

# ***Computer Science 425*** ***Distributed Systems***

***CS 425 / CSE 424 /  
ECE 428***

**Klara Nahrstedt**  
**August 25, 2009**

# **Acknowledgement**

- **The slides during this semester are based on ideas and material from the following sources:**
  - Slides prepared by Professors M. Harandi, J. Hou, I. Gupta, N. Vaidya, Y-Ch. Hu, S. Mitra.
  - Slides from Professor S. Gosh's course at University of Iowa.

# ***Course Staff – how to meet us***

## **In person (Office Hours):**

- **Klara Nahrstedt: 3104 Siebel Center**
  - Every Wednesday 9-10am, and Every Thursday 9-10am
- **TA, Ying Huang: 207 Siebel Center**
  - Every Monday: 3-4pm, and Every Thursday 3:30-4:30pm

## **Virtually:**

- **Newsgroup: class.cs425 (most preferable, monitored daily)**
- **Email (turnaround time may be longer than newsgroup)**
  - Klara Nahrstedt: [klara@illinois.edu](mailto:klara@illinois.edu)
  - Ying Huang: [huang23@illinois.edu](mailto:huang23@illinois.edu)

# How will you Learn?

*Take a look at handout “Course Information and Schedule”*

- <http://www.cs.uiuc.edu/class/fa09/cs425>
- Text: Colouris, Dollimore and Kindberg (4<sup>th</sup> edition)
- Lectures
- Homework Sets (4 Sets)
  - Approx. one every two weeks
  - Solutions need to be typed, figures can be hand-drawn
- Programming assignments (3 MPs)
  - Incremental, in 3 stages
  - We will build a *peer to peer system* for mobile phones
  - **Form a group up to 3 members for each programming assignment**
  - Programming Language needed: Java
- Exams
  - **Midterm – Tuesday, October 13, 2-3:15pm in class**
  - **Final Exam – Wednesday, December 16, 7-10pm, location TBD**

# On the Textbook

- Text: Colouris, Dollimore and Kindberg (**4<sup>th</sup> edition**). “White book”.
- The 3<sup>rd</sup> edition will suffice for most material too. However, we will refer to section, chapter, and problem numbers **only in the 4<sup>th</sup> edition**.
  - The 3<sup>rd</sup> edition may have a different numbering for some HW problems (that we give from the textbook). Make sure you solve the right problem; the responsibility is yours (no points for solving the wrong problem!)

# ***What assistance is available to you?***

- **Lectures**
  - lecture slides will be placed online at course website
    - » “Tentative” version before lecture
    - » “Final” version after lecture
- **Homework** – office hours to help you (without giving you the solution)
  - 2 Homework Assignments before Midterm
  - 2 Homework Assignments before Final
- **Course Prerequisite:** Operating Systems/Systems Programming (CS 241 or CS 423 or instructor permission) and Java programming language
- **Compass tool** as grade-book where all grades will be posted.

# Programming Assignments

## MP1 :

Peer Registration to Server  
P2P Socket Programming  
Peer-Server Socket Programming  
GUI Design (register, view other  
Members send/receive msg)

**We will use Eclipse Emulator**

## MP2:

Build P2P File Sharing with  
Membership Service,  
Functions: File Search,  
Insert, Delete, Browse

**We will use Eclipse Emulator and G1 Phones**

## MP3:

Design and Implement a P2P application  
(e.g., find a group member , SVN: cooperative  
Editing, chat: group conversation log, fault-tolerant  
P2P file sharing using coding, application that utilizes  
G1 phone hardware such as WiFi, Bluetooth, GPS,  
accelerometers

**We will use Eclipse and G1 Phones**

Peer Registration  
Server

G1 Phone (Peer)

Underlying  
Infrastructure-based  
WiFi /TCP/IP Network

Peer

Peer

# ***Course Logistics***

- **Exams, Homework and Machine Problems (Programming Assignments) deadlines on the website**
  - <http://www.cs.uiuc.edu/class/fa09/cs425/lectures.html>
- **Grade Distribution:**
  - Homework: 20% (each HW 5% of the grade)
  - Programming Assignments: 40% (MP1:10%, MP2: 16%, MP3: 14%)
  - Midterm: 10%
  - Final Exam: 30% (comprehensive)



