Distributed Systems: **Peer to Peer Networks**

Dr. Aidan Murphy

School of Computer
Science and Informatics
University College Dublin
Ireland

Distributed Systems: **Routing Overlays versus IP Routing**

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 middleware platforms are able to 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***

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

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 and hosts
  - Self-organizing properties of the middleware platforms results in support costs that are independent of the number of hosts and clients

P2P Networks
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

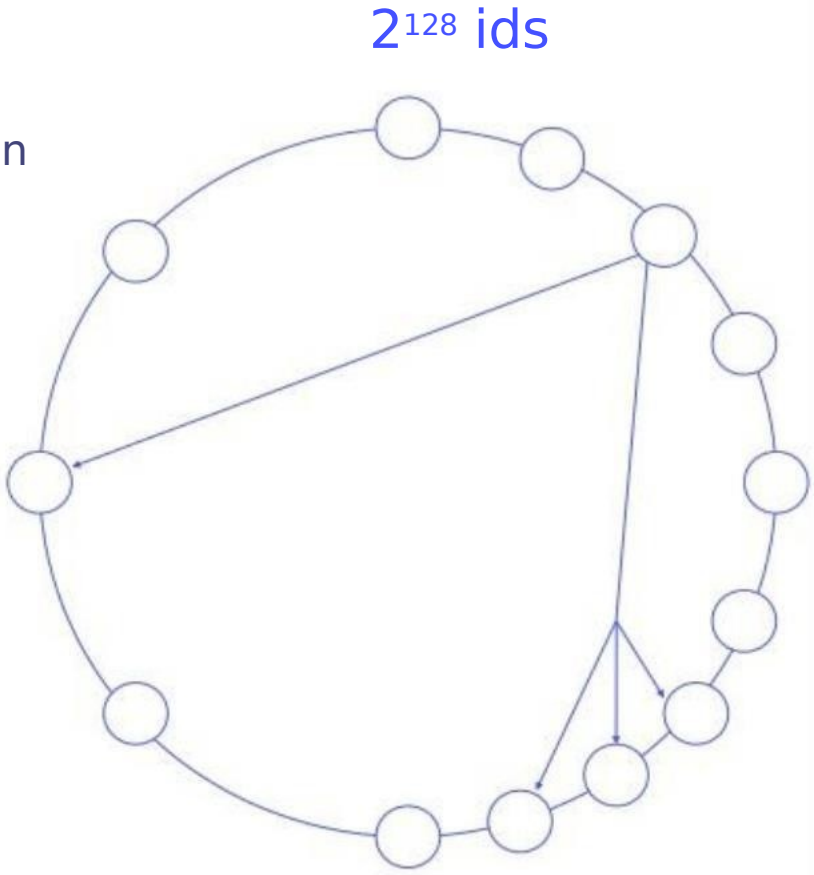
- The Routing Overlay steps involve the use of an underlying protocol (normally UDP) to transport a message to a Pastry node that is “closer” to its destination.
  - Closeness here refers to the distance in a logical (not physical) space.

- 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

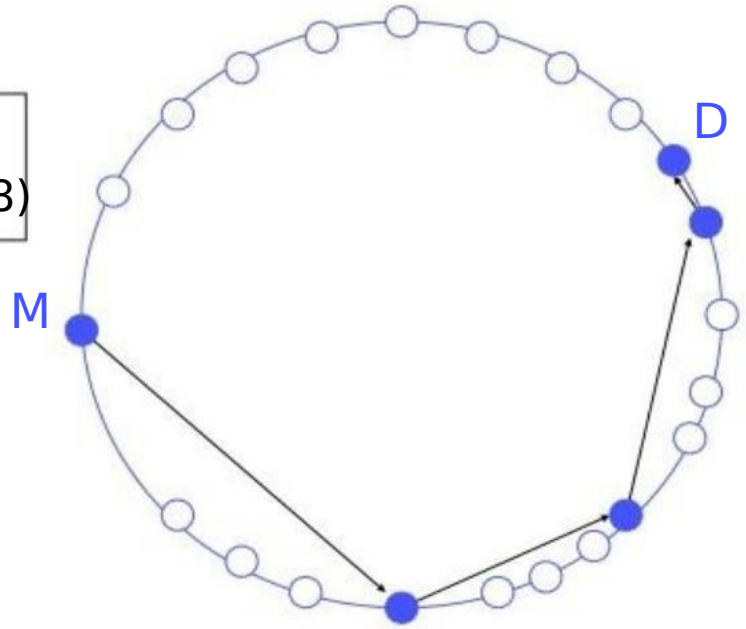
- Pastry employs a routing mechanism known as *prefix routing* to determine routes for delivery of messages based on the GUID that they are addressed to.
  - GUID space is treated as **circular**.
    - GUID 0's lowest neighbour is 2^{128}-1.
  - Each active node stores a leaf set:
    - This is a vector 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 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

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.

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

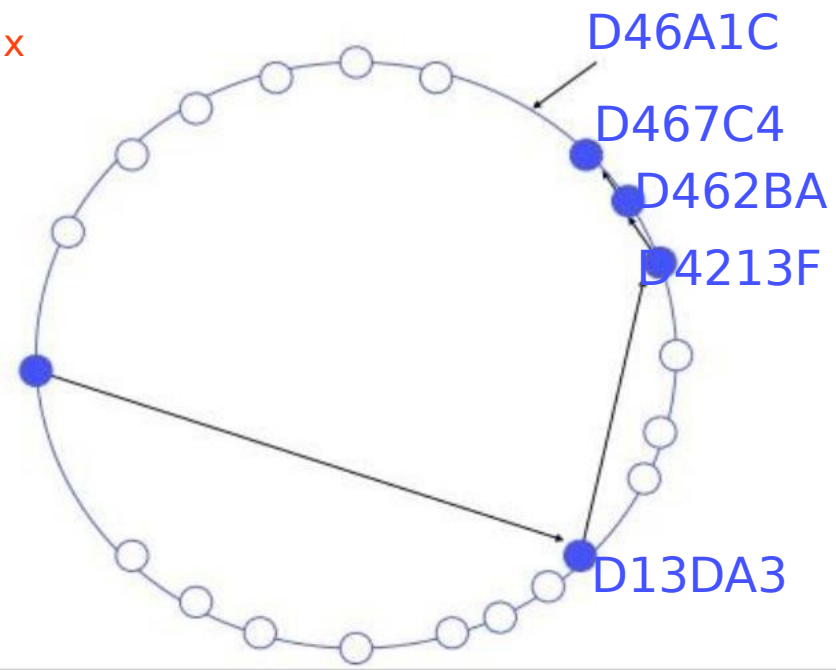
  { //there is no entry in the routing table

  Forward M to any node in L
  or R with a common prefix of length i, but a GUID that is numerically closer.

