

PIMP Pervasive Interaction Mobile Platform



IPerG

Integrated Project on Pervasive Gaming

- ➔ **Karl-Petter Åkesson – kalle@sics.se**
- ➔ **Olov Ståhl – olovs@sics.se**



What is PIMP?

- ➔ **PIMP is a Java library that can be used to build distributed applications**
- ➔ **Focus is on distributed ubicomp applications, supporting various devices, typically also including sensor and actuator hardware**
- ➔ **The conceptual model behind focuses on making the whole system understandable by an end-user**
 - Encapsulating functionality
 - Obvious dependencies
 - Rapid and simple re-configurability



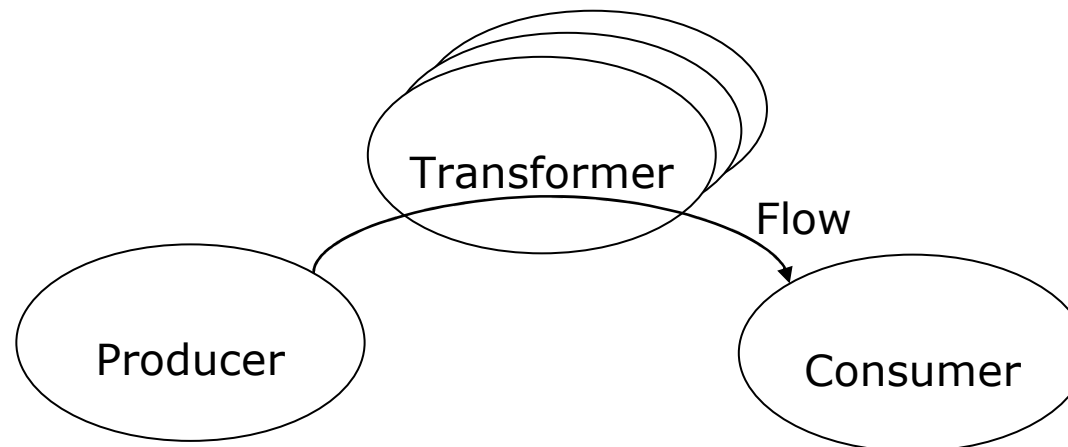
Background

- ⇒ **Pervasive/Ubiquitous Computing artifacts in an living environment**
 - Need adaptation to users
 - Impossible to design for every possible place
 - Things change over time
- ⇒ **Empower the user!**
 - Make the system understandable
 - DIY-attitude
- ⇒ **Example areas - has its origin in**
 - Intelligent houses
 - Pervasive games
- ⇒ **The one modifying is not necessarily the end-user**
 - Pervasive game organizer vs. player



PIMP conceptual model

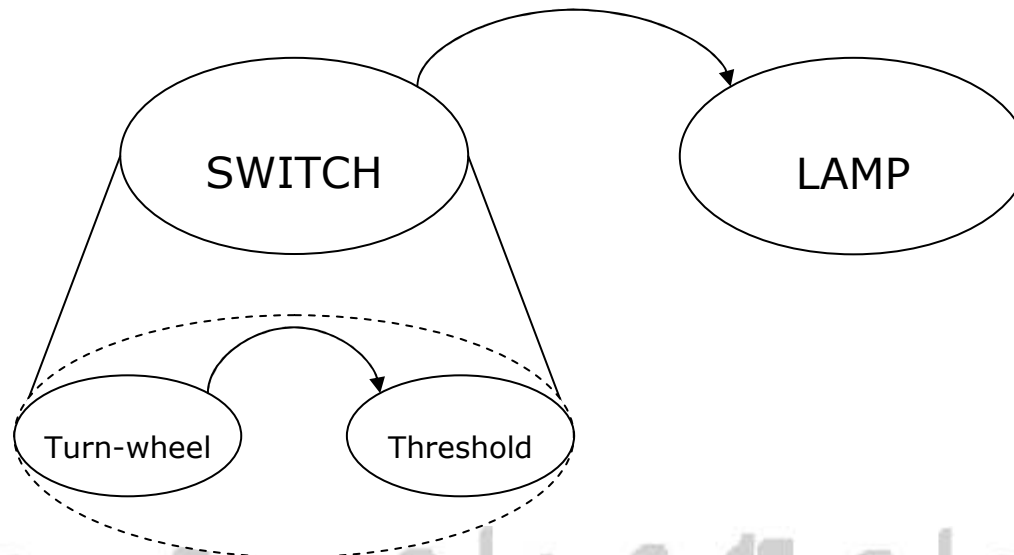
- **Data producers**
- **Data consumers**
- **Data flows**
- **Data transformers**



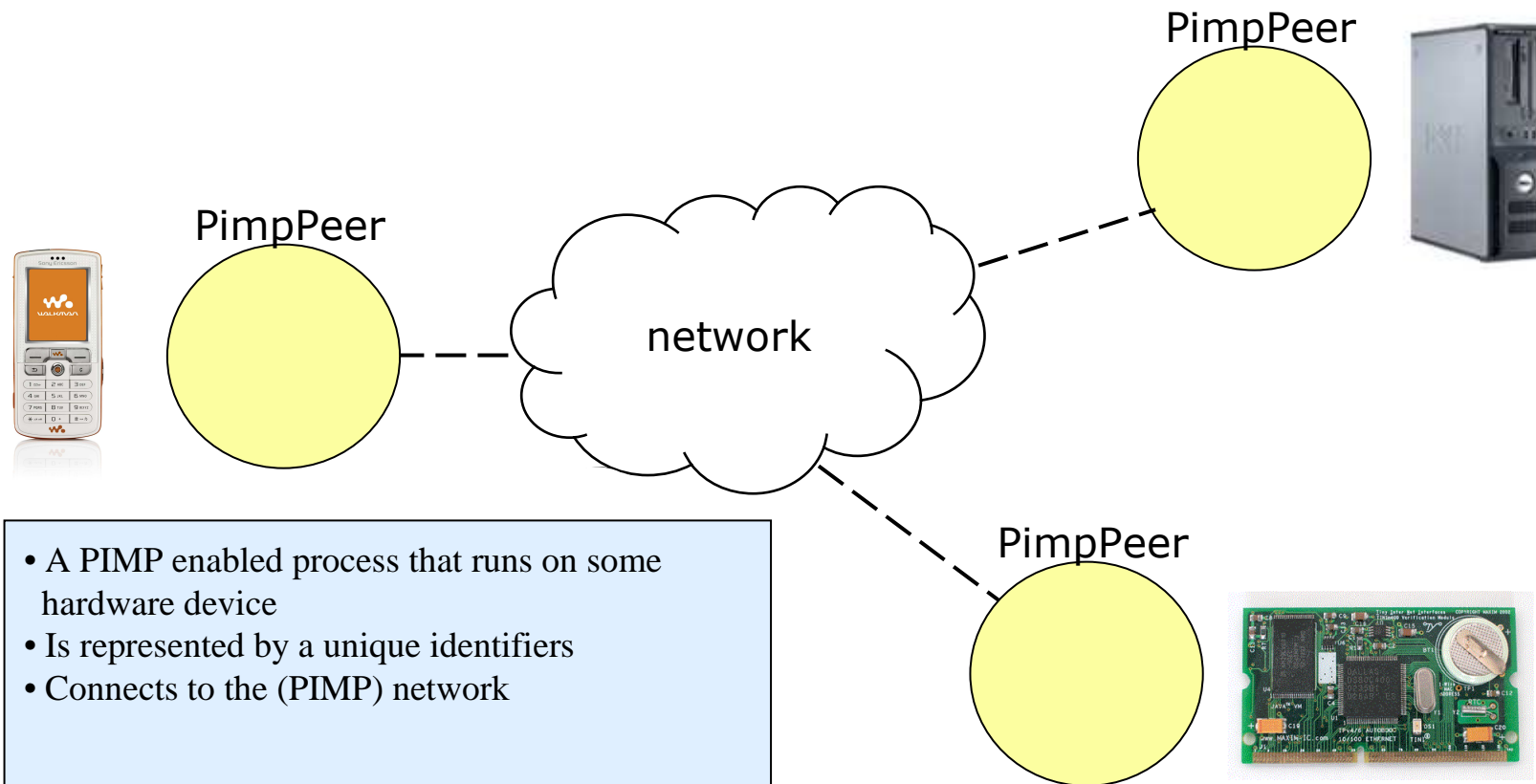
PIMP my world

➔ PIMP's model to empower the user

- o Connect functionality to allow change somewhere generate results elsewhere
- o Functionality is connected by creating links
- o Hierarchical model to allow encapsulation



