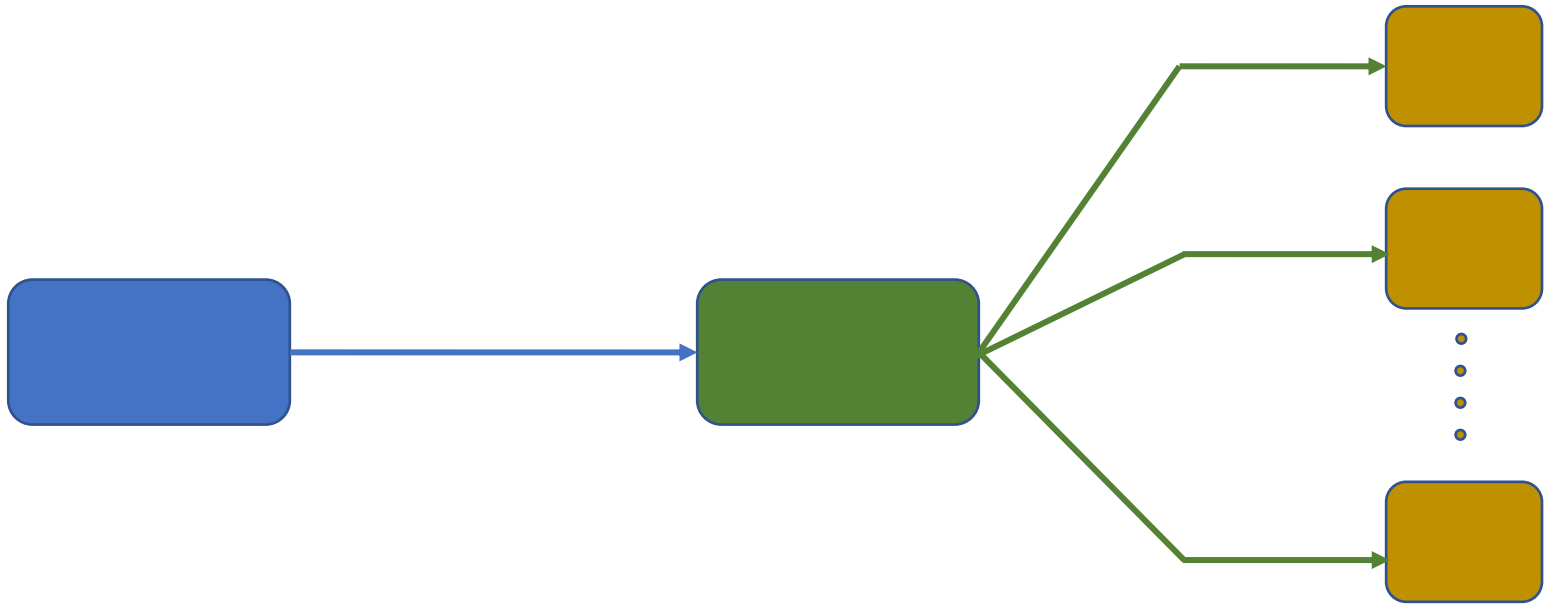
Event Driven Architecture

Memi Lavi
www.memilavi.com

Event Driven Architecture

- A software architecture paradigm that uses events as the mean of communication between services
- Often called EDA
- Has three main components

