Distributed Systems

Architecture, processes, and communication

TDTS04 – Computer Networks and Distributed Systems

Carl Magnus Bruhner, ADIT/IDA

LINKÖPING UNIVERSITY

Based on slides by M. van Steen and A. S. Tanenbaum, Distributed Systems, 4th ed., distributed-systems.net

Architecture of distributed systems

LINKÖPING
UNIVERSITY

Architectural styles

Basic idea

A style is formulated in terms of

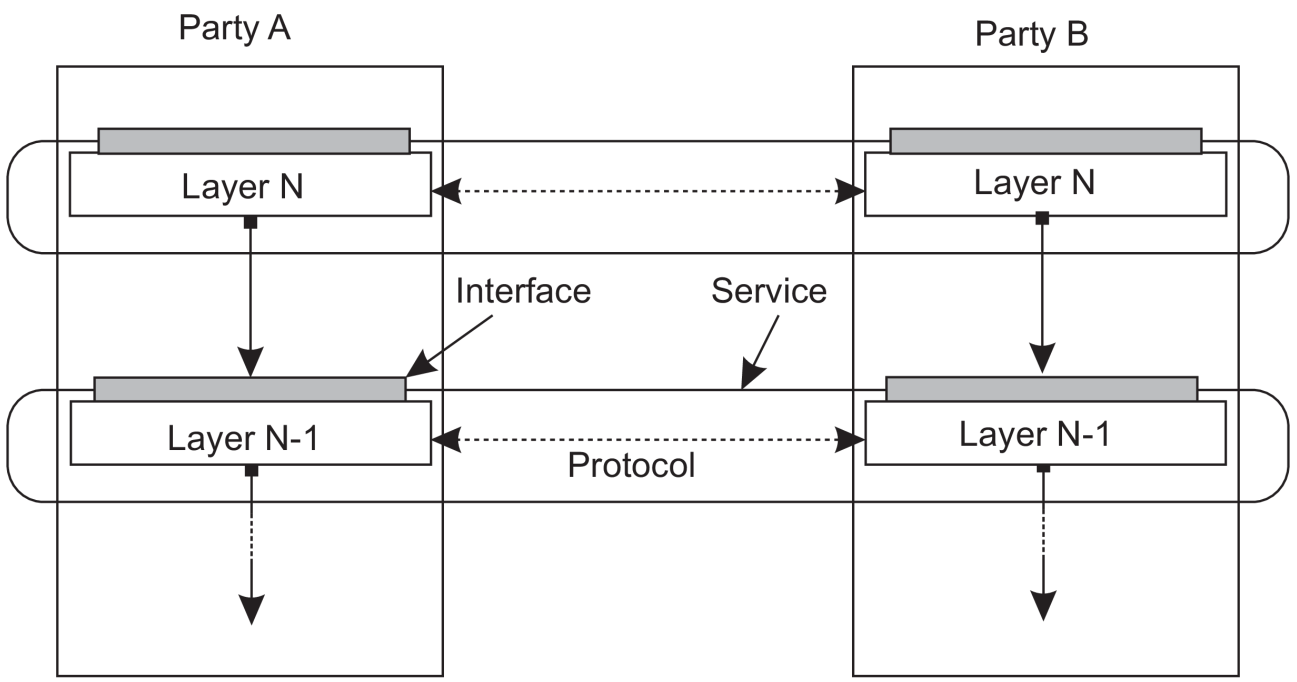
- (replaceable) components with well-defined interfaces
- the way that components are connected to each other
- the data exchanged between components
- how these components and connectors are jointly configured into a system.

Connector

A mechanism that mediates communication, coordination, or cooperation among components. Example: facilities for (remote) procedure call, messaging, or streaming.

Example: communication protocols

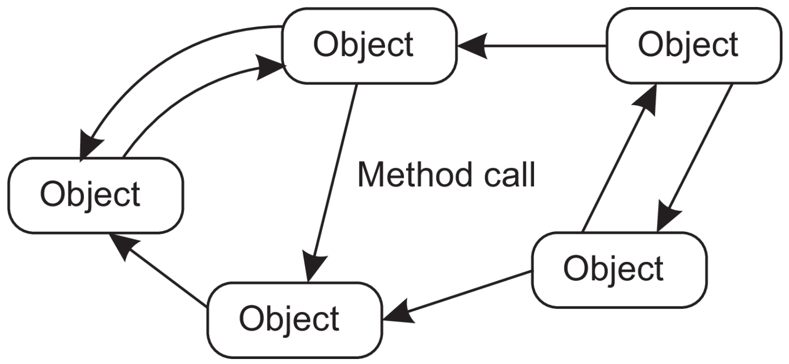
Protocol, service, interface

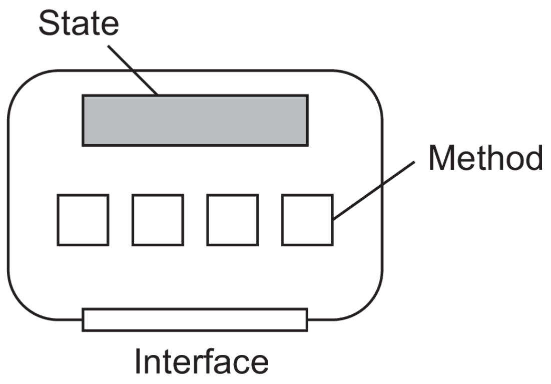


Object-based style

Essence

Components are objects, connected to each other through procedure calls. Objects may be placed on different machines; calls can thus execute across a network.





Encapsulation

Objects are said to encapsulate data and offer methods on that data without revealing the internal implementation.

Object-based styles

- Common Object Request Broker Architecture (CORBA)
- Java Remote Method Invocation (Java RMI)—the object-oriented equivalent of remote procedure calls (RPC)
- *…not used much today, so you don't need to dive into this even though it is mentioned in the course syllabus.*

RESTful architectures

Representational State Transfer

Essence

View a distributed system as a collection of resources, individually managed by components. Resources may be added, removed, retrieved, and modified by (remote) applications.

1. Resources are identified through a single naming scheme
2. All services offer the same interface
3. Messages sent to or from a service are fully self-described
4. After executing an operation at a service, that component forgets everything about the caller

Basic operations

Available as of HTTP methods

```
GET /index.html HTTP/1.1
Host: liu.se
...
```

Operation	Description
PUT	Create a new resource
GET	Retrieve the state of a resource in some representation
DELETE	Delete a resource
POST	Modify a resource by transferring a new state

Example: Amazon's Simple Storage Service

Essence

Objects (i.e., files) are placed into buckets (i.e., directories). Buckets cannot be placed into buckets. Operations on `ObjectName` in bucket `BucketName` require the following identifier:

```
http://BucketName.s3.amazonaws.com/ObjectName
```

Typical operations

All operations are carried out by sending HTTP requests:

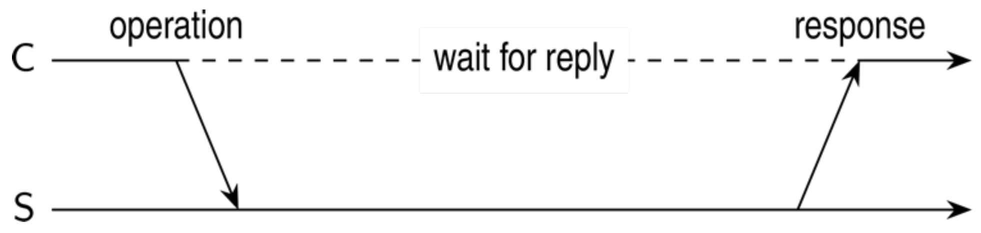
- Create a bucket/object: `PUT`, along with the URI
- Listing objects: `GET` on a bucket name
- Reading an object: `GET` on a full URI

Centralized system architectures

Basic Client–Server Model

Characteristics:

- There are processes offering services (servers)
- There are processes that use services (clients)
- Clients and servers can be on different machines
- Clients follow request/reply model regarding using services

