Imperial College London

Computer Networks and Distributed Systems

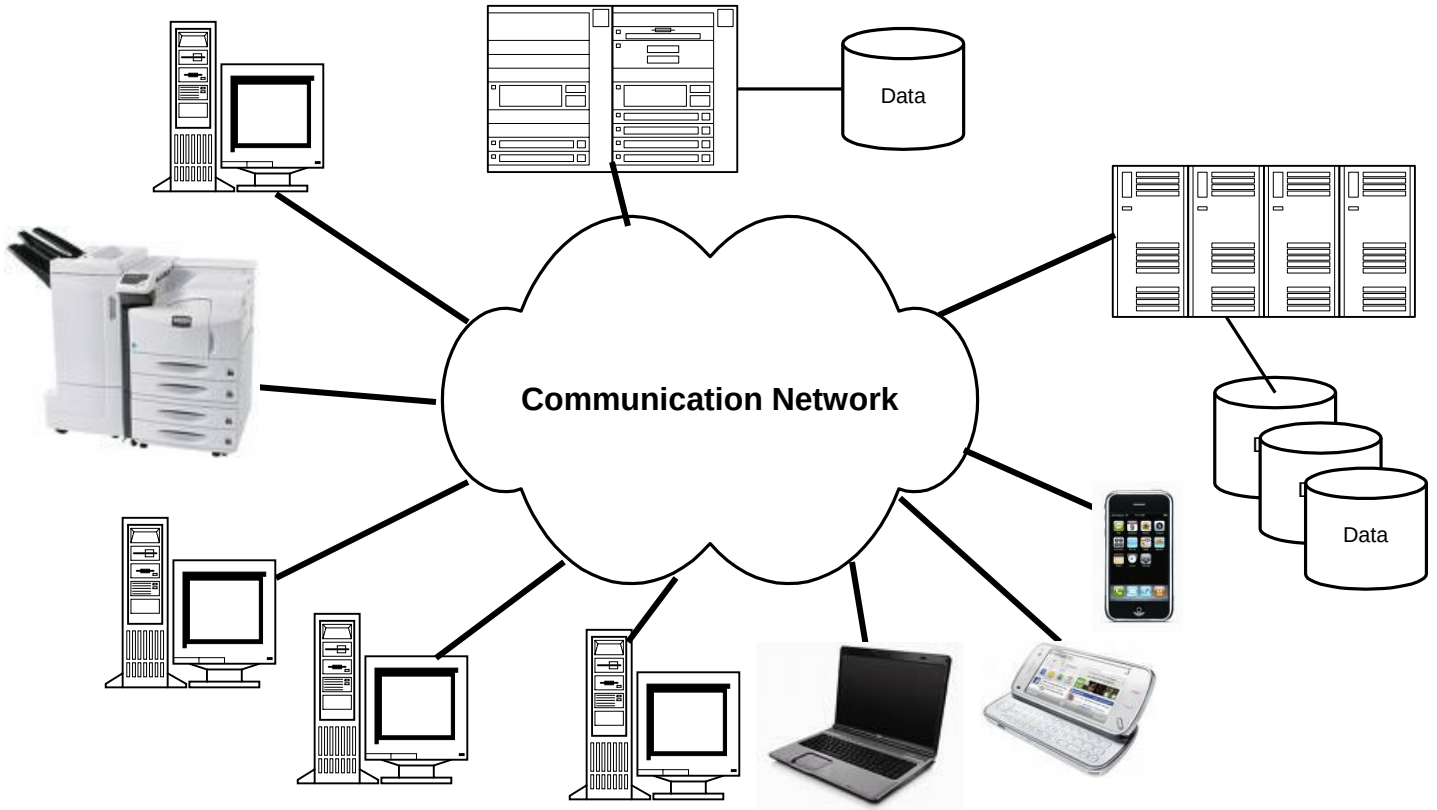
Distributed Systems – Wrap-Up

Course 527 – Spring Term 2014-2015

Anandha Gopalan

a.gopalan@imperial.ac.uk
http://www.doc.ic.ac.uk/~axgopala

What is a Distributed System?

