Microservices with Domain-Driven Design #3/3

Credit – copy paste engineer

https://www.youtube.com/watch?v=EBjfiuJsYe4

เมนูวันนี้ (1/3)

Frontend talks to Microservices

- API Gateway
- Authentication

Microservices talk to Microservices

- Synchronous
- Asynchronous: Message Broker

Logging and Monitoring

เมนูวันนี้ (2/3)

Data Access/Manipulation

- Repository Pattern
- Aggregate Pattern
- CQRS

Event-based Communication

- Event Sourcing
- Transactional Outbox

เมนูวันนี้ (3/3)

Hexagonal Architecture

Demo + Code with Python FastAPI & Kafka & MongoDB

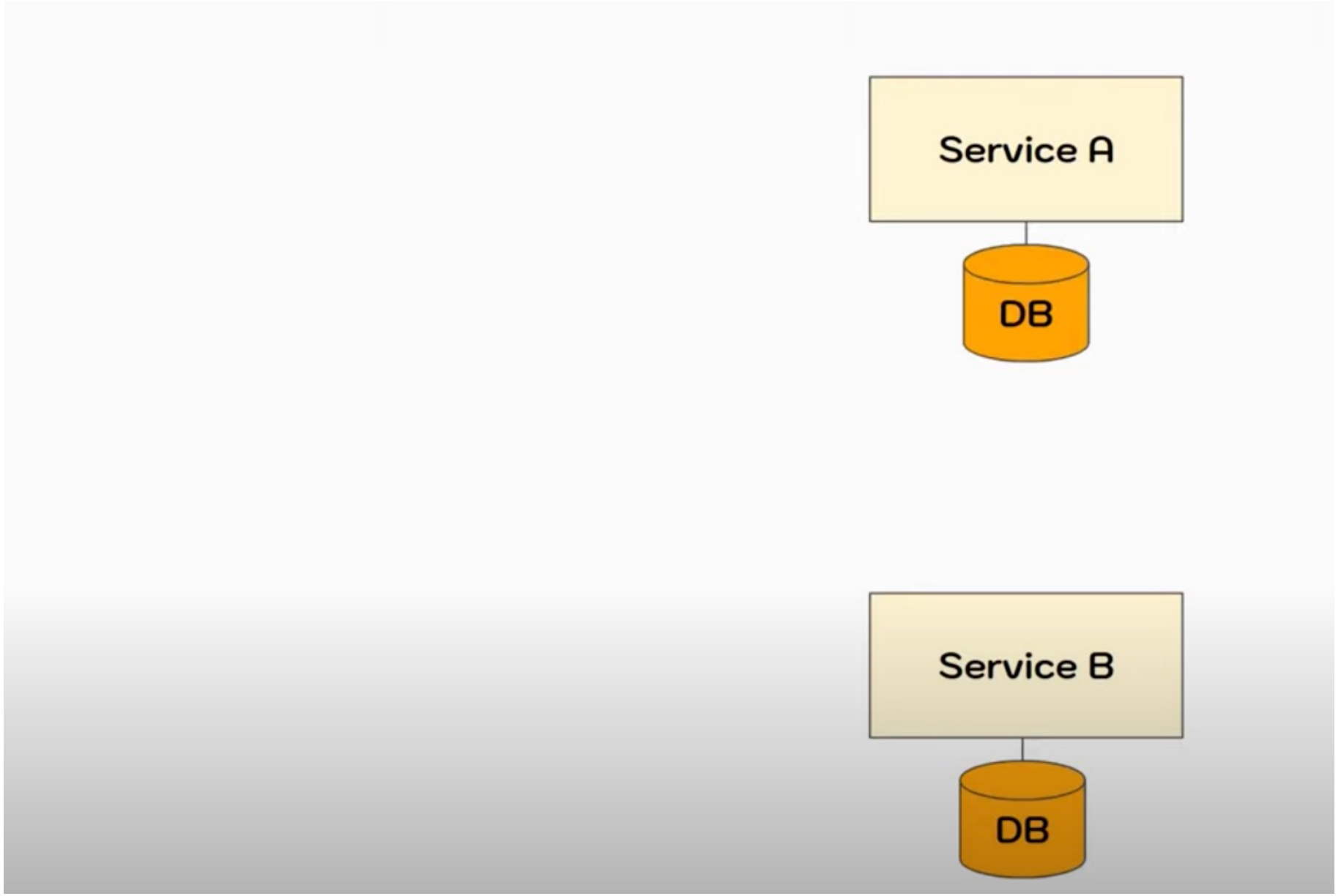
- Message Broker
- Transactional Outbox
- Repository Pattern
- Aggregate Pattern

ถ้าทำ Microservices

แล้วต้องใช้ architecture ตามนี้ ?

ถ้าทำ Domain-Driven Design

แล้วต้องใช้ architecture ตามนี้ ?


Service A
DB
Service B
DB

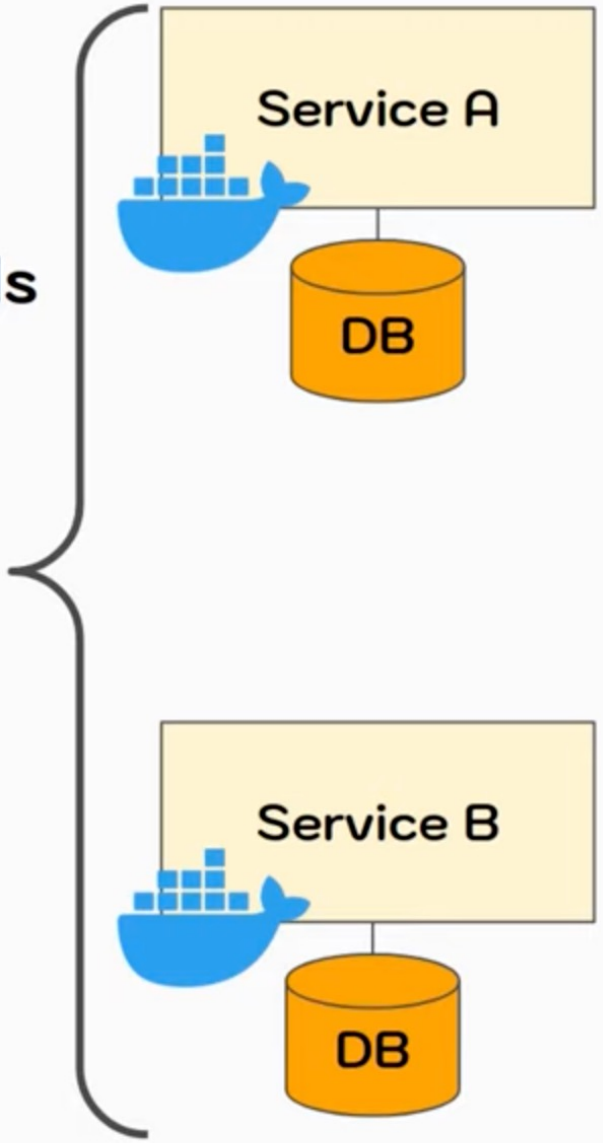
รันบน container orchestration tools เช่น

Kubernetes Cluster

AWS ECS

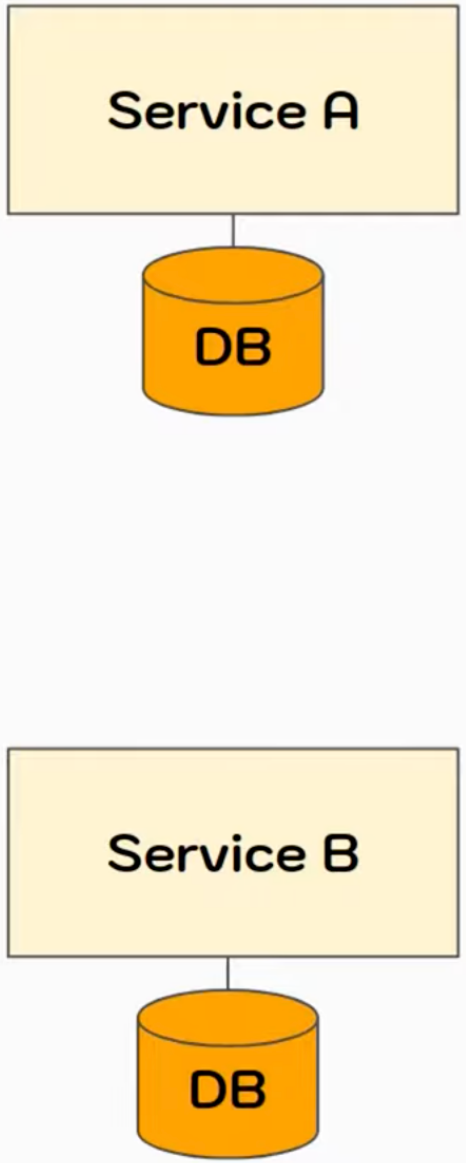
Cloud Run

AWS Fargate

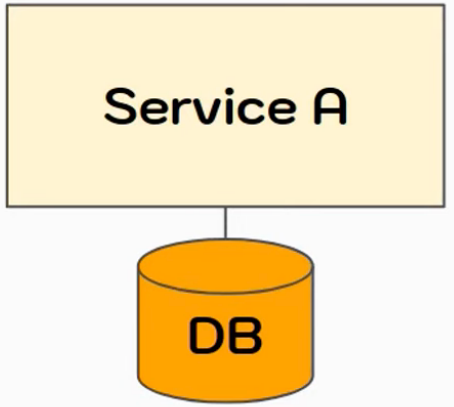


ปัญหา:

service อยู่แยกกัน



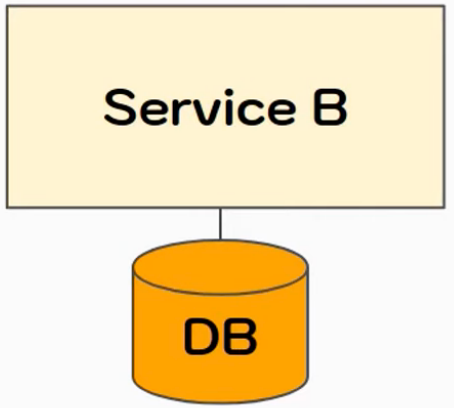
IP: xxx.xxx.xxx.xxx

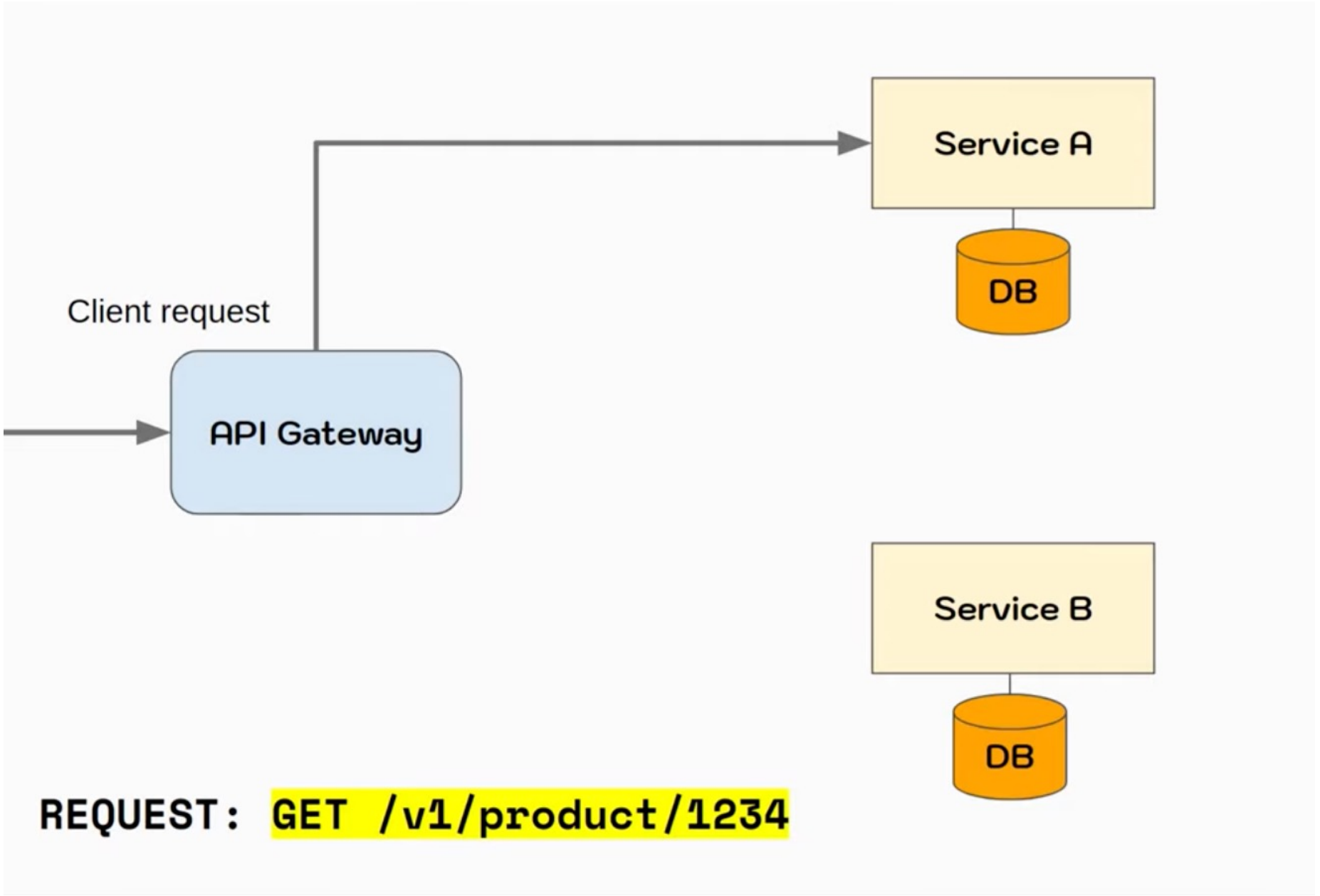


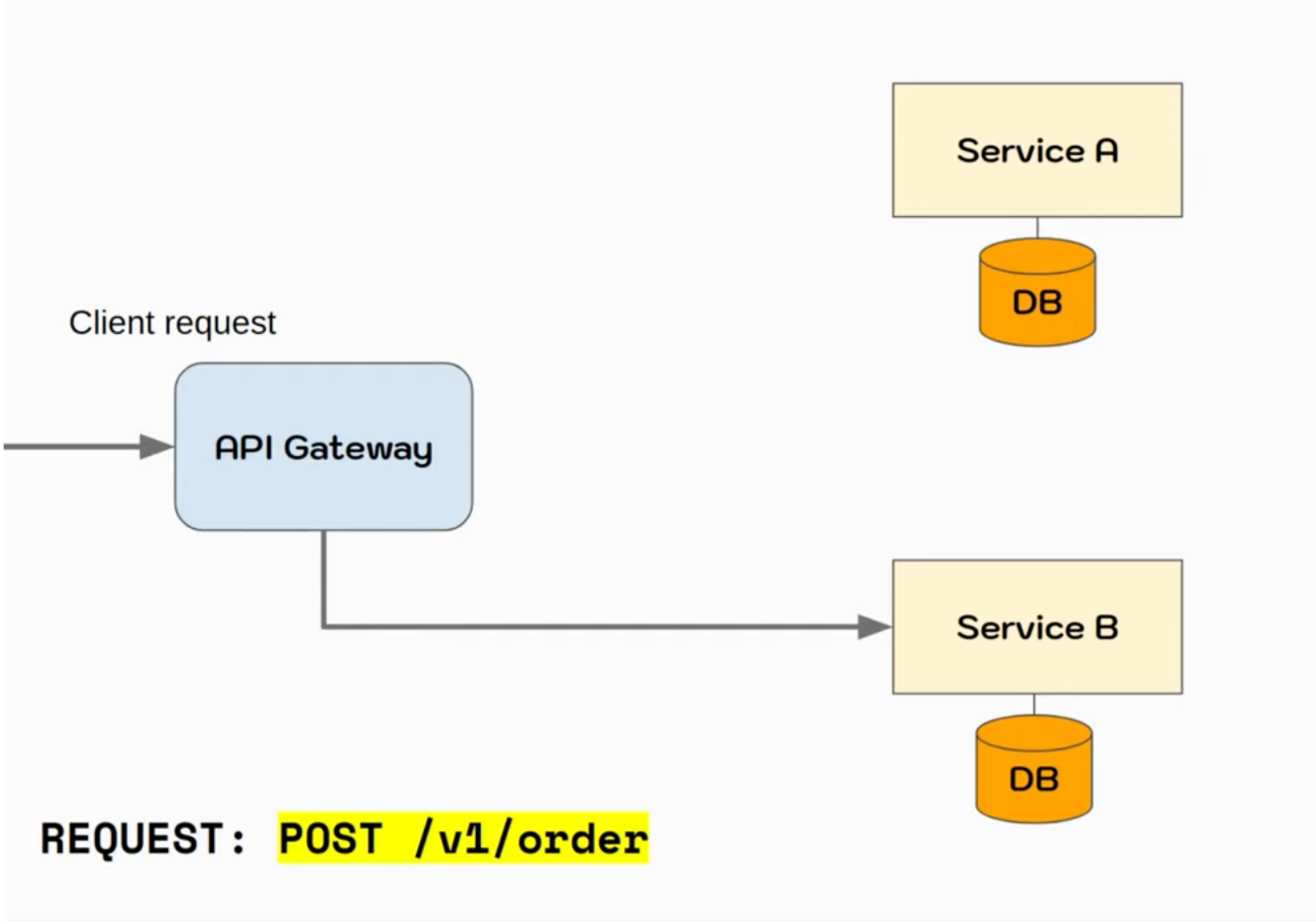
Client request

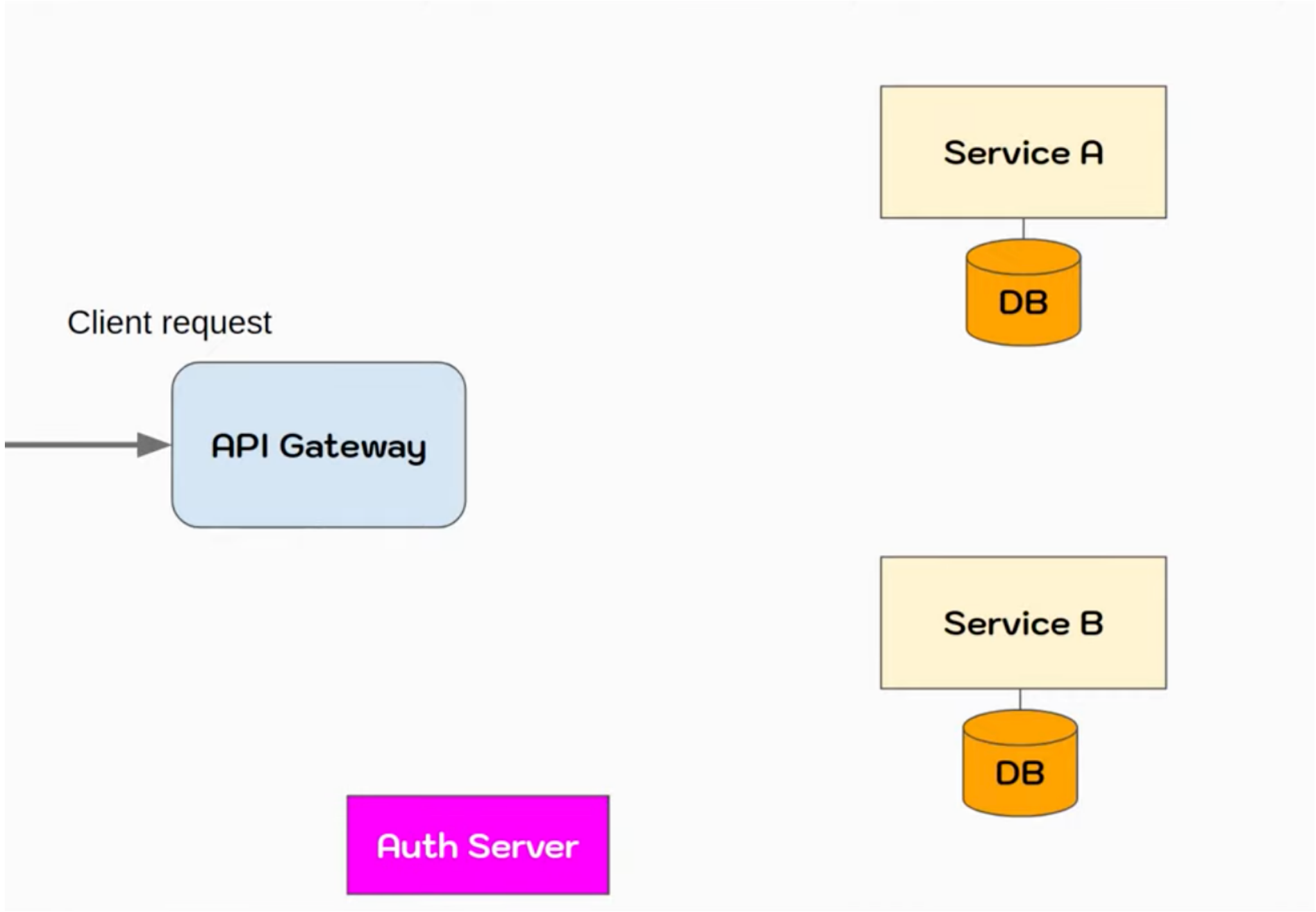
?

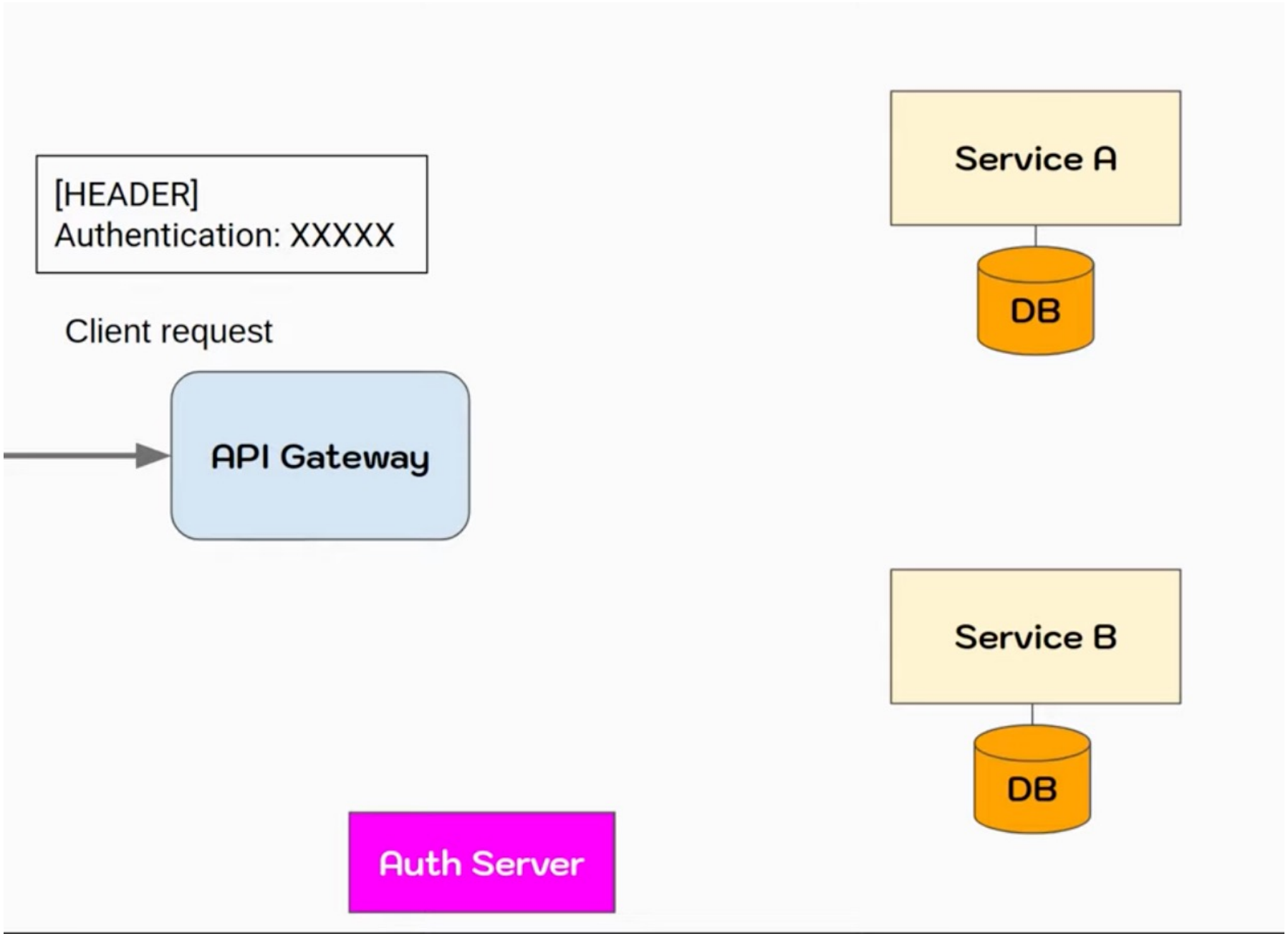
IP: yyy.yyy.yyy.yyy

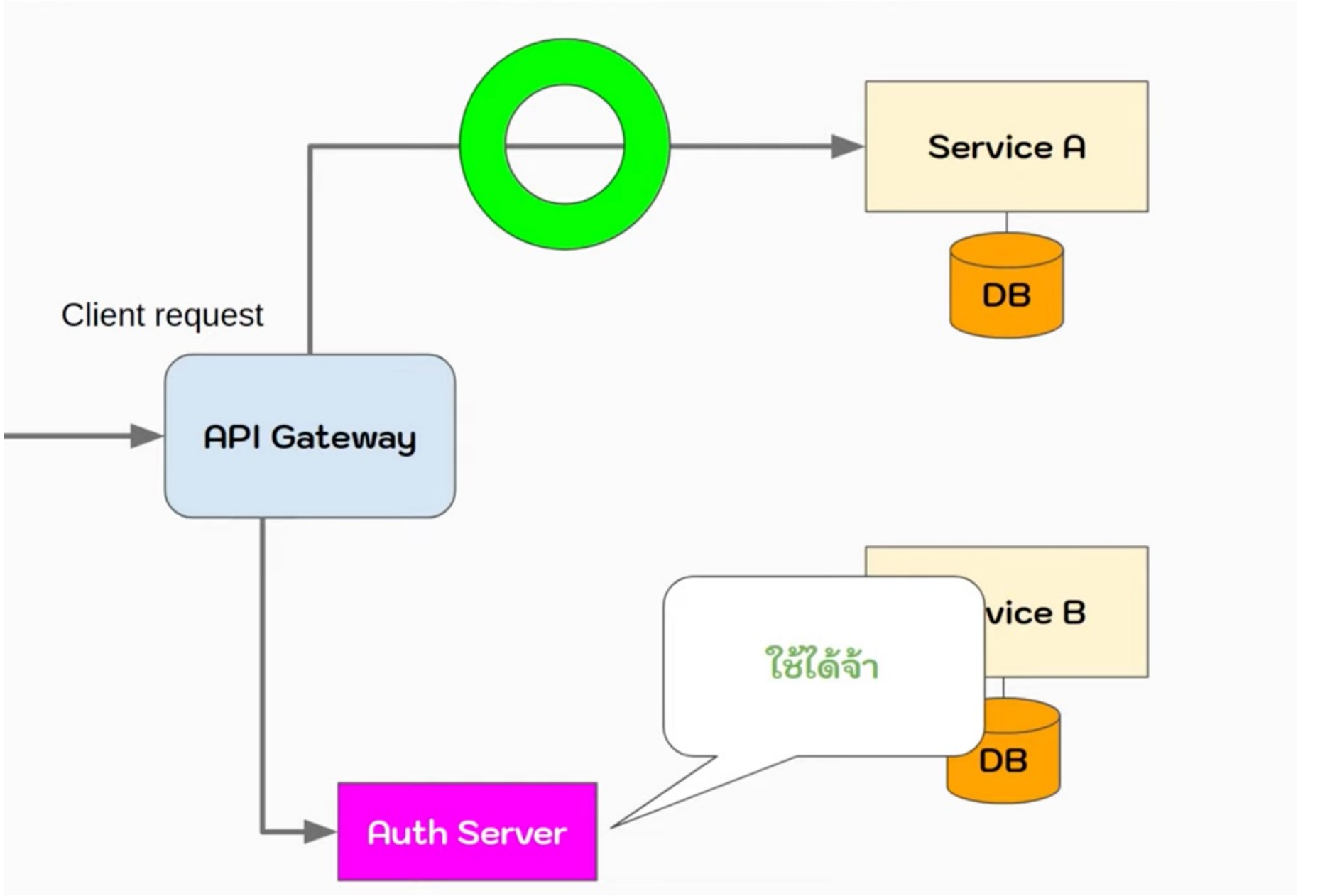


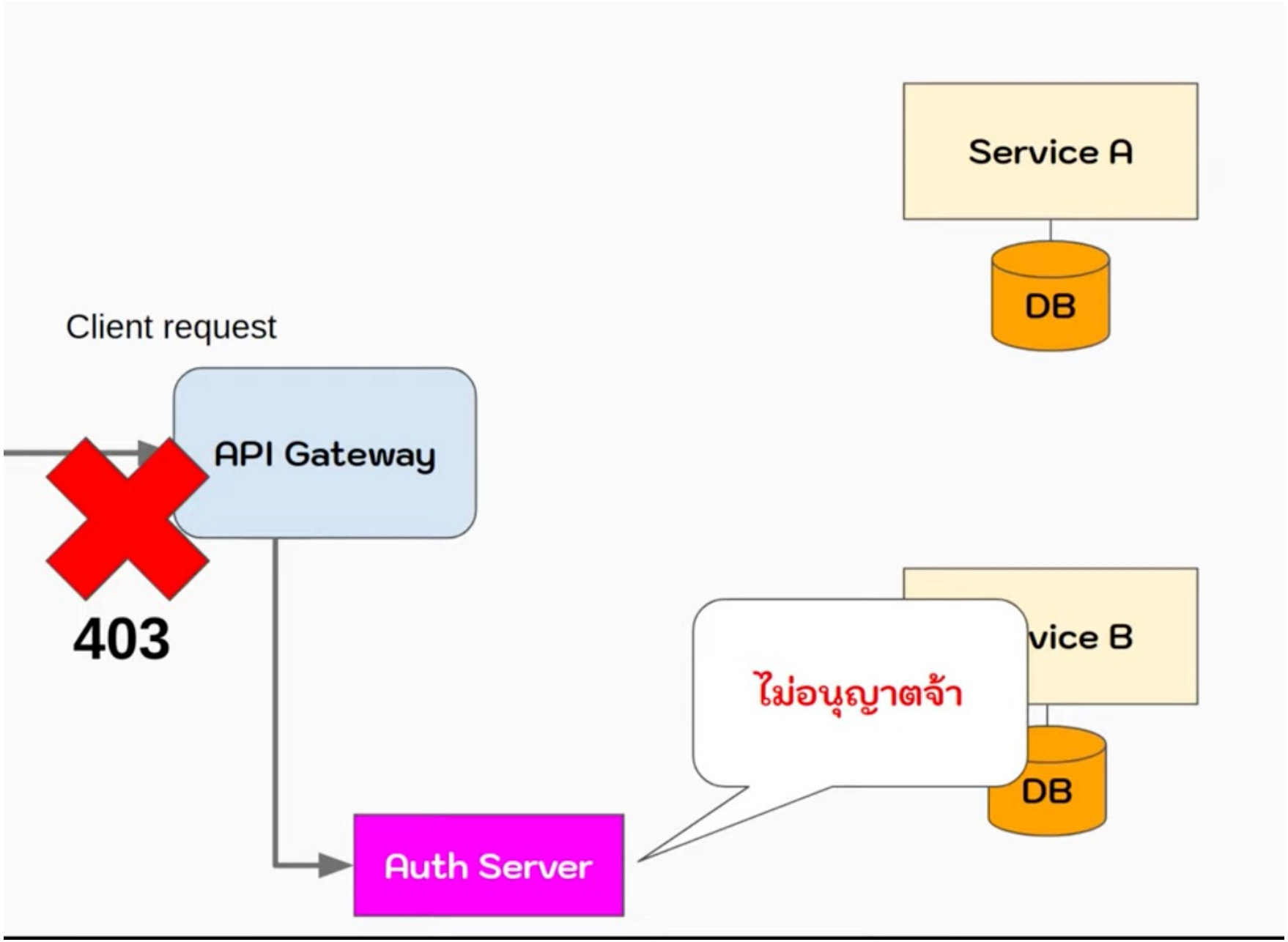

Service A
DB
Client request
API Gateway
Service B
DB
REQUEST: GET /v1/product/1234


Service A
DB
Client request
API Gateway
Service B
DB
REQUEST: POST /v1/order


Service A
DB
Client request
API Gateway
Service B
DB
Auth Server


[HEADER]
Authentication: XXXXX
Client request
API Gateway
Service A
DB
Service B
DB
Auth Server


Service A
DB
Client request
API Gateway
vice B
ใช้ได้จ้า
DB
Auth Server


Service A
DB
Client request
API Gateway
403
vice B
ไม่อนุญาตจ้า
DB
Auth Server

ตัวอย่าง API Gateway ในตลาด

เมนูวันนี้ (1/3)

Frontend talks to Microservices

- API Gateway
- Authentication

Microservices talk to Microservices

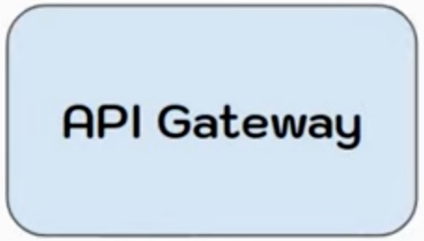
- Synchronous
- Asynchronous: Message Broker