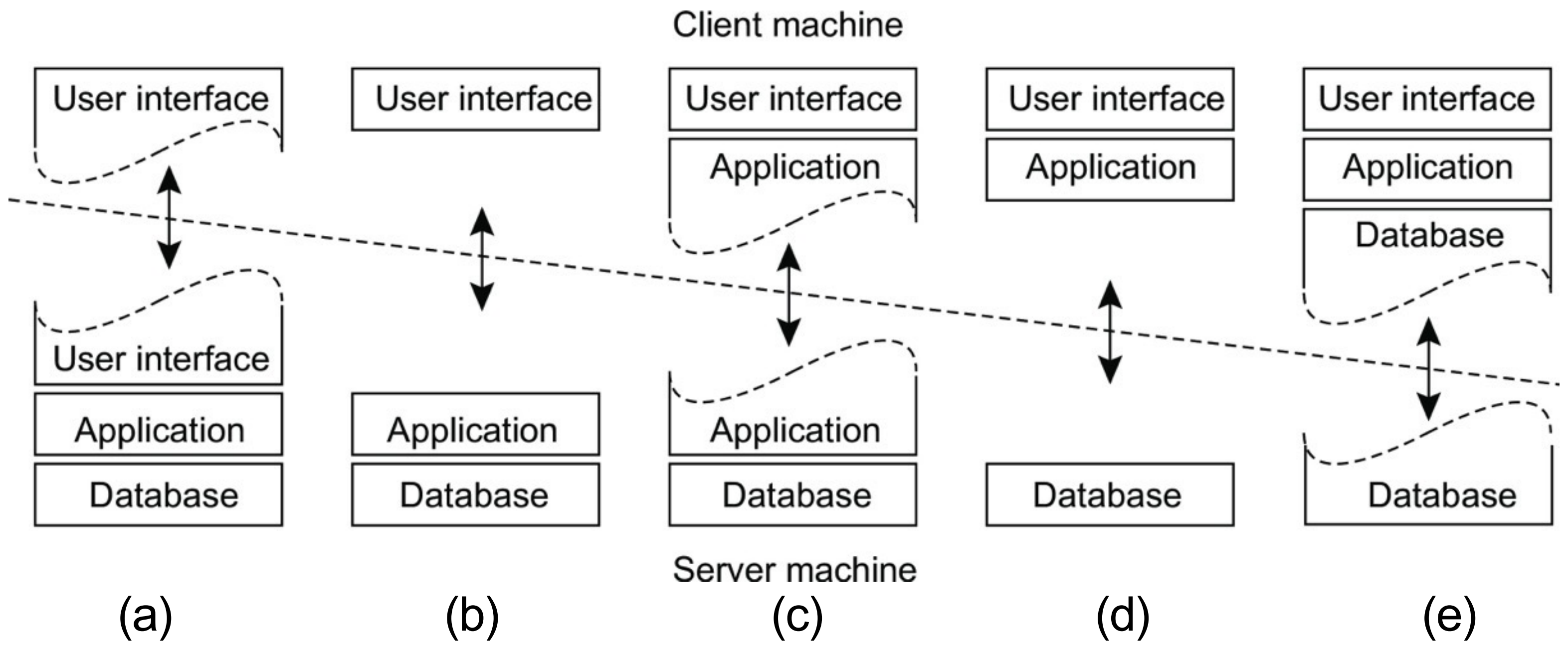


Multi-tiered centralized system architectures

Some traditional organizations

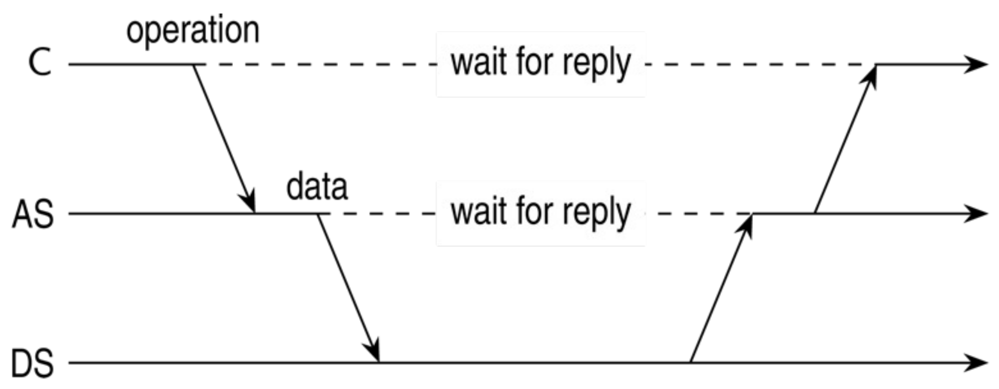
- Single-tiered: dumb terminal/mainframe configuration
- Two-tiered: client/single server configuration
- Three-tiered: each layer on separate machine

Traditional two-tiered configurations



Being client and server at the same time

Three-tiered architecture

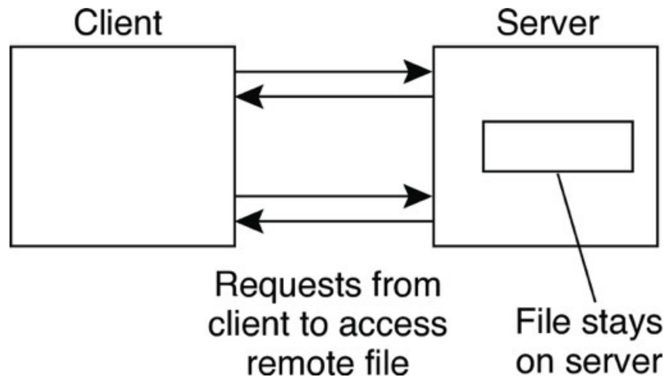


Example: The Network File System

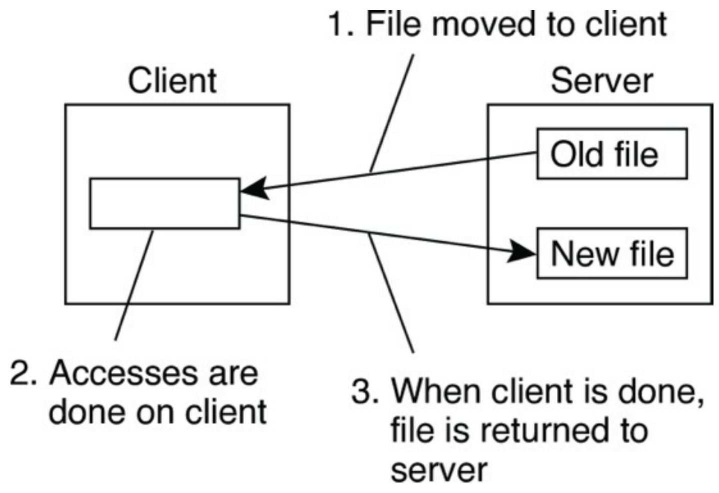
Foundations

Each NFS server provides a standardized view of its local file system: each server supports the same model, regardless the implementation of the file system.

The NFS remote access model



Remote access

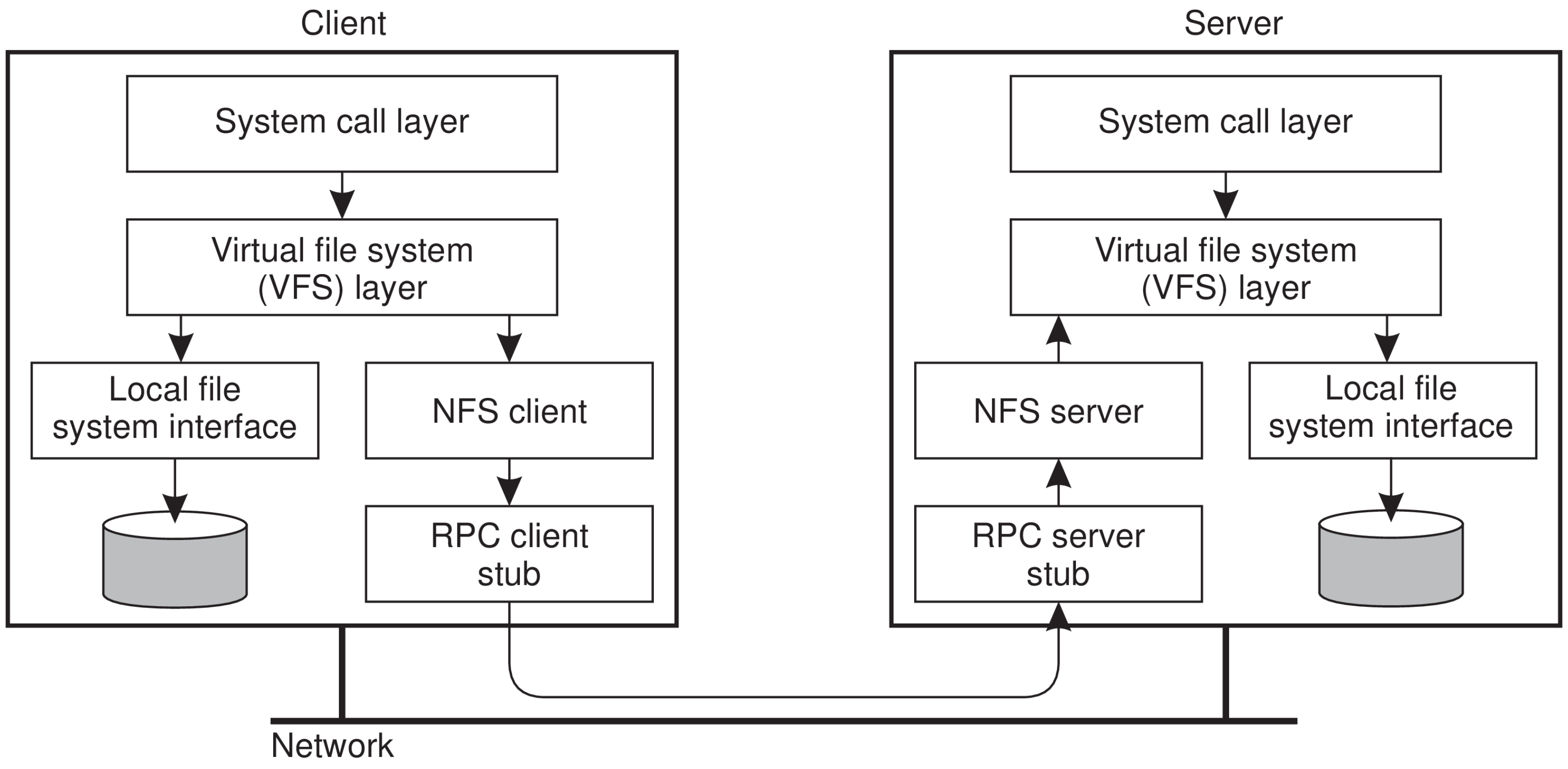

1. File moved to client
Client
Server
Old file
New file
2. Accesses are done on client
3. When client is done, file is returned to server

Upload/download

Note

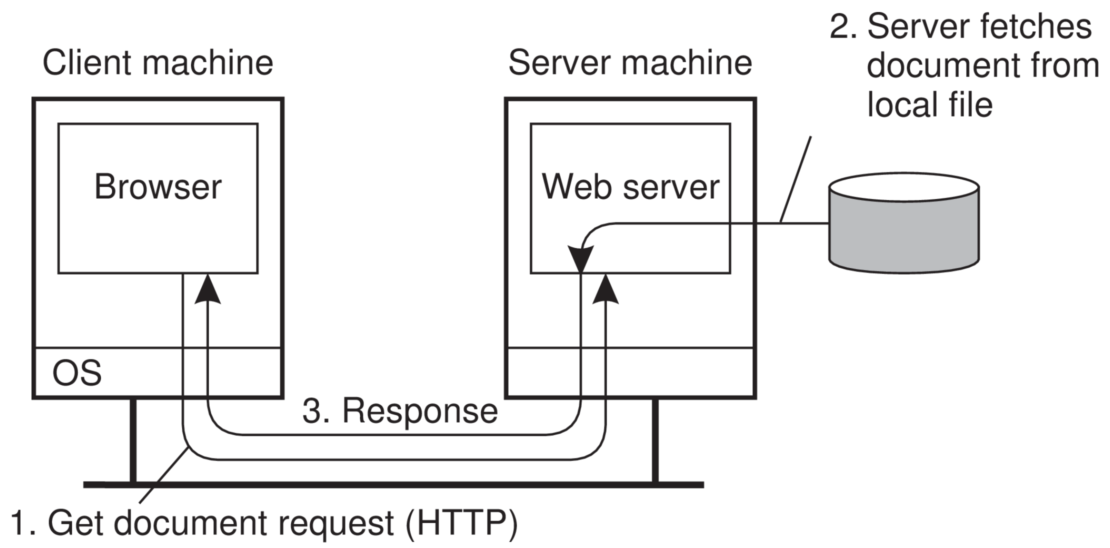
FTP is a typical upload/download model. The same can be said for systems like Dropbox.

NFS architecture


Client
Server
System call layer
Virtual file system (VFS) layer
Local file system interface
NFS client
RPC client stub
System call layer
Virtual file system (VFS) layer
NFS server
Local file system interface
RPC server stub
Network

Example: Simple Web servers

Back in the old days...