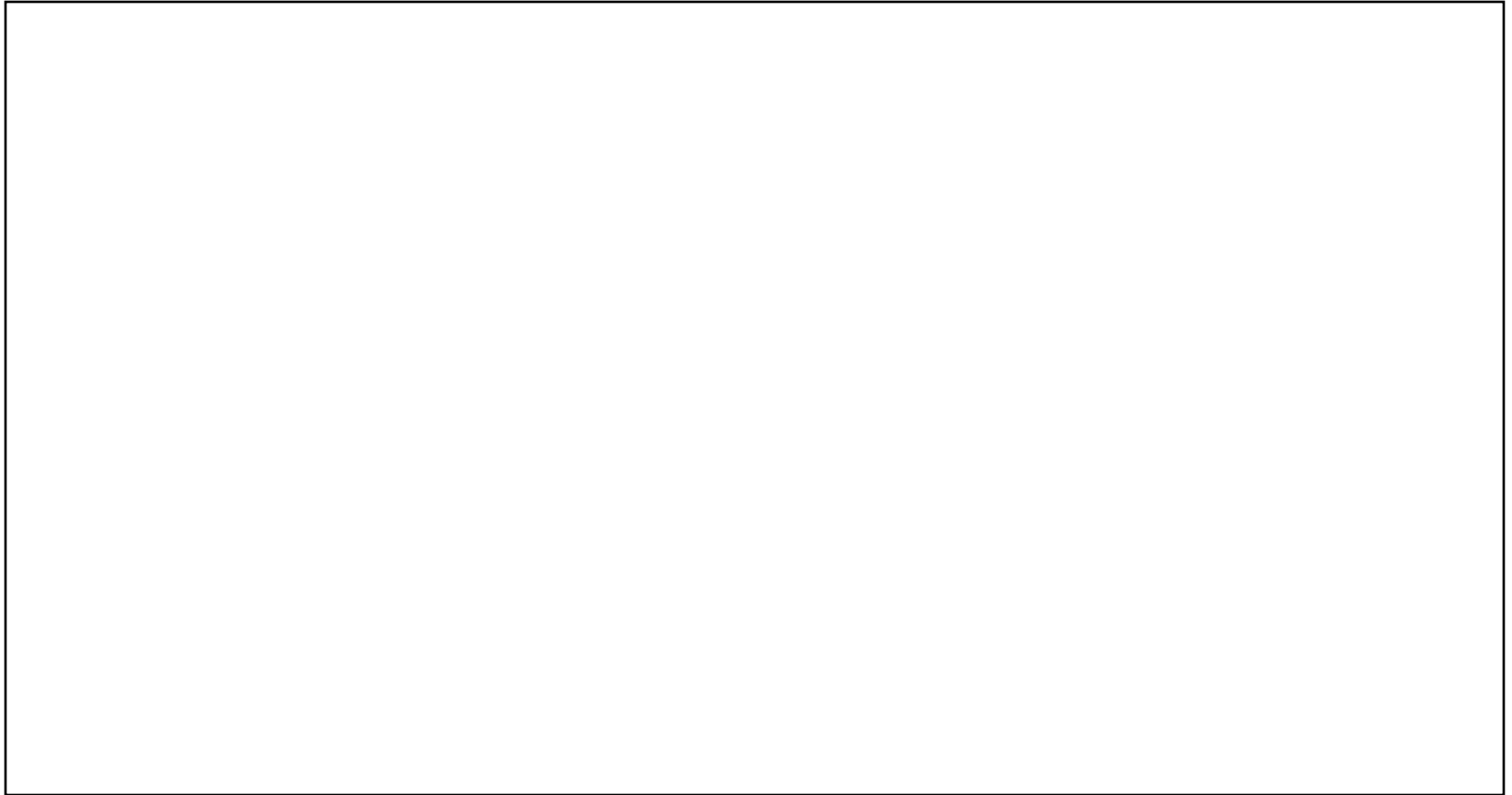
# ***To Do List***

- **Create groups of 2-3 students** for machine problems programming assignments (use newsgroup, TA to find a group member)
- **Deadline for group creation September 1 (Tuesday)** – email TA the names of your group members
  - Based on the group names TSG will create group directories where the group can build the MPs
- **Check carefully the class website will all the deadlines for homework, machine problems, exams**

