Distributed Information Processing

14th Lecture

Eom, Hyeonsang (엄현상)
Department of Computer Science & Engineering
Seoul National University

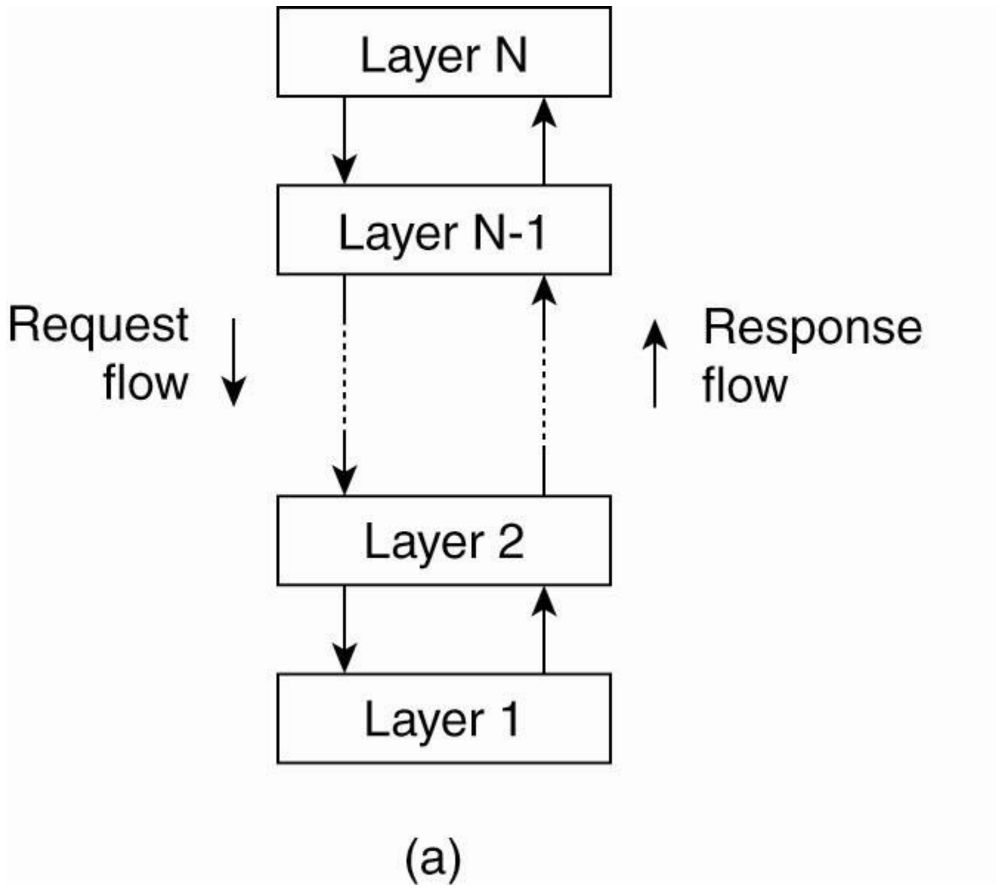
Outline

- Architectures
- Peer-to-Peer Computing
  - Introduction
  - Chord
- Q&A

Architectures

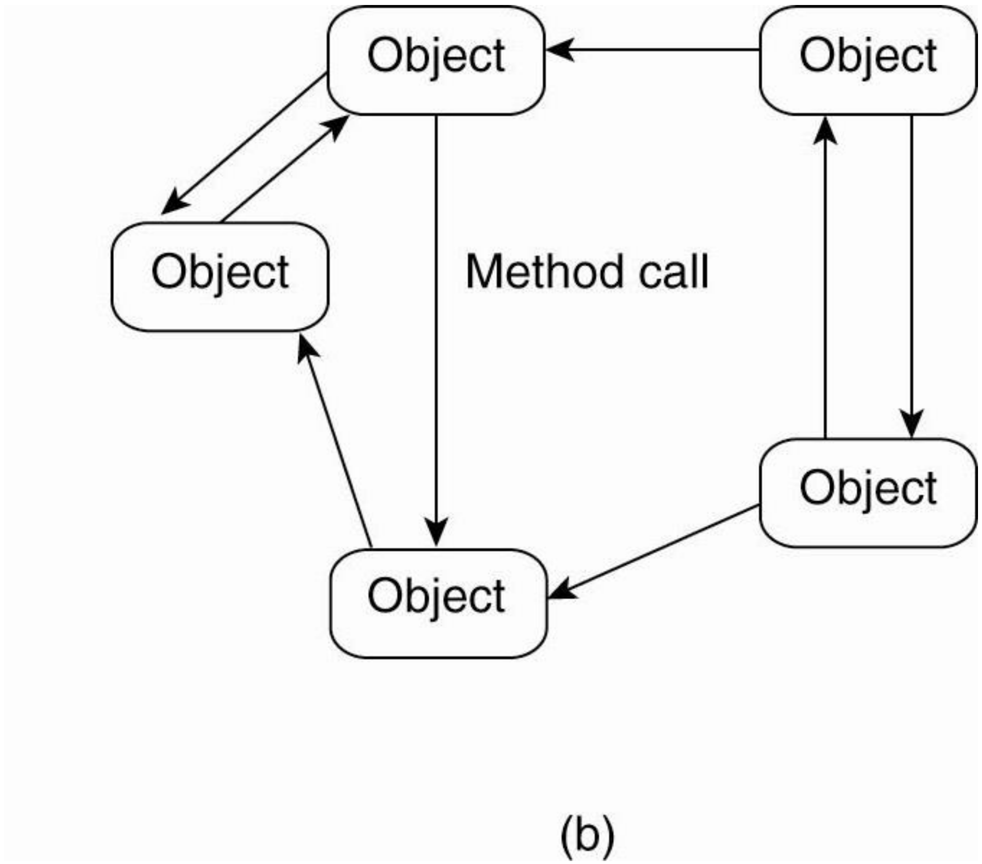
- Software Architecture
  - How Software Components Are Organized
  - How Software Components Should Interact
- System Architecture
  - Final Instantiation of a Software Architecture
- Important Styles of Architecture for (Autonomic) Distributed Systems
  - Layered Architectures
  - Object-Based Architectures
  - Data-Centered Architectures
  - Event-Based Architectures

