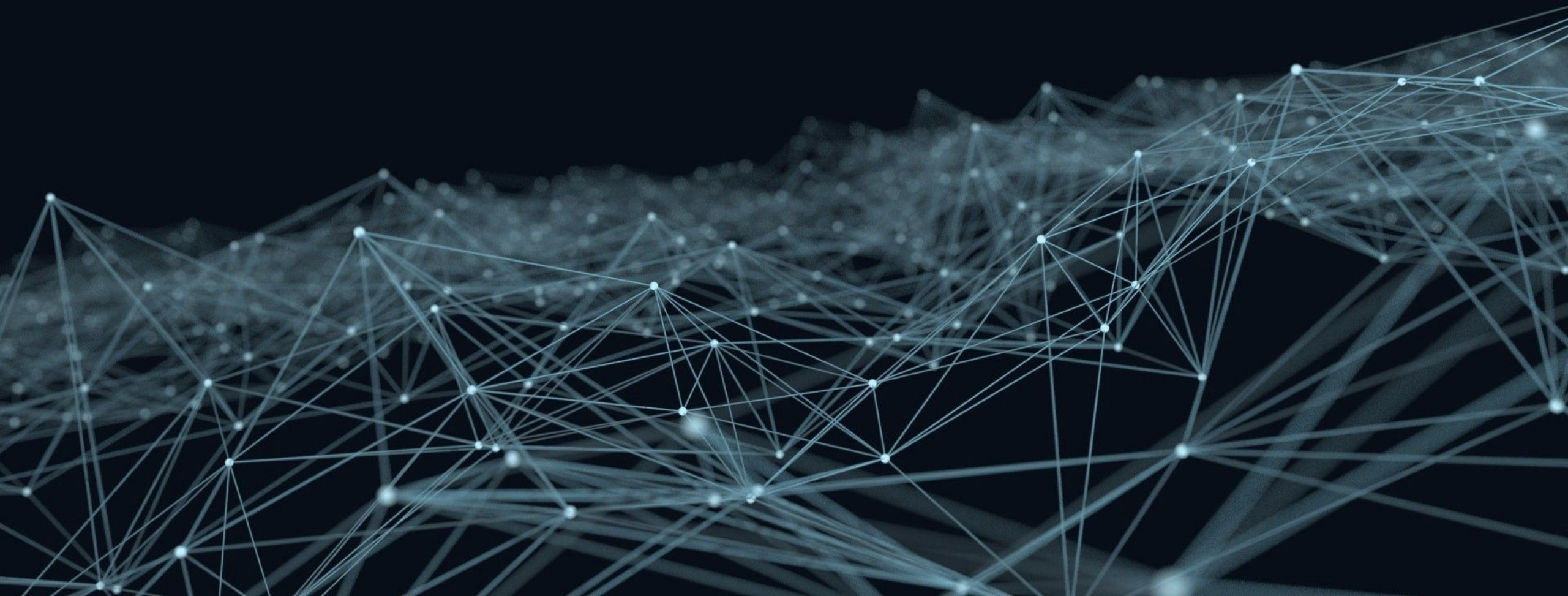
Microservices

Microservice Architecture



Review

- Microservices are a software architectural pattern
- There are specific challenges that they solve effectively
  - These are generally issues around scaling in both the development and operations
  - Microservices are not a “magic” bullet, sometime they are the wrong choice
- Deploying microservices requires
  - Supporting technologies in operations – for example: Kubernetes, Kafka, Docker
  - Analysis and design techniques for building a component based software architecture
  - Code and application design techniques to make code “microservices ready”
  - Production techniques to support the successful deployment of a microservice

Modeling Microservices

- A core principle of microservices is that they model a business domain or a domain of activity
- These domains are often complex and wide ranging in scope
- Very often, they have evolved a domain structure that is
  - A hierarchy of sub-domain layers
  - The structure is recursive
  - Each layer is made up of modular components
  - The components work together by sending well-defined messages
  - Messages are sent through interfaces
  - Components are cohesive and loosely coupled
- This is a sort of idealized version of services “in the wild”
  - There are many factors that can make this organization disfunctional

In Real Life

- A hospital provides a health care service
- Made up of individual microservices
  - Laboratory
  - X-Ray
  - Pharmacy
  - Medical Staffing
- Each microservice:
  - Specializes in a specific domain activity
  - Only that microservice performs that domain activity (the pharmacy doesn't do x-rays for example)
  - Each microservice operates autonomously

In Real Life

- Microservices request services from each other
  - They do not need to know the internal workings of the other microservice
- Requests are made through interfaces
  - Called APIs in software engineering
  - These are often paper based in the real world
  - Requisitions and official forms and paperwork used to make requests of a service
  - Response to a request is often a report of some kind
- Microservices make sense intuitively

