Distributed Systems:
Peer to Peer Networks

Anca Jurcut
E-mail: anca.jurcut@ucd.ie

School of Computer Science and Informatics
University College Dublin
Ireland

From Previous Lecture…

- **Andrew File System**
- **Peer to Peer Networks:** recommended reading <Chapter 10: "Peer – To- Peer Systems"> of *"Distributed Systems, Concepts and Design" by George Coulouris, Jean Dollimore and Tim Kindberg, Fourth edition, 2005*
  - What is P2P?
    - A set of networked devices that have **equal responsibility** and which **share resources and workload** as necessary
    - Each device is both a **client** and a **server**
  - Examples of P2P Systems
    - File Sharing Applications e.g. Gnutella, BitTorrent, Kazaa, …
    - Instant Messengers and Telephony e.g. Skype, Yahoo, MSN
    - Data Processing Services e.g. Seti@Home
  - Characteristics of P2P Systems
    - The key problem to be addressed when building a P2P system is **how to locate the resources** that you need

From Previous Lecture…

- Accessing (Locating) Resources in P2P Systems
- Types of P2P Systems: 3 main Types
  - **Centralized Systems:** peer connects to server which coordinates and manages communication. First generation systems addressed this problem through a **centralised server** that contained a global resource list (not scalable, legal issues).
    E.g. SETI@home
  - **Decentralized Systems**: peers run independently with no central services; discovery is decentralized and communication takes place between the peers. Second generation systems **decentralised the resource list** to improve robustness (and to sidestep the legal issues ;-)
    E.g. Gnutella, Pastry
  - **Hybrid systems**: peers connect to a server to discover other peers; peers manage the communication themselves (e.g. Napster). Third generation systems have focused on making file sharing an **anonymous activity** and on techniques for improving **resource search** speed.

- **P2P Middleware Systems**
  - Aims: to be **application independent; scalability:** to share resources, storage and data present in computers *at the edges of the internet* on a global scale
  - Provide reference architecture that allowed researchers to focus of specific P2P issues

From Previous Lecture…

- P2P Middleware systems are based on 2nd generation architectures
- **Resource discovery (searching):** algorithms used to carry out the search *within P2P middleware systems* are known as **routing overlay algorithms**
- **Routing Overlay**
  - **Basic idea**
  - **Resource GUIDS (**globally unique identifier(s))
  - **Distributed Hash Tables (DHT)**

Distributed Systems:

Routing Overlays versus IP Routing

IP Routing - The Basics

- **IP routing** = primary communication mechanism for the Internet, which operates at the Network Layer.
  - Set of Protocols that determine the path that data follows in order to travel across multiple networks from its source to its destination.
  - Data is routed through a series of routers, and across multiple networks.
  - Router = device that connects two or more networks.
  - IP Protocols enable routers to build up a forwarding table that correlates final destinations with next hop addresses.
  - Each data packet contains address information.
    - Router uses it to determine:

      - if the source and destination are on the same network

      - if the data packet must be transferred from one network to another

Routing Overlays vs IP Routing

- **1. Scalability**
  - **IP Routing:** IPv4 is limited to 2^{32} addressable nodes (in reality the number is about 3 billion).
    - Due to its hierarchical nature, many of these addresses cannot be used.
    - Due to the explosion of the Internet, this isn’t big enough
    - IPv6 is a potential solution (2^{128} addressable nodes), but suffers from the same hierarchy issue.
  - **Routing Overlays:** P2P systems can address more objects.
    - The GUID name space is very large (> 2^{128}).
    - Uses a flat structure - all addresses can be used.

Routing Overlays vs IP Routing

- **2. Load Balancing**

  - **IP Routing:** Loads on routers are determined by network topology and associated traffic patterns.
  - **Routing Overlays:** Object locations can be randomized and hence traffic patterns are not determined by the network topology.

- **3. Network Dynamics (Addition/Deletion of Nodes)**

  - **IP Routing:** IP routing tables are updated asynchronously on a best-efforts basis with time constants of the order of one hour.
  - **Routing Overlays:** Routing tables can be updated synchronously or asynchronously with fractions of a second delays.

Routing Overlays vs IP Routing

- **4. Fault Tolerance**
  - **IP Routing:** Redundancy is designed into an IP network by its managers, ensuring tolerance of a single router or network connectivity failure.
    - n-fold tolerance is expensive
  - **Routing Overlays:** Routes and object references can be replicated n-fold, ensuring tolerance of n failures of nodes or connections.
- **5. Target Identification**
  - **IP Routing:** Each IP address maps to exactly one target node.
  - **Routing Overlays:** Messages can be routed to the nearest replica of a target node.

Routing Overlays vs IP Routing

- **6. Security and Anonymity**

  - **IP Routing:** Addressing is only secure when all nodes are trusted.
    - Anonymity for the owners of addresses is not achievable.

  - **Routing Overlays:** Security can be achieved even in environments with limited trust.
    - A limited degree of anonymity can be provided.

Distributed Systems:
Peer to Peer Systems – Summary

Summary so far …

- P2P architectures were first shown to support very large scale data sharing through *Napster* and its descendants for digital music sharing.
- Their use broke copyright laws.
- However, that does not diminish their technical significance!

- Limitations (i.e. copyright infringement) within centralised systems such as *Napster* lead to the developed of more sophisticated decentralized systems such as *Gnutella.*

Summary so far …

- Subsequent research resulted in P2P middleware platforms that deliver requests to data objects regardless of where they are located on the Internet.
  - Objects are addressed using GUID's (pure names containing no IP addresses - thus providing anonymity).
  - Objects are placed at nodes using some mapping infrastructure (e.g. DHT model) that is specific to each middleware system.
  - Delivery is performed by a ***routing overlay algorithm*** in the middleware (*application layer*).
    - Maintains routing tables and forwarding requests along a route which is determined by calculating the distance according to the chosen mapping function.

Summary so far …

- P2P Middleware platforms adds
  - **Integrity guarantees** based on the use of a secure hash function to generate the GUID's
  - **Availability guarantees** based on the replication of objects at several nodes and on fault tolerant routing algorithms

- These platforms have been deployed in several large scale pilot applications and continue to be refined and evaluated
  - Recent results suggest that the technology is ready for deployment in applications involving large numbers of users who wish to share large amounts of data objects

Summary so far …

- **Benefits of Peer to Peer Systems:**
  - Ability to exploit un-used resources (storage, processing) in host computers
  - Scalability to support large numbers of clients and hosts whilst maintaining excellent balancing of loads on network links an hosts
  - Self-organizing properties of the middleware platforms results in support costs that are independent of the number of hosts and clients
- **Weaknesses of Peer to Peer Systems:**
  - Their use of the storage of **mutable data objects** is relatively costly compared to a trusted centralized service
  - Their **promise** of anonymity does not yet **guarantee** anonymity

Distributed Systems:
Case Study 1: Pastry

Pastry GUIDs

- In Pastry all nodes and objects are assigned a 128-bit GUID.
  - Node GUIDs are computed by applying a secure hash function (SHA-1) to an associated public key.
  - Object GUIDs are computed by applying a secure hash function to the object's name or some part of the object's stored state.
- The resulting GUID is randomly distributed in the range 0 to $2^{128}-1$.
- A key feature of Pastry is that:
  - In a network with N participating nodes, the routing algorithm will correctly route a message addressed to any GUID in O(log n) steps.
  - If the GUID identifies an active node then the message is delivered.
  - Otherwise the message is delivered to the active node **whose GUID is numerically closest to it**.
  - Active nodes take responsibility for processing requests addressed to all objects in their numerical neighbourhood.