  }

  }

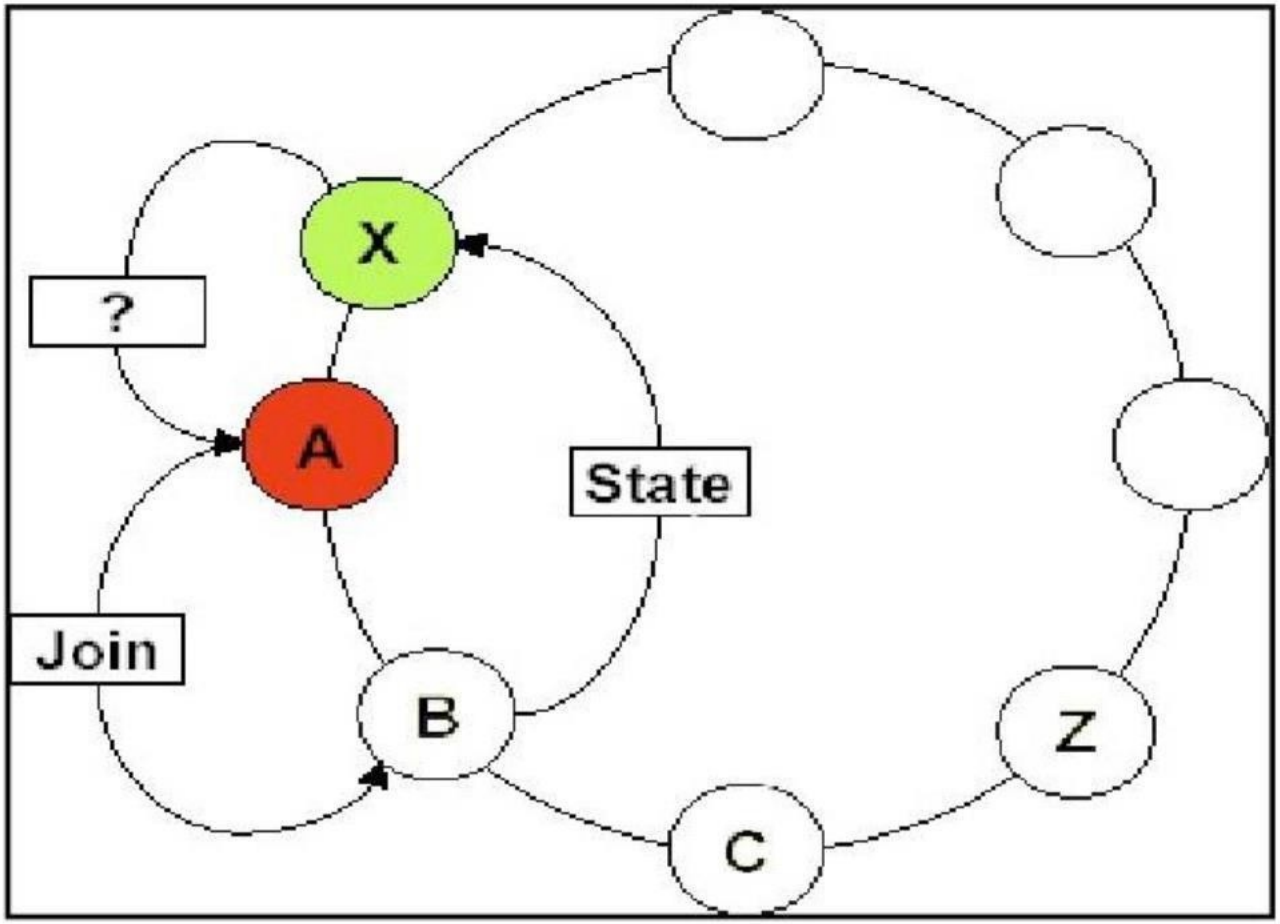
where R[p,i] is the element at row p, column i of the routing table R



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

P2P Networks: Case Study 2: BitTorrent

Introduction

- **Peer to Peer File Sharing Protocol** used for distributing large amounts of data

- 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 typical 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 connect to the specified **tracker** - which tells them from which peers to download the pieces of the file(s).

