

Fantastic 9

Yves Goeleven

yves@goeleven.com
@yvesgoeleven



Yves Goeleven



Freelance Solution Architect

- My mission is to simplify distributed software development (on Azure)
- Advisory consulting, coaching & training
 - Currently advising teams @ Q-Park & Cegeka
- More info at: www.goeleven.com
- Still building software (for) myself
 - www.clubmanagement.io
 - www.messagehandler.net

Claim check Registry Singleton Prototype
Query object Builder Data Transfer Object
Identity Map Table data gateway Value Object
Specification Mediator Aggregator Adapter Pipes and filters
Table module Transaction script Decorator
Proxy Active record Observer Composite Strategy
Normalizer Entity Template Bridge
Gateway Memento Wire tap
Record Set Front Controller
Flyweight Splitter Repository Factory Wrapper
Interpreter Unit of work
Aggregate Root Application Controller
Mapper Service Layer
Facade Domain model

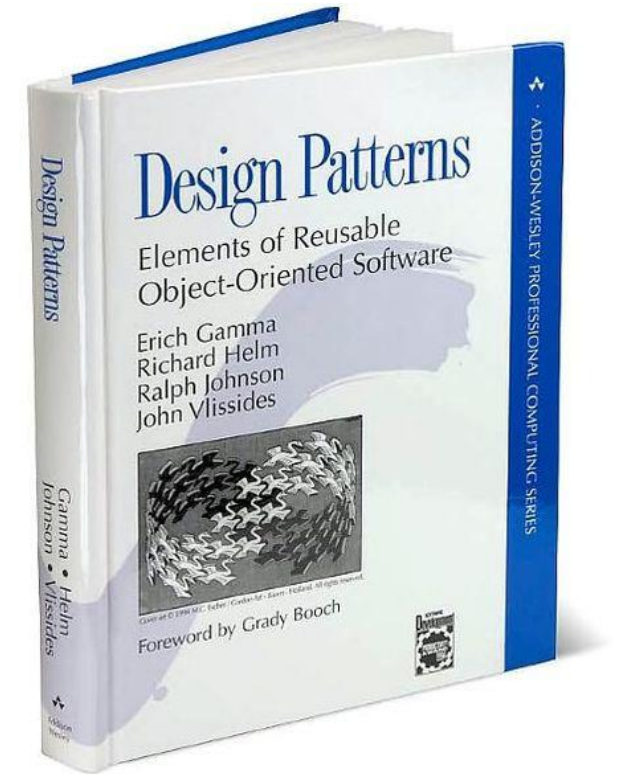
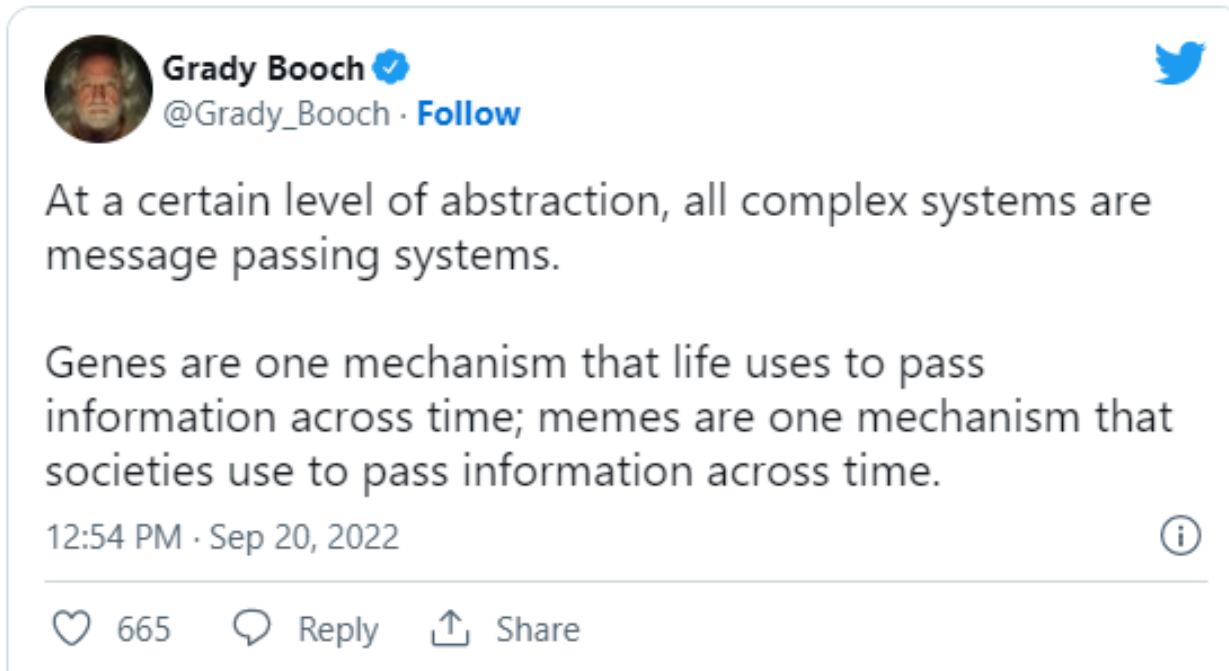


Complexity is our enemy



Why 9 ?

Once upon a time a software legend revealed the secret





Messaging

Passing information across time

Einstein's theory of relativity: space and time are two sides of the same coin.

Time only consists of 3 divisions

Past, present and future

| Past | Present | Future |
|--|--|--|
| <ul style="list-style-type: none">- Event- Fact- Certain outcome | <ul style="list-style-type: none">- State- Properties- Slowly evolving | <ul style="list-style-type: none">- Command- Intent- Uncertain outcome |

Booking
Received

Purchase Order

Confirm
Purchase
Order






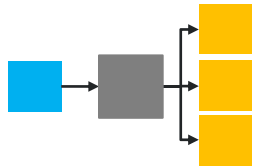



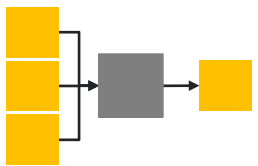
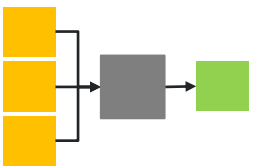


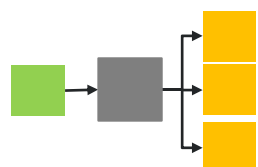

Which leaves us with 9 transitions

These are your 9 fantastic processing patterns

| In\Out | Command  | Event  | State  |
|---|---|---|---|
| Command  | ? | ? | ? |
| Event  | ? | ? | ? |
| State  | ? | ? | ? |

My fantastic 9

Feel free to steal them

| In\Out | Command  | Event  | State  |
|---|---|--|---|
| Command  | <p>Delegation</p>  | <p>Aggregate Root</p>  | <p>Downstream Activity</p>  |
| Event  | <p>Reaction</p>  | <p>Event Stream Processing</p>  | <p>Projection</p>  |
| State  | <p>Task processing</p>  | <p>Event Generator</p>  | <p>State Transformation</p>  |

