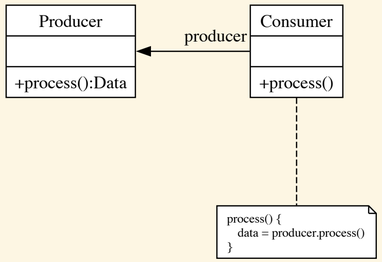
Object-Oriented Programming

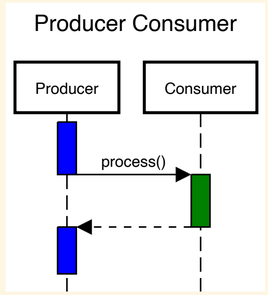
Design Pattern Message Queue

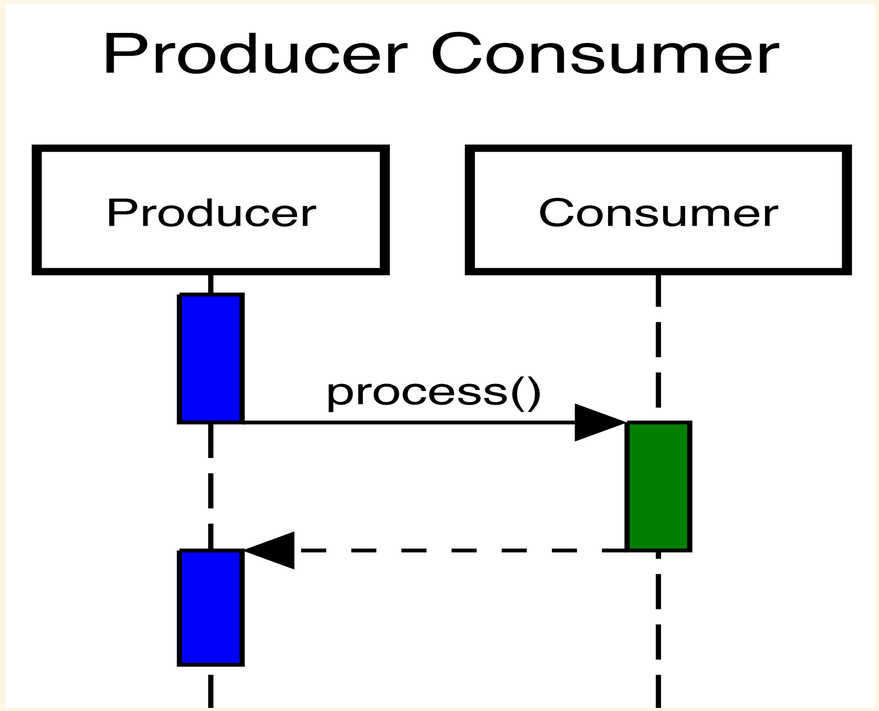
Michael L. Collard, Ph.D.

Department of Computer Science, The University of Akron

Motivation


Producer
+process():Data
producer
Consumer
+process()
process() {
data = producer.process()
}


Producer Consumer
Producer
Consumer
process()



Issues

- *Producer* has direct knowledge of the *Consumer* interface
- *Producer* blocks while *Consumer* processing occurs
- Does not easily support multiple Producers
- Does not easily support multiple Consumers

Message Queue

The message queue pattern decouples the sender and receiver of a message by introducing an intermediary component, the message queue.