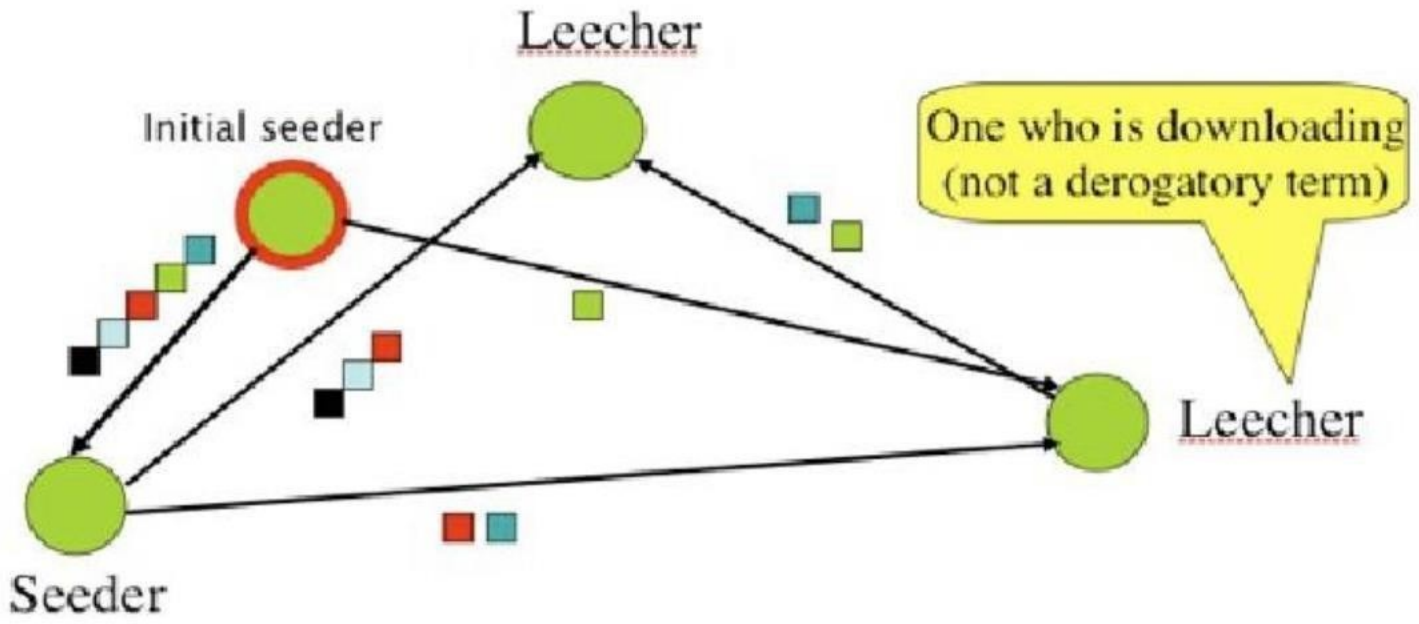
Basic Components

- **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.

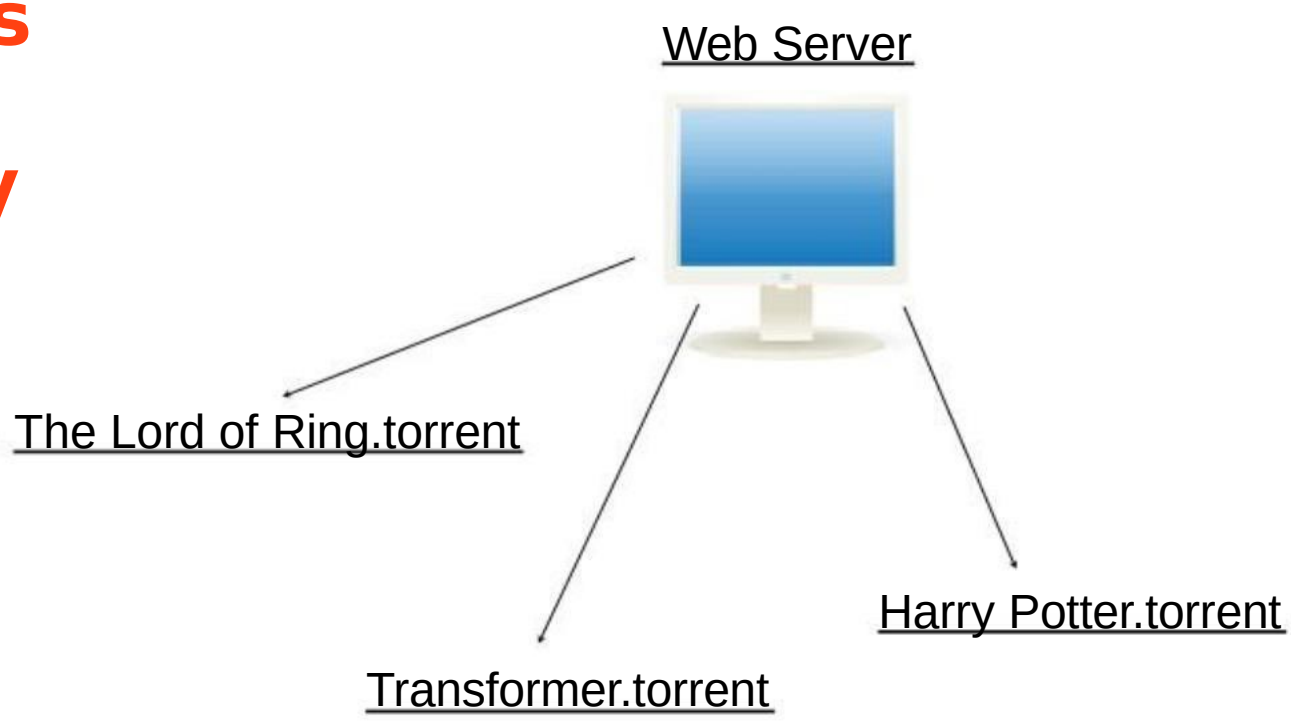
Initial seeder = a peer that provides the initial copy.



File Sharing

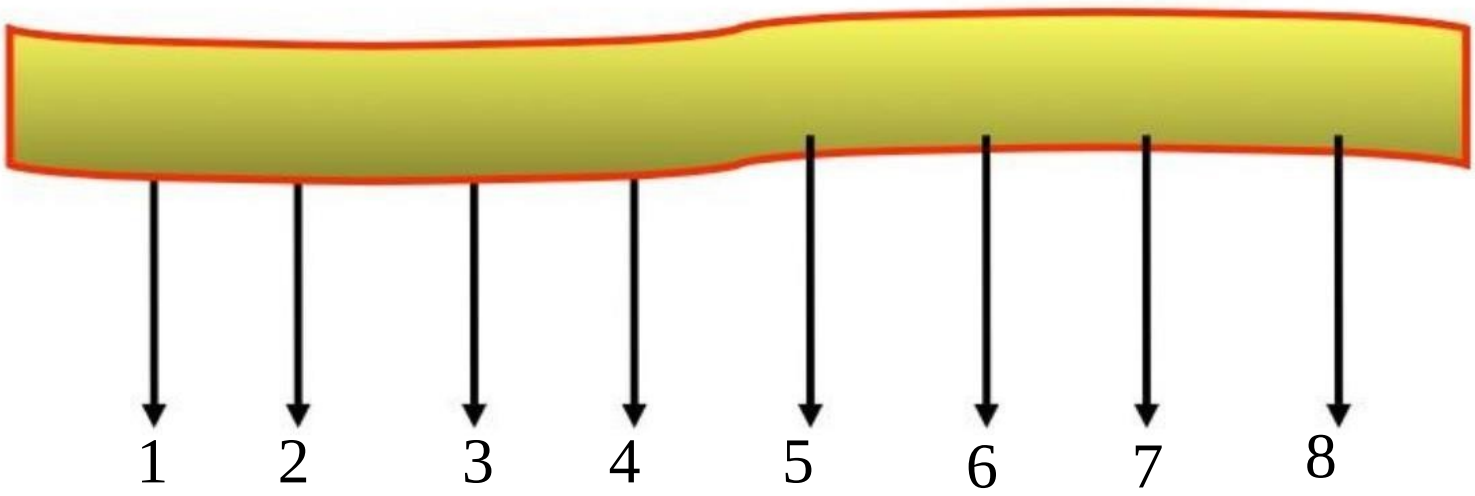
- Obtain a .torrent on a public domain site such as:
  - http://bt.LOR.net
  - http://bt.HarryPotter.com/

