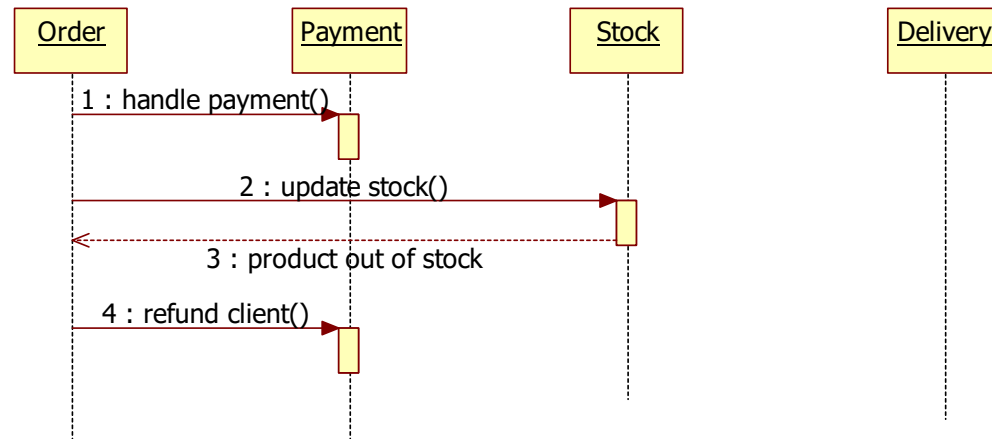
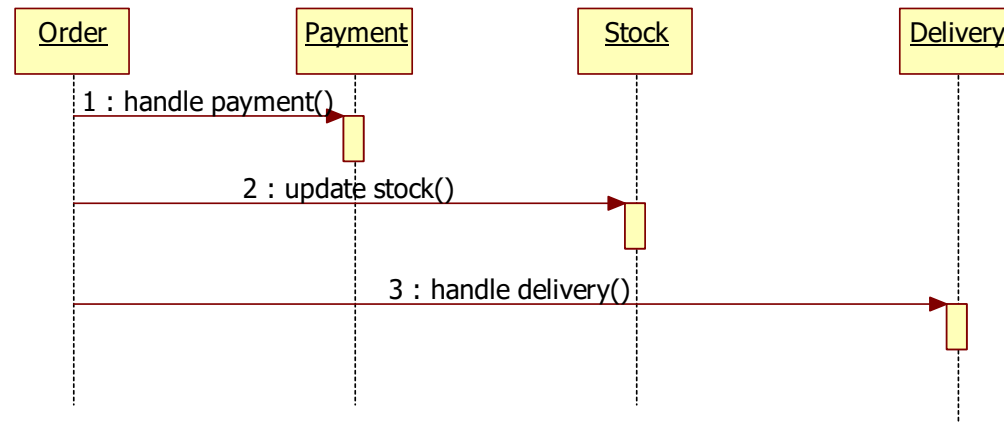


