Research Group
Distributed Systems

Christian-Albrechts-Universität zu Kiel
Technische Fakultät

Distributed Systems
DHTs & BitTorrent

Olaf Landsiedel

Last Time

- Fault Tolerance III
  - Virtual Synchrony
  - Commit Protocols: 2PC, 3PC
  - Recovery from failures: Checkpoints & logs

Applications: This lecture and next two

- "Dark side"
  - DHTs
  - BitTorrent

Today

  - TOR

App II

- "White side"
  - Google file system

App II

  - Map Reduce
  - Amazon Dynamo

App III

Maybe Hadoop and Spark…

Today

- Distributed Hash Tables (DHTs)
  - We touched on them
    - In the lecture on “naming”
    - Maybe in our computer networks lecture
  - Today
    - Go deeper, focus on resilience, scalability, etc.
- BitTorrent (today) & TOR (next time)
  - I think you should know how these work

Part I

Distributed Hash Tables (DHTs)

Introduction

- What is a hash table?
  - Store for (key, value) pairs
    - (key, value) pair:
      example (“personnummer”, “address”)
    - Add, remove, lookup: O(1), efficient operations
    - Search for value: O(n), not efficient
  - Typical efficient operations
    - `put(key, value)`
    - `value = get(key)`
- What is a distributed hash table?

Introduction

- What is a distributed hash table?
  – The distributed version of a hash table ;-)
  – Each node stores a subset of (key, value) pairs
- Questions?
  – How to make this efficient?
    - log(n) or similar, operations for put, get
    - Scale to thousands of nodes
  – How to deal with churn?
    - Churn: node failures, nodes leaving, nodes joining, …
    - Adding redundancy

Selected DHTs

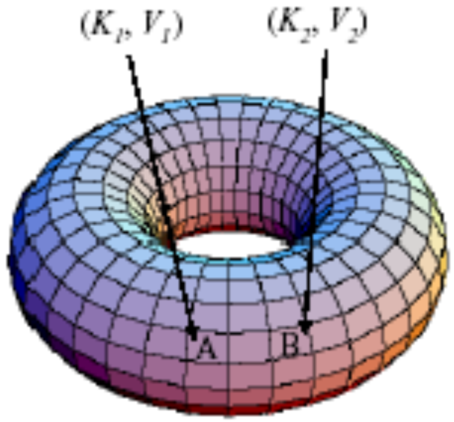
- Today, we discuss one DHT:
  - CAN: Content Addressable Network
    - Do not mix up with: CAN bus

- Chord
  - Another DHT
  - Already discussed in the naming lecture

Content Addressable Network (CAN)

- Each node & each data item
  – a unique *id* in an *d*-dimensional Cartesian space on a *d*-torus

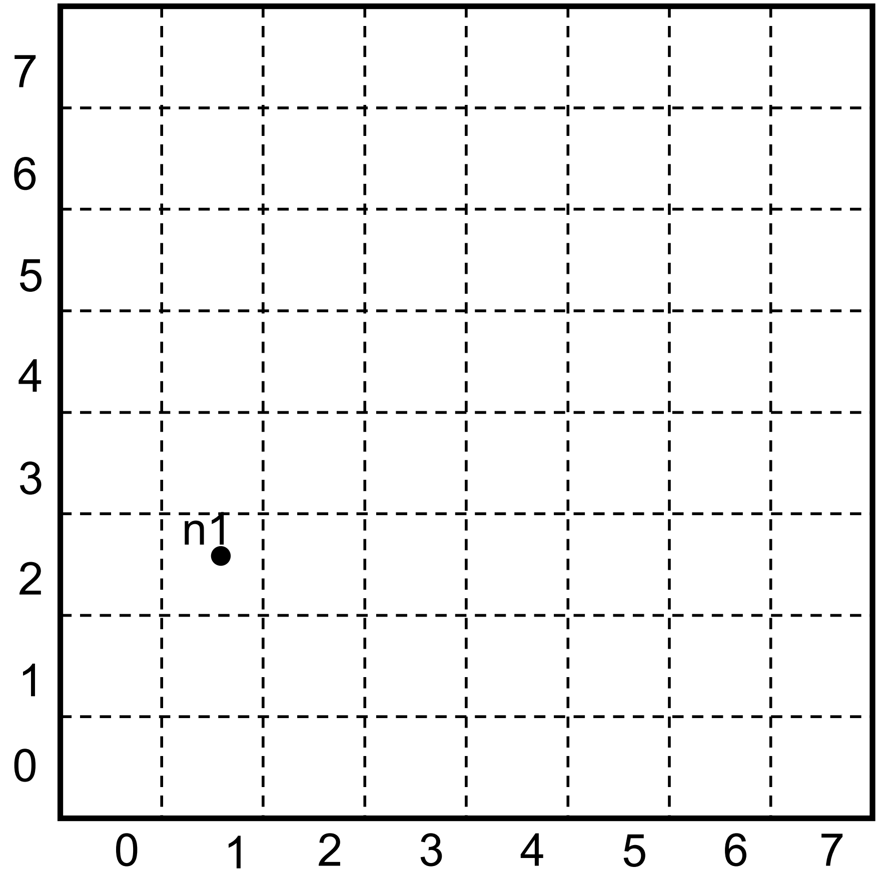
- Properties
  – Routing table size O(*d*)
  – Guarantees that an item is found in at most $d*n^{1/d}$ steps
    - *n* is the total number of nodes



2-torus

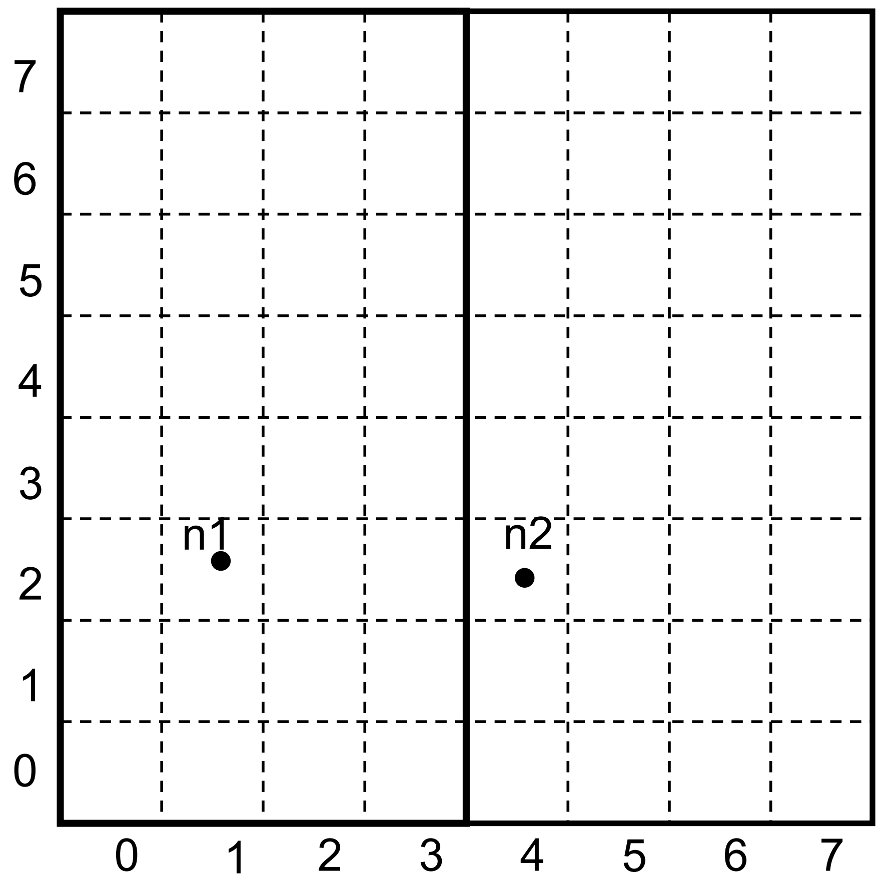
CAN Example: Two Dimensional Space

- Assume d=2
  - For simplicity
- Address space divided between nodes
- Together nodes cover the entire space
- Each node covers either a square or a rectangular area Example:
  - Node n1:(1, 2) first node that joins → cover the entire space
- Note: we use 2-dim for simplicity
  - It can have arbitrary dimensions



