Cloud Design Patterns

Ravindu Nirmal Fernando

SLIIT | March 2025

Load Balancing

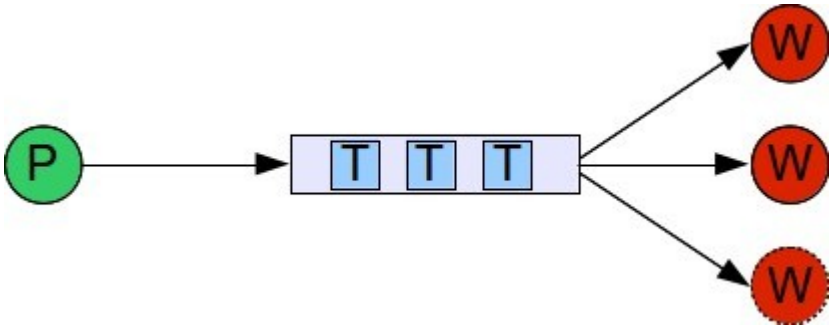
- Improves the distribution of workloads across multiple computing resources
  - Some resources will be busy while others are idle
- Aims to
  - Optimize resource use
  - Maximize throughput
  - Minimize response time
  - Avoid overload of any single resource

Load Balancing

- Counter by distributing load equally
  - When cost of problem is well understood (e.g., matrix multiplication, known tree walk) this is possible
- Some other problems are not that simple
  - Hard to predict how workload will be distributed
  - dynamic load balancing used
  - But require communication between tasks
- 2 methods for dynamic load balancing
  - Task queues vs. work stealing

Task Queues

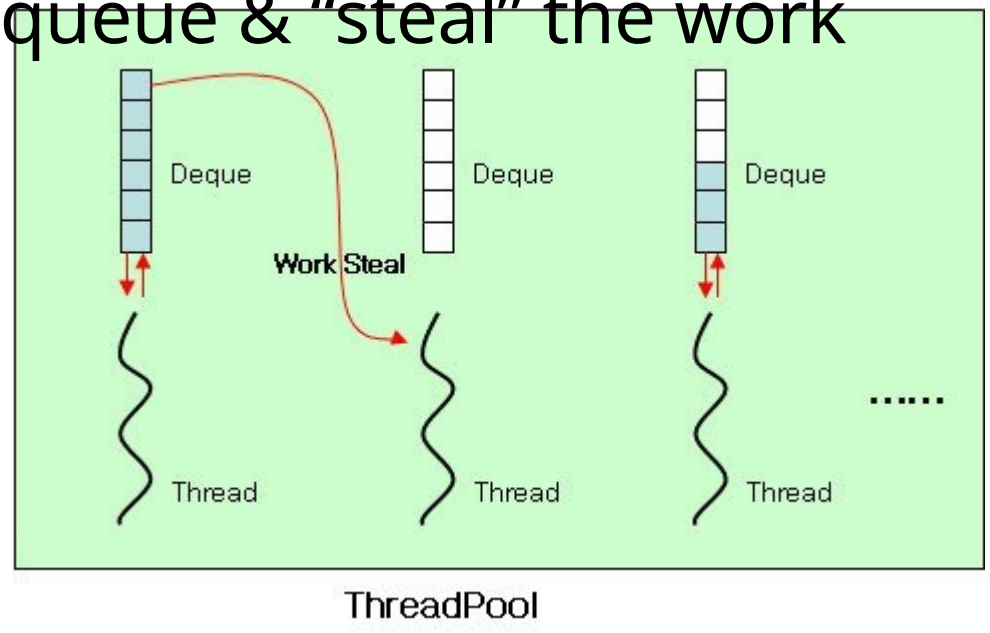
- Multiple instance of task queues (producer consumer)
- Threads comes to the task queue after finishing a task & grab next task
- Typically run with a pool of workers