... or less

System categories require only a subset

Line Of Business

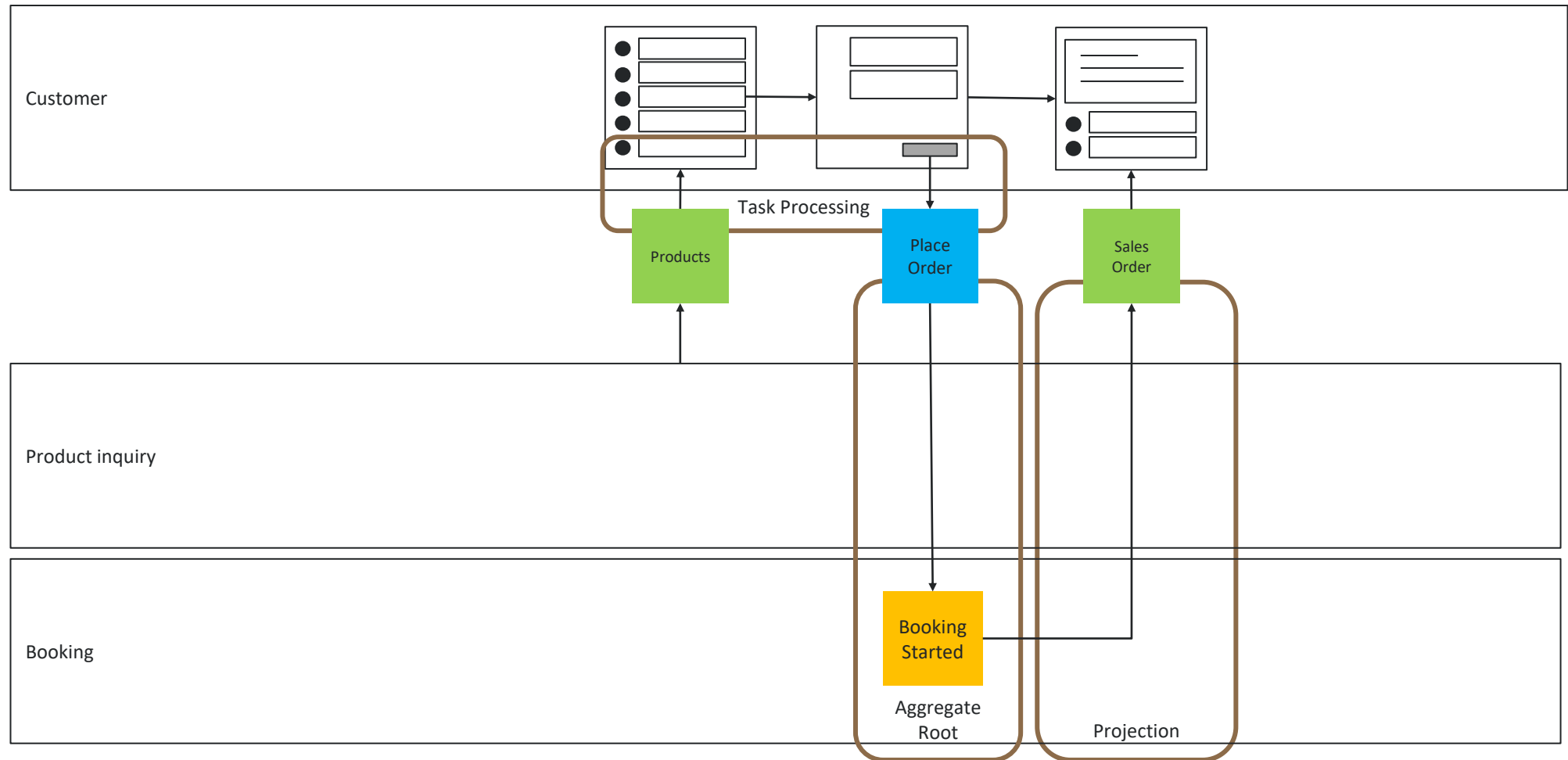
Internet Of Things

Business Intelligence



Line Of Business

Task processing -> Aggregate Root -> Projection





Aggregate root

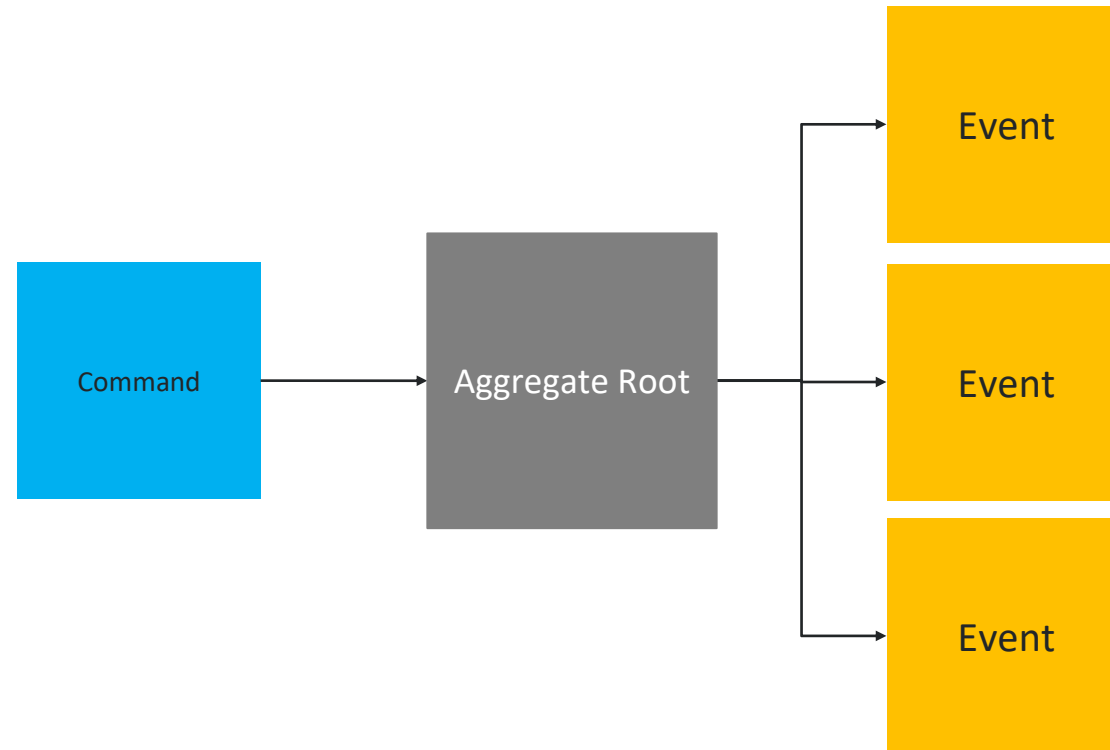
(The decider)

Decides what will happen in the system.

Aggregate root

Decides upon commands, records decisions as events

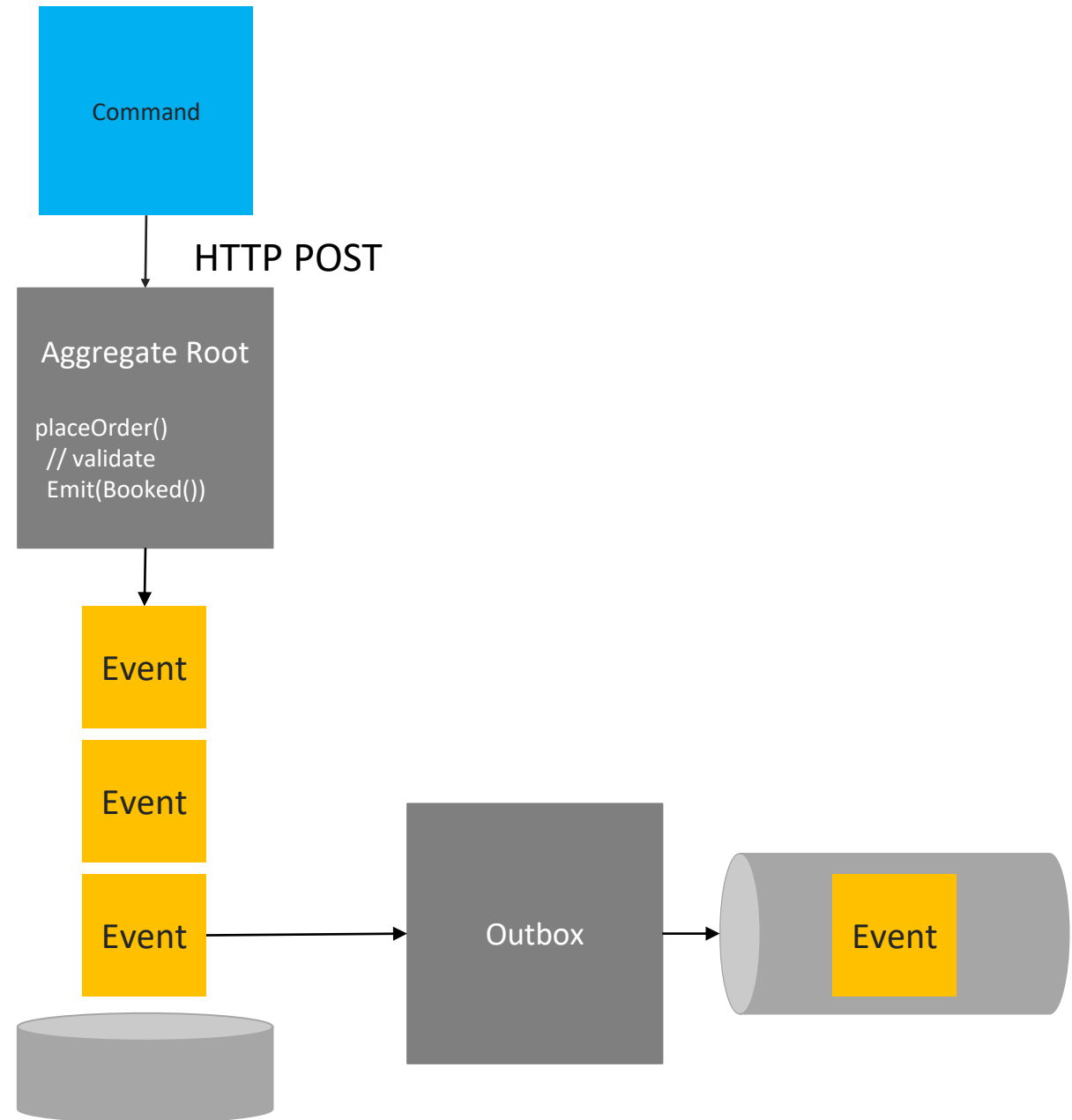
- Use to protect the system when taking a meaningful decision



Event sourcing

Hosted in web api

- Past decisions stored in an event store
- Easy to deduplicate
 - Just check existence event
- Optional outbox
 - When distribution is needed