Event Driven Architecture Components

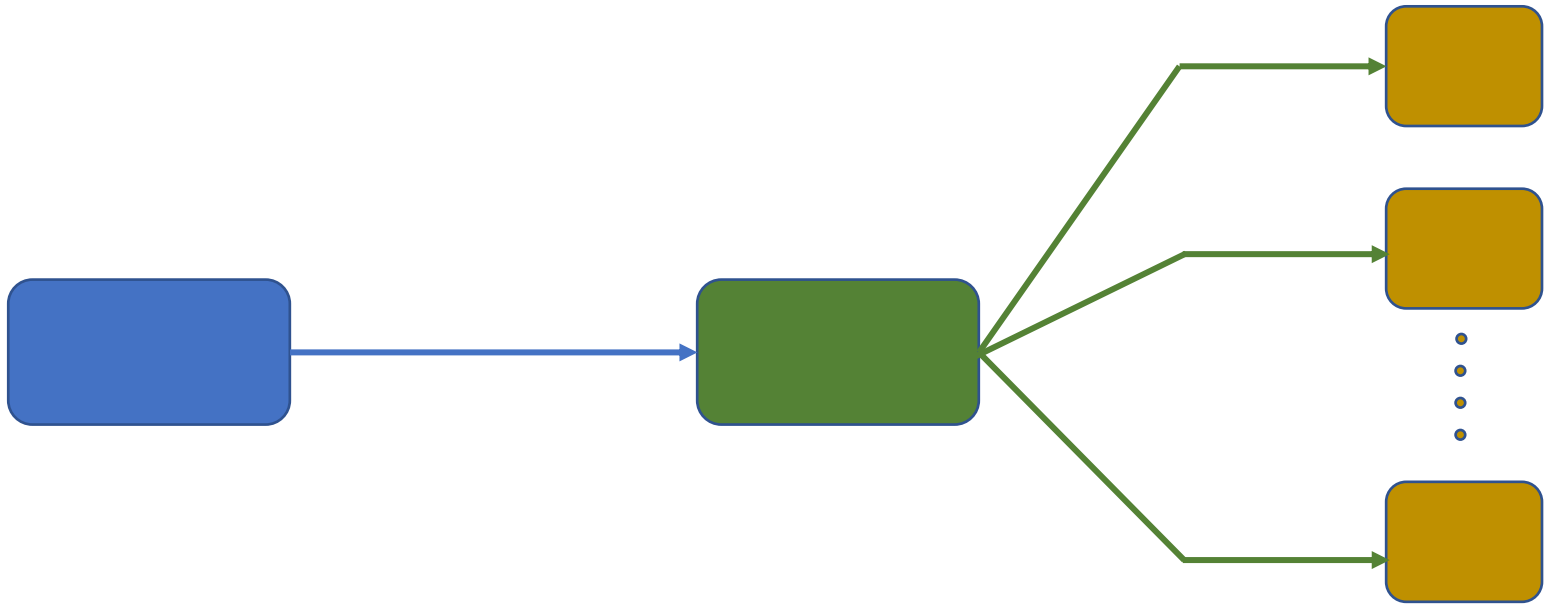


Producer

Channel

Consumer

Producer



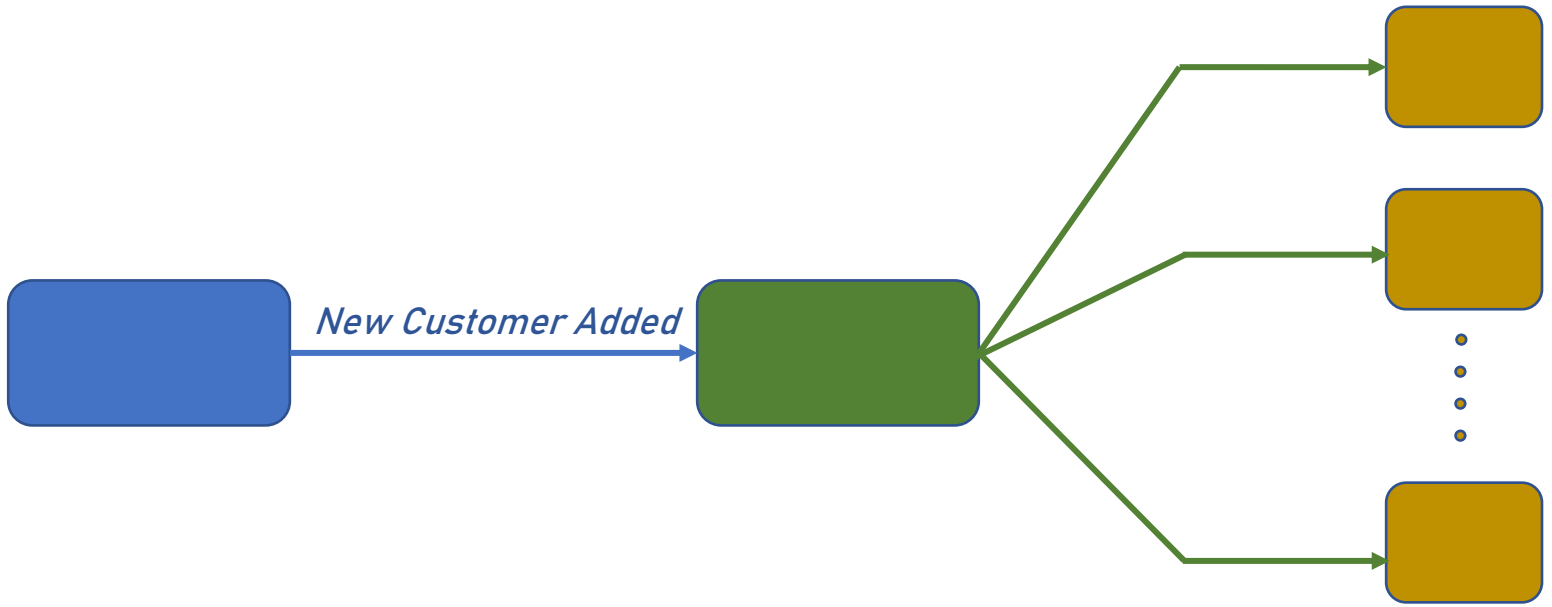
Producer

Producer

- The component / service sending the event
- Often called Publisher
- Usually sends event reporting something the component done
- Examples:
  - Customer service -> *New Customer Added* event
  - Inventory service -> *Item Sold Out* event

