Events

Memi Lavi
www.memilavi.com

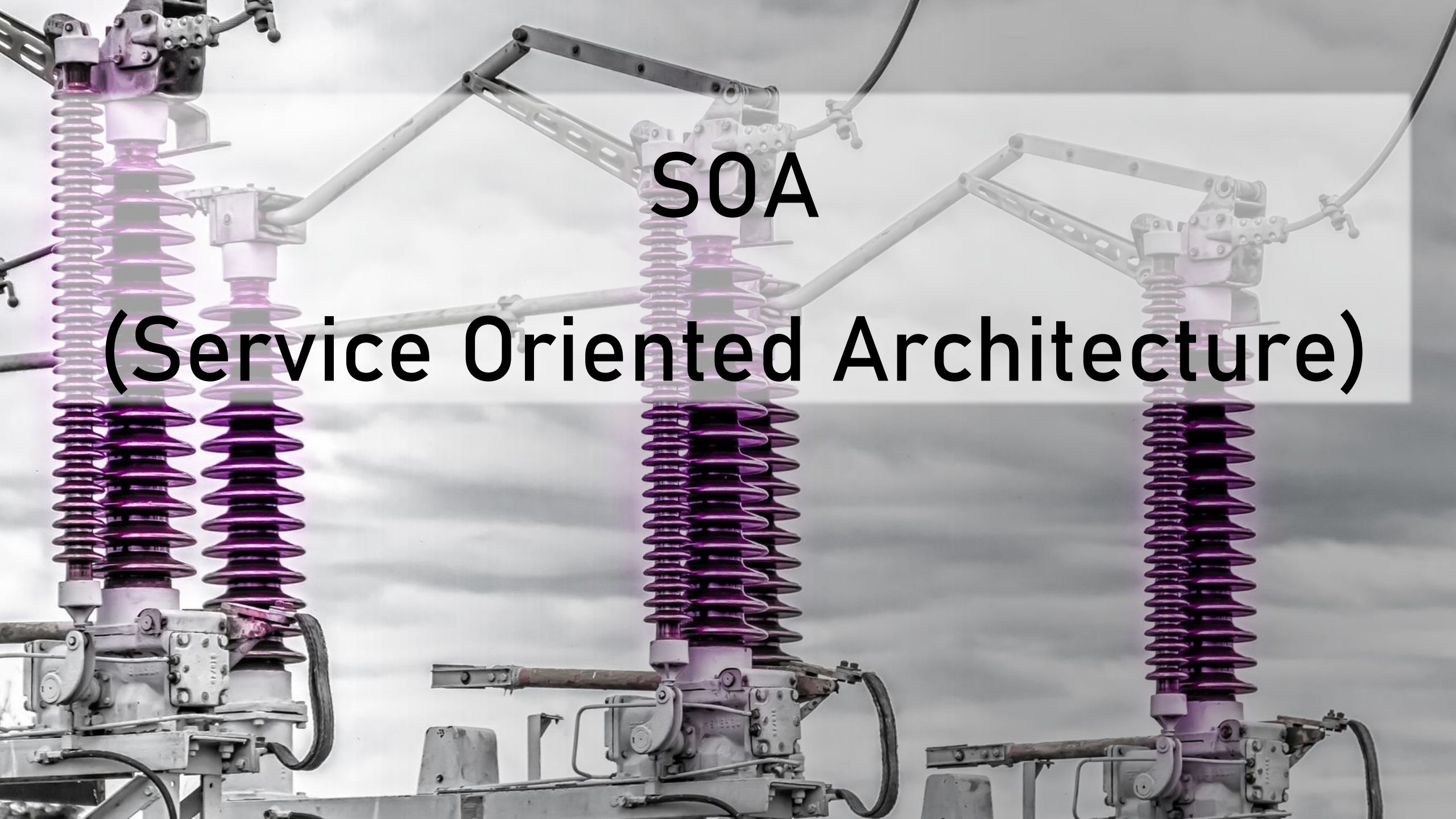
Events

- The cornerstone of event driven architecture
- Require a well-defined definition
- Evolved from other architectures