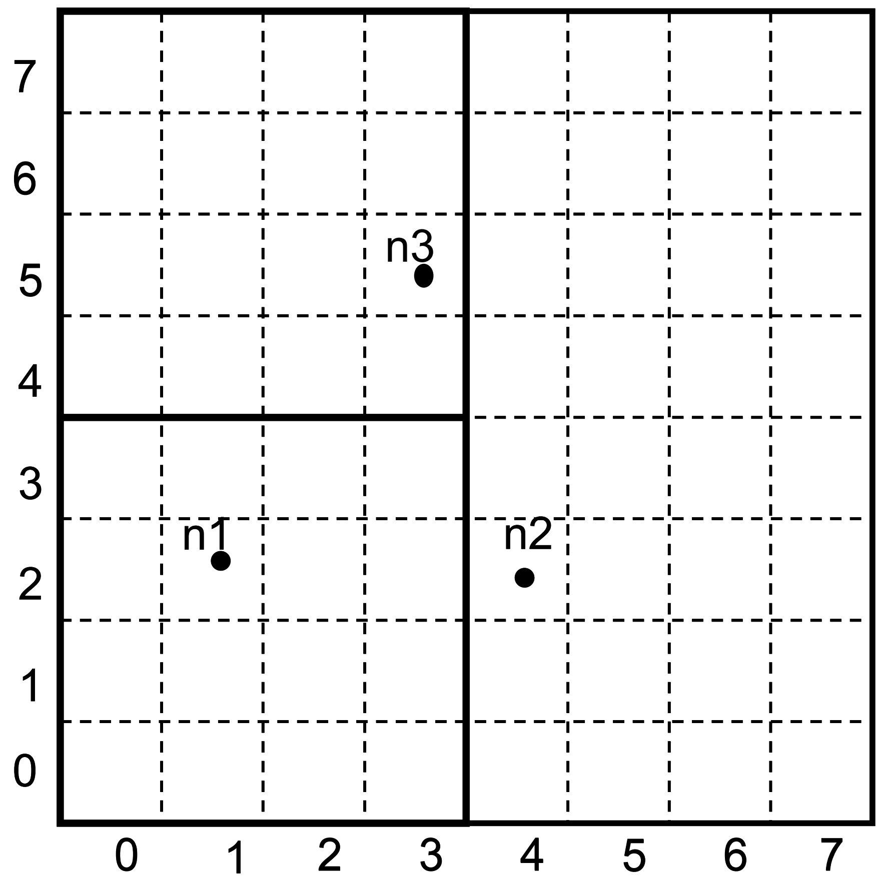
CAN Example: Two Dimensional Space

- Node n2:(4, 2) joins → space is divided between n1 and n2



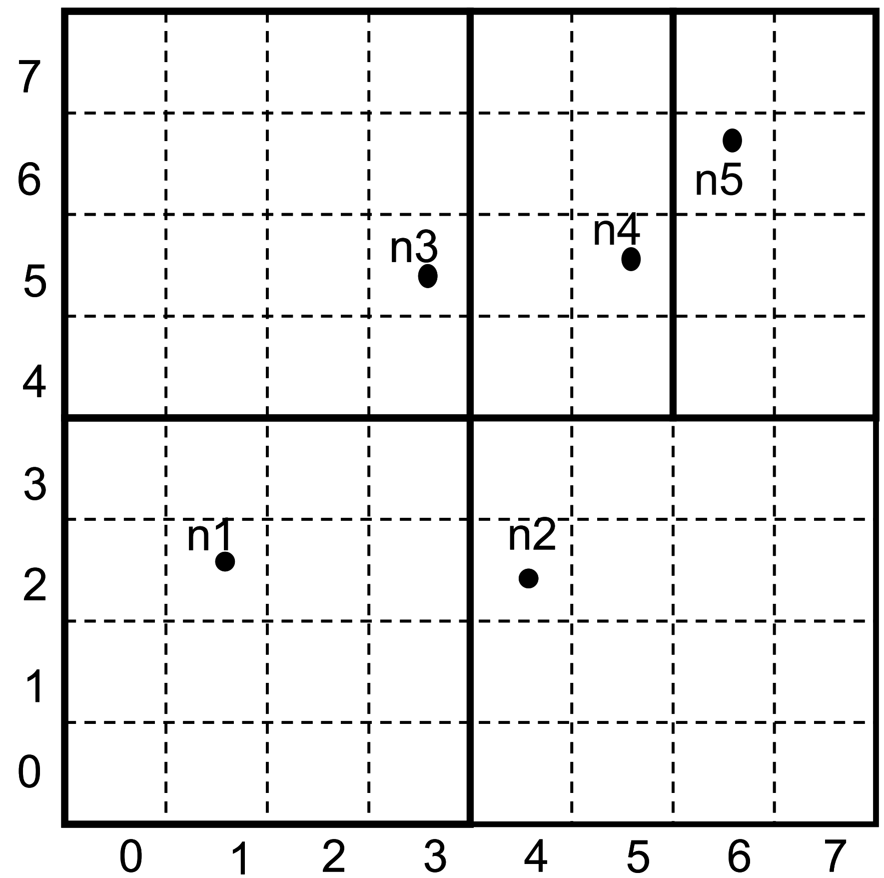
CAN Example: Two Dimensional Space

- Node n3: (3, 5) joins
  - → space is divided between n1 and n3



CAN Example: Two Dimensional Space

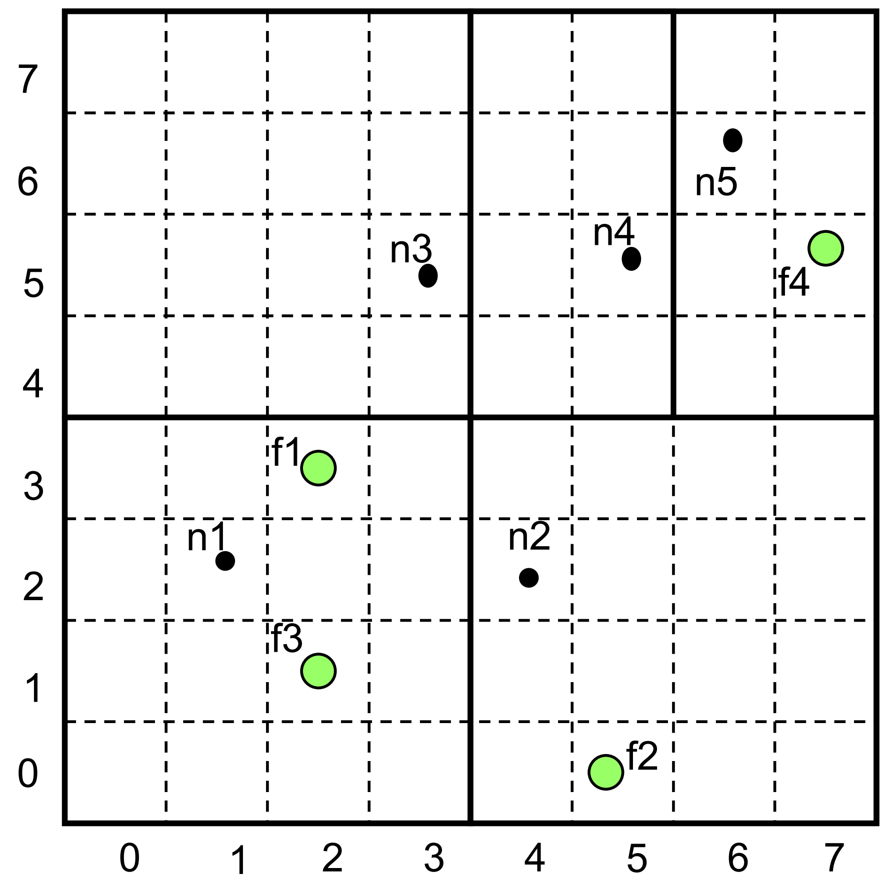
- Nodes n4:(5, 5) and n5:(6,6) join



CAN Example: Two Dimensional Space

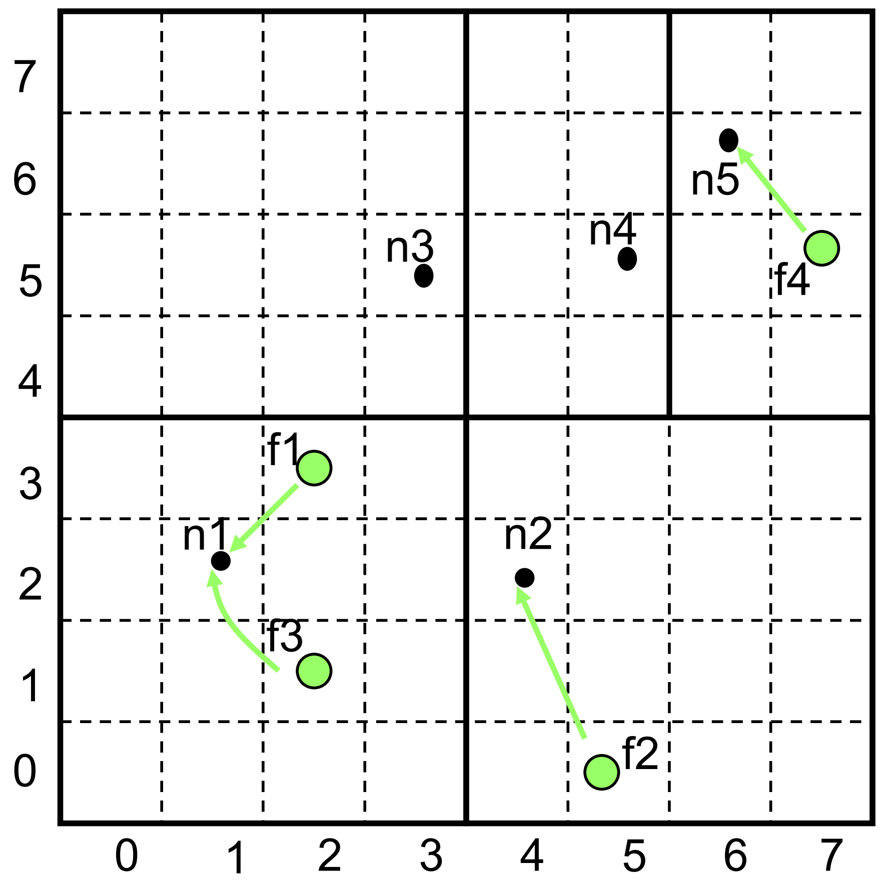
- Nodes: n1:(1, 2); n2:(4,2); n3:(3, 5); n4:(5,5);n5:(6,6)
- Items: f1:(2,3); f2:(5,0); f3:(2,1); f4:(7,5);

- Which nodes stores which data item?



