Lecture 1

Crypto & P2P

Crypto

1. Hash
2. symmetric encryption
3. public-key (asymmetric) encryption

Hashing

1. signature any kind of bytes.
2. applied to any size data.
3. produces a fixed-length output.

Hashing

example

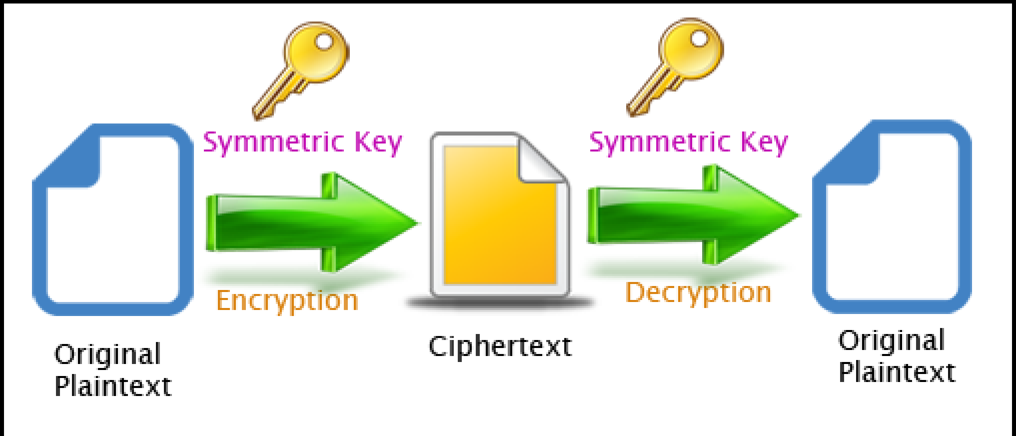
```
Hash_Example$go run Gohash.go -t HelloWorld
872e4e50ce9990d8b041330c47c9ddd11bec6b503ae9386a99da8584e9bb12c4
Hash_Example$
```

Hashing

Live example…

symmetric encryption

1. A very old historical confidentiality tool
2. Two way encrypt



symmetric encryption

Live example…

Asymmetric encryption

Math behind...

Ref: https://www.youtube.com/watch?v=Jt5EDBOcZ44

Asymmetric encryption

Select Two Primes P & Q

$P = 53$ $Q = 59$

Start to calculate first part of the Public Key = n

$PQ = n$ Or $53 \times 59 = 3127$

We also need a small exponent e

e must:

1) Be an Integer
2) Not be a factor of n
3) $1 < e < \varphi(n)$

For this example $e = 3$

Our public key is made up of n & e or 3127 & 3

We need to calculate $\varphi(n)$

$$\varphi(n) = (P-1)(Q-1)$$

$$\varphi(n) = (53-1)(59-1)$$

$$\varphi(n) = 3016$$

Now Calculate Private Key = d

$$d = \frac{2(\varphi(n)) + 1}{e}$$

$$2011 = \frac{2(3016) + 1}{3}$$

Private Key

$d = 2011$

We now have everything we need

1) Our public key is made up of n & e or **3127** & **3**
2) Our Private Key of d or **2011**

A simple example. Lets encrypt “HI”

Convert letters to numbers: H = 8 & I = 9

“89”

c = Encrypted Data

$c = 89^e \bmod n$

$1394 = 89^3 \bmod 3127$

Decrypt the data with the Private Key

c = Encrypted Data

d = Private Key

n = Public Key

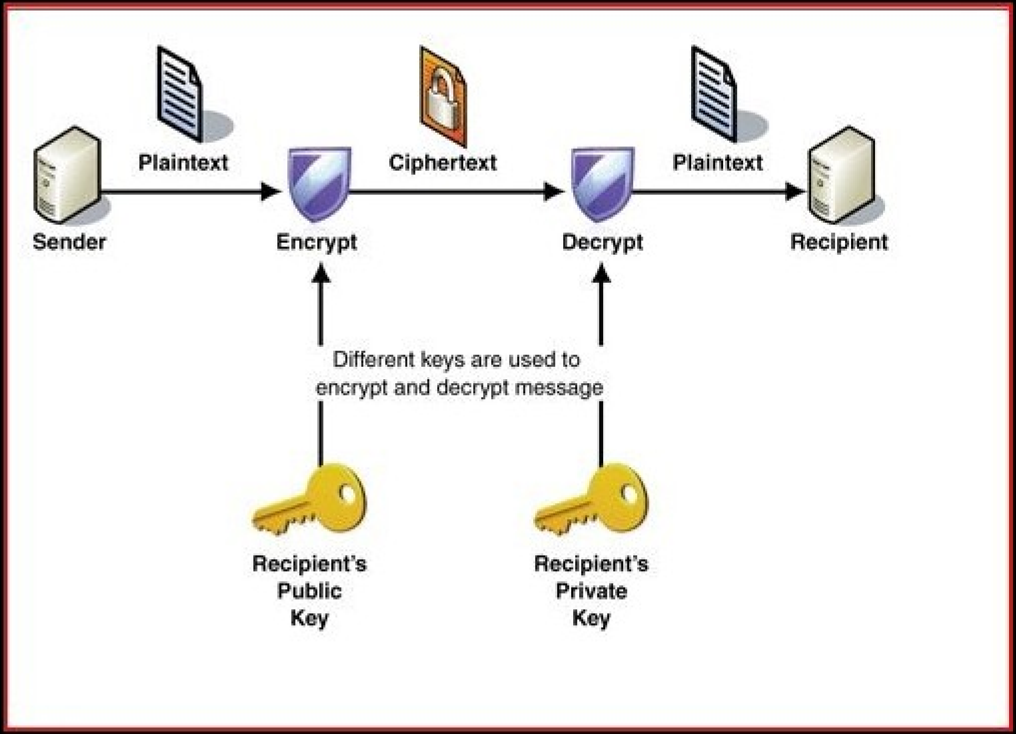
Decrypted Data = $c^d \bmod n$

$89 = 1394^{2011} \bmod 3127$

8 = H 9 =I

“HI”

Asymmetric encryption



Asymmetric encryption

Live example…

Asymmetric encryption

Using in TLS

1. Confidentiality.
2. message digest.
3. Certificate Authentication.

P2P

Peer-to-Peer Computing

Sameh El-Ansary

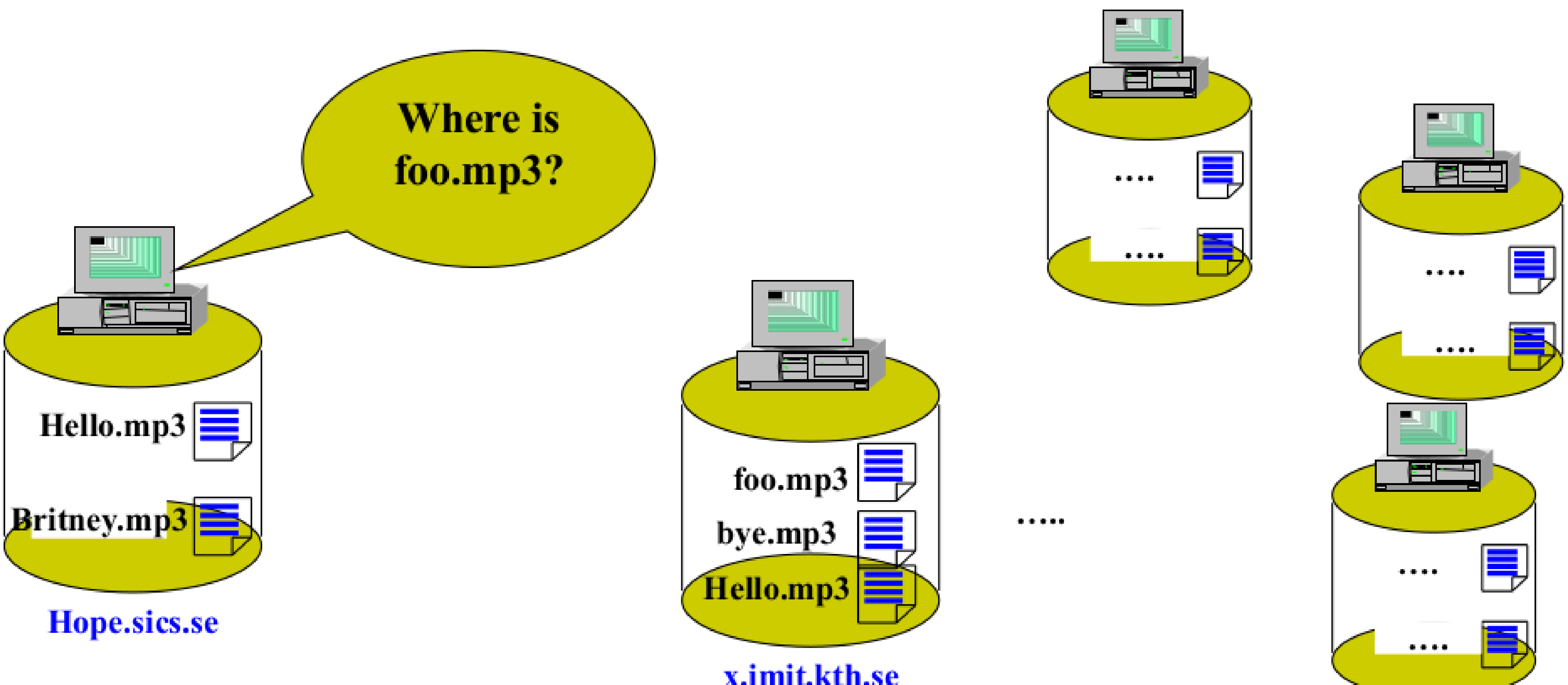
Nile University

Outline

- What is P2P?
- What problems come with it?
- Evolution (Discovery Related)
  - 1st Generation:
    - Napster
  - 2nd Generation: Flooding-Based systems
    - Gnutella
  - 3rd Generation: Distributed Hash Tables
    - Chord, DKS, Pastry etc…
- Applications

Let us see how did it all start..

