



# ***Grid-Proof Programming Models***

***Jason Maassen***  
*jason@cs.vu.nl*



vrije Universiteit



vl·e

# ***Before lunch ...***

- We looked at how Ibis makes Grids user friendly:
  - **JavaGAT** provides an easy-to-use API for the various flavours of Grid middleware
  - **Zorilla** provides a configuration free alternative to existing Grid middleware
  - **SmartSockets** provides an easy-to-use library that solves connectivity problems



# ***Remaining problems***

- However, grids are still:
  - **Heterogeneous**: many differences in hardware, performance, OS, libraries, etc.
  - **Faulty**: machines may be claimed by others, lose contact, or simply crash
  - **Malleable**: the set of available machines varies constantly
- Therefore we need ways to make applications **Grid Proof**



# ***Grid Proof***

- **Heterogeneity:**
  - Solved by using modern languages that run in a managed execution environment:
    - Safer and more portable
    - Easy to deploy and less dependencies
      - no compilation on grid sites
      - no libraries, headers, scripts, compilers, build tools...
  - **Java**, Python, Fortress, C#, ...
    - Language level virtualization
    - Alternative: system level virtualization

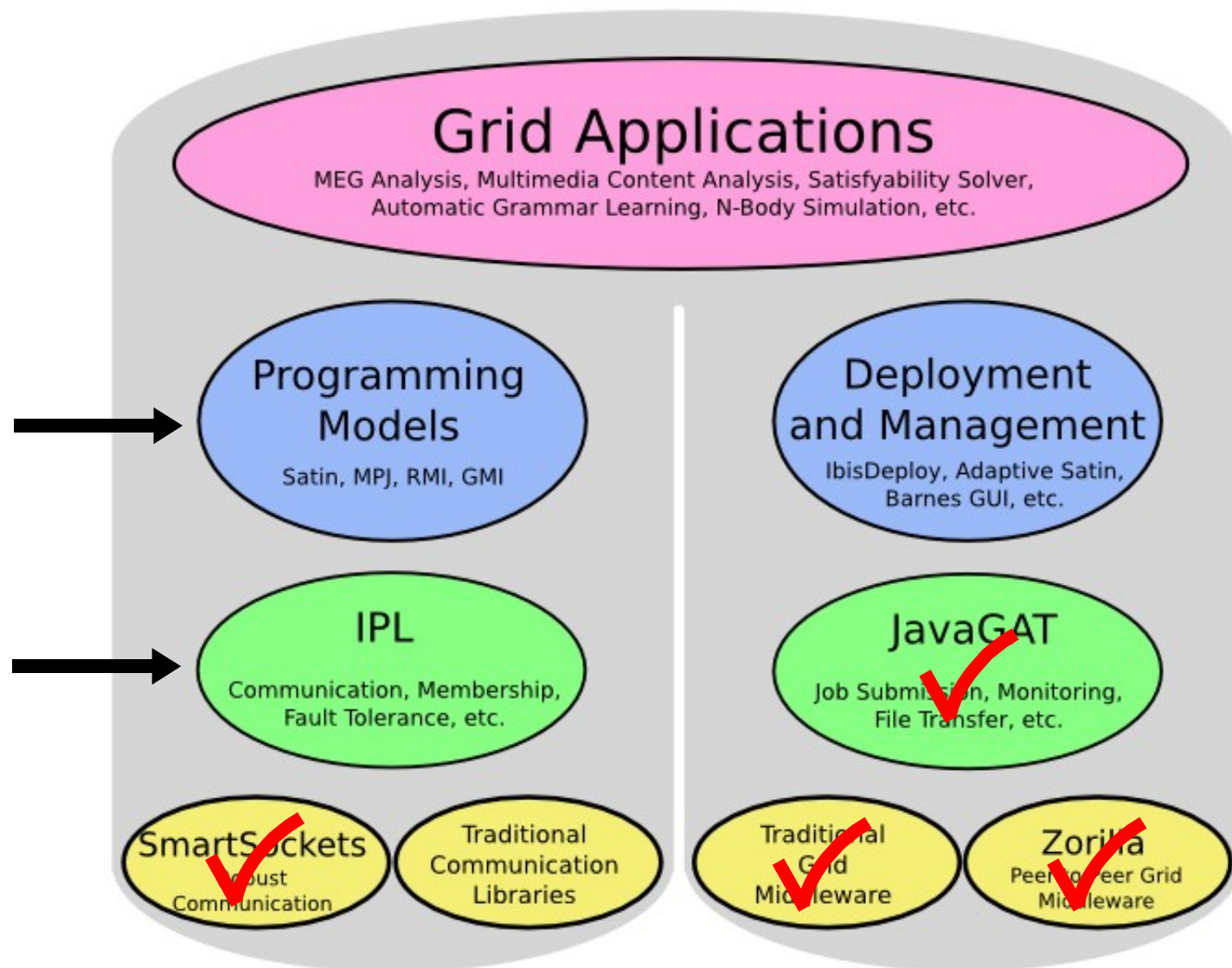


# ***Grid Proof -- cont'd***

- **Fault tolerance:**
  - We cannot prevent machines crashing
  - However, we can provide mechanisms to detect crashes ...
  - .. and use those to implement fault-tolerant programming models!
- **Malleability:**
  - Cleanly adding / removing machines
  - Handled by same mechanism



# Overview



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

- Simple API for Grid Communication
  - Flexible communication model
    - connection oriented messaging
    - abstract addressing scheme
  - Malleability/Fault Tolerance
    - notifications when machines join/leave
    - open & closed world (not just SMPD)
  - Serialization
    - send bytes, doubles, objects, etc.



# ***IPL***

- Clean & abstract API
- Designed with Grids in mind
  - Hides network specific details
    - hostnames, IP addresses, MPI ranks, etc
    - Uses **IbisIdentifier** instead
  - Results in more portable applications
- Implemented on top of TCP, MPI, MX...



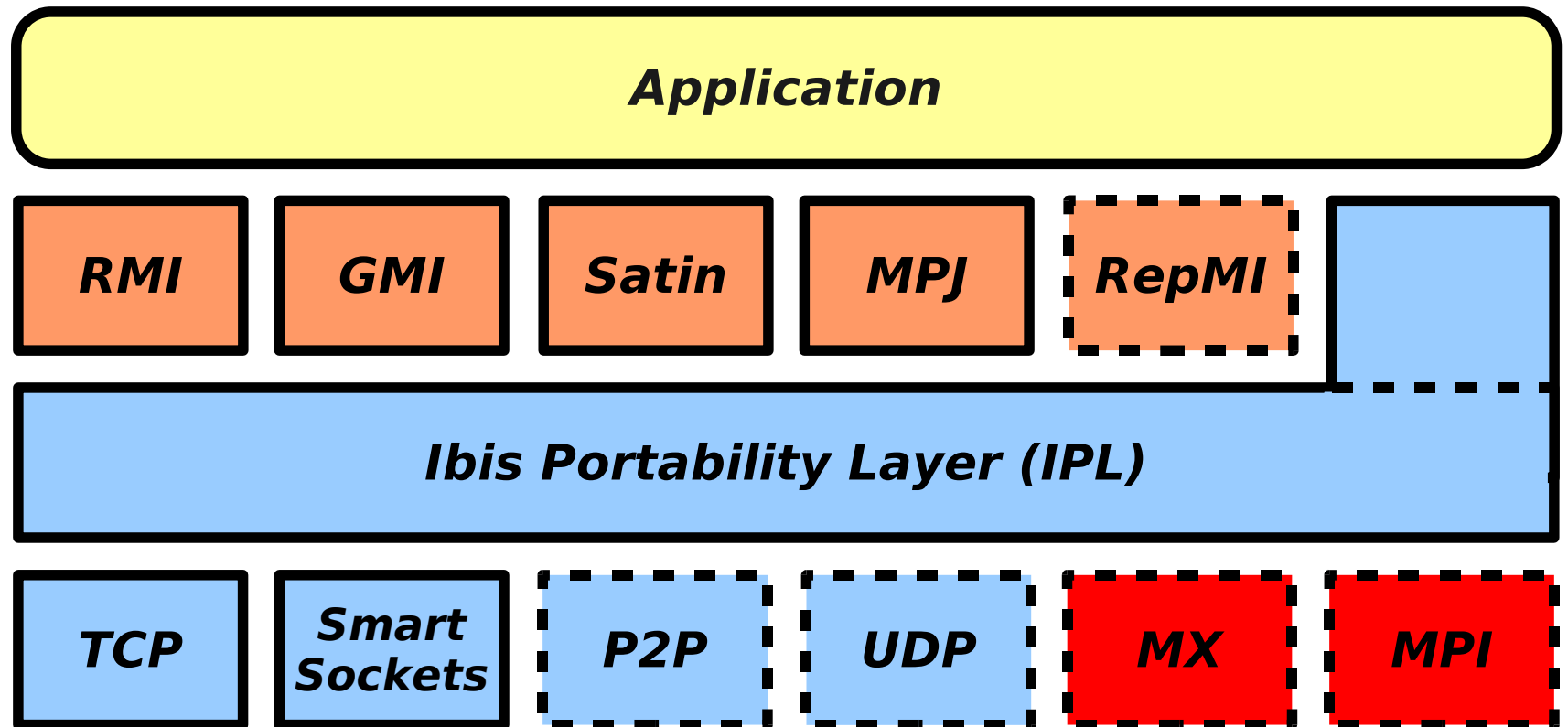


# ***IbisIdentifiers***

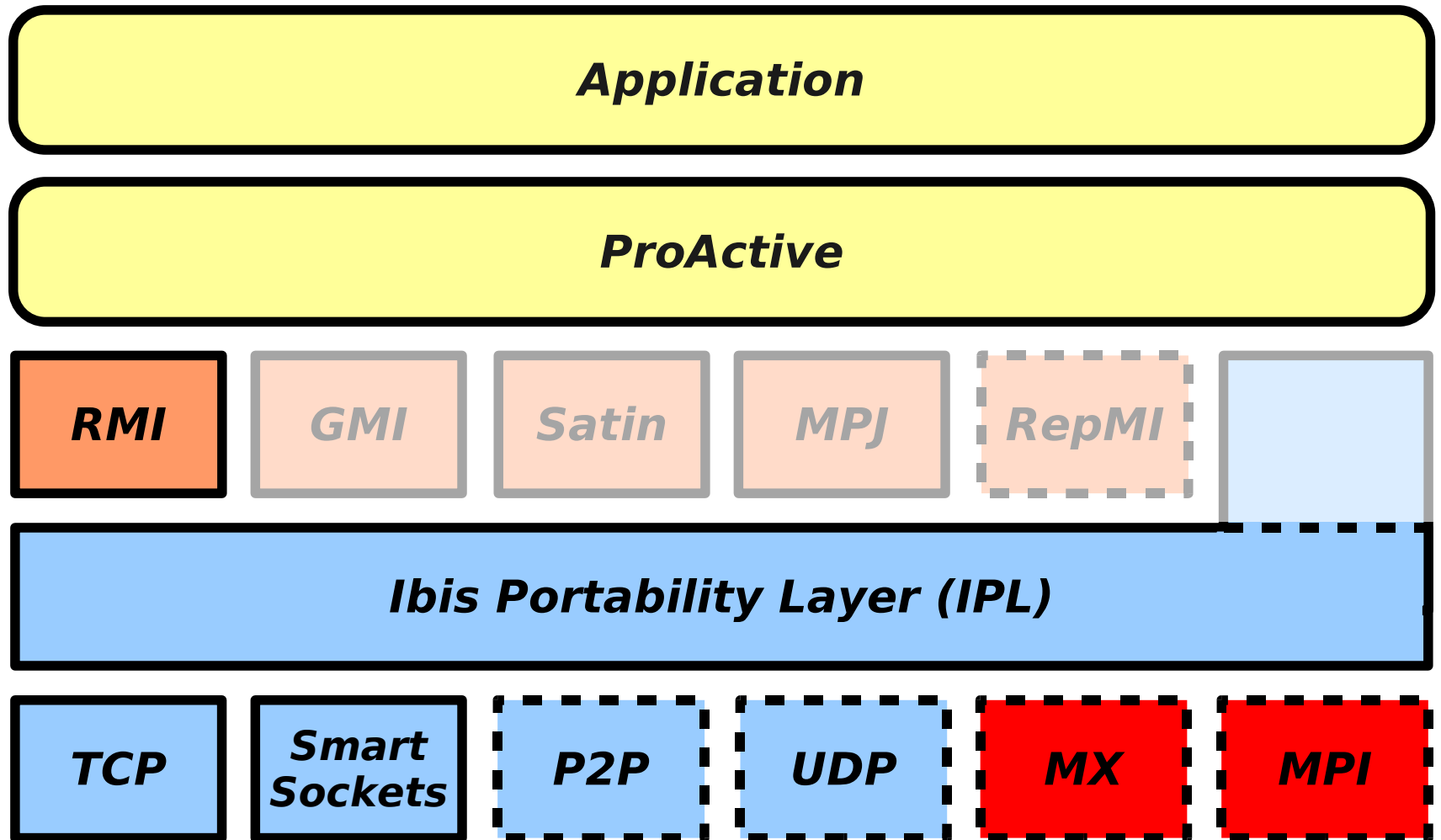
- IbisIdentifier:
  - Abstract 'machine address' object
  - Uniquely identifies an IPL instance
- Why don't we use ranks ?
  - Complicates malleability
    - what happens to the ranks when machines leave or crash ?
  - Ranks can be implemented using IbisIdentifiers!