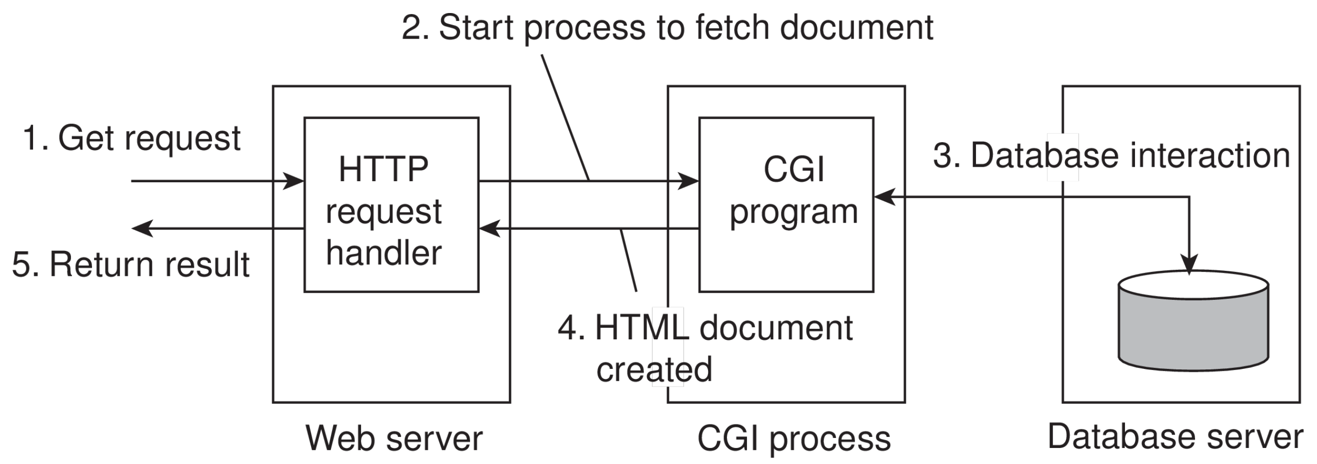
2. Server fetches document from local file
Client machine
Server machine
Browser
Web server
OS
3. Response
1. Get document request (HTTP)

...life was simple:

- A website consisted as a collection of HTML files
- HTML files could be referred to each other by a hyperlink
- A Web server essentially needed only a hyperlink to fetch a file
- A browser took care of properly rendering the content of a file

Example (cnt'd): Less simple Web servers

Still back in the old days...



...life became a bit more complicated:

- A website was built around a database with content
- A Webpage could still be referred to by a hyperlink
- A Web server essentially needed only a hyperlink to fetch a file
- A separate program (Common Gateway Interface) composed a page
- A browser took care of properly rendering the content of a file

Structured P2P

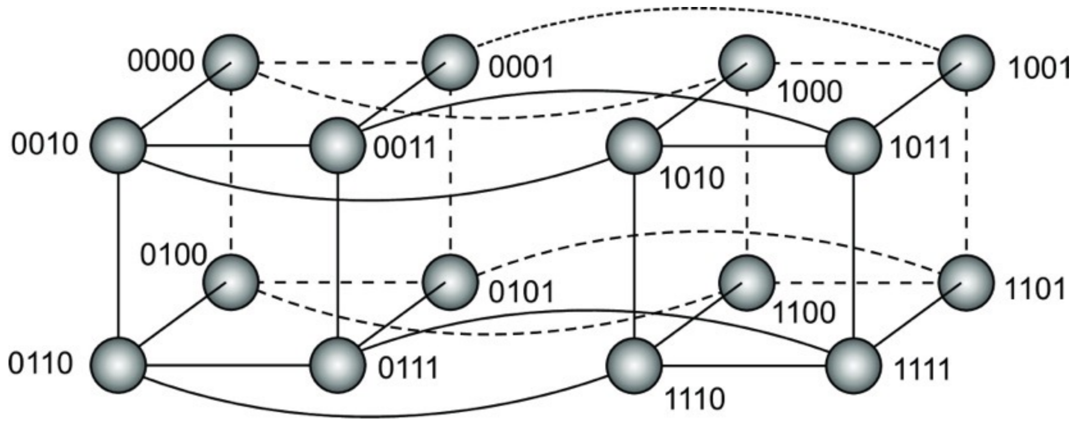
Essence

Make use of a semantic-free index: each data item is uniquely associated with a key, in turn used as an index. Common practice: use a hash function

$$key(data\ item) = hash(data\ item\text{'s}\ value).$$

P2P system now responsible for storing (*key*,*value*) pairs.

Simple example: hypercube



Looking up d with key $k \in \{0, 1, 2, \ldots, 2^4 - 1\}$ means routing request to node with identifier k.

Unstructured P2P

Essence

Each node maintains an ad hoc list of neighbors. The resulting overlay resembles a random graph: an edge $\langle u, v\rangle$ exists only with a certain probability $P[\langle u, v\rangle]$.

Searching

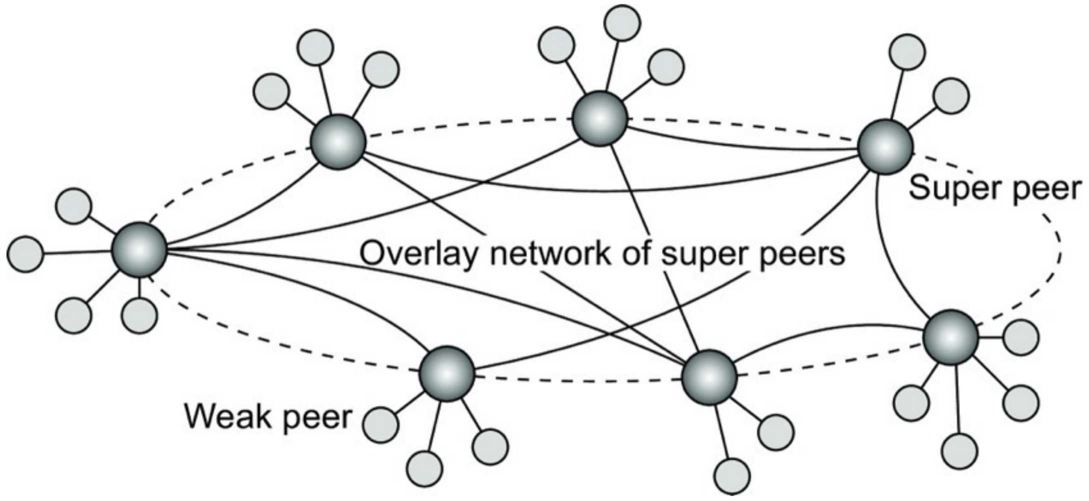
- Flooding: issuing node u passes request for d to all neighbors. Request is ignored when receiving node had seen it before. Otherwise, v searches locally for d (recursively). May be limited by a Time-To-Live: a maximum number of hops.
- Random walk: issuing node u passes request for d to randomly chosen neighbor, v. If v does not have d, it forwards request to one of *its* randomly chosen neighbors, and so on.

Super-peer networks

Essence

It is sometimes sensible to break the symmetry in pure peer-to-peer networks:

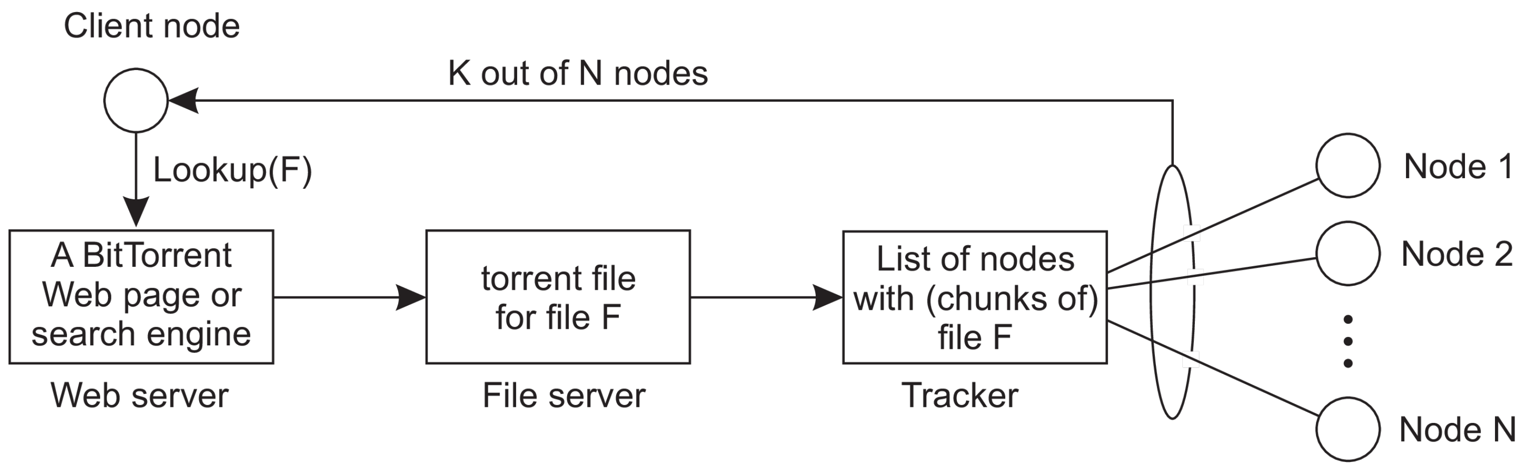
- When searching in unstructured P2P systems, having index servers improves performance
- Deciding where to store data can often be done more efficiently through brokers.