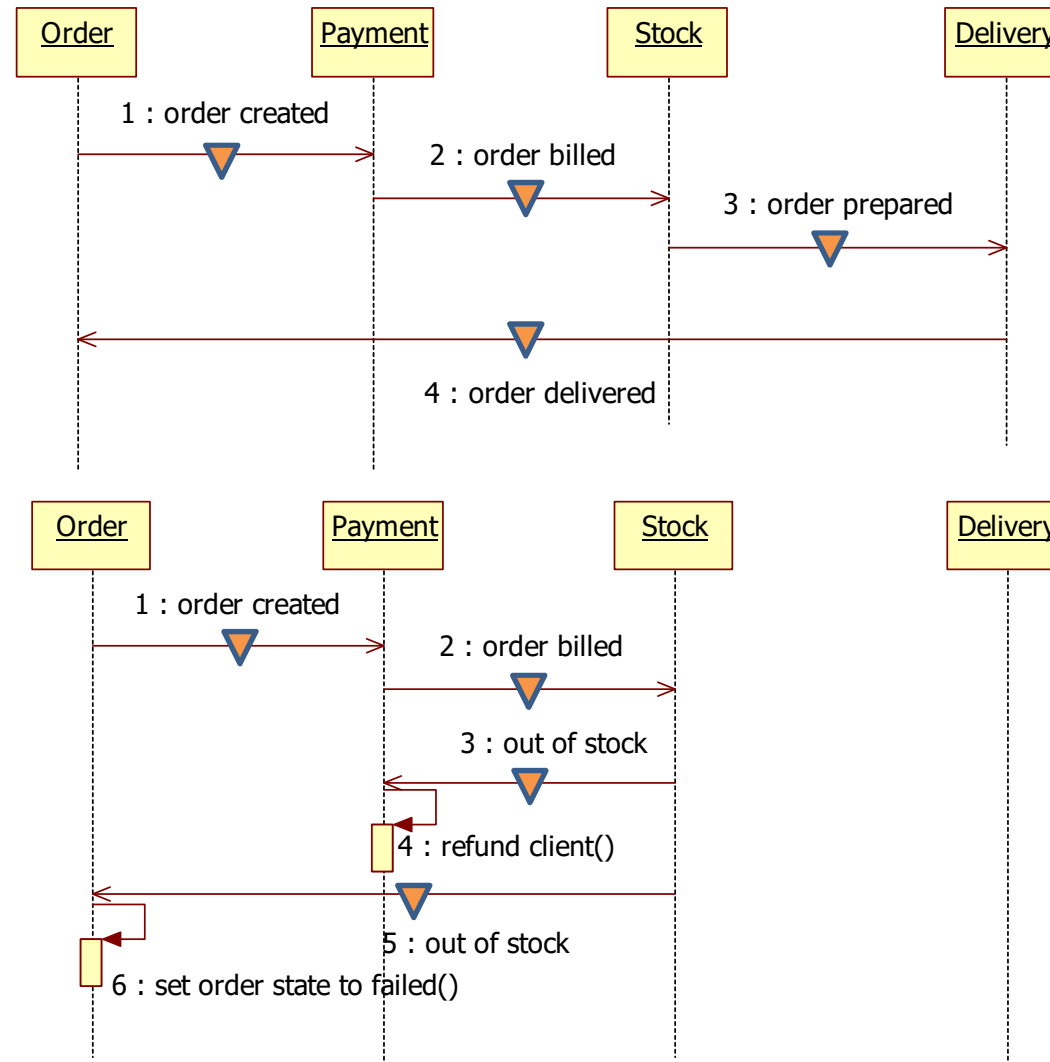
# EDA



# Synchronous (REST) calls



# Asynchronous events (messaging)

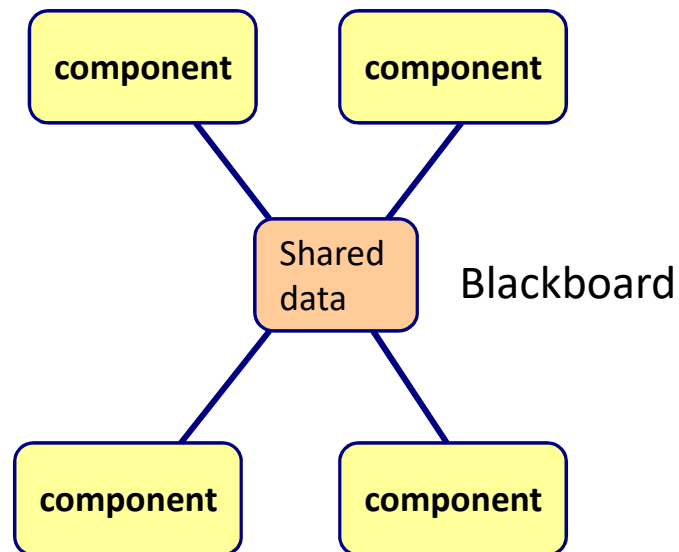


# BLACKBOARD



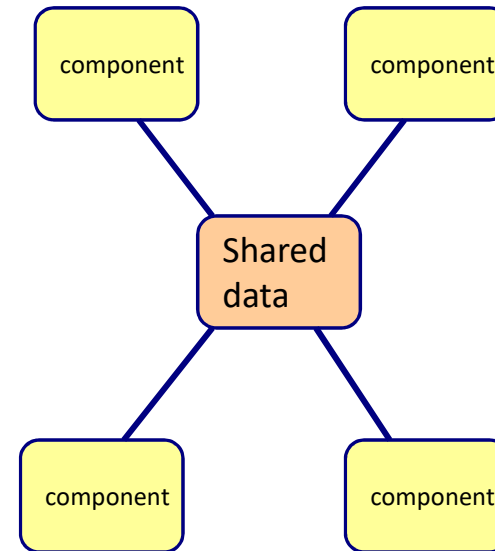
# Blackboard

- Used for non deterministic problems
  - There is no fixed straight-line solution to a problem
- Every component adds her information on the blackboard