Pastry Routing Overlay

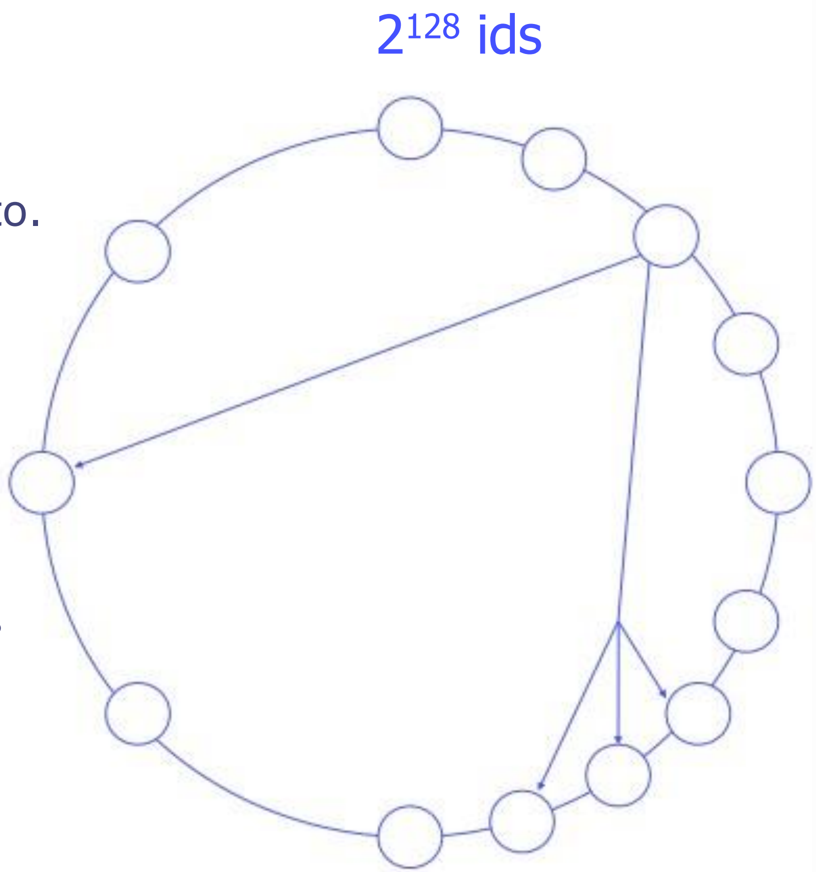
- The Routing Overlay steps involve the use of an underlying protocol (normally UDP) to transport a message to a Pastry node that is "closer" to its destination.
  - Closeness here refers to the distance in a logical (not physical) space.

- The real transport of a message across the Internet between two Pastry nodes may require a number of IP hops.
  - To minimise the risk of unnecessarily extended transport paths, Pastry uses a locality metric to select appropriate neighbours when setting up the routing tables used at each node.
  - This metric is based on network distance in the underlying network.

- The Routing Overlay is fully self-organising:
  - When a new node joins the overlay, they obtain the data needed to construct a routing table and other required state from existing members in O(log N) messages.
  - When a node departs or fails, the remaining nodes detect its absence and cooperatively reconfigure themselves.

A Basic Routing Algorithm

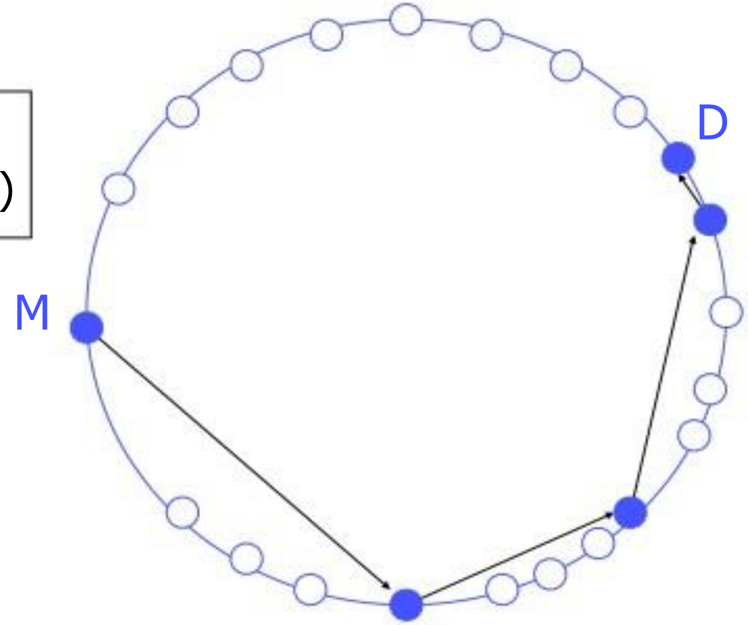
- Pastry employs a routing mechanism known as *prefix routing* to determine routes for delivery of messages based on the GUID that they are addressed to.
- GUID space is treated as **circular**.
  - GUID 0's lowest neighbour is $2^{128}-1$.
- Each active node stores a leaf set:
  - This is a vector L (of size *2ℓ*) containing GUIDs and IP addresses of the nodes whose GUIDs are numerically closest on either side of its own (ℓ above and ℓ below).
- Leaf sets are maintained by Pastry as nodes join and leave.
  - Even after node failure, the sets will be corrected in a short period of time.



The Pastry Routing Algorithm – Simple Version

- A Pastry system with correct leaf sets of size at least 2 can route messages to any GUID as follows:
  - Any node A that receives a message M with a destination address D routes the message by:
    - comparing D with its own GUID, and
    - with each of the GUIDs in its Leaf set
  - It forwards M onto the node that is numerically closest to D.

Here ℓ = 4
(normally ℓ =8)



Each step M is forwarded to, is a step closer to D than the current node

M is eventually delivered to D

Pastry Node

- Leaf Set (L)
  - A set of nodes that are numerically closest in the nodeId space to the present Node. Half larger and half smaller than the current node.
- Routing Table (R)
  - The routing table consists of a number of rows, where row *i* containing nodes sharing *i* initial digits of the nodeId with the local node
- Leaf set (L) and Routing table (R) used to ensure M is delivered in O(log N) steps

NodeId 10233102

Leaf set

SMALLER		LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table

-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Neighborhood set

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

Pastry Node (2)

- Neighborhood Set (M):

  - Contains nodeIds and IP addresses of the |M| nodes that are closest (according to the proximity metric) to the local node.

  - The neighbor set is used as a starting point for maintaining locality properties in the routing table.

NodeId 10233102

Leaf set

SMALLER		LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table

-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Neighborhood set

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

The Full Pastry Routing Algorithm

- Each Pastry node maintains a tree-structured routing table giving GUIDs and IP addresses for a set of nodes spread throughout the entire range of 2^{128} possible GUIDs.
  - There is increased density of coverage for GUIDs numerically closer to its own.

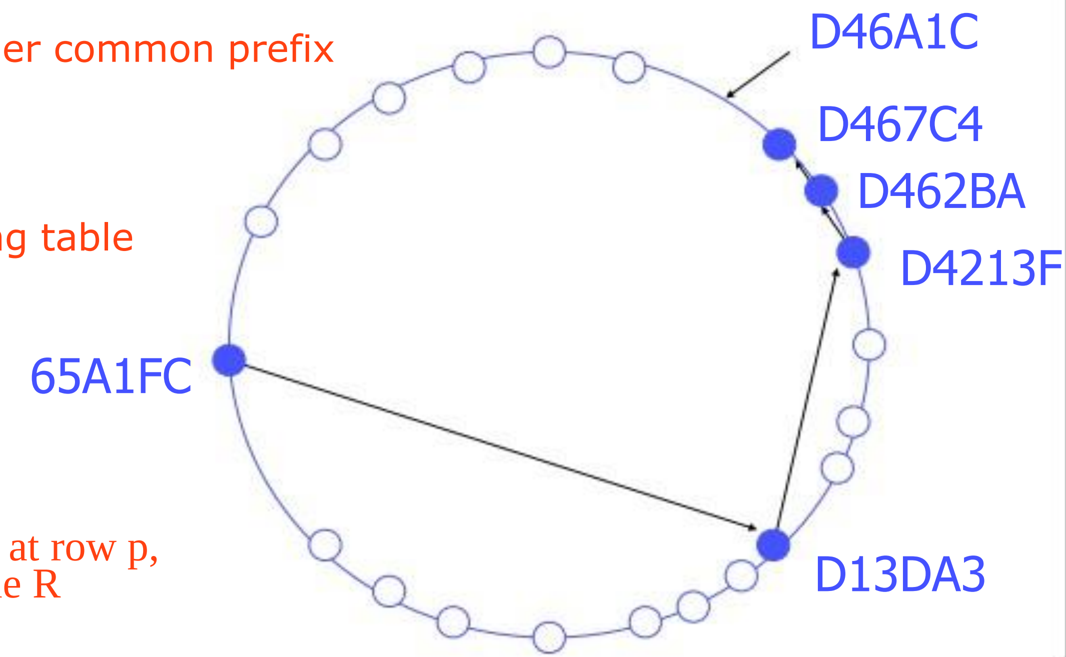
- Below is a partial routing table for node 65A1…