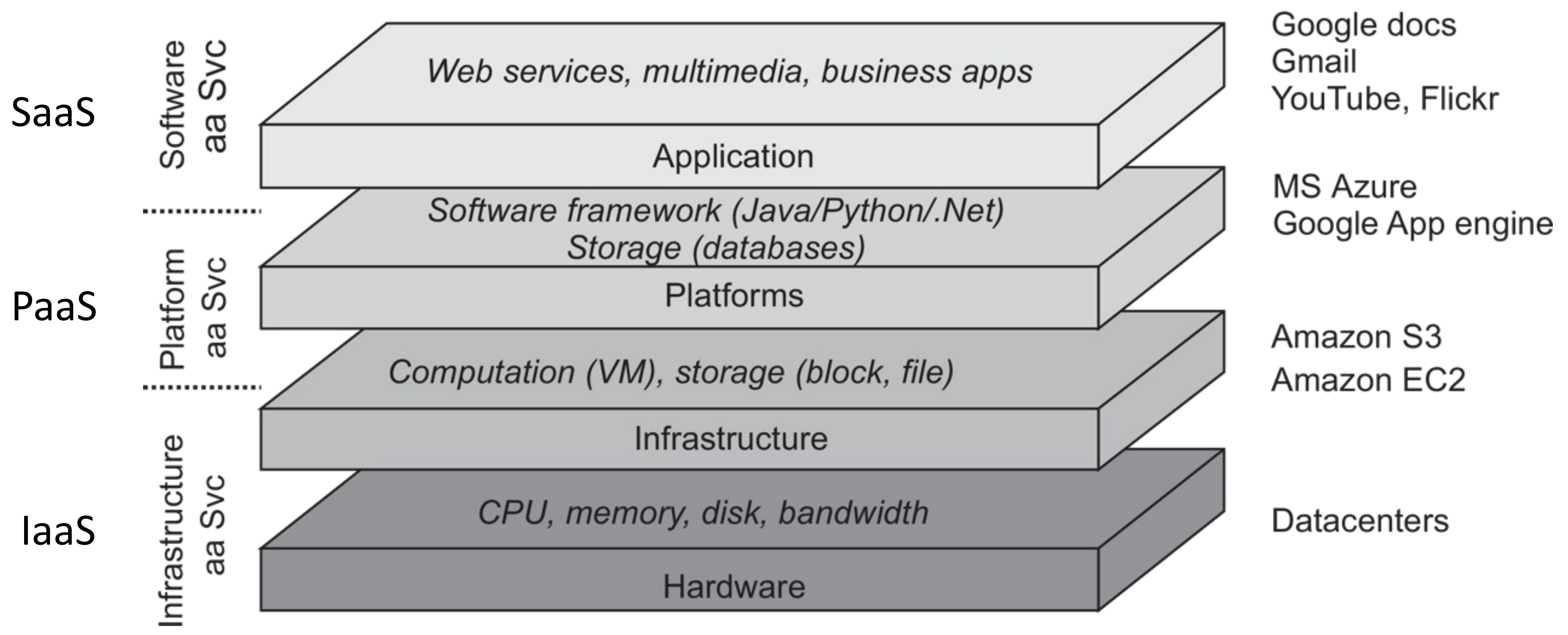
Collaboration: The BitTorrent case

Principle: search for a file *F*

- Lookup file at a global directory ⇒ returns a torrent file
- Torrent file contains reference to tracker: a server keeping an accurate account of active nodes that have (chunks of) *F*.
- *P* can join swarm, get a chunk for free, and then trade a copy of that chunk for another one with a peer *Q* also in the swarm.



Cloud computing



Cloud computing

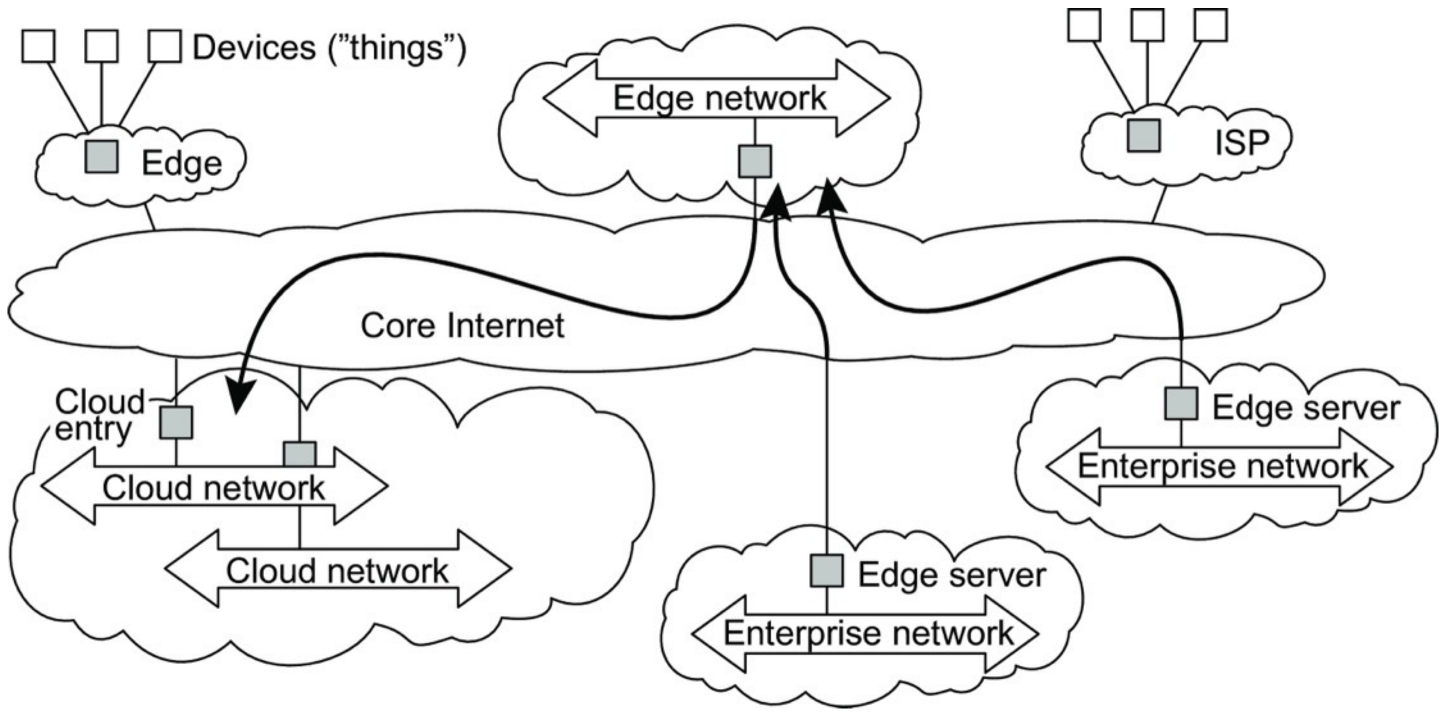
Make a distinction between four layers

- Hardware: Processors, routers, power and cooling systems. Customers normally never get to see these.
- Infrastructure: Deploys virtualization techniques. Evolves around allocating and managing virtual storage devices and virtual servers.
- Platform: Provides higher-level abstractions for storage and such. Example: Amazon S3 storage system offers an API for (locally created) files to be organized and stored in so-called buckets.
- Application: Actual applications, such as office suites (text processors, spreadsheet applications, presentation applications). Comparable to the suite of apps shipped with OSes.

Edge-server architecture

Essence

Systems deployed on the Internet where servers are placed at the edge of the network: the boundary between enterprise networks and the actual Internet.



Reasons for having an edge infrastructure

Commonly (and often misconceived) arguments

- Latency and bandwidth: Especially important for certain real-time applications, such as augmented/virtual reality applications. Many people underestimate the latency and bandwidth to the cloud.
- Reliability: The connection to the cloud is often assumed to be unreliable, which is often a false assumption. There may be critical situations in which extremely high connectivity guarantees are needed.
- Security and privacy: The implicit assumption is often that when assets are nearby, they can be made better protected. Practice shows that this assumption is generally false. However, securely handling data operations in the cloud may be trickier than within your own organization.

Edge orchestration

Managing resources at the edge may be trickier than in the cloud

- Resource allocation: we need to guarantee the availability of the resources required to perform a service.
- Service placement: we need to decide when and where to place a service. This is notably relevant for mobile applications.
- Edge selection: we need to decide which edge infrastructure should be used when a service needs to be offered. The closest one may not be the best one.

Observation

There is still a lot of buzz about edge infrastructures and computing, yet whether all that buzz makes any sense remains to be seen.

Processes in distributed systems

LINKÖPING
UNIVERSITY

VMs and cloud computing

Three types of cloud services

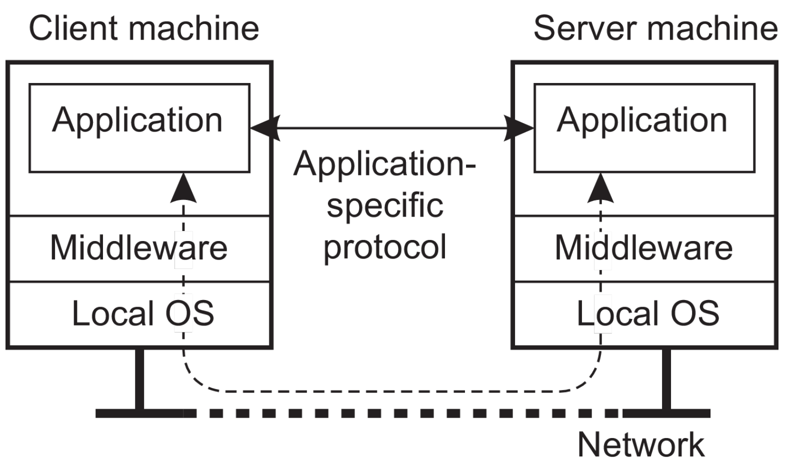
- Infrastructure-as-a-Service covering the basic infrastructure
- Platform-as-a-Service covering system-level services
- Software-as-a-Service containing actual applications

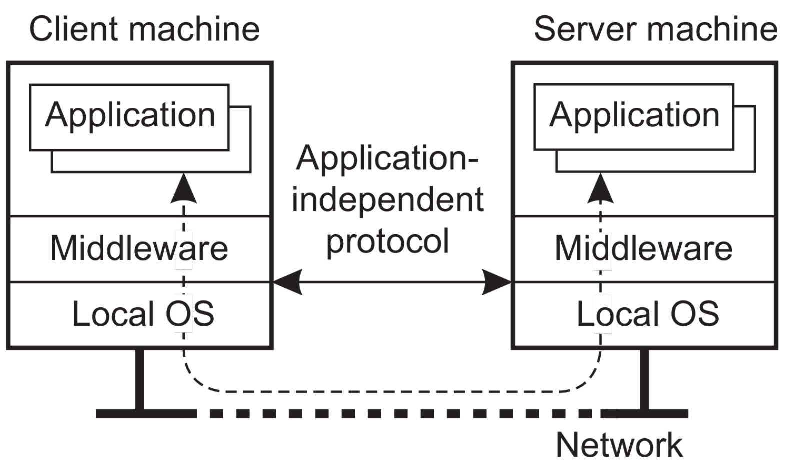
IaaS

Instead of renting out a physical machine, a cloud provider will rent out a VM (or VMM) that may be sharing a physical machine with other customers ⇒ almost complete isolation between customers (although performance isolation may not be reached).

