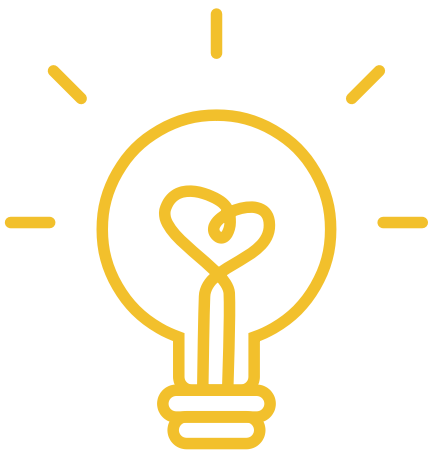
Idempotency of **commands** in distributed systems

PHPers #10 Poznań

Max Małecki 22.02.2023

Good morning!



As the **Last**

PHP developer

Bitnoise

Uniwersytet im. Adama Mickiewicza w Poznaniu

PHP

Passed LinkedIn Skill Assessment

Endorsed by Mariusz Gil and 8 others who are highly skill

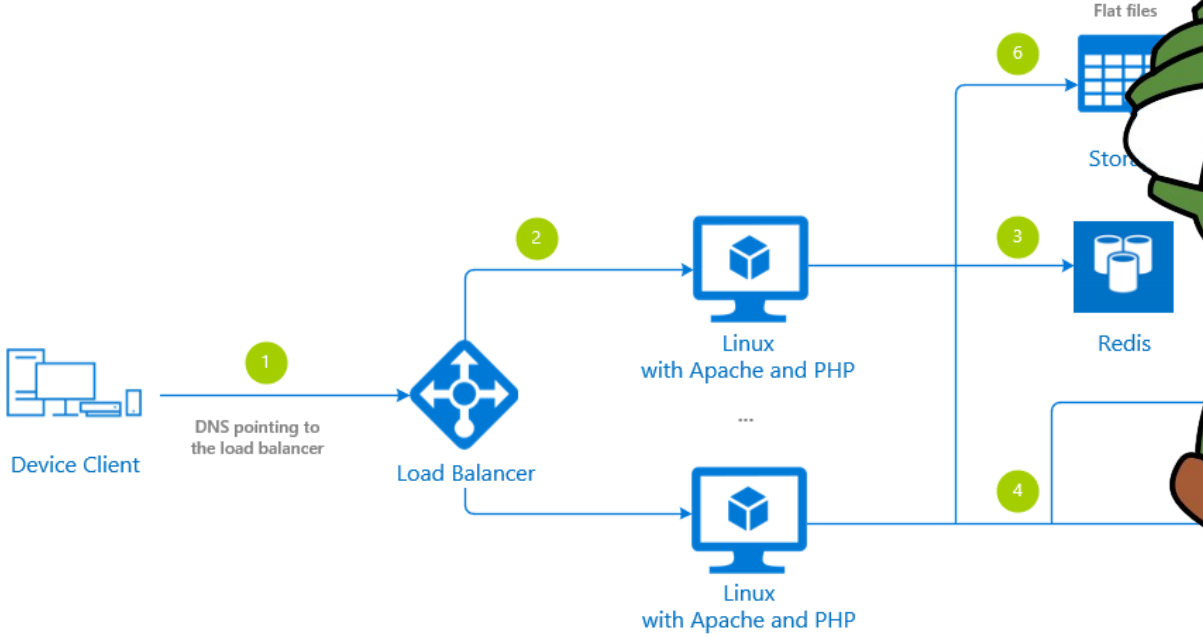
Endorsed by 2 colleagues at Bitnoise

Show all 4 details →

Warmup Questions!

Who is running microservices in production?

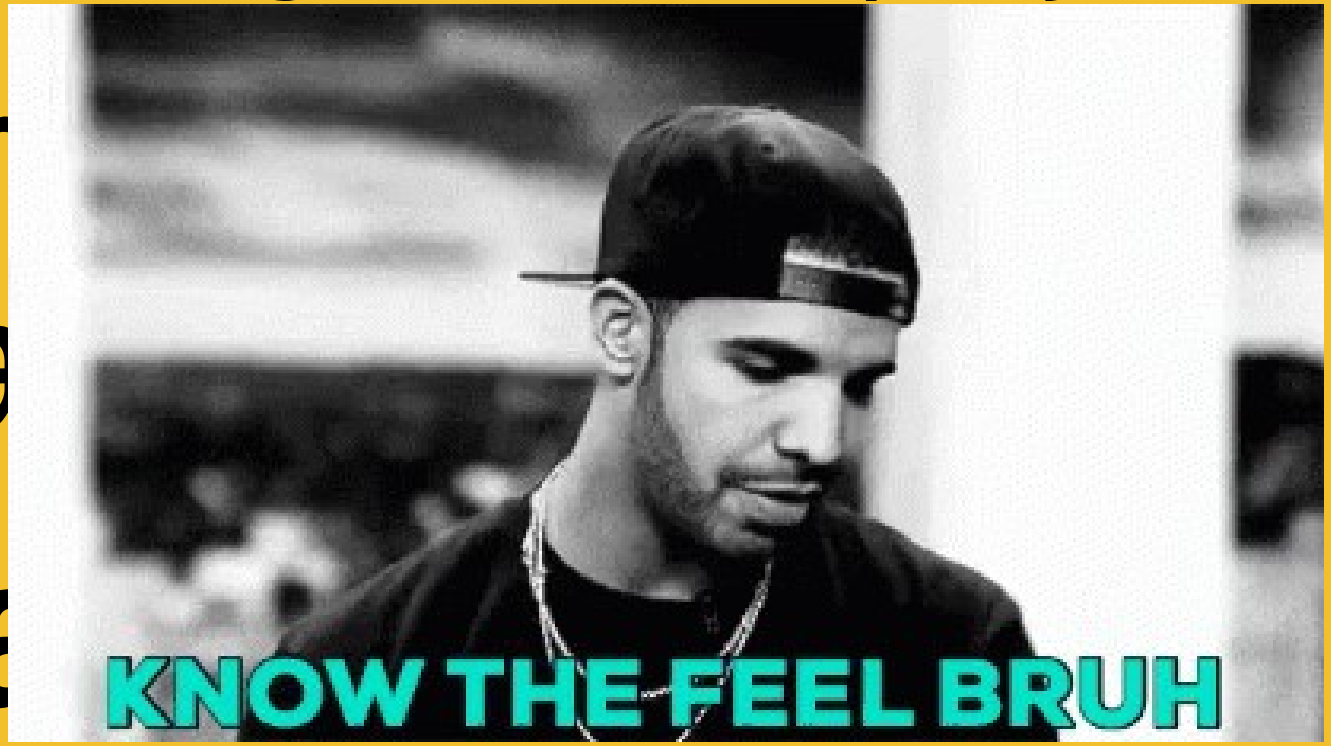
Microservices?



