



# Cloud Design Patterns

Ravindu Nirmal Fernando

SLIIT | March 2025

# Design Patterns

---

A generally reusable solution to a recurring problem

- ✓ A template to solve the problem
- ✓ Best practices in approaching the
- ✓ problem Improve developer communication

# Cloud Application Development

---

## Issues

### Availability

- The guaranteed proportion of time that the system is functional

SLA – Service Level Agreement

Availability (%)	Downtime per
99	3.7 days
99.9	9 hours
99.95	4.4 hours
99.99	1 hour
99.999	5 minutes

# Cloud Application Development

---

## Issues

### Data Management

- ✓ Typically hosted in different locations and across multiple servers for performance, scalability and availability
- ✓ Maintaining consistency and synchronizing

### Design and Implementation

- ✓ Consistent and coherent component design
- ✓ Improves ease of deployment and maintenance
- ✓ Reusability of components

# Cloud Application Development

---

## Issues

### $\pi$ Messaging

- ✓ Messaging infrastructure to connect distributed components and services
- ✓ Asynchronous messaging

### $\pi$ Design and Implementation

- ✓ Consistent and coherent component design
- ✓ Improves ease of deployment and maintenance
- ✓ Reusability of components

# Cloud Application Development Issues

---

## $\pi$ Management and Monitoring

- ✓ Cloud applications run in in a remote servers with limited control

## $\pi$ Performance and Scalability

- ✓ Responsiveness of a system to execute any action within a given time interval
- ✓ Handle increases in load without impact on performance
- ✓ How to handle variable workloads?