Client-server interaction

Distinguish application-level and middleware-level solutions





Virtual desktop environment

Logical development

With an increasing number of cloud-based applications, the question is how to use those applications from a user's premise?

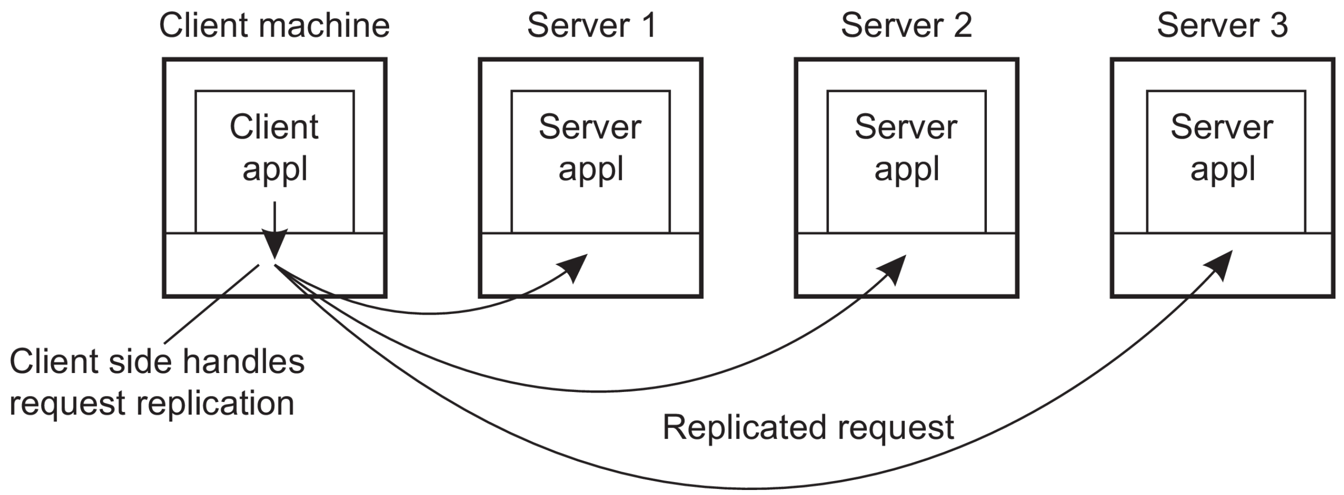
- Issue: develop the ultimate networked user interface
- Answer: use a Web browser to establish a seamless experience

The Google Chromebook

Client-side software

Generally tailored for distribution transparency

- Access transparency: client-side stubs for RPCs
- Location/migration transparency: let client-side software keep track of actual location
- Replication transparency: multiple invocations handled by client stub:



- Failure transparency: can often be placed only at client (we're trying to mask server and communication failures).

Servers: General organization

Basic model

A process implementing a specific service on behalf of a collection of clients. It waits for an incoming request from a client and subsequently ensures that the request is taken care of, after which it waits for the next incoming request.

Two basic types

- Iterative server: Server handles the request before attending a next request.
- Concurrent server: Uses a dispatcher, which picks up an incoming request that is then passed on to a separate thread/process.

Observation

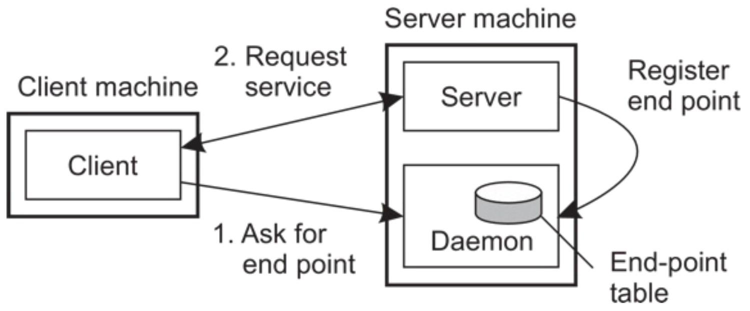
Concurrent servers are the norm: they can easily handle multiple requests, notably in the presence of blocking operations (to disks or other servers).

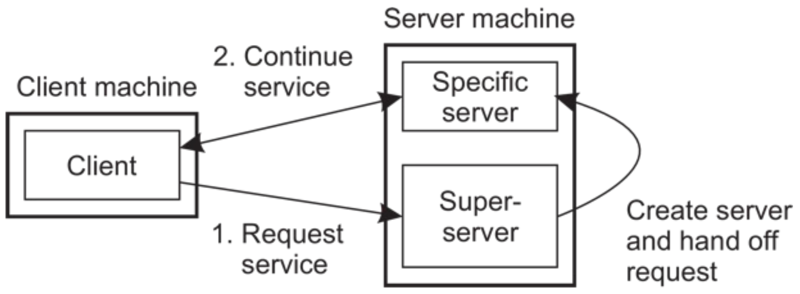
Contacting a server

Observation: most services are tied to a specific port

ftp-data	20	File Transfer [Default Data]
ftp	21	File Transfer [Control]
telnet	23	Telnet
smtp	25	Simple Mail Transfer
www	80	Web (HTTP)

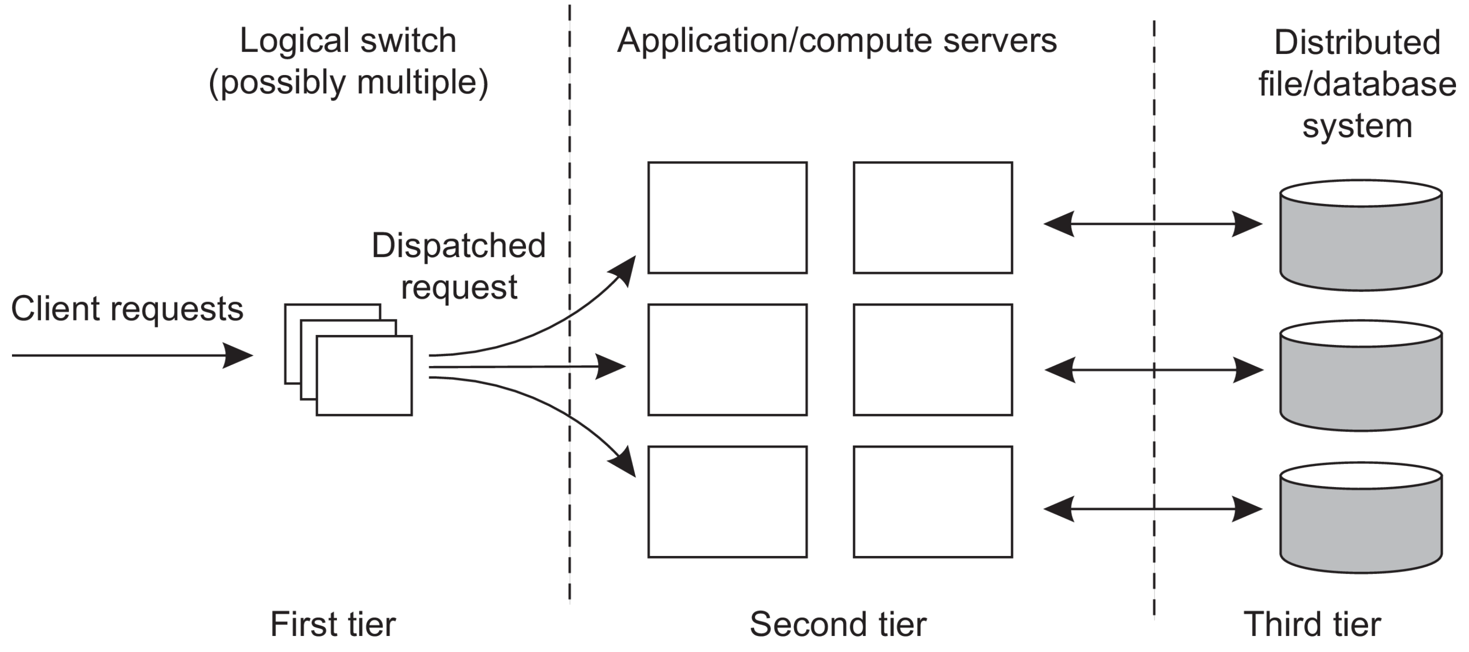
Dynamically assigning an end point: two approaches





Three different tiers

Common organization



Crucial element

The first tier is generally responsible for passing requests to an appropriate server: request dispatching

When servers are spread across the Internet

Observation

Spreading servers across the Internet may introduce administrative problems. These can be largely circumvented by using data centers from a single cloud provider.

Request dispatching: if locality is important

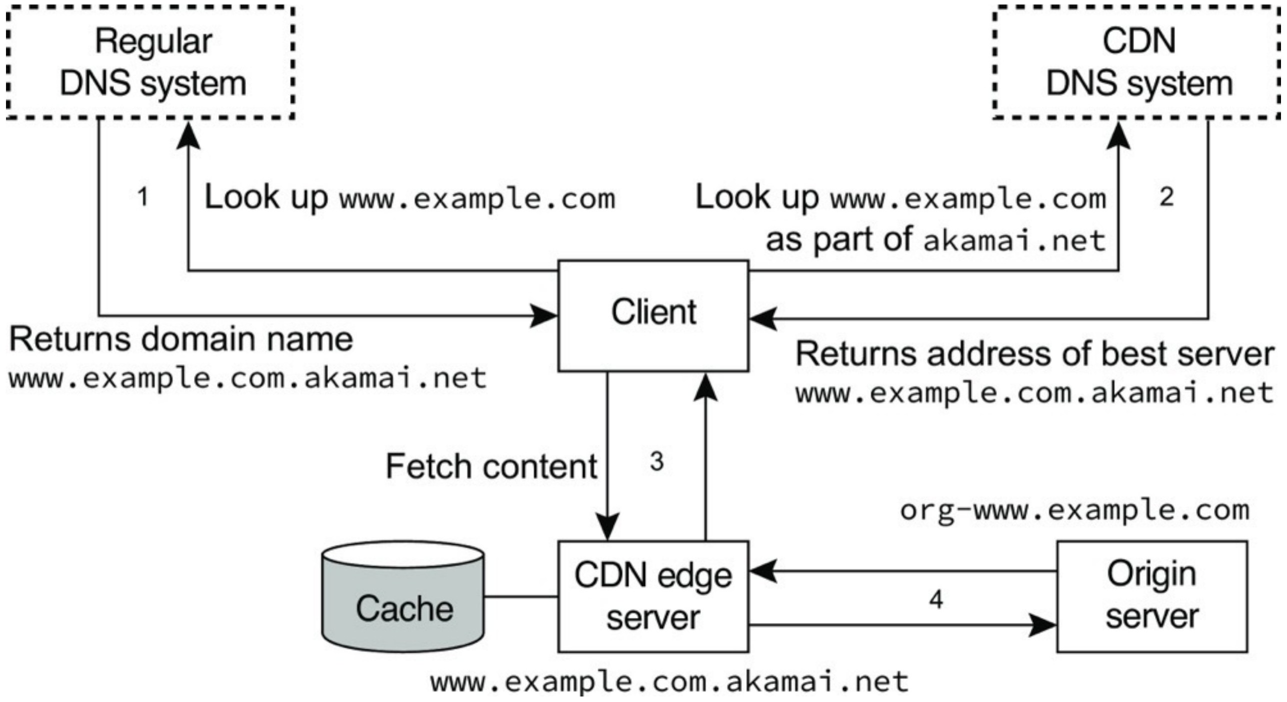
Common approach: use DNS:

1. Client looks up specific service through DNS - client’s IP address is part of request
2. DNS server keeps track of replica servers for the requested service, and returns address of most local server.

