ID2203 - Tutorial 1

Distributed Systems, Advanced Course

Cosmin Arad icarad@kth.se
Tallat Shafaat tallat@kth.se
Seif Haridi haridi@kth.se

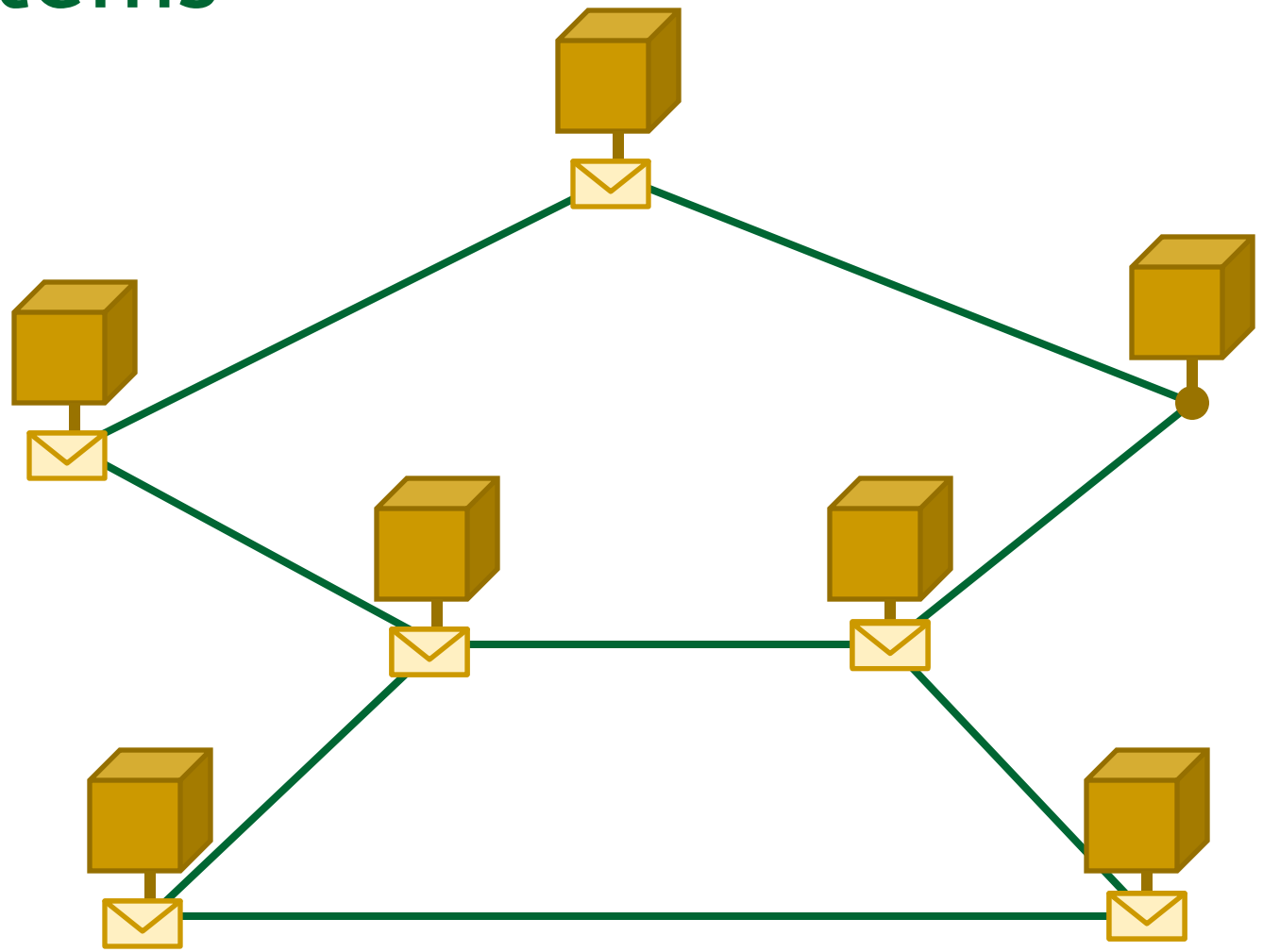
KTH - The Royal Institute of Technology

Overview

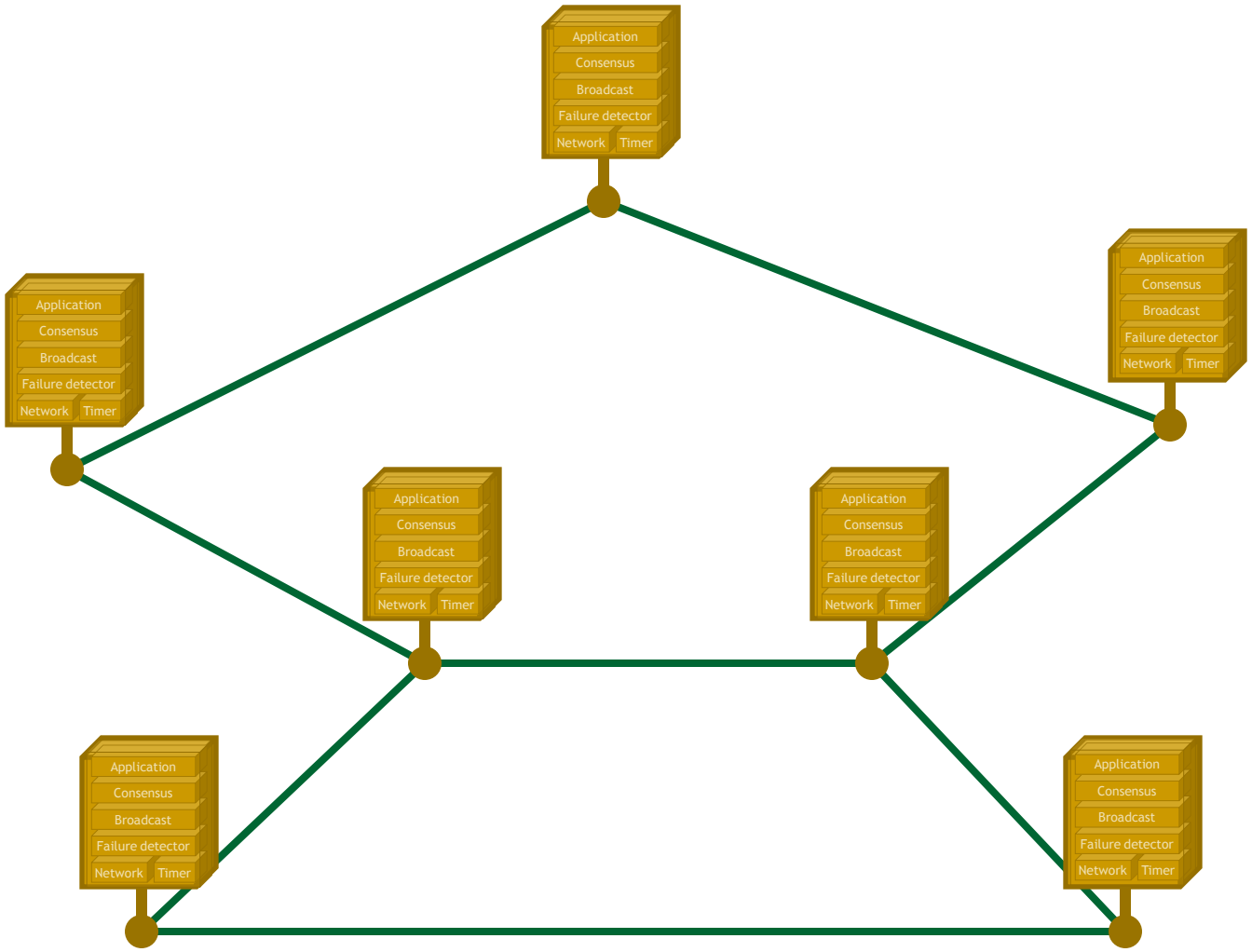
- Introduction to Kompics
- Relation to the textbook
- Assignments framework
- First assignment