# Cloud Application Development

---

## Issues

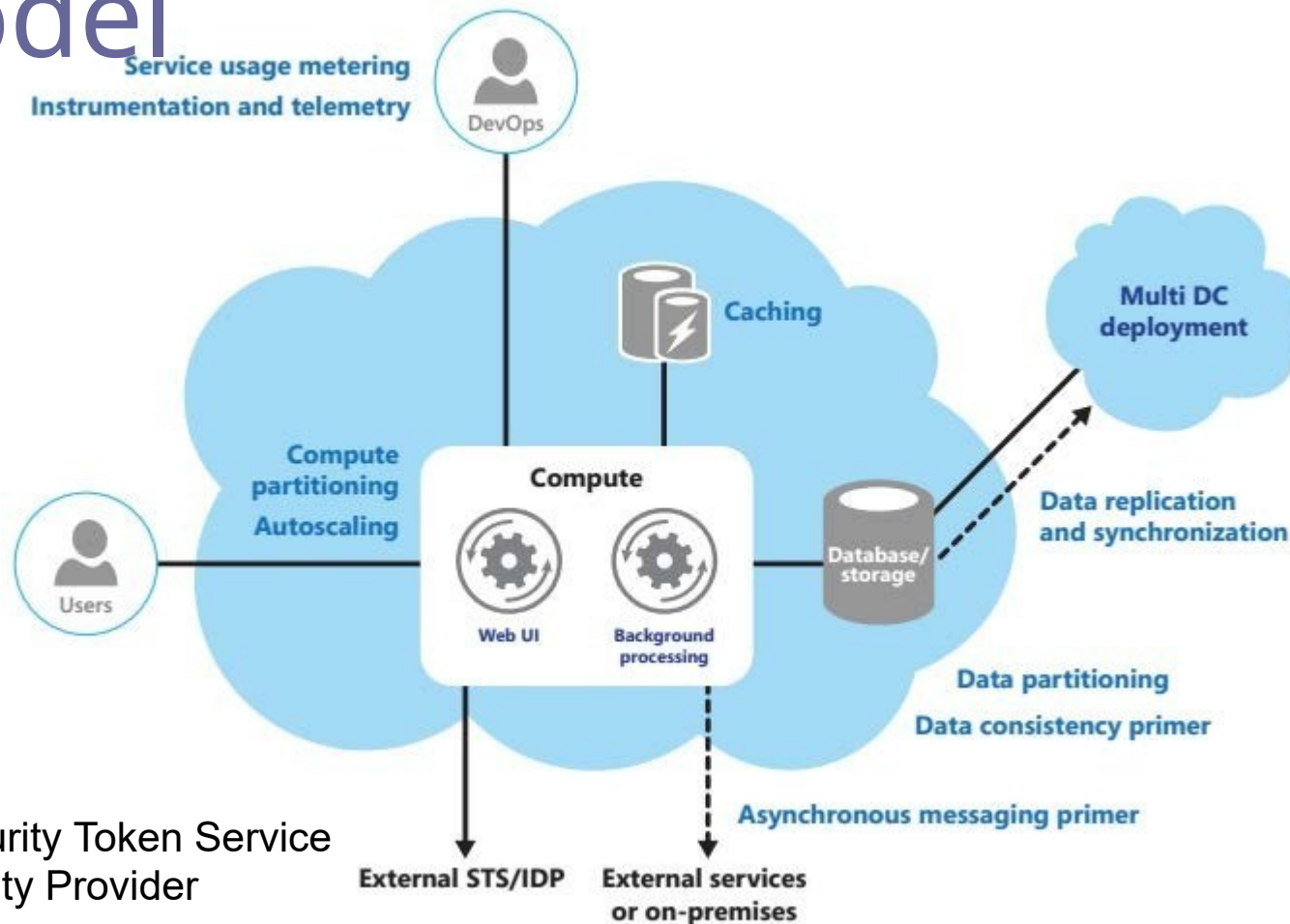
### $\pi$ Resiliency

- ✓ Ability of the application to gracefully handle and recover from failures
- ✓ Applications are more prone to failure in cloud environments

### $\pi$ Security

- ✓ Prevent malicious or accidental actions outside of the designed usage
- ✓ Prevent disclosure or loss of information

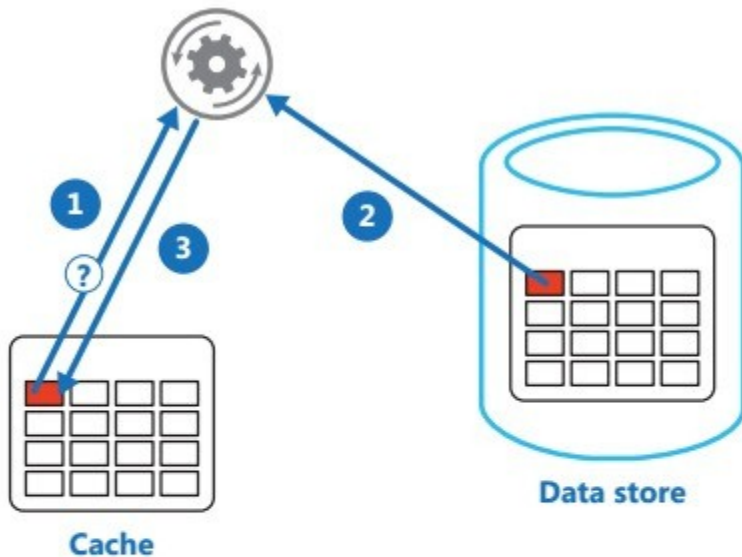
# High-Level Model



STS – Security Token Service  
IDP – Identity Provider



# Cache-Aside Pattern



- 1: Determine whether the item is currently held in the cache.
- 2: If the item is not currently in the cache, read the item from the data store.
- 3: Store a copy of the item in the cache.

- $\pi$  Load on demand data into a cache from a data store
- $\pi$  Pros
  - ✓ Increased performance
- $\pi$  Cons
  - ✓ Maintaining consistency between data in cache & data in underlying data store
- $\pi$  Solutions
  - ✓ Azure Cache AWS ElastiCache
  - ✓ Google App Engine memcache
  - ✓ Redis Cache
  - ✓

