



# Cloud Design Patterns

Ravindu Nirmal Fernando

SLIIT | March 2025

# Load Balancing

---

$\pi$  Improves the distribution of workloads across multiple computing resources

- Some resources will be busy while others are idle

$\pi$  Aims to

- Optimize resource use
- Maximize throughput
- Minimize response time
- Avoid overload of any single resource

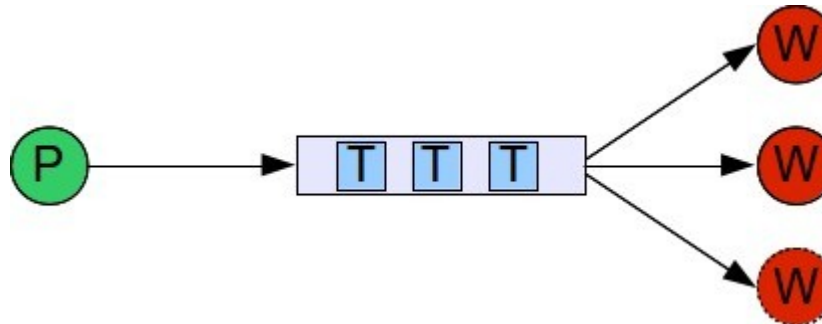
# Load Balancing

---

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

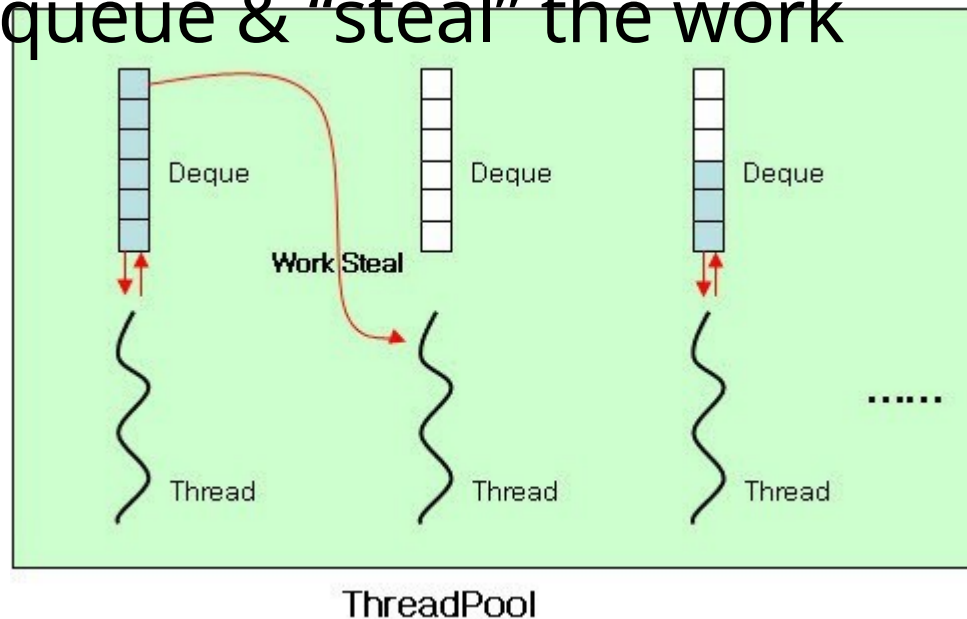
# Task Queues

- 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



# Work Stealing

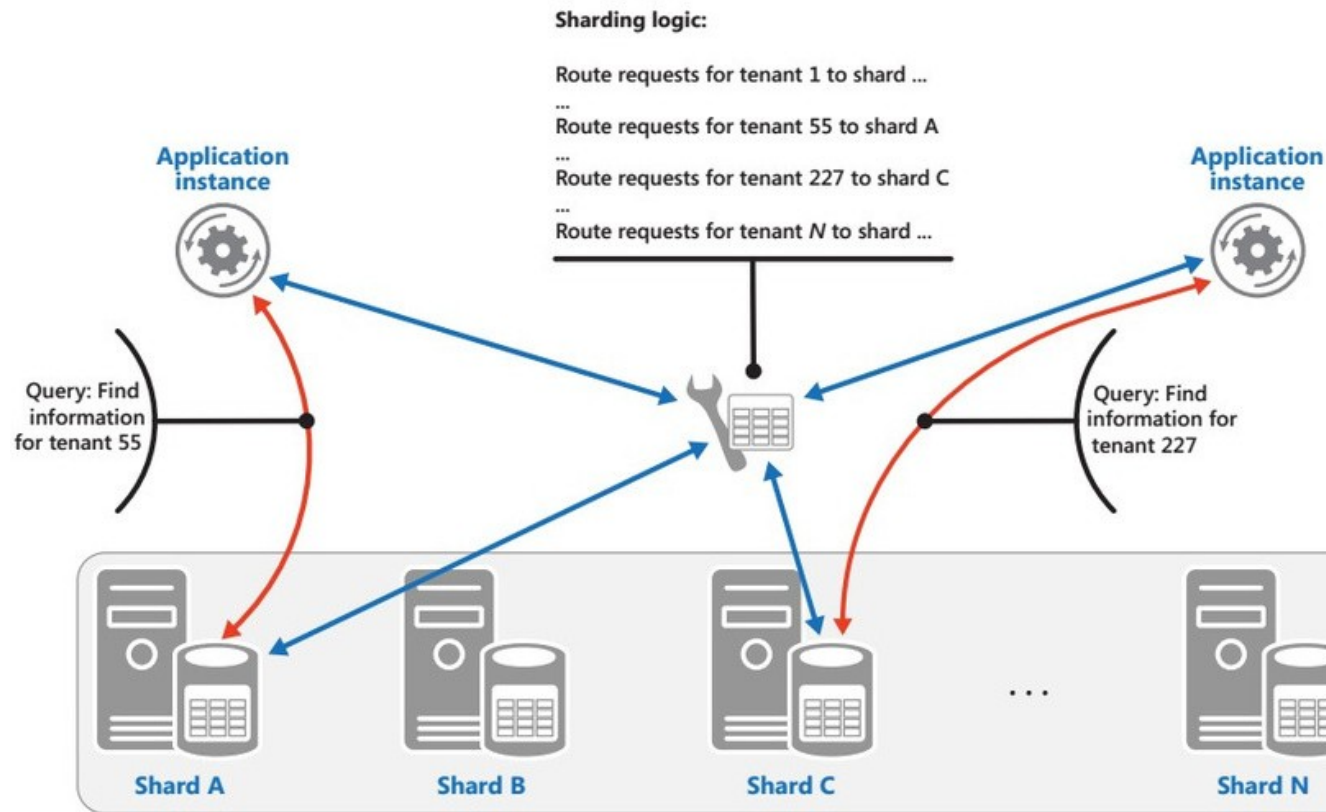
- π Every worker has a task queue
- π When 1 worker runs out of work, it goes to other worker's queue & "steal" the work