CAN Example: Two Dimensional Space

- Each item is stored by the node who owns its mapping in the space

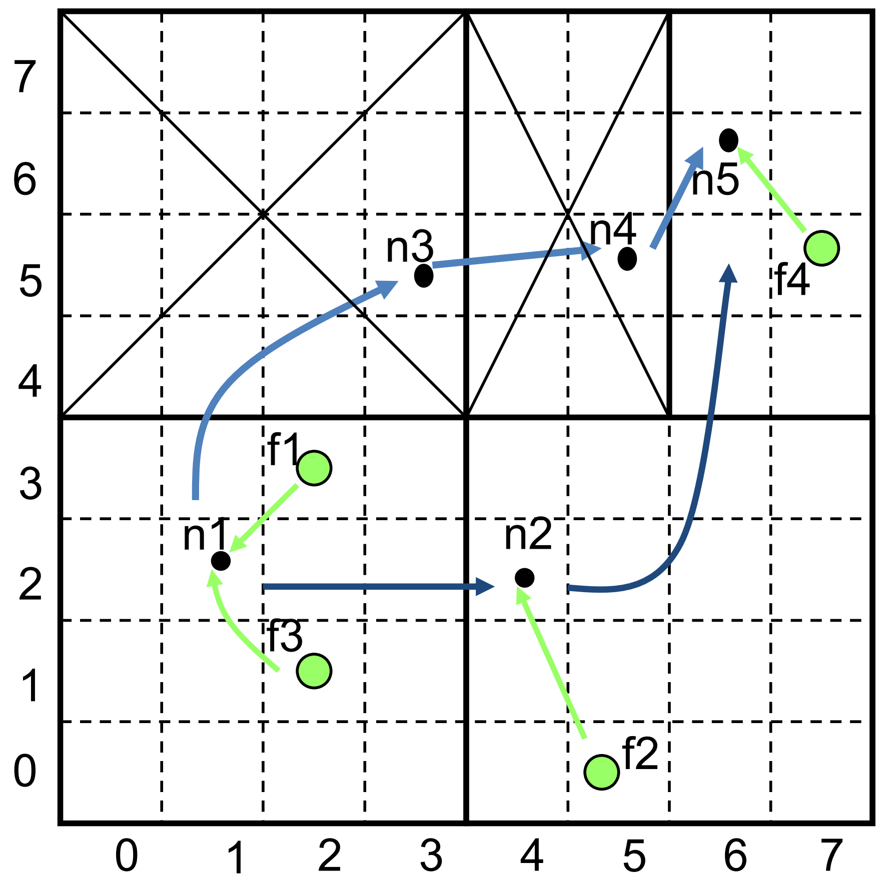


CAN until now

- We know how
  - The coordinate space is split-up between the nodes
  - A node responsible for a data item is selected
- Next
  - Queries: lookup a data item
  - Join: find the area you are responsible for
  - Departure: how nodes leave

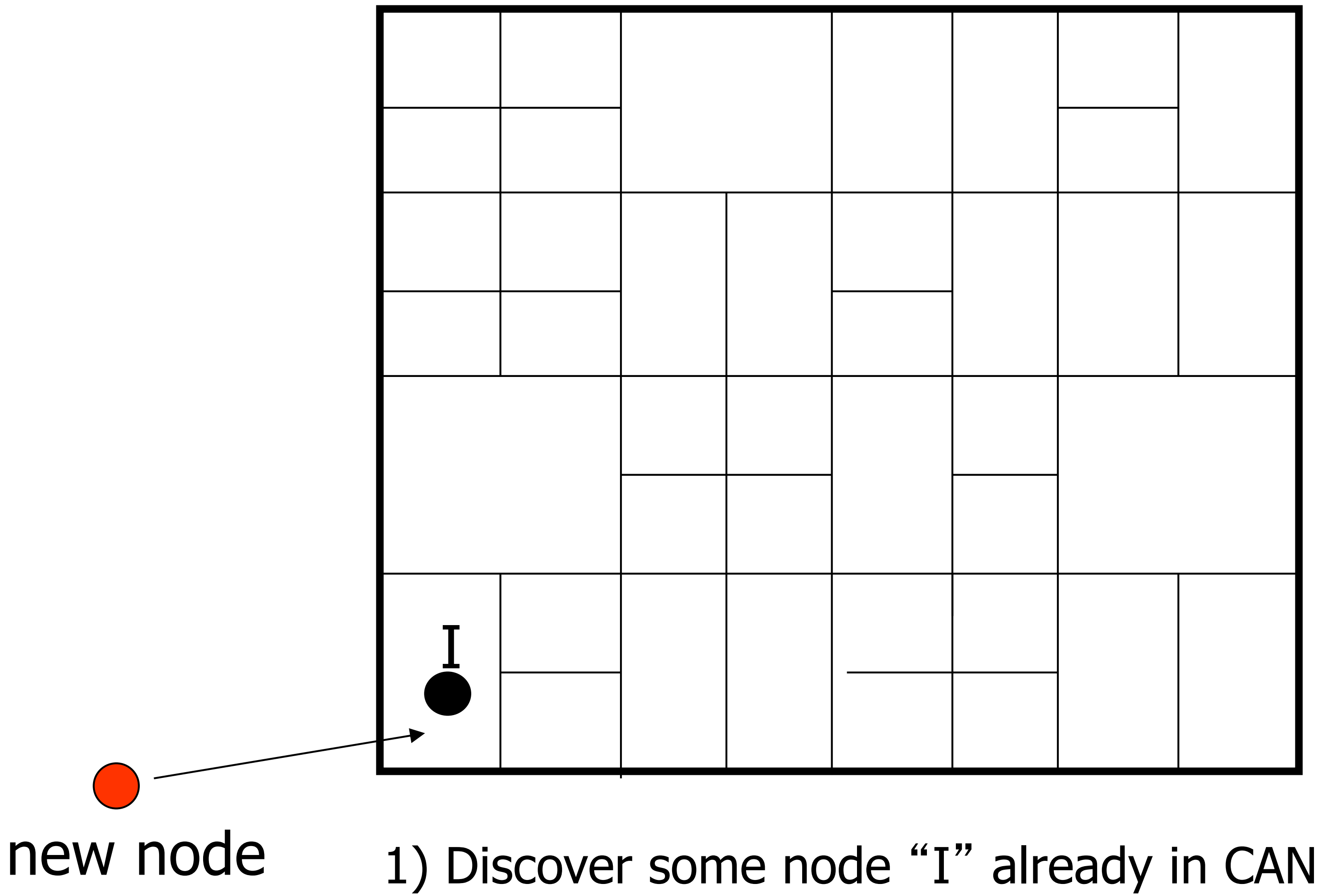
CAN: Query Example

- Each node knows its neighbors in the *d*-dimensional space
- Forward query to the neighbor that is closest to the query *id*
- Example: assume n1 queries f4
- Can route around some failures