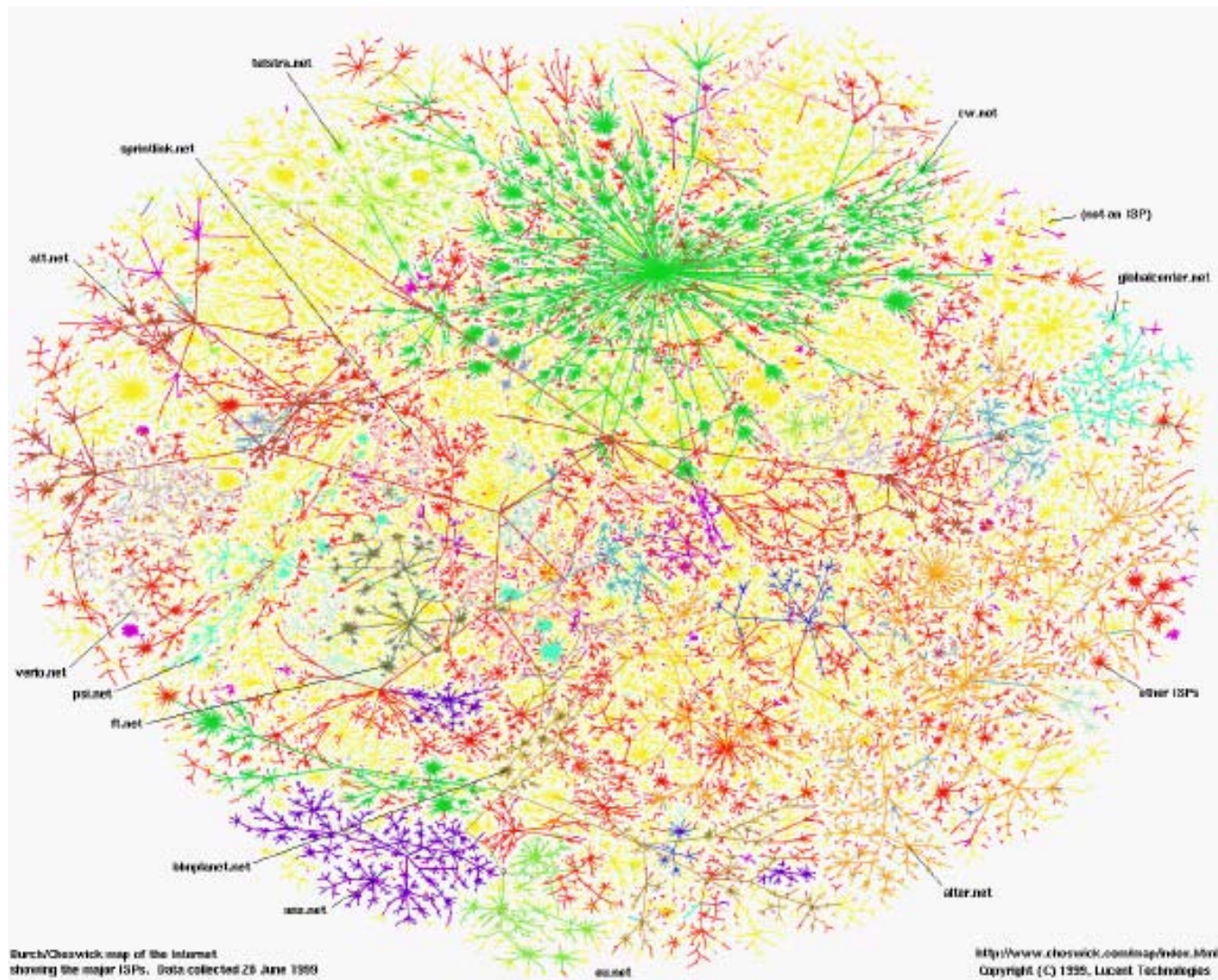
# ***Our Only Goal Today***

**To Define the Term Distributed System**

***Can you name some examples of Distributed Systems?***

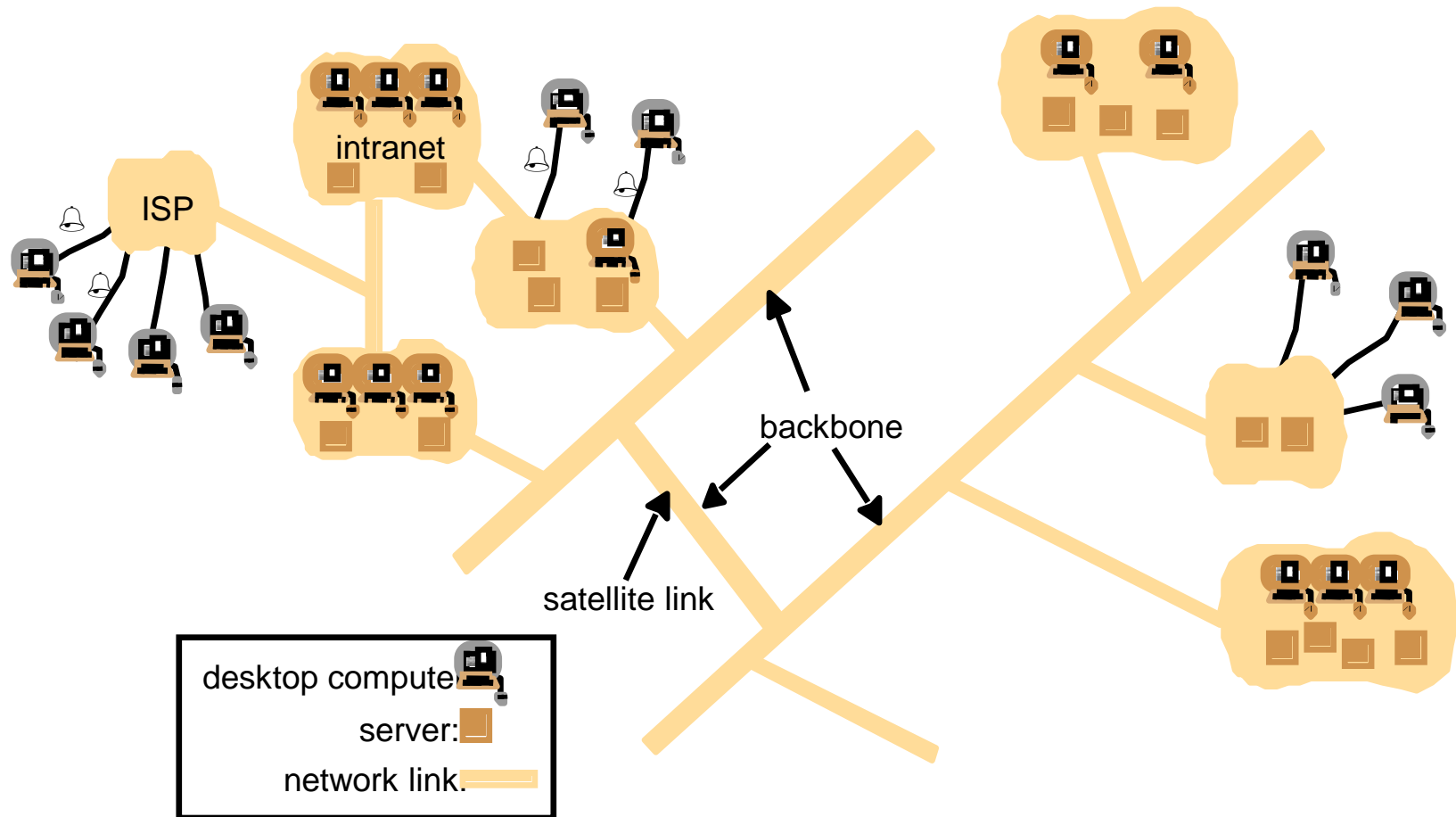
A large, empty rectangular box with a thin black border, intended for the user to write their answer to the question above.

