Lecture#2: Distributed Systems Models and Architectures

S. M. Ahsan Kazmi

Outline

- Traditional DS Models
- Emerging DS Models
- Uniprocessor and Distributed OS
- Architectural Styles
- System Architecture

Distributed Systems Models

- **Minicomputer model** (e.g., early networks)
  - Each user has a local machine
  - Local processing but can fetch remote data (files, databases)
- **Workstation model** (e.g., Sprite)
  - Processing can also migrate
- **Client-server Model** (e.g., V system, world wide web)
  - User has a local workstation
  - Powerful workstations serve as servers (file, print, DB servers)

Distributed System Models (cont.)

- **Cluster computing systems / Data centers**
  - LAN with a cluster of servers + storage
    - Linux, Mosix, ..
    - Used by distributed web servers, scientific applications, enterprise applications

- **Grid computing systems**
  - Cluster of machines connected over a WAN
  - SETI @ home

- **WAN-based clusters / distributed data centers**
  - Google, Amazon, …

Emerging Models

- **Distributed Pervasive Systems**
  - “smaller” nodes with networking capabilities
    - Computing is “everywhere”
  - Home networks: Windows Media Center, …
  - Mobile computing: smartphones, iPods, Car-based PCs
  - Sensor networks

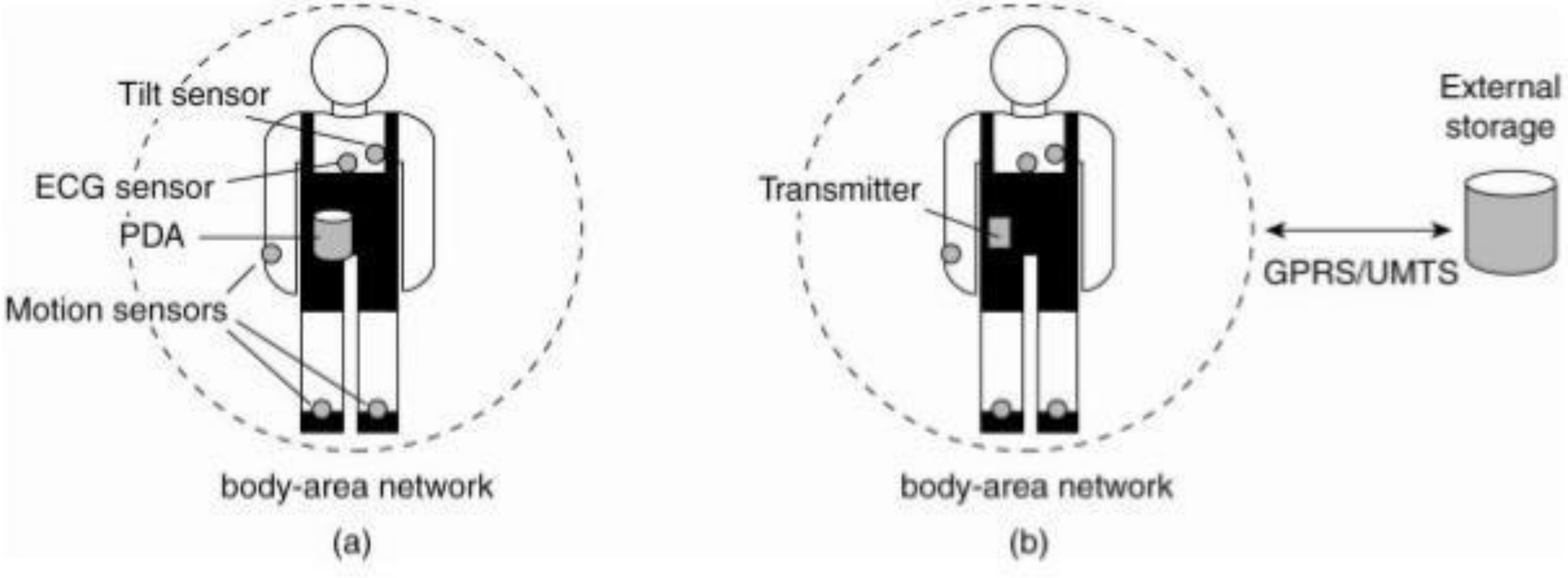
Distributed Pervasive Systems

Overlapping subtypes:

- **Ubiquitous computing systems**:
  - pervasive and continuously present, i.e., there is a continuous interaction between system and user.
- **Mobile computing systems**:
  - pervasive, but the emphasis is on the fact that devices are inherently mobile.
- **Sensor (and actuator) networks**:
  - pervasive, with emphasis on the actual (collaborative) sensing and actuation of the environment.

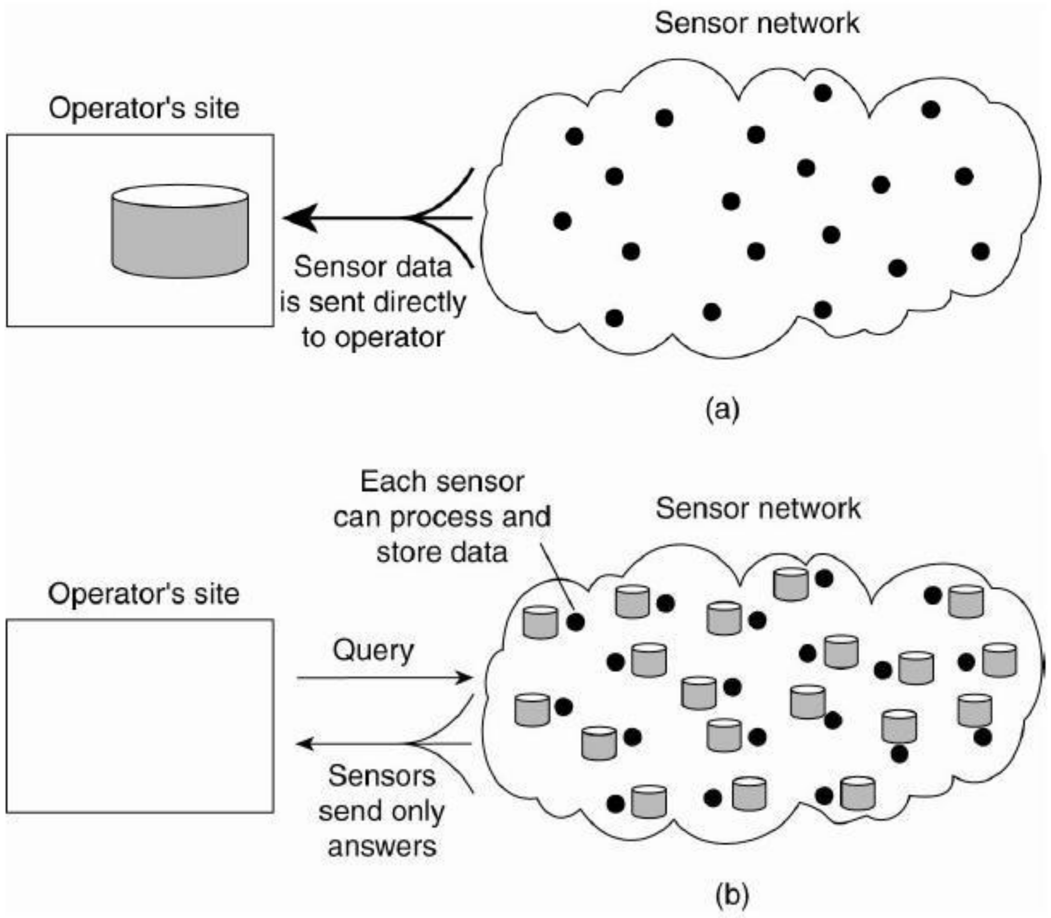
Distributed Pervasive systems (cont.)

