



Cloud Design Patterns



- » Skan.ai - chief Architect
- » Ai.robotics - chief Architect
- » Genpact - solution Architect
- » Welldoc - chief Architect
- » Microsoft
- » Mercedes
- » Siemens
- » Honeywell



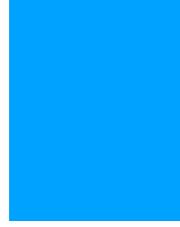
Mubarak



- Cloud Arch Styles
- Cloud Arch Patterns
- Case studies

System

Arch Styles (big pattern)

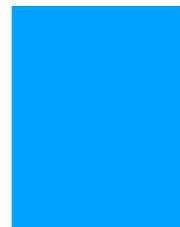


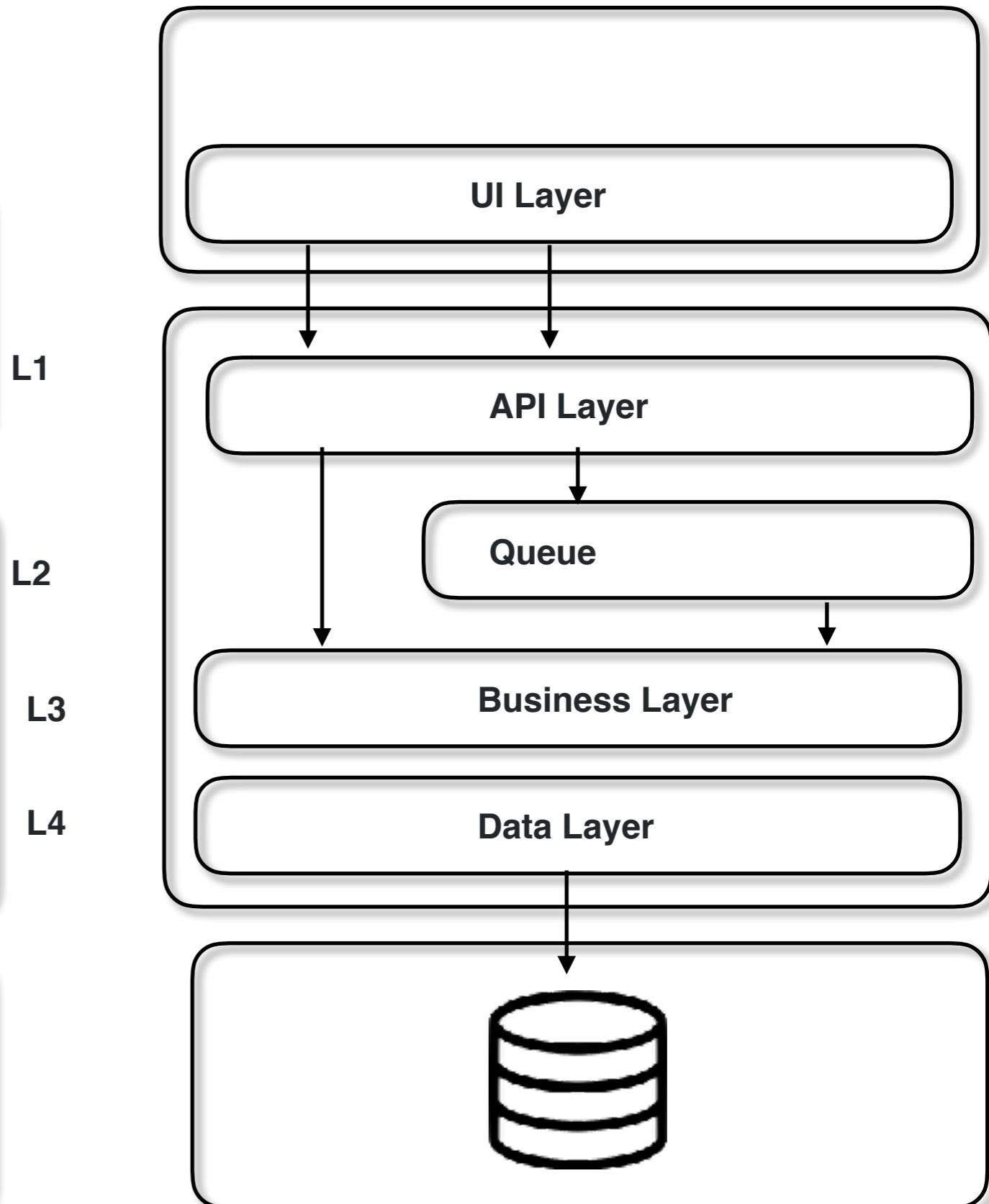
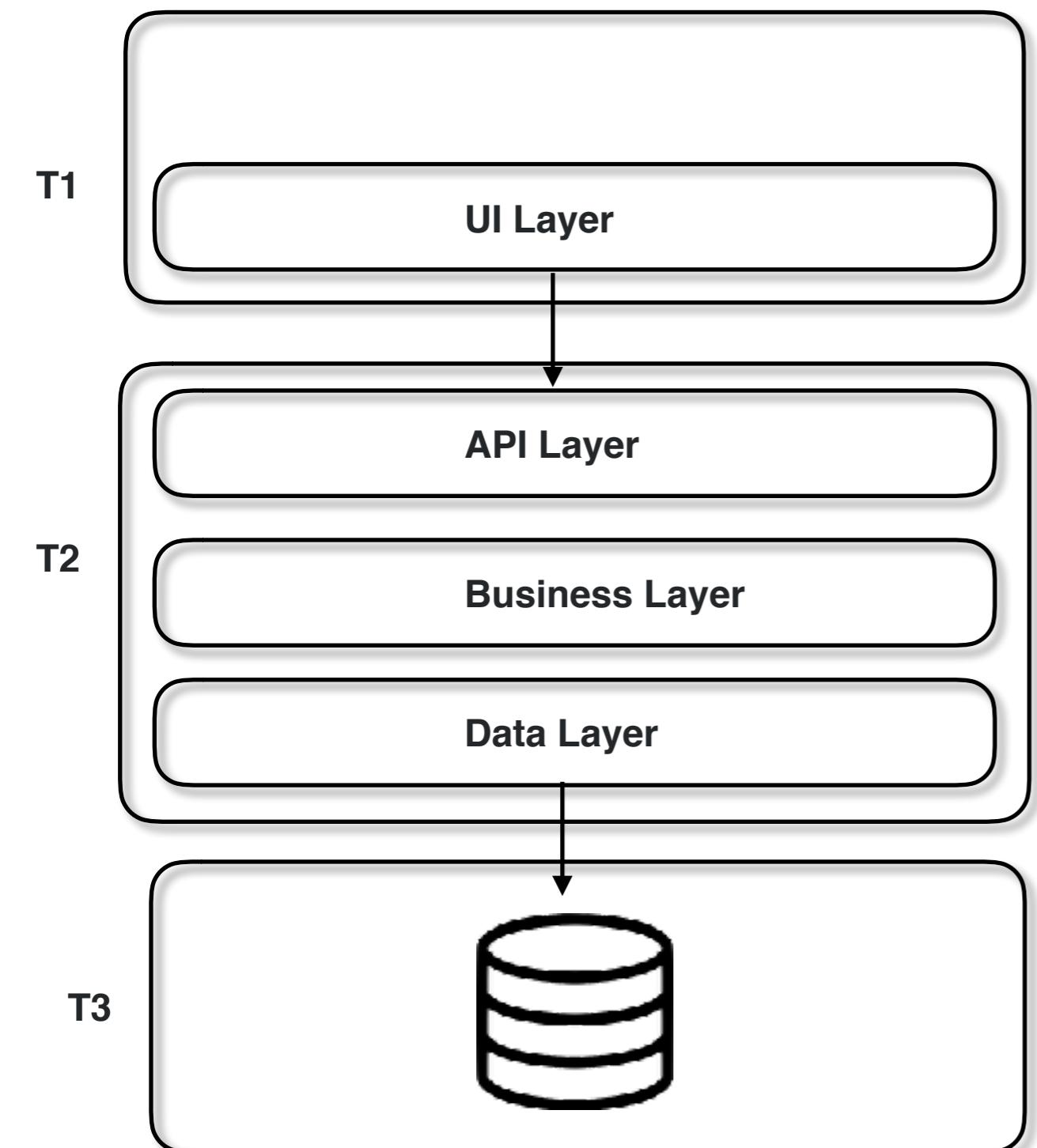
Arch tactics
(small pattern)

Arch patterns

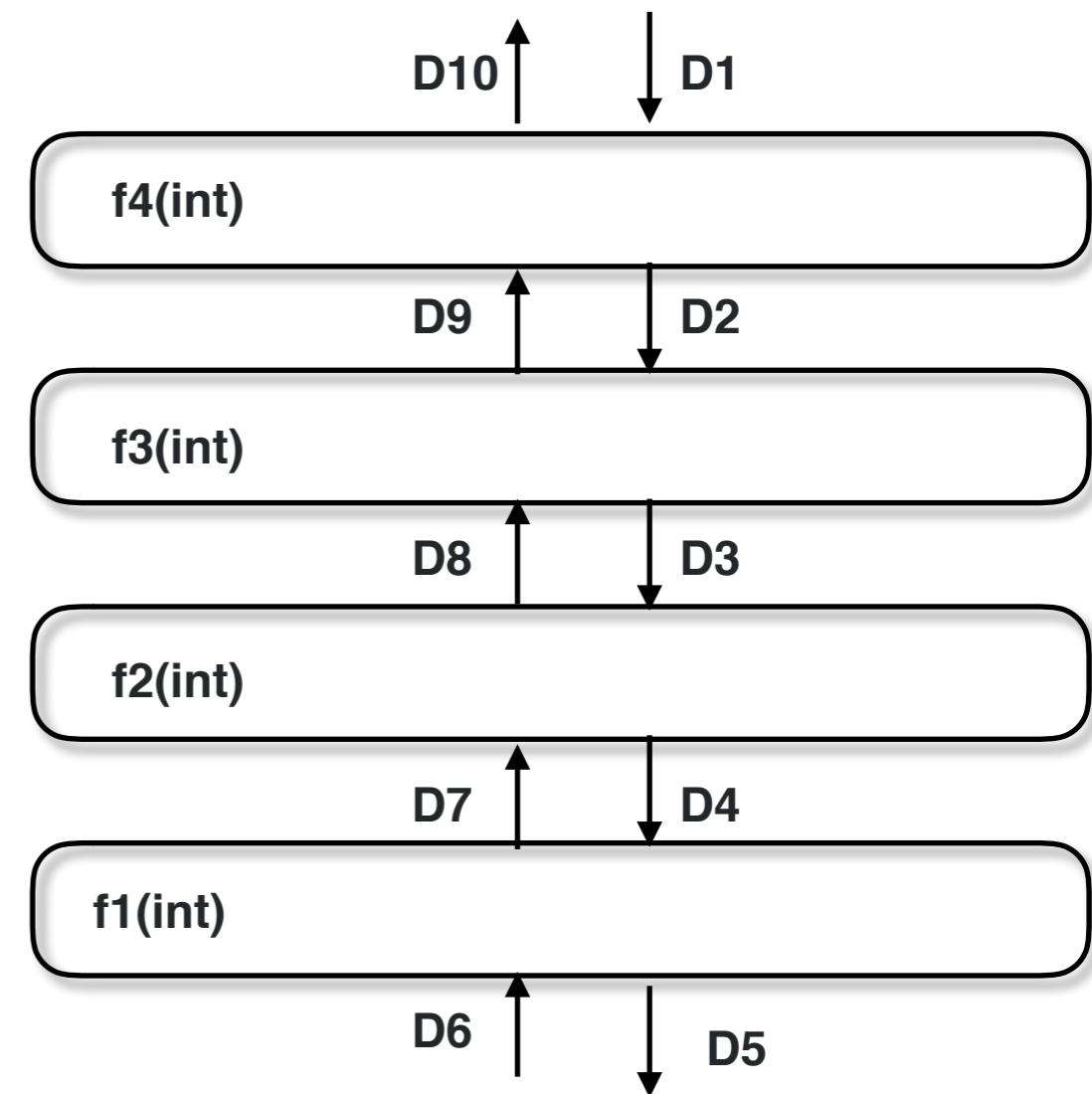
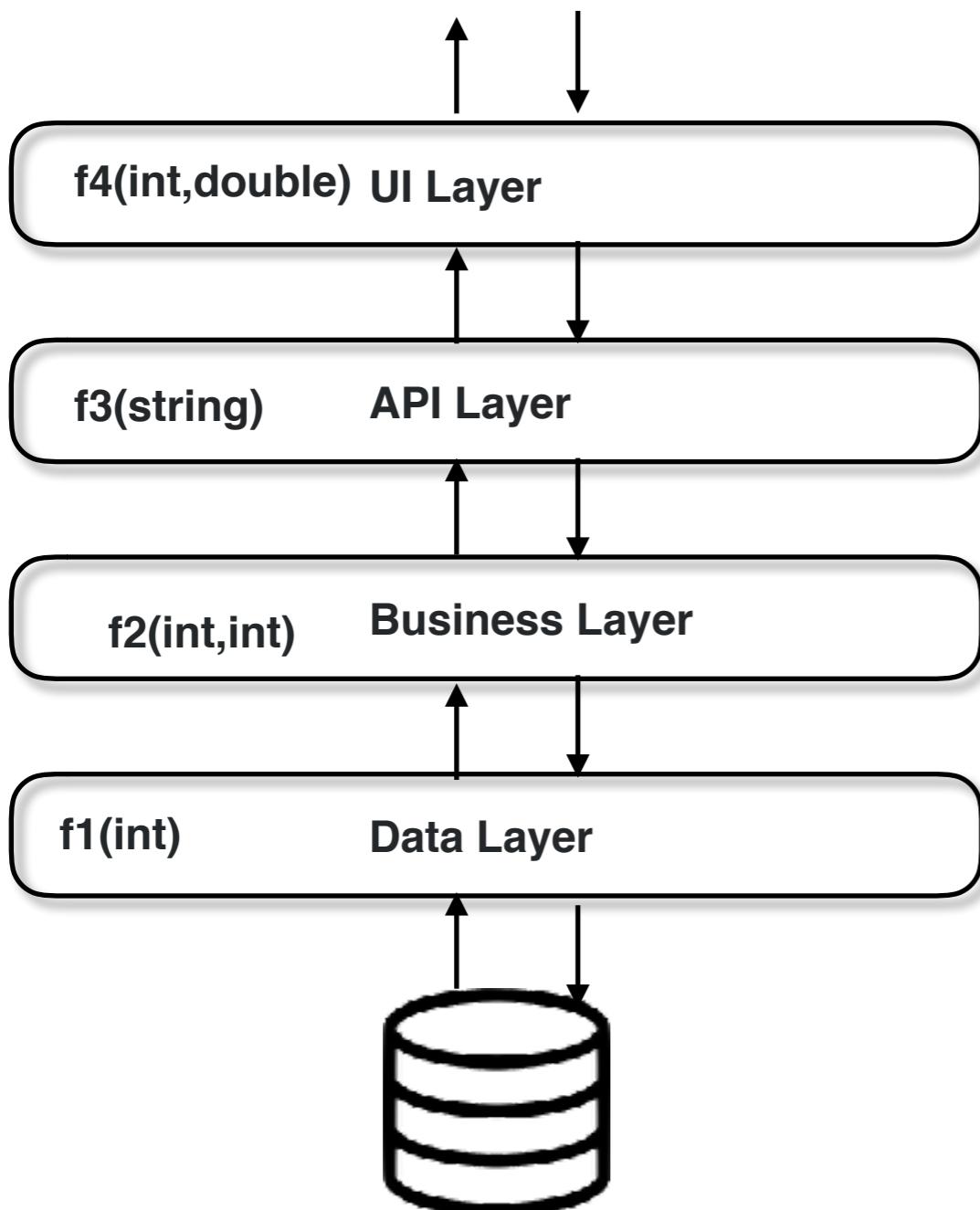


Arch patterns





**Web socket
Web Hooks
SSE**



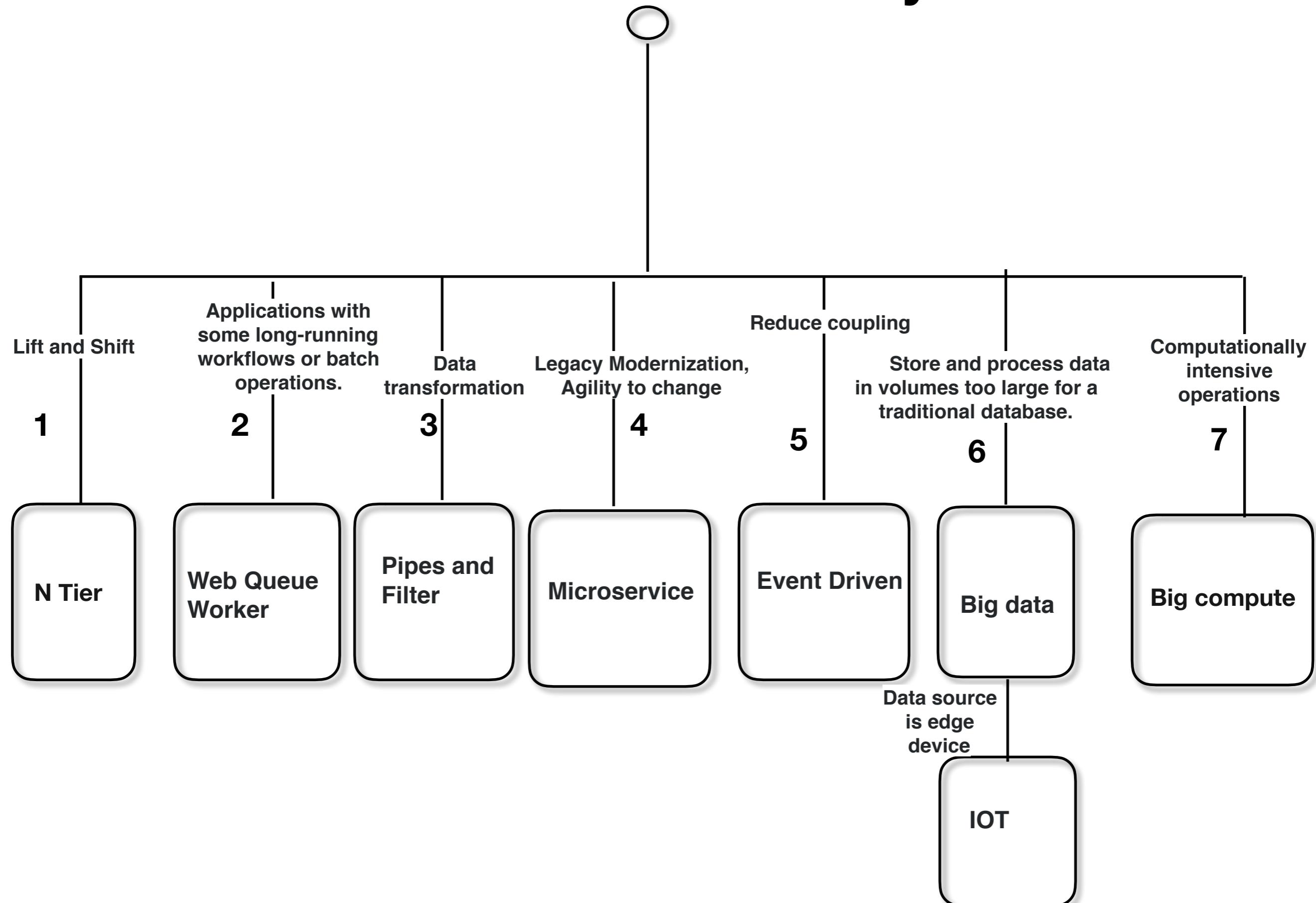
**Homogenous
(all filters expose same contract)**

SOC
(each layer expose different contract)

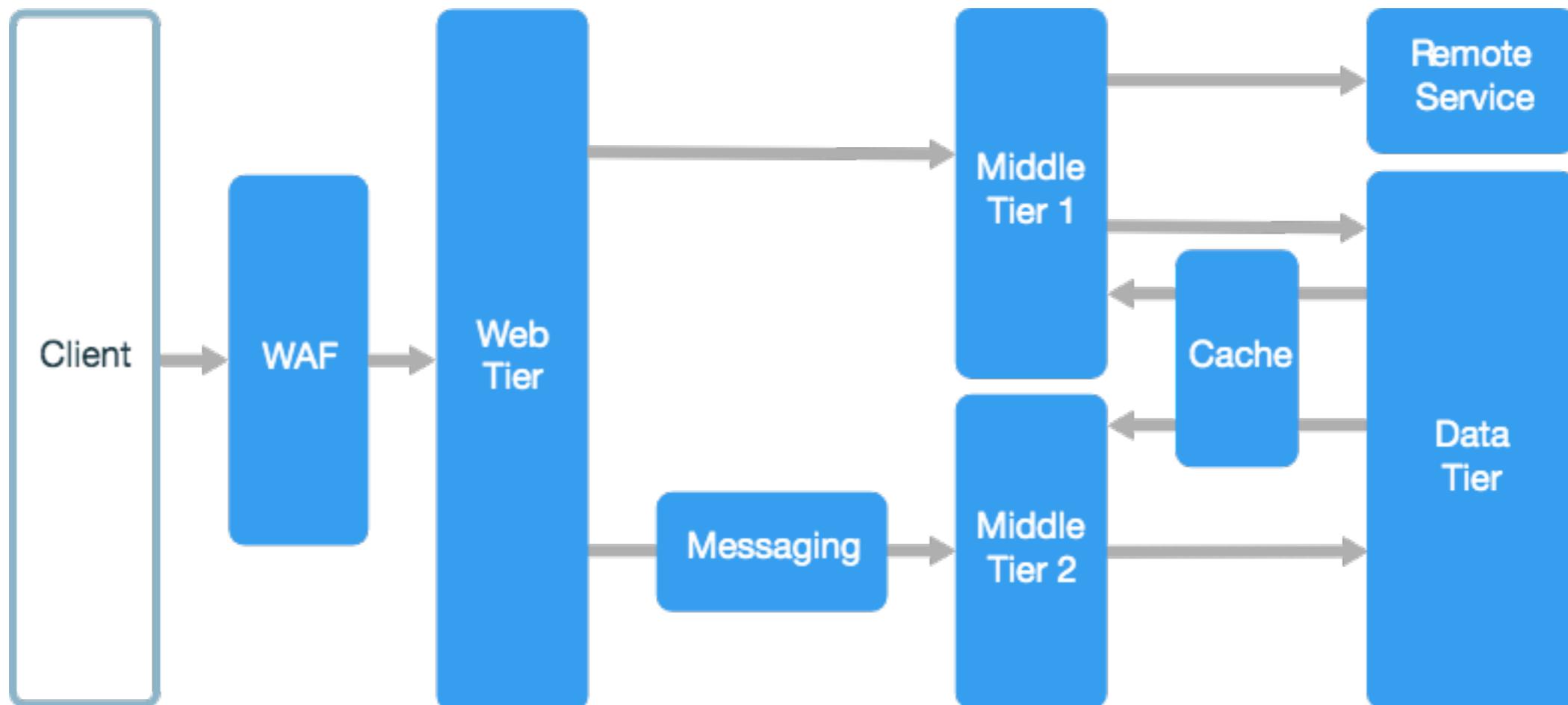


Cloud Architecture Styles

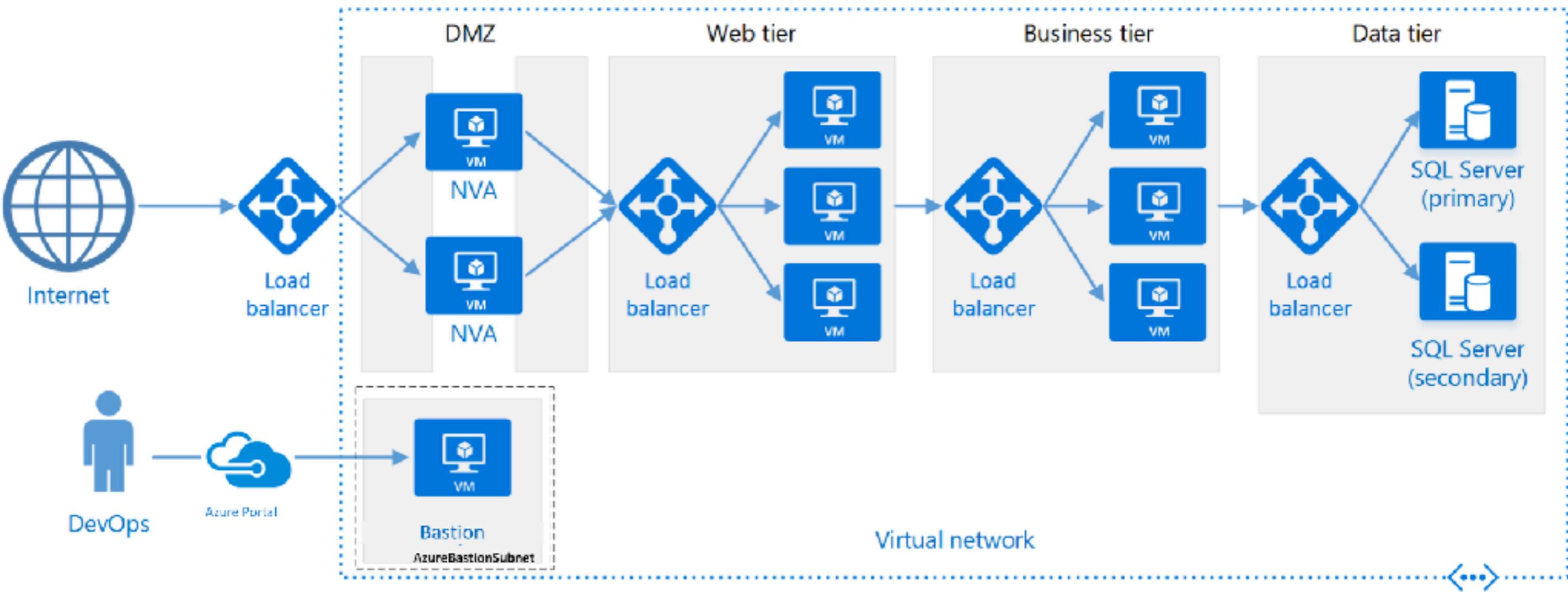
Cloud Architecture Styles



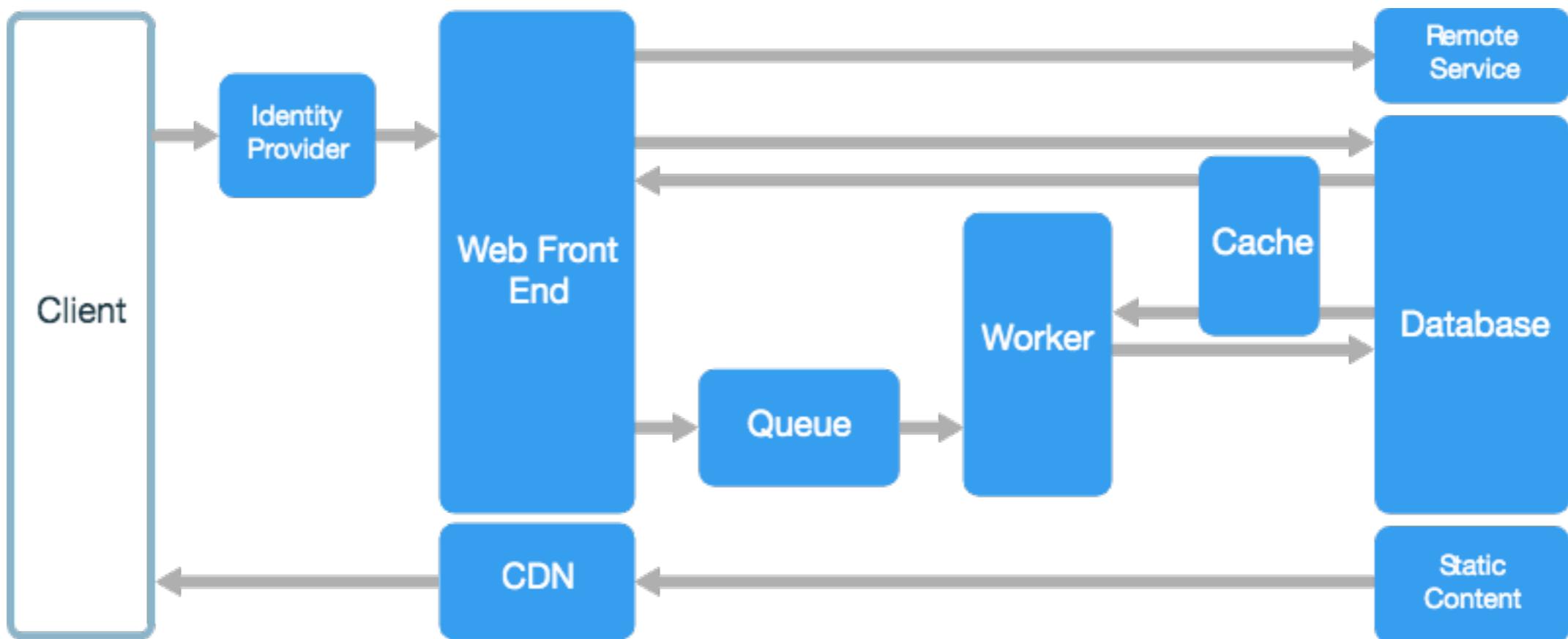
N Tier Reference Architecture



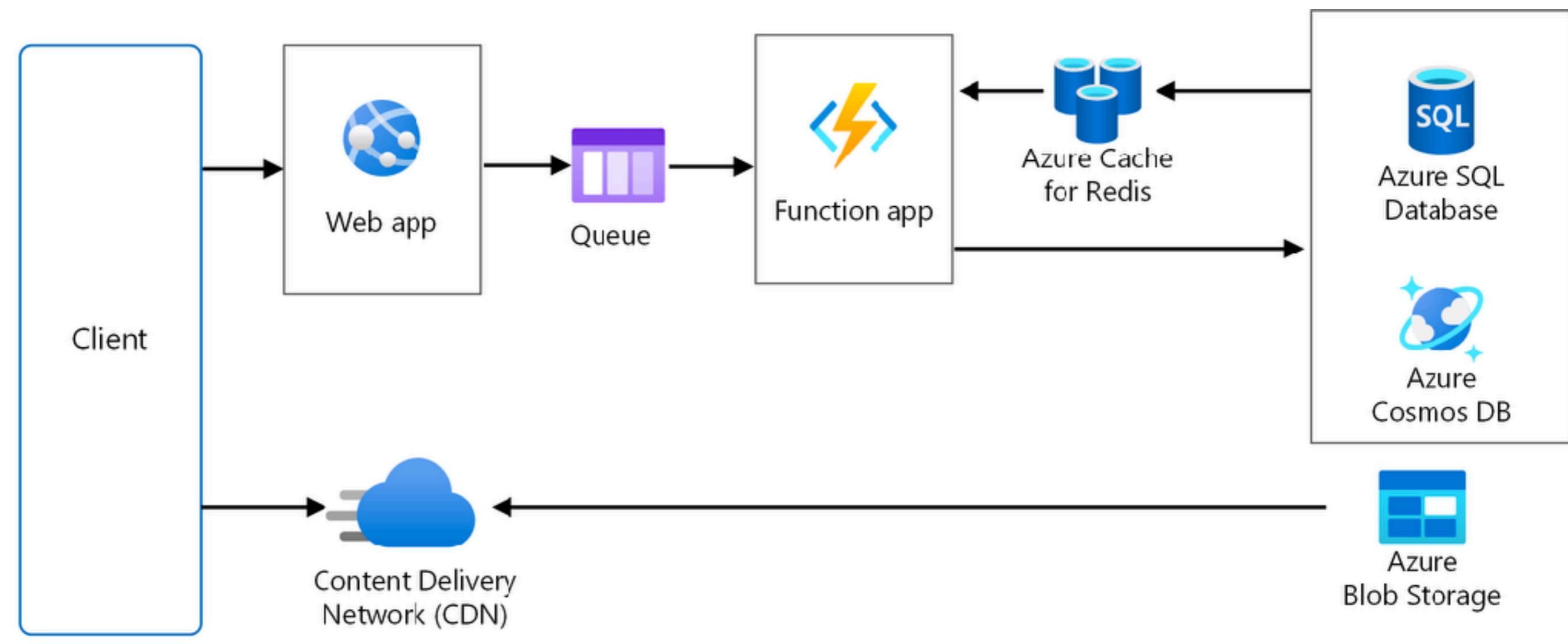
- Migrating an on-premises application to Azure with minimal refactoring.



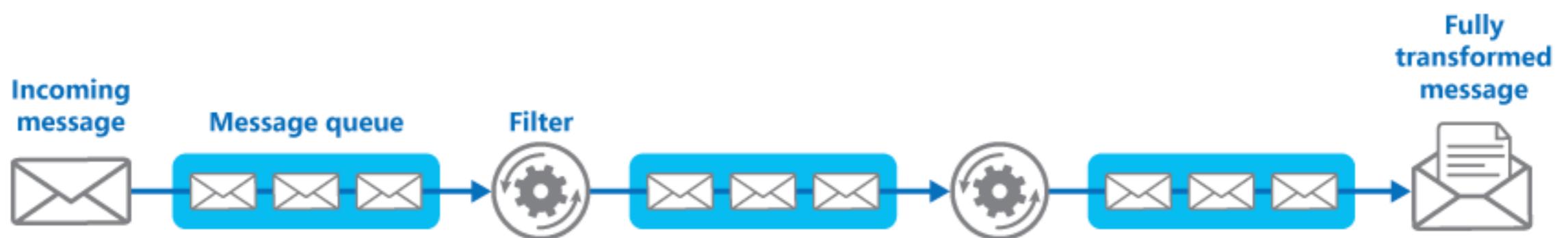
Web Q Worker

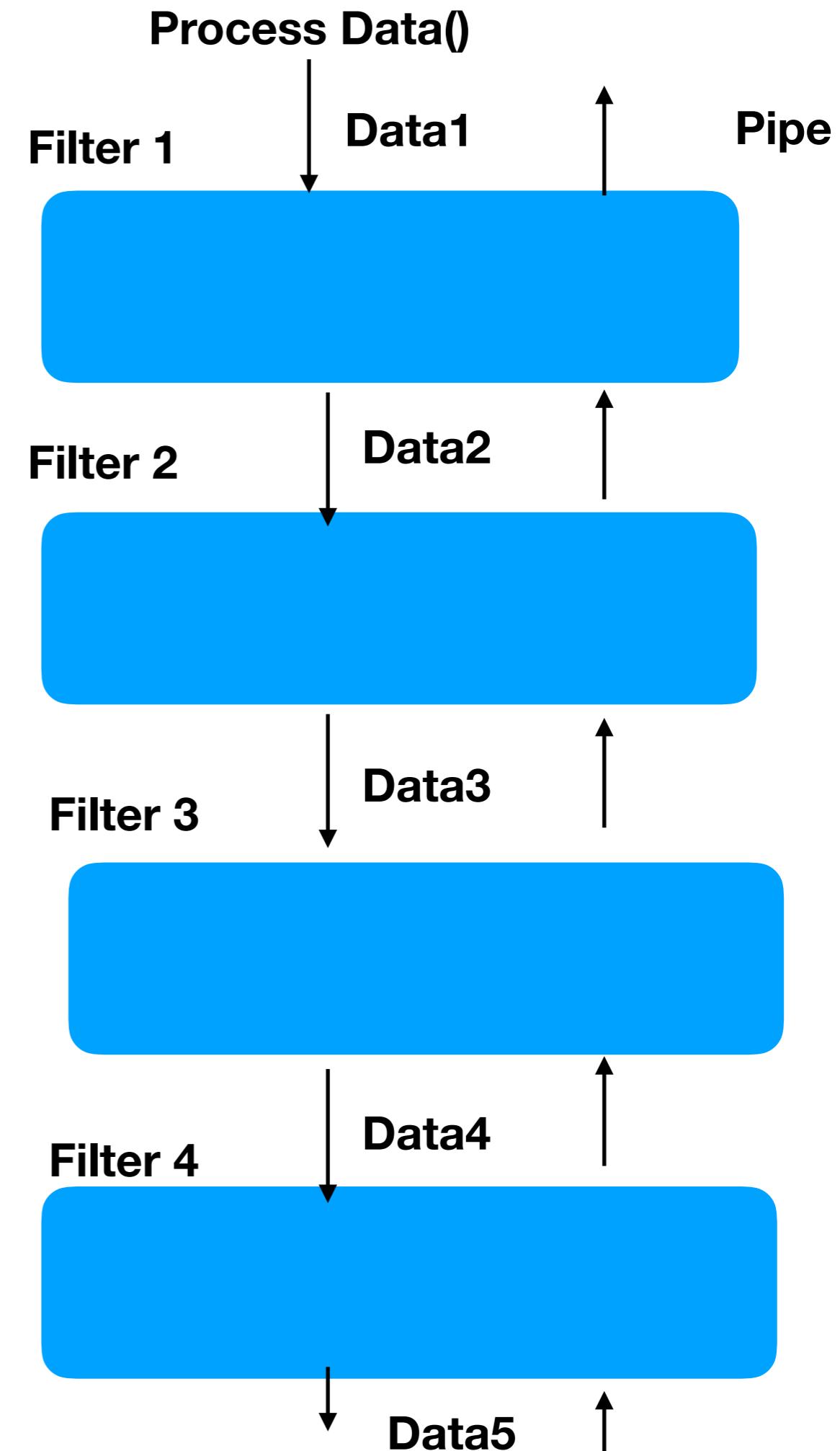
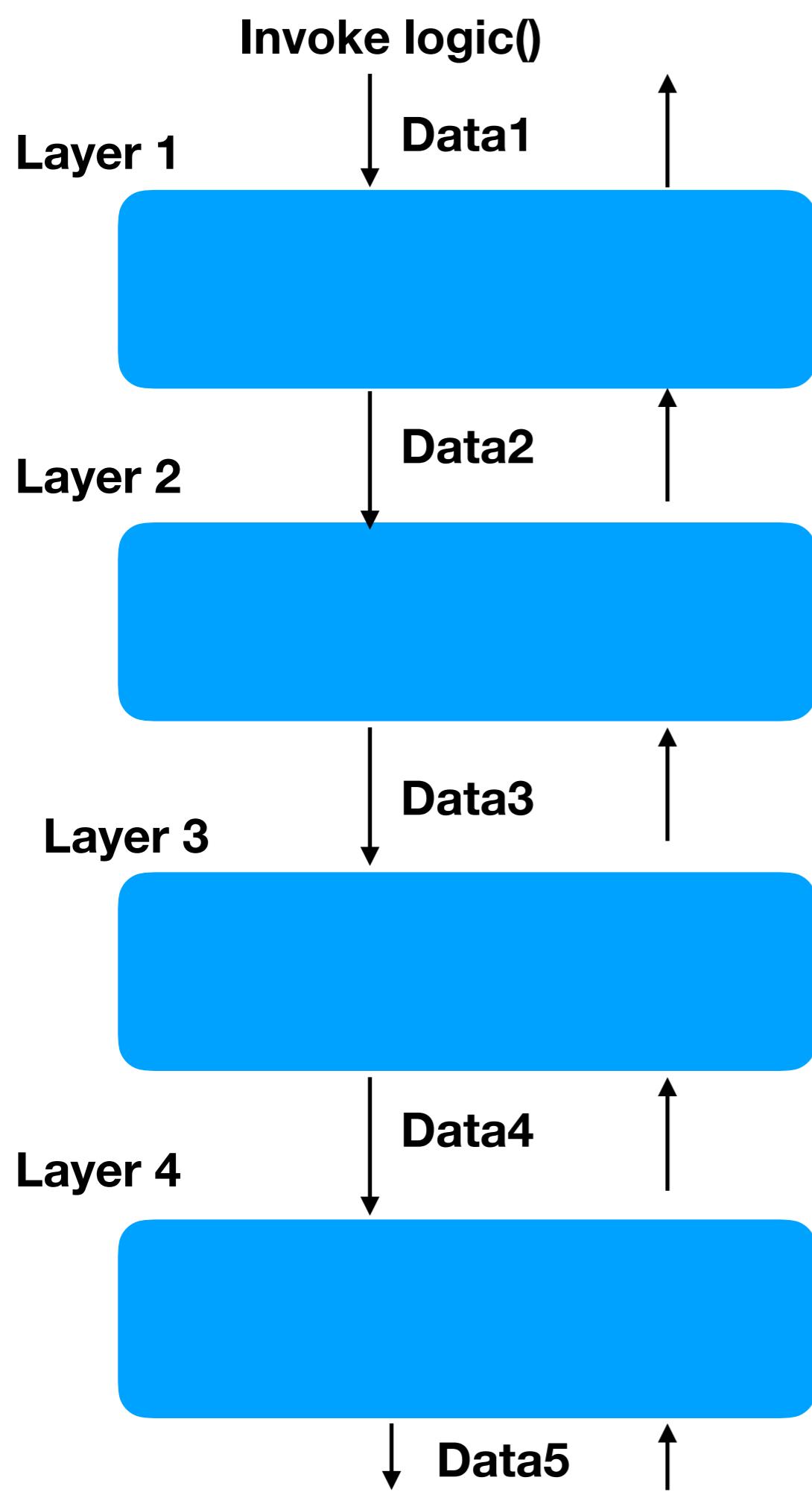


- Applications with some long-running workflows or batch operations.

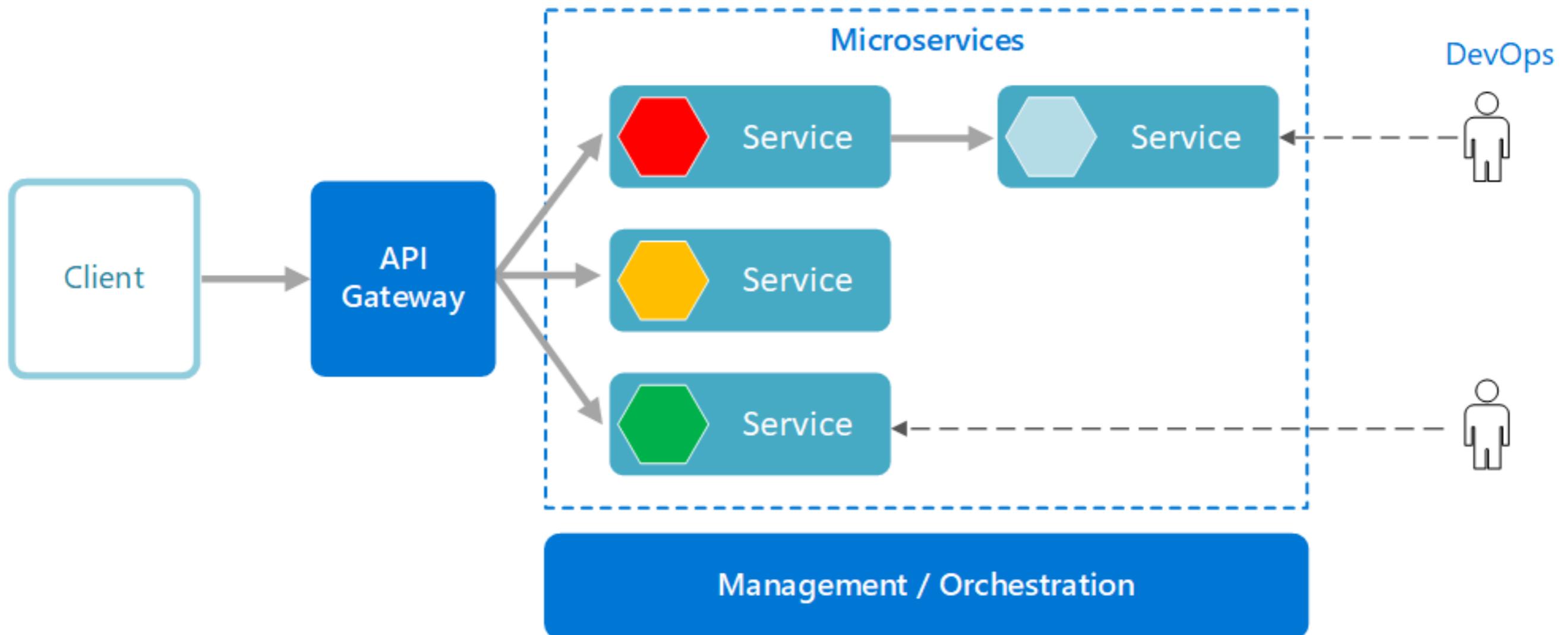


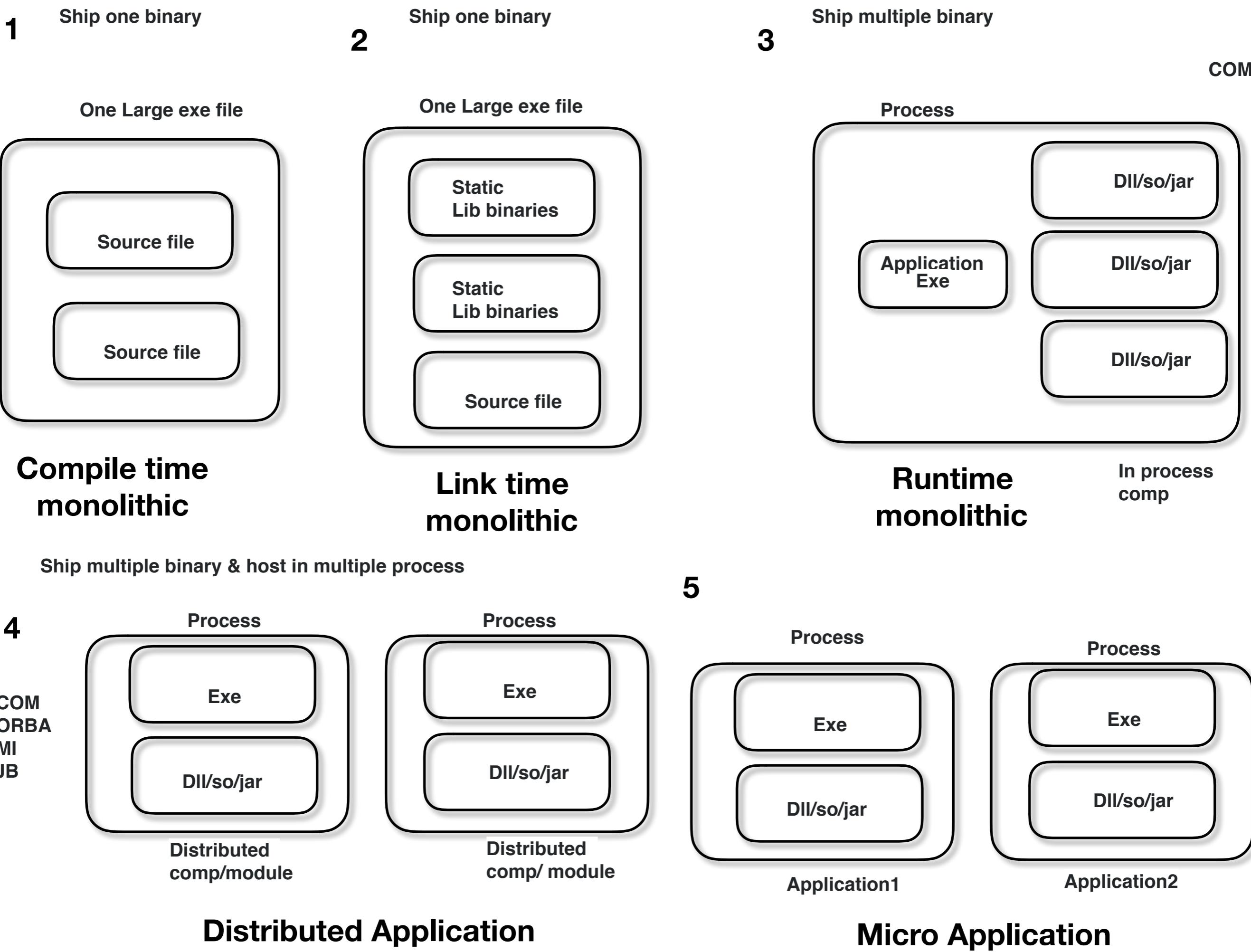
Pipe and Filter



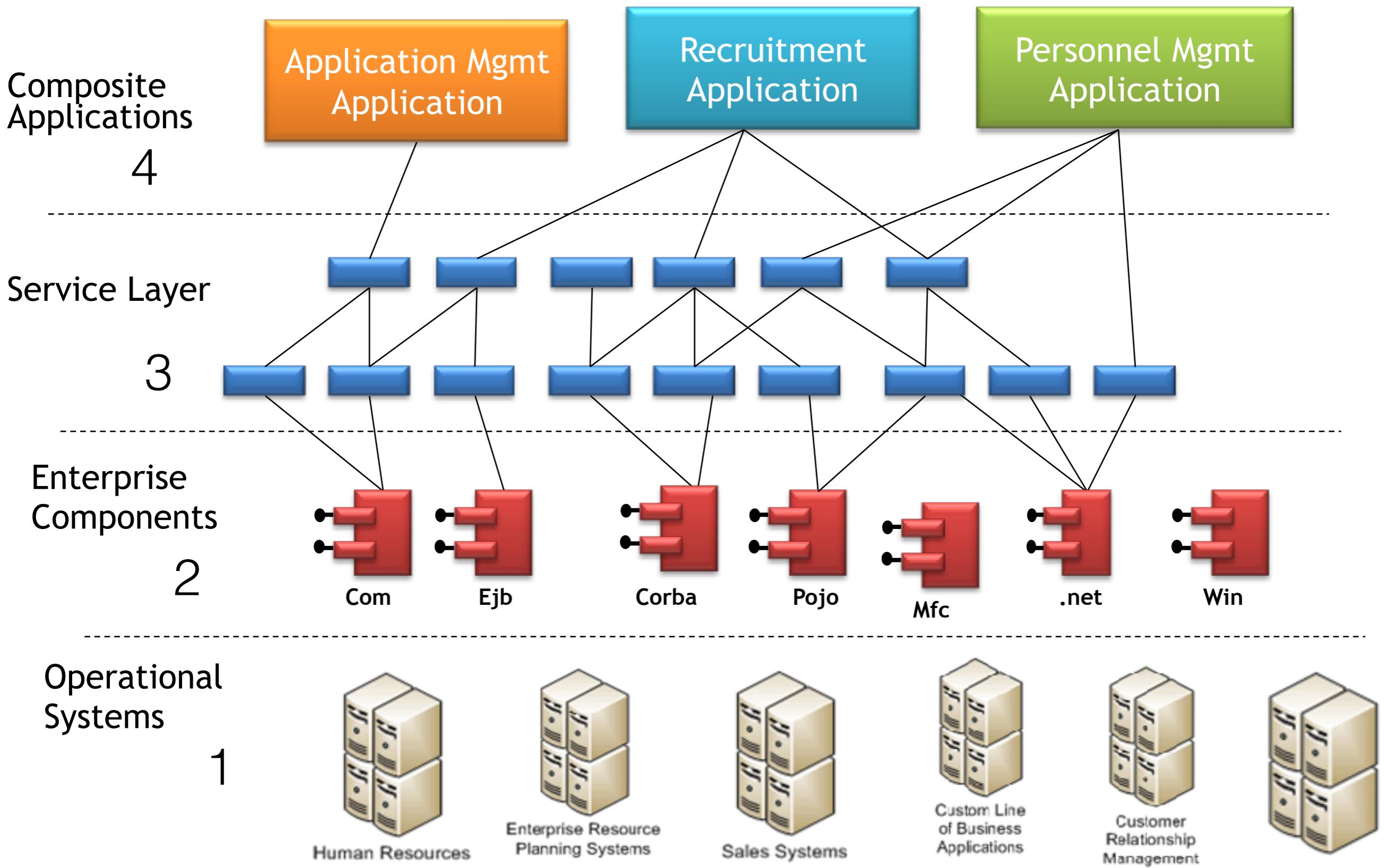


Microservice





Service Oriented Architecture



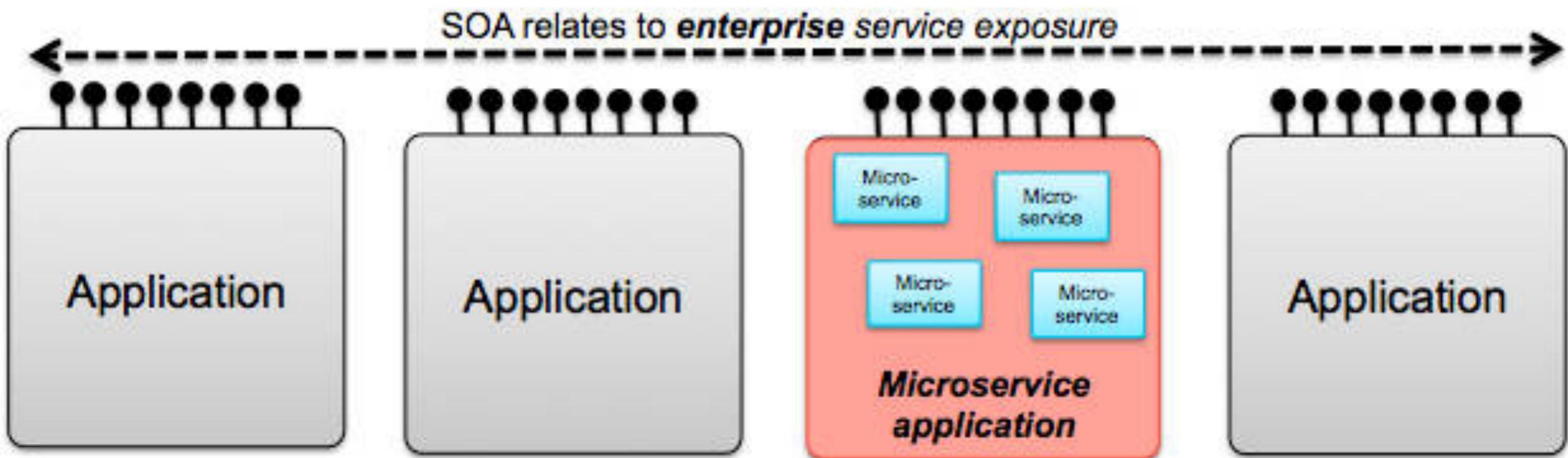
ToDo Portal
(Single Page Application)

ToDo API Service
(Api Application)

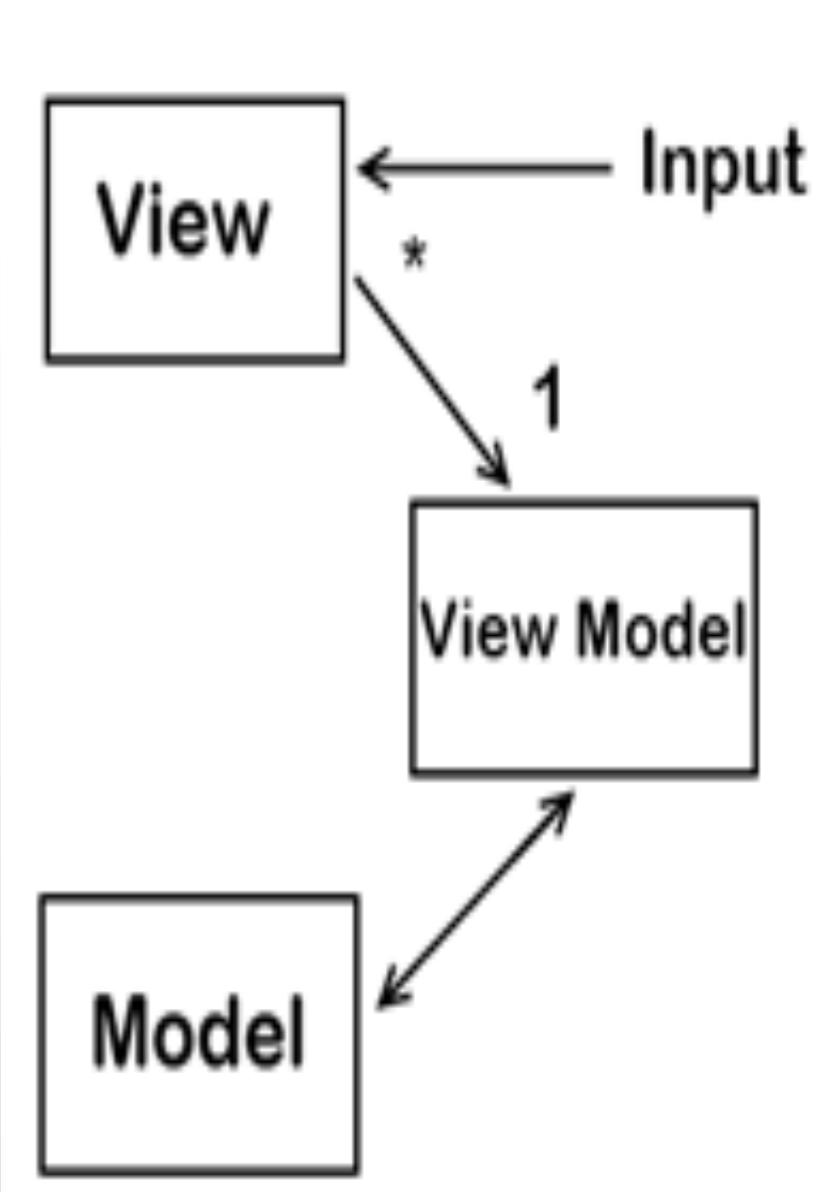
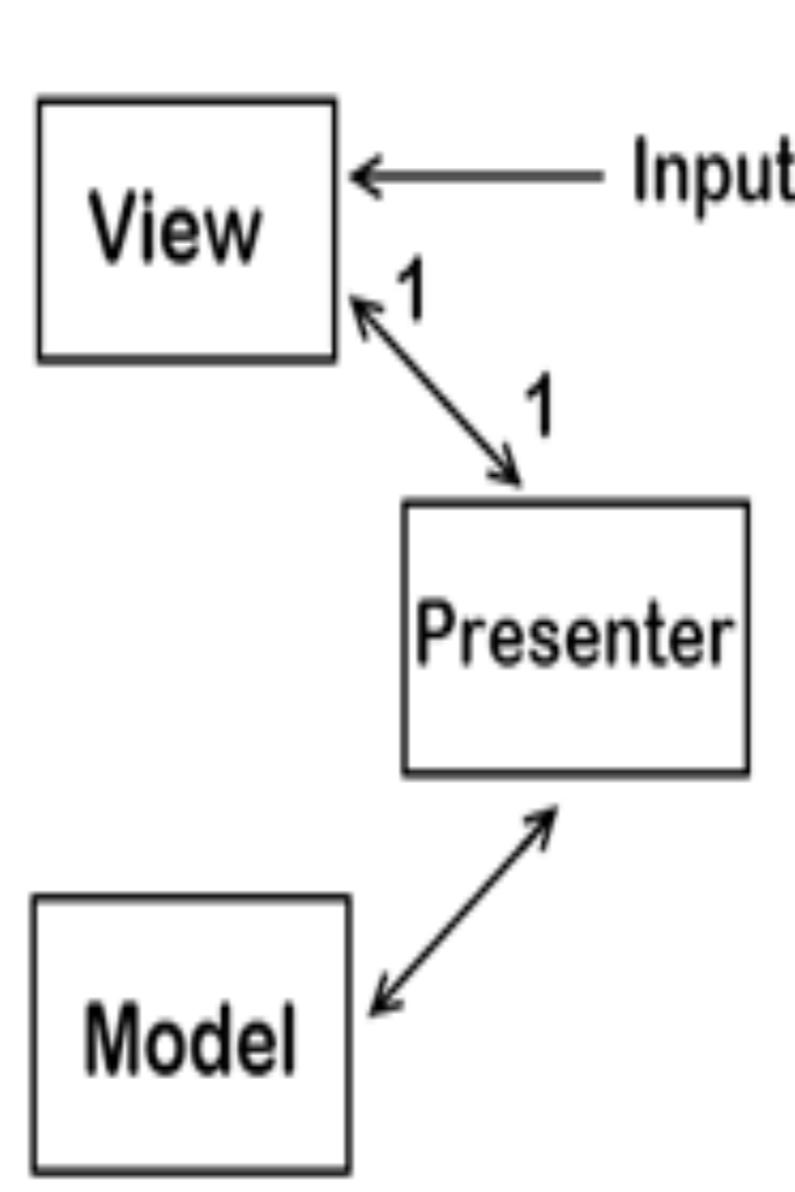
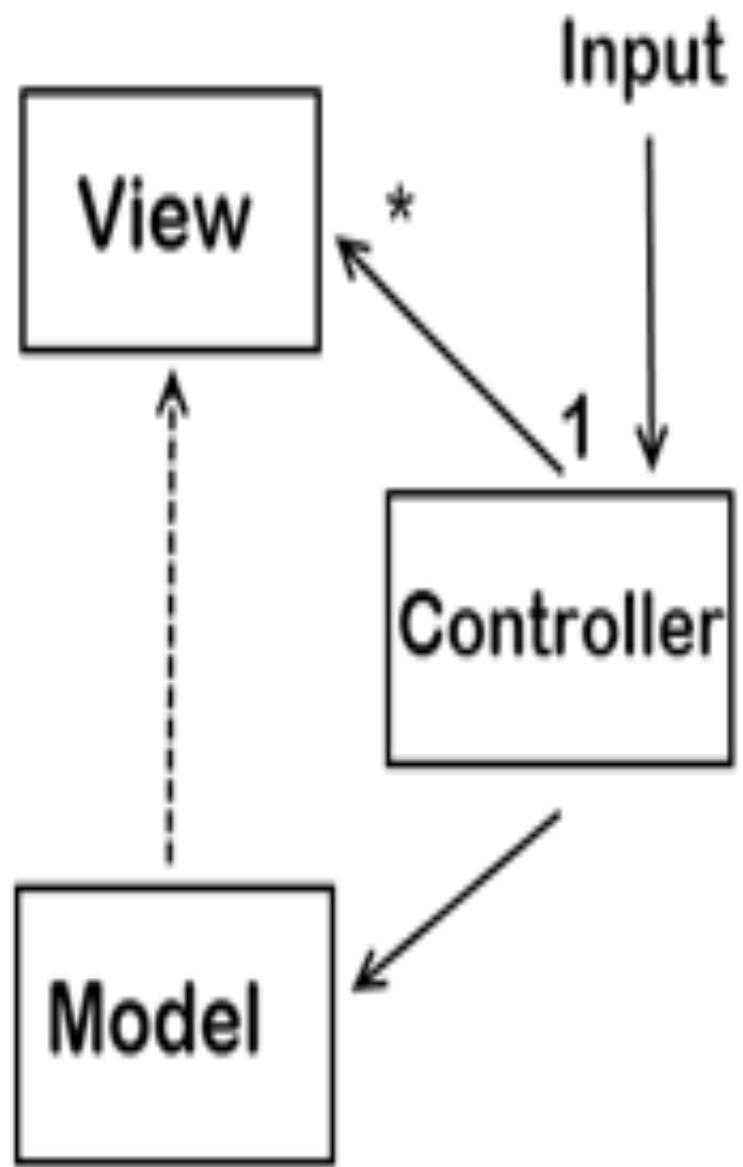


