Remote Procedure Call
Waldo et al, "A note on ..."
Technical tidbits

ID2010

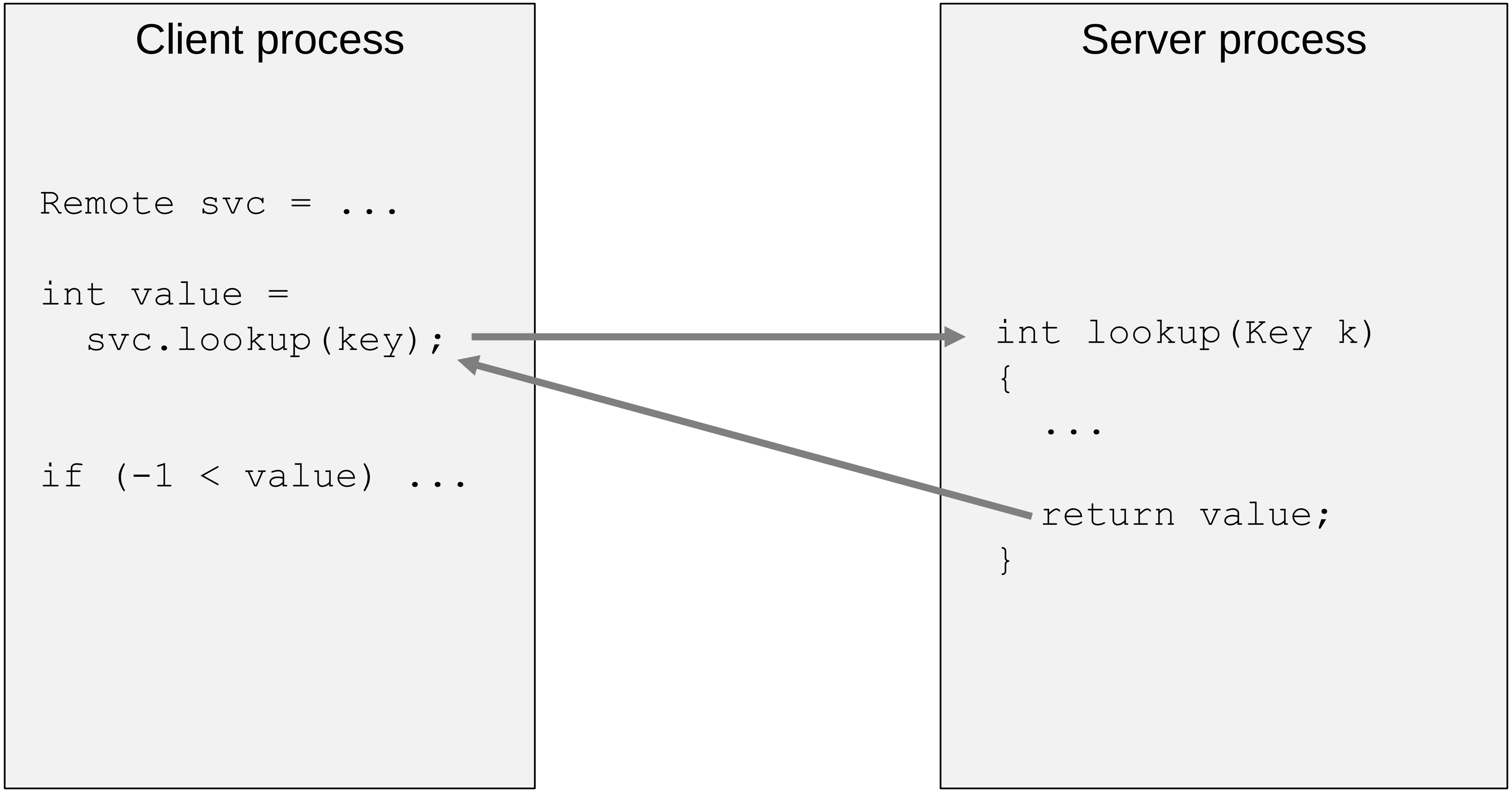
2022

RPC

- Remote Procedural Call (RPC)
- Remote Method Invocation (RMI)
- Message Oriented Middleware
- Sockets

Remote Procedure Call

programming view



Client process

```
Remote svc = ...

int value =
  svc.lookup(key);


if (-1 < value) ...
```

