

# MoCha

a coordination middleware based  
on mobile channels

**Juan Guillen-Scholten**

juan@cwil.nl

<http://homepages.cwi.nl/~juan>

Invited lecture at the  
“Coordination and Component Composition”  
Course

Leiden University, April 2005.



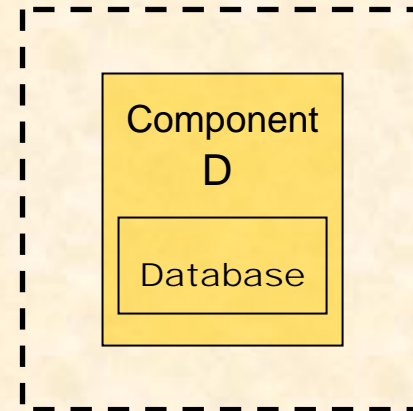
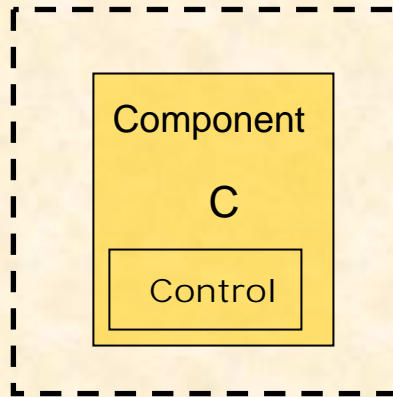
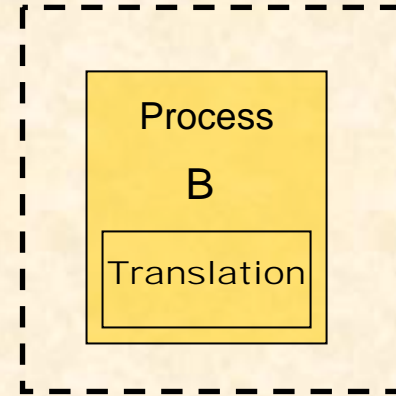
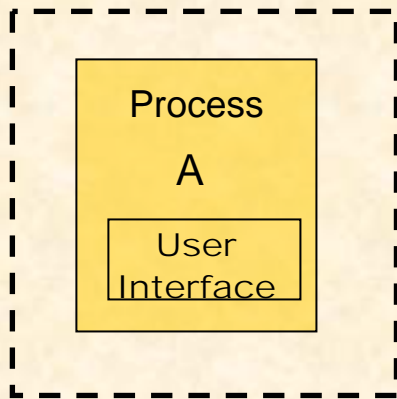
# Overview



- ✓ Coordination/communication
- ✓ Channels
  - Benefits of Channels
  - Mobility of Channels
  - Mobile Agent Example
- ✓ MoCha
- ✓ easyMoCha
  - Interface
  - Channel types
- ✓ Demo
- ✓ Conclusion & Future work

# Coordination/Communication

---



How to compose and coordinate processes/components in distributed environments.

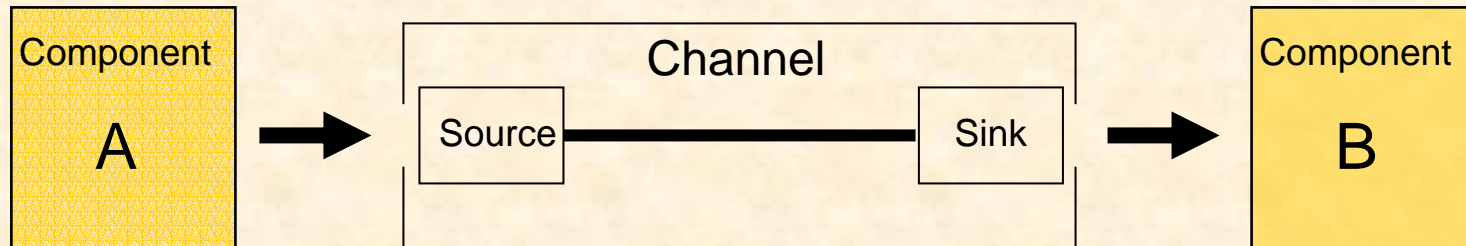
# Coordination/Communication

---

- Several communication and coordination mechanisms:
  - ✓ **Synchronous communication.** [RPC, RMI]
  - ✓ **Messaging.** [JMQ, JMS, COM+]
    - explicit targeted
    - publish-and-subscribe
  - ✓ **Events.** [JavaBeans]
    - Producer -> consumer/event listeners
  - ✓ **Shared data Spaces** [Linda, JavaSpaces, SPLICE]
    - tuples -> <- shared data space
  - ✓ **Channels** [JCSP, Pict, Nomadic Pict, MoCha]

# Channels

---



- A channel consists of two distinct ends: Usually (*source, sink*), but also (*source, source*) and (*sink, sink*).
- Components write to the source-end, and read from the sink-end.
- The dataflow is locally *one-way*.
- Channels are point-to-point/peer-to-peer.
- The communication is *anonymous*, the components do not know each other.
- Channels can be synchronous or asynchronous (FIFO, set, bag, etc).
- Channels can be mobile.

# Benefits of Channels

---

In comparison with the other coordination mechanism, channels share many of their architectural strengths while offering some benefits.

