Backend Basic Knowledge

Deriving backend architecture from constraints perspective

warren.wu@visionwx.com

2022-09-13 13:30-14:30

后端基础知识

从约束视角推导后端架构

warren.wu@visionwx.com

2022-09-13 13:30-14:30

“ The more constraints one imposes, the more one frees one
—— *Igor Stravinsky* ”

“ 一个人施加的约束越多，他就越能解放自己
—— *伊戈尔·斯特拉文斯基* ”

Approach

adding constraints incrementally to gain architectural properties

推导方式

渐进式地添加约束以获取架构特性

3 parts

- systerm level architecture constraints
- container level architecture constraints
- component level architecture constraints

三个部分

- 系统层架构约束
- 容器层架构约束
- 组件层架构约束

part1 systerm level architecture constraints

- No constraints
- Client-Server
- Stateless
- Cache
- Layered systerm
- Code on demand
- Uniform interface

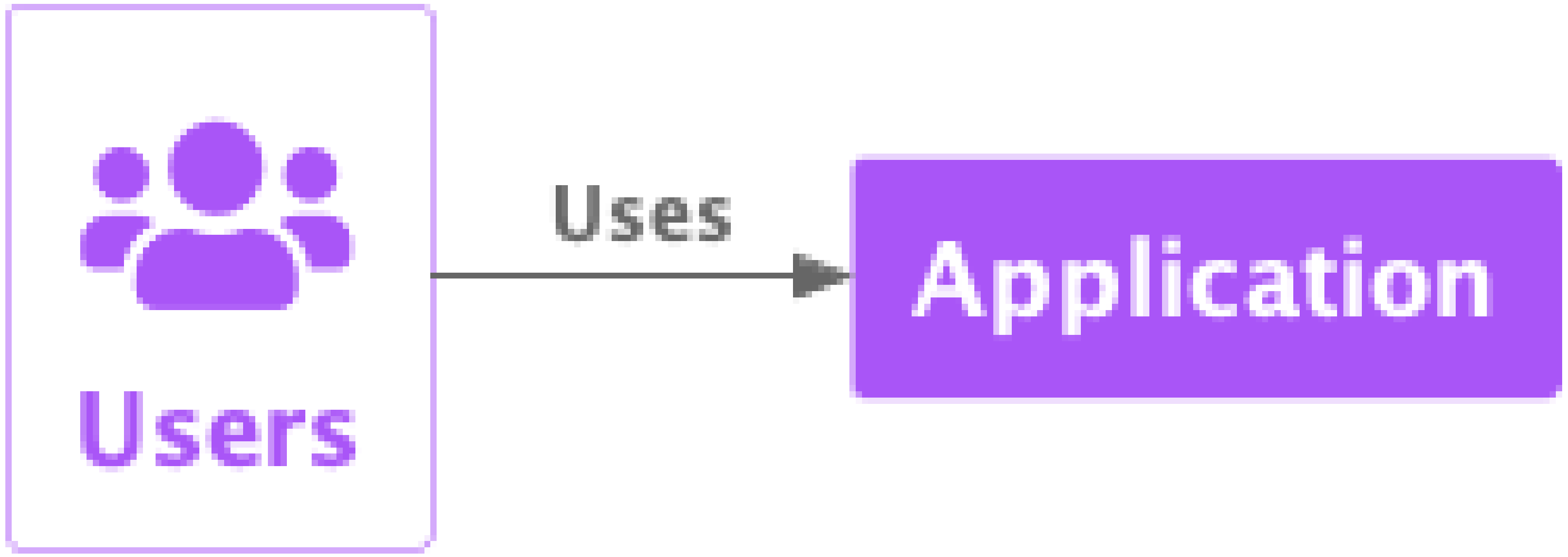
第一部分 系统层架构约束

- 无约束
- 客户端-服务端
- 无状态
- 缓存
- 分层系统
- 统一接口风格

No constraints

无约束

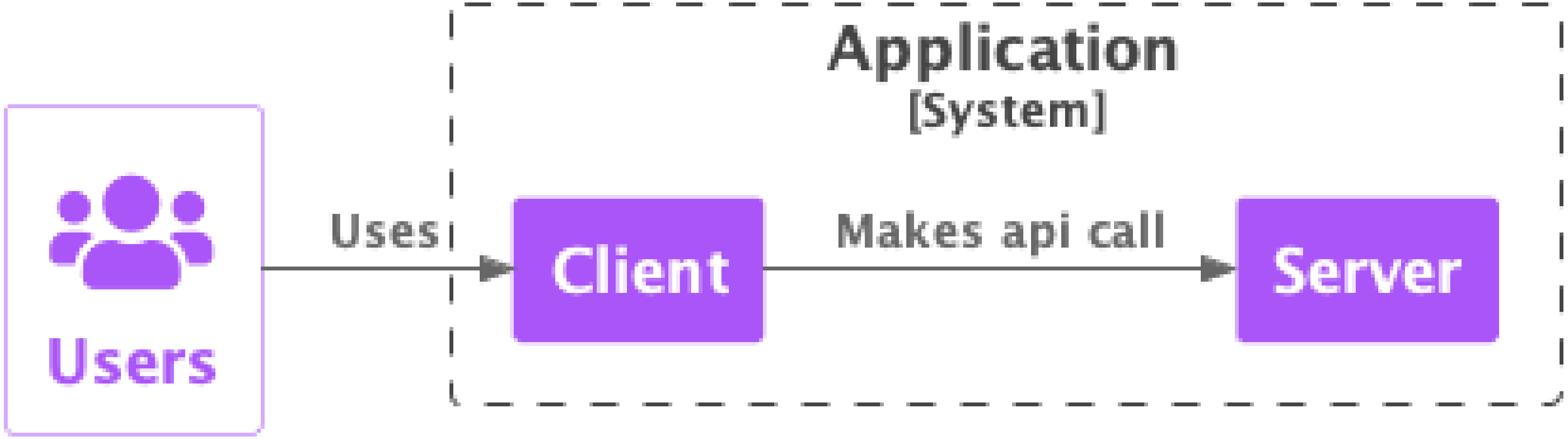
No Contraints



Client-Server

客户端-服务端模式

Client–Server Architecture



Legend

▯ person

▯ container

▯ system boundary (dashed, transparent)

