Distributed System Models & Architecture

Module A2

Distributed & Cloud Computing

Sheheryar Malik, Ph.D.

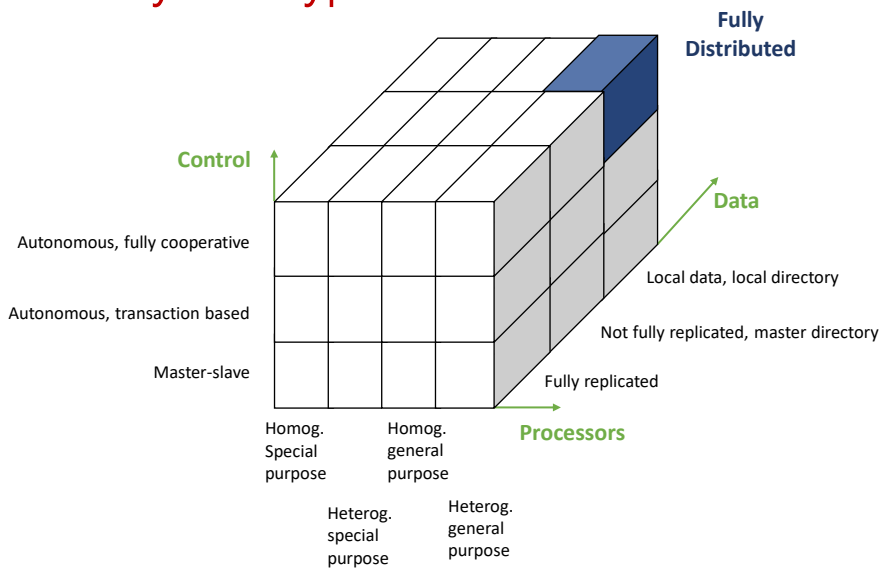
1

Difficulties for Distributed Systems

- Widely varying models of use
  - wide variations in workload
  - connectivity problems
  - latency & bandwidth requirements
- Wide range of operating environments
  - heterogeneity: H/W, OS, network
  - widely differing scales
- Distribution related inherent problems
  - non-synchronized clocks
  - conflicting data/state updates
  - many modes of failures (H/W + S/W)
- External threats
  - attacks on data integrity & secrecy
  - denial of service

2

Distributed System Types



Architecture Models

- Placement of components
  - patterns for distribution
    - data & processing
  - interplay of performance, reliability, security and cost
- Interrelationships between components
  - functional roles
  - patterns of communication
- Classification of processes
  - Servers & Clients
  - Peers

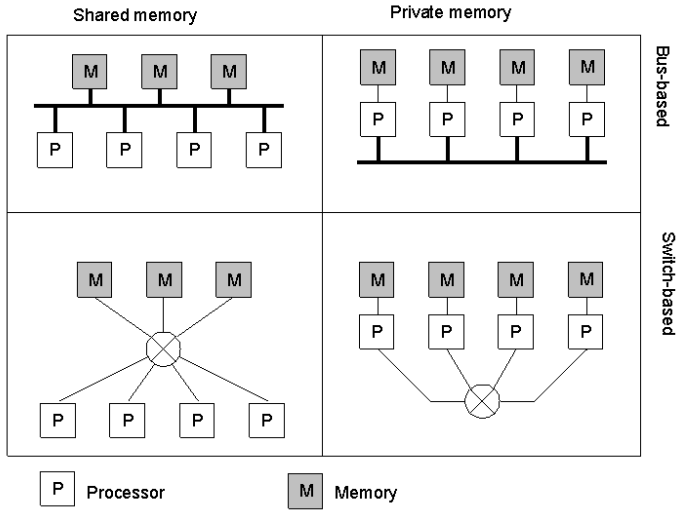
More Dynamic Systems

- Mobile code
  - delegation of tasks to a process
    - e.g. download code, execute locally
- Spontaneous networking
  - enable computers & other mobile devices to be added/removed transparently
    - discover available services
    - publish interfaces to services

Hardware Concept

- Multiprocessor architecture
  - Tightly coupled systems
  - Connected through shared memory
- Multicomputer architecture
  - Loosely coupled systems
    - also called Cluster of Workstations
  - Connected through network cables/ wireless
- Bus based or switch based