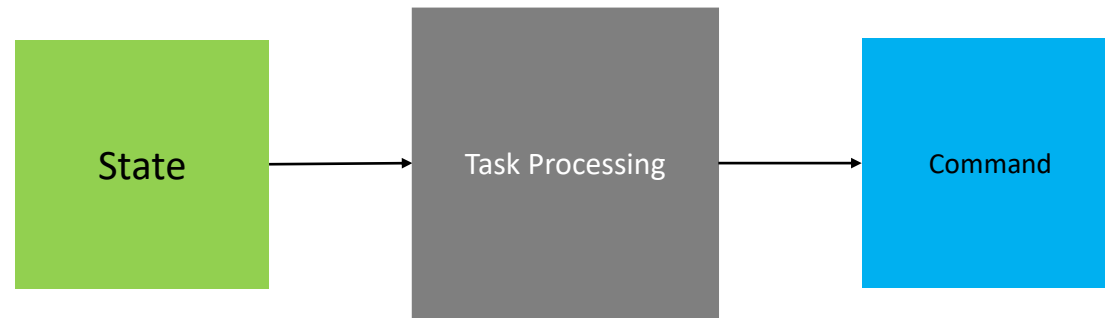
Task Processing

Makes choices, based on current state

Task Processing

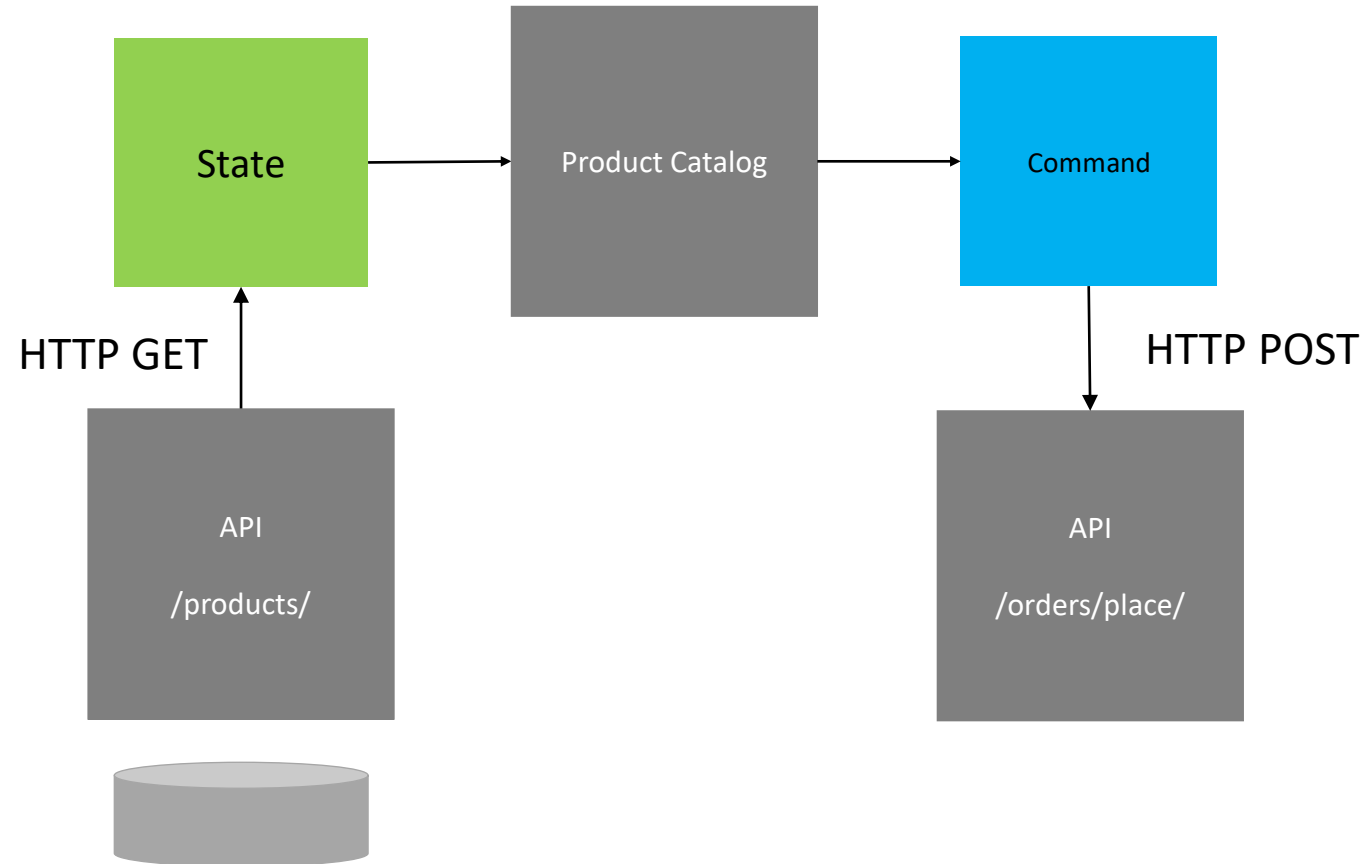
Makes choices, by invoking an action after looking at state

- Use this pattern every time a task needs to be invoked after a certain condition has become true.
- Sometimes called: the todo-list pattern



Conditions unknown

Let a user make the choices.



Conditions unknown

I was considering to let AI make the choices

- AI automation tools: LangChain, AutoGPT, ...
- Until... goblin.tools (by Bram De Buyser) came up with this

☰ Yves Goeleven 😊 ☰

▼

☰ 1. Gather the necessary materials. ✎ ☰

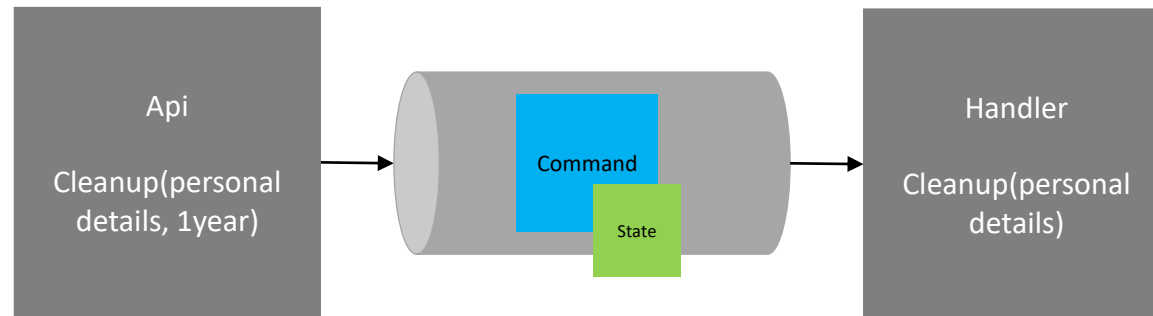
☰ 2. Heat the oven to 375°F. ✎ ☰

☰ 3. Combine the dry ingredients in a bowl. ✎ ☰

Conditions known / only limited to time

Can be automated by code

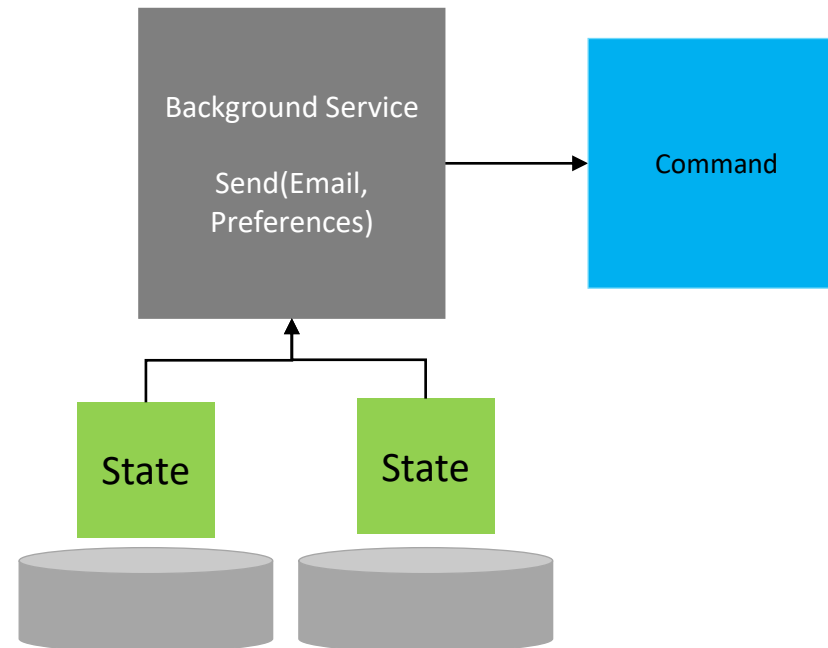
- Embed full state in the command
- Optionally use delayed delivery (feature queuing transport).



Complex known conditions

Can be automated with code

- Background service continuously queries state until condition is met,
- Then invokes command (usually in process)





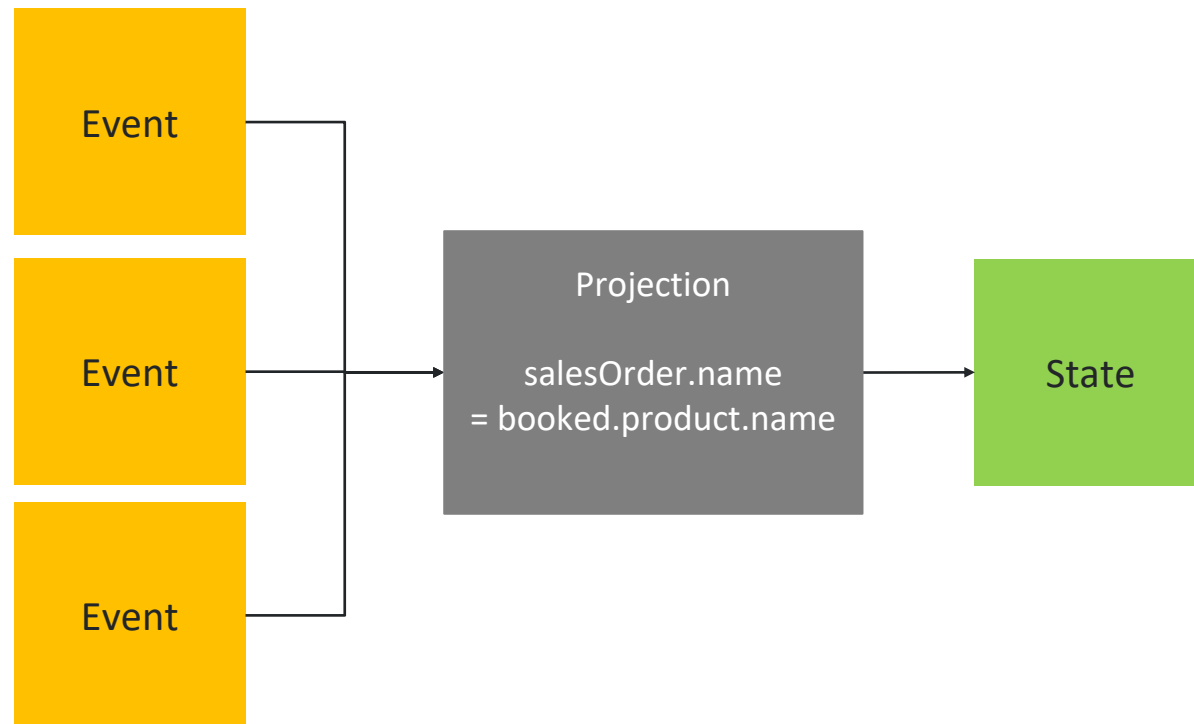
Projection

Turn event history into human readable state

Projection

Turn event history into human readable state

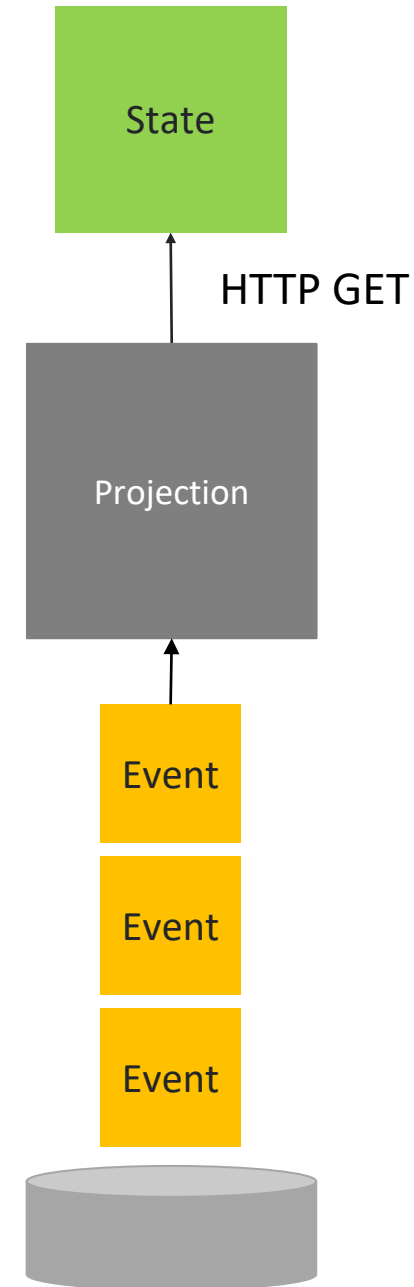
- Use this pattern every time a user needs to see the current state of the system.