.torrents are typically hosted on a web server



File Sharing…

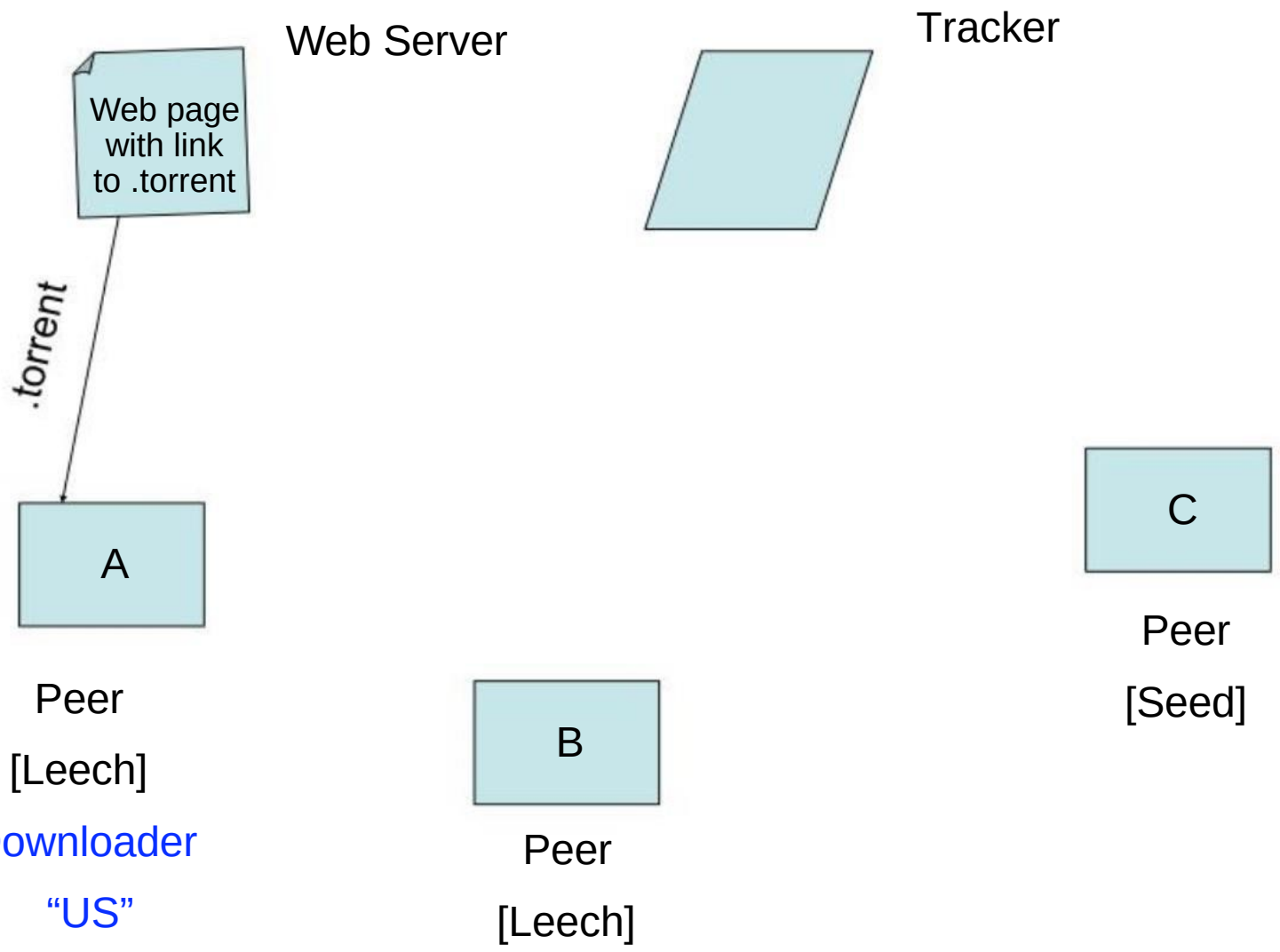
- Large files are broken up into pieces of sizes between 64KB and 1MB
  - normally 512KB segments are used