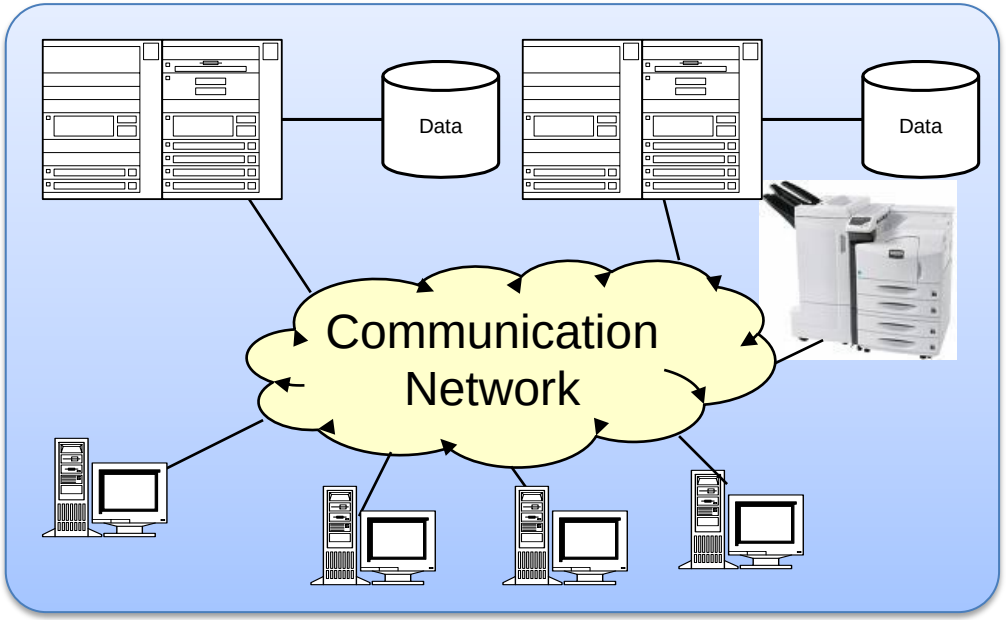


Definition

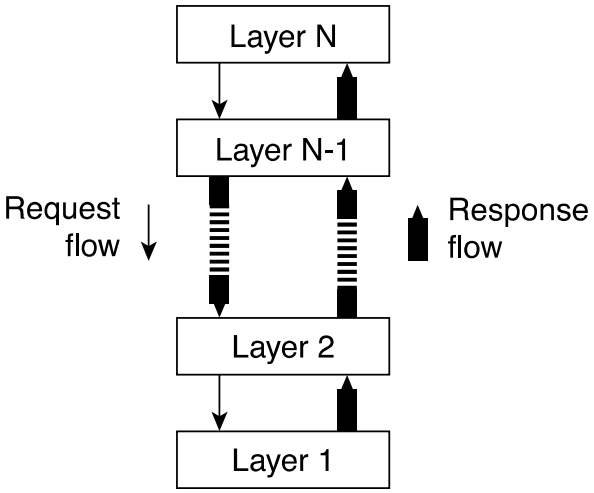
- A distributed system consists of a collection of autonomous computers interconnected by a computer network and equipped with distributed system software to form an integrated computing facility

- Components interact and cooperate to achieve a common goal

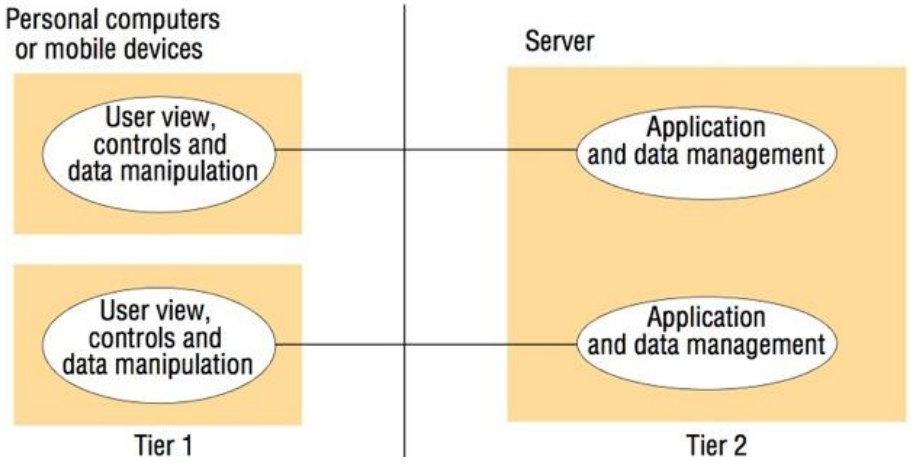
Processes co-ordinate by means of messages transferred over a communication network



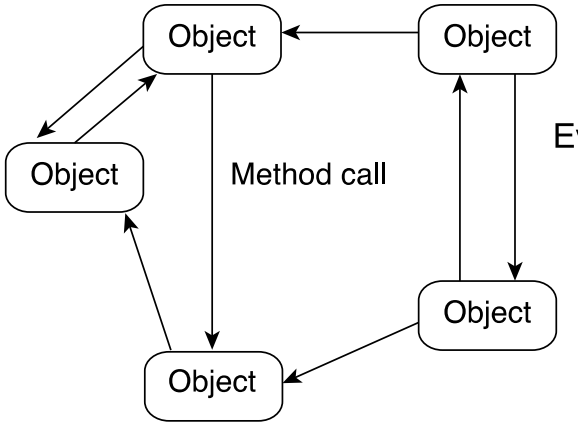
Architectural Styles



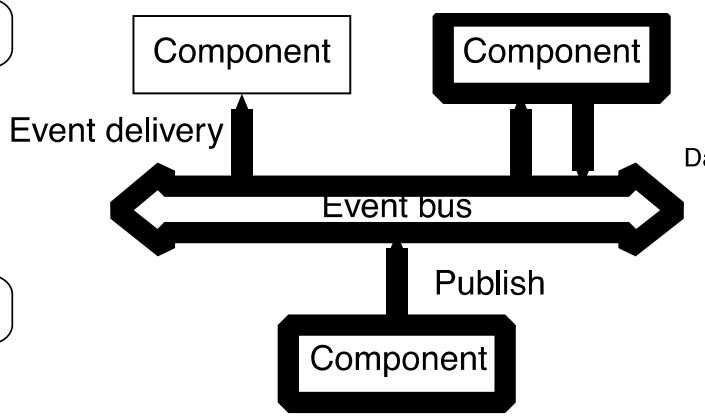
Layers



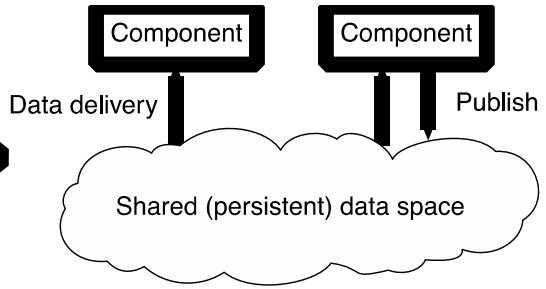
Tiers



Peers (Objects, Proc.,…)

