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: Naptster, 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

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

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

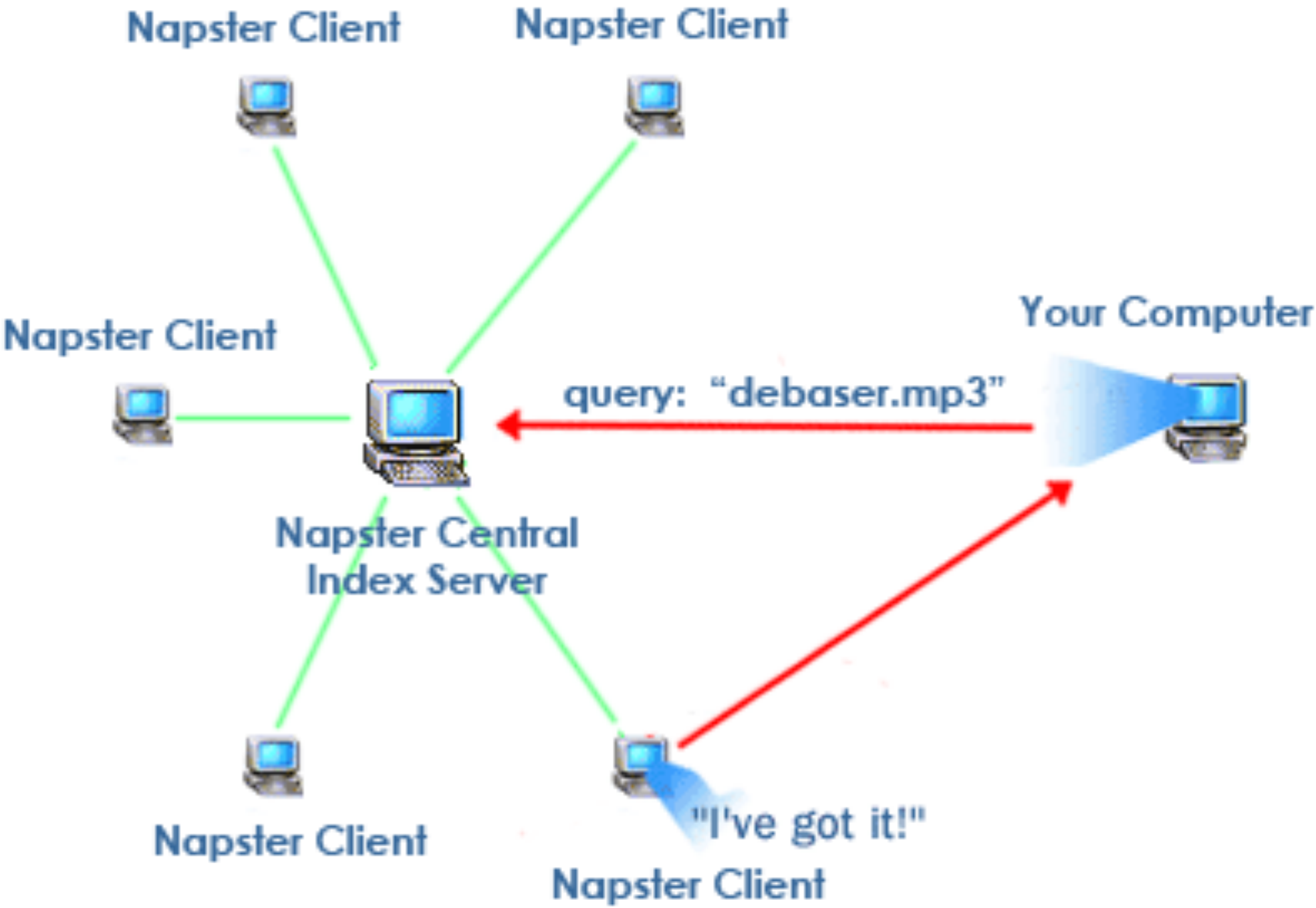
- Managing resources:
  - search(*k*) should return the peers p that contain the resource *r*, where Key(*r*) = *k*
  - insert(*k*,*r*) should add a resource *r* with key *k*
  - delete(*k*) should delete the resource *r*, where Key(*r*) = *k*

- Implementation of Network and Data Management Protocols
  - Differs among different types of P2P systems

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)

Napster Protocol



Gnutella

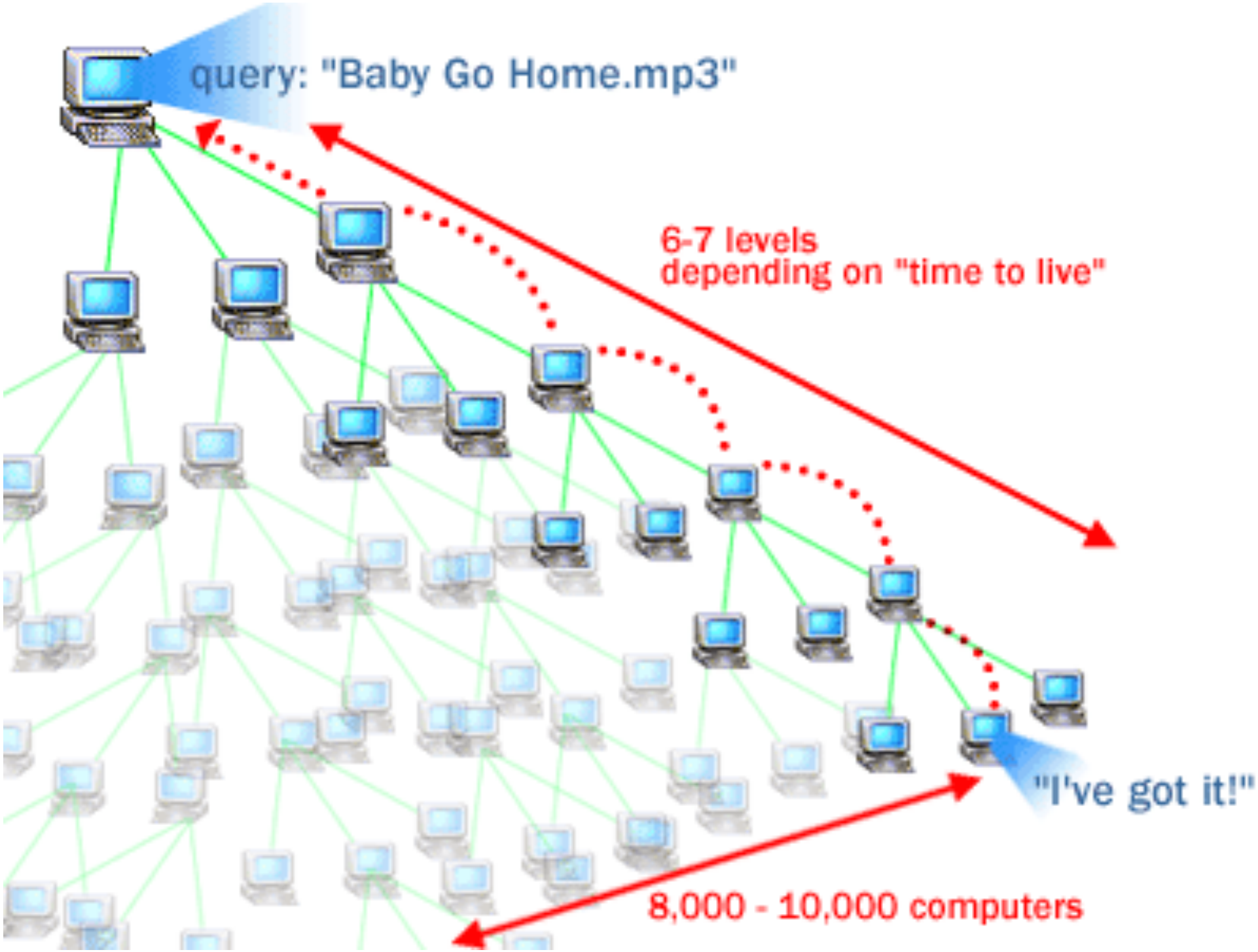
- History:
  - Created by two developers from Nullsoft in 2000
  - Mainly used for exchanging files (originally intended for exchanging recipes)
  - Protocol was reverse engineered from the software binary
- Gnutella is a P2P protocol (not an application)
  - Many client applications implement the Gnutella 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)


query: "Baby Go Home.mp3"
6-7 levels
depending on "time to live"
"I've got it!"
8,000 - 10,000 computers

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 overheard, 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

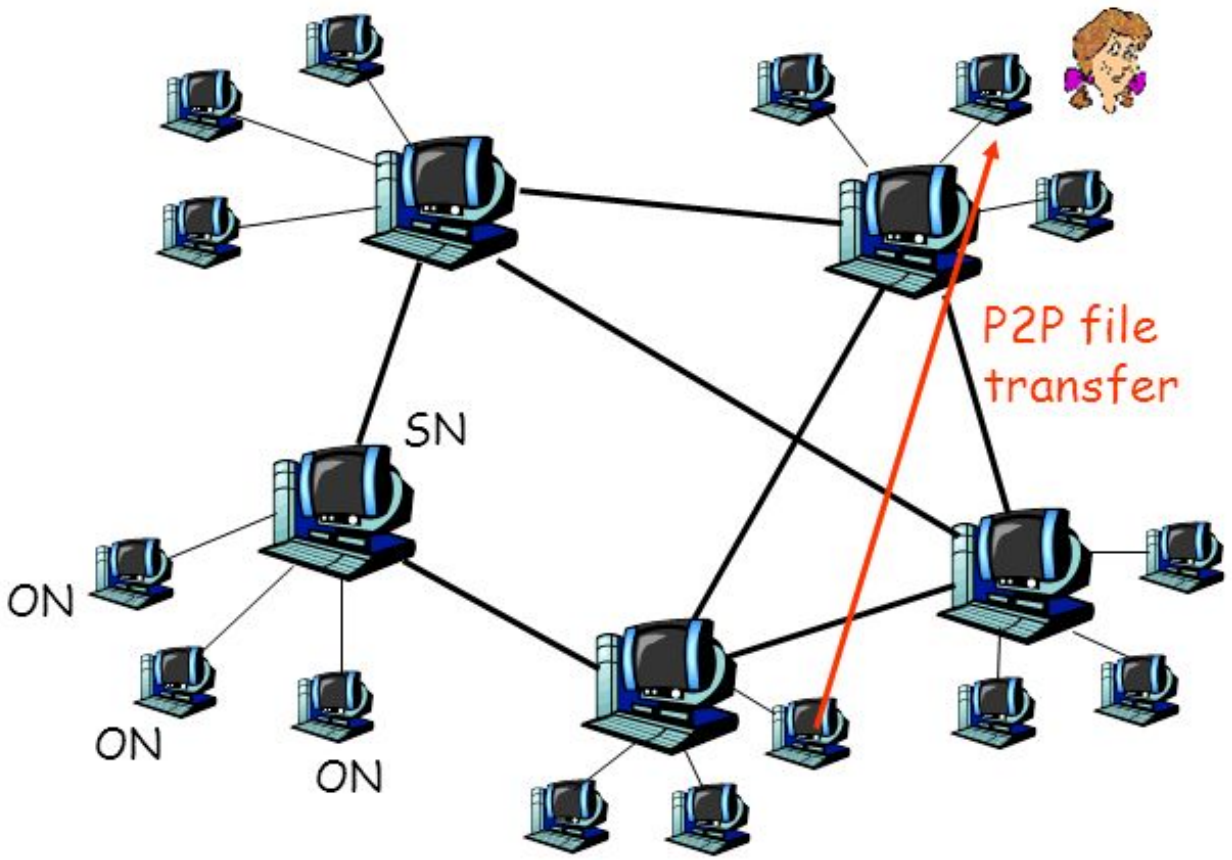
Super-Peer Architecture

- 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

FastTrack

ON =
ordinary node

SN =
super node



Performance Comparison of P2P Techniques

Approach	*Latency*	*Messages*	*Update cost*	*Storage*
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	N/C (max), 1 (avg)
DHT (Chord)	log(*n*)	log(*n*)	log(*n*)	log(*n*)