Component Model for Distributed Systems

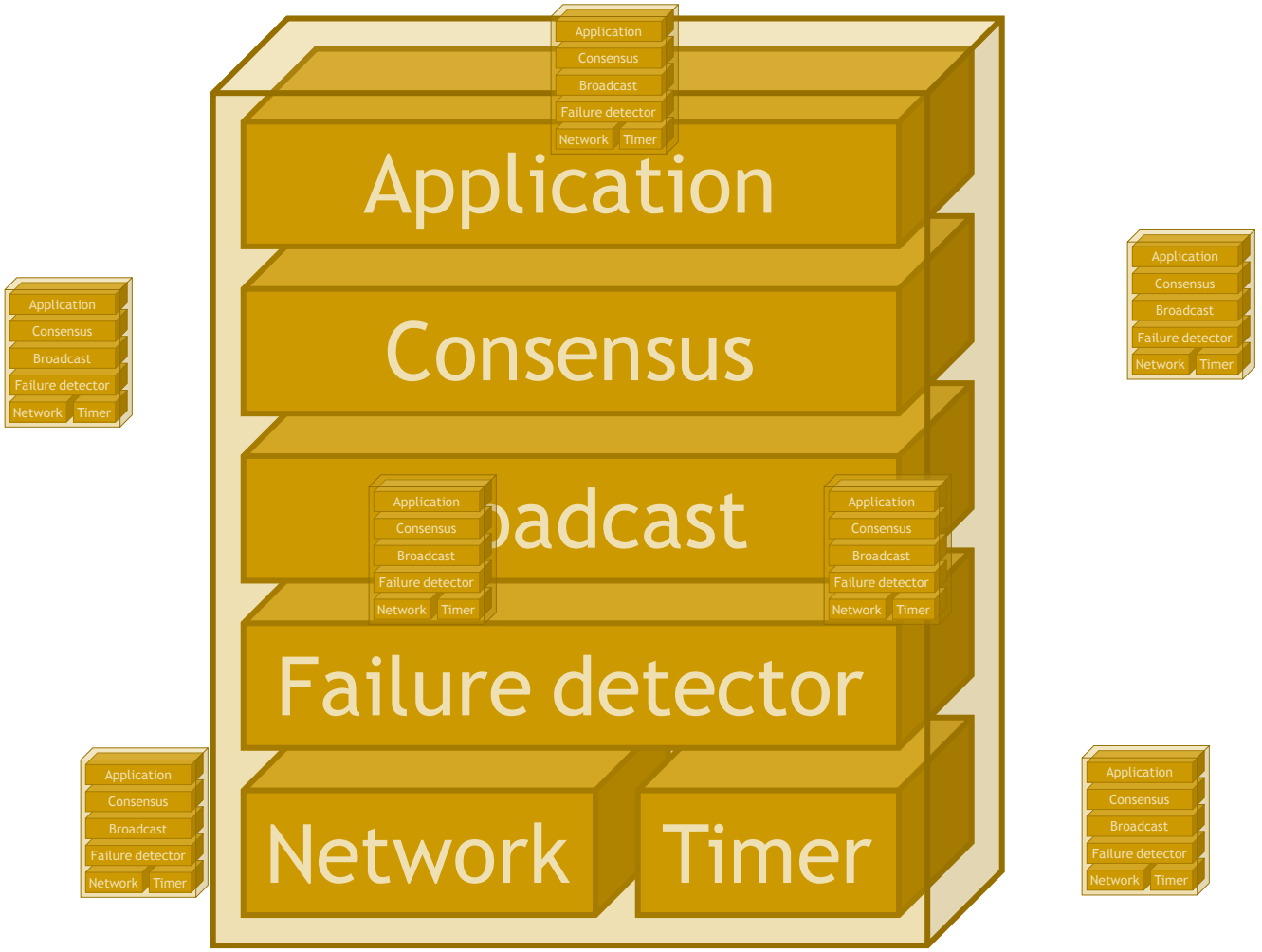
We want to build distributed systems



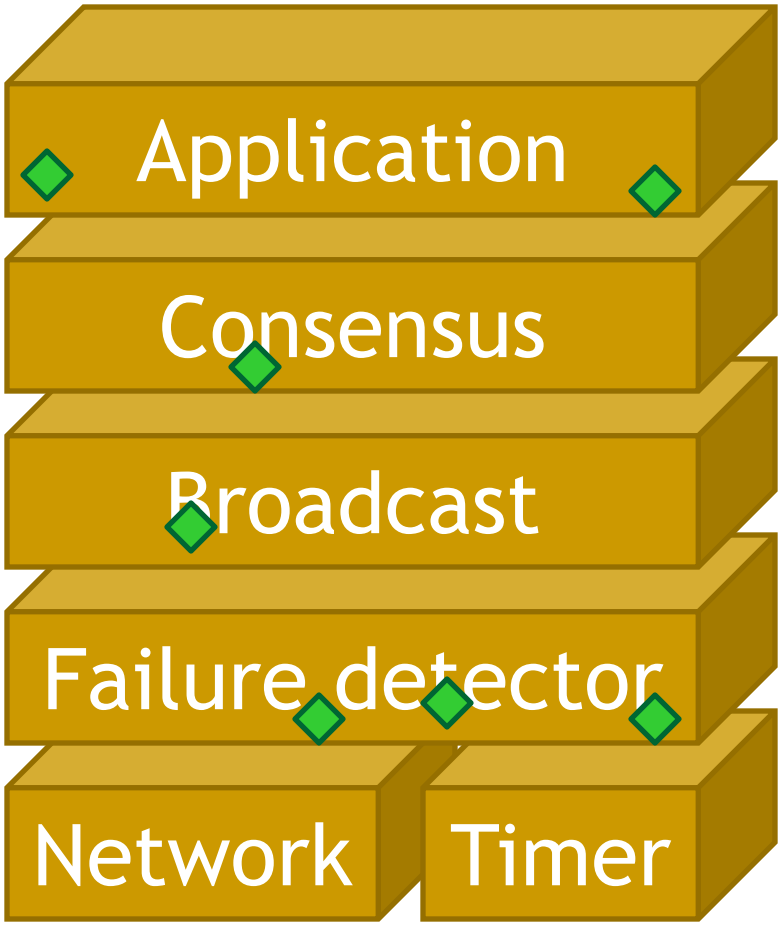
by composing distributed protocols



Implemented as reactive components

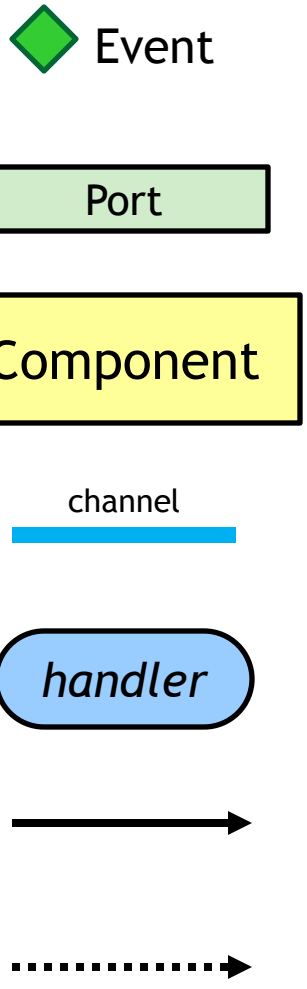


with message-passing concurrency



Concepts in Kompics

- Event
- Port
- Component
- Channel
- Handler
- Subscription
- Publication / Trigger



Events

- Events are passive immutable objects
  - with typed attributes / fields
- Events are typed and can be sub-typed

```
class Message extends Event {
   Address source;
   Address destination;
}
class DataMessage extends Message {
   Data data;
   int sequenceNumber;
}
```