# 1. Internet

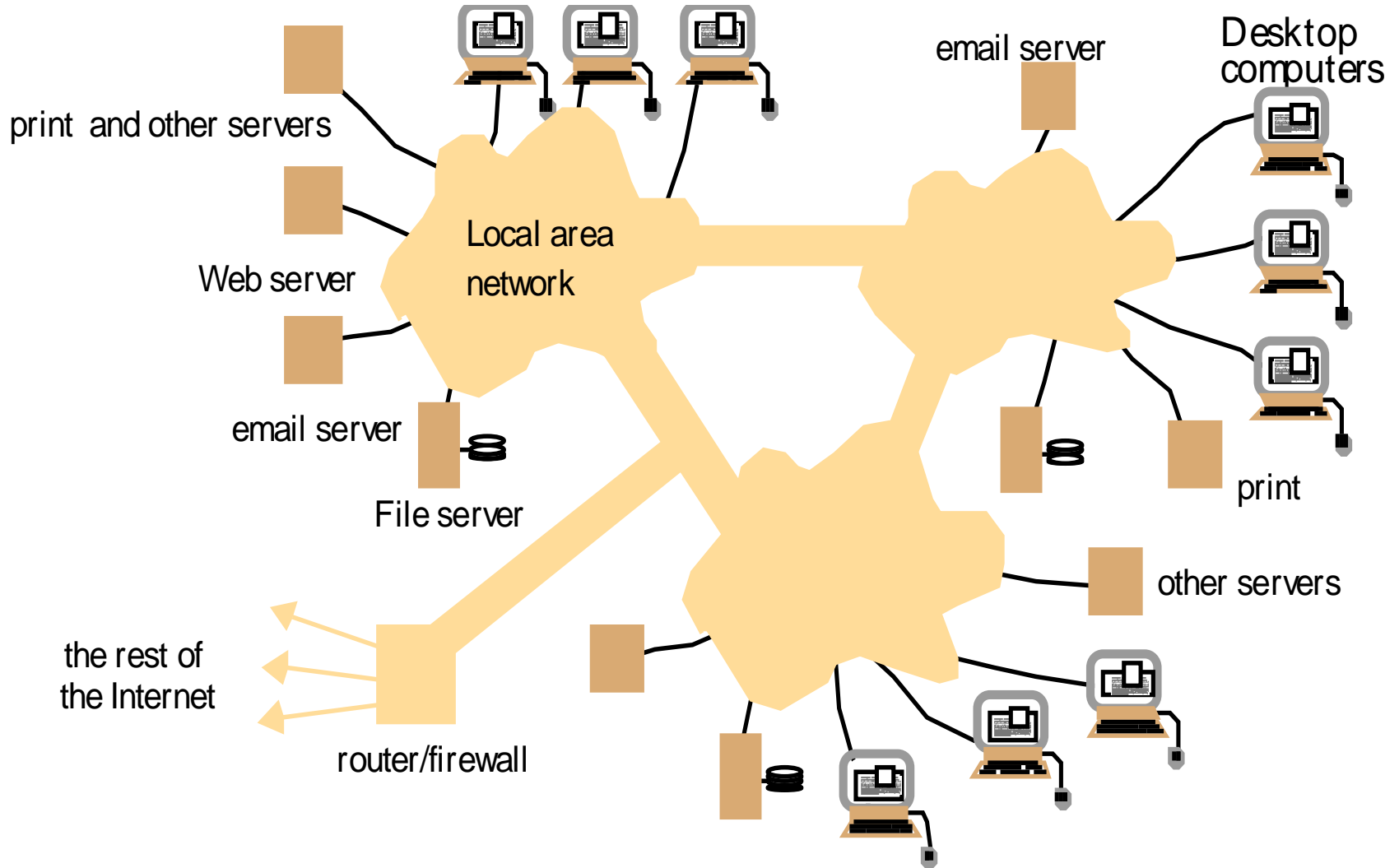


Internet Mapping Project, color coded by ISPs

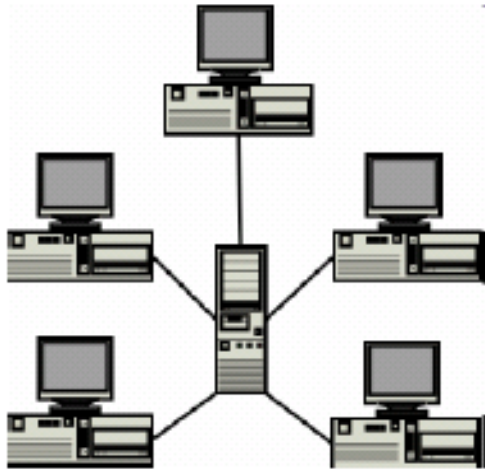
# *A typical portion of the Internet*



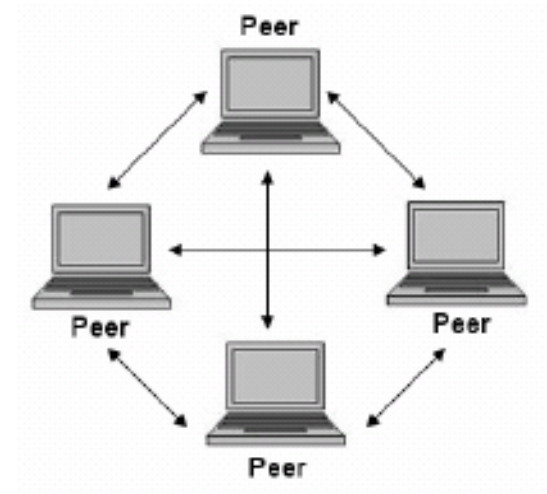
# *A typical intranet*



## 2. Peer-to-Peer Overlays



client-server  
e.g., NFS



peer-to-peer  
e.g., KaZaa, napster





### 3. Sensor Network



Checking the structural integrity



## 4. Distributed Mobile Robots



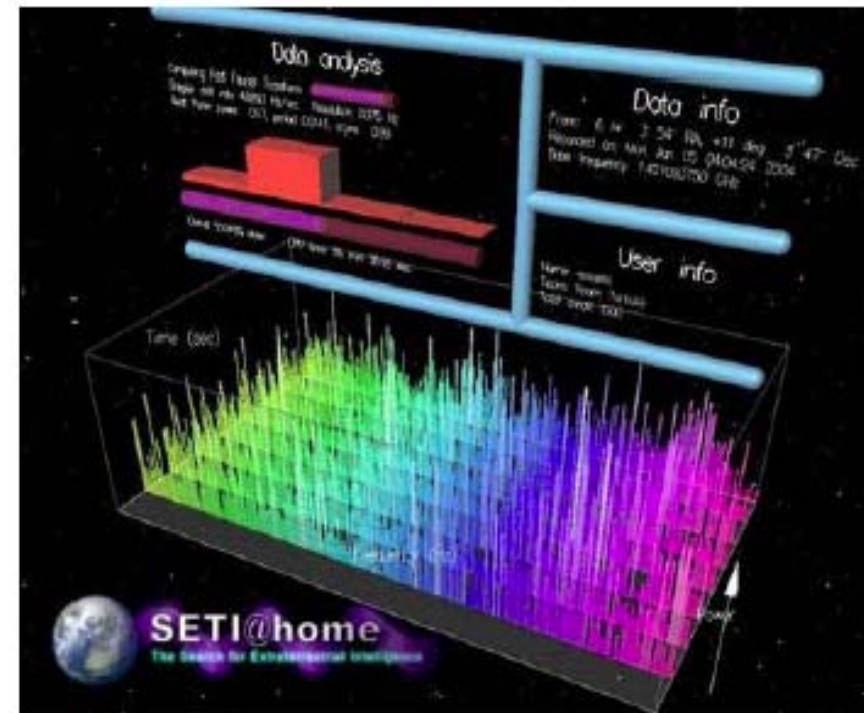
Kiva robot

500 robots coordinate to  
manage inventory in a  
