Ibis as Glue

Jason Maassen

Computer Systems Group
Department of Computer Science
VU University, Amsterdam, The Netherlands

Short recap...

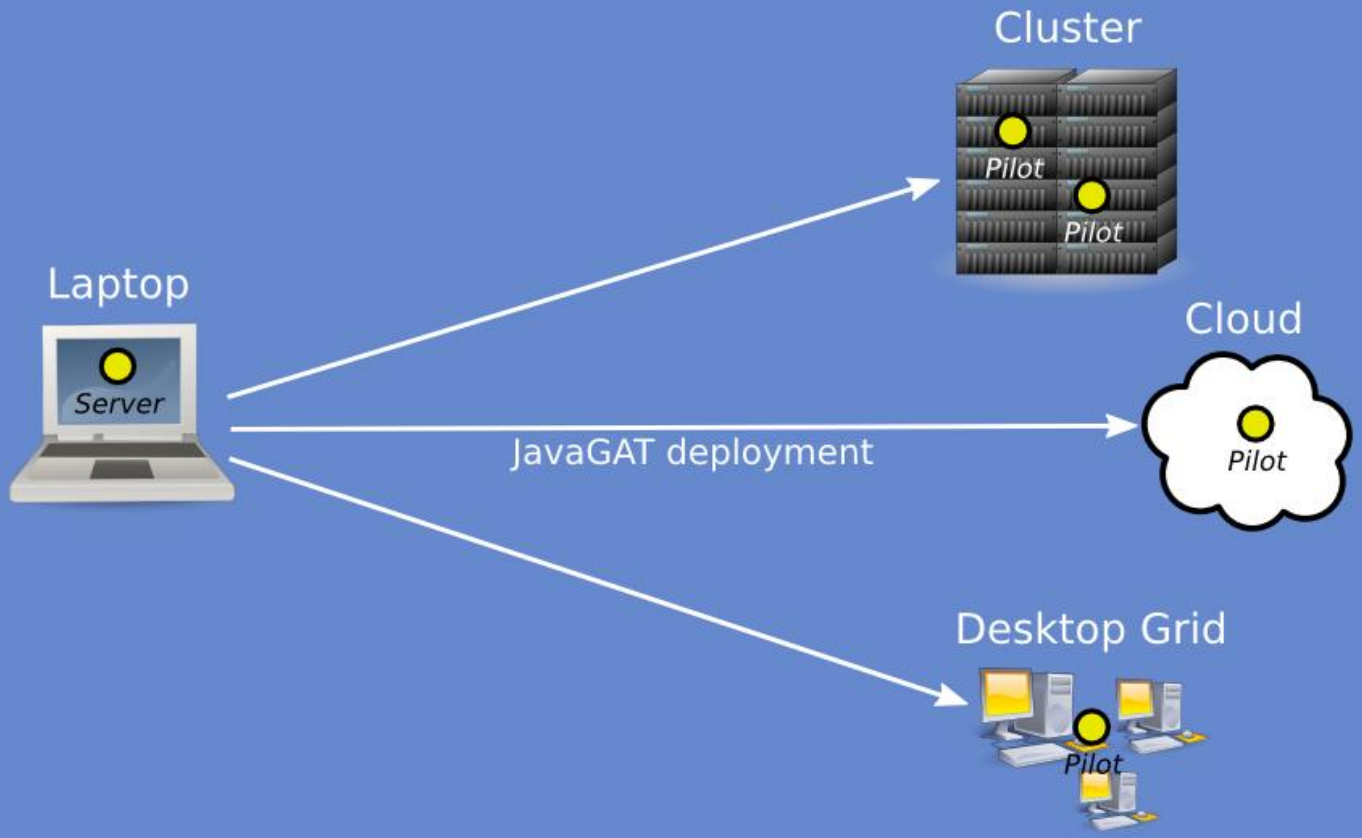
- JavaGAT acts as master key (or passepartout):
  - Able to access many different types of resources.
    - List of supported middleware is still increasing!
  - Abstracts away from (most) details of the middleware.
    - Improves portability of applications!
  - Simple and easy to use API.

- Easy way to deploy applications.

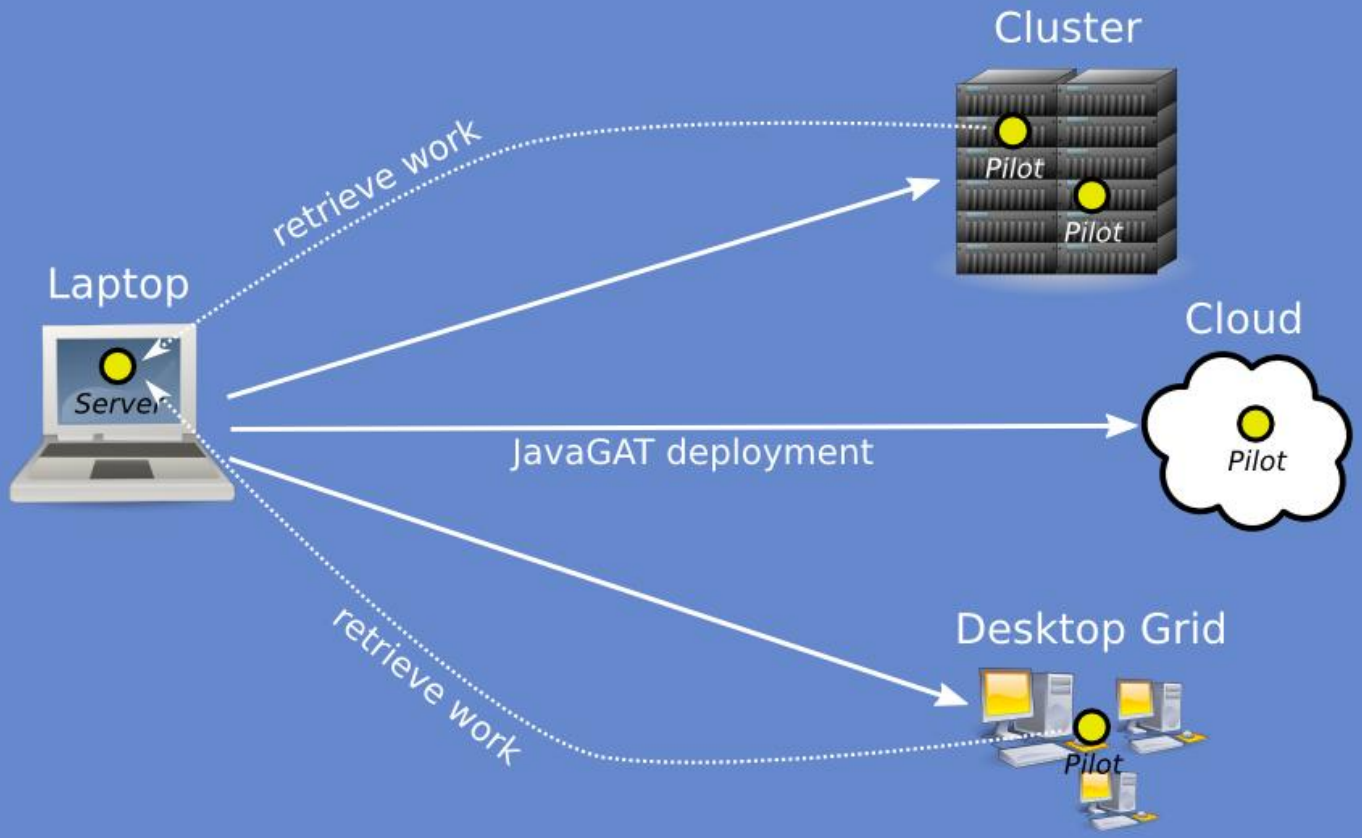
However...

- With many jobs performance may be an issue:
  - Each job is submitted individually
  - Expensive with small jobs and long queueing times
- Solution: pilot jobs (many systems exists)
  - First acquire resources by submitting a generic job
  - Then use own software infrastructure to run jobs on these resources, bypassing the queues alltogether
  - Decouples workload submission from resource selection and job execution

Example (pilot jobs)



Example (pilot jobs)

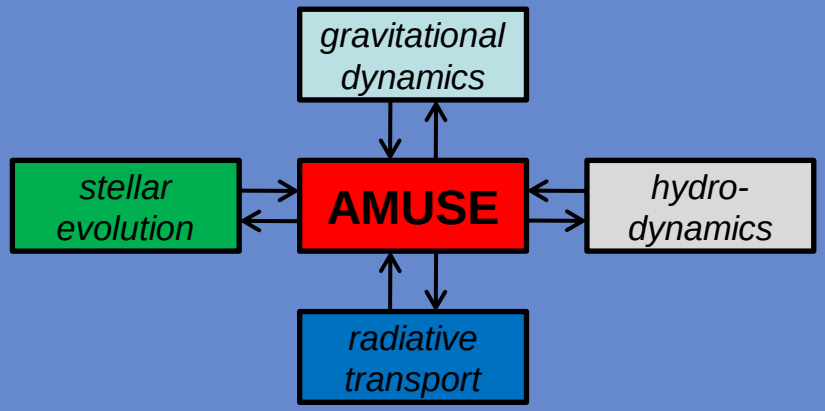


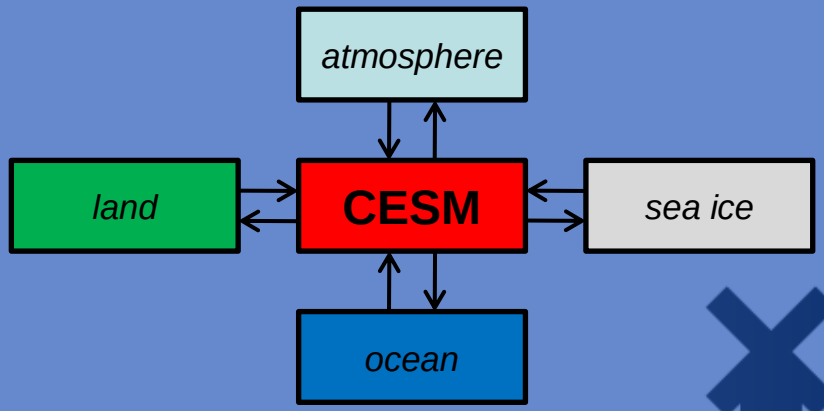
Pilot Job

- Need communication between resources!
- Nothing new about pilot jobs
  - Many frameworks exists
- However: it is a good example to illustrate the "Ibis as glue" part of our tutorial!
  - Glue a jungle of resources into a single resource pool

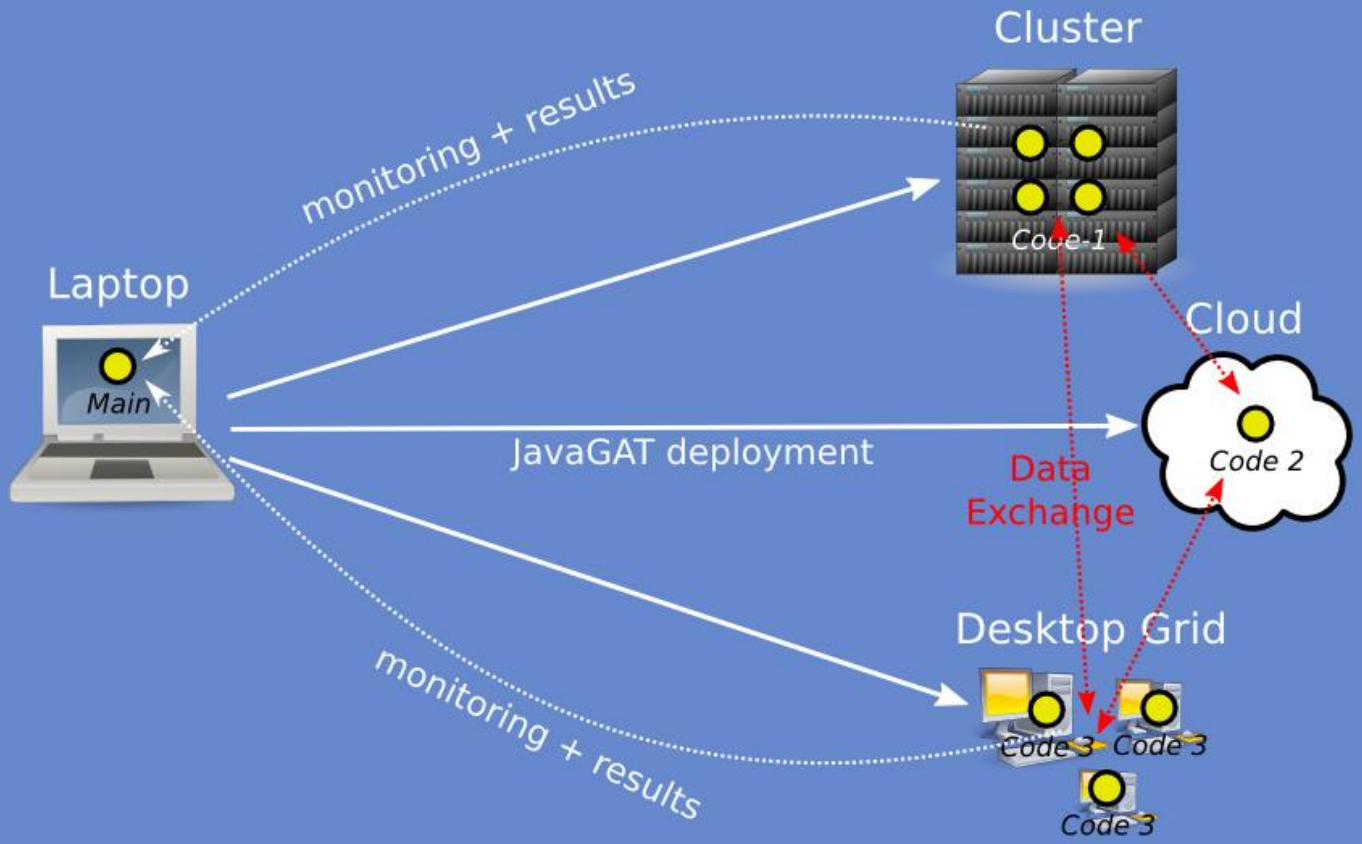
In addition ...

- Not all applications use task farming!
- Many recent applications require multiple codes to run concurrently on a collection of resources
  - Coupled codes
  - AMUSE (astrophysics), CESM (climate modelling), ...





Example (coupled codes)



We need communication!

- Both pilot jobs and coupled codes need communication between resources
  - Connecting clusters, grids, clouds, … is hard!
- Many sites have connectivity issues
  - Firewalls
  - Network Address Translation (NAT)
  - Non-routed networks
  - Multi homing
  - Mis-configured machines
  - ...

Existing communication libraries

- Sockets is too low-level for daily use
  - Only point-to-point
  - No resource management
- MPI is too inflexible
  - Focus on SPMD model
  - Little support for malleability or fault tolerance
  - Hard to use in heterogeneous environments
- Neither can handle firewalls/NAT/etc.

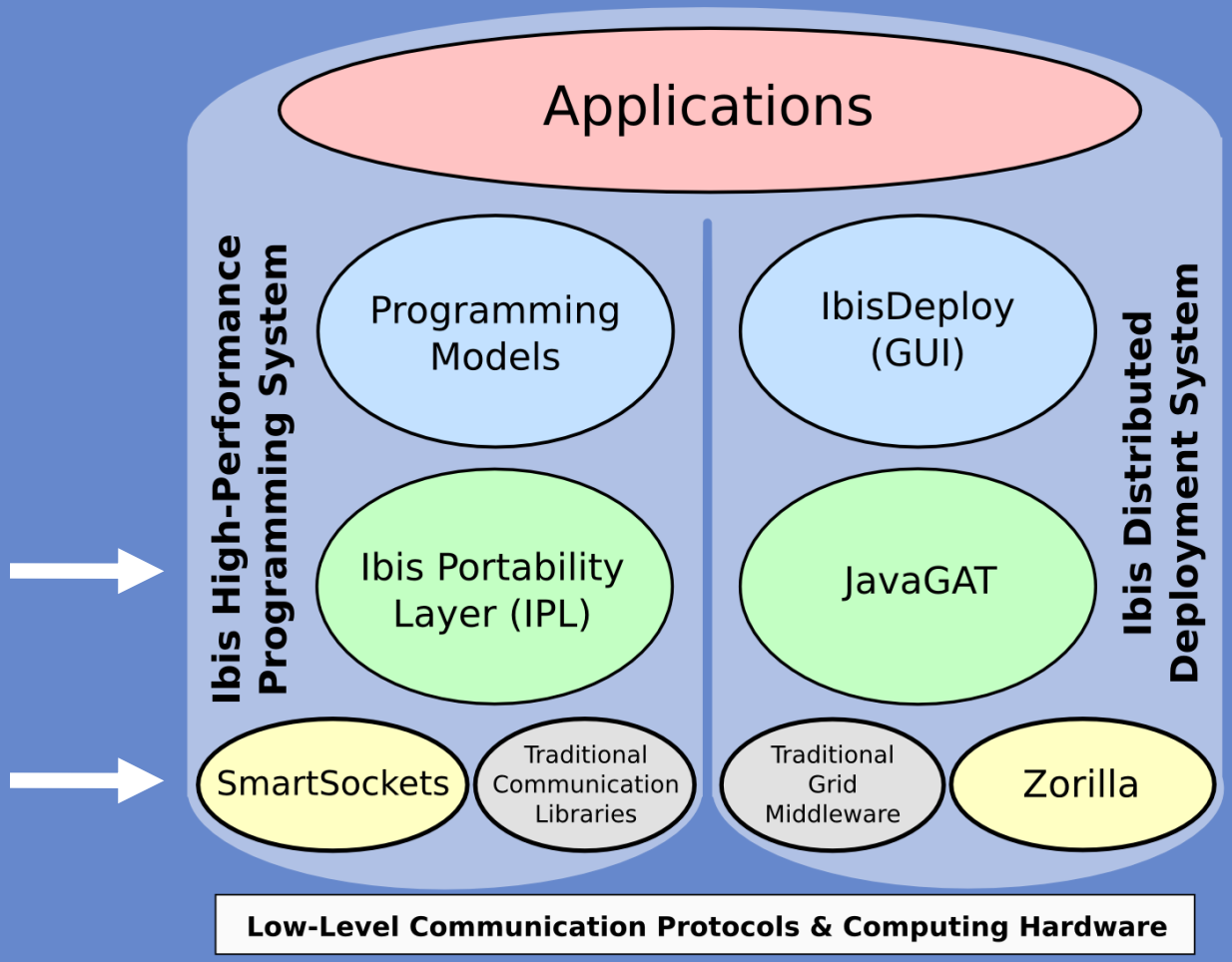
What do we need

- Something that solves the connectivity issues
  - Less help from the user is better!
- Better resource tracking
  - Malleability: resources come and go
  - Fault Tolerance: resources may crash at any time
  - Robust and globally unique naming
- Flexible communication primitives
  - Multicast or many-to-one communication
  - Efficient serialization of complex data structures

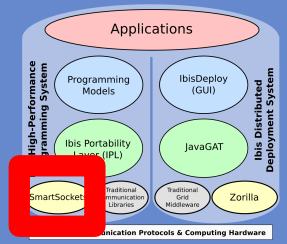
Ibis

- Ibis offers “Jungle proof” communication:
  - SmartSockets
    - Sockets library (on top of regular TCP/IP)
    - Solves low-level connectivity problems
  - Ibis Portability Layer (IPL)
    - Offers high-level communication primitives and resource tracking

Where are we ?