Major commonalities and additional benefits are:

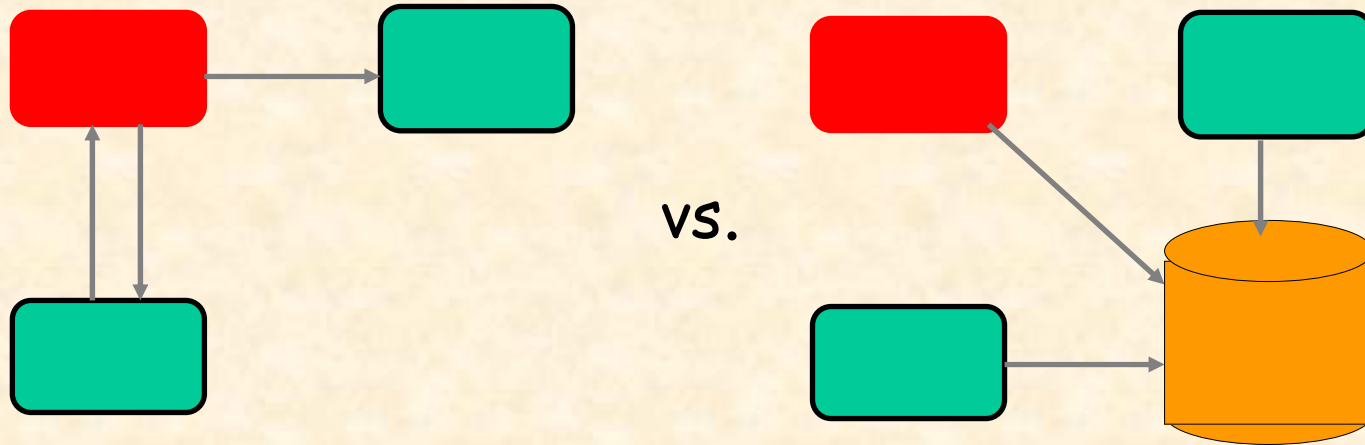
✓ *Security.*

- no possibility for a third party to read the data while it is in the channel

# Benefits of Channels

---

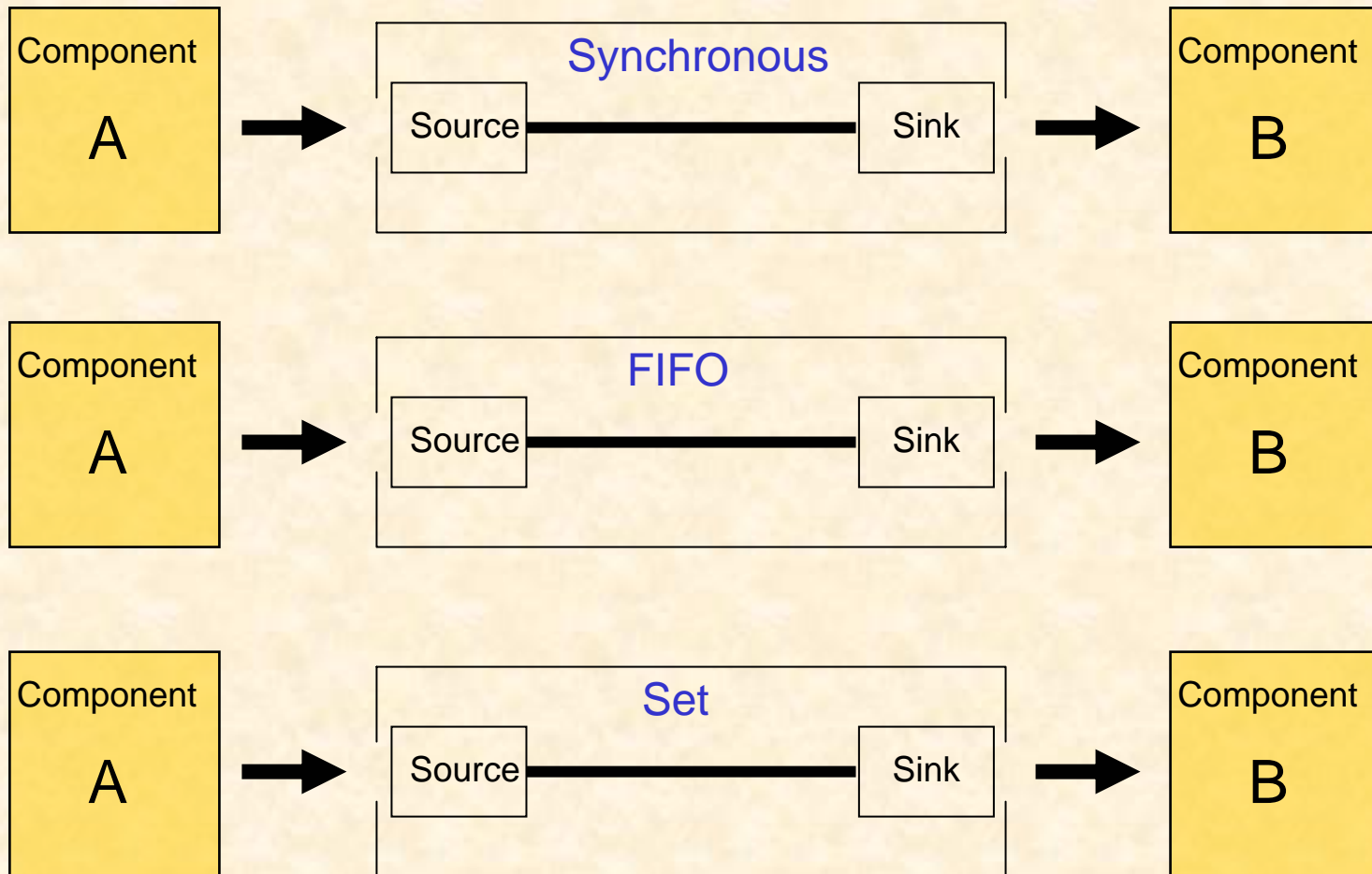
