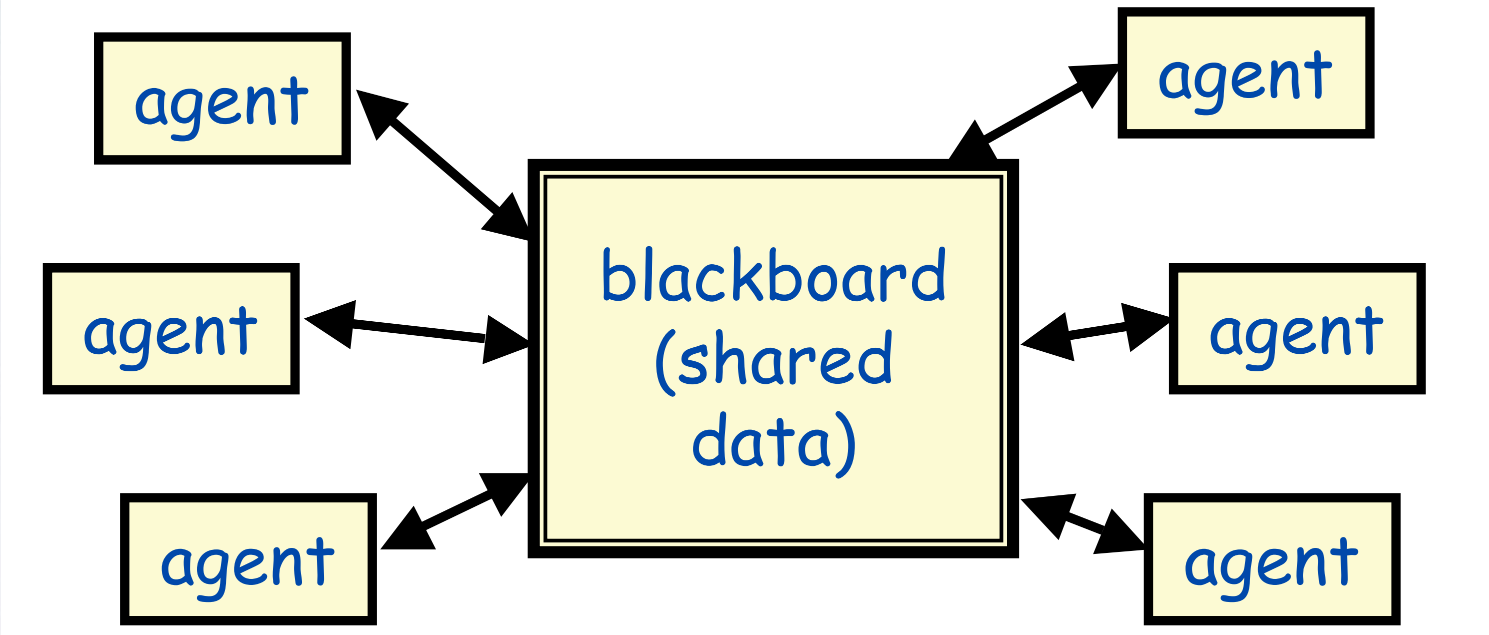
Application (client)
Database (server)

Presentation layer (user interface)
Business Logic
Database

Presentation layer (user interface)
Applications
Domain Entities
Database

UI1	UI2	UI3	UI4
App1	App2	App3	App4
Domain Entities			
Database			

REPOSITORIES



Examples

- databases
- blackboard expert systems
- programming environments

Interesting properties

- adding new applications (agent) is easy
- **reduce the need to duplicate complex data**

