# Ibis
## Communication Library and Programming Models

**Jason Maassen**
jason@cs.vu.nl
**Rob van Nieuwpoort**
rob@cs.vu.nl

**Friday 27 April 2007**
**Vrije Universiteit Amsterdam**

## Overview

- Philosophy / design / implementation
  - Why Ibis ?
  - Design
  - Communication Library Model
  - Cool features
  - Programming Models
    - Satin
    - MPJ
    - GMI

## *We are interested in…*

- Parallel applications on “the Grid”
  - Single site runs
    - Grid == big collection of clusters
      - Only communicate within cluster
      - Use fast local network (Myrinet/Infiniband/…)
  - Multi site runs
    - Grid == big processor pool
      - Communicate between clusters
      - Use regular network & internet

## So why Ibis?

- Ideally, grid computing should be “fire and forget”
  - Develop application locally
  - Submit to some grid (using GAT)
    - Finds some suitable site(s)
    - Transfers your application and data to the sites, and runs it.
    - Returns the result

## ***Problems***

- Lots of problems
  - Resource selection
  - Data transfer
  - Security and authentication
  - ...
  - Heterogeneity
- GAT and SAGA solve part of this

## *Problems*

- Grids are heterogeneous:
  - Intel / PowerPC / Mips / Arm / ...
  - Windows / Linux / Unix / OSX / ...
  - Different OS/library/tool versions
  - Different networks
- Compiled (C/MPI) apps. huge pain:
  - Need executable for every combination of CPU/network/OS/libraries etc.
  - Hard to connect sites together.
  - Makes 'fire & forget' runs really hard...

vrije Universiteit

## *Solution (partly)*

- So, we use Java instead C or Fortran
  - No recompilation required
  - Runs (almost) anywhere
    - Doesn't work on supercomputers such as Hitachi SR8000, IBM BlueGene ...
    - ... but most sites have clusters anyway!
  - Acceptable performance
- But ... only part of the solution!

## *How about 'portable' communication?*

- Class libraries are portable, but ...
  - Sockets are too low-level
  - RMI model/performance is limited
- Most parallel libraries not suited ...
  - Example: MPIJava requires native code
    - needs recompilation
    - only supports static (fixed size) runs
    - multi-cluster MPI is a bit hard
    - not all applications are SPMD

## *Ibis*

- Solution: Ibis Communication Library!
  - A “run-anywhere” communication library
  - Just send it along with your application!

- Plus: flexible communication models
  - Malleability & Fault-Tolerance
    - Change number of machines during the run
  - More than just unicast communication
    - More about this later

## *Ibis 2.0*

- In this tutorial we describe Ibis 2.0
  - Ideas are the same, but interface is cleaned-up
  - Ibis 1.x interface stable for 4 years
  - Ibis 2.0 inteface changed according to the lessons we learned from the previous versions

## *Ibis*

- Portability vs. performance
  - On a single site run you often want to use the fast local network
  - Ibis allows specialized implementation
    - Designed for Myrinet, Infiniband, etc.
    - Usually use native code
    - Installed in advance
      - not portable
      - cannot be shipped with application

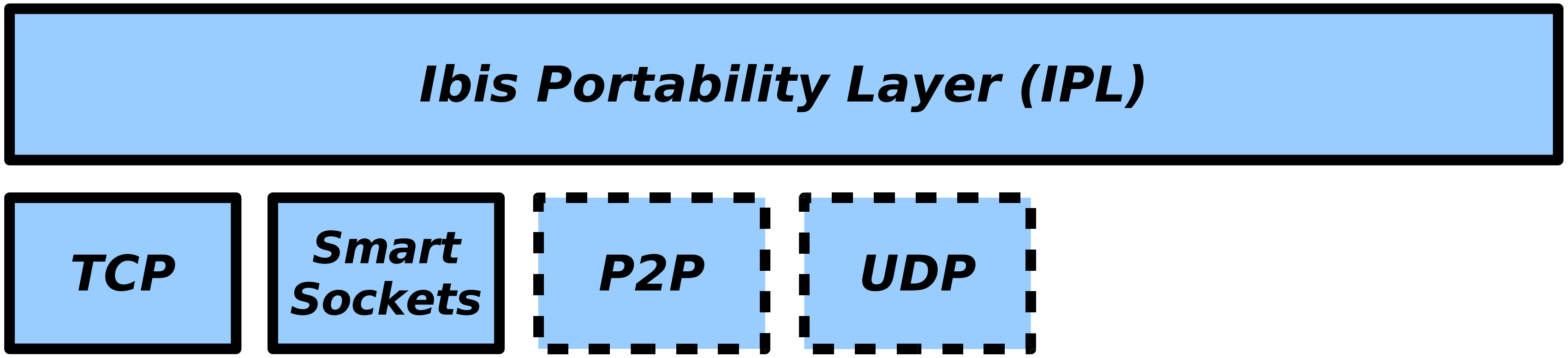
## *Ibis*

- As a result, there may be multiple Ibis' available on a site
  - Automatically choose 'best' at startup
  - Based on requirements specified by
    - Application & user (using properties)
- Not every appl. needs all features
  - Pick one at startup that suits your needs....

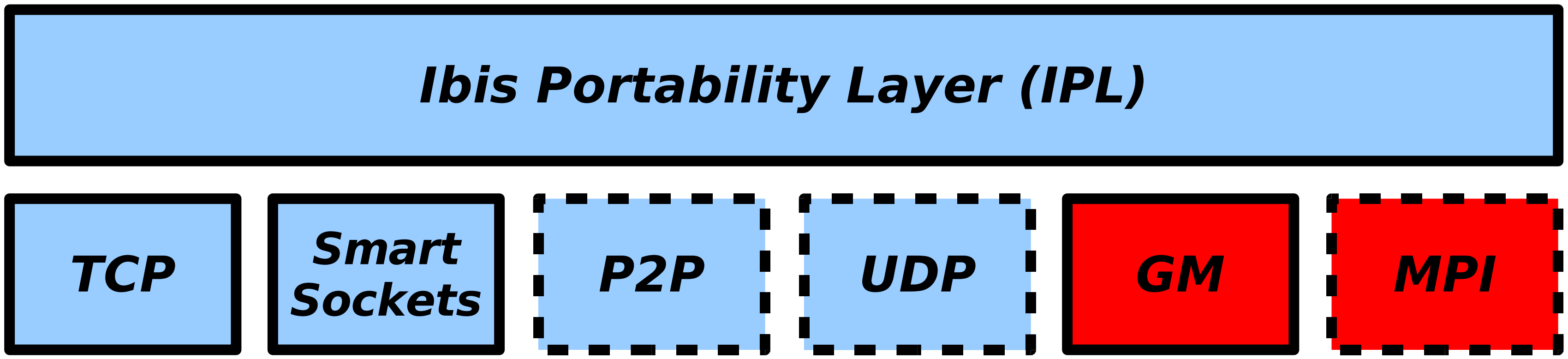
# Ibis Design

***Ibis Portability Layer (IPL)***

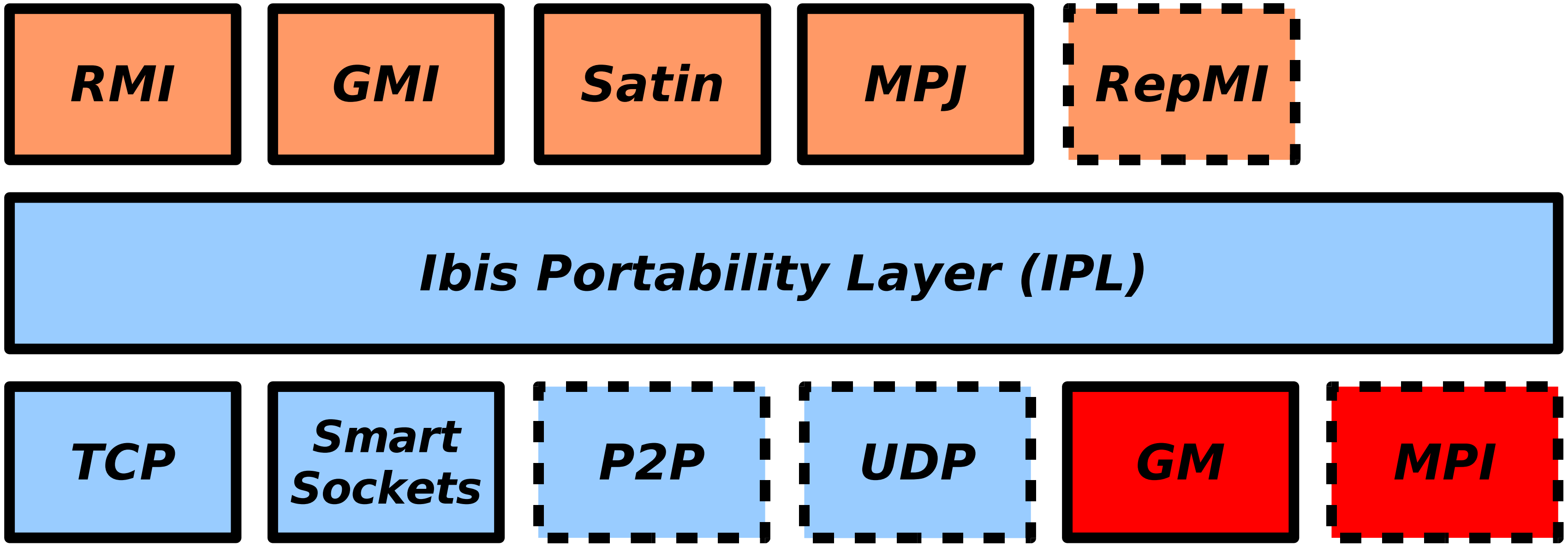
# Ibis Design



# Ibis Design

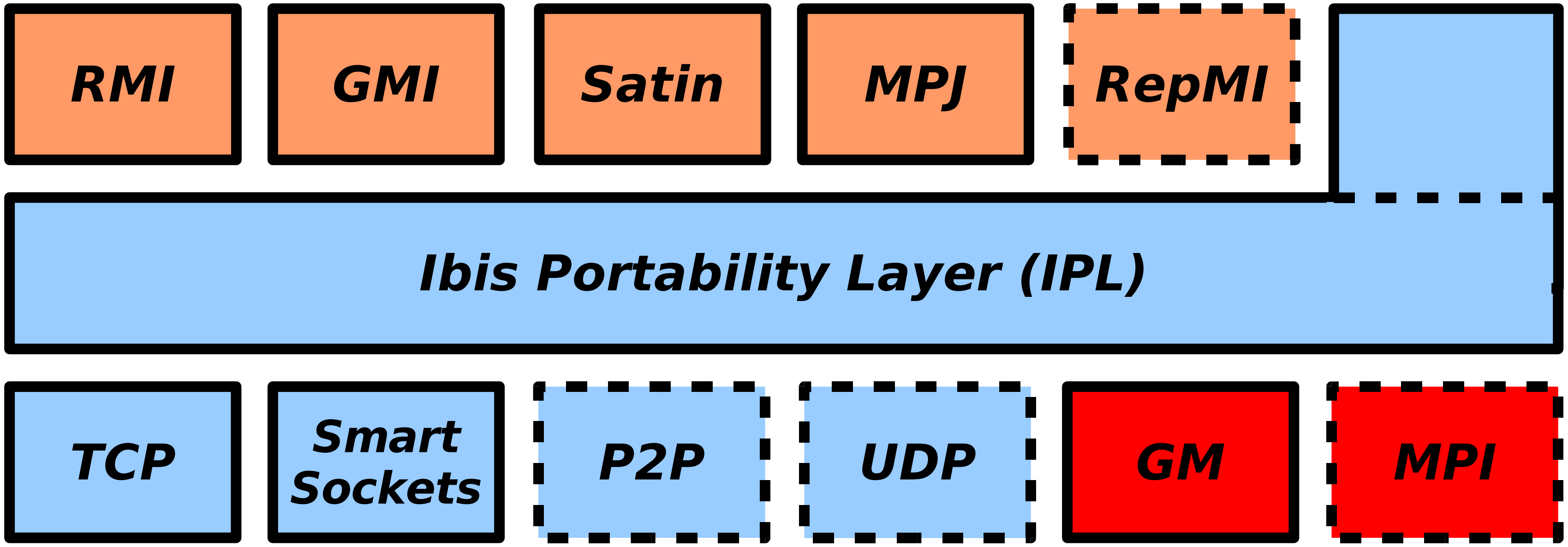


# Ibis Design

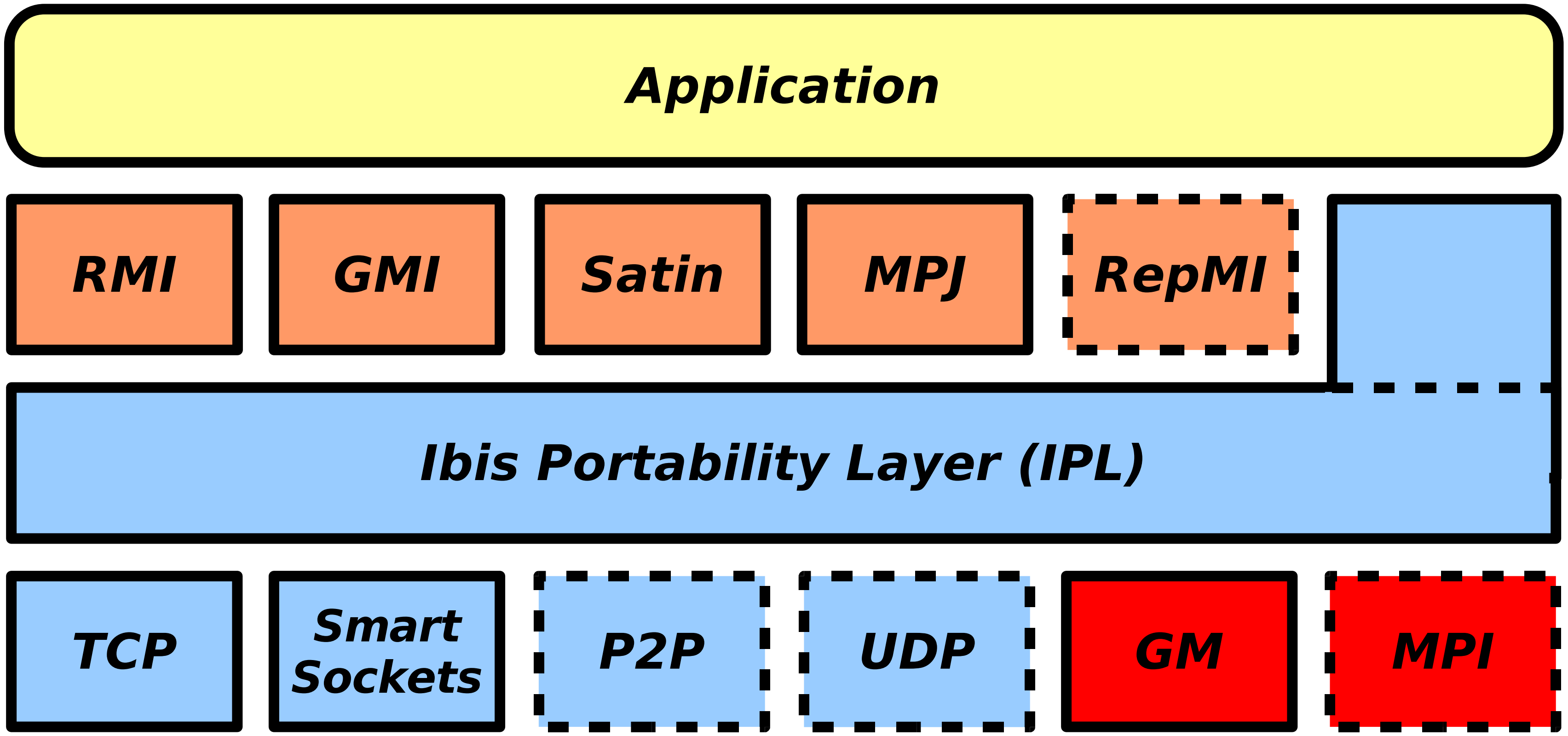


RMI
GMI
Satin
MPJ
RepMI
Ibis Portability Layer (IPL)
TCP
Smart Sockets
P2P
UDP
GM
MPI

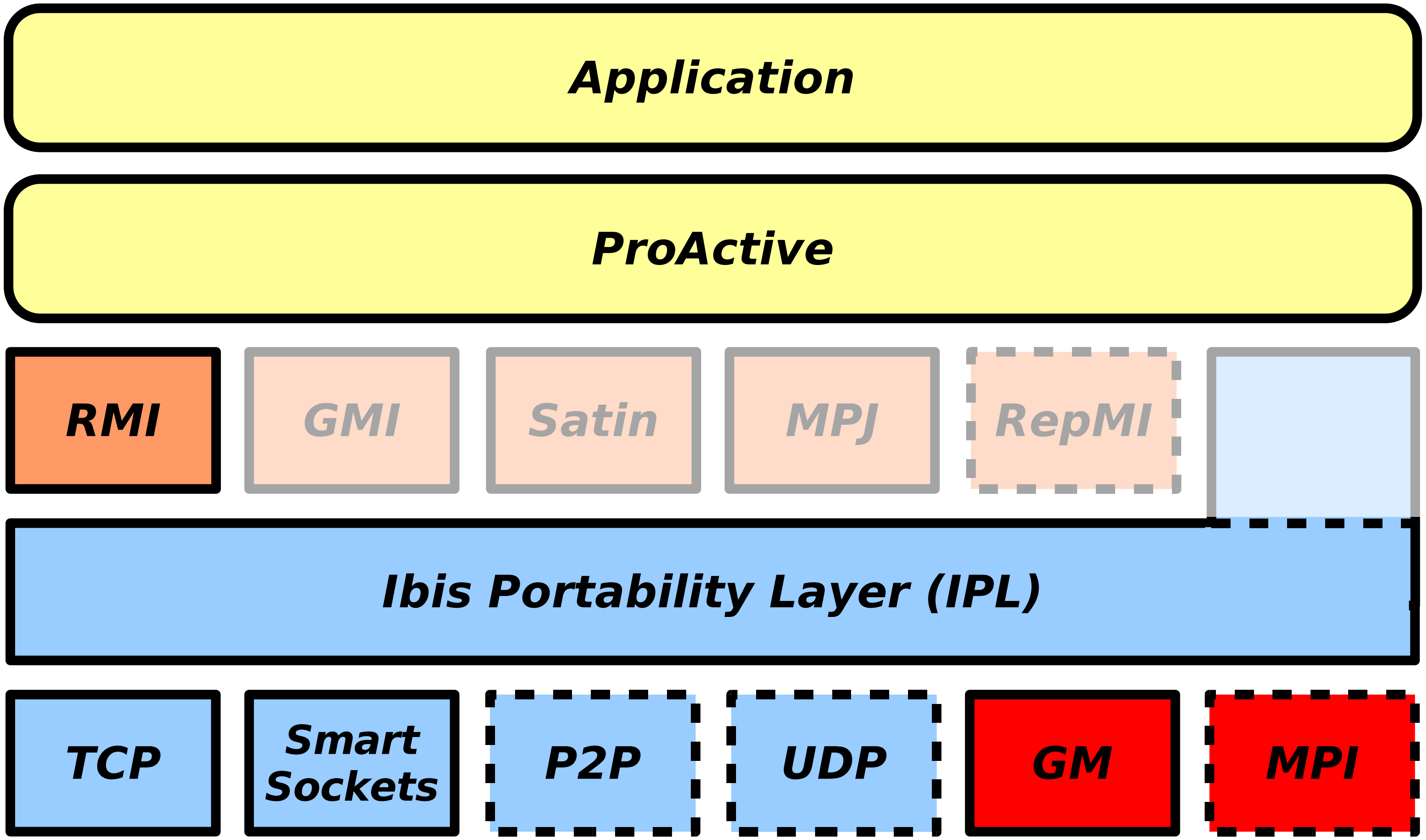
# Ibis Design


RMI
GMI
Satin
MPJ
RepMI
Ibis Portability Layer (IPL)
TCP
Smart Sockets
P2P
UDP
GM
MPI

# Ibis Design


Application
RMI
GMI
Satin
MPJ
RepMI
Ibis Portability Layer (IPL)
TCP
Smart Sockets
P2P
UDP
GM
MPI

# Ibis Design


Application
ProActive
RMI
GMI
Satin
MPJ
RepMI
Ibis Portability Layer (IPL)
TCP
Smart Sockets
P2P
UDP
GM
MPI

# *Ibis Portability Layer*

- Basic Ibis programming interface
  - Reasonably simple (5 classes, 16 interfaces, 10 exceptions)
- Contains methods for
  - Loading an Ibis
  - Malleability/Membership
    - adding & removing machines
  - Connection handling
  - Communication primitives (low-level)

## *Basic Idea*

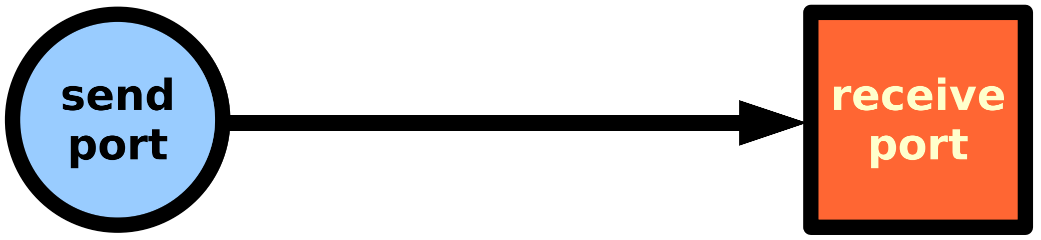
- Abstract away from implementation
  - use abstract addressing scheme
    - hides real network addressing
  - use abstract communication primitives
    - hides real communication primitives
- Results in more portable applications
  - same application runs on sockets and MPI without changing any code

## *IbisIdentifiers*

- In a parallel/distributed application
  - Each process has an Ibis instance
  - Each instance has an IbisIdentifier
- IbisIdentifier:
  - Uniquely identifies an Ibis instance
  - Abstracts away from the implementation
    - e.g. hostnames, IP addresses, MPI-ranks, etc.
  - Makes your application a bit more portable

## Communication

- 'Low-level' communication model
- Unidirectional pipes
- Two end points
- Connection oriented (allows streaming)

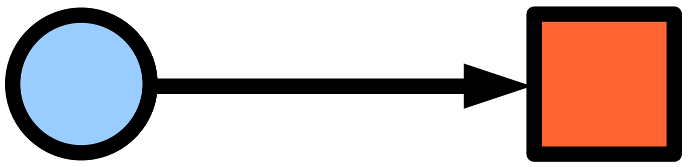


## *Send & receive ports*

- Can be connected in arbitrary ways

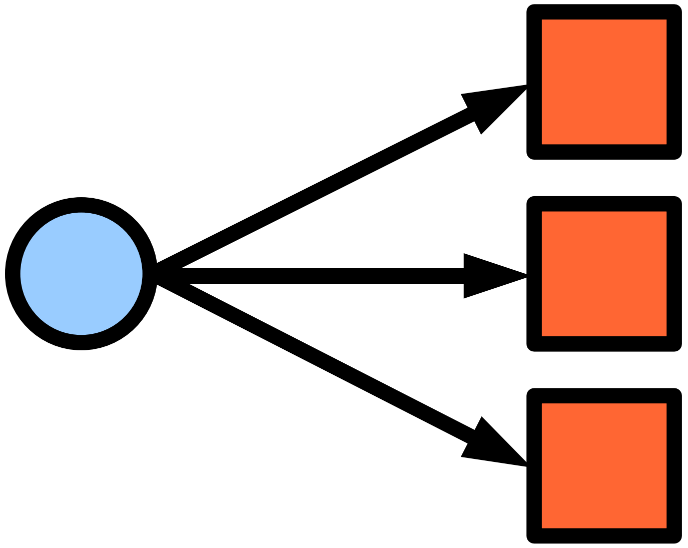
## *Send & receive ports*

- Can be connected in arbitrary ways
- One to one (unicast) ...



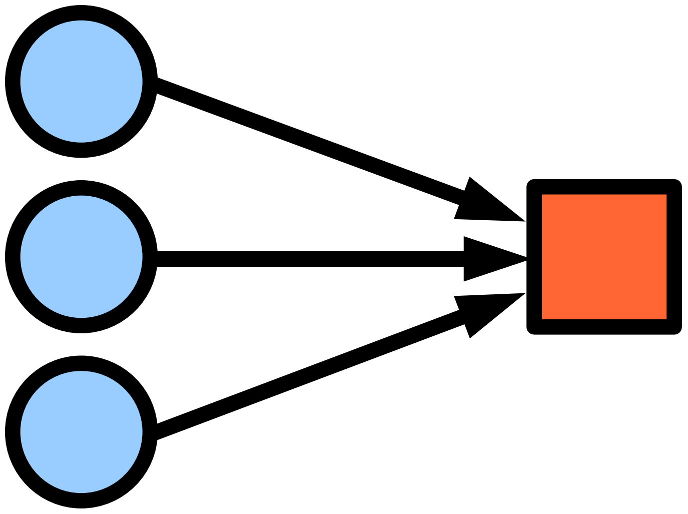
## *Send & receive ports*

- Can be connected in arbitrary ways
- ... one to many (multicast) ...



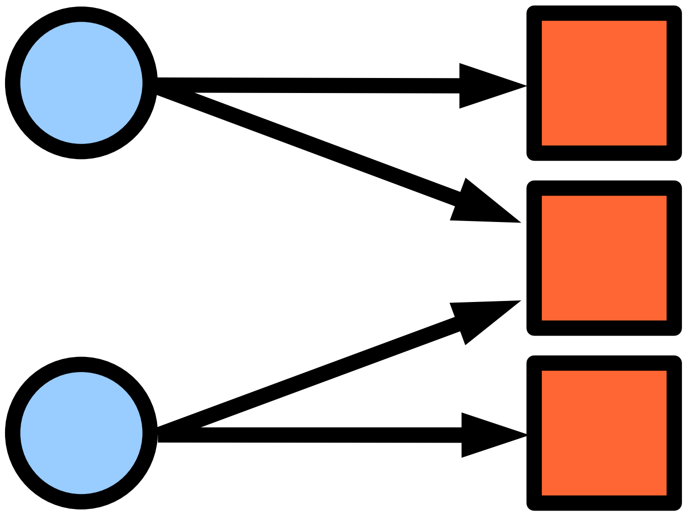
# Send & receive ports

- Can be connected in arbitrary ways
- ... many to one ...



# *Send & receive ports*

- Can be connected in arbitrary ways
- ... or some combination!

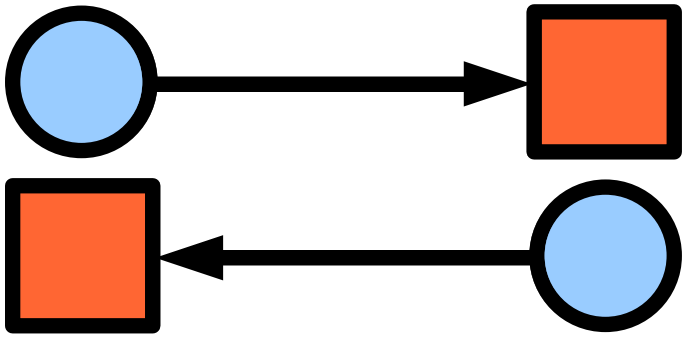


## *Send & receive ports*

- Advantages:
  - Very simple & abstract model
  - Easy to implement using TCP/UDP/MPI/etc.
  - Allows multicast, many-to-one, etc.
    - Useful for parallel programs
  - Allows efficient implementation
    - Can be implemented using efficient low-level primitives (i.e., mpi-broadcast)
    - Other models do prevent this (e.g., RMI)

## *Send & receive ports*

- Disadvantage:
  - Simplicity may cause some overhead...
  - Example: need two pairs for RPC / RMI

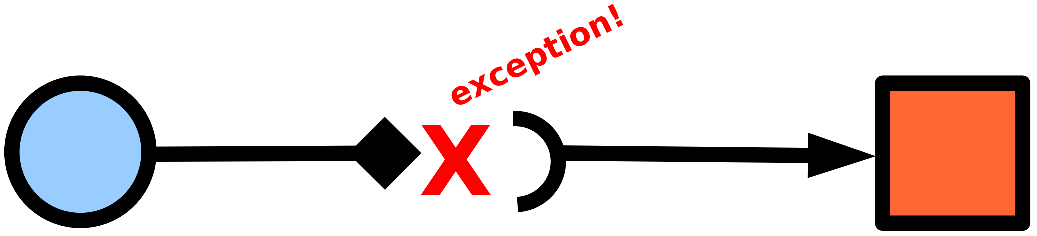


## *Port Types*

- All ports have a <u>type</u>
  - Consists of a set of required capabilities:
    - Connection patterns
      - Unicast, many-to-one, one-to-many, many-to-many.
    - Communication properties:
      - Fifo ordering, numbering, reliability.
    - Serialization properties:
      - bytes, data, object
    - Message delivery:
      - Explicit receipt, automatic upcalls, polling

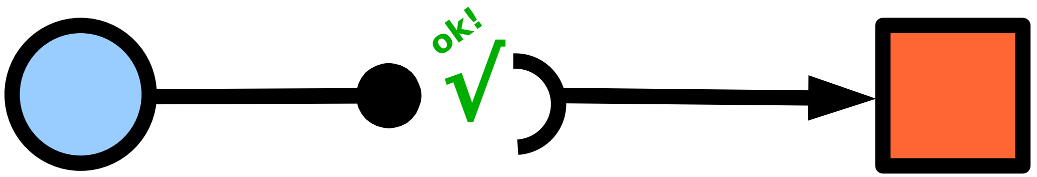
# Port Types

- Defined at runtime
  - Specify set of capabilities
- Types must match when connecting!