Source:

<http://karlsenchoi.blogspot.com>

# Sharding Pattern



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

# Sharding Pattern (Cont)

---

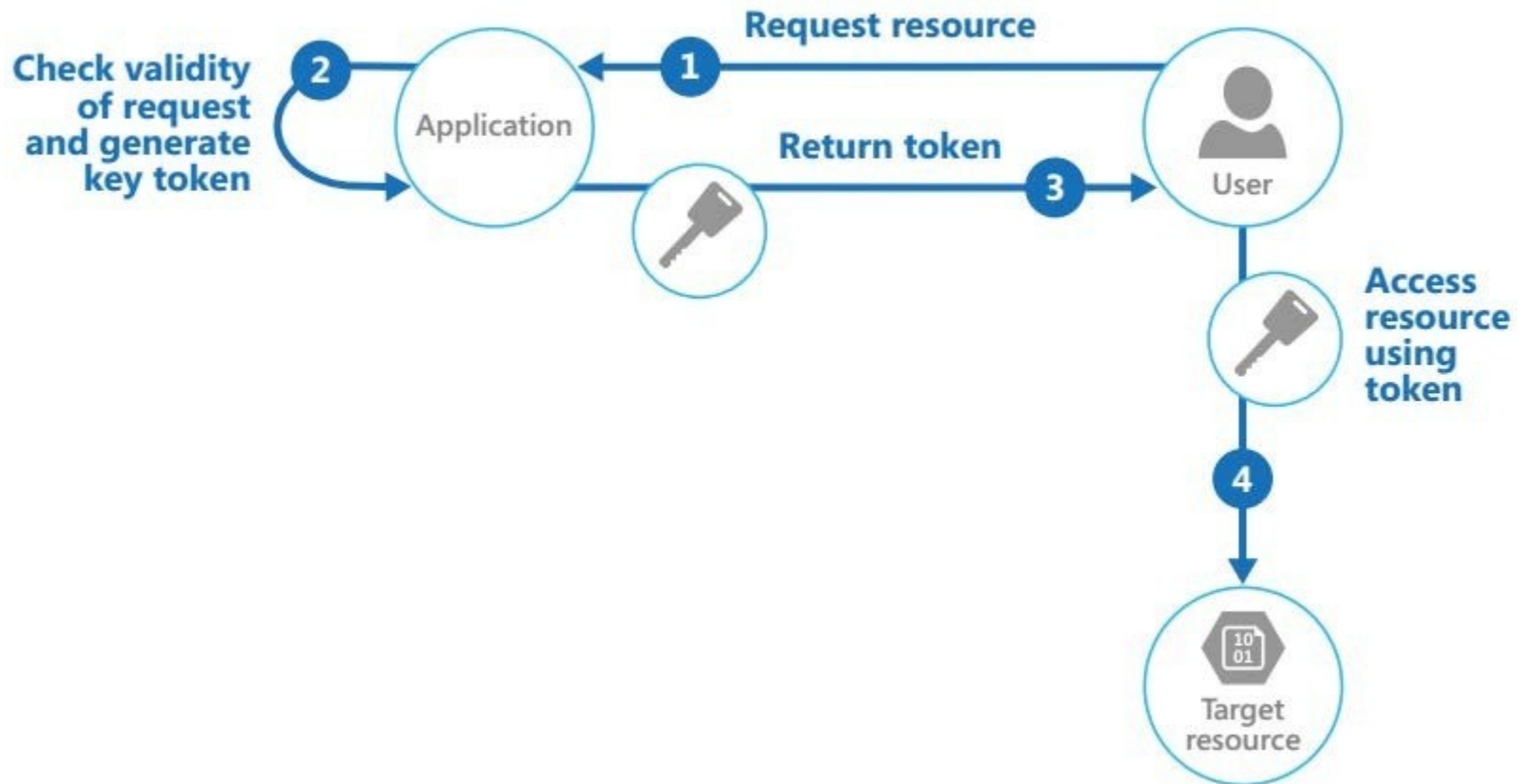
## $\pi$ When

- ✓ Limited storage space
- ✓ Large computation
- ✓ requirement Network
- ✓ bandwidth Geographical constraints

