

Peer-to-Peer Systems

Internet Technologies and Applications

Contents

- Describing and Classifying P2P
- Resource Location in P2P
 - General concepts
 - Unstructured P2P Systems, e.g. Gnutella
 - Hierarchical P2P Systems, e.g. Fasttrack
 - Structured P2P Systems, e.g. DHTs, Chord and BitTorrent
- Comparing P2P Systems
- We focus on file sharing as example of P2P systems
 - Many other applications like distributed file systems, web caching, online gaming, distributed computing, ...

Motivation of P2P Systems

- The Internet enables large-scale distributed applications
 - Searching (Google), directories (Yahoo!), auctions (eBay), ...
 - Most of the applications use a centralised architecture
 - Information is stored centrally on company servers
 - Users (clients) access the information from servers
 - These centralised, distributed applications require significant development, infrastructure, maintenance and administration
 - In 2003, Google used cluster of 15,000 Linux workstations as search engine server
- An alternative to centralised distributed applications?
 - In recent years, peer-to-peer systems have provided large-scale distributed applications using decentralised architecture
 - File sharing applications most popular: Napster, Gnutella, BitTorrent
 - No longer require large, complex centralised servers
 - For many applications, scalability, load-sharing and fault-tolerance of P2P systems makes them very attractive

Client/Server versus P2P

- Client/server systems are asymmetric and centralised
 - Clients request data or functionality from a server
 - To cope with large number of potential clients, server may be replicated (e.g. many physical servers, although conceptually only one server)
 - Disadvantages:
 - Single point of failure – if server fails, the entire service is unusable
 - Bandwidth bottlenecks at server
 - Advantages:
 - Easy to control access to resources and functionality (including providing security)
 - Simple and efficient algorithms
- P2P systems are symmetric and decentralised
 - No distinction between clients and servers; a peer may act as either depending on current objective
 - Individual peers must cooperate with each other
 - Advantages:
 - No longer depend on a single server (no single point of failure)
 - Addition of new users, leads to new server and clients (scales well)
 - Disadvantages:
 - Complex algorithms needed
 - Hard to control access and provide security

Practical Benefits of P2P Systems

- Storage
 - Each peer stores some resources (files)
 - Large amounts of storage space available
- Bandwidth
 - No bottlenecks at central servers
 - Can download parts of files from multiple sources
- Knowledge
 - Peers (users) may classify their resources, making searches easier
 - Similar to Yahoo! Directory classifying resources

Classifying P2P

- P2P systems include applications, protocols and algorithms
- Examples:
 - The Internet (IP) is a P2P system (although many Internet applications are client/server-based)
 - eBay at a user-level is a P2P system (although some protocols used are client/server-based)
- P2P applications may still use a client/server programming model
 - E.g. use TCP/IP sockets: a server process listens for connections, a client process initiates connections
- However, the peer computer will usually run both a server and client process

Classifying P2P by Detail

- Another classification of P2P systems is by the levels of detail (or generality) that they provide:
 - P2P Applications
 - Systems used for specific applications
 - Examples: Gnutella, Napster, Kazaa, ...
 - P2P Platforms
 - Provide generic architecture for building P2P applications
 - Example: JXTA platform
 - P2P Algorithms
 - Algorithms, especially for resource location, that are used by applications and platforms

Resource Location in P2P Systems

- Resource location is a fundamental problem of P2P systems
 - How do you locate a resource in a P2P network?
 - (In a centralised network, you go direct to the server)
- The problem:
 - Given a group of peers G , each peer has an address, p , and stores some resources, $R(p)$, and each resource is identified by a key, k :
 - If you have a key, k , for resource r , find the peer p that stores the resource r
 - To do so, you need an index that maps keys (k) to peers (p)
 - A peer will not store the entire index, and hence must send requests to locate resources to its neighbours N
- Require two protocols to perform:
 - Network Maintenance: nodes can join/leave a group G
 - Data Management: peers can search/insert/delete resources from the group

Network Management Protocol

- If a new node n wants to join a group, the node must know an existing member p
 - Send a join message to p
 - If a new node joins the group the neighbourhood information and index information may be re-organised
- If a node n wants to leave a group, send a leave message to p (or all members of group)
 - Often, the leave is implicit, i.e. there is no leave message
 - E.g. a peer is turned off or fails or has no network connectivity

Data Management Protocol

- Managing resources:
 - $\text{search}(k)$ should return the peers p that contain the resource r , where $\text{Key}(r) = k$
 - $\text{insert}(k,r)$ should add a resource r with key k
 - $\text{delete}(k)$ should delete the resource r , where $\text{Key}(r) = k$
- Implementation of Network and Data Management Protocols
 - Differs among different types of P2P systems

Implementation Choices

- Unstructured versus Structured
 - Unstructured: no information is kept about resources on other peers
 - Nodes are independent of each other; failure resistant
 - Structured: peers store information about other peers' resources
 - Search is much more efficient
- Flat versus Hierarchical
 - Flat: all nodes are equivalent (play same role)
 - Fully distributed, failure resistant
 - Hierarchical: some nodes have special functionality, e.g. only some nodes can search
 - Such is much more efficient

Napster

- History
 - One of the original file sharing applications, released in 1999
 - Reached between 25million and 40million users in 2000/2001
 - Shutdown in 2001/02 due to legal challenges and bankruptcy
- Napster was an application and protocol
- Characteristics
 - Directory based architecture
 - Clients send requests to central server to locate resources (not P2P)
 - Clients then access other peers directly (P2P)
 - Efficient, but lacks several of the benefits of other P2P systems (scalability, fault-tolerance)

Gnutella

- History:
 - Created by two developers from Nullsoft in 2000
 - Mainly used for exchanging files (originally intended for exchanging recipes)
 - Protocol was reverse engineered from the software binary
- Gnutella is a P2P protocol (not an application)
 - Many client applications implement the Gnutella protocol
 - Morpheus, Limewire, Gnucleus, ...
- Characteristics:
 - Unstructured P2P system
 - Flat architecture
 - Loosely coupled

Gnutella Protocol

- Message types:
 - Network maintenance: Ping, Pong
 - Data management: Query, QueryHit, Push
- Message distribution
 - Messages are broadcast with Time To Live (TTL) decremented by 1 each time
 - If receive a message with $TTL > 0$ (and not received before), then forward message to all peers you have connections with
 - Responses to Query messages are sent along same path
- Joining the Gnutella network
 - A new peer, P , must contact an existing peer with a Ping message
 - There are dedicated servers that list known peers – initially the new peer must contact one of these
 - Peers receiving Ping message can cache new peers, P , IP address/port and respond with Pong message including IP address, port and total size of files it shares
 - New peer, P , selects C (e.g. 4) of the peers who returned a Pong, and creates permanent connection to them
 - If connections to these C peers are lost, the P can find new peers to permanently connect to

Gnutella Protocol

- Locating files:
 - P sends a Query message to permanent peers, including search criteria
 - If a peer X can satisfy criteria, it returns QueryHit listing all matches
 - HTTP can then be used to access the file
 - Otherwise, forward message to all of X 's permanent peers
- List of peers:
 - Peer P learns about peers from Ping/Pong, QueryHits and Push messages
 - P caches a list of peers for future use (e.g. if one of the C permanent connections fails)
- Firewalls:
 - If server peer is behind a firewall, requesting peer may not download file with HTTP
 - Requesting peer sends Push message to server peer, indicating where the server peer can “push” the file to (e.g. upload)