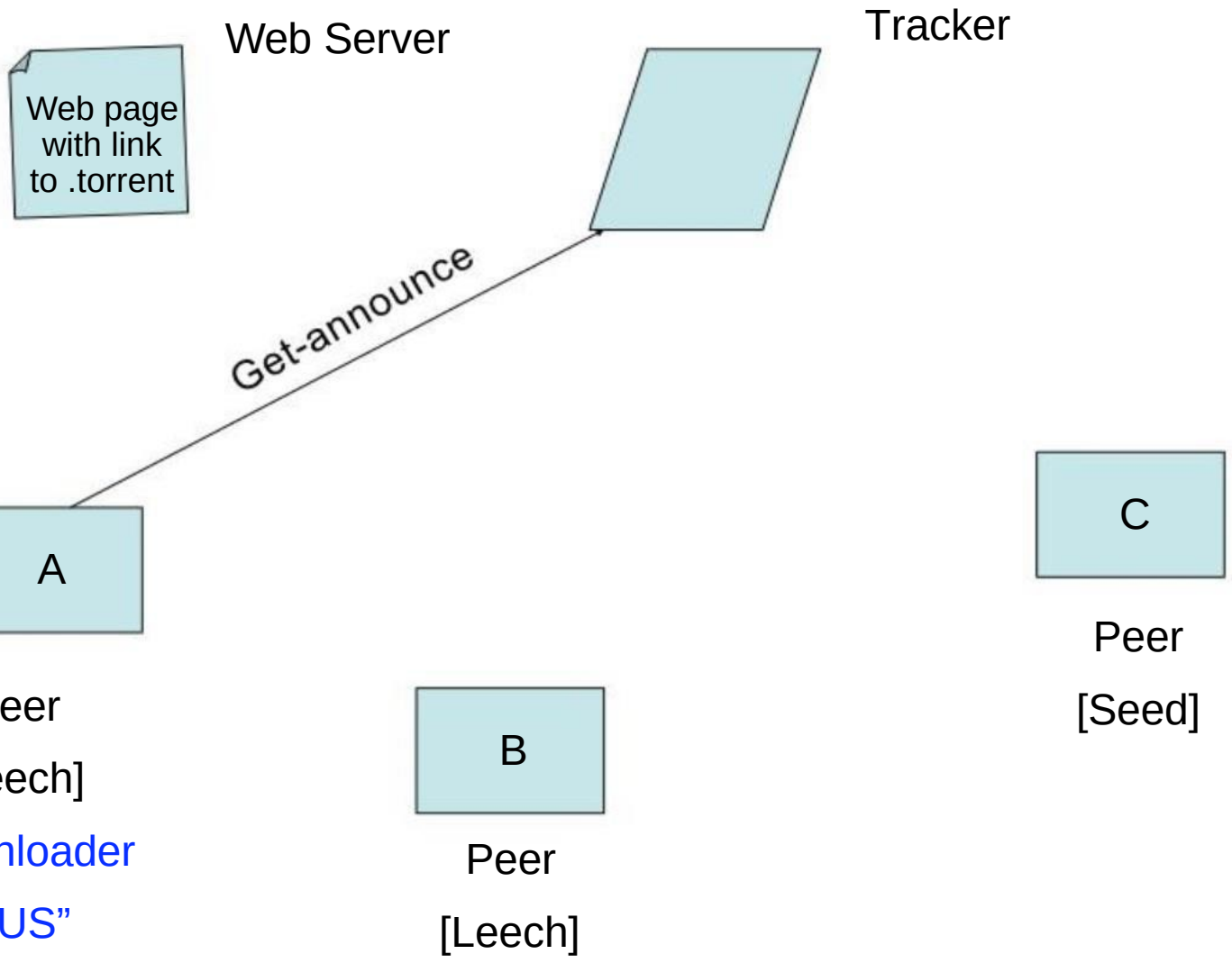
Basic Idea

- 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.

