Peer-to-Peer Applications

Reading: 9.4

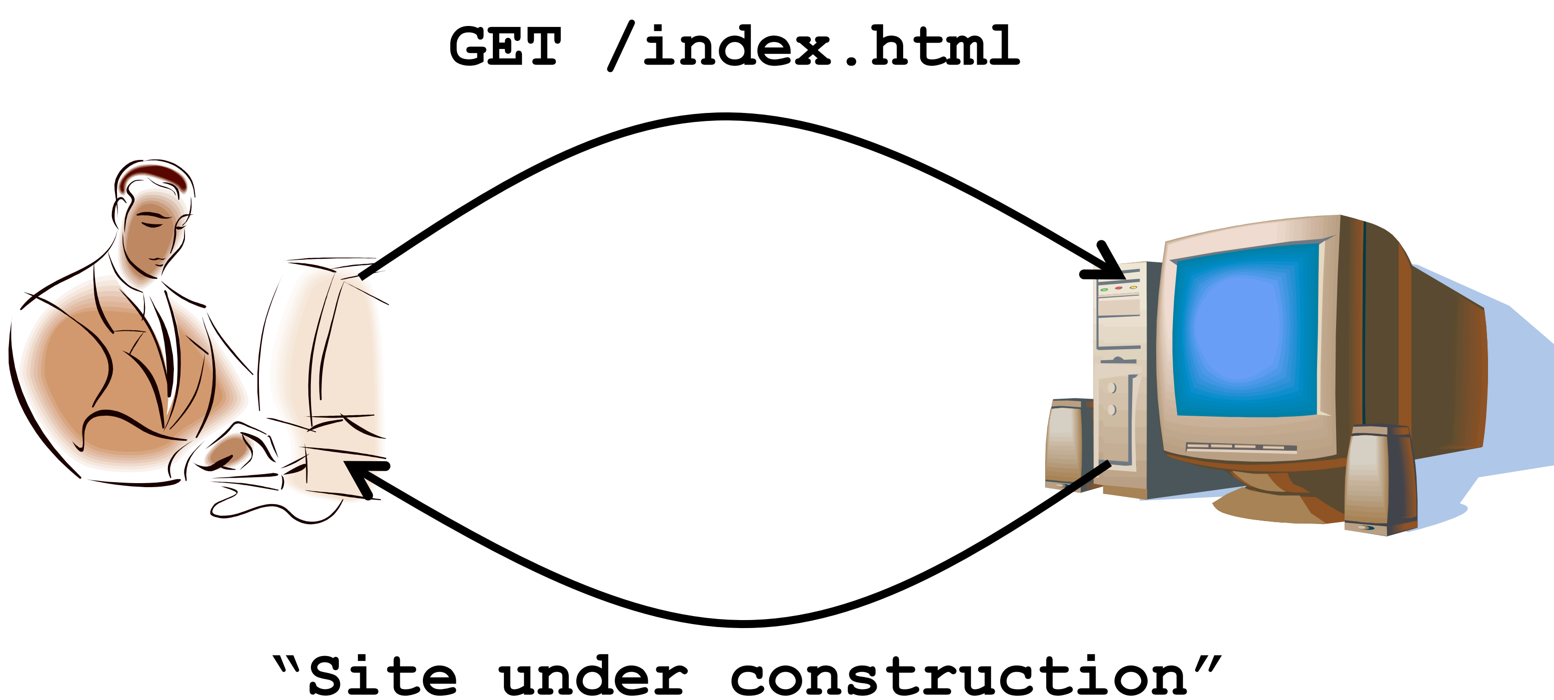
Acknowledgments: Lecture slides are from Computer networks course thought by Jennifer Rexford at Princeton University. When slides are obtained from other sources, a reference will be noted on the bottom of that slide and full reference details on the last slide.