✓ *Architectural Expressiveness.*



# Benefits of Channels

---

✓ *Transparent exogenous coordination.*

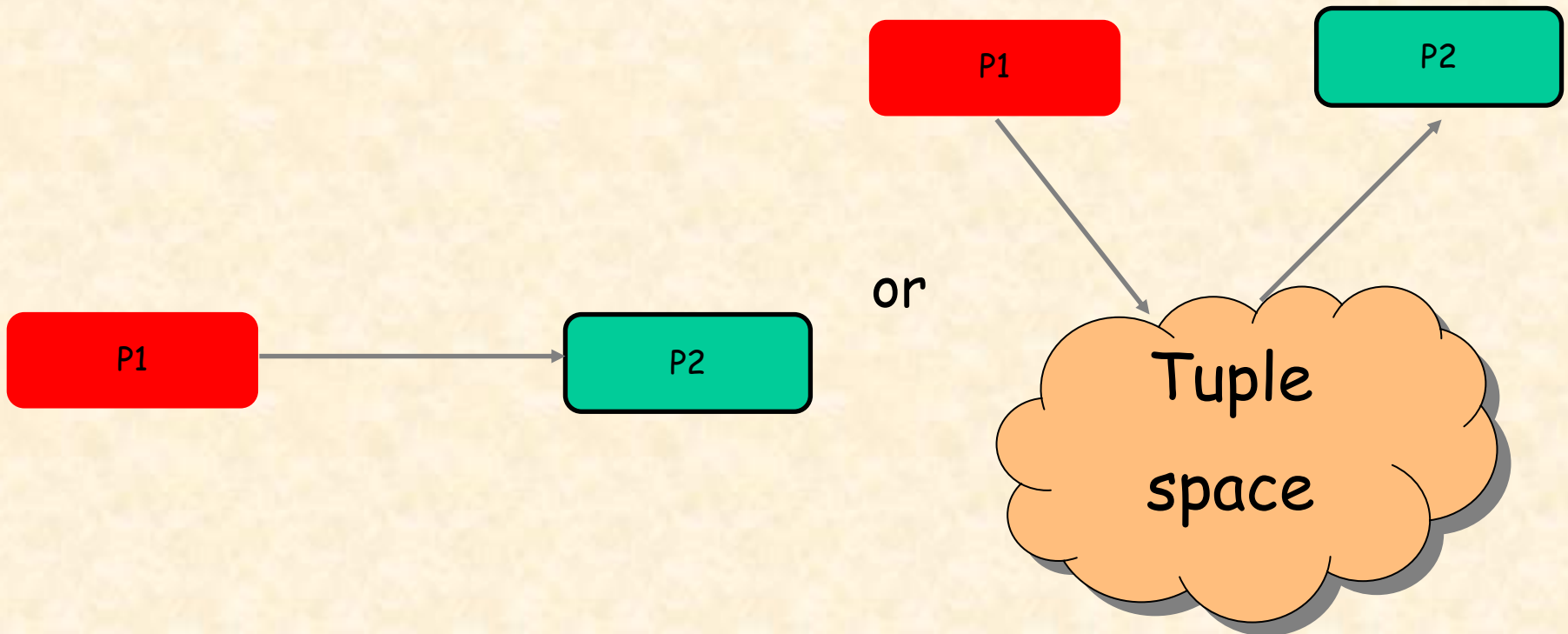


# Benefits of Channels

---

✓ *Efficiency.*

- Channels can efficiently implement both point-to-point and share data space architectures.