DataMessage $\subseteq$ Message

Ports

Port

Direction

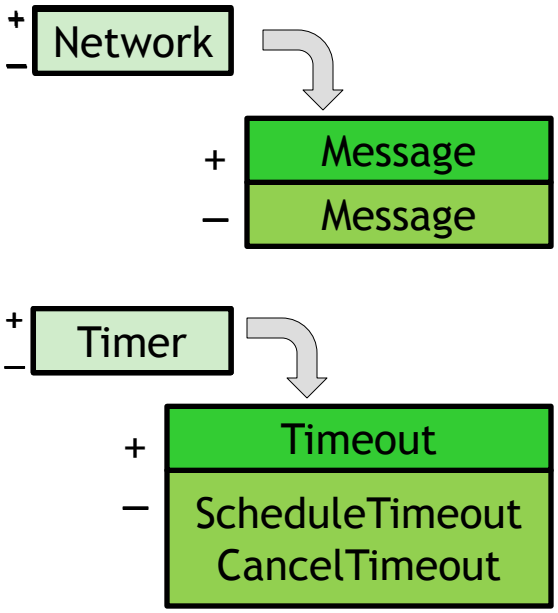
Direction

Ports

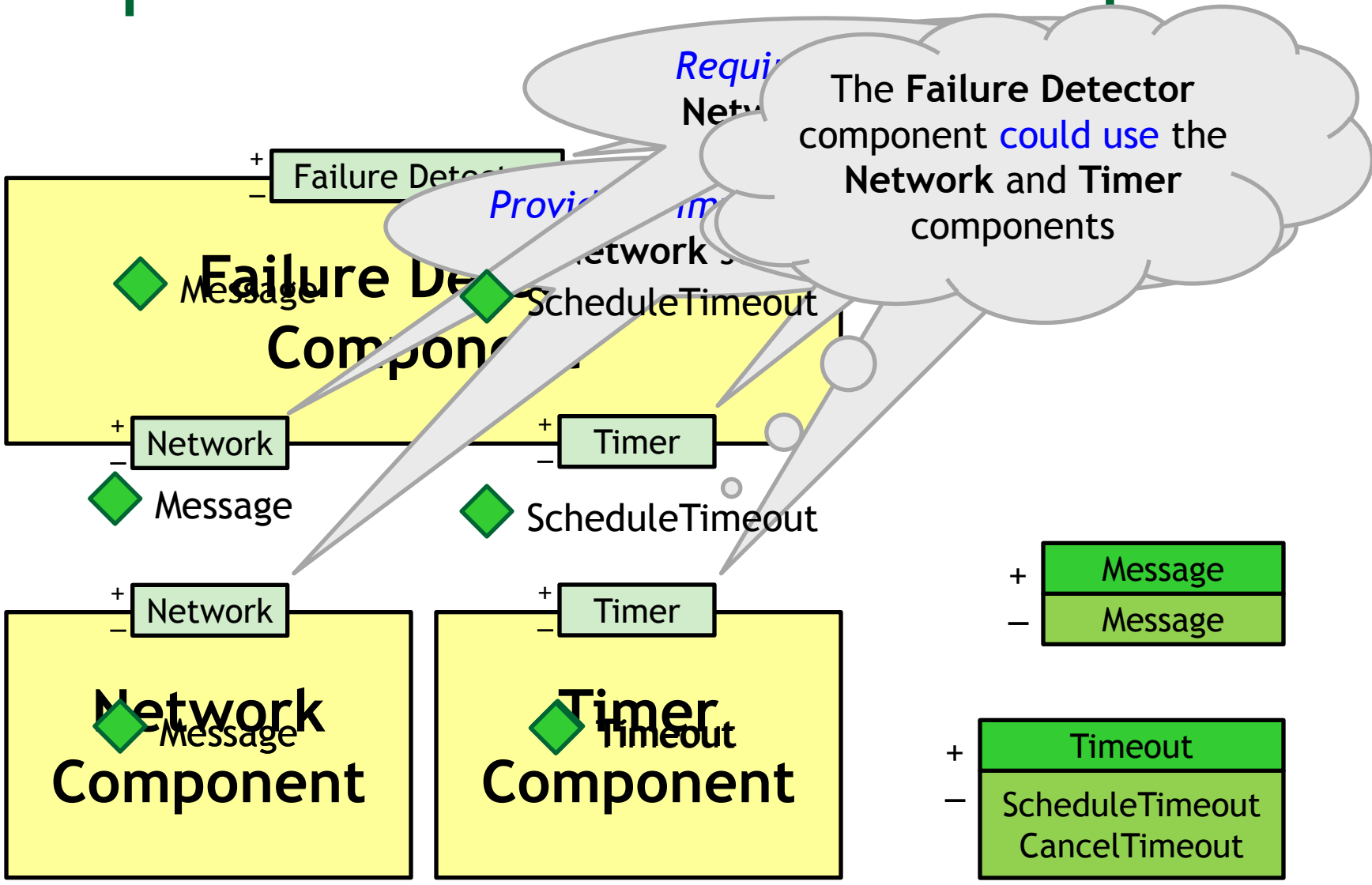
Port

- are bidirectional event-based comp interfaces
  - have a **positive (+)** and a **negative (−)** direction
- A *port type* consists of 2 sets of event types
  - one set of event types for each direction, **+** and **−**
  - represents a service/protocol abstraction

```
class Network extends PortType {{
        positive(Message.class);
        negative(Message.class);
}}
class Timer extends PortType {{
        positive(Timeout.class);
        negative(ScheduleTimeout.class);
        negative(CancelTimeout.class);
}}
```

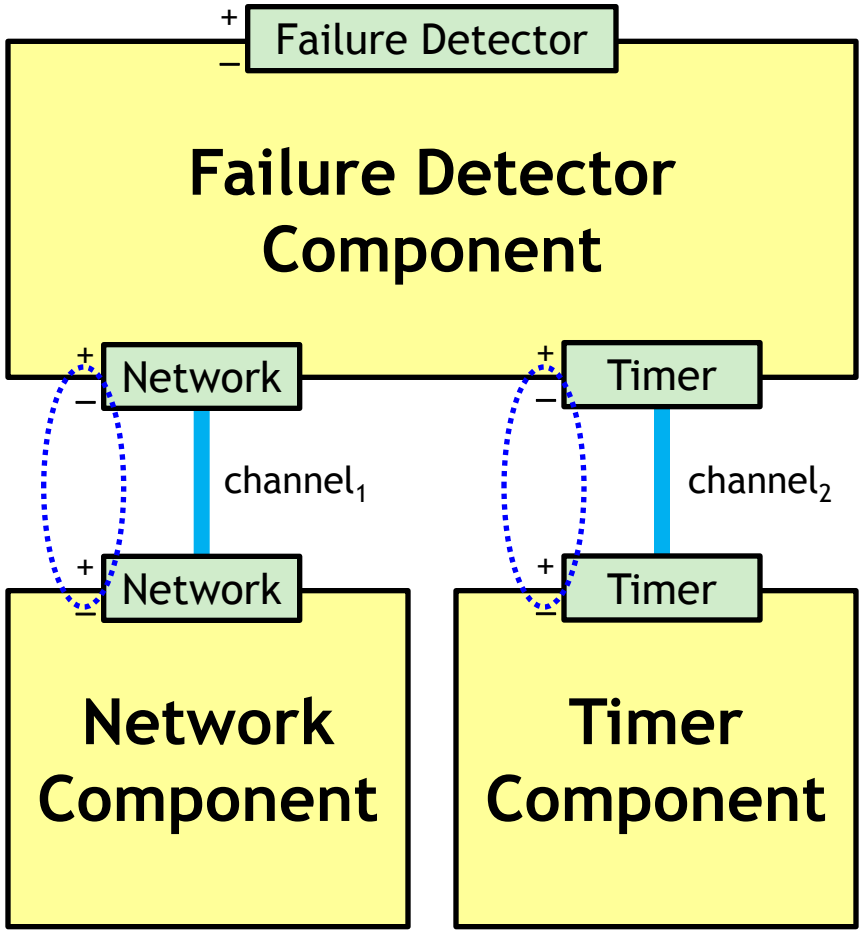


Components with Ports Example



Channels

- Channels connect *complementary* ports of the *same* type
  - + to −
  - − to +
- Channels forward events in *FIFO* order in *both* directions



