HY556 - Distributed Systems

Professor: Panagiota Fatourou

Spring 2011

Definition of a Distributed System (1)

In this course, a distributed system will be:

> A collection of independent components that communicate and coordinate their actions by passing messages. This collection appears to its users as a single coherent system.

Examples

- Internet
- Intranet (portion of the Internet managed by an organization)
- Mobile and ubiquitous computing

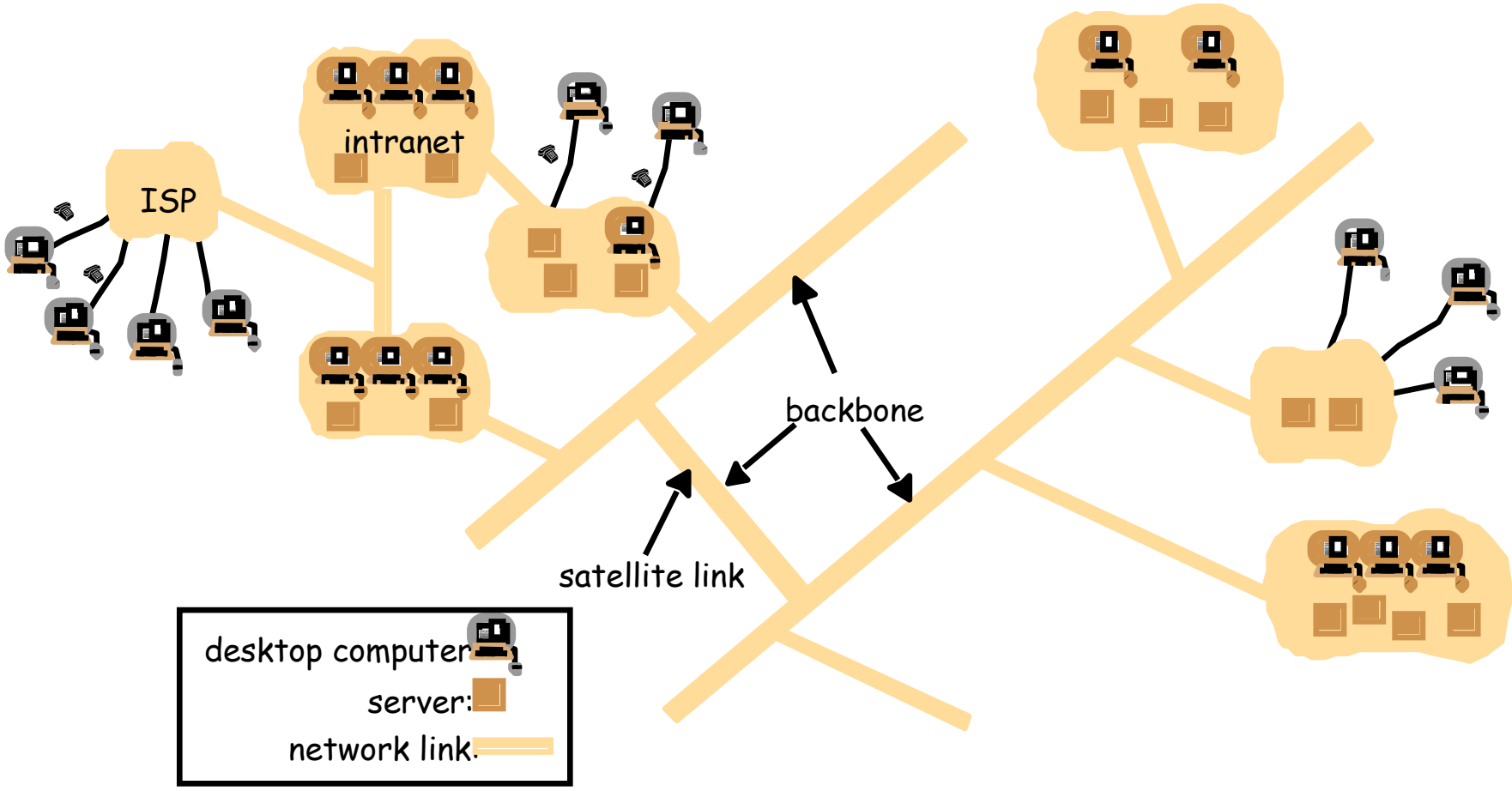
Consequences of the definition

- Concurrency
  - Concurrent program execution
  - Sharing of resources
  - Increase of system capacity by adding more resources
- No global clock
  - There are limits to the accuracy with which computers may synchronize their clocks
- Independent failures
  - Computer isolation
  - Node or link crashes
  - Byzantine failures