## Port Types

- Defined at runtime
  - Specify set of capabilities
- Types must match when connecting!



## Port Types

- Forces programmer to specify how each communication channel is used
  - Prevents bugs
    - Exception when contract is breached
  - Allows efficient impl. to be selected
    - Unicast only ?
    - Bytes only ?
    - Can save a lot complexity!

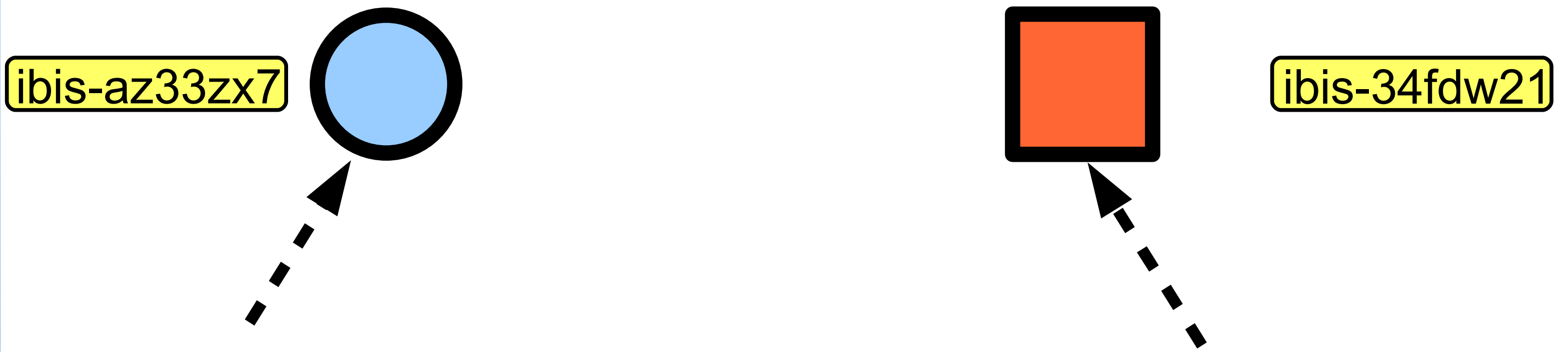
# Connection setup

- Two options:
  - 1) Using a IbisIdentifier and a name
    - Name specifies the receiveport
      - Unique per Ibis instance
    - Human-readable (usually)
  - 2) Using a ReceivePortIdentifier
    - Uniquely identifies a receiveport
    - Created when ReceivePort is created
    - Can be passed around between Ibis instances.

# Connection setup (1)

ibis-az33zx7
ibis-34fdw21
Create Ibis
Create Ibis

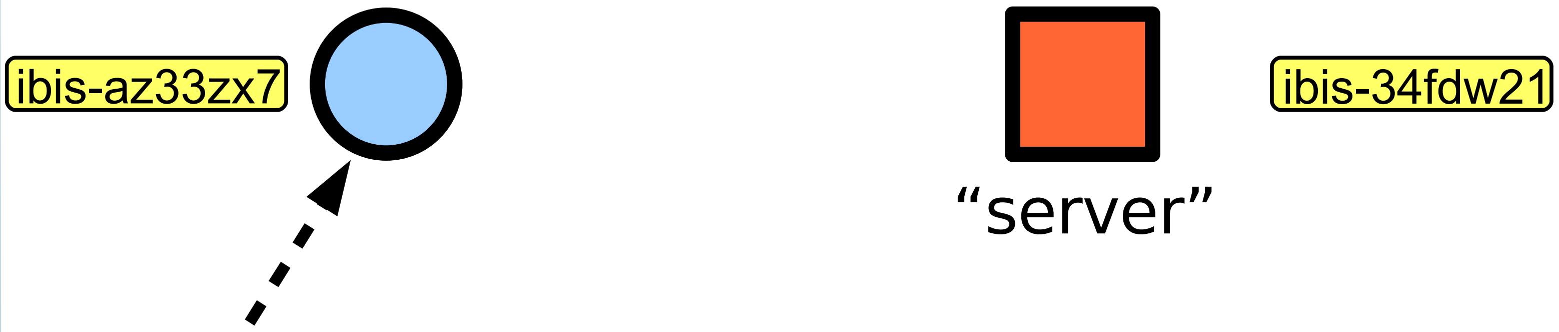
# Connection setup (1)



Create SendPort

Create ReceivePort
“server”

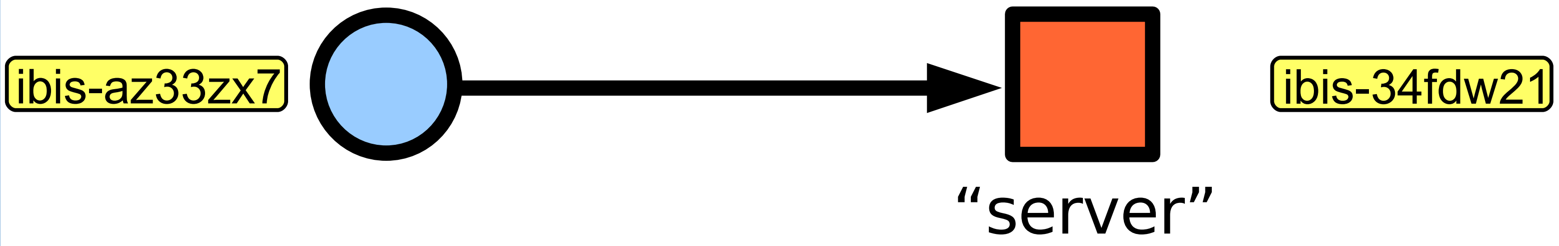
# Connection setup (1)



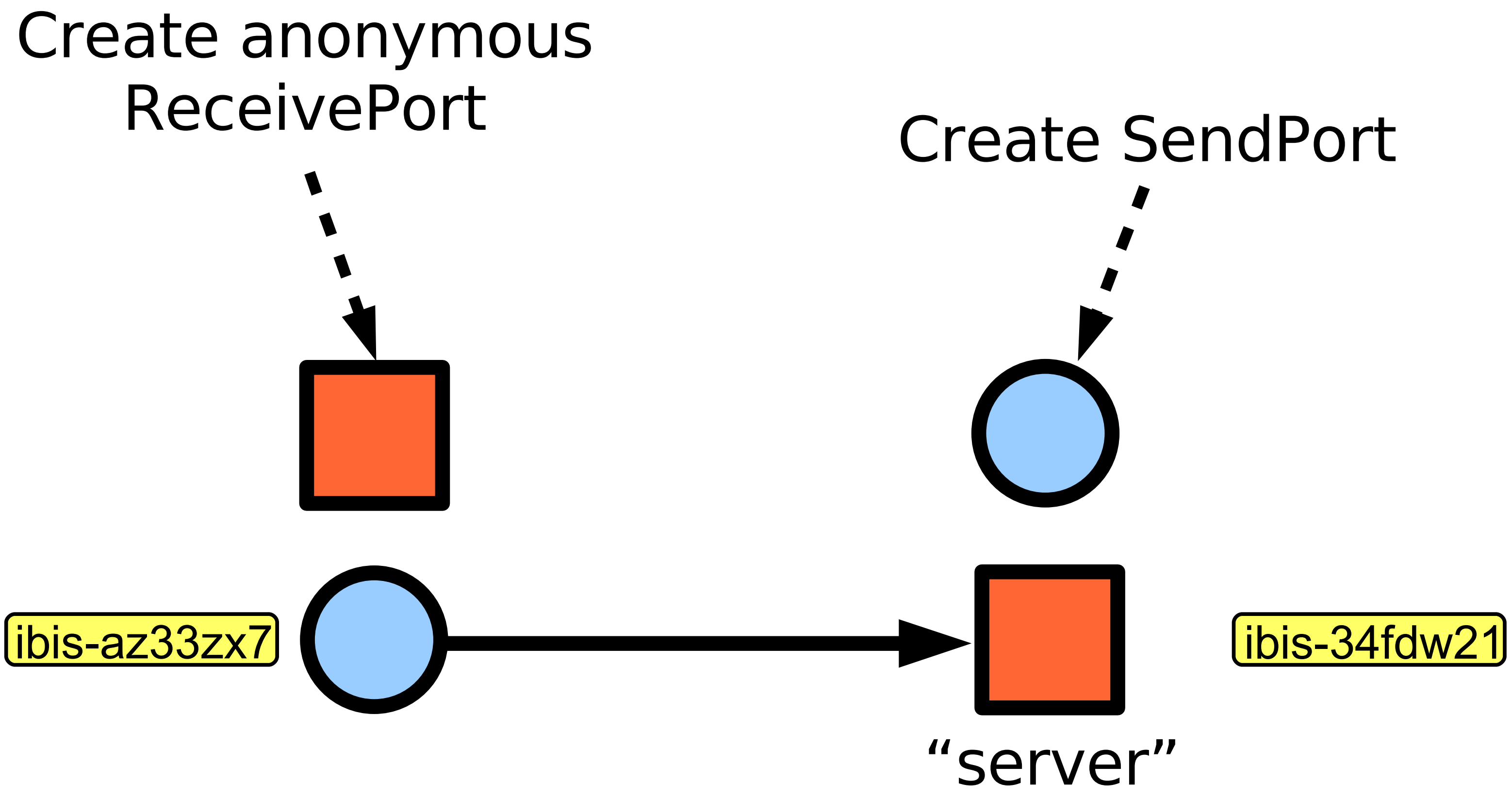
connect( ibis-34fdw21, "server")

(How do you get this ? Explained later!)

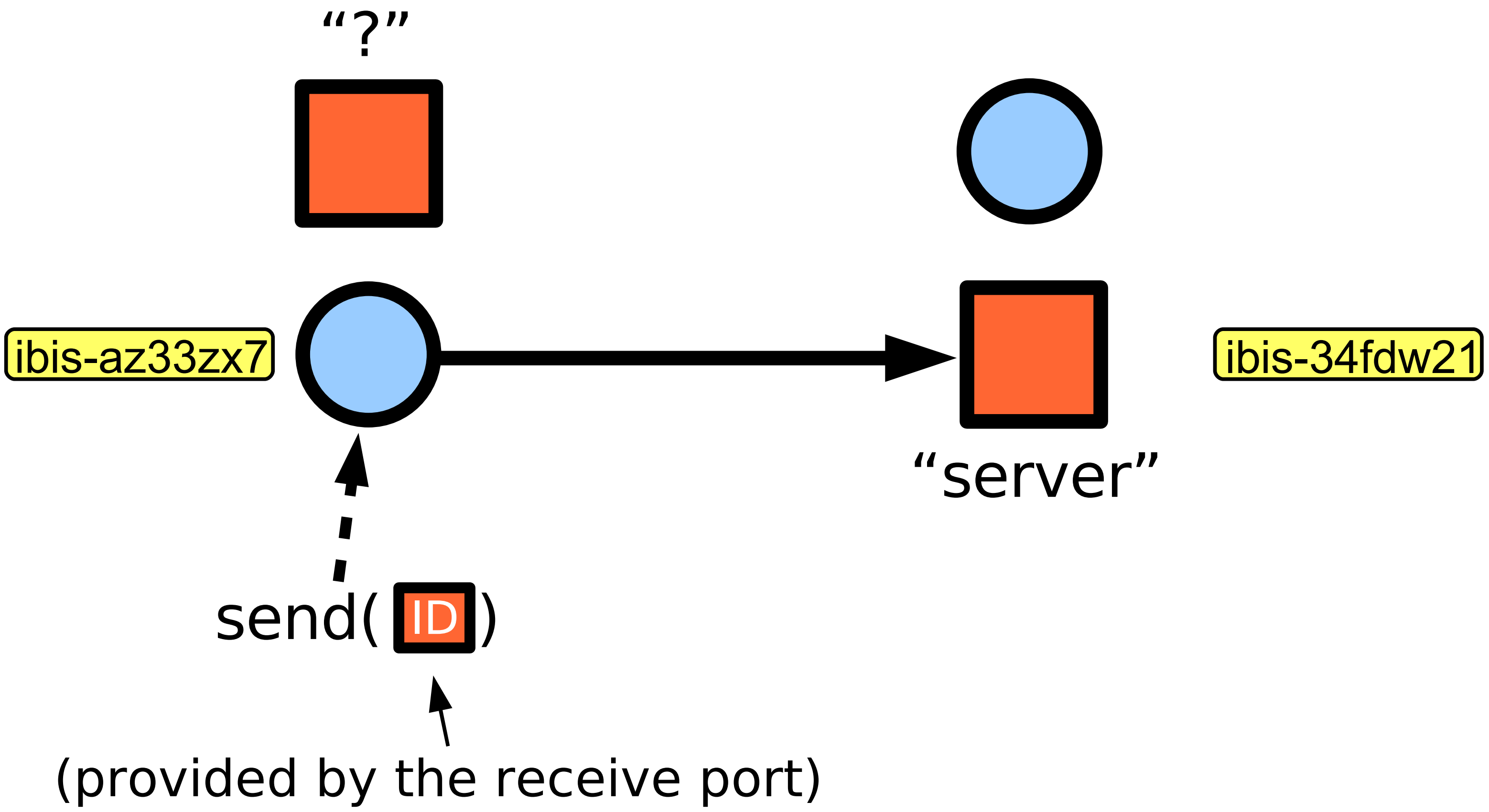
# Connection setup (1)



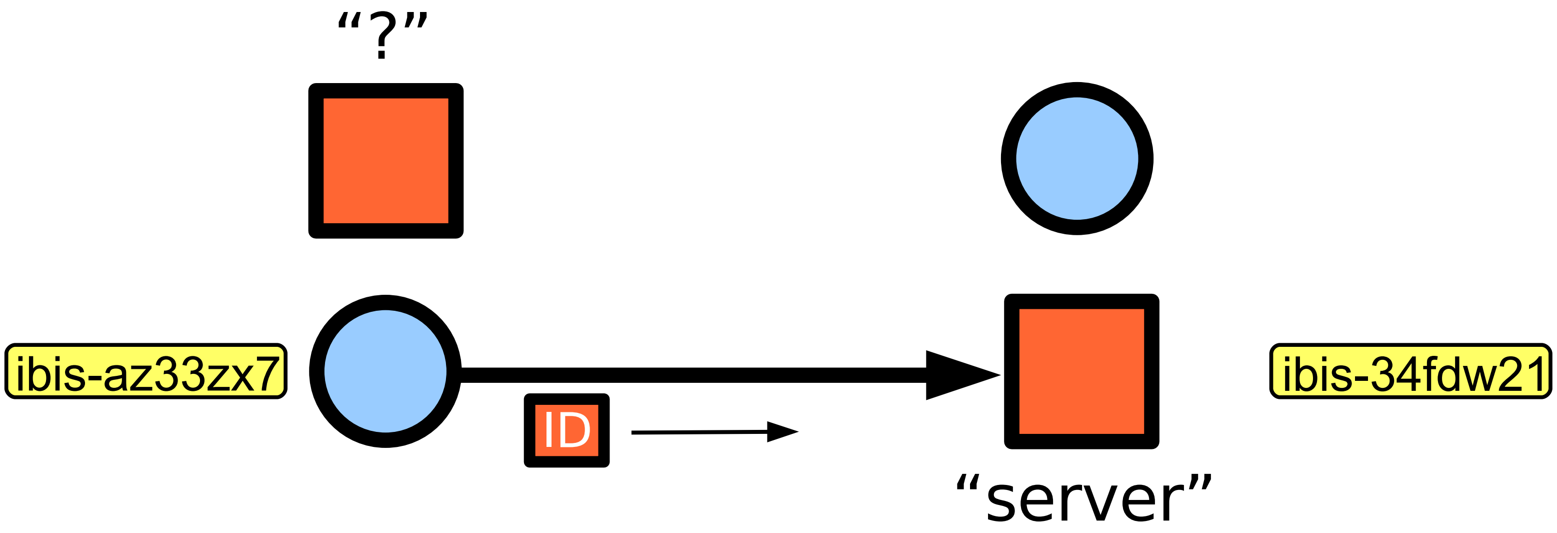
# Connection setup (2)



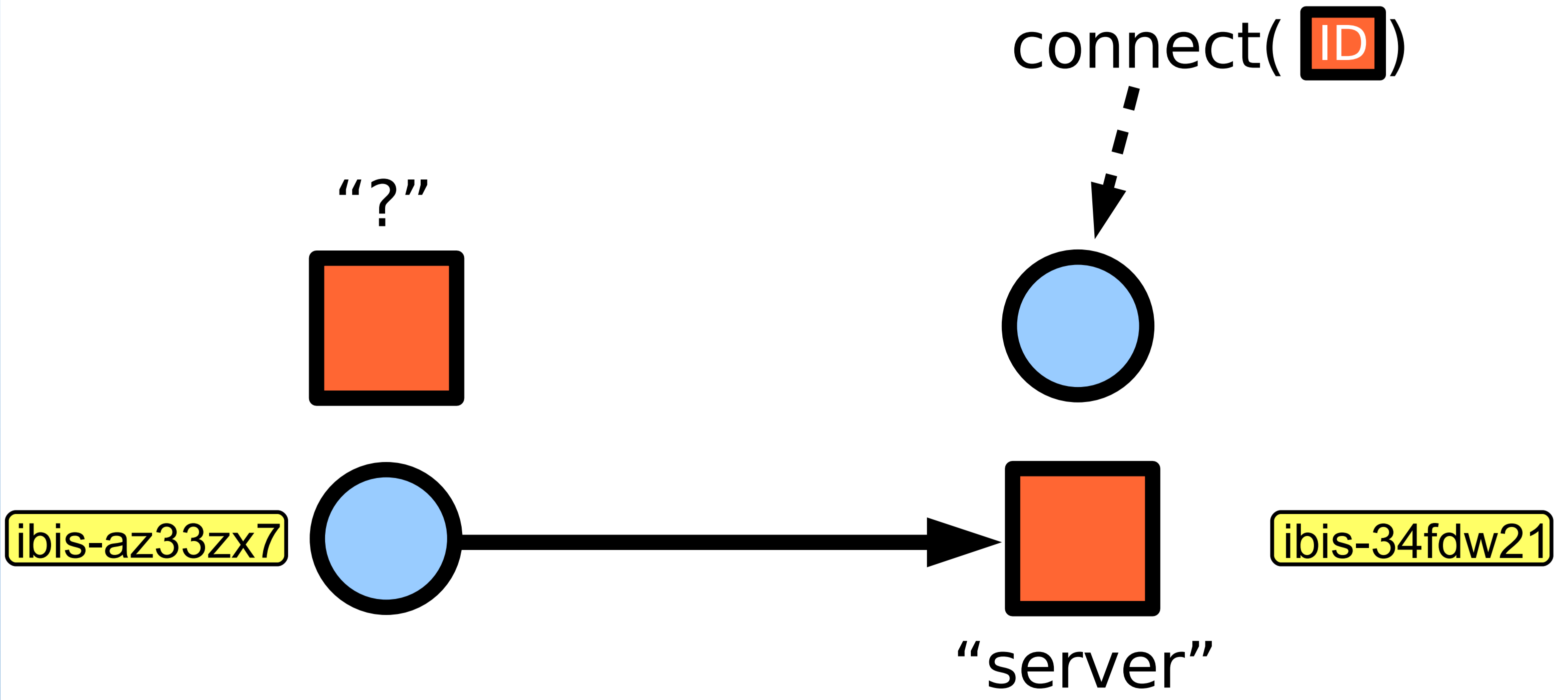
# Connection setup (2)



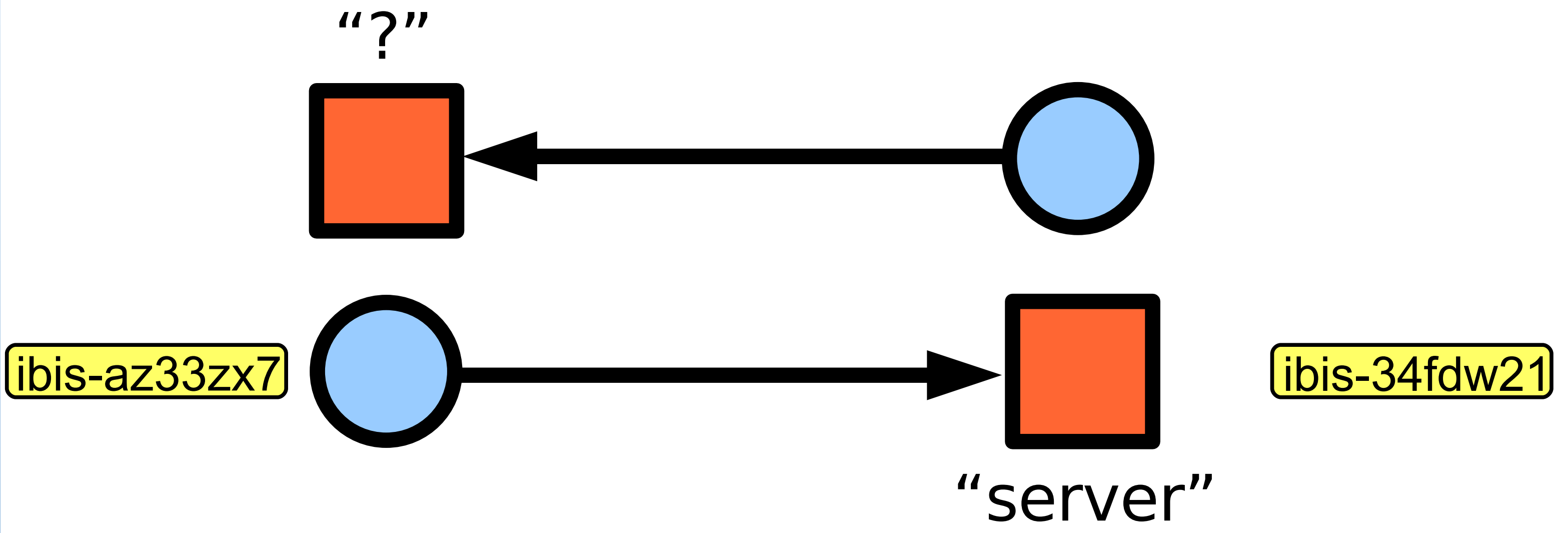
# Connection setup (2)



# Connection setup (2)



# Connection setup (2)

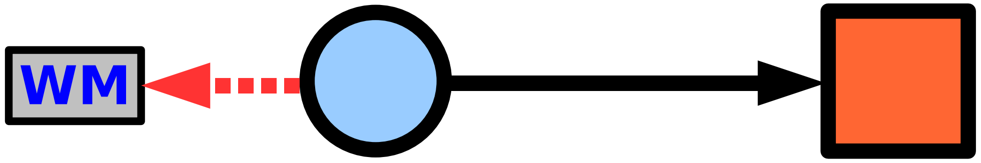


## *Messages*

- Ports communicate using 'messages'
- Contain read or write methods for
  - Primitive types (byte, int, ...)
  - Object
  - Arrays slices (partial write / read in place)
- Unlimited message size

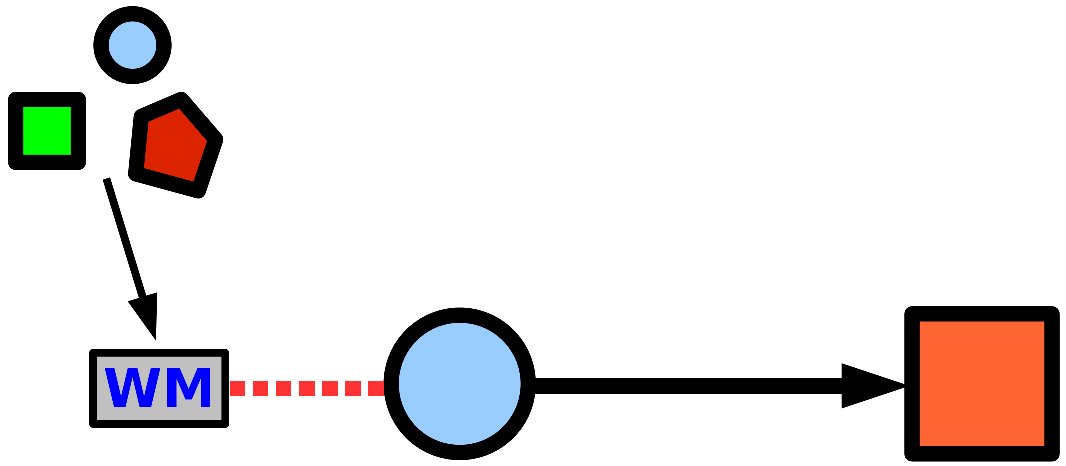
## Messages

- Get WriteMessage from SendPort



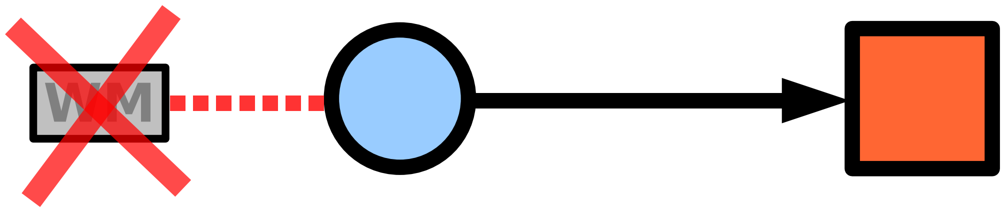
# Messages

- Write data into WriteMessage



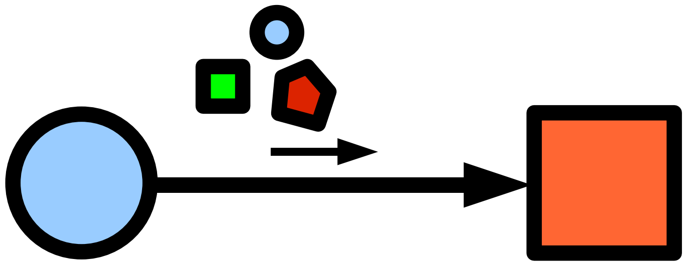
# *Messages*

- Finish the WriteMessage



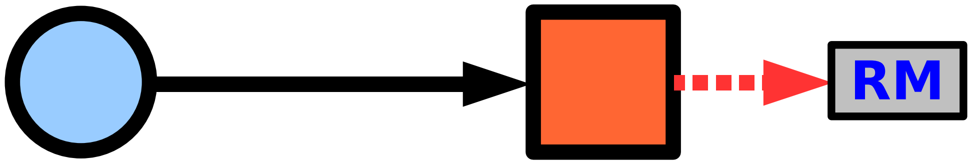
## *Messages*

- Data is send to ReceivePort



# Messages

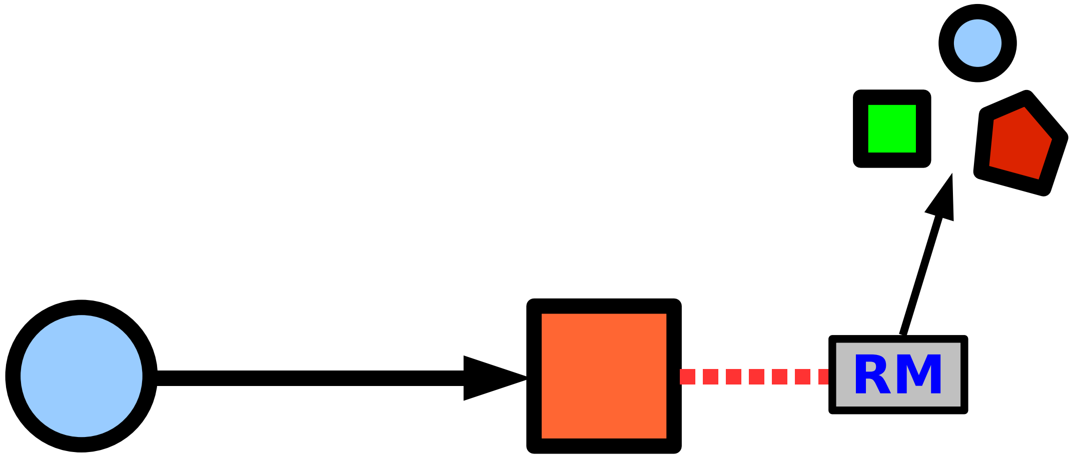
- ReceivePort produces ReadMessage
  - Explicit receive or callback (upcall)



vrije Universiteit

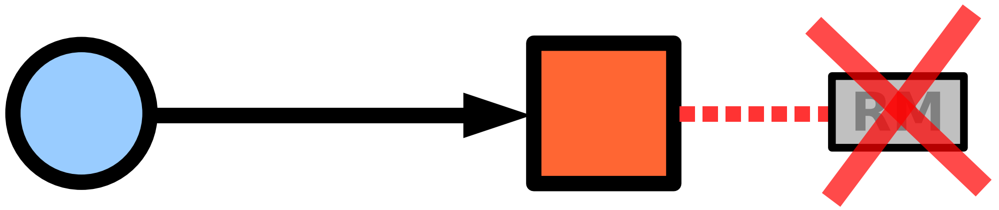
## Messages

- Read data from ReadMessage



## Messages

- Finish the ReadMessage

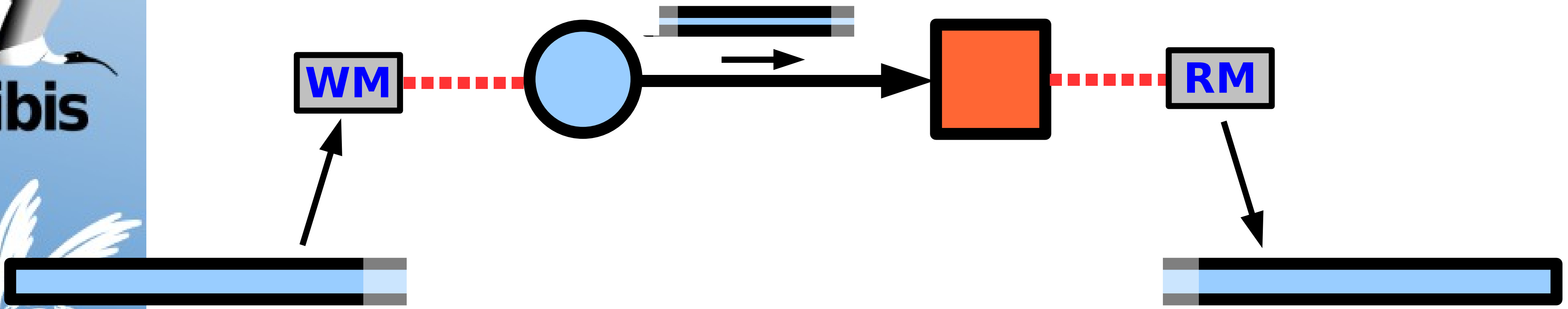


## Messages

- Done!