Ubiquitous computing systems



Distributed Pervasive Systems (cont.)

Sensor network



Uniprocessor Operating Systems

- An OS acts as a resource manager or an arbitrator
  - Manages CPU, I/O devices, memory
- OS provides a virtual interface that is easier to use than hardware

- Structure of uniprocessor operating systems
  - Monolithic design (e.g., MS-DOS, early UNIX)
    - One large kernel that handles everything
  - Layered design
    - Functionality is decomposed into N layers
    - Each layer uses services of layer N-1 and implements new service(s) for layer N+1

Multiprocessor Operating Systems

- Like a uniprocessor operating system
- Manages multiple CPUs transparently to the user
- Each processor has its own hardware cache
  - Maintain consistency of cached data

Distributed Operating System

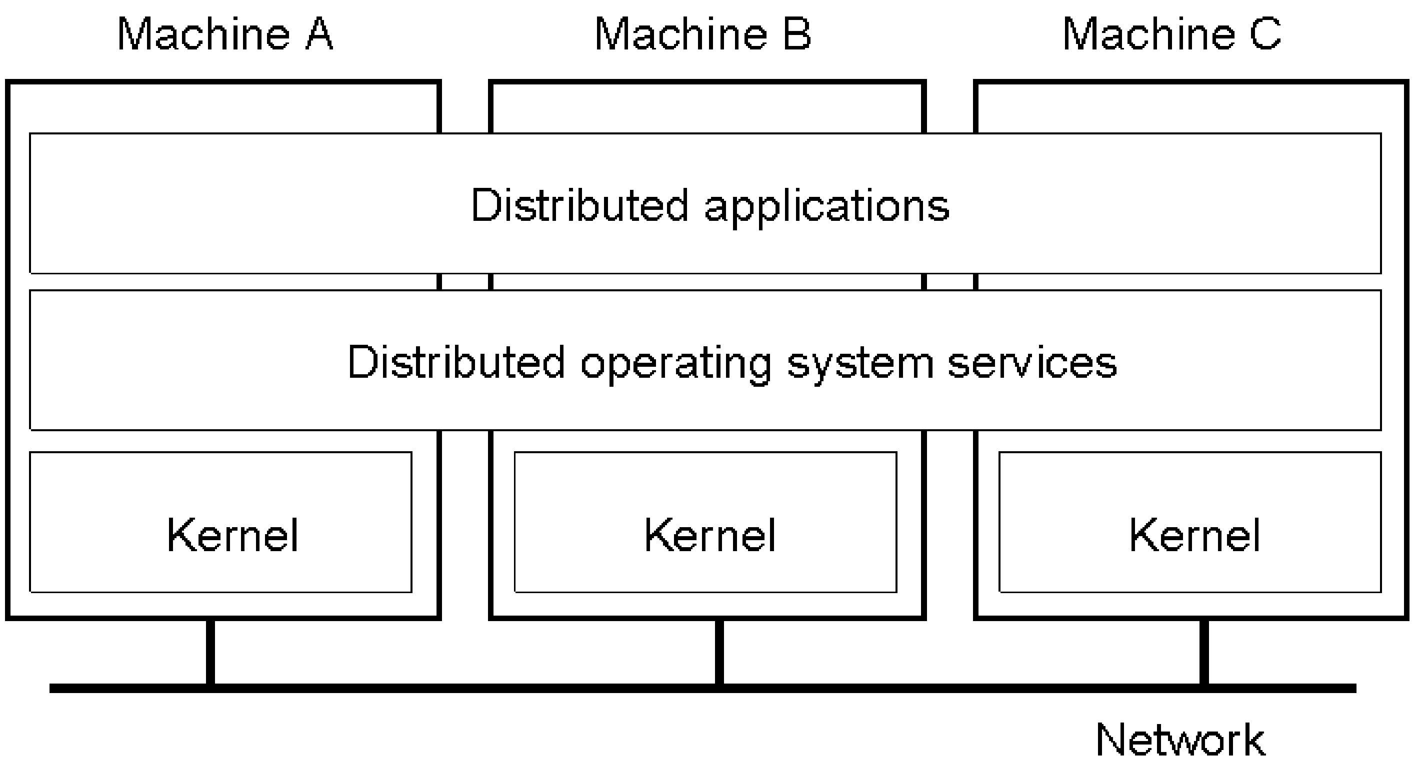
- Manages resources in a distributed system
  - Seamlessly and transparently to the user
- Looks to the user like a centralized OS
  - But operates on multiple independent CPUs
- Provides transparency
  - Location, migration, concurrency, replication,…
- Presents users with a virtual uniprocessor

Types of Distributed OSs

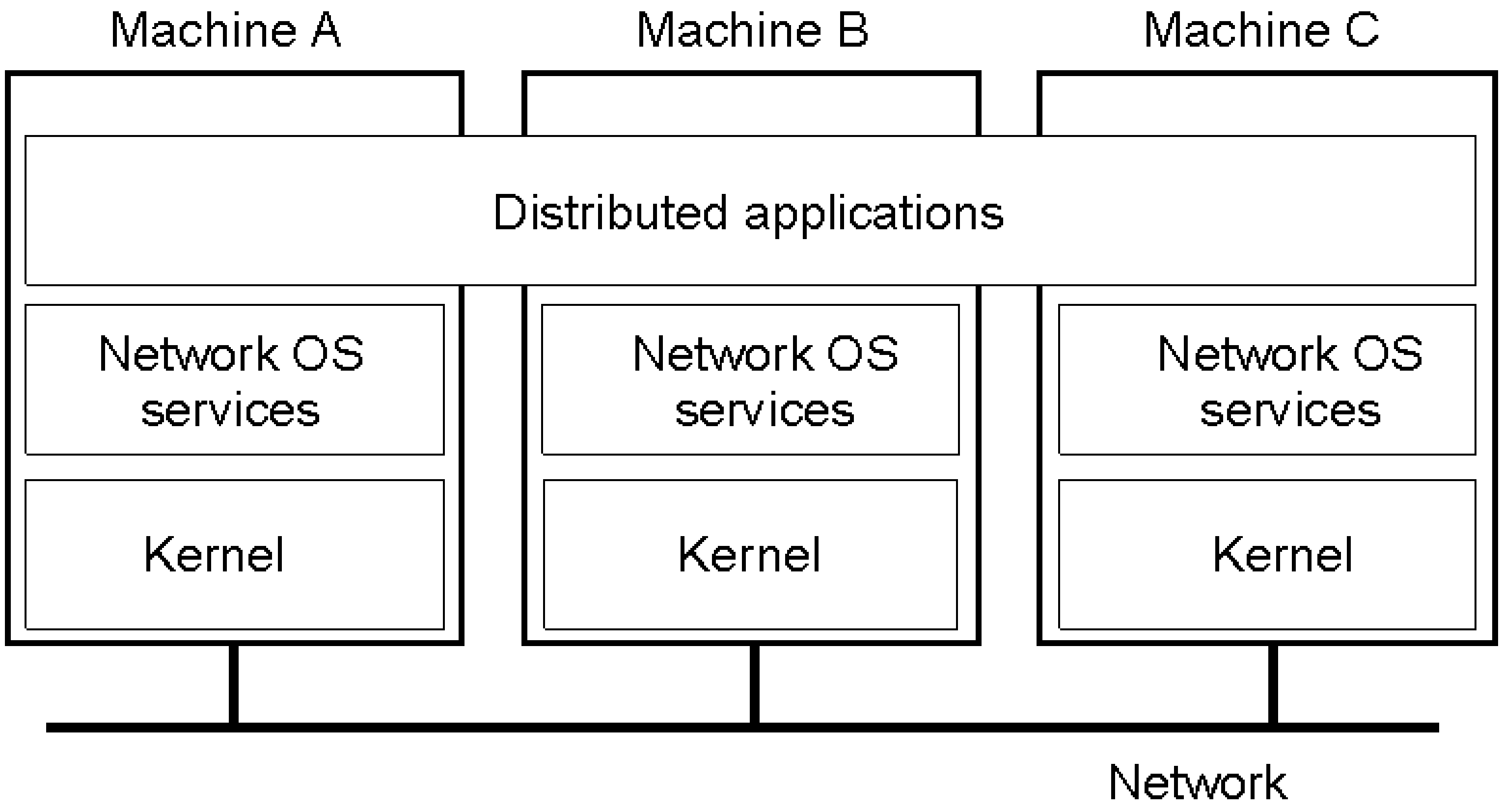
System	Description	Main Goal
DOS	A tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer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