# Blackboard

- Common data structure
  - Extension is no problem
  - Change is difficult
- Easy to add new components
- Tight coupling for data structure
- Loose coupling for
  - Location
  - Time
  - Technology(?)
- Synchronisation issues



# Blackboard

## ■ Benefits

- Easy to add new components
- Components are independent of each other
- Components can work in parallel

## ■ Drawbacks

- Data structure is hard to change
  - All components share the same data structure
- Synchronization issues



# EVENT SOURCING





# Store the state of a system

---



## Structural representation

List of ordered goods

Payment information

Shipping information





# Store the events of a system

---



## Event representation

-  Add item #1
-  Add item #2
-  Add payment info
-  Update item #2
-  Remove item #1
-  Add shipping info



# Event sourcing

---

- Instead of storing the state of an entity in a database, you store the series of events that lead up to the state.
- Storing all of the events increases the analytical capabilities of a business.
- Instead of just asking what the current state of an entity is, a business can ask what the state was at any time in the past



# Event sourcing

---

- For each aggregate
  - Identify (state changing) domain events
  - Define event classes
- Example:
  - Shopping cart
    - ItemAddedEvent
    - ItemRemovedEvent
    - CheckedOutEvent
  - Order
    - OrderCreated
    - OrderApproved
    - OrderShipped



# Storing events

## ■ Traditional

ID	status	data...
101	accepted	...

Store entity data

## ■ Event sourcing

Entity ID	Entity type	Event ID	Event type	Event data...
101	Order	901	OrderCreated	...
101	Order	902	OrderApproved	...
101	Order	903	OrderShipped	...

Store state changing events



# Advantages of storing events

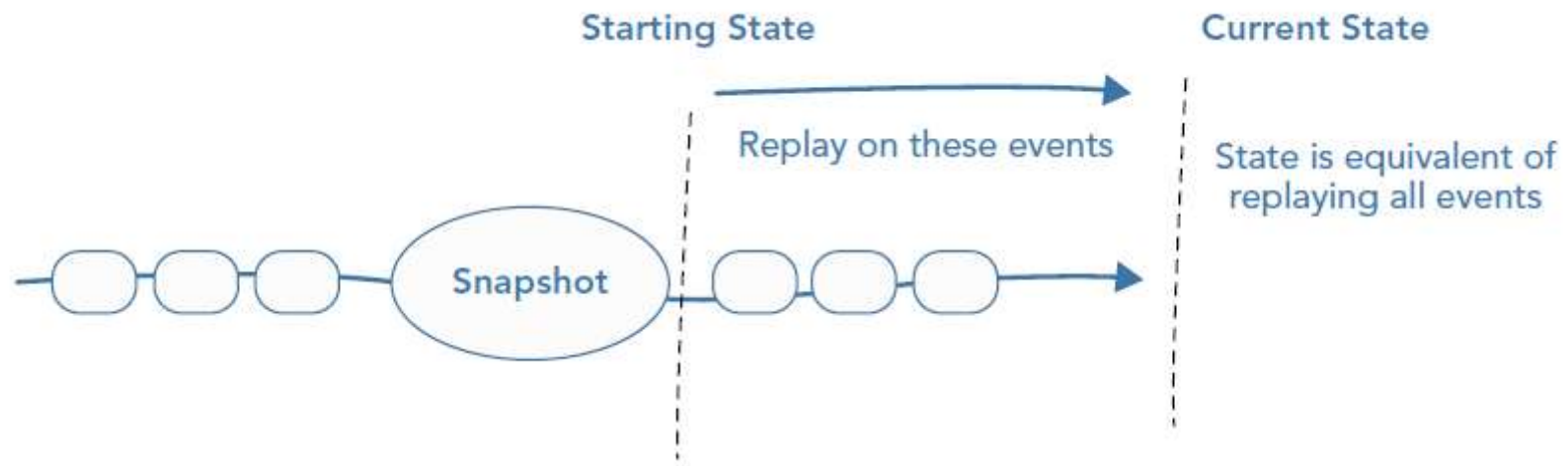
---

- You don't miss a thing
  - Business can analyze history of events
  - Bugs can be solved easier
- Can be replayed
- Events are immutable