## $\pi$ 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

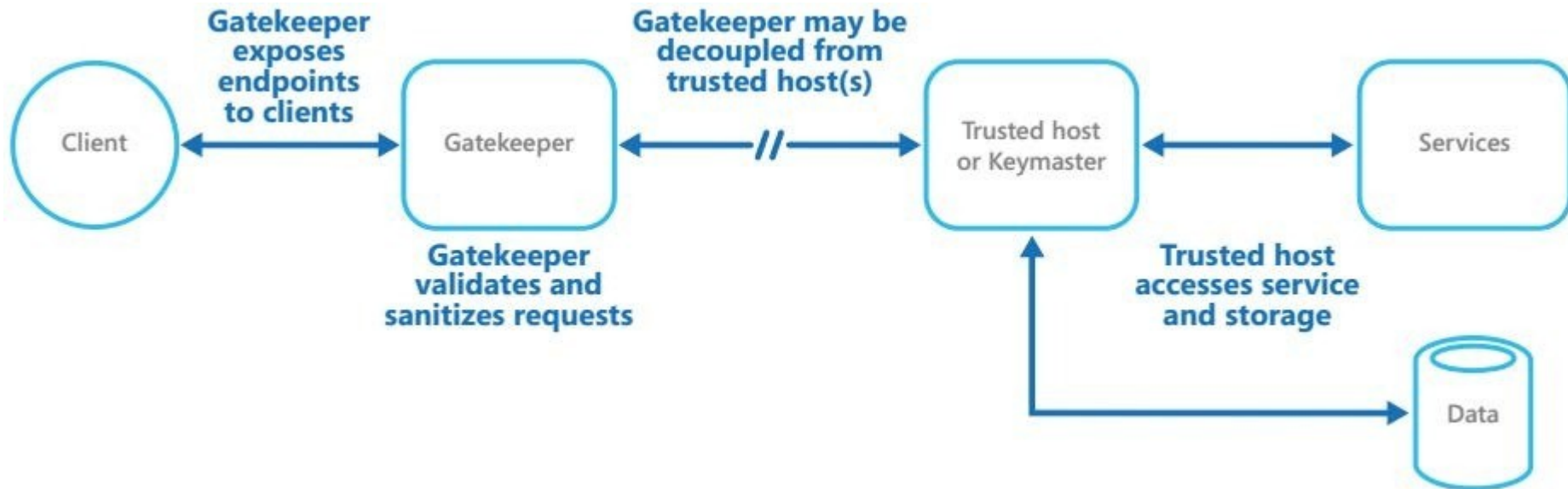
# Valet Key Pattern



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



# Gatekeeper Pattern



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

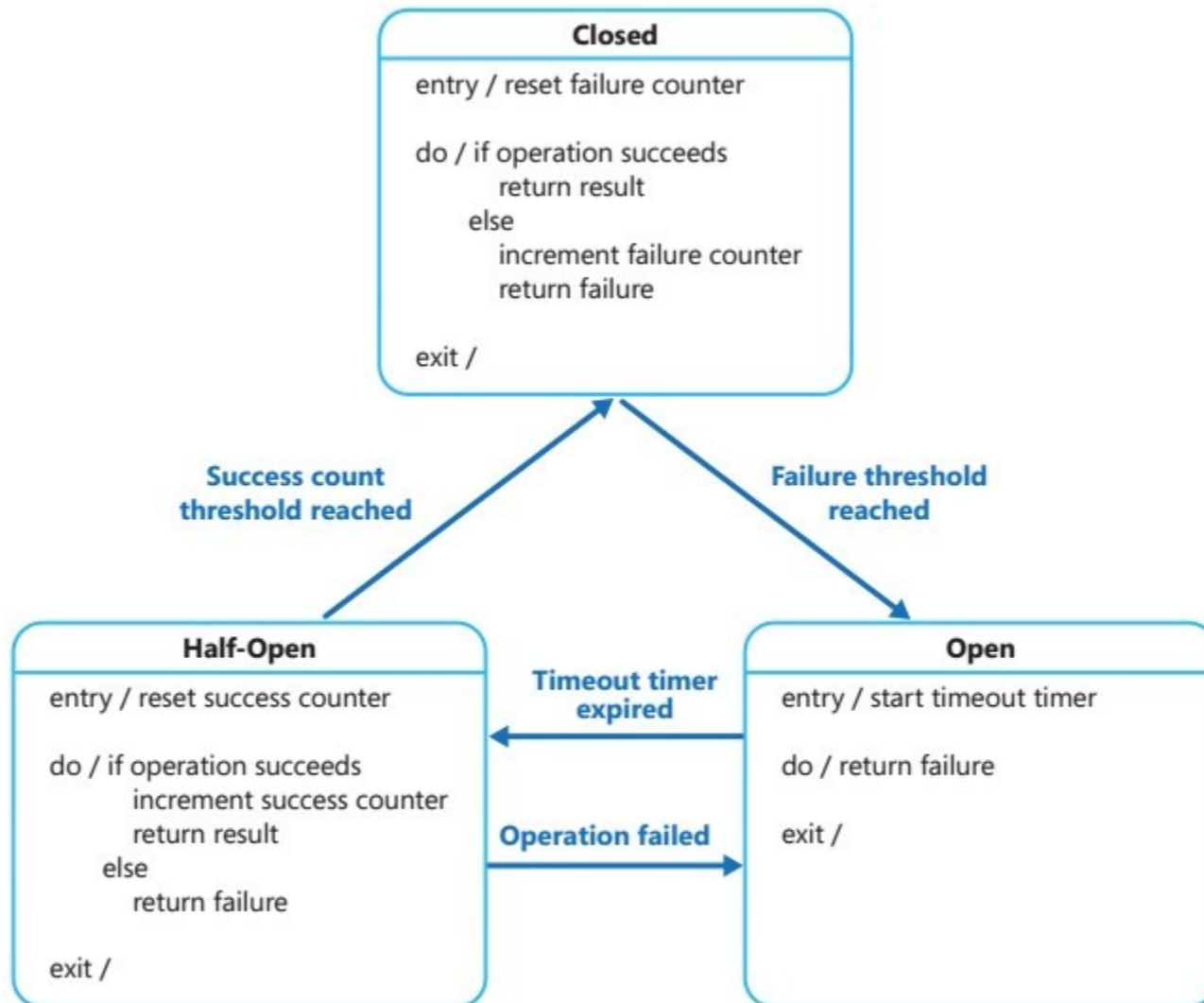
# Gatekeeper Pattern (Cont.)

---

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

# Circuit Breaker Pattern

---



# Circuit Breaker Pattern

---

## (Cont.)

- $\pi$  When
  - Handle faults that may take a variable amount of time to rectify when connecting to a remote service/resource
- $\vee$  When a simple retry will not work
- $\vee$  Prevent application from getting tied-up due to retry

## $\pi$ Half-Open State

- $\vee$  Allow checking whether service is responding by issuing a limited set of requests
- $\vee$  Prevent repeated system failures due to rapid load/  
volume