Multicomputer Operating Systems

Example: MOSIX cluster - single system image



Network Operating System



Comparison between Systems

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

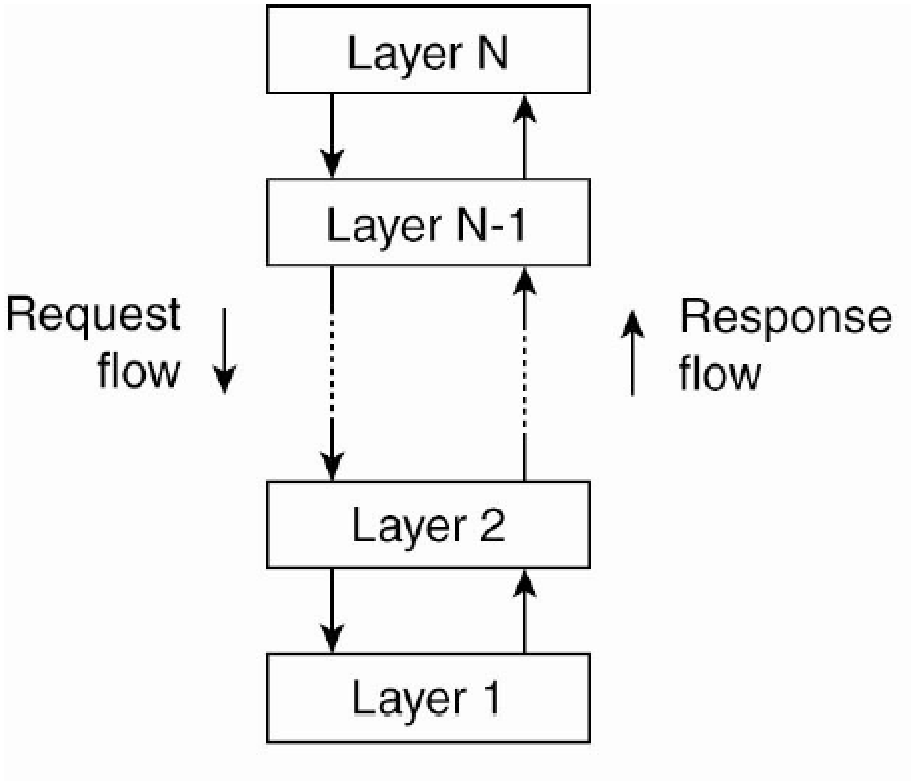
Architectural Styles

- Architectural style describes a particular way how we can organize a collection of components distributed over the various machines

- Important styles of architecture for distributed systems
  - Layered
  - Object-based
  - Event-based
  - Resource-centered

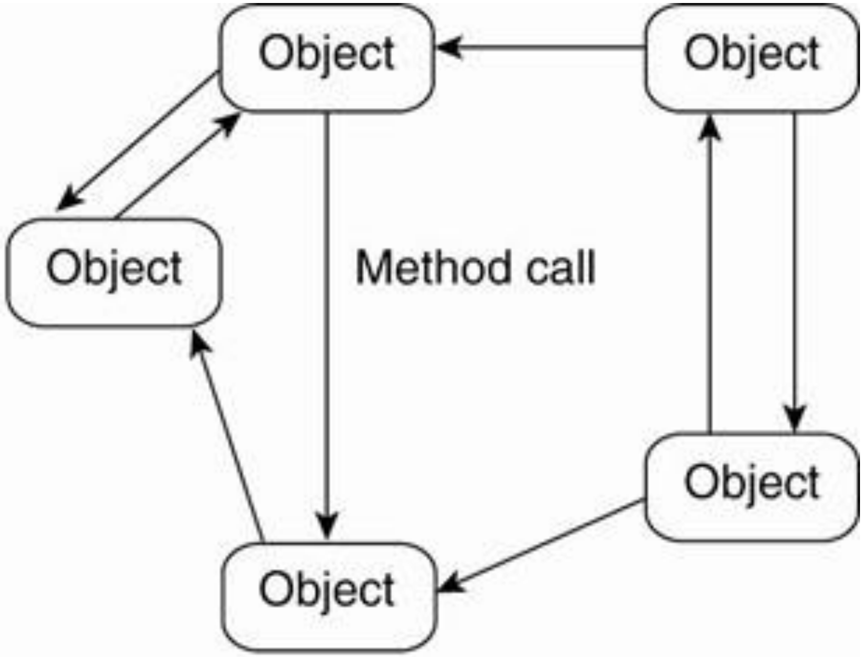
Layered Architectures

Layered system organizations: Each layer uses previous layer to implement new functionality that is exported to the layer above



Object-based Architectures

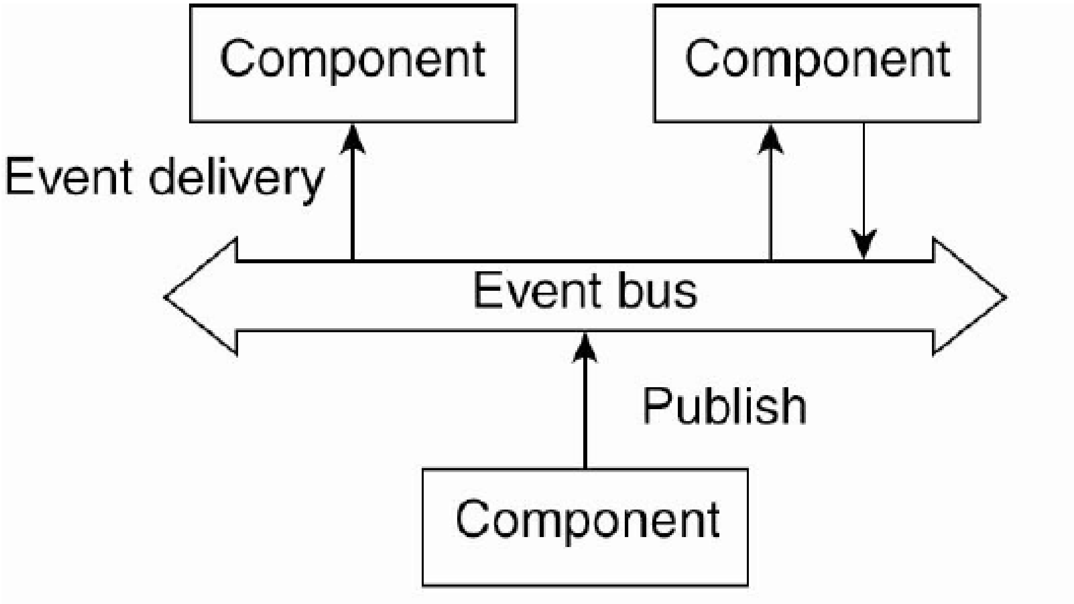
- Each object correspond to a component
- Typically, each object contains
  - Set of methods
  - State(s)
  - Interface