COVID-19 VIRUS LABORATORY TEST REQUEST FORM[1]

Submitter information			
NAME OF SUBMITTING HOSPITAL, LABORATORY, or OTHER FACILITY*			
Physician			
Address			
Phone number			
Case definition:[2]	☐ Suspected case ☐ Probable case		
Patient info			
First name		Last name	
Patient ID number		Date of Birth	Age:
Address		Sex	☐ Male ☐ Female ☐ Unknown
Phone number			
Specimen information			
Type	☐ Nasopharyngeal and oropharyngeal swab ☐ Bronchoalveolar lavage ☐ Endotracheal aspirate ☐ Nasopharyngeal aspirate ☐ Nasal wash ☐ Sputum ☐ Lung tissue ☐ Serum ☐ Whole blood ☐ Urine ☐ Stool ☐ Other:		
All specimens collected should be regarded as potentially infectious and you must contact the reference laboratory before sending samples. All samples must be sent in accordance with category B transport requirements.			
Please tick the box if your clinical sample is post mortem ☐			
Date of collection		Time of collection	
Priority status			
Clinical details			
Date of symptom onset:			
Has the patient had a recent history of travelling to an affected area?	☐ Yes ☐ No	Country Return date	
Has the patient had contact with a confirmed case?		☐ Yes ☐ No ☐ Unknown ☐ Other exposure:	
Additional Comments			

Microservices Core Concepts

- Microservices are small, independent, composable services
- Each service can be accessed by way of a well-known API format
  - Like REST, GraphQL, gRPC or in response to some event notifications
- Breaks large business processes into basic cohesive components
  - These basic process actions are implemented as microservices
  - The business process is executed by making calls to these microservices
- Each microservice provides one cohesive service
  - Microservices do not exist in isolation
  - They are part of a larger organization framework

Microservices Core Concepts

- Microservices coordinate with other microservices to accomplish the tasks normally handled by a monolithic application
- Microservices communicate with each other synchronously or asynchronously
- Microservices make application components easier to develop and maintain
- BUT A LOT HARDER TO MANAGE

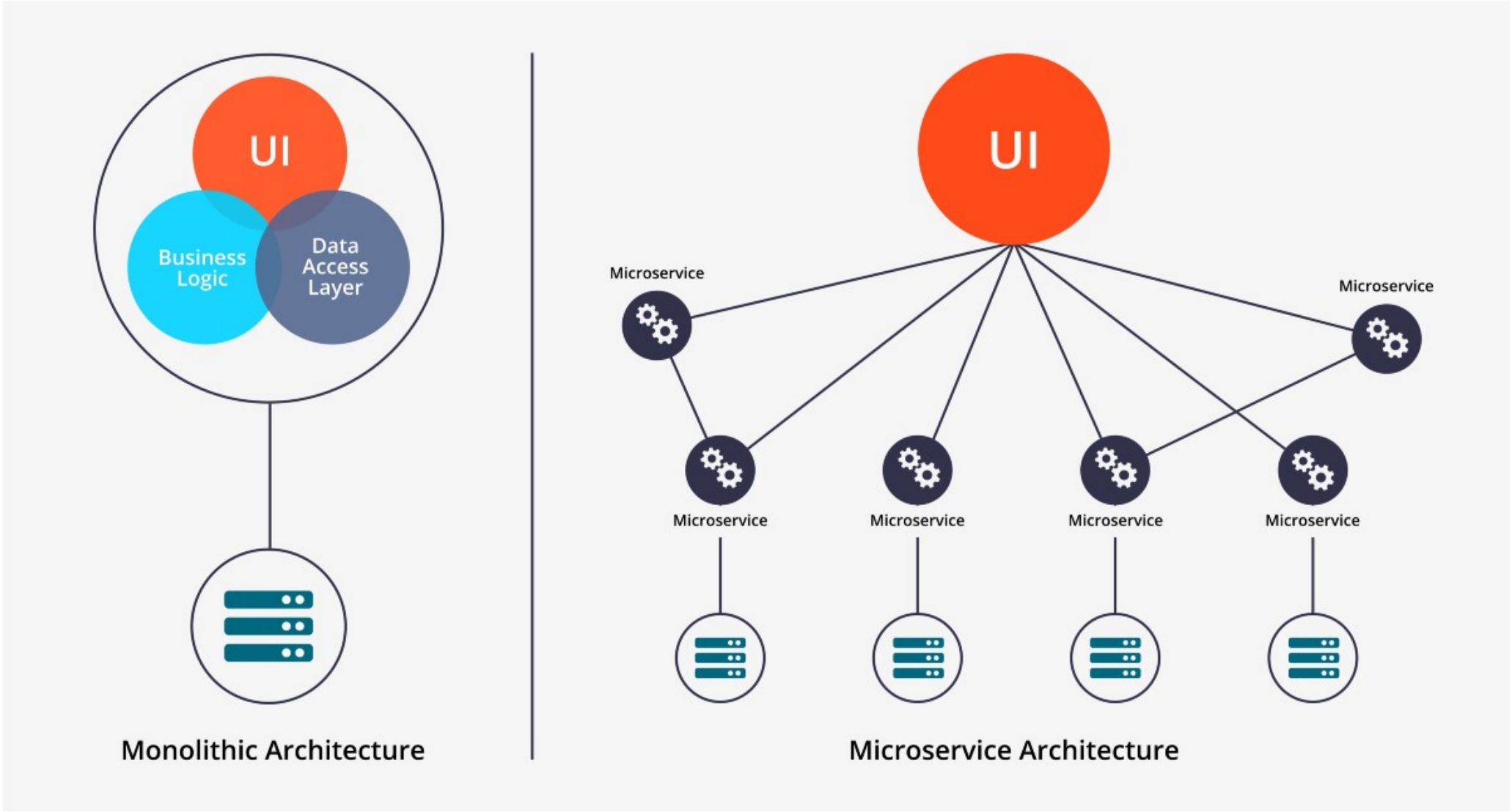
Characteristics of Microservice Deployments