warehouse



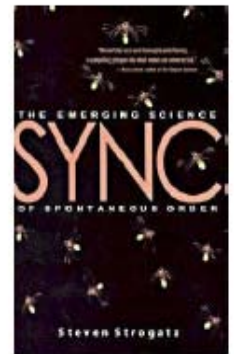
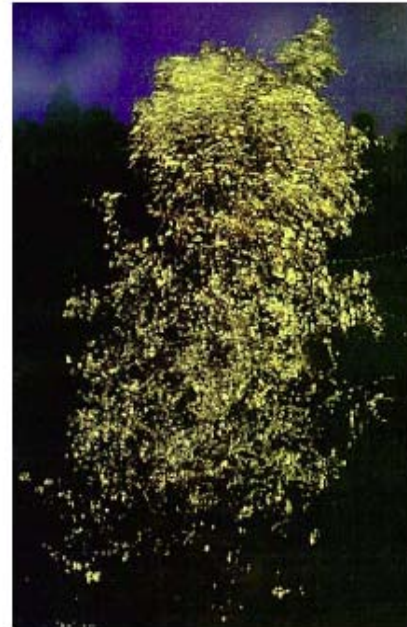
# 5. Distributed Computation Grids

- **Seti@Home**
- **Uses Internet-connected computers in the Search for Extraterrestrial Intelligence**
- **Downloads and analyzes telescope data chunks on your computer and sends back results to Seti servers**

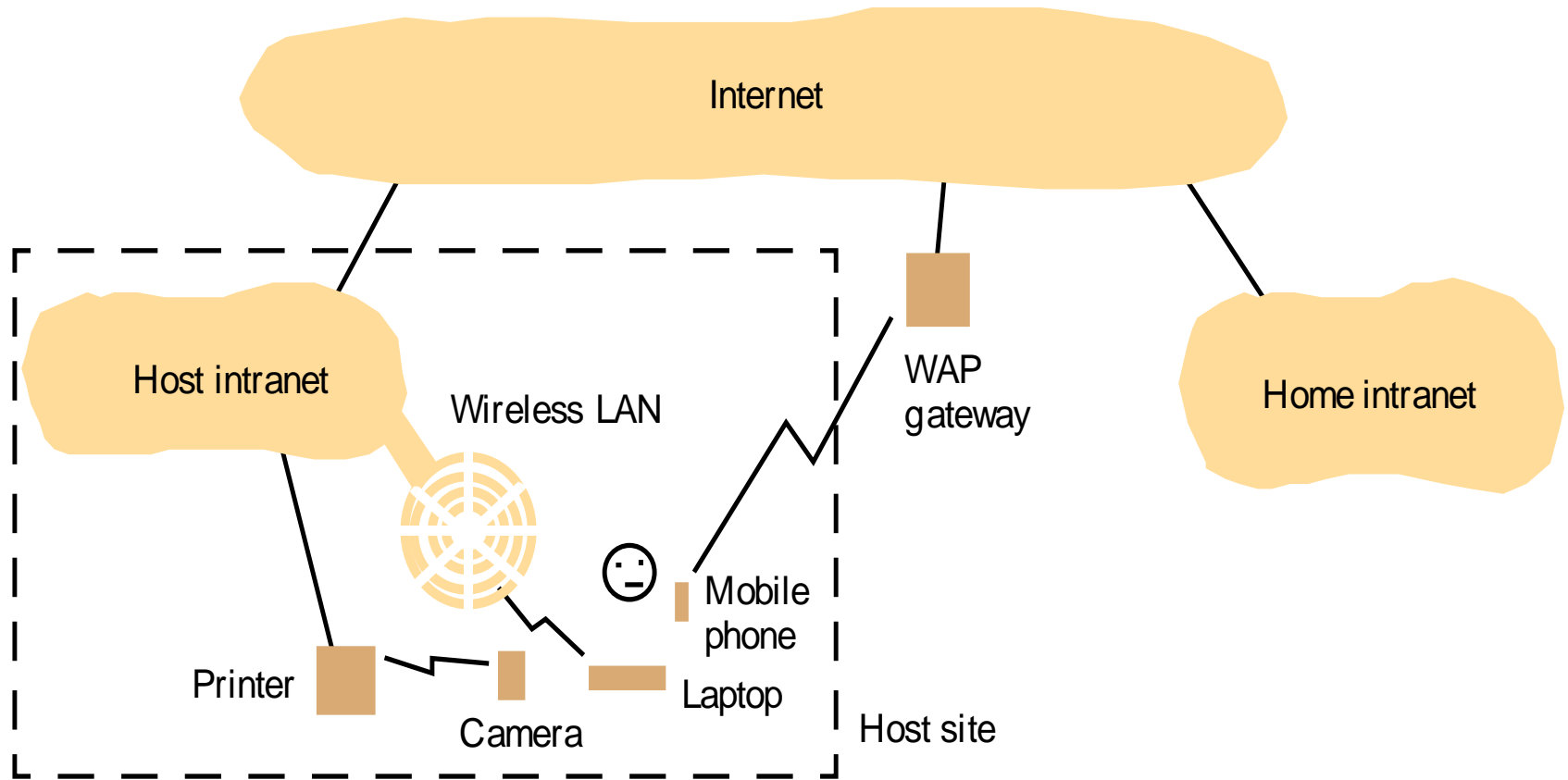


## 6. *Distributed Structures in Nature*

- **Tropical Fireflies synchronize their flashes precisely among large groups**
- **Example of biological synchronicity**