Event-based Architecture

Communicate via a common repository

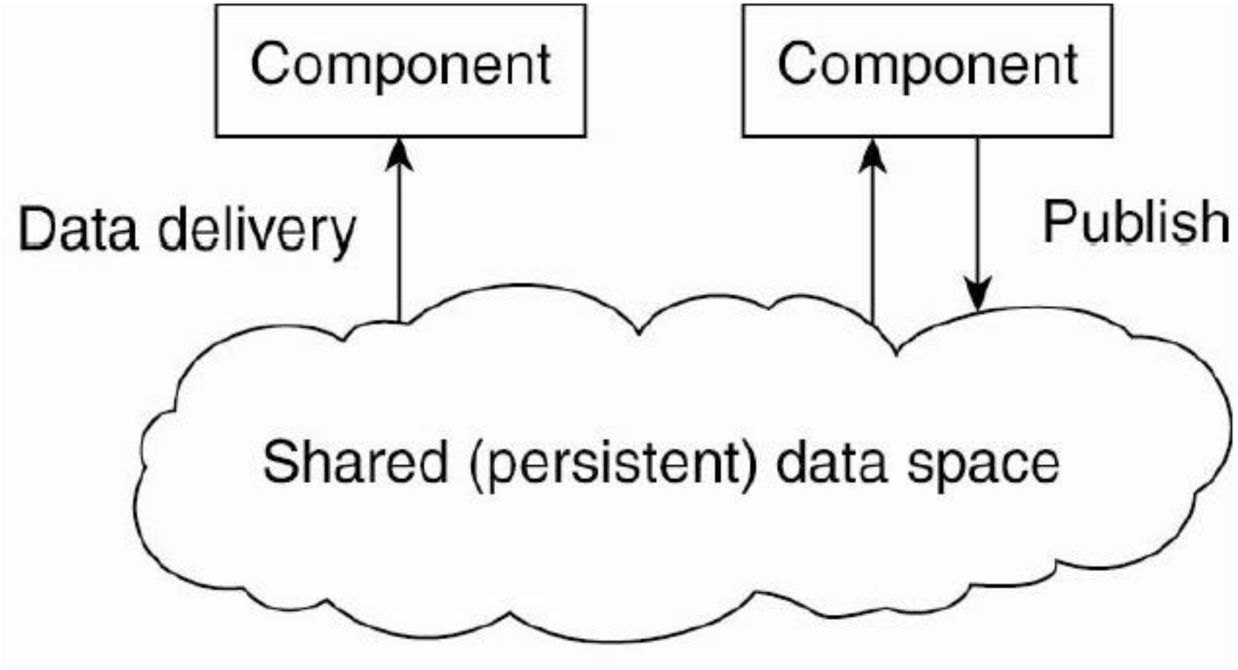
- Use a publish-subscribe paradigm
- Consumers subscribe to types of events
- Events are delivered once published by any publisher



Shared Data-space Architectures

"Bulletin-board" architecture

- Decoupled in space and time
- Post items to shared space; consumers pick up at a later time



System Architecture

System architecture – describe the placement of software components on physical machines

The realization of architecture may be

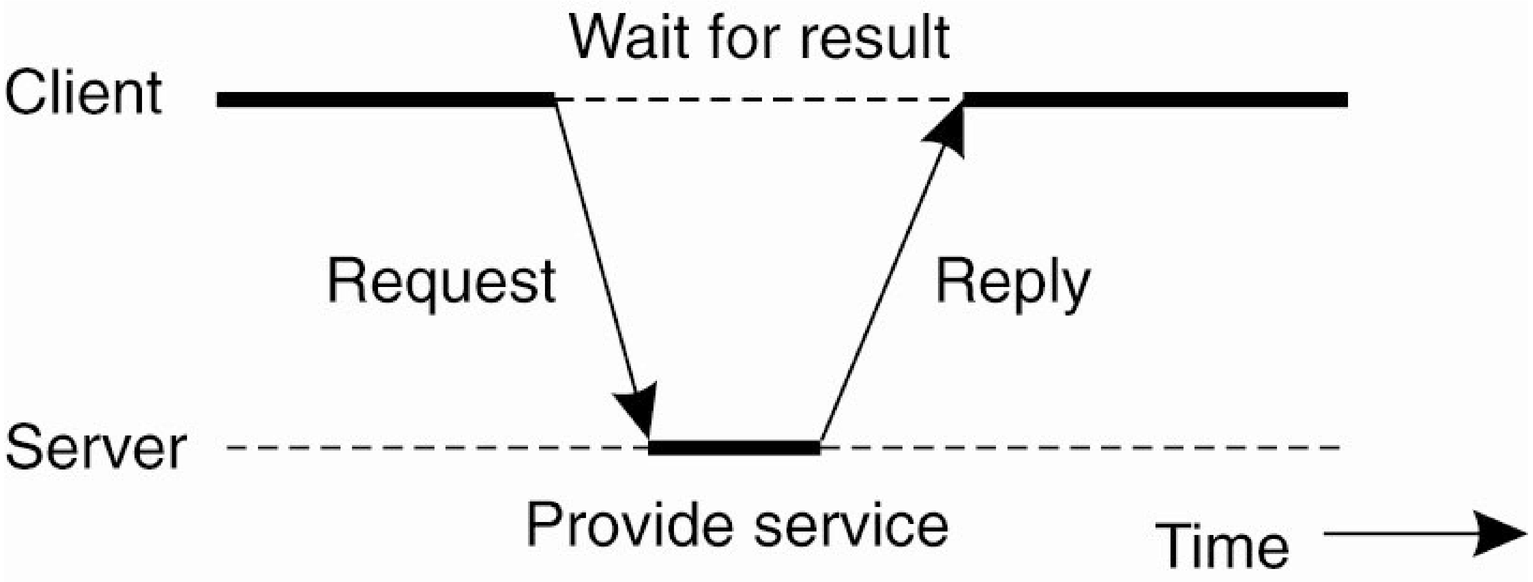
- **Centralized**. Client-server architecture
- **Decentralized**. Peer-to-peer architecture
- **Hybrid**. Combination of Centralized and Decentralized architectures

(according to Tanenbaum and van Steen)

