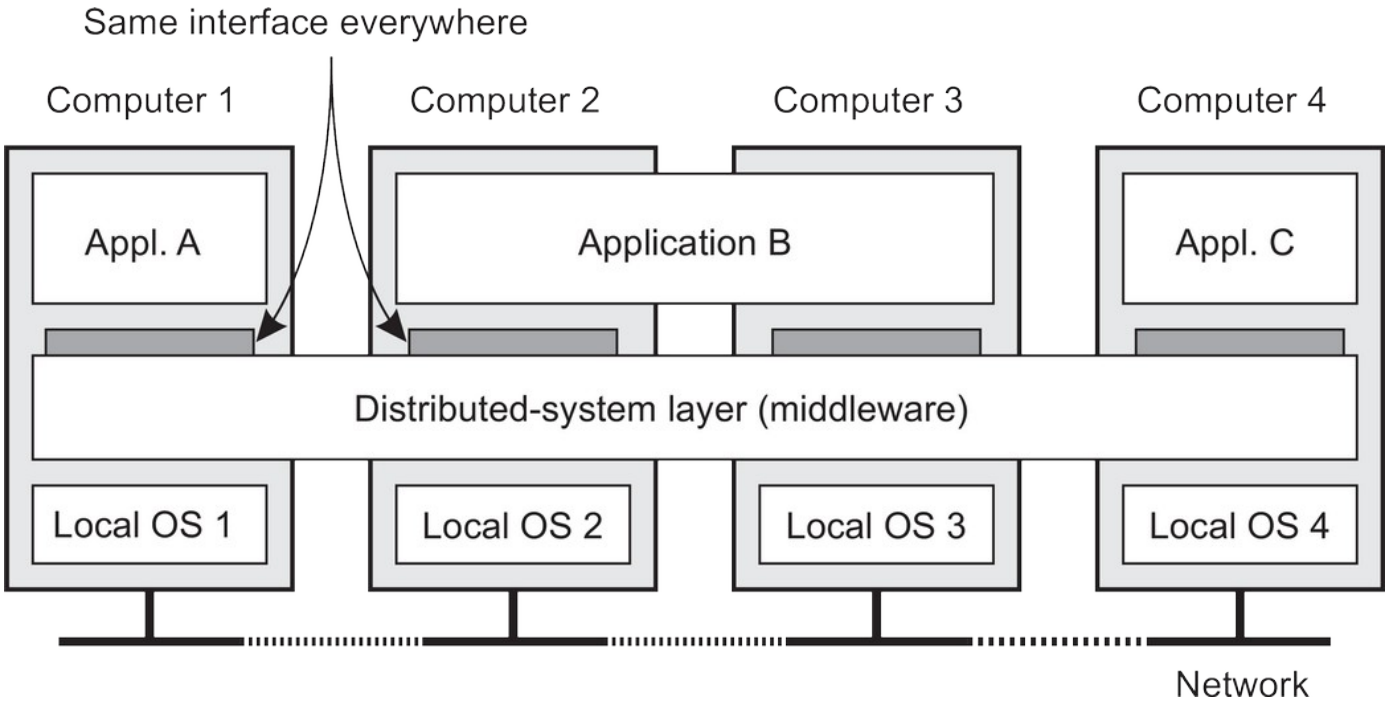
Arquiteturas de Sistemas Distribuídos

Adaptado por Vinícius Hax a partir dos slides “Chapter 2: Architectures” de Tanenbaum e Van Steen

Middleware: the OS of distributed systems

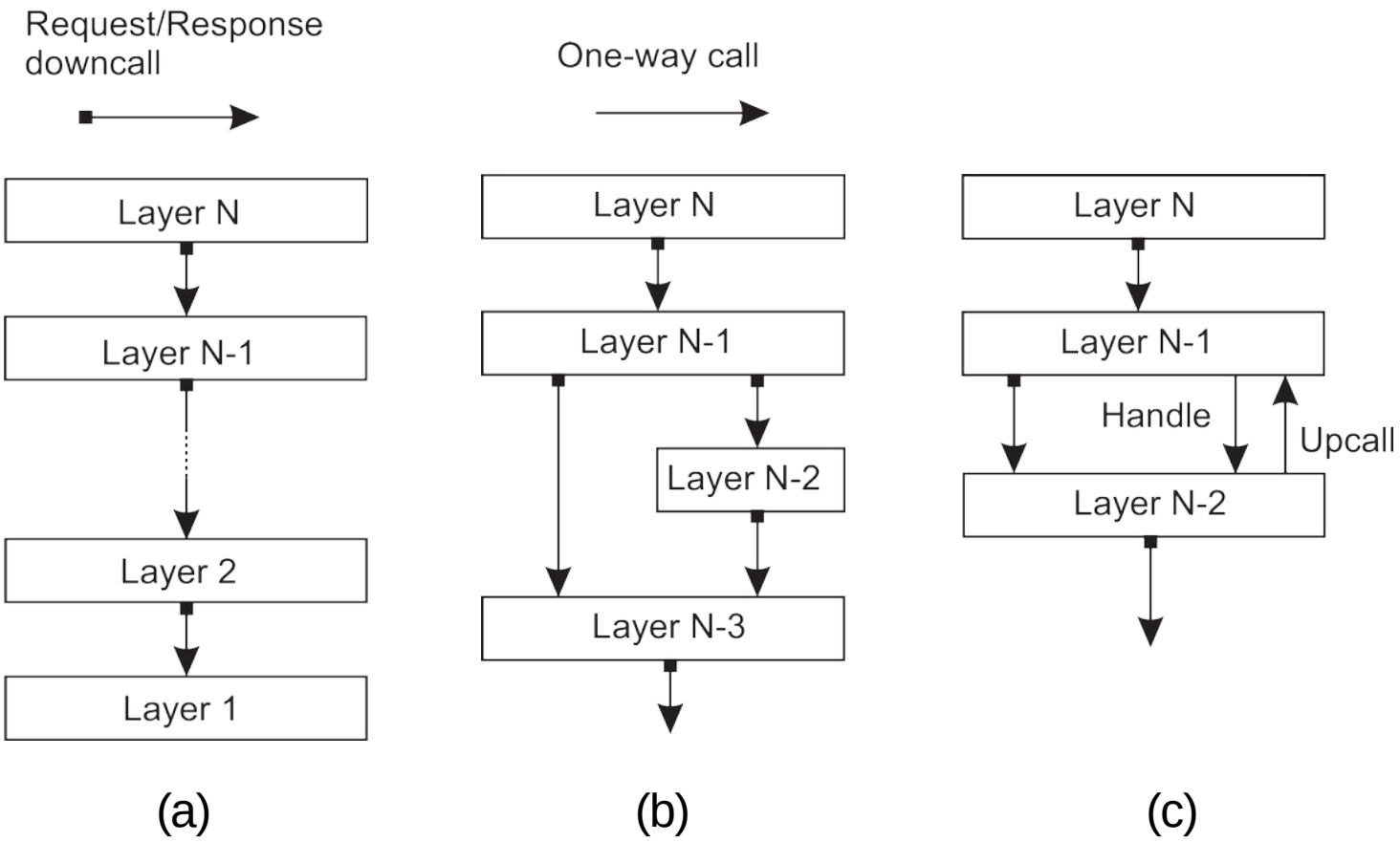


What does it contain?

Commonly used components and functions that need not be implemented by applications separately.