# Circuit Breaker Pattern

---

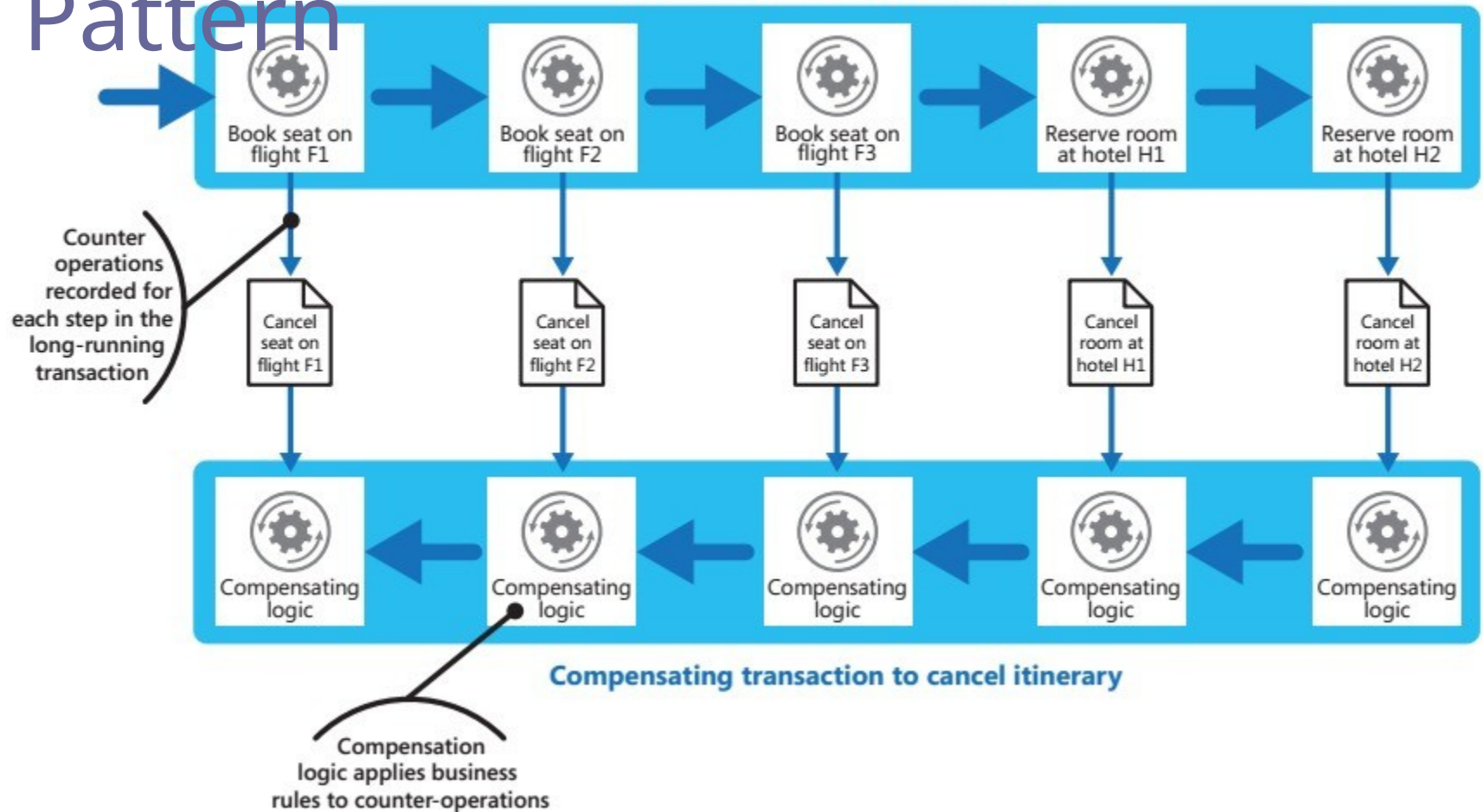
## (Cont.) Parameters

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

# Compensating Transaction

## Pattern

Operation steps to create itinerary