Source:
http://blog.zenika.com

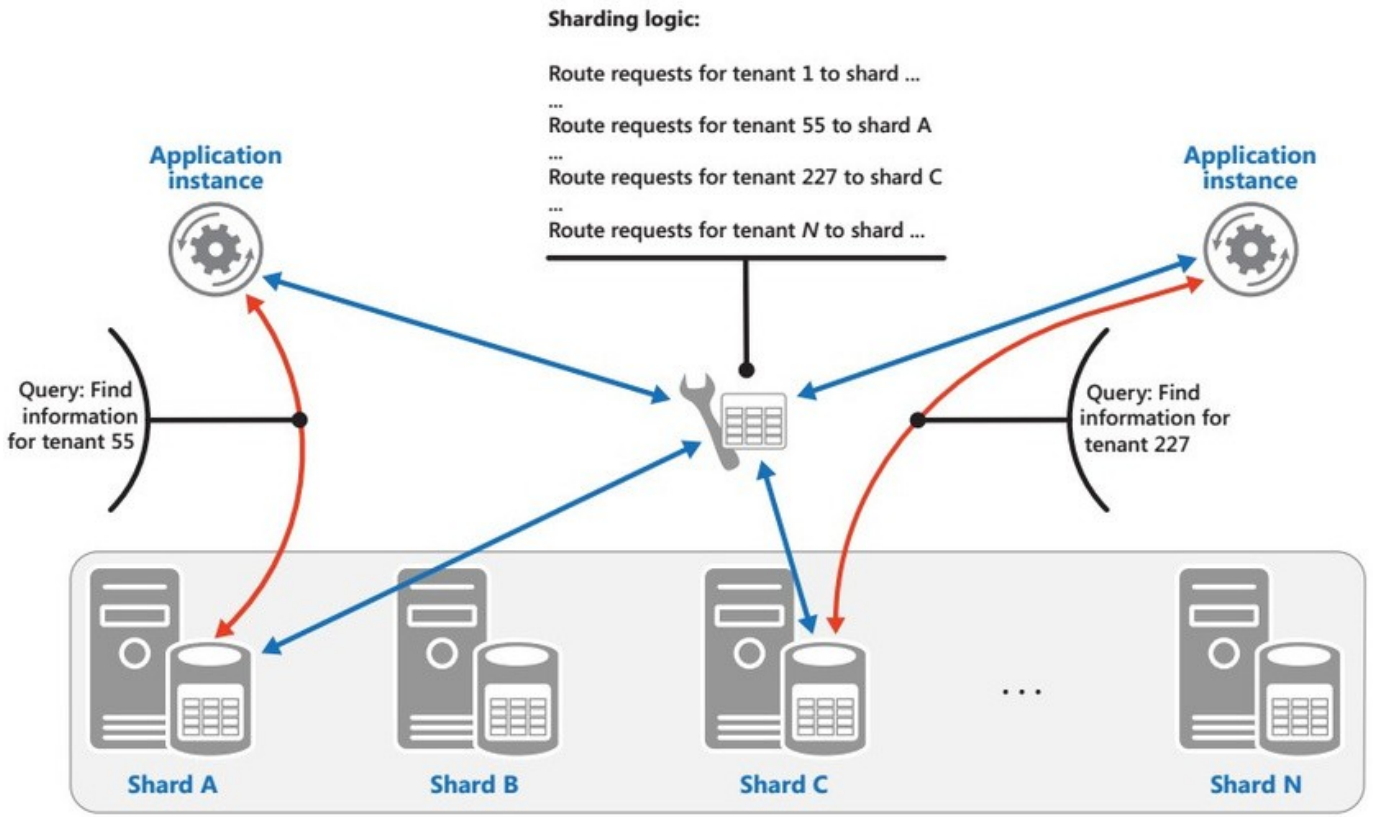
Work Stealing

- Every worker has a task queue
- When 1 worker runs out of work, it goes to other
  worker's queue & "steal" the work