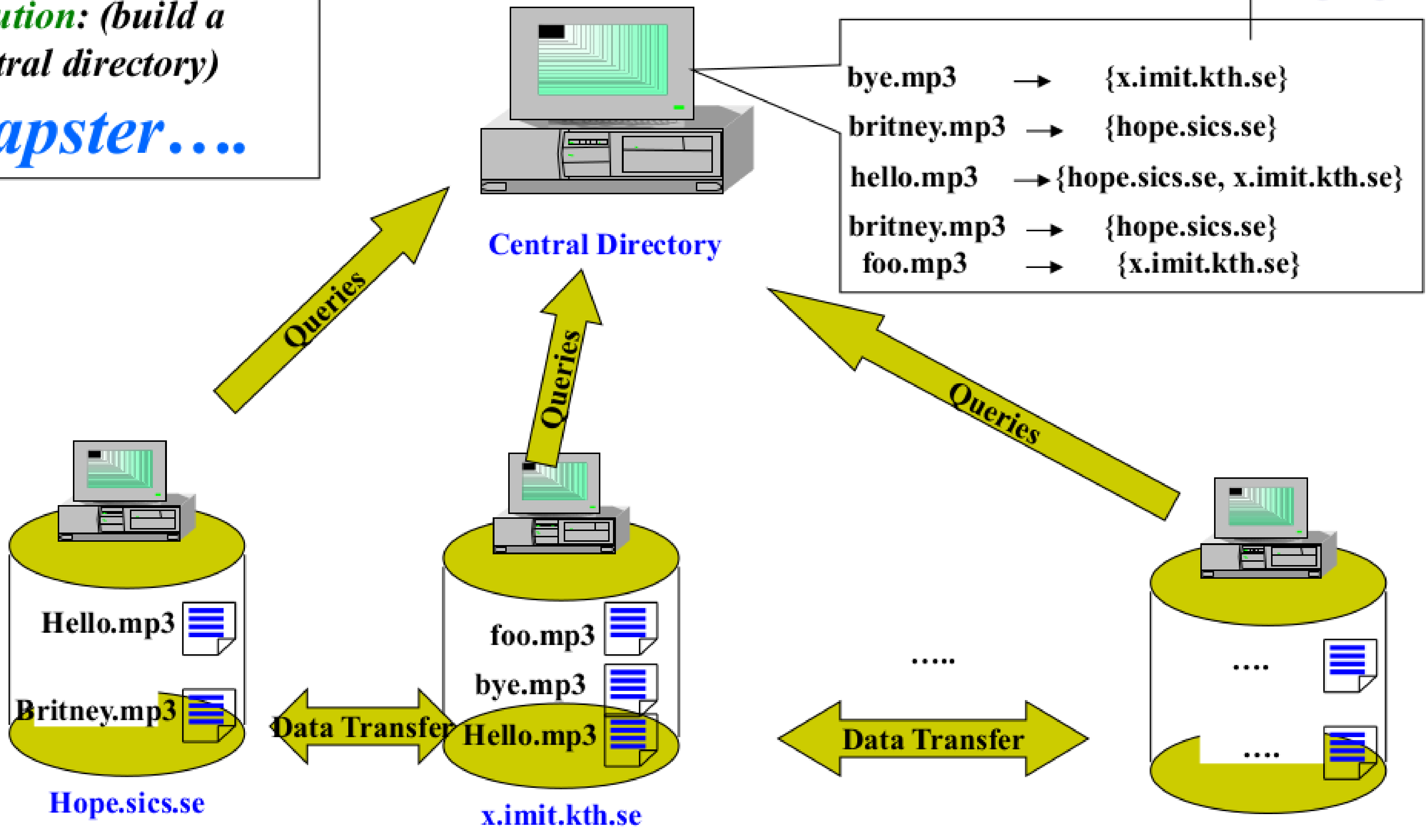
- Users storing data items on their machines
- Other users are interested in this data
- Problem: ***How does a user know which other user(s) in the world have the data item(s) that he desires?***



1st Generation of P2P systems (Central Directory + Distributed Storage)

Solution: (build a central directory)

Napster….



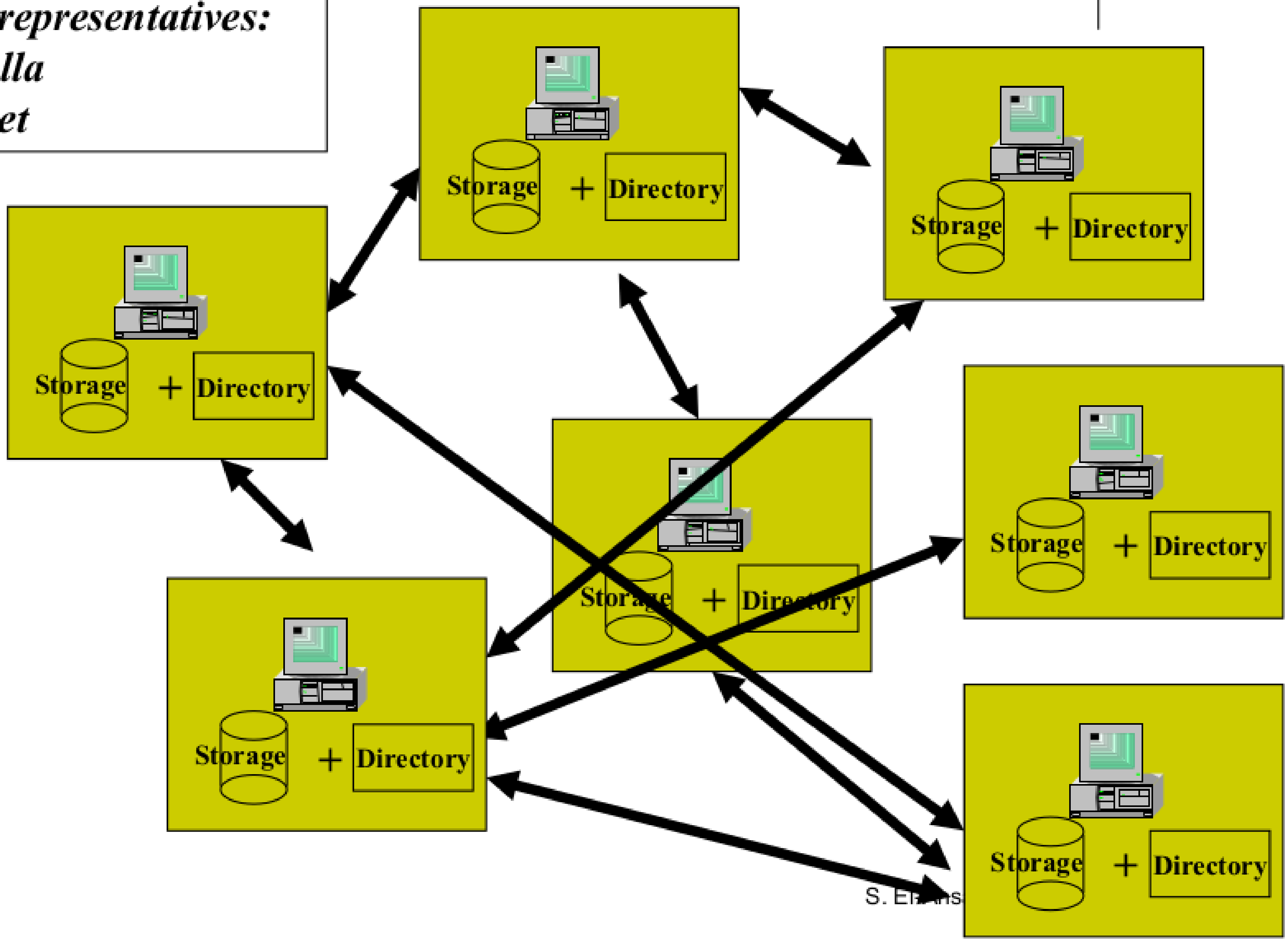
The End of Napster

- Since users of Napster stored copyrighted material, the service was stopped for legal reasons
- But…. a second generation appeared…
- Discussion
  - Scalability
  - Failure

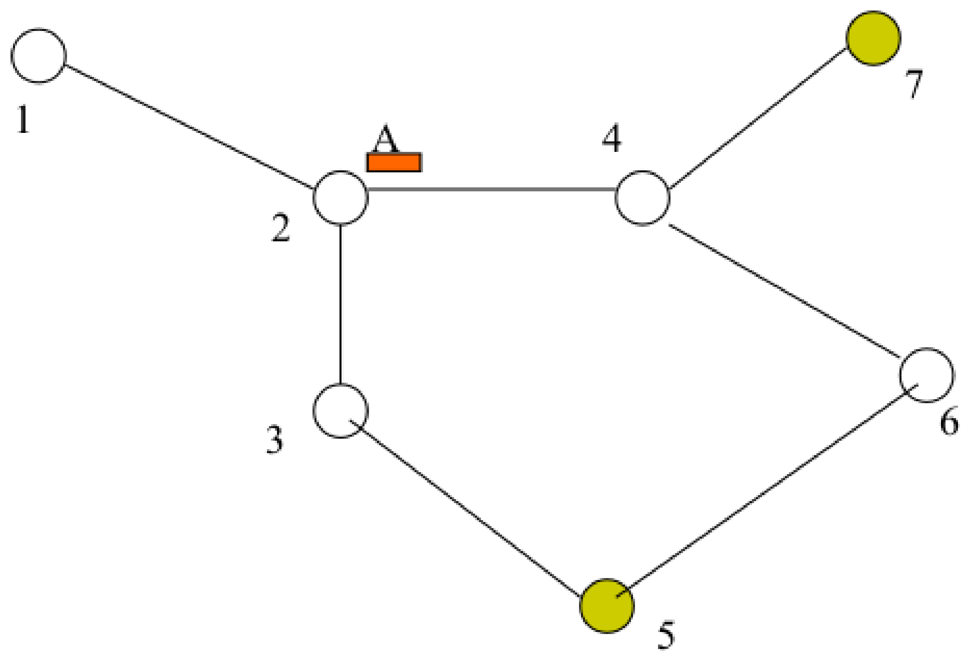
Q

2nd Generation (Random Overlay Networks Distributed Directory + Distributed Storage)