Architectural Styles



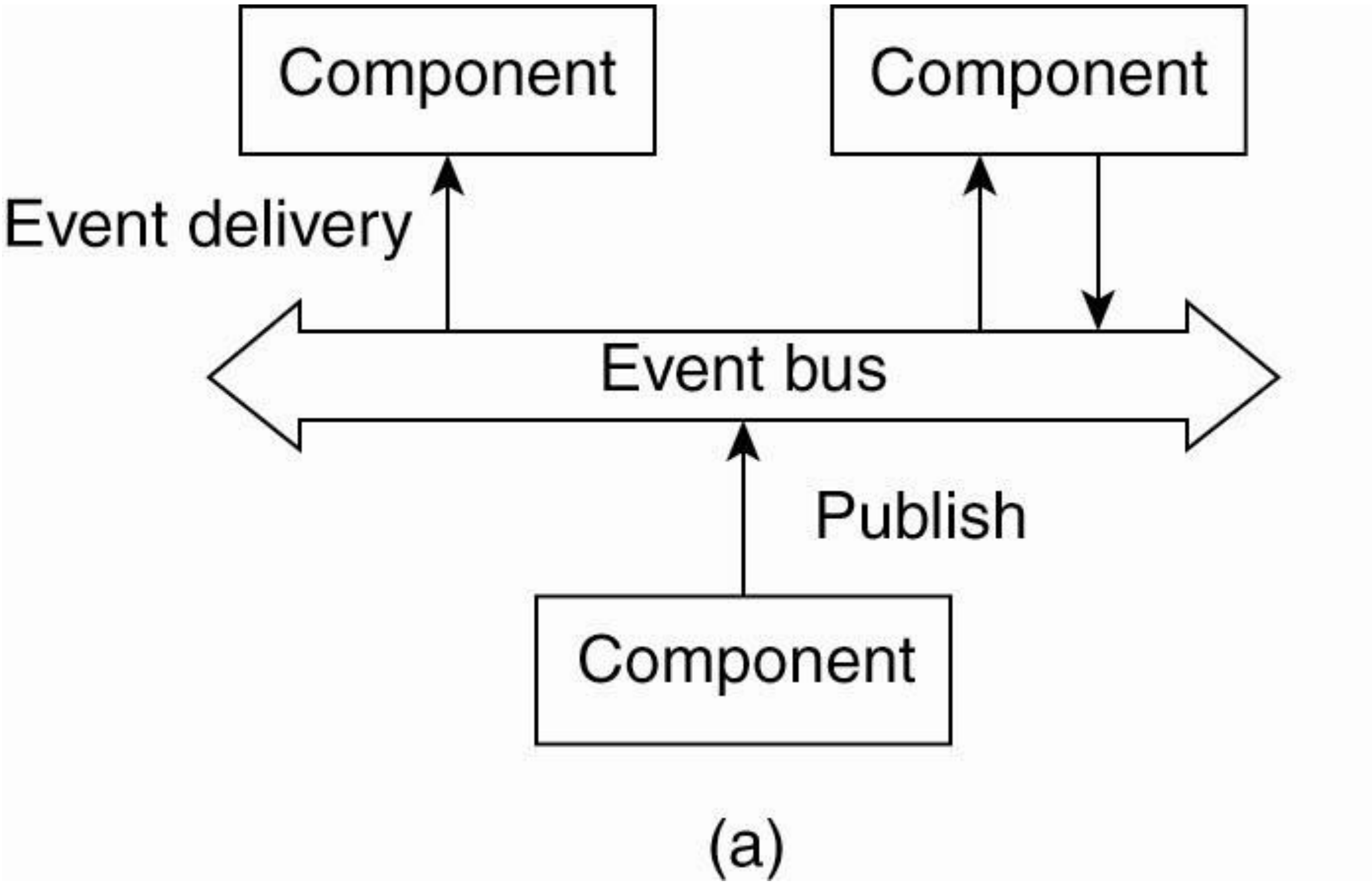
(a) layered architectural style

Architectural Styles (Cont'd)



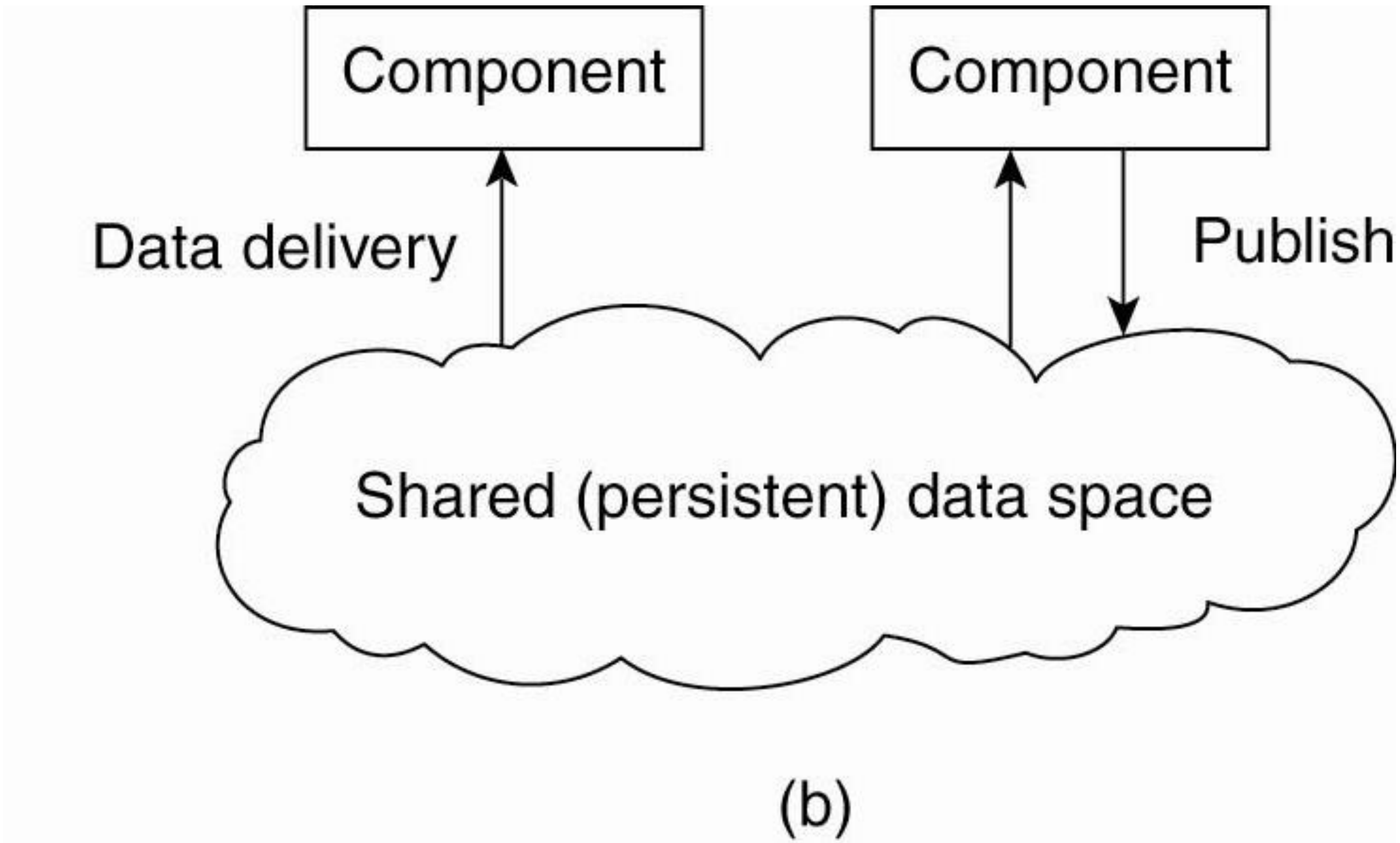
(b) The object-based architectural style