client-server benefits and trade-off

- scalability +
- simplicity +
- evolvablity +

客户端-服务端模式优点和代价

- 可伸缩性 +
- 简单性 +
- 可演进性 +

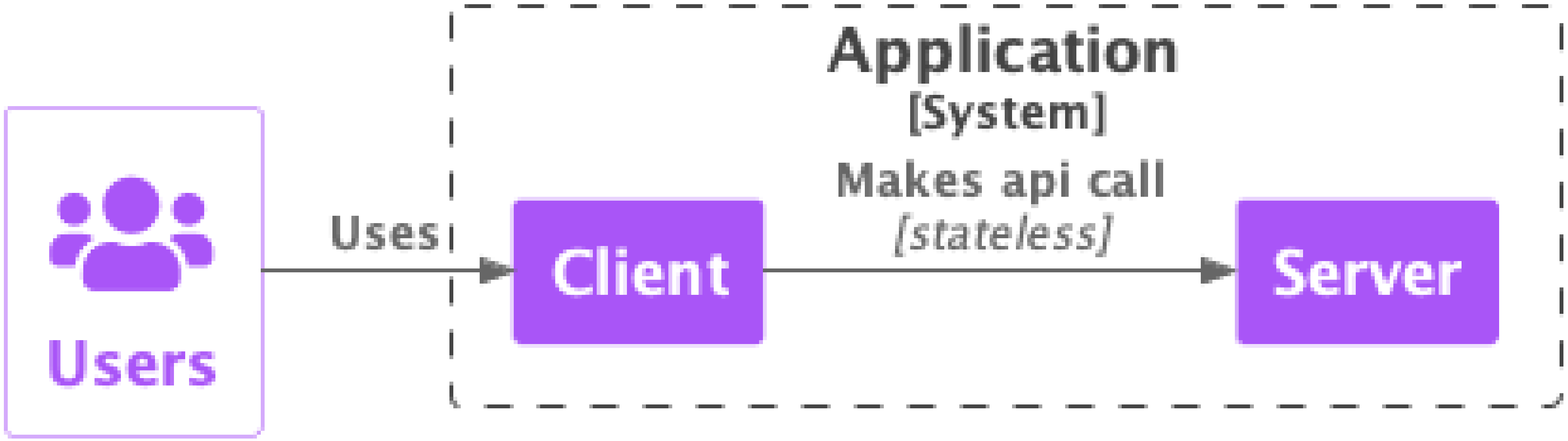
Stateless

Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server.

无状态

从客户端到服务端的每个请求必须包含理解请求所必须的全部信息，不能任何依赖服务端存储的上下文

Client–Stateless–Server Architecture



Legend

▯ person
▯ container
▯ system boundary (dashed, transparent)

stateless benefits and trade-off

- network performance -
- scalability +
- visibility +
- reliability +

无状态的优点和代价

- 网络性能 -
- 可伸缩性 +
- 可观测性 +
- 可靠性 +

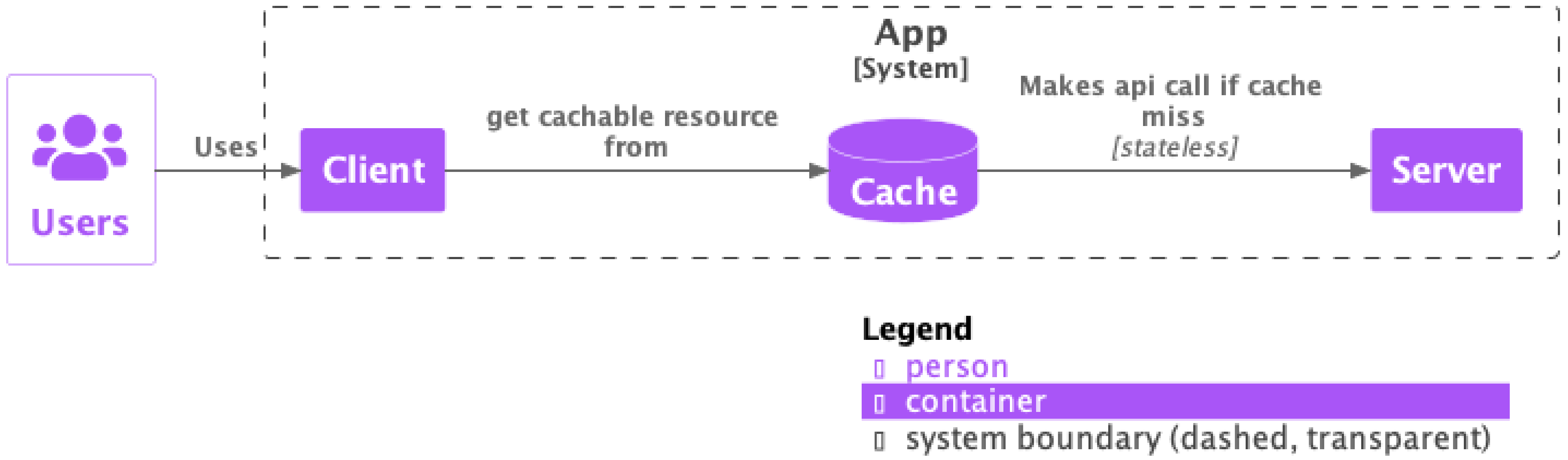
cache

The data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable

缓存

请求的响应中的数据必须显式或者隐式地标志是否可缓存

Client-Cache-Stateless-Server Architecture



App
[System]
Users
Uses
Client
get cachable resource from
Cache
Makes api call if cache miss
[stateless]
Server
Legend
▯ person
▯ container
▯ system boundary (dashed, transparent)