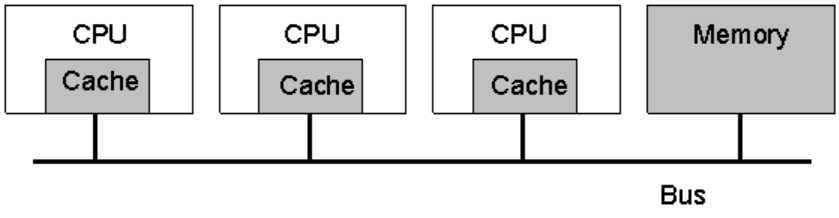
Basic Organization



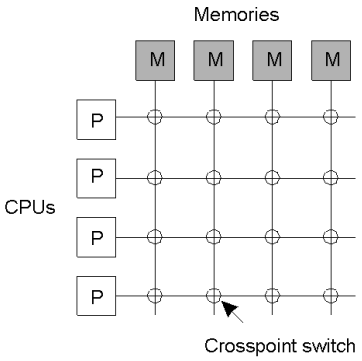
Different basic organizations and memories in distributed computer systems

Multiprocessor Architecture (Bus based)

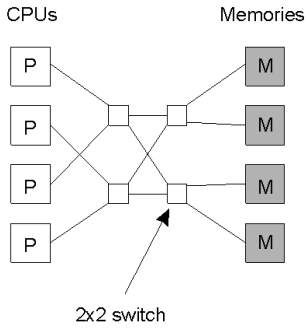
- A bus-based multiprocessor



Multiprocessor Architecture (Switch based)



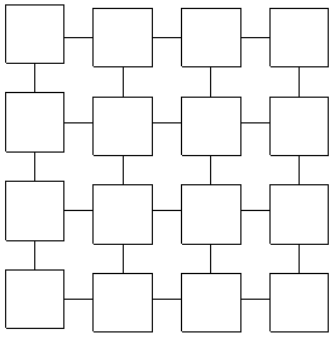
A crossbar switch



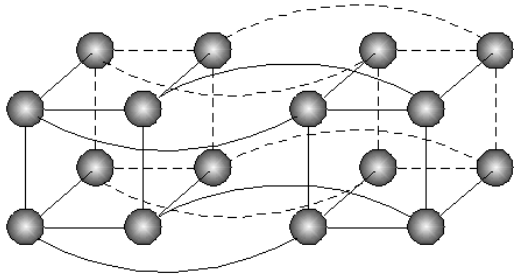
An omega switching network

9

Multicomputer Systems



Grid



Hypercube

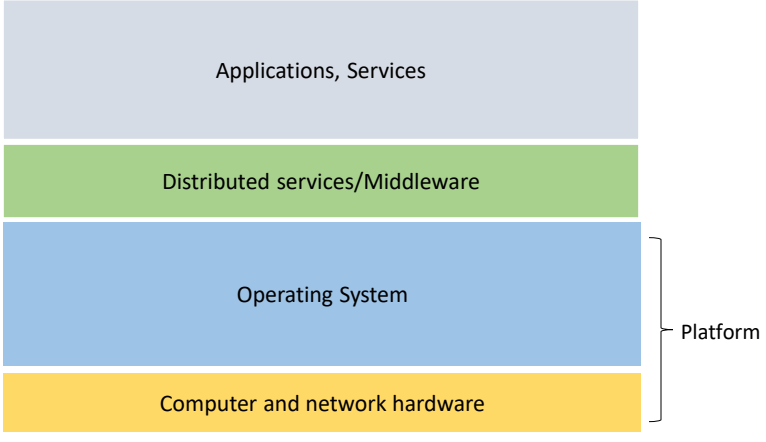
10

Software Concepts

System	Description	Main Goal
DOS	Tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer atop of NOS implementing general-purpose services	Provide distribution transparency