- Light-weight, independent, and loosely-coupled
- Each has its own codebase
- Responsible for a single unit of business functionality
- Uses the best technology stack for its use cases
- Has its own DevOps plan for test, release, deploy, scale, integrate, and maintain independent of other services
- Deployed in a self-contained environment
- Communicates with other services by using well-defined APIs and simple protocols like REST over HTTP
- Responsible for persisting its own data and keeping external state

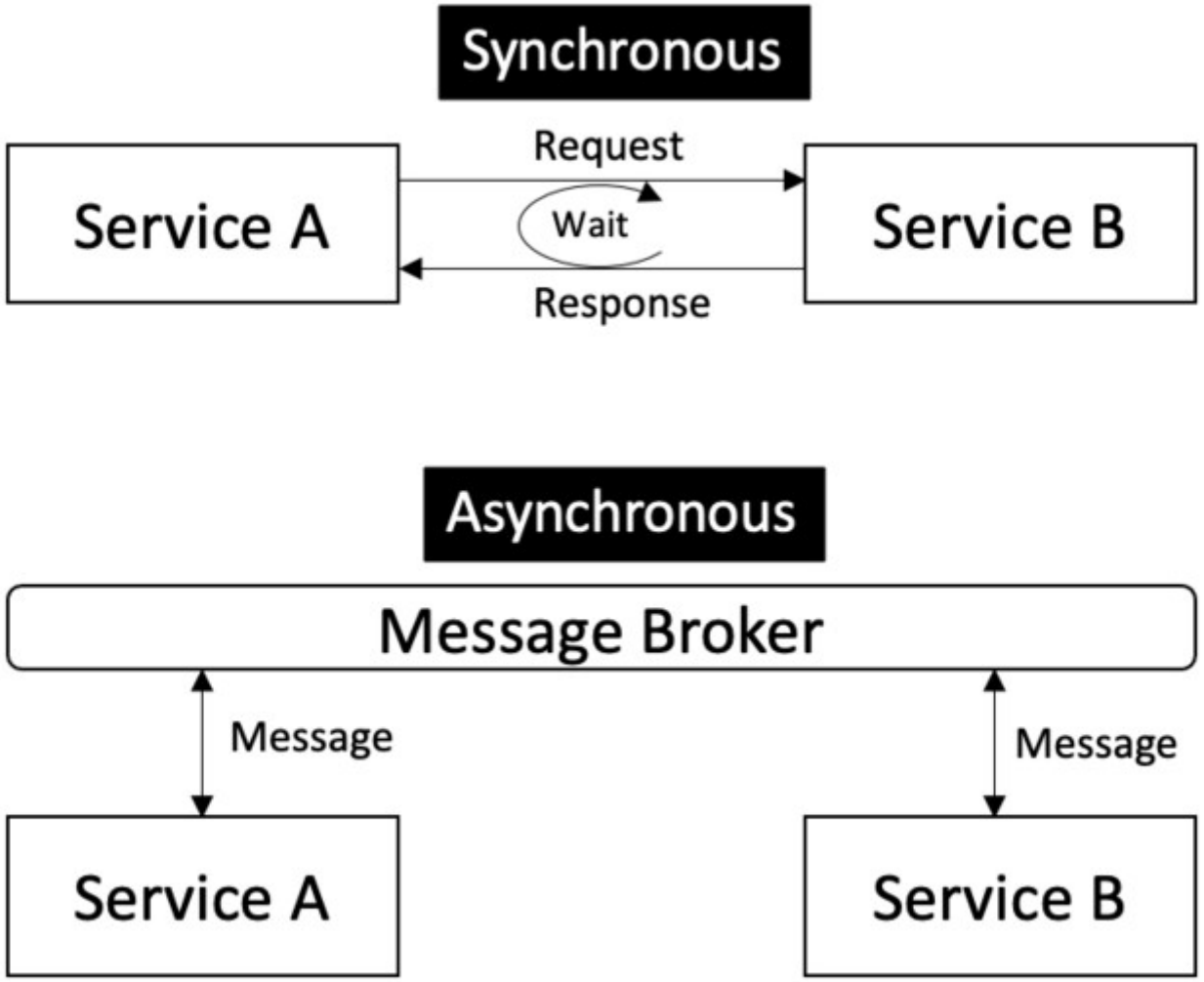
Types of Microservices

- Primary distinction Request versus Event based
- Request based
  - A request is made of a service from a specific client
  - Some sort of response is expected by the client
- Event based
  - The service is responding to a series of events that are occurring somewhere
  - Events don’t expect a response
- These are often referred to as synchronous versus asynchronous
  - This is proving to be an inadequate classification

Monolith to Microservice



Basic Architectural Approaches



Request Based Microservice

- Generally part of a logic flow
- Implementation of a some sort of scenario or process
- For example: Online purchase
- There is a clear workflow
  - User logs in and is taken to home screen
  - User searches for item and selects item
  - User places order
  - Payment is approved
  - Request for shipment is generated and sent to fulfillment
- Each step may be handled by a different microservice
  - Account services, payment service, catalog service, etc.