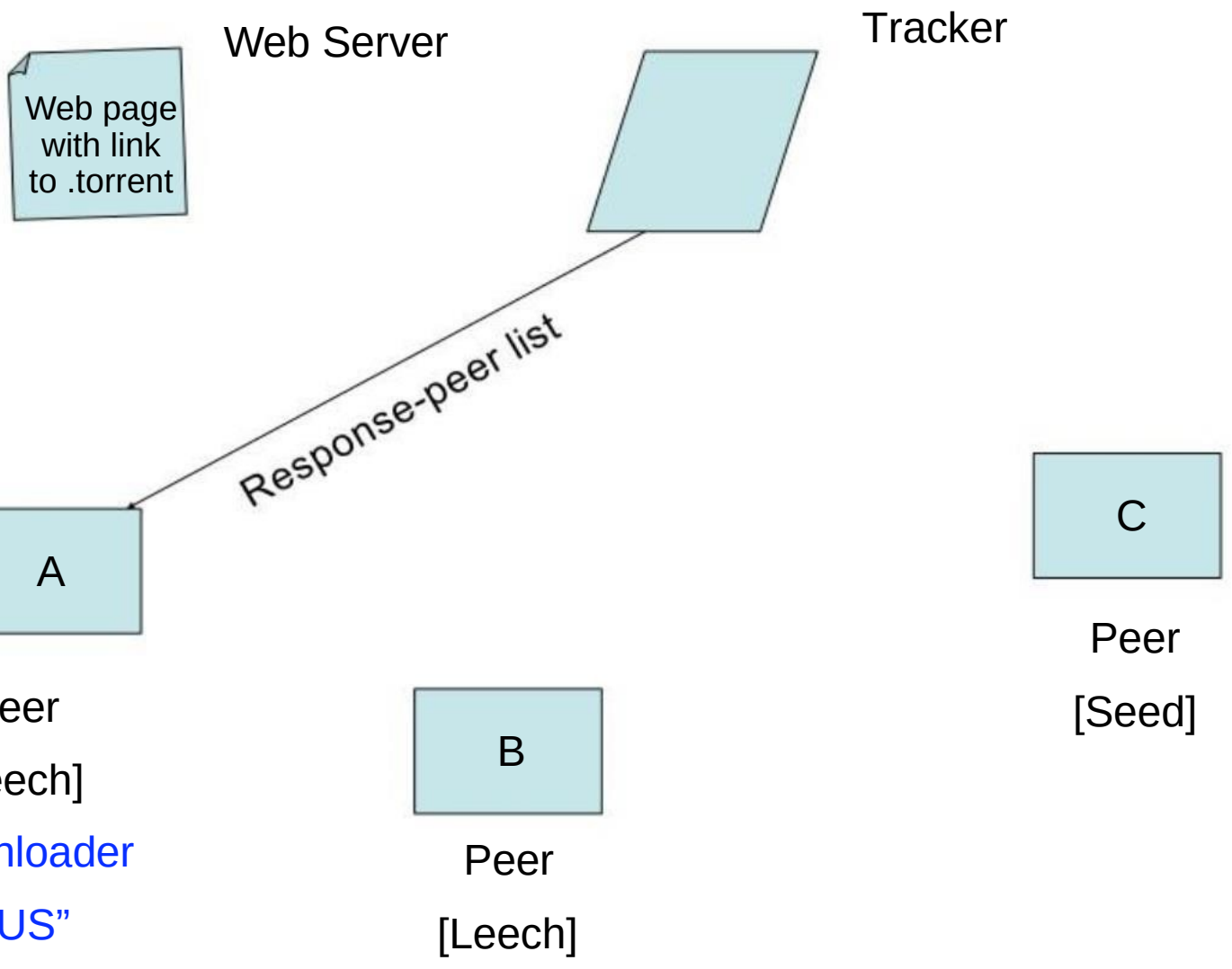
Overview – System Components



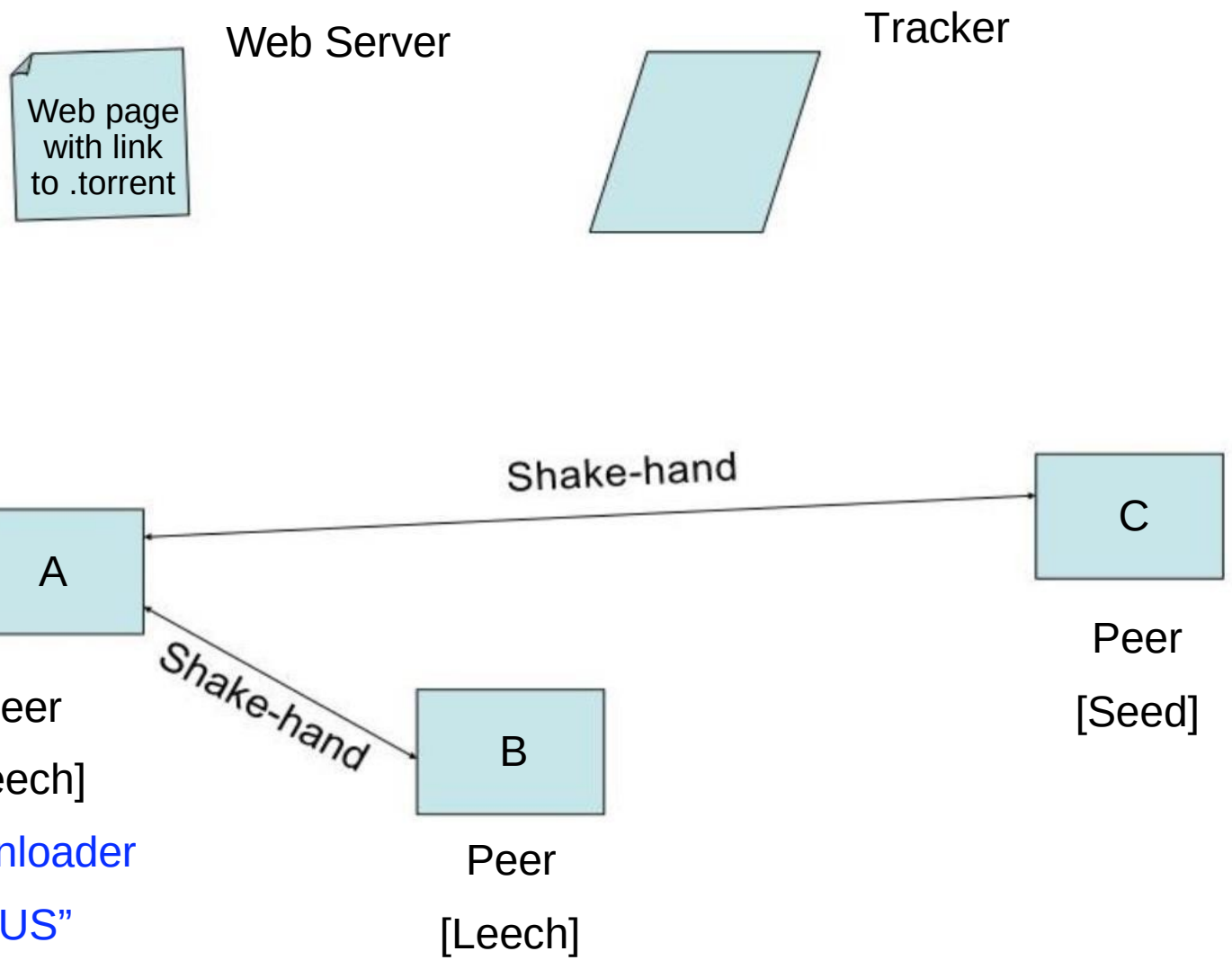
Overview – System Components(2)



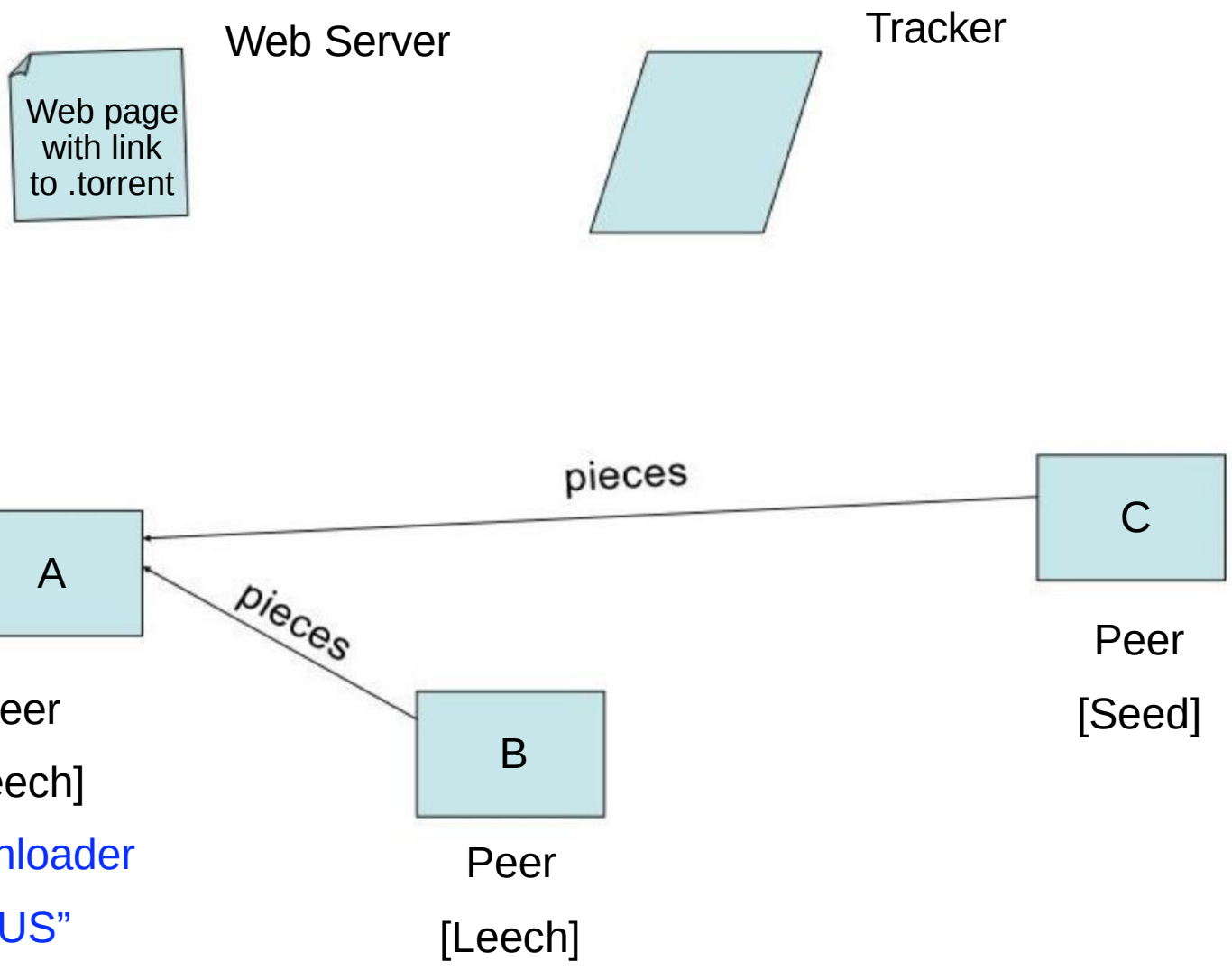
Overview – System Components(3)