Producer

- The producer sends the event to the Channel



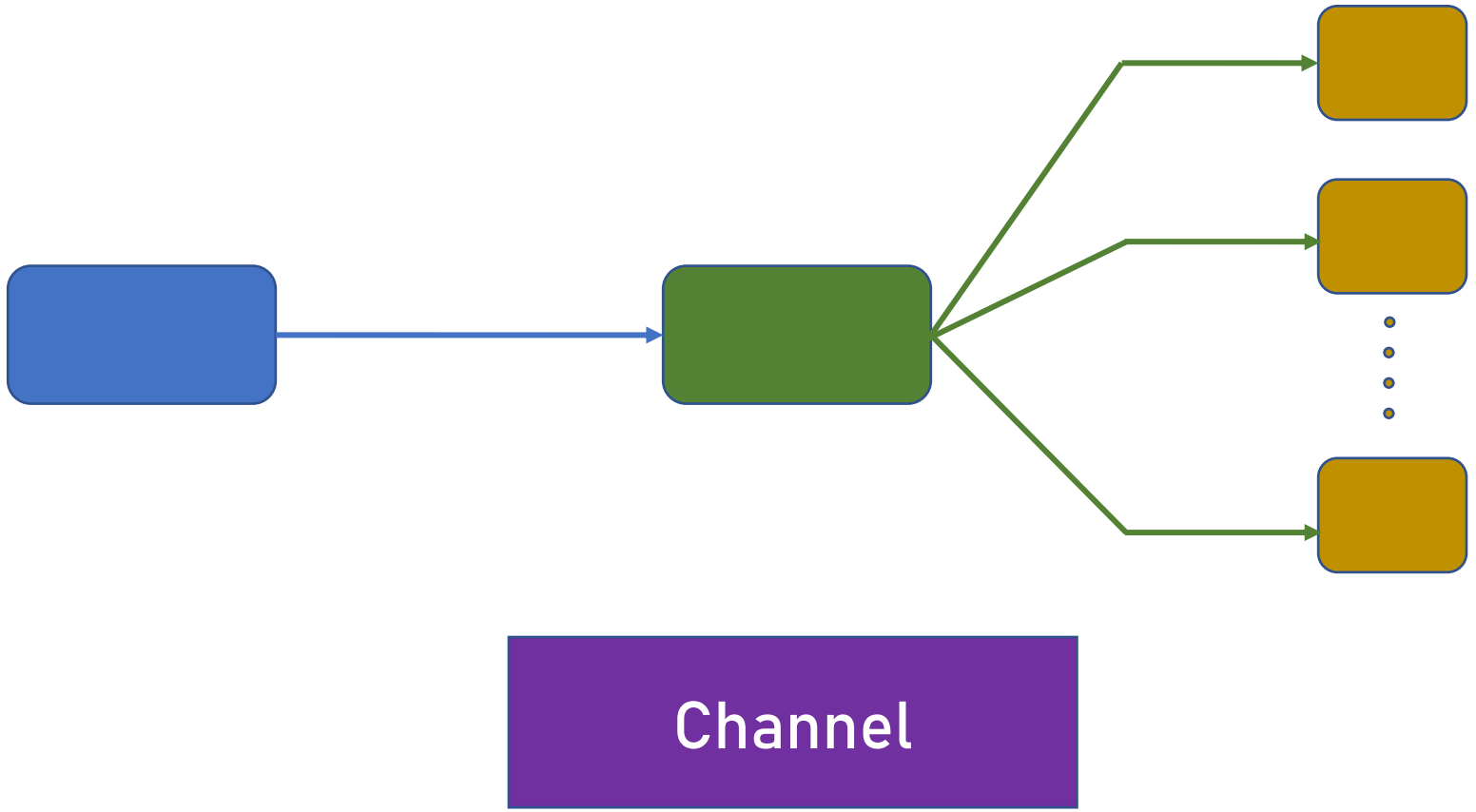
Producer

Channel

Producer

- Exact method of calling the channel depends on the channel
- Usually using a dedicated SDK developed by the channel vendor
- Utilizes some kind of network call, usually with specialized ports and proprietary protocol
- I.e.: RabbitMQ listens on port 5672 and uses the AMQP protocol
- Producer can be developed using any development language

Channel



Channel

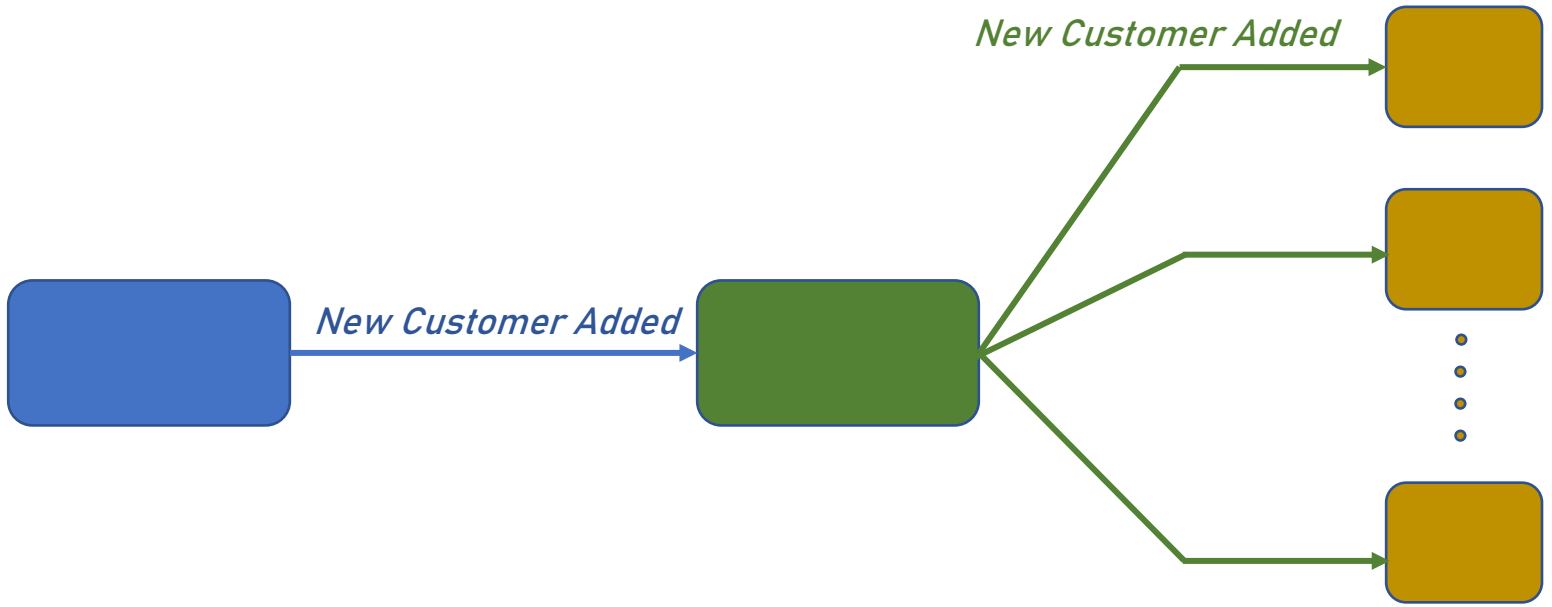
- The most important component in the Event Driven Architecture
- Responsible for distributing the events to the relevant parties
- The channel places the event in a specialized queue, often called Topic or Fanout
- Consumers listen to this queue and grab the event

Channel

- Note:
  - Implementation details vary wildly between channels
  - RabbitMQ works differently than Kafka that works differently than WebHooks etc
  - Always dive deep into the docs of the channel you're using
  - We'll use RabbitMQ and SignalR in the implementation section