ToDo Calendar Service
(Background Application)

ToDo Prediction Service
(Background Application)



Microservices relate to
application architecture



Runtime Mono
(traditional)

Microservice

Developer Effort/ Less time

Performance (response time)

Security

Infra Cost

Break an app into smaller manageable piece

	Mod1 Mod2	App1 App2	W	Score
Share Database / Storage	Yes	No	2	
Share same Virtual Infra	Yes	No	3	
Share Source Control	Yes	No	2	
Share CI/CD (Build pipeline)	Yes	No	3	
Share domain code	No	No No No	3	
Fun Requirements	Application level	It owns	1	
SCRUM Team / Sprint	Application Scope	Its owns	1	
Acceptance Test Cases	Application Scope	Its owns	1	
Architecture	Application Scope	Its owns	1	
Technology Stack / Fwks	Application Scope	Its owns	1	

Application

Modules

Modules

Modules

3 or 4

Application

Small App

Small App

Small App

Modules

Modules

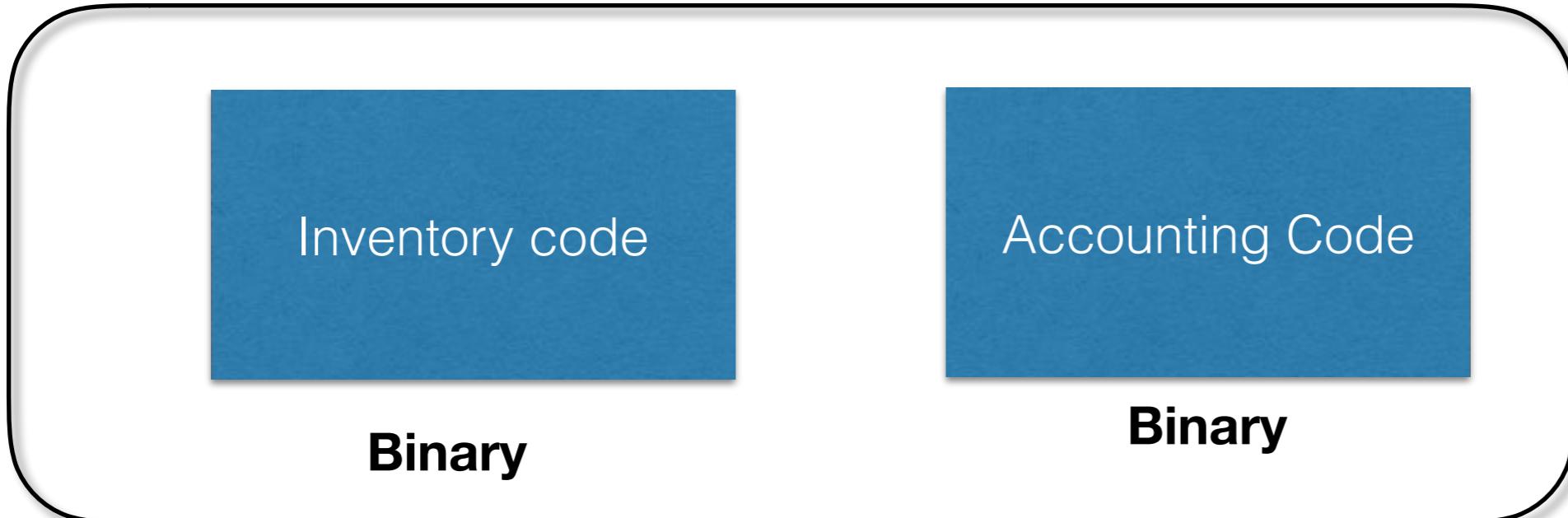
Modules

Modules

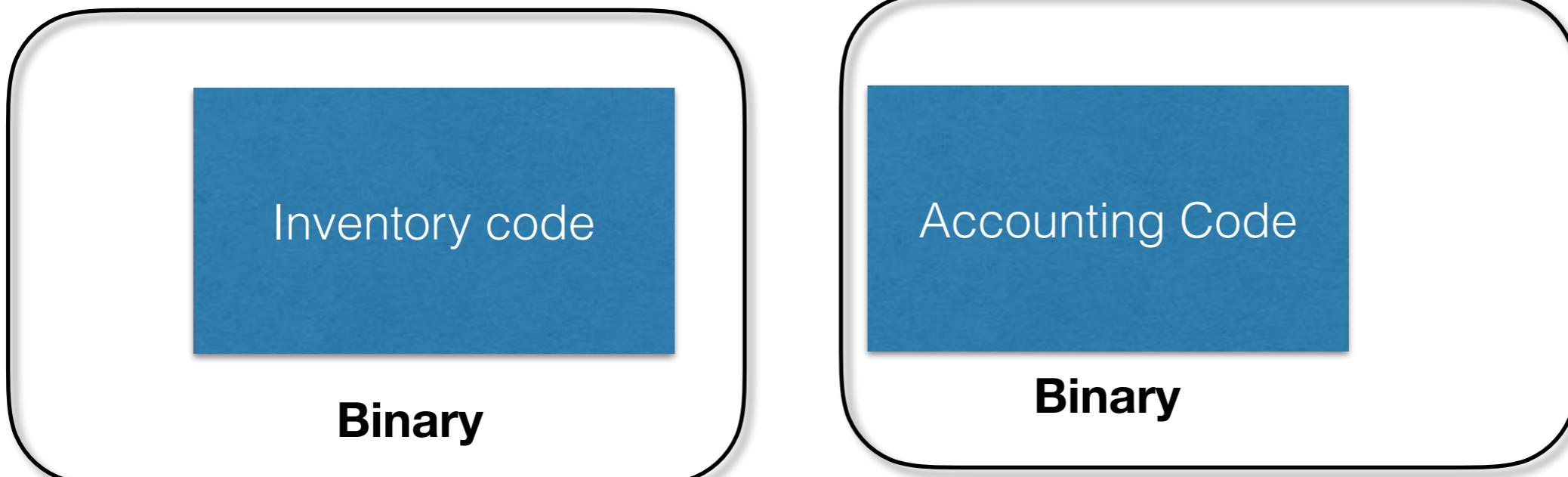
Modules

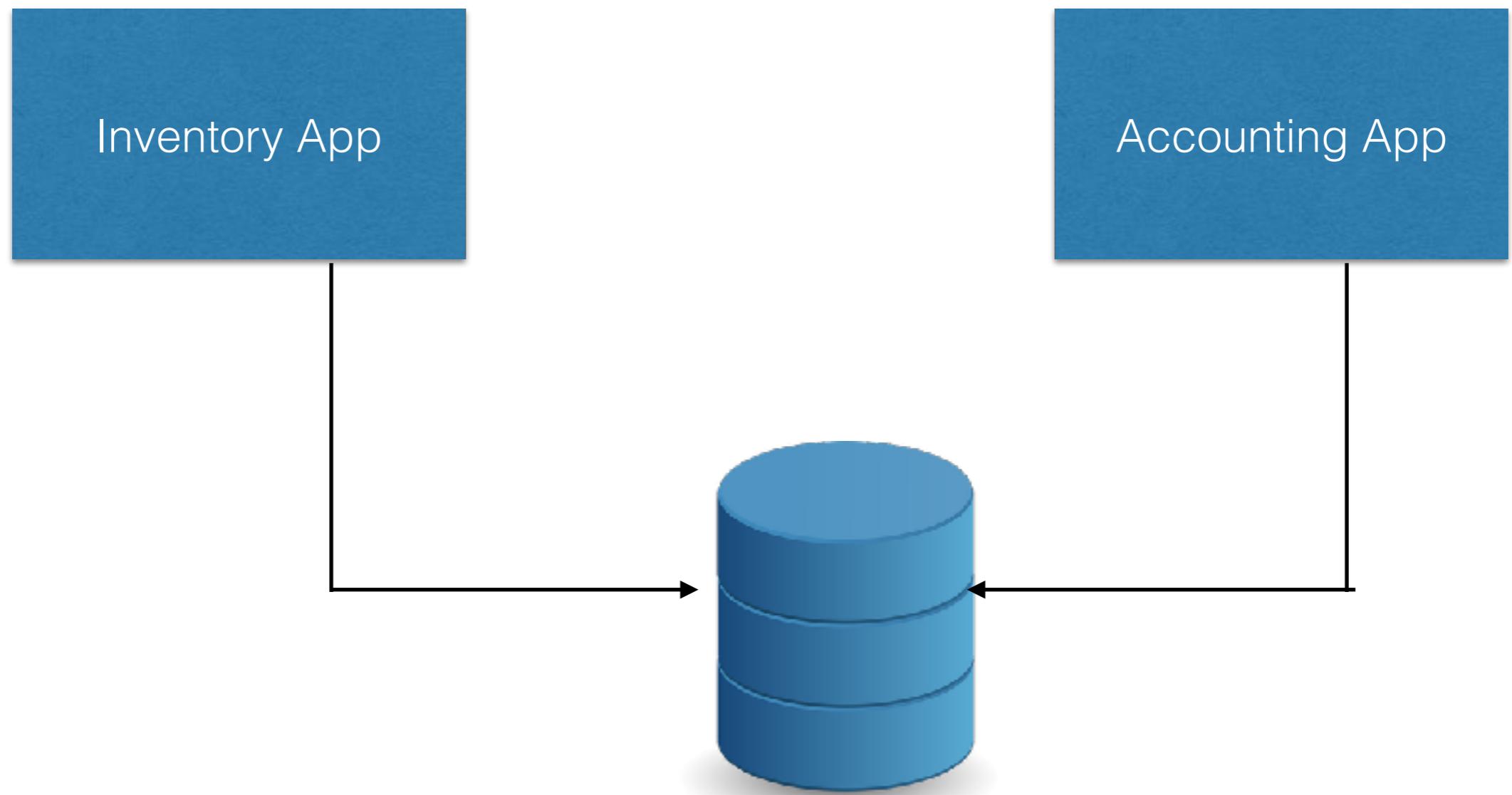
Modules

App

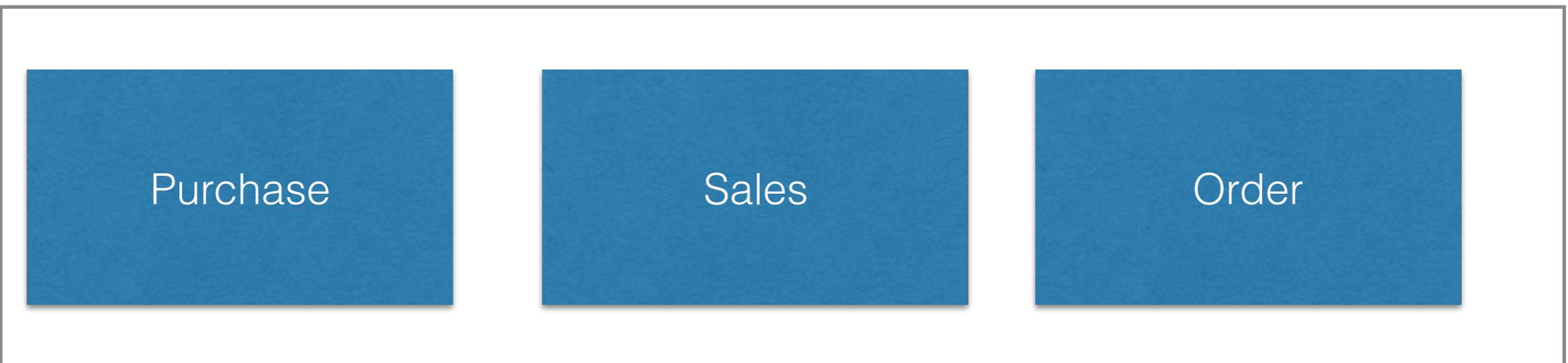


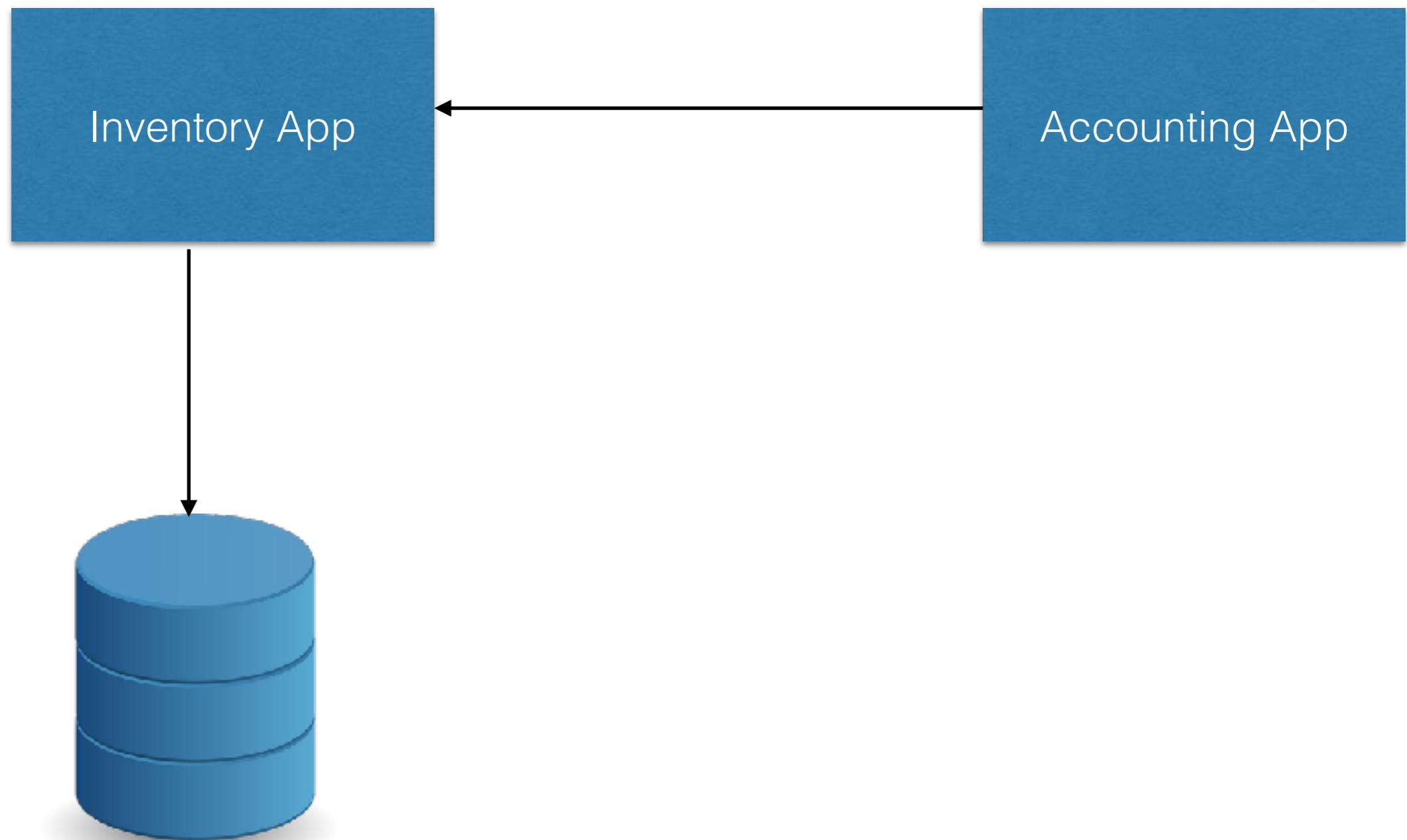
App

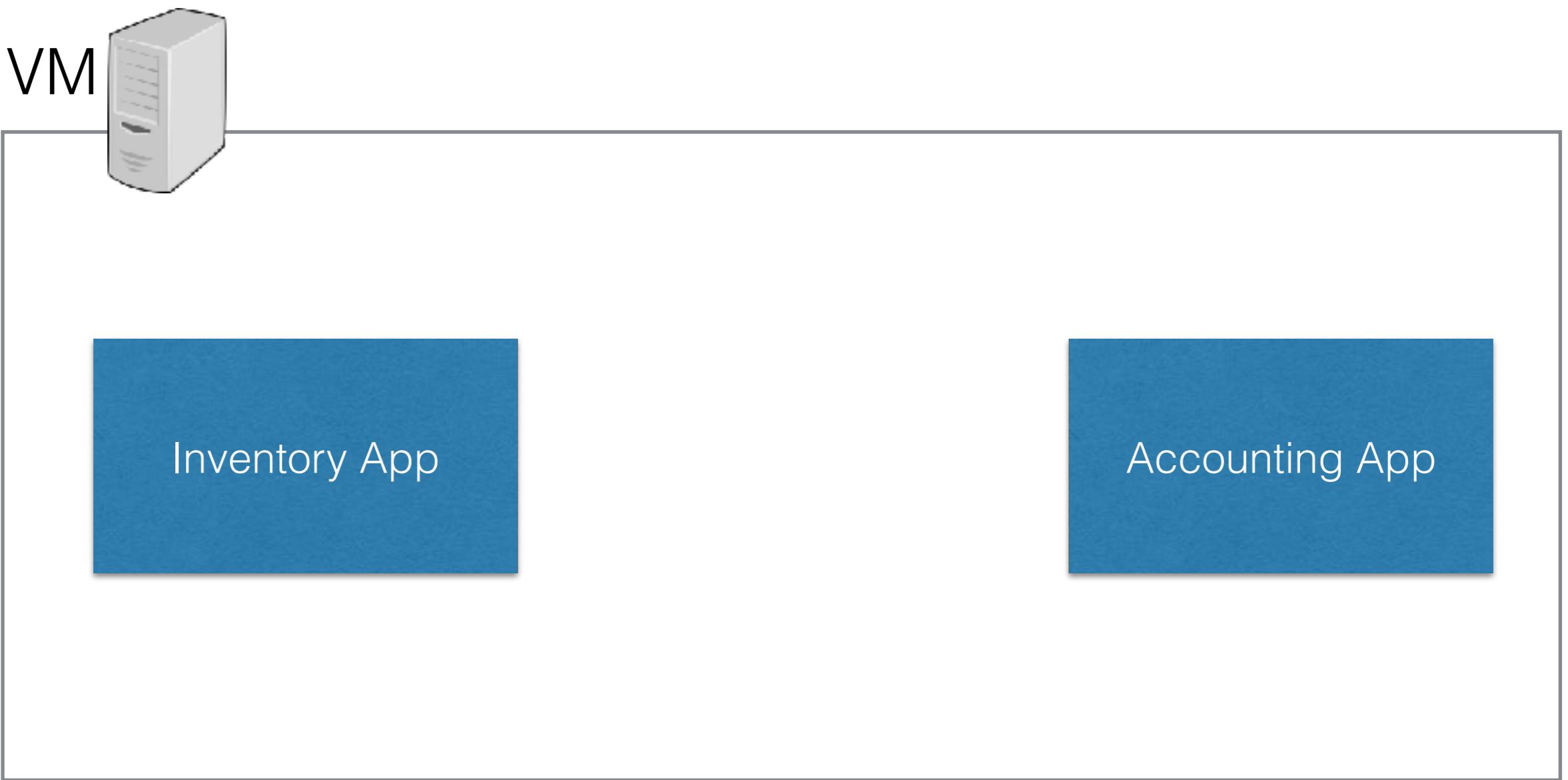




Inventory Application







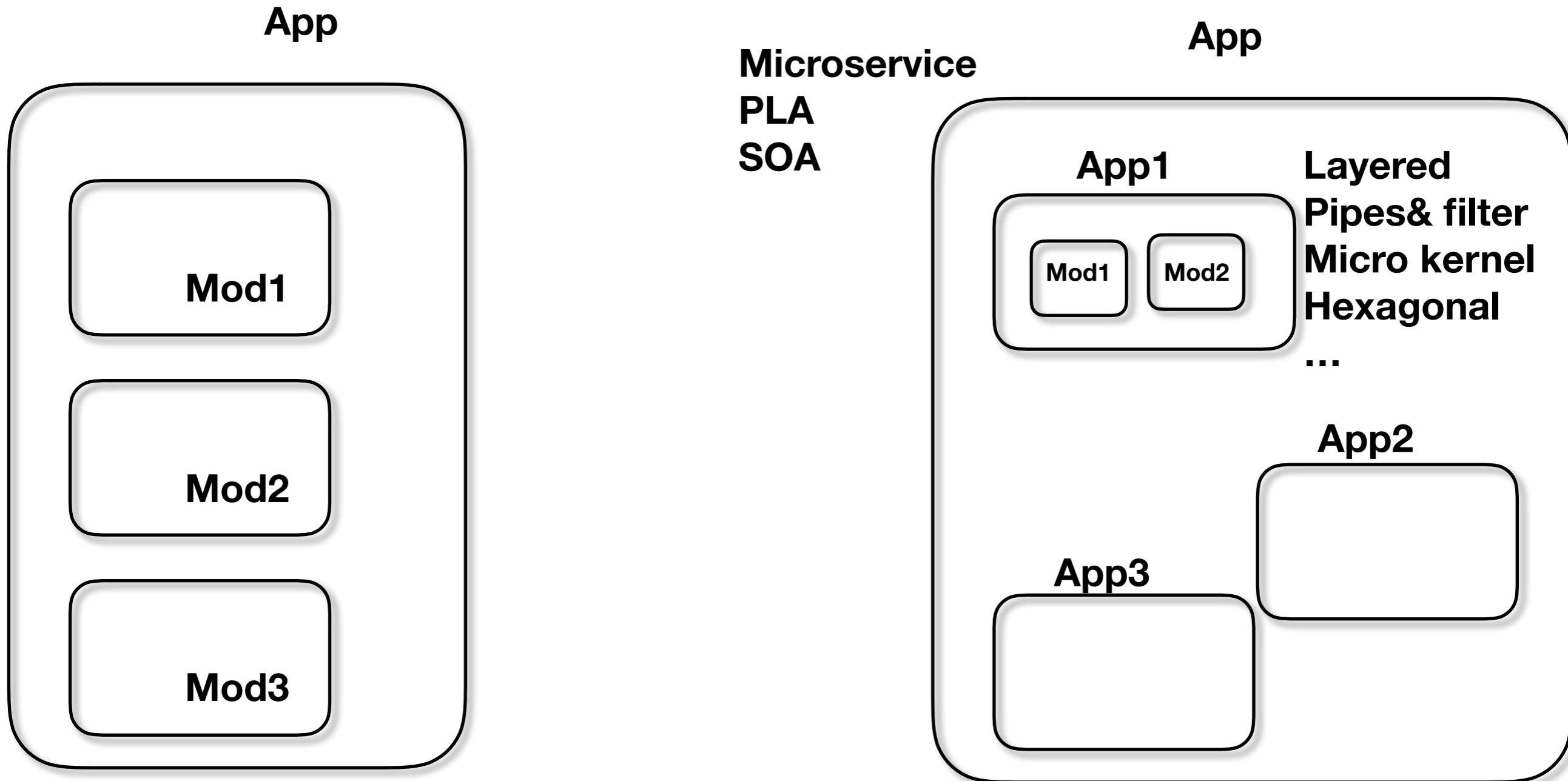
APP

APP

APP

APP

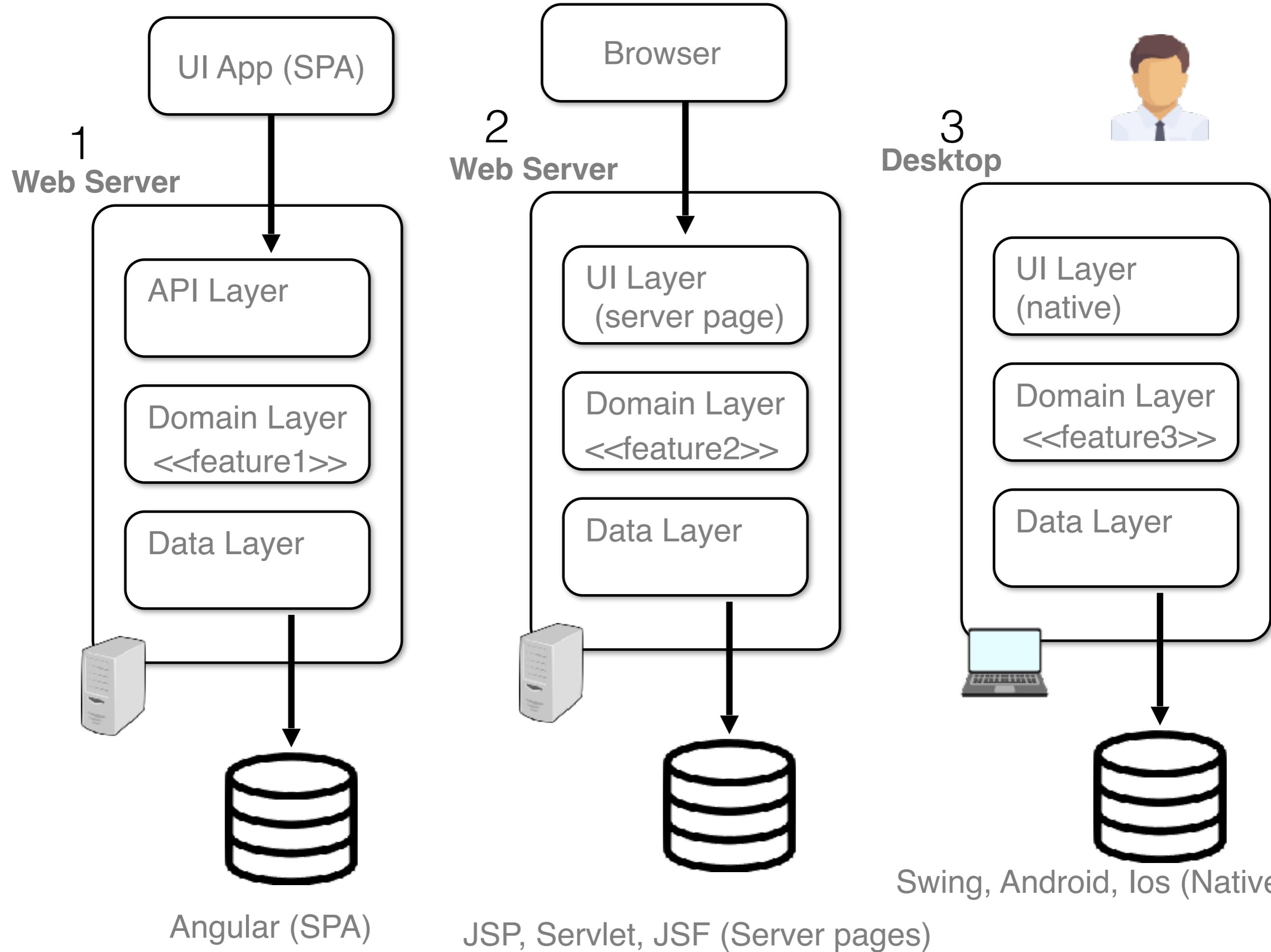
APP

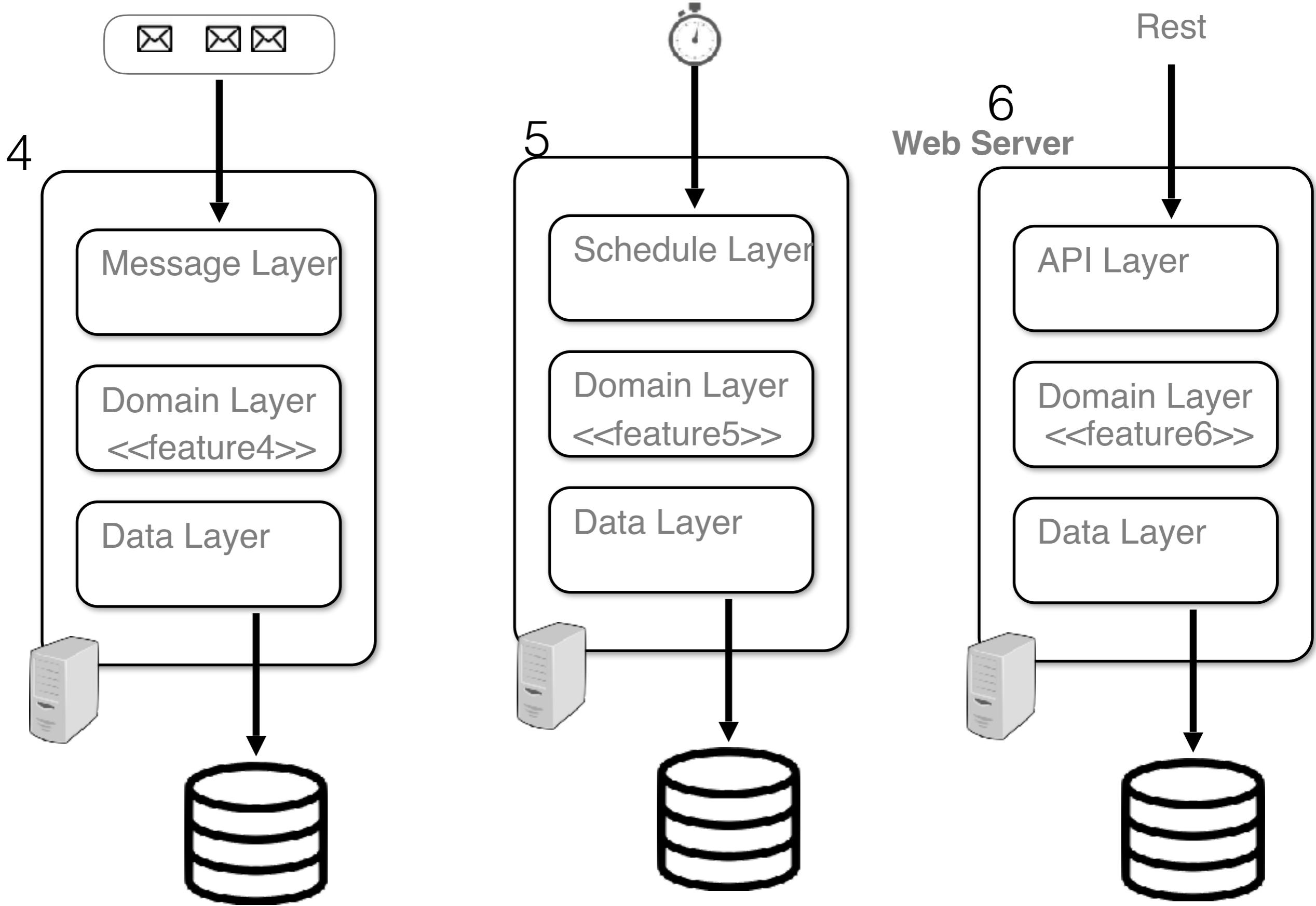


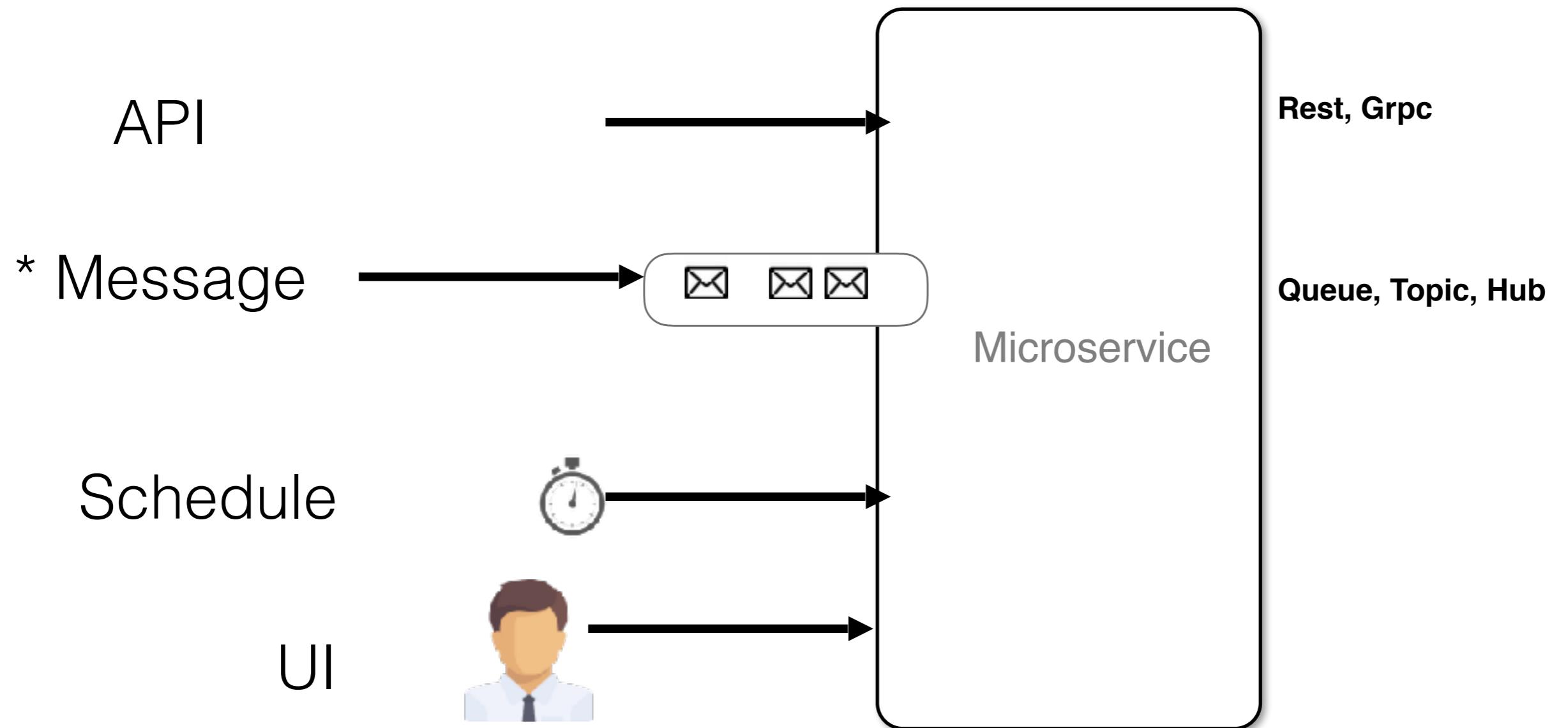
Types of Microservice

App

App



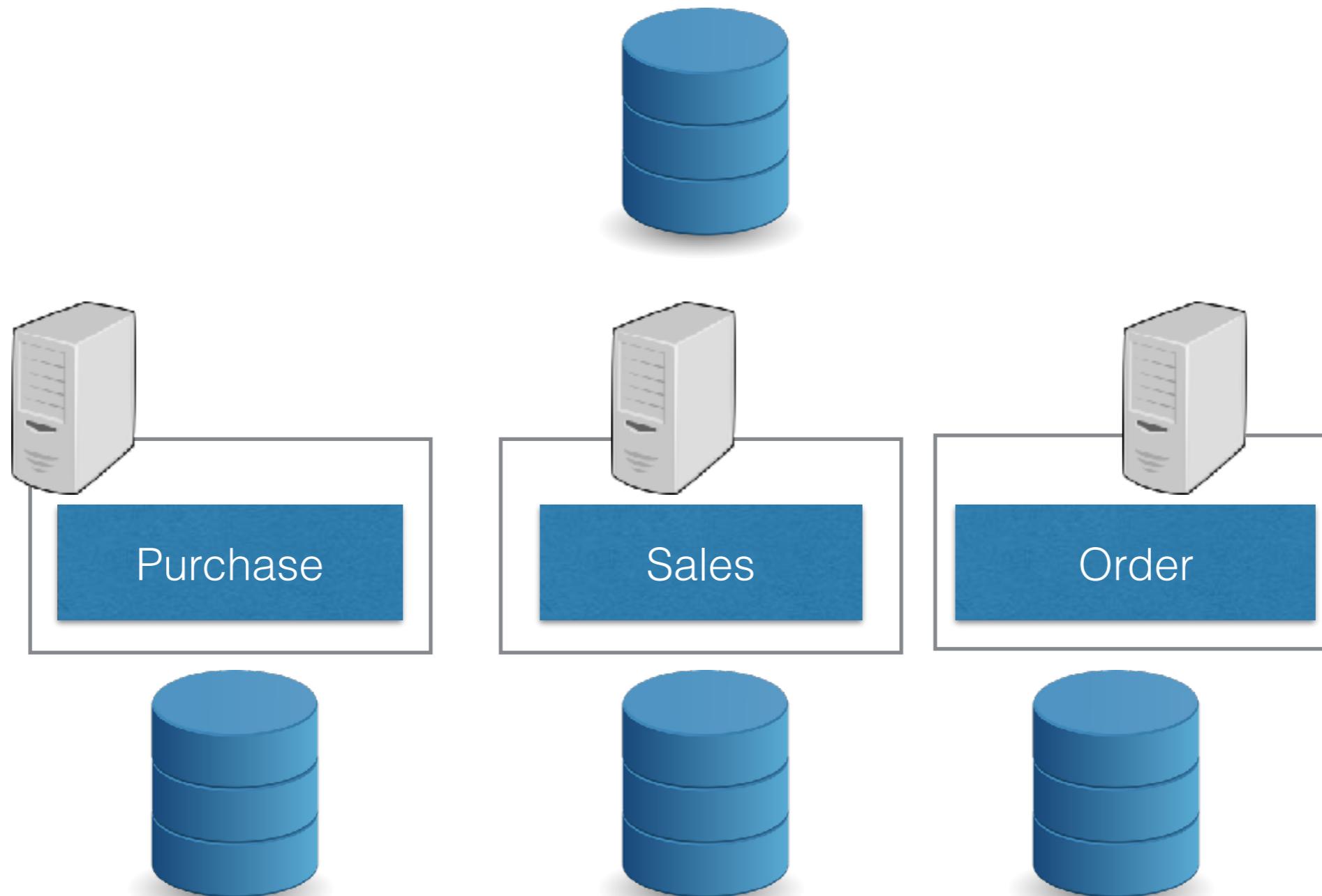
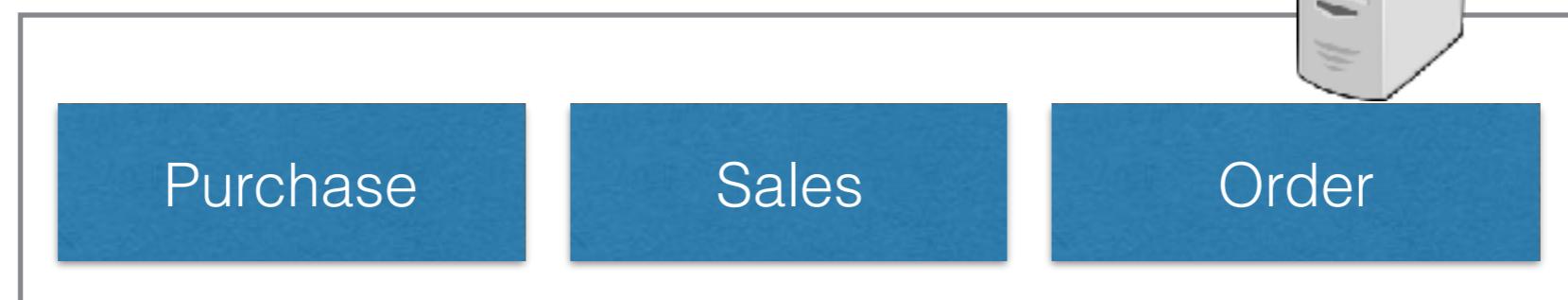




	Pros/ Cons	Solution
Development time	--	
Micro-service practices Learning Curve	--	
Resource Performance (CPU, Memory, I/O)	--	Grpc, protobuf, ...
Db Transaction Management	--	SAGA
Views / Report / Dash board/ join	--	Materialized View
Infra Cost	--	Containerization
First time Deployment effort	--	Automation
Debugging, Error Handling (End to End)	--	Distributed Tracing (Jaeger)
Integration Test		
Log Mgmt (debug/ error)	--	Centralize EFK, ELK, Splunk
Config Mgmt	--	Centralize Consul, Zookeeper, ...
Authentication (who)	--	Centralize (Oauth2, OpenID connect, SAML, ...)
Authorization (what can they do)	--	Centralize (Claims)
Audit Log mgmt (who accessed what)	--	Centralize (Event Store, ...)
Monitoring / Alerting	--	Centralize
Data Security and Privacy (transit, storage)	--	
Build Pipeline (CI/CD)	--	Automation
Agile Architecture (Agility to change)	+++	fwk, technology, platform, ...
Feature Shipping (Agility to ship)/ CD	++	Timing of when to goto production
Scalability (volume - request, data, compute)	+	
Availability	+	
Ability to do Polygot	+	
Fault Isolation	+	

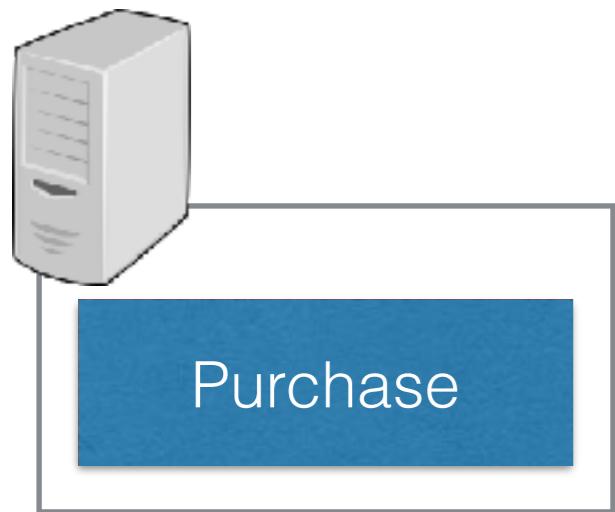
	Pros/ Cons	Solution
Development time	--	
Micro-service practices Learning Curve	--	
Resource Performance (CPU, Memory, I/O)	--	
Db Transaction Management	--	SAGA
Views / Report / Dash board/ join	--	Materialized View, CQRS
Infra Cost	--	Containerization
First time Deployment effort	--	Automation (chef, puppet)
Debugging, Error Handling (End to End)	--	Distributed Tracing (Jaeger, ...)
Integration Test	---	
Log Mgmt (debug/ error)	--	Centralize - ELK, EFK, Splunk
Config Mgmt	--	Centralize - Consul
Authentication (who)	--	Centralize - OAuth2, SSO, SAML, ...
Authorization (what can they do)	--	Centralize - ?
Audit Log mgmt (who accessed what)	--	Centralize - Event Sourcing
Monitoring / Alerting	--	Centralize
Data Security and Privacy (transit, storage)	--	
Build Pipeline (CI)	--	Automation
Agile Architecture (Agility to change)	+++	
Feature Shipping (Agility to ship)/ CD	++	
Scalability (volume - request, data,	+	
Availability	+	
Ability to do Polygot	+	
Fault Isolation	+	

Inventory Application



1 phase commit

```
db.Begin()  
Cmd  
Cmd  
Cmd  
db.Commit()
```



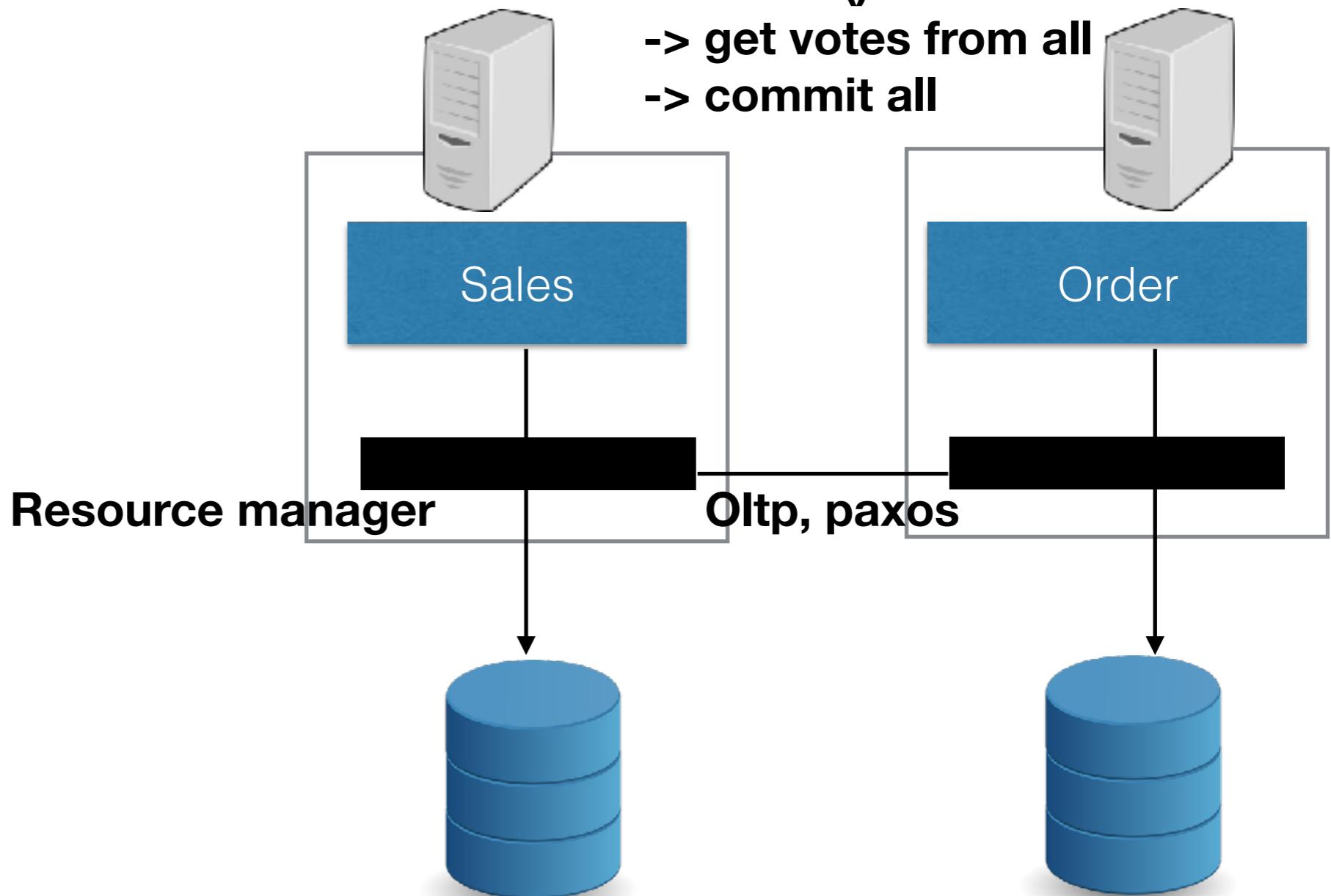
2 phase commit

```
rm.Begin()  
Cmd on db1  
Cmd on db2  
Cmd on db 1
```

...