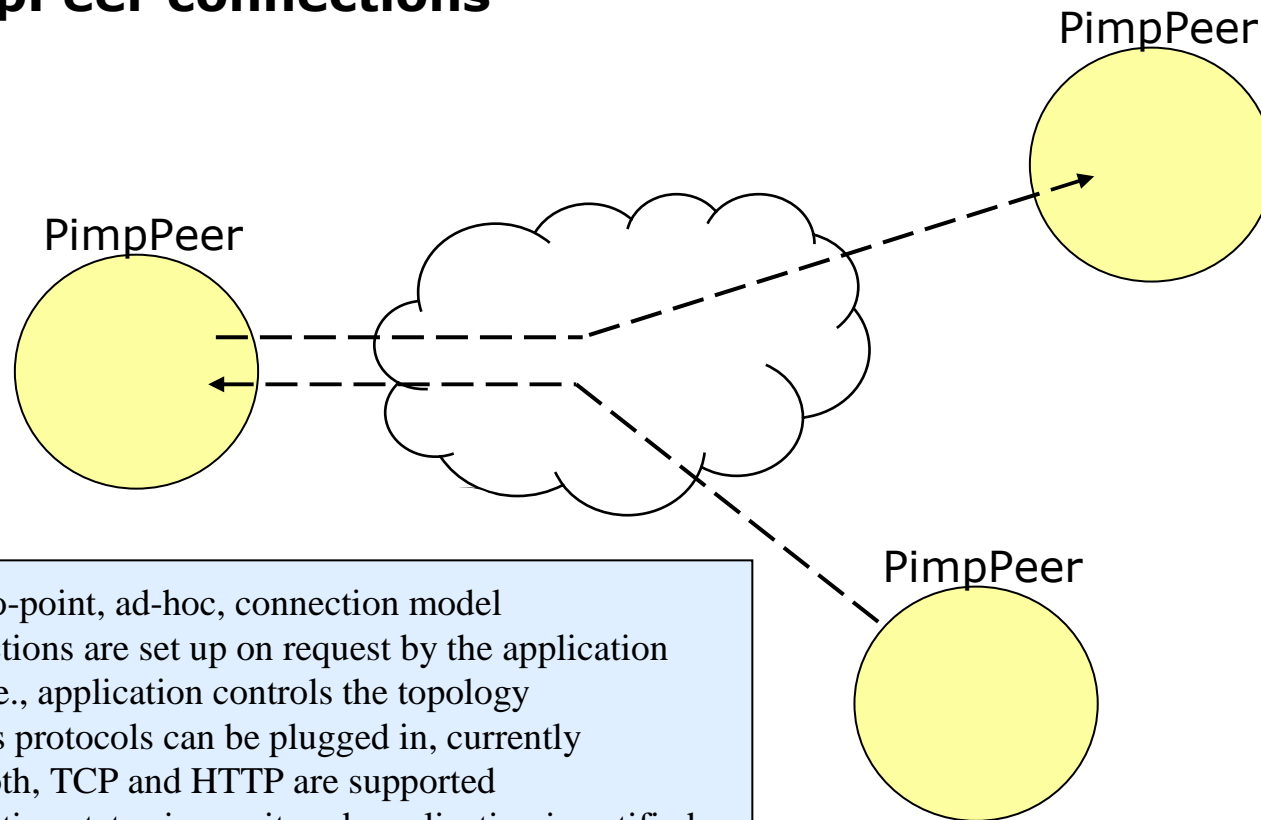
PimpPeer



- A PIMP enabled process that runs on some hardware device
- Is represented by a unique identifiers
- Connects to the (PIMP) network



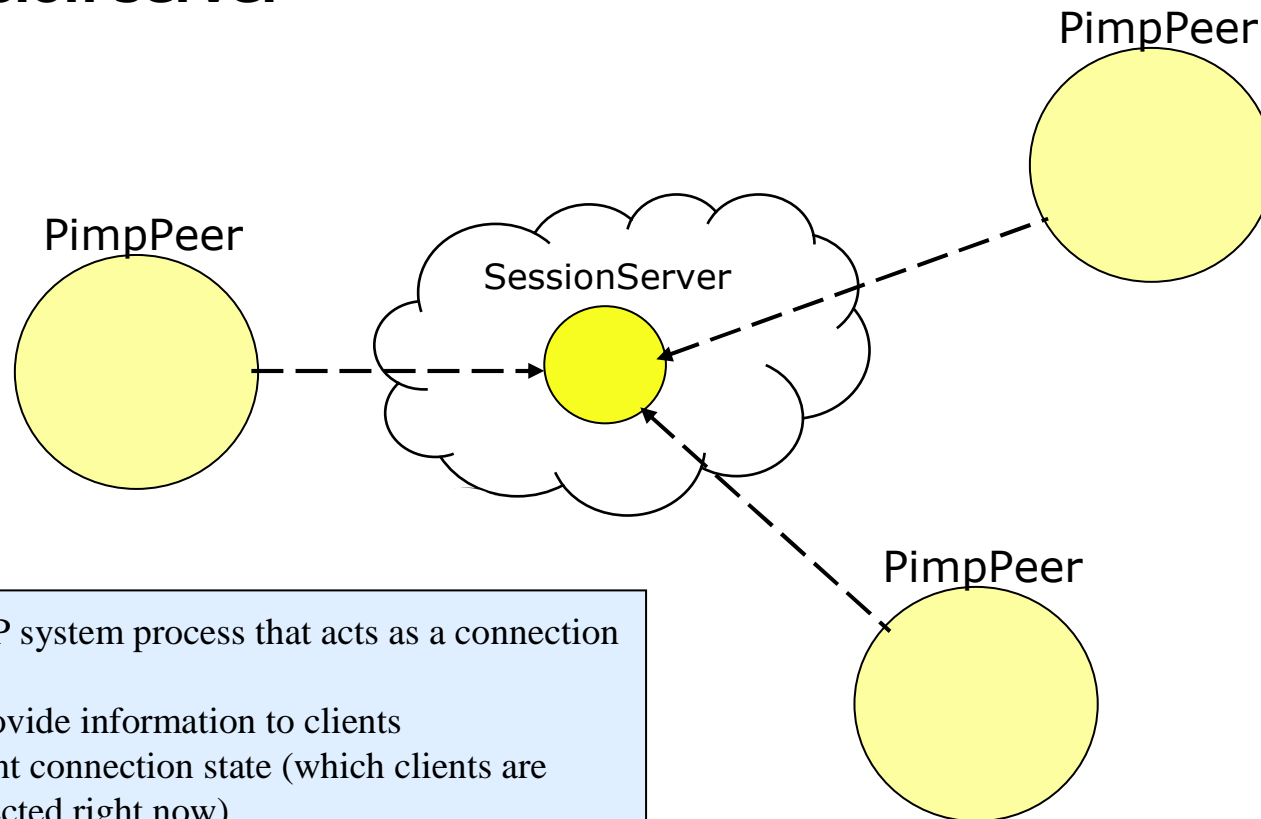
PimpPeer connections



- Point-to-point, ad-hoc, connection model
- Connections are set up on request by the application code, i.e., application controls the topology
- Various protocols can be plugged in, currently Bluetooth, TCP and HTTP are supported
- Connection status is monitored, application is notified
- Connections are re-established automatically if broken