## Messages or streams ?

- Message size is unlimited
  - Data may be forwarded at any time
  - Both S. & R. messages alive at same time
  - There's streaming!



## Restrictions

- Must write and read data in same order
- A port can have only one message 'alive' at a time
- Messages are not thread safe (but ports are)
- The sender may block if the receiver is too slow (there may be flow-control)

## *Serialization*

- Ibis supports 3/4 types of serialization
  - Bytes (no serialization at all)
  - Data (only primitive types/arrays)
  - Object (graphs of object + previous)
    - Sun (standard Sun serialization)
    - Ibis (efficient Ibis serialization)

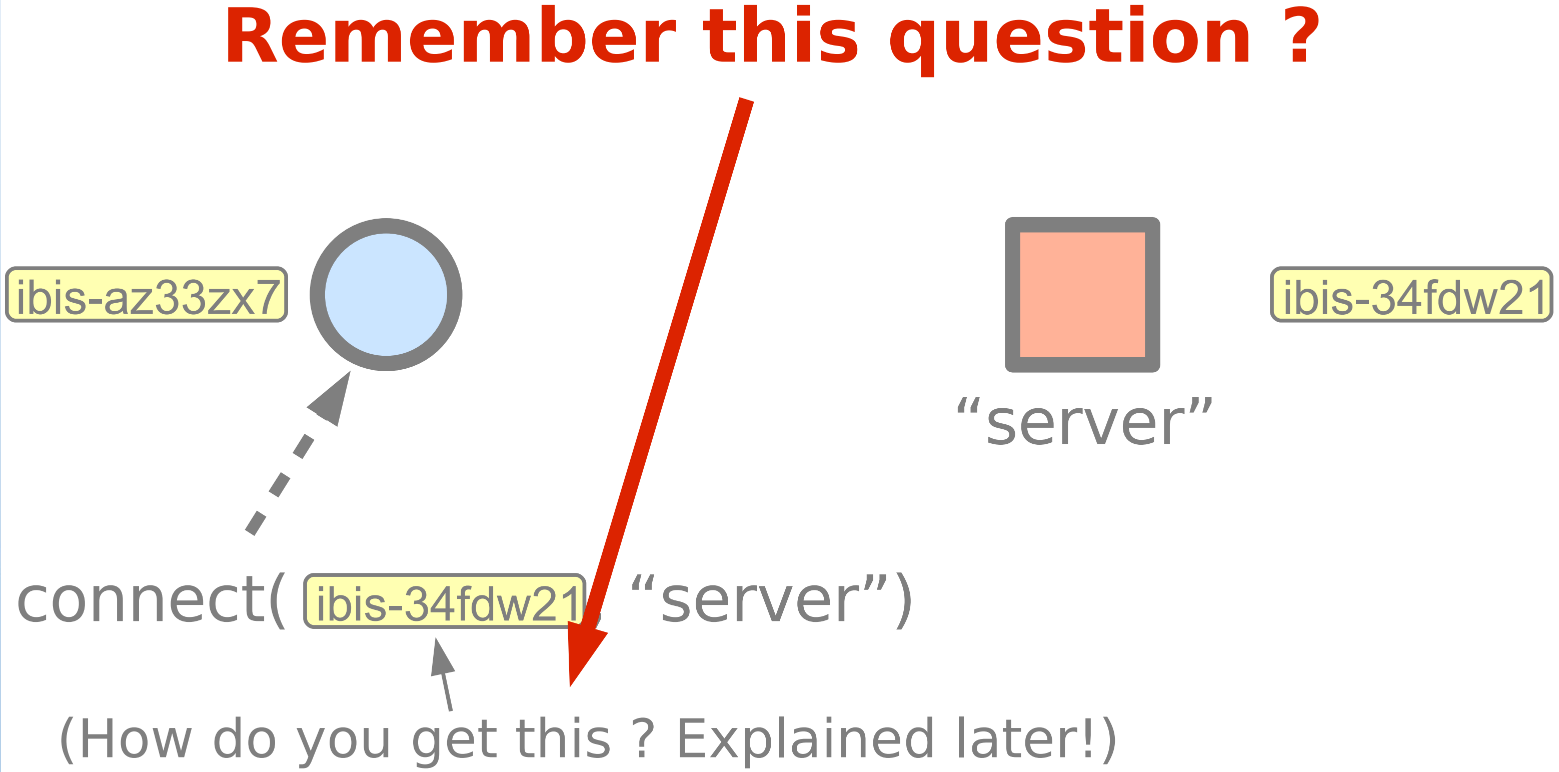
## *Ibis Serialization*

- Based on bytecode-rewriting
  - Adds serialization and deserialization code to serializable types
  - Prevents reflection overhead during (de-)serialization
  - Has fallback mechanism for non-rewritten classes
- Experimented with runtime rewriting

## Short Recap

- First create PortType
- PortType creates Send & ReceivePort
  - Type is checked when connecting
- Several ways to connect
  - Abstact addressing
- Use Messages to communicate
  - Allows streaming
  - 3/4 types of serialization

# Connection setup (1)


Remember this question ?
ibis-az33zx7
ibis-34fdw21
"server"
connect( ibis-34fdw21, "server")
(How do you get this ? Explained later!)

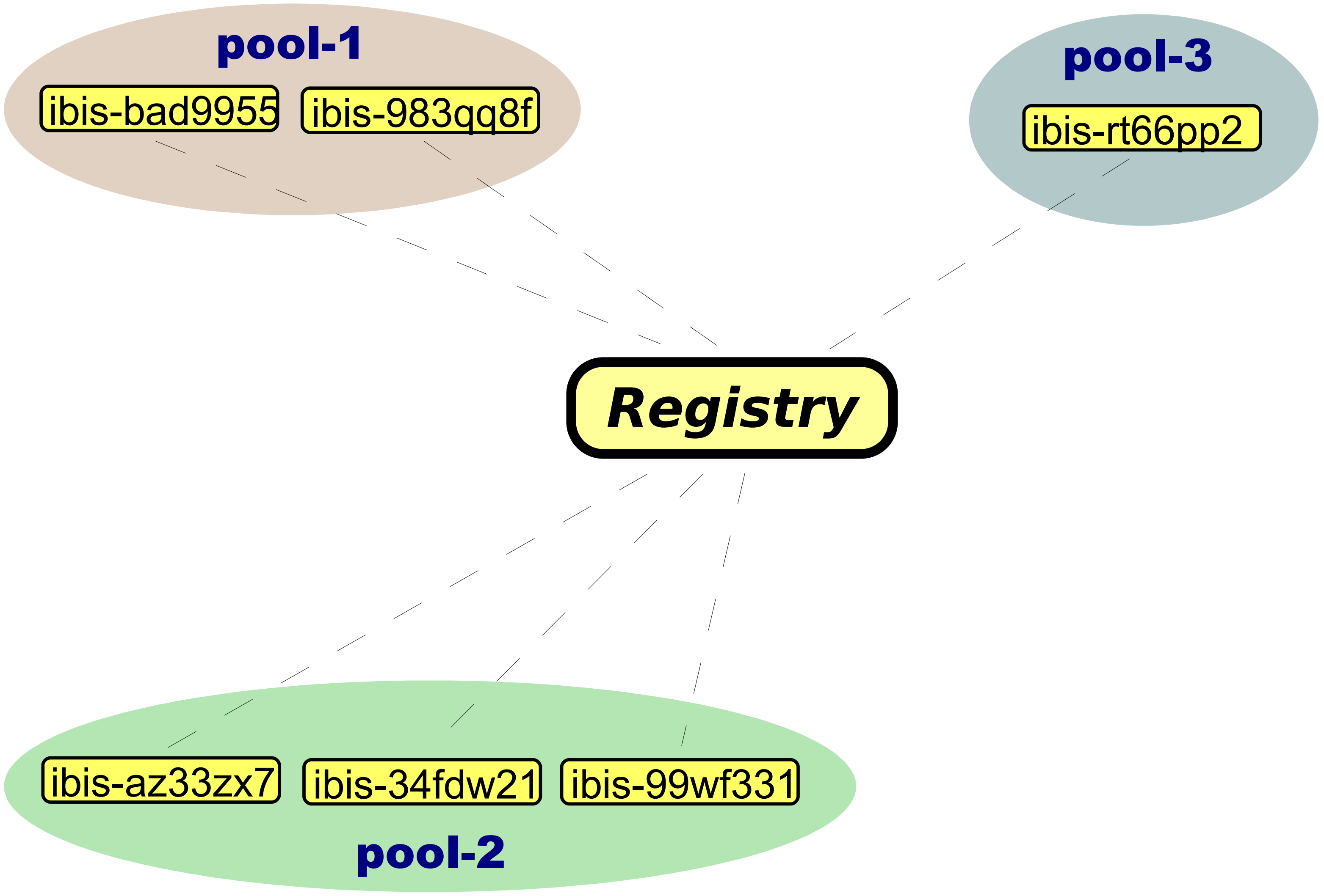
## *Pools & Malleability*

- Ibis instances are part of a pool
  - Either variable size or fixed (create-once)
    - Fixed used by 'legacy' MPI-type applications
- Membership information
  - Can subscribe to pool information
    - Updates when Ibis instances join or leave pool
  - Useful for determining who's participating
    - Also used for fault-tolerance

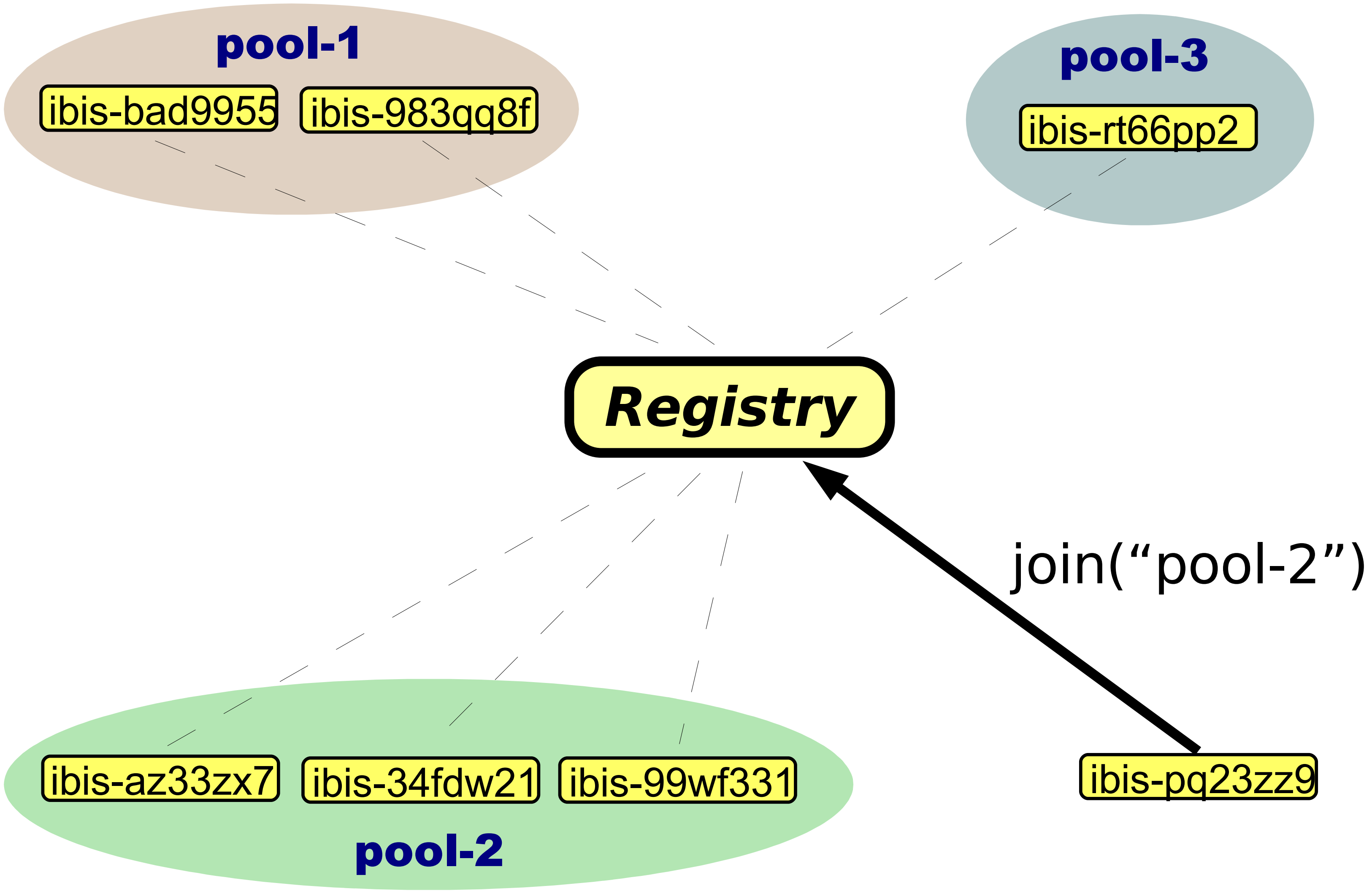
## ***Membership updates***

- Callbacks or explicit calls
  - If required, they can be delivered in the same order on all Ibis instances
- Uses external server (Registry)
  - Registry can handle multiple pools
  - No communication between pools