Channel

- The Channel distributes the event to the Consumers



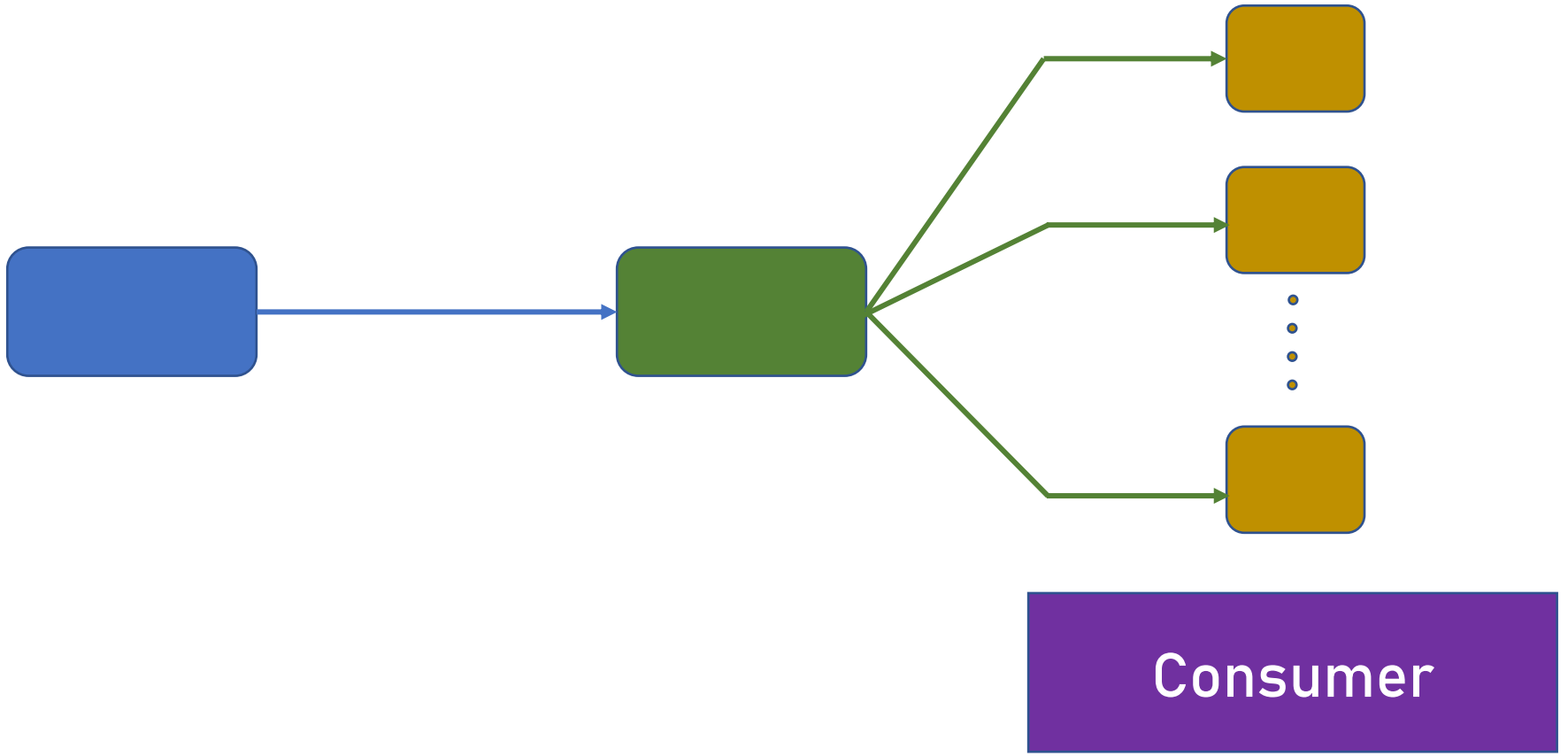
Producer

Channel

Channel

- The channel's method of distribution varies between channels
- Can be:
  - Queue
  - REST API call
  - Proprietary listener

Consumer

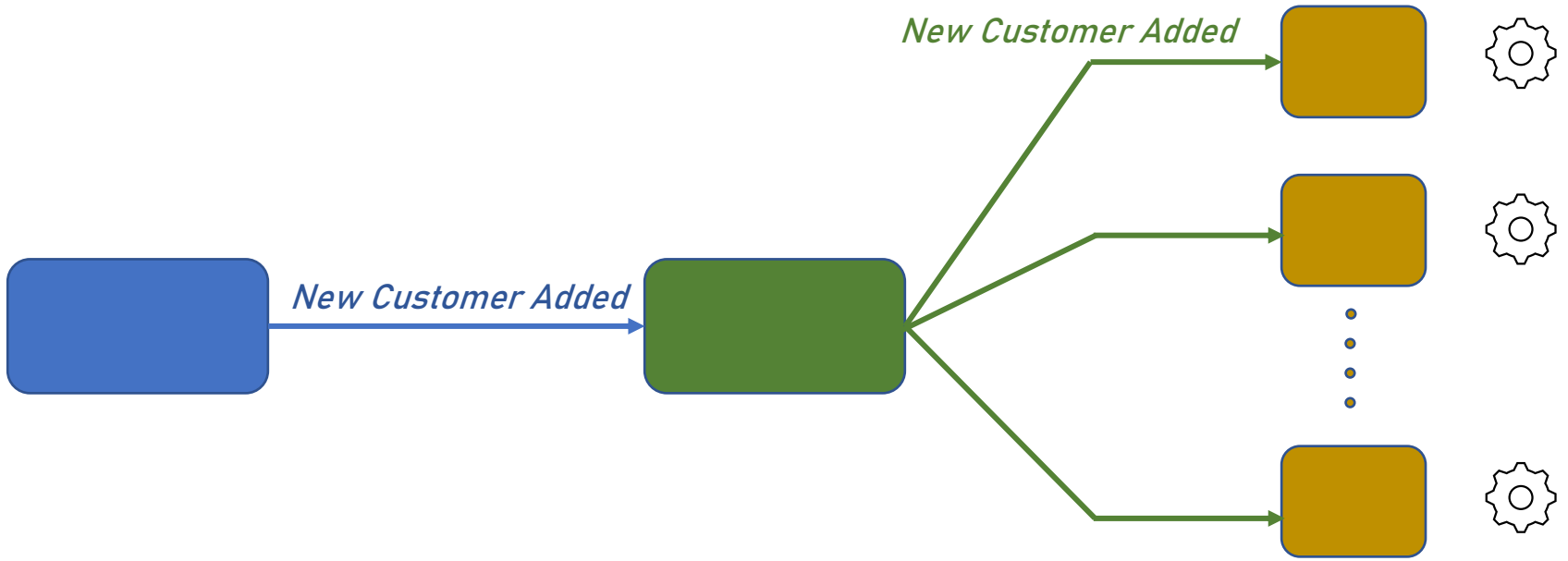


Consumer

- The component that receives the event sent by the Producer and distributed by the Channel
- Can be developed in any development language compatible with the Channel's libraries (if any)
- Processes the event
- Sometimes – reports back when processing is complete (Ack)