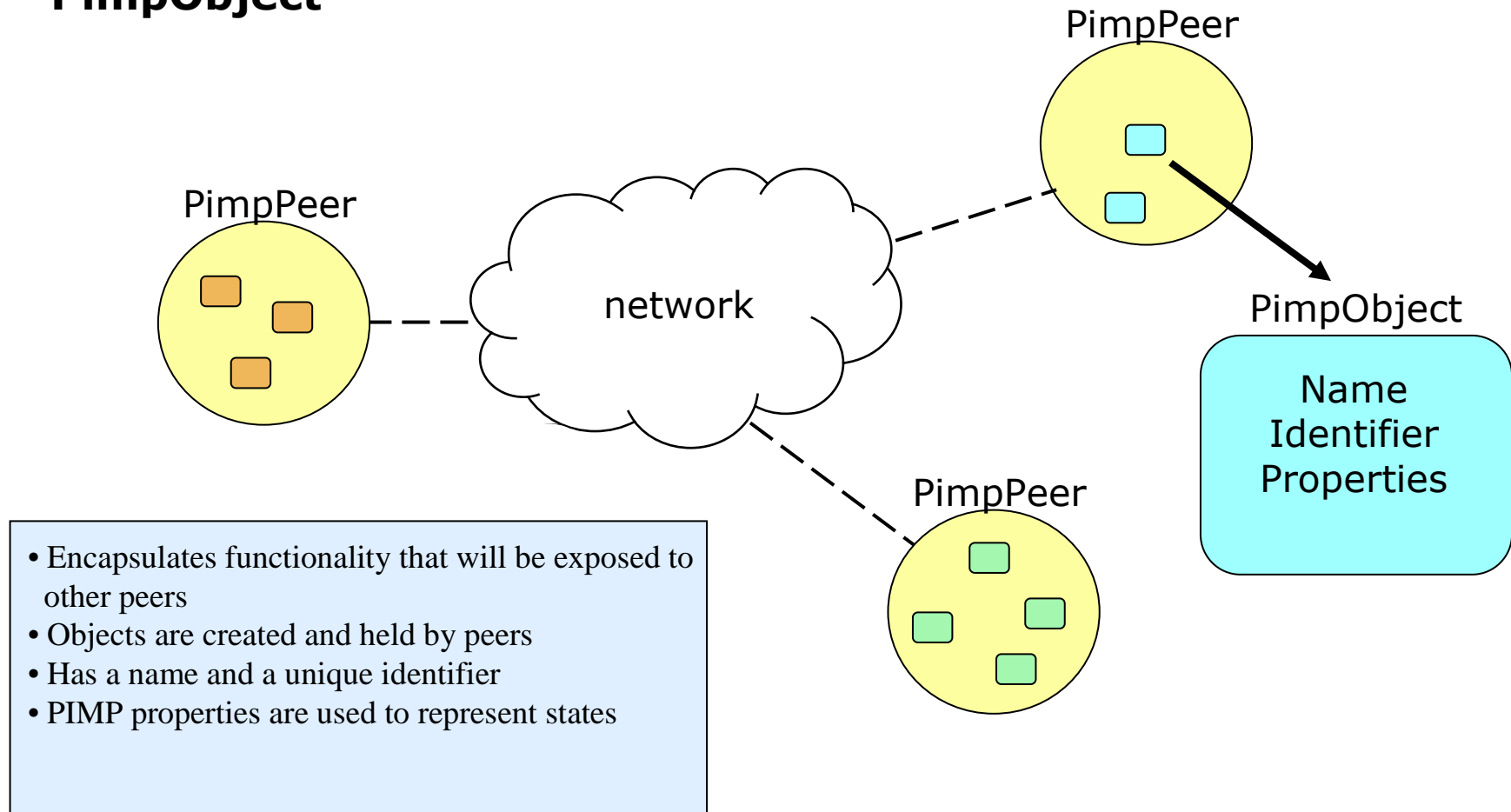
Session server



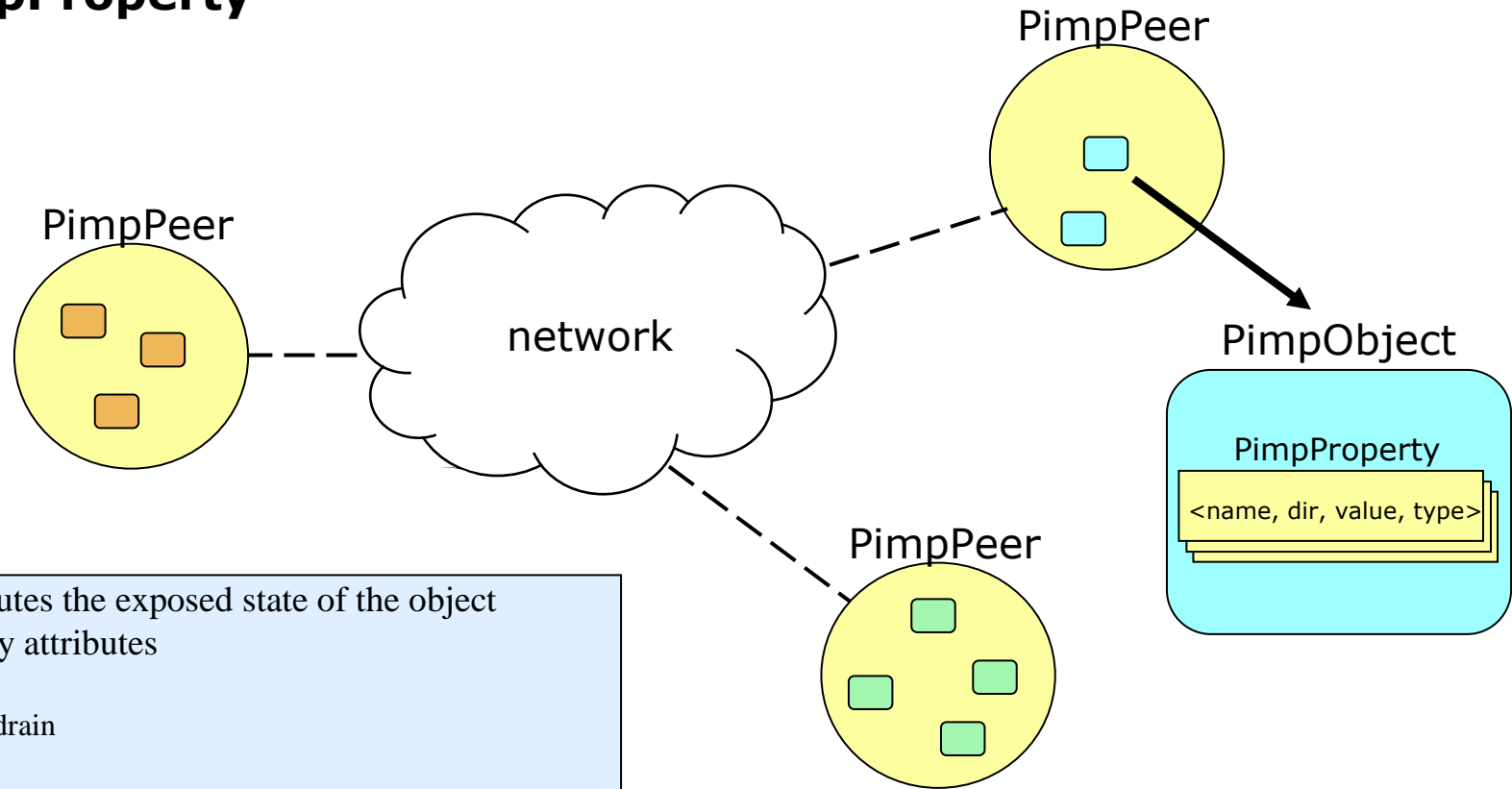
- A PIMP system process that acts as a connection server
- Can provide information to clients
 - current connection state (which clients are connected right now)
 - changes to the connection state (connects and disconnects)