# ***IPL and Friends***



# Example



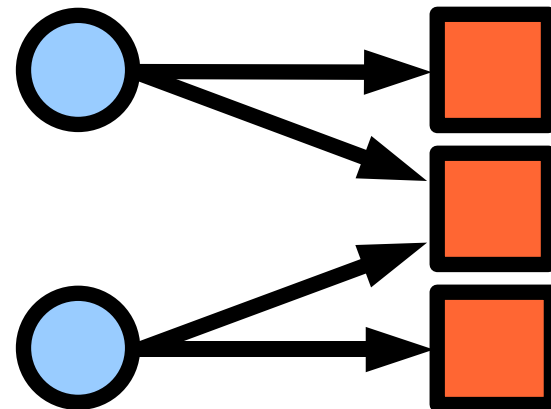
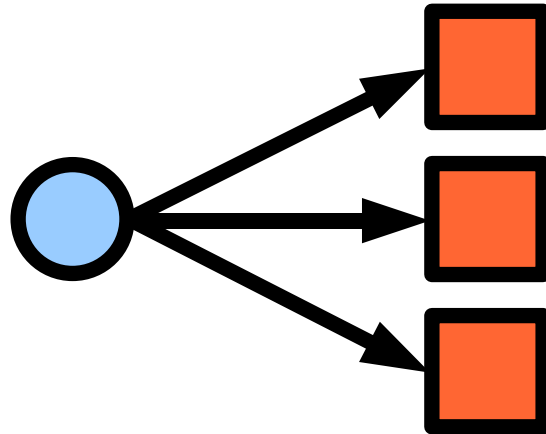
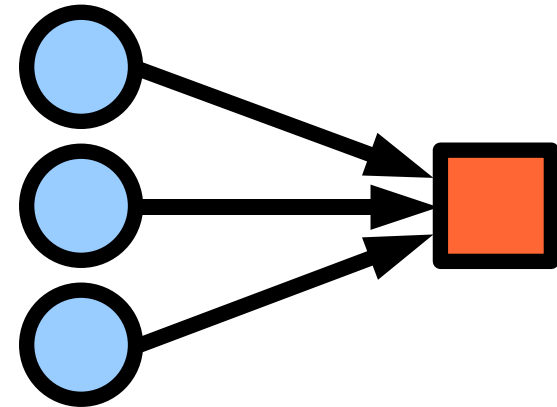
# ***Communication***

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



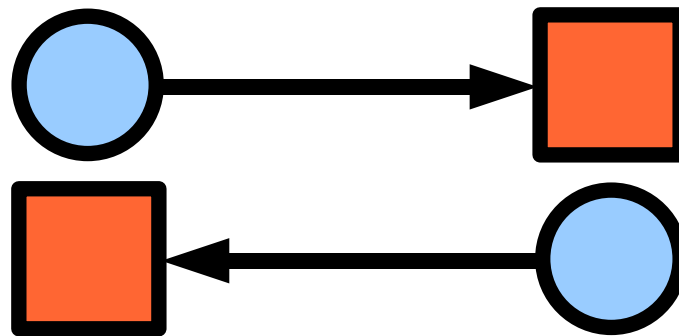
# ***Send & receive ports***

- Can be connected in arbitrary ways



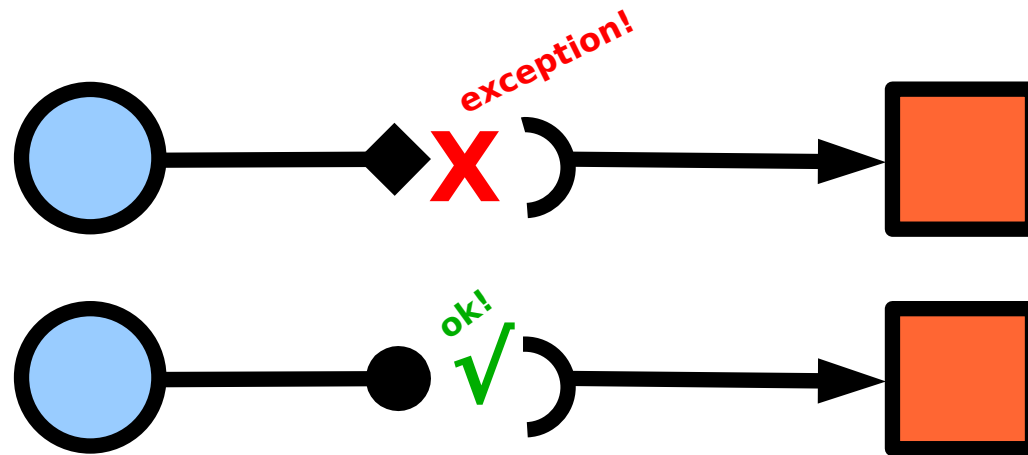
# ***Send & receive ports***

- 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***

- 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!





# ***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



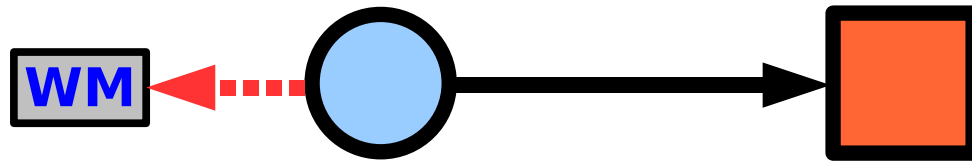
# ***Messages***

- 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



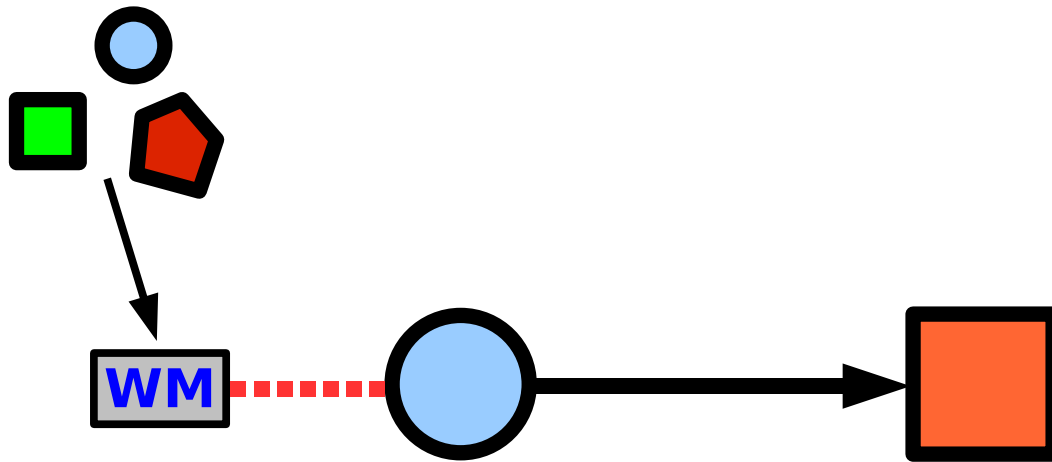
# ***Messages***

- Get WriteMessage from SendPort



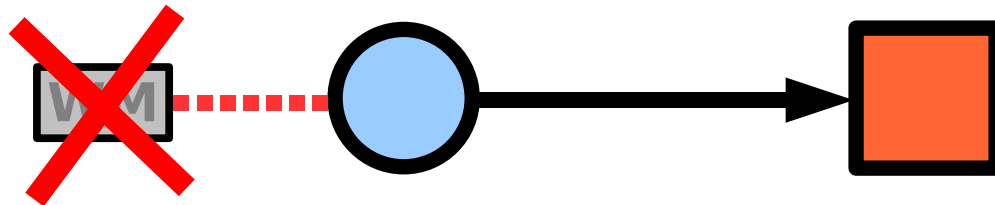
# ***Messages***

- Write data into WriteMessage



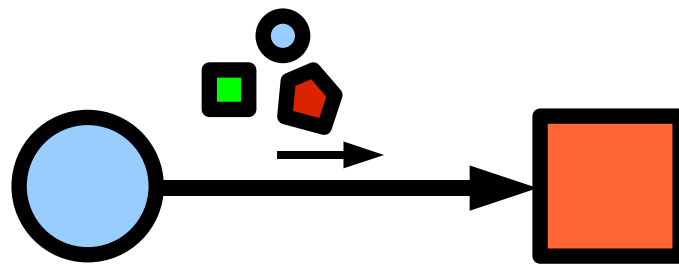
# ***Messages***

- Finish the WriteMessage



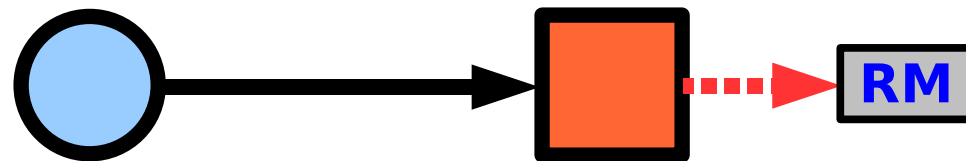
# ***Messages***

- Data is send to ReceivePort



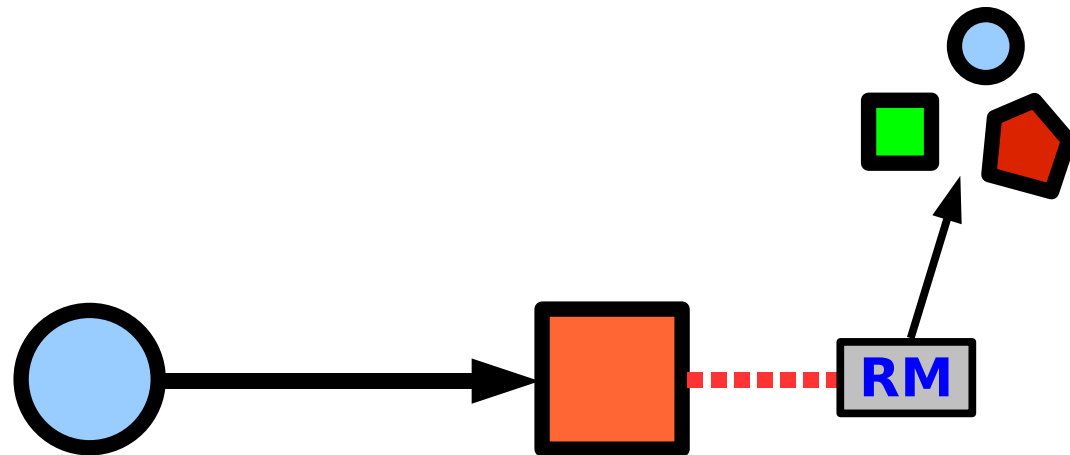
# ***Messages***

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



# ***Messages***

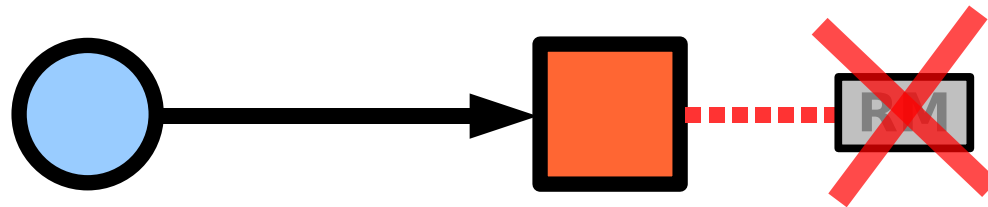
- Read data from ReadMessage





# ***Messages***

- Finish the ReadMessage



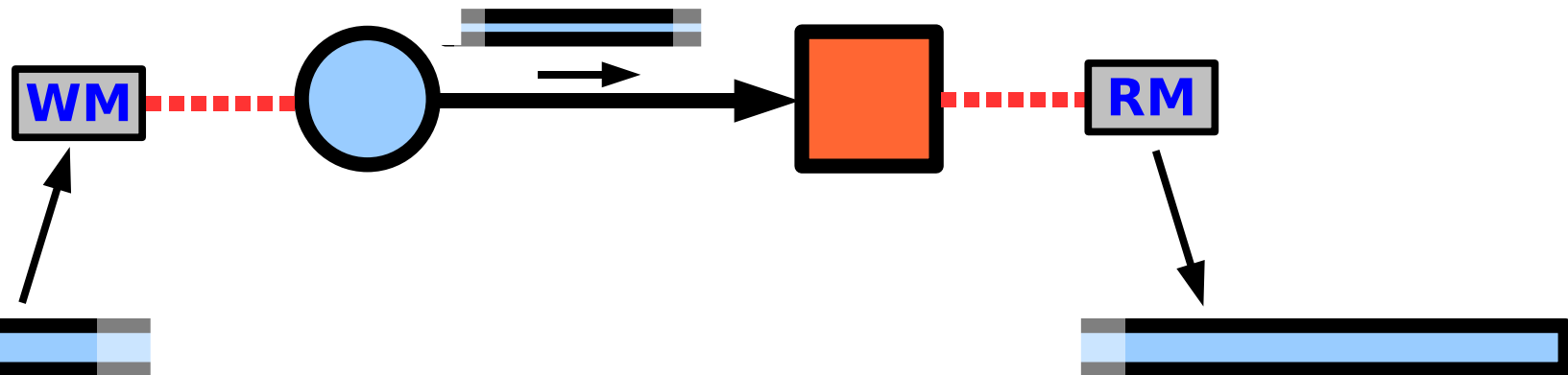
# ***Messages***

- 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!