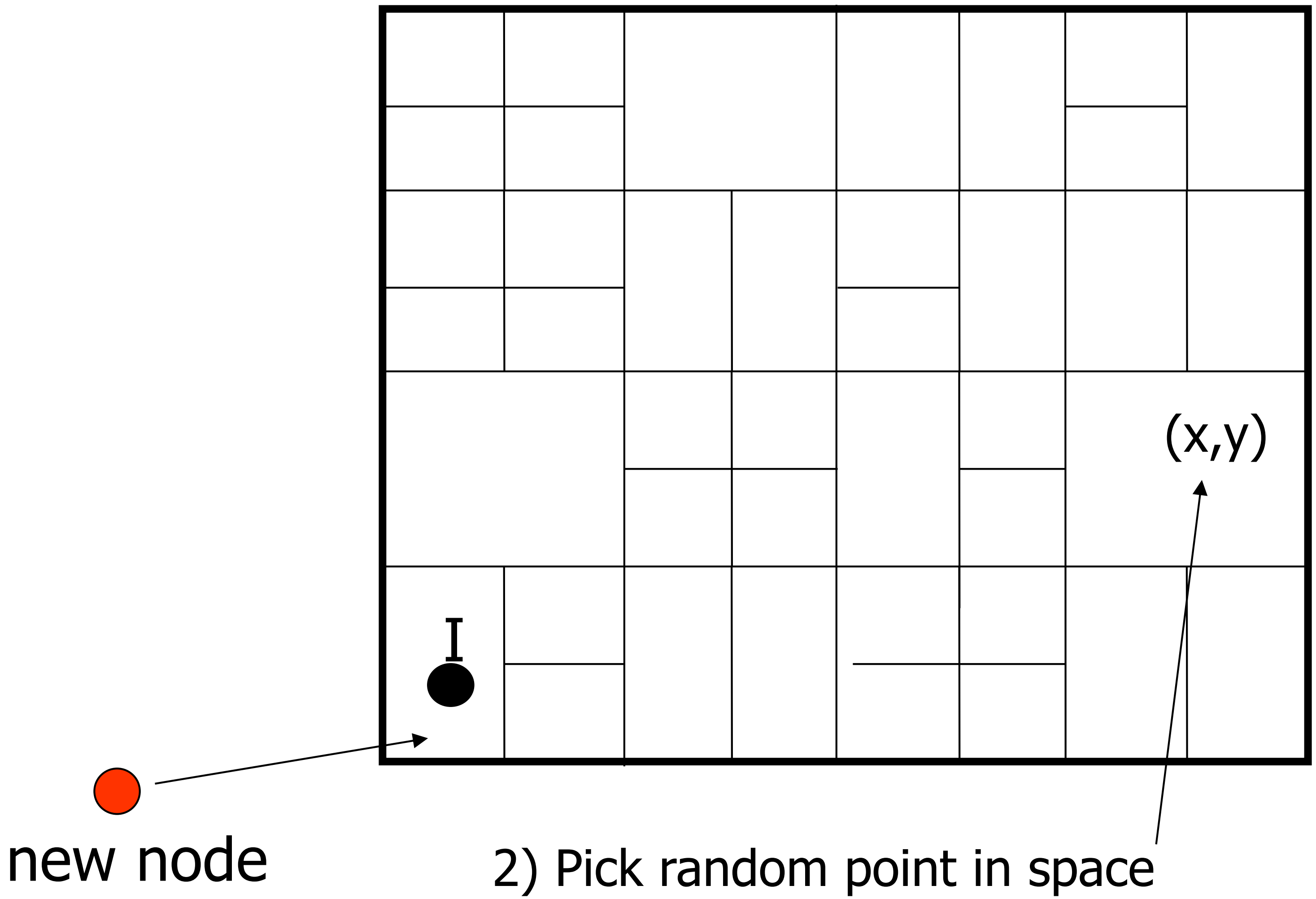
- Note:
  - Coordinate space wraps around
    - Can go from 7 to 0 and vice versa

CAN: Node Joining

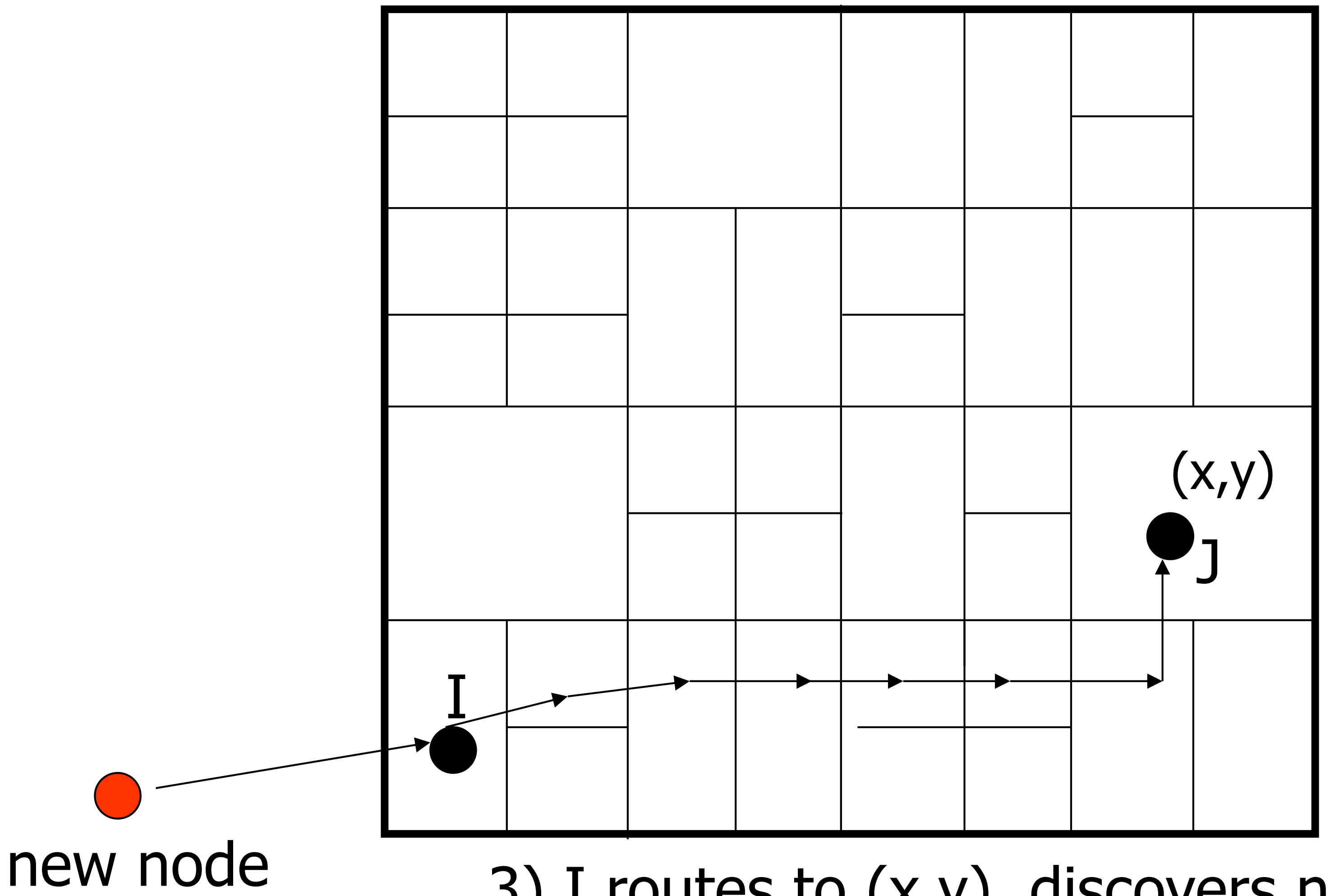


1) Discover some node "I" already in CAN

CAN: Node Joining

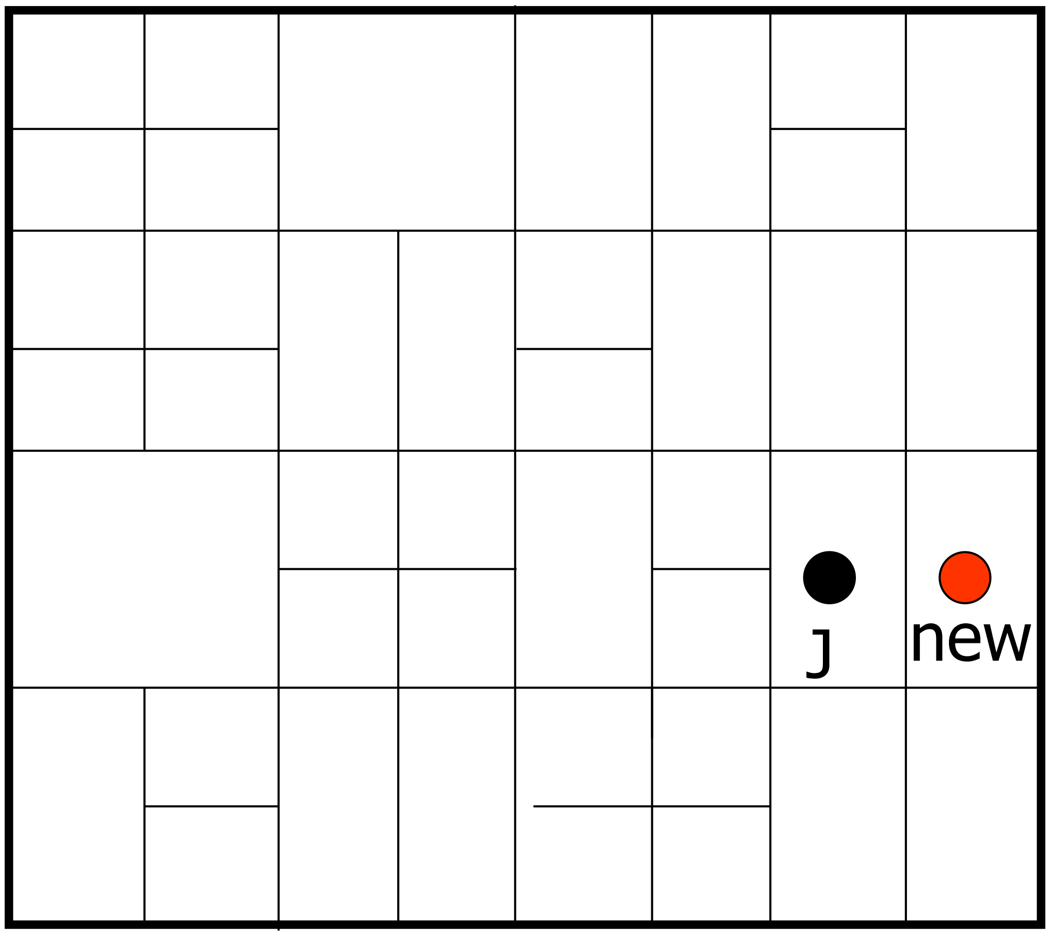


CAN: Node Joining



3) I routes to (x,y), discovers node J

CAN: Node Joining



4) split J's zone in half… new node owns one half

Node Departure

- What shall a node do before leaving?