Event handlers

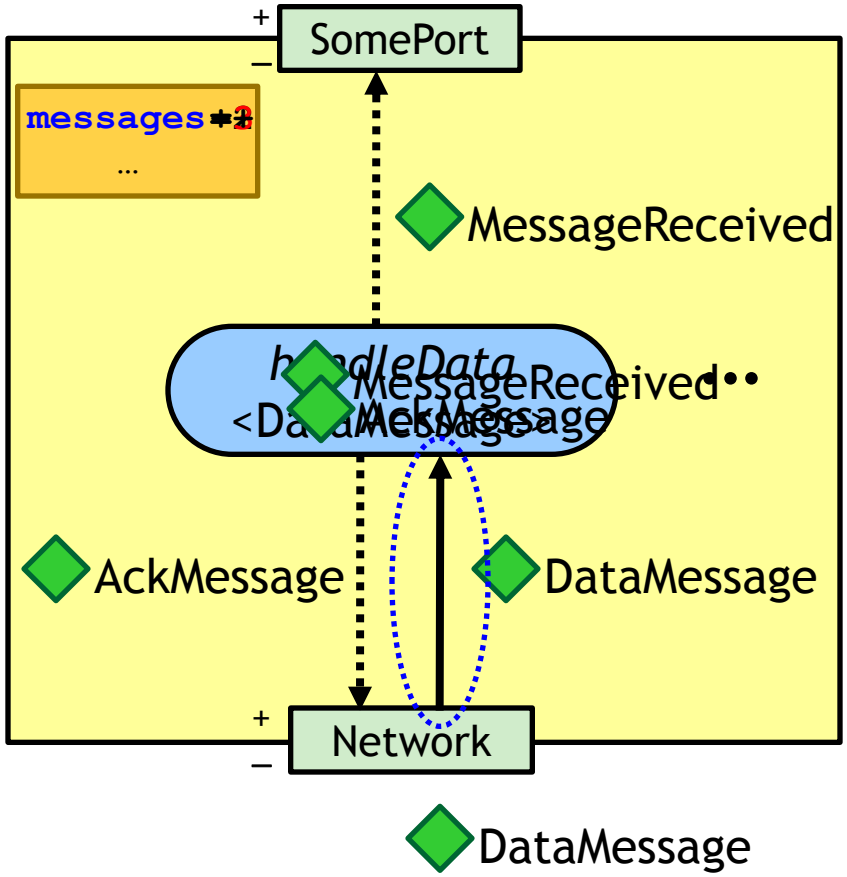
handler

- A handler is a first-class component procedure
- accepting a particular type of events
- executed reactively upon receiving an event
  - may mutate the local state and trigger new events
  - handlers of one component are mutually exclusive

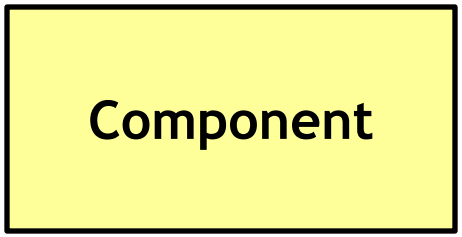
```
Handler<DataMessage> handleData = new Handler<DataMessage>() {
    public void handle(DataMessage dm) {
        messages++;                          // component state update
        trigger(new MessageReceived(dm.data), somePort);
        trigger(new AckMessage(myAddress, dm.source,
                dm.sequenceNumber), network); // event triggering
  }
};
```

Subscriptions & Publications

- A subscription binds an event handler, *h*, to a local component port *p*
- Let *e* be the type of *h*
  - let *f* be a supertype of *e*
  - *f* must come in on *p*
- After subscription
  - *h* will handle all events of any type *d*, subtype of *e*, coming in on *p*