Main representatives:
Gnutella
Freenet



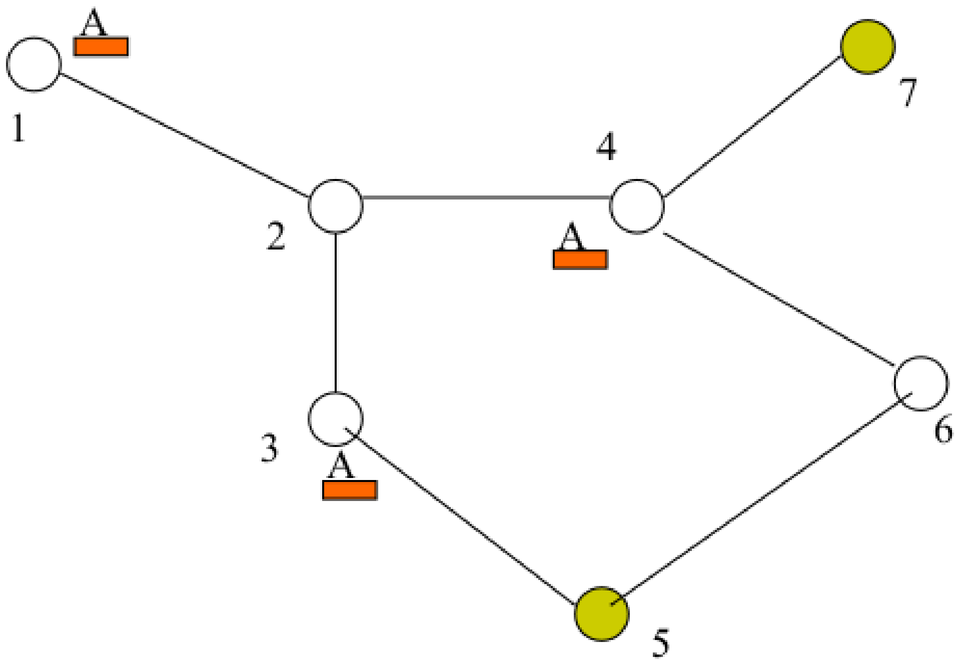
Gnutella search mechanism



Steps:

- Node 2 initiates search for file A

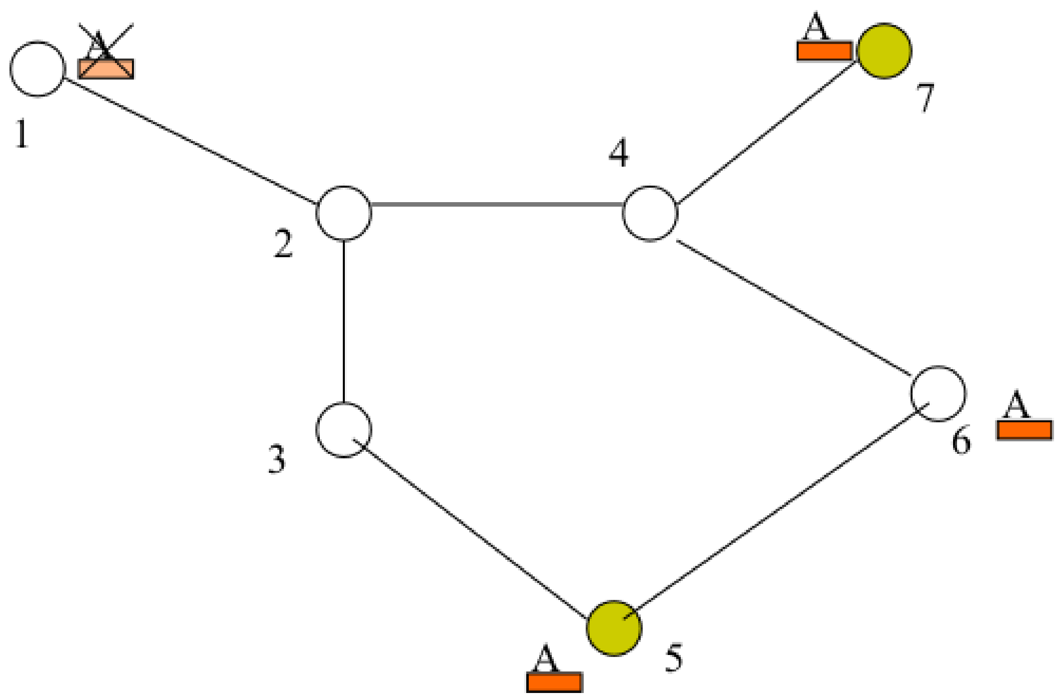
Gnutella Search Mechanism



Steps:

- Node 2 initiates search for file A
- Sends message to all neighbors

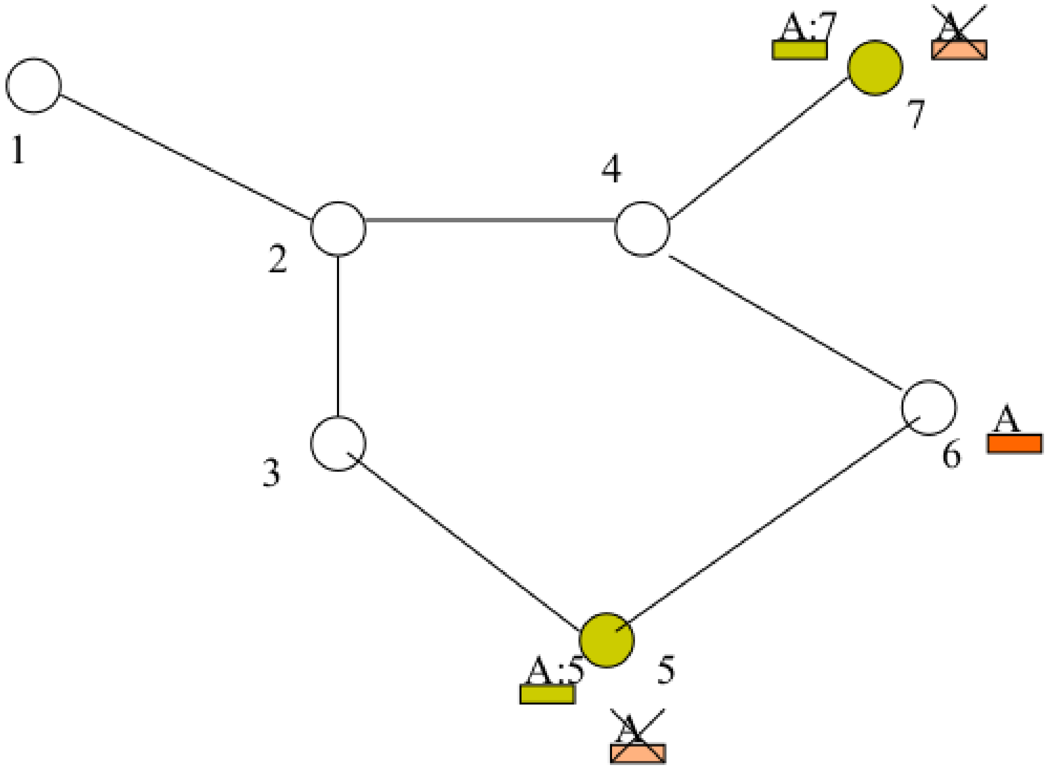
Gnutella Search Mechanism



Steps:

- Node 2 initiates search for file A
- Sends message to all neighbors
- Neighbors forward message

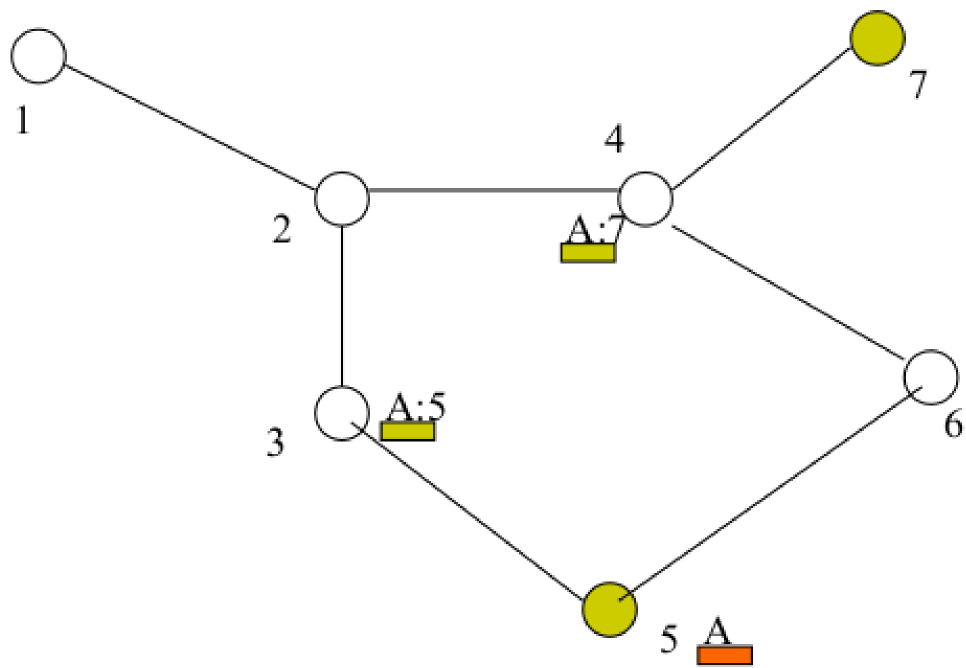
Gnutella Search Mechanism



Steps:

- Node 2 initiates search for file A
- Sends message to all neighbors
- Neighbors forward message
- Nodes that have file A initiate a reply message

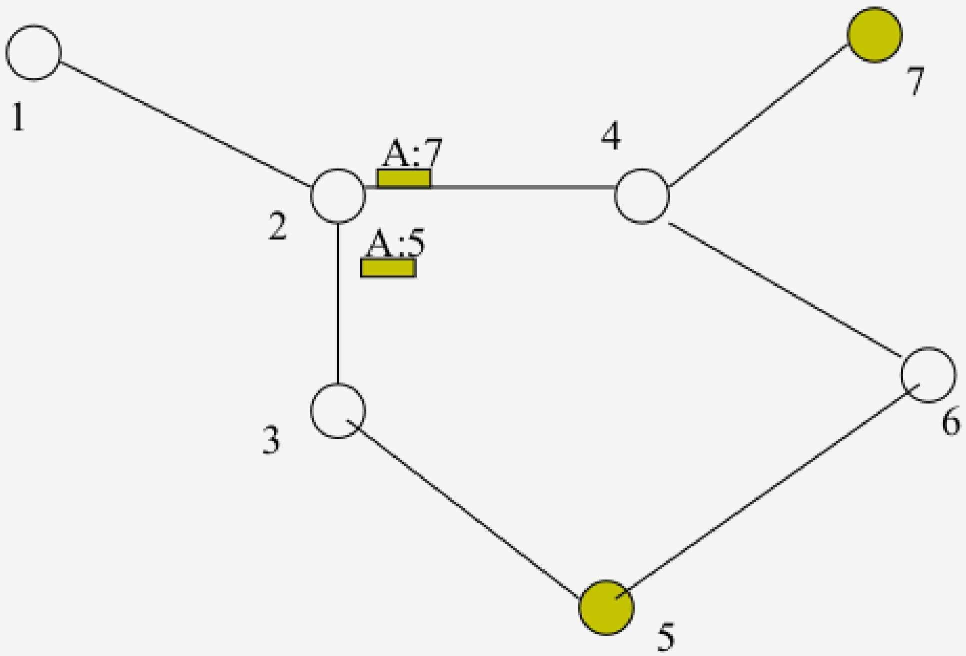
Gnutella Search Mechanism



Steps:

- Node 2 initiates search for file A
- Sends message to all neighbors
- Neighbors forward message
- Nodes that have file A initiate a reply message
- Query reply message is back-propagated

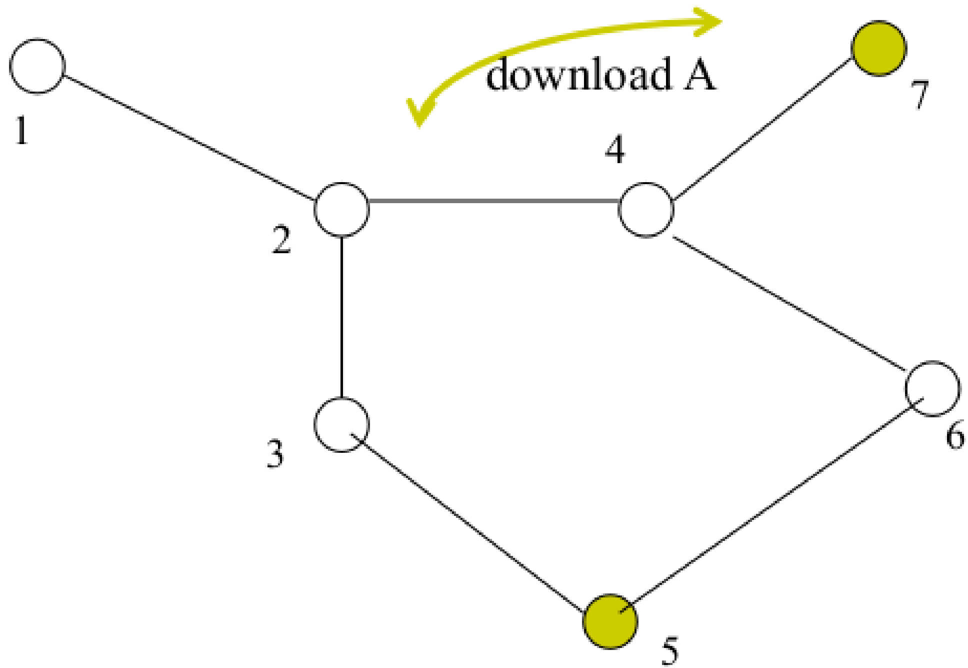
Gnutella Search Mechanism



Steps:

- Node 2 initiates search for file A
- Sends message to all neighbors
- Neighbors forward message
- Nodes that have file A initiate a reply message
- Query reply message is back-propagated

Gnutella Search Mechanism

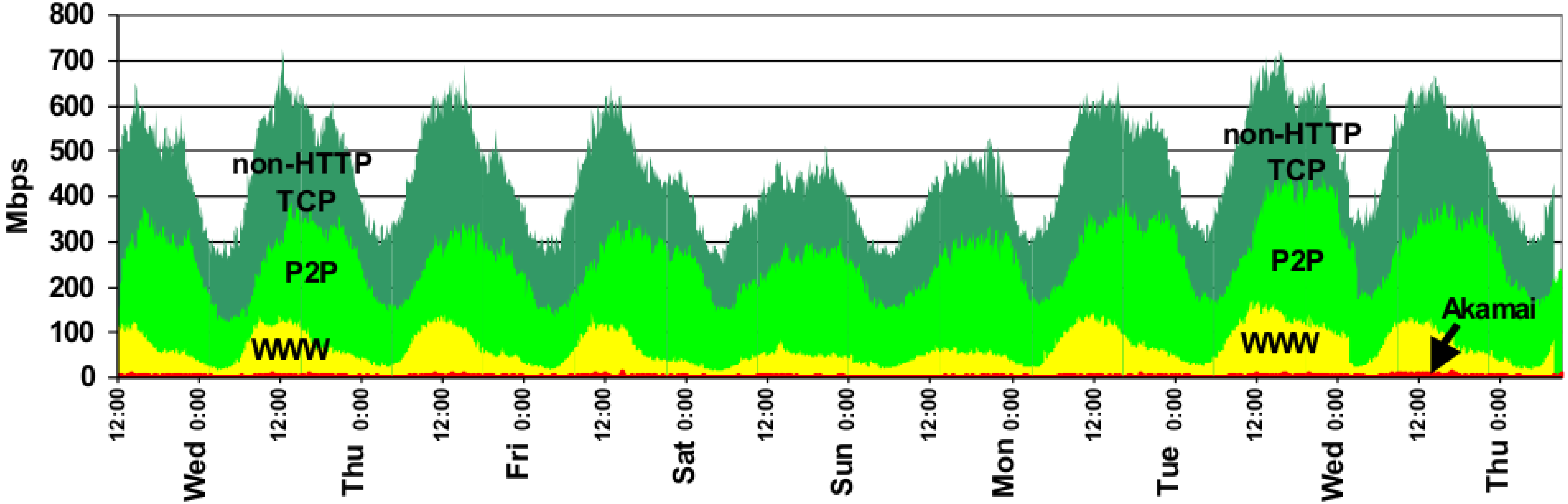


Steps:

- Node 2 initiates search for file A
- Sends message to all neighbors
- Neighbors forward message
- Nodes that have file A initiate a reply message
- Query reply message is back-propagated
- File download

- Note: file transfer between clients behind firewalls is not possible; if only one client, X, is behind a firewall, Y can request that X push the file to Y

P2P Why Should we care?



Breakdown of UW TCP bandwidth into HTTP Components (May 2002)

- 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

Gnutella Summary

- Uses a flooding-based algorithm
- Simple
- Robust
- Created too much control traffic
  - Actually became a problem for ISPs
- Low guarantees
- The problem motivated academic research to create a better solution
- Discussion:
  - How do we know the first nodes?
  - Can we reduce this traffic without drastic changes?

Q

Key idea in DHTs

1st, 2nd Generation: *Each data item is stored in the machine of its creator/downloader*

Third Generation (DHTs): *The id of a data item determines the machine on which it is going to be stored*

Simplest Example: Consistent Hashing using a Ring

Consistent Hashing using a Ring (3/6)

- In the following example, we are using a hashing function with a range of 0-15, i.e. with a maximum of 16 nodes.