Source:
http://karlsenchoi.blogspot.com

Sharding Pattern

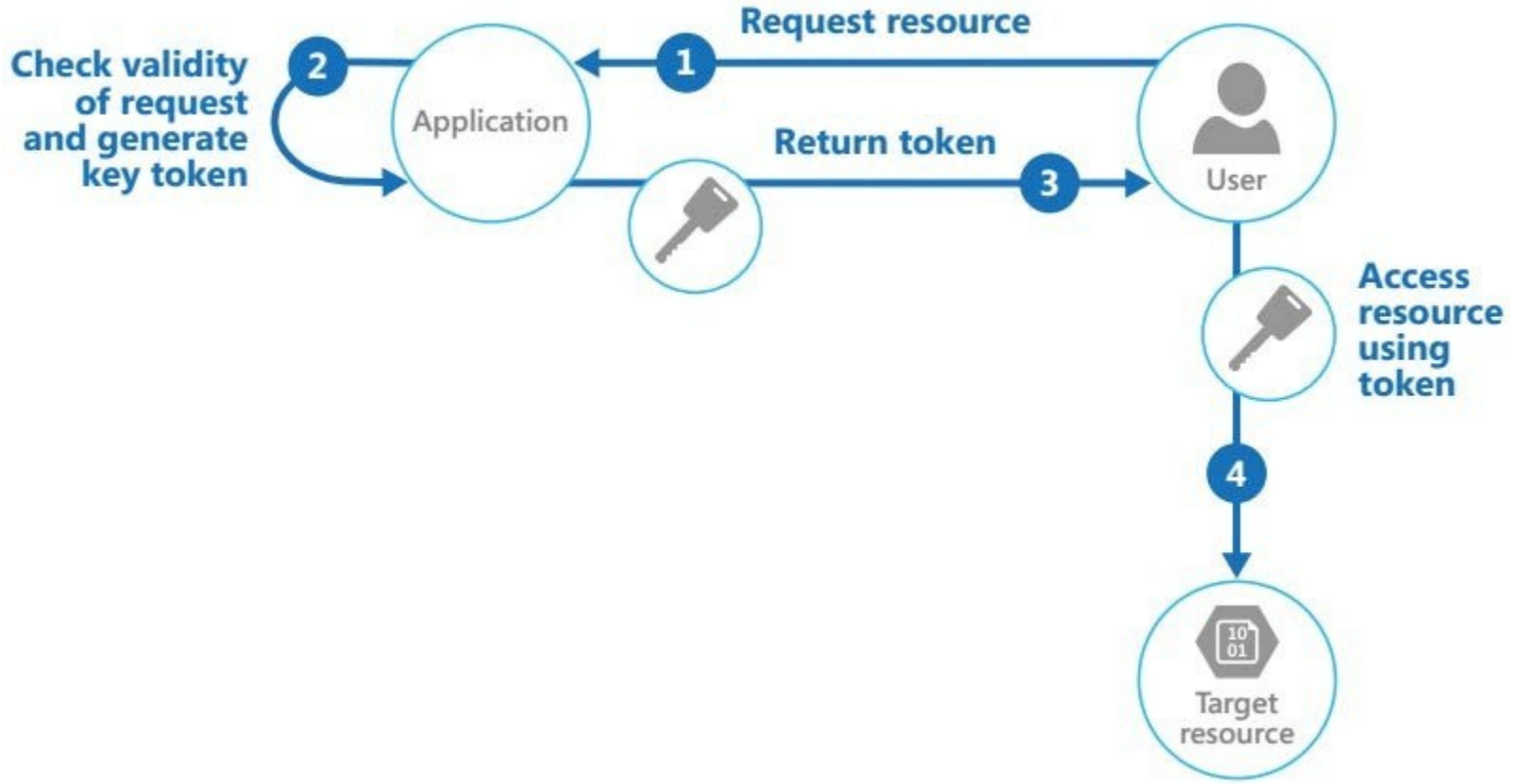


- Divide a data store into a set of horizontal partitions or shards to improve scalability

Sharding Pattern (Cont)

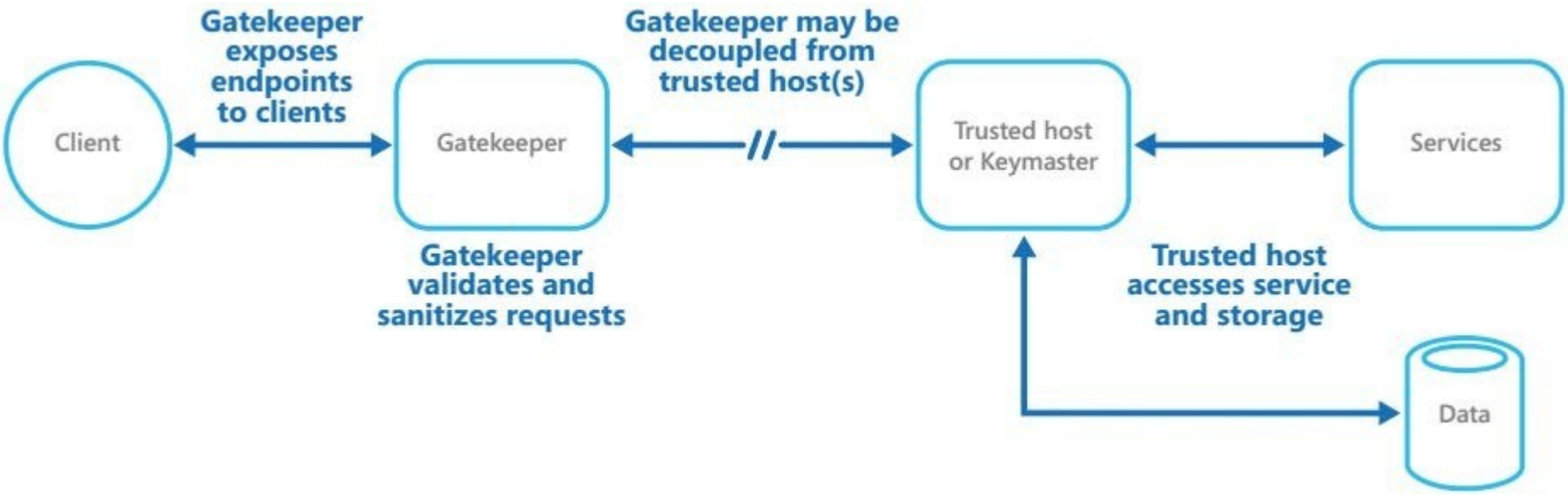
- When
  - Limited storage space
  - Large computation
  - requirement Network
  - bandwidth Geographical constraints
- Sharding Strategies
  - Lookup Strategy – map request using a shard key
  - Range Strategy – groups related items together in the same shard
  - Hash Strategy – shard decided based on hashing data attributes

Valet Key Pattern



- Use a token or key that provides client with restricted direct access to a specific resource or service

