Lecture 8

Events, Alerts and Notifications

The concept

- Event: a change occurring in some context or state.
- For example, in interactive applications, actions performed by the user on objects are events that cause changes in the objects that maintain the state of the application; the objects responsible for displaying a view of the current state are notified whenever the state changes.
- For distributed event-based systems, multiple objects placed at different locations can be notified of events taking place at an object.

The publish-subscribe paradigm

- An object that generates events publishes the type of events that it will make available for observation by other objects.
- Objects that want to receive notifications of events published by an object subscribe to the types of events that are of interest to them.
- Objects that represent events are called notifications. Notifications may be stored, sent in messages, queried etc.
- An event particularly relevant becomes an alert.

- When a publisher experiences an event, subscribers that expressed an interest in that type of event will receive notifications.

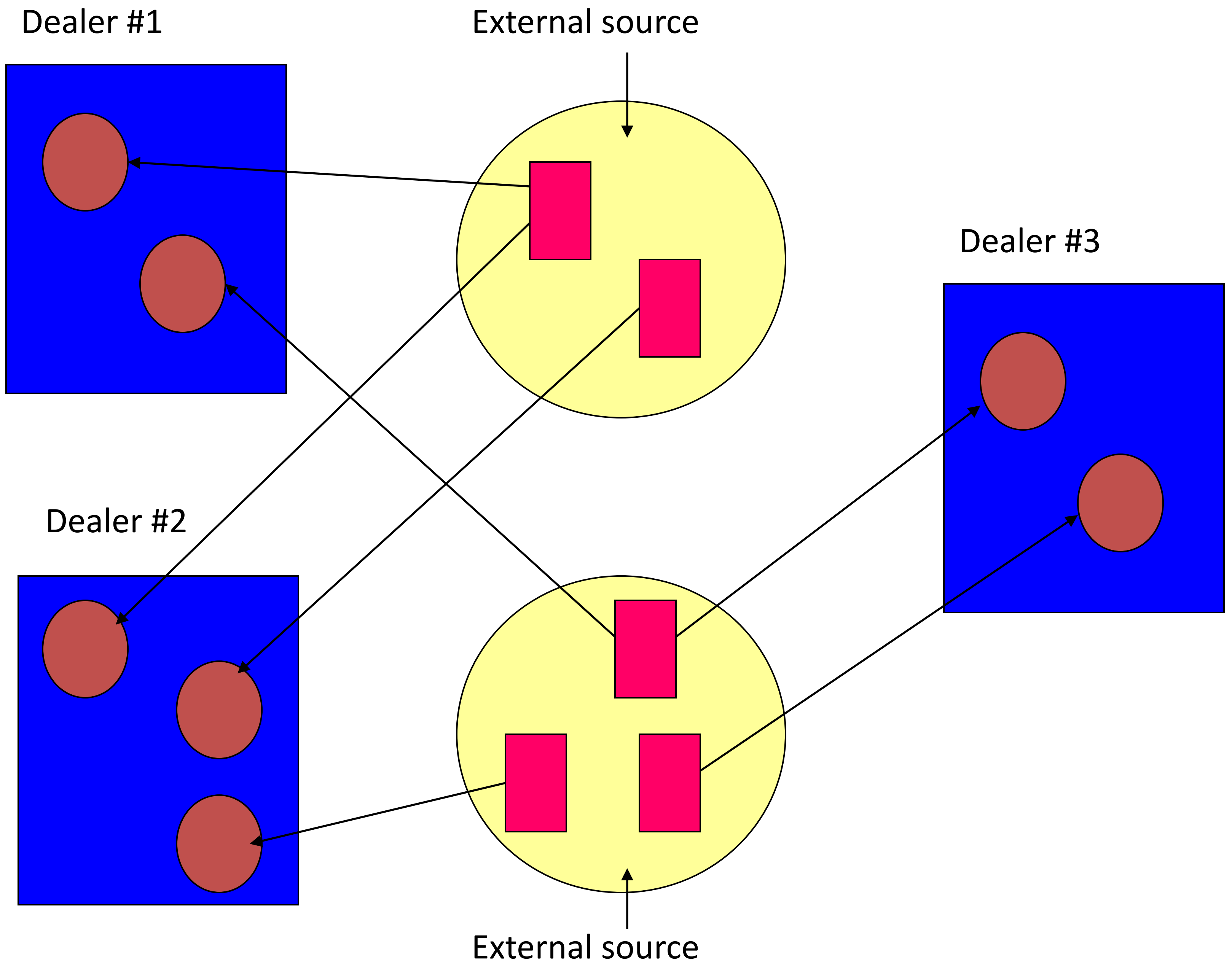
Characteristics of distributed event-based systems