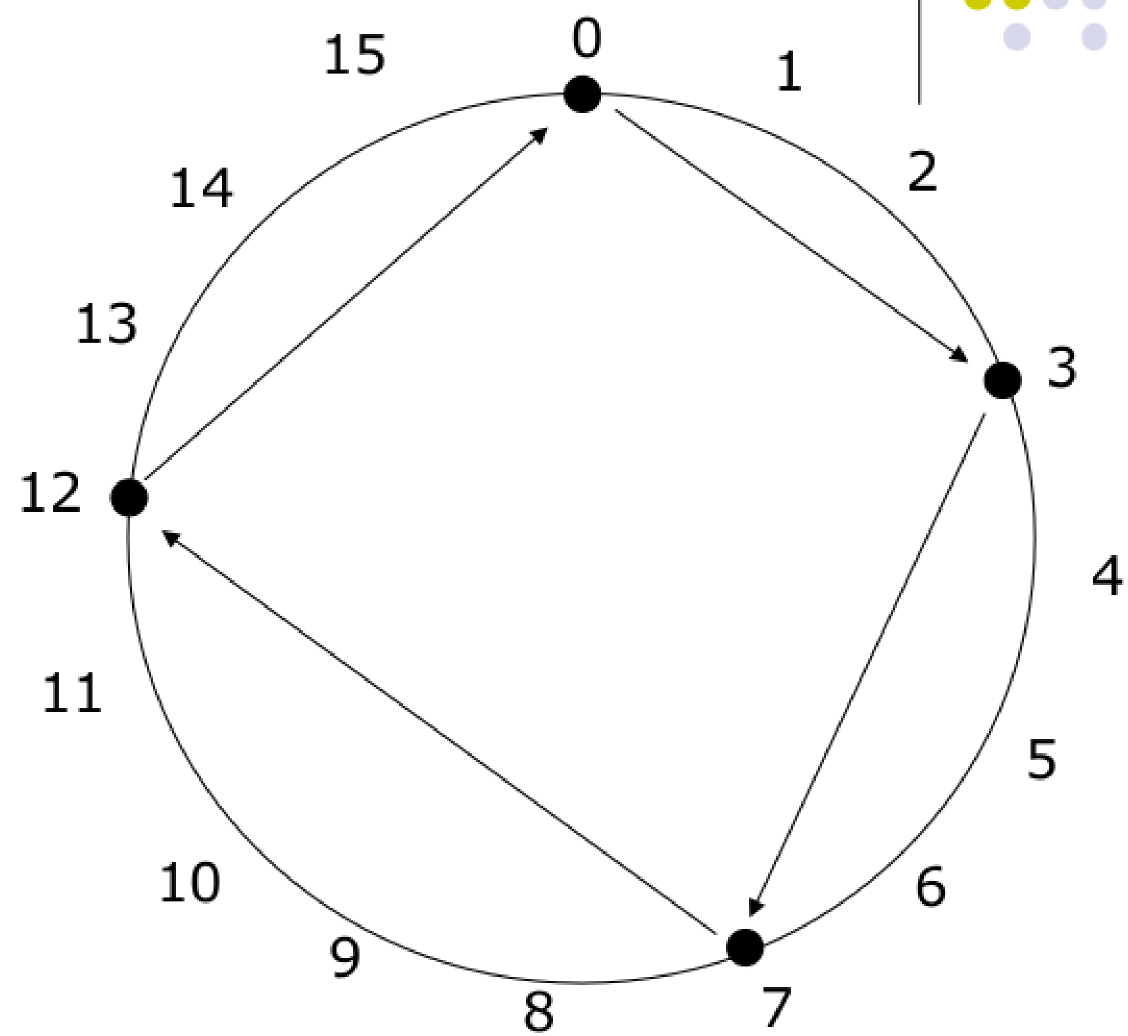
- We treat this range as a circular id space

-*Succ(x)* is the first node on the ring with id greater than or equal x, where x is the id of a node or a document

-Every node n has one successor pointer to *Succ(n+1)*

-**Thus, the nodes are forming a ring.**

- **Q:** how can we build this ring? Failures, joins etc….

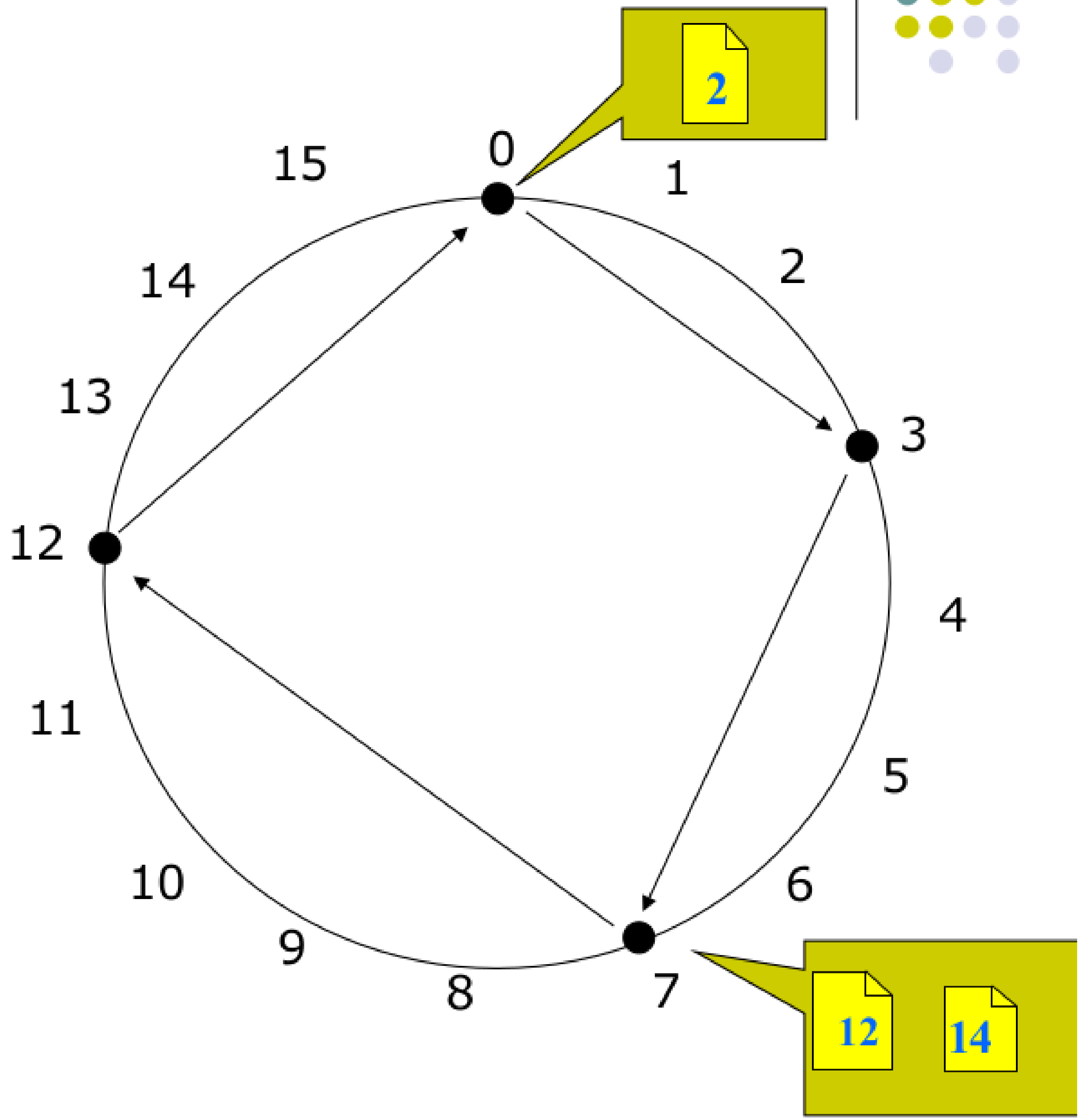


Consistent Hashing using a Ring (4/6)

- Using this ring, we can decide which document is stored at which node.

- Initially, node 0 stored doc 2 and node 7 store docs 9,14

- However, using a DHT scheme for storing files, <u>this will not be the case</u>



Consistent Hashing using Ring (5/6)

- The policy is: A doc with id *y*, would be stored at *Succ(y)*

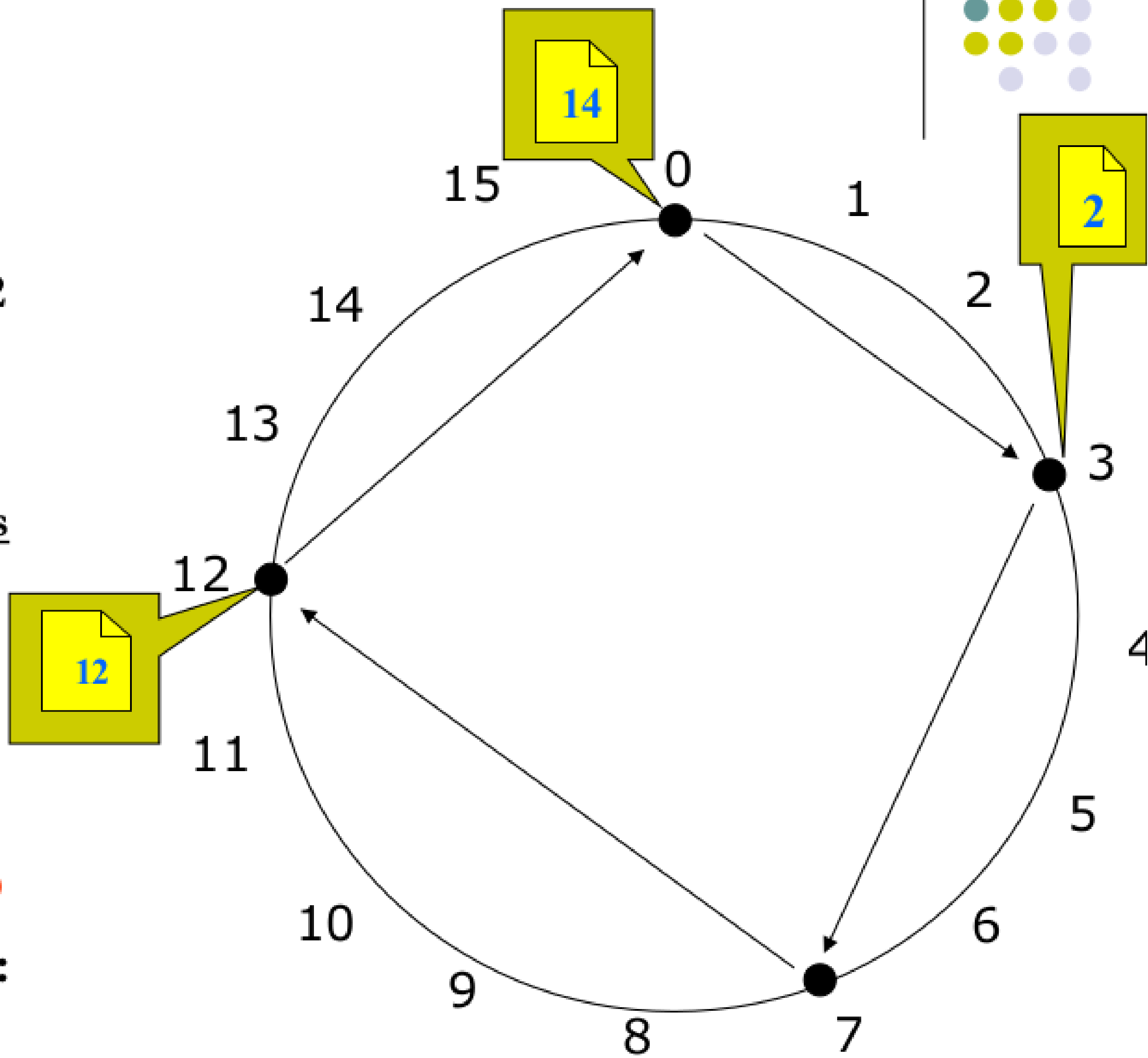
- So, node 0 gets to store doc 14, node 3 to store doc 2, and node 12 to store doc 12