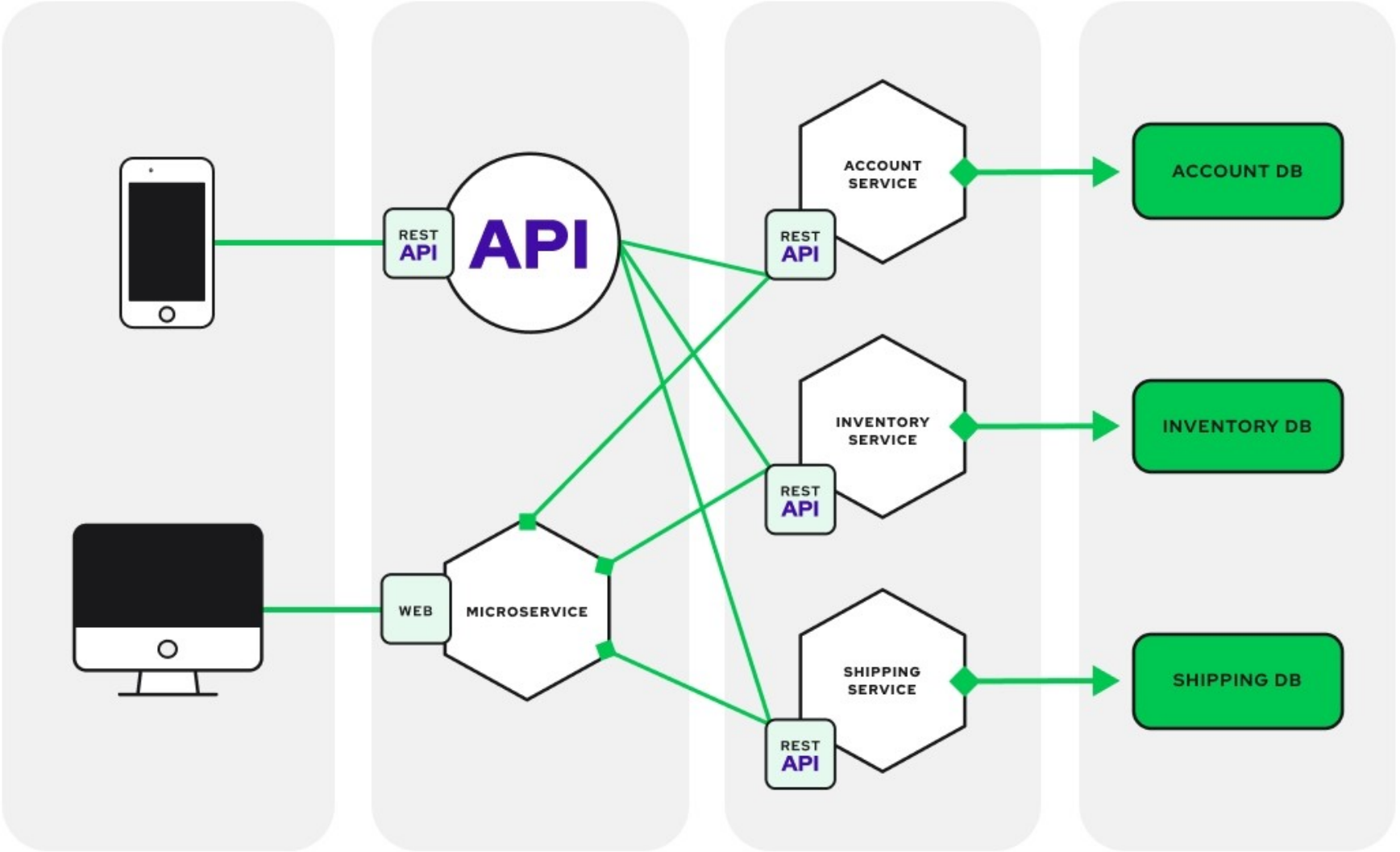
Request Based Microservice

- Often referred to as a synchronous service
- Each request requires a reply
- The service may “block” waiting for a reply’
  - This is the more common use of the term synchronous
- However, request based microservices may be asynchronous
  - Often referred to as “fire and forget”
  - The process can continue while waiting for a reply
- In the sales example
  - The request for shipment can be made
  - But the sale ends without getting a confirmation of the shipment details
  - The confirmation can be send later (maybe email) once the order is processed

A Subtle Distinction

- What do we mean by a reply?
- It could mean that we have the answer to our request
  - “Your payment has been successfully processed, here is your receipt”
  - “You have been authenticated to the system, access token enclosed”
- Or we just have the receipt of our request acknowledged
  - “You order has been received, you will receive a confirmation email later”
  - “Your request has been received and you will receive a ticket number once it has been entered into the system”
- Request based microservices
  - Can be synchronous or asynchronous
  - But tend to be the better option when dealing with synchronous types of processing