Consumer

- The Consumer receives and processes the event



Producer

Channel

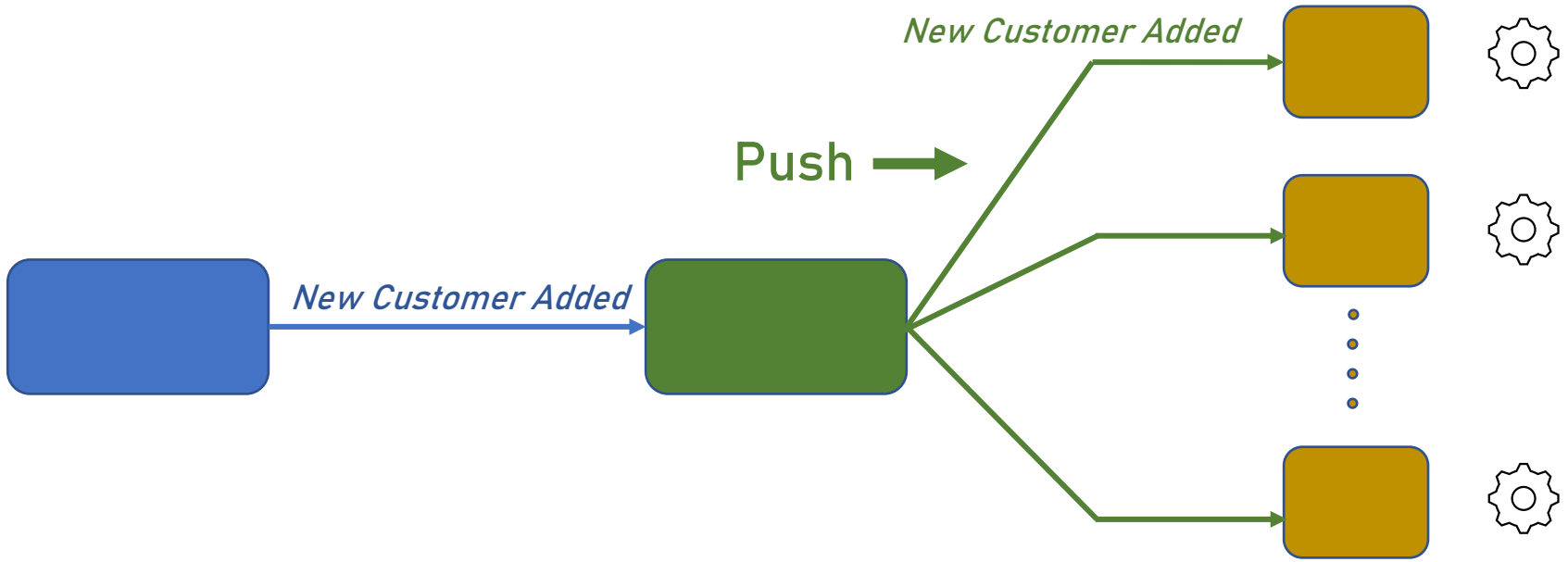
Consumers

Consumer

- Consumer gets the event using either:
  - Push
  - Pull
- The method depends on the channel

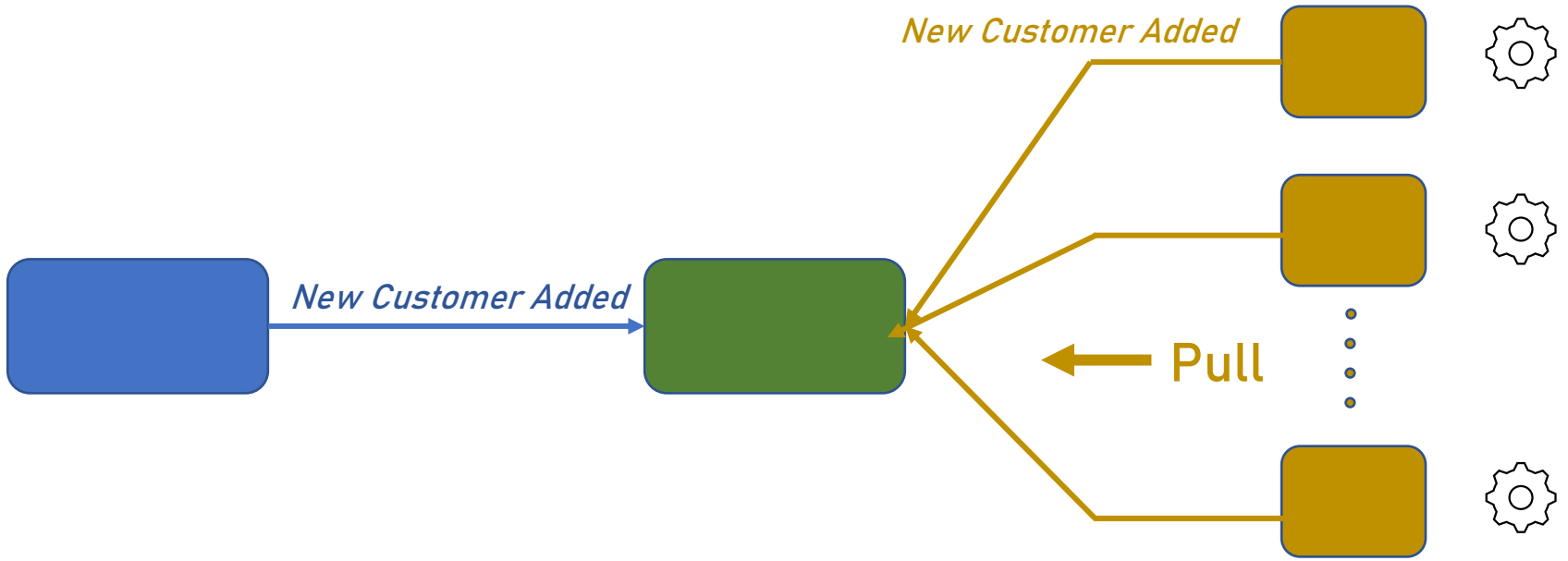
Push

- The Channel pushes the event to the Consumers



Pull

- The Consumers poll the Channel for new events



Producer

Channel

Consumers

Advantages of EDA

- Event Driven Architecture has a lot of advantages over other architecture paradigms
- As a quick refresher…