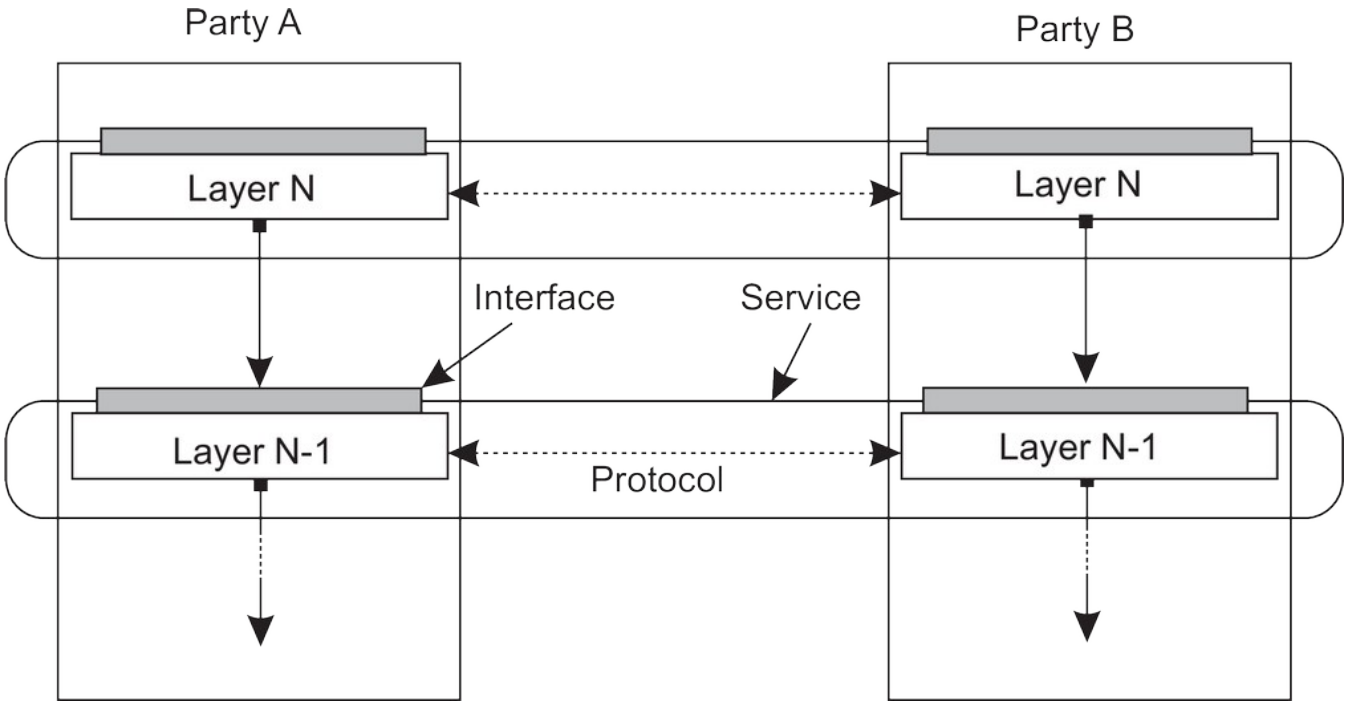
Layered architecture

Different layered organizations



Example: communication protocols

Protocol, service, interface



Two-party communication

Server

```
1  from socket  import *
2
3  s  = socket(AF_INET, SOCK_STREAM)
4  (conn, addr) = s.accept()  # returns new socket and addr. client
5  while True:                          # forever
6    data = conn.recv(1024)     # receive data from client
7    if not data: break          # stop if client stopped
8    msg = data.decode()+"*"  # process the incoming data into a response
9    conn.send(msg.encode())    # return the response
10 conn.close()                  # close the connection
```

Client

```
1  from socket  import *
2
3  s  = socket(AF_INET, SOCK_STREAM)
4  s.connect((HOST, PORT)) # connect to server (block until accepted)
5  msg = "Hello World"          # compose a message
6  s.send(msg.encode())         # send the message
7  data = s.recv(1024)          # receive the response
8  print(data.decode())         # print the result
9  s.close()                    # close the connection
```

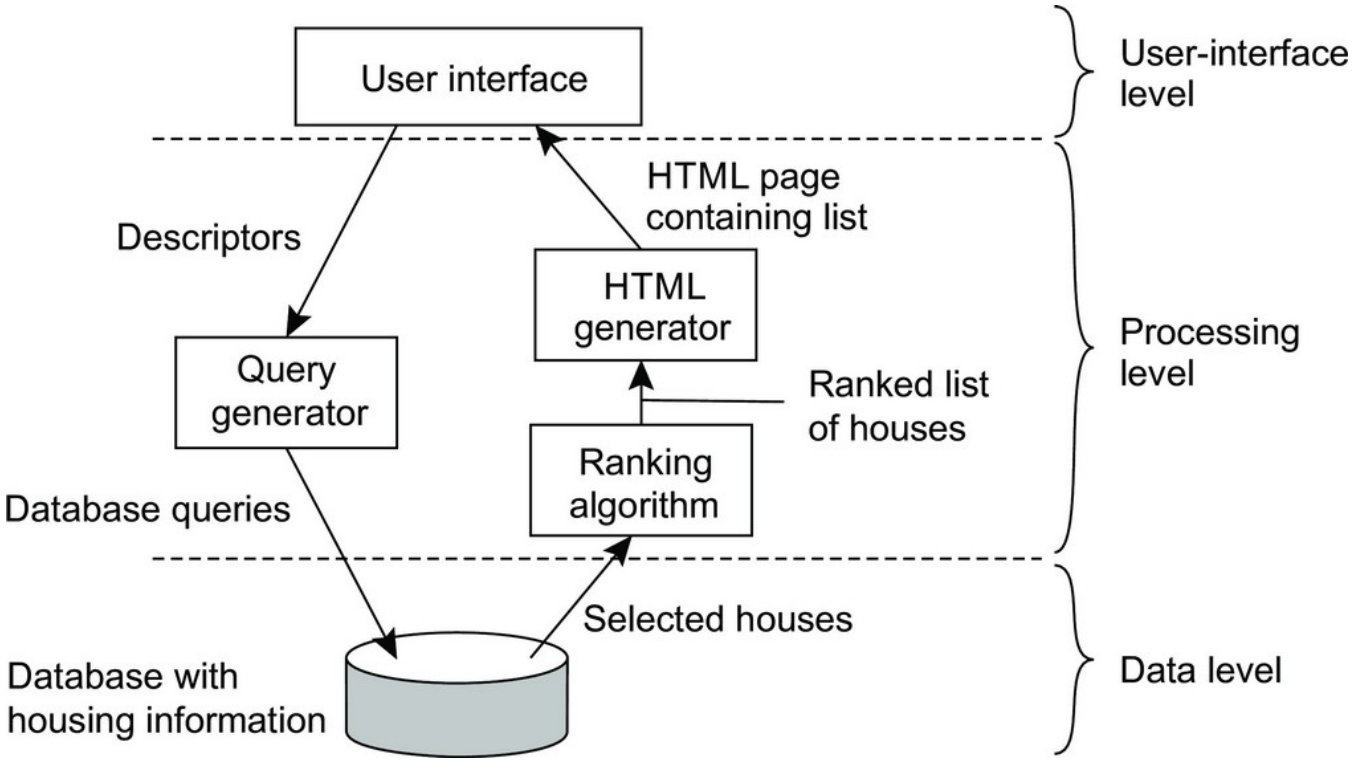
Application Layering

Traditional three-layered view

- Application-interface layer contains units for interfacing to users or external applications
- Processing layer contains the functions of an application, i.e., without specific data
- Data layer contains the data that a client wants to manipulate through the application components