# Cache-Aside Pattern (Cont.)

---

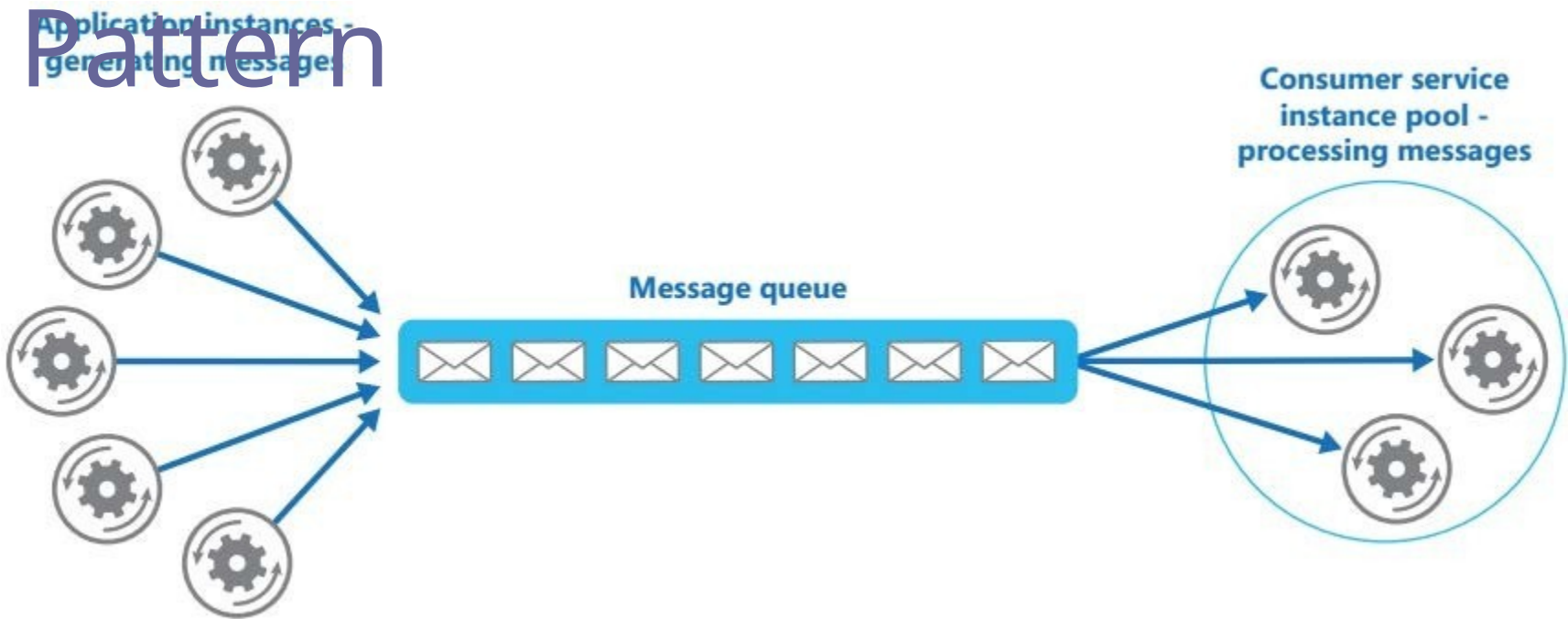
## $\pi$ When

- ✓ Read/write performance

## $\pi$ Parameters

- ✓ What to cache
- ✓ Lifetime of cached data
- ✓ Cache size
- ✓ Evicting data In Memory
- ✓ Caching

# Competing Consumers



$\pi$  Multiple concurrent consumers to process messages received on same channel

$\pi$  Goals

- ✓ Optimize throughput, improve scalability & availability, load balancing

# Competing Consumers Pattern

---

(Cont.)

$\pi$

When

- ✓ Independent tasks that can be processed parallel
- ✓ Volume of work is highly variable
- ✓ High availability