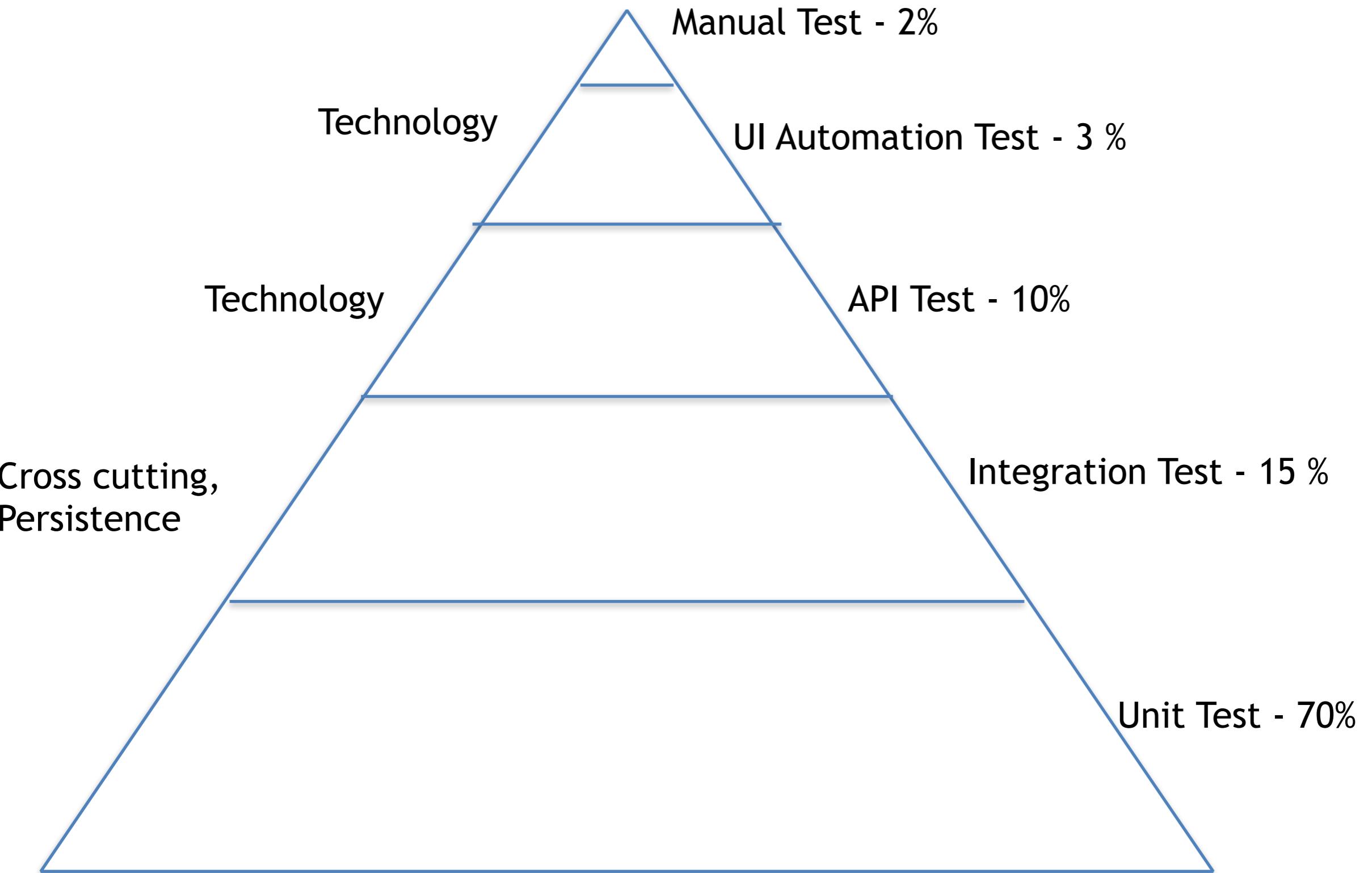
```
rm.Commit()
```

- > get votes from all
- > commit all



Decentralize Domain
Centralize Tech

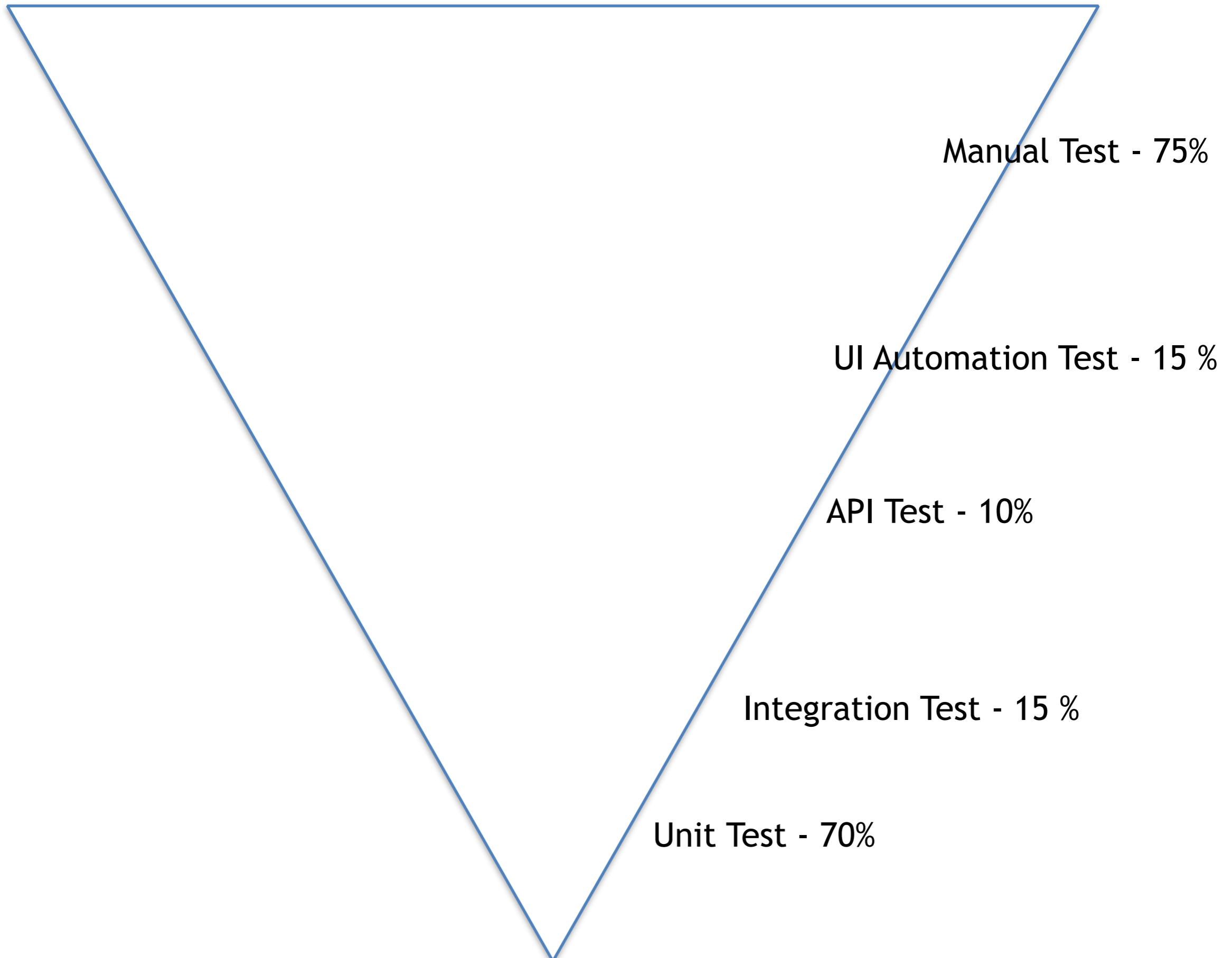
Pyramid test

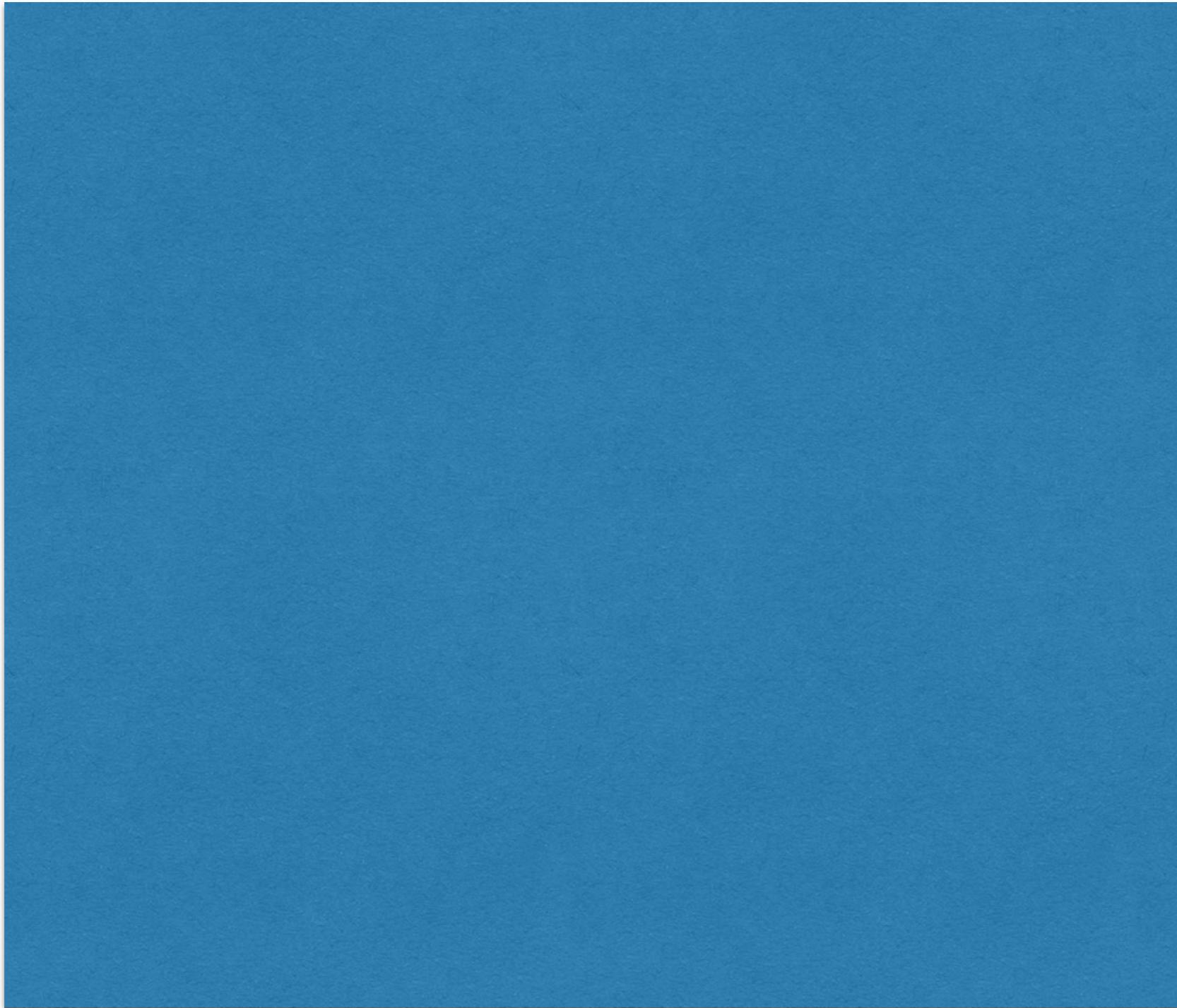


Unit Test

- Documentation for Code
- Design Code
- Regression
- Find Bugs

Its working don't touch it





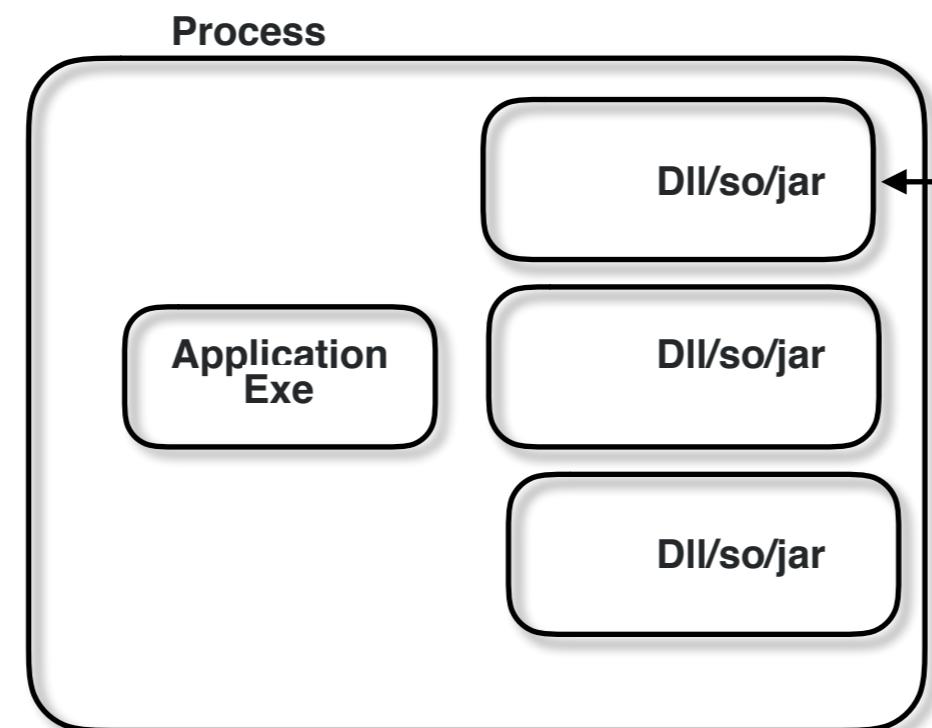
Manual Test - 2%

UI Automation Test - 3 %

API Test - 10%

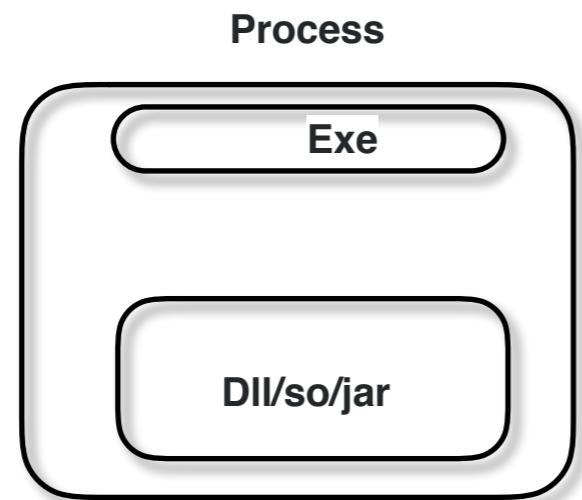
Integration Test - 15 %

Unit Test - 70%

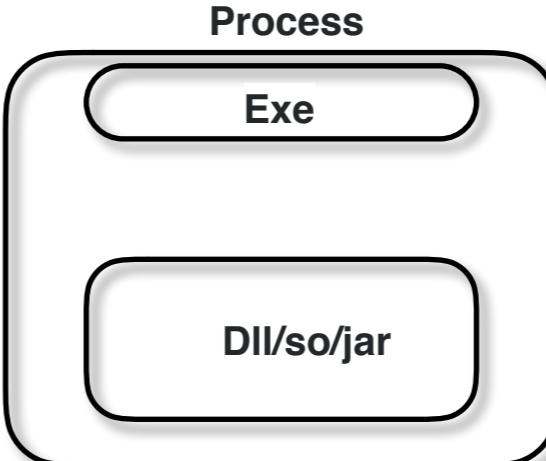


**Runtime
monolithic**

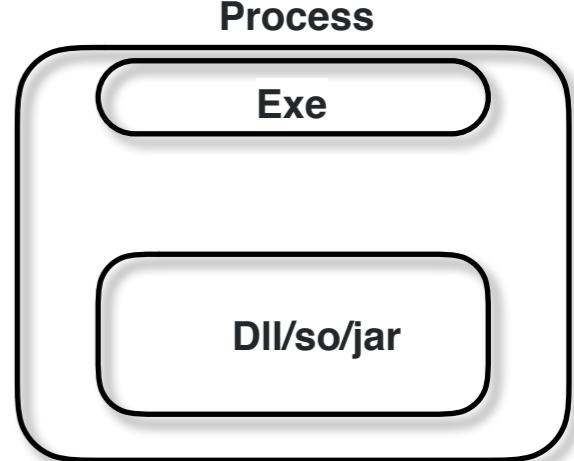
In process
comp



Application1



Application2



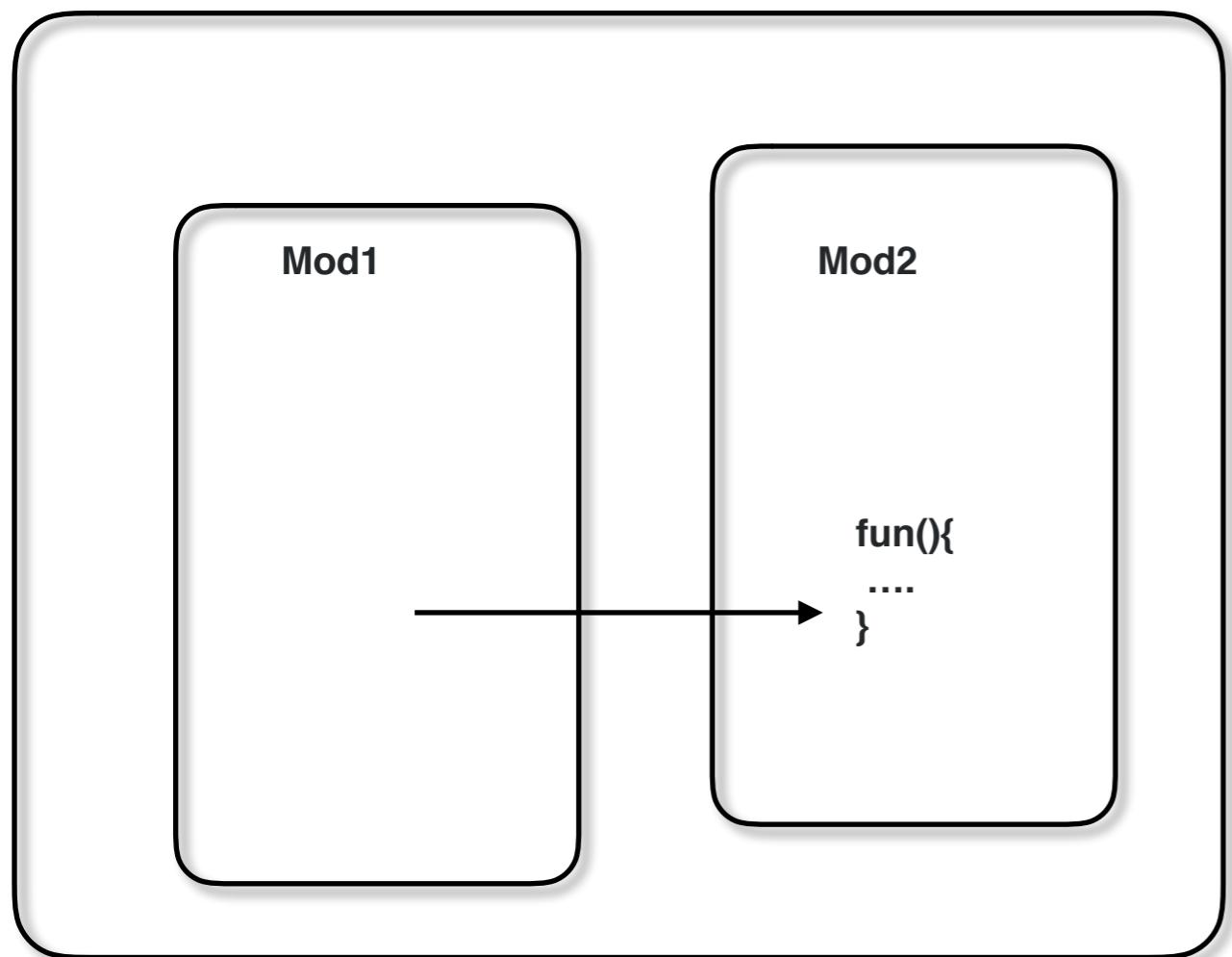
Feature A

Micro Application

1. Performance

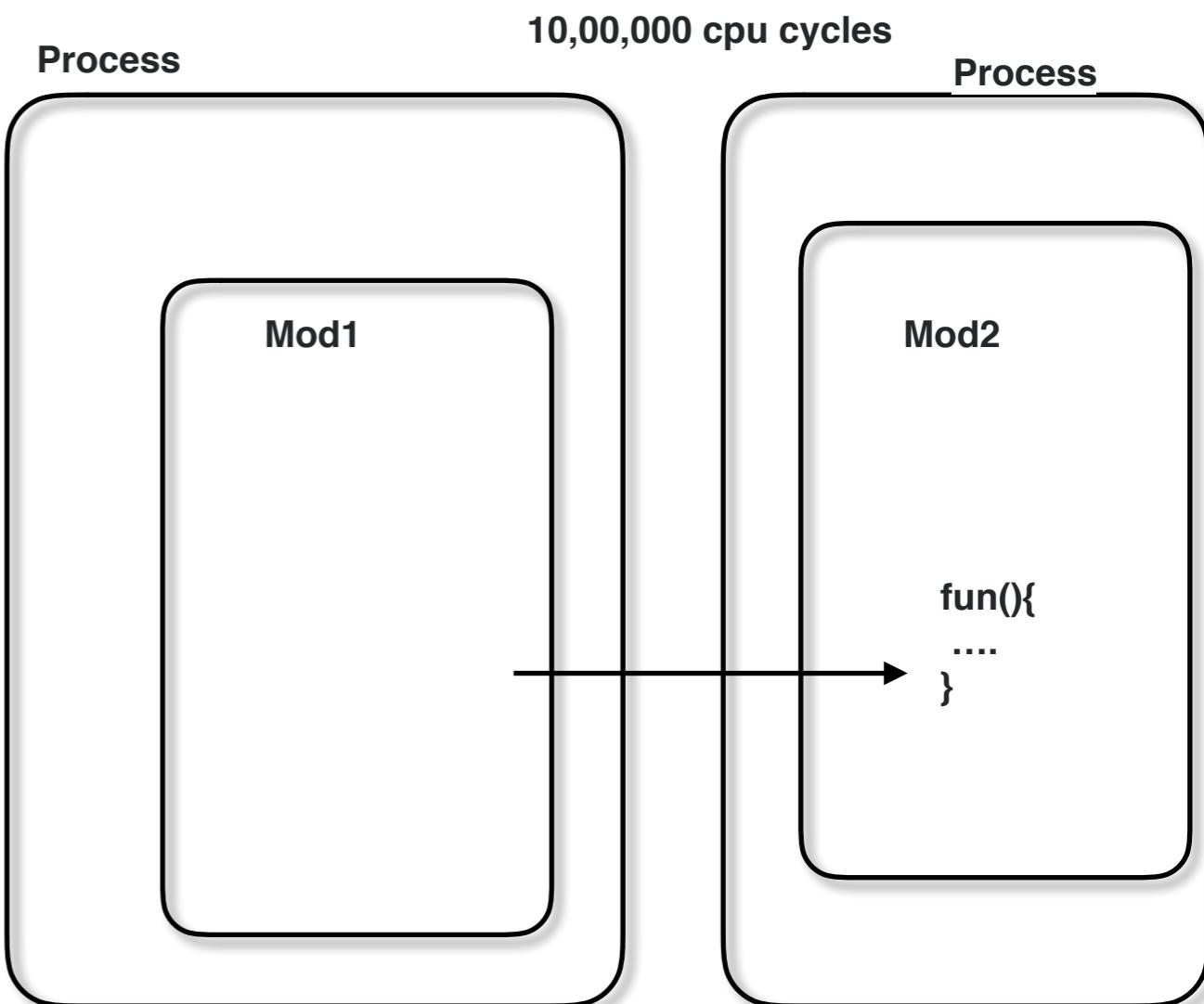
Operation	Cpu Cycles
• 10 + 12	3
• Calling a in-memory Method	10
• Create Thread	2,00,000
• Destroy Thread	1,00,000
• Database Call	40,00,000
• Distributed Fun Call	20,00,000
Write to disk	10,00,000

Process



10 cpu cycles

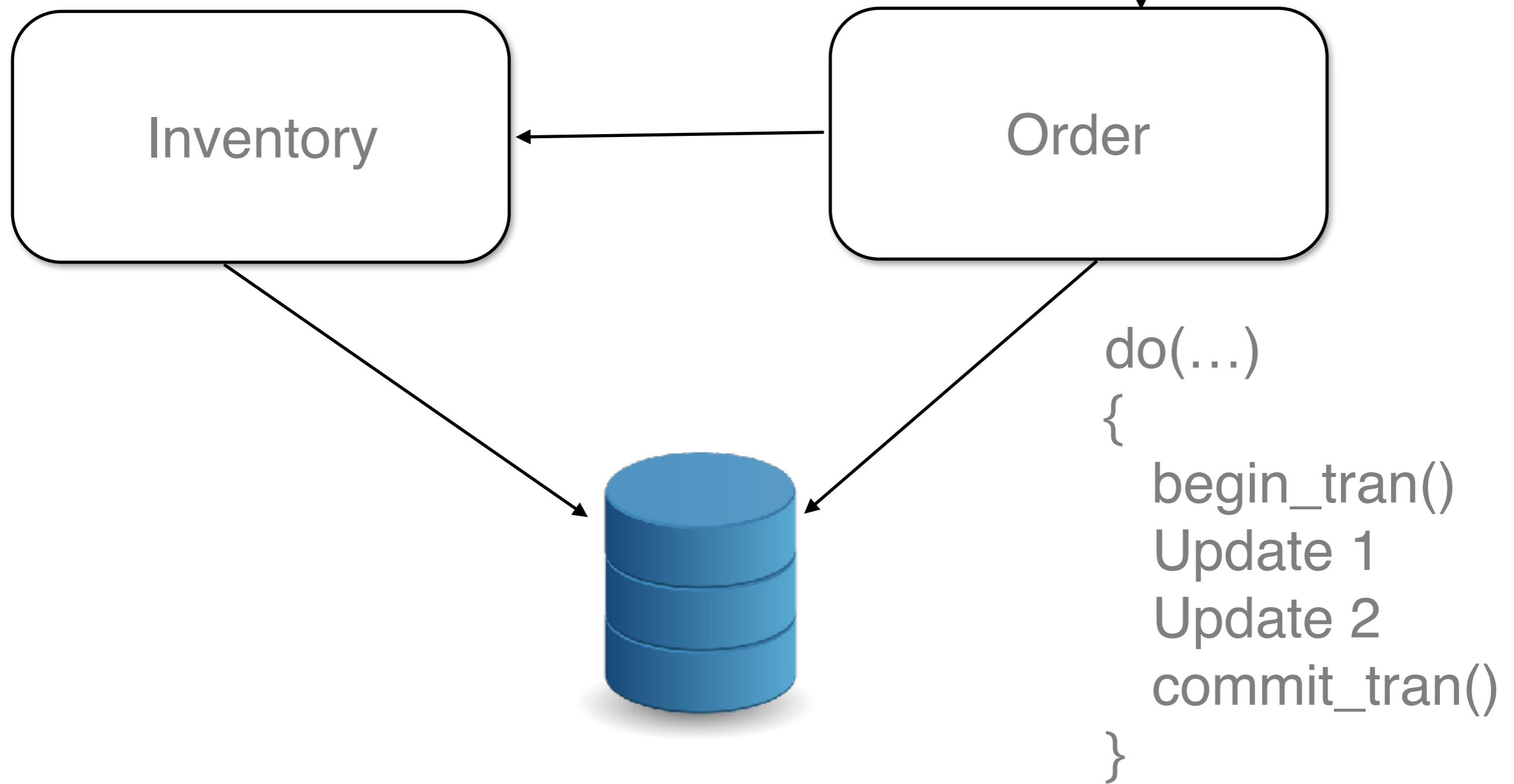
Process



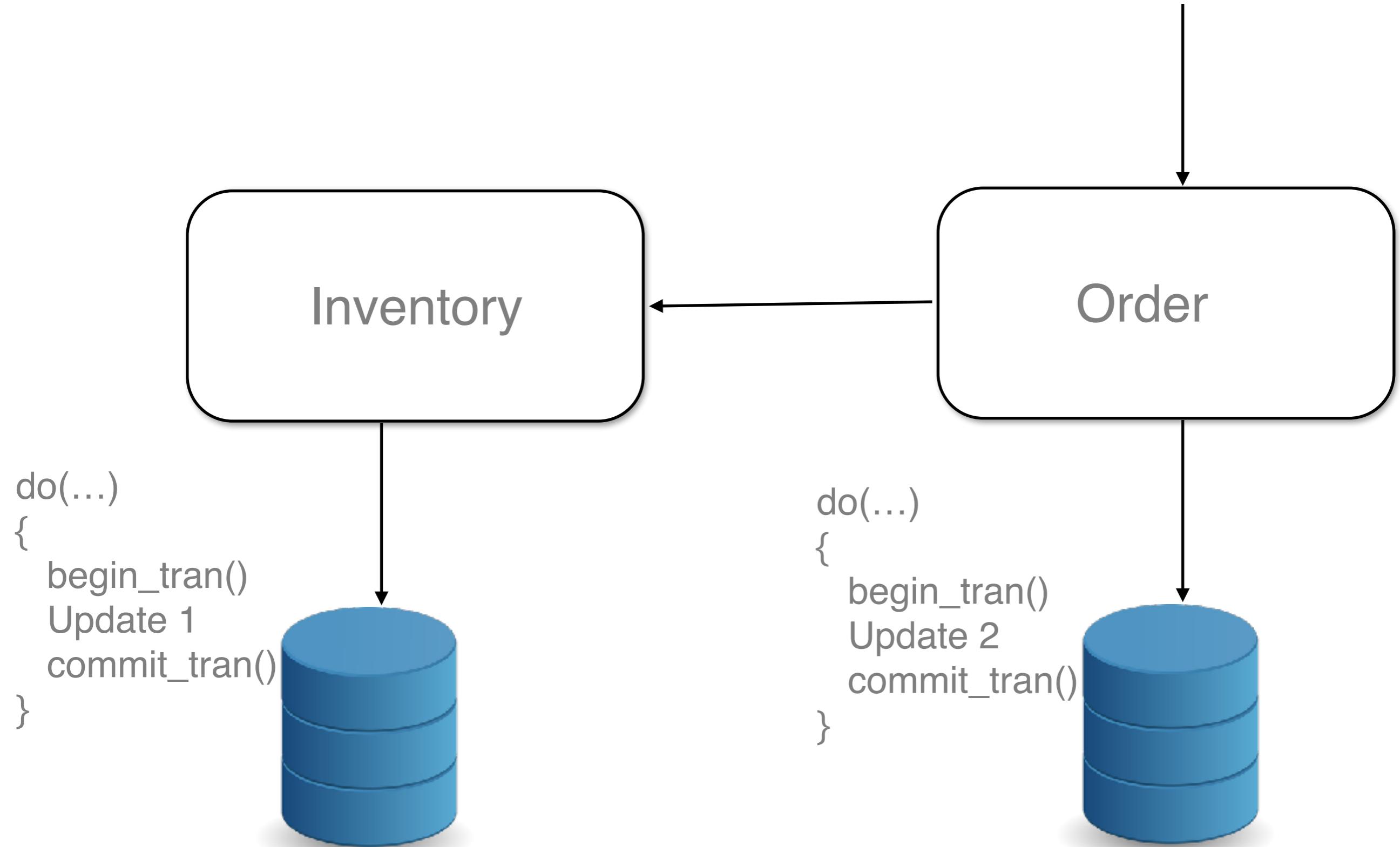
10,00,000 cpu cycles

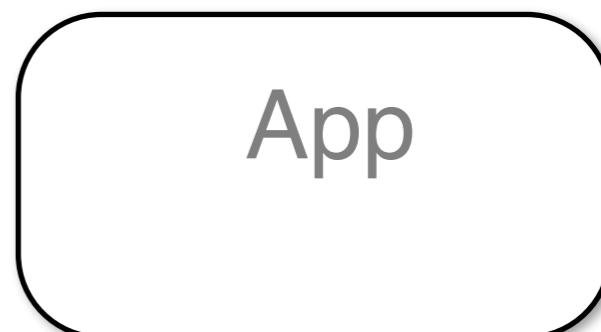
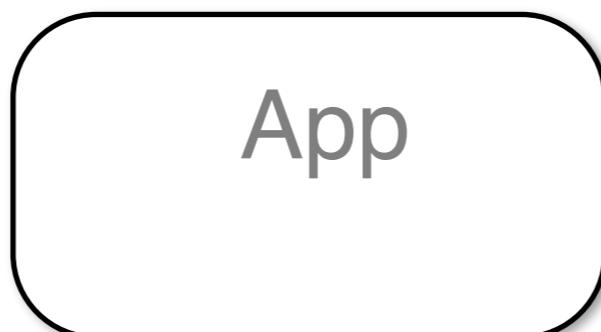
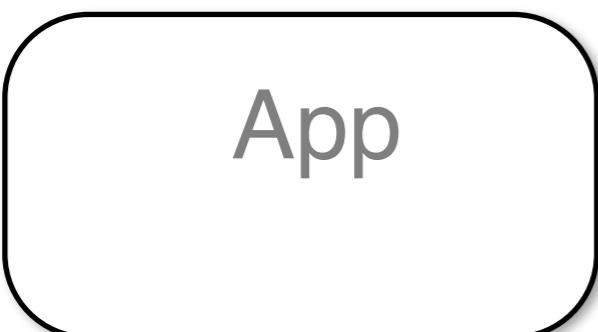
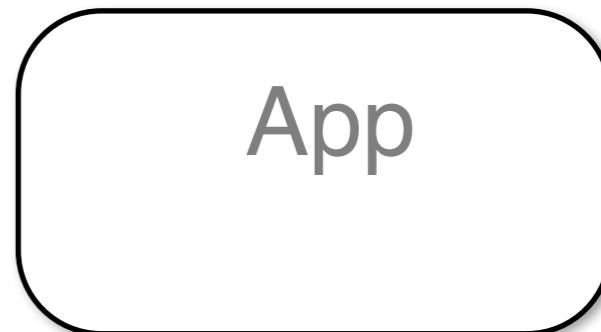
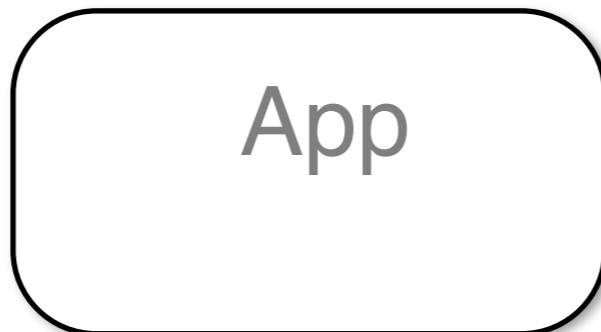
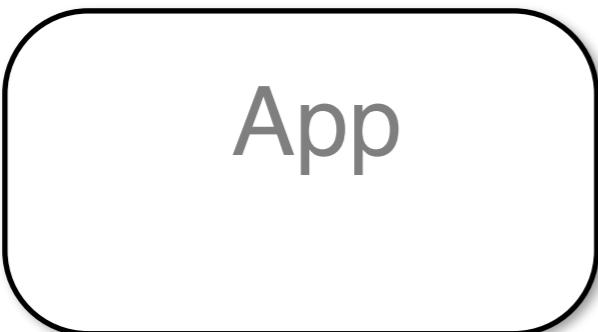
Process

2. Db transaction



2. Db transaction





App

Mod

Mod

Mod



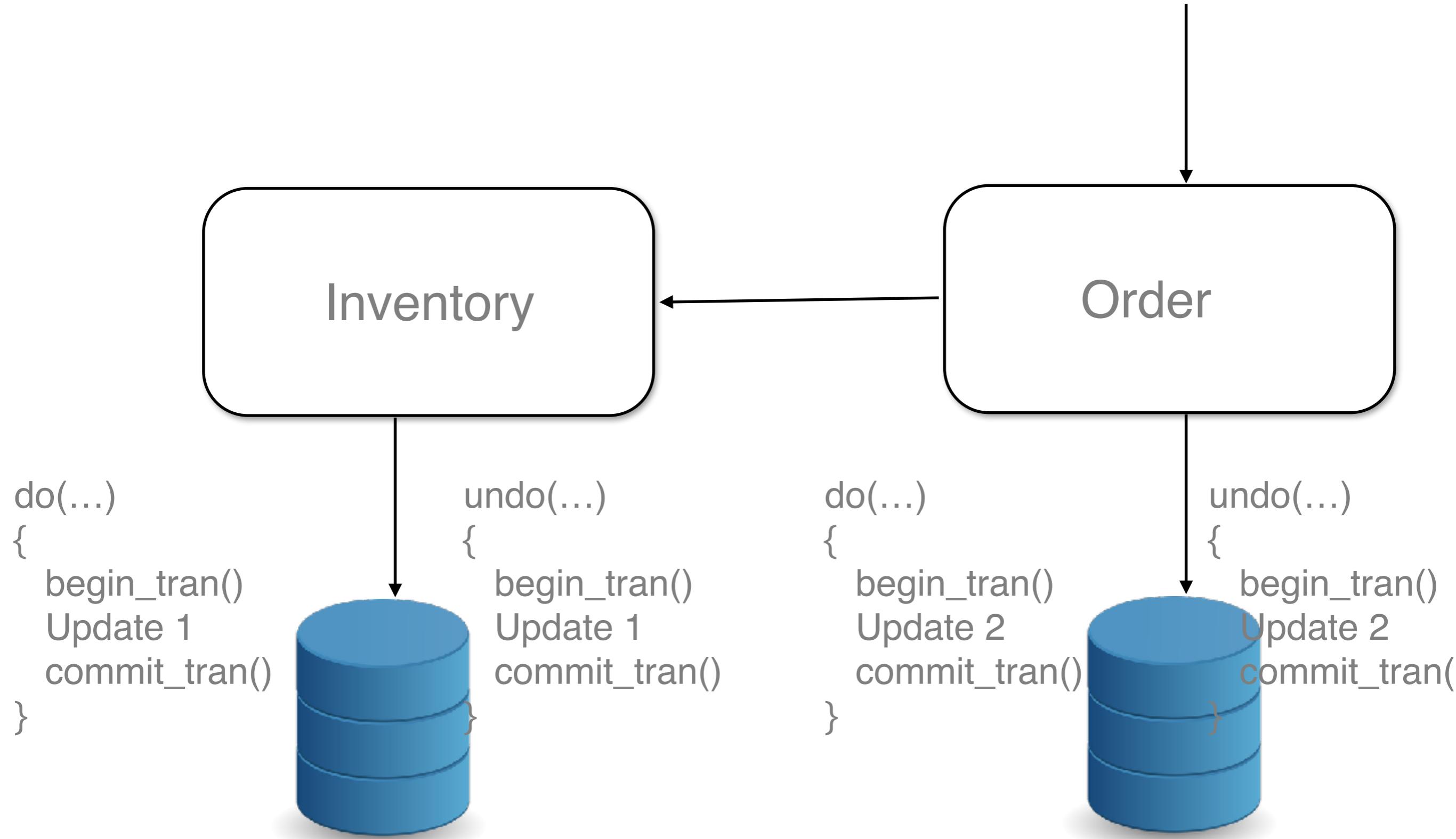
App

App

App

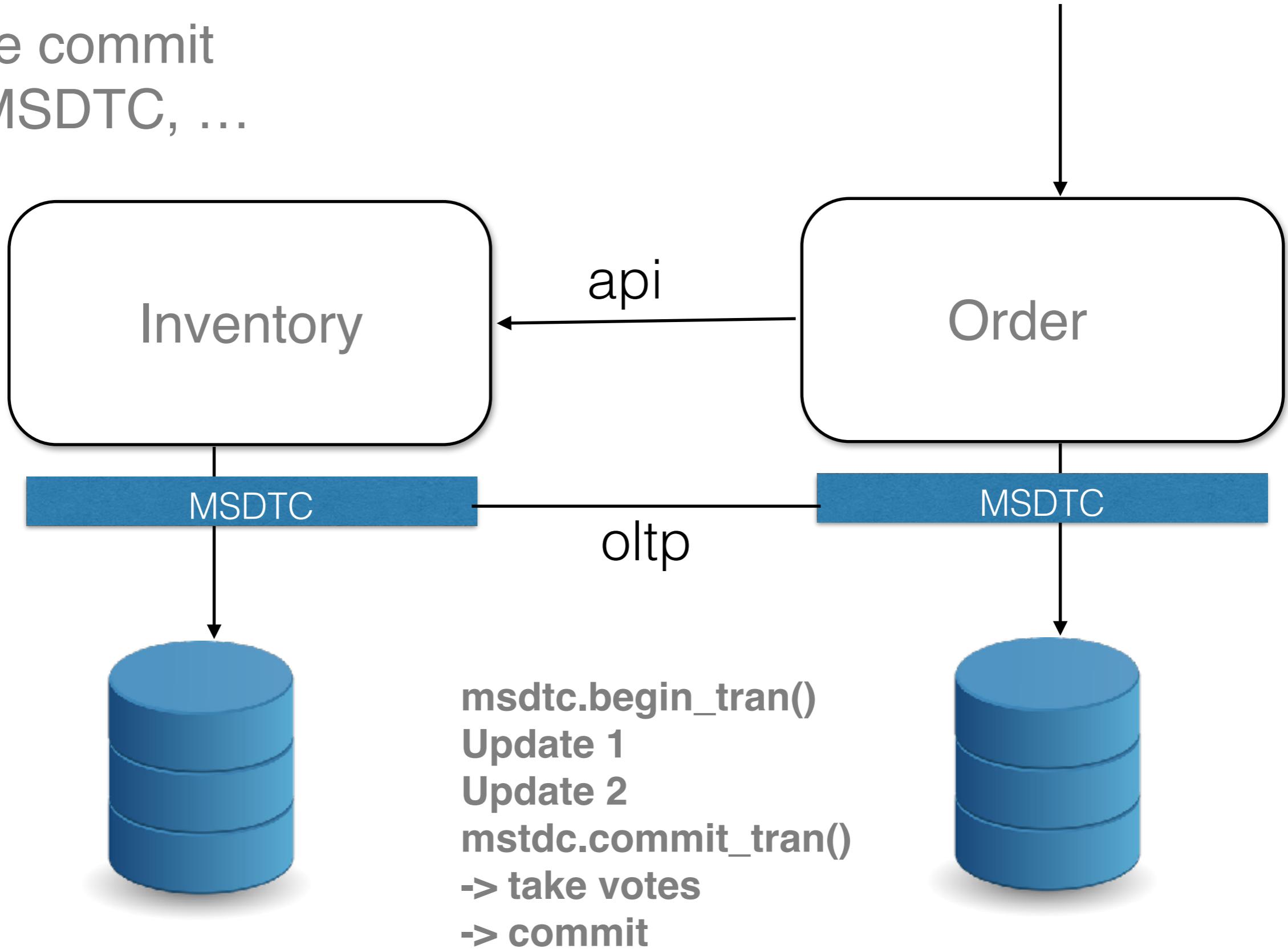


2. Db transaction

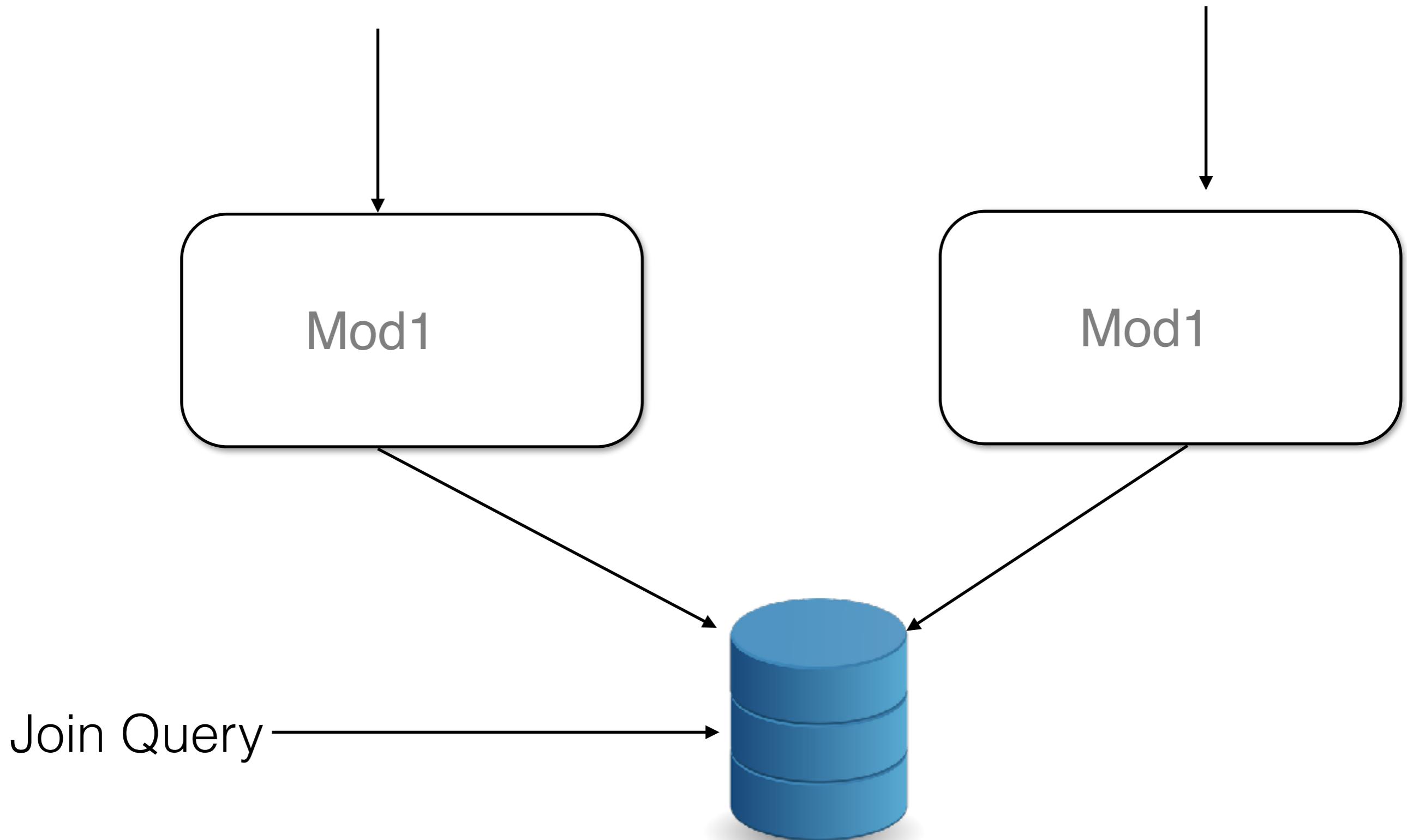


2. Db transaction

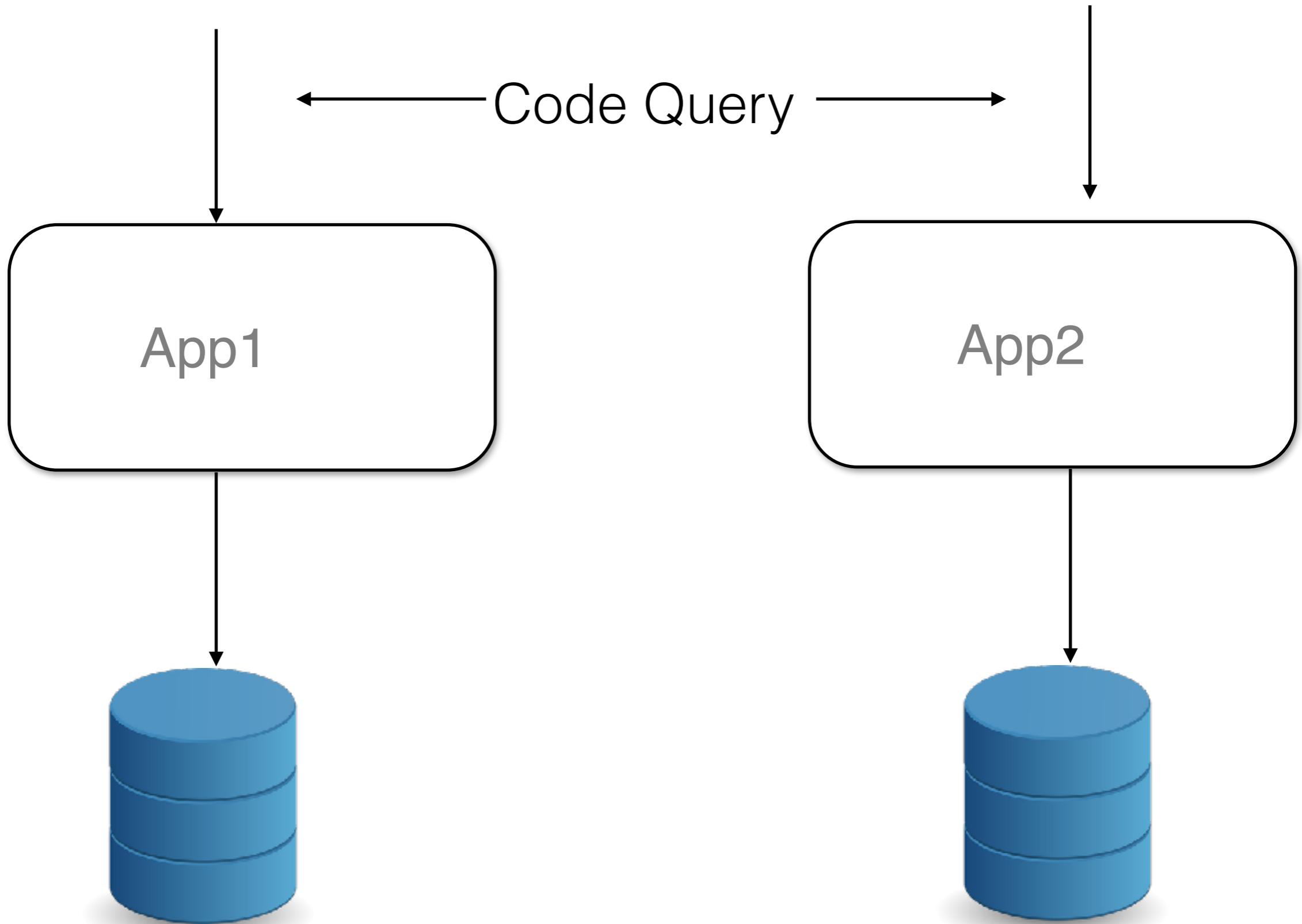
2 phase commit
JTX , MSDTC, ...



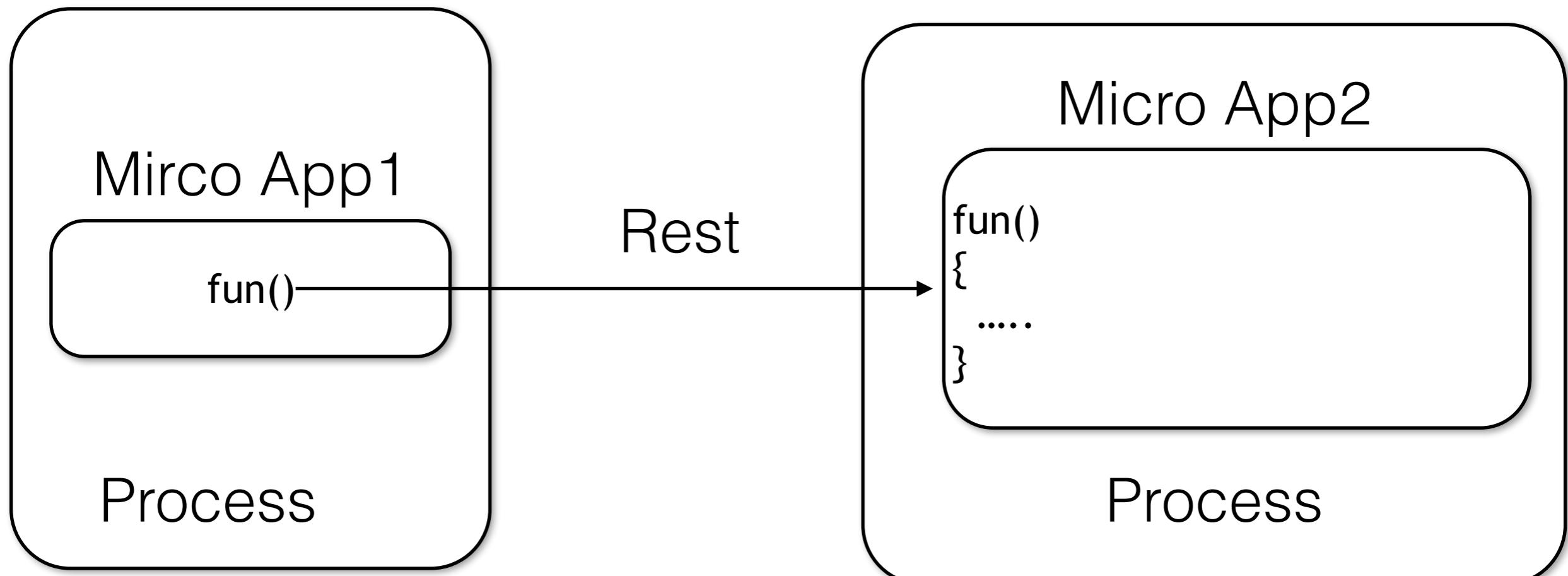
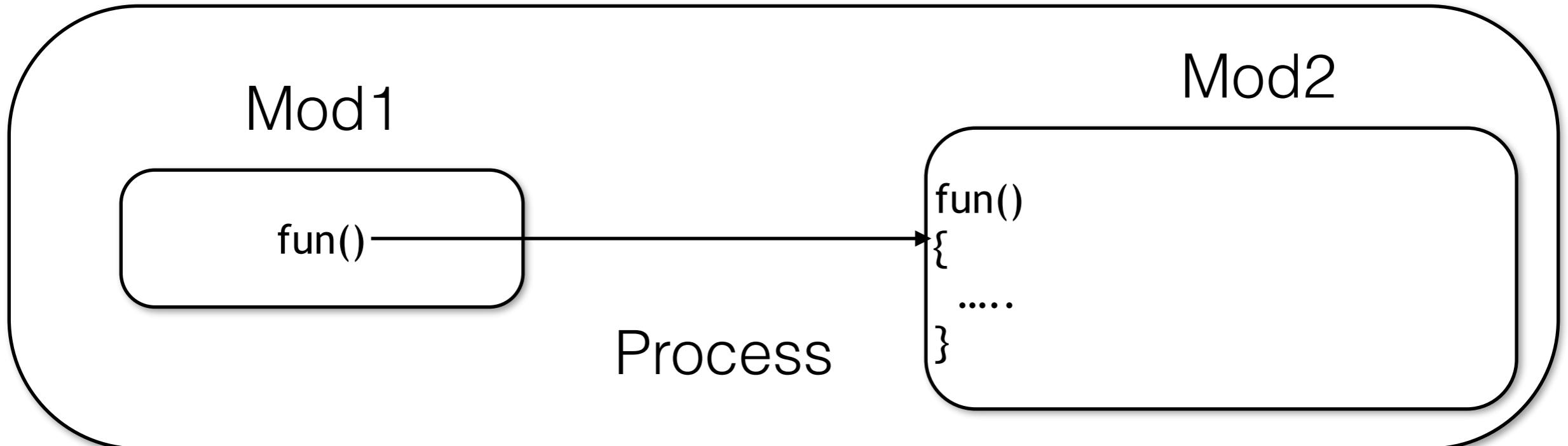
3. Db query



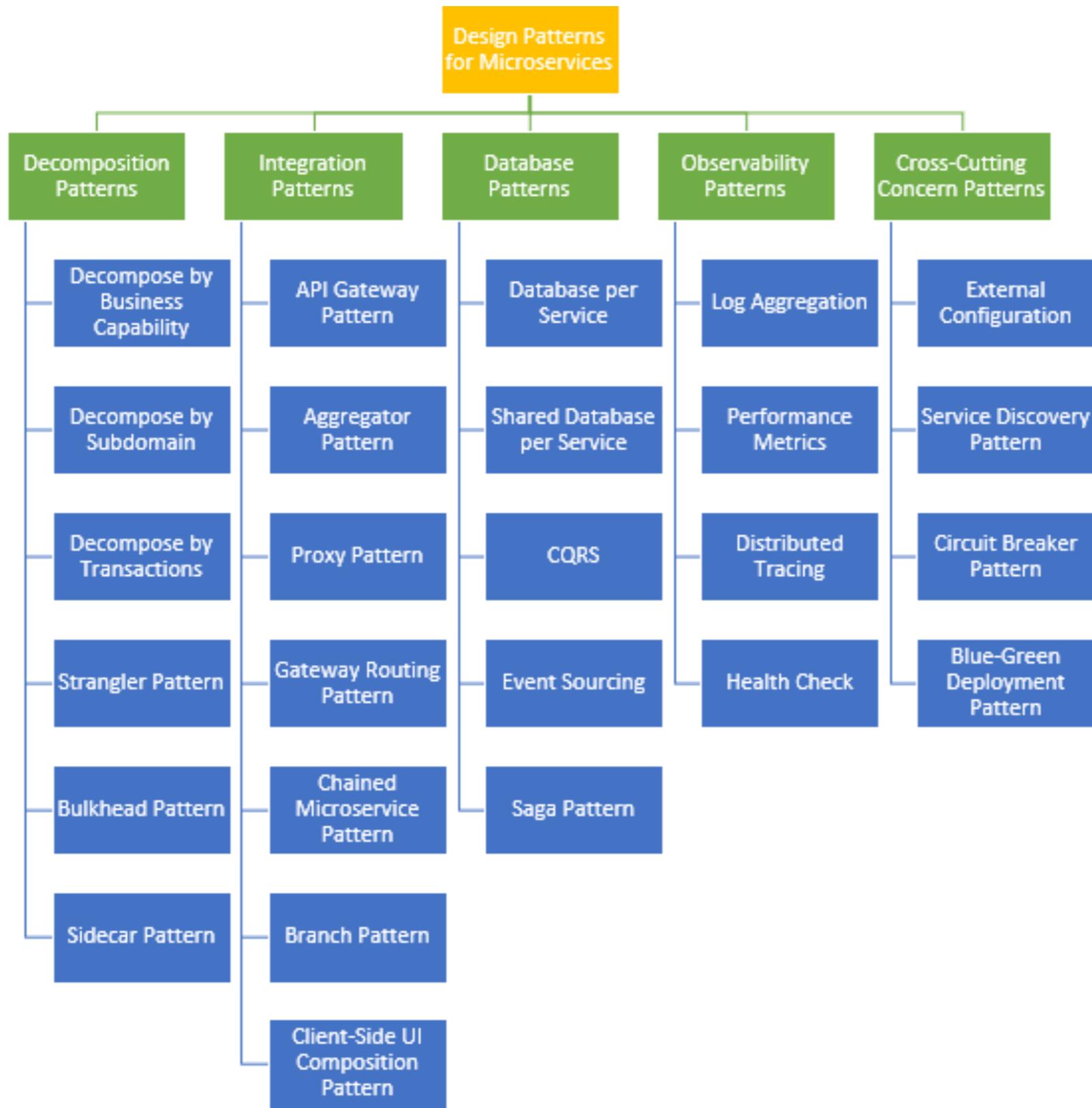
3. Query



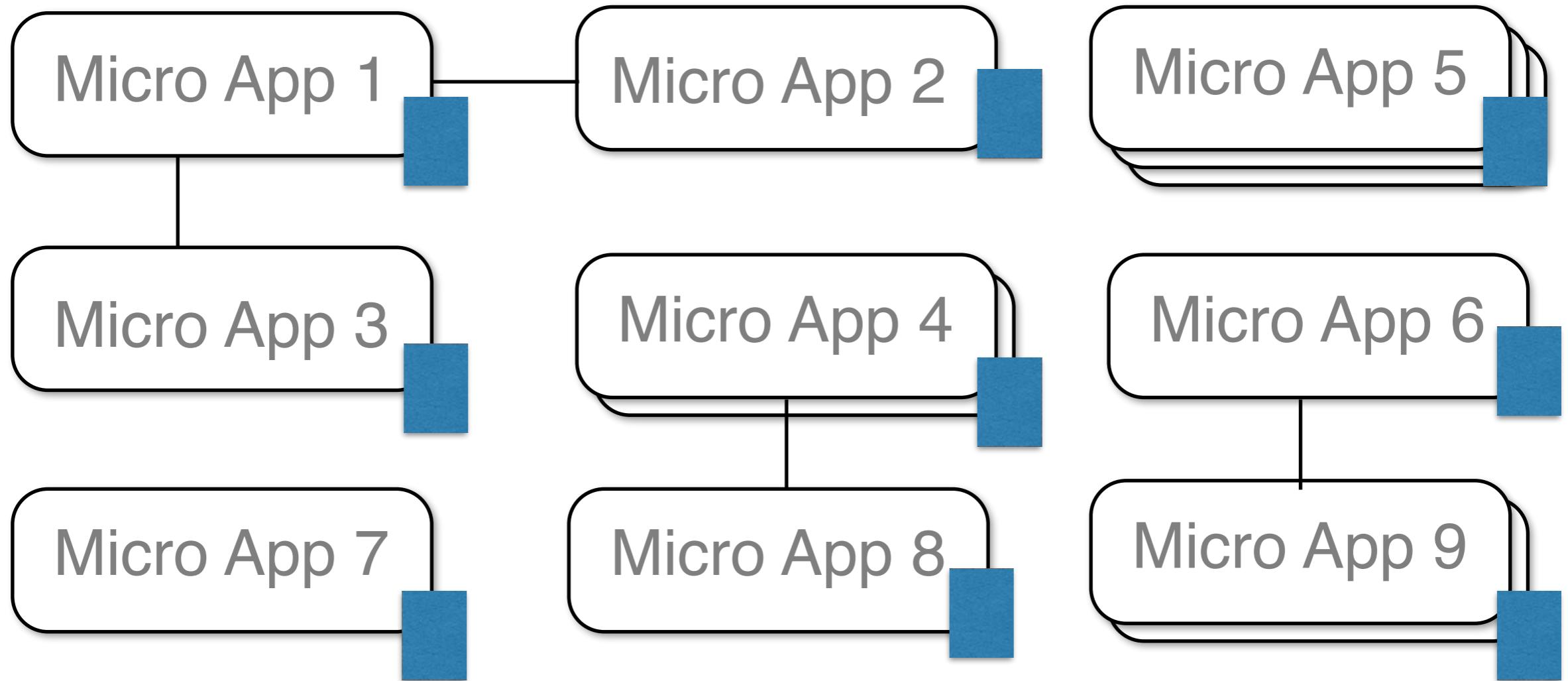
4. Development time



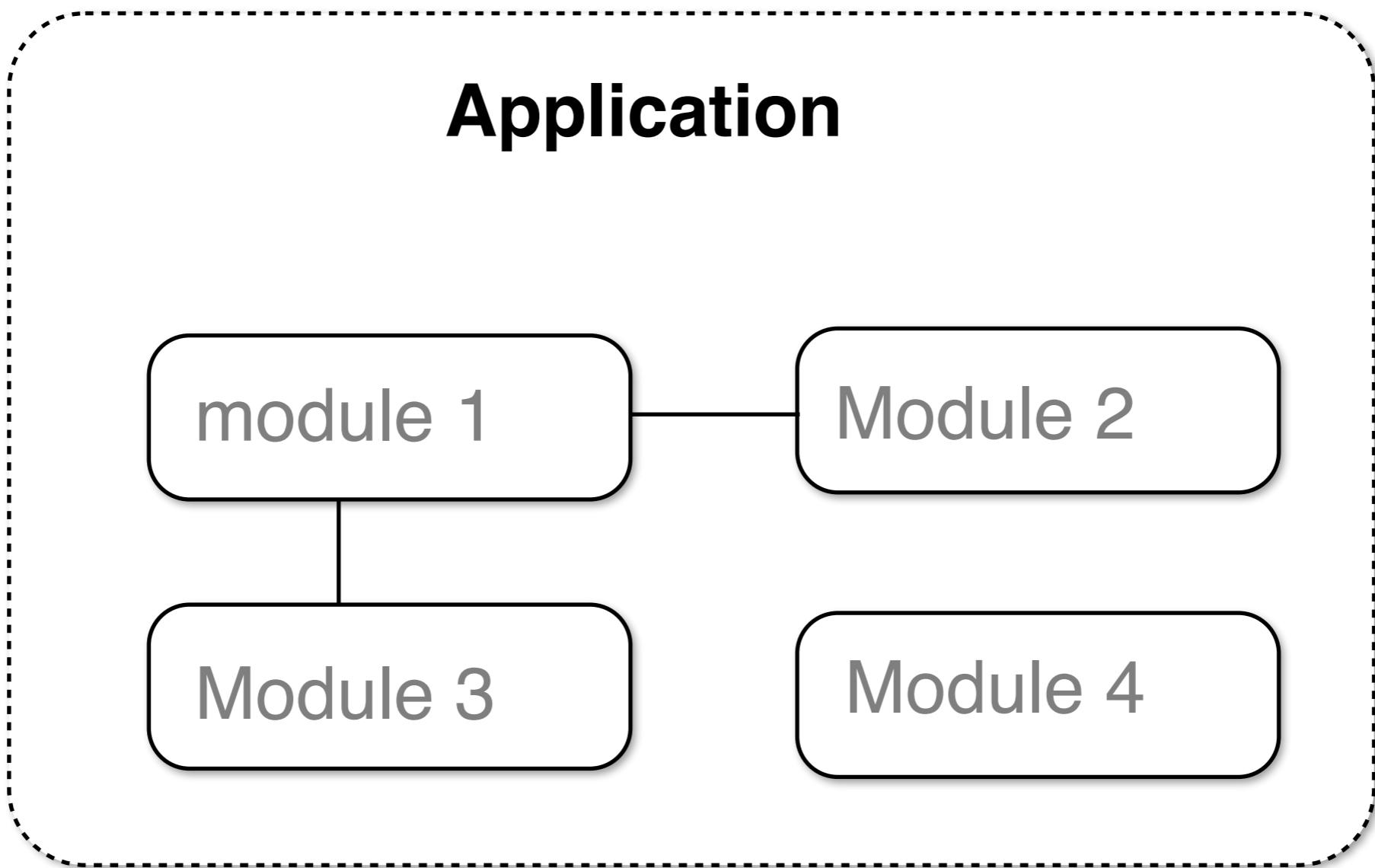
5. Learning Curve



6. Infra Cost



7. Debugging



Dev & Ops Teams



Log Data

Web Logs
App Logs
Database Logs
Container Logs

Metrics Data

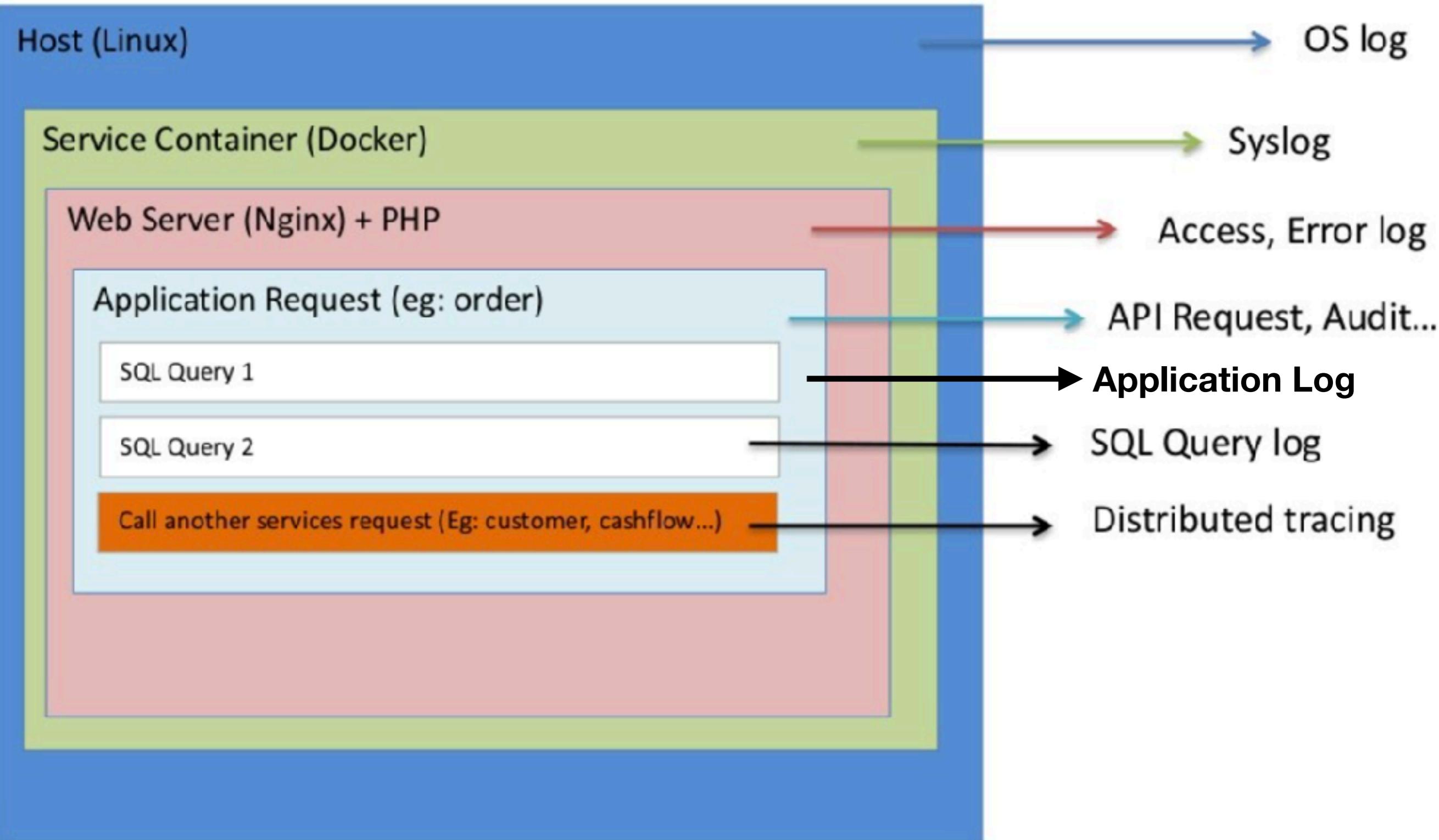
Container Metrics
Host Metrics
Database Metrics
Network Metrics
Storage Metrics

