Client-Server Computing

Dominic Duggan
Stevens Institute of Technology
Partly based on material by Ken Birman

1

COMMUNICATION MODELS

2

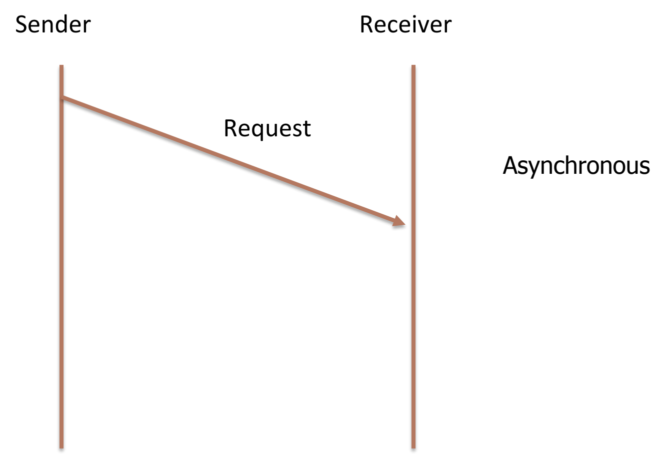
Communication Models

- Message-Passing
  - Send, receive, perhaps separate reply
  - We will assume message-passing
- Shared Memory
  - Reading/writing shared global variables
  - Distributed shared memory
    - Barrier synchronization

Blocking vs Non-blocking Message-Passing

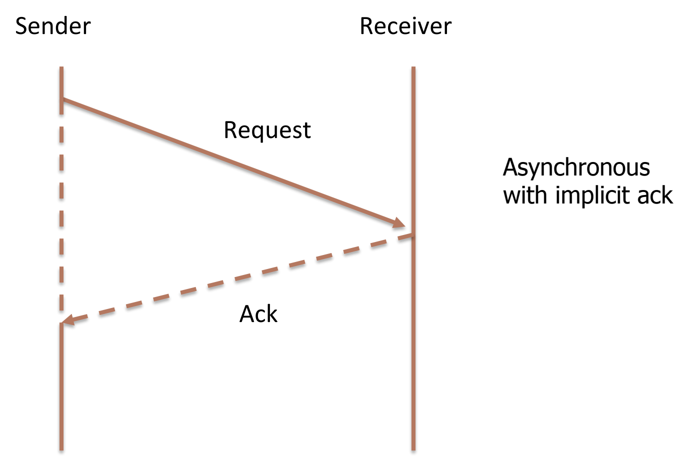
- Synchronous/Blocking
  - Ex: RPC
  - Sender waits for ack
  - Pro: confirmation of receipt, flow control
  - Con: latency (especially over WAN)
- Asychronous/Nonblocking
  - Ex: UDP sockets
  - Sender does not wait for ack
    - Don't care if message received
    - Synchronize later with **promise/future**
    - More complicated programming