Applications
Ibis High-Performance Programming System
Programming Models
Ibis Portability Layer (IPL)
SmartSockets
Traditional Communication Libraries
IbisDeploy (GUI)
JavaGAT
Traditional Grid Middleware
Zorilla
Ibis Distributed Deployment System
Low-Level Communication Protocols & Computing Hardware

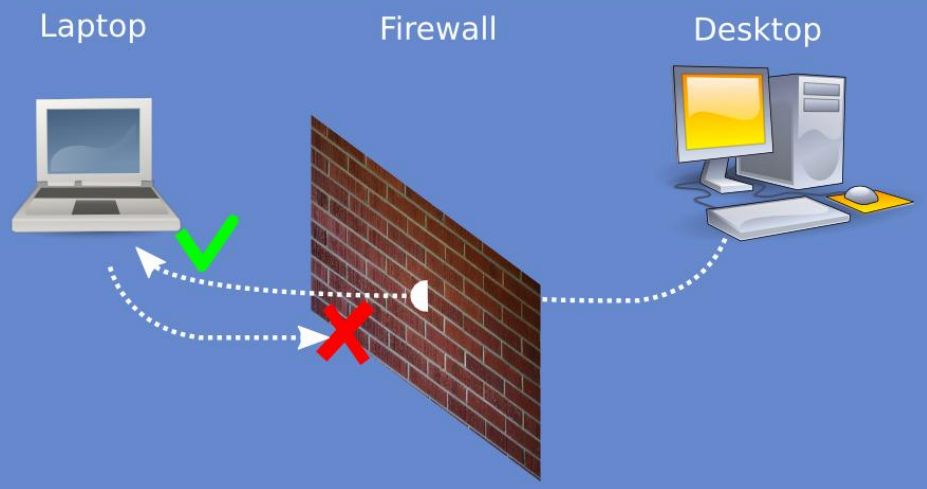
SmartSockets
What problems does it solve ?



- Unreachable machines:
  - Behind firewall / NAT or on private network
- Machine identification:
  - Machines have multiple IPs
  - Multiple machines have the same (private) IP

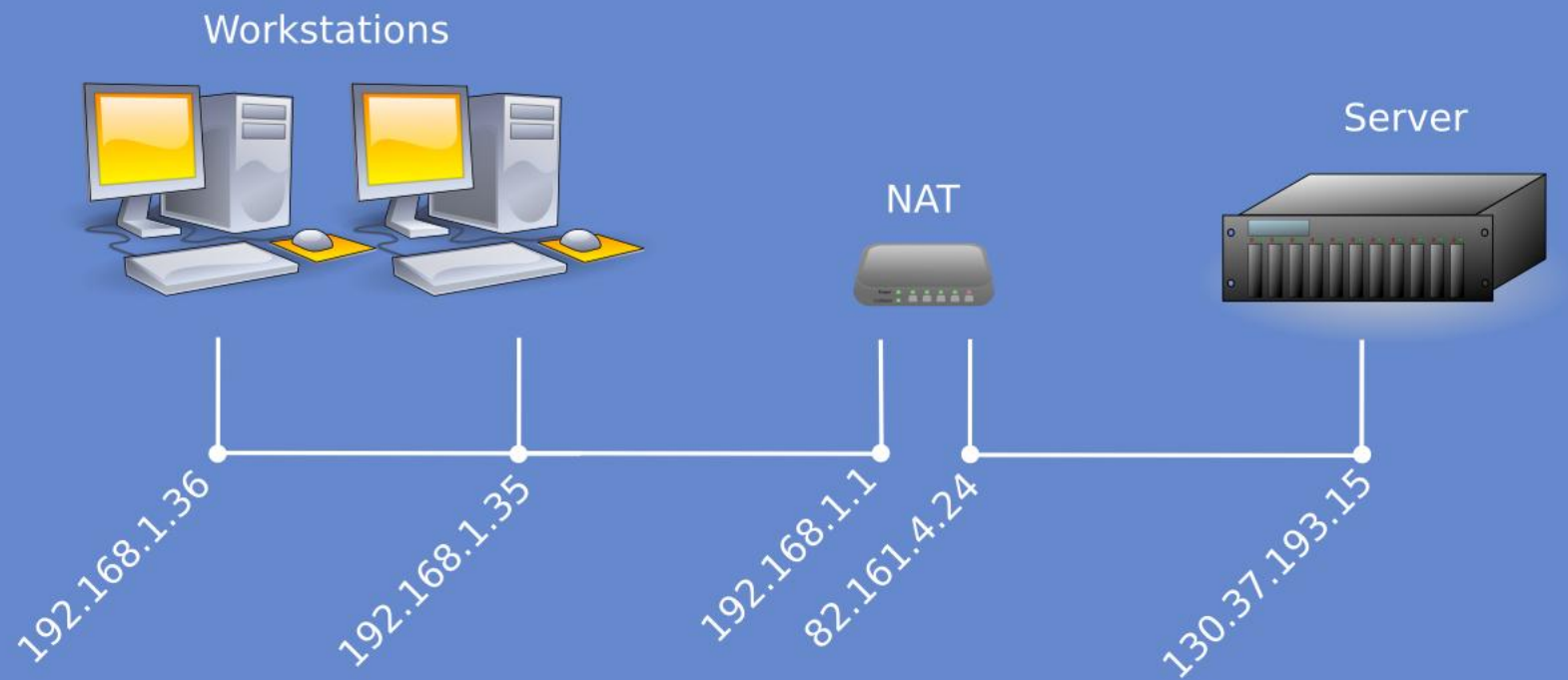
Problem 1: Firewalls

- Blocks 'inappropriate' connections
  - Usually only blocks incoming connections
  - Some also block outgoing connection

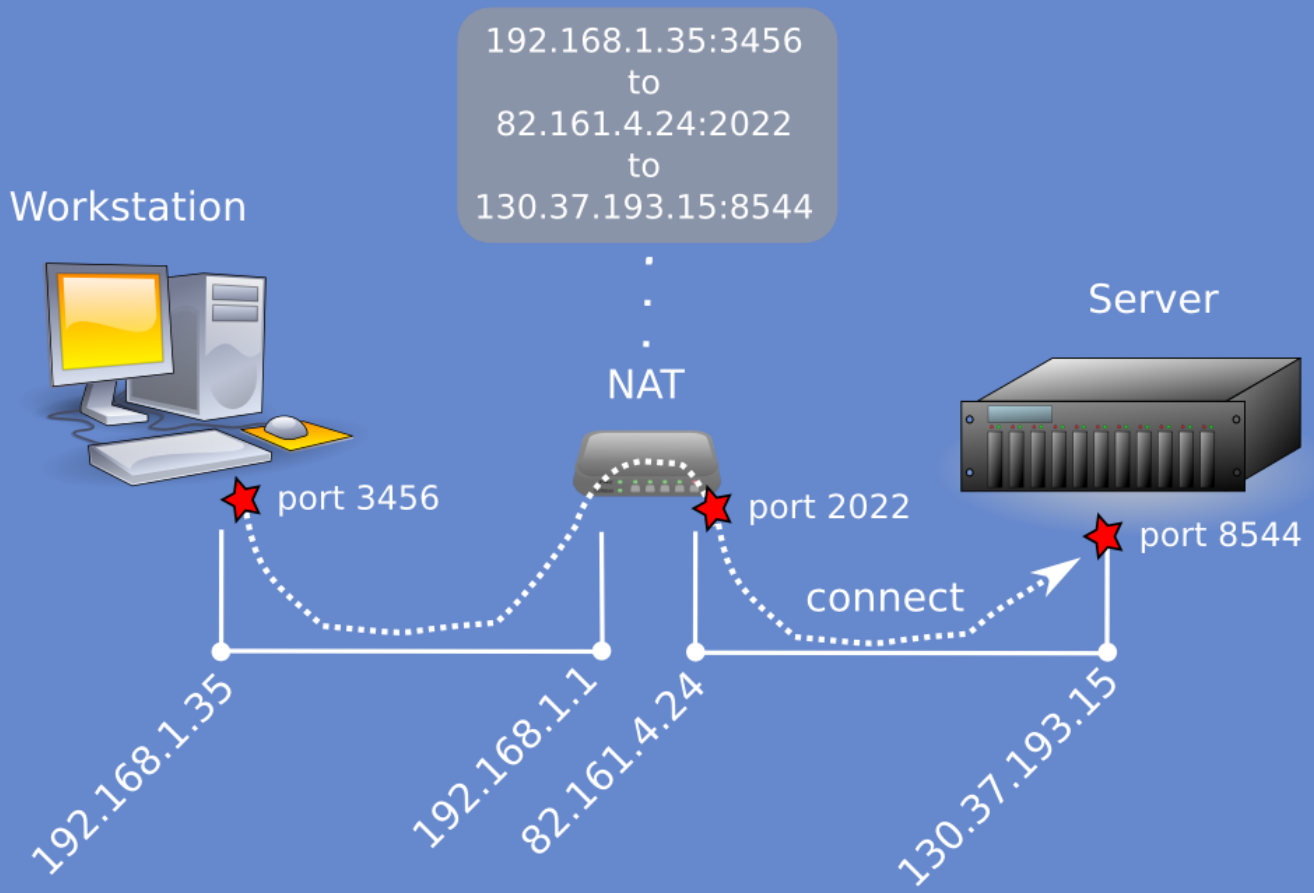


Problem 2:
Network Address Translation

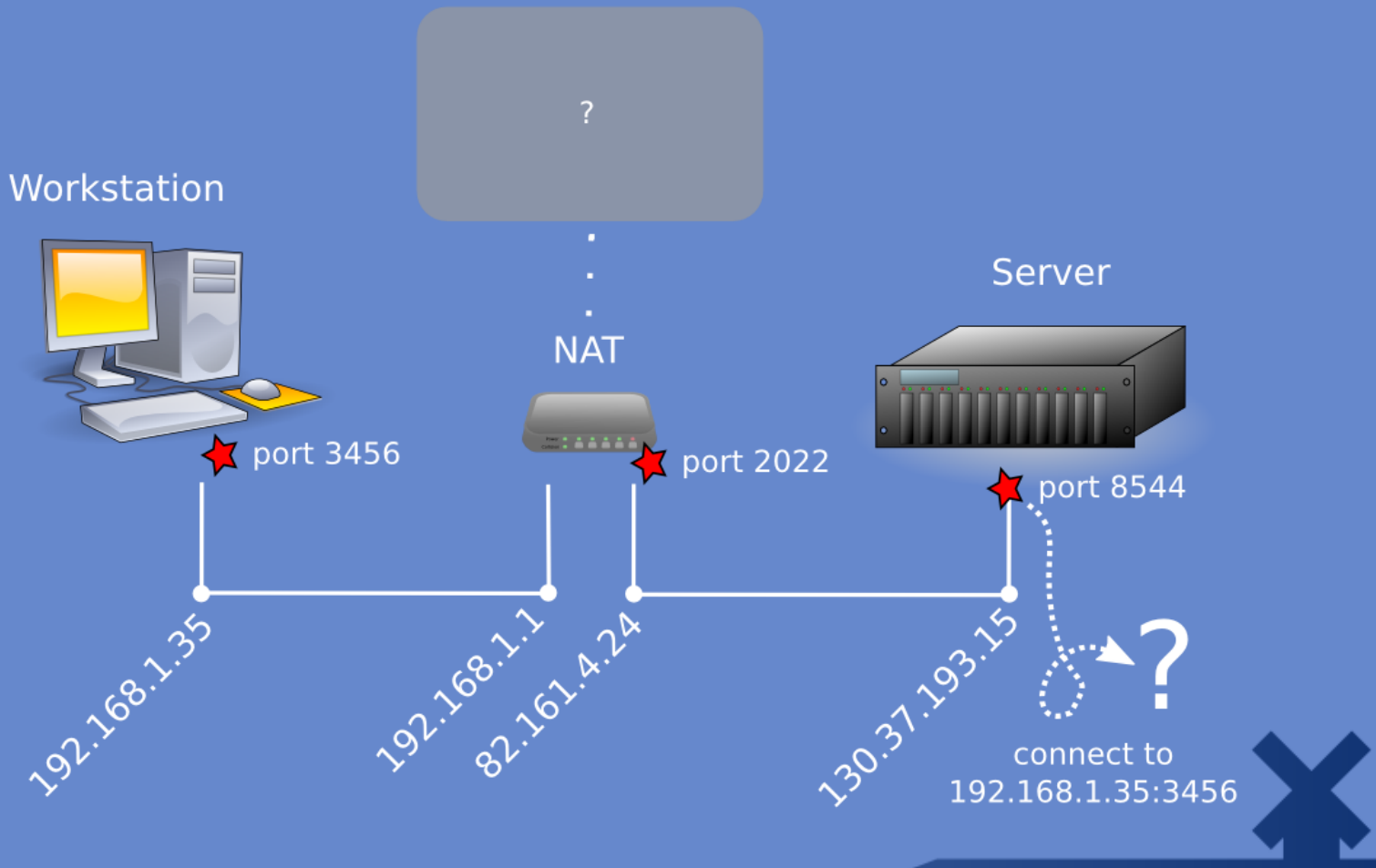
- Allows multiple machines to share an IP address