Issues with Gnutella

- Simple broadcast of messages is inefficient
 - Example: if TTL is 7, and C is 4, a single Gnutella message may generate 26,240 messages in network
 - Every node that receives a request scans its local database (time consuming)
 - There are methods to improve the broadcast messaging:
 - Expanding ring search
 - Start with TTL=1. If no result found, set TTL=2 and try again. Then try TTL=3 and so on.
 - Random walker search
 - K random walkers are sent by requesting peer. Subsequent peers only send requests to one neighbour, but with high TTL.
 - Can greatly reduce the message overhead, but increases the search time

Fasttrack

- History:
 - Developed in 2001
 - Several of the Fasttrack networks (e.g. Grokster, Kazaa) have been shutdown or limited by legal suits
- Fasttrack is a P2P protocol (not application)
 - Clients implementing Fasttrack include: Kazaa, Grokster and iMesh
- Characteristics:
 - Hierarchical P2P system
 - Super-peer architecture

Super-Peer Architecture

- Aim to combine efficiency of centralised architecture (e.g. Napster) with robustness of flat architecture (e.g. Gnutella)
- Three types of peers:
 - Super-super-peers: used on startup to provide list of super-peers
 - Super-peers: maintain index information and forward messages between other super-peers (similar to Gnutella)
 - Ordinary peers: contact super-peers to advertise resources and access index information (i.e. search). Similar to a centralised approach
- Client software (e.g. Kazaa) can dynamically change node from peer to super-peer
 - Depends on computer and network speed; only powerful computers with high bandwidth will become super-peers

Distributed Hash Tables

- History
 - Motivated by disadvantages of Napster (centralised), Gnutella (inefficient) and similar P2P protocols for file sharing
 - Research has been used for file sharing, instant messaging, distributed file systems, web caching and other fields
- DHTs are P2P algorithms (not protocol or application)
 - Chord, Pastry and Tapestry are specific DHT algorithms
 - BitTorrent is an example protocol/application that uses DHTs
 - Coral Content Distribution Network also uses DHTs
- Characteristics
 - Structured P2P system
 - Flat architecture
 - Tightly coupled

Chord: an example DHT

- N nodes in network
 - Aim to distribute files amongst the nodes, and locate the files
- Consistent hashing is used to assign ID's to nodes and resources
 - SHA-1 hash of node IP address produces 160-bit ID
 - SHA-1 hash of file name produces 160-bit key, k
- Visualise nodes as circle, ordered by ID
- Resource with key k is stored at node with $ID = k$
 - If node with $ID = k$ does not exist, resource is stored at node with next highest ID
- Node n joins network:
 - Need to reassign keys from successor(n)
- Node n leaves the network:
 - Need to reassign keys to successor(n)

Chord: Inserting and Searching

- To insert data, $\text{insert}(data)$
 - Node calculates hash of data (e.g. the file) to get k
 - Routing is used to find the node that stores key k ($node_k$)
 - Data is stored on the node
- To search for data, $\text{search}(k)$
 - Node calculates hash of data to get k
 - Routing is used to find the node that stores k
 - Any access method (e.g. HTTP) is used to retrieve data from $node_k$

Chord: Routing

- Simple Routing

- Each node n maintains route to $\text{successor}(n)$
 - Node n knows the IP address/port number of $\text{successor}(n)$
- A simple (but naïve) approach is to then try to find key by checking each subsequent successor
 - Example: node 0 knows the IP address/port of node 1; node 1 knows IP address/port of node 3; and so on
 - But may have to traverse all nodes to find key
- But can provide more efficient search than this (at expense of maintaining more connections)

- Routing in Chord

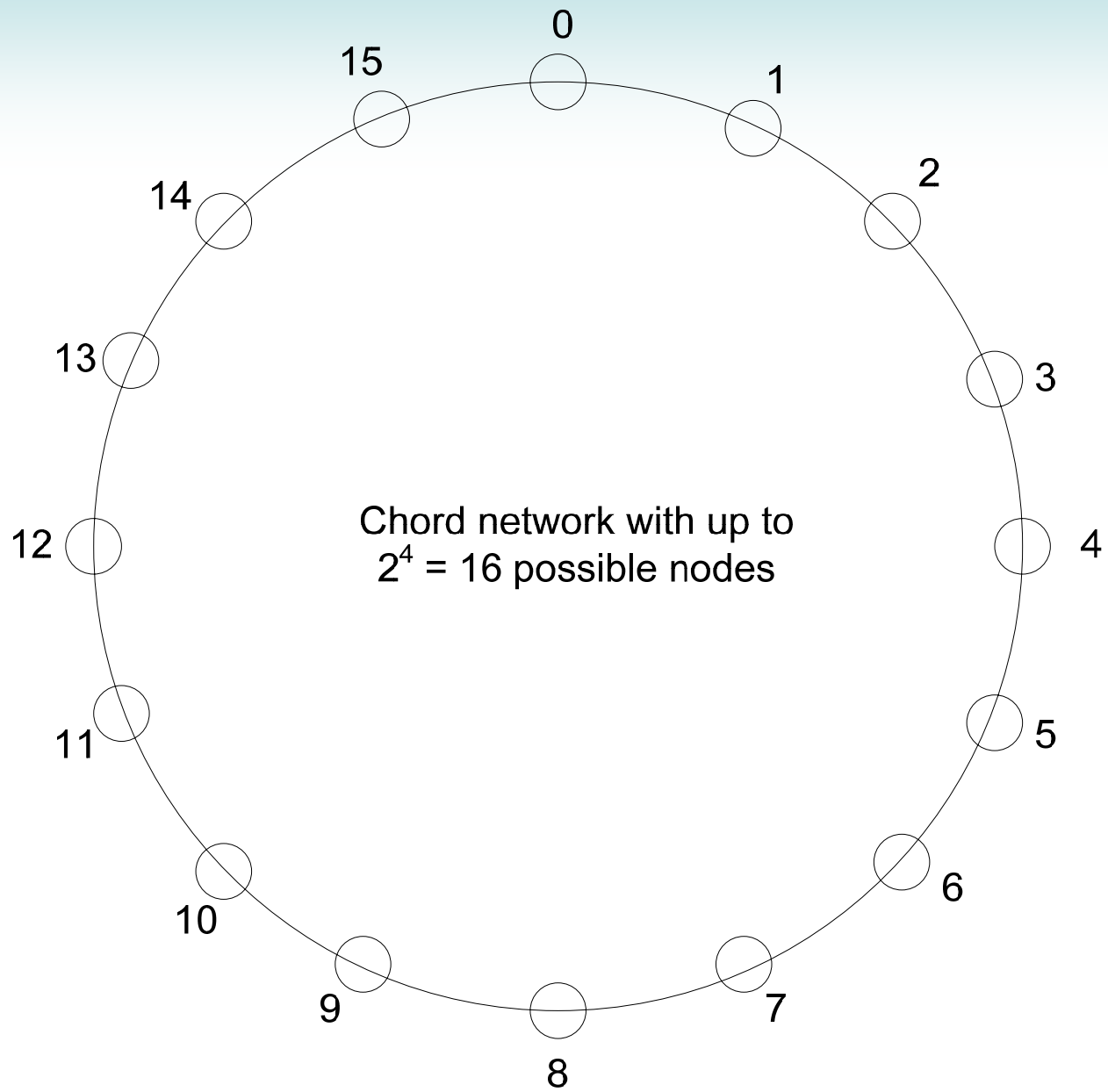
- Each node n maintains route to first node that succeeds n by 2^{i-1}
 - Example: node 0 knows route to 1, 2, 4, 8 (or next subsequent node if does not exist); node 1 knows route to 2, 3, 5, 9; and so on
- Search queries are sent to closest node to the requested key k

