RISK AND RESILIENCE BOOTCAMP

TEKsystems Global Services

WORKFORCE DEVELOPMENT

OPERATIONS

This section is a more detailed look the IT operational environment from a risk perspective

- Business
- Technology
- Data
- Cyber

OPERATIONS

- Operations is where systems, data, and users converge
  - Delivers the business value to end users and customers
  - In critical sectors, operational stability creates business resilience and customer trust
  - Failures in production environments can escalate quickly into
    - Financial loss
    - Regulatory exposure, and reputational damage
  - Operations monitoring and production support ensure that
    - Systems perform within expected parameters
    - Anomalies are detected early
    - Incidents are handled efficiently
    - Services are restored quickly and reliably
  - Operational resilience, as defined by DRII:
    - *"The ability of an organization to continue to deliver its products and services at acceptable predefined levels following a disruptive incident"*

OPERATIONS

- The operational environment
  - Production systems are the live environments supporting business services
    - For example: online banking, payment gateways, trading platforms
  - Pre-production environments (Dev, QA, Staging) exist to test and validate before release.
  - Operations teams manage and monitor the production stack continuously
    - Often called Site Reliability Engineering (SRE) or IT Operations Center Management
  - Key goals of operations
    - *Availability:* keeping systems up and running 24/7
    - *Performance:* maintaining responsiveness under varying loads
    - *Integrity:* ensuring no data corruption or unauthorized change
    - *Reliability:* predictable, stable service delivery over time
    - *Recoverability:* to restore service promptly after incidents

OPERATIONS TECHNOLOGIES

- The technology stacks are always evolving
  - New technology is always being developed
    - For example: Internet, more powerful processors, larger data storage capacities
  - New uses for technology emerge
    - For example: on-line banking
- The historical phases of operational systems
  - Legacy monolith systems – usually mainframes
  - Distributed client-server systems often based on web technologies
  - Microservices, characteristic of modern systems
  - Cloud services which can provide an operations environment

OPERATIONS TECHNOLOGIES

- All stacks have to supply common core functions
  - *Incident management*
    - Logging, classifying, triaging, and resolving incidents
  - *Problem management*
    - Analyzing root causes and recurring faults to prevent repetition
  - *Change management*
    - Ensuring system changes are tested, reviewed, and implemented safely
  - *Configuration management*
    - Maintaining a configuration management database to track and document dependencies within the Ops environment
  - *Release management*
    - Controlling deployment into production with rollback and verification steps

MONOLITH APPLICATION

Characterized by a single code base

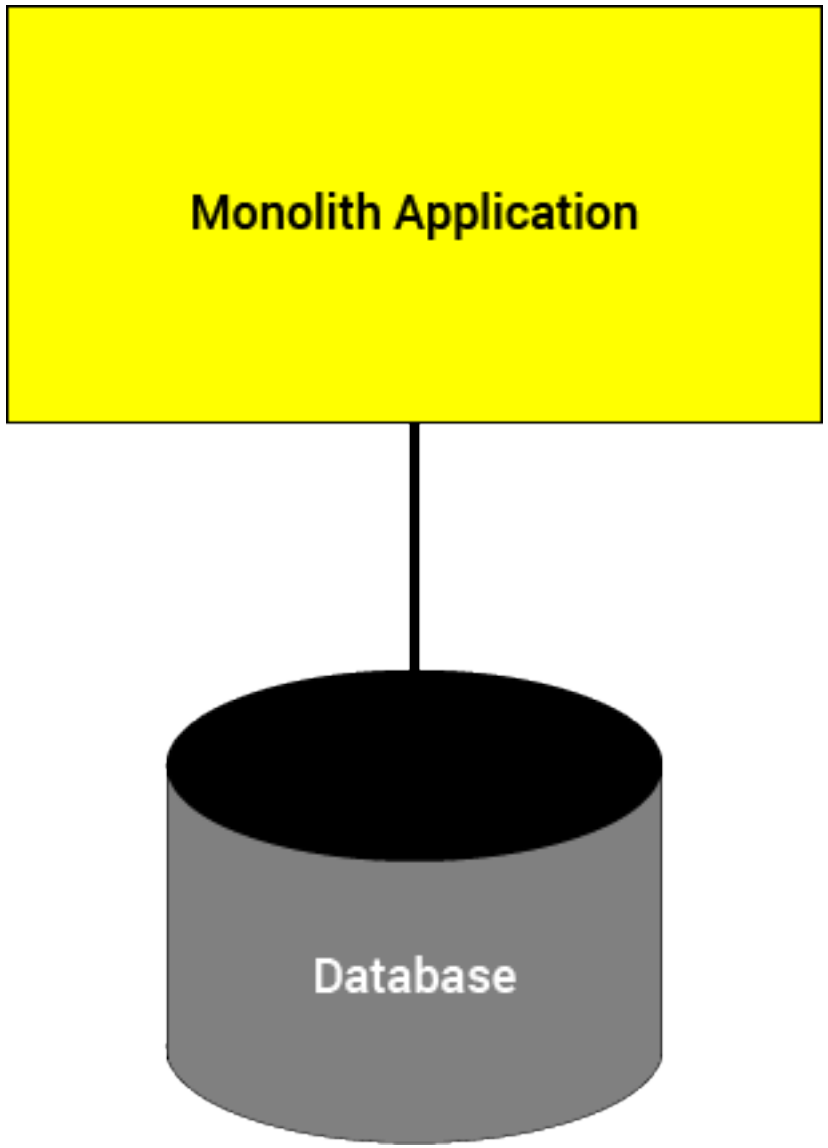
- May be modular at the programming language level

Integrated with a single database

- All code uses a common schema

“Monolith” means

- If a change to the code base or to the data schema
- Then the entire application needs to be redeployed



BUSINESS ANALOGY

Start-up businesses are monoliths

- There are a few people who do everything

The enterprise is small enough that this model works

- It's actually counterproductive to have a highly structured departmental organization with just a few employees

Early versions of applications are similar

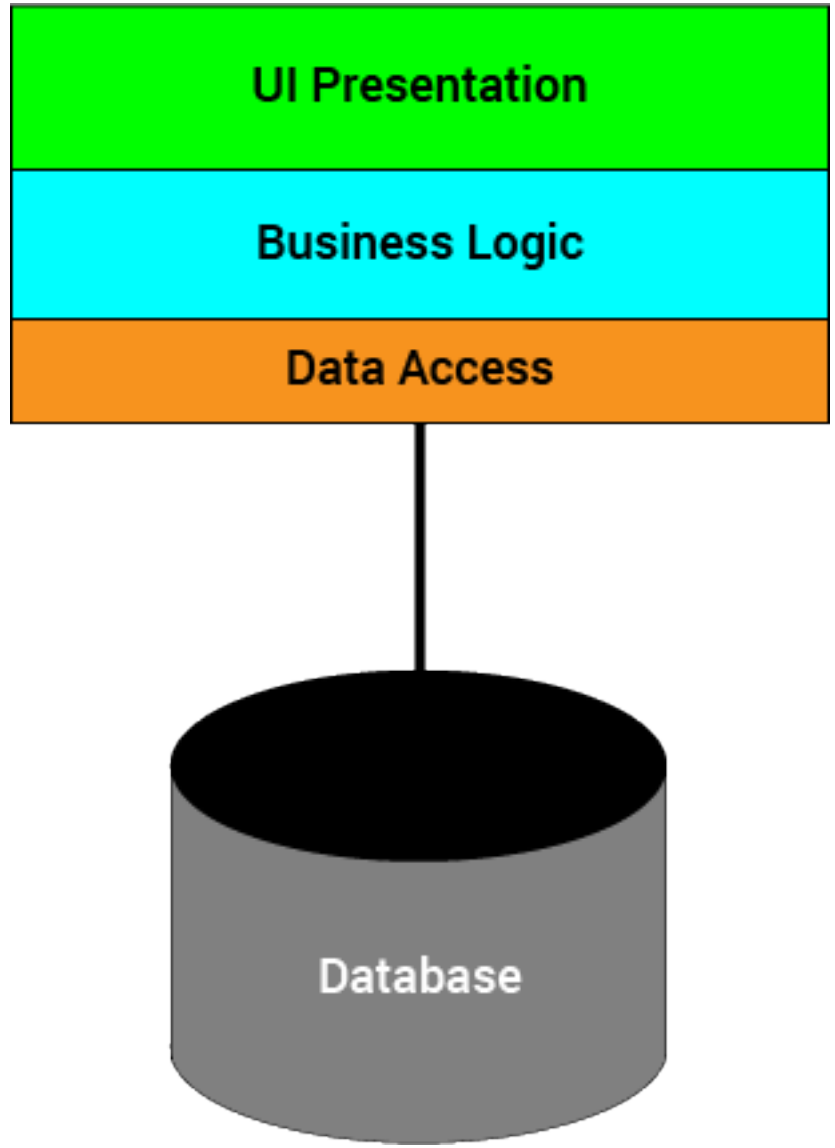
- Simple enough that all of the code is manageable
- Single code base for the whole app
- Flat or minimal architectural structure

MODULAR MONOLITH

Code is modularized
Organized along job descriptions

- Front end dev has their modules
- Programmers have their modules
- Data engineers have their modules

Shows up historically as a n-tier architecture



BUSINESS ANALOGY

As the start-up grows, it adds more people
The original organization starts to become ineffective

- The entrepreneurial “all hands-on deck” model doesn’t scale well
- Processes become chaotic
- Difficult to manage
- Productivity stalls

At some point the business must modularize

- Usually by creating specialized departments
- Accounting, sales, HR, etc.

MODULAR MONOLITH

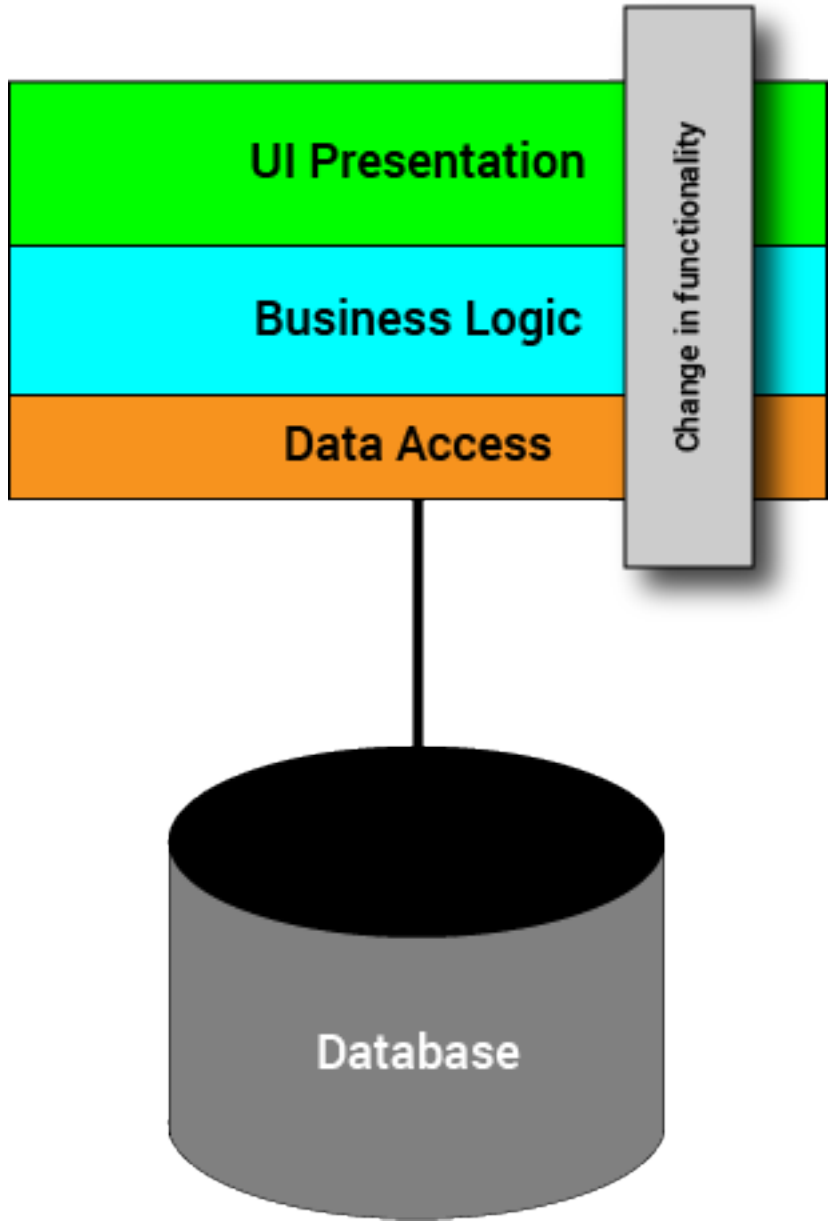
The application is still a monolith
A change in functionality