Centralized Organizations

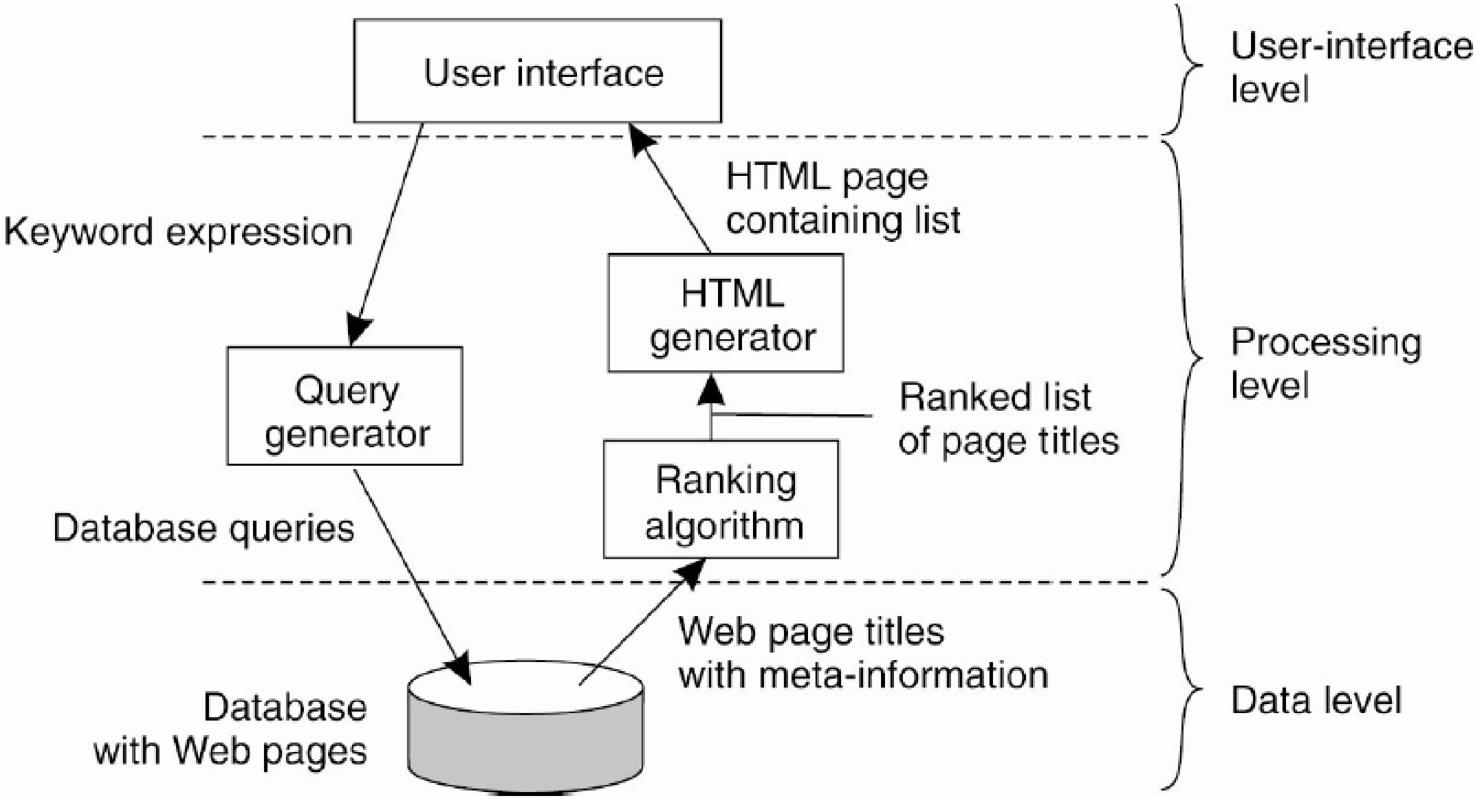
Traditional client-server architecture

- Logical separation on server and client functionality
- Application layering
    - User-interface level
    - Processing level
    - Data level



Application Layering

- **User-interface level** contains units for an application's UI
- **Processing level** contains the functions of an application
- **Data level** contains the data that user wants to manipulate through the application components



Search engine architecture with 3 layers

Multi-tiered Architectures

- The simplest organization is to have only two types of machines:
- A client machine containing only the programs implementing (part of) the user interface level
- A server machine containing the rest,
  - the programs implementing the processing and data level

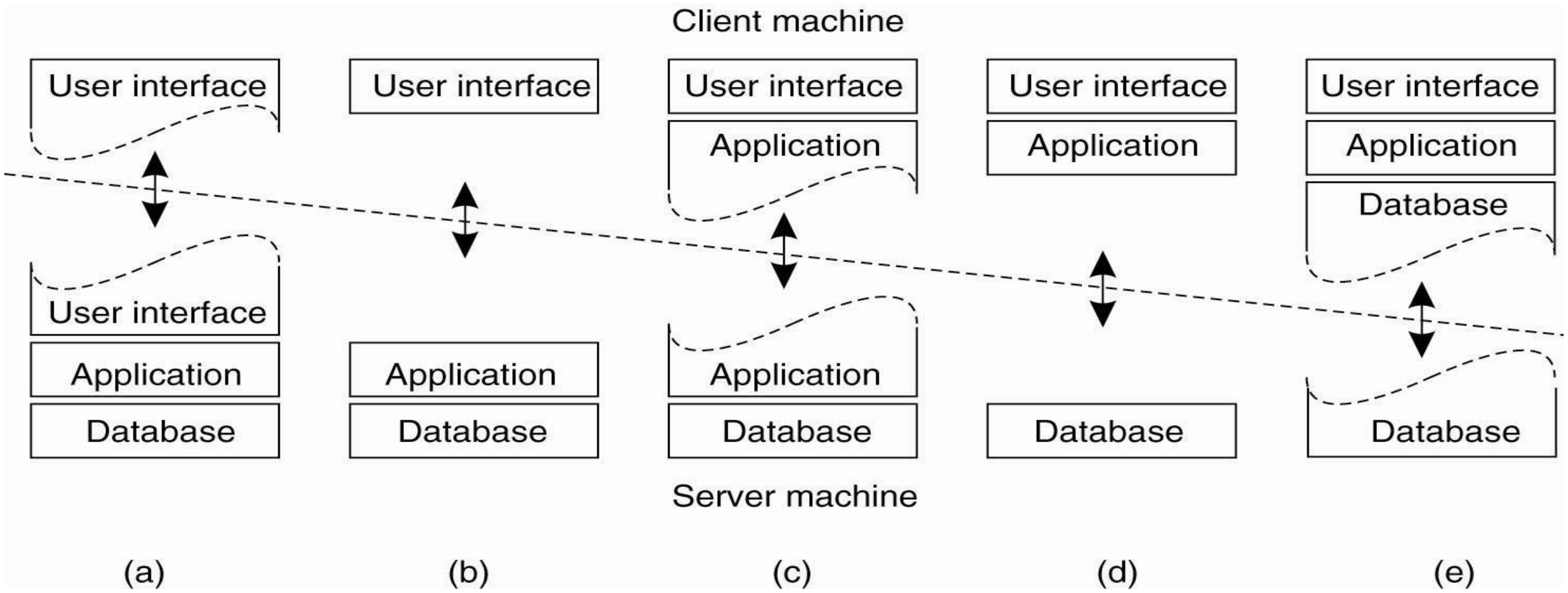
Two-tiered: Thin vs Fat client

- Thin client
  - Server provides processing and data management
  - Client provides a simple graphical display
  - No performance issues with the client
  - Easier to manage, more reliable, less money for the helpdesk