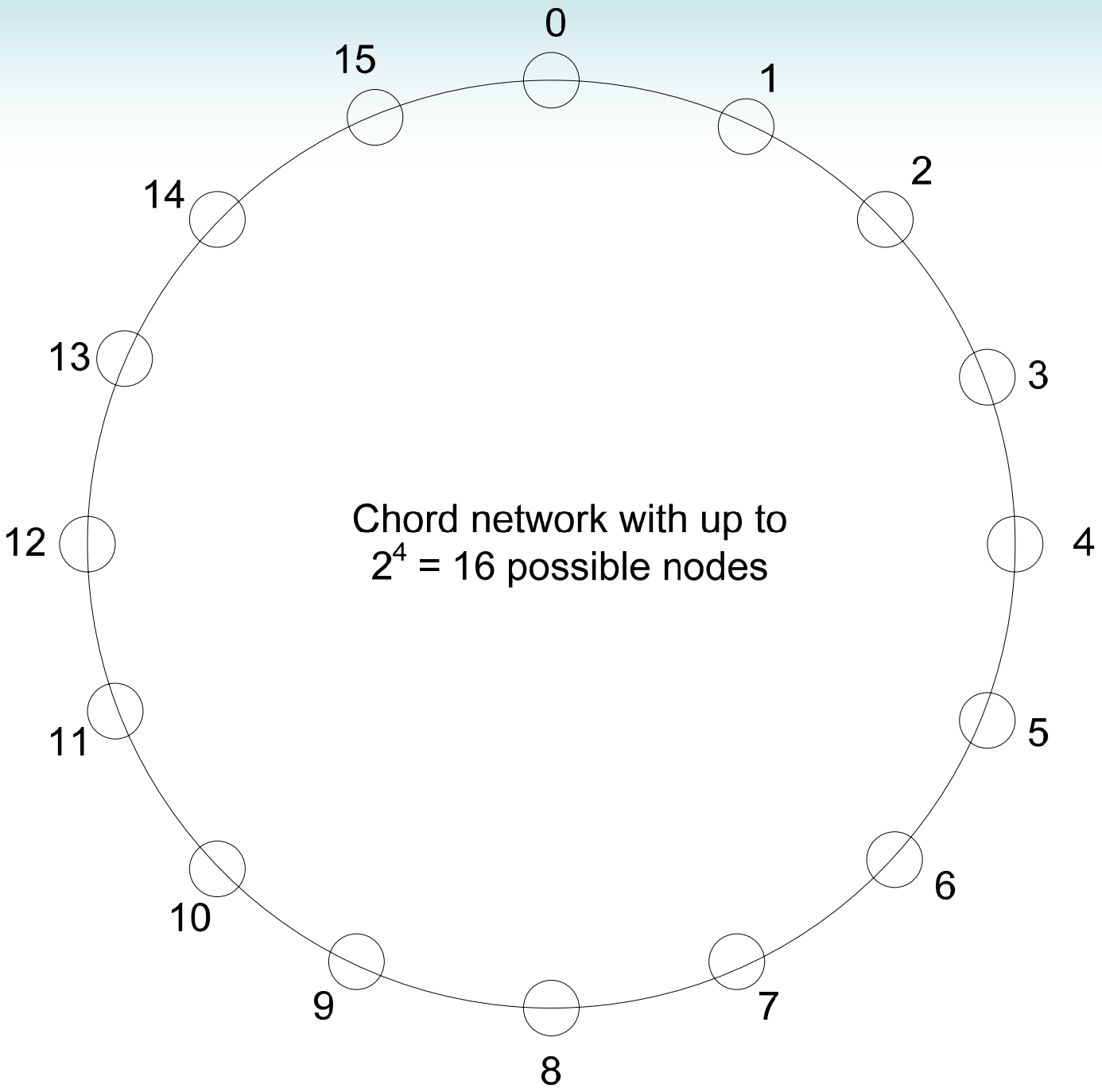
n = number of peers; *C* = number of super-peers

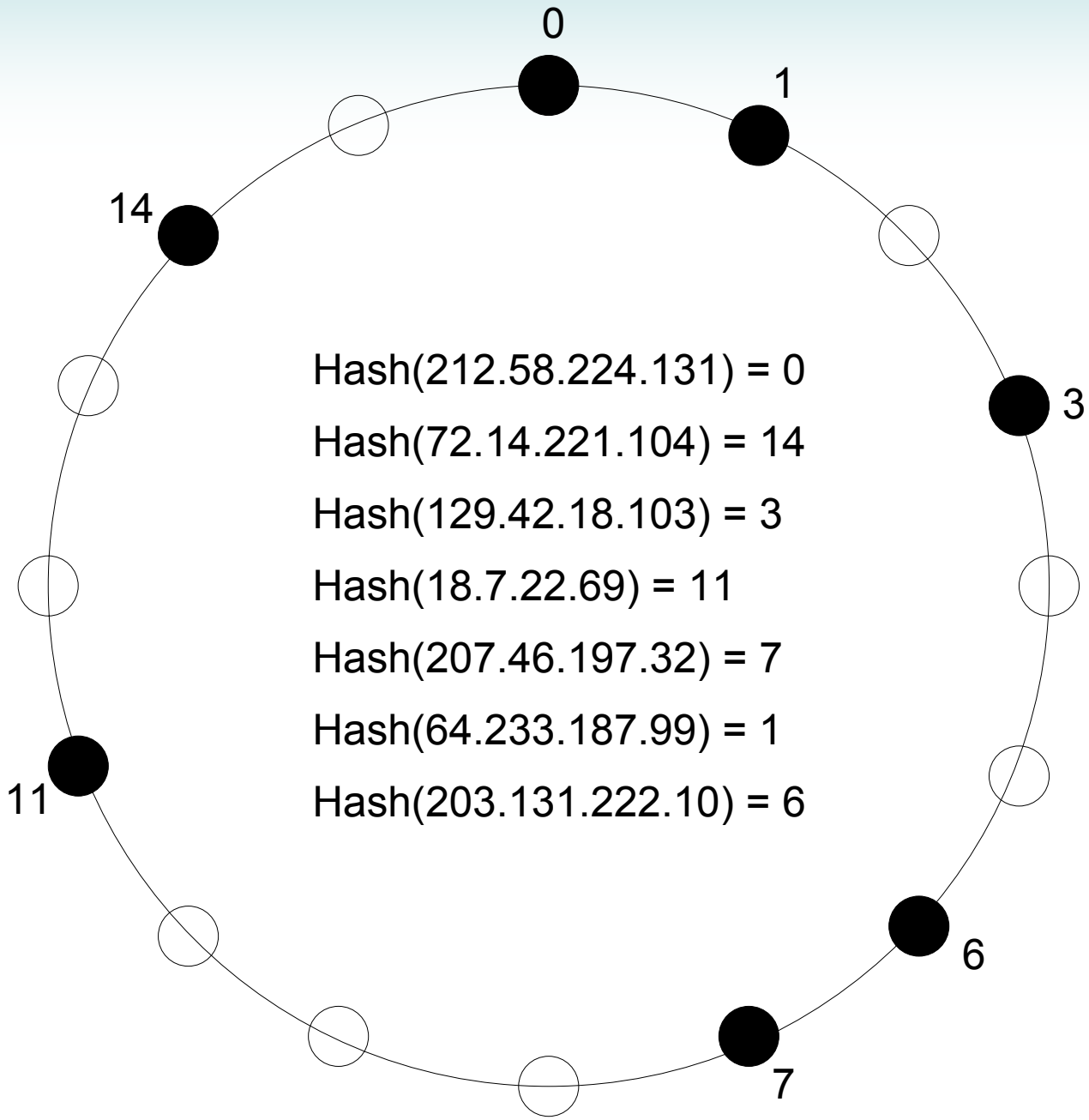
Distributed Hash Tables

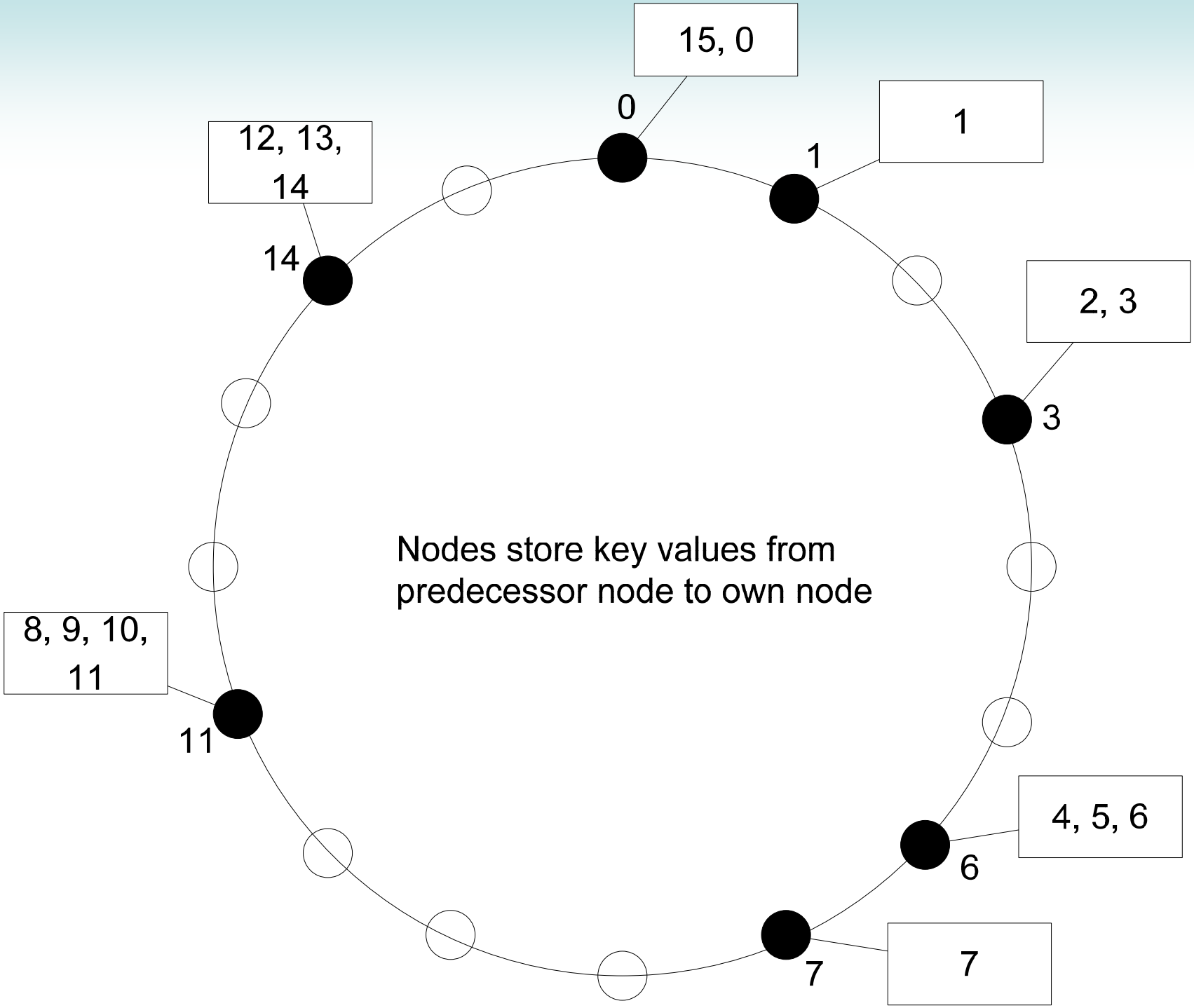
- History
  - Motivated by disadvantages of Napster (centralised), Gnutella (inefficient) and similar P2P protocols for file sharing
  - Research has been used for file sharing, instant messaging, distributed file systems, web caching and other fields
- DHTs are P2P algorithms (not protocol or application)
  - 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*)