Request Based



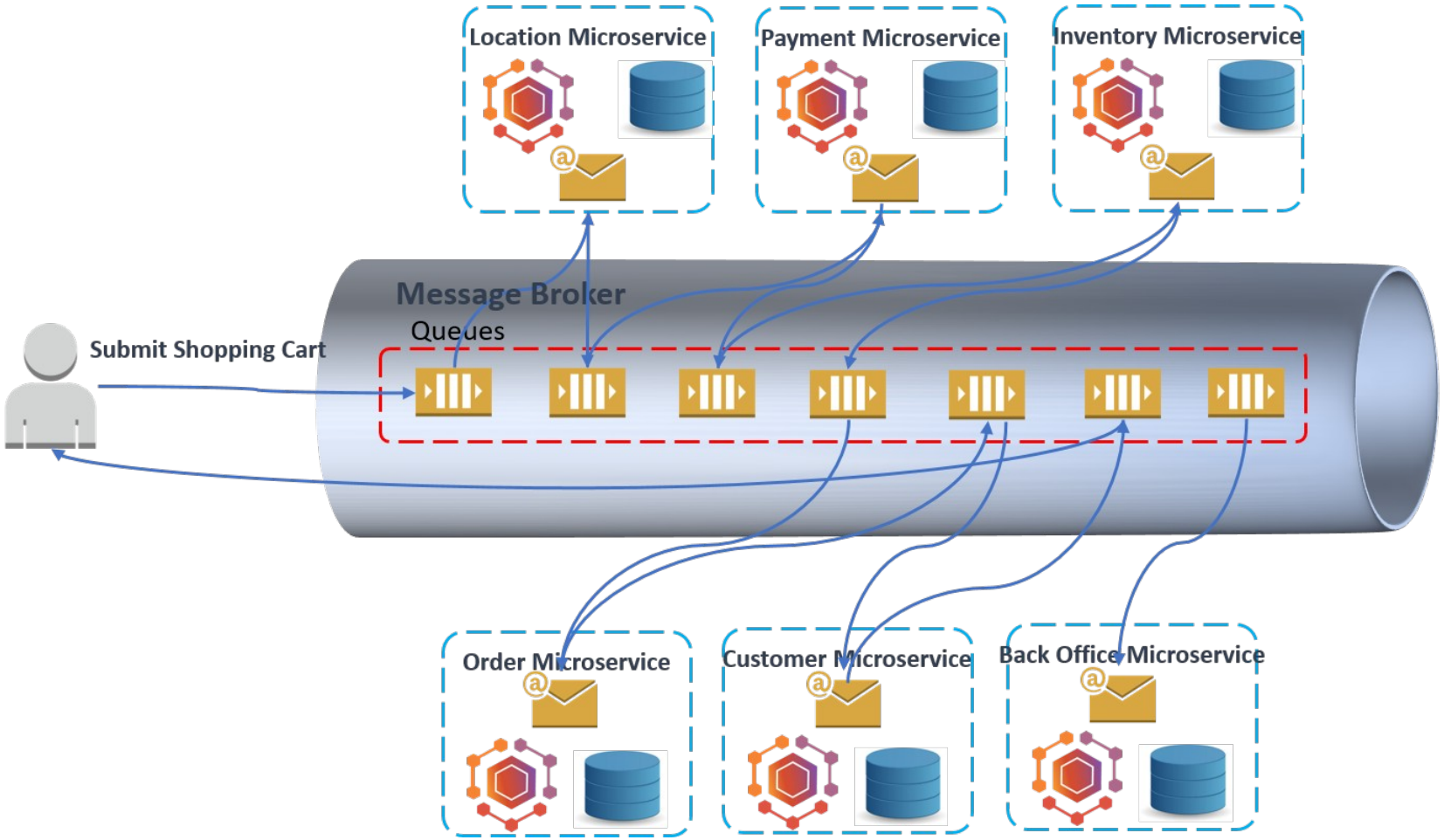
Event Based Microservice

- Responds to a stream of events
- Events may require a response by the system
  - But response may or may not involve a reply to the event source
- Typical use cases
- Event Response
  - Events might occur that require a response that is not a reply to the event source
  - Monitoring heat sensors in a nuclear reactor
  - The response it a reaction by the system to the event, not a reply to the sensor
- Event Streaming
  - Events are just collected and processed for later use
  - Collecting CPS location of cell phones
  - Collecting point of sales data for data analytics
  - No reply or response required

Event Based Microservice

- Event Reply
  - Events may require a reply
  - Sales transaction fraud monitoring
  - Normal sales do not require a reply
  - Fraudulent sales are get a “decline transaction” reply
- The first two cases are clearly asynchronous
  - No reply is generated
- The last case is probably synchronous
  - It often is better designed as request based microservice
  - It depends on how long the asynchronous response window is
  - And whether the transaction has to be blocked or can be reversed later