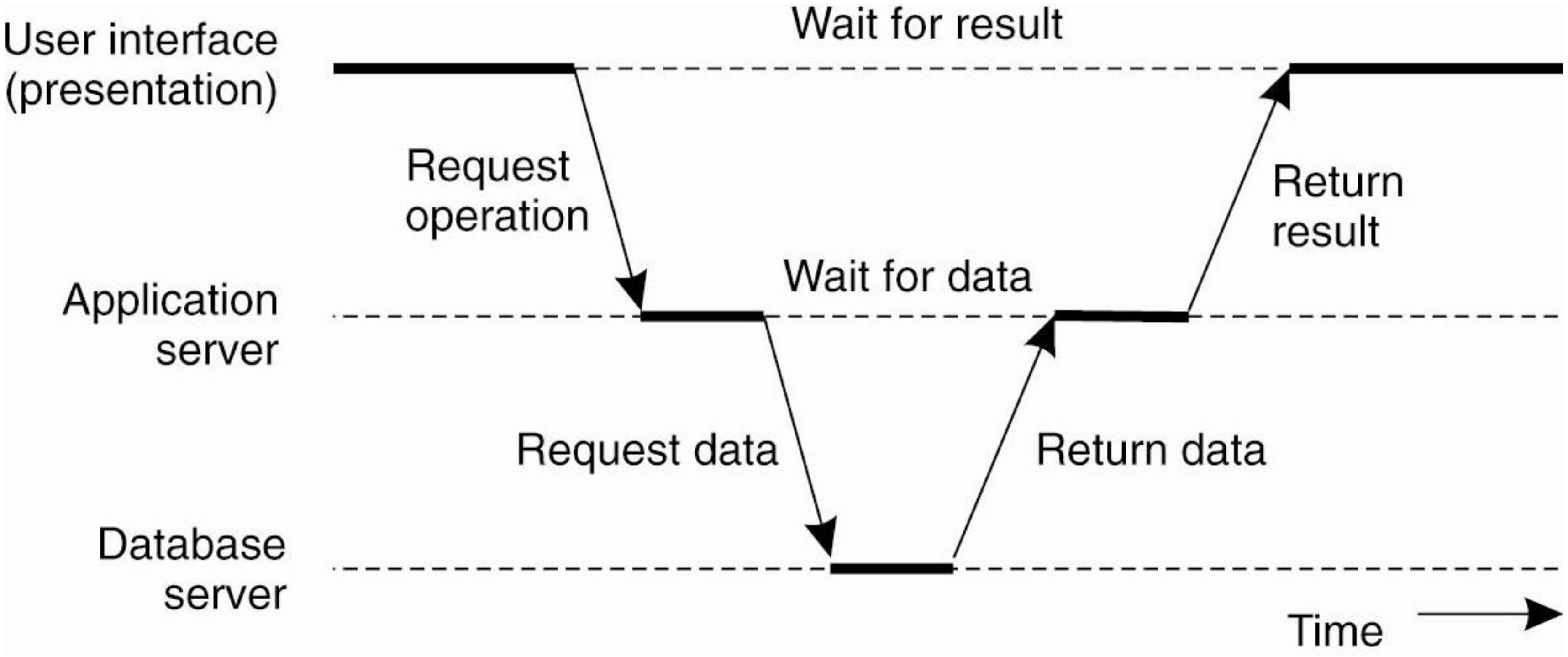
- Fat client
  - Some data processing on the client
  - Reduces workload at a server; more scalable
  - Harder to manage by a system administrator

A Spectrum of Choices

- Single-tiered – mainframe configuration
- Two-tiered – client/single server configuration
- Three-tiered – several server configurations



Three-tier Web Applications



Decentralized Architectures

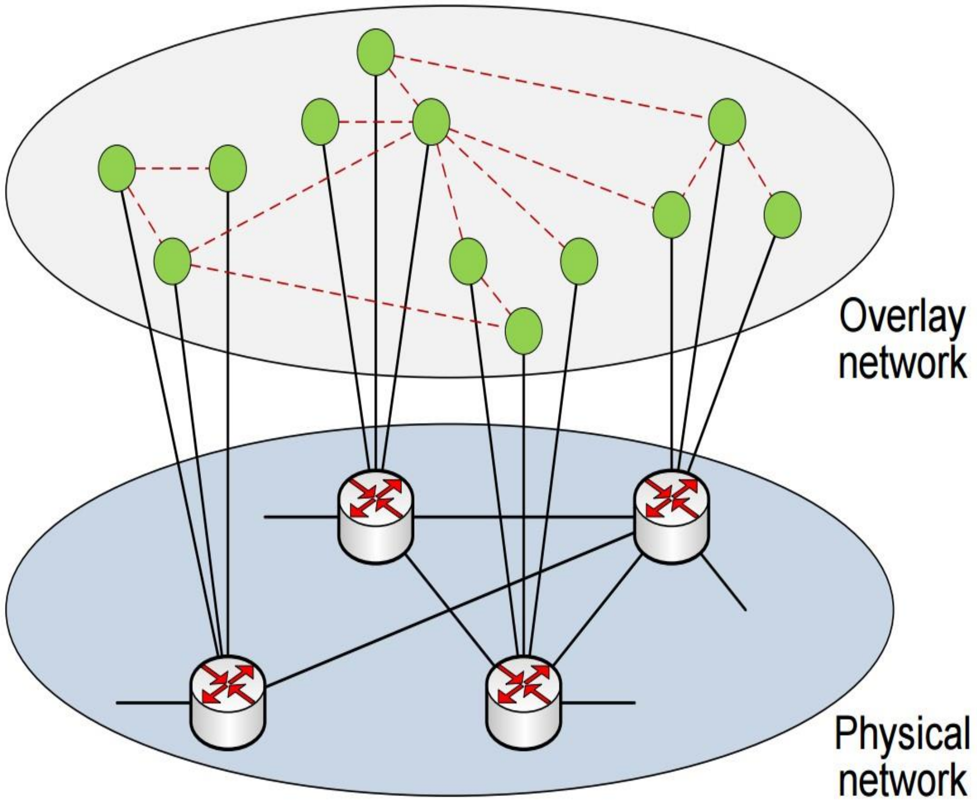
- **Structured P2P**: nodes are organized as a specific distributed structure
- **Unstructured P2P**: nodes randomly select neighbors
- **Hierarchical P2P**: some nodes have special functions

The overlay network connects nodes in the P2P system.

- Nodes in the overlay use their own addressing system for data handling
- Nodes may forward requests to locations that may not be known by the requester

Overlay Network

- Logical network which built on top of a physical network
- Each node in the P2P system knows how to contact several other nodes
- The overlay network may be
  - structured (overlay network constructed using a deterministic procedure)
  - unstructured (randomized algorithms)