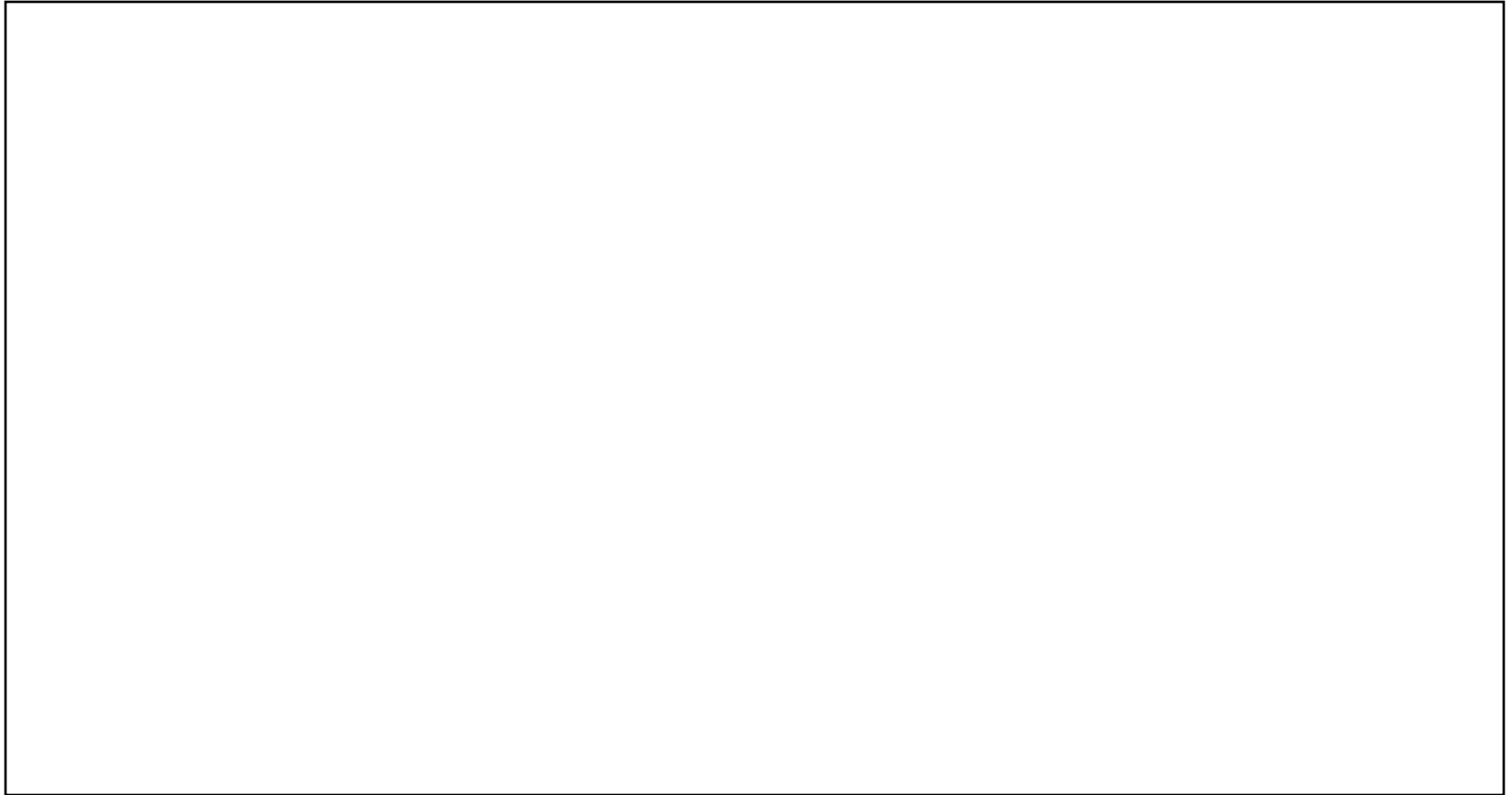
## 7. Portable and handheld devices in a distributed system



# ***Can you name some examples of Distributed Systems?***

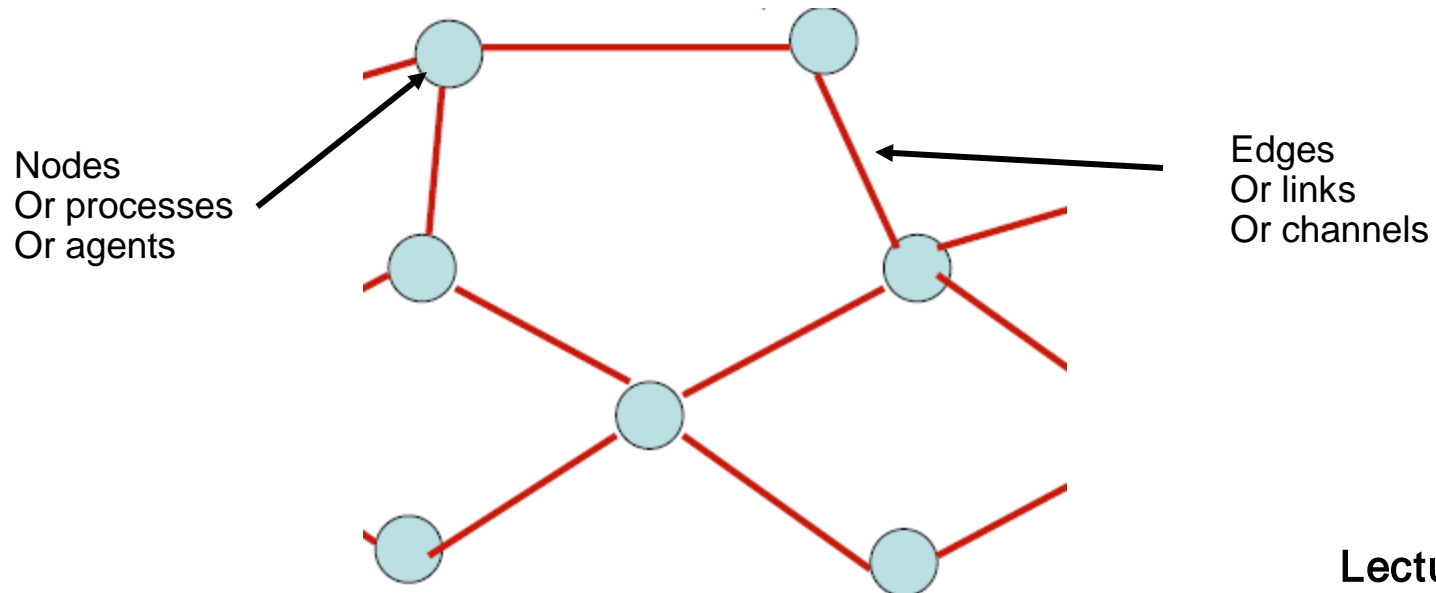
- **Client-Server**
- **Web**
- **Internet**
- **Mobile Cell-Phone Network**
- **Sensor Network**
- **Network of Robots**
- **DNS (Domain Name Service)**
- **Napster, KaZaA (peer to peer file sharing overlayss)**
- **Synchronized Fireflies**
- **Computing Grid**