cache benefits and trade-off

- user-perceived performance +
- efficiency +

缓存的优点和代价

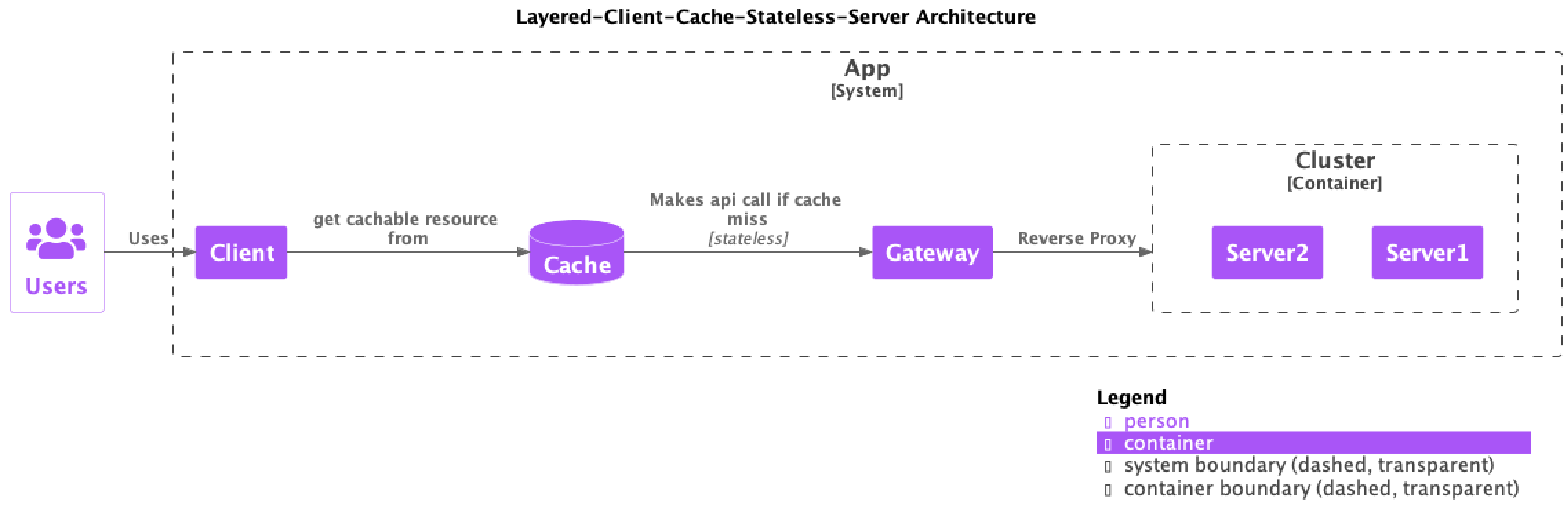
- 用户可感的性能 +
- 高效性 +

layered systerm

Each component cannot "see" beyond the immediate layer with which they are interacting.

分层系统

除了直接交互的层，每一个组件不能"看见"的其他层


Layered-Client-Cache-Stateless-Server Architecture
App
[System]
Cluster
[Container]
Users
Uses
Client
get cachable resource from
Cache
Makes api call if cache miss
[stateless]
Gateway
Reverse Proxy
Server2
Server1
Legend
person
container
system boundary (dashed, transparent)
container boundary (dashed, transparent)

layered systerm benefits and trade-off

- network performance -
- user-perceived performance -
- scalability +
- simplicity +
- evolvablity +
- reuseability +
- portability +

分层系统的优点和代价

- 网络性能 -
- 用户感知性能 -
- 可伸缩性 +
- 简单性 +
- 可演进性 +
- 可重用性 +
- 可移植性 +

uniform interface

- identification of resources
- manipulation of resources through representations
- self-descriptive messages

统一接口风格

- 资源可唯一识别
- 通过表征操作资源
- 自描述消息

Example

- GET /workspaces/1/trickles?limit=5&memberId=2
- POST /workspaces/1/groups/3/trickles
- PATCH /workspaces/1/trickles/5
- DELETE /workspaces/1/trickles/5

uniform interface benefits and trade-off

- network performance -
- simplicity +
- configurable +
- reuseability +
- visibility +

part2 container level architecture constraints

- No Constraints
- Layered
- Domain layer constraints
- Outbound layer constraints
- Inbound layer constraints
- Application layer

第二部分 容器级别架构约束

- 无约束
- 分层
- 领域层约束
- 向外层约束
- 向内层约束
- 应用层约束

No constraints

use whatever style you like if you can implement within 200 lines of codes, and you are sure it is not going to grow in future