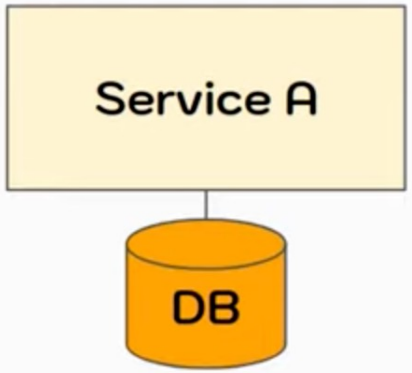
Logging and Monitoring

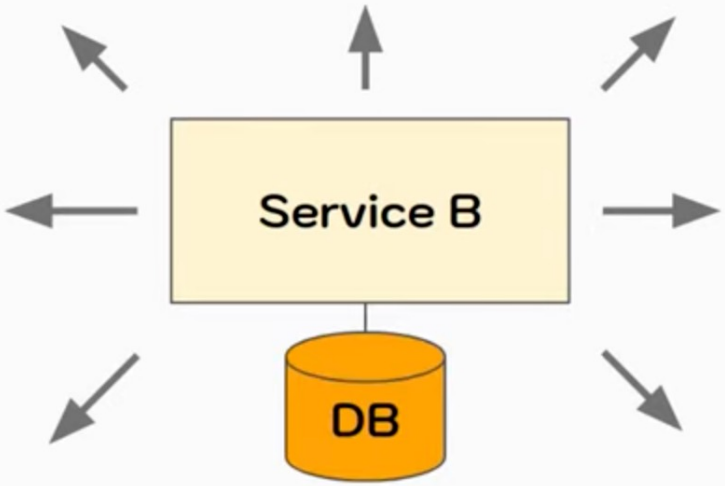
ปัญหา:

service ต้องมีการสื่อสารกัน

Client request





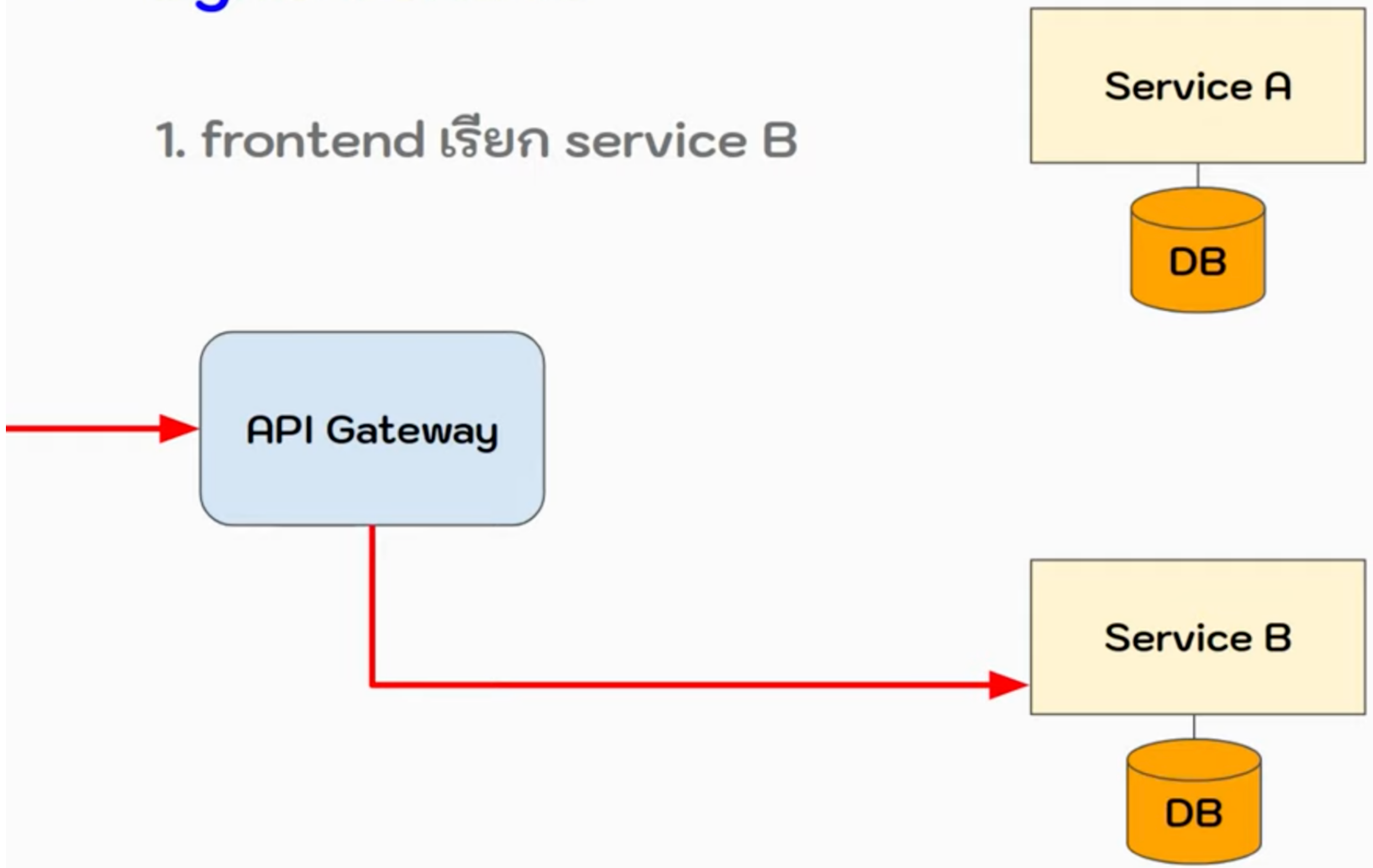


การสื่อสารระหว่าง services

- Synchronous ส่ง request ไป, รอจนเสร็จ, รับ response
- Asynchronous ไม่รอ response

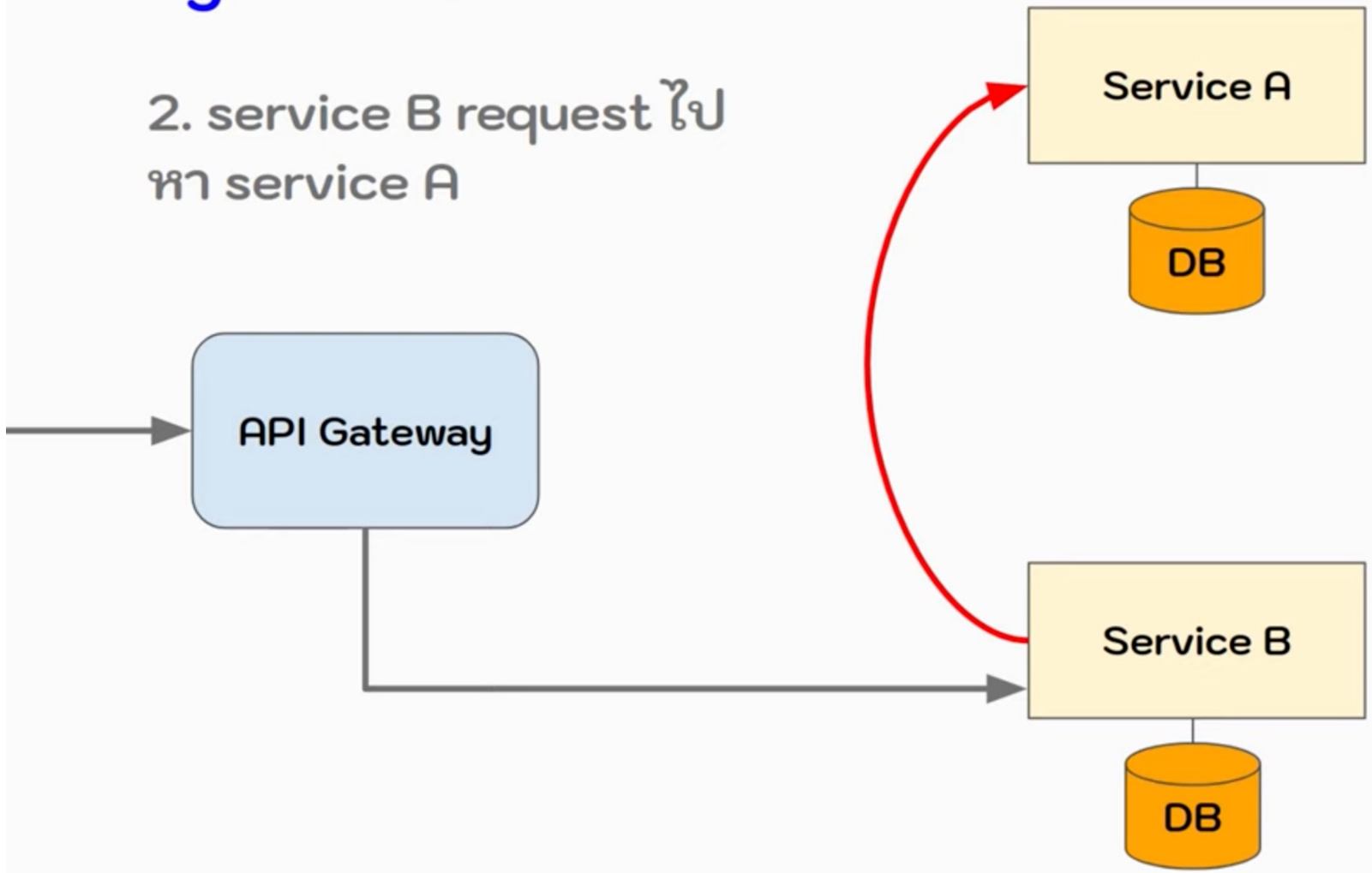
- **Synchronous**

1. frontend เรียก service B



- **Synchronous**

2. service B request ไป หา service A

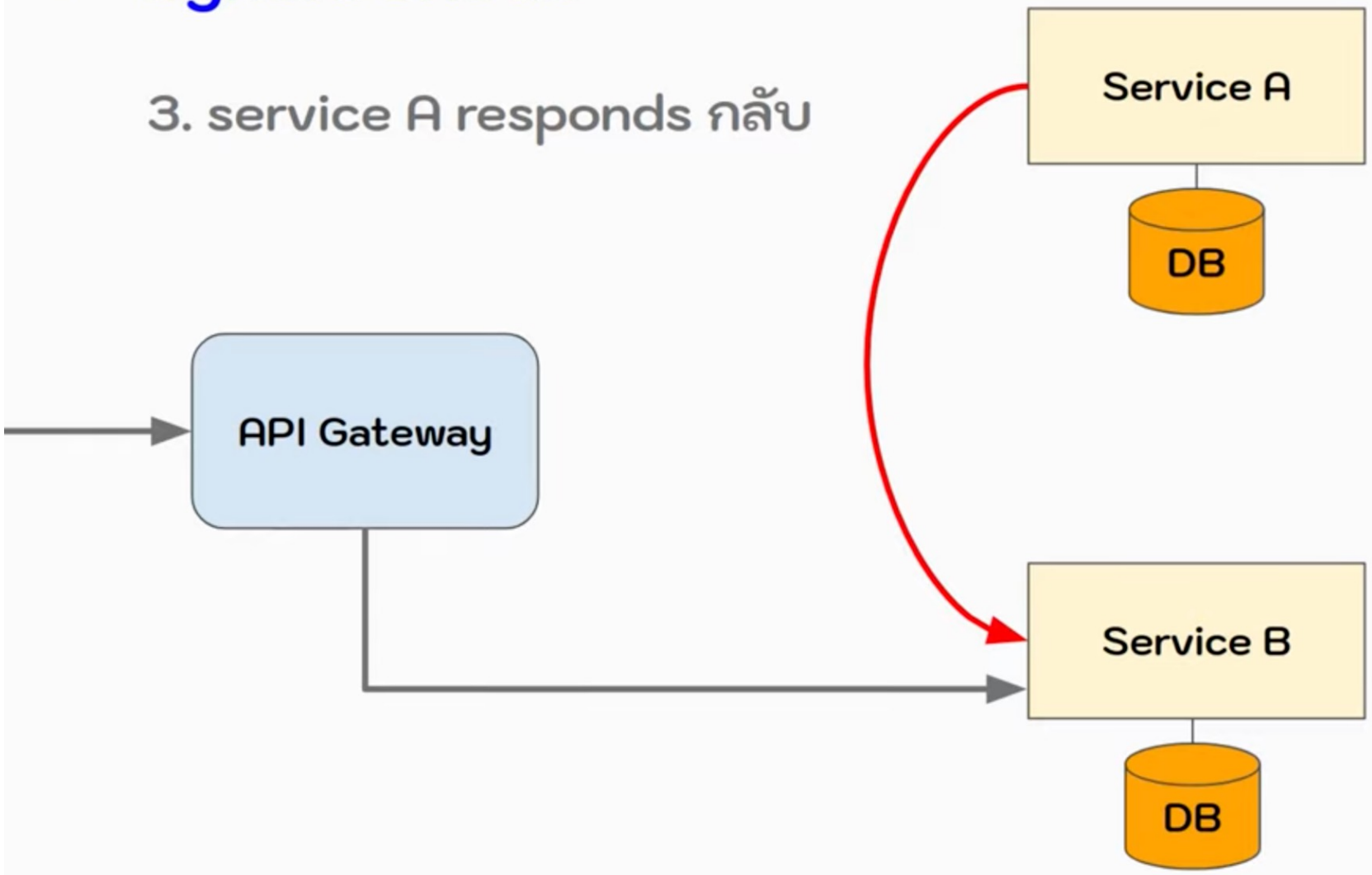


request ด้วย:

- REST API
- gRPC

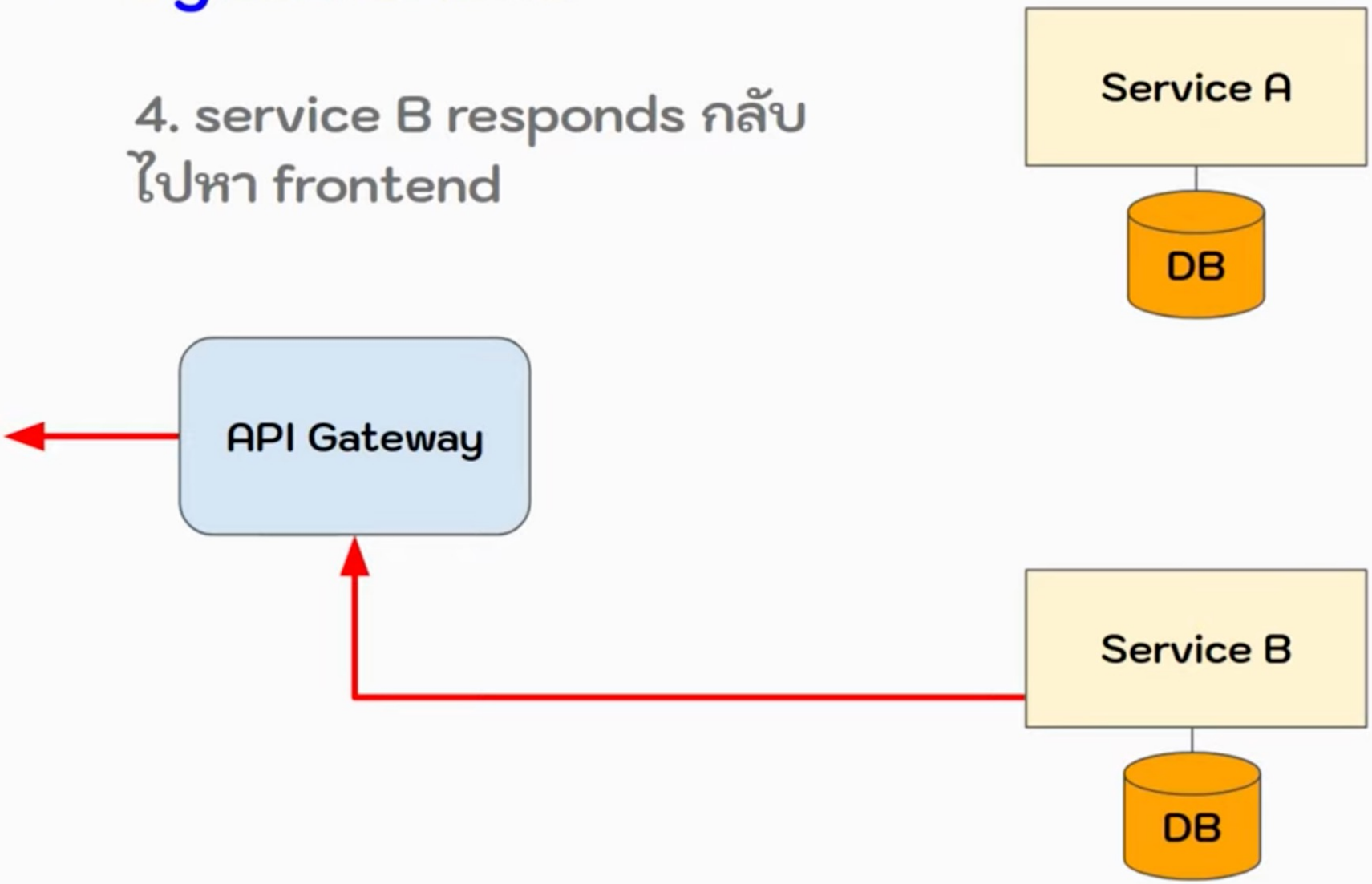
- **Synchronous**

3. service A responds กลับ



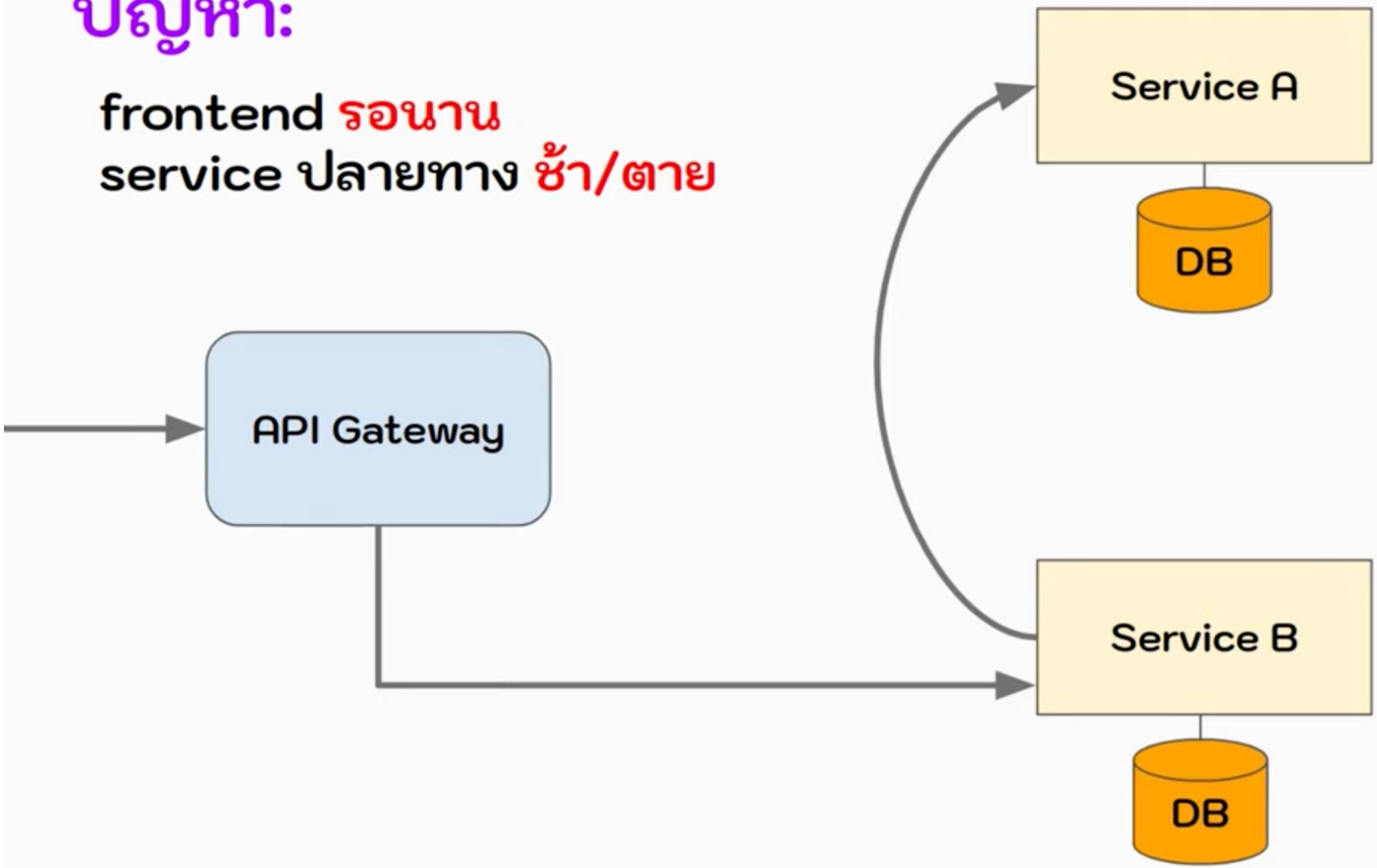
- **Synchronous**

4. service B responds กลับไปหา frontend

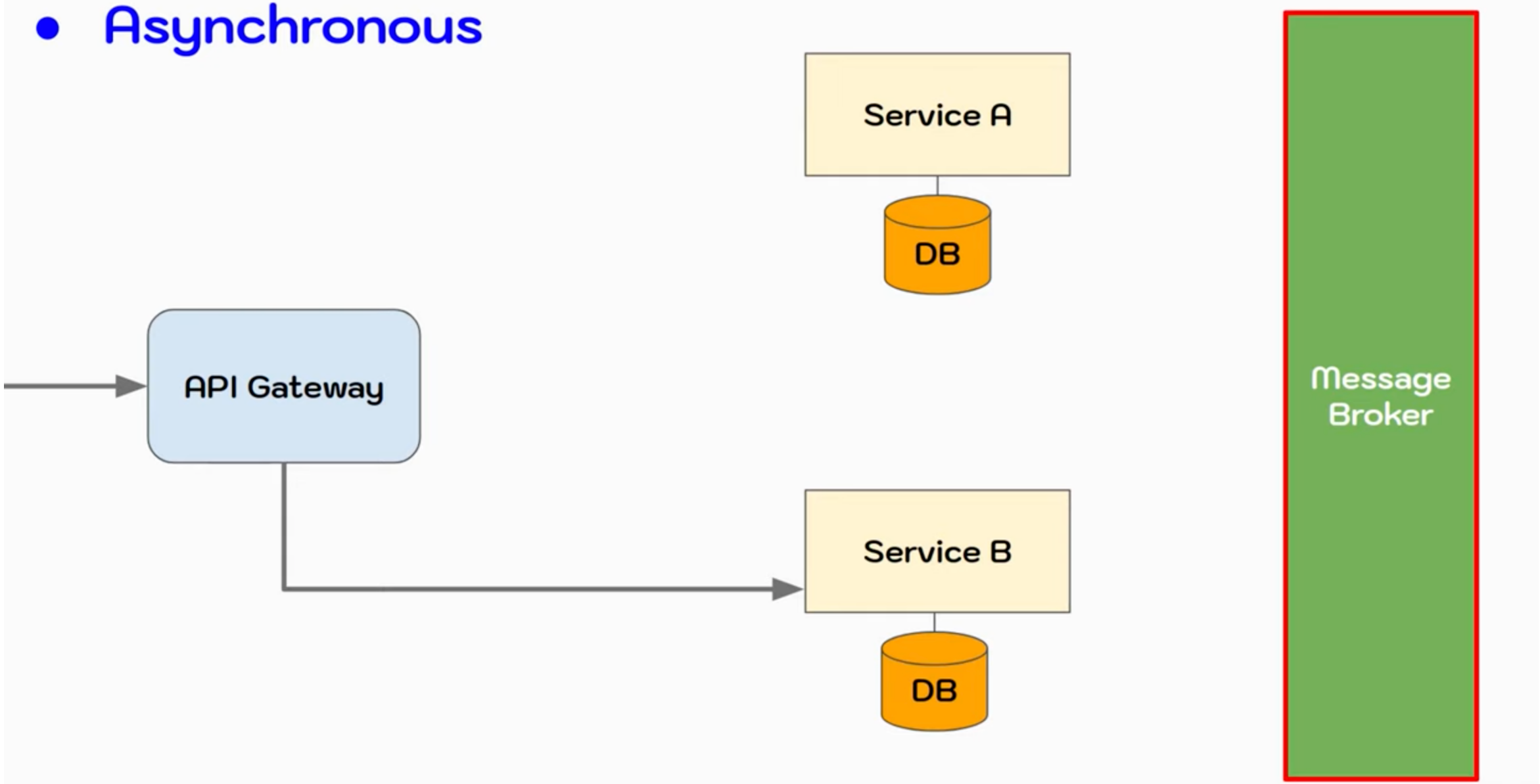


ปัญหา:

frontend รอนาน
service ปลายทาง ช้า/ตาย

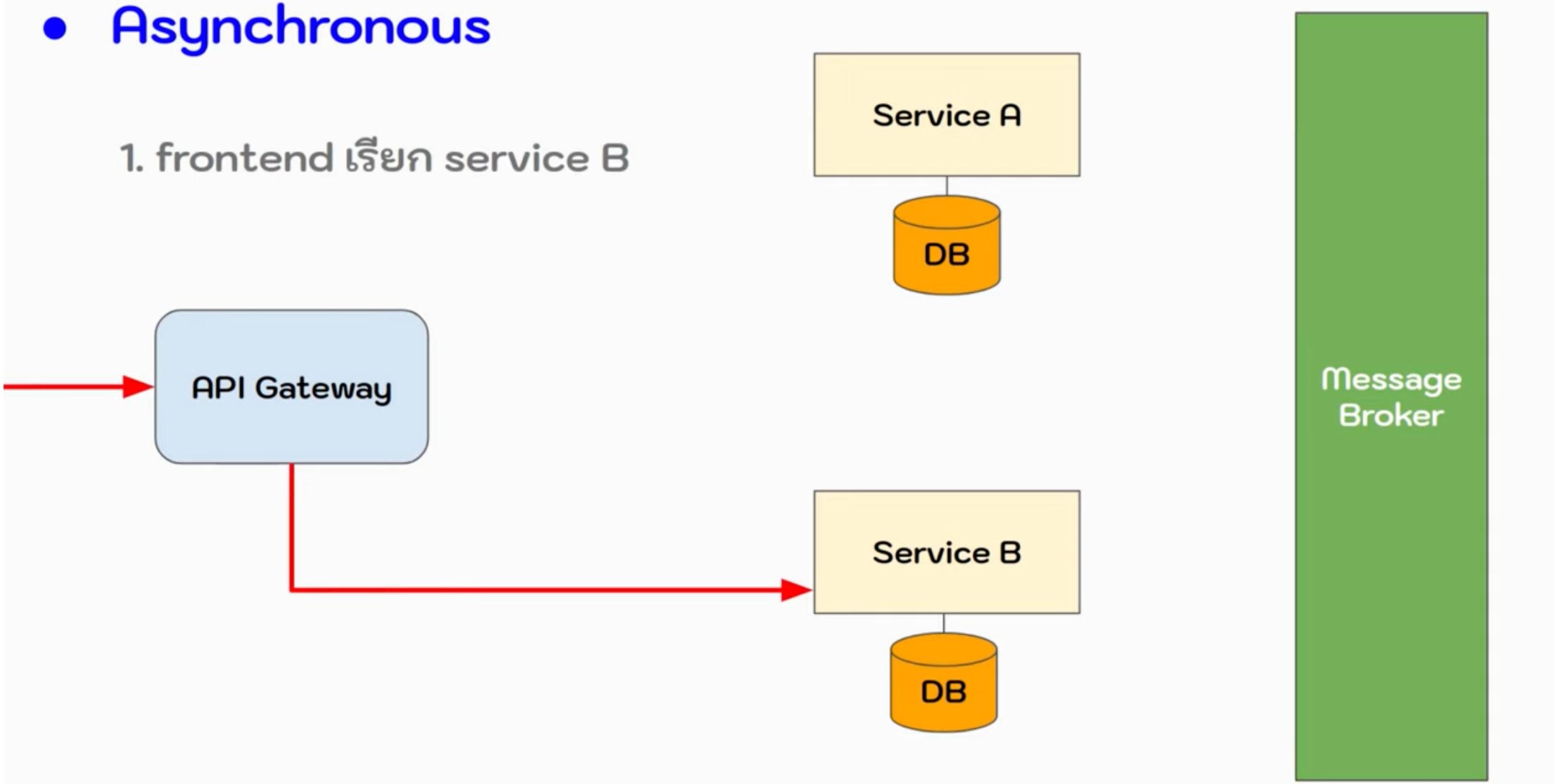


- Asynchronous



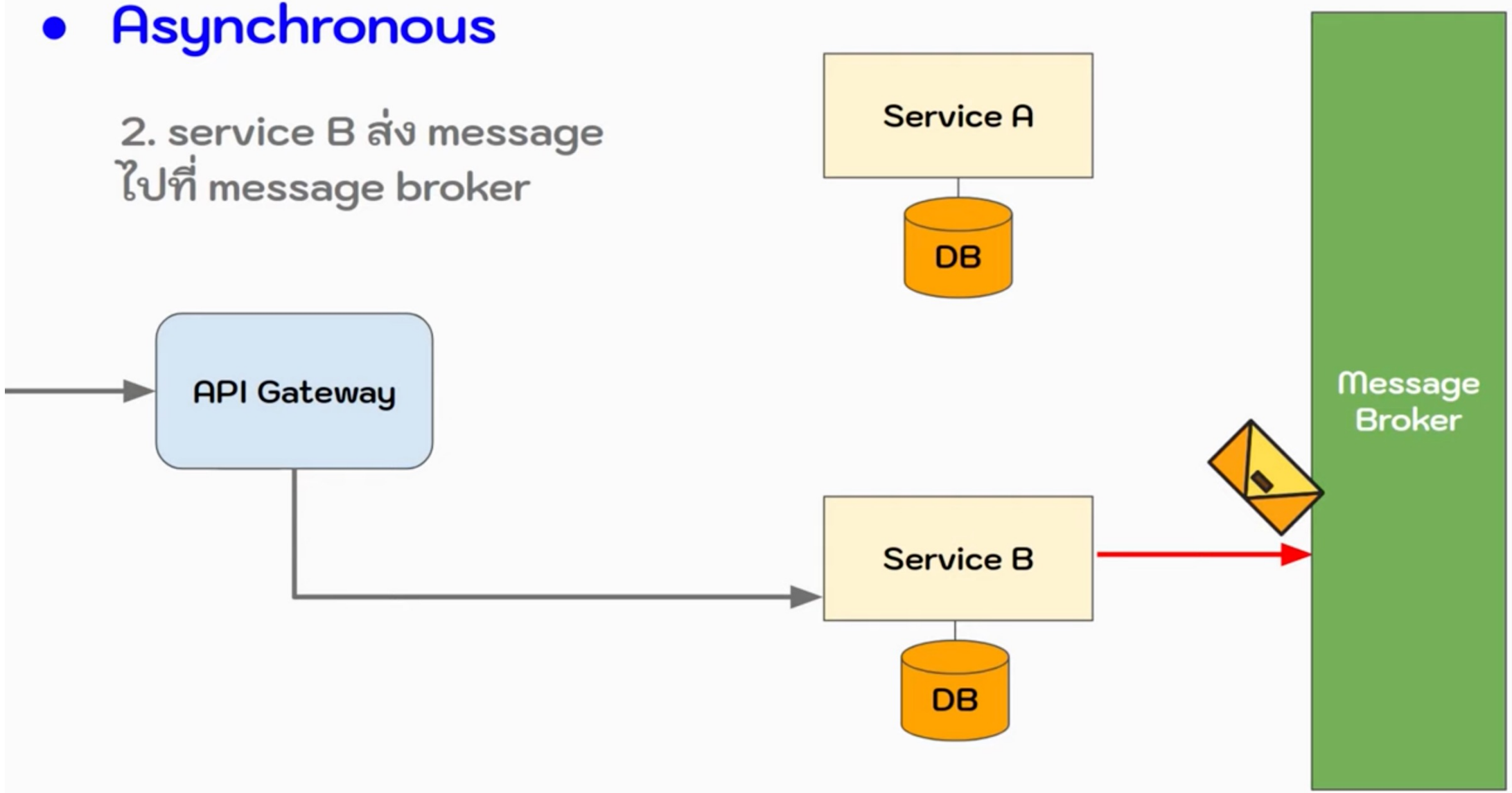
- **Asynchronous**

1. frontend เรียก service B



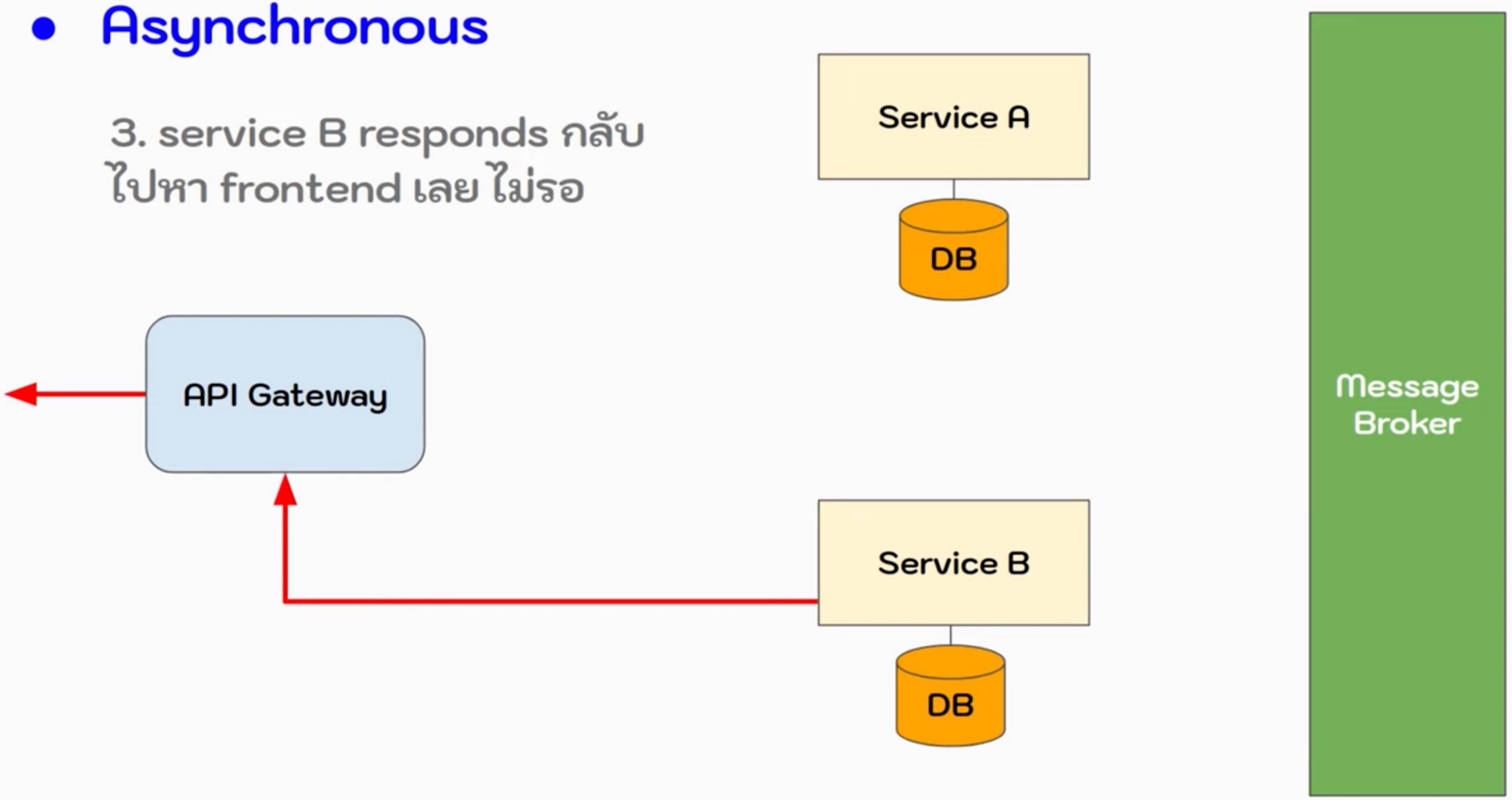
- **Asynchronous**

2. service B ส่ง message ไปที่ message broker



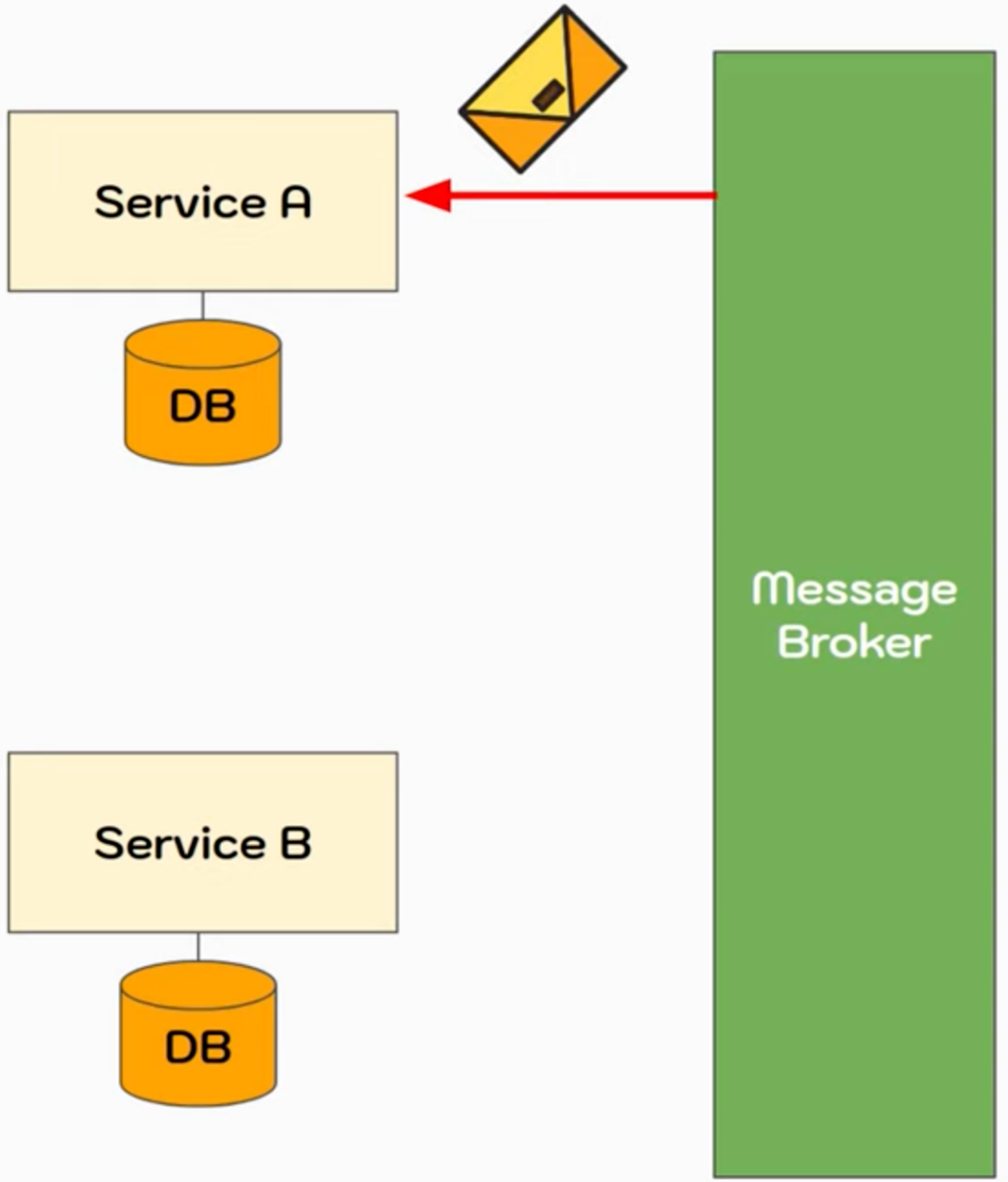
- **Asynchronous**

3. service B responds กลับไปหา frontend เลย ไม่รอ



- **Asynchronous**

4. service A ได้รับ message และเริ่มทำงาน

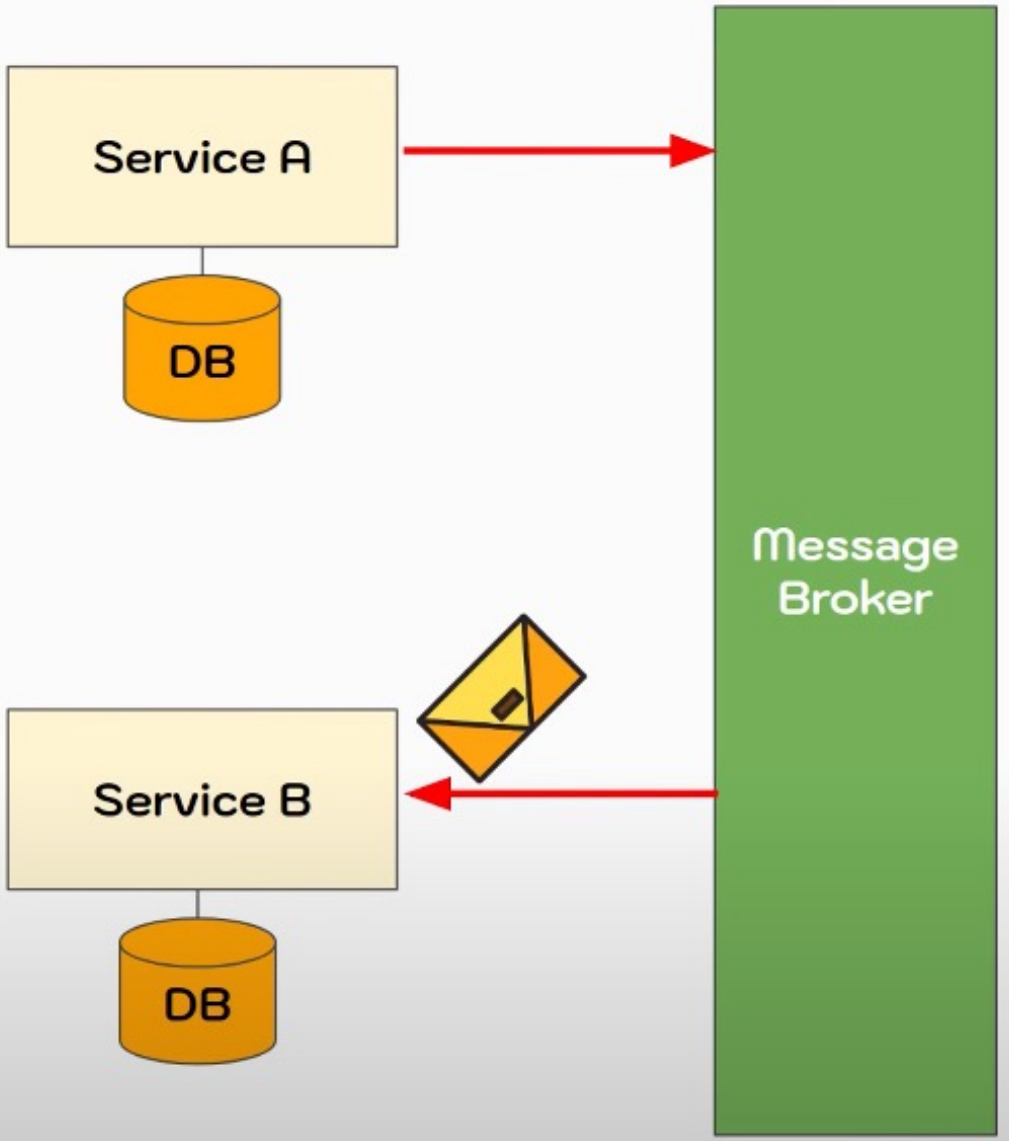


API Gateway

- **Asynchronous**

[5. service A ส่ง message response กลับไปหา B]

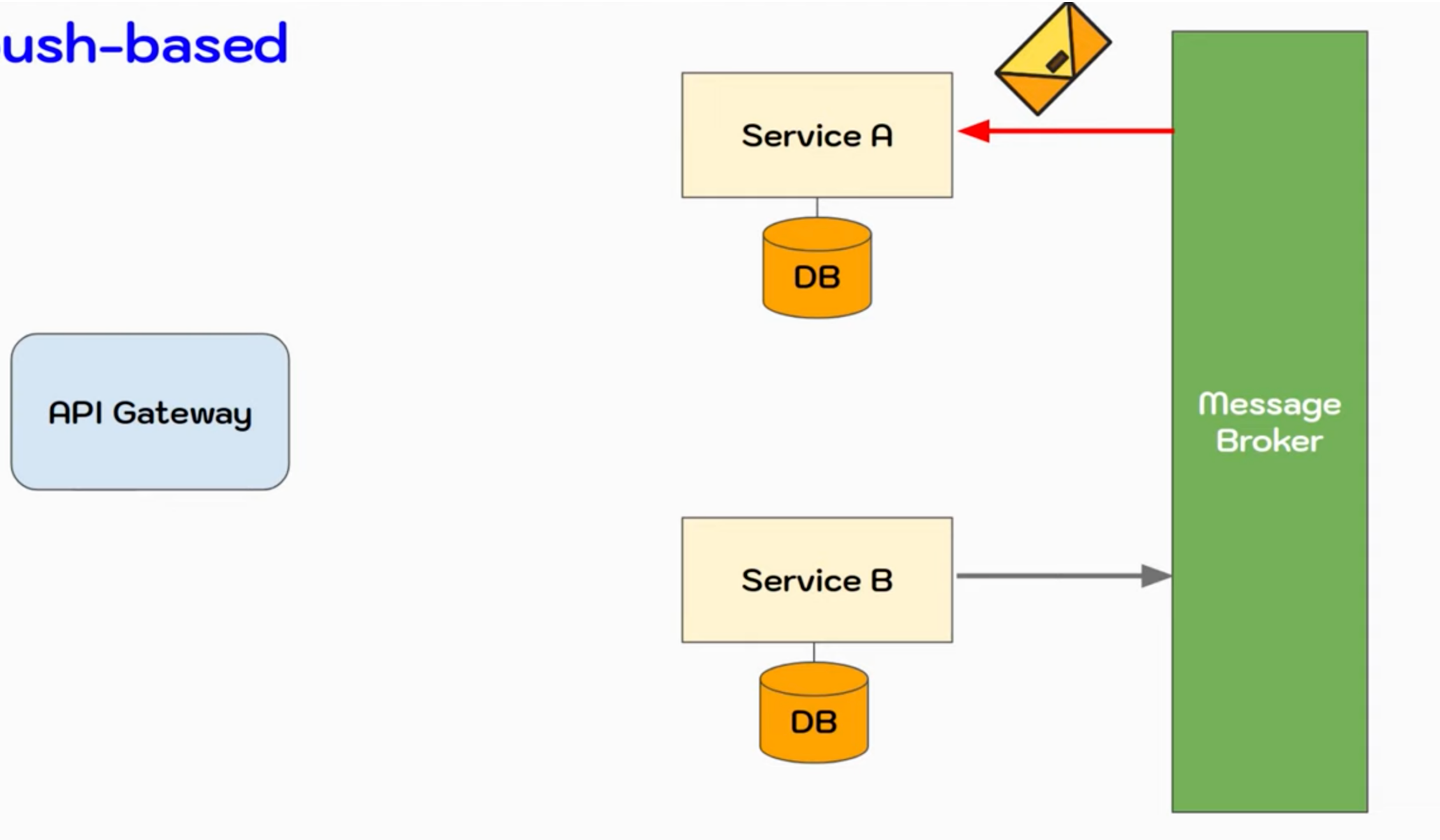
API Gateway



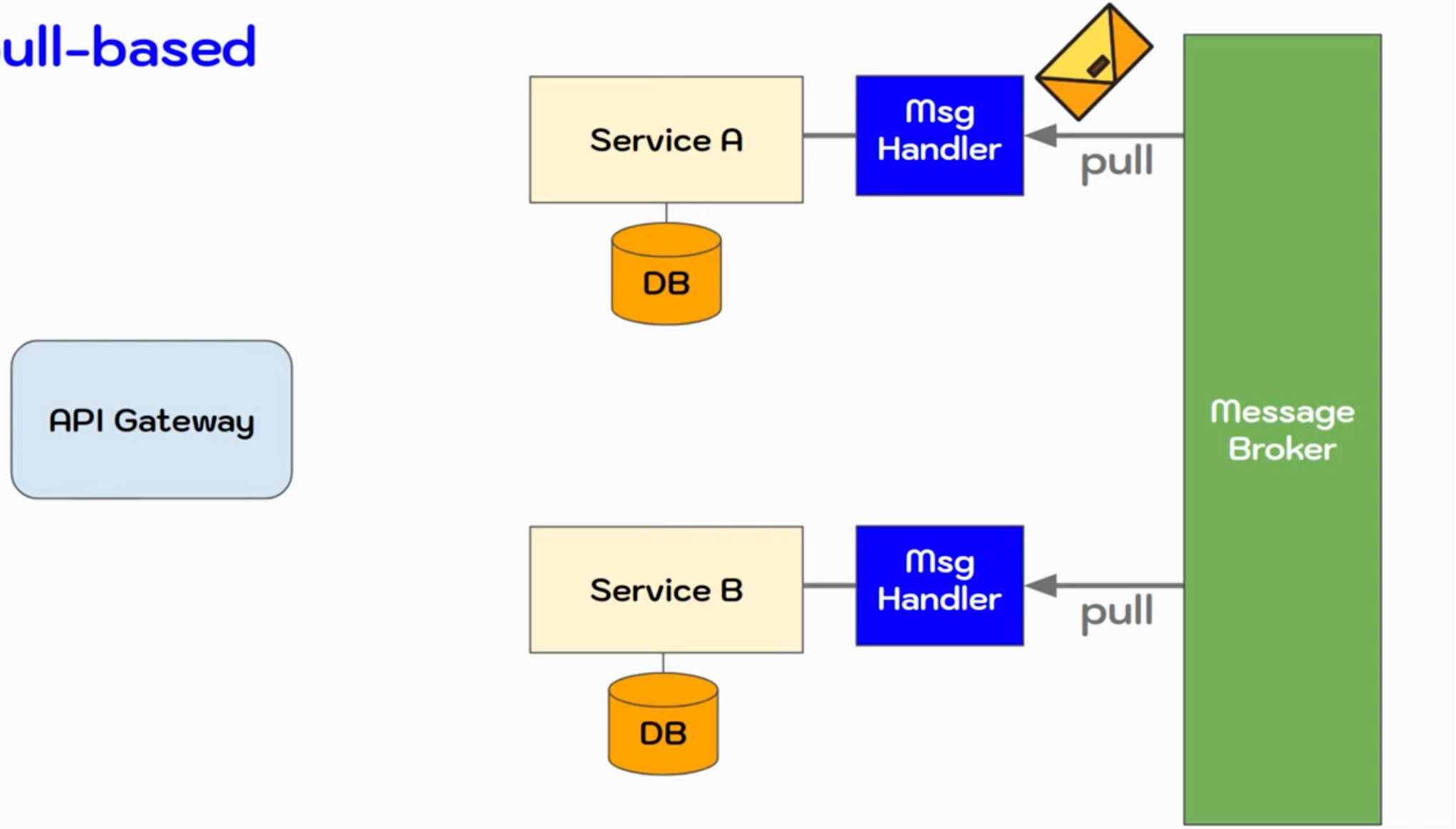
ประเภทของ message broker

- push-based message broker ยิง message ไปให้ service
- pull-based service มาเช็คเองว่ามี message หรือเปล่า

- push-based



- pull-based


Service A
Msg Handler
pull
DB
API Gateway
Message Broker
Service B
Msg Handler
pull
DB

ตัวอย่าง Message Broker ในตลาด

(push)

(pull)

Google Cloud Pub/Sub

(push&pull)

เมนูวันนี้ (1/3)

Frontend talks to Microservices

- API Gateway
- Authentication

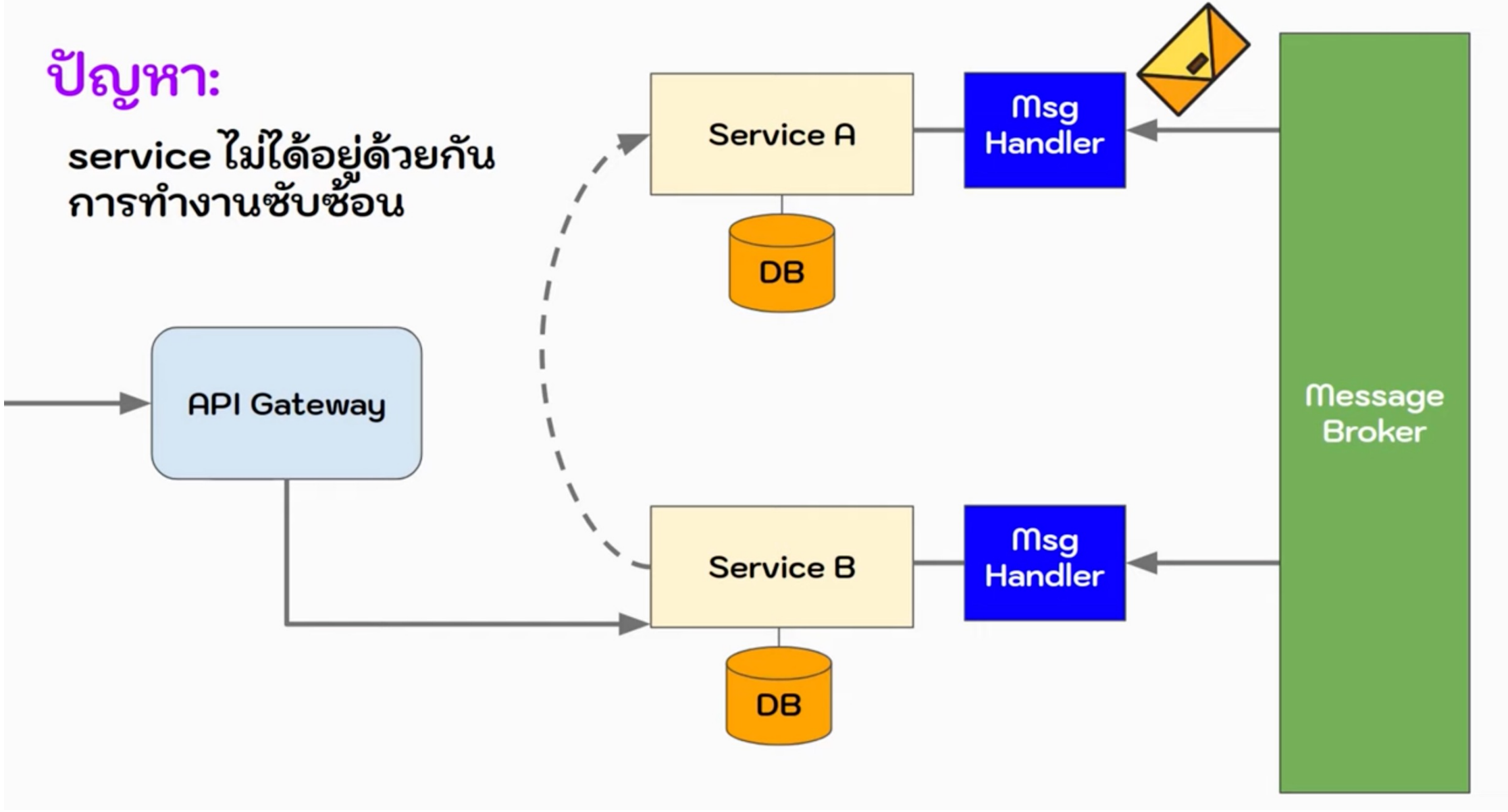
Microservices talk to Microservices

- Synchronous
- Asynchronous: Message Broker

Logging and Monitoring

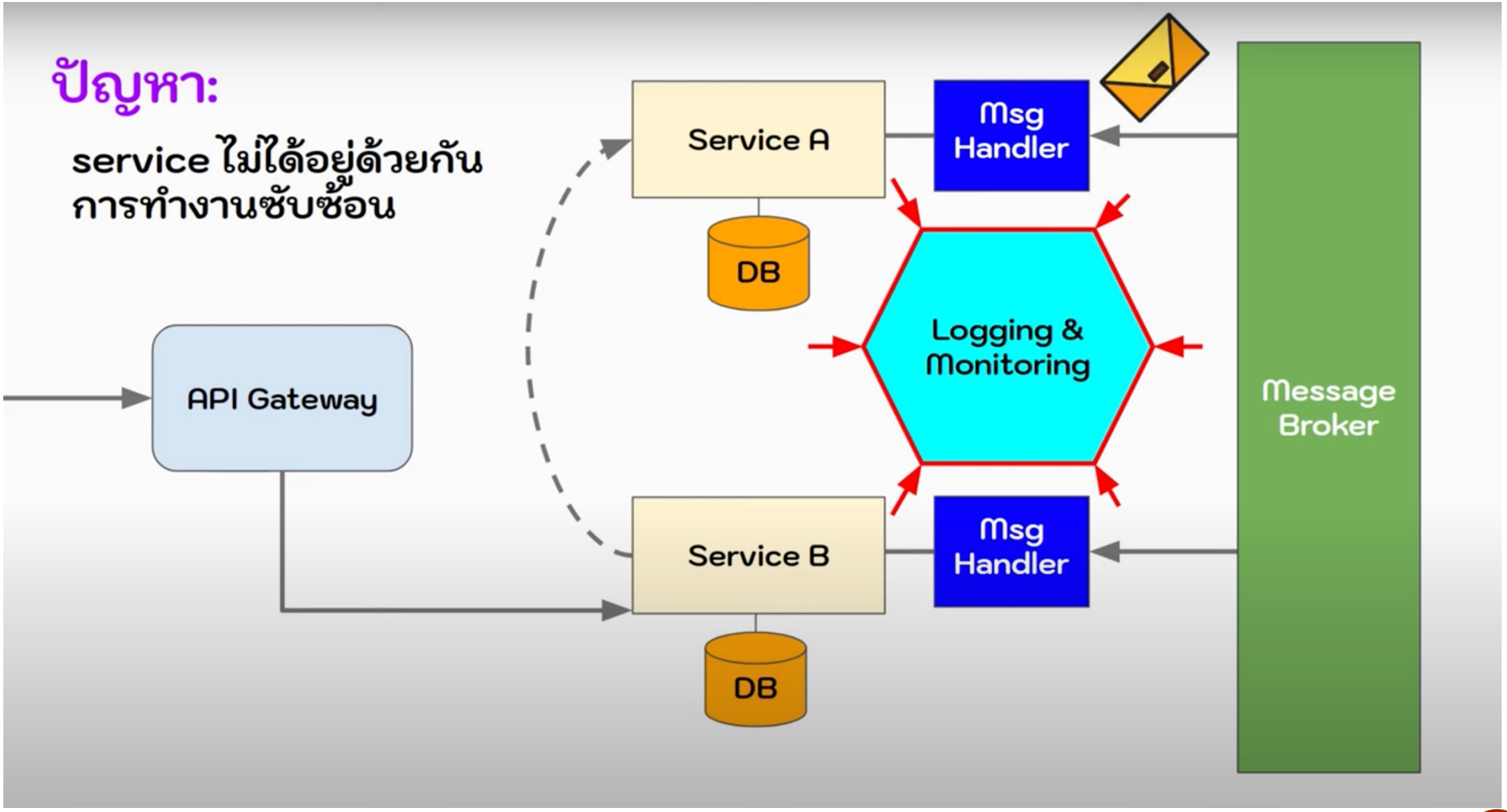
ปัญหา:

service ไม่ได้อยู่ด้วยกัน
การทำงานซับซ้อน



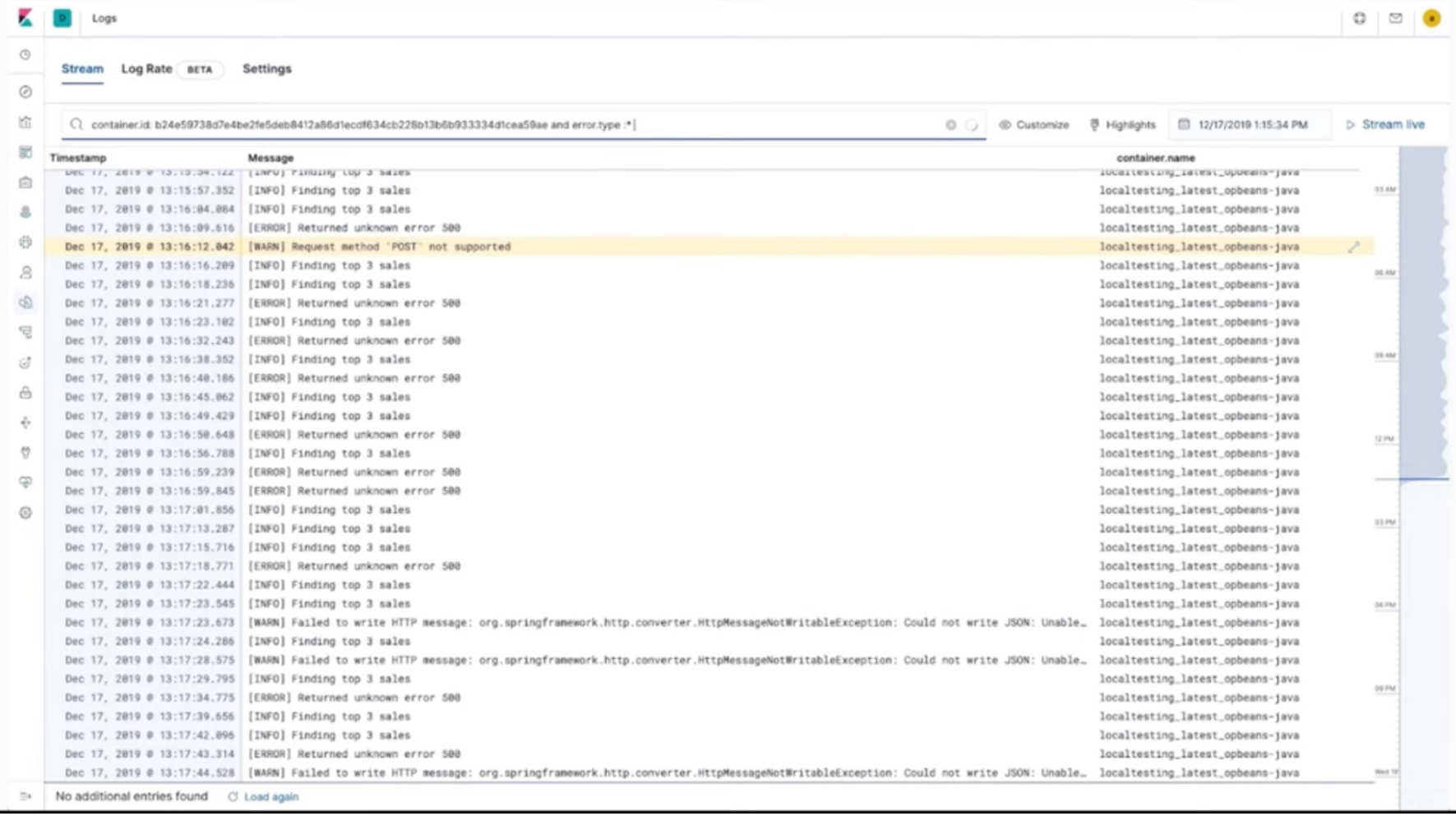
ปัญหา:

service ไม่ได้อยู่ด้วยกัน
การทำงานซับซ้อน

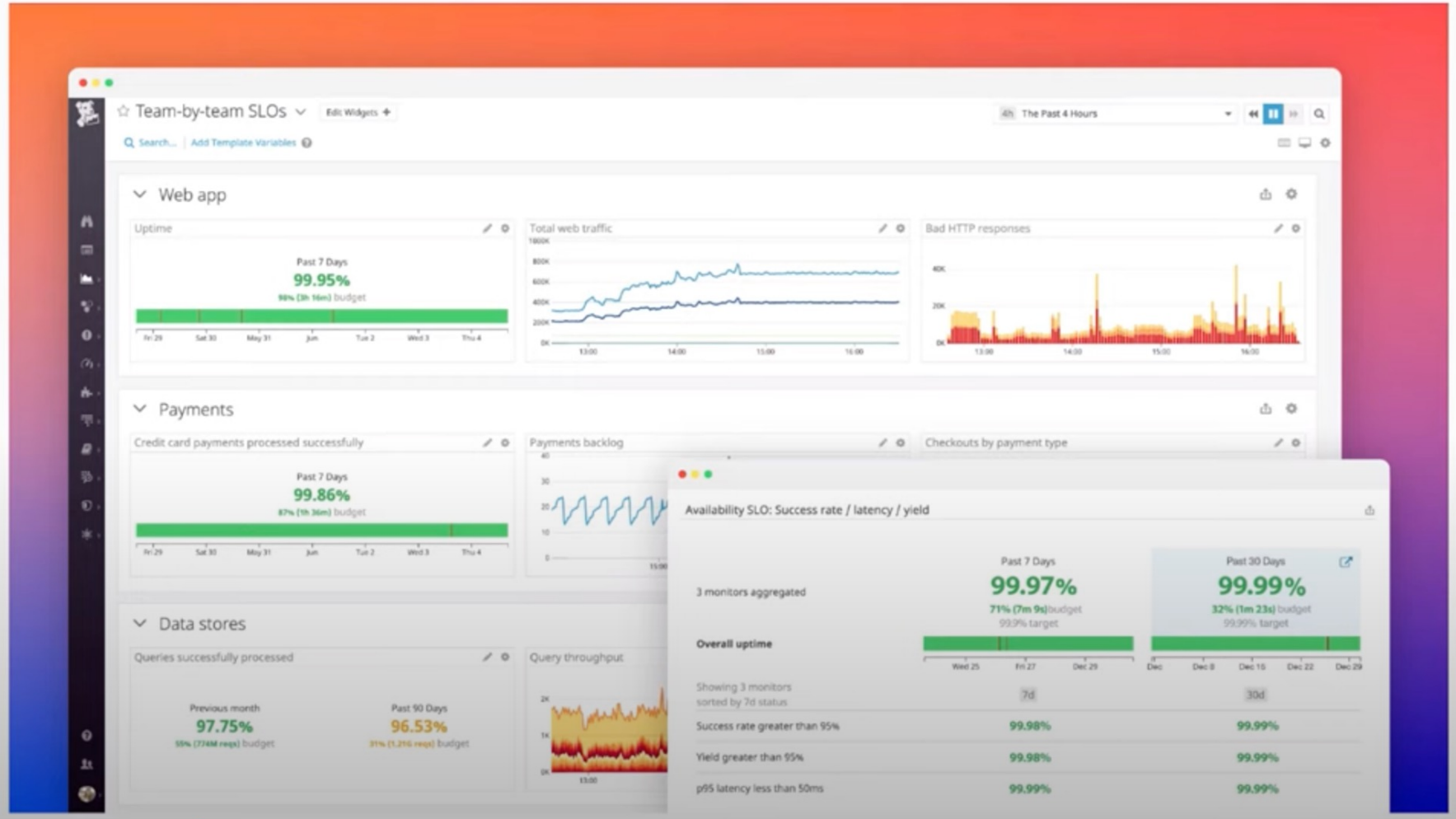


Infrastructure Monitoring

Logging


Logs
Stream
Log Rate BETA
Settings
container.id: b24e59738d7e4be2fe5deb8412a86d1ecdf634cb226b13b6b933334d1cea59ae and error.type :*
Customize
Highlights
12/17/2019 1:15:34 PM
Stream live
Timestamp
Message
container.name
Dec 17, 2019 @ 13:15:57.352 [INFO] Finding top 3 sales localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:16:04.084 [INFO] Finding top 3 sales localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:16:09.616 [ERROR] Returned unknown error 500 localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:16:12.042 [WARN] Request method 'POST' not supported localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:16:16.209 [INFO] Finding top 3 sales localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:16:18.236 [INFO] Finding top 3 sales localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:16:21.277 [ERROR] Returned unknown error 500 localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:16:23.102 [INFO] Finding top 3 sales localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:16:32.243 [ERROR] Returned unknown error 500 localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:16:38.352 [INFO] Finding top 3 sales localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:16:40.186 [ERROR] Returned unknown error 500 localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:16:45.862 [INFO] Finding top 3 sales localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:16:49.429 [INFO] Finding top 3 sales localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:16:50.648 [ERROR] Returned unknown error 500 localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:16:56.788 [INFO] Finding top 3 sales localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:16:59.239 [ERROR] Returned unknown error 500 localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:16:59.845 [ERROR] Returned unknown error 500 localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:17:01.856 [INFO] Finding top 3 sales localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:17:13.287 [INFO] Finding top 3 sales localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:17:15.716 [INFO] Finding top 3 sales localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:17:18.771 [ERROR] Returned unknown error 500 localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:17:22.444 [INFO] Finding top 3 sales localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:17:23.545 [INFO] Finding top 3 sales localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:17:23.673 [WARN] Failed to write HTTP message: org.springframework.http.converter.HttpMessageNotWritableException: Could not write JSON: Unable… localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:17:24.286 [INFO] Finding top 3 sales localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:17:28.575 [WARN] Failed to write HTTP message: org.springframework.http.converter.HttpMessageNotWritableException: Could not write JSON: Unable… localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:17:29.795 [INFO] Finding top 3 sales localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:17:34.775 [ERROR] Returned unknown error 500 localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:17:39.656 [INFO] Finding top 3 sales localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:17:42.096 [INFO] Finding top 3 sales localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:17:43.314 [ERROR] Returned unknown error 500 localtesting_latest_opbeans-java
Dec 17, 2019 @ 13:17:44.528 [WARN] Failed to write HTTP message: org.springframework.http.converter.HttpMessageNotWritableException: Could not write JSON: Unable… localtesting_latest_opbeans-java
No additional entries found
Load again