bro...

You got in the project:

- Pur
- Da
- Da



Did your project:

- Spams sentry with **UniqueConstraintViolations** without any reason?
- Does your endpoints often return **500** status on retry?

Idempotency of **commands** in distributed systems

PHPers #10 Poznań

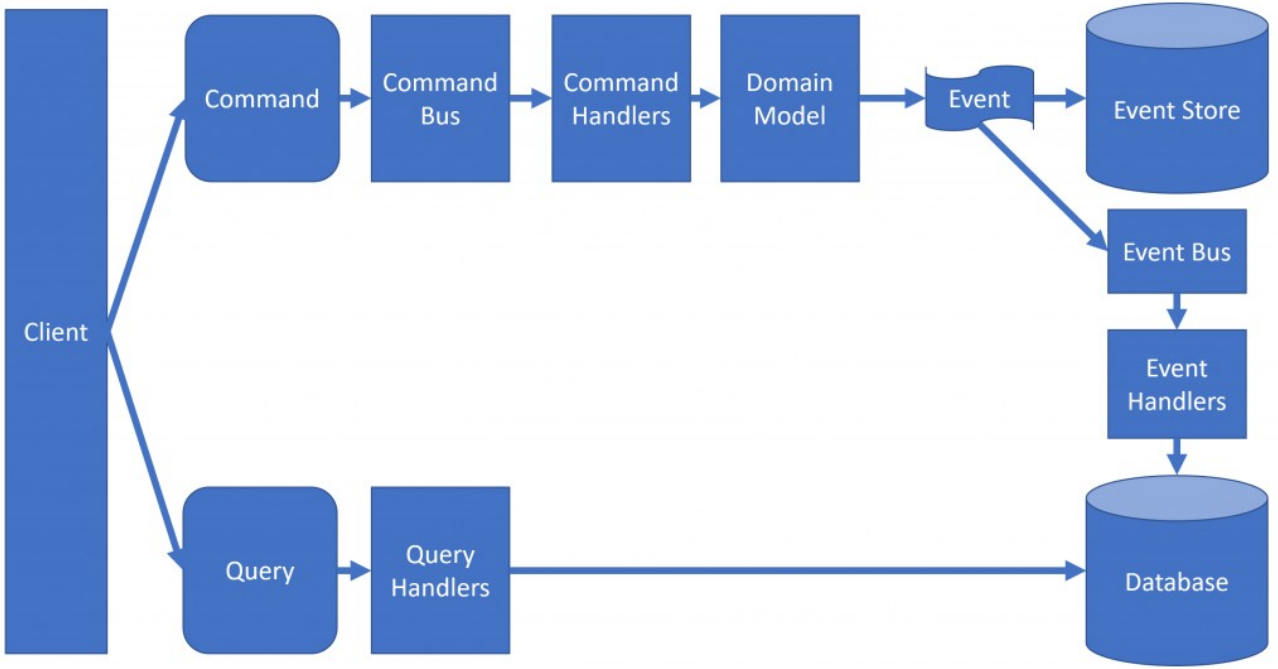
Max Małecki 22.02.2023

Let’s start from the very beginning

00 CQRS

Command Query Responsibility Segregation

CQRS



00

Command

It’s a class that represents a change to be made in the system.

00

Query

It’s a class that is responsible for data retrival.

00

Command Bus

This is a class that contains all the information about commands and handler assigned to them.

00

Command Handler

This is a class responsible for handling the command.

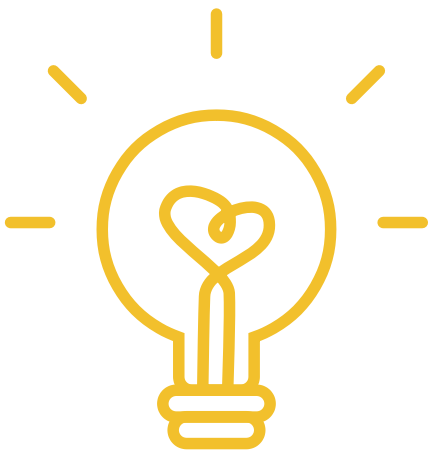
Now you can CQRS

01 Idempotence

It’s a characteristic of a single argument function that the equation below is true.