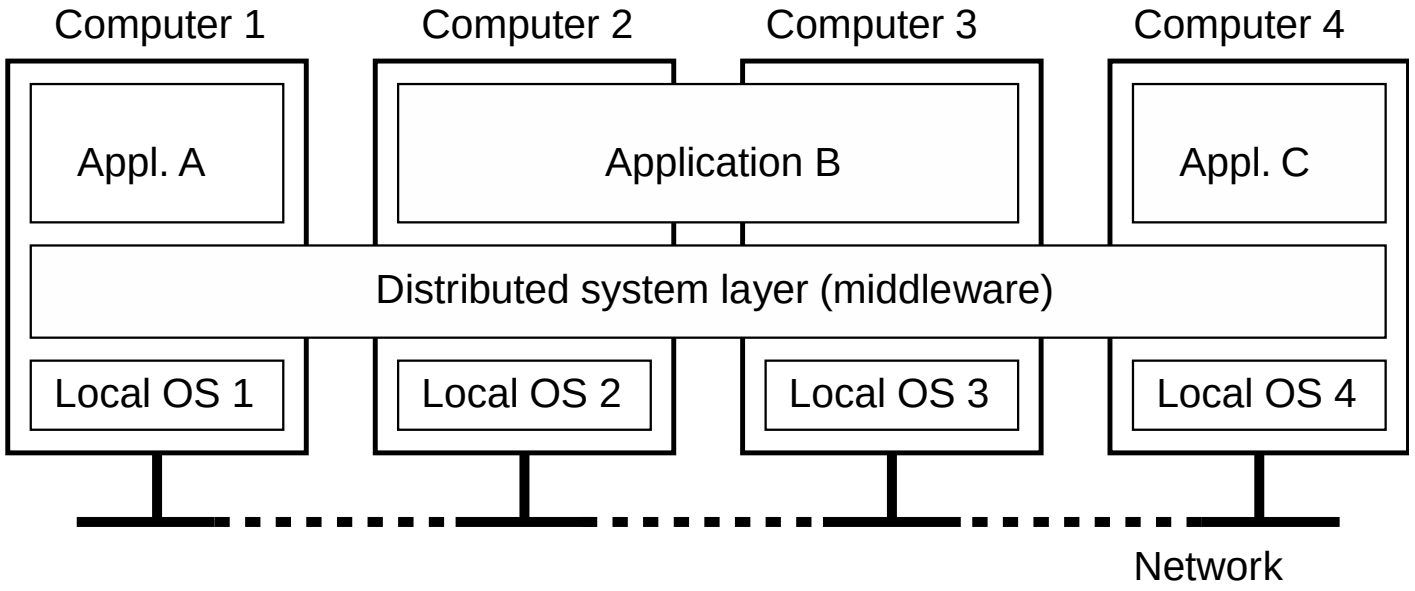


Event Based



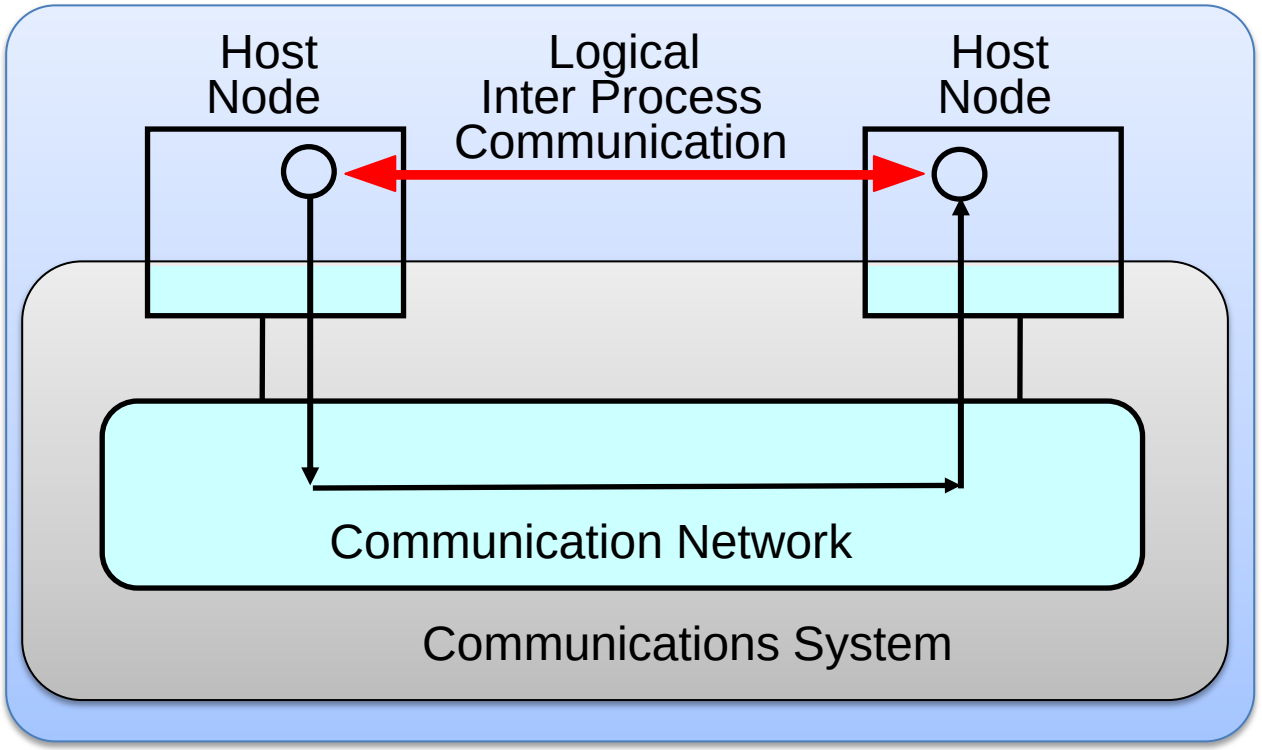
Shared memory based

So What is a Distributed System?