Application Layering

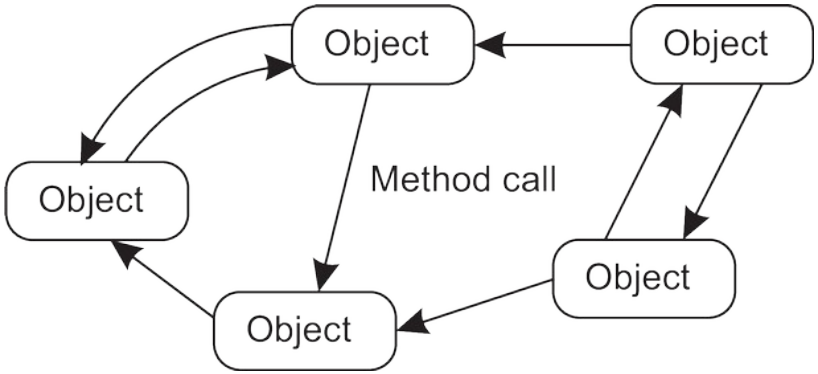
Example: a simple search engine

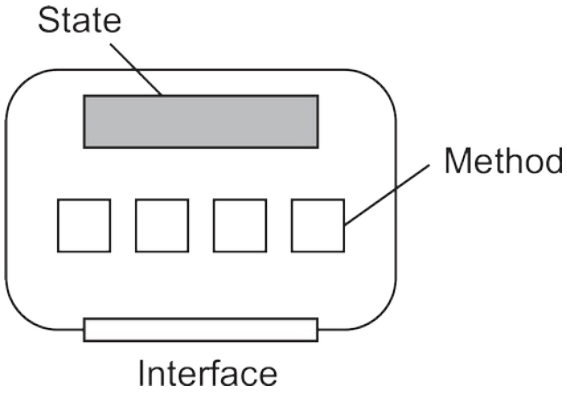


Object-based style

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.

RESTful architectures

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

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

On interfaces

Issue

Many people like RESTful approaches because the interface to a service is so simple. The catch is that much needs to be done in the parameter space.

Amazon S3 SOAP interface

Bucket operations	Object operations
ListAllMyBuckets	PutObjectInline
CreateBucket	PutObject
DeleteBucket	CopyObject
ListBucket	GetObject
GetBucketAccessControlPolicy	GetObjectExtended
SetBucketAccessControlPolicy	DeleteObject
GetBucketLoggingStatus	GetObjectAccessControlPolicy
SetBucketLoggingStatus	SetObjectAccessControlPolicy

On interfaces

Simplifications

Assume an interface `bucket` offering an operation `create`, requiring an input string such as `mybucket`, for creating a bucket “mybucket.”

On interfaces

Simplifications

Assume an interface `bucket` offering an operation `create`, requiring an input string such as `mybucket`, for creating a bucket “mybucket.”

SOAP

```
import bucket
bucket.create("mybucket")
```

On interfaces

Simplifications

Assume an interface `bucket` offering an operation `create`, requiring an input string such as `mybucket`, for creating a bucket “mybucket.”

SOAP

```
import bucket
bucket.create("mybucket")
```

RESTful

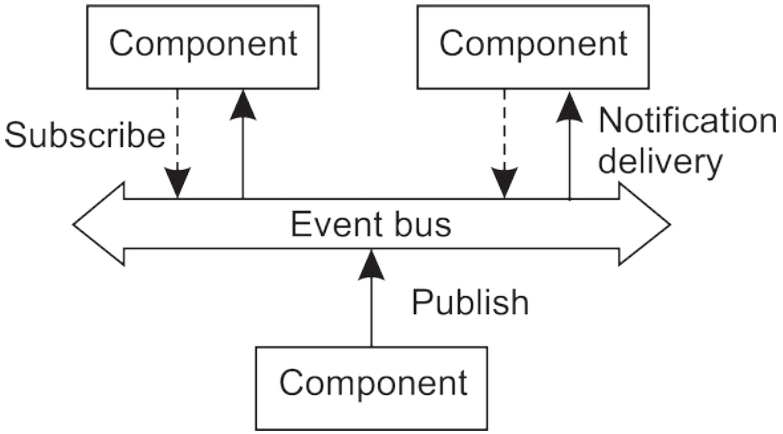
```
PUT
"https://mybucket.s3.amazo
nsws.com/"
```

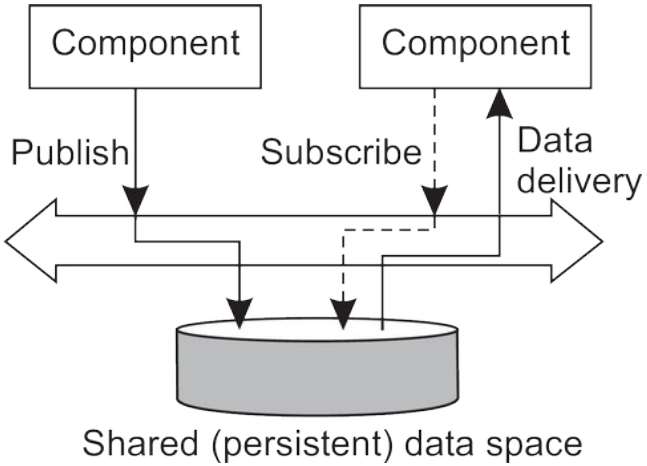
Event based

Temporal and coupling

	Temporally coupled	Temporally decoupled
Referentially coupled	Direct	Mailbox
Referentially decoupled	Event-based	Shared data space

Event-based and Shared data space

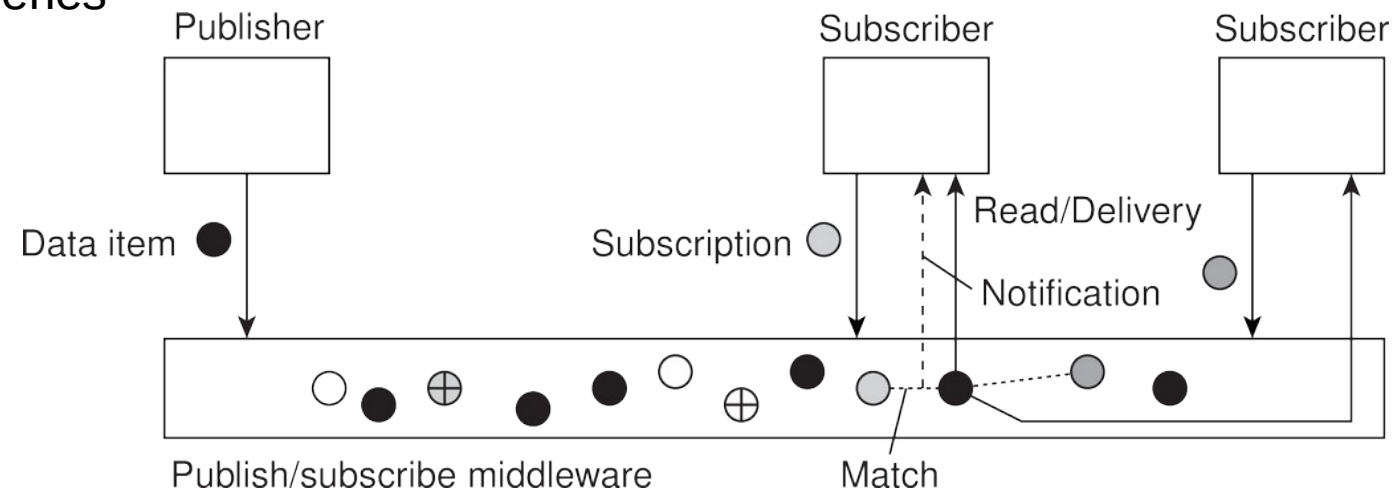




Publish and subscribe

Issue: how to match events?

