Module 1
Introduction

What's a Distributed System?

A distributed system is a collection of independent computers that appear to the users of the system as a single computer

- Example:
  - ➡ a network of workstations allocated to users
  - ➡ a pool of processors in the machine room allocated dynamically
  - ➡ a single file system (all users access files with the same path name)
  - ➡ user command executed in the best place (user workstation, a workstation belonging to someone else, or on an unassigned processor in the machine room)

Why Distributed?

Economics	Microprocessors offer a better price/ performance than mainframes
Speed	A distributed system may have more total computing power than a mainframe
Inherent distribution	Some applications involve spatially separated machines
Reliability	If one machine crashes, the system as a whole can still survive
Incremental growth	Computing power can be added in small increments

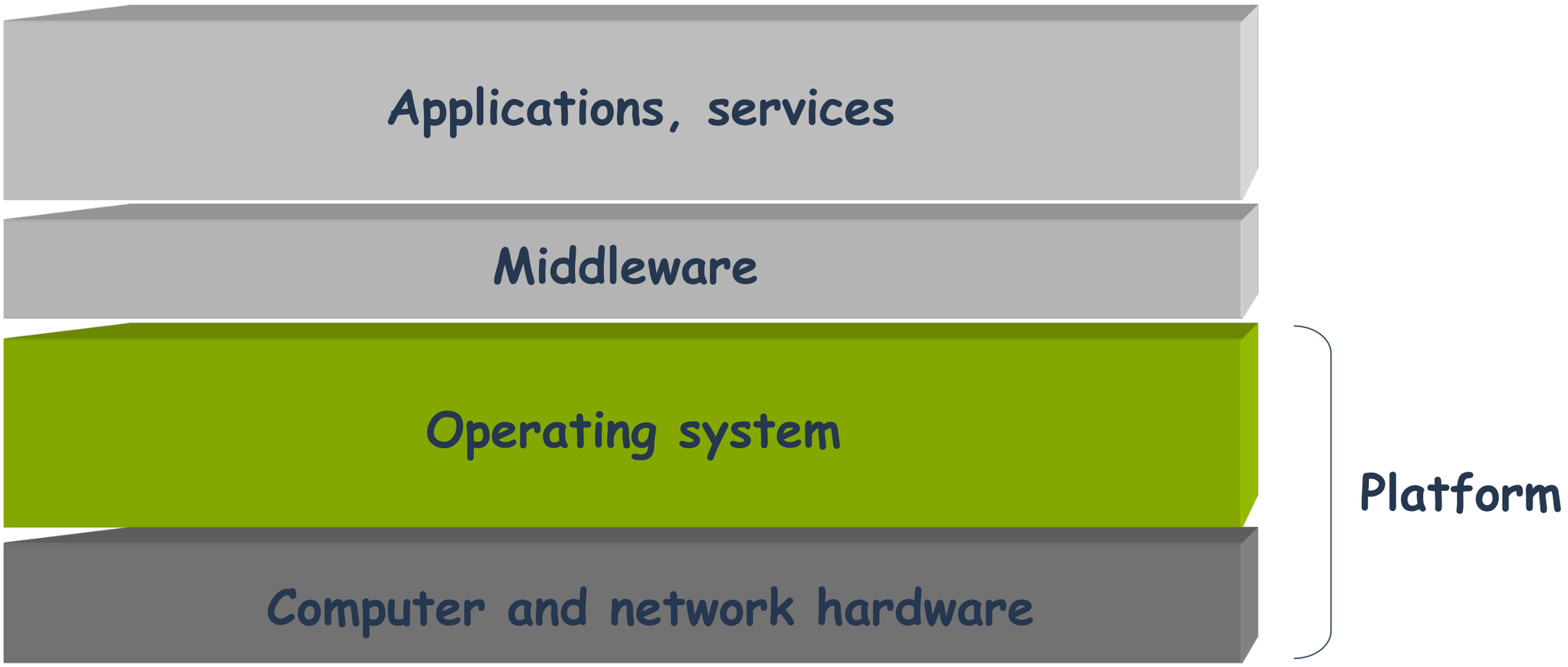
Primary Features

- Multiple computers
  - ➡ Concurrent execution
  - ➡ Independent operation and failures
- Communications
  - ➡ Ability to communicate
  - ➡ No tight synchronization (no global clock)
- "Virtual" Computer
  - ➡ Transparency

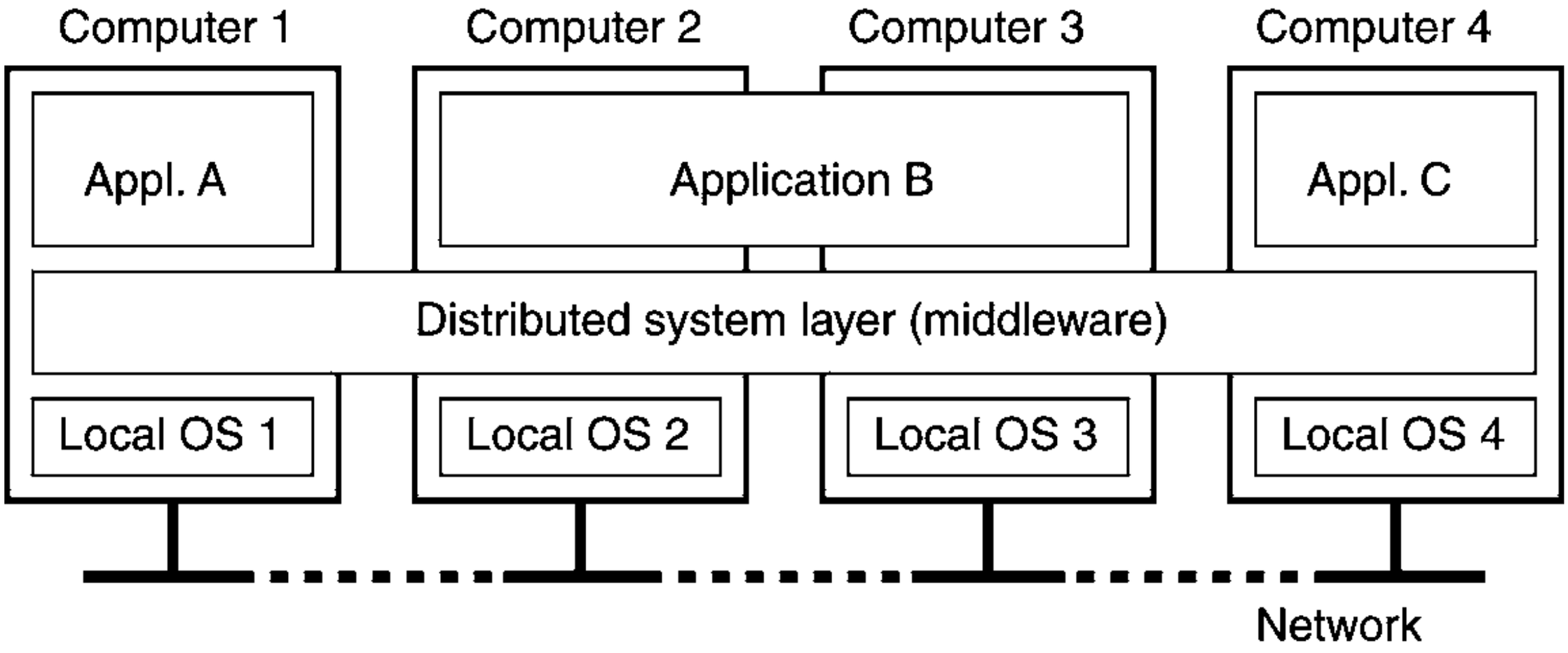
Types of Transparency

Access	local and remote resources are accessed using identical operations
Location	resources are accessed without knowledge of their location
Concurrency	several processes operate concurrently using shared resources without interference between them
Replication	multiple instances of resources appeare as a single instance
Failure	the concealment of faults from users
Mobility	the movement of resources and clients within a system (also called migration transparency)
Performance	the system can be reconfigured to improve performance
Scaling	the system and applications can expand in scale without change to the system structure or the application algorithms

Typical Layering in DSs

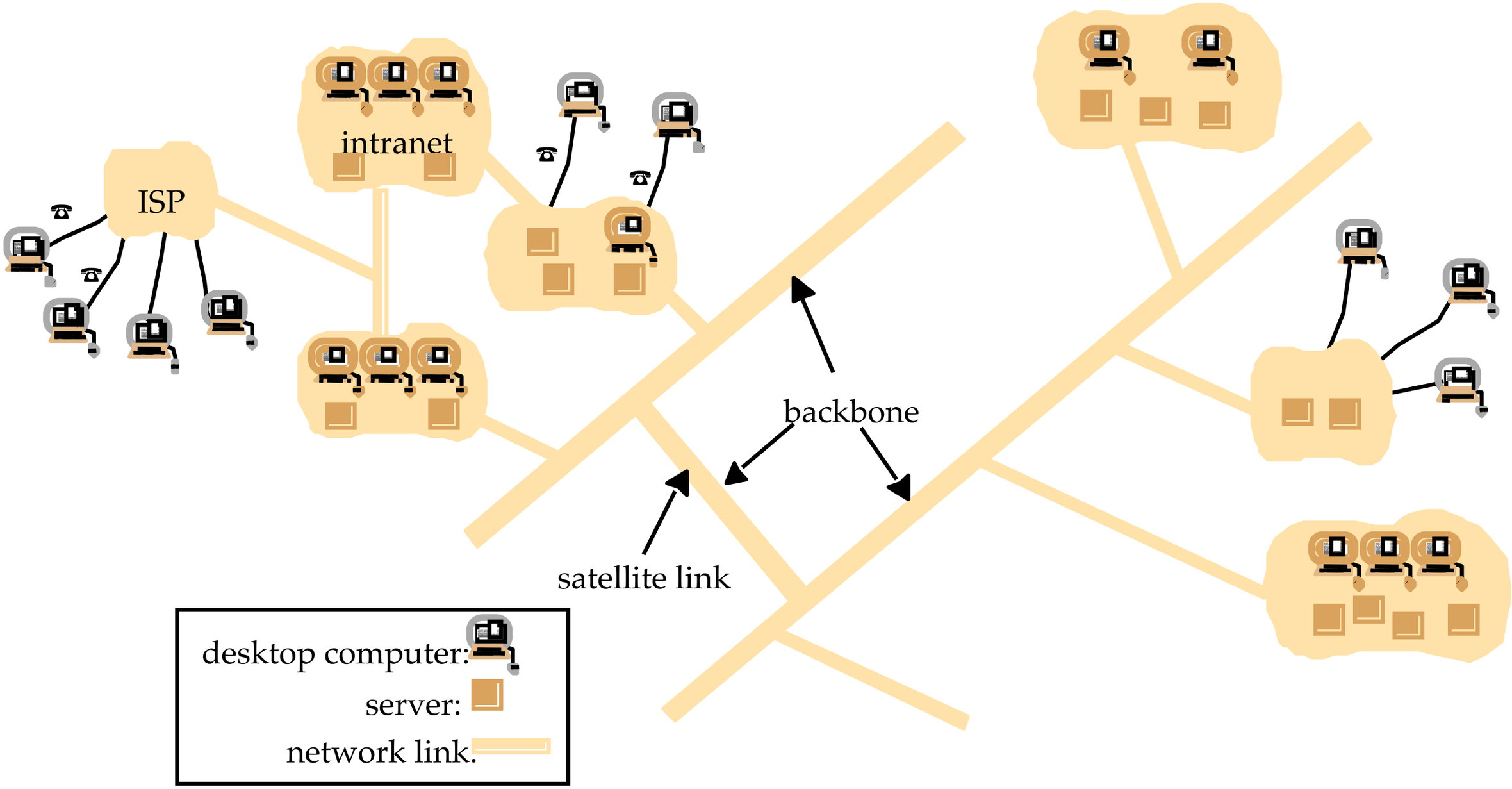


Distributed System as Middleware

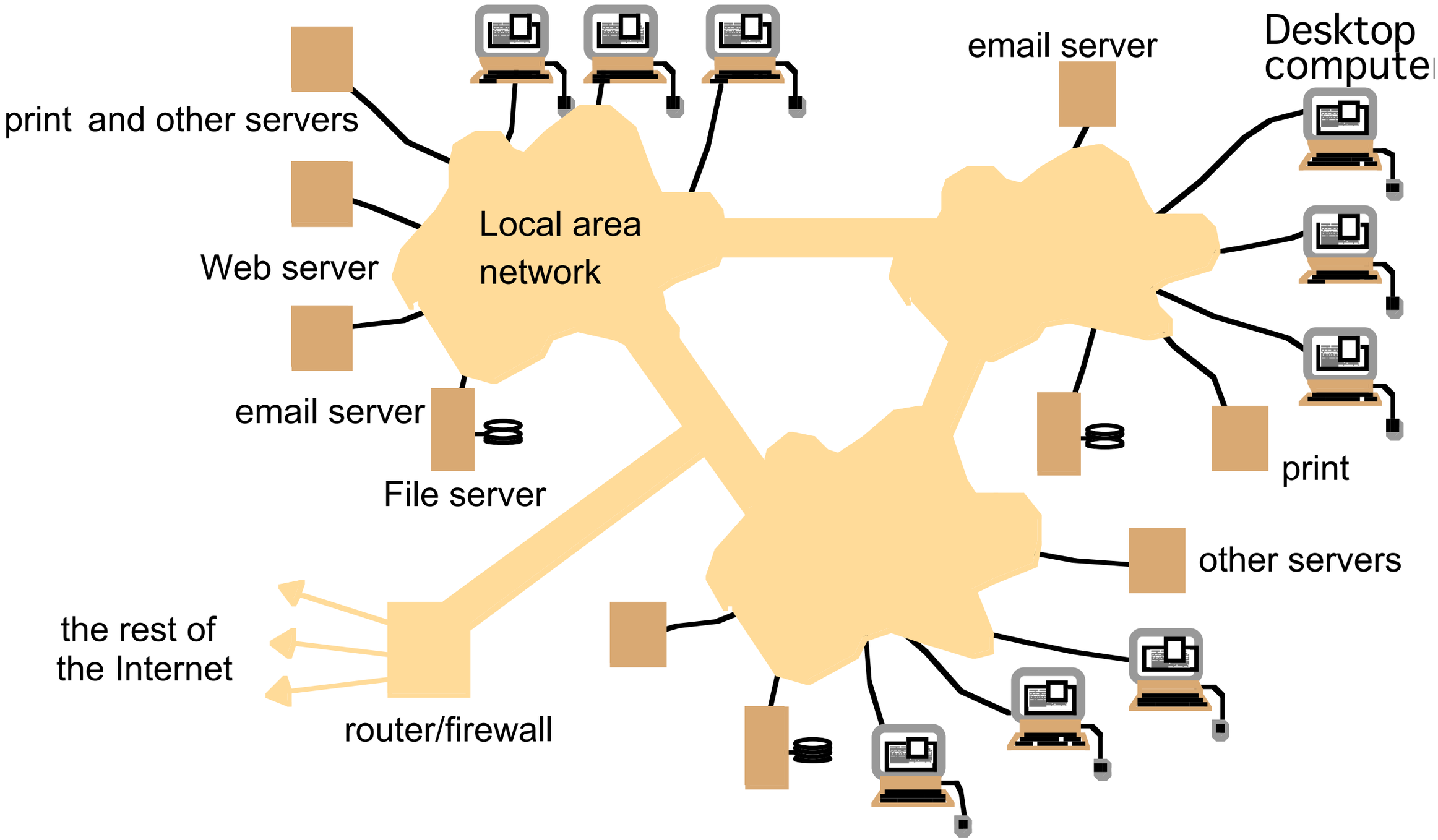


From Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e

Example – Internet (Portion of it)

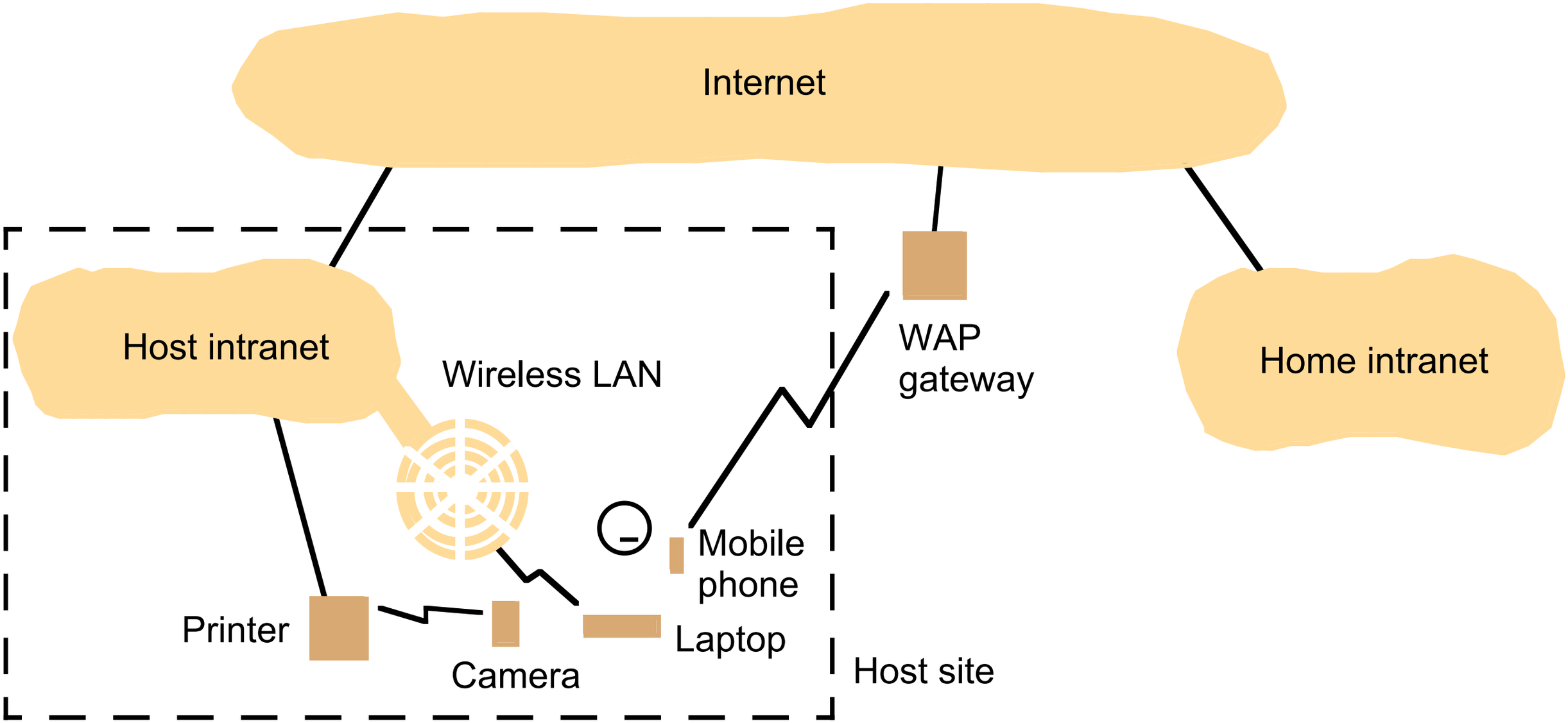


Example – An Intranet

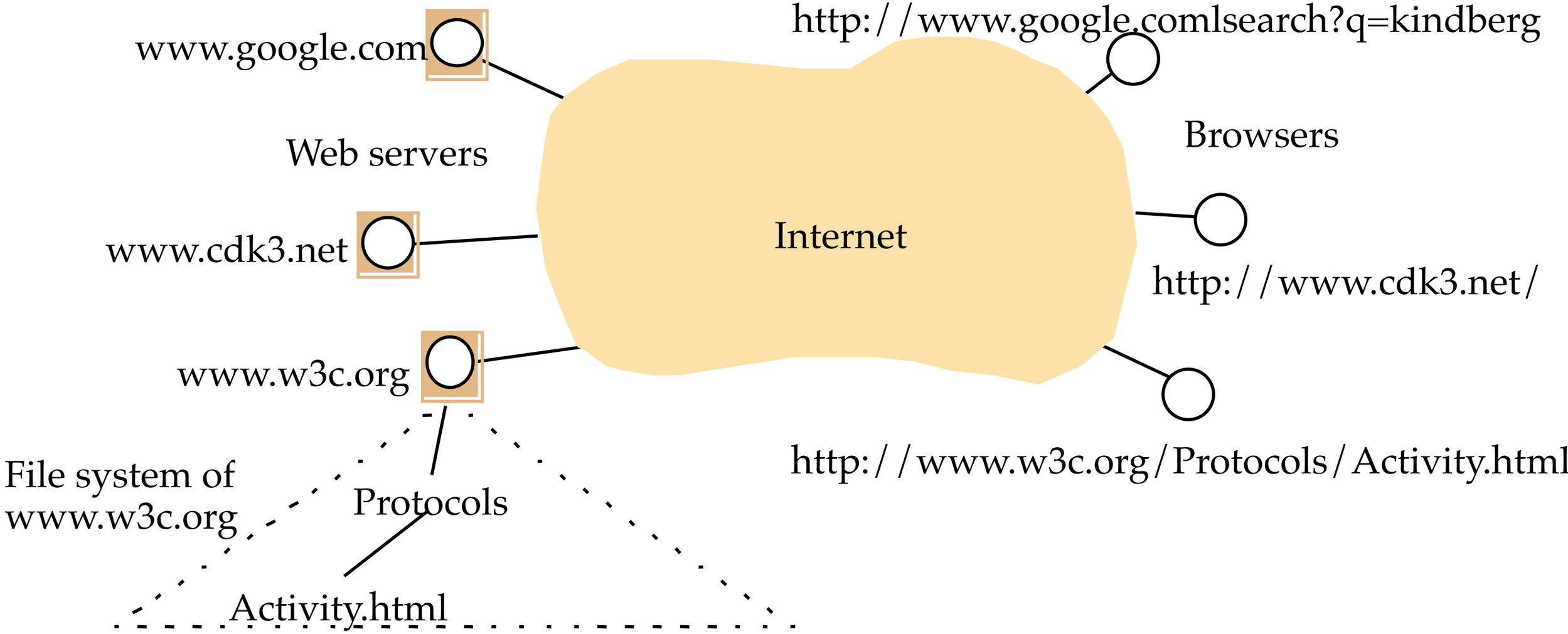


From Coulouris, Dollimore and Kindberg, Distributed Systems: Concepts and Design, 3rd ed.

