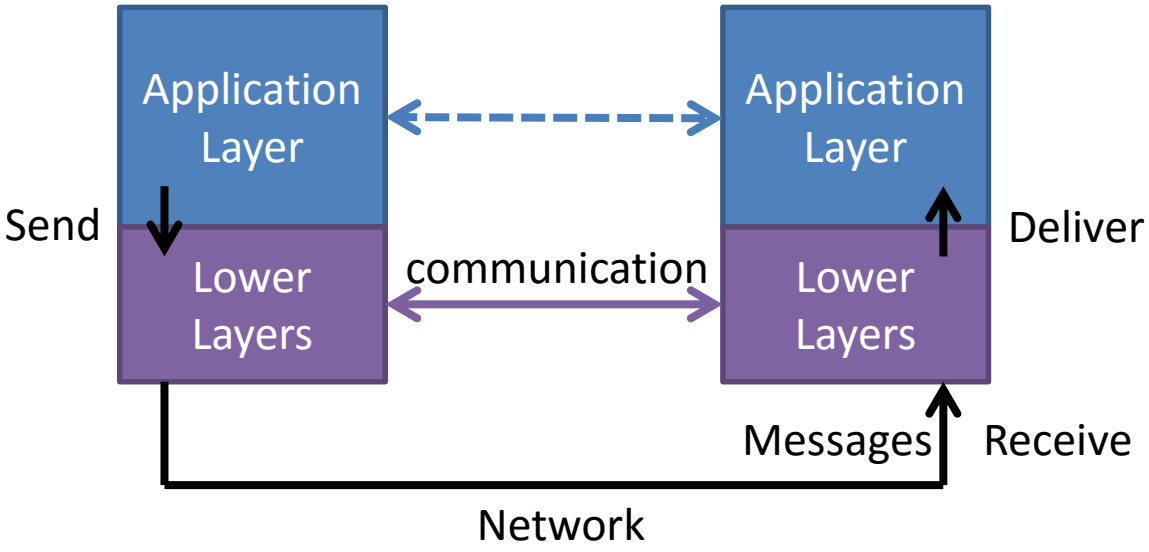
Appia

vanilladb.org

Terminology: Receive and Deliver



Terminology: Receive and Deliver

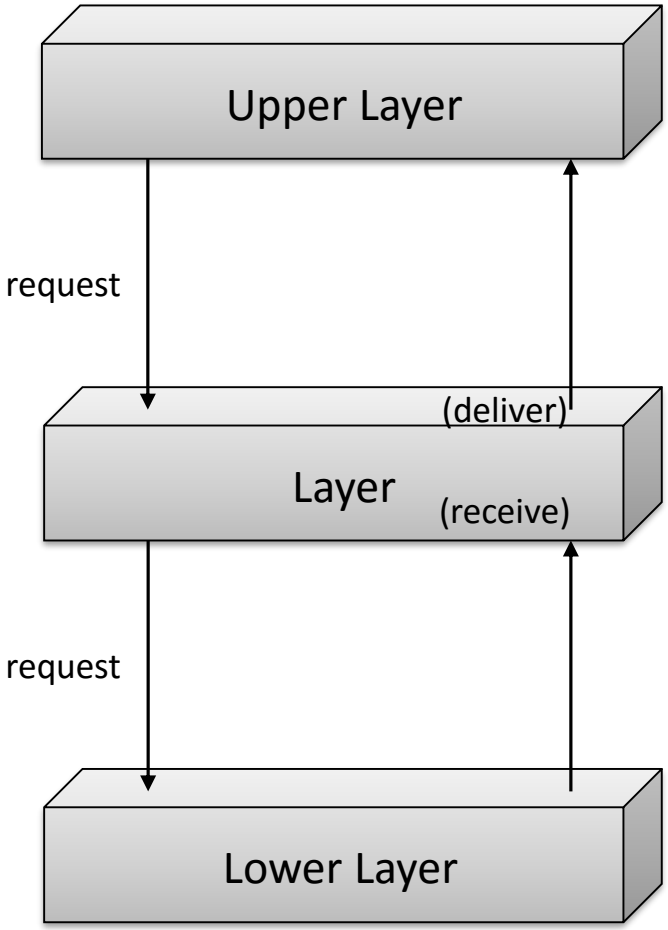
- There are many layers in network connections
- Messages are first ***received*** by lower layers, buffered, and some algorithms are executed to ensure some guarantees
- When the guarantees are satisfied, the message can be ***delivered*** to upper layer
- By ensuring easier guarantee in lower layers, higher layers can have more powerful guarantees