- Assume events are described by (attribute,value) pairs
- topic-based subscription: specify a "attribute = value" series
- content-based subscription: specify a "attribute ∈ range" series

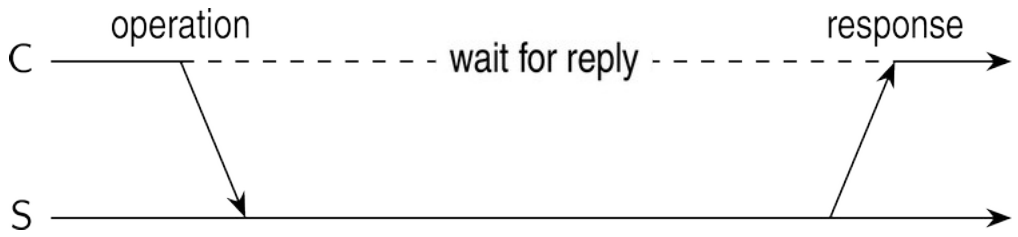


Centralized system architectures

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

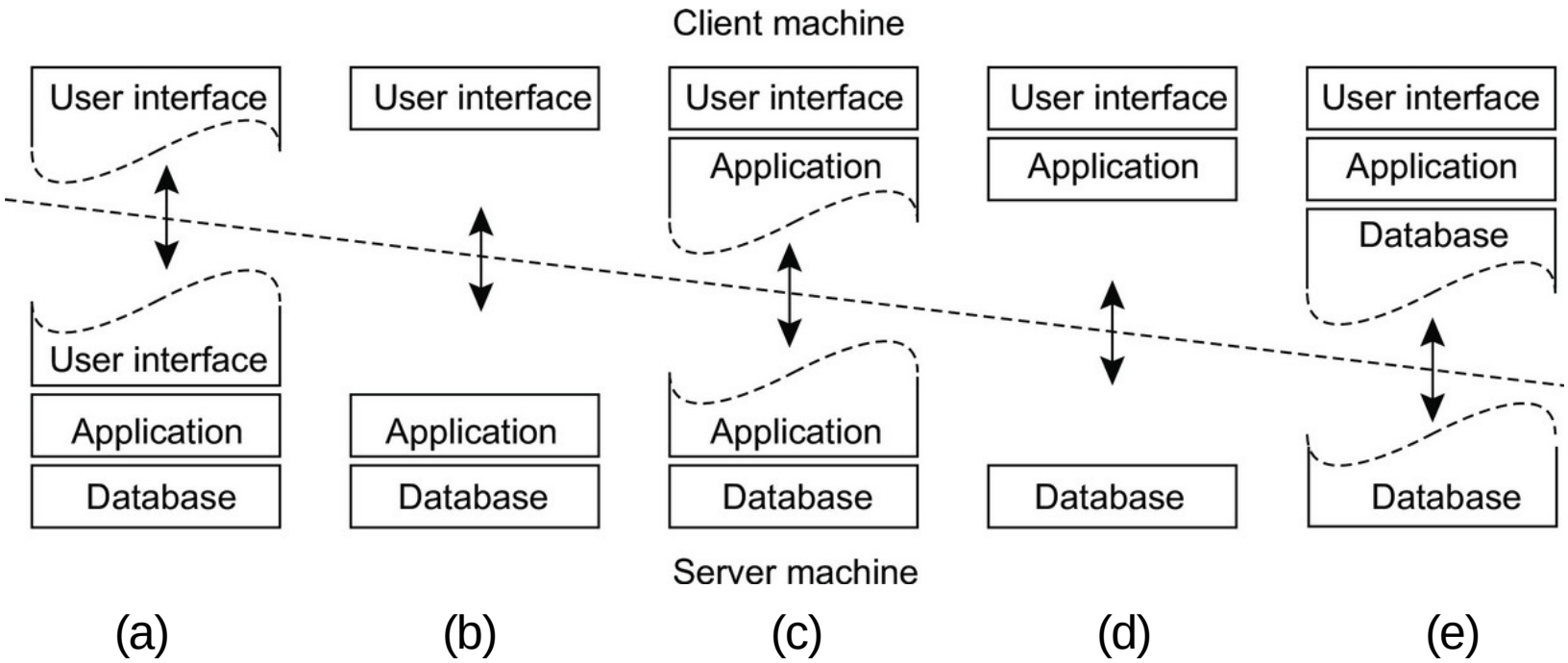


Multi-tiered centralized system architectures

Some traditional organizations

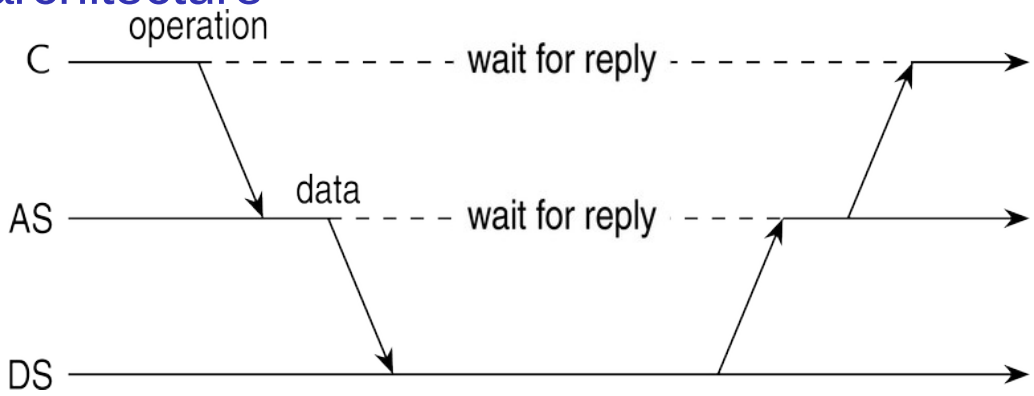
- Single-tiered: dumb terminal/mainframe configuration
- Two-tiered: client/single server configuration
- Three-tiered: each layer on separate machine