# ***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

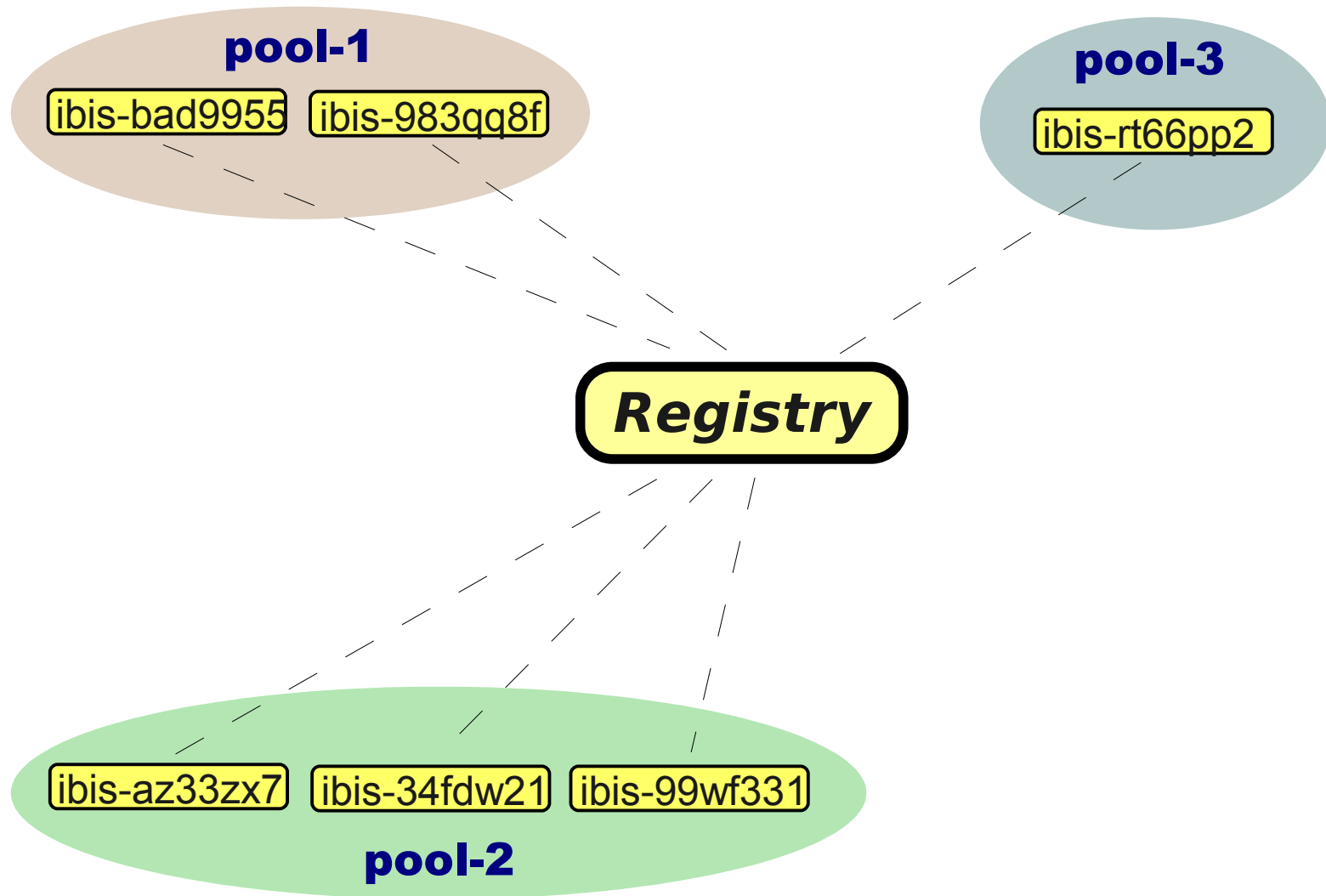


# ***Fault-Tolerance and Malleability***

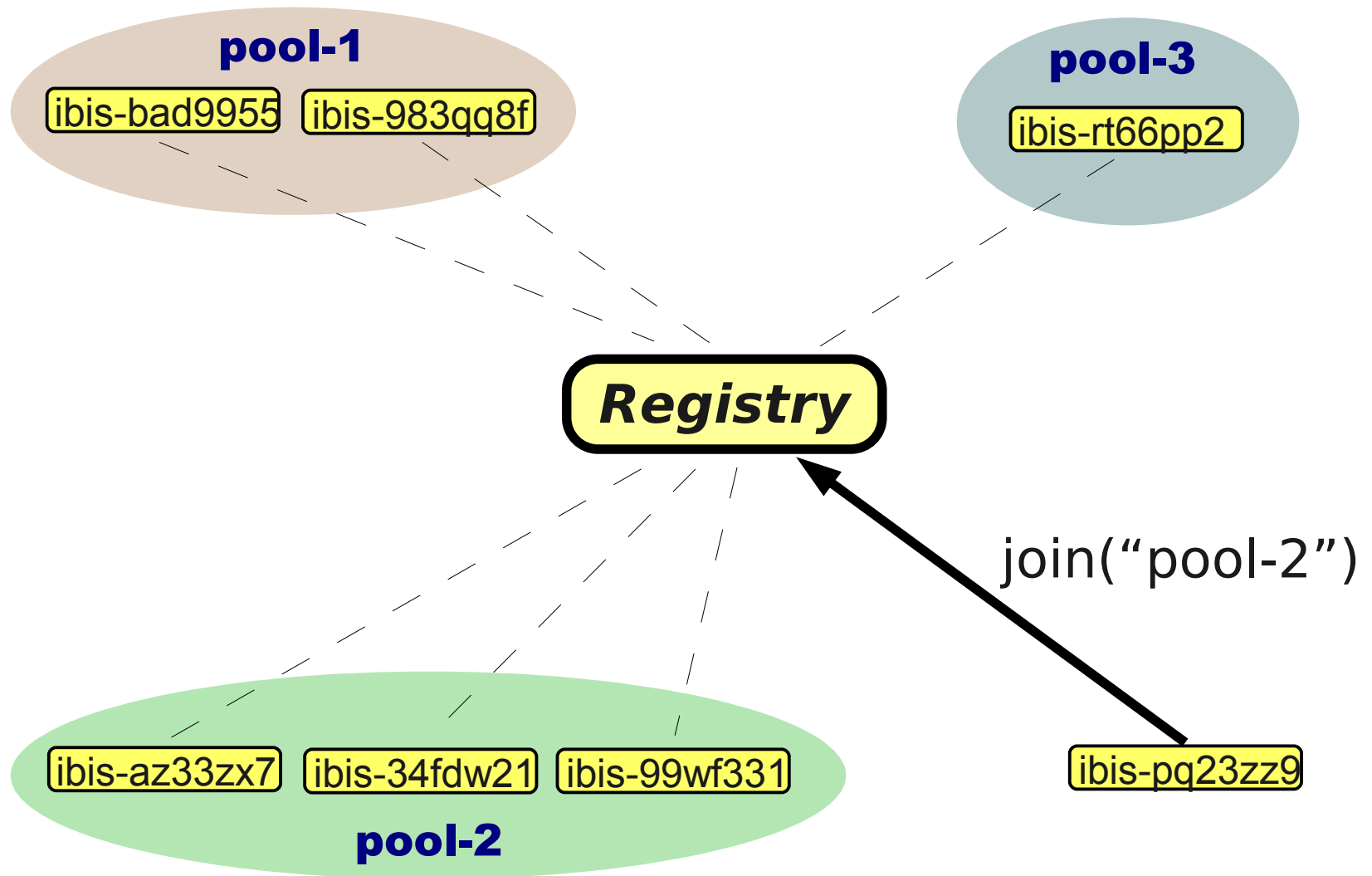
- IPL can provide callback mechanism
  - notifies the application when a machine joins, leaves or crashes
- Implemented using a **registry**
  - server that manages who participates in the application run
- Example shows a centralized version
  - also have broadcast tree and gossiping implementations (improve scalability)