Example – Mobile Environment



Example - Web



Challenges (I)

- Heterogeneity
  - Networks
  - Hardware
  - OS
  - Programming Languages
  - Implementations
- Openness
  - Can you extend and re-implement the system?
  - Public and published interfaces
  - Uniform communication mechanism

Challenges (II)

- Scalability
  - ➡ Can the system behave properly if the number of users and components increase?
    - ✦ Scale-up: increase the number of "users" while keeping the system performance unaffected
    - ✦ Speed-up: improvement in the system performance as system resources are increased
  - ➡ Impediments
    - ✦ Centralized data
      - ✓ A single file
    - ✦ Centralized services
      - ✓ A single server
    - ✦ Centralized algorithms
      - ✓ Algorithms that "know it all"

Challenges (III)

- Failure handling
  - ➡ Partial failures
    - ✦ Can non-failed components continue operation?
    - ✦ Can the failed components easily recover?
  - ➡ Failure detection
  - ➡ Failure masking
  - ➡ Failure tolerance
  - ➡ Recovery
  - ➡ An important technique is replication
    - ✦ Why does the space shuttle has 3 on-board computers?

Challenges (IV)

- Concurrency
  - Multiple "clients" sharing a resource ➔how to you maintain integrity of the resource and proper operation (without interference) of these clients?
  - Example: bank account
    - Assume your account has a balance of $100
    - You are depositing from an ATM a cheque for $50 into that account
    - Someone else is withdrawing from the same account $30 using another ATM but at the same time
    - What should the final balance of the account be?
      - $120
      - $70
      - $150

Challenges (V)

- Transparency
  - ➡ Not easy to maintain
  - ➡ Example:

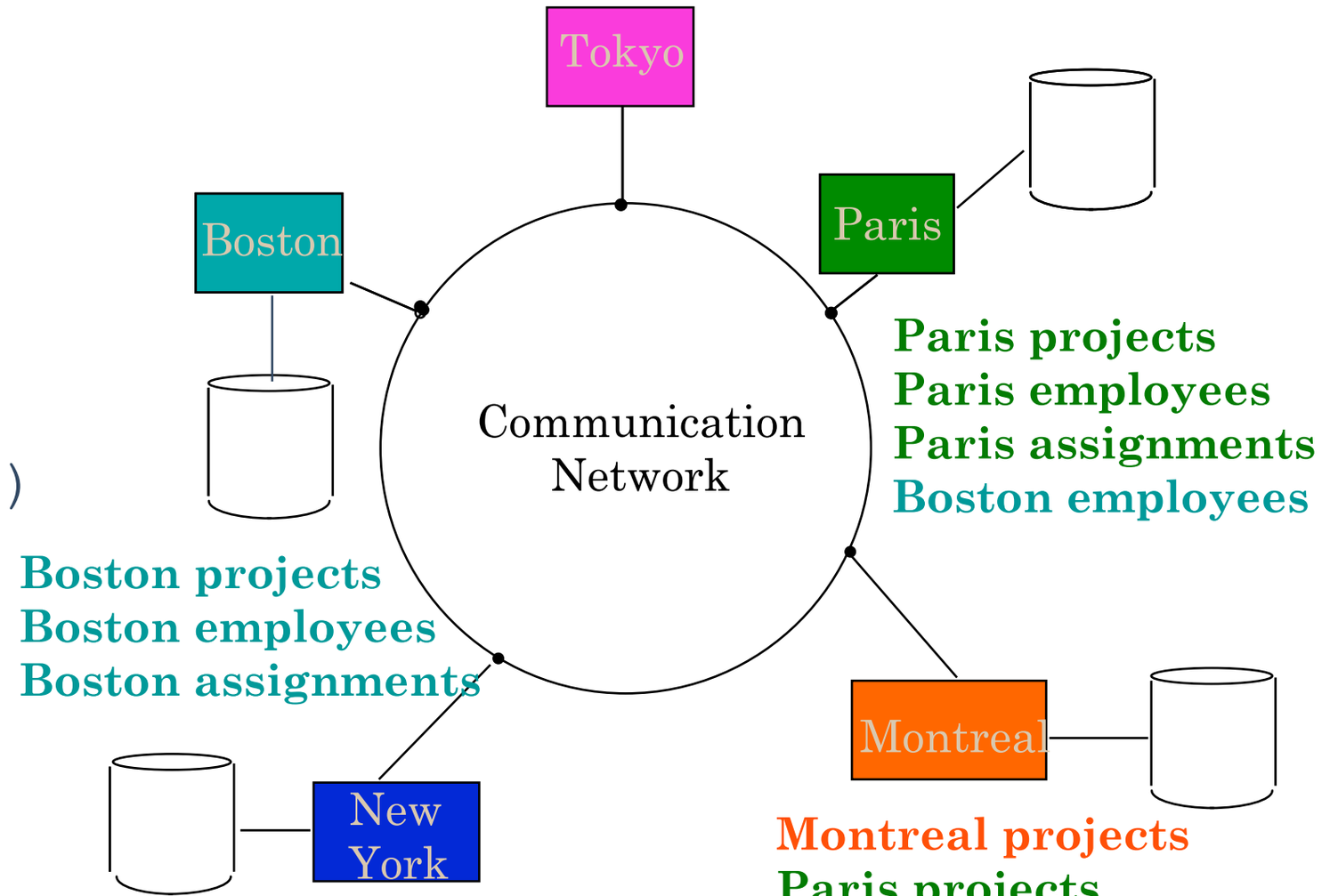
EMP (ENO, ENAME, TITLE, LOC)

PROJECT (PNO, PNAME, LOC)

PAY (TITLE, SAL)

ASG (ENO, PNO, DUR)

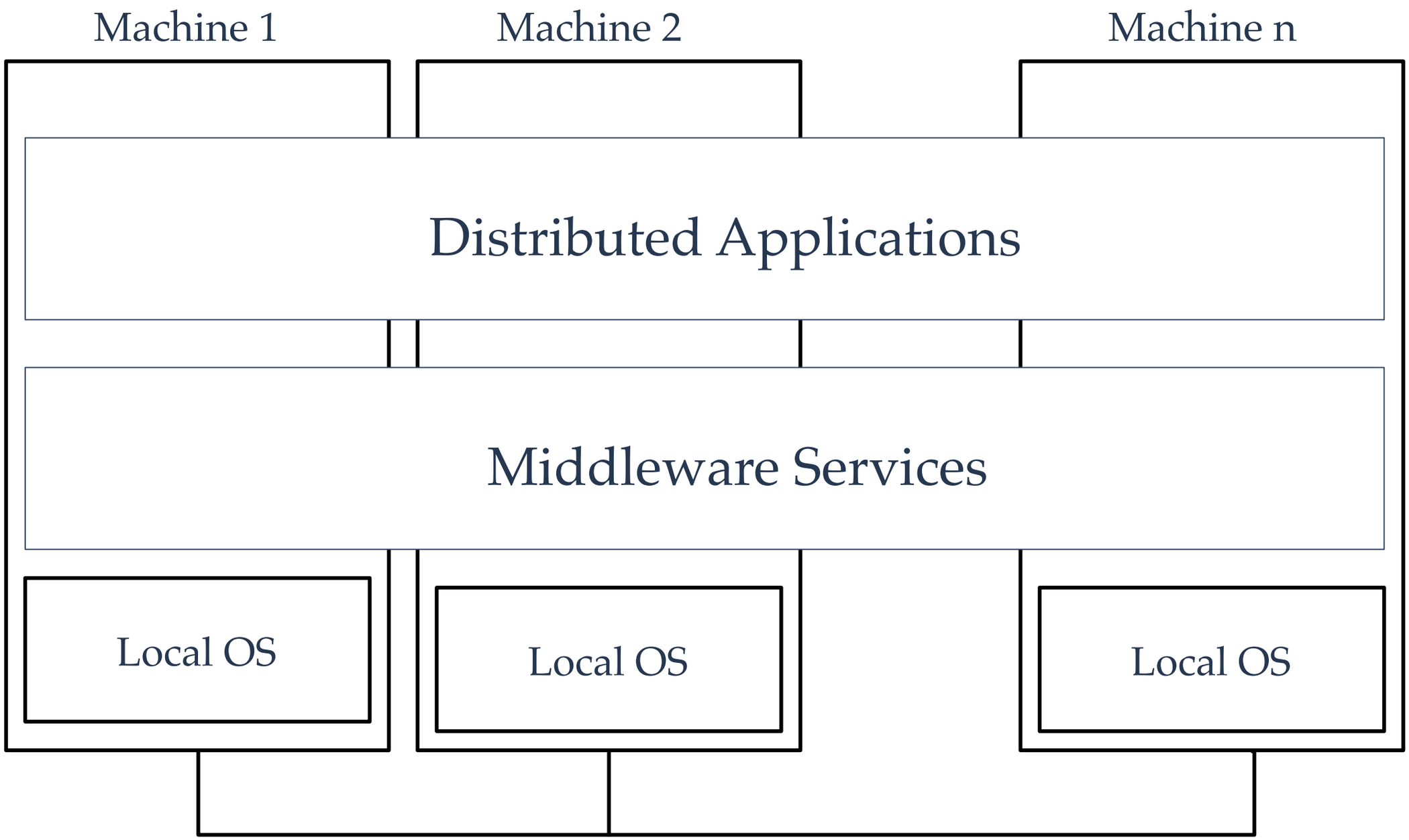
```
SELECT  ENAME,SAL
FROM    EMP,ASG,PAY
WHERE   DUR > 12
AND     EMP.ENO = ASG.ENO
AND     PAY.TITLE = EMP.TITLE
```



Pitfalls when Developing Distributed Systems

- False assumptions made by first time developer:
  - The network is reliable.
  - The network is secure.
  - The network is homogeneous.
  - The topology does not change.
  - Latency is zero.
  - Bandwidth is infinite.
  - Transport cost is zero.
  - There is one administrator.

Software Layers



Layers

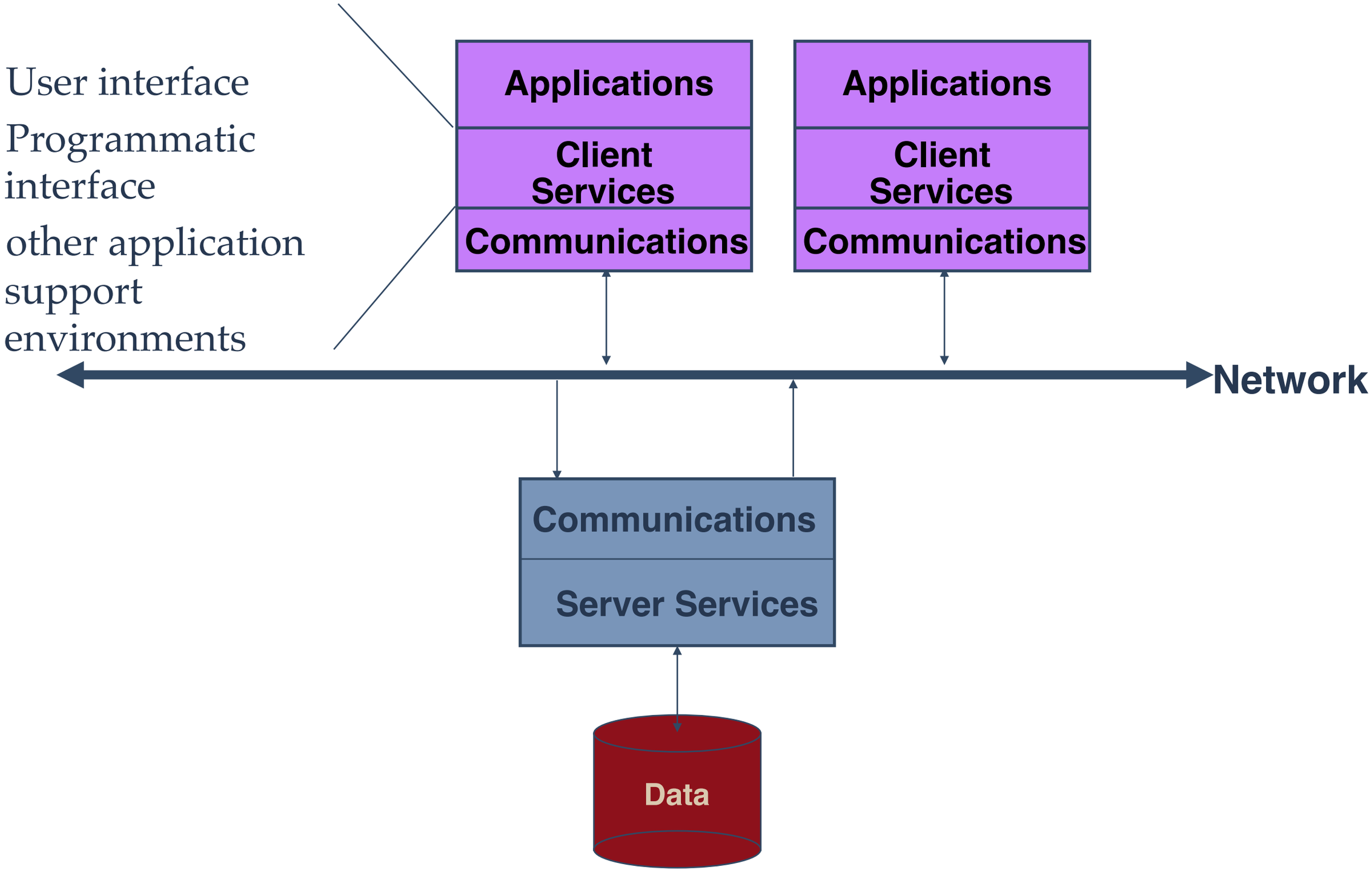
- Platform
  - ➡ Fundamental communication and resource management services
- Middleware
  - ➡ Provides a service layer that hides the details and heterogeneity of the underlying platform
  - ➡ Provides an “easier” API for the applications and services
  - ➡ Can be as simple as RPC or as complex as OMA
    - ✦ RPC (Remote Procedure Call): simple procedure call across remote machine boundaries
    - ✦ OMA (Object Management Architecture) : an object-oriented platform for building distributed applications
- Applications
  - ➡ Distributed applications, services
  - ➡ Examples: e-mail, ftp, etc

System Architectures

- Client-server
  - Multiple-client/single-server
  - Multiple-client/multiple-servers
  - Mobile clients
- Multitier systems
- Peer-to-peer systems

Multiple-Client/Single Server

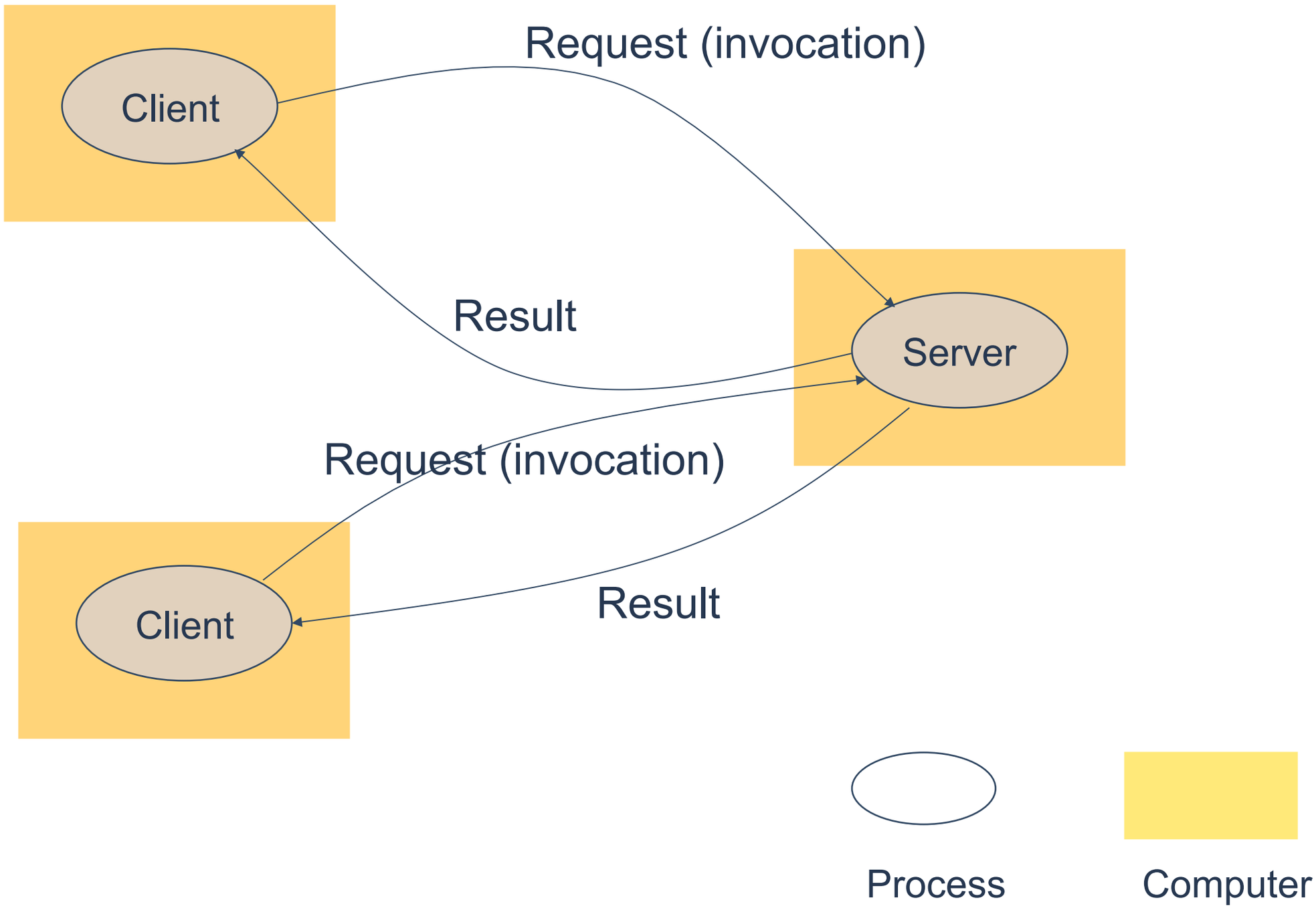
- User interface
- Programmatic interface
- other application support environments



Advantages of Client/Server Computing

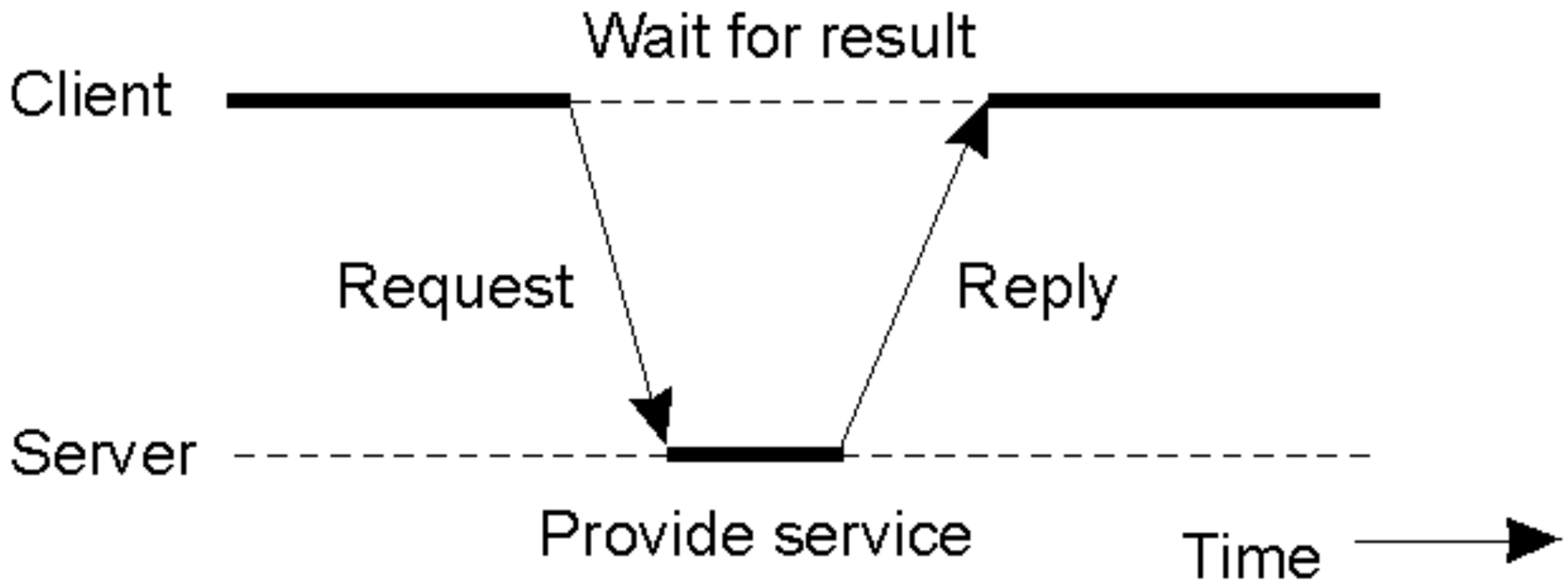
- More efficient division of labor
- Horizontal and vertical scaling of resources
- Better price/performance on client machines
- Ability to use familiar tools on client machines
- Client access to remote data (via standards)
- Full DBMS functionality provided to client workstations
- Overall better system price/performance

Client-Server Communication



Client-Server Timing

- General interaction between a client and a server.



An Example Client and Server (1)

- The *header.h* file used by the client and server.

```
/* Definitions needed by clients and servers.          */
#define TRUE              1
#define MAX_PATH          255   /* maximum length of file name          */
#define BUF_SIZE          1024  /* how much data to transfer at once    */
#define FILE_SERVER       243   /* file server's network address        */

/* Definitions of the allowed operations */
#define CREATE            1     /* create a new file                    */
#define READ              2     /* read data from a file and return it  */
#define WRITE             3     /* write data to a file                 */
#define DELETE            4     /* delete an existing file              */
```

```
/* Error codes. */
#define OK                0     /* operation performed correctly        */
#define E_BAD_OPCODE      -1    /* unknown operation requested          */
#define E_BAD_PARAM       -2    /* error in a parameter                 */
#define E_IO              -3    /* disk error or other I/O error        */

/* Definition of the message format. */
struct message {
      long source;              /* sender's identity                    */
      long dest;                /* receiver's identity                  */
      long opcode;              /* requested operation                  */
      long count;               /* number of bytes to transfer          */
      long offset;              /* position in file to start I/O        */
      long result;              /* result of the operation              */
      char name[MAX_PATH];      /* name of file being operated on       */
      char data[BUF_SIZE];      /* data to be read or written           */
};
```

From Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms

An Example Client and Server (2)

- A sample server.

```
#include <header.h>
void main(void) {
     struct message ml, m2;          /* incoming and outgoing messages          */
     int r;                          /* result code                             */

     while(TRUE) {                   /* server runs forever                     */
          receive(FILE_SERVER, &ml);      /* block waiting for a message        */
          switch(ml.opcode) {             /* dispatch on type of request        */
               case CREATE:    r = do_create(&ml, &m2); break;
               case READ:      r = do_read(&ml, &m2); break;
               case WRITE:     r = do_write(&ml, &m2); break;
               case DELETE:    r = do_delete(&ml, &m2); break;
               default:        r = E_BAD_OPCODE;
          }
          m2.result = r;                  /* return result to client            */
          send(ml.source, &m2);           /* send reply                         */
     }
}
```

An Example Client and Server (3)

- A client using the server to copy a file.