Traditional two-tiered configurations



Being client and server at the same time

Three-tiered architecture

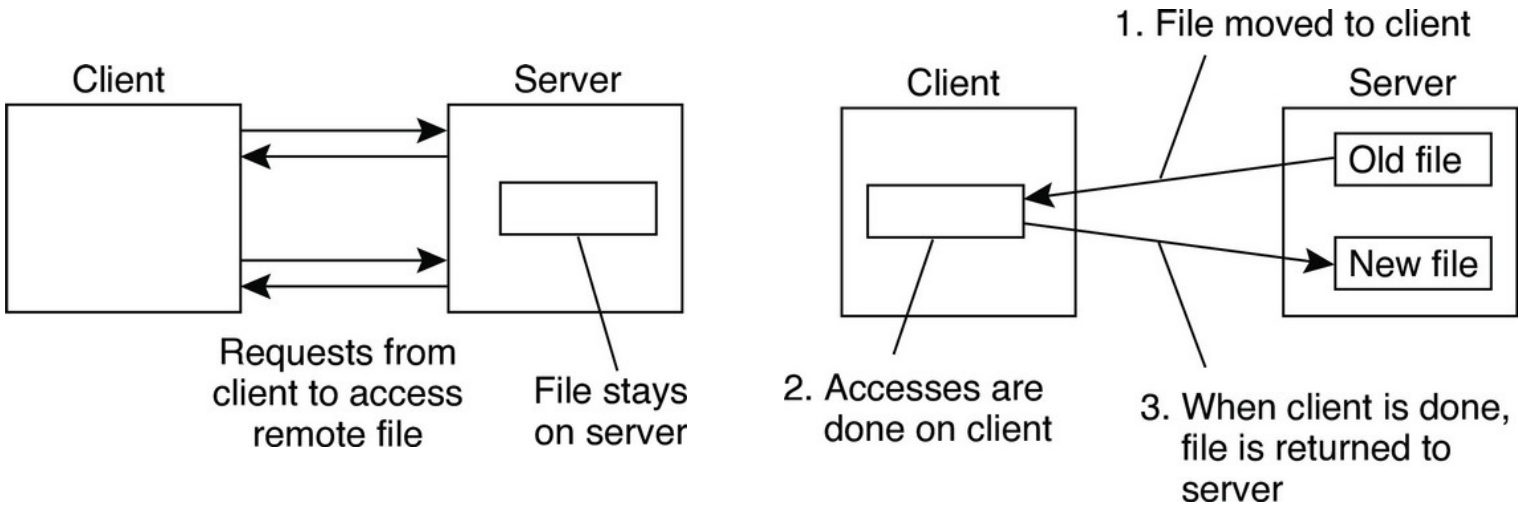


Example: The Network File System

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



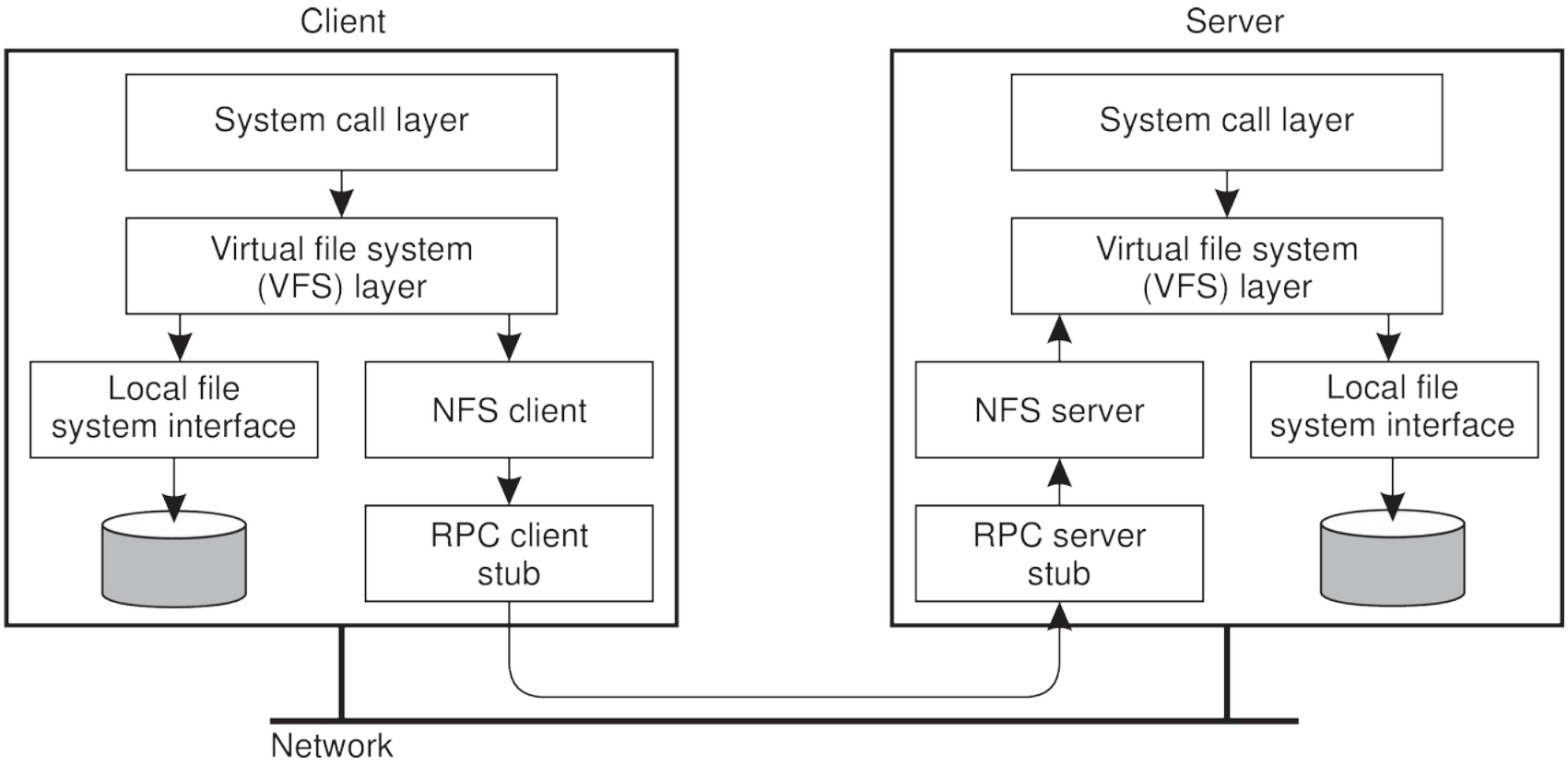
Remote access

Upload/download

Note

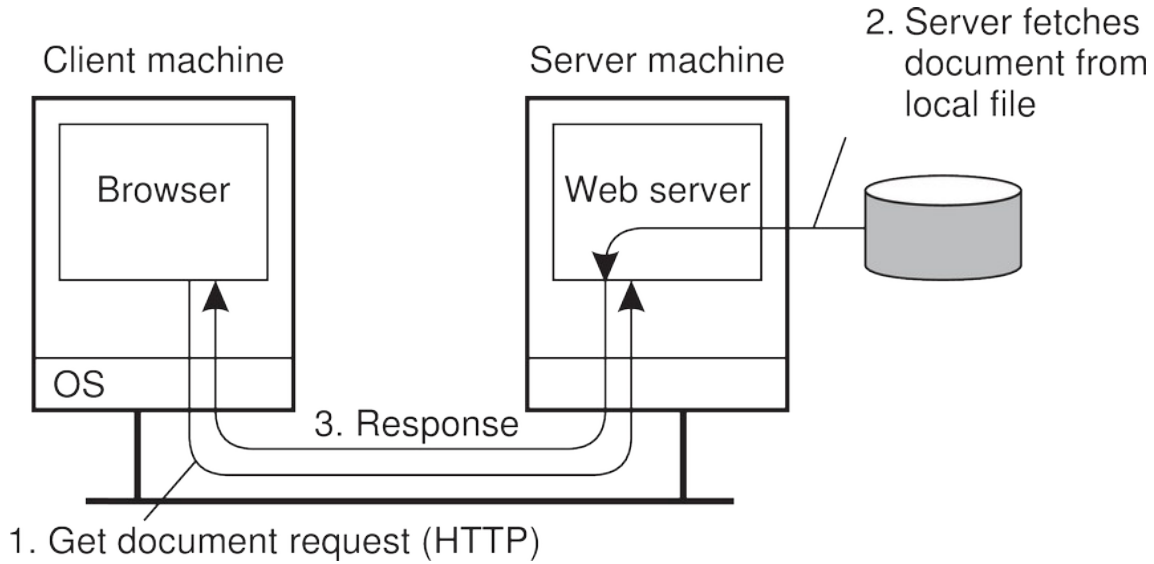
FTP is a typical upload/download model. The same can be said for systems like Dropbox.

