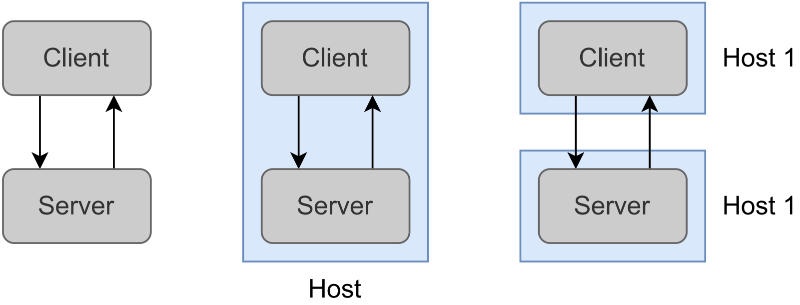
Chapter 2

Architectures

September, 20th

Introduction

- Software architecture
  - How software components are organized and interact
- System architecture
  - How software components are instantiated on real machines



Software architecture

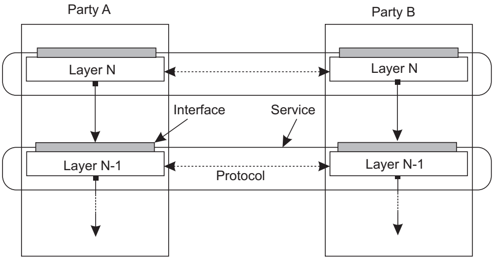
Architectural Styles

Software architecture

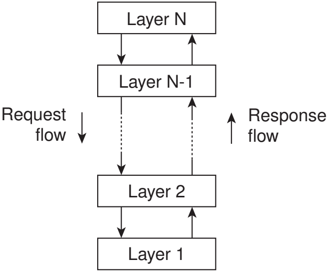
- Module (Component)
  - Unit with well-defined interfaces
  - Unit of deployment
- Important styles of architectures for distributed systems
  - Layered architectures
  - Object-based architectures
  - Data-centered architectures
  - Event-based architectures
  - Service-oriented architectures
  - Resource-oriented architectures

Layered architectures

- Component at layer L_i can call components in layer L_{i-1} but not components in layer L_{i+1}
- Example: networked protocols



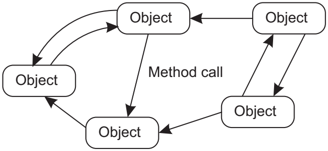
Layered communication-protocol stack



The layered architectural style

Object-based architectures

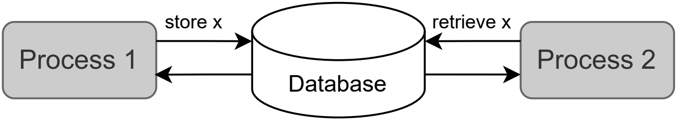
- Each object is a component, connected through a remote procedure call mechanism



The object-based architectural style

Data-centered architectures

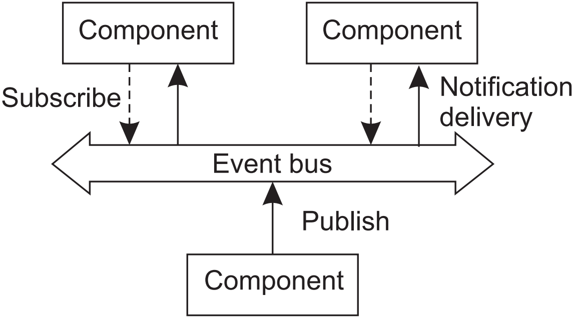
- Processes communicate through a common (active or passive) repository
- For example, applications can communicate through a shared distributed file system or database



Event-based architectures

aka publish-subscribe systems

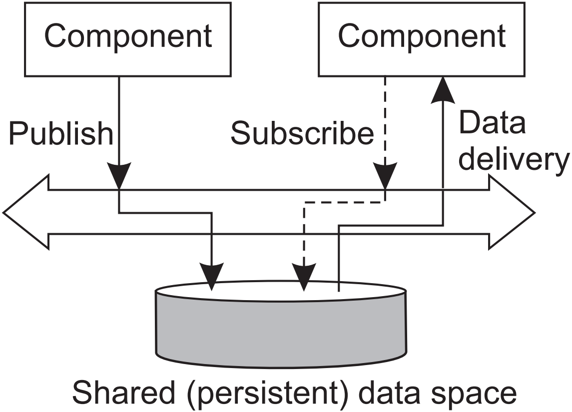
- Processes communicate through the *propagation of events*
- Processes *publish* events and the middleware ensures that processes that *subscribed* to those events will receive them
- Processes are loosely-coupled (i.e., don’t refer to each other)