(a)

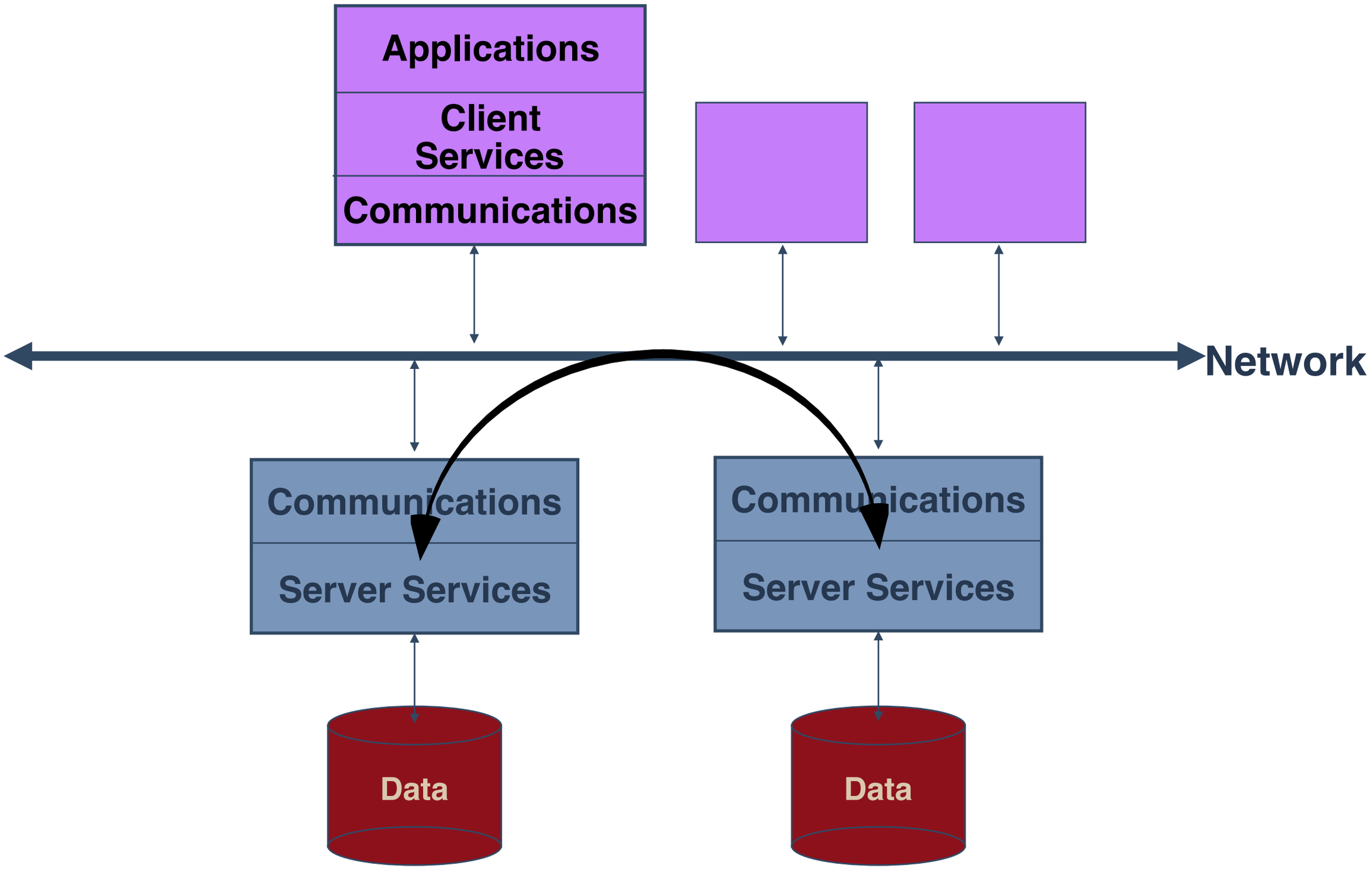
```
#include <header.h>
int copy(char *src, char *dst){
     struct message ml;                    /* procedure to copy file using the server     */
     long position;                        /* message buffer                              */
     long client = 110;                    /* current file position                       */
                                           /* client's address                            */

     initialize( );                        /* prepare for execution                       */
     position = 0;
     do {
          ml.opcode = READ;                /* operation is a read                         */
          ml.offset = position;            /* current position in the file                */
          ml.count  = BUF_SIZE;                                                            /* how many bytes to read*/
          strcpy(&ml.name, src);           /* copy name of file to be read to message     */
          send(FILESERVER, &ml);           /* send the message to the file server         */
          receive(client, &ml);            /* block waiting for the reply                 */

          /* Write the data just received to the destination file.                        */
          ml.opcode = WRITE;               /* operation is a write                        */
          ml.offset = position;            /* current position in the file                */
          ml.count  = ml.result;           /* how many bytes to write                     */
          strcpy(&ml.name, dst);           /* copy name of file to be written to buf      */
          send(FILE_SERVER, &ml);          /* send the message to the file server         */
          receive(client, &ml);            /* block waiting for the reply                 */
          position += ml.result;           /* ml.result is number of bytes written        */
     } while( ml.result > 0 );             /* iterate until done                          */
     return(ml.result >= 0 ? OK : ml result);  /* return OK or error code                */
}
```

From Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms

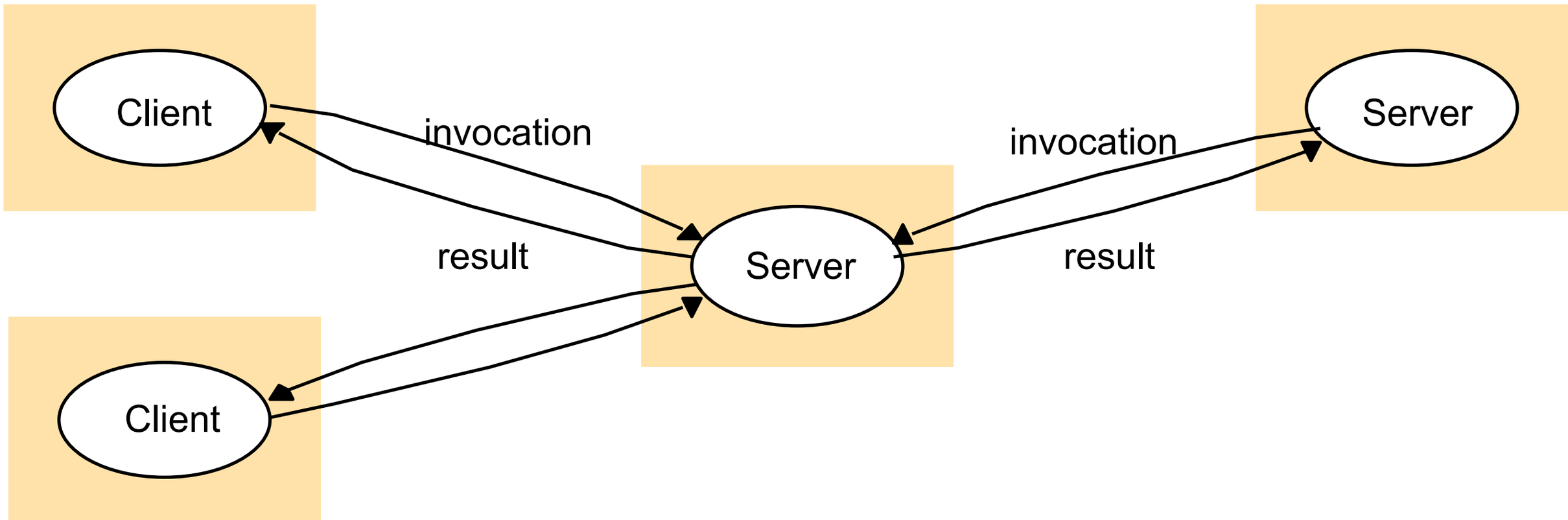
Problems With Multiple-Client/Single Server

- Server forms bottleneck
- Server forms single point of failure
- System scaling difficult

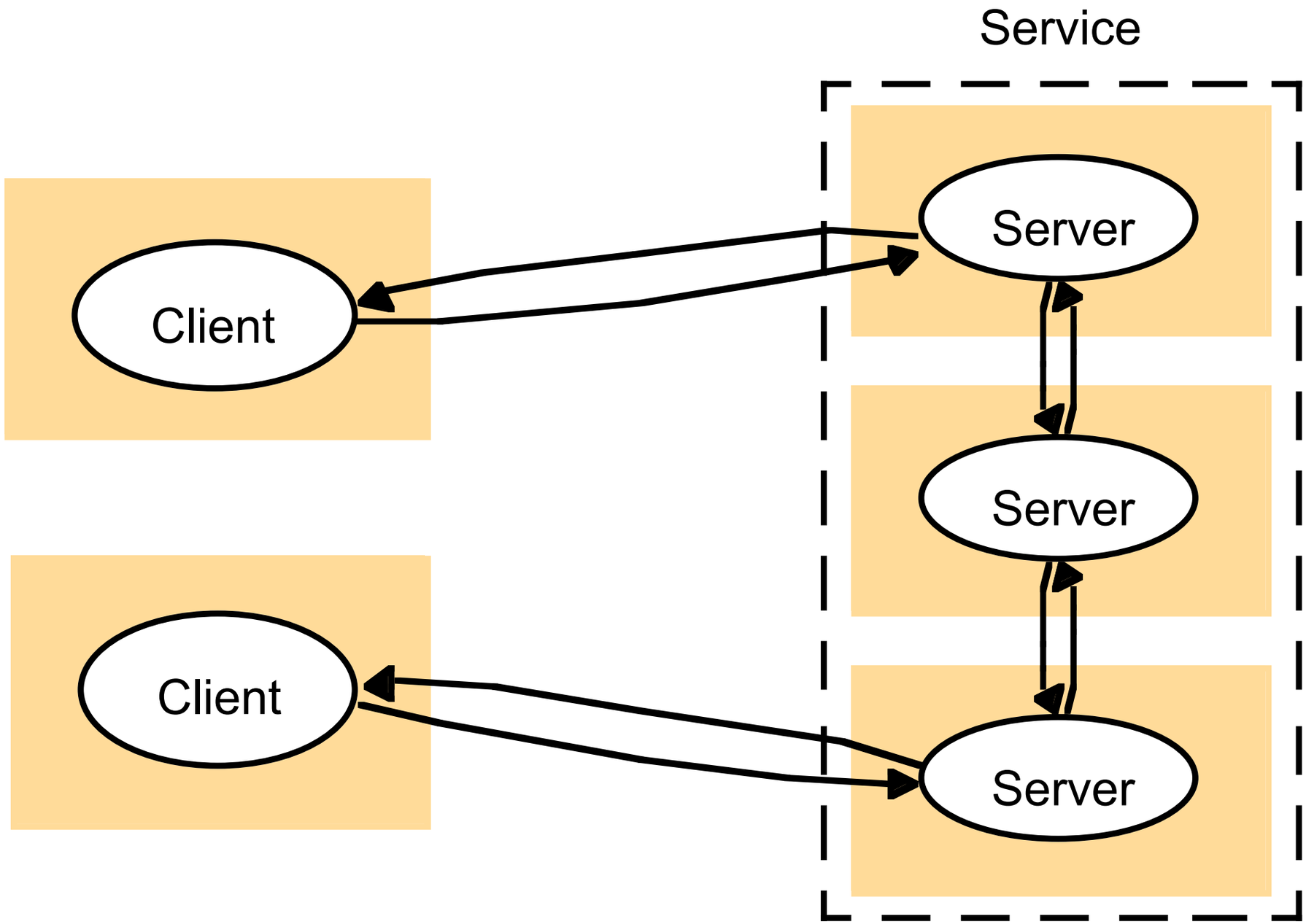
Multiple Clients/Multiple Servers



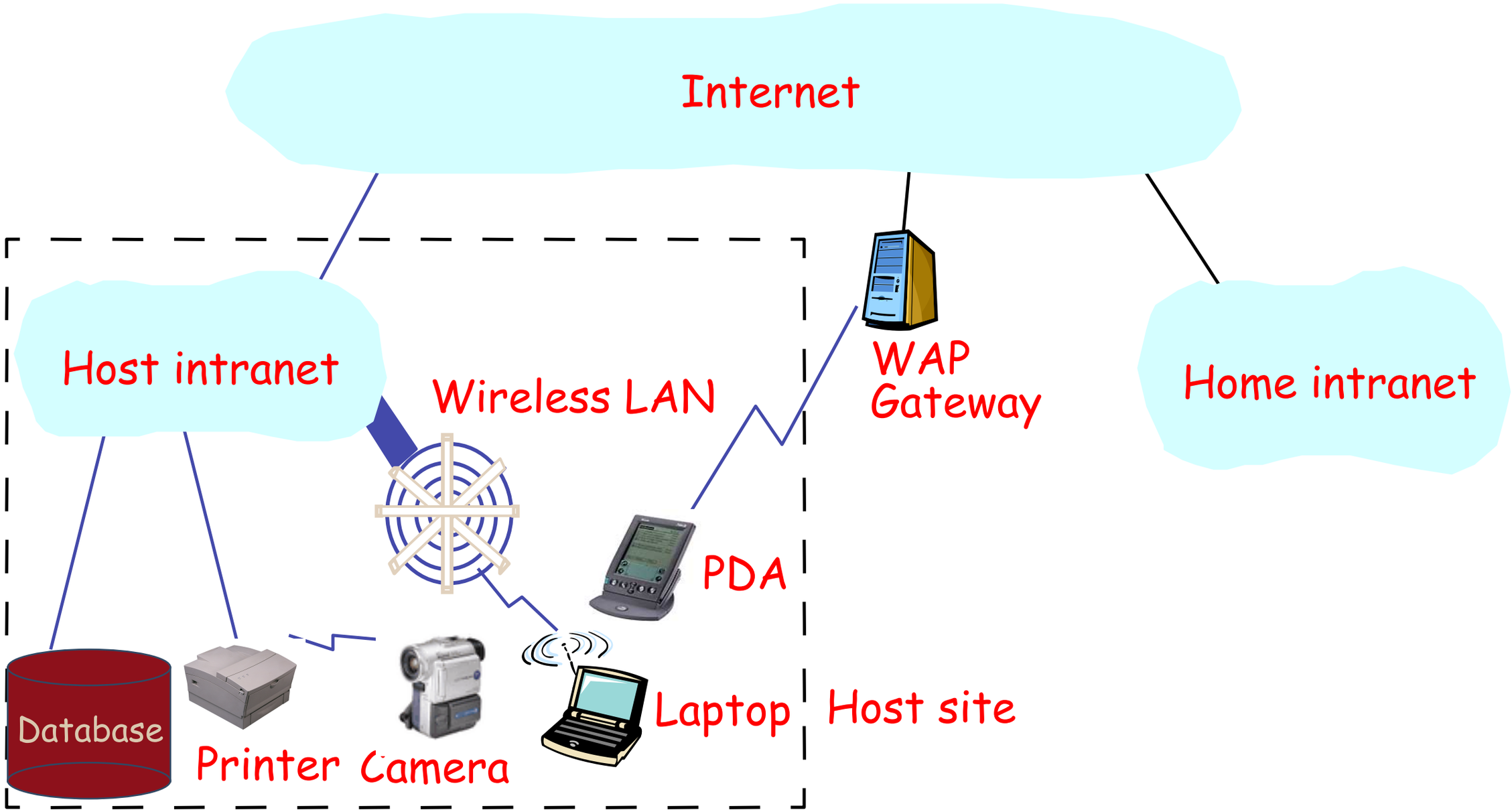
Multiple-Client/Multiple-Server Communication



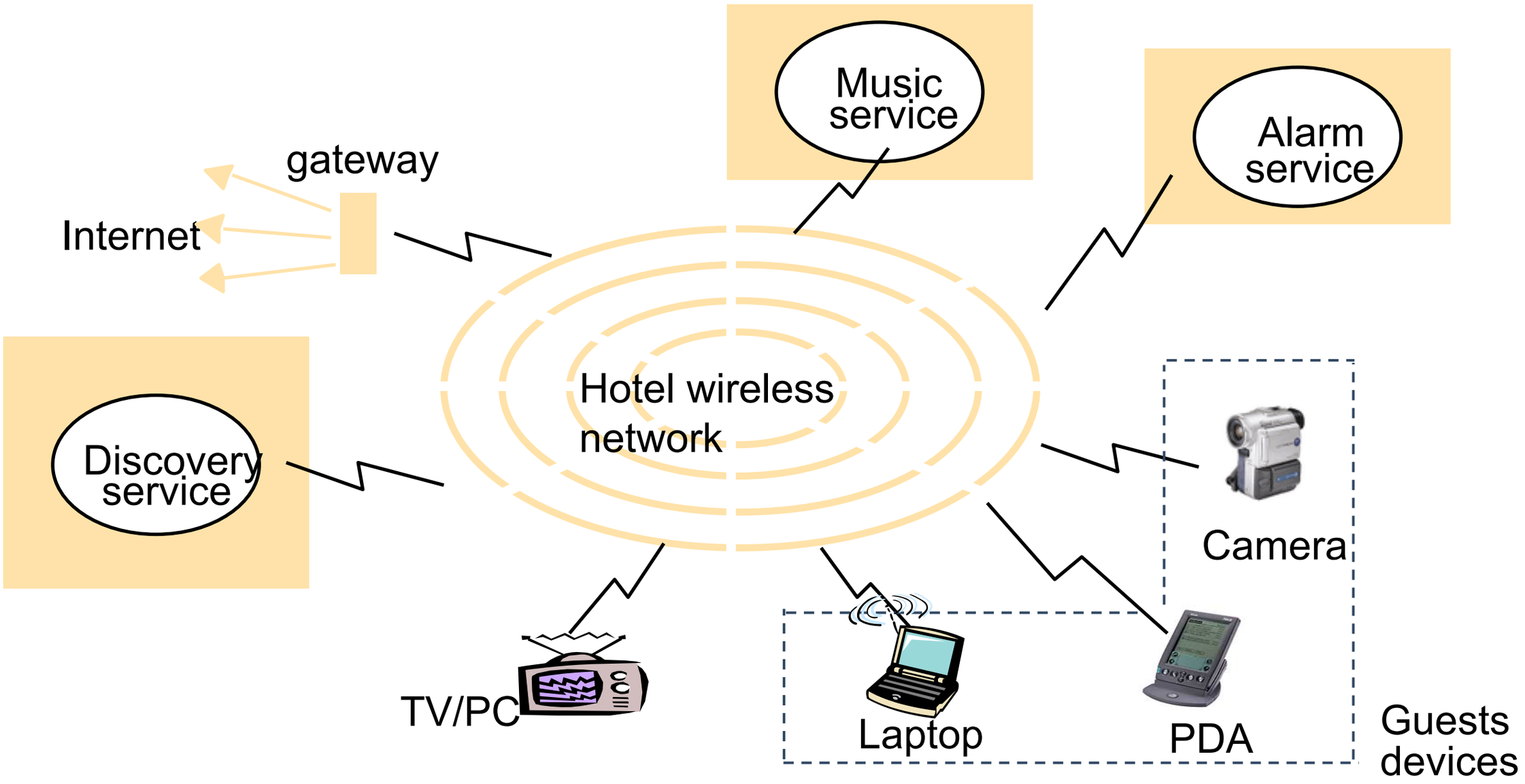
Service Across Multiple Servers



Mobile Computing



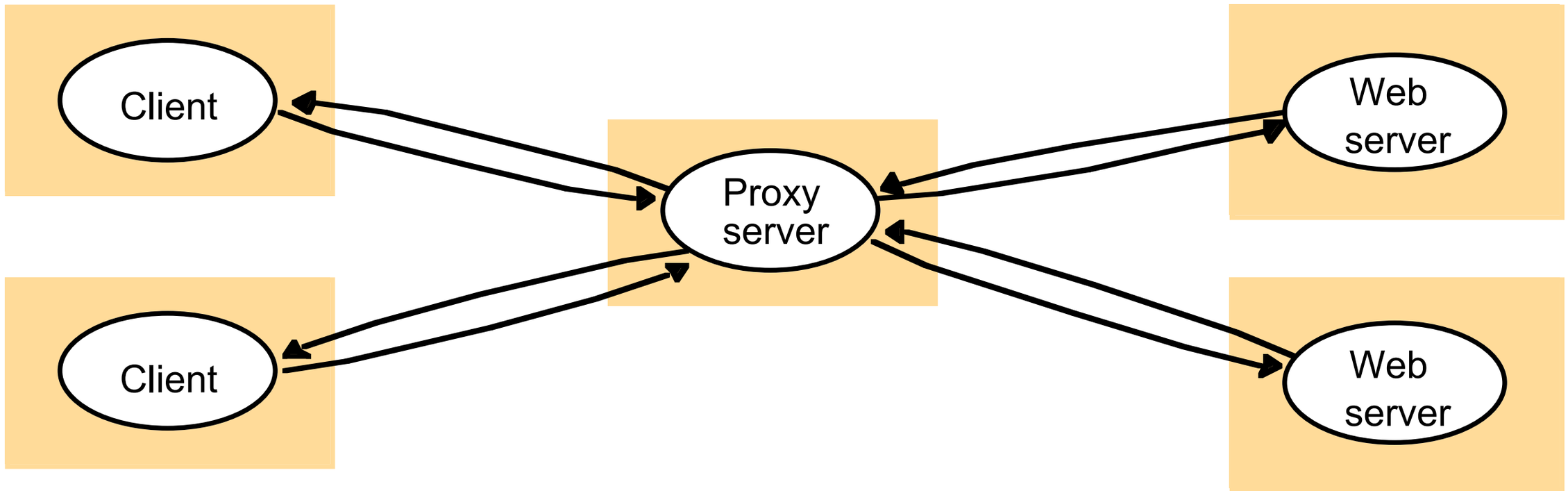
Example Mobile Computing Environment



These types of environments are commonly called “spontaneous systems” or “pervasive computing”