Problems with Command and Query

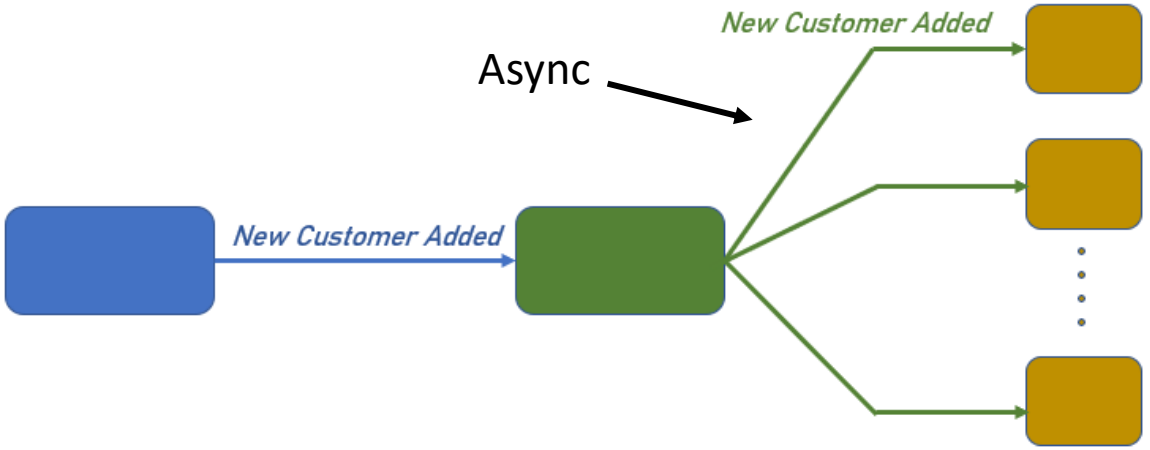
- Three major problems with command and query:

Performance

Coupling

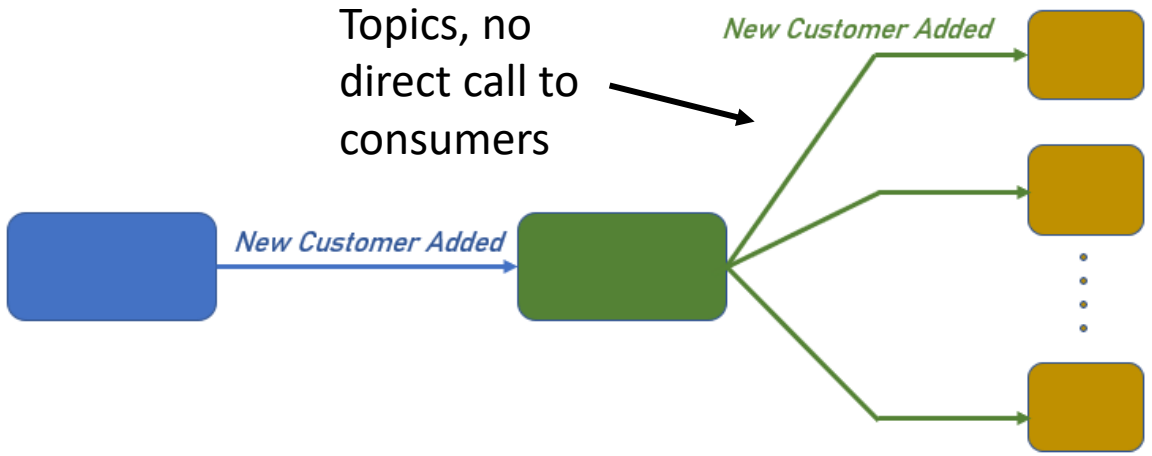
Scalability

Performance



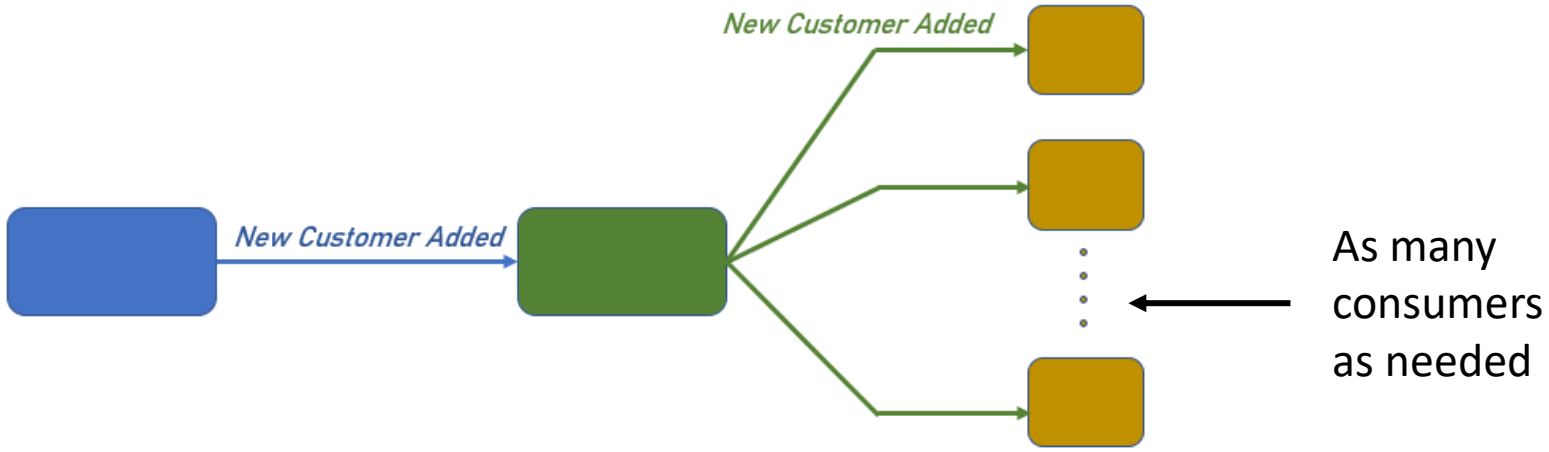
- EDA is an asynchronous architecture
- The Channel does not wait for response from consumer
- No performance bottlenecks