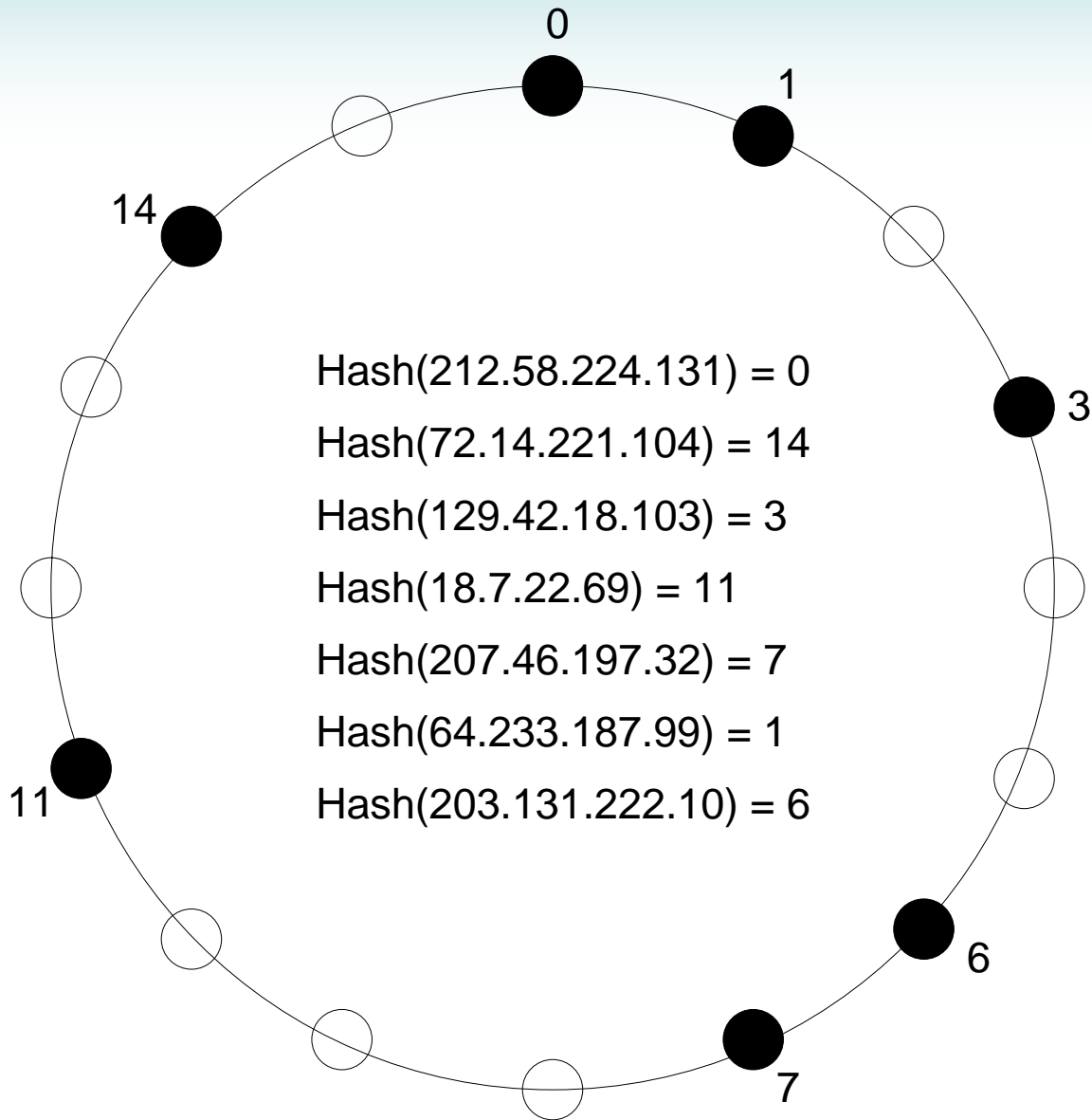
Chord: Routing Example

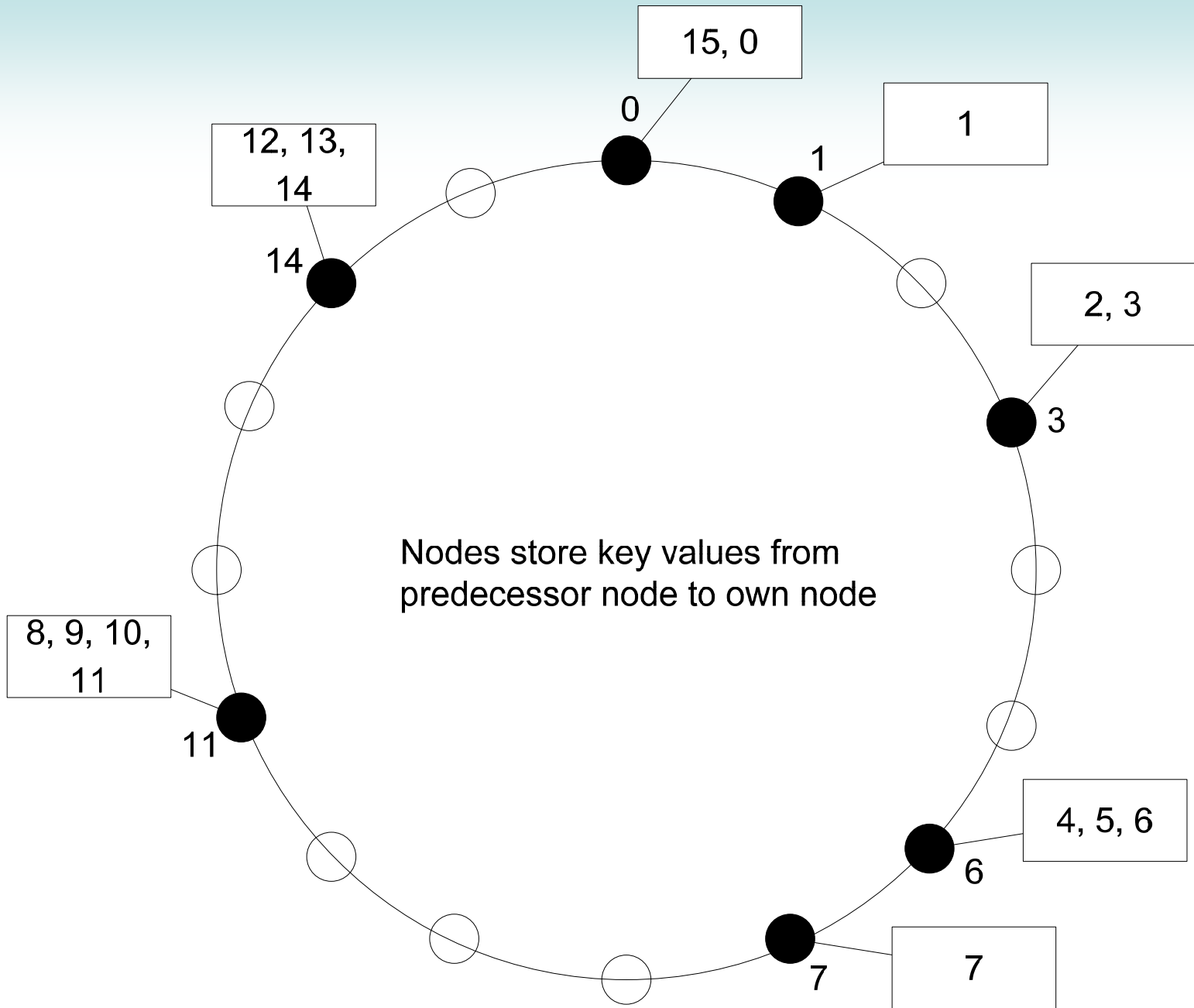
- Node 0 routing (or finger) table:
 - Start = 1; Interval = 1 \rightarrow 1; Successor = 1
 - Start = 2; Interval = 2 \rightarrow 3; Successor = 3
 - Start = 4; Interval = 4 \rightarrow 7; Successor = 7
 - Start = 8; Interval = 8 \rightarrow 15; Successor = 9
- (The 3rd line can be read as: “in order to find a node with key 4, 5, 6 or 7, then send to node 7”)
- Node 9 routing (or finger) table:
 - Start = 10; Interval = 10 \rightarrow 10; Successor = 11
 - Start = 11; Interval = 11 \rightarrow 12; Successor = 11
 - Start = 13; Interval = 13 \rightarrow 0; Successor = 14
 - Start = 1; Interval = 1 \rightarrow 8; Successor = 1