Goals of Today's Lecture

- Scalability in distributing a large file
  - Single server and *N* clients
  - Peer-to-peer system with *N* peers
- Searching for the right peer
  - Central directory (Napster)
  - Query flooding (Gnutella)
  - Hierarchical overlay (Kazaa)
- BitTorrent
  - Transferring large files
  - Preventing free-riding

Clients and Servers

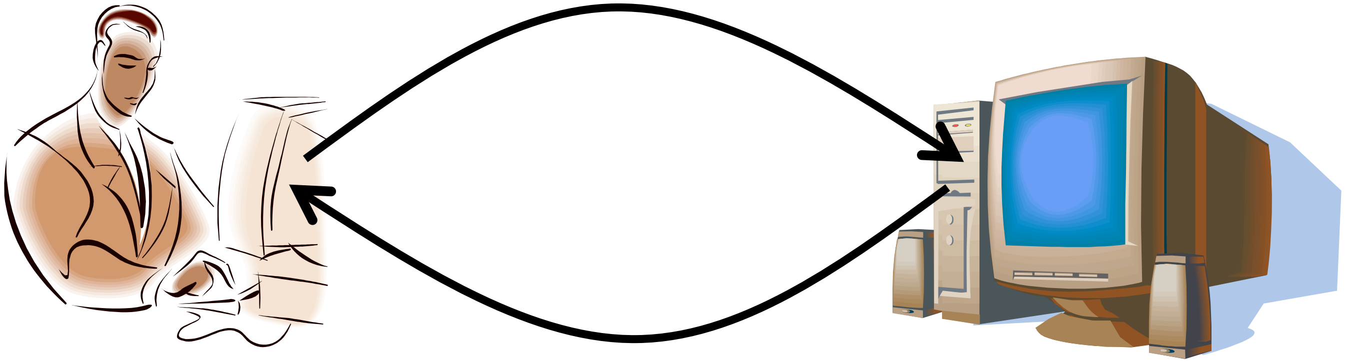
- Client program
  - Running on end host
  - Requests service
  - E.g., Web browser
- Server program
  - Running on end host
  - Provides service
  - E.g., Web server