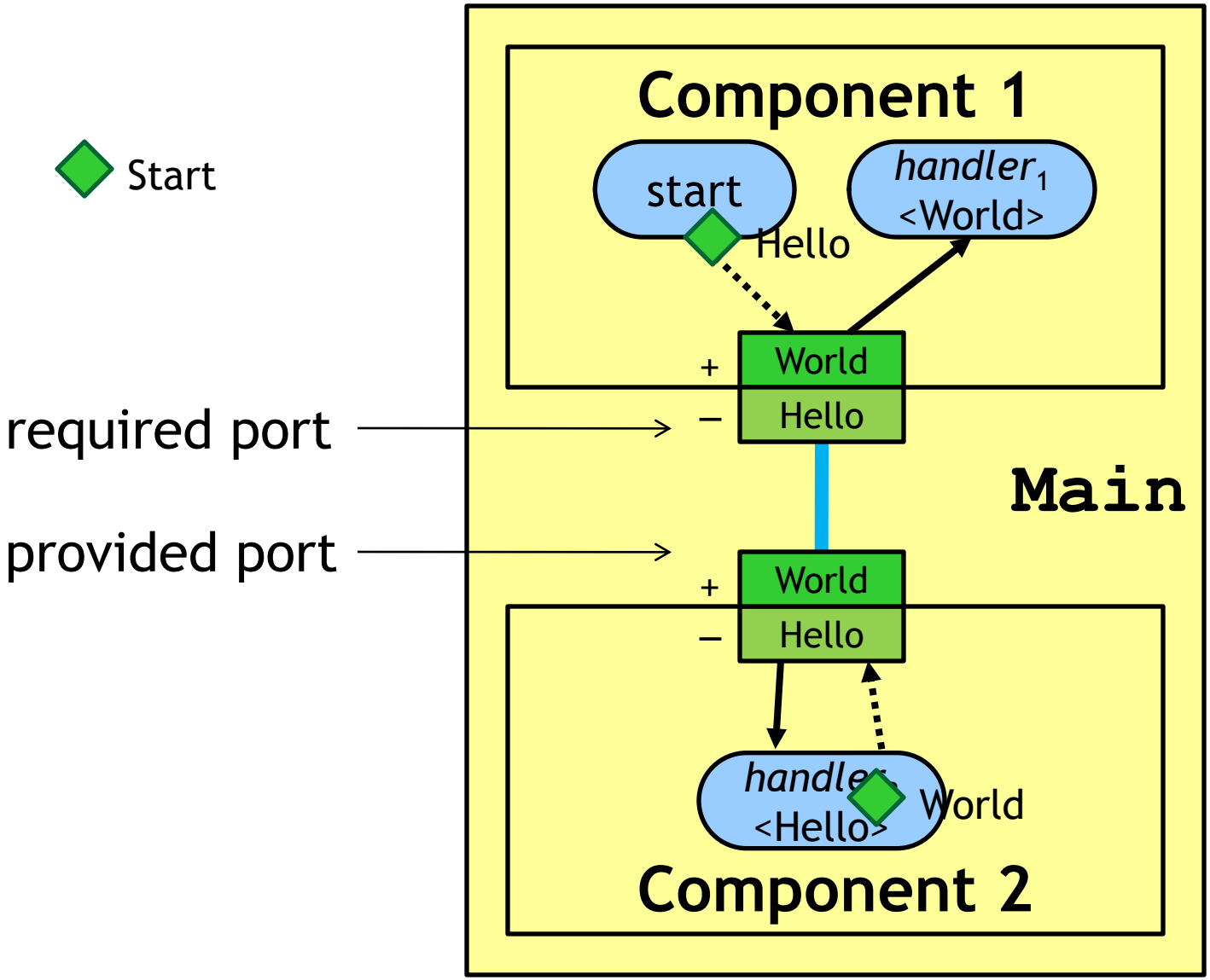


Components



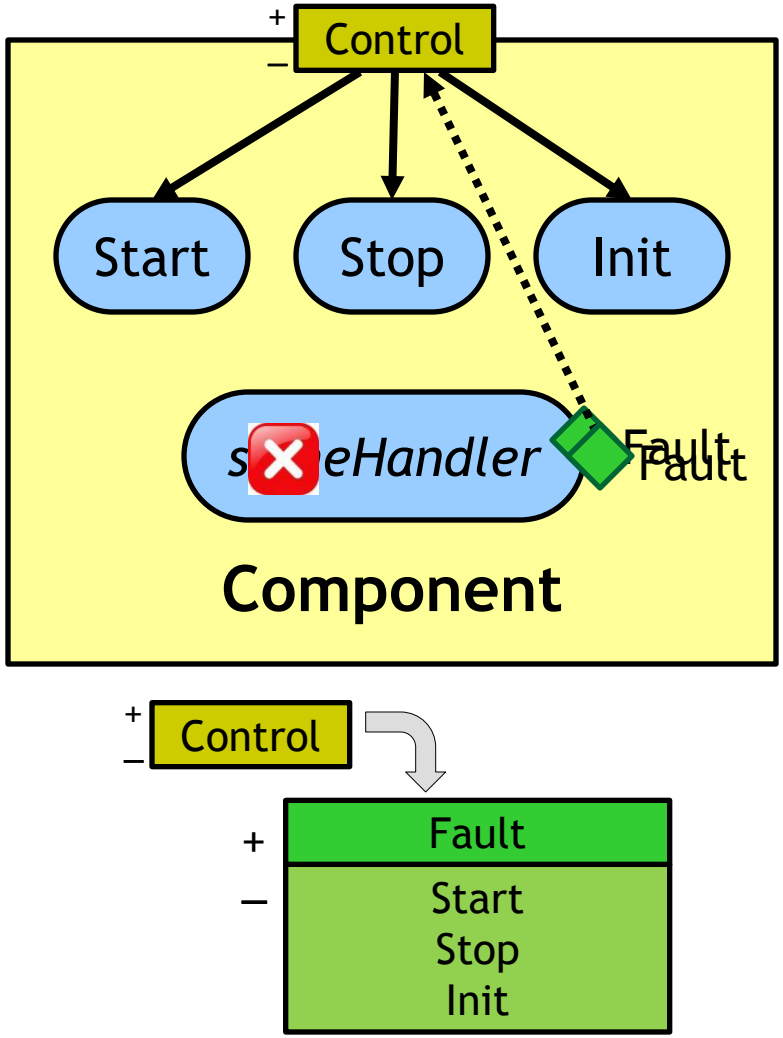
- are instantiated from component definitions
  - component definitions are Java classes
- A component instance is an object containing
  - local state variables
  - ports (provided or required interfaces)
  - event handlers
  - subscriptions
  - encapsulated subcomponents
  - channels
- form a containment hierarchy rooted at **`Main`**