- But how can we do this?

- Simple, <u>if the successor pointers are already there</u>, the two operations, get and put would be simply done by following them sequentially

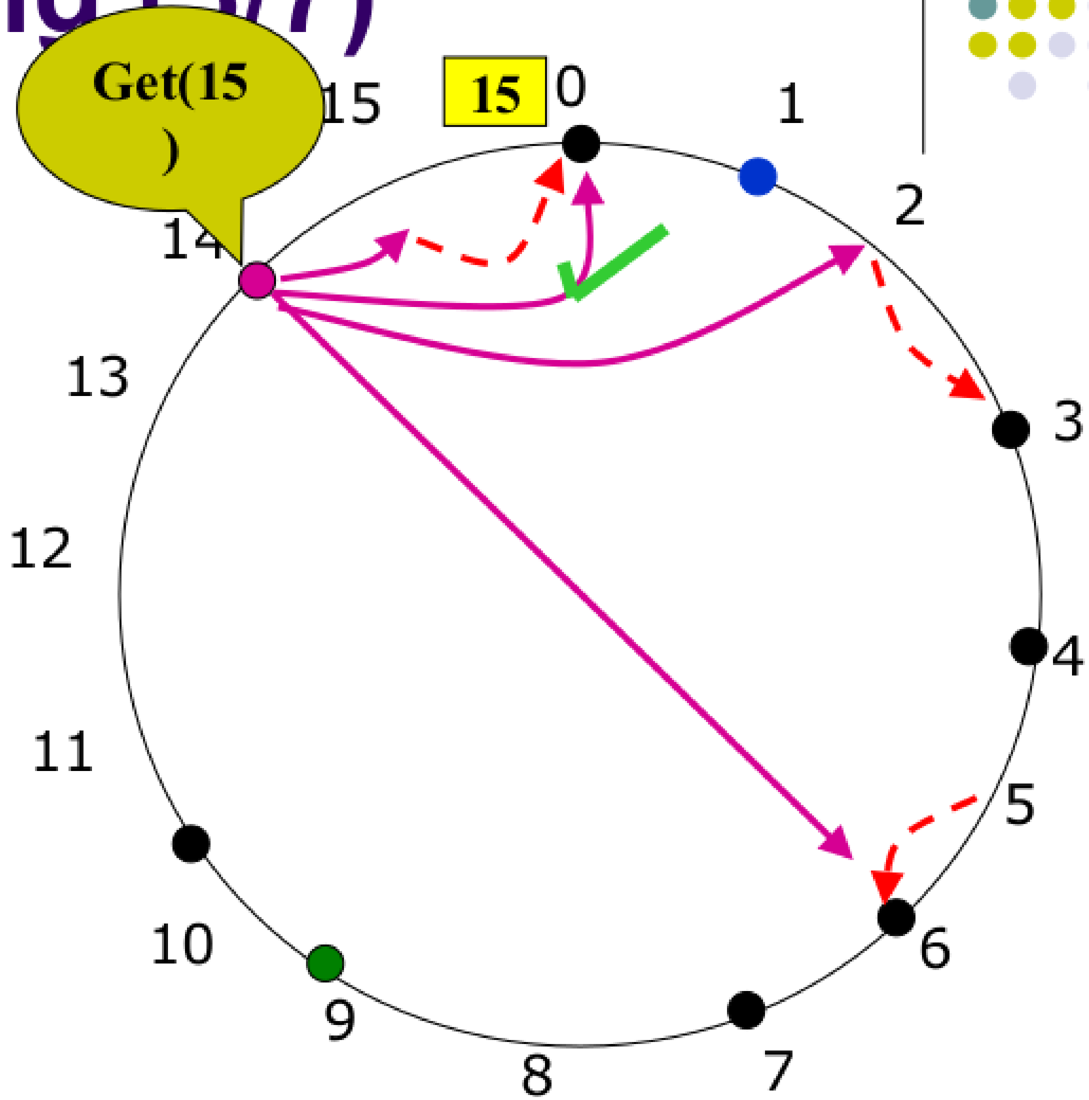
- From any node, you can do:
`put( hash(plan.doc), plan.doc )`

- From any node, you can also do:
`get(hash(plan.doc))`



Chord – Routing (3/7)

- Routing table size: *M*, where $N = 2^M$
- Every node n knows successor($n + 2^{i-1}$) , for $i = 1..M$
- Routing entries = $log_2(N)$
- $log_2(N)$ hops from any node to any other node