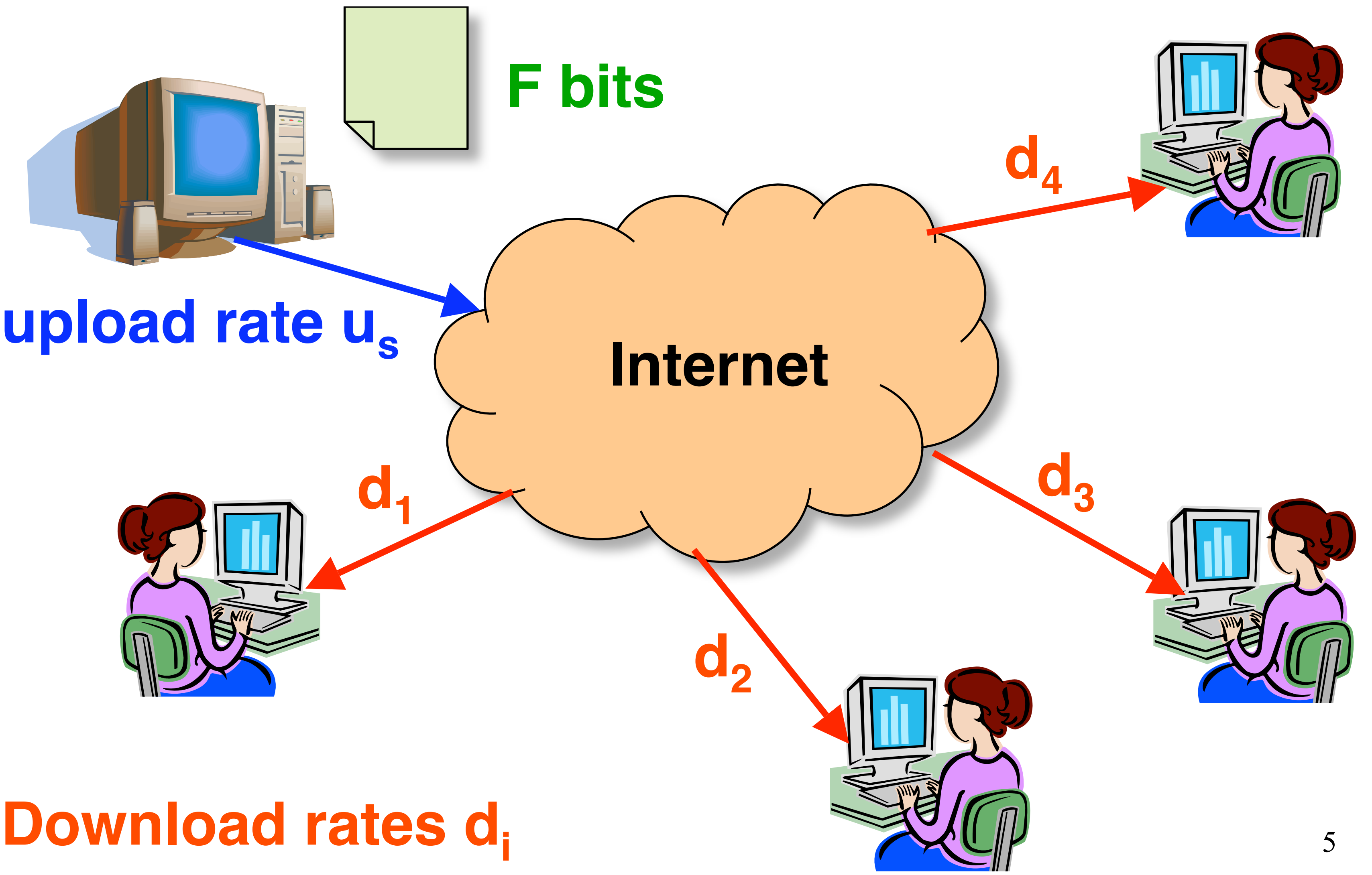
Client-Server Communication

- Client "sometimes on"
  - Initiates a request to the server when interested
  - E.g., Web browser on your laptop or cell phone
  - Doesn't communicate directly with other clients
  - Needs to know the server's address

- Server is "always on"
  - Services requests from many client hosts
  - E.g., Web server for the www.cnn.com Web site
  - Doesn't initiate contact with the clients
  - Needs a fixed, well-known address