# Compensating Transaction Pattern (Cont.)

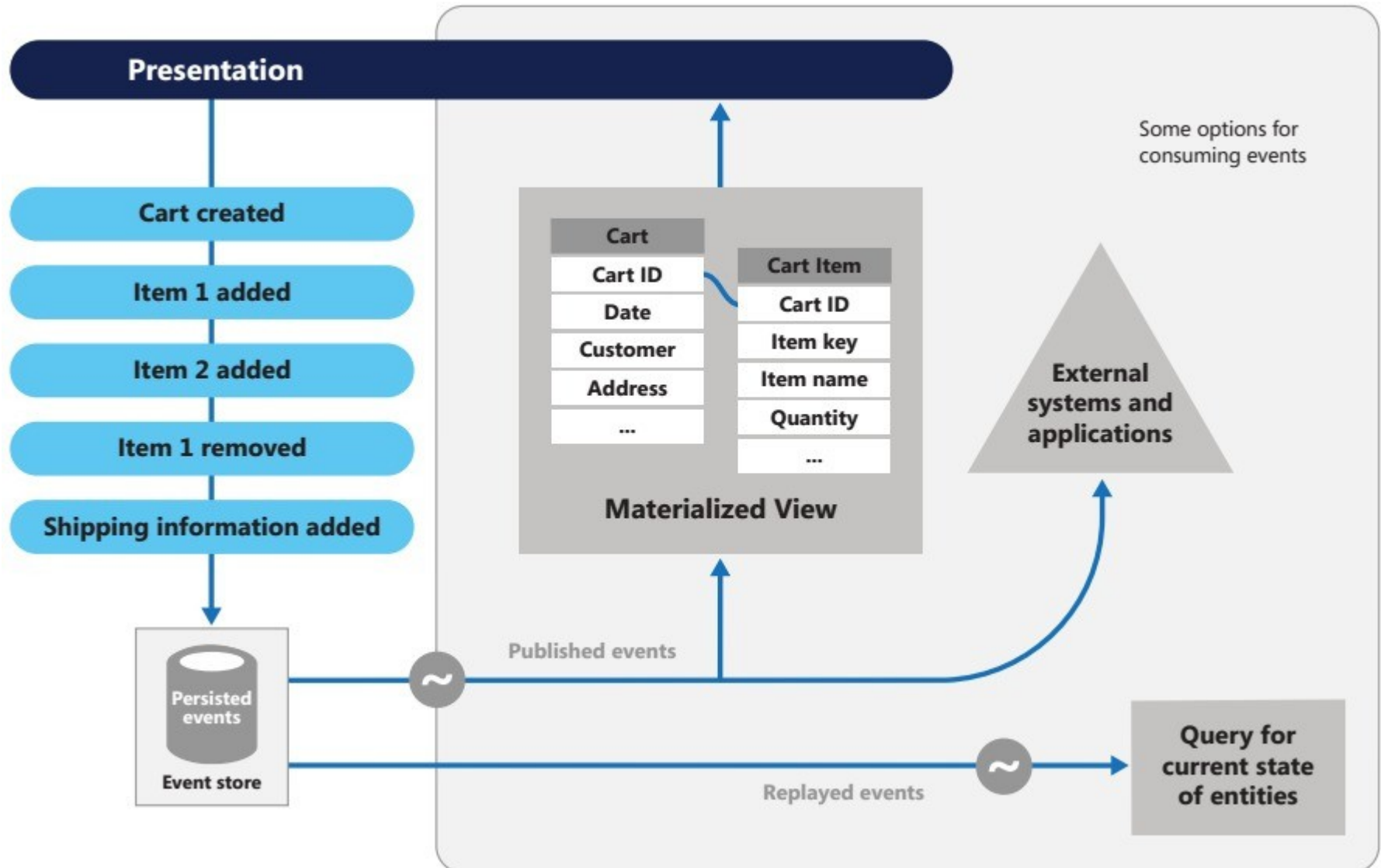
$\pi$  Undo work performed by a series of steps, which together define an eventually consistent operation