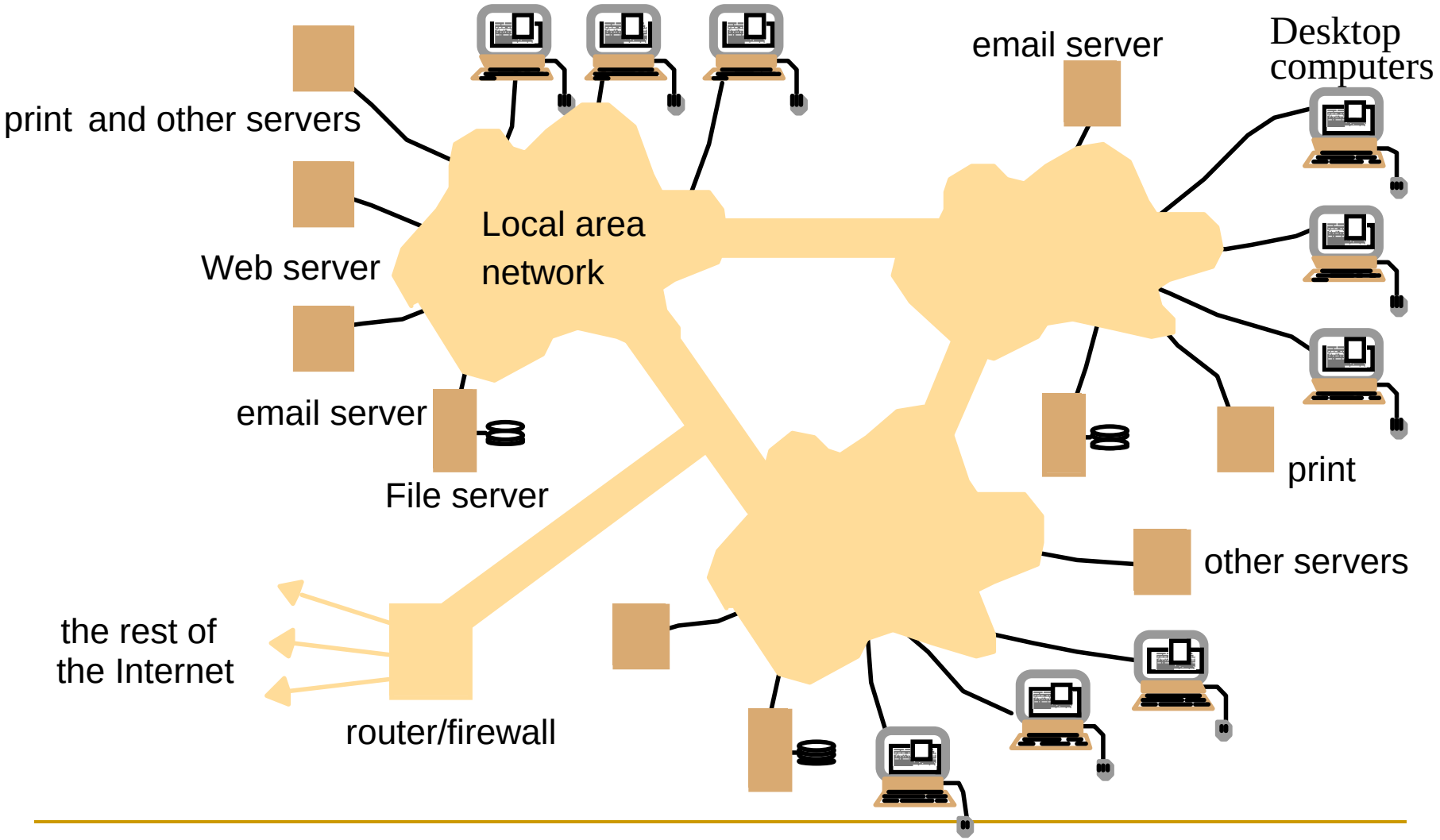
Resources

- The term resource characterizes the range of things that can usefully be shared in a distributed system.
  - Hardware Components
    - Disks, printers, etc.
  - Software-defined entities
    - Files, databases, data objects of all kinds, etc.