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

The Full Pastry Routing Algorithm

Algorithm that routes message M to Destination D

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



where R[p,i] is the element at row p, column i of the routing table R

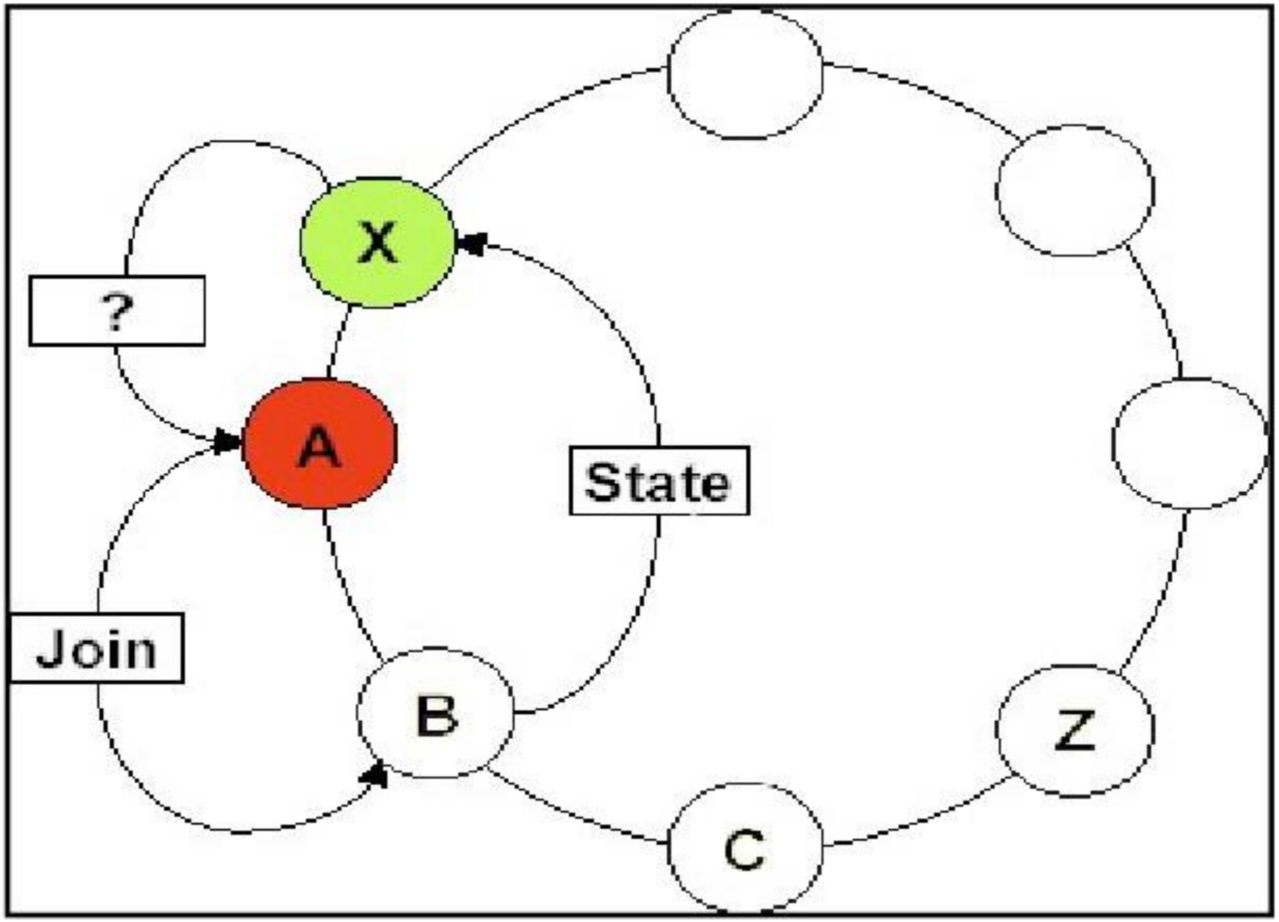
Pastry API

- **nodeId = pastryInit(Credentials, Application)**
  - Causes the local node to join an existing Pastry network (or start a new one), initialize all relevant state, and return the local node’s nodeId.
- **Route(msg, key)**
  - Causes Pastry to route the message to the node whose nodeId is numerically closest to the key.
- **Deliver(msg, key)**
  - called by Pastry when a message is received and the local node’s nodeId is numerically closest to the key.
- **Forward(msg, key, nextId)**
  - Called by Pastry just before the message is forwarded to the specified node. Allows the message to change the message content or target nodeId.
- **newLeafs(leafSet)**
  - Called by Pastry when there is a change in the local node’s leaf set.

Node Arrival

- For node with GUID X to join a Pastry system:
  - the new node computes it's own GUID X (by applying the SHA-1 hash function to it's public key)
  - Find a **nearby node A** (WRT network distance) that is part of pastry
    - determined using a *Nearest Neighbour algorithm*
  - Send a join message to Node A with GUID X
  - Node A will route message towards Node Z whose GUID is numerically closest to Node X
  - Nodes A, Z and all nodes in the path will send their state to X
  - Node X builds its Leaf Set and Routing Table and informs concerned nodes

Node Arrival



Node Arrival

- New node X initializes its Leaf Set (L), Routing Table (R) and Neighbour Set (N) as follows:
  - **Neighborhood Set:** is initialized with A’s (closest in proximity metric) neighborhood set
  - **Leaf Set:** Since Z is closest numerically to X:
    - X’s leaf set is initialized with Z’s leaf set.
  - **Routing Table:**
    - row 0 (R_0) of A’s routing table used to initialize X row 0
    - Row 1 (R_1) of node B’s routing table used to initialize X row 1
    - …
- Node X transmits a copy of its resulting state to all nodes in its neighborhood set (M), leaf set (L) and routing table (R).
- Each node updates own state based to include the new node.

Node Departure

- **Objective:** Maintain state integrity
- Node is considered to have failed when none of it's neighbours can communicate with it
- When this happens: need to update the leaf sets that contains the GUID of the failed node.
- If the failed node is it the leaf set L:
  - the failed node's neighbor (node that detects the failure) contacts a live node in **L** and asks for its leaf table **L'**, which it uses to repair the leaf set
- If the failed node was identified in the routing table:
  - routing of messages can proceed when entries are no longer live
  - Contact a live node in the same row for its entry of the same row
  - If no such node exists, contact a node in previous row for its entry

Analysis of Pastry

- Pastry is a generic peer-to-peer content location and routing system
  - Supports Replication
  - Fault-resistant
  - Scales well

- Used for a range of applications:
  - PAST: large scale p2p file sharing system
  - SCRIBE: Group communication system
  - Squirrel: decentralize p2p web cache
  - SplitStream: content streaming/distribution system

- Takes into account locality properties of nodes in the underlying transport network when routing messages

PASTRY and PAST

- PAST (persistent storage utility) is build on PASTRY.
  - It uses a *fileId*, computed as the hash of the file's name and owner, as a Pastry key for a file.
  - Replicas of the file are stored on the k Pastry nodes with nodeIds numerically closest to the fileId.
    - **k = replication factor chosen to ensure high probability of availability**
  - A file can be looked up by sending a message via Pastry, using the fileId as the key.
  - By definition, the lookup is guaranteed to reach a node that stores the file as long as one of the k nodes is live.
  - Moreover, it follows that the message is likely to first reach a node near the client, among the k nodes; that node delivers the file and consumes the message.

- Pastry's notification mechanisms allow PAST to maintain replicas of a file on the k nodes closest to the key, despite node failure and node arrivals, and using only local coordination among nodes with adjacent nodeIds.

Distributed Systems:
Case Study 2: BitTorrent

Introduction

- Peer to Peer File Sharing Protocol used for distributing large amounts of data
- One of the most common protocols used today - accounts for ~27% to 55% of all internet traffic (feb, 2009)
- Released in the summer of 2001 by Bram Cohen
- Basic Idea:
  - Chop file into many pieces
  - Replicate DIFFERENT pieces on different peers as soon as possible
  - As soon as a peer has a complete piece, it can trade it with other peers
  - Hopefully, we will be able to assemble the entire file at the end
- Consequence: can distribute large files without the heavy load on the source computer and network

Introduction

- BitTorrent efficient content distribution system using *file swarming (i.e. File Sharing)*
  - *- swarm = set of peers that are participating in distributing the same files*

- Usually **does not perform** all the functions of a typically P2P system such as **searching**

File Sharing – how it works

- To share a file or group of files, a peer first creates a **.torrent file**
- **.torrent file** = small file that contains:
  - meta data about file(s) to be shared
  - information about the tracker - computer that coordinates the file distribution
- Peers first obtain a **.torrent file**, and then connects to the specified **tracker** - which tells them from which peers to download the *pieces* of the file(s).