Architectural Styles (Cont'd)



(a) The event-based architectural style

Architectural Styles (Cont'd)



(b) The shared data-space architectural style

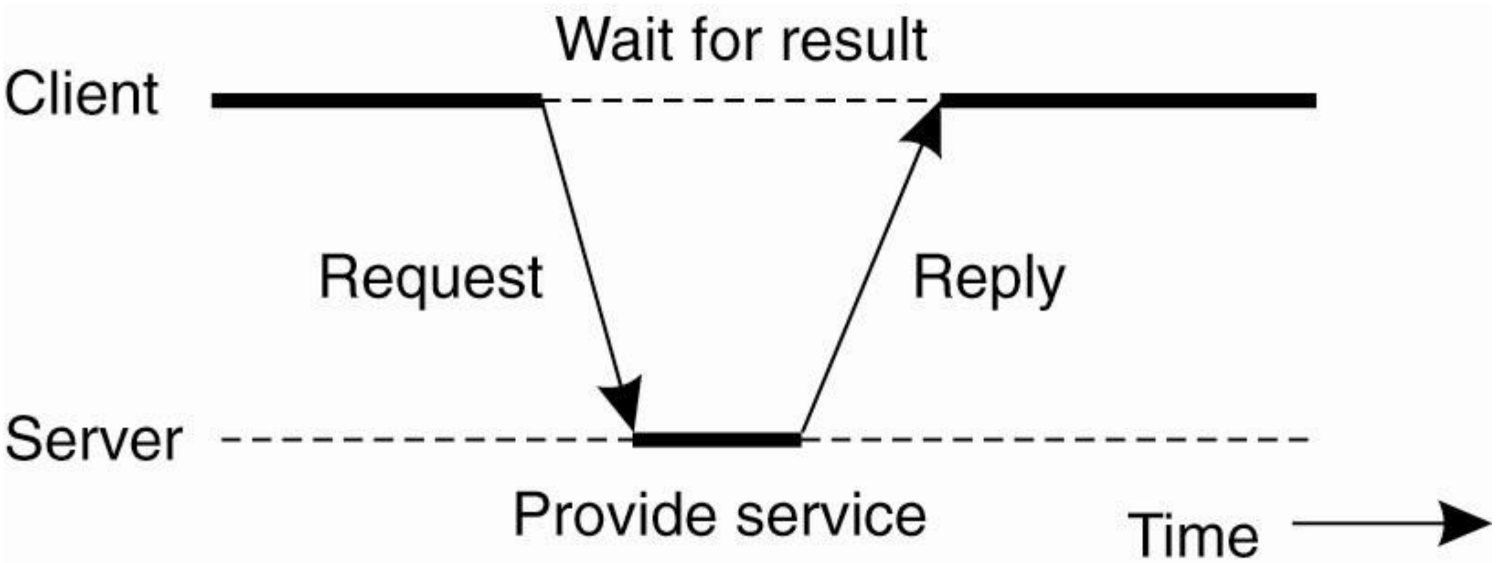
System Architectures

- Centralized Architecture
  - Clients That Request Services from Servers
  - Support for Vertical Distribution
    - Placing different components on different machines
- Decentralized Architecture
  - Process Being a Client and a Server
  - Support for Horizontal Distribution
    - Spitting up a client or server physically into logically equivalent parts with each part operating on its own share of data set

Peer-to-Peer Architectures

- Overlay Network
  - Network in which the nodes are formed by the processes and the links represent the possible communication channels
- Structured P2P Architecture
  - Overlay network is Constructed using a deterministic procedure
    - Distributed Hash Table (DHT)
- Unstructured P2P Architecture
  - Overlay network is Constructed using a random algorithm