Microservices Architecture

- Based on loosely-coupled services
- Each service in its own process
- Lightweight communication protocols
- Polyglot
  - No platform dependency between services
- Replaces two legacy architectures

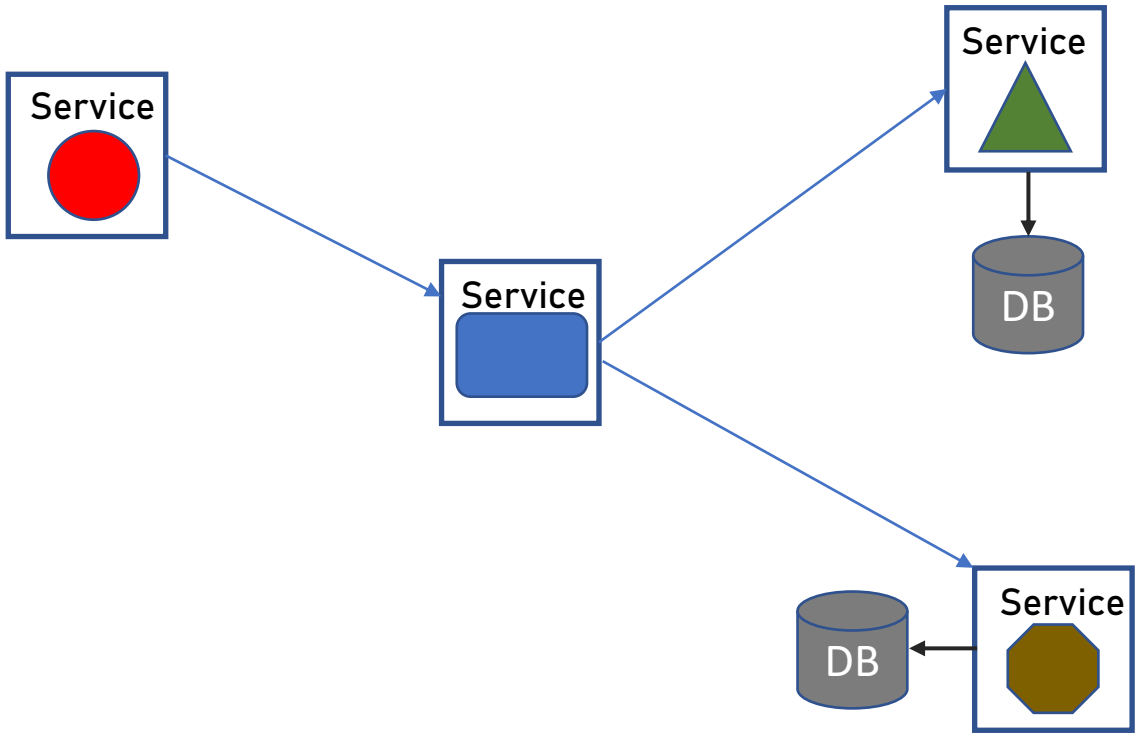

Monolith



SOA

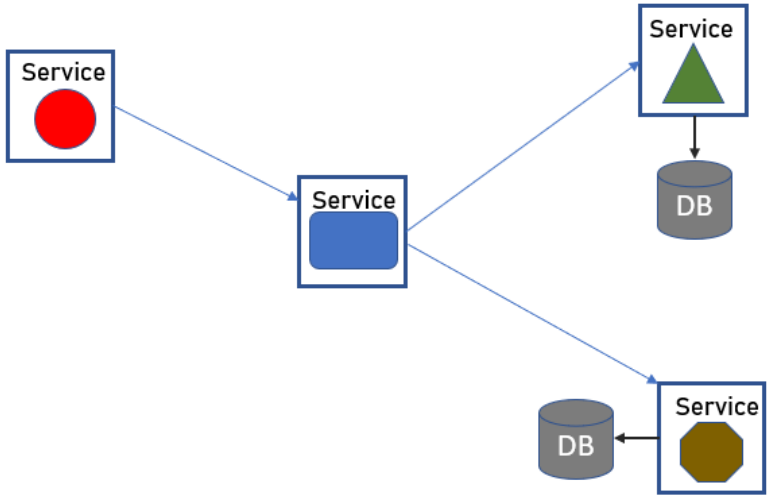
(Service Oriented Architecture)