Service-wise metrics

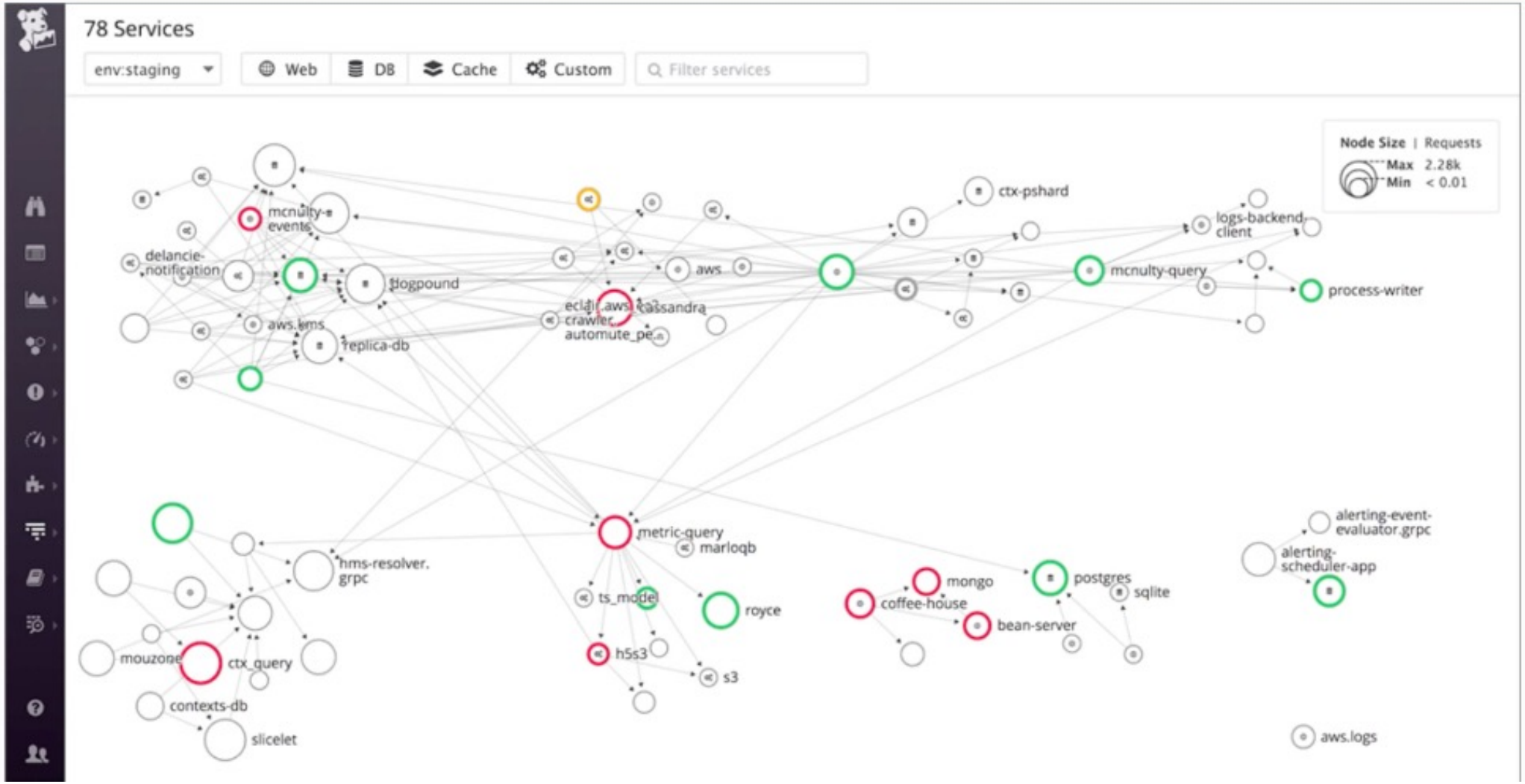


Error Notification

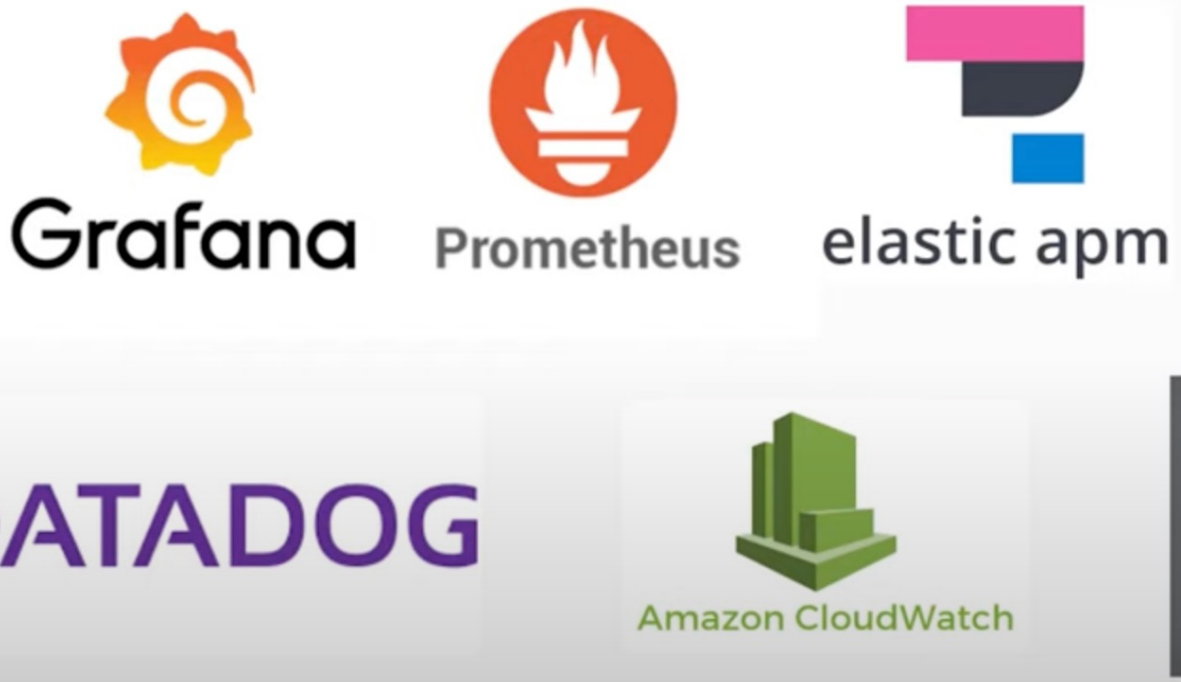
slack

Jira

Service Map


78 Services
env:staging
Web
DB
Cache
Custom
Filter services
Node Size | Requests
Max 2.28k
Min < 0.01
mcnulty-events
delancie-notification
logpound
aws.kms
replica-db
aws
cassandra
crawler
automute_pe
ctx-pshard
logs-backend-client
mcnulty-query
process-writer
hms-resolver.grpc
mouzone
ctx_query
contexts-db
slicelet
metric-query
marloqb
ts_model
royce
h5s3
s3
mongo
coffee-house
postgres
sqlite
bean-server
alerting-event-evaluator.grpc
alerting-scheduler-app
aws.logs

ตัวอย่าง Logging & Monitoring tools ในตลาด



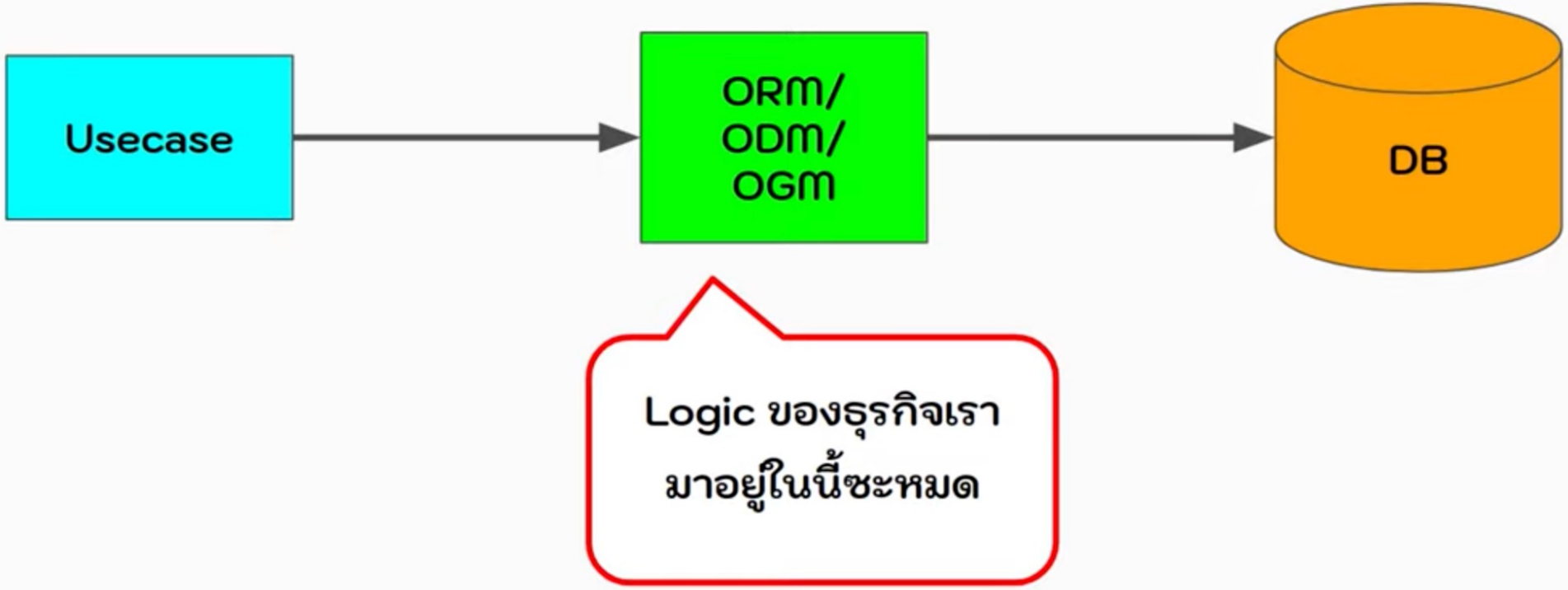
เมนูวันนี้ (2/3)

Data Access/Manipulation

- Repository Pattern
- Aggregate Pattern
- CQRS

Event-based Communication

- Event Sourcing
- Transactional Outbox


Usecase
ORM/
ODM/
OGM
DB
Logic ของธุรกิจเรา
มาอยู่ในนี้ซะหมด

Python

```python
orders = all_orders.from_buyer(1234)

for order in orders:
    item = order.items.from_name("ABC")
    if item.amount > 0:
        item.amount = 1

    all_orders.save(order)
```

SQL

```sql
UPDATE Sales.Order P
SET P.amount = 1

FROM Sales.OrderItem I

WHERE P.id = I.orderId
AND P.buyerId = 1234
AND I.name = 'ABC'
```

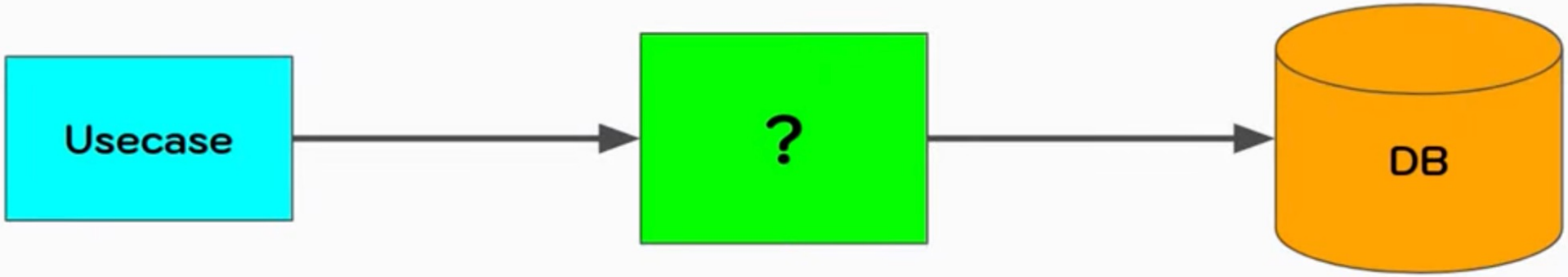
SQLAlchemy

```
stmt = (
  update(Order).
  values(amount=1).
  where(Order.c.id == OrderItem.c.order_id).
  where(Order.c.buyer_id == 1234).
  where(OrderItem.c.order_id == “ABC”)
)
session.execute(stmt)
```

*อันนี้ถ้าเขียนผิด ขออภัยนะครับ

ปัญหา:

implement logic บนภาษา programming ยังไง
ทำ operation ต่าง ๆ ให้เป็น transaction ยังไง



```
all_orders = dict()

all_orders[1234] = new_order

...

order = all_orders.pop(1234)

order.status = "PAID"

all_orders[1234] = order
```

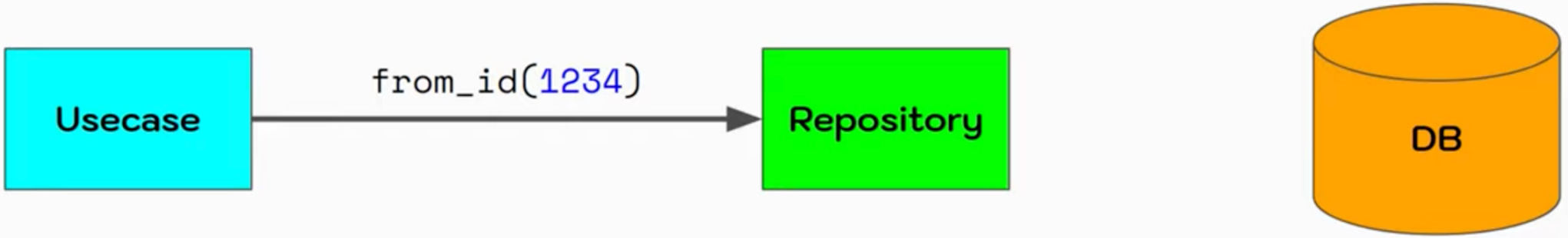
เพิ่ม object

เอา object ออกมาใช้

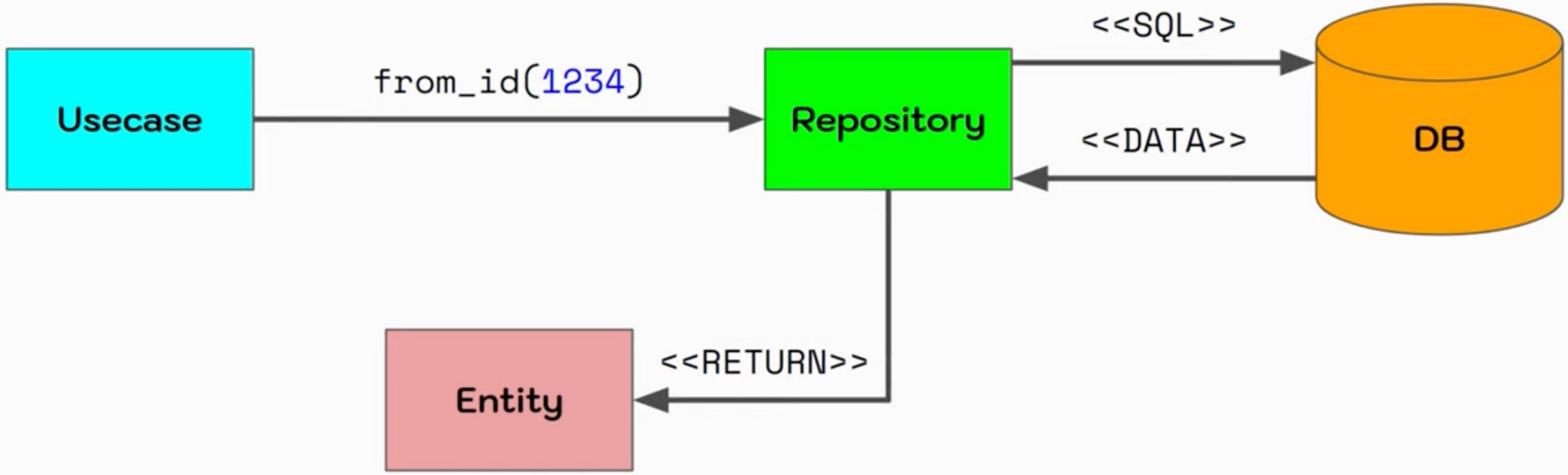
แก้ไข object

save object กลับเข้าไป

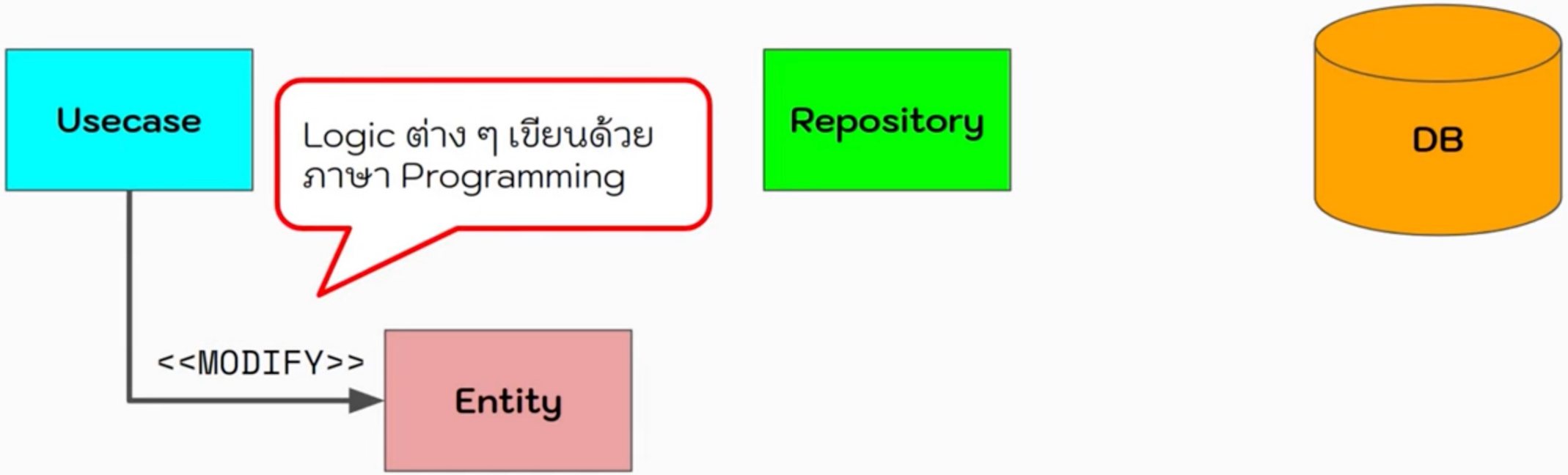
- Repository Pattern



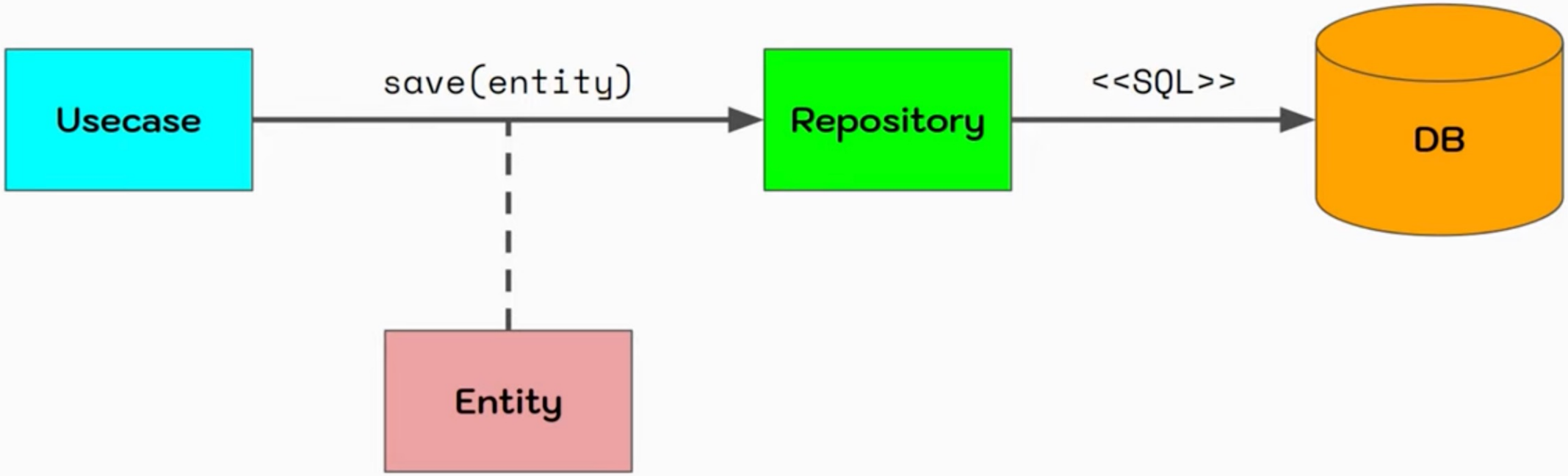
- Repository Pattern

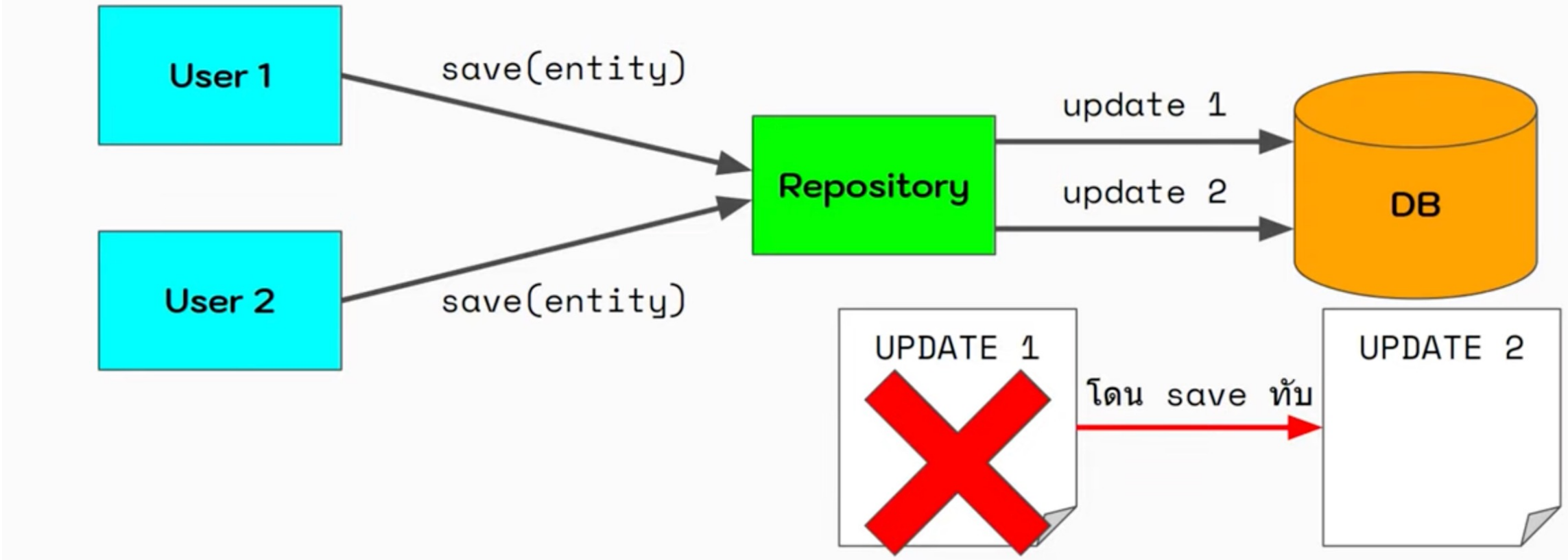


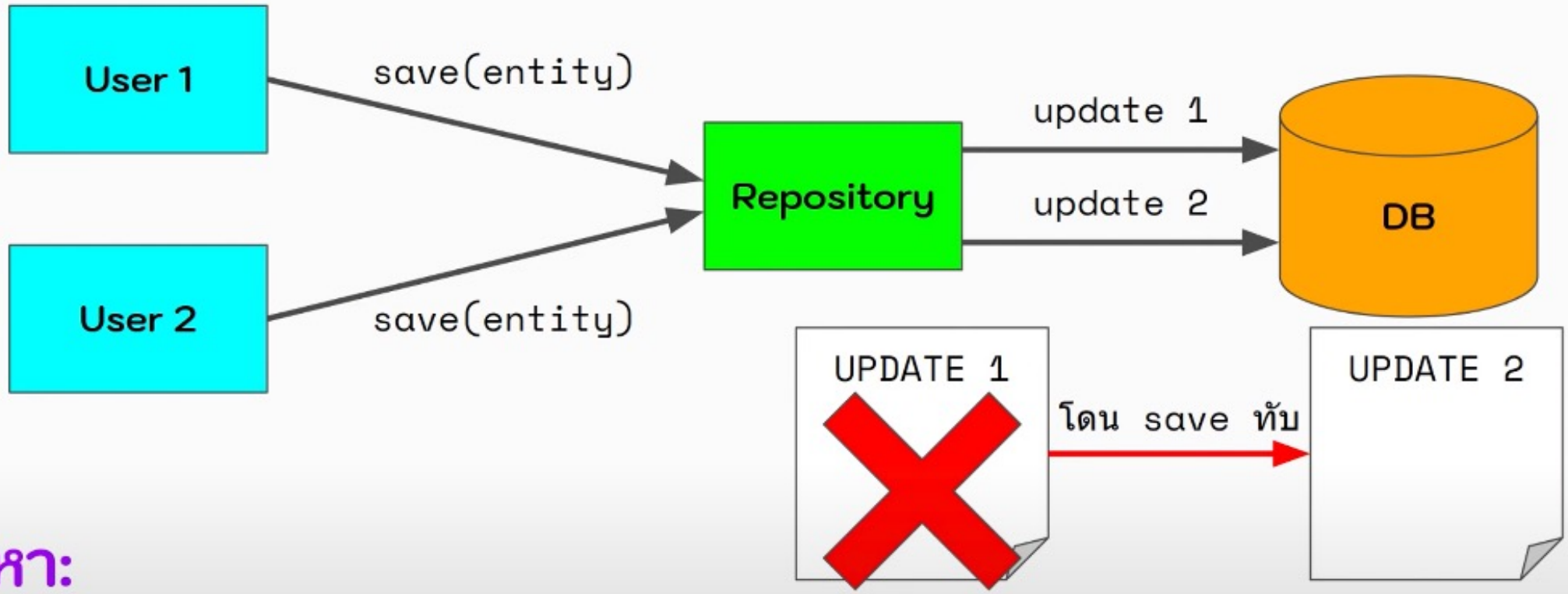
• Repository Pattern



- **Repository Pattern**



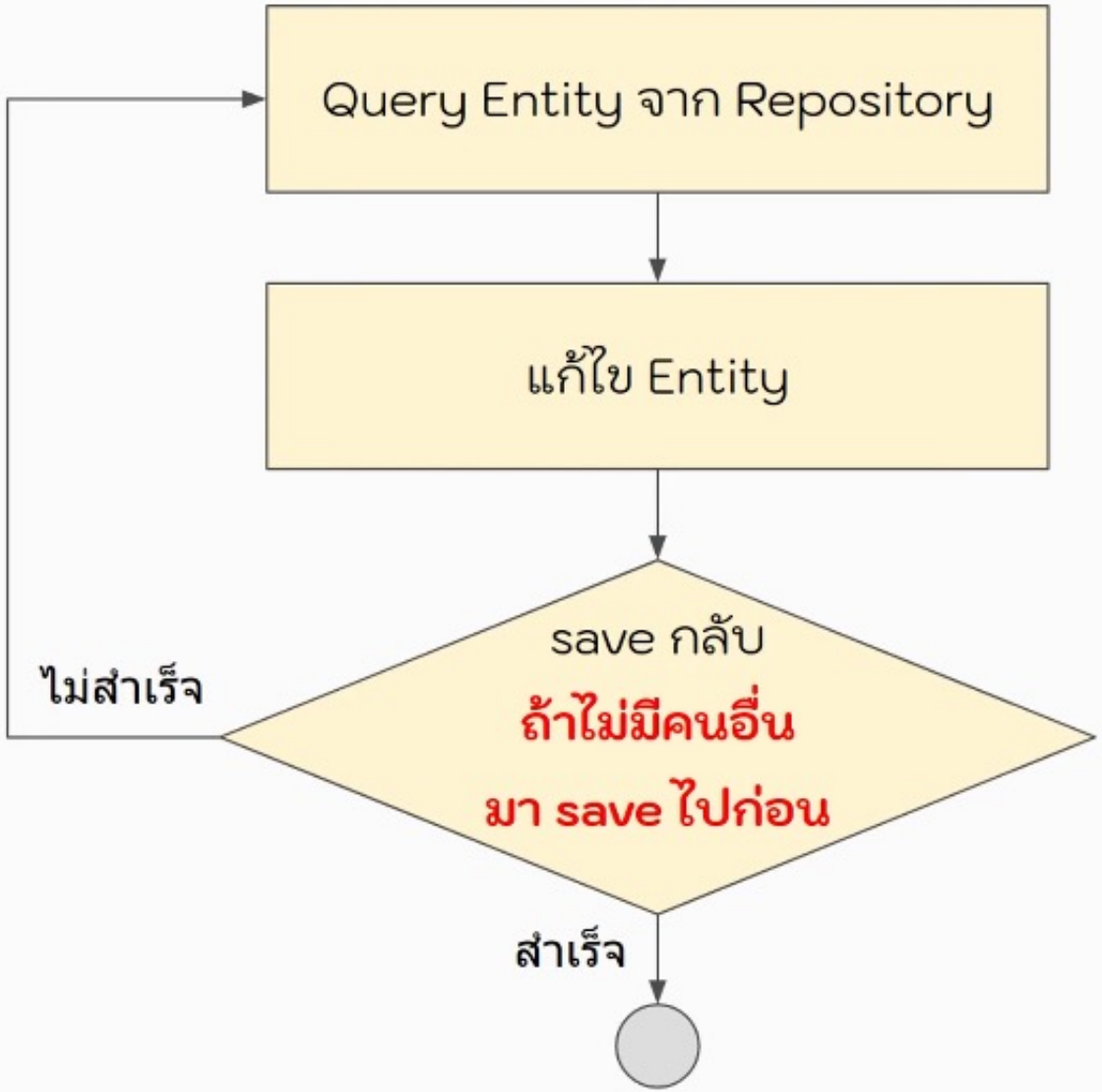

User 1
User 2
save(entity)
save(entity)
Repository
update 1
update 2
DB
UPDATE 1
โดน save ทับ
UPDATE 2



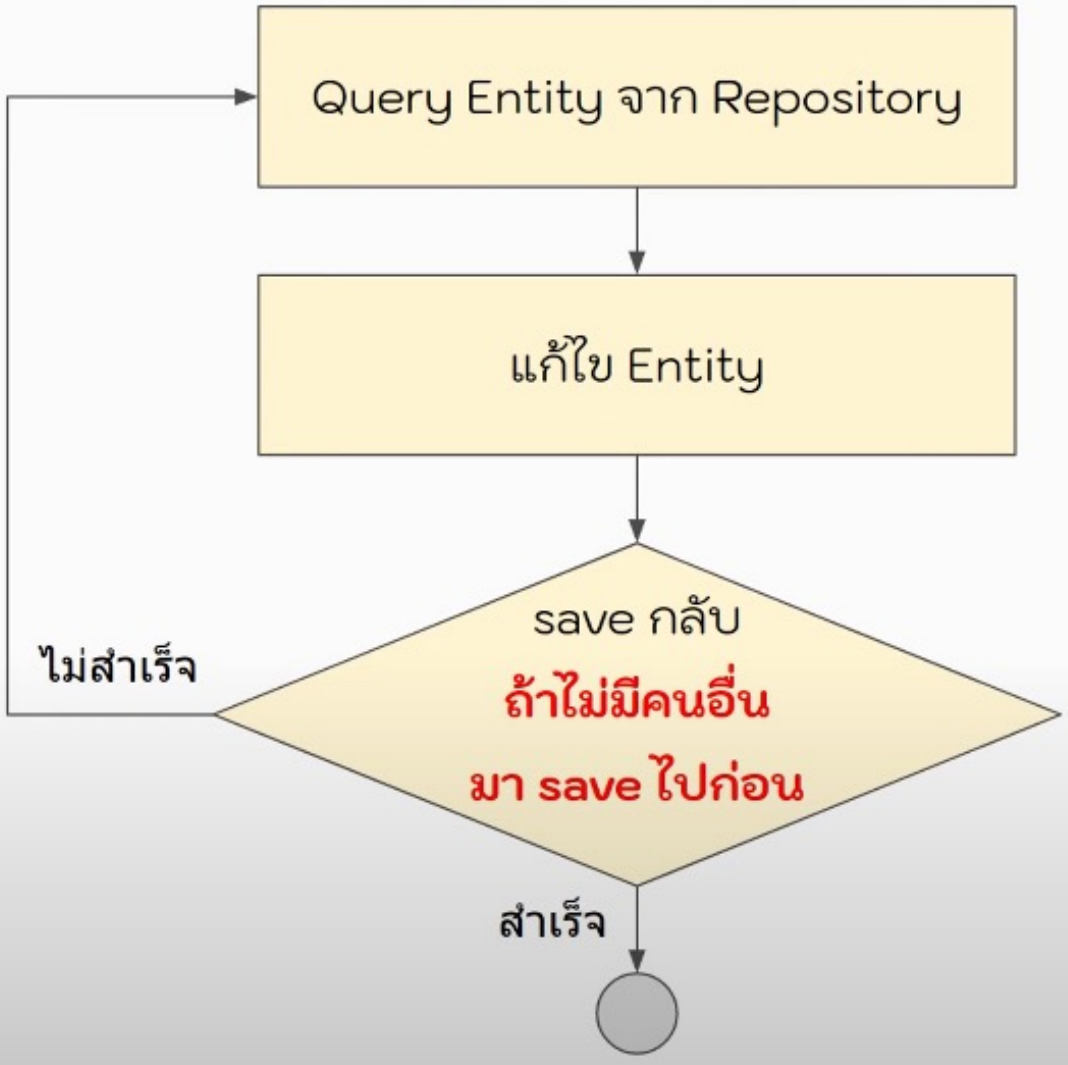
ปัญหา:

~~implement logic บนภาษา programming ยังไง~~
ทำ operation ต่าง ๆ ให้เป็น transaction ยังไง