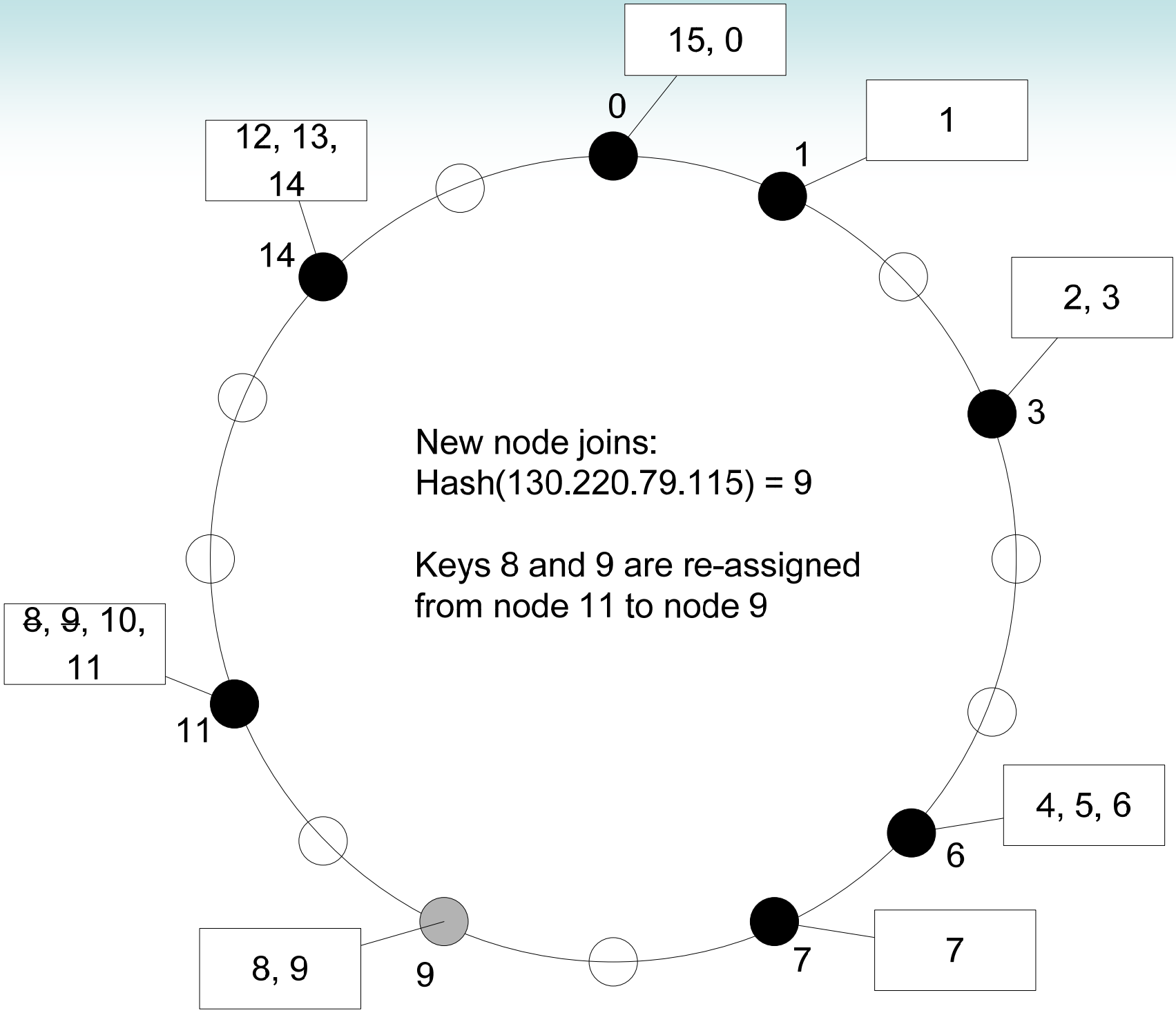

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Chord network with up to
2^4 = 16 possible nodes

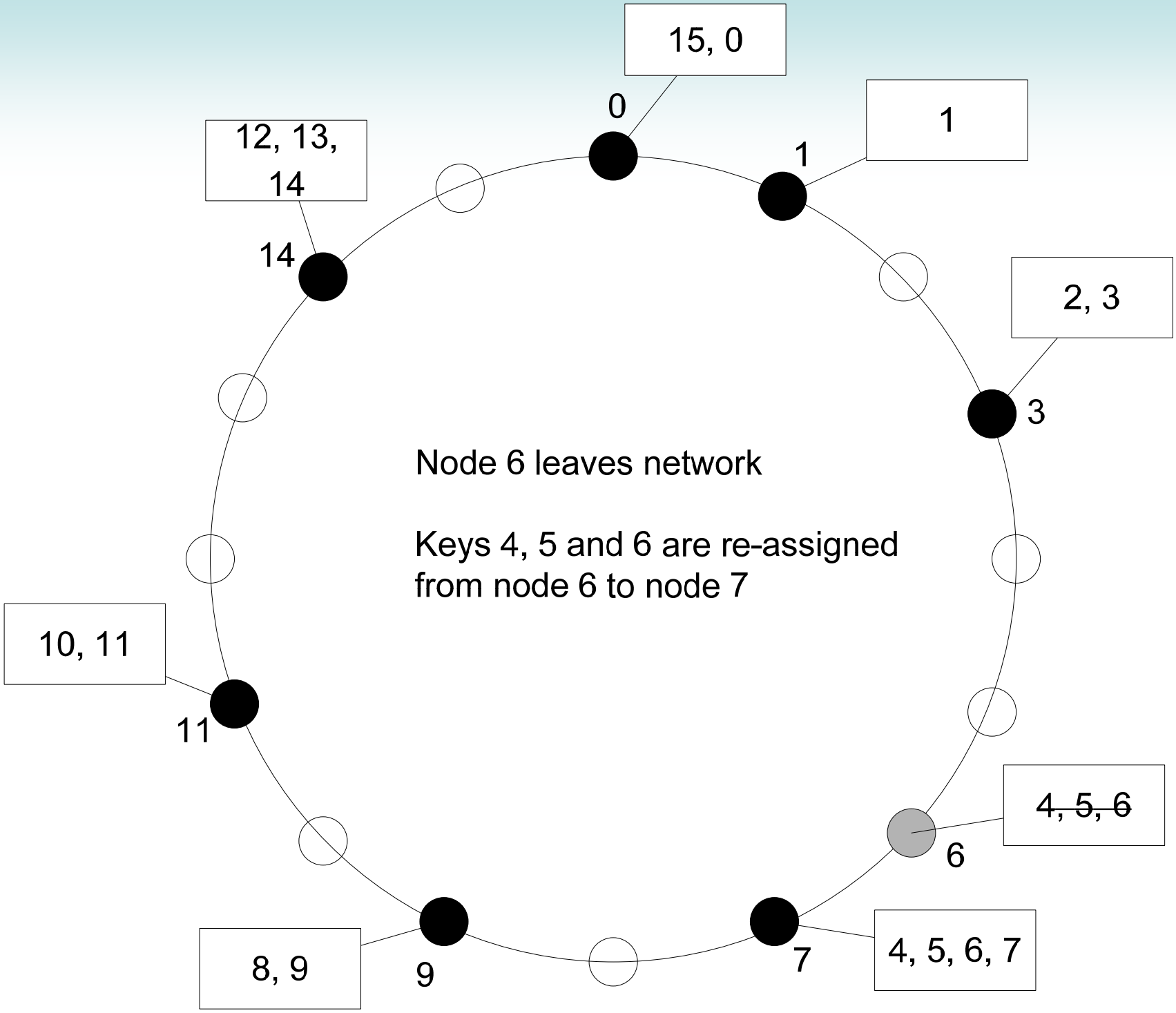

0
1
3
6
7
11
14
Hash(212.58.224.131) = 0
Hash(72.14.221.104) = 14
Hash(129.42.18.103) = 3
Hash(18.7.22.69) = 11
Hash(207.46.197.32) = 7
Hash(64.233.187.99) = 1
Hash(203.131.222.10) = 6


15, 0
0
1
1
12, 13, 14
14
2, 3
3
Nodes store key values from predecessor node to own node
8, 9, 10, 11
11
4, 5, 6
6
7
7

Chord: Inserting and Searching

- To insert data, 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, 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 $node_k$


15, 0
0
1
1
12, 13, 14
14
2, 3
3
New node joins:
Hash(130.220.79.115) = 9
Keys 8 and 9 are re-assigned from node 11 to node 9
8, 9, 10, 11
11
4, 5, 6
6
8, 9
9
7
7


15, 0
0
1
1
12, 13, 14
14
2, 3
3
Node 6 leaves network
Keys 4, 5 and 6 are re-assigned from node 6 to node 7
10, 11
11
4, 5, 6
6
8, 9
9
7
4, 5, 6, 7

Chord: Routing

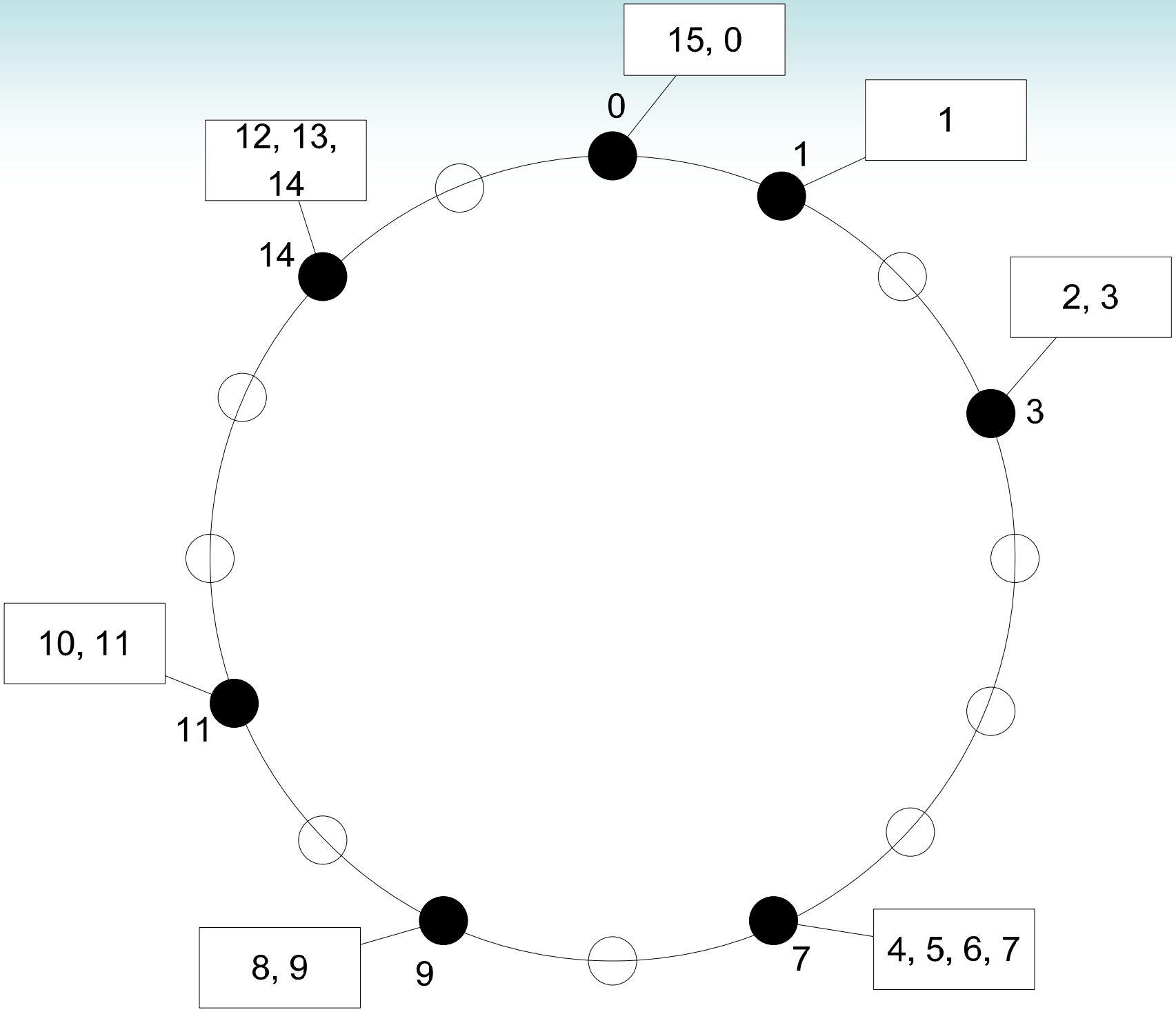
- Simple Routing
  - Each node *n* maintains route to successor(*n*)
    - Node *n* knows the IP address/port number of successor(*n*)
  - A simple (but naïve) approach is to then try to find key by checking each subsequent successor
    - Example: node 0 knows the IP address/port of node 1; node 1 knows 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*