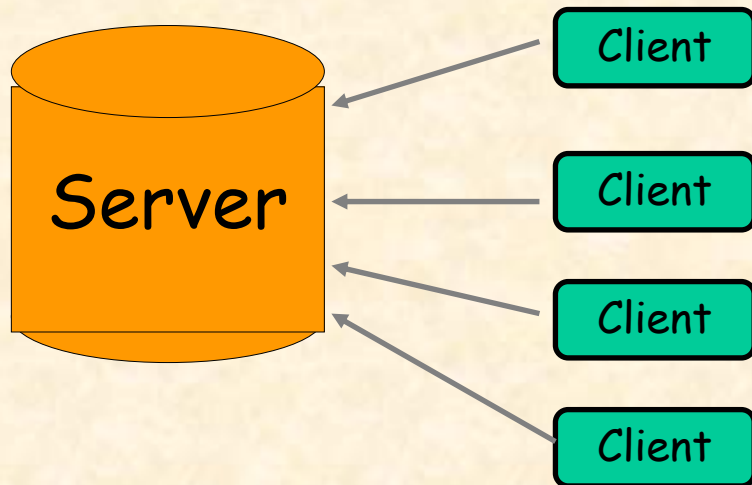


# Benefits of Channels

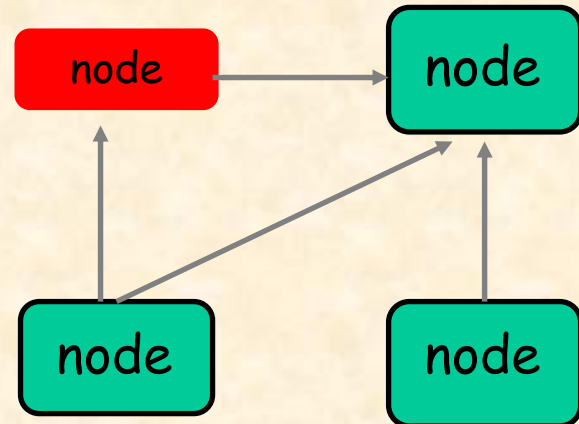
---

✓ *Efficiency.*

- Channels can efficiently implement centralized client/server and decentralized peer-to-peer architectures.

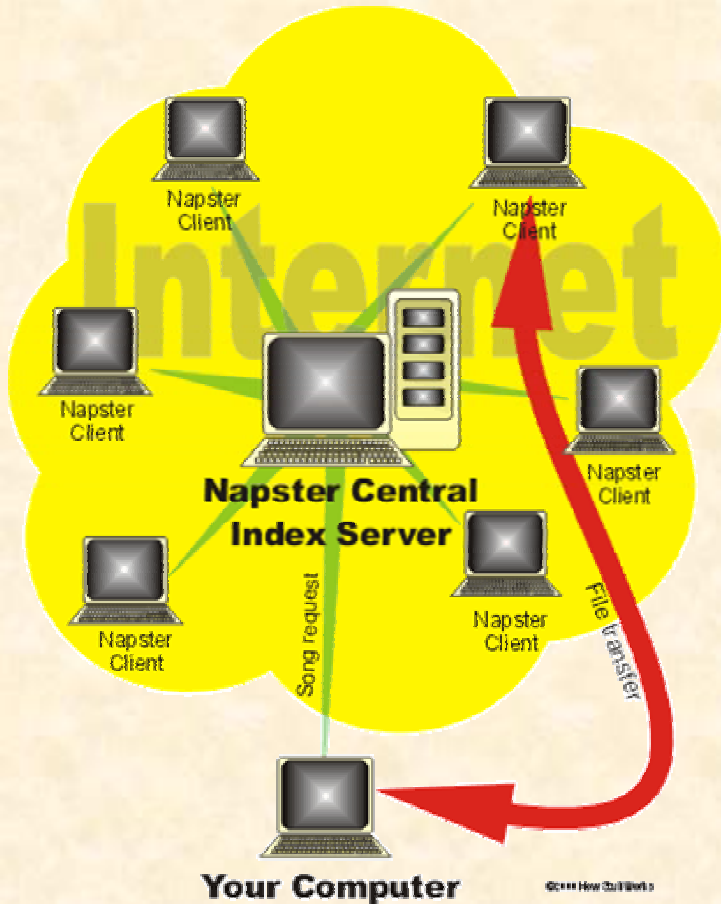


or



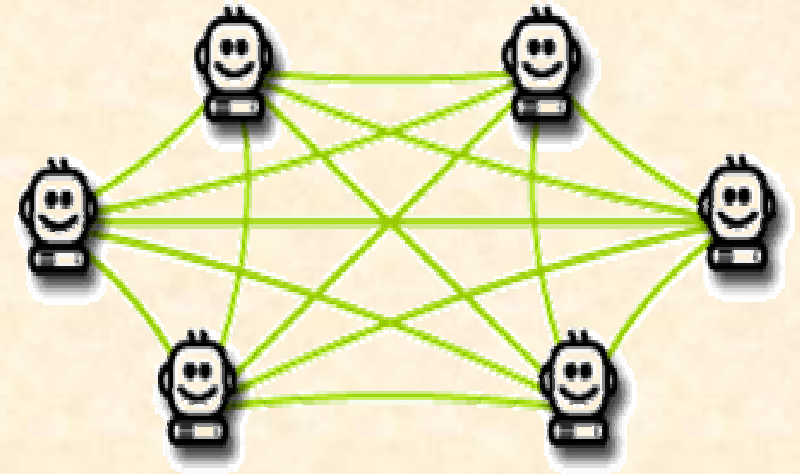
# Benefits of Channels

Example of client/server architecture and peer-to-peer



**Napster**

(half client/server, half p2p)



**Kazaa Lite**

(fully peer-to-peer)