Centralized Architectures

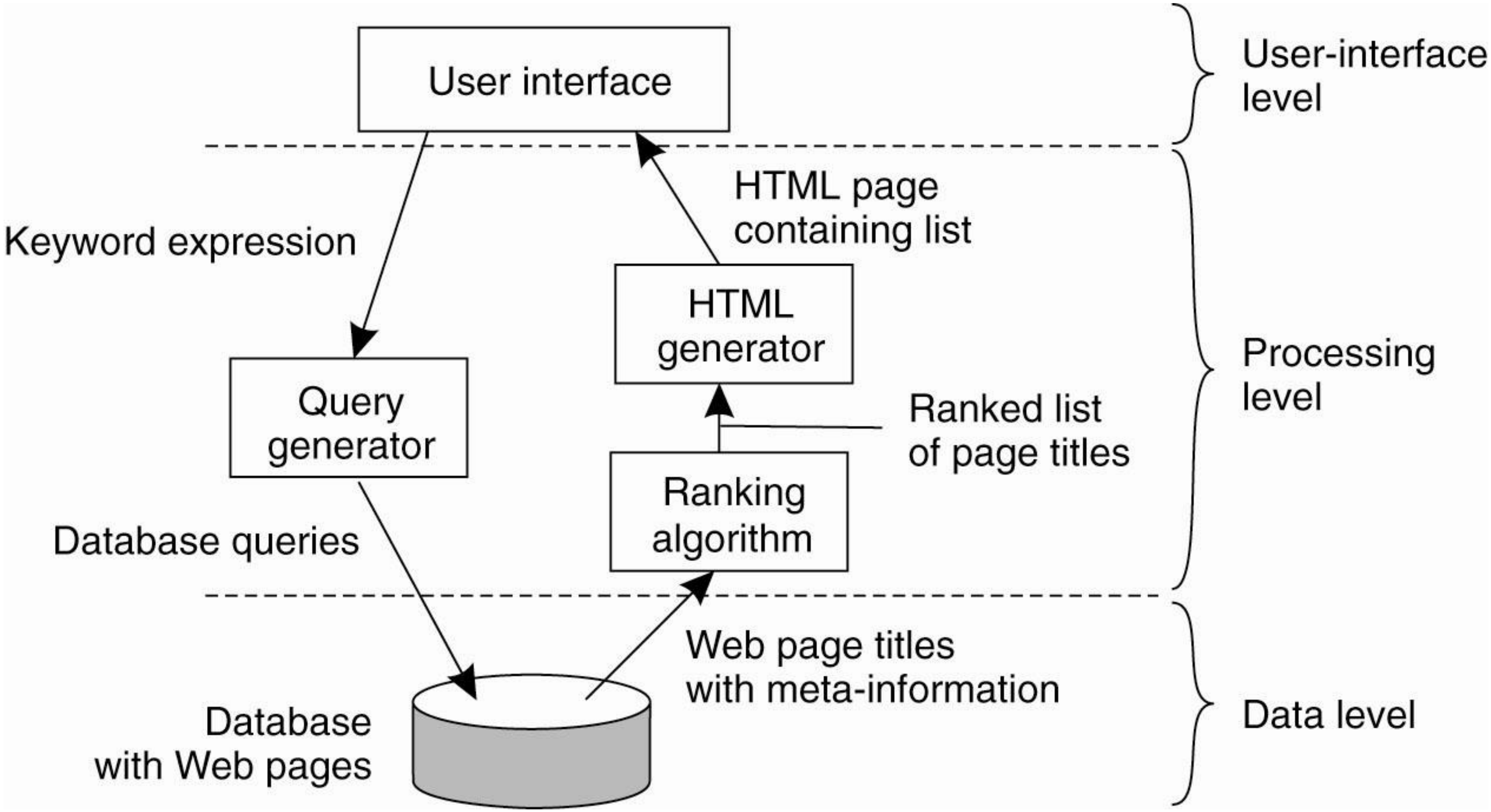


General interaction between a client and a server

Application Layering

- Following Layered Architectural Style
  - User-Interface Level
  - Processing Level
  - Data level

Application Layering (Cont'd)

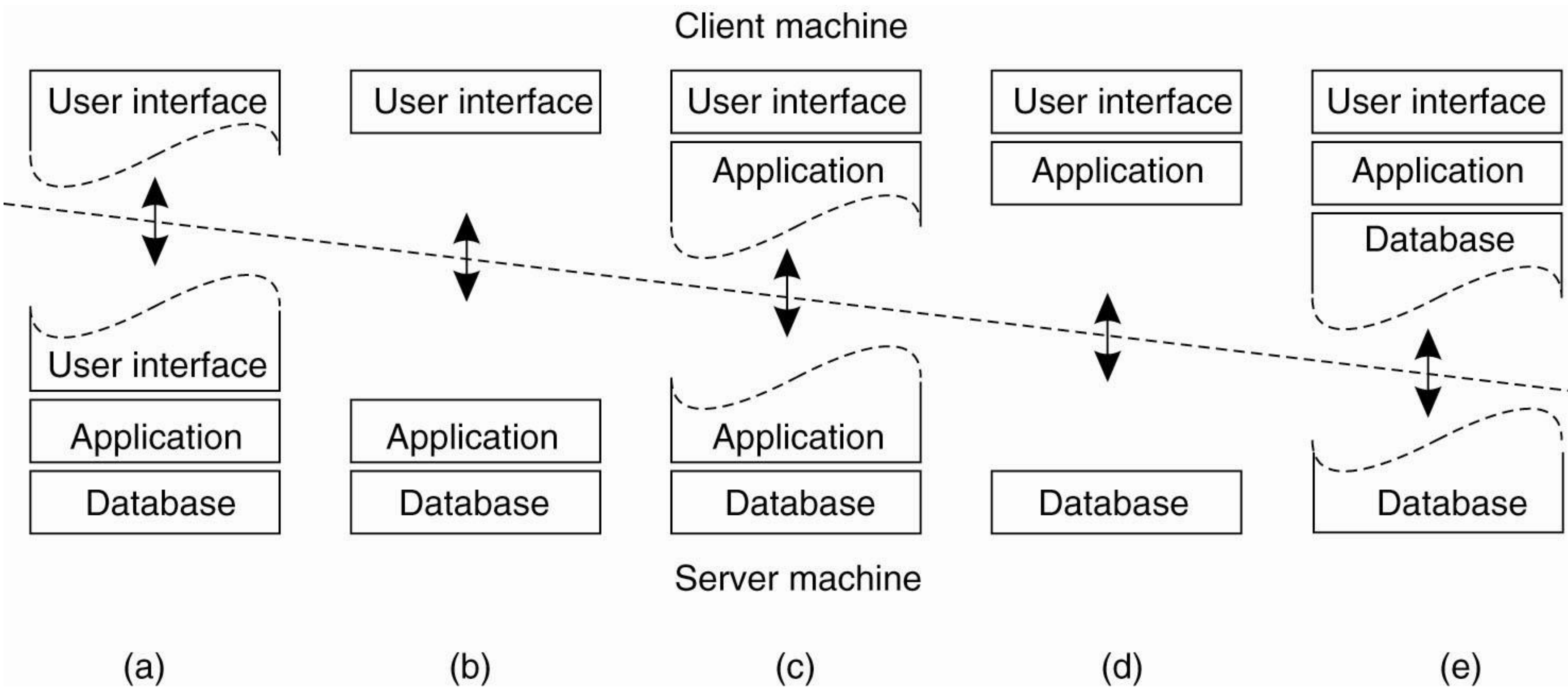


The simplified organization of an Internet search engine into three different layers