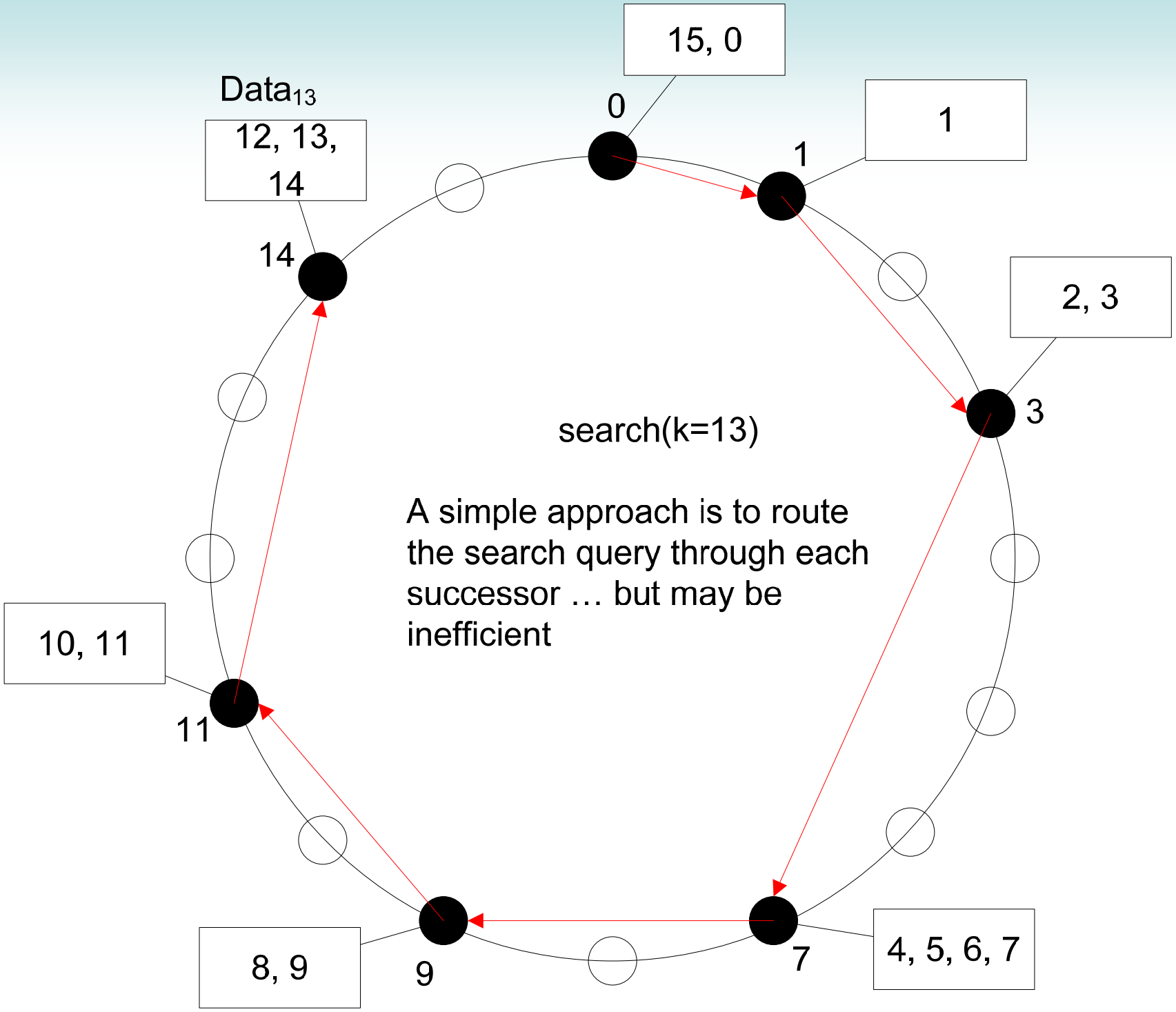
Chord: Routing Example

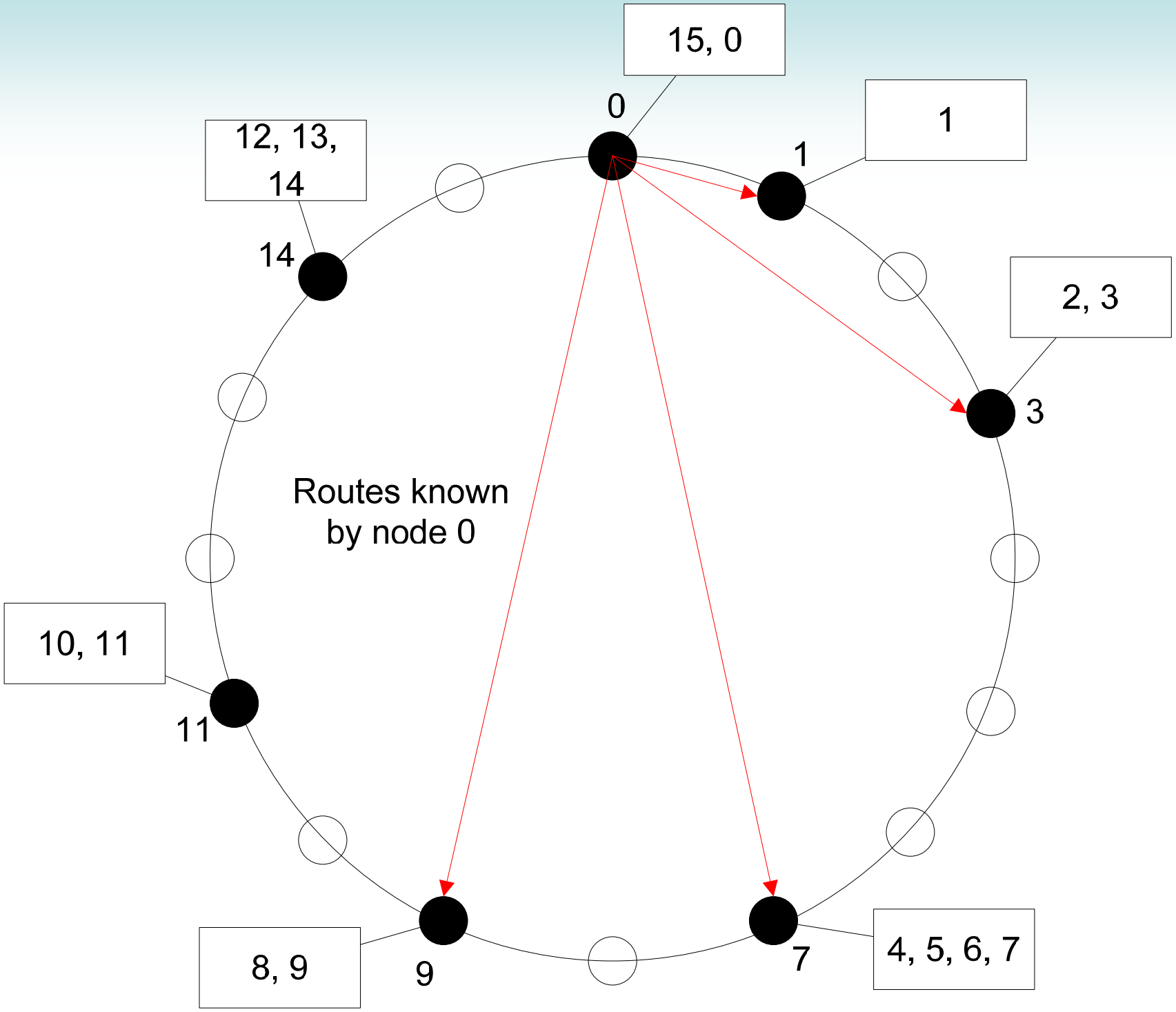
- Node 0 routing (or finger) table:
  - Start = 1; Interval = 1 → 1; Successor = 1
  - Start = 2; Interval = 2 → 3; Successor = 3
  - Start = 4; Interval = 4 → 7; Successor = 7
  - Start = 8; Interval = 8 → 15; Successor = 9
- (The 3rd line can be read as: “in order to find a node with key 4, 5, 6 or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