Message-passing



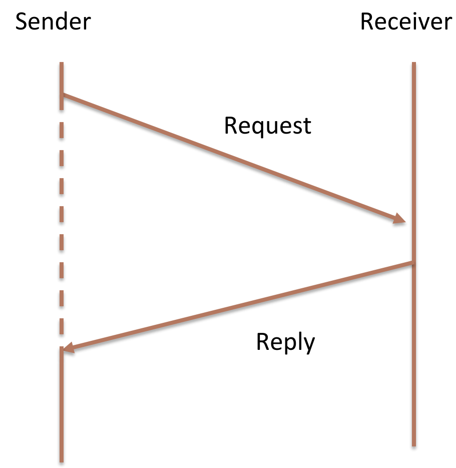
5

Message-passing



6

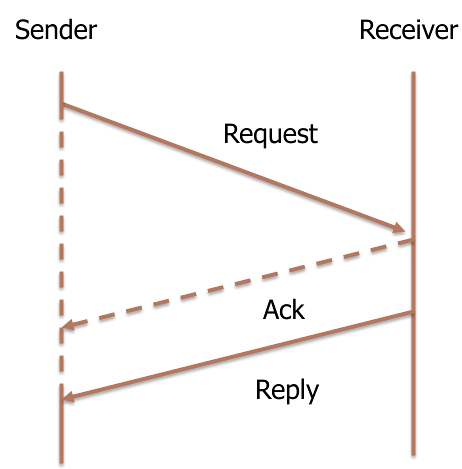
Message-passing



Synchronous with explicit reply

7

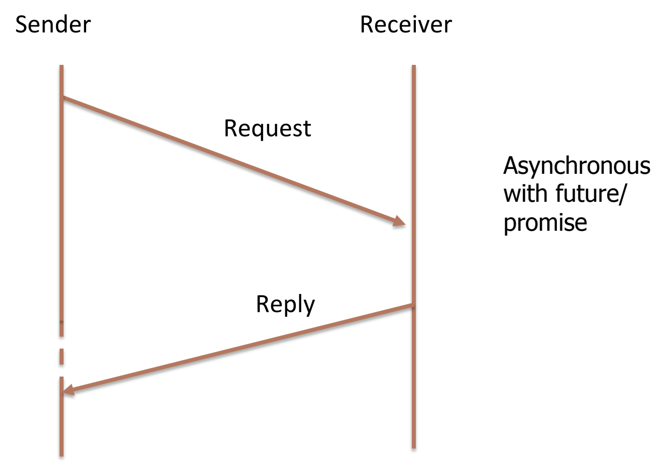
Message-passing



Synchronous with ack & reply

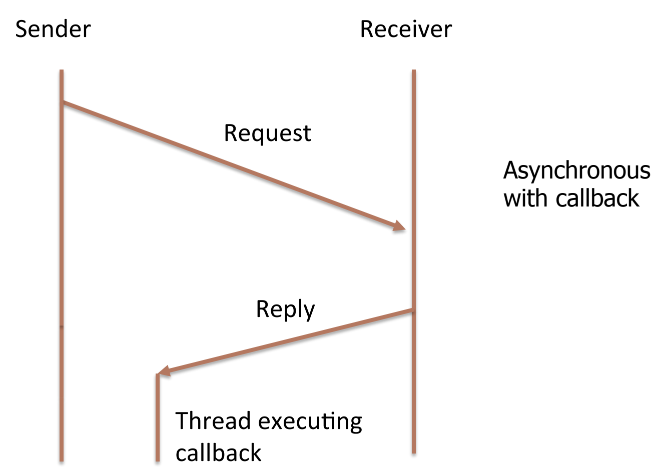
8

Message-passing



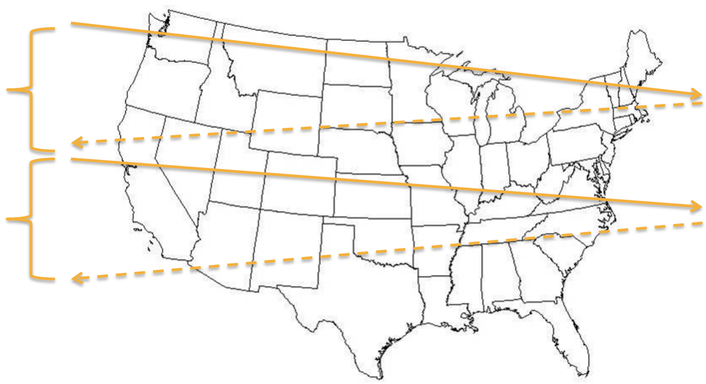
9

Message-passing



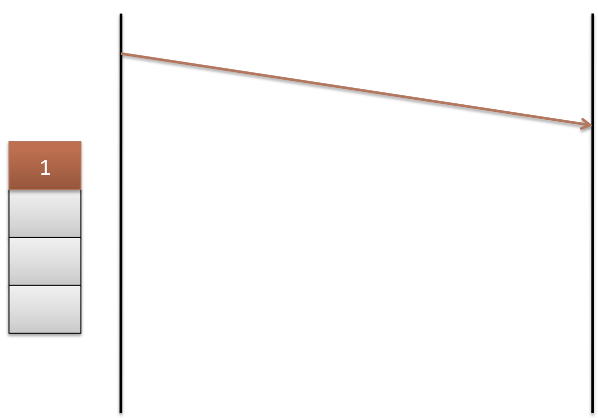
10

Latency



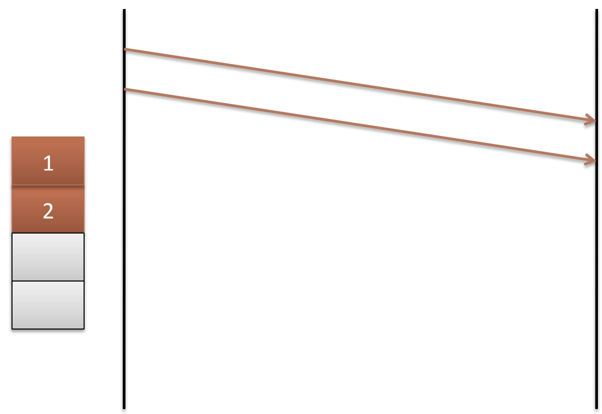
11

Pipelining



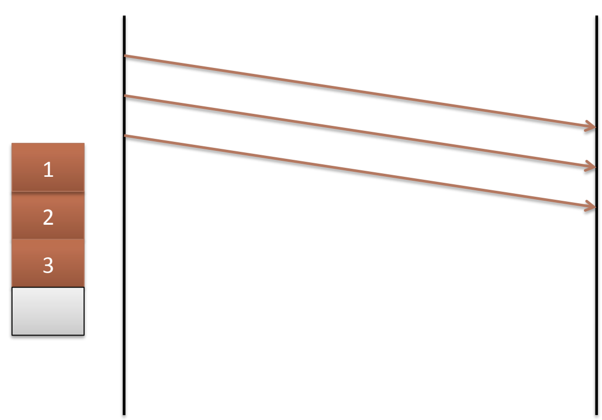
12

Pipelining



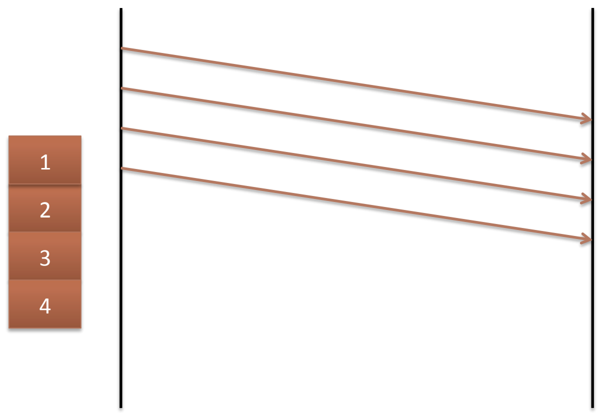
13

Pipelining



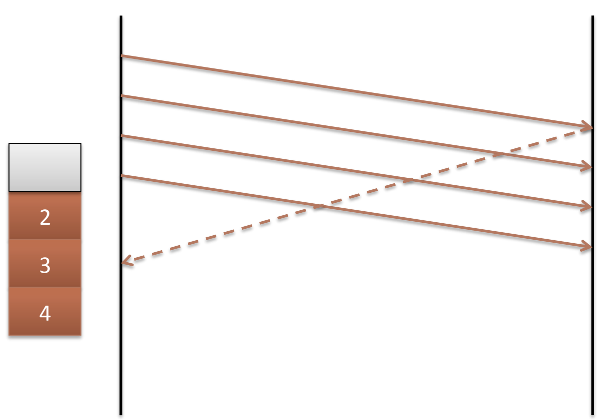
14

Pipelining



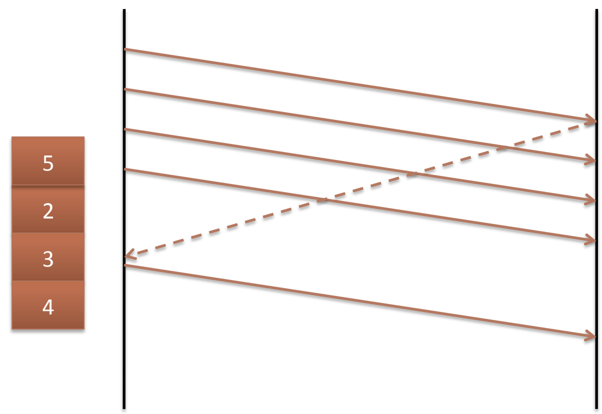
15

Pipelining



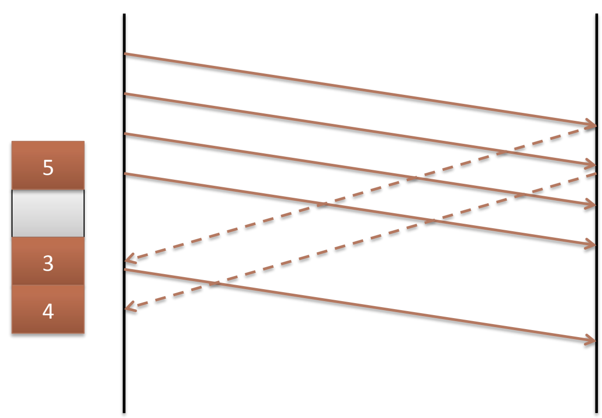
16

Pipelining



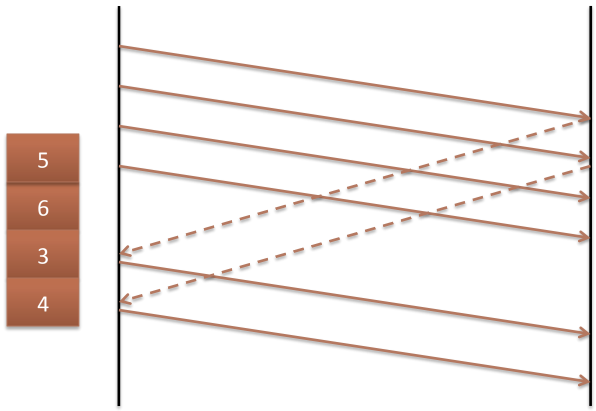
17

Pipelining

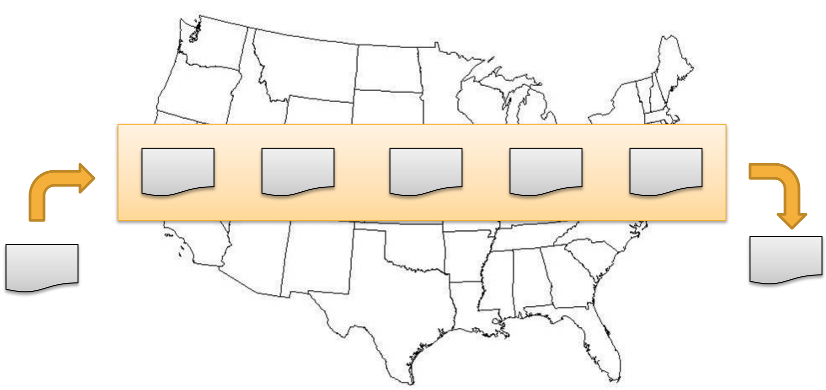


18

Pipelining



Pipelining



SOCKET-BASED COMMUNICATION

21

Classic view of network API

- Start with host name (maybe)

foo.bar.com

22

Classic view of network API

- Start with host name
- Get an IP address

23

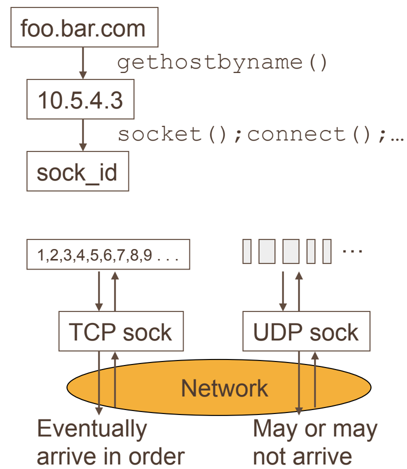
Classic view of network API

- Start with host name
- Get an IP address
- Make a socket (protocol, address)

24

Classic view of network API

- Start with host name
- Get an IP address
- Make a socket (protocol, address)
- Send byte stream (TCP) or packets (UDP)



25

Sockets

- Berkeley Unix 4.2BSD
- API for Internet transport protocols
- The foundation of the Internet
- Two main kinds:
  – Datagram sockets (UDP/IP)
  – Stream sockets (TCP/IP)
  – Also raw sockets (IP, ICMP)

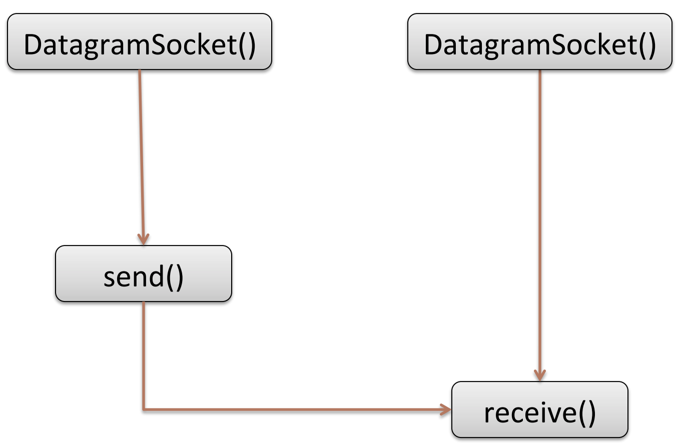
26

Datagram Sockets

- Unreliable
  - In practice, reliable in LANs
- Low overhead
- Applications
  - Queries
  - Notifications
  - Small messages

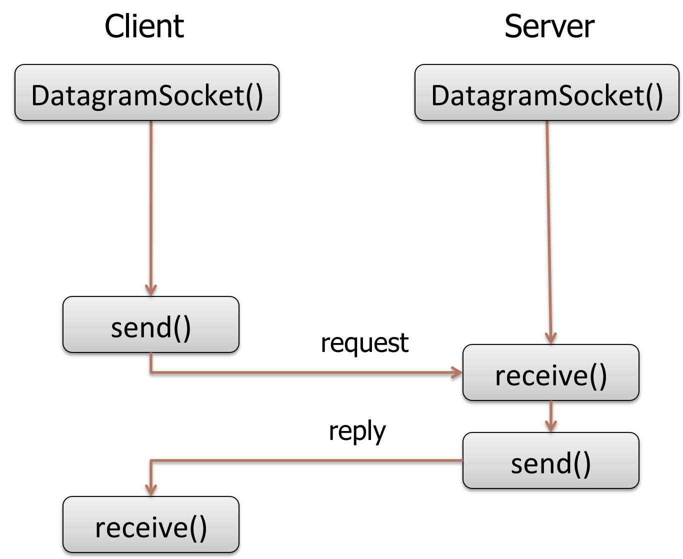
27

Simple Message Passing (Java)



28

Client-Server (Java)



29

Network Addresses in Java

```
public final class InetAddress implements Serializable {
  // host may be DNS address
  static InetAddress getByName (String host);
  static InetAddress getLocalHost ();

  byte[] getAddress ();
  String getHostName ();
  String getHostAddress ();
  boolean isMulticastAddress ();
}
```

30

Datagram Packets in Java

```
public class DatagramPacket {
  public DatagramPacket (byte ibuf[], int ilen);
  public DatagramPacket (byte ibuf[], int ilen,
                         InetAddress dest, int port);

  synchronized InetAddress getAddress ();
  synchronized int getPort ();
  synchronized byte[] getData ();
  synchronized int getLength ();
}
```

31

Datagram Sockets in Java

```
Public class DatagramSocket {
  public DatagramSocket ();
  public DatagramSocket (int port);

  void send (DatagramPacket p);
  void receive (DatagramPacket p);
  void close ();

  synchronized void setSoTimeout (int timeout);
}
```

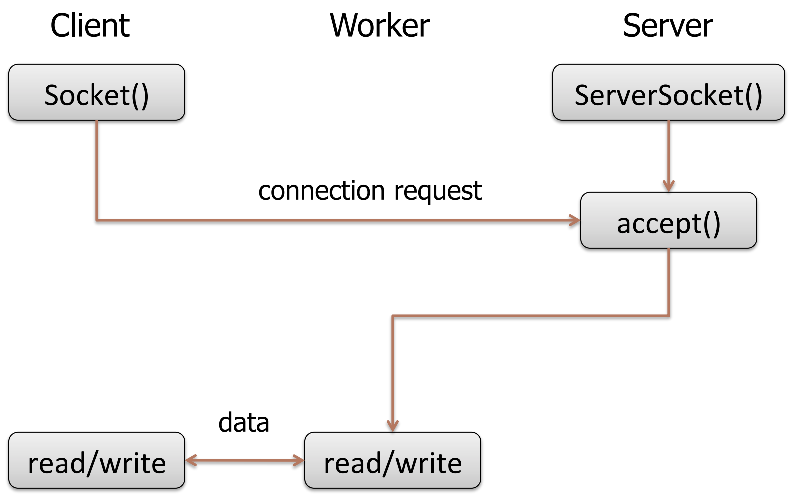
32

STREAM SOCKETS

Stream Sockets

- Based on TCP/IP protocol
- Client makes connection request
- Server listens for connection requests
  - stream socket returned to client and server
- Stream socket looks/smells/feels like a file
  - Reliable, ordered byte stream
- New server thread

Stream Sockets (Java)