Gatekeeper Pattern

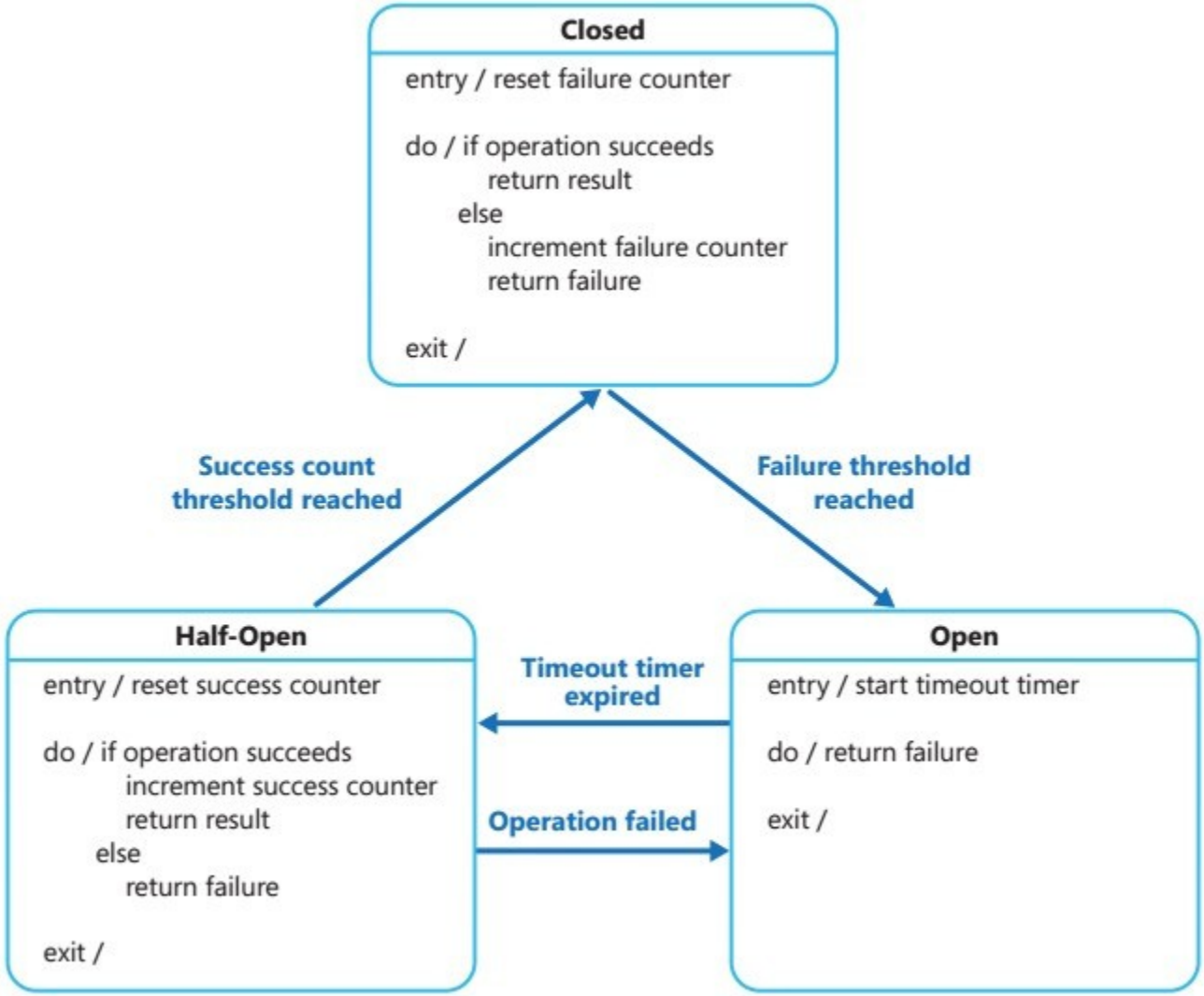


- Protect applications & services using a dedicated host instance that acts as a broker
  - Validates & sanitizes requests
  - Passes requests & data between them

Gatekeeper Pattern (Cont.)

- Only function is to validate & sanitize requests Should use secure communication between gatekeeper & trusted hosts
- Internal end point must connect only to gatekeeper
- Gatekeeper must run in limited privilege mode
- May use multiple gatekeepers for availability

Circuit Breaker Pattern



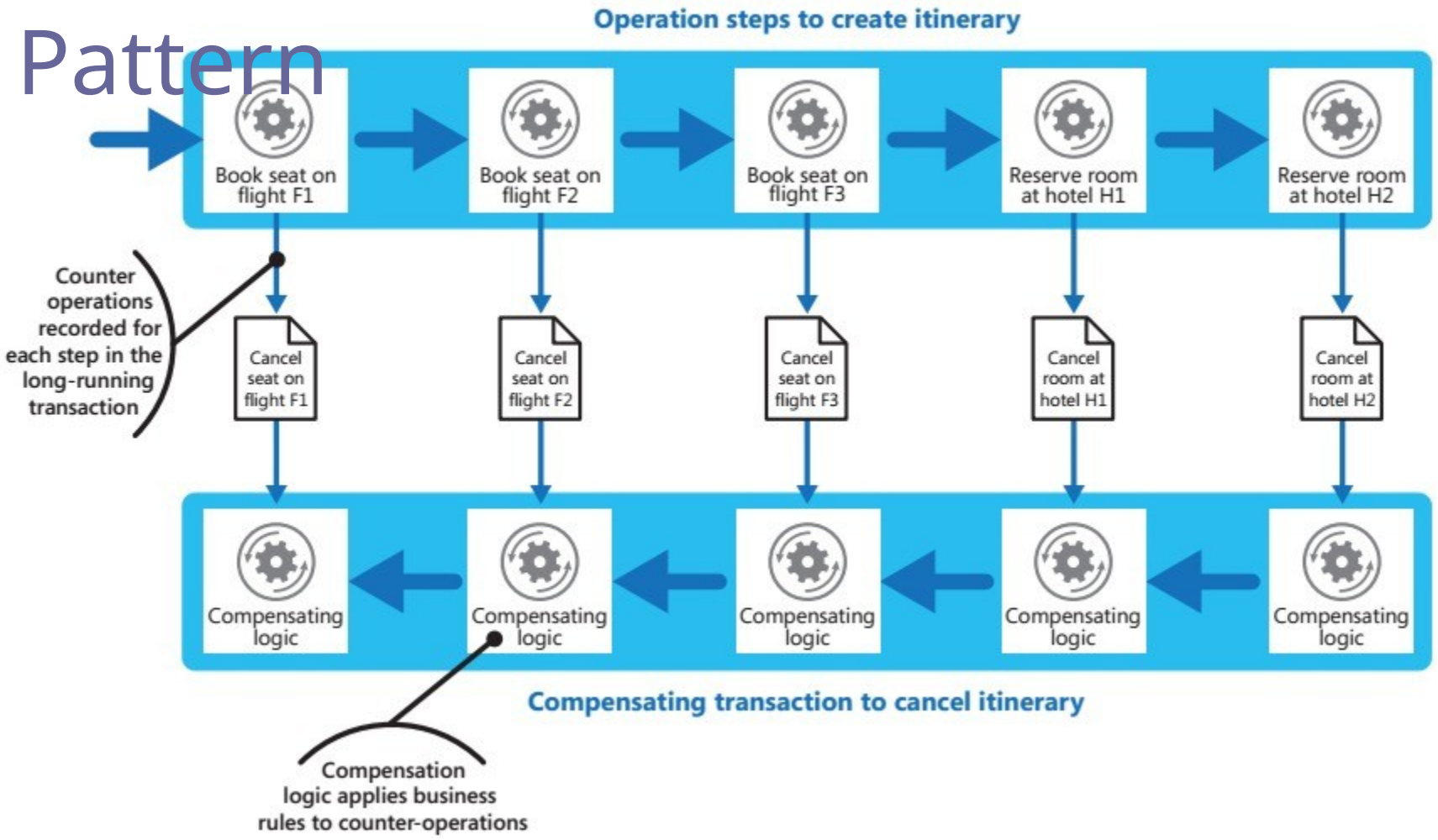
Circuit Breaker Pattern (Cont.)