Examples of Distributed Systems
A typical portion of the Internet



Examples of Distributed Systems - A typical Intranet

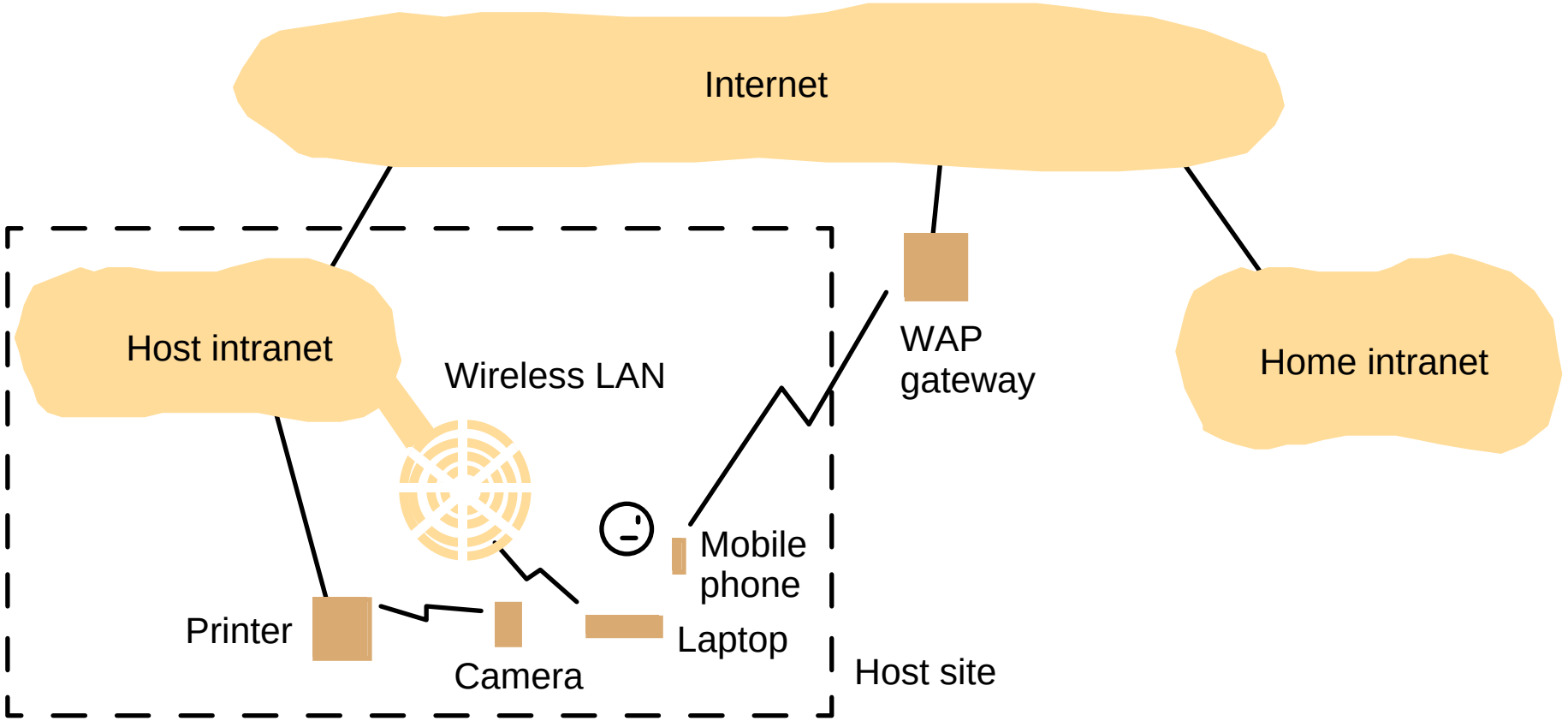


Mobile and Ubiquitous Computing- Portable and handheld devices in a distributed system

- **Mobile Computing**
  - Laptops
  - Handheld devices (personal digital assistants, mobile phones, video cameras, digital cameras
  - Wearable devices (smart watches)
  - Location-aware or content-aware computing

- Ubiquitous Computing
  - The term **ubiquitous** is intended to suggest that small computing devices will become so pervasive in everyday objects that their computational behavior will be transparently tied up with their physical function.
    - Devices embedded in appliances (washing machines, wi-fi systems, cars, etc.)
    - universal remote control devices
    - Watch when the washing is done

Mobile and Ubiquitous Computing- Portable and handheld devices in a distributed system



Resource Sharing

- **Service**: a distinct part of a computer system that manages a collection of related resources and present their functionality to users and applications.
  - File service, printing services, electronic payment services, etc.
- A server is a running program (a process) on a distributed system that accepts requests from programs and responds appropriately.
- The requesting processes are called clients. Clients invoke operations.
- Remote invocation: a complete interaction between a client and a server.

- **Examples**
  - WWW (web browsers, web servers)
  - Email
  - Networked printers

World Wide Web

- WWW allows to the user
  - to retrieve and view documents of many types (audio, video, etc.)
  - to interact with an unlimited set of services
- Hypertext structure
  - links (hyperlinks) from documents to other documents
- Open system
  - Operation is based on communication standards and document standards that are freely published and well implemented.
    - (many types of browsers, many types of servers, etc.)
  - Open to the types of resources that can be published or shared on it
    - New-image formats can be supported
      - (helper applications and plug-ins)

World Wide Web

- HyperText Markup Language (HTML)

```
<IMG
    SRC=http://www.cdk4.net/WebExample/Images/earth.jpg
<P>
Welcome to Earth! Visitors may also be interested in taking a
    look at the
<A HREF=http://www.cdk4.net/WebExample/moon.html>
    Moon</A>
</P>
```