- Affects each layer because related changes have to be made in each layer

Interacts with the data model

- The data model may constrain what changes can be made
- Changing the data model might break other parts of the app

The modules and the database often show high coupling, due to how the modules are defined



PROS AND CONS OF MONOLITHS

- Pros
  - Simple to develop
  - Simple to test
  - Simple to deploy
  - Simple to scale horizontally
    - Run multiple copies behind a load balancer
  - Although persistence is problematic when using horizontal scaling
- Cons
  - The size of the application can slow down the start-up time
  - The entire application must be redeployed on each update
  - Monolithic applications can also be challenging to scale vertically
  - Reliability issues – easy to crash the whole application
  - Difficult to migrate to new technologies

SCALING IN SYSTEMS

- Scaling is an increase in size or quantity along some dimension
- Can take place in the development or operations space
- Development scaling
  - Increase in complexity, functionality or volume of code
  - These dimensions are often related
  - Business analogy is a company increasing the range of services and products they offer or expanding into different markets (like a Canadian company expanding into Europe)
- Operational scaling
  - Increase in the amount of activity of a system
  - Throughput, load, transaction time, simultaneous users, etc
  - Traditional monoliths tend to be scalable only to a limited degree
  - There is a certain level of complexity after which they become unmaintainable

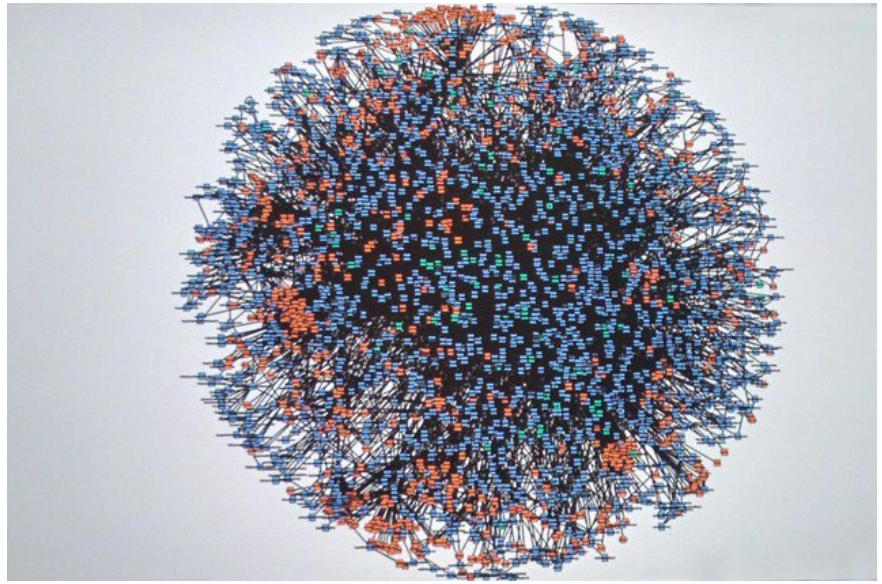
OPERATIONAL COMPLEXITY

Modern applications have to deal with

- Petabytes of streaming data
- Billions of transactions
- Mission critical fault tolerance
- Non-functional requirements
- Throughput, response time, loading, stress
- Disaster recovery, transaction time

Results in a “death star” architecture

- Image: Amazon in 2008
- The complexity of the operational architecture is now industrial strength



IT FAILURES AND COMPLEXITY

The United States is losing almost as much money per year to IT failure as it did to the [2008] financial meltdown. However the financial meltdown was presumably a onetime affair. The cost of IT failure is paid year after year, with no end in sight. These numbers are bad enough, but the news gets worse. According to the 2009 US Budget [02], the failure rate is increasing at the rate of around 15% per year.

Is there a primary cause of these IT failures? If so, what is it?.... The almost certain culprit is complexity.... Complexity seems to track nicely to system failure.

Once we understand how complex some of our systems are, we understand why they have such high failure rates.

We are not good at designing highly complex systems. That is the bad news. But we are very good at architecting simple systems. So all we need is a process for making the systems simple in the first place.

Roger Sessions

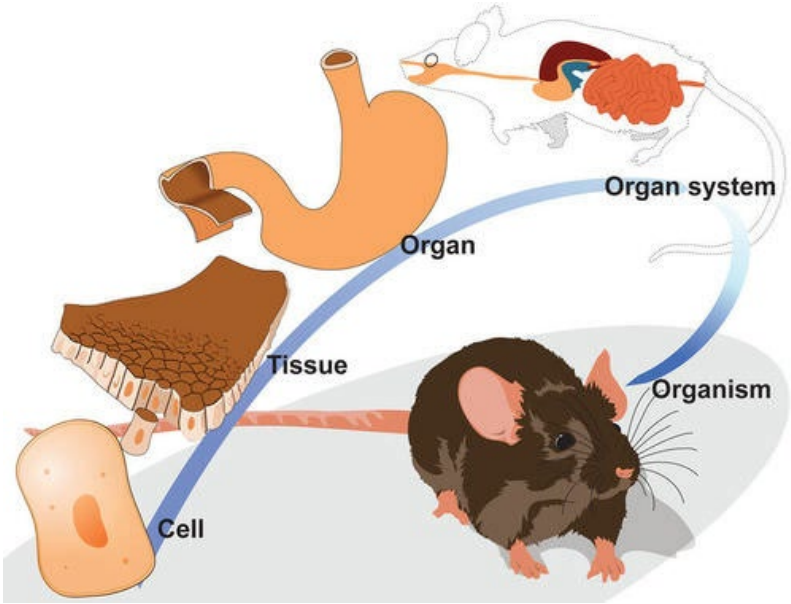
NATURAL SYSTEMS ORGANIZATION

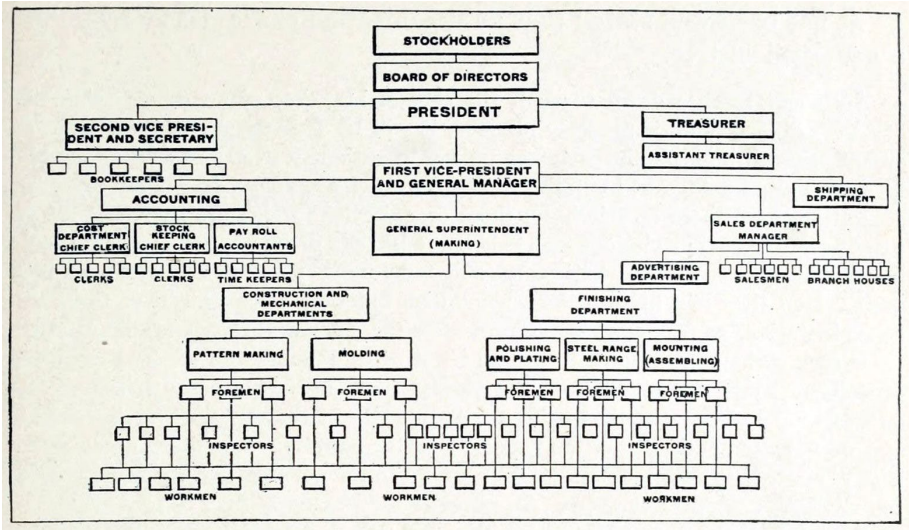
Naturally occurring systems scale successfully

- Biological processes, social organization etc.
- They all show similarities in organization regardless of domain

Natural occurring systems tend to be

- Recursive in structure
- Hierarchical or layered
- Modular at each layer
- Loosely coupled and highly cohesive





OPERATIONAL COMPLEXITY IN PRACTICE

In production, systems have to scale operationally

Consider a diner

- During the lunch and dinner hour rush, the number of servers and cooks have to scale up
- There has to be clear boundaries on what each role does
- During non-peak hours, the amount of staff can be scaled down

OPERATIONAL COMPLEXITY IN PRACTICE

Operations are specialized

Tasks are broken down into sub-tasks, and sub-tasks broken down into sub-sub-tasks, and so on

At some point

- Specialized systems or agents perform an individual sub-task
- These are all coordinated

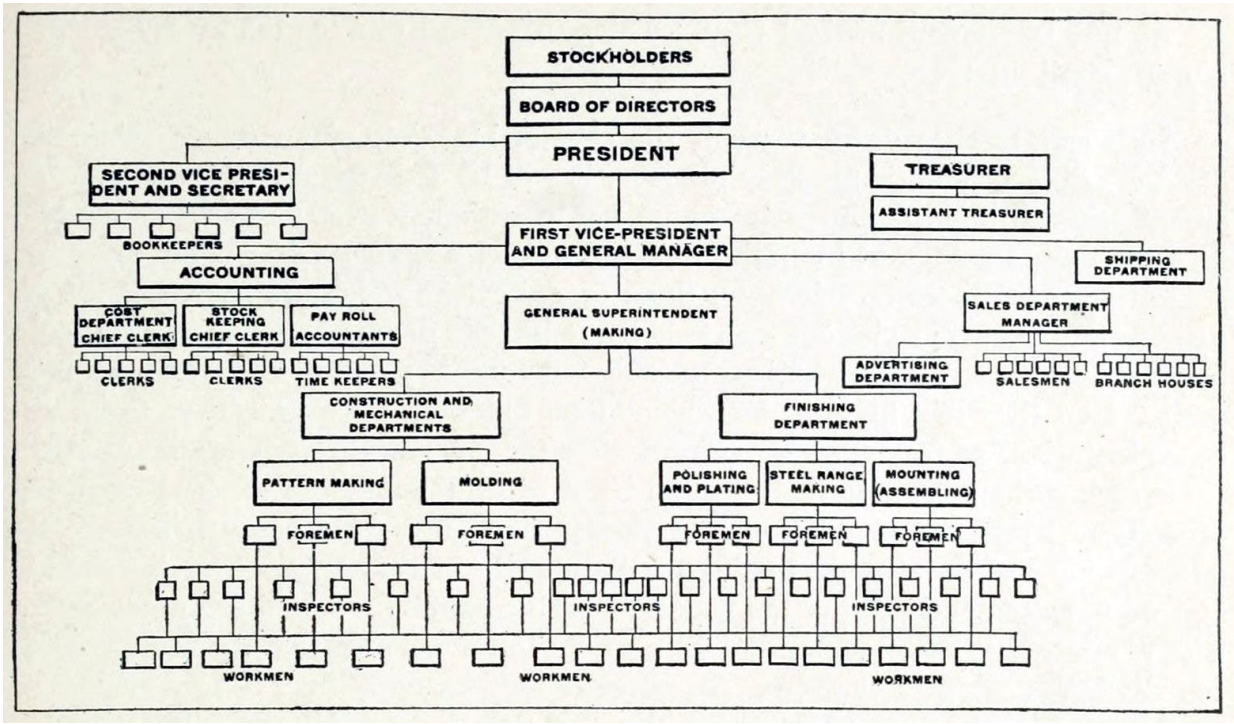
For example, the specialized positions on a sports team