# Competing Consumers Pattern

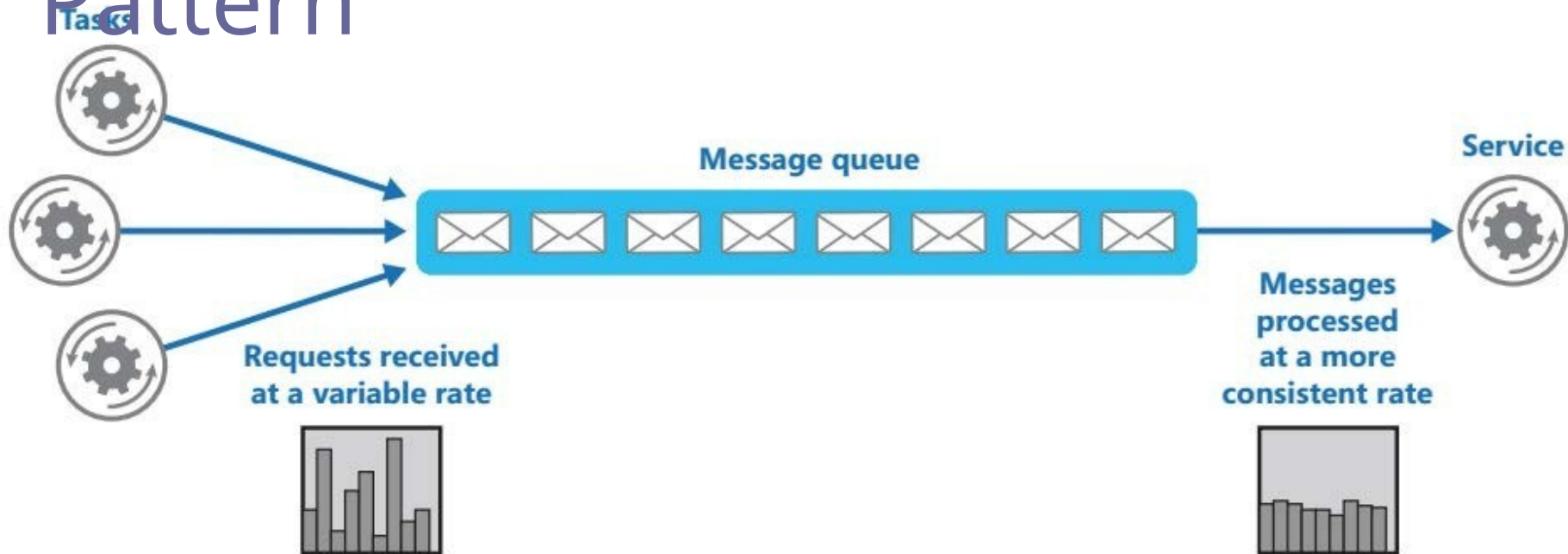
---

(Cont.)

## Parameters

- ✓ Queue size
- ✓ Scaling
- ✓ Not losing messages
- ✓ Preserving message ordering
- ✓ Resiliency
- ✓ Poison/malformed messages
- ✓ Returning results

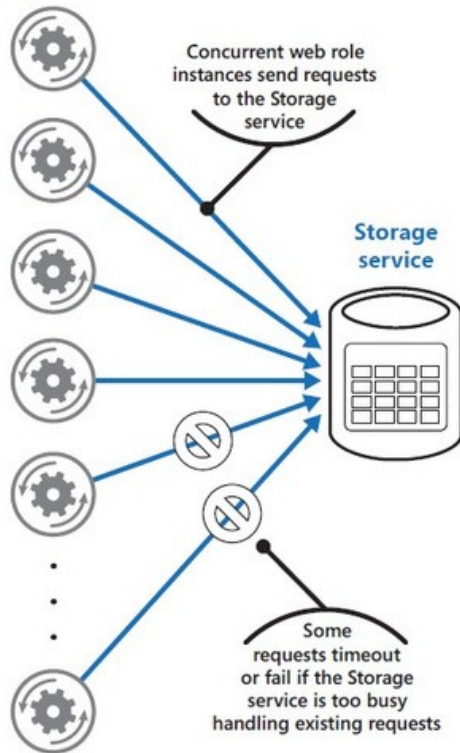
# Queue-Based Load Leveling Pattern



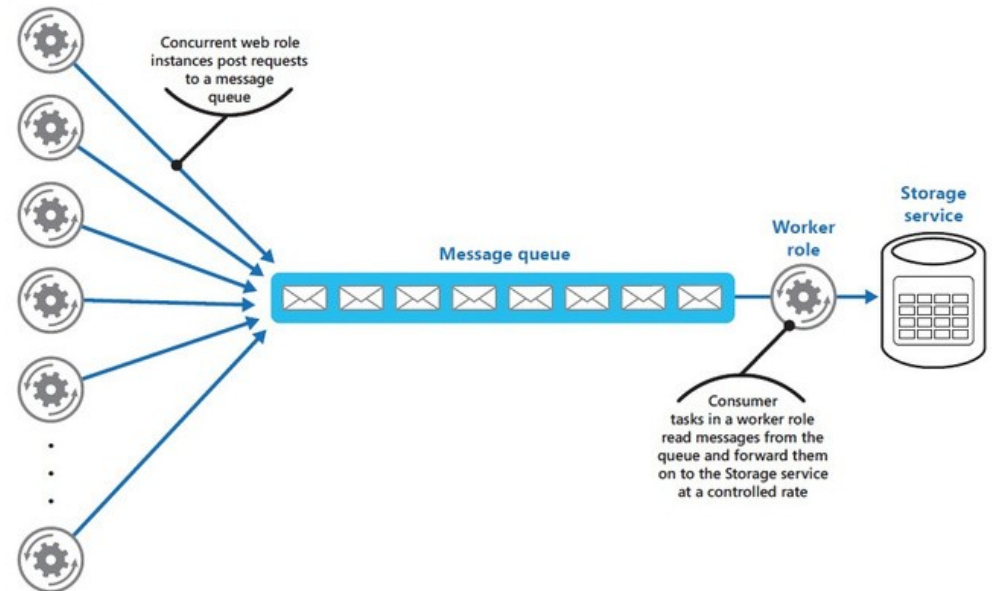
- $\pi$  To smooth intermittent heavy loads that may otherwise cause the service to fail or the task to time out

