Distributed Systems

4 DS Architectures

Architectural Style

System Architectures

May-04-2009

Summer Term 2009

System Architecture Group

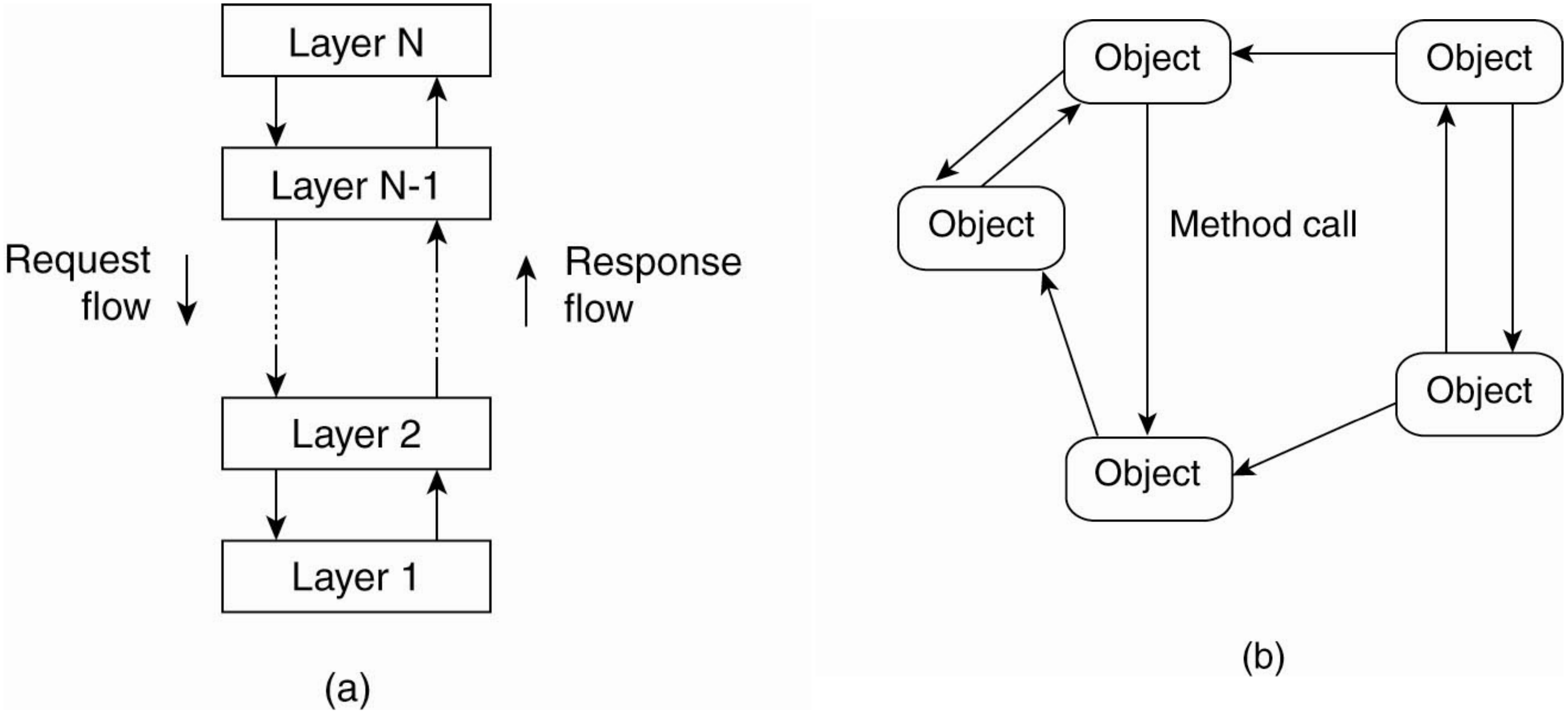
Roadmap of Today

- Architectural Styles
- Software Architectures
- System Architectures
  - Centralized SA (Client/Server)
  - Decentralized SA (P2P)
  - Hybrid SA

Architectural Styles of DS

- Layered architectures
  - Traditional software architecture
- Object-based architectures
  - Modern software architectural style
- Client/Server Systems
  - Well-understood and in use world-wide
- Peer to Peer System (P2P)
  - Depending on P2P protocol highly scalable

Layered vers. Object Based Architecture

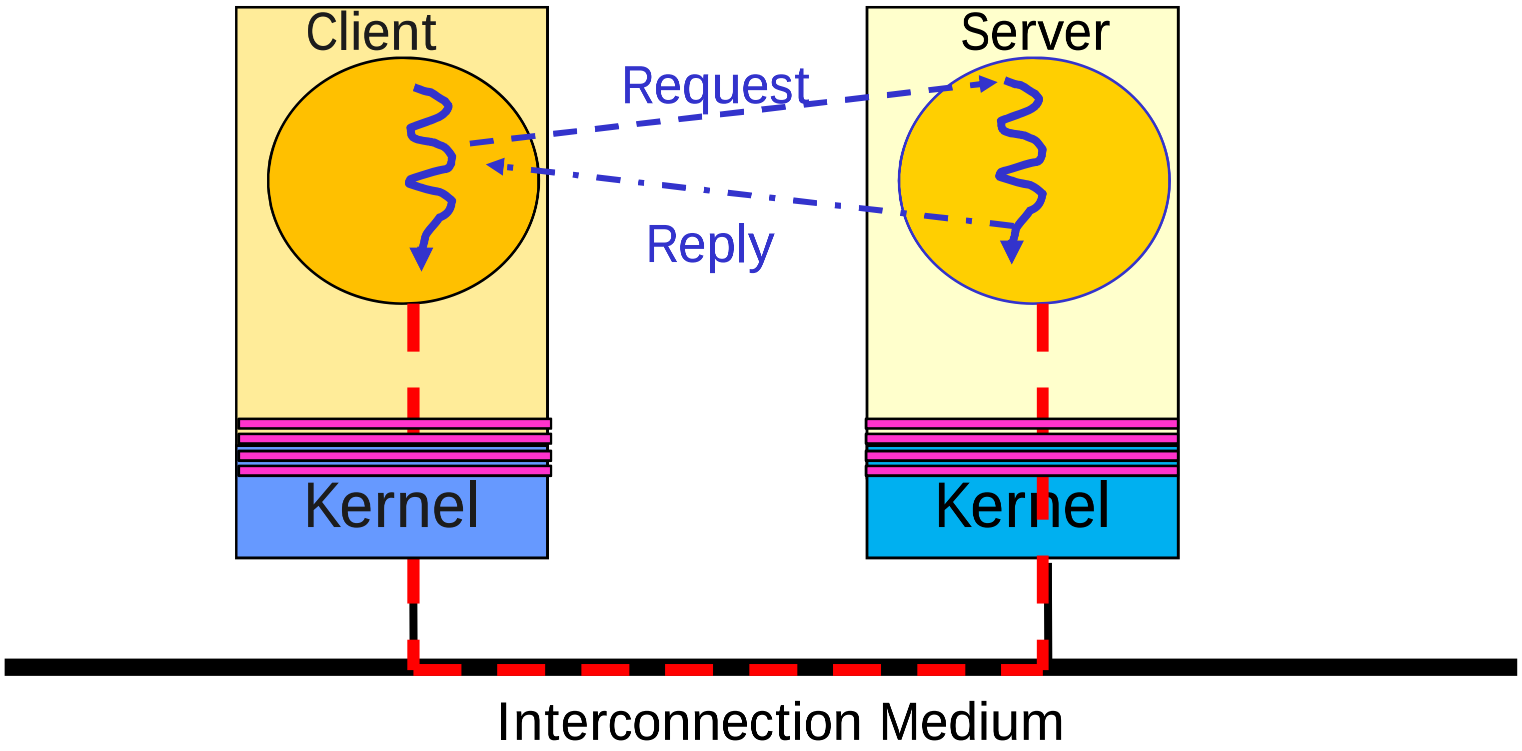


Observation:

(a) Layered style used for client/server systems

(b) Object based style used for distributed object systems

Client/Server Model

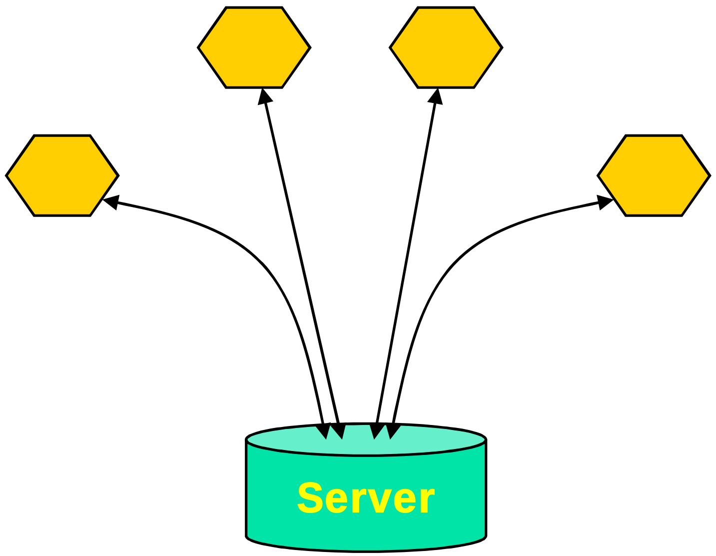


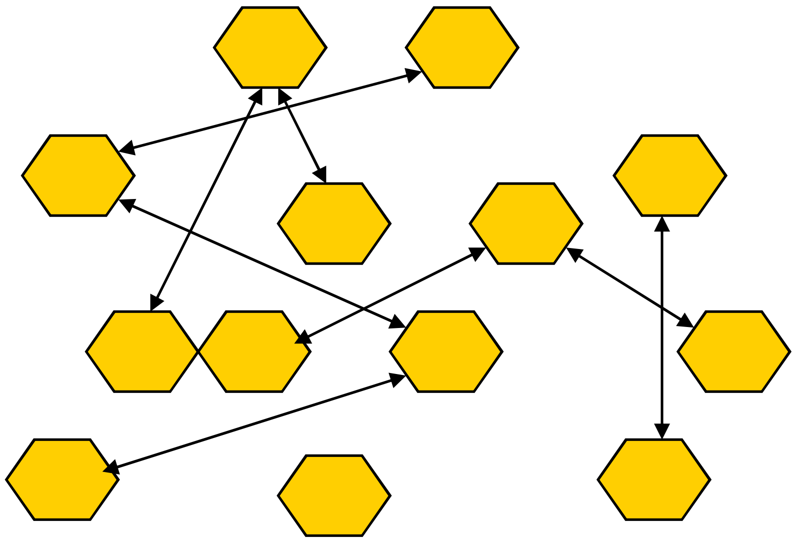
Remark:

Though logically communication is between client and server, the kernels & communication layers of both nodes are involved

P2P Systems

- The term refers to a kind of distributed computing system in which the “main” service is provided by having the client systems talk directly to one-another
- In contrast, traditional systems are structured with servers at the core and clients around the edges

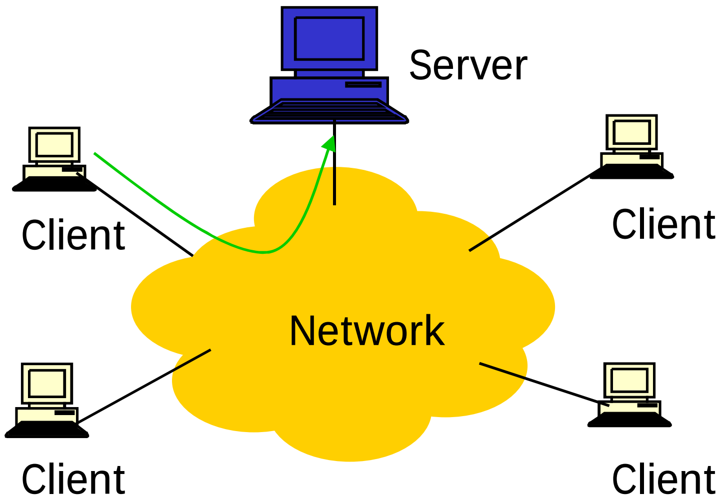




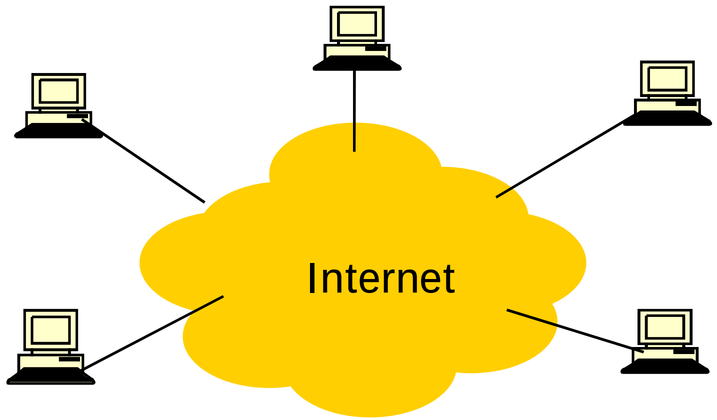
Client/Server versus P2P

- Centralized administration
- Trusted infrastructure
- Server must be prepared to scale with client base
- Server vulnerable to faults and malicious attacks

- Self-organizing
- No required infrastructure beyond connectivity
- Self-scaling (“organic” growth)
- More reliable and fault-tolerant
- *What about availability?*



Wikipedia (see http://en.wikipedia.org/wiki/Client-server)

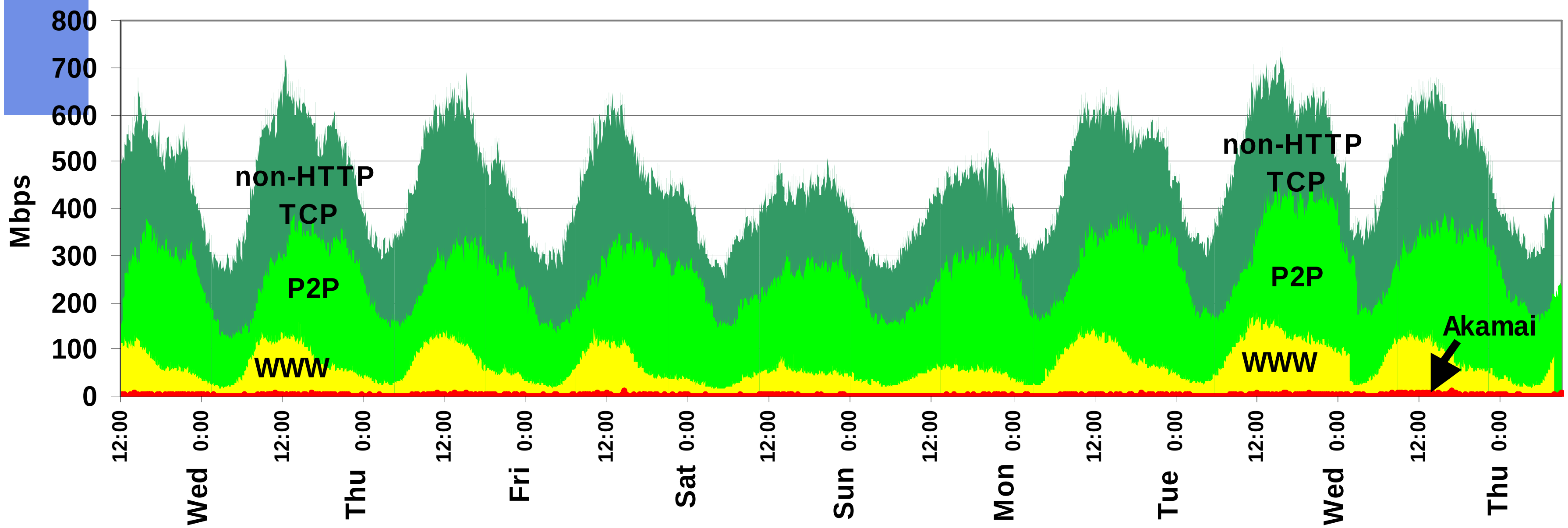


e.g. Gnutella

An important Topic?