The event-based architectural style

Shared data-space architectures

- Similar to data-centered and event-based architectures



Shared data-space architectural style

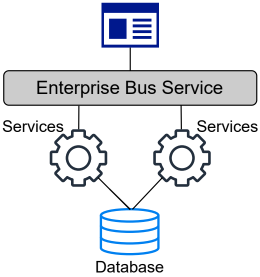
Architectural styles and coupling

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

Coupling in component coordination

Service-oriented architectures

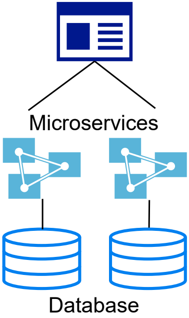
- Service interface and implementation
- Service contract
- Service provider
- Service consumer
- Service registry/repository
- Services possibly composed of other services
- e.g., Web Services



Service-oriented architectures

Microservices

- Latest in SOA
- No clear consensus on microservices vs services
- Independent processes communicating over network via messaging
- Services organized around business capabilities
- Possibly implemented using different languages



Resource-oriented architectures

RESTful (representational state transfer) architectures

- Collection of resources, individually managed by components
- Resources added, removed, retrieved, modified
  - Single naming scheme
  - All services same interface
  - **Self-describing messages ⇔ no sessions**

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

RESTful operations

Resource-oriented architectures

RESTful (representational state transfer) architectures

Example: Amazon Simple Storage Service (S3)

- Objects (i.e., files) placed into buckets (i.e., directories). Buckets can not be placed in buckets
- Operations on object `o` in bucket `b` requires the following identifier: `http://b.s3.amazonaws.com/o`
- Typical operations (via HTTP requests)
  - Create bucket/object: PUT with URI
  - Listing objects: GET on bucket
  - Reading object: GET on full URI

System Architectures

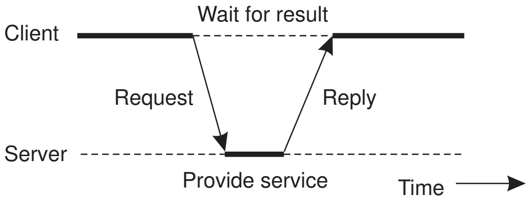
System architectures

- Decide for software components (modules), where to place each component, and how they physically interact
- Two main types
  - Centralized architectures
  - Decentralized architectures

Centralized architectures

Client-server model

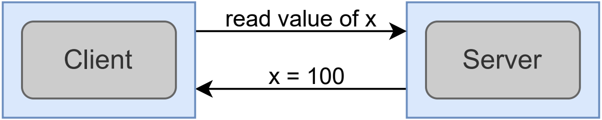
- Server implements some service
- Client requests service and waits for reply



General interaction between a client and a server

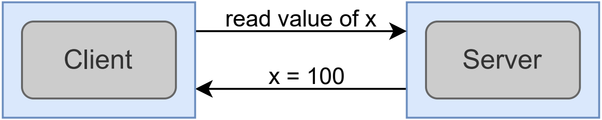
Centralized architectures

Client-server: communication



Centralized architectures

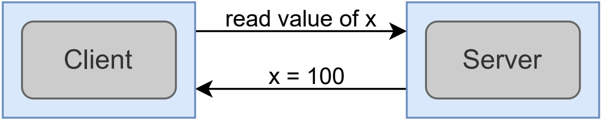
Client-server: communication



- Connectionless-protocol
  - Highly efficient, but...
  - More complex to handle transmission failures

Centralized architectures

Client-server: communication



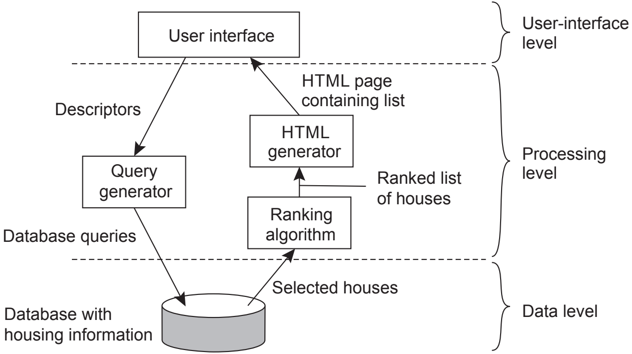
- Connectionless-protocol
  - Highly efficient, but...
  - More complex to handle transmission failures
- Reliable connection-oriented protocol
  - Relatively low performance
  - More appropriate for wide-area networks
  - Requests and replies use the same connection