# ***What is a Distributed System?***



# What is a Distributed System?

- It is a collection of entities (computers, robots, fireflies,..) where
  - Each of them is **autonomous, asynchronous** and [possibly] **failure-prone**
  - Communicating through [possibly] **unreliable channels** (Ethernet, wireless network, vision,...)
  - To perform some **common function** (detection, computation, communication)



# ***Why Distributed Systems?***

- **Geographic distributed of processes**
- **Resource sharing** (as used in P2P networks)
- **Computing speed up** (as applied in grid and clusters)
  - MapReduce – distributed computing framework to parallelize operation on large datasets (webpages)
  - Seti@Home
- **Fault tolerance** (as applied in Google Distributed File System (GFS))
  - On top of GFS is BigTable (Google Distributed Database)



- **CS 425 is an introduction to the key concepts for**
  - **Designing**
  - **Analyzing and**
  - **Implementing**

## **Distributed Systems**

# ***Key Design Issues***

- **Knowledge is local**
- **Clocks are not synchronized**
- **No shared address space**
- **Topology and routing**
- **Scalability and adaptability**
- **Faulty nodes and channels**
  - Processes may crash, processes may misbehave
  - Variable latency and bandwidth

# ***Fundamental Problems in Distributed Systems and their Analysis***

- Time synchronization
- Leader election
- Mutual exclusion
- Distributed snapshot
- Routing
- Consensus
- Replica management
- Transactions

**We will study and analyze algorithms for these types of problems**

# ***Approaching Implementation of Distributed Systems***

- In implementing distributed systems we have to know the nature of underlying **computing and networking infrastructure**
- What can we assume?
  - How large are message delays and latencies?
  - Are processor clocks synchronized?
  - Do processors and links fail?
  - Could processes become malicious?
  - Are channels secure from eavesdropping and corruption?
  - What are the allowed primitive HW/OS operations? What are the atomic operations?
- Different answers give rise to different models for a distributed system
- We will study algorithms for fundamental problems in the context of certain **models**

# Problems and Models

- Time synchronization
- Leader election
- Mutual exclusion
- Distributed snapshot
- Routing
- Consensus
- Replica management
- Transactions

- Message Passing Model
  - Synchronous (with bounded delays)
  - Asynchronous (with unbounded delays)
  - Partially synchronous (with bounded but unknown delays)
- Shared memory model

• . . . . .

# ***Example: Simple Communication Model Message Passing***

- **Assume distributed system topology as a graph  $G=(V,E)$** 
  - **V** = set of nodes (sequential processes)
  - **E** = set of edges (links or channels which could be unidirectional or bidirectional)
- **Actions by a process**
  - **Internal action**: sequential process computes
  - **Send action**: sends a message (put a message in channel) and perform computation
  - **Receive action**: receives a message (message taken out from channel) and performs computation

# Weak vs Strong Models

- One object (or operation) of a **strong model** - more than one objects(or operations) of a **weaker model**
- Often, weaker models are synonymous with **fewer restrictions**
- One can add layers (**additional restrictions**) to create a **stronger model** from weaker one

- **High Level Language** model is stronger than **assembly language** model
- **Asynchronous** is weaker than **synchronous** model
- **Bounded delay** is stronger than **unbounded delay** (channel)

# ***Model Transformation and Implementation***

- **Stronger models**
  - Simplify reasoning, but
  - Need extra work to implement
- **Weaker models**
  - Are easier to implement
  - Have a closer relationship with real world

- **Can model X be implemented using model Y?**
- **Examples**
  - Transformation of message passing to share memory
  - Transformation non-FIFO to FIFO channel