- ... or at least, it gets a lot of press
  - Recording industry claims that p2p downloads are killing profits!
    - Used to be mostly file sharing, but now online radio feeds (RSS feeds) are a big deal too
  - University of Washington study showed that 80% of their network bandwidth was spent on music/video downloads!
    - DVDs are largest, and accounted for the lion's share
    - A great many objects were downloaded many times
    - Strangely, many downloads took months to complete...
    - Most went to a tiny handful of machines in dorm rooms

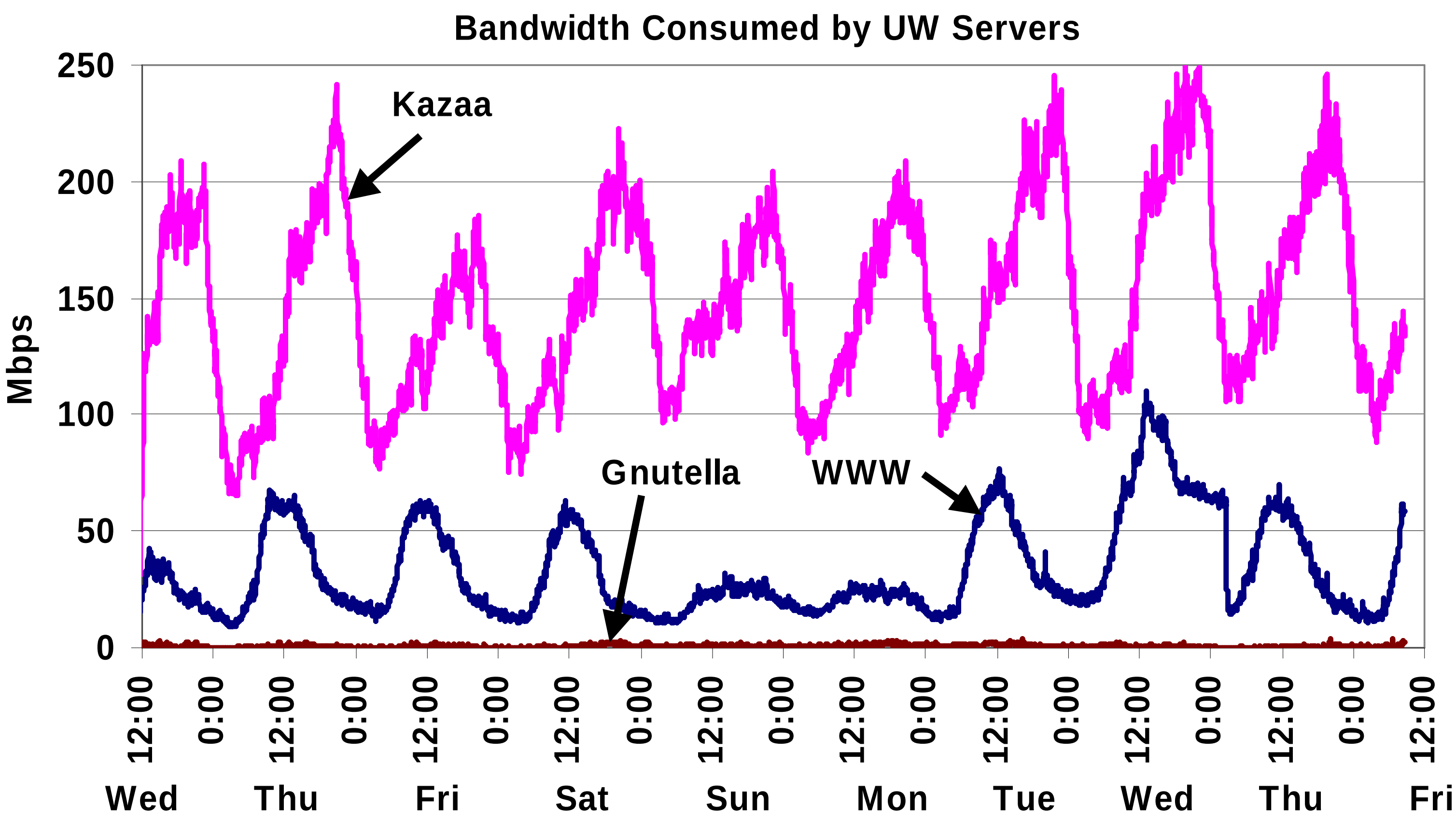
Where has all the Bandwidth gone?



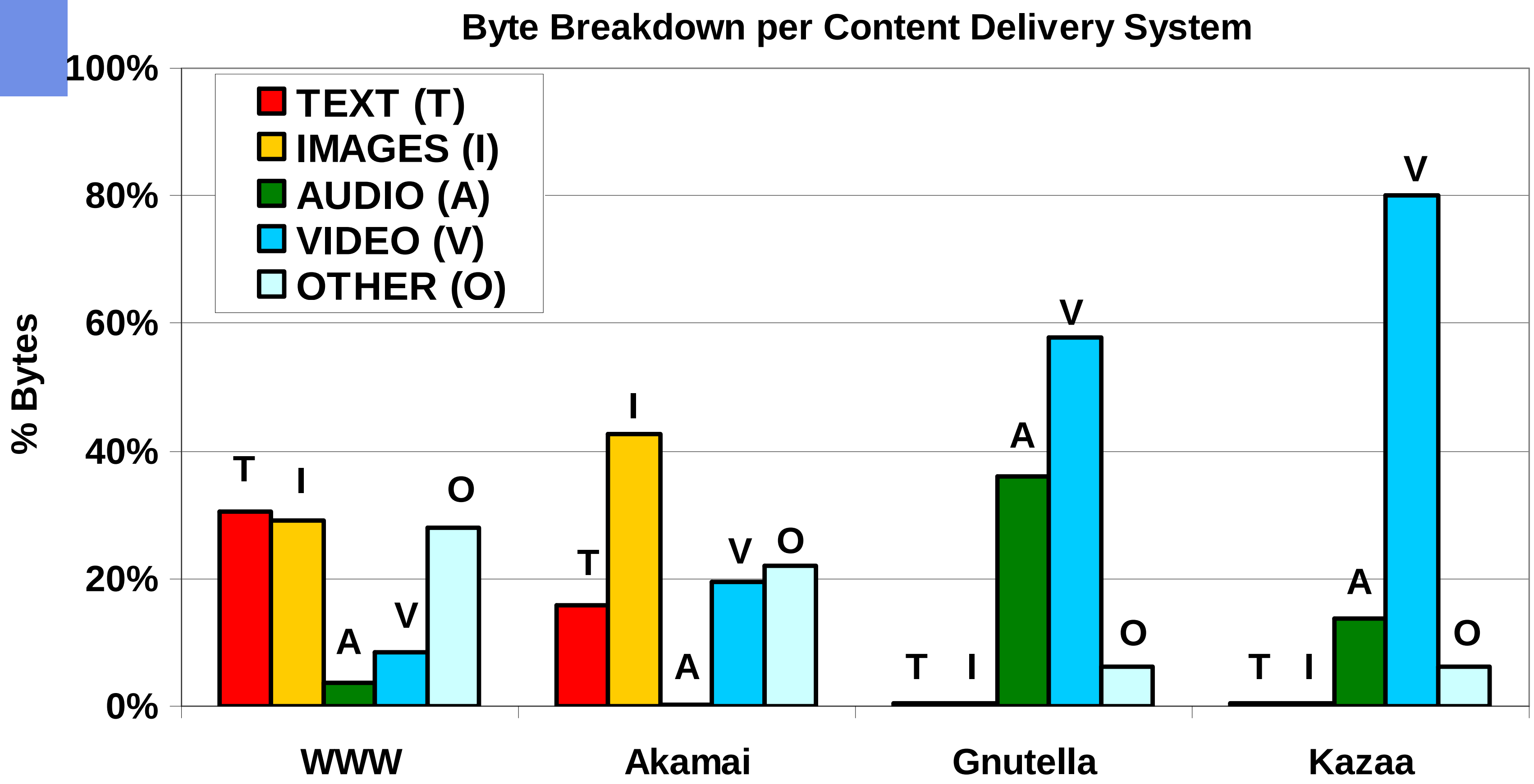
- WWW = 14% of TCP traffic; P2P = 43% of TCP traffic
- **P2P dominates WWW in bandwidth consumed!!**

Source: Hank Levy. See http://www.cs.washington.edu/research/networking/websys/pubs/osdi_2002/osdi.pdf

Bandwidth consumed by UW Servers (outbound traffic)



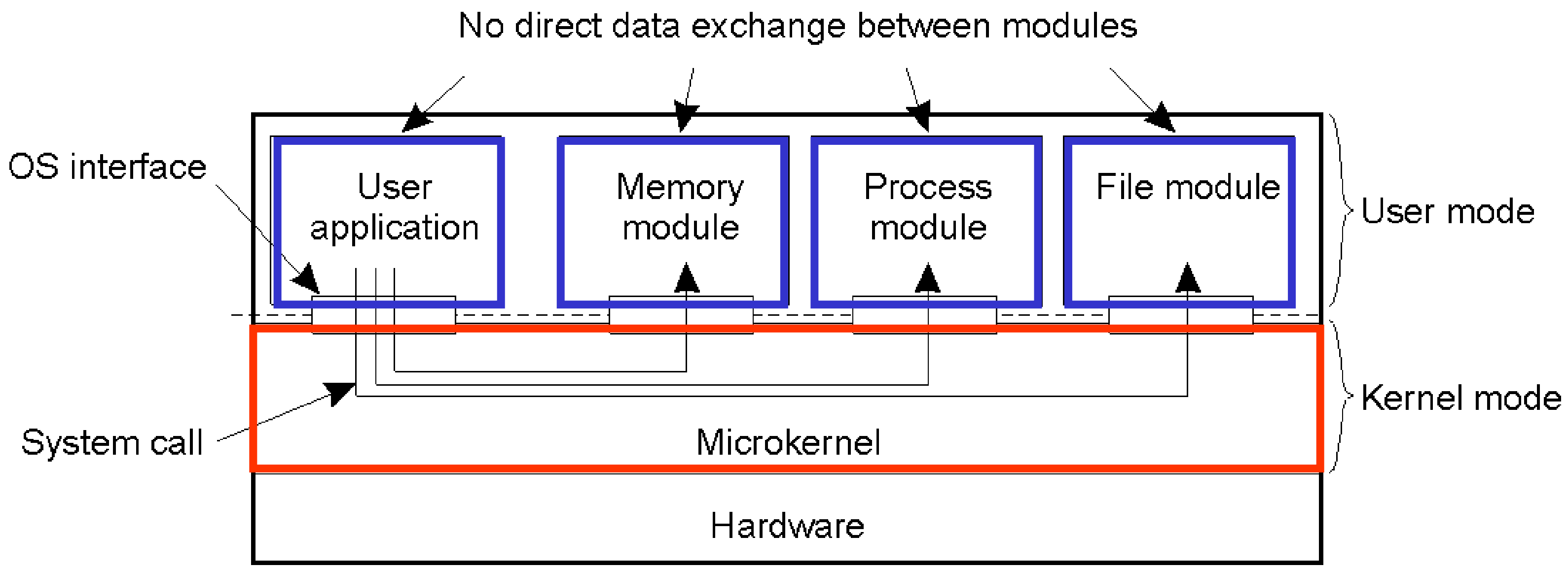
Object Types for Different Systems



Software Architectures

- Layered Systems
- Network OS
- Distributed OS
- Middleware

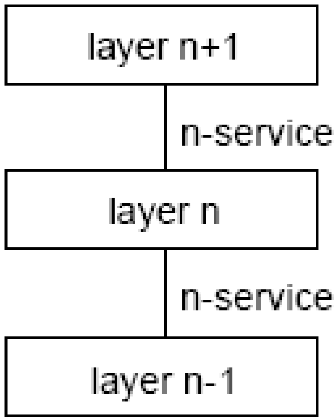
Local System Architecture



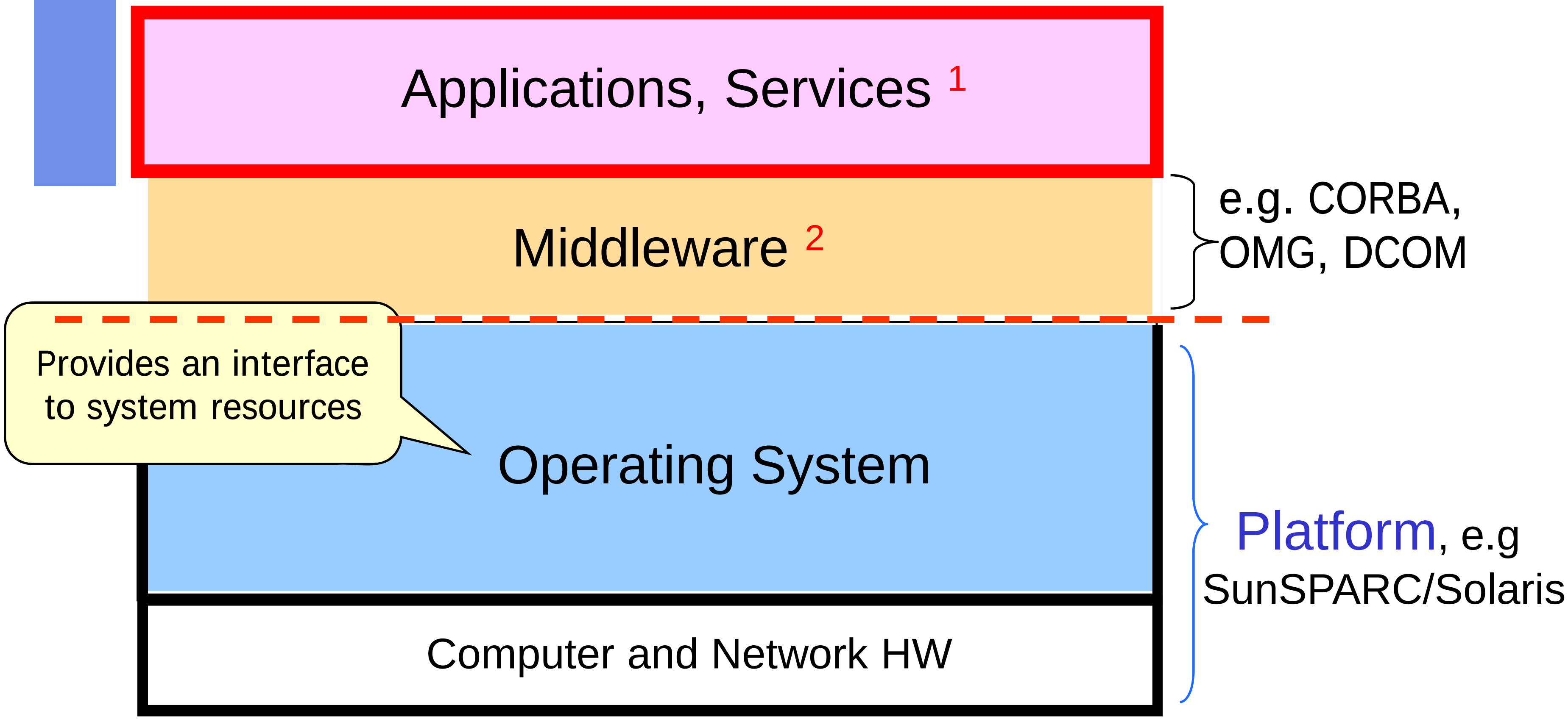
- Applications separated from privileged µkernel
- Clients/servers protected within address spaces
- A µkernel does not imply a flat system architecture,
  ⇒ add software layers, whenever appropriate