- Abstracting from the specificities of every node to consider the computing environment as a single coherent system
- Requires: interaction abstractions, middleware and services
- What is the middleware trying to do for you?

Inter Process Communication (IPC)



How do we achieve this?

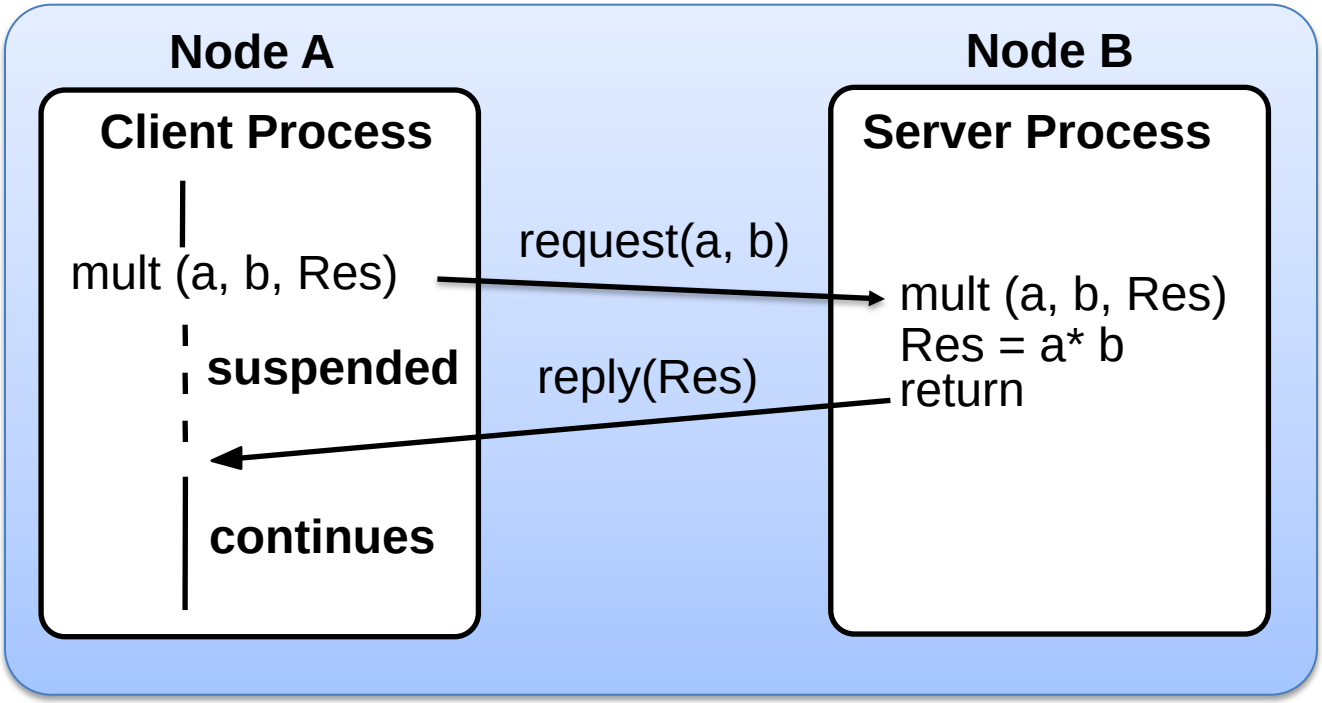
- RPC (Remote Procedure Call)
- RMI (Remote Method Invocation)

Why RPC/RMI ?

- What is needed?
  - Primitives to call method and send/receive required information (parameters, result, etc.) without the programmer having to worry about how it is done under the hood

- Why not just use Sockets (UDP/TCP)?
  - Too low-level an abstraction to be productively used by programmers
  - No method for marshalling a set of data values into a single contiguous chunk of bytes which can be sent as a message
  - Data heterogeneity (representation on different machines via different compilers) is not addressed
  - Programming paradigms such as client-server is cumbersome
  - …..