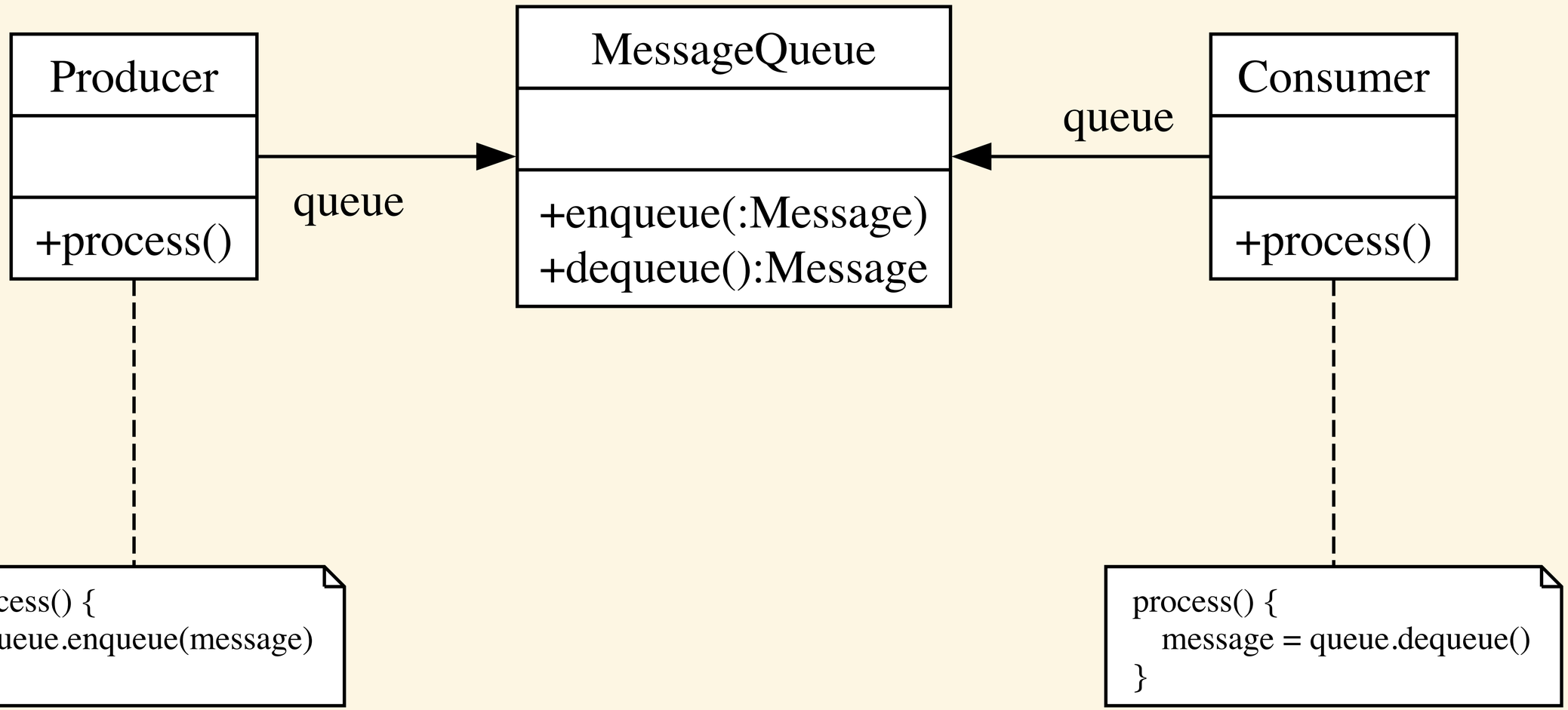
- Messages are placed in a queue by the *sender*