# Queue-Based Load Leveling Pattern

Web role instances

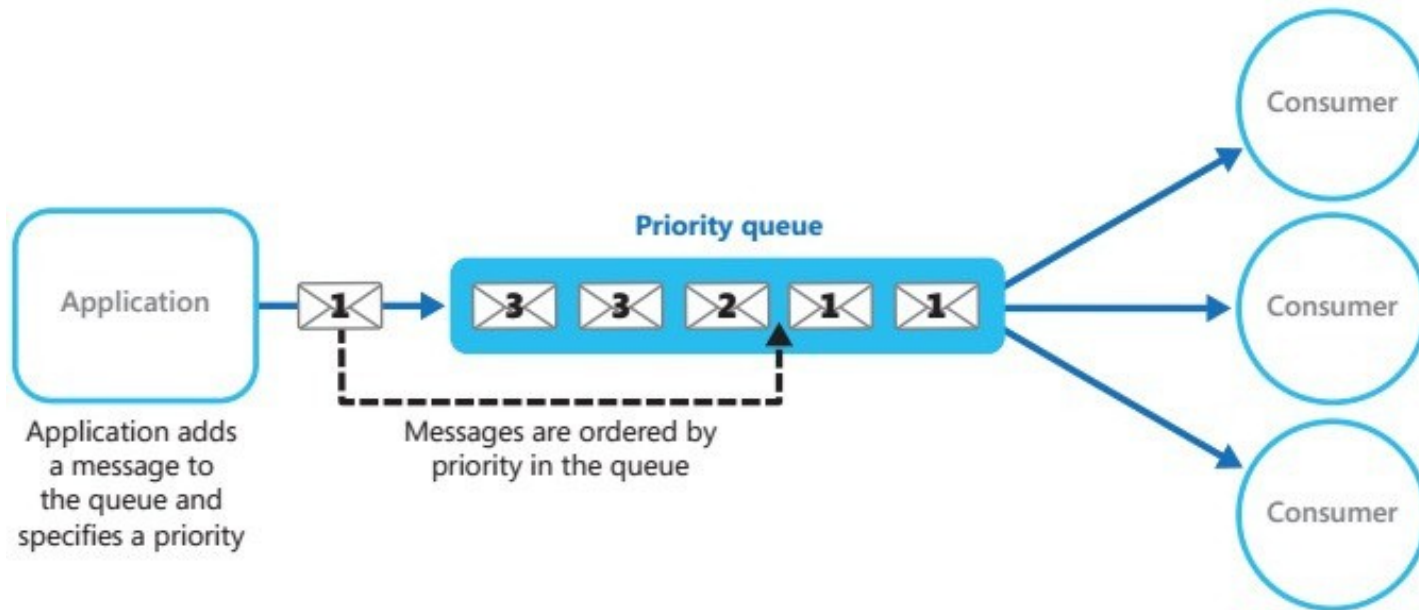


Web role instances



# Priority Queue Pattern

---



- $\pi$  Prioritize requests sent to services so that requests with a higher priority are received & processed quickly