Client Stream Sockets in Java

```
public class Socket {
  public Socket (InetAddress address, int port);
  InputStream getInputStream ();
  OutputStream getOutputStream ();
  synchronized void close();

  void setTcpNoDelay (boolean on);
  void setSoLinger (boolean on, int val);
  void setSoTimeout (int timeout);
  static setSocketImplFactory (SocketImplFactory fac);
```

Server Stream Sockets in Java

```
public class ServerSocket {
  public ServerSocket (int port);
  public ServerSocket (int port, int backlog);
  void Socket accept ();
  void close ();

  void setToTimeout (int timeout);
  protected final void implAccept (Socket s);
  static setSocketImplFactory (SocketImplFactory fac);
}
```

37

Extended Server Sockets

```
class SSLServerSocket extends ServerSocket { ...
  public Socket accept () {
      SSLSocket s = new SSLSocket (certChain, privateKey);
      // create an unconnected client SSLSocket, that we'll
      // return from accept
      implAccept (s);
      s.handshake ();
      return s; }
}

class SSLSocket extends Socket { ...
  public SSLSocket(CertChain c, PrivateKey k) {
      super(); ...
  }
}
```

38

FTP EXAMPLE

39

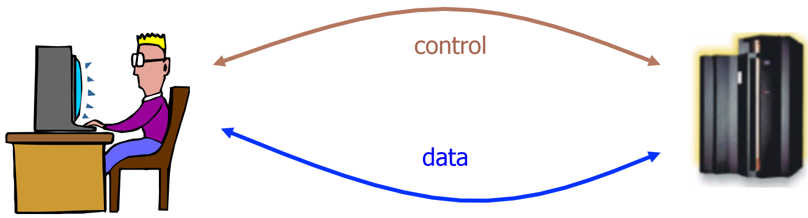
FTP: File Transfer Protocol

- Client connected to ftp server (ftpd)
  - List directory contents
  - Change directory
  - Get file contents
  - Put file contents
- Stateful protocol
  - Server maintains client state

40

FTP Data Transfer

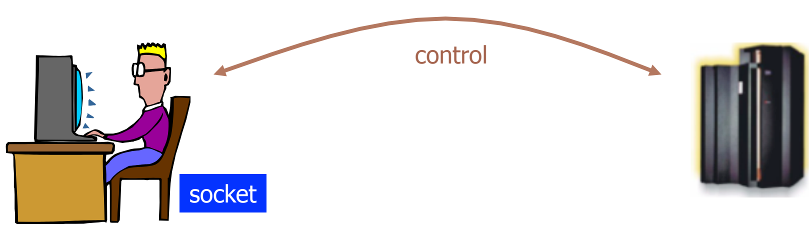
- Control connection for commands
- Use separate data connection to:
  - Send lists of files (LIST)
  - Retrieve a file (RETR)
  - Upload a file (STOR)



41

Creating the data connection: active mode

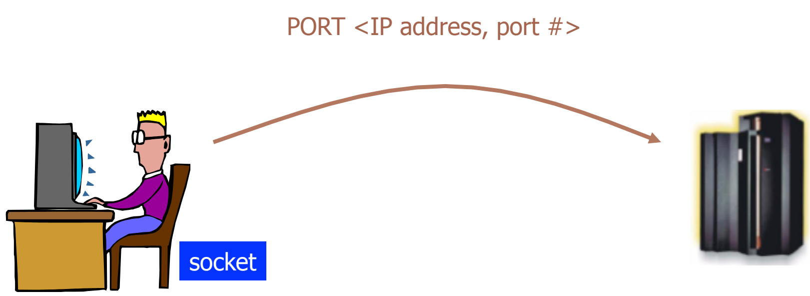
- Client acts like a server
  - Creates a socket
    - Assigned an ephemeral port number by the OS
  - Listens on socket
  - Waits to hear from FTP server



42

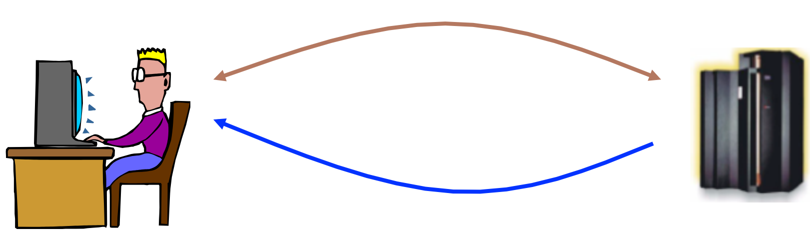
Creating the data connection: active mode

- Client tells port number to the server
  - Via PORT command on control connection



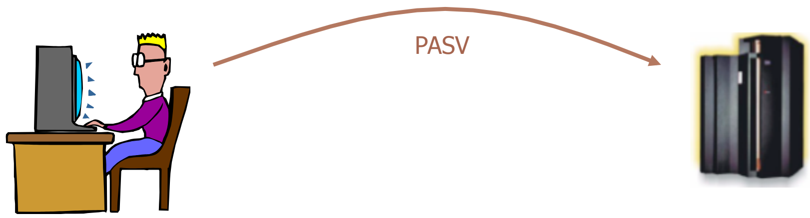
Creating the data connection: active mode

- Server initiates the data connection
  - Connects to the socket on the client machine
  - Client accepts, to complete the connection
- Data now flows along second connection



Creating the data connection: passive mode

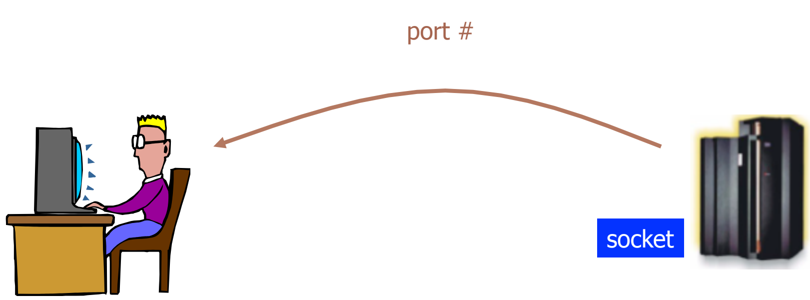
- Client tells the server to go into passive mode



45

Creating the data connection: passive mode

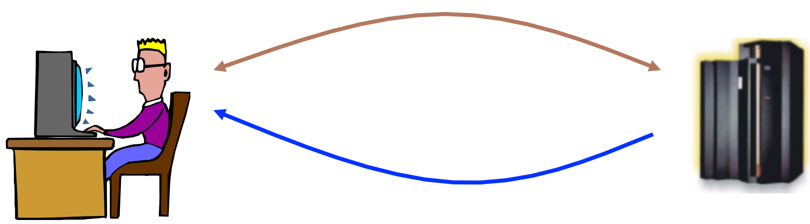
- Server creates socket, responds with port number



46

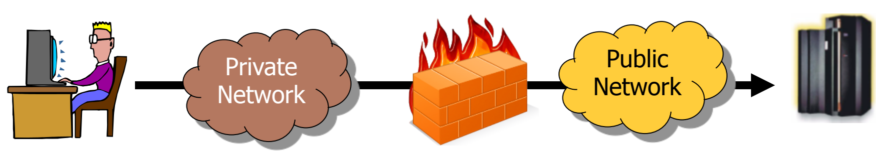
Creating the data connection: passive mode

- Client initiates the data connection
  - Connects to the socket on the server machine
  - Server accepts, to complete the connection
- Data now flows along second connection

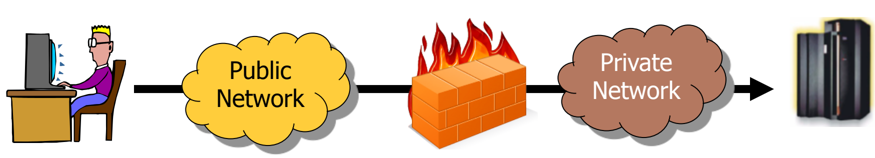


47

Active vs Passive Mode



PASV reply returns a public IP address



PORT command includes a public IP address

48

Server File Transfer

```
ServerSocket listenTo = new ServerSocket (0, 1);
// 0 means any port
…. send listenTo.getLocalPort() to client….

Socket xfer = listenTo.accept ();
InputStream is = new FileOutputStream (file);
   OutputStream os = xfer.getOutputStream ();
   byte[] buf = new [512] byte ();
   int nbytes = is.read (buf, 0, 512);
   while (nbytes > 0) {
      os.write (buf, 0, nbytes);
      nbytes = is.read (buf, 0, 512);
   }
   is.close();  os.close();
```

49

Client File Transfer

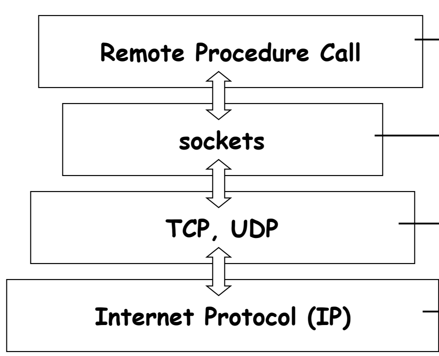
```
   String file;
   int setupPort;

   Socket xfer = new Socket (this.server, setupPort);
   InputStream is = xfer.getInputStream ();
   OutputStream os = new FileOutputStream (file);
   byte[] buf = new [512] byte ();
   int nbytes = is.read (buf, 0, 512);
   while (nbytes > 0) {
      os.write (buf, 0, nbytes);
      nbytes = is.read (buf, 0, 512);
   }
   is.close();  os.close();
 }
```

50

REMOTE PROCEDURE CALL

RPC as a Programming Abstraction



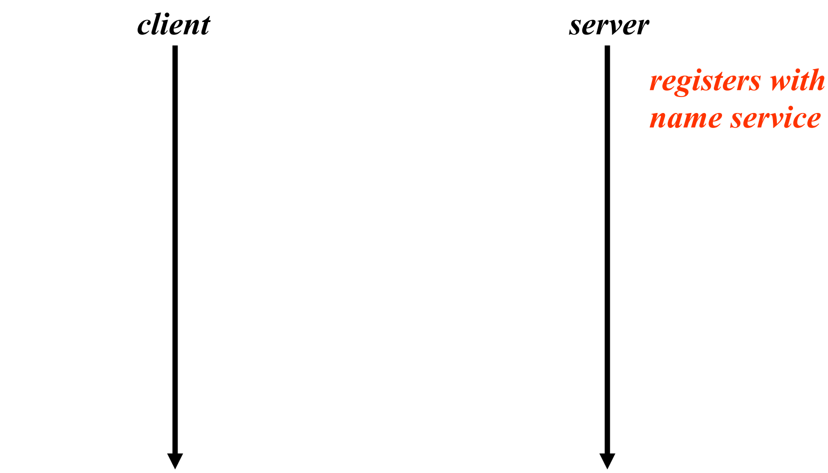
Remote Procedure Call:
hides communication details behind a procedure call and helps bridge heterogeneous platforms

sockets:
operating system level interface to the underlying communication protocols