Server process

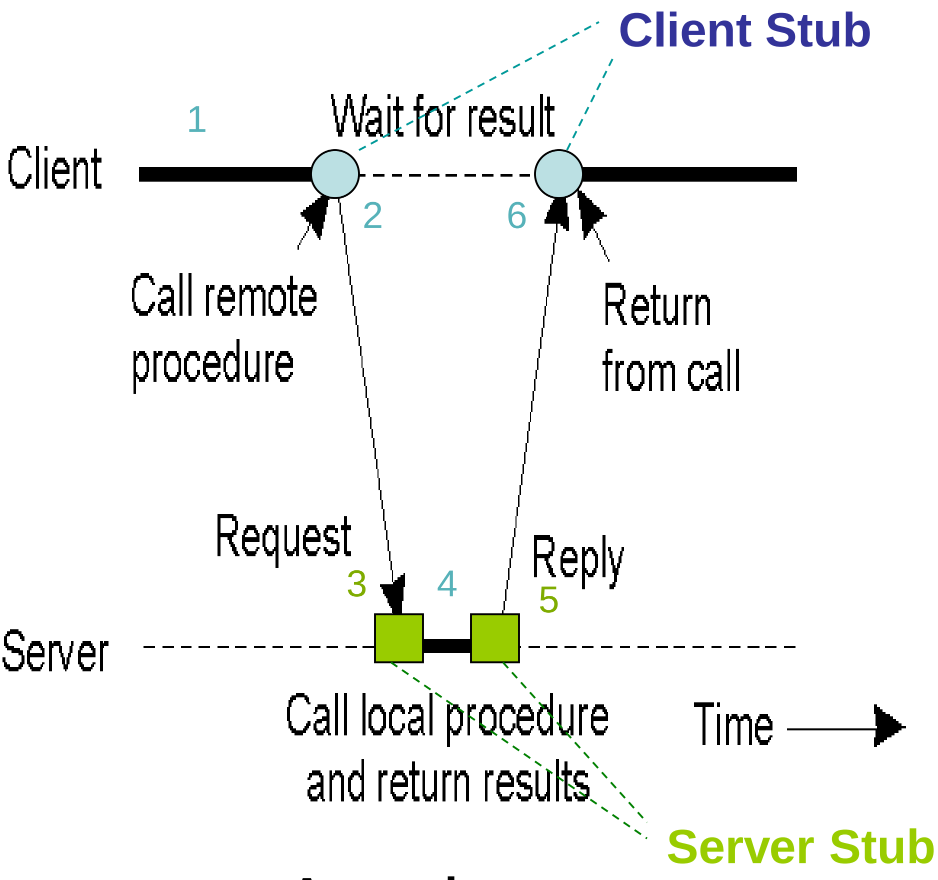
```
int lookup(Key k)
{
  ...

  return value;
}
```

Remote Procedure Call (RPC)

Extend the procedure call over the network by allowing programs to call procedures located on other machines through Stubs:

1. Client program calls *client stub* to place a remote procedure call
2. Client stub builds a request message and sends to remote server
3. *Server stub* receives the message and unpacks parameters, calls the local procedure
4. Procedure executes and returns result to the server stub
5. Server stub packs it in message, and sends back to client stub
6. Client stub unpacks result, returns to client program



A synchronous Client/Server Model

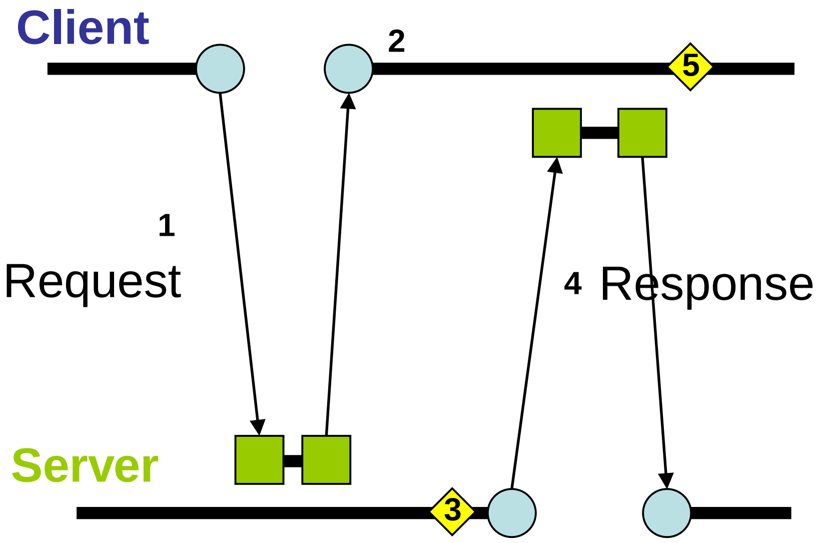
Remote Procedure Call (RPC)