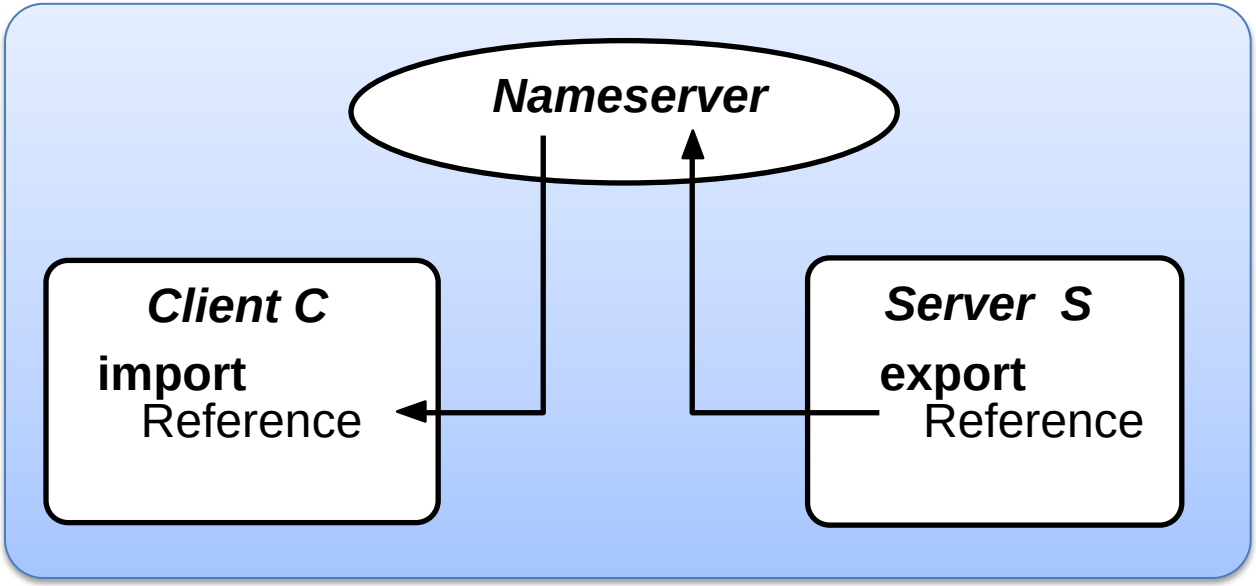
RPC Interaction



- What do we need to know in order to achieve this?
  - Clients need to know location of server
  - Clients needs to know the method signature

Binding

- Maps an RPC interface used by a client to the implementation of that interface provided by a server
  - Server needs to export the reference to the interface
  - Client needs to import the reference to the interface

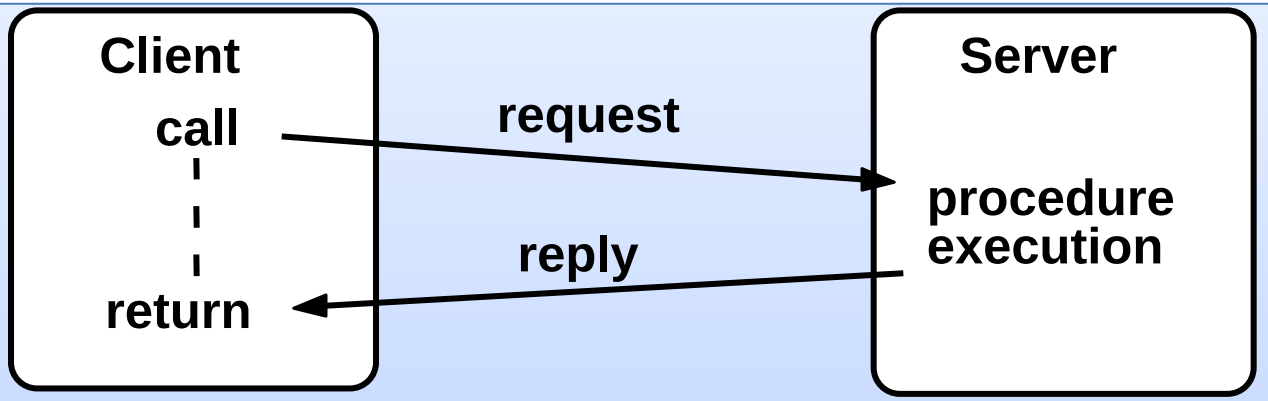


Stub Procedures

- Client has a local stub for every remote procedure it can call
- Server has local stub for every procedure which can be called by a remote client
- Stub procedures perform
  - Parameter marshalling (packing) → assemble parameters in communication system messages
  - Unpack received messages and assign values to parameters
  - Transform data representations if necessary
  - Access communication primitives to send/receive messages
- Stubs can be generated automatically from an interface specification
- No need to worry about the low-level details

RPC Failures

- Remote procedure calls differ from local procedure calls in the ways that they can fail



Client machine may crash during/after call

Duplicates or messages may be lost as a result of comms failure or corruption

Server machine may crash before request is received or during execution before reply is sent

Name server failure - cannot locate server

RPC Failures – Call Semantics

- Different call semantics possible
  - Depending on fault-tolerant measures employed
- Maybe (Best-Effort) Call Semantics
  - Send message and hope call works
- At-Least-Once Call Semantics
  - Try up-to n times for call to work → useful for idempotent operations
- At-Most-Once Call Semantics
  - Procedure executed at-most once
  - Most commonly used
- Transaction Call Semantics
  - Procedure is either completely executed or not at all

RPC Summary

- Interaction primitive similar to (but not the same) as well known procedure call
- Often specific to a particular programming language or operating system
- RPC calls suspend client for network roundtrip delay + procedure execution time
- Not suitable for multimedia streams or bulk data transfer
- What is the need for RMI (Remote Method Invocation)?
  - To invoke methods on remote objects (Java's answer to RPC)