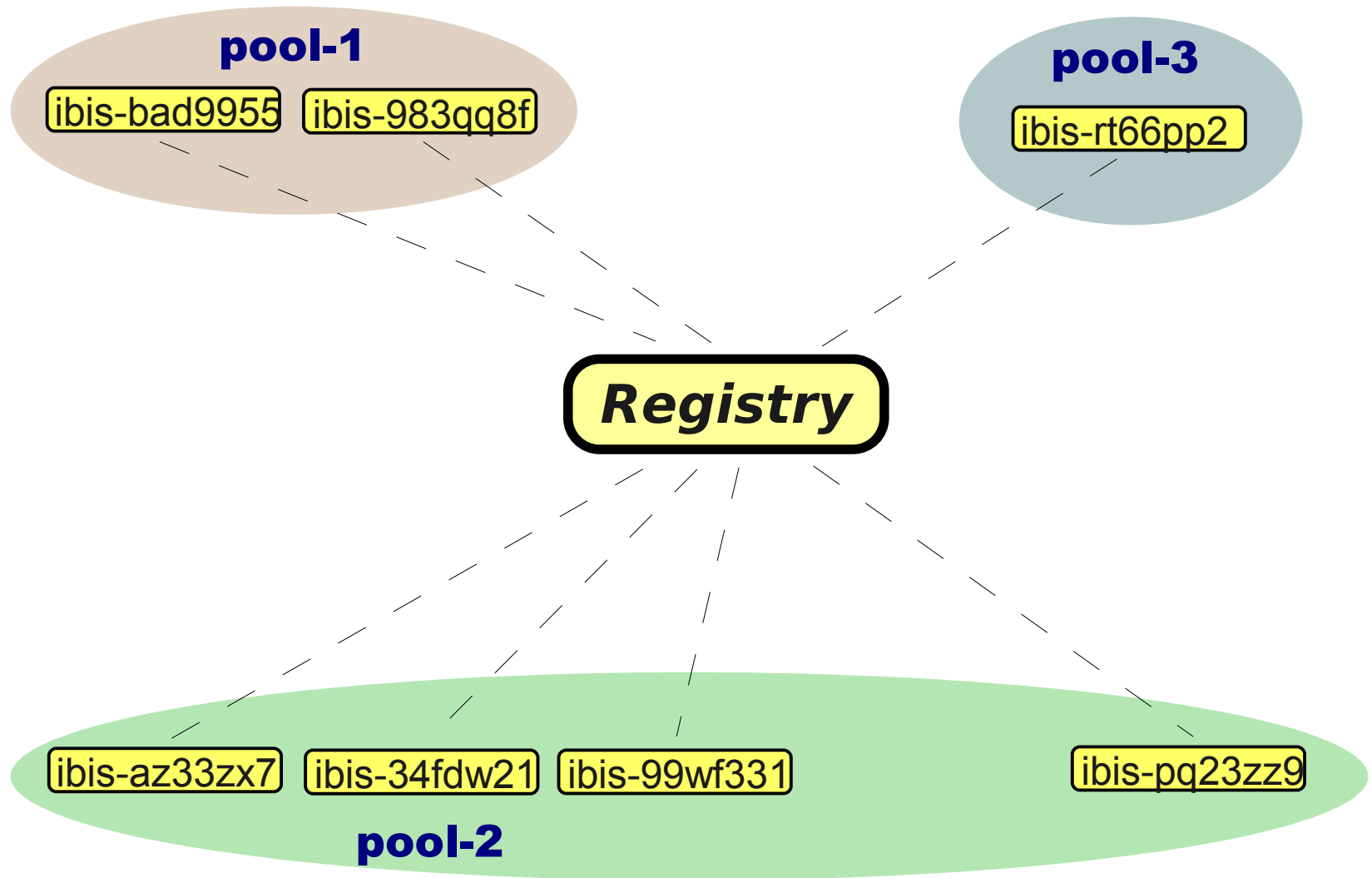
# ***Pools & Malleability***



# Pools & Malleability

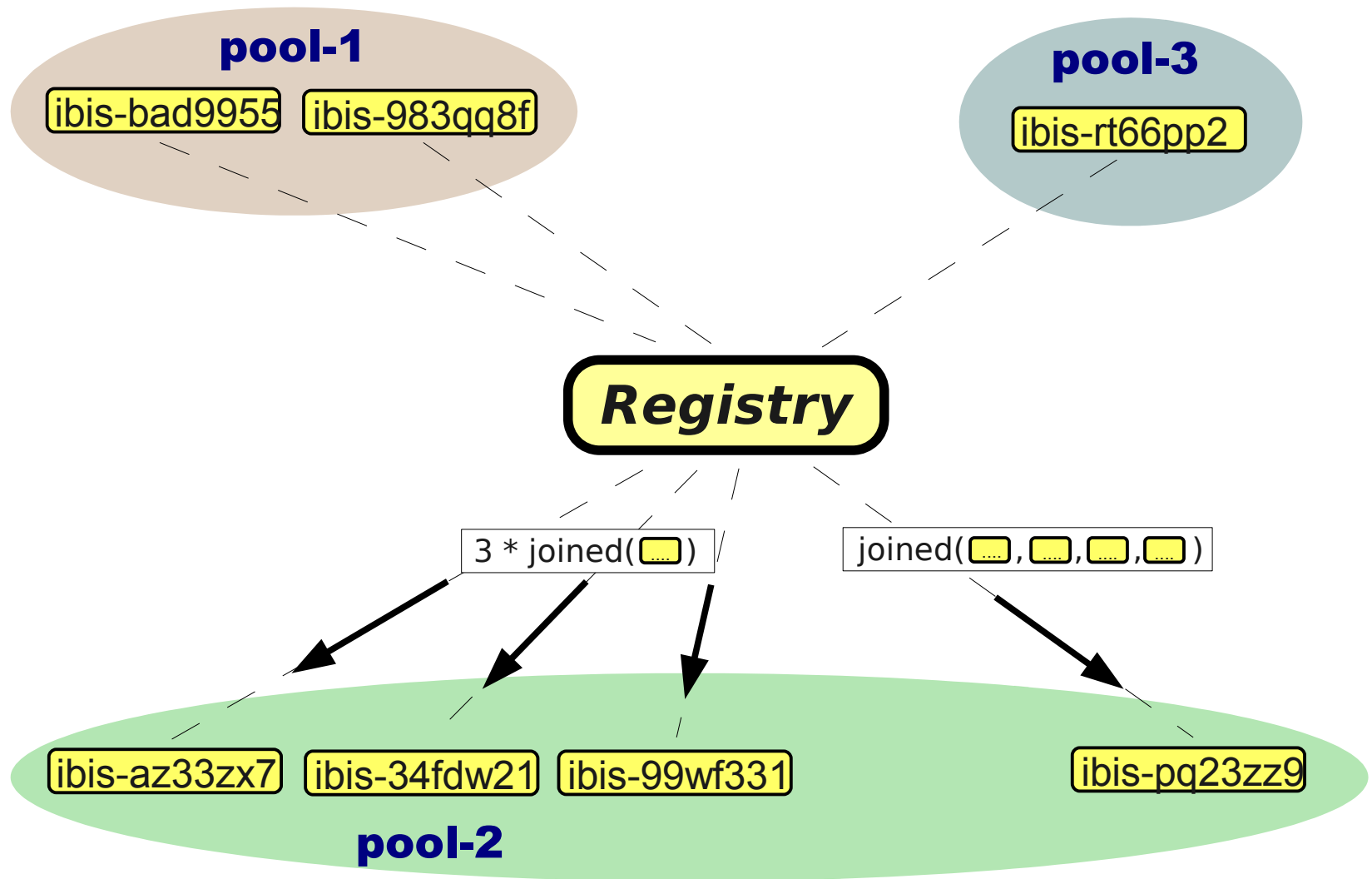


# ***Pools & Malleability***

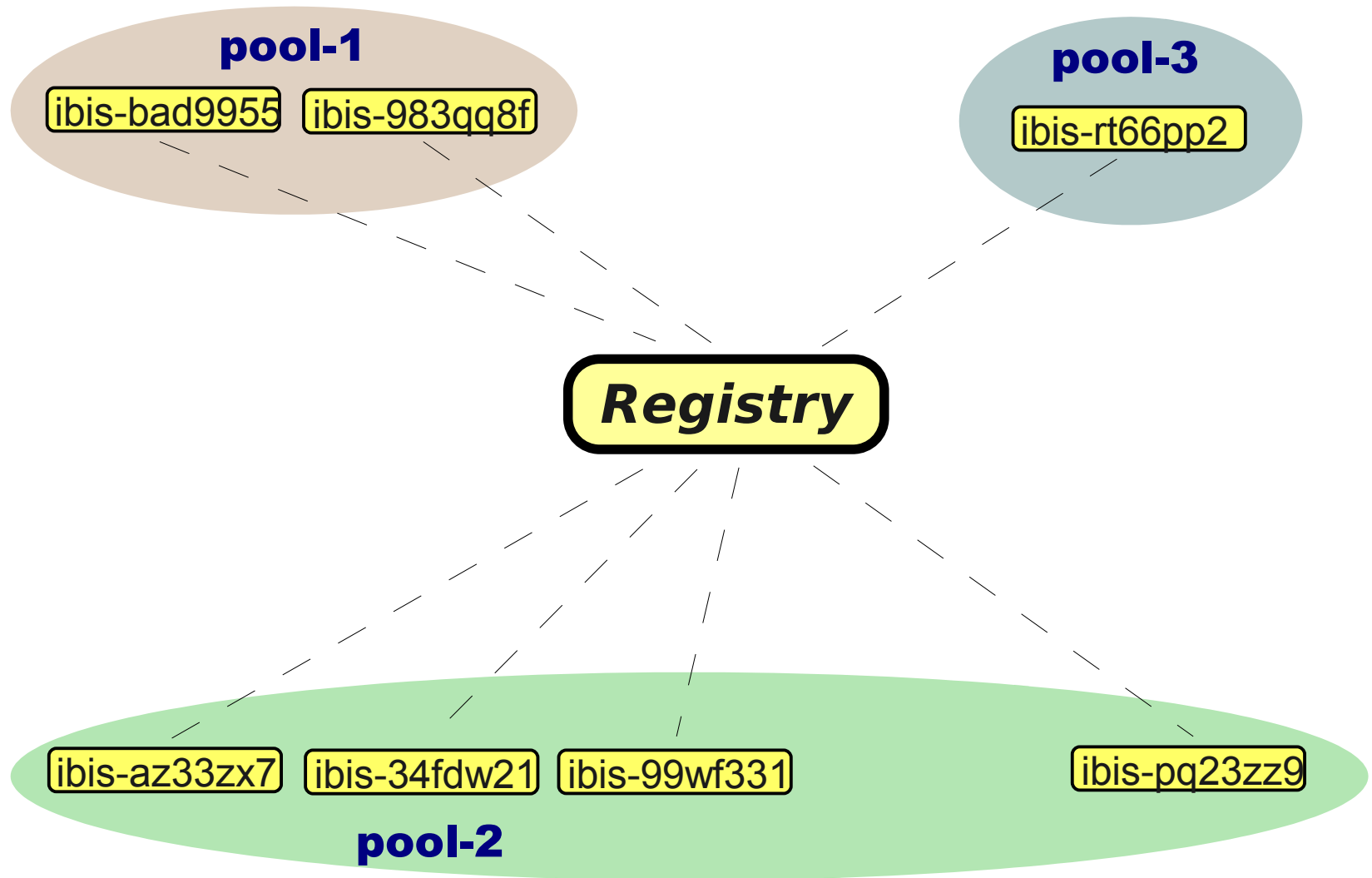




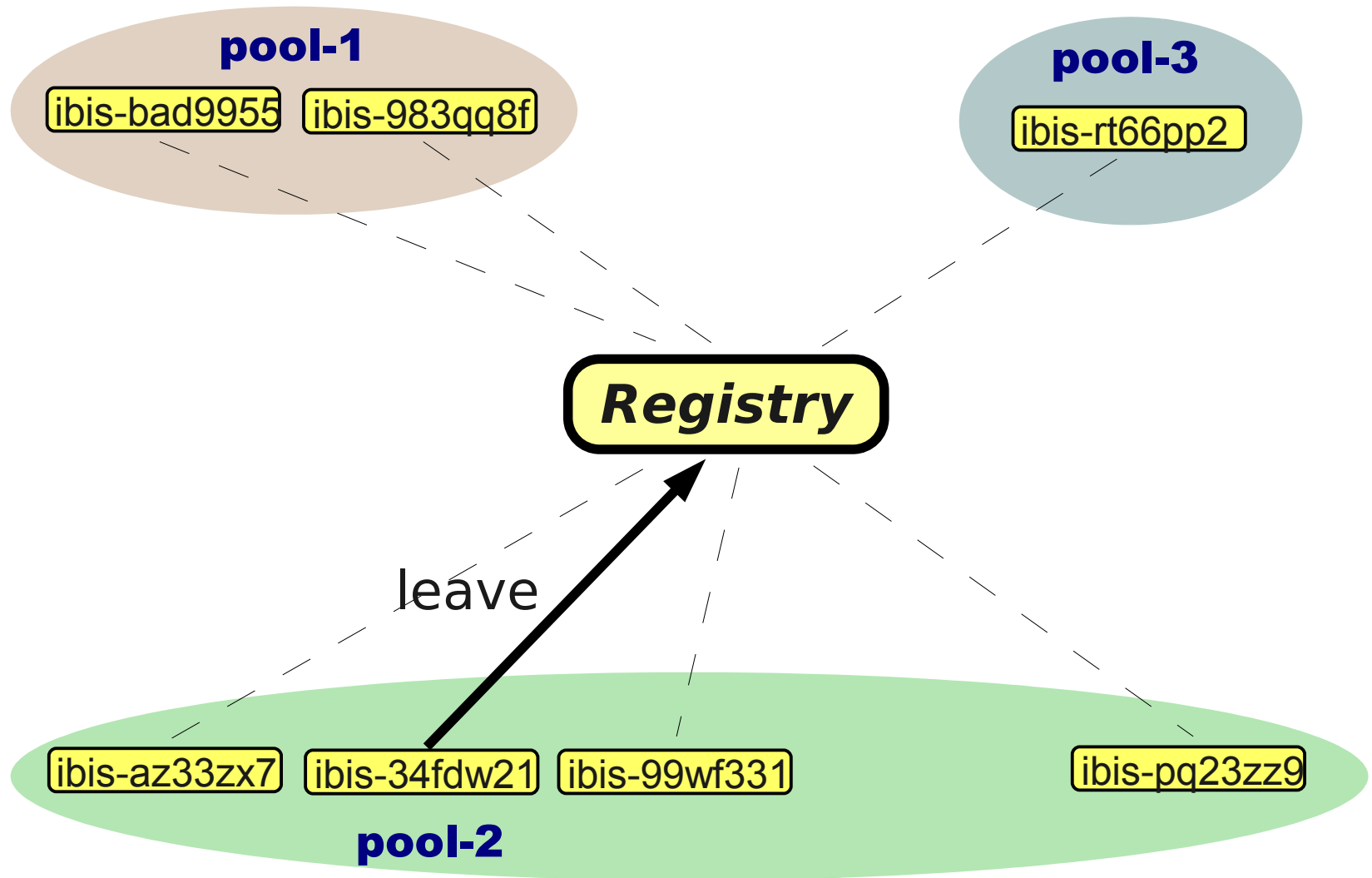
# Pools & Malleability



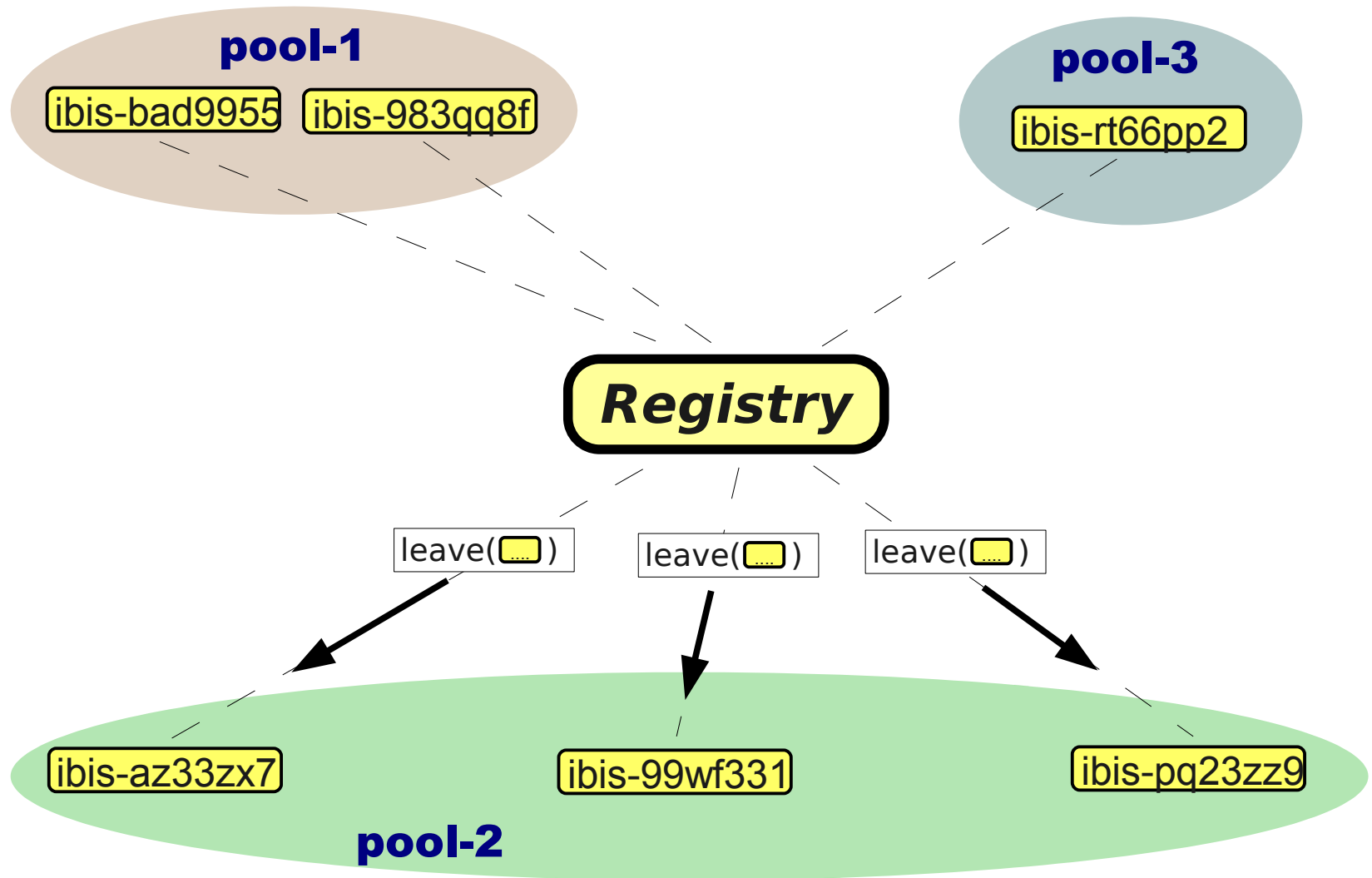
# ***Pools & Malleability***



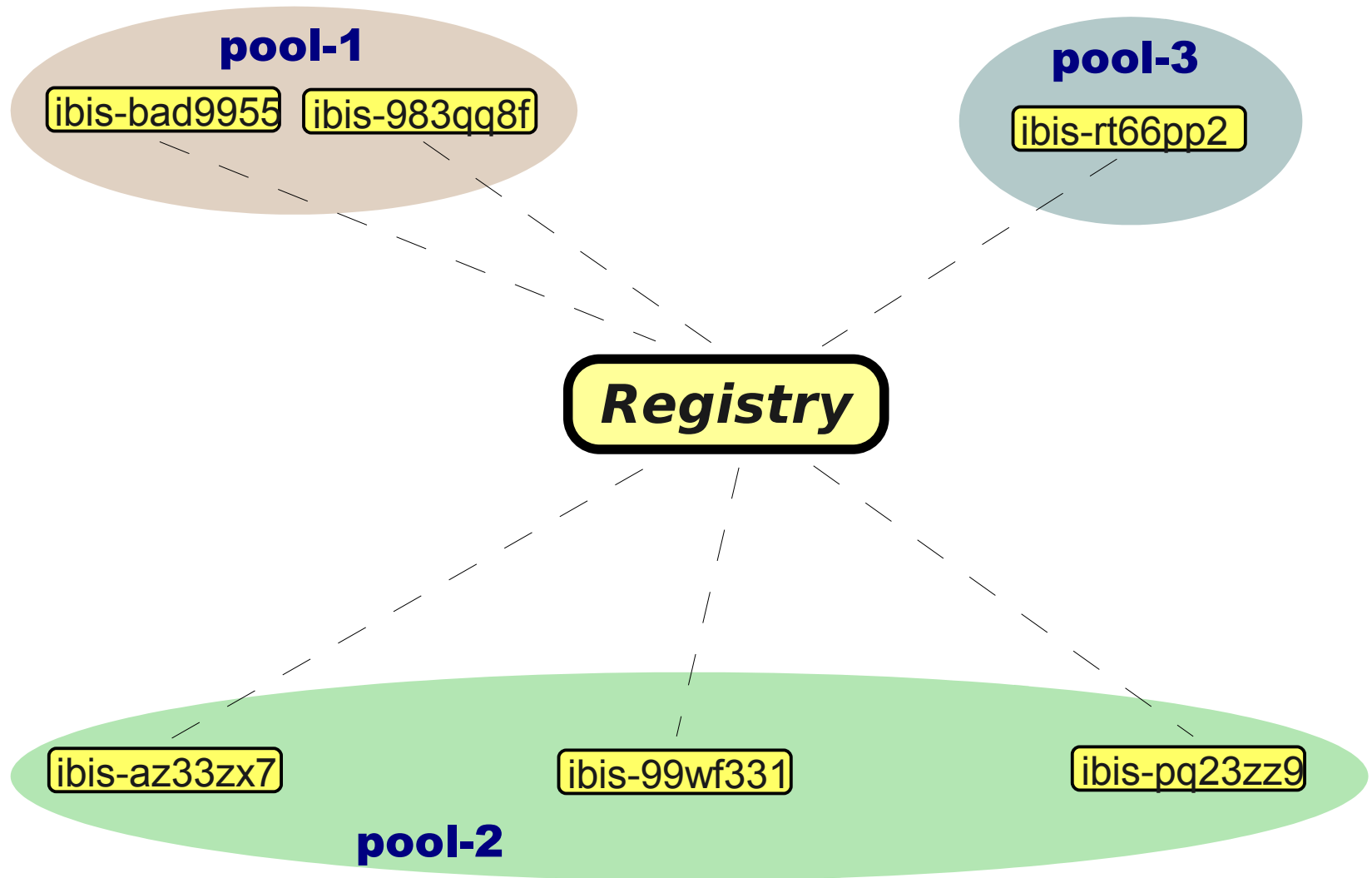