- When
  - Handle faults that may take a variable amount of time to rectify when connecting to a remote service/resource
  - When a simple retry will not work
  - Prevent application from getting tied-up due to retry
- Half-Open State
  - Allow checking whether service is responding by issuing a limited set of requests
  - Prevent repeated system failures due to rapid load/ volume

Circuit Breaker Pattern (Cont.)

- Parameters
  - Types of exceptions
  - Handling exceptions
  - Logging & replay
  - Testing failed operations
  - Manual reset

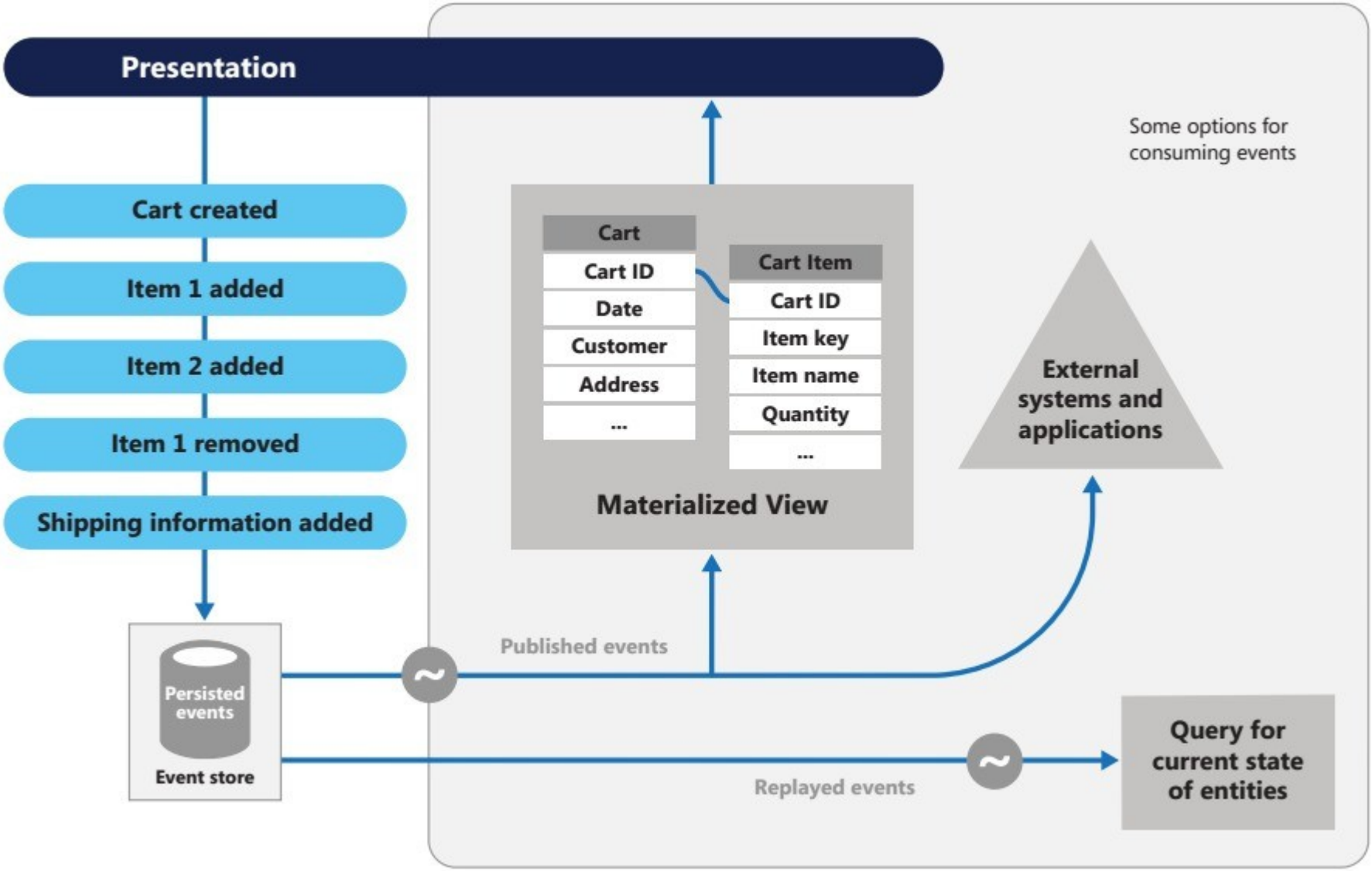
Compensating Transaction Pattern



Compensating Transaction Pattern (Cont.)