NFS architecture



Example: Simple Web servers

Back in the old days...

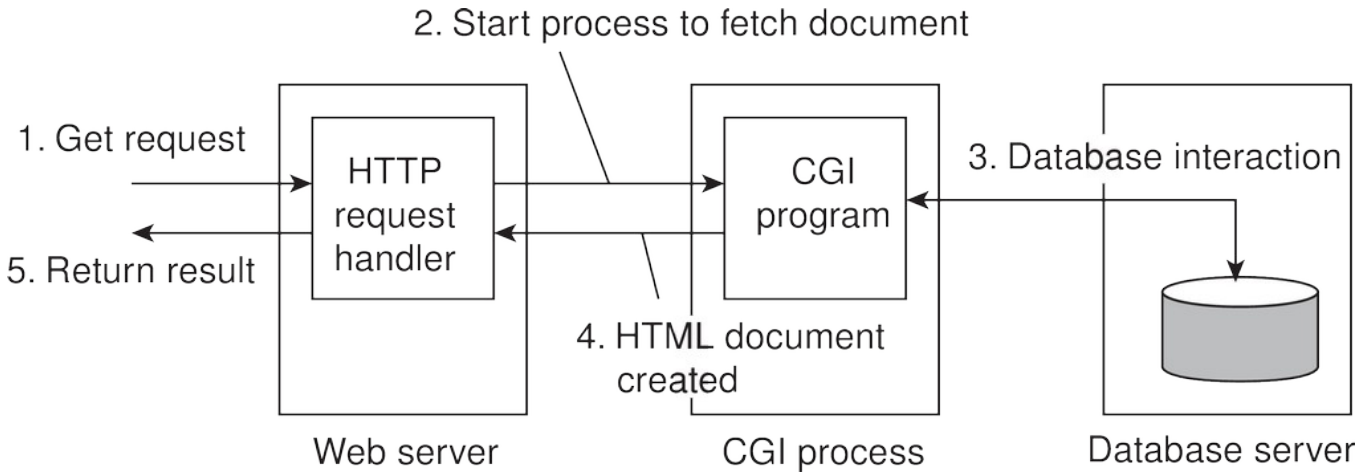


...life was simple:

- 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

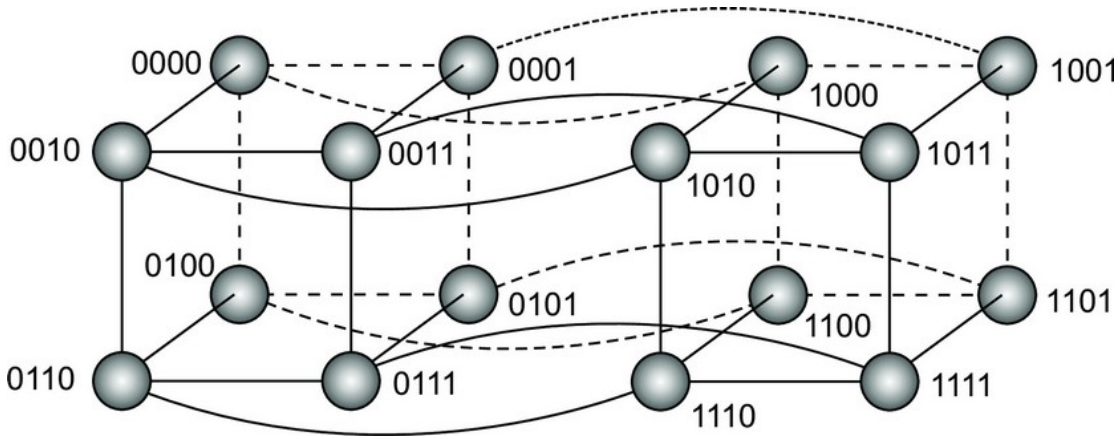
Essence

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

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

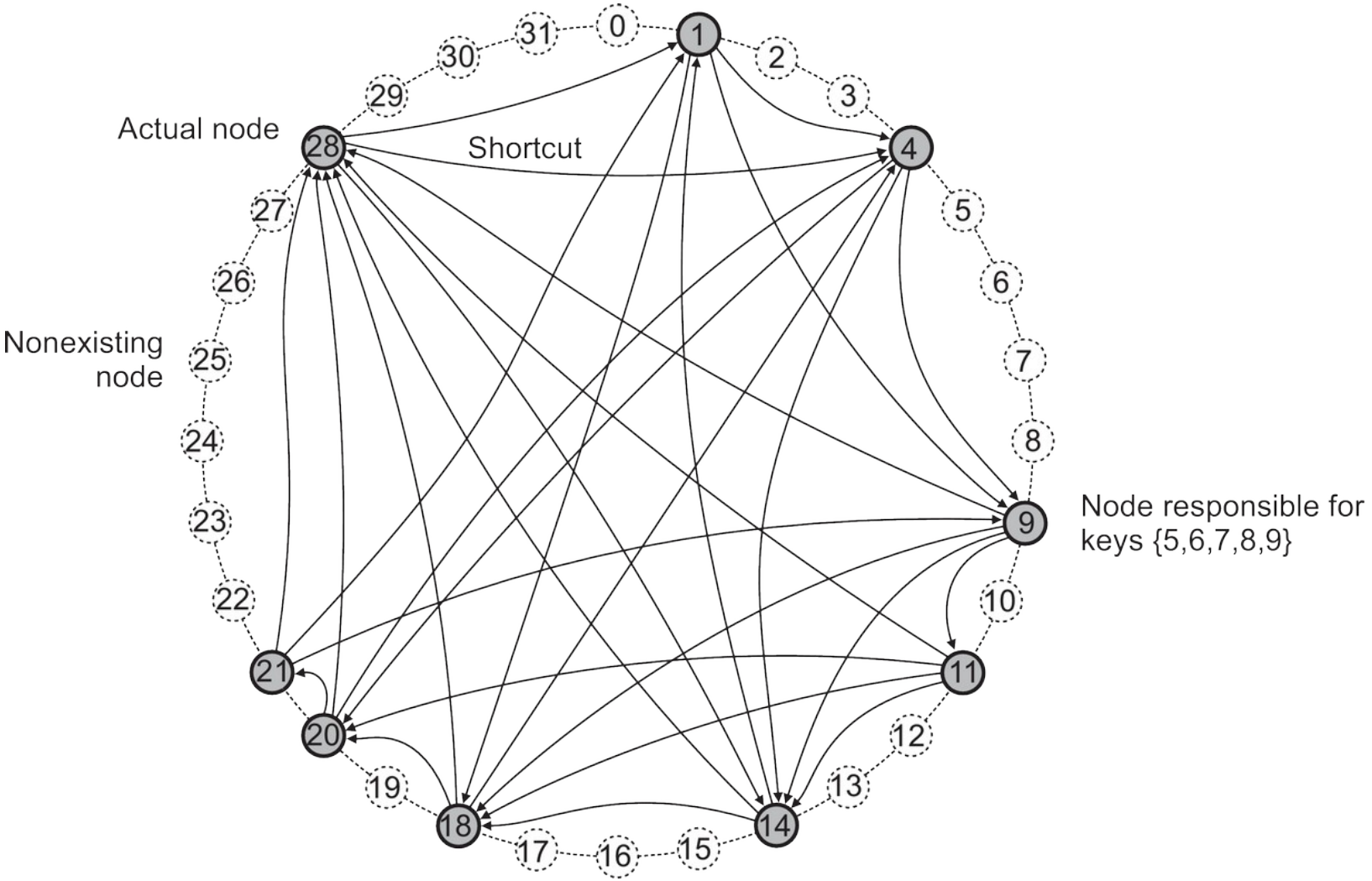
P2P system now responsible for storing (*key*,*value*) pairs.