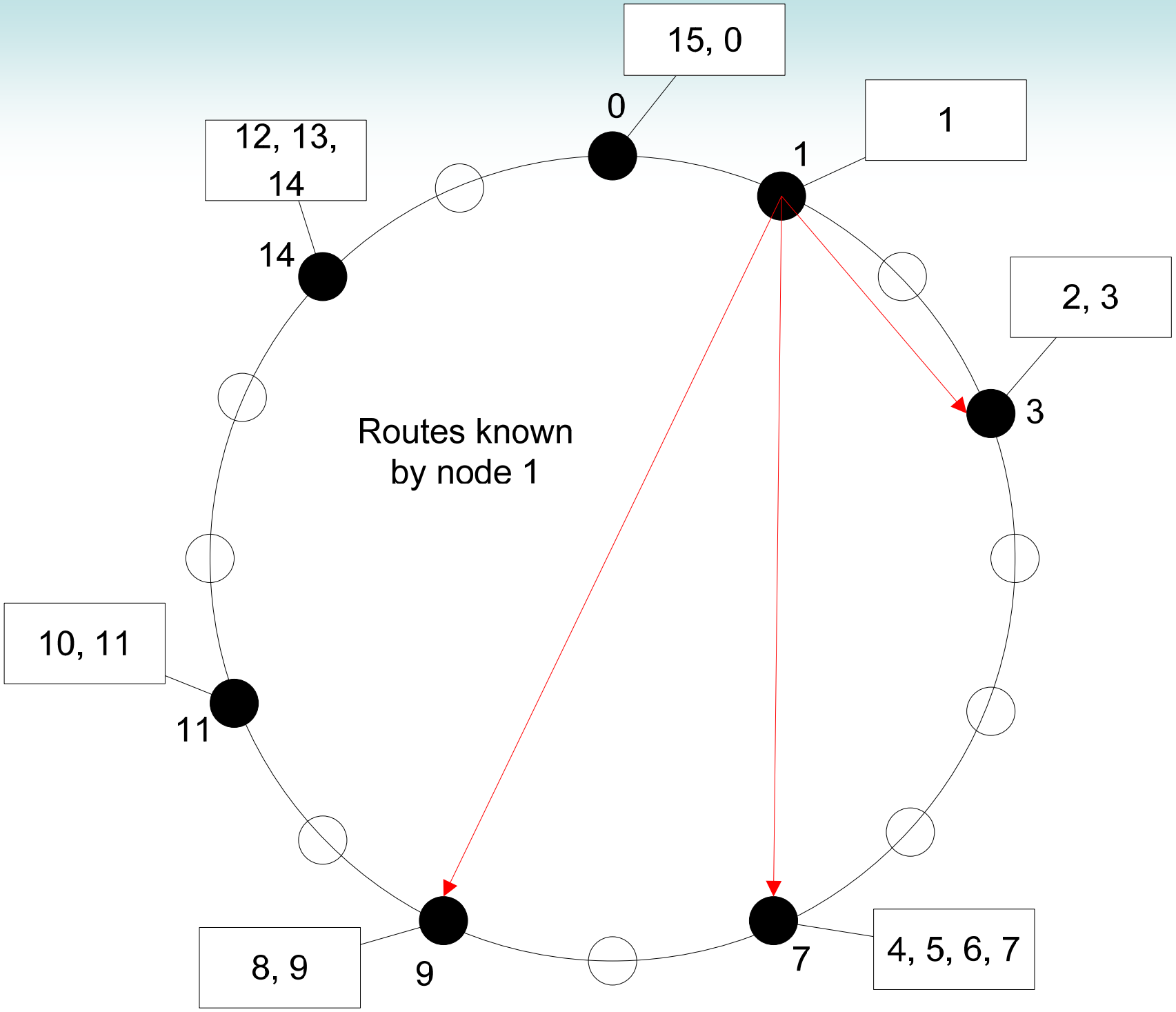
Chord: Routing Example

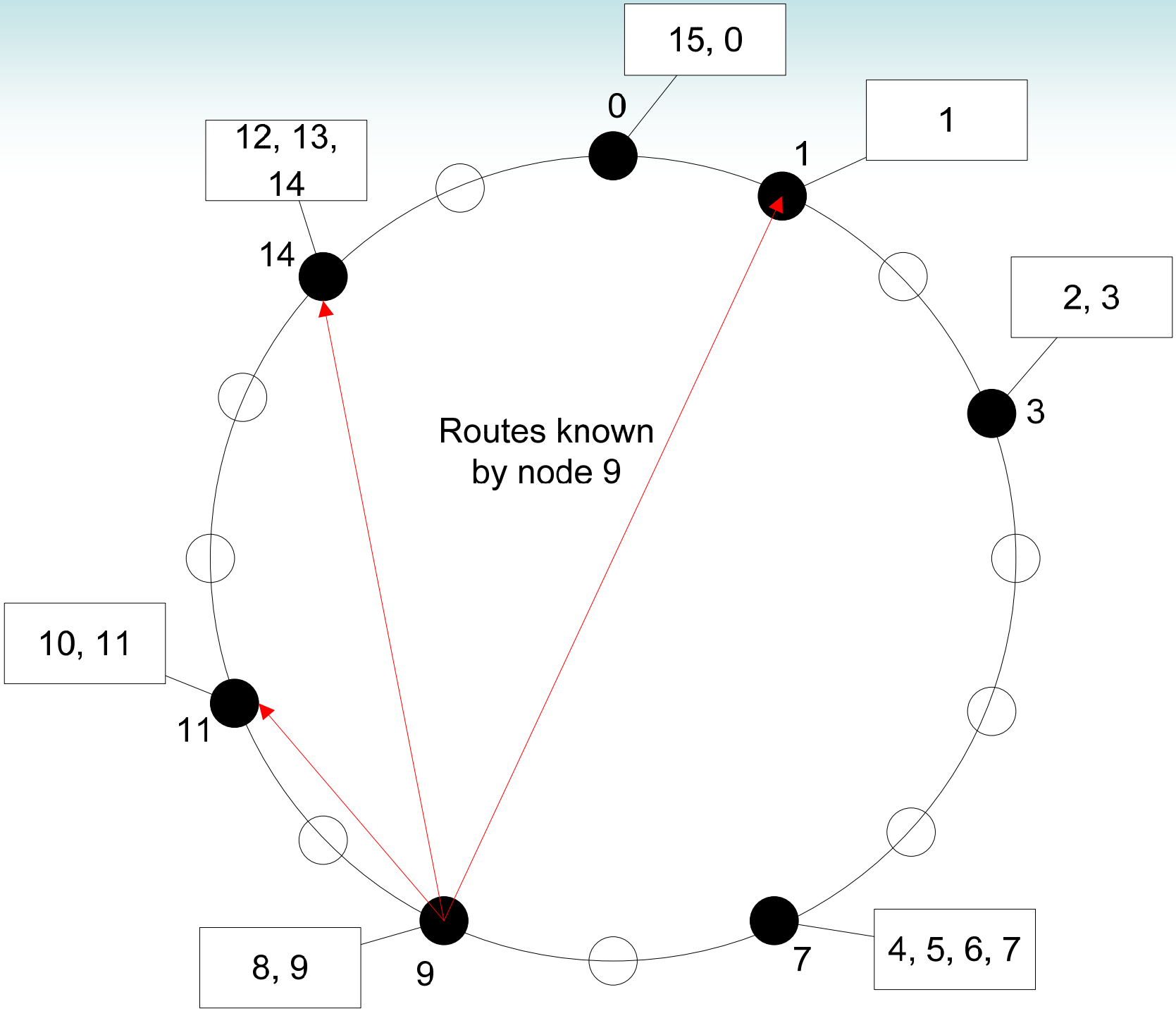
- Assume a search(k=13) is performed at node 0
  - Node 0 knows node 9 has coverage of the keys from 8 to 15
  - Node 0 sends a search message to node 9
  - Node 9 does not have the data with key 13
  - Node 9 knows node 14 has coverage of the keys from 13 to 0
  - Node 9 sends a search message to node 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

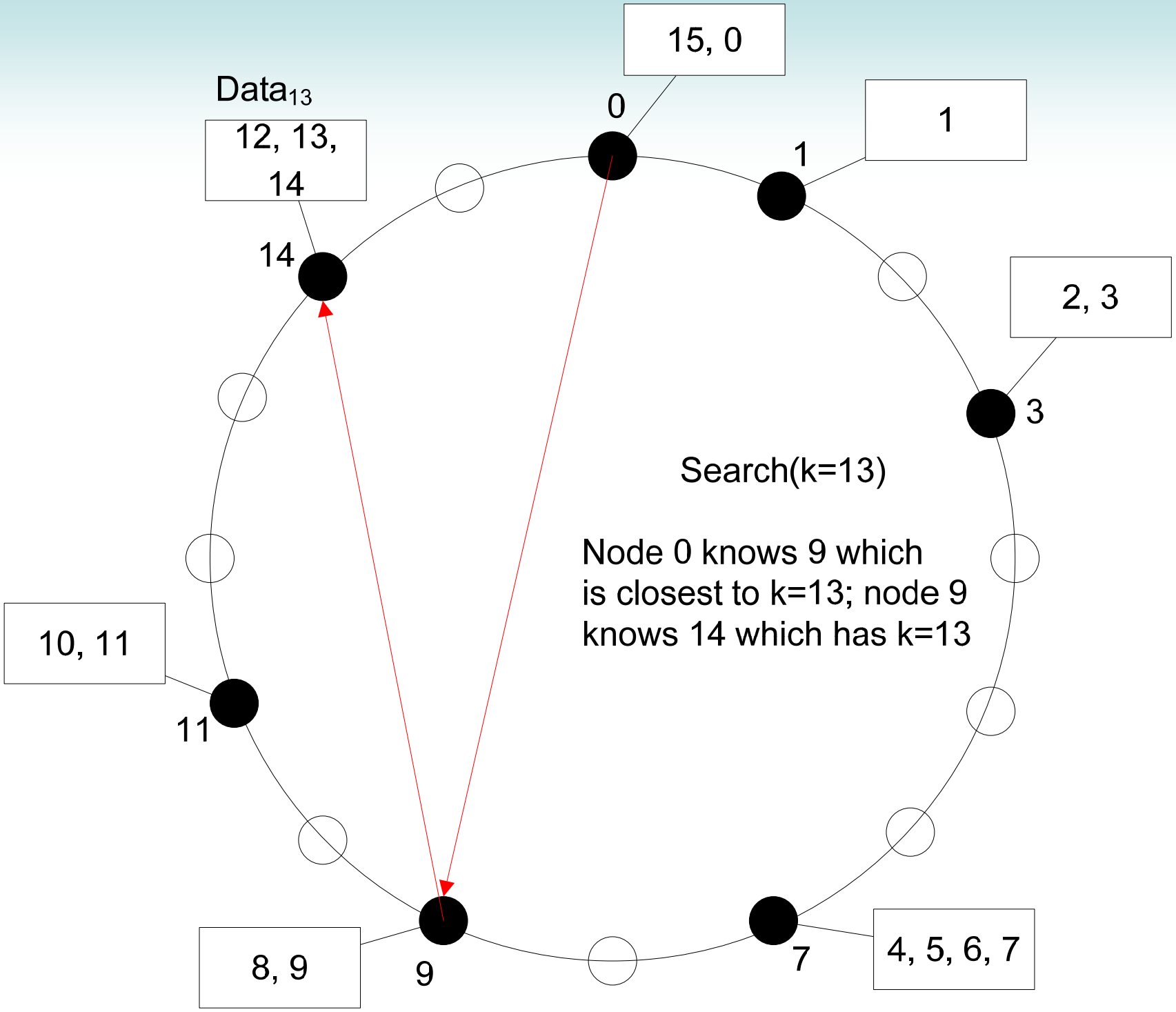

15, 0
0
1
1
12, 13,
14
14
2, 3
3
10, 11
11
8, 9
9
7
4, 5, 6, 7


15, 0
Data13
12, 13, 14
0
1
1
14
2, 3
3
search(k=13)
A simple approach is to route the search query through each successor … but may be inefficient
10, 11
11
8, 9
9
7
4, 5, 6, 7


15, 0
0
1
1
12, 13,
14
14
2, 3
3
Routes known
by node 0
10, 11
11
8, 9
9
7
4, 5, 6, 7


15, 0
0
1
1
12, 13,
14
14
2, 3
3
Routes known
by node 1
10, 11
11
8, 9
9
7
4, 5, 6, 7


15, 0
0
1
1
12, 13,
14
14
2, 3
3
Routes known
by node 9
10, 11
11
8, 9
9
7
4, 5, 6, 7


15, 0
Data13
12, 13, 14
0
1
1
14
2, 3
3
Search(k=13)
Node 0 knows 9 which is closest to k=13; node 9 knows 14 which has k=13
10, 11
11
8, 9
9
7
4, 5, 6, 7

Pastry

- Routing overlay with 128-bit GUIDs
- GUID computed by a secure hash function such as SHA-1 (see Chapter 11) using public key of node
- Typically hash function is applied to the object name (or another known part of the object state)
- $0 <= GUID <= 2^{128} - 1$

Routing performance:

- Order of log N steps when N nodes participate in system
- Routing overlay built over UDP
- Network distance between nodes based on hop counts and round trip latency – used to set up routing tables at each node

Pastry

Routing algorithm stage 1:

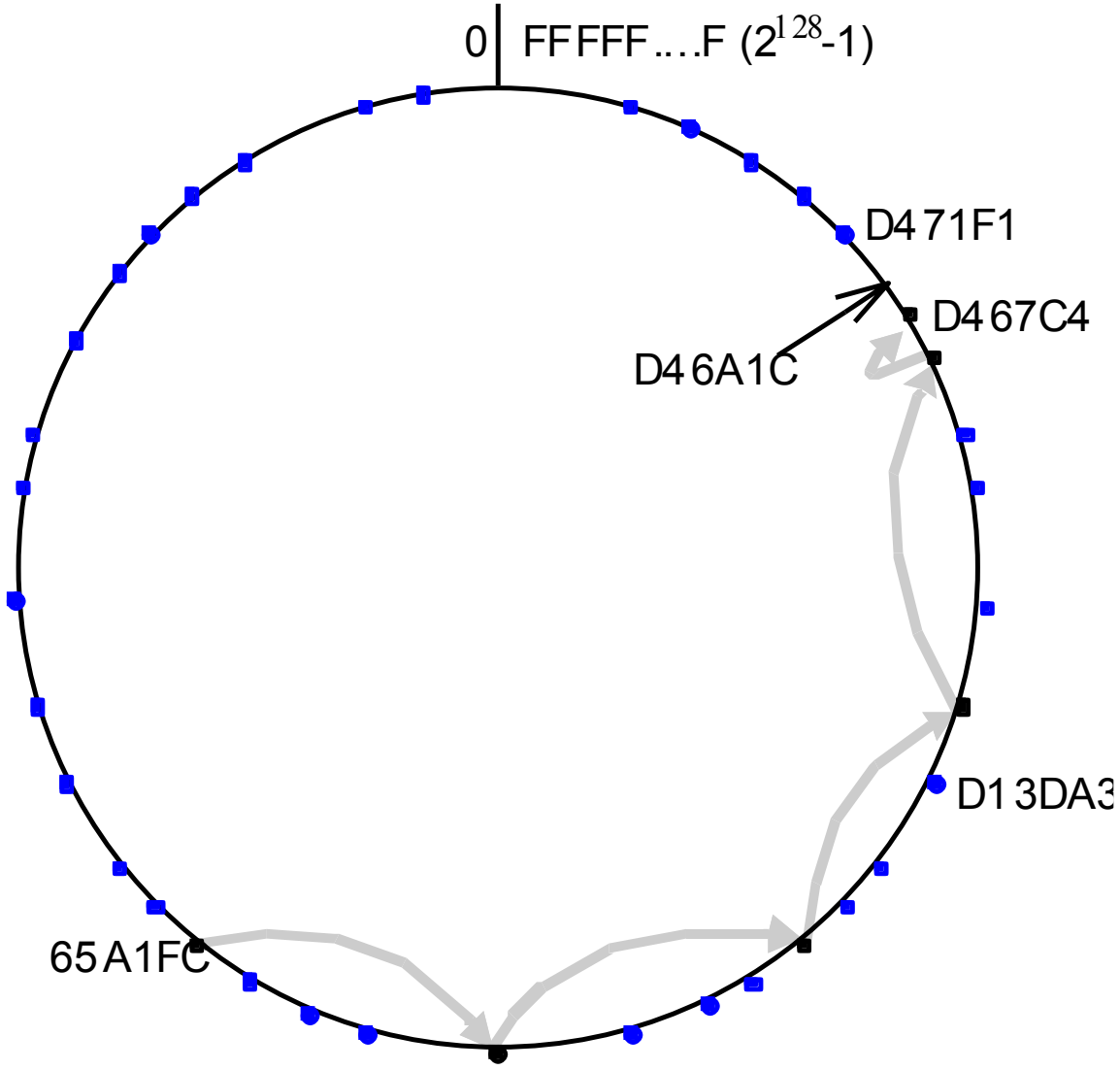
- Each node keeps a *leaf set* contains nodes closest to current node
- GUID space can be visualized as a circle
- If node A receives a routing request for node D, it finds closest node in its leaf set to D to forward the message

Routing algorithm stage 2:

- Each node keeps a *tree-structured* routing table
- Entries include (GUID, IP address) of a set of nodes

Circular routing alone is correct but inefficient

Based on Rowstron and Druschel [2001]



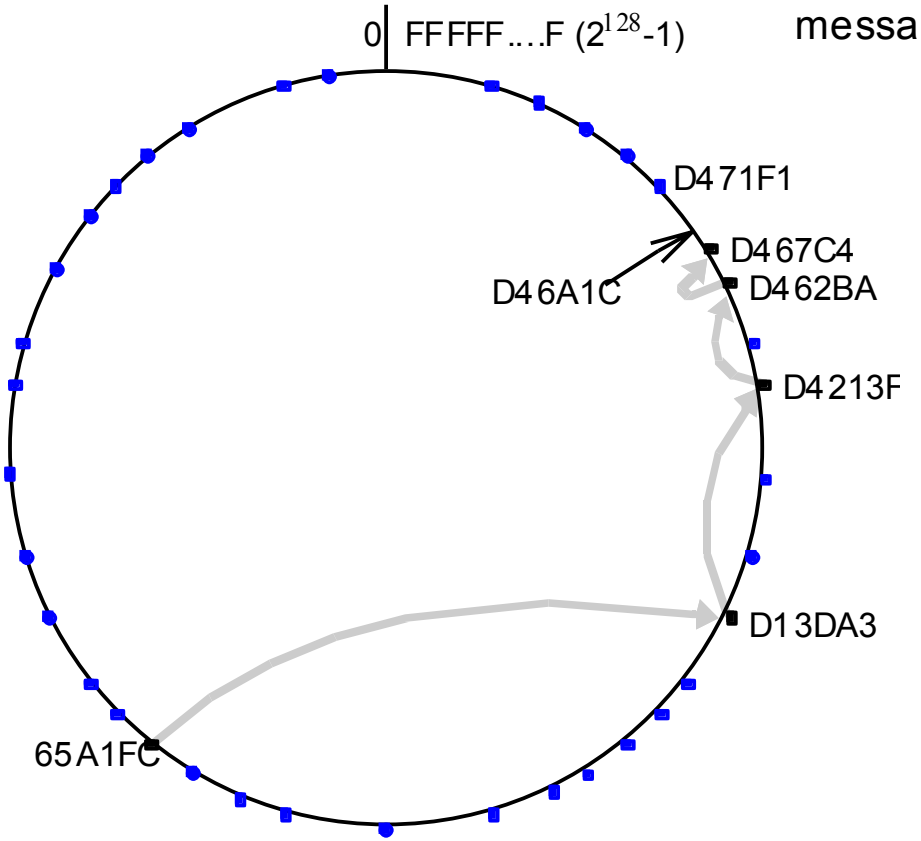
The dots depict live nodes. The space is considered as circular: node 0 is adjacent to node (2128-1). The diagram illustrates the routing of a message from node 65A1FC to D46A1C using leaf set information alone, assuming leaf sets of size 8 (l = 4). This is a degenerate type of routing that would scale very poorly; it is not used in practice.

First four rows of a Pastry routing table

p =	GUID prefixes and corresponding nodehandles n															
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	n	n	n	n	n	n		n	n	n	n	n	n	n	n	n
1	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6F	6E	6F
	n	n	n	n	n		n	n	n	n	n	n	n	n	n	n
2	650	651	652	653	654	655	656	657	658	659	65A	65B	65C	65D	65E	65F
	n	n	n	n	n	n	n	n	n	n		n	n	n	n	n
3	65A0	65A1	65A2	65A3	65A4	65A5	65A6	65A7	65A8	65A9	65AA	65AB	65AC	65AD	65AE	65AF
	n		n	n	n	n	n	n	n	n	n	n	n	n	n	n

The routing table is located at a node whose GUID begins 65A1. Digits are in hexadecimal. The *n*'s represent [GUID, IP address] pairs specifying the next hop to be taken by messages addressed to GUIDs that match each given prefix. Grey- shaded entries indicate that the prefix matches the current GUID up to the given value of *p*: the next row down or the leaf set should be examined to find a route. Although there are a maximum of 128 rows in the table, only log16 *N* rows will be populated on average in a network with *N* active nodes.

Pastry routing example Based on Rowstron and Druschel [2001]



Routing a message from node 65A1FC to D46A1C. With the aid of a well-populated routing table the message can be delivered in ~ $\log_{16}(N)$ hops.

Pastry's routing algorithm

To handle a message M addressed to a node D (where $R[p,i]$ is the element at column i, row p of the routing table):

1. If ($L_{-l} < D < L_l$) { // the destination is within the leaf set or is the current node.
2. Forward M to the element L_i of the leaf set with GUID closest to D or the current node A.
3. } else { // use the routing table to despatch M to a node with a closer GUID
4. find p, the length of the longest common prefix of D and A. and i, the $(p+1)^{th}$ hexadecimal digit of D.
5. If ($R[p,i] \neq null$) forward M to $R[p,i]$ // route M to a node with a longer common prefix.
6. else { // there is no entry in the routing table
7. Forward M to any node in L or R with a common prefix of length i, but a GUID that is numerically closer.

}

}

Performance Comparison of P2P Techniques

Approach	*Latency*	*Messages*	*Update cost*	*Storage*
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
  - structured 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

BitTorrent

Terminology and Concepts

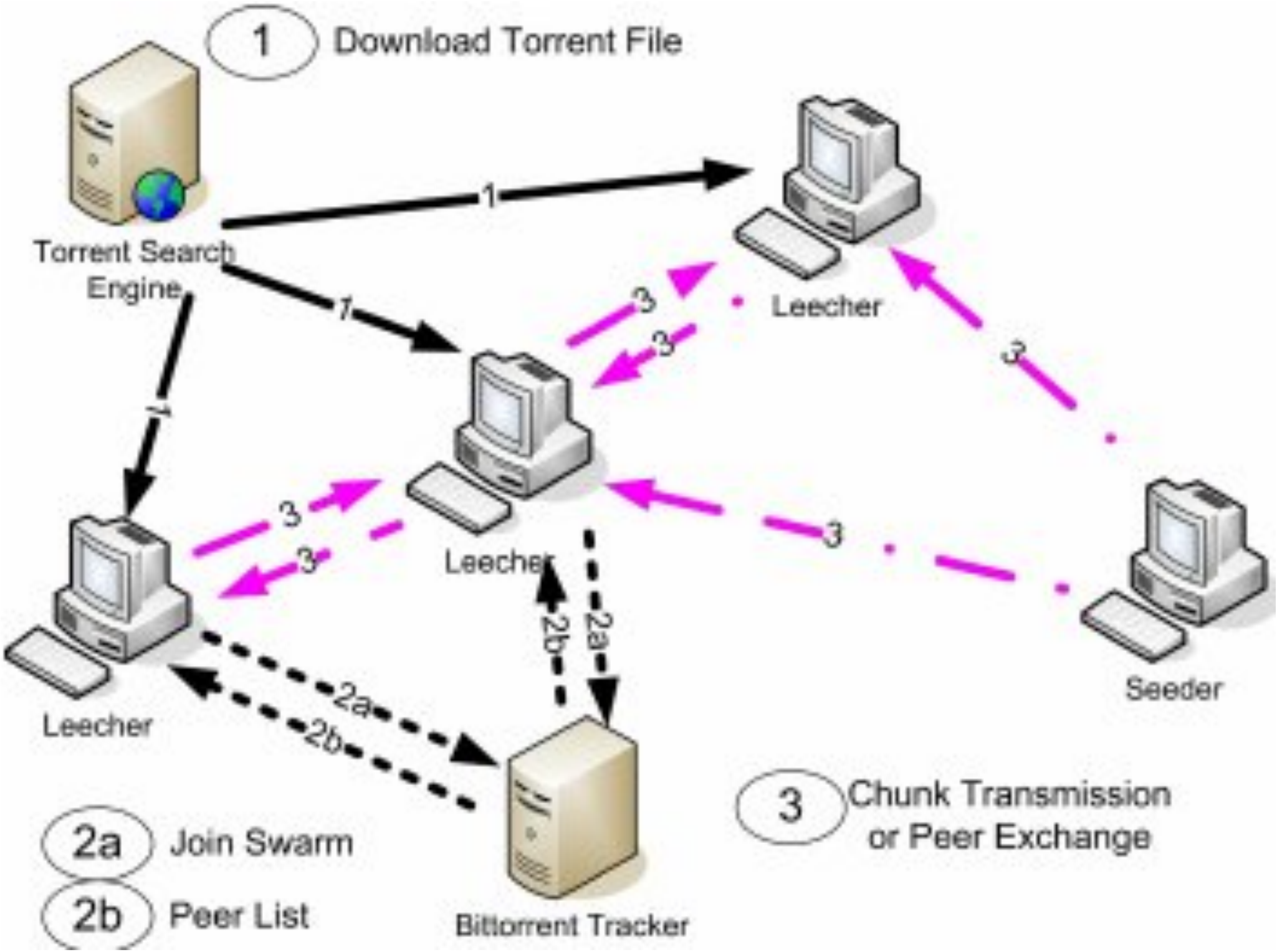
- A file (or set of files) is referred to as a *torrent*
  - A *.torrent* file is a specific file that describes the torrent
- Users accessing a torrent are called *peers*
  - Downloading and/or uploading
- A peer that has downloaded the entire torrent is a *seed*
  - That user is *seeding* the torrent
- A peer that has not yet downloaded the entire torrent is a *downloader* or a *leech*
  - Normally a leecher has downloaded more than they have uploaded
- A *swarm* is the set of users connected to a torrent
  - All the peers, including seeds

