



Robust Communication

Jason Maassen
jason@cs.vu.nl



vrije Universiteit



vl·e

Introduction

- Before lunch we looked at
 - Programming models
 - Grid applications
 - Problems encountered in Grid computing

- Many of the problems are related to communication!



Basic problems

- Many sites have connectivity issues
 - Firewalls
 - Network Address Translation (NAT)
 - Non-routed networks
 - Multi homing
 - Mis-configured machines
 - ...
- This makes it hard to use a combination of sites



In addition ...

- We need more advanced features:
 - Resource Tracking:
 - Malleability: machines come and go during the application lifetime
 - Fault Tolerance: machines may crash at any time
 - Efficient serialization of complex data structures
 - Multicast or many-to-one comm.
 - ...

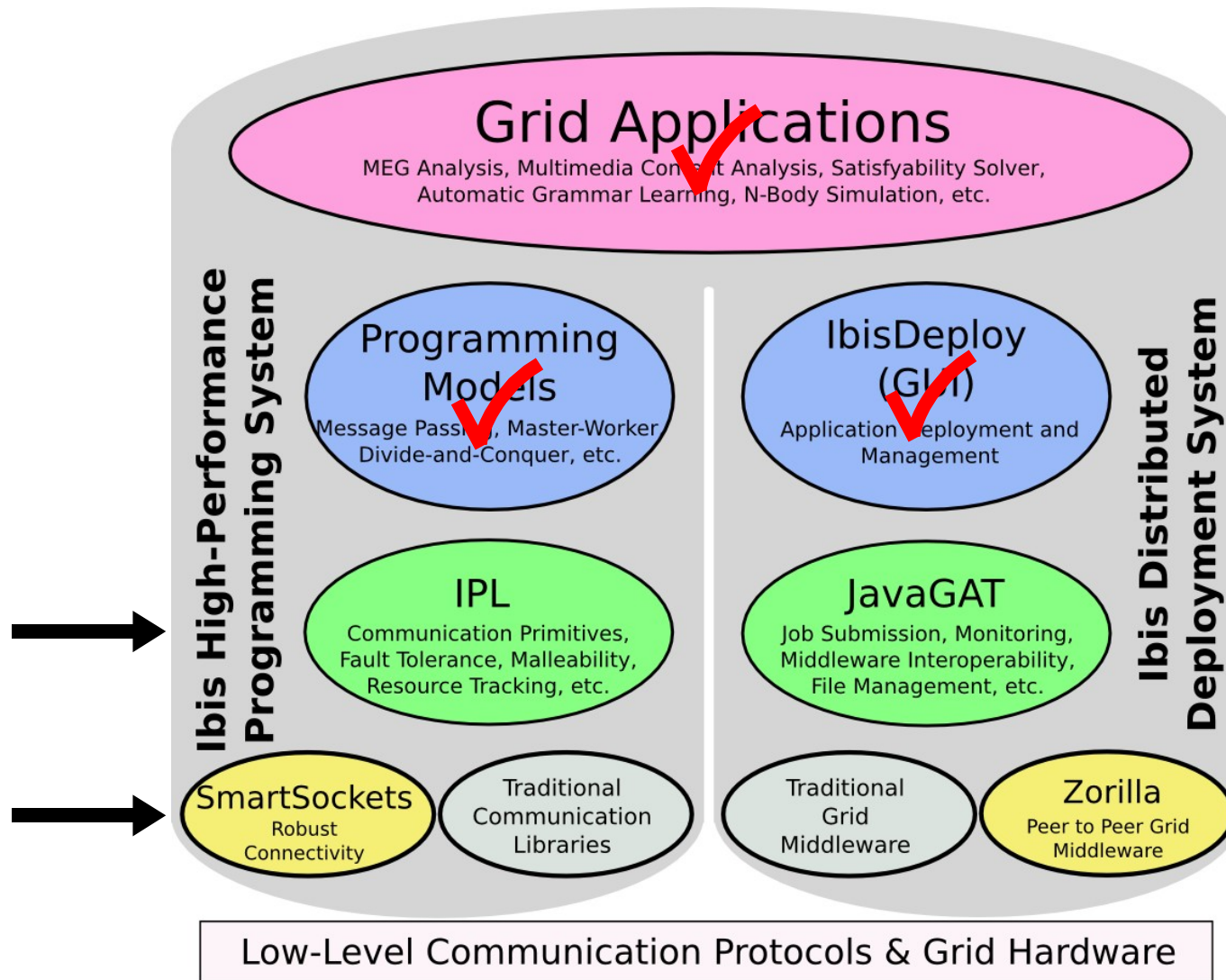


Existing libraries

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



Overview



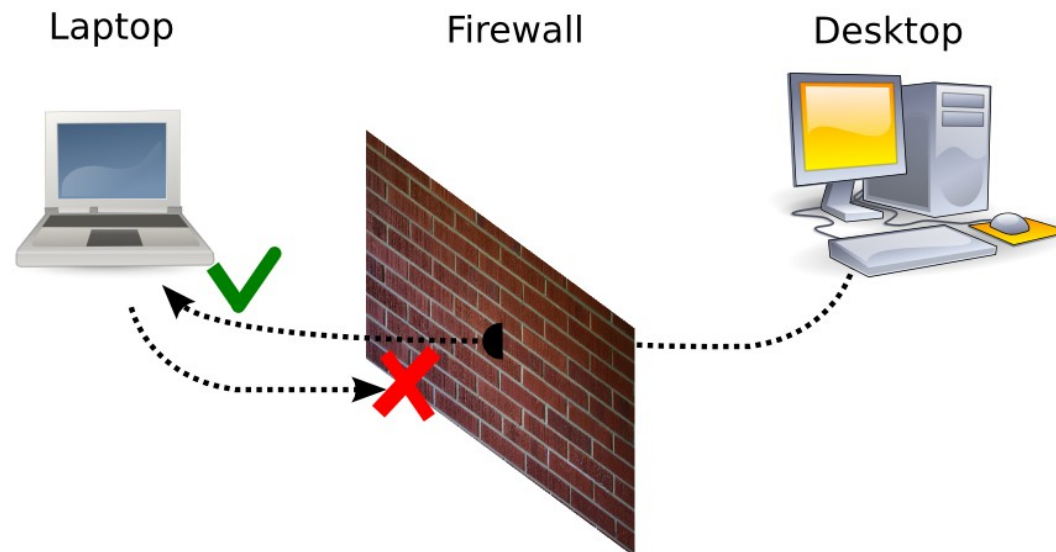
SmartSockets

- The SmartSockets library
 - Offers a socket-like interface
 - Addressing is different
 - Detects connectivity problems
 - Tries to solve them automatically
 - With as little help from the user as possible
 - Integrates existing and several new solutions into one library
- User friendly connection setup!



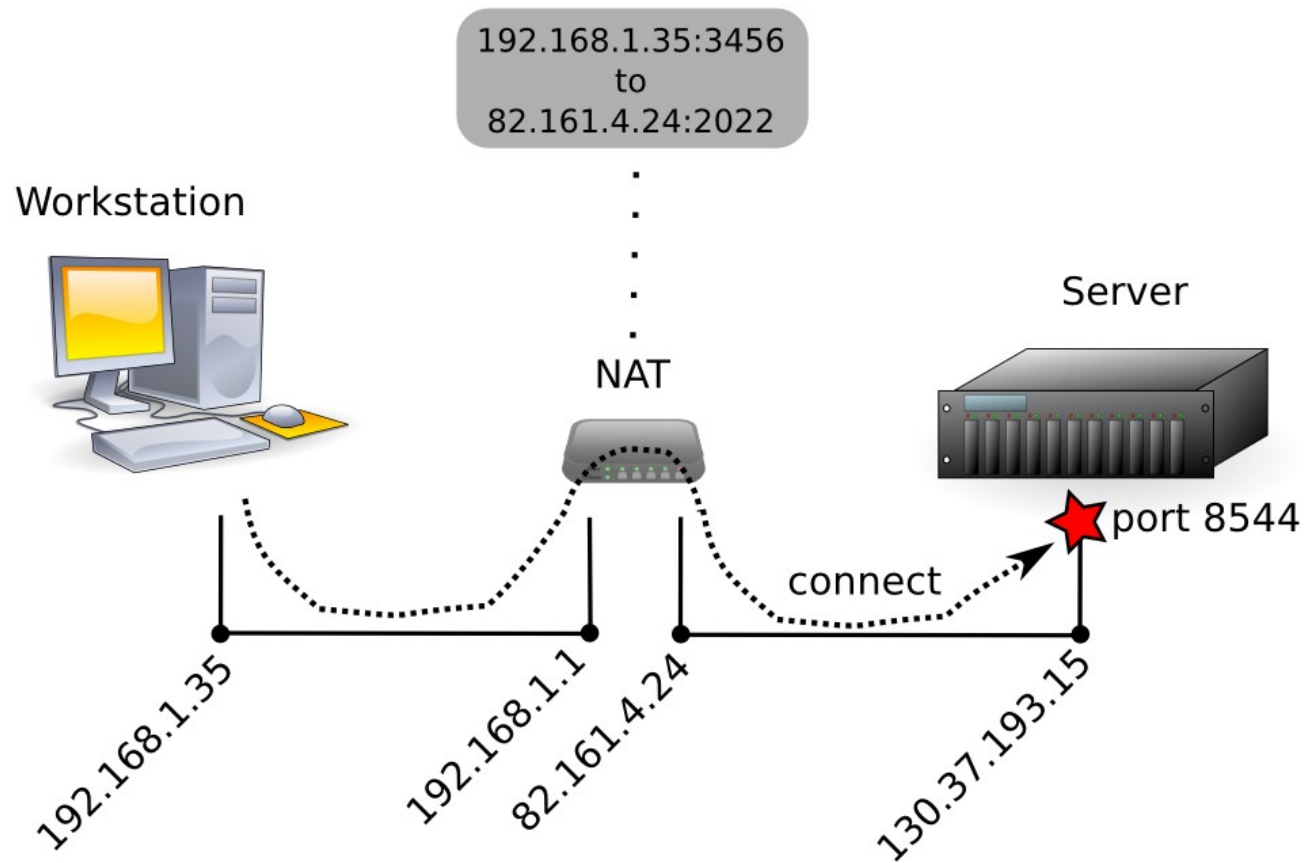
Problem 1: Firewalls

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



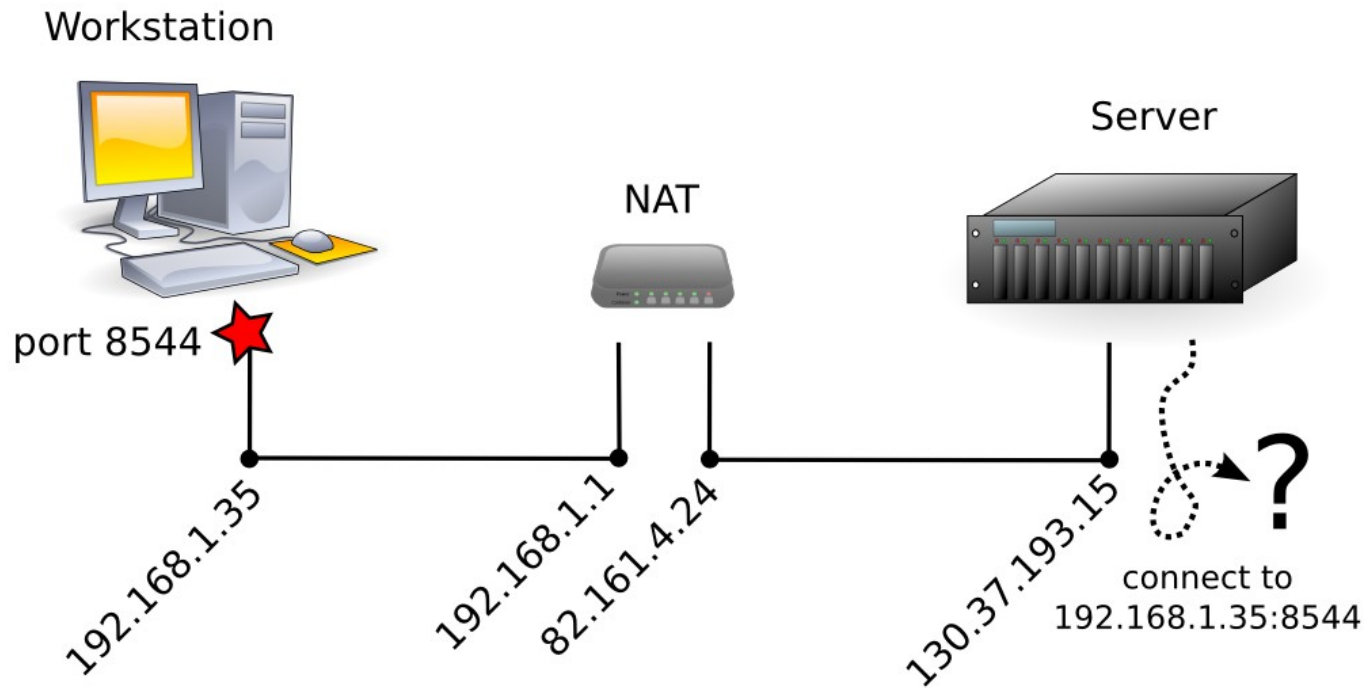
Problem 2: Network Address Translation

- Maps IP from one range into another
 - Works fine for outgoing connections



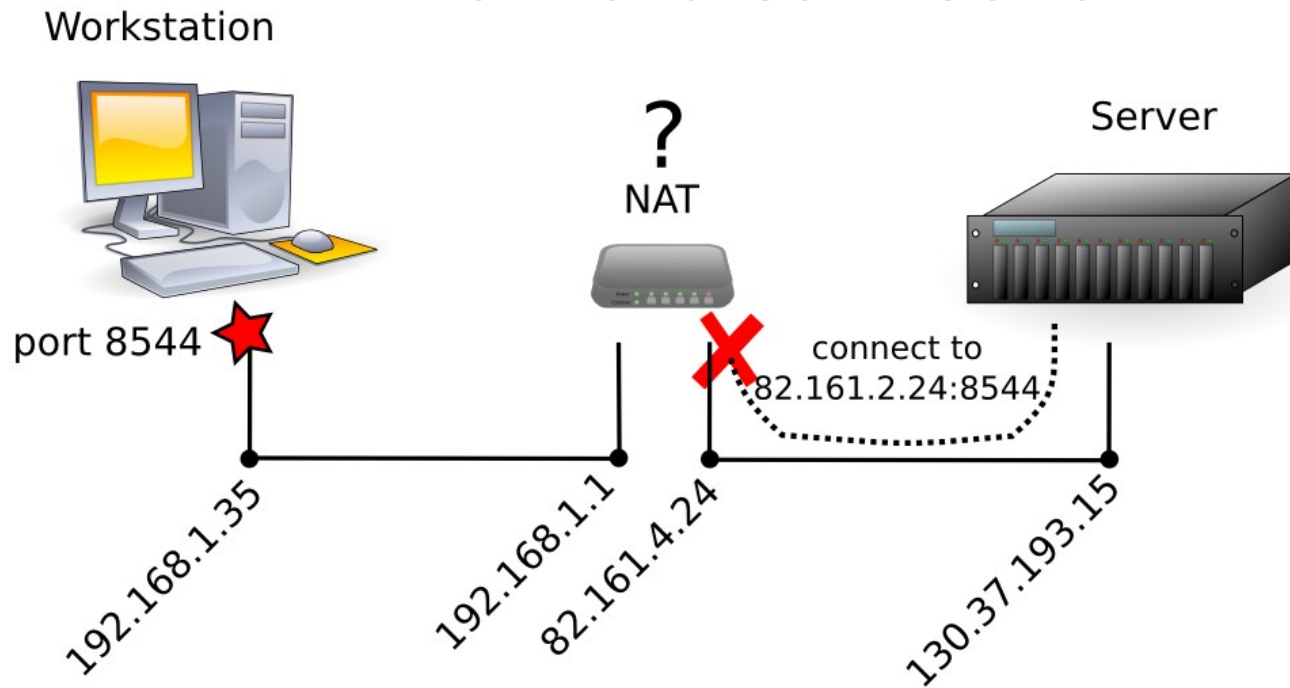
Problem 2: Network Address Translation

- Problem: incoming connections
 - Target address is invalid on internet



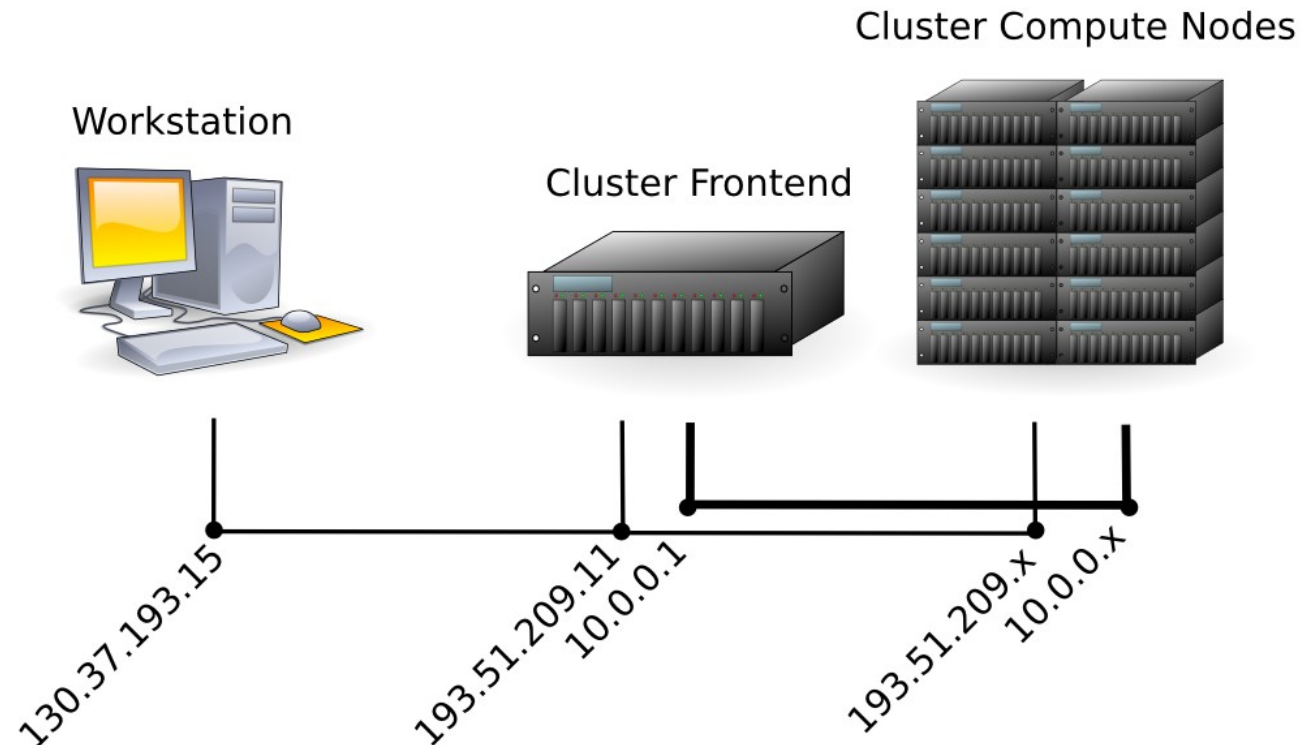
Problem 2: Network Address Translation

- Problem: incoming connections
 - Target address is invalid on internet
 - NAT device does not know where to forward connection