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 .

Example: Chord



lookup(3)@9 : 28 → 1 → 4

Unstructured P2P

Essence

Each node maintains an ad hoc list of neighbors.

Searching

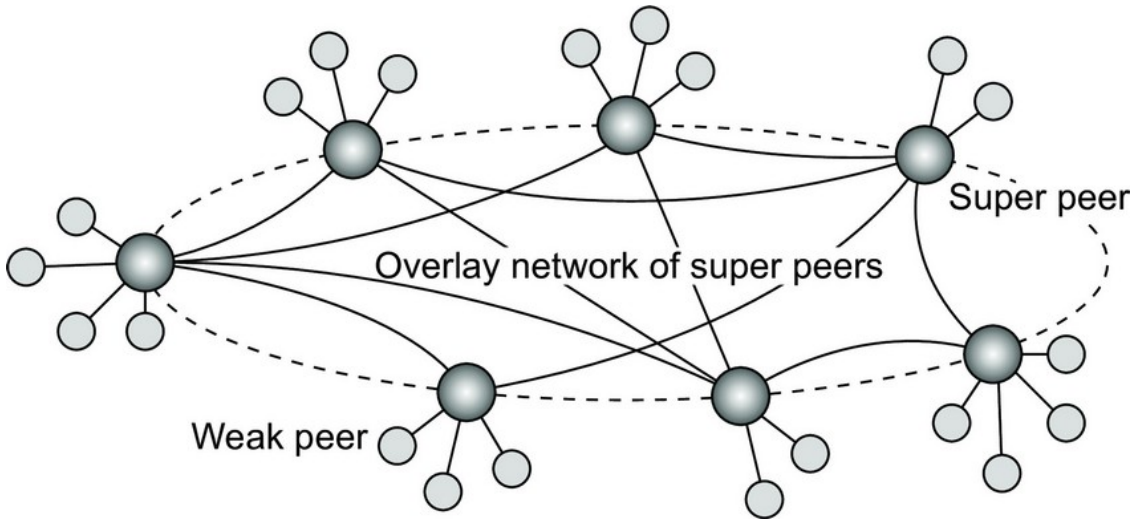
- Flooding: issuing node *u* passes request for *d* to all neighbors. Request is ignored when receiving node had seen it before. Otherwise, *v* searches locally for *d* (recursively). May be limited by a Time-To-Live: a maximum number of hops.
- Random walk: issuing node *u* passes request for *d* to 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:

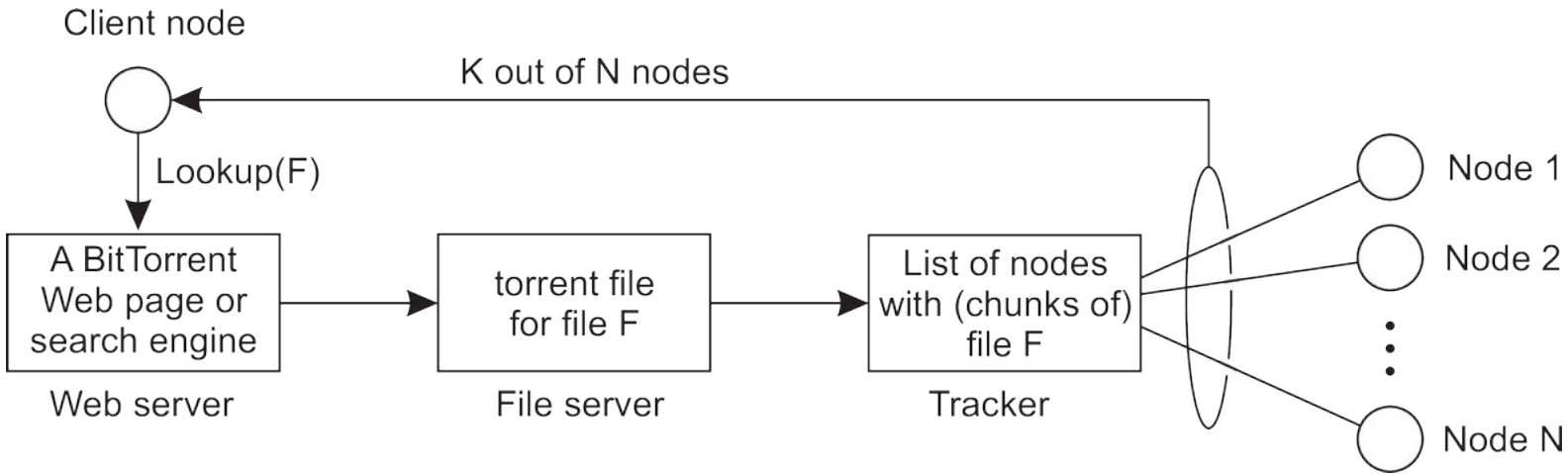
- When searching in unstructured P2P systems, having index servers improves performance