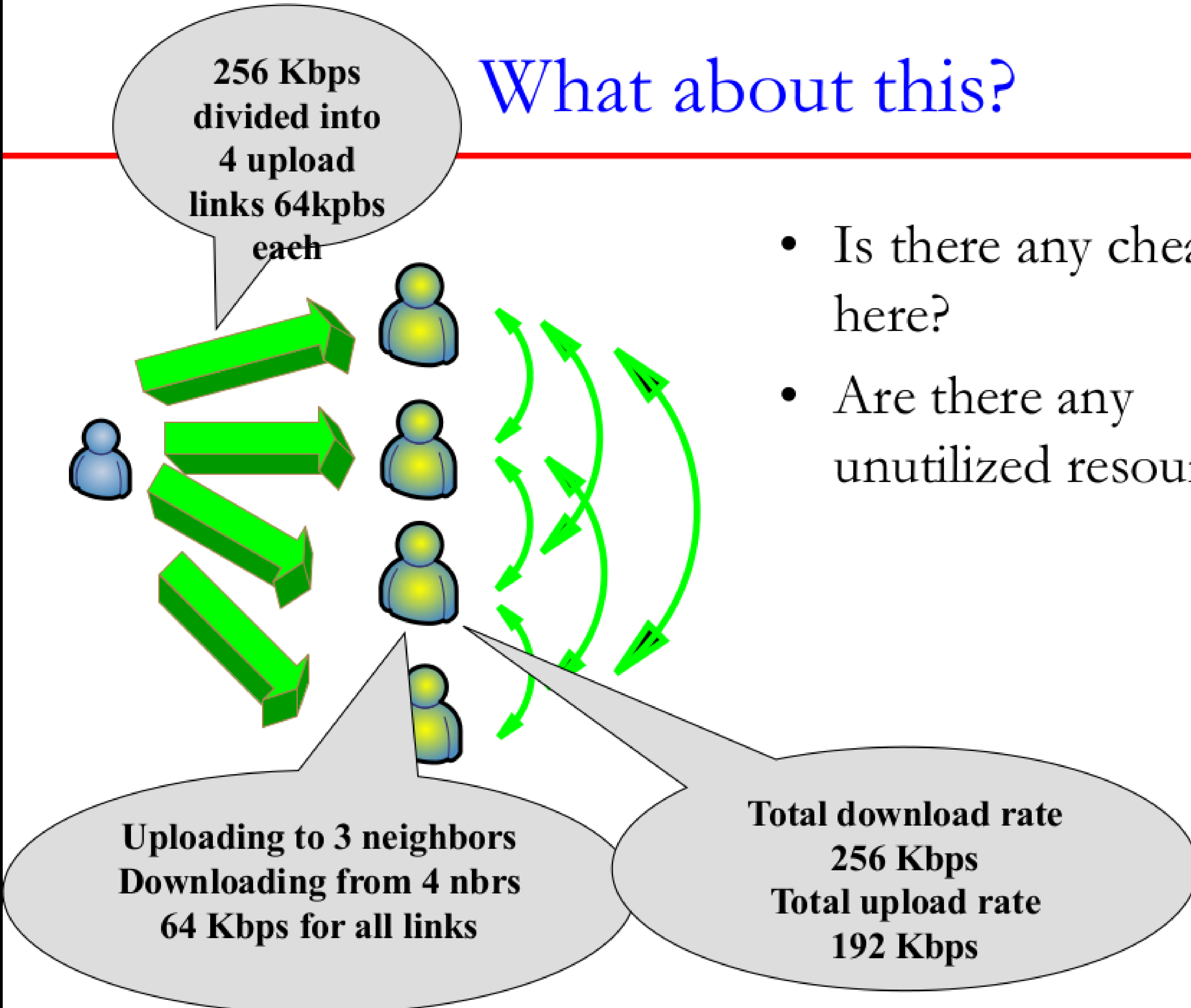


On Content Distribution
&
BitTorrent

Sameh El-Ansary

CIT-614

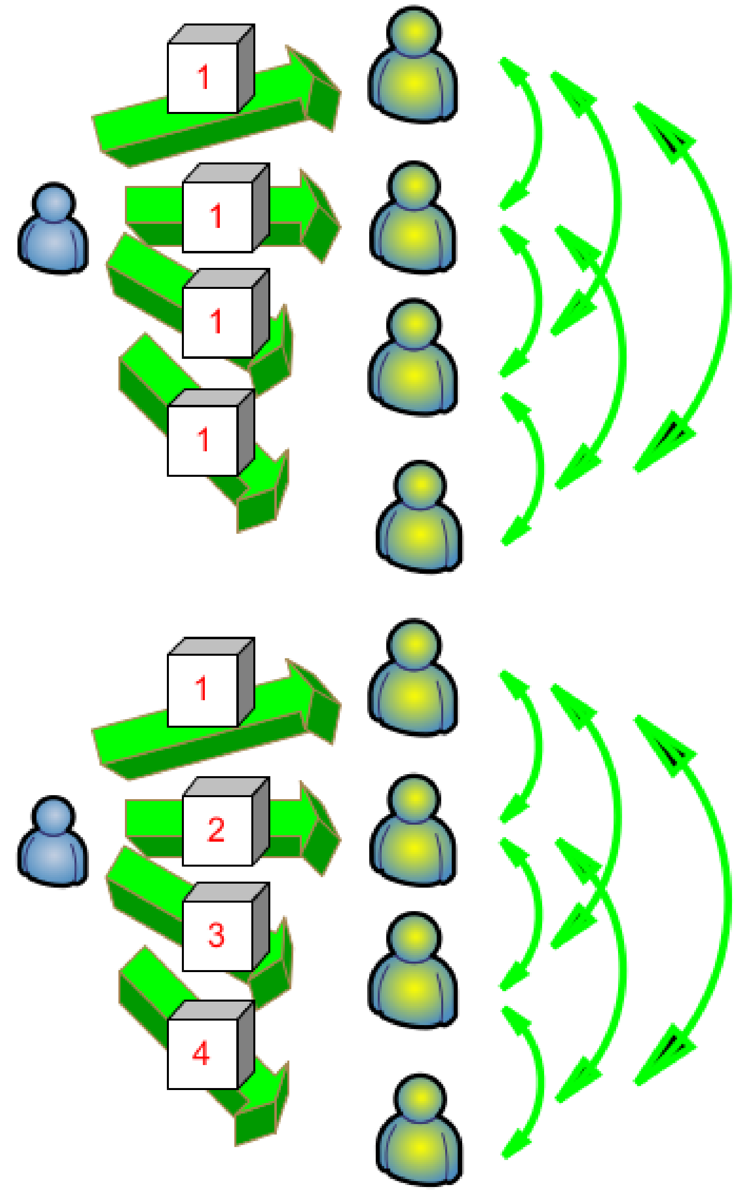
What about this?



- Is there any cheating here?
- Are there any unutilized resources?

Is the choice of peers the only thing that matters?

- Even though it is possible to download at full speed, we might be forced to download slower
- Example: Downloading the first piece.
- Which scenario is better?



BitTorrent Terminology

- Seeds
- Leechers
- Tracker
- Torrent file/meta-info file
- Choking/Unchoking
- Rarest-First
- Tit-for-Tat
- Pareto Efficiency
- Etc..

Seeds

- Machines that have a complete copy of the file
- They are usually selfish and do not want to wait after they get the file
- At least the first seed has to stay to serve one complete copy of the file
- In general at all times all pieces of a file must be around or the process fails

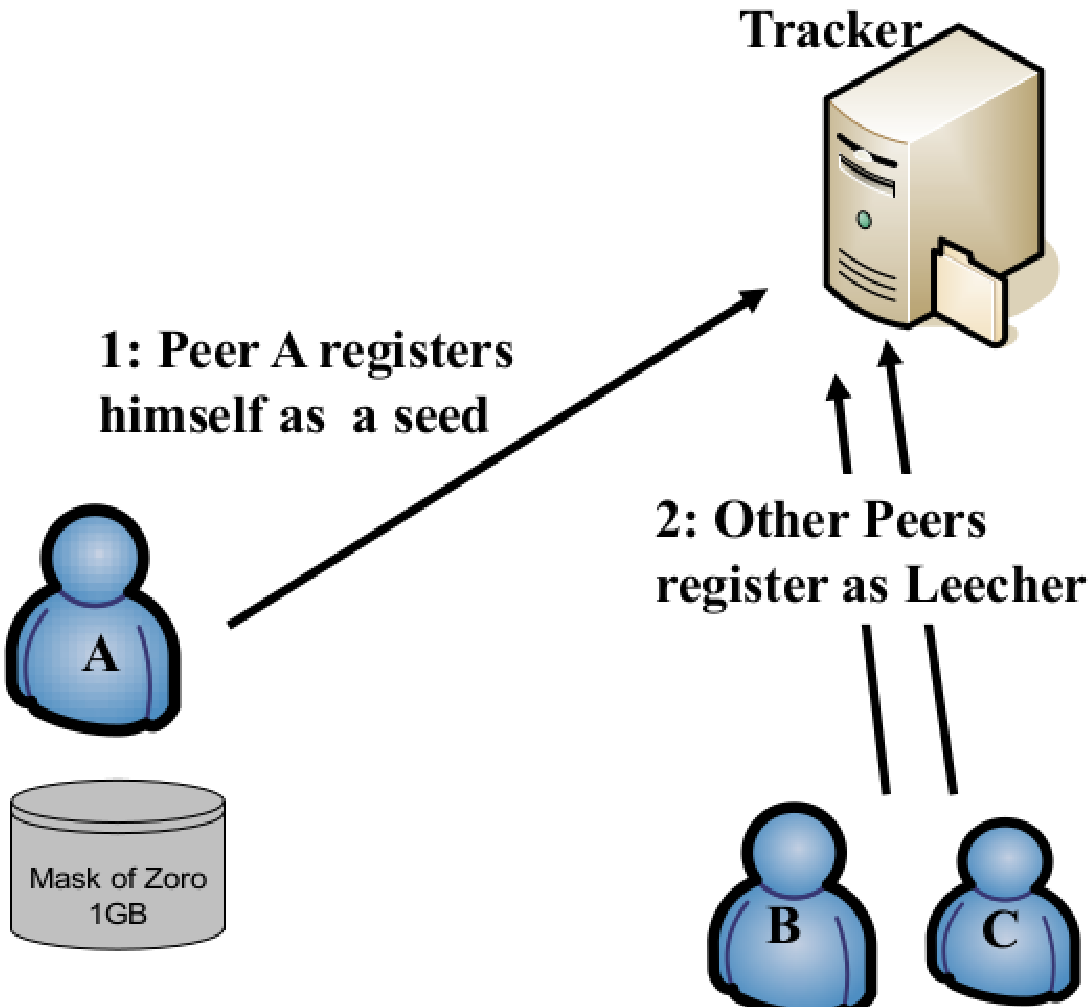
Leechers

- Any peer who does not have a complete file is called a leecher
- He stays a leecher and eventually becomes a seed

Tracker

- A peer that keeps track of:
  - Who are the seeds and the leechers
  - Which pieces each peer has

Back to the simple solution ..