Hello World! Example

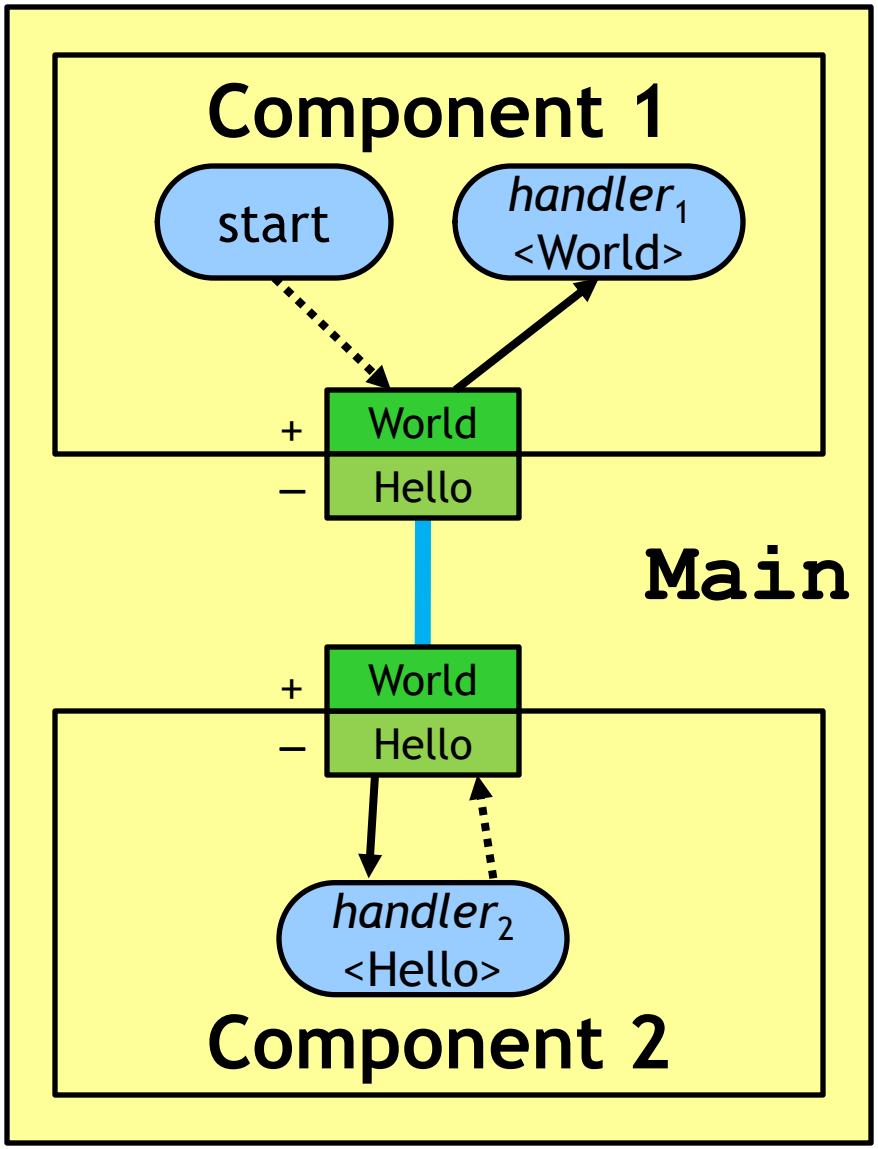


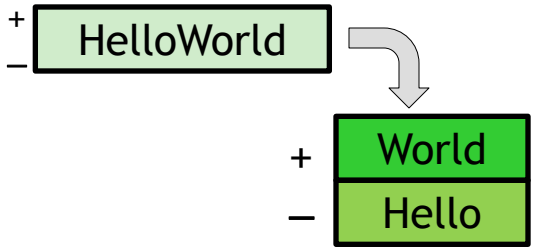
Control port

- Every component has a Control port
  - by default (provided)
  - not shown in diagrams
  - allows component to handle lifecycle events
- Exceptions / faults not caught inside a handler
  - wrapped into Fault event
  - triggered on control port



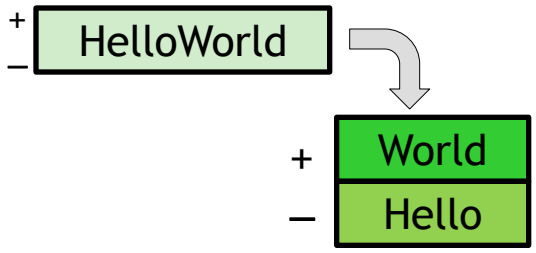
Hello World! Example





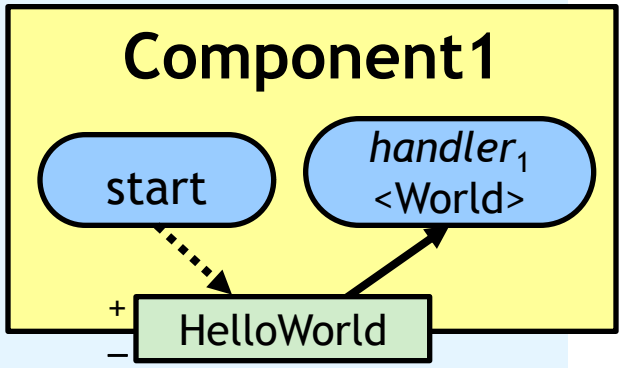
Source code: Events and Port

```
public final class Hello extends Event {
   private final String message;
   public Hello(String m) {
        message = m;
   }
   public String getMessage() {
        return message;
   }
}

public final class World extends Event {
}

public class HelloWorld extends PortType{{
   positive(World.class);
   negative(Hello.class);
}}
```



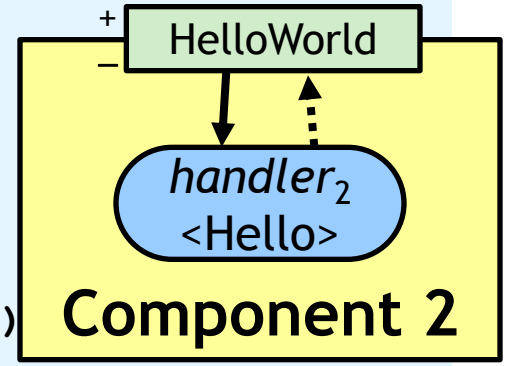
Source code: Component1

```
public class Component1 extends ComponentDefinition {

  private Positive<HelloWorld> hwPort = positive(HelloWorld.class);

  public Component1() {
    System.out.println("Component1 created.");
    subscribe(startHandler, control);
    subscribe(worldHandler, hwPort);
  }

  Handler<Start> startHandler = new Handler<Start>() {
    public void handle(Start event) {
      System.out.println("Component1 started. Triggering Hello...");
      trigger(new Hello("Hi there!"), hwPort);
  }};
  Handler<World> worldHandler = new Handler<World>() {