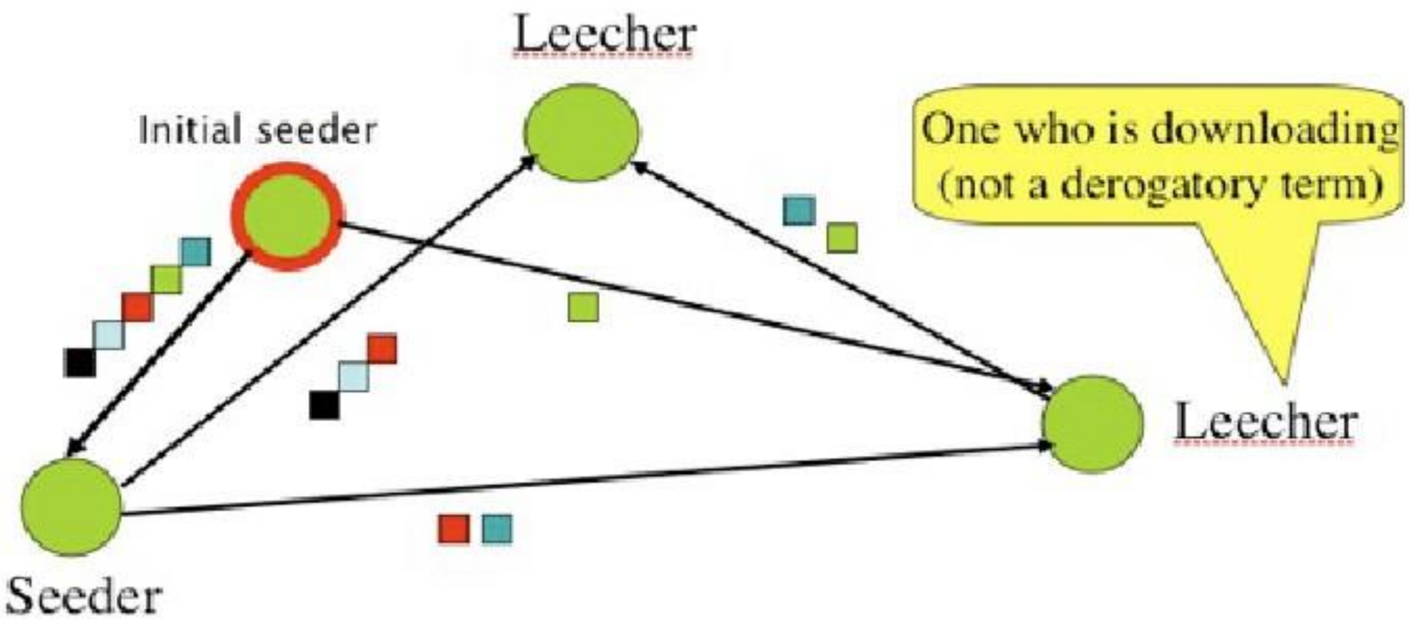
Basic Components

- **Seed**
  - Peer that has the entire file
- **Leech**
  - Peer that has an incomplete copy of the file
- **.torrent File**
  - the URL of the tracker
  - Pieces of the file: <hash1, hash2, .. , hash n>
  - piece length
  - name of the file
  - length of the file
- **A Tracker**
  - central server - keeps a list of all peers participating in the swarm
  - coordinates the file distribution
  - Allows peers to find each other
  - status information (i.e. completed or downloading)
  - Returns a random list of peers

Seeder v's Initial Seeder

Seeder = a peer that provides the complete file.

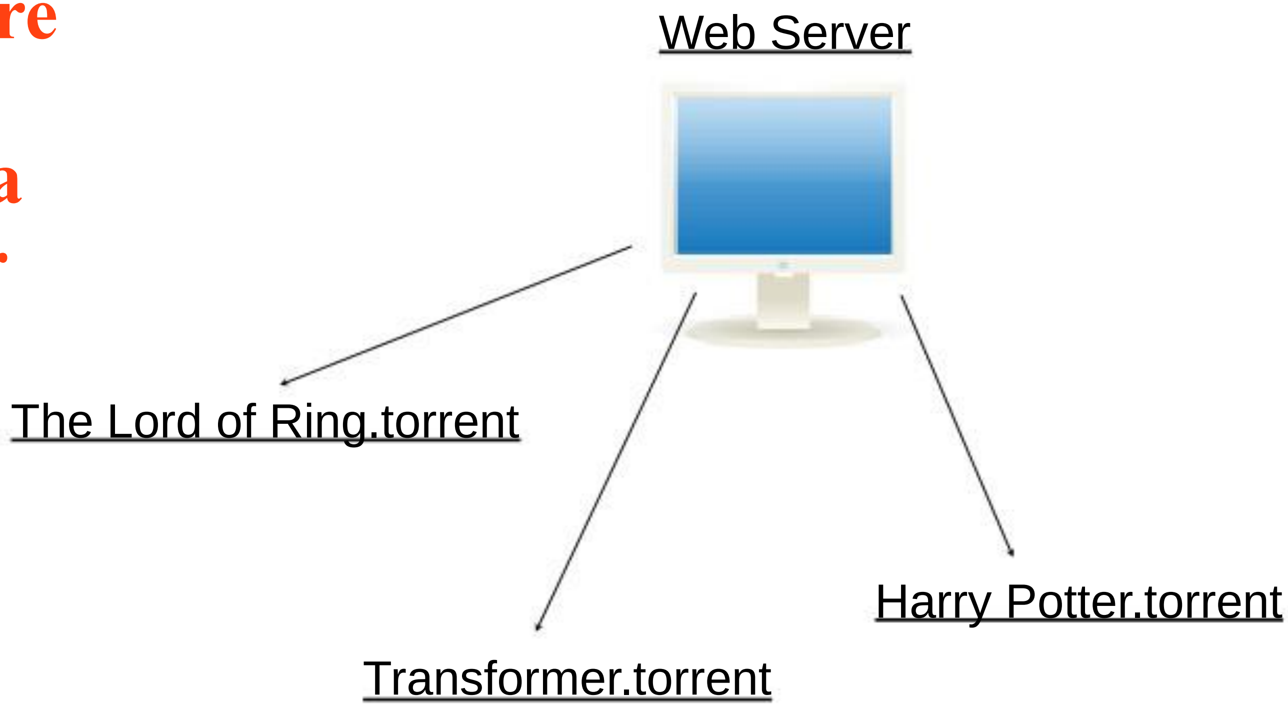
Initial seeder = a peer that provides the initial copy.



File Sharing

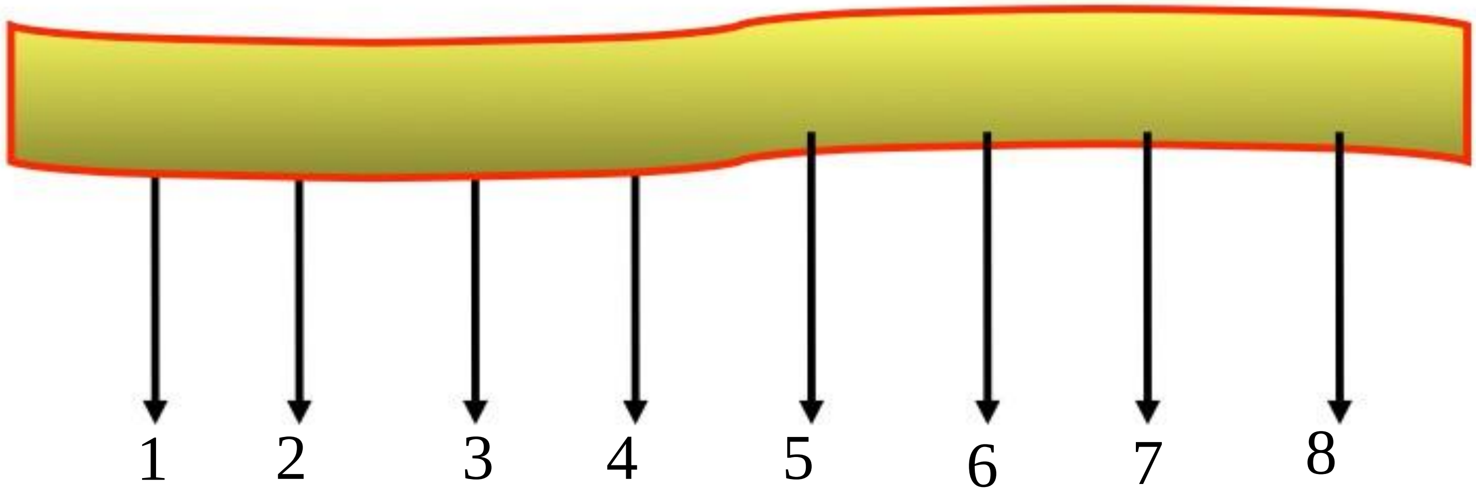
- Obtain a .torrent on a public domain site such as:
  - http://bt.LOR.net
  - http://bt.HarryPotter.com/