- Heterogeneous: event-generating objects publish the types of events they offer; subscribers provide interfaces for receiving notifications.

- Asynchronous: notifications are sent asynchronously.

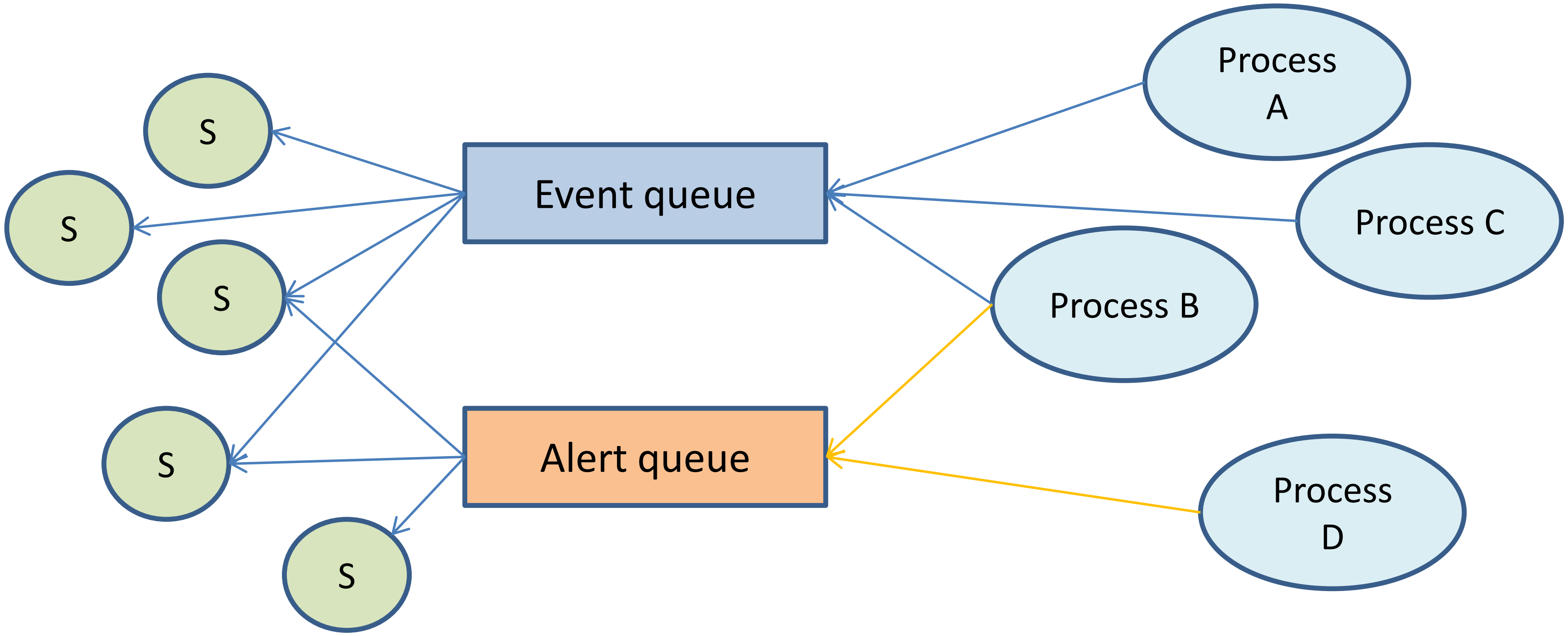
Example 1: the dealing room

- Dealers use computers to check the latest information about the market prices of the stocks they deal in.
- The market price for a single named stock is represented by an object with several variables.
- Updates of some or all variables arrive from several different external sources.
- Each update of a stock object is regarded as an event.
- The stock object experiencing such events notifies all of the dealers who subscribed to it.
- A dealer process creates an object to represent each named stock that the user asks to have displayed. This is the subscriber.