$$f\left(f\left(x\right)\right)=f\left(x\right)$$

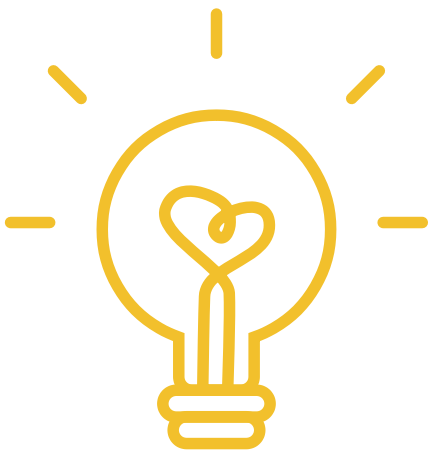
Noooo!
Not Math again



02 Idempotence

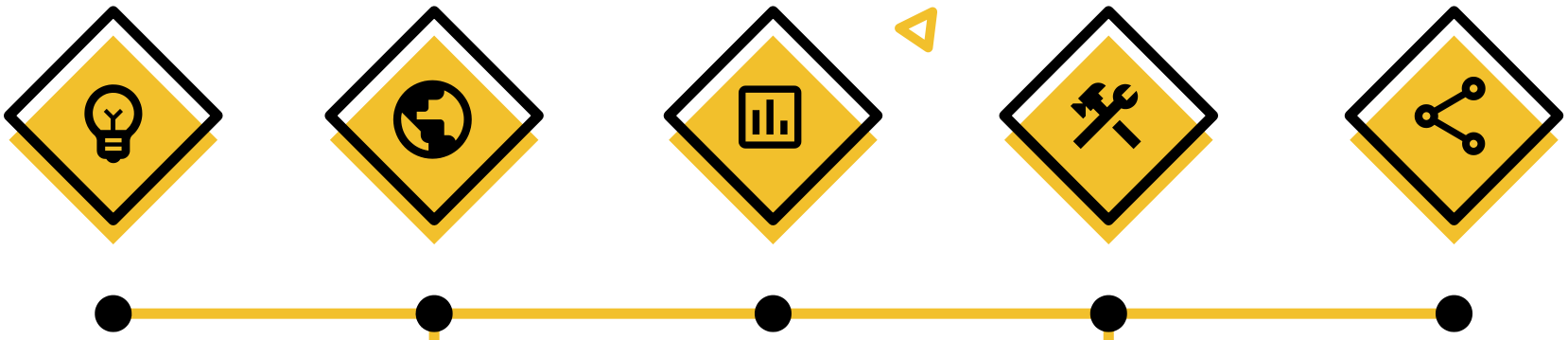
Command called that can be called multiple times and another calls doesn’t change the result of the first execution.

Life example



Black Friday

Idempotentność w Black Friday



send Request

POST /order

TIMEOUT

php-fpm don’t respond to ngnix

Retry Request

POST /order

TIMEOUT

PDO responded with timeout of transaction.

What a bargin!

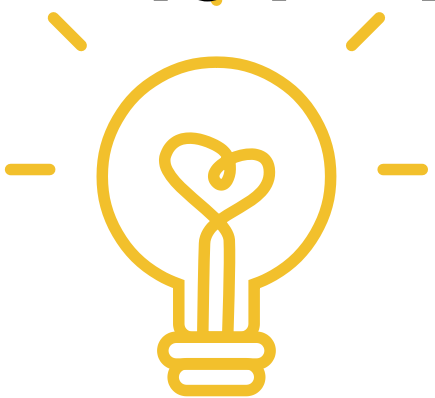
We just bought same thing twice!

And we need to pay for the return delivery.