.torrents are typically hosted on a web server



File Sharing…

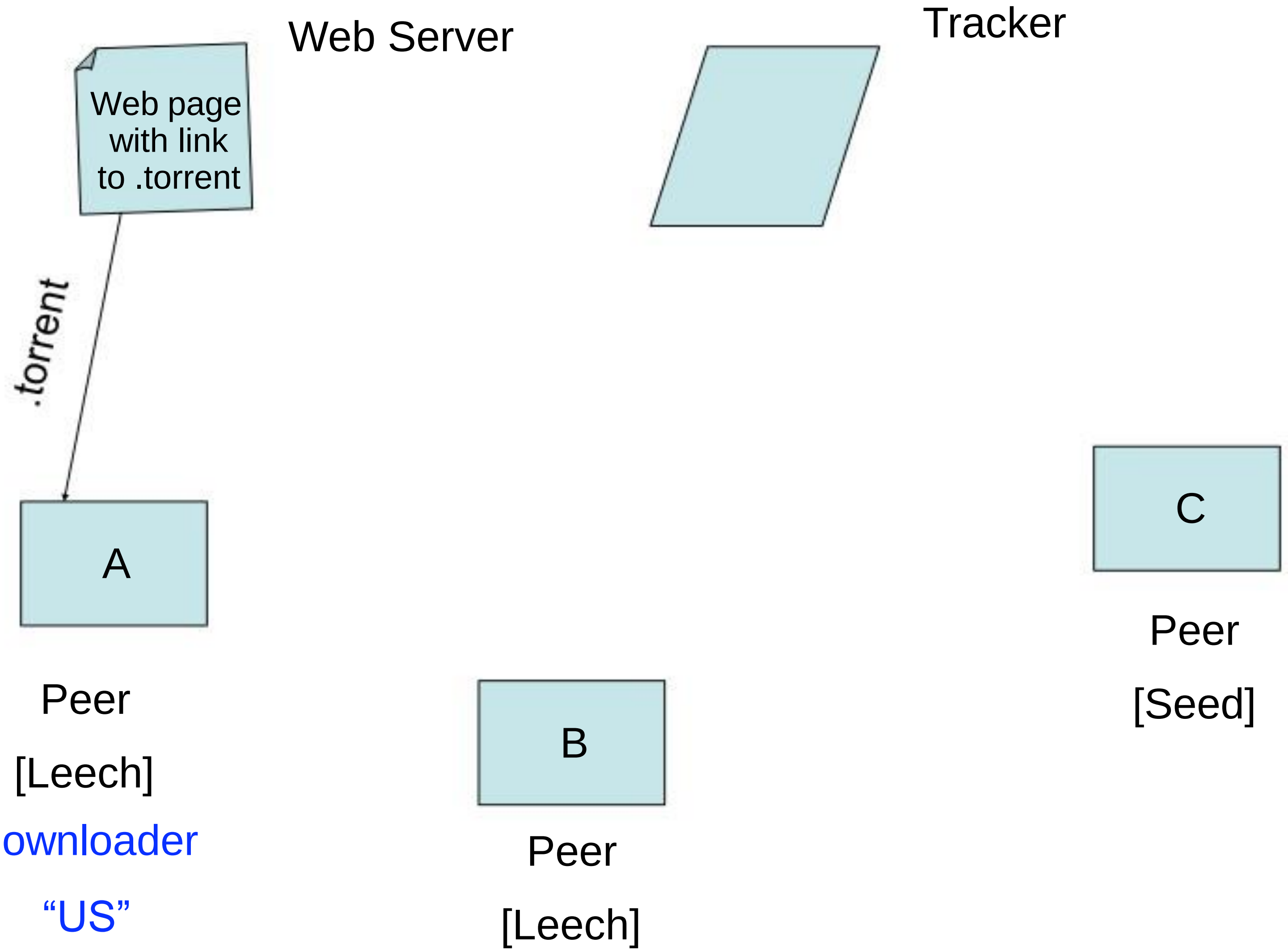
- Large files are broken up into pieces of sizes between 64KB and 1MB
  - normally 512KB segments are used



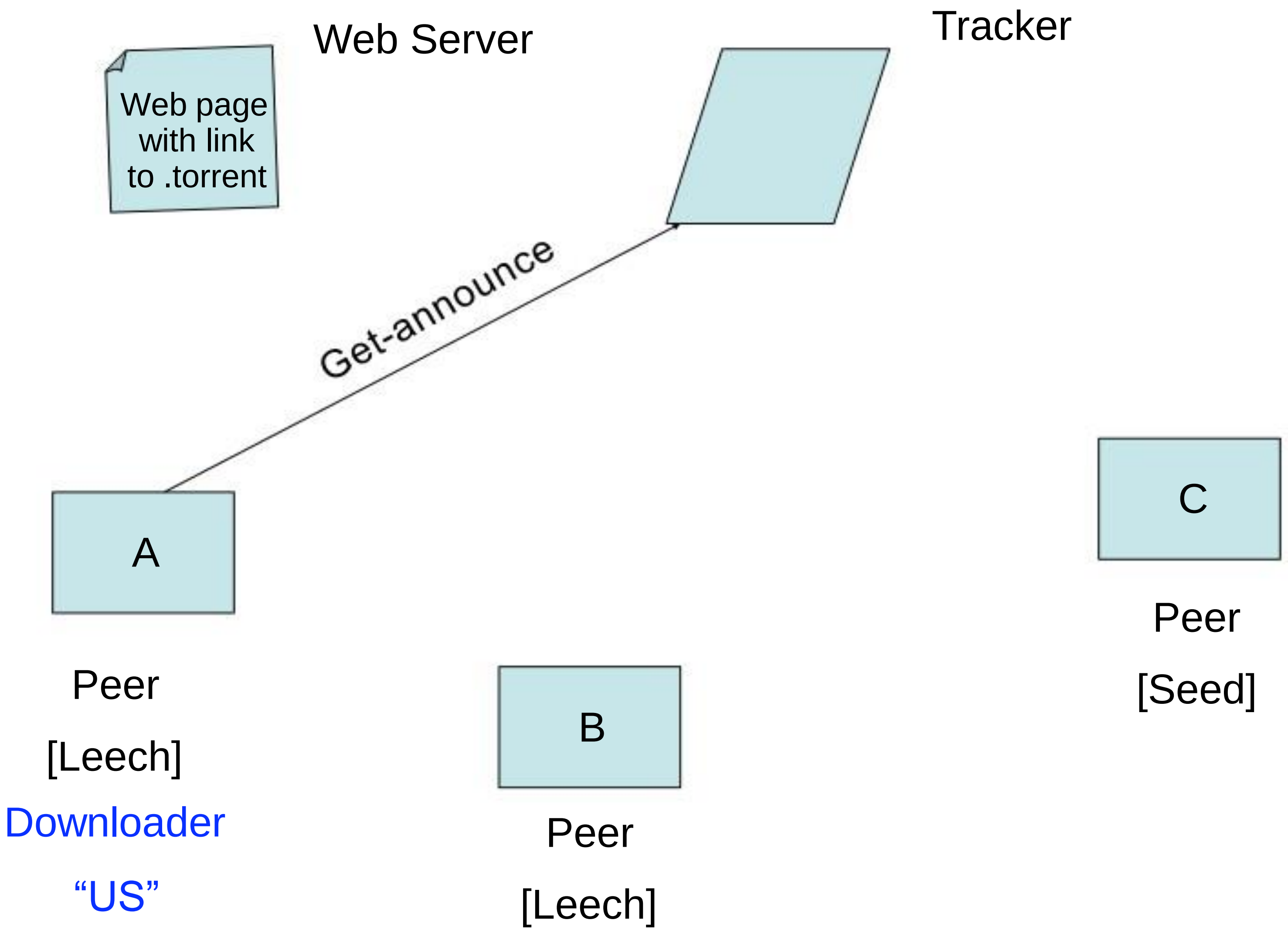
Basic Idea

- Initial seeder chops file into many pieces.
- Leecher first locates the **.torrent** file that directs it to a **tracker**, which tells which other peers are downloading that file. As a leecher downloads pieces of the file, replicas of the pieces are created. ***More downloads mean more replicas available***
- As soon as a leecher has a complete piece, it can potentially share it with other downloaders.
- Eventually each leecher becomes a seeder by obtaining all the pieces, and assembles the file. Verifies the checksum.