Centralized architectures

Typical layered *software* architecture

- The **user-interface** level: all details necessary to the interface
- The **processing** level: contains typically the applications
- The **data** level: where the actual data is placed

Example:

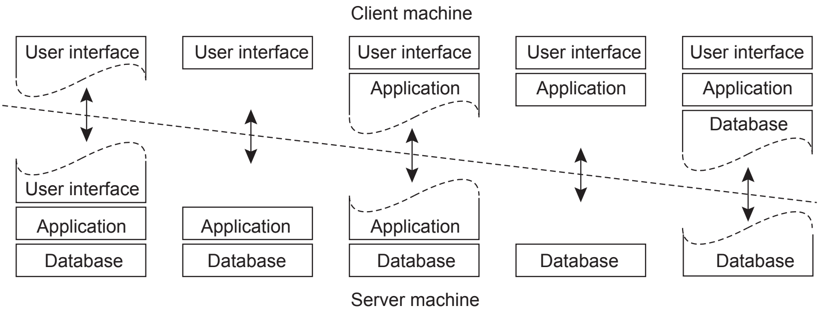


The simplified organization of an Internet search engine into three layers

Centralized architectures

Multi-tiered *system* architectures

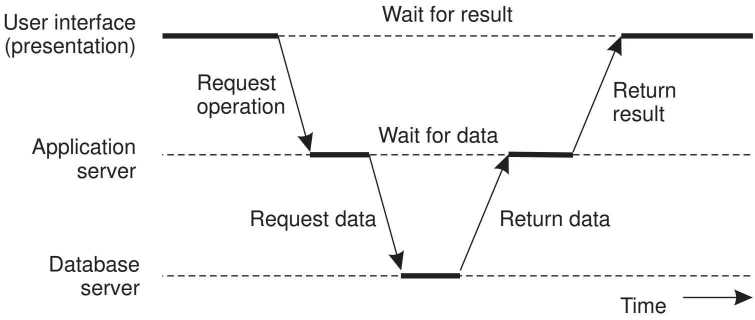
- How to distribute the many levels among machines?
- The simplest organization is to have only two types of machines:
  - Client machine: parts implementing user-interface level
  - Server machine: parts implementing processing and data level



Alternative client-server organizations.

Centralized architectures

Physically three-tier system architectures



An example of a server acting as client

Decentralized architectures

- Vertical distribution
  - Vertical distribution: achieved by placing logically different components on different machines
  - Example: physically three-tier applications
- **Horizontal distribution**
  - Clients (and servers) are physically split up into logically equivalent parts, each part operating on its own share of the data
  - Example: peer-to-peer systems

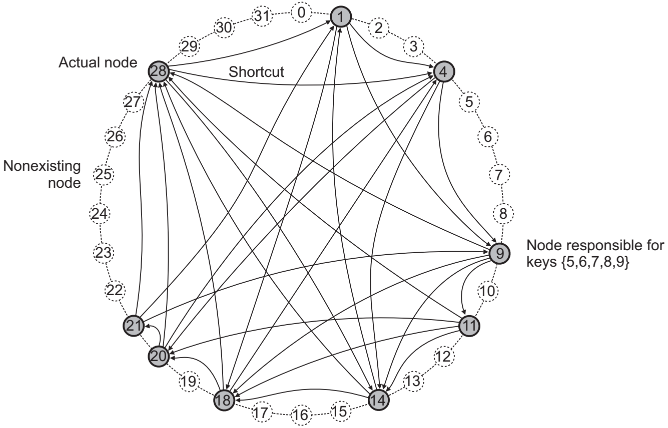
Decentralized architectures

Structured peer-to-peer architectures

- Overlay network: nodes are processes and links are possible communication channels (e.g., TCP connections)
- Deterministic procedure to build overlay network
- Distributed hash table (DHT)
  - Data items are assigned a random key (from a 128-bit space)
  - Nodes are also assigned a random key in the space
  - How to map keys to nodes so that lookup is efficient

Structured peer-to-peer architectures

Chord DHT



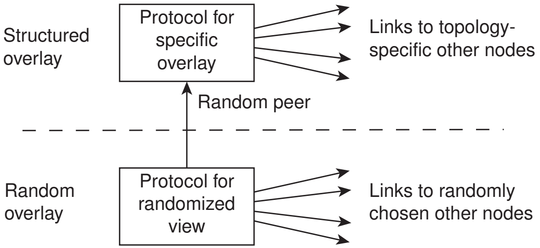
The mapping of data items onto nodes in Chord.