# ***Pools & Malleability***



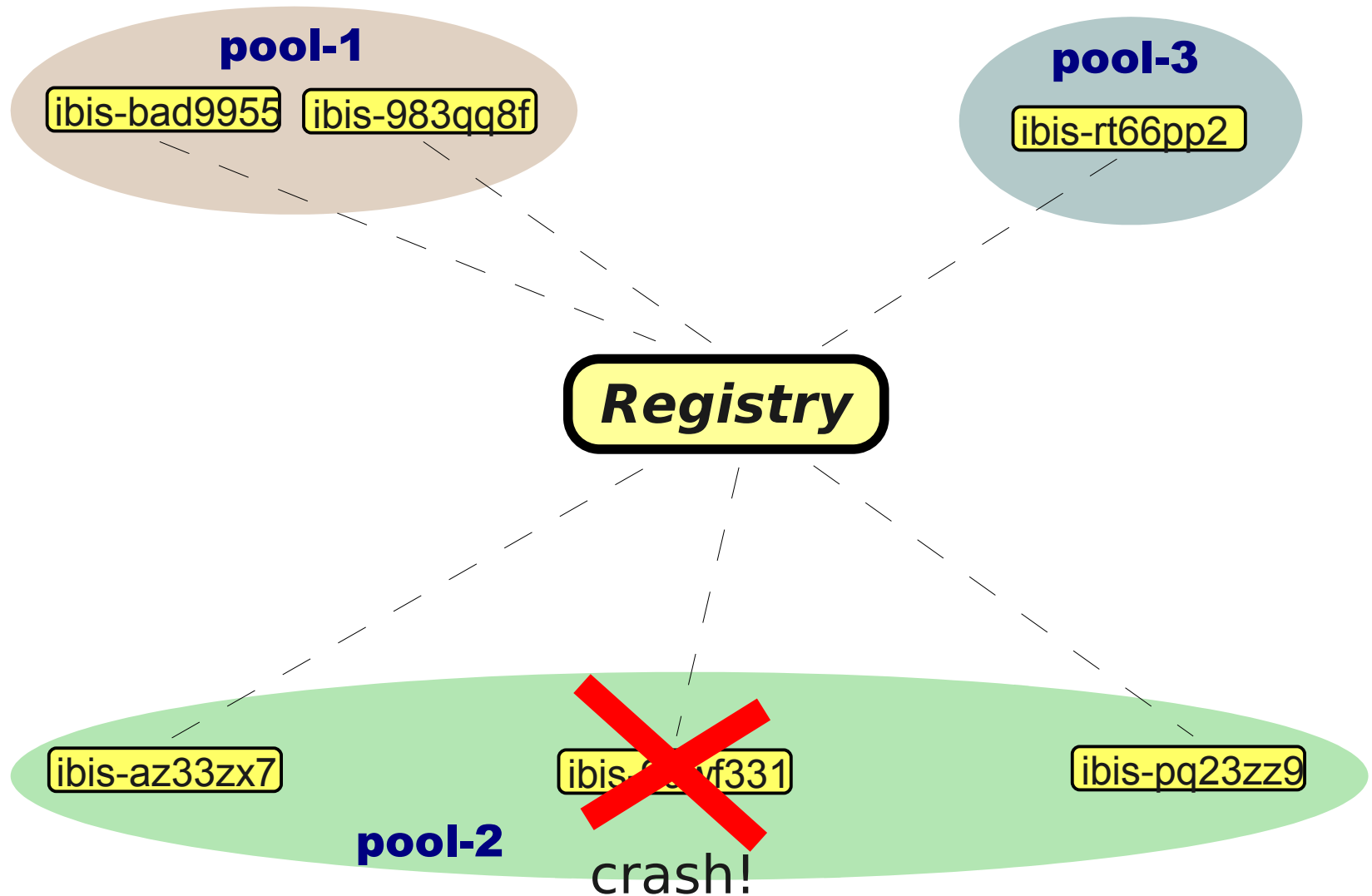
# Pools & Malleability



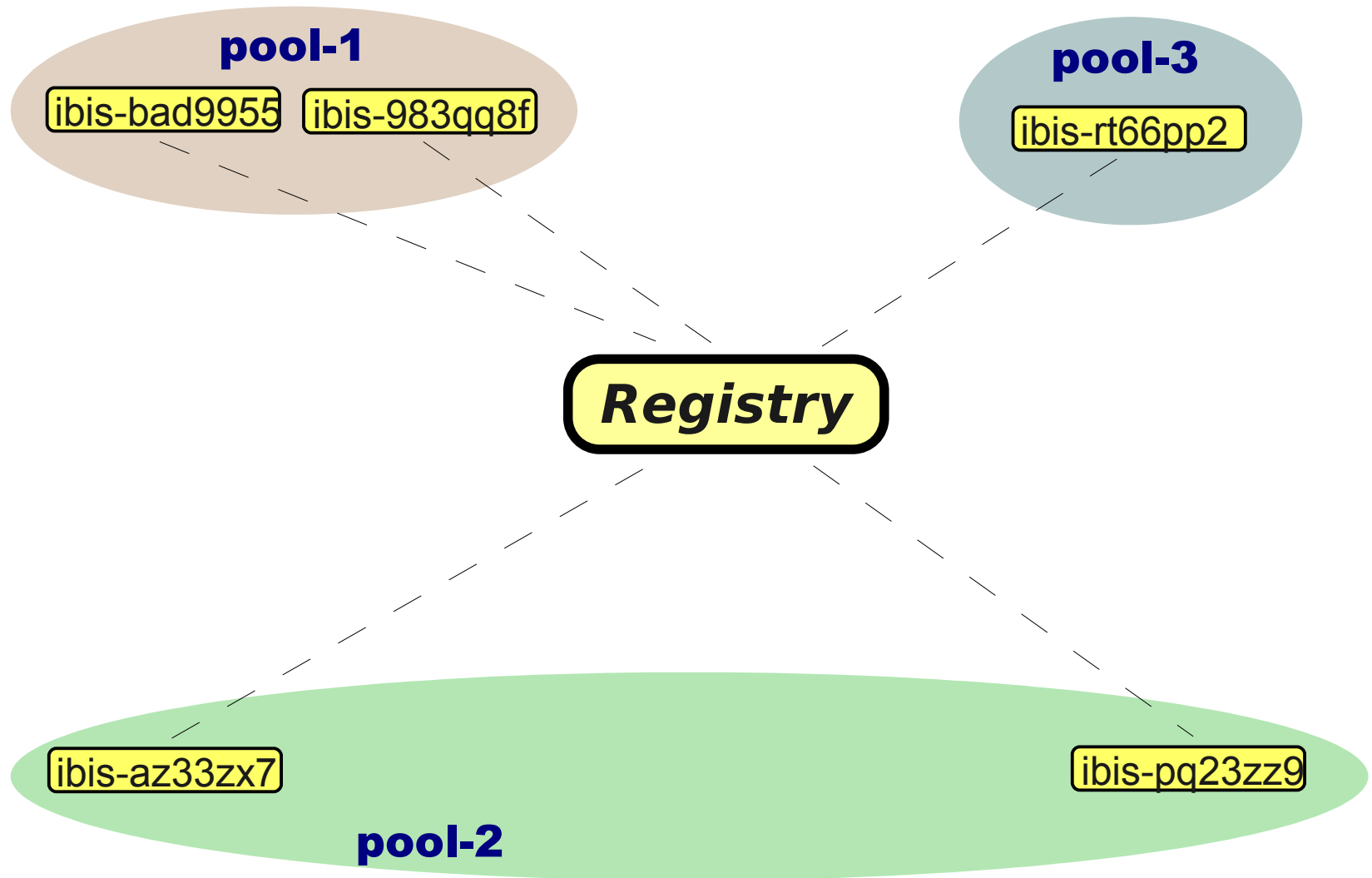
# ***Pools & Malleability***



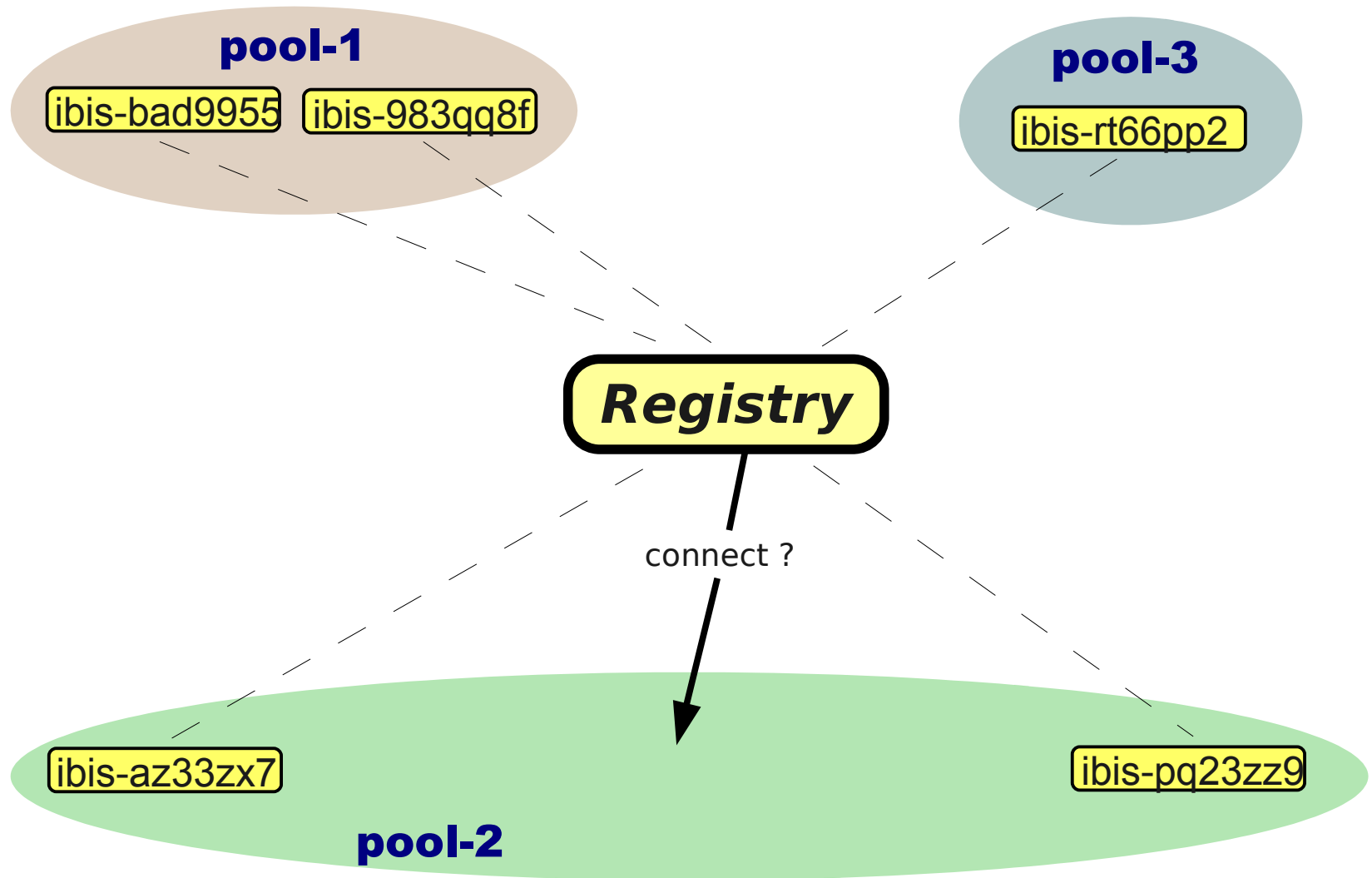
# ***Pools & Malleability***



# ***Pools & Malleability***

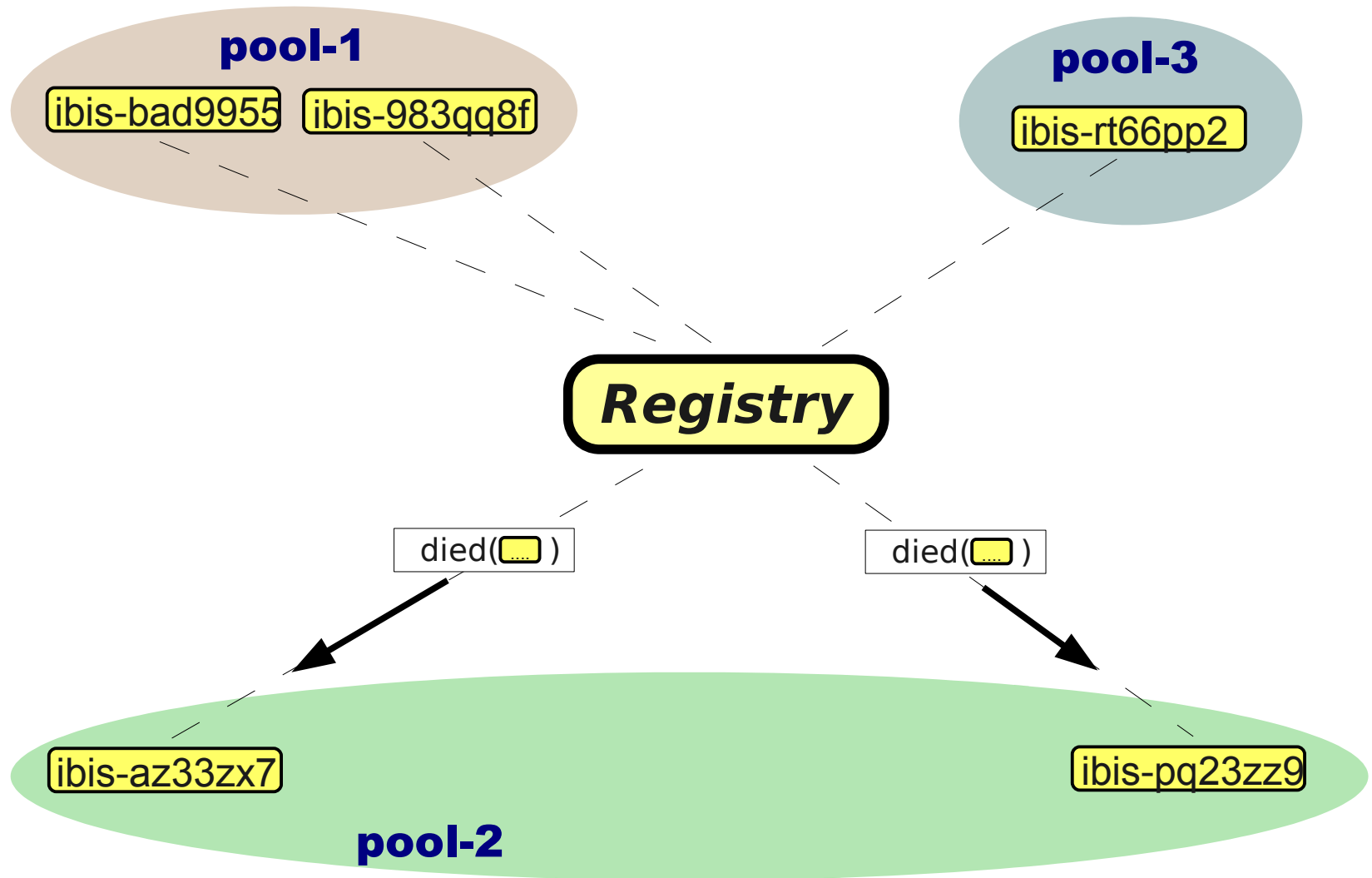


# ***Pools & Malleability***

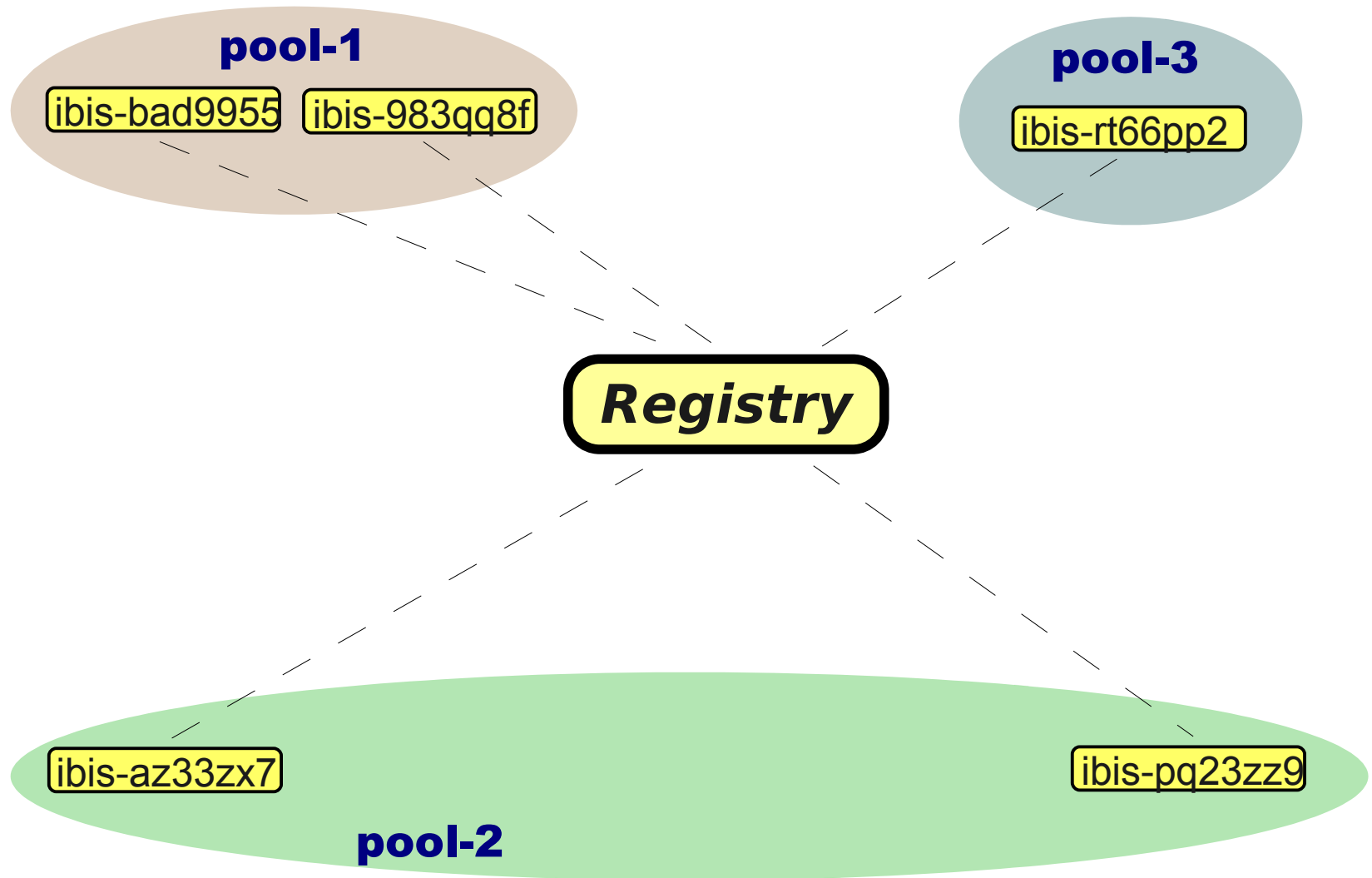




# Pools & Malleability



# ***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***

- IPL 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







# ***D-Grid***

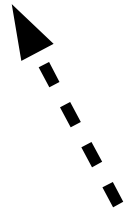
- Groot, GAT gebruikt bij
  - Max Planck inst. (Munchen)
    - Workflow (ProC)
      - Astronomen – simulaties in workflow
      - GUI op GAT
      - Henri weet meer ?
  - Potsdam (Berlijn, Alexander)
    - contactpers. / PR.