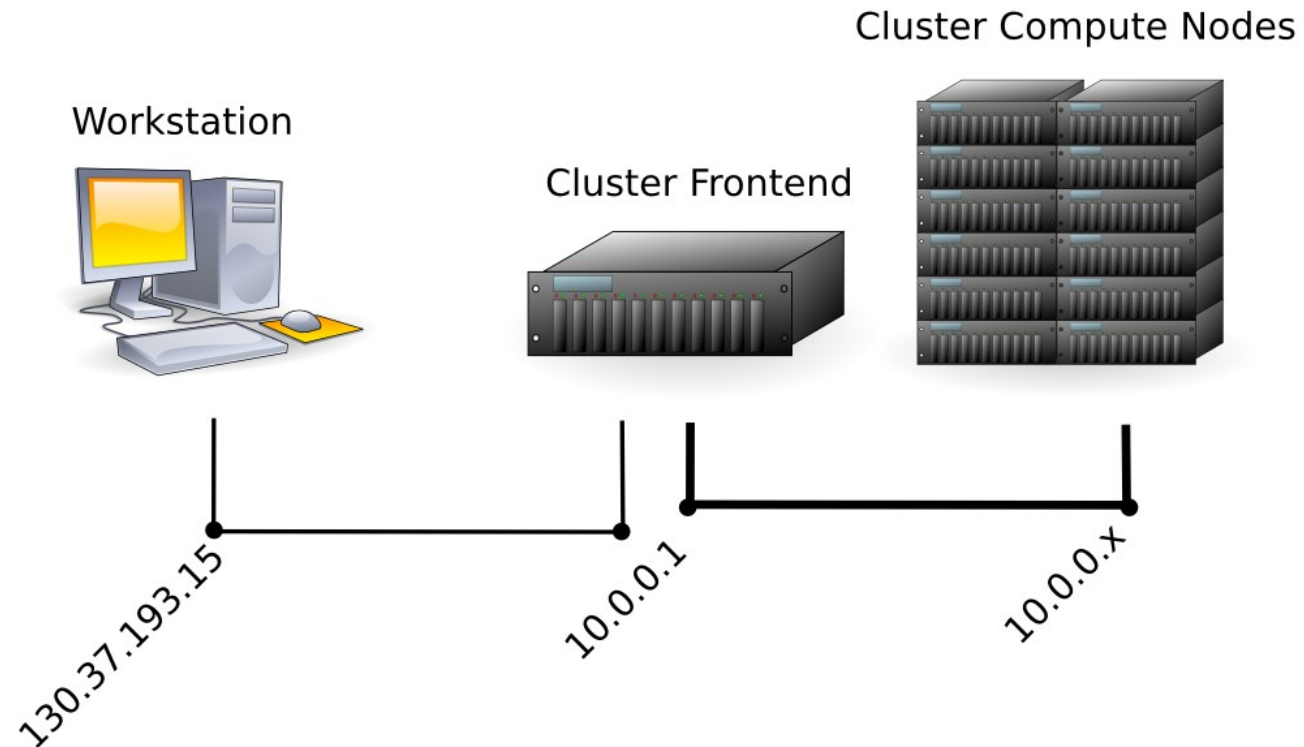
Problem 3: Multi Homing

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



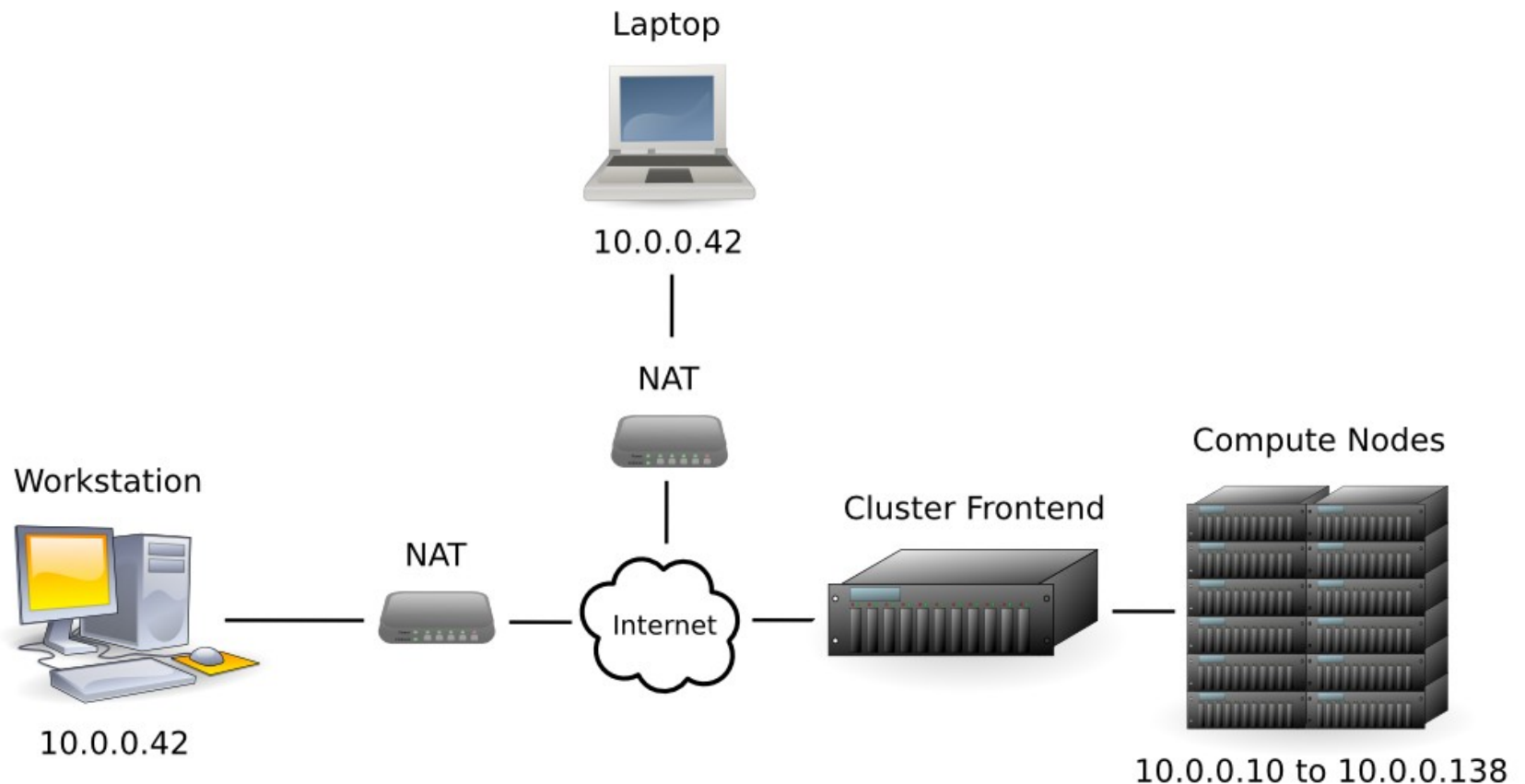
Problem 4: Non-routed Networks

- Some sites do not route between the local network and internet
 - Only the frontend is reachable



Problem 5: Machine Identification

- NAT/non-routed networks lead to machine identification problems



Smart Addressing

- SmartSockets extends addressing
 - Not just a single IP:port combination
- Add all machine addresses
- Optionally add extra information
 - External address + port (for NAT)
 - SSH contact information
 - UUID (if entire address is private)



Addressing Examples

130.37.193.15-42611
public address

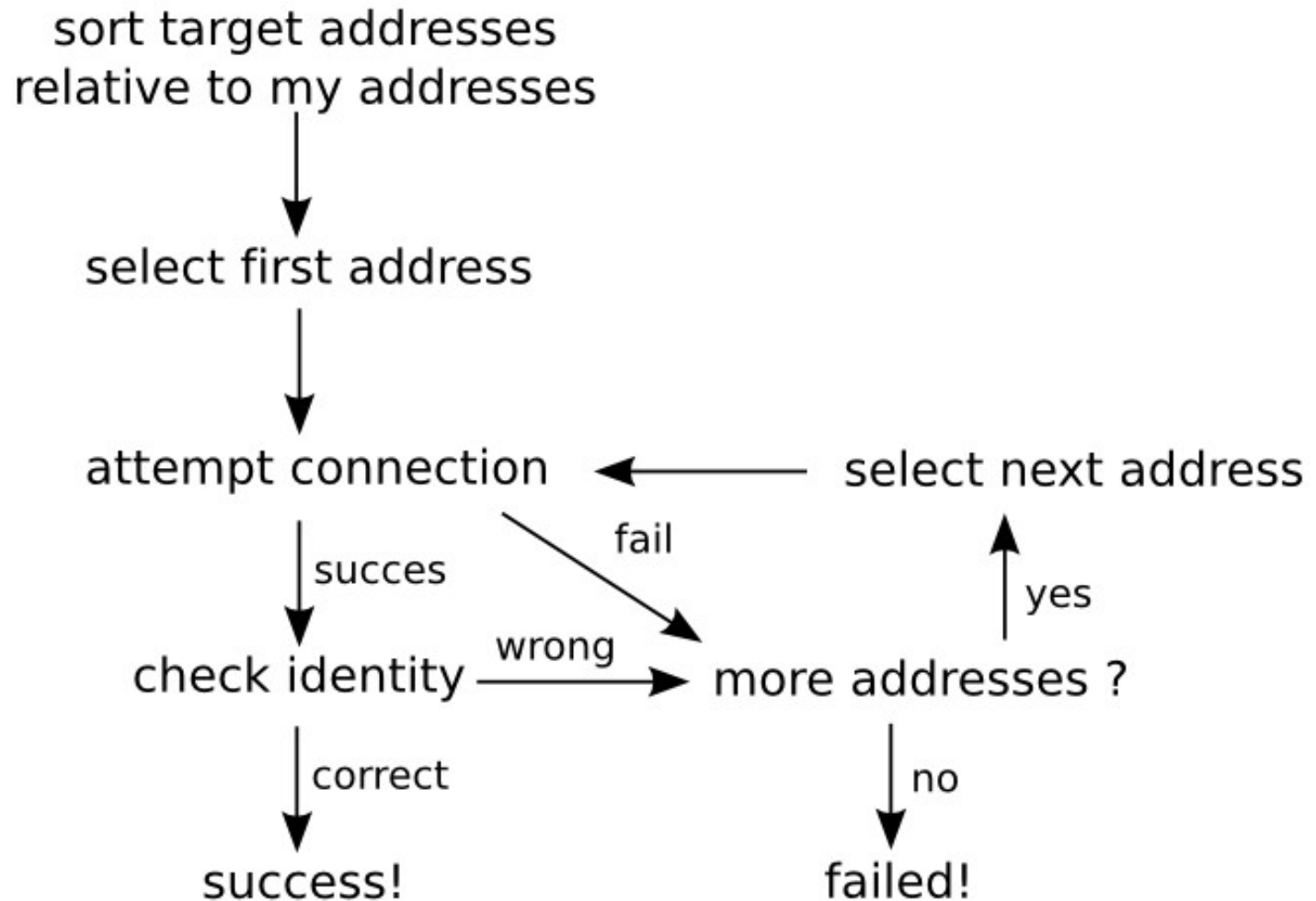
130.37.193.15-42611~jason
public address SSH user

130.37.197.201-42611/10.153.0.201/10.141.0.73-42611
public address private addresses

{82.161.4.24-20122}/192.168.1.36-42611#a6.e5.01.1a.ad.f1
external address with port forwarding private address UUID unique to host



Creating a Connection

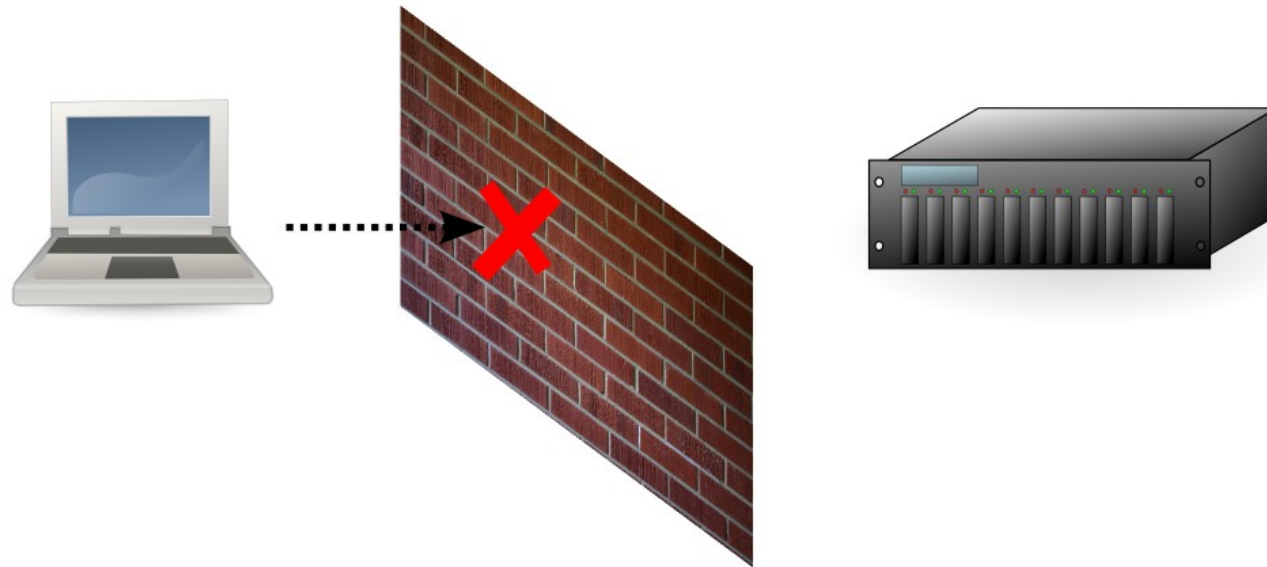


Using smart addresses...