- Messages are retrieved from the queue by the *receiver*

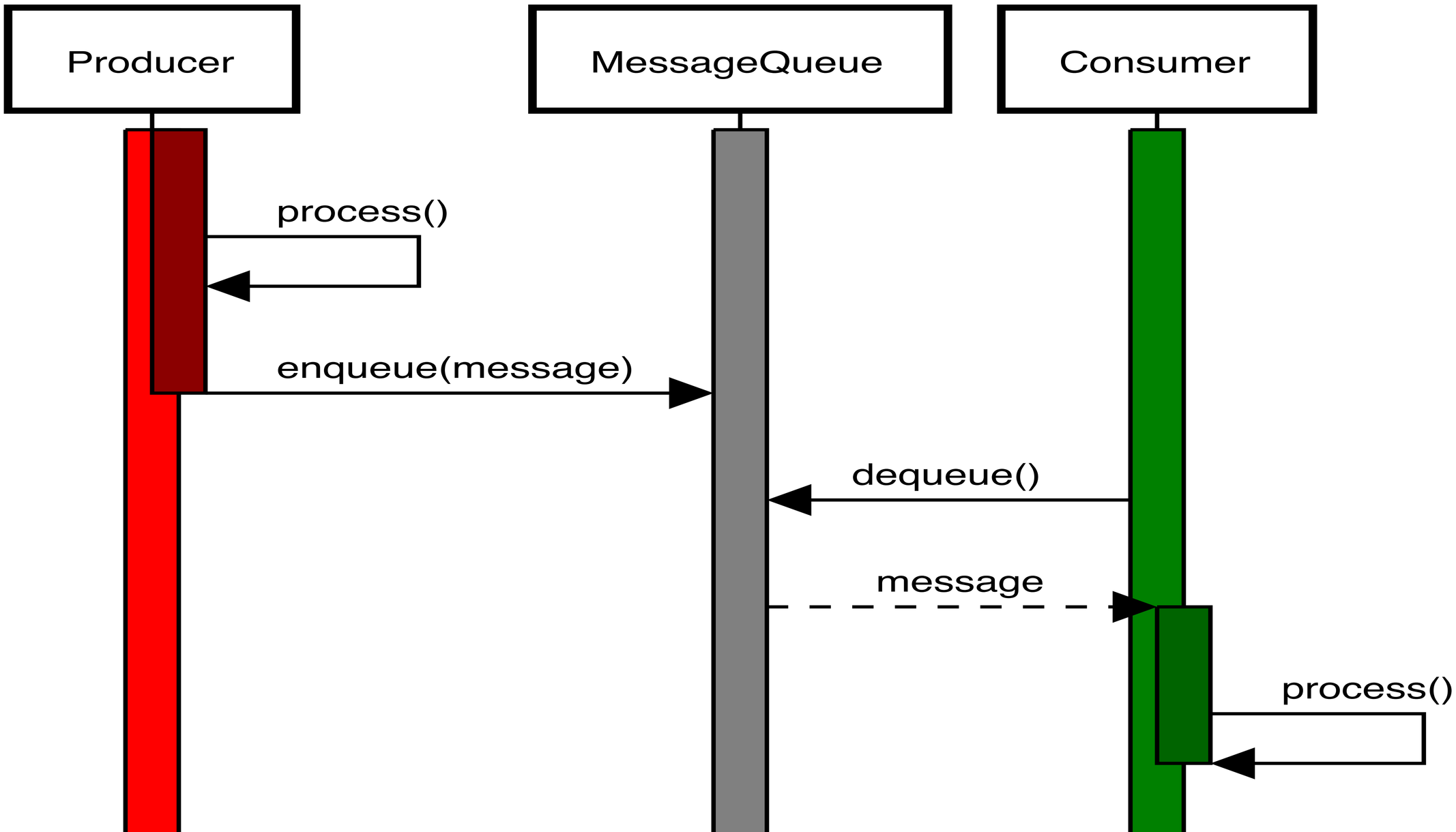
- Typically, the *sender* and *receiver* are in a separate *thread* or *process*