Multitiered Architectures

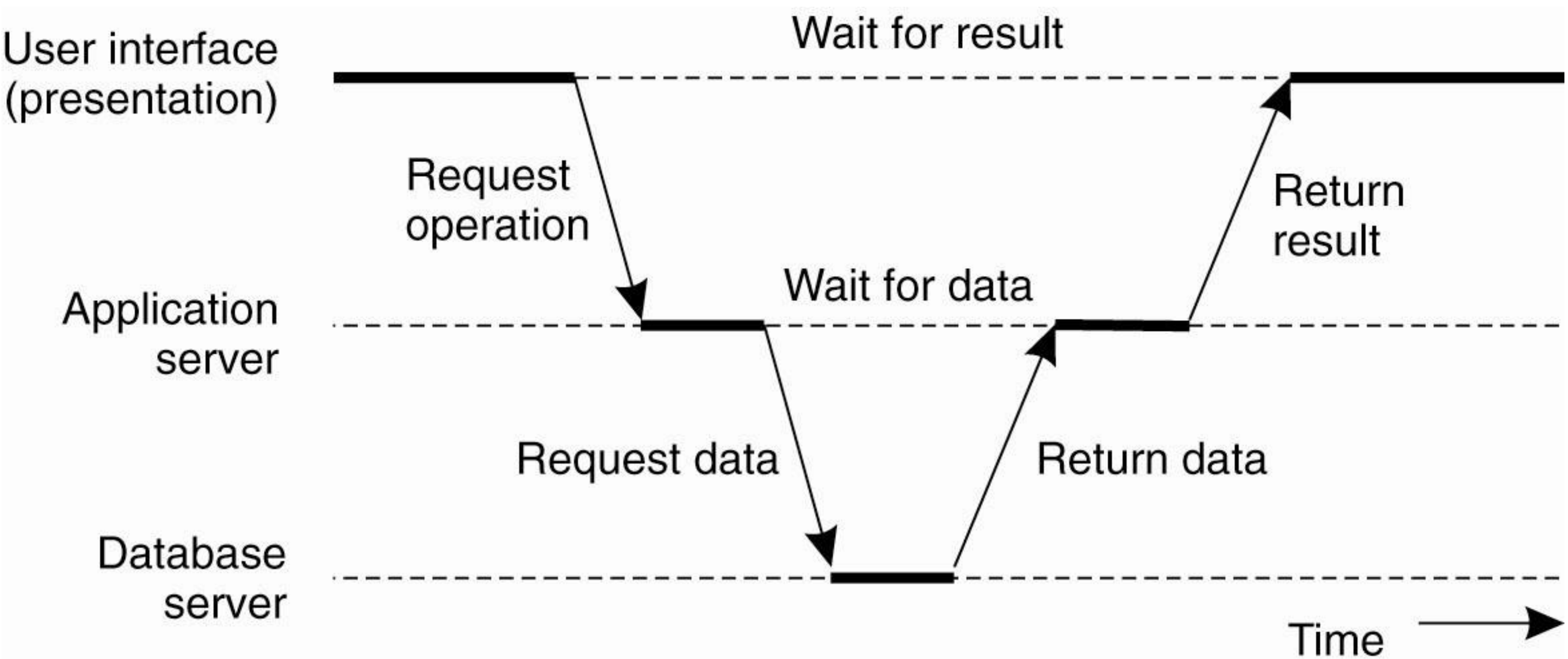
- The simplest organization is to have only two types of machines:
- A client machine containing only the programs implementing (part of) the user-interface level
- A server machine containing the rest,
  - the programs implementing the processing and data level

Multitiered Architectures (Cont'd)



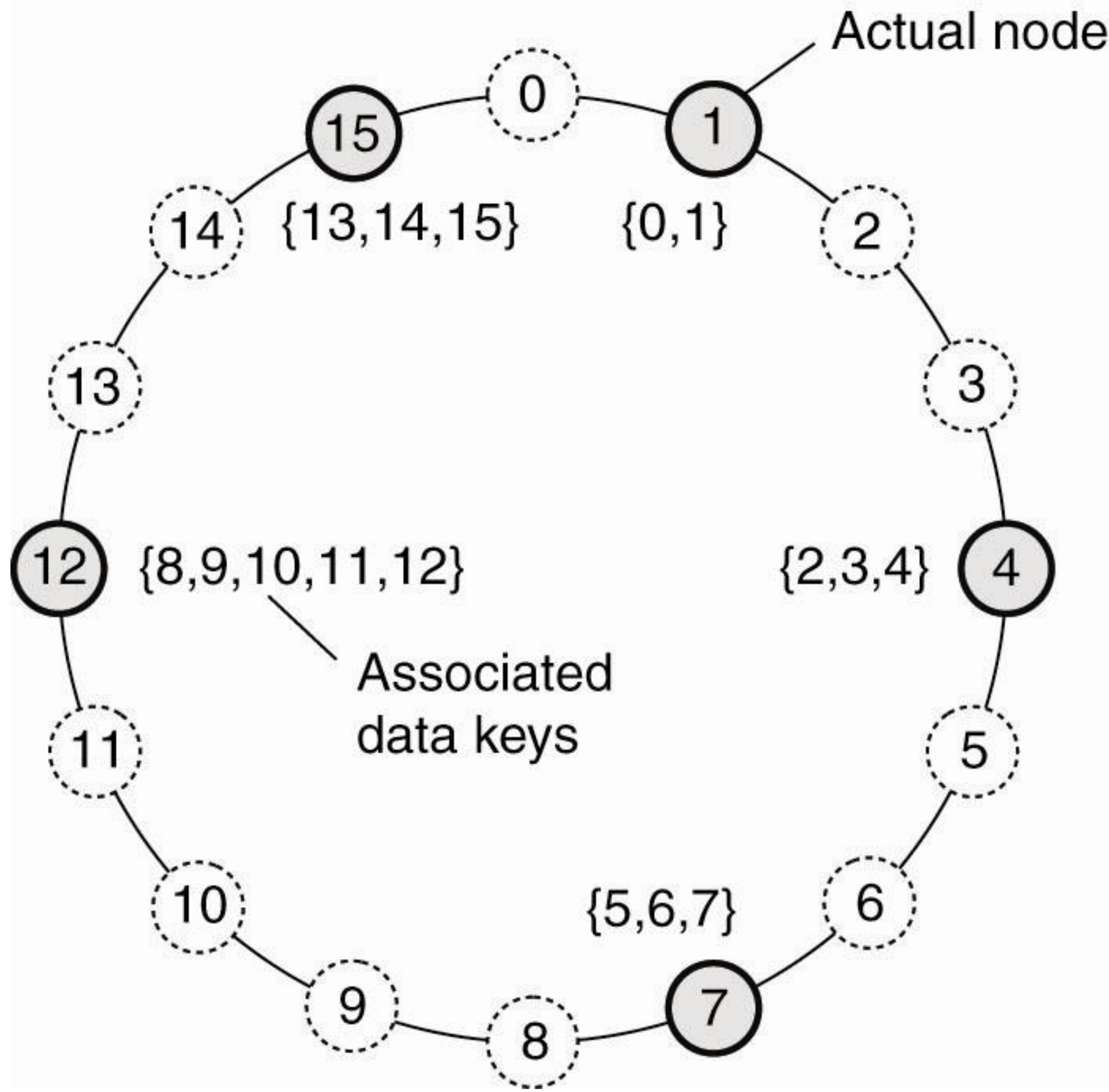
Alternative client-server organizations (a)–(e)