Same person

On demand projection for session consistency

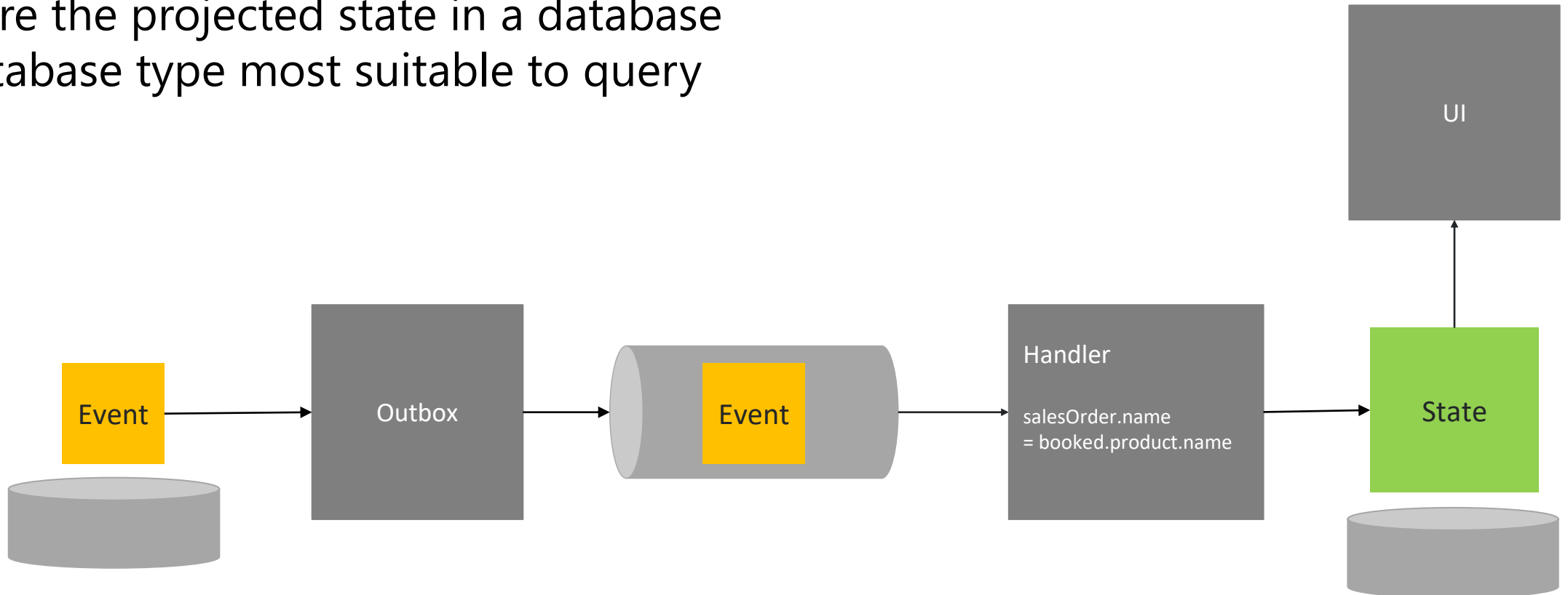
- Same person that invoked a command
- Will also query for to the result



Another person

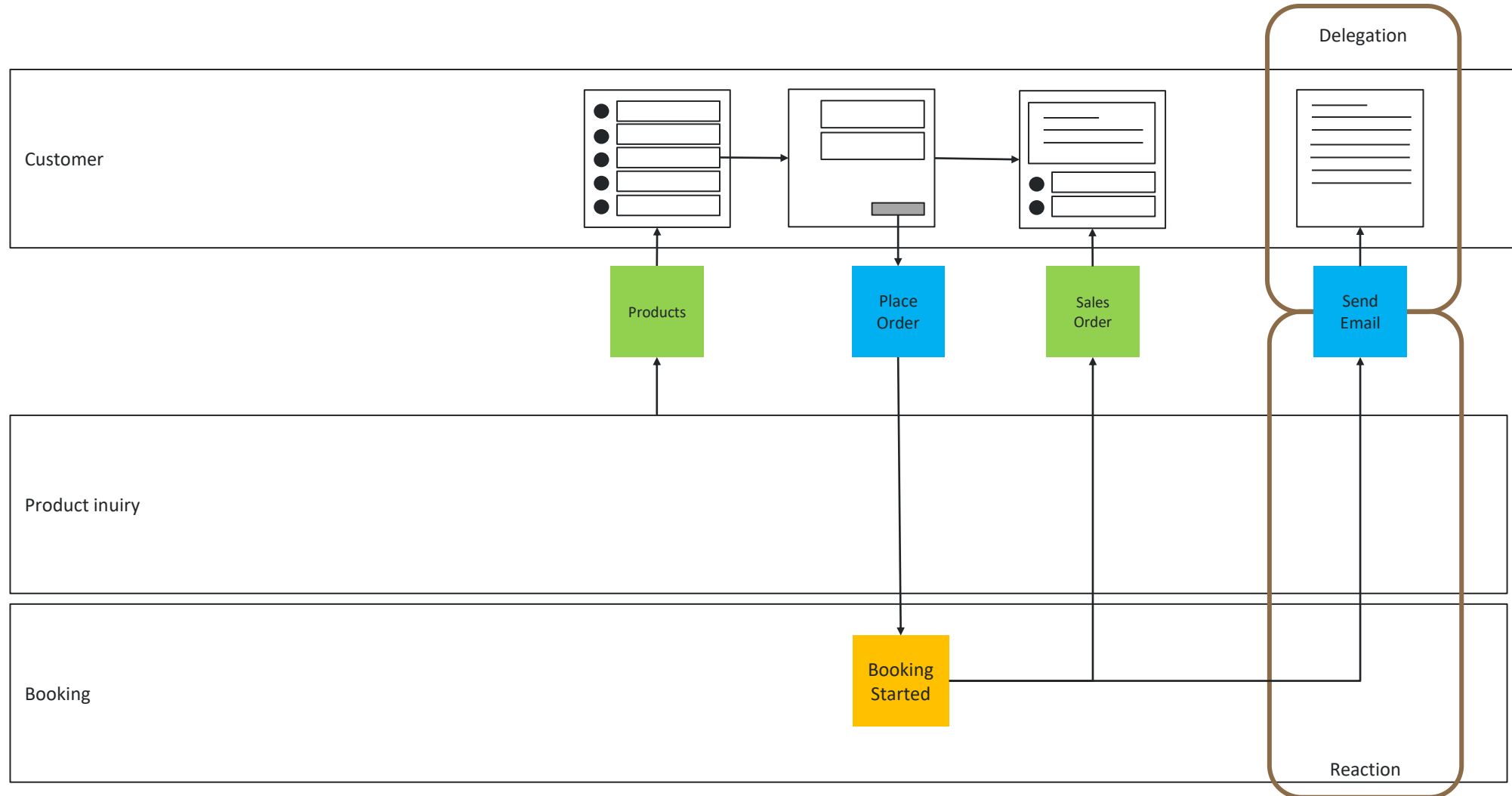
Eventual consistency and polyglot persistence

- Another person wants to see the result of a command (typically, a bit later)
- Store the projected state in a database
- Database type most suitable to query



What you've seen so far was CQRS + ES

Common additions: Reaction & Delegation



Reaction

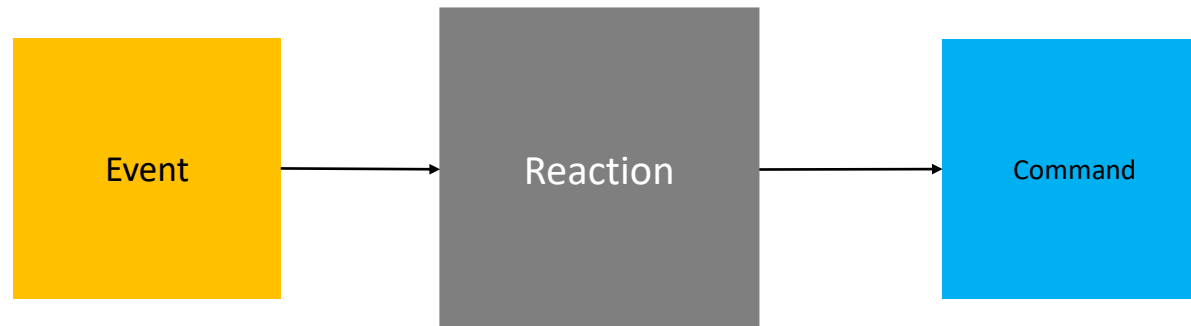
Take action in response to an event



Reaction

Invoke an action in response to an event

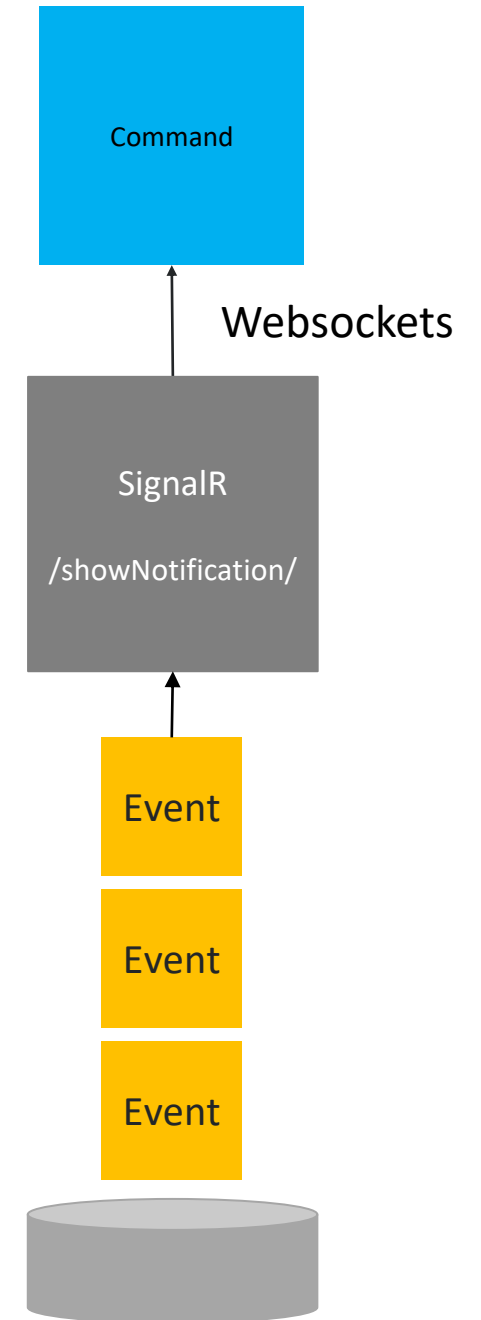
- Use this pattern every time something needs to happen in reaction to something else that happened before.