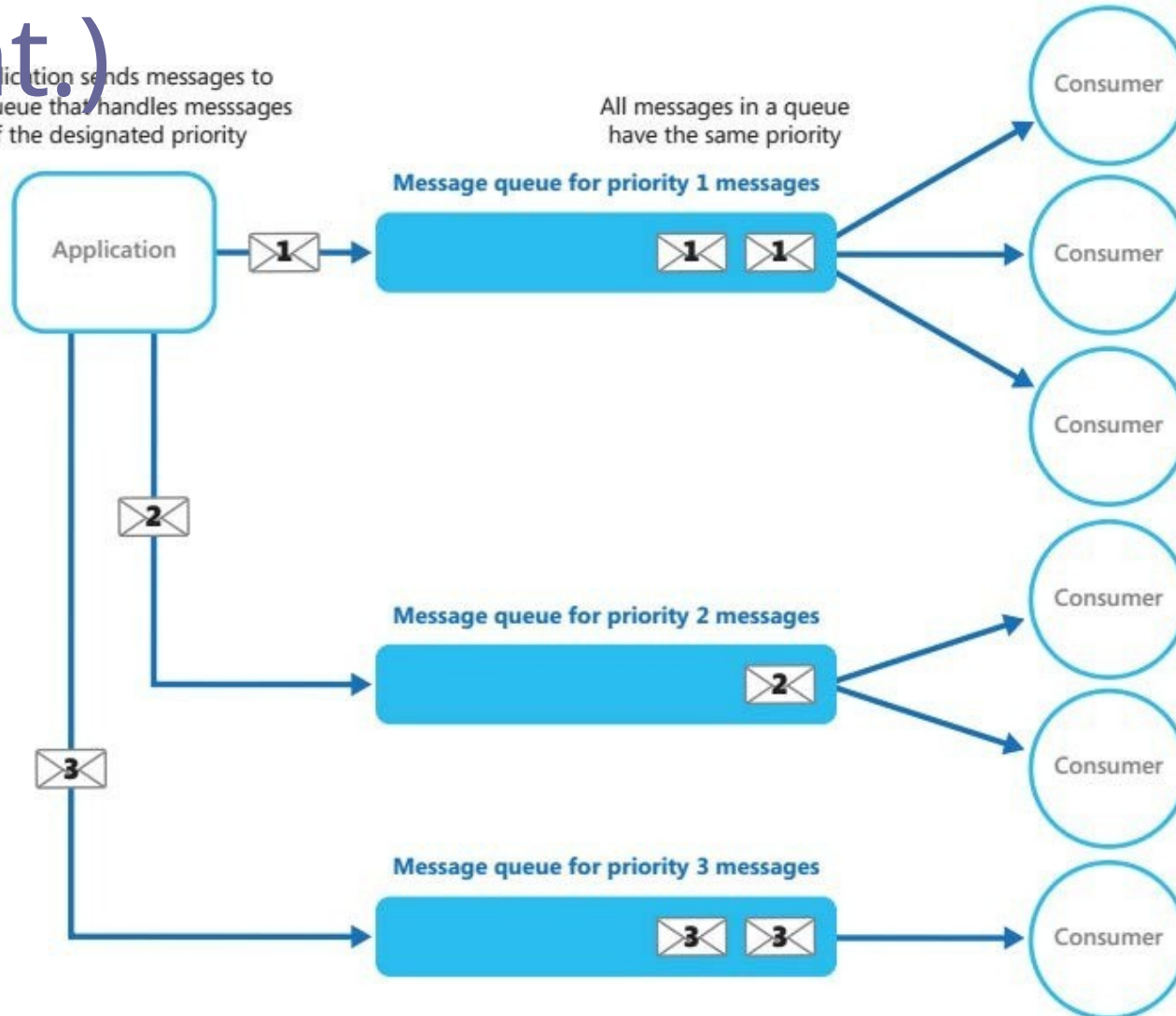


# Priority Queue Pattern

(Cont.)

Application sends messages to the queue that handles messages of the designated priority