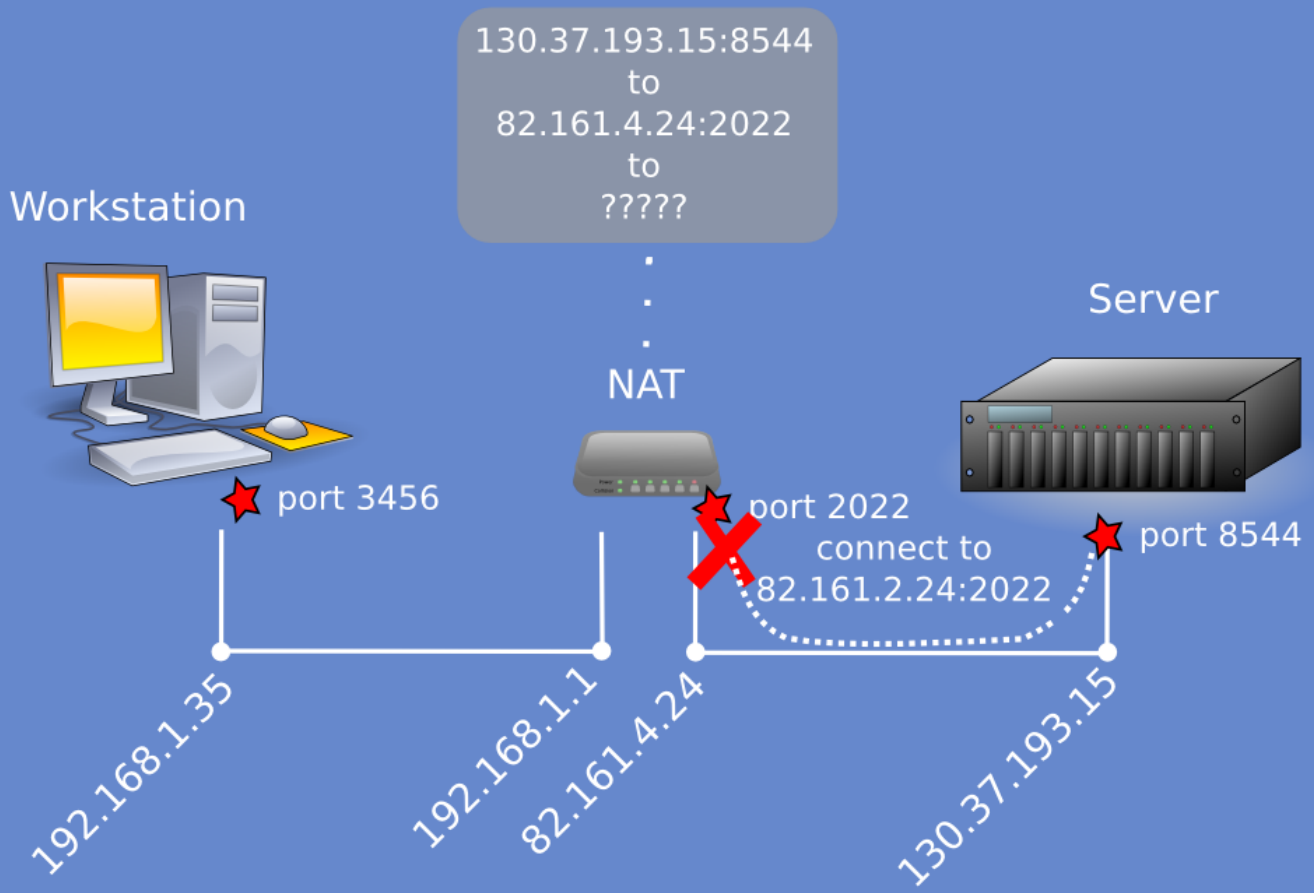
Problem 2:
Network Address Translation



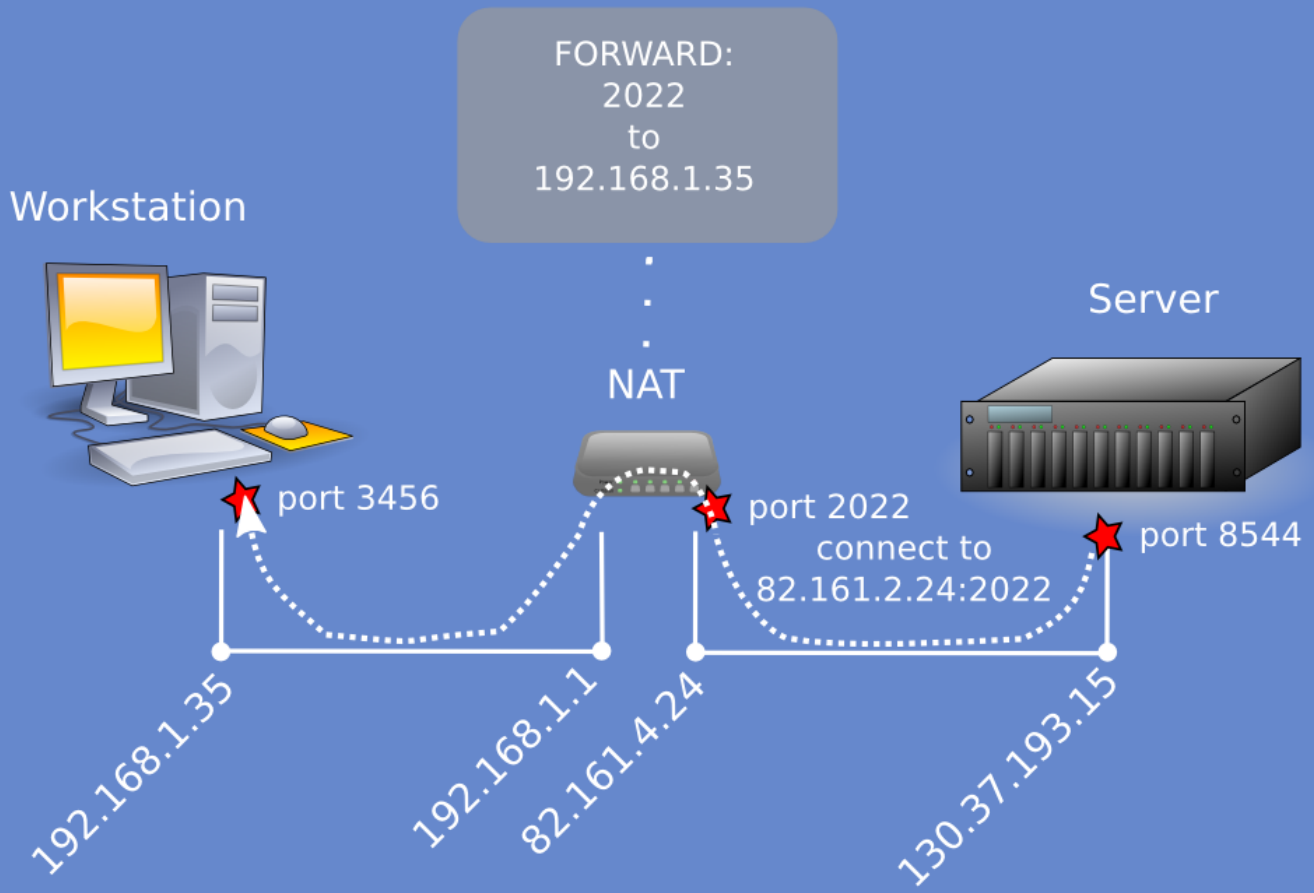
Problem 2:
Network Address Translation



Problem 2:
Network Address Translation

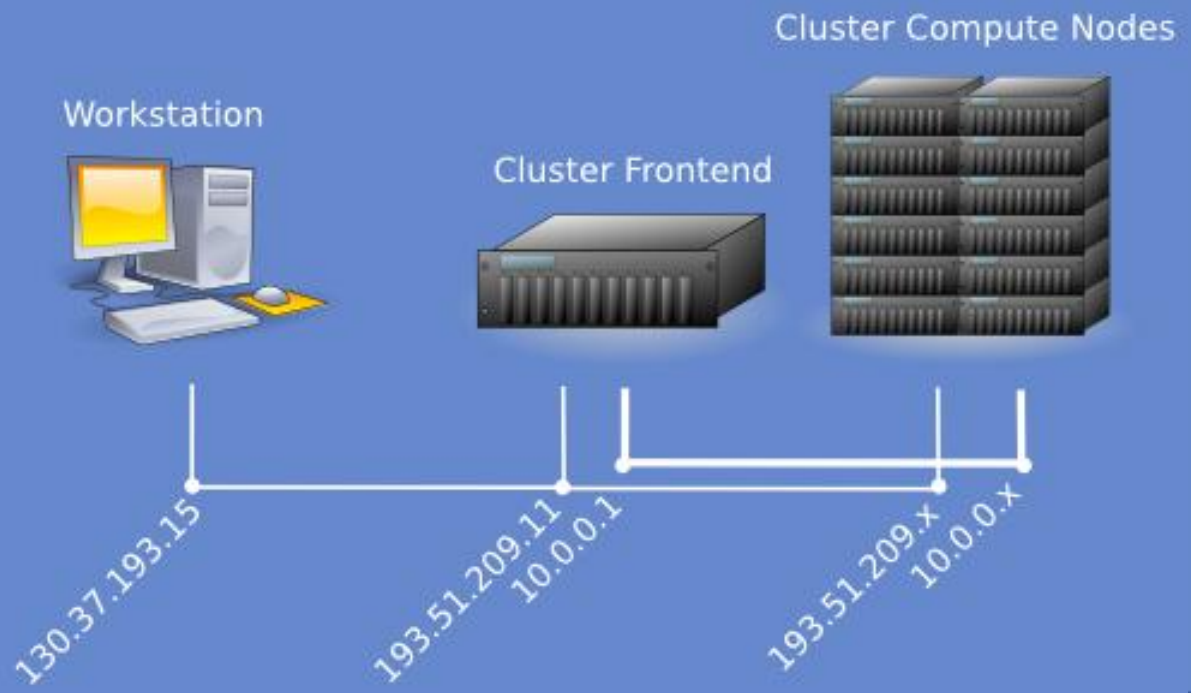


Problem 2:
Network Address Translation



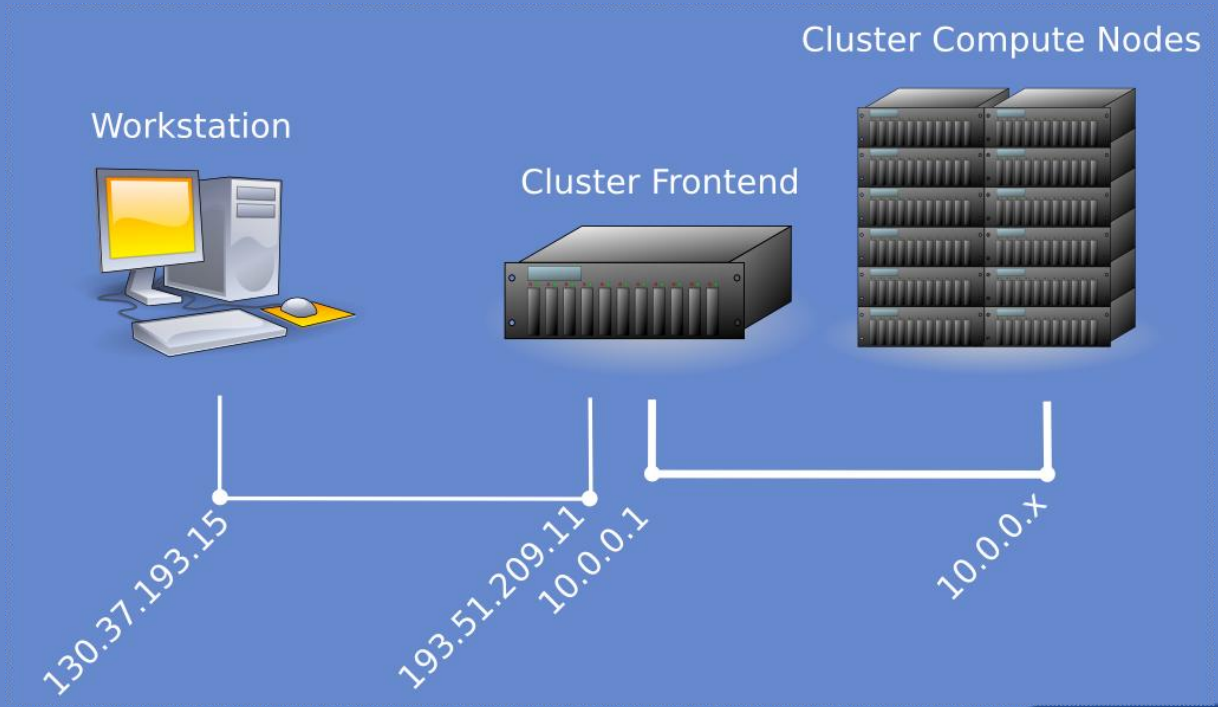
Problem 3: Multi Homing

- Some sites have multiple networks
  - The target address depends on the source of the connection



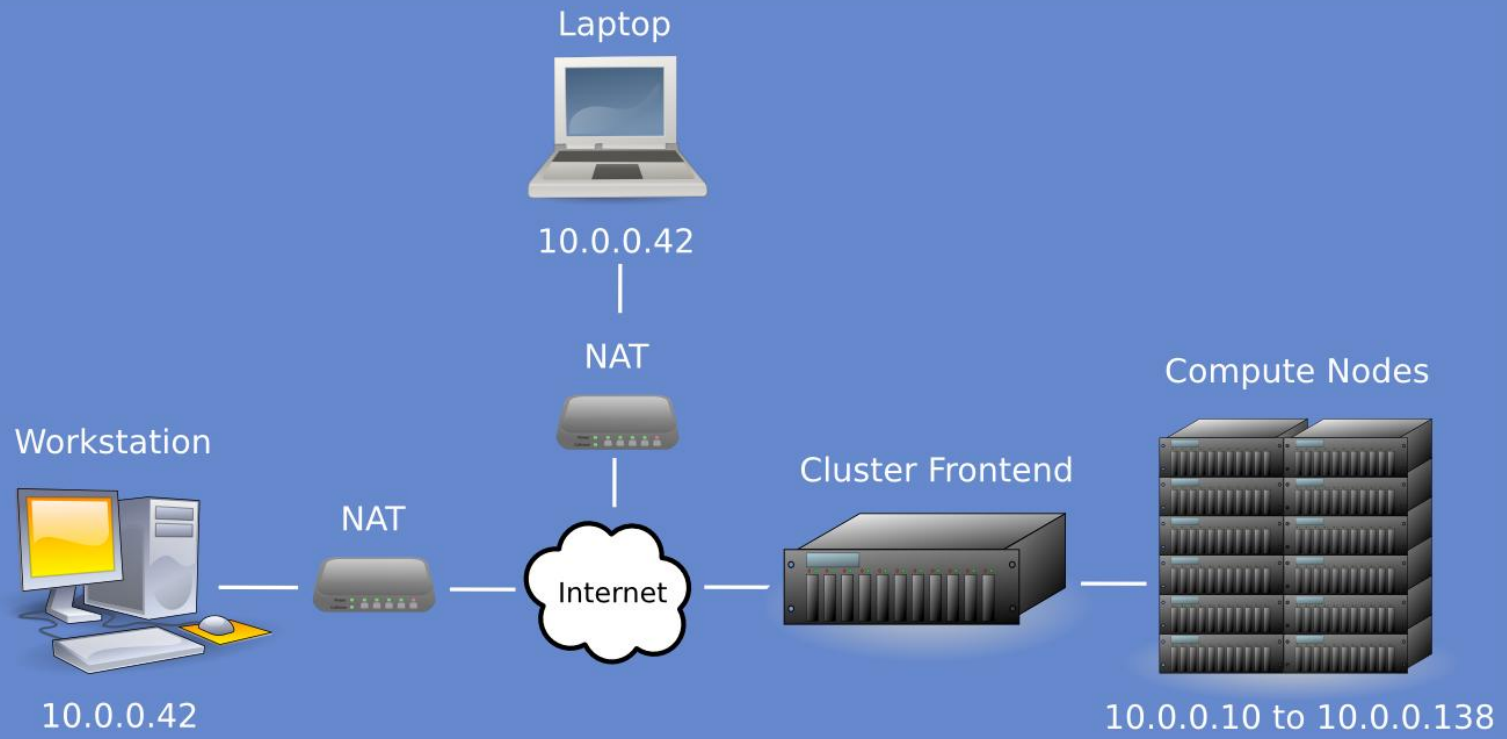
Problem 4: Non-routed Networks

- No route between local network and internet
  - Only the frontend is reachable



Problem 5:
Machine Identification

- Private IPs (NAT/non-routed) lead to machine identification problems



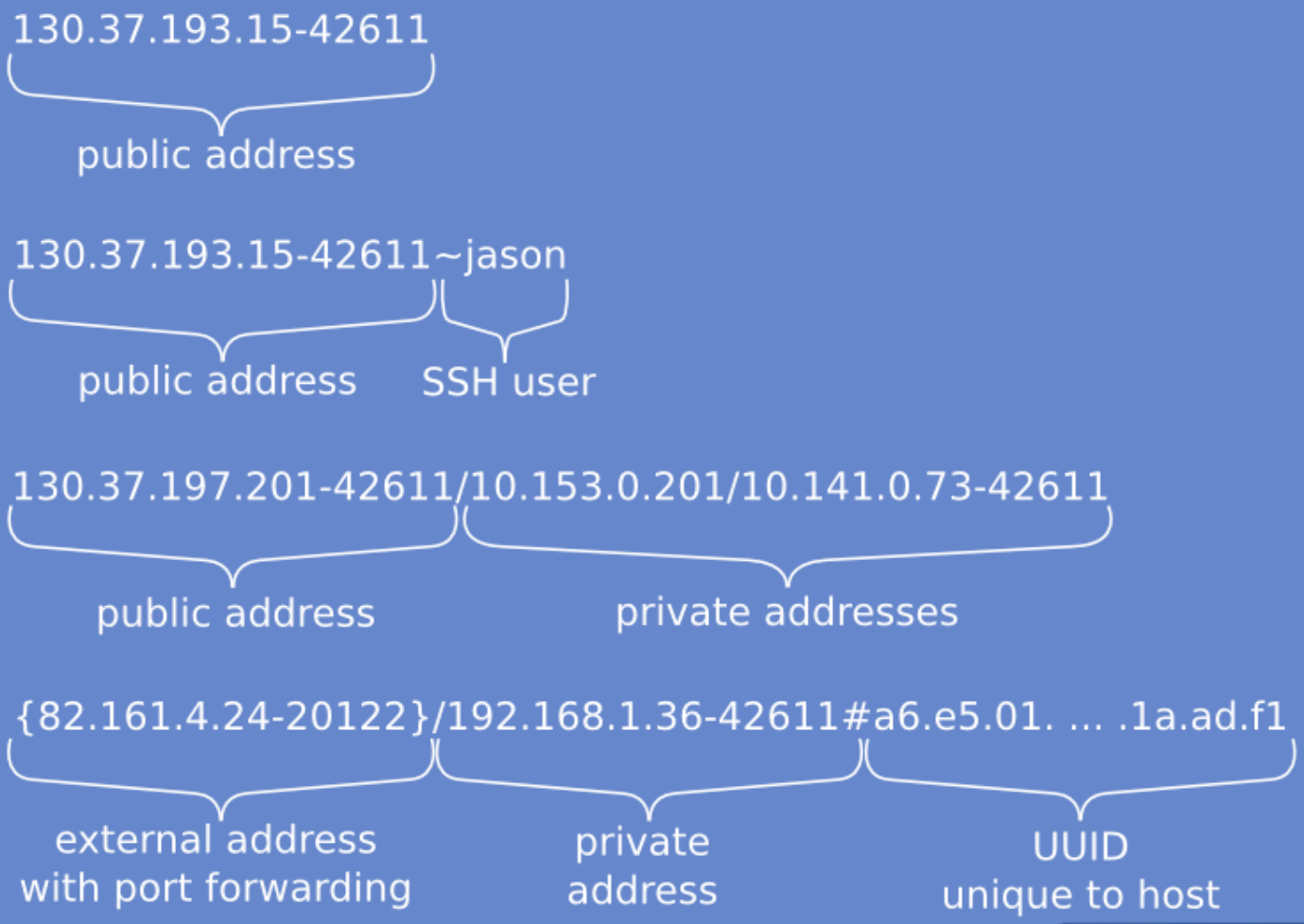
SmartSockets Solutions

- Detect and solve connectivity problems using:
  - Smart Addressing
  - Side channel
  - SSH Tunneling (pass through firewalls)
  - STUN (detect external IP of NAT)
  - UPnP (automatic port forwarding)
  - ...
- Solutions integrated into single socket-like library
  - Mostly transparent to user

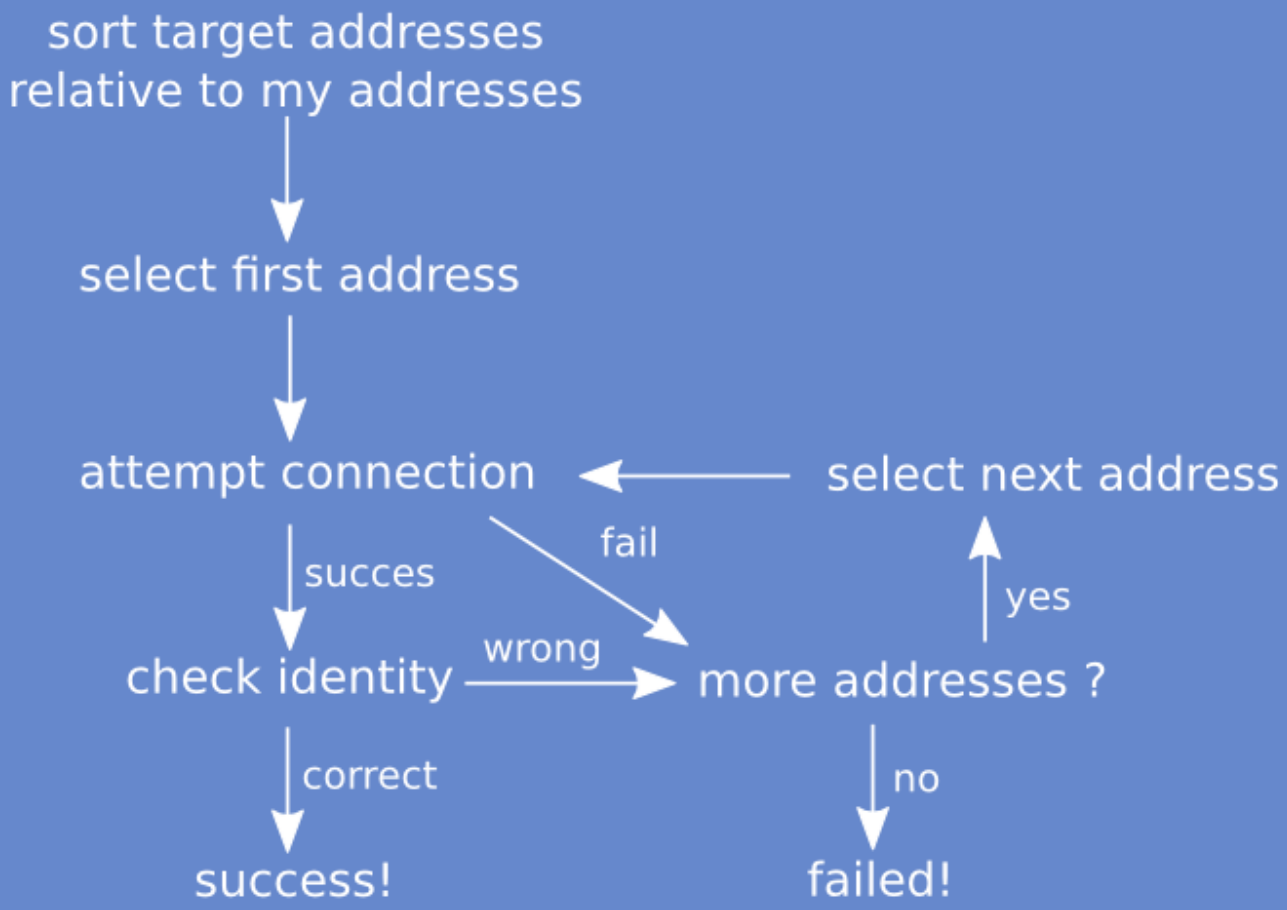
Smart Addressing

- Instead of using a single IP:port combination for each process we use:
  - All machine addresses
  - Add extra information
    - External address + port for NAT (STUN, UPnP)
    - SSH contact information
    - UUID (if entire address is private)
    - …

Addressing Examples



Creating a Connection



Using Smart Addresses

- This solves machine identification problems
  - All addresses are known with multi-homing
  - Each identity is unique, even with private IPs
  - The identity is always checked on connection setup
- Still assumes anyone can create a connection
  - This will not help when target is behind NAT/Firewall
  - To solve this we need a side channel