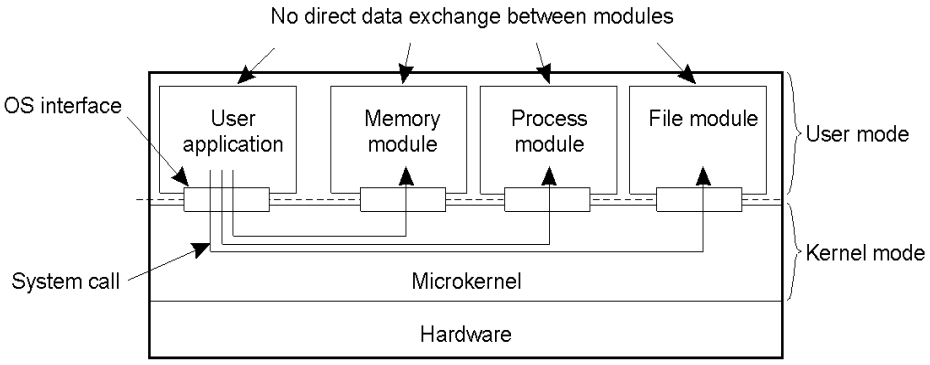
11

Layers in Distributed System Model



12

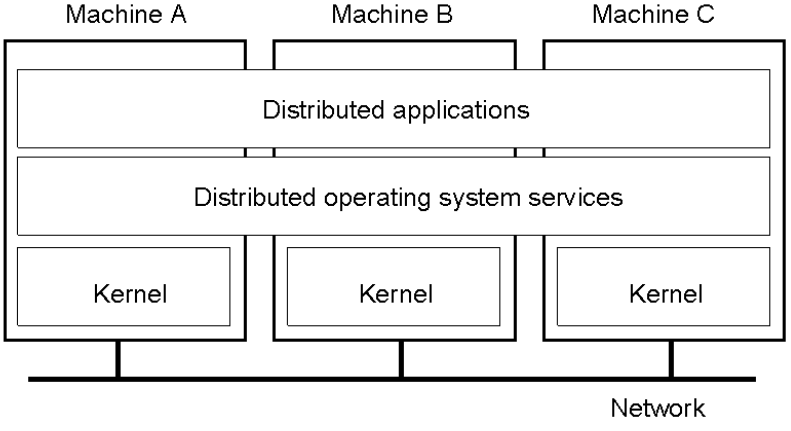
Uniprocessor Operating Systems



Separating applications from operating system code through a microkernel

13

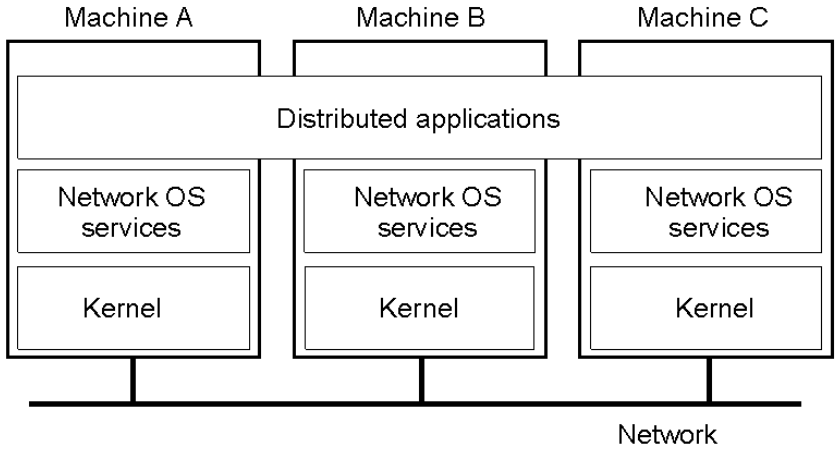
Distributed Operating Systems



General structure of a multicomputer distributed operating system

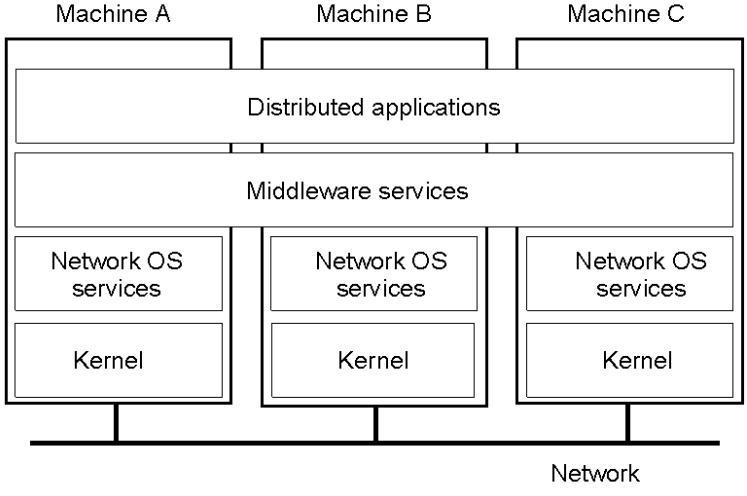
14

Network Operating System



15

Positioning Middleware



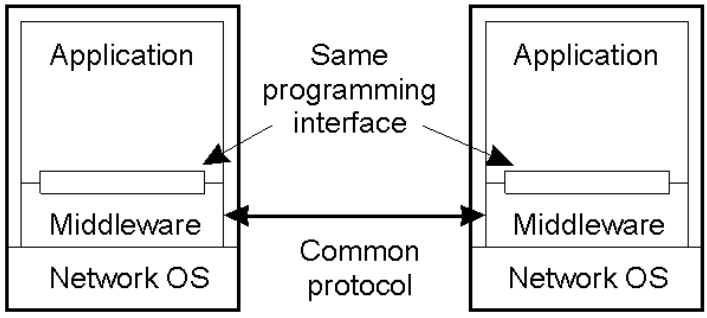
General structure of a distributed system as middleware

16

Middleware

- It is the fundamental building block of a distributed system
  - in the form of processes or objects
- It provides the distribution transparencies
- Provides abstractions
  - remote method invocation
  - group of processes
  - notification of events
  - replication of shared data
  - real-time transmission of multimedia streams

Middleware and Openness



In an open middleware-based distributed system, the protocols used by each middleware layer should be the same, as well as the interfaces they offer to applications

Comparison between Systems

Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	High
Openness	Closed	Less-open	Open	Open