COMPLEX SYSTEMS

Frequently, complexity takes the form of a hierarchy, whereby a complex system is composed of interrelated subsystems that have in turn their own subsystems, and so on, until some lowest level of elementary components is reached

Courtois

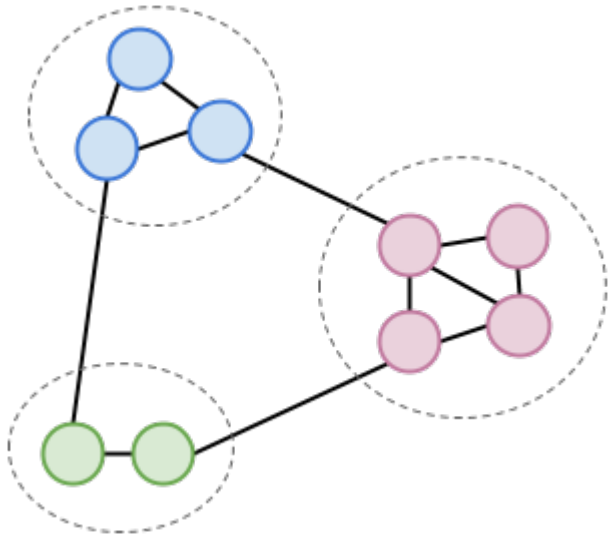
On Time and Space Decomposition of Complex Structures
Communications of the ACM, 1985, 28(6)



COMPLEX SYSTEMS

Intra-component linkages are generally stronger than inter-component linkages. This fact has the effect of separating the high frequency dynamics of the components – involving the the internal structure of the components – from the low frequency dynamics – involving the interaction among components

Simeon



COMPLEX SYSTEMS

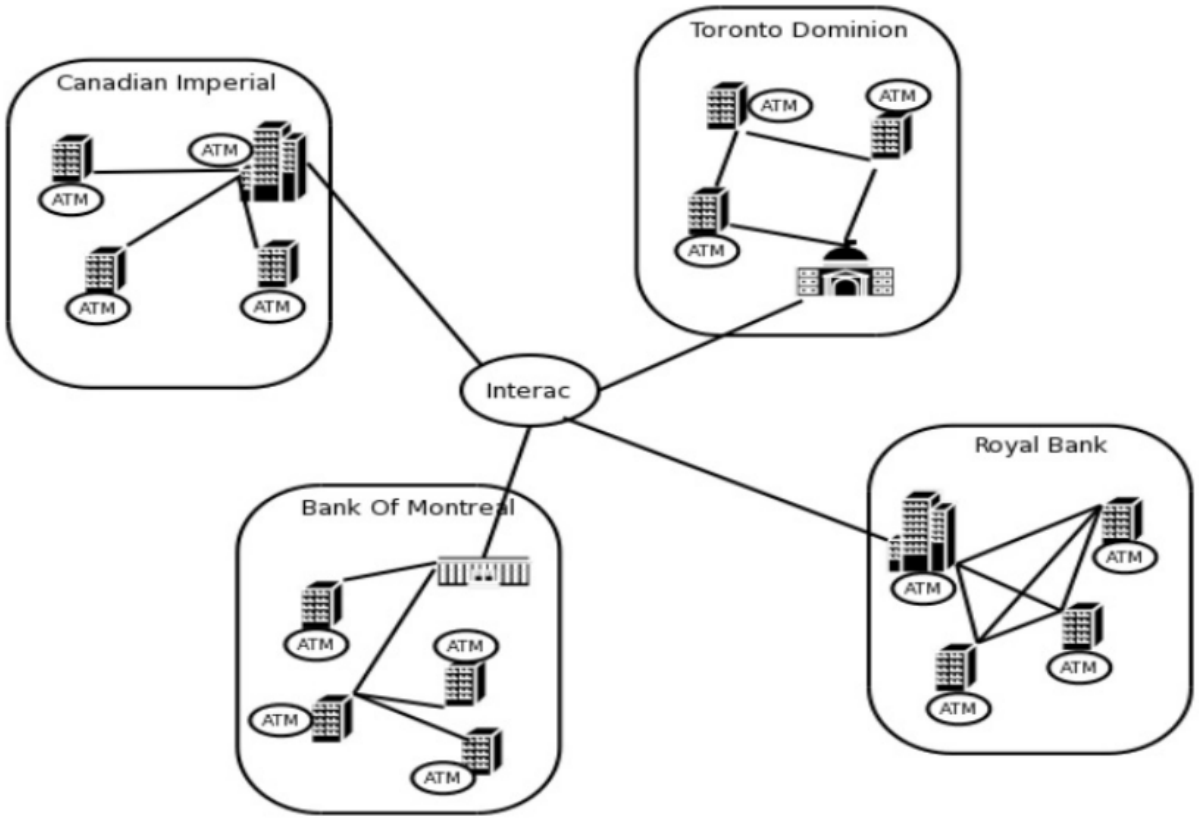
Hierarchical systems are usually composed of only a few different kinds of sub-systems in various combinations and arrangements

Simeon

A complex system that works is invariably found to have evolved from a simple system that worked. A complex system designed from scratch never works and can never be patched up to make it work. You have to start over, beginning with a simple working system.

John Gall

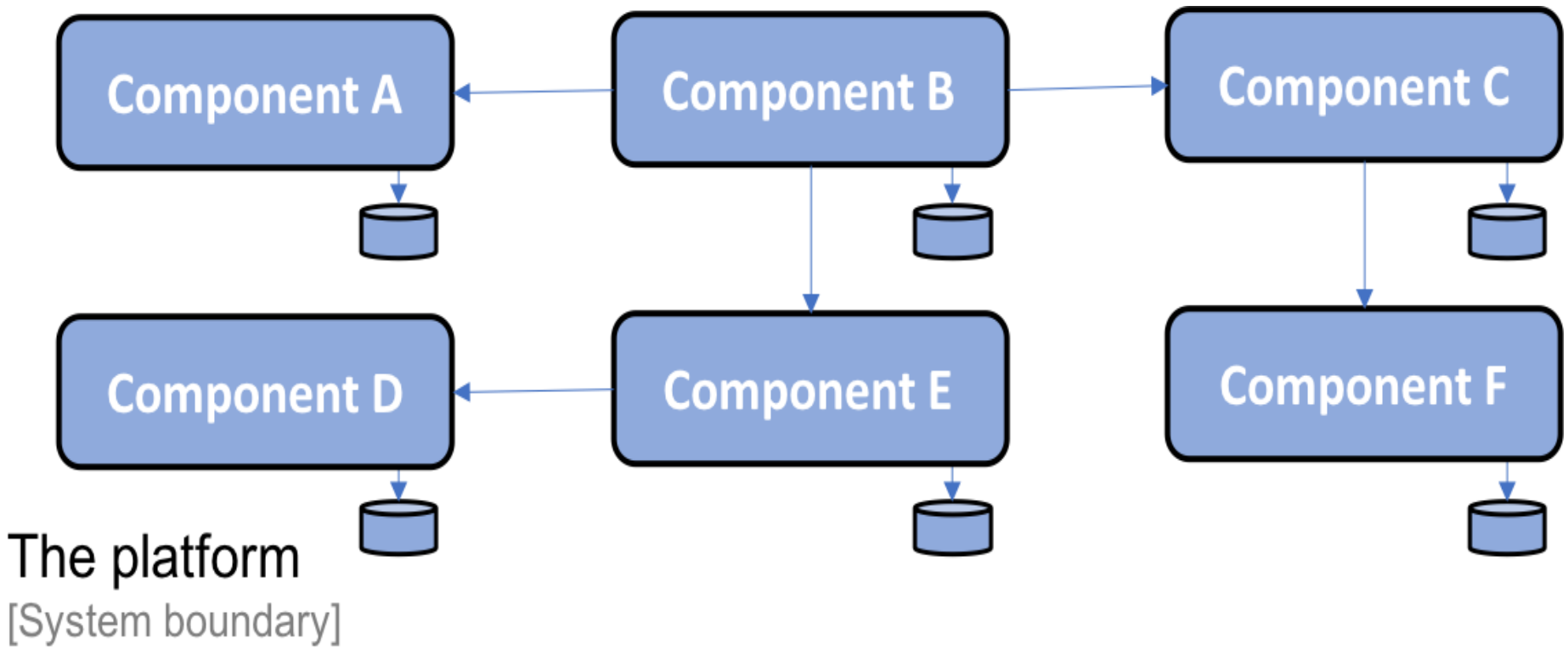
Systemantics: How Systems Really Work and How They Fail
1975