$\pi$  Implement a

$\nu$  workflow As operation proceeds, system records information about each step & how the work by that step can be undone

$\nu$  If operation fails at any point, workflow back through steps it has completed while performing undoing work that reverses each step

# Event Sourcing Pattern





# Event Sourcing Pattern (Cont.)

---

$\pi$  Record full series of events than current

$\pi$  state Pros

- ✓ Avoid requirement to synchronize

- data traditional Create, Read, Update, & Delete (CRUD) model too slow

- $\pi$  Improve performance with eventual consistency

- ✓ Scalability

- ✓ Responsiveness

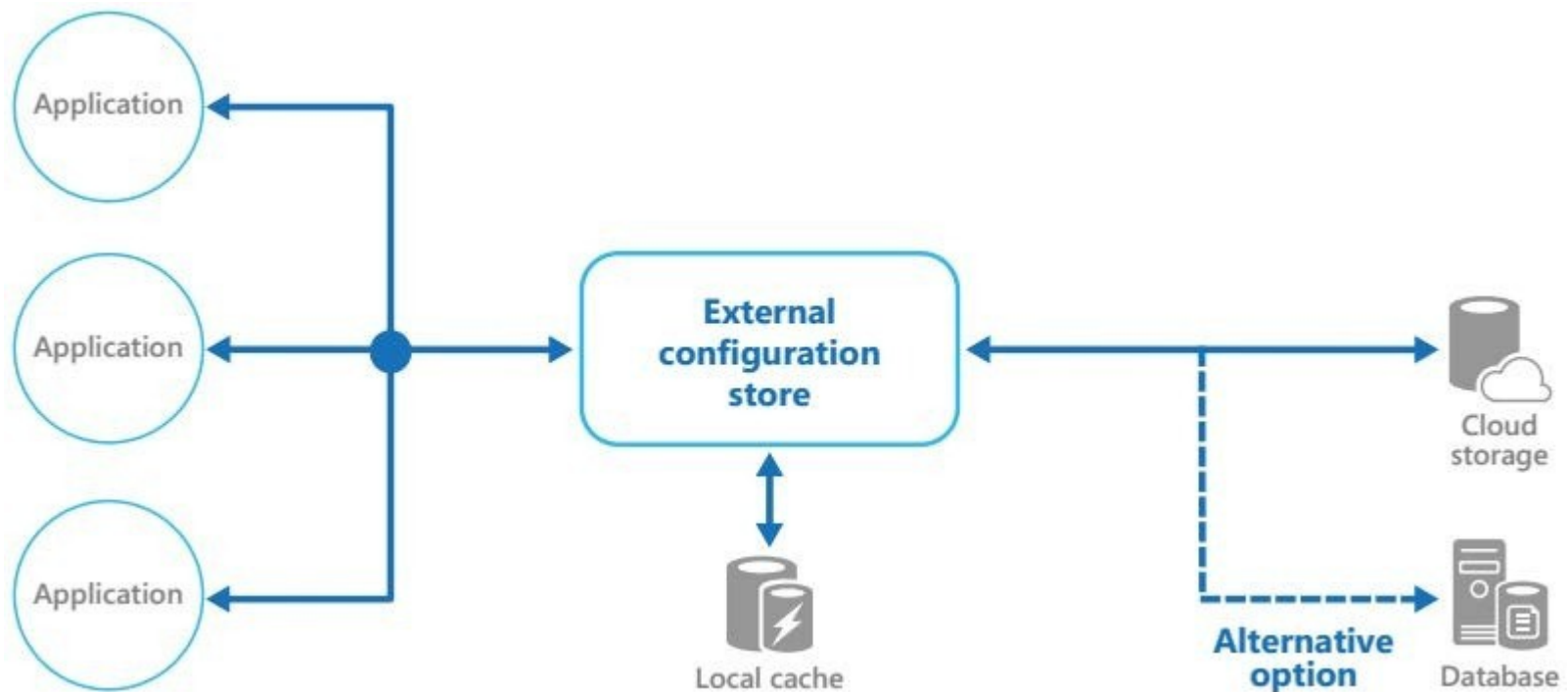
- ✓ Provide consistency for transactional