TCP, UDP:
User Datagram Protocol (UDP) transports data packets without guarantees
Transmission Control Protocol (TCP) verifies correct delivery of data streams

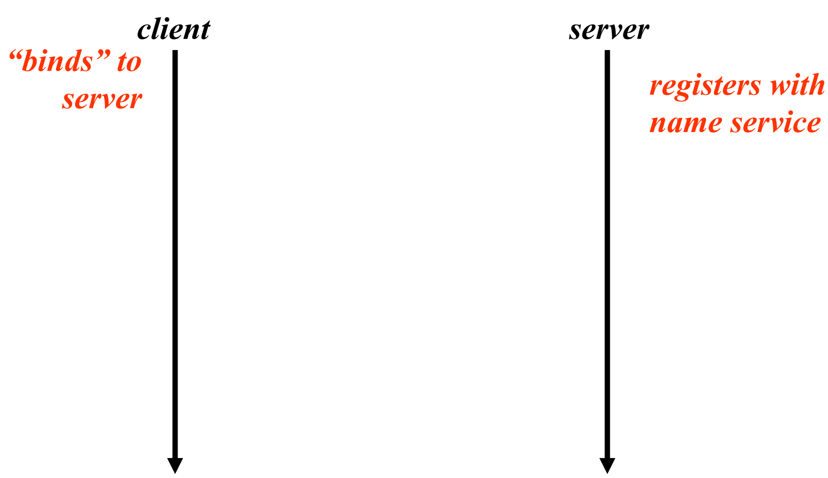
Internet Protocol (IP):
moves a packet of data from one node to another