• Optimistic Locking



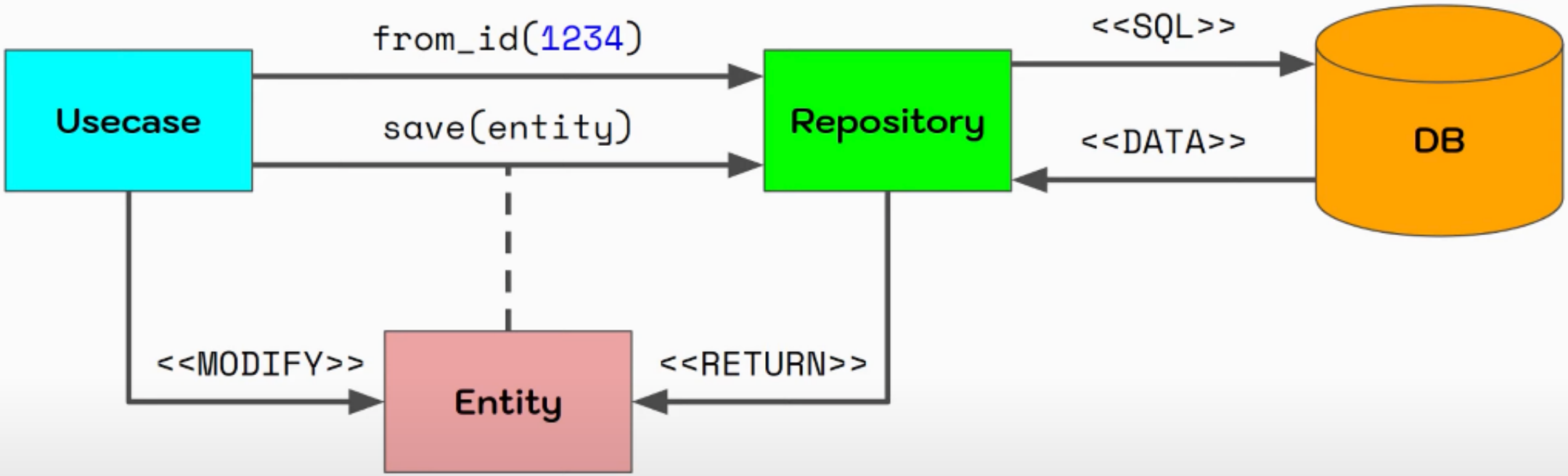
• Optimistic Locking



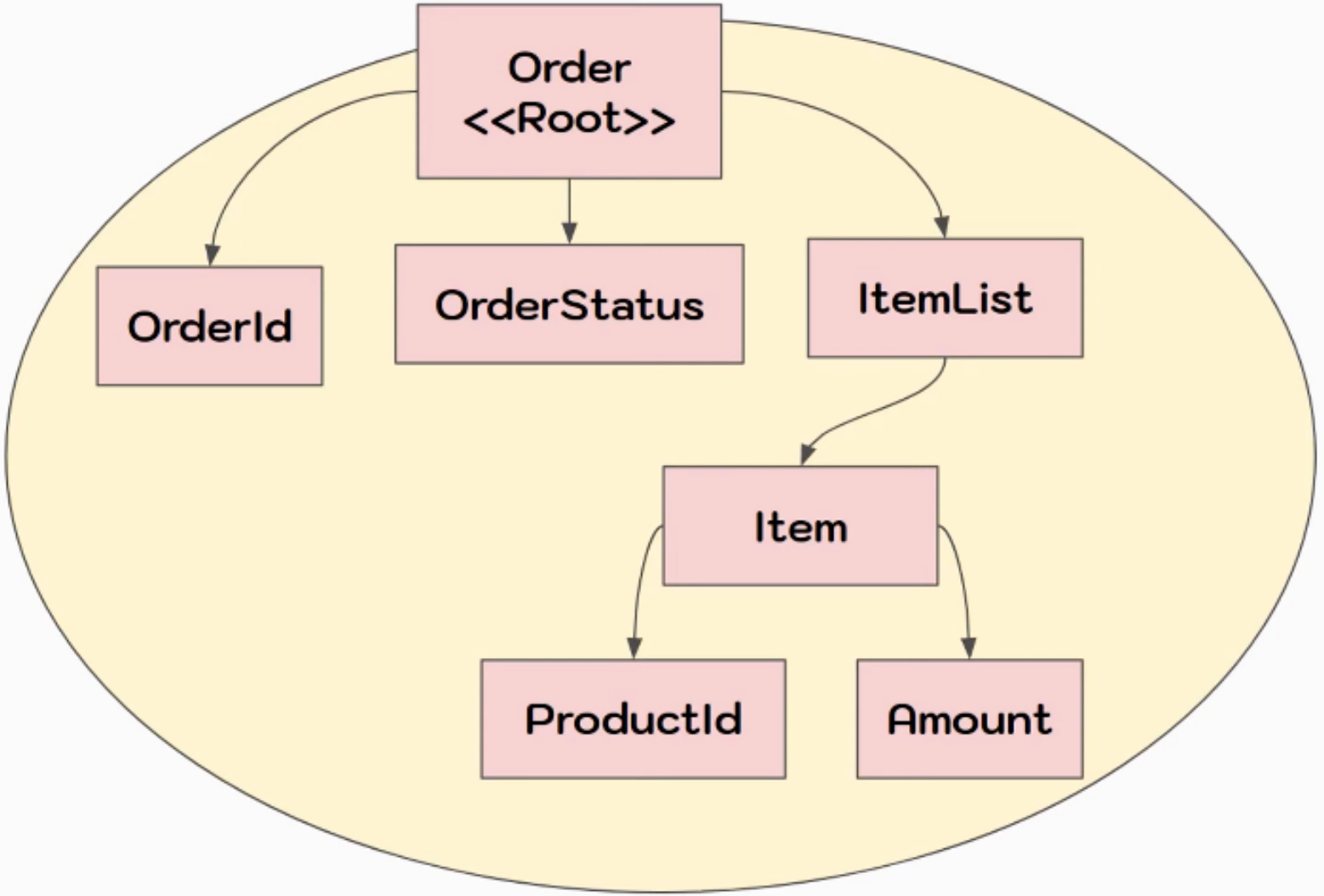
version ล่าสุดตอนนี้ = 3

```
UPDATE ...
SET ..., version = version + 1
WHERE ... AND version = 3
```

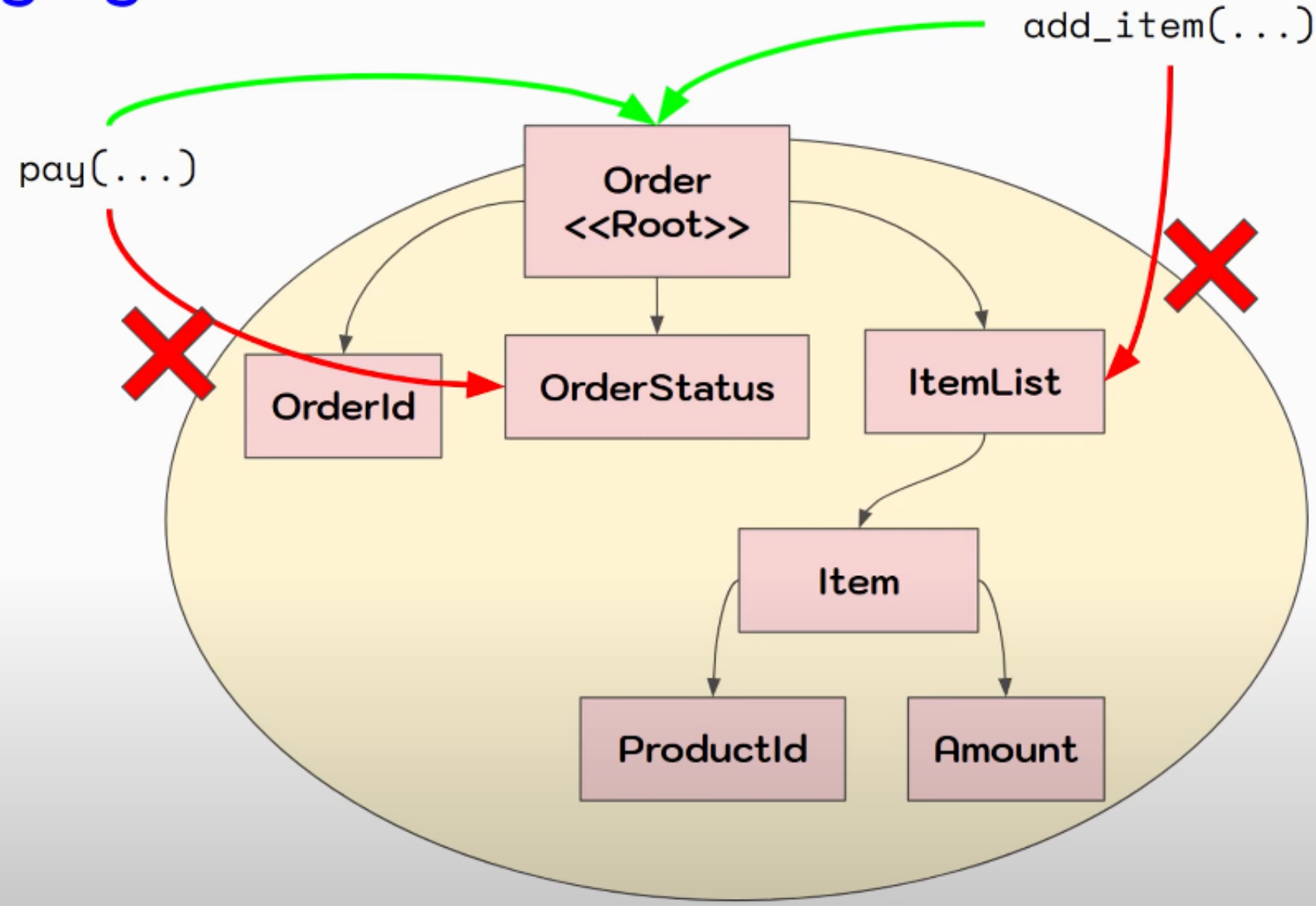
Repository Pattern



- Aggregate Pattern



Aggregate Pattern



ไม่ใช้ aggregate

```
order = all_orders.from_id(5555)


if order.status == “PAID”:
  raise OrderAlreadyPaidException()
else:
  order.items.append(new_item)


all_orders.save(order)
```

ไม่ใช้ aggregate

```
order = all_orders.from_id(5555)

if order.status == “PAID”:
  raise OrderAlreadyPaidException()
elif order.status == “CANCELLED”:
  raise OrderAlreadyCancelledException()
else:
  order.items.append(new_item)

all_orders.save(order)
```

นิยาม aggregate

```
class Order(Aggregate):

  ...

  def add_item(self, item):
    if self.status == “PAID”:
      raise OrderAlreadyPaidException()
    elif self.status == “CANCELLED”:
      raise OrderAlreadyCancelledException()
    else:
      self.items.append(new_item)
```

ใช้ aggregate

```
order = all_orders.from_id(5555)

order.add_item(new_item)

all_orders.save(order)
```

ใช้ aggregate

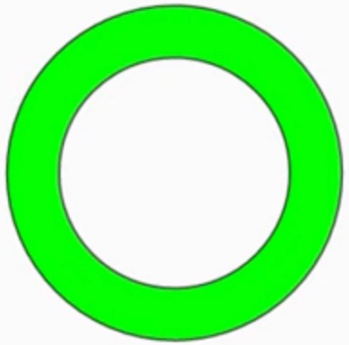
```
order = all_orders.from_id(5555)

order.pay()

all_orders.save(order)
```

ใช้ aggregate

```
order = all_orders.from_id(5555)

order.cancel()

all_orders.save(order)
```

Method ที่มีความหมาย ในมุมธุรกิจ

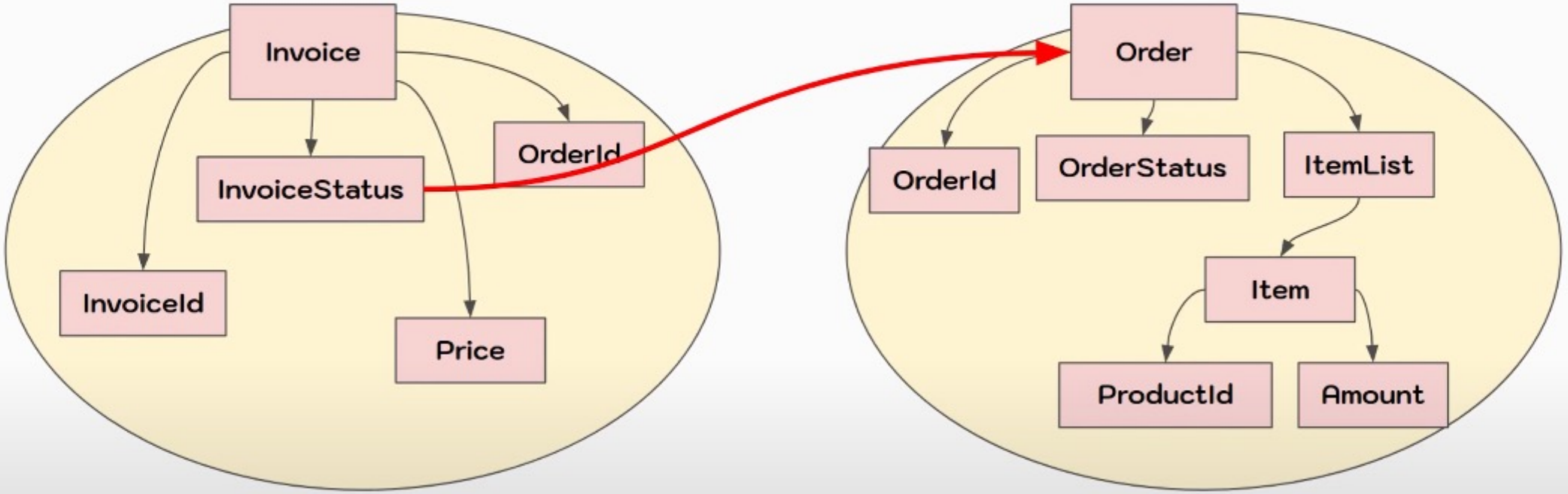
```
order.add_item(item)

order.pay()

order.cancel()
```



getter/setter

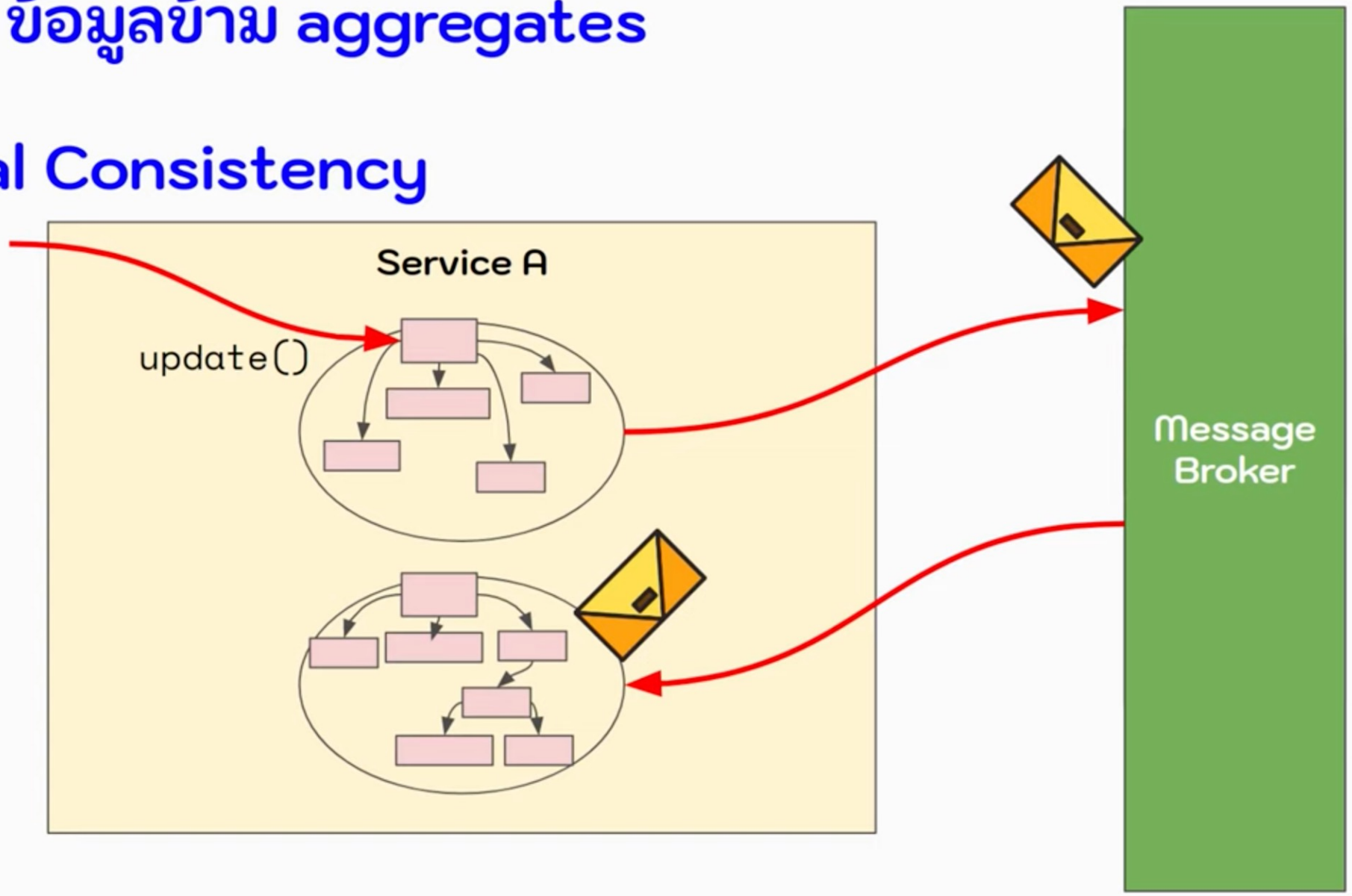
```
order.get_items()

order.get_status()

order.set_status("PAID")
```

- Update ข้อมูลข้าม aggregates



- Update ข้อมูลข้าม aggregates

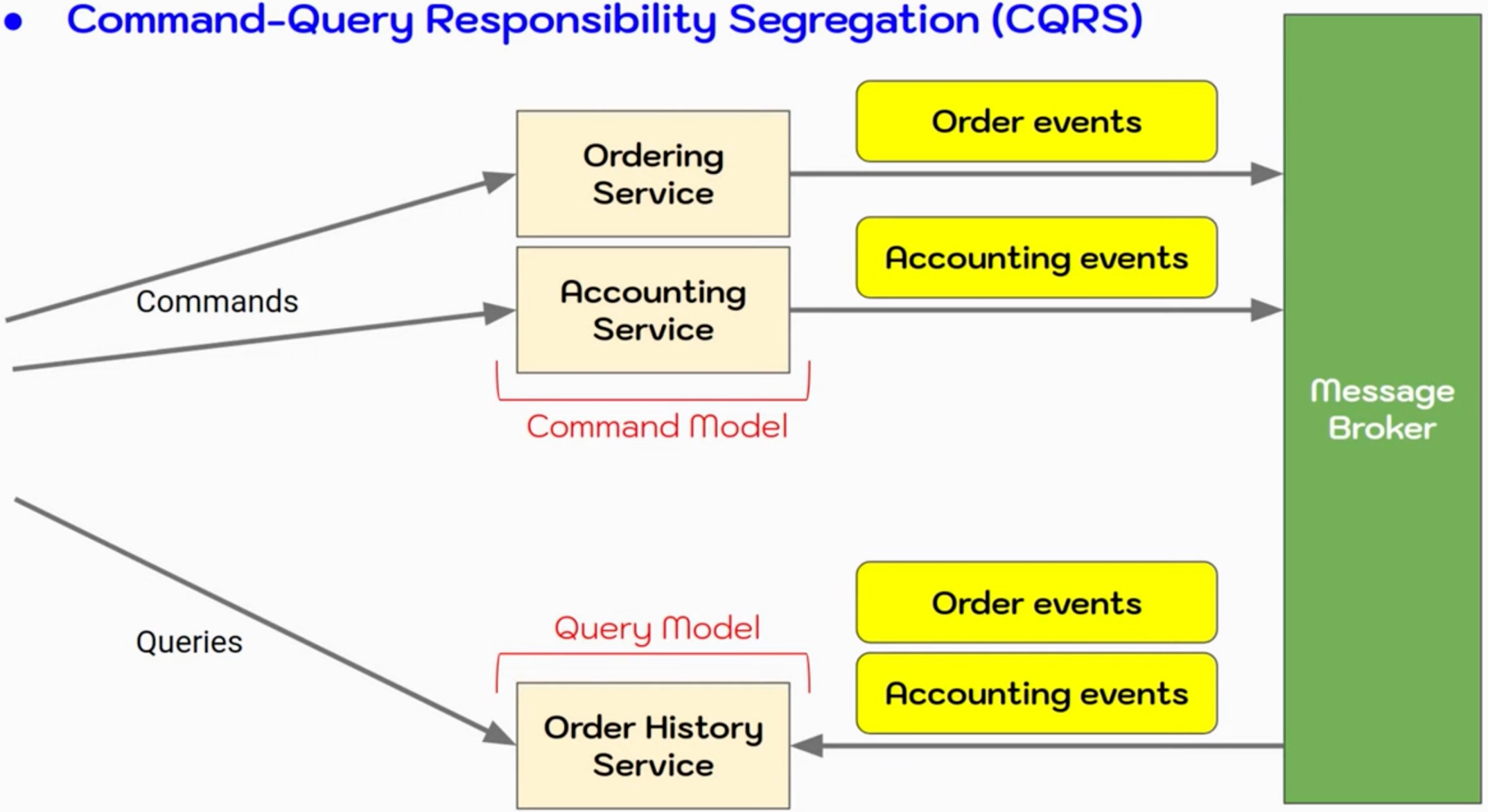
Eventual Consistency



ออกแบบ aggregate (ฉบับย่อ)

- เริ่มจาก Aggregate เล็ก ๆ
- ค่อย ๆ เพิ่ม object ที่เกี่ยวข้อง หรือต้อง update พร้อมกันทันที
- update Aggregate อื่นโดยใช้ Eventual Consistency เท่านั้น
- ให้ความสำคัญกับ Bounded Context (ดูคลิป part 1-2)

- Command-Query Responsibility Segregation (CQRS)



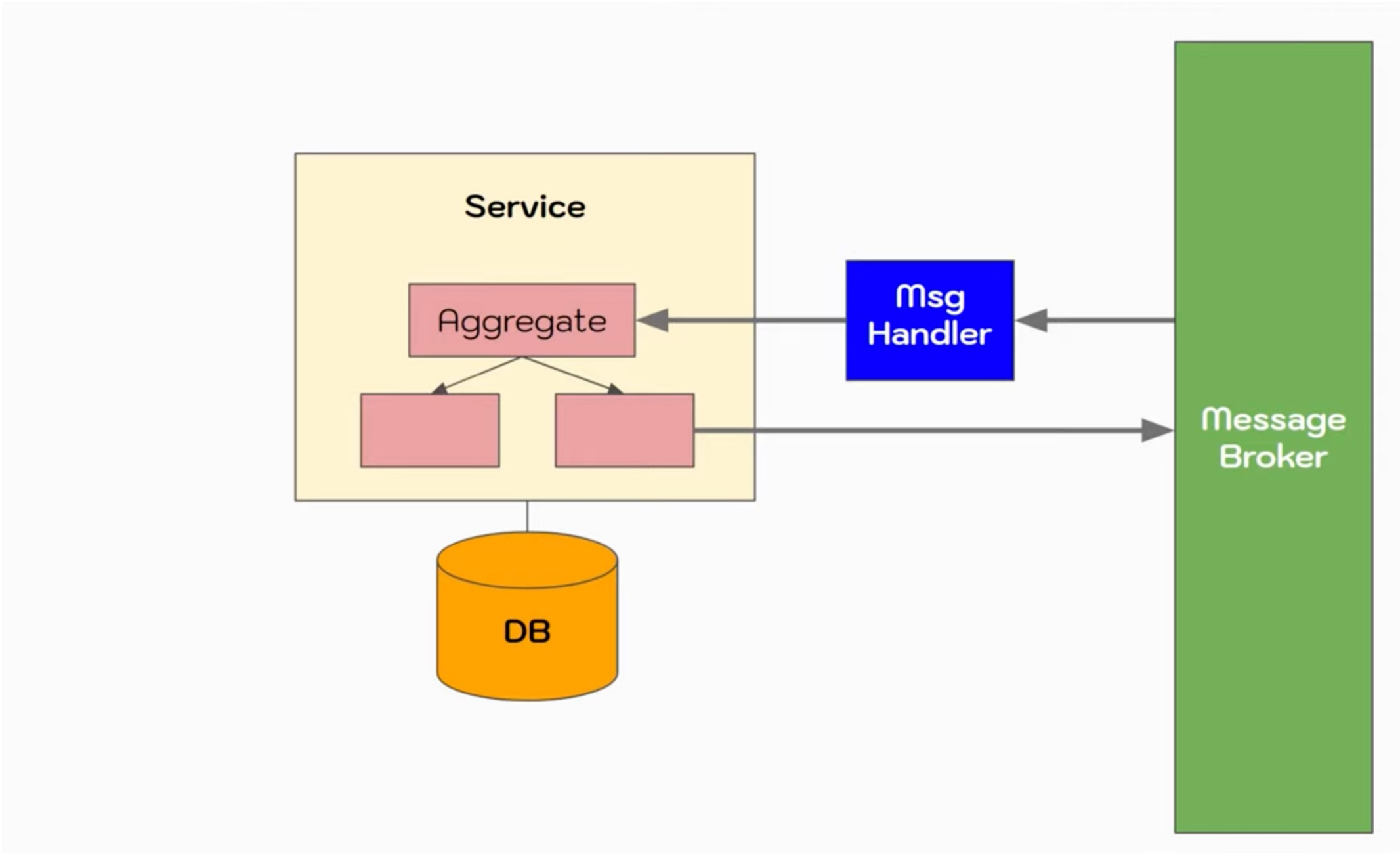
เมนูวันนี้ (2/3)

Data Access/Manipulation

- Repository Pattern
- Aggregate Pattern
- CQRS

Event-based Communication

- Event Sourcing
- Transactional Outbox


Service
Aggregate
Msg
Handler
Message
Broker
DB

Submit Order

```
order = all_orders.from_id(5555)


order.submit()


all_orders.save(order)   # update ข้อมูลใน db
```

Submit Order

```
DomainEvent.publish(OrderSubmittedEvent(...))

order = all_orders.from_id(5555)

order.submit()

all_orders.save(order)  # update ข้อมูลใน db
```

Submit Order

```
order = all_orders.from_id(5555)

order.submit()

all_orders.save(order)  # update ข้อมูลใน db

DomainEvent.publish(OrderSubmittedEvent(...))
```

Submit Order

```
DomainEvent.publish(OrderSubmittedEvent(...))

order = all_orders.from_id(5555)

order.submit()

all_orders.save(order)  # update ข้อมูลใน db
```

สำเร็จ 

ไม่สำเร็จ 

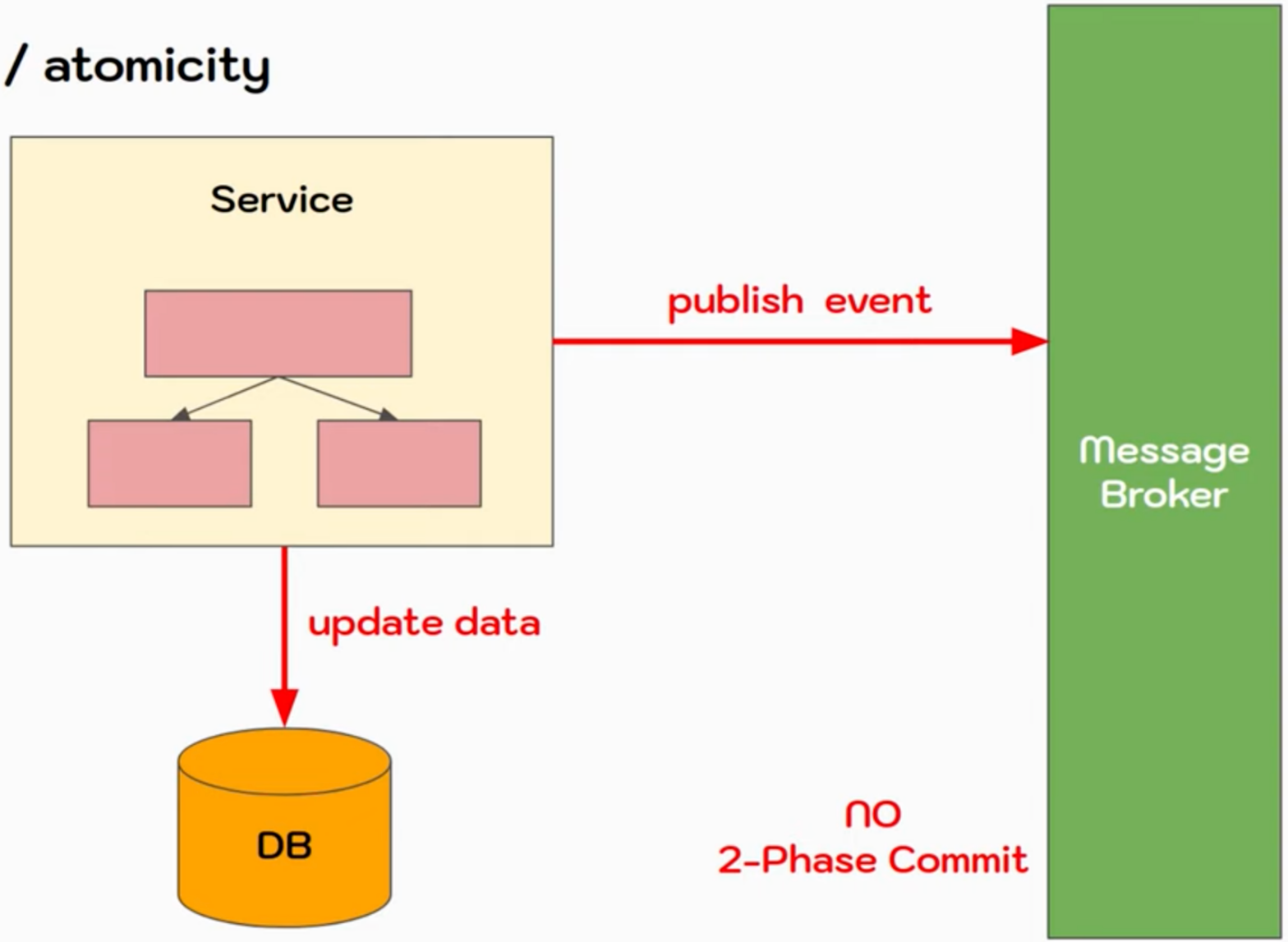
Submit Order

```
order = all_orders.from_id(5555)

order.submit()

all_orders.save(order)  # update ข้อมูลใน db

DomainEvent.publish(OrderSubmittedEvent(...))
```

สำเร็จ 

ไม่สำเร็จ 

ปัญหา: reliability / atomicity

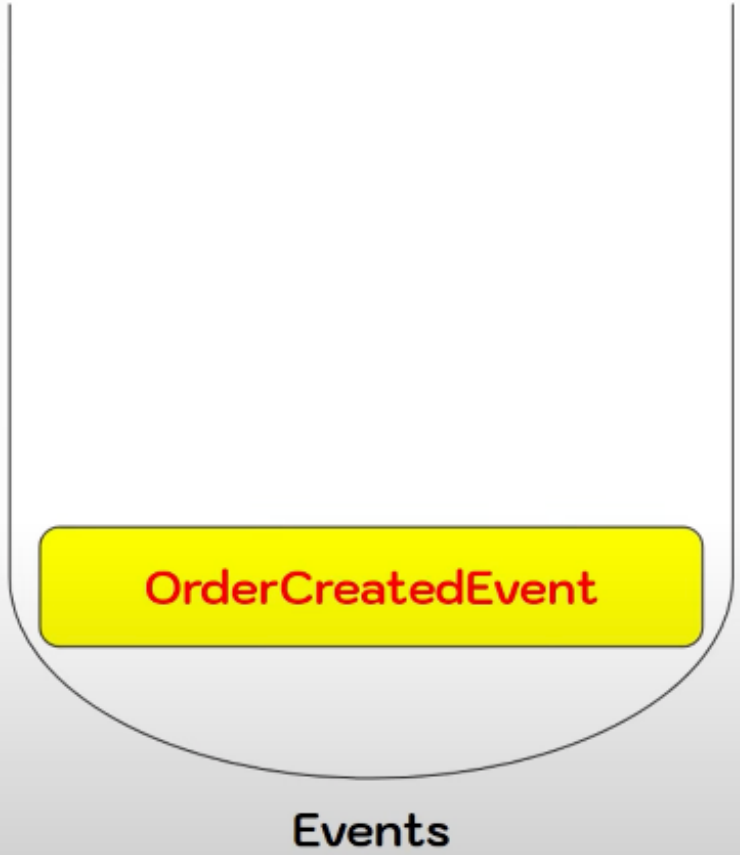


- Event Sourcing

Event Sourcing

สถานะปัจจุบัน Order

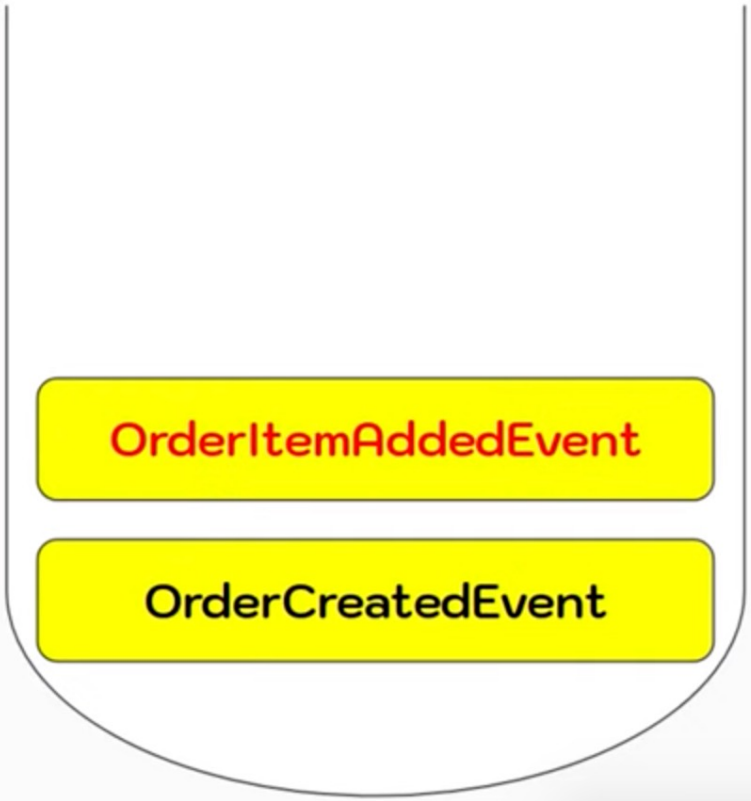
```
{
  id: 1234,
  buyerId: 9999,
  items: [],
  status: “pending”
}
```