Server Distributing a Large File



Download rates d_i

Server Distributing a Large File

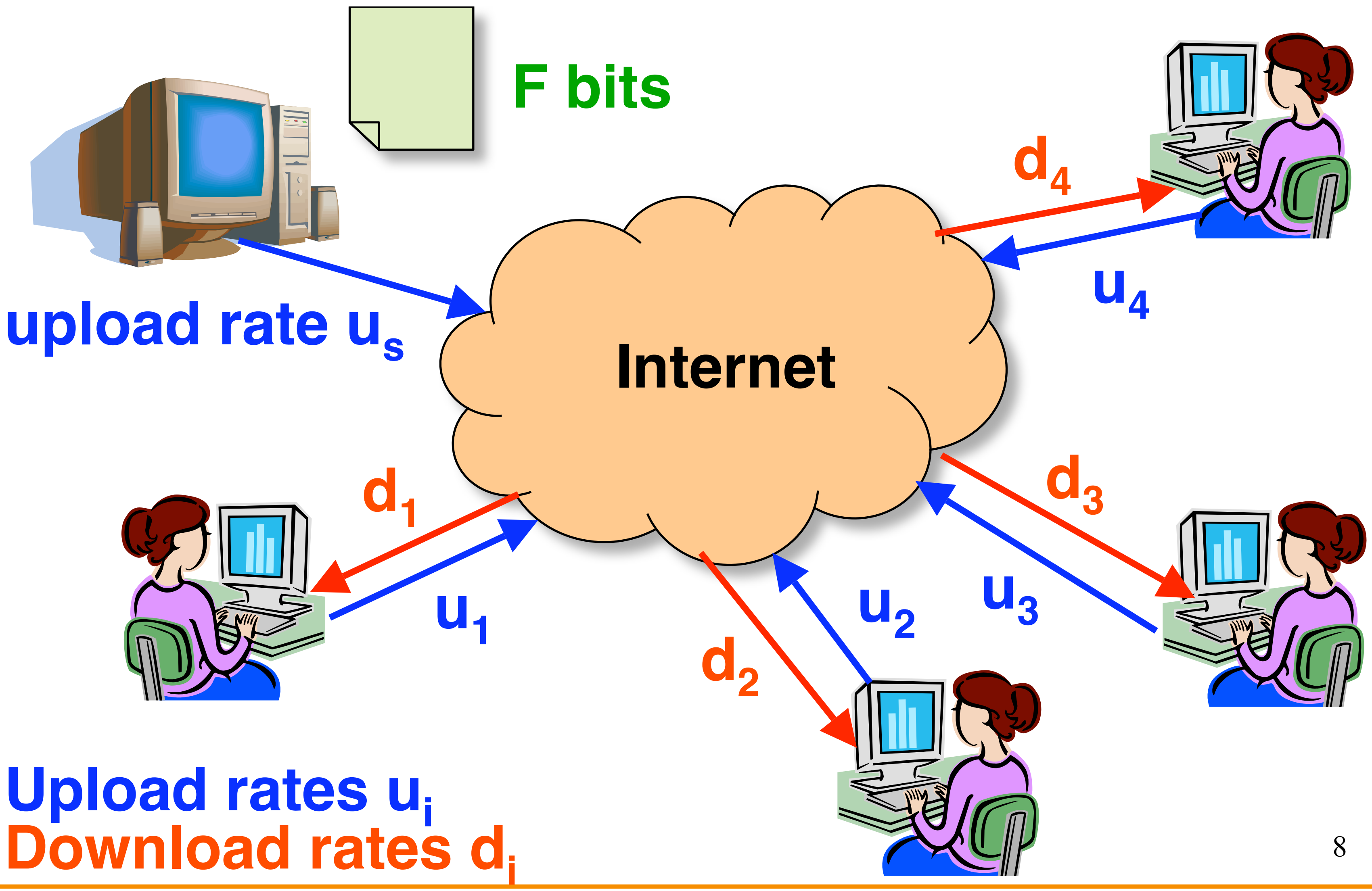
- Server sending a large file to N receivers
  - Large file with F bits
  - Single server with upload rate u_s
  - Download rate d_i for receiver i
- Server transmission to N receivers
  - Server needs to transmit NF bits
  - Takes at least NF/u_s time
- Receiving the data
  - Slowest receiver receives at rate $d_{min} = min_i\{d_i\}$
  - Takes at least F/d_{min} time
- Download time: $max\{NF/u_s, F/d_{min}\}$

Speeding Up the File Distribution

- Increase the upload rate from the server
  - Higher link bandwidth at the one server
  - Multiple servers, each with their own link
  - Requires deploying more infrastructure

- Alternative: have the receivers help
  - Receivers get a copy of the data
  - And then redistribute the data to other receivers
  - To reduce the burden on the server

Peers Help Distributing a Large File



Upload rates u_i
Download rates d_i

Peers Help Distributing a Large File

- Start with a single copy of a large file
  - Large file with F bits and server upload rate u_s
  - Peer i with download rate d_i and upload rate u_i
- Two components of distribution latency
  - Server must send each bit: min time F/u_s
  - Slowest peer receives each bit: min time F/d_{min}
- Total upload time using all upload resources
  - Total number of bits: NF
  - Total upload bandwidth $u_s + sum_i(u_i)$
- Total: $max\{F/u_s, F/d_{min}, NF/(u_s+sum_i(u_i))\}$