All messages in a queue have the same priority

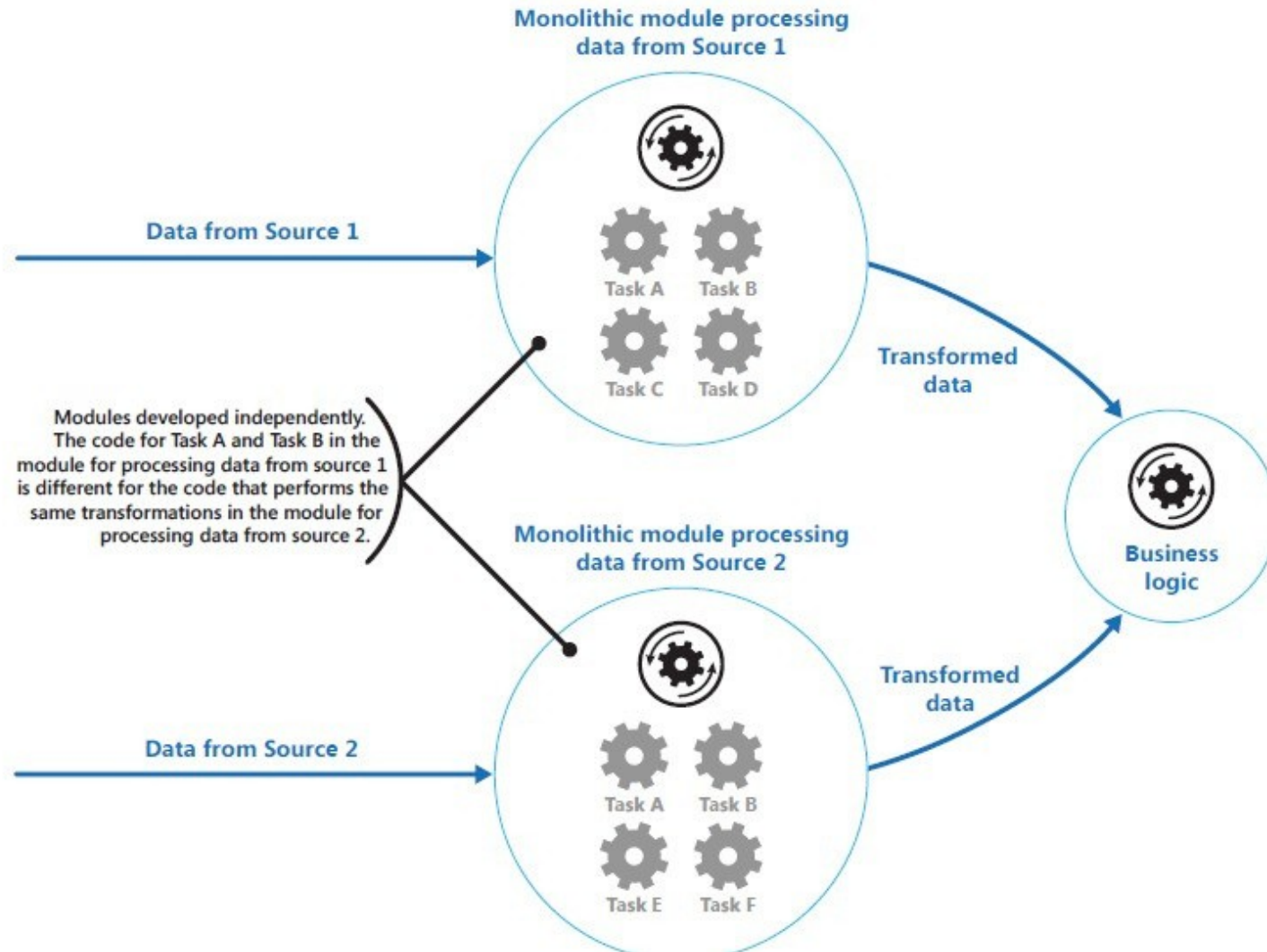


# Priority Queue Pattern (Cont.)

---

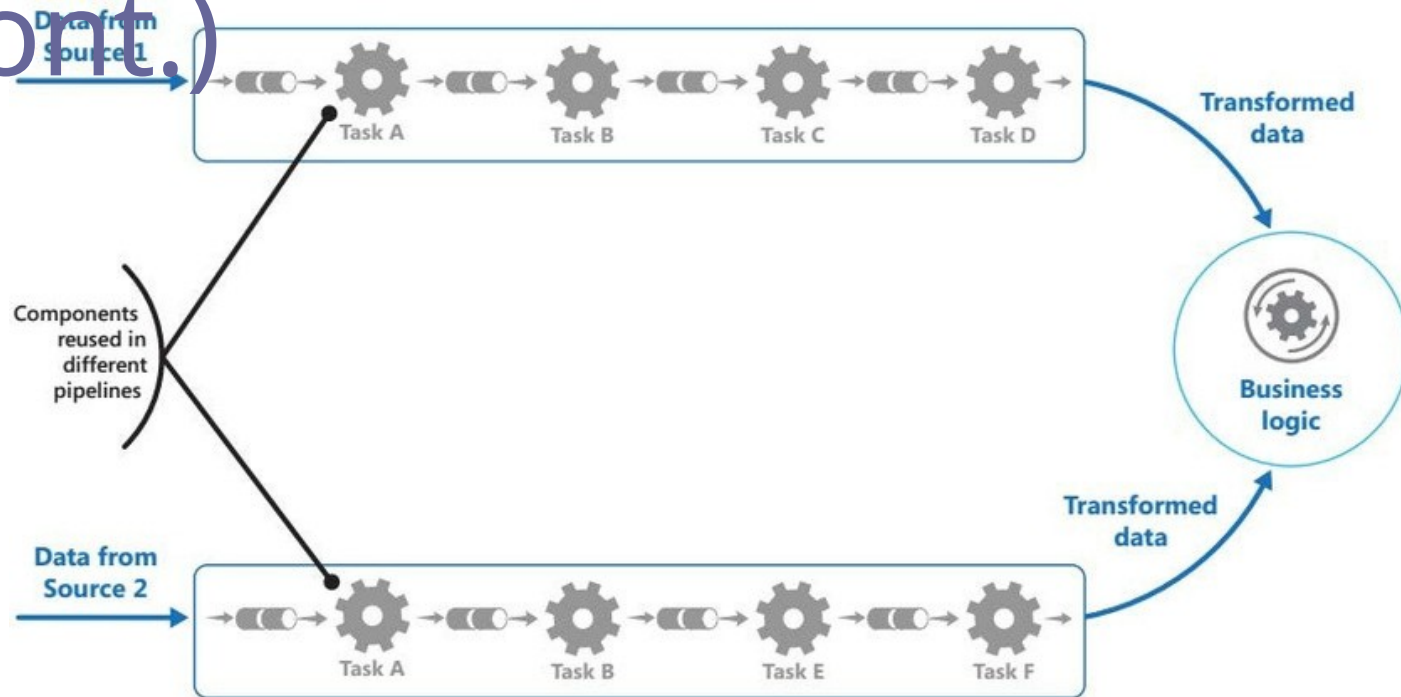
- $\pi$  When,
- ✓ The system handles multiple tasks that have different priorities
  - ✓ Different users should be served with different priorities

# Pipes & Filters Pattern



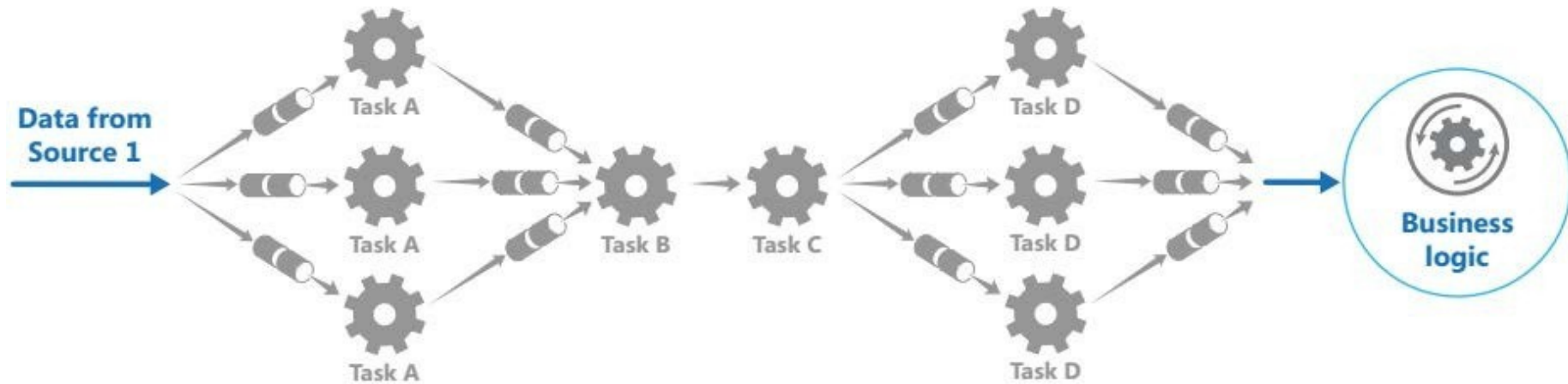
# Pipes & Filters Pattern

(Cont.)



- $\pi$  Decompose a task that performs complex processing into a series of discrete elements that can be reused

# Pipes & Filters Pattern – With Load Balancing



$\pi$ When,

- ✓ Application can be decomposed to steps
- ✓ Steps have different scalability requirements
- ✓ Flexibility of processing
- ✓ Need distributed processing