Disadvantages

- blackboard becomes a bottleneck

REPOSITORY

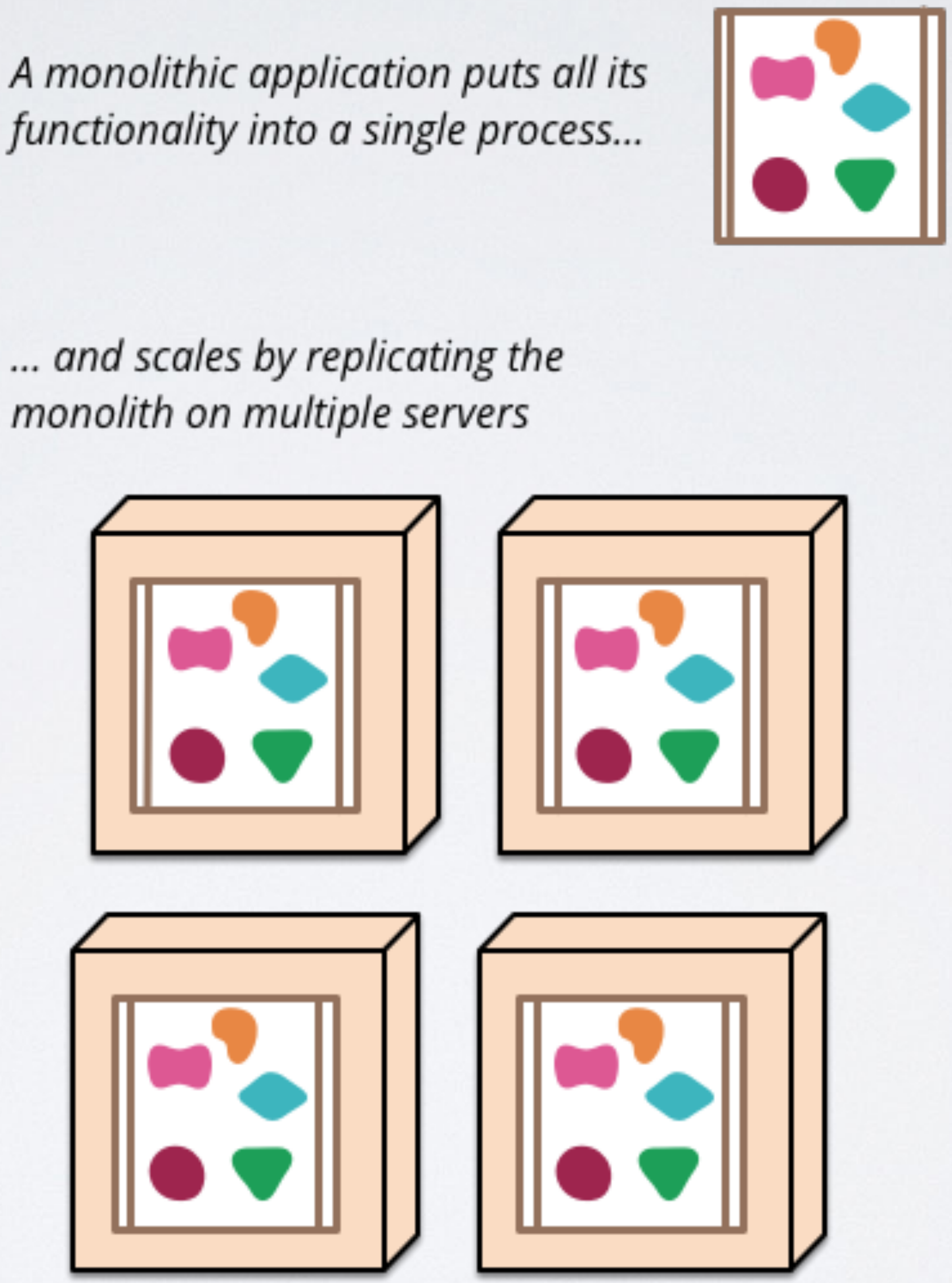
Sub-systems access and modify a single data structure