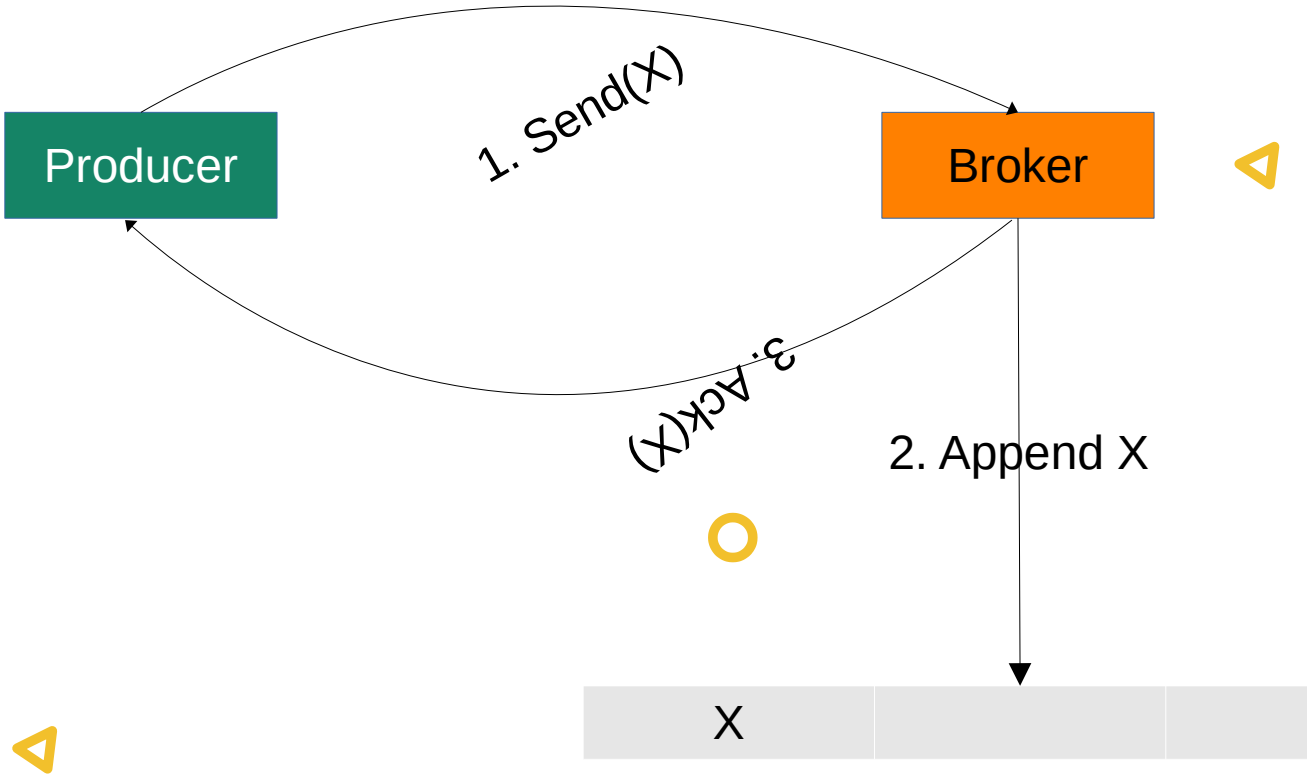

TE GRAŻYNKA POPACZ NO,
CUKIER WE PROMOCJI
WEZNE ZE 100 KILO
PACZAIZM.PL

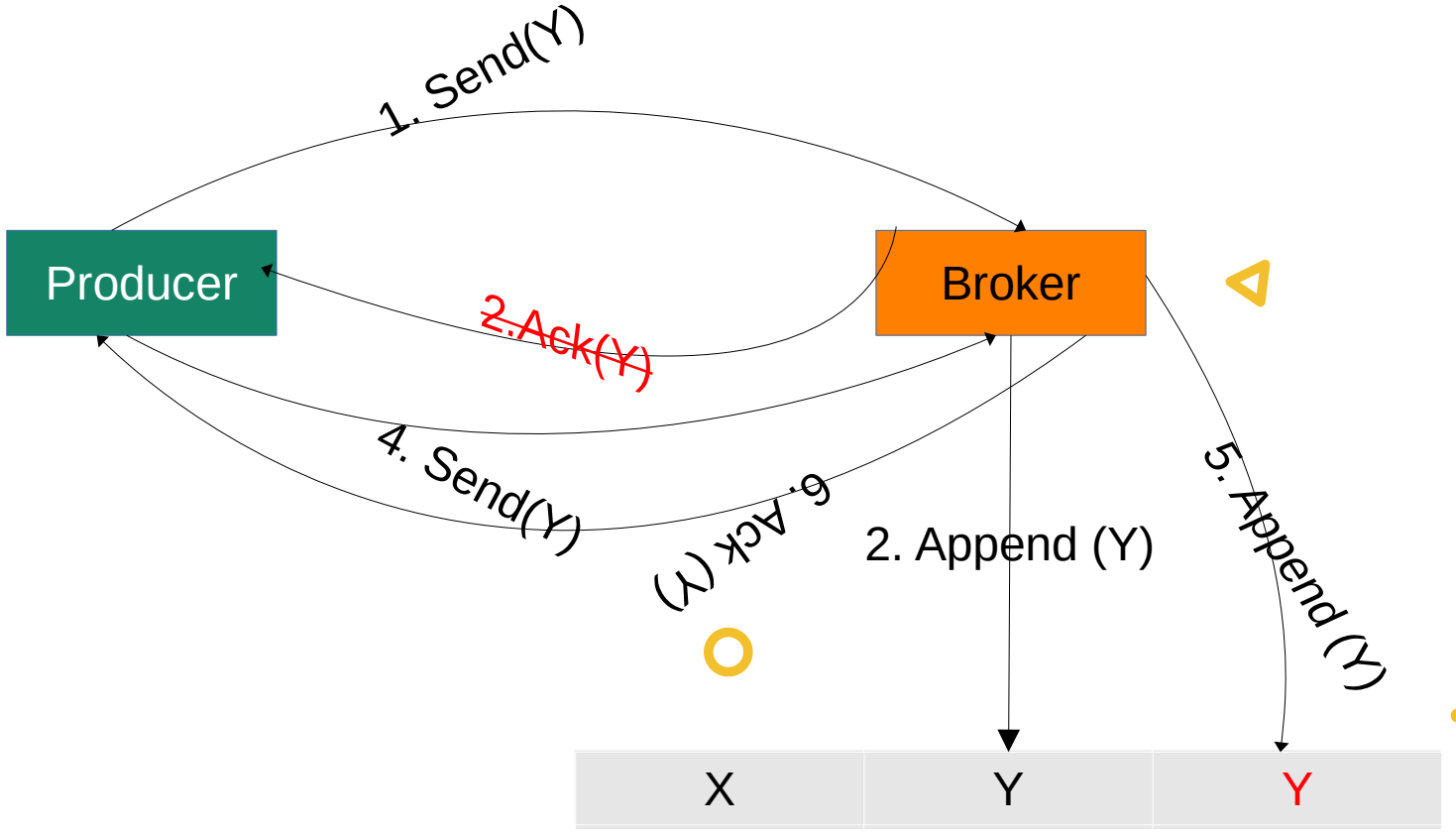
Idempotence?