PimpObject



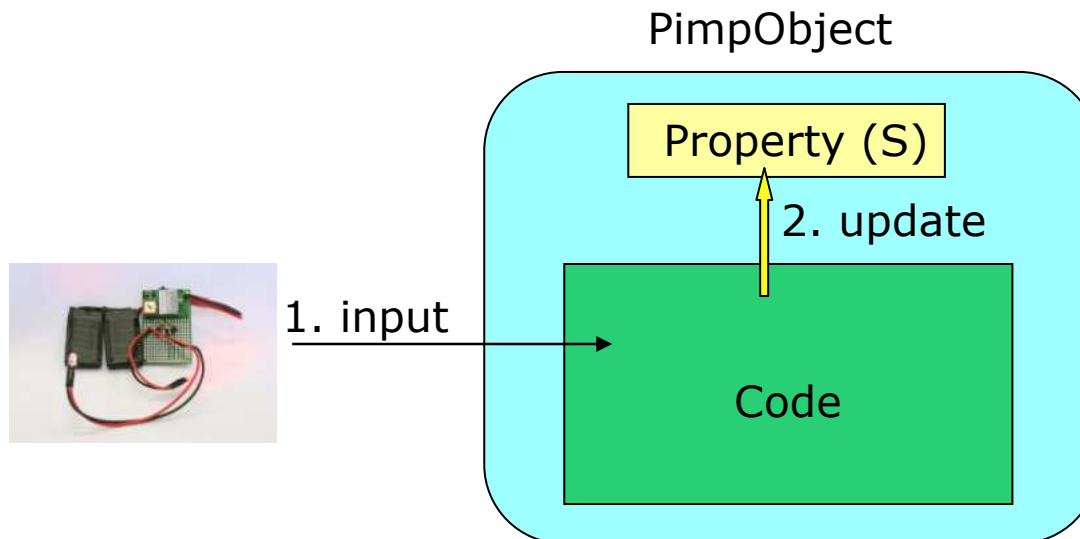
PimpProperty



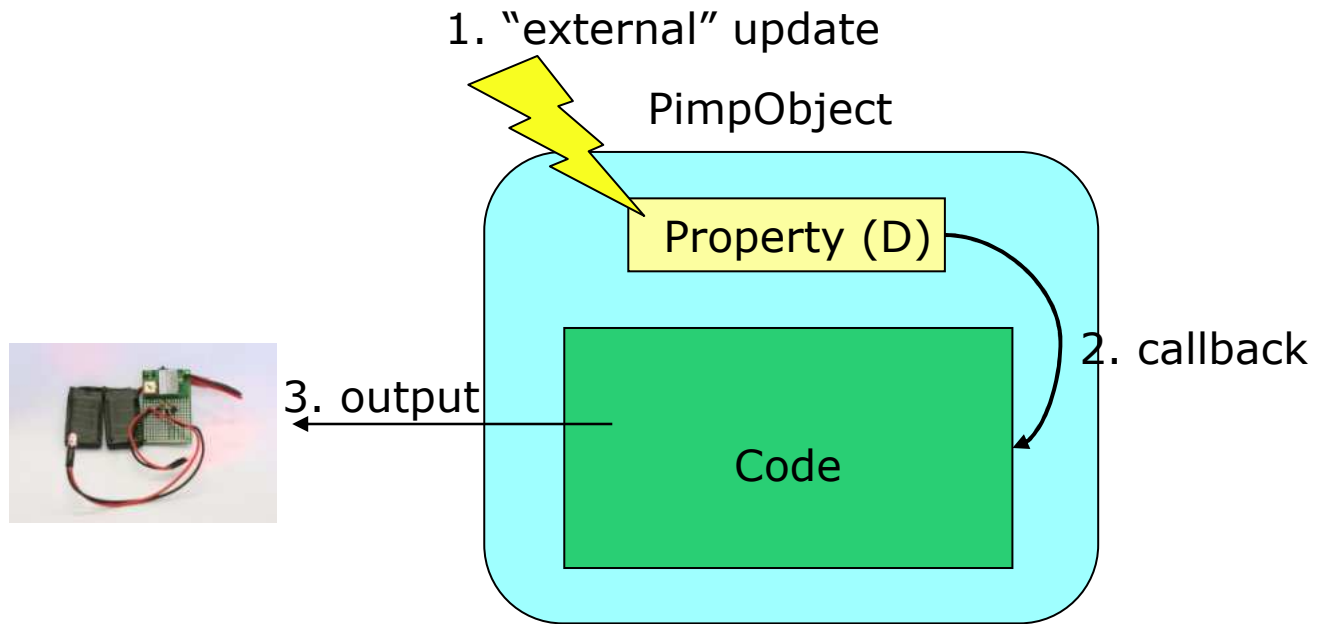
- Constitutes the exposed state of the object
- Property attributes
 - name
 - source/drain
 - value
 - value type
- Can be part of class definition or added to objects dynamically during runtime
- Introspection