1: Peer A registers himself as a seed

2: Other Peers register as Leechers

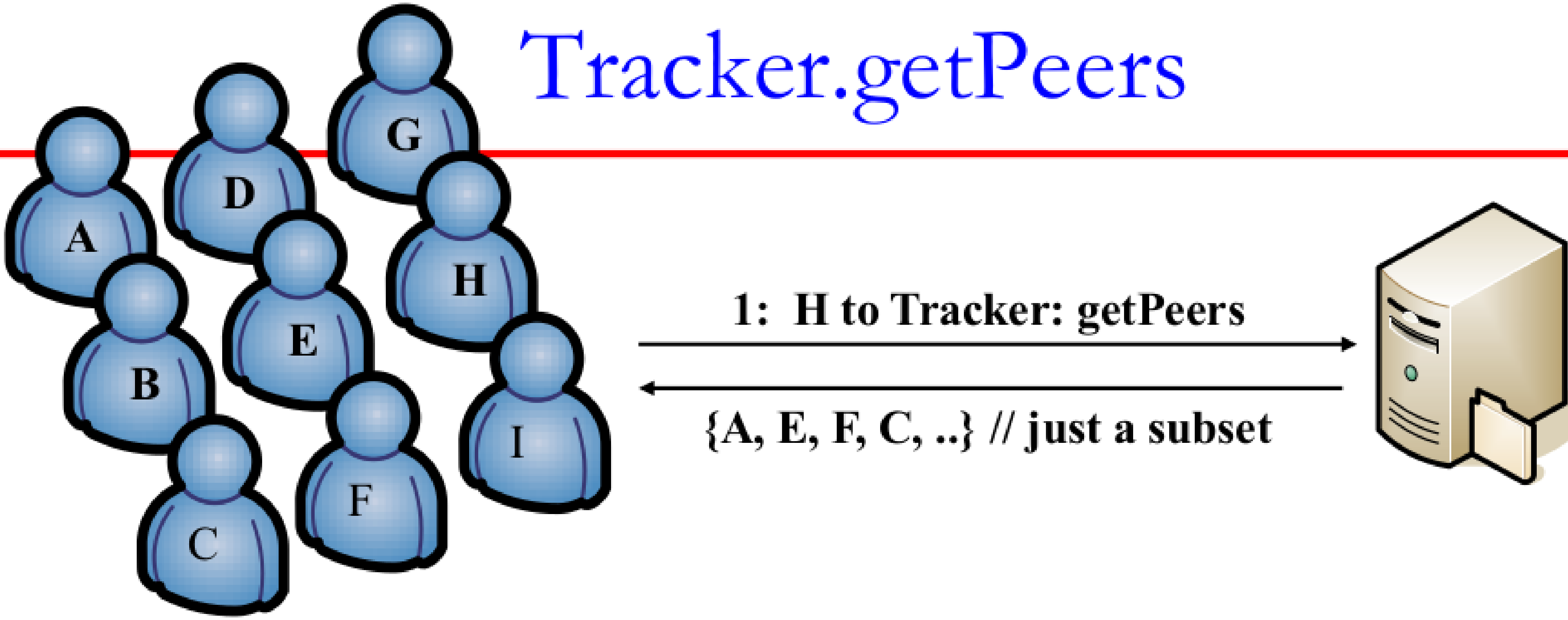
3: The tracker builds a table like this

Peer	Bitfield
A	11111111
B	00000000
C	01100000

4: Peers ask the tracker:
- to select for them random peers as much as they need
- ask the tracker to select pieces at random from those peers (or ask the peers directly)

5: Leechers start to have pieces

Tracker.getPeers



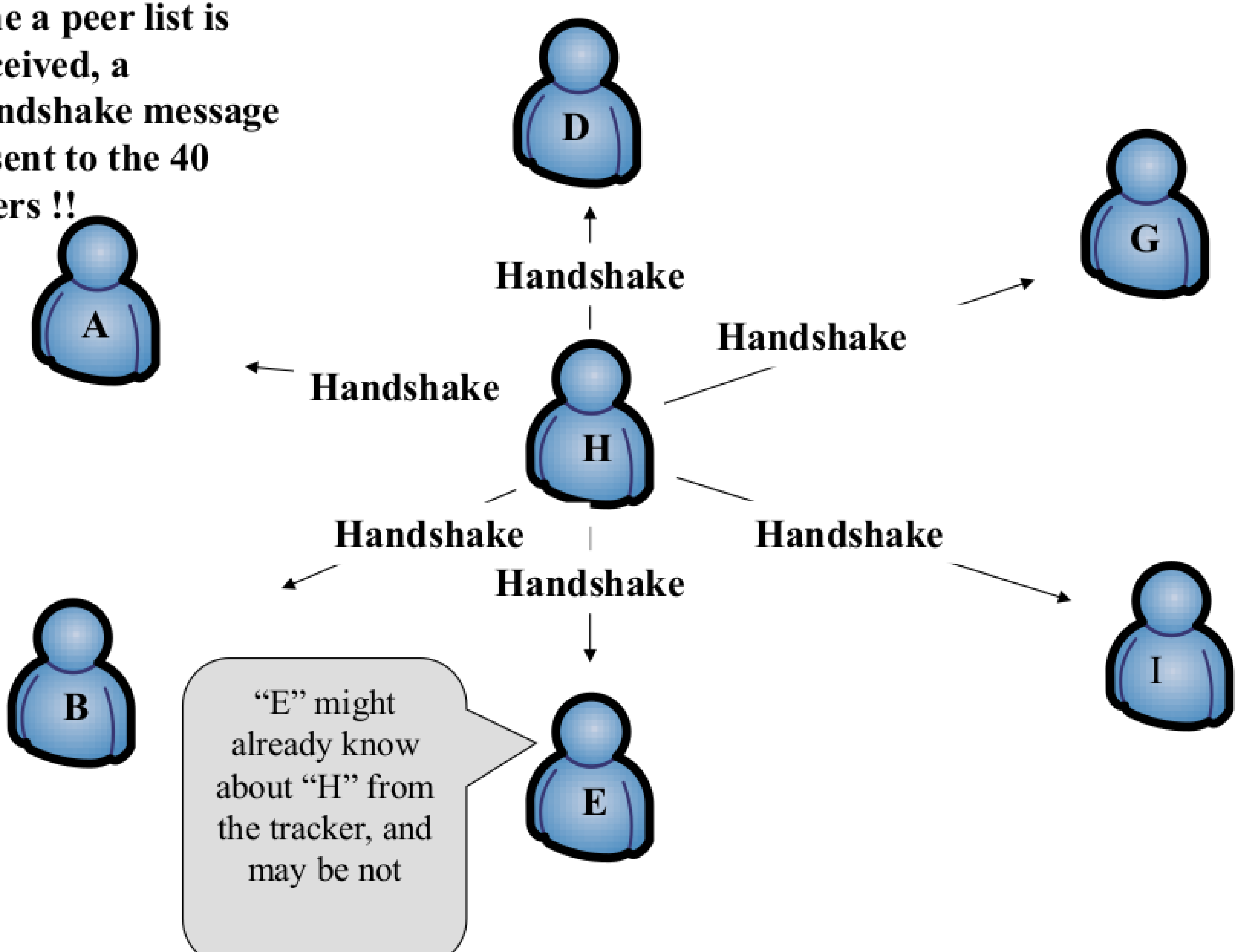
The sole role of the tracker in BitTorrent is to give a list of peers.

When asked for a list of peers, the tracker returns a subset of the peers, typically 40 peers

Peer also occasionally send statistics to him about what they are doing

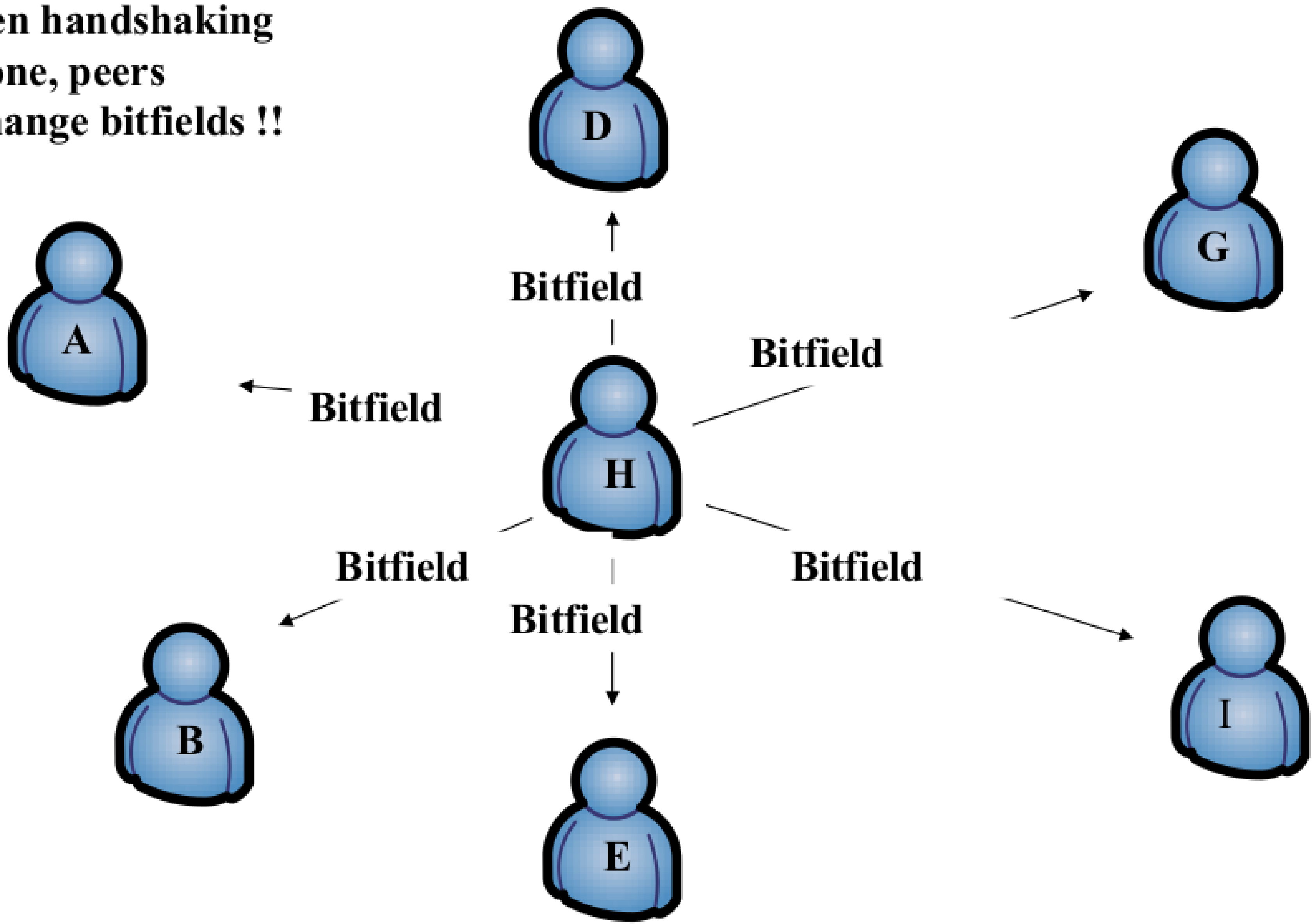
Handshaking

One a peer list is received, a handshake message is sent to the 40 peers !!



Exchanging Bitfields (1)

When handshaking is done, peers exchange bitfields !!



After the exchange..

- Every leecher knows exactly what the other peers have and vice versa
- Each link is labeled from both sides by two flags:
  - Choked
  - Interested