Software Layers

- Breaking up the complexity of systems by designing them through layers and services
  - Layer: group of closely related and highly coherent functionalities
  - Service: functionality provided to a superior layer
- Examples of layered software systems:
  - OSes, e.g. kernel & other services
  - Computer network protocol architectures (ISO/OSI)



Typical Layers in DS



[1] Network Time Service via NTP (= Network Time Protocol)

[2] Main task of middleware is

- hiding heterogeneity
- providing an easy and portable programming model

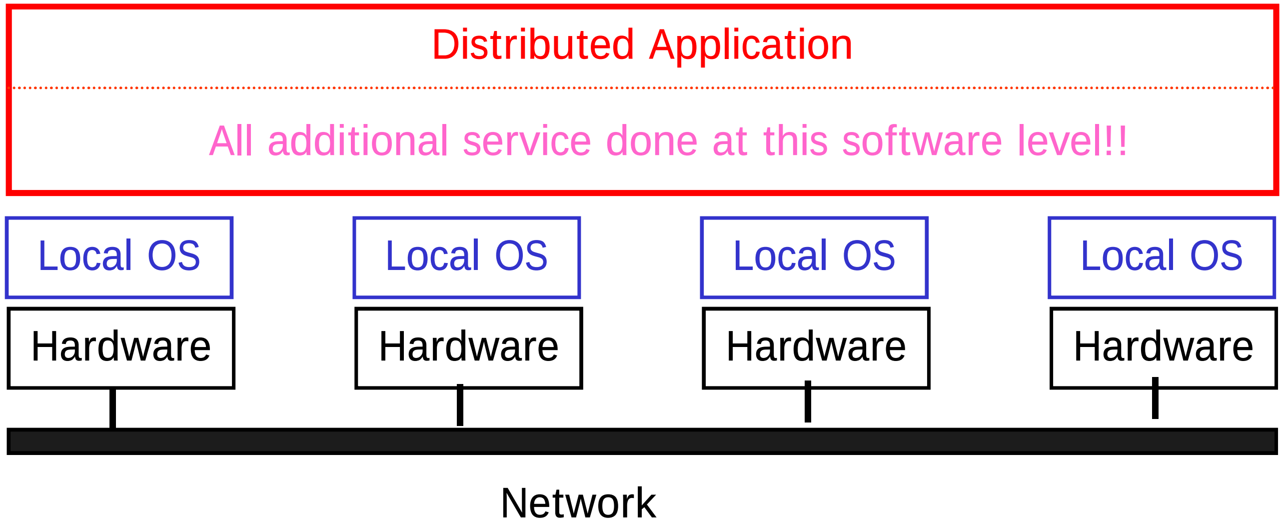
Potential System Support

- Potential support for distributed applications
  - No support
  - Network Operating Systems (NOS)
  - Middleware Systems
  - Distributed Operating Systems (DOS)

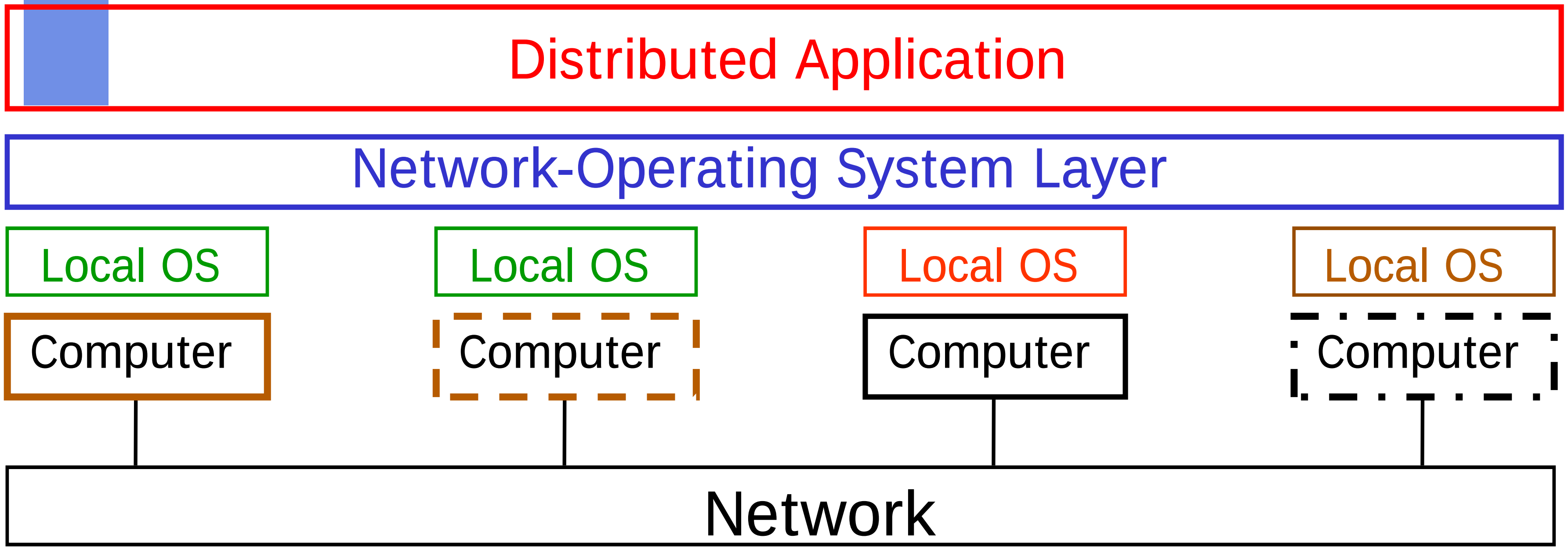
System	Description	Main Goal
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN, MAN, and WAN)	Offer local services to remote clients
Middleware	Additional layer on top of NOS implementing general-purpose services	Provide distribution transparency
DOS	Tightly-coupled operating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

No Application Support

- No local OS supports a distributed application
- Distributed application must handle:
  - Identification of each “remote” application or system component
  - Communication protocols
  - All possible error conditions

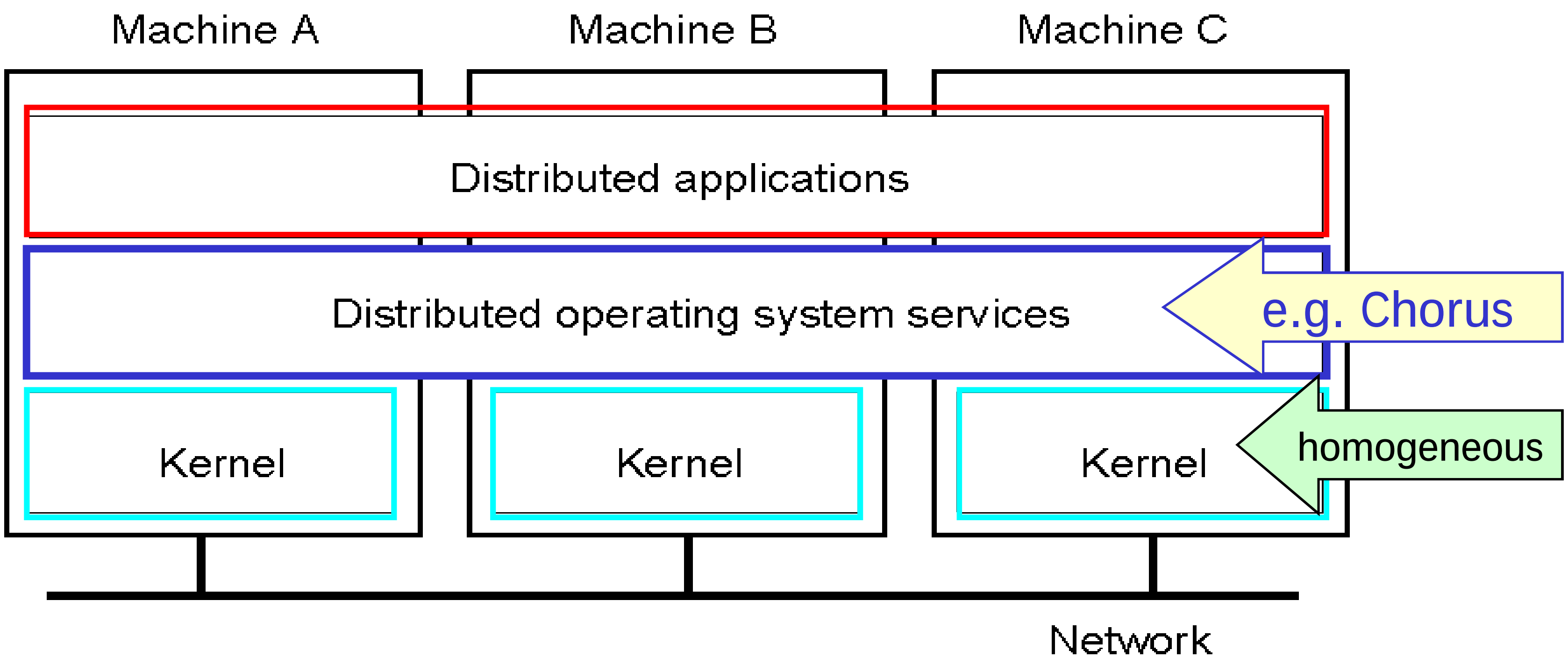


Network Operating System



Design:
You add another software layer on top of all local OSes offering functions needed for the DS, e.g. NFS

Multi-Computer Operating System



- General structure of a multicomputer operating system
- Data structures for OS no longer in a shared main memory, e.g. support for a distributed shared memory
- Each node with a local kernel + inter-node communication

Network System versus DS

- Computer network: the autonomous computers are explicitly visible (have to be addressed explicitly)
- Distributed system: existence of multiple autonomous computers is transparent
- However:
  - Many problems in common
  - In some sense networks (or parts of them, e.g. name services) are also DS, and
  - Normally, every DS relies on services provided by a computer network

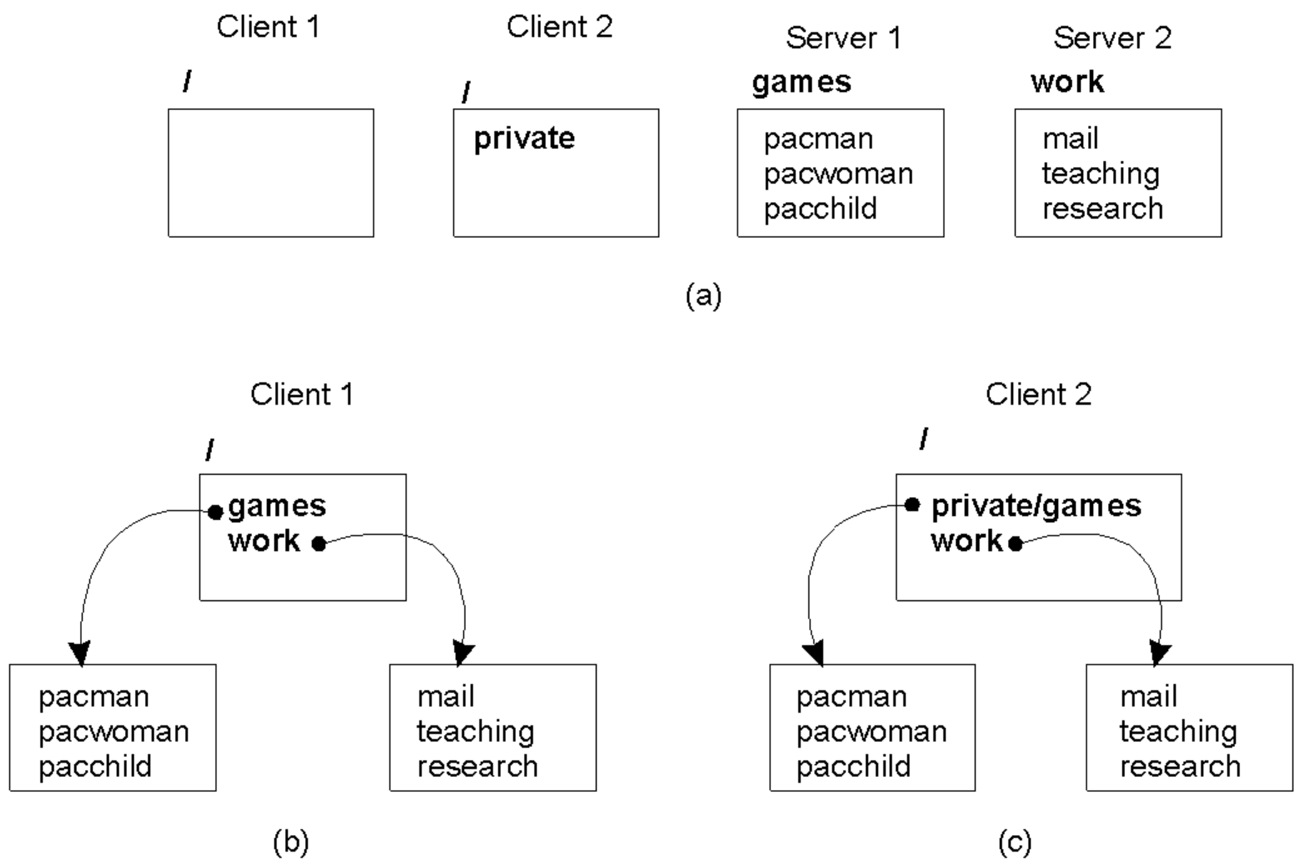
Example 1: Network-OS

Given a LAN of WSs, each user has a WS of its own, all commands run locally.