Distributed Service Model

Centralized Architecture

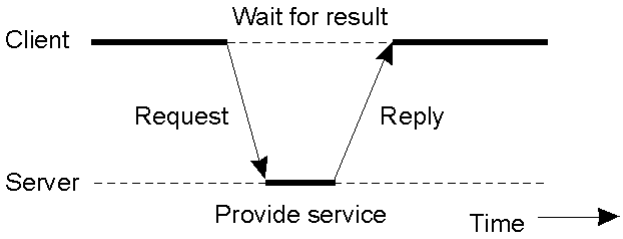
Client/Server model

Decentralized Architecture

Peer to Peer model

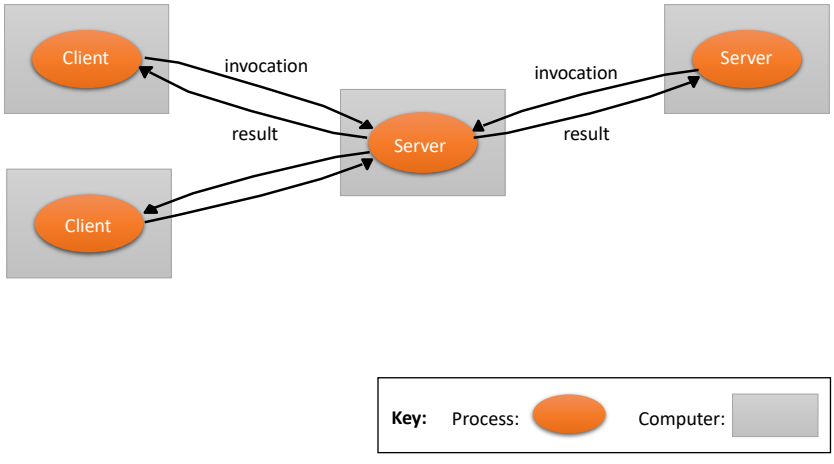
Client/Server Model

- 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 w.r.t. to using services

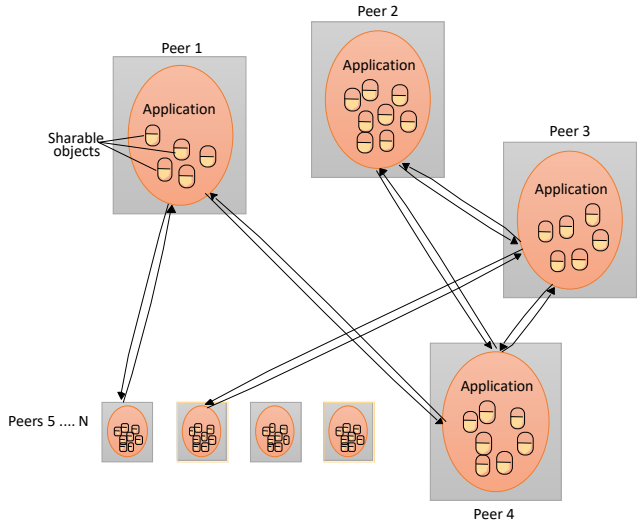


General interaction between a client and a server

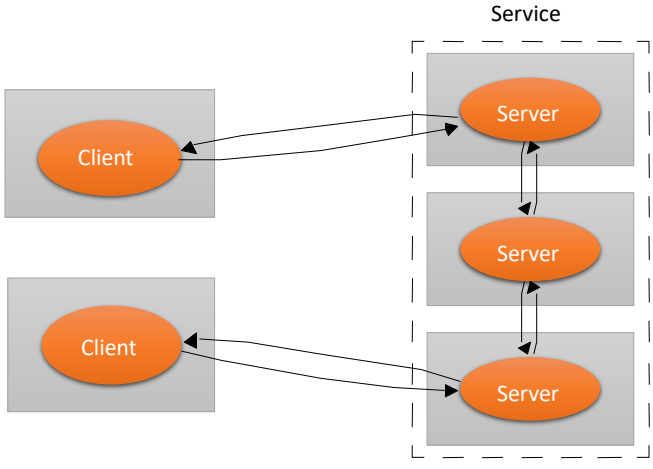