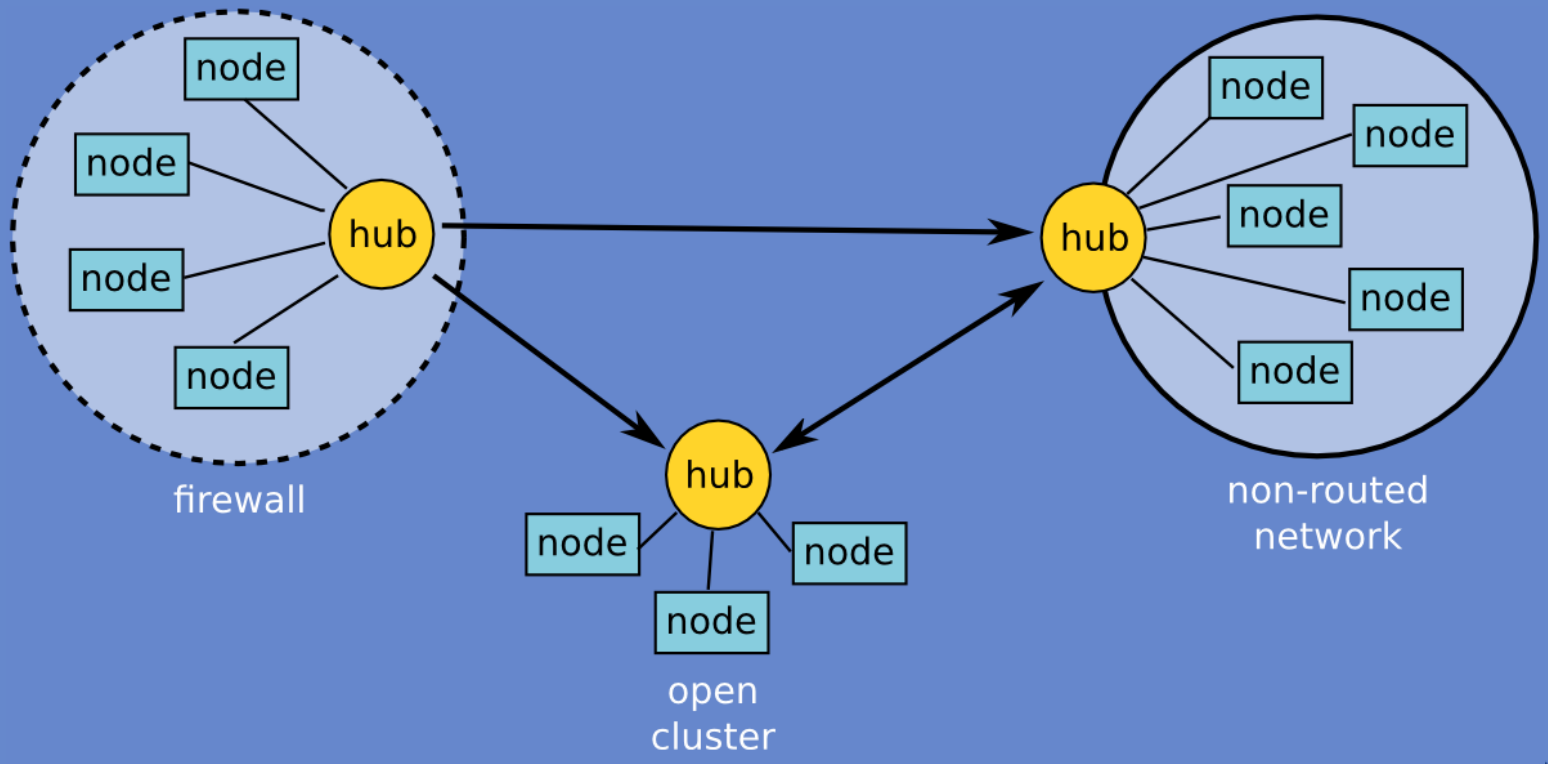
Side channel

- Overlay network implemented using a set of hubs
  - Support processes for the application
  - Started in advance
- Hubs are run on machines with 'more connectivity'
  - Such as cluster frontends, 'open' machines, etc.
- How / where you start them is a separate problem
  - Solved by IbisDeploy (shown later)

Hubs

- Similar to a peer-to-peer overlay network
- Hubs connect to each other
  - Gossip information about other hubs
    - Automatically discover new hubs and routes
  - Need to set up spanning tree (or better)
    - Use direct connections and SSH tunnels
- Clients connect to a 'local' hub
  - Use as side channel for connection setup

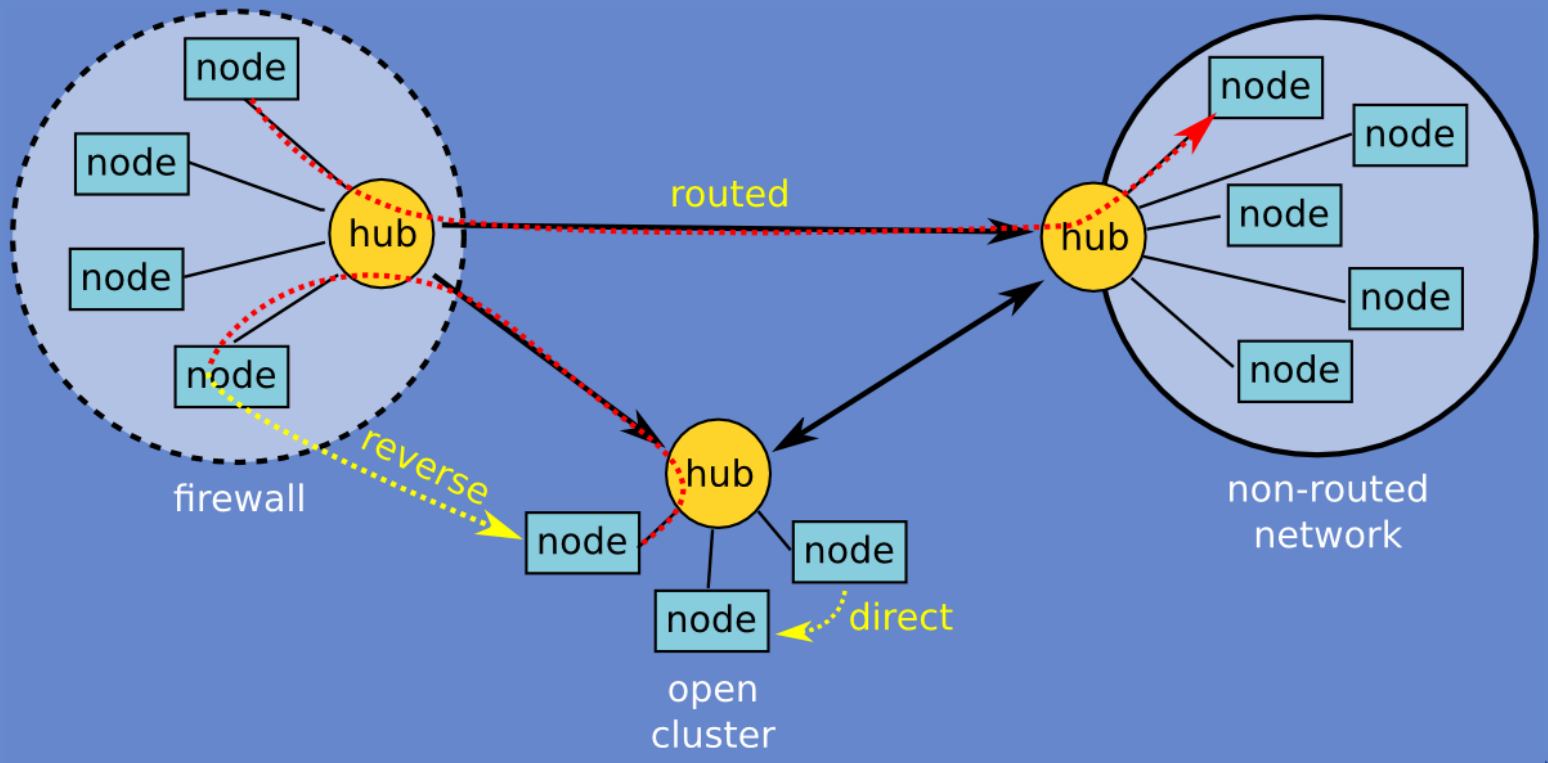
Hub Overlay Network



Advanced Connection Setup

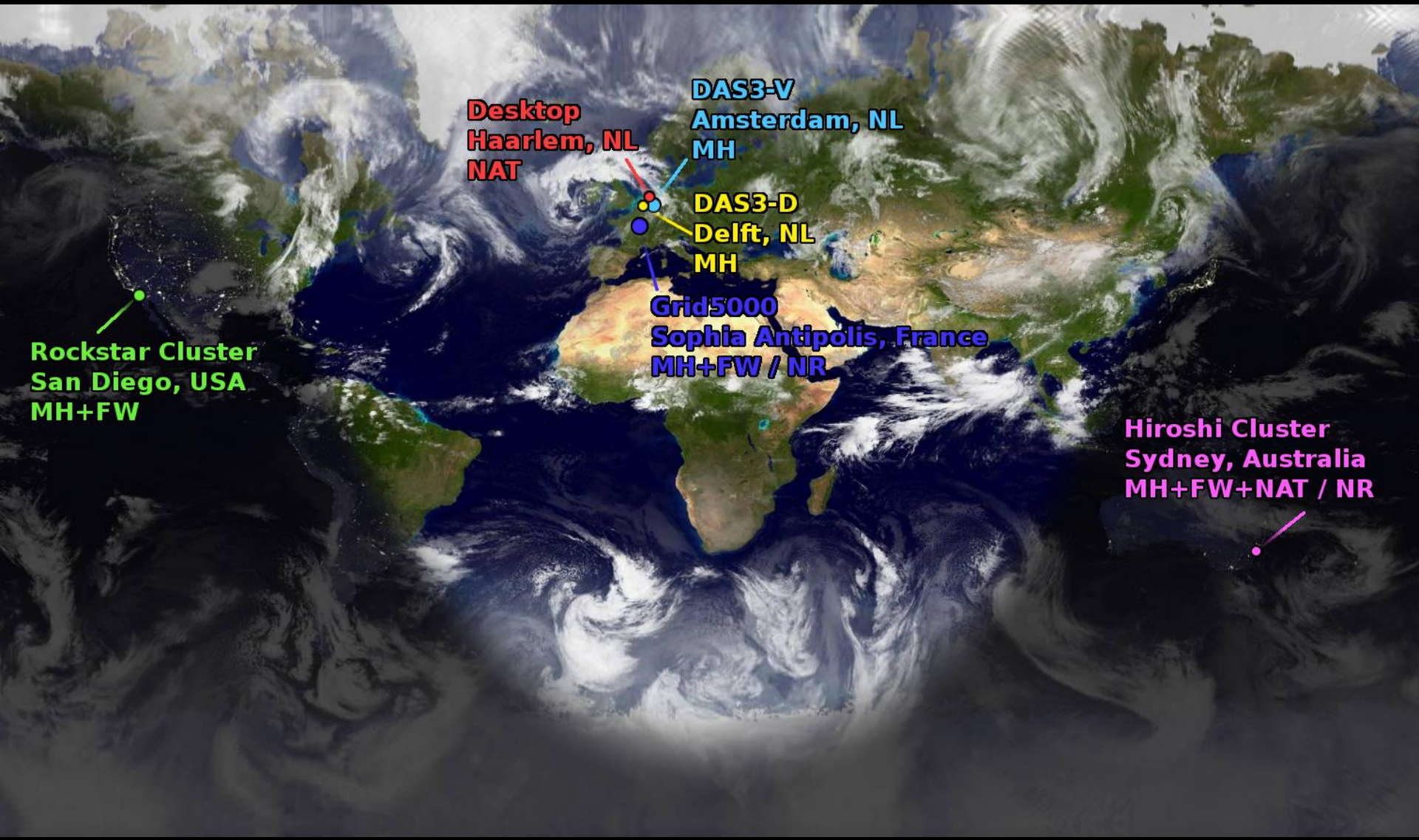
- Reverse direction of connection setup
  - Instruct target to set up connection (using hub)
  - Results in direct connection
- Splicing (switched off by default)
  - Connection setup from both sides (using hub)
  - Results in direct connection
- Route via overlay
  - Create virtual connection using hubs
  - Forward all data over side channel
  - Results in indirect connection

Advanced Connection Setup



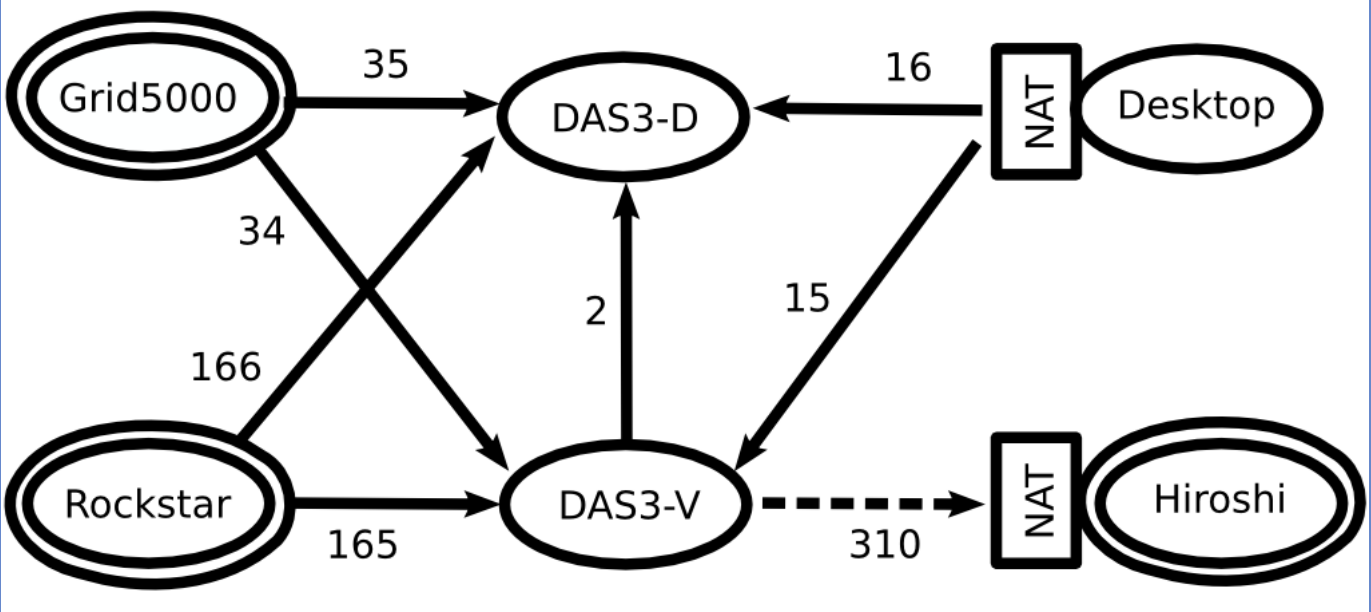
SmartSockets
Overview of solutions

- Machine identification:
  - Smart addressing
  - Identity check at connection setup
- Unreachable machines:
  - SSH tunnels
  - Reverse connection setup
  - (Splicing)
  - Routing over hubs


Desktop
Haarlem, NL
NAT
DAS3-V
Amsterdam, NL
MH
DAS3-D
Delft, NL
MH
Grid5000
Sophia Antipolis, France
MH+FW / NR
Rockstar Cluster
San Diego, USA
MH+FW
Hiroshi Cluster
Sydney, Australia
MH+FW+NAT / NR

Overlay Network

Created by hubs



Evaluation

Table 3: Connection setup time of SmartSockets (time in milliseconds).

	Source					
Target	DAS3-V	DAS3-D	Rockstar	Grid5000	Hiroshi	Desktop
DAS3-V		4.9^{d} (2.4)	332^{d} (166)	68^{v}	595^{v}	33^{d} (17)
DAS3-D	4.9^{d} (2.4)		335^{d} (167)	70^{v}	595^{v}	33^{d} (18)
Rockstar	500^{r}	503^{r}		206^{v}	718^{v}	182^{v}
Grid5000	35^{v}	38^{v}	206^{v}		593^{v}	54^{v}
Hiroshi	630^{v}	603^{v}	750^{v}	670^{v}		640^{v}
Desktop	49^{r}	52^{r}	183^{v}	84^{v}	606^{v}	

Annotations indicate connection style: *d* for direct, *r* for reverse, *s* for splicing, and *v* for routed. When applicable, the connection setup time of regular sockets is shown between brackets.

- Regular TCP/IP only worked in 6 out of 30
- SmartSockets worked in 30 out of 30

Evaluation

Table 4: Roundtrip latency of SmartSockets (time in milliseconds).

	Source					
Target	DAS3-V	DAS3-D	Rockstar	Grid5000	Hiroshi	Desktop
DAS3-V		2.3 (2.3)	166 (166)	56	528	14 (14)
DAS3-D	2.3 (2.3)		167 (167)	57	533	15 (15)
Rockstar	166	167		205	590	195
Grid5000	56	57	205		524	50
Hiroshi	528	529	590	522		539
Desktop	14	15	190	43	522	

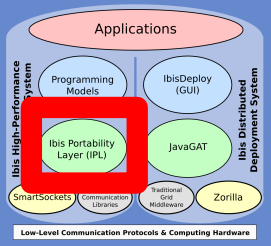
When applicable, the roundtrip latency of regular sockets is shown between brackets.

Table 5: Throughput of SmartSockets (in Mbit/second).

	Source					
Target	DAS3-V	DAS3-D	Rockstar	Grid5000	Hiroshi	Desktop
DAS3-V		182 (183)	2.6 (2.5)	2.5	0.25	0.65 (0.65)
DAS3-D	185 (186)		2.6 (2.5)	2.6	0.26	0.65 (0.65)
Rockstar	2.8	2.7		6.9	0.23	0.65
Grid5000	7.6	8.2	2.4		0.20	0.65
Hiroshi	0.73	0.73	0.70	0.73		0.61
Desktop	3.3	3.3	2.2	2.2	0.25	

When applicable, the throughput of regular sockets is shown between brackets.

However….