- Undo work performed by a series of steps, which together define an eventually consistent operation
- Implement a workflow
  - As operation proceeds, system records information about each step & how the work by that step can be undone
  - If operation fails at any point, workflow reverts back through steps it has completed while performing work that reverses each step

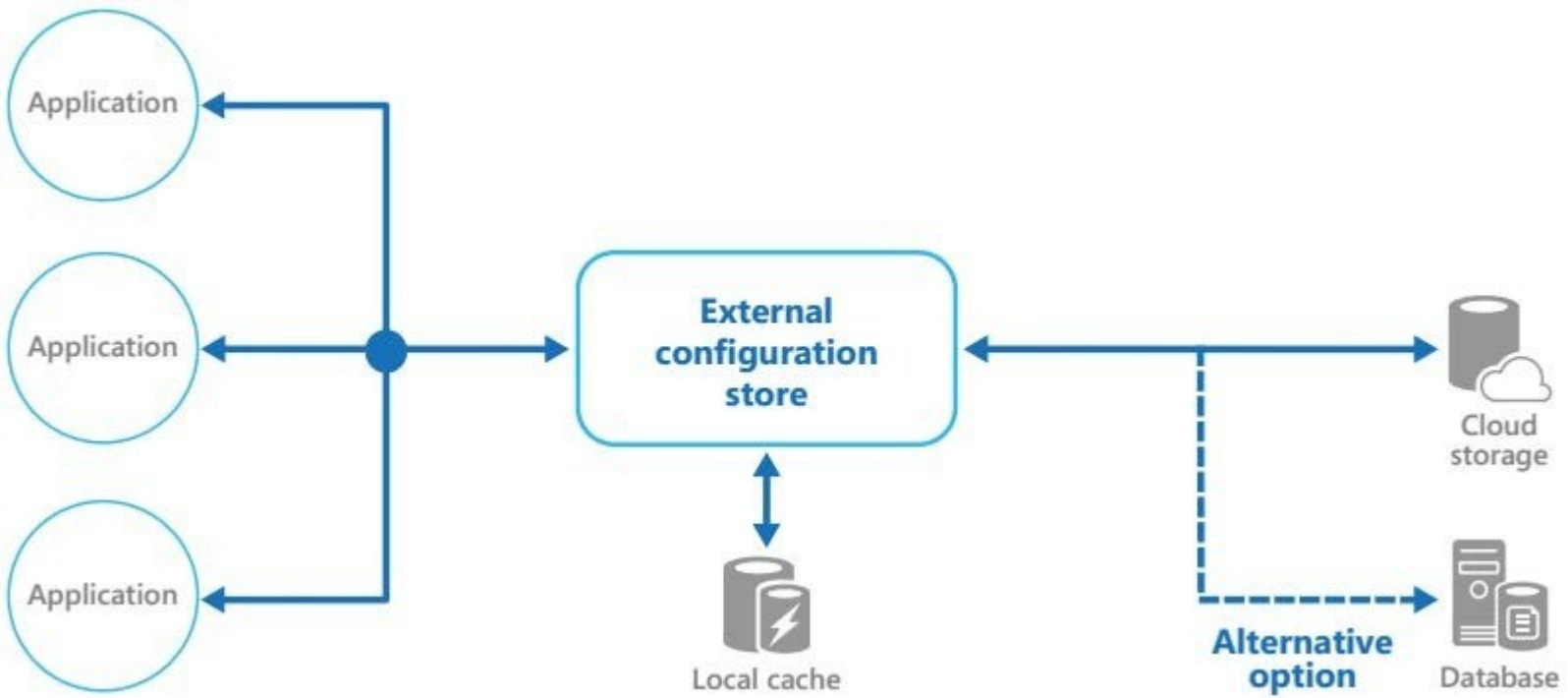
Event Sourcing Pattern



Event Sourcing Pattern (Cont.)