Client/Server Model



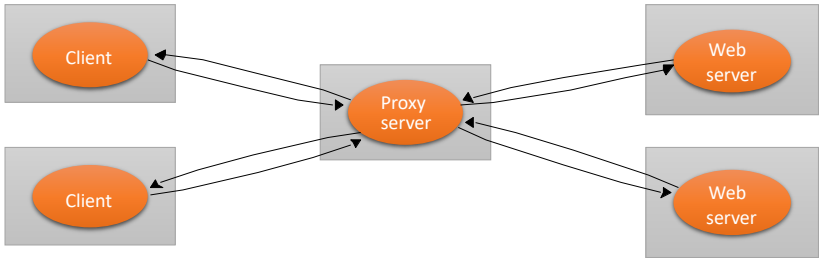
Peer-to-peer Model



A Service Provided by Multiple Servers



Web Proxy Server



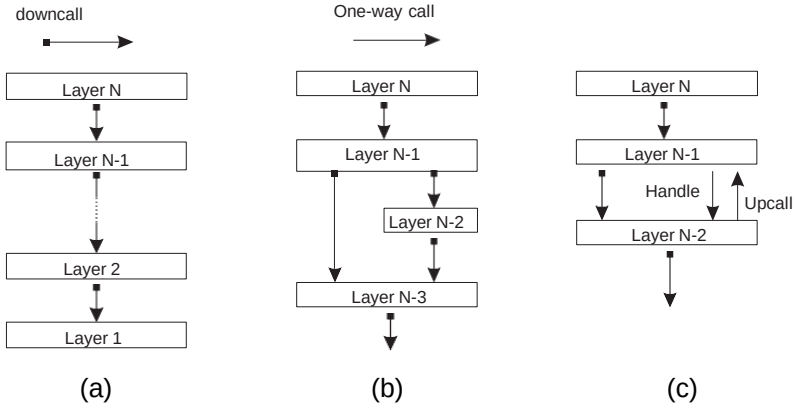
Architectural Styles

- 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

Architectural Styles

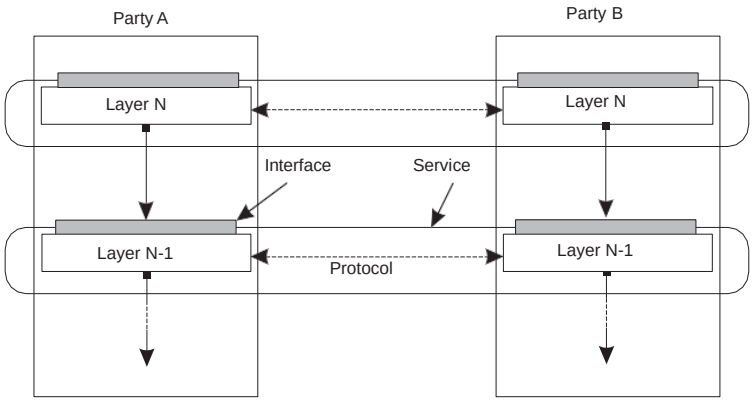
- Layered architectures
- Object-based architectures
- Resource-centered architectures
- Event-based architectures