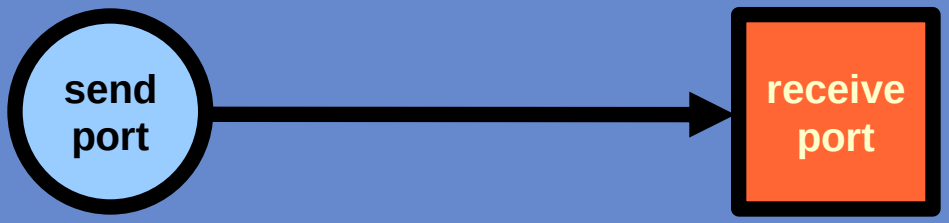
- SmartSockets is great, but too low level
- For Jungle computing we need support for
  - Malleability
  - Fault Tolerance
  - Robust and globally unique naming
  - Flexible communication primitives
- Provided by the Ibis Portability Layer (IPL)

Ibis Portability Layer
(IPL)

- Simple API for Jungle Communication
  - Flexible communication model
    - Connection oriented messaging
    - Abstract addressing scheme
  - Resource tracking (JEL model)
    - Notifications when machines join/leave/crash
  - Efficient serialization
    - Send bytes, doubles, objects, etc.
  - Portable:
    - SmartSockets, TCP, UDP, MPI, MX, BlueTooth,…

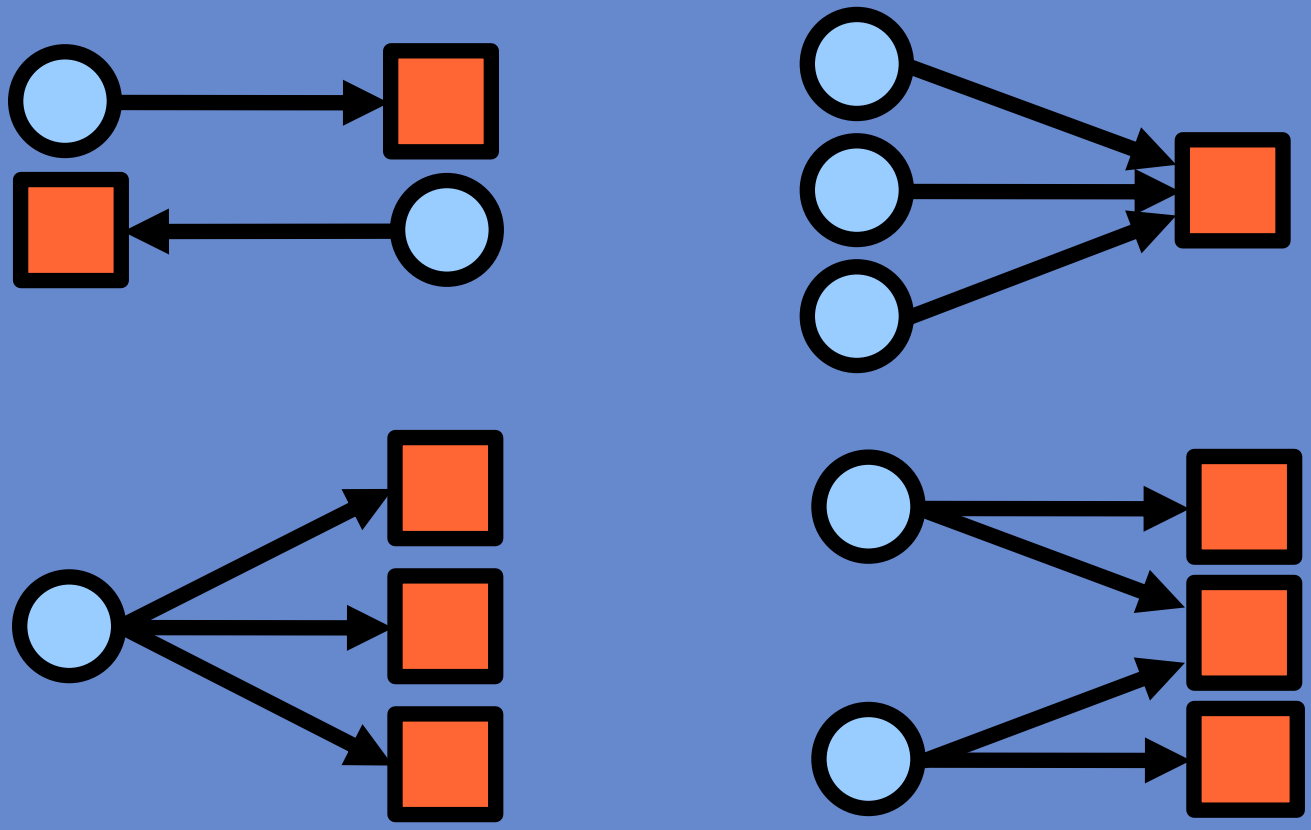
Communication Model

- Simple communication model
  - Unidirectional pipes
  - Two end points (send and receive ports)



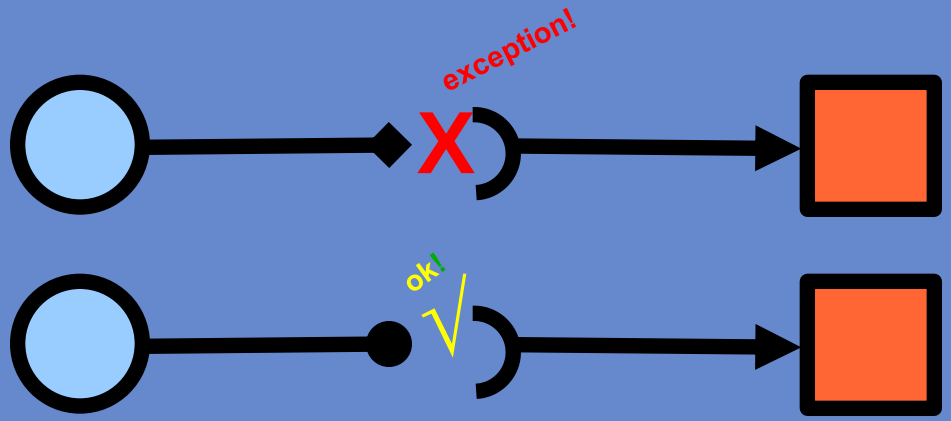
- Connection oriented
  - Allows streaming (good with high latency)

Flexible Communication Model



Port Types

- All send and receive ports have a type
  - Defined at runtime
  - Specify set of capabilities
- Types must match when connecting!



Port Types

- Consists of a set of capabilities:
  - Connection patterns:
    - Unicast, many-to-one, one-to-many, many-to-many
  - Communication properties:
    - Fifo ordering, numbering, reliability
  - Serialization properties:
    - Bytes, primitive types, objects
  - Message delivery:
    - Explicit receipt, automatic upcalls, polling

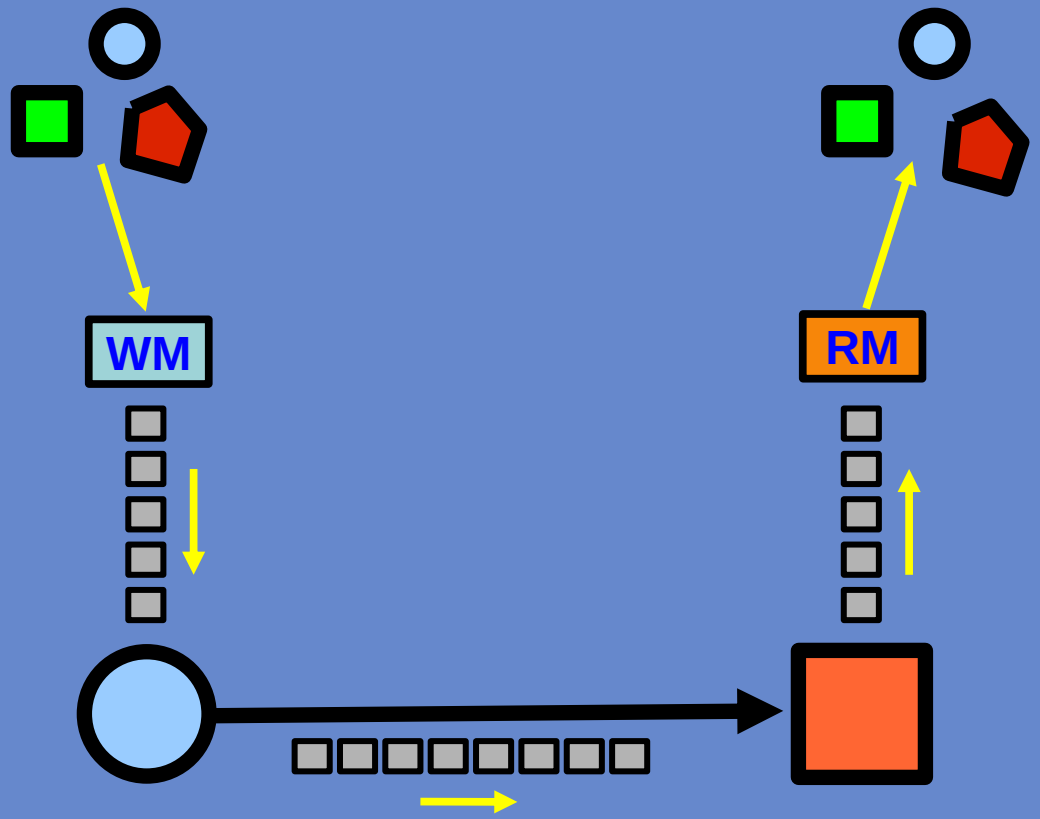
Port Types

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

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
  - Streaming

Messages Example



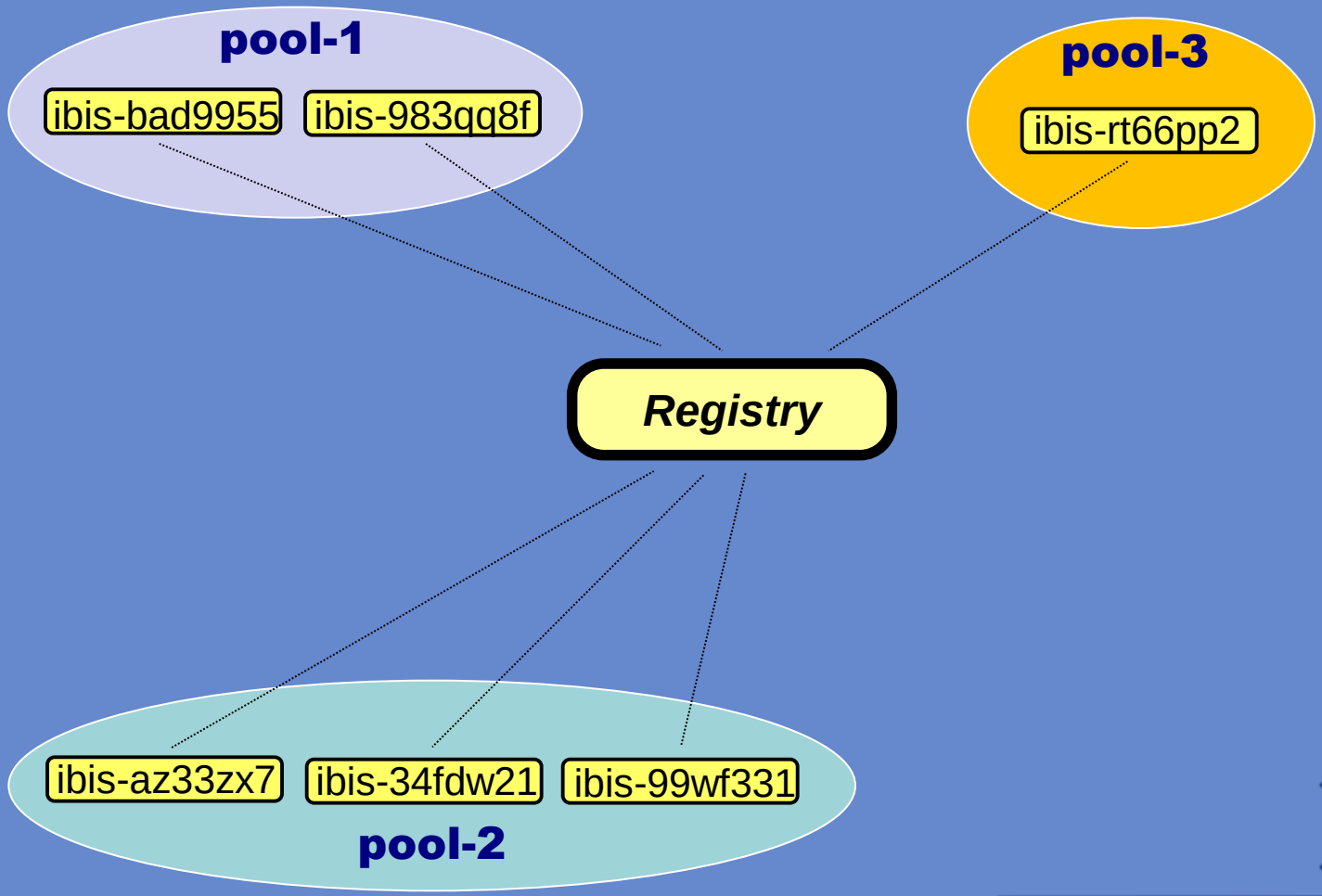
Abstract addressing

- IbisIdentifier:
  - Abstract ‘proces address' object
  - Hides network specific details
    - Examples: SmartSockets addresses, hostnames, IP addresses, MPI ranks, etc.
- Results in more portable applications
  - Independent of network infrastructure
- Why don't we use ranks ?
  - Hard to support malleability and fault-tolerance!

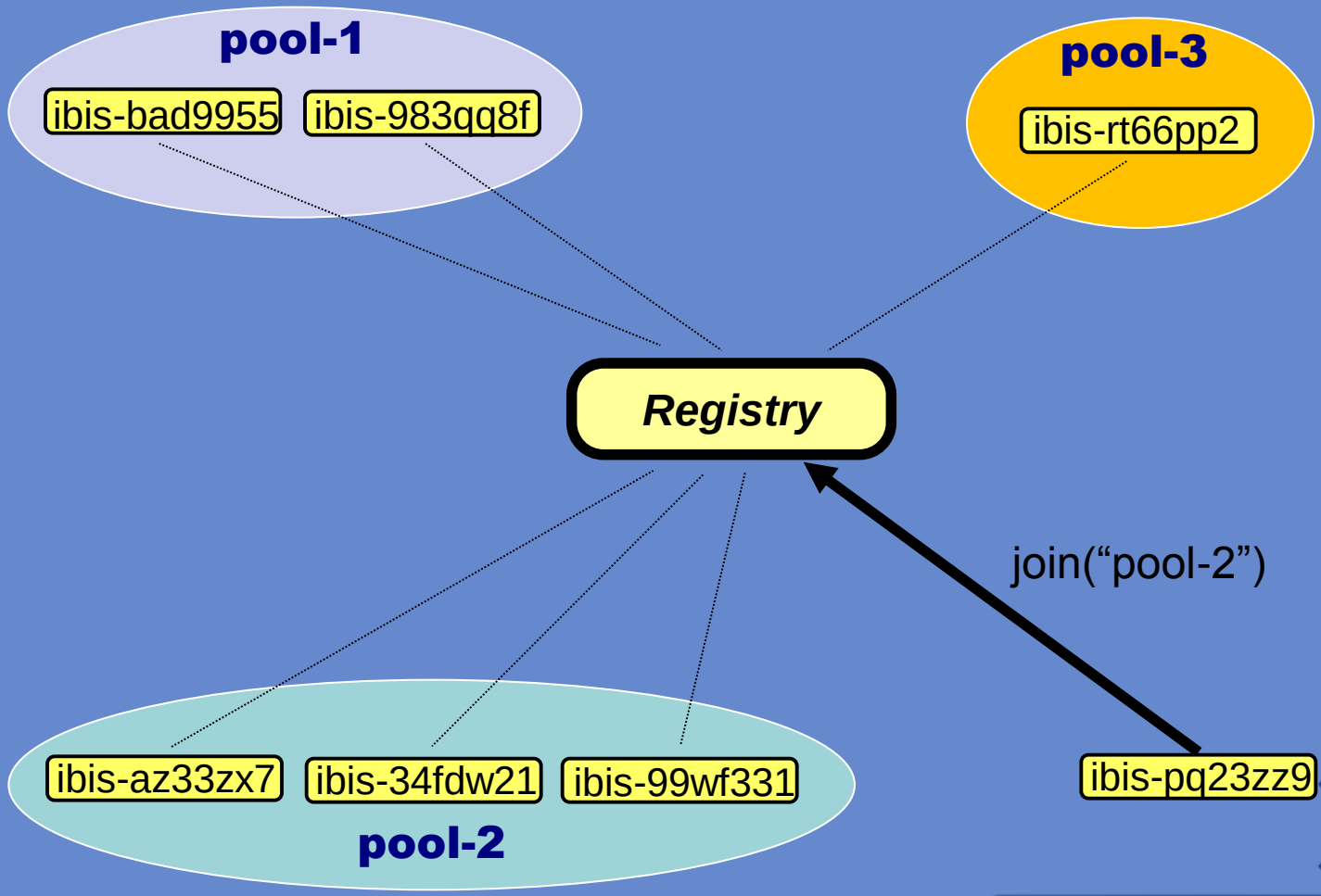
Resource Tracking

- IPL offers JEL (join, elect, leave) model
  - Application gets signal when someone joins or leaves
  - Supports elections for distributed decision making
    - Allows machines to be elected as “master”
  - Can ensure totally ordered notifications
- Implemented using separate registry component
  - Server that tracks application participants
  - Can track multiple applications simultaneously, each in its own pool