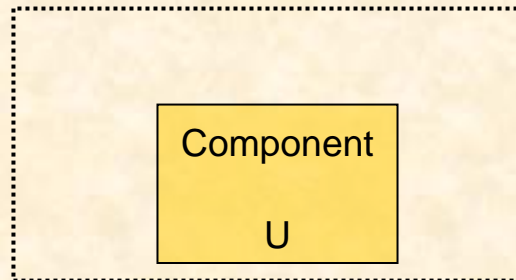
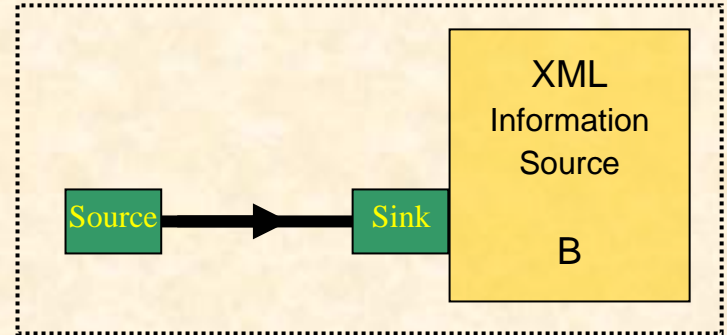
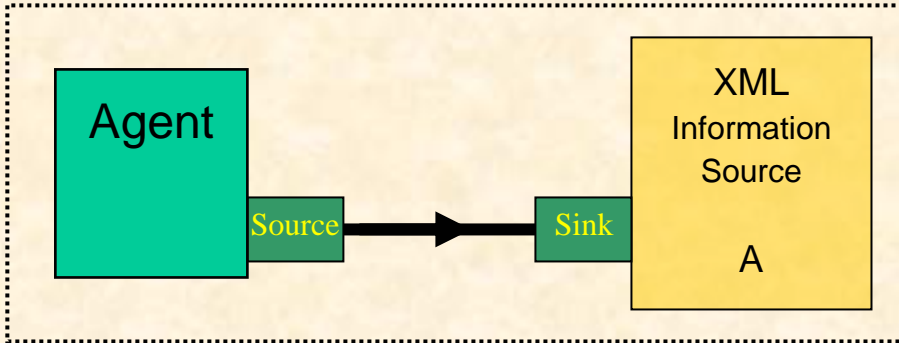
# Mobility of Channels

---

- *Logical* mobility: A channel is called *mobile* when the identities of its channel-ends can be passed on through channels to other components in the system.
- *Physical* mobility: In distributed systems, the ends of a mobile channel can physically move from one location to another.
- Because communication via channels is anonymous, when a channel-end moves the component(s) at its other end do not notice anything of the movement.
- Mobility allows *dynamic reconfiguration* of channel connections among components in a system, a property that is very useful and even crucial in systems where the components themselves are mobile.