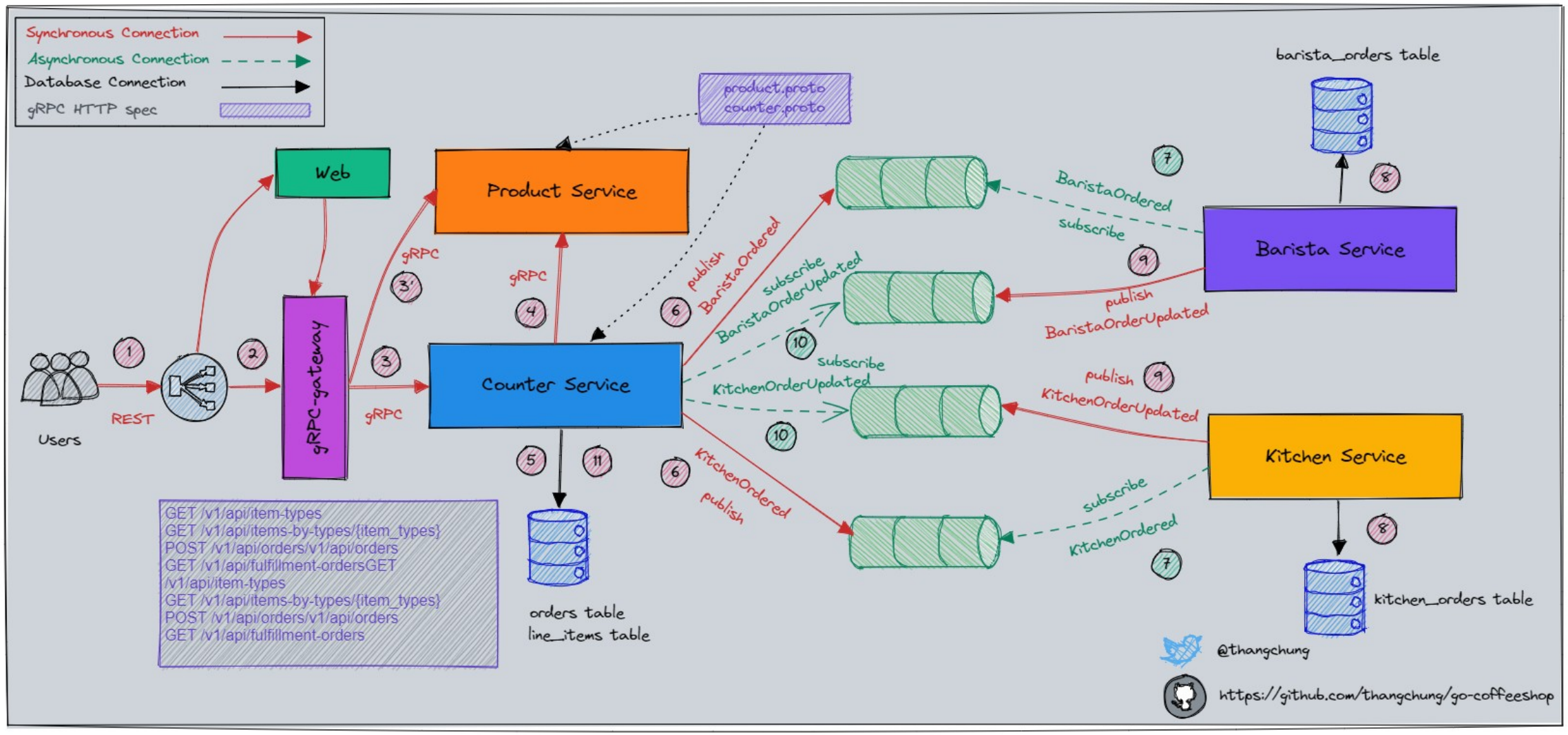
Event Based


Location Microservice
Payment Microservice
Inventory Microservice
Message Broker
Queues
Submit Shopping Cart
Order Microservice
Customer Microservice
Back Office Microservice

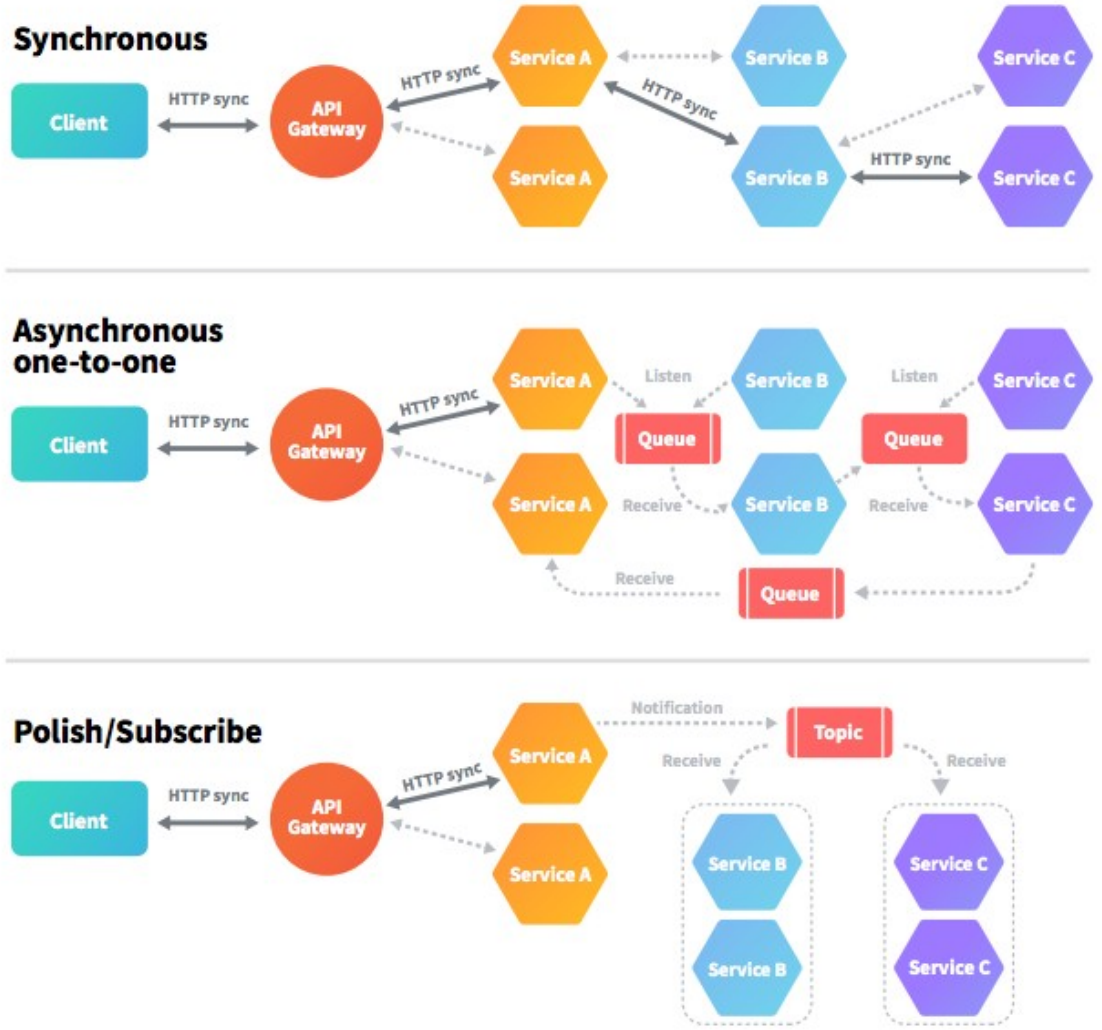
Hybrid Microservice

- Generally, microservice architecture is hybrid
- Parts of the service are message based
- Parts of the service are even based
- Coupling
  - Message based are more tightly coupled – they require the message be sent to an endpoint
  - Event based are more loosely coupled because requests are buffered in a queue