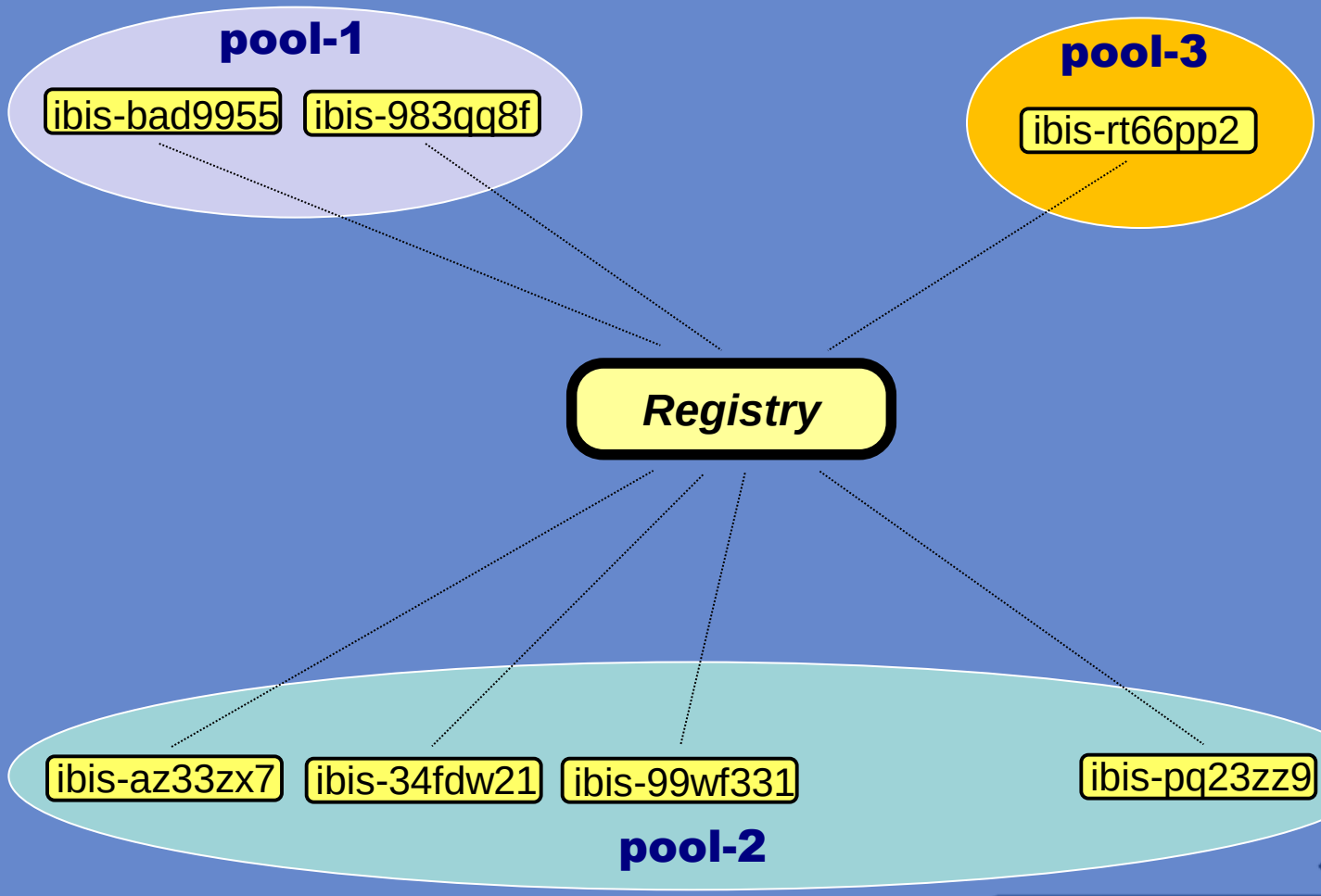
Pools & Malleability



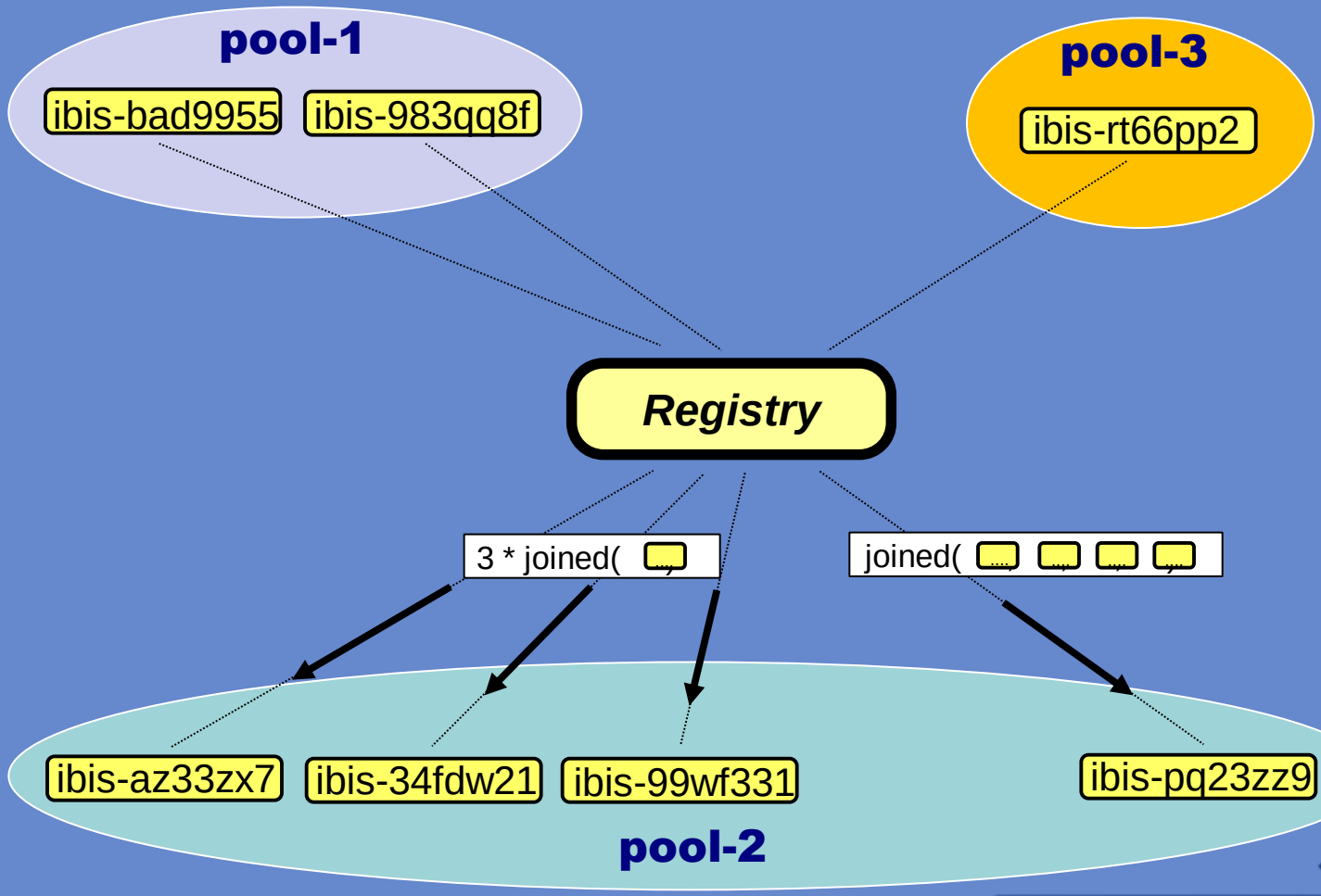
Pools & Malleability



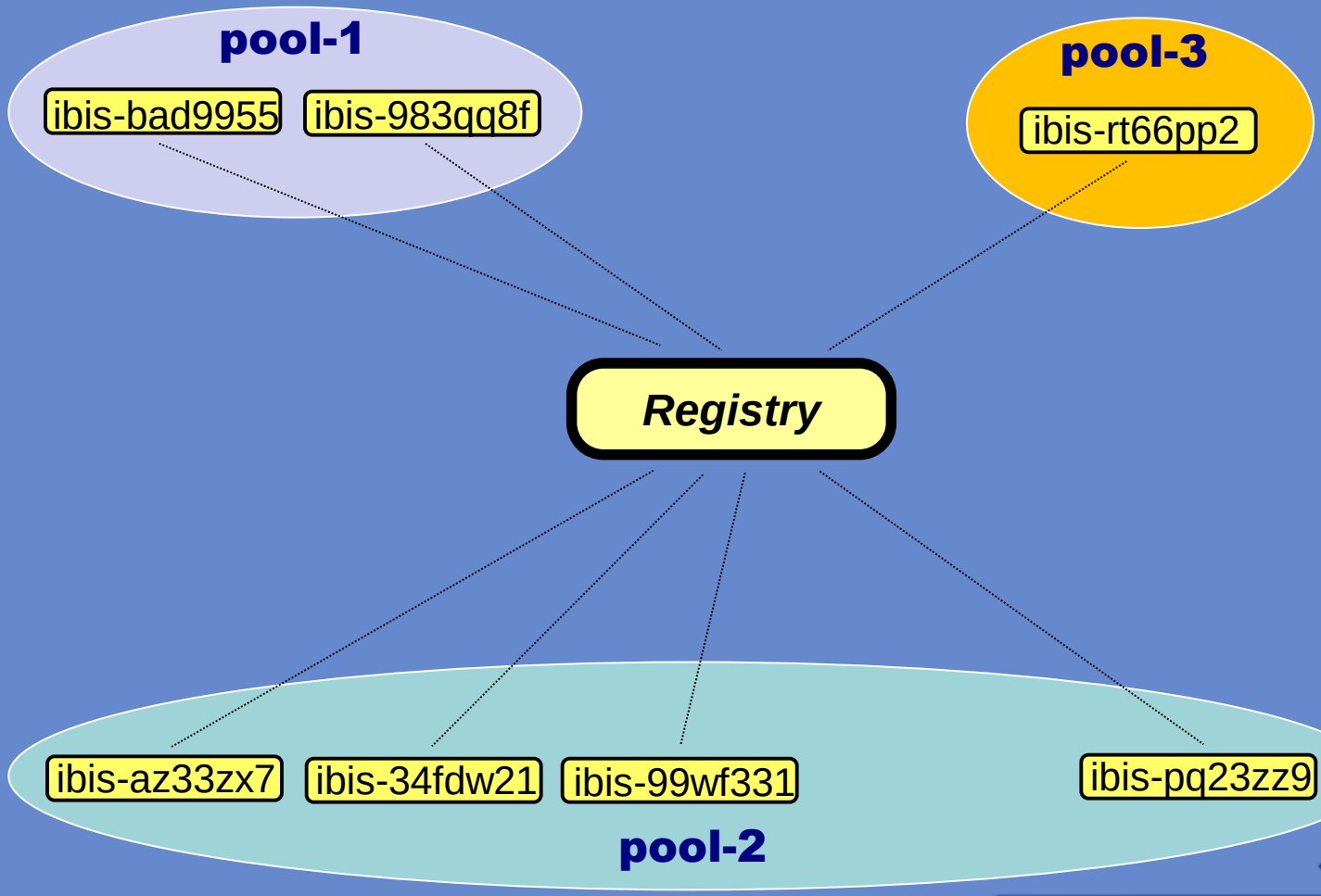
Pools & Malleability



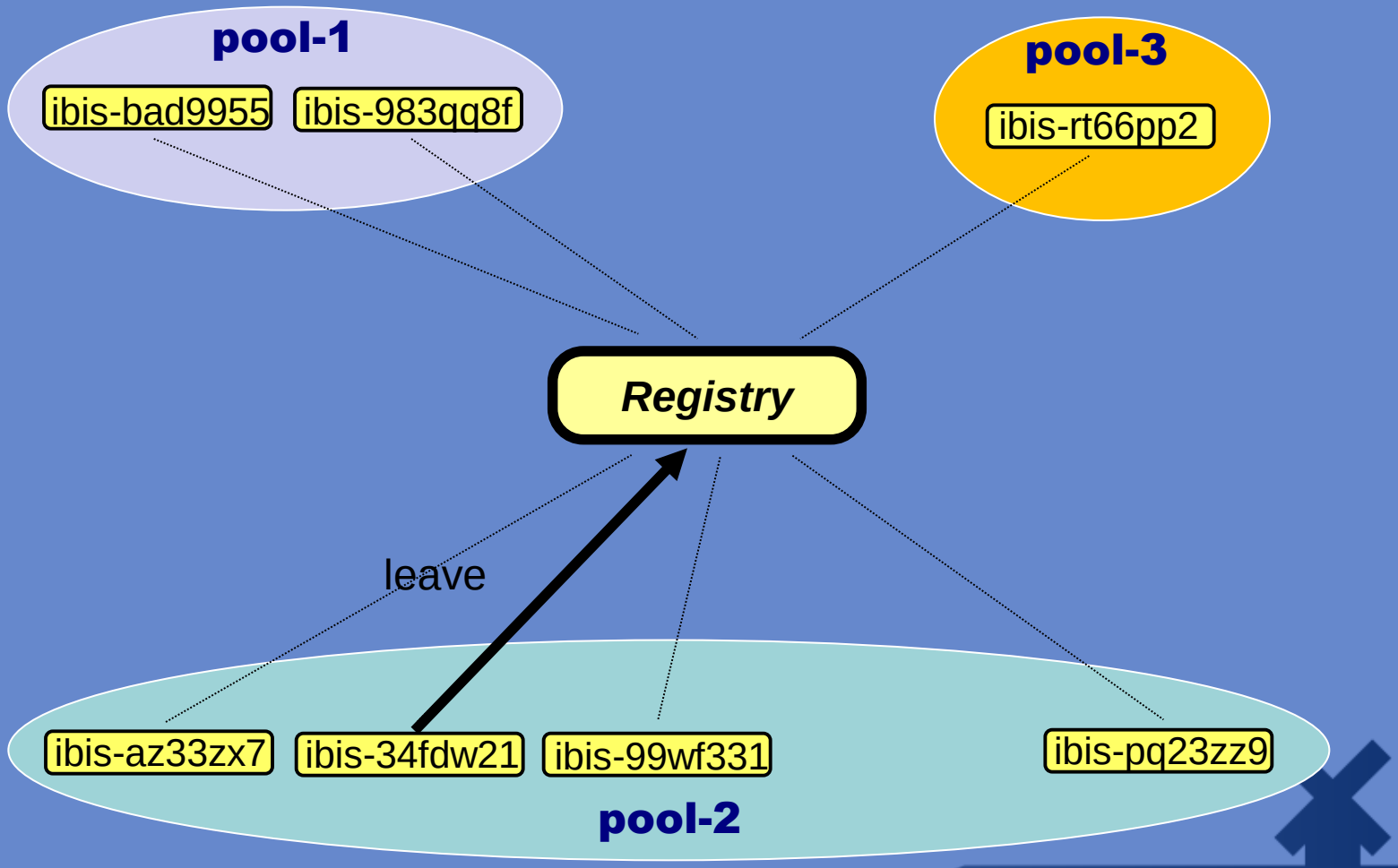
Pools & Malleability



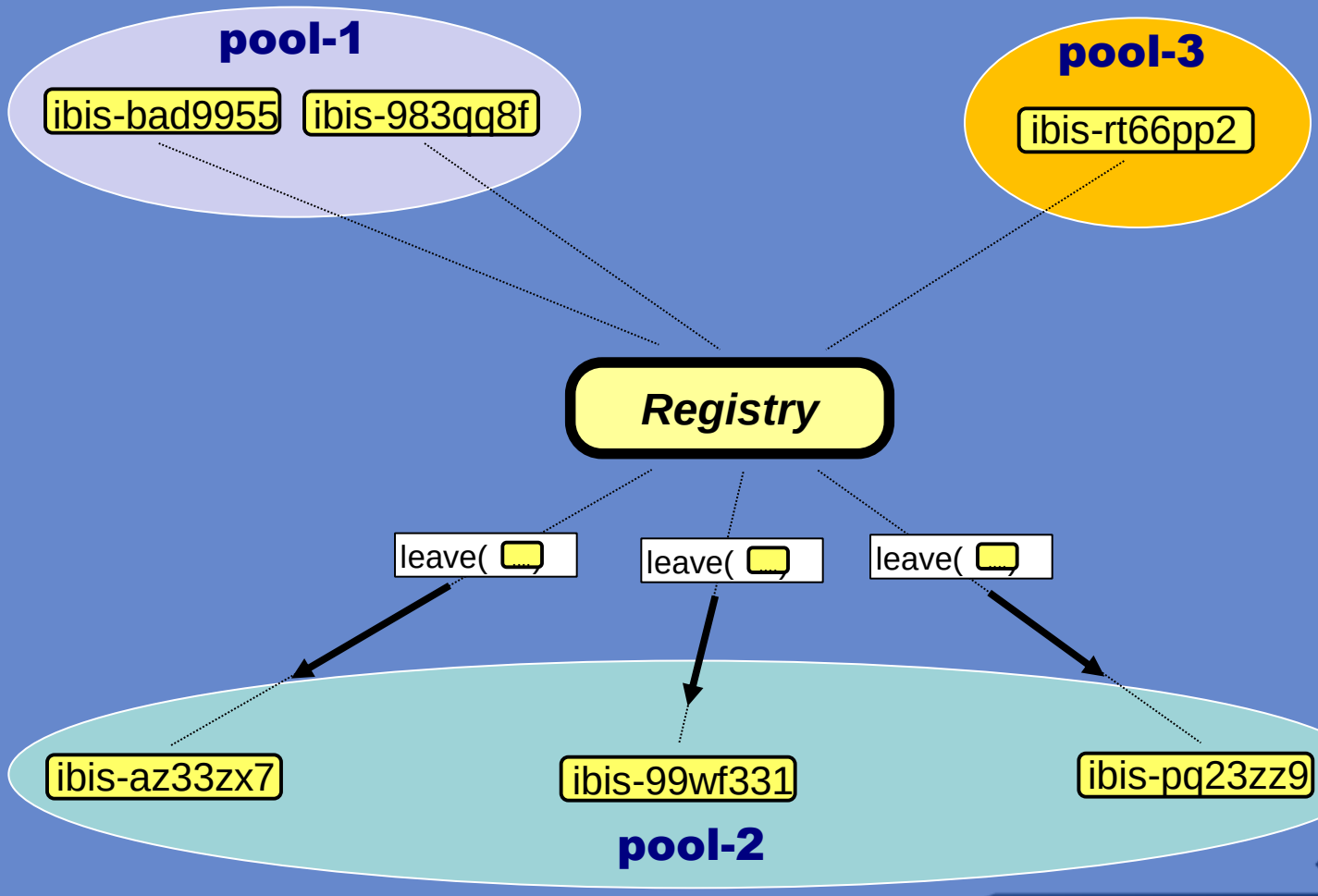
Pools & Malleability



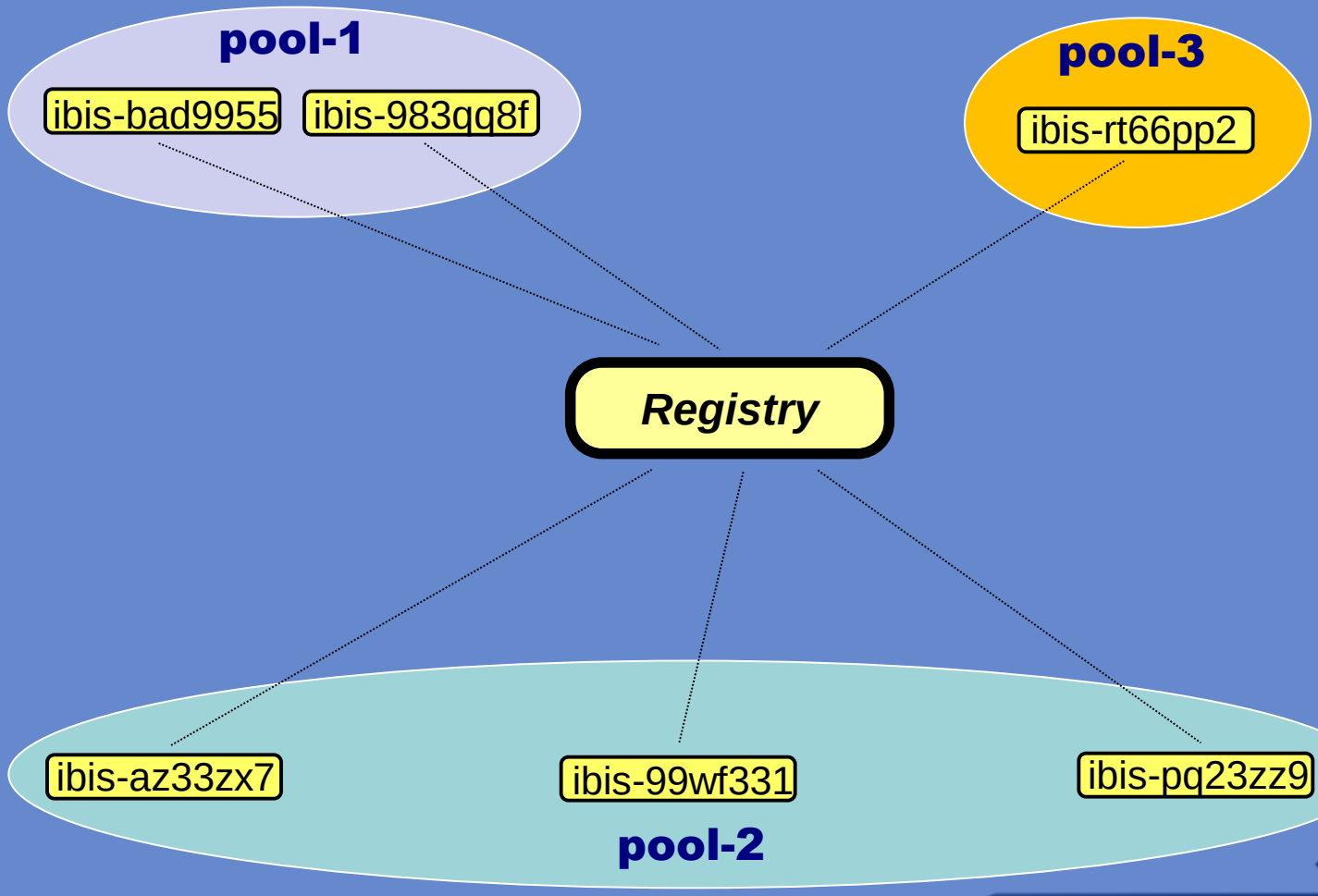
Pools & Malleability



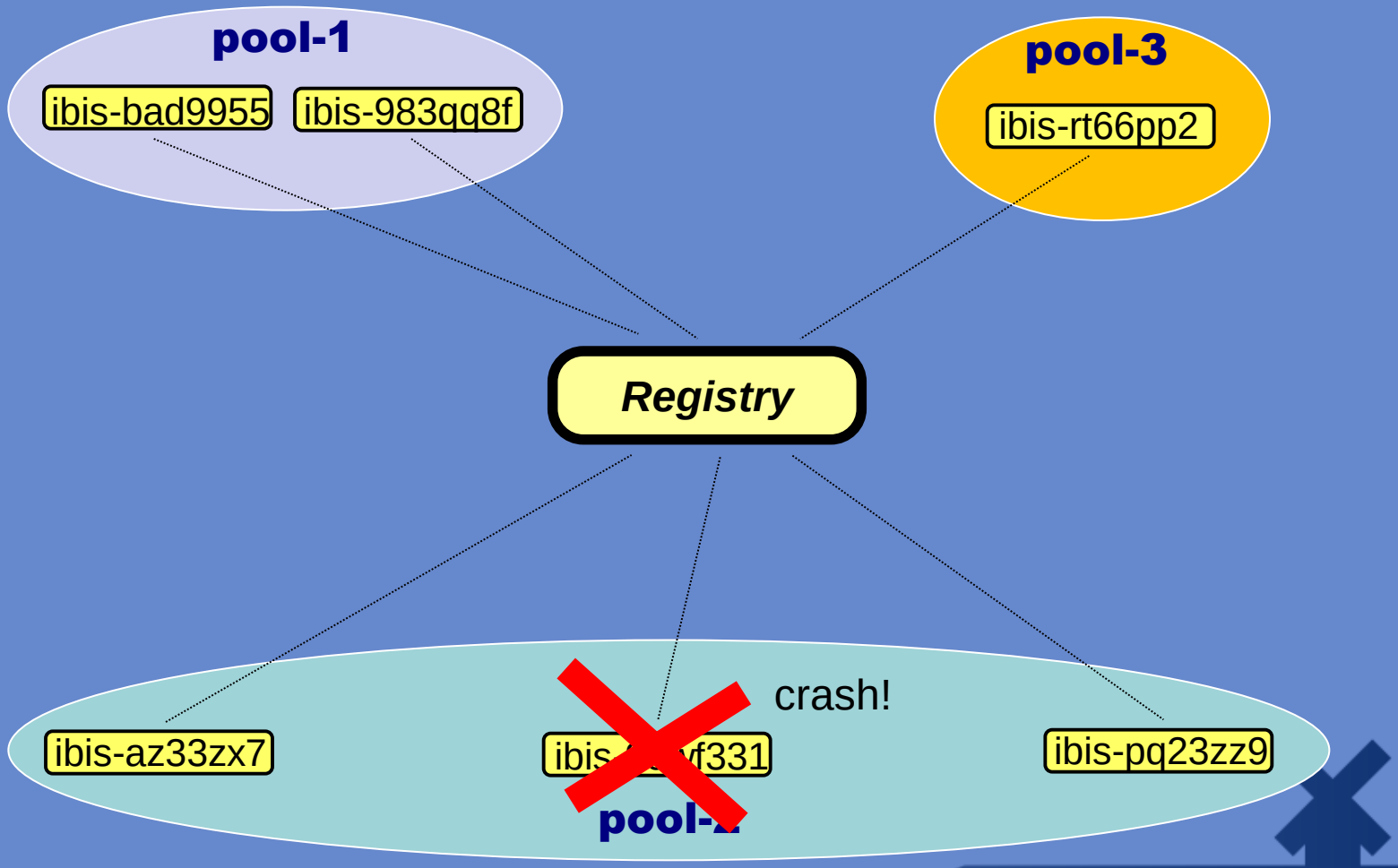
Pools & Malleability



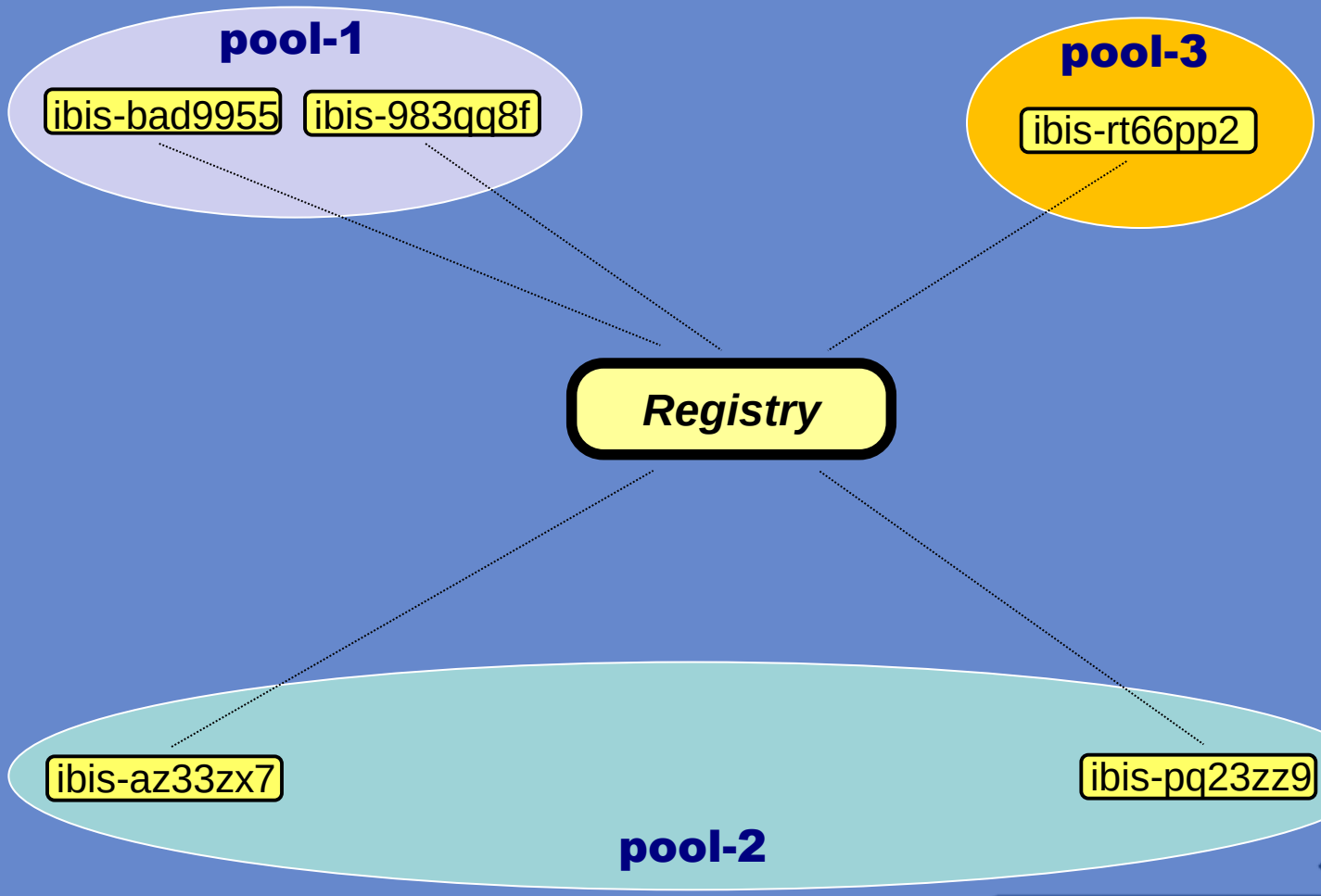
Pools & Malleability



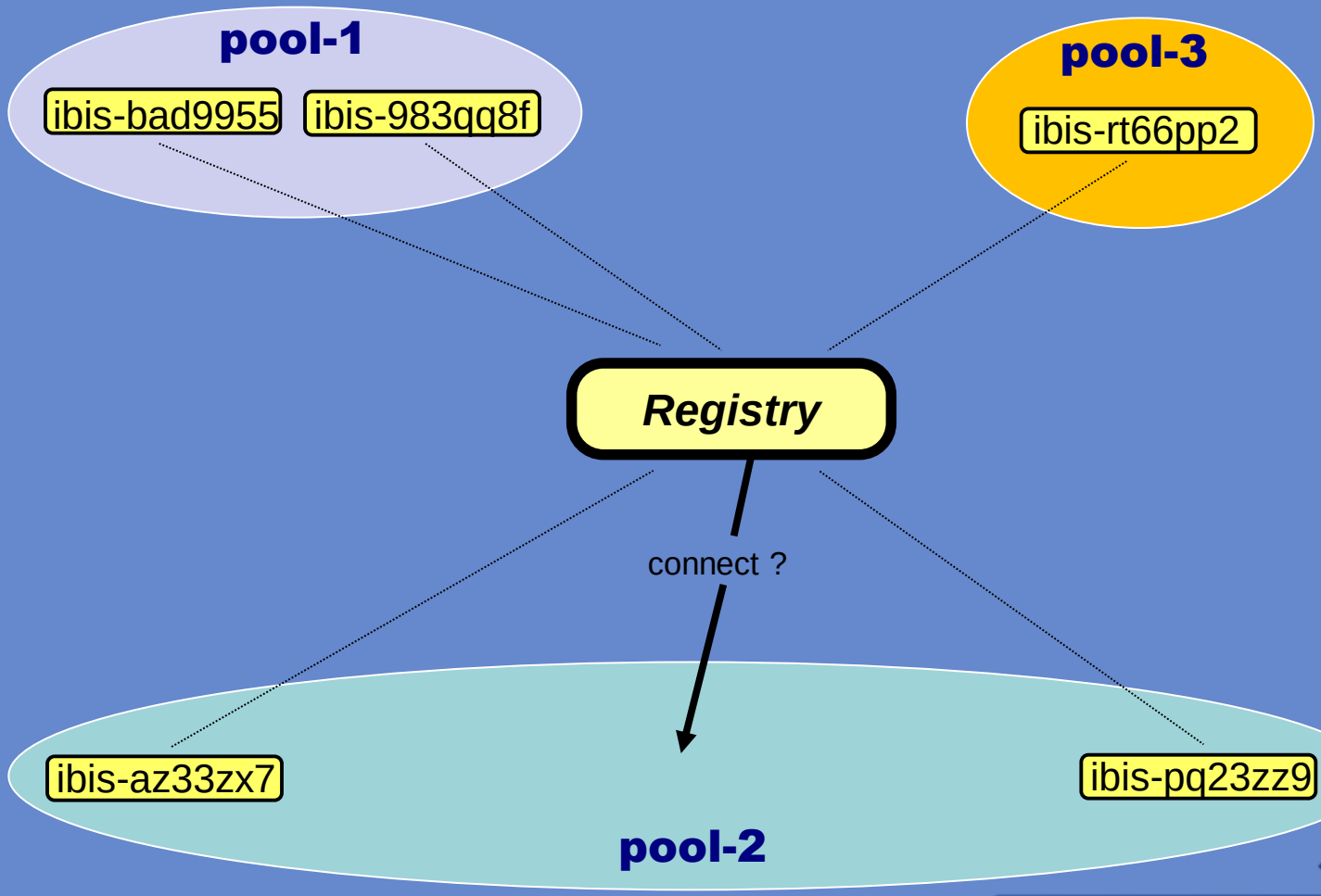
Pools & Malleability



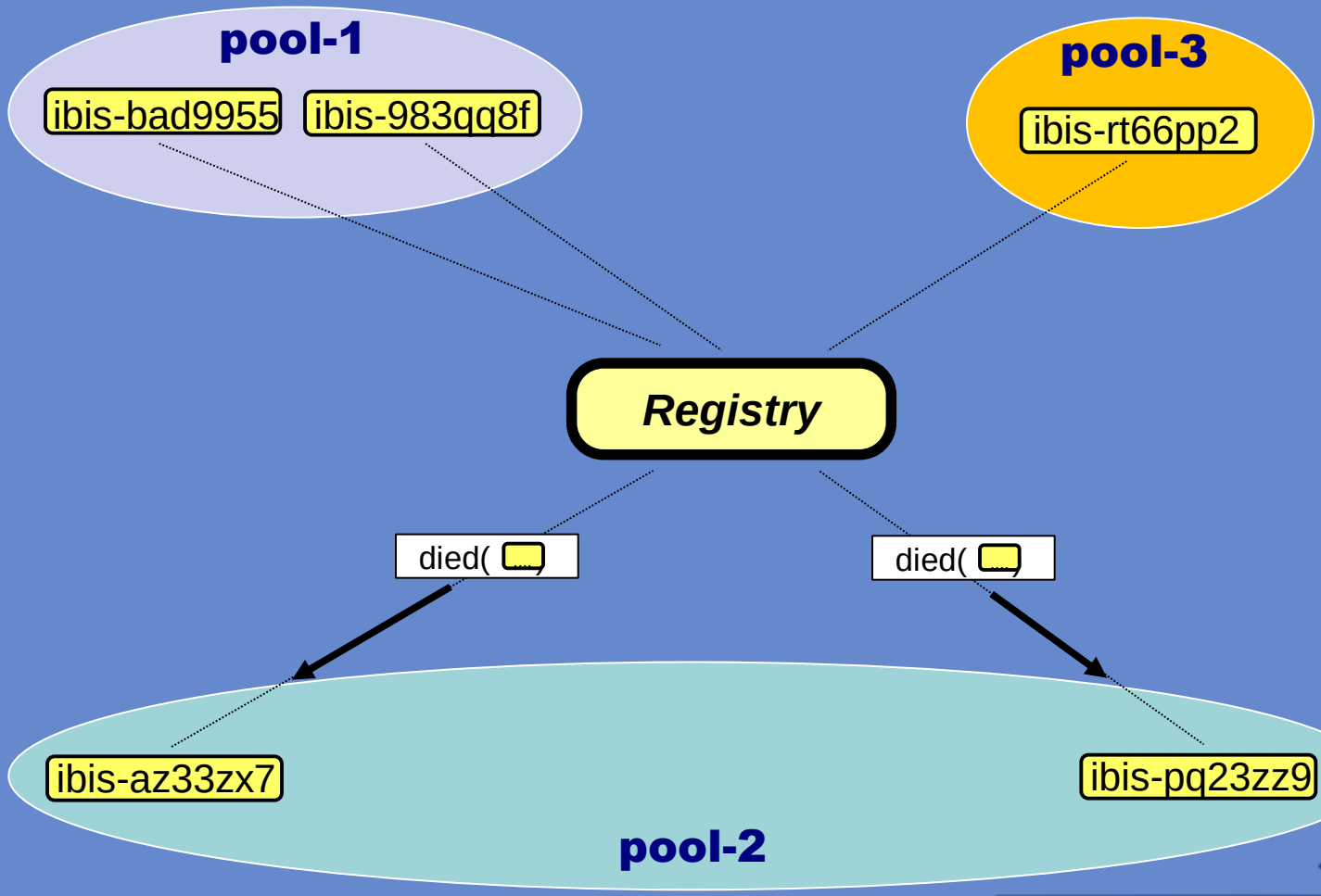
Pools & Malleability



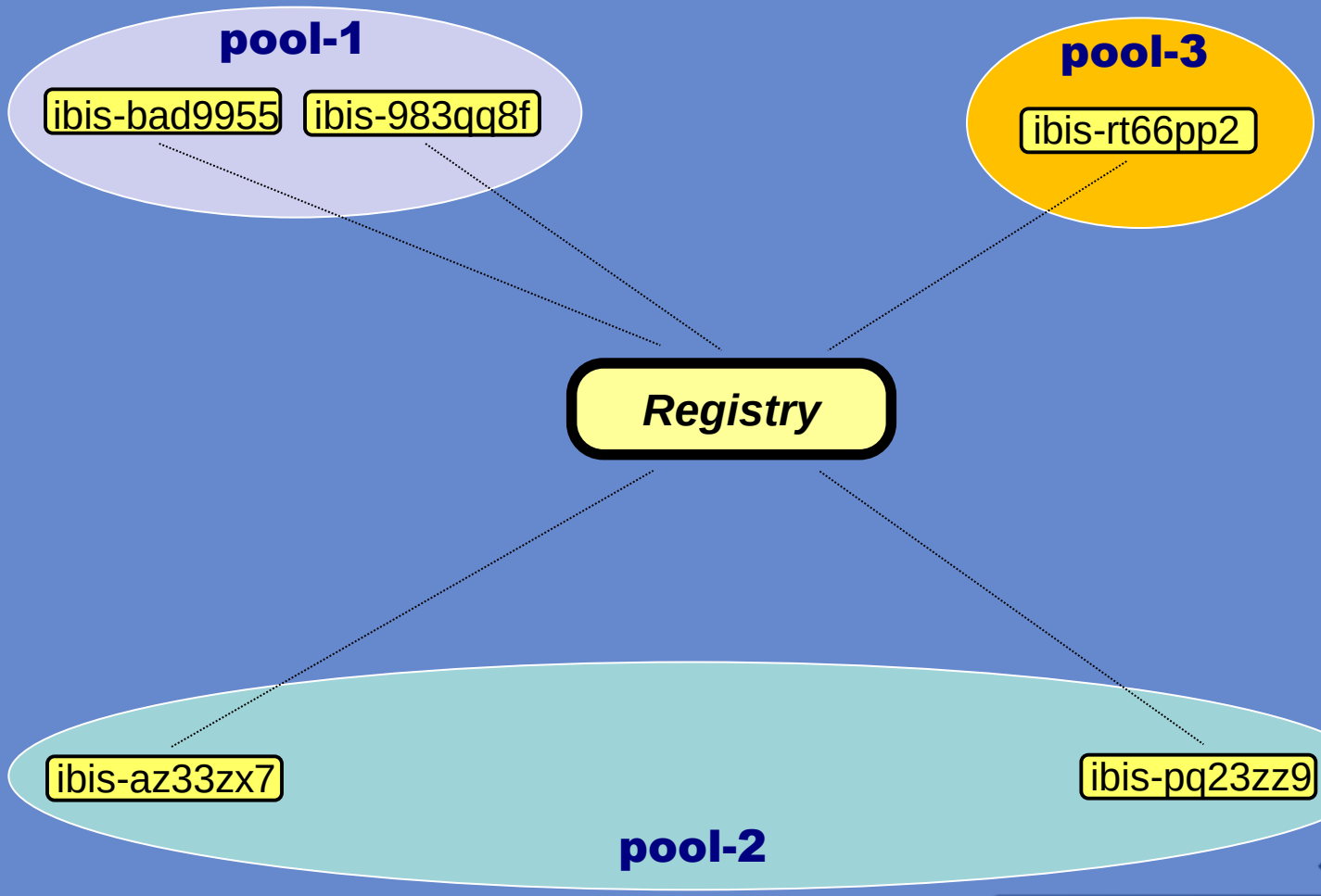
Pools & Malleability