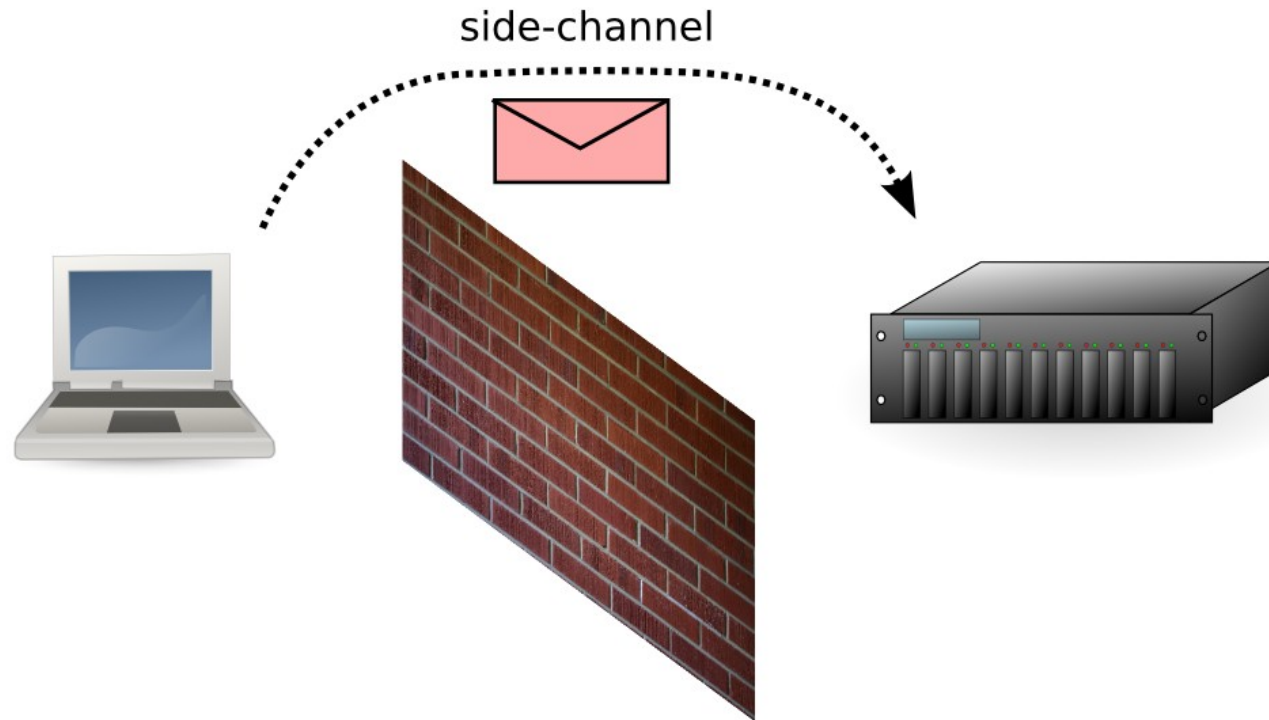
- This solves multi homing and machine identification problems (3 & 5)
- Assumes anyone can create a connection
 - This will not help when the target is behind a NAT / Firewall
- To solve this we need a side channel!



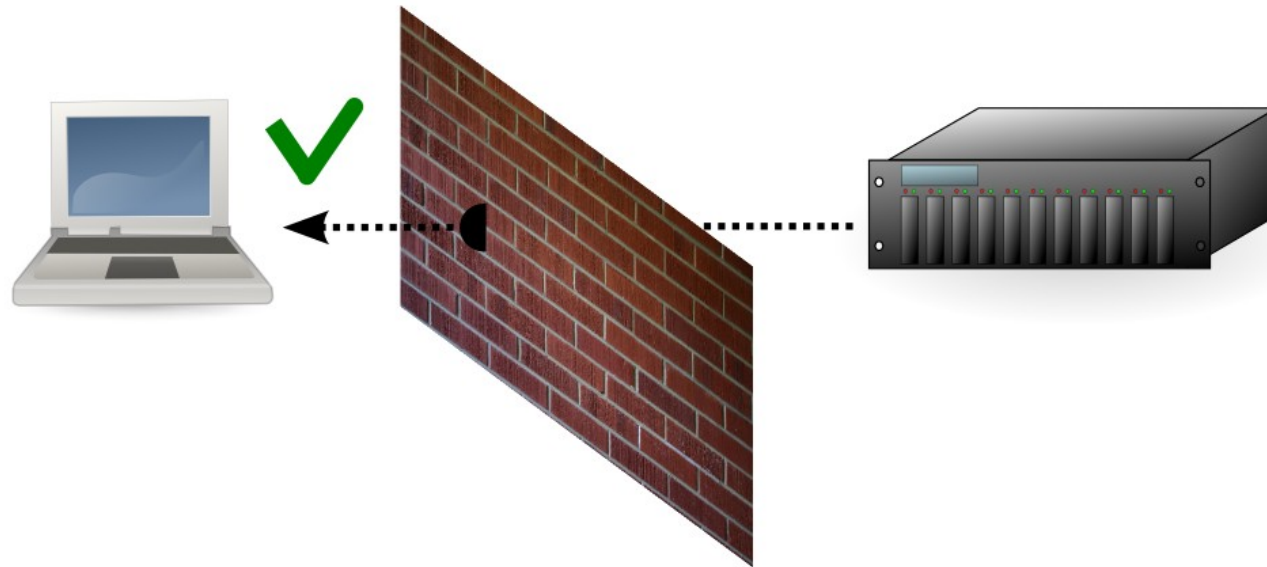
Side channel (example)



Side channel (example)



Side channel (example)



Side channel

- SmartSockets uses set of *hubs* to implement a *side channel*
 - Support processes for the application
 - Started in advance
- Hubs are run on machines with 'more connectivity'
 - Such as cluster frontends, 'open' machines, etc.

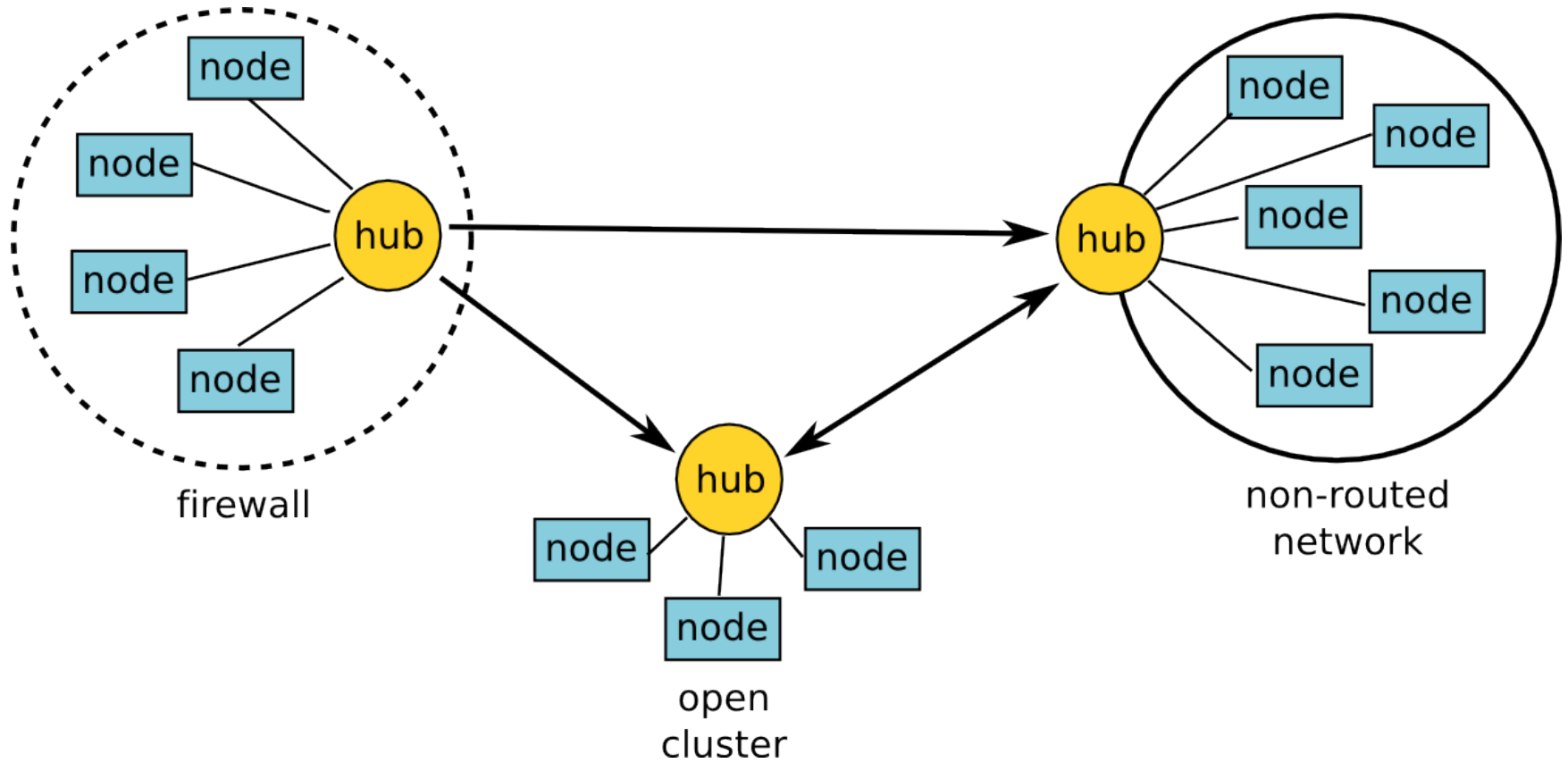


Hubs

- Hub connect to each other
 - Need to set up spanning tree (or better)
 - Use direct connections and SSH tunnels
 - Gossip information and client messages
 - Automatically discover new hubs and routes
- Clients connect to a 'local' hub
 - When needed, use network of hubs as side channel for connection setup



Example

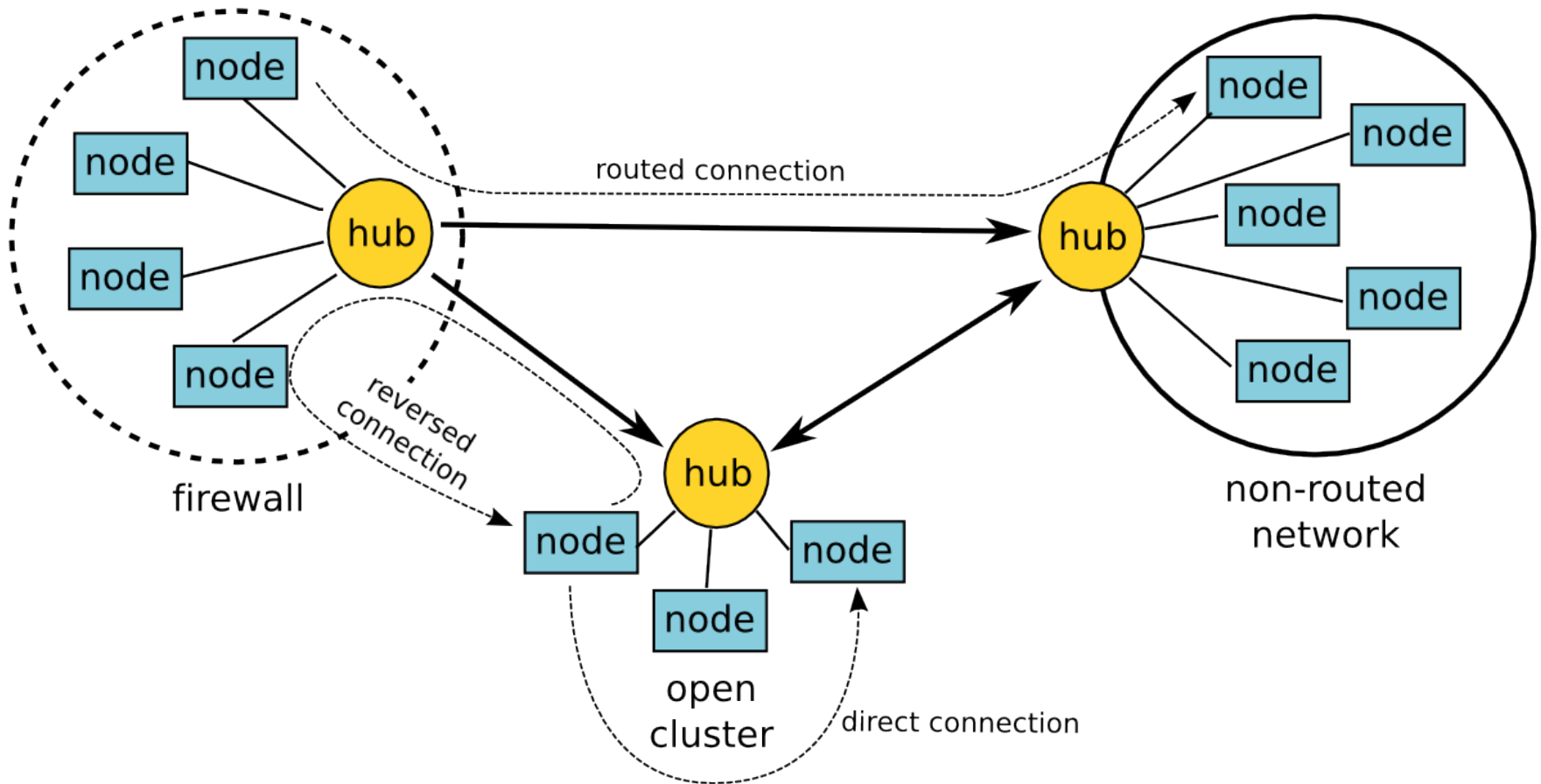


Reverse

- Reverse direction of connection setup
 - Send message to target using hub
 - Wait for incoming (direct) connection
- Results in **direct connection**
 - Only connection setup is different
- Solves Firewall/NAT problems (1 & 2)
 - Assumes one side can create connection
 - Does not work with multiple firewalls or NATs, or non-routed networks