Decentralized architectures

Unstructured peer-to-peer architectures

- Randomized procedure to build overlay network
- Each node has a list of neighbors, constructed in a “random” way
- A typical issue is how to build a random graph
- Data items are assumed to be randomly placed on nodes
- Finding a data item:
  - flooding the network
  - random walk
- Example: Gnutella

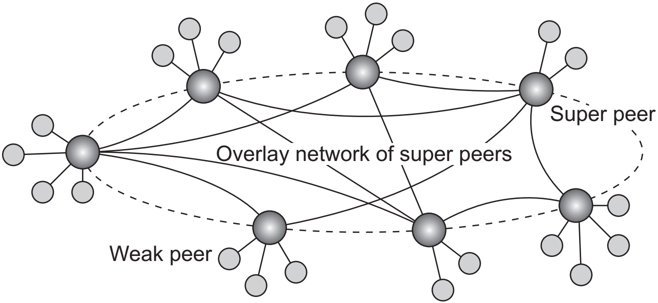
Topology management of overlay networks



A two-layered approach for constructing and maintaining specific overlay topologies using techniques from unstructured peer-to-peer systems.

Hierarchically organized peer-to-peer networks

- **Superpeers:** special nodes that keep an index of data items
- Regular nodes join the system by connecting to a superpeer



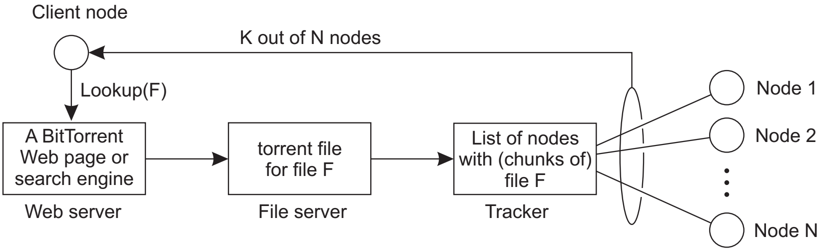
A hierarchical organization of nodes into a superpeer network

Hybrid architectures

Collaborative systems

Example: BitTorrent searching for file `f`

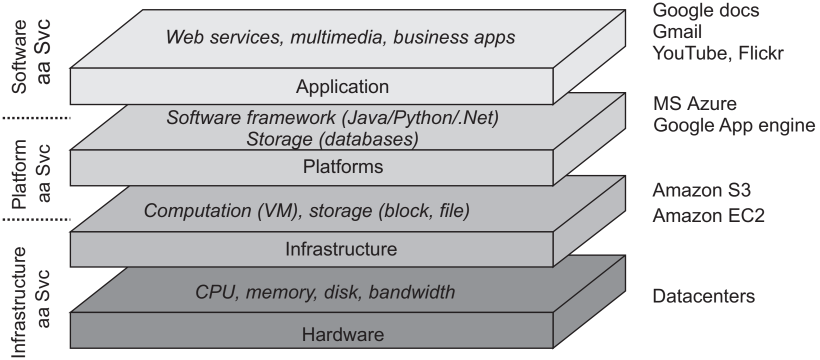
- Lookup `f` at global directory ⇒ returns torrent file with tracker
  - Server keeping account of active nodes with chunks of `f`
- Join swarm, get free chunk, rest via exchanging