Comparing the Two Models

- Download time
  - Client-server: $max\{NF/u_s, F/d_{min}\}$
  - Peer-to-peer: $max\{F/u_s, F/d_{min}, NF/(u_s+sum_i(u_i))\}$
- Peer-to-peer is self-scaling
  - Much lower demands on server bandwidth
  - Distribution time grows only slowly with N
- But…
  - Peers may come and go
  - Peers need to find each other
  - Peers need to be willing to help each other

Challenges of Peer-to-Peer

- Peers come and go
  - Peers are intermittently connected
  - May come and go at any time
  - Or come back with a different IP address
- How to locate the relevant peers?
  - Peers that are online right now
  - Peers that have the content you want
- How to motivate peers to stay in system?
  - Why not leave as soon as download ends?
  - Why bother uploading content to anyone else?

Locating the Relevant Peers

- Three main approaches
  - –Central directory (Napster)
  - –Query flooding (Gnutella)
  - –Hierarchical overlay (Kazaa, modern Gnutella)
- Design goals
  - –Scalability
  - –Simplicity
  - –Robustness
  - –Plausible deniability

Peer-to-Peer Networks: Napster

- Napster history: the rise
  - January 1999: Napster version 1.0
  - May 1999: company founded
  - December 1999: first lawsuits
  - 2000: 80 million users

Shawn Fanning, Northeastern freshman

- Napster history: the fall
  - Mid 2001: out of business due to lawsuits
  - Mid 2001: dozens of P2P alternatives that were harder to touch, though these have gradually been constrained
  - 2003: growth of pay services like iTunes
- Napster history: the resurrection
  - 2003: Napster name/logo reconstituted as a pay service

Napster Technology: Directory Service

- User installing the software
  - Download the client program
  - Register name, password, local directory, etc.
- Client contacts Napster (via TCP)
  - Provides a list of music files it will share
  - … and Napster's central server updates the directory
- Client searches on a title or performer
  - Napster identifies online clients with the file
  - … and provides IP addresses
- Client requests the file from the chosen supplier
  - Supplier transmits the file to the client
  - Both client and supplier report status to Napster

Napster Technology: Properties

- Server's directory continually updated
  - Always know what music is currently available
  - Point of vulnerability for legal action
- Peer-to-peer file transfer
  - No load on the server
  - Plausible deniability for legal action (but not enough)
- Proprietary protocol
  - Login, search, upload, download, and status operations
  - No security: cleartext passwords and other vulnerability
- Bandwidth issues
  - Suppliers ranked by apparent bandwidth & response time

Napster: Limitations of Central Directory

- Single point of failure
- Performance bottleneck
- Copyright infringement

File transfer is decentralized, but locating content is highly centralized

- So, later P2P systems were more distributed
  - Gnutella went to the other extreme…

Peer-to-Peer Networks: Gnutella

- Gnutella history
  - 2000: J. Frankel & T. Pepper released Gnutella
  - Soon after: many other clients (e.g., Morpheus, Limewire, Bearshare)
  - 2001: protocol enhancements, e.g., “ultrapeers”
- Query flooding
  - Join: contact a few nodes to become neighbors
  - Publish: no need!
  - Search: ask neighbors, who ask their neighbors
  - Fetch: get file directly from another node

Gnutella: Query Flooding

- Fully distributed
  - No central server
- Public domain protocol
- Many Gnutella clients implementing protocol