The basic RPC protocol



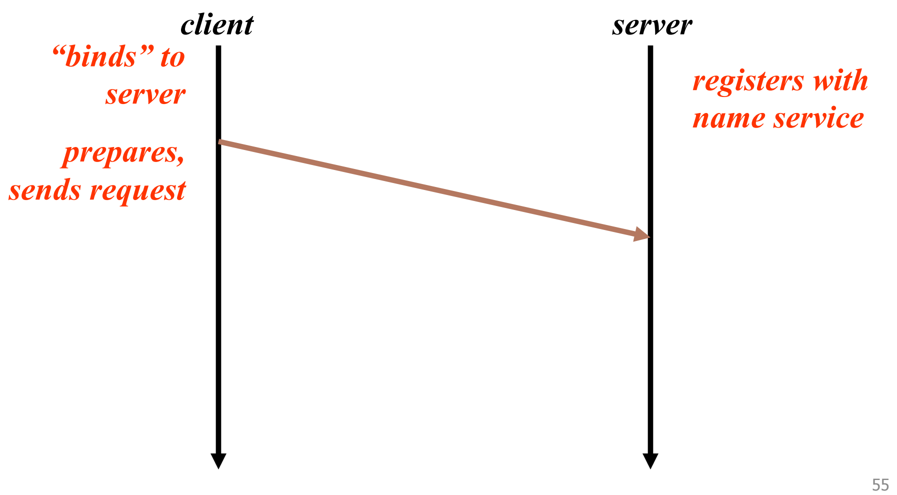
53

The basic RPC protocol

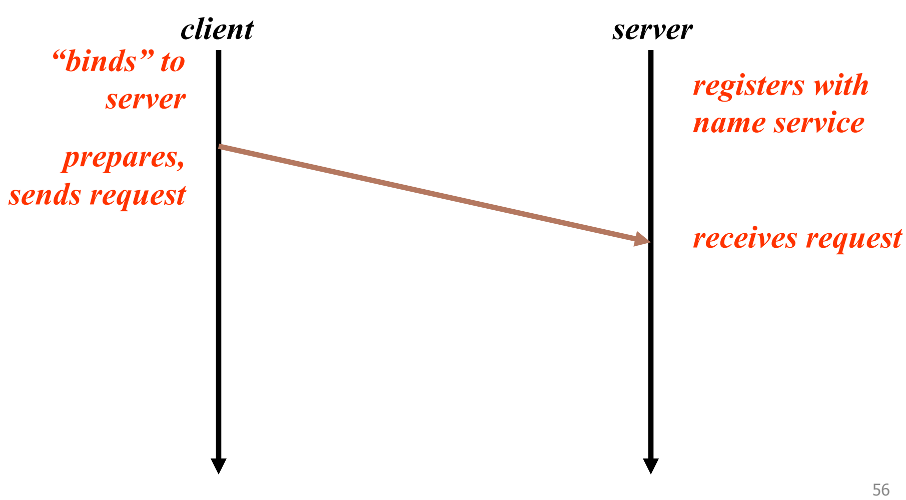


54

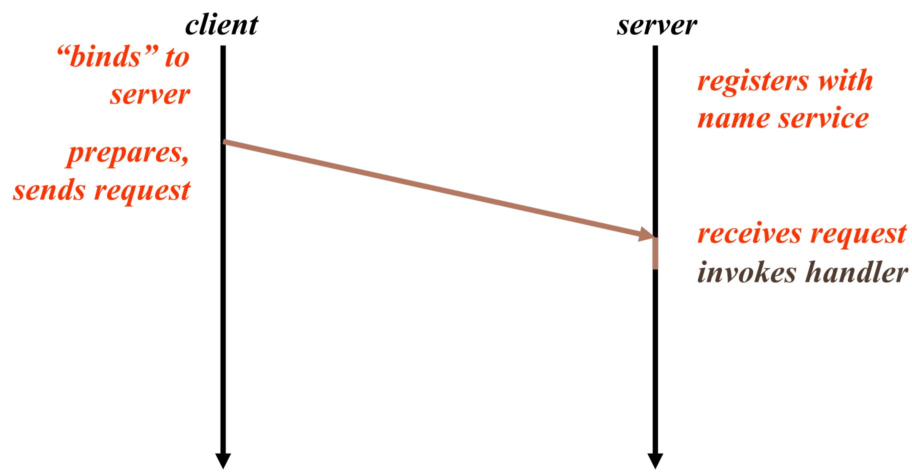
The basic RPC protocol



The basic RPC protocol

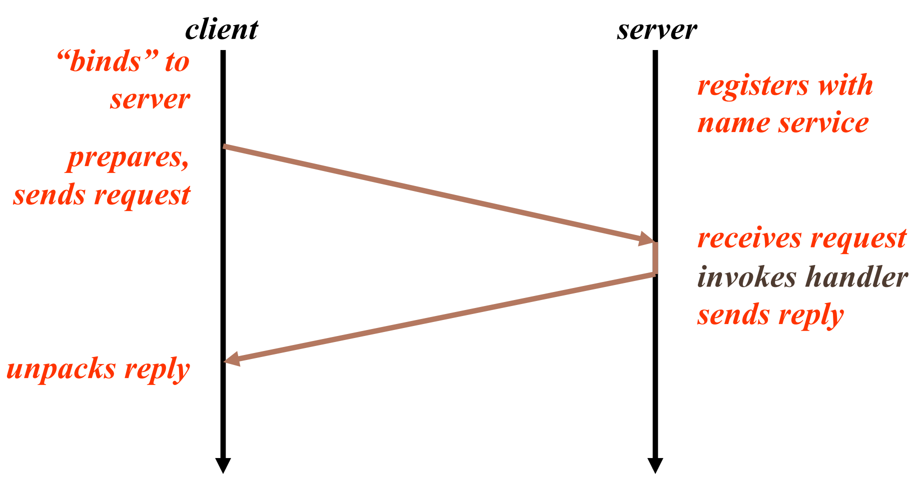


The basic RPC protocol

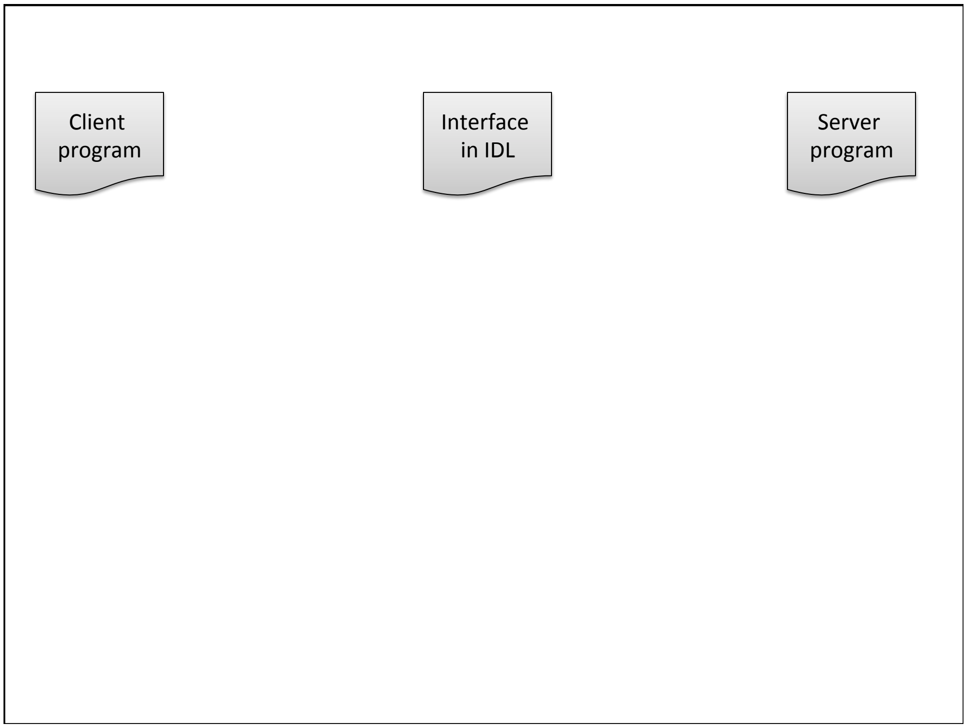


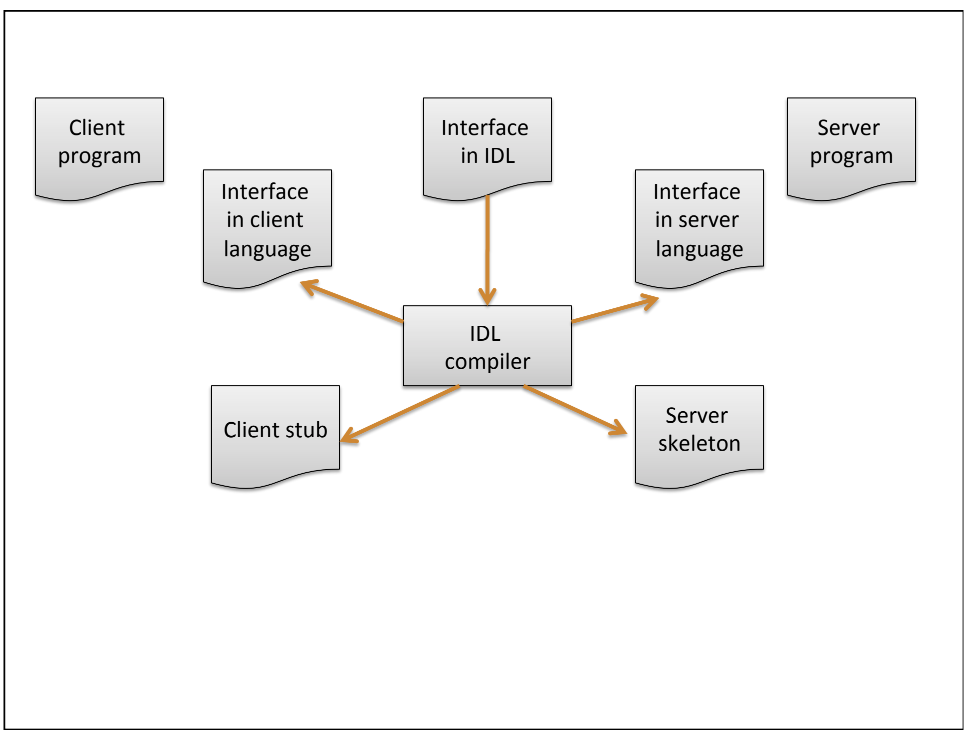
57

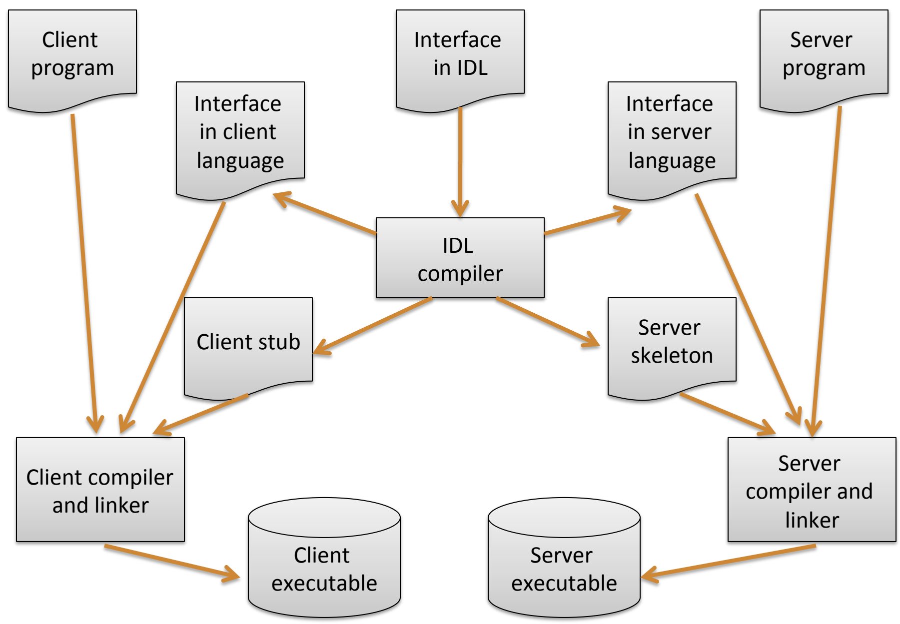
The basic RPC protocol



58


Client program
Interface in IDL
Server program

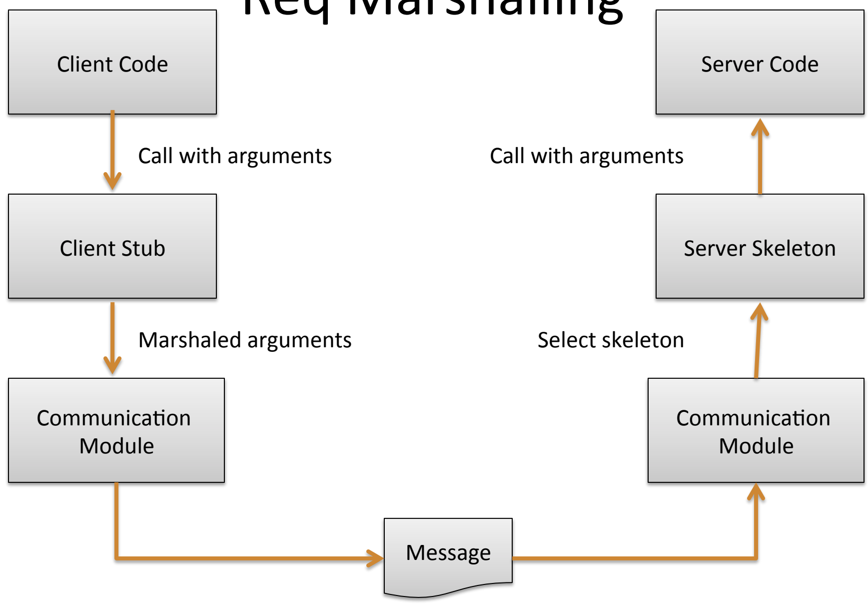

Client program
Interface in IDL
Server program
Interface in client language
Interface in server language
IDL compiler
Client stub
Server skeleton


Client program
Interface in IDL
Server program
Interface in client language
Interface in server language
IDL compiler
Client stub
Server skeleton
Client compiler and linker
Server compiler and linker
Client executable
Server executable

RPC Binding


Client Process
(2) bind
(3) handle
Naming and Directory Service
(1) register
Server Process

Req Marshalling



Data in messages

- Marshalling
- Byte ordering issues (big-endian versus little endian), strings (some CPUs require padding), alignment, etc
- As fast as possible
  – yet must also be very general

64

Fancy argument passing

- RPC transparent for simple calls
  - New forms of exceptions
- Complex structures, pointers, big arrays?
- Most limit size, types of RPC arguments.

RPC WITH LOST MESSAGES

RPC Semantics in the Presence of Failures

- The client is unable to locate the server
- The request message from the client to server is lost
- The reply message from the client is lost
- The server crashes after sending a request
- The client crashes after sending a request

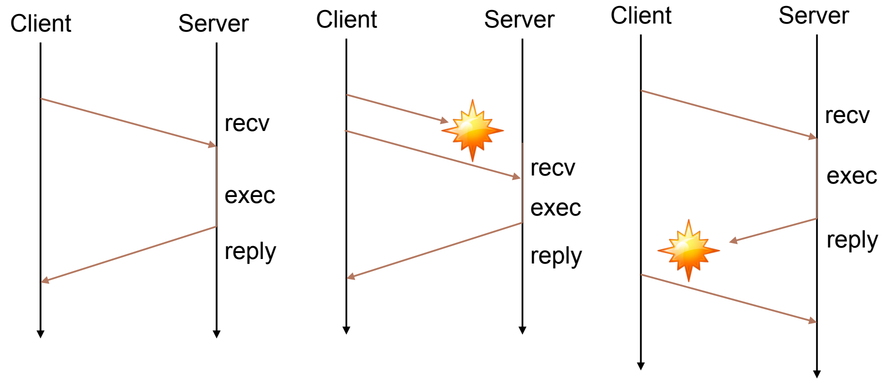
67

Client is Unable to Locate Server

- Causes:
  - – server down
  - – server moved
  - – different version of server, …
- Fixes
  - – Network failure exceptions
    - Transparency is lost

68

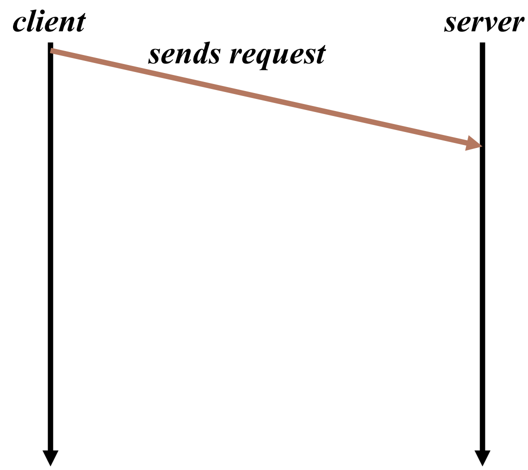
Messages Lost



Lost Request Message

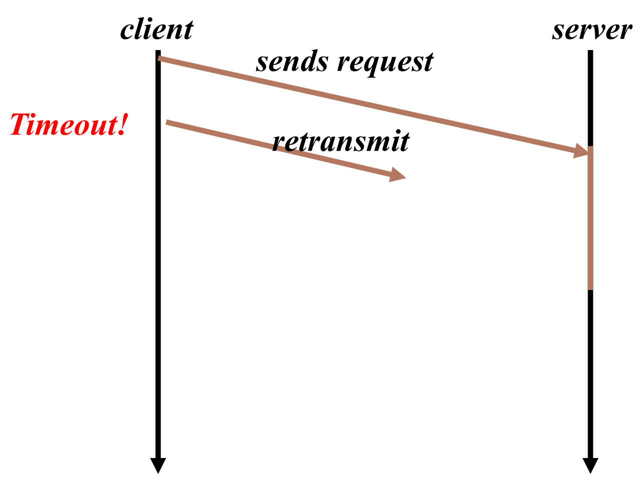
- Easiest to deal with
- Just retransmit the message!
- If multiple message are lost then
  - "client is unable to locate server" error