Event Sourcing

สถานะปัจจุบัน Order

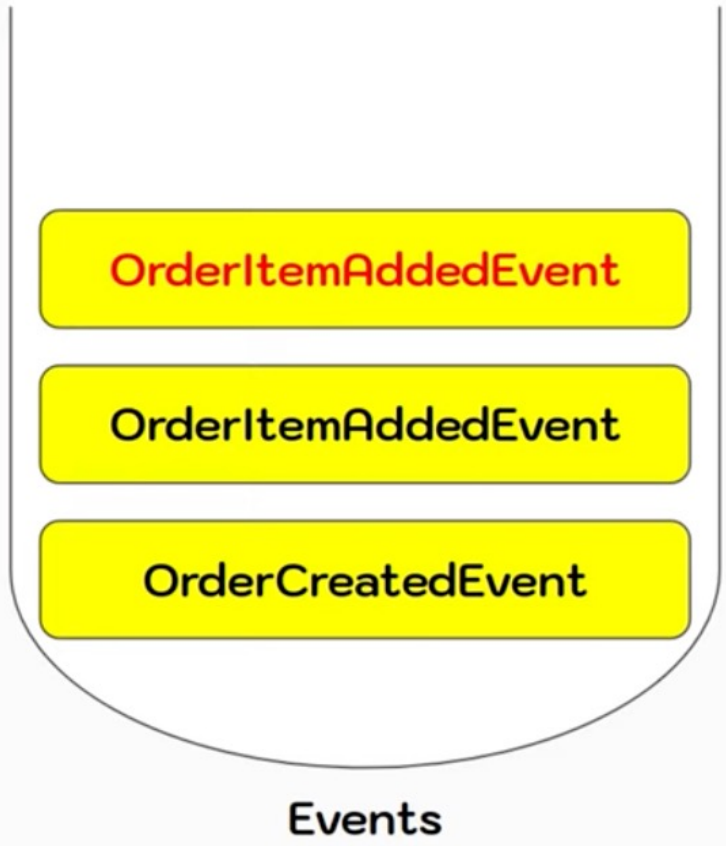
```
{
  id: 1234,
  buyerId: 9999,
  items: [
    {productId: 1111, amount: 3}
  ],
  status: “pending”
}
```



Event Sourcing

สถานะปัจจุบัน Order

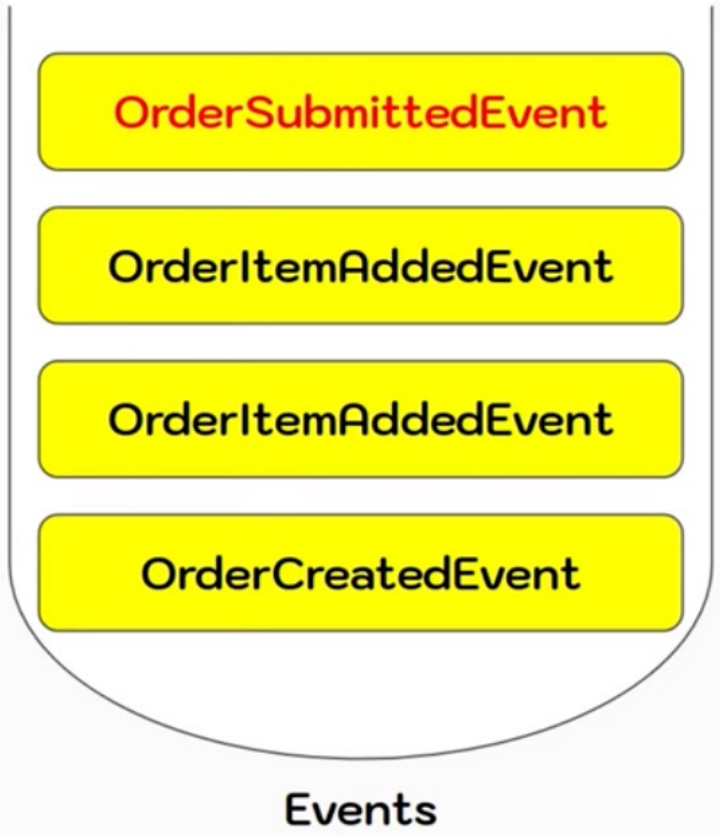
```
{
  id: 1234,
  buyerId: 9999,
  items: [
    {productId: 1111, amount: 3},
    {productId: 2222, amount: 8}
  ],
  status: “pending”
}
```



Event Sourcing

สถานะปัจจุบัน Order

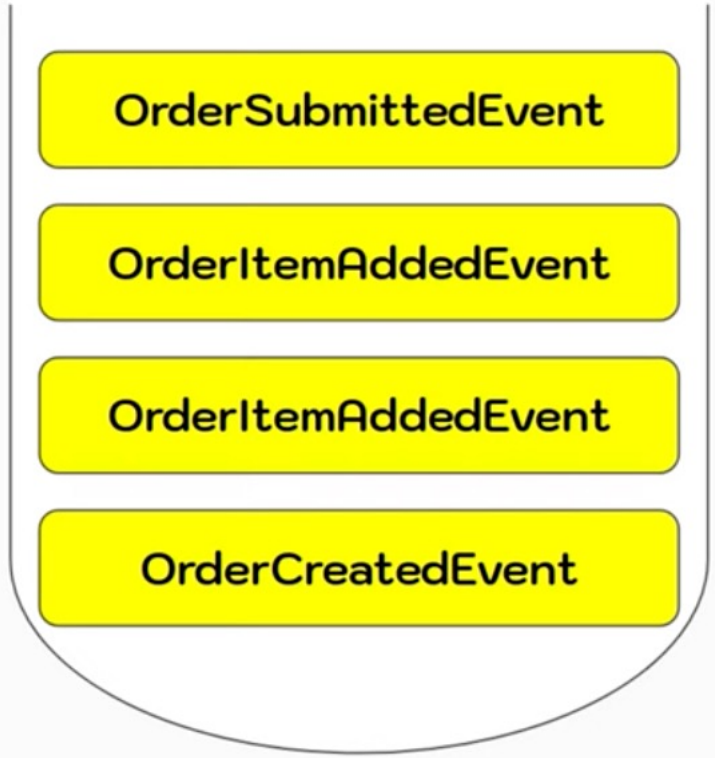
```
{
  id: 1234,
  buyerId: 9999,
  items: [
    {productId: 1111, amount: 3},
    {productId: 2222, amount: 8}
  ],
  status: “submitted”
}
```



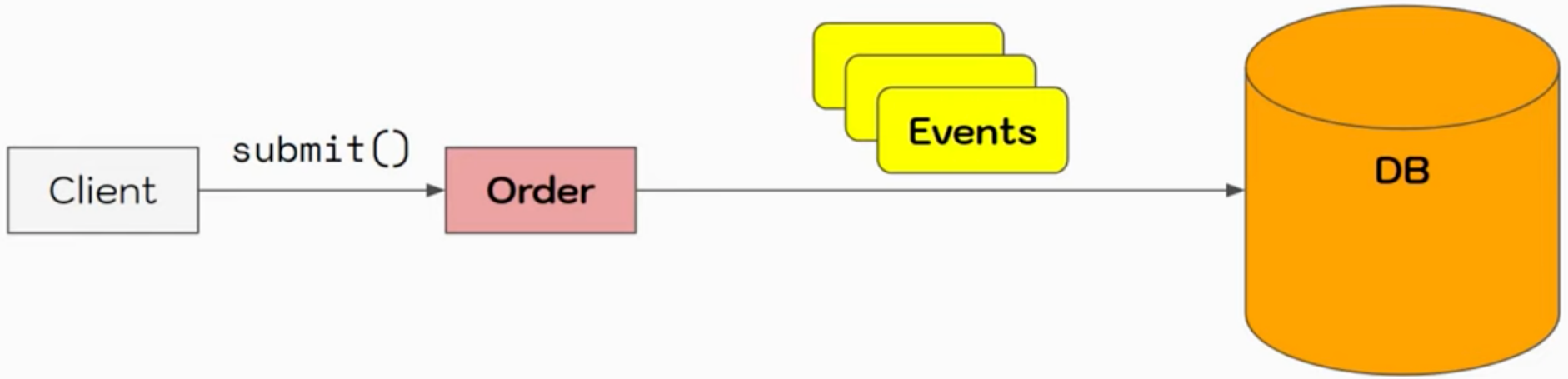
Event Sourcing

สถ ะปัจจุบัน Order

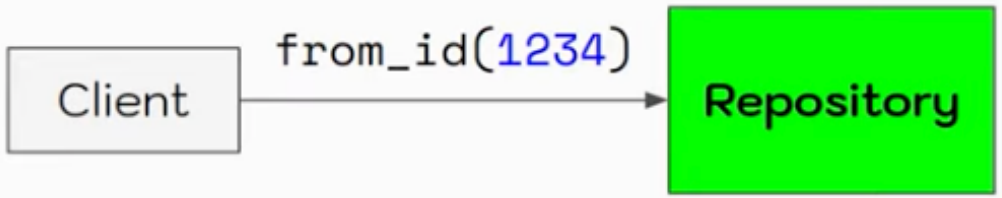
```
{
  id: 1
  buyerId      9,
  items: [
    {productId:          amount: 3},
    {product     2      mount: 8}
  ],
  stat      submitted"
}
```