Hybrid Example



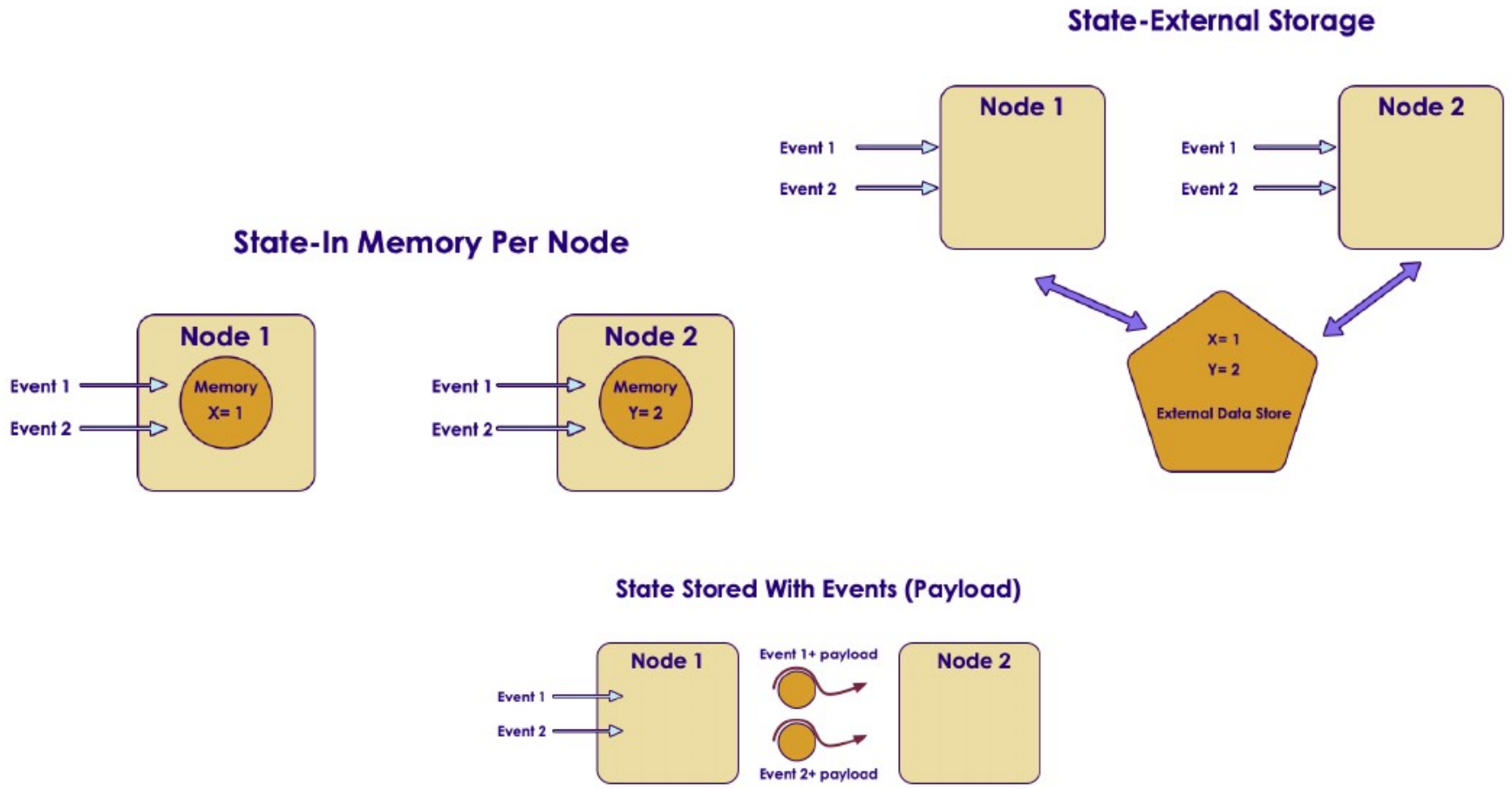
Hybrid Interaction Patterns



Design Consideration – Transactional State

- Does an incoming message require a synchronous response?
- Does the incoming message represent a request or an event?
  - Requests usually kick off a business process or use case
  - Events just happen “out there” and can be dealt with in isolation
  - Events tend to stream from event sources, requests originate from clients
- Do we need to maintain state?
  - Business processes are often require tracking transactional state
  - Events are usually stateless
- What sort of scaling requirements do we have?
  - How we manage transactional state may depend on our scaling requirements

State Representation



End of Module