Overcoming lost packets



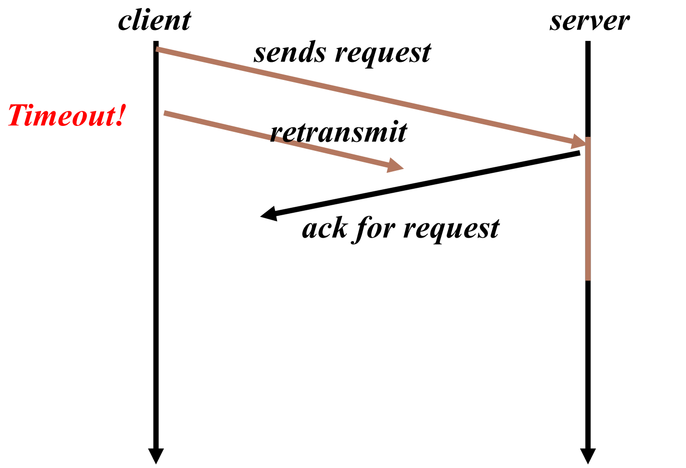
71

Overcoming lost packets

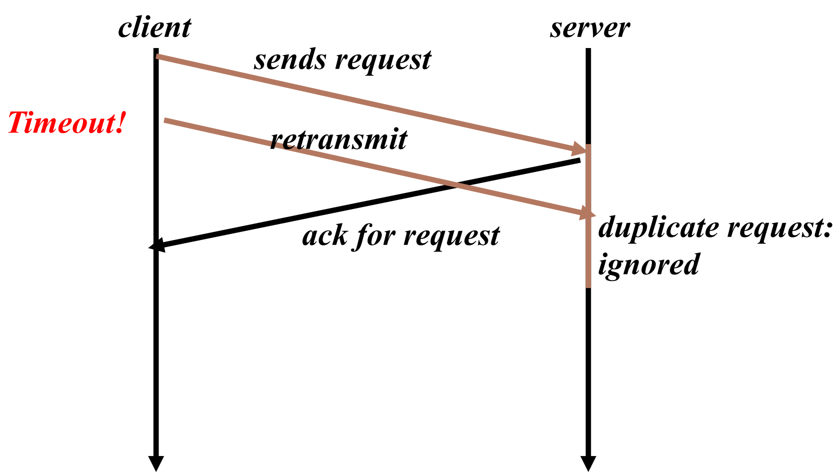


72

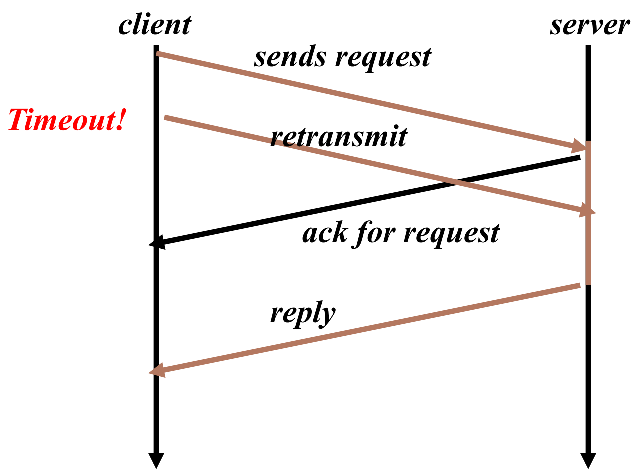
Overcoming lost packets



Overcoming lost packets

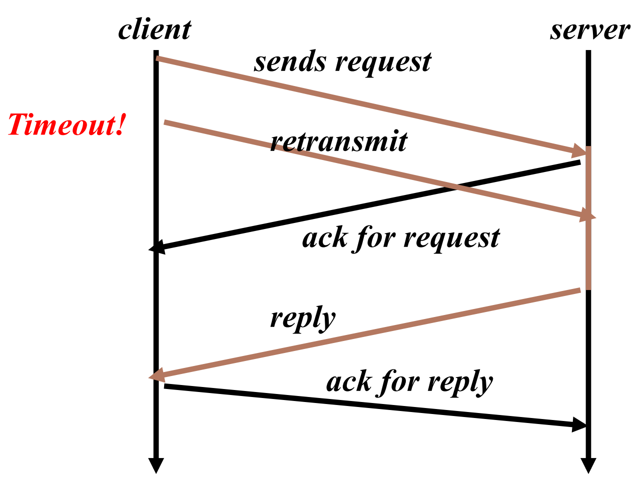


Overcoming lost packets



75

Overcoming lost packets



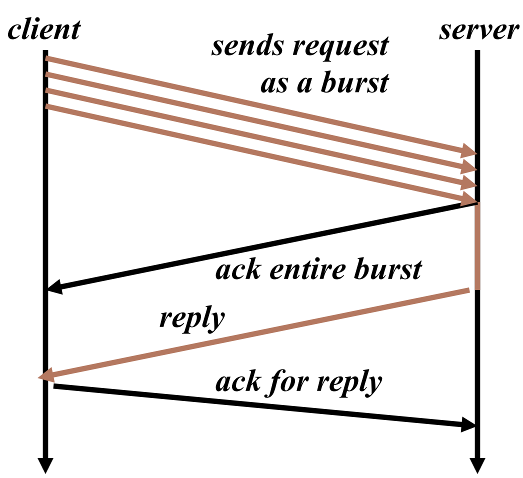
76

Costs in fault-tolerant version?

- Acks are expensive.
  - Suppress the initial ack
- Retransmission is costly.
  - Tune the delay
- Big messages
  - ack a burst at a time

77

Big packets



78

Lost Reply Message

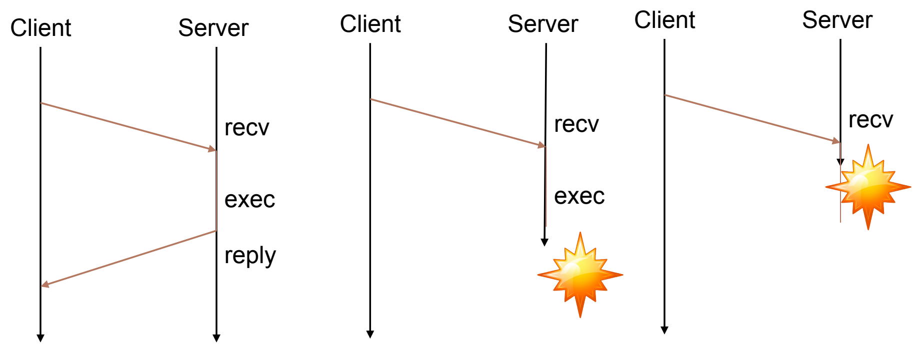
- Did server execute the procedure or not?
- Possible fixes
  - Retransmit the request
    - Only works if operation is idempotent
  - What if operation not idempotent?
    - Assign unique sequence numbers to every request
    - **Shared state between client and server…**

79

RPC WITH CRASHES

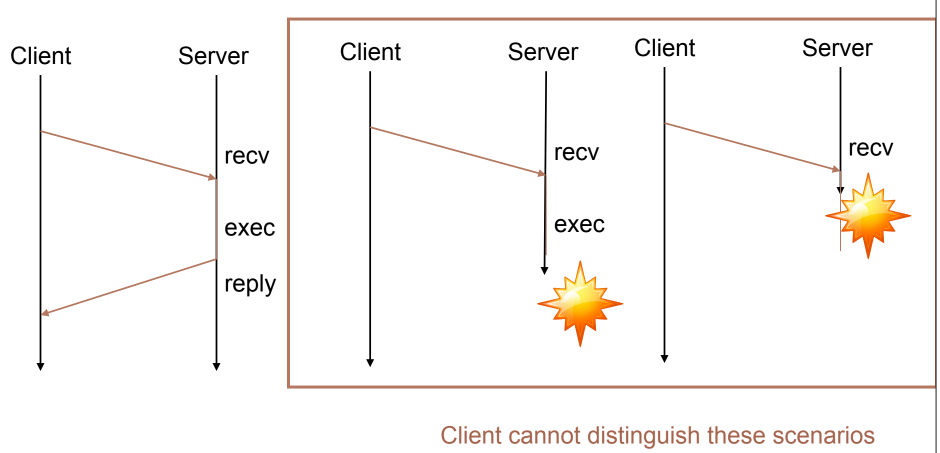
80

Server Crashes



81

Server Crashes



82

Server Crashes

- Three possible semantics
  - At least once semantics
    - Client keeps trying until it gets a reply
  - At most once semantics
    - Client gives up on failure
  - Exactly once semantics
    - Can this be correctly implemented?

83

Impossibility of Exactly Once Semantics (1)

- Example: Print server
  - Client wants to print a document on the server
  - Three possible events at server:
    - Reply with completion message (R)
    - Print the text (P)
    - Crash (C)

84

Impossibility of Exactly Once Semantics (2)

- These events (R, P, C) can occur in six different orderings:

Reply	Reply	**Crash**	Print	Print	**Crash**
Print	**Crash**	(Reply)	Reply	**Crash**	(Print)
Crash	(Print)	(Print)	**Crash**	(Reply)	(Reply)

Impossibility of Exactly Once Semantics (3)

- Assume server crashes, later recovers, and notifies clients
- What should clients do?

Client reissue strategy	Server (strategy R → P)		
	RPC	RC(P)	C(RP)
Always	DUP	OK	OK
Never	OK	ZERO	ZERO
Only when ACK	DUP	OK	ZERO
Only when not ACK	OK	ZERO	OK

Impossibility of Exactly Once Semantics (4)

- Assume server crashes, later recovers, and notifies clients
- What should clients do?

Client reissue strategy	Server (strategy P → R)		
	PRC	PC(R)	C(PR)
Always	DUP	DUP	OK
Never	OK	OK	ZERO
Only when ACK	DUP	OK	ZERO
Only when not ACK	OK	DUP	OK

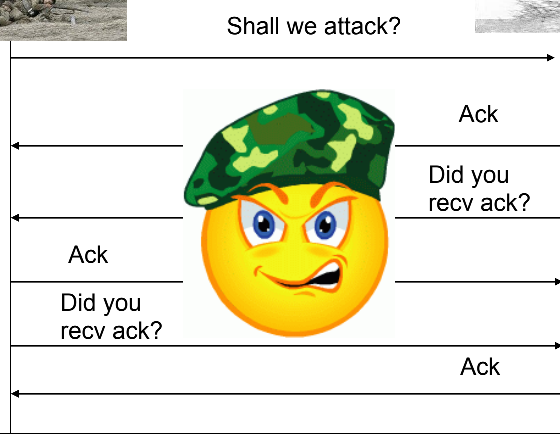
87

Impossibility of Exactly Once Semantics (5)

- Can we fix this?
- Fundamental problem: distributed agreement
- Analogy: Coordinating attacks of allied armies
  - Allied armies on opposing hillsides
  - Messengers travel through valley occupied by enemy
  - Omission failures (not Byzantine!)

88

Impossibility of Exactly Once Semantics (6)



89

Client Crashes

- Let's the server computation **orphan**
- Orphans can
  - Waste CPU cycles
  - Lock files
  - Client reboots and it gets the old reply immediately

90

Client Crashes: Possible Solutions

- Extermination:
  - Client log, kill orphan on reboot
  - Disadvantage: log overhead
- Reincarnation:
  - Client broadcasts new "epoch" when reboots
  - New epoch: servers kill orphans
  - Disadvantage: network partition

91

Client Crashes: Possible Solutions

- Expiration:
  - Each RPC is given a lease T
  - Must renew lease before expiration
  - If client reboots after T sec, all orphans are gone
  - Problem: what is a good value of T?