RMI vs. RPC

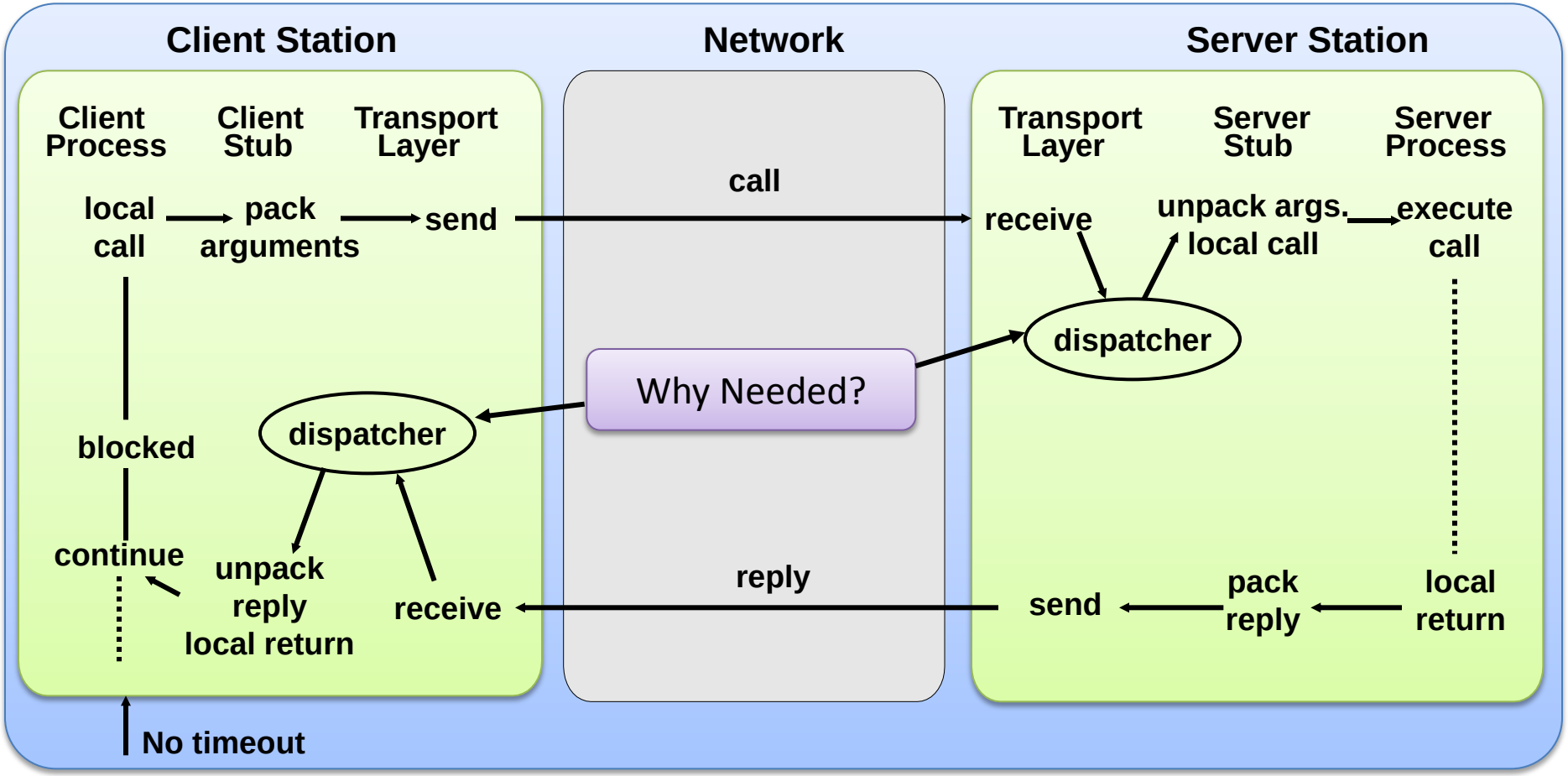
- Call remote methods in objects
- Can dynamically download stub
- Allows dynamic invocation
  - Interfaces can change at run-time
- Can return objects
  - More remote references distributed amongst other JVM instances
  - Better error handling

- Call remote functions exported
- Stub is compiled into the client

RMI Interaction

1. Server binds its object with the name server (Naming.bind/rebind)
2. Client requests an object from the server (Naming.lookup)
3. Server returns a stub representing the remote object back to the client
4. Client invokes a method on the stub → as if it's a local object
5. Stub marshals the request parameters and sends request to server proxy
6. Server proxy unmarshalls the request parameters
7. Proxy invokes requested method on Server and returns result
8. Result is marshalled and passed back to Client stub
9. Stub unmarshalls the return value and passes it to Client

Implementation



At most once semantics

client receives reply – procedure executed exactly once

on failure i.e. no reply received – don't know

Need to worry about parameters requiring multiple messages

Dispatcher

- Server needs dispatcher to map incoming calls onto relevant procedure
- Dispatcher in client passes incoming reply message to relevant stub procedure
- Interface compiler generates a number (or name) for each procedure in interface – inserted into call message by client stub procedure
- Dispatcher at server receives all call messages and uses procedure number (name) to identify called procedure

RMI Dispatcher

- Java uses reflection and a generic dispatcher
- Upon request, the dispatcher
  - unmarshalls method object,
  - uses method information to unmarshall arguments
  - converts remote object reference to local object reference
  - calls method object's invoke method supplying local object reference and arguments
  - when method executed, marshalls result or exceptions into reply message and sends it back to client