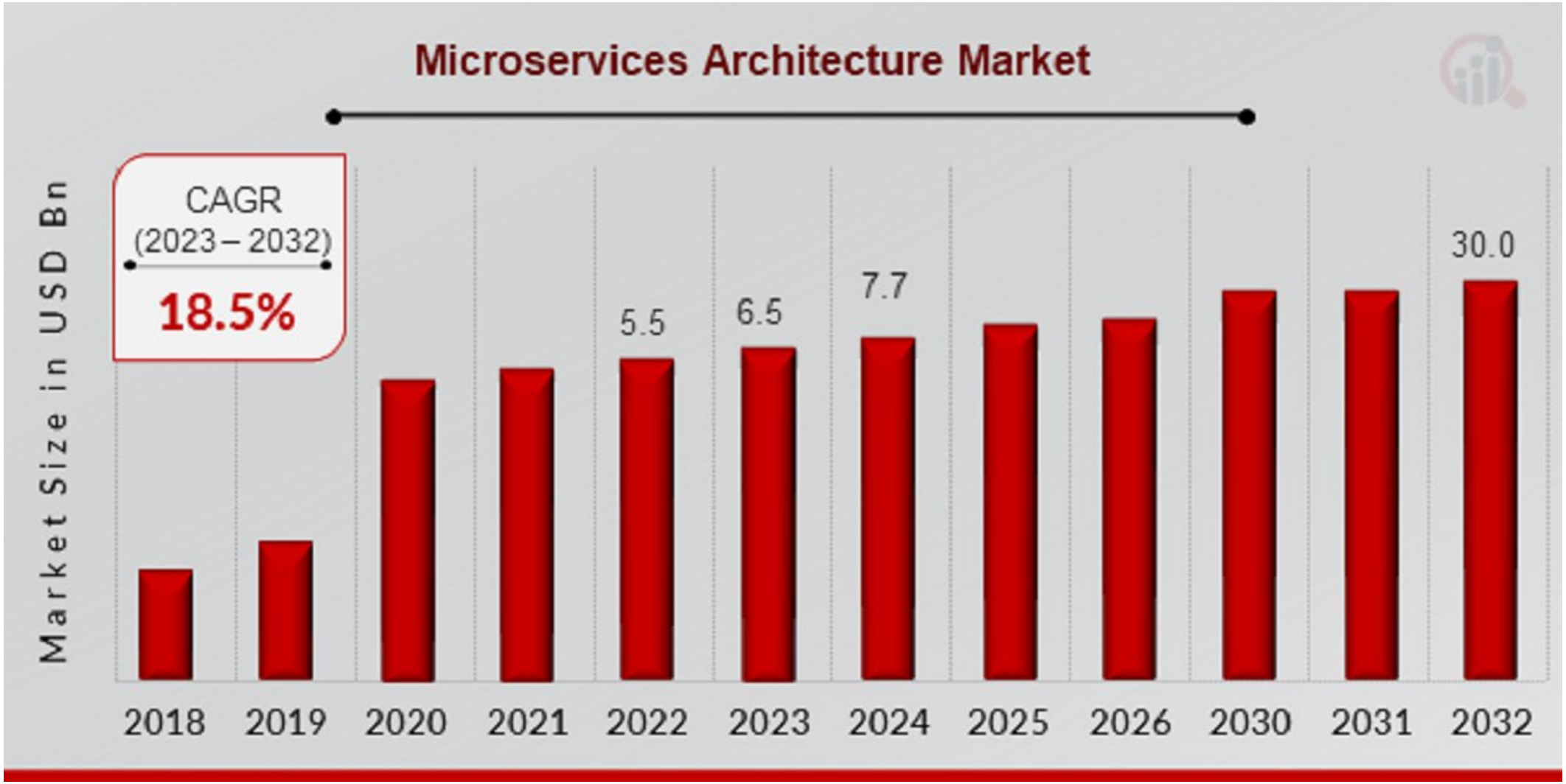
MICROSERVICES

"The Microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API."

MICROSERVICES



MICROSERVICES



MICROSERVICES DRIVERS

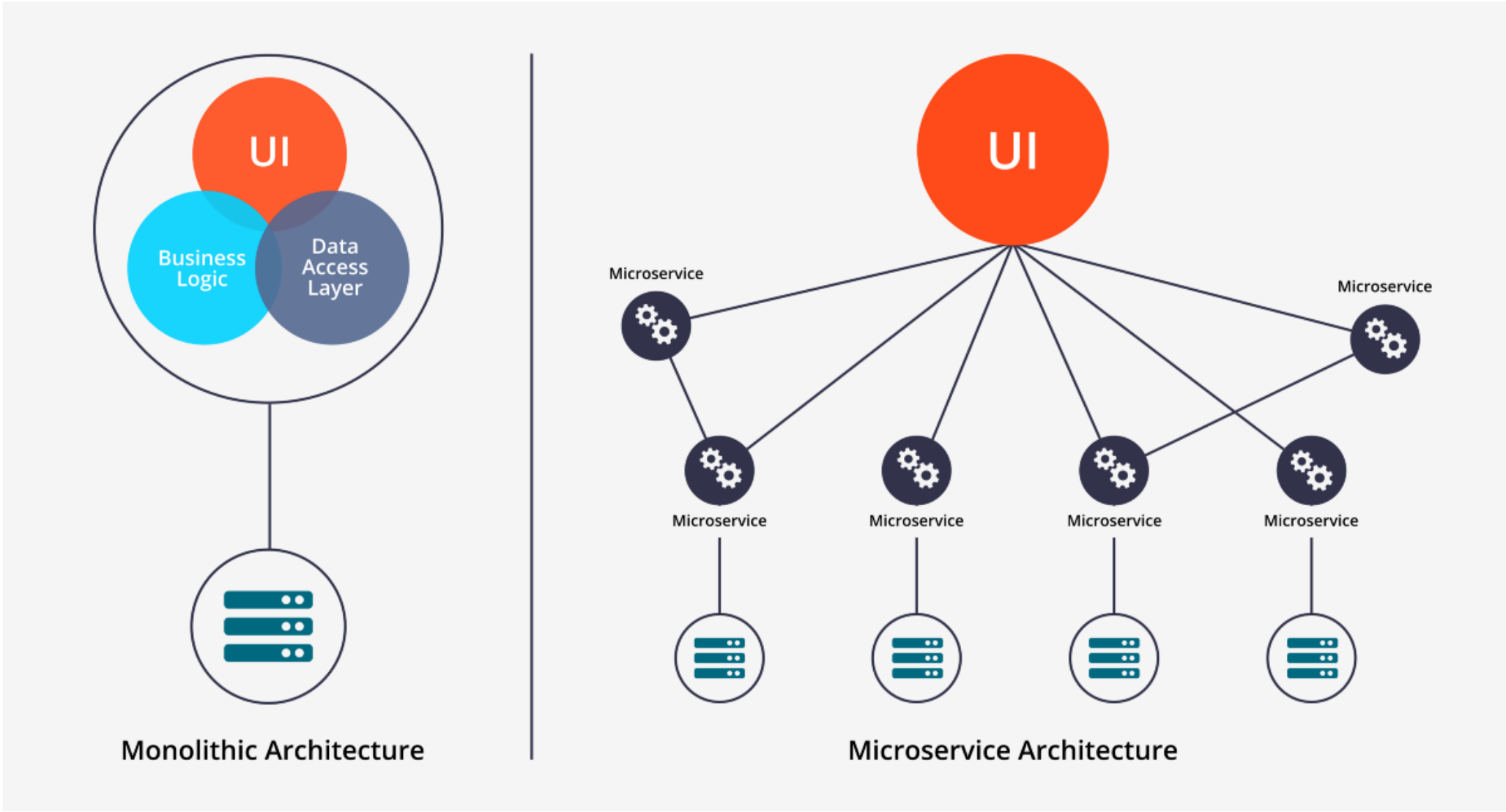
MICROSERVICES CORE CONCEPTS

- 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
  - Communicate with each other synchronously or asynchronously
  - Make application components easier to develop and maintain
  - Are a lot harder and more complicated to manage when deployed

MONOLITH TO MICROSERVICE



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

BUSINESS DRIVERS

- Business Agility
  - Speed of change is faster with a more modular architecture
  - React quicker to feature and enhancement requirements
  - Easy integration with new technology, processes (Mobile, Cloud, Continuous DevOps)
- Composability
  - Allows for reuse of capability and functionality
  - Allows for integration with other internal and external services
  - Reduces technical debt and replication
- Robustness and Migration
  - A single microservice failure does not bring down an application
  - The business functionality is always available
  - Updated functionality can be rolled out in a controlled and safe manner
  - Smaller components reduces risk exposure

BUSINESS DRIVERS

- Scalability
  - Services can scale up to handle peak loads, or down to save costs
- Enable Polyglot Development
  - Different technologies can be used for different services
  - No lock-in to a single technology across the whole business

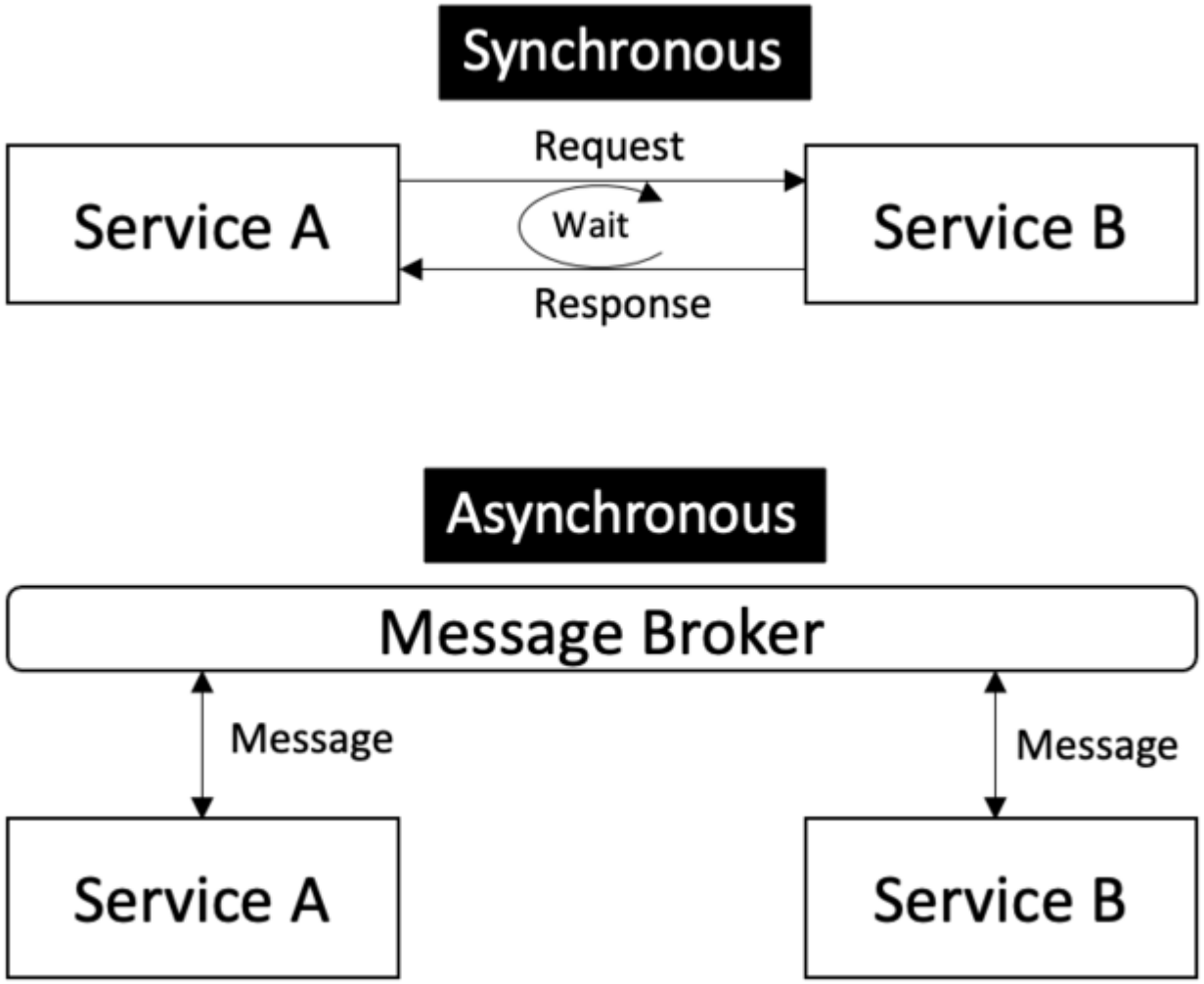
PROS AND CONS OF MICROSERVICES

- Pros
  - Easy to develop individually
  - Easy to understand individually
  - Easy to deploy individually
  - Easy to monitor each service
  - Flexible Release Schedule
  - Use standard APIs (JSON, XML)
- Cons
  - Requires retooling
  - Requires more deployments
  - Requires translation (JSON, XML)
  - Requires more monitoring
  - Operations configuration can be very complex
  - System can be very complex

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
  - Not quite accurate

BASIC ARCHITECTURAL APPROACHES



REQUEST BASED MICROSERVICE

- Generally part of a logic flow
- Implementation of 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.