Extend the procedure call over the network by allowing programs to call procedures located on other machines

1. The client calls the server to place a service request
2. While the server processes other requests, the client keeps executing
3. The server sees the request, and responds to it
4. The server calls back to the client with the result
5. The client sees the response and acts upon it

+ Higher parallelism

- Higher complexity



An asynchronous and symmetric Client/Server Model

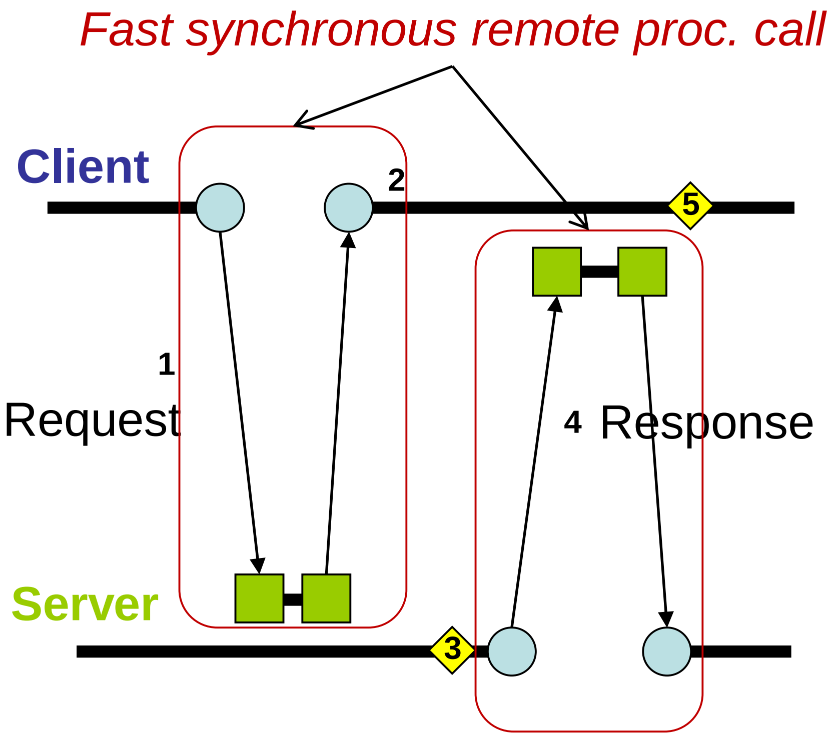
Remote Procedure Call (RPC)

Extend the procedure call over the network by allowing programs to call procedures located on other machines

1. The client calls the server to place a service request
2. While the server processes other requests, the client keeps executing
3. The server sees the request, and responds to it
4. The server calls back to the client with the result
5. The client sees the response and acts upon it

+ Higher parallelism

- Higher complexity



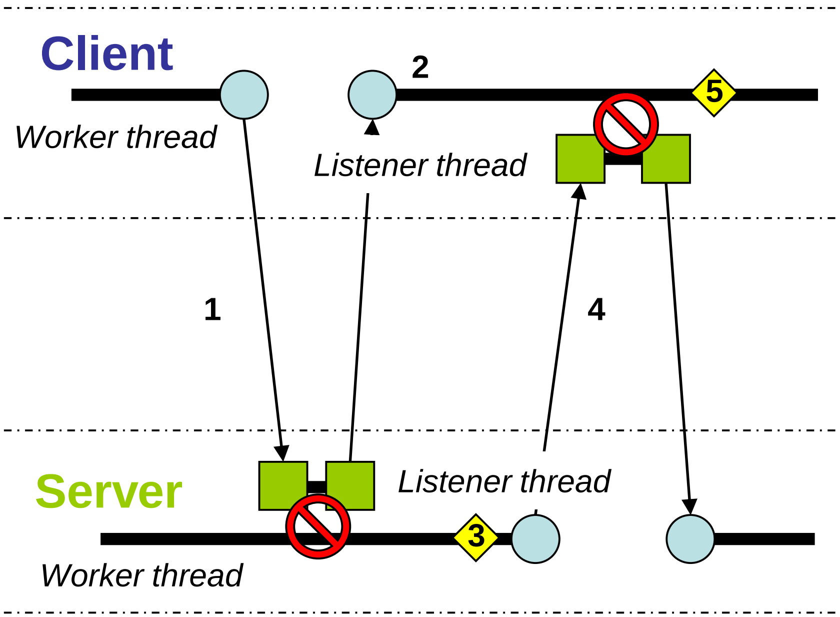
An asynchronous and symmetric Client/Server Model