- he may use **`rlogin`**, i.e. to get a specific service
- his WS tends to be a terminal of the remote machine.
- each user must know where the service is located
- at any instance of time he can only use one remote machine
- a copy service may be installed, e.g.

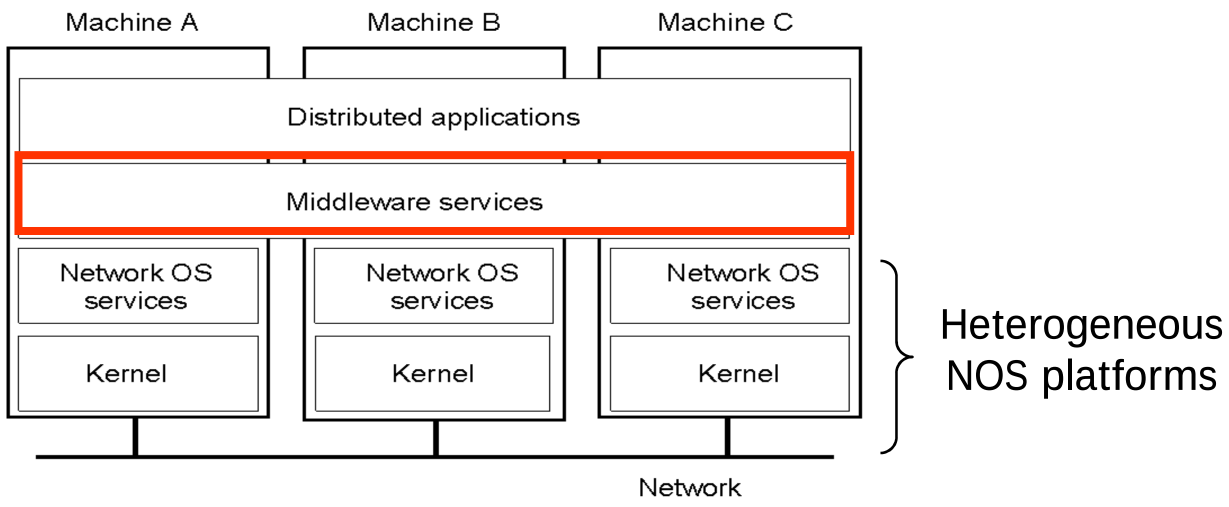
```
rcp machine1:file 1   machine2:file2
```

Example 2: Network-OS



Different clients can have a different view onto the file system

Middleware



- *Functionality of middleware?*
- *Paradigms, the middleware is based upon?*
- Built upon abstractions of commodity OSes
  - process model and
  - message passing
- Middleware runs in user space

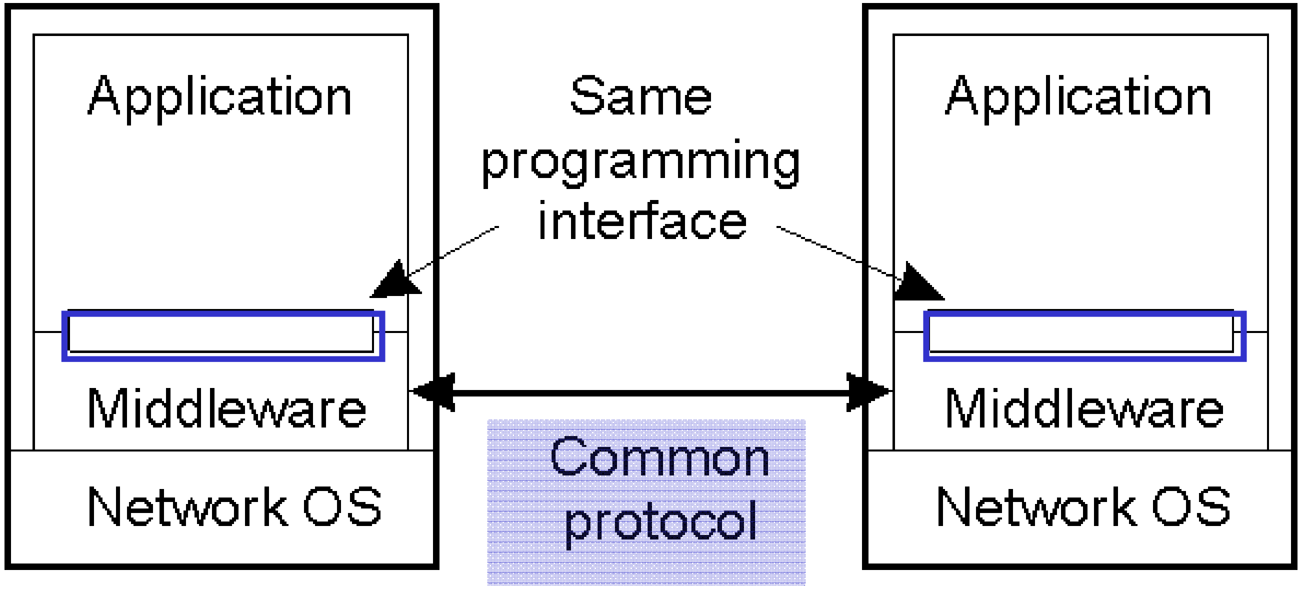
Middleware Services

- High-level communication facilities
  - Access transparency
- Naming
  - Location transparency
  - Scalability
- Persistence
  - Recoverability
- Distributed Transactions
- Security
- Availability

Why will Middleware win?

- Builds on commonly available abstractions of network OSes (tasks, processes, messages)
- Examples: RPC, NFS, CORBA, DCOM, J2EE, .NET
- There ar also languages (or language modifications) designed for distributed computing(e.g. Erlang, Ada, Limbo, etc.)
- Usually runs in user space
- Raises level of programming, i.e. less error-prone
- Independent of OS, network protocol. Programming language, etc., i.e. increased flexibility

Openness & Middleware



- In an open middleware-based DS, protocols used by each middleware layer should be the same, as well as the interfaces they offer to applications $\Rightarrow$
- Improve portability + migration

Characteristics of DS Architectures

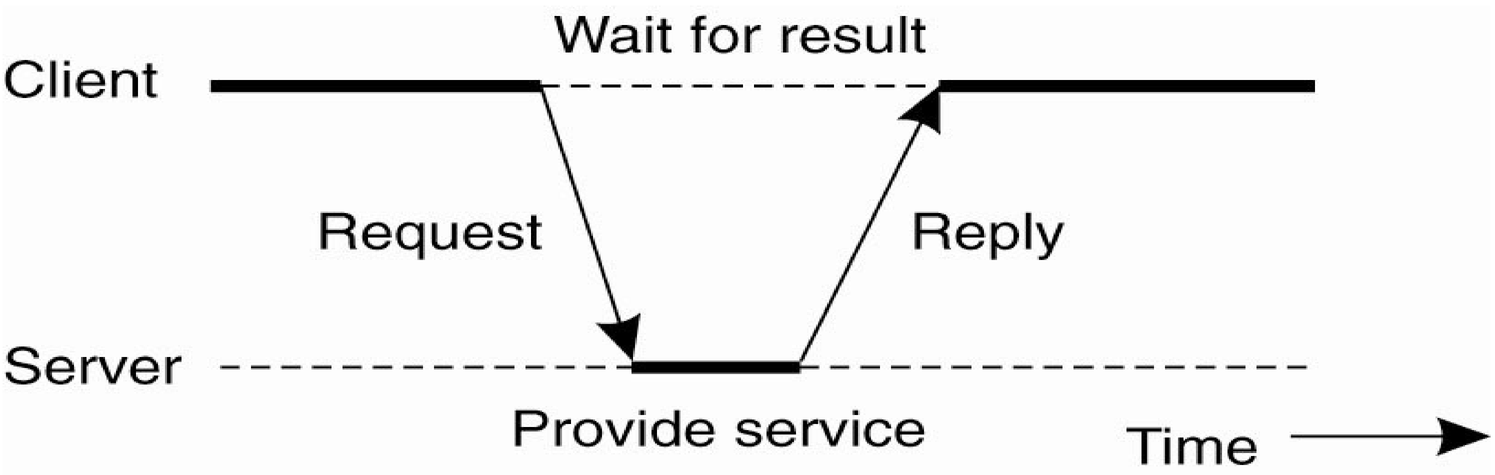
Item	Distributed OS		Middleware	NOS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	High	Low
Same OS on all nodes	Yes	Yes	**No**	**No**
Number of OS copies	**1**	**N**	**N**	**N**
Basis for communication	Shared memory + Messages	Messages	Model specific	Files
Resource management	Global,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

System Architectures

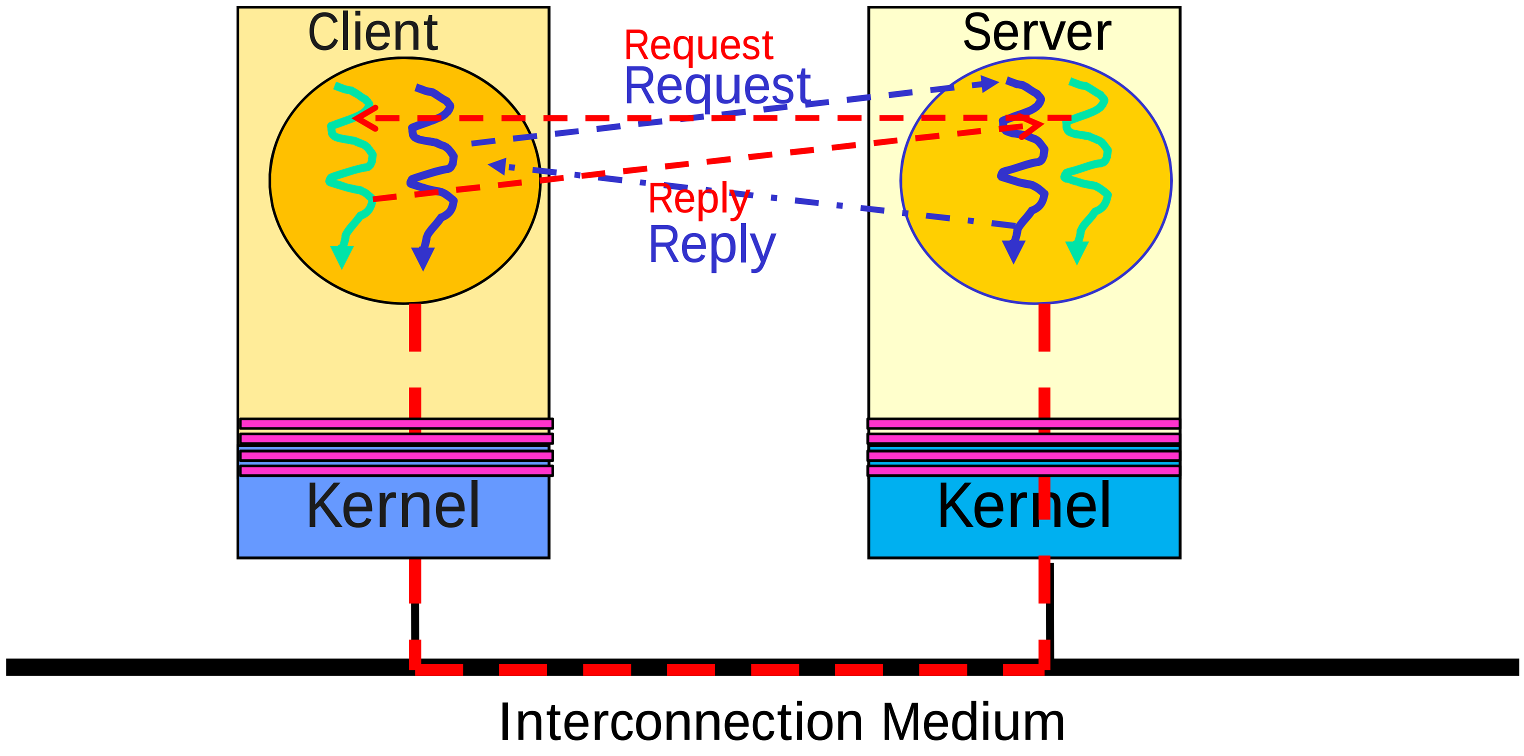
- Centralized SA (Client/Server)
- Decentralized SA (P2P)
- Hybrid SA

Centralized Architectures

- Basic Client/Server Model: Characteristics
  - There are processes/tasks offering services (servers)
  - There are processes/tasks that use services (clients)
  - Clients and servers can be distributed across different nodes
  - Clients follow the usual request/reply interaction model with respect to using services



Client/Server Model



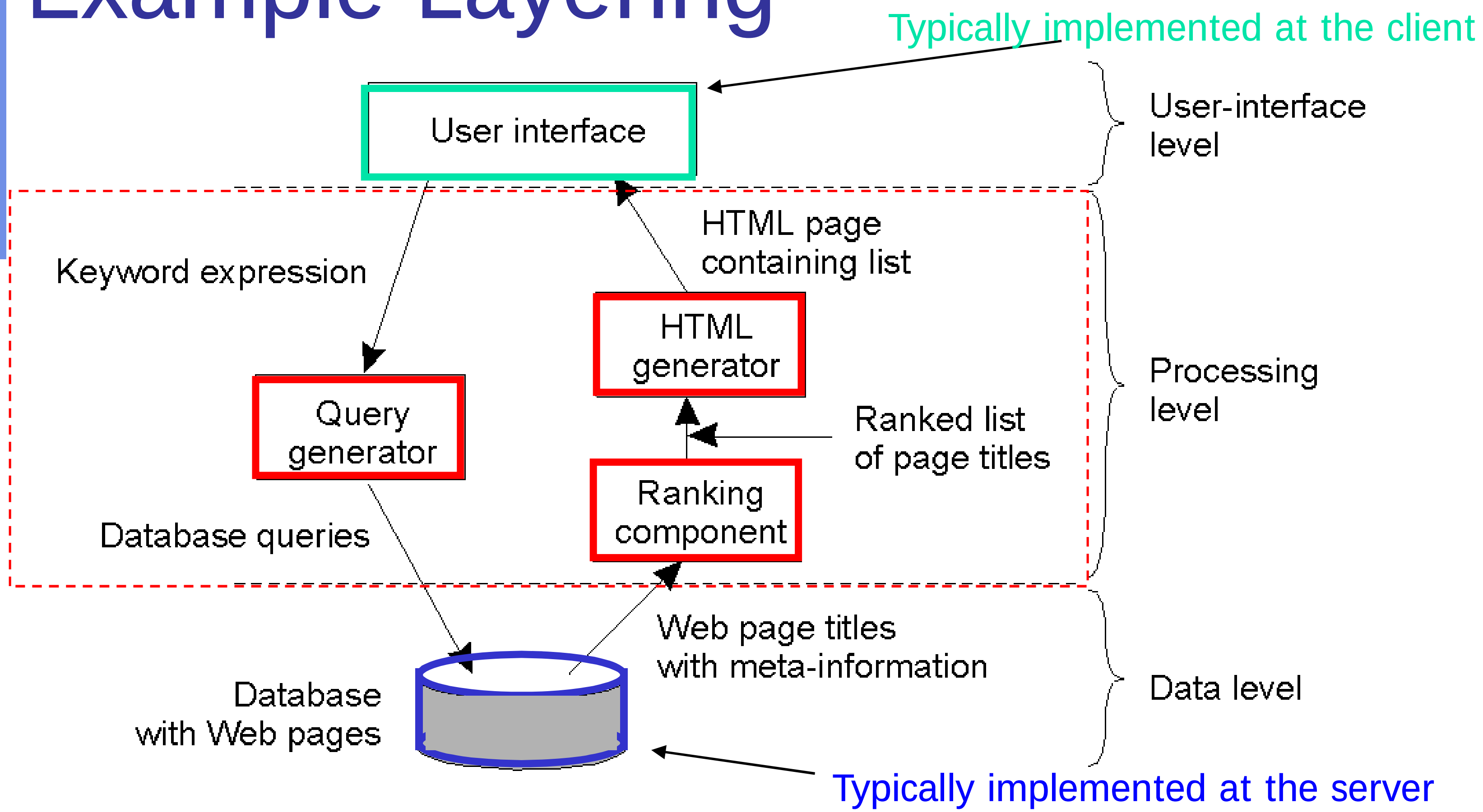
Remark:

Clients & servers imply a hierarchical order (layering)-
Sometimes roles might change

Application Layering (1)

- Recall layers of the general architectural style
- Layering of a DB based client/server model
  - User-interface layer
  - Processing layer
  - Data level
- This layering is found in many DS, using traditional DB techniques and accompanying application
- *Question: Where to implement each layer?*

Example Layering

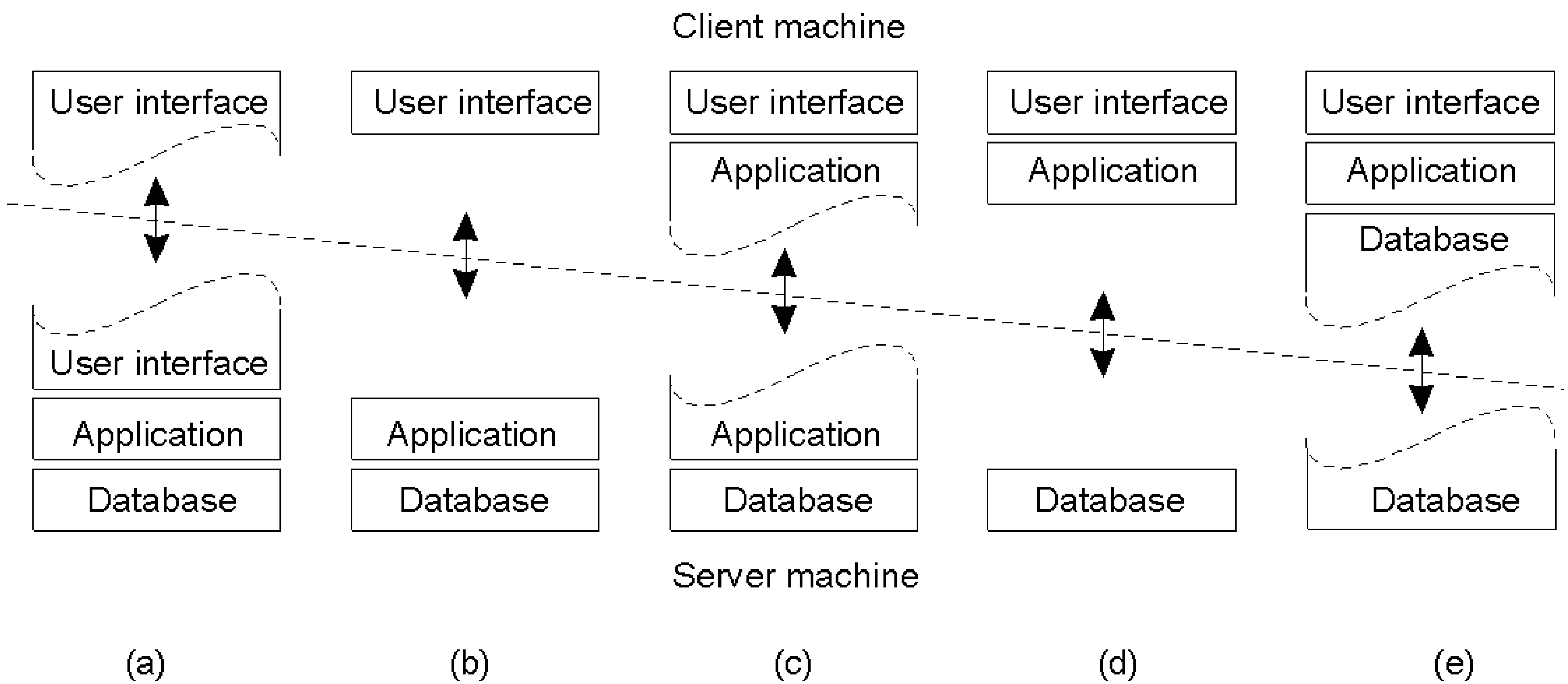


- Organization of an Internet search engine into 3 different layers
- Similar organization: Decision support system for a broker

N-Tiered Architectures

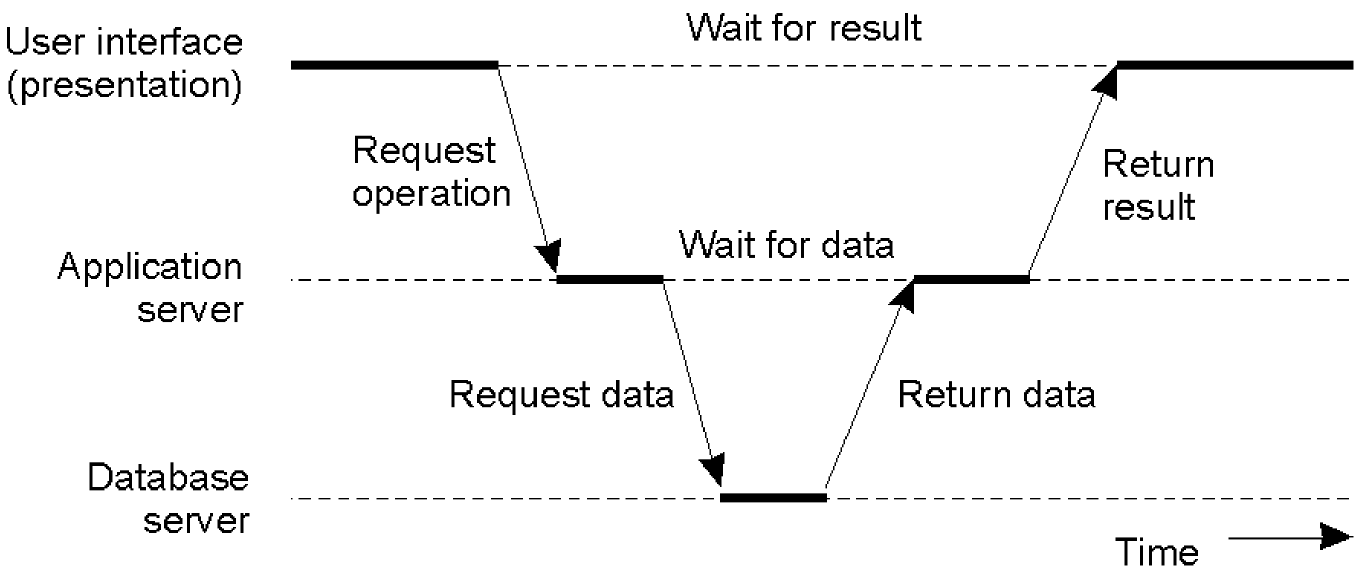
- Single-tiered: old terminal/mainframe configuration
- Two-tiered: classical client/server configuration
  - Client machine contains only the programs implementing (part of) the user-interface level
  - Server machine contains the rest, i.e. programs implementing the processing and data level
- Three-tiered: each layer on a separate node
- …

Traditional Two-Tiered Architectures



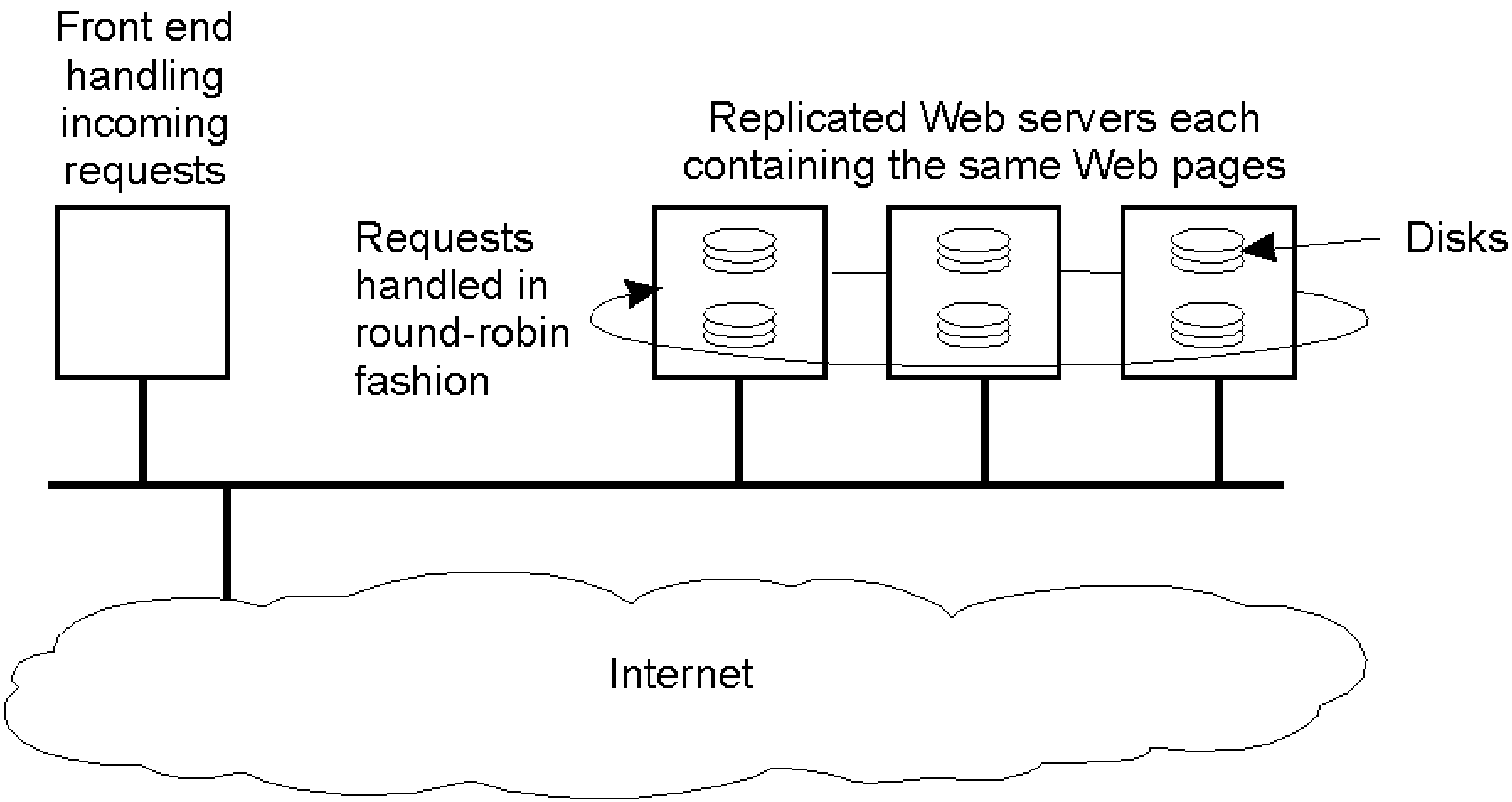
- Alternative client-server organizations (a) – (e).

Three-Tiered Architectures



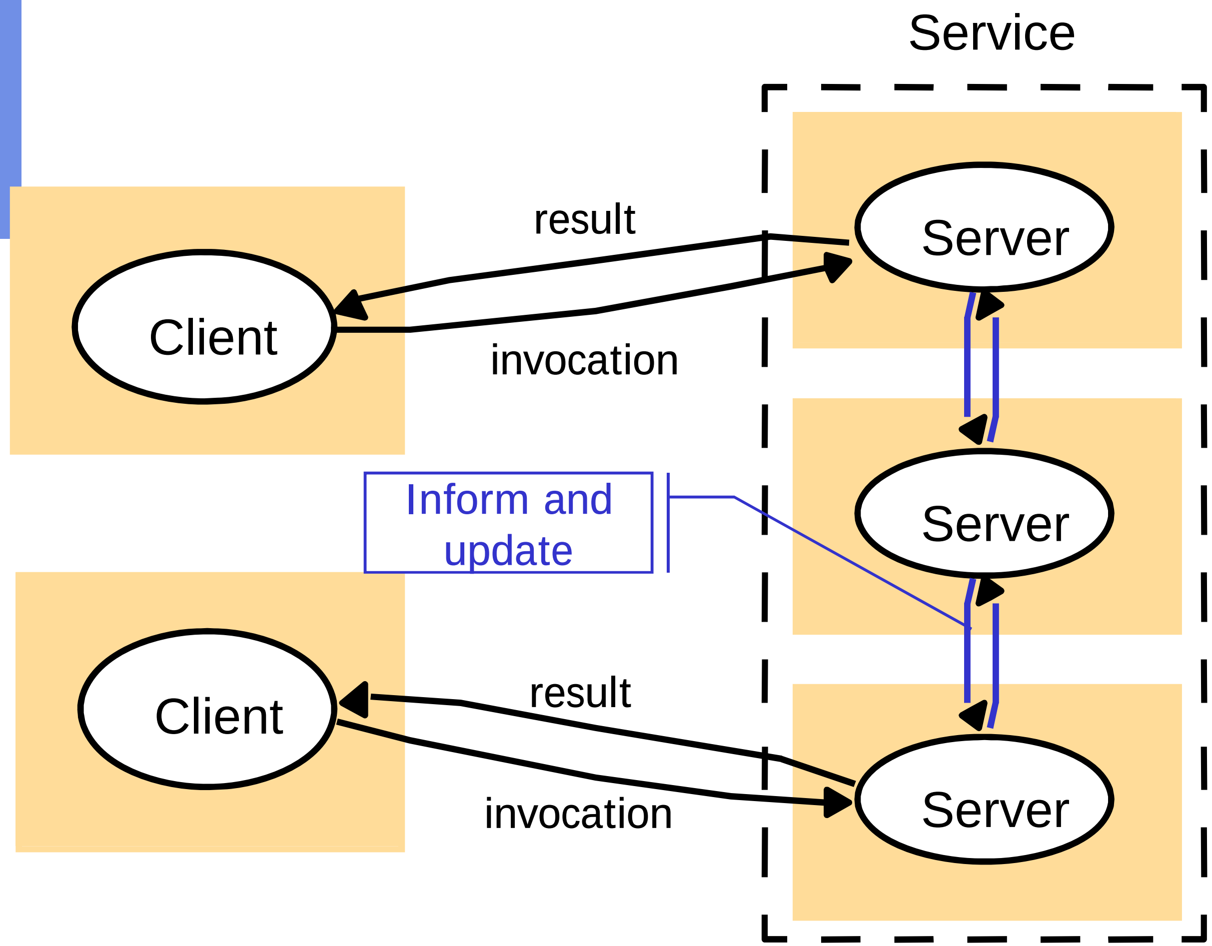
- An example of a server acting as a client in a three-tiered system architecture
- A transaction monitor coordinates all separate transactions that potentially need more than one data base server