Example Cont'd

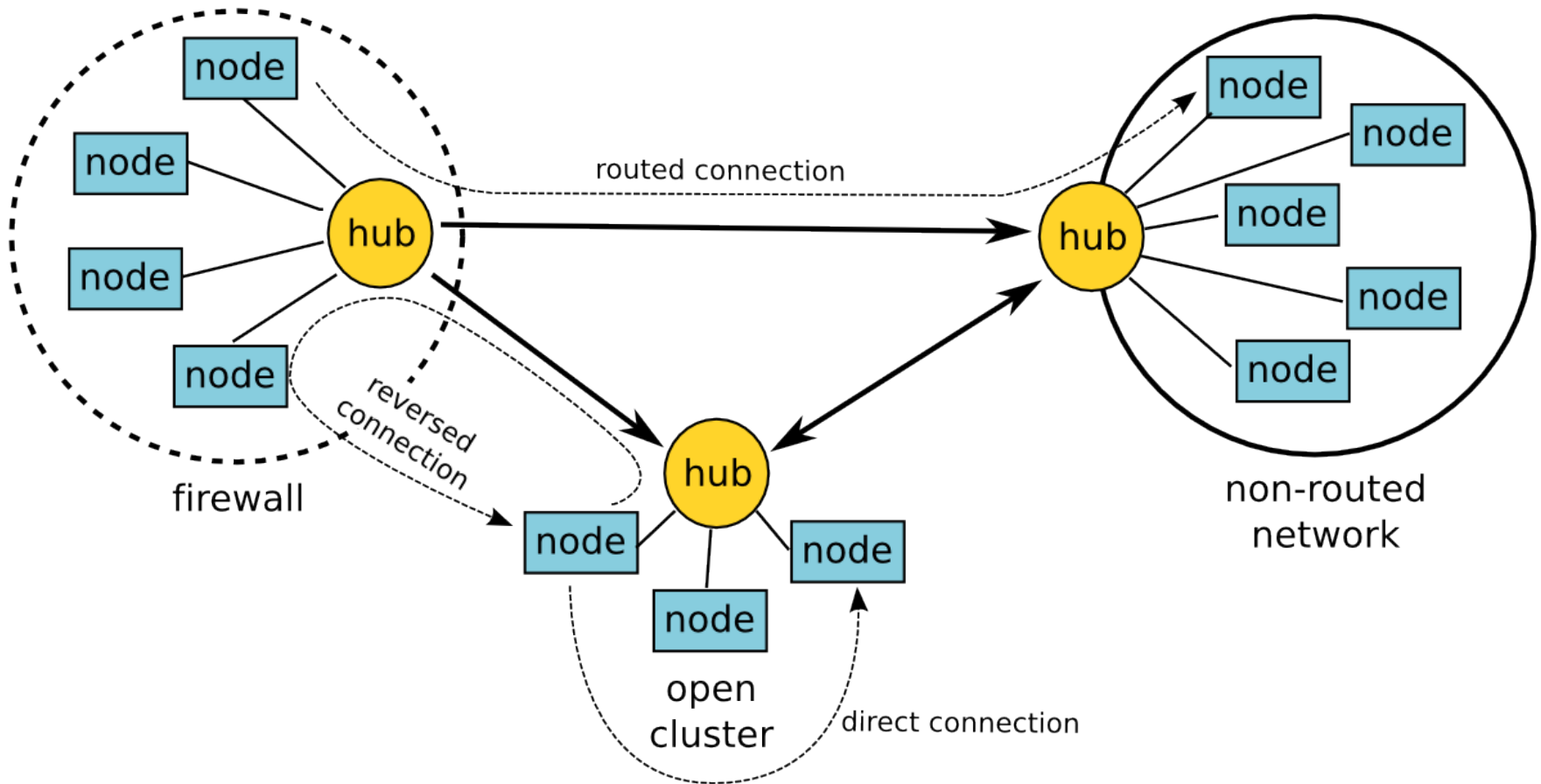


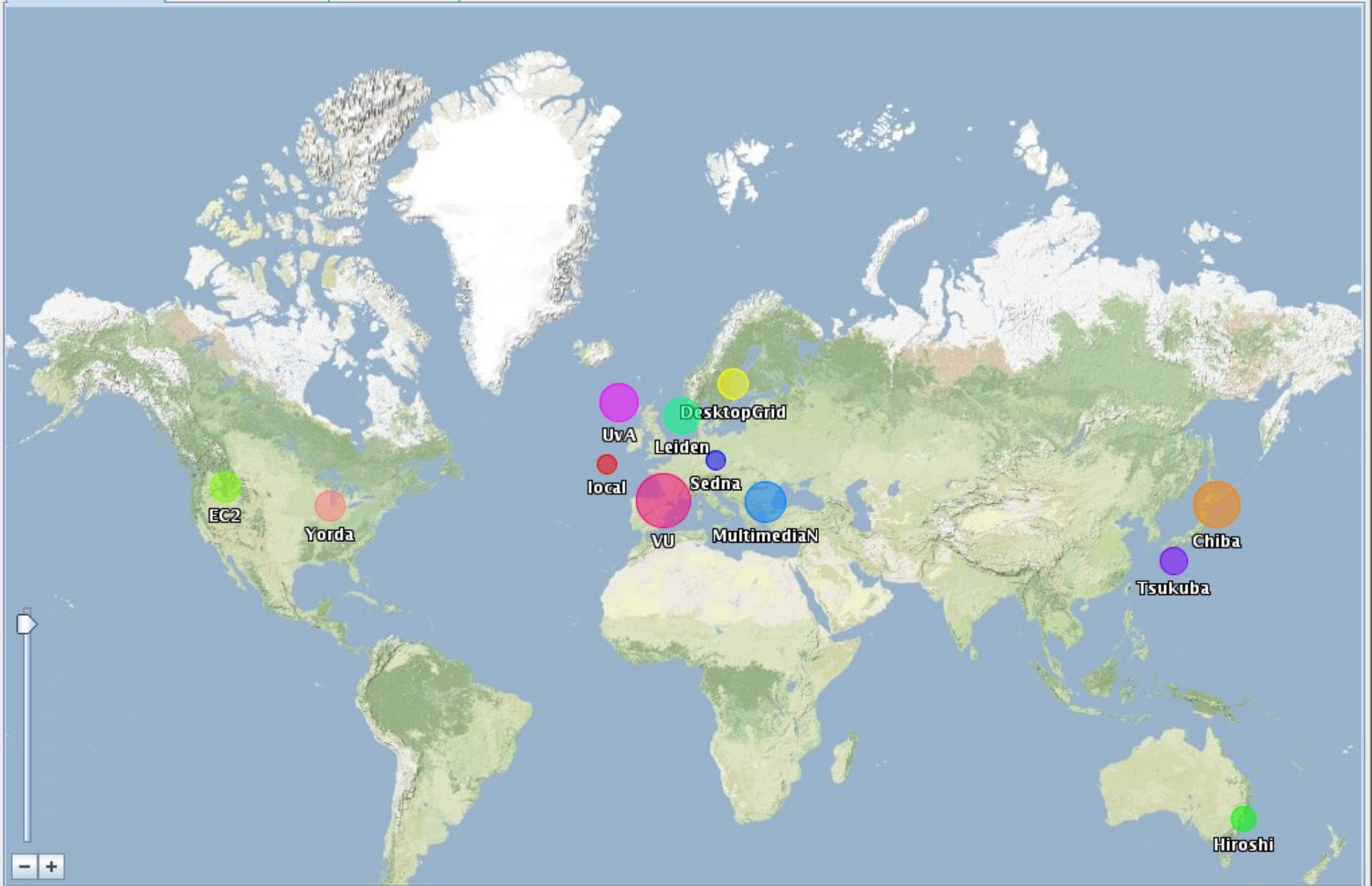
Routed

- Create *virtual connection* using hubs
 - Forward all data over side channel
- Results in **indirect connection**
 - Usually lower performance
 - Still looks like a socket!
- Solves non-routed problem (4)
 - Also solves the multi-firewall/NAT case



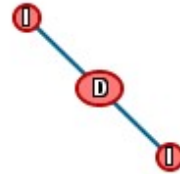
Example Cont'd



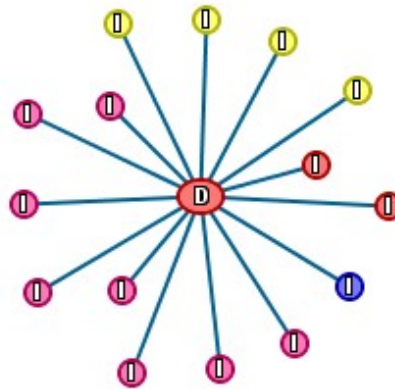


Possible connections

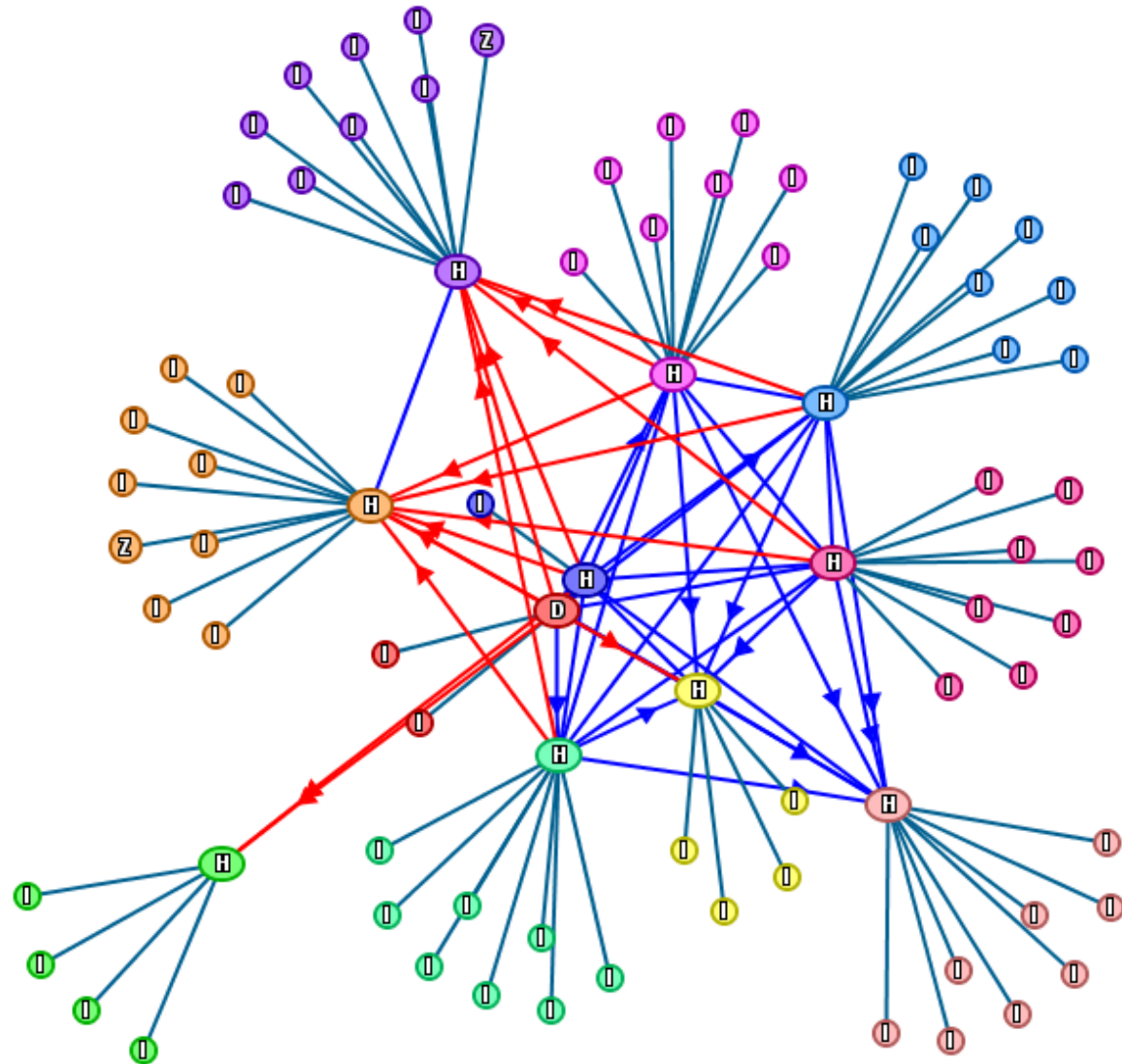
no SmartSockets & desktop w. firewall



Possible connections no SmartSockets & desktop open



Possible connections with SmartSockets



Summary

- Communicating on Grids is hard
 - Many connectivity problems occur
 - Takes a lot of work to find the problems and work around them
- SmartSockets reduces this into a single problem:
 - **How to set up a spanning tree of hubs**
- The rest is done automatically!

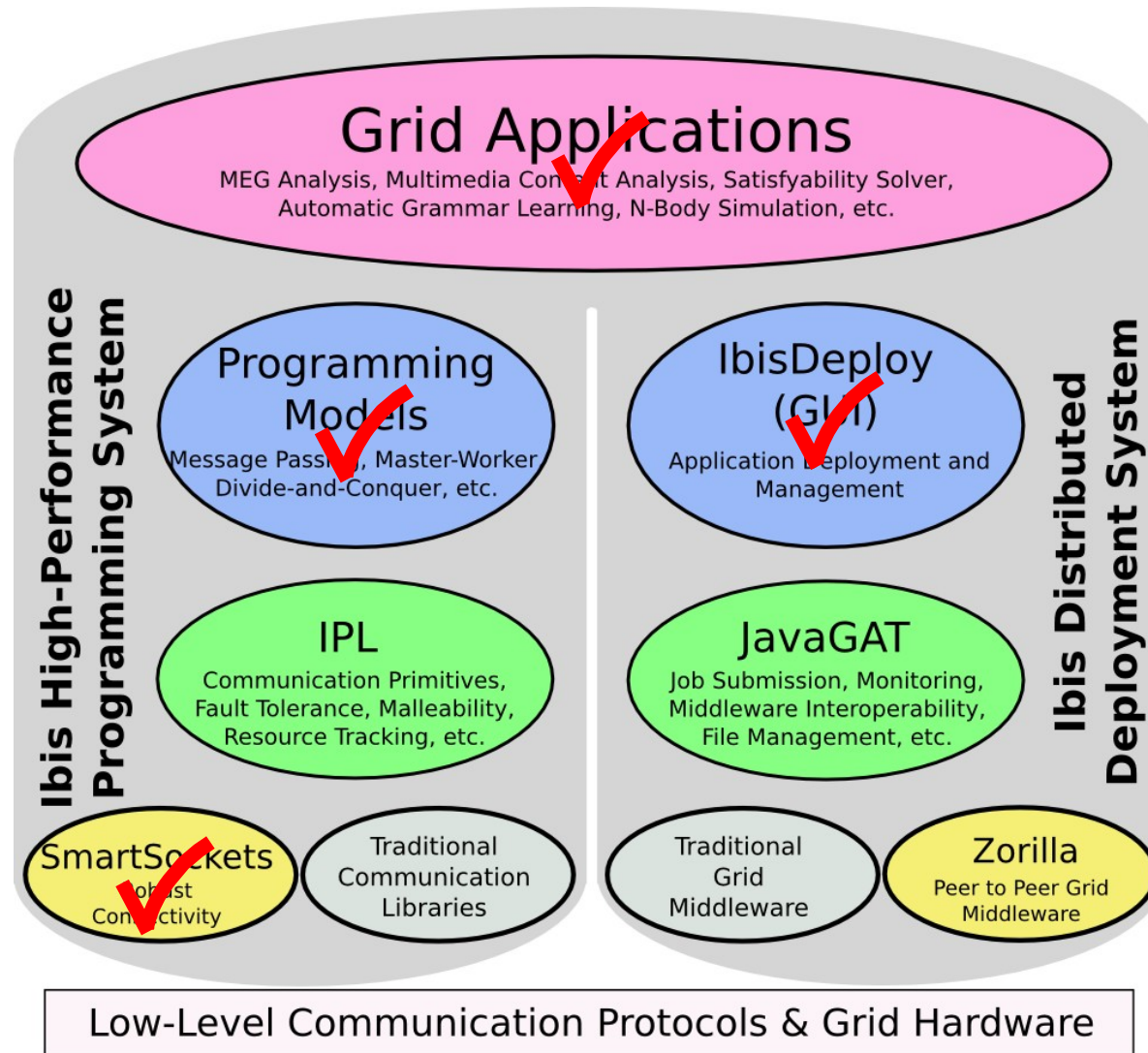


Next step

- But that's not all...
- Sockets is too low level for daily use
- Applications need:
 - Resource Tracking:
 - Malleability / fault-tolerance
 - Multicast / many-to-one
 - Sending complex data types
 - etc...



Overview



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

- Simple API for Grid Communication
 - Flexible communication model
 - connection oriented messaging
 - abstract addressing scheme
 - Resource tracking
 - notifications when machines join/leave/crash
 - Efficient serialization
 - send bytes, doubles, objects, etc.
- Portable!



Abstract addressing ***(IbisIdentifiers)***

- IbisIdentifier:
 - Abstract 'machine address' object
 - Uniquely identifies an IPL instance
 - Hides network specific details
 - SmartSockets addresses, hostnames, IP addresses, MPI ranks, etc
- Results in more portable applications
 - Implemented on top of TCP, MPI, MX...



IbisIdentifiers

- Why don't we use ranks ?
 - What happens to the ranks when machines leave or crash ?
- Ranks can be implemented using IbisIdentifiers!



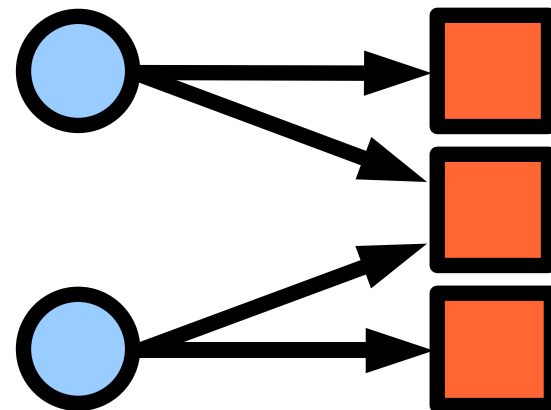
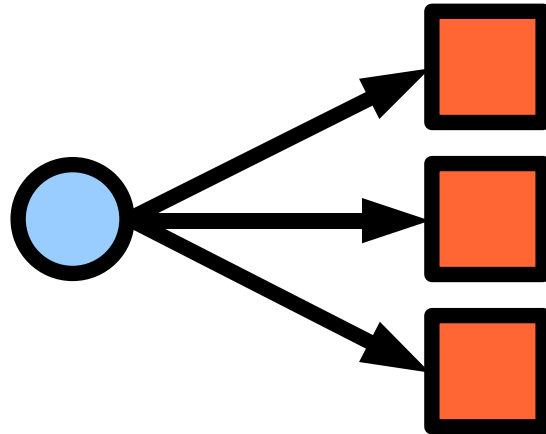
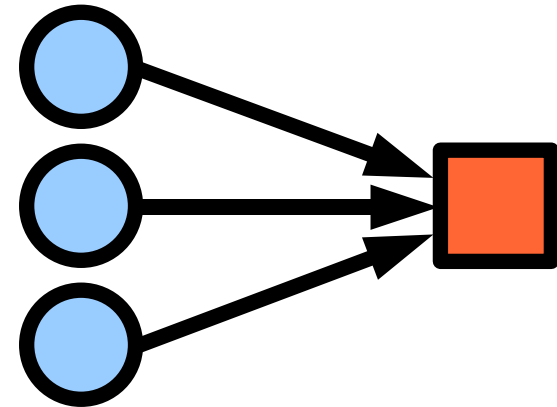
Communication

- Simple communication model
- Unidirectional pipes
- Two end points
- Connection oriented
 - allows streaming (good with high latency)



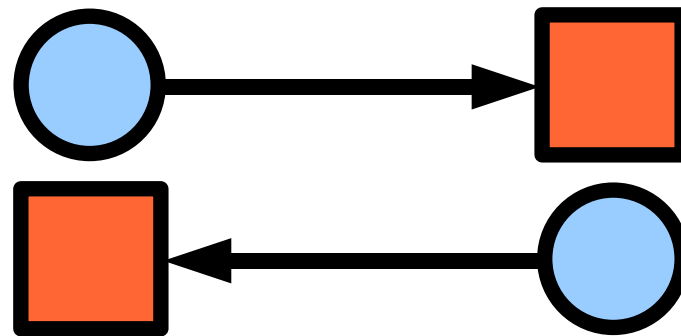
Send & receive ports

- Can be connected in arbitrary ways



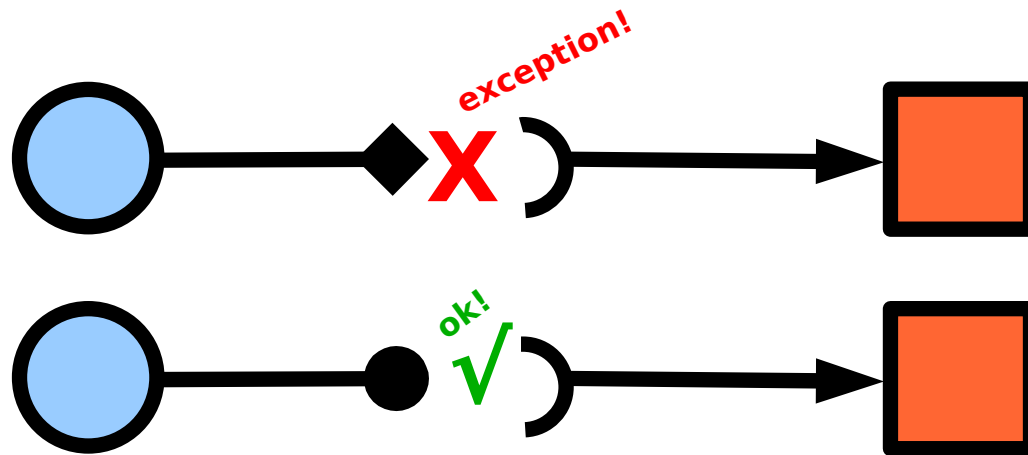
Send & receive ports

- Simplicity may cause some additional administration...
 - Example: need two pairs for RPC / RMI



Port Types

- All 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, data, object
 - 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 ?
 - Bytes only ?
 - Can save a lot complexity!