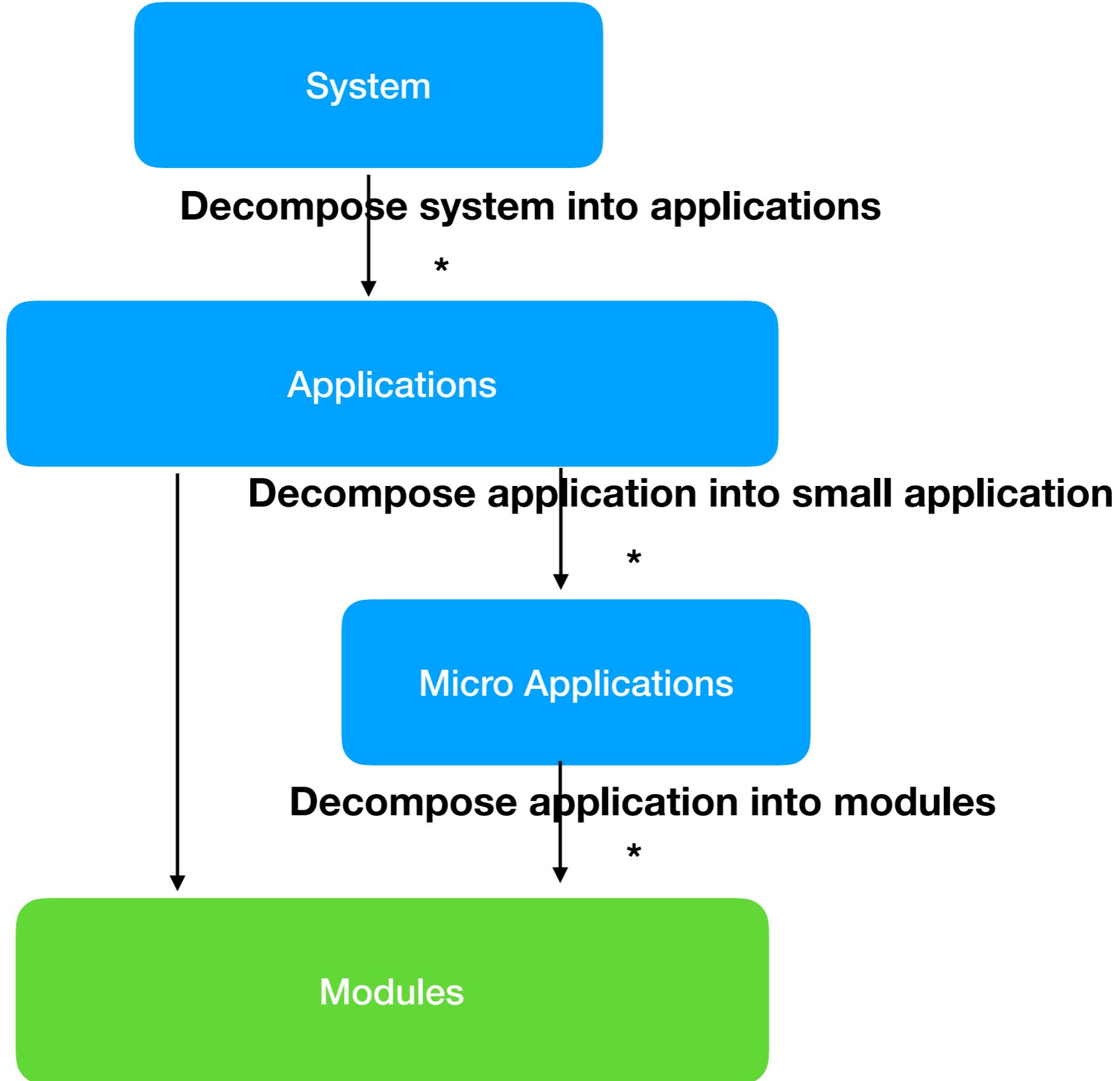
APM Data

Real User Monitoring
Txn Perf Monitoring
Distributed Tracing

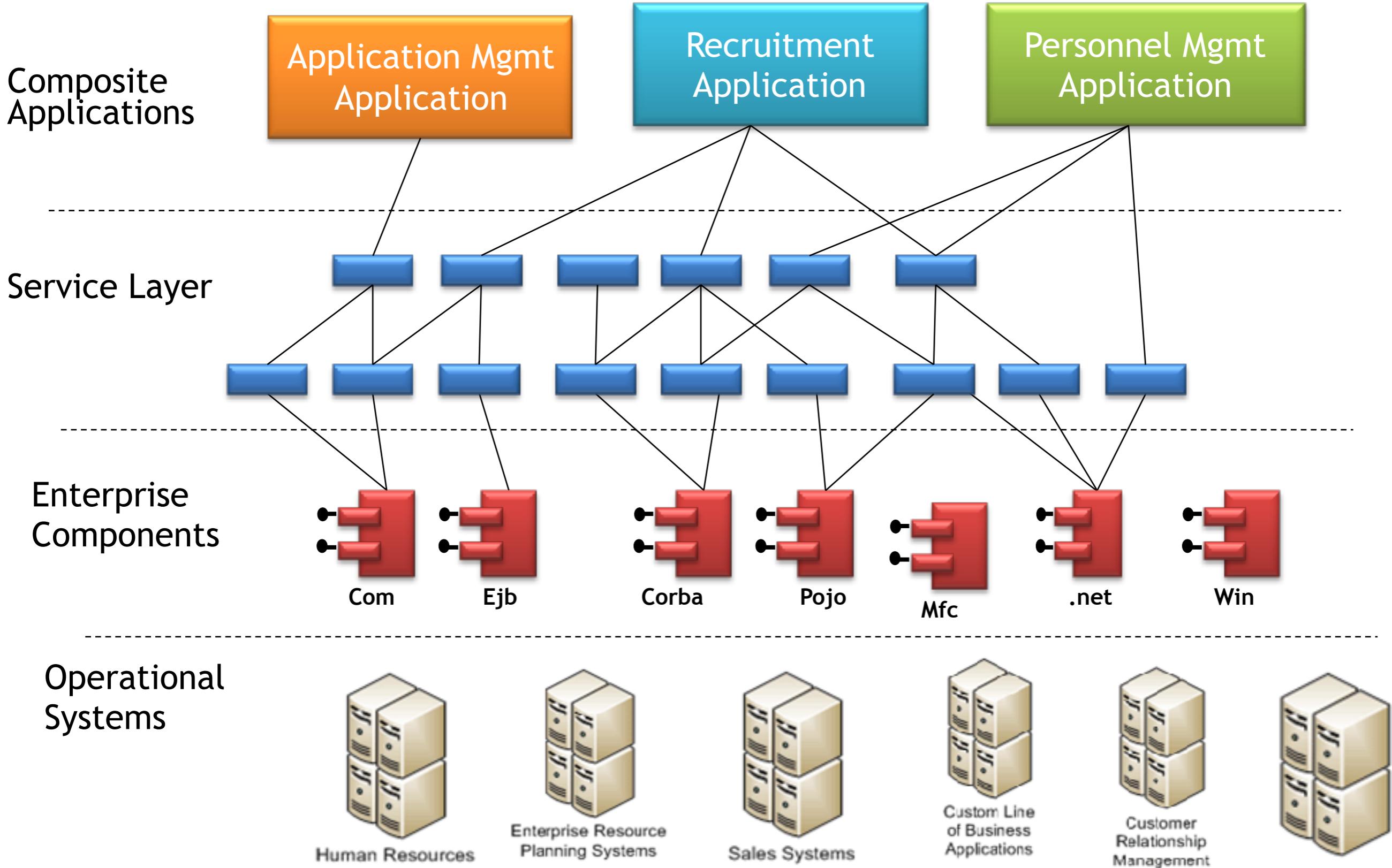
Uptime Data

Uptime
Response Time



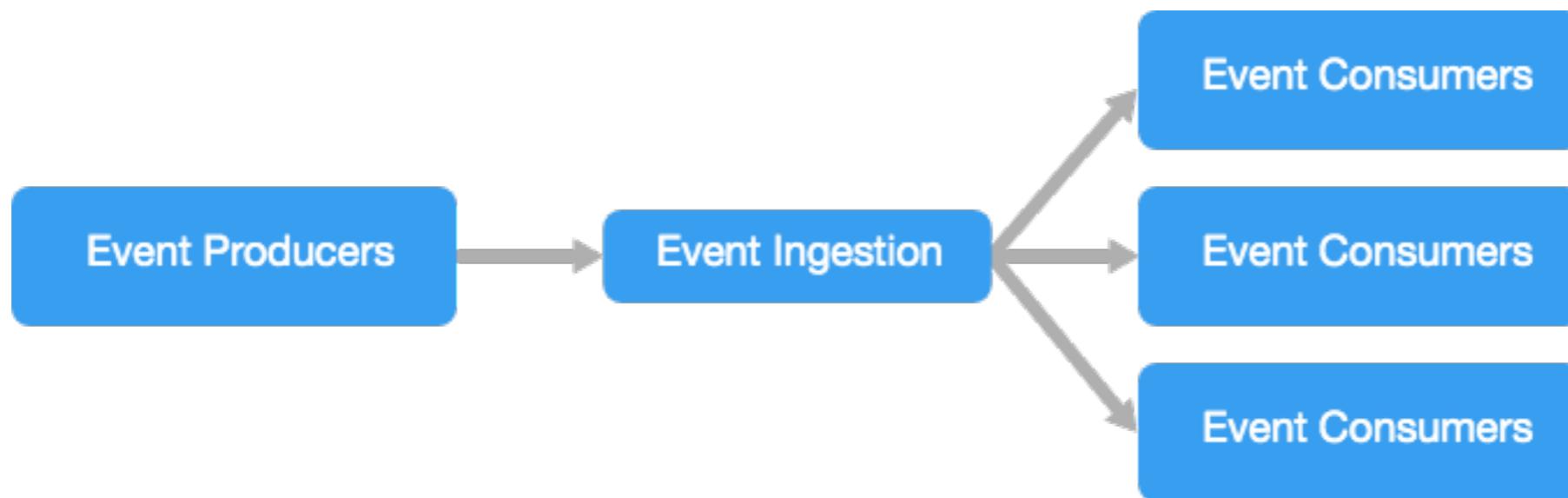


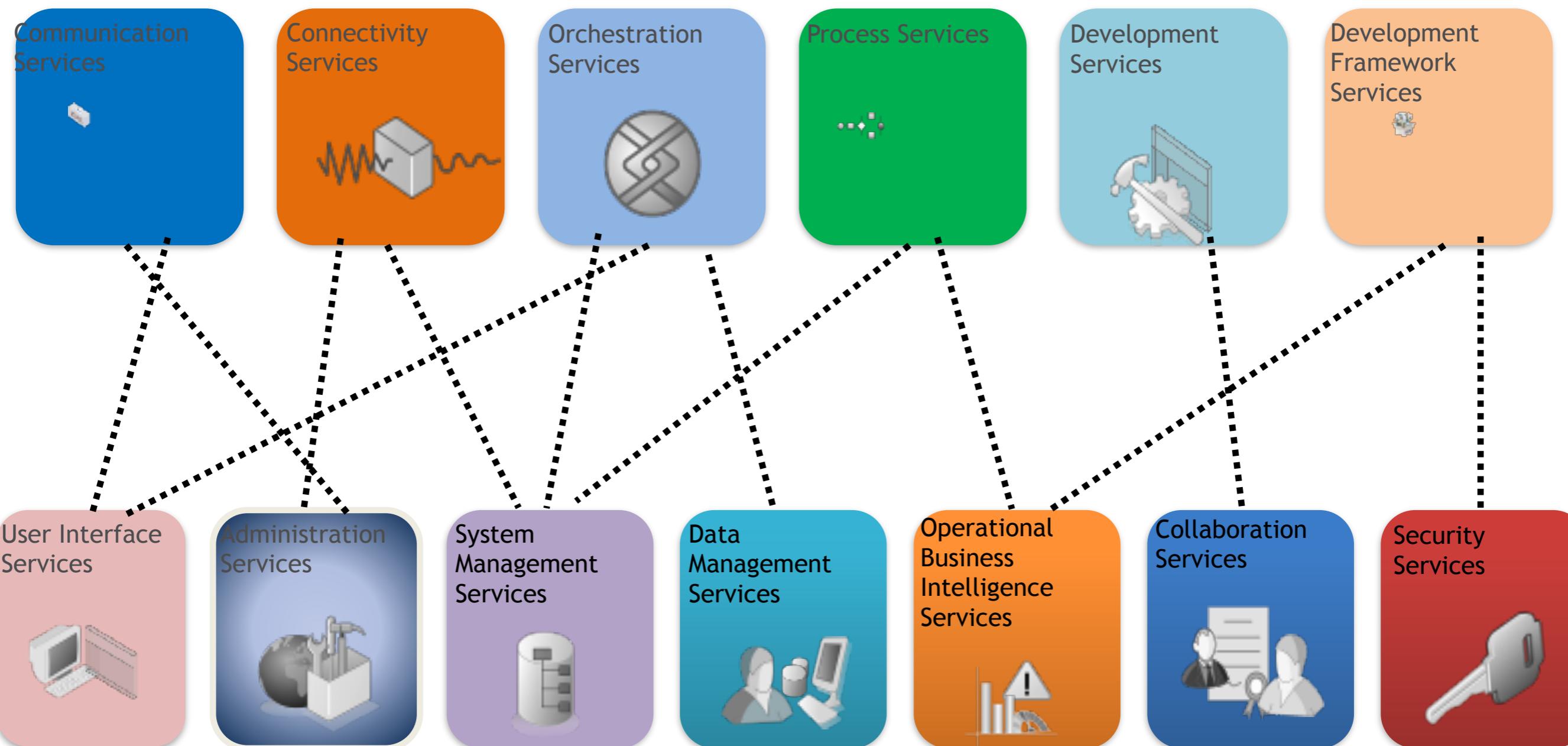
Service Oriented Architecture

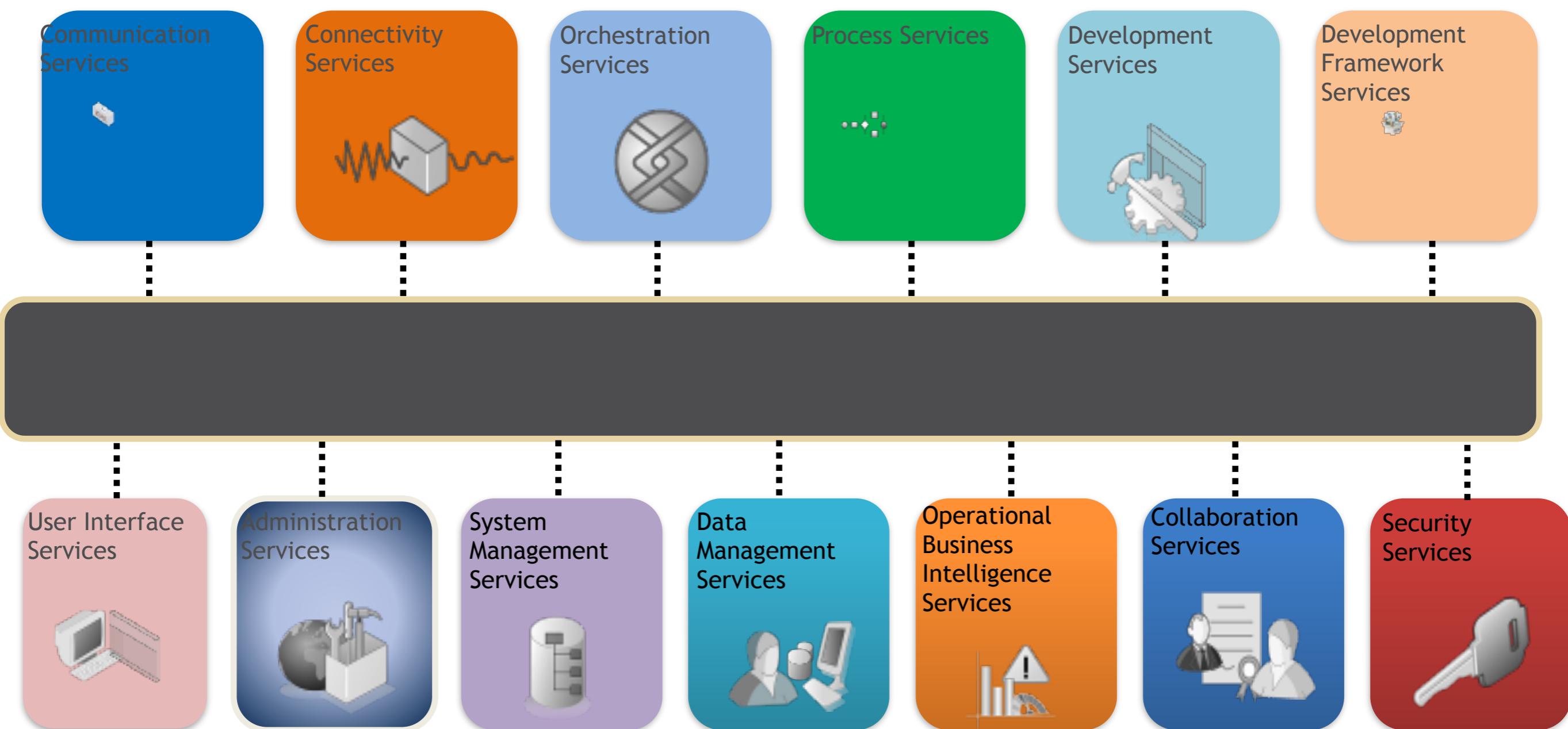


- Almost all the successful microservice stories started with a monolith that got too big and was broken up.
- Almost all the cases where I've heard of a system that was built as a microservice system from scratch have ended up in serious trouble.

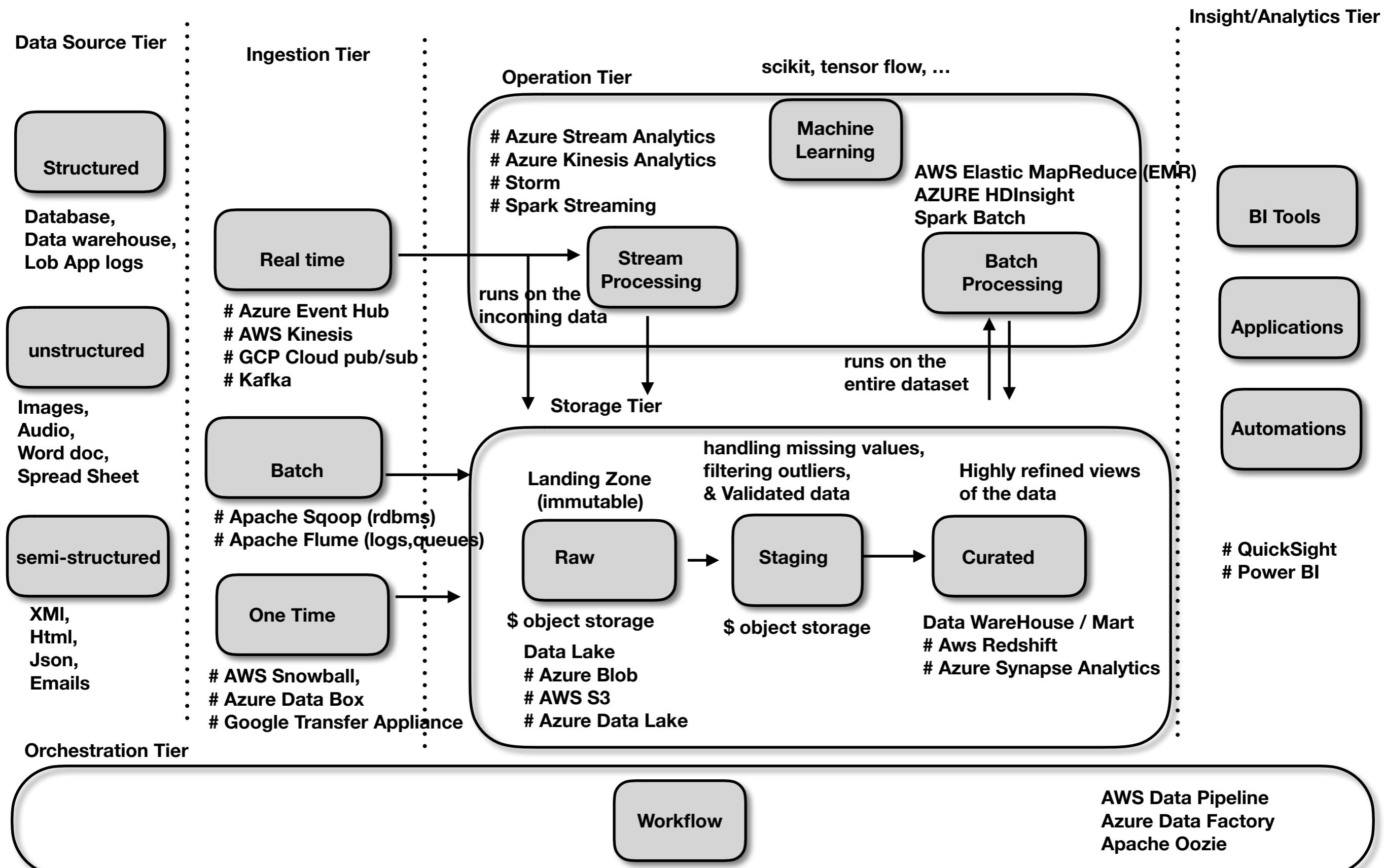
Event Driven Architecture



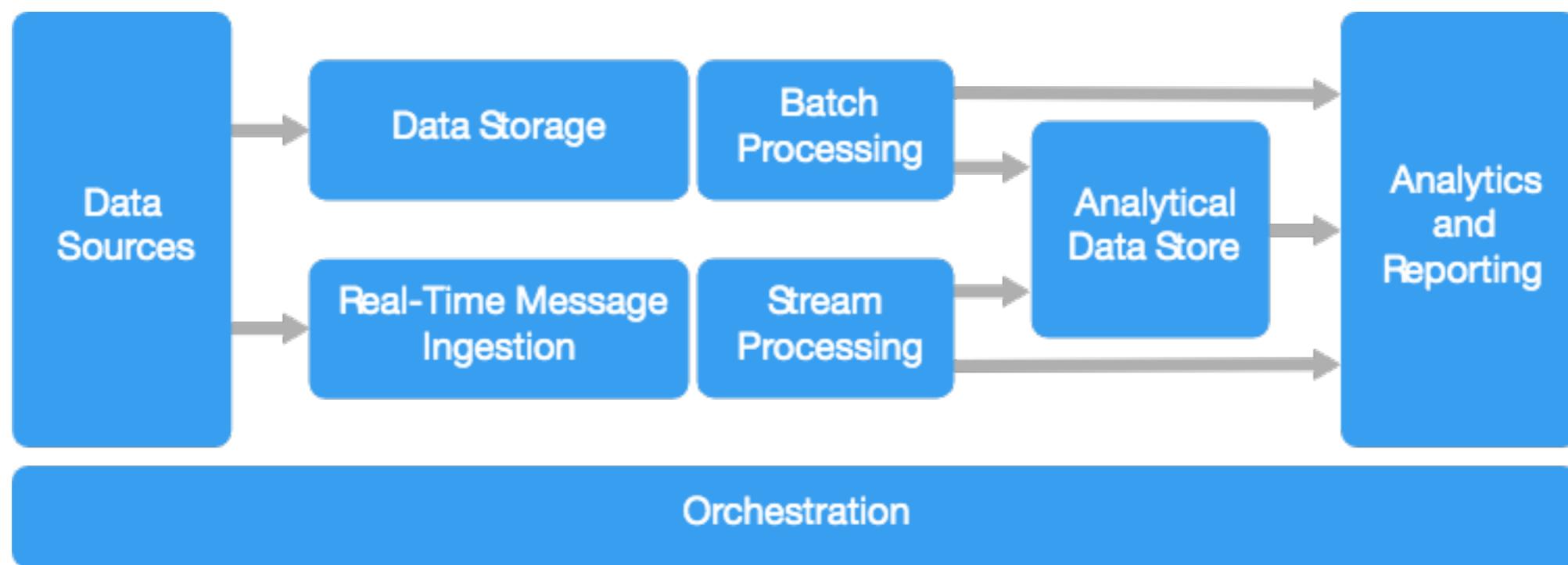




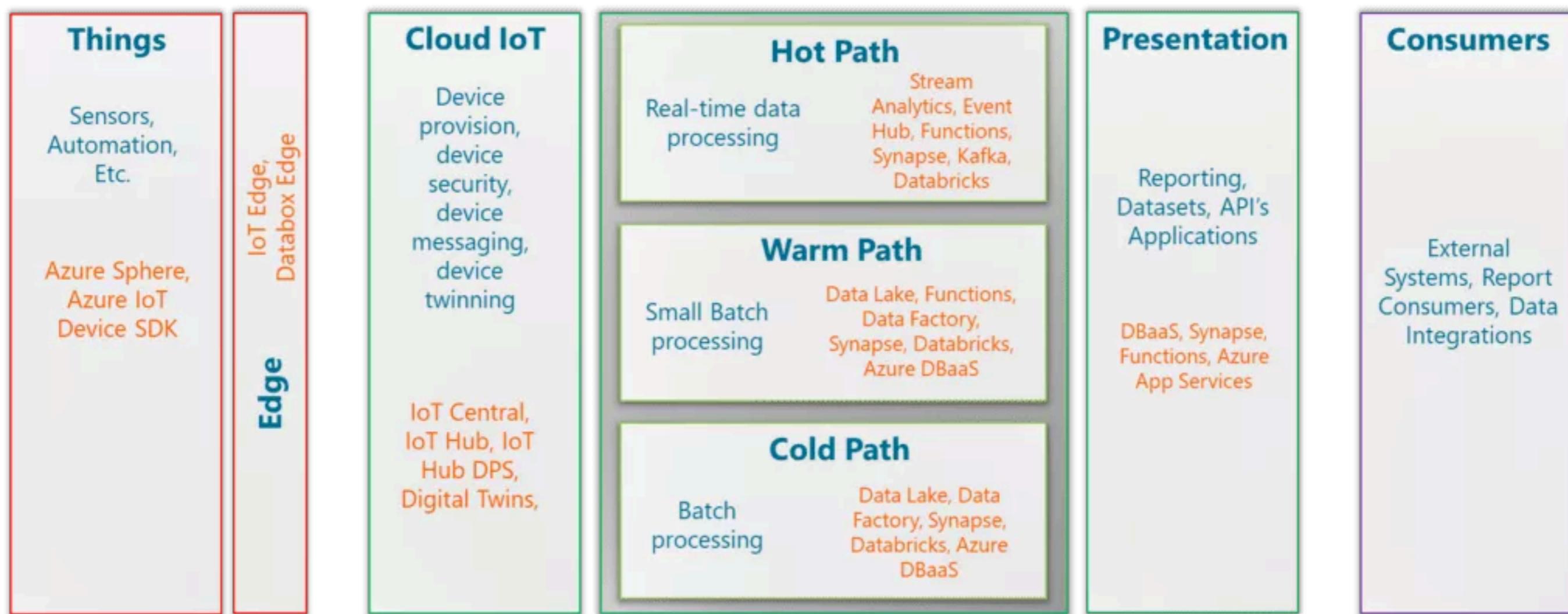
BigData Reference Architecture

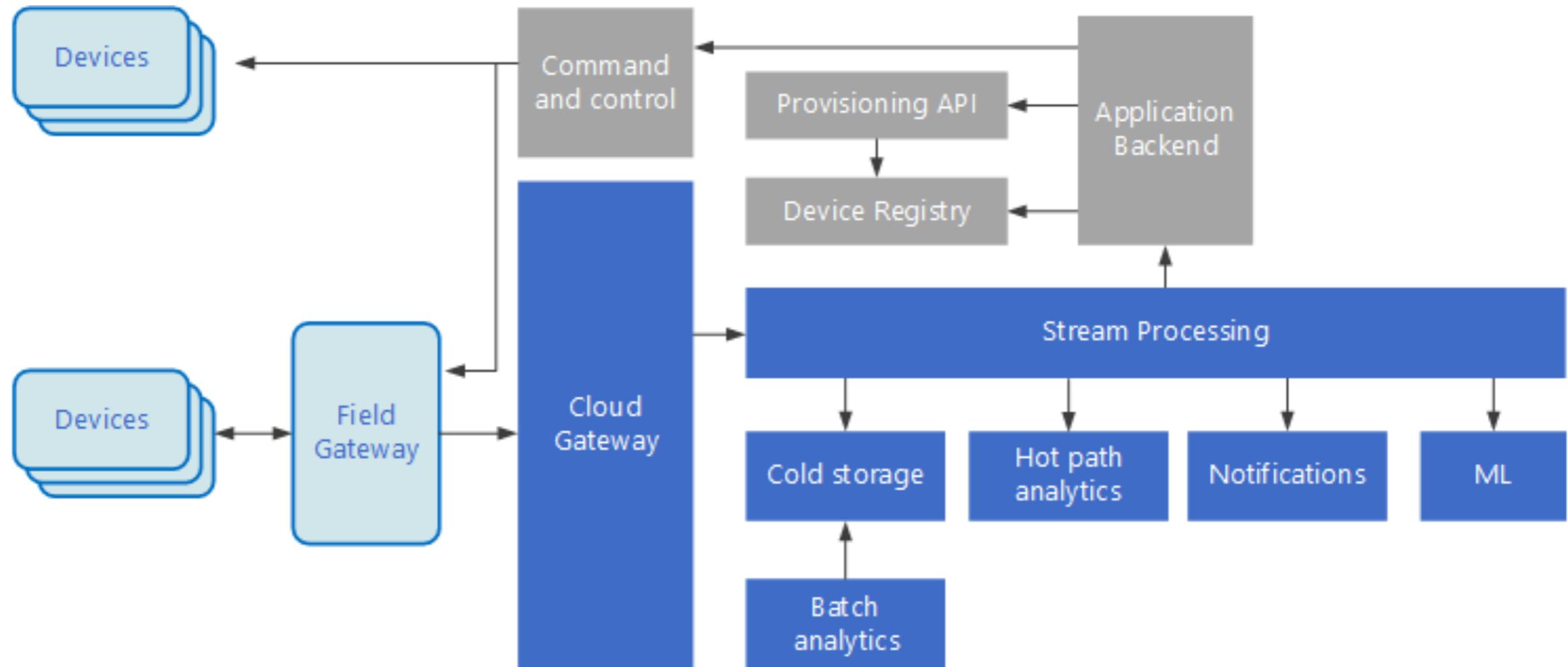


Big data architecture style

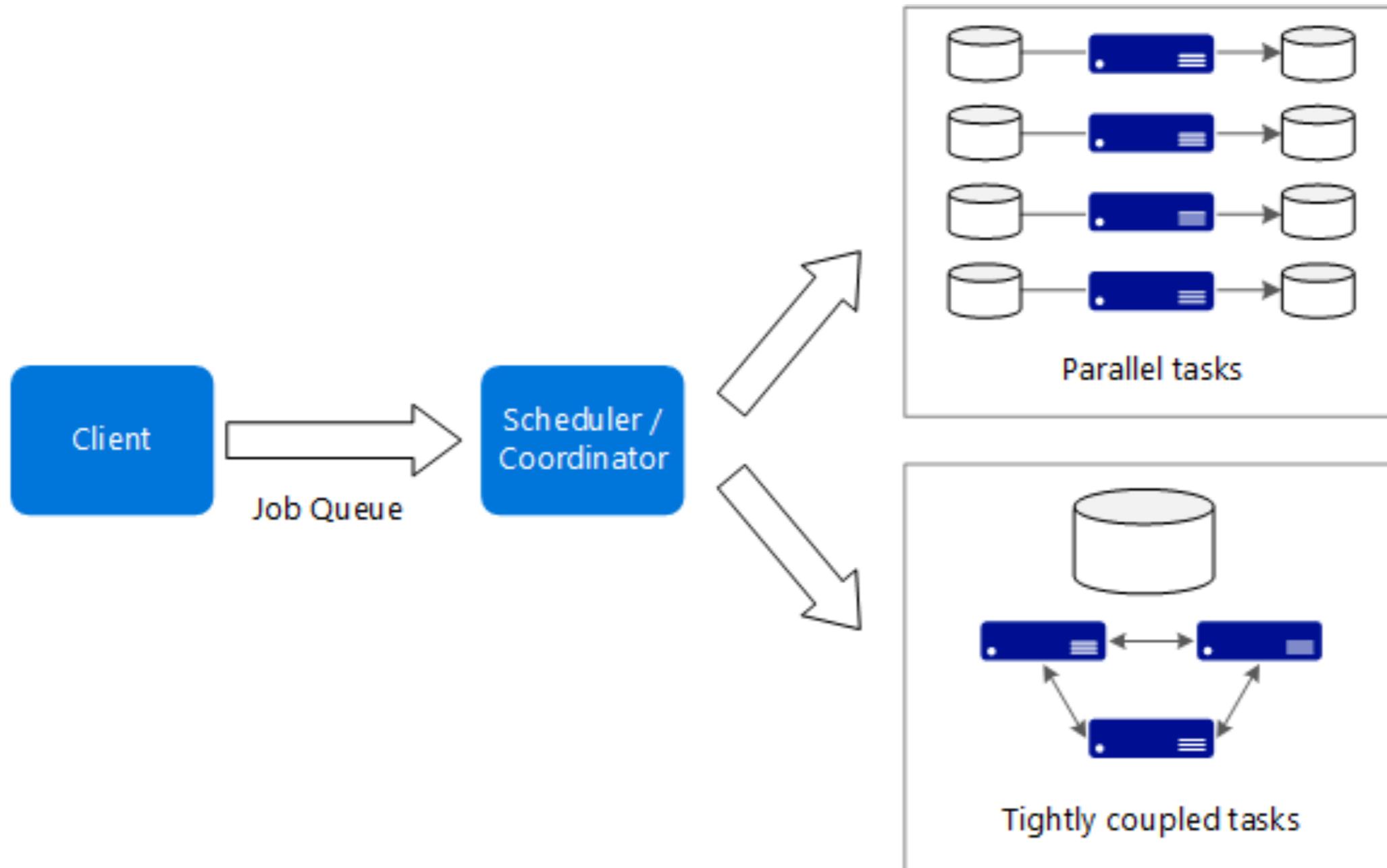


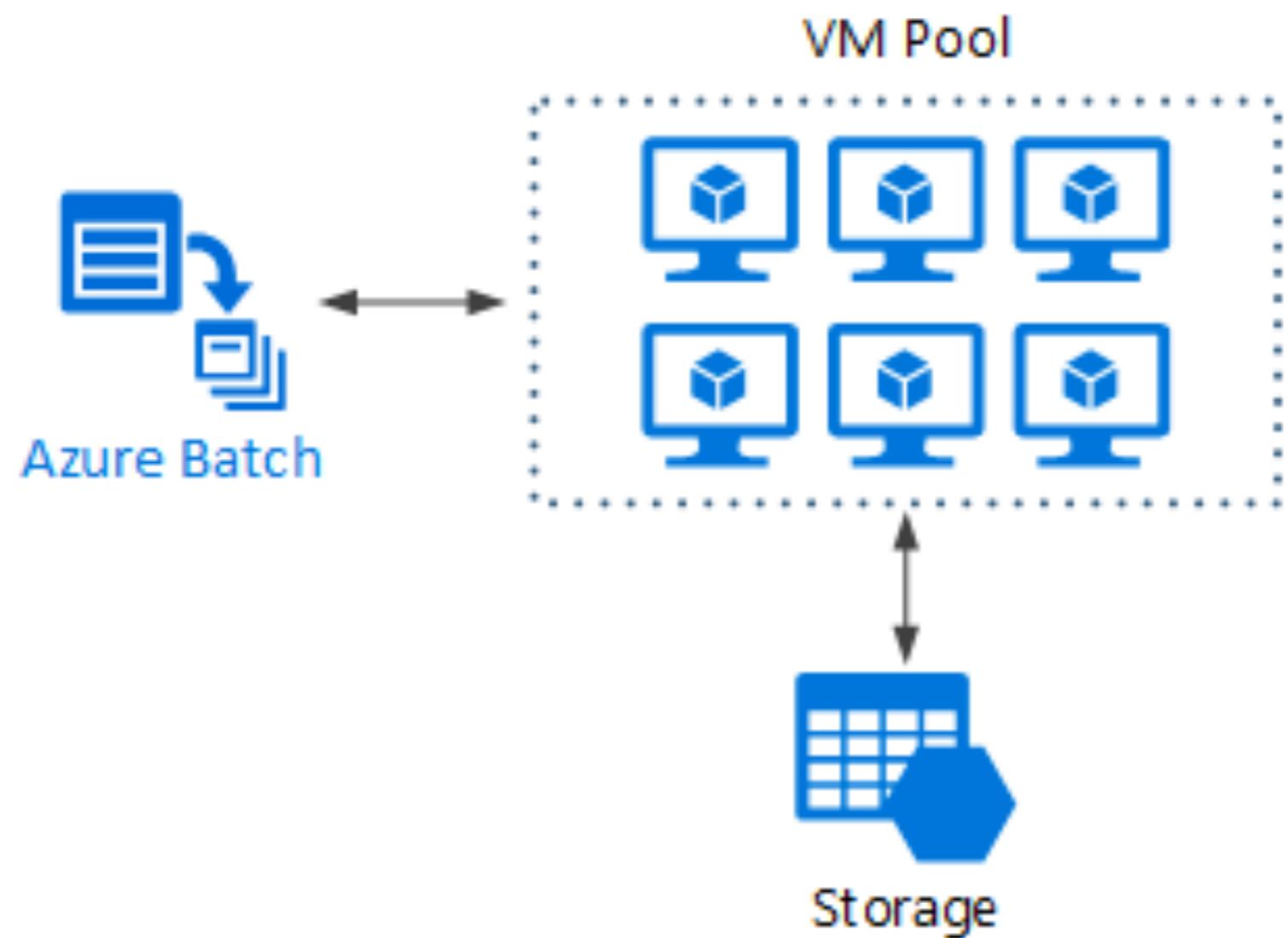
IOT Architecture

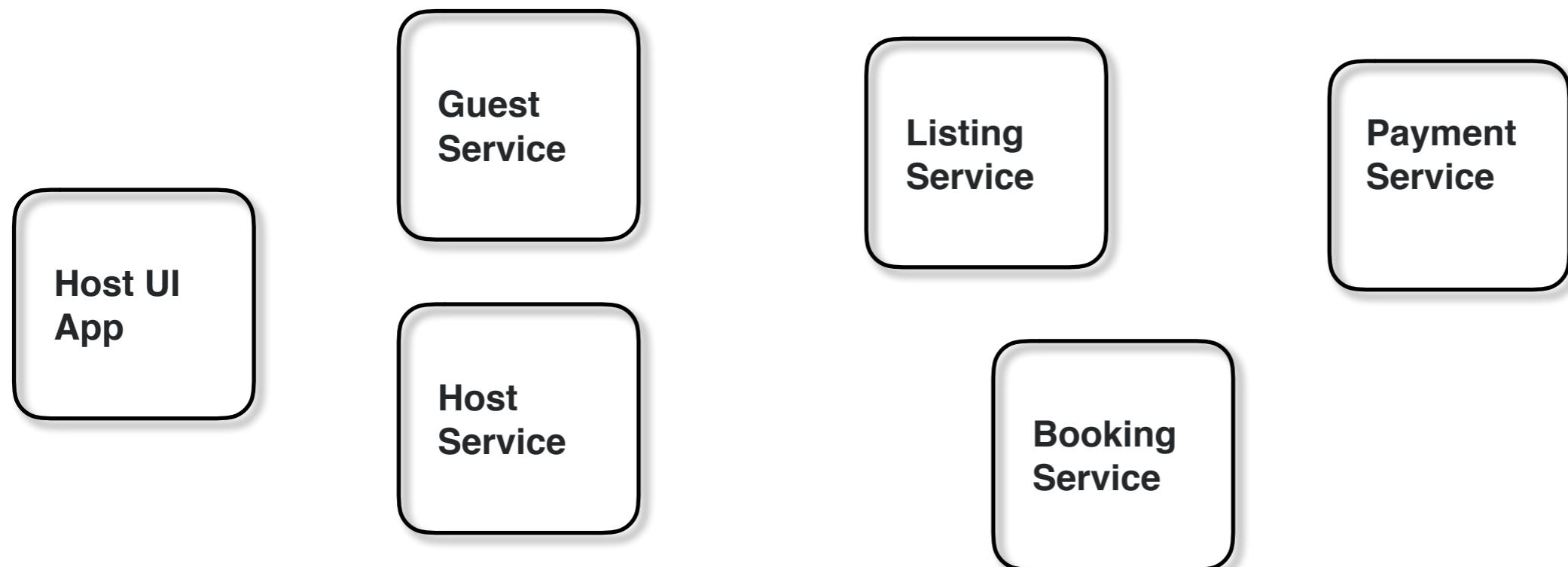


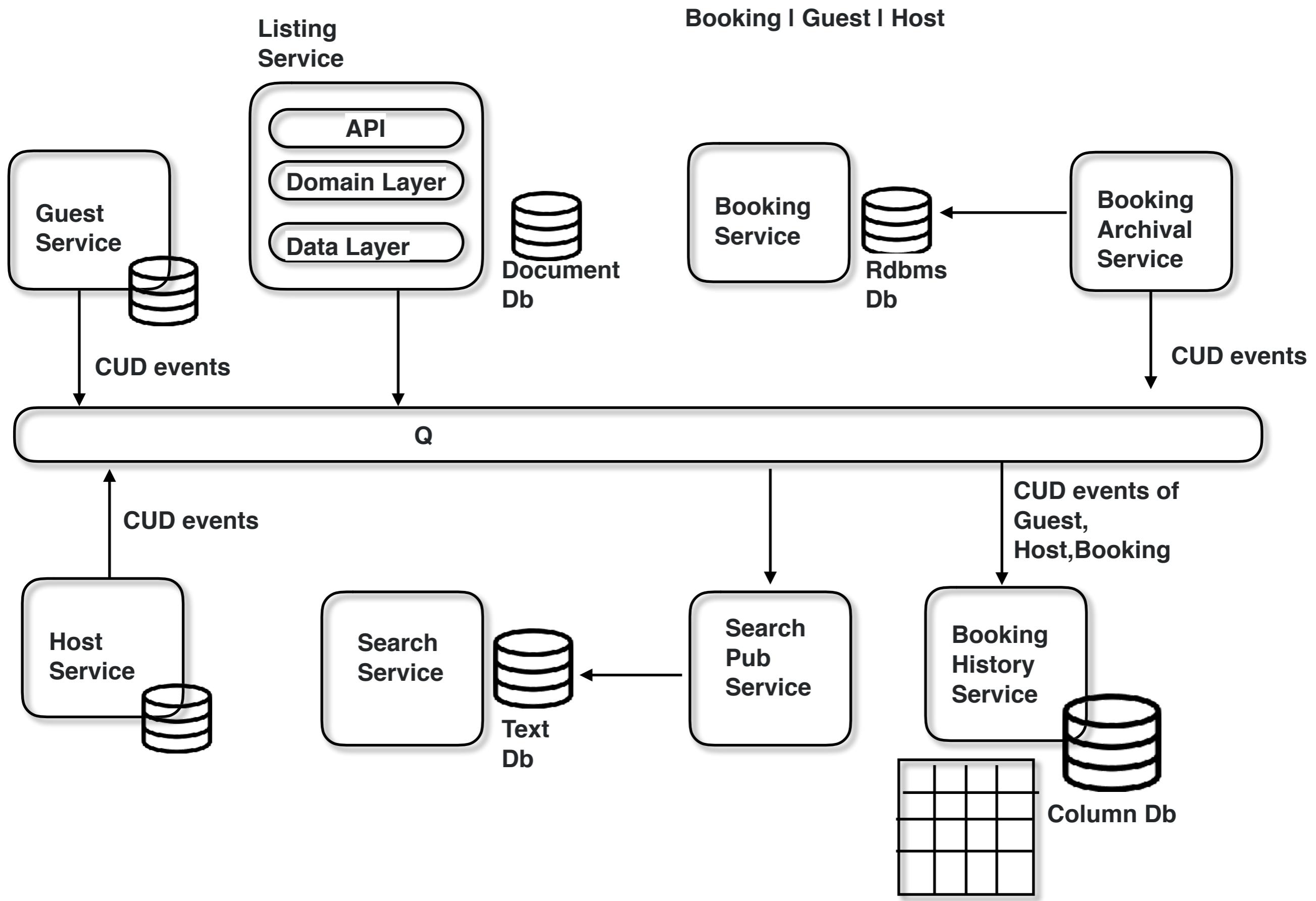


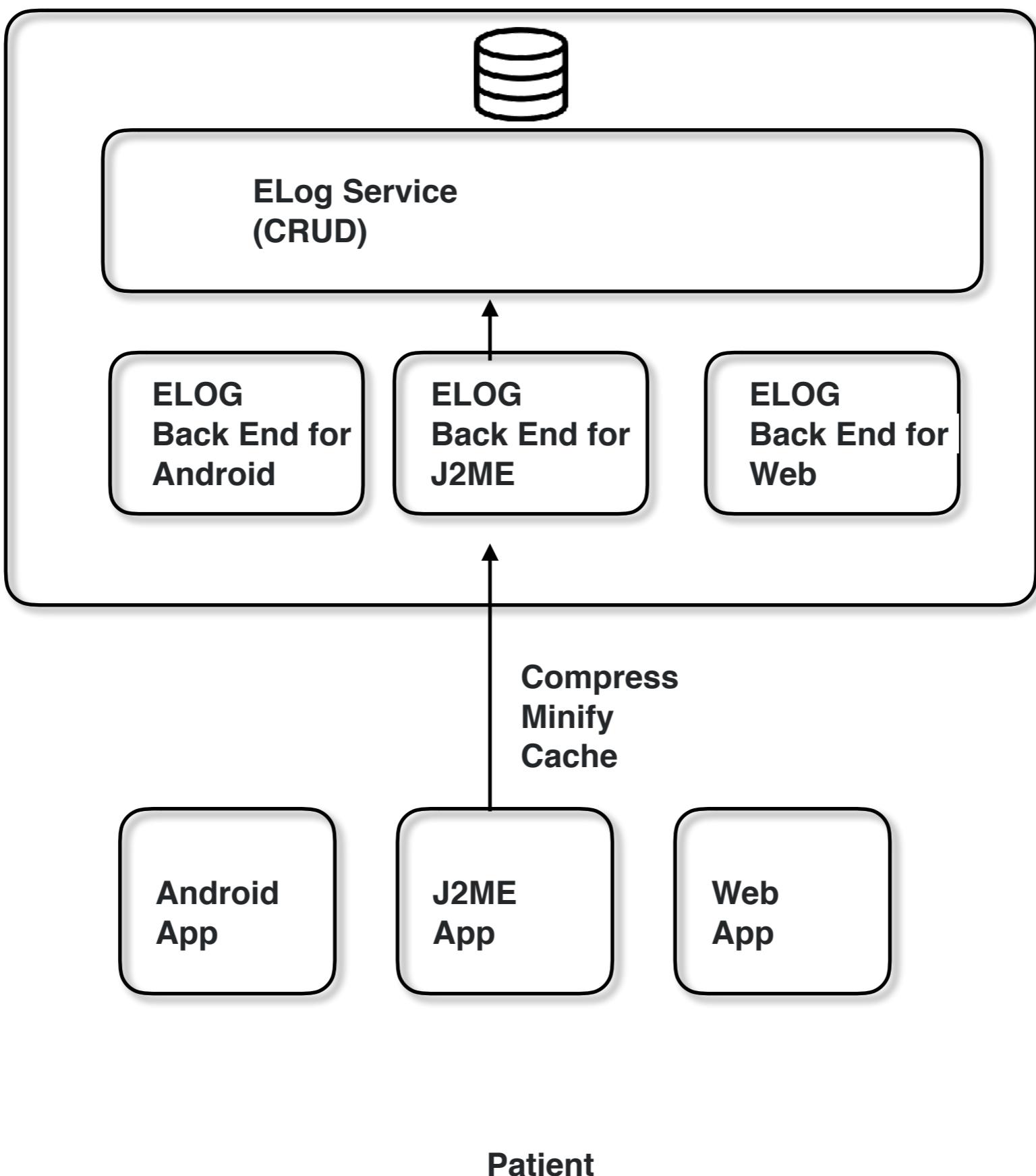
Big compute architecture style





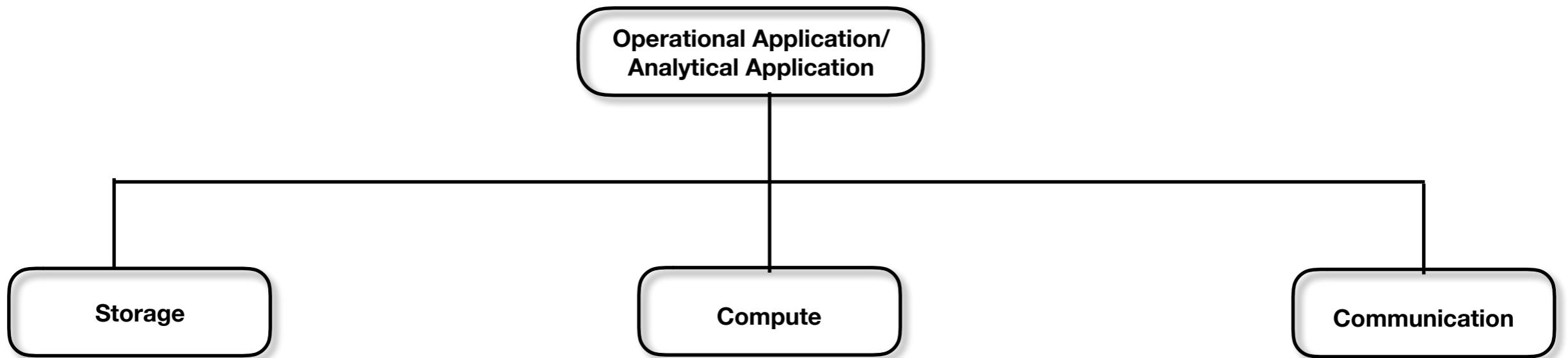


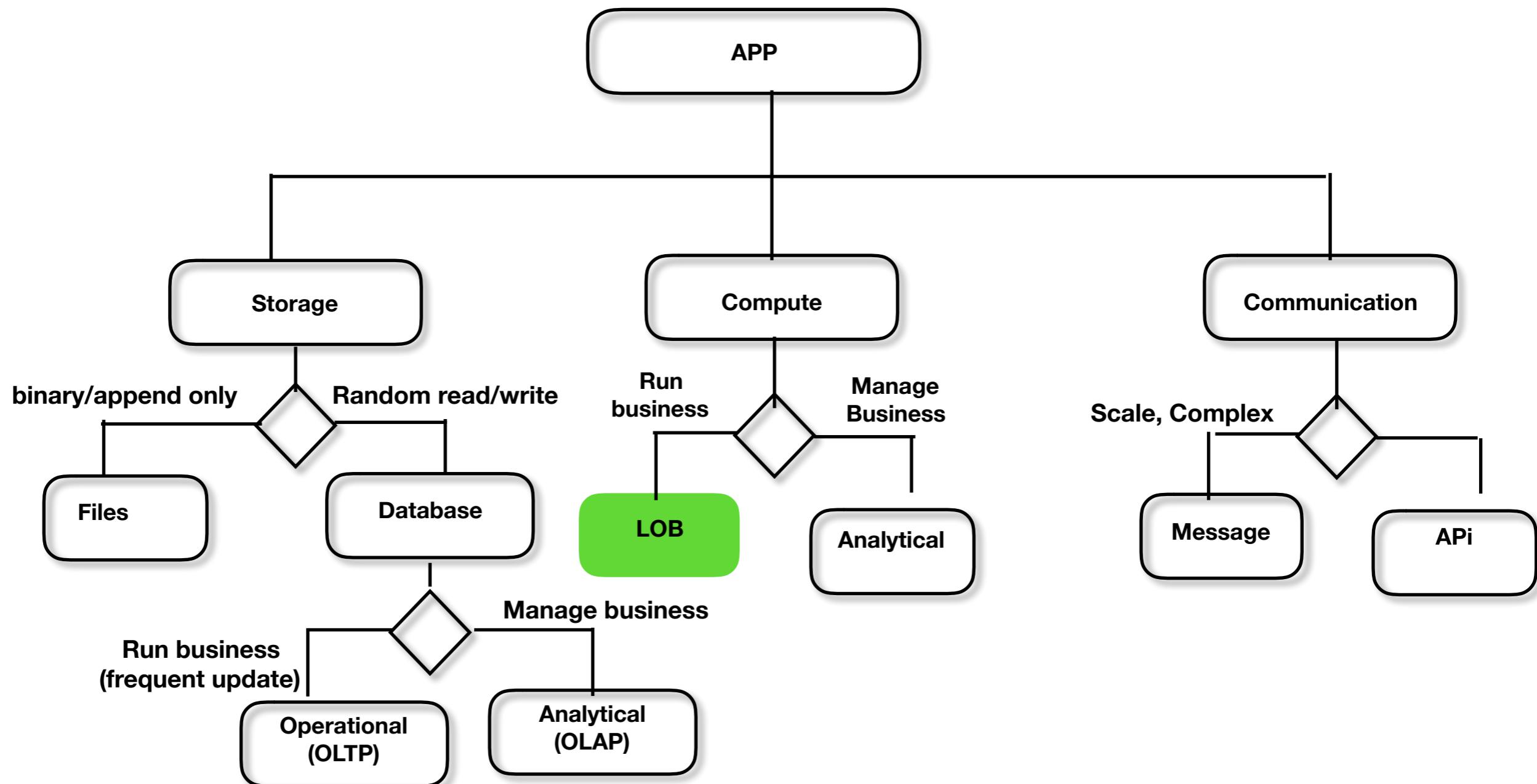




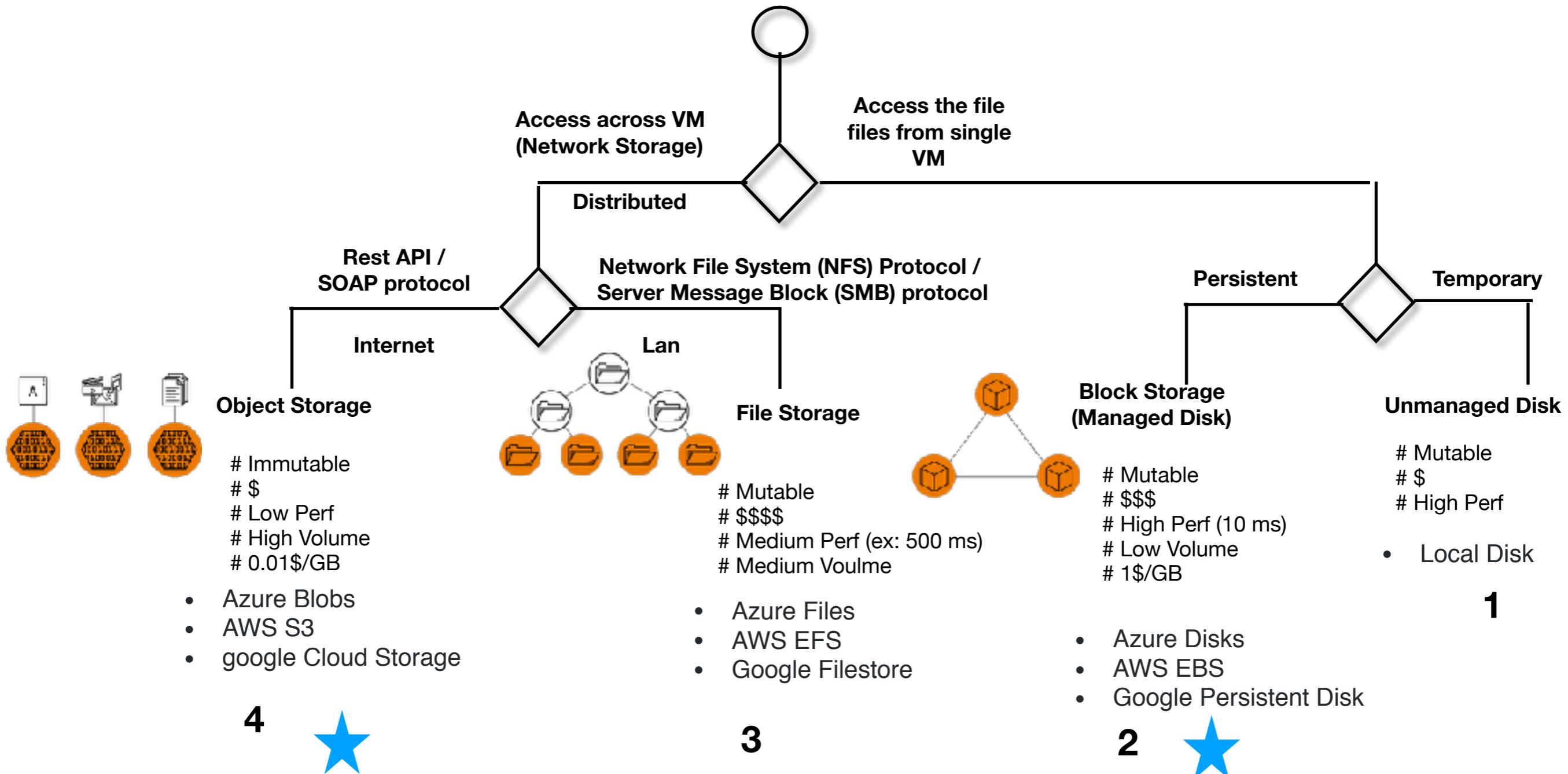
Patient

Choosing Technology ?