Remote Procedure Call (RPC)

Extend the procedure call over the network by allowing programs to call procedures located on other machines

1. The client calls the server to place a service request
2. While the server processes other requests, the client keeps executing
3. The server sees the request, and responds to it
4. The server calls back to the client with the result
5. The client sees the response and acts upon it

+ Higher parallelism
- Higher complexity



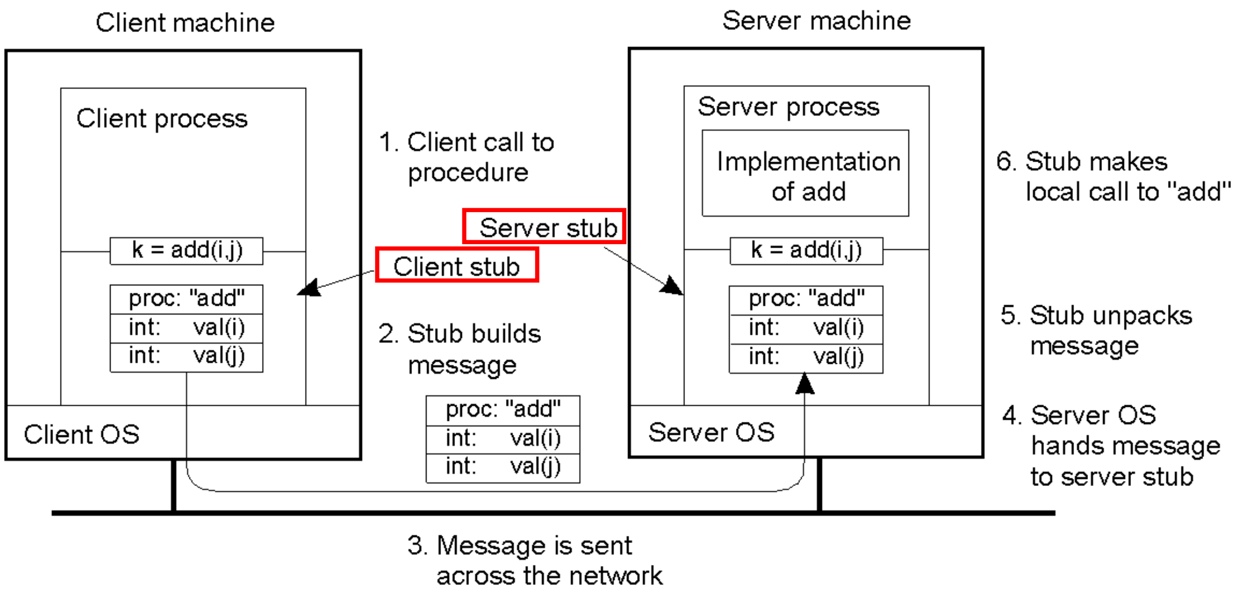
An asynchronous and symmetric Client/Server Model

Client & Server Stubs

The Stubs take charge of:

1) Building the RPC message (parameters and results), also called marshaling and unmarshaling