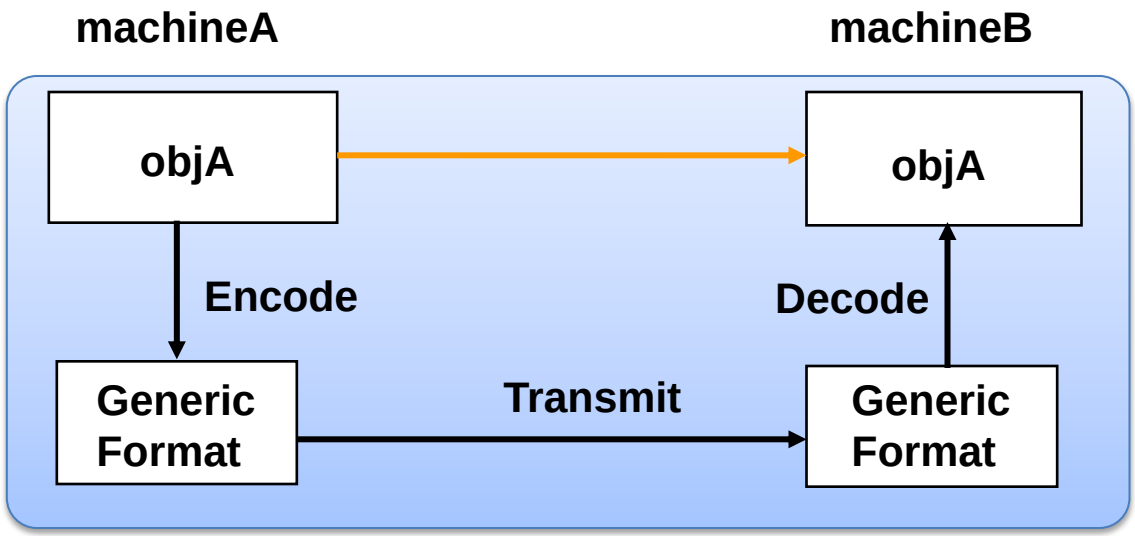
Implementation

- Achieved sending and receiving information to allow for RPC/RMI → anything missing?
  - How do you agree on a data format such that what the client receives/sends is the same as the data from/to the server

Data Representation

- Language heterogeneity
  - Data structures represented differently
  - Lack of some data structures
- Processor heterogeneity
  - Representations of Characters, Integers, Reals, …
  - Big endian vs. Little endian
- Transform representations when transferring data
  - Can have transformation between each pair of machines → N * (N – 1) translators for N machines

Data representation



- Should be optional to avoid unnecessary overheads
  - machineA and machineB use same representation

Data Representation

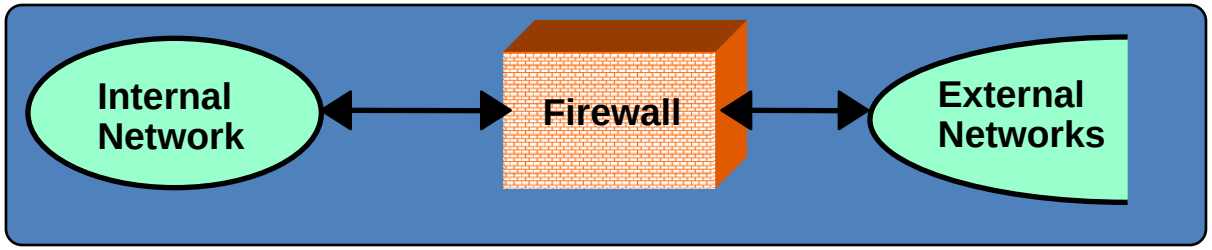
- XDR (Standard External Data Representation)
  - Uses different encoding for fixed and variable length data
- XML (Extensible Markup Language)
  - Uses explicit tags for encoding data
- Complex data types must be “flattened” for transfer to a remote machine (or to disc store) and addresses transformed to local references (e.g. array index)

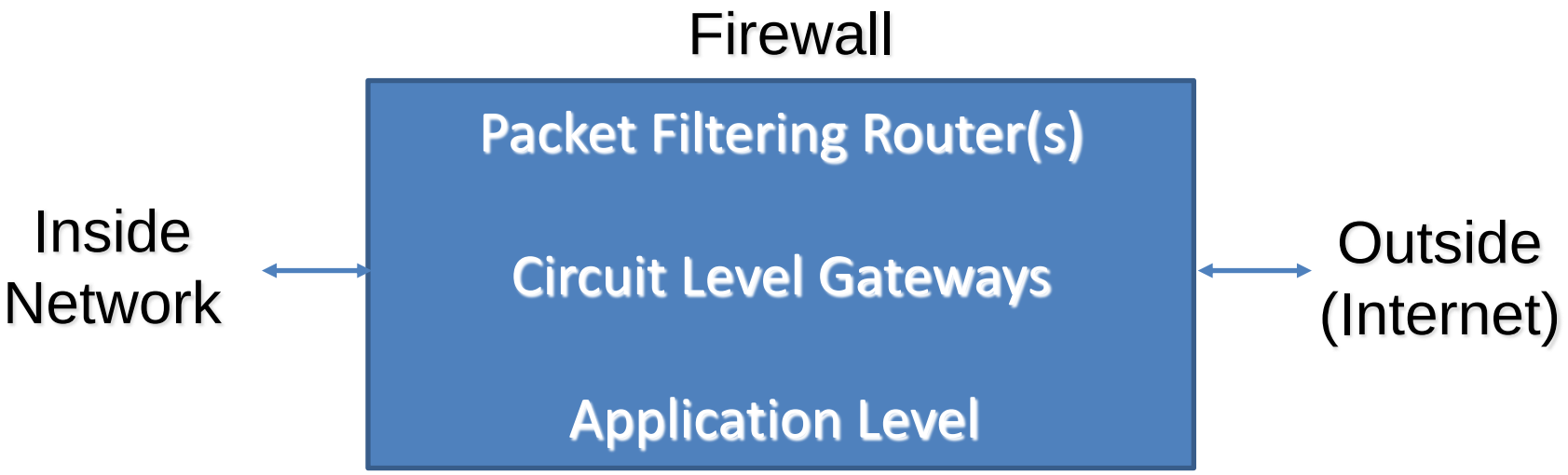
Data Representation

- Java uses Object Serialization
  - Object is represented as a sequence of bytes that includes the object's data as well as information about the object's type and the types of data stored in the object

Firewalls

A security gateway between internal and external networks





Analyze inbound packets and based upon existing rules decide to block/allow packet