P2P System Characteristics

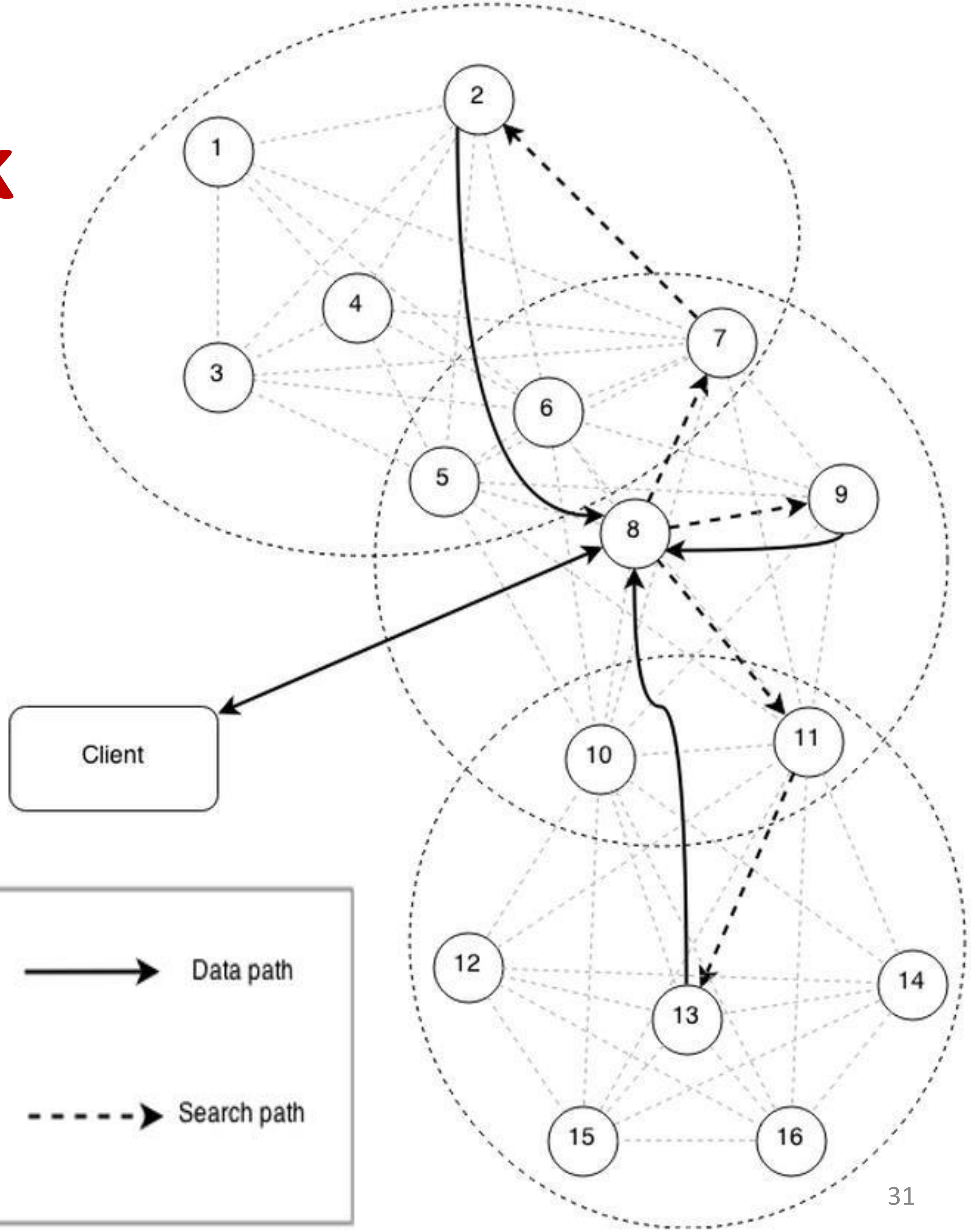
- The design ensures that each user contributes resources to the system
- All the nodes in a peer-to-peer system have the same functional capabilities and responsibilities
- The correct operation does not depend on the existence of any centrally administered systems (pure p2p)
- Key issues & challenges
  - Choice of an algorithm for the placement of data across many hosts
  - Balance the workload and ensures availability without adding extra overhead

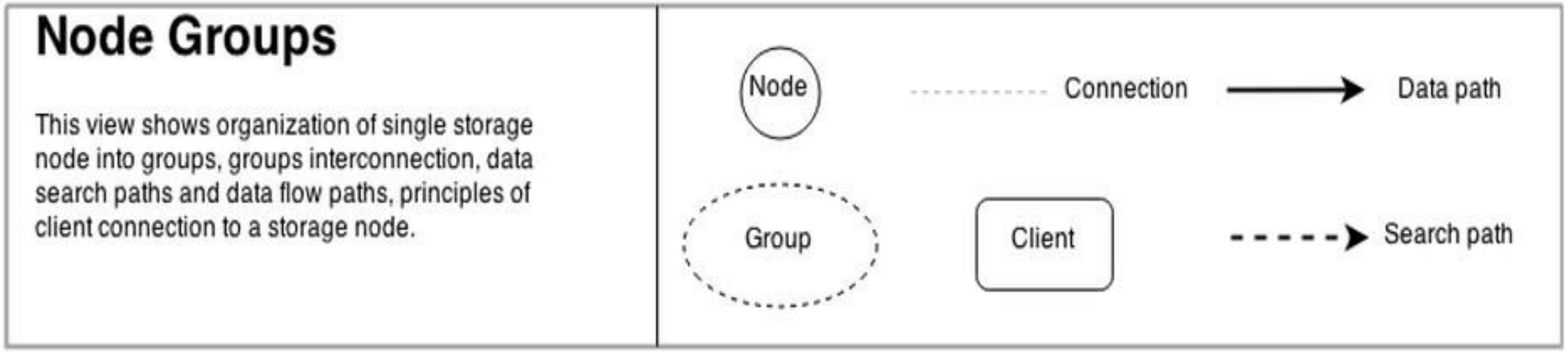
Schema of a P2P Network

Nodes can:

- join the system
- leave the system
- publish a resource
- search a resource
- unplublish a resource

Resources are replicated





Structured P2P Network

- **Basic features** of structured P2P overlays:
  - Structure – to accommodate participating nodes and data in the overlay
    - Structured overlays use several different geometries (rings, trees, hypercubes, etc.)
  - Routing algorithm – to locate nodes in the overlay and insert/retrieve data to/from them
  - Join/leave mechanisms – to enable self-organization and fault tolerance
  - The primary goal is to enable the deterministic lookup (i.e., access guarantees with certain time bounds)

Example: Chord

- Chord was developed at MIT
- Originally published in 2001 at the Sigcomm conference
- Chord’s overlay routing principle quite easy to understand
    - Paper has mathematical proofs of correctness and performance
- Uses DHT to locate objects
- Many projects at MIT around Chord

Unstructured P2P

- Unstructured P2P organizes the overlay network by using randomized Algorithms
- Each node knows about a subset of nodes
  - The subset could be chosen in very different ways: physically close nodes, nodes that joined at about the same time, etc.
- Data randomly mapped to some nodes in the system

Unstructured P2P: Locating the data

- **Flooding**. Node A sends a lookup query to all its neighbors. A neighbor responds, or forwards (floods) the request.
  - Limited flooding (maximum number of forwarding) TTL
  - Probabilistic flooding (flood only with a certain probability)

- **Random walk**. Randomly select a neighbor B. If B has the answer, it replies, otherwise B randomly selects one of its neighbors.
  - Parallel random walk - works well with replicated data

Structured vs Unstructured

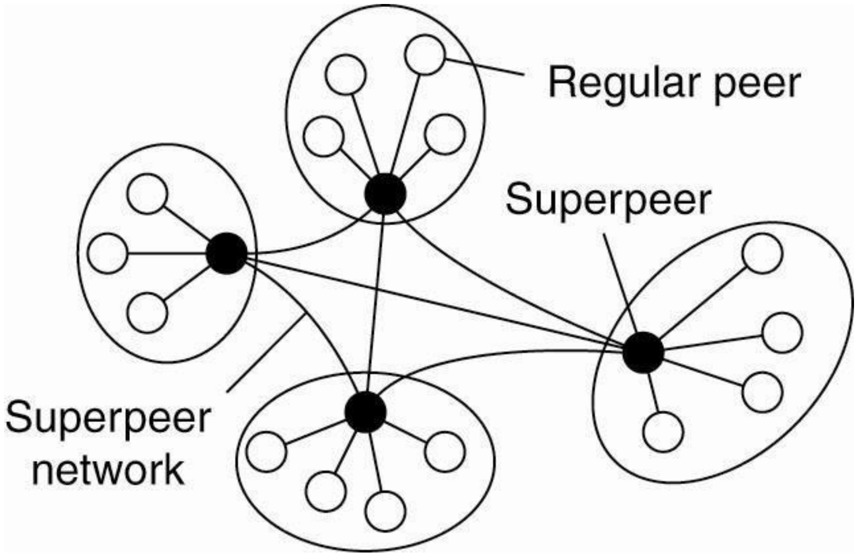
- Structured networks typically guarantee that if an object is in the network it will be available in a bounded amount of time