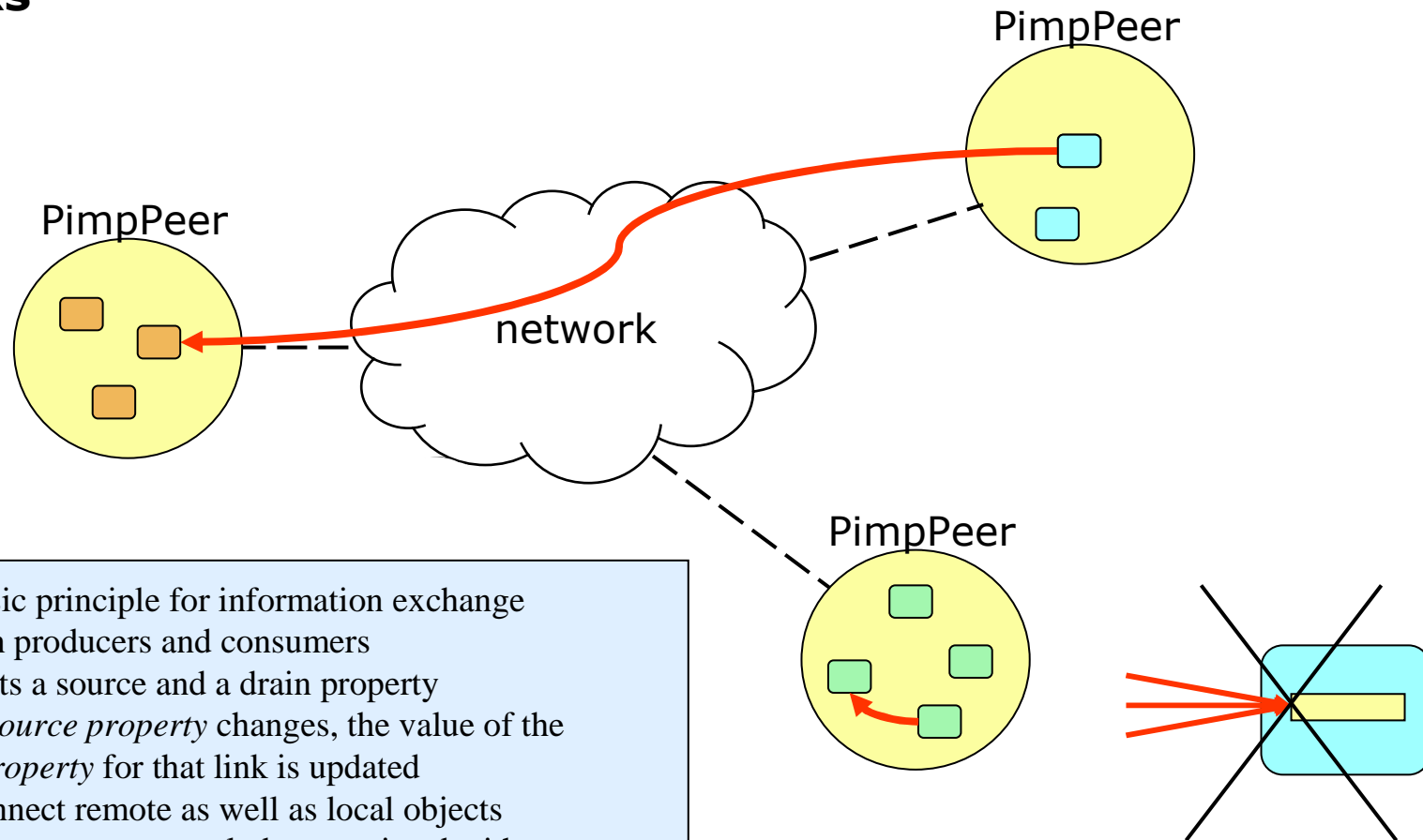
Producing data - source property (S)



Consuming data - drain property (D)