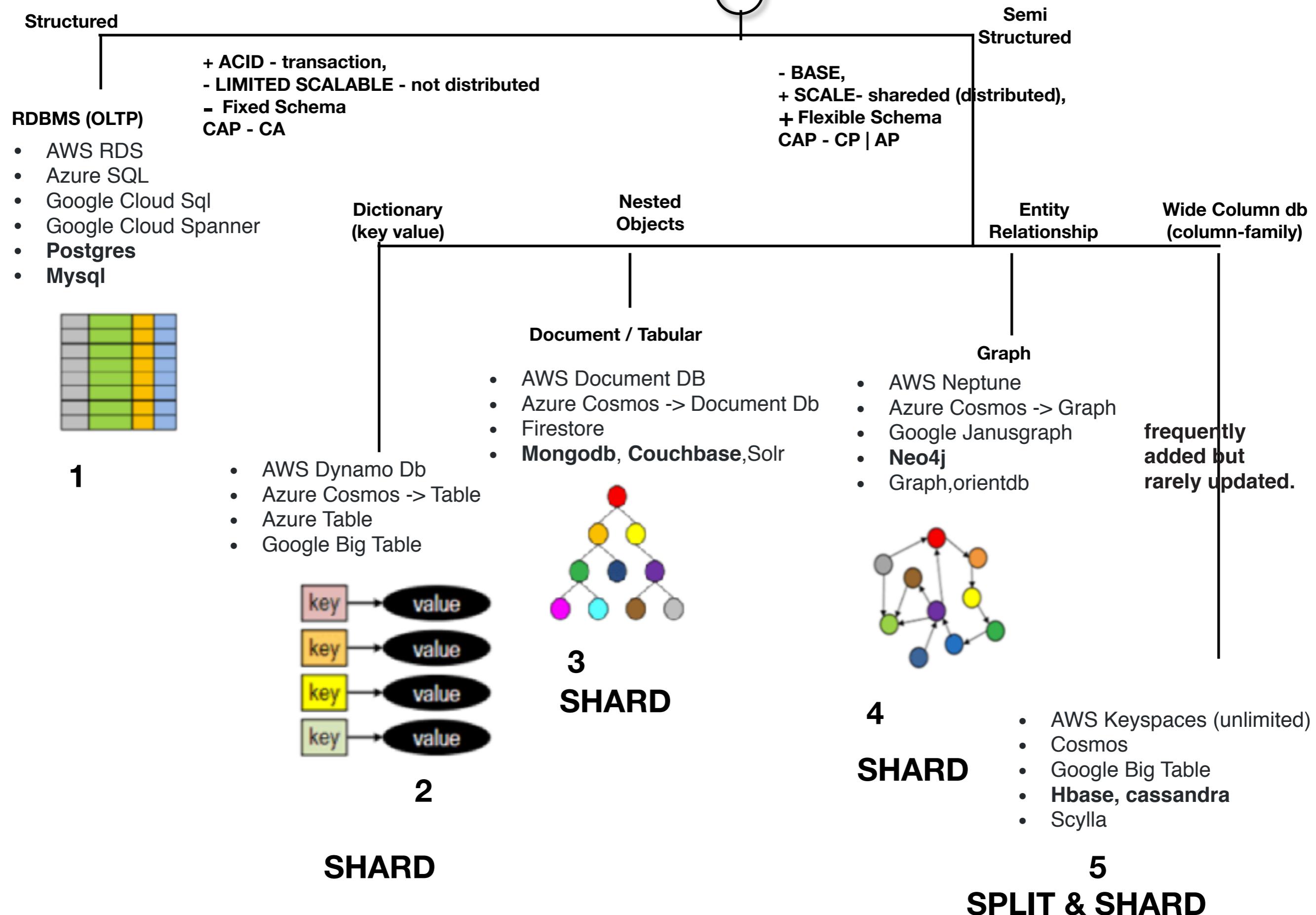


Binary Storage



* AWS DataSync subscription is required to provide support for Server Message Block (SMB) protocol

Operational



Wide Column db (column-family)

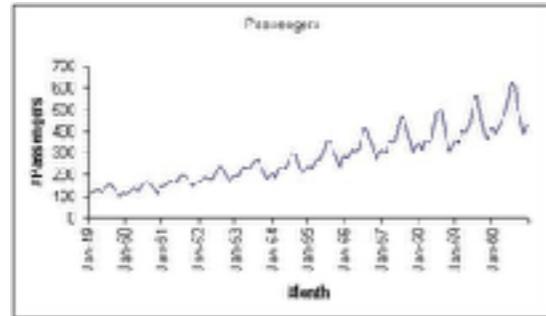
- AWS Keyspaces (unlimited)
- Cosmos
- Google Big Table
- **Hbase, cassandra**
- Scylla

⋮

KairosDB (cassandra)

Time Series

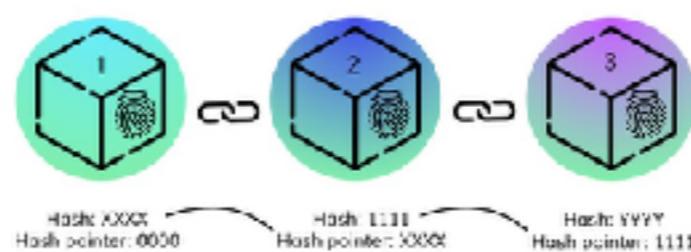
- AWS Timestream
- InfluxDB, Scylla
- OpenTSDB,
- KairosDB
-



Audit Trial

Immutable Logger

- AWS Quantum Ledger
- Azure SQL Database Ledger
- Google ?
- Hyper ledger Fabric



LIZARD GLOBAL



geo Spatial

- AWS Keyspaces
- Azure Cosmos
- Big Table, Big Query
- **Solr, PostGIS**
- Mongodb (GeoJSON)

Analytical



rowkey1	column family (CF11)			column family (CF12)		
	column111	column112	column113	column121	column122	column123
vendor111	value111	vendor112	value112	vendor121	value121	value121
vendor112	value112	vendor122	value122	vendor122	value122	value122
vendor113	value113	value123	value123	value123	value123	value123
value114	value114	value124	value124	value124	value124	value124



Read

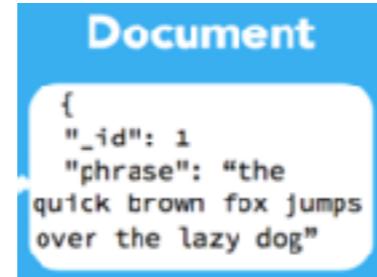
Columnar db (column store)

Cloud DWH (OLAP)

- AWS Redshift (max: 8 PB)
- **Azure Synapse**
- GCP BigQuery
- Snow flakes
- Kudu, Druid, Pinot
- Click house

Read

Text Database



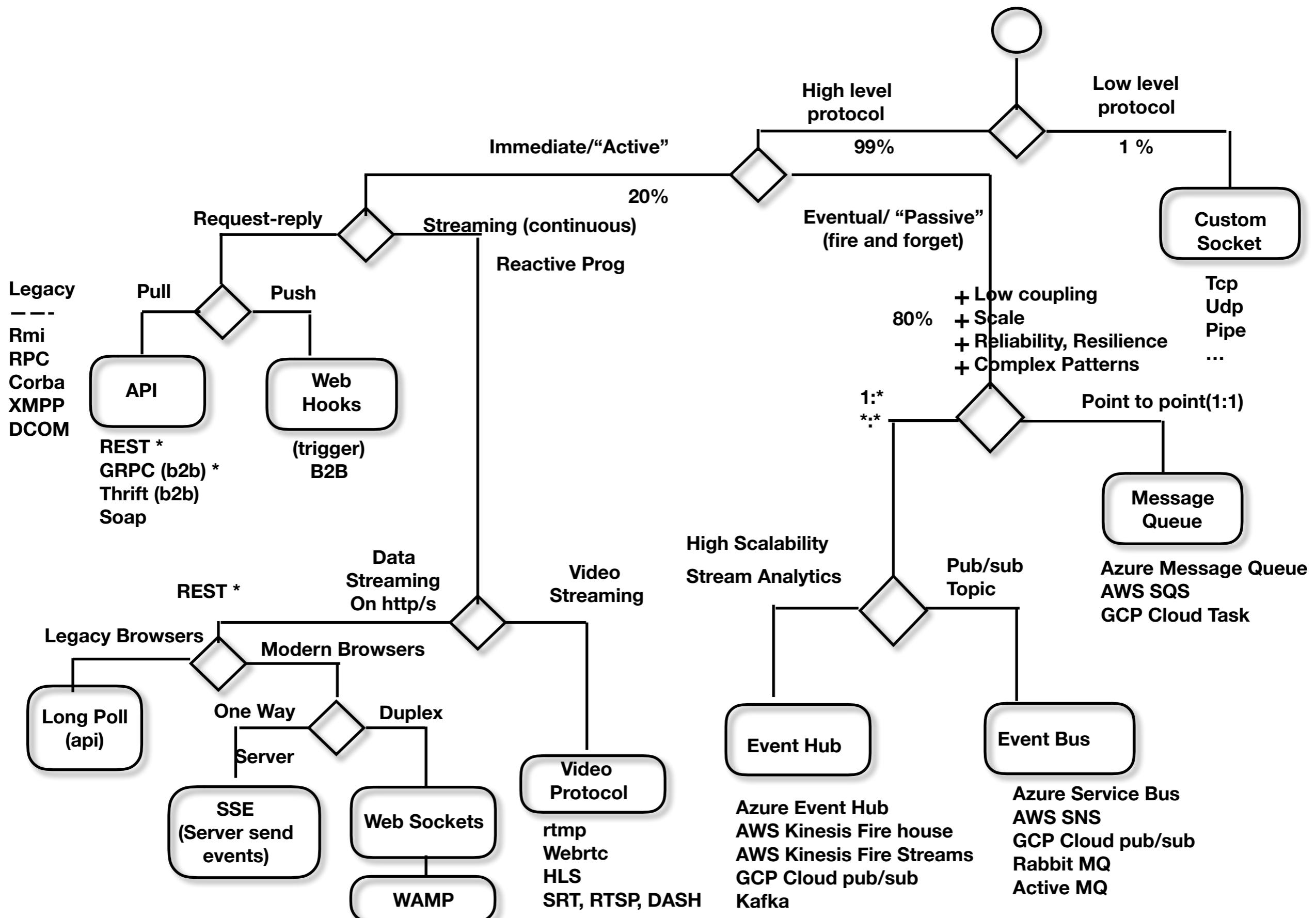
- AWS **Elastic Search**
- AWS cloud Search
- Azure Cognitive Search
- Google Search API
- Elastic Search, **Solr**

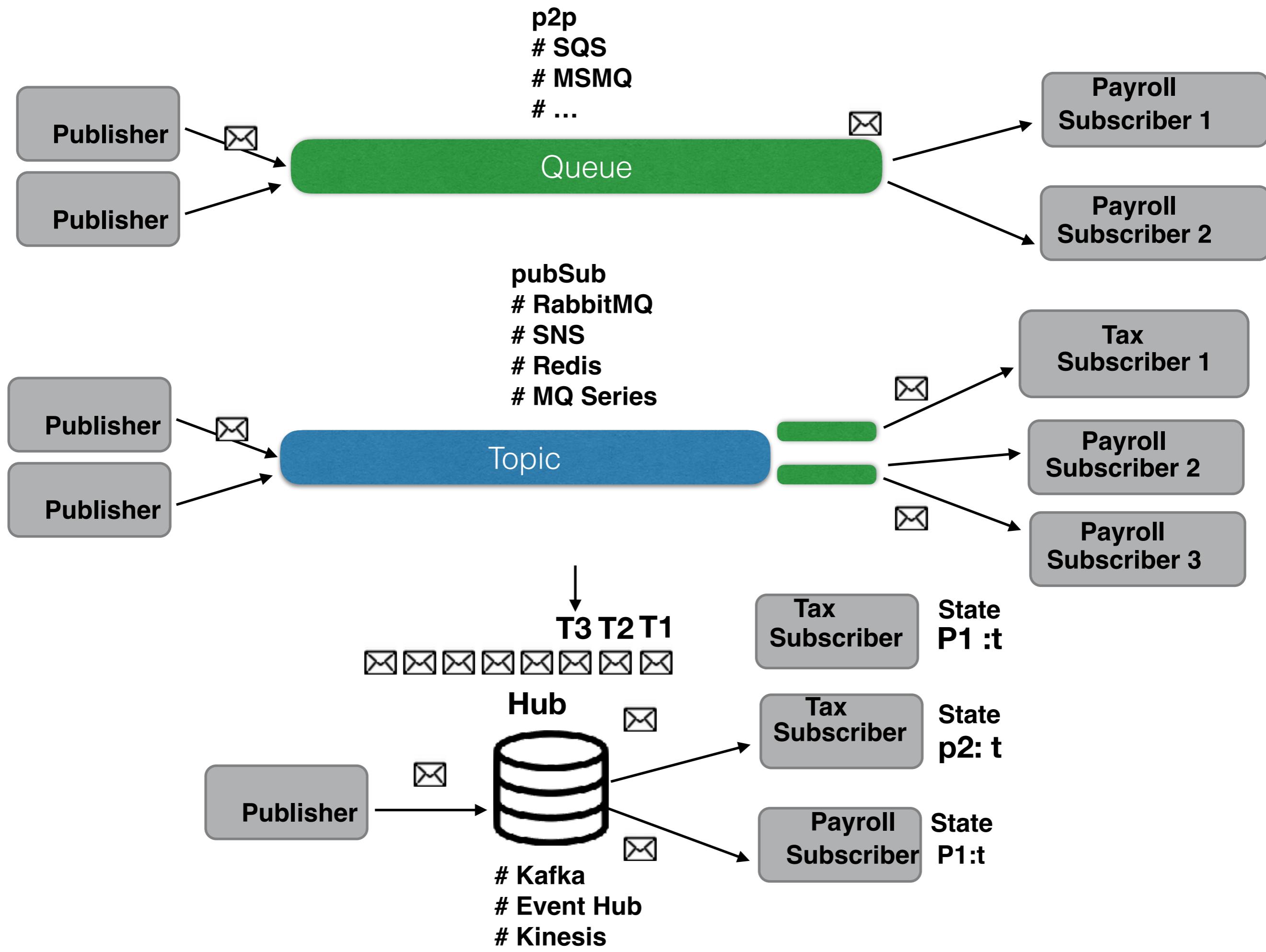
Read/write

Wide Column db (column-family)

- AWS Keyspaces (unlimited)
- Cosmos
- Google Big Table
- **Hbase, cassandra**
- Scylla

Choose Communication protocol





Parallelism

Data Parallelism

**# same code
different data**

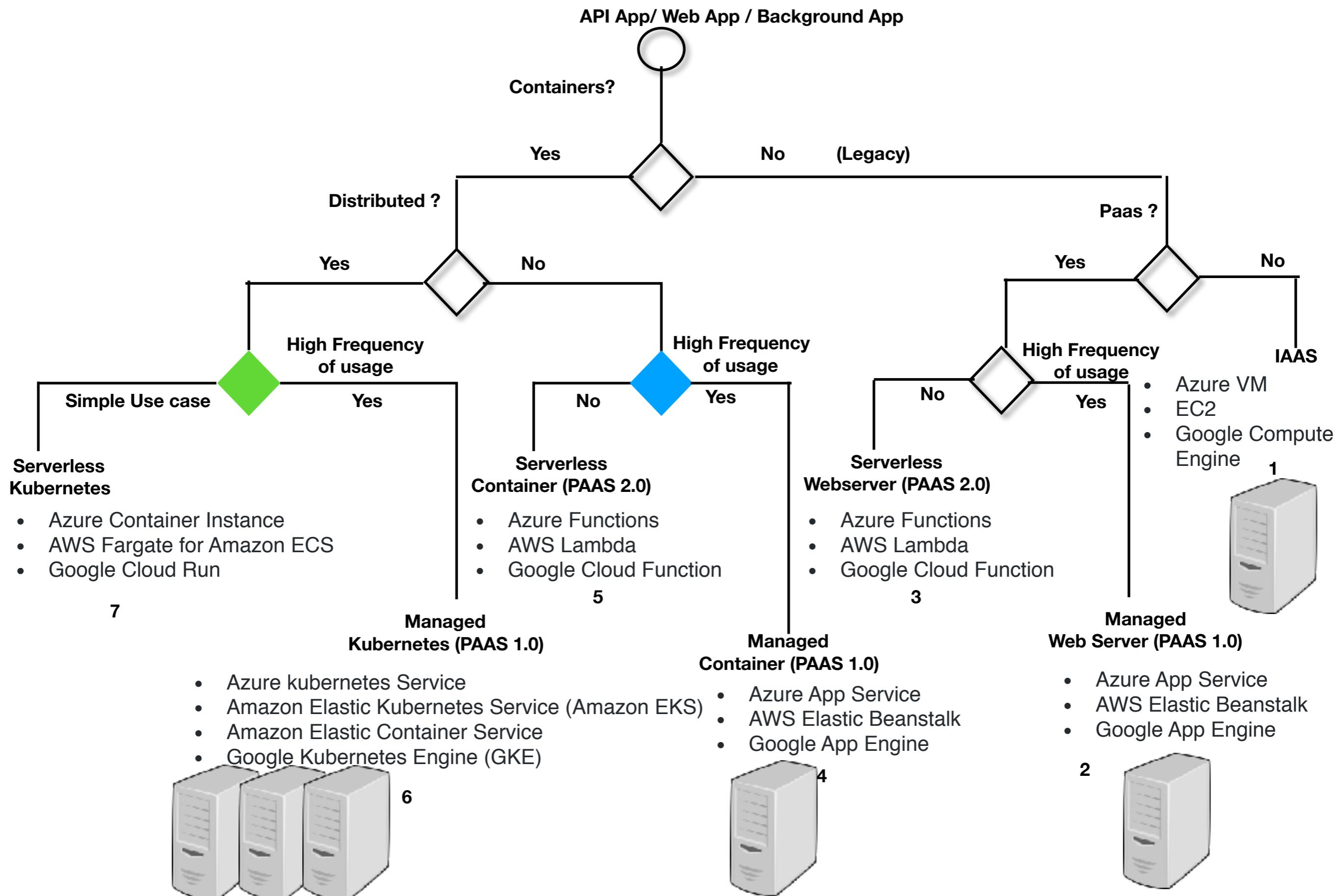
**process salary of
all employees**

Task Parallelism

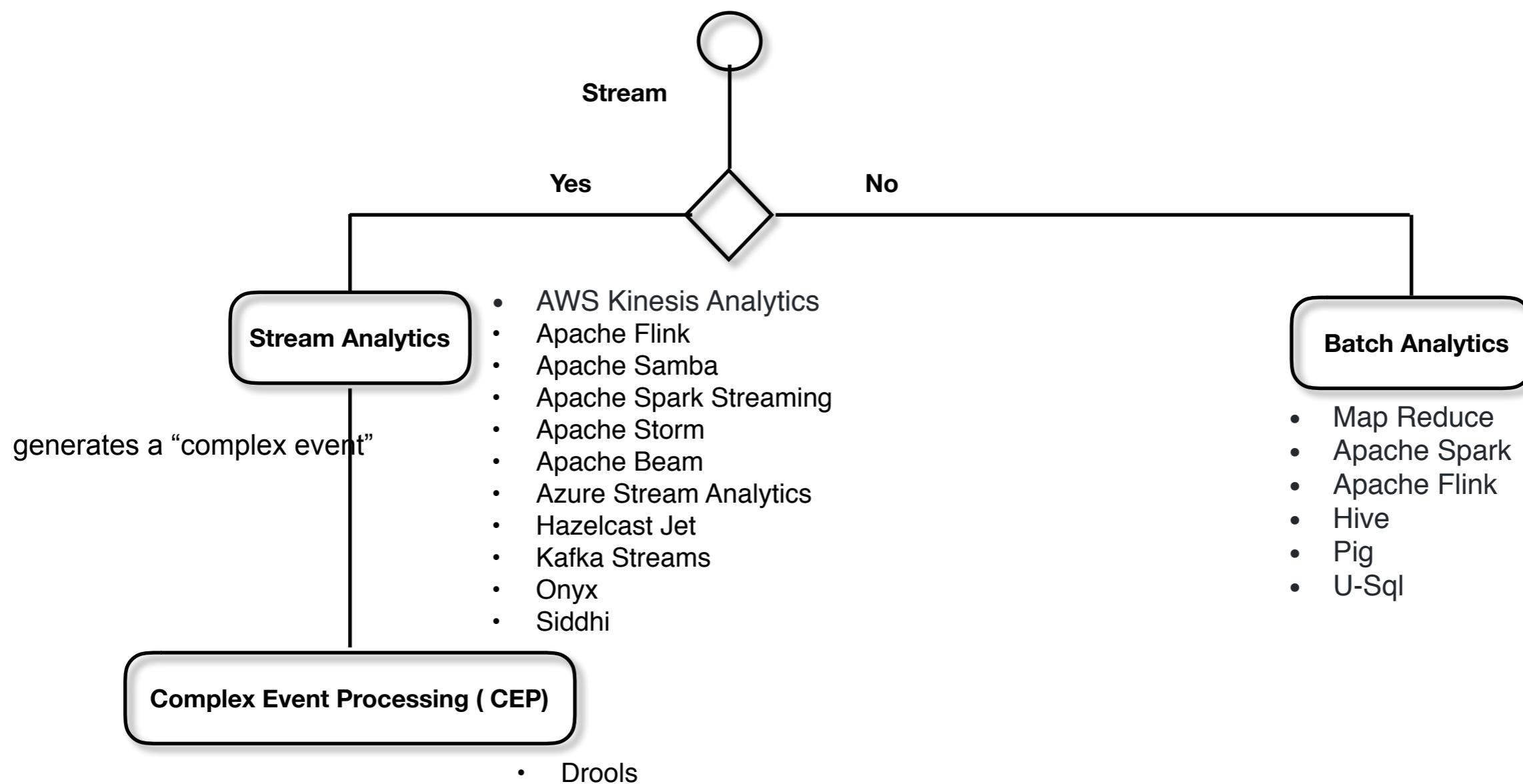
**# different code
different data**

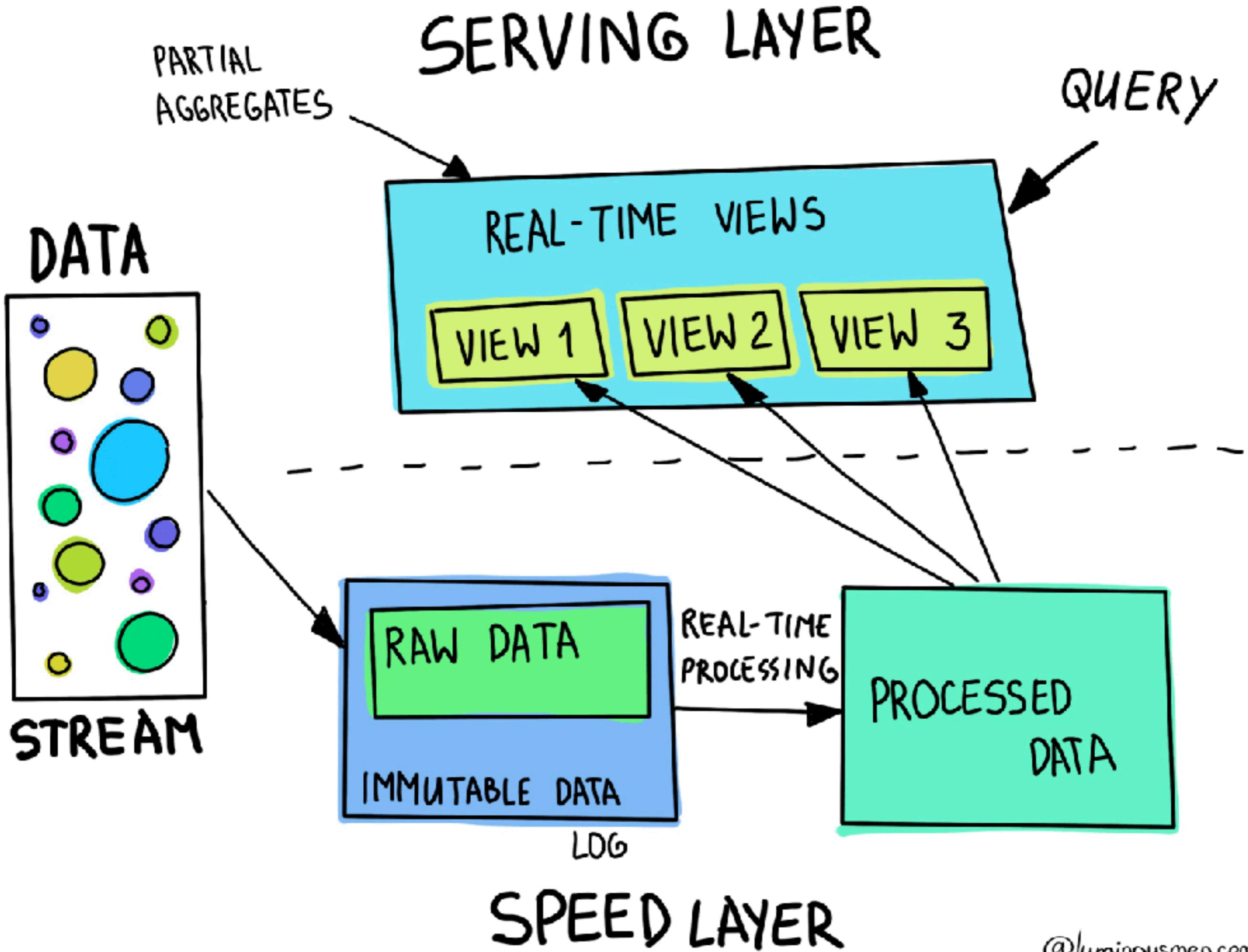
**When Order is
created,
Dispatch the order
Update RO Level**

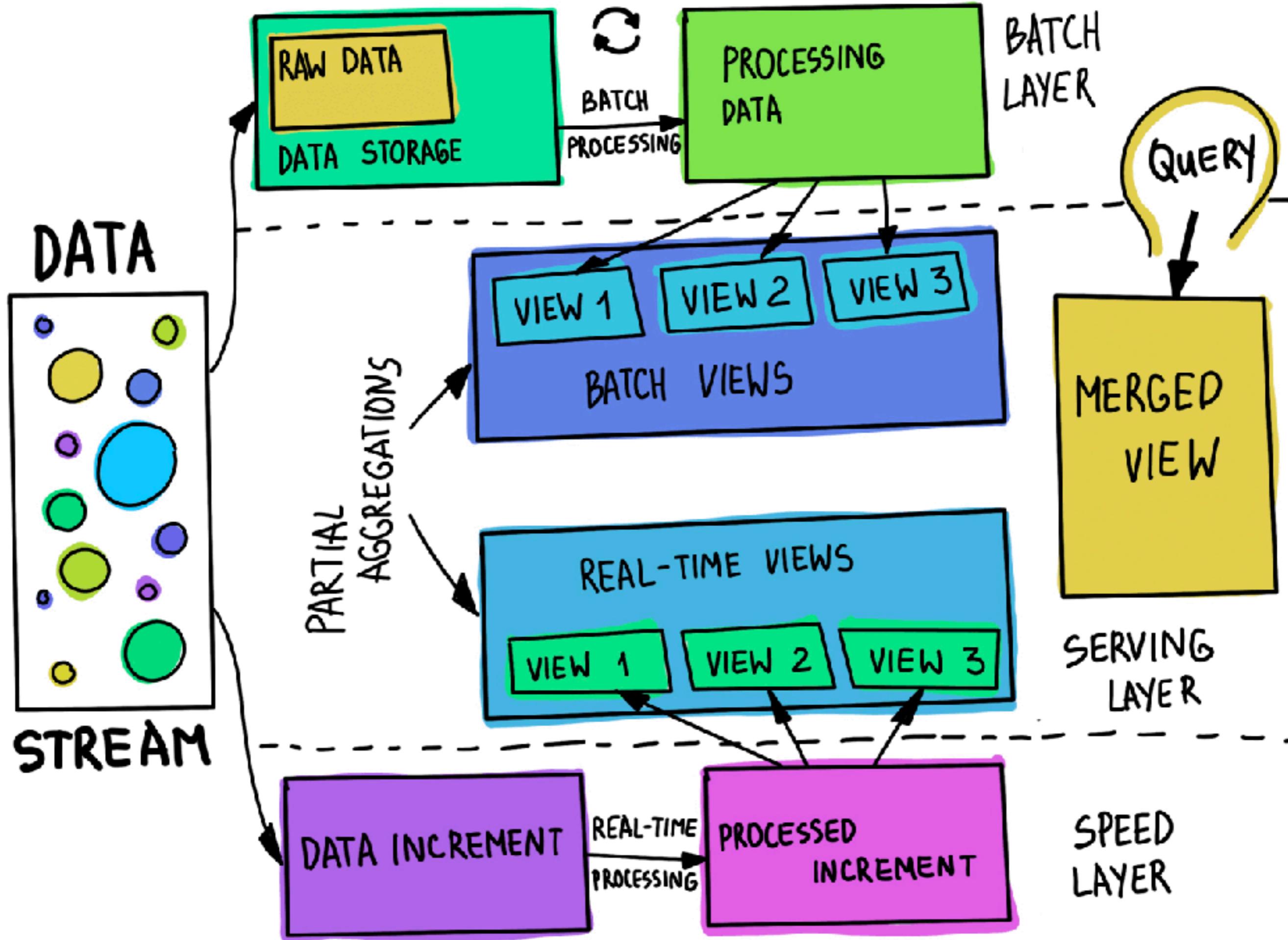
Choose Operational Compute

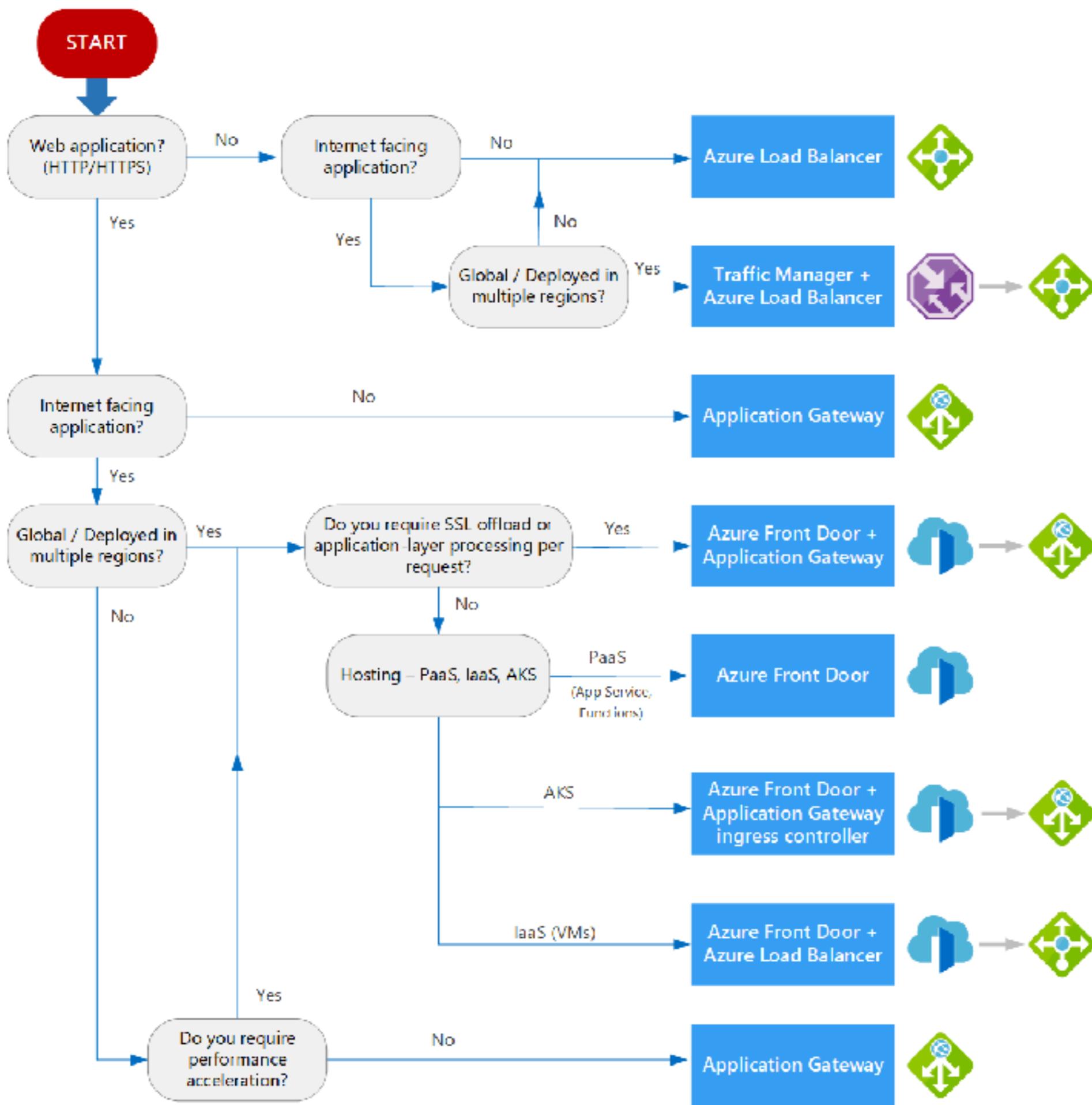


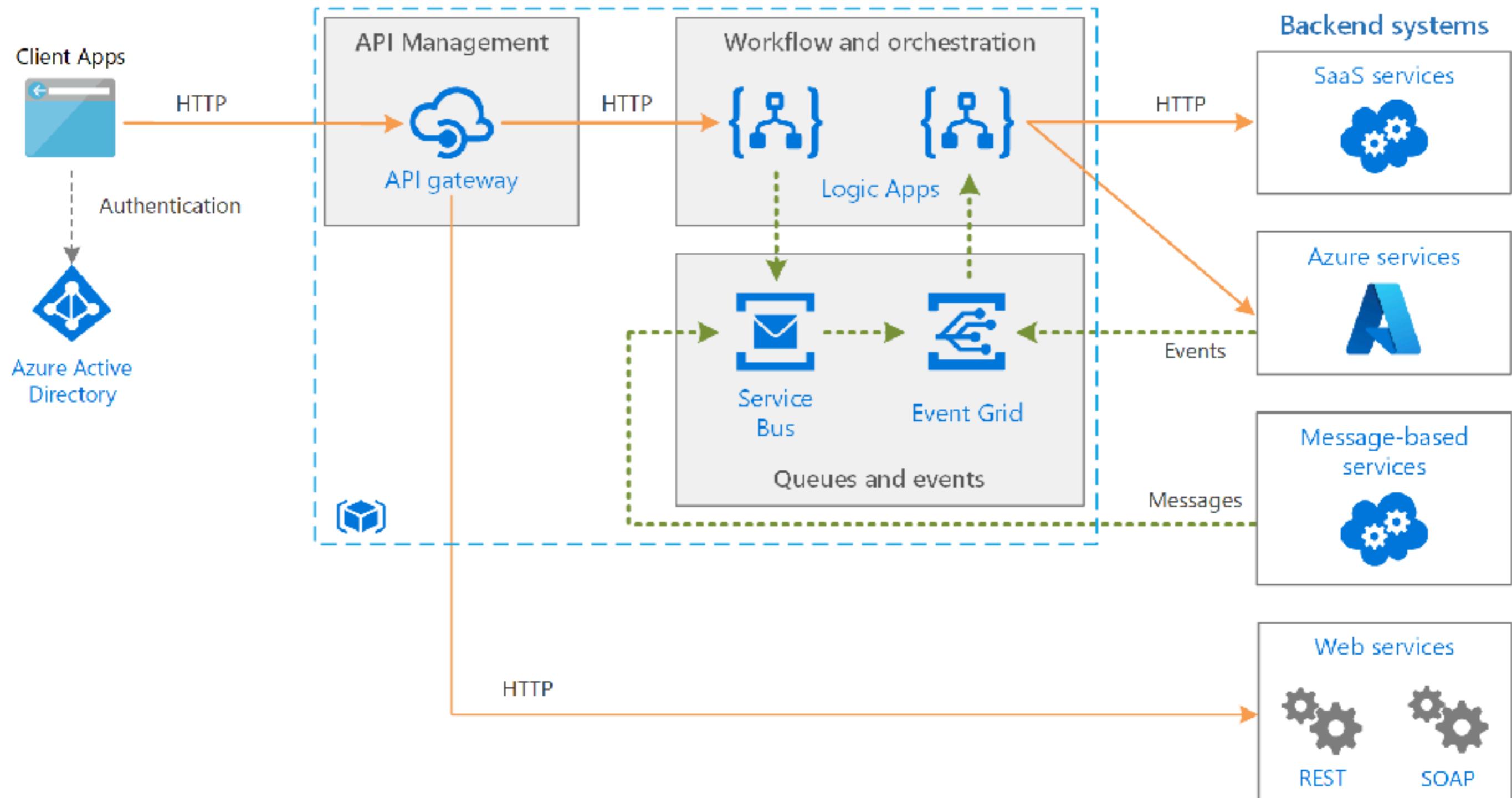
Choose Analytical Compute





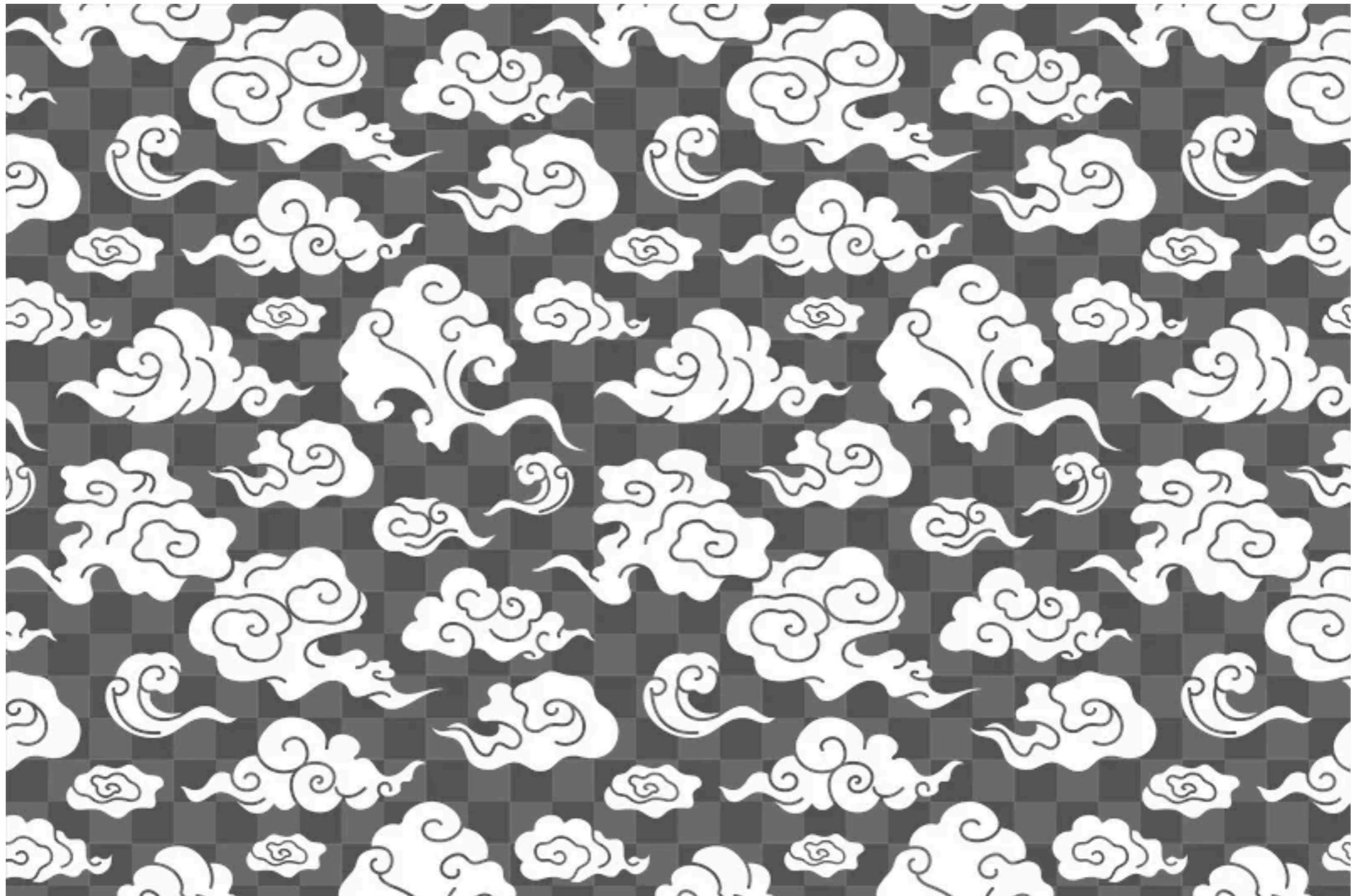




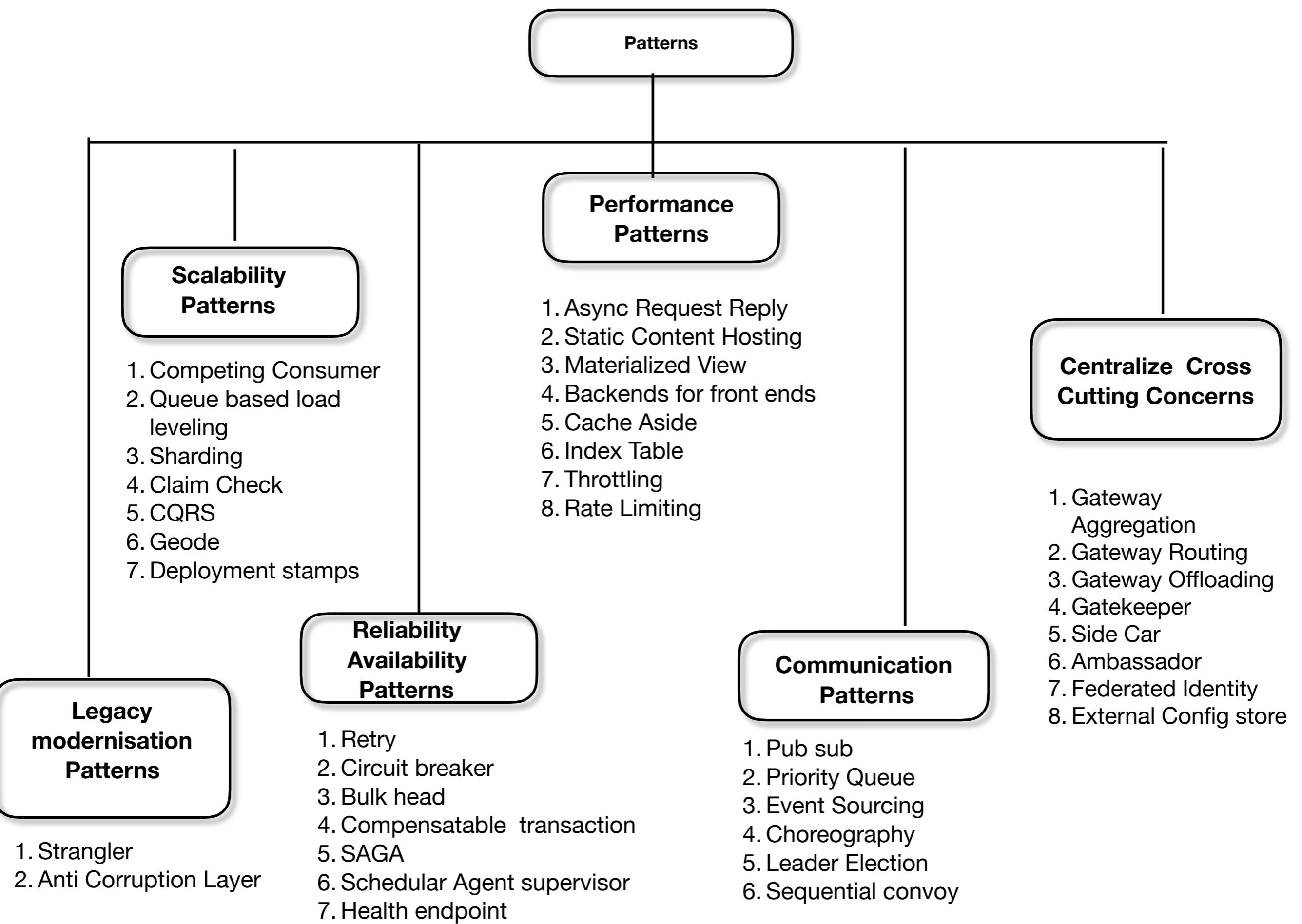


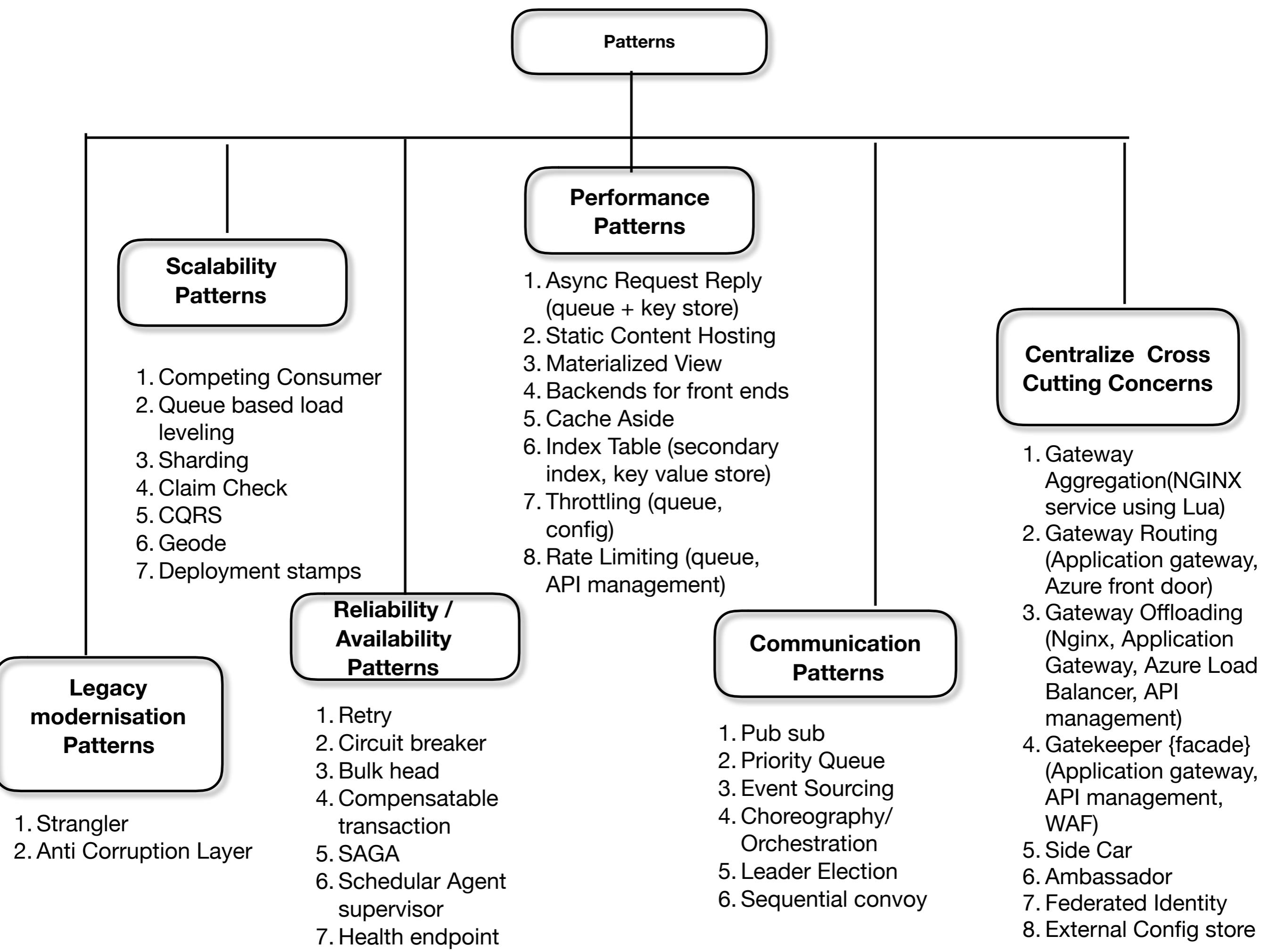
While API Gateways and API management can be used interchangeably, strictly speaking an API gateway refers to the individual proxy server, while API management refers to the overall solution of managing APIs in production which includes a set of API gateways acting in a cluster, an administrative UI, and may even include additional items such as a developer portal for customers to sign up and generate new API keys.

Service	Global/regional	Recommended traffic
Azure Front Door	Global	HTTP(S)
Traffic Manager	Global	non-HTTP(S)
Application Gateway	Regional	HTTP(S)
Azure Load Balancer	Regional	non-HTTP(S)



Architecture patterns





Maintainability

- Edge Workload config

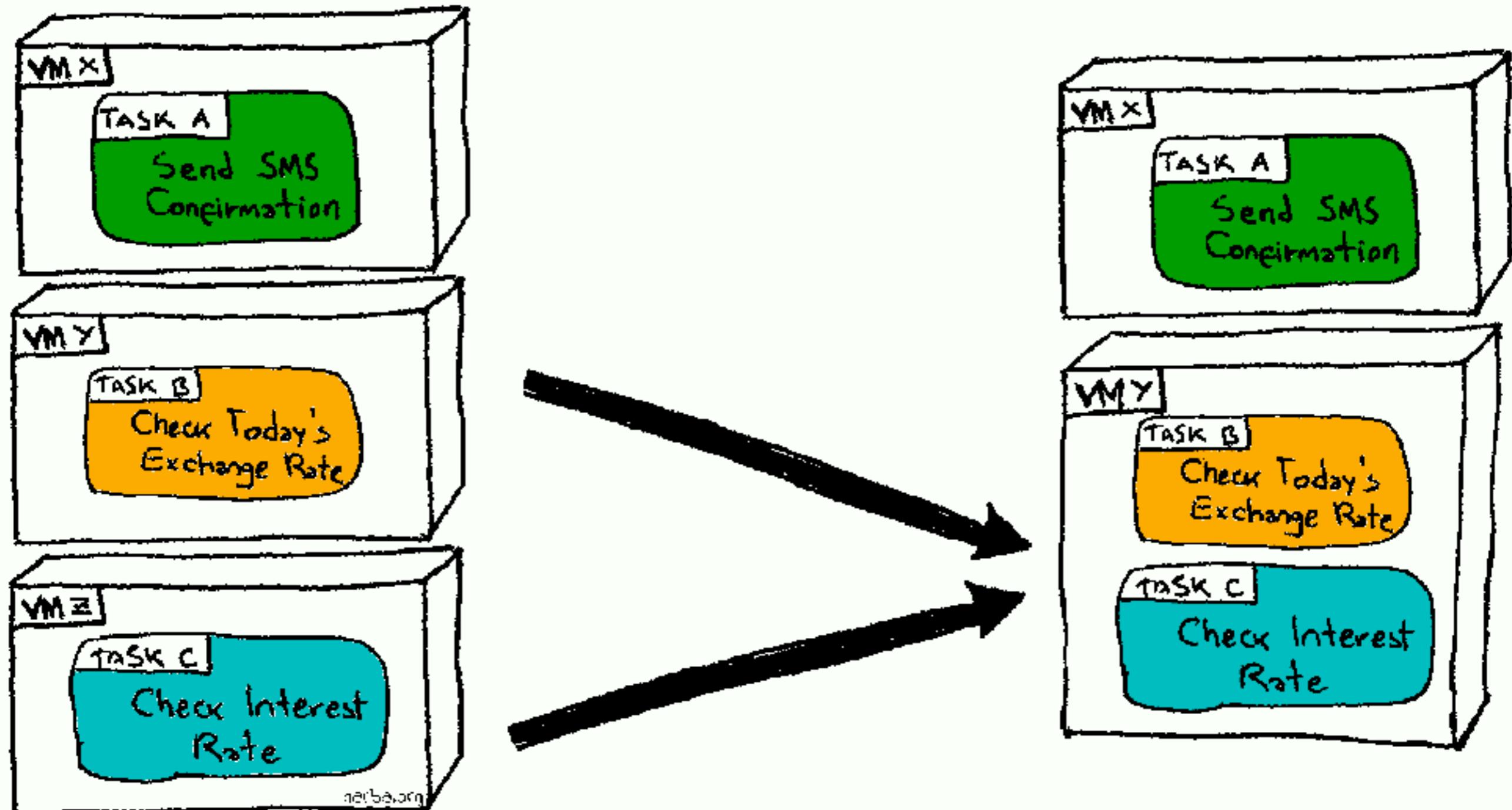
Cost

- Compute Resource consolidation

Security

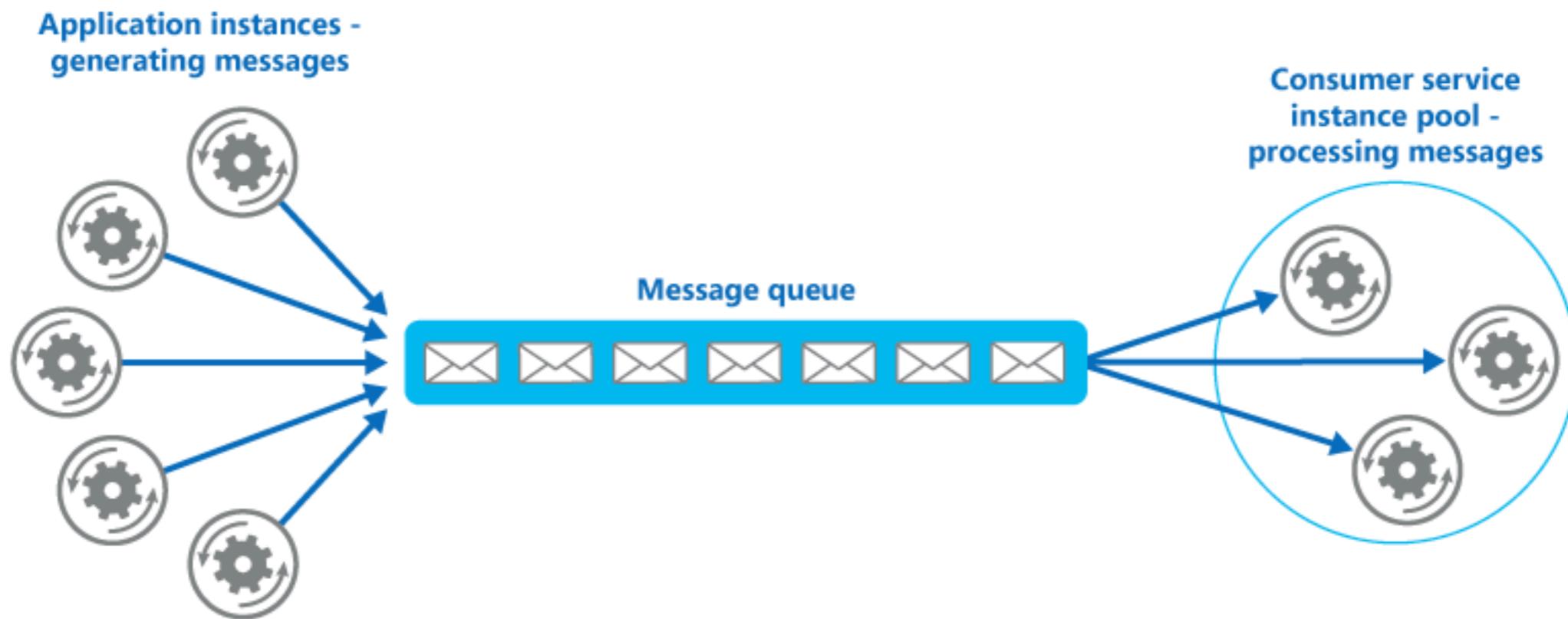
- Valet Key

Compute Resource Consolidation Pattern



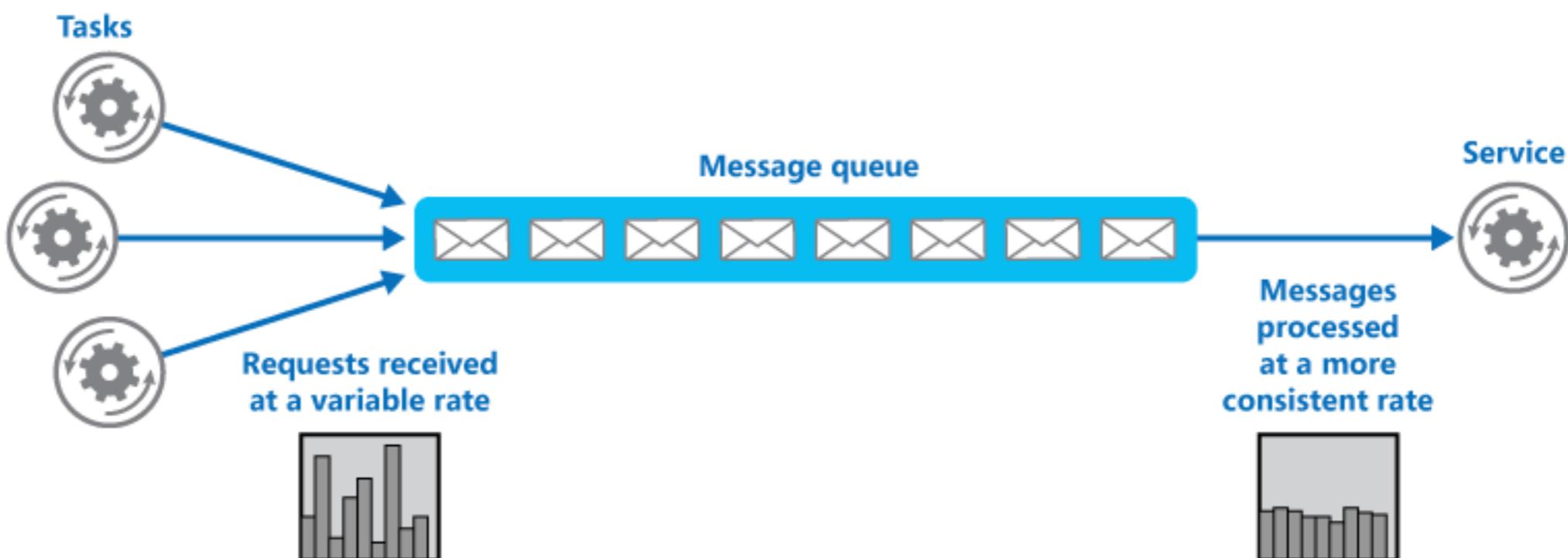
Scalability Patterns

Competing Consumers pattern



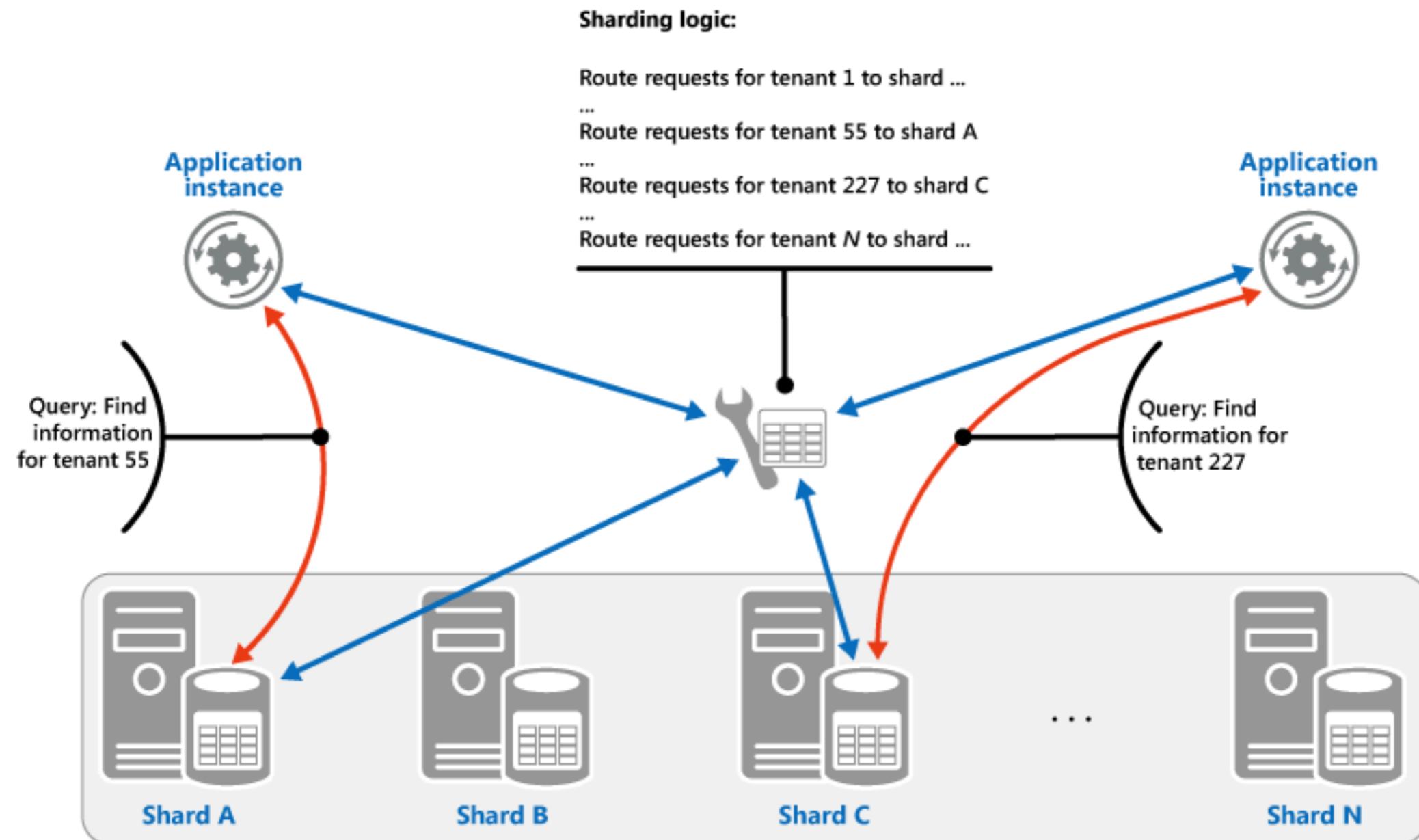
- The volume of work is highly variable, requiring a scalable solution.