Pools & Malleability



Pools & Malleability



Registry

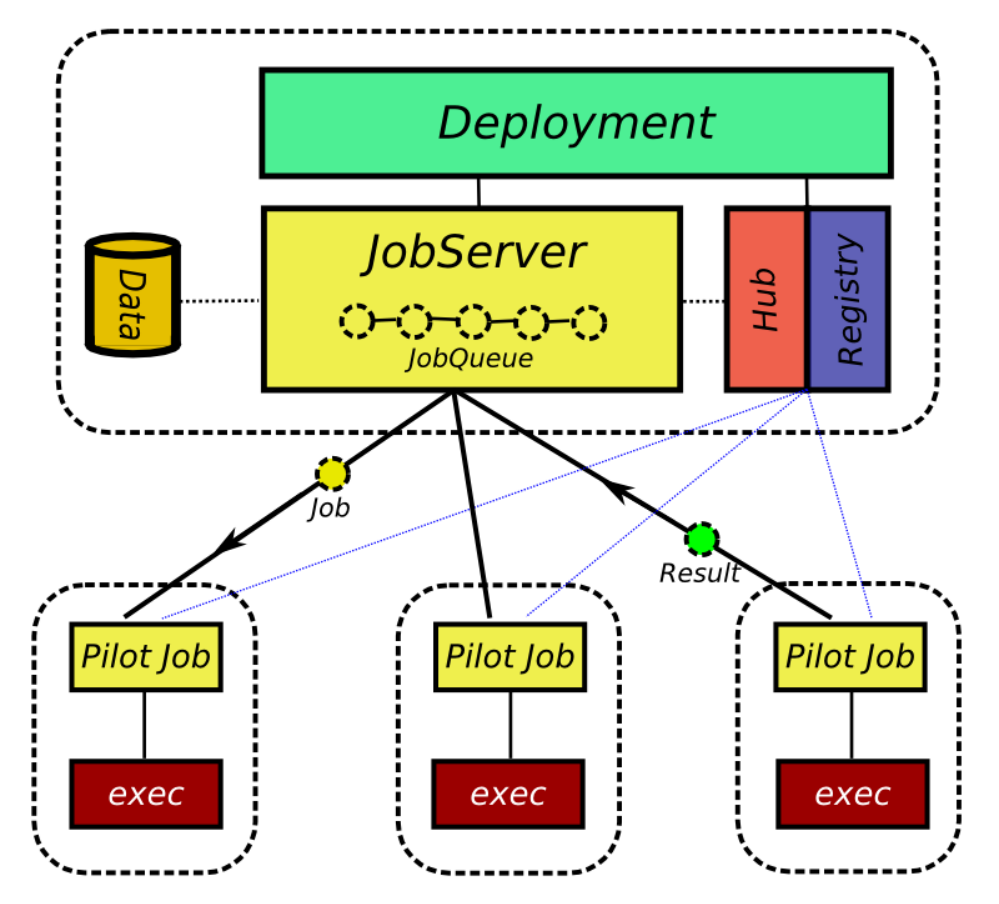
- Many implementations
  - Centralized, broadcast, gossiping, etc.
  - Different tradeoffs in functionality, complexity, robustness, scalability and consistency
- Application can select the functionality and consistency that is needed
  - Reducing functionality or consistency further improves scalability

IPL Example:
Back to the Pilot Jobs

- Now we have Jungle proof communication it should be easy to create a pilot job framework
- Recipy:
  - Create a Pilot Job framework using the IPL
  - Submit this to the resources using JavaGAT
  - Distribute work using the Pilot Job framework
- We'll use this as an IPL code example!