Terminology and Concepts

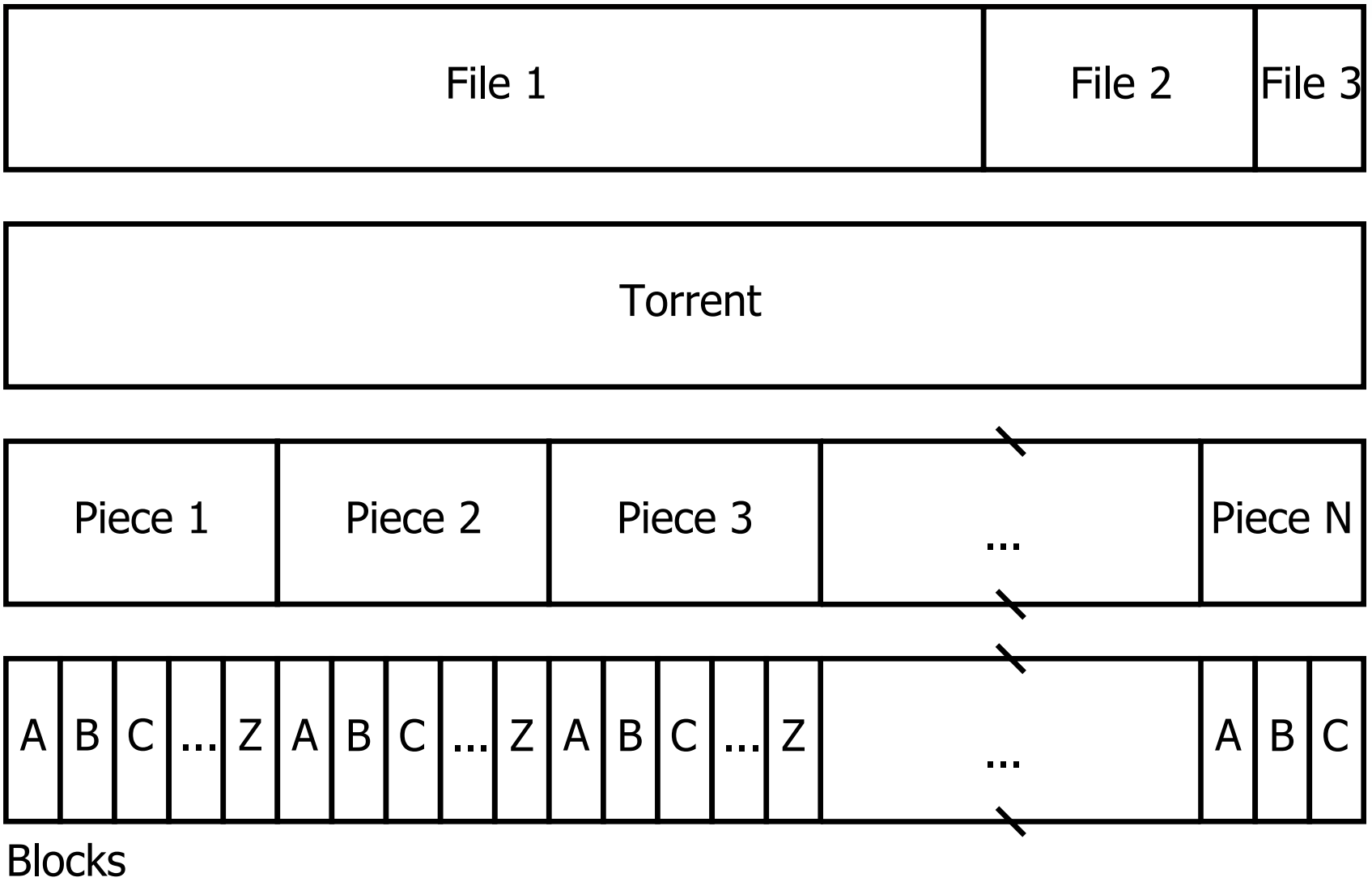
- Files are divided into *pieces*
  - When a piece is downloaded by a peer, it is then made available for upload
- Pieces are divided into *blocks*

- A *tracker* is a server that manages the list of peers in a swarm
  - Keeps a list of peer IP addresses and port numbers

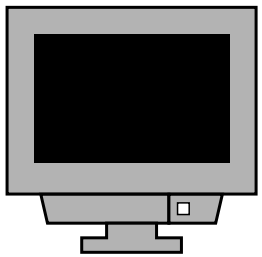
- An *indexer* is a server that manages a list of torrents
  - Normally a website that allows search for .torrent files by name and other descriptive data
  - A .torrent file also lists the address of the tracker


1 Download Torrent File
Torrent Search Engine
1
1
1
Leecher
3
3
3
Leecher
3
3
Leecher
2a
2b
2b
2a
Bittorrent Tracker
Seeder
2a Join Swarm
2b Peer List
3 Chunk Transmission or Peer Exchange

Files, Torrent, Pieces and Blocks

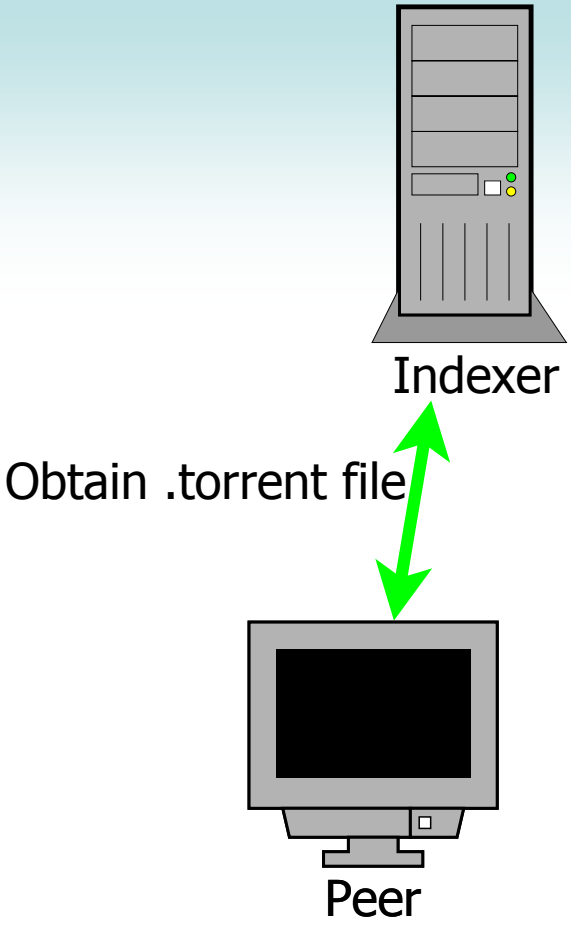


Steps using BitTorrent



A peer is a BitTorrent client software running on a computer. It can be identified by the IP address of the computer and the applications port number.

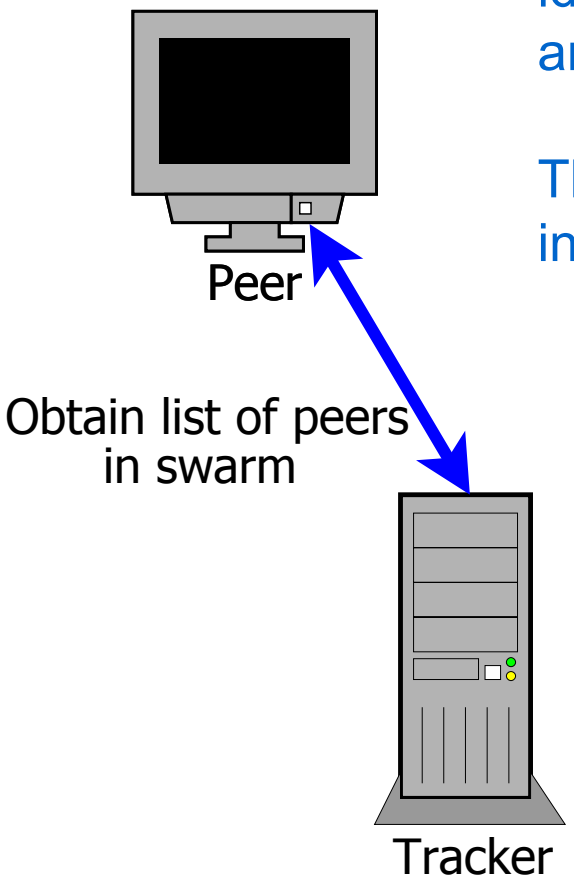
Initially the peer does not know any other peers.



To start downloading a file, the corresponding .torrent file is needed.

The peer contacts an indexer, which has a list of known torrents.

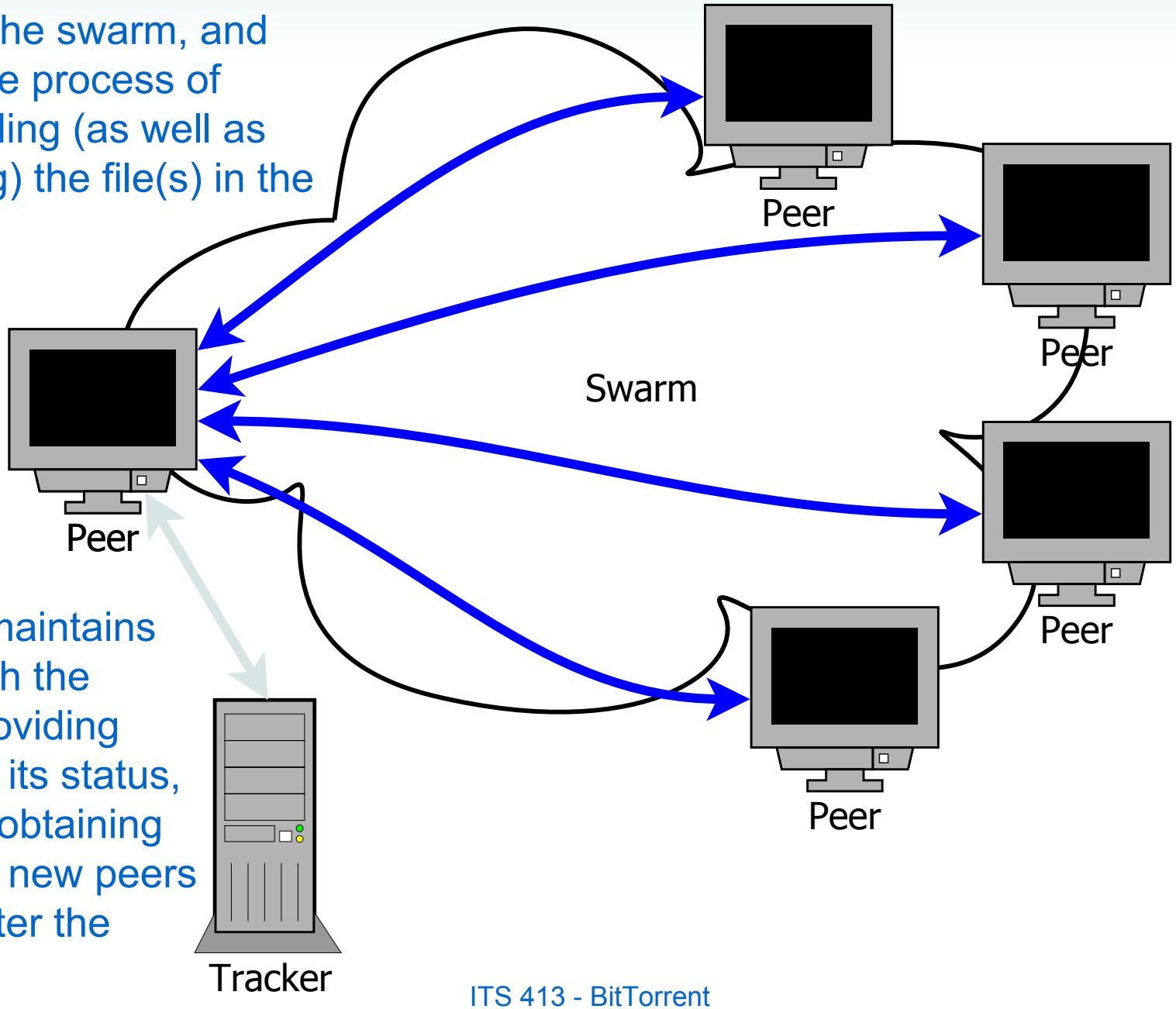
The process of obtaining the .torrent file is *not* part of the BitTorrent protocol. Normally it is performed using a web browser contacting a web server (the index). The .torrent file may be downloaded using HTTP, and then loaded in the BitTorrent client software.