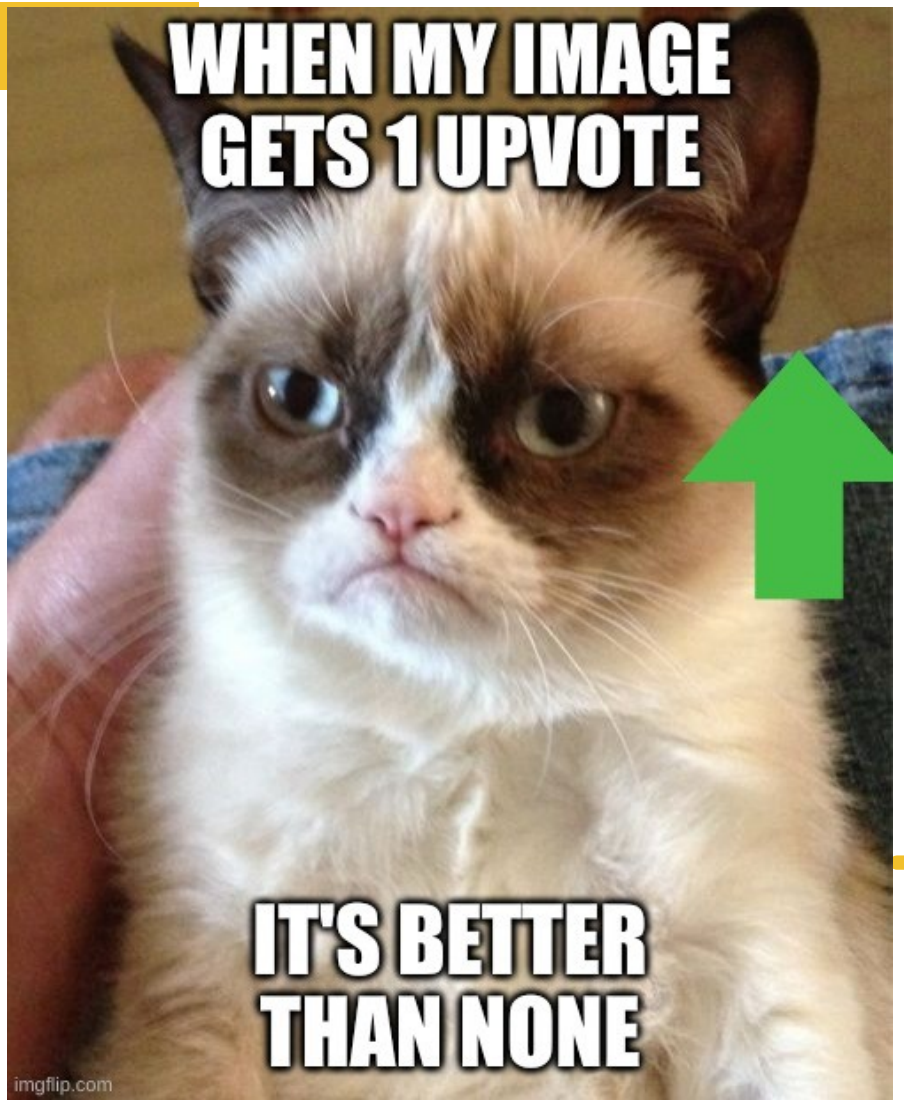
404 – Not fo



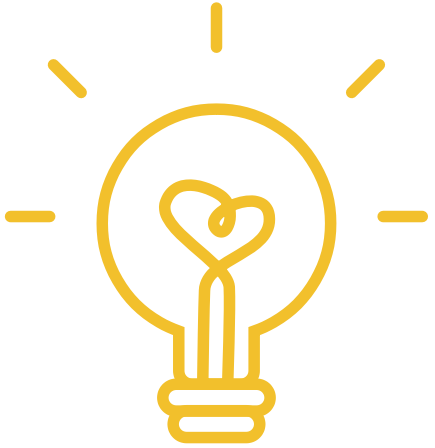
Maybe something more microserviceish?


Producer
1. Send(X)
Broker
3. Ack(X)
2. Append X
X


1. Send(Y)
Producer
Broker
2.Ack(Y)
4. Send(Y)
6. Ack (Y)
2. Append (Y)
5. Append (Y)
X
Y
Y


WHEN MY IMAGE
GETS 1 UPVOTE
IT'S BETTER
THAN NONE
imgflip.com

Yes. This producer isn’t idempotent

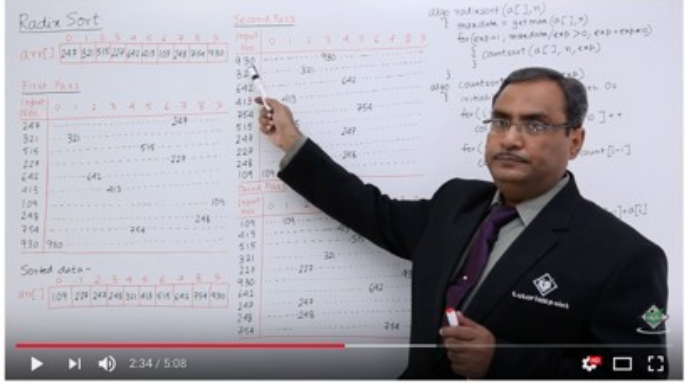


Warning!

We reach the new frontier.

frontier of youtube tutorials .

Sorts 11 Radix Sort



Radix Sort Example

04 CQRS i REST

Commands instead of verbs!


How dare you!

CQRS & REST

Controller

1) Validates the input
2) Builds a command
3) Dispatches command to Command Bus
4) Catches exceptions
5) Returns an response

Command Bus

1) Calls the handler

Handler

1) Execute the business logic for given command.
2) In case of failure it throws Exceptions
3) It triggers events