Modern Client/Server Architecture



- Example of horizontal distribution of a Web service

Multiple Servers per DS



Partition or replication of server data:

Exam. partition: www

Exam. replication & partition: DNS

Server Architectures

- 1 single threaded server per DS on node n_x
  - \- single point of failure
  - \+ simple solution
- 1 single threaded server per node, but n>1 servers per DS
  - \- maintaining consistency
  - \+ improved availability
- 1 multi-threaded server per DS
  - \- ...
  - \+ ...
- 1 multi-threaded server per node, and n>1 servers per DS
  - \- ...
  - \+ ...
- *Further models?*

Decentralized Architectures

- Structured P2P
  (More details in later lectures)
- Unstructured P2P
- Hybrid

Decentralized Architectures

Observation: There is a trend towards P2P systems

- Structured P2P: nodes are organized following a specific distributed data structure (DHT)
- Unstructured P2P: nodes have randomly selected neighbors
- Hybrid P2P: some nodes are appointed special functions in a well-organized fashion

Not in our focus

Note:

In virtually all cases we are dealing with overlay networks: data is routed over connections setup between the nodes (cf. application-level multicasting)

List of P2P Systems

- Napster MP3 Sharing
  - first hybrid P2P)
  - (not a clean P2P, still a central server,
  - but decentralized reources)
- Gnutella
  - First version an untructured P2P
  - Self organizing, but not that scalable

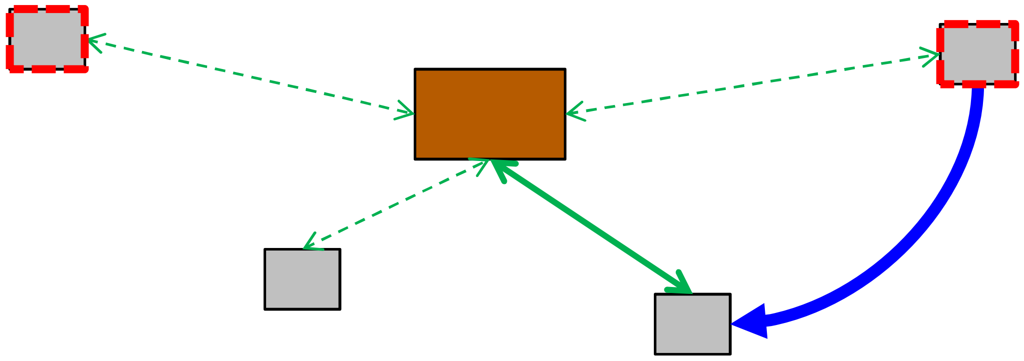
File Sharing

- DHT based P2P
  - Chord (Berkeley, MIT)
  - CAN (Berkeley, ICSI/ICIR)
  - Pastry (Rice, Microsoft)
  - Freenet
- JXTA

more details in a future lecture

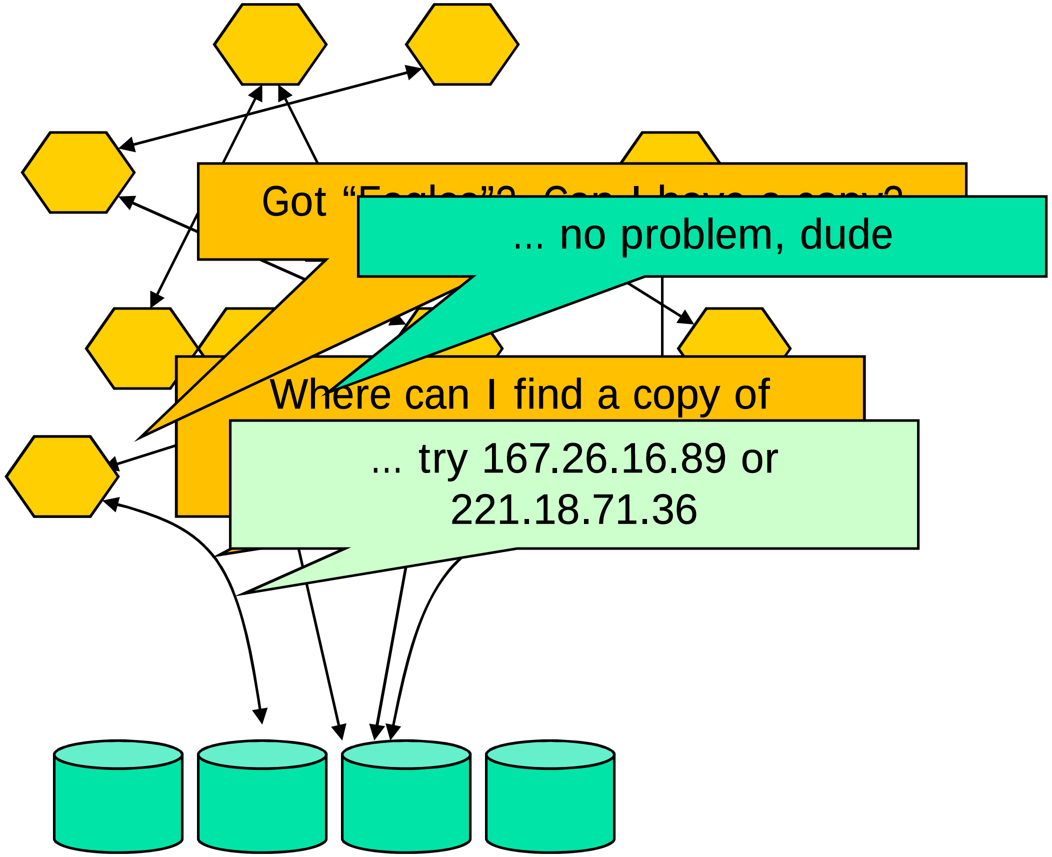
Napster

- First P2P killer application (1999-2001)
- Illegal exchange of MP3 music files
- Centralized Directory Servers (centralized index)
  - Administration of node addresses and files at involved peers
  - Lookup via central servers
  - Servers build the web pages clients see
  - MP3 files are distributed amongst the peers
  - Actual MP3 or DVD downloads are done from client to client



Napster

Having obtained a top-level page listing peers with copies of music or other content desired, a client can download the files directly from the peer



Data center builds the pages users see when they access Napster

Napster Extensions

- OpenNap-network
  - Multiple statically networked directory server
    - Improved reliability and availability
    - No single point of failure anymore
  - Support for any file format
- Characteristics
  - Scalability is limited by centralized directory servers
  - Not a pure P2P system
- Analysis (April 2001)
  - OpenNap ~ 80 directory server
  - ~ 50 000 users online
  - More than 10 000 000 files
  - More than 55 TB data

Why did Napster go this way?

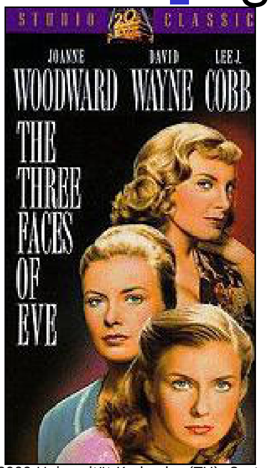
- When service launched, developers hoped to work around legal limits on sharing media
  - They reasoned: let client systems advertise “stuff”
  - If some of that stuff happens to be music, that’s the responsibility of the person who does it
  - The directory system “helps clients advertise wares” but doesn’t “endorse” the sharing of protected intellectual property. Client who chooses to do so is violating the law
  - They make their money on advertising they insert
- Judges saw it differently...
  - “Napster’s clear purpose is to facilitate theft of intellectual property...”

Technical Issues with Napster

- Many clients just aren’t accessible or if accessible only for a very short time
  - Firewalls can limit incoming connections to clients
  - Many client systems come and go (churn)
  - Round trip times to Nepal are slow...
- Clients might withdraw a file unexpectedly
  - E.g. if low on disk space, or if they download something on top of a song they aren’t listening to anymore
- Industry has attacked the service... and not just in court of law
  - Denial of service assaults on core servers
  - Some clients lie about content (e.g. serve Frank Sinatra in response to download for Eminem)
  - Hacking Napster “clients” to run the protocol in various broken (disruptive) ways
  - And trying to figure out who is serving which files, in order to sue those people

Fundamental Problems?

- If we assume clients serve up the same stuff people download, the number of sources for a <u>less popular item</u> will be very small
- Under assumption that churn is a constant, these less popular items will generally not be accessible.
- But experiments show that clients fall into two categories:
  - Well-connected clients that hang around
  - Poorly-connected clients that also churn
  - ... this confuses the question
- One can have, some claim, as many electronic personas as one has the time and energy to create. *– Judith S. Donath.*
- So-called “Sybil attack....”
  - Attacker buys a high performance computer cluster
  - It registers many times with Napster using a variety of IP addresses (maybe 10’s of thousands of times)
  - Thinking these are real, Napster lists them in download pages. Real clients get poor service or even get snared
  - Studies show that no p2p system can easily defend against Sybil attacks!



Refined Napster

- Early Napster just listed anything. Later:
  - Enhanced directory servers to probe clients, track their health. Uses an automated reporting of download problems to trim “bad sources” from list
  - Ranks data sources to preferentially list clients who…
    - Have been up for a long time, and
    - Seem to have fast connections, and
    - Appear to be “close” to the client doing the download (uses notion of “Internet distance”)
  - Implement parallel downloads and even an experimental method for doing “striped” downloads (first block from source A, second from source B, third from C, etc)
    - Leverages asymmetric download/uplink speeds

Meanwhile, P2P took off

- By the time Napster was ruled illegal, it had 15 million users. 5 million of them joined in just a few months!

- With Napster out of business, a vacuum arose
  - Some users teamed up to define an open standard called “Gnutella” and to develop many protocol implementations
  - Gnutella eliminates the servers
    - Judge singled it out in deciding that Napster was illegal
    - Also, a true peer-to-peer network seems harder to defeat than one that is only partly peer-to-peer
    - Credo: “All information should be free”

Unstructured P2P Architectures[1]

Unstructured P2P systems maintain a random graph

Basic principle: Each node is required to be able to contact a randomly selected other node:

- Let each peer maintain a partial view of the network, consisting of c other nodes
- Each node P periodically selects a node Q from its partial view
- P and Q exchange information and exchange members of their respective partial views

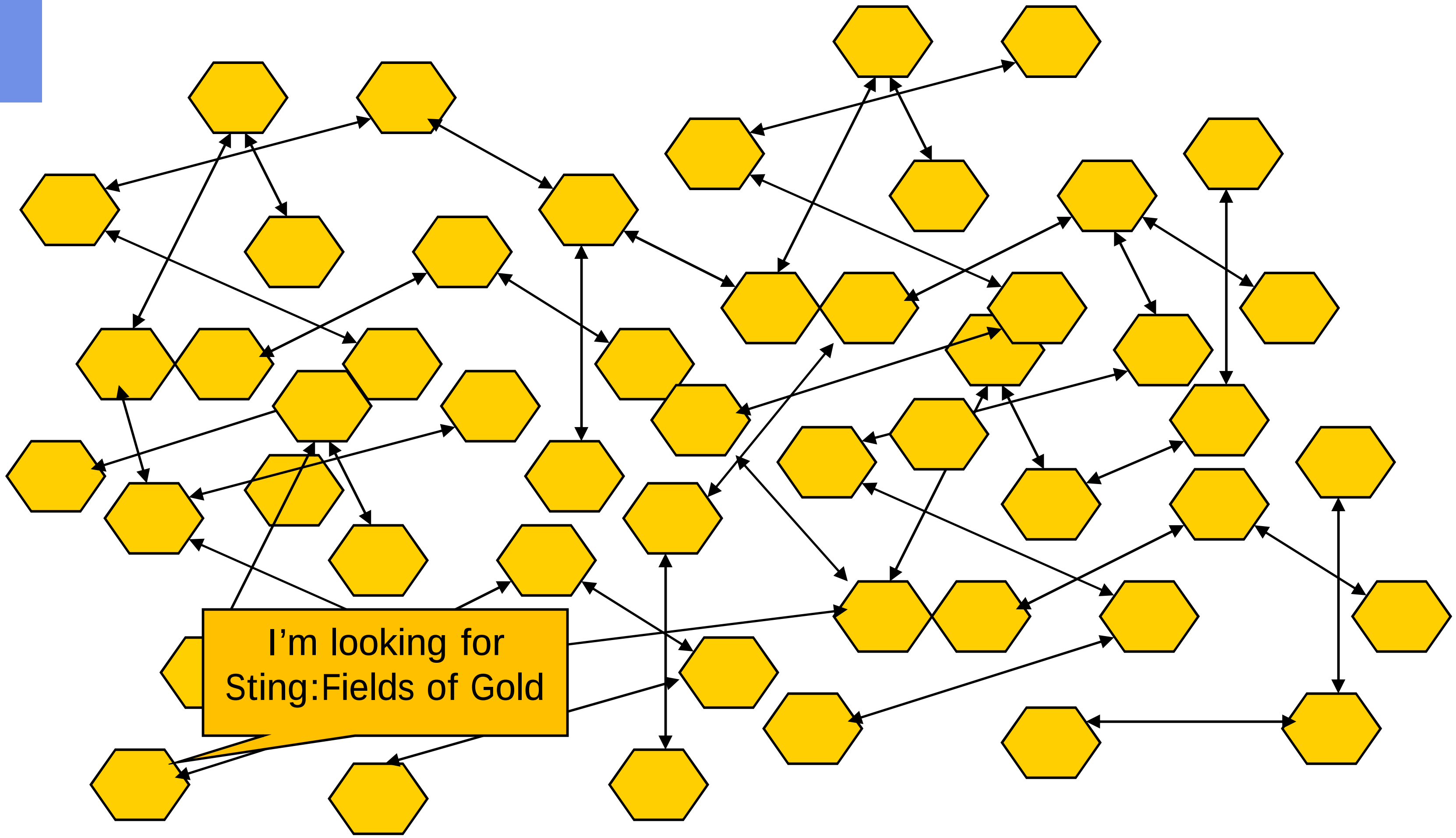
Observation: It turns out that –depending on the exchange protocol- randomness, but also robustness of the network can be maintained