Architecture of a BitTorrent network

Cloud computing

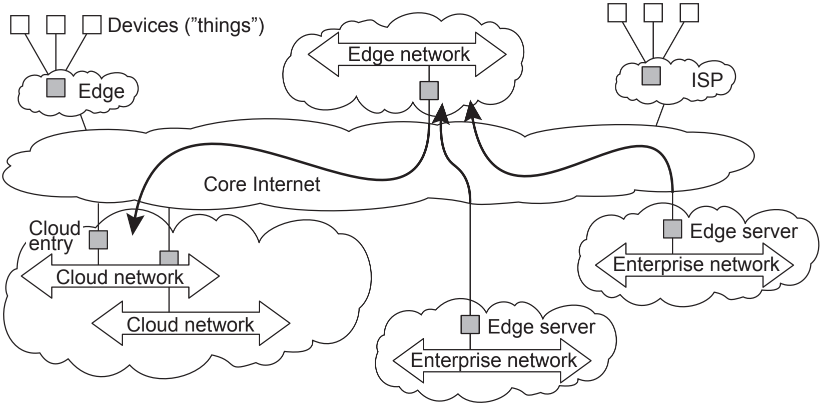
Abstraction layers



The organization of clouds

Edge computing

Servers at boundary between enterprise networks and Internet

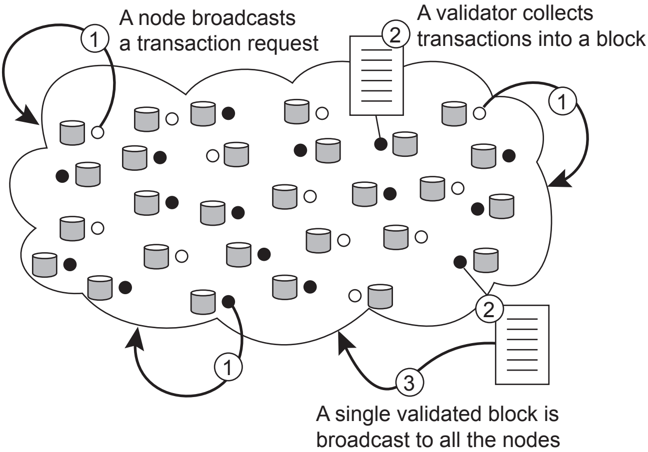


Edge computing

Motivation

- **Latency and bandwidth (BW)**:
  - important for real-time applications e.g. augmented reality
  - latency (and BW) to cloud are underestimated (overestimated)
- **Reliability**:
  - connection to cloud can be unreliable
  - often high connectivity guarantees are required
- **Security and privacy**:
  - resources are not always better protected in edge data centers,
  - but, security handling in cloud is trickier than within organization.

Blockchains

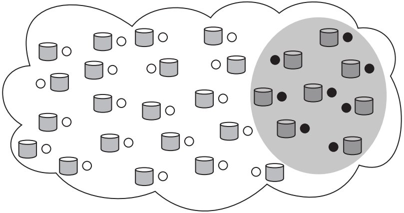


- Blocks are immutable
- Blocks organized into append-only chain (“ledger”)
- Who can append?

Blockchains

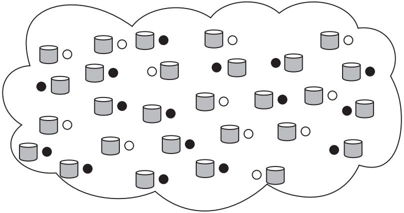
Permissioned/distributed

- Subset of servers decide
- 2/3 correct ones needed
- only 10s of servers



Permissionless/decentralized

- Participants elect leader (who can append) block
- Fair, robust, secure election is hard at large scale



Architectures vs Middleware

Architectures vs Middleware

Motivation

- Middleware provides a degree of distribution transparency
- How does the middleware relate to architectural issues?
- Some middleware systems follow/induce an object-based architectural style (e.g., CORBA) while others follow the event-based architectural style (e.g., TIB/Rendezvous)
- How to accommodate different architectural styles?
- How to mediate between application (software) and middleware (besides APIs)?

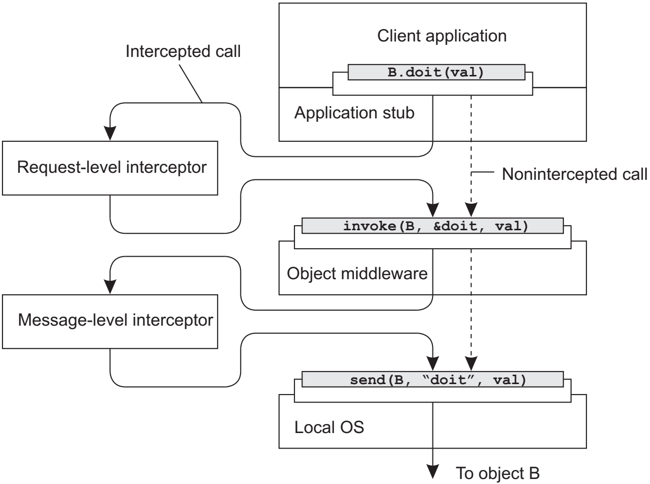
Architectures vs Middleware

Interceptors

- Offer a means to adapt the middleware
- Mechanism to break the usual flow of control and allow other (application specific) code to be executed
- Improve software management (e.g., instrumenting the code)

Architectures vs Middleware

Interceptors



Using interceptors to handle remote-object invocations

Architectures vs Middleware

Related Concepts

- Wrappers and adapters
  - Provide *similar* **interface** as original abstraction
  - Implements additional logic before/after invoking original interface or alternative logic instead of original one
  - Additional module
- Proxies/stubs
  - Provide similar interface as original abstraction
  - Usually own components vs wrappers and adapters