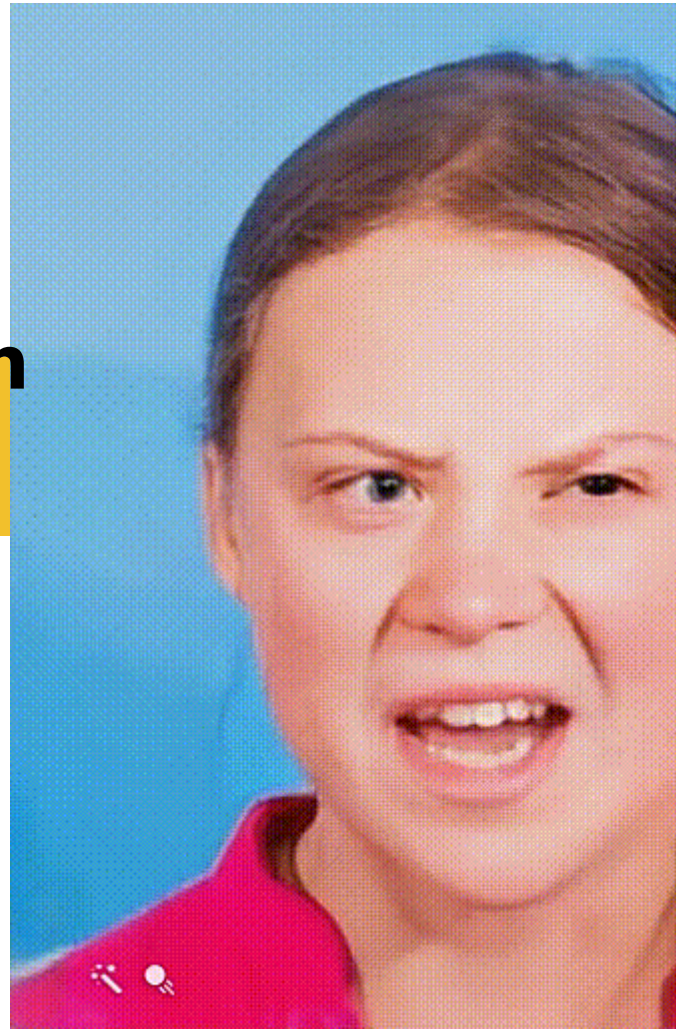
Classical REST CRUD

Idempotence

REST CRUD

HTTP	Path	Action	Command
GET	/product/{id} /products	::show($id) ::index()	
POST	/product	::new()	CreateProductCommand($name, $desc, $price, $stock, $status)
PUT	/product/{id}	::edit($id)	UpdateProductCommand($name, $desc, $price, $stock, $status)
PATCH	/product/{id}	::edit($id)	EnableProductCommand($id) DisableProductCommand($id) UpdateProductPriceCommand($id, $price)
DELETE	/product/{id}	delete($id)	RemoveProductCommand($id)

Let’s do this with Commands. Shall we?

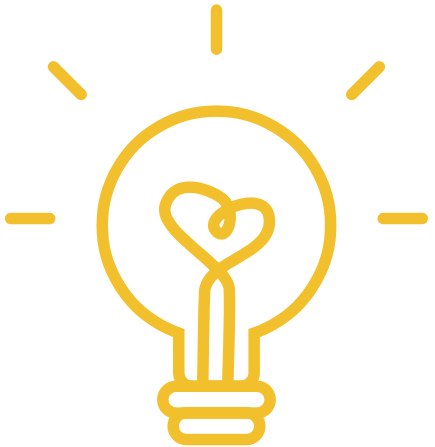


Idempotence

CQRS & REST

HTTP	Path	Command
POST	/createProduct	CreateProductCommand($name, $desc, $price, $stock, $status)
POST	/product/{id}	UpdateProductCommand($name, $desc, $price, $stock, $status)
POST	/product/{id}/enable	EnableProductCommand($id)
POST	/product/{id}/disable	DisableProductCommand($id)
POST	/product/{id}/remove	RemoveProductCommand($id)

Now we gonna take care of Idempotence



Idempotence

CQRS & REST

HTTP	Path	Command
POST	/product/create_or_update	CreateOrUpdateProductCommand($name, $desc, $price, $stock, $status)
POST	/product/{id}	UpdateProductCommand($name, $desc, $price, $stock, $status)
POST	/product/{id}/update_price	UpdateProductPriceCommand($id, $price)
POST	/product/{id}/enable	EnableProductCommand($id)
POST	/product/{id}/disable	DisableProductCommand($id)
POST	/product/{id}/remove	RemoveProductCommand($id)


Command names are better than REST verbs
CHANGE MY MIND
imgflip.com

Pros:

- While building the REST we create commands. So our commands are transport agnostic. CLI / Messages / API
- Getting rid of big fat controllers
- Controller is only for validation and response
- We can trigger REST purists

Cons:

- Don’t see any.

Cool. But where are microservices?

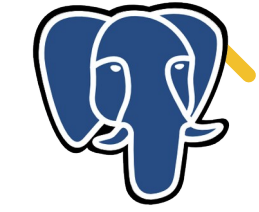
05 Message Bus / Broker

from Command Bus to Message Bus

What a name. But we are changing only

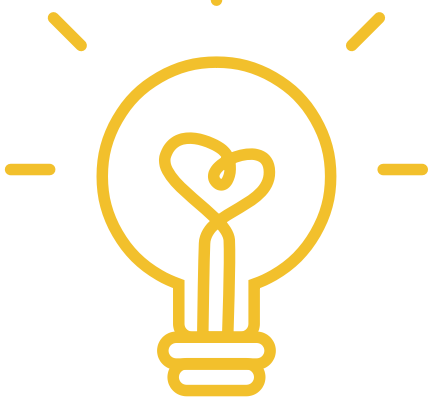
the transport.


kafka


PostgreSQL

Okay. Not only

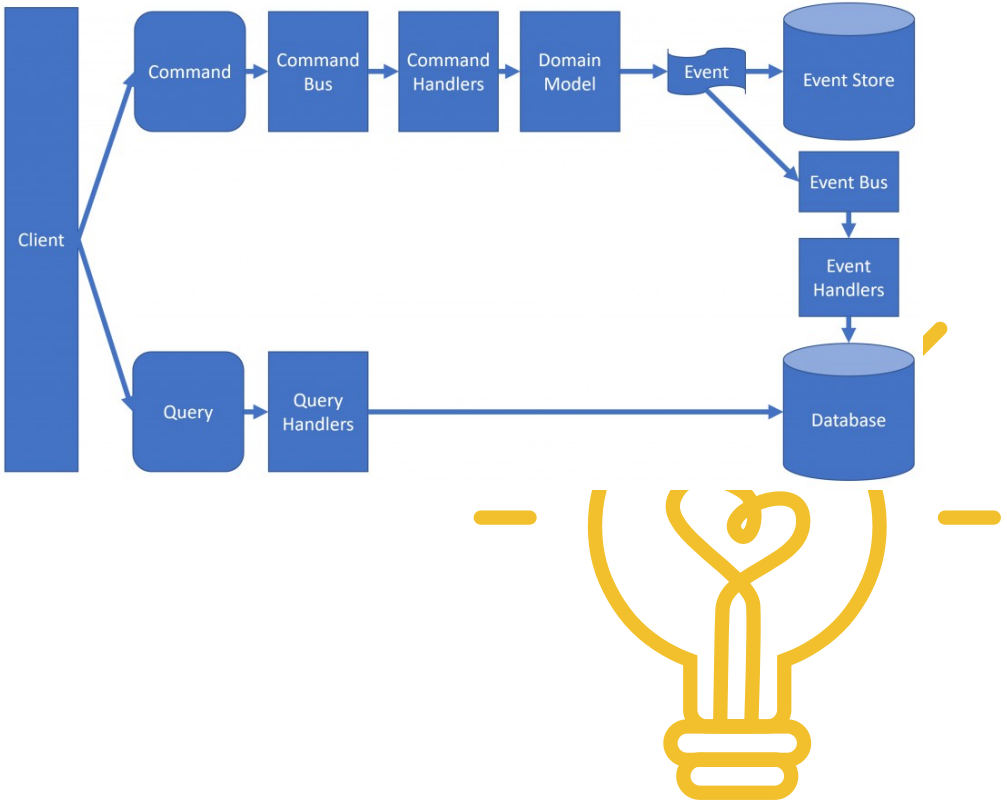
But thanks to that we can delegate workloads to other microservices