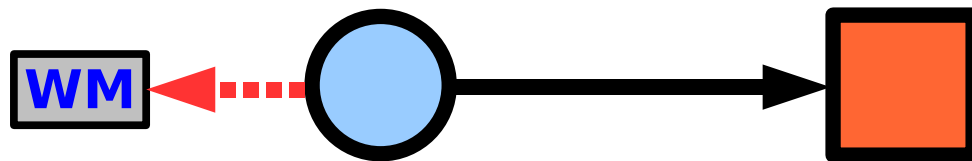
Messages

- 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



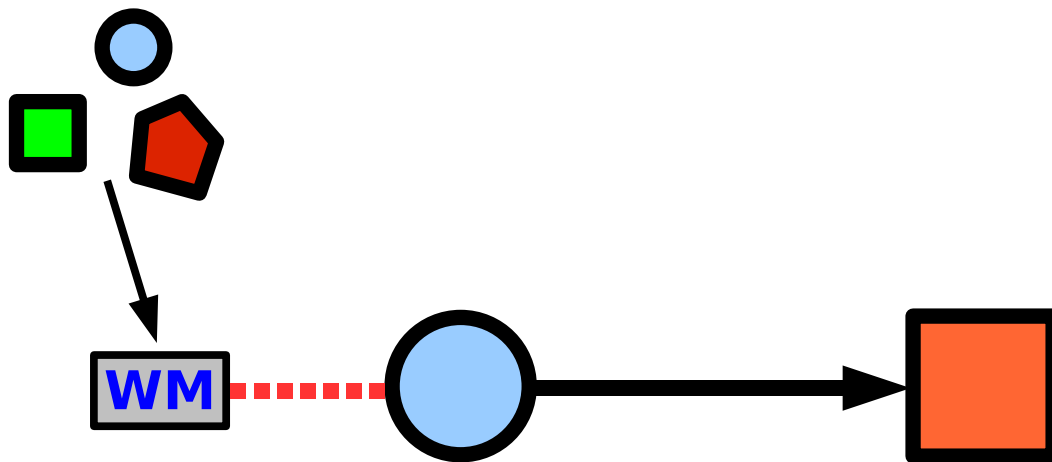
Messages

- Get WriteMessage from SendPort



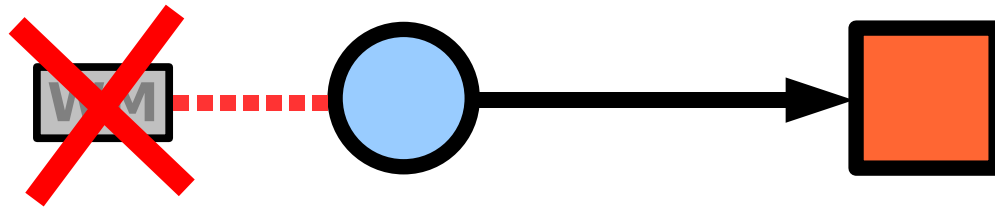
Messages

- Write data into WriteMessage



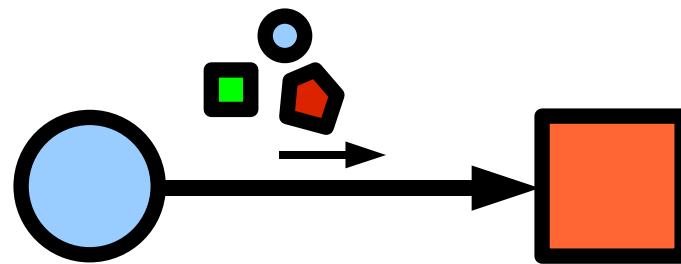
Messages

- Finish the WriteMessage



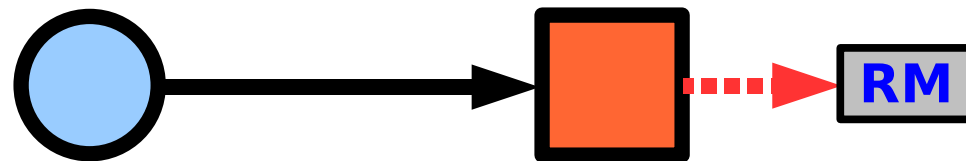
Messages

- Data is send to ReceivePort



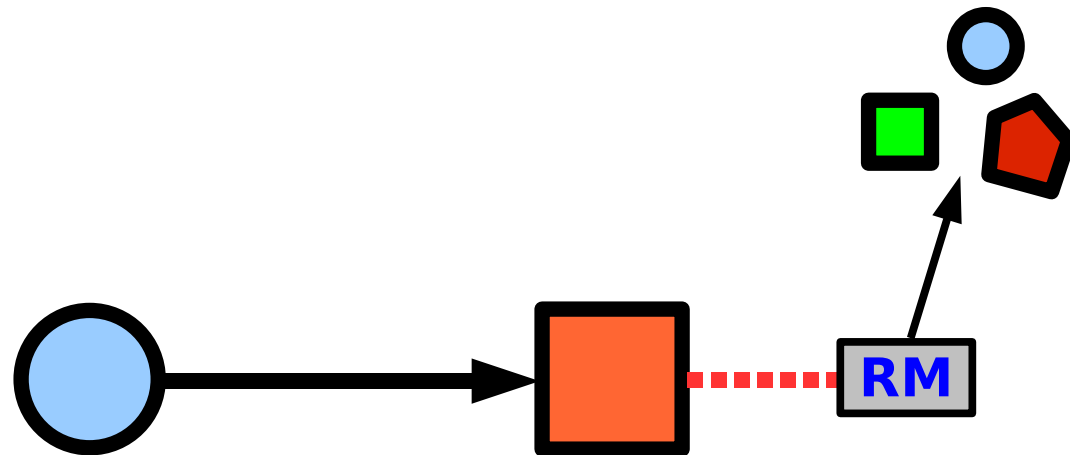
Messages

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



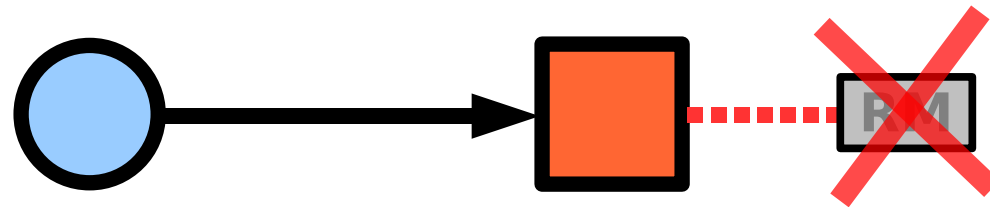
Messages

- Read data from ReadMessage



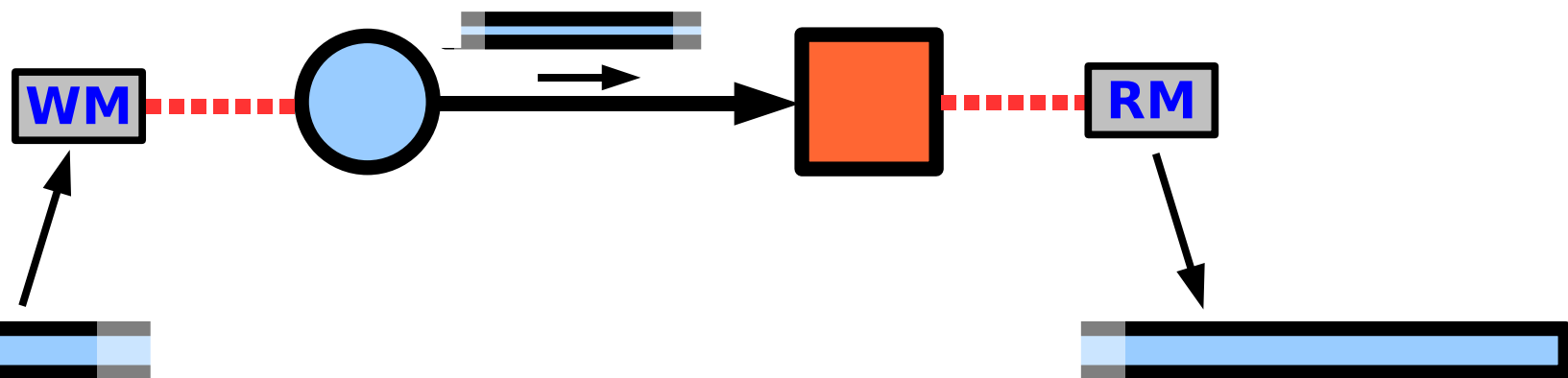
Messages

- Finish the ReadMessage



Messages or streams ?

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

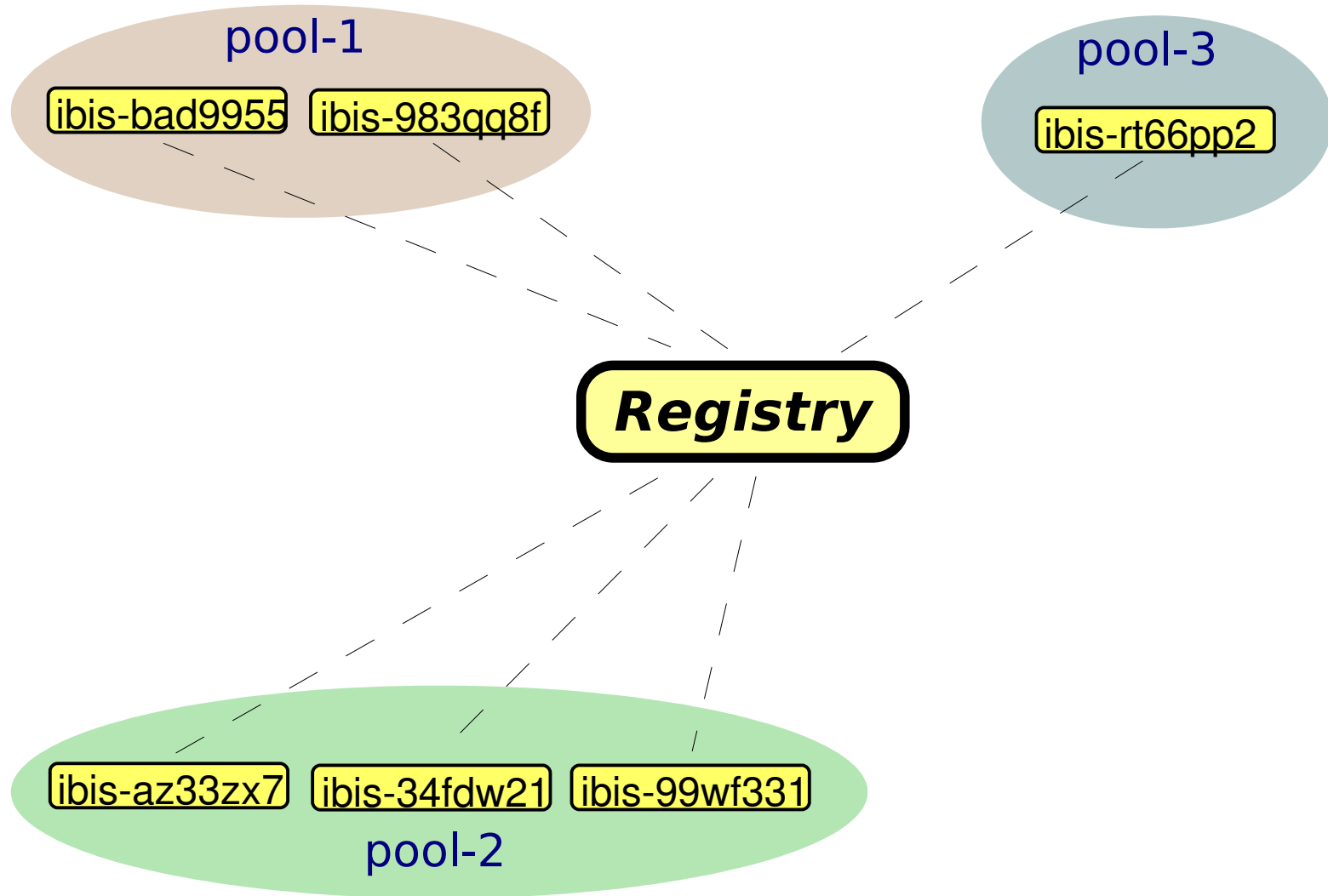


Resource Tracking

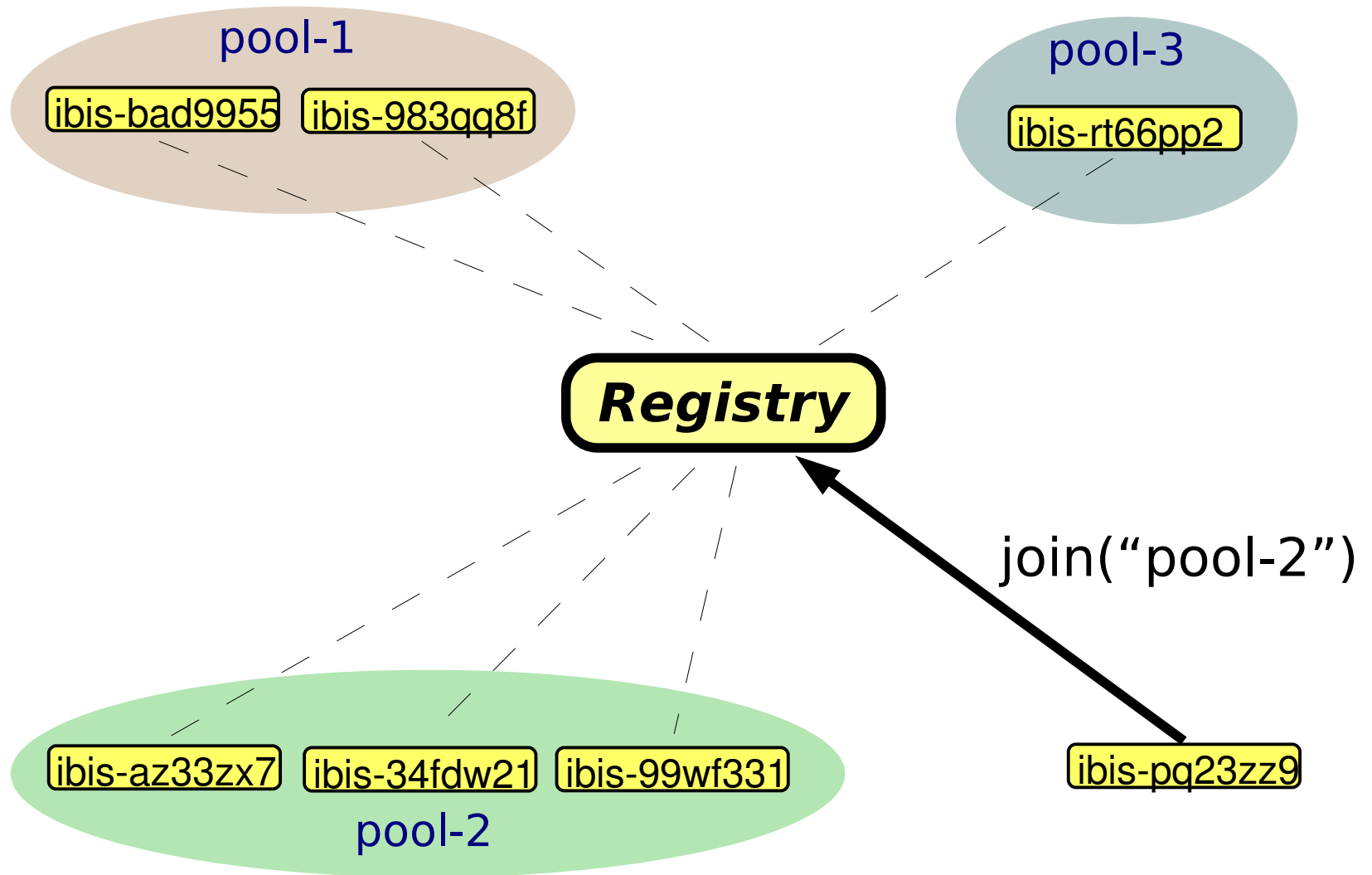
- JEL (join, elect, leave) model
 - notifies the application when a machine joins, leaves or crashes
- Example shows a centralized version
 - server that tracks application participants
 - also have broadcast tree and gossiping implementations (improve scalability)



