Remote Method Invocation (RMI)
- RPCs applied to distributed objects
- Class: object-oriented abstraction
- Object: instance of class
  - Encapsulates data
  - Exports methods: operations on data
  - Separation between interface and implementation

Distributed Objects
- Interface resides on one machine, object on another
- RMIs allow invoking methods of remote objects
- Use proxies, skeletons, binding
- Allow passing of object references as parameters
Basic RMI Operation

Proxies and Skeletons
- Proxy: client stub
  - Maintains server ID, endpoint, object ID
  - Does parameter marshalling
  - In practice, can be downloaded/constructed on the fly
- Skeleton: server stub
  - Does demarshalling and passes parameters to server
  - Sends result to proxy

Binding a Client to an Object
- Loading a proxy in client address space
- Implicit binding:
  - Bound automatically on object reference resolution
- Explicit binding:
  - Client has to first bind object
  - Call method after binding

Parameter Passing
- Less restrictive than RPCs
- Supports system-wide object references
- Copy local objects, pass references of remote objects
Java RMI
- java.rmi package
- 3 components:
  - Remote object interface
  - Client code: Invokes remote object methods
  - Server code: Implements remote object
- RMI compiler: Generates stub and skeleton code
- Can pass local objects as well as remote references

Serialization
- Used to pass objects
- Default serialization: Flatten all fields of object
- User-defined serialization:
  - Provide methods for reading and writing object
  - Can choose what fields or data to write and read
- Allows passing objects over streams
  - Files, sockets, etc.
- RMI uses serialization for parameter marshalling

Message-oriented Communication
- Allows various combinations of persistence and synchronization
- Different protocols implement different combinations
  - MPI: Transient Communication
  - Message-queuing: Persistent communication

Message-Passing Interface (MPI)
- Sockets designed for network communication (e.g., TCP/IP)
  - Support simple send/receive primitives
  - General-purpose
- MPPs and COWs require
  - Advanced primitives
  - Other forms of buffering, synchronization
- Large number of incompatible proprietary libraries and protocols
  - Need for a standard interface
Message-Passing Interface (MPI)
- Message-passing interface (MPI)
  - Hardware independent
  - Designed for parallel applications
- Communication between groups of processes
  - Each endpoint is a (groupID, processID) pair
- Messaging primitives support different forms of transient communication
  - MPI_bsend: transient asynchronous
  - MPI_ssend: delivery-based transient synchronous

Message-Queuing Model
- Support asynchronous persistent communication
  - Intermediate storage for message while sender/receiver are inactive
- Communicate by inserting messages in queues
- Loosely coupled communication
  - Sender is only guaranteed that message will be eventually inserted in recipient’s queue
  - No guarantees on when or if the message will be read
- Examples: IBM Websphere MQ, MS Message Queuing, Amazon Simple Queue Service

Message-Queuing Entities
- Queue managers: Handle message sending/receiving from queues
  - Storage, communication, notification
- Relays: Message routers
- Message brokers: App-level gateways
  - Can transform messages between formats
- Publish/Subscribe systems
  - Brokers match applications to messages

Stream-oriented Communication
- So far:
  - Communication in terms of independent units (RPC calls, messages, etc.)
  - No strict timing relation between these units
- What if we want to transmit:
  - Videos
  - Sensor data
Stream-oriented Communication
- Stream: Sequence of data units
  - E.g.: files, pipes, videos
- Discrete data streams
  - Need to transmit in order, no other constraints.
    - E.g.: Temperature sensor readings
- Continuous media streams
  - Fundamental temporal relations between different data items.
    - E.g.: Video frames

Continuous Media Streams
- QoS requirements:
  - Timing/rate constraints: e.g.: frame rate
  - Jitter constraints
  - Synchronization constraints: audio+video streams
- Techniques:
  - Client-side: Buffering
  - Server-side: Stream shaping
  - Multi-stream synchronization