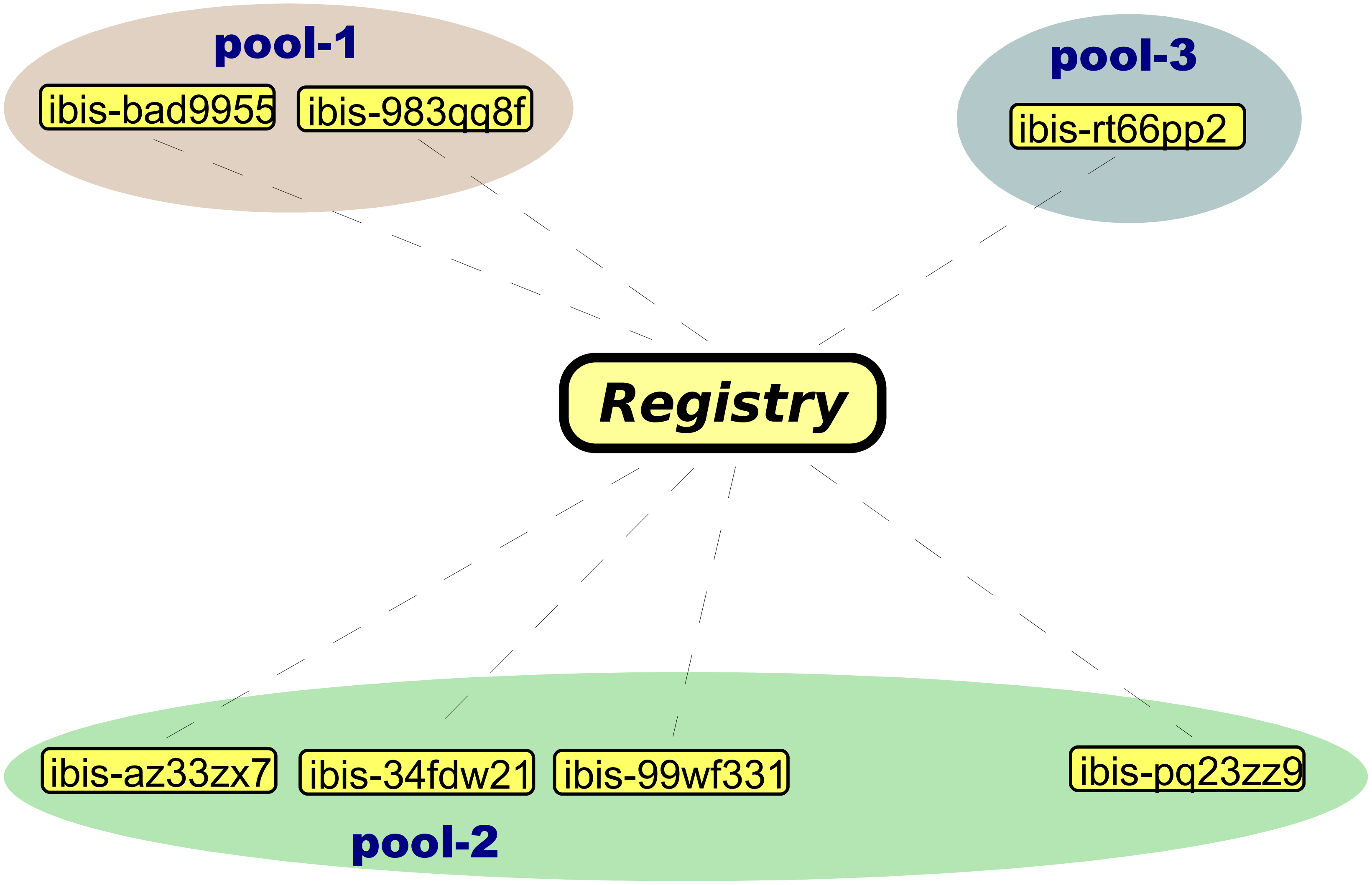
# Pools & Malleability



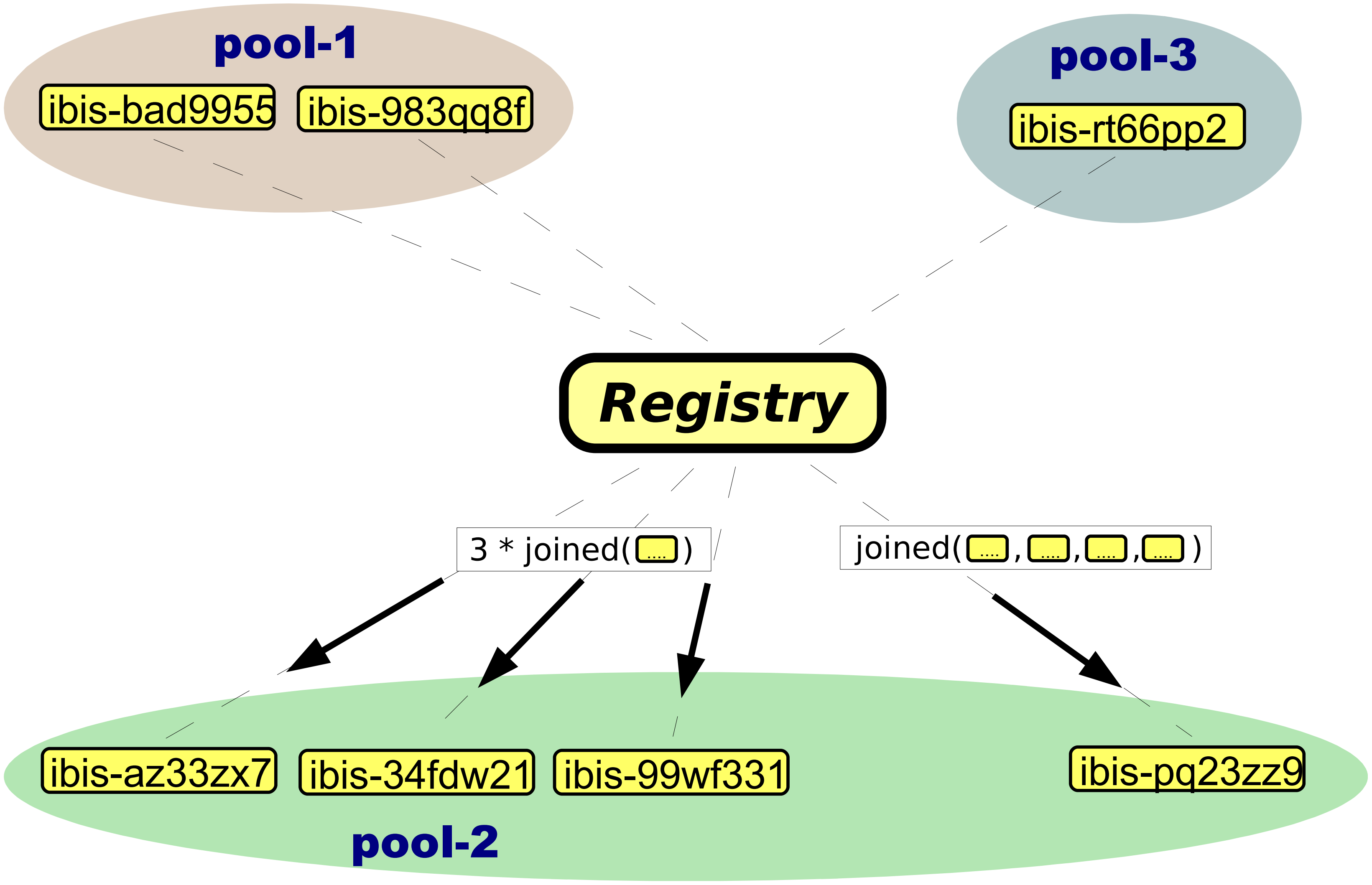
# Pools & Malleability



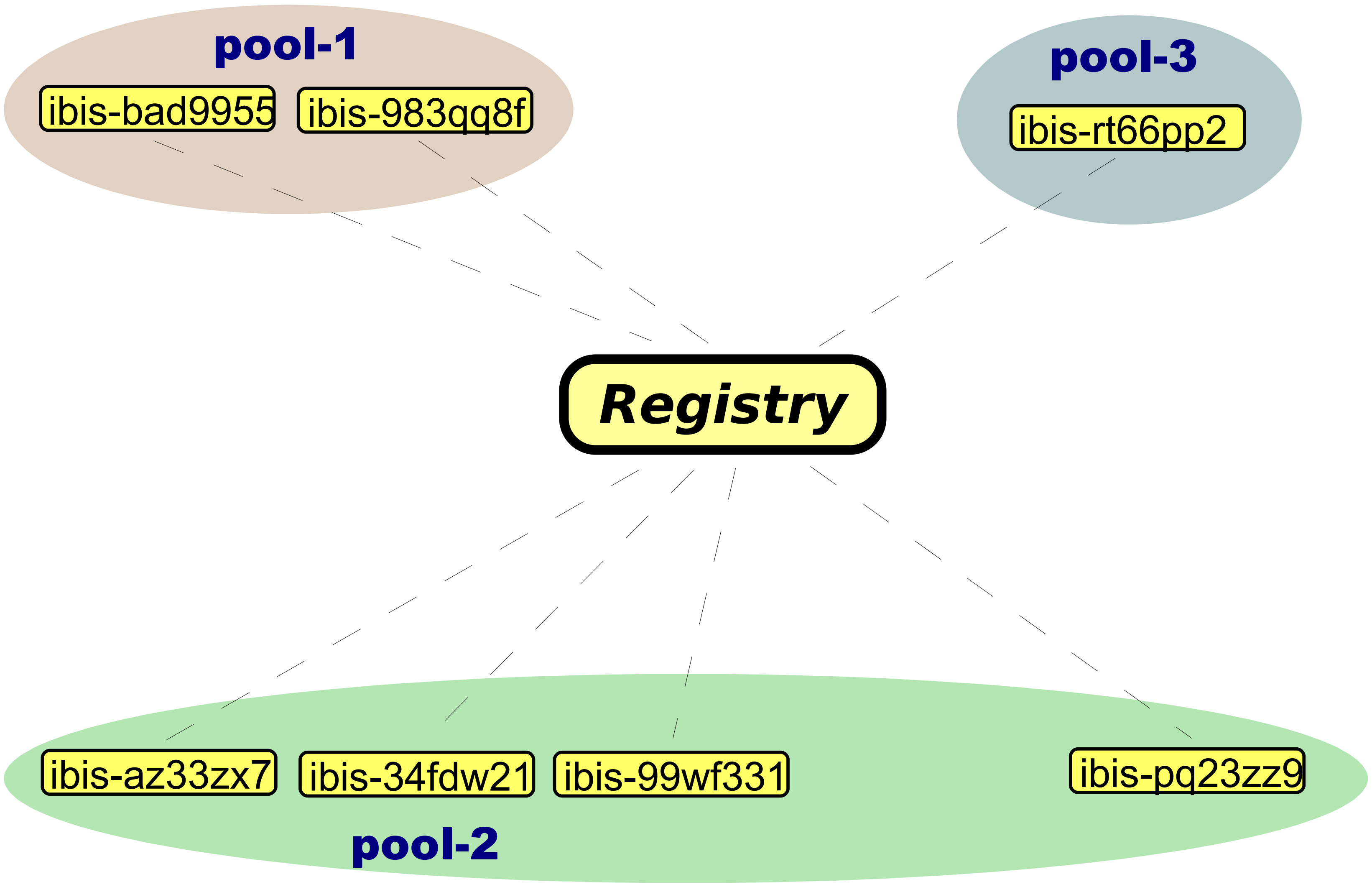
# Pools & Malleability



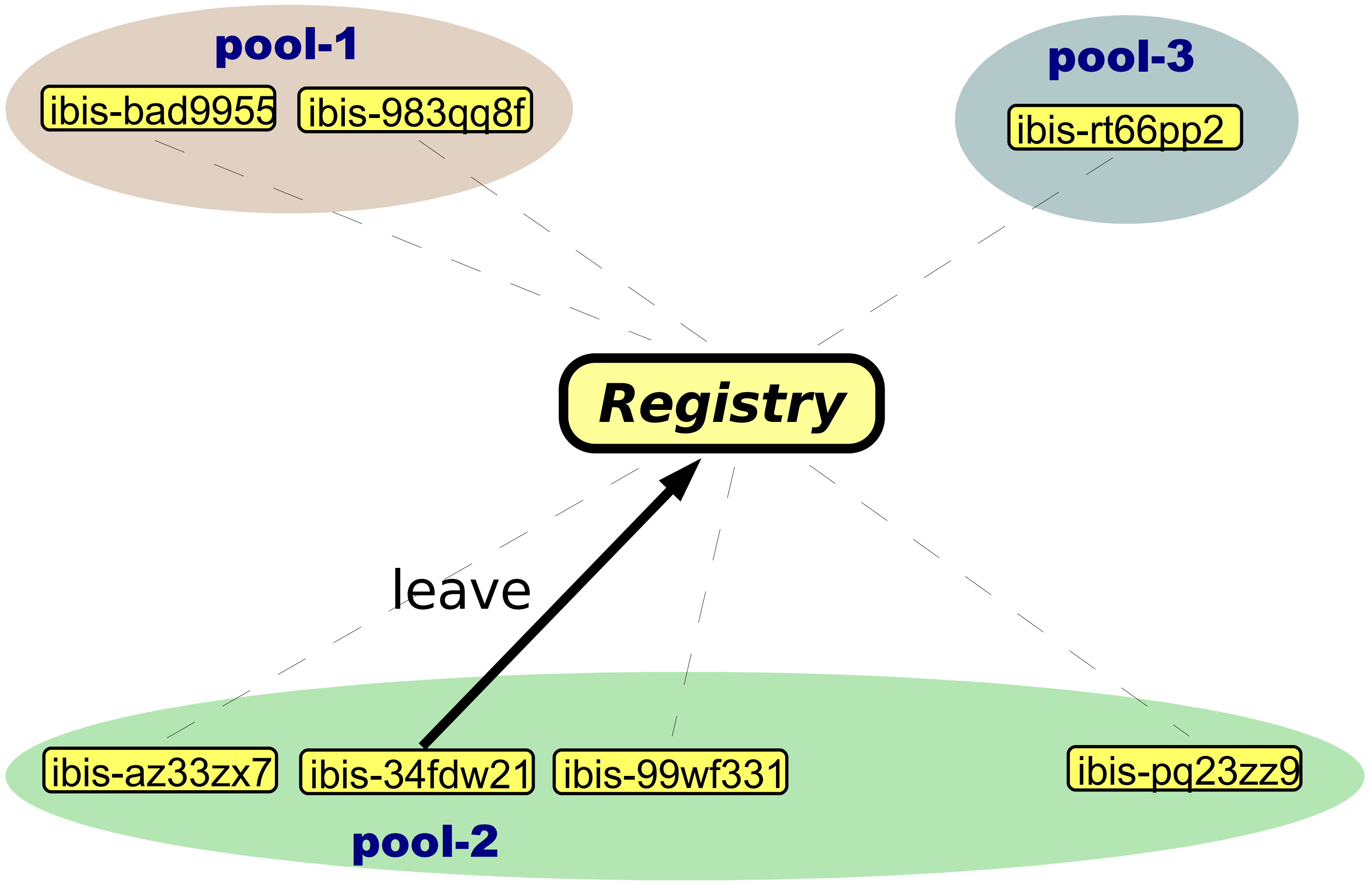
# Pools & Malleability



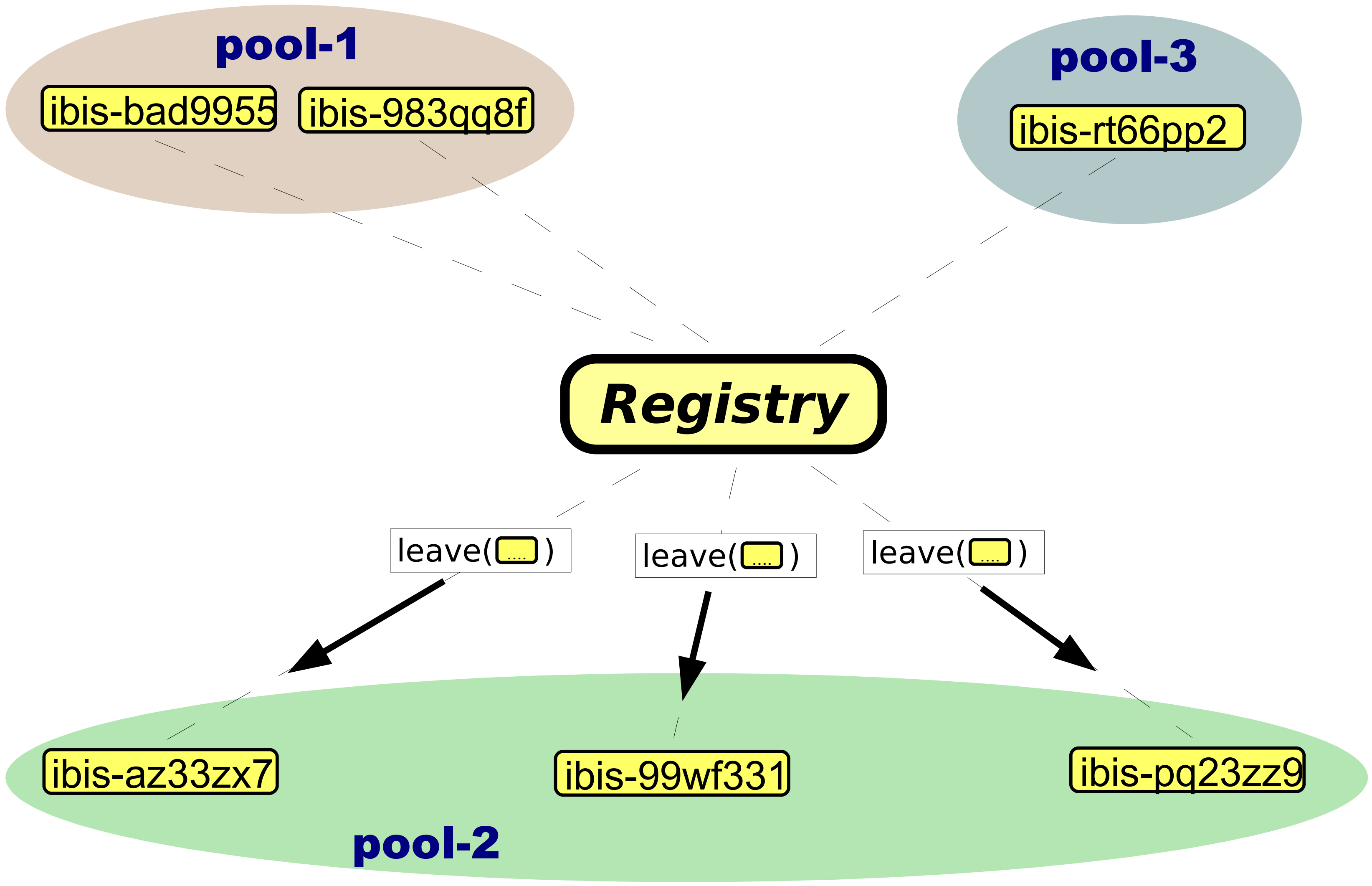
# Pools & Malleability



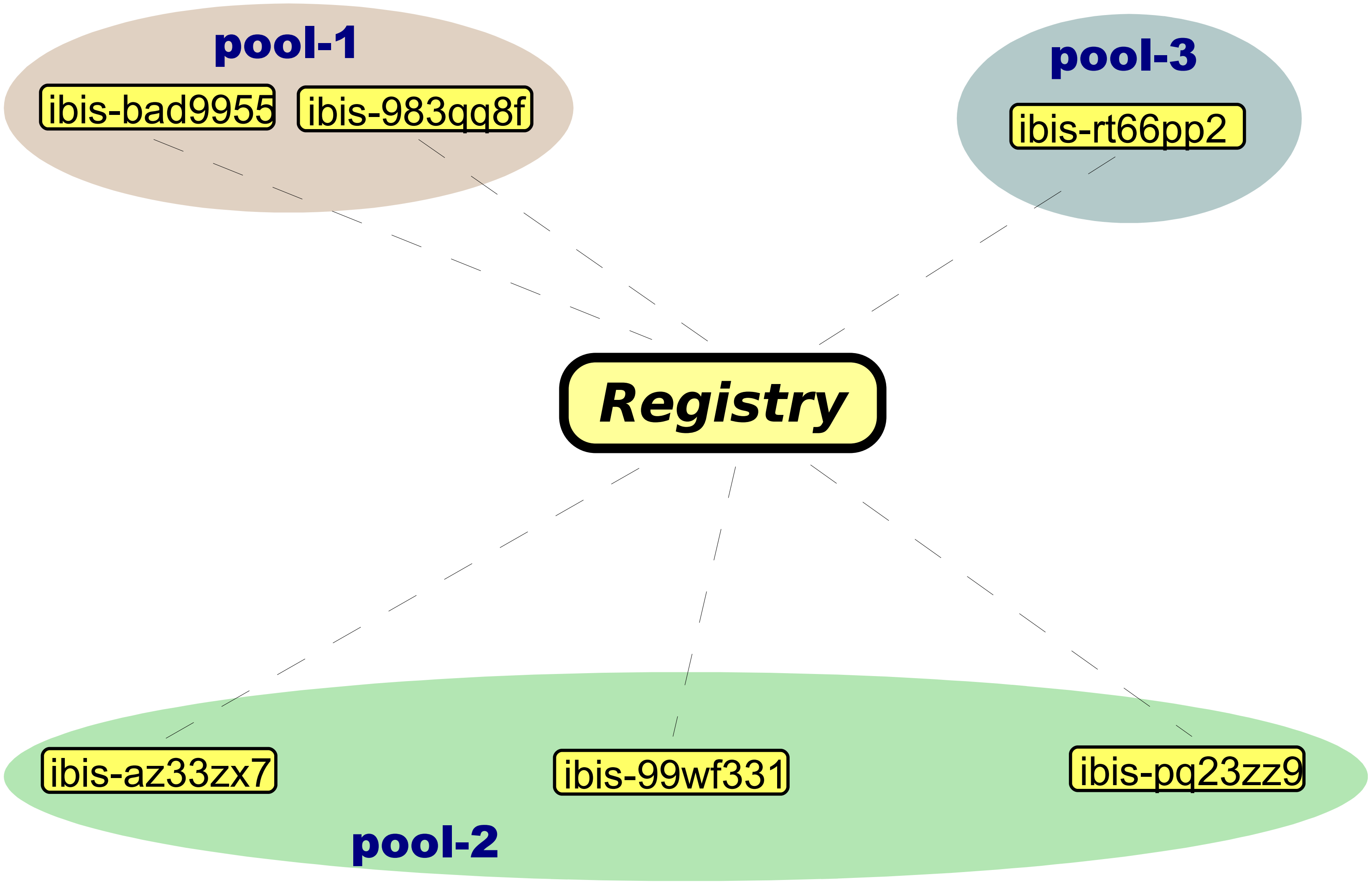
# Pools & Malleability



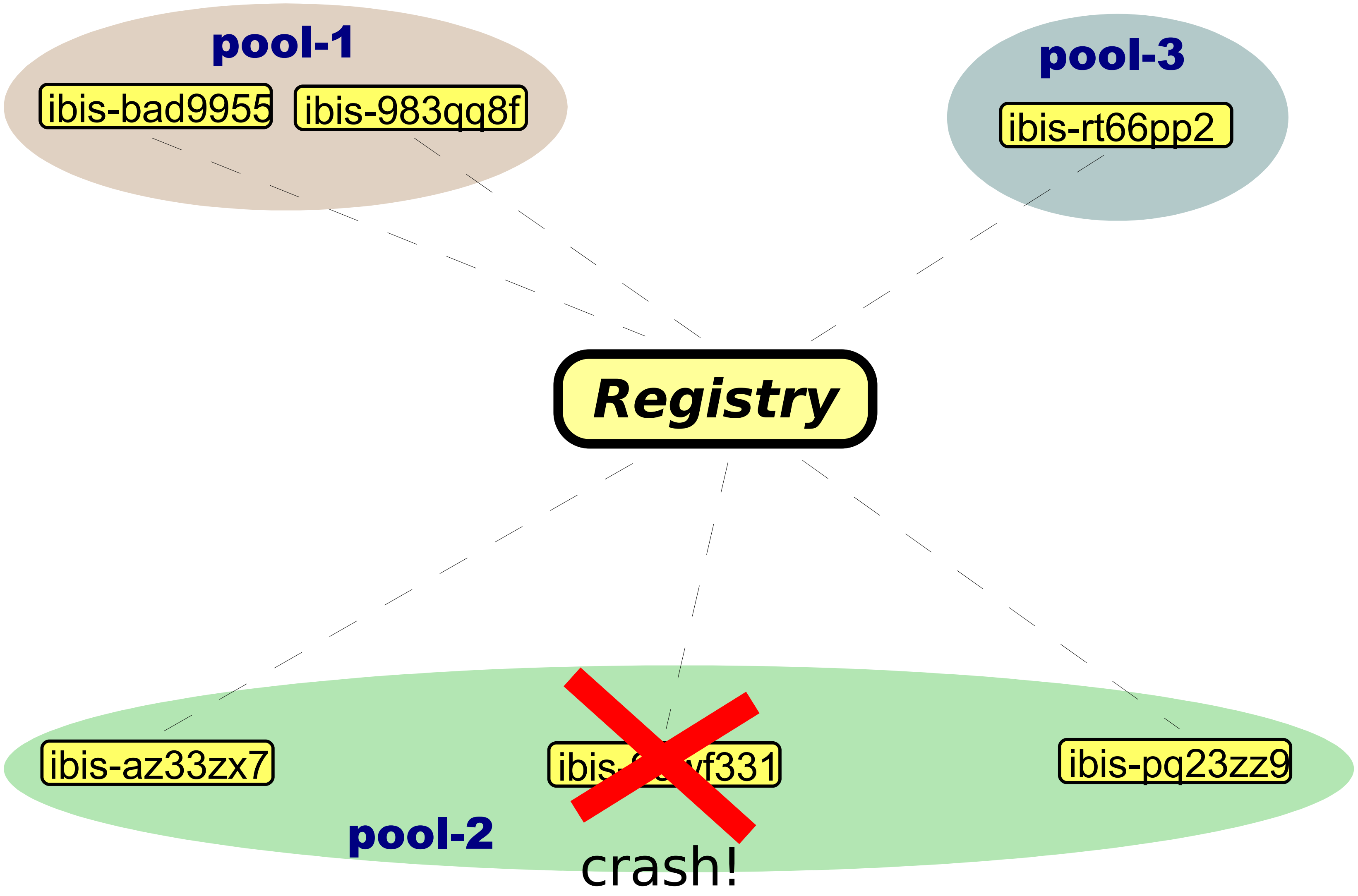
# Pools & Malleability



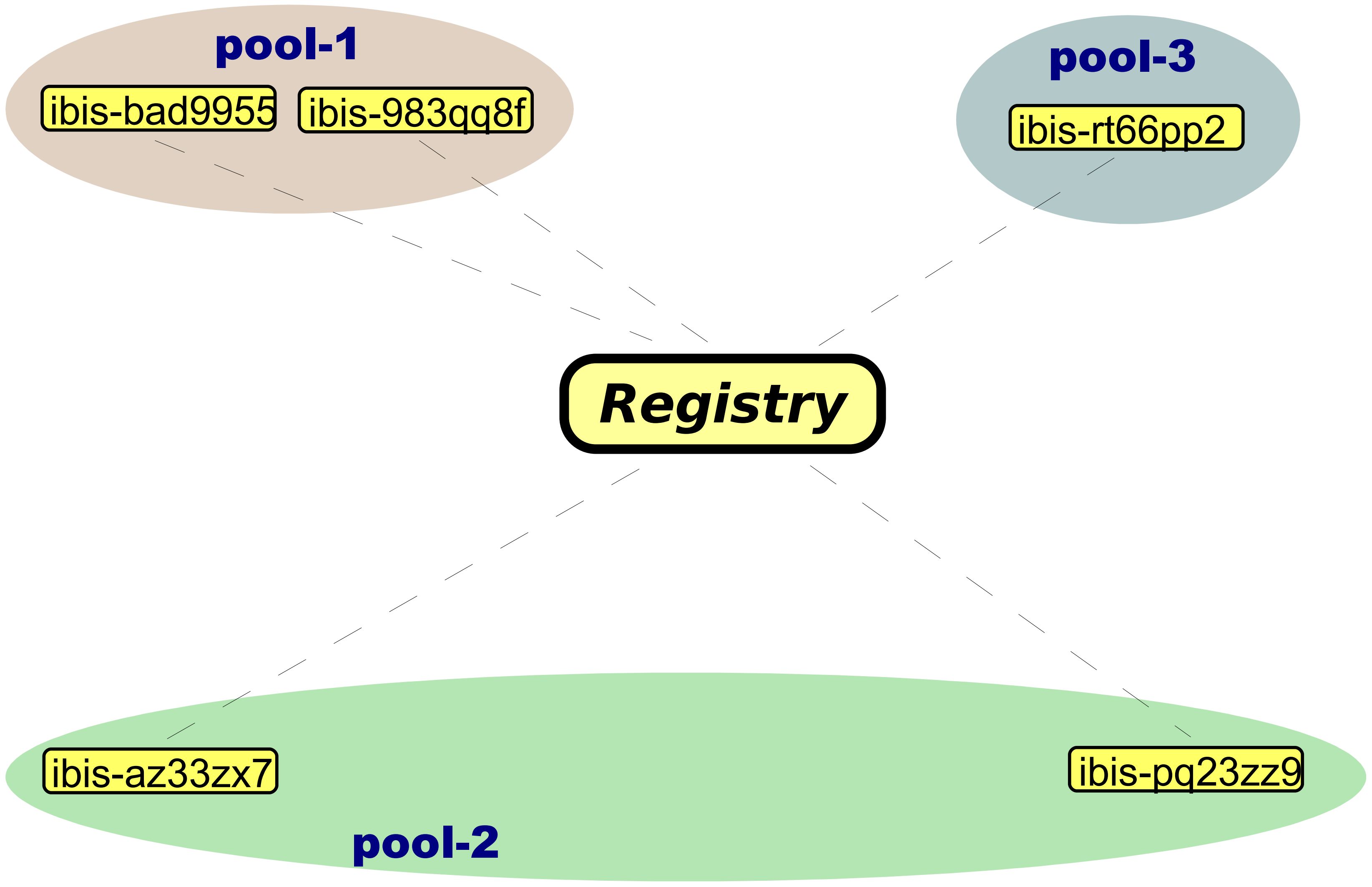
# Pools & Malleability



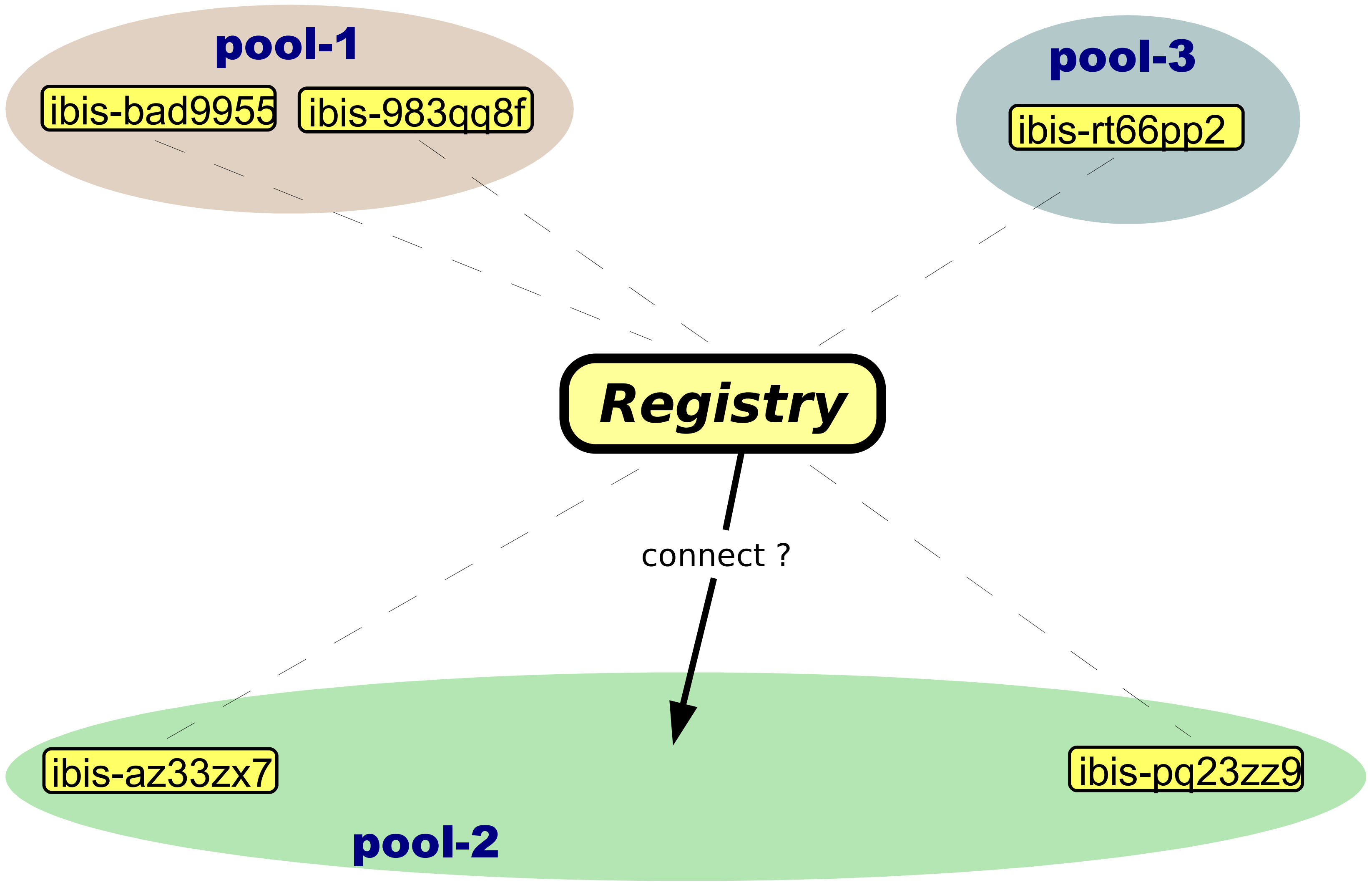
# Pools & Malleability



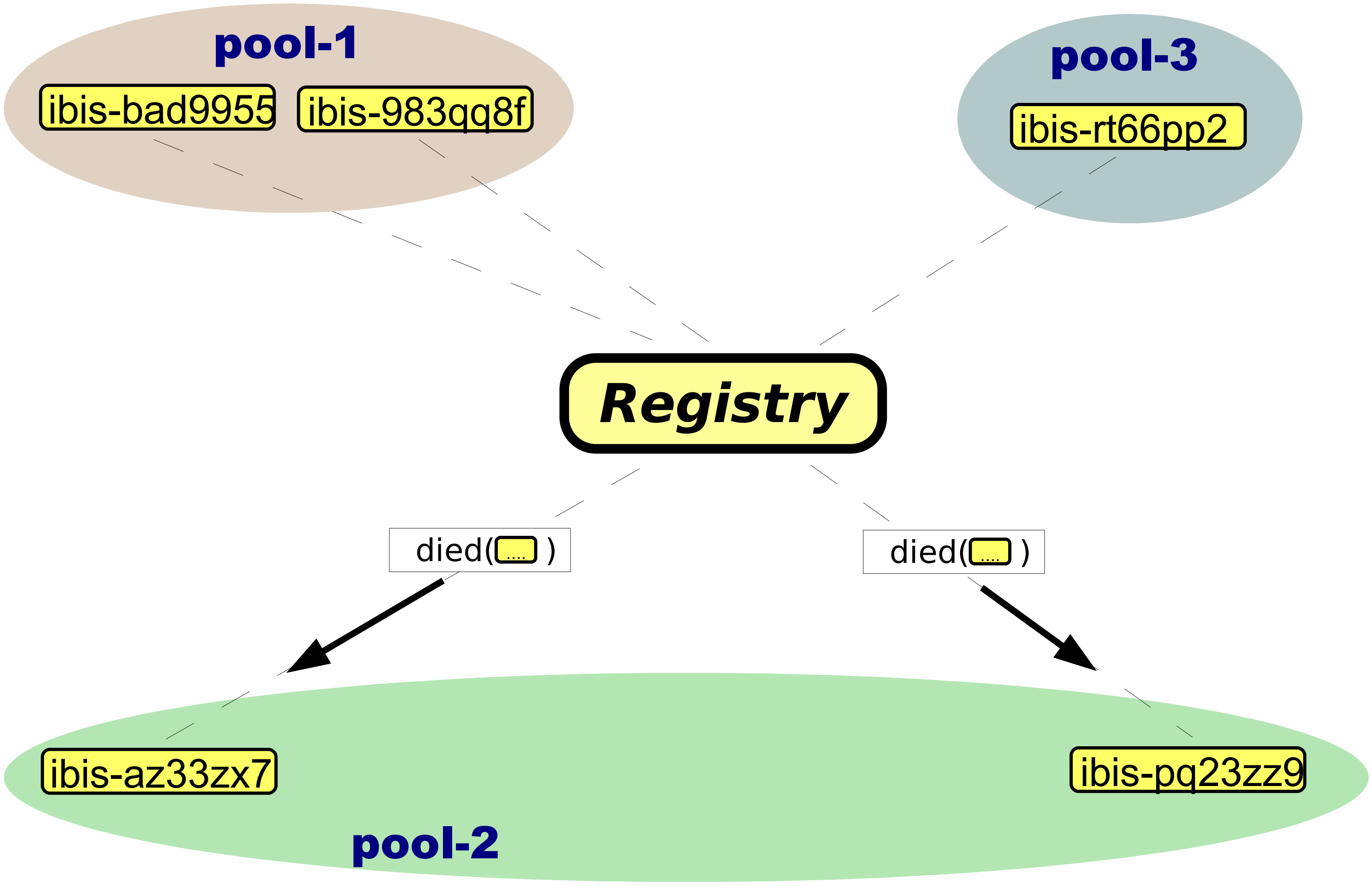
# Pools & Malleability



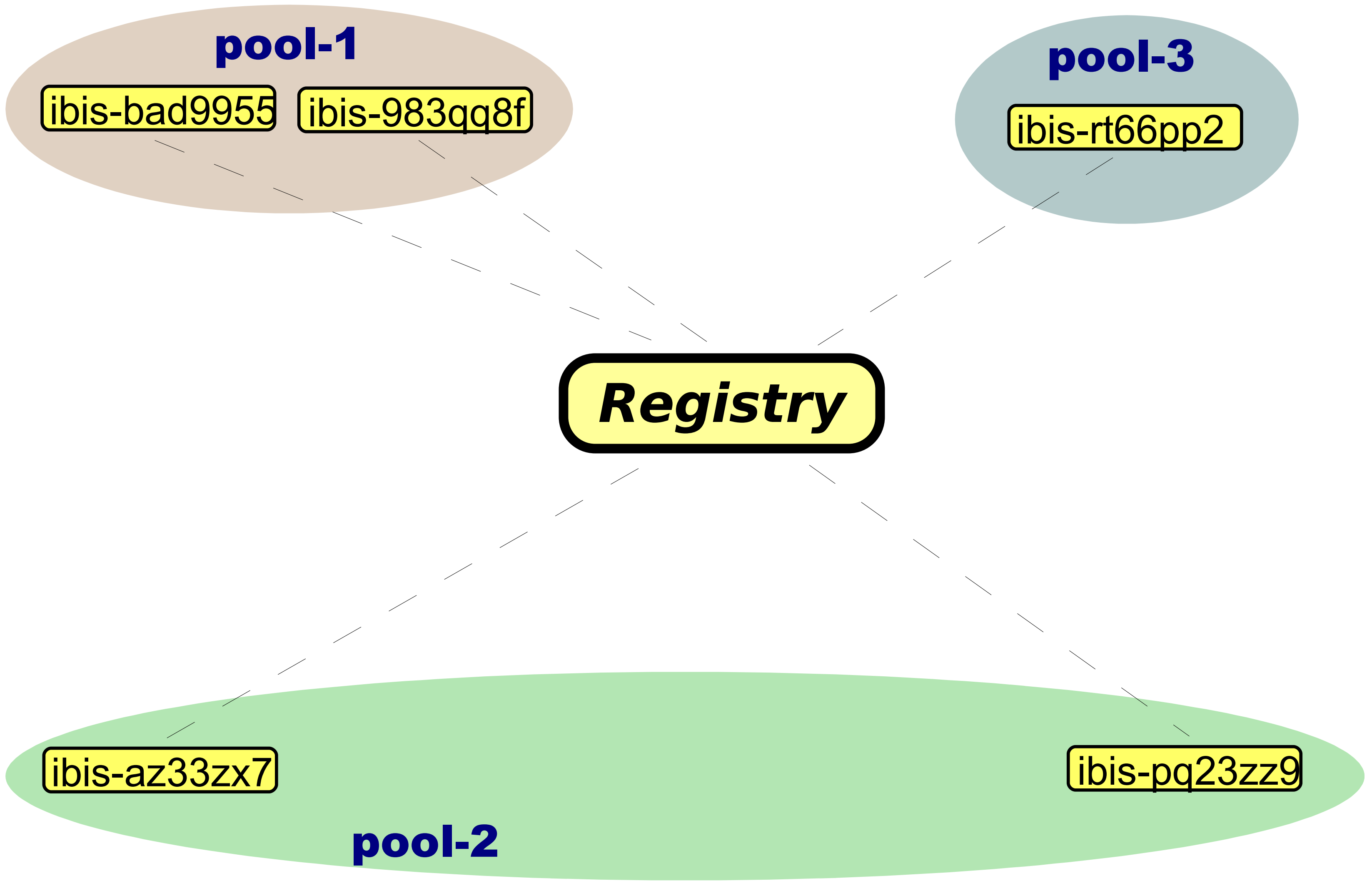
# Pools & Malleability



# Pools & Malleability



# Pools & Malleability



## *Elections*

- Registry offers an 'election' mechanism
  - Allows a group of Ibisses to determine who's in charge
- Each election
  - Has a name (String)
  - Produces IbisIdentifier of the winner
  - Is not democratic
  - You can also be 'an observer'