Layered Architecture



Different layered organizations Request/Response

Example: Communication Protocols

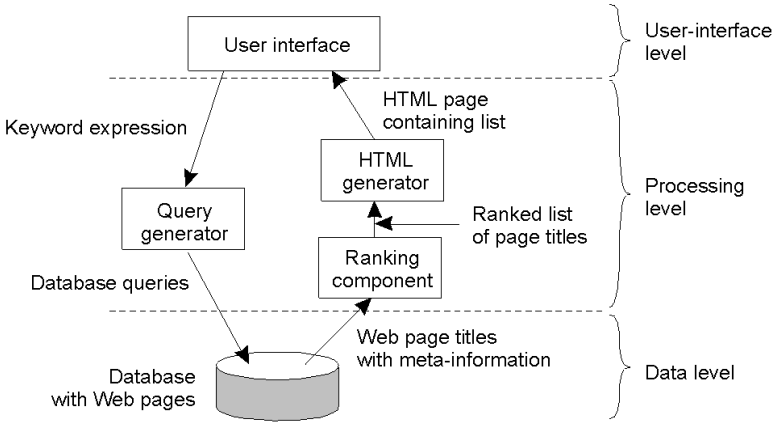


Protocol, service, interface

Application Layering

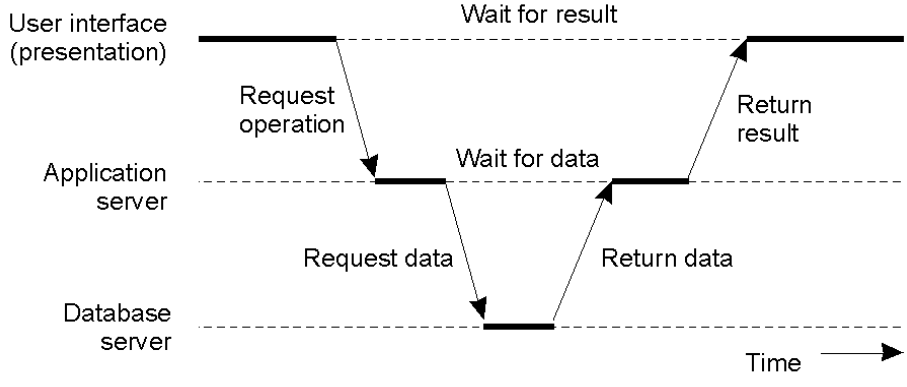
- Layer architecture delegates the responsibilities to different layers
- This layering is found in many distributed information systems, using traditional database technology and accompanying applications
- Traditional three-layered view
  - Application-interface layer
    - contains units for interfacing 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



The general organization of an Internet search engine into three different layers