Firewall Components

- Packet Filtering Routers (a.k.a. Chokes, Screening Filters)
  - Analyse each packet in isolation and decide
- Circuit-level Gateways (a.k.a Stateful inspection firewalls)
  - Relay's connections and maintains connection state → open, authorised connections
  - Can also authenticate users
  - Can drop connections based on destination, incorrect connection packets, time, volume, etc.
- Application-level Gateways (a.k.a Proxy firewalls)
  - Most advanced
  - Can block/filter/report based on app-level msg. content
  - Can scan for data leaks, viruses, etc.
  - Can rewrite data
  - In the midst of a 'logical' connection allowing it to monitor traffic

- Gateway apps normally run on so called BASTION HOSTS

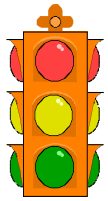
Bastion Host

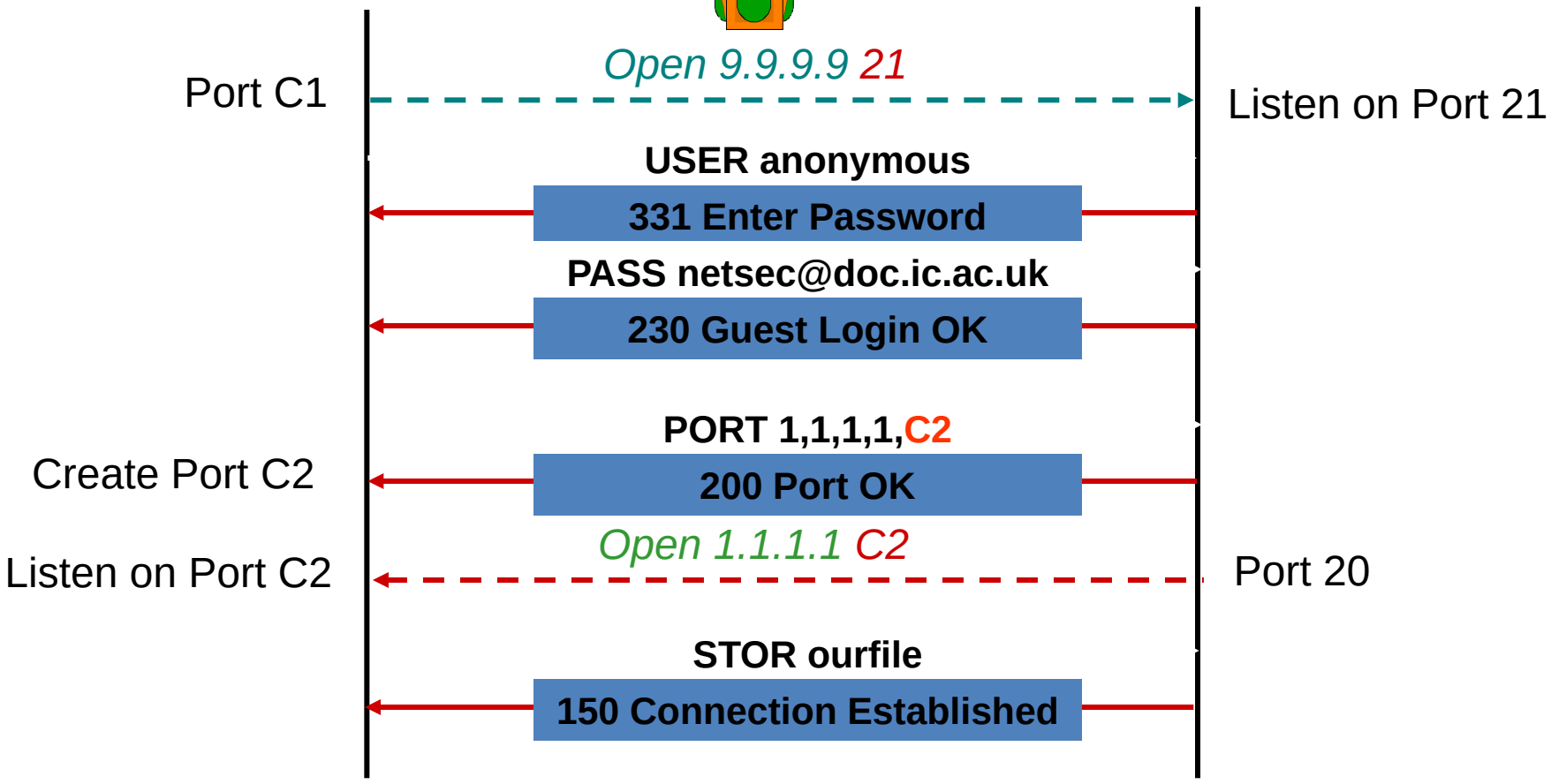
- Runs application-level and circuit-level gateways
- Can run other servers too
- Performs auditing / accounting
- Should run a "Trusted"/Secure OS
- Administer via a dedicated terminal

Minimal OS

- Remove inessential applications, utilities, services, e.g. cc, awk, sed, ld, X11
- Set file permissions, turn on file quotas, process limits etc.
- No regular user accounts
- No NFS mounts
- Make filesystems read-only if possible

Filtering FTP (Dynamic Callbacks)





FTP Rules

Rule	Dir.	Action	Src Addr	Src Port	Dest Addr	Dest Port	TCP Flags	Description
1	Out	Allow	1.1.1.1	*	9.9.9.9	21		FTP outgoing
2	In	Allow	9.9.9.9	21	1.1.1.1	*	ACK	Only replies allowed
3	In	Allow	9.9.9.9	20	1.1.1.1	>1023		
4	Out	Allow	1.1.1.1	*	9.9.9.9	20	ACK	