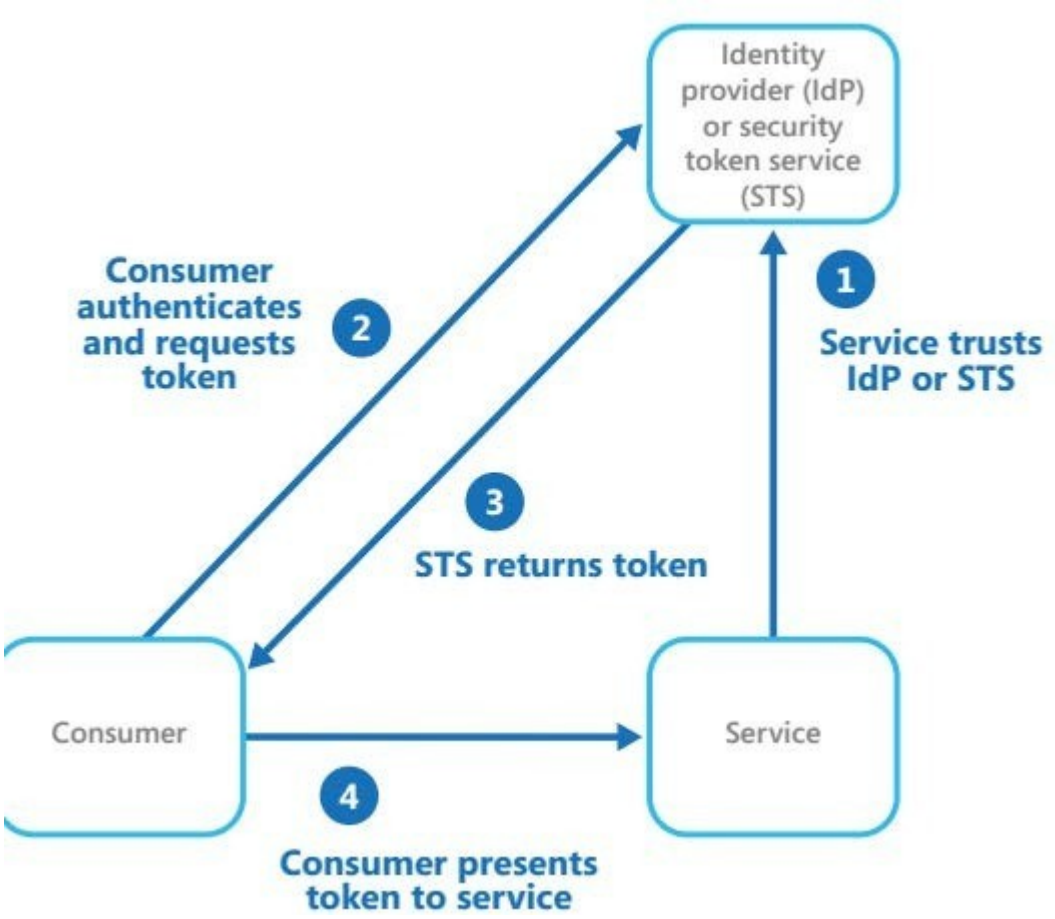
- Record full series of events than current
- state Pros
  - Avoid requirement to synchronize data
    - Traditional Create, Read, Update, & Delete (CRUD) model too slow
    - Improve performance with eventual consistency
  - Scalability
  - Responsiveness
  - Provide consistency for transactional
  - data Full audit trails
- Cons
  - Consistency relaxed

External Configuration Store Pattern



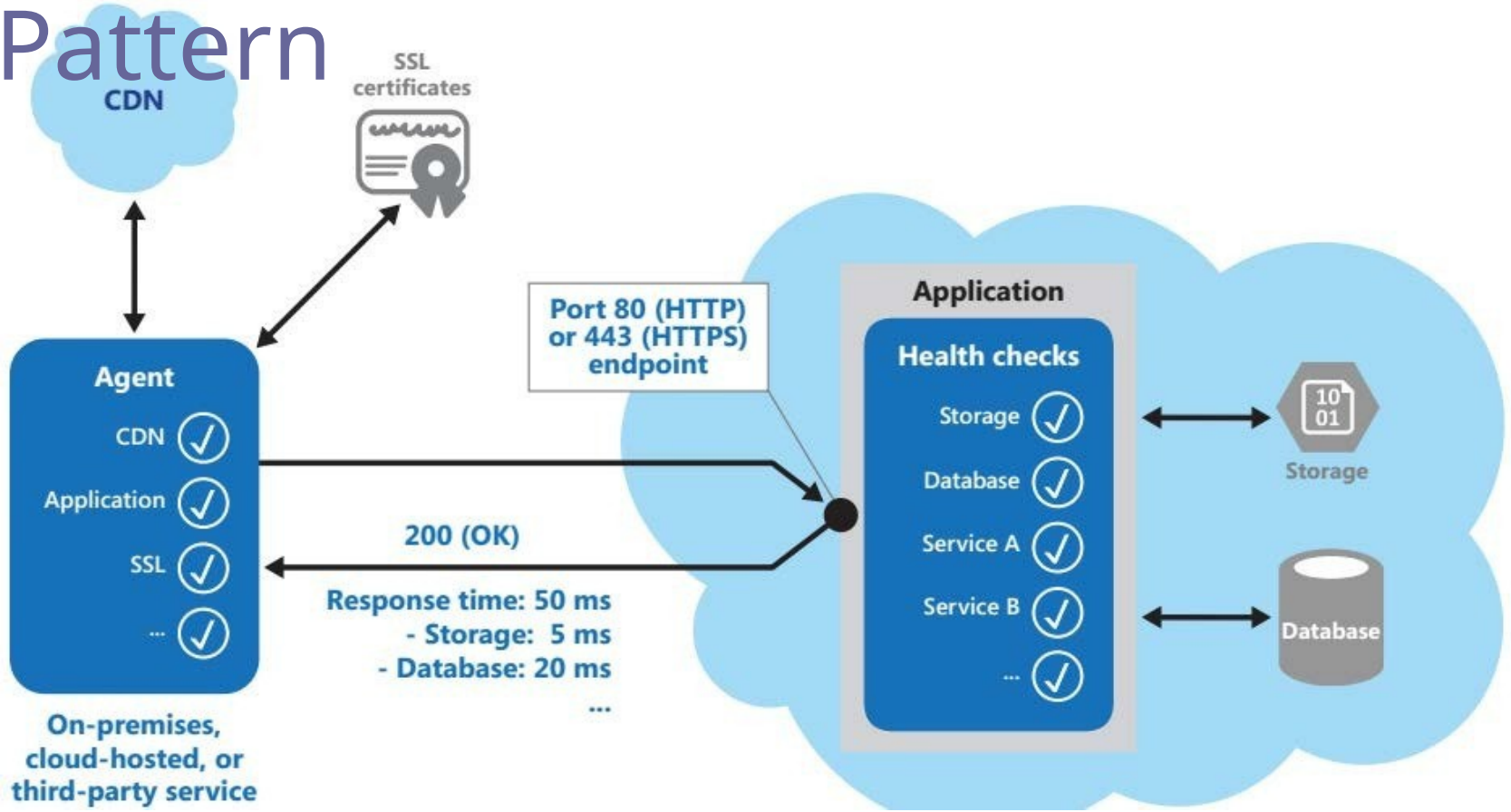
- Move configuration information out of application deployment package to a central location