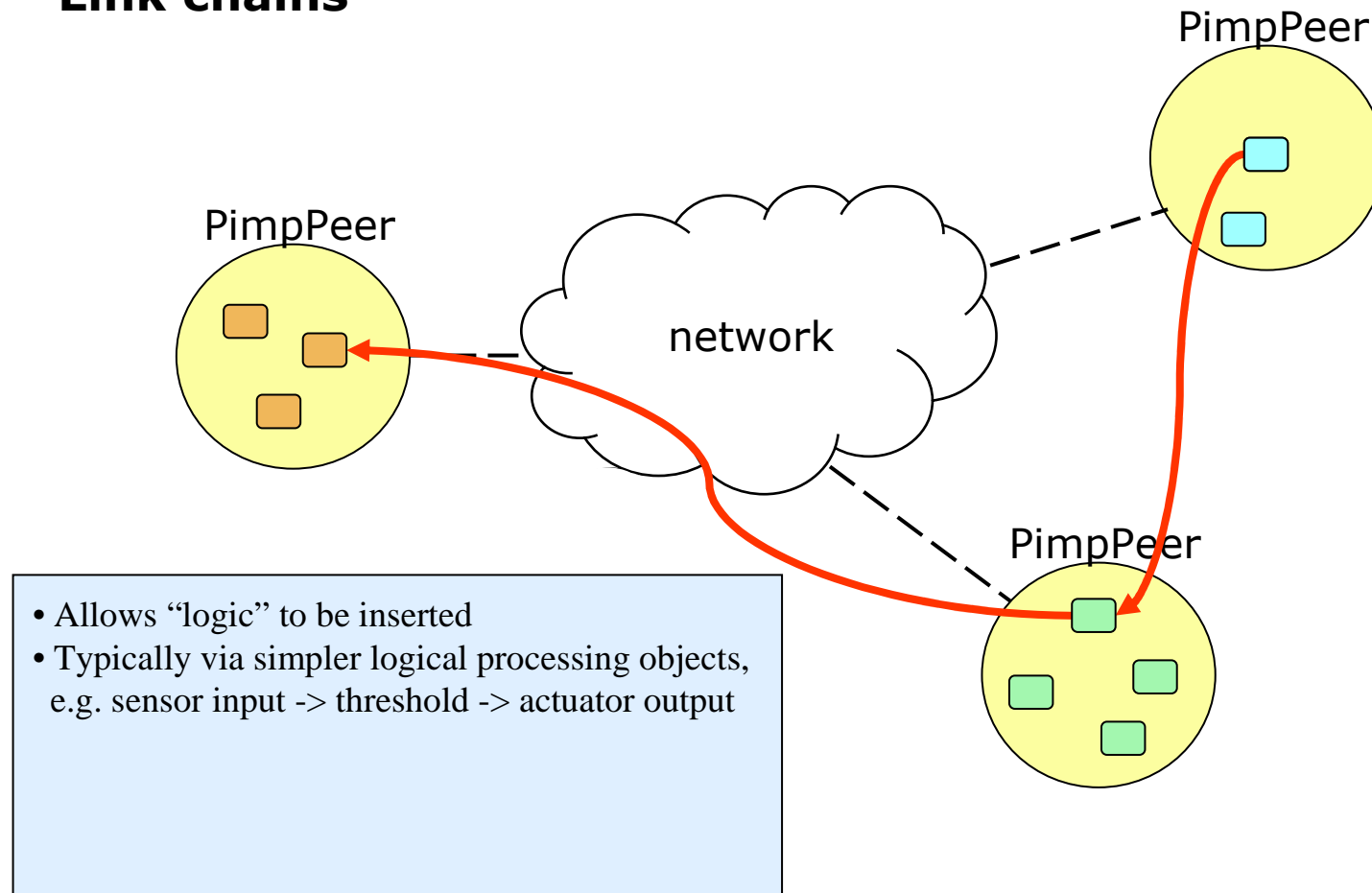


Links

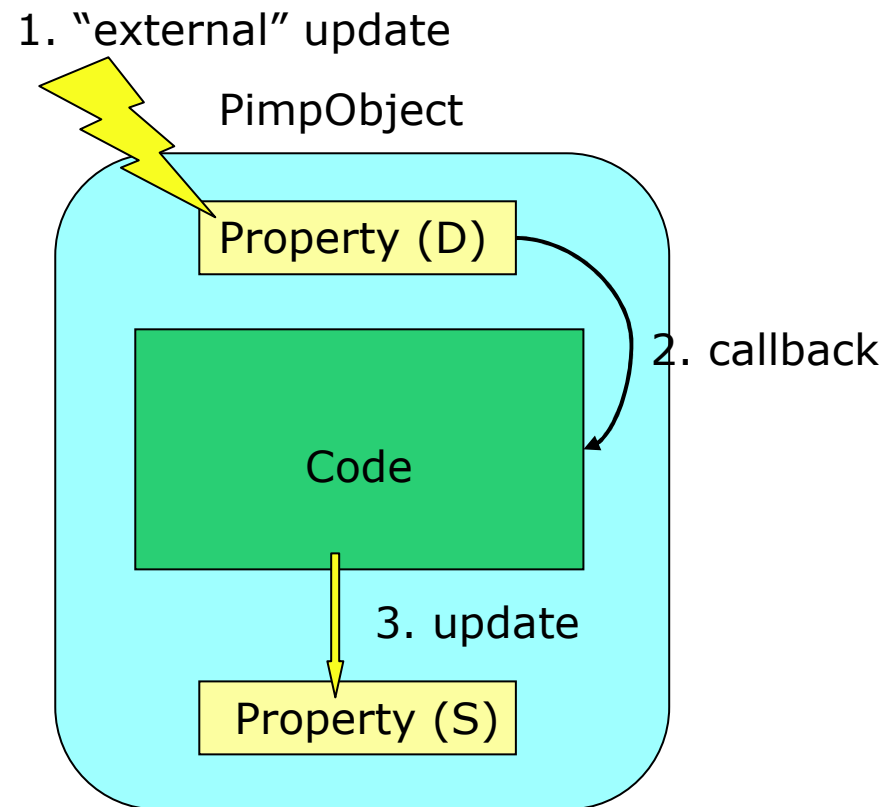


- The basic principle for information exchange between producers and consumers
- Connects a source and a drain property
- When *source property* changes, the value of the *drain property* for that link is updated
- Can connect remote as well as local objects
- A drain property can only be associated with one source

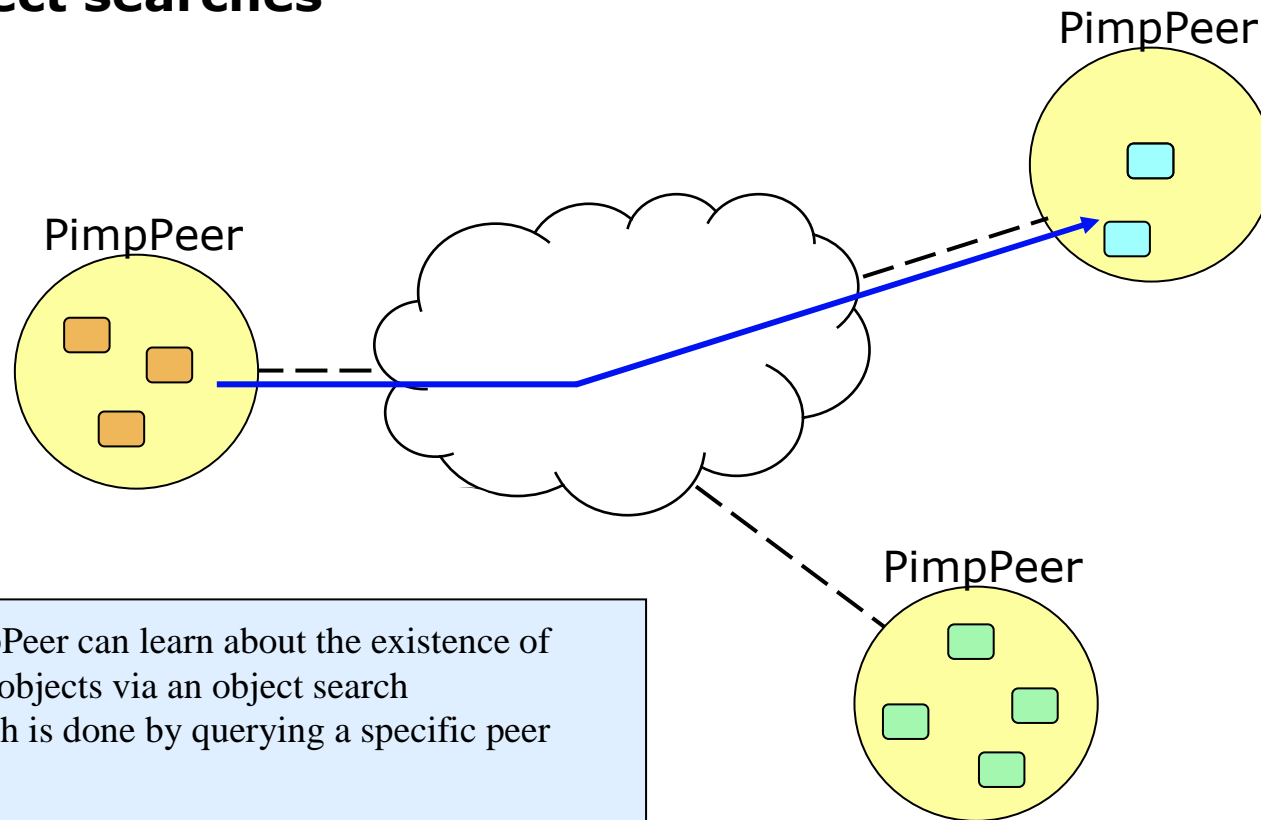
Link chains



Transforming data



Object searches



- A PimpPeer can learn about the existence of remote objects via an object search
- A search is done by querying a specific peer
 - name
 - class
 - property
- The remote peer will return the identifiers of objects matching the query

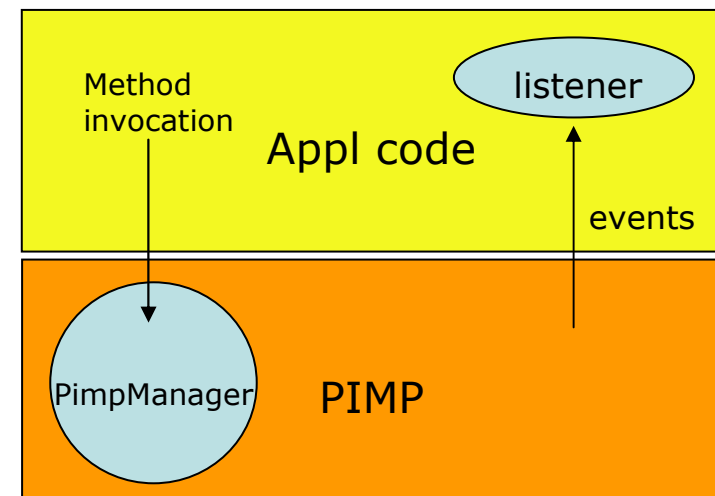
PIMP condensed

- ➔ **A *Pimp Object* is any java class that extends the PimpObject class. A PimpObject shall represent a well defined functional unit.**
- ➔ **A *Pimp Property* is a shared property for a PimpObject, shared in the sense that it can connect to or be connected to other PimpProperties .**
- ➔ **A *Link* is a connection between two PimpProperties . These links are uni-directional, with information transmitted from the source to the drain.**
- ➔ **A *Pimp Peer* is a Java application that creates a number of PimpObjects and through an instance of a PimpManager makes these interconnectable with other PimpObjects in other PimpPeers.**
- ➔ ***PimpManager* is a single instance class that manages links between PimpObjects in the local PimpPeer or remote ones.**
- ➔ ***PimpMidlet* is a skeleton for a PimpPeer on a mobile phone.**
- ➔ **A *Pimp Application* is a cluster of connected PimpPeers with interlinked PimpObjects.**



What is PIMP really?