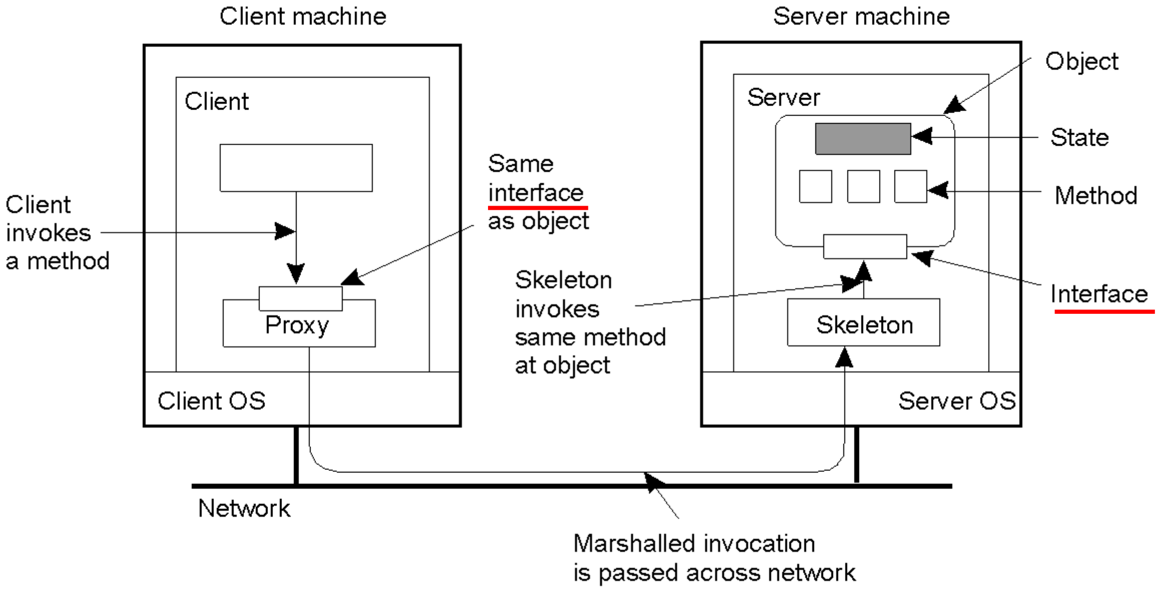
2) Establishing the connection to transfer messages.



Remote Method Invocation

Java

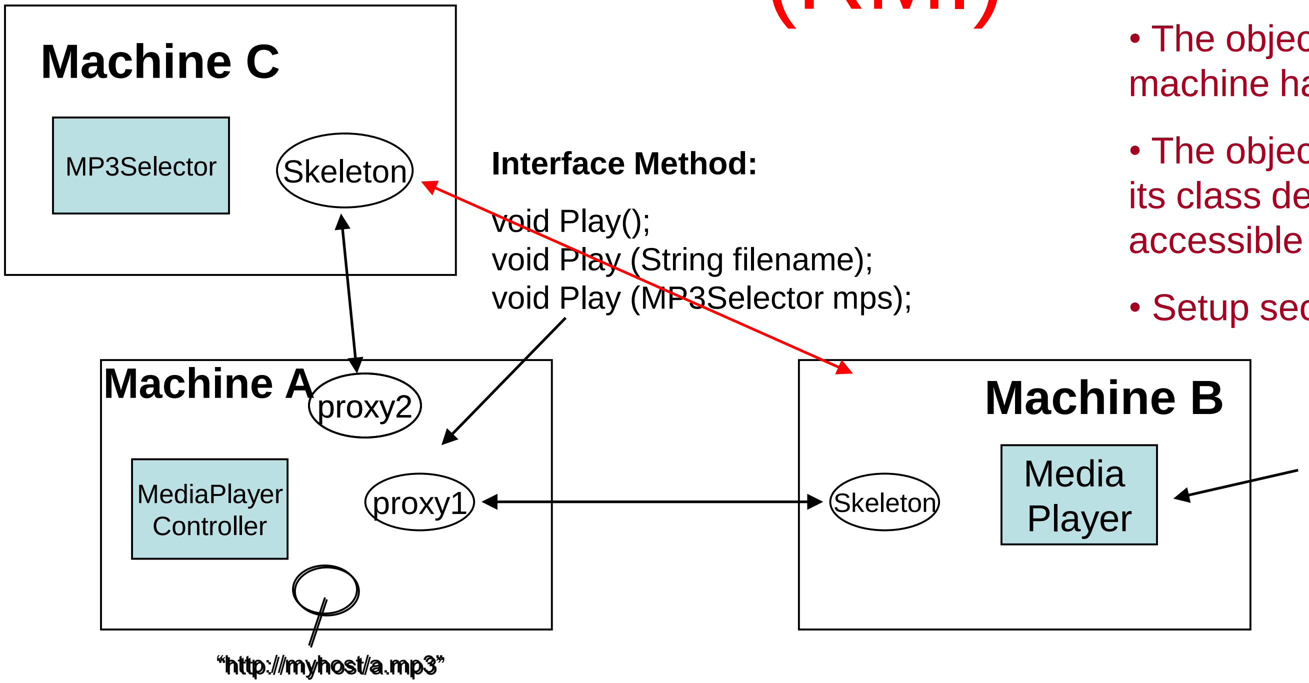
- Object-oriented technology encapsulates data, (**state/Property)** and operations (**method**) on those data
- This encapsulation offers a better transparency for system design and programming
- The principle in RPC can be equally applied to objects



- Client uses **proxy** (a local representative of the remote object) to operate with the remote one.
- Proxy/Skeleton is analog to the stubs in RPC, in addition, it presents an object view.