无约束

如果你能用低于200行代码实现, 并且未来也不会超过这个规模，那就用你最喜欢的方式

layered

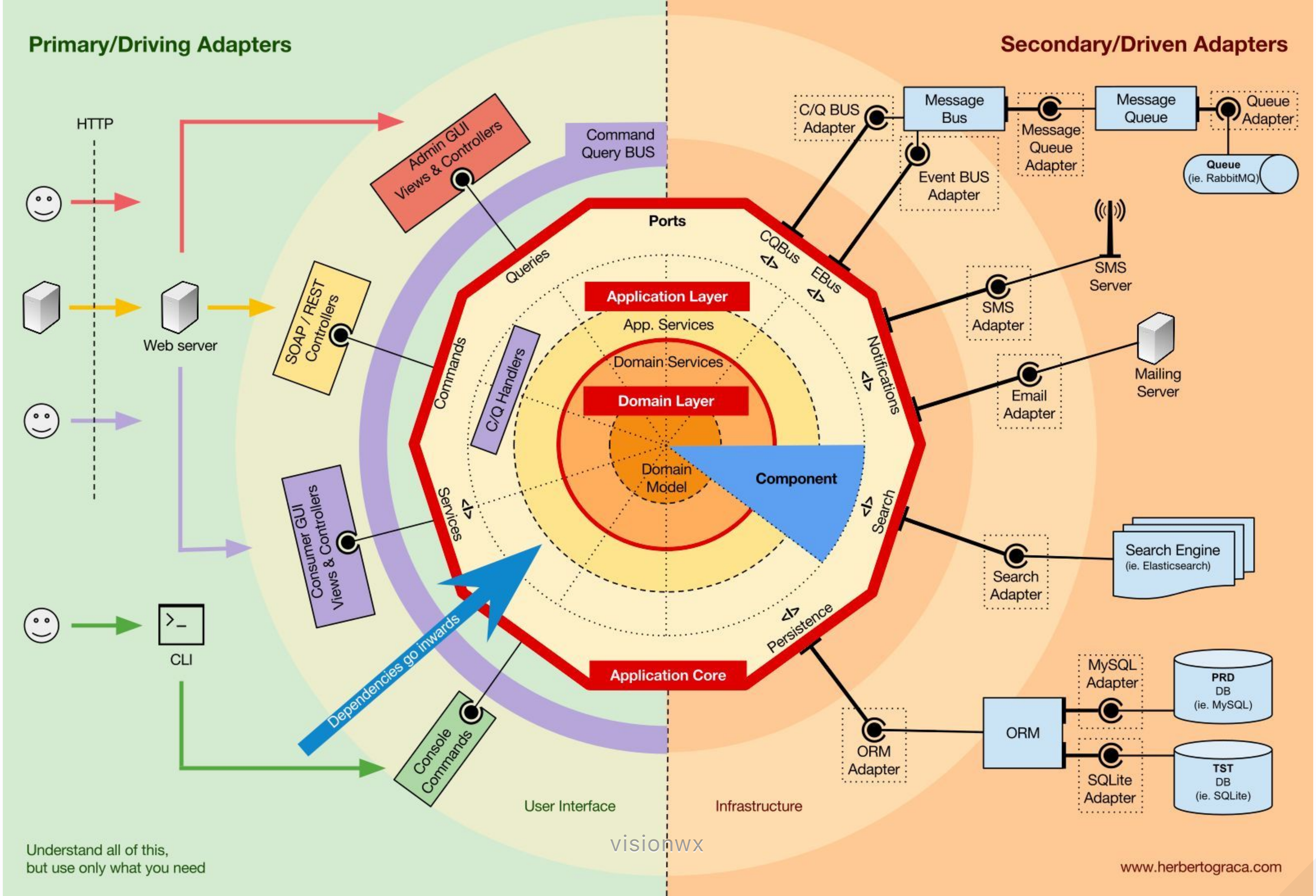
options:

- 2 layer: inbound + outbound
- 3 layer: inbound + domain + outbound
- 4 layer: inbound + application + domain + outbound
- hexgonal: adapters(application(domain))

分层

选项：

- 2层： 向内层+向外层
- 3层： 向内层+领域层+向外层
- 4层： 内向层+应用层+领域层+外向层
- 六边形： 适配器层(应用层(领域层)))



Primary/Driving Adapters
Secondary/Driven Adapters
HTTP
Web server
CLI
Admin GUI Views & Controllers
SOAP / REST Controllers
Consumer GUI Views & Controllers
Console Commands
Command Query BUS
Ports
Queries
Commands
Services
C/Q Handlers
Application Layer
App. Services
Domain Services
Domain Layer
Domain Model
Component
Application Core
Dependencies go inwards
CQBus
EBus
Notifications
Search
Persistence
C/Q BUS Adapter
Message Bus
Event BUS Adapter
Message Queue Adapter
Message Queue
Queue Adapter
Queue (ie. RabbitMQ)
SMS Server
SMS Adapter
Mailing Server
Email Adapter
Search Engine (ie. Elasticsearch)
Search Adapter
ORM Adapter
ORM
MySQL Adapter
PRD DB (ie. MySQL)
SQLite Adapter
TST DB (ie. SQLite)
User Interface
Infrastructure
Understand all of this, but use only what you need
www.herbertograca.com

如何抉择

- 纯技术服务：2层就够
- 有业务逻辑：至少要3层
- 需要多进程、事件驱动、需要切面：建议4层
- 需要多入口、减少基础设施依赖：建议六边形

Domain level constraints

options:

- transaction script
- table module
- domain model

领域层约束

选项：

- 事务脚本
- 表模块
- 领域模型

transaction script

use process to organize business logic, each process comes from one request from api/ui level