Message Bus

Communication between microservices

1) We’re adding another interface in app besides CLI i HTTP
2) Thanks to that our command are not limited by HTTP that adds 1 second for TCP frame
3) We can copy-paste message contract from our api documentation
4) Our commands start to be asynchronous



Let’s add some events!

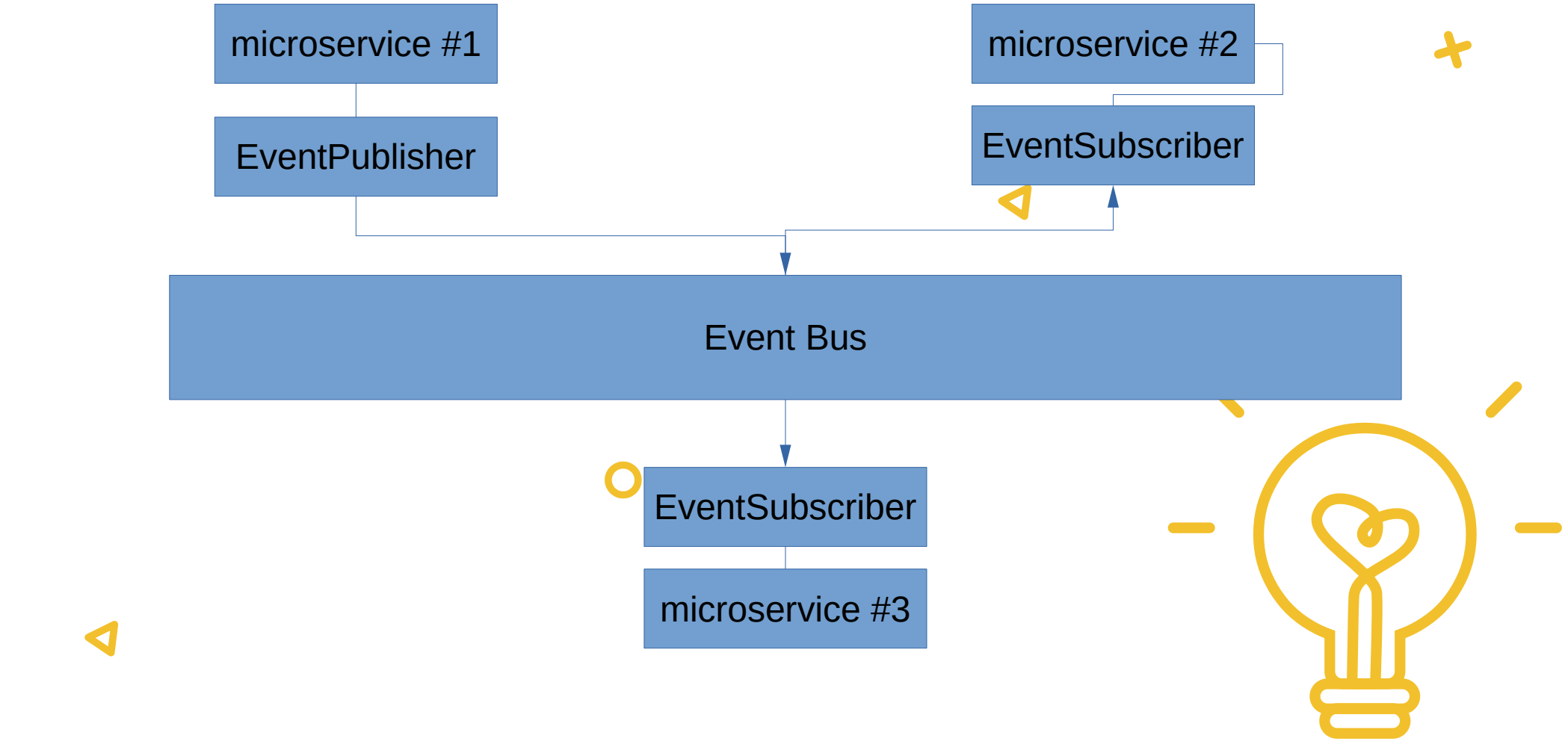
06 Event Driven Architecture

In the way to event bus

Event Driven Architecture

- In our context:
  - Every Command Handler emits an Events.
  - Events can be local – and go to the event store
  - Events can be global – and they go to the EventBus
  - Global Event should have contract
  - Global Event can be listened by Event Subscribers from many different micro-services in our system.