    public void handle(World event) {
      System.out.println("Component1 received World event.");
  }};
}
```

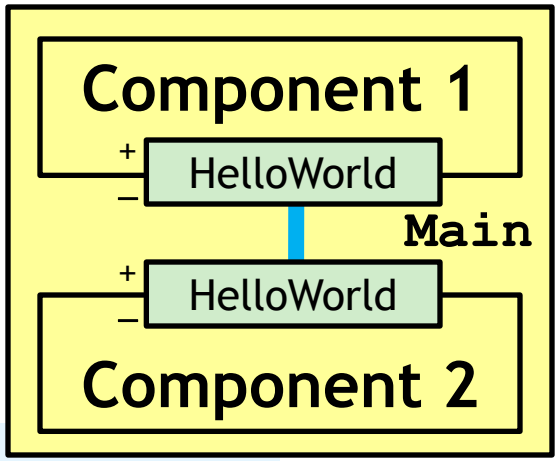


Source code: Component2

```
public class Component2 extends ComponentDefinition {
  private Negative<HelloWorld> hwPort = negative(HelloWorld.class);
  public Component2() {
    System.out.println("Component2 created.");
    subscribe(startHandler, control);
    subscribe(helloHandler, hwPort);
  }
  Handler<Start> startHandler = new Handler<Start>() {
    public void handle(Start event) {
      System.out.println("Component2 started.");
  }};
  Handler<Hello> helloHandler = new Handler<Hello>() {
    public void handle(Hello event) {
      System.out.println("Component2 received Hello event with "
                         + "message: " + event.getMessage());
      trigger(new World(), hwPort);
  }};
}
```



Source code: **Main**



- **Main** is a Java main class

```
public class Main extends ComponentDefinition {
   private Component component1, component2;
   public Main() {
        System.out.println("Main created.");
        component1 = create(Component1.class);
        component2 = create(Component2.class);
        connect(component1.getNegative(HelloWorld.class),
                component2.getPositive(HelloWorld.class));
   }
   public static void main(String[] args) {
        Kompics.createAndStart(Main.class);
        Kompics.shutdown();
   }
}
```