## *Pools & Malleability*

- This is just one example of a registry
  - Centralized implementation
  - Other implementations exist
    - none of them are interesting at the moment
  - Ongoing research
    - scalability issues
    - peer-to-peer techniques
    - distributed election mechanisms

## *Creating an Ibis*

- First step in application
    - IPL is only abstract classes & interfaces
- Ibis selects implementation for you
    - Multiple may be available
    - Selected on the basis of required capabilities and port types
        - Specify the needs of the application

# Selecting an Ibis

**IbisCapabilities:**
**closed world**
**registry upcalls.**

+

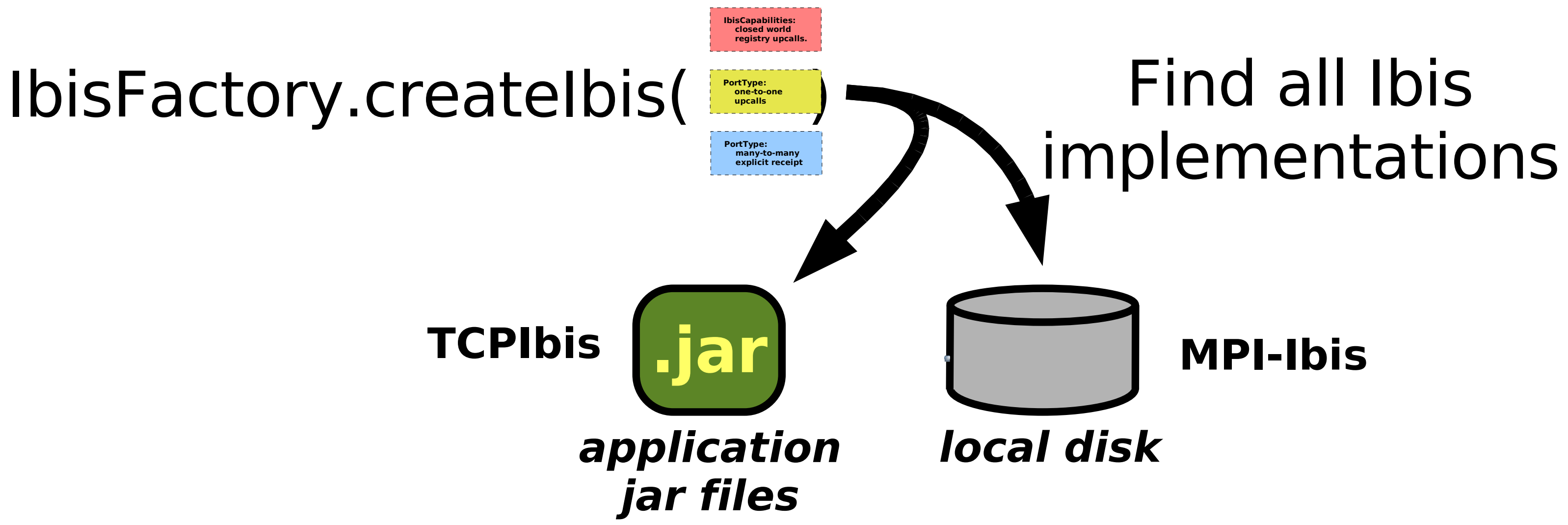
**PortType:**
**one-to-one**
**upcalls**

+

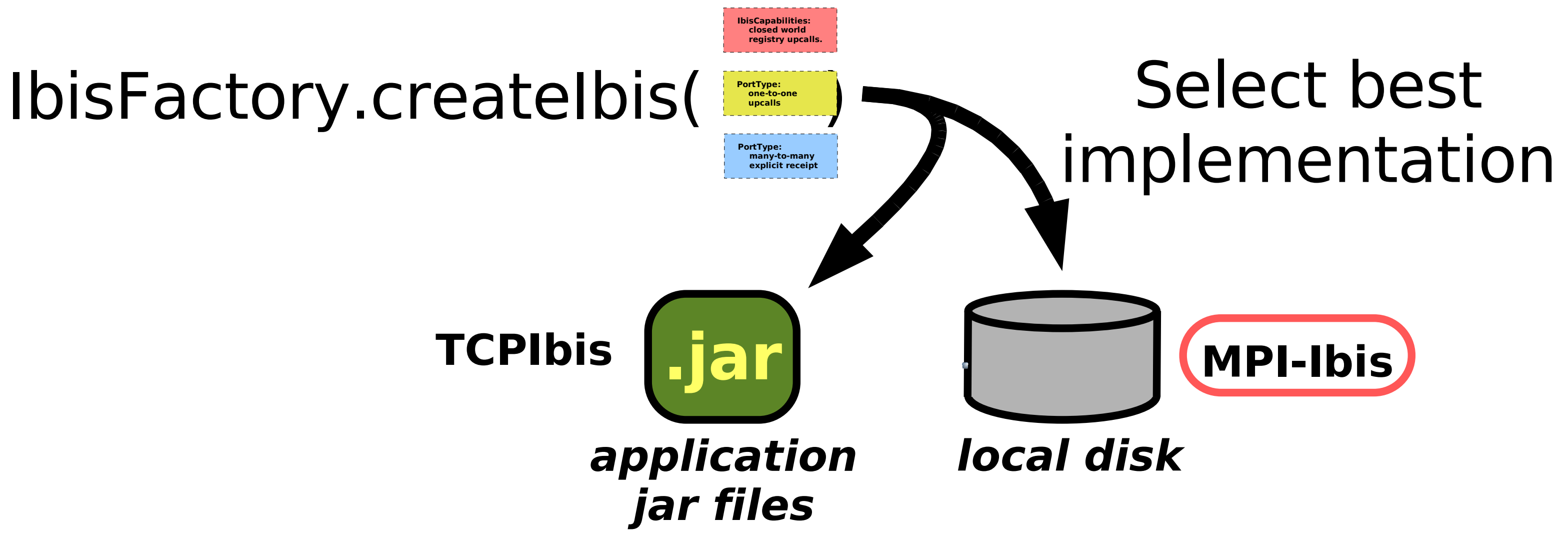
**PortType:**
**many-to-many**
**explicit receipt**

IbisFactory.createIbis(...)

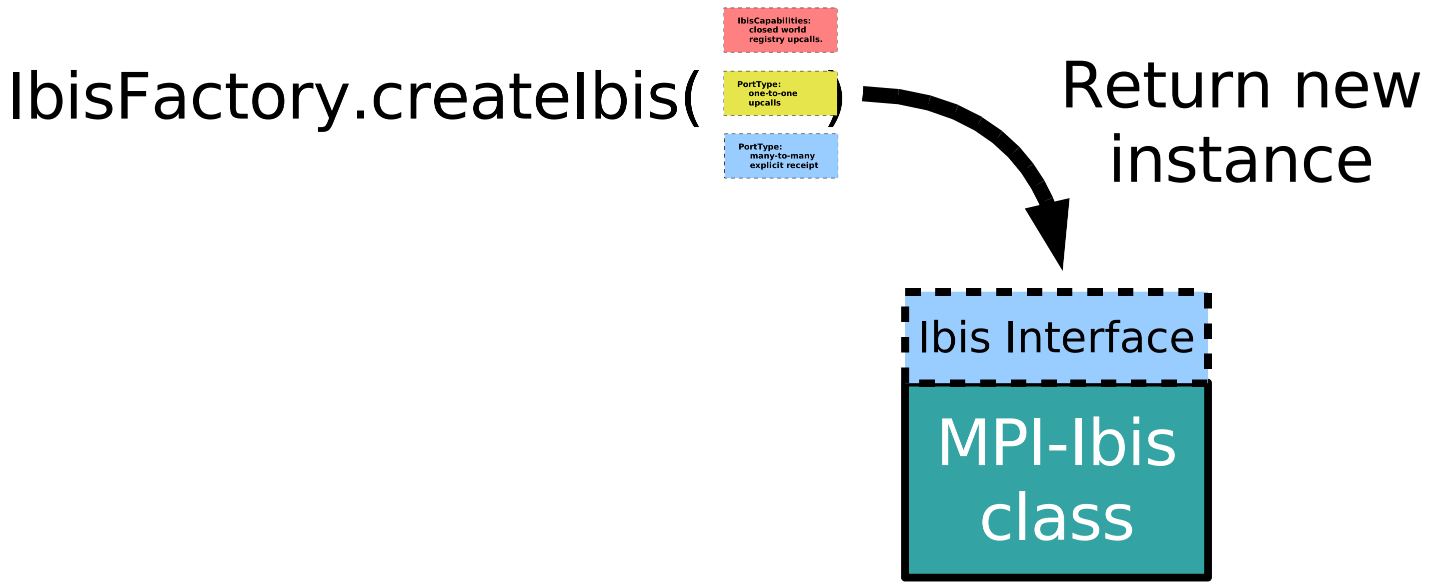
# Selecting an Ibis


IbisCapabilities: closed world registry upcalls.
PortType: one-to-one upcalls
PortType: many-to-many explicit receipt
IbisFactory.createIbis( )
Find all Ibis implementations
TCPIbis
.jar
application jar files
local disk
MPI-Ibis

# *Selecting an Ibis*


IbisFactory.createIbis( )
IbisCapabilities: closed world registry upcalls.
PortType: one-to-one upcalls
PortType: many-to-many explicit receipt
Select best implementation
TCPIbis
.jar
application jar files
local disk
MPI-Ibis

# Selecting an Ibis


IbisFactory.createIbis( )
IbisCapabilities: closed world registry upcalls.
PortType: one-to-one upcalls
PortType: many-to-many explicit receipt
Return new instance
Ibis Interface
MPI-Ibis class

## *Capabilities*

- Similar to Java properties
  - Set of boolean properties
    - “serialization.object”
    - “communication.reliable”
    - “connection.onetoone”
  - Act as switches
    - Select which features in the API are required by the application

## Capabilities

- Extensible
  - Introduce features without IPL changes
    - Just add more capabilities
  - Allows impl. specific capabilities
- Pitfalls
  - No compile time checks (only runtime)
    - Just strings
    - Sensitive to typos

## PortType Capabilities

| **Capability** | ***Description*** |
|---|---|
| connection.onetoone | Unicast |
| connection.onetomany | Multicast |
| connection.manytoone | Many to one |
| connection.manytomany | Multicast + many to one |
| communication.reliable | Reliable messages |
| communication.fifo | Fifo ordered messages |
| communication.numbered | Numbered messages |
| receive.explicit | Explicit receipt |
| receive.autoupcalls | Callback on receipt |
| receive.pollupcalls | Callback on receipt (polling required) |
| serialization.byte | Only send (arrays of) bytes |
| serialization.data | Only send (arrays of) primitive types |
| serialization.object | Send objects, don't care how |
| serialization.object.ibis | Send objects using Ibis serialization |
| serialization.object.sun | Send objects using standard ser. |

# *Ibis Capabilities*

| **Capability** | ***Description*** |
|---|---|
| registry.elections | Support elections |
| registry.worldmodel.closed | Fixed set of machines (fixed pool) |
| registry.membership | Support membership updates |
| ... | |
| ibis.malleable | Malleability support |

# Code example

```
package demo.ipl;

import ibis.ipl.*;

public class Example {

    public static void main(String args[]) throws Exception {

              // Step 1: create ibis
              PortType type = new PortType(
                PortType.COMMUNICATION_RELIABLE,
                PortType.SERIALIZATION_OBJECT,
                PortType.RECEIVE_EXPLICIT,
                PortType.CONNECTION_ONE_TO_ONE);

              IbisCapabilities cap = new IbisCapabilities(
                IbisCapabilities.ELECTIONS,
                IbisCapabilities.MALLEABLE);

              Ibis ibis = IbisFactory.createIbis(cap, null, null, type);

              // Step 2: elect server
              Registry reg = ibis.registry();
              IbisIdentifier server = reg.elect("Server");
              boolean amServer = server.equals(ibis.identifier());

              if (amServer) {
                        // Step 3, create port
                        ReceivePort rp = ibis.createReceivePort(type, "server");
                        rp.enableConnections();

                        // Step 4, receive message and read data
                        ReadMessage rm = rp.receive();
                        String tmp = (String) rm.readObject();
                        rm.finish();

                        System.out.println("Client says: " + tmp);

                        // Step 5, close port
                        rp.close();
              } else {
                        // Step 3, create send port and connect
                        SendPort sp = ibis.createSendPort(type);
                        sp.connect(server, "server");

                        // Step 4, get message and write data
                        WriteMessage wm = sp.newMessage();
                        wm.writeObject("Hello World");
                        wm.finish();

                        // Step 5, close port
                        sp.close();
              }

              // Step 6, clean up
              ibis.end();
    }
}
```

# Code example

```
package demo.ipl;

import ibis.ipl.*;

public class Example {

    public static void main(String args[]) throws Exception {

        // Step 1: create ibis
        PortType type = new PortType(
          PortType.COMMUNICATION_RELIABLE,
          PortType.SERIALIZATION_OBJECT,
          PortType.RECEIVE_EXPLICIT,
          PortType.CONNECTION_ONE_TO_ONE);

        IbisCapabilities cap = new IbisCapabilities(
          IbisCapabilities.ELECTIONS,
          IbisCapabilities.MALLEABLE);

        Ibis ibis = IbisFactory.createIbis(cap, null, null, type);

        // Step 2: elect server
```

```
rver");
.identifier());

eReceivePort(type, "server");

and read data
();
adObject();

says: " + tmp);

 and connect
ndPort(type);
);

write data
sage();
");
```

```
 //Step 1: create ibis
PortType type = new PortType(
   PortType.COMMUNICATION_RELIABLE,
   PortType.SERIALIZATION_OBJECT,
   PortType.RECEIVE_EXPLICIT,
   PortType.CONNECTION_ONE_TO_ONE);

IbisCapabilities cap = new IbisCapabilities(
   IbisCapabilities.ELECTIONS,
   IbisCapabilities.MALLEABLE);

Ibis ibis = IbisFactory.createIbis(cap, null, null, type);
```

# Code example

```
package demo.ipl;

import ibis.ipl.*;

public class Example {

    public static void main(String args[]) throws Exception {

              // Step 1: create ibis
              PortType type = new PortType(
                PortType.COMMUNICATION_RELIABLE,
                PortType.SERIALIZATION_OBJECT,
                PortType.RECEIVE_EXPLICIT,
                PortType.CONNECTION_ONE_TO_ONE);

              IbisCapabilities cap = new IbisCapabilities(
                IbisCapabilities.ELECTIONS,
                IbisCapabilities.MALLEABLE);

              Ibis ibis = IbisFactory.createIbis(cap, null, null, type);

              // Step 2: elect server
              Registry reg = ibis.registry();
              IbisIdentifier server = reg.elect("Server");
              boolean amServer = server.equals(ibis.identifier());

              if (amServer) {
                        // Step 3, create port
                                                            t(type, "server");

                                                              ta



                                                            mp);




                                                            t
                        SendPort sp = ibis.createSendPort(type);
                        sp.connect(server, "server");

                        // Step 4, get message and write data
                        WriteMessage wm = sp.newMessage();
                        wm.writeObject("Hello World");
                        wm.finish();

                        // Step 5, close port
                        sp.close();
              }

              // Step 6, clean up
              ibis.end();
    }
}
```

```
// Step 2: elect server
Registry reg = ibis.registry();
IbisIdentifier server = reg.elect("Server");
boolean amServer = server.equals(ibis.identifier());
```

# Code example

```
package demo.ipl;

import ibis.ipl.*;

public class Example {

    public static void main(String args[]) throws Exception {

              // Step 1: create ibis
              PortType type = new PortType(
                PortType.COMMUNICATION_RELIABLE,
                PortType.SERIALIZATION_OBJECT,
                PortType.RECEIVE_EXPLICIT,
                PortType.CONNECTION_ONE_TO_ONE);

              IbisCapabilities cap = new IbisCapabilities(
                IbisCapabilities.ELECTIONS,
                IbisCapabilities.MALLEABLE);

              Ibis ibis = IbisFactory.createIbis(cap, null, null, type);

              // Step 2: elect server
              Registry reg = ibis.registry();
              IbisIdentifier server = reg.elect("Server");
              boolean amServer = server.equals(ibis.identifier());

              if (amServer) {
                        // Step 3, create port
                        ReceivePort rp = ibis.createReceivePort(type, "server");
                        rp.enableConnections();

                        // Step 4, receive message and read data
                        ReadMessage rm = rp.receive();
```

```
if (amServer) {
        // Step 3, create port
        ReceivePort rp = ibis.createReceivePort(type, "server");
        rp.enableConnections();
```

```
                        WriteMessage wm = sp.newMessage();
                        wm.writeObject("Hello World");
                        wm.finish();

                        // Step 5, close port
                        sp.close();
              }

              // Step 6, clean up
              ibis.end();
    }
}
```

# Code example

```
package demo.ipl;

import ibis.ipl.*;

public class Example {

    public static void main(String args[]) throws Exception {

              // Step 1: create ibis
              PortType type = new PortType(
                PortType.COMMUNICATION_RELIABLE,
                PortType.SERIALIZATION_OBJECT,
                PortType.RECEIVE_EXPLICIT,
                PortType.CONNECTION_ONE_TO_ONE);

              IbisCapabilities cap = new IbisCapabilities(
                IbisCapabilities.ELECTIONS,
                IbisCapabilities.MALLEABLE);

              Ibis ibis = IbisFactory.createIbis(cap, null, null, type);

              // Step 2: elect server
              Registry reg = ibis.registry();
                                ver = reg.elect("Server");
                                 server.equals(ibis.identifier());

                                3, create port
                                ort rp = ibis.createReceivePort(type, "server");
                                eConnections();

                                4, receive message and read data
                                age rm = rp.receive();
                        String tmp = (String) rm.readObject();
                        rm.finish();

                        System.out.println("Client says: " + tmp);

                        // Step 5, close port
                        rp.close();
              } else {
                        // Step 3, create send port and connect
                        SendPort sp = ibis.createSendPort(type);
                        sp.connect(server, "server");

                        // Step 4, get message and write data
                        WriteMessage wm = sp.newMessage();
                        wm.writeObject("Hello World");
                        wm.finish();

                        // Step 5, close port
                        sp.close();
              }

              // Step 6, clean up
              ibis.end();
    }
}
```

```
// Step 3, create send port and connect
SendPort sp = ibis.createSendPort(type);
sp.connect(server, "server");
```

# Code example

```
package demo.ipl;

import ibis.ipl.*;

public class Example {

    public static void main(String args[]) throws Exception {

              // Step 1: create ibis
              PortType type = new PortType(
                PortType.COMMUNICATION_RELIABLE,
                PortType.SERIALIZATION_OBJECT,
                PortType.RECEIVE_EXPLICIT,
                PortType.CONNECTION_ONE_TO_ONE);

              IbisCapabilities cap = new IbisCapabilities(
                IbisCapabilities.ELECTIONS,
                IbisCapabilities.MALLEABLE);

              Ibis ibis = IbisFactory.createIbis(cap, null, null, type);

              // Step 2: elect server
              Registry reg = ibis.registry();
                             server = reg.elect("Server");
                             r = server.equals(ibis.identifier());


                             ep 3, create port
                             vePort rp = ibis.createReceivePort(type, "server");
                             ableConnections();

                             ep 4, receive message and read data
                             essage rm = rp.receive();
                             g tmp = (String) rm.readObject();
                             nish();

                        System.out.println("Client says: " + tmp);

                        // Step 5, close port
                        rp.close();
              } else {
                        // Step 3, create send port and connect
                        SendPort sp = ibis.createSendPort(type);
                        sp.connect(server, "server");

                        // Step 4, get message and write data
                        WriteMessage wm = sp.newMessage();
                        wm.writeObject("Hello World");
                        wm.finish();

                        // Step 5, close port
                        sp.close();
              }

              // Step 6, clean up
              ibis.end();
    }
}
```

```
// Step 4, get message and write data
WriteMessage wm = sp.newMessage();
wm.writeObject("Hello World");
wm.finish();
```

# Code example

```
package demo.ipl;

import ibis.ipl.*;

public class Example {

    public static void main(String args[]) throws Exception {

              // Step 1: create ibis
              PortType type = new PortType(
                PortType.COMMUNICATION_RELIABLE,
                PortType.SERIALIZATION_OBJECT,
                PortType.RECEIVE_EXPLICIT,
                PortType.CONNECTION_ONE_TO_ONE);

              IbisCapabilities cap = new IbisCapabilities(
                IbisCapabilities.ELECTIONS,
                IbisCapabilities.MALLEABLE);

              Ibis ibis = IbisFactory.createIbis(cap, null, null, type);

              // Step 2: elect server
              Registry reg = ibis.registry();
              IbisIdentifier server = reg.elect("Server");
              boolean amServer = server.equals(ibis.identifier());

              if (amServer) {
                        // Step 3, create port
                        ReceivePort rp = ibis.createReceivePort(type, "server");
                        rp.enableConnections();

                        // Step 4, receive message and read data
                        ReadMessage rm = rp.receive();
                        String tmp = (String) rm.readObject();
                        rm.finish();

                                              ient says: " + tmp);



                                               port and connect
                                              ateSendPort(type);
                                              rver");

                                               and write data
                                              ewMessage();
                                              World");



    }
}
```

```
// Step 4, receive message and read data
ReadMessage rm = rp.receive();
String tmp = (String) rm.readObject();
rm.finish();

System.out.println("Client says: " + tmp);
```

# Code example

```
package demo.ipl;

import ibis.ipl.*;

public class Example {

    public static void main(String args[]) throws Exception {

              // Step 1: create ibis
              PortType type = new PortType(
                PortType.COMMUNICATION_RELIABLE,
                PortType.SERIALIZATION_OBJECT,
                PortType.RECEIVE_EXPLICIT,
                PortType.CONNECTION_ONE_TO_ONE);

              IbisCapabilities cap = new IbisCapabilities(
                IbisCapabilities.ELECTIONS,
                IbisCapabilities.MALLEABLE);

              Ibis ibis = IbisFactory.createIbis(cap, null, null, type);

              // Step 2: elect server
              Registry reg = ibis.registry();
              IbisIdentifier server = reg.elect("
              boolean amServer = server.equals(ib

              if (amServer) {
                        // Step 3, create port
                        ReceivePort rp = ibis.cre
                        rp.enableConnections();

                        // Step 4, receive message and read data
                        ReadMessage rm = rp.receive();
                        String tmp = (String) rm.readObject();
                        rm.finish();

                        System.out.println("Client says: " + tmp);

                        // Step 5, close port
                        rp.close();
              } else {
                        // Step 3, create send port
                        SendPort sp = ibis.createSen
                        sp.connect(server, "server")

                        // Step 4, get message and w
                        WriteMessage wm = sp.newMess
                        wm.writeObject("Hello World"
                        wm.finish();

                        // Step 5, close port
                        sp.close();
              }

              // Step 6, clean up
              ibis.end();
    }
}
```

```
// Step 5, close port
rp.close();
```

```
// Step 5, close port
sp.close();
```

# Code example

```
package demo.ipl;

import ibis.ipl.*;

public class Example {

    public static void main(String args[]) throws Exception {

              // Step 1: create ibis
              PortType type = new PortType(
                PortType.COMMUNICATION_RELIABLE,
                PortType.SERIALIZATION_OBJECT,
                PortType.RECEIVE_EXPLICIT,
                PortType.CONNECTION_ONE_TO_ONE);

              IbisCapabilities cap = new IbisCapabilities(
                IbisCapabilities.ELECTIONS,
                IbisCapabilities.MALLEABLE);

              Ibis ibis = IbisFactory.createIbis(cap, null, null, type);

              // Step 2: elect server
              Registry reg = ibis.registry();
              IbisIdentifier server = reg.elect("Server");
              boolean amServer = server.equals(ibis.identifier());

              if (amServer) {
                        // Step 3, create port
                        ReceivePort rp = ibis.createReceivePort(type, "server");
                        rp.enableConnections();

                        // Step 4, receive message and read data
                        ReadMessage rm = rp.receive();
                        String tmp = (String) rm.readObject();
                        rm.finish();

                        System.out.println("Client says: " + tmp);

                        // Step 5, close port
                        rp.close();
              } else {
                        // Step 3, create send port and connect
                        SendPort sp = ibis.createSendPort(type);
                        sp.connect(server, "server");

                        // Step 4, get message and write data
                        WriteMessage wm = sp.newMessage();
                        wm.writeObject("Hello World");
                        wm.finish();

                        // Step 5, close port
                        sp.close(
              }

              // Step 6, clean up
              ibis.end();

    }
}
```

```
//Step, clean up
ibis.end();
```

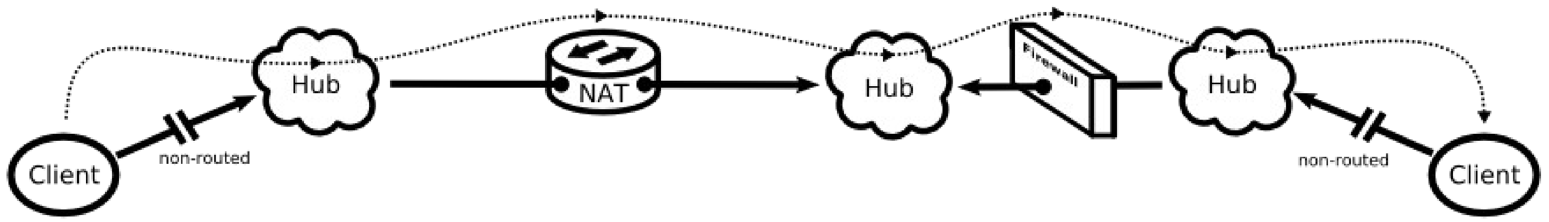
## Code example

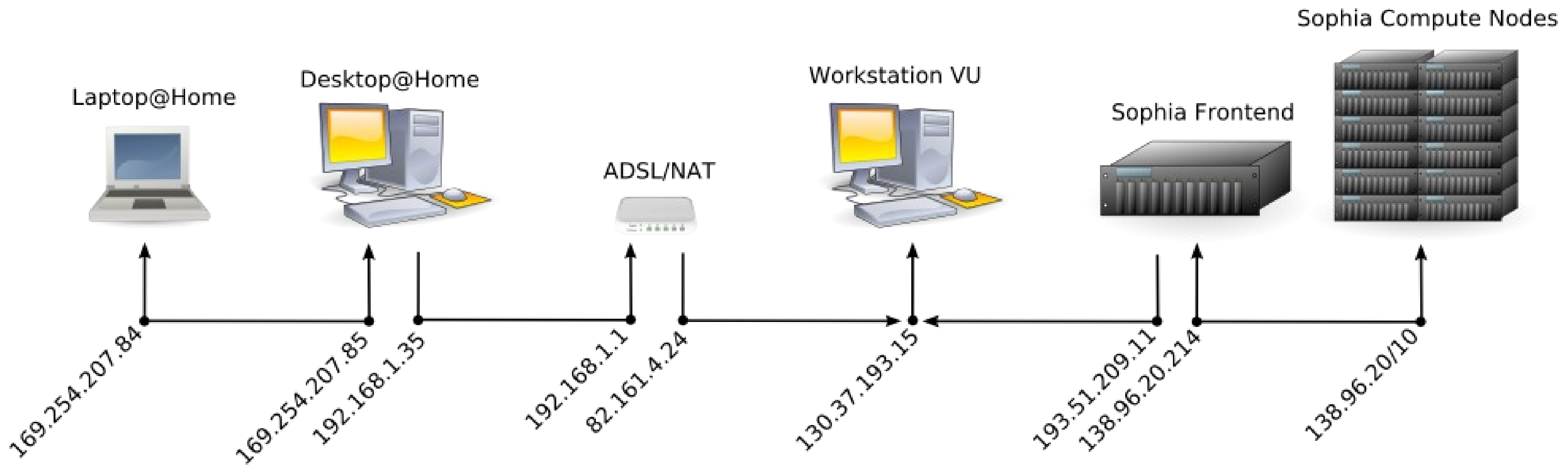
- Live demo

## *Other cool features*

- Ibis can handle 'complicated network setups' using SmartSockets
- Real-world examples:
  - Multiple networks
    - Mix local and global IPs depending on target
  - NAT / firewalls
    - Using UPnP, STUN, TCP splicing, etc.
    - Routing messages through external points
  - Non-routed networks
    - Routing messages through external points

# Example


Client
non-routed
Hub
NAT
Hub
Firewall
Hub
non-routed
Client


Laptop@Home
Desktop@Home
ADSL/NAT
Workstation VU
Sophia Frontend
Sophia Compute Nodes
169.254.207.84
169.254.207.85
192.168.1.35
192.168.1.1
82.161.4.24
130.37.193.15
193.51.209.11
138.96.20.214
138.96.20/10

## Programming models

- Remote Method Invocation (RMI)
- Group Method Invocation (GMI)
- Satin (Divide & Conquer)
- MPJ (MPI Java 'standard')
- Others are being developed
  - Balutek (data parallel)
  - Replicated Method Invocation (RepMI)

## *Satin*

- Parallel Divide-and-conquer
  - Divide work into independent parts
  - Spawn sub-jobs
  - Combine sub-results
  - Repeat recursively
- Master-Worker is a subset of this
  - Only one level of recursion
- Targeted at the grid (and clusters)

# Sequential Fibonacci

```
public long fib(int n) {
        if (n < 2) return n;

        long x = fib(n - 1);
        long y = fib(n – 2);

        return x + y;
}
```

# Parallel Fibonacci

```
interface FibInterface extends ibis.satin.Spawnable {
    public long fib(int n);
}
public long fib(int n) {
        if (n < 2) return n;

        long x = fib(n - 1);
        long y = fib(n – 2);
        sync();
        return x + y;
}
```

## Parallel Fibonacci

```
interface FibInterface extends ibis.satin.Spawnable {
    public long fib(int n);
}
public long fib(int n) {
        if (n < 2) return n;

        long x = fib(n - 1);
        long y = fib(n – 2);
        sync();
        return x + y;
}
```

Mark methods as

Spawnable.