Pilot Job Framework

Design



Example Code
Deployment

```
package tutorial20.glue;

import ibis.ipl.server.Server;

public class Main {

    static class Resource {
        String brokerURI;
        String javaLocation;

        Resource(String brokerURI, String javaLocation) {
            this.brokerURI = brokerURI;
            this.javaLocation = javaLocation;
        }
    }

    static int number = 0;

    public static JobDescription prepareJob(String serverAddress,

    public static void cleanup(LinkedList<Job> jobs)

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

Example Code

```
package tutorial

import ibis.ipl.

public class Mai

    static class
        String b
        String j

        Resource
            this
            this
        }
    }

    static int n

    public stati

    public stati

    public stati
}
```

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

    LinkedList<Resource> resources = new LinkedList<Resource>();

    String[] arguments = null;
    String executable = null;
    String inputdir = null;
    String outputdir = null;

    // ...parse parameters here....

    Server reg = new Server(new Properties());

    Properties p = new Properties();
    p.put("ibis.server.address", reg.getAddress());
    p.put("ibis.pool.name", "PILOT");

    JobServer jobServer = new JobServer(executable, arguments, inputdir,
            outputdir, p);

    LinkedList<Job> gatJobs = new LinkedList<Job>();

    for (Resource resource : resources) {
        ResourceBroker broker = GAT.createResourceBroker(new URI(
                resource.brokerURI));
        Job gatJob = broker.submitJob(prepareJob(reg.getAddress(),
                resource.javaLocation));
        gatJobs.add(gatJob);
    }

    jobServer.run();

    cleanup(gatJobs);
    GAT.end();
}
```

Example Code
Deployment

```
package tutorial20.glue;

import ibis.ipl.server.Server;

public cla

    static
        St
        St

        Re


        }
    }

    static

    public

    public

    public
}
```

```
public static JobDescription prepareJob(String serverAddress,
        String javaLocation) throws GATObjectCreationException {

    JavaSoftwareDescription sd = new JavaSoftwareDescription();

    HashMap<String, String> properties = new HashMap<String, String>();
    properties.put("ibis.server.address", serverAddress);
    properties.put("ibis.pool.name", "PILOT");

    sd.setExecutable(javaLocation);
    sd.setJavaClassPath("ipl/*:GAT-examples.jar:.");
    sd.setJavaSystemProperties(properties);
    sd.setJavaMain("tutorial20.glue.PilotJob");

    sd.setStdout(GAT.createFile("stdout-" + number + ".txt"));
    sd.setStderr(GAT.createFile("stderr-" + number + ".txt"));

    sd.addPreStagedFile(GAT.createFile("lib/GAT-examples.jar"));
    sd.addPreStagedFile(GAT.createFile("lib/ipl"), GAT.createFile("ipl"));
    sd.addPreStagedFile(GAT.createFile("log4j.properties"));

    number++;

    return new JobDescription(sd);
}
```

Example Code
Pilot Job

```
package tutorial20.glue;

import ibis.ipl.Ibis;

public class PilotJob {
    Ibis ibis;
    ReceivePort rp;
    SendPort sp;

    PilotJob() throws Exception {

    Job getWork(Result previousResult) throws Exception {

    void run() throws Exception {

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

Example Code

Pilot Job

```
package tutorial20.glue;

import ibis.ipl.Ibis;

public class
    Ibis ibi
    ReceiveP
    SendPort

    PilotJob

    Job getW

    void run() throws Exception {

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

```
public static void main(String[] args) {
    try {
        new PilotJob().run();
    } catch (Exception e) {
        System.err.println("PilotJob failed: " + e);
        e.printStackTrace(System.err);
    }
}
```

Example Code

Pilot Job

```
PilotJob() throws Exception {
    ibis = IbisFactory.createIbis(Shared.ibisCapabilities, null,
            Shared.portTypeServer, Shared.portTypeSlave);

    IbisIdentifier server = ibis.registry().getElectionResult("JobServer");

    rp = ibis.createReceivePort(Shared.portTypeSlave, "receiver");
    rp.enableConnections();

    sp = ibis.createSendPort(Shared.portTypeServer);
    sp.connect(server, "receiver");
}
```

```
package

import

public

    PilotJob() throws Exception {

    Job getWork(Result previousResult) throws Exception {

    void run() throws Exception {

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

Example Code

Pilot Job

```
PilotJob() throws Exception {
    ibis = IbisFactory.createIbis(Shared.ibisCapabilities, null,
            Shared.portTypeServer, Shared.portTypeSlave);

    IbisIdentifier server = ibis.registry().getElectionResult("JobServer");

    rp = ibis.createReceivePort(Shared.portTypeSlave, "receiver");
    rp.enableConnections();

    sp = ibis.createSendPort(Shared.portTypeServer);
    sp.connect(server, "receiver");
}
```

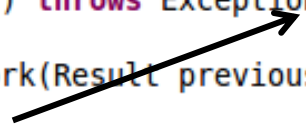
```
package

import

public

    PilotJob() throws Exception {

    Job getW

    void run

    public

}
```

```
static PortType portTypeSlave = new PortType(
        PortType.COMMUNICATION_RELIABLE, PortType.SERIALIZATION_OBJECT,
        PortType.RECEIVE_EXPLICIT, PortType.CONNECTION_ONE_TO_ONE);

static PortType portTypeServer = new PortType(
        PortType.COMMUNICATION_RELIABLE, PortType.SERIALIZATION_OBJECT,
        PortType.RECEIVE_EXPLICIT, PortType.CONNECTION_MANY_TO_ONE);

static IbisCapabilities ibisCapabilities = new IbisCapabilities(
        IbisCapabilities.ELECTIONS_STRICT);
```

Example Code
Pilot Job

```
package tutorial20.glue;

import ibis.ipl.Ibis;

public class PilotJob {
    Ibis ibis;
    ReceivePort rp;
    SendPort sp;

    PilotJob() throws Exception

    Job getWork(Result previous

    void run() throws Exception {

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

```
void run() throws Exception {
    Job job = getWork(new Result());

    while (!job.empty) {
        Result result = job.execute();
        job = getWork(result);
    }

    ibis.end();
}
```

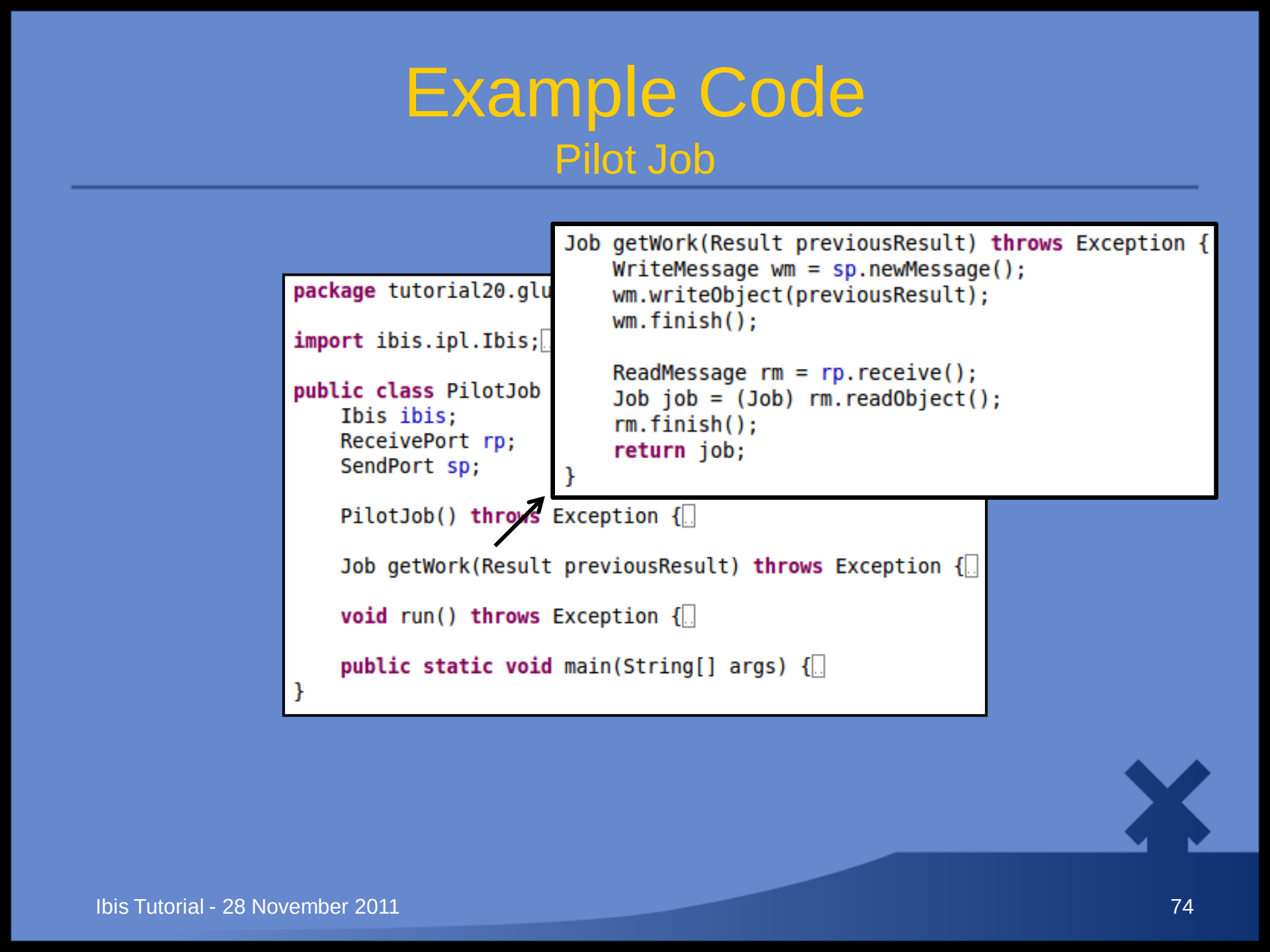


Example Code

Pilot Job

```
package tutorial20.glu

import ibis.ipl.Ibis;

public class PilotJob
    Ibis ibis;
    ReceivePort rp;
    SendPort sp;

    PilotJob() throws Exception {

    Job getWork(Result previousResult) throws Exception {

    void run() throws Exception {

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

```
Job getWork(Result previousResult) throws Exception {
    WriteMessage wm = sp.newMessage();
    wm.writeObject(previousResult);
    wm.finish();

    ReadMessage rm = rp.receive();
    Job job = (Job) rm.readObject();
    rm.finish();
    return job;
}
```



Example Code

Job Server Code

```
package tutorial20.glue;

import java.io.File;

public class JobServer {
    String inputDir, outputDir;
    LinkedList<Job> jobs = new LinkedList<Job>();
    Ibis ibis;
    ReceivePort rp;
    HashMap<IbisIdentifier, SendPort> workers =
            new HashMap<IbisIdentifier, SendPort>();

    JobServer(String executable, String[] arguments, String inputDir,

    SendPort getSendPort(IbisIdentifier target) throws IOException {

    void removeSendPort(IbisIdentifier target) throws IOException {

    void sendReply(IbisIdentifier target, Job job) throws IOException {

    void processResult(Result result) throws IOException {

    void handleRequest() throws IOException, ClassNotFoundException {

    void run() throws Exception {
}
```

Example Code

```
JobServer(String executable, String[] arguments, String inputDir,
        String outputDir, Properties p) throws Exception {
    this.inputDir = inputDir;
    this.outputDir = outputDir;

    for (String file : Shared.listFiles(inputDir, ".jpg"))
        jobs.add(new Job(executable, arguments, file, "out-" + file));

    ibis = IbisFactory.createIbis(Shared.ibisCapabilities, p, true, null,
            Shared.portTypeServer, Shared.portTypeSlave);
    ibis.registry().elect("JobServer");

    rp = ibis.createReceivePort(Shared.portTypeServer, "receiver");
    rp.enableConnections();
}
```

```
package tutorial20

import java.io.Fil

public class JobSe
    String inputDi
    LinkedList<Job
    Ibis ibis;
    ReceivePort rp
    HashMap<IbisId
            ne Ha

    JobServer(String executable, String[] arguments, String inputDir,

    SendPort getSendPort(IbisIdentifier target) throws IOException {

    void removeSendPort(IbisIdentifier target) throws IOException {

    void sendReply(IbisIdentifier target, Job job) throws IOException {

    void processResult(Result result) throws IOException {

    void handleRequest() throws IOException, ClassNotFoundException {

    void run() throws Exception {
}
```

Example Code

Job Server Code

```
package tutorial20.glue;

import java.io.File;

public class JobServer {
    String inputDir, outputDir;
    LinkedList<Job> jobs = new LinkedList<Job>();
    Ibis ibis;
    ReceivePort rp;
    HashMap<IbisIdentifier, SendPort> workers =
            new HashMap<IbisIdentifier, SendPort>();

    JobServer(String executable, String[] arguments, String inputDir,

    SendPort getSendPort(IbisIdentifier target) throws IOException {

    void removeSendPort(IbisIdentifier target) throws IOException {

    void sendRep

    void process

    void handleR

    void run() throws Exception {
}
```

```
void run() throws Exception {
    while (jobs.size() > 0 || workers.size() > 0) {
        handleRequest();
    }
}
```

Example Code

Job Server Code

```
package tutorial20.glue;

import java.io.File;

public class JobServer {
    String inputDir, outputDir;
    LinkedList<Job> jobs = new LinkedList<Job>();
    Ibis ibis;
    ReceivePort rp;
    HashMap<IbisIdentifier, SendPort> workers =
            new HashM

    JobServer(String

    SendPort getSendP

    void removeSendPo

    void sendReply(Ib

    void processResul

    void handleRequest() throws IOException, ClassNotFoundException {

    void run() throws Exception {
}
```

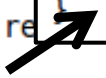
```
void handleRequest() throws IOException, ClassNotFoundException {
    ReadMessage rm = rp.receive();
    IbisIdentifier target = rm.origin().ibisIdentifier();
    Result result = (Result) rm.readObject();
    rm.finish();

    Job job = (jobs.size() > 0 ? jobs.removeFirst() : new Job());

    sendReply(target, job);
    processResult(result);
}
```

Example Code

Job Server Code

```
package tutorial20.glue;

import java

public clas
    String
    LinkedL
    Ibis ib
    Receive
    HashMap


    JobServ

    SendPor

    void re

    void sendReply(IbisIdentifier target, Job job) throws IOException {

    void processResult(Result result) throws IOException {

    void handleRequest() throws IOException, ClassNotFoundException {

    void run() throws Exception {
}
```

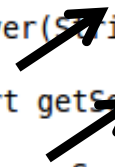
```
void sendReply(IbisIdentifier target, Job job) throws IOException {
    SendPort sp = getSendPort(target);

    if (!job.empty) {
        job.setInput(Shared.read(inputDir + File.separator + job.inputFile));
    }

    WriteMessage wm = sp.newMessage();
    wm.writeObject(job);
    wm.finish();

    if (job.empty) {
        removeSendPort(target);
    }
}
```



Example Code

```
package tutorial20.

import java.io.File

public class JobSer
    String inputDir
    LinkedList<Job>
    Ibis ibis;
    ReceivePort rp;
    HashMap<IbisIde
            new Has

    JobServer(Strin

    SendPort getSendPort(IbisIdentifier target) throws IOException {

    void removeSendPort(IbisIdentifier target) throws IOException {

    void sendReply(IbisIdentifier target, Job job) throws IOException {

    void processResult(Result result) throws IOException {

    void handleRequest() throws IOException, ClassNotFoundException {

    void run() throws Exception {
}
```

```
SendPort getSendPort(IbisIdentifier target) throws IOException {
    SendPort sp = workers.get(target);
    if (sp == null) {
        sp = ibis.createSendPort(Shared.portTypeSlave);
        sp.connect(target, "receiver");
        workers.put(target, sp);
    }
    return sp;
}

void removeSendPort(IbisIdentifier target) throws IOException {
    SendPort sp = workers.remove(target);

    if (sp != null) sp.close();
}
```



Live DEMO

Conclusions

- IPL+SmartSockets = Jungle proof communication
  - SmartSockets solves low-level connectivity problems
  - IPL offers high-level communication primitives and resource tracking
- Adding JavaGAT makes it very easy to create applications that can on a jungle of resources
  - Pilot Job framework is a very simple example

Further improvements ?

- Improve hubs placement for IPL applications
  - Need automatic hub deployments
- Most information needed by JavaGAT is static
  - Individual resources don't change much over time
  - Neither do applications
  - Can't we store this info in configuration files?
- Can't we do all this in a "point and click" style ?

- Solution: Ibis Deploy (after the break)

Elections

- JEL also offers an 'election'
  - Allows a group to determine who's special
  - Ranks don't work in a malleable set of resources!
- Each election
  - Has a name (String)
  - Produces IbisIdentifier of the winner
  - Is not democratic
  - You can also be 'an observer'

Ibis Capabilities

- When initializing the application must specify:
  - The PortTypes it is going to use
    - Defines what kind of communication you need
  - The Resource tracking behaviour it needs
    - Defines what level of malleability you need
  - Optional: the preferred IPL implementation
    - SmartSockets, MX, MPI, etc.
- This allows the runtime to check if the requested combination is feasable