Passing Object by Value or Reference (RMI)

- The object to send to another machine has to be *Serializable*.
- The object to send has to have its class definition in a publicly accessible *codebase*.
- Setup security policy file



Interface Method:

void Play();
void Play (String filename);
void Play (MP3Selector mps);

Interface Method:

void Play ();
void Play (String filename);
void Play (MP3Selector mps);

Three cases:

1) Play () without parameters. // only method name will be sent

2) Play ("http://myhost/a.mp3") // send filename as a copied object (value/copy)

```
3) Play (mps) {                          // send the copy of proxy2
       play(mps.getLatestMP3())          // (reference to MP3Selector)
   }
```

https://docs.oracle.com/javase/tutorial/rmi/index.html

Conclusion (RPC & RMI)

- To be able to access a remote object, a local stub (proxy) which refers to the remote object is required.
- The stub appears as a local object, but delivers the received accesses to the remote object.
- The stub can be passed (e.g. in Java RMI) to other programs (on remote computers) to share the access to the same remote object.

Conclusion (RPC & RMI)

- Another way to access a remote object is to make a cloned local copy.
- This improves performance by removing the call delay over the network, but …
- Consistency becomes an issue if they need to be synchronized since they are now two independent objects (from the same class) in the network.

Conclusion (RPC & RMI)

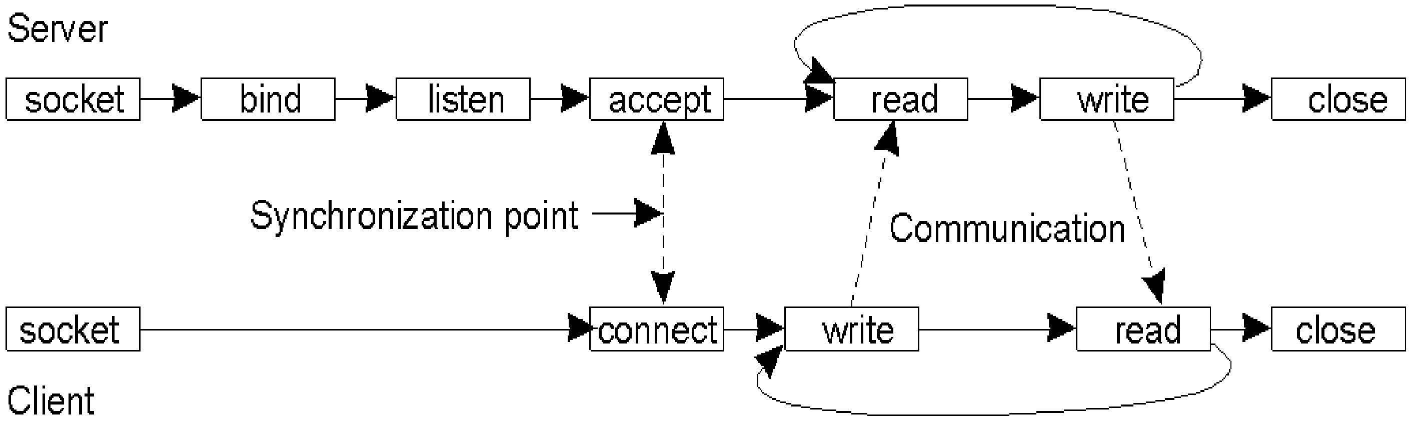
- Stubs, Proxies and Skeletons …
  - hides the complexity of marshaling and unmarshaling.
  - hides the network communication
  - enhances the access transparency to the upper-layer applications.

Conclusion (RPC & RMI)

- RPC and RMI use a transient synchronous communication model:
  - – The sender blocks until it receives a reply from the other side.
  - – This model is not suitable for pervasive computing scenarios where time is critical.

Berkeley Sockets

TCP/UDP Network communication like plug-in sockets



Connection-oriented communication (TCP) pattern using sockets.