- This way, the sender and receiver are *asynchronous*

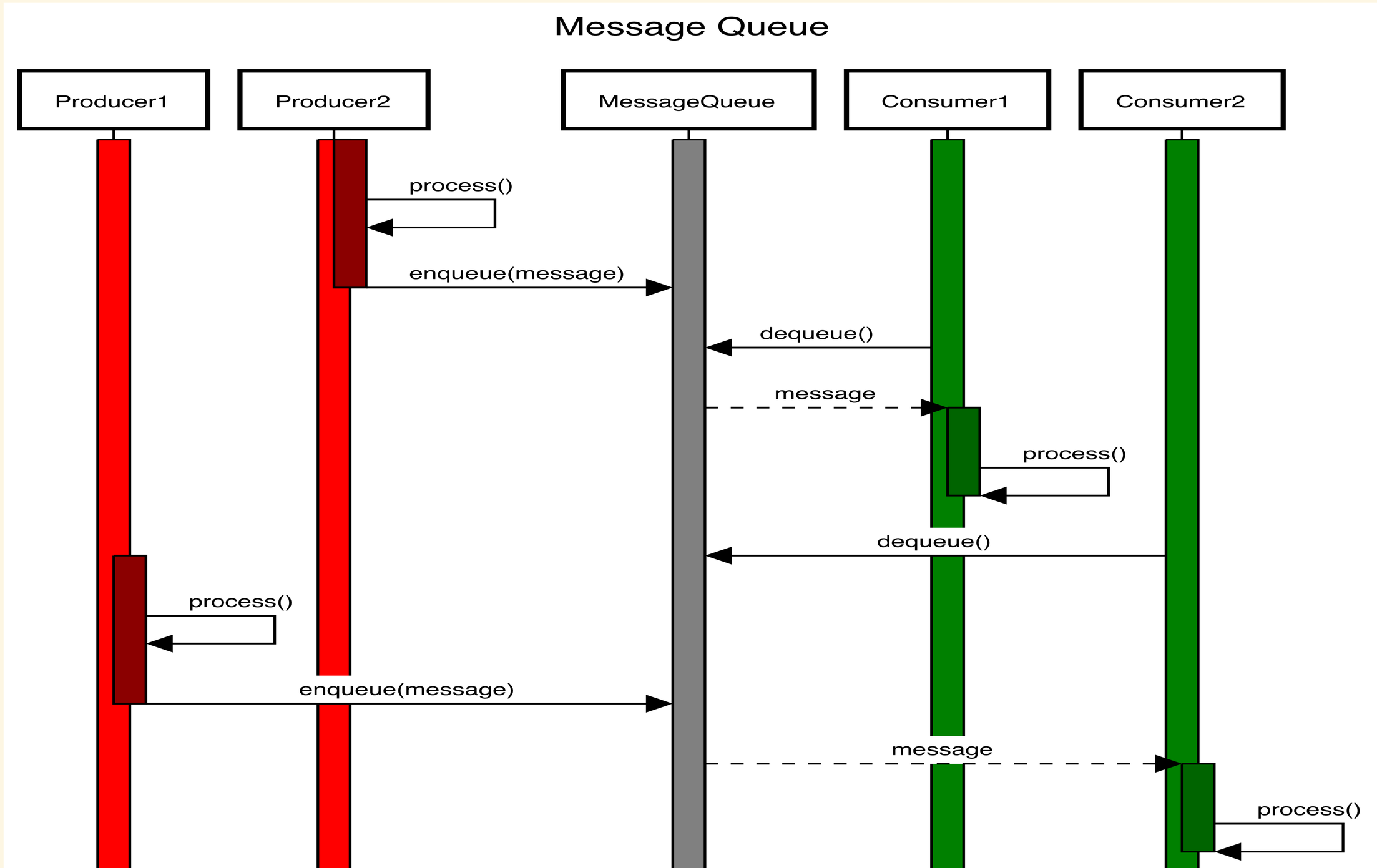
Message Queue



Message Queue



Message Queue

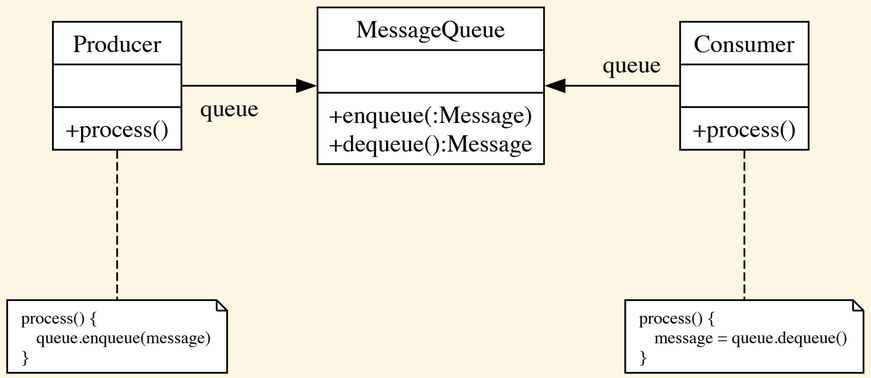


Implementation

The message queue can be implemented using a variety of technologies

- In-memory queues
- Disk-based queues
- Distributed message brokers

Components



- *Producers* The components responsible for generating and sending messages to the message queue