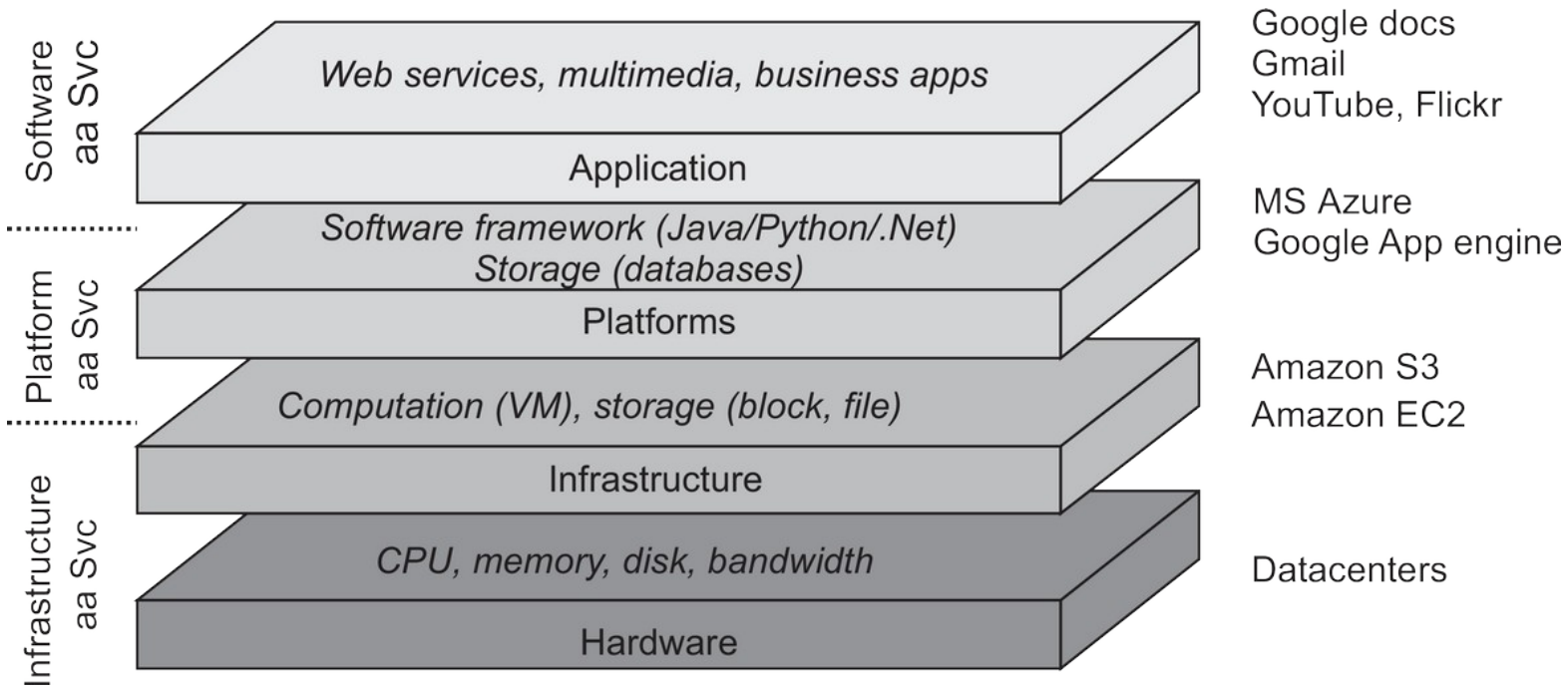
Collaboration: The BitTorrent case

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



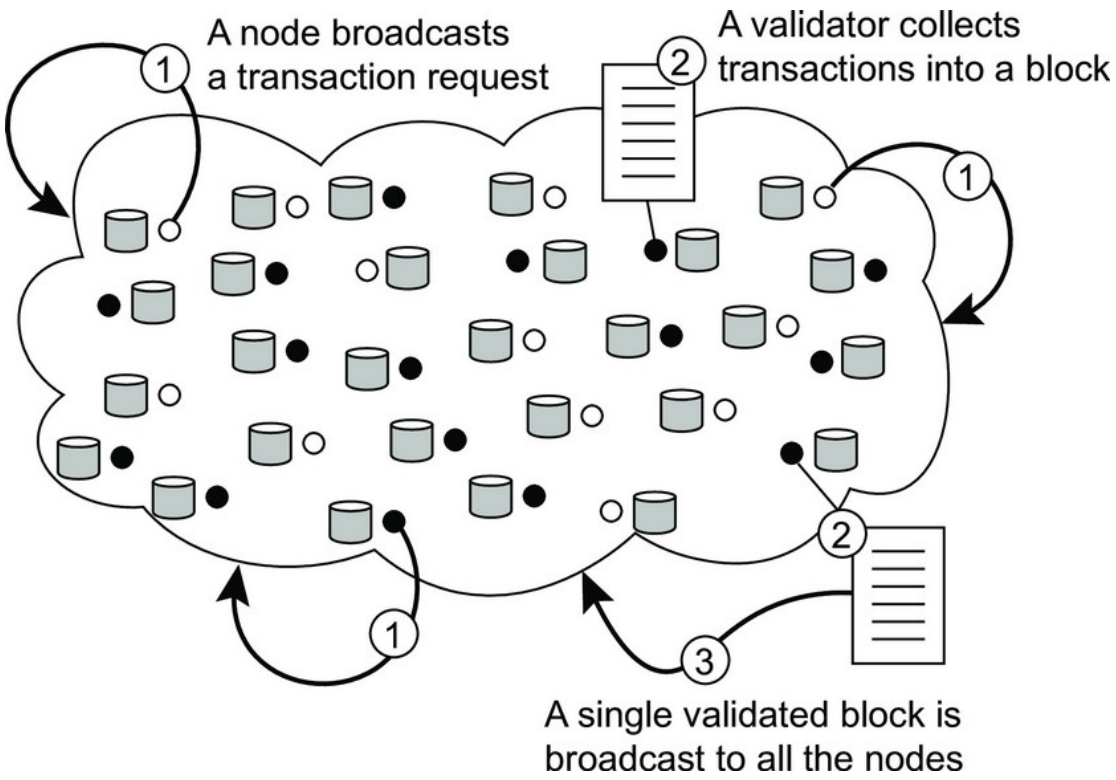
Cloud computing

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.

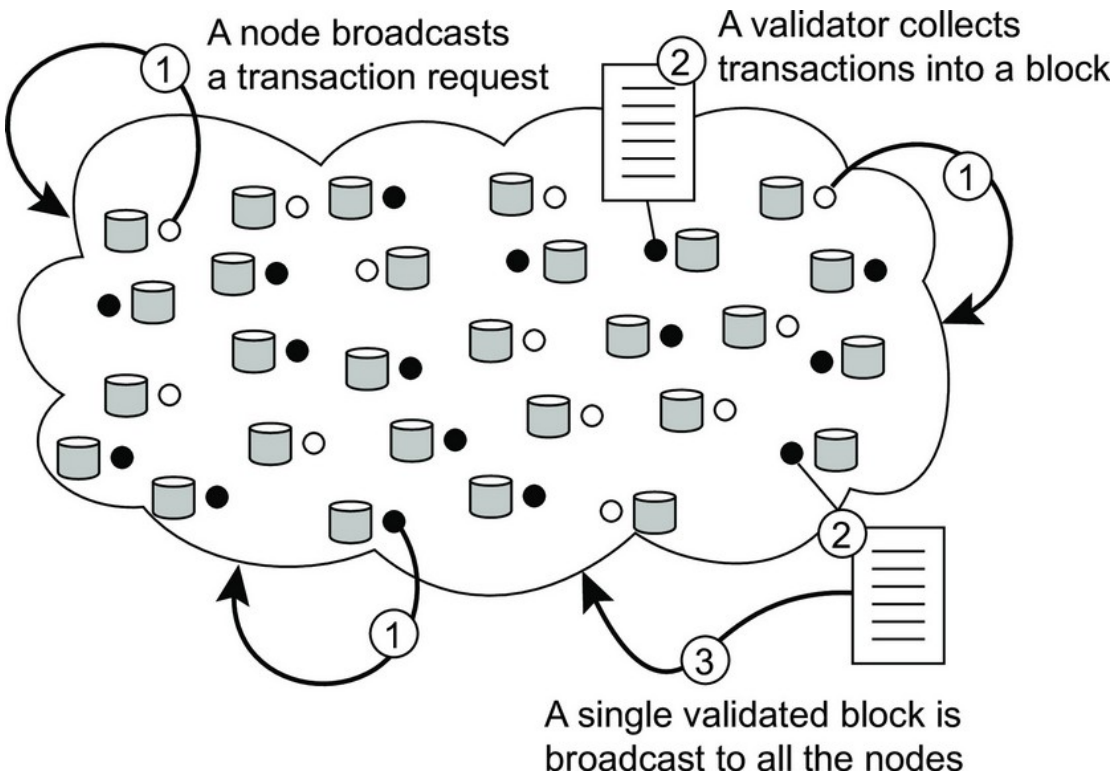
Blockchains

Principle working of a blockchain system



Blockchains

Principle working of a blockchain system



Observations

- Blocks are organized into an unforgeable append-only chain
- Each block in the blockchain is immutable ⇒ massive replication
- The real snag lies in who is allowed to append a block to a chain