- **A Java jar**
 - Exist both for Java 2 Standard Edition and Java 2 Mobile Edition
- **Will most likely become open source**
- **Runs on a variety of platforms**
 - PCs
 - Mobile phones
 - TINI embedded system
 - SNAP embedded system



Current status

- ➔ **Stable enough to run a game for several days that has been implemented with PIMP.**
- ➔ **Current version of the PIMP runtime includes a bit too much, e.g. support classes for scanning for Bluetooth devices and a generic serial communication library.**
- ➔ **The platform has not yet been used to create any complex setups where links are dynamically changed.**
- ➔ **Current version has not been tested towards any benchmarking requirements.**



Strengths and weaknesses

- ➔ **The major strength of PIMP is its model for how applications are created by combinations of independent components that can exist anywhere.**
 - o Openness towards the end-user
 - o Users can on a top level examine different components that make up a service.
 - o If required the user can easily remove or set up new connections between components.
 - o The model also supports hierarchical modeling thus an end-user can decide to examine one component in more detail by opening it up and see what sub-components it is made up of and how these are inter-connected.
- ➔ **The major weakness of the platform is that it requires careful design and a new model to learn for the application developer.**
 - o A simpler and a more straight forward solution seem more attractive in the beginning but once the different components starts to get re-used the strengths become apparent.



Goals and future plans

- ➔ **Clean up and minimize the core needed to implement a PIMP application.**
- ➔ **With the current software structure a PimpObject only implemented using Java 2 ME classes can not be accessed in any Java 2 SE due to underlying code**
- ➔ **Currently Pimp allows one source to be linked to several different drains, though one drain can only have one source connected in to it.**
 - Allow for multiple sources connected to one drain
- ➔ **Hierarchical modeling of components is not yet supported by the software.**
 - A future version should provide means to define a set of interconnected components as a super component as well as define what properties it exports as sources and drains.
- ➔ **The final purpose of the J2ME MIDlet is not set**
 - Currently used as a generic runtime with support for some specific external devices.
 - Clarify functionality and behavior
 - Plan for a plug-in behavior should be defined and implemented



How to create a PIMP application.

- **Conceptual walk through how to create a simple application**
- **Will explain suggestions for object and how these will be connected**
- **Task: Create a burglary alarm**
- **Available hardware**
 - Door sensor
 - Mobile phone with SMS

- **Idea – receive a SMS every time the door is open**



➔ **Door sensor**

- o DoorSensor object
- o DoorState property set to open or closed

➔ **Mobile phone**

- o SMSSendObject
- o Text property the text in the SMS
- o Sends SMS to receiver each time Text is changed

➔ **Need a way to enter some text**

➔ **Need to let a change in DoorState result in a SMS**



➔ **Text input**

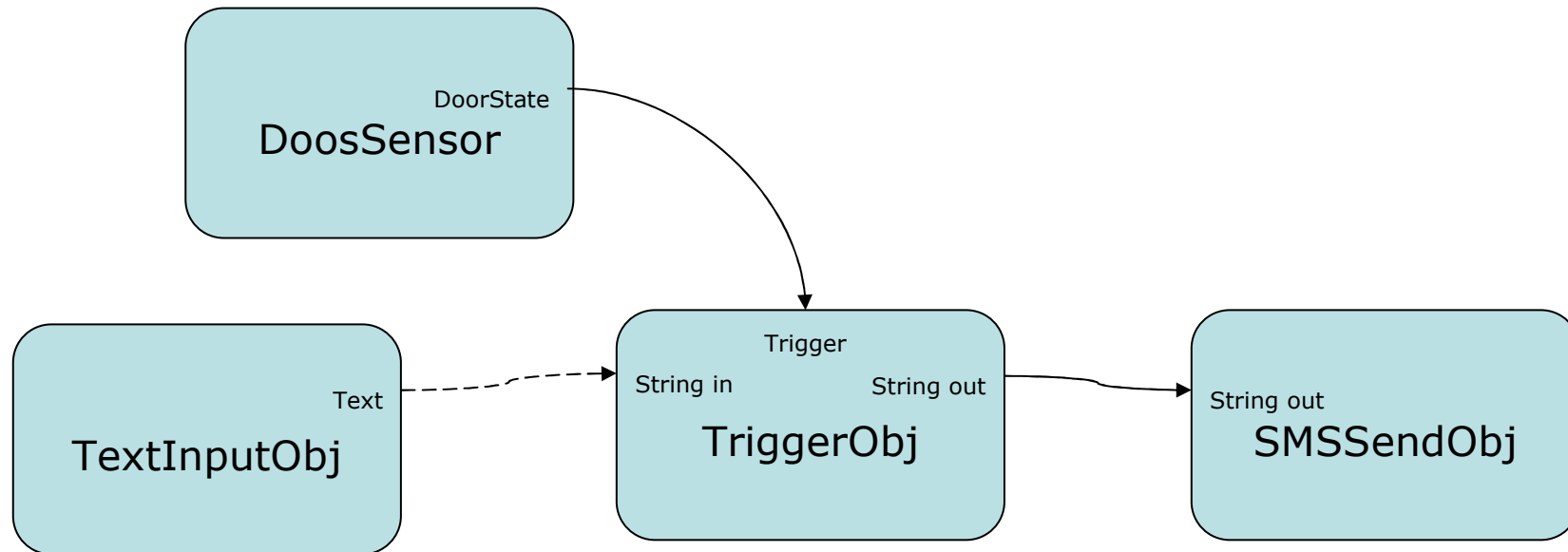
- o TextInputObject
- o Text property changed each time the user enter a new text

➔ **Create a trigger**

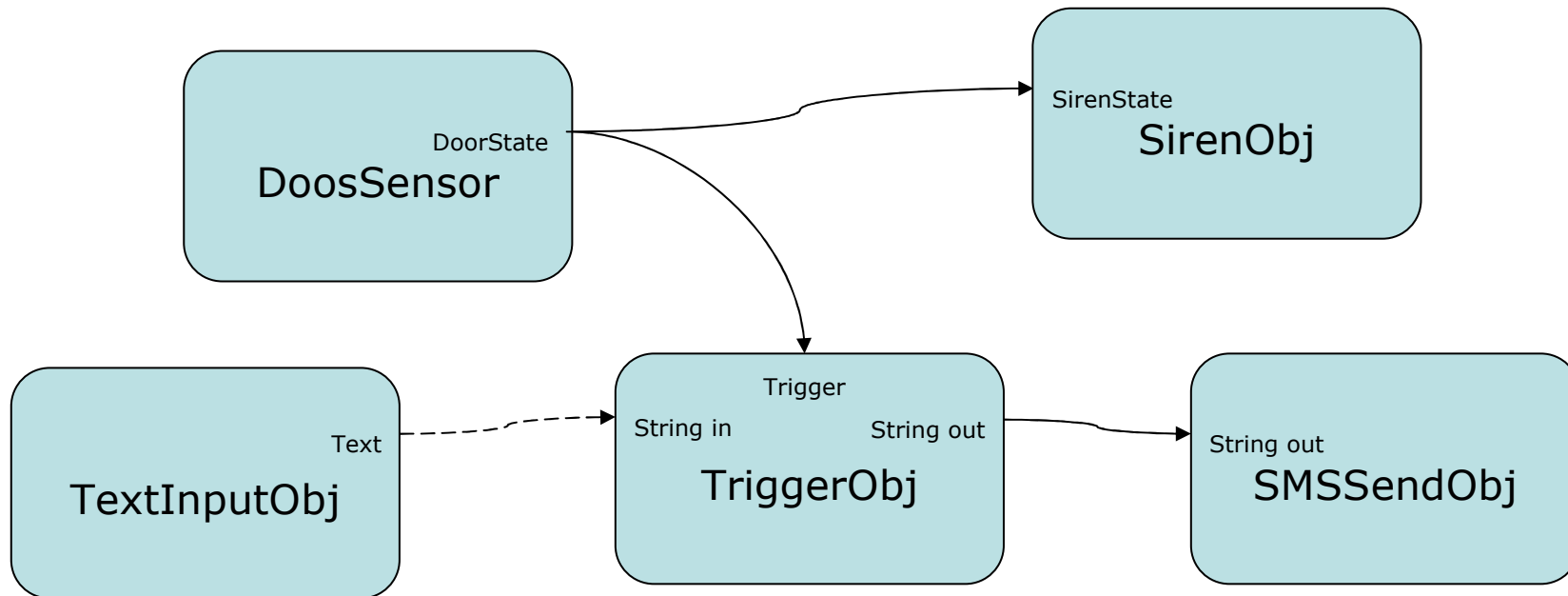
- o TriggerObject
- o Sets an output property to the value of a corresponding input property when a trigger property changes