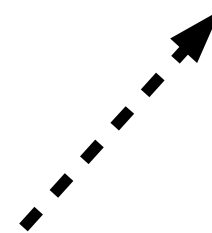
# ***Connection setup (1)***

ibis-az33zx7



Create Ibis

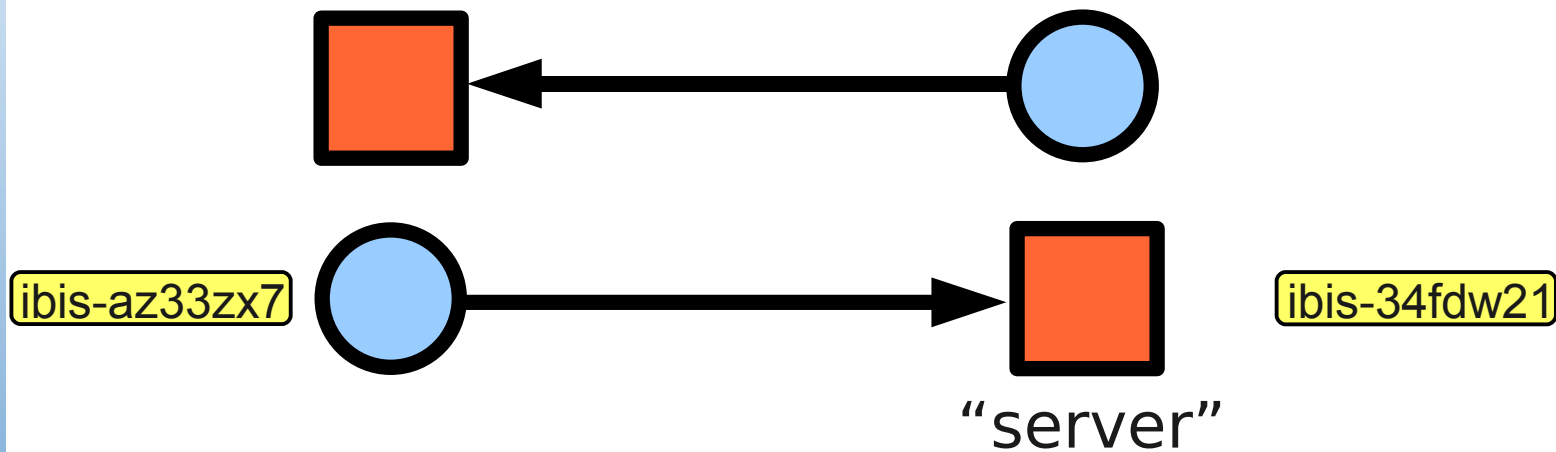
ibis-34fdw21



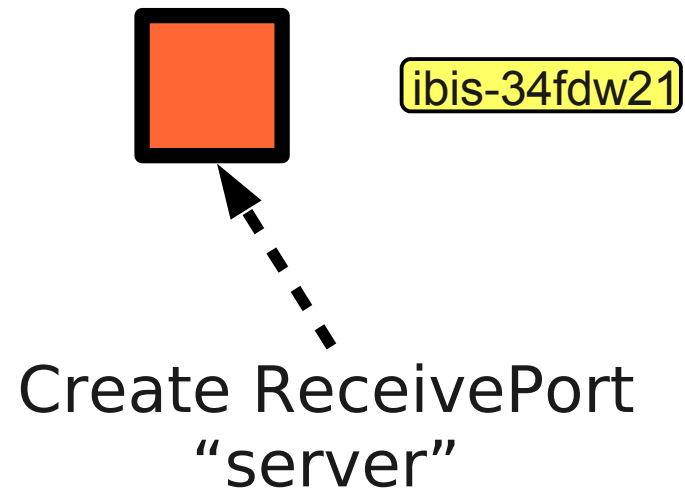
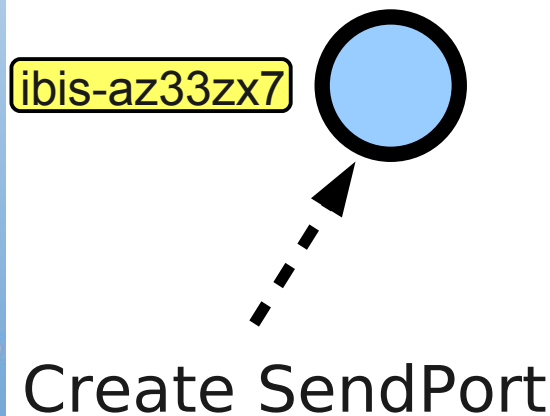
Create Ibis