[1]Unstructured P2P not in our focus

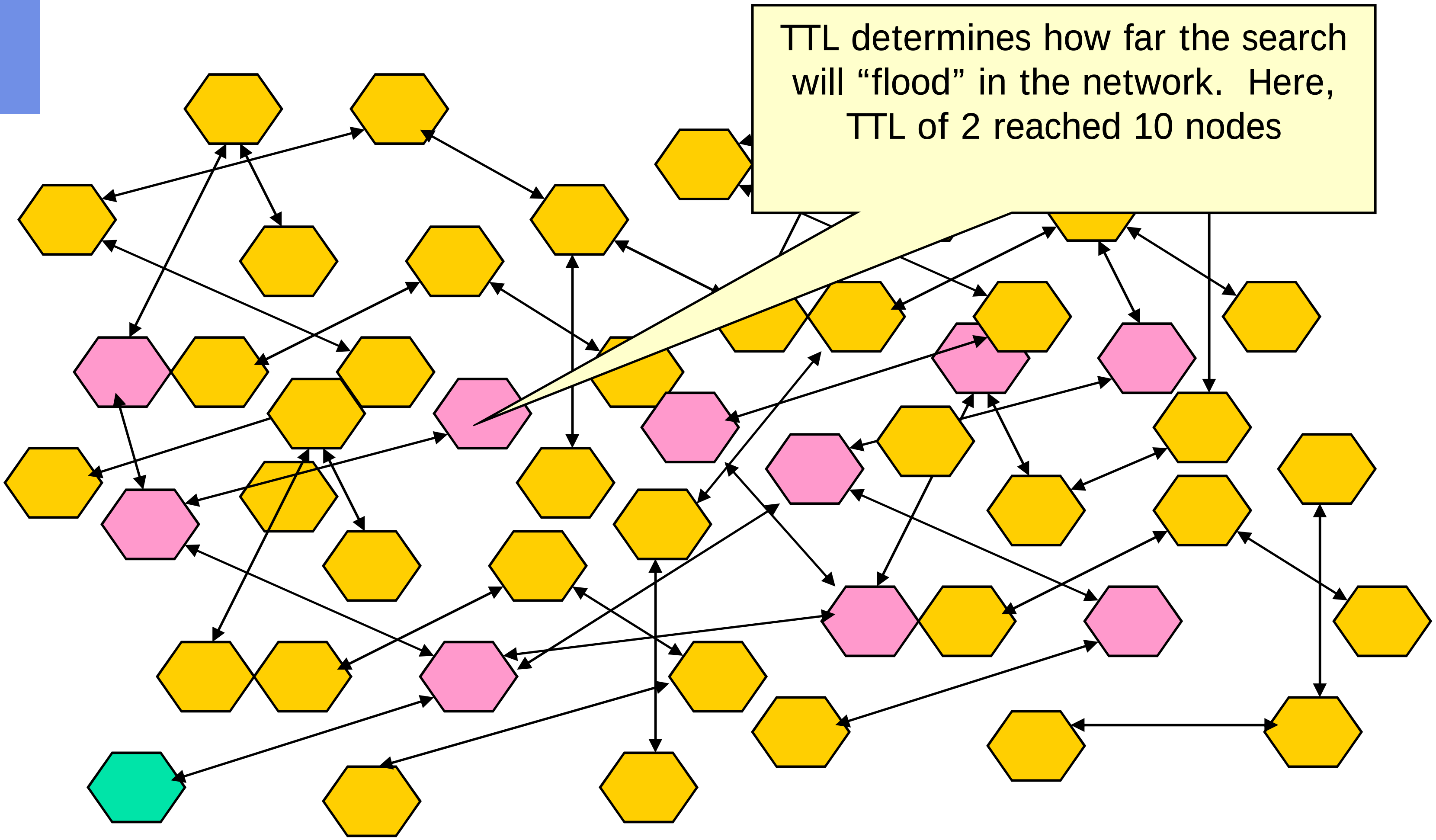
Gnutella Fundamentals

- User joins the network using a broadcast with increasing TTL values
  - *"Is anyone out there?"*
  - Links itself to the first Gnutella node to respond
- To find content, protocol searches in a similar way
  - Broadcasts "I'm looking for Eminem:WhackHer"
  - Keeps increasing TTL value... eventually gives up if no system respond
  - Hopefully, popular content will turn up nearby

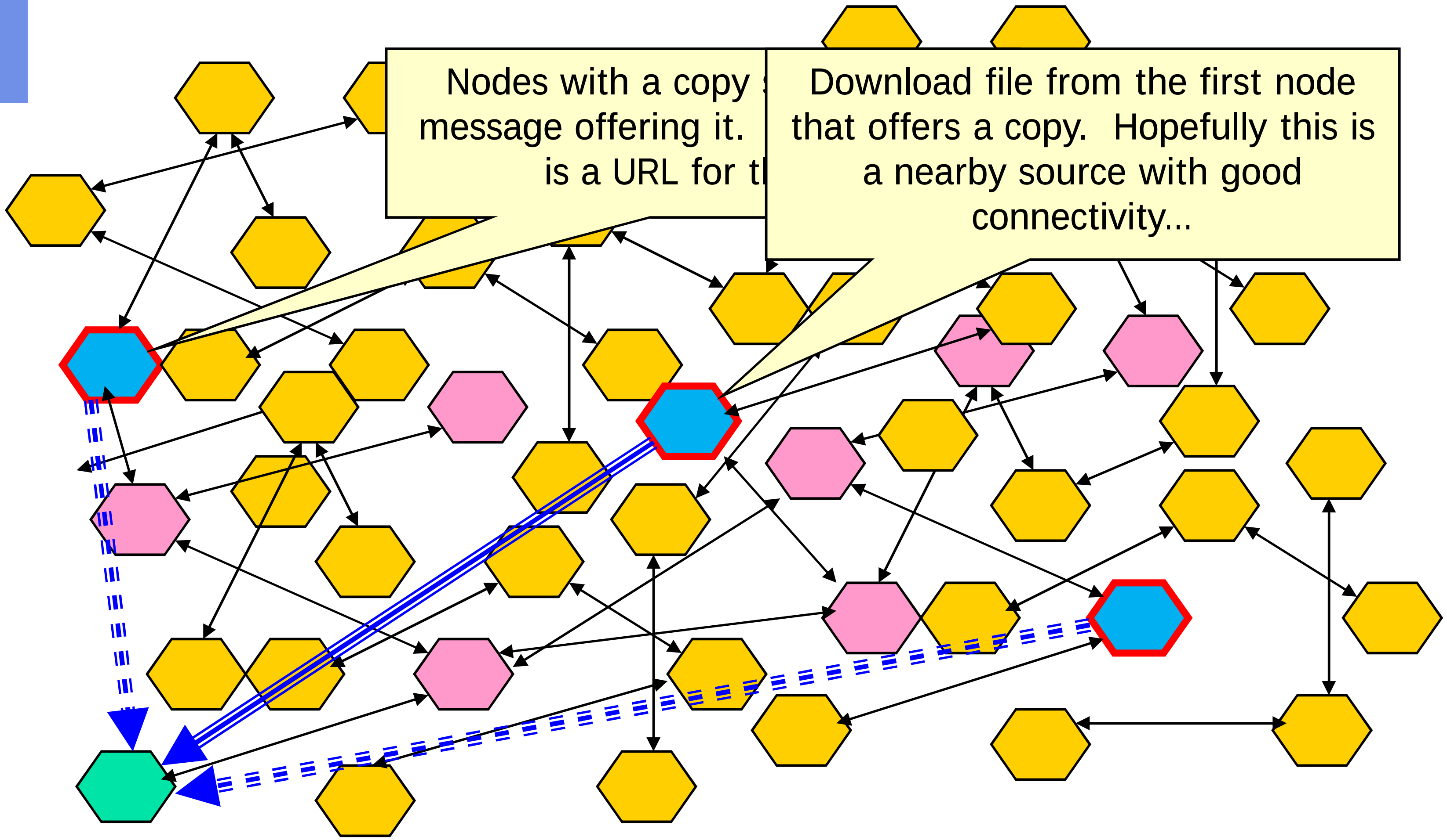
“Self-Organized“ Overlay Network



Search in Overlay-Network



Download in Gnutella



Gnutellas Main Issues

- In experimental studies of the system
  - Very high rates of join requests and queries are sometimes observed
  - Departures (churn) found to disrupt the Gnutella communication graph
  - Requests for rare or misspelled content turn into world-wide broadcasts
    - Rare is... um... rare. Misspellings are common.

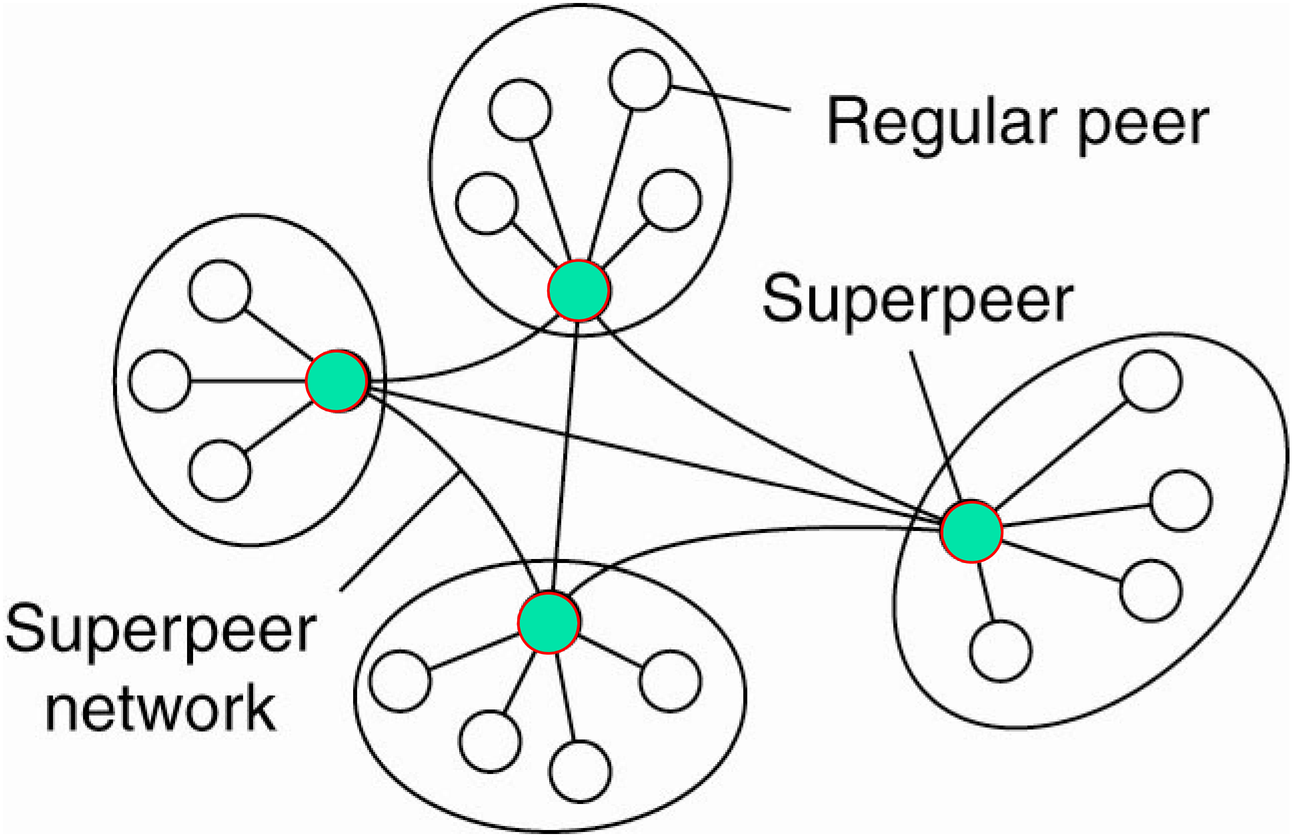
Gnutella Protocol

- Peers are connected via TCP links
- Queries are flooded via the Gnutella network
  - TCP broadcast of ping and query messages
- Identify routing loops via pseudo-unique message IDs (UUID)
  - UUID has 128 bits, containing a timestamp, pseudo-number and the MAC address
  - Double UUIDs are possible, but not very probable
  - Temporary buffering of UUIDs of already received messages
  - Skip double messages
- Performance breakdown in August 2000 because many low budget nodes have been overloaded
- Next Generation Gnutella with super peers

Super Peers

- Unstructured P2P tend to become less scalable due to their indeterminism
- Often flooding the complete net is the only possibility
- Super peers, i.e. specific management nodes maintain an index of all data items
- Super pees can also be used in Content Delivery Networks (CDN), where each regular peer offers resources (e.g. storage for hosting web pages)
  - Super peer (~broker) can find an appropriate candidate having enough capacity to store more web pages

Super-Peers



- A hierarchical organization of nodes into a super-peer network (compare to clan/chief model)

Structured P2P

Research: Structured P2P Systems

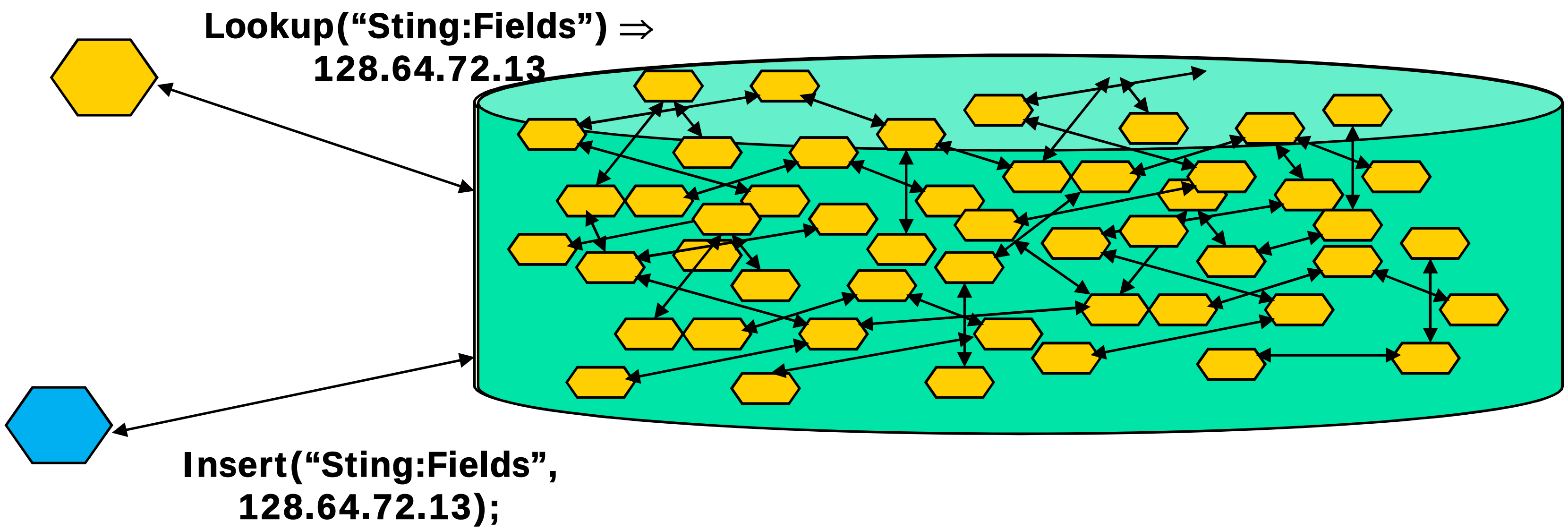
- Universities were first to view P2P as an interesting research area
  - MIT Chord: “distributed hash table” DHT
  - Berkeley
    - CAN: “Content addressable network”
    - Tapestry (similar to Pastry)
  - Rice Pastry, Chord alike protocol
  - Cornell Kelips and Beehive (using replication)
- All systems separate the “indexing” problem from actual storage of the data objects

Distributed Hash Tables (DHT)

- Idea is to support a simple index with API:
  - Insert(key, value) – saves (key,value) tuple
  - Lookup(key) – looks up key and returns value
- Implement it in a P2P network, not a server...
  - Exactly how we implement it varies
  - Normally, each P2P client has only a part of all the tuples, i.e. it must route a query to the right place

Goal: Avoid flooding of the P2P system to find the location of the desired object, file, etc. ...