Coupling



- The producer sends events to the channel
- The channel distributes events to topics / queues
- Both have no idea who's listening to the event (except in WebHooks)
- No coupling

Scalability

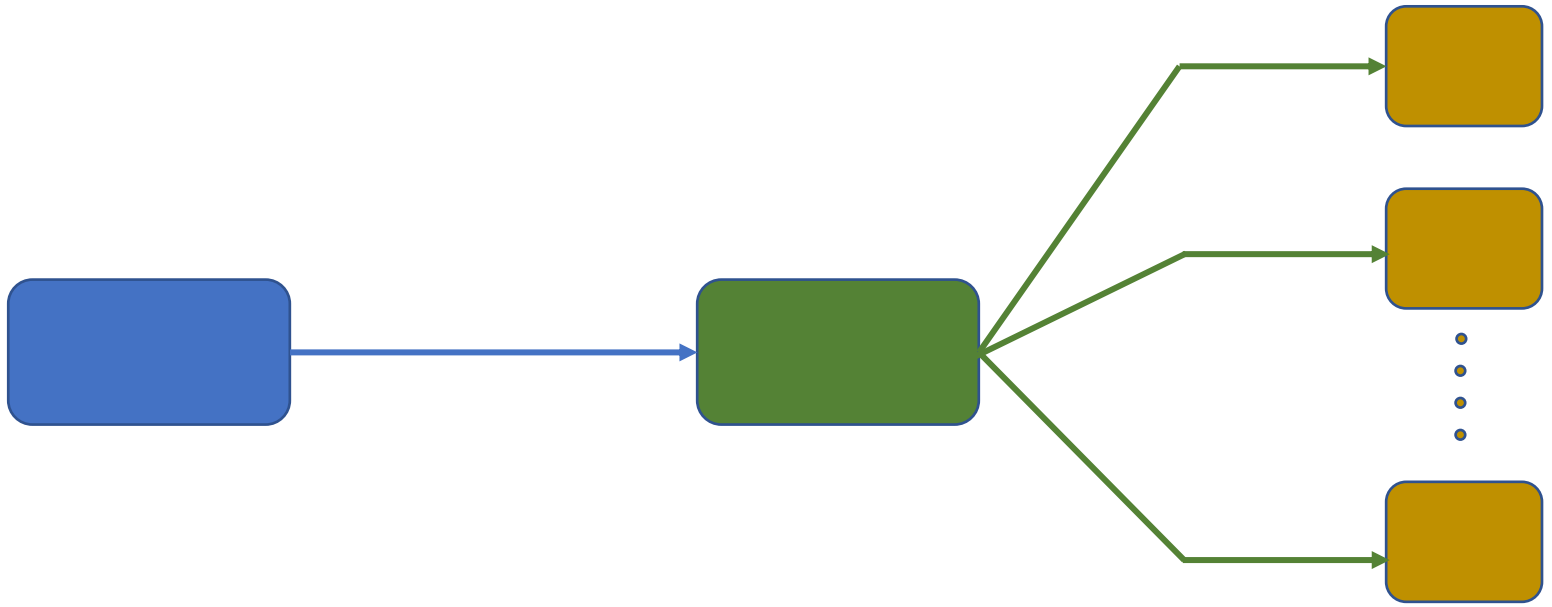


- Many consumers can listen to events from channel
- More can be added as needed
- Channel doesn't care, producer doesn't know
- Fully scalable

EDA and Pub/Sub

- Event Driven Architecture is often mentioned with Pub/Sub
- Pub/Sub = Publish and Subscribe
- A messaging pattern used by Event Driven Architecture

Components of Pub/Sub



Publisher

Broker

Subscriber

EDA and Pub/Sub

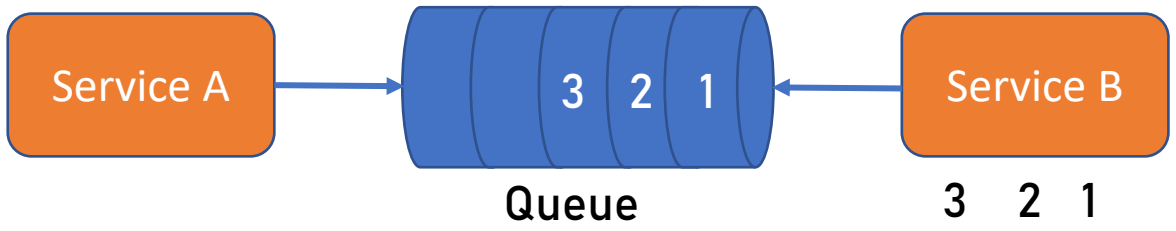
- Event Driven Architecture and Pub/Sub are extremely similar
- Main difference:
  - EDA describes the whole architecture of the system
  - Pub/Sub is a messaging pattern used by the system
    - Not exclusively!

EDA and Pub/Sub

- For example:
  - *“My Event Driven Architecture uses mainly Pub/Sub for inter-service communication, but I do have some REST APIs for synchronous queries.”*

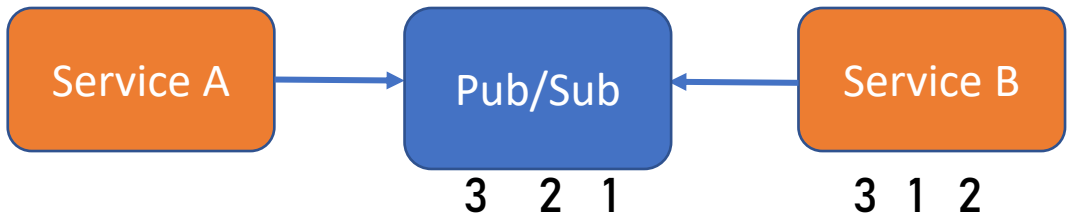
Ordering in EDA

- Messaging engines often guarantee the order of the messages
- Popular mainly in traditional queues



Ordering in EDA

- With Event Driven Architecture (especially with Pub/Sub) ordering is not always guaranteed
- Ordering might be affected by consumer latency, code performance and more