Multitiered Architectures

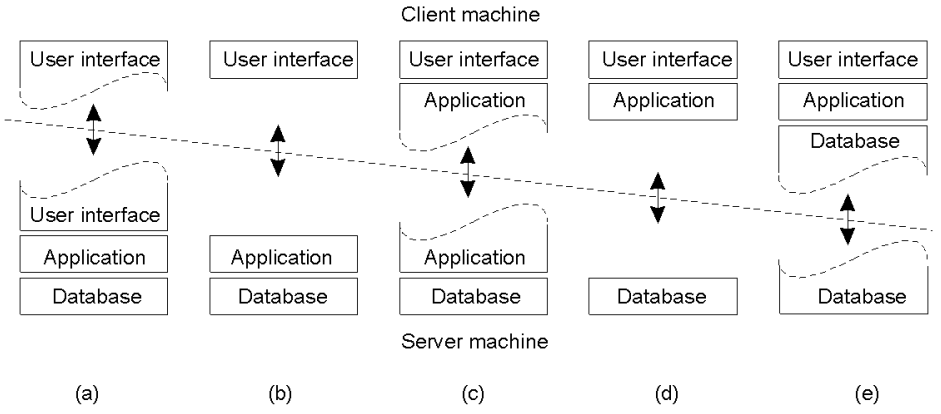


An example of a server acting as a client

Multi-tiered Physical Architectures

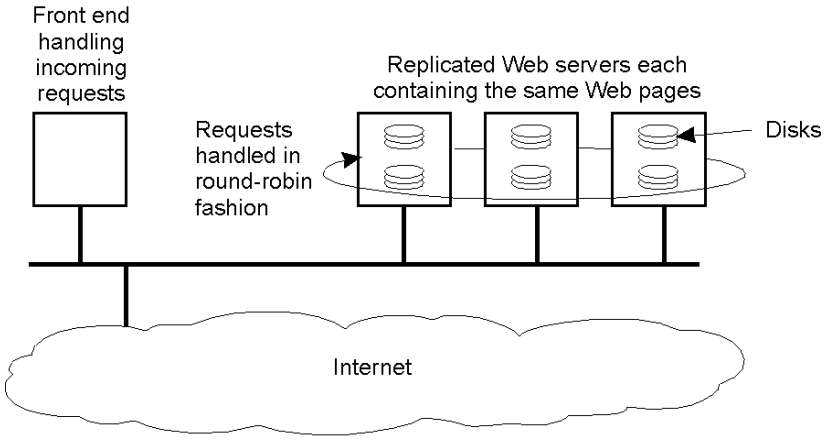
- Single-tiered
  - dumb terminal/mainframe configuration
- Two-tiered
  - client/single server configuration
- Three-tiered
  - each layer on separate machine

Multi-tiered Architectures



Alternative client-server organizations (a) – (e)

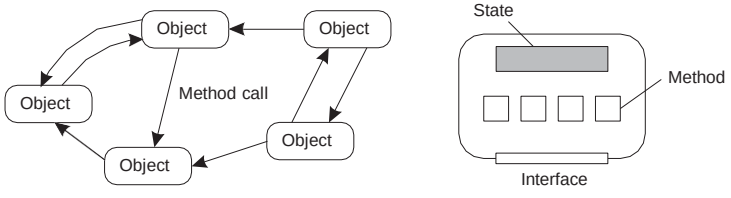
Modern Architectures



An example of horizontal distribution of a Web service.

Object-based Architectures

- 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



Resource-centered Architectures (RESTful)

- View a distributed system as a collection of resources, individually managed by components
- Resources may be added, removed, retrieved, and modified by (remote) applications
  - Resources are identified through a single naming scheme
  - All services offer the same interface
  - Messages sent to or from a service are fully self-described
  - 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

- 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

Resource-centered Architectures

- 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

Resource-centered Architectures

- 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")
```

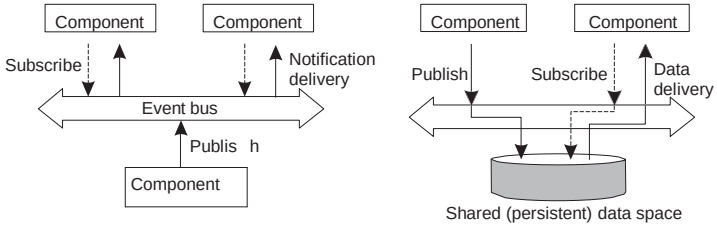
- RESTful

```
PUT "http://mybucket.s3.amazonsws.com/"
```

Event-based Architectures (Publish-subscribe)

Coordination: Temporal and referential coupling

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



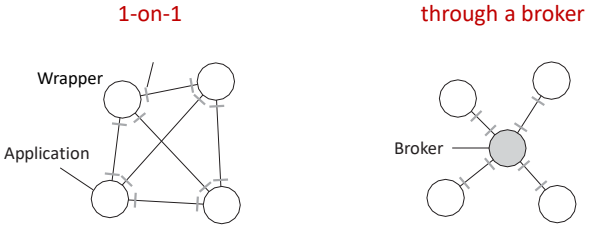
Event-based and Shared data space

Using legacy to build middleware

- Problem
  - The interfaces offered by a legacy component are most likely not suitable for all applications
- Solution
  - A wrapper or adapter offers an interface acceptable to a client application. Its functions are transformed into those available at the component

Organizing Wrappers

Two solutions:



Complexity with N applications

- 1-on-1: requires $N \times (N-1) = O(N^2)$ wrappers
- broker: requires $2N = O(N)$ wrappers

Alternative Organizations

- Vertical distribution
  - Comes from dividing distributed applications into three logical layers, and running the components from each layer on a different server (machine)
- Horizontal distribution
  - A client or server may be physically split up into logically equivalent parts, but each part is operating on its own share of the complete data set
- Peer-to-peer architectures
  - Processes are all equal: the functions that need to be carried out are represented by every process ⇒ each process will act as a client and a server at the same time (i.e., acting as a servant)