Output

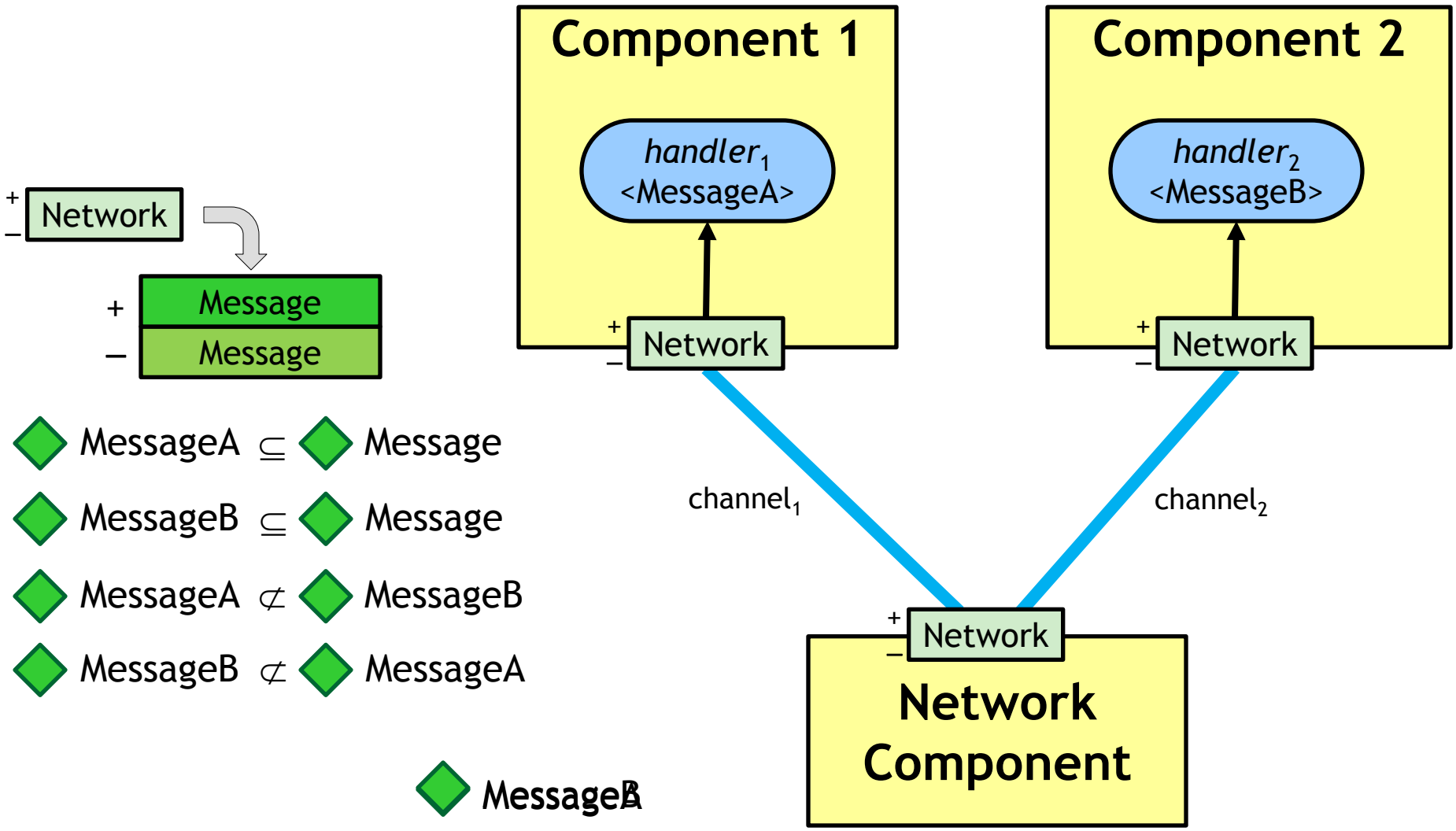
```
prompt:$ java Main
Main created.
Component1 created.
Component2 created.
Component1 started. Triggering Hello...
Component2 started.
Component2 received Hello event with message: Hi there!
Component1 received World event.
prompt:$ _
```

- Kompics runtime creates and starts **`Main`**
  - **`Main`** recursively creates and starts **`c1`**, **`c2`**
  - **`c1`**'s start handler triggers Hello event
    - handled by **`c2`**, which triggers World event
      - handled by **`c1`**

Scheduling / Canceling a Timeout


```
class MyComponent extends ComponentDefinition {
   Positive<Timer> timer = positive(Timer.class); //required
     {
       // scheduling a timeout
       long delay = 5000;
       ScheduleTimeout st = new ScheduleTimeout(delay);
       st.setTimeoutEvent(new MyTimeout(st));
       UUID timeoutId = st.getTimeoutEvent().getTimeoutId();
       trigger(st, timer);
       // canceling a timeout
       CancelTimeout ct = new CancelTimeout(timeoutId);
       trigger(ct, timer);
     }
}
```

Publish / Subscribe



Software engineering view

- Events and ports are interfaces
  - service abstractions, modules
  - packaged together as libraries
- Components are implementations
  - provide or require modules / interfaces
  - dependencies on provided / required modules
    - expressed as library dependencies
    - multiple implementations for some module
      - separate libraries
    - deploy-time composition

Relation to the textbook

How we use Kompics to model the abstractions in the textbook / course

- Modules – ports, with examples
- Algorithms – components
- Events are the same
- Introduce Pp2p, Flp2p, Timer.
  - **To be updated shortly…**

Assignments framework

How we run distributed systems and create experiment scenarios

- Topology – description, code
- Scenario – description, code
- Assignment0 - description
- DelayLink enforces delays
- DelayDropLink enforces delays and drops
- Application enforces a script of requests
- Together used to design execution scenarios
- Launcher description; window, kill, input/all
  - **To be updated shortly…**

To terminate an experiment type

Ctrl+K

First assignment

Failure detectors

- Architectures
- Ports / events
- Algorithms
  - **To be updated shortly…**

Good luck!