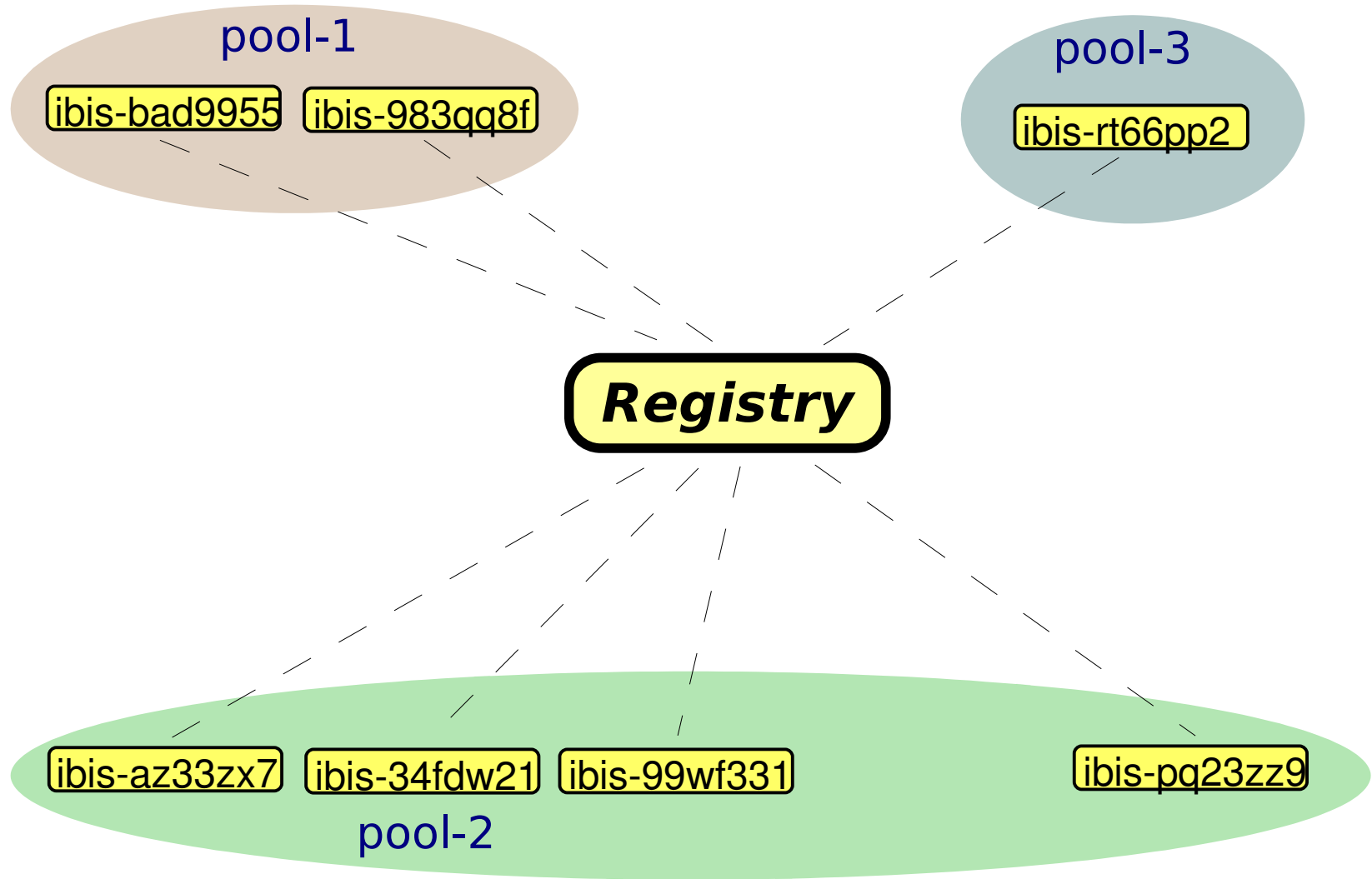
Pools & Malleability



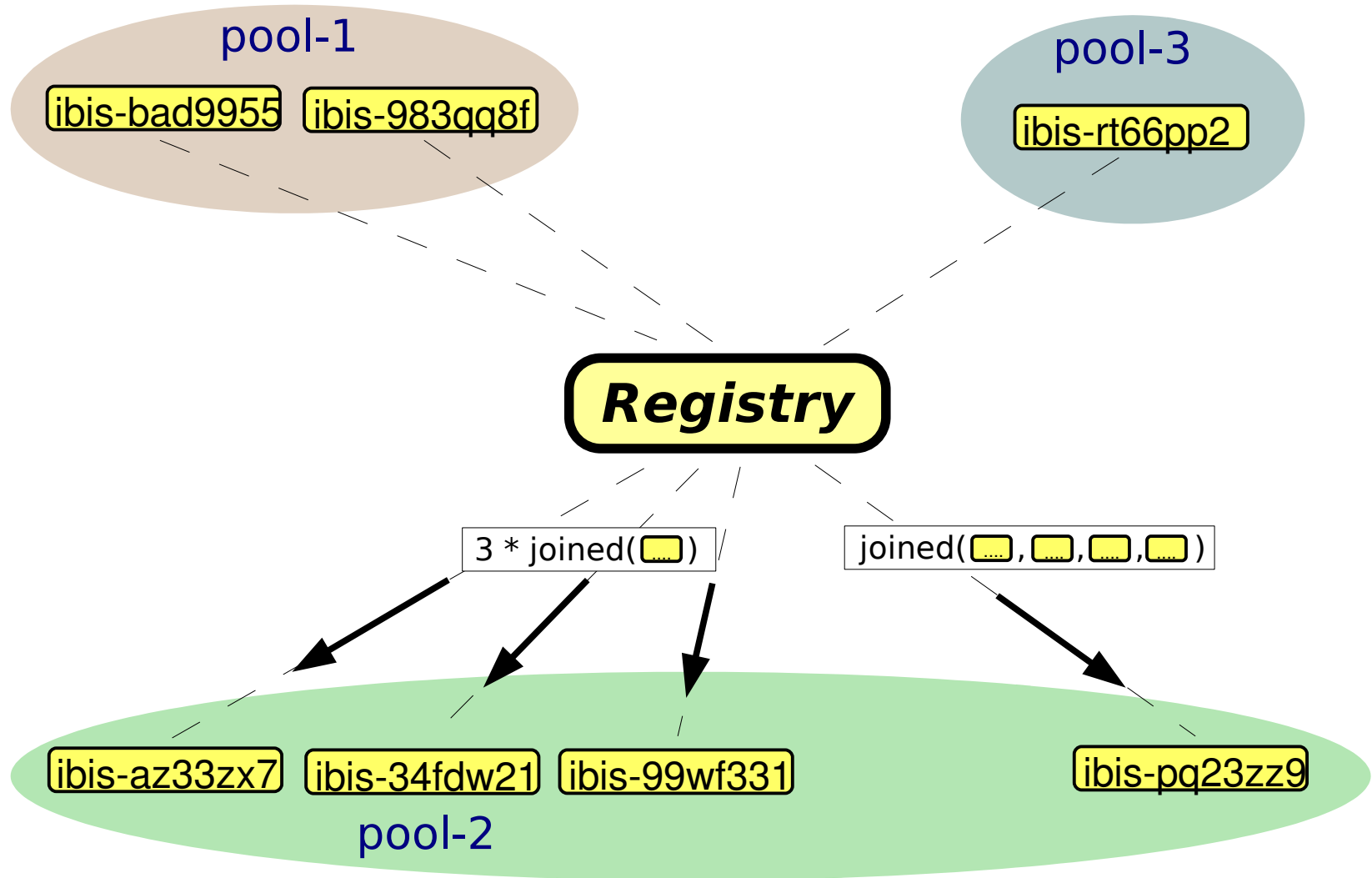
Pools & Malleability



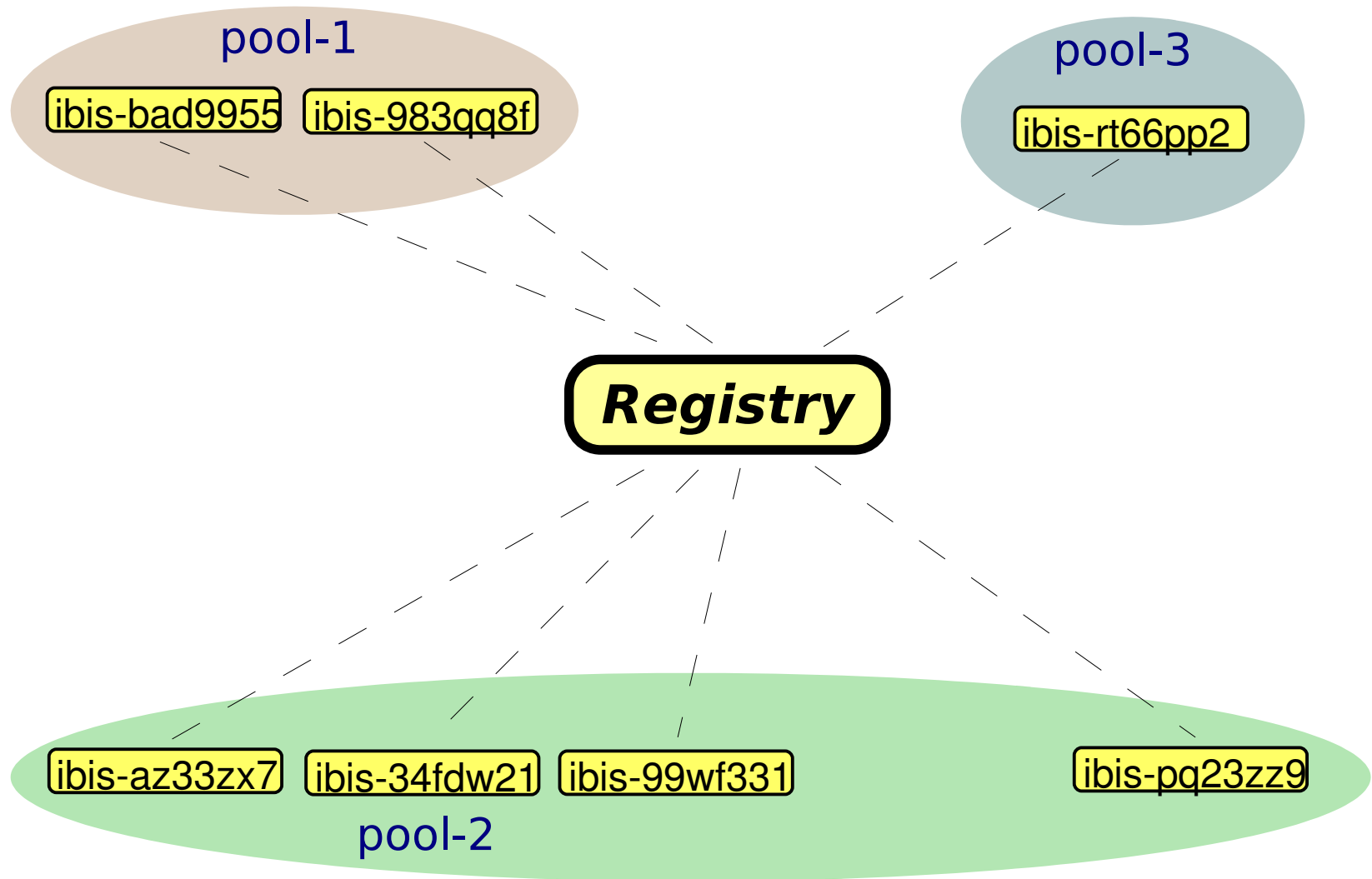
Pools & Malleability



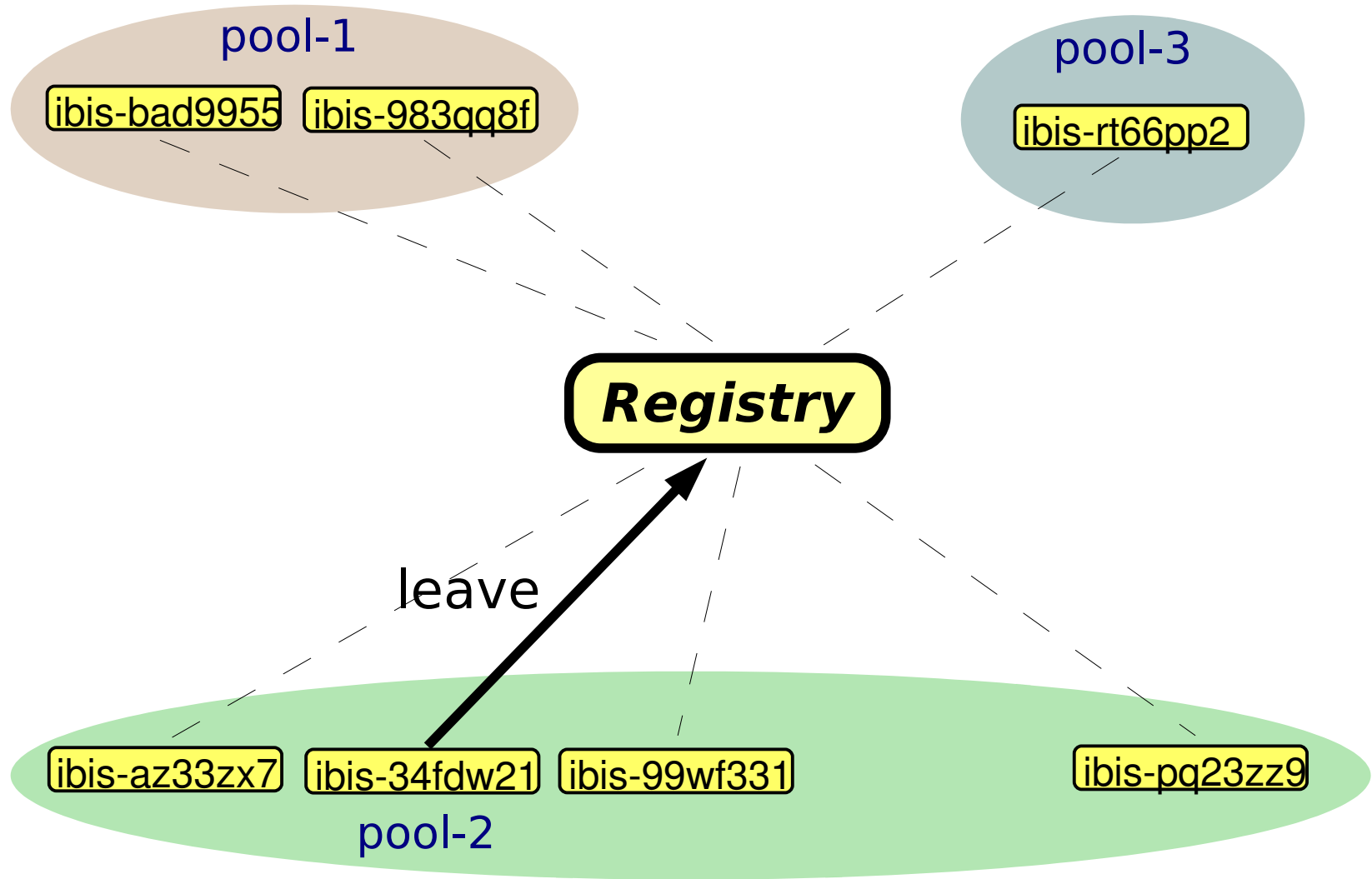
Pools & Malleability



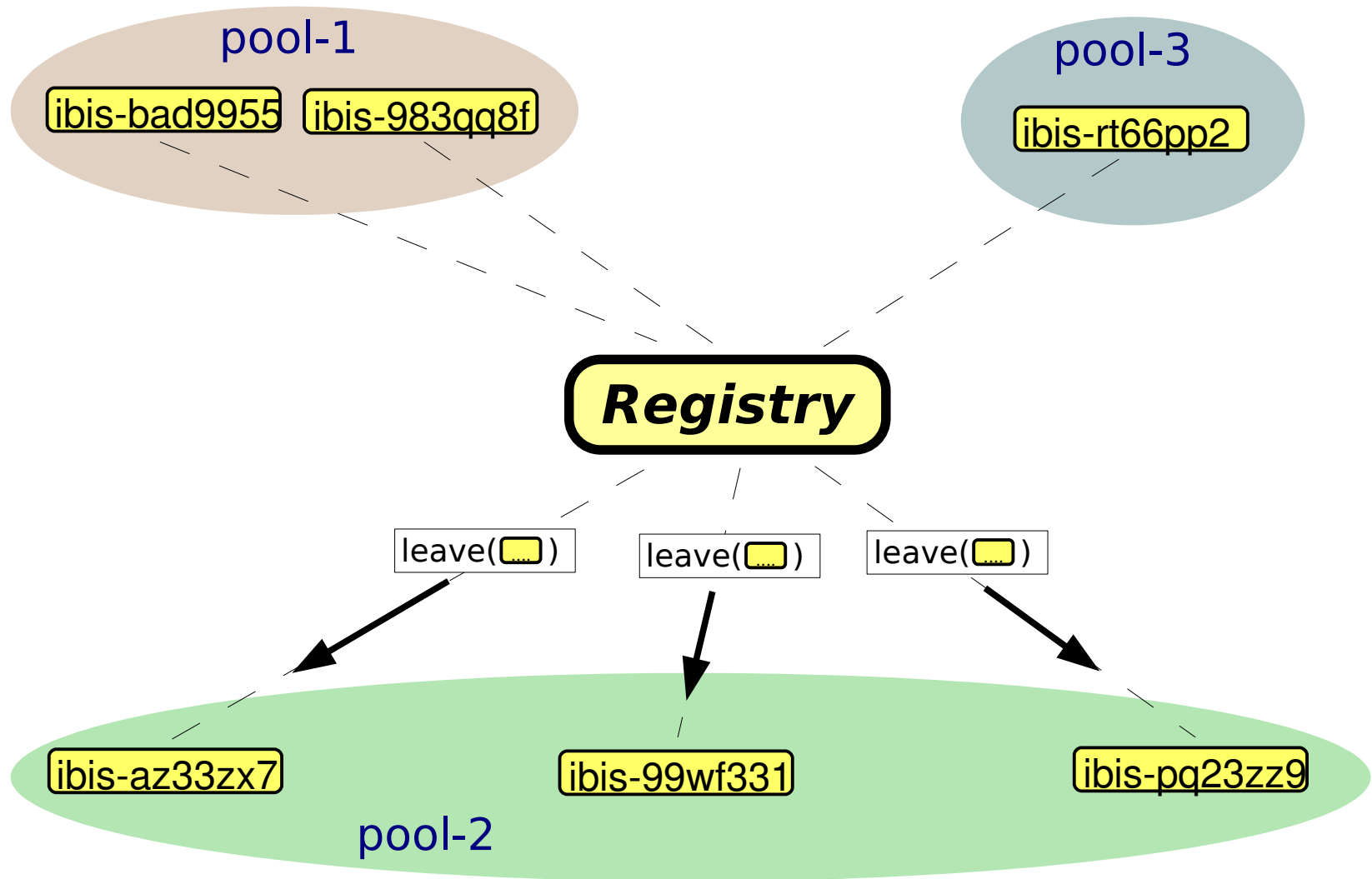
Pools & Malleability



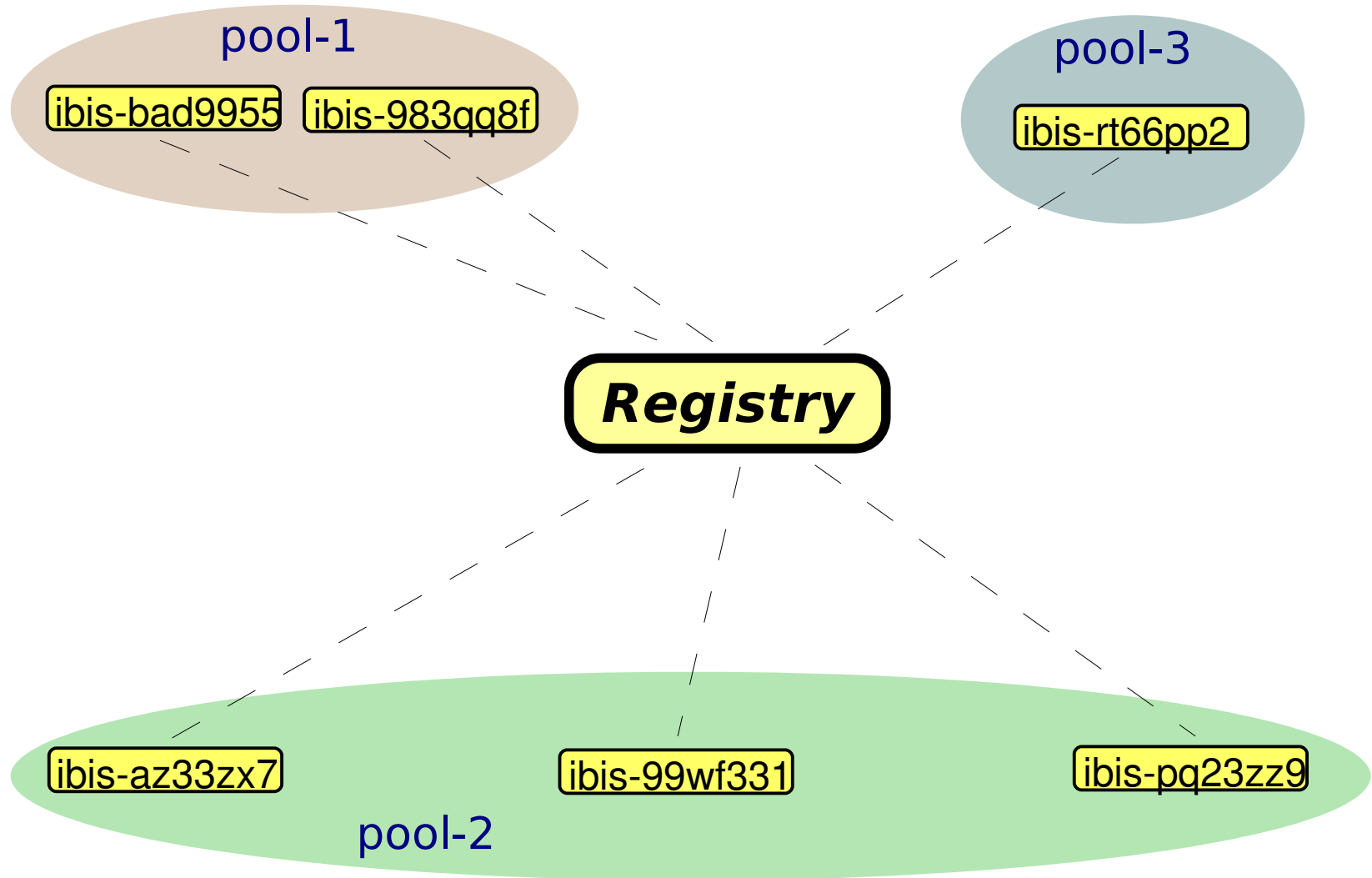
Pools & Malleability



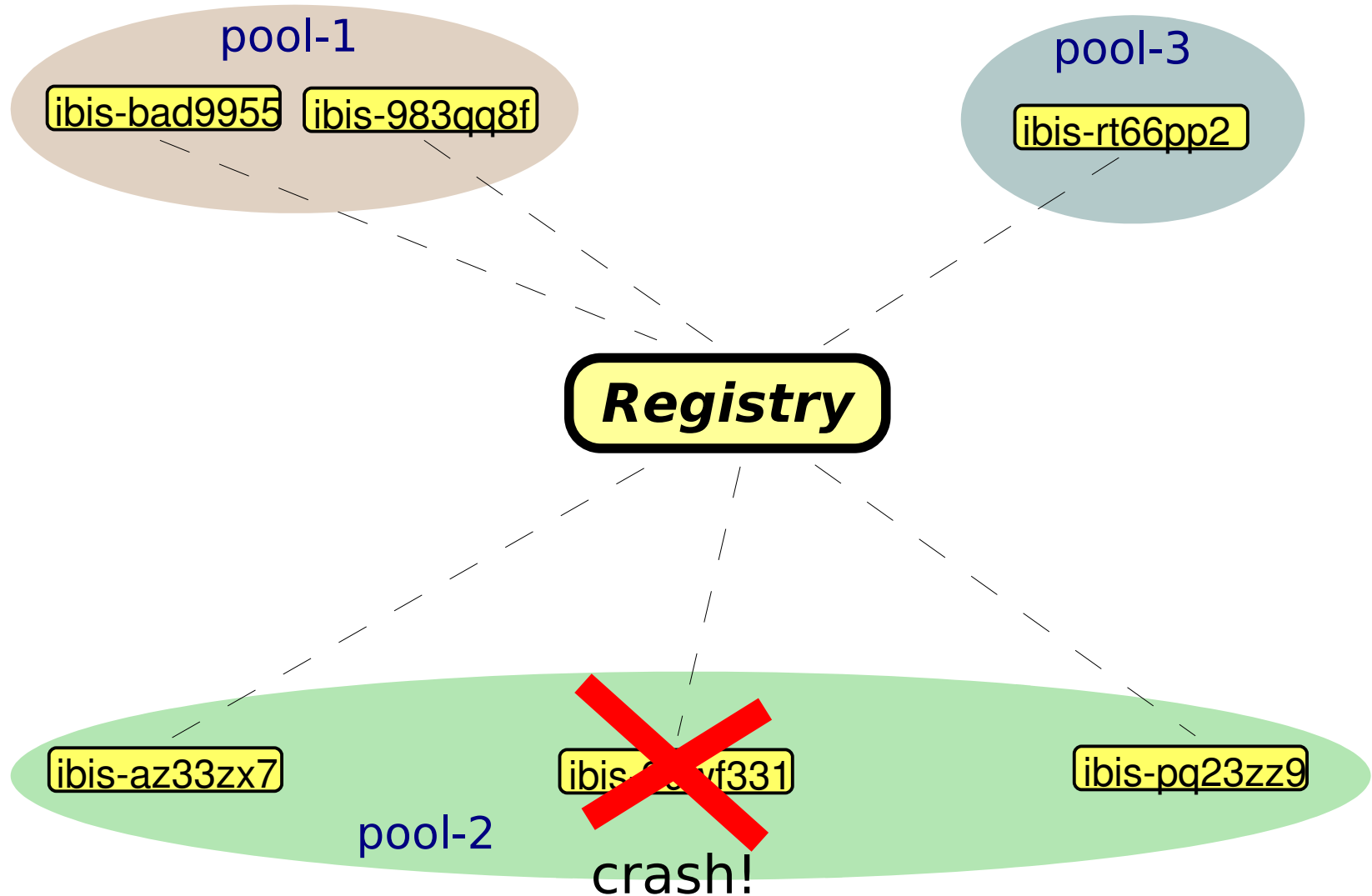
Pools & Malleability



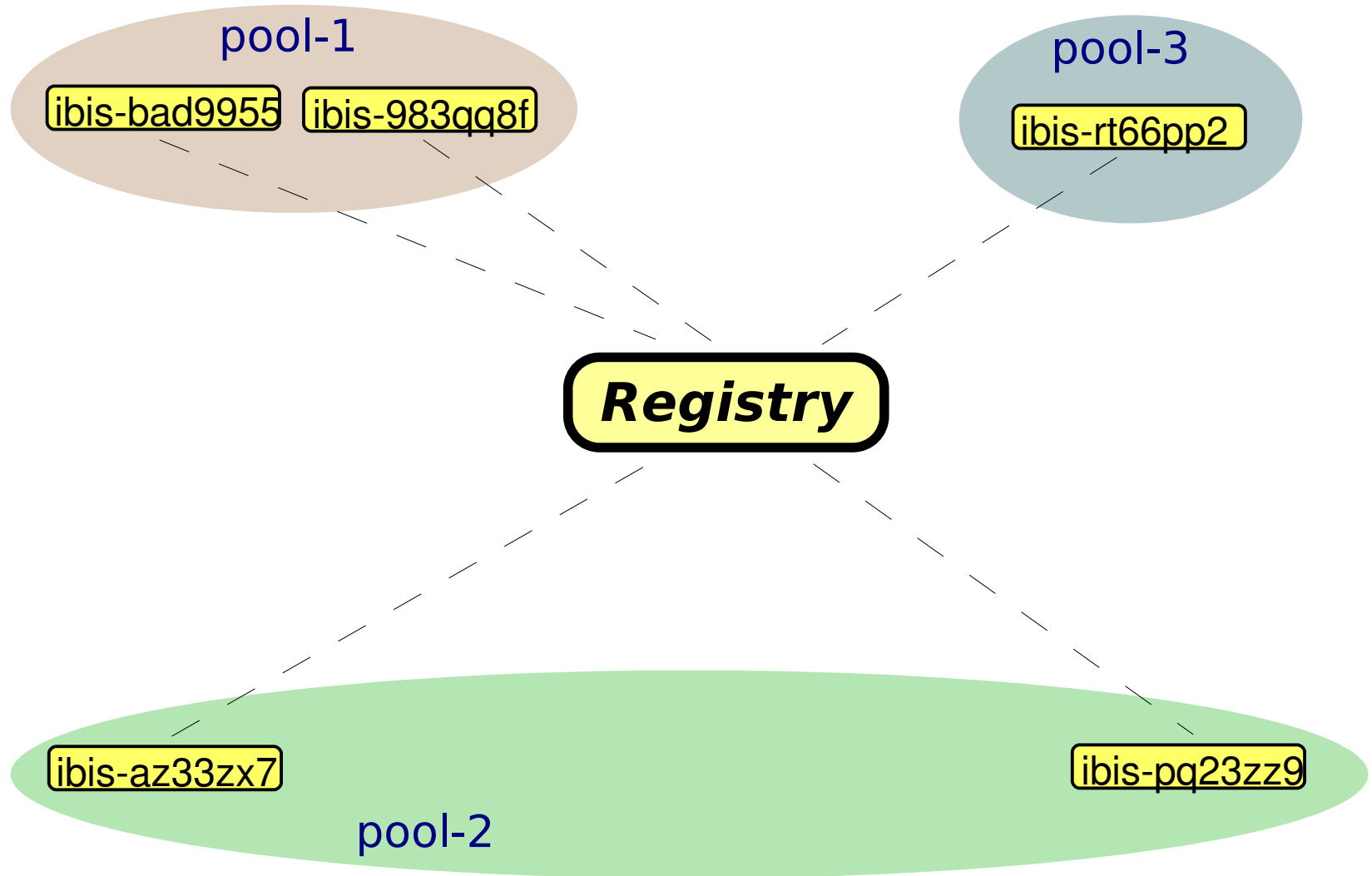
Pools & Malleability



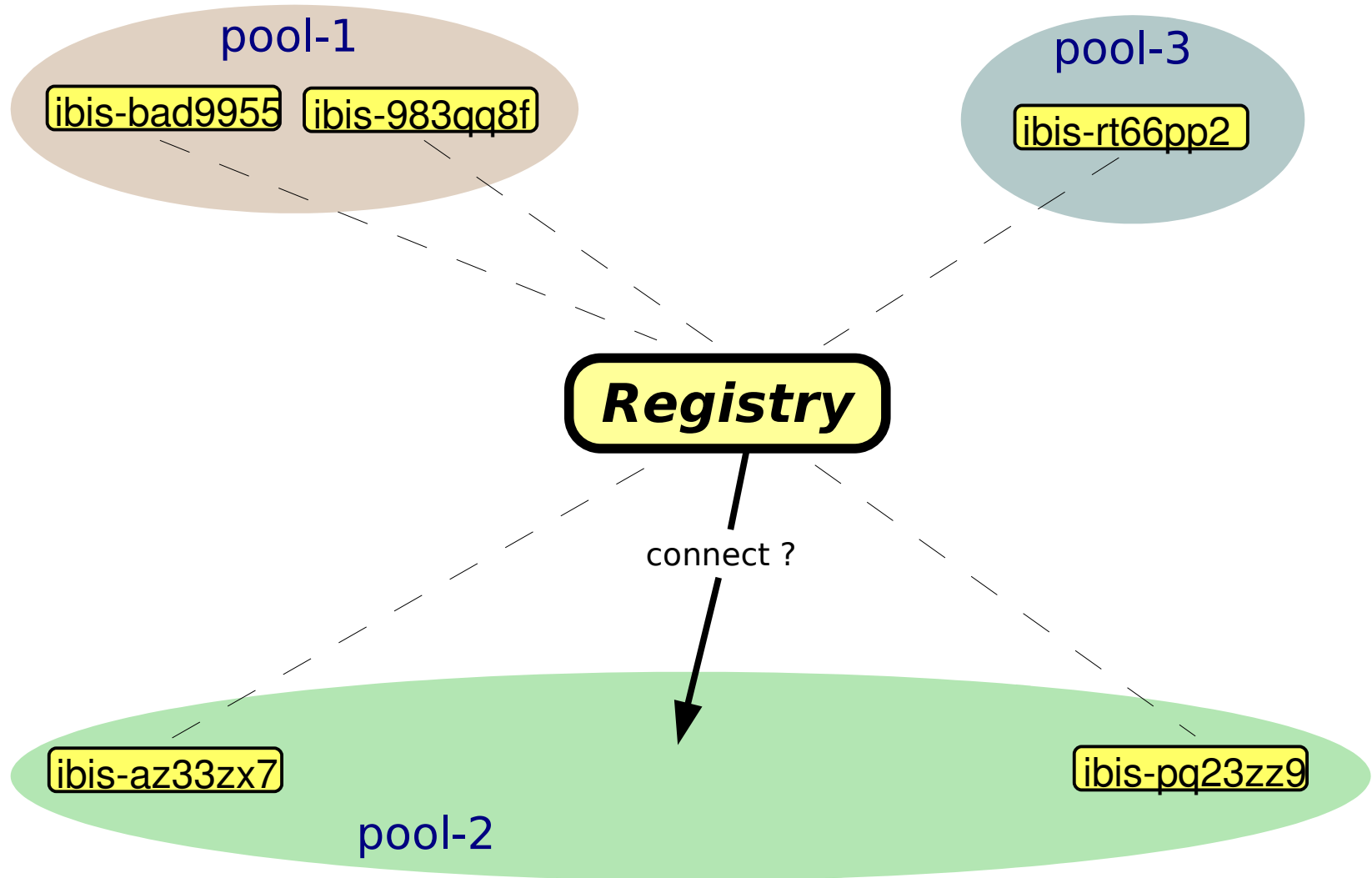
Pools & Malleability



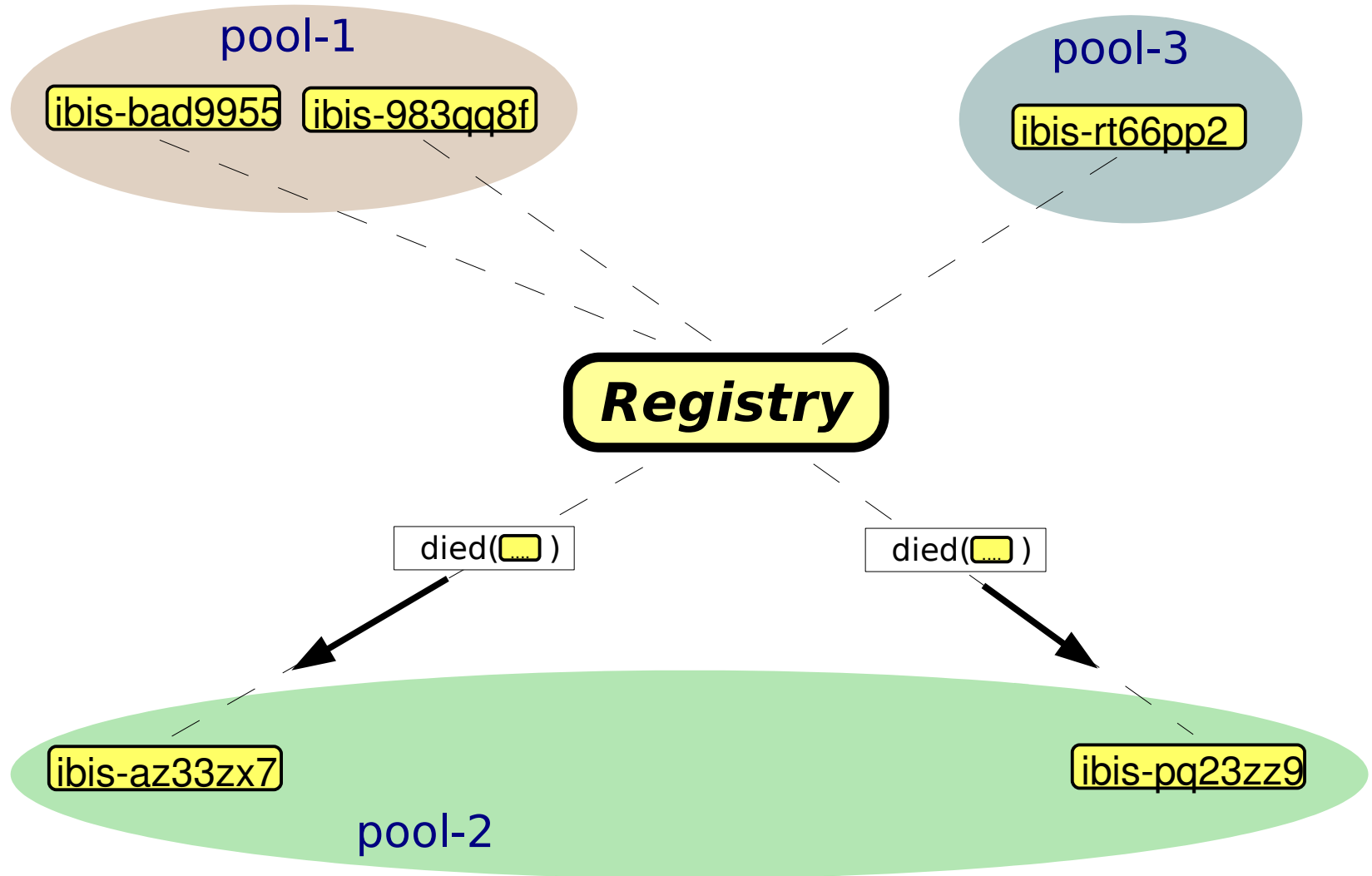
Pools & Malleability



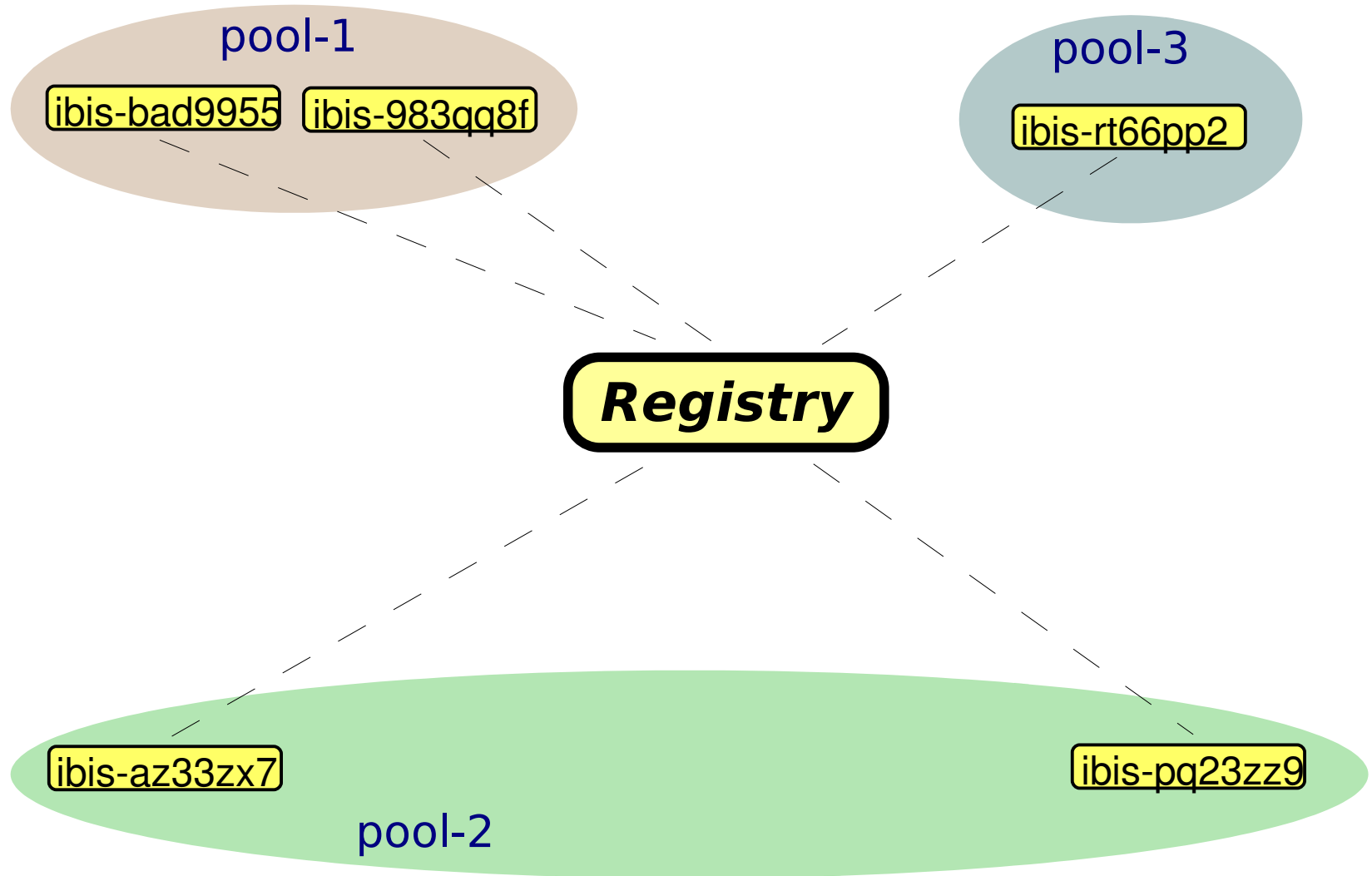
Pools & Malleability



Pools & Malleability



Pools & Malleability



Registry

- Many implementations