# ***Distributed Systems Design Goals***

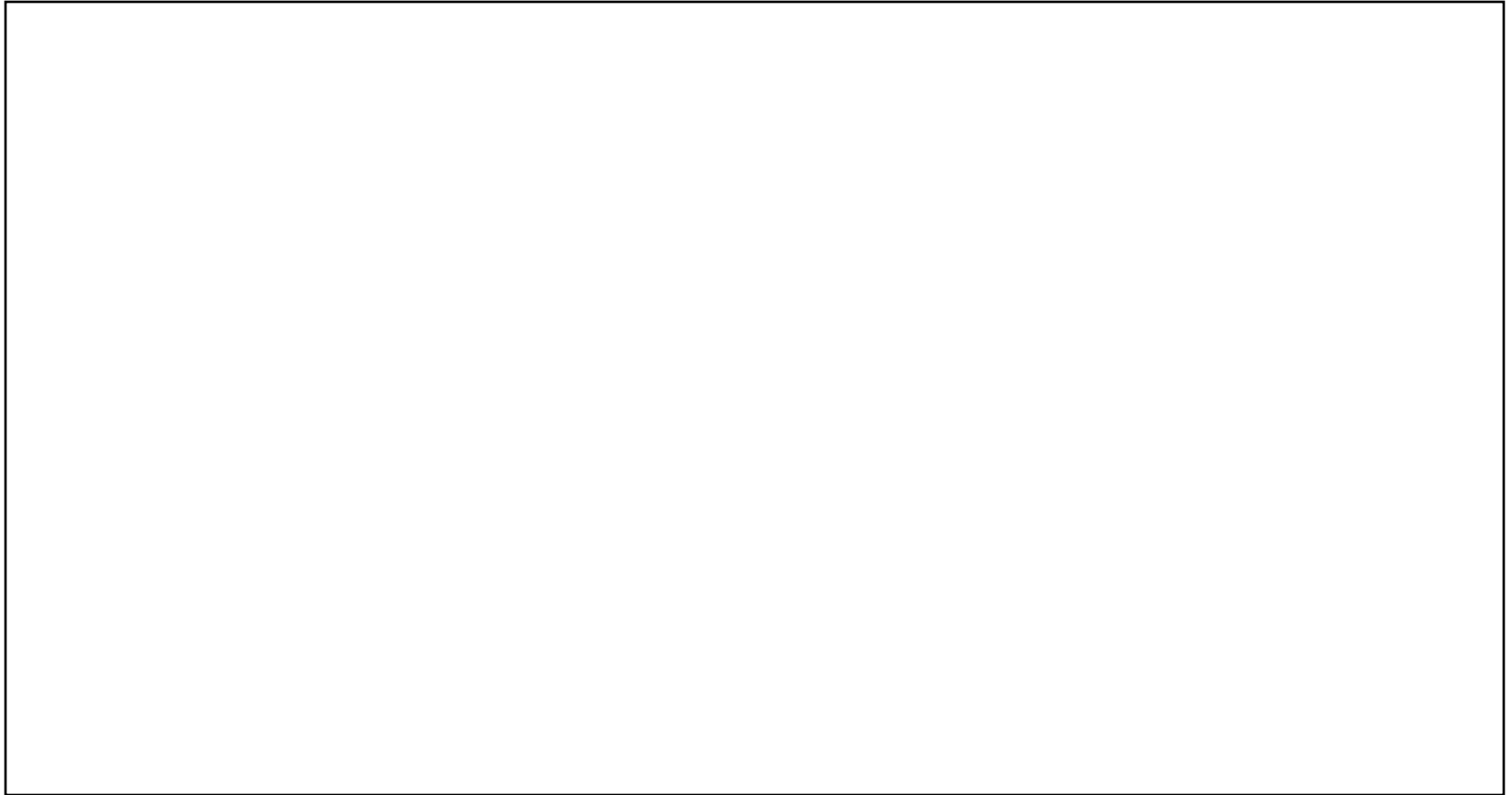
- **Common Goals:**

- **Heterogeneity** – can the system handle different types of PCs and devices?
- **Robustness** – is the system resilient to host crashes and failures, and to the network dropping messages?
- **Availability** – are data, services always there for clients?
- **Transparency** – can the system hide its internal workings from the users?
- **Concurrency** – can the server handle multiple clients simultaneously?
- **Efficiency** – is it fast enough?
- **Scalability** – can it handle 100 million **nodes**? (nodes=clients and/or servers)
- **Security** – can the system withstand hacker attacks?
- **Openness** – is the system extensible?

# ***Readings***

- **For today's lecture: Chapter 1**
- **For next Thursday's lecture:**
  - Read sections 11.1-11.4
  - Fill out and return Student InfoSheet

# ***Additional Slides***



# Heterogeneity

- **Variety and differences among**
  - Networks, hardware, Operating Systems
  - Programming languages
  - Implementations
- **Middleware** – software layer that provides programming abstract as well as
  - Masking heterogeneity of the underlying networks, hardware, operating systems and programming languages
  - Providing uniform computational models, e.g., remote object invocation, remote event notification
- **Virtual machine**
  - Compiler for a particular language generates code for VMs, instead for particular hardware code to assist execution of mobile code

# Openness

- Degree to which new resource-sharing services can be added and used
- Requires **publication of interfaces** for access to shared resources
- Requires **uniform communication** mechanism
- Conformance of each component to the published standard must be **tested and verified**

- **Security has three components**
  - **C**onfidentiality (protection against disclosure to unauthorized individuals)
  - **I**ntegrity (protection against alternation or corruption)
  - **A**vailability (protection against interference with means to access the resources)
- **Two security challenges (examples):**
  - Denial of service attacks
  - Mobile code security

# Scalability

- **System is said to be scalable if it will remain effective when there is a significant increase in the number of resources and users**
  - Controlling **cost of resources**
  - Controlling **performance loss**
  - Preventing **software resources running out** (e.g., IP addresses)
  - Avoiding **performance bottlenecks**

# Failure Handling

- **Failure in DS is partial – some components fail while the rest is functional**
  - **Detecting failures** (remote site crash or delay in message transmission)
  - **Masking failures** (message retransmission, file replication)
  - **Tolerance for failure** (clients give up after a predetermined number of attempts and take other actions)
  - **Failure recovery** (check-point and rollback recovery)
  - **Redundancy** (multi-path routing, replicated database, replicated DNS)



# Concurrency

- Is a problem when **two or more users access to the same resource** by two at the same time
  - Each resource is encapsulated as an object and invocations are executed in concurrent threads
  - Concurrency can be maintained by use of semaphores and other mutual exclusion mechanisms

# Transparency

- **Concealment of the separation of components from users:**
  - **Access transparency:** local and remote resources can be accessed using identical operations
  - **Location transparency:** resources can be accessed without knowing their whereabouts
  - **Concurrency transparency:** processes can operate concurrently using shared resources without interferences
  - **Failure transparency:** faults can be concealed from users/applications
  - **Mobility transparency:** resources/users can move within a system without affecting their operations
  - **Performance transparency:** system can be reconfigured to improve performance
  - **Scaling transparency:** system can be expanded in scale without change to the applications

# ***Transparency Examples***

- **Distributed File System** allows access transparency and location transparency
- **URLs** are location transparent, but are not mobility transparent
- **Message retransmission** governed by TCP is a mechanism for providing failure transparency
- **Mobile phone** is an example of mobility transparency