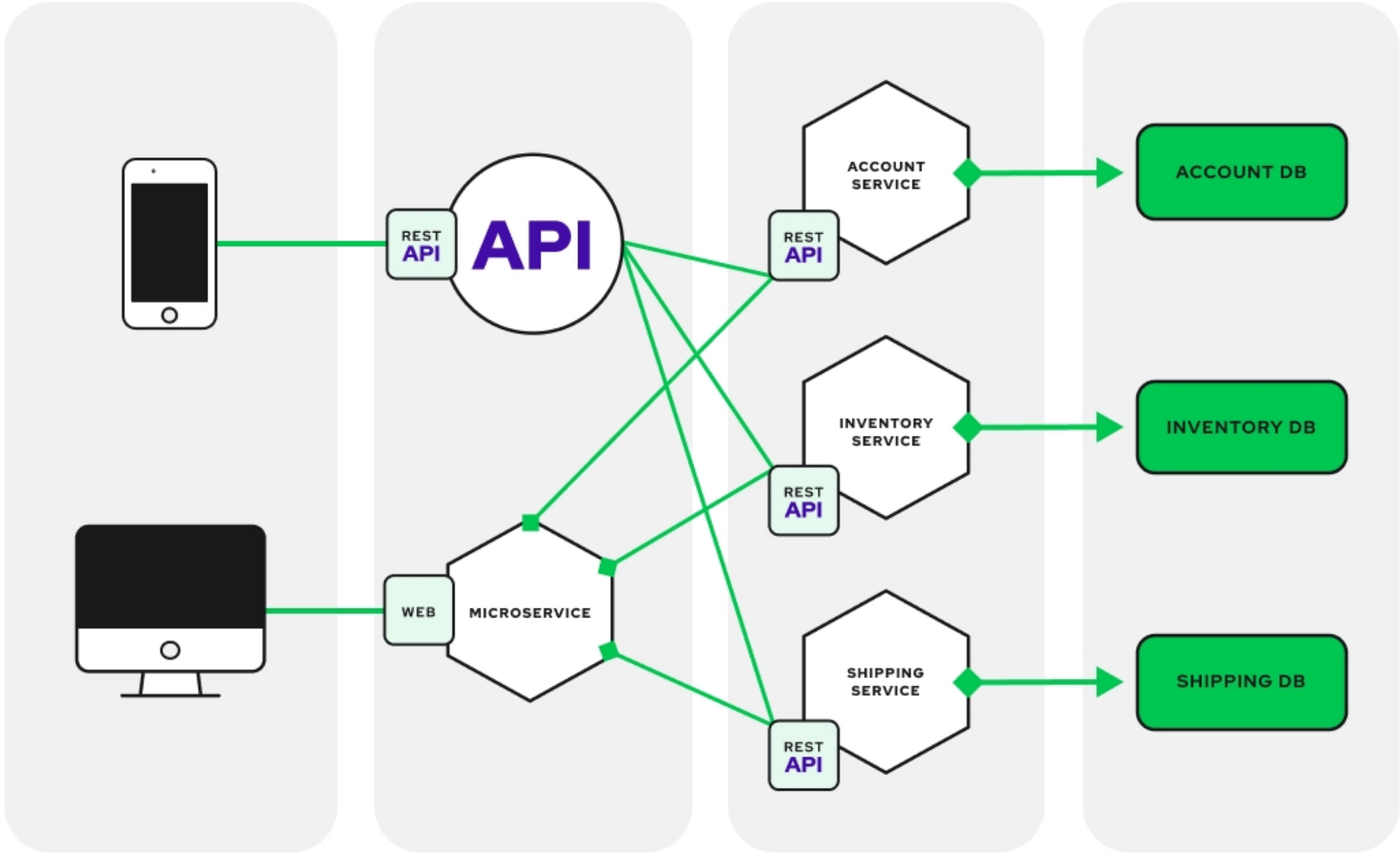
REQUEST BASED MICROSERVICE

- Often referred to as a synchronous service
- Each request requires a reply
  - The service may “block” waiting for a reply
  - This is a common use of the term synchronous
- However, request based microservices may be asynchronous
  - Often referred to as “fire and forget”
  - The sending application process can continue while waiting for a reply
- In the sales example
  - The request for shipment can be made
  - But the sale interaction ends without getting a confirmation of the shipment details
  - The confirmation can be sent 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
    - “Your 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

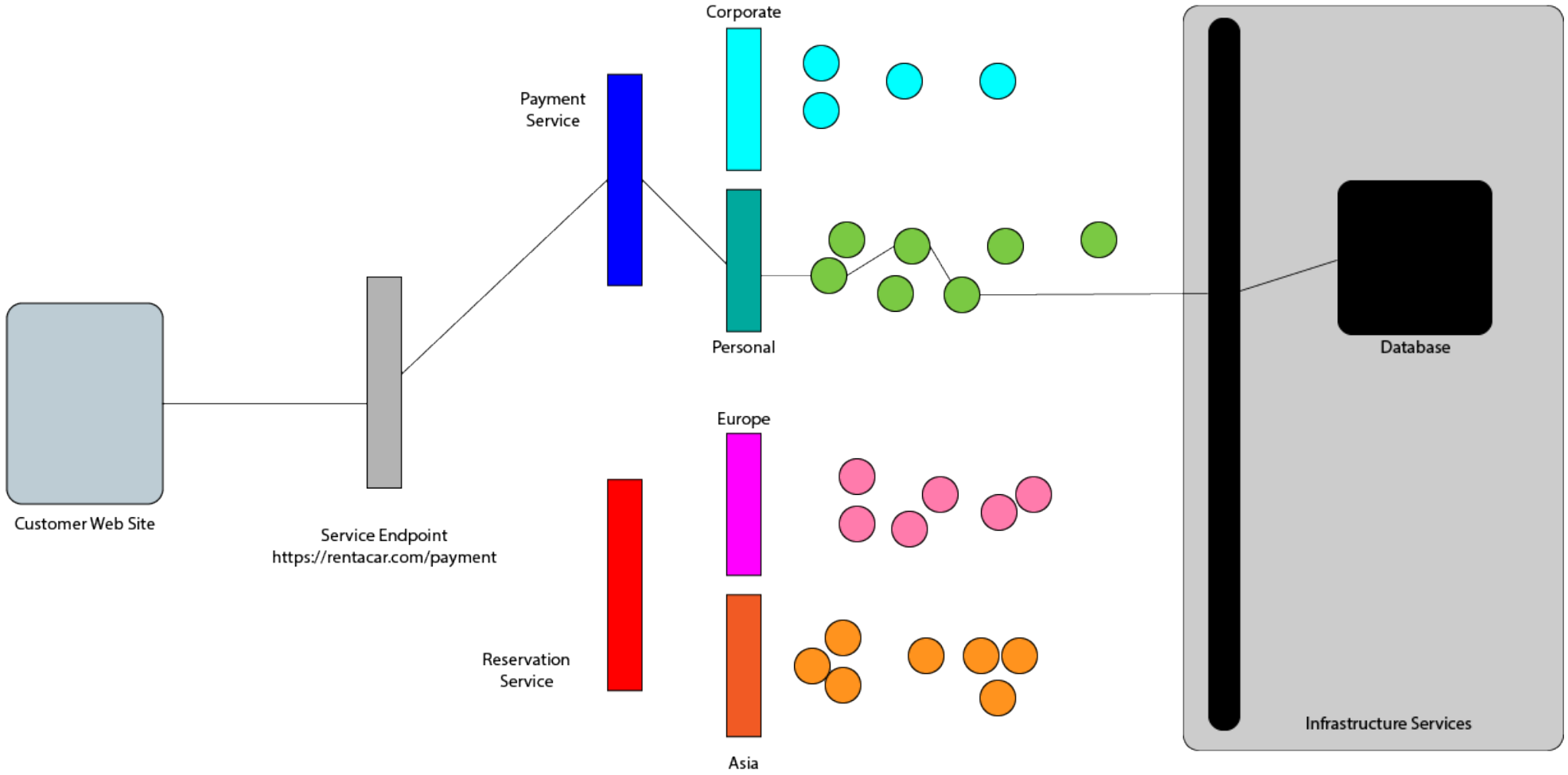
REQUEST BASED



PROCESSING A REQUEST

- Request arrives at an endpoint
  - Depending on the service requested
    - The message is routed to an internal service that handles that type of request
    - That service routes the message to a running process to handle the process
    - The process retrieves any state data it needs from the data service
    - Request is processed
    - State data store is updated
    - Response returned to service that returns to the endpoint that received the request
  - More than one process type may be involved
    - Processes are stateless and horizontally scale-able
    - The main challenge: how do you orchestrate the flow of messages?

PROCESSING A REQUEST

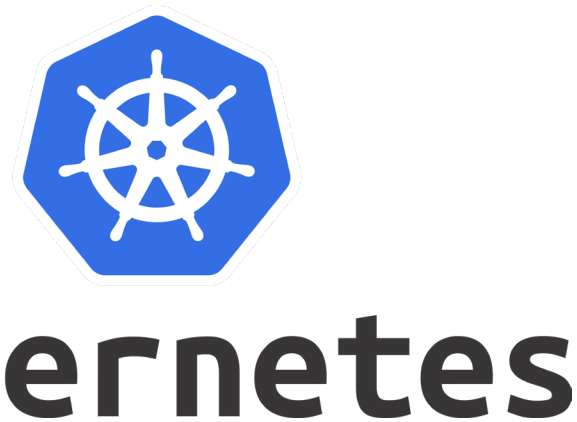


THE OPERATIONS CHALLENGE

- Processes are usually deployed in Docker or similar containers
  - But we have to solve:
    - Coordinating the activity of possibly thousands of containers that need to work together
    - Creating and maintaining connections between containers
    - Ensure the whole system operates well enough to meet Service Level Agreements (SLAs)
  - We need to deal with non-functional requirements
    - Loading, throughput, stress, response times
    - Disaster recovery
    - Security
  - The lack of an effective way to do this was a major historical impediment to the deployment of microservice based applications