92

FALLACIES IN DISTRIBUTED COMPUTING

93

The network is reliable

- Servers may experience software failures
- Routers may drop packets due to buffer exhaustion
- Infrastructure: need hardware and software redundancy
- Software: deal with message loss
  - WS-ReliableMessaging: not persistent through node failures

94

Latency is zero

- Okay on a LAN, not on a WAN
- More problematic than bandwidth
  - 30ms to send ping from Europe to US and back
- Minimize packet size
  - Want other side to start processing data quickly
  - Pipeline parallelism, need a LARGE window
- Can kill an AJAX application (backend latency)

95

Bandwidth is infinite

- VOIP, video, IPTV pushing demands
- Packet loss limits bandwidth
  - Ex: NY/LA rtt = 40ms, say packet loss is 0.1%
  - Suppose MTU = 1500, throughput ≤ 6.5 Mbps
  - Suppose MTU = 9000, throughput ≤ 40 Mbps
  - Constraint is time to recover from packet loss
- Conclusion: maximize packet size!

96

The Network is Secure

- Many defenses stop at the perimeter (firewall)
- Defense in depth:
  - Enclaves within enterprise network
  - Services should always validate inputs
  - Secure internal communications
    - Network level: IPsec virtual networks
    - Transport level: SSL/TLS

97

Topology doesn't change

- Authentication & authorization:
  - Single sign-on, authorization server infrastructure
- Naming: Reference services by DNS name rather than IP address
  - But what if DNS name changes?
- Routing:
  - IP makes routing decisions on per-hop basis
  - WS-Addressing adopts same idea
  - Message broker/enterprise service bus
    - Publish-subscribe semantics

98

There is one administrator

- Within an enterprise:
  - Database, web server, network, Linux, Windows, mainframe administrators
- Across enterprise boundaries:
  - Example: The cloud!
  - Something breaks, and you need to work with outside administrators to diagnose and fix
  - Upgrades and version compatibility

99

Transport cost is zero

- Overhead of communication stack
  - Time to marshal/unmarshall
- Any network deployment requires cost-benefit analysis
  - Benefits of infrastructure
  - Costs of purchase, running, maintenance

100

The network is homogeneous

- The motivation for CORBA
  - …and then SOAP-based Web services
  - …but Java EE and WCF Web services are not interoperable…
- Heterogeneity is inevitable with IT economics
  - Which technology would you like to be locked into today?

101

Broad comments on RPC

- RPC is not very transparent
- Failure handling not evident
  - What to do with timeout?
- Performance work:
  - from 75ms RPC to RPC over InfiniBand with 75usec round-trip (later)

102

JAVA RMI

103

Java Remote Method Invocation (RMI)

- Distributed garbage collection
- Local vs remote objects
  - Local objects passed *by value*
- Dynamic stub downloading
- Remote exceptions

104

Defining Remote Interfaces

```
import java.rmi.*;
public interface IServer extends Remote {
   public void cd (String filename)
         throws RemoteException;
   public String[] dir () throws RemoteException;
}
```

105

Defining a Remote Implementation

```
import java.rmi.*;
public class Server extends UnicastRemoteObject
                    implements IServer {
   private Stack<String> cwd;
   private String path() { ... }
   public cd (String dir) throws RemoteException {
      cwd.push(dir);
   }
   public String[] dir () throws RemoteException {
      return new File(path()).list();
   }
```

106

RMI Registry

Flat non-persistent namespace

```
interface Registry extends Remote {
    Remote lookup (String name);
    void bind (String name, Remote obj);
}
class LocateRegistry extends Object {
   static Registry getRegistry (String host, int port);
   static Registry createRegistry (int port);
}
```

107

Factory Objects

```
import java.rmi.*;
interface IServerFactory extends Remote {
  public IServer createServer () throws RemoteException;
}

class ServerFactory extends UnicastRemoteObject
                    implements IServerFactory {
   public IServer createServer () throws RemoteException {
      return new  Server (...);
   }
}
```

108

Example Server

```
import java.rmi.*;
class Server {
   public static void main (String[] args) {
      System.setSecurityManager
                  (new RMISecurityManager());
      ServerFactory stub = new ServerFactory (...);
      Registry registry =
         LocateRegistry.createRegistry(serverPort);
      registry.rebind(serverName, stub);
   }
}
```


Example Client

```
import java.rmi.*;
class Client {
   public static void main (String[] args) {
      System.setSecurityManager
         (new RMISecurityManager ());
      IServerFactory factory;
      // Look up factory in registry
      IServer server = factory.createServer();
   }
}
```

Putting it all together

- Compile the client and server

```
javac Client Server
```

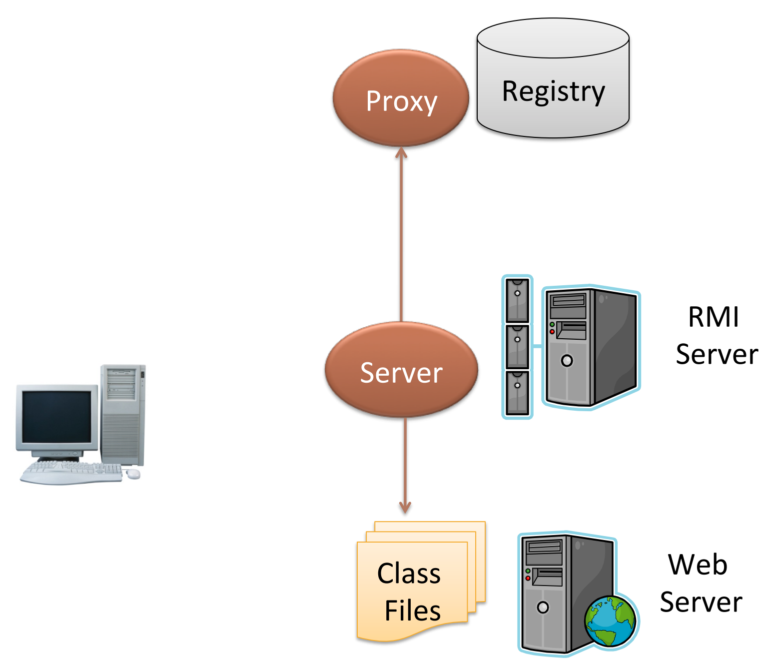
- Run the registry (on guinness)

```
rmiregistry 5000 &
```

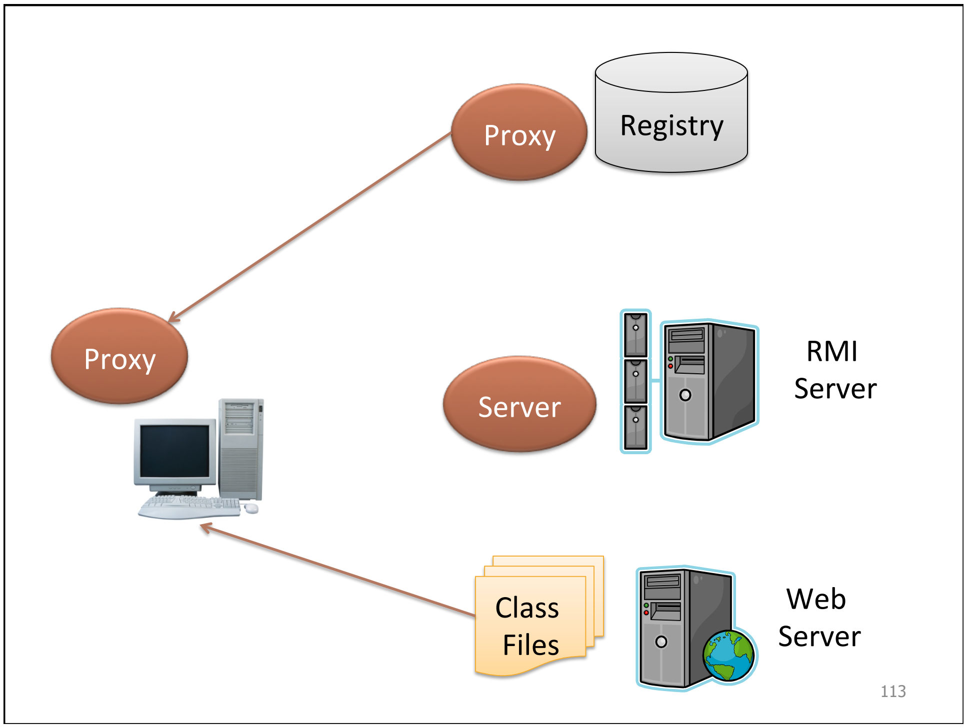
- Run server on server host, client anywhere

```
java Server &
java Client
```

111



112



Dynamic Stub Loading

- URL for missing class codebase specified:
  - Dynamically in object reference (URL)
  - Statically in `java.rmi.server:codebase` property
- Disable downloading by setting
  - `java.rmi.server:useCodebaseOnly` to true
- `RMIClassLoader` loads stubs across network
- Security manager must be installed

114

LOCAL RPC

115

RPC versus local procedure call

- Restrictions on argument sizes and types
- New error cases:
  - Bind operation failed
  - Request timed out
  - Argument "too large"
- Costs may be very high

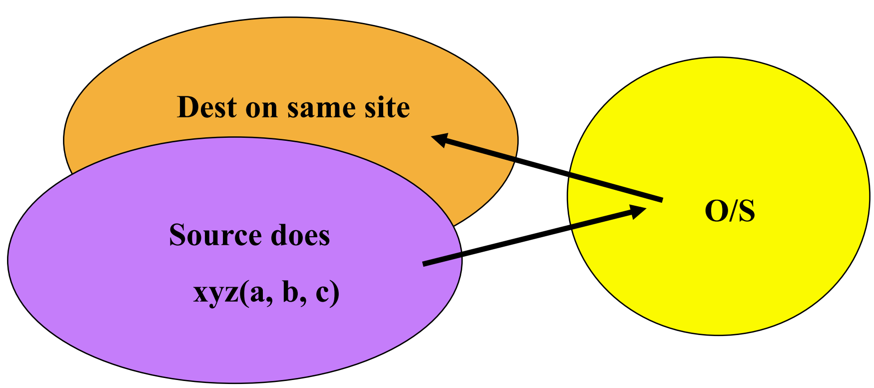
116

RPC costs in case of local destination process

- Often, the destination is right on the caller's machine
  - Caller builds message
  - Send system call, then receive, blocks, context switch
  - Message copied into kernel, then out to dest.
  - Context switch
  - Dest computes result
  - Repeated in reverse direction
  - If scheduler is a process, may context switch 4 times!