事务脚本

使用过程来组织业务逻辑，每个过程对应来自inboud层的一个请求

domain model exmaple

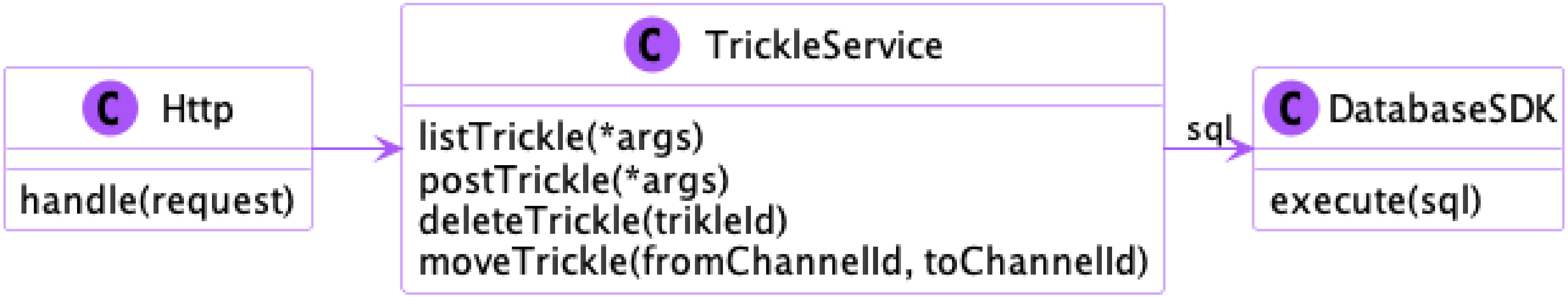


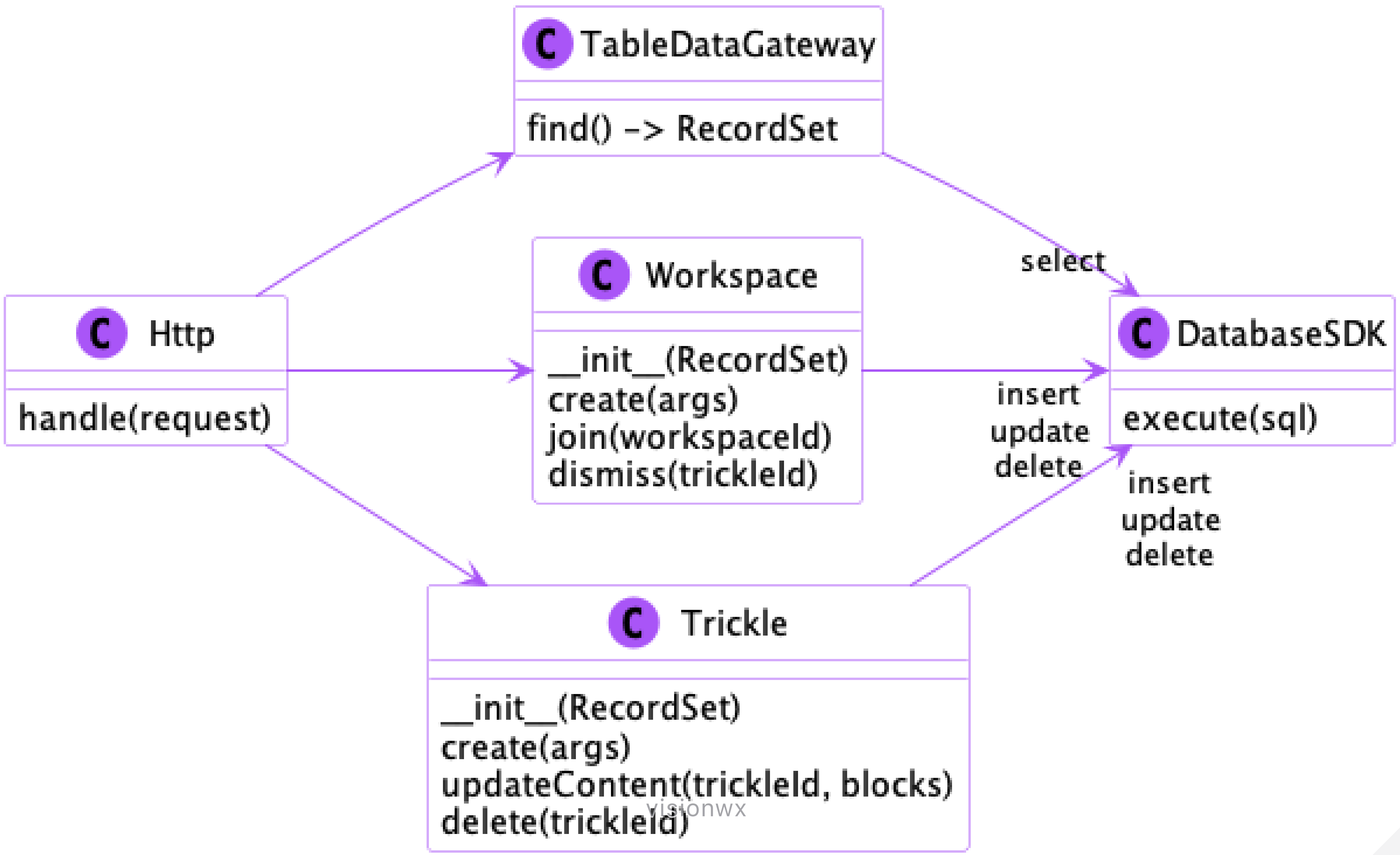
table module

map one class to one table in db, use signle class instance to perform various operation