Event Bus


microservice #1
EventPublisher
microservice #2
EventSubscriber
Event Bus
EventSubscriber
microservice #3

Event Handler

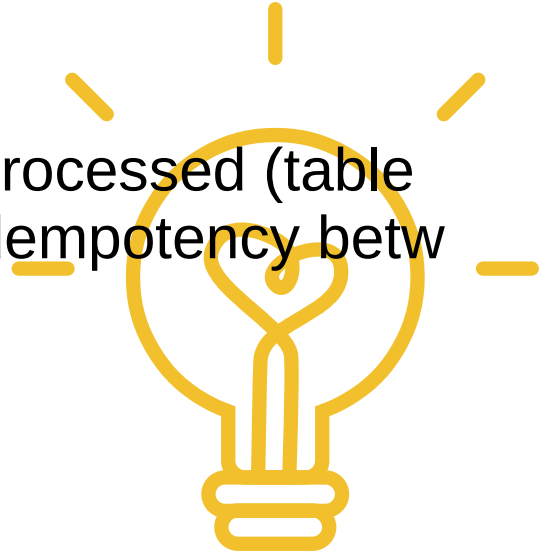
- Can listen to multiple events
- Can dispatch a command on command bus
- Can’t emit another event.

07 Indepotent Consumer Pattern

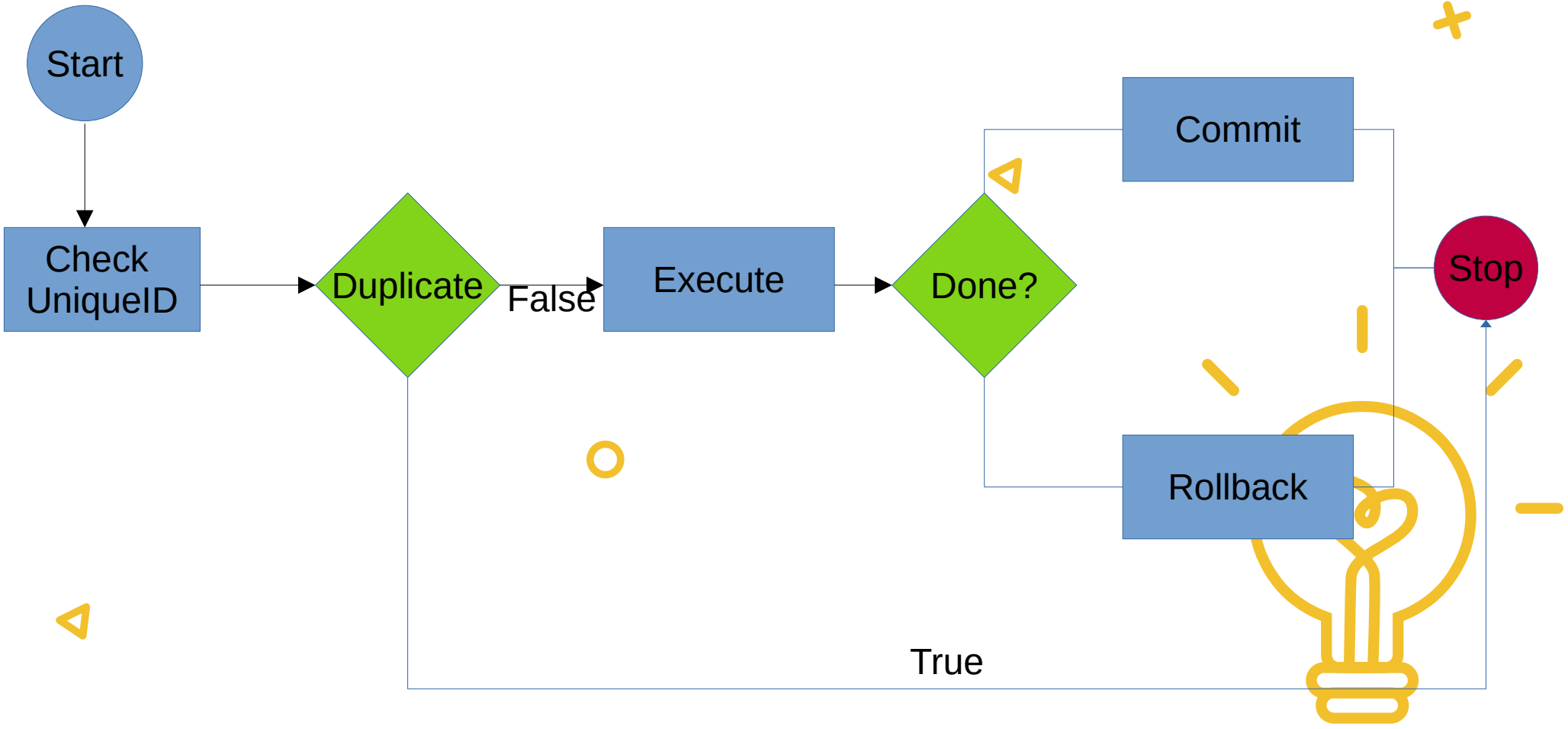
Yes it exists

Idempotent Consumer

- Why is it special:
  - It works like every consumer.
  - It deduplicates Messages and Events.
- What it needs?
  - Unique id for each processed message
  - Repository of a message ids that been already processed (table in db, index in redis/memchache), to keep the idempotency betw



Idempotent Consumer



Indepotent Consumer

- Distributed system are using message brookers, who try to deliver message **at least once**.
- Consumer can recognize duplicates and process a message just once skipping detected duplicates.
- When you **scale your consumers**, you must remember that duplicate **storage must be shared**.

08 Summary

Idempontency Pros

- We’re increasing system resilience
  - Just sopped spamming errors in the logs
  - In case of fire we allow ourselves to retry request/command
- Parallel processing is possible now!
  - We can accept same event/message from multiple instances
  - But remember it isn’t a cure for every consumer race condition.
- Finally we take care of data consistency
  - Each request (no matter it’s quantity) only represent single entity/resource in the system

09 What Next?

What next?

- You may check how Apache Kafka is dealing with idempotency in consumers and producers
- Check in your projects how many times you implement idempotent commands without realizing they are idempotent

10 Questions?

Yes this presentation will be available online.

Yes the code example is on mine github.

https://github.com/emgiezet/phpers10

THANK YOU

maxmalecki.com
twitter: @mgz
github: @emgiezet