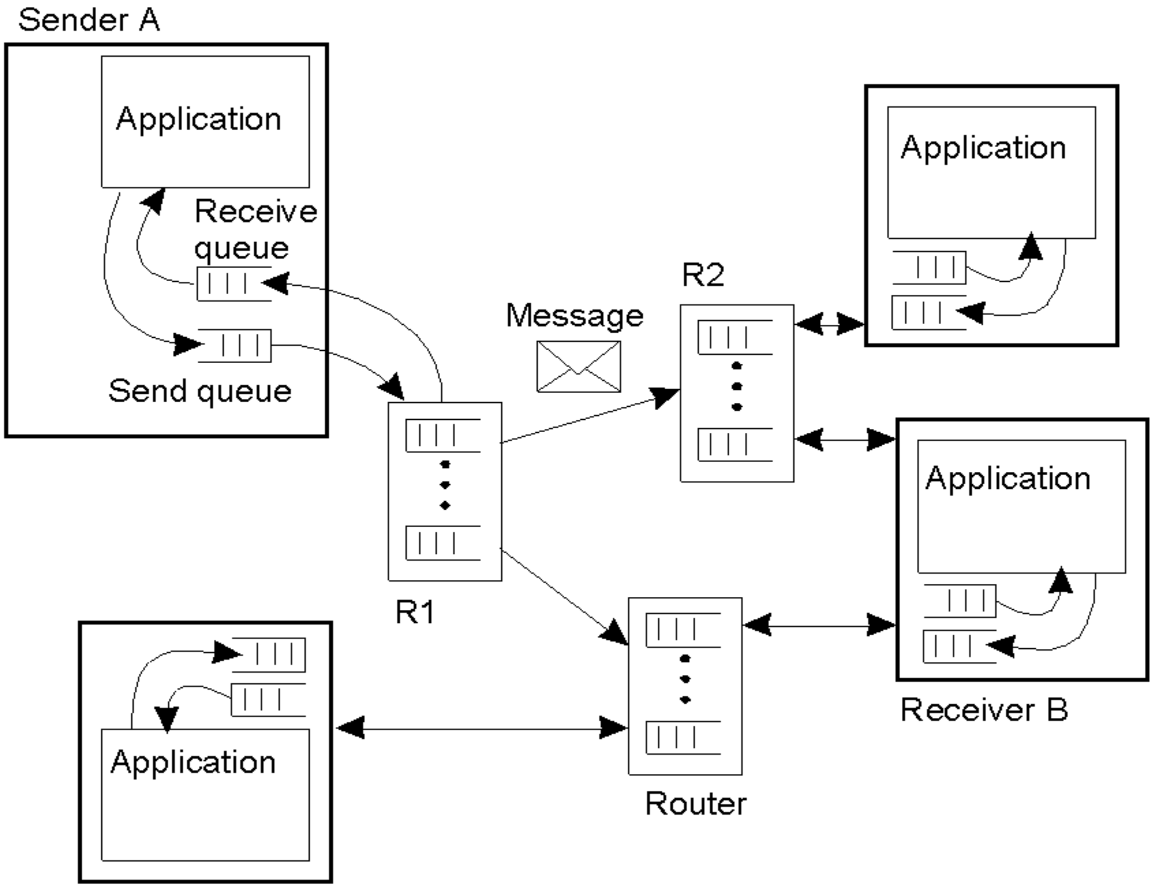
- UDP communication is asynchronous, so does not have the synchronization point as in TCP
- UDP server just creates a UDP socket and then receives (blocking), and UDP client has no “connect” phase to block, but just sends.
- UDP port =/= TCP port, they may use the same port number without conflict

Message-Oriented Middleware

- Socket communication gives an easy-to-use abstraction to network programming.
- Sockets are supported by most programming languages and operating systems supporting networks.
- To achieve efficiency and simplicity, many middlewares are implemented in terms of message delivery based on (hidden) socket communication.
- This is called Message-oriented middleware (MOM).
- *Examples: MQTT, XMPP, IBM MQ, Amazon SNS.*

General Architecture of a Message-Queuing System

- Messages are delivered in a sorting-storing-forwarding fashion
- Applications are loose-coupled by asynchronous messages (events)
- R1, R2 are Message Servers in MOM
- In email systems, R1, R2 are email servers



Also needed are namespaces and name lookup services.