表模块

一个类对应数据库中的一个表，使用单一的类实例来进行的各种操作程序

table module example



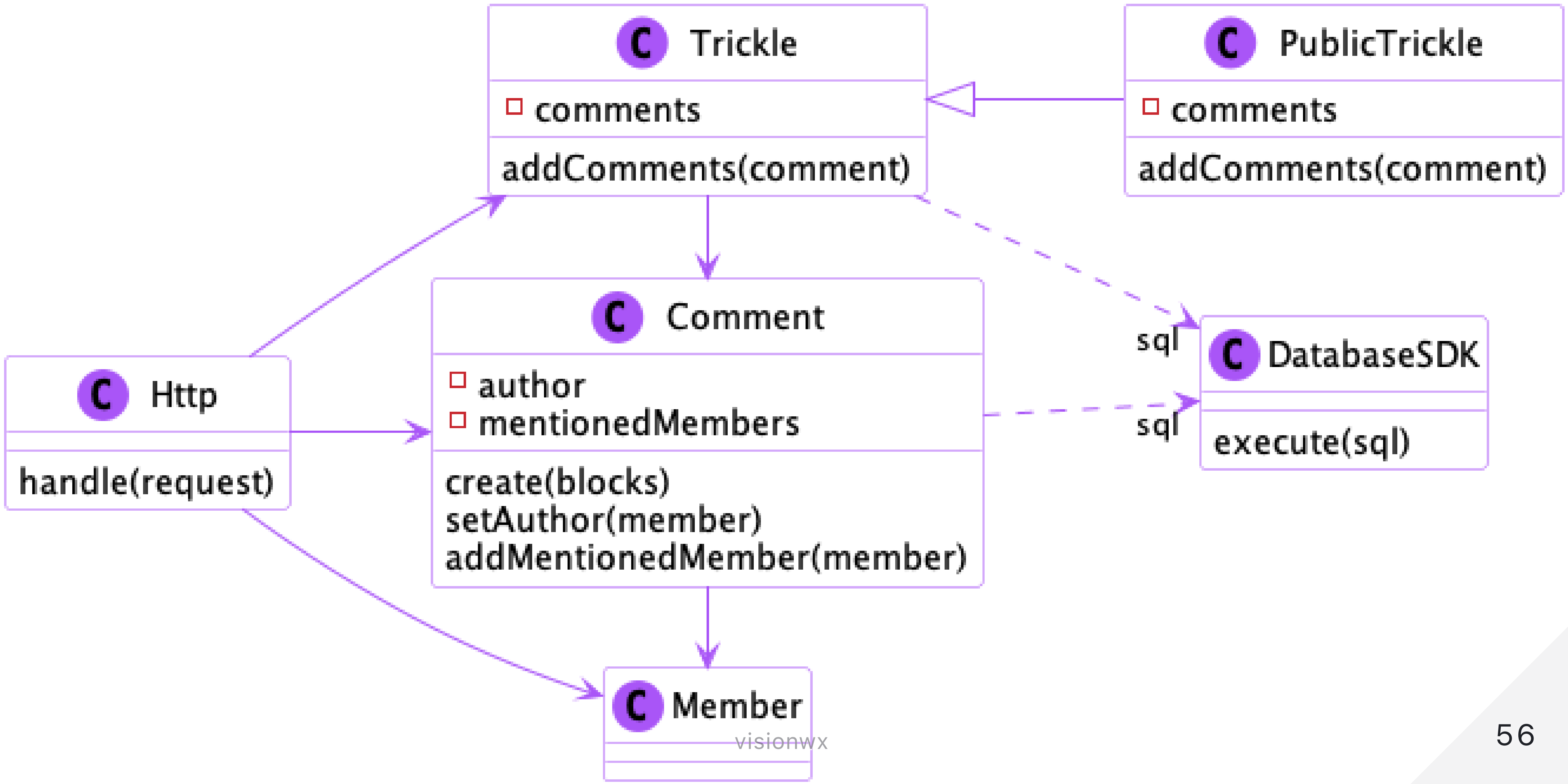
domain model

more attributes, relation network, support inheritance

领域模型

更多值属性，更复杂的关系网络，支持继承

domain model exmaple



how to choose

- major concern: domain complexity
- learning cost: ts < tm < dm
- complexity ceiling: ts < tm < dm

如何抉择

- 主要考虑： 领域复杂性
- 学习成本： 事务脚本 < 表模块 < 领域模型
- 复杂度上限： 事务脚本 < 表模块 < 领域模型

outbound layer constraints

options:

- table data gateway
- row data gateway
- active records
- object relation mapping

外向层约束

选项：

- 表数据门户
- 行数据门户
- 活跃记录
- 对象关系映射

table data gateway

an object that act as an entry to access to a table in database.
one instance handles all the rows in the table

表数据门户

一个充当数据库表访问入口的对象
一个实例处理表中所有的行

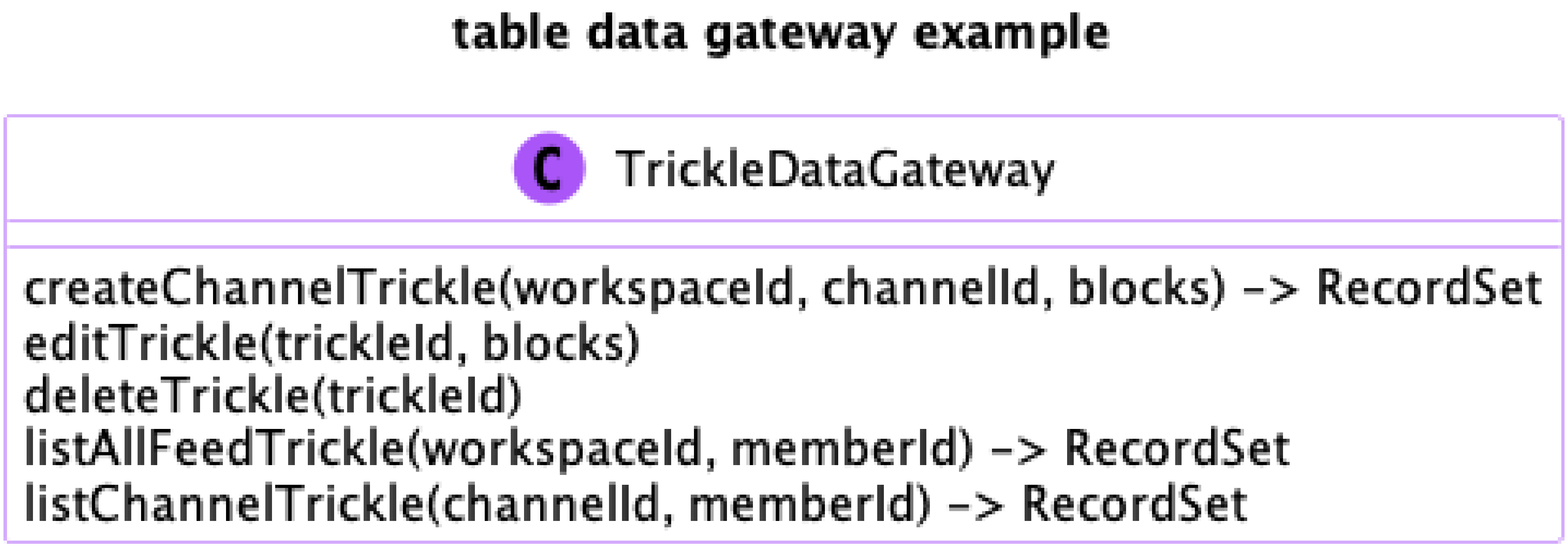


table data gateway example

TrickleDataGateway

```
createChannelTrickle(workspaceId, channelId, blocks) -> RecordSet
editTrickle(trickleId, blocks)
deleteTrickle(trickleId)
listAllFeedTrickle(workspaceId, memberId) -> RecordSet
listChannelTrickle(channelId, memberId) -> RecordSet
```