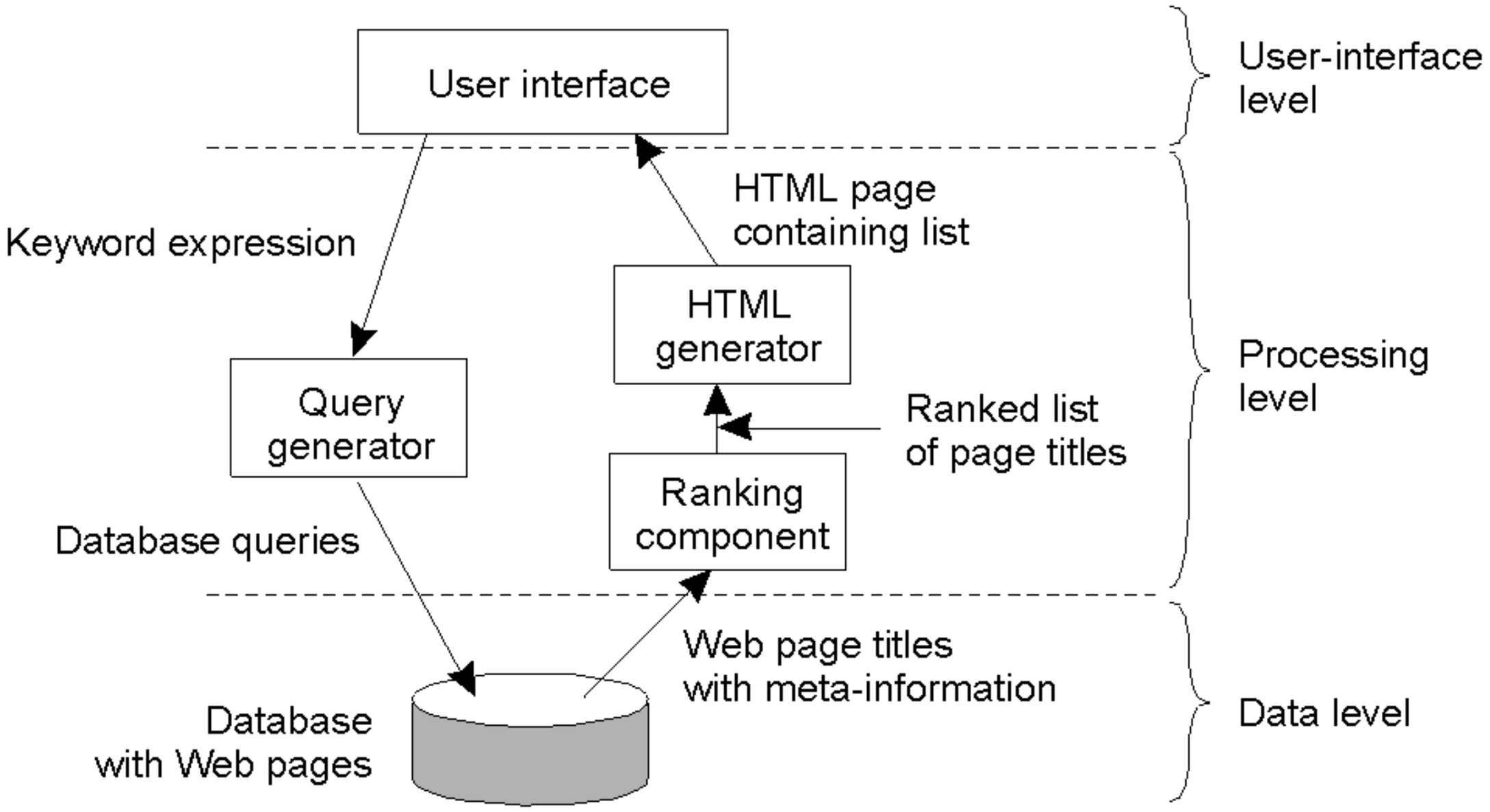
Multitier Systems

- Servers are clients of other servers
- Example:
  - Web proxy servers



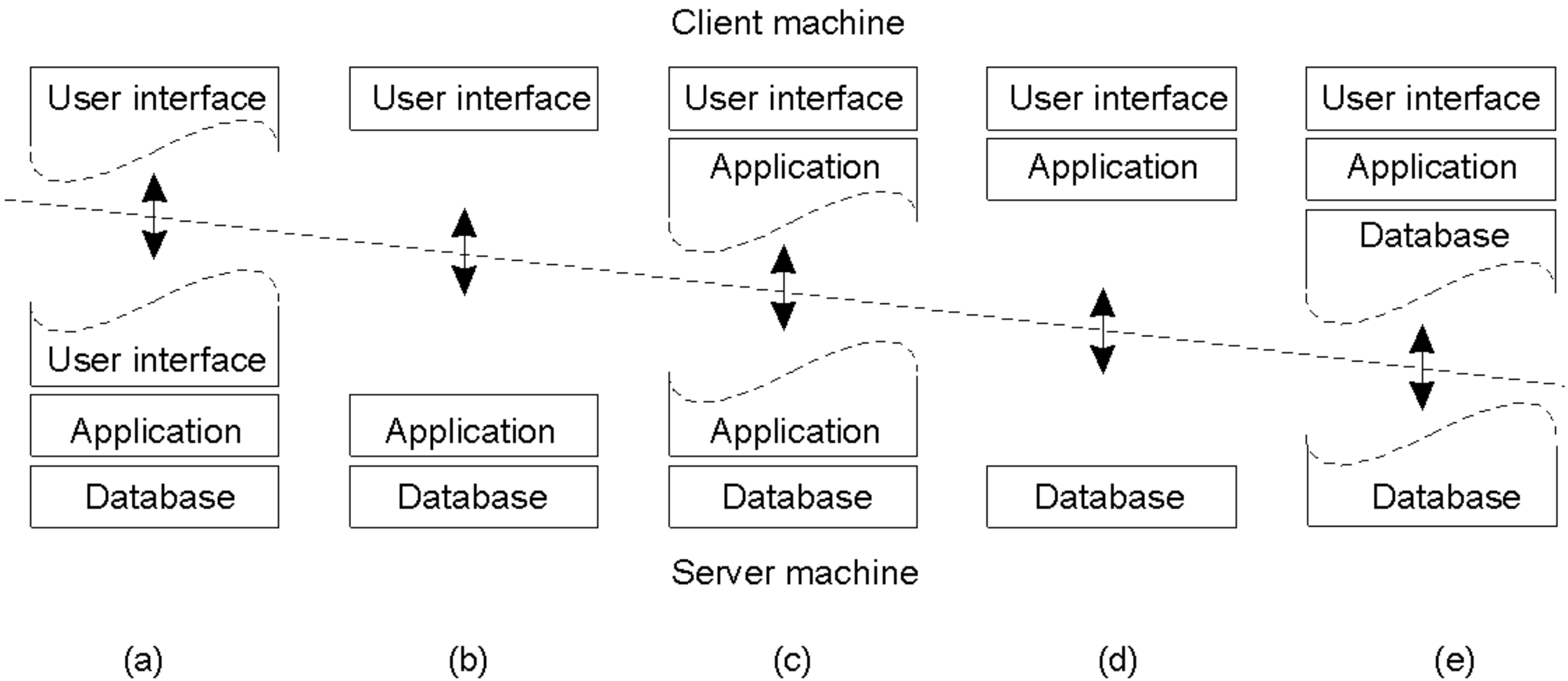
Multitier Systems (2)

- Example: Internet Search Engines

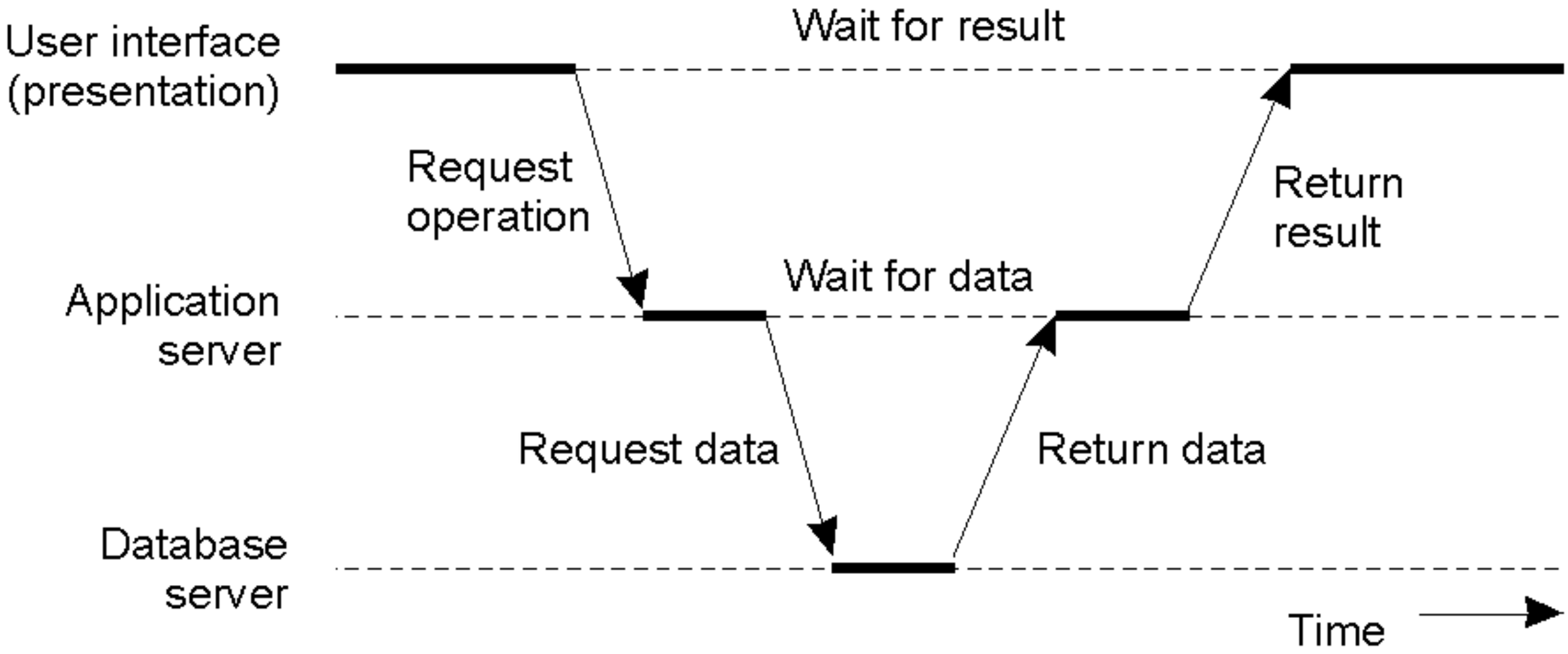


From Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e

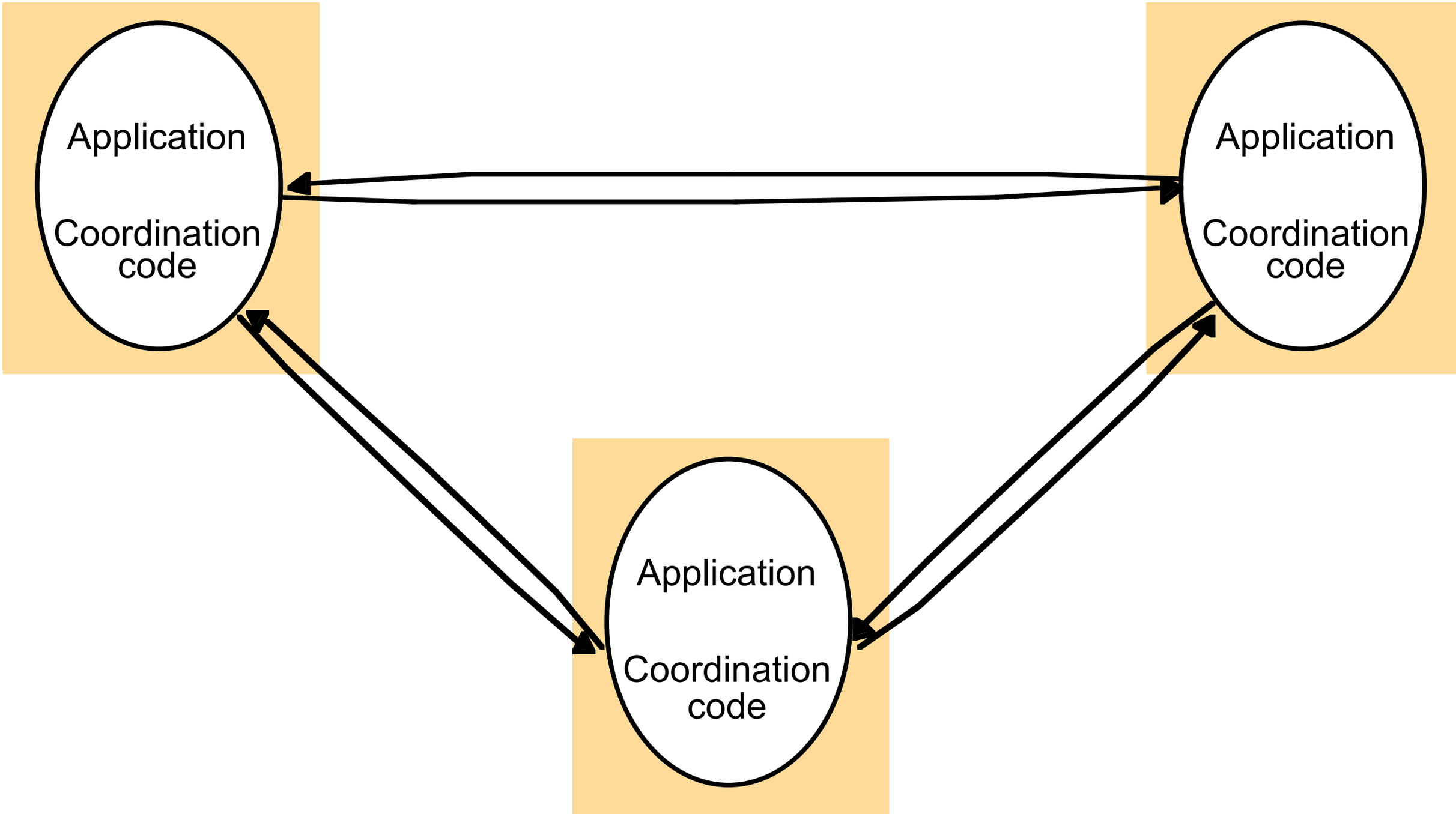
Multitier System Alternatives



Communication in Multitier Systems



Peer-to-Peer Systems



Motivations

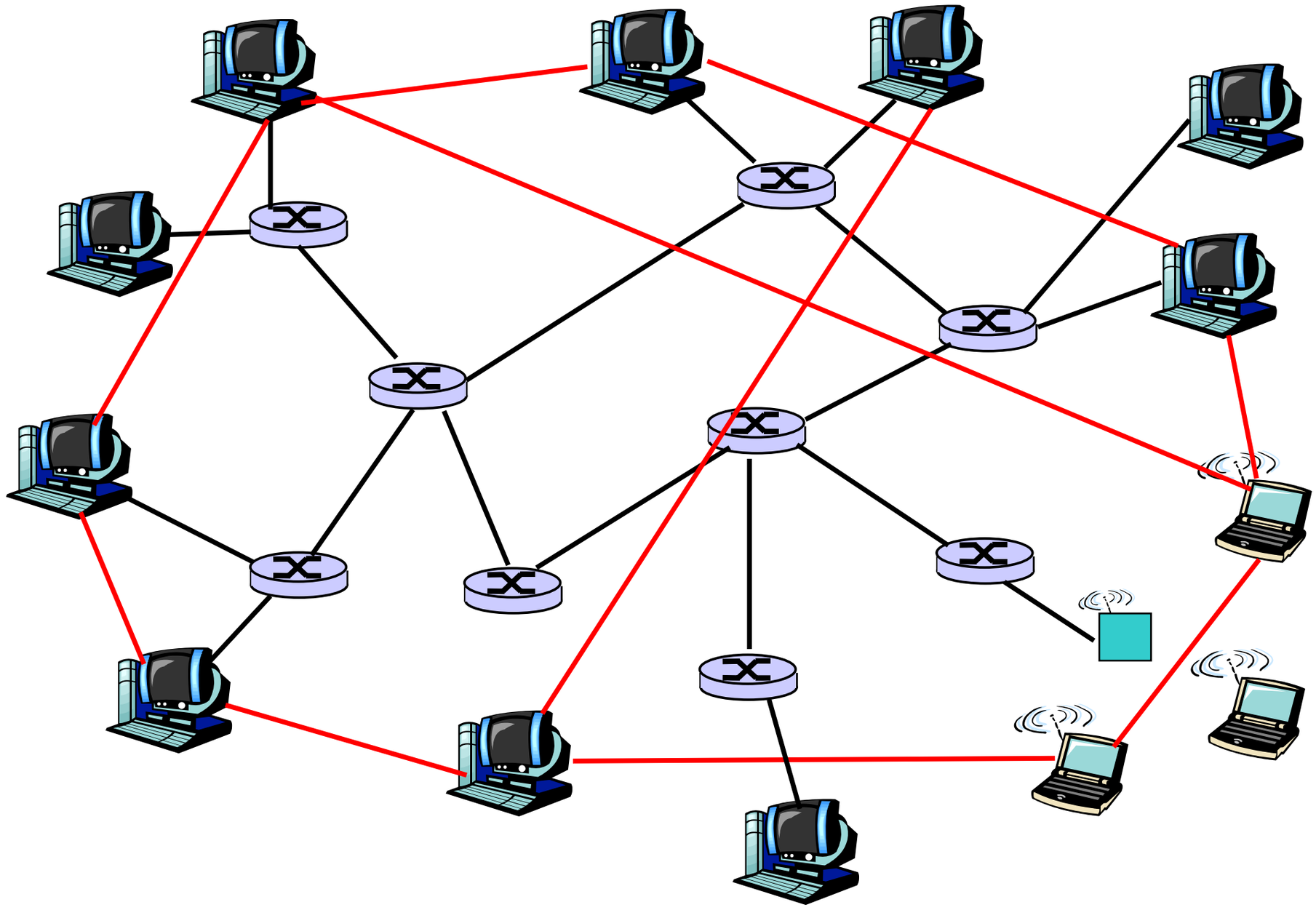
- Issues with client/server systems
  - ➡ Scalability issues
  - ➡ Single point of failure
  - ➡ Centralized administration
  - ➡ Unused resources at the edge
- P2P systems
  - ➡ Each peer can have same functionality (symmetric nodes)
  - ➡ Peers can be autonomous and unreliable
  - ➡ Very dynamic: peers can join and leave the network at any time
  - ➡ Decentralized control, large scale
  - ➡ Low-level, simple services
  - ➡ File sharing, computation sharing, computation sharing

Potential Benefits of P2P Systems

- Scale up to very large numbers of peers
- Dynamic self-organization
- Load balancing
- Parallel processing
- High availability through massive replication

Overlay Networks

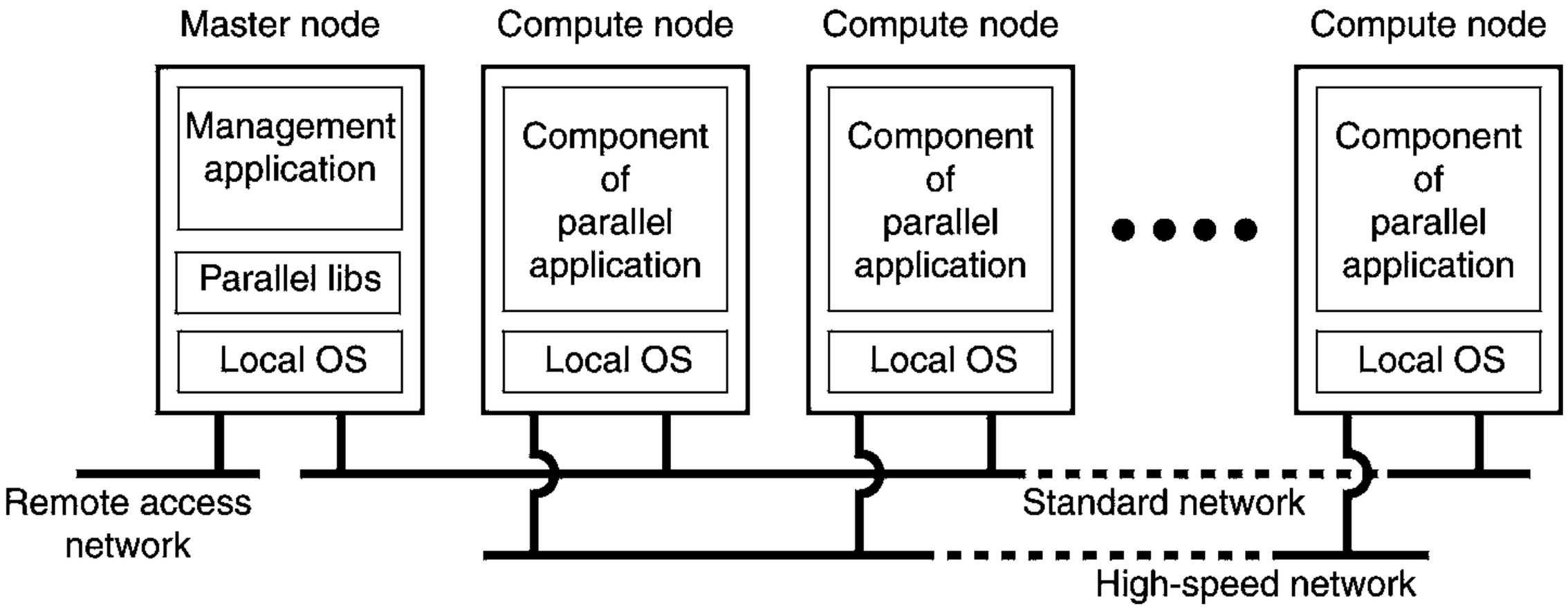
overlay edge



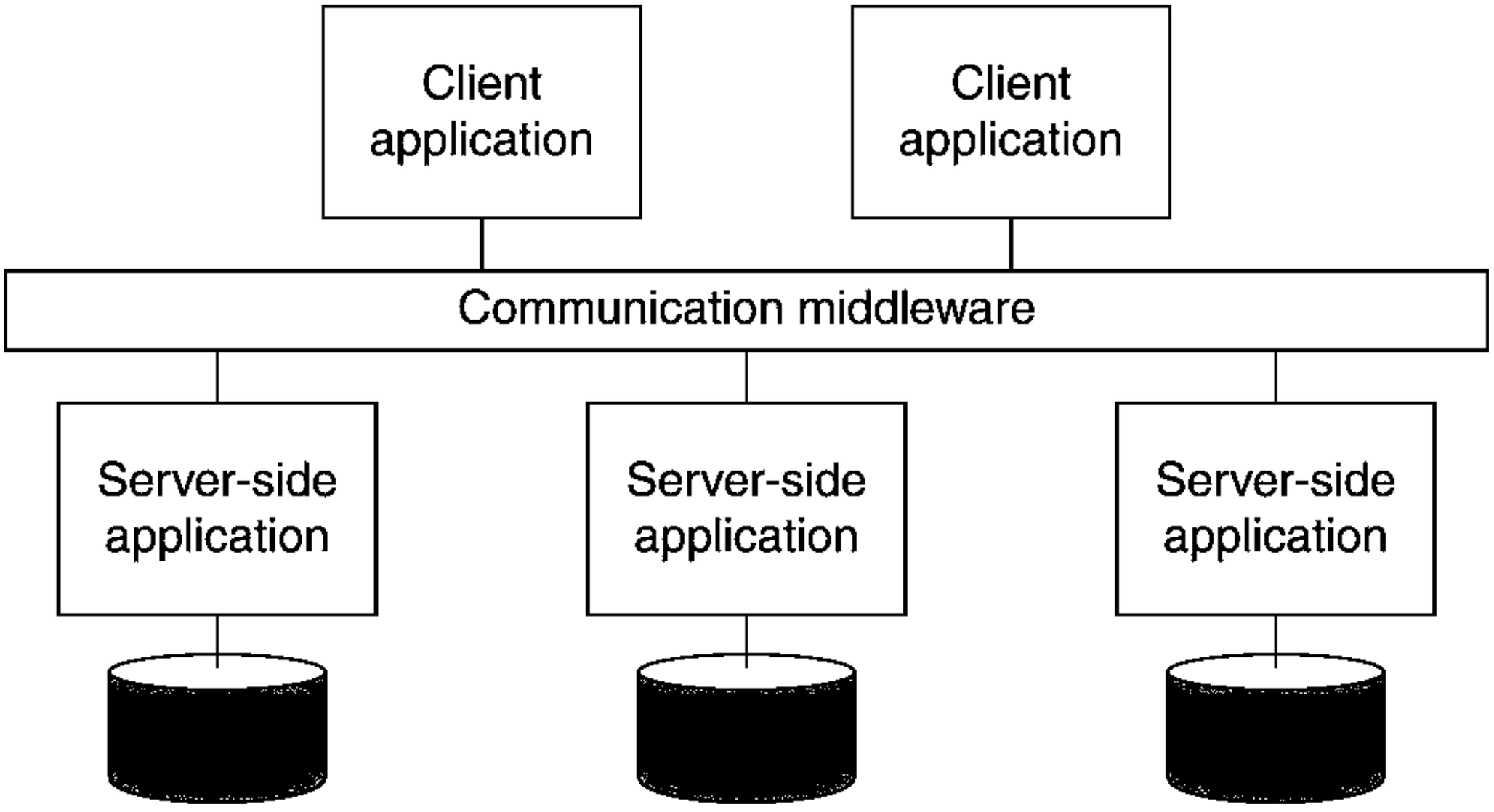
P2P Network Topologies

- Pure P2P systems
  - Unstructured systems
    - e.g. Napster, Gnutella, Freenet, Kazaa, BitTorrent
  - Structured systems (DHT)
    - e.g. LH* (the earliest form of DHT), CAN, CHORD, Tapestry, Freepastry, Pgrid, Baton
- Super-peer (hybrid) systems
  - e.g. Edutela, JXTA
- Two issues
  - Indexing data
  - Searching data