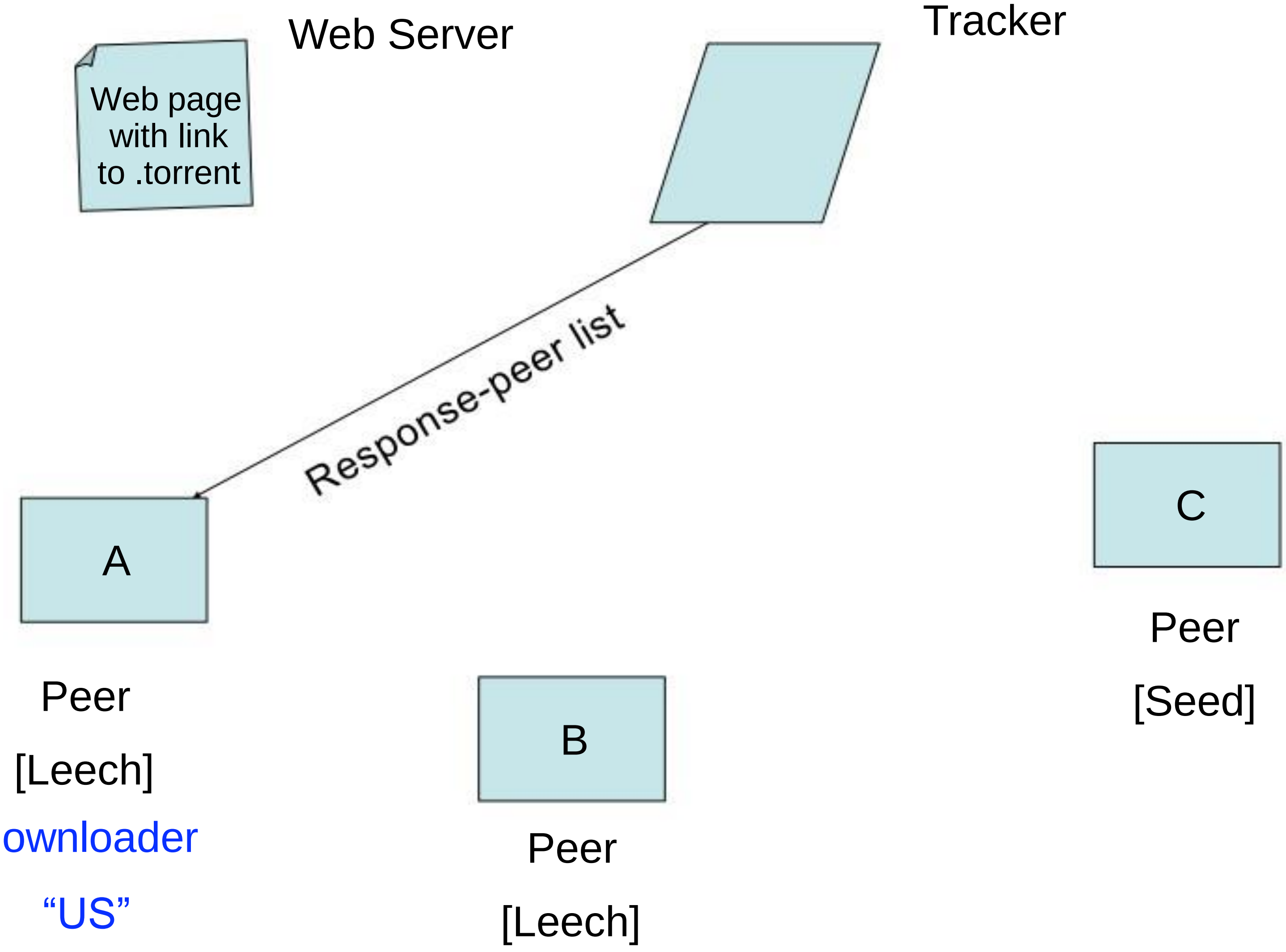
Overview – System Components



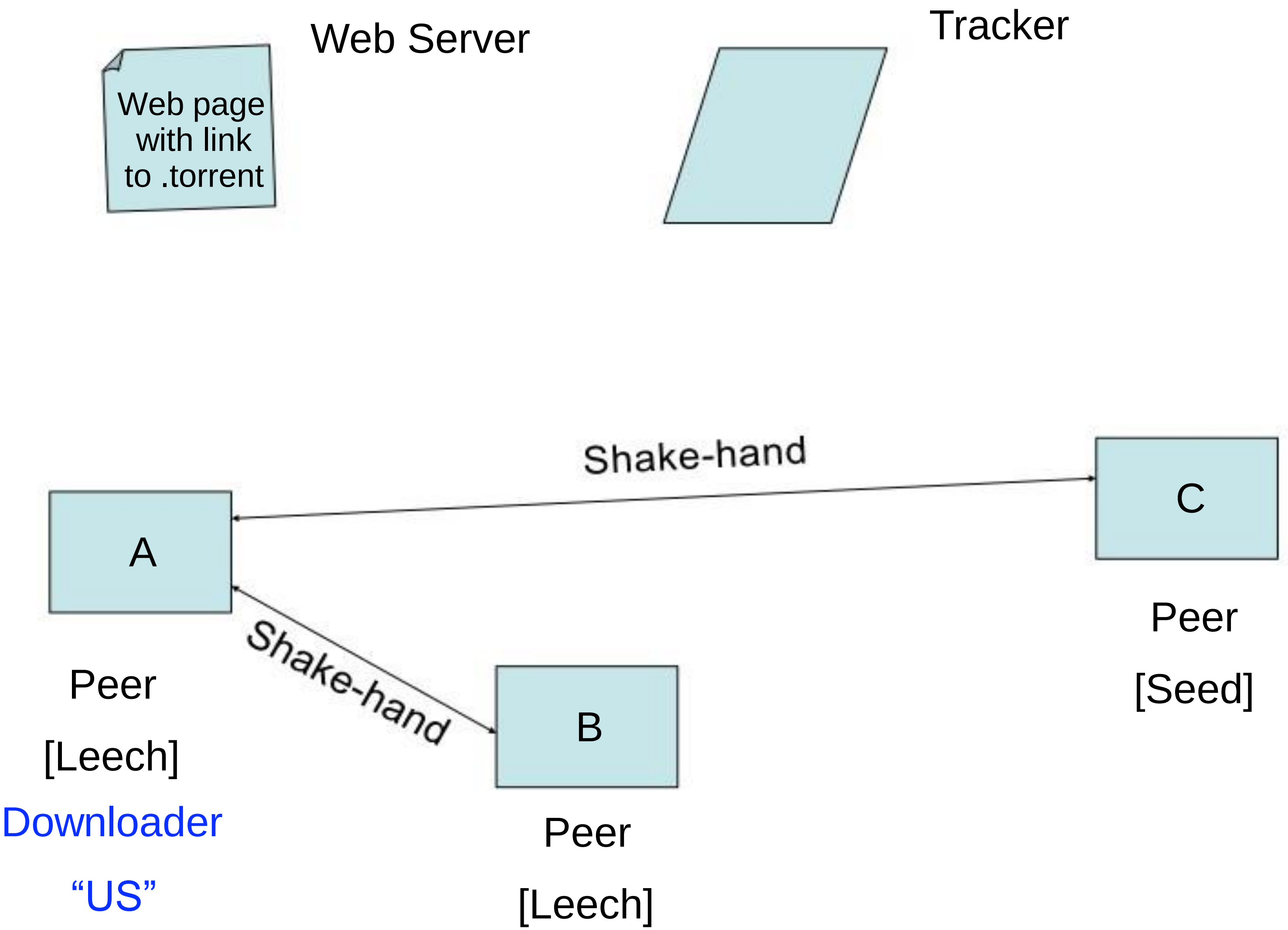
Overview – System Components(2)