The .torrent file describes the file(s), as well as identifies the tracker for the torrent (by an IP address and port number)

The peer contacts the tracker to obtain the list of peers in the swarm for the torrent.

The peer contacts other peers in the swarm, and begins the process of downloading (as well as uploading) the file(s) in the torrent.



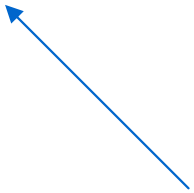
The peer maintains contact with the tracker, providing updates of its status, as well as obtaining address of new peers as they enter the swarm.

.torrent File

- The .torrent file describes the contents of the files as well as the tracker that manages the swarm
- A text file with various (encoded) fields:
  - info: describes the file(s) of the torrent. Includes:
    - File names, file length, length of pieces in torrent, and for each piece the SHA1 hash of the piece
  - announce: URL of the tracker
    - An optional list of backup trackers may also be included
  - creation date: when the torrent was created
  - created by: name of software that created torrent
  - comment: textual description of the contents

Example .torrent File

d8:announce39:http://torrent.ubuntu.com:6969/announce
7:comment29:Ubuntu CD releases.ubuntu.com
13:creation datei1225365524e
4:infod
6:lengthi732766208e
4:name28:ubuntu-8.10-desktop-i386.iso
12:piece lengthi524288e
6:pieces27960:
þý“!├ .¶ ÈTO ™┼ □ îðÿB2ÅÉÒ□óý¨úþ9tz>©‘U…



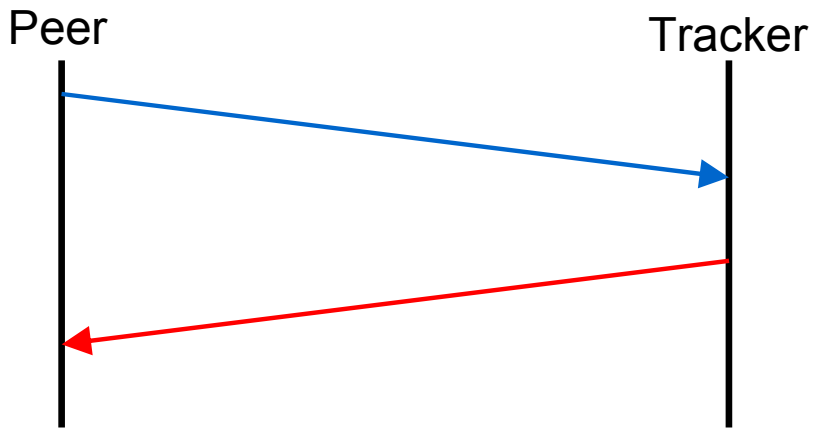
Hash values of all 27960 pieces

Peer to Tracker Communications

- The Tracker implements a HTTP web server
- Peers communicate with the tracker using HTTP
  - Peer sends HTTP GET requests. URL contains information about the peers activities in the torrent:
    - ID of peer
    - Port number the client is listening on
    - Bytes uploaded/downloaded so far; Bytes remaining to download
    - Status of download (started, stopped, completed)
    - …
  - Tracker sends HTTP responses, containing plain text:
    - Interval that client should send requests to tracker
    - Number of peers completed (seeds) and incomplete (leechers) in the swarm
    - A list of peers (peer ID, IP address, port number)
    - Warning/error messages
    - …

Example Peer/Tracker Exchange

```
GET /announce?info_hash=3%82%0d%b6%dd%5eY%28%d2%3b%c8%11%bb%ac%2fJ%e9L%b8%82
&peer_id=-UT1810-_1%02A%ba%d4%9a%0c%e8JBB
&port=11541          &uploaded=0          &downloaded=0        &left=732766208
&corrupt=0           &key=82D434AB        &event=started       &numwant=200
&compact=1           &no_peer_id=1        &ipv6=192.168.1.2
HTTP/1.1
Host: torrent.ubuntu.com:6969
User-Agent: uTorrent/1810
Accept-Encoding:gzip
```




```
HTTP/1.0 200 OK
Content-Length: 362
Content-Type: text/plain
Pragma: no-cache

d8:completei1328e10:incompletei97e8:intervali1800e5:peers300:<.#...Z .....
```

Peer Exchange Protocol

- A TCP connection is created from a peer (client) to a remote peer and they exchange messages to inform of current status and download blocks
  - At the start, the client sends a Handshake message to remote peer
    - Informs remote peer of PeerID
  - This may be followed by a Bitfield message, where the remote peer indicates the set of pieces it has available for download

- Once the Handshake is sent, a peer (client or remote peer) may send:
  - Request message to request a specific block
  - Piece message which includes a block
    - Note: although called “Piece” message, it is sending a block
  - Have message to indicate a piece a peer has
  - Cancel message to cancel a previous Request for a block
  - Keepalive message to keep the TCP connection open
    - In the case nothing has been sent between the peers recently

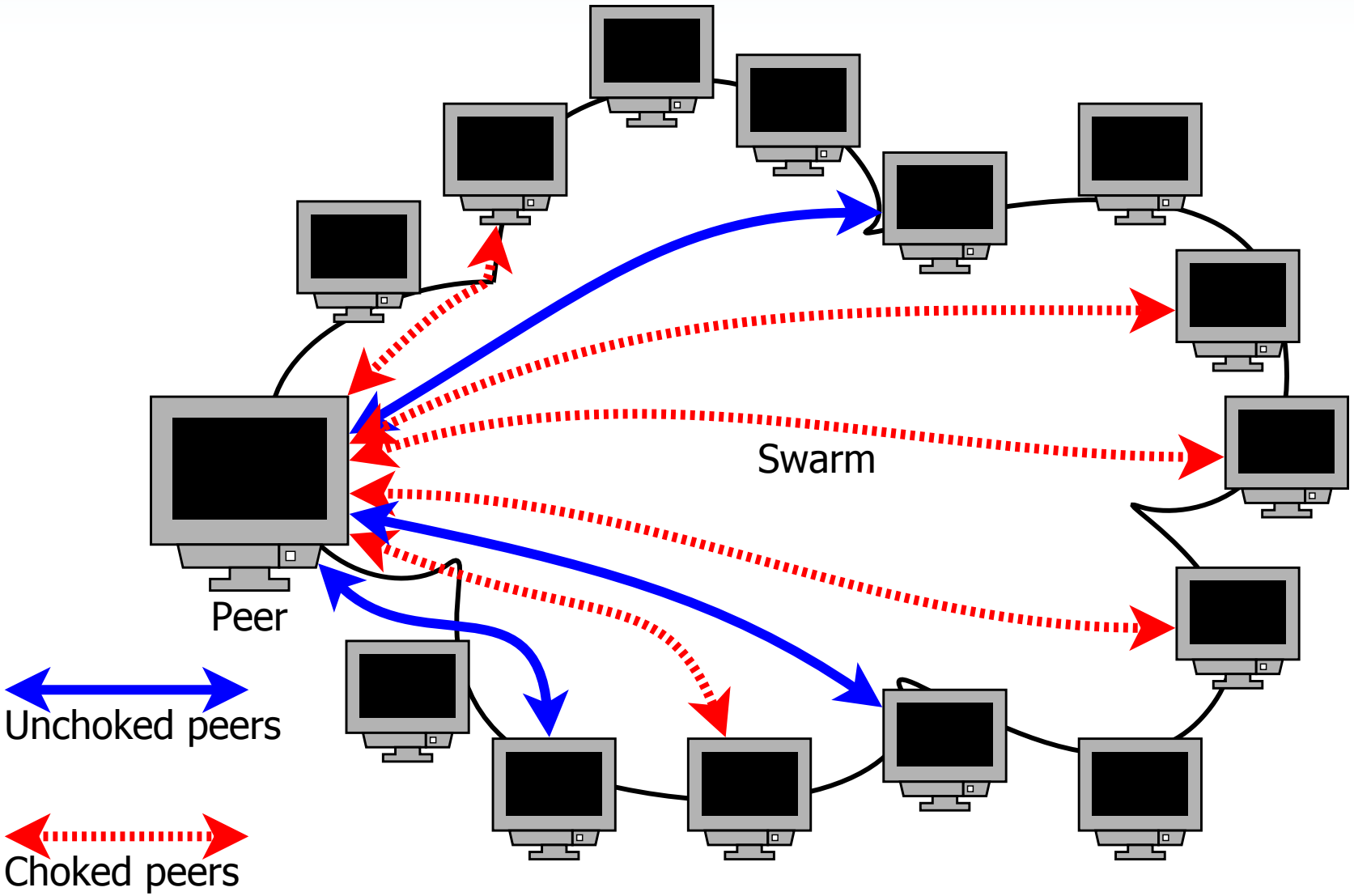
Peer Exchange Protocol

- BitTorrent aims to implement a ‘tit-for-tat’ algorithm for data transfer
  - For every one piece the client downloads, it uploads one piece
  - Aim to maintain equal amount of upload/download
  - BitTorrent applies this while a peer is downloading a torrent
    - Once the peer has completed (i.e. becomes a Seed), there is no way for the protocol to enforce the peer to keep uploading
- Peers uses two concepts:
  - Choking: when a client is choked by a remote peer, then no requests will be handled by the remote peer
  - Interested: when a remote peer is interested in the client, then it would like to receive a piece that the client has
  - Client maintains status of remote peer (values may be true or false):
    - am_choking: the client is choking the remote peer
    - am_interested: the client is interested in the remote peer
    - peer_choking: the remote peer is choking the client
    - peer_interested: the remote peer is interested in the client

Peer Exchange Protocol

- The peers may send following messages to change state:
  - Choked message
  - Unchoked message
  - Interested message
  - Uninterested message
- Each peer implements a choking algorithm to determine when to choke/unchoke peers
  - Need to consider many factors, including the download rate from different peers
- For a torrent, a client may have connections with many (10's of peers), but generally only several (4 to 10) will be unchoked at anyone time

Peers in a Swarm



Torrent, Pieces and Blocks

- Although there is no size limits, using BitTorrent is usually most efficient for large sizes, e.g. MBytes to GBytes
- Pieces are normally 256KB, 512KB or 1MB
- Blocks are normally 16KB

- Which pieces does a client download first?
  - Random is one option
  - Rarest-first is another (more common) option
    - A client peer keeps track of the pieces all other peers in the swarm have
      - Original Bitfield message, and regular Have messages
    - Client peer downloads the pieces which are least common, so that it will become more common
- *Availability* of a torrent: number of copies of torrent in swarm
  - 1 = all pieces are available at peers in the swarm; 0.5 = only 50% of pieces are available (you will not be able to download entire file); 20 = 20 copies of the torrent are available

Performance Issues

- How many peers to connect to?
  - About 10 to 30 for each torrent
- How many peers to download from (unchoked)?
- What are the optimal piece and block sizes?
- What is the best choking algorithm?
  - When do you try a different peer (in the hope that they will offer faster download)?
- How to ensure users upload content?
- How to maintain fairness with other applications?
  - Remember, TCP aims to provide fairness amongst connections. If BitTorrent has 100 connections and web browser has only 10 connections, there may be extreme unfairness
- …