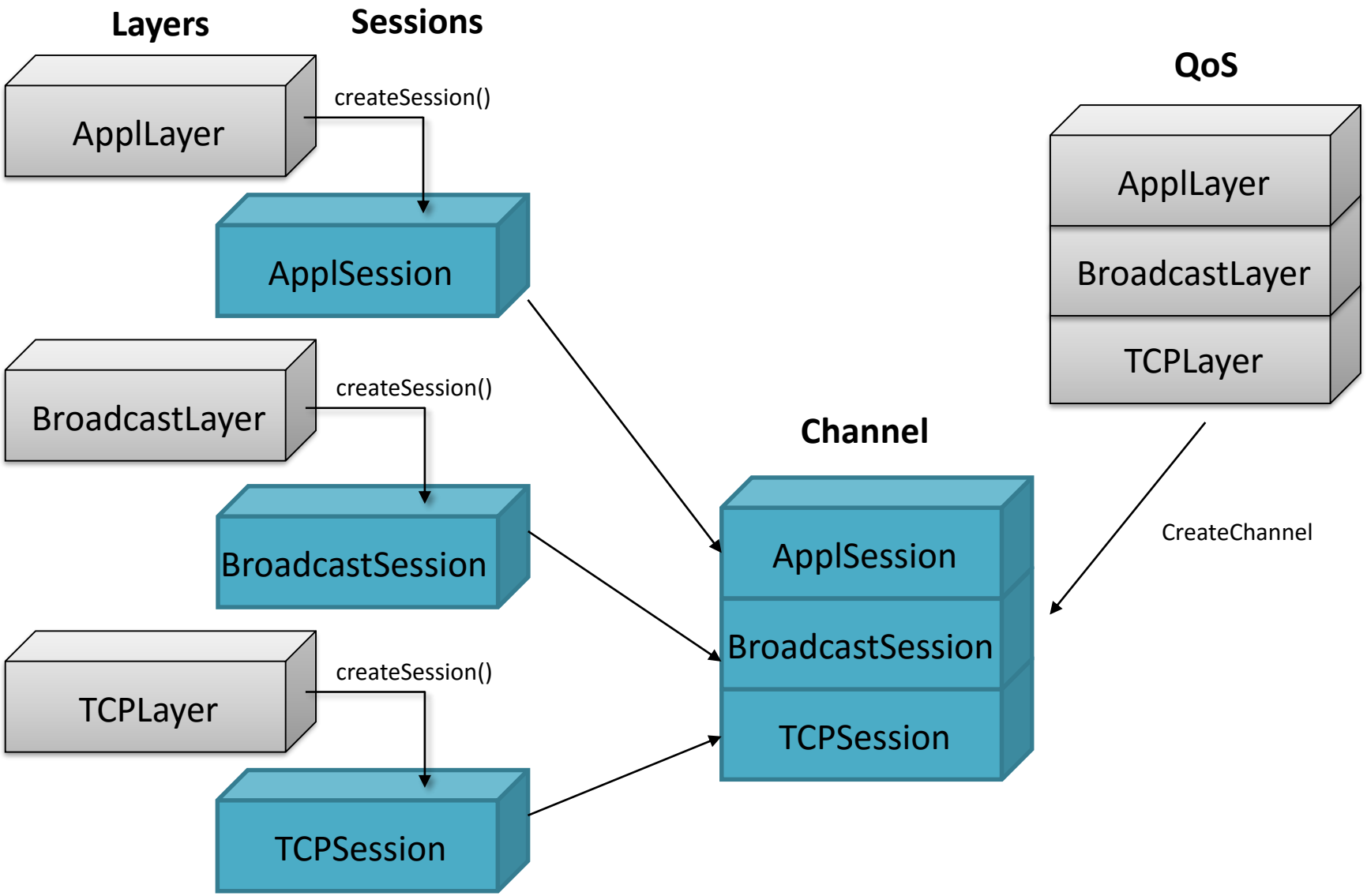
Appia

- Appia is a Java-based, open source toolkit that simplifies the implementation of layered communication protocols
  - Mostly at the Application layer
- We use Appia to implement various group communication protocols

Appia

- Programmer can compose a ***QoS*** (a protocol stack) with several layers
  - The implementation of a layer is a ***session***
  - An instance of a QoS is a ***channel***