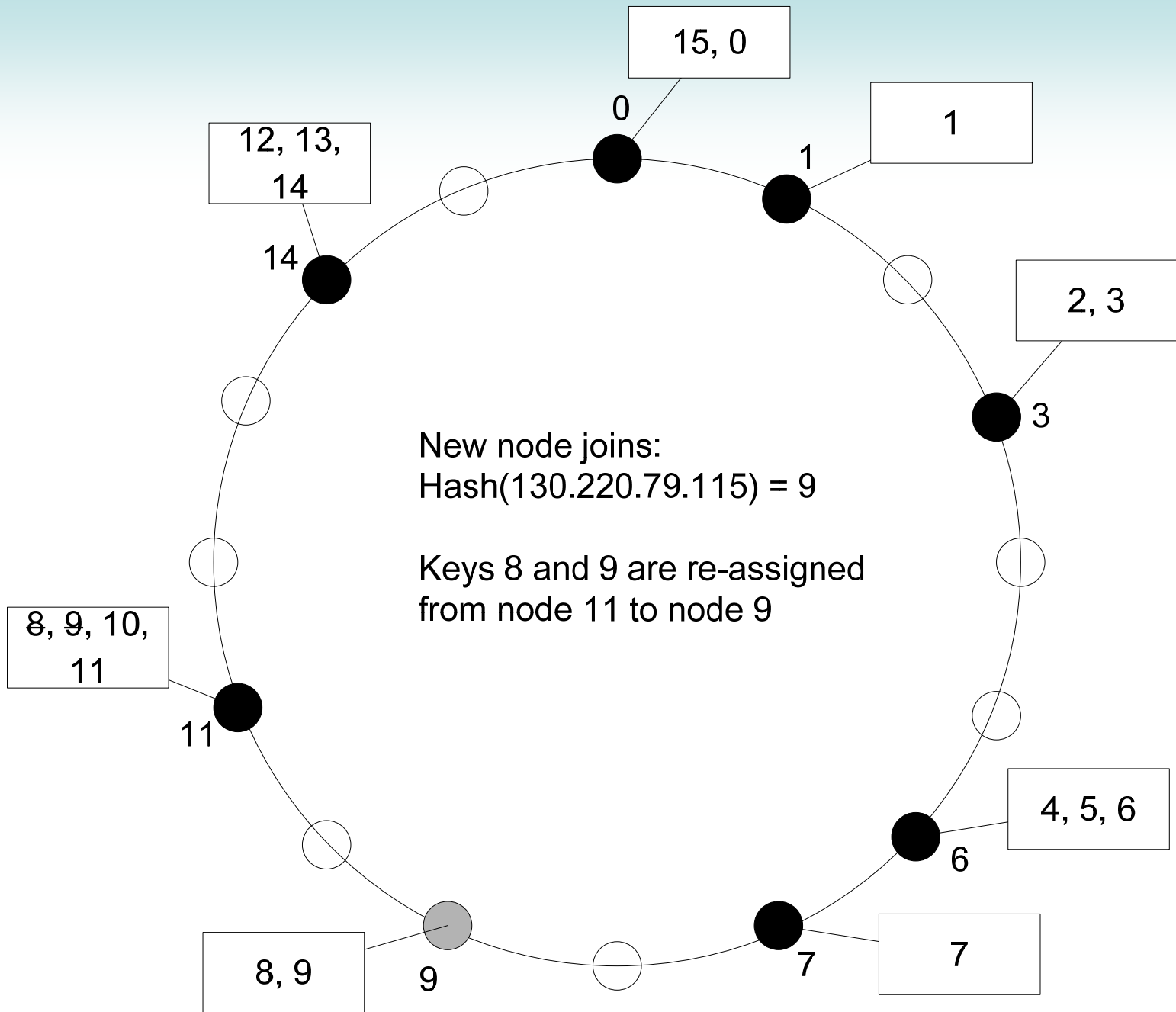
Chord: Routing Example

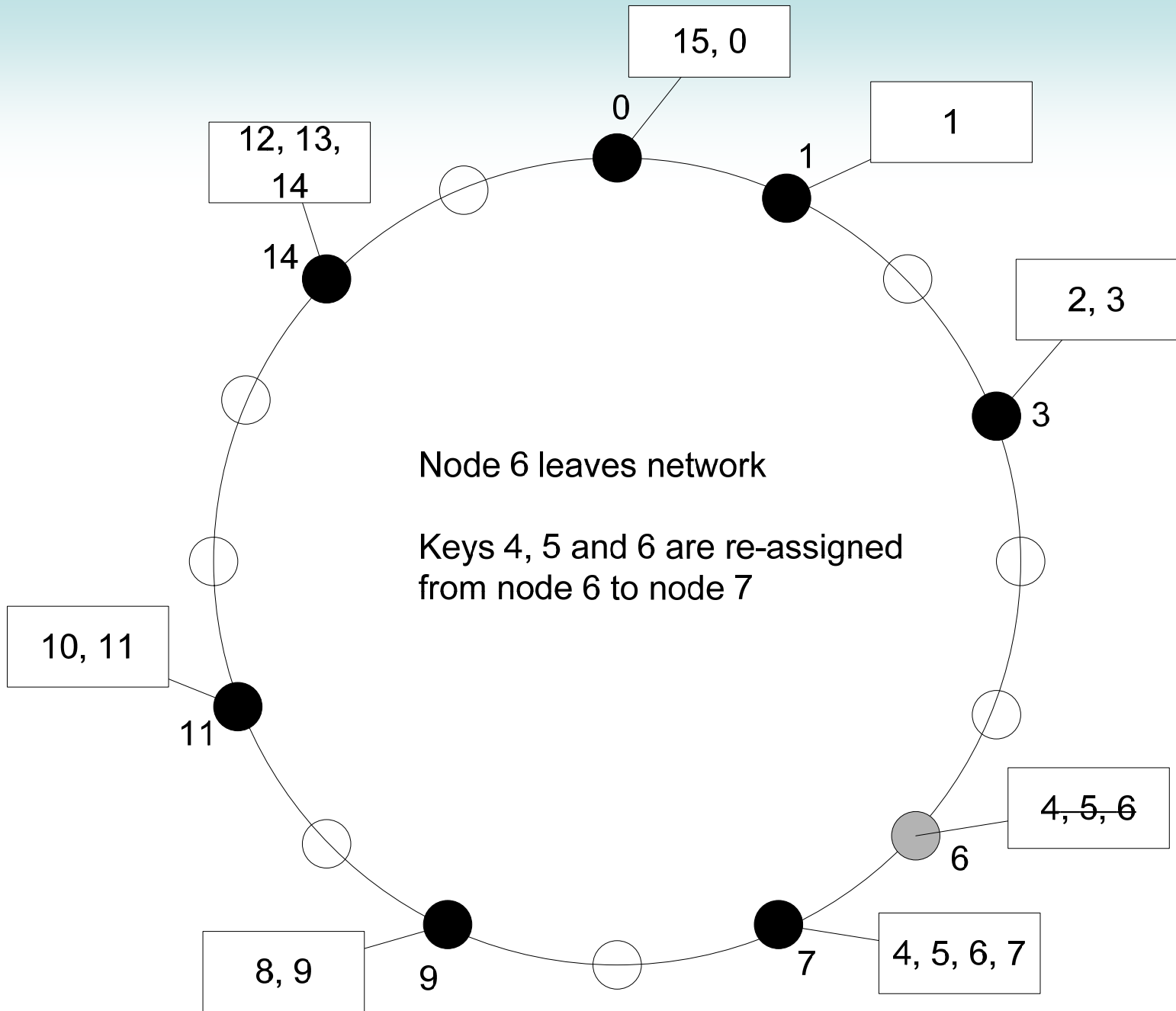
- Assume a search($k=13$) is performed at node 0
 - Node 0 knows node 9 has coverage of the keys from 8 to 15
 - Node 0 sends a search message to node 9
 - Node 9 does not have the data with key 13
 - Node 9 knows node 14 has coverage of the keys from 13 to 0
 - Node 9 sends a search message to node 14
 - Node 14 has the data with key 13
 - Node 14 responds directly to node 0
 - Assumes the original search query includes node 0's IP address/port
- Key benefits:
 - A node stores information about a small number of other nodes
 - Routes to m nodes, if there are up to 2^m nodes in the network
 - This is good – reduces amount of maintenance between nodes
 - A node can quickly locate the resource