Client transparency

To keep client unaware of distribution, let DNS resolver act on behalf of client. Problem is that the resolver may actually be far from local to the actual client.

A simplified version of the Akamai CDN



Regular DNS system
CDN DNS system
1
Look up www.example.com
Look up www.example.com as part of akamai.net
2
Client
Returns domain name www.example.com.akamai.net
Returns address of best server www.example.com.akamai.net
Fetch content
3
org-www.example.com
Cache
CDN edge server
4
Origin server
www.example.com.akamai.net

Important note

The cache is often sophisticated enough to hold more than just passive data. Much of the application code of the origin server can be moved to the cache as well.

Communication in distributed systems

Transport Layer

Important

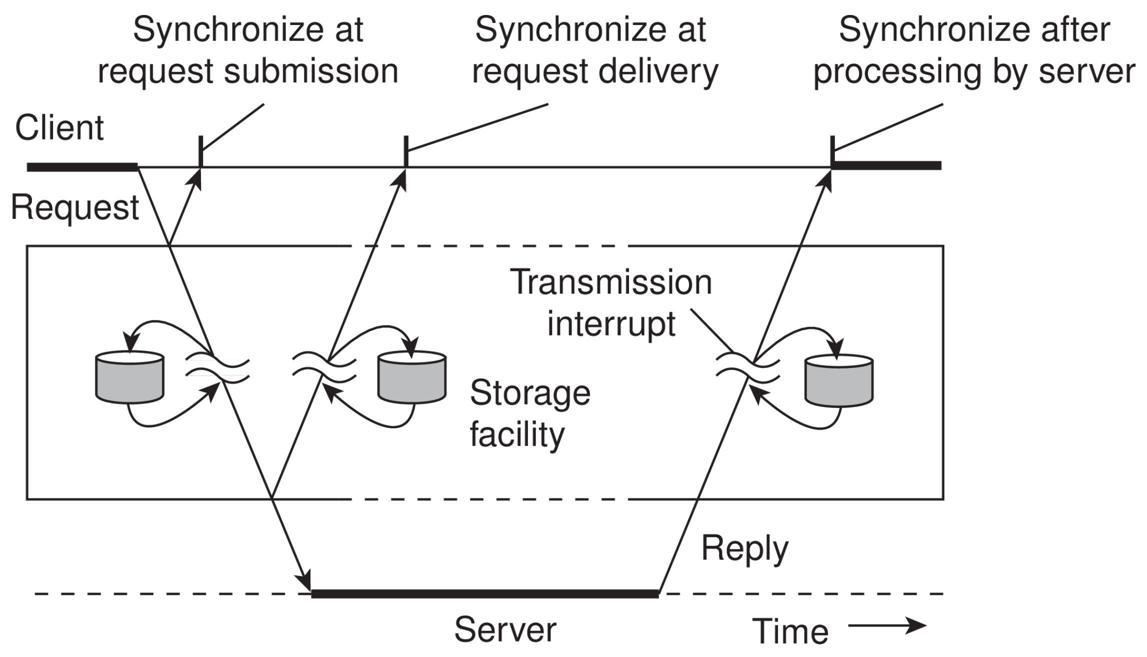
The transport layer provides the actual communication facilities for most distributed systems.

Standard Internet protocols

- TCP: connection-oriented, reliable, stream-oriented communication
- UDP: unreliable (best-effort) datagram communication

Types of communication

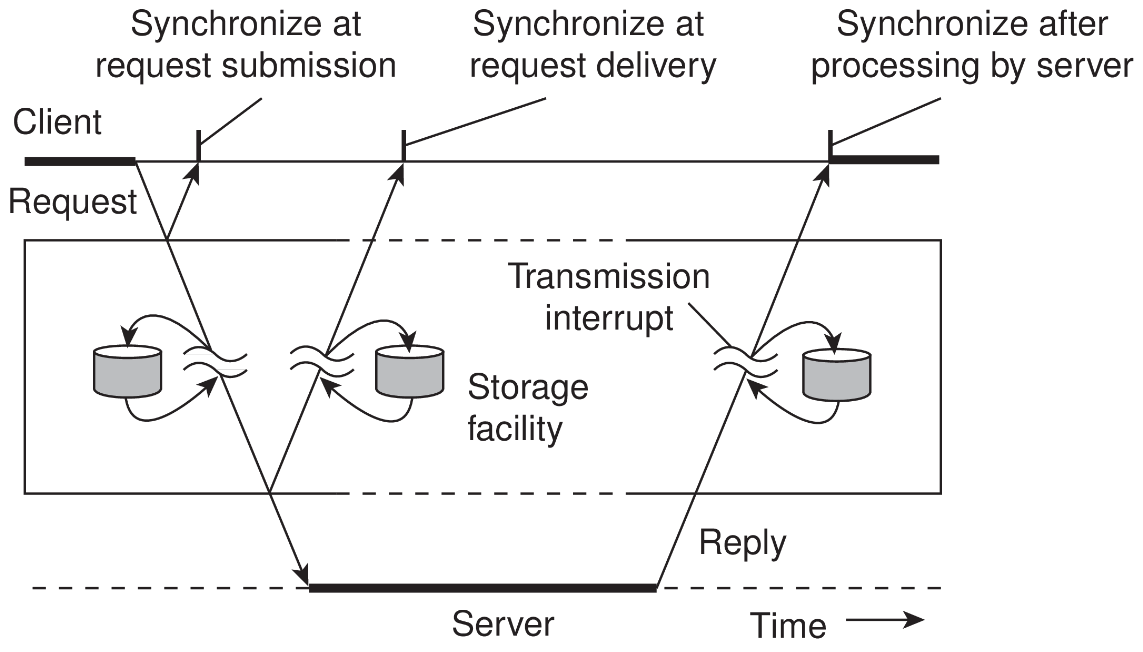
Distinguish...


Synchronize at request submission
Synchronize at request delivery
Synchronize after processing by server
Client
Request
Transmission interrupt
Storage facility
Reply
Server
Time

- Transient versus persistent communication
- Asynchronous versus synchronous communication

Types of communication

Transient versus persistent

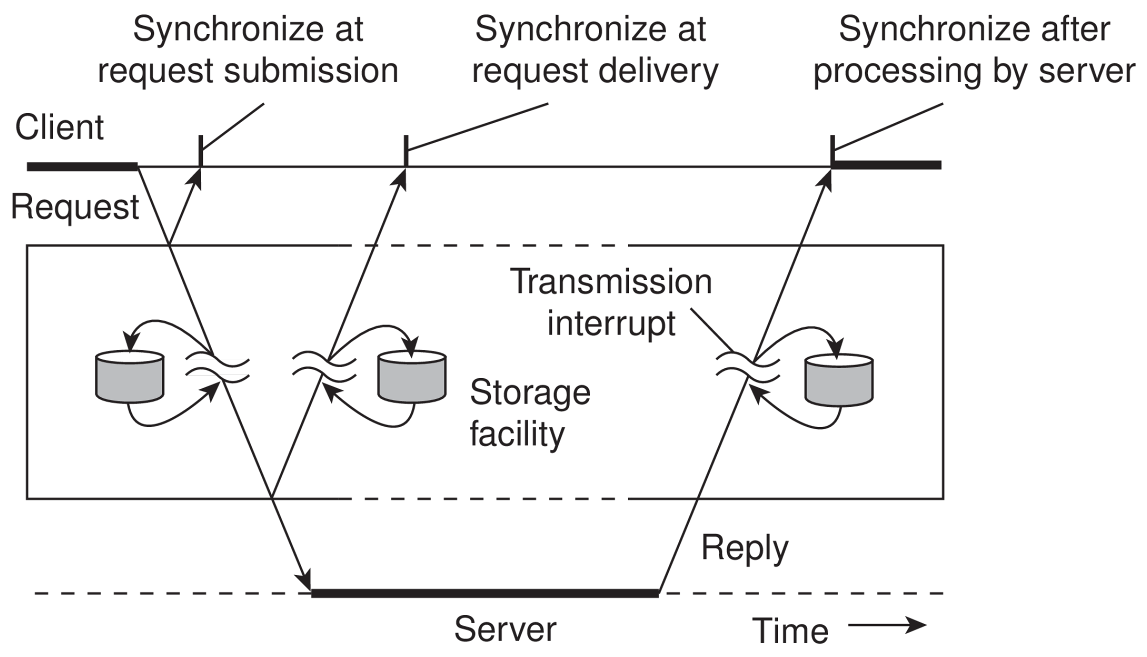


Synchronize at request submission
Synchronize at request delivery
Synchronize after processing by server
Client
Request
Transmission interrupt
Storage facility
Reply
Server
Time

- Transient communication: Comm. server discards message when it cannot be delivered at the next server, or at the receiver.
- Persistent communication: A message is stored at a communication server as long as it takes to deliver it.