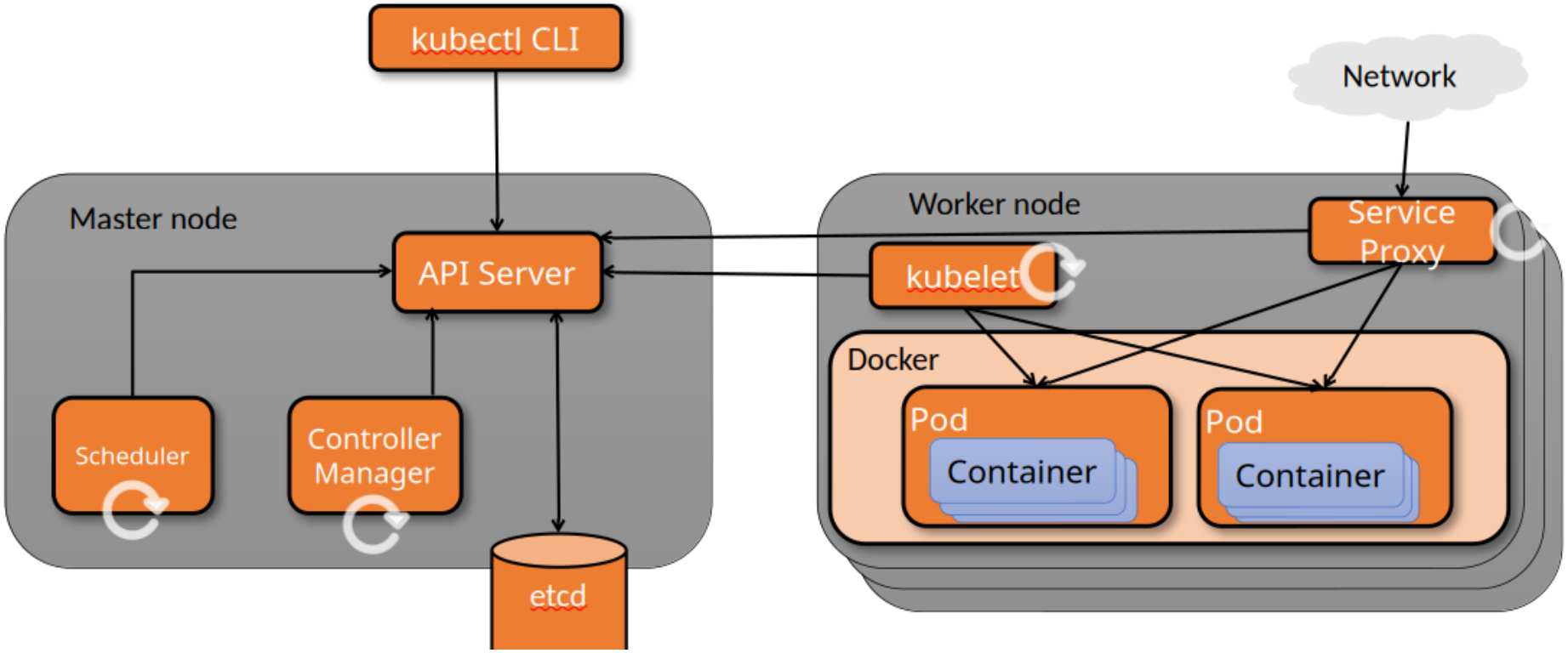
KUBERNETES

- Kubernetes is a container orchestration manager
  - Not the only manager
  - Docker Swarm does the same
  - Kubernetes is “industrial strength”
- Orchestration:
  - Manages “clusters” of containers
  - Provides service discovery
  - Manages scaling and failover
  - Works at the Ops level
  - Infrastructure as Code

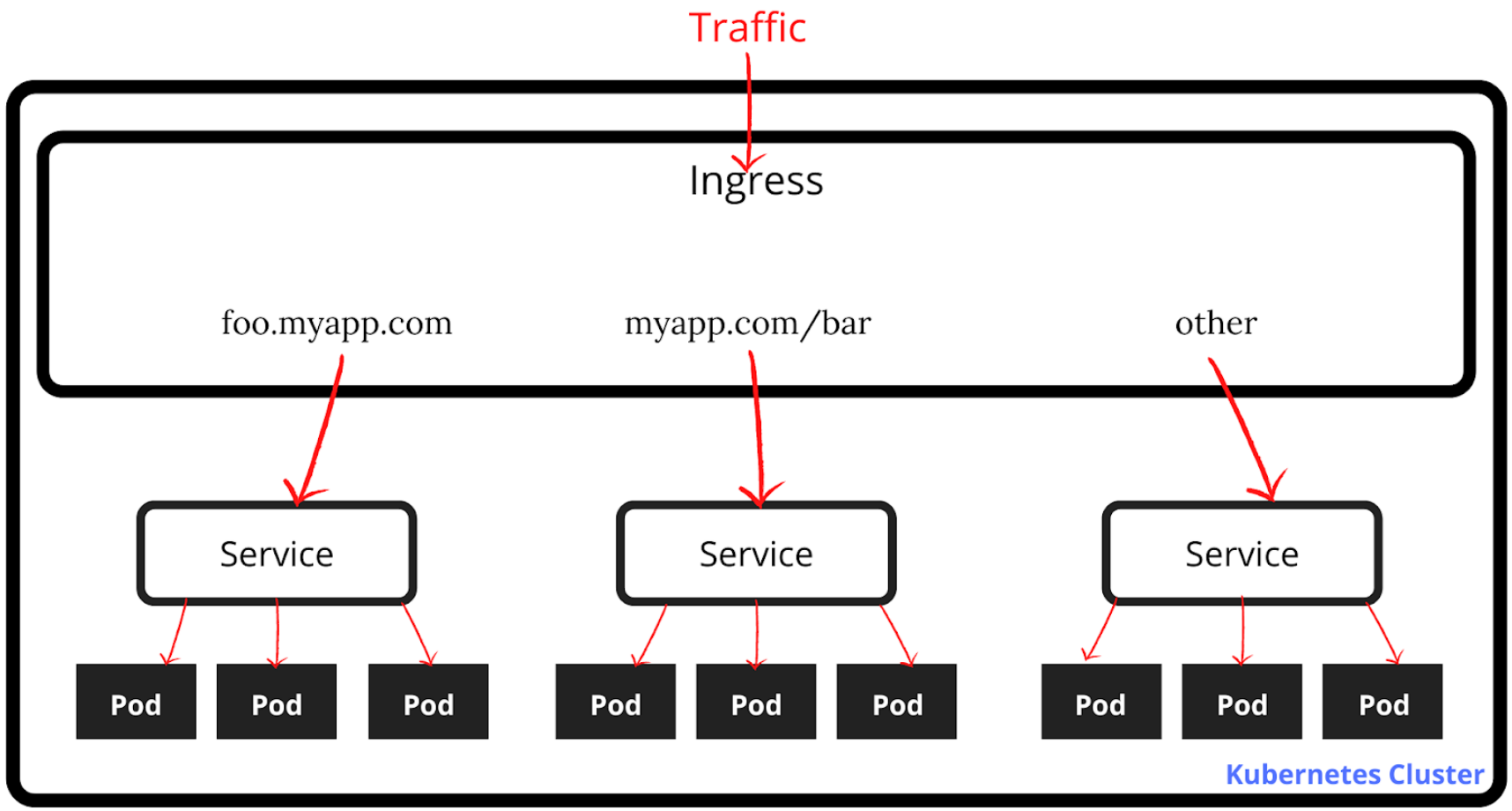


KUBERNETES ARCHITECTURE

- Kubernetes nodes can be physical hosts or VM's running a container-friendly Linux



KUBERNETES SERVICES



COMMON DEPLOYMENT

- Modern microservices that are request based tend to use a common set of tools
  - *Containers*
    - Small, self contained units of functionality
    - Designed to follow all the good design principles: cohesive, supple, etc
    - Dynamically horizontally scalable
  - *Pods*
    - Wrapper around a container or set of containers to create a usable microservice
    - Adds the necessary interface and infrastructure so the containers can be managed by the orchestration tool
  - *Orchestration*
    - A control system that manages the pods and running environment
    - Routes incoming requests to the appropriate microservice

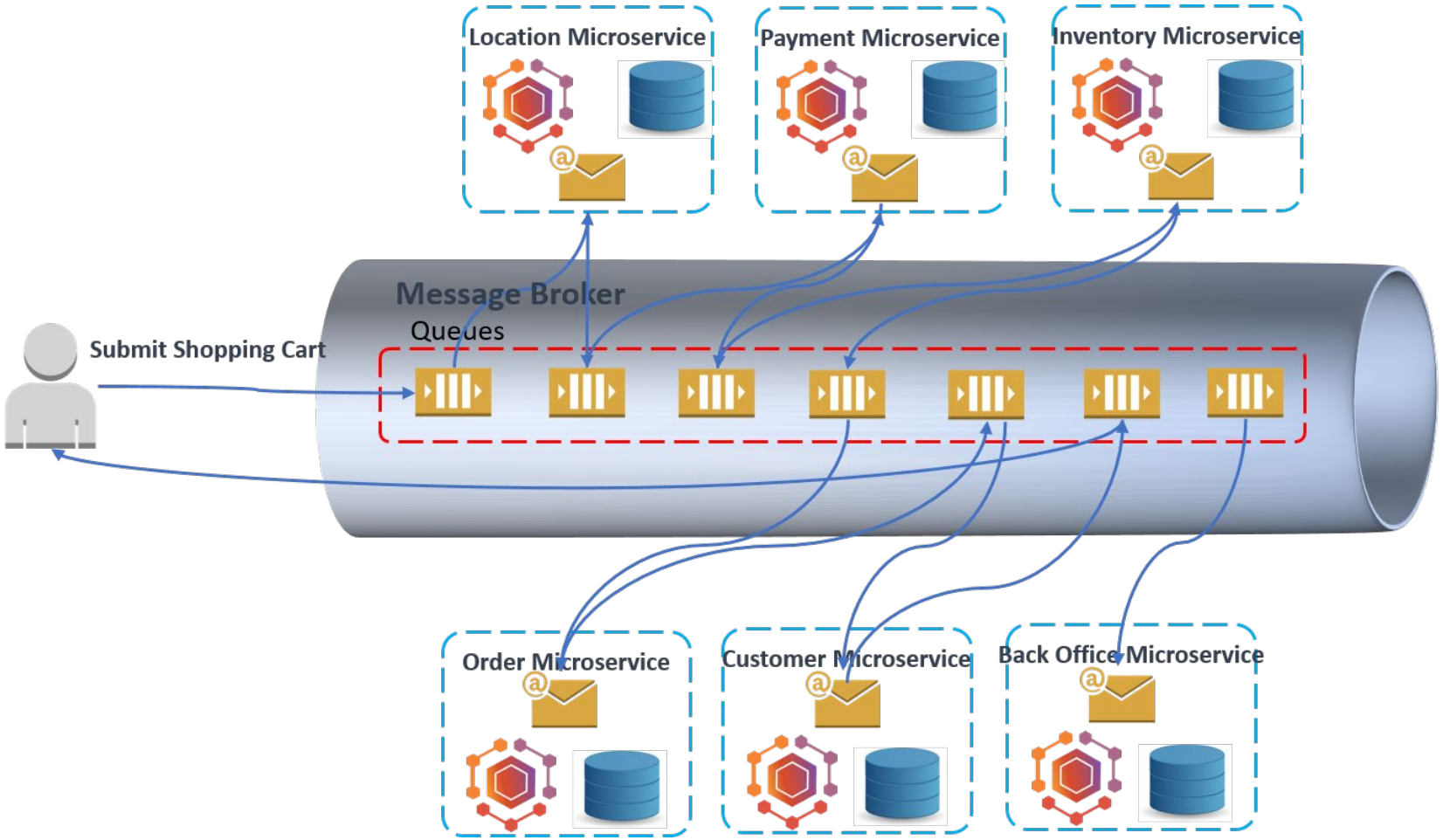
EVENT BASED MICROSERVICE

- Responds to a stream of events
  - Asynchronous in design
  - Events may require a response by the system
    - But response may or may not involve a reply to the event source
    - The response is often an event sent to another service
    - Architectural implementation of the Command and Chain of Responsibility design patterns
  - Example: Heat sensor
    - A heat sensor in a power plant takes a heat reading
    - An event is sent to the monitoring microservice
    - Monitoring service creates new events in response
    - Sends an event to the data recording service to log the data
    - If the reading is anomalous, it sends an event to an alert services