Concurrency & data consistency

Disadvantage: Possibly performance bottlenecks and reduced modifiability

E.g. databases, IDE's, tuple spaces

MICROSERVICE

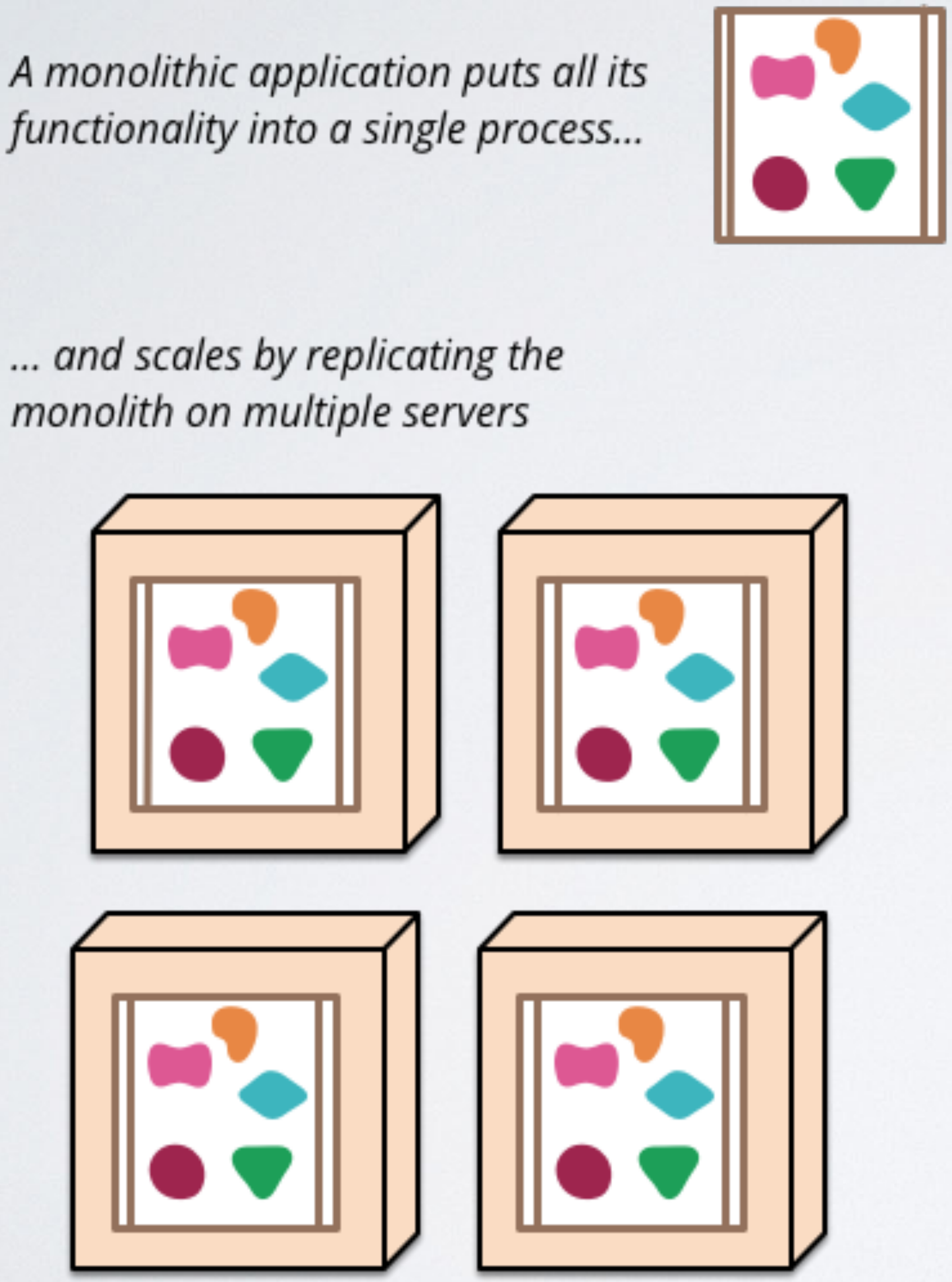


A monolithic application puts all its functionality into a single process...