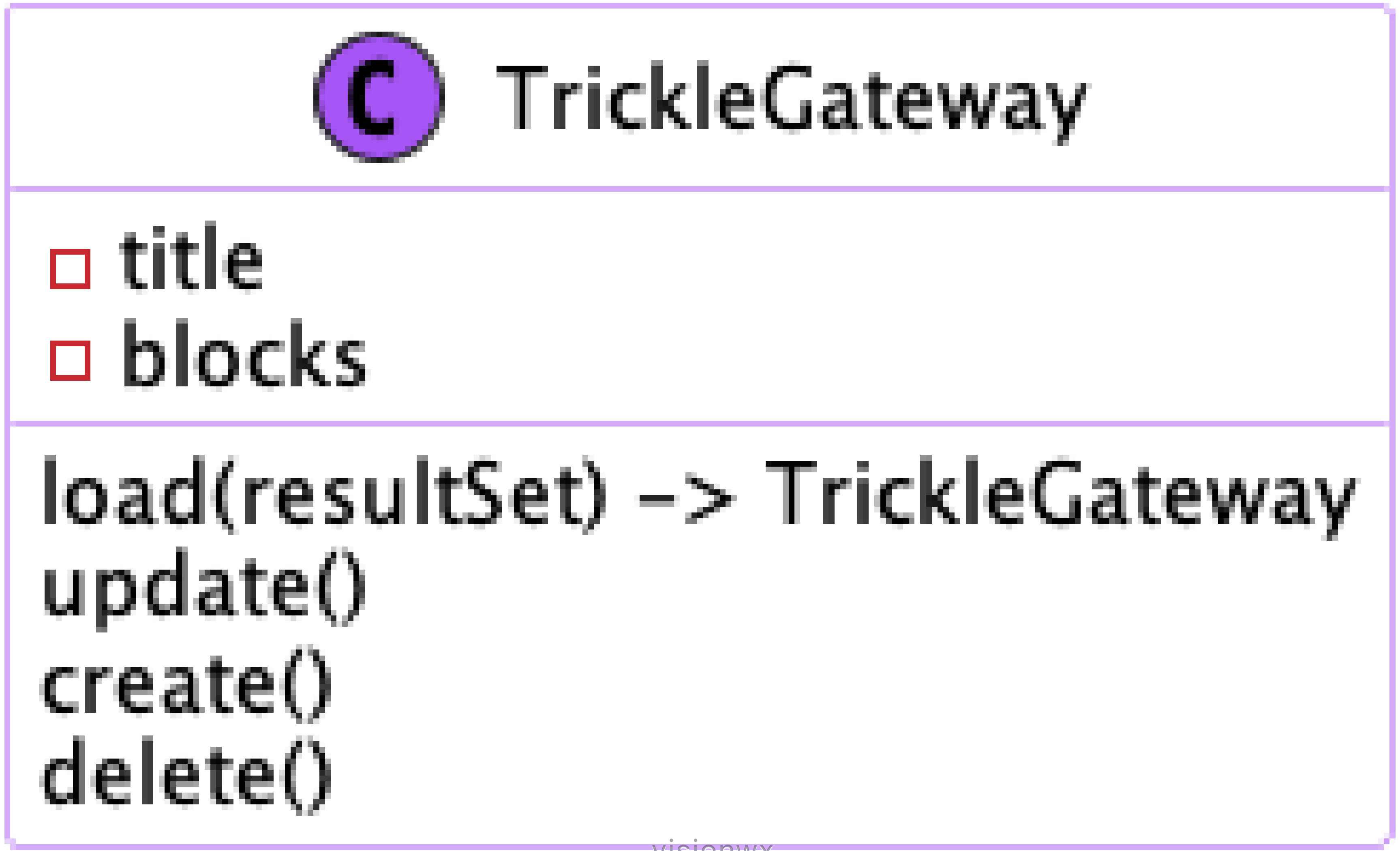
row data gateway

an object that act as an entry to access to a row in database.
one instance handles one row in the table

