COMP2511

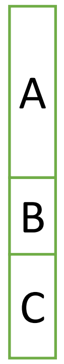
Synchronous vs Asynchronous Software Design

Prepared by

Dr. Ashesh Mahidadia

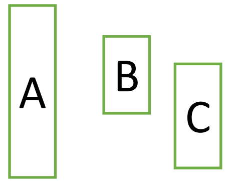
What is Synchronous programming?

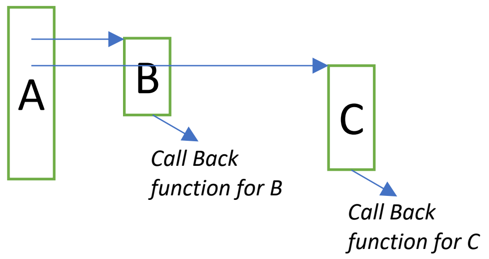
- In *synchronous* programming, operations are carried out in order.
- The execution of an operation is dependent upon the completion of the preceding operation.
- Tasks (functions) A, B, and C are executed in a sequence, often using one thread.



What is Asynchronous programming?

- In *asynchronous programming*, operations are carried out independently.
- The execution of an operation is not dependent upon the completion of the preceding operation.
- Tasks (functions) A, B, and C are executed independently, can use multiple threads/resources.





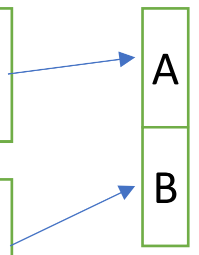
Example: Synchronous vs Asynchronous programming

Synchronous

```
function getRecord(key) {
    establish database connection
    retrieve the record for key
   return record;
}

function display(rec){
   display rec on the web page
}
```

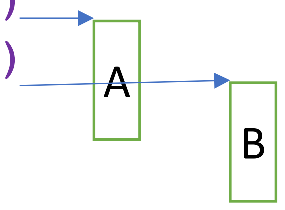
```
rec = getRecord('Rita’);
display(rec)
```

```
rec = getRecord('John');
display(rec)
```



Asynchronous

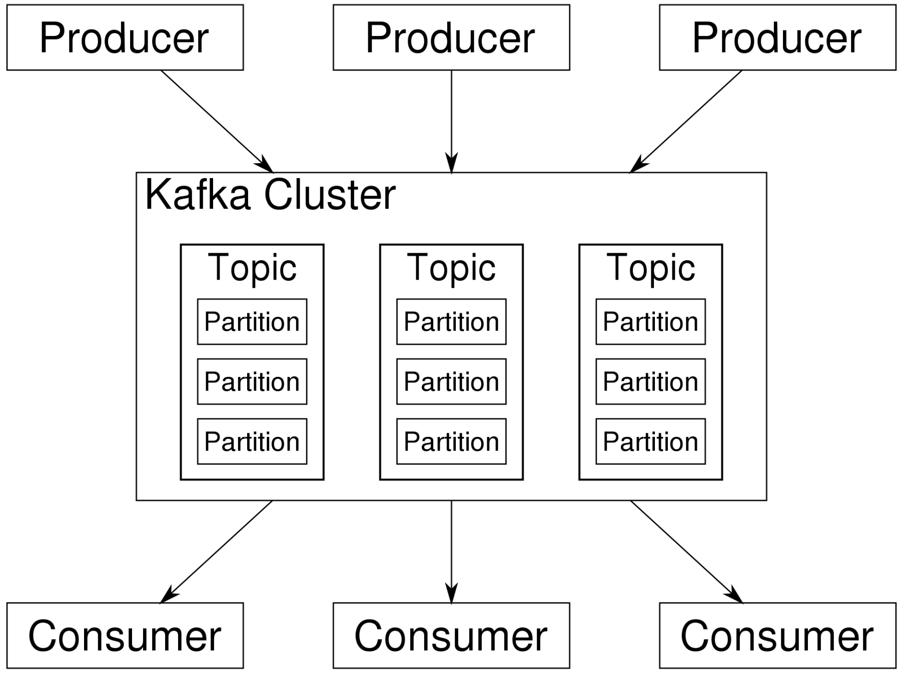
```
function getRecord(key, callback) {
    establish database connection
    retrieve the record for key
   callback(record);
}

function display(rec){
   display rec on the web page
}

getRecord(‘Rita’, display)
getRecord(‘John’, display)
```



Kafka: An Example of Asynchronous Software Design

- Today, streams of data records, including streams of events, are continuously generated by many online applications.
- A streaming platform enables the development of applications that can continuously and easily consume and process streams of data and events.
- Apache Kafka (Kafka) is a free and open-source distributed streaming platform useful for building, *real time* or *asynchronous*, event-driven applications.
- Kafka offers loose coupling between *producers* and *consumers.*
- Consumers have the option to either consume an event in real time or *asynchronously* at a later time.
- Kafka maintains the chronological order of records/events, ensuring fault tolerance and durability.
- To increase scalability, Kafka separates a topic and stores each partition on a different node.

- *Producer API* – Permits an application to *publish* streams of records/events.
- *Consumer API* – Permits an application to *subscribe* to topics and processes streams of records/events.


Producer
Producer
Producer
Kafka Cluster
Topic
Partition
Partition
Partition
Topic
Partition
Partition
Partition
Topic
Partition
Partition
Partition
Consumer
Consumer
Consumer