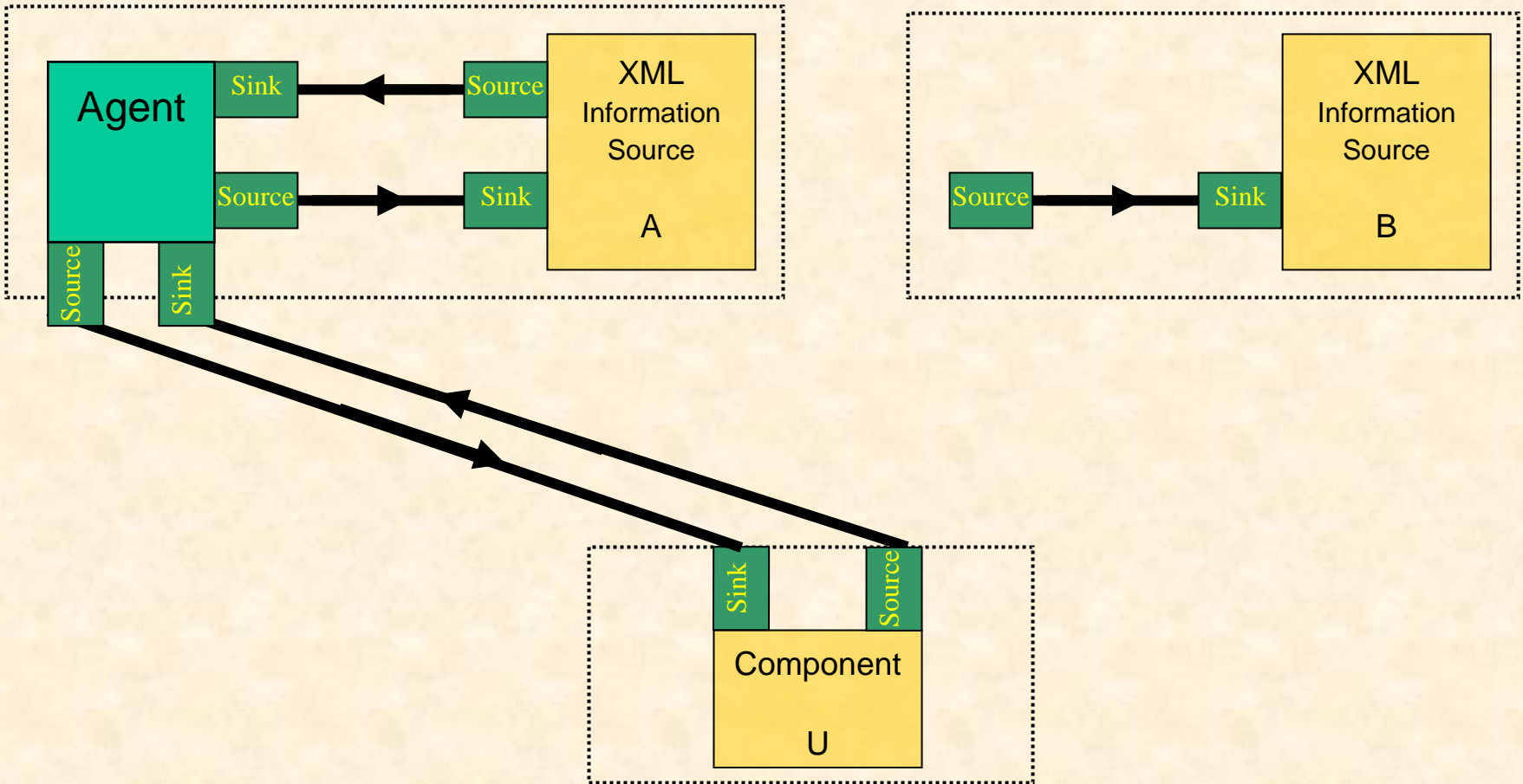
# Mobile Agent Example

---



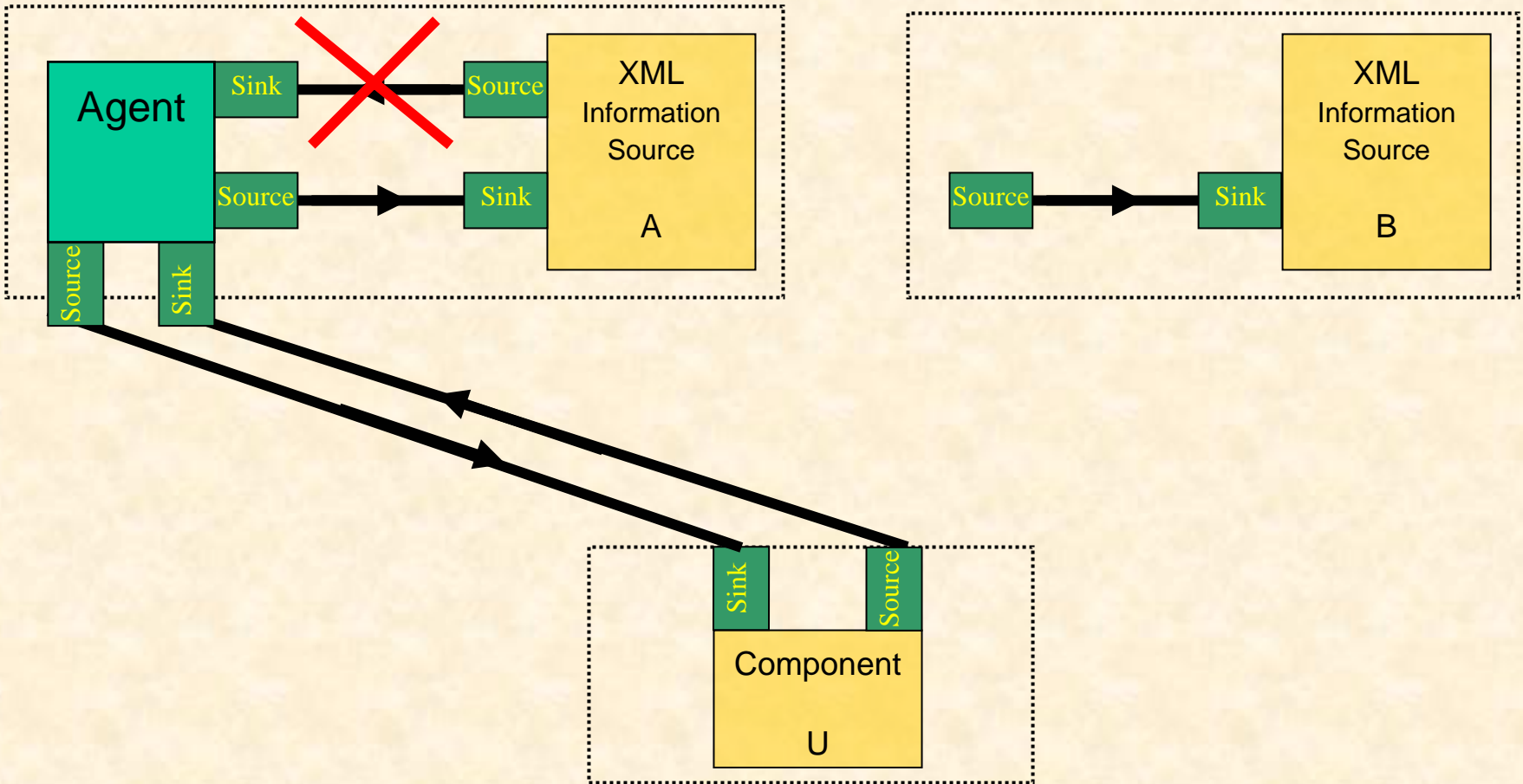
An Example: A mobile agent looking for Mocha-bean prices.

# Mobile Agent Example



An Example: A mobile agent looking for MoCha-bean prices.