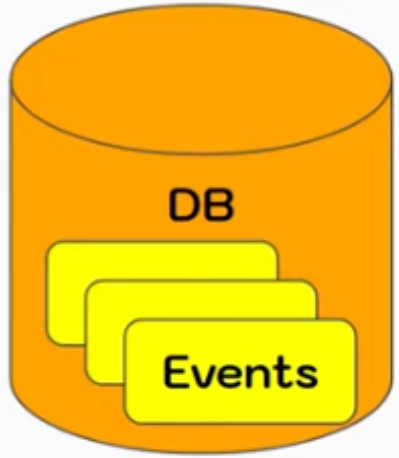


- **Event Sourcing**

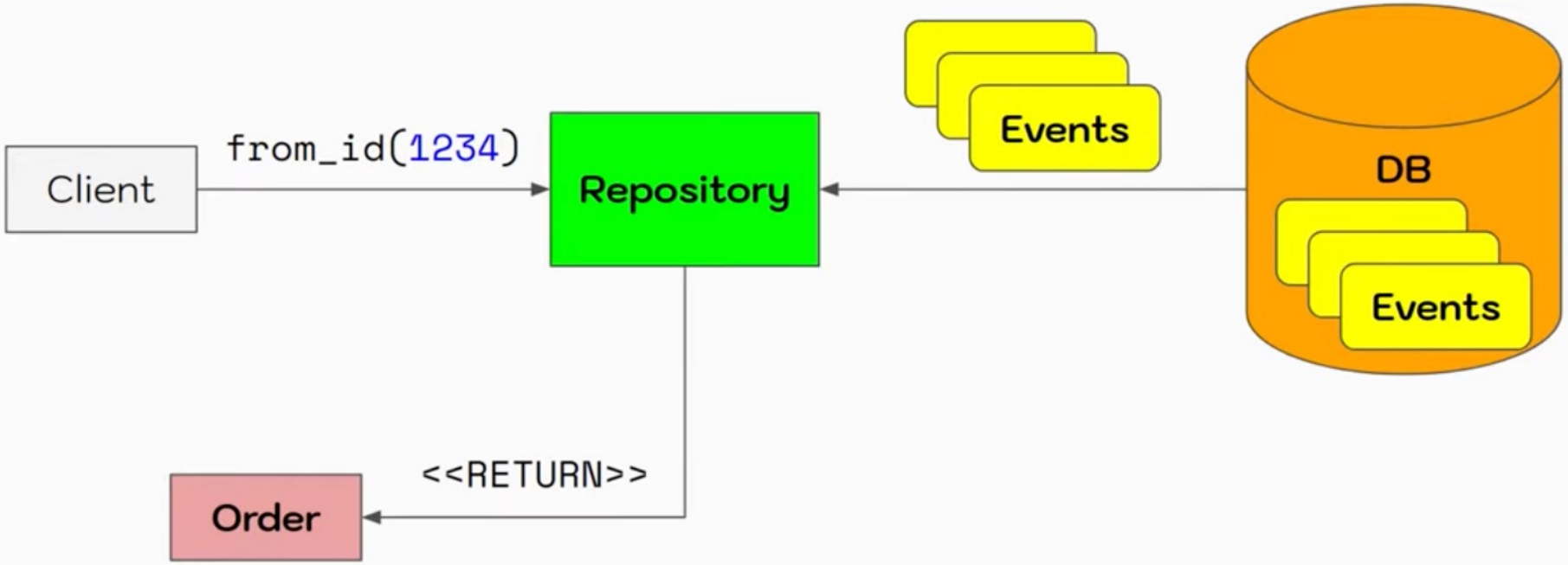


Event Sourcing

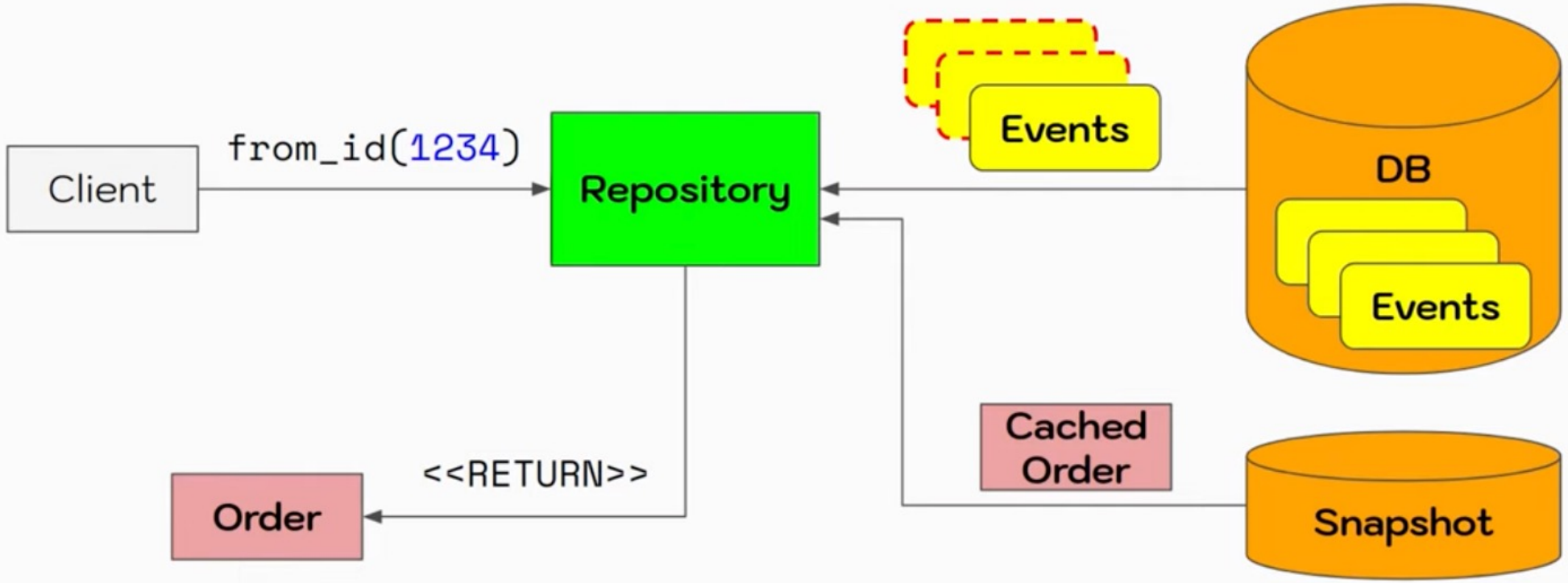




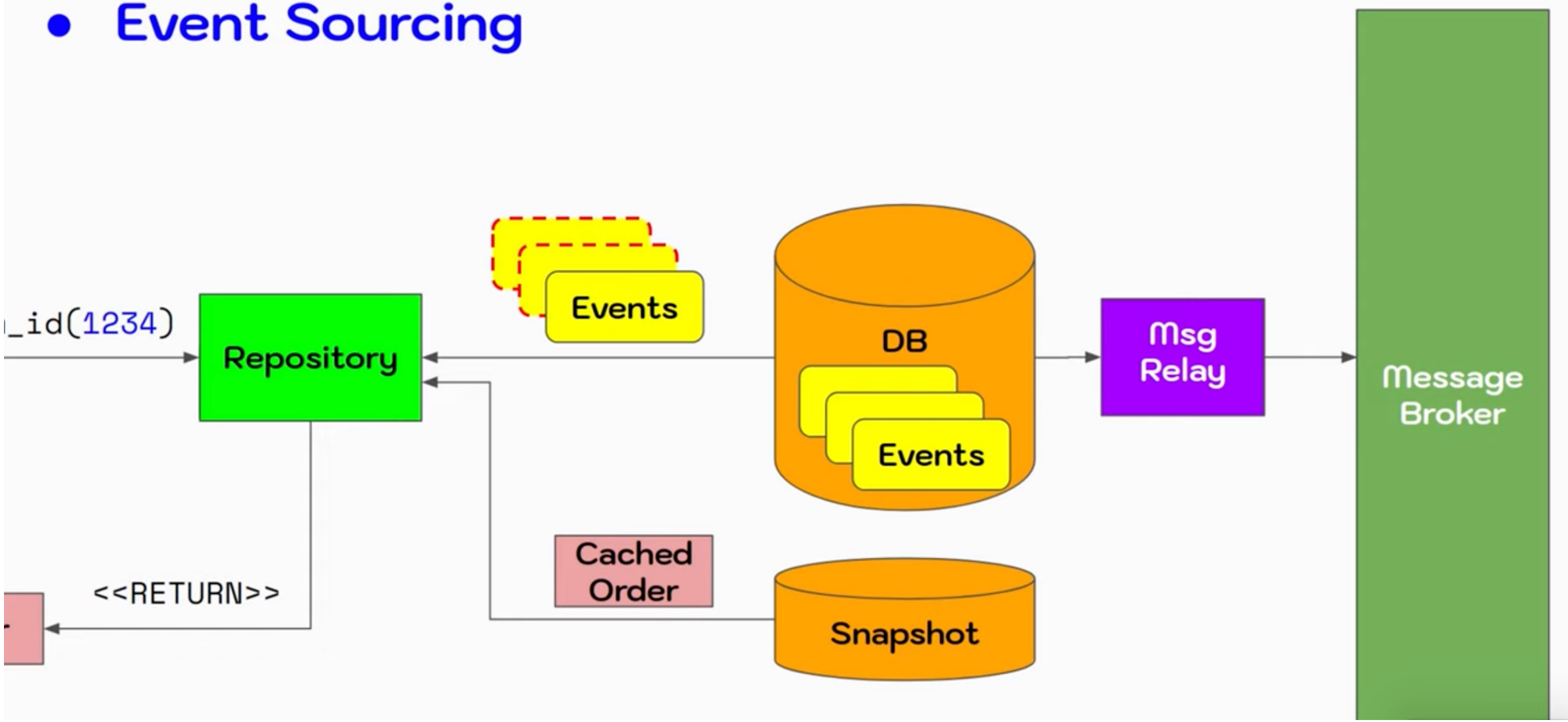
- **Event Sourcing**



• Event Sourcing

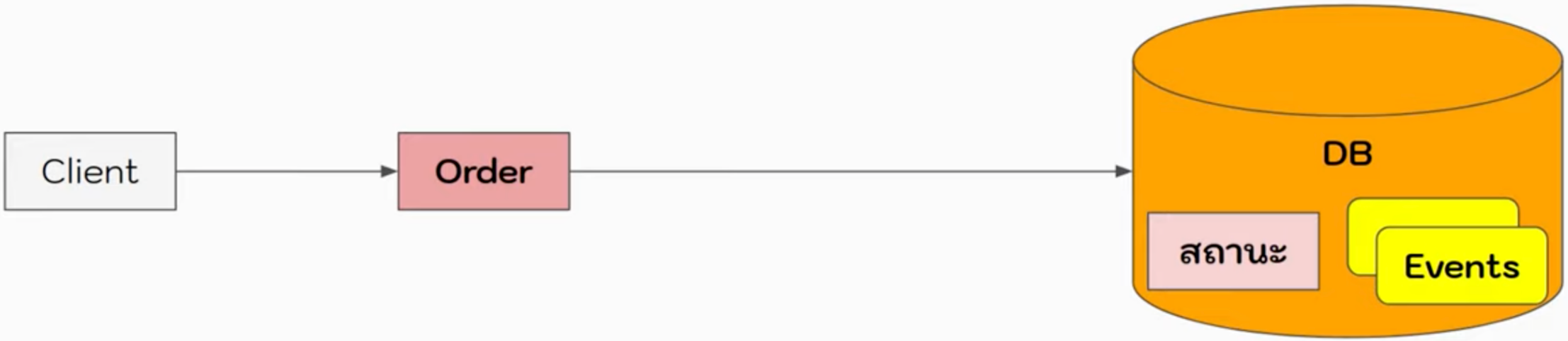


- Event Sourcing

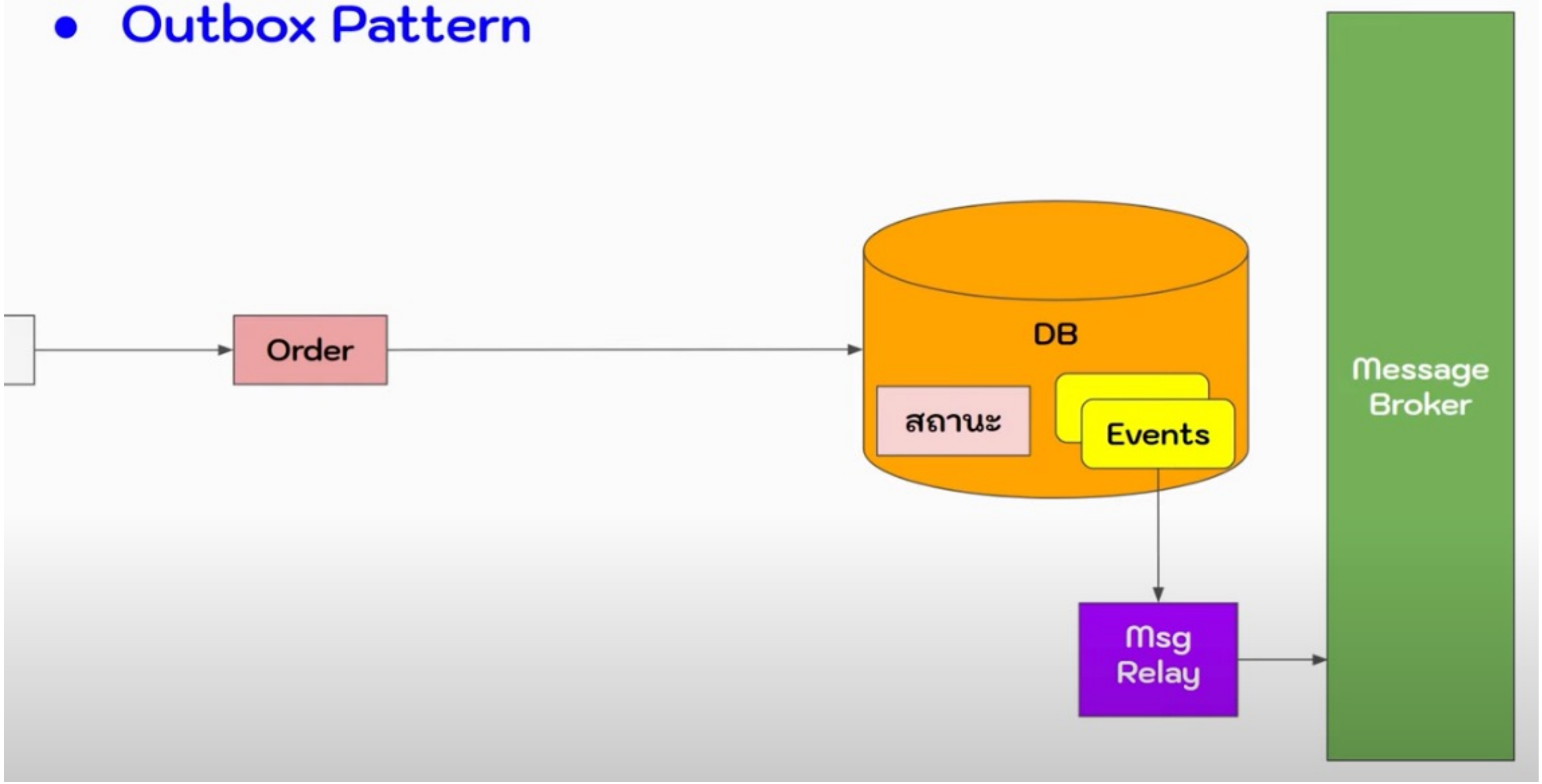


- **Outbox Pattern**

- **Outbox Pattern**

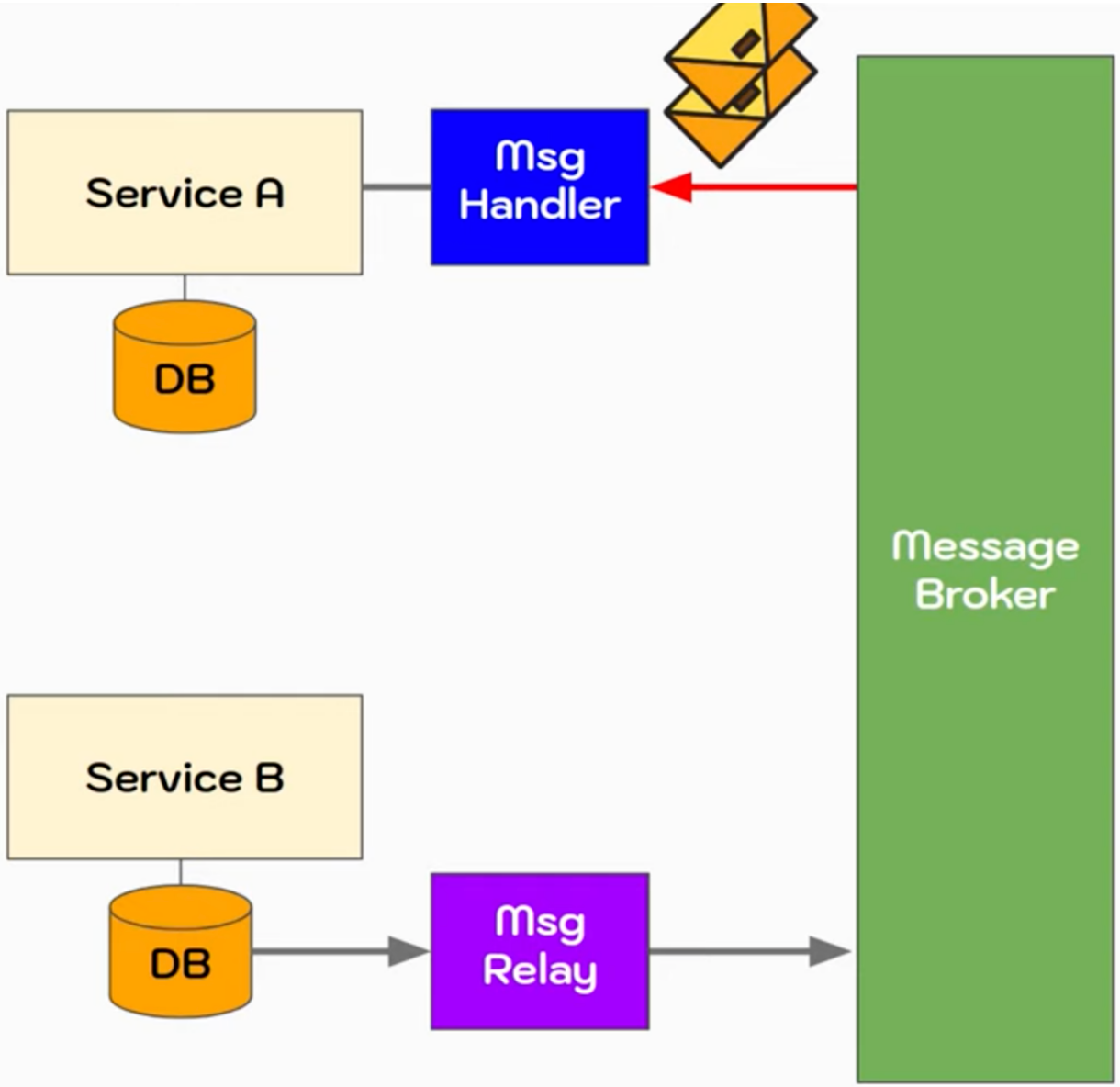


- Outbox Pattern



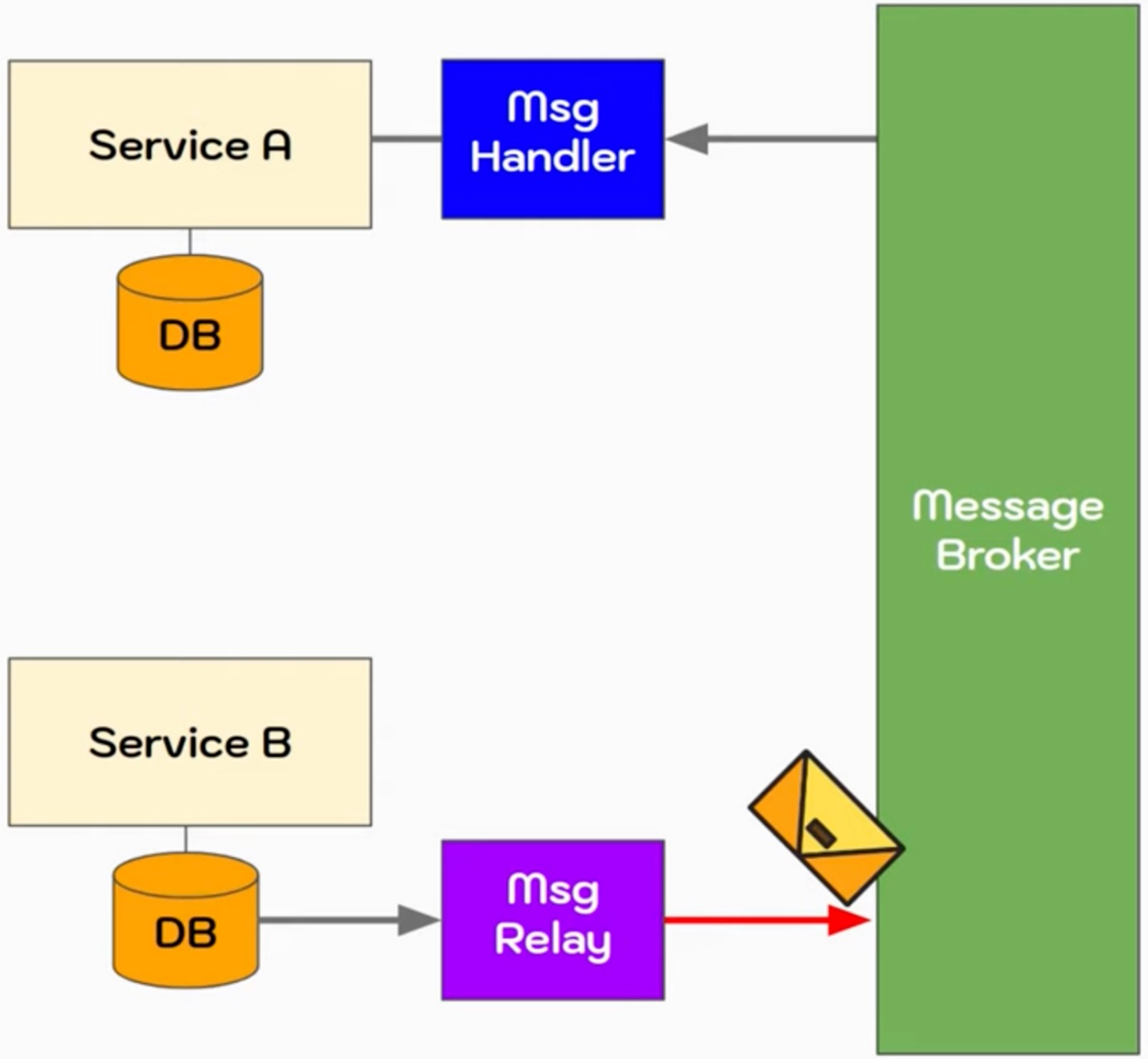
- **Inbox Pattern**

Idempotent Receiver



- **Inbox Pattern**

Idempotent Receiver

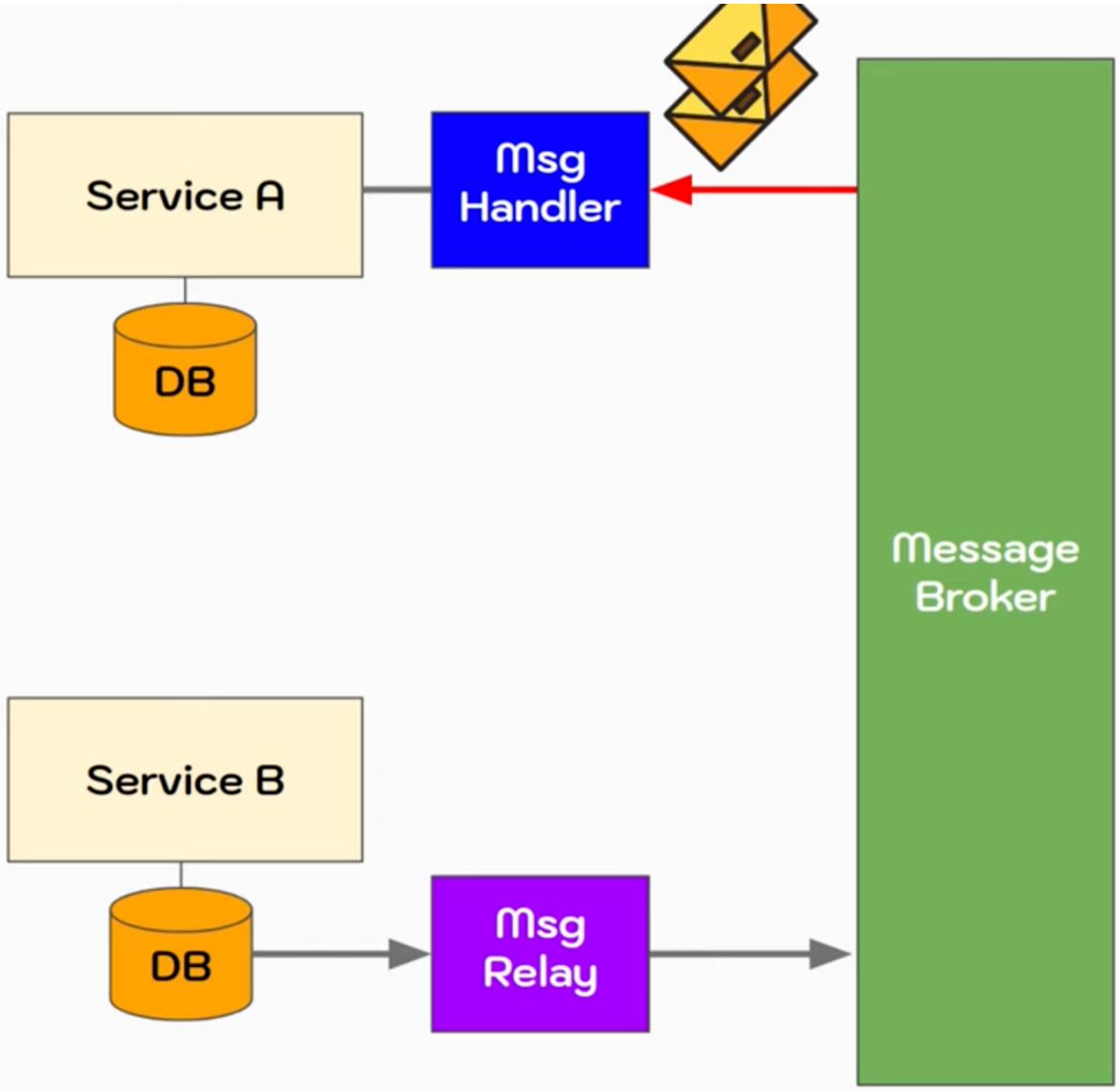


สาเหตุ 1

ฝั่ง producer publish ซ้ำ

- **Inbox Pattern**

Idempotent Receiver



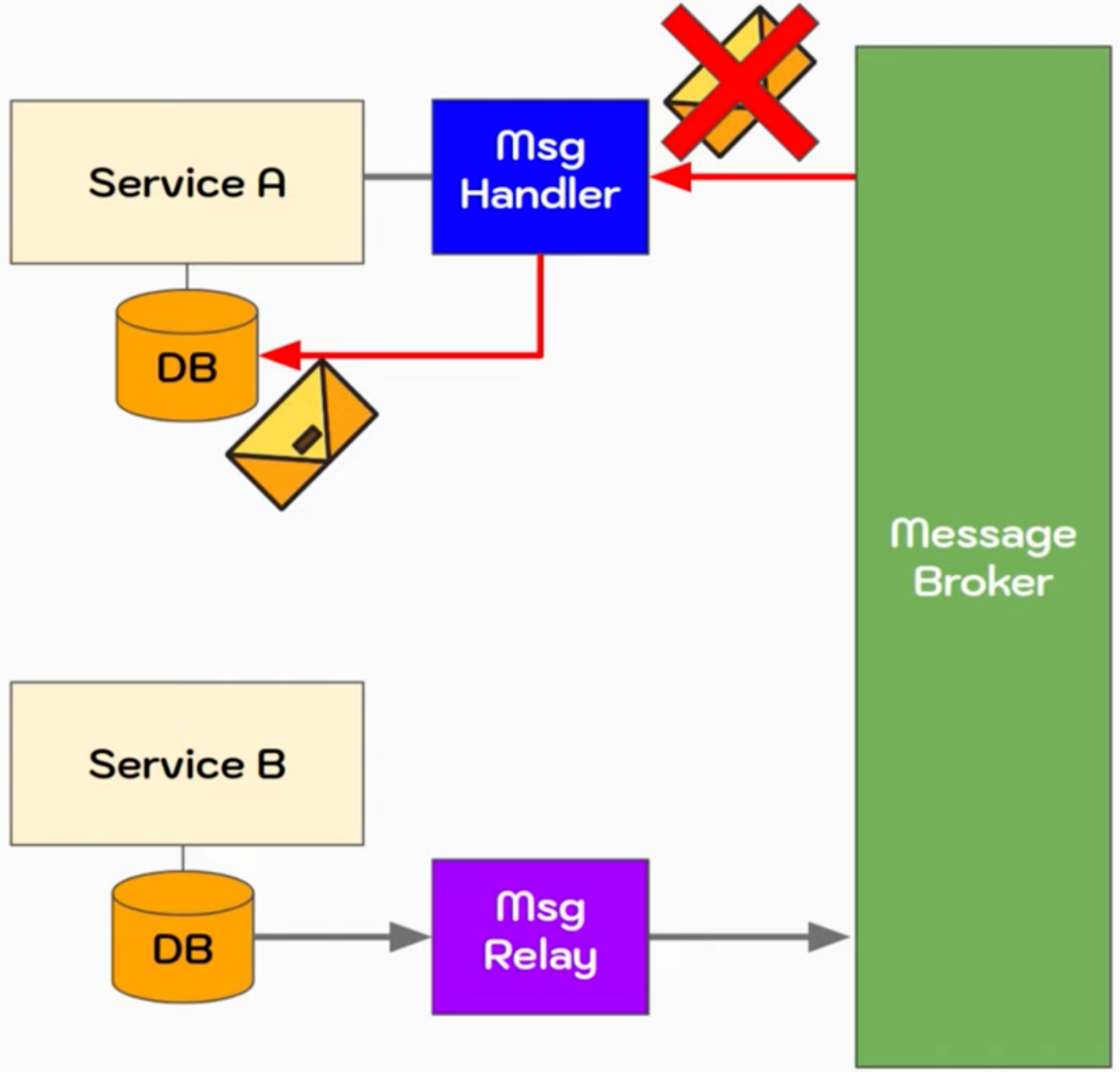
สาเหตุ 2

ฝั่ง consumer อ่านมาซ้ำ

- **Inbox Pattern**

Idempotent Receiver

แก้ปัญหาโดยใช้ **Unique ID** ลงในระบบ **DB**
เพื่อให้แน่ใจว่า “**message** ไม่ซ้ำ”



เมนูวันนี้ (3/3)

Hexagonal Architecture

Demo + Code with Python FastAPI & Kafka & MongoDB

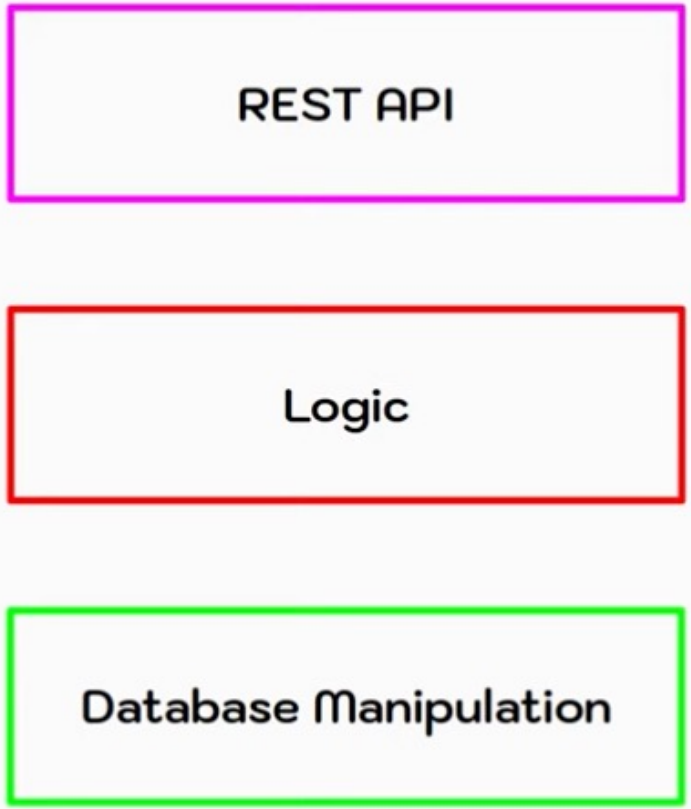
- Message Broker
- Transactional Outbox
- Repository Pattern
- Aggregate Pattern
- Hexagonal Architecture

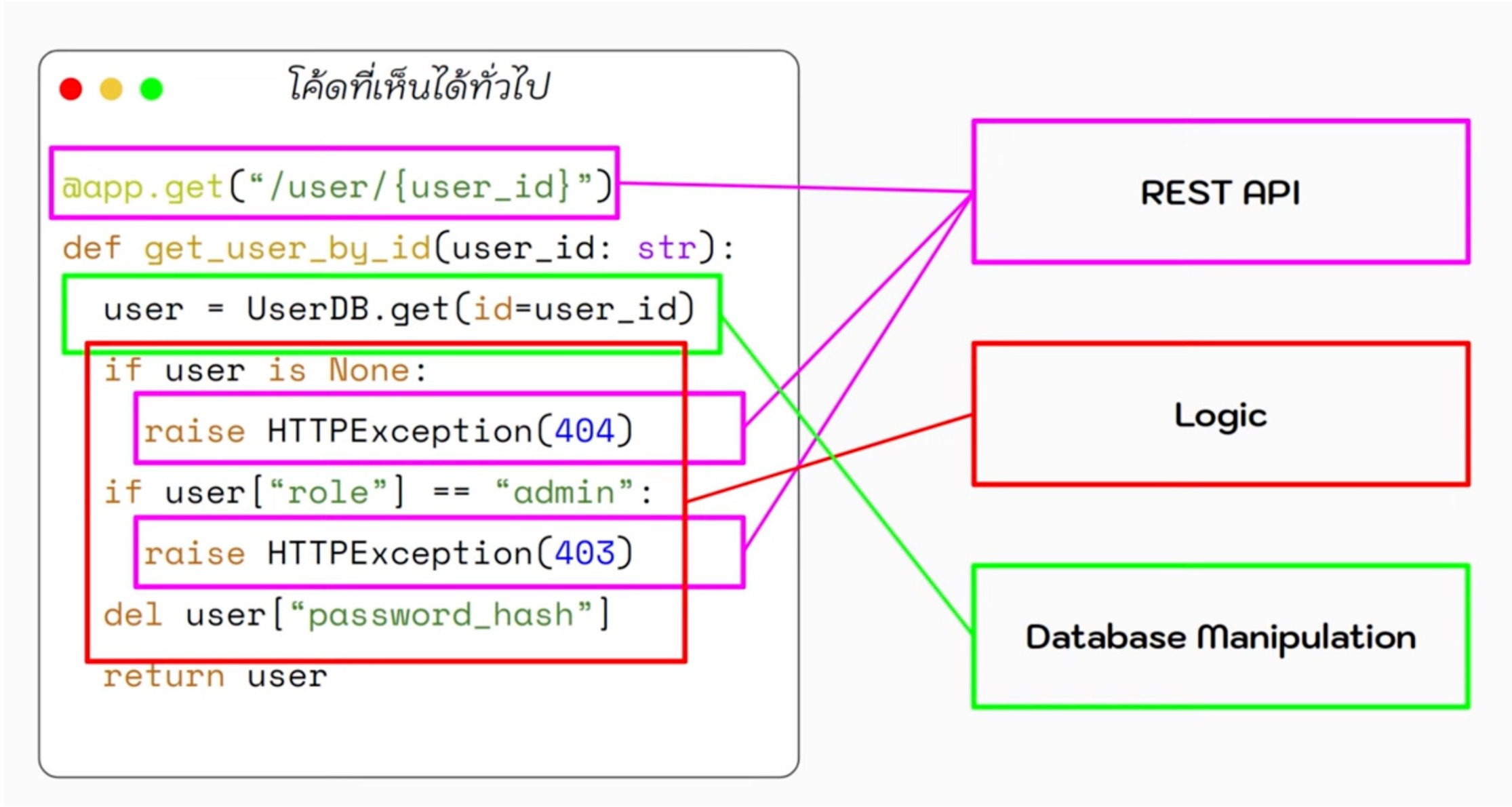
โค้ดที่เห็นได้ทั่วไป

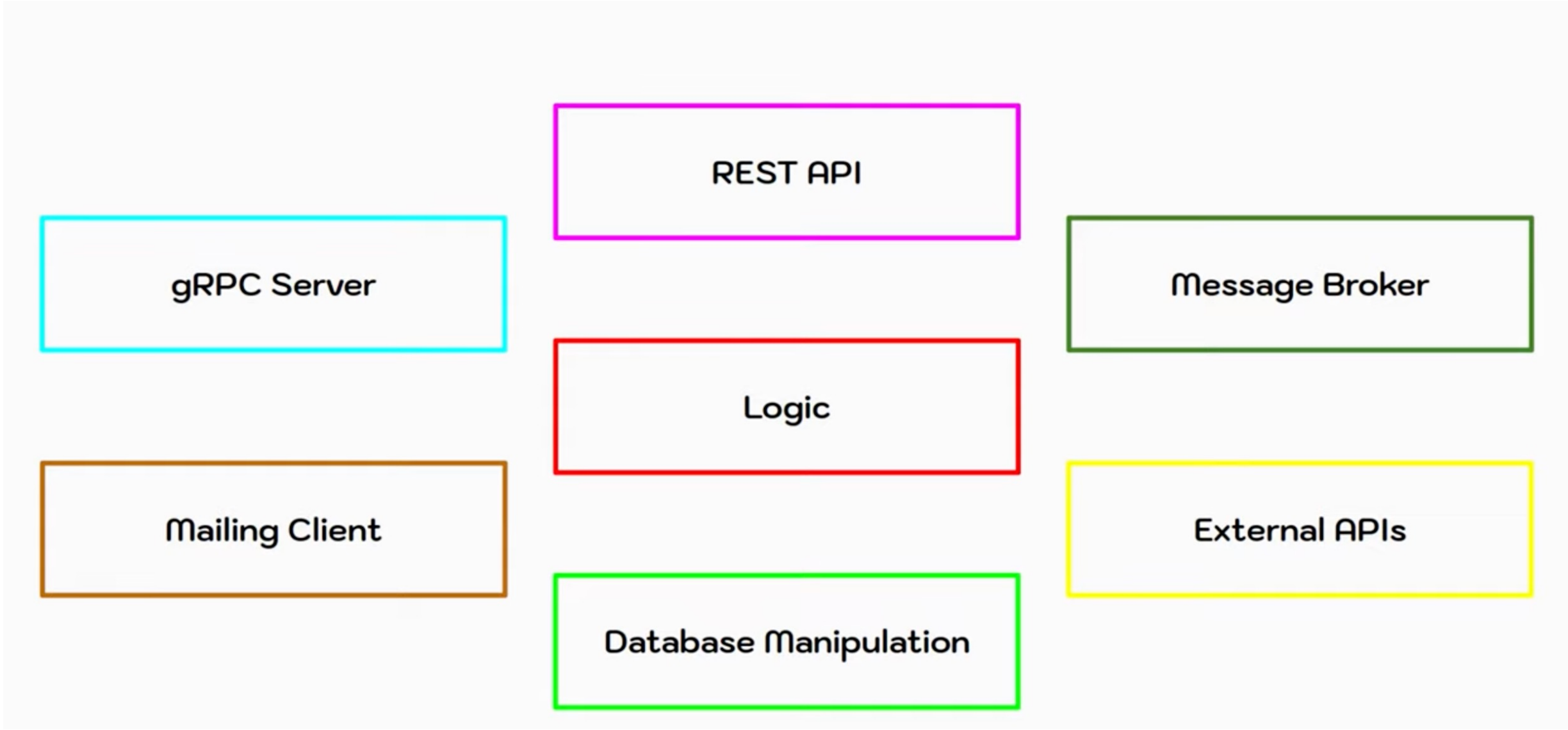
```python
@app.get("/user/{user_id}")
def get_user_by_id(user_id: str):
  user = UserDB.get(id=user_id)
  if user is None:
    raise HTTPException(404)
  if user["role"] == "admin":
    raise HTTPException(403)
  del user["password_hash"]
  return user
```

โค้ดที่เห็นได้ทั่วไป

```python
@app.get("/user/{user_id}")
def get_user_by_id(user_id: str):
  user = UserDB.get(id=user_id)
  if user is None:
    raise HTTPException(404)
  if user["role"] == "admin":
    raise HTTPException(403)
  del user["password_hash"]
  return user
```

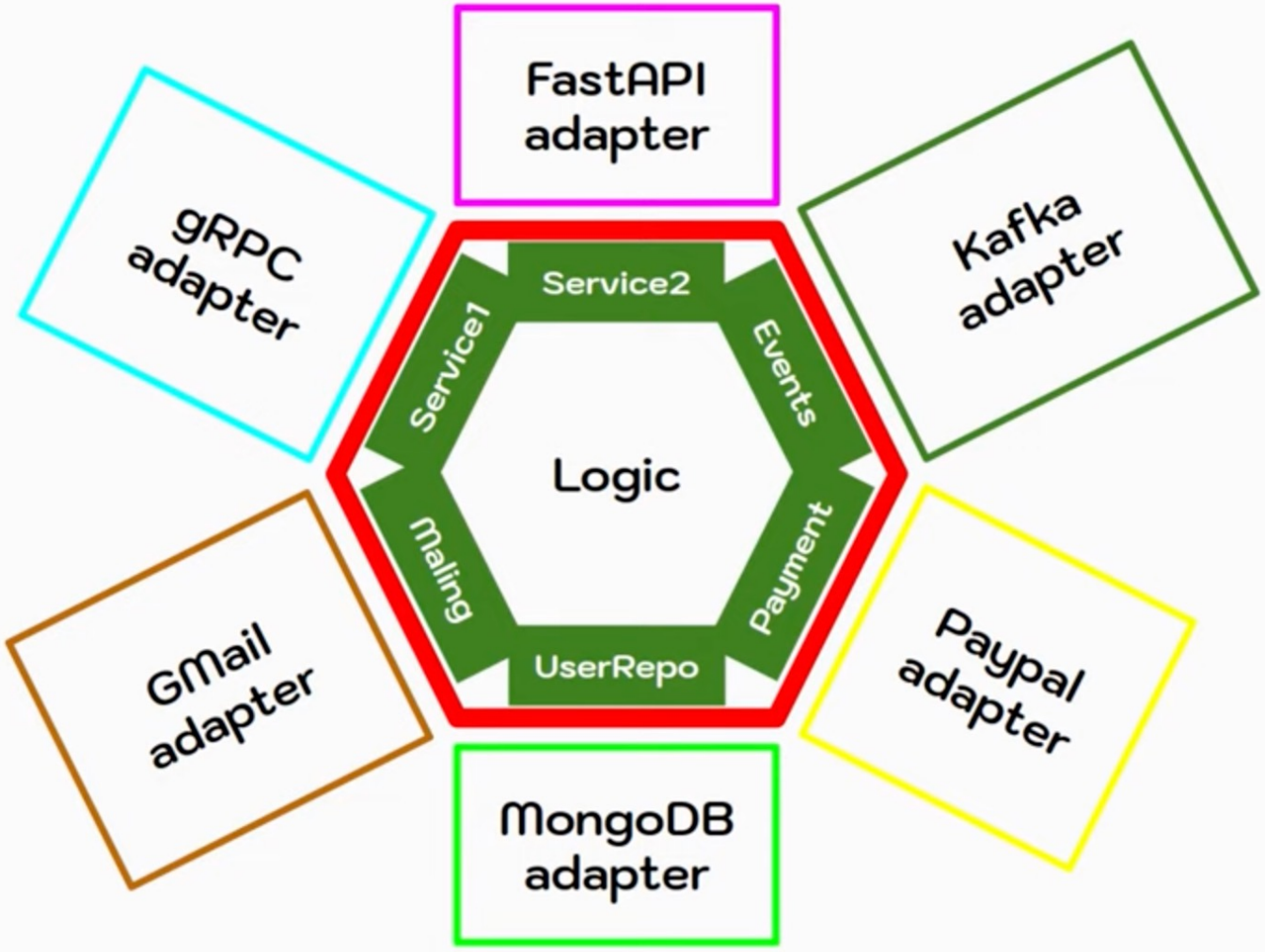



โค้ดที่เห็นได้ทั่วไป
@app.get("/user/{user_id}")
def get_user_by_id(user_id: str):
user = UserDB.get(id=user_id)
if user is None:
raise HTTPException(404)
if user["role"] == "admin":
raise HTTPException(403)
del user["password_hash"]
return user
REST API
Logic
Database Manipulation


REST API
gRPC Server
Message Broker
Logic
Mailing Client
External APIs
Database Manipulation

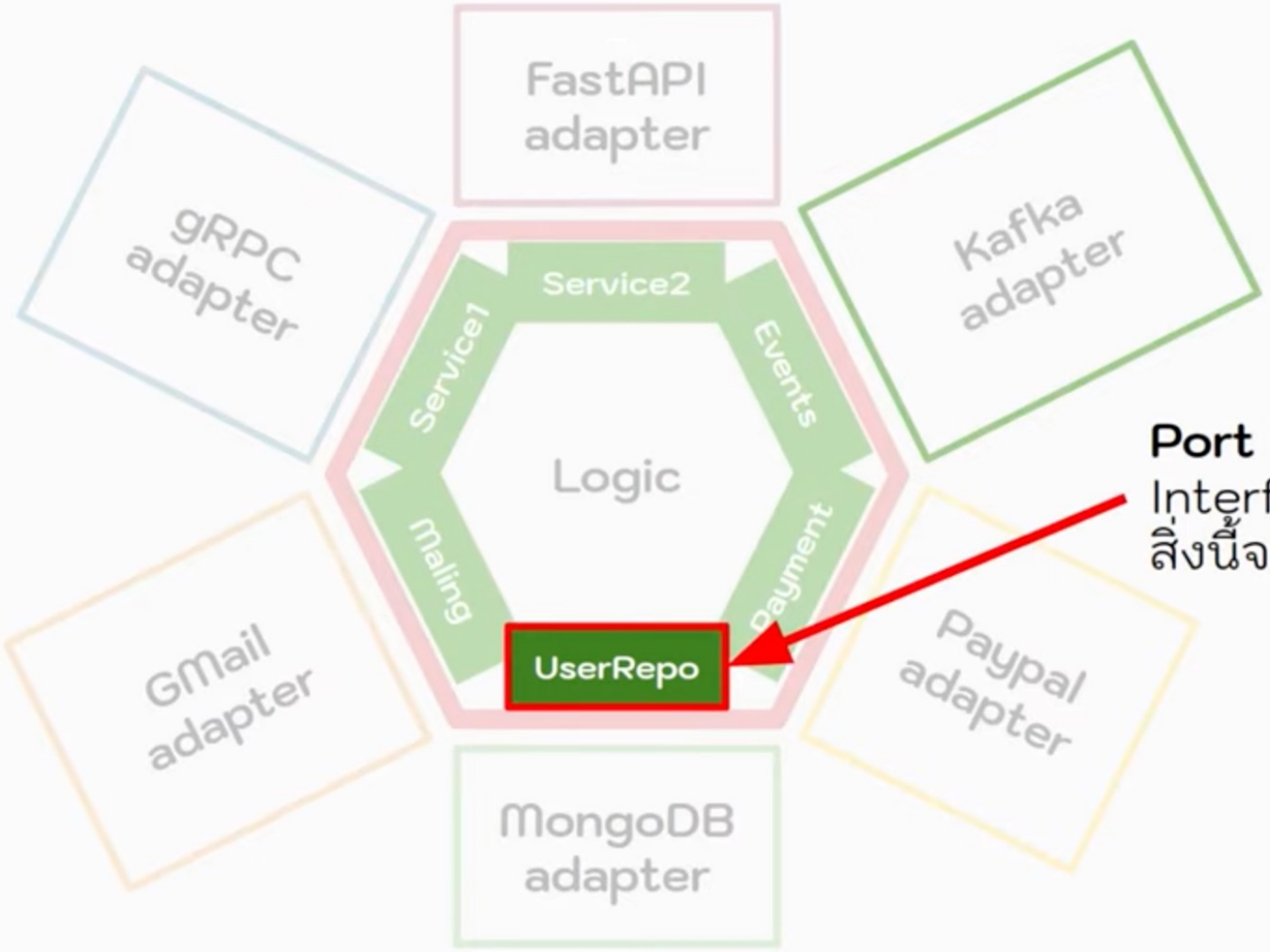
Clean Architecture

- Hexagonal Architecture



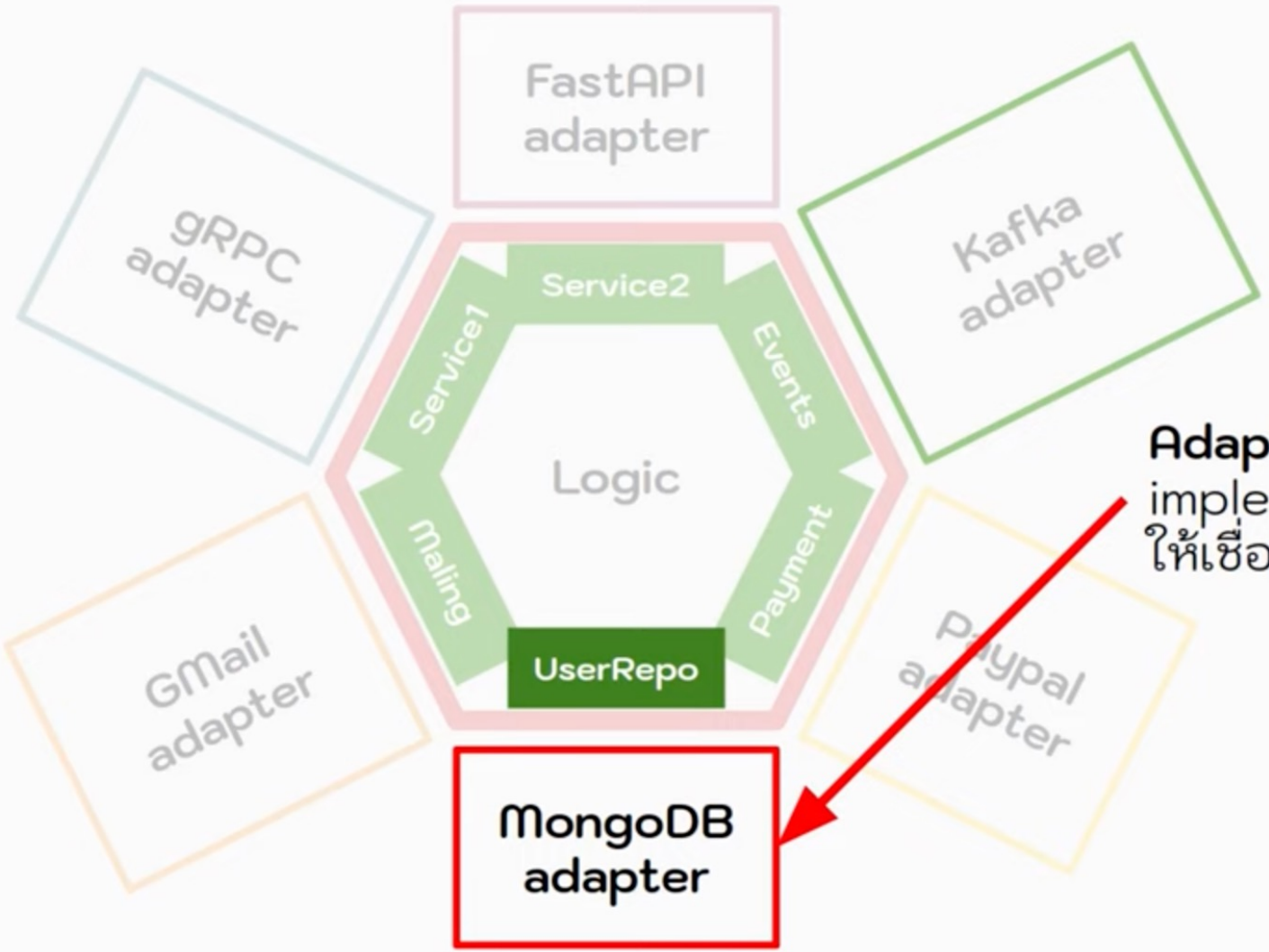

HDMI
adapter
Logic

Hexagonal Architecture



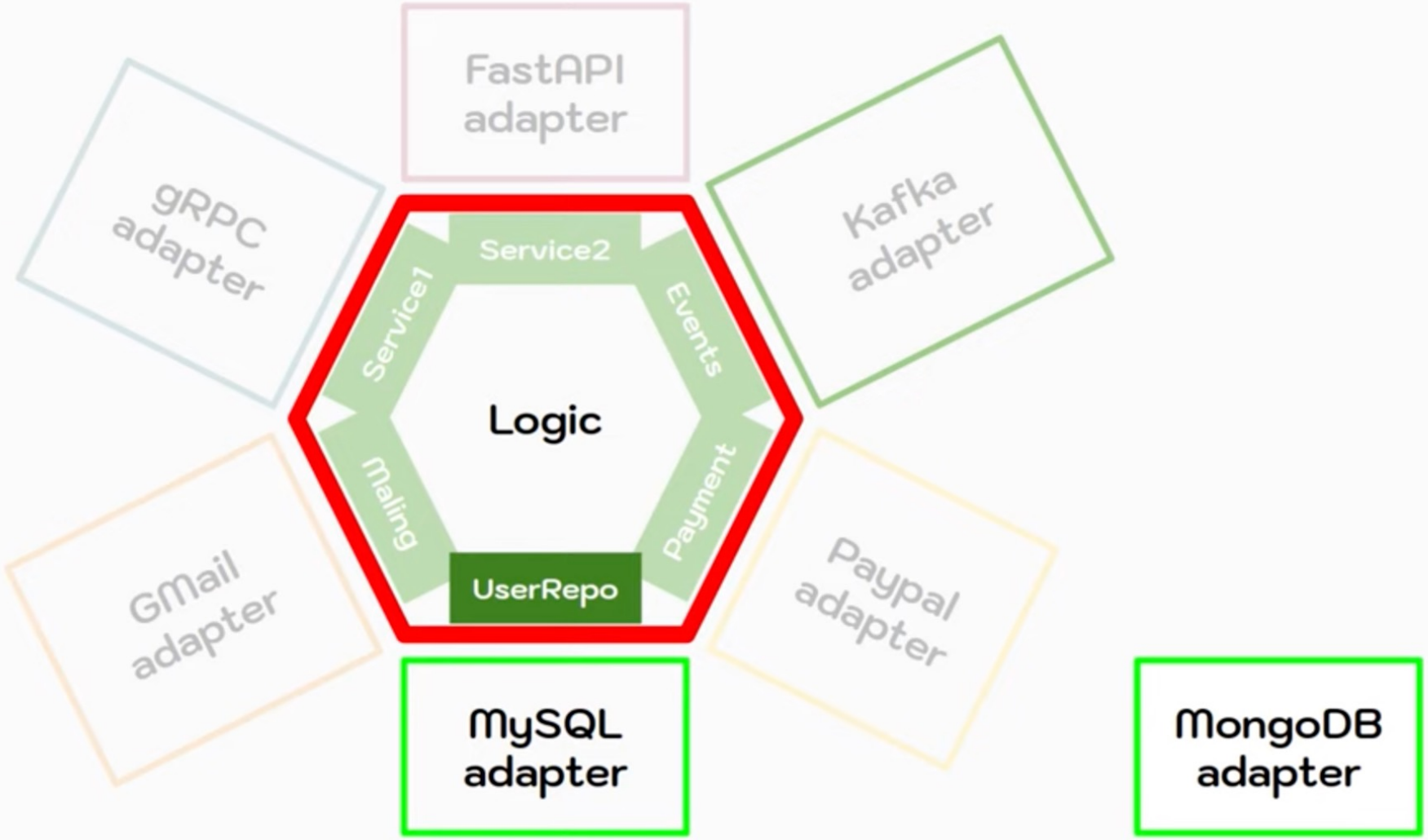
Port
Interface ที่นิยามว่าสิ่งนี้จะทำอะไรได้บ้าง