Multitiered Architectures (Cont'd)



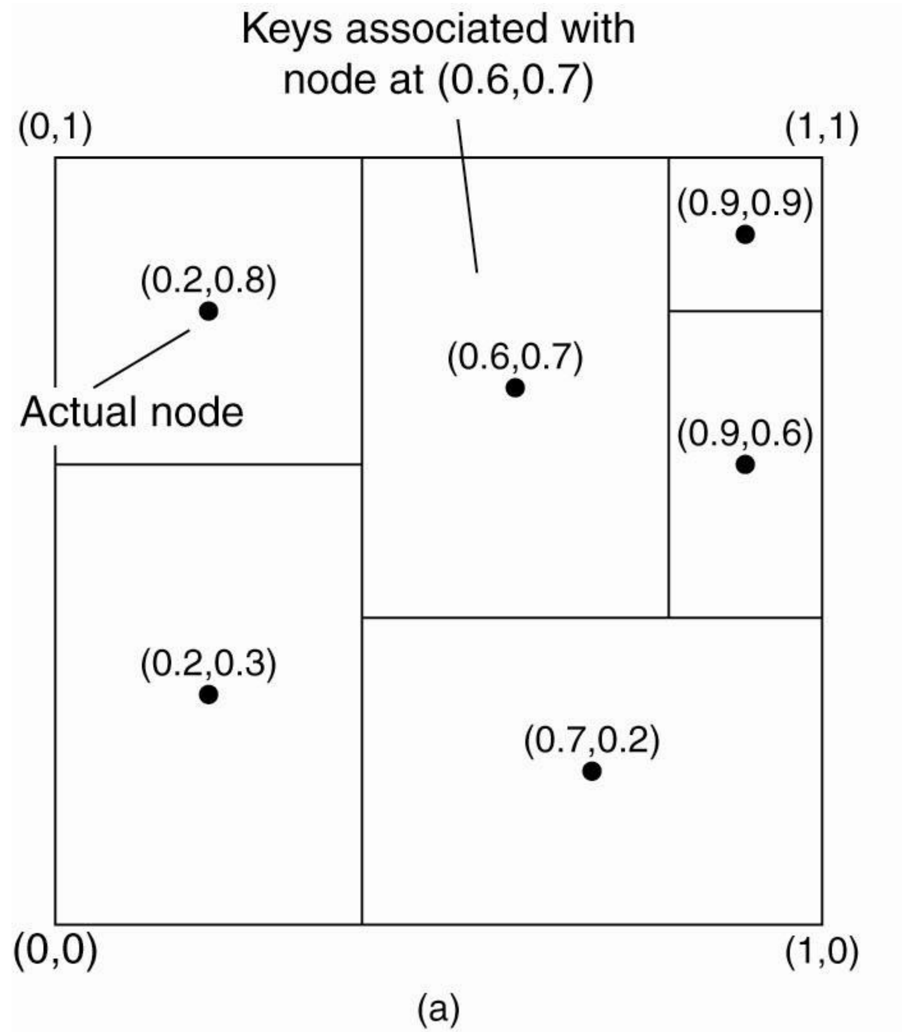
An example of a server acting as client

Structured Peer-to-Peer Architectures



The mapping of data items onto nodes in Chord

Structured Peer-to-Peer Architectures (Cont'd)



The mapping of data items onto nodes in CAN

Peer-to-Peer (P2P) Computing

- Definition
  - Computing by Sharing Data & Resources on a Very Large Scale w/o Server Requirements
- Important Characteristics
  - Each Node's Resource Contribution
  - Same Functional Capabilities & Responsibilities of Nodes
  - No Central Administration
  - Limited Degree of Anonymity
  - Unpredictable Availability
  - Fault Tolerance

Key Issue: Efficient Data Placement & Access

1st-Generation P2P Systems

- File Sharing and Storage Applications
  - Napster Music Exchange Service
    - Use of central servers to locate files
  - Gnutella
    - Distributed service using scoped broadcast queries

Main Problem: Limited Scalability or No Guarantee That Files Can Be Located

2nd-Generation P2P Systems

- Middleware
  - Application-Independent Management of Distributed Resources on a Global Scale
    - Routing Overlay for locating nodes and objects
      - Scalable
      - Load balanced
      - Adaptive to network dynamics
      - Fault tolerant
      - Efficiently discovering
      - Secure

Using Randomly Distributed Keys to Determine the Placement of Objects and to Retrieve Them