- Unstructured networks offer no guarantees.
  - For example, some will only forward search requests a specific number of hops
  - Random graph approach means there may be loops
  - Graph may become disconnected

Hierarchically organized P2P Networks

Superpeers (in some sources ultrapeers)

- Maintain indexes to some or all nodes in the system
  - Supports resource discovery
  - Act as servers to regular peer nodes, peers to other superpeers



- Improve scalability by controlling floods

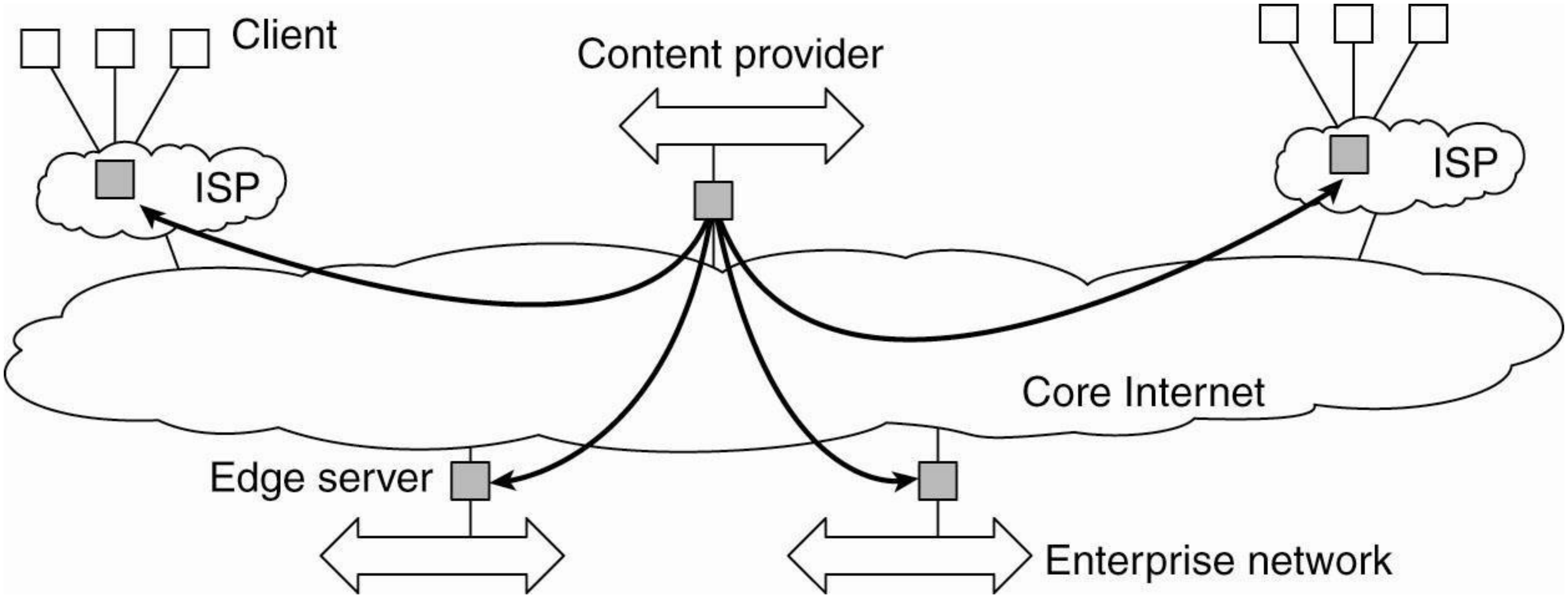
- Can also monitor the state of the network

- Example: Skype

Hybrid Architectures

- Combine client-server and P2P architectures
  - **Edge-server systems**; e.g., ISP, which act as servers to their clients, but cooperate with other edge servers to host shared content
  - **Collaborative distributed systems**; e.g., BitTorrent, which supports parallel downloading and uploading chunks of data.
    - First, interact with client-server system, then operate in decentralized manner

Edge-server Systems



Viewing the Internet as consisting of a collection of edge servers

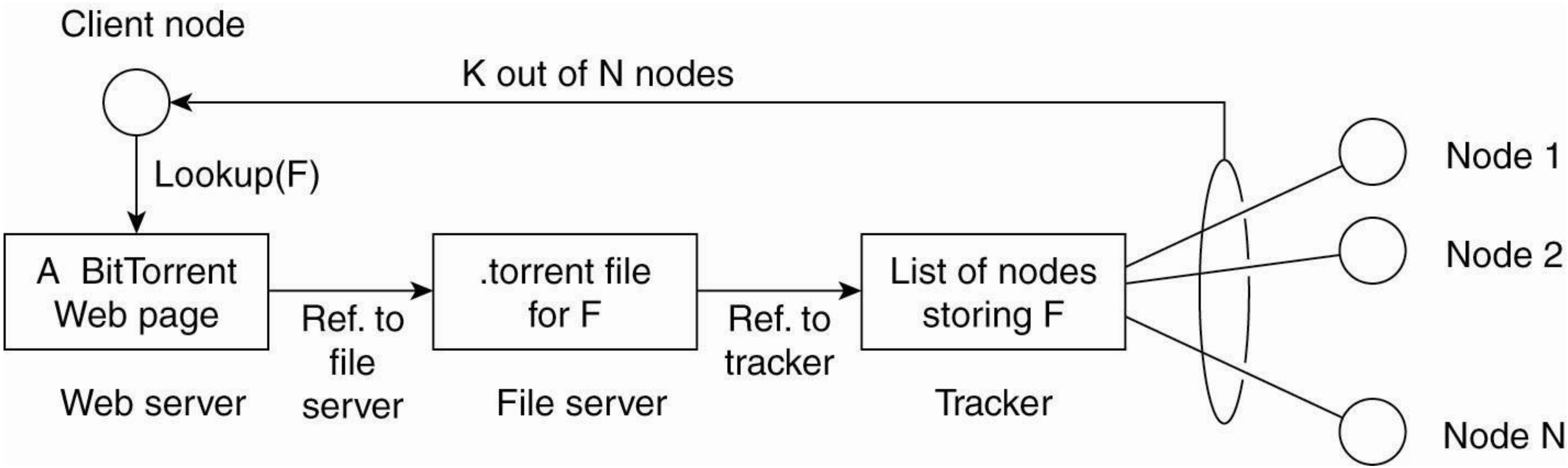
Collaborative Distributed Systems

Example: BitTorrent

- Clients contacts a global directory (Web server) to locate
    - a *.torrent* file with the information needed to locate a **tracker**
    - a server that can supply a list of active nodes (peers) that have chunks of the desired file
- Using this information, clients can download the file in chunks from multiple sites in the network

- Clients must also provide file chunks to other users

BitTorrent



Client node
K out of N nodes
Lookup(F)
A BitTorrent Web page
Web server
Ref. to file server
.torrent file for F
File server
Ref. to tracker
List of nodes storing F
Tracker
Node 1
Node 2
Node N

References

- Distributed Systems: Principles and Paradigms by Tanenbaum and van Steen, chapter 1
- Distributed Systems: Principles and Paradigms by Tanenbaum and van Steen, chapter 2

Thank you!

Questions?