- How do deal with nodes that just leave?
  - Crashed, disconnected, …

Node Departure

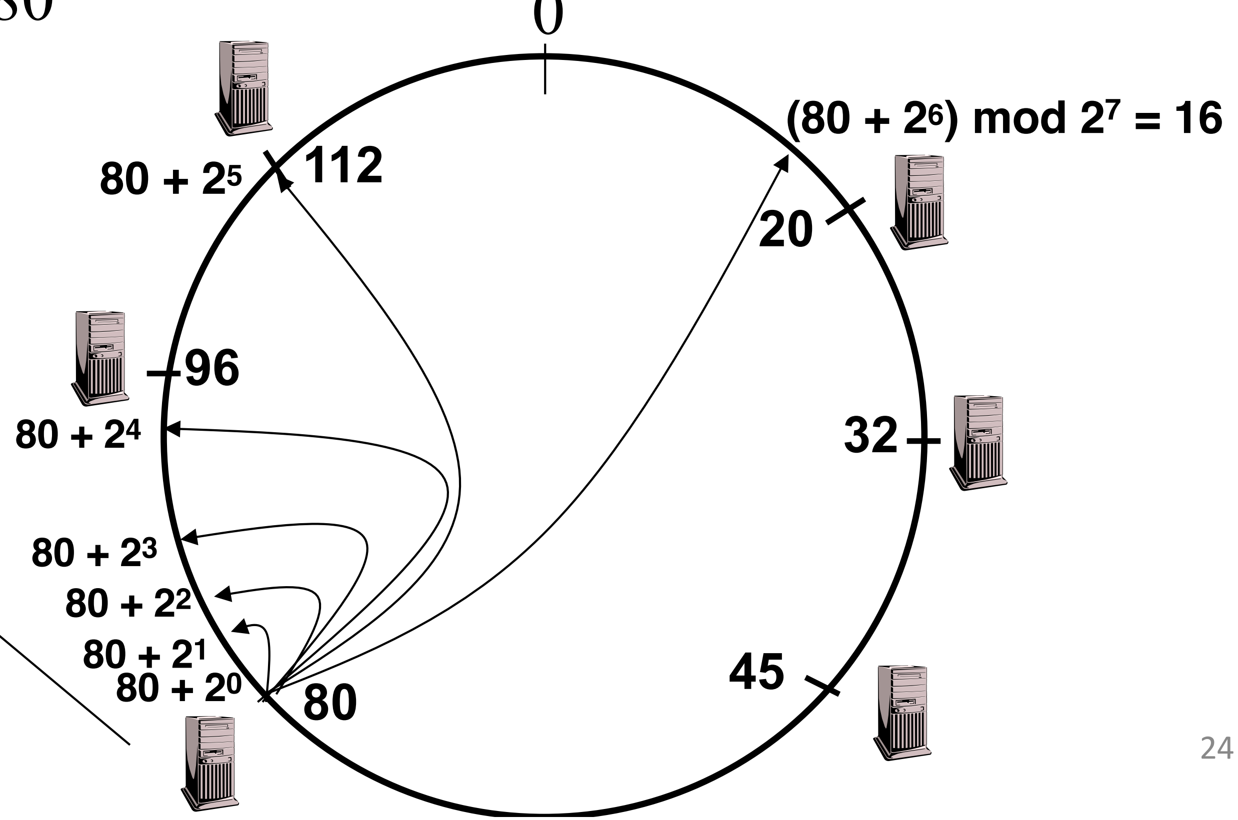
- Controlled leave
  - Node explicitly hands over its zone and the associated (key,value) database to one of its neighbors
- Sudden leave
  - Incase of network failure this is handled by a take-over algorithm
    - Ping all neighbors regularly, on timeout: node has left
    - Neighboring node takes over the space
  - Problem
    - Data is lost
  - Solution
    - Every node has a backup of its neighbors

Chord

- Another DHT
  - Discussed in the naming lecture

Finger Table at 80

i	$ft[i]$
0	96
1	96
2	96
3	96
4	96
5	112
6	20



Summary

- Distributed Hash Tables are a key component of scalable and robust overlay networks
- CAN: O(d) state, $O(d*n^{1/d})$ distance
- Chord: O(log n) state, O(log n) distance
- Both can achieve stretch < 2
  - Stretch: overhead of the DHT route
- Simplicity is key

DHT: Discussion

- What do they provide to the application?
  - put / get interface
- What is this good for?
  - key,value storage and lookup
- What is this not good for?
  - Other operations such as search
    - Example: All items that start with “XYZ”

DHT: Discussion

- Scalability
  - How is it achieved?
    - small state per node
    - log(n) or similar for put, get
- Robustness
  - How is it achieved?
    - replication on neighboring nodes

Questions

- Recap
  - CAN: lookup, join, leave?

- olafland.polldaddy.com/s/dhts
  - DHTs are very scalable
  - DHTs have a single point of failure
  - DHTs provide: put, get, put+get, put+get+search
  - Lookup complexity in Chord and CAN?
  - DHTs provide a delete operation

Questions

- Recap
  - CAN: lookup, join, leave?

- olafland.polldaddy.com/s/dhts
  - DHTs are very scalable
    - yes
  - DHTs have a single point of failure
    - no
  - DHTs provide: put, get, put+get, put+get+search
    - Put+get
  - Lookup complexity in CAN?
    - $O(d*n^{1/d})$
  - DHTs provide a delete operation
    - No, soft-state: items time out when not refreshed (but we did not discuss this)

DHTs

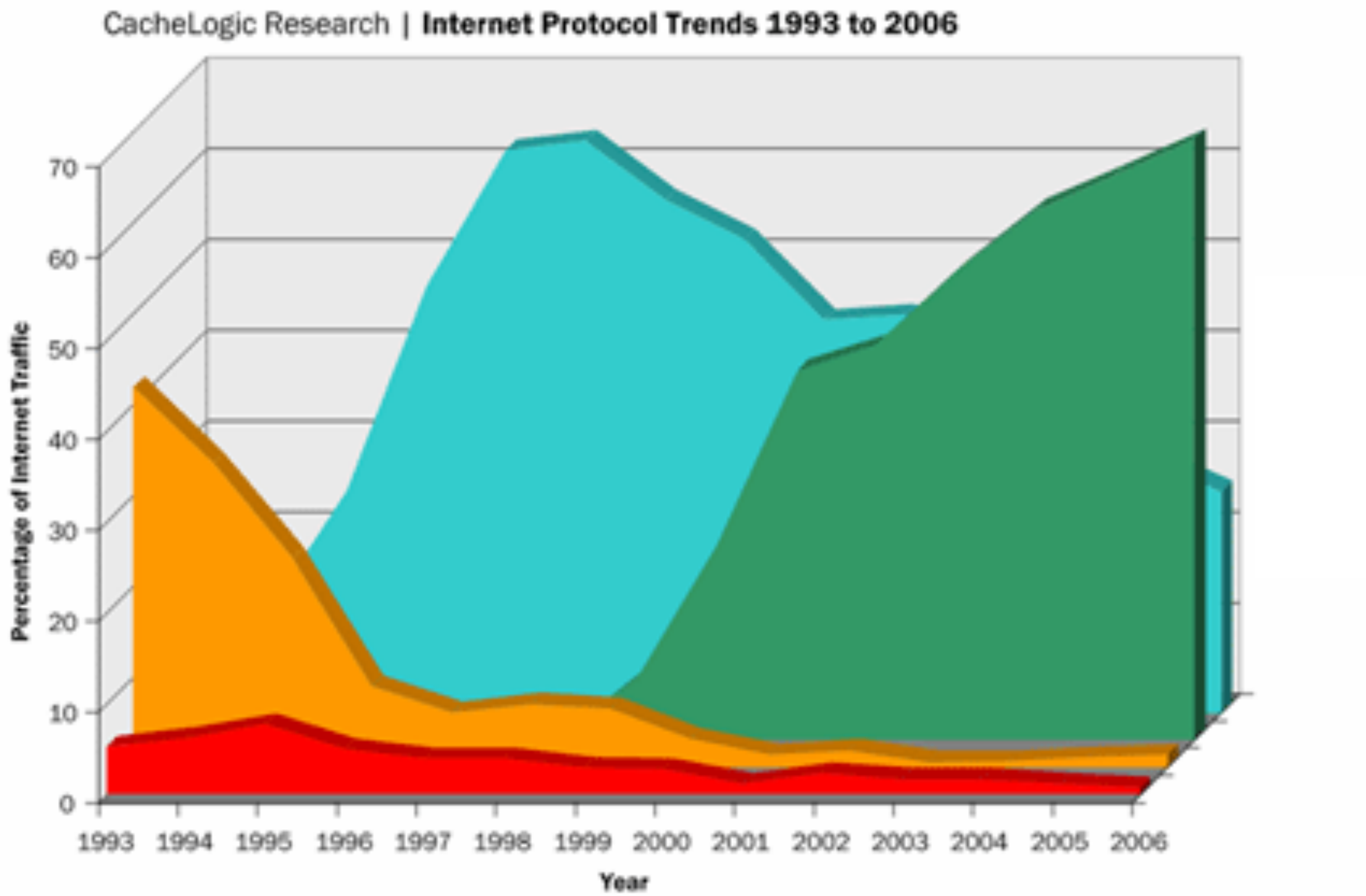
- We discussed the basic concept

- Today: DHTs in many forms and modifications
  - Two examples
    - BitTorrent (next topic)
    - Amazon Dynamo (next time)