 - centralized, broadcast, gossiping, etc.
 - different tradeoffs in complexity, robustness and consistency
- You can select the functionality and consistency that is needed
 - reducing functionality or consistency further improves scalability



Elections

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



Summary

- IPL offers an abstract model
 - Connection oriented message passing
 - Hides network details (for portability)
- Supports fault tolerance / malleability
 - **No system-level fault tolerance!**
 - Only offers the means to implement fault tolerance in application / runtime system!
- Higher level models (Satin) can offer transparent fault tolerance



Hands on

- Later today we have a hands on session
- Chat application:
 - Good example of the IPL/JEL model
 - Machines may join/leave at any time!

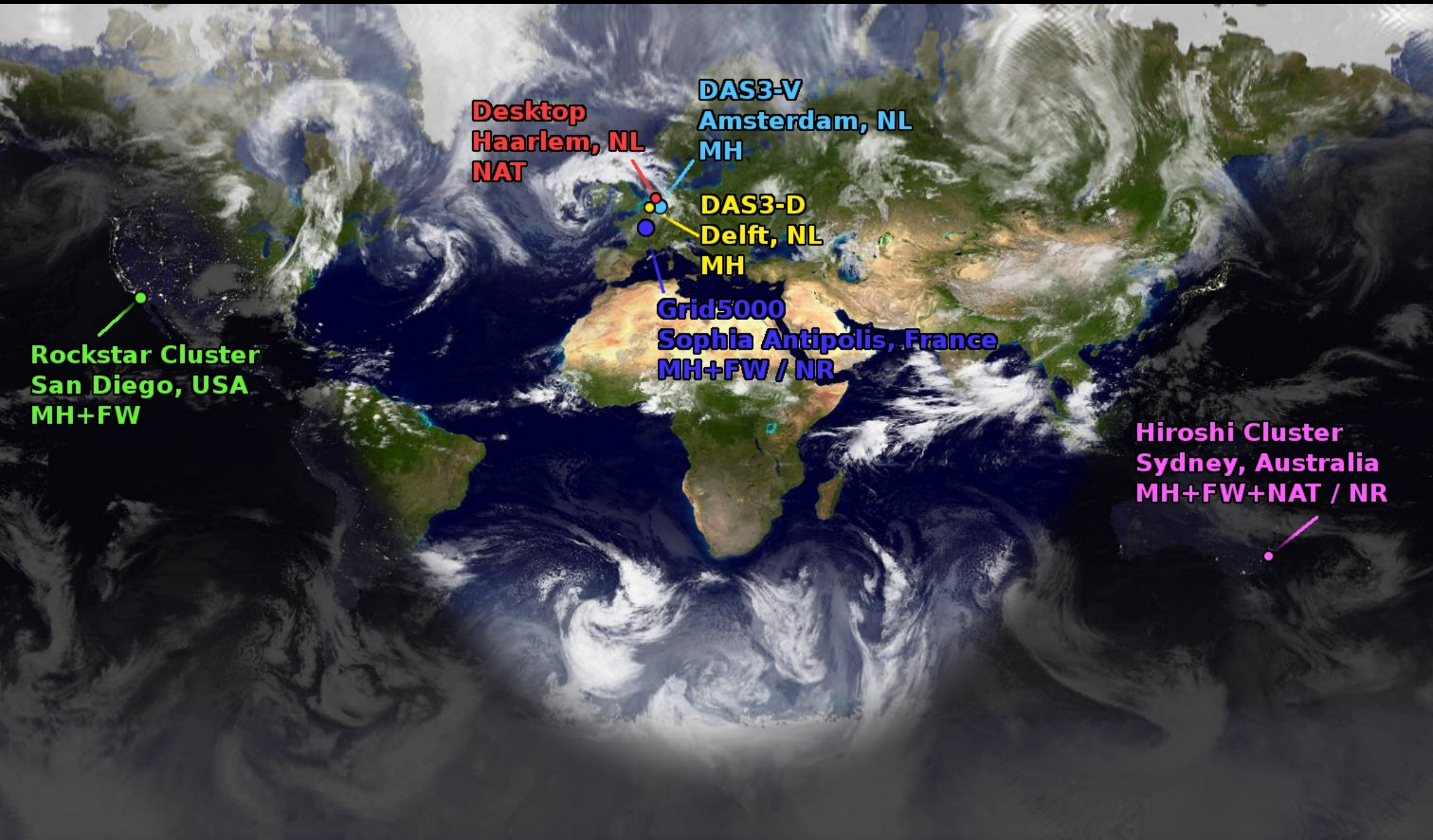




Ibis Serialization

- 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





Rockstar Cluster
San Diego, USA
MH+FW

Desktop
Haarlem, NL
NAT

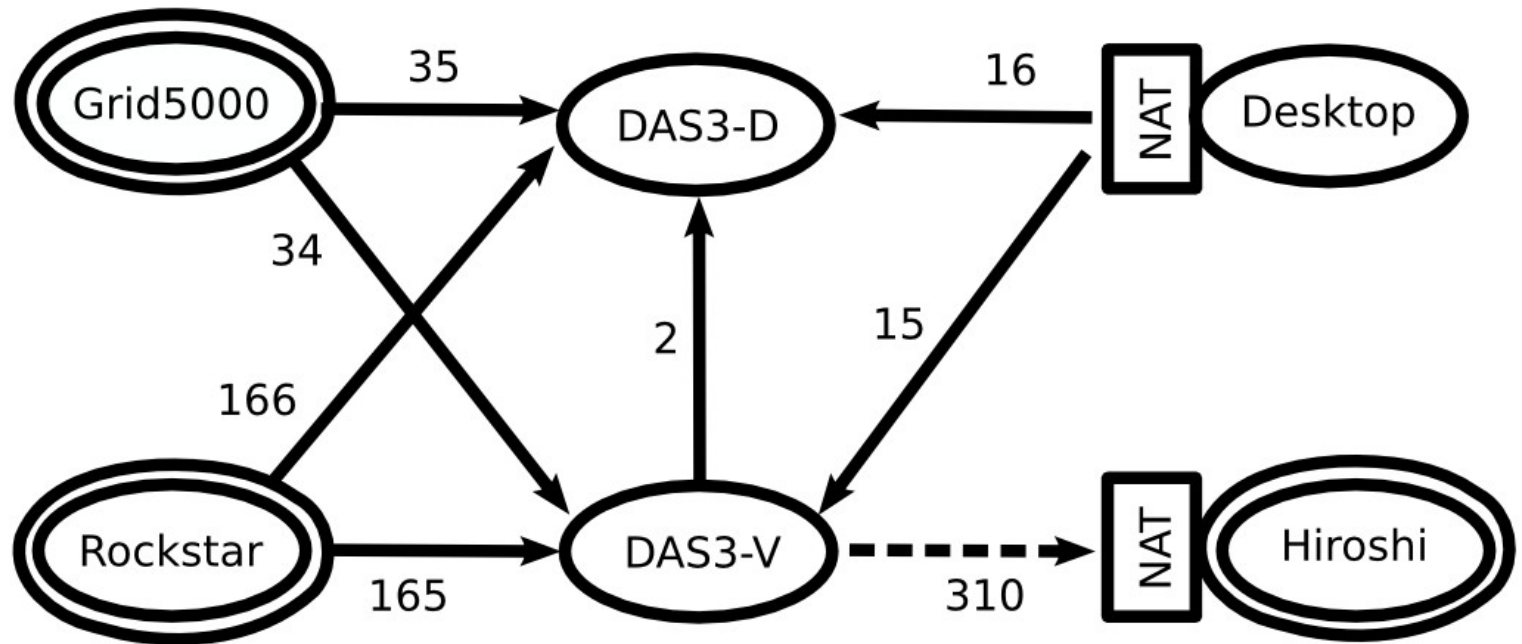
DAS3-V
Amsterdam, NL
MH

DAS3-D
Delft, NL
MH

Grid5000
Sophia Antipolis, France
MH+FW / NR

Hiroshi Cluster
Sydney, Australia
MH+FW+NAT / NR

Hub Network



Evaluation

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

<i>Target</i>	<i>Source</i>					
	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.



Evaluation

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

<i>Target</i>	<i>Source</i>					
	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).

<i>Target</i>	<i>Source</i>					
	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.