# Snapshots

- Intermediate steps in an event stream that represent the state after replaying all previous events
  - Can increase performance when streams are very long



# STREAM BASED ARCHITECTURE





# Stream based systems

---

- Continuous stream of data
  - Stock market systems
  - Social networking systems
  - Internet of Things (IoT) systems
  - Systems that handle sensor data
  - System that handle logfiles
  - Systems that monitor user clicks
  - Car navigator software



# But also

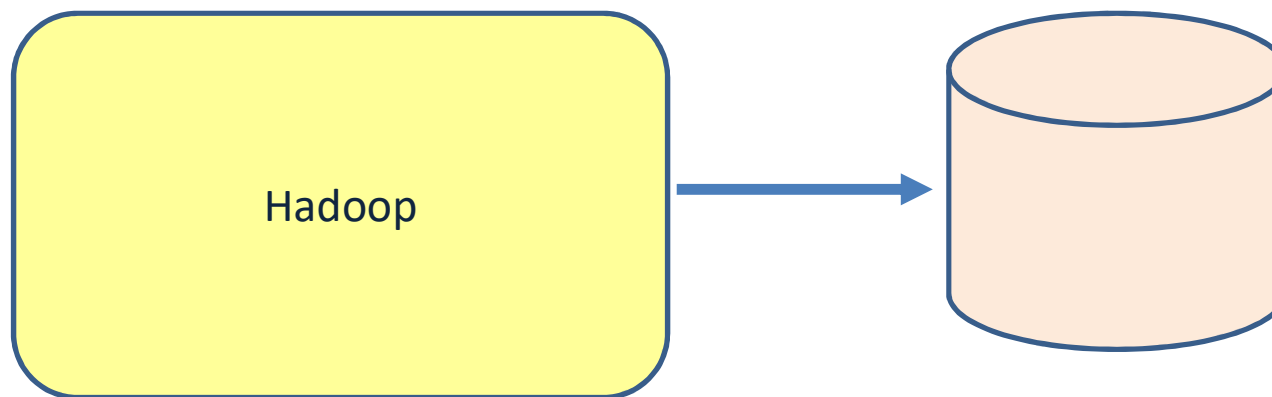
- 
- Stream of purchases in web shop
  - Stream of transactions in a bank
  - Stream of actions in a multi user game
  - Stream of bookings in a hotel booking system
  - Stream of user actions on a web application
  - ...



# Batch processing

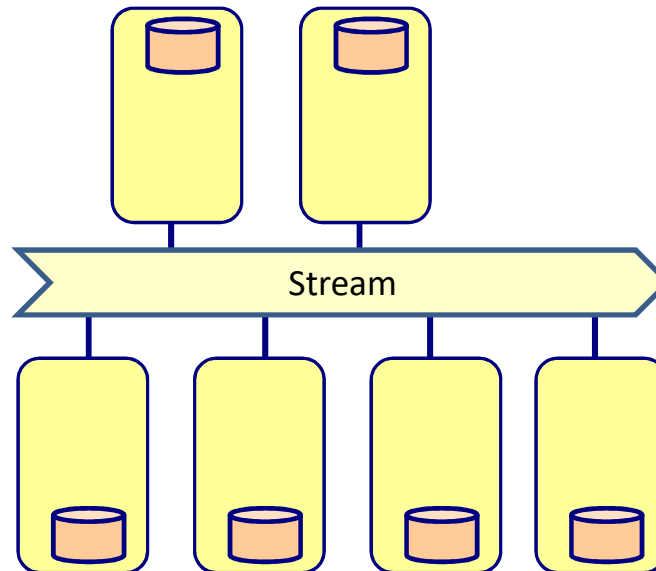
---

- First store the data in the database
- Then do queries (map-reduce) on the data
- Queries over all or most of the data in the dataset.
- Latencies in minutes to hours



# Stream processing

- Handle the data when it arrives
- Handle event (small data) by event
- Latencies in seconds or milliseconds