- ✓ data Full audit trails

$\pi$  Cons

- ✓ Consistency relaxed

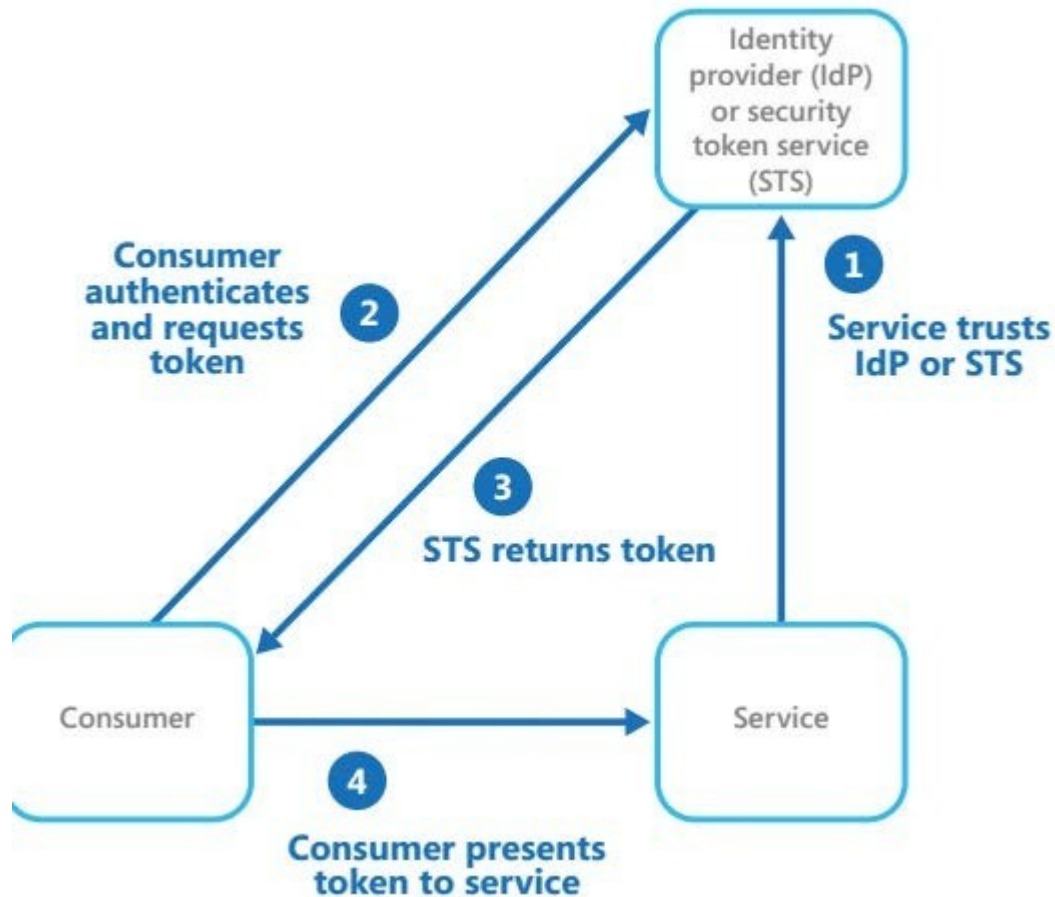
# External Configuration Store Pattern



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

# Federated Identity Pattern

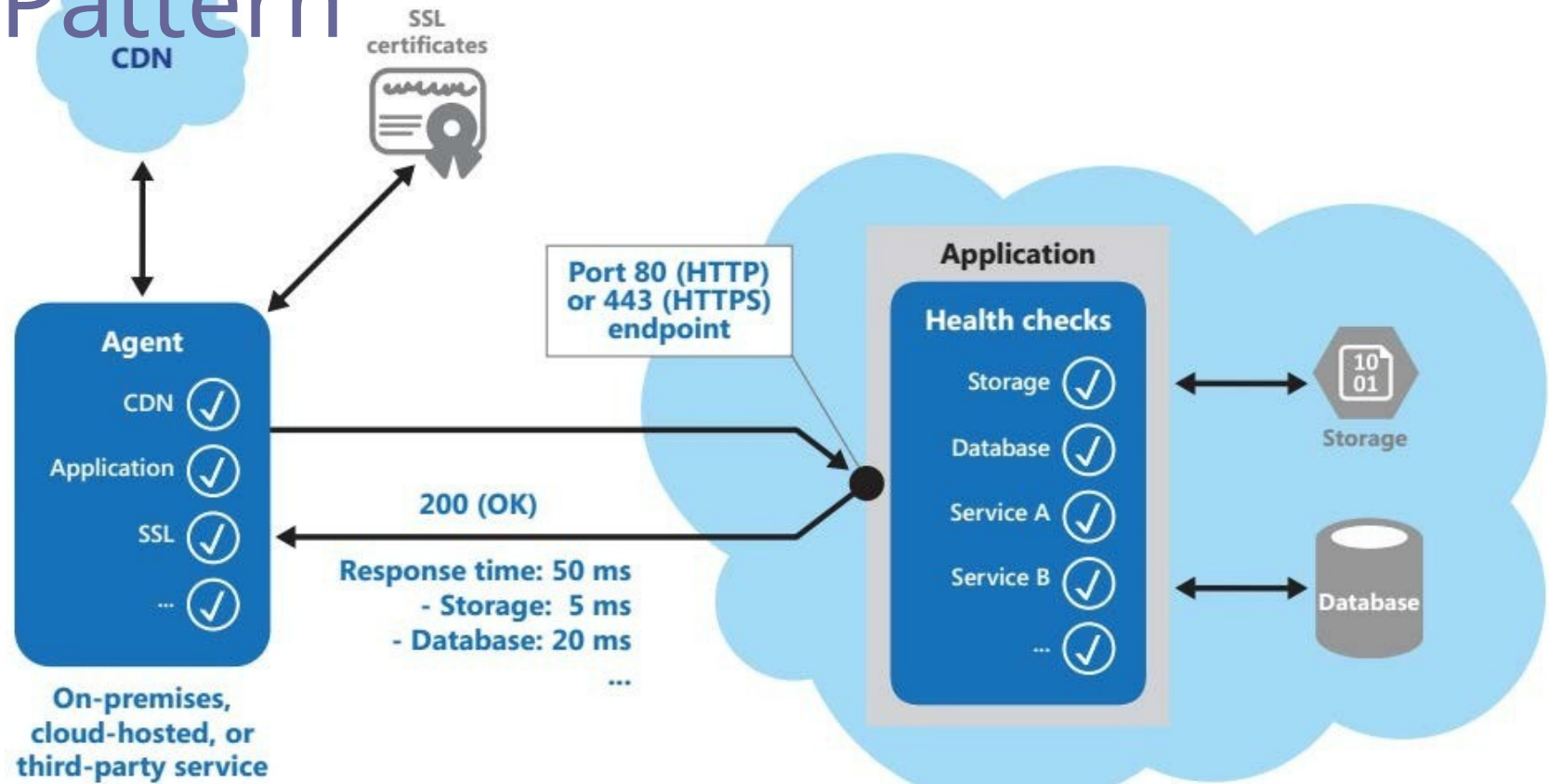
---