Typical Microservices System



Microservices Communication

- Perhaps the most important part in microservices architecture
- Dictates performance, scalability, implementation and more
- Event Driven Architecture handles the communication part

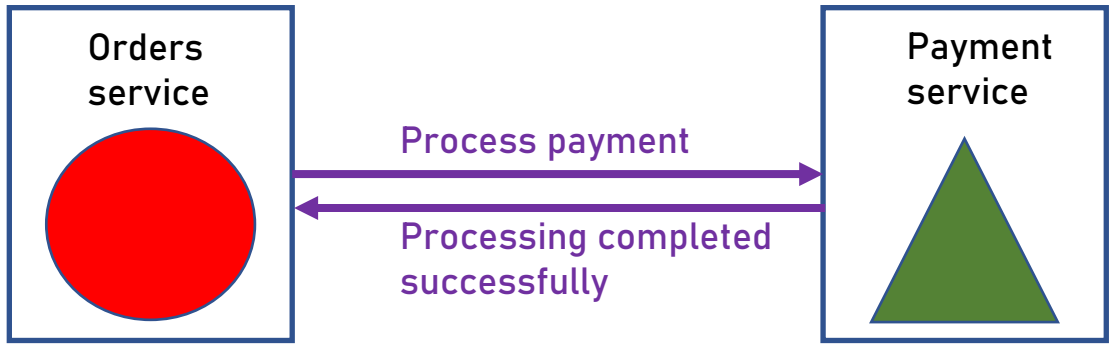


Command and Query

- The classic communication between services
- Services either:
  - Send command
  - Query for data

Command

- Service asks another service to do something



- There might be a response to the command, usually a success or failure indicator

Query

- Service asks another service for data

- There's always a response to the query, containing the data

Command and Query

- Main characteristics:

Command

- Do something
- Usually synchronous
- Sometimes returns a response
- Calling service needs to know who handles the command

Query

- Retrieve data
- Almost always synchronous
- Always returns a response
- Calling service needs to know who handles the query

Problems with Command and Query

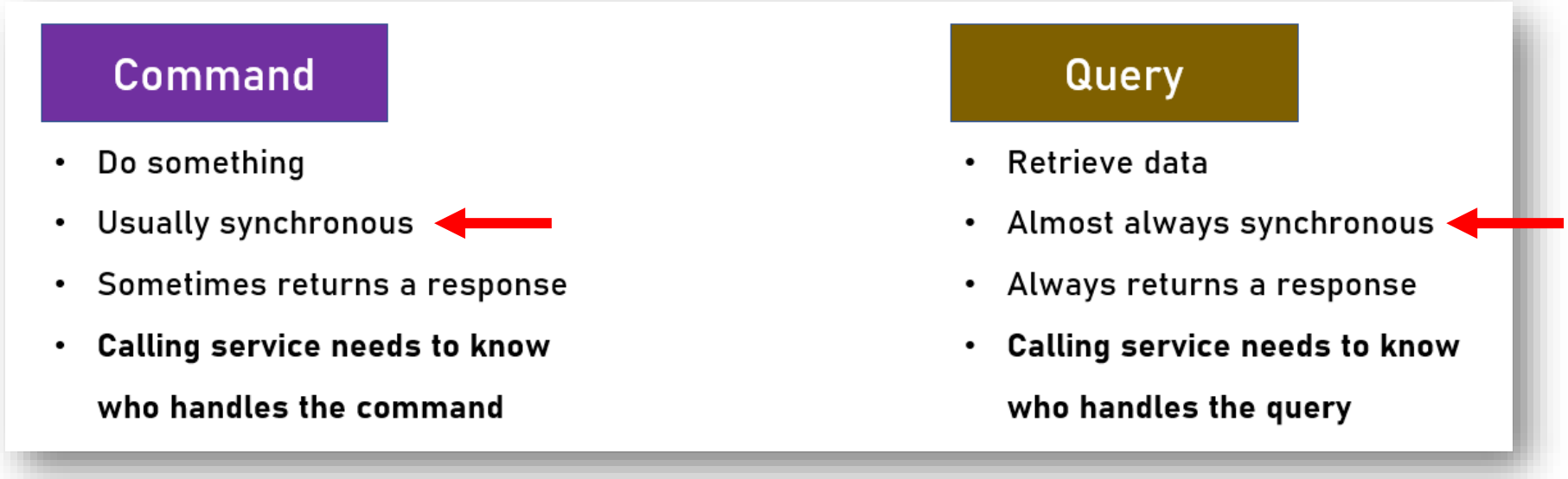
- Three major problems with command and query:

Performance

Coupling

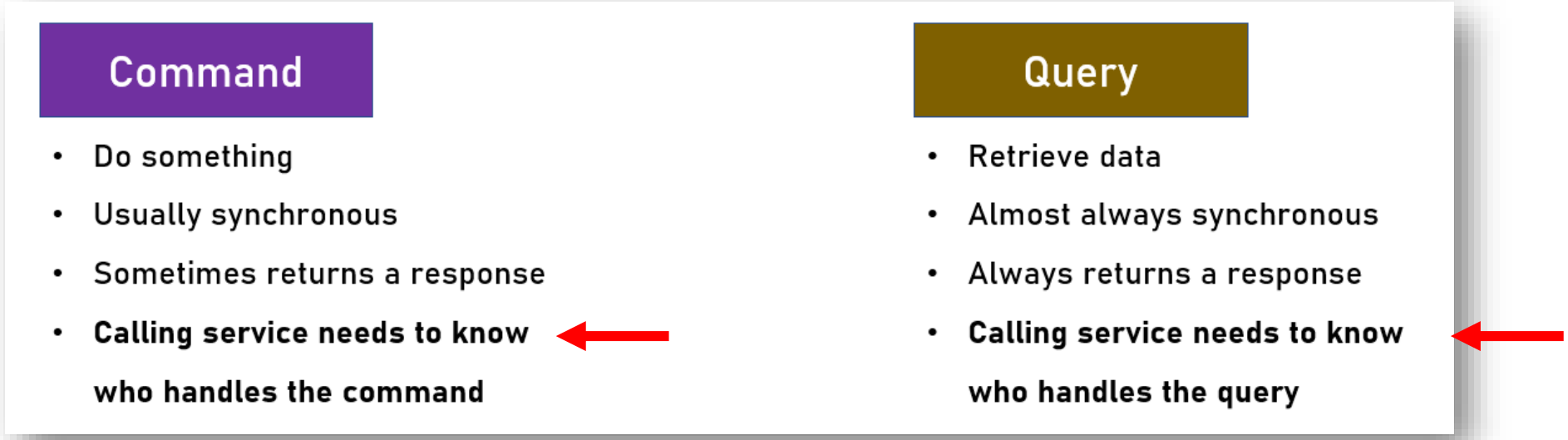
Scalability

Performance



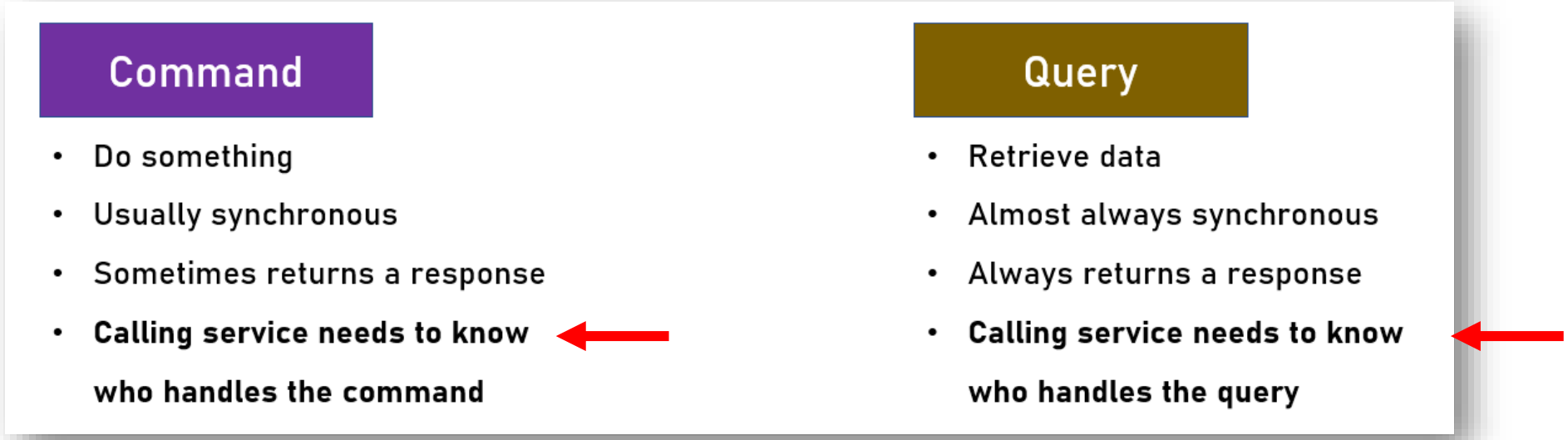
- Synchronous = the calling service waits for the command / query to complete
- Potential for performance hit

Coupling



- The calling service calls a specific service
- If the called service changes – the calling service has to change too
- More work, more maintenance

Scalability



- The calling service calls a single instance of a service
- If this instance is busy – there's a performance hit
- Adding another instances is possible, but difficult
  - Add load balancer, configure probes etc.

Event

- Indicates that something happened in the system