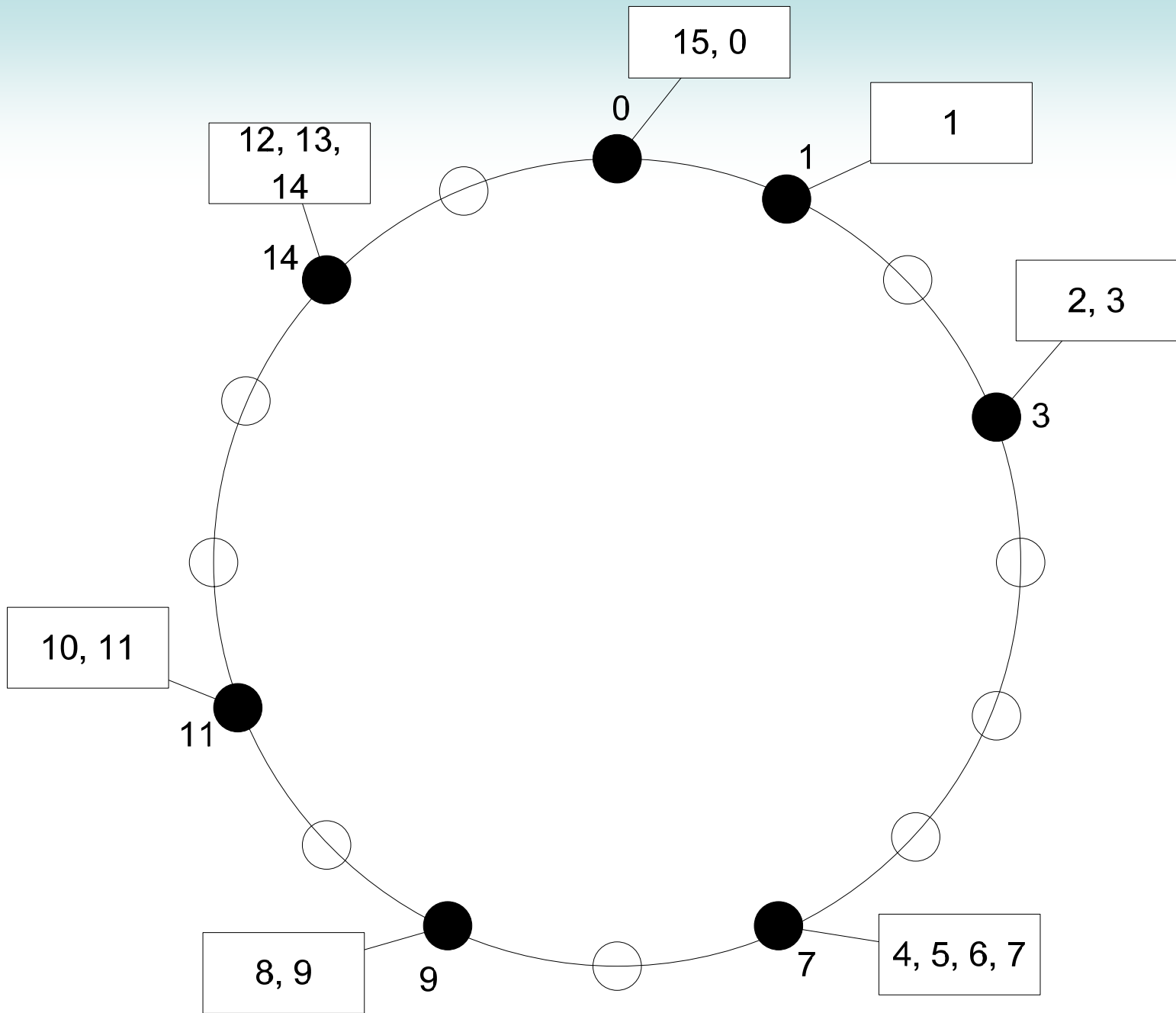


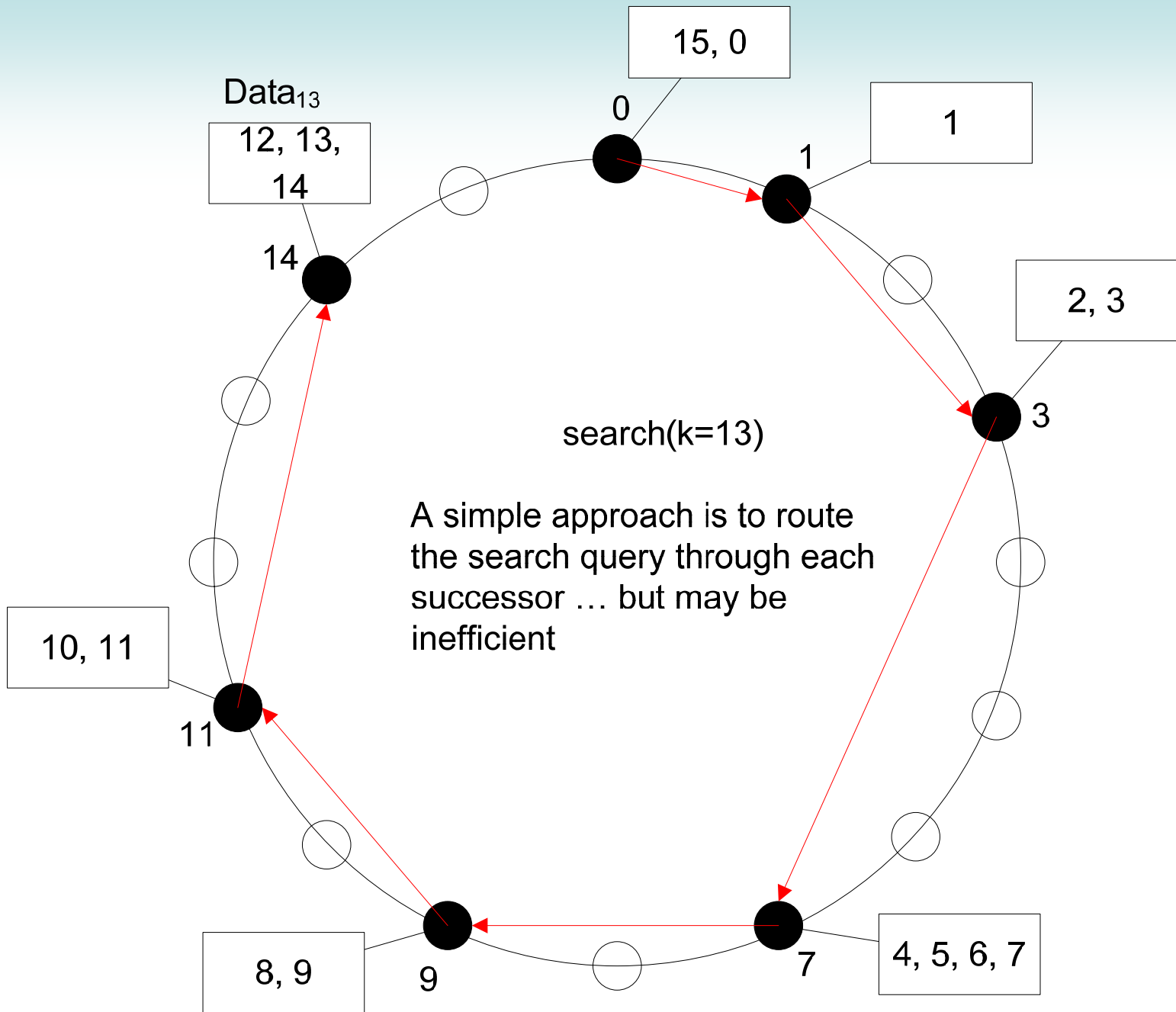


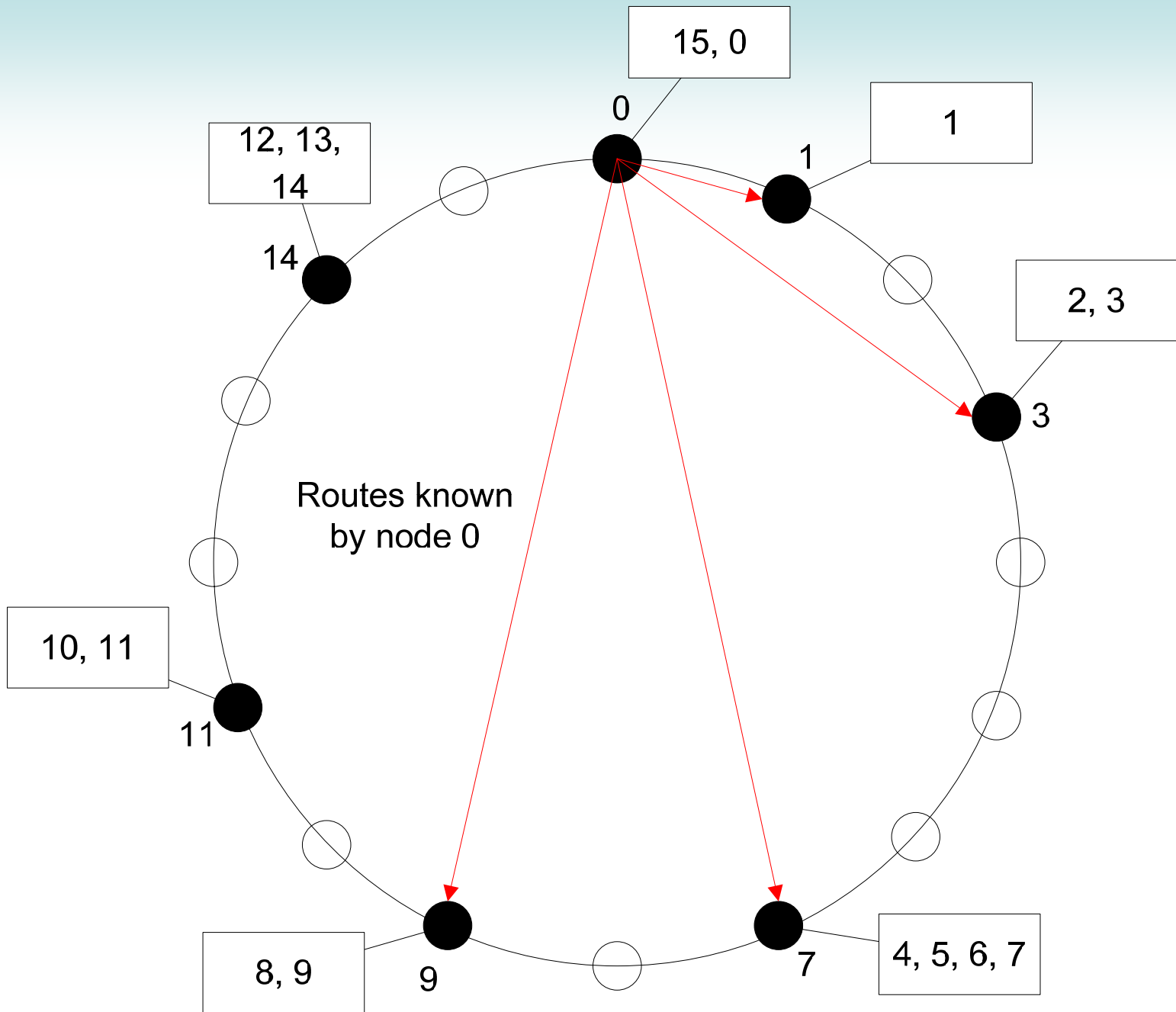


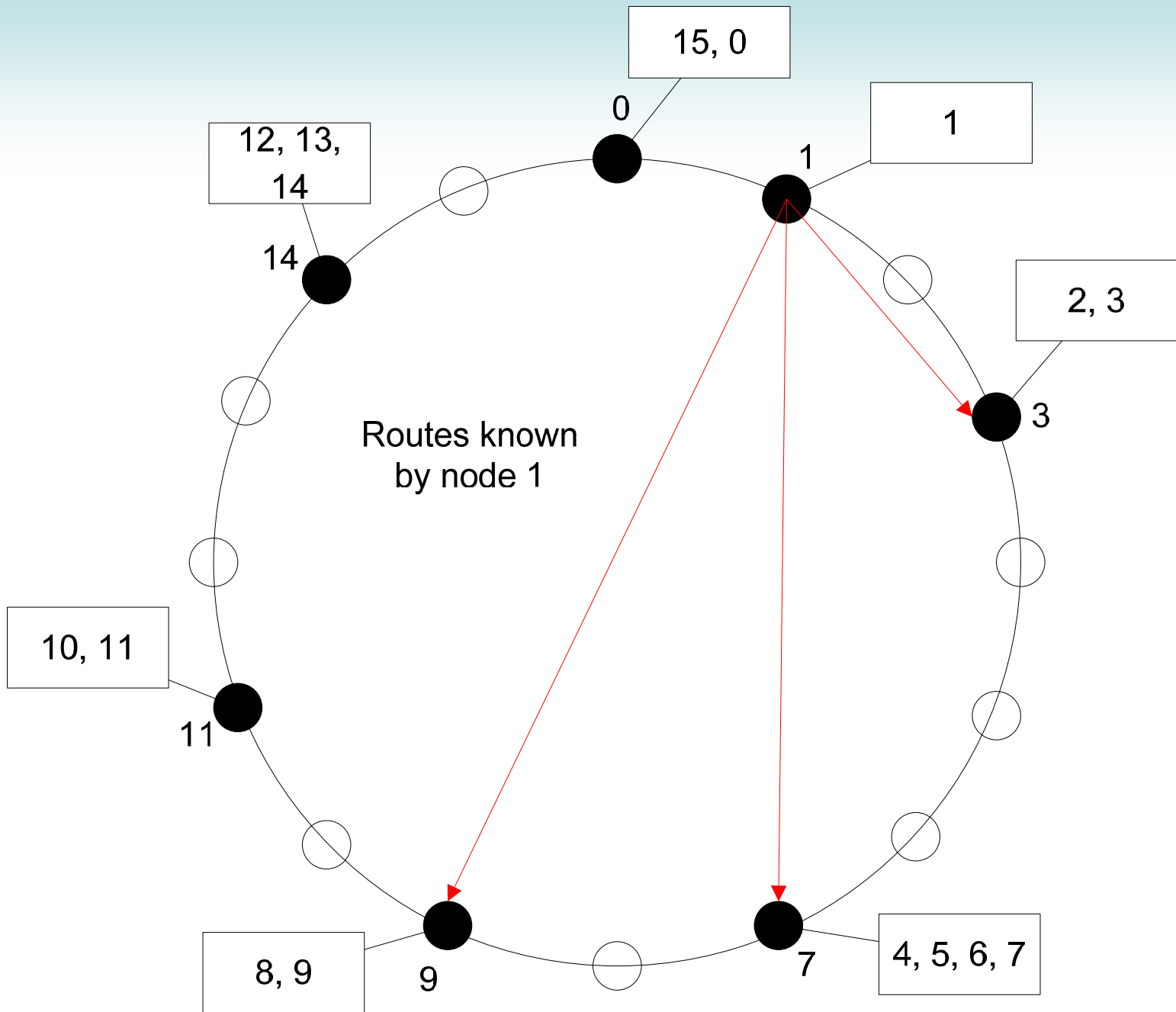


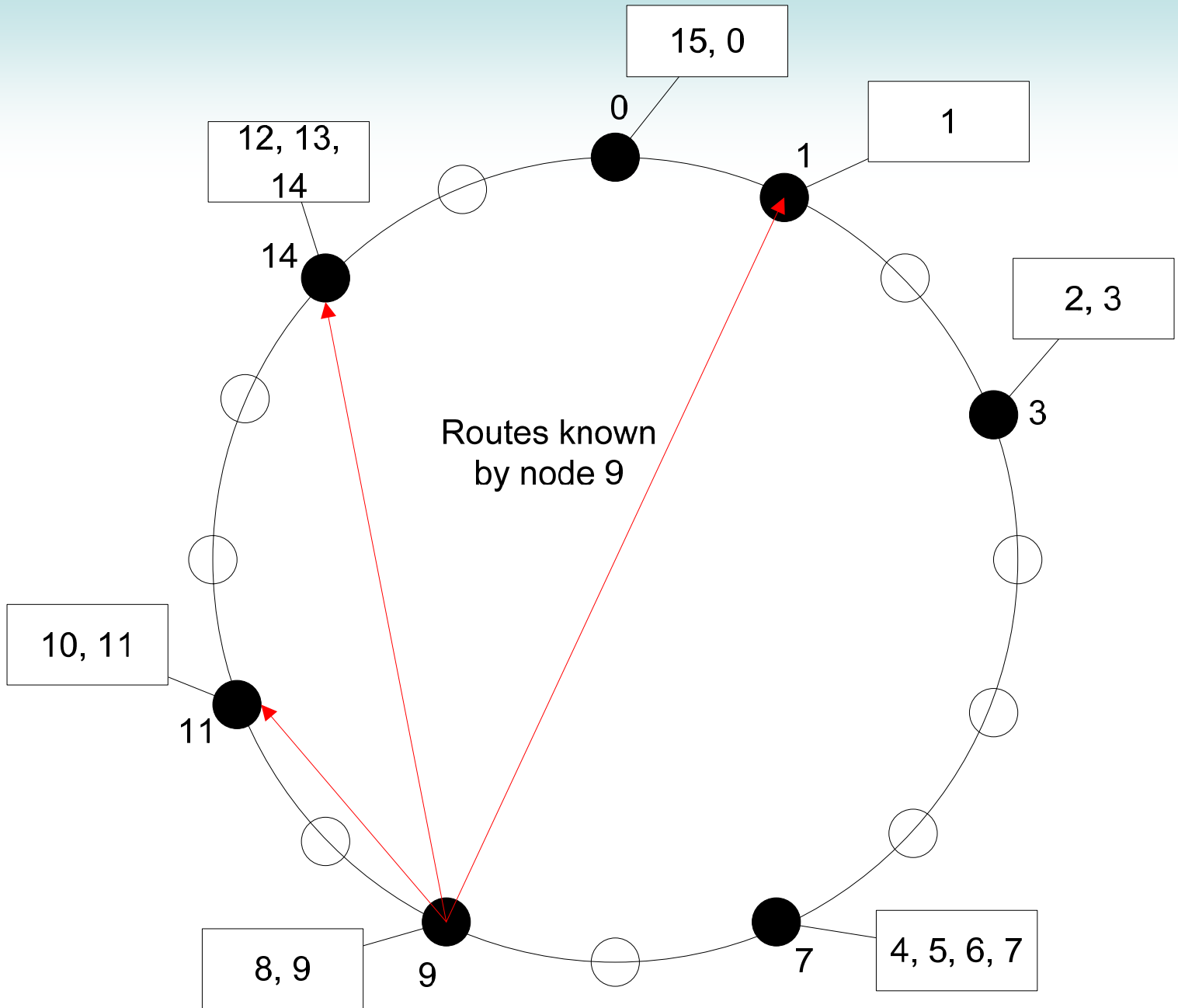


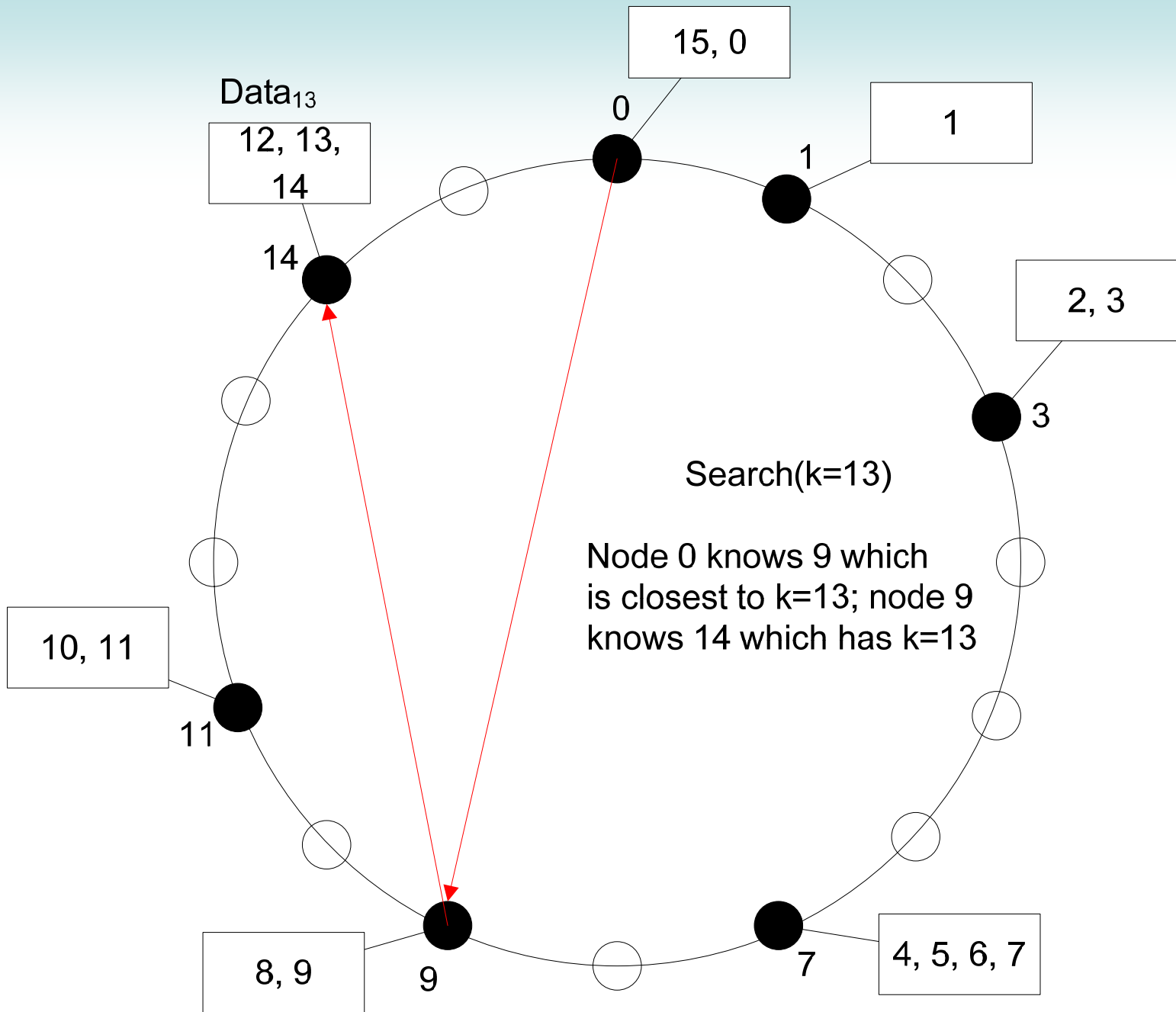












Performance Comparison of P2P Techniques

<i>Approach</i>	<i>Latency</i>	<i>Messages</i>	<i>Update cost</i>	<i>Storage</i>
Unstructured (Gnutella)	$\log(n)$	N	1	1
Directory Server (Napster)	1	1	1	n (max), 1 (avg)
Full replication	1	1	n	n
Super-peers (Fasttrack)	$\log(c)$	C	1	C (max), 1 (avg)
DHT (Chord)	$\log(n)$	$\log(n)$	$\log(n)$	$\log(n)$

n = number of peers; C = number of super-peers