  - Contents of a web page
  - .html
  - web server name
  - tags
  - URL

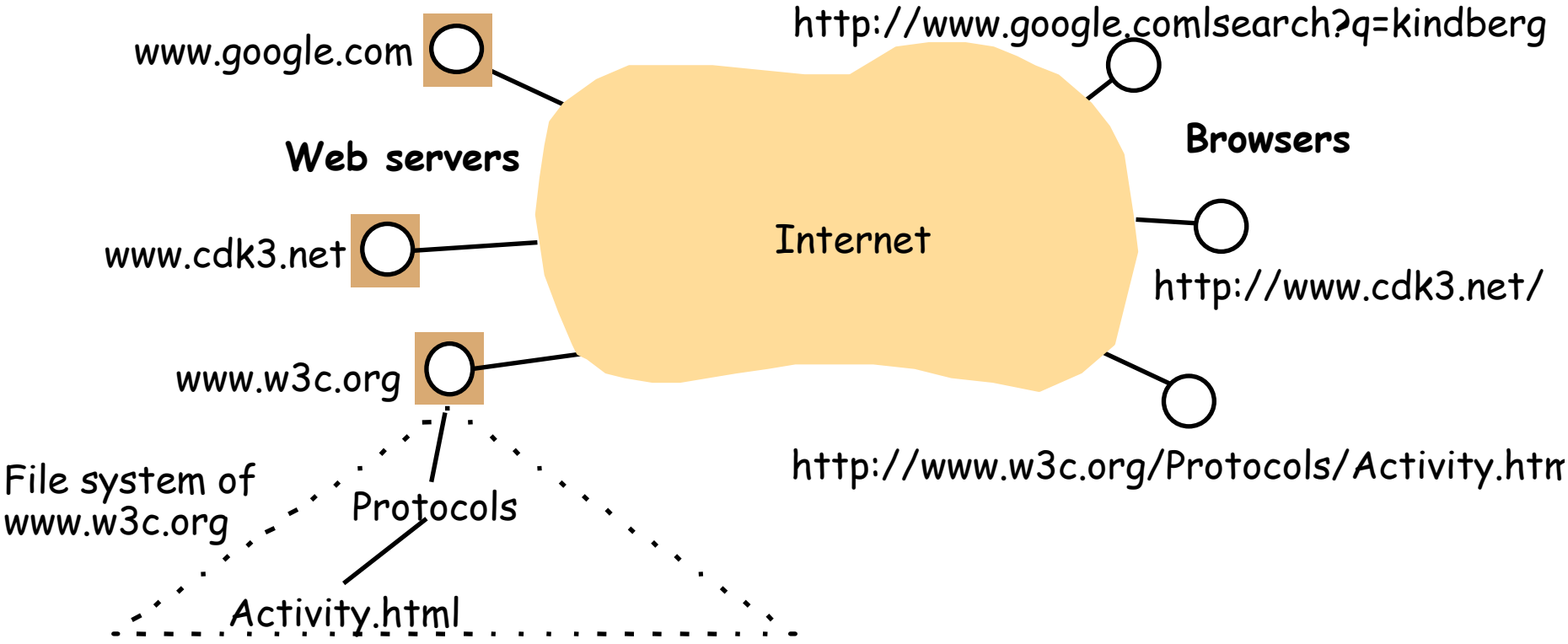
World Wide Web

- Uniform Resource Locators (URLs)
  - Every URL has two top-level components

    Scheme: scheme-specific-identifier

    - Common schemes: mailto:, ftp:, http:, nntp: (a Usenet newsgroup), mid: (e-mail message)
  - New types of resources can be added
    - <name of new-type of resource>: (a protocol is needed for accessing the new type of resources -> add a plug-in: gives the capability to a browser to use the new protocol

Web servers and web browsers



http://servername[:port] [/pathname] [?query] [#fragment]

~username, ~/public_html, index.html

World Wide Web

- Client-server system with standard rules for interaction (HyperText Transfer Protocol-HTTP)
  - Request-reply interaction
  - Content types
    - Not all browsers manage all types of content
    - Browsers include a list of the type of contents they are interested in when they issue a request
    - The server includes the content type in the reply message (the strings that denote the content type are called MIME types -> standardized
  - One resource per request
  - Simple access control

World Wide Web

- Dynamic pages (forms)
  - URL designates a program on the server
  - Common Gateway Interface (CGI): program that web servers run to generate content for their clients
  - Java-scripts (are downloaded with a web form to provide better-quality interaction with the user)
  - Applet: is of more general functionality than Java-scripts. Applets are downloaded automatically and run when the browser fetches a corresponding web page.
    - May access the network
    - provide customized user interfaces
    - JAVA based

World Wide Web

- Dangling links
- Users often get lost in the hyperspace
- Search engines
- Scalability issues for popular web servers