- π Delegate authentication to an external identity provider

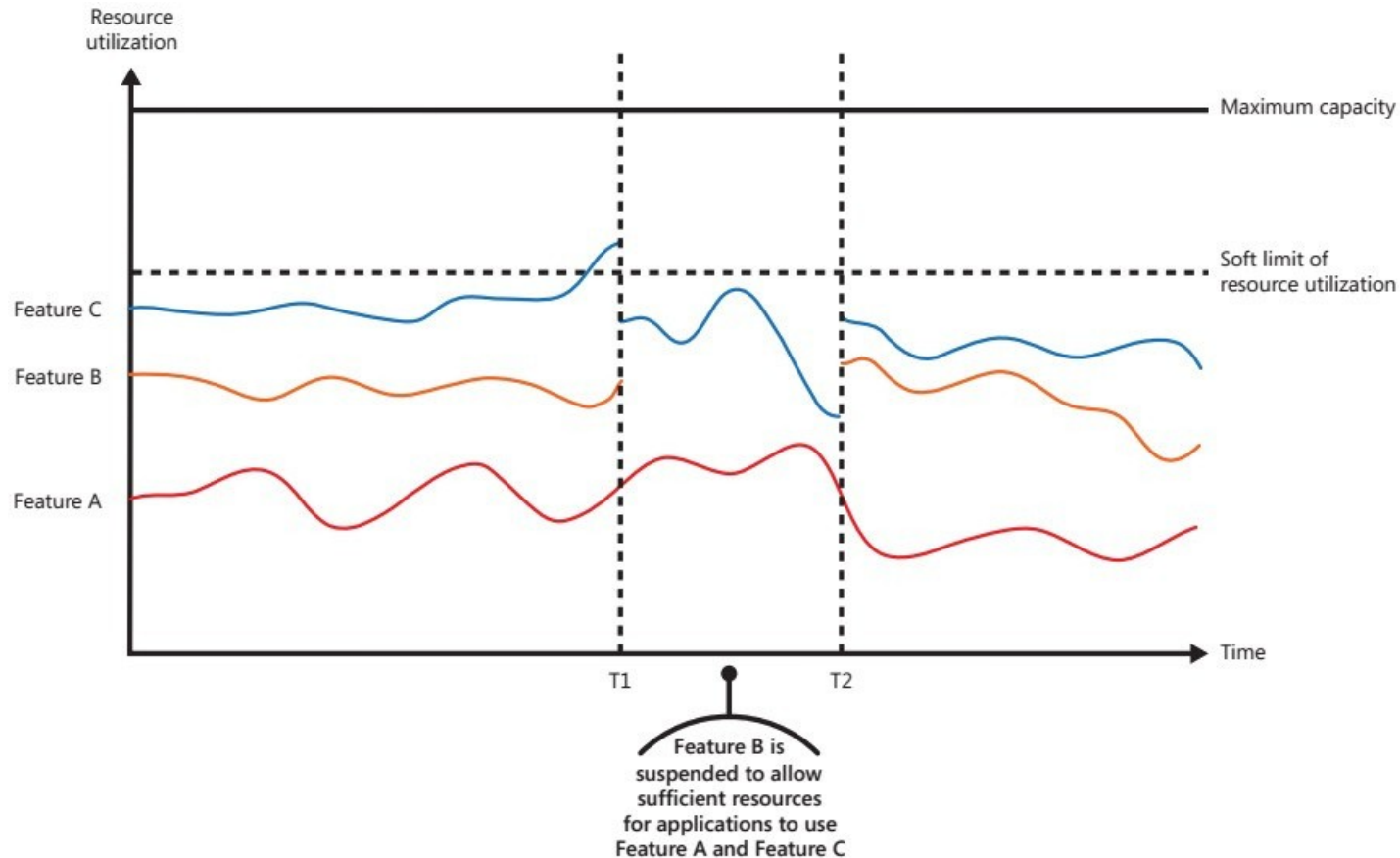
# Health Endpoint Monitoring

## Pattern



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

# Throttling Pattern



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