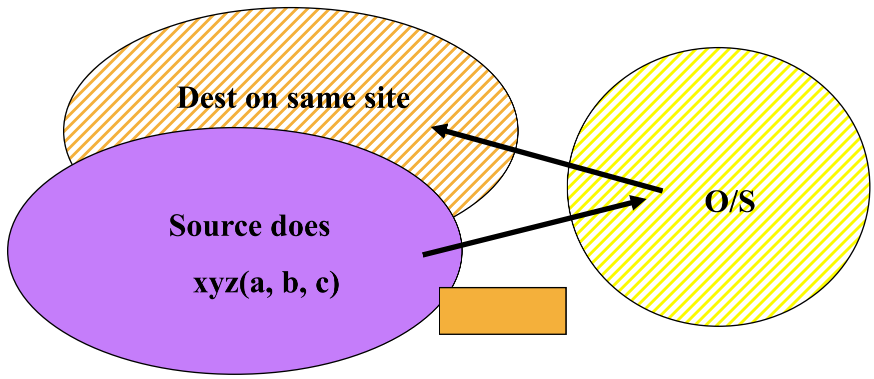
117

RPC example



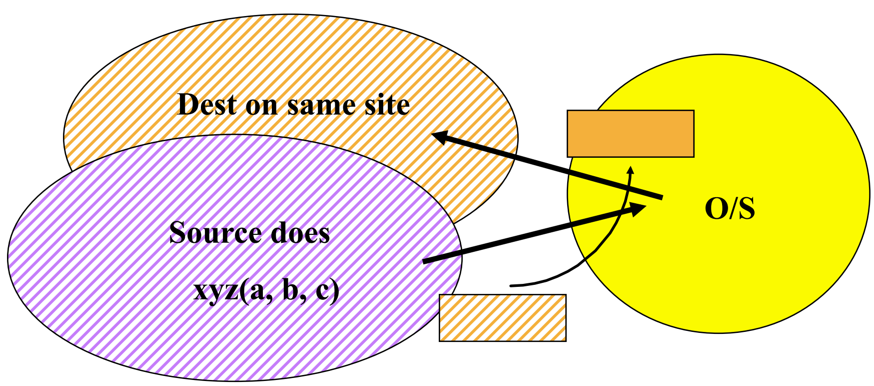
118

RPC in normal case



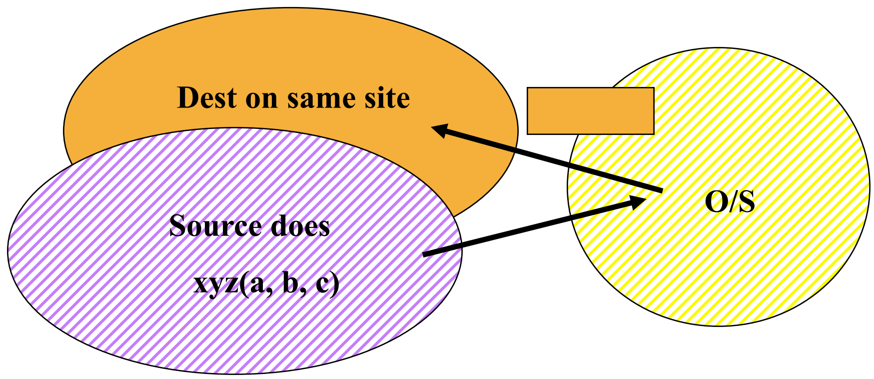
119

RPC in normal case



120

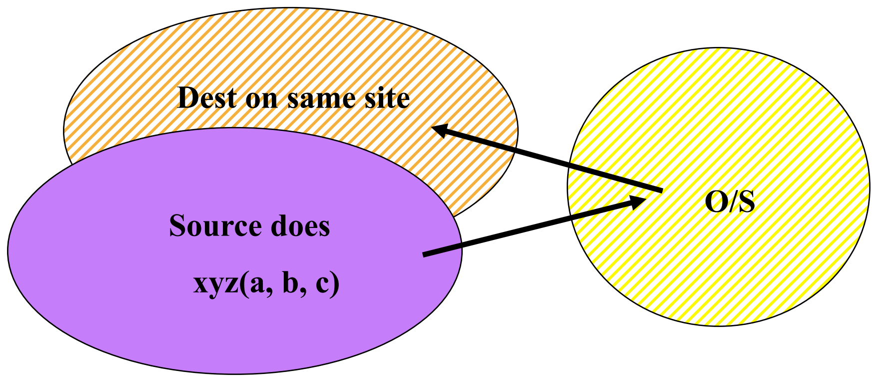
RPC in normal case



Important optimizations: LRPC

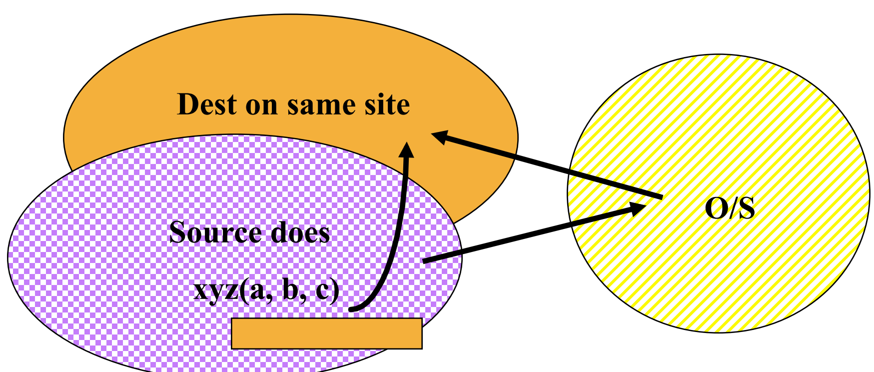
- Lightweight RPC (LRPC):
- Uses memory mapping to pass data
- Reuse kernel thread
- Single system call: send_rcv or rcv_send

LRPC



123

LRPC



arguments directly visible through remapped memory

124

LRPC performance impact

- 10-fold improvement over hand-optimized RPC implementation
- Two memory remappings, no context switch
- 50 times faster than standard RPC
- Easy to ensure exactly once

FBUFS

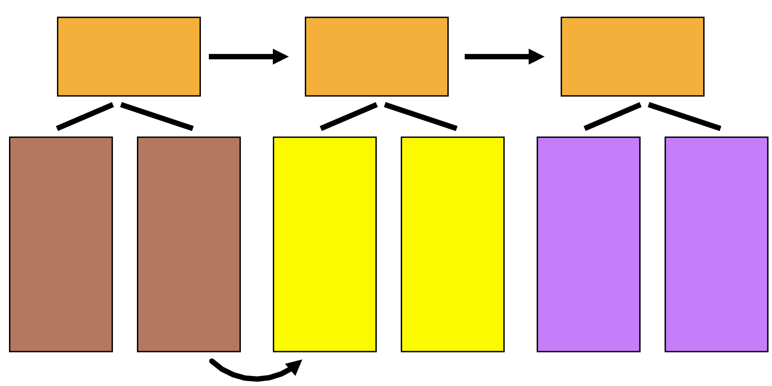
Fbufs

- Speed up layered protocols
- Buffer management overhead
- Solution: "cache" buffers
  - memory management & protection
- Stack layers share memory

127

Fbufs

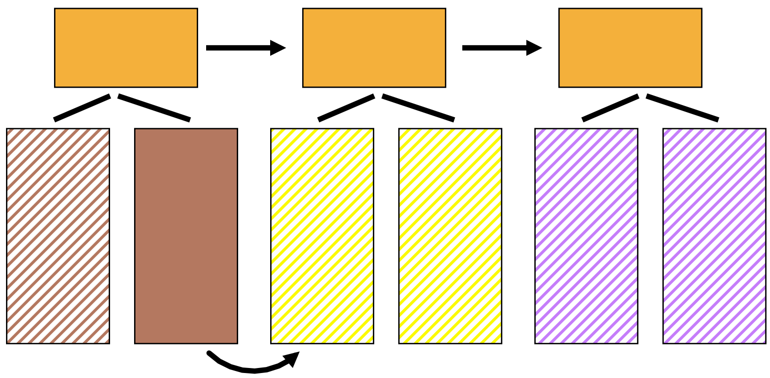
control flows through stack of layers, or pipeline of processes



128

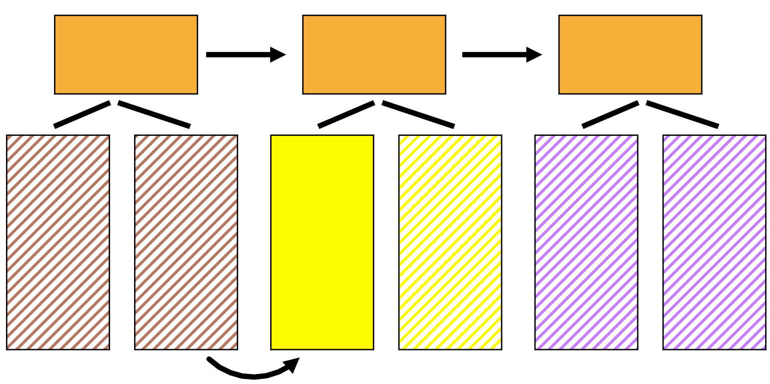
Fbufs

control flows through stack of layers, or pipeline of processes



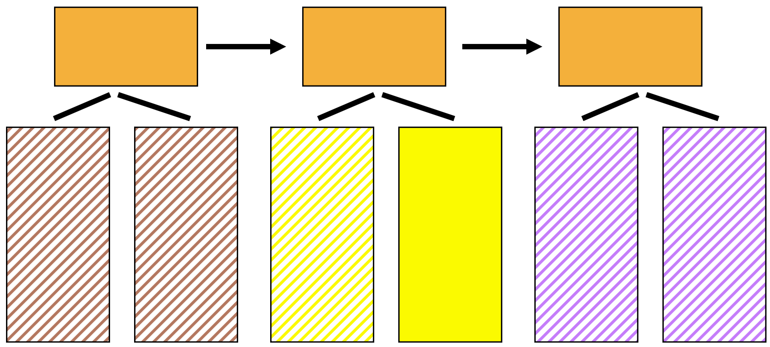
Fbufs

control flows through stack of layers, or pipeline of processes



Fbufs

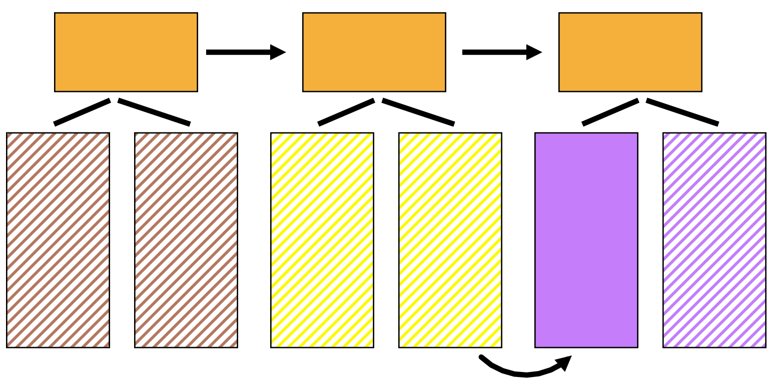
control flows through stack of layers, or pipeline of processes



131

Fbufs

control flows through stack of layers, or pipeline of processes



132

Where are Fbufs used?

- Most kernels use similar ideas
- Many application-layer libraries

133