Transient reactions

Best effort execution of the command

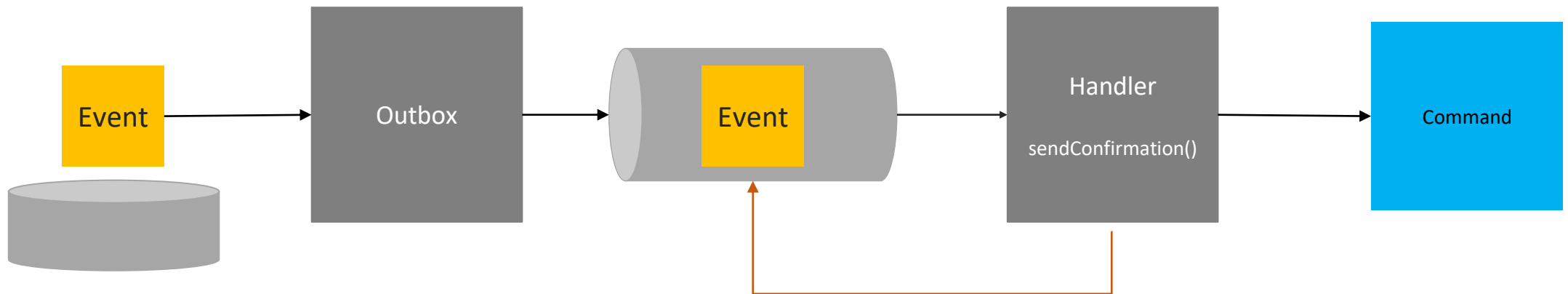
- Notify a person who may or may not be paying attention
- Through a transient channel (e.g., Websockets/SignalR)



Guaranteed Reaction

Queues offer guaranteed delivery

- Obligation to perform a command (e.g., sending an order confirmation)
- Use a transport with delivery guarantees





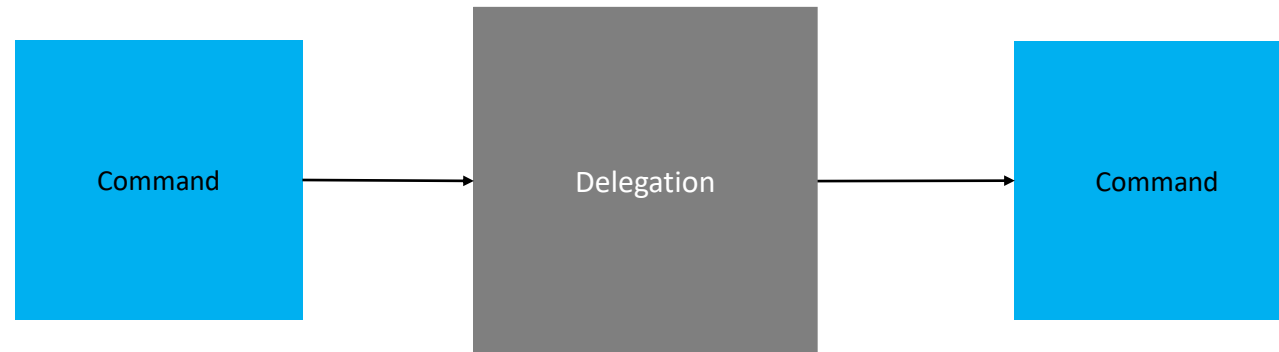
Delegation

Delegate the execution of a command to another component

Delegation

Delegate the execution of a command to another component

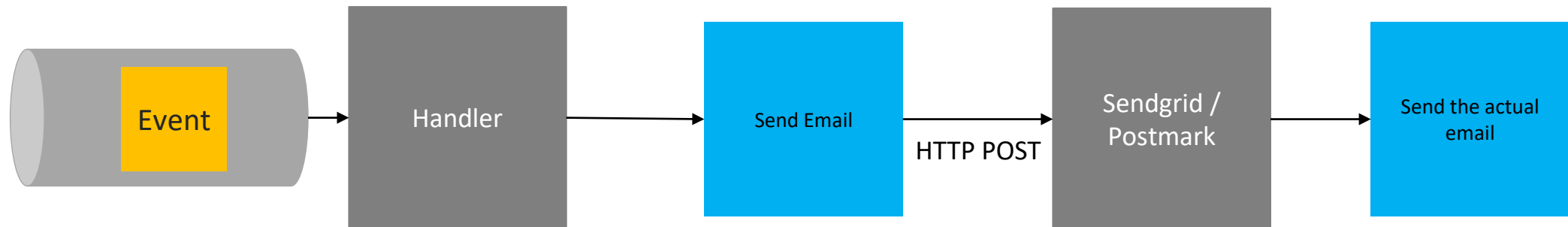
- Use this pattern every time something needs to happen in another (specialized) location.



Seemingly simple commands

That turn out to be bloody b**tch*s

- Email (SPF, DMARC, DKIM, ...)
- Payments (PA-DSS, PCI-DSS, 3DS, LoA EMVCo, ...)
- Delegate to the pro's