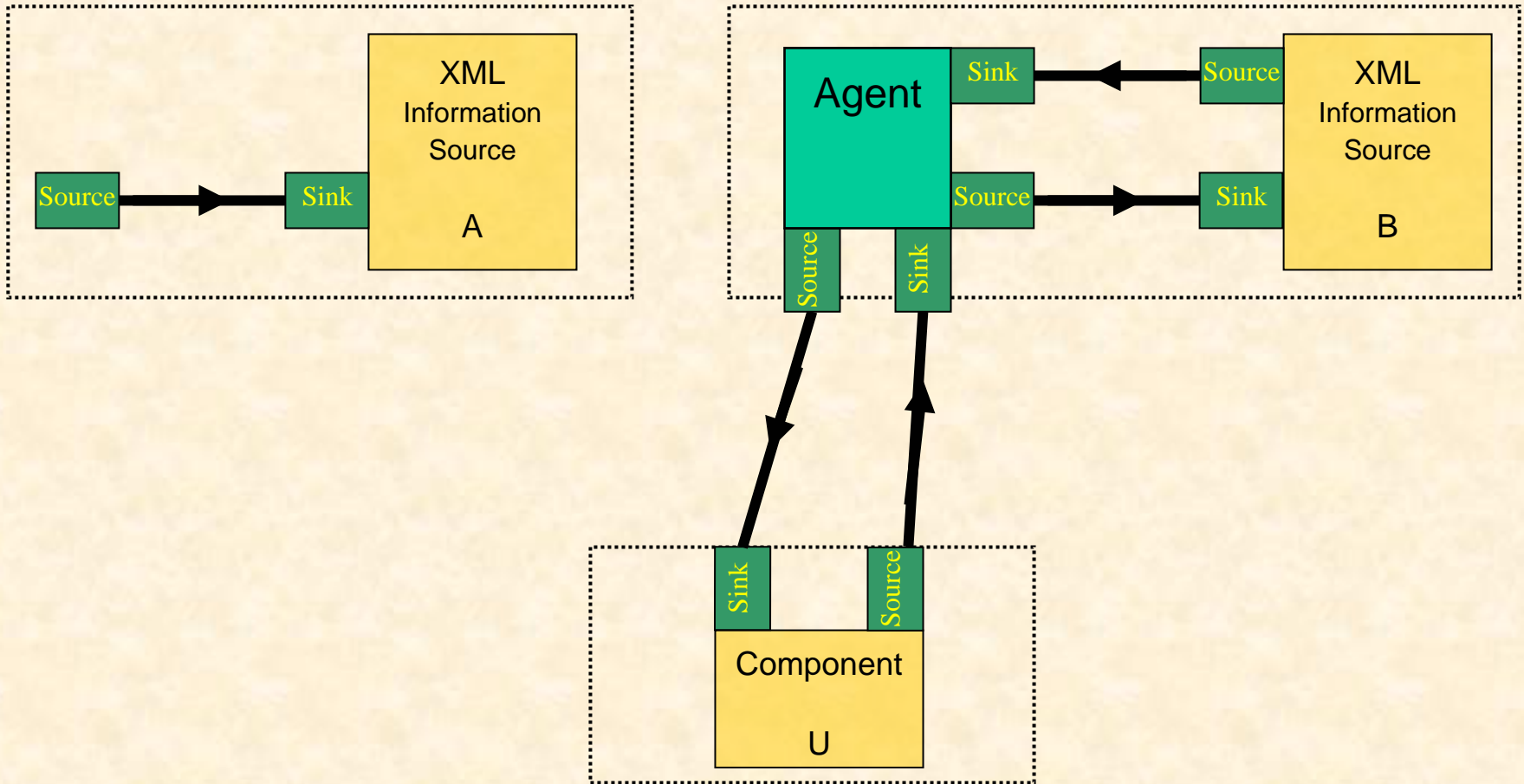
# Mobile Agent Example



An Example: A mobile agent looking for MoCha-bean prices.