Dealer #1
External source
Dealer #3
Dealer #2
External source

Example 2: industrial setting

- Industrial processes send event and alert notifications to subscribers.



Types of events

- An event source can generate events of one or more different types.
- Each event has attributes: name, id of object that generated the event, the operation, its parameters and the time (or a sequence number).
- Types and attributes are used both in subscribing to events and in notifications.
- When subscribing to an event, the type of event is specified, sometimes modified with a criterion as to the values of the attributes.

Scalability of the system

- Proxy nodes can be used to create scalable notification trees.

- A third party, the notification manager, can be used to register publishers and subscribers – it can be a large database. For reliability purposes, the manager can be replicated.

- Subscribers can belong to different administrative domains.

- Subscribers belonging to the same admin domain can be jointly managed.

New add: brokers

- A broker decouples the publisher (that experiences changes of state resulting in events) from its subscribers.

- Roles for brokers:
  - forwarding: carries out all the work of sending notifications…
  - filtering of notifications: reduces the number of notifications received according to some predicate on the content of each notification;
  - patterns of events: subscribers can specify patterns of events they are interested in. A pattern defines a relationship among several events.
  - notification mailboxes: notifications need to be delayed until subscriber(s) is (are) ready to receive them.

How can an object pass information to another one ?

- An object can pass an object reference (subscriber) to the object that is the event source or broker.
- The event source will then invoke one of the subscriber's methods to notify it.
- A subscriber interface is defined for each major event type and interested objects then register implementations of that interface with the subscriber.

RMI Callbacks

- For a subscriber to register itself with the remote event source, it must invoke a remote method and pass an object reference to the remote subscriber interface it defines.

- Example: a server provides information about temperature changes, which can be accessed remotely.

The Subscriber Interface

- This interface defines a remote object with a single method.
- When an event occurs, the method is invoked and the new temperature is passed as a parameter.

```
interface TemperatureSubscriber extends java.rmi.Remote

{

        public void temperatureChanged(double temperature)

                throws java.rmi.RemoteException;

}
```

The Event Source Interface

- Allows a subscriber to register/deregister and provide additional methods.

```
interface TemperatureSensor extends java.rmi.Remote

{

        public double getTemperature() throws java.rmi.RemoteException;

        public void  addTemperatureSubscriber ( TemperatureSubscriber subscriber )

                throws java.rmi.RemoteException;

        public void  removeTemperatureSubscriber ( TemperatureSubsriber subscriber

                throws java.rmi.RemoteException;

}
```

The Implementation of the Event Source Interface

- A TemperatureSensorServer class is defined; it acts as an RMI server. This server will notify subscribers, as a client.
- The server must extend UnicastRemoteObject, to offer a service, and implement the TemperatureSensor interface.
- After creating an instance of the service and registering with the rmiregistry, the server will launch a thread, responsible for updating the value of the temperature (using a random number generator).
- When a change occurs, registered subscribers are notified, by reading from a list stored in a java.util.Vector object. This list is modified by the addTemperatureSubscriber and removeTemperatureSubscriber remote methods.

Mobile event notification systems

- Event notifications can originate from 3rd-party publishers.

- Apple Push Notification Service (APNS)
  - Stores the last notification until a new one arrives.
  - If the device has no connection or doesn’t connect for a long time, all stored notifications are discarded.
  - Accepted payload is small, 2 KB.

- Firebase Cloud Messaging (Google CM)
  - The app/subscriber needs a key in the manifest file in order to have access to the service.
  - GCM provides queuing and feedback on the delivery status of each notification. Payload is up to 4 KB. The message (notification or data) can be stored up to 4 weeks.

Summing up

- Dynamic distributed systems require means to inform interested parties about events that correspond to state changes.
- System’s entities: publishers, subscribers and brokers.
- Notifications: messages that correspond to events’ occurrences.
- Performance issues, including reliability, need to be considered.