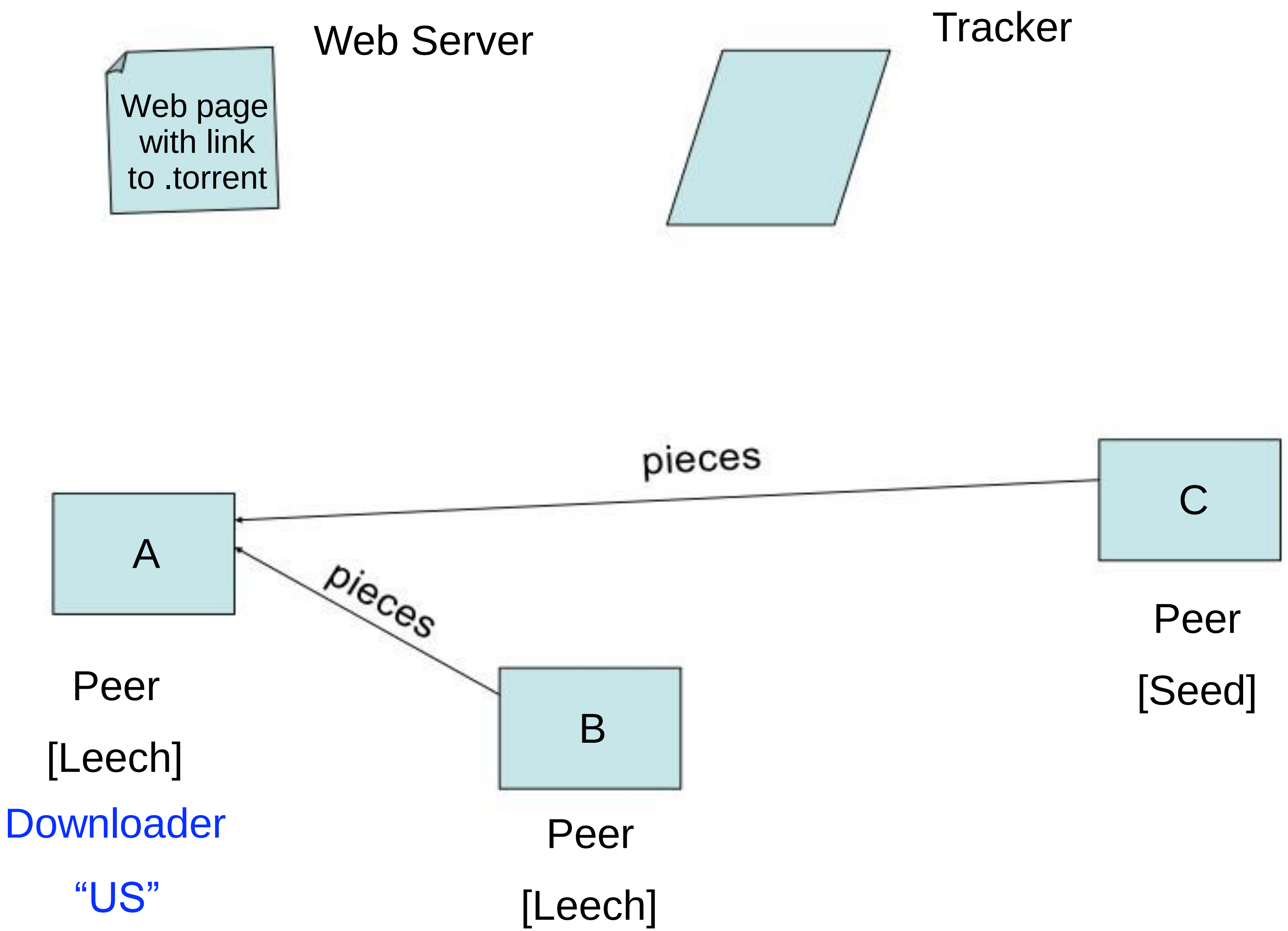
Overview – System Components(3)



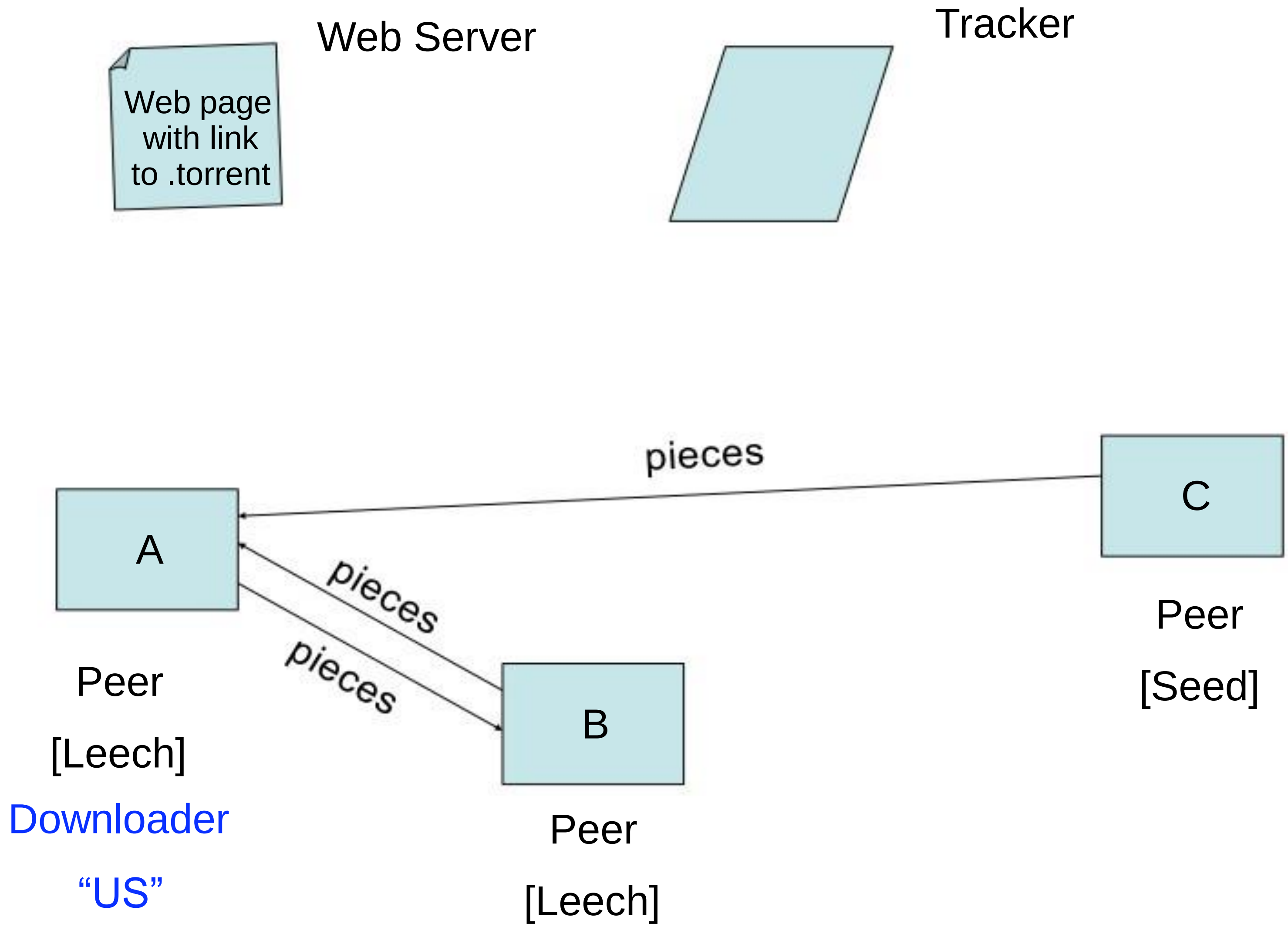
Overview – System Components (4)



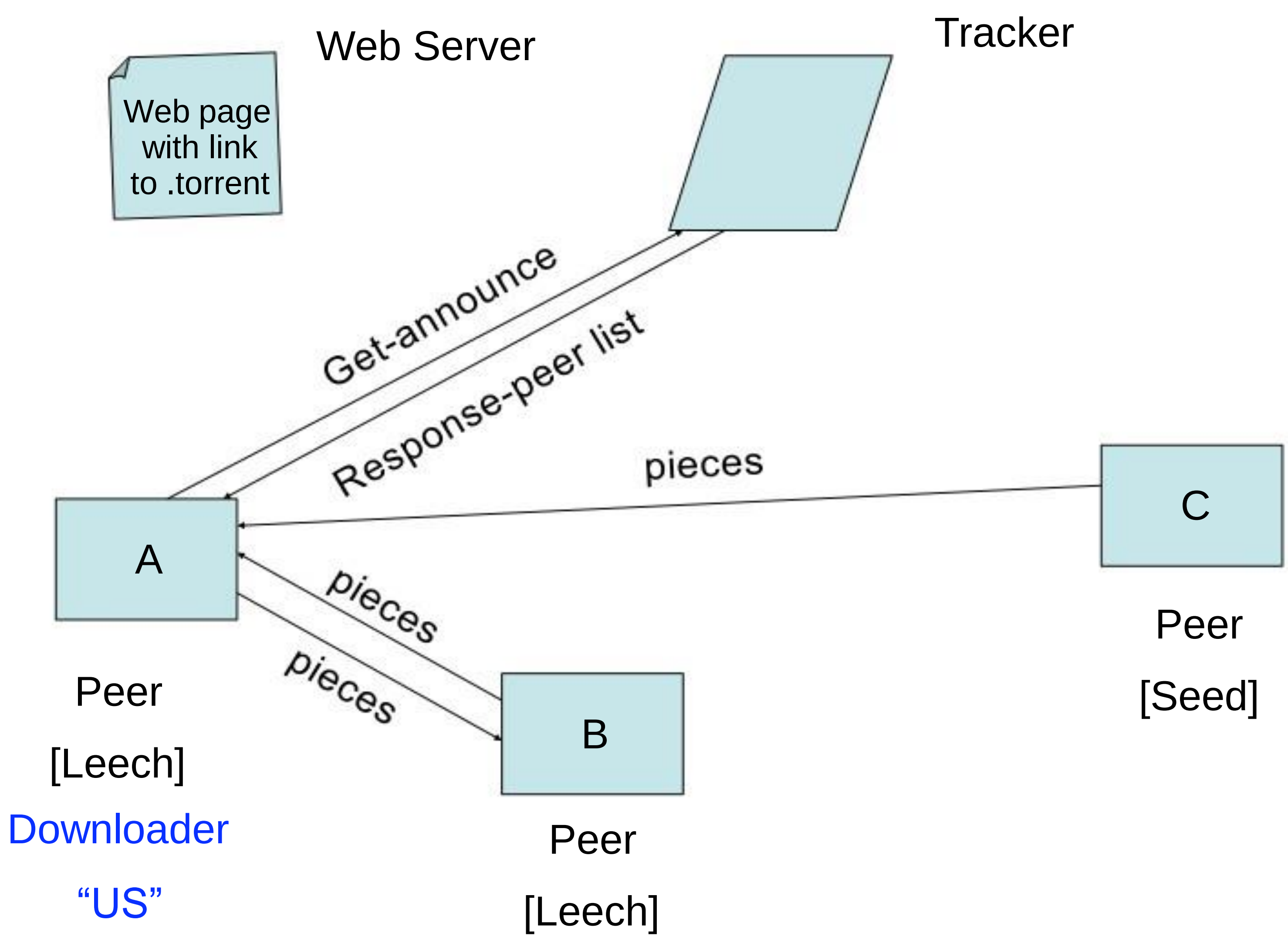
Overview – System Components (5)