... and scales by replicating the monolith on multiple servers

https://www.martinfowler.com/articles/microservices.html

MICROSERVICE

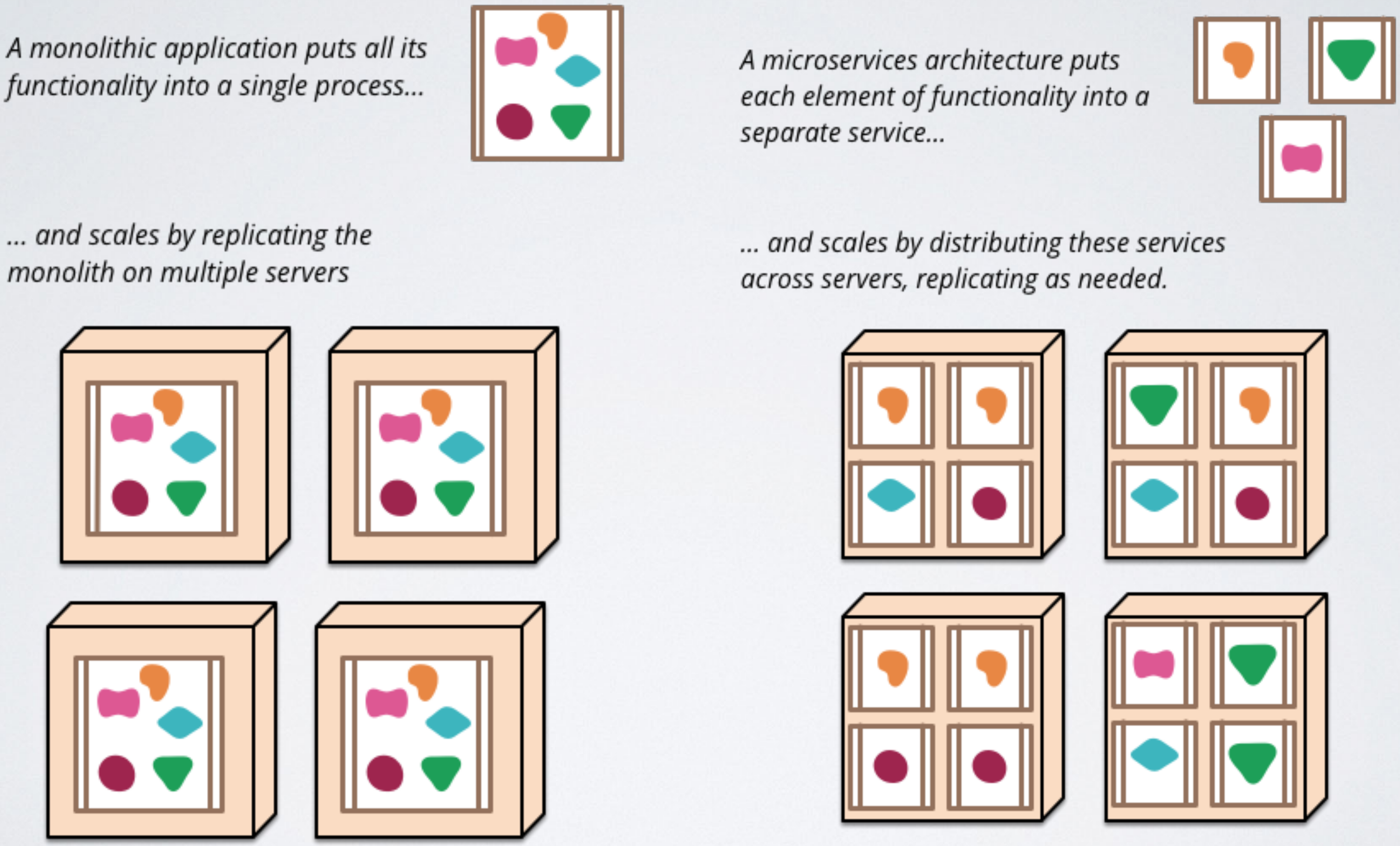


A monolithic application puts all its functionality into a single process...