Hexagonal Architecture

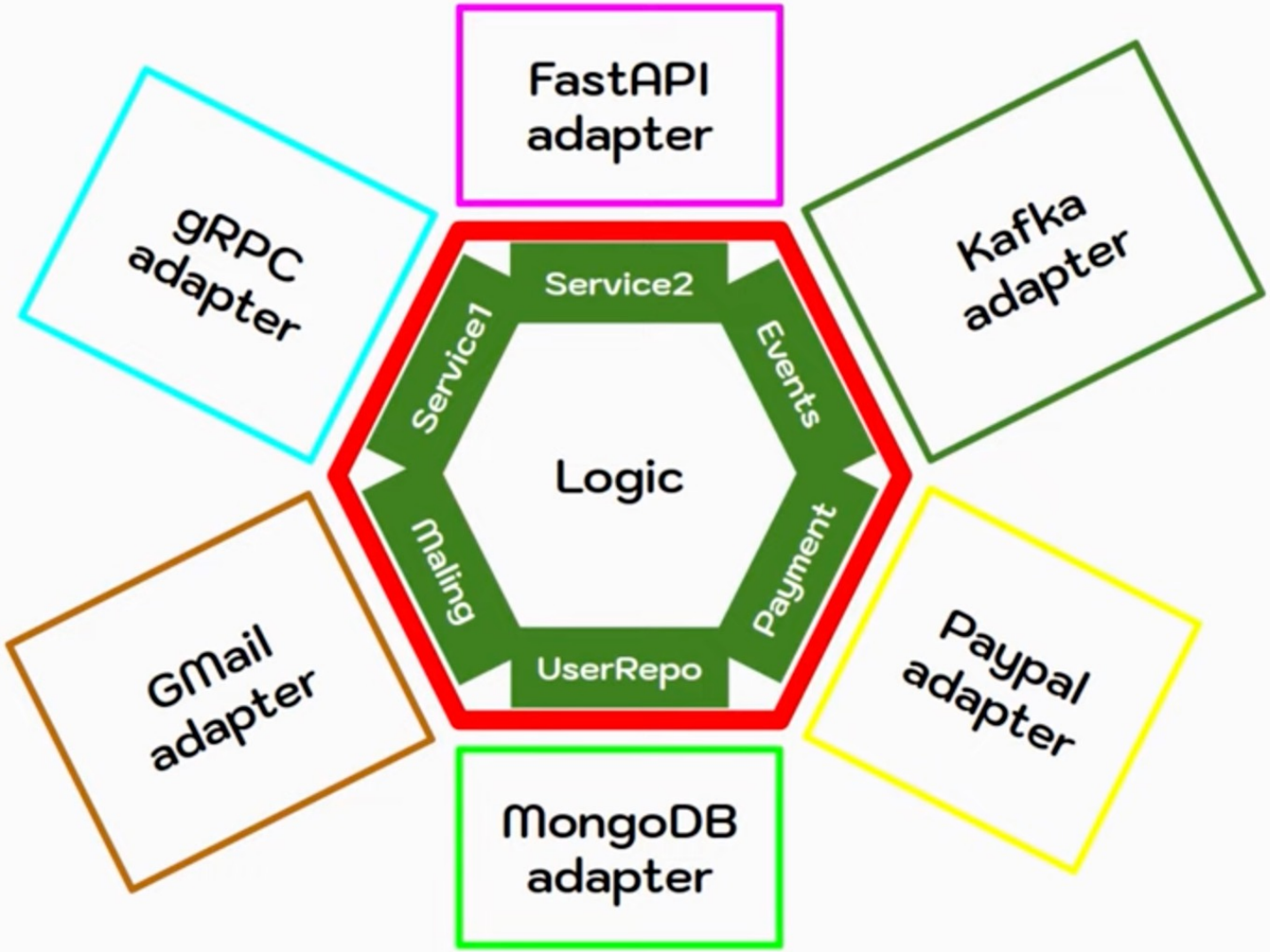


Adapter
implement จาก Port
ให้เชื่อมต่อระบบนอกได้จริง

- Hexagonal Architecture



Hexagonal Architecture

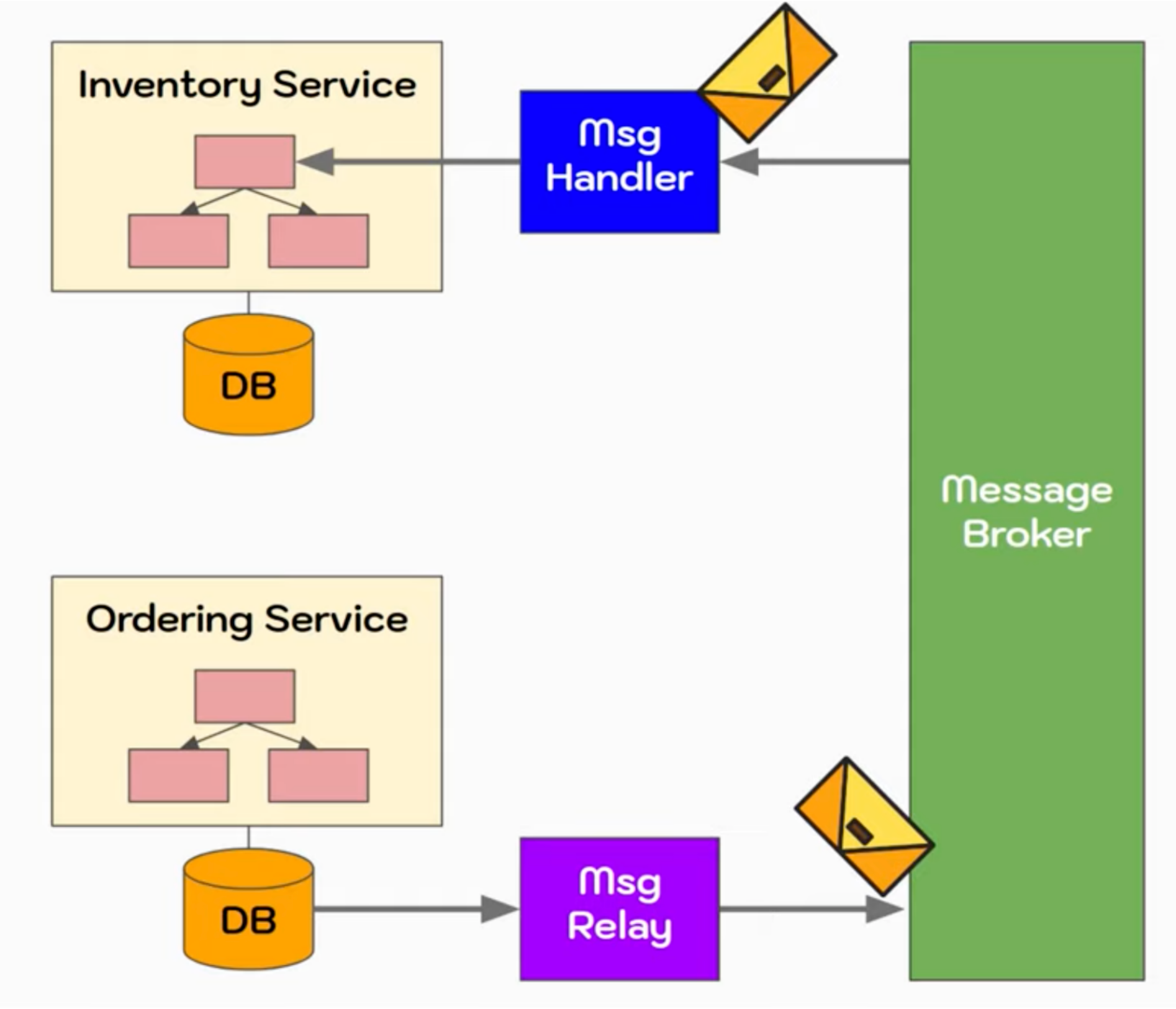


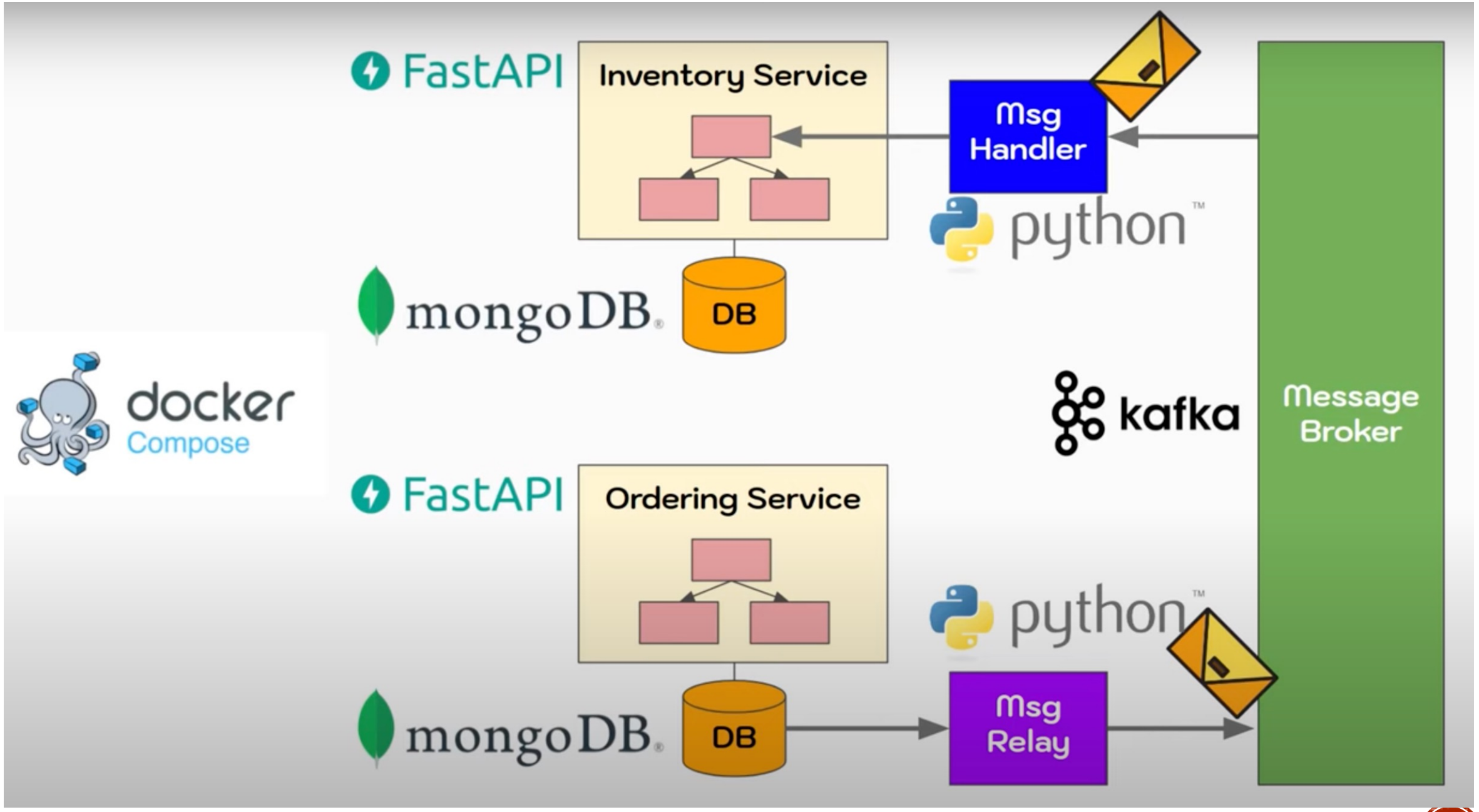
เมนูวันนี้ (3/3)

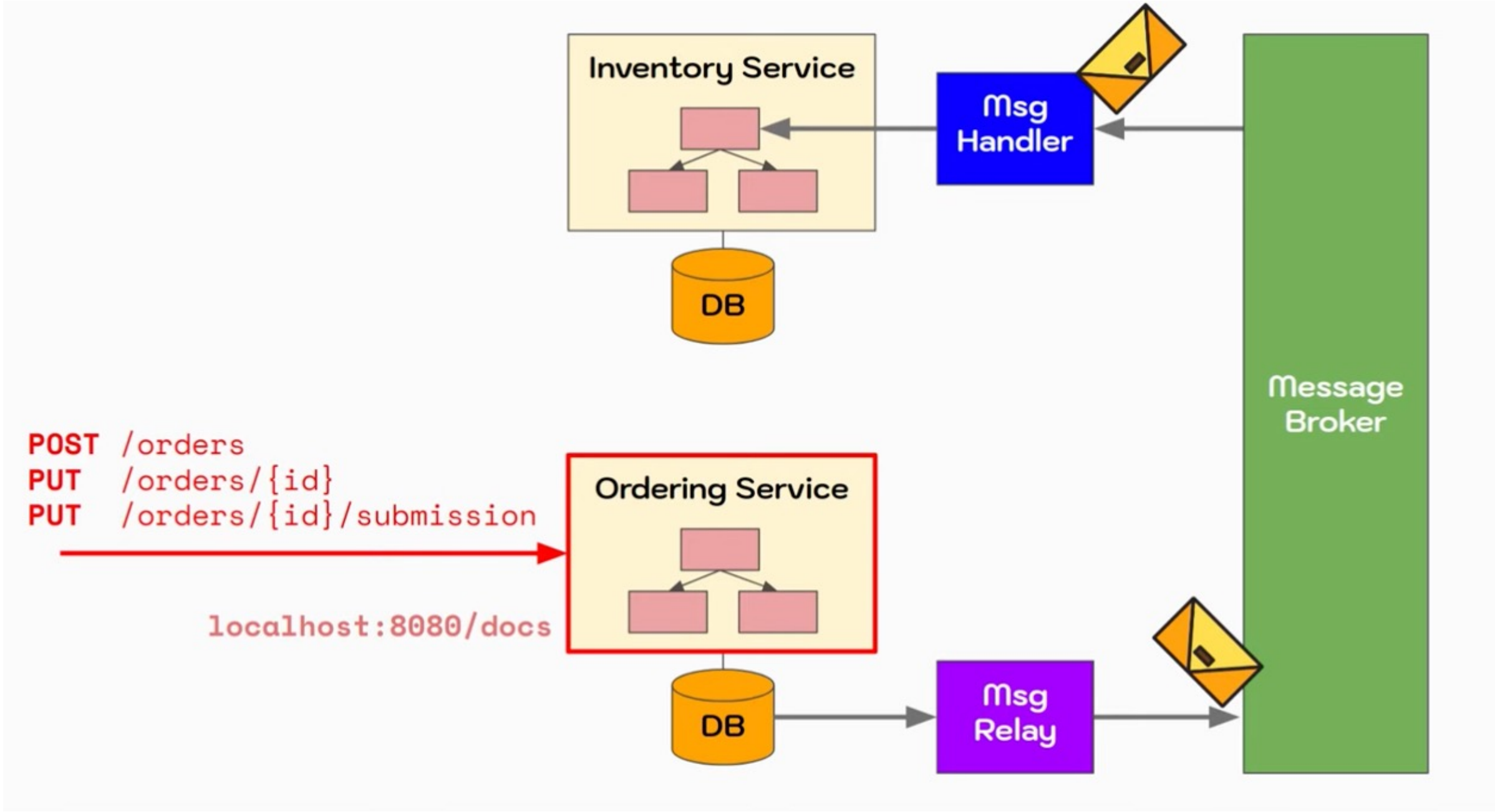
Hexagonal Architecture

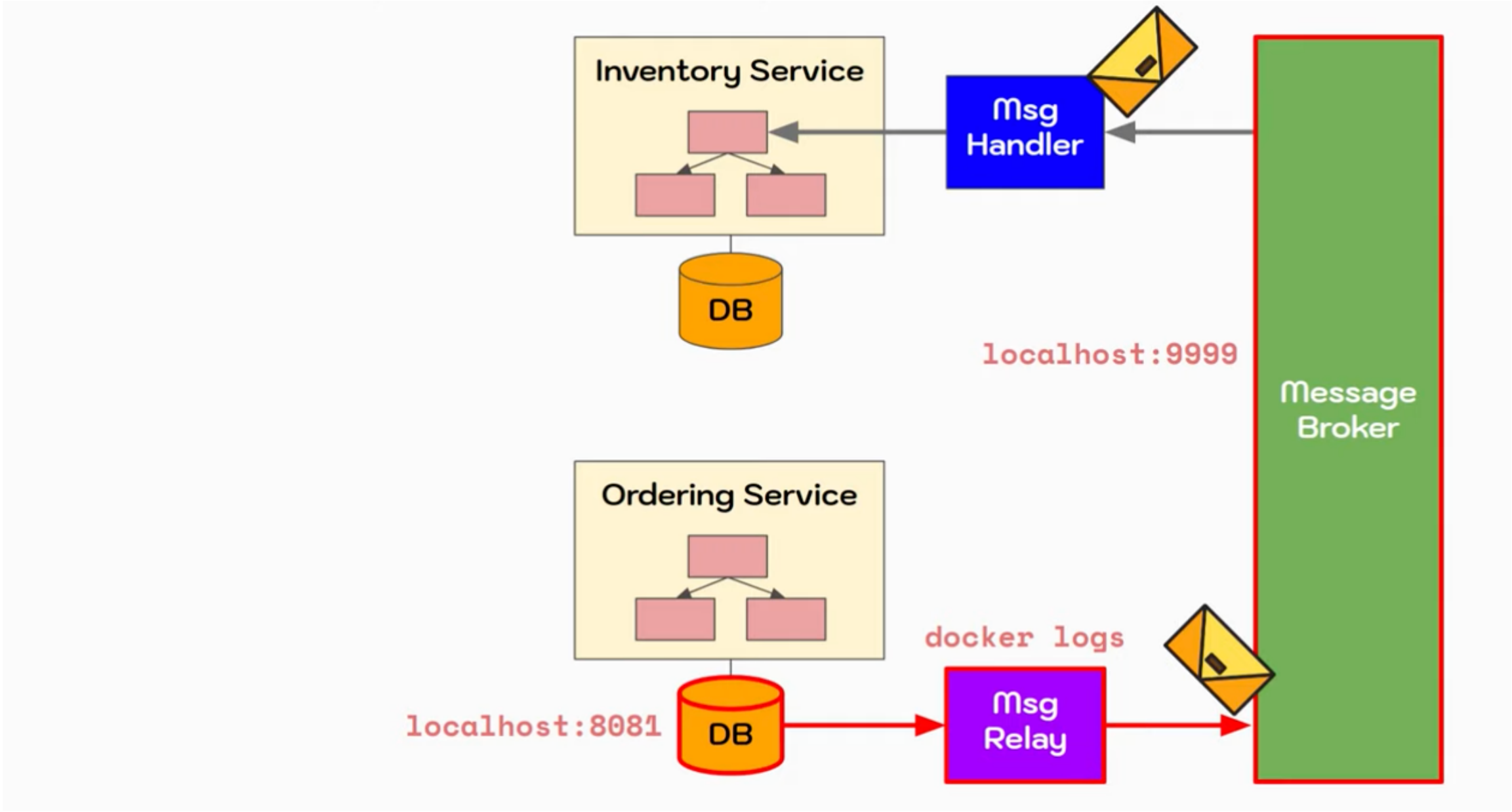
Demo + Code with Python FastAPI & Kafka & MongoDB

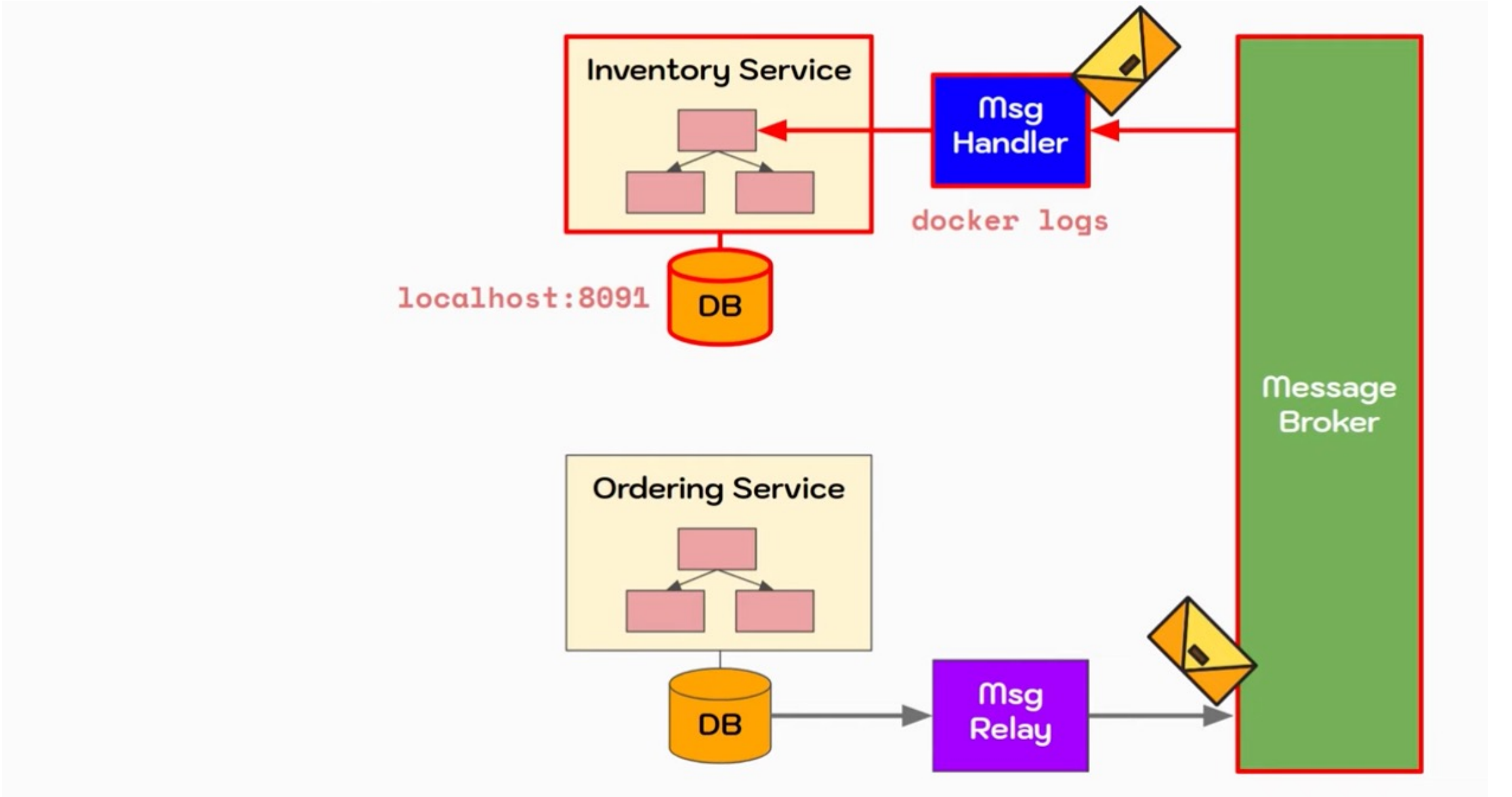
- Message Broker
- Transactional Outbox
- Repository Pattern
- Aggregate Pattern
- Hexagonal Architecture

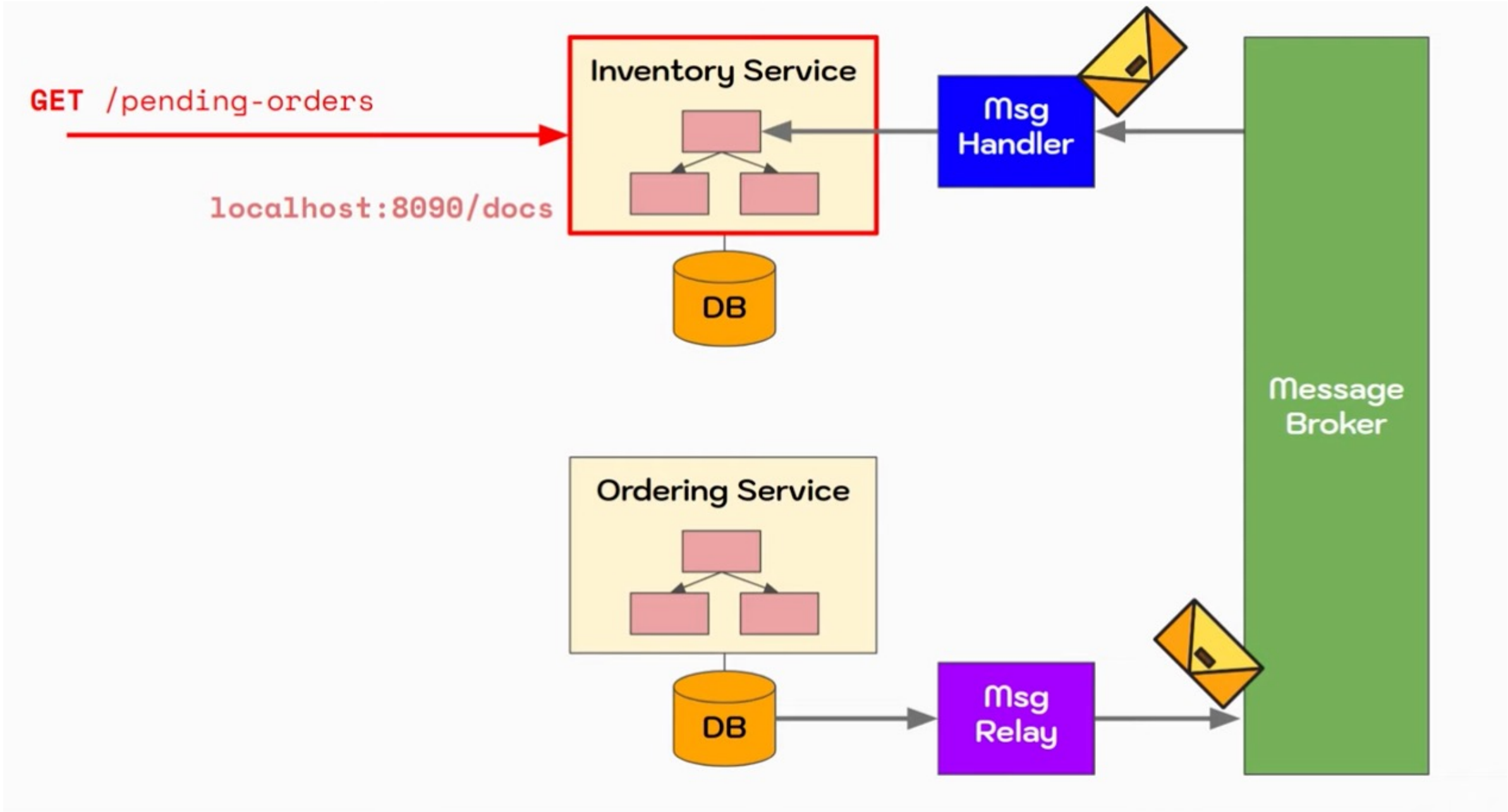

Inventory Service
DB
Msg Handler
Message Broker
Ordering Service
DB
Msg Relay


FastAPI
Inventory Service
Msg Handler
python
mongoDB
DB
docker
Compose
kafka
Message Broker
FastAPI
Ordering Service
python
mongoDB
DB
Msg Relay

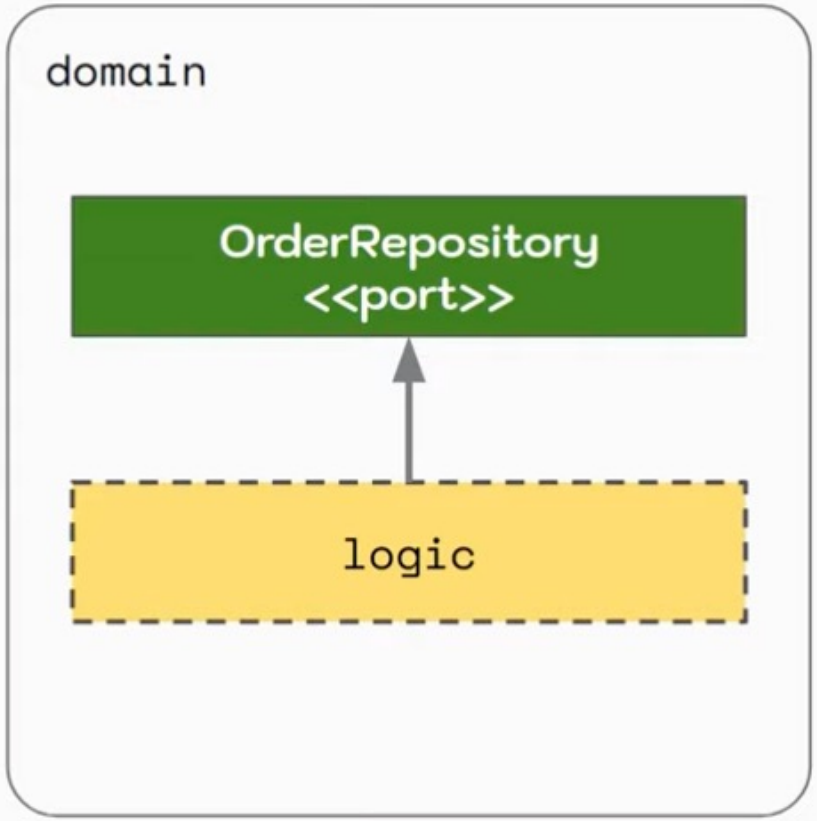

Inventory Service
DB
Msg Handler
Message Broker
POST /orders
PUT /orders/{id}
PUT /orders/{id}/submission
Ordering Service
localhost:8080/docs
DB
Msg Relay


Inventory Service
Msg Handler
DB
localhost:9999
Message Broker
Ordering Service
docker logs
localhost:8081
DB
Msg Relay

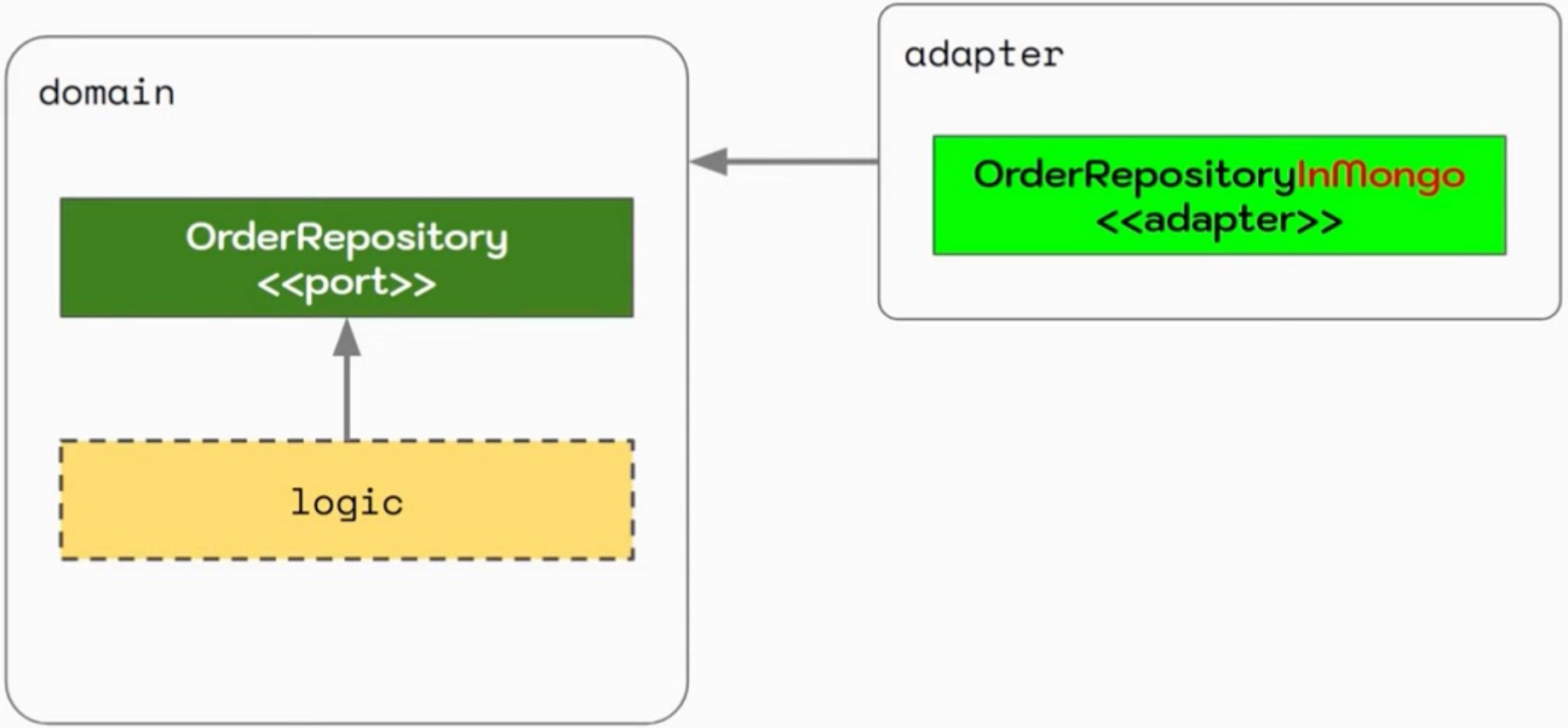

Inventory Service
Msg Handler
docker logs
localhost:8091
DB
Message Broker
Ordering Service
DB
Msg Relay


GET /pending-orders
localhost:8090/docs
Inventory Service
DB
Msg Handler
Message Broker
Ordering Service
DB
Msg Relay

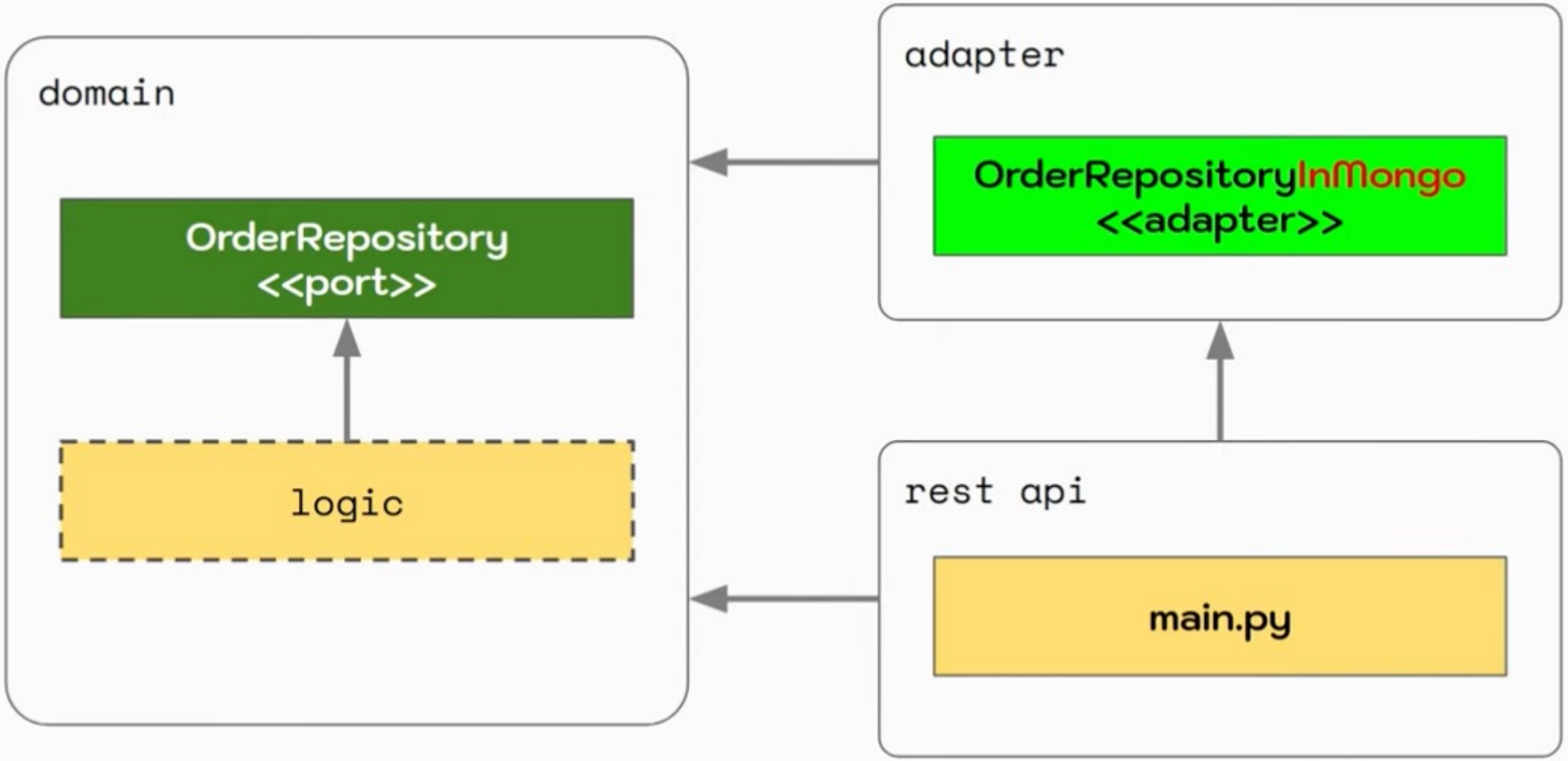
- # Hexagonal Architecture



- # Hexagonal Architecture



• Hexagonal Architecture



• Hexagonal Architecture