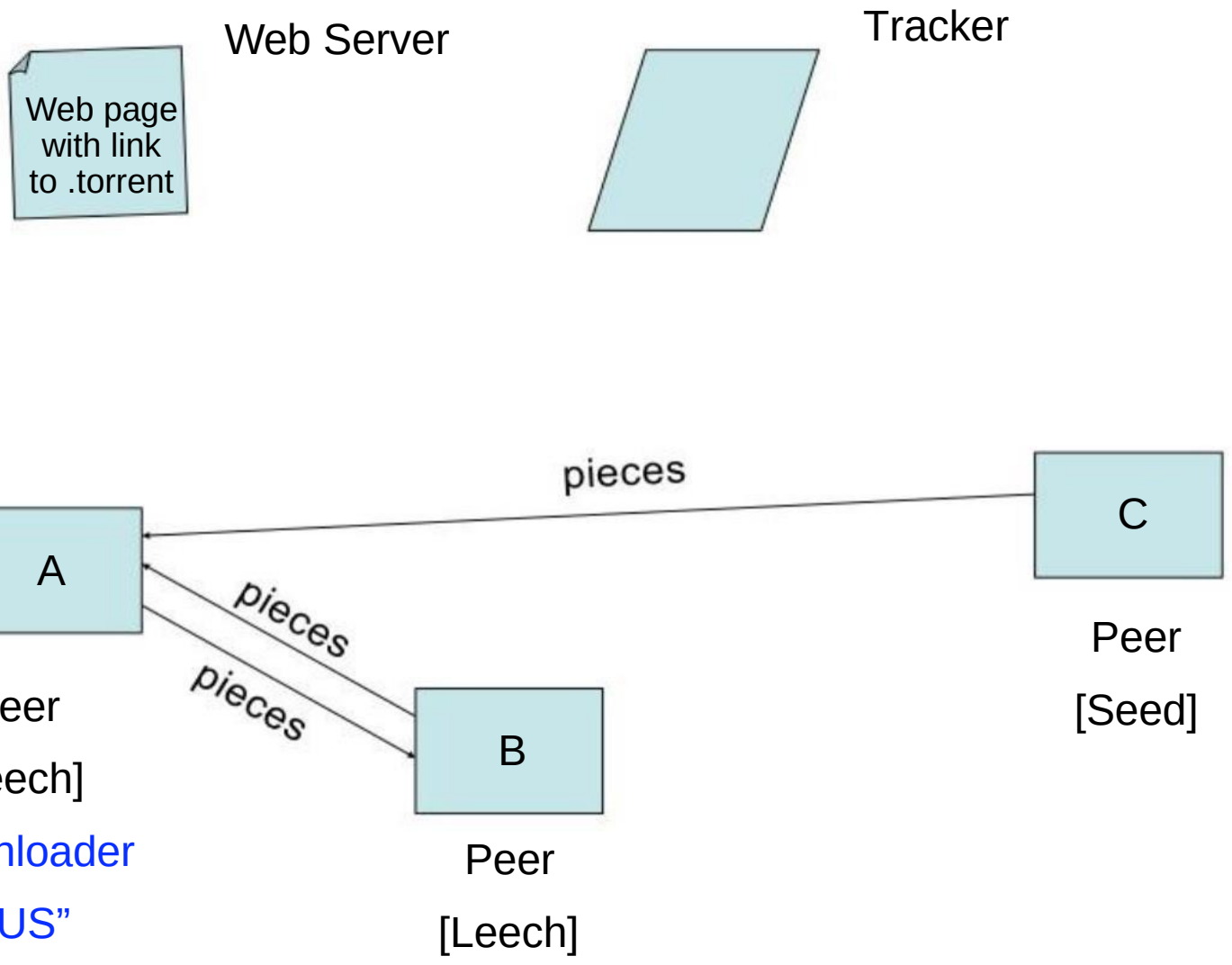
Overview – System Components (4)