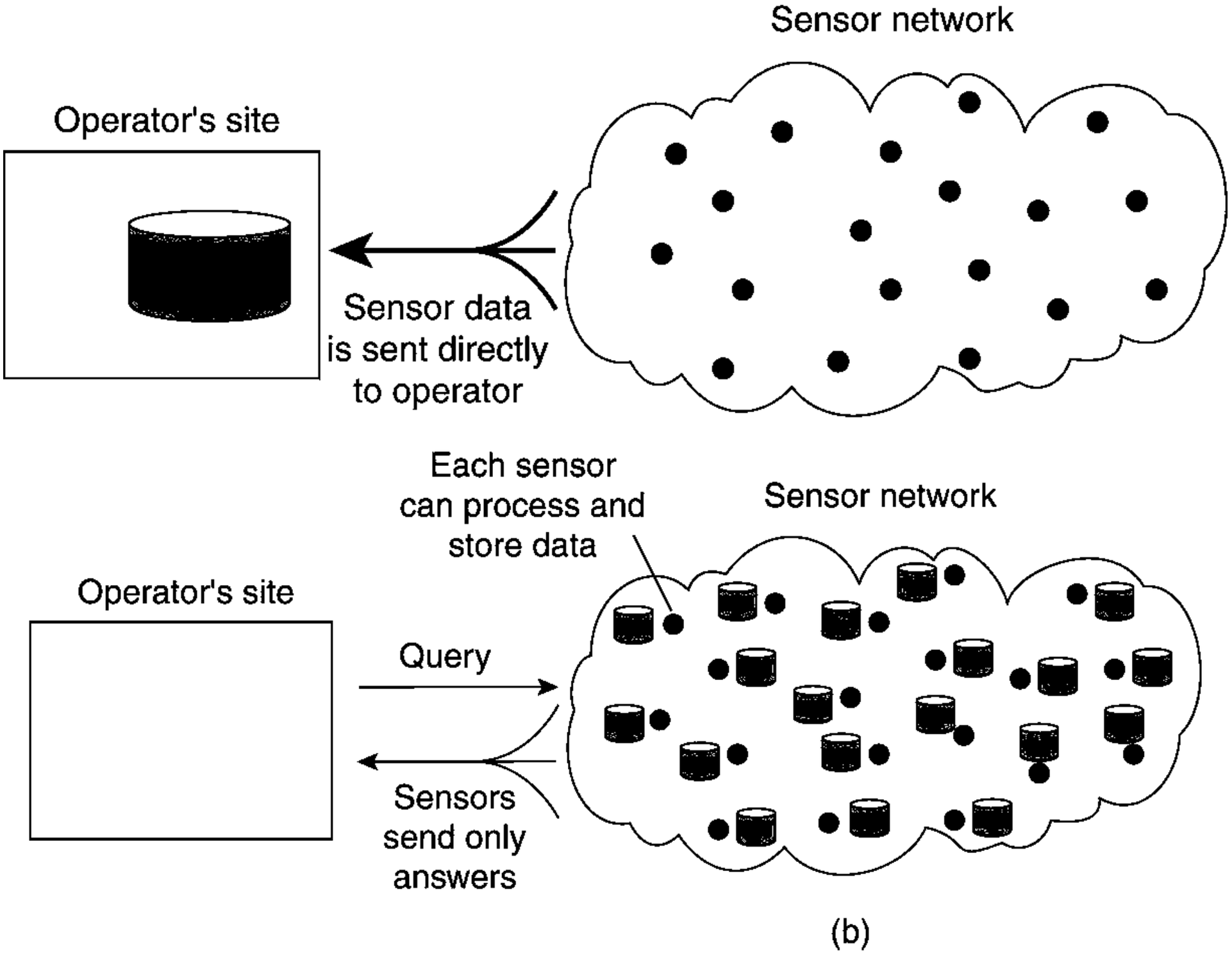
Cluster Computing Systems



Enterprise Application Integration

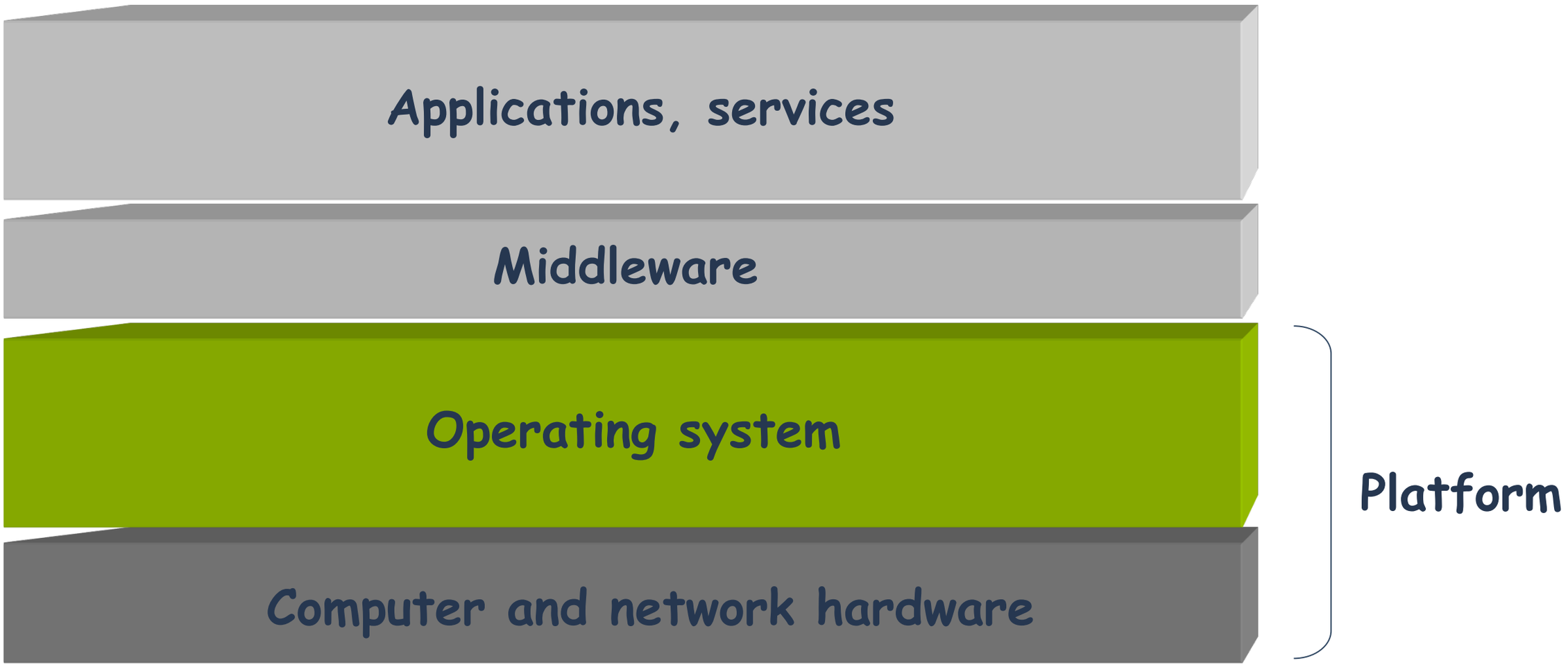


Sensor Networks

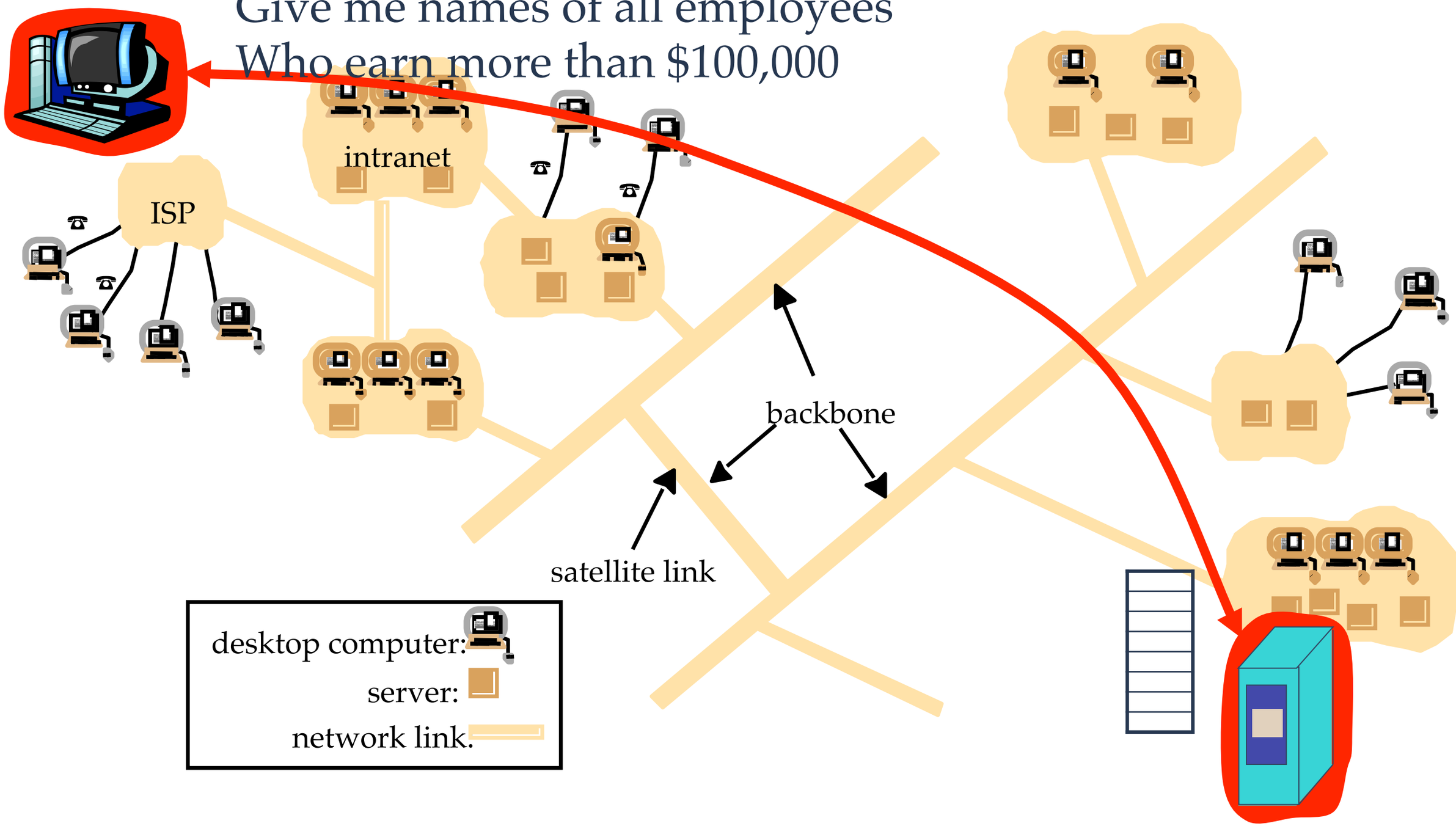


(b)

What is the Platform?



Networks and Communication



Issues

- How do the request and response get transmitted between the requestor and the server?
  - ➡ Protocols to facilitate communication
  - ➡ Moving the request and reply messages through the network
- What are the modes of communication between the requestor and the responder?
  - ➡ Connectionless service
  - ➡ Connection-oriented service

What's the Internet: "nuts and bolts" view

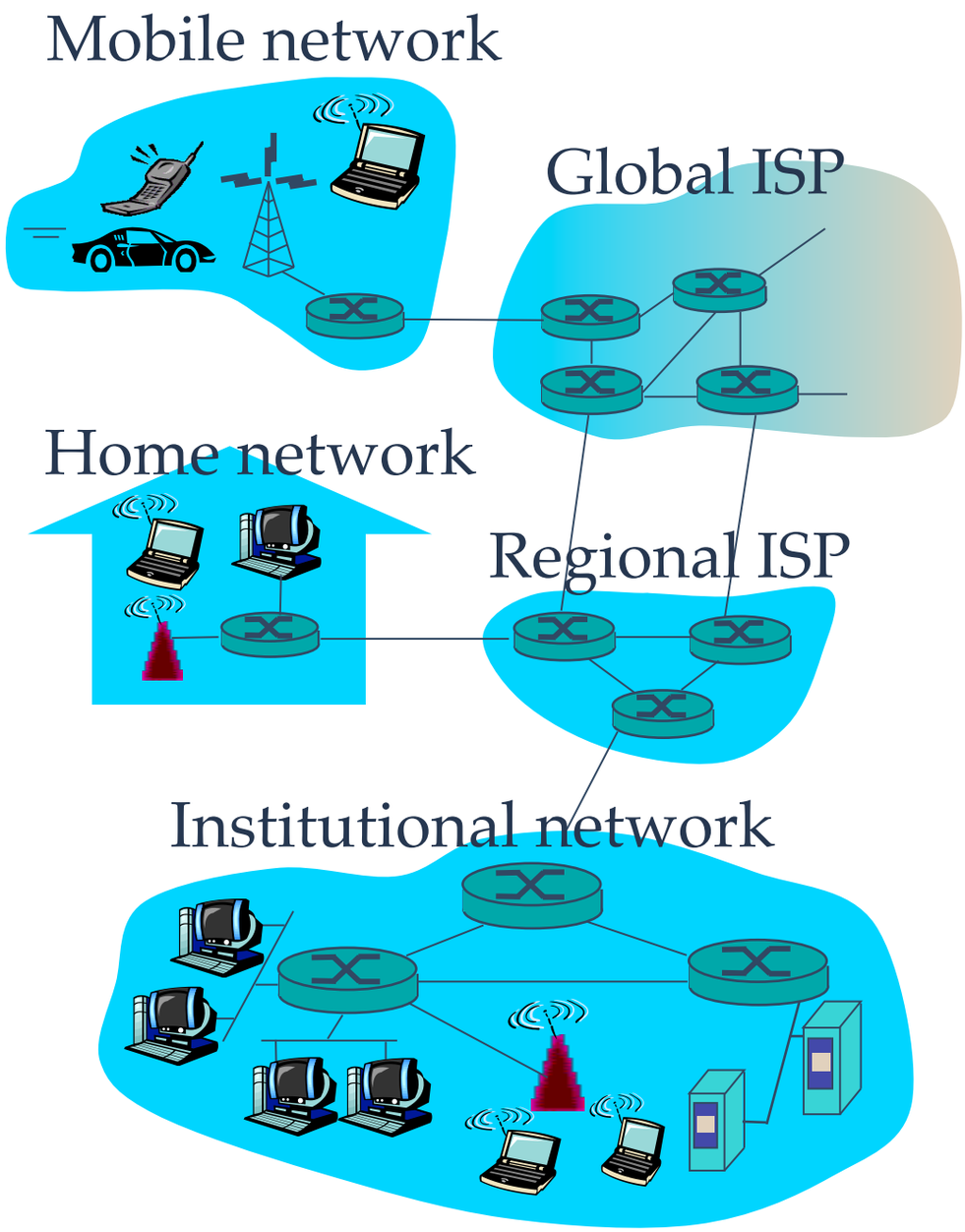
PC

server

wireless laptop

cellular handheld

- millions of connected computing devices: *hosts = end systems*
  - ➡ running *network apps*



access points

wired links

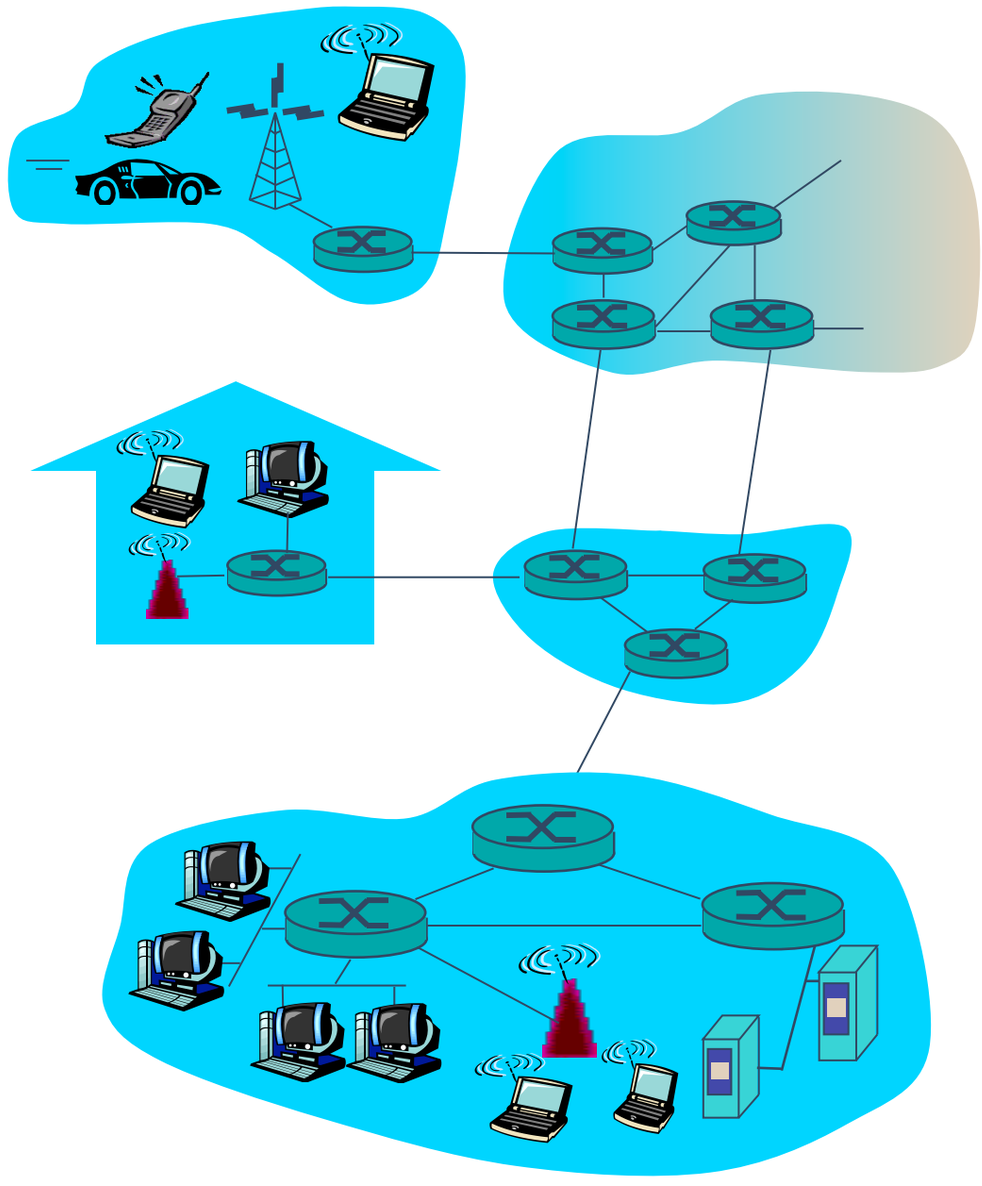
- *communication links*
  - ➡ fiber, copper, radio, satellite
  - ➡ transmission rate = *bandwidth*

router

- *routers:* forward packets (chunks of data)

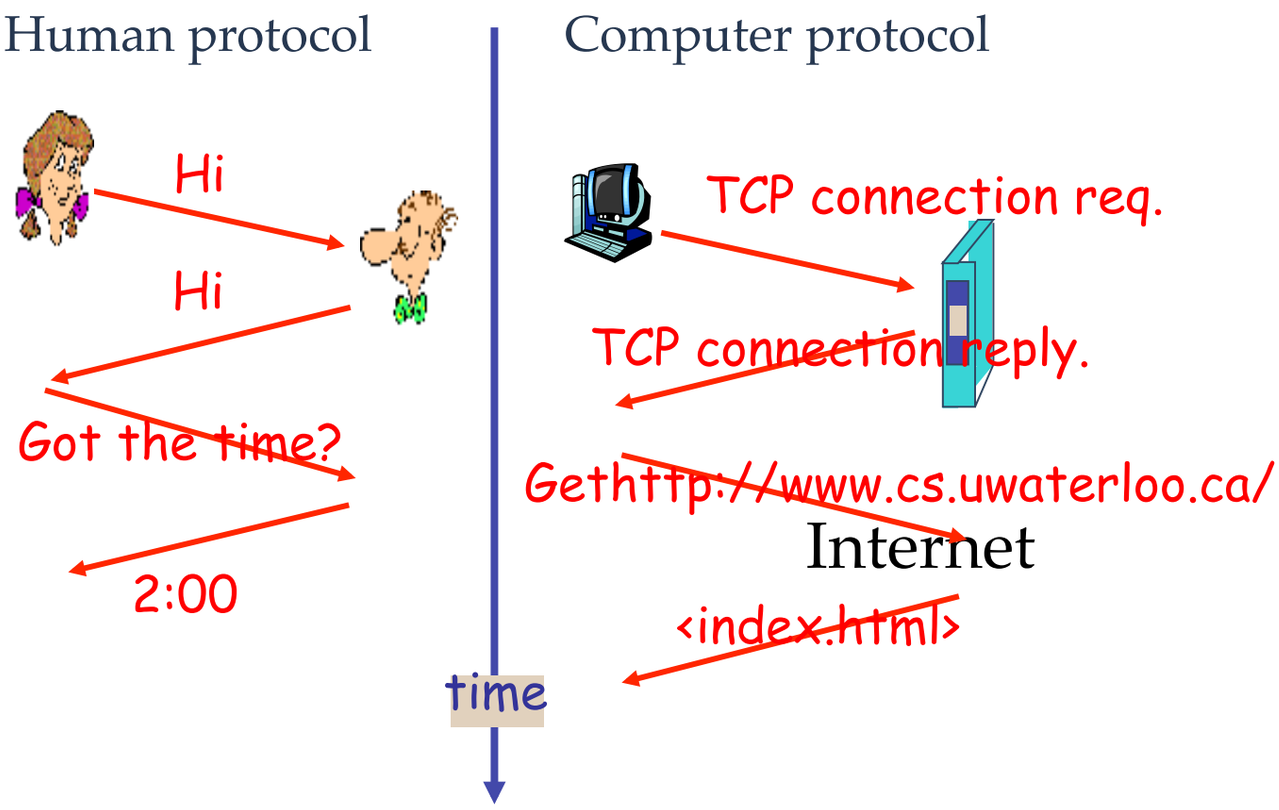
What's the Internet: a service view

- communication *infrastructure* enables distributed applications:
  - Web, VoIP, email, games, e-commerce, file sharing
- communication services provided to apps:
  - reliable data delivery from source to destination
  - "best effort" (unreliable) data delivery



What is a protocol?