Implementing Key-Based Routing (KBR) Interface: Routing of Messages to a Live Node Responsible for the Destination Key

IP vs Overlay Routing

	IP	*Application-level routing overlay*
Scale	IPv4 is limited to 2^32 addressable nodes. The IPv6 name space is much more generous (2^128), but addresses in both versions are hierarchically structured and much of the space is pre-allocated according to administrative requirements.	Peer-to-peer systems can address more objects. The GUID name space is very large and flat (>2^128), allowing it to be much more fully occupied.
Load balancing	Loads on routers are determined by network topology and associated traffic patterns.	Object locations can be randomized and hence traffic patterns are divorced from the network topology.
Network dynamics (addition/deletion of objects/nodes)	IP routing tables are updated asynchronously on a best-efforts basis with time constants on the order of 1 hour.	Routing tables can be updated synchronously or asynchronously with fractions of a second delays.
Fault tolerance	Redundancy is designed into the IP network by its managers, ensuring tolerance of a single router or network connectivity failure. *n*-fold replication is costly.	Routes and object references can be replicated *n*-fold, ensuring tolerance of *n* failures of nodes or connections.
Target identification	Each IP address maps to exactly one target node.	Messages can be routed to the nearest replica of a target object.
Security and anonymity	Addressing is only secure when all nodes are trusted. Anonymity for the owners of addresses is not achievable.	Security can be achieved even in environments with limited trust. A limited degree of anonymity can be provided.

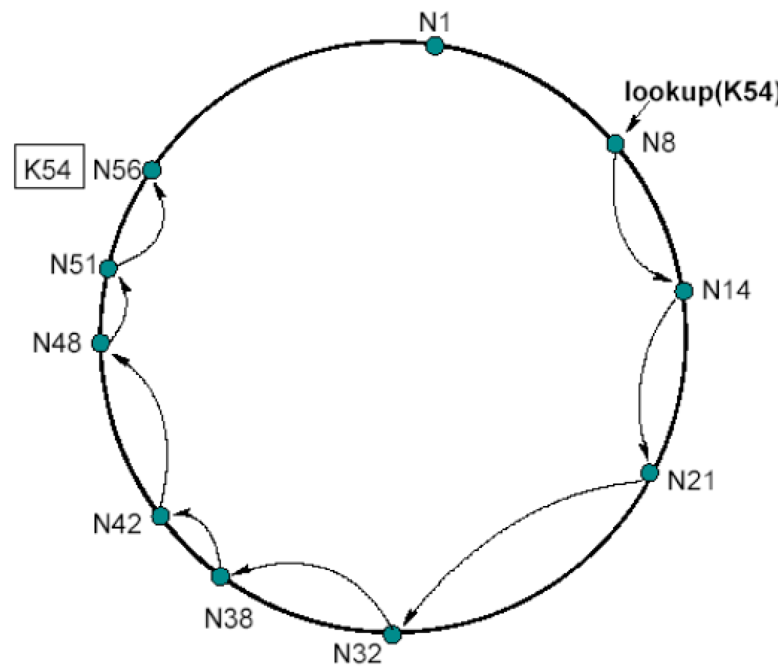
Structured P2P Overlay Networks

- Supporting Higher-Level Interfaces
  - Distributed Hash Table (DHT)
    - Basic Interface: put(), get(), remove()
    - E.g., Pastry
  - Distributed Object Location & Routing (DOLR)
    - Basic Interface: publish(), unpublish(), routeToObject()
    - E.g., Tapestry
- Ignoring/Considering Network Distances
  - Shortest Overlay-Hop Routing
    - E.g., Chord
  - Locally Optimal Routing
    - E.g., Tapestry

Chord Protocol [Keifer03]

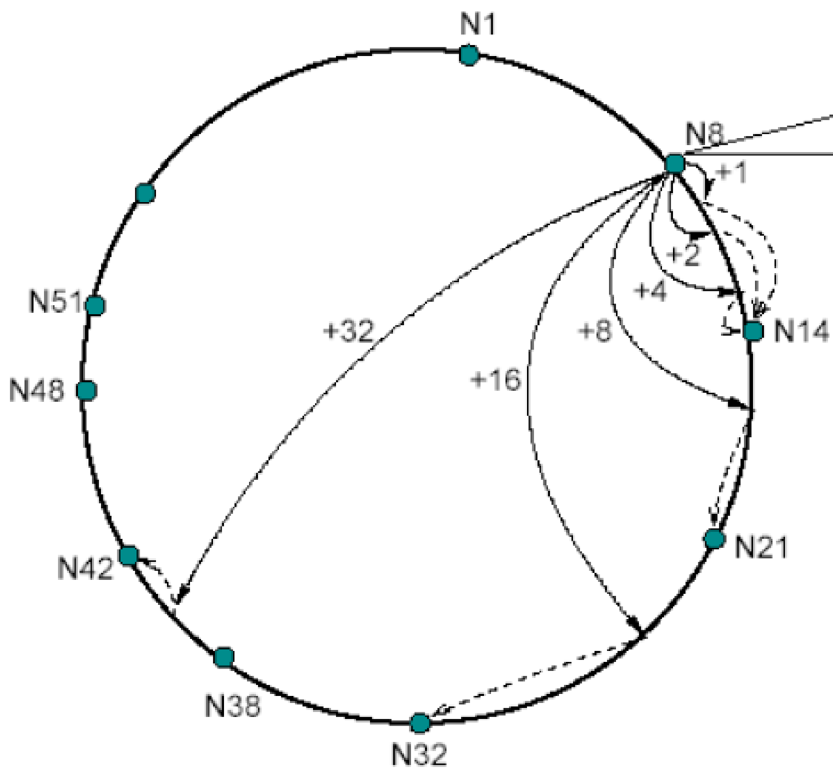
- Simple Key Location

// ask node n *to find the successor of* id
n.find_successor(id)
  if (id ϵ (n; *successor*])
    return *successor*;
  else
    // forward the query around the circle
  return *successor.find_successor*(id);



Cord Protocol (Cont'd)

- Scalable Key Location



Finger table

N8 + 1	N14
N8 + 2	N14
N8 + 4	N14
N8 + 8	N21
N8 +16	N32
N8 +32	N42

$$finger[i] = successor\ (n + 2^{i-1})$$

Cord Protocol (Cont'd)