They can run in parallel.

# Parallel Fibonacci

```
interface FibInterface extends ibis.satin.Spawnable {
    public long fib(int n);
}
public long fib(int n) {
        if (n < 2) return n;

        long x = fib(n - 1);
        long y = fib(n – 2);
        sync();
        return x + y;
}
```

Mark methods as

Spawnable.

They can run in parallel.

Wait until spawned methods are done.

vrije Universiteit

## *Satin features*

- Satin distributes jobs across machines

- Load-balancing is done automatically
  - Uses random stealing
  - Algorithm has been proven to be optimal on homogeneous systems
  - Additional highly-efficient grid-aware algorithms

# *Satin features*

- Malleability
  - Add/remove machines on the fly
- Fault-tolerance
  - When a machine leave suddenly (crashes) the others continue the computation and automatically recompute the lost work
- Shared Objects (added recently)
  - Allows machines to share 'global data'

## *Satin Applications*

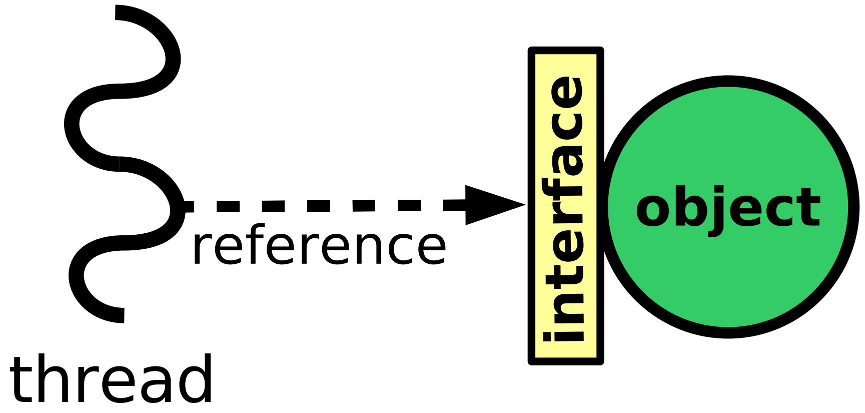
- More interesting applications
  - Satisfiability solver
  - Gene sequencing
  - N-body simulations
  - Grammar-based text analysis
  - Game-tree search
  - Raytracing
  - Numerical functions
  - ...

# *Satin Applications*

- More interesting applications
  - Satisfiability solver
  - Gene sequencing
  - N-body simulations $\longleftarrow$ Demo!
  - Grammar-based text analysis
  - Game-tree search
  - Raytracing
  - Numerical functions
  - ...

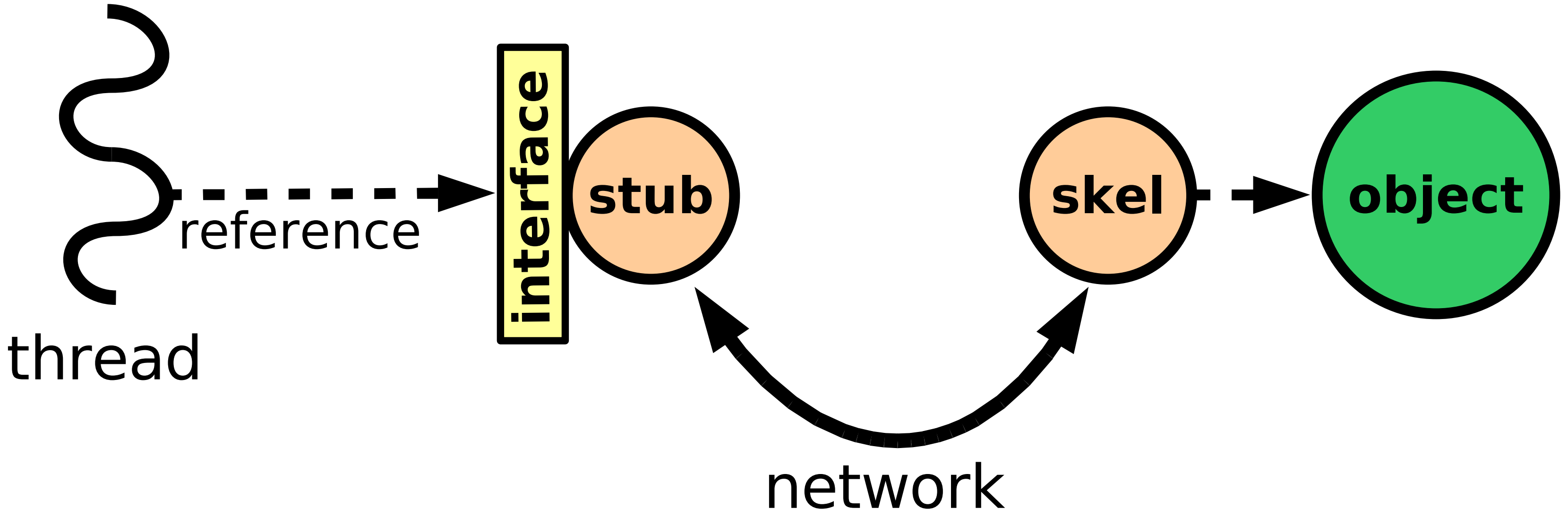
# *Ibis RMI*

- Replacement for Sun RMI
  - Has the same interface
  - Used different stub compiler (rmic)
    - Generates Ibis specific stubs/skeletons



# *Ibis RMI*

- Replacement for Sun RMI
  - Has the same interface
  - Used different stub compiler (rmic)
    - Generates Ibis specific stubs/skeletons



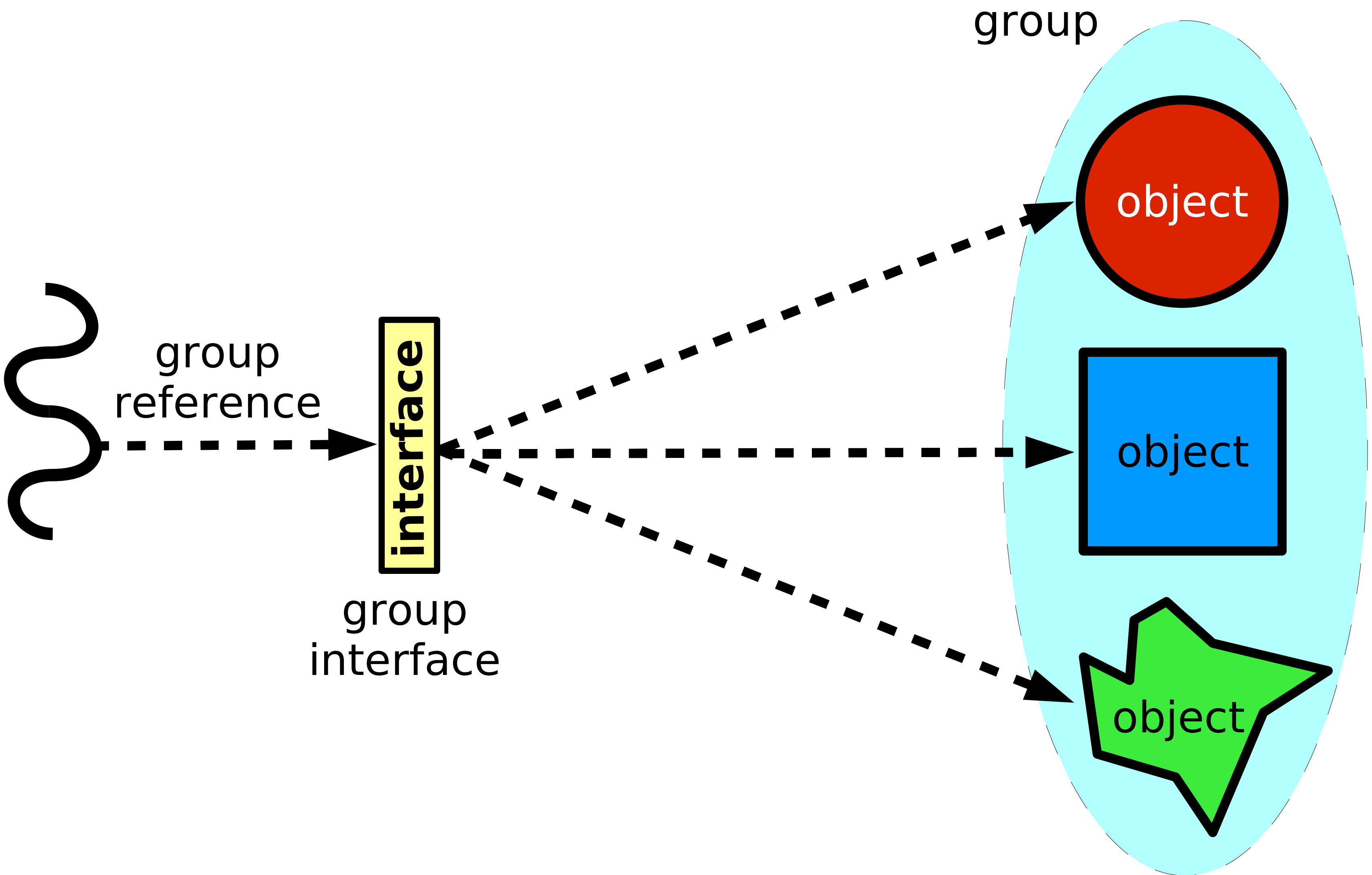
## *Ibis RMI*

- Not interoperable with Sun RMI
  - uses a different protocol
- No socket factories
  - Ibis doesn't have to use sockets!
- No activatable objects

## *GMI*

- Generalized RMI model
  - Allows communication with groups
    - A single stub refers to an entire group
  - Allows more 'advanced' communication
    - By offering different ways of forwarding a method invocation and handling the reply

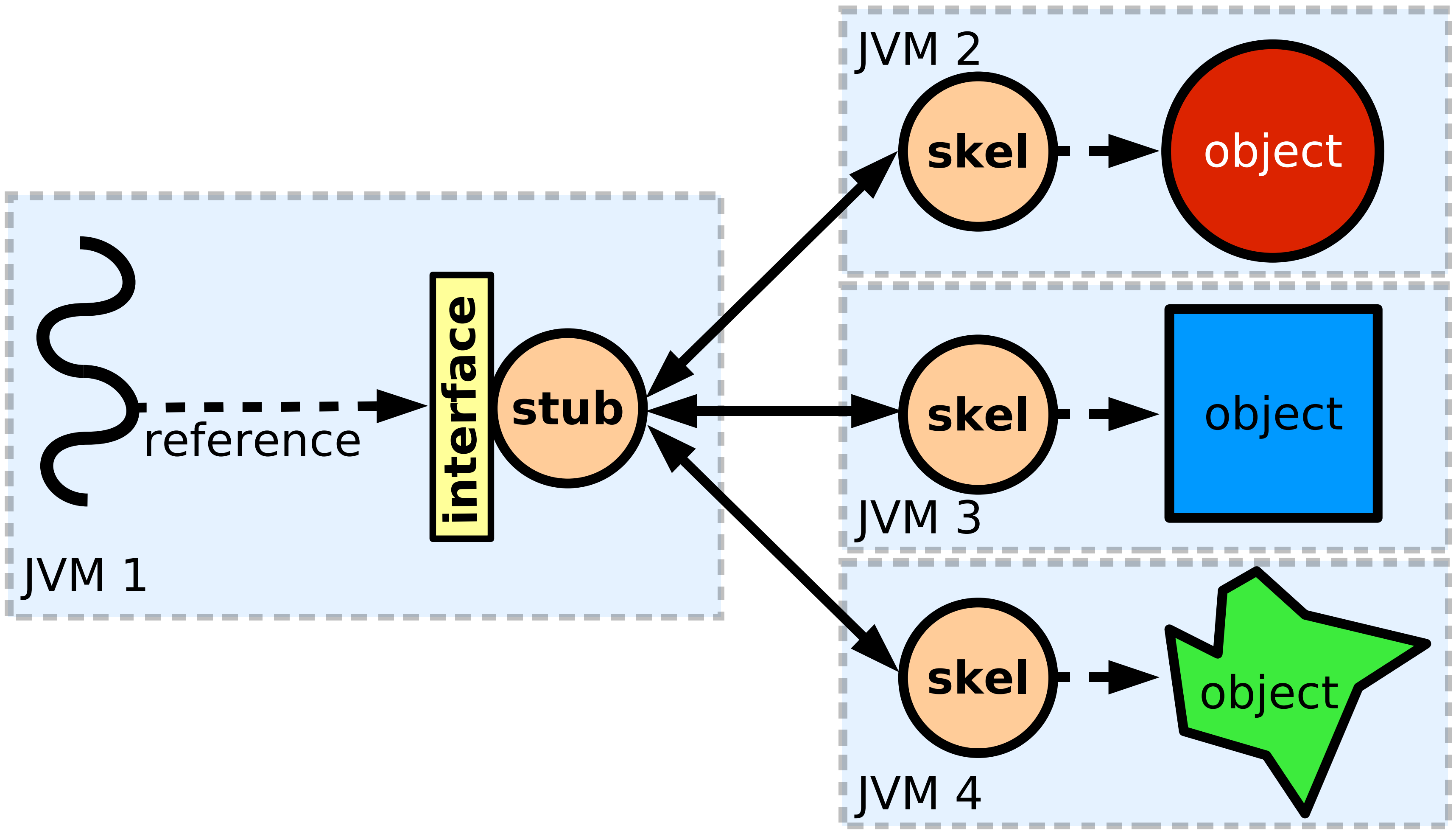
# GMI Example



ibis

vrije Universiteit

# GMI Implementation



vrije Universiteit

## *Group operations*

- The group reference can be configured
  - How is a method invocation handled
  - How is the method result handled
  - Configuration per method

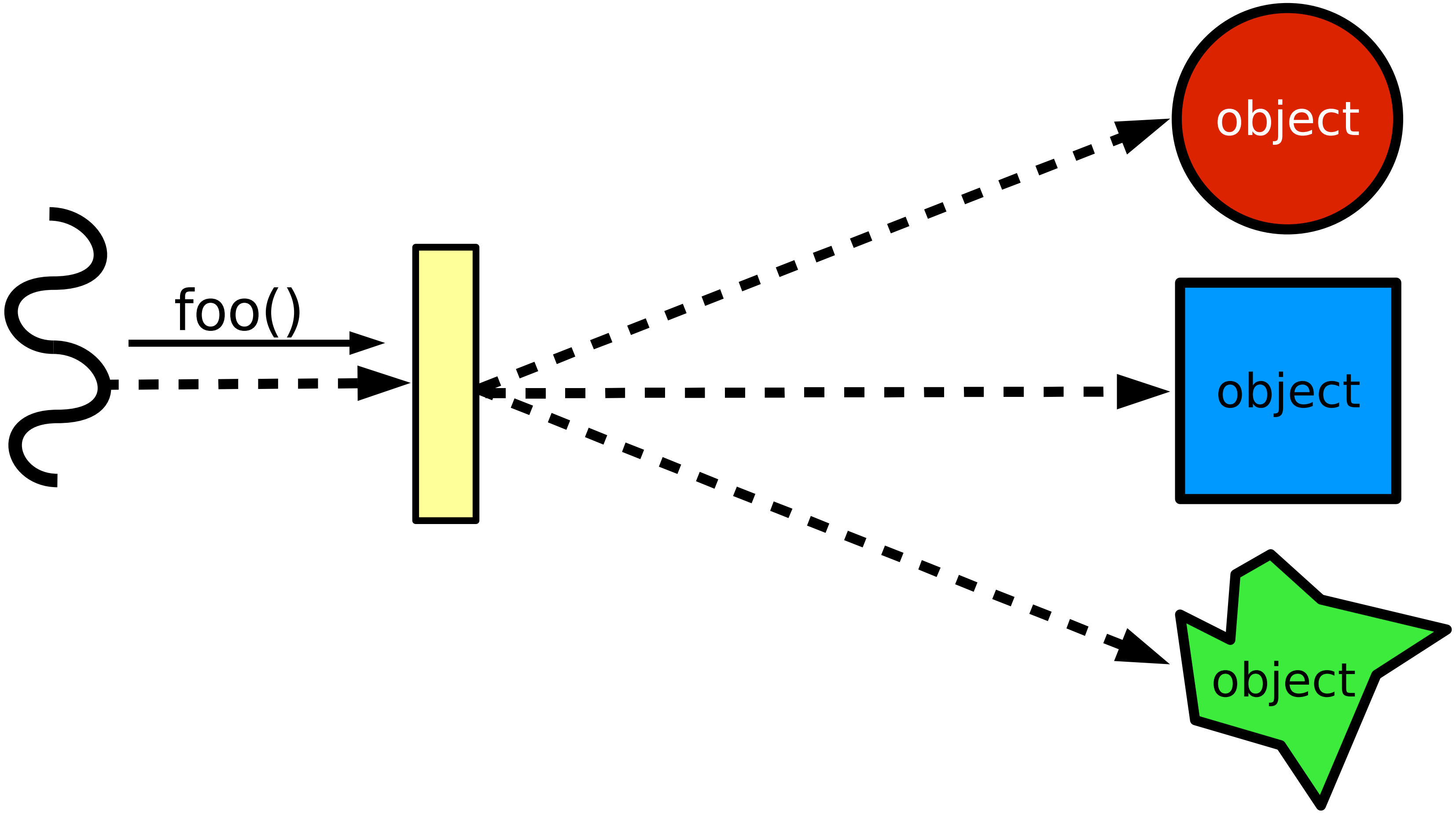
- Implemented by selecting different communication code in the generated stubs and skeletons

## *Invocation Schemes*

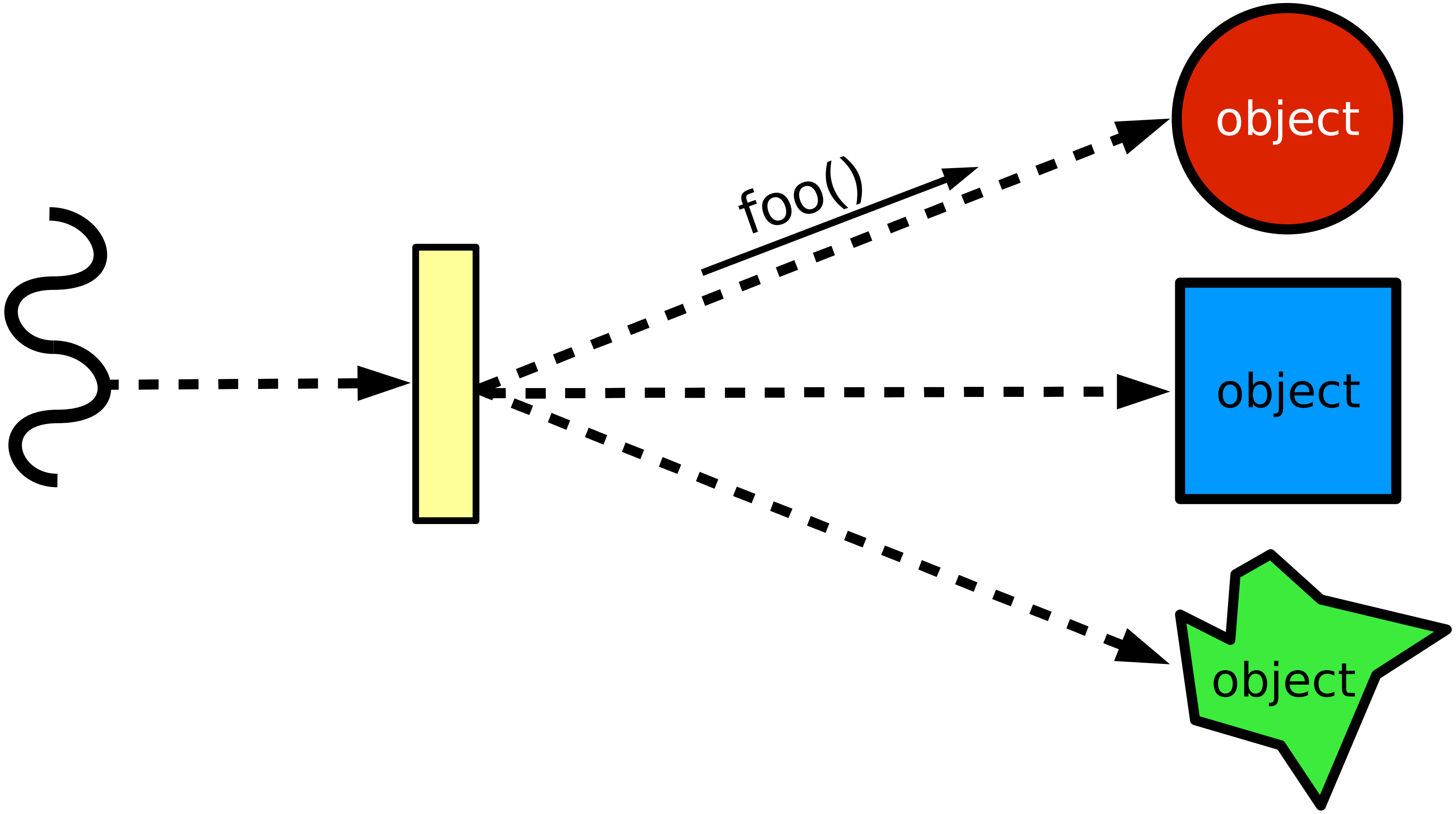
- **Single**
  - Forward to 1 object in group
- **Group**
  - Forward to all objects in group
- **Personalized**
  - Forward to all objects, but personalize parameters for each target
- **Combined**
  - Combine several invocation into one, then foward to the group using one of the above