- *Consumers* The components responsible for processing messages received from the message queue

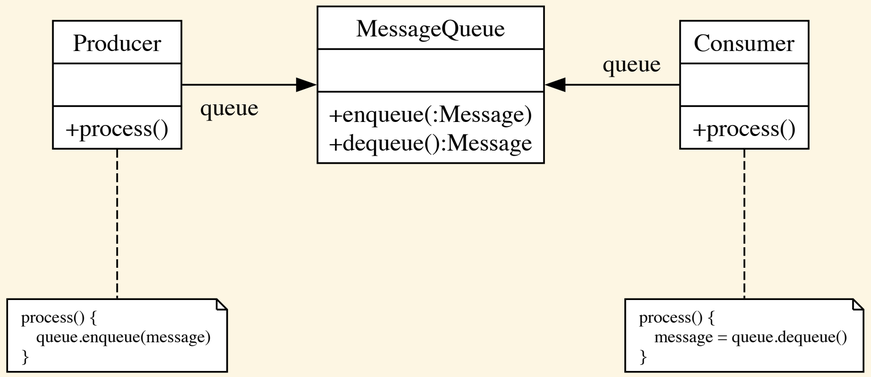
- *MessageQueue* A buffer-like data structure that stores and manages messages in a First-In-First-Out (FIFO) manner. It can be either in-memory or persistent.

- *Mess* that facili and

Key Concepts

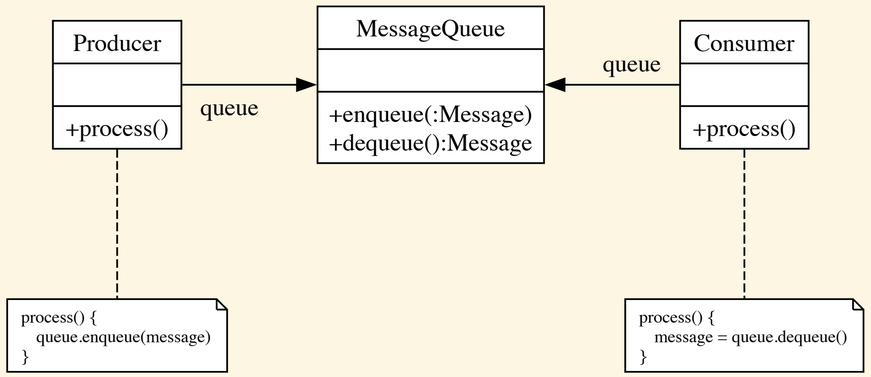
- **Asynchronous Communication** Message queues enable asynchronous communication, allowing producers and consumers to interact without waiting for each other to complete their tasks
- **Decoupling** Message queues promote the *separation of concerns* and reduce dependencies between components, making the system more maintainable and less prone to cascading failures
- **Scalability** Message queues can help to distribute workload evenly among consumers, which can be scaled horizontally to handle increased demand
- **Resilience** Message fault tolerance by en not lost if a consume unavailable
- **Routing and Filterin** provide advanced ro capabilities, allowing specific consumers b
- **Load Balancing** Mes the workload among preventing overload system performance

Advantages



- **Improved system responsiveness** By using asynchronous communication, components can continue processing other tasks without waiting for responses, resulting in a more responsive system
- **Flexibility** Decoupling allows for independent development, deployment, and scaling of components, making it easier to adapt to changing requirements
- **Enha** can b or w unav and i
- **Simp** erro bett mec

Disadvantages



- **Increased complexity** Implementing a message queue pattern can introduce additional complexity to a system, such as managing message brokers and ensuring message delivery
- **Deb** and **using** chall natu
- **Latency** Asynchronous communication can introduce some latency in processing messages, which might not be suitable for time-sensitive applications