Overview – System Components (6)



Overview – System Components (7)



Tracker Protocol

- communicates with clients via HTTP/HTTPS

- client GET request
  - info_hash: uniquely identifies the file
  - peer_id: chosen by and uniquely identifies the client
  - client IP and port
  - numwant: how many peers to return (defaults to 50)
  - status: bytes uploaded, downloaded, left

- tracker GET response
  - interval: how often to contact the tracker
  - list of peers, containing peer id, IP and port
  - status: complete, incomplete

Goals

- Efficiency

  - Fast downloads

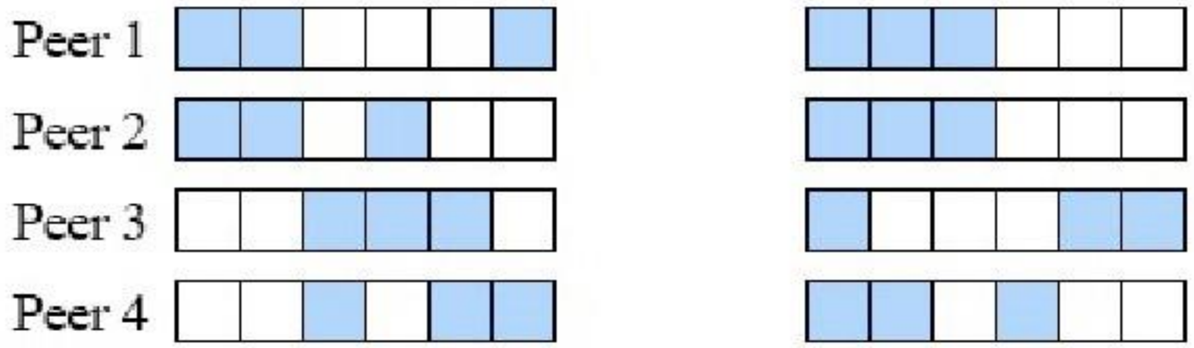
- Reliability

  - tolerant to dropping peers
  - ability to verify data integrity (SHA-1 Hashes)

Efficiency

- Ability to download from many peers yields to fast downloads
- Minimise **piece overlap** among peers to allow each peer to exchange with as many other peers as possible
- To minimise piece overlap:
  - download random pieces
  - prioritize the rarest pieces, aiming towards **uniform piece distribution** (all pieces are copied across peers the same number of times)

Piece Overlap



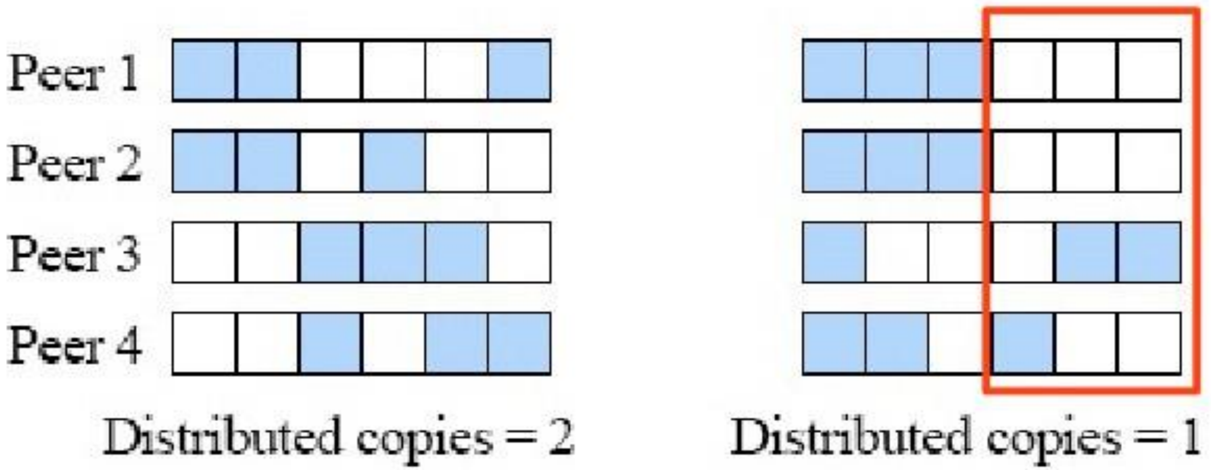
- Small overlap
  - Every peer can exchange pieces with all other peers
  - The bandwidth can be well utilised
- Big overlap
  - Only a few peers can exchange pieces
  - The bandwidth is under utilised

Reliability

- Be tolerant against dropping peers
  - each dropped peer means a decreased piece availability
- Maximise piece redundancy
  - maximise the number of distributed copies of each piece
  - ensures high availability

Distributed Copies

- The number of distributed copies is the number of copies of the rarest piece



Distributed Copies

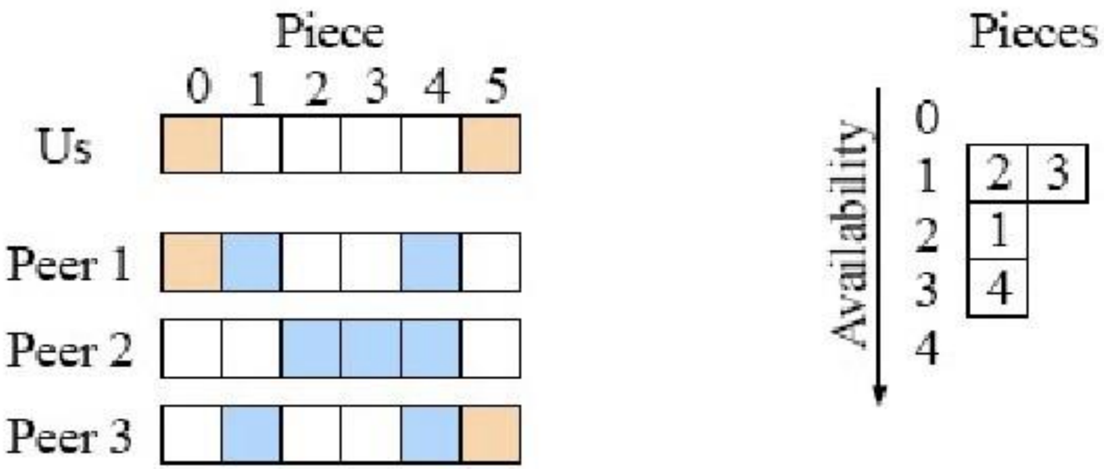
- To maximise the distributed copies - maximise the availability of the rarest pieces
- To increase the availability of a piece - download it
- To maximise the distributed copies:
  - download the rarest piece first

Rarest First

- The piece picking algorithm used in Bittorrent is called *rarest first*
- Picks a **random** piece from the set of **rarest** pieces
- No peer has global knowledge of piece availability, it is approximated by the availibility among neighbours

Rarest First

- Pick a **random** piece from the set of **rarest** pieces {2, 3}
- Ignore pieces that we already have



BitTorrent Summary

- BitTorrent works by using trackers to maintain lists of seeds and leechers associated with each shared file.
- In contrast with Napster, the BitTorrent server does not contain information about the names of the files that are being shared.
  - It only uses the info_hash identifier
- Further, BitTorrent does not download whole files - it downloads only parts of files.
  - This means that it is very difficult to identify who is downloading what files…
  - At the same time, it provides significant improvements in download times!!
- Aims: Efficiency and Reliability

Distributed Systems:
Case Study 3: Plaxton Mesh

For Reading -- will not be questioned in the exam