Distributed Indexing



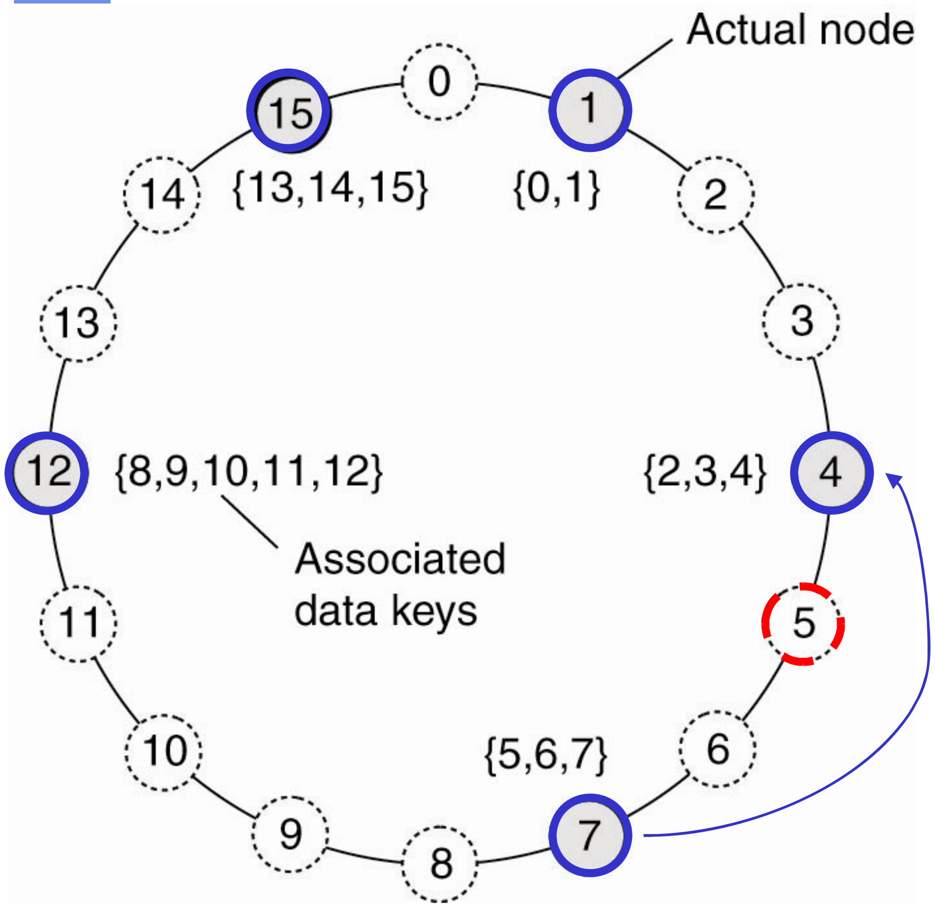
Some Details

- Keep in mind:
  - There are lots of protocols that can solve this problem: the protocol used is not part of the problem statement
  - Some DHTs allow updates (e.g. if data moves, or nodes crash). Others are write once.
  - Most DHTs allow many tuples with the same key and can return the whole list, or a random subset of size k, etc

What should we insert in a DHT?

- Normally, we want to keep the values small... like an IP address
  - So the (key,value) pairs might tell us where to look for something but probably not the actual thing
  - Value could be (and often is) a URL
- Once we have the DHT running we can use it to build a P2P file system

Structured P2P Example: Chord



Initially, the data item with key 5 is on node 7, cause 7 is the largest id with id ≥ 5

A new node first gets its logical ID, e.g. 5

Then it does a lookup(ID=5) ⇒ network address of succ(5) = 7
Contact node 7 and get its predecessor, i.e. network address of 4

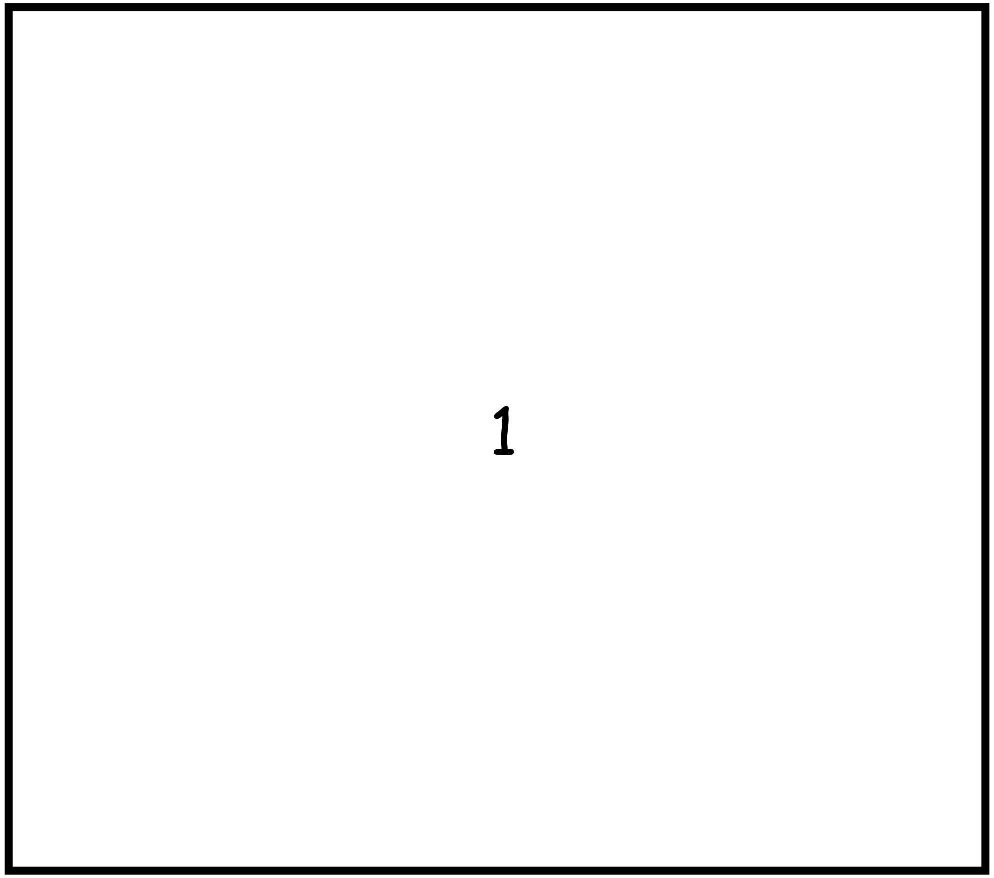
Copy all data items with key 5 from 7 to 5

- Mapping of data items onto logical nodes in Chord
- Each data item has a key, each node has its logical id
- Both are "randomly hashed" (e.g. 120 or 160 bit long)

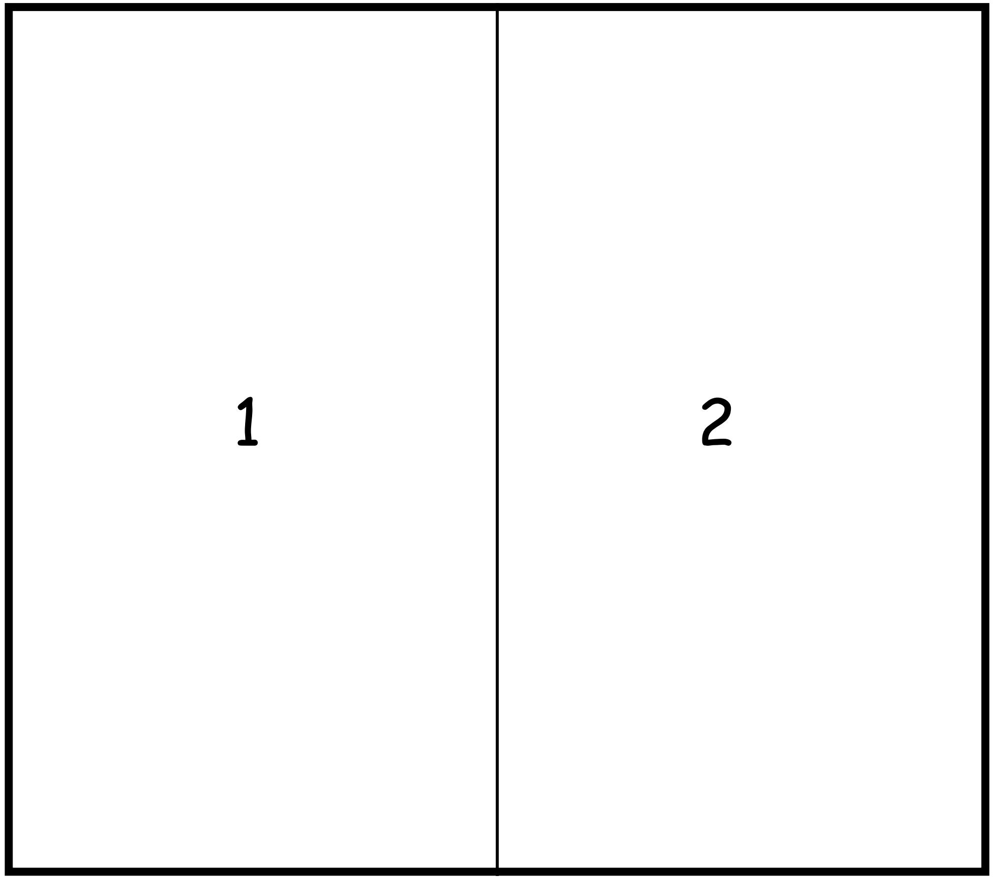
Content Addressable Network (CAN)

- CAN deploys a d-dimensional Cartesian Coordinate Space (ddCCS)
- Each node has a unique key = a point in the ddCCS and an associated region
- Each data item has a key that belongs to one of the regions
- Ratnasamy, S. et al.: "A Scalable Content-Adressable Network", Proc. SIGCOMM ACM, 2001"
(slides in additional literature on our course site)

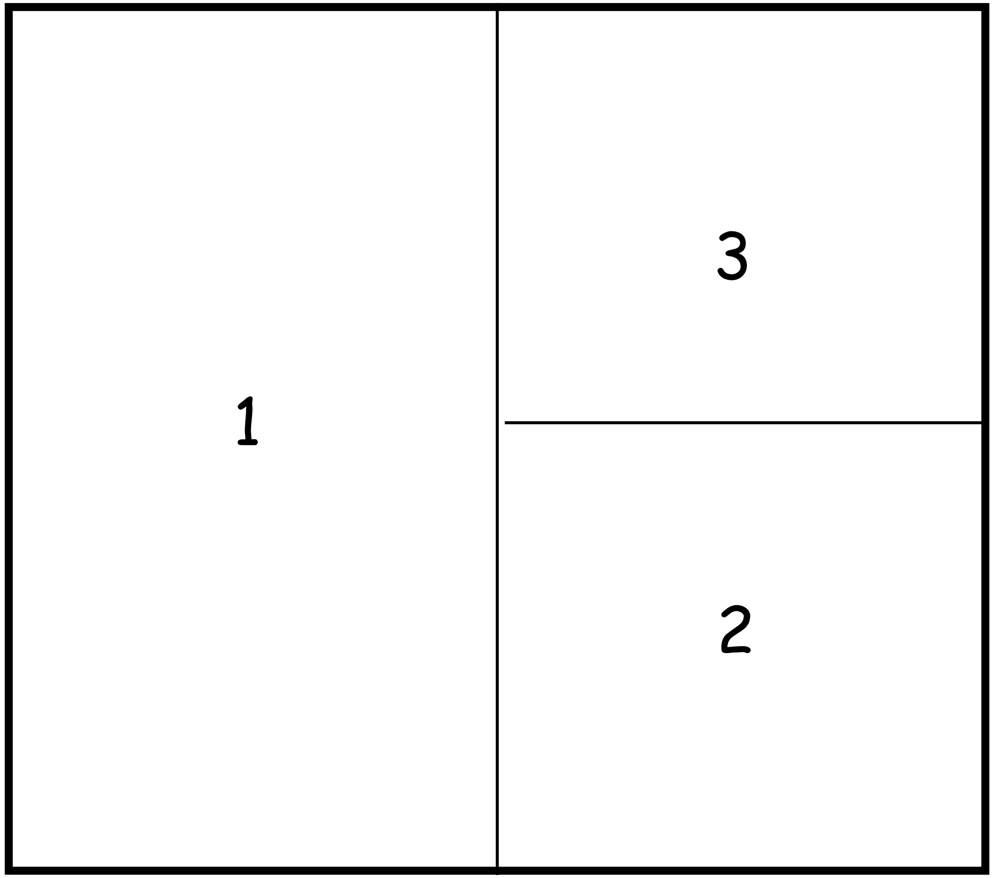
CAN: Simple Example



CAN: Simple Example



CAN: Simple Example



CAN: Simple Example