Challenges - Heterogeneity

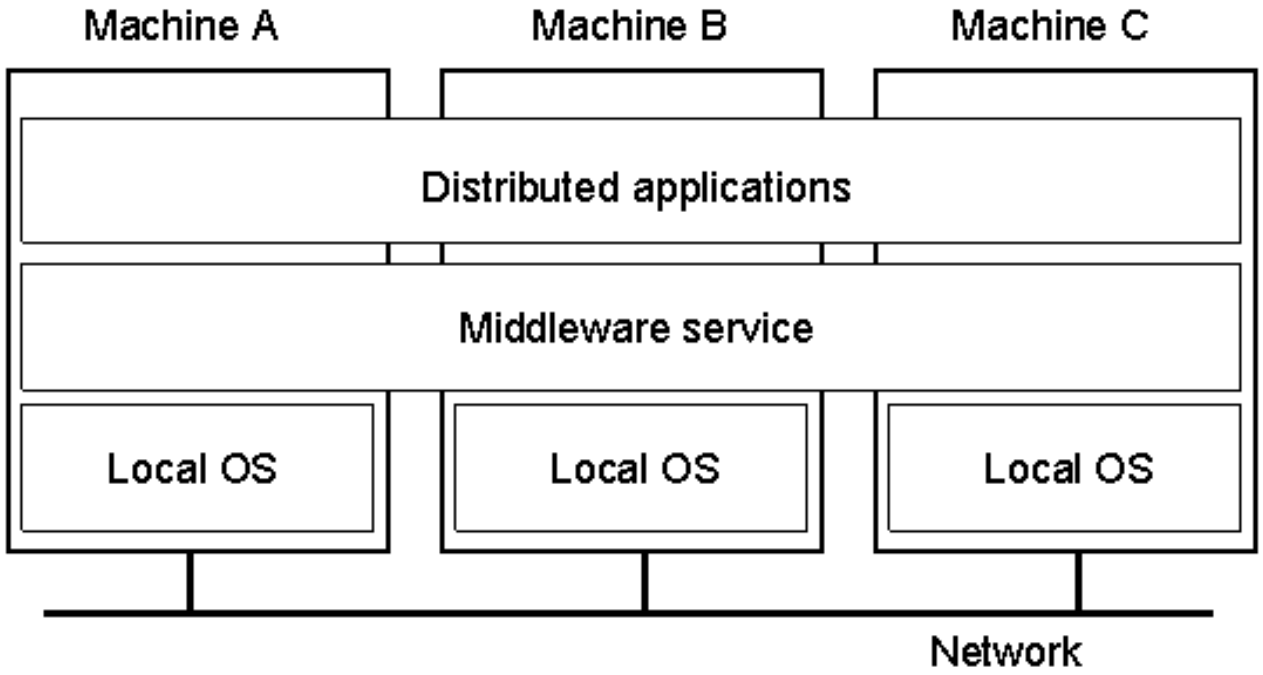
- Heterogeneity applies to all of the following:
  - Networks
    - Use Internet protocols for communication
  - Computer hardware
    - Data types may be represented in different ways on different sorts of hardware
  - Operating systems
    - The interface for implementing the internet protocol is not the same (messages in Unix or in Windows)
  - Programming languages
    - Different representations for characters and data structures (arrays, structs, etc.)
  - Implementations by different developers
    - Standards need to be agreed and adopted

Challenges - Heterogeneity

Middleware

- A software layer that provides a programming abstraction as well as masking the heterogeneity of the underlying networks, hardware, operating systems and programming languages.
  - CORBA (Common Object Request Broker)
  - Implemented over the Internet protocols
  - Provides a uniform computational model for use by the programmers of servers and distributed applications.

Challenges - Heterogeneity



A distributed system organized as middleware.
Note that the middleware layer extends over multiple machines.