Links



Simple to extend and re-use



How to write programs that use Pimp

- **Create PIMP objects**
- **Create PIMP peers**
- **Create links**



Creating PIMP objects

- ➔ **Decide which classes to use**
 - Sub-class PimpObject
 - Instantiate PimpObject
- ➔ **Decide which properties are needed**
 - Property creation can happen in the object class constructor
=> all instances will have these properties
 - Properties can also be added dynamically to individual instances
- ➔ **Make discoverable for remote peers**
- ➔ **Write property object callback code (in classes that use drain properties)**



Sub-classing PimpObject

➔ **The “standard” constructor should call the super (PimpObject) constructor with a name, description and identifier argument.**

➔ **Name**

- o Shall be meaningful for a user

➔ **Description**

- o Shall explain the objects functionality to a user
- o Mention the properties
- o E.g. “The property ‘output’ is set to the value of the ‘input’ property when the trigger property ‘trigger’ is changed.”

➔ **Add an empty public constructor**

```
public class MyClass extends PimpObject{
    public MyClass() {
    }

    public MyClass(String name) {
        super(name, objDescr, null);

        createPimpProperty(propName
                            initialValue,
                            direction,
                            valueType);

        setDiscoverable(true);
    }
    public void update(String propName) {
        ...
    }
}
```



Sub-classing PimpObject

- ➔ **Create properties using createPimpProperty() with name, initial value, direction and type of the value arguments.**
- ➔ **Name**
 - Shall be meaningful for a user
- ➔ **Initial value**
 - The value the property shall have when created
- ➔ **Direction**
 - Either PimpObject.SOURCE or PimpObject.DRAIN
- ➔ **Type of the value**
 - The type of the property
 - Types defined in org.iperg.pimp.types

```
public class MyClass extends PimpObject{
    public MyClass() {
    }

    public MyClass(String name) {
        super(name, objDescr, null);

        createPimpProperty(propName
                            initialValue,
                            direction,
                            valueType);

        setDiscoverable(true);
    }
    public void update(String propName) {
        ...
    }
}
```



Make discoverable for remote peers

- ➔ **setDiscoverable(true) makes this object and its properties visible to remote peers performing searches**

```
public class MyClass extends PimpObject{
    public MyClass() {
    }

    public MyClass(String name) {
        super(name, objDescr, null);

        createPimpProperty(propName
                            initValue,
                            direction,
                            valueType);

        setDiscoverable(true);
    }
    public void update(String propName) {
        ...
    }
}
```



Add a property update callback method

- ➔ Add a `update()` method
- ➔ Called each time a Drain property is updated. `propName` is set to the name of that property.

➔ e.g.

```
if(propertyName.equals("trigger")) {  
    setPimpProperty("output",  
        getPimpProperty("input"));
```

```
public class MyClass extends PimpObject{  
    public MyClass() {  
    }  
  
    public MyClass(String name) {  
        super(name, objDescr, null);  
  
        createPimpProperty(propName  
            initialValue,  
            direction,  
            valueType);  
  
        setDiscoverable(true);  
    }  
    public void update(String propName) {  
        ...  
    }  
}
```



Instantiate PimpObject

- ➔ **Similar code structure as in the sub-classing case**
- ➔ **Needs to register for callbacks**

```
public class MyClass implements
    PimpObjectUpdateCallbackListener {
    public MyClass() {
    }

    public void setup() {
        PimpObject myObj = new
            PimpObject(name, objDescr, null);

        myObj.addPimpProperty(propName
            initialValue,
            direction,
            valueType);
        myObj.setDiscoverable(true);

        myObj.addUpdateCallback(this);
    }

    public void update(String propName) {
    }
}
```



How to create a PimpPeer

- ➔ **Different kinds of Peers**
 - Clients
 - Configurators
- ➔ **Only informs a SessionServer to treat them slightly differently**

- ➔ **Basic steps for all PimpPeers**
- ➔ **1. Retrieve network settings**
- ➔ **2. Create a connection**



How to create a PimpPeer

➔ Retrieve network settings

➔ XML Java properties file

```
<entry key="ServerHost">localhost</entry>
<entry key="ServerPort">1000</entry>
<entry key="ClientPort">4242</entry>
```

```
NetworkSettings settings = new
    NetworkSettings();
try {
    settings.readSettings();
} catch (FileNotFoundException fnfe) {
    Info.message("ERROR: " +
        fnfe.getMessage(), 1);
    System.exit(-1);
} catch (IOException ioe) {
    Info.message("ERROR: " +
        ioe.getMessage(), 1);
    System.exit(-1);
}
```



How to create a PimpPeer

➔ Establish connection

```
PimpManager pm = new PimpManager(procName,  
    PimpProcess.CLIENT,  
    "tcp://" + settings.getServerHost() + ":" +  
    settings.getServerPort(),  
    settings.getClientPort(), null, null);  
  
pm.connectToPart();
```



Create links

➔ Find remote objects

```
IpIdentifier input[] = PimpManager.findPimpObjectsByName(client1Id,  
    "Input", 1);
```

```
IpIdentifier output[] = PimpManager.findPimpObjectsByName(client2Id,  
    "Output", 1);
```

➔ Create the link

```
PimpManager.createPropertyLink(client1Id, input[0], "Value",  
    client2Id, output[0], "Value");
```



Example peers

➔ A peer hosting an object

- o Establish connection
- o Create an instance of the PimpObject
- o Change or react on changes in PimpProperties

➔ A peer manipulating links

- o Listen to events, e.g. CLIENT_CONNECTED
- o Establish connection
- o React on the different clients that connect/disconnect e.g. create a link



Create a Link manipulator

➔ Event handler – catch active peers

```
public void handleEvent(IpEvent event) {
    if (event.getType().equals(PimpProcess.CLIENT_CONNECTED)) {
        IpNetworkEvent e = (IpNetworkEvent)event;
        try {
            IpInputStream s = e.getPayload();
            IpIdentifier clientId = (IpIdentifier)s.readObject();
            String clientName = s.readString();
            // store the clientId and clientName
        } catch(IOException ioe) {
            Info.message(this, ".handleEvent(): " + ioe.getMessage(), 1);
        }
    }
}
```



Create a Link manipulator

➔ Listen to events

```
IpSystem.addEventHandler(this);
```

➔ Establish connection

```
PimpManager pm = new PimpManager(procName,  
    PimpProcess.CONFIGURATOR,  
    "tcp://" + settings.getServerHost() + ":" +  
    settings.getServerPort(),  
    settings.getClientPort(), null, null);  
pm.connectToPart();
```



- **Three skeleton files**
 - o InputPeer.java
 - o OutputPeer.java
 - o LinkingPeer.java
- **Create a object with a source property reflecting number from the input field**
- **Create an object with a drain property that will display the value of the property**
- **Create the code to setup a link between the input property and the output property**
- **java -cp pimp_j2se.jar InputPeer**
- **java -cp pimp_j2se.jar org.iperg.pimp.session.SessionServer**