EVENT BASED MICROSERVICE

- Events may require a response by the system
  - But response may or may not involve a reply to the event source
  - The response is often an event sent to another services
- Typical use cases
  - Event Response
    - Events might occur that require a response that is not a reply to the event source
    - For example: monitoring heat sensors in a nuclear reactor
    - The response is 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
    - For example: collecting GPS location of cell phones
    - Collecting point of sales data for data analytics
    - No reply or response required

EVENT BASED MICROSERVICE

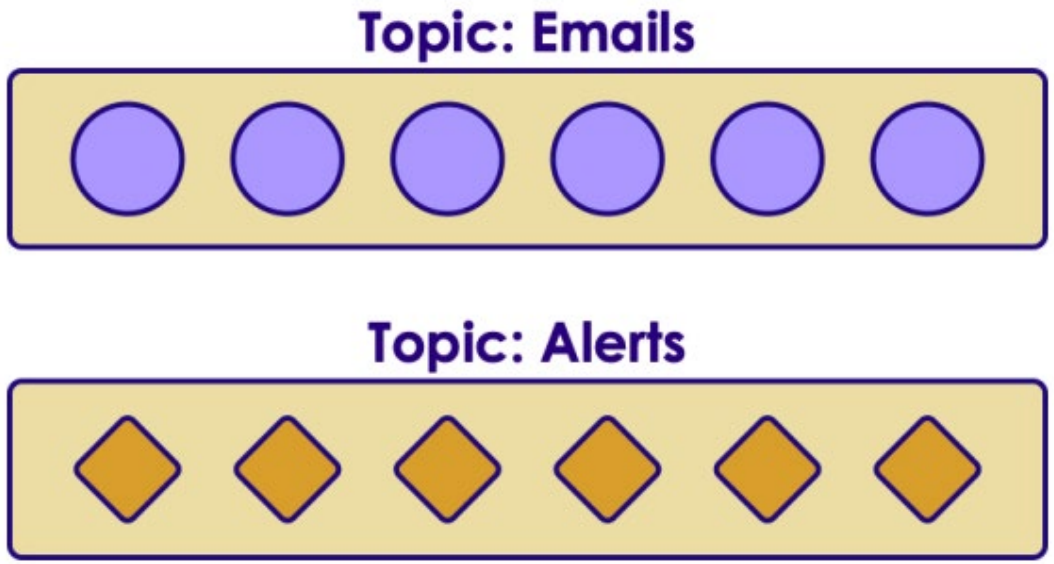


EVENT BASED MICROSERVICE

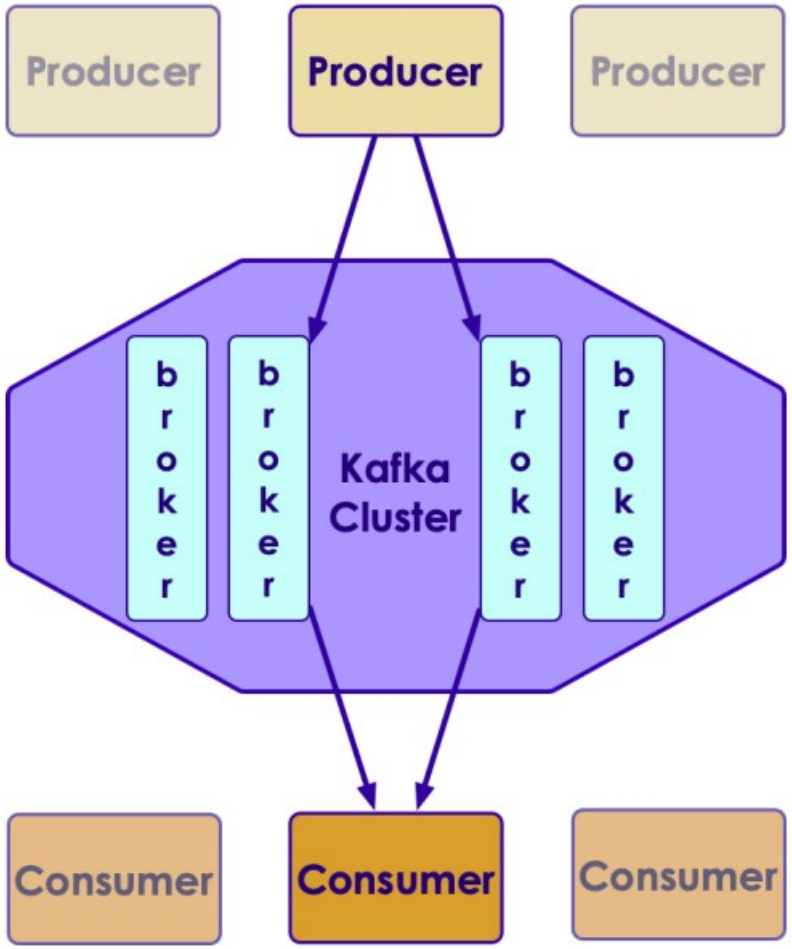
- Instead of messages, we think in terms of events
  - An event is some data item of interest in the domain
- Instead of a cluster like Kubernetes, the main artifact is a stream or queue
  - Publishers put events onto the queue
  - Subscribers get events off of the queue
  - Generally referred to as the “pub-sub” model
- The most common technology used is Kafka
  - Acts as a asynchronous buffer between publishers and subscribers

KAFKA CONCEPTS

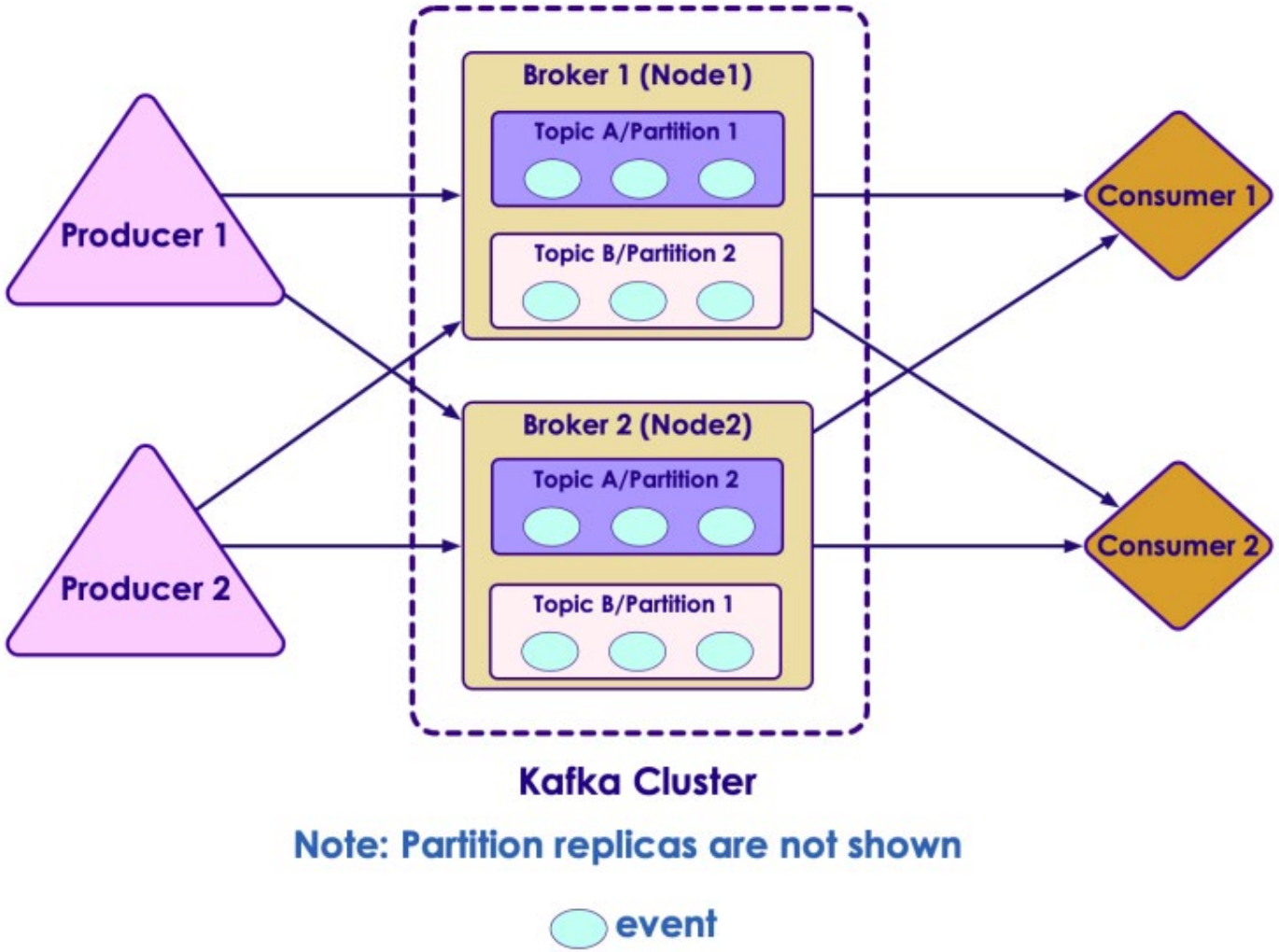
- In Kafka a basic unit of data is a 'message'
  - Message can be email / connection request / alert event
- Messages are stored in 'topics'
  - Topics are like 'queues'
  - Sample topics could be: emails / alerts