- Scalable Key Location

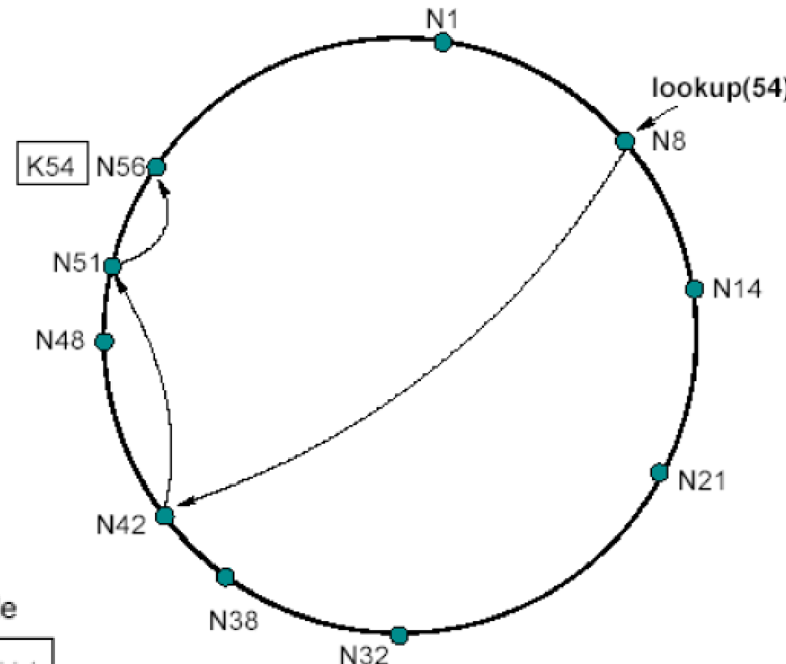
// ask node n *to find the successor of* id
n.find_successor(id)
if (id ϵ (n; *successor*])
return *successor*;
else n0 = closest_preceding_node(id);
return n0.*find successor*(id);

// search the local table for the highest
// predecessor of id
n.closest_preceding_node(id)
for i = m **downto** 1
if (*finger*[i] ϵ (n; id))
return *finger*[i];
return n;

Is This Necessary?

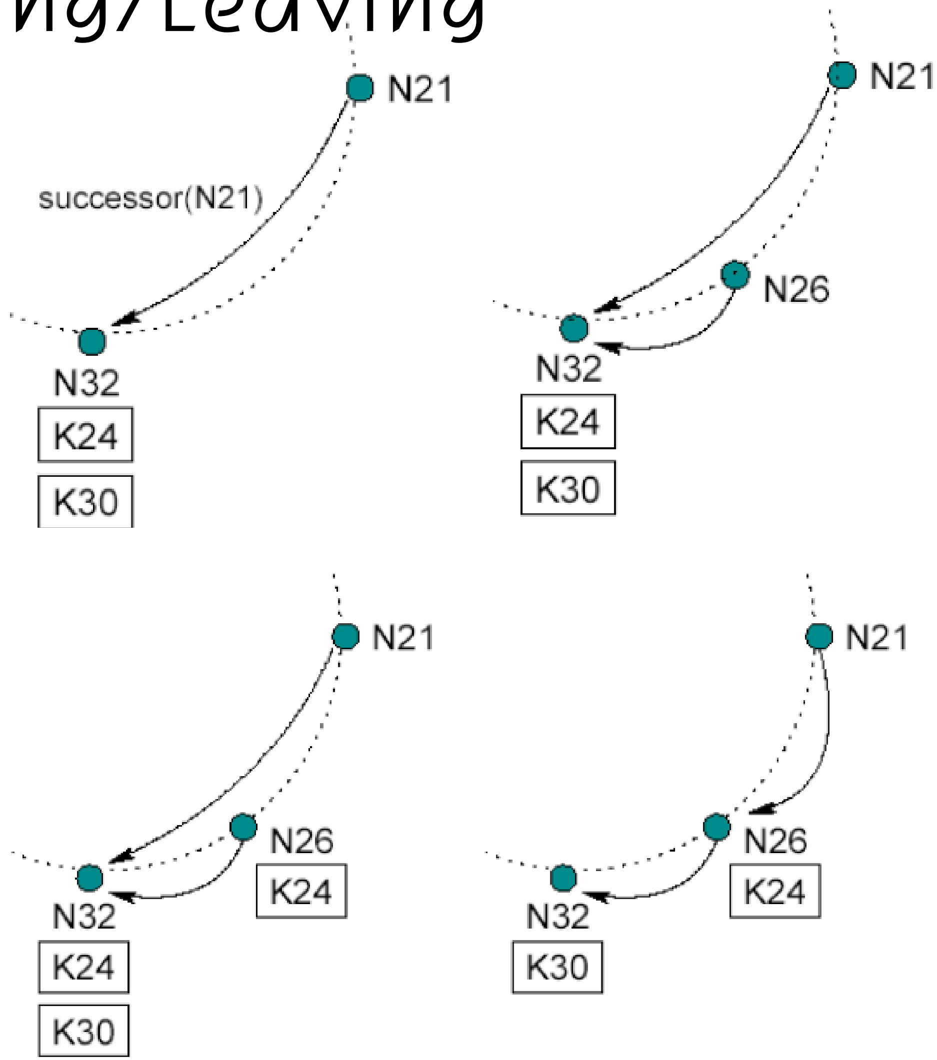
Finger table

N8 + 1	N14
N8 + 2	N14
N8 + 4	N14
N8 + 8	N21
N8 +16	N32
N8 +32	N42



Cord Protocol (Cont'd)

- Node Joining/Leaving



Cord Protocol (Cont'd)

- Properties of Chord
  - Load Balance
    - Acting as a Distributed Hash Function
  - Decentralization
    - Fully distributed
  - Scalability
    - Lookup cost growing as the log of # of nodes
  - Availability
    - Enabling the node responsible for a key to be found via automatic internal-table adjustment
  - Flexible naming
    - Using flat key-space

Reference

- [Keifer03] C. Keifer, "Cord: A Scalable Peer-to-Peer Look-Up Protocol for Internet Applications (by R. Morris, *et al*)," Writeup, Department of Computer Science, Saarland University, November 2002