System categories

Line Of Business

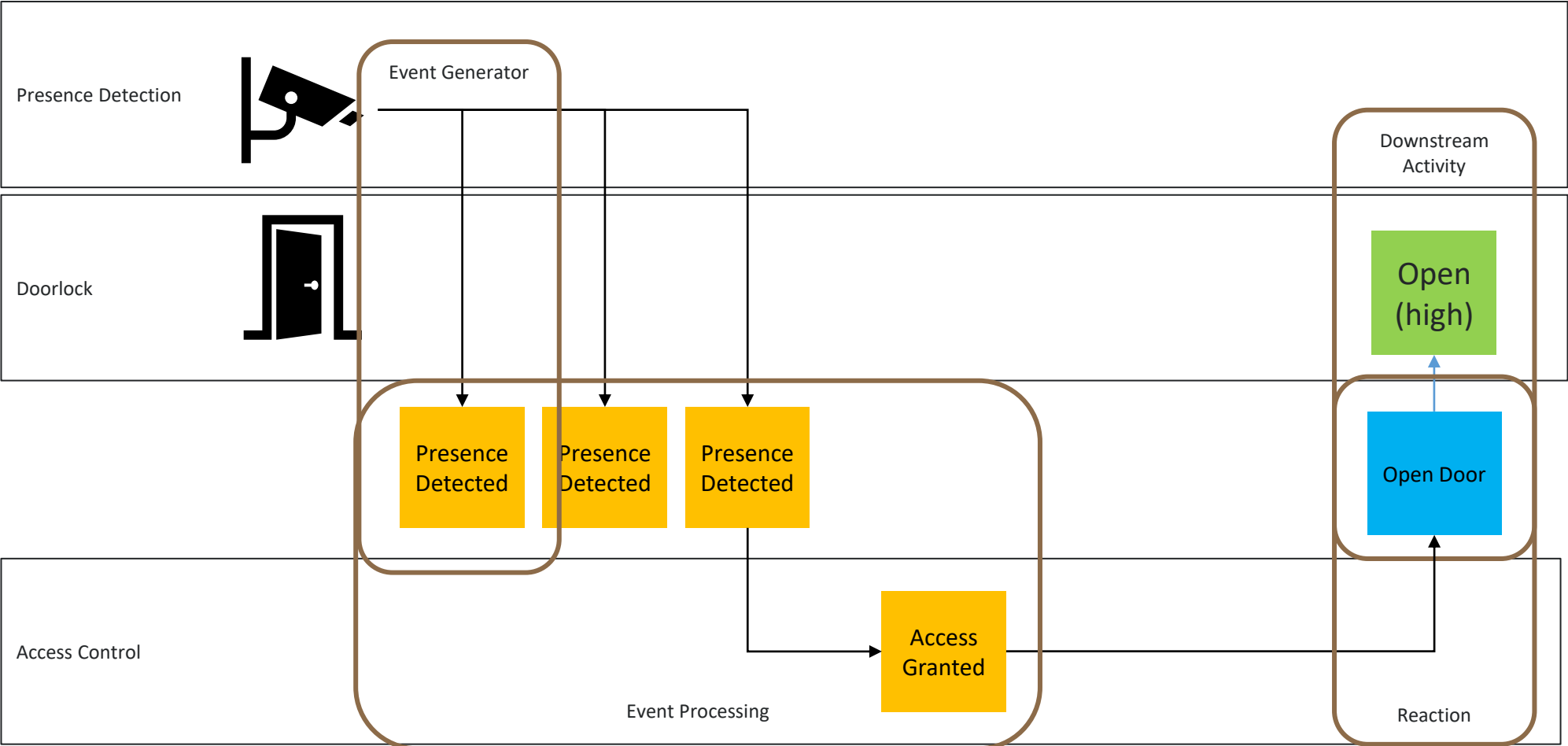
Internet Of Things

Business Intelligence



Internet Of Things

Event Generator -> Event Processing -> Reaction -> Downstream Activity



Event Generator

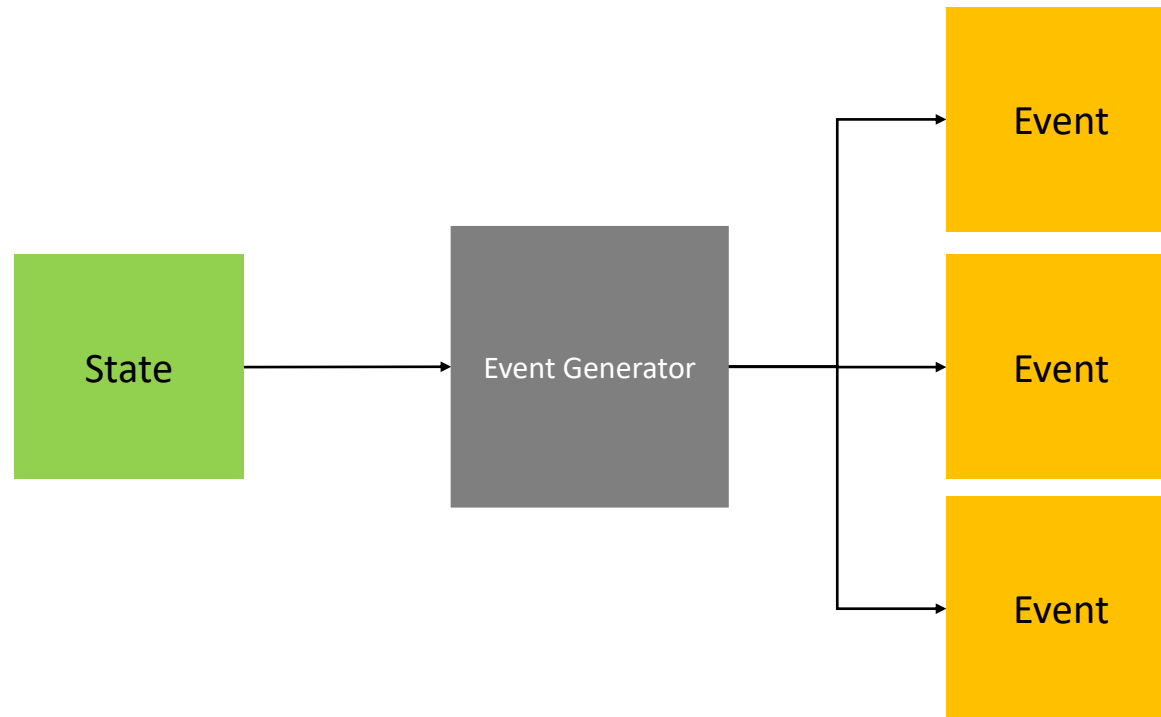
Turn state changes into events



Event generator

Turn state changes into events

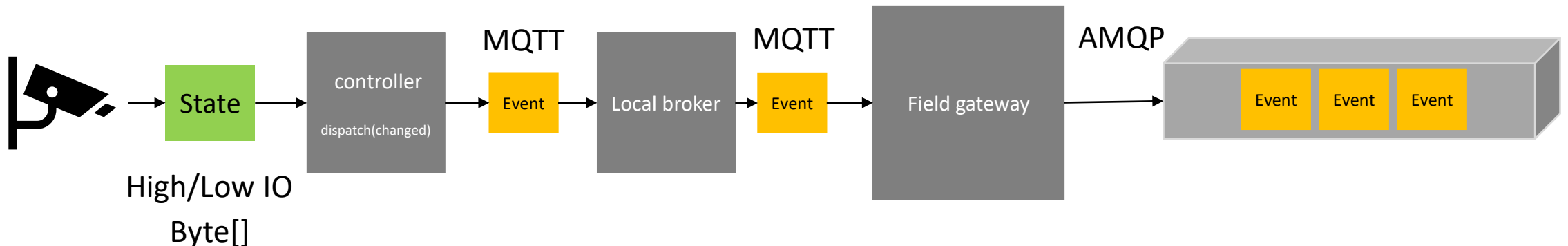
- Typical use is to capture state changes, in the real world, through sensors.
- Virtual event generators can be useful to detect changes in databases (Change Data Capture / Cosmos DB Change feed).



A typical field setup

The event generator is typically hosted in a (micro) controller

- Hardware reports state
- Attached (micro) controller detects state changes, reports as discrete events
- Discrete event are forwarded to local broker (mqtt based)
- Field gateway captures all events and appended to streaming / log based transport (e.g. IoT Hub or Event Hub)



Stream Processing

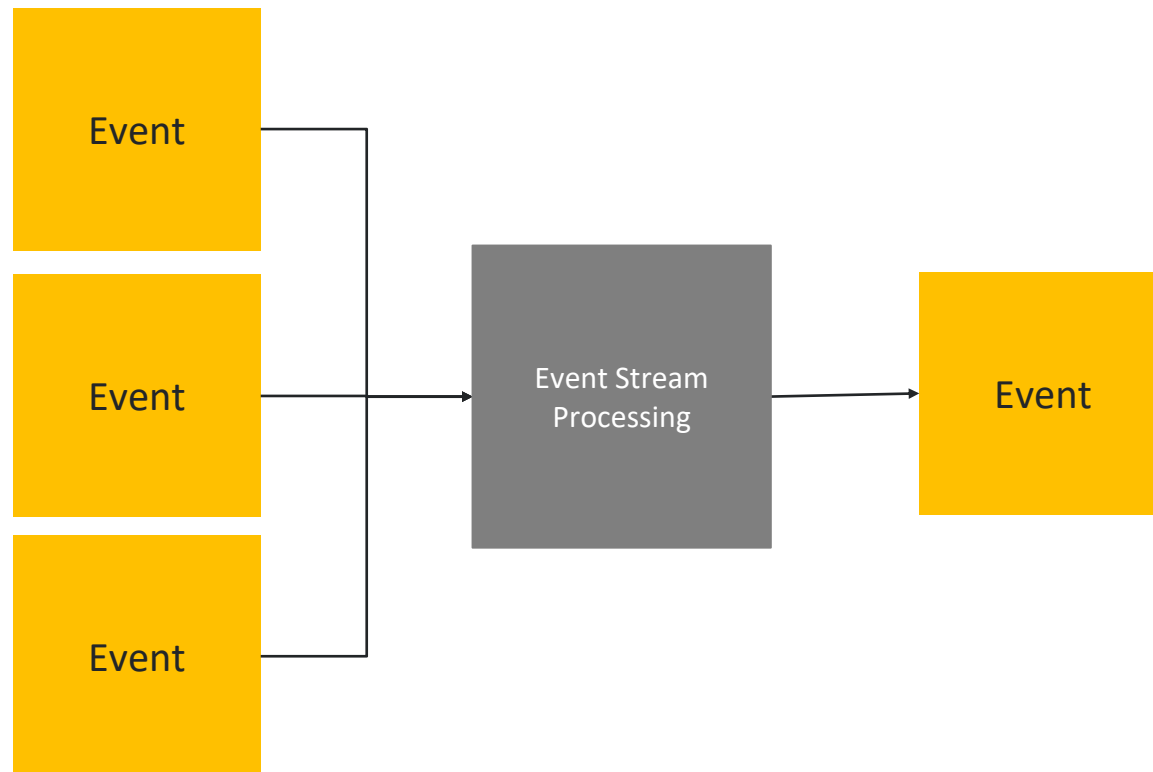
Derive events from other events



Event stream processing

Derive events from event history

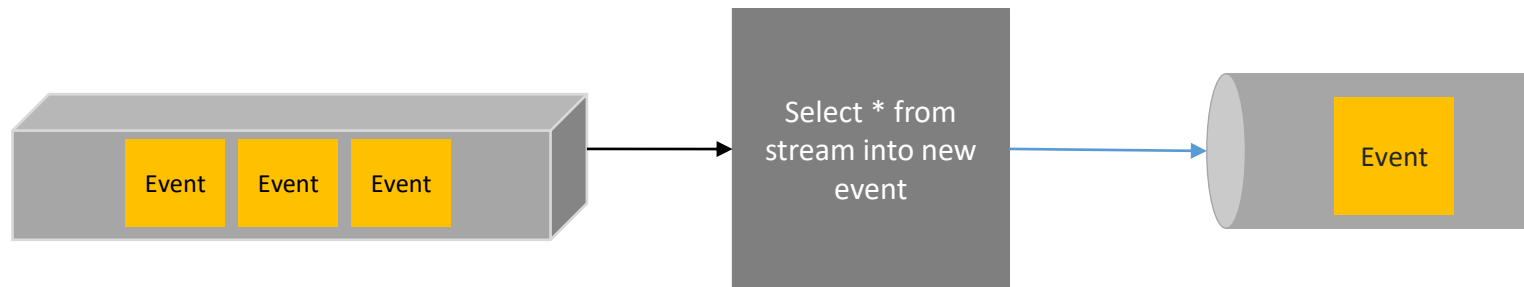
- Use to filter, count, group, join or window event streams
- Into a new derived event.



Needs a standing query engine

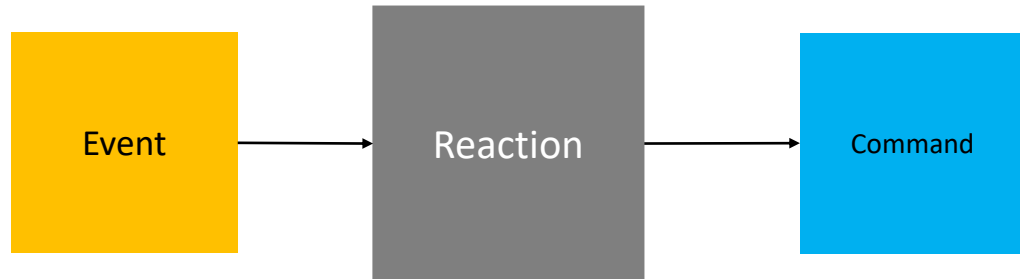
Consuming from a streaming transport

- Azure Databricks (Java/Scala or python)
- Azure Stream Analytics (SQL)
- Reactive / Reaqtive Extensions (C#) (by Bart De Smet)
- Neural Nets