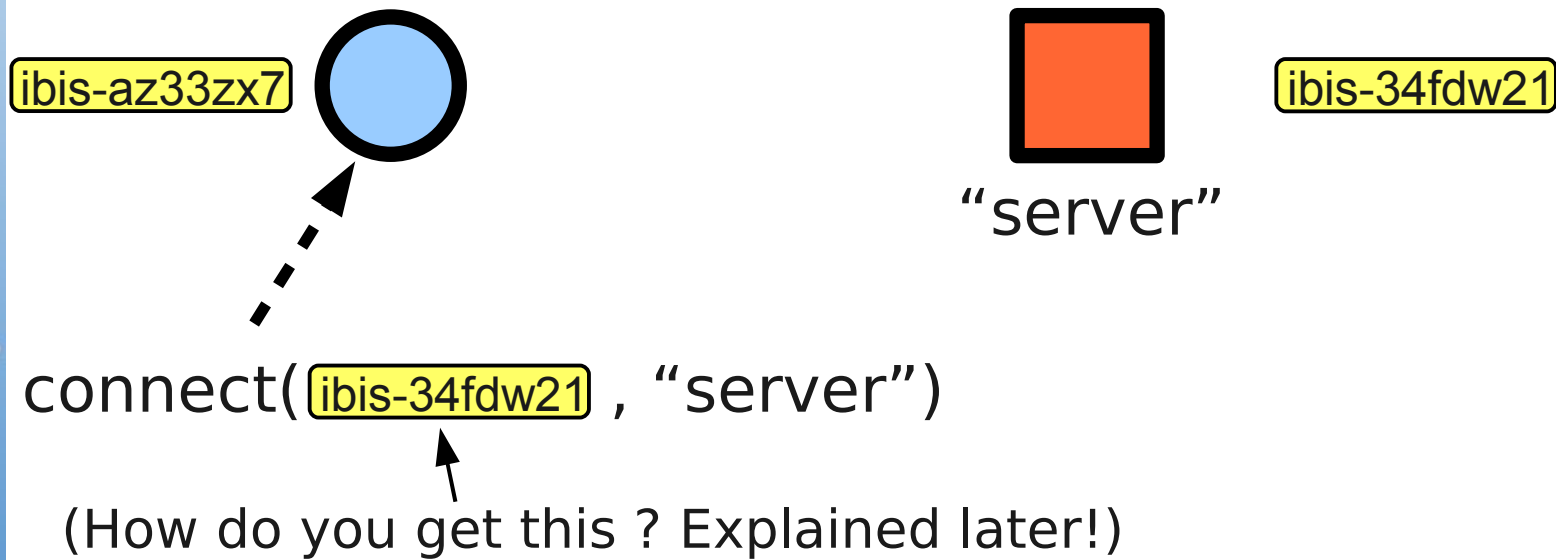
# *Connection setup (2)*



# *Connection setup (1)*



# Connection setup (1)



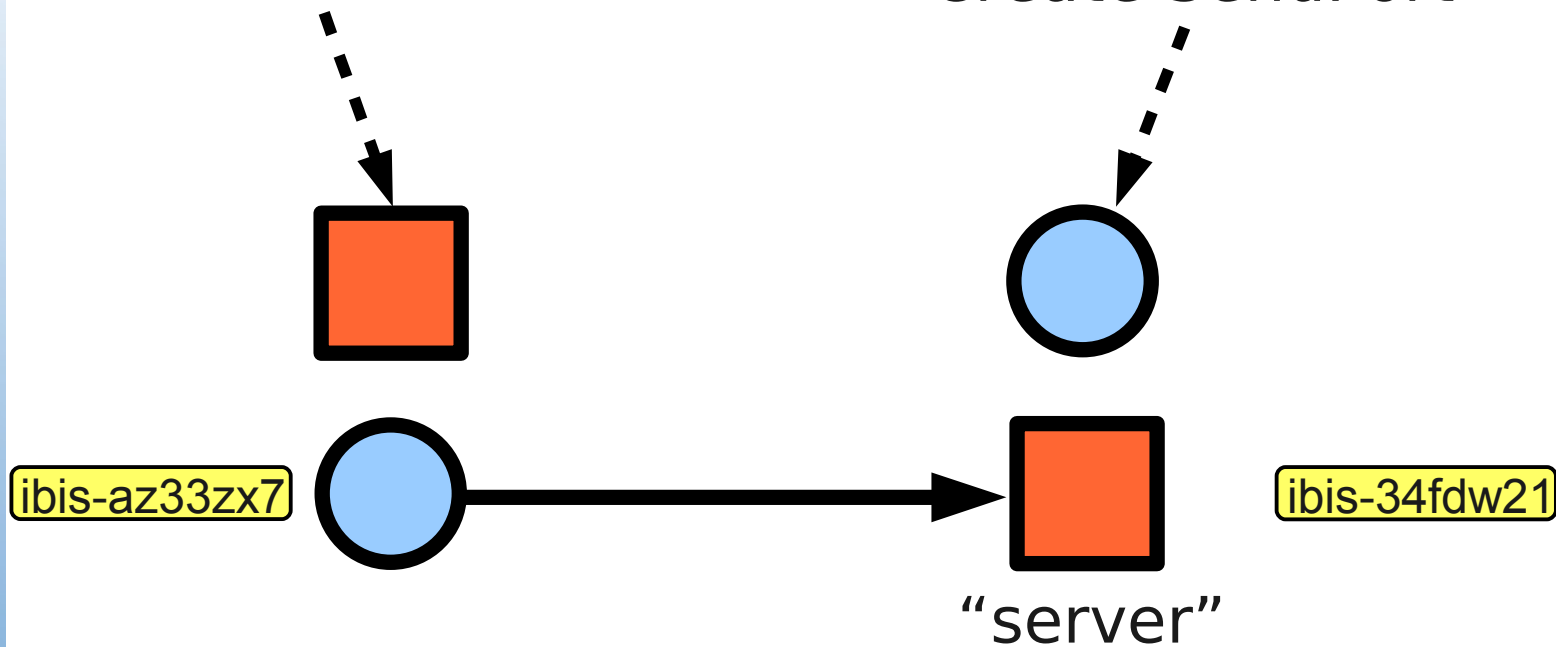
# *Connection setup (1)*



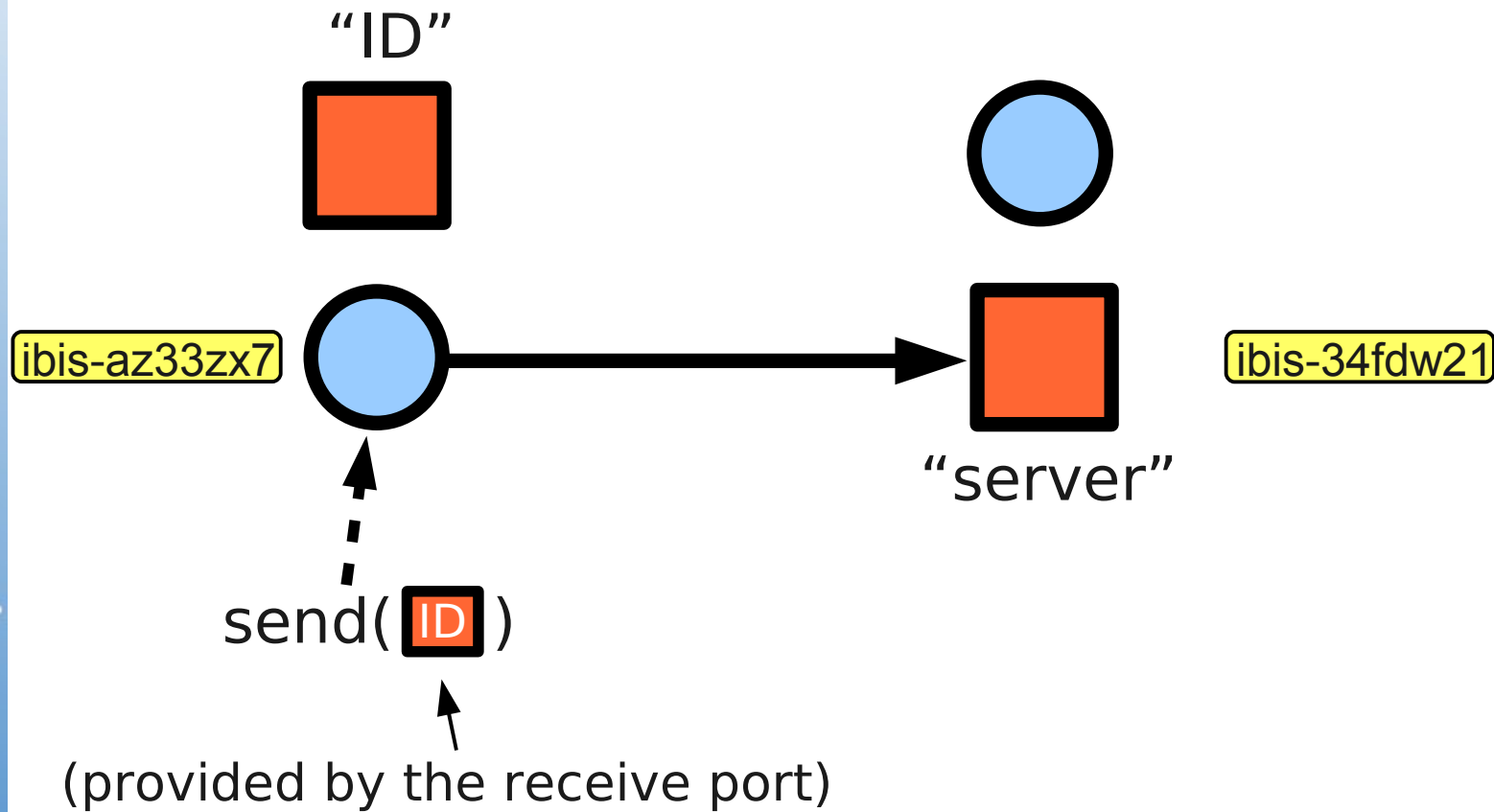
# Connection setup (2)

Create anonymous  
ReceivePort

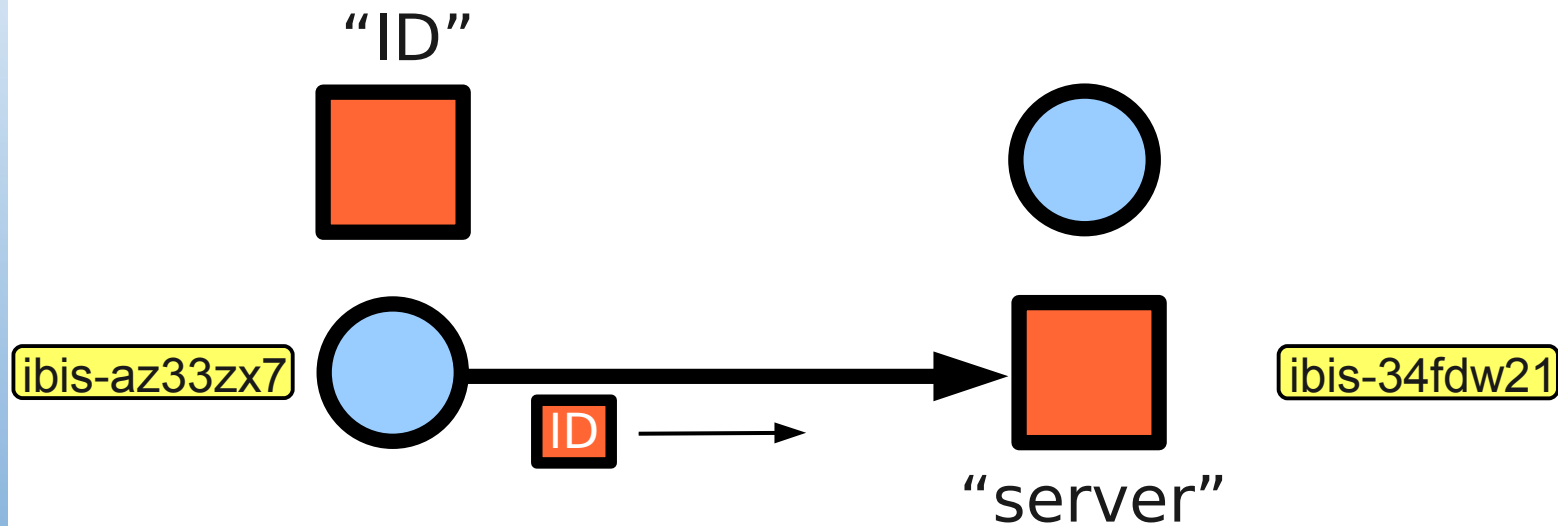
Create SendPort



# Connection setup (2)

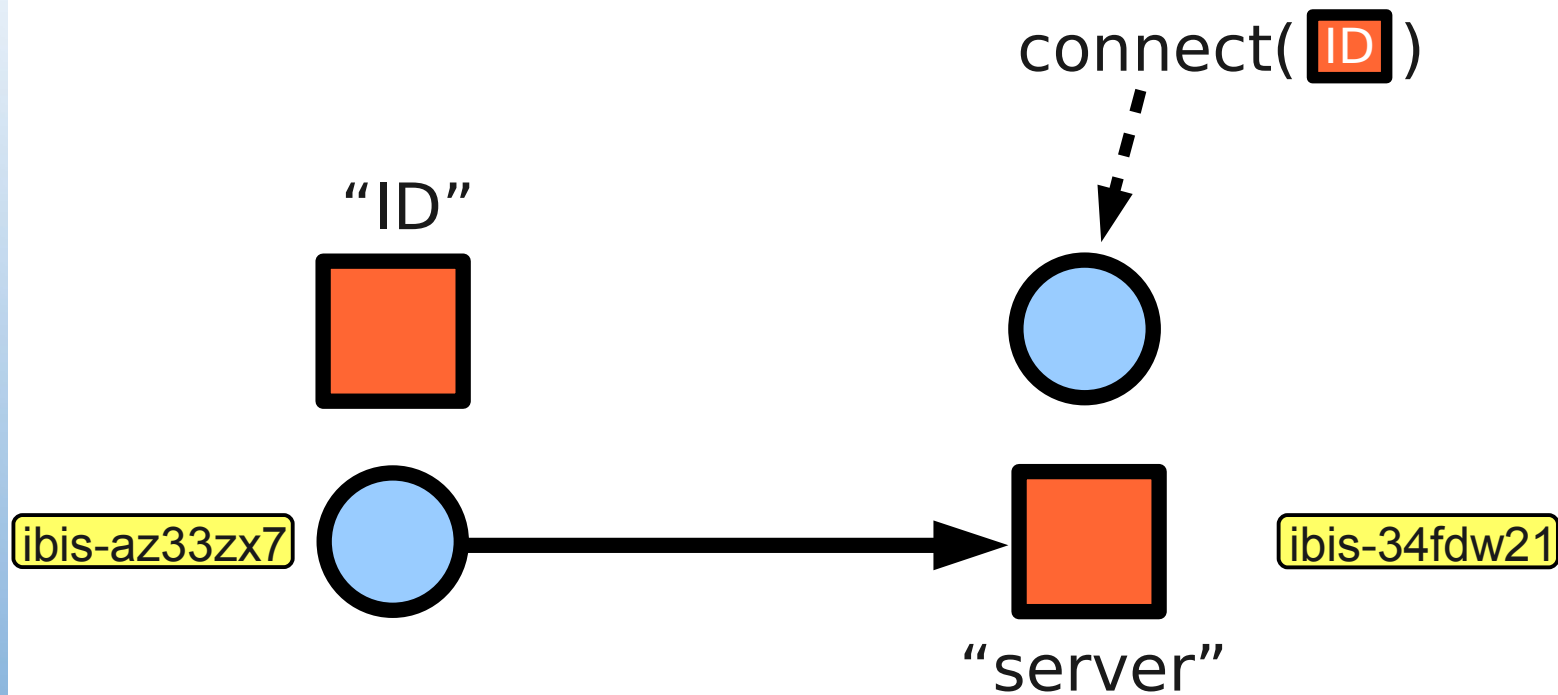


# Connection setup (2)





# Connection setup (2)



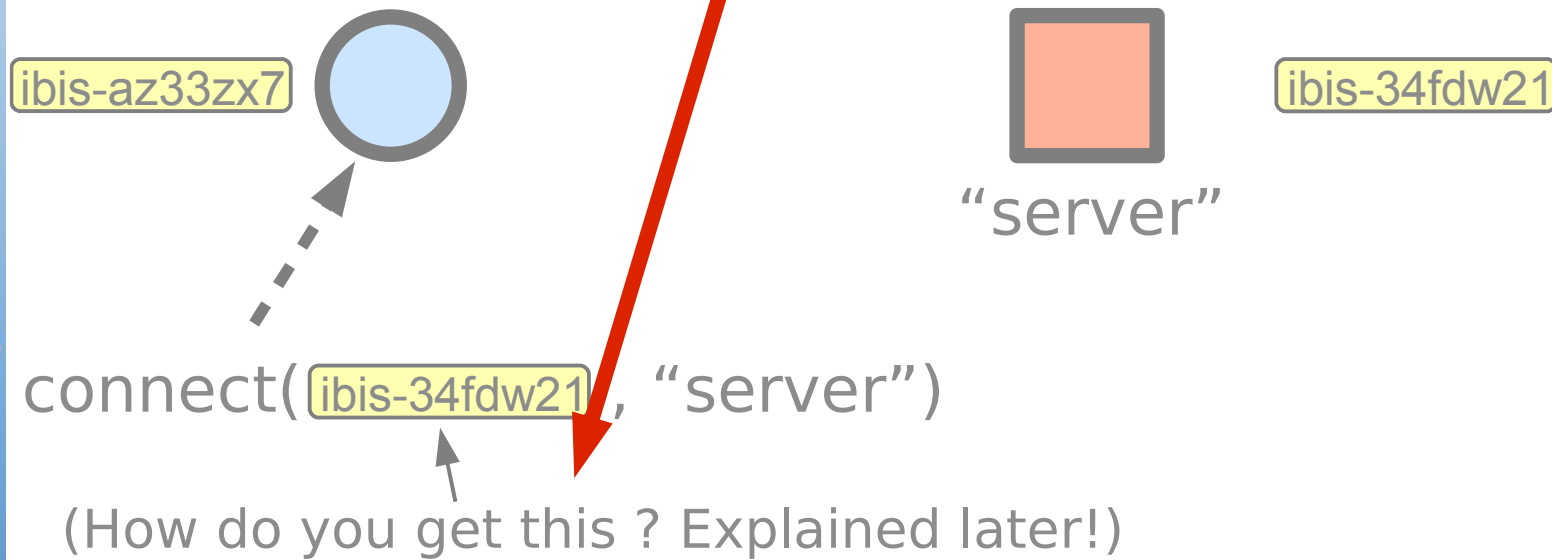
# ***Short Recap***

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



# Connection setup (1)

**Remember this question ?**



# ***IbisIdentifiers***

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



# ***Crashes and Malleability***

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