Federated Identity Pattern



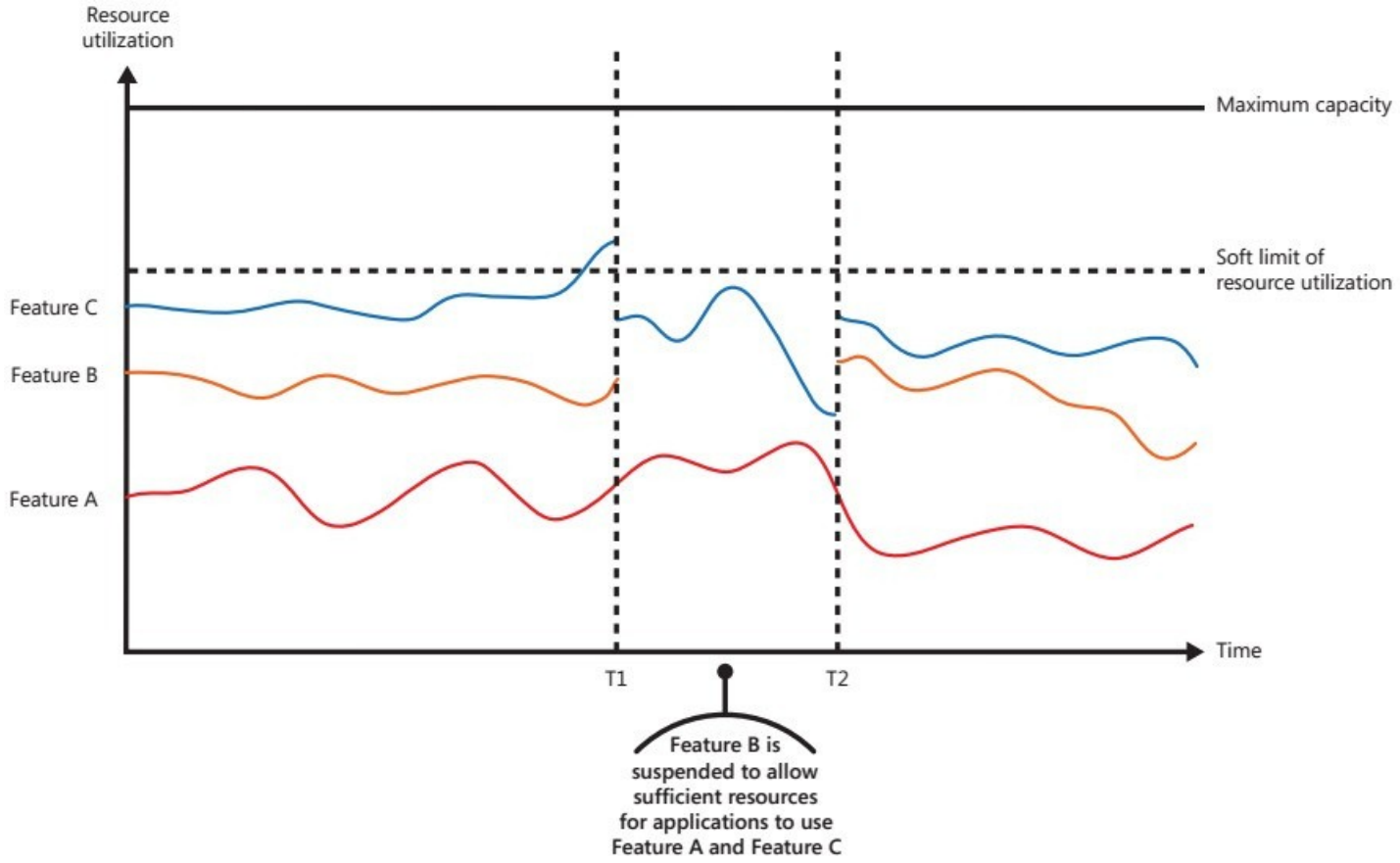
- Delegate authentication to an external identity provider

Health Endpoint Monitoring Pattern



- Functional checks within an application that external tools can access through exposed endpoints at regular intervals

Throttling Pattern



- Control consumption of resources used by an instance
- Allow system to continue to function & meet SLA even when an increase in demand places an extreme load on resources