Ordering in EDA

- If ordering is important, make sure to select a channel that supports this capability
- Examples:
  - RabbitMQ supports it
  - SignalR does not
- We'll use both in the case study section

Orchestration and Choreography

- Event Driven Architecture usually employs one of two architectural styles

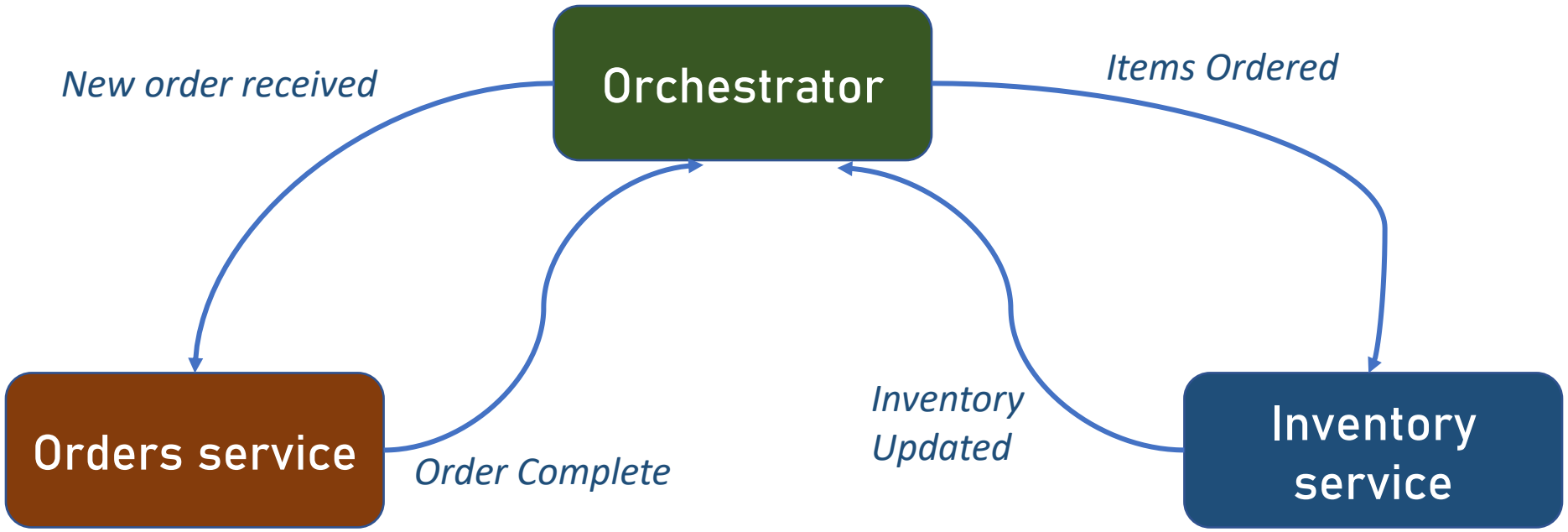
Orchestration

Choreography

Orchestration

- Flow of events in the system is determined by a central orchestrator
- Orchestrator receives output from components and calls the next component in the flow
- The next component sends the output back to the orchestrator etc.

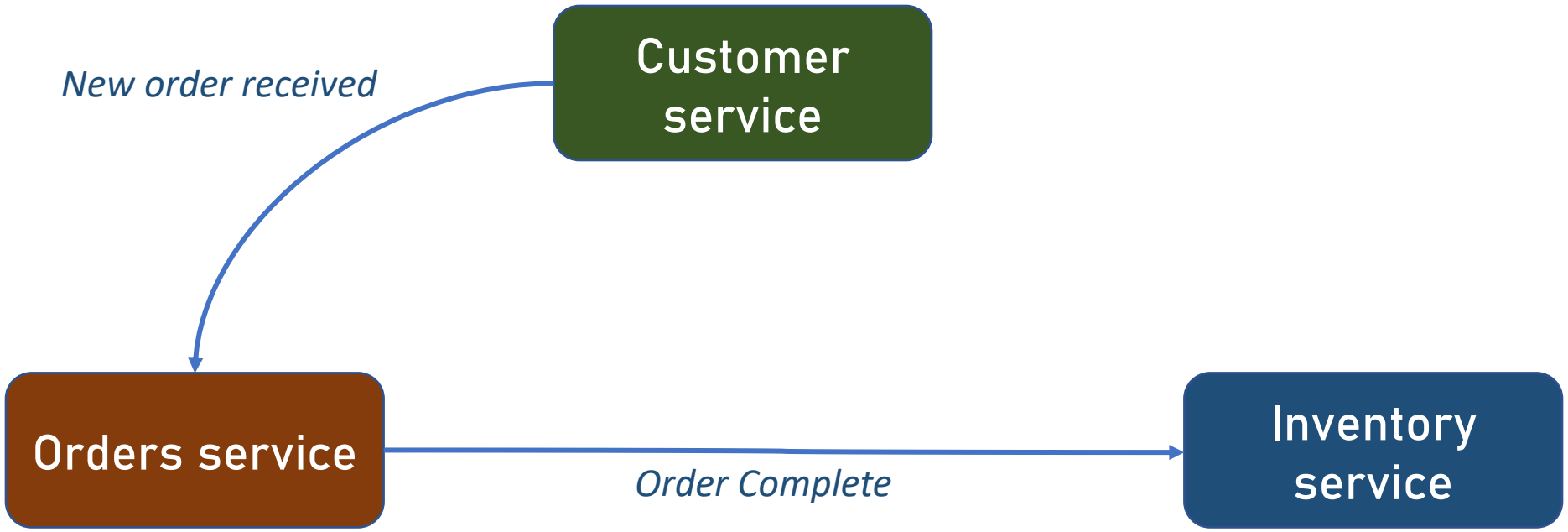
Orchestration



Choreography

- No central “knowing all” component
- Each component notifies about the status of events
- Other components listen to the events and act accordingly

Choreography



Orchestration and Choreography

Orchestration

- Logic is defined in a single place – easier to maintain
- Central traffic gateway – easier monitoring and logging

Choreography

- Performance – no middleman
- Reliability – if one component fails, the rest still work

Orchestration and Choreography

- Not constrained to EDA only
- Can be used with other types of communication
- Became popular with EDA