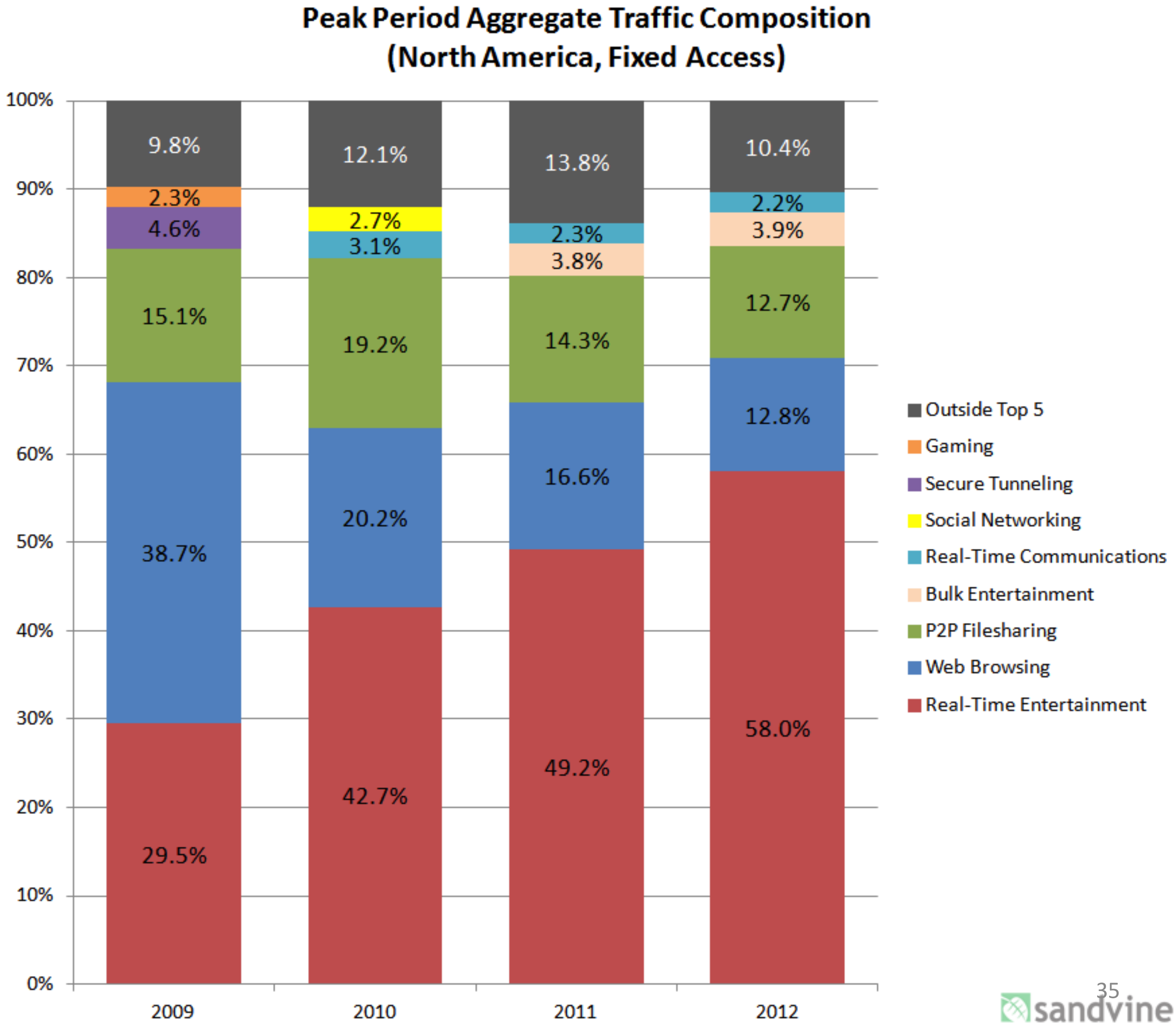
Part II

BitTorrent

Internet Traffic



Internet Traffic Today



sandvine

Internet Traffic

- Peer-To-Peer:
  - Mainly BitTorrent (today)

BitTorrent

- Peer-To-Peer content distribution
  - "File sharing"
- Written by Bram Cohen (in Python) in 2001
- Concept
  - Each file split into smaller pieces
  - Nodes request desired pieces from neighbors
  - Pieces not downloaded in sequential order
  - Encourages contribution by all nodes

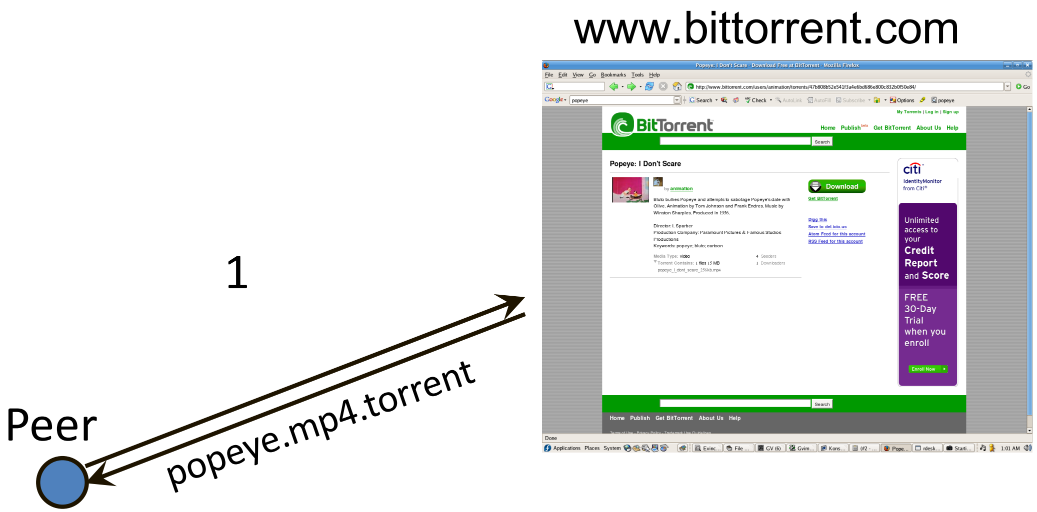
BitTorrent Swarm

- Swarm
  - Set of peers all downloading the same file
  - Organized as a random mesh
    - List of peers: from tracker (to be discussed later)
    - Discovered by peer exchange protocol (PeX)
      - Nodes exchange lists of peers
  - Each node knows list of pieces downloaded by neighbors
  - Node requests pieces it does not own from neighbors
    - Exact method explained later

How a node enters a swarm for file “popeye.mp4”

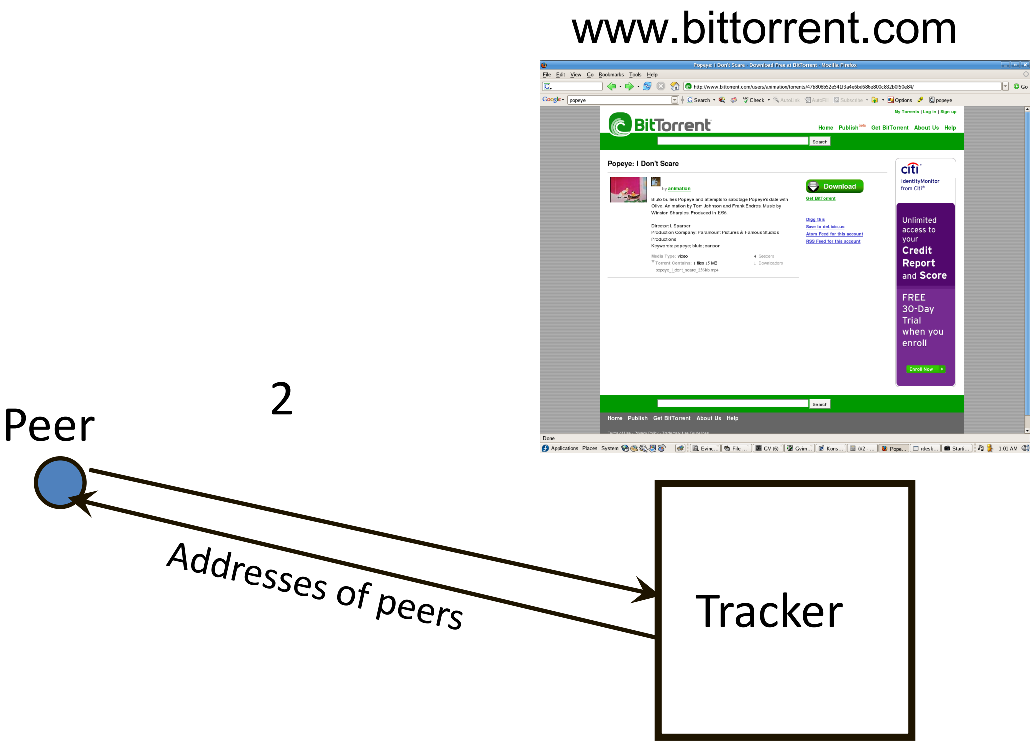
- File popeye.mp4.torrent hosted at a (well-known) webserver
- The .torrent has address of tracker for file
- The tracker, which runs on a webserver as well, keeps track of all peers downloading file