Followed by a reaction

To invoke a downstream activity





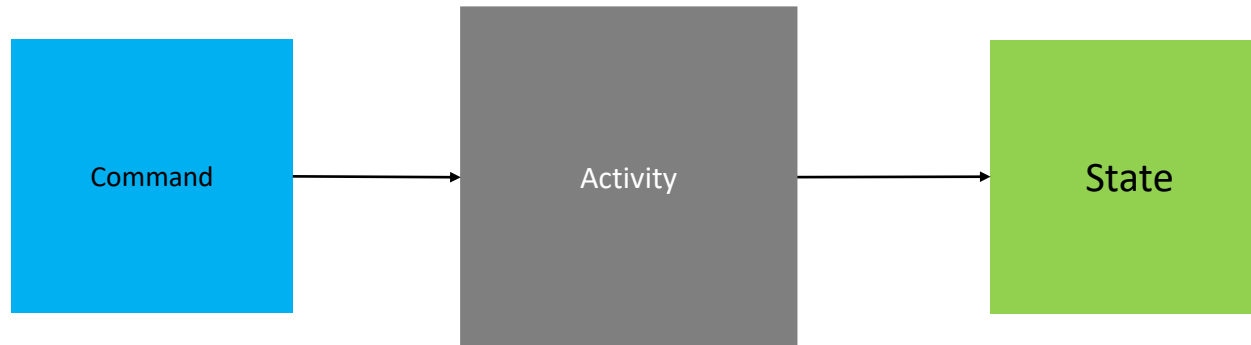
Downstream Activity

Takes action on reality

Downstream activity

Takes action on reality

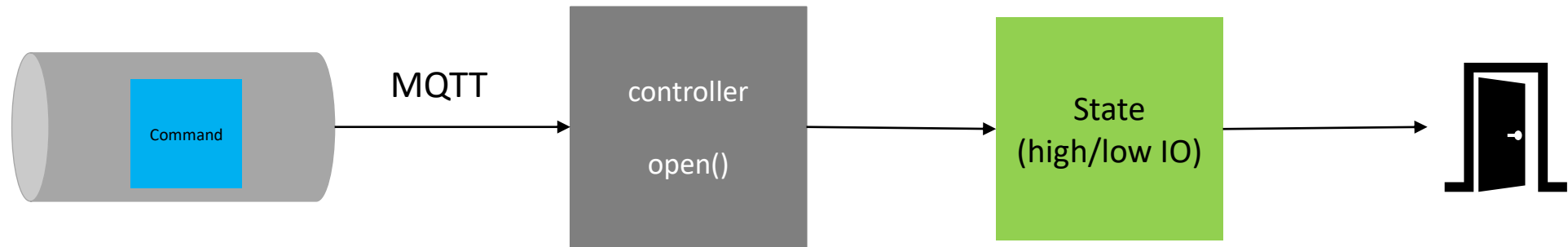
- Use this pattern every time a command needs to result in an immediate state change of the system.



A typical field setup

Activity hosted in a (micro) controller

- Command arrives via a queue (e.g. IoT Hub cloud-to-device messages)
- Activity code hosted in a (micro) controller
- Activity sets IO state of an actuator (e.g. relay) to high or low.



System categories

Line Of Business

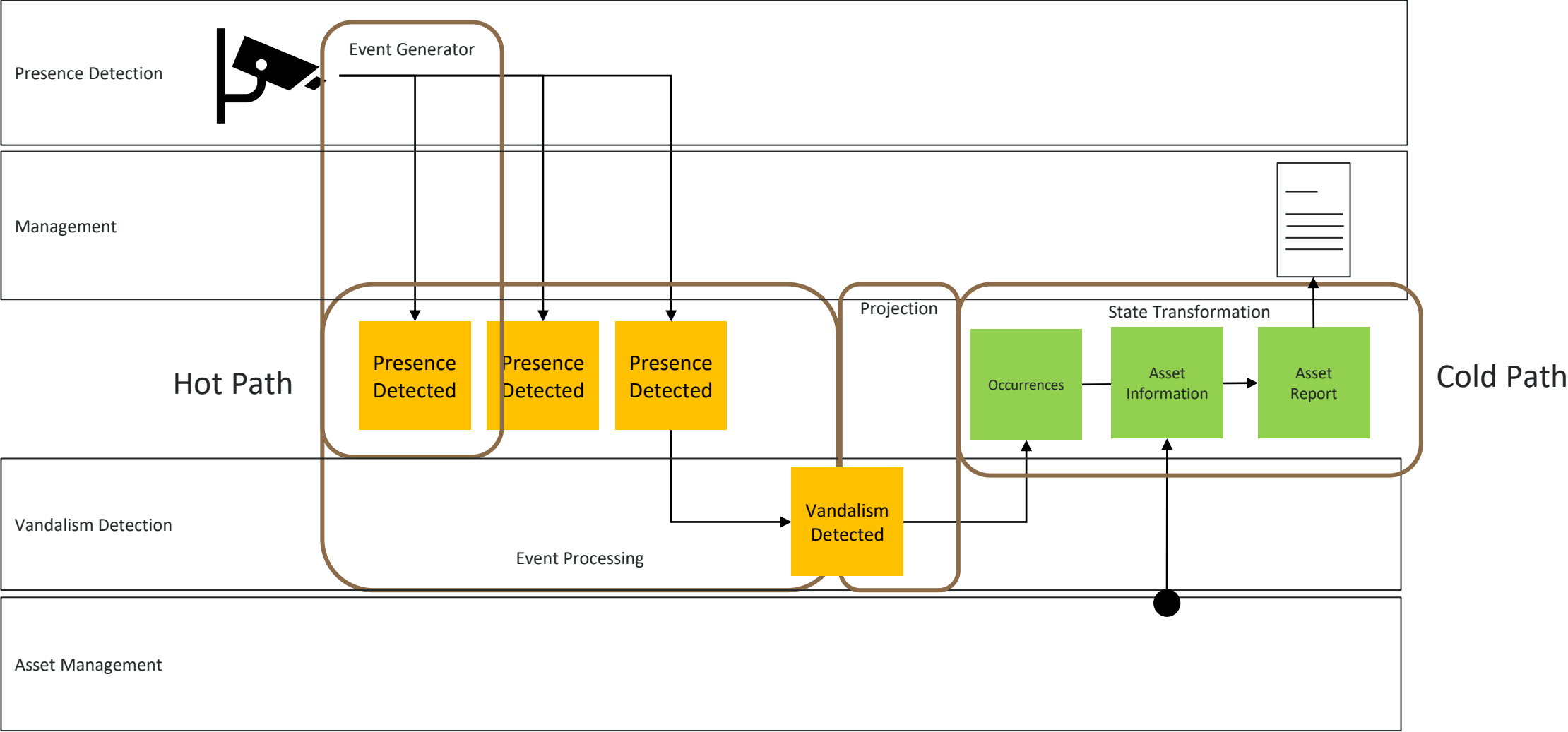
Internet Of Things

Business Intelligence



Business Intelligence

Event Generator -> Event Processing -> Projection -> State Transformation





State Transformation

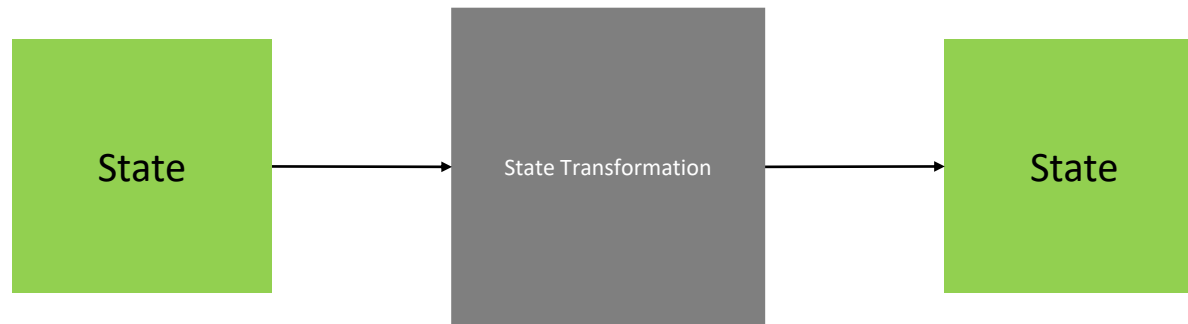
Change the shape of state



State transformation

Change the format of state

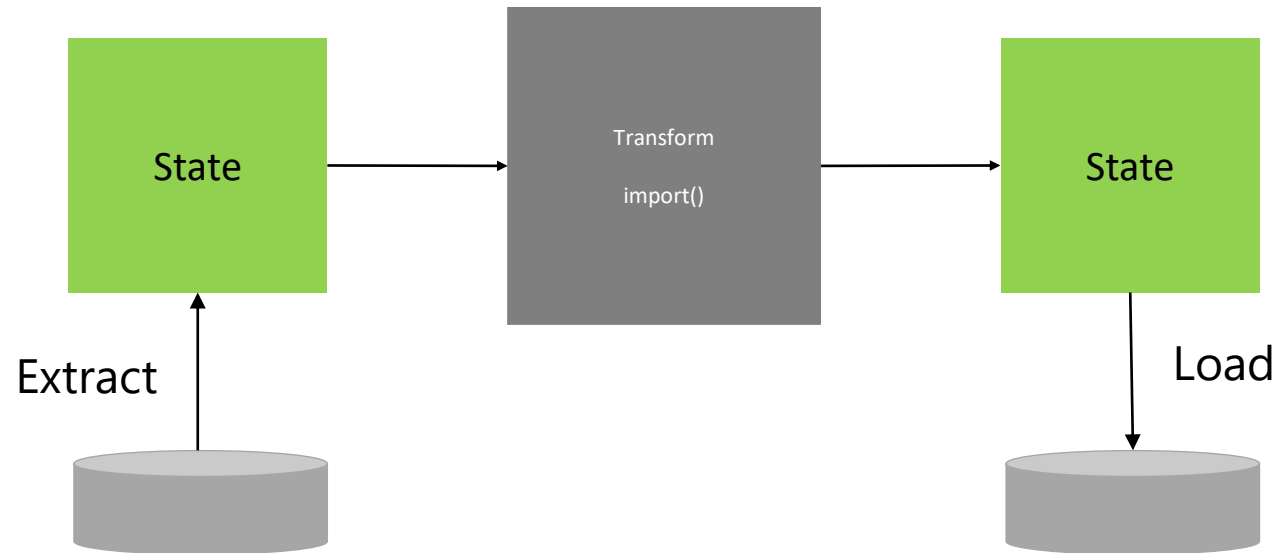
- Use this pattern to store, or show, existing data in a different structure.