Comparison of P2P Techniques

- Search Capabilities
 - Currently, unstructured and hierarchical P2P systems (e.g. Napster, Gnutella, Fasttrack) support any type of search criteria
 - E.g. a search phrase is handled locally on a peer – it can use traditional database and pattern matching techniques
 - `search(thammasat)` can return all data that contains “thammasat” or related to “thammasat”
 - This is one reason for their popularity, despite lower performance
 - Structured techniques like DHTs are limited by the structure of keys used
 - In basic form, only support equality predicate
 - E.g. `search(k)` will only return data that has exact key k
 - There are techniques in development to enhance structured techniques for better search criteria

Comparison of P2P Techniques

- Replication
 - Many peers in P2P networks are unreliable, offline
 - The data may be replicated in the network to make it more accessible
 - Natural replication occurs in Gnutella, Napster etc. because after peers download a file, they then make it available for others to download
 - Unstructured networks like Chord, Pastry can support or controlled replication of data
 - E.g. `insert(data,k)` in Chord stores copies of the data at multiple nodes

P2P Issues

- Security
 - Most systems have minimal or no security mechanisms
 - Trust and reputation management is needed
 - Need to be able to trust peers to provide accurate results and data
 - Reputation schemes allow peers to gain positive/negative feedback
 - Anonymity versus Identification
 - Anonymity: try to hide who is accessing resources, provide free-speech
 - E.g. Freenet makes it very hard for tracing where data came from; hence hard to legally prove who is storing/distributing illegal content
 - Identification: necessary in commercial systems to manage access and trust
 - Denial of service attacks
 - E.g. False routing information in Chord can make the system useless (cannot find keys)
 - E.g. easy to flood a Gnutella network, severely reducing performance