How a node enters a swarm for file "popeye.mp4"



- File popeye.mp4.torrent hosted at a (well-known) webserver
- The .torrent has address of tracker for file
- The tracker, which runs on a webserver as well, keeps track of all peers downloading file

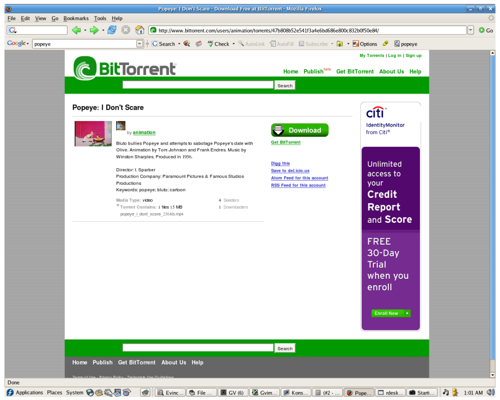
How a node enters a swarm for file "popeye.mp4"

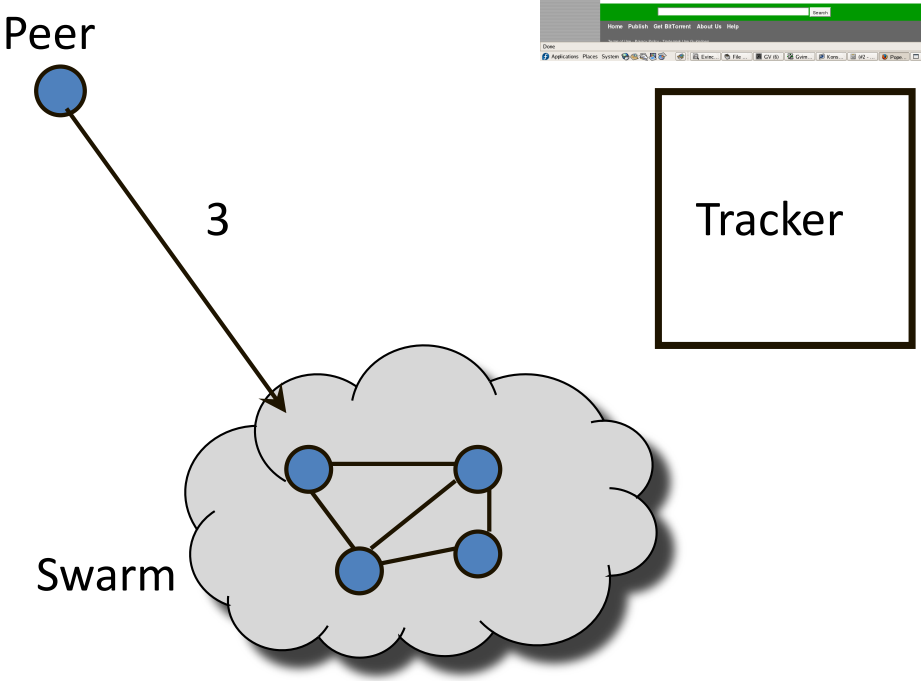


- File popeye.mp4.torrent hosted at a (well-known) webserver
- The .torrent has address of tracker for file
- The tracker, which runs on a webserver as well, keeps track of all peers downloading file

How a node enters a swarm for file "popeye.mp4"

www.bittorrent.com





- File popeye.mp4.torrent hosted at a (well-known) webserver
- The .torrent has address of tracker for file
- The tracker, which runs on a webserver as well, keeps track of all peers downloading file -> swarm

Contents of .torrent file

- URL of tracker
- Piece length
  - Configured by the first uploaded
  - Common value 256 KB
  - 512KB or 1024 KB for larger files
- SHA-1 hashes of each piece in file
  - For reliability
- “files” – allows download of multiple files

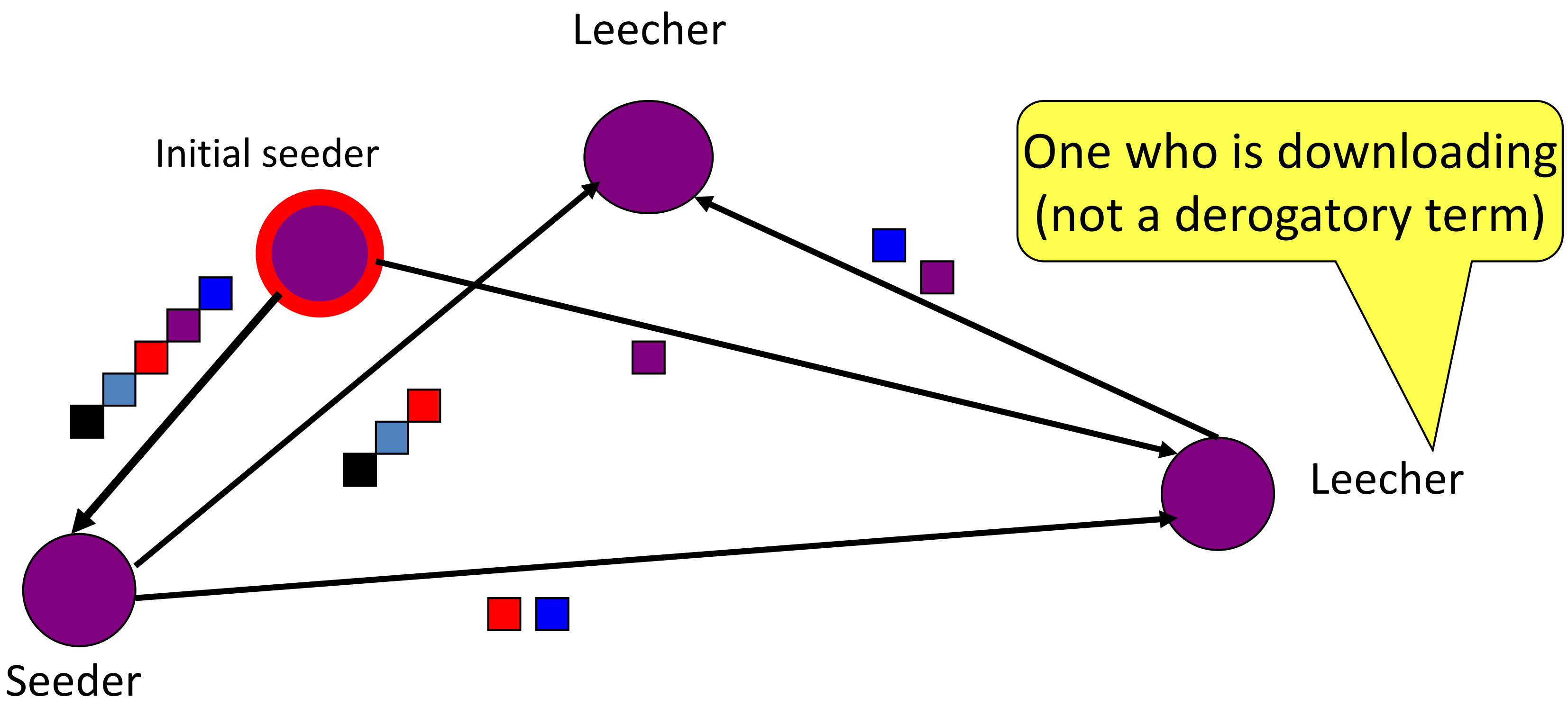
Terminology

- Seed: peer with the entire file
  - Original Seed: The first seed
- Leech: peer that's downloading the file
  - Fairer term might have been "downloader"
- Sub-piece: Further subdivision of a piece
  - The "unit for requests" is a subpiece
  - But a peer uploads only after assembling complete piece