- A protocol defines the format and the order of messages sent and received among network entities, and the actions taken on message transmission and receipt
- Human protocols:
  - "What's the time?"
  - "I have a question"
  - introductions
- Network protocols:
  - machines rather than humans
  - all communication activity in governed by protocols



Protocol "Layers"

Networks are complex, with many "pieces":

- hosts
- routers
- links of various media
- applications
- protocols
- hardware, software

Question:

Is there any hope of *organizing* structure of network?

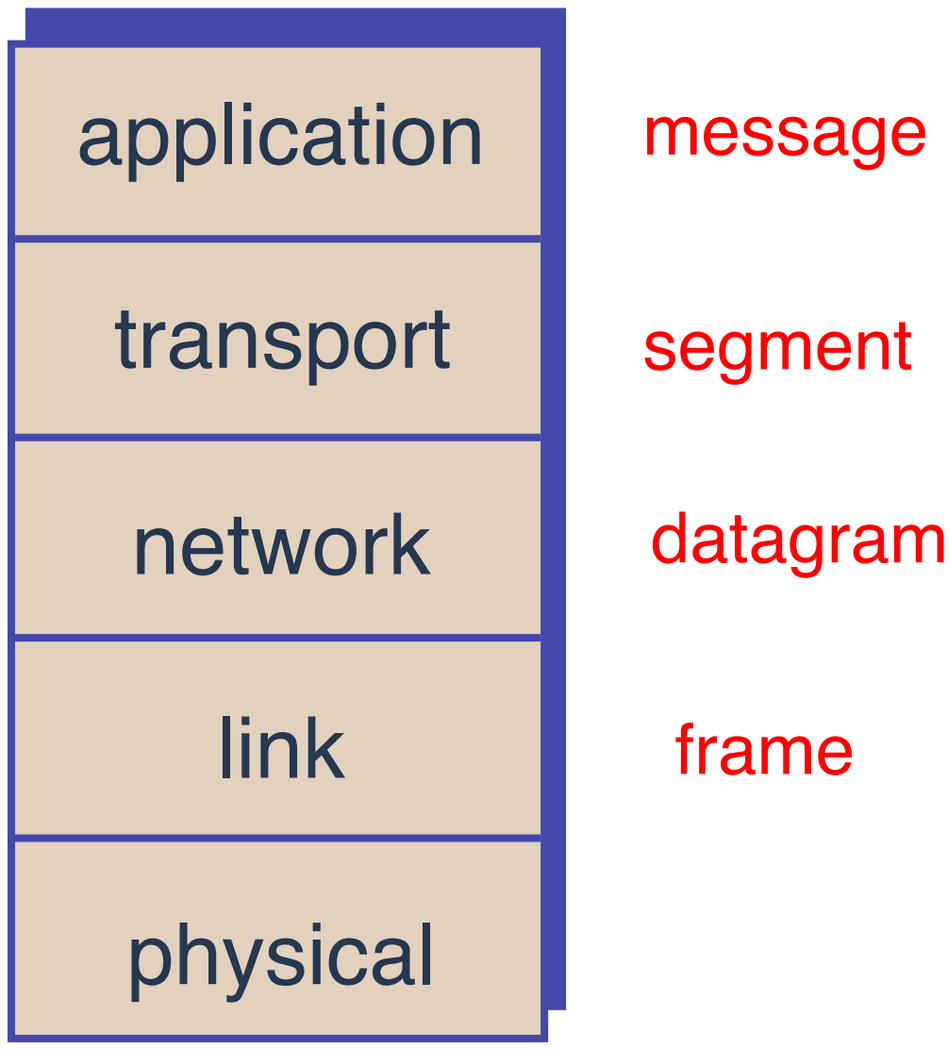
Or at least our discussion of networks?

Layered Architecture

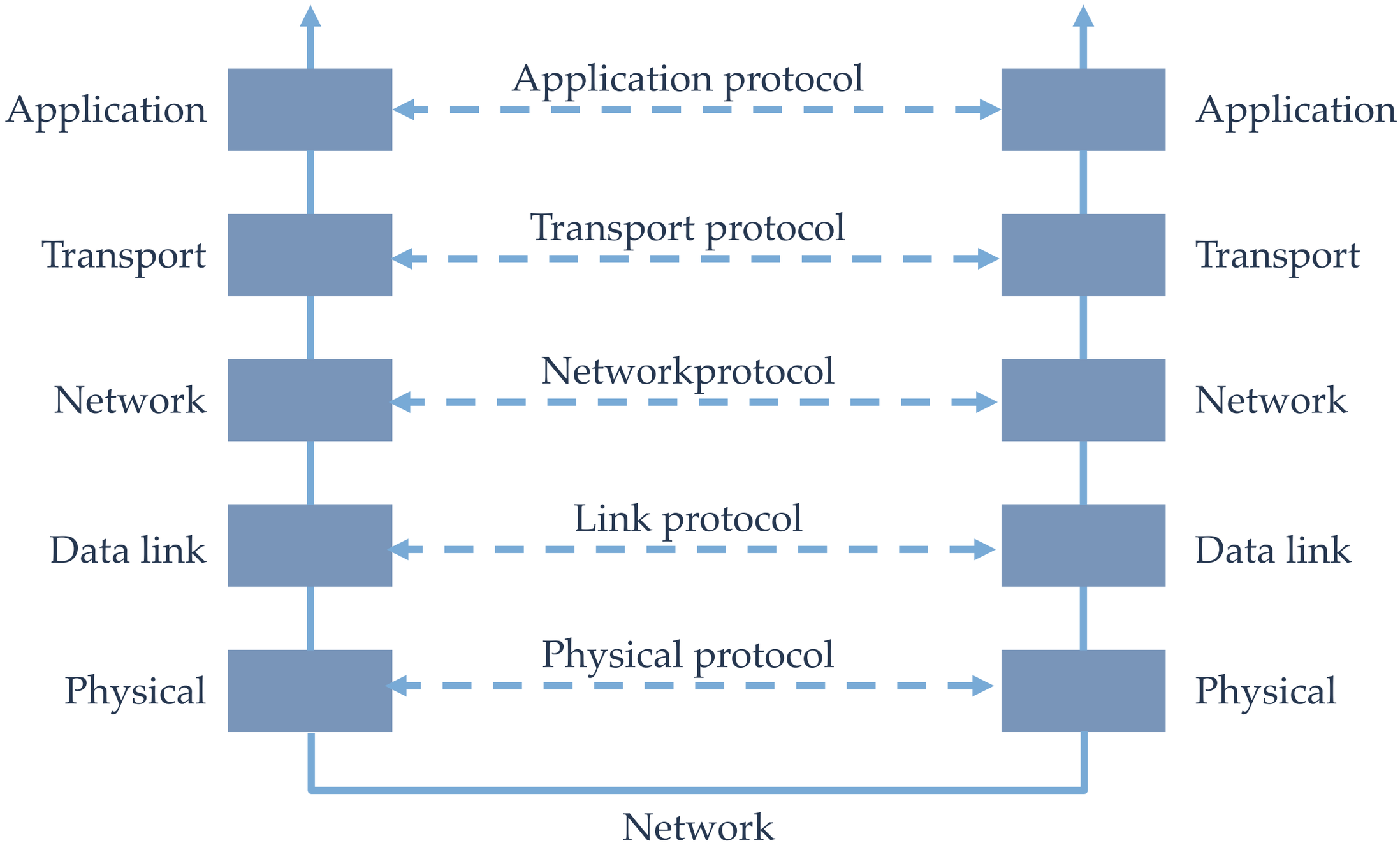
- Many requirements result in a large and complex system
- Introduce modularity by providing required functionality in layers
- Each layer provides a well defined set of services to next upper layer and exploits only services provided by next lower layer
- Can change implementation of a layer without affecting rest of the system
  - ➡ E.g., need to replace only a single layer when moving from wireless to wired network access

Internet Protocol Stack

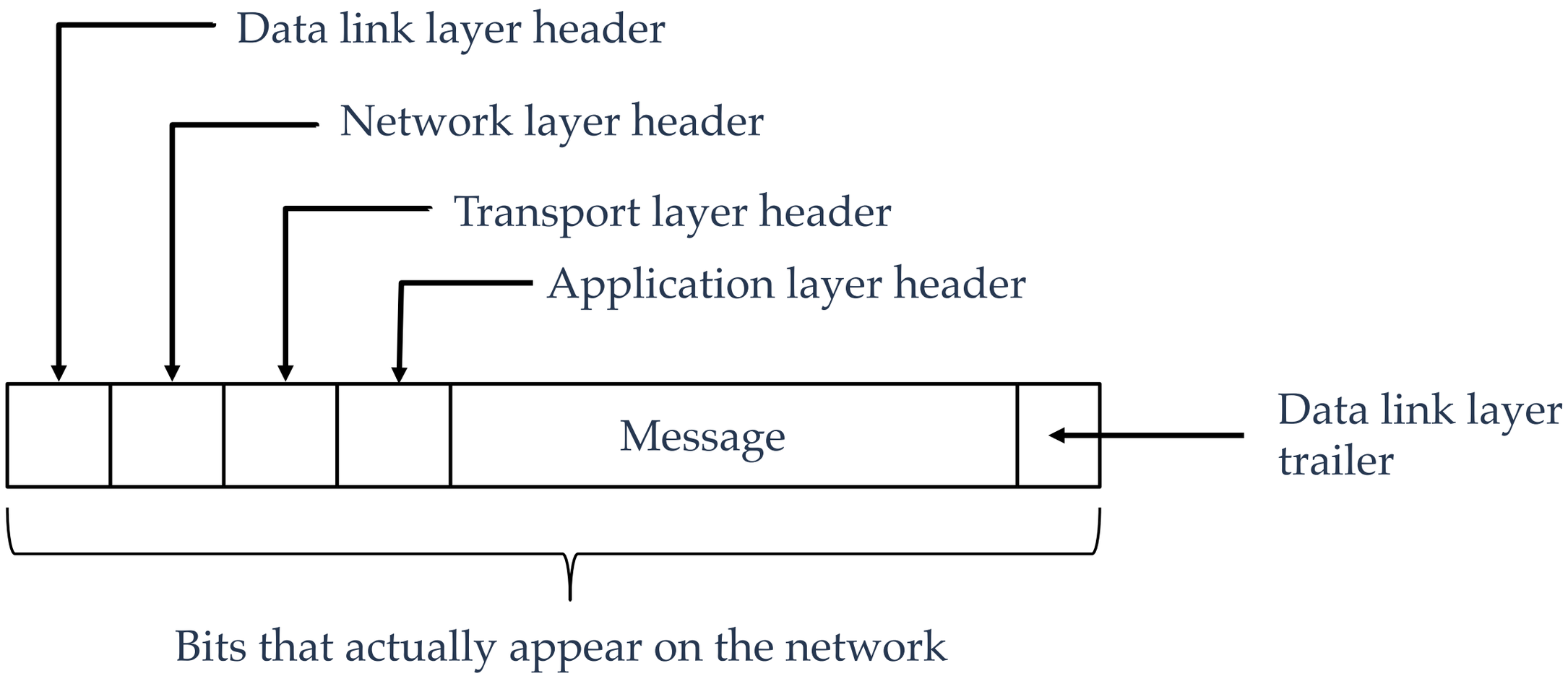
- application: supporting network applications
  - ➡ ftp, smtp, http
- transport: host-host data transfer
  - ➡ tcp, udp
- network: routing of datagrams from source to destination
  - ➡ ip, routing protocols
- link: data transfer between neighboring network elements
  - ➡ ppp, ethernet
- physical: bits “on the wire”