Queue-Based Load Leveling pattern



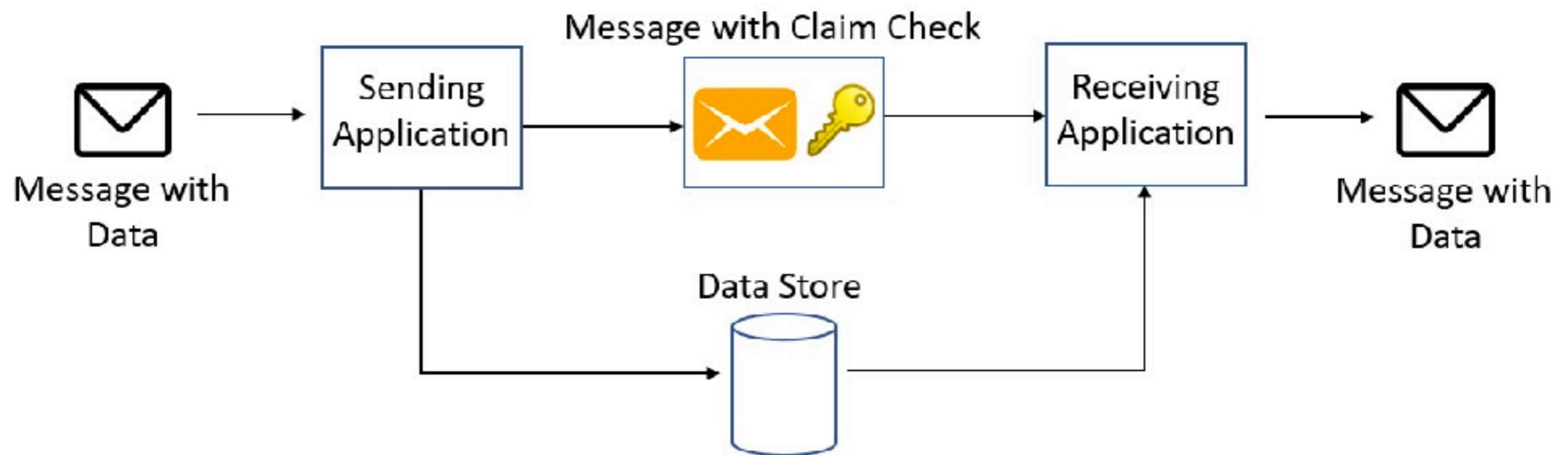
useful to any application that uses services that are subject to overloading.

Sharding



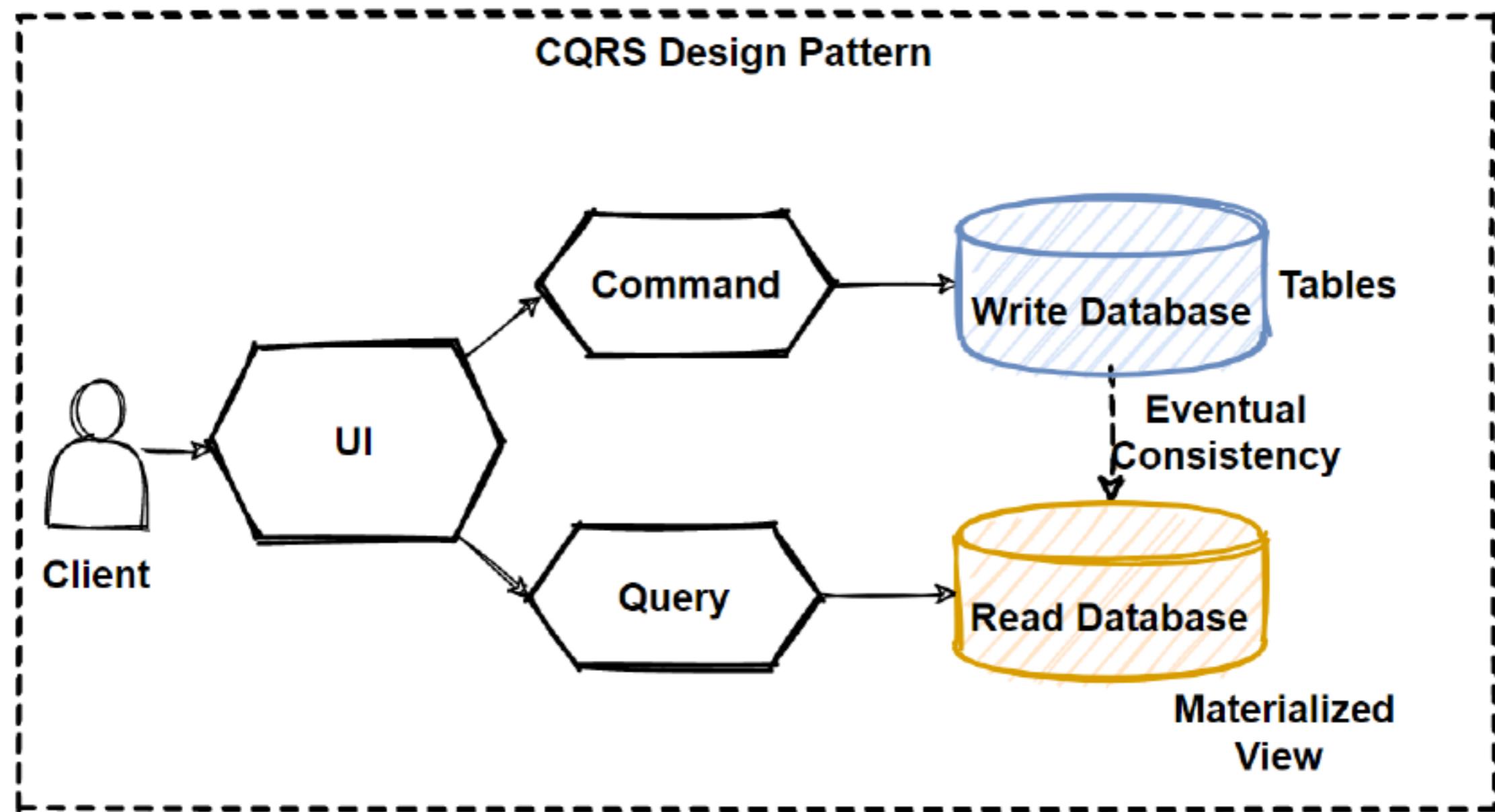
when a data store is likely to need to scale beyond the resources available to a single storage node

Claim-Check pattern



used whenever a message cannot fit the supported message limit of the chosen message bus

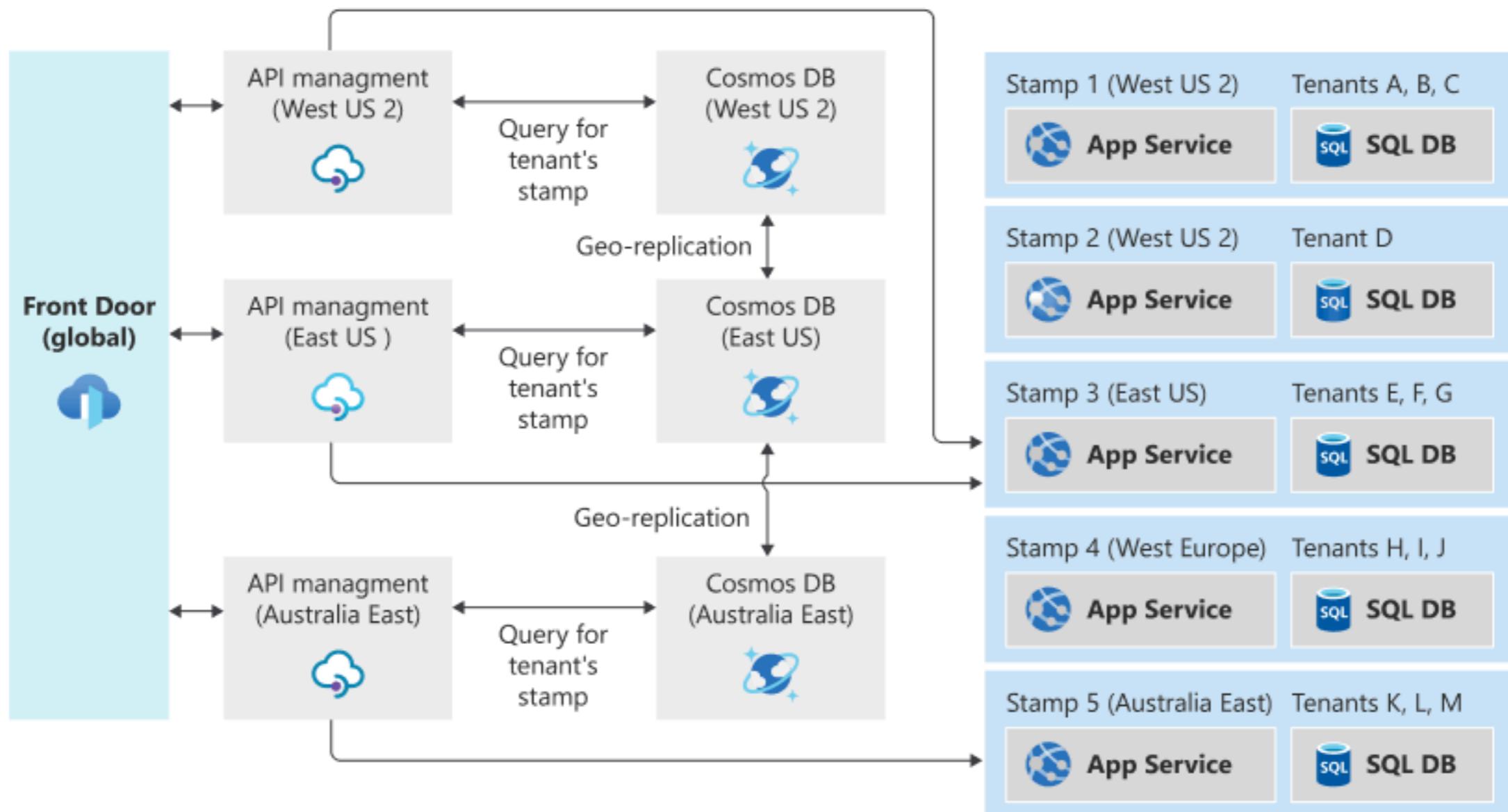
CQRS pattern



- Scenarios where one team of developers can focus on the complex domain model that is part of the write model, and another team can focus on the read model and the user interfaces.

Consider applying CQRS to limited sections of your system where it will be most valuable.

Deployment Stamps pattern



In a deployment stamp architecture, multiple independent instances of your system are deployed and contain a subset of your customers and users. In geodes, all instances can serve requests from any users, but this architecture is often more complex to design and build

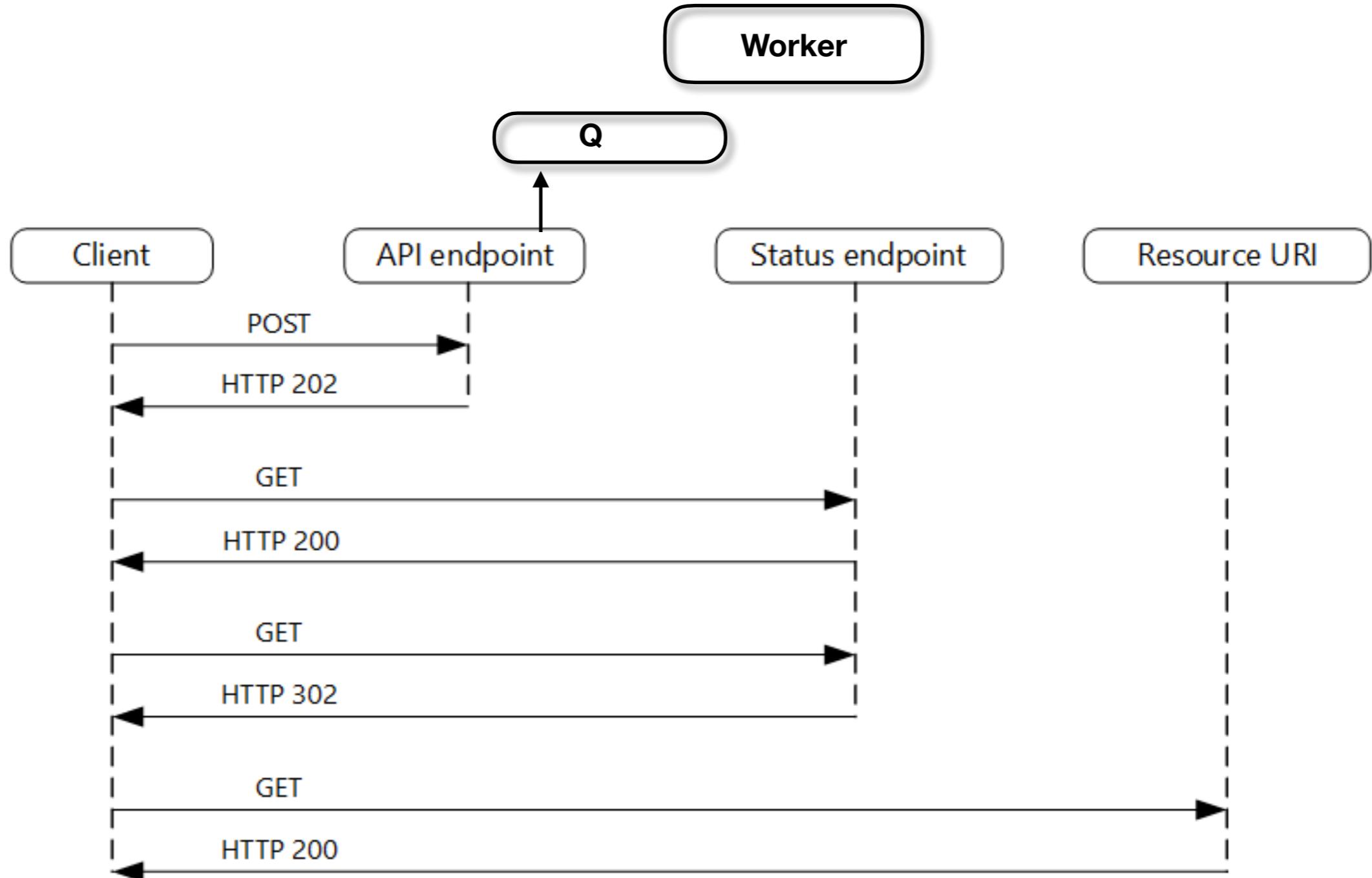


If the user travels to California and then accesses the system, their connection will likely be routed through the West US 2 region because that's closest to where they are geographically when they make the request. However, the request has to ultimately be served by stamp 3 because that's where their data is stored. The traffic routing system ensures that the request is routed to the correct stamp.

Each geode is behind a global load balancer, and uses a geo-replicated read-write service like Azure Cosmos DB to host the data plane, ensuring cross-geode data consistency. Data replication services ensure that data stores are identical across geodes, so *all* requests can be served from *all* geodes. You can use Azure Traffic Manager or Azure Front Door for fronting the geodes, while Azure Cosmos DB can act as the replication backplane.

Performance Patterns

Asynchronous Request-Reply

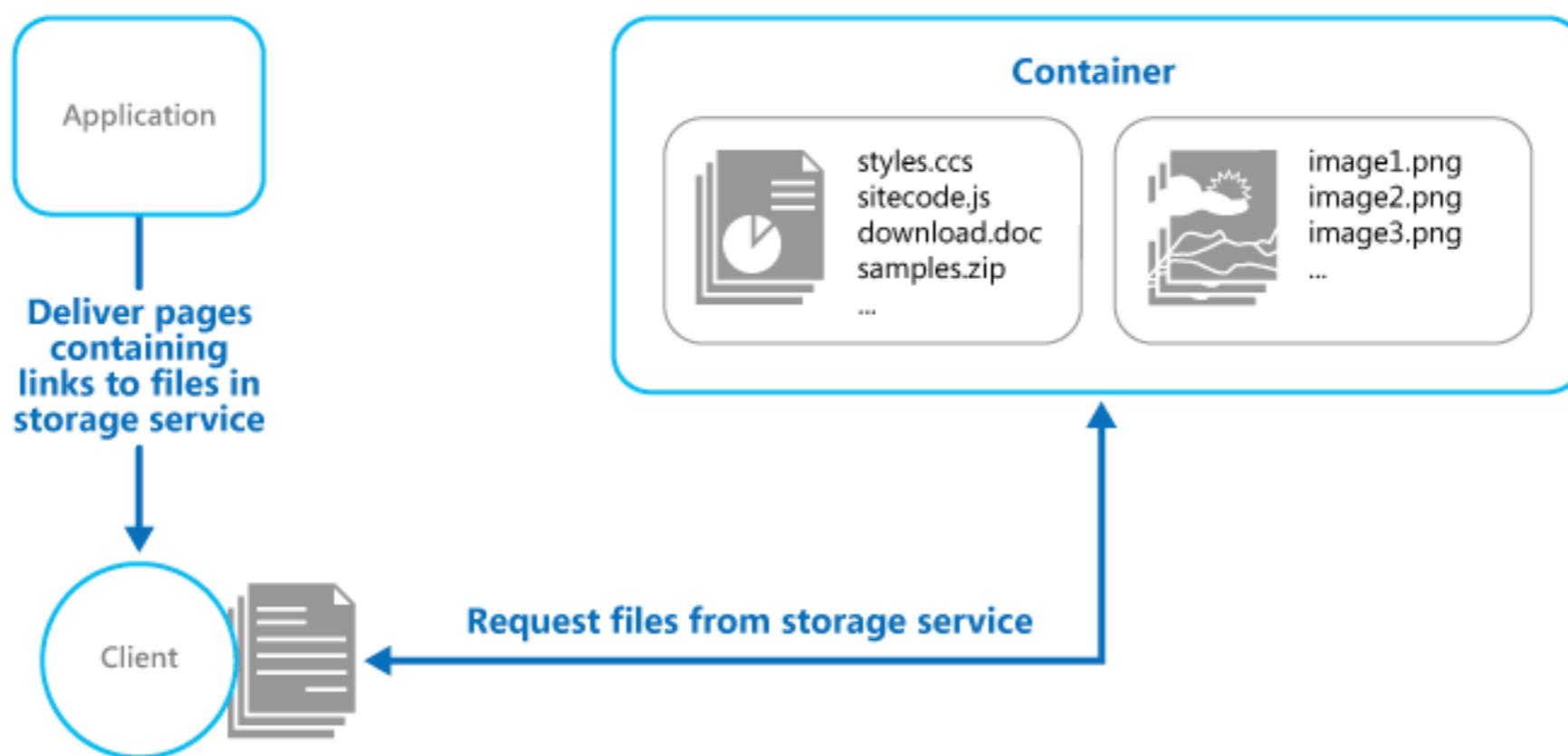


- Service calls that need to be integrated with legacy architectures that don't support modern callback technologies such as Notification, WebSockets or webhooks.

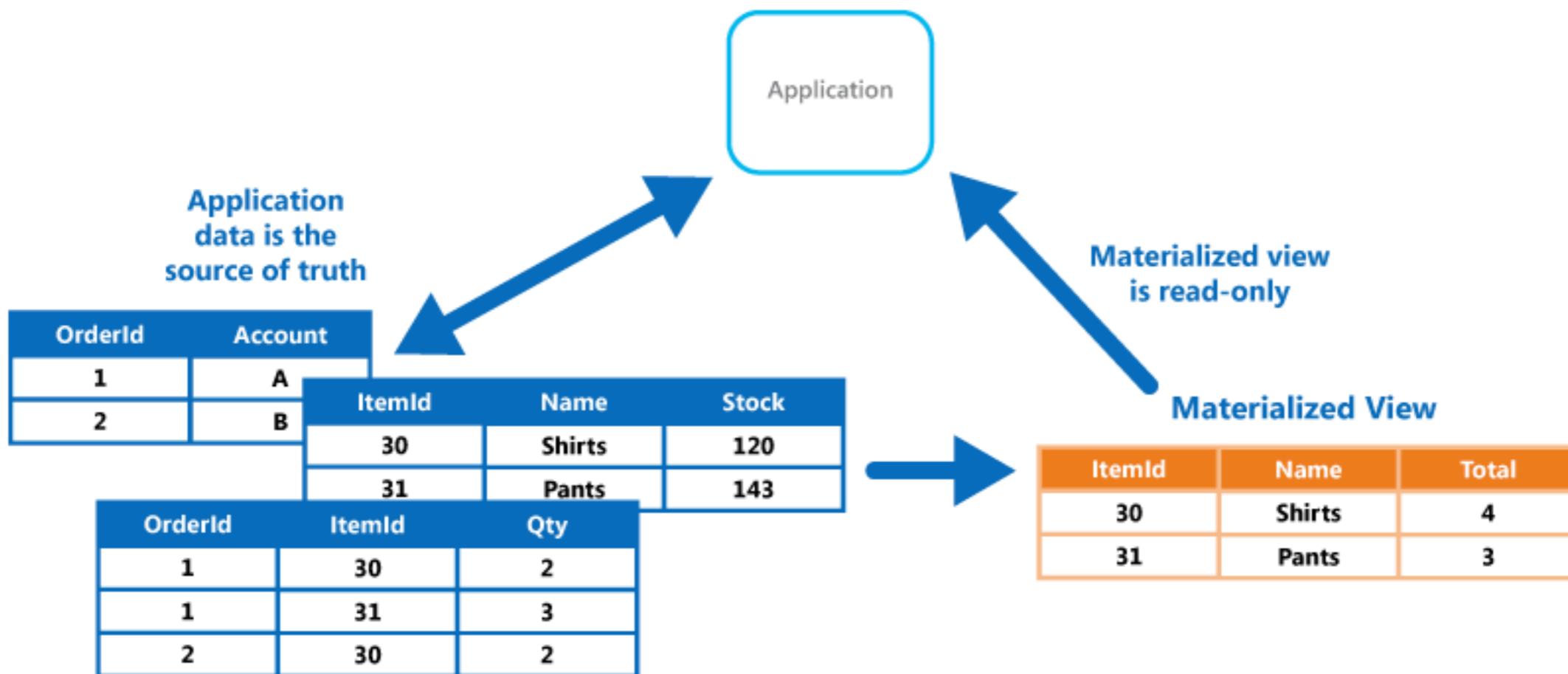
Static Content Hosting

Html
Js
Css
Image

Jsp
Aspx
Rest api

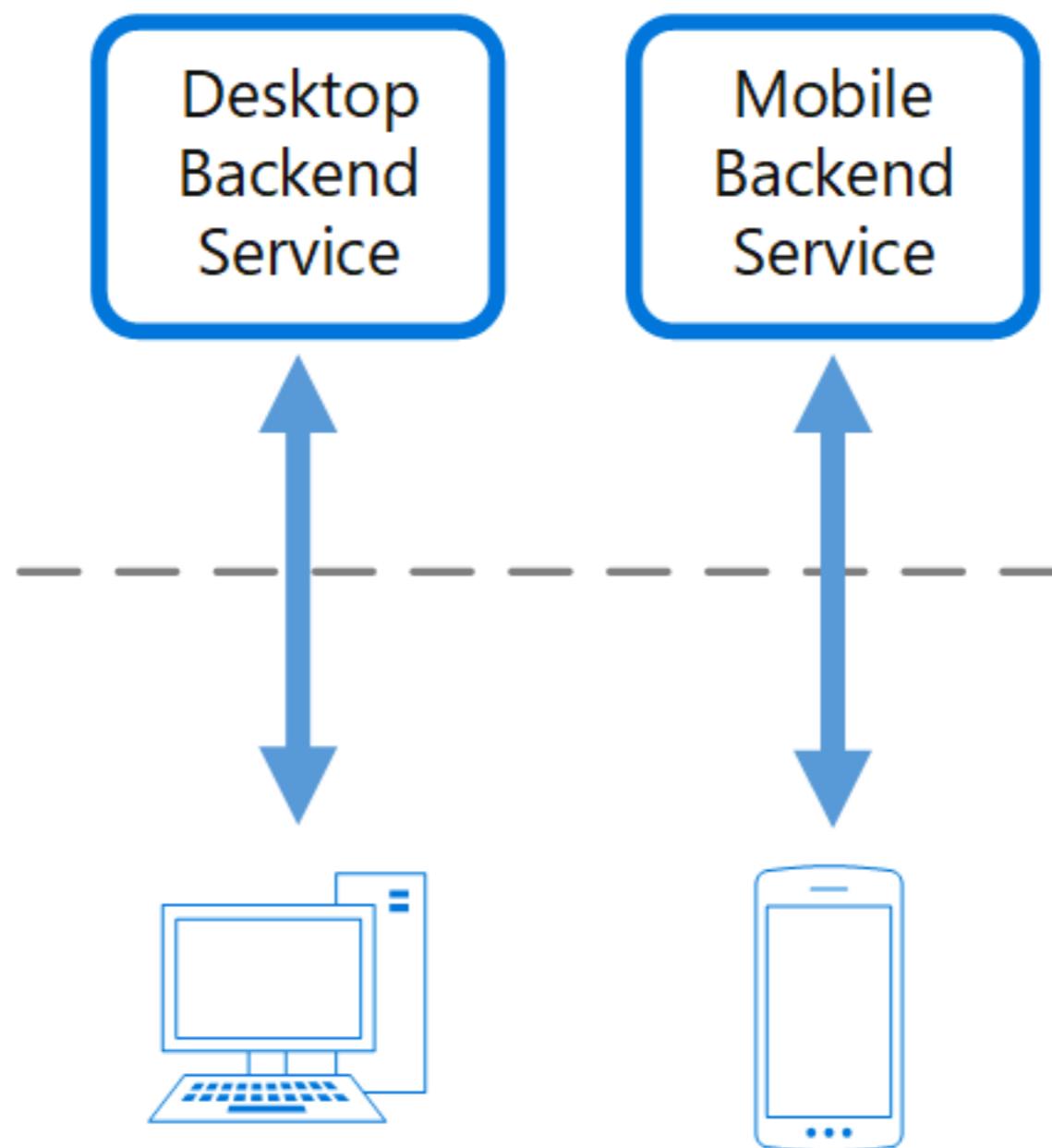


Materialized View



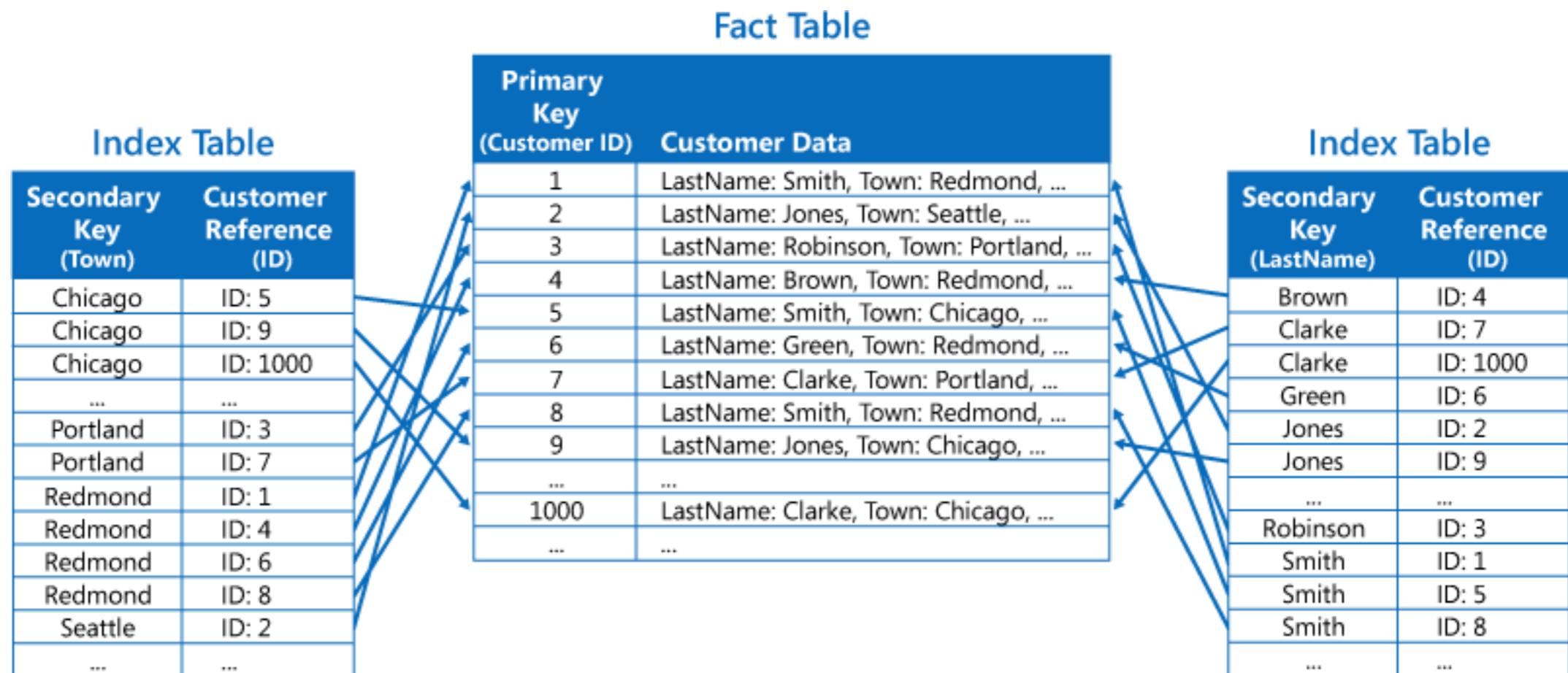
data that's difficult to query directly, or where queries must be very complex to extract data that's stored in a normalized, semi-structured, or unstructured way.

Backends for Frontends



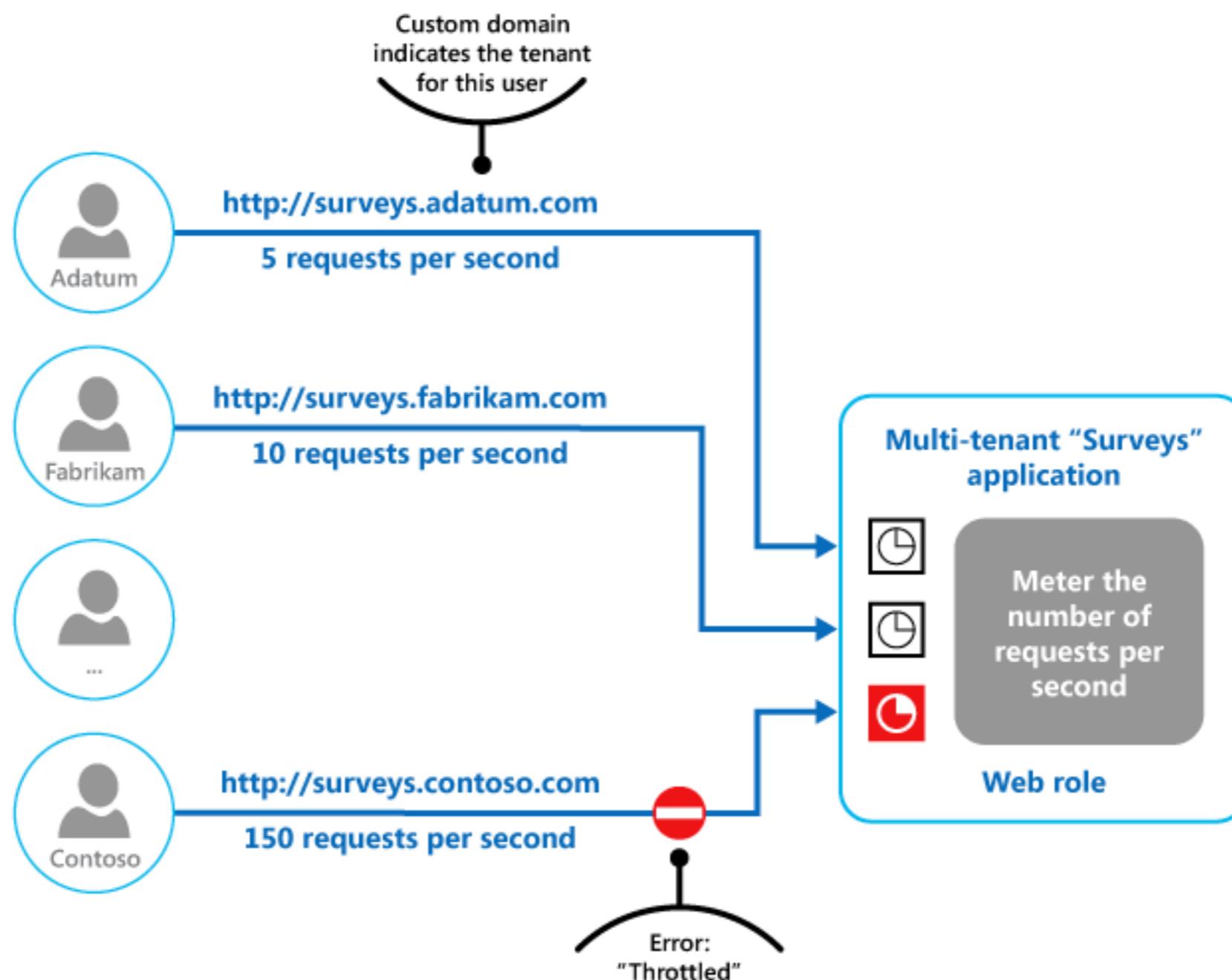
useful when you want to avoid customizing a single backend for multiple interfaces

Index Table



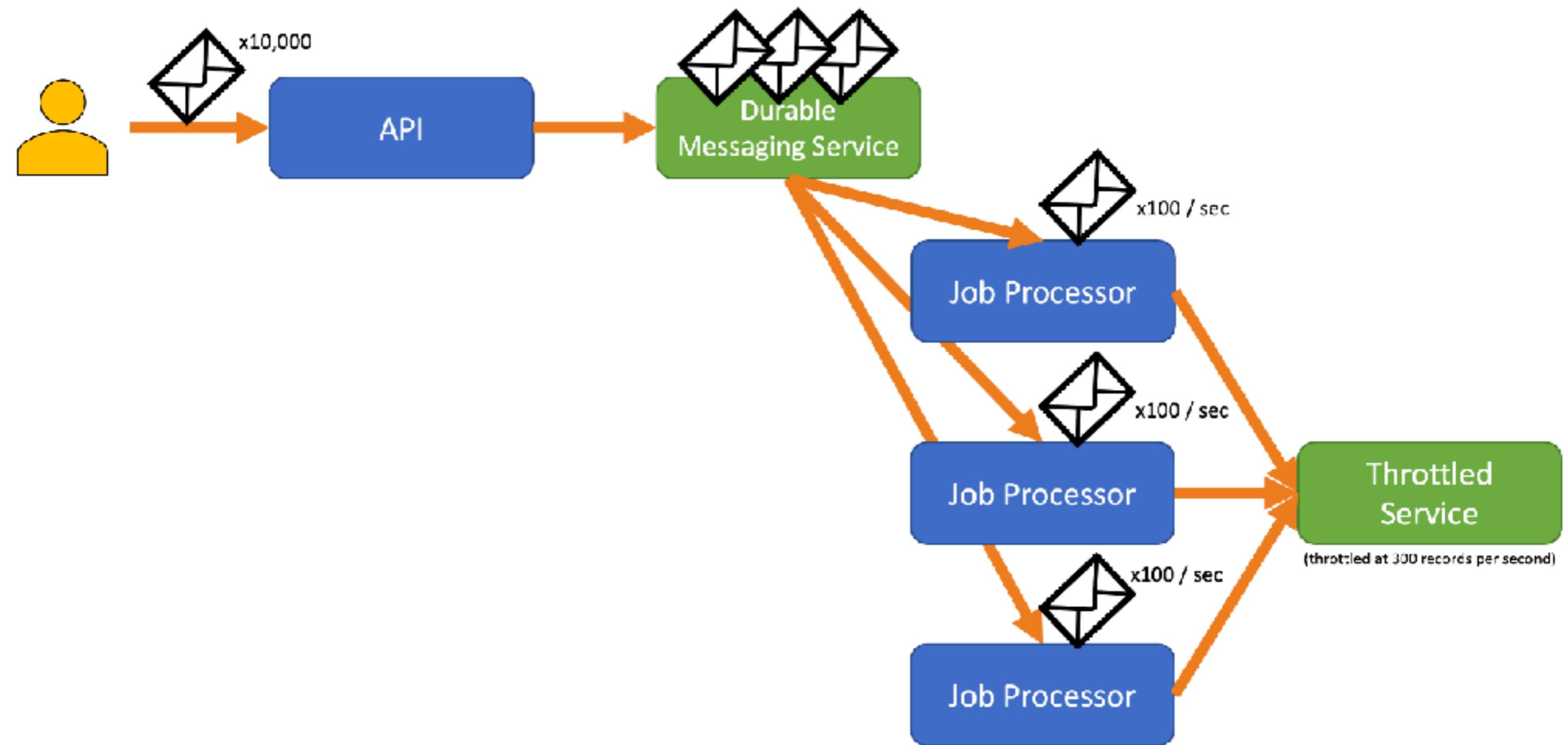
Use this pattern to improve query performance when an application frequently needs to retrieve data by using a key other than the primary

Throttling



To handle bursts

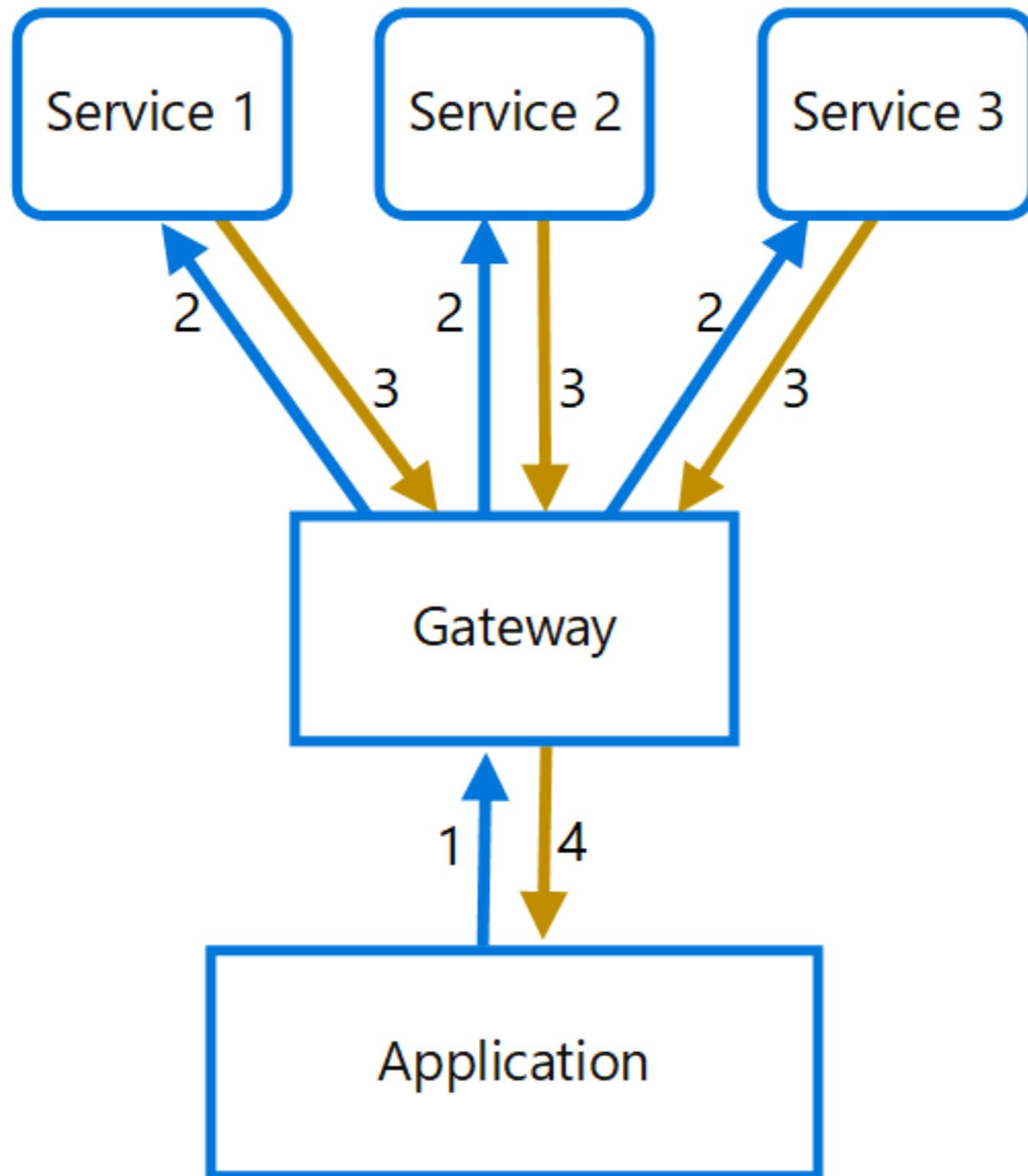
Rate Limiting



- Reduce throttling errors raised by a throttle-limited service.

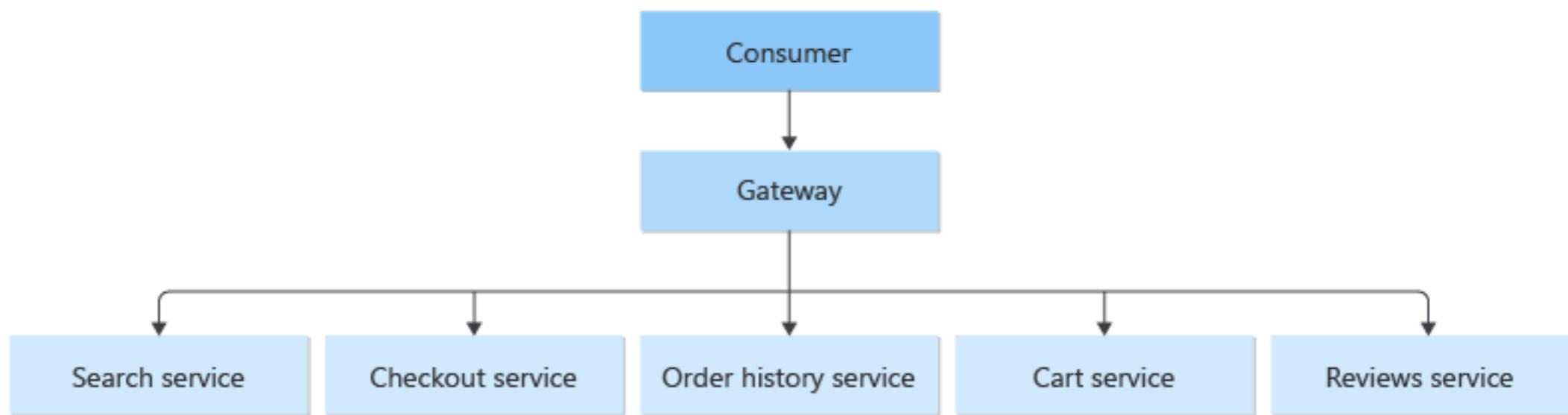
**Centralize Cross
Cutting Concerns**

Gateway Aggregation



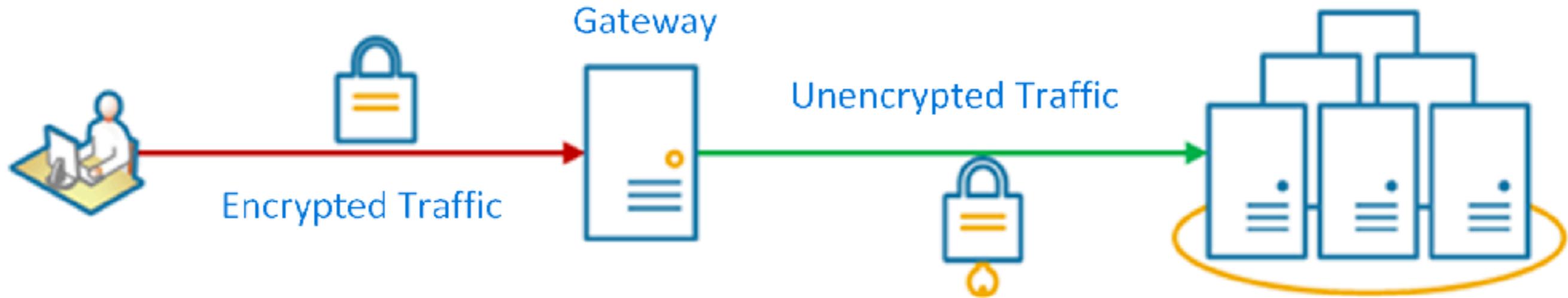
- A client needs to communicate with multiple backend services to perform an operation.

Gateway Routing



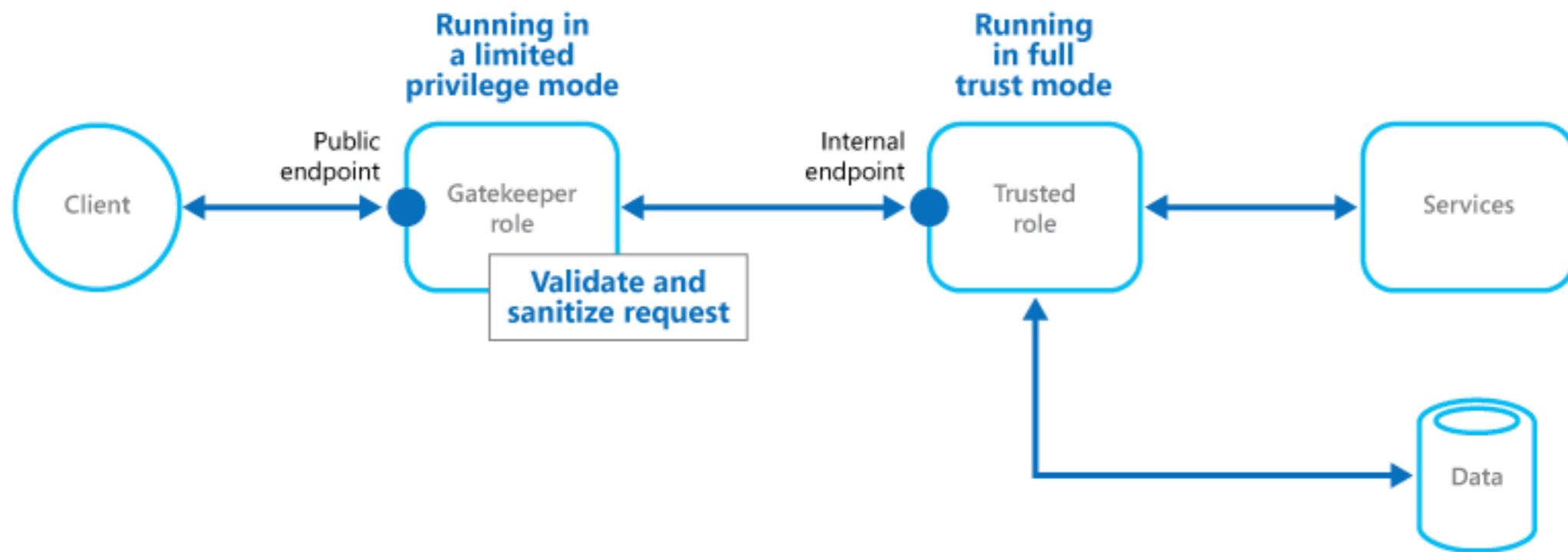
- abstract backend services from the clients
- Routing lets you control what version of the service is presented to the clients

Gateway Offloading



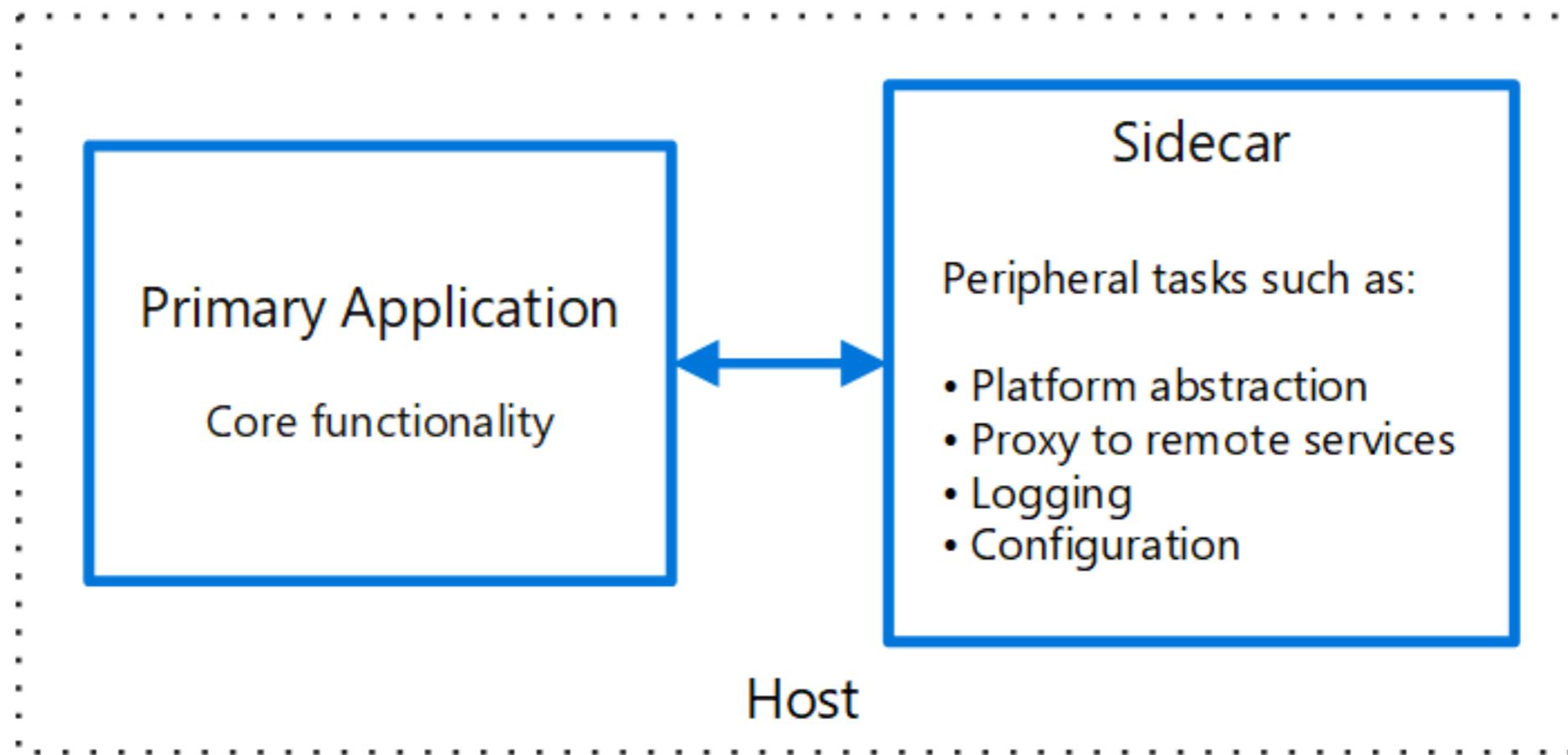
- wish to move the responsibility for issues such as network security, throttling, or other network boundary concerns to a more specialized team.

Gatekeeper



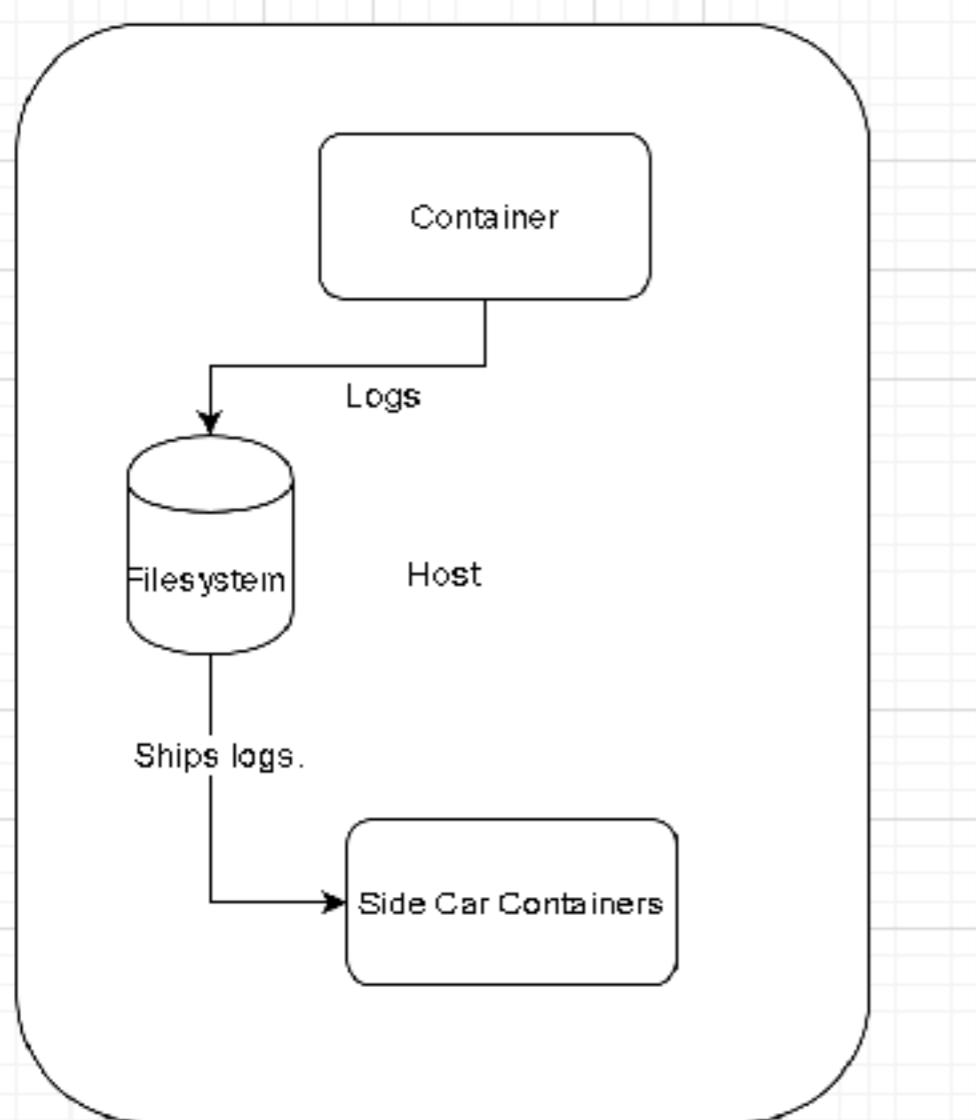
acts like a firewall in a typical network topography. It allows the gatekeeper to examine requests and make a decision about whether to pass the request on to the trusted host that performs the required tasks.