Types of communication

Places for synchronization


Synchronize at request submission
Synchronize at request delivery
Synchronize after processing by server
Client
Request
Transmission interrupt
Storage facility
Reply
Server
Time

- At request submission
- At request delivery
- After request processing

Client/Server

Some observations

Client/Server computing is generally based on a model of transient synchronous communication:

- Client and server have to be active at the time of communication
- Client issues request and blocks until it receives reply
- Server essentially waits only for incoming requests, and subsequently processes them

Drawbacks synchronous communication

- Client cannot do any other work while waiting for reply
- Failures have to be handled immediately: the client is waiting
- The model may simply not be appropriate (mail, news)

Messaging

Message-oriented middleware

Aims at high-level persistent asynchronous communication:

- Processes send each other messages, which are queued
- Sender need not wait for immediate reply, but can do other things
- Middleware often ensures fault tolerance

Remote procedure call (RPC)

- Used to call procedures located on other machines
- Transparent to the program/programmer—looks like a local procedure
- Utilizes stubs (client and server) that the procedures are passed to/from
- Parameters marshaling is the passing of parameters, which is easier said than done due to differences between machines/programs

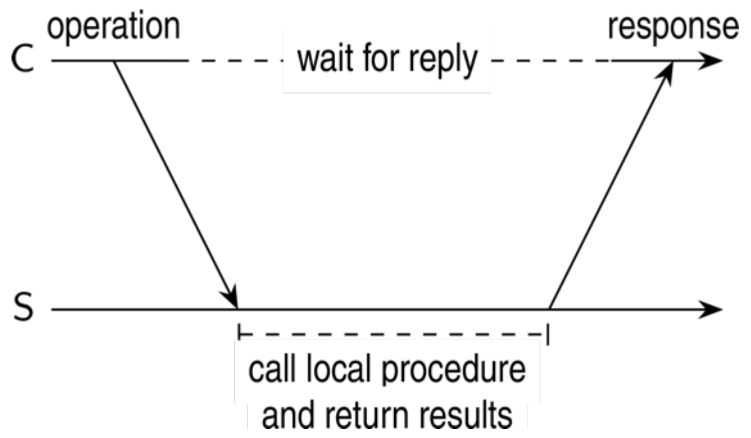
Basic RPC operation

Observations

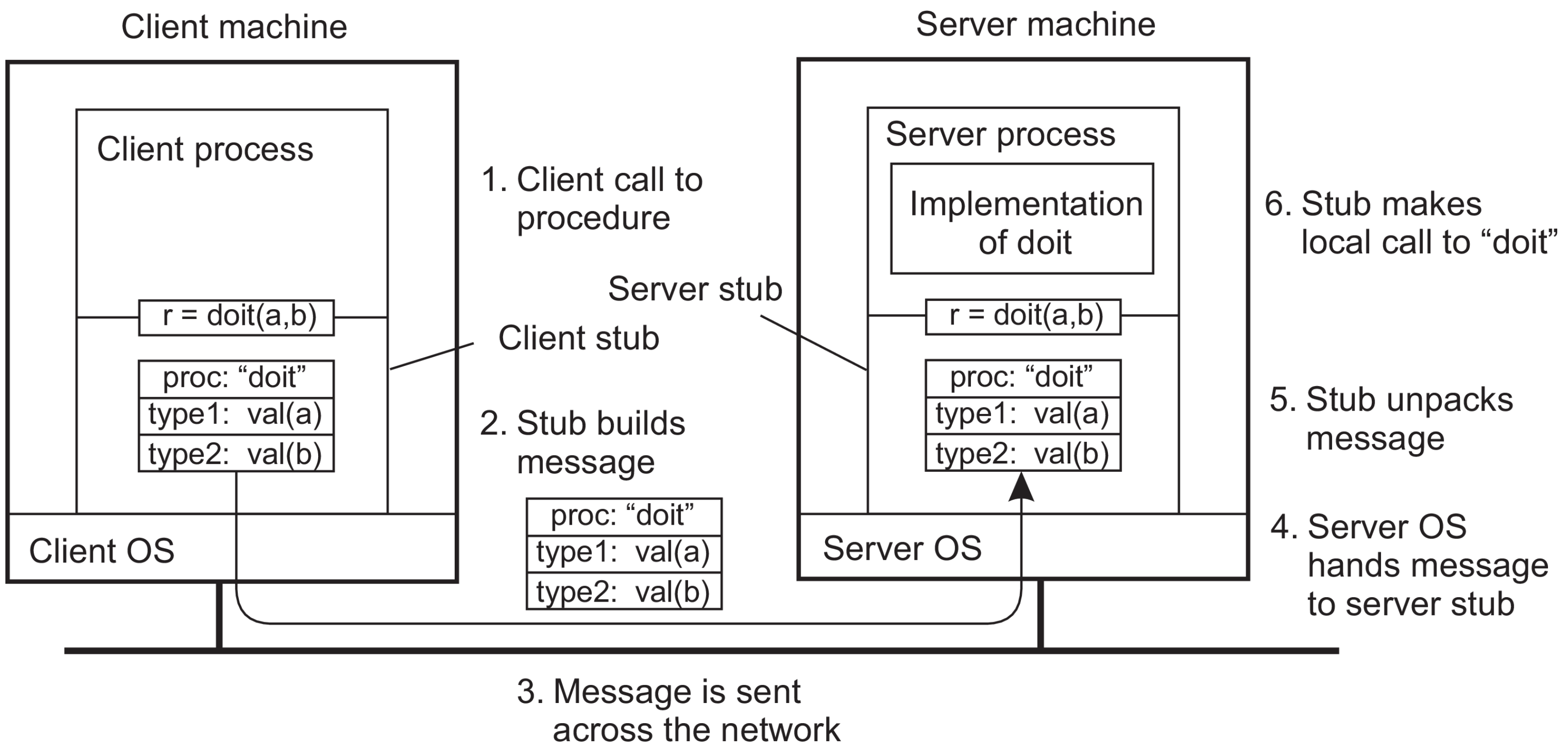
- Application developers are familiar with simple procedure model
- Well-engineered procedures operate in isolation (black box)
- There is no fundamental reason not to execute procedures on separate machine

Conclusion

Communication between caller & callee can be hidden by using procedure-call mechanism.



Basic RPC operation



1. Client procedure calls client stub.
2. Stub builds message; calls local OS.
3. OS sends message to remote OS.
4. Remote OS gives message to stub.
5. Stub unpacks parameters; calls server.
6. Server does local call; returns result to stub.
7. Stub builds message; calls OS.
8. OS sends message to client's OS.
9. Client's OS gives message to stub.
10. Client stub unpacks result; returns to client.

RPC: Parameter passing

There’s more than just wrapping parameters into a message

- Client and server machines may have different data representations (think of byte ordering)
- Wrapping a parameter means transforming a value into a sequence of bytes
- Client and server have to agree on the same encoding:

- How are basic data values represented (integers, floats, characters)
- How are complex data values represented (arrays, unions)

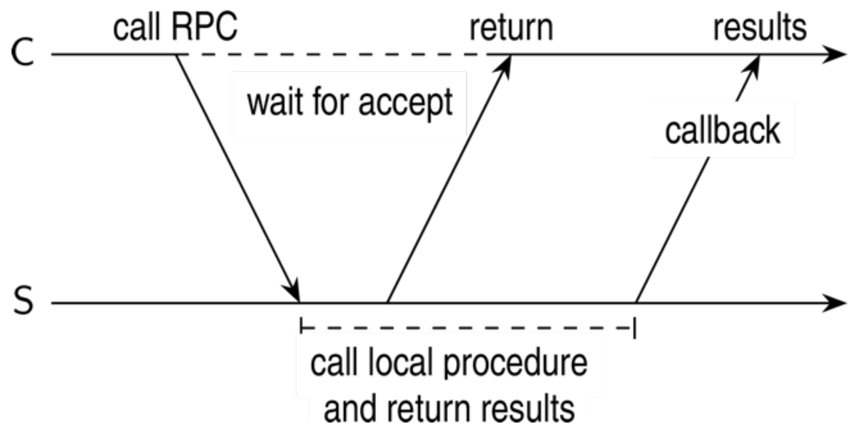
Conclusion

Client and server need to properly interpret messages, transforming them into machine-dependent representations.

Asynchronous RPCs

Essence

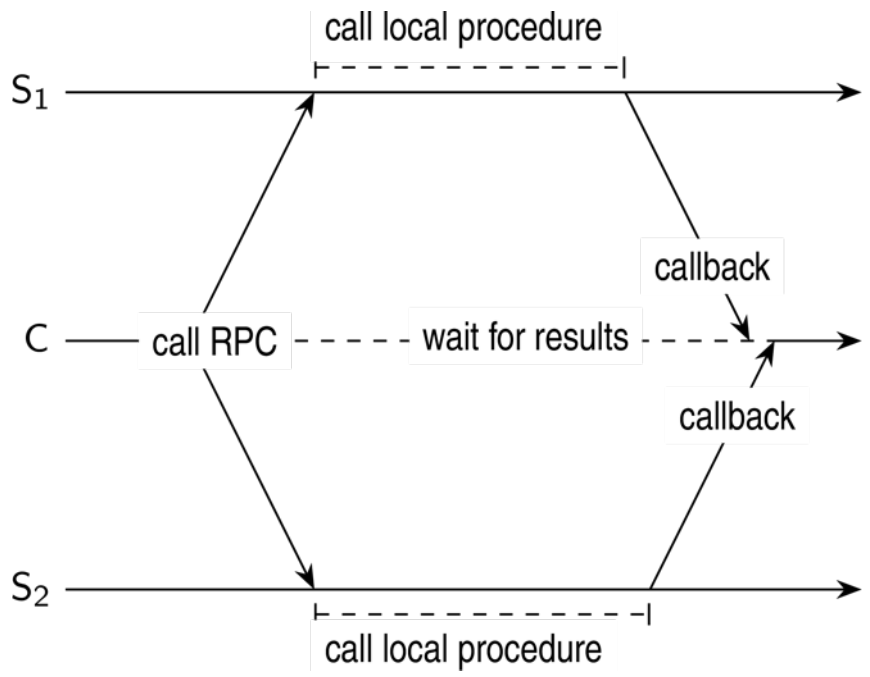
Try to get rid of the strict request-reply behavior, but let the client continue without waiting for an answer from the server.