KAFKA ARCHITECTURE



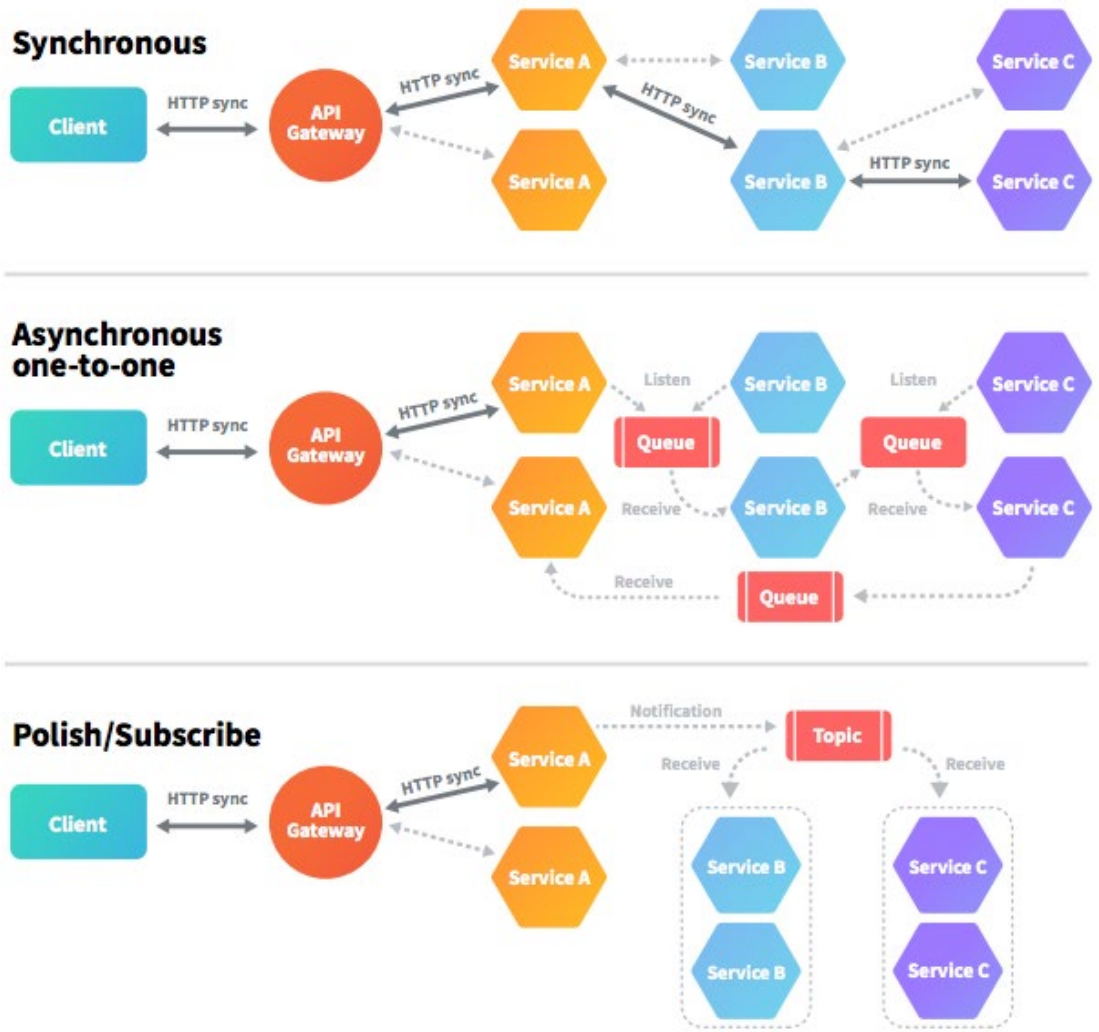
KAFKA ARCHITECTURE



HYBRID MICROSERVICE

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

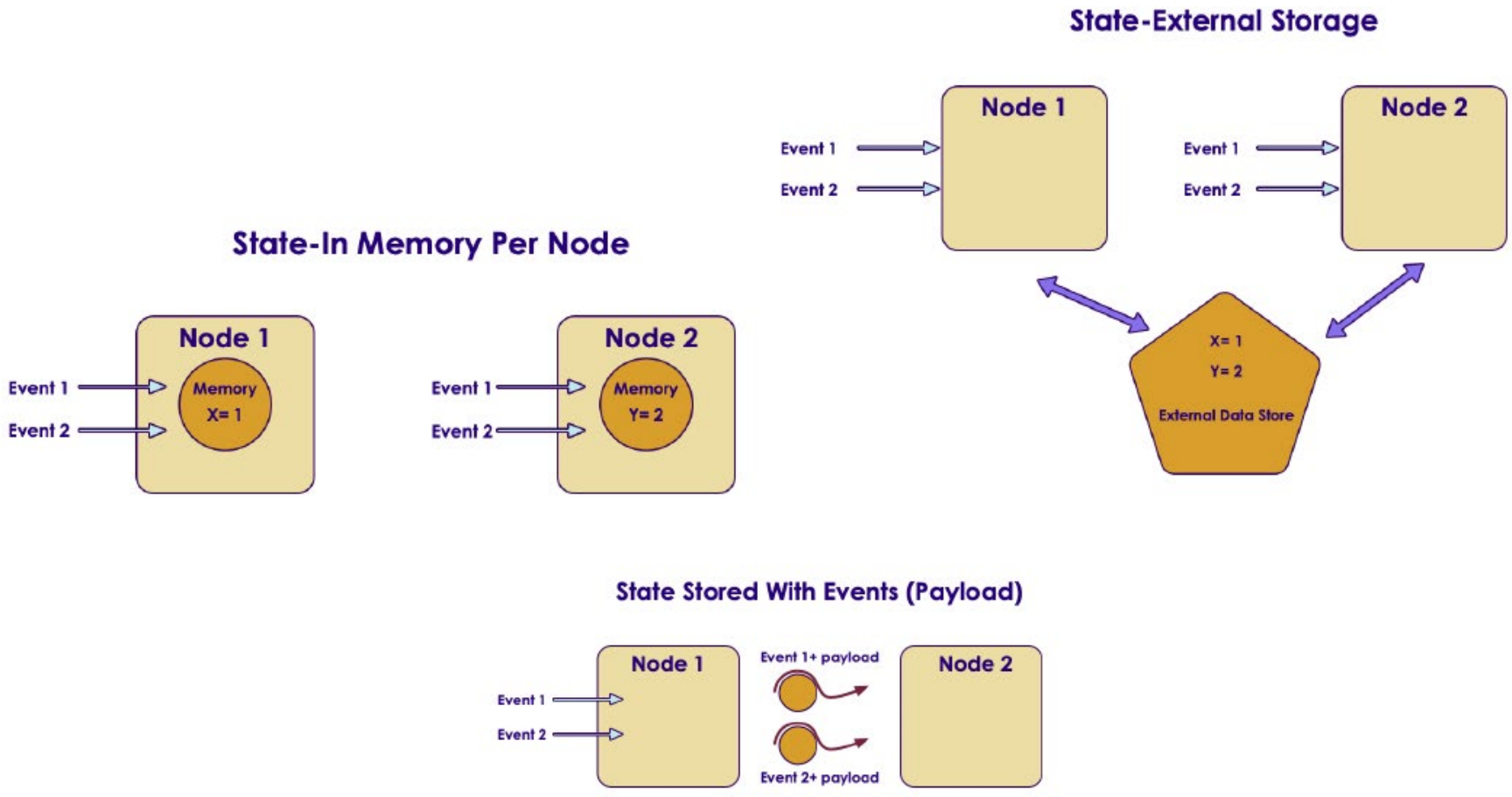
KAFKA ARCHITECTURE



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 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



OPERATIONS MONITORING

- Monitoring is both proactive (detect early) and reactive (respond fast)
  - It spans multiple layers of the IT architecture

Layer	Examples of What's Monitored	Tools / Technologies
Infrastructure	Servers, mainframes, networks, storage, power, HVAC	Nagios, Zabbix, SolarWinds, AWS CloudWatch, Azure Monitor
Application	Response times, transaction success rates, API errors	AppDynamics, New Relic, Dynatrace, Datadog
Database / Data Services	Query latency, replication lag, corruption, locks	Oracle Enterprise Manager, SQL Profiler, Prometheus exporters
Middleware / Microservices	Container health, message queues, dependencies	Grafana, Prometheus, ELK (Elasticsearch–Logstash–Kibana), OpenTelemetry
User Experience	Front-end page loads, mobile response, synthetic transactions	Catchpoint, Pingdom, BrowserStack
Security Events	Logins, access attempts, anomalies	SIEM tools (Splunk, IBM QRadar, Azure Sentinel)

OPERATIONS MONITORING

- Modern production support environments rely on
  - *NOC (Network Operations Center):* focuses on network health and uptime
  - *SOC (Security Operations Center):* monitors and responds to security threats
  - *SRE (Site Reliability Engineering):* automates reliability through IaC and observability
  - *Ops Command Centers:* integrated cross-domain response for high-impact incidents
- Monitoring is effective when it is
  - *Integrated*: unified dashboards across systems
  - *Contextual*: aligned with business processes and risk criticality
  - *Actionable*: alerts correlate to meaningful incidents, not noise

OPERATIONS MONITORING TOOLS

- Legacy operations (mainframes and monoliths)
  - Typically centralized systems with batch processes and predictable workloads
  - Monitoring focuses on job completion, I/O throughput, and storage utilization
- Distributed and web-based operations
  - Multi-tier architectures (web, app, DB)
  - Use of middleware, load balancers, and APIs
  - Emphasis on end-to-end monitoring and dependency mapping

OPERATIONS MONITORING TOOLS

- Cloud and hybrid monitoring
  - Cloud-native services (AWS, Azure, GCP) provide integrated metrics, logging, and alerting
  - Hybrid architectures require federation of monitoring tools from the different operations environments
  - Use of AIOps (AI for IT Operations) to detect anomalies and predict outages
    - Machine learning is used to do effective anomaly detection and deviations from baselines
    - Does not replace human oversight and decision making
    - Helps manage the massive amounts of data collection
    - Helps filter out events that might be missed otherwise

OPERATIONS MONITORING TOOLS

- DevOps and CICD
  - Monitoring embedded in the pipeline
  - Automated tests, deployment health checks, rollback automation
  - “Shift-left monitoring” goal is catching operational risks earlier in the lifecycle

COMMON CHALLENGES IN OPERATIONS

Challenge	Description	Resilience / Risk Implication
Alert Fatigue	Too many alerts, too little prioritization; operators miss critical issues.	Critical incidents go unnoticed; resilience compromised.
Tool Fragmentation	Multiple, unintegrated tools with inconsistent data.	Slow incident correlation; delays in root-cause analysis.
Shadow IT & Cloud Sprawl	Unmanaged assets outside central monitoring.	Blind spots increase risk exposure.
Insufficient Automation	Manual remediation leads to longer MTTR (Mean Time To Repair).	Prolonged downtime; reduced reliability.
Data Overload / Lack of Analytics	High data volume but low actionable insight.	Poor situational awareness in crisis.
Skill Gaps in Ops Teams	Limited cross-domain understanding (e.g., cloud + security).	Poor incident response coordination; higher operational risk.
Change Risk	Rapid deployments or config errors introduce instability.	Increased incident frequency; resilience erosion.

RISK AND RESILIENCE

Resilience Function	Operational Monitoring Link
Preventative Controls	Early detection of anomalies; health checks; automated scaling to prevent failure.
Detective Controls	Continuous monitoring, alerting, SIEM correlation, observability pipelines.
Corrective Controls	Incident response, rollback, failover, service restoration.
Adaptive Controls	Learning from incidents; tuning thresholds; predictive analytics.

BEST PRACTICES

- Define critical services and dependencies
  - Identify systems supporting mission-critical processes (e.g., payments, authentication)
  - Prioritize monitoring coverage accordingly
- Integrate security and operations monitoring
  - Correlate performance metrics with security events for unified situational awareness
- Implement observability architecture
  - Combine metrics, logs, and traces into a unified big picture

BEST PRACTICES

- Use predictive analytics and AIOps
  - Detect anomalies before they become incidents
  - Automate triage and escalation
- Embed resilience metrics
  - Track MTBF, MTTR, SLA breaches, and incident trends
  - Link them to risk appetite and tolerance thresholds
- Conduct regular drills and war-games
  - Use real monitoring data to simulate outage and recovery scenarios
  - Assess monitoring effectiveness and operator readiness

PROCESS MATURITY

Maturity Level	Resilience Capability	Characteristics in Practice
Level 1 – Initial	**Fragile.** Recovery relies on individuals, not processes.	Crisis response is improvised; high downtime; unclear ownership.
Level 2 – Repeatable	Partial resilience; some known recovery steps.	Some procedures exist but not tested; reliance on tribal knowledge.
Level 3 – Defined	Baseline resilience; structured recovery.	Documented SOPs and runbooks guide restoration and escalation.
Level 4 – Managed	**Adaptive resilience.** Metrics drive readiness and improvement.	Recovery rehearsed; simulation data used for preparedness.
Level 5 – Optimized	**Resilience by design.** Processes self-monitor and self-correct.	Automation, orchestration, and AI-driven remediation minimize downtime.

Q&A AND OPEN DISCUSSION