Extract transform load

Extract transform load

- Azure Databricks (Java/Scala or Python)
- Azure Data Factory (Visual)
- Roll your own (C#)

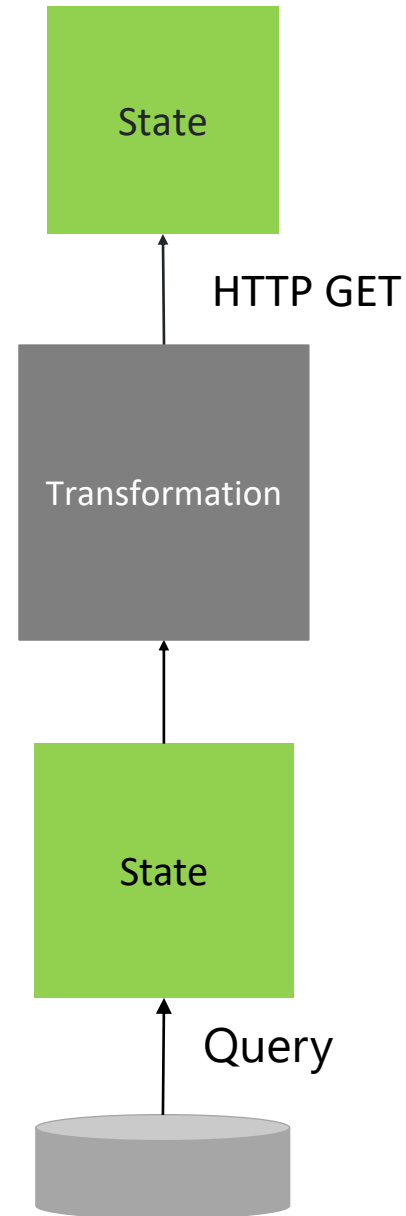


Reporting

Show transformed data in UI

- Dashboards
- Reports
- PDF/Excel/....
- HTTP GET API's

- Are all state transformations



Bonus Integration styles

Orchestration

Choreography



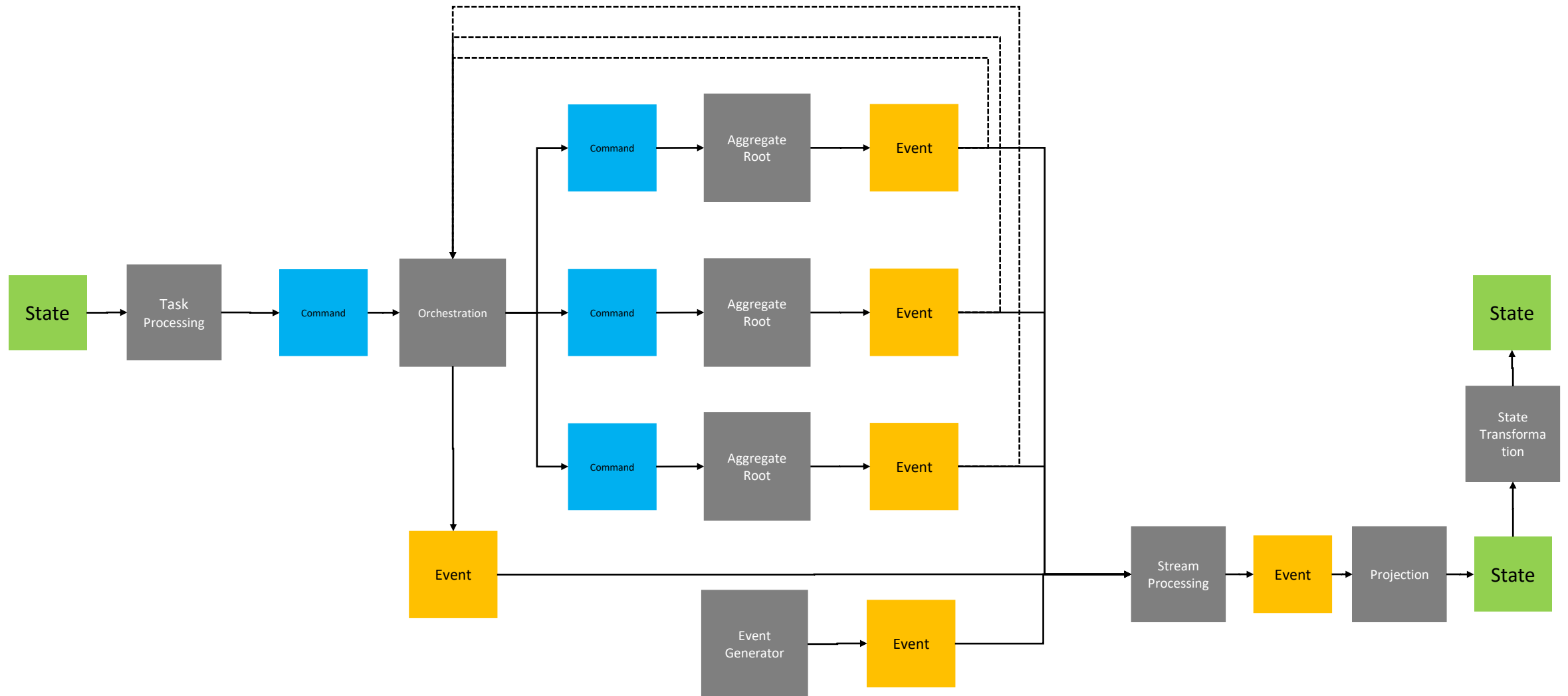


Orchestration

Delegation based integration style

Orchestration

Integration style achieved by chaining delegations



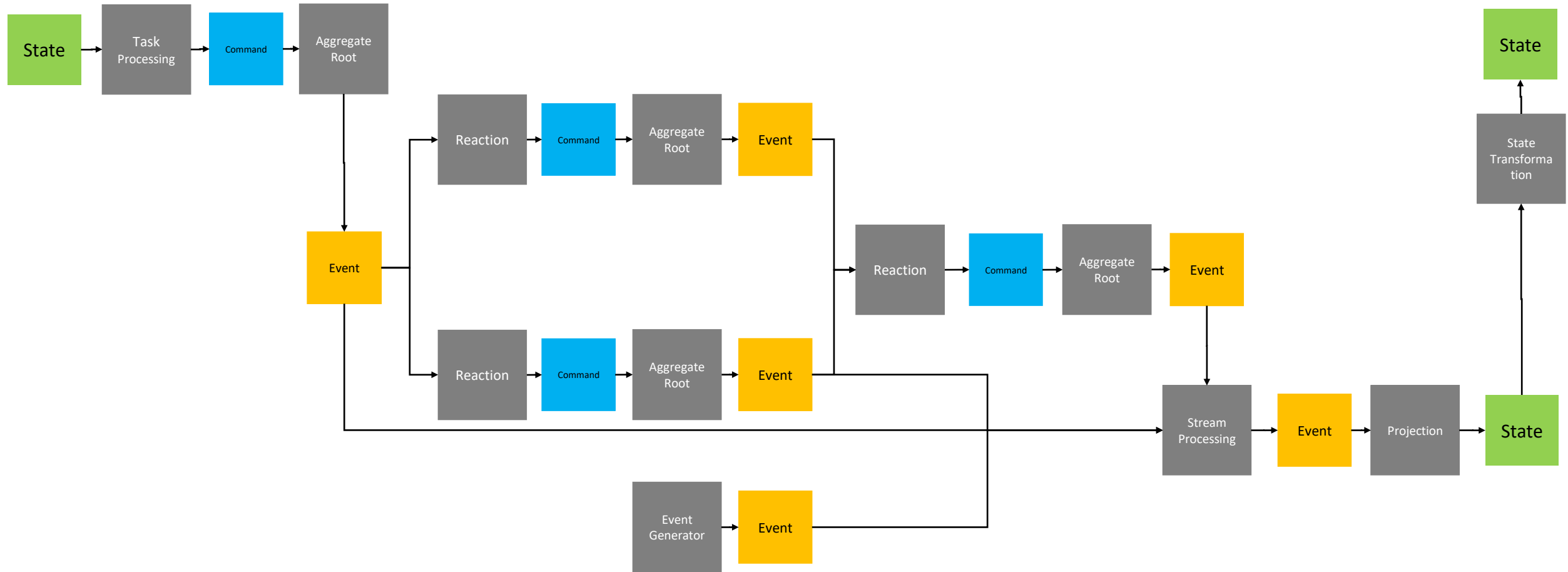
Choreography

Reaction based integration style



Choreography

Integration style achieved by chaining reactions



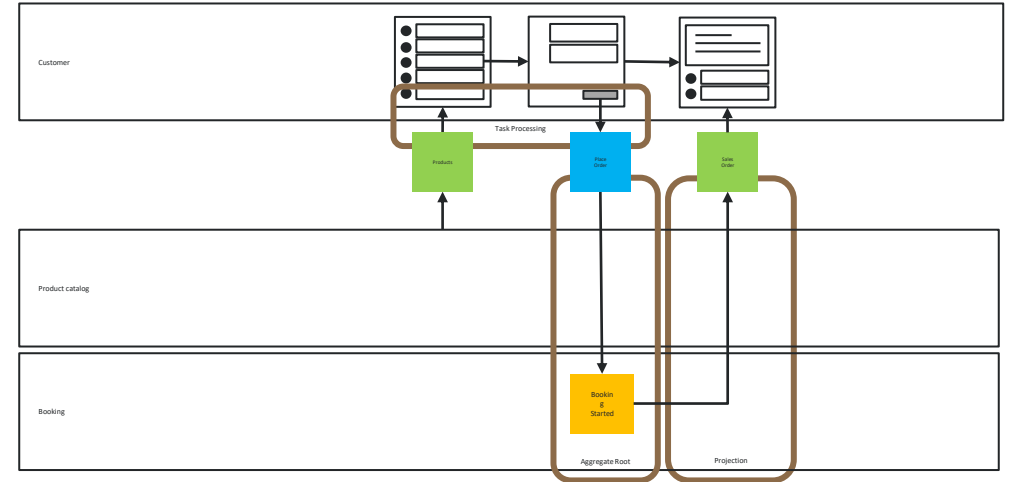


We reached the end of our journey

Key take aways

Heuristics to reason about distributed system design and pattern

- **Focus** on **information flow over time**
 - Not on data structure
- **3 types** of information:
 - In accordance with divisions of time
 - **command, event, state**
- **9 transition patterns**
 - Understand your system category to reduce further
- Feel free to choose your own



| In\Out | Command ■ | Event ■ | State ■ |
|---|---|---|--|
| Command ■ | ? | ? | ? |
| Event ■ | ? | ? | ? |
| State ■ | ? | ? | ? |

Or feel free to use mine

Example code available, in C#, based on MessageHandler 4.0

- Quick starts for each of the presented patterns
- End-to-end scenario for line of business application (e-commerce)



<https://www.messagehandler.net/patterns/>



yves@goeleven.com