Sending out multiple RPCs

Essence

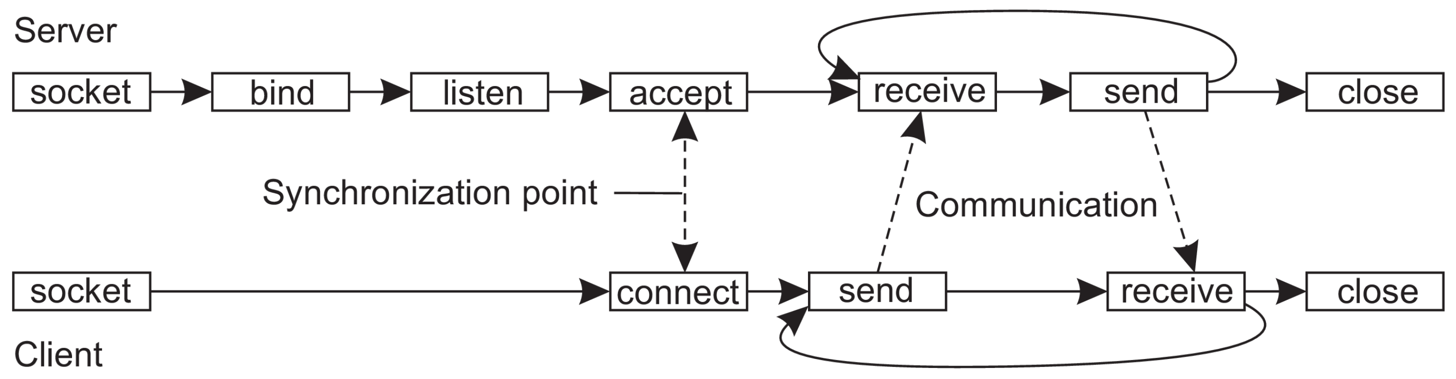
Sending an RPC request to a group of servers.



Transient messaging: sockets

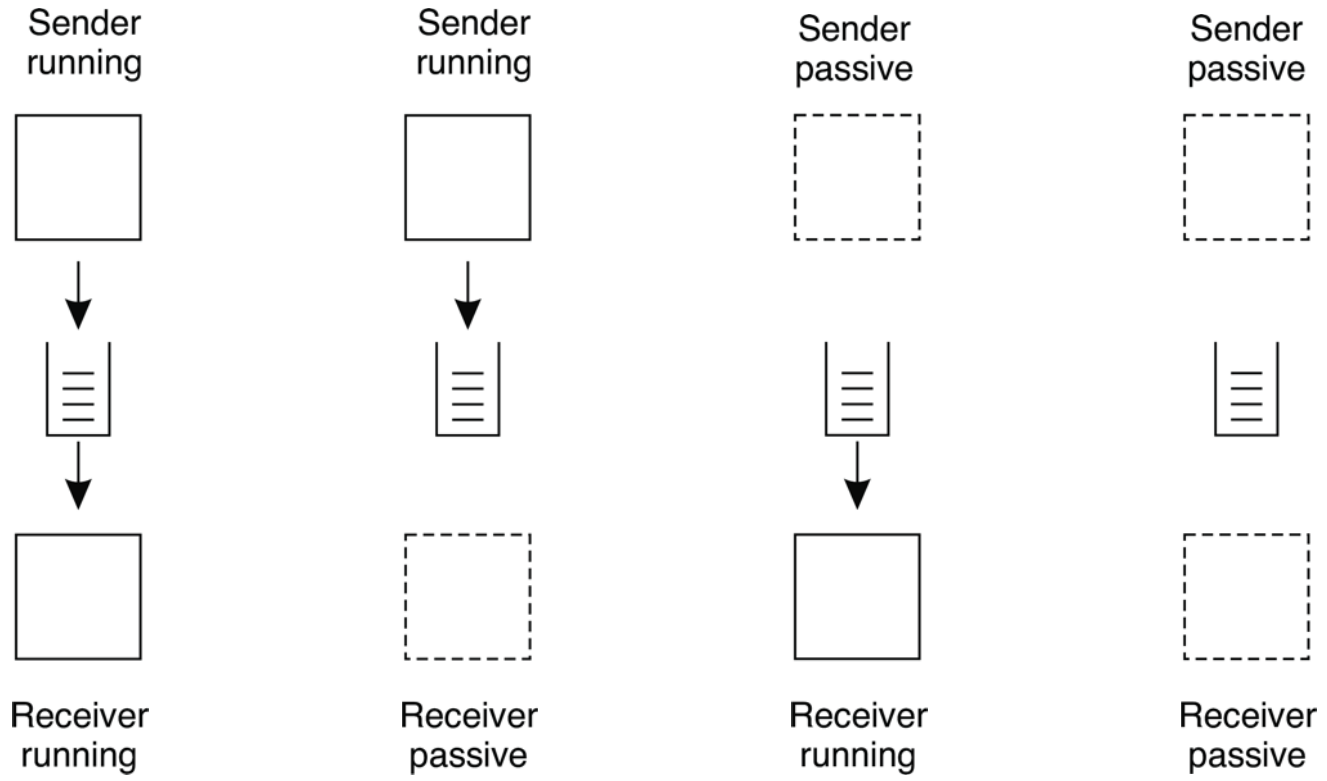
Berkeley socket interface

Operation	Description
`socket`	Create a new communication end point
`bind`	Attach a local address to a socket
`listen`	Tell operating system what the maximum number of pending connection requests should be
`accept`	Block caller until a connection request arrives
`connect`	Actively attempt to establish a connection
`send`	Send some data over the connection
`receive`	Receive some data over the connection
`close`	Release the connection



Queue-based messaging

Four possible combinations



Message-oriented middleware

Essence

Asynchronous persistent communication through support of middleware-level queues. Queues correspond to buffers at communication servers.

Operations

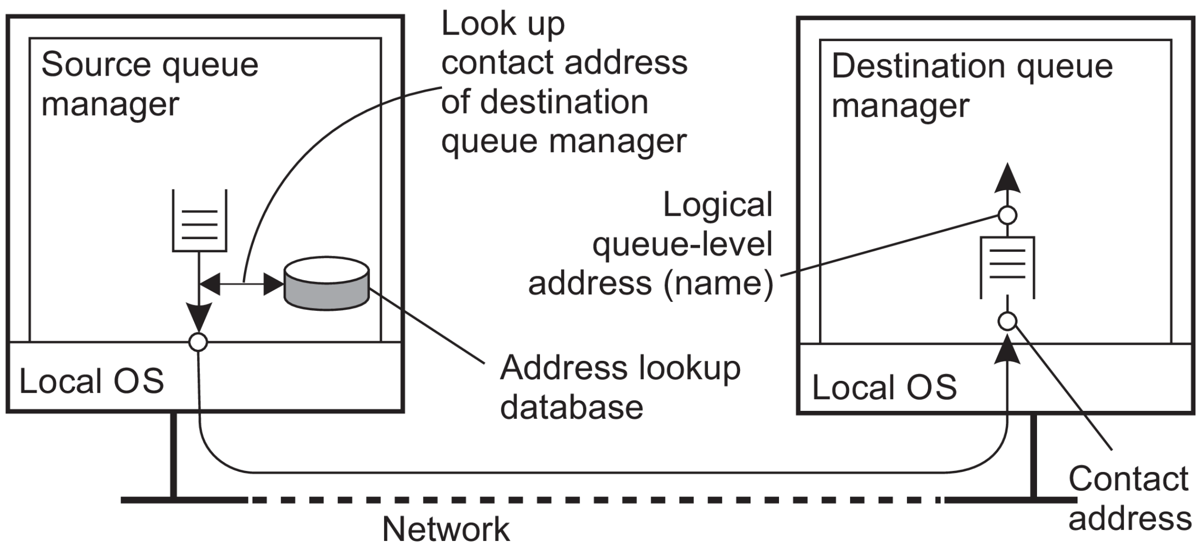
Operation	Description
`PUT`	Append a message to a specified queue
`GET`	Block until the specified queue is nonempty, and remove the first message
`POLL`	Check a specified queue for messages, and remove the first. Never block
`NOTIFY`	Install a handler to be called when a message is put into the specified queue

General model

Queue managers

Queues are managed by queue managers. An application can put messages only into a local queue. Getting a message is possible by extracting it from a local queue only ⇒ queue managers need to route messages.

Routing



Source queue manager
Look up contact address of destination queue manager
Destination queue manager
Logical queue-level address (name)
Local OS
Address lookup database
Local OS
Contact address
Network

Message broker

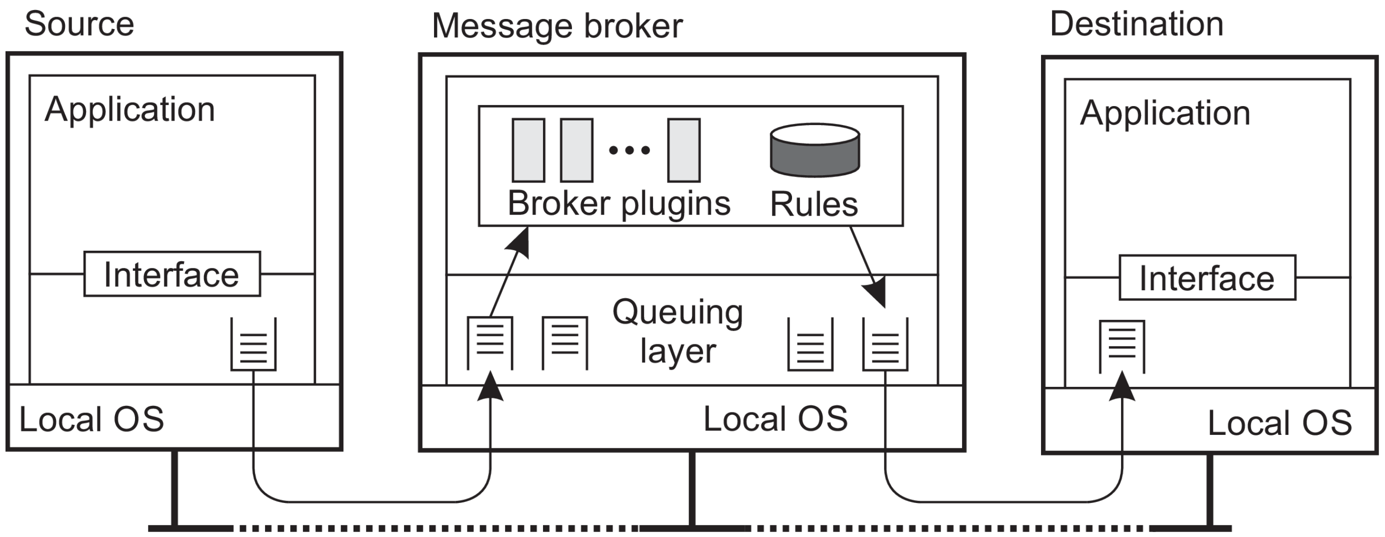
Observation

Message queuing systems assume a common messaging protocol: all applications agree on message format (i.e., structure and data representation)

Broker handles application heterogeneity in an MQ system

- Transforms incoming messages to target format
- Very often acts as an application gateway
- May provide subject-based routing capabilities (i.e., publish-subscribe capabilities)

Message broker: general architecture



➥ **Next lecture:** Coordination and naming

Questions?

Questions/feedback: carl.magnus.bruhner@liu.se

LINKÖPING UNIVERSITY