行数据门户

一个充当数据源中单条记录入口的对象
每行运行一个实例

row data gateway example



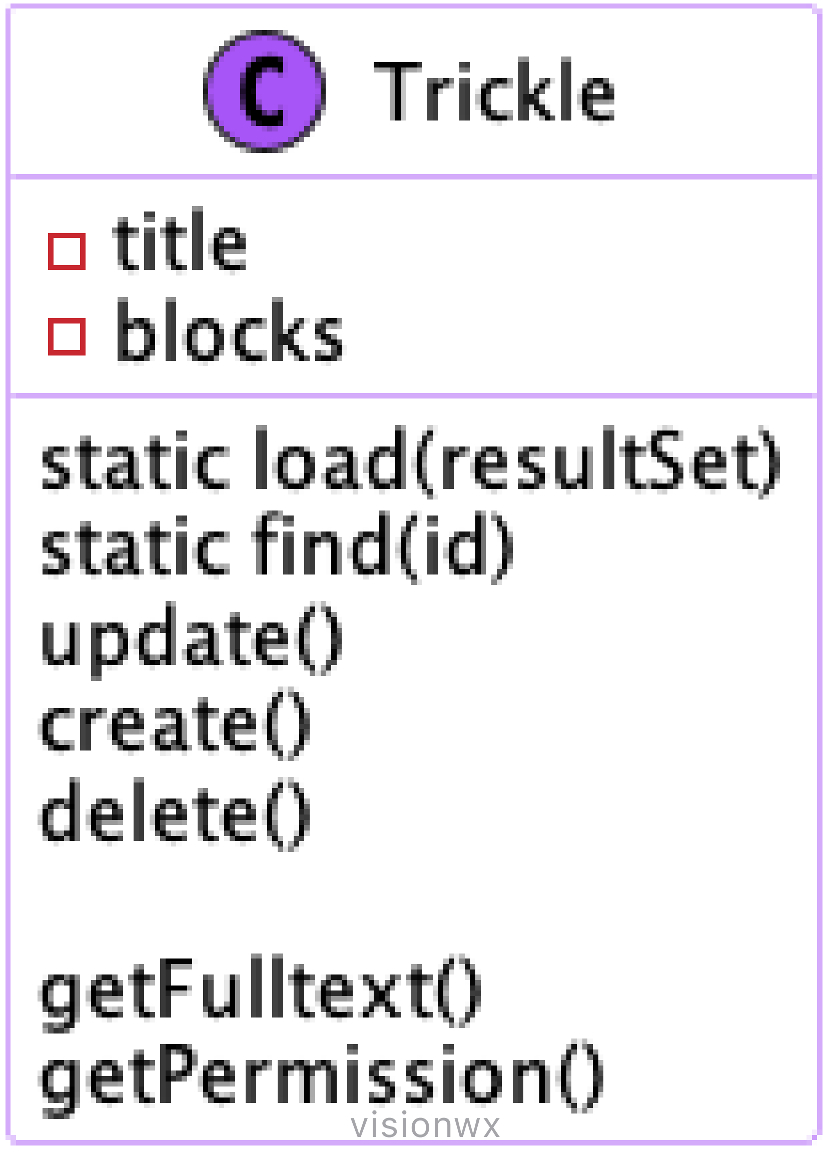
active record

An object that wraps a row in a database table or view, encapsulates database access, and adds domain logic to this data

活动记录

一个对象，它包装数据库表或视图中某一行，封装数据库访问，并在这些数据上增加了领域逻辑。

active record



```
static load(resultSet)
static find(id)
update()
create()
delete()

getFulltext()
getPermission()
```

unit of work

Maintains a list of objects affected by a business transaction and coordinates the writing out of changes and the resolution of concurrency problems

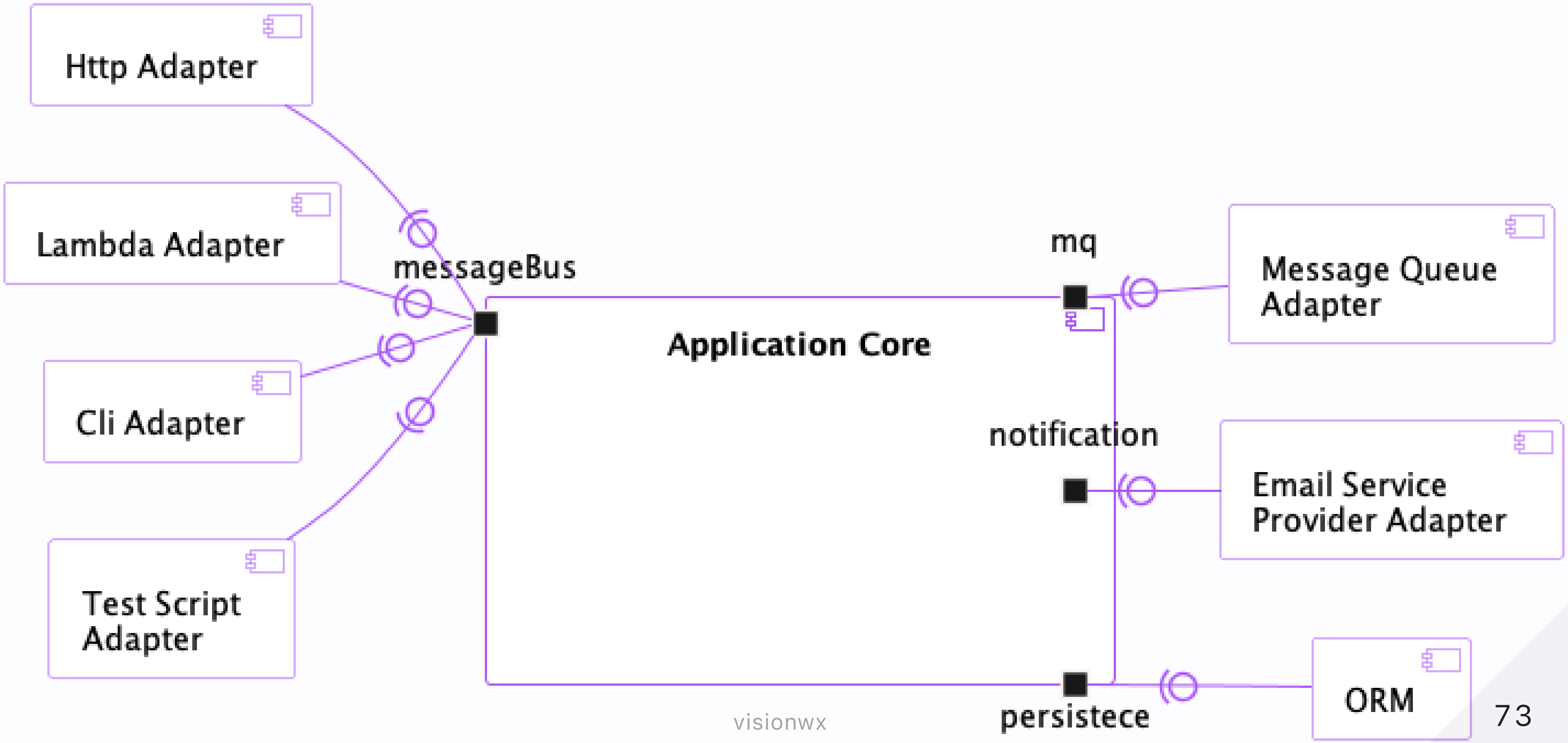
工作单元

维护一系列被业务事务影响的object，并协调变更的写入和并发冲突的解决

ports & adapters

- wrap protocols and infrastructure as adapters

ports & adaptors pattern



message bus

Comand Query Seperate

part3 component level architecture constraints

- programming paradigm
  - structured programming
  - object oriented programming
  - functional programming
- design principles
  - SRP
  - OCP
- static typing