vrije Universiteit

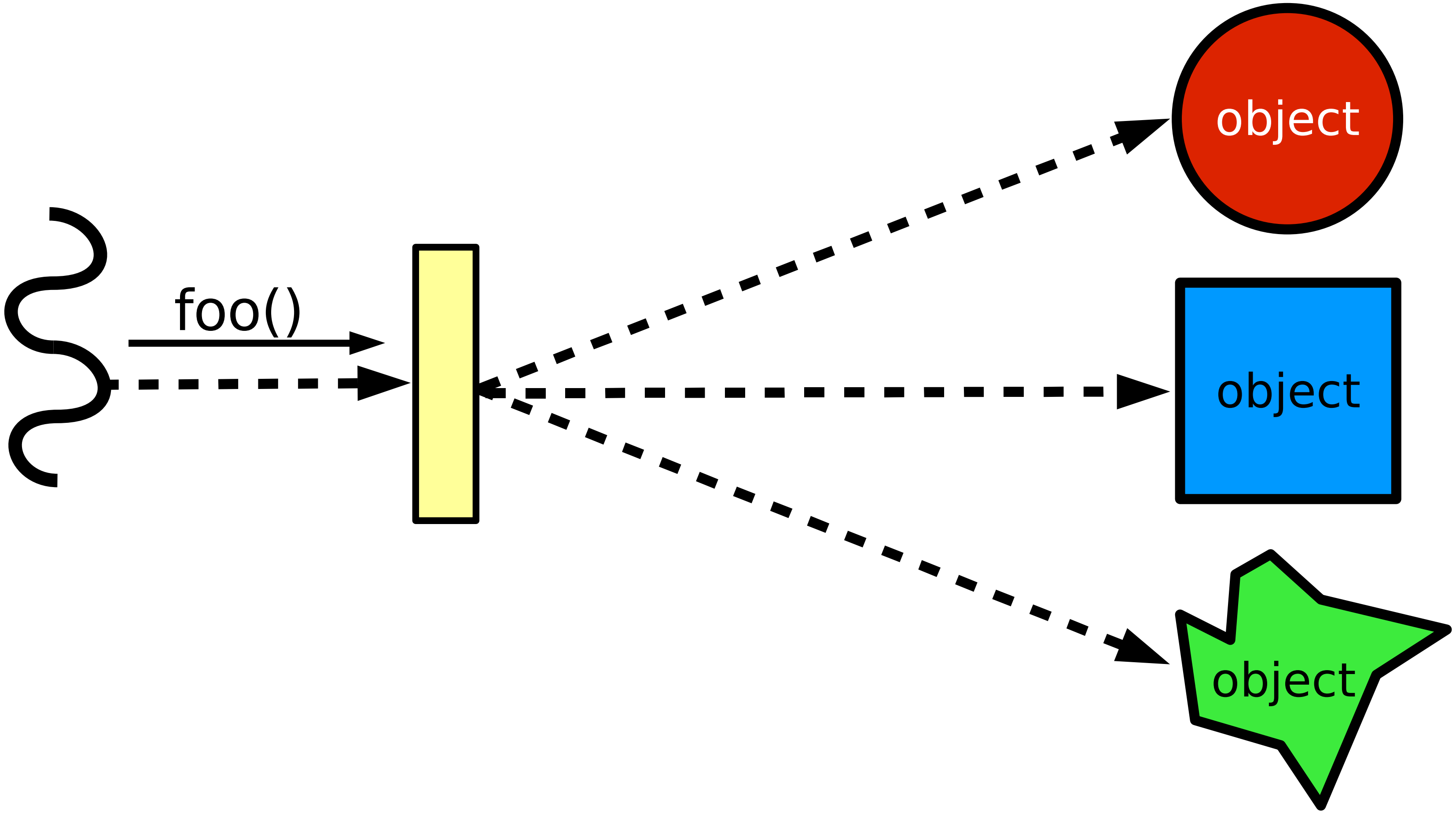
## *Single*



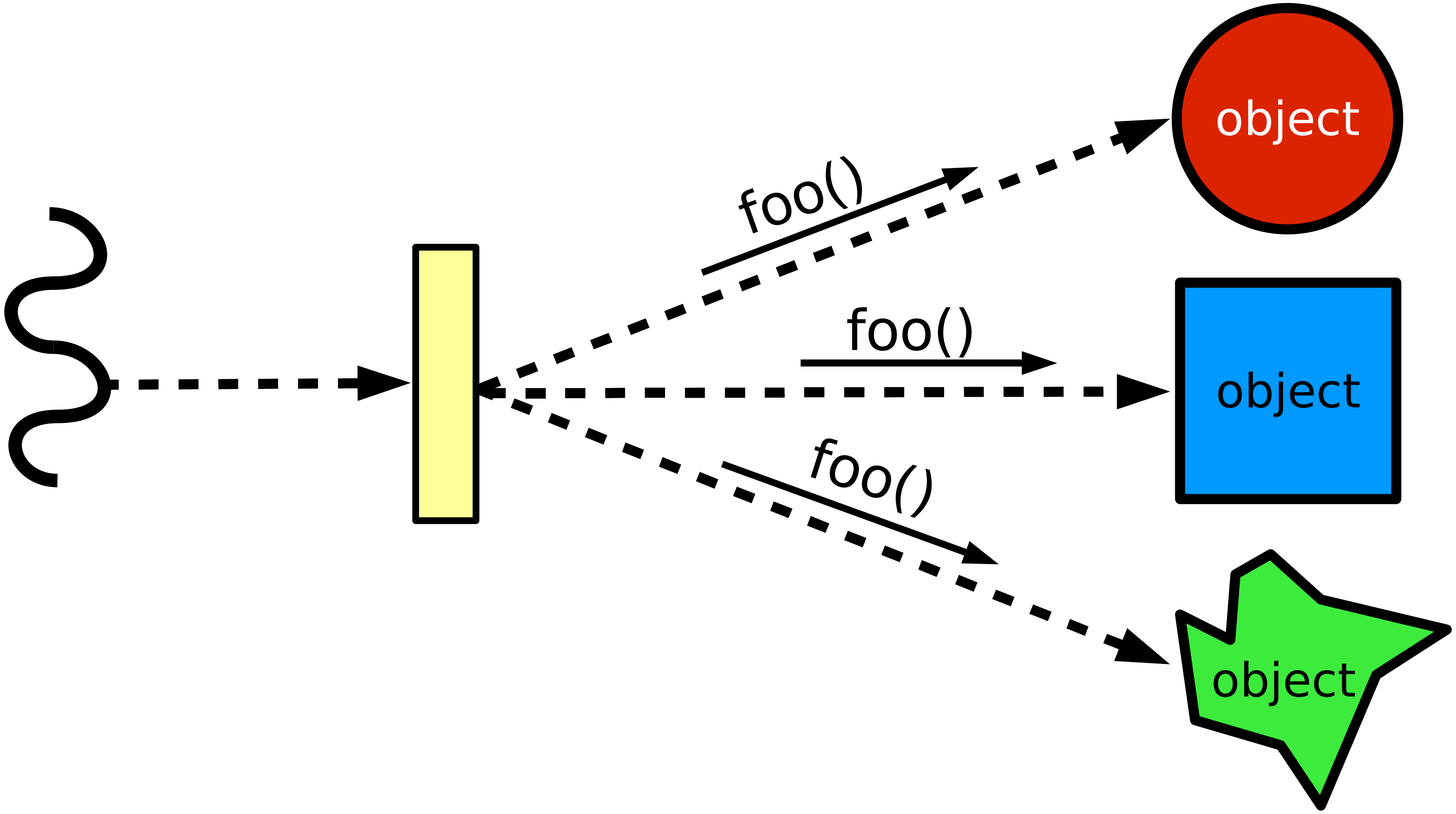
# Single



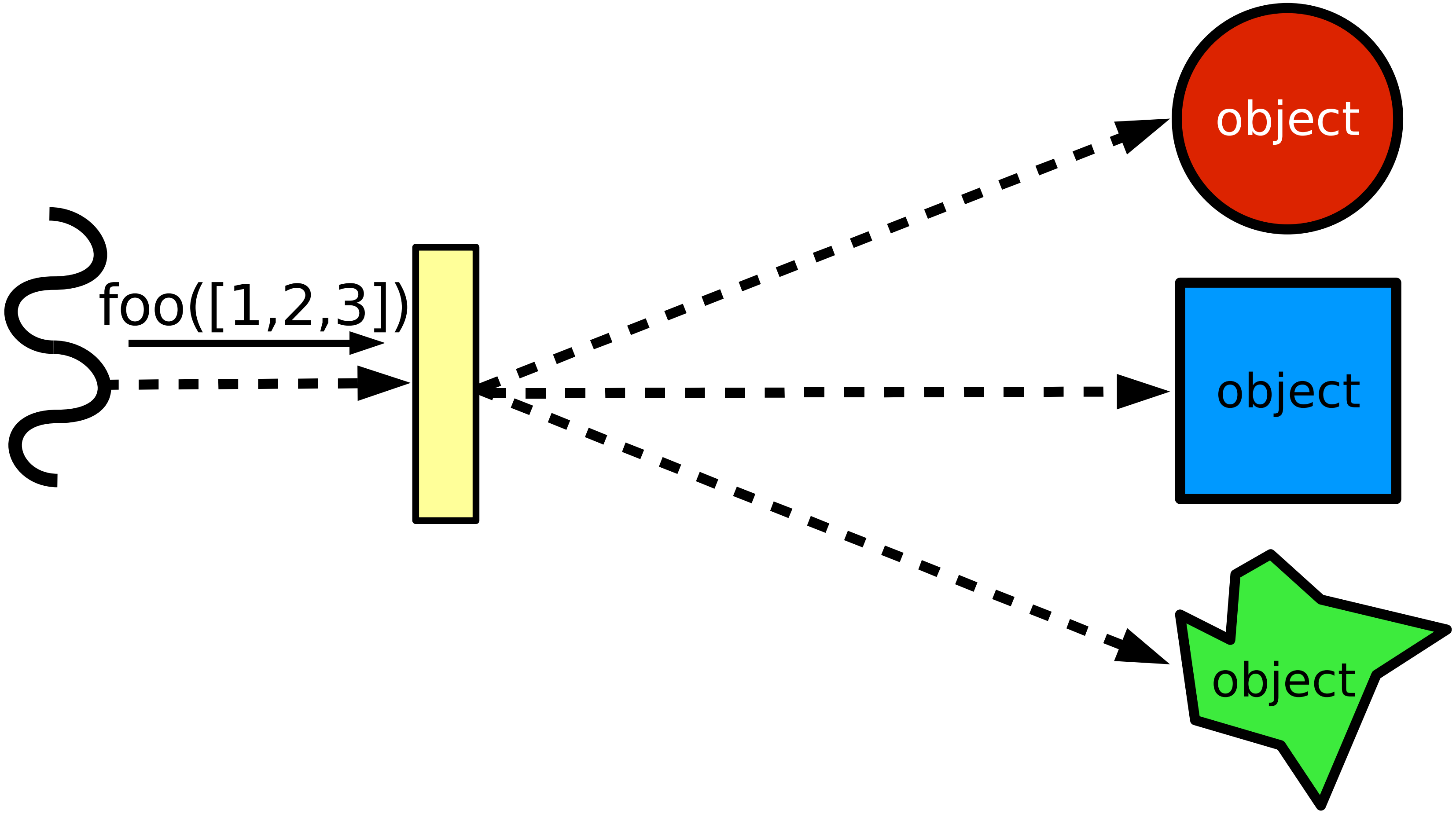
## Group



## Group

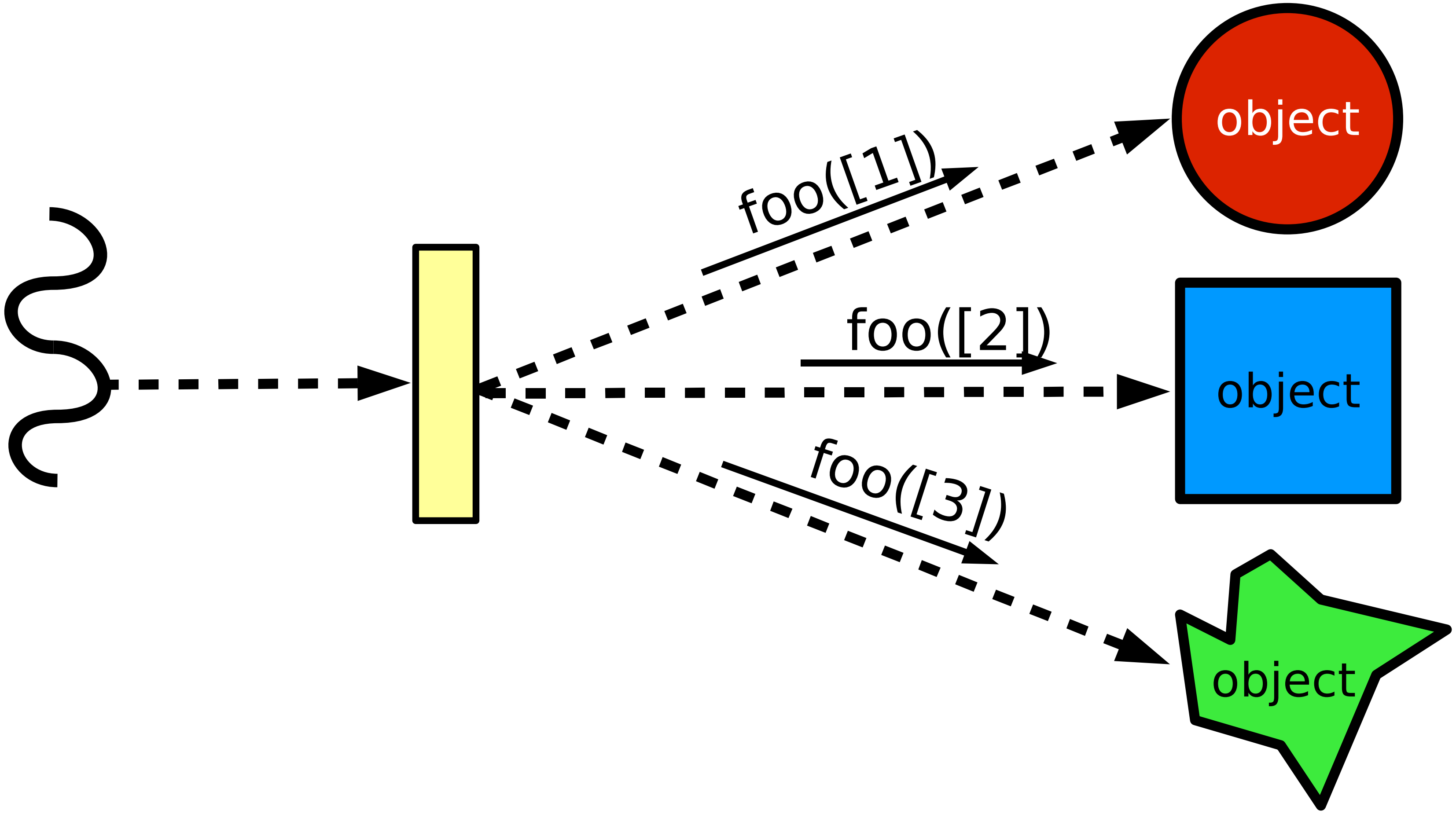


## *Personalized*

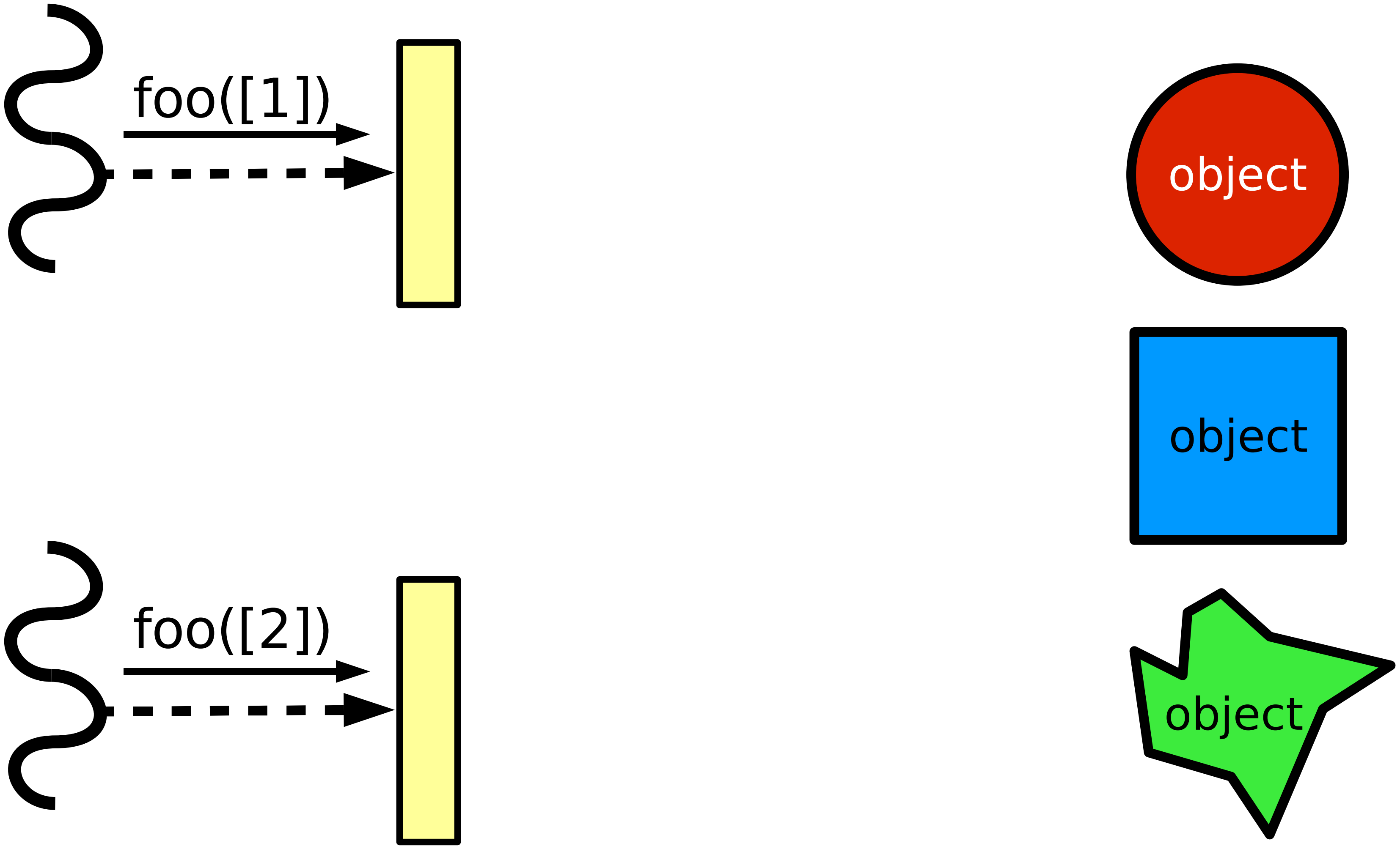


vrije Universiteit

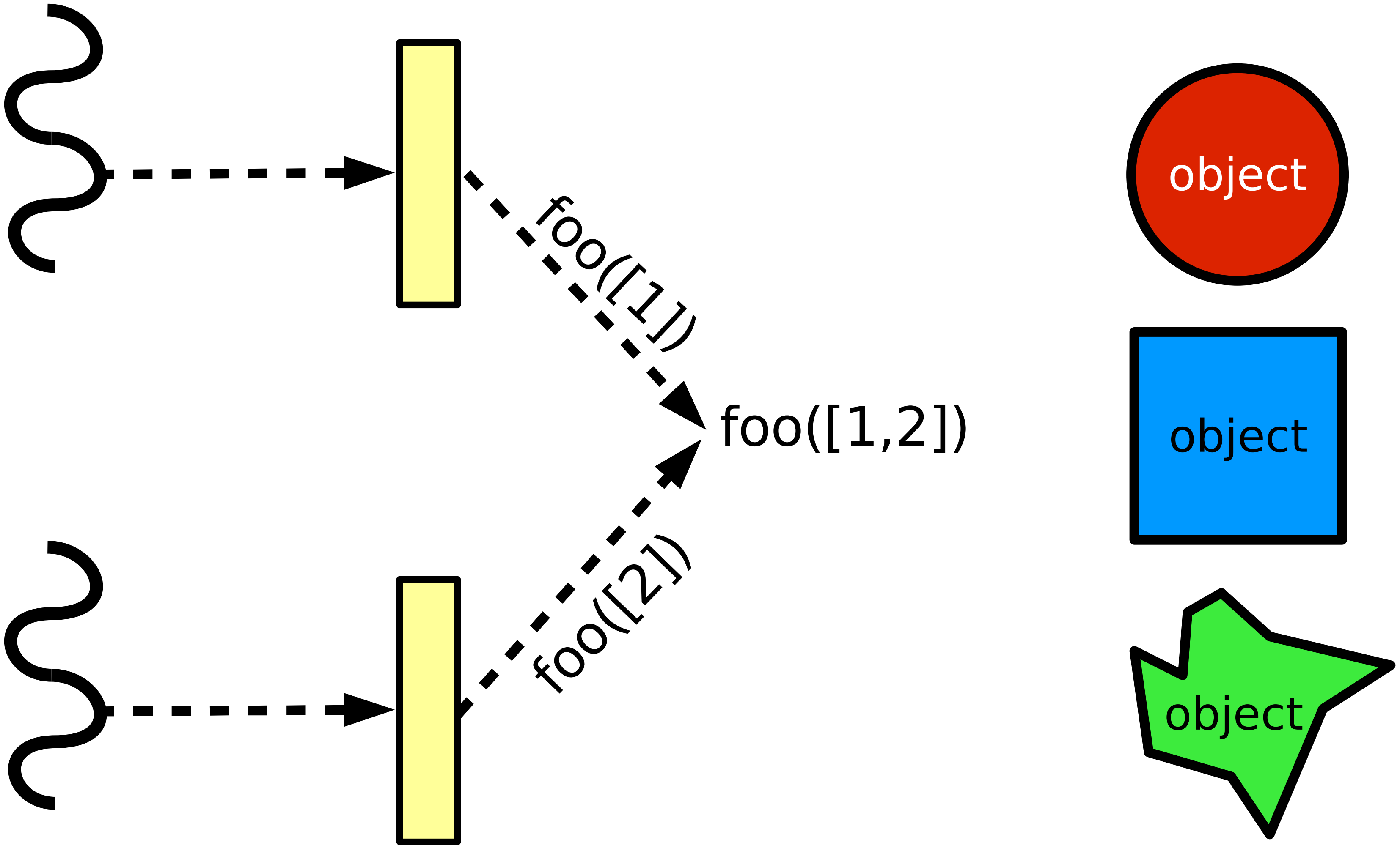
# Personalized



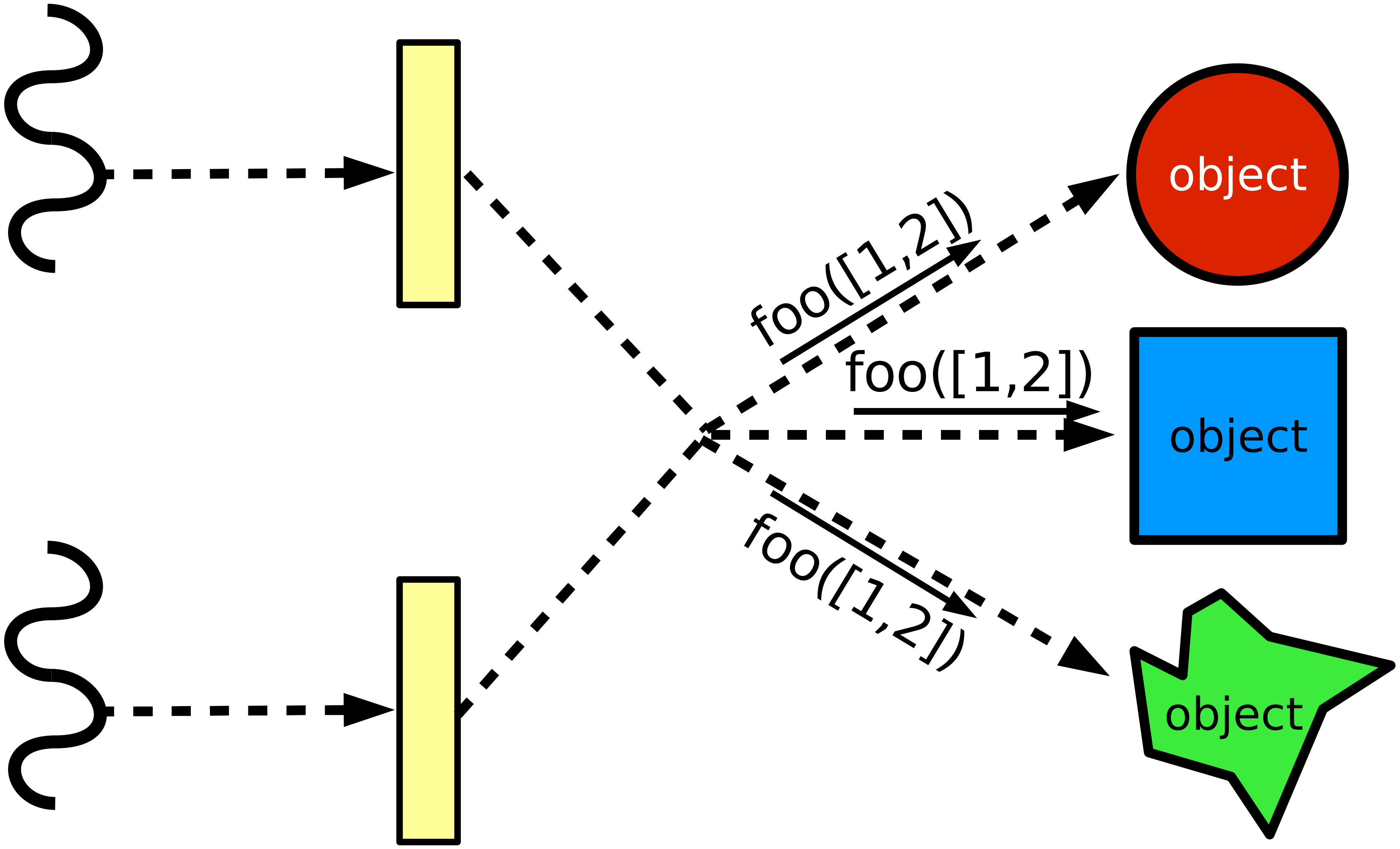
# Combined



# Combined



# Combined

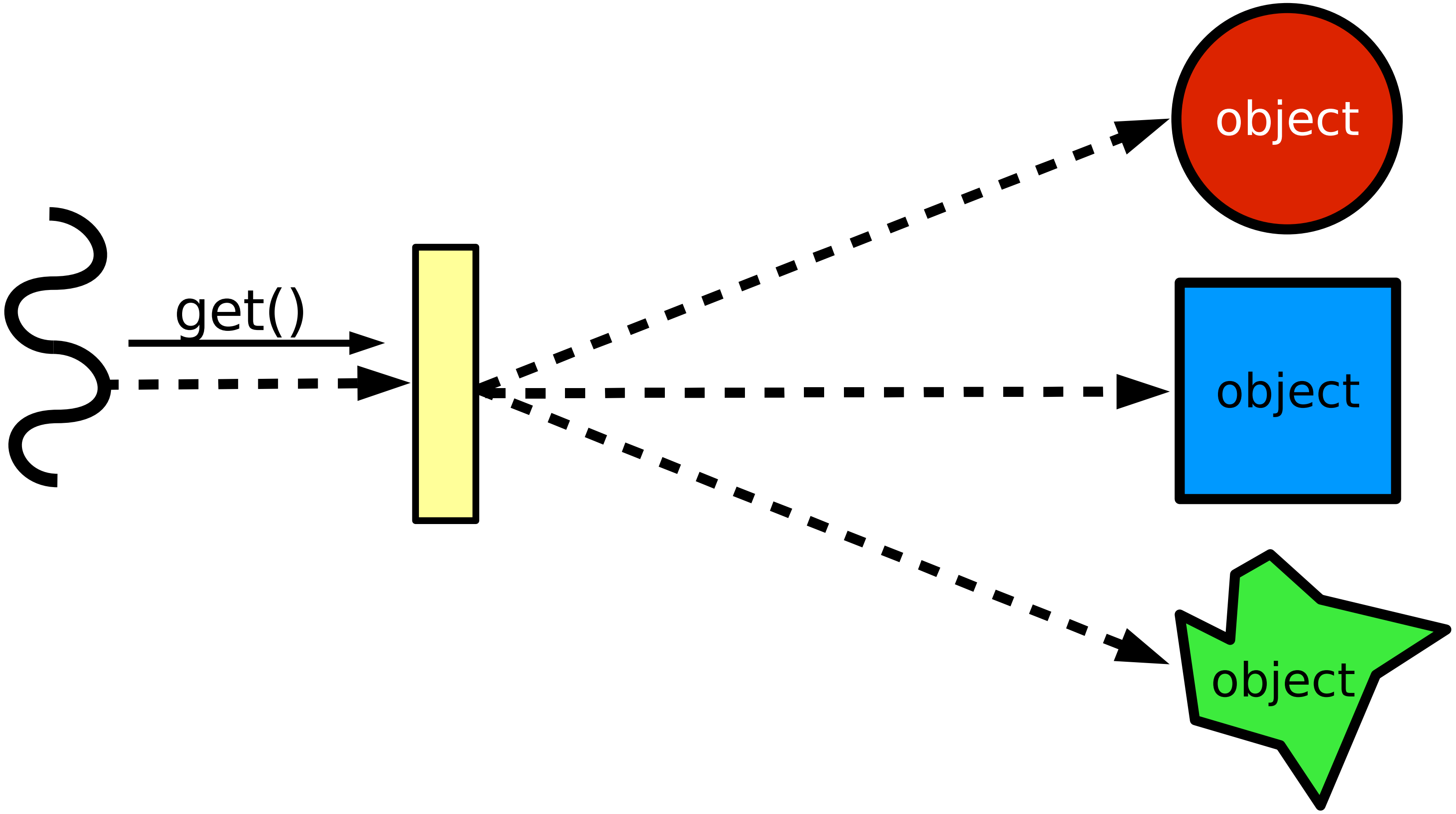


vrije Universiteit

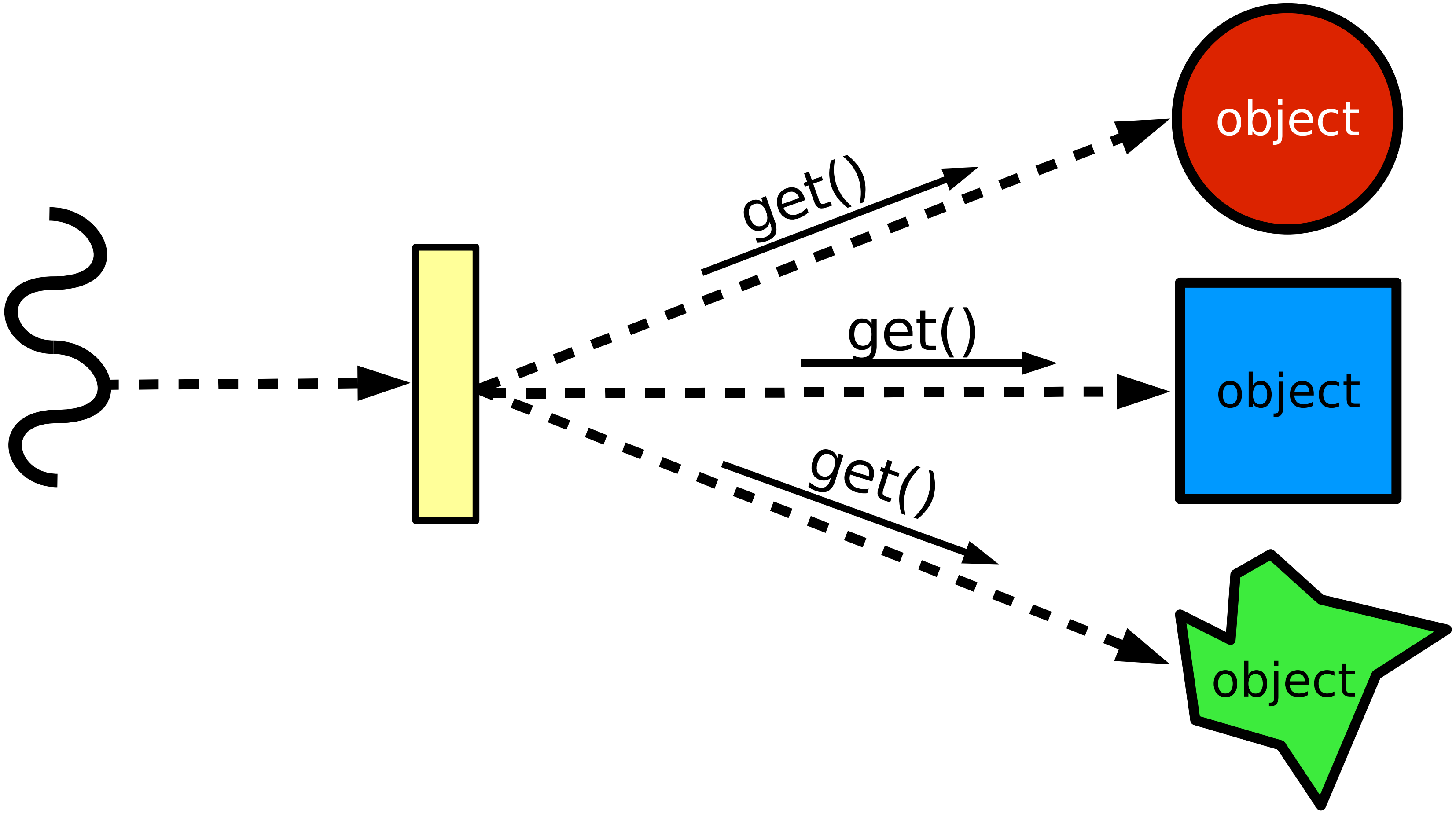
## *Reply handling schemes*

- **Discard**
- **Return**
- **Forward**
  - Reply is forwarded to a seperate object
- **Combine**
  - Multiple replies are combined into one
- **Personalize**
  - A personalized result is returned to each participant of a combined invocation