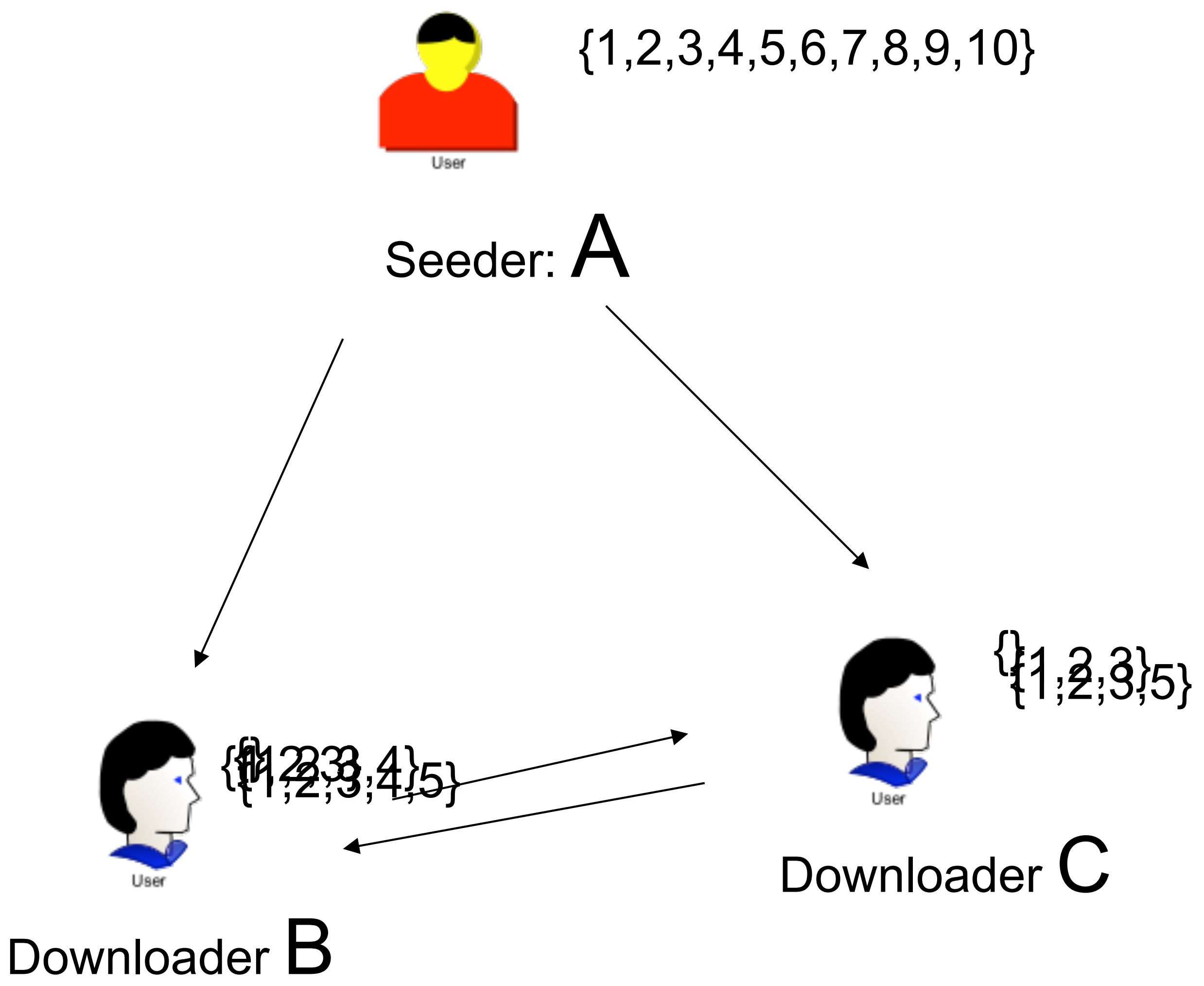
BitTorrent Lingo

Seeder = a peer that provides the complete file.

Initial seeder = a peer that provides the initial copy.



Simple example



Pieces

- A file is split into many pieces
  - Which pieces shall a node request?
  - To ensure good performance
    - For itself?
    - For all nodes?

Peer-peer transactions: Choosing pieces to request

- Rarest-first: Look at all pieces at all peers, and request piece that is owned by fewest peers
  - Increases diversity in the pieces downloaded
    - Avoids case where a node and each of its peers have exactly the same pieces
    - Increases throughput
  - Increases likelihood all pieces still available even if original seed leaves before any one node has downloaded entire file

Choosing pieces to request

- Random First Piece:
  - When peer starts to download, request random piece.
    - So as to assemble first complete piece quickly
    - Then participate in uploads
  - When first complete piece assembled, switch to rarest-first

Choosing pieces to request

- End-game mode:
  - When requests sent for all sub-pieces, (re)send requests to all peers.
  - To speed up completion of download
  - Cancel request for downloaded sub-pieces

Free Riding

- Goal
  - Avoid free-riding
    - Nodes only downloading
      - If everybody does this, the system would not work
  - How to fix?
    - Tit-for-tat:
      - e.g. equivalent retaliation
      - I give if you give, too

Tit-for-tat as incentive to upload

- Want to encourage all peers to contribute
- Peer *A* said to choke peer *B* if it (*A*) decides not to upload to *B*
- Each peer (say *A*) unchokes at most 4 *interested* peers at any time
  - The three with the largest upload rates to *A*
    - Where the tit-for-tat comes in
  - Another randomly chosen (Optimistic Unchoke)
    - To periodically look for better choices
    - Idea: If I unchocked a node, it might in turn unchoke me (-> I get more data)

Anti-snubbing

- A peer is said to be snubbed if each of its peers chokes it
- To handle this, snubbed peer stops uploading to its peers

➢ Optimistic unchoking done more often
  – Hope is that we will discover a new peer that will upload to us

Why BitTorrent took off

- Good design
  - Scalable
  - Search is "outsourced"
    - Search is difficult in Peer-To-Peer Systems
  - Allows uploading from hosts that have downloaded parts of a file
  - Avoids free-riding

Why BitTorrent took off

- Many Practical Reasons
  - Working implementation (Bram Cohen) with simple well-defined interfaces for plugging in new content
  - Many recent competitors got sued / shut down
    - Napster, Kazaa, …
    - Search was slow and unreliable in these
  - Does not do "search" per se
    - Users use well-known, trusted sources to locate content
      - i.e., Internet search engines
    - Avoids the pollution problem, where garbage is passed off as authentic content

Pros and cons of BitTorrent

- Pros
  - Proficient in utilizing partially downloaded files
  - Discourages "freeloading"
    - By rewarding fastest uploaders
  - Encourages diversity through "rarest-first"
    - Extends lifetime of swarm
- Works well for "hot content"

Pros and cons of BitTorrent

- Cons
  - Assumes all interested peers active at same time; performance deteriorates if swarm "cools off"
  - Even worse: no trackers for obscure content

Pros and cons of BitTorrent

- Dependence on centralized tracker: pro/con?
  - ☹ Single point of failure: New nodes cannot enter swarm if tracker goes down
  - Lack of a search feature
    - ☺ Prevents pollution attacks
    - ☹ Users need to resort to out-of-band search: well known torrent-hosting sites / plain old web-search

"Trackerless" BitTorrent

- To be more precise, "BitTorrent without a centralized-tracker"
- Uses a **<u>Distributed Hash Table</u>** (Kademlia DHT)
  - Key, value pair: "tracker file name", "tracker file"
  - Use simple put/get to upload/retrieve file
  - DHT is resilient to node failure: uses replication etc.
- Tracker DHTs run by normal end-hosts
  - The BitTorrent client
    - Trackerless BitTorrent is supported by most BitTorrent clients
  - Not a web-server anymore
    - Remove single point of failure
  - Node responsible for its id in the DHT
    - As discussed for CAN, Chord etc.

BitTorrent: Ethical Challenges

- Ethical challenges?
  - Use to distributed legal and illegal content
  - Hard to remove illegal content
    - No central authority / point of control
    - Pro's & cons of this design
      - Hard to remove illegal content
        - » Copyrighted movies etc.
      - Hard to remove content that some governments etc. do not want to be distributed
        - » Documentations of corruption etc.

BitTorrent

- BitTorrent Summary
  - Basic
  - Tit-for-tat
  - Trackerless

- http://olafland.polldaddy.com/s/bittorrent
  - In BitTorrent you can only share legal data
  - In BitTorrent you can search for the content you are interested in
  - BitTorrent is scalable
  - BitTorrent: Each swarm has a single point of failure
  - "Trackerless" BitTorrent: Each swarm has a single point of failure
  - Free riding in BitTorrent

BitTorrent

- BitTorrent Summary
  - Basic
  - Tit-for-tat
  - Trackerless

- http://olafland.polldaddy.com/s/bittorrent
  - In BitTorrent you can only share legal data
    - no
  - In BitTorrent you can search for the content you are interested in
    - no
  - BitTorrent is scalable
    - yes
  - BitTorrent: Each swarm has a single point of failure
    - Yes, the tracker
  - “Trackerless” BitTorrent: Each swarm has a single point of failure
    - No, as now the tracker is in the DHT which includes redundancy
  - Free riding in BitTorrent
    - Possible to a (very) small amount (hope for optimistic unchoking)

Next Time

- More applications
  - TOR
  - Google file system

Questions?

- In part, inspired from / based on slides from
  - Scott Shenker
  - Ion Stoica
  - Vivek Vishnumurthy
  - Sukumar Ghosh
  - Vitaly Shmatikov