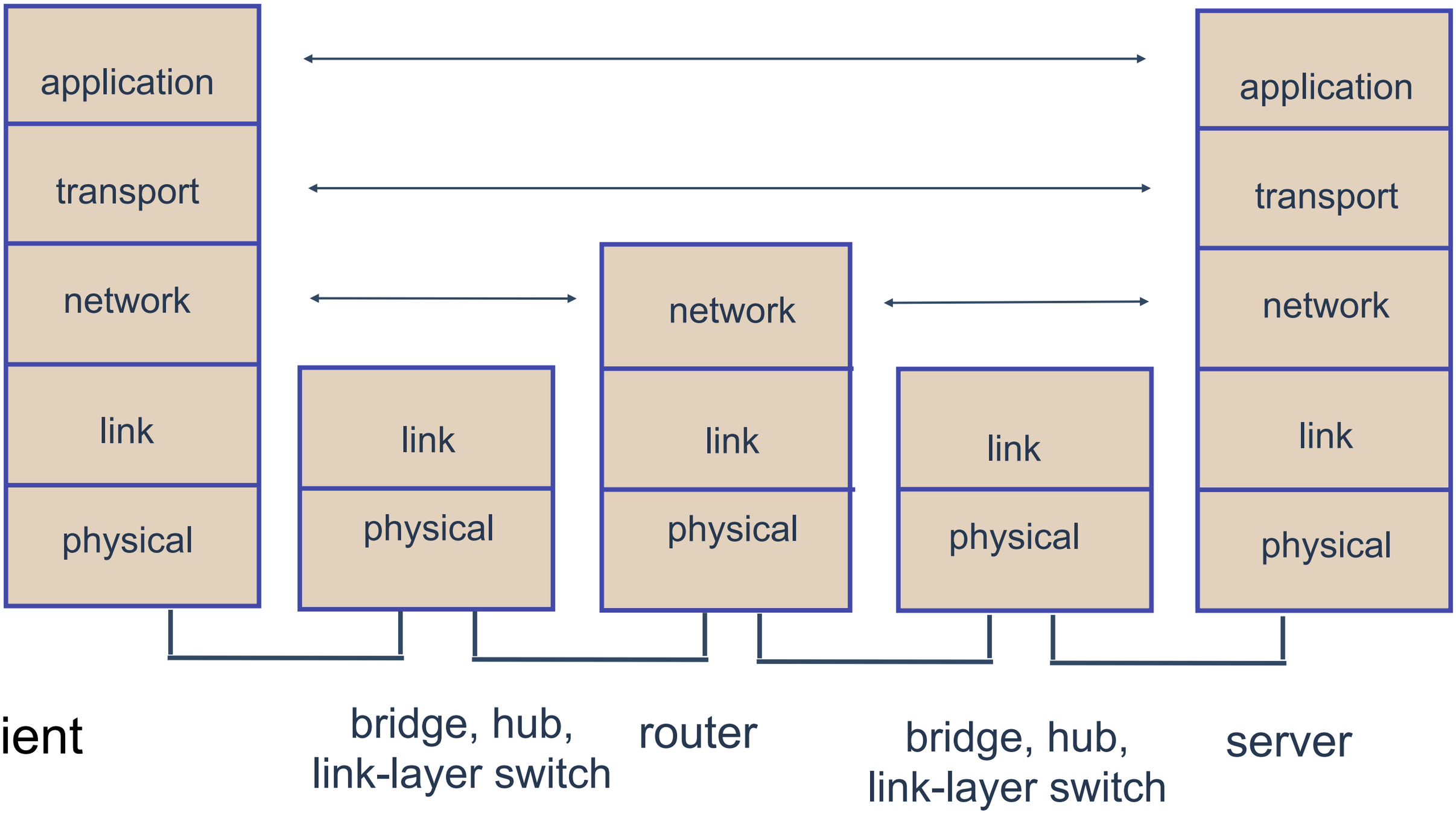
Layered Architecture



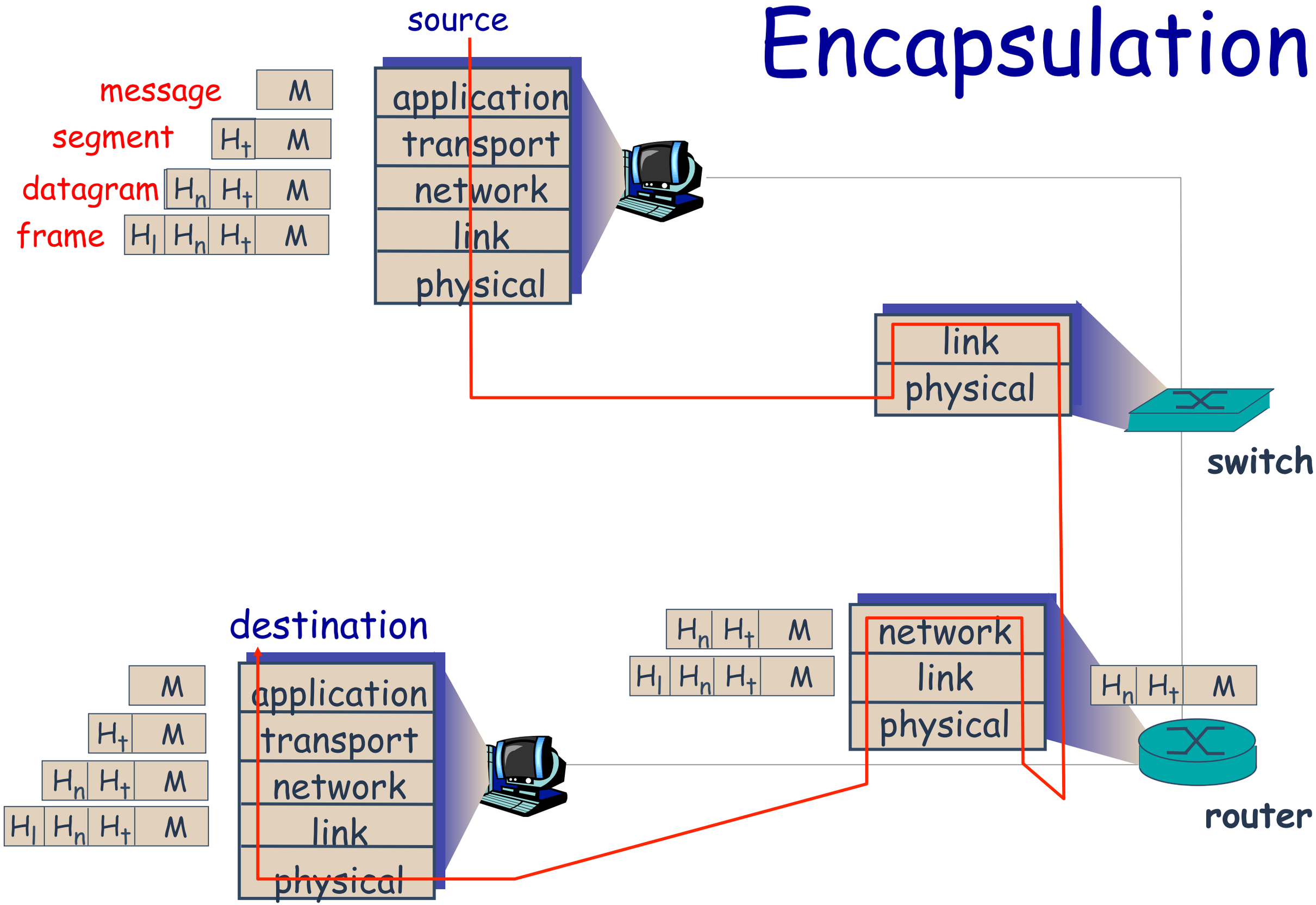
Message Format



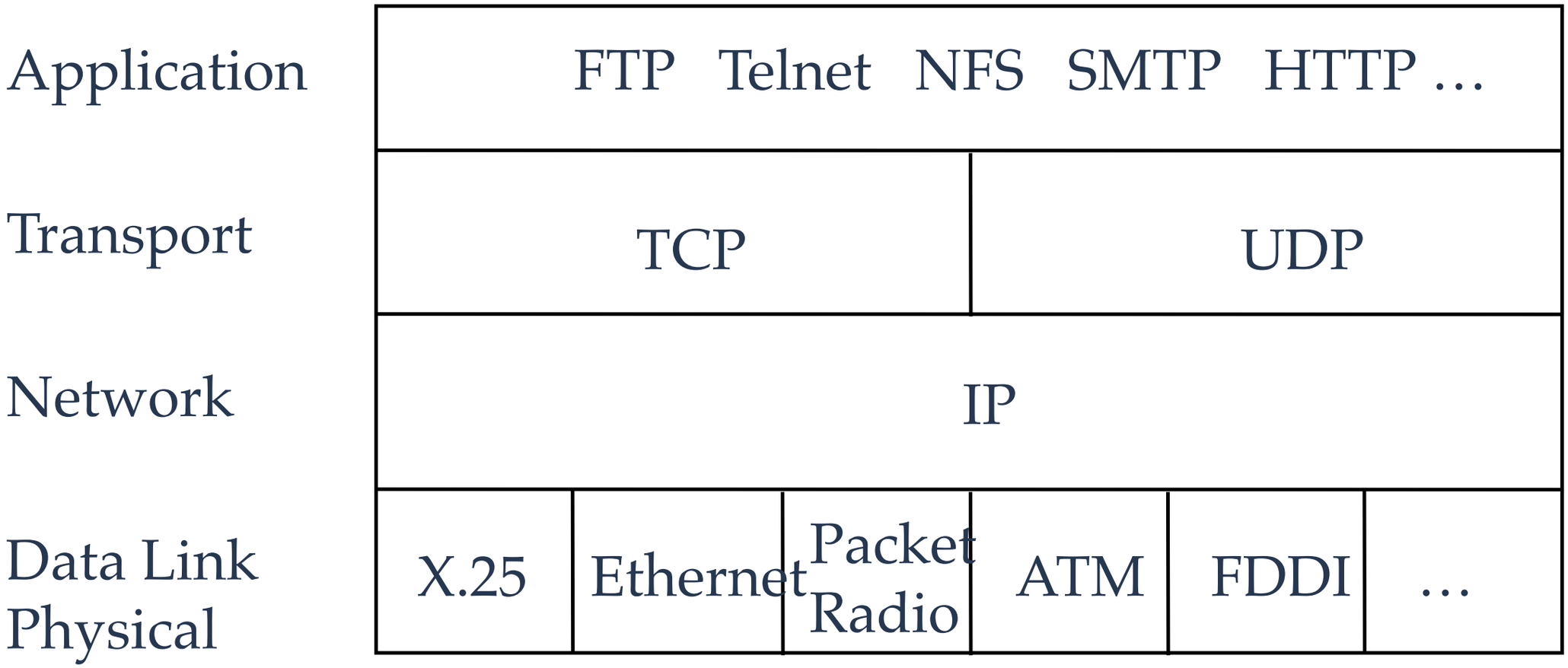
Only Endnodes Implement all Layers



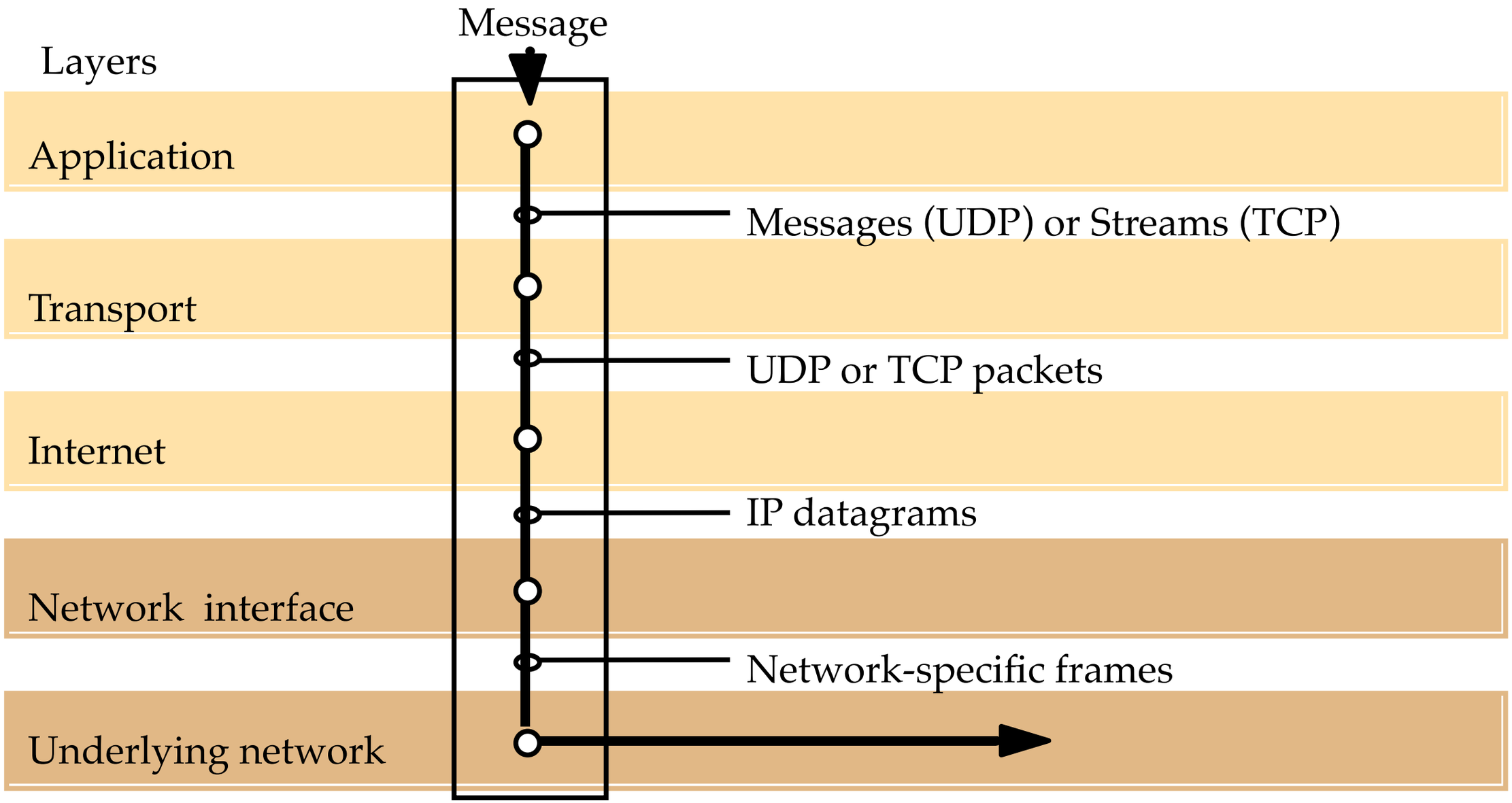
Encapsulation



Internet Protocols

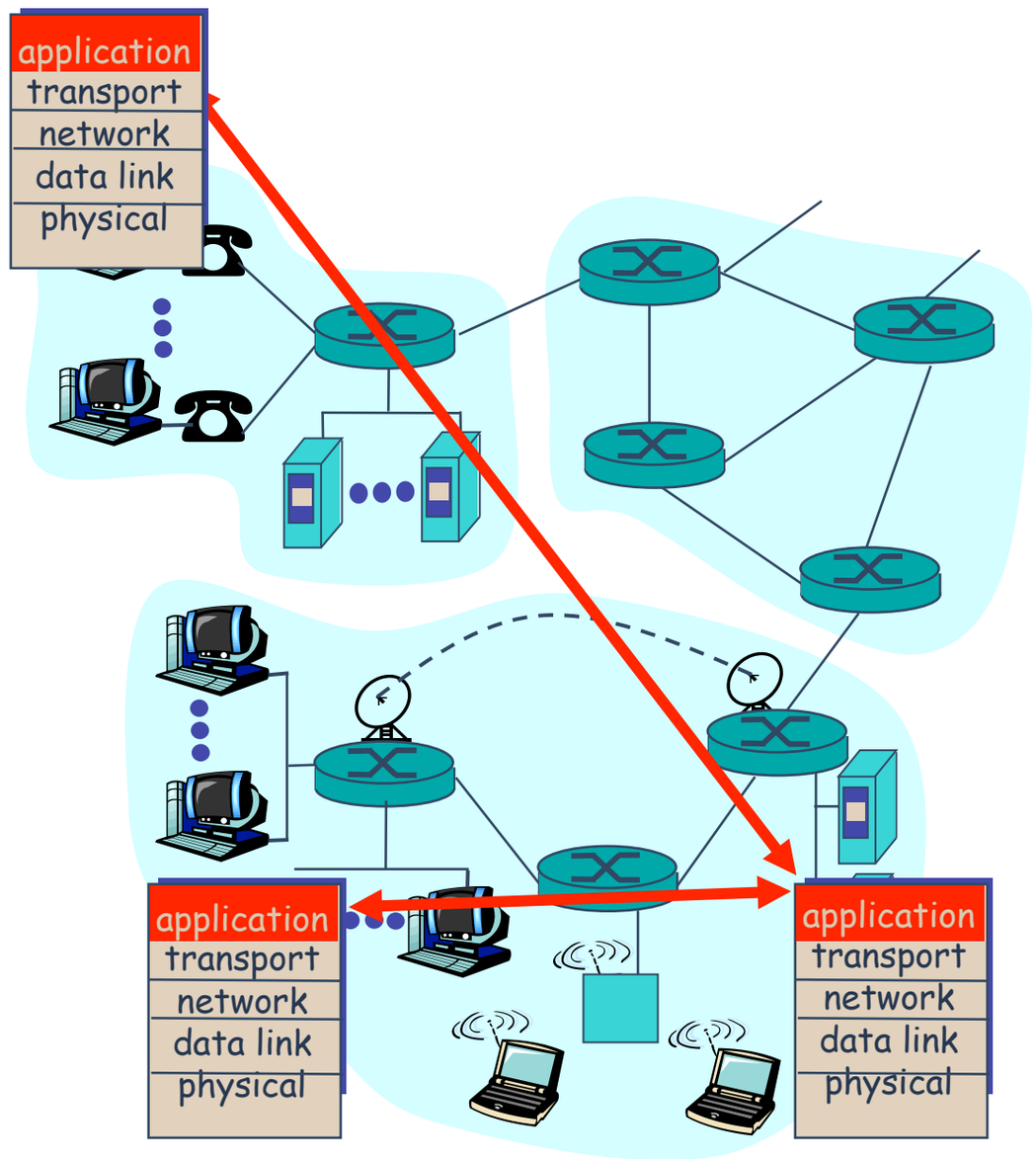


TCP/IP Layers



Applications and Application-Layer Protocols

- Application: communicating, distributed processes
  - ➡ running in network hosts in "user space"
  - ➡ exchange messages to implement app
  - ➡ e.g., email, file transfer, the Web
- Application-layer protocols
  - ➡ one "piece" of an app
  - ➡ define messages exchanged by apps and actions taken
  - ➡ use services provided by lower layer protocols



Application-Layer Protocols (2)

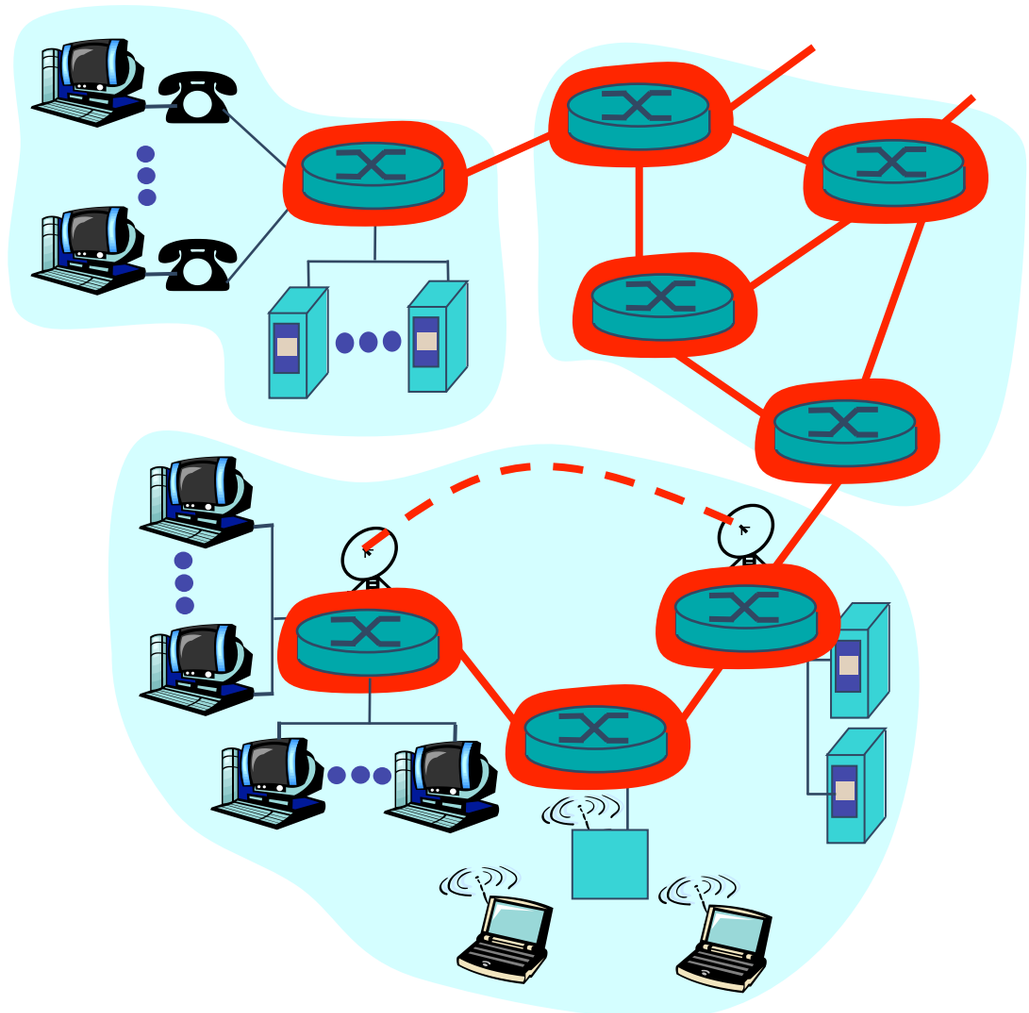
- API: application programming interface
  - Defines interface between application and transport layer
  - socket: Internet API
    - two processes communicate by sending data into socket, reading data out of socket
- What transport services does an application need?
  - Data loss
    - some apps (e.g., audio) can tolerate some loss
    - other apps (e.g., file transfer) require 100% reliable data transfer
  - Bandwidth
    - some apps (e.g., multimedia) require a minimum amount of bandwidth to be "effective"
    - other apps ("elastic apps") make use of whatever bandwidth they get
  - Timing
    - some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

Network Core – How Data Move Through the Network

- Network is a mesh of interconnected routers
- Two ways of setting up a connection between two computers

➡ circuit switching: dedicated circuit per call – e.g., telephone net

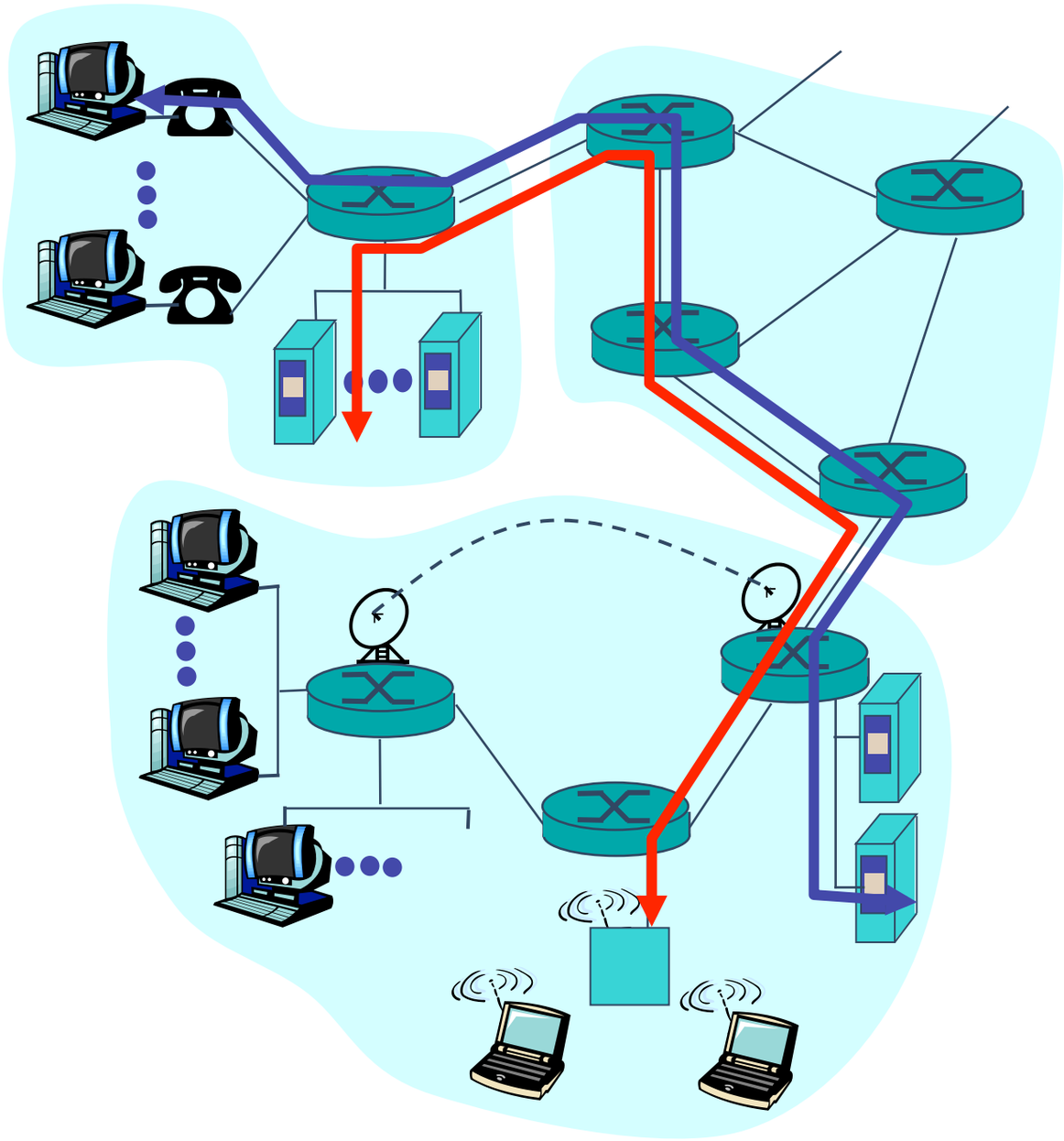
➡ packet-switching: data sent thru the network in discrete "chunks"



Circuit Switching

End-end resources reserved for "call"

- link bandwidth, switch capacity
- dedicated resources: no sharing
- circuit-like (guaranteed) performance
- call setup required

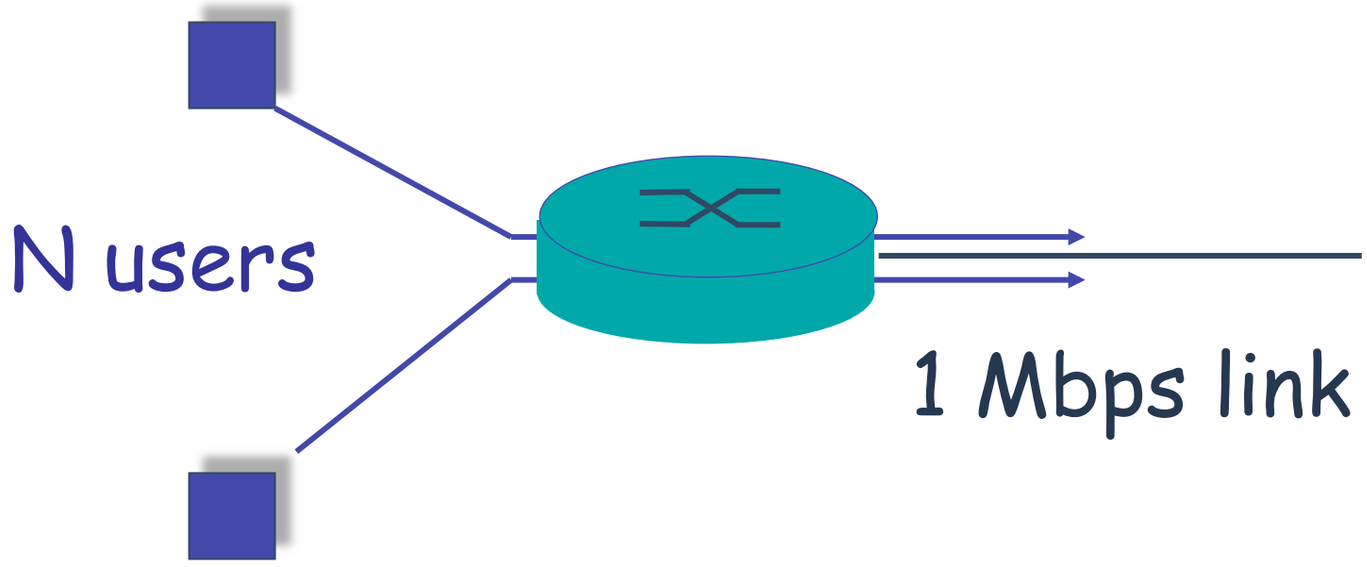


Packet Switching

- Each end-end data stream divided into packets
  - User A, B packets share network resources
  - Each packet uses full link bandwidth
  - Resources used as needed
- Resource contention:
  - Aggregate resource demand can exceed amount available
  - Congestion: packets queue, wait for link use
- Store and forward: packets move one hop-at-a-time (store-and-forward)
  - Transmit over link
  - Wait turn at next link

Packet Switching vs Circuit Switching

- Packet switching allows more users to use network!
- 1 Mbit link; each user:
  - 100Kbps when "active"
  - active 10% of time
- circuit-switching:
  - 10 users
- packet switching:
  - 35 users, probability > 10 active less than .0004