Challenges - Heterogeneity

Mobile Code

- Applets

Virtual Machine

- Provides a way of making code executable on any hardware
  - The compiler generates code for a virtual machine (of a particular language) instead of hardware order code
  - The virtual machine of the language needs to be implemented once for each type of hardware to enable programs to run.

Challenges - Openness

- Determines whether the system can be extended and re-implemented in various ways.
- Expresses the degree to which new resources sharing services can be added and be made available for use by client programs.
- Is achieved by providing appropriate specification and documentation of the key software interfaces of the components of a system

Example

- Documents of the Internet protocols -> RFCs (Requests For Comments)
- Provide a uniform communication mechanism and published interfaces for access to shared resources.
- Each added component should conform to the published standard.

Challenges - Security

- Confidentiality
  - Protection against disclosure to unauthorized users
- Integrity
  - Protection against alteration or corruption
- Availability
  - Protection against interference with the means to access the resources
  - Denial of service attacks
- Security of mobile code

Challenges - Scalability

Concept	Example
Centralized services	A single server for all users
Centralized data	A single on-line telephone book
Centralized algorithms	Doing routing based on complete information

Examples of scalability limitations.

Challenges - Scalability

- Controlling the cost of physical resources
  - As the demand for a resource grows, it should be possible to extend the system, at reasonable cost, to meet it.
- Controlling the performance loss
  - Algorithms that use hierarchical structures scale better than those that use linear structure.
- Preventing software resources running out
  - IP Addresses (32 bits are no longer enough) -> 128 bit Internet addresses
- Avoiding performance bottlenecks
  - Use of decentralized algorithms
  - Caching and replication (frequently accessed shared resources)

Challenges – Transparency

Transparency	Description
Access	Enables local and remote resources to be accessed using identical operations
Location	Enables resources to be accessed without knowledge of their physical or network location
Concurrency	Enables several processes to operate concurrently using shared resources without interference between them.
Replication	Enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers
Failure	Hide the failure and recovery of a resource
Mobility	Allows the movement of resources and clients within a system without affecting the operation of users or programs.
Performance	Allows the system to be reconfigured to improve performance as loads vary.
Scaling	Allows the system and applications to expand in scale without change to the system structure or the application algorithms.

Challenges – Failure Handling

- Detecting failures
  - Use of checksums to detect corrupted data
- Masking failures
  - Messages can be re-transmitted when they fail to arrive
  - File data can be written to a pair of disks to ensure availability
- Tolerating failures
  - Involve the user in tolerating failures rather than having him/her waiting for ever (web browsers that fail in contacting the web server)
- Recovery from failures
  - Design of software so that the state of permanent data can be recovered or "rolled back" after a server crash.
- Redundancy
  - two different routes between any two routers in the Internet
  - replicate the name table in DNS
  - replication of databases)

Challenges – Concurrency

- Any object that represents a shared resource must be responsible for ensuring that it operates correctly in a concurrent system
  - Example: banking accounts
- Synchronization of operations to ensure data consistency

An Example Client and Server (1)

```
/* 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           */
};
```

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

An Example Client and Server (2)

```
#include <header.h>
void main(void) {
     struct message m1, m2;                    /* incoming and outgoing messages      */
     int r;                                    /* result code                         */

     while(TRUE) {                             /* server runs forever                 */
          receive(FILE_SERVER, &m1);           /* block waiting for a message         */
          switch(m1.opcode) {                  /* dispatch on type of request         */
               case CREATE:    r = do_create(&m1, &m2); break;
               case READ:      r = do_read(&m1, &m2); break;
               case WRITE:     r = do_write(&m1, &m2); break;
               case DELETE:    r = do_delete(&m1, &m2); break;
               default:        r = E_BAD_OPCODE;
          }
          m2.result = r;                       /* return result to client             */
          send(m1.source, &m2);                /* send reply                          */
     }
}
```

A sample server.

An Example Client and Server (3)

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

A client using the server to copy a file