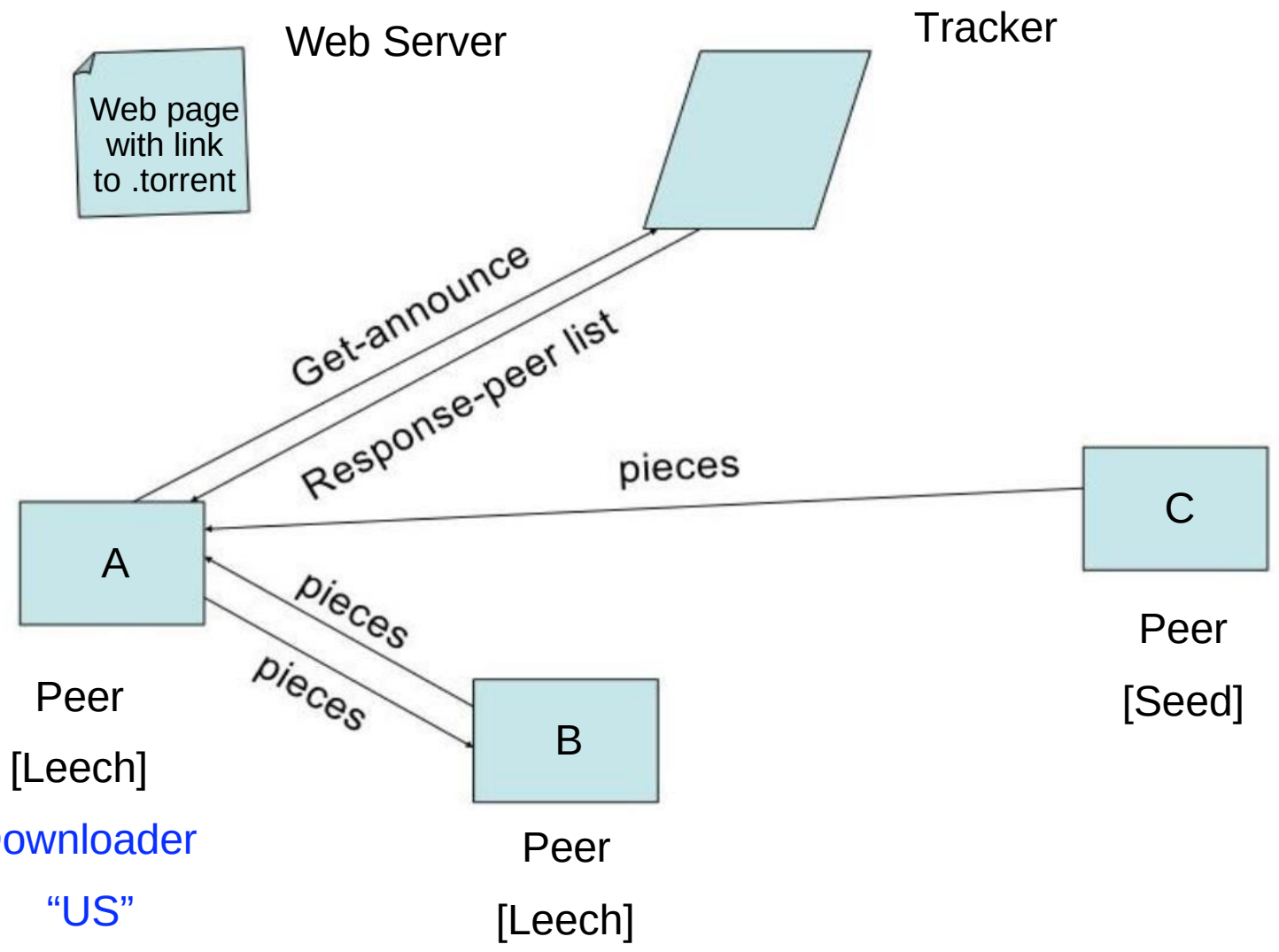
Overview – System Components (5)



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

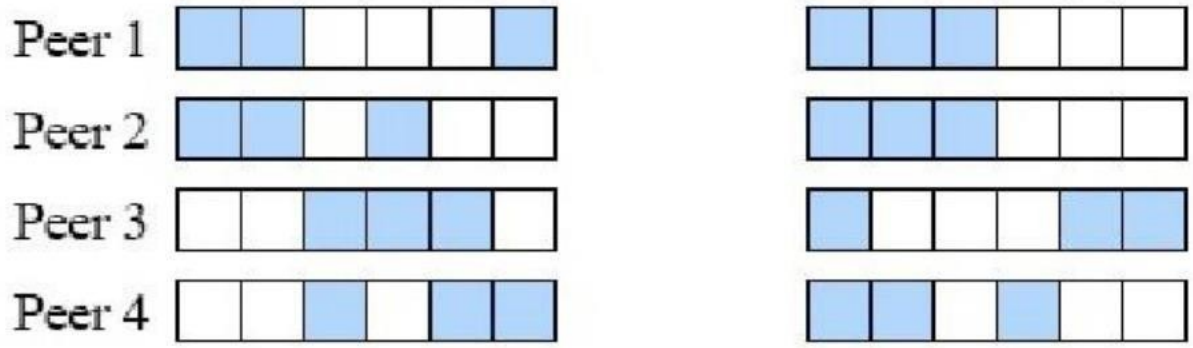
Goals

- 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



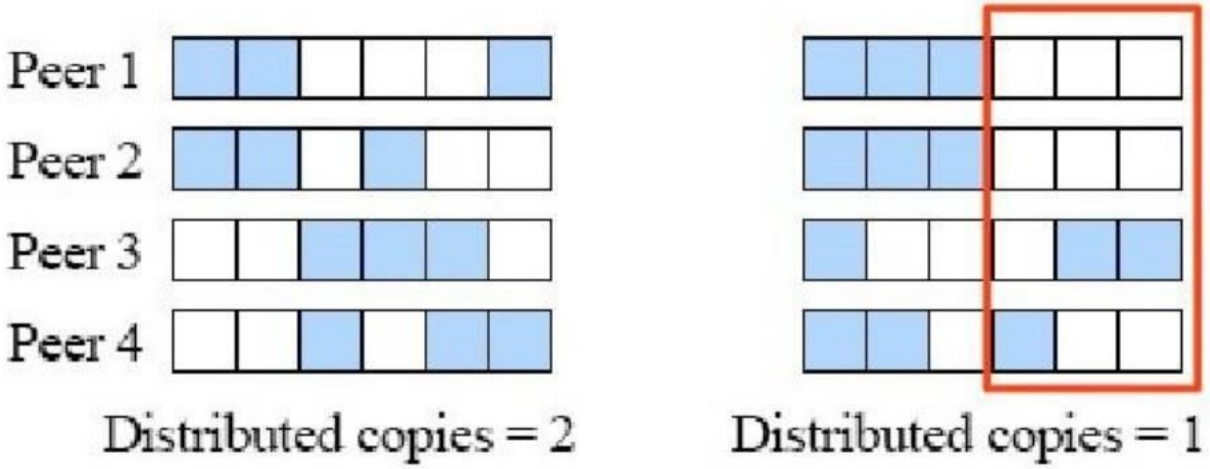
- Small overlap
  - Every peer can exchange pieces with all other peers
  - The bandwidth can be well utilised
- Big overlap
  - Only a few peers can exchange pieces
  - The bandwidth is under utilised

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

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

Thank you

For general enquries, contact:

aidan.murphy@ucd.ie