Optional: discovery, aliases, redundancy, load balancing

Summary

- Remote Procedural Call (RPC)
- Remote Method Invocation (RMI)
- Sockets
- Message Oriented Middleware

A note on distributed computing Waldo et al (1994)

- Distributed computing is different from local computing
- The programmer should be aware of this
- The language should not hide it

A (false) vision of unified computing

- Object-oriented design conquers all
- Remote objects appear to be local
- Failures depend on the implementation, not on design

Hard problems in distributed computing

- Latency
  - *Remote calls are slower than local calls*
- Memory access
  - *no pointers => data must be copied*
- Partial failure
  - *network, system, process, state (and back)*
- Concurrency
  - *indeterminism, multiple calls*

Standard Interfaces

```
while (true) {
  try {
    context->remove(name);
    break;
  }
  catch (NotFoundInContext) {
    break;
  }
  catch (NetworkServerFailure) {
    continue;
  }
}
```

Section 5, page 9:
Client 1 tries to remove 'name'.
Client 1 fails, reconnects, tries again.
Due to partial failure, 'name' was actually removed.

Client 2 adds 'name'.

Client 1 finally succeeds, and removes 'name' a second time.

The ABA problem

Network File System (NFS)

- API adopted from local filesystem for compatibility
- Soft mounts expose errors to applications
- Hard mounts hide errors but hang applications
- The distributed model requires a centralized resource (the human administrator)

Interfaces and design

- Design remote interfaces to expect breakdowns
- Accept that the programmer must be informed about breakdowns
- E.g. `java.rmi.RemoteException`

Technical tidbits

- The ABA problem and tagged values
- Java wait() and spurious wakeups

Tagged values

- The ABA problem: The history and context of A has changed while a thread was blind

```
while (true) {
  oldValue = data.get()
  newValue = f(oldValue)
  if data.compareAndSet(oldValue, newValue)
    break
}
```

A bit-by-bit comparison of oldValue and *data*.value is not enough

Tagged values

- The ABA solution: tagged values

```
while (true) {
  [tag,oldValue] = data.get()
  newValue = f(oldValue)
  if data.compareAndSet([tag,oldValue], newValue)
    break
}
```

The tag counts the number of assignments to data.value.

data.value == [1000, -47]   compareAndSet([999, -47], 21) fails
compareAndSet([1000, -47], 21) succeeds
data.value == [1001, 21]

Tag sizes should be 48-64 bits to ensure long-term operation.

wait() and spurious wakeups

- java.lang.Object.
  - wait(long timeout)
  - wait(long timeout, int nanos)
  - wait()

```
synchronized (obj) {
    ...
    obj.wait();
    ...
}
```

Thread waits on:

- notify(), notifyAll()
- interrupt
- timer

Spurious wakeups can still happen

wait() and spurious wakeups

- java.lang.Object.wait(...) // all three versions

```
synchronized (obj) {
    ...
    while (<condition does not hold>)
        obj.wait();
    ...
}
```

Thread waits on:

- notify(), notifyAll()
- interrupt
- timer

Spurious wakeups can still happen

wait() and spurious wakeups

- java.lang.Object.wait(...) // all three versions

```
synchronized (obj) {
    ...
    while (<condition does not hold>)
        obj.wait();
    ...
}
```

On spurious wakeup:

- The thread has the monitor, but,
- No other thread has notified
- A timer may not have expired

Thread waits on:

- notify(), notifyAll()
- interrupt
- timer

Spurious wakeups can still happen

wait() and spurious wakeups

- java.lang.Object.wait()

```
synchronized (queue) {
    ...
    while (queue.isEmpty())
        obj.wait();
    ...
}
```

On spurious wakeup:

- The thread has the monitor, but,
- No other thread has notified
- A timer may not have expired

Thread waits on:

- notify(), notifyAll()

Spurious wakeups can still happen

wait() and spurious wakeups

- java.lang.Object.wait(long timeout)

```
synchronized (obj) {
    long t = System.currentTimeMillis();
    long elapsed = System.currentTimeMillis() - t;
    while (elapsed < delay) {
        obj.wait(delay - elapsed);
        elapsed = System.currentTimeMillis() - t;
    }
    ...
}
```

On spurious wakeup:

- The thread has the monitor, but,
- No other thread has notified
- A timer may not have expired

Thread waits on:

- timer

Spurious wakeups can still happen

End