Sidecar

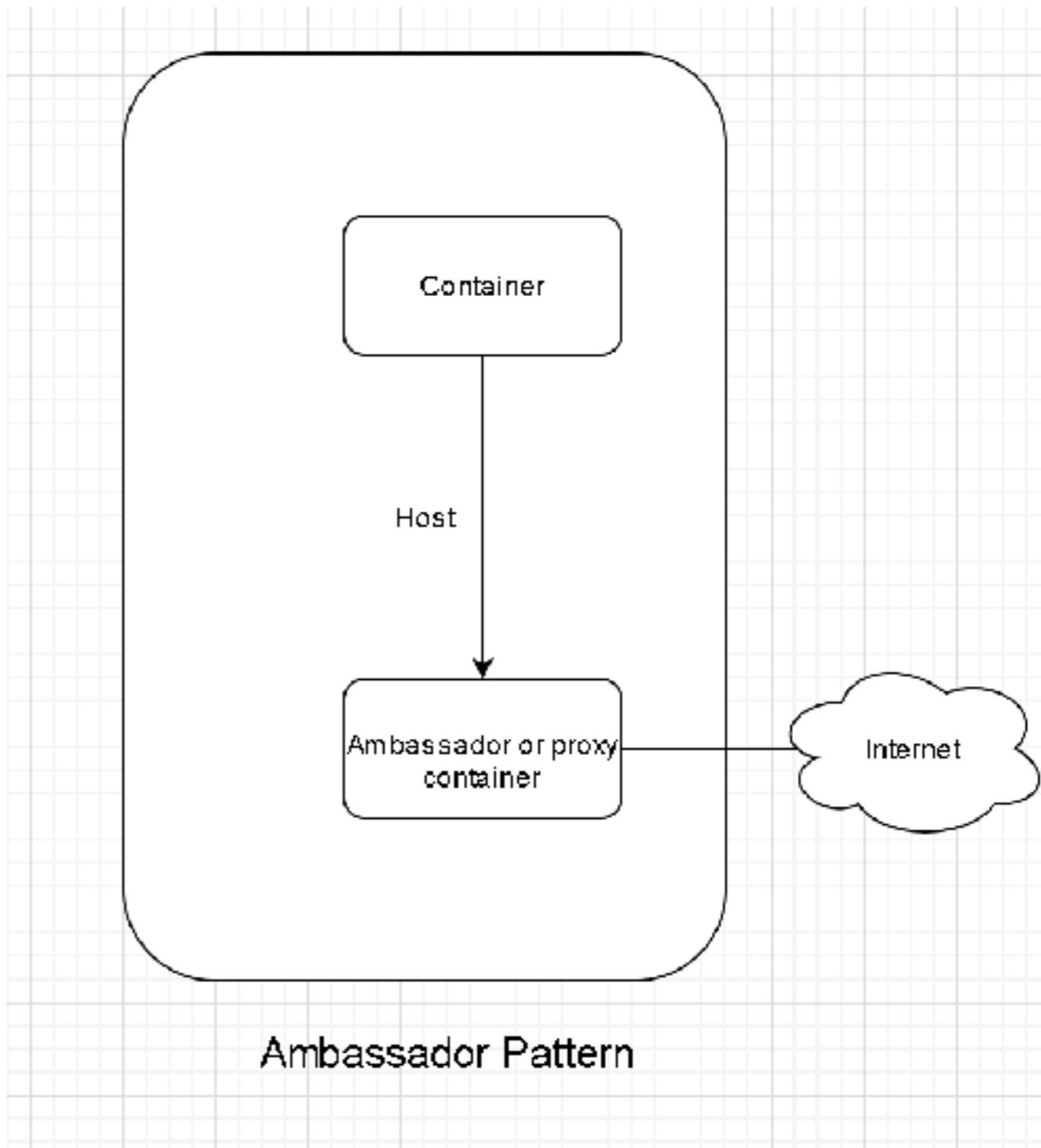


- Because of its proximity to the primary application, there's no significant latency when communicating between them.

Ambassador



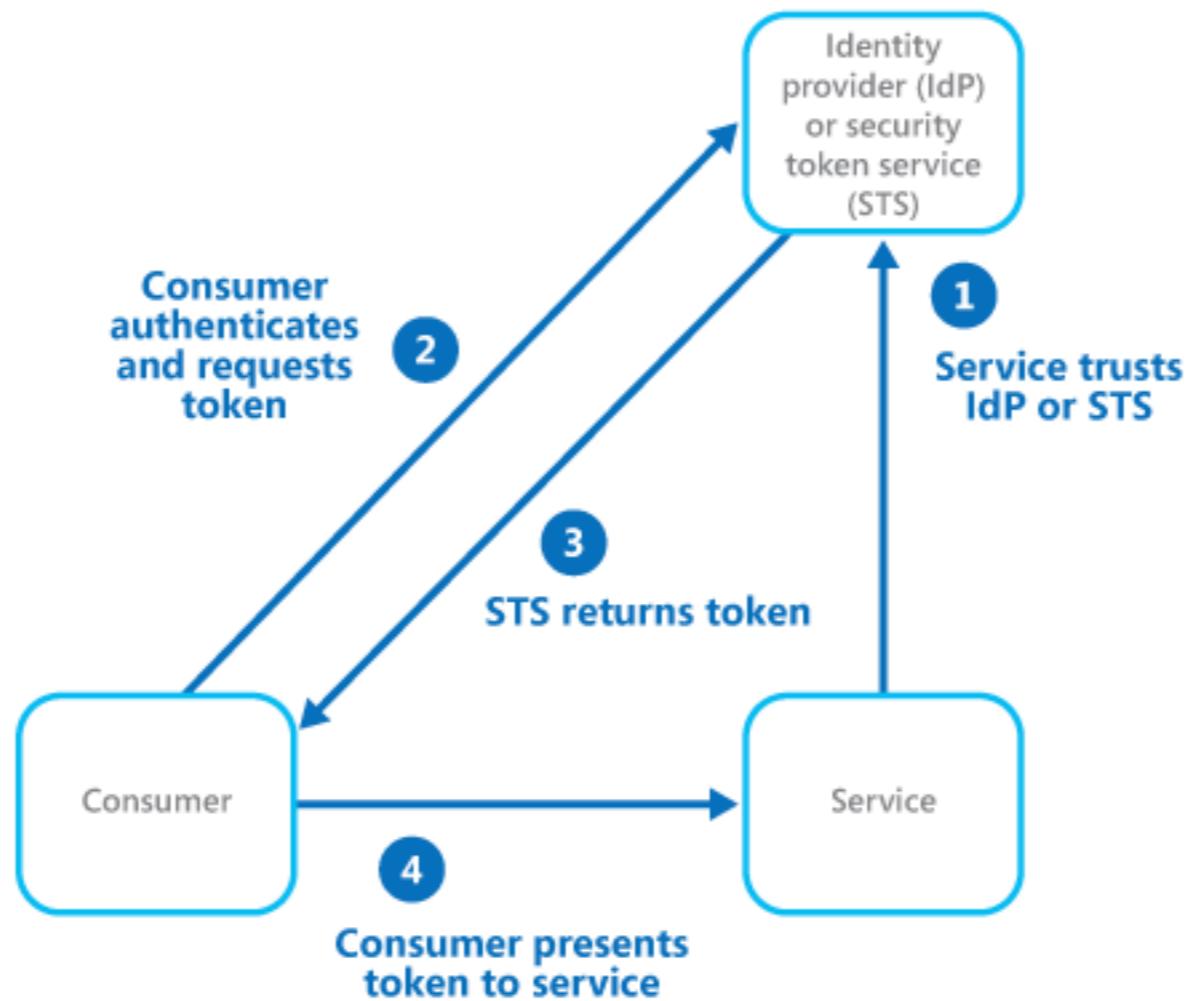
SideCar Pattern



Ambassador Pattern

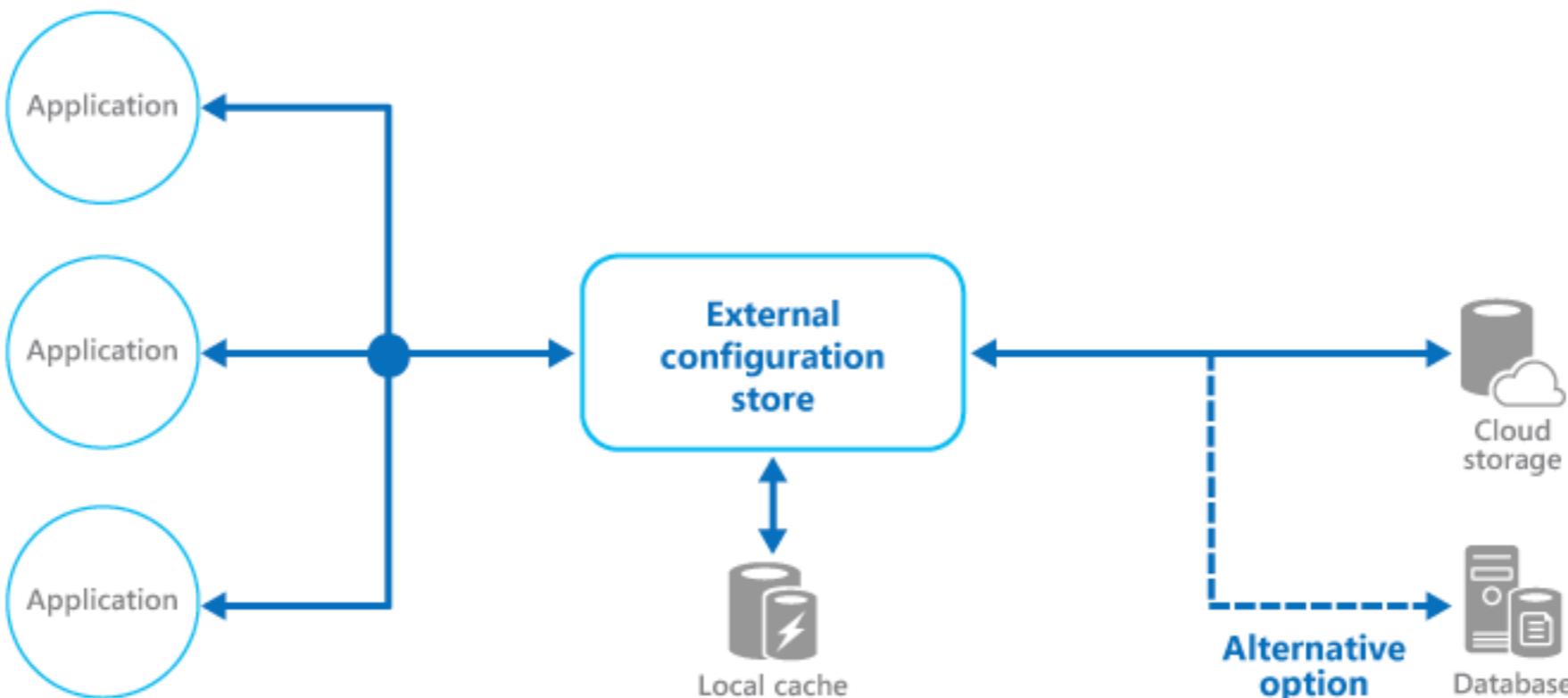
This pattern is almost the same as sidecar pattern the difference is that in ambassador pattern every interaction to the world goes through the ambassador container.

Federated Identity



Delegate authentication to an external identity provider.

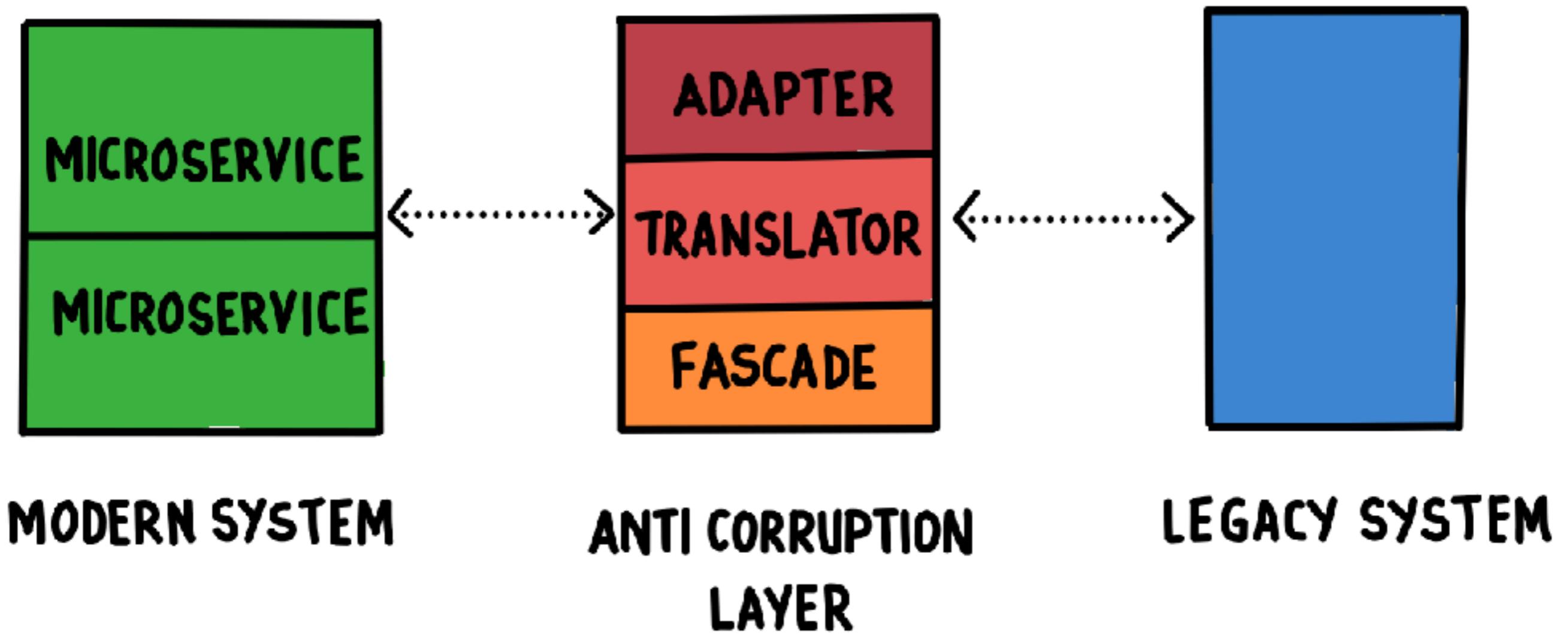
External Configuration Store



Move configuration information out of the application deployment package to a centralized location.

Legacy modernisation

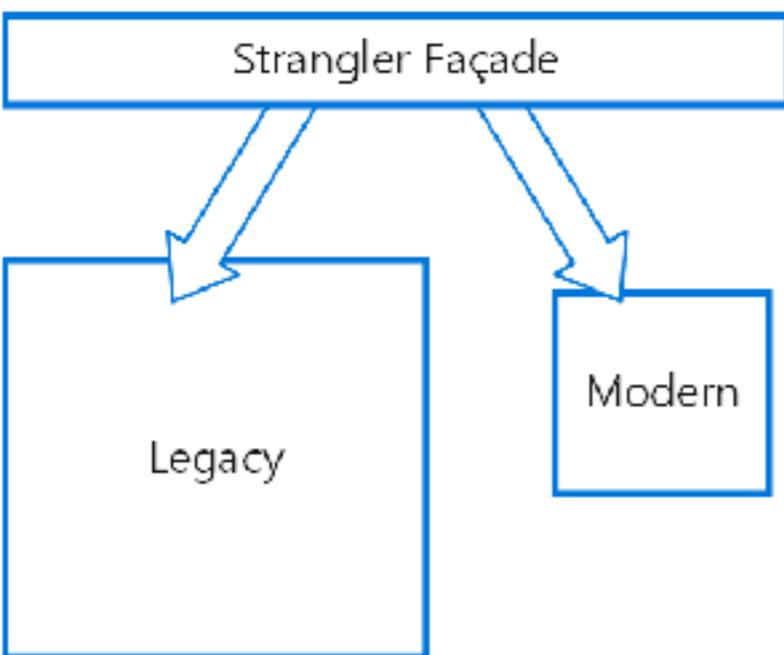
Anti-corruption Layer



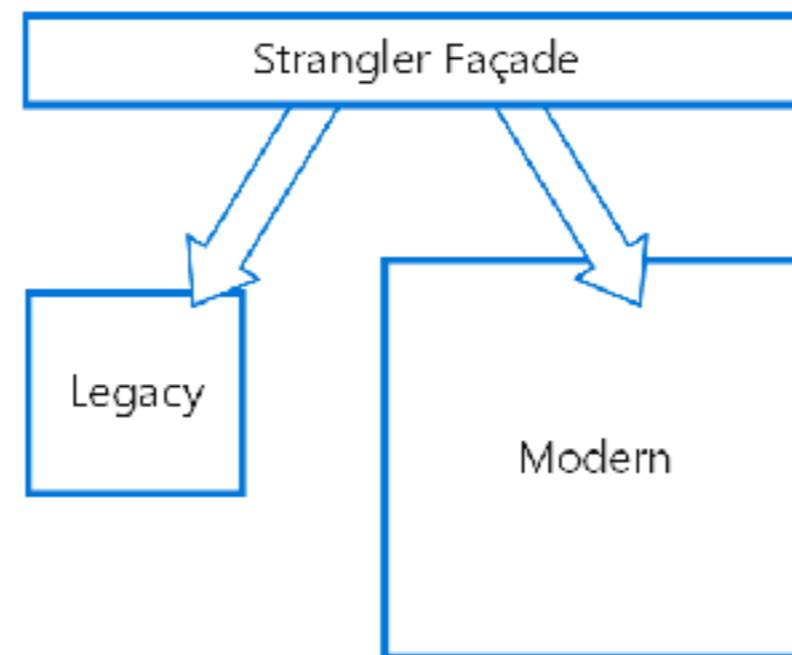
- How do you prevent a legacy monolith's domain model from polluting the domain model of a new service.
- The Anti Corruption Layer uses a combination of Cascade, Adapter, and Translators to isolate the system

Strangler Fig

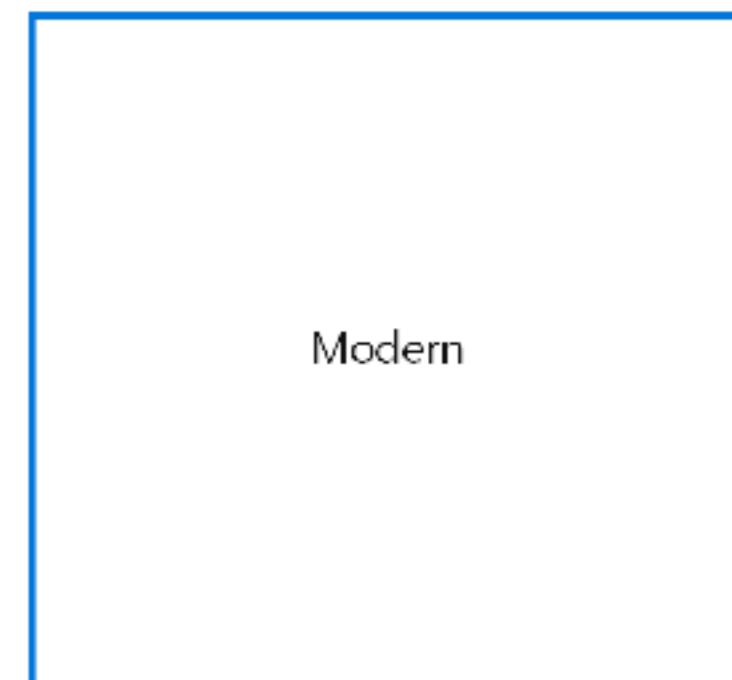
Early migration



Later migration



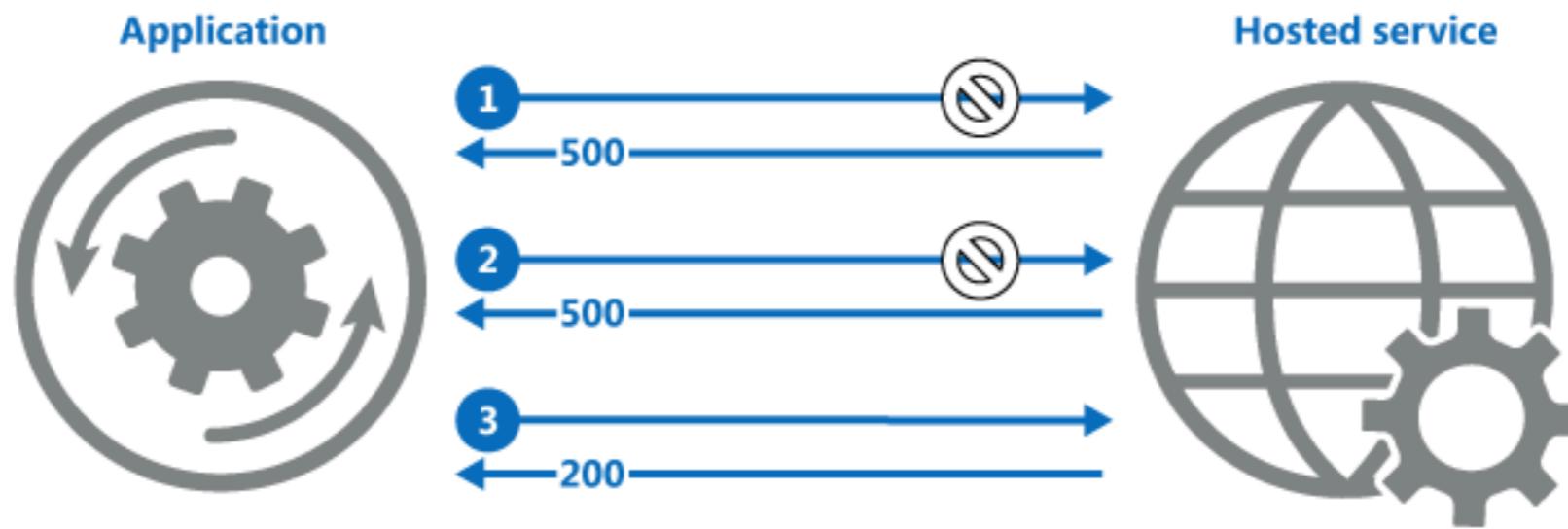
Migration complete



Incrementally replace specific pieces of functionality with new applications and services. Create a façade that intercepts requests going to the backend legacy system. The façade routes these requests either to the legacy application or the new services.

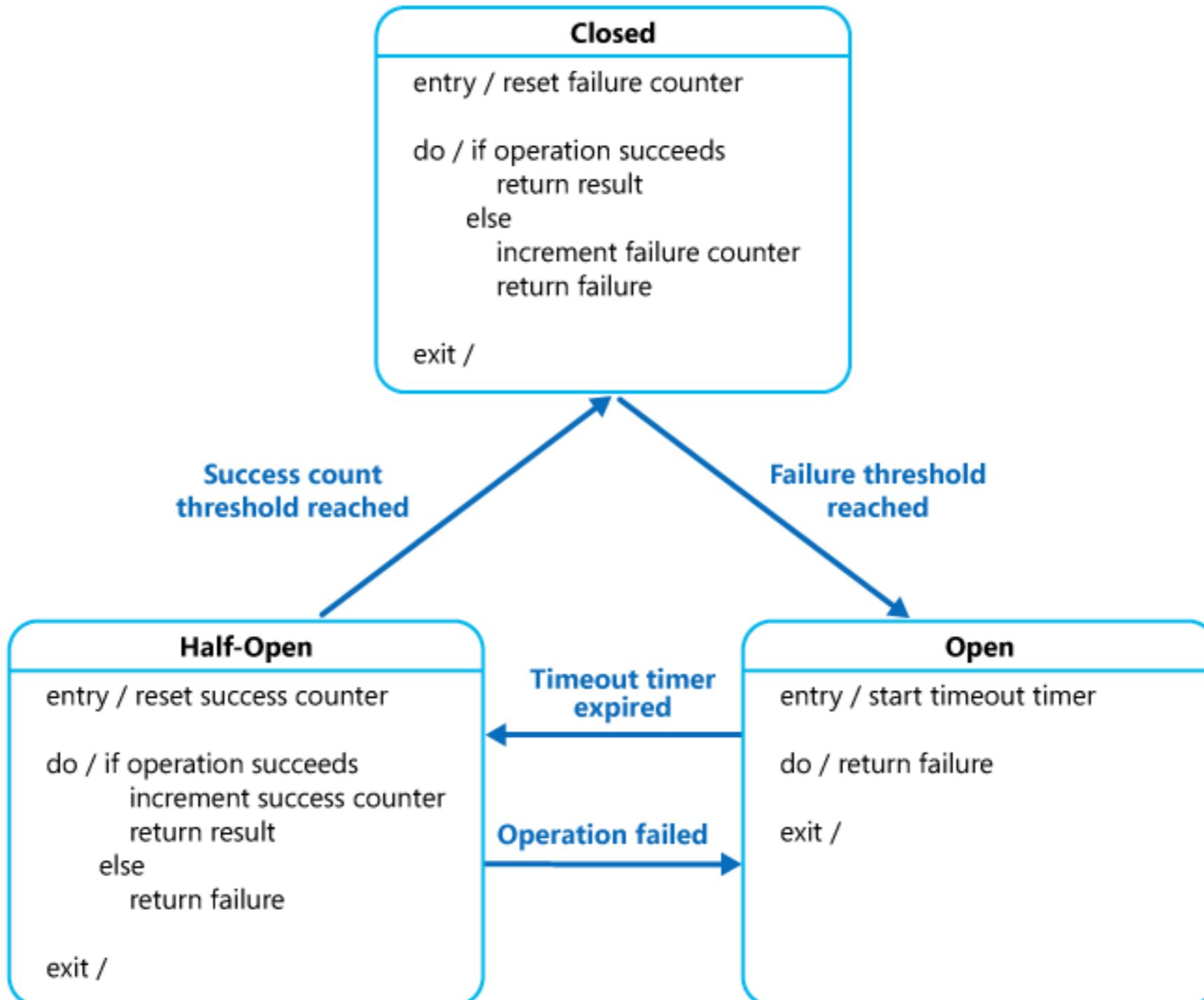
Reliability Patterns

Retry



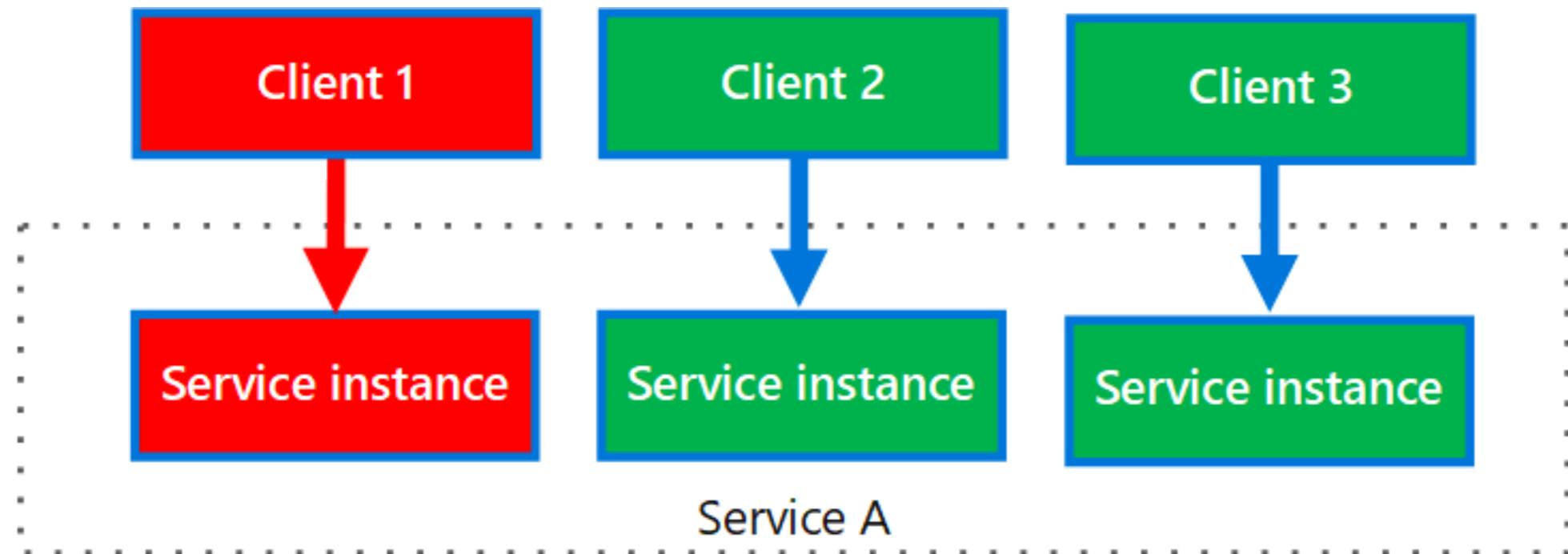
- 1: Application invokes operation on hosted service. The request fails, and the service host responds with HTTP response code 500 (internal server error).
- 2: Application waits for a short interval and tries again. The request still fails with HTTP response code 500.
- 3: Application waits for a longer interval and tries again. The request succeeds with HTTP response code 200 (OK).

Circuit Breaker



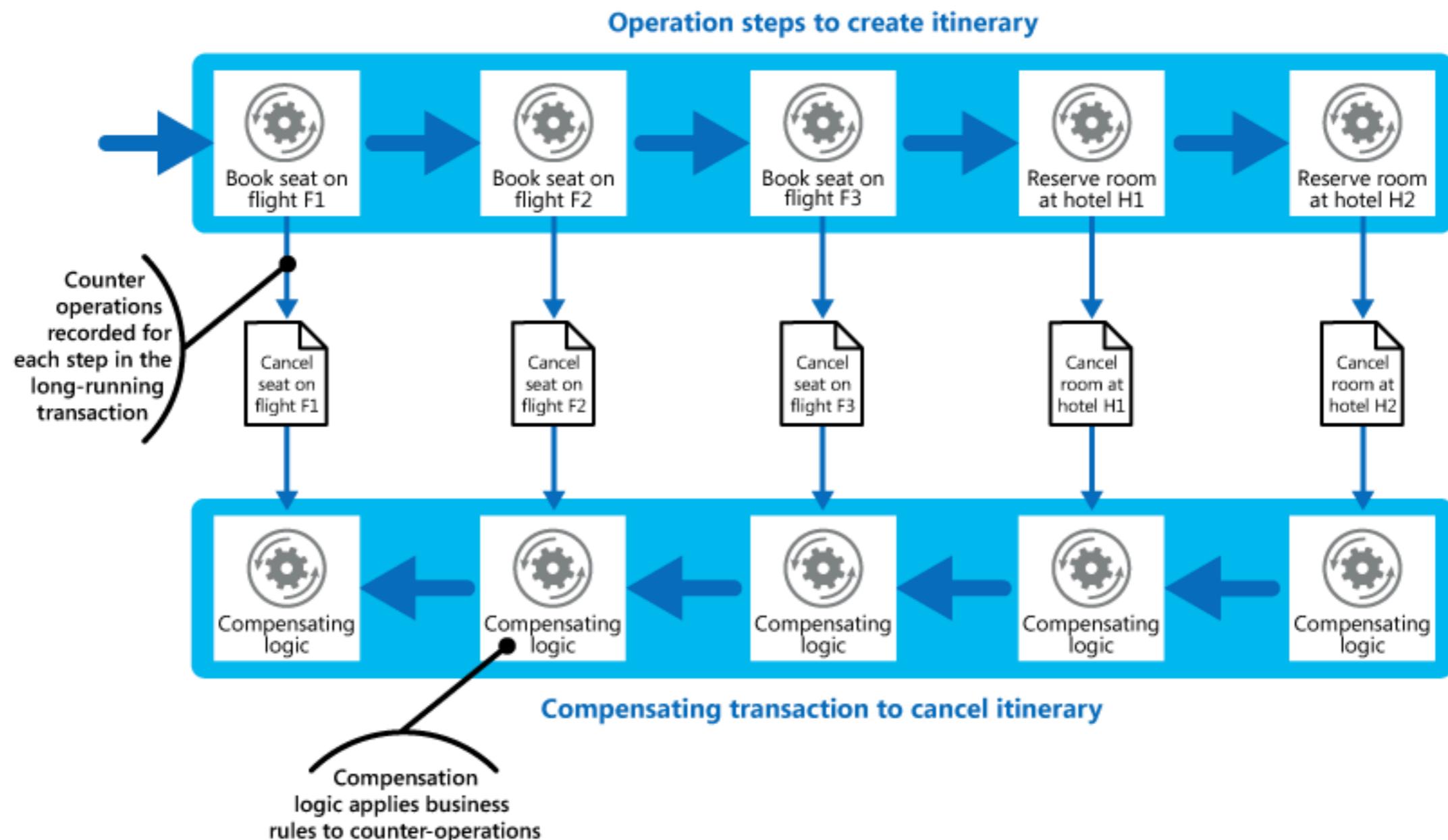
Fail Fast. This can improve the stability and resiliency of an application.

Bulkhead



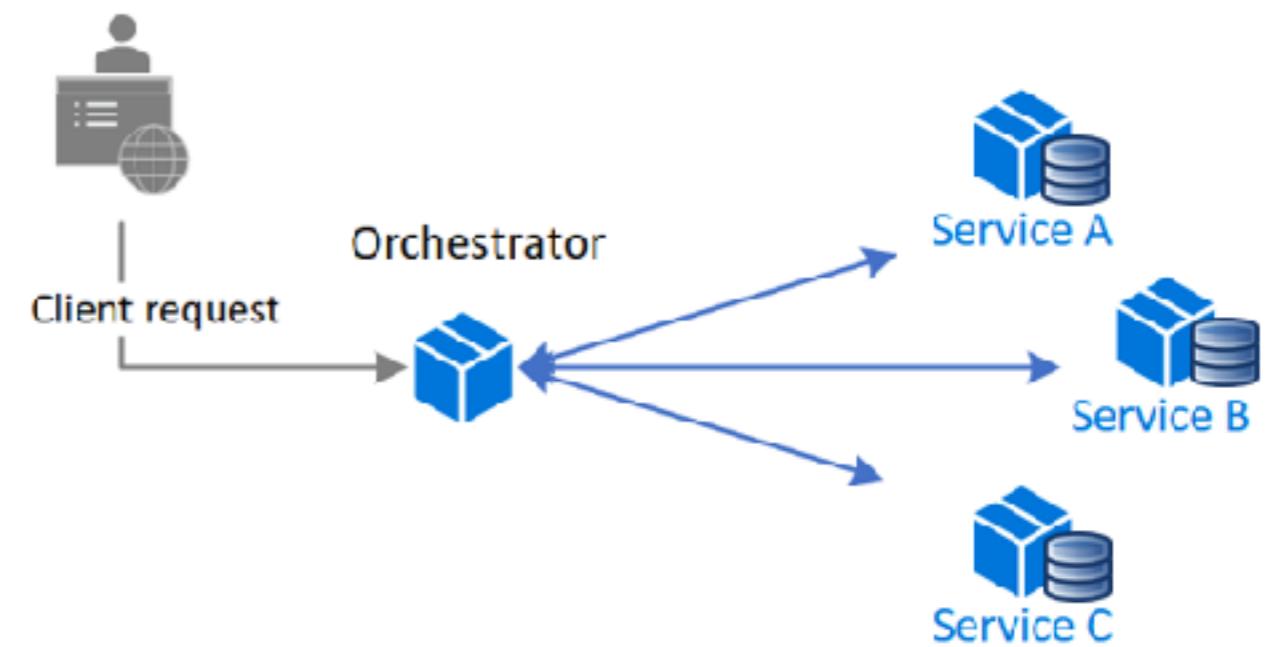
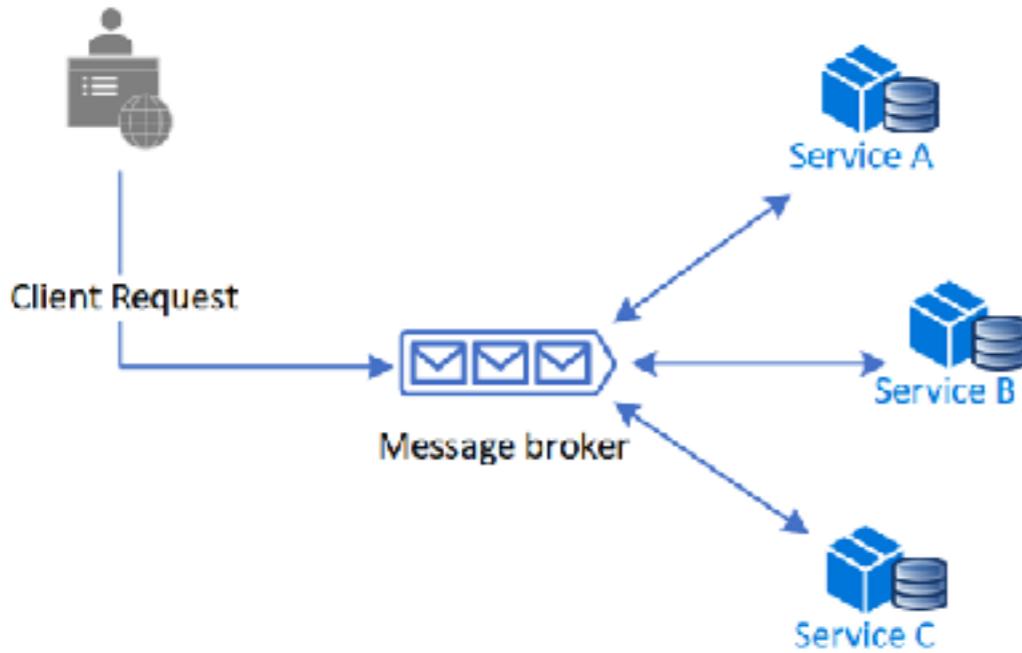
Each tenant is assigned a separate service instance. Tenant 1 has made too many requests and overwhelmed its instance.

Compensating Transaction



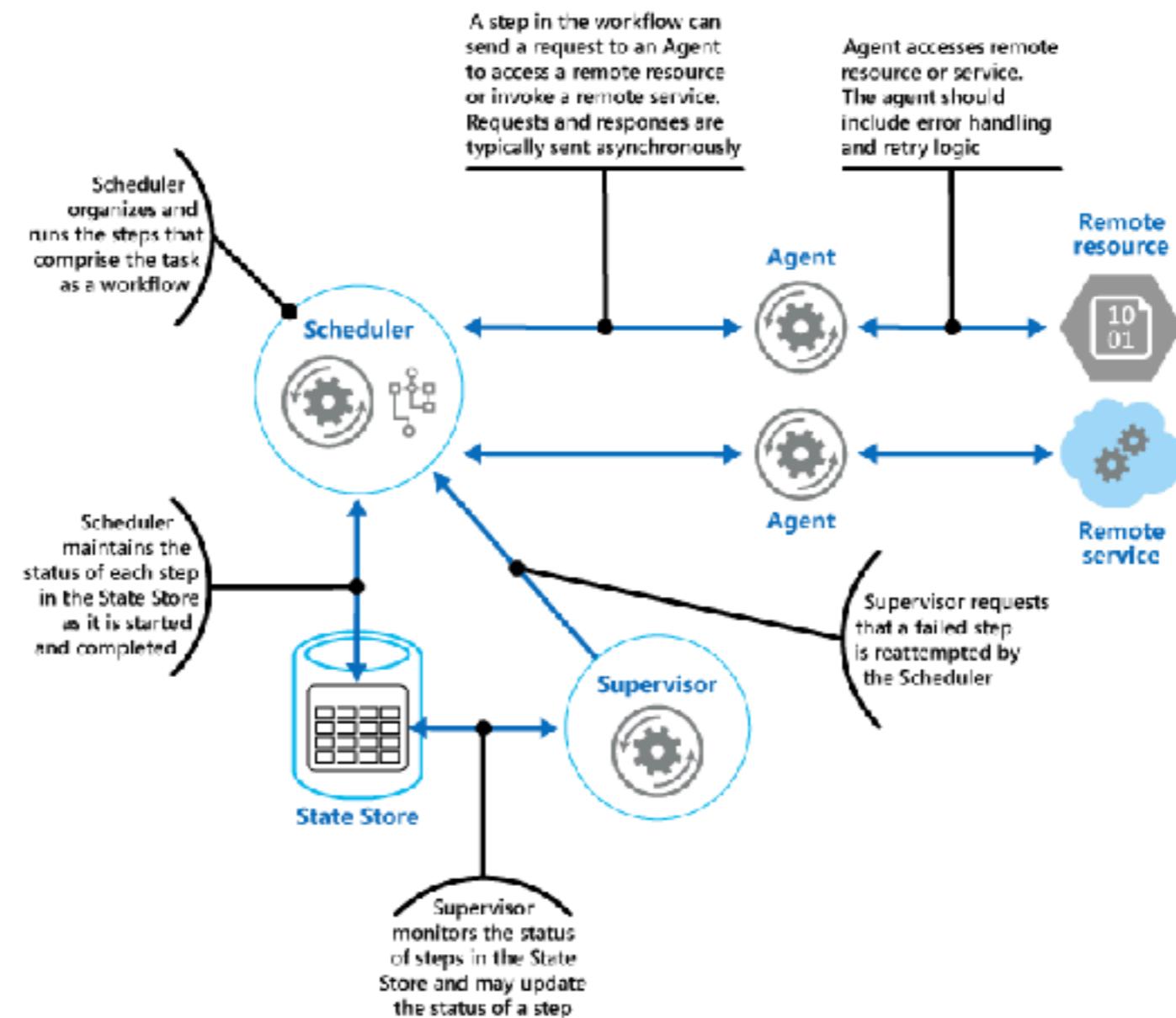
Undo the work performed by a series of steps, which together define an eventually consistent operation, if one or more of the steps fail.

Saga distributed transactions



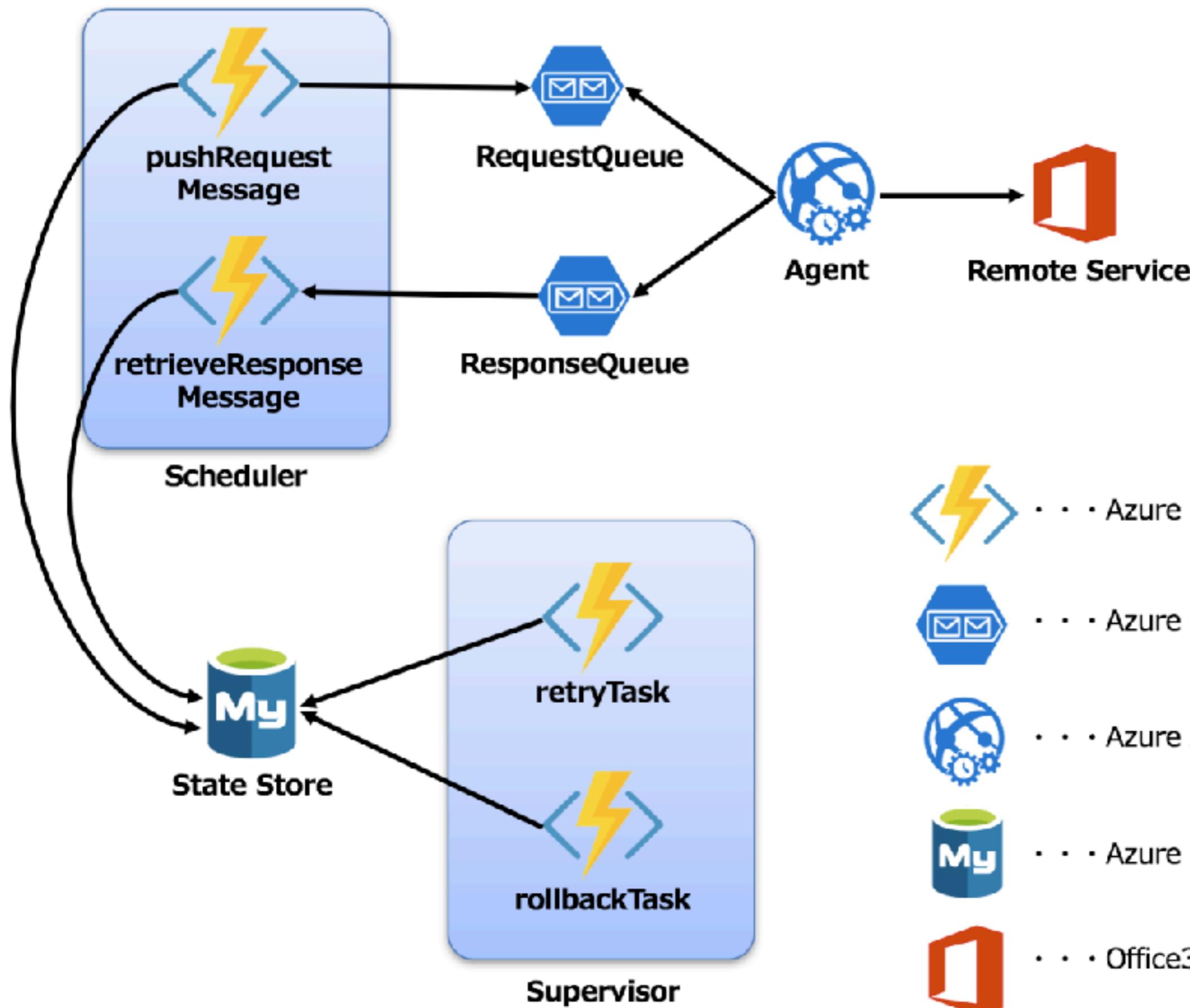
The Saga pattern provides transaction management using a sequence of *local transactions*. A local transaction is the atomic work effort performed by a saga participant. Each local transaction updates the database and publishes a message or event to trigger the next local transaction in the saga. If a local transaction fails, the saga executes a series of *compensating transactions* that undo the changes that were made by the preceding local transactions.

Scheduler Agent Supervisor



Access to remote services is subject to network congestion, overload on the remote service side, occurrence of throttling, and suspension of data center operations due to disasters such as earthquakes and fires. It fails for various reasons. **The Retry pattern** can be used to recover in a short period of time, but if a persistent error that cannot be easily recovered occurs, one of the following measures is required.

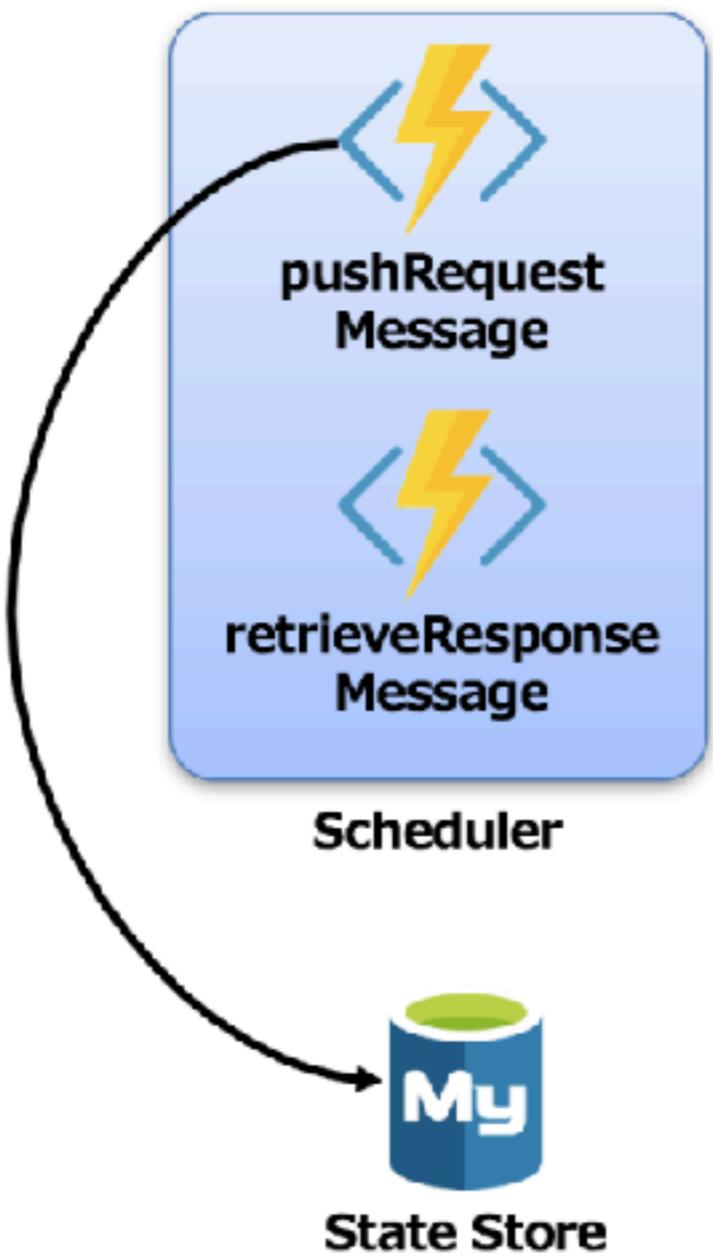
- Retry processing to the remote service.
- Rollback any changes to remote services to keep the system in a consistent state.
- Notify the administrator by email, and the administrator manually recovers.



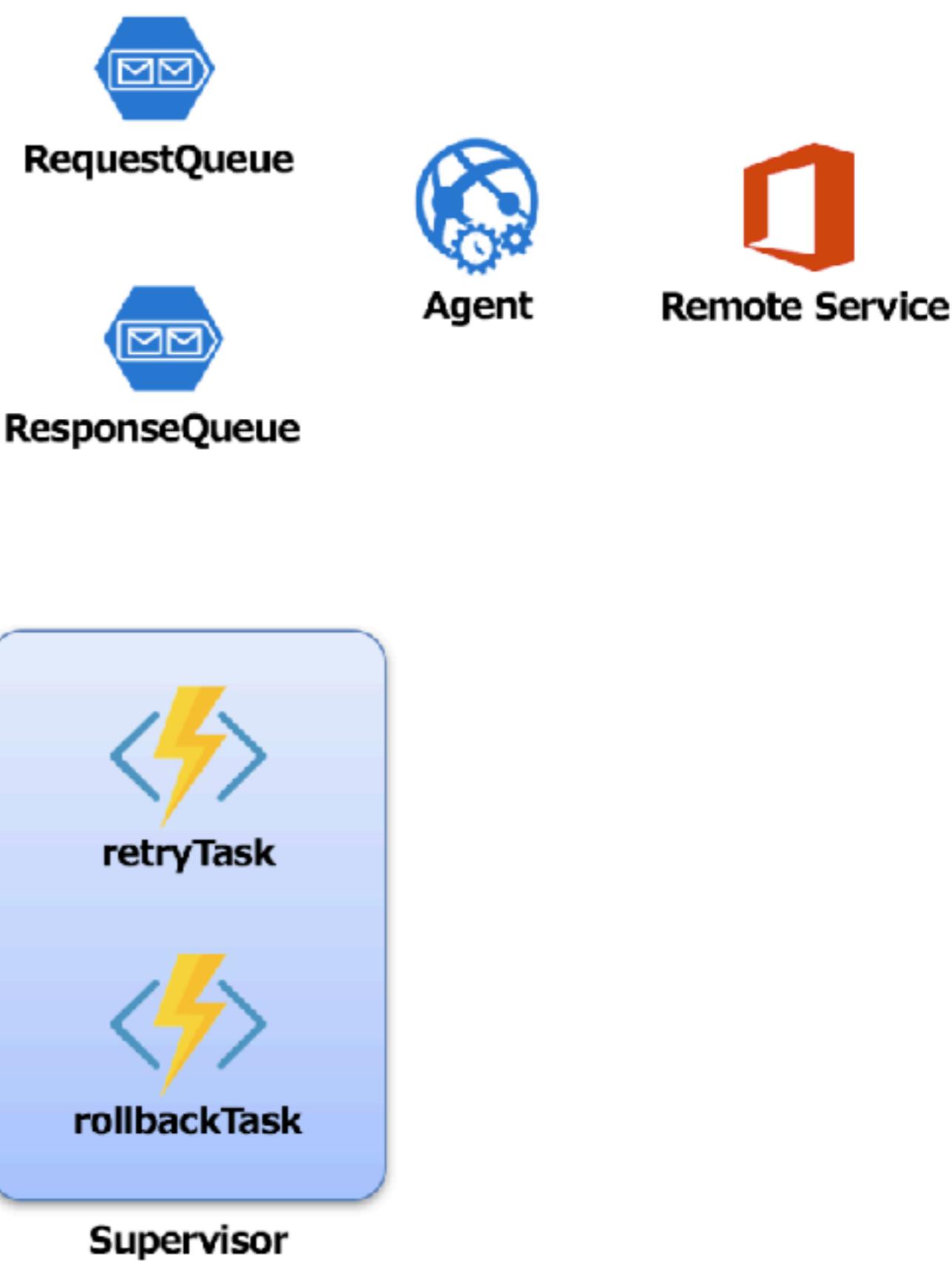
- • • Azure Functions
- • • Azure Queue Storage
- • • Azure App Service WebJobs
- • • Azure Database for MySQL
- • • Office365

Steps

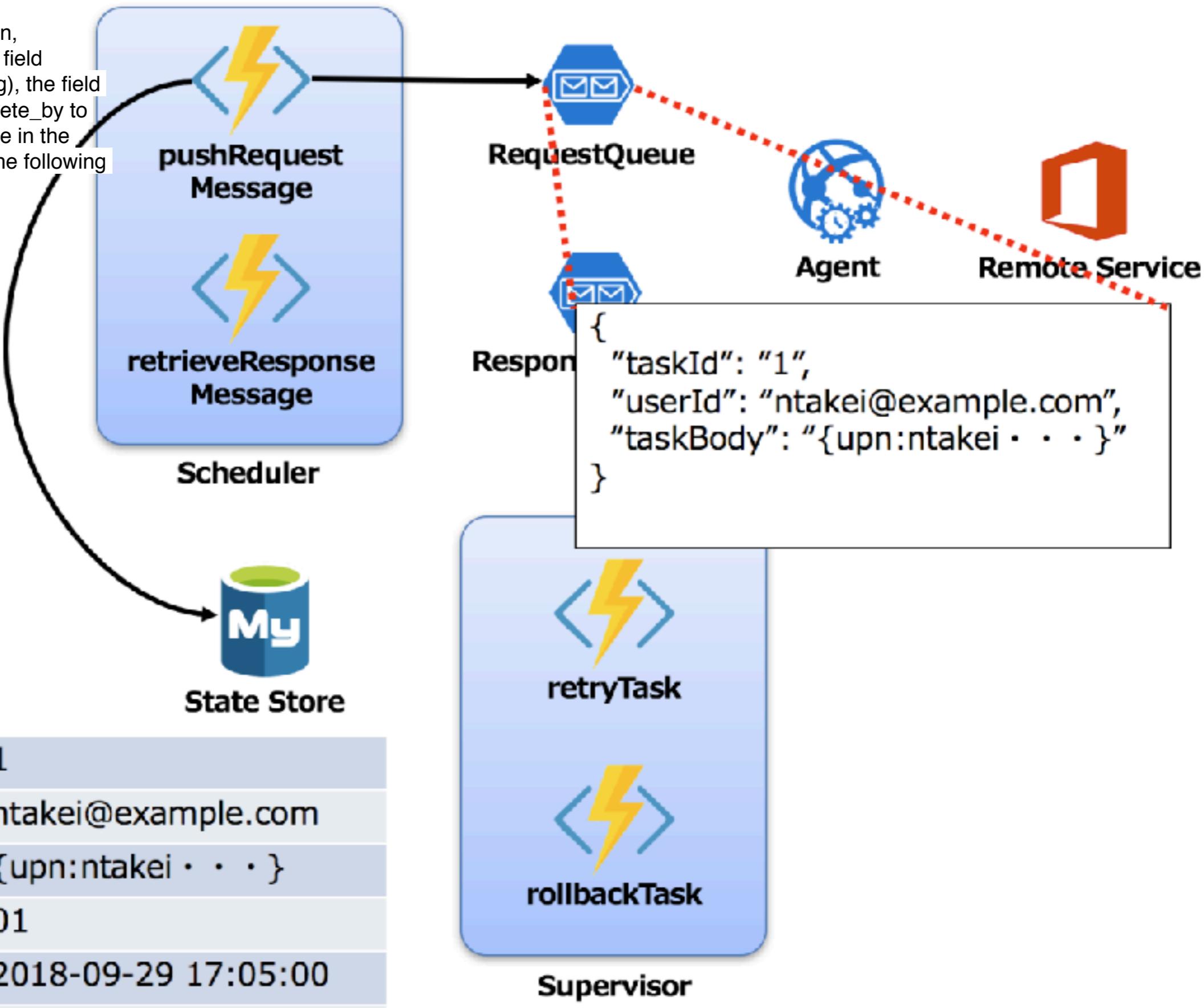
The Azure Function pushRequestMessage, which acts as a Scheduler, retrieves a task with field process_state of 00 (unprocessed) and locked_by of NULL in table t_state_store.