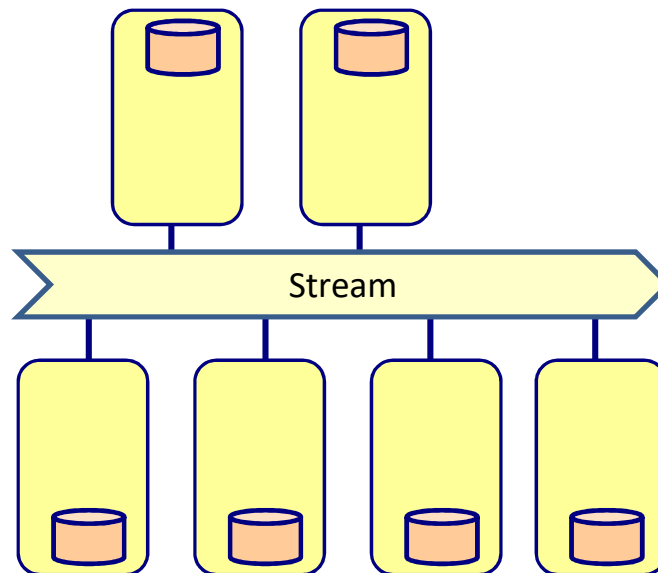


# Stream based architecture

---

Works good for applications with:

1. high volume data
2. high frequency changes



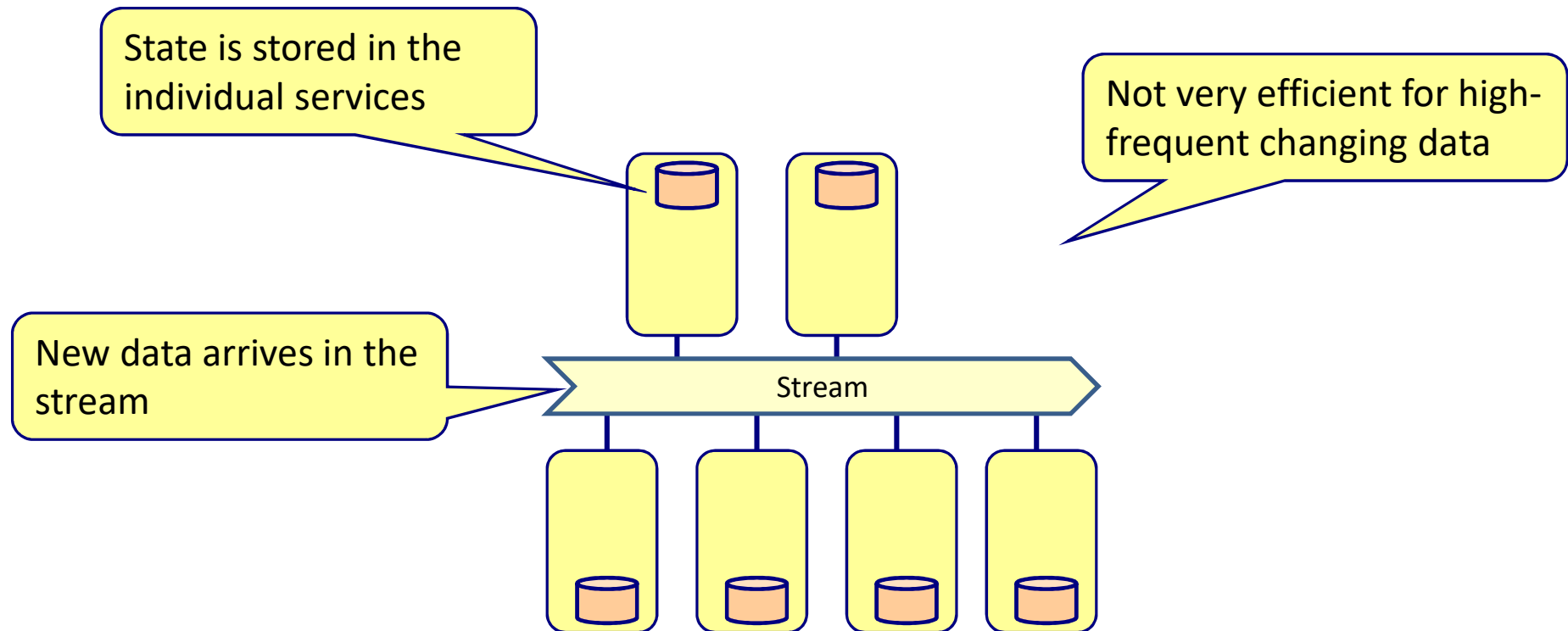
# Stream



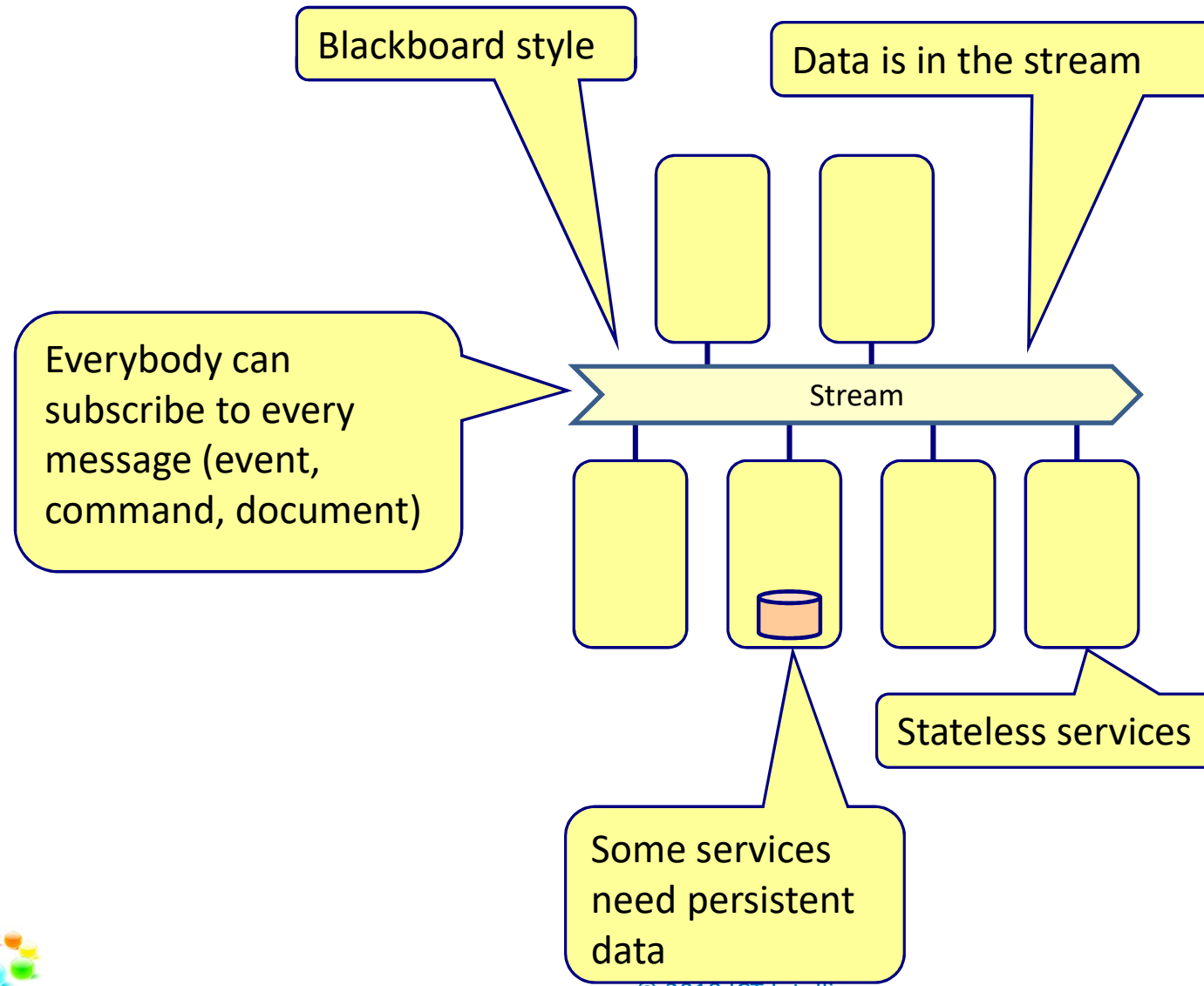
- Stream is just a sequence of events
- Implemented with a distributed messaging system
  - Kafka
  - MapR