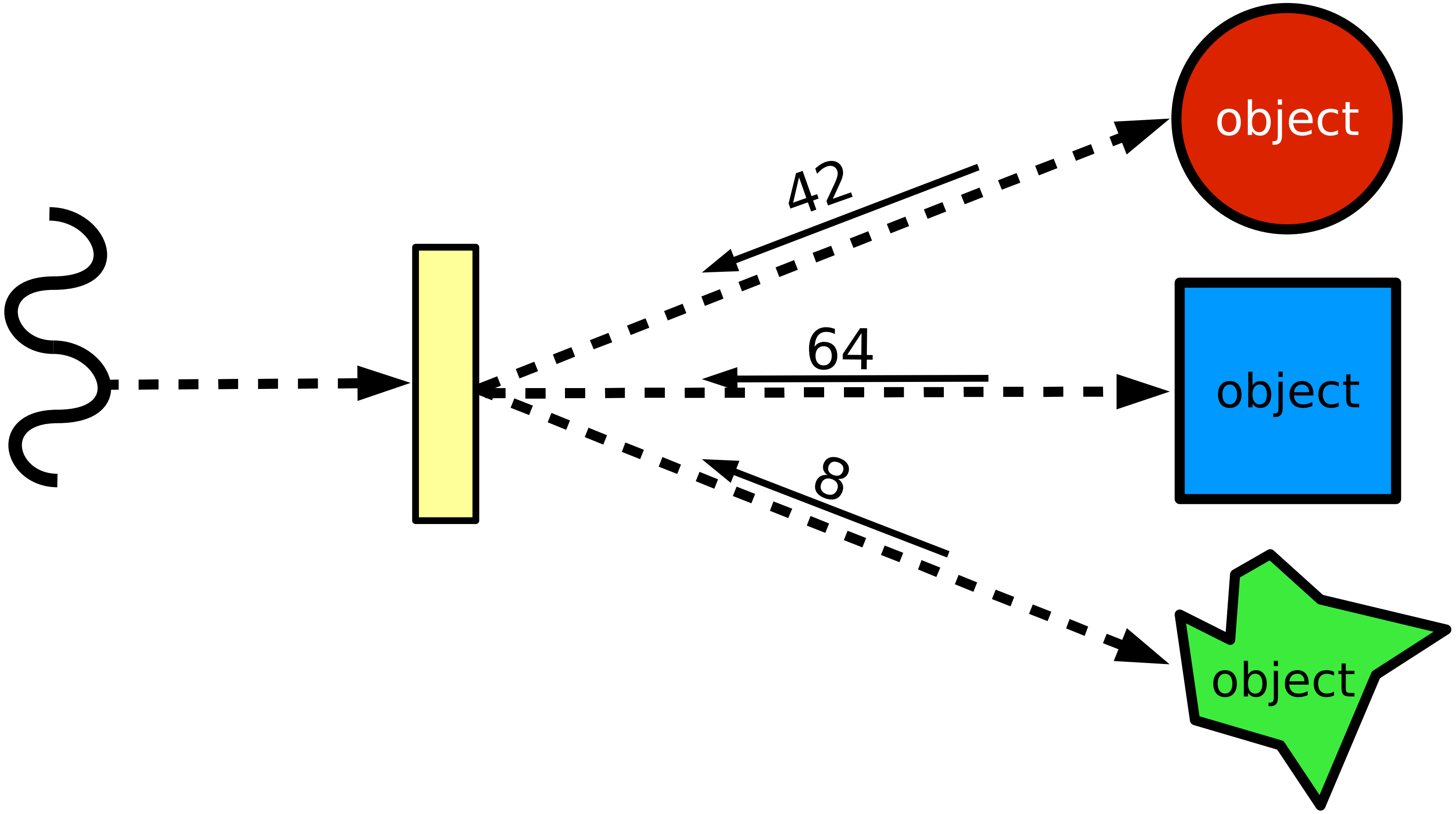
## Result Combining



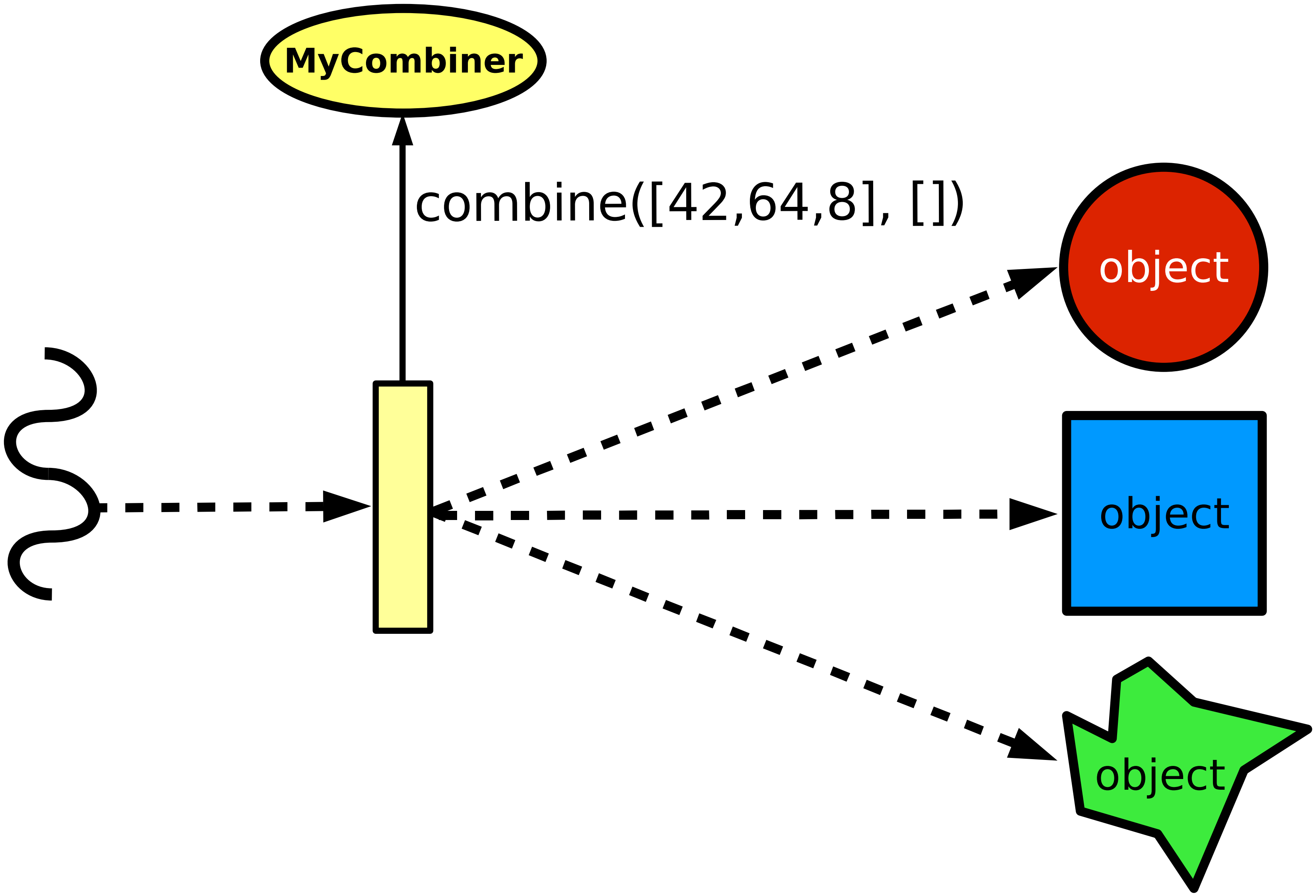
# Result Combining



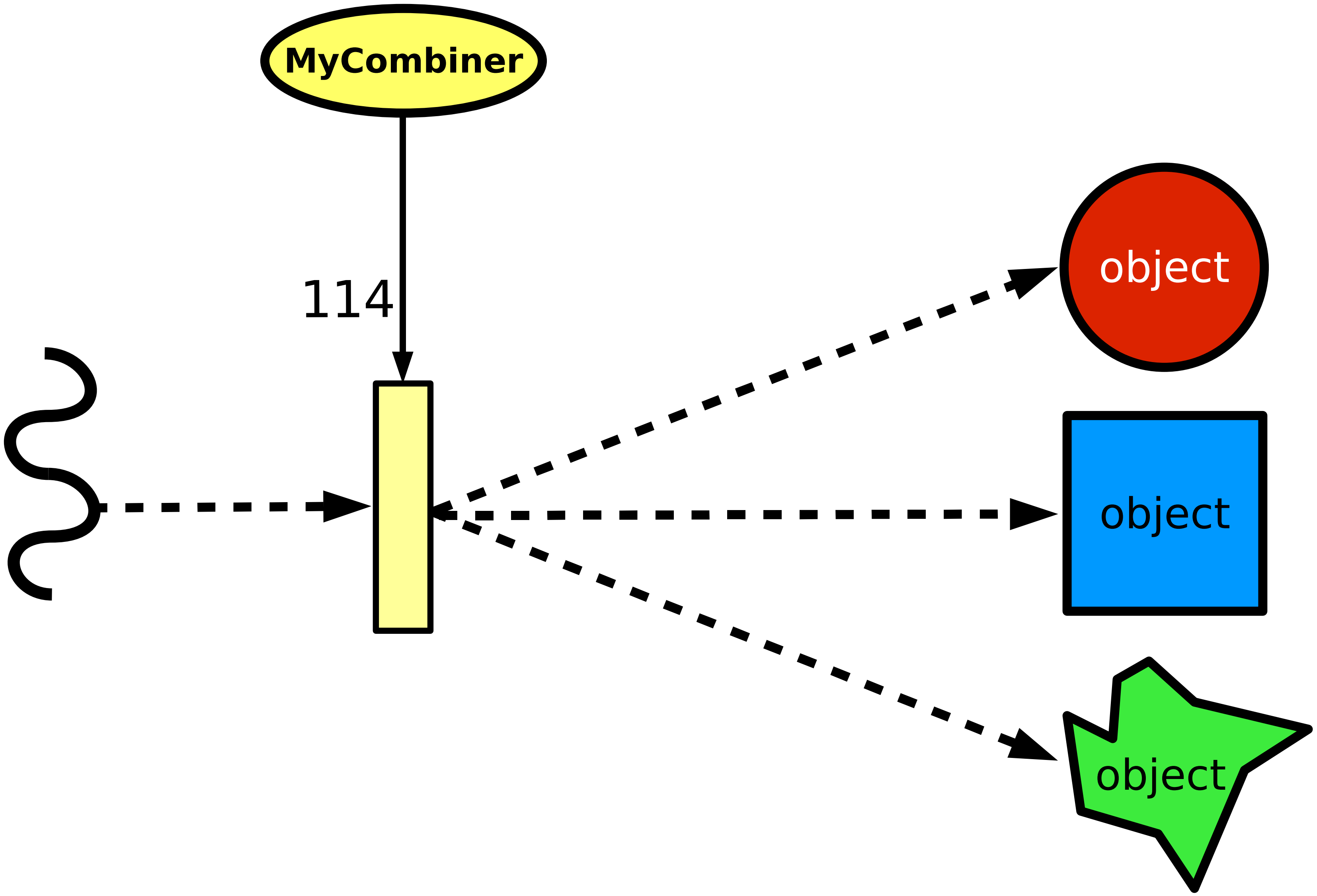
# Result Combining



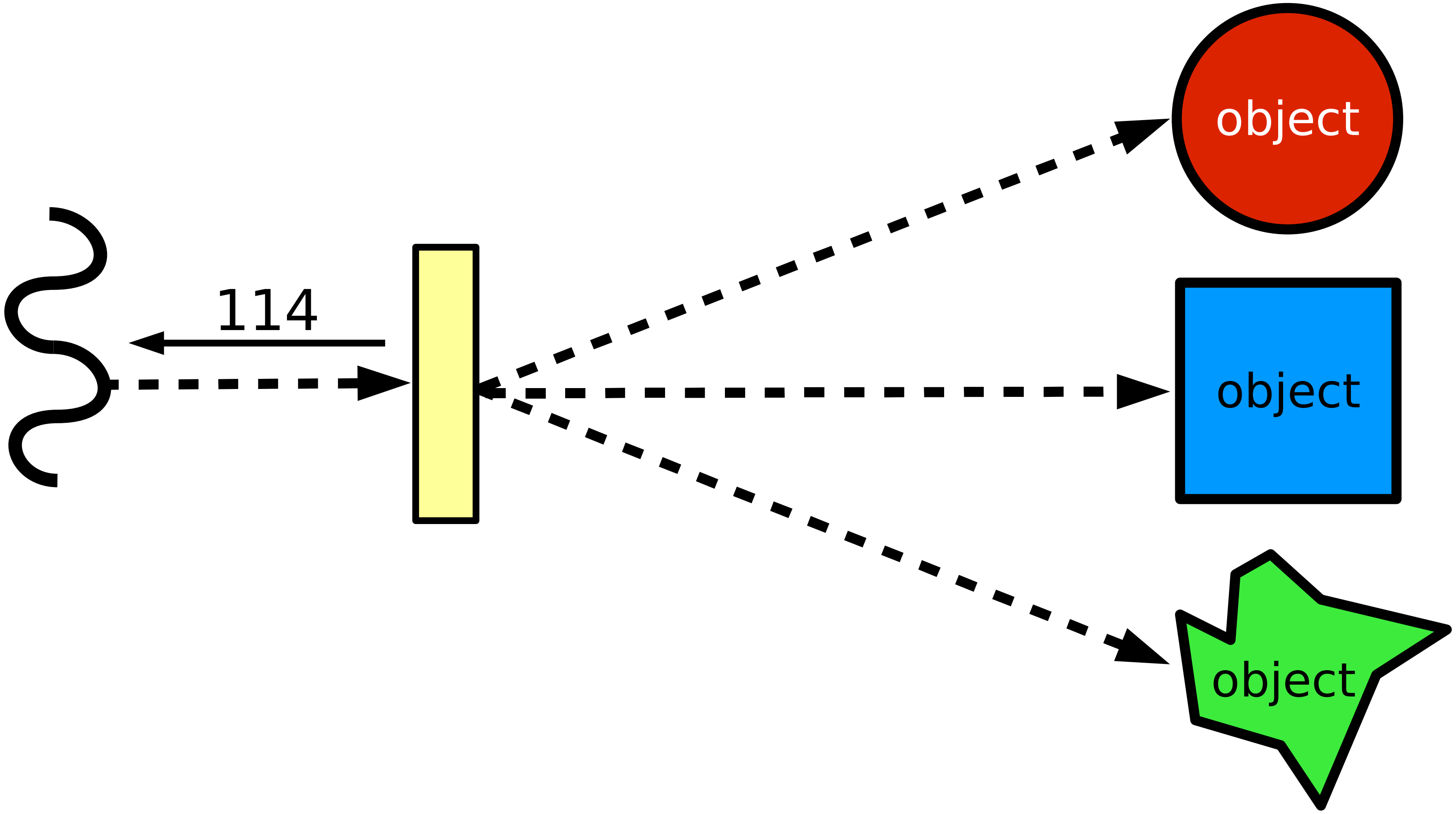
# Result Combining



# Result Combining



# Result Combining



# GMI Communication

| ***Operation*** | ***Invocation*** | ***Reply*** |
|---|---|---|
| RMI | Single | Return |
| Async. RMI | Single | Discard |
| Future | Single | Forward |
| | | |
| Broadcast | Group | Discard |
| Scatter | Personalized | Discard |
| Reduce results | Group | Combine (binomial) |
| Gather results | Group | Combine (flat) |
| | | |
| Reduce inv. | Combine + Single | Discard |
| Gather inv. | Combine + Single | Discard |

## Code example

```
public interface i_SimpleGroup extends GroupInterface {
    void ping();
}

public class SimpleGroup extends GroupMember implements i_SimpleGroup {

    public SimpleGroup() {
        super();
    }

    public void ping() {
        System.out.println("ping");
    }
}

public class MulticastNoReply {
    public static void main(String[] args) throws Exception {

        int rank = Group.rank();
        int size = Group.size();

        // Create the group
        if (rank == 0) {
            Group.create("GroupNoReply", i_SimpleGroup.class, size);
        }

        // Everyone adds an object
        SimpleGroup s = new SimpleGroup();
        Group.join("GroupNoReply", s);

        if (rank == 0) {
            // Perform lookup to get group reference
            i_SimpleGroup g = (i_SimpleGroup) Group.lookup("GroupNoReply");

            // Configure reference to perform group invocation
            GroupMethod m = Group.findMethod(g,"void ping()");
            m.configure(new GroupInvocation(), new DiscardReply());

            // Perform the invocation
            g.ping();
        }

        // Done
        Group.exit();
    }
}
```

# Code example

```
public interface i_SimpleGroup extends GroupInterface {
    void ping();
}

public class SimpleGroup extends GroupMember implements i_SimpleGroup {

    public SimpleGroup() {
        super();
    }
```

```
public interface i_SimpleGroup extends GroupInterface {
    void ping();
}
```

```
    public static void main(String[] args) throws Exception {

        int rank = Group.rank();
        int size = Group.size();

        // Create the group
        if (rank == 0) {
            Group.create("GroupNoReply", i_SimpleGroup.class, size);
        }

        // Everyone adds an object
        SimpleGroup s = new SimpleGroup();
        Group.join("GroupNoReply", s);

        if (rank == 0) {
            // Perform lookup to get group reference
            i_SimpleGroup g = (i_SimpleGroup) Group.lookup("GroupNoReply");

            // Configure reference to perform group invocation
            GroupMethod m = Group.findMethod(g,"void ping()");
            m.configure(new GroupInvocation(), new DiscardReply());

            // Perform the invocation
            g.ping();
        }

        // Done
        Group.exit();
    }
}
```

# Code example

```
public interface i_SimpleGroup extends GroupInterface {
    void ping();
}

public class SimpleGroup extends GroupMember implements i_SimpleGroup {

    public SimpleGroup() {
        super();
    }

    public void ping() {
        System.out.println("ping");
    }
}

public class MulticastNoReply {
    public static void main(String[] args) throws Exception {

        int rank = Group.rank();
        int size = Group.size();
```

```
public class SimpleGroup extends GroupMember implements i_SimpleGroup {

   public SimpleGroup() {
      super();
   }

   public void ping() {
      System.out.println("ping");
   }
}
```

```
        Group.exit();
    }
}
```

# Code example

```
public interface i_SimpleGroup extends GroupInterface {
    void ping();
}

public class SimpleGroup extends GroupMember implements i_SimpleGroup {

    public SimpleGroup() {
        super();
    }

    public void ping() {
        System.out.println("ping");
    }
}

public class MulticastNoReply {
    public static void main(String[] args) throws Exception {

        int rank = Group.rank();
        int size = Group.size();

        // Create the group
        if (rank == 0) {
            Group.create("GroupNoReply", i_SimpleGroup.class, size);
        }
```

```
public class MulticastNoReply {
   public static void main(String[] args) throws Exception {

      int rank = Group.rank();
      int size = Group.size();
```

```
            // Perform the invocation
            g.ping();
        }

        // Done
        Group.exit();
    }
}
```

# Code example

```
public interface i_SimpleGroup extends GroupInterface {
    void ping();
}

public class SimpleGroup extends GroupMember implements i_SimpleGroup {

    public SimpleGroup() {
        super();
    }

    public void ping() {
        System.out.println("ping");
    }
}

public class MulticastNoReply {
    public static void main(String[] args) throws Exception {

        int rank = Group.rank();
        int size = Group.size();

        // Create the group
        if (rank == 0) {
            Group.create("GroupNoReply", i_SimpleGroup.class, size);
        }

        // Everyone adds an object
        SimpleGroup s = new SimpleGroup();
        Group.join("GroupNoReply", s);

        if (rank == 0) {
            // Perform lookup to get group reference
```

```
// Create the group
if (rank == 0) {
    Group.create("GroupNoReply", i_SimpleGroup.class, size);
}
```

```
        // Done
        Group.exit();
    }
}
```

# Code example

```
public interface i_SimpleGroup extends GroupInterface {
    void ping();
}

public class SimpleGroup extends GroupMember implements i_SimpleGroup {

    public SimpleGroup() {
        super();
    }

    public void ping() {
        System.out.println("ping");
    }
}

public class MulticastNoReply {
    public static void main(String[] args) throws Exception {

        int rank = Group.rank();
        int size = Group.size();

        // Create the group
        if (rank == 0) {
            Group.create("GroupNoReply", i_SimpleGroup.class, size);
        }

        // Everyone adds an object
        SimpleGroup s = new SimpleGroup();
        Group.join("GroupNoReply", s);

        if (rank == 0) {
            // Perform lookup to get group reference
            i_SimpleGroup g = (i_SimpleGroup) Group.lookup("GroupNoReply");

            // Configu
            GroupMet
            m.configu

            // Perform
            g.ping();
        }

        // Done
        Group.exit();
    }
}
```

```
// Everyone adds an object
SimpleGroup s = new SimpleGroup();
Group.join("GroupNoReply", s);
```

# Code example

```
public interface i_SimpleGroup extends GroupInterface {
    void ping();
}

public class SimpleGroup extends GroupMember implements i_SimpleGroup {

    public SimpleGroup() {
```

```
if (rank == 0) {
    // Perform lookup to get group reference
    i_SimpleGroup g = (i_SimpleGroup) Group.lookup("GroupNoReply");

    // Configure reference to perform group invocation
    GroupMethod m = Group.findMethod(g,"void ping()");
    m.configure(new GroupInvocation(), new DiscardReply());

    // Perform the invocation
    g.ping();
}
```

```
        if (rank == 0) {
            // Perform lookup to get group reference
            i_SimpleGroup g = (i_SimpleGroup) Group.lookup("GroupNoReply");

            // Configure reference to perform group invocation
            GroupMethod m = Group.findMethod(g,"void ping()");
            m.configure(new GroupInvocation(), new DiscardReply());

            // Perform the invocation
            g.ping();
        }
```

```
        // Done
        Group.exit();
    }
}
```

vrije Universiteit

# Code example

```
public interface i_SimpleGroup extends GroupInterface {
    void ping();
}

public class SimpleGroup extends GroupMember implements i_SimpleGroup {

    public SimpleGroup() {
        super();
    }

    public void ping() {
        System.out.println("ping");
    }
}

public class MulticastNoReply {
    public static void main(String[] args) throws Exception {

        int rank = Group.rank();
        int size = Group.size();

        // Create the group
        if (rank == 0) {
            Group.create("GroupNoReply", i_SimpleGroup.class, size);
        }

        // Everyone adds an object
        SimpleGroup s = new SimpleGroup();
        Group.join("GroupNoReply", s);

        if (rank == 0) {
            // Perform lookup to get group reference
            i_SimpleGroup g = (i_SimpleGroup) Group.lookup("GroupNoReply");

            // Configure reference to perform group invocation
            GroupMethod m = Group.findMethod(g,"void ping()");
            m.configure(new GroupInvocation(), new DiscardReply());

            // Perform the invocation
            g.ping();
        }

        // Done
        Group.exit();
    }
}
```

```
// Done
Group.exit();
```

## Code Example

- Live demo

## GMI

- MPI-style programming
- But not necessarily SPMD
  - Can also do client-server style applications
- Does not have Grid optimizations yet
  - Can be done
- Fault-tolerance support is harder
  - Model isn't as 'clean' as Satin

## *After the Break*

- Hands-on session
  - Installing Ibis.
  - Running applications
  - Writing your own applications

***END***

## *Function Objects*

- Some operations need user defined functions
  - Personalizing a method invocation
  - Combining a result or invocation
  - Forwarding of results
- GMI uses function objects
  - Extend a class from the GMI package

## *Result Combiners*

- Use 'combiner' to merge the results of an invocation
- FlatCombiner
  - Combines all results in one go
  - Similar to 'gather' operation of MPI
- BinomialCombiner
  - Pairwise combines results
  - Similar to 'reduce' operation of MPI

# FlatCombiner

```
public class FlatCombiner {

    public boolean combine(boolean[] results, Exception[] ex)
    public byte combine(byte[] results, Exception[] ex)
    public char combine(char[] results, Exception[] ex)
    public short combine(short[] results, Exception[] ex)
    public int combine(int[] results, Exception[] ex)
    public long combine(long[] results, Exception[] ex)
    public float combine(float[] results, Exception[] ex)
    public double combine(double[] results, Exception[] ex)

    public Object combine(Object[] results, Exception[] ex)

    public void combine(Exception[] exceptions)
}
```

## FlatCombiner

```
public class MyCombiner extends FlatCombiner {

   public int combine(int[] results, Exception[] ex) {

      int sum = 0;

      for (int i=0;i<results.length;i++) {
         sum += results[i];
      }

      return sum;
   }
}
```

# FlatCombiner

```
// Get a group reference
X g = (X) Group.lookup("your group");


// Configure reference to perform group invocation,
// and combine the replies using 'MyCombiner'
GroupMethod m = Group.findMethod(g, "int get()");
m.configure(new GroupInvocation(),
        new CombineReply(new MyCombiner()));


// Perform the invocation
int result = g.get();
```

vrije Universiteit

## FlatCombiner Demo

- Live demo

## *Overview*

- Programming models
  - IPL (bare bones)
  - RMI (remote invocation)
  - GMI (group communication)
  - Satin (divide and conquer)
  - MPJ (MPI to Java binding)
- Hands-on session
  - How to roll your own Ibis applications

## *Connection setup*

- Ibis- and ReceivePortIdentifiers
  - Hide implementation details
  - Independent of
    - IP-addresses
    - Host names
    - Port numbers
    - MPI-ranks, etc...

  - Abstract way of addressing machines and connection endpoints

## *GMI*

- Group
  - Contains 1 or more objects
    - Fixed size (set when it is created)
  - All objects must implement the same group interface
    - But objects may have different type!
  - Unique name
    - Used in lookup (produces group reference)
  - Group members have rank
    - Ranks are 'per-group'