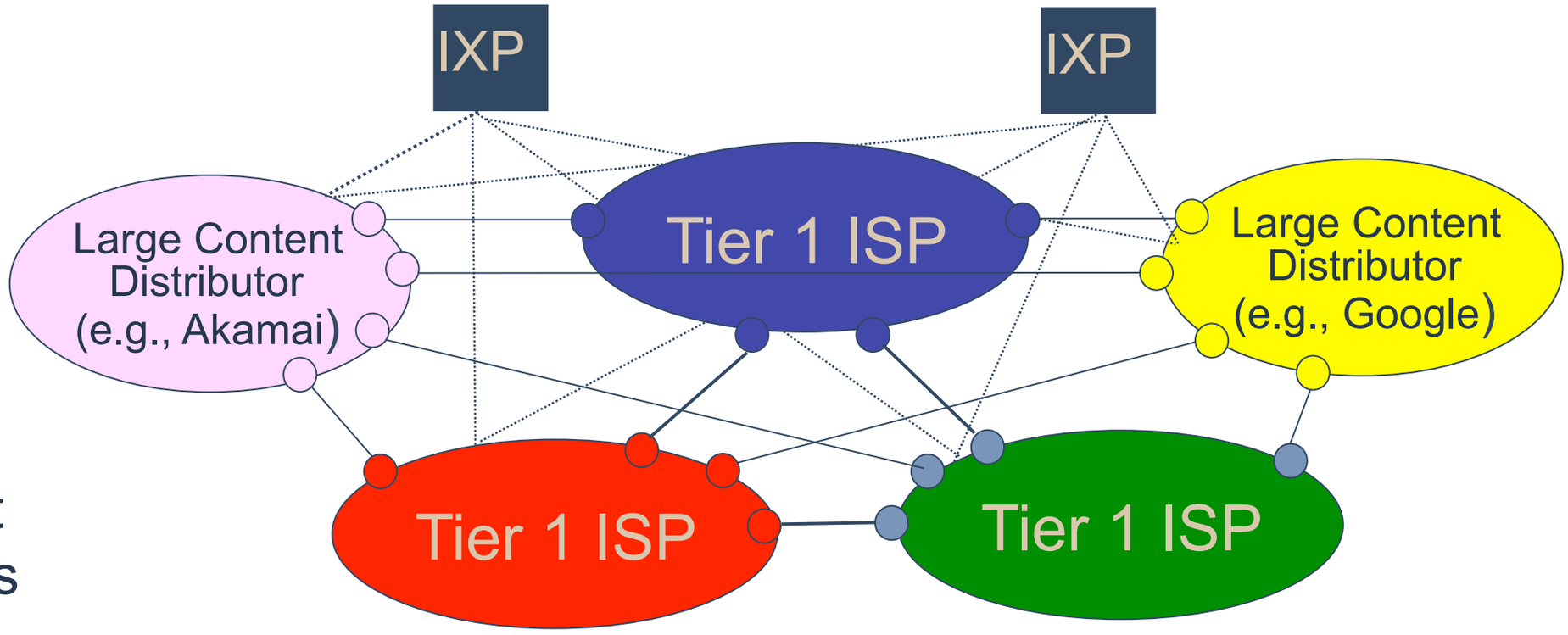
Packet Switching vs Circuit Switching (2)

- Packet switching is great for bursty data
  - ➡ resource sharing
  - ➡ no call setup
- It incurs excessive congestion: packet delay and loss
  - ➡ protocols needed for reliable data transfer, congestion control
- How to provide circuit-like behavior?
  - ➡ bandwidth guarantees needed for audio/video apps
  - ➡ still an unsolved problem, but solutions such as ATM have been developed

Internet structure: network of networks

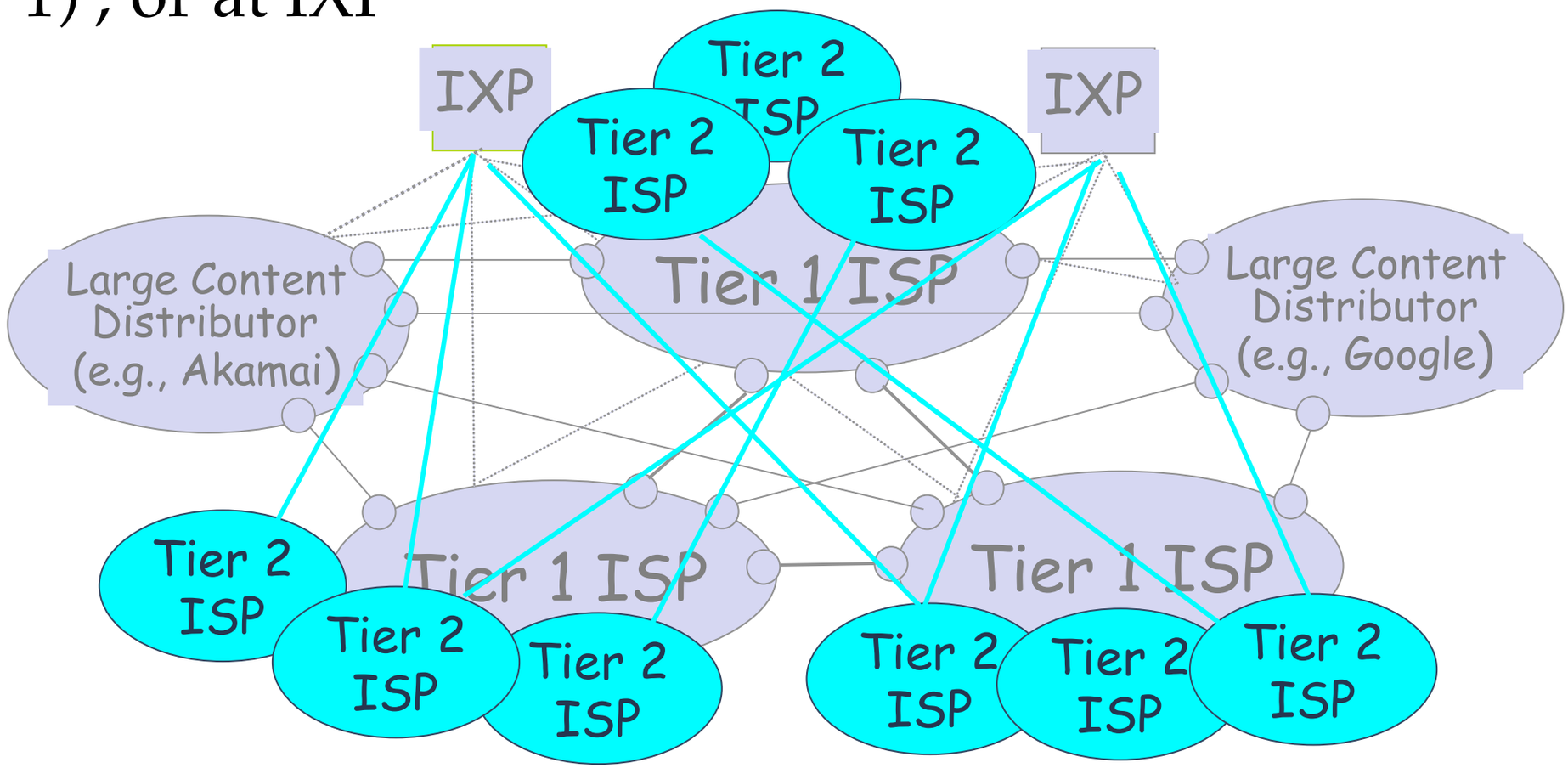
- Roughly hierarchical
- At center: small # of well-connected large networks
  - ➡ "tier-1" commercial ISPs (e.g., Rogers, Telus, Bell, Verizon, Sprint, AT&T, Qwest, Level3), national & international coverage
  - ➡ large content distributors (Google, Akamai, Microsoft)
  - ➡ treat each other as equals (no charges)

Tier-1 ISPs & Content Distributors, interconnect (peer) privately … or at Internet Exchange Points IXPs



Internet structure: network of networks

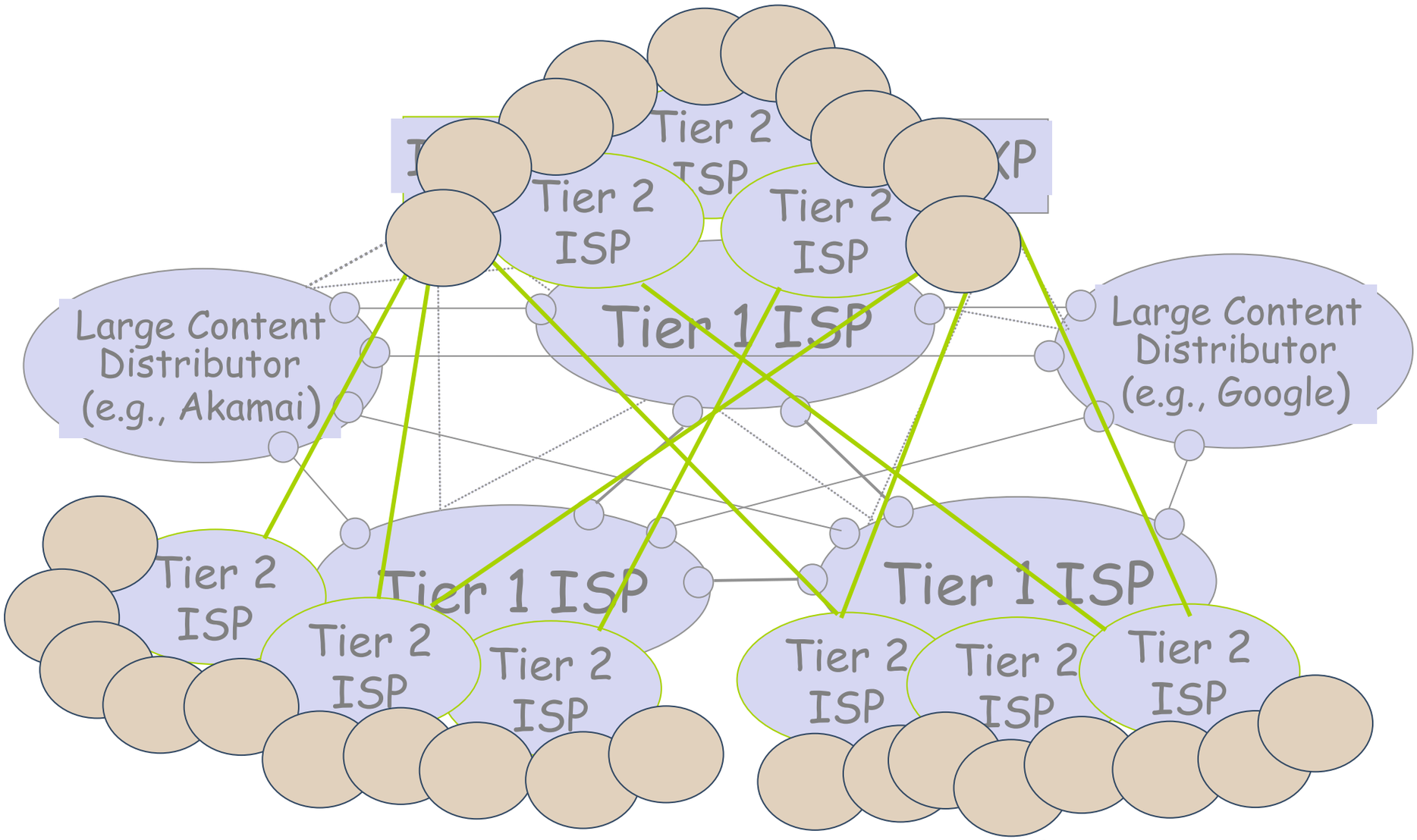
- "tier-2" ISPs: smaller (often regional) ISPs
  - ➡ connect to one or more tier-1 *(provider)* ISPs
    - ✦ each tier-1 has many tier-2 *customer nets*
    - ✦ tier 2 pays tier 1 provider
- tier-2 nets sometimes peer directly with each other (bypassing tier 1) , or at IXP



From Kurose & Ross, Computer Networking: A Top-Down Approach, 5e

Internet structure: network of networks

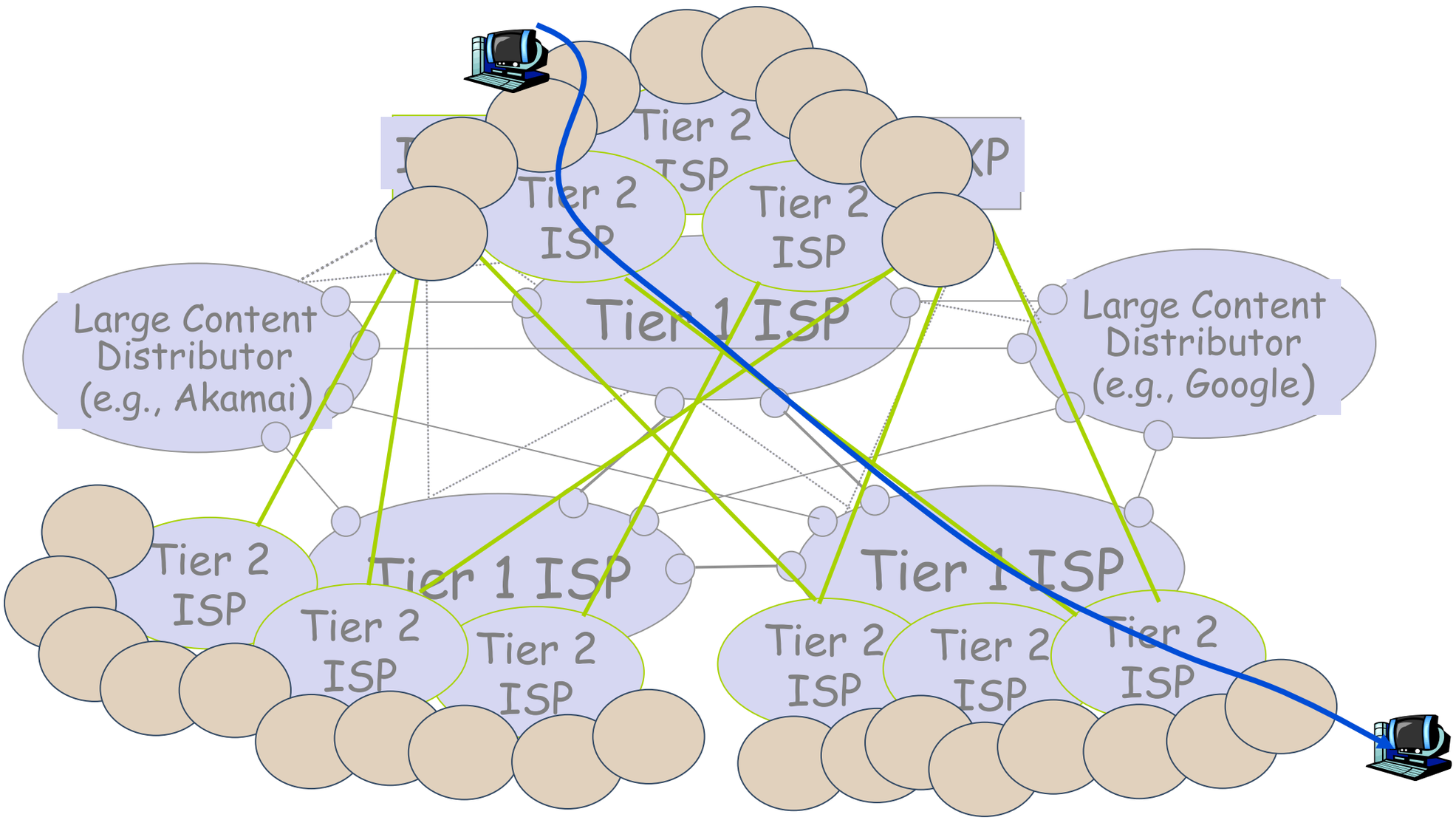
- "Tier-3" ISPs, local ISPs
- customer of tier 1 or tier 2 network
  - ➡ last hop ("access") network (closest to end systems)



From Kurose & Ross, Computer Networking: A Top-Down Approach, 5e

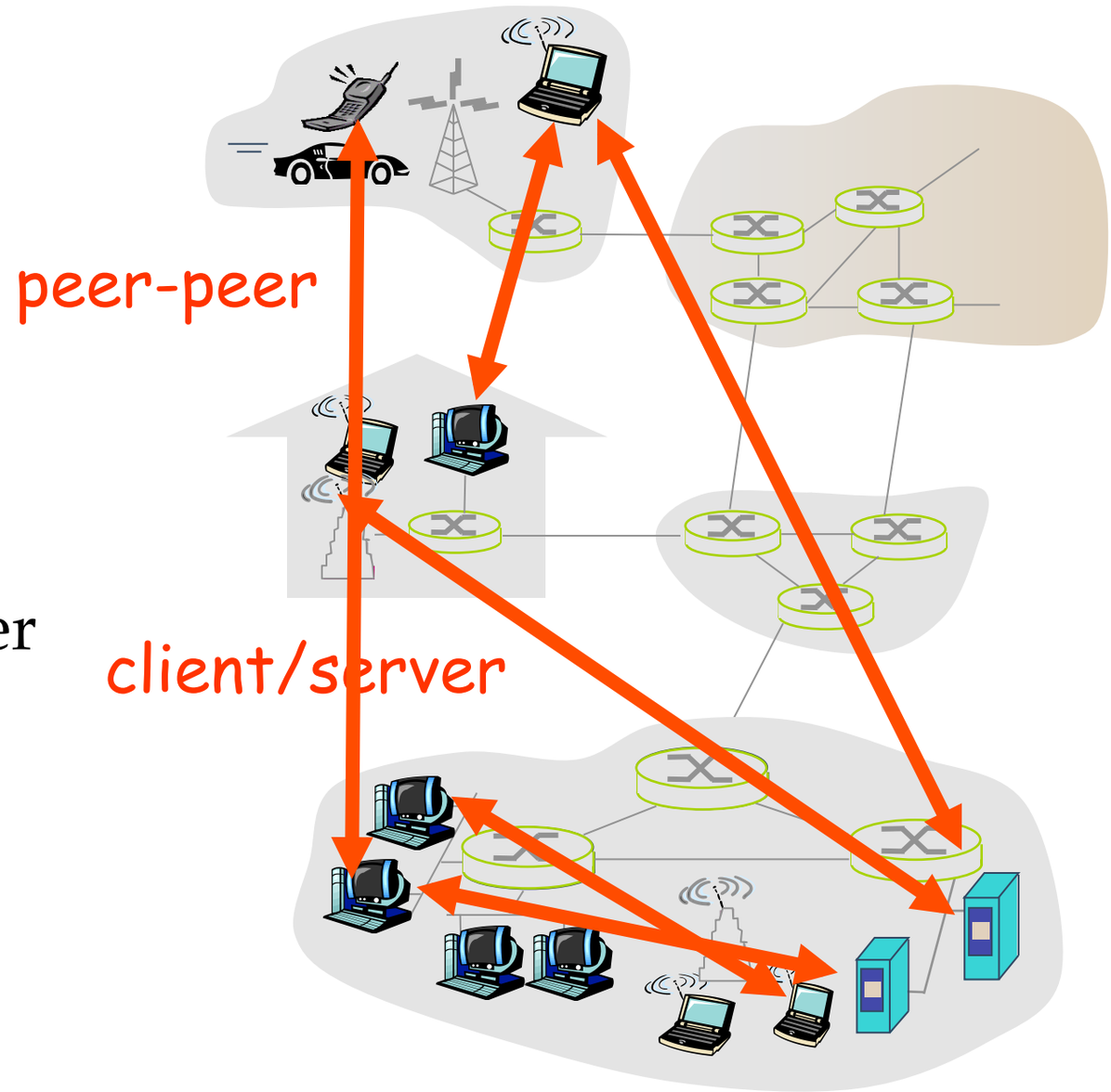
Internet structure: network of networks

- a packet passes through *many* networks from source host to destination host



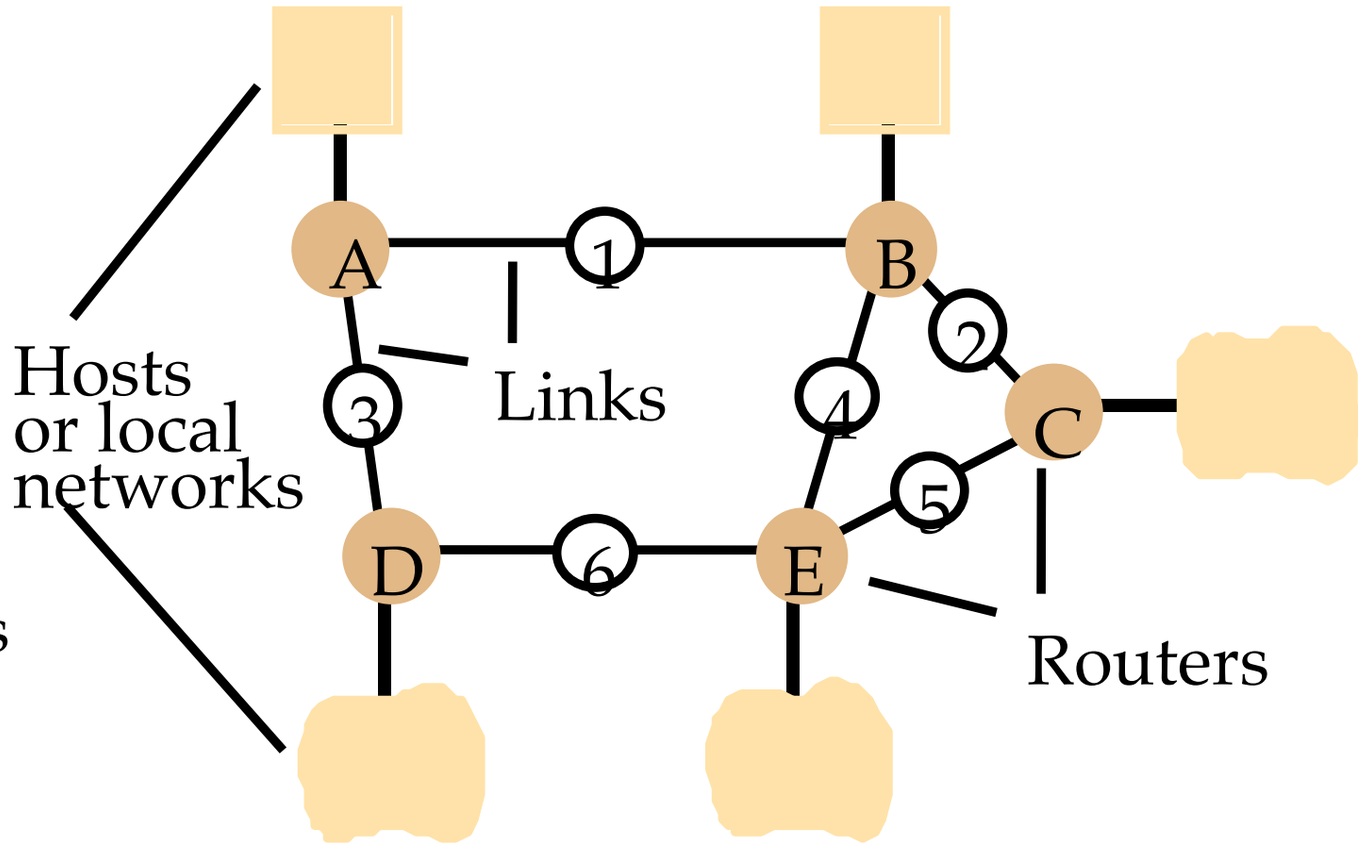
The Network Edge: How to Connect to the Network

- end systems (hosts):
  - run application programs
  - e.g. Web, email
  - at "edge of network"
- client/server model
  - client host requests, receives service from always-on server
  - e.g. Web browser/server; email client/server
- peer-peer model
  - minimal (or no) use of dedicated servers
  - e.g. Skype, BitTorrent



Routing

- Goal: move packets among routers from source to destination
- It is an issue in packet switched networks
- datagram network:
  - ➡ destination address determines next hop
  - ➡ routes may change during session
  - ➡ analogy: driving, asking directions
- virtual circuit network:
  - ➡ each packet carries tag (virtual circuit ID), tag determines next hop
  - ➡ fixed path determined at call setup time, remains fixed thru call
  - ➡ routers maintain per-call state



Network Taxonomy