Overlay network: graph

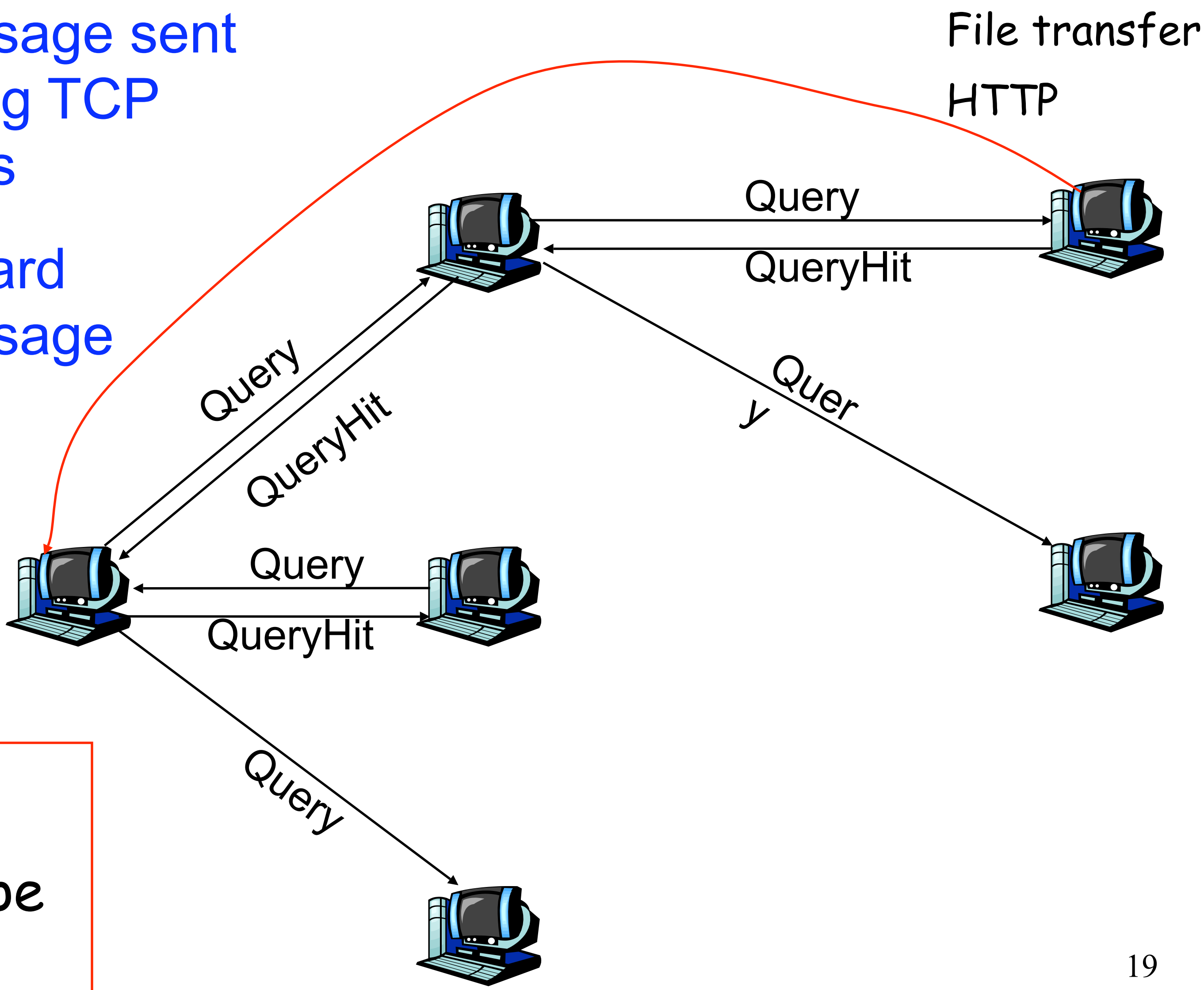
- Edge between peer X and Y if there's a TCP connection
- All active peers and edges is overlay net
- Given peer will typically be connected with < 10 overlay neighbors

Gnutella: Protocol

- Query message sent over existing TCP connections
- Peers forward Query message
- QueryHit sent over reverse path

Scalability:

limited scope flooding



Gnutella: Peer Joining

- Joining peer X must find some other peers
  - Start with a list of candidate peers
  - X sequentially attempts TCP connections with peers on list until connection setup with Y
- X sends Ping message to Y
  - Y forwards Ping message.
  - All peers receiving Ping message respond with Pong message
- X receives many Pong messages
  - X can then set up additional TCP connections

Gnutella: Pros and Cons