- There's never a response to the event

Event

- Main characteristics:

Event

- Something happened
- Asynchronous
- Never returns a response
- Calling service has no idea who handles the event

Contents of Event

- Two types of event data:

Complete

- Contains all the relevant data
- Usually entity data
- No additional data is required for the event processing
- Example:

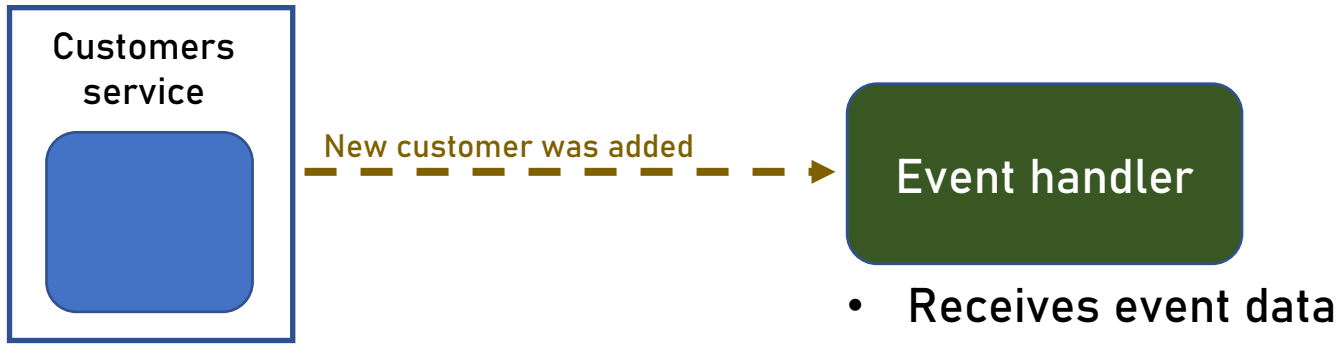
```
event_type: CustomerCreated
customer_id: 17
first_name: David
last_name: Jones
join_date: 2022-03-15
```

Pointer

- Contains pointer to the complete data of the entity
- Complete data usually stored in a database
- Event handler needs to access the database to retrieve complete data
- Example:

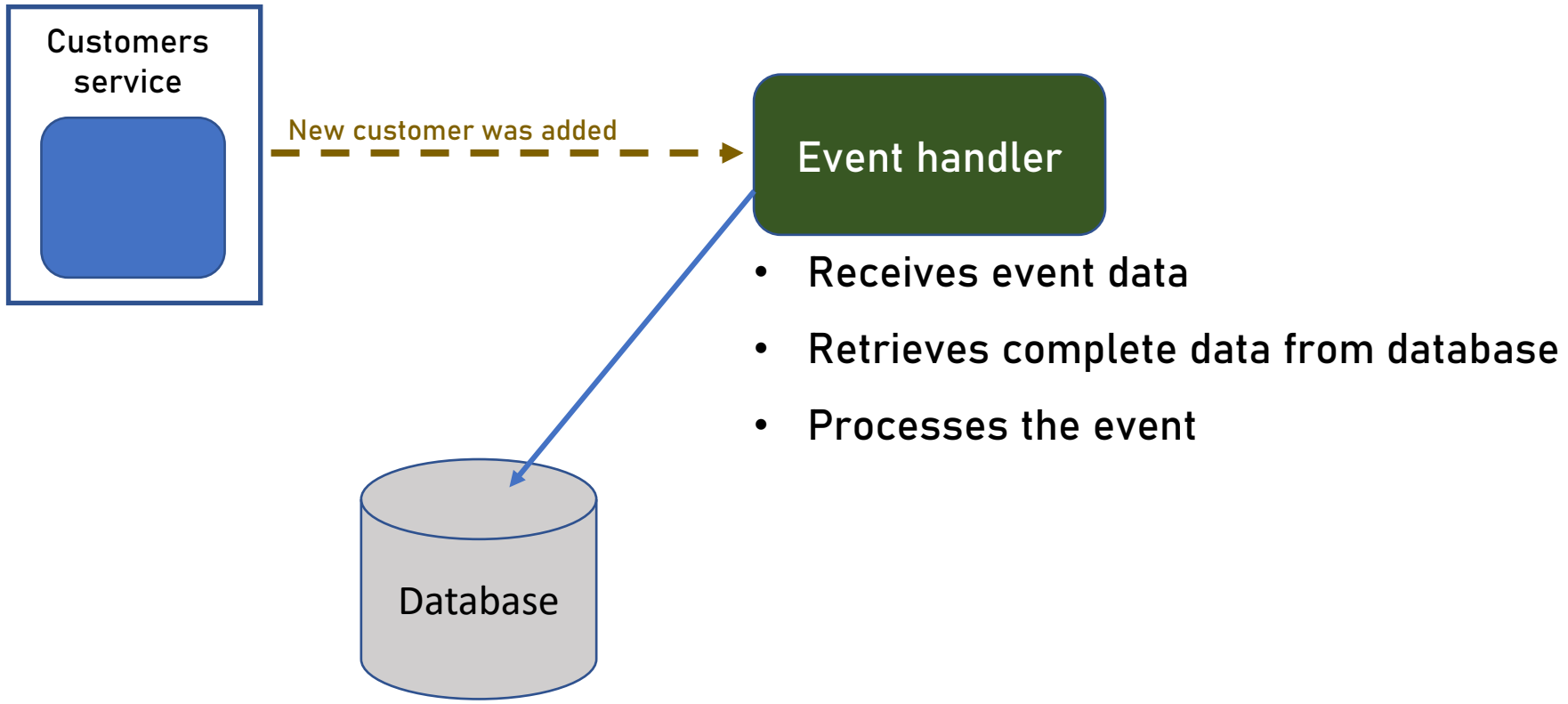
```
event_type: CustomerCreated
customer_id: 17
```

Flow of Complete Event Handling



- Receives event data
- Processes the event

Flow of Pointer Event Handling



- Receives event data
- Retrieves complete data from database
- Processes the event

Complete vs Pointer

- When to use which?

Complete

- The better approach
- Makes the event completely autonomous
- Can get out of the system boundaries

Pointer

- Use when:
  - Data is large
  - Need to ensure data is up-to-date
    - Assuming database is a single-source-of-truth