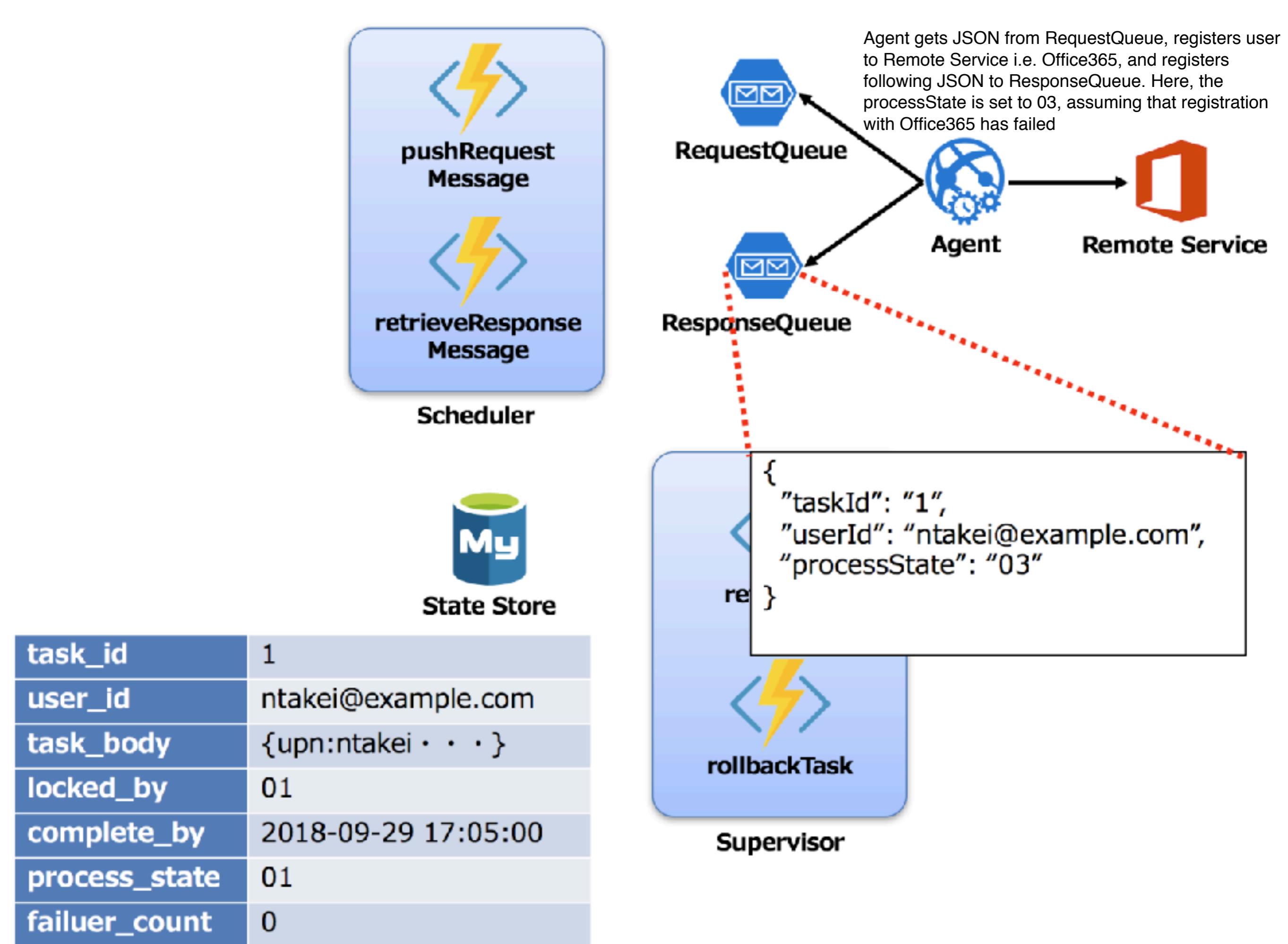
task_id	1
user_id	ntakei@example.com
task_body	{upn:ntakei · · · }
locked_by	NULL
complete_by	NULL
process_state	00
failure_count	0

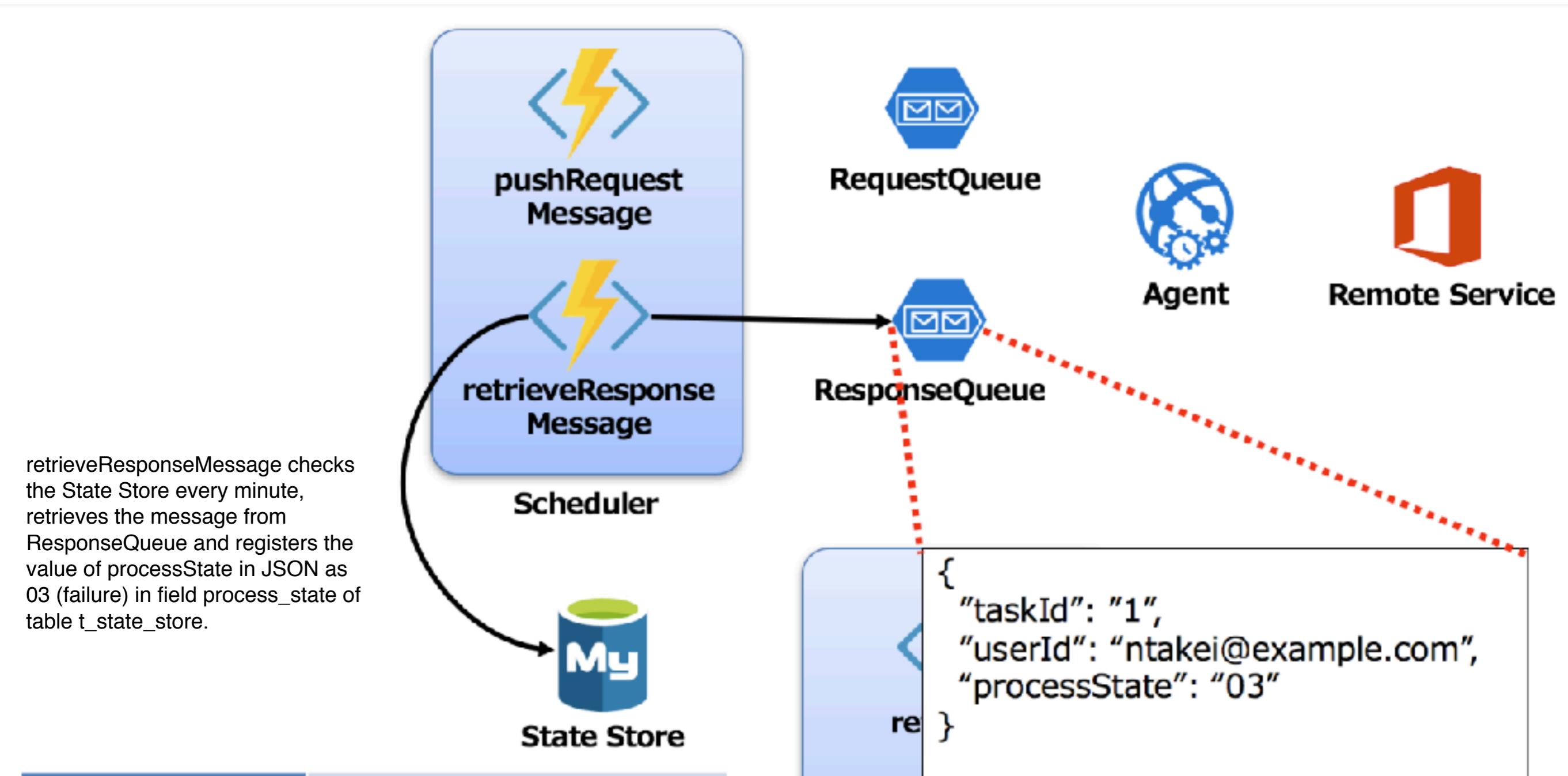


After getting the task information, pushRequestMessage sets the field process_state to 01 (processing), the field locked_by to 01, and the complete_by to 10 minutes after the current time in the table t_state_store. And push the following JSON to RequestQueue.



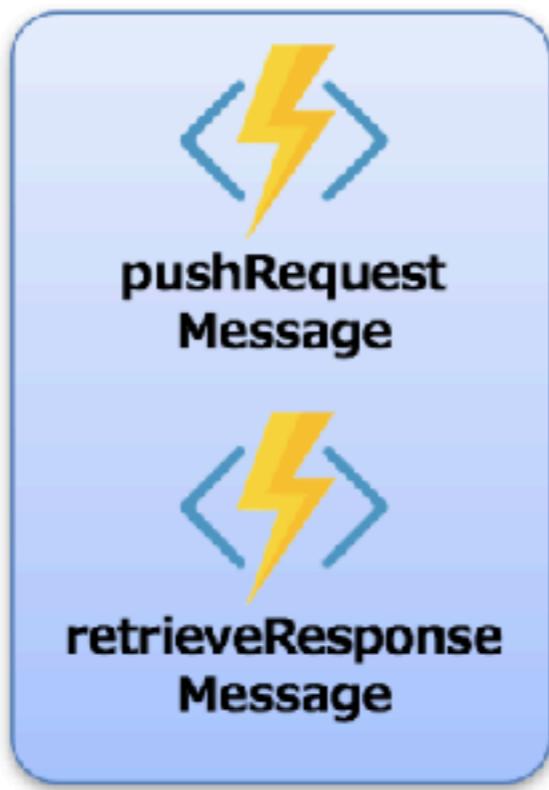
task_id	1
user_id	ntakei@example.com
task_body	{upn:ntakei · · · }
locked_by	01
complete_by	2018-09-29 17:05:00
process_state	01
failuer_count	0



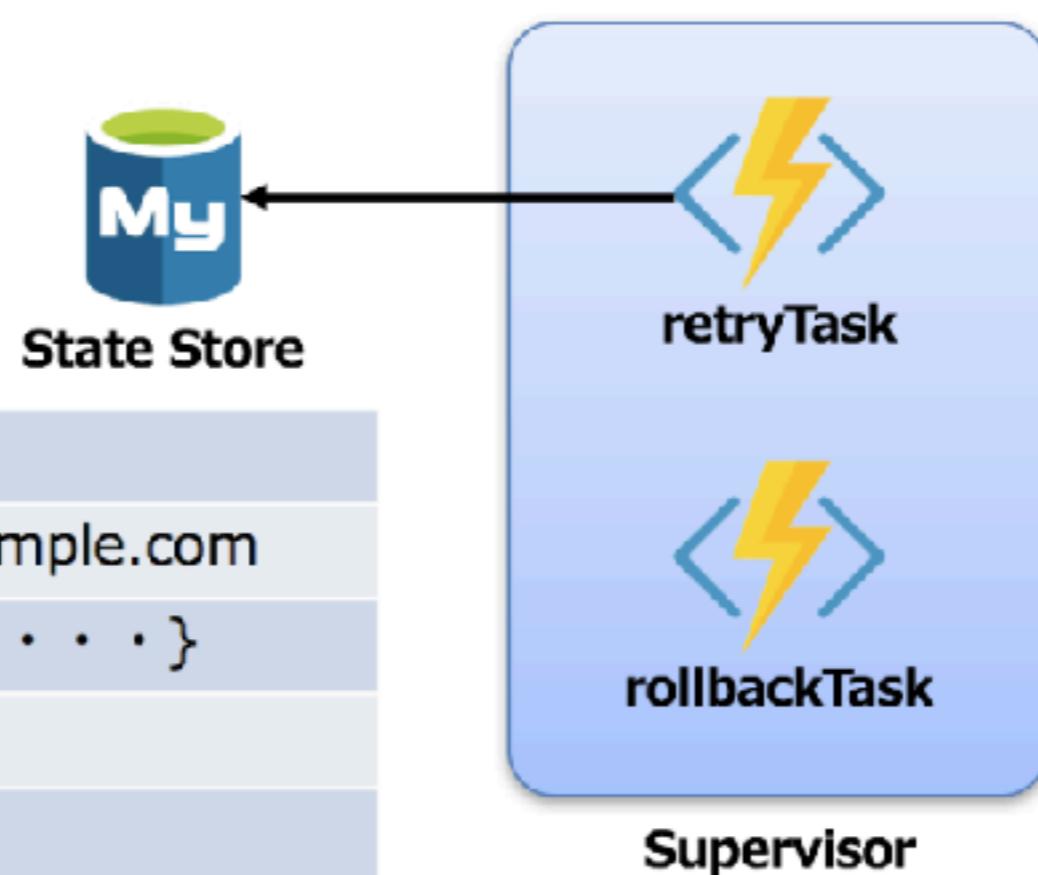


retrieveResponseMessage checks the State Store every minute, retrieves the message from ResponseQueue and registers the value of processState in JSON as 03 (failure) in field process_state of table t_state_store.

task_id	1
user_id	ntakei@example.com
task_body	{upn:ntakei · · · }
locked_by	01
complete_by	2018-09-29 17:05:00
process_state	03
failuer_count	0



retryTask checks the State Store every minute and updates tasks with a process_state of 03 (failed) as follows: Then, the scheduler extracts the following records and requests processing to the agent again. In other words, it is a retry.



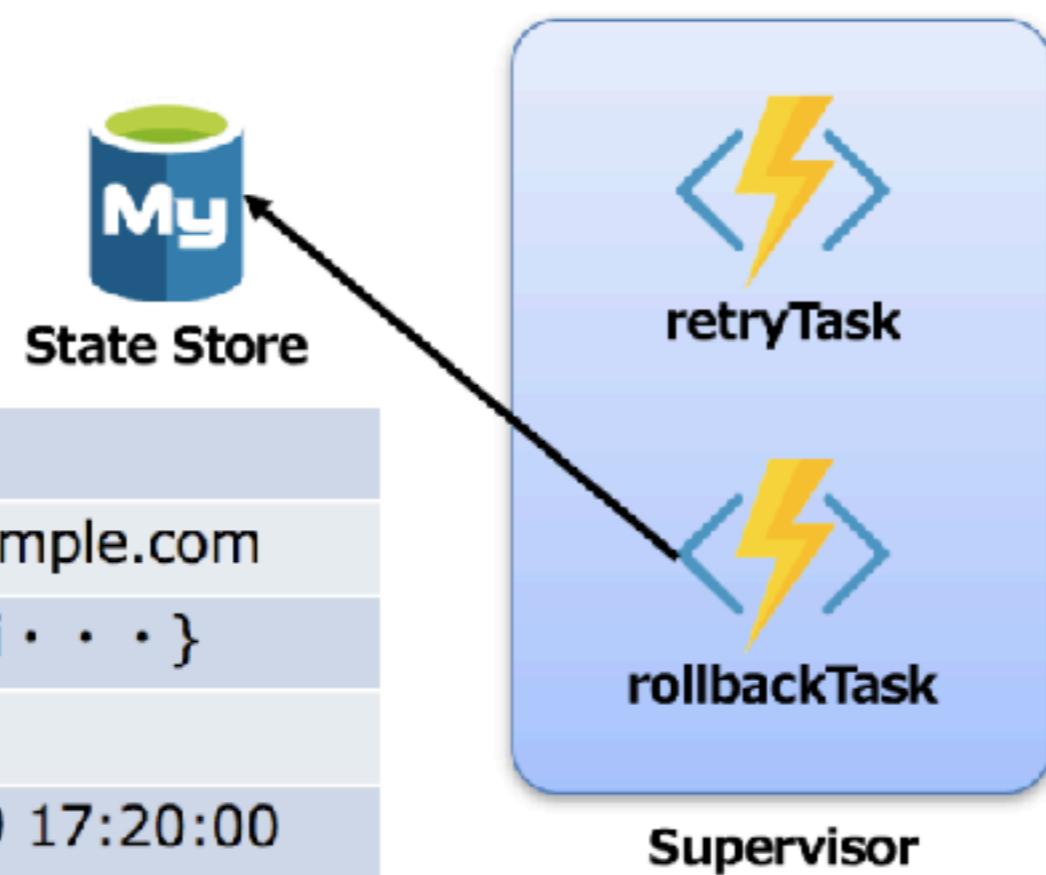
task_id	1
user_id	ntakei@example.com
task_body	{upn:ntakei . . . }
locked_by	NULL
complete_by	NULL
process_state	01
failuer_count	1



Scheduler



Then, let's say that we repeat the retry and finally fail three times. Supervisor's rollbackTask function determines that this is an unusual failure in Office365, gives up retry, rolls back, or notifies the administrator by e-mail.



Supervisor

task_id	1
user_id	ntakei@example.com
task_body	{upn:ntakei . . . }
locked_by	01
complete_by	2018-09-29 17:20:00
process_state	03
failuer_count	3

(1) First, register the contents (user name, password, etc.) of the process to be registered with the Remote Service (a service on the cloud such as Office365) in the State Store. State store implementation can be file, relational database, NoSQL database, or anything as long as data can be persisted. In addition, the following information is registered.

- Execution state of the task (open, in progress, successful, failed)
- Estimated completion time of the task
- Task failure count

(2) The Scheduler periodically checks the State Store, extracts tasks whose execution status is "Unprocessed", and requests the Agent to execute the task. At this time, change the State Store as follows.

- Task execution status: Processing
- Estimated task completion time: current time + task processing timeout time

(3) Upon receiving a request from Scheduler, Agent accesses Remote Service and executes processing.

(4) Agent returns the result of processing for Remote Service to Scheduler.

(5) Scheduler receives execution results from Agent and changes State Store as follows.

- Task execution status: success or failure

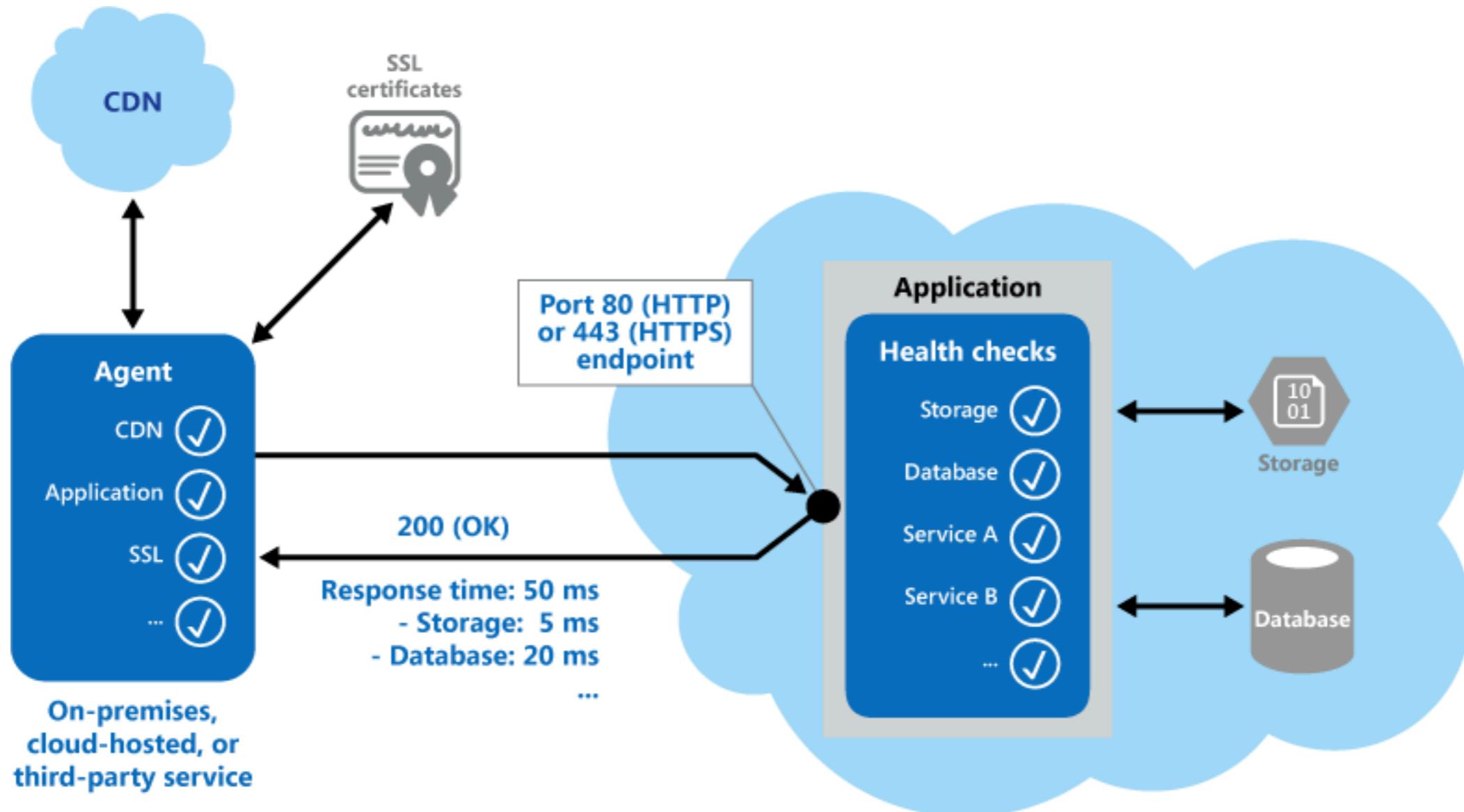
(6) Supervisor periodically checks the State Store and performs the following processing.

- If the number of task failures is within a specified number of times and the task execution status is failed, or if the scheduled completion time of the task has passed, the task execution status is changed to unprocessed. (In other words, let the Agent process again)
- If the number of task failures exceeds the specified number of times, recovery by retrying is impossible, notify by email, and the administrator manually corrects, rolls back each service, etc. Appropriate processing is performed according to the requirement

The State Store has a table called t_state_store with the following fields:

task_id	ID that uniquely identifies the task (Auto Increment)
user_id	ID of the user for whom the task is to be executed
task_body	Information necessary for remote service processing (UPN of Office365, etc.)
locked_by	Instance ID of the scheduler that performs the process (It is assumed that there are multiple schedulers, but in this blog there is only one, so 01 is registered as a lock)
complete_by	Estimated time the task will be completed
process_state	Task execution status (00: Not processed, 01: Processing, 02: Succeeded, 03: Failed)
failure_count	Number of times the Agent failed to process a task

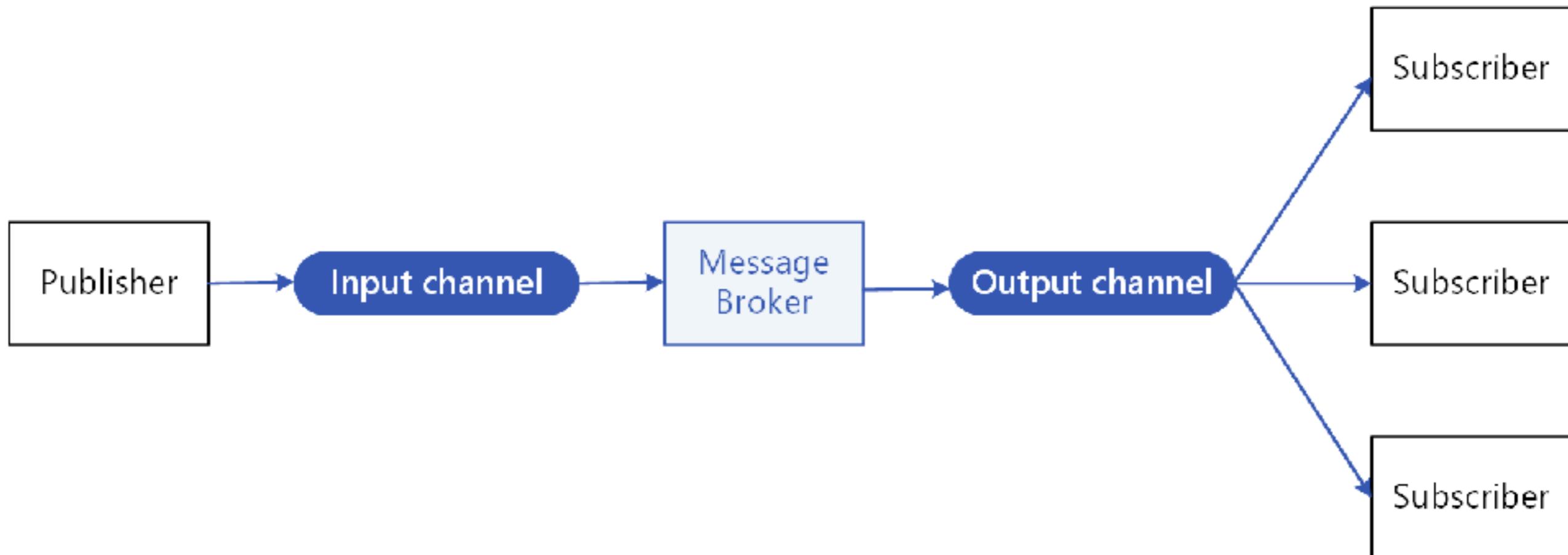
Health Endpoint Monitoring



Implement functional checks in an application that external tools can access through exposed endpoints at regular intervals.

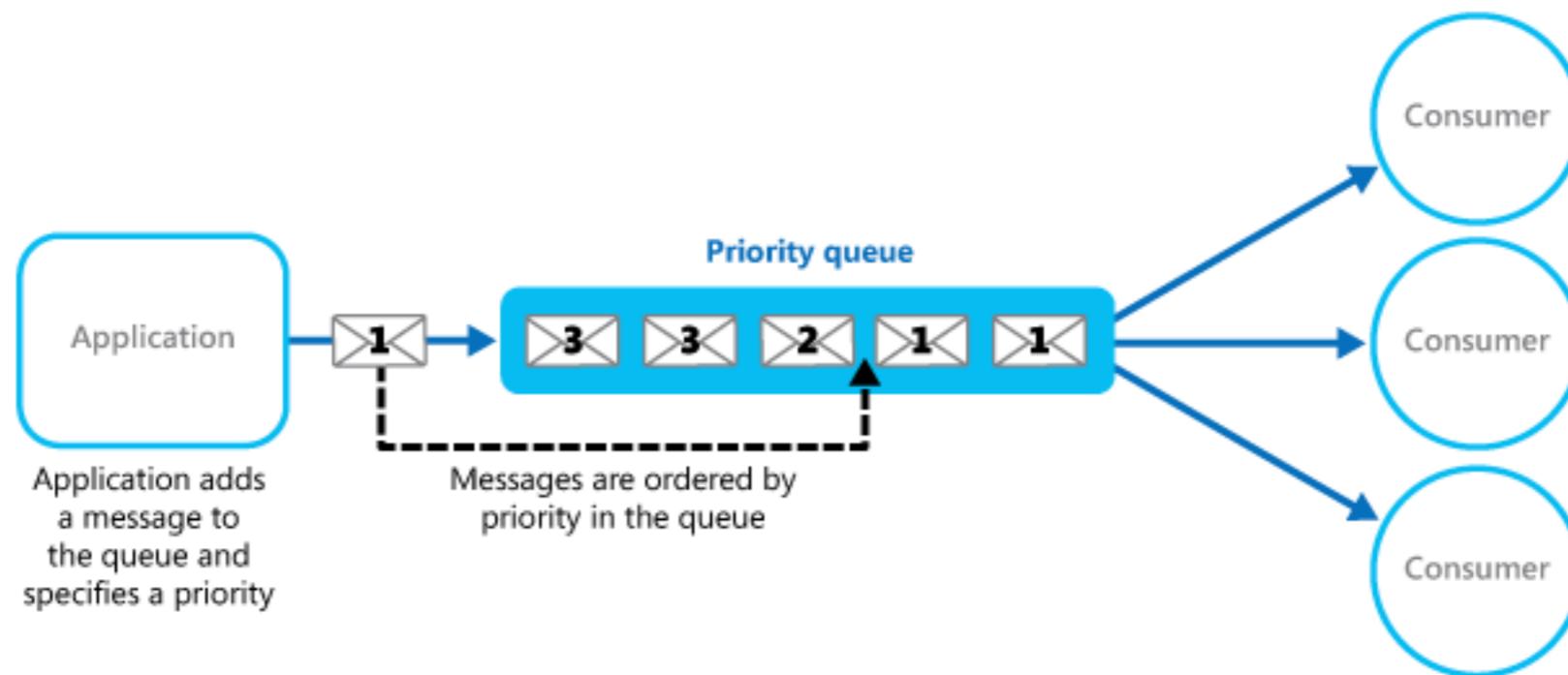
Communication

Publisher-Subscriber



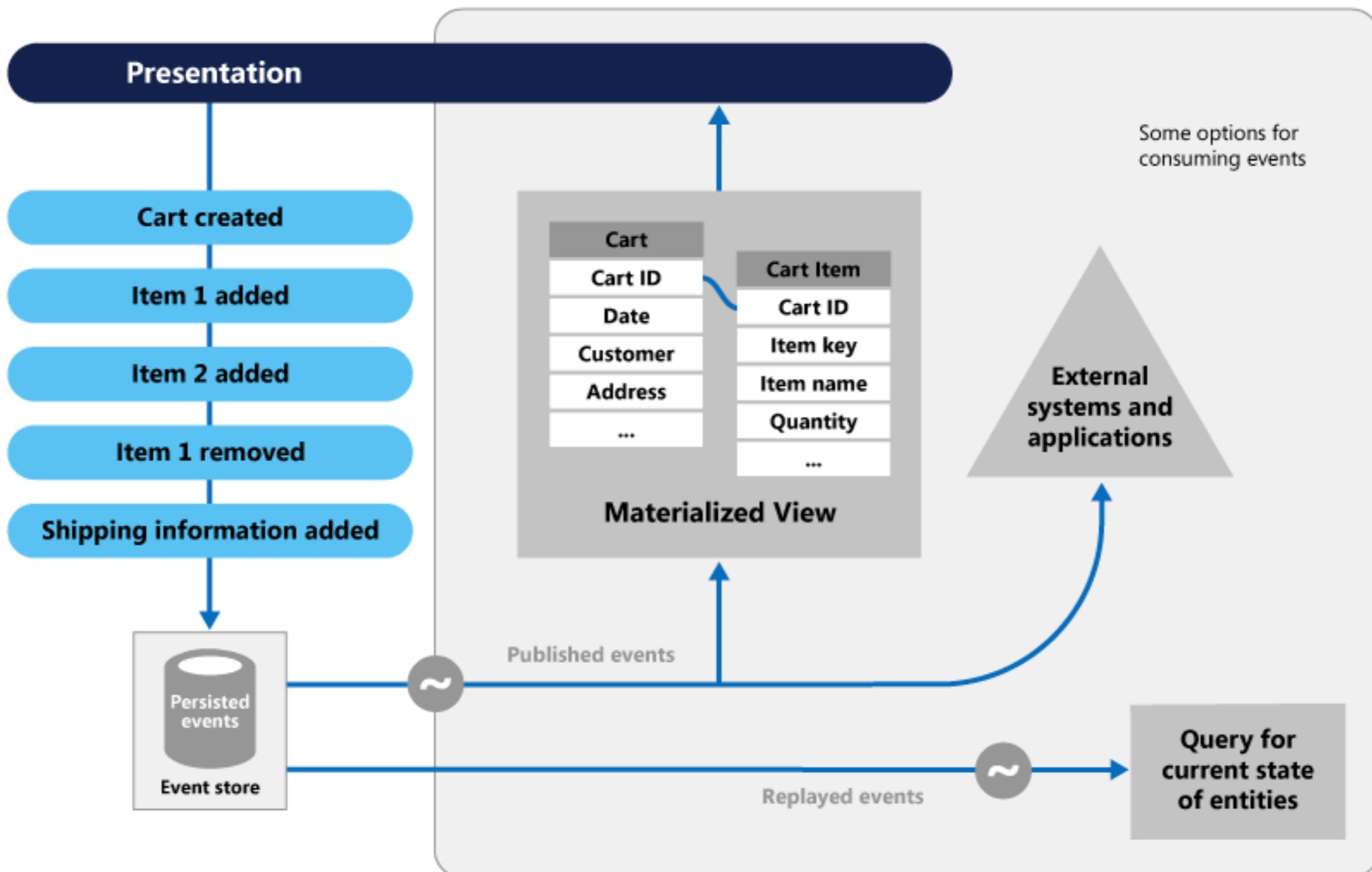
- An application needs to broadcast information to a significant number of consumers.

Priority Queue



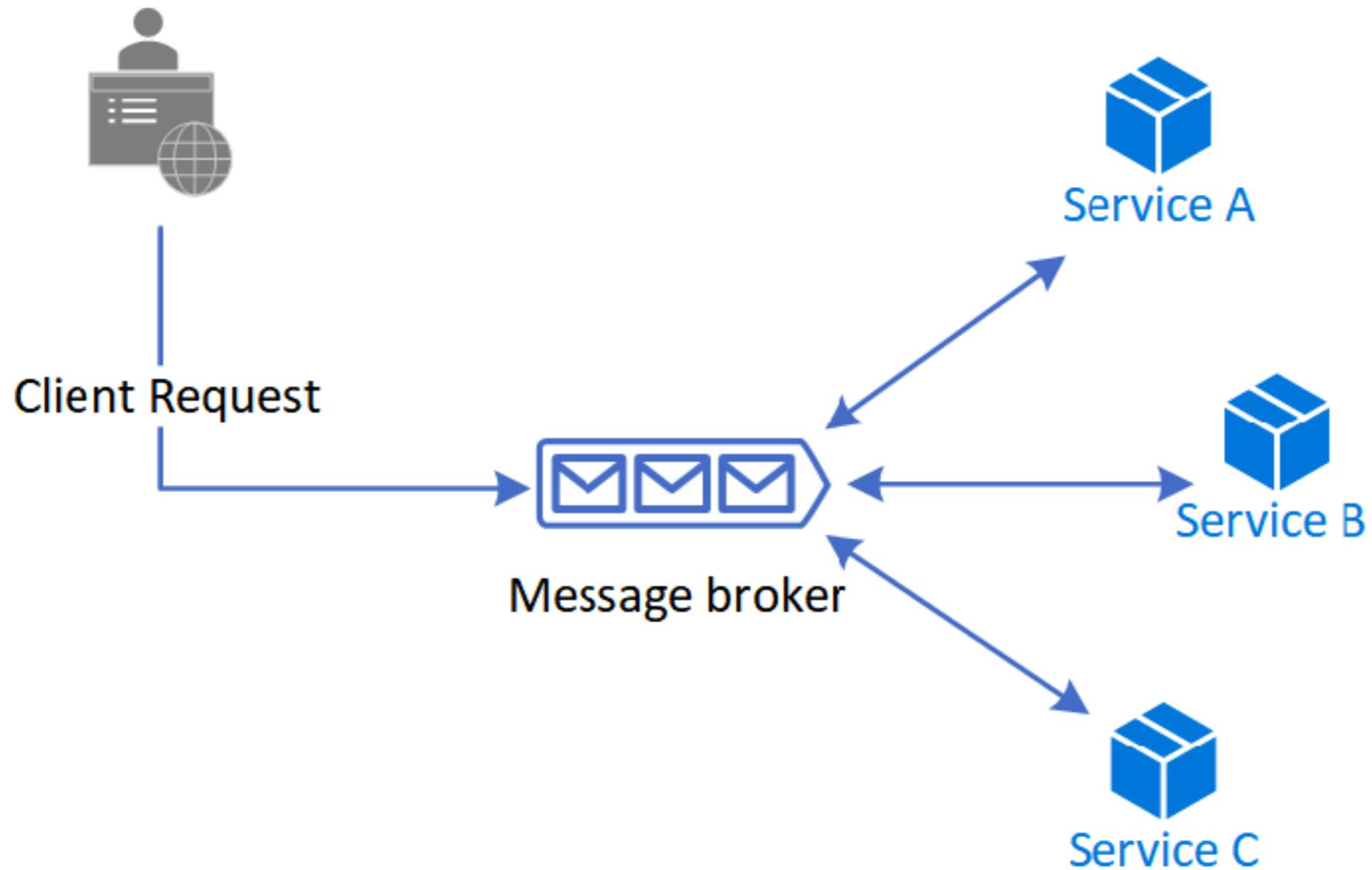
Prioritize requests sent to services so that requests with a higher priority are received and processed more quickly than those with a lower priority.

Event Sourcing



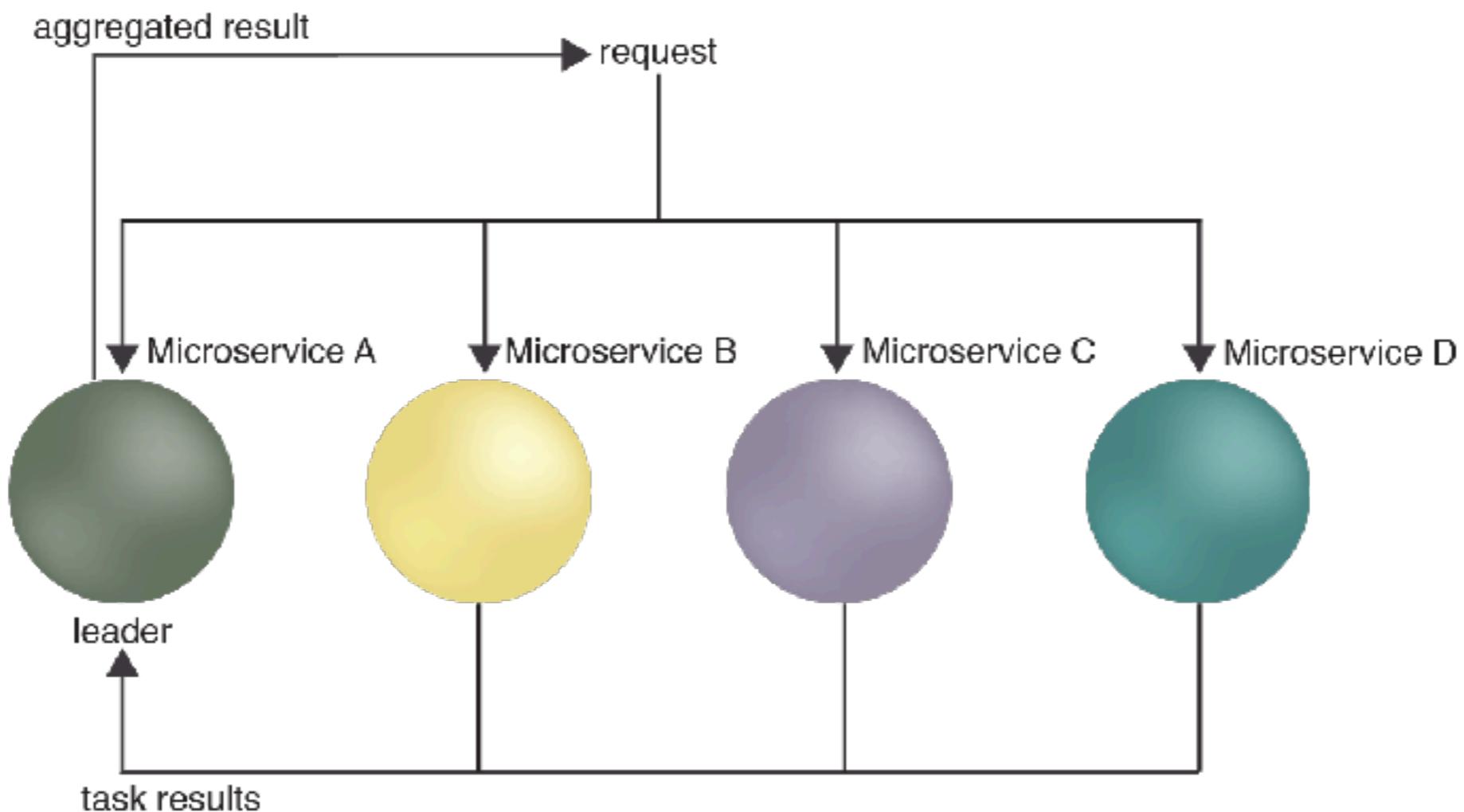
Instead of storing just the current state of the data in a domain, use an append-only store to record the full series of actions taken on that data.

Choreography



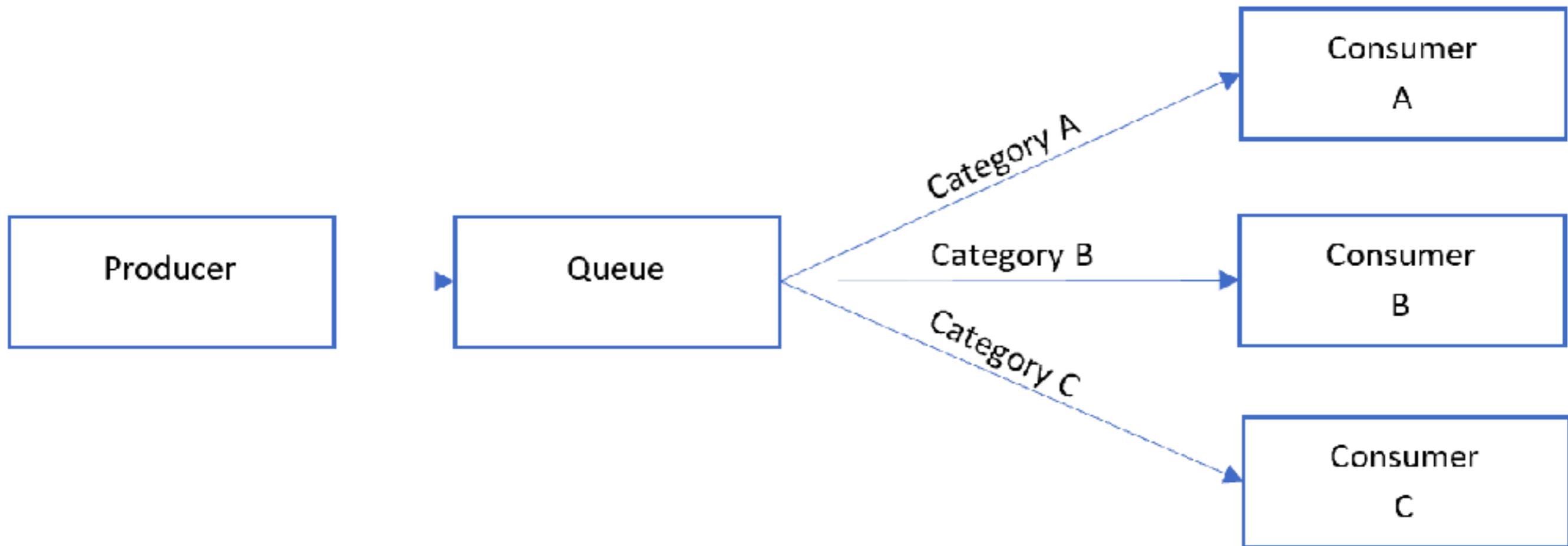
Have each component of the system participate in the decision-making process about the workflow of a business transaction, instead of relying on a central point of control.

Leader Election



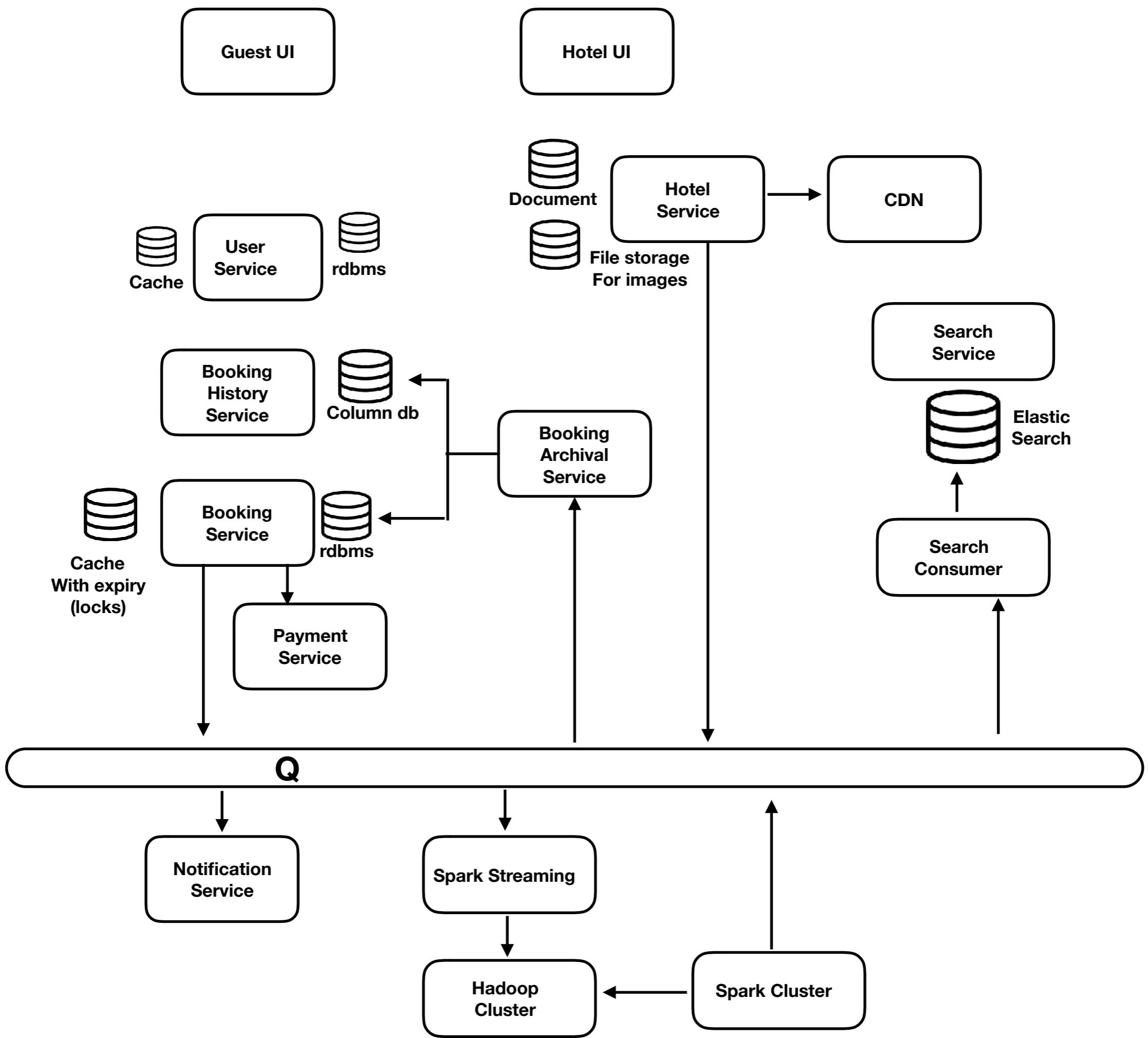
Coordinate the actions performed by a collection of collaborating instances in a distributed application by electing one instance as the leader that assumes responsibility for managing the others. , leader election can introduce new failure modes and scaling bottlenecks. Because of these complications, we carefully consider other options before implementing leader election.

Sequential Convoy

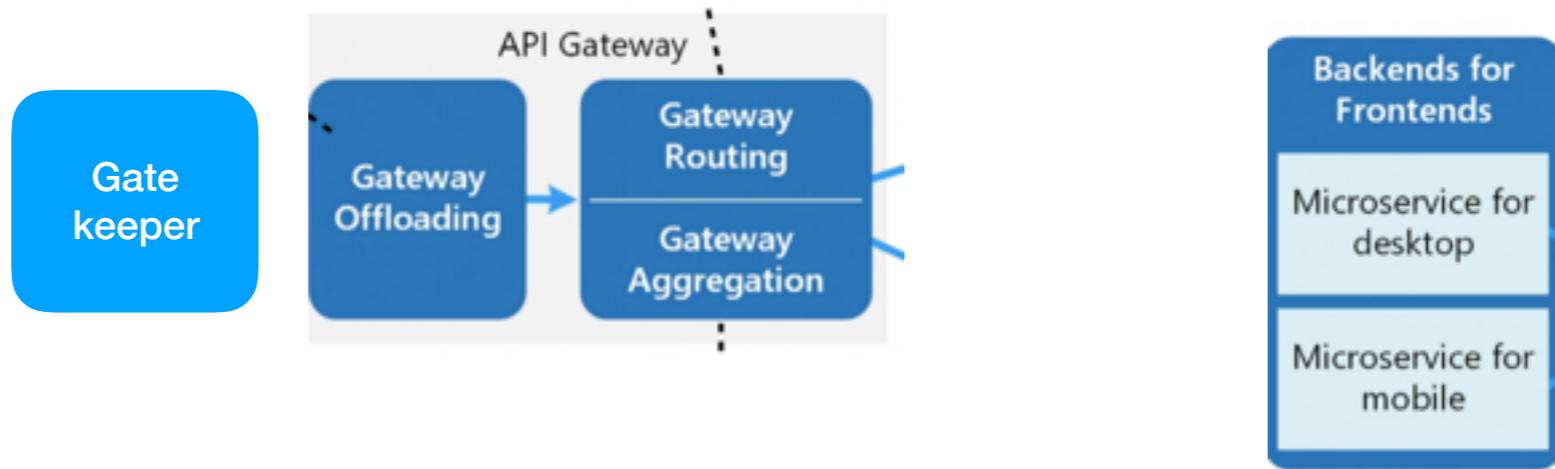


- You have messages that arrive in order and must be processed in the same order. Push related messages into categories within the queuing system, and have the queue listeners lock and pull only from one category, one message at a time.

Case Study



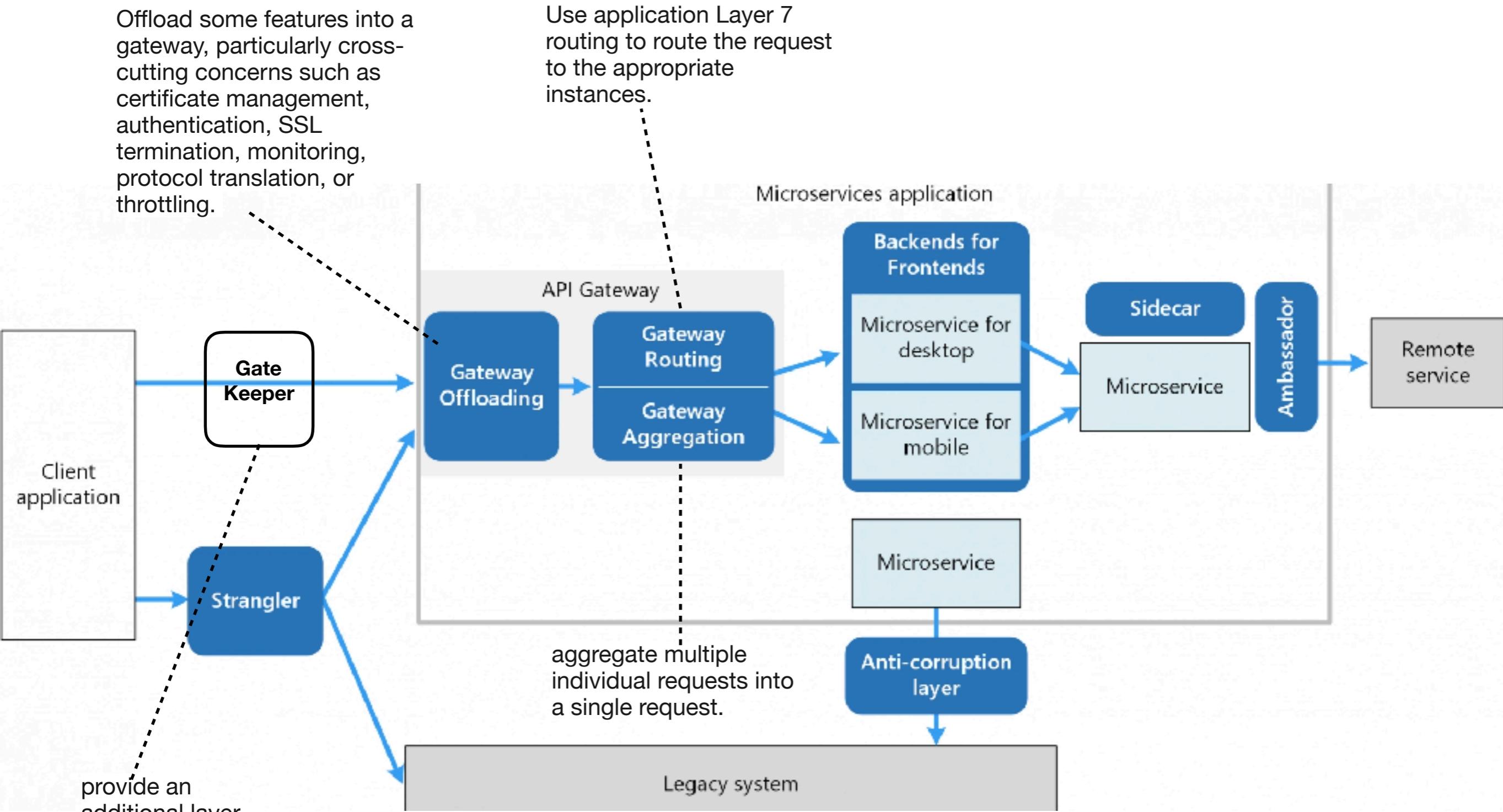
- Azure Front door



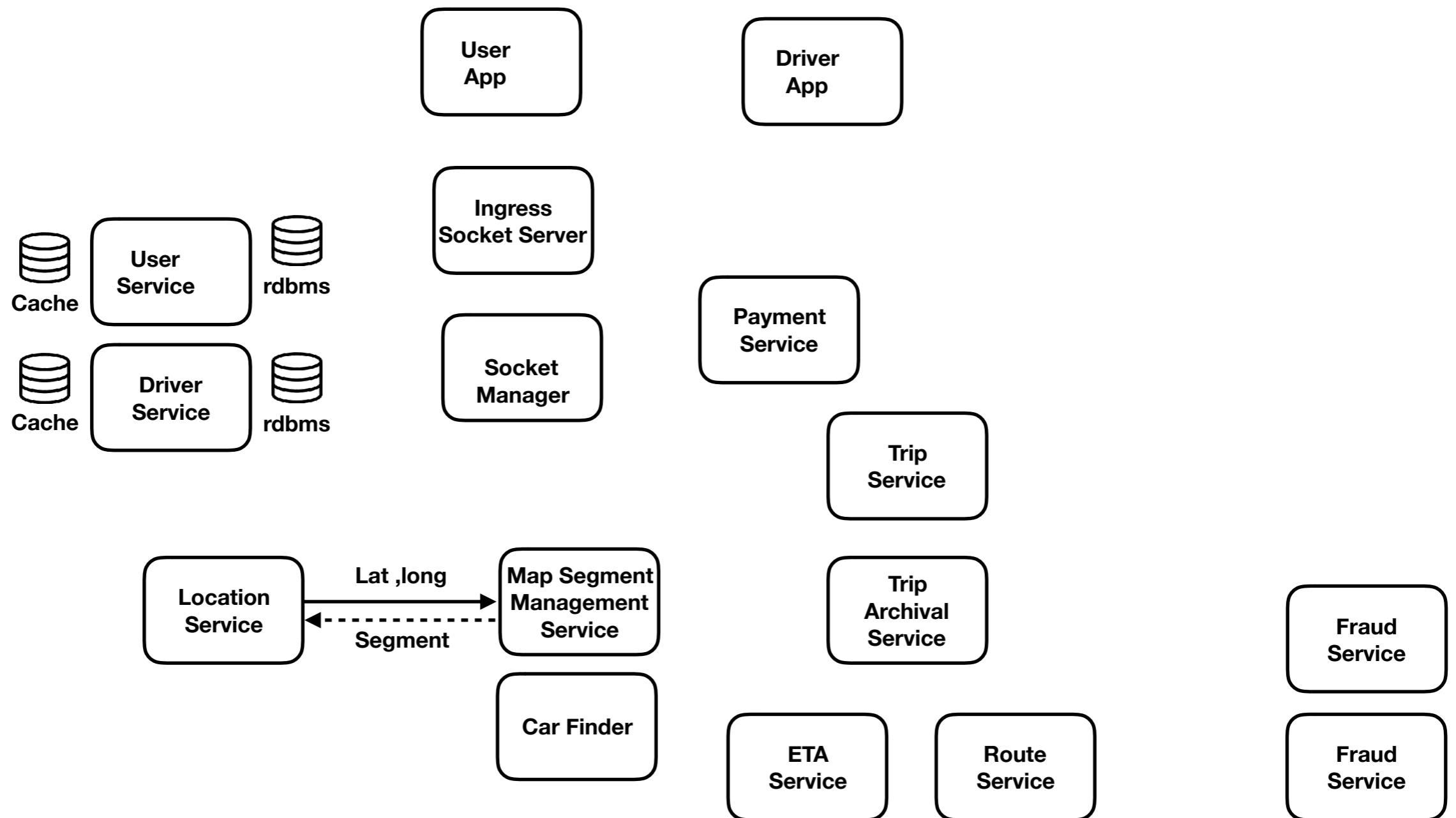
- External Config store

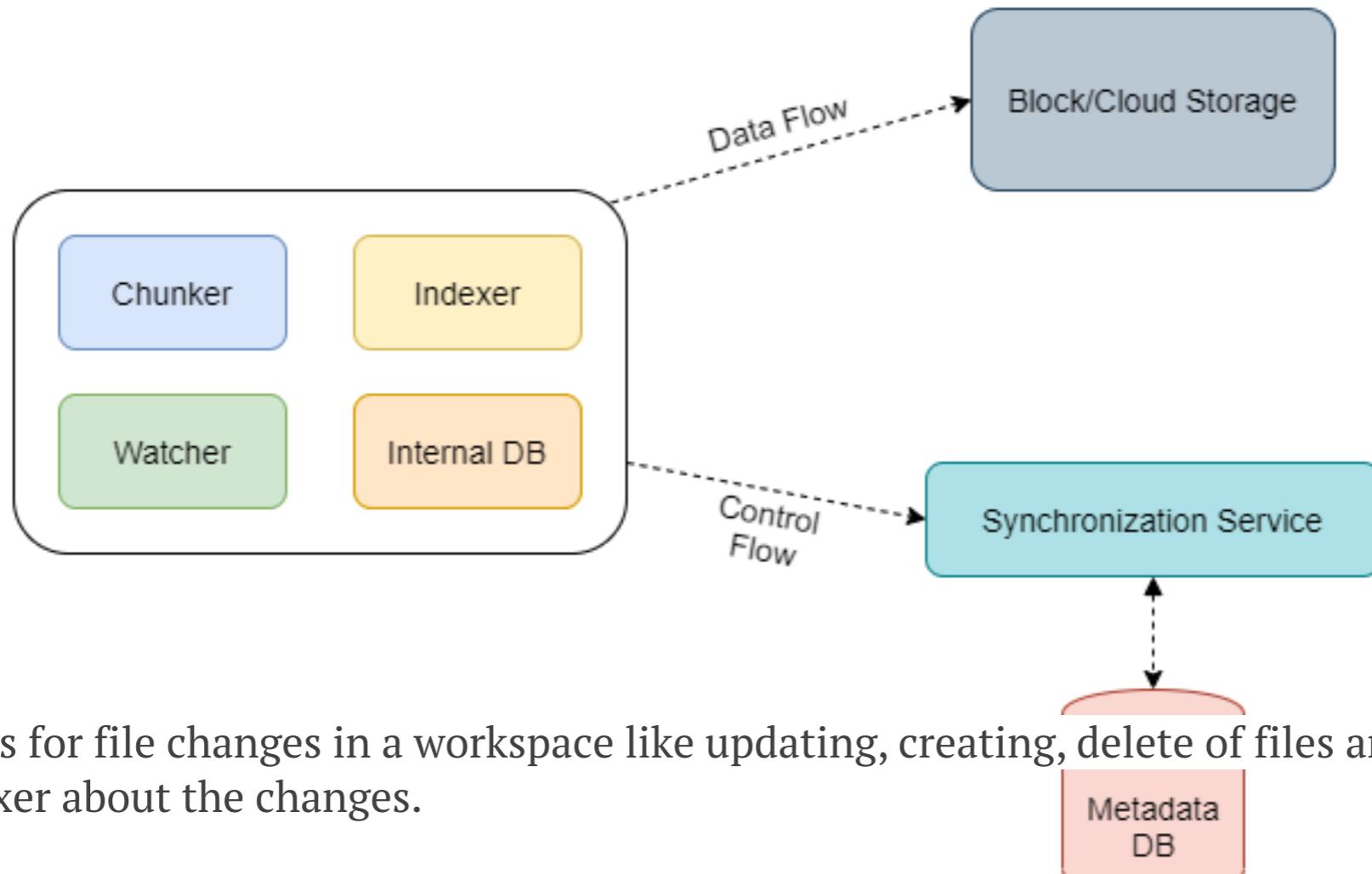


- Federated identity



provide an additional layer of security and limit the system's attack surface.





Watcher monitors for file changes in a workspace like updating, creating, delete of files and folders. The watcher notifies the Indexer about the changes.

Indexer listens for the events from the watcher and updates the Client Metadata Database with information about the chunks of the modified file.

Chunker – The chunker, responsible for splitting the file into smaller pieces, is present within the client. The chunker is also responsible for detecting the chunks that are updated by the client so that only these specific chunks are uploaded to the cloud storage, thus saving bandwidth. It also uploads these chunks to the S3 and notifies the indexer with the hash and the url received from the S3.

Chunker will also reassemble the file by putting together the chunks in their correct order.