# Mobile Agent Example

---

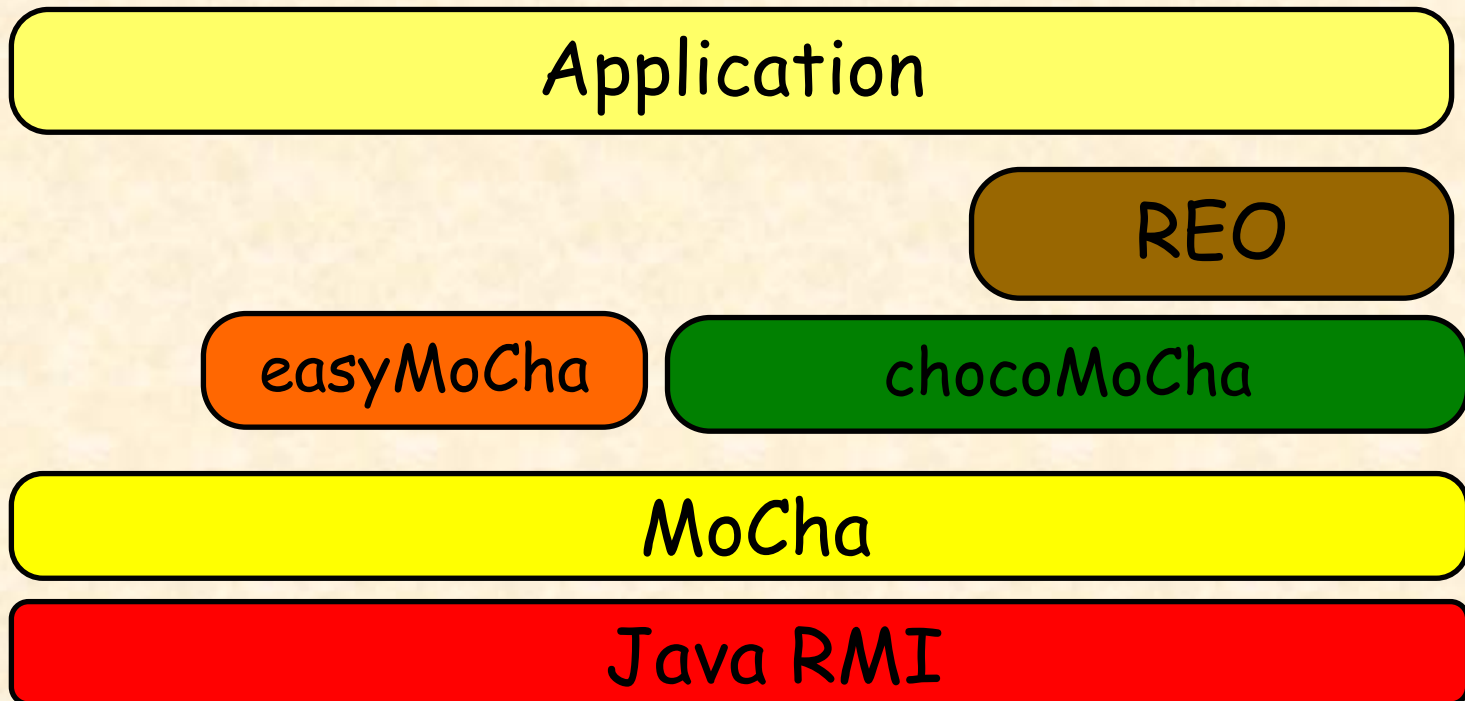


An Example: A mobile agent looking for MoCha-bean prices.

# MoCha

---

- *MoCha* is a middleware with the aim of providing applications with directed, anonymous, **distributed** and **mobile** communication channels.



# easyMoCha (interface)

---

The interface of easyMoCha provides 5 classes to the user: MoChaLocation, MobileChannel, ChannelEnd, SourceEnd and SinkEnd.

MoChaLocation:

Create an instance of this class to create a *logical* location in the network. EasyMoCha internally associates this location with a specific IP-address + port + virtual machine.

```
MoChaLocation loc = new MoChaLocation();
```

You need a location for both creation of channels and movement of channel-ends.

# easyMoCha (interface)

---

MobileChannel :

An instance of this class has two attributes (`ce1`, `ce2`) that refer to the channel-ends of a particular channel. Upon creation the user can specify the desired channel type and at which location the channel has to be created.

```
MobileChannel chan = new MobileChannel(location, type);
```

You need a location for both creation of channels and movement of channel-ends.

# easyMoCha (interface)

---

ChannelEnd:

The super-class of SourceEnd and SinkEnd. Useful when it doesn't matter what kind of channel-end you are dealing with, or you know the type for sure.

```
ce.move(locationCWI);
```

```
ce1.equals(ce2);
```

```
ce1.sameChannel(ce2);
```

```
ce1.write(data); // ce1 must be of type SourceEnd.
```

```
ce2.read(); // ce2 must be of type SinkEnd.
```

# easyMoCha (interface)

---

SourceEnd:

The source-end of a channel. Has no constructor thus can only be obtained from creating an instance of `MobileChannel`. Major operations are:

```
write(serializable object);
```

```
writeInteger(Integer);
```

```
writeDouble(Double);
```

```
...
```

```
move(location);
```

```
equals(ChannelEnd);
```

```
equalsChannel(ChannelEnd);
```

```
getStatus(); // full, empty?
```

# easyMoCha (interface)

---

SinkEnd:

The sink-end of a channel. Has no constructor thus can only be obtain from creating an instance of `MobileChannel`. Major operations are:

```
read(serializable object);
```

```
take(serializable object);
```

```
takeInteger(Integer);
```

```
takeDouble(Double);
```

```
...
```

```
move(location);
```

```
equals(ChannelEnd);
```

```
equalsChannel(ChannelEnd);
```

```
getStatus(); // full, empty?
```

# (easy)MoCha Channel Types

---

- ✓ Synchronous
- ✓ Lossy synchronous
- ✓ Filter
- ✓ Synchronous drain
- ✓ Synchronous spout
- ✓ Asynchronous unbounded FIFO
- ✓ Asynchronous bounded FIFO (FIFO n)
- ✓ Asynchronous drain
- ✓ Asynchronous spout
- ✓ Drain
- ✓ Spout

# Producer – Consumer Demo

---

Producer - Consumer Demo.

# Conclusion & Future Work

---

MoCha is an middleware for network systems based on mobile channels. It provides:

- *A clear separation of concerns* between the coordination and the computational aspects of a system ->
- *transparent exogenous coordination.*
- support of *dynamic* distributed systems, due to mobile channel-ends.