Upper Layer
request
(deliver)
Layer
(receive)
request
Lower Layer


Layers
Sessions
ApplLayer
createSession()
ApplSession
BroadcastLayer
createSession()
BroadcastSession
TCPLayer
createSession()
TCPSession
QoS
ApplLayer
BroadcastLayer
TCPLayer
Channel
ApplSession
BroadcastSession
TCPSession
CreateChannel

```
private static Channel getBebChannel(ProcessSet processes) {
    /* Create layers and put them on a array */
    Layer[] qos = { new TcpCompleteLayer(), new BasicBroadcastLayer(),
          new SampleApplLayer() };

    /* Create a QoS */
    QoS myQoS = null;
    try {
       myQoS = new QoS("Best Effort Broadcast QoS", qos);
    } catch (AppiaInvalidQoSException ex) {
       System.err.println("Invalid QoS");
       System.err.println(ex.getMessage());
       System.exit(1);
    }
    /* Create a channel. Uses default event scheduler. */
    Channel channel = myQoS
          .createUnboundChannel("Best effort Broadcast Channel");
    /*
     * Application Session requires special arguments: filename and . A
     * session is created and binded to the stack. Remaining ones are
     * created by default
     */
    SampleApplSession sas = (SampleApplSession) qos[qos.length - 1]
          .createSession();
    sas.init(processes);
    ChannelCursor cc = channel.getCursor();
    /*
     * Application is the last session of the array. Positioning in it is
     * simple
     */
    try {
       cc.top();
       cc.setSession(sas);
    } catch (AppiaCursorException ex) {
       System.err.println("Unexpected exception in main. Type code:"
             + ex.type);
       System.exit(1);
    }
    return channel;
}
```

Build up a channel

Layers & Events

- A Layer needs several types of ***events*** for different usages
  - Provide
    - Events that the protocol creates
  - Require
    - Events that the protocol requires to work
    - Usually the events from the lower layers that is used by this layer
  - Accept
    - Events that the protocol accepts (from upper or lower layer)

```
public BasicBroadcastLayer() {
    /* events that the protocol will create */
    evProvide = new Class[0];

    /*
     * events that the protocol require to work.
This is a subset of the
     * accepted events
     */
    evRequire = new Class[3];
    evRequire[0] = SendableEvent.class;
    evRequire[1] = ChannelInit.class;
    evRequire[2] = ProcessInitEvent.class;

    /* events that the protocol will accept */
    evAccept = new Class[4];
    evAccept[0] = SendableEvent.class;
    evAccept[1] = ChannelInit.class;
    evAccept[2] = ChannelClose.class;
    evAccept[3] = ProcessInitEvent.class;

  }
```

Handling Events

- When an event is processed by a session, Appia core will call the “handle(Event event)” method to process the event
- The session calls the methods to handle different events

```
public void handle(Event event) {
    if (event instanceof ChannelInit)
        handleChannelInit((ChannelInit) event);
    else if (event instanceof ProcessInitEvent)
        handleProcessInitEvent((ProcessInitEvent) event);
    else if (event instanceof SendableEvent) {
        if (event.getDir() == Direction.DOWN)
            bebBroadcast((SendableEvent) event);
        else
            pp2pDeliver((SendableEvent) event);
    }
}
```

Receives an event to be broadcasted from the upper layer: calls the “bebBroadcast” method

Receives an event from lower perfect link layer: calls the “pp2pDeliver” method

Handling Events

- The bebBroadcast method is the method that actually broadcast the event

```
private void bebBroadcast(SendableEvent event) {

    SampleProcess[] processArray = this.processes.getAllProcesses();
    SendableEvent sendingEvent = null;

    for (int i = 0; i < processArray.length; i++) {
      try {
        if (i == (processArray.length - 1))
          sendingEvent = event;
        else
          sendingEvent = (SendableEvent) event.cloneEvent();
        sendingEvent.source = processes.getSelfProcess()
            .getSocketAddress();
        sendingEvent.dest = processArray[i].getSocketAddress();
        sendingEvent.setSourceSession(this);

        if (i == processes.getSelfRank())
          sendingEvent.setDir(Direction.UP);

        sendingEvent.init();
        sendingEvent.go();
      } catch (CloneNotSupportedException e) {
        e.printStackTrace();
        return;
      } catch (AppiaEventException e) {
        e.printStackTrace();
        return;
      }
    }
  }
```

Sending Events

- To send an event to another process, just setup the parameters of the event, and call the “go()” method. Appia core will handle the rest of the work

```
try {
        SendableEvent sendingEvent = new SendableEvent();
        // set source and destination of event message
        sendingEvent.source = processes.getSelfProcess()
              .getSocketAddress();
        sendingEvent.dest = processArray[1].getSocketAddress();
        // sets the session that created the event.
        sendingEvent.setSourceSession(this);
        // if it is the "self" process, send the event upwards
        if (i == processes.getSelfRank())
           sendingEvent.setDir(Direction.UP);
        // initializes and sends the message event
        sendingEvent.pushObject();
        sendingEvent.init();
        sendingEvent.go();
} catch (AppiaEventException e) {
        e.printStackTrace();
        return;
}
```