# Where is the data?

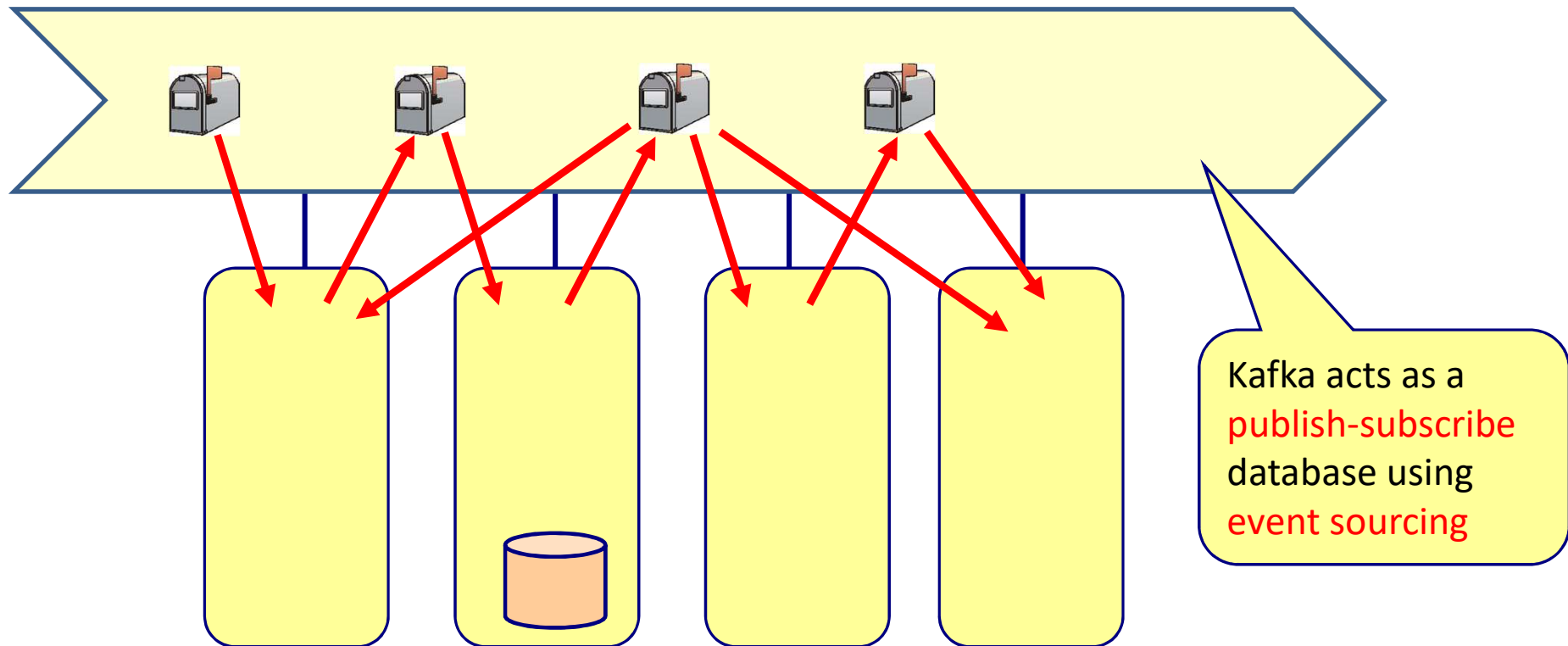


# Where is the data?

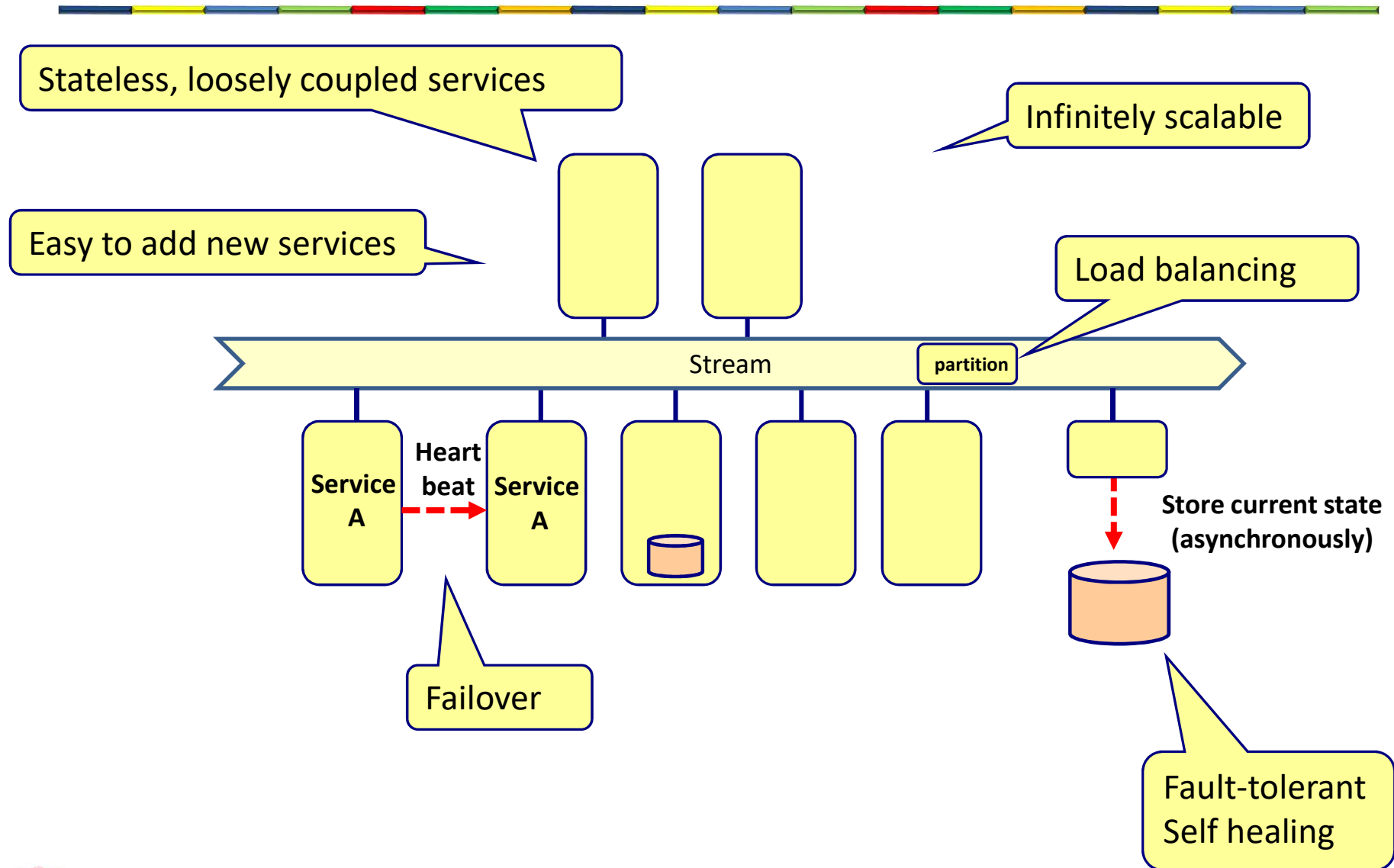




# Publish-subscribe and event sourcing



# Stream based architecture



# KAFKA



# Kafka producer: sending an object

```
@Service
public class Sender {
    @Autowired
    private KafkaTemplate<String, Person> kafkaTemplate;

    @Value("${app.topic.greetingtopic}")
    private String topic;

    public void send(Person person){
        System.out.println("sending person="+person.getFirstName()+" "
                           +person.getLastName()+" to topic="+ topic);
        kafkaTemplate.send(topic, person);
    }
}
```

```
public class Person {

    private String firstName;
    private String lastName;
    ...
}
```



# Kafka consumer: receiving an object

```
@Service
public class Receiver {

    @KafkaListener(topics = "${app.topic.greetingtopic}")
    public void receive(@Payload Person person,
                       @Headers MessageHeaders headers) {
        System.out.println("received message="+ person.getFirstName()+" "
                           +person.getLastName());
    }
}
```

```
public class Person {

    private String firstName;
    private String lastName;
    ...
}
```



# The configuration

---

## application.properties

```
spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.consumer.group-id= gid
spring.kafka.consumer.auto-offset-reset= earliest
spring.kafka.consumer.key-deserializer=
org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer=
org.springframework.kafka.support.serializer.JsonDeserializer
spring.kafka.producer.key-serializer=
org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer=
org.springframework.kafka.support.serializer.JsonSerializer
spring.kafka.consumer.properties.spring.json.trusted.packages=kafka

app.topic.greetingtopic= greetingtopic
```

JsonSerializer  
and  
JsonDeserializer

Add trusted  
packages for  
consumer