... and scales by replicating the monolith on multiple servers

https://www.martinfowler.com/articles/microservices.html

MICROSERVICE



A monolithic application puts all its functionality into a single process...

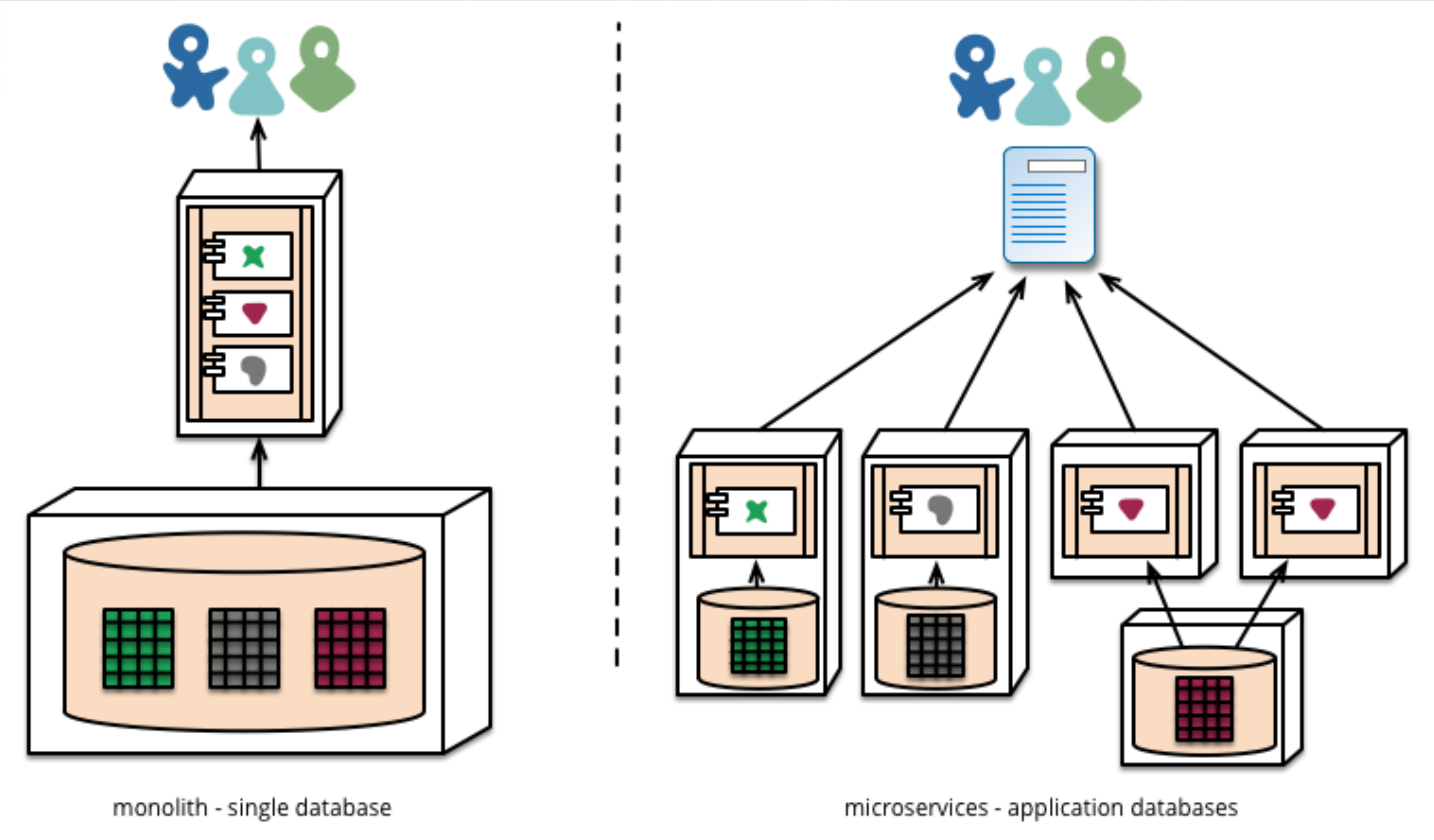
... and scales by replicating the monolith on multiple servers

A microservices architecture puts each element of functionality into a separate service...

... and scales by distributing these services across servers, replicating as needed.

https://www.martinfowler.com/articles/microservices.html

MICROSERVICE


monolith - single database
microservices - application databases

https://www.martinfowler.com/articles/microservices.html

SERVERLESS

No Server?

src: https://aws.amazon.com/lambda/

SERVERLESS

No Server? **Not quite!**

src: https://aws.amazon.com/lambda/

SERVERLESS

No Server? **Not quite!**

No maintenance! No provisioning!

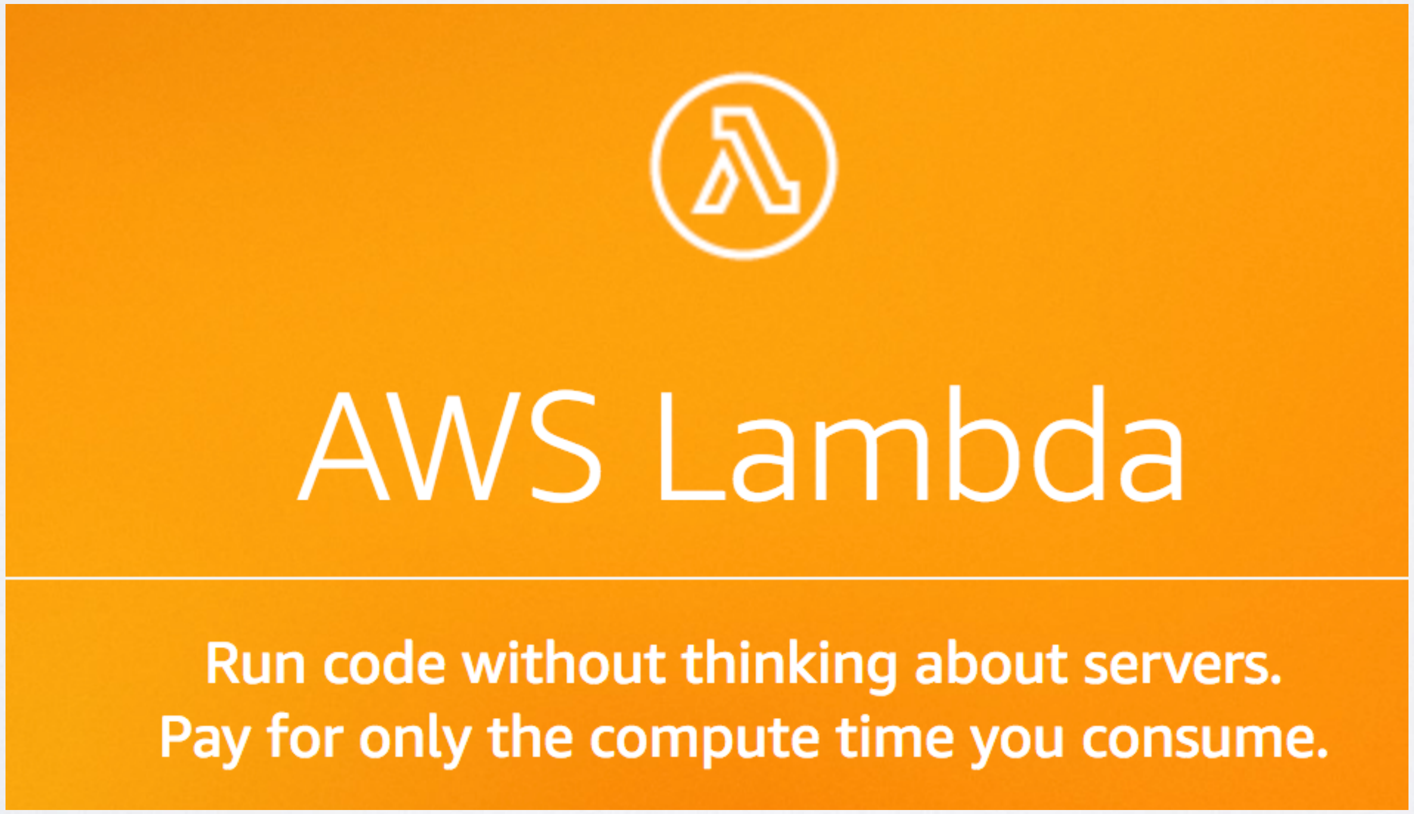
src: https://aws.amazon.com/lambda/

SERVERLESS

No Server? **Not quite!**

No maintenance! No provisioning!

Utilise other (third party) services



AWS Lambda

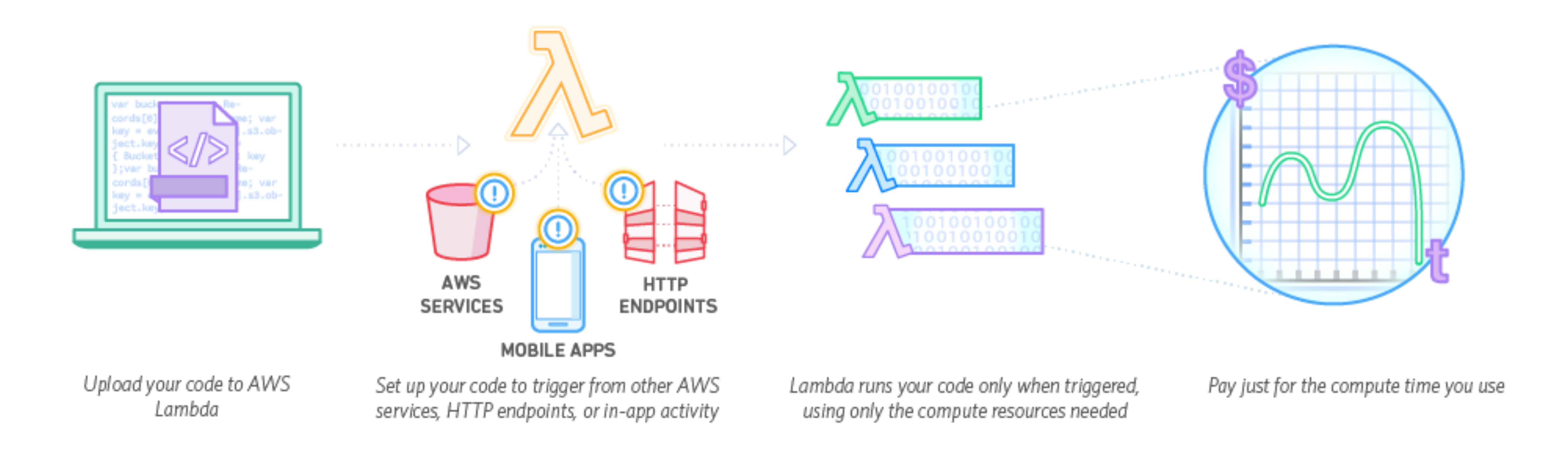
Run code without thinking about servers.
Pay for only the compute time you consume.

Video

src: https://aws.amazon.com/lambda/

SERVERLESS

- Lambdas are stateless functions (pure functions)



Upload your code to AWS Lambda

Set up your code to trigger from other AWS services, HTTP endpoints, or in-app activity

Lambda runs your code only when triggered, using only the compute resources needed

Pay just for the compute time you use

src: https://aws.amazon.com/lambda/

CONCLUDING REMARKS

CONCLUDING REMARKS

What **software architecture** is and why it is interesting

Who are the **stakeholders**

What **software qualities** does software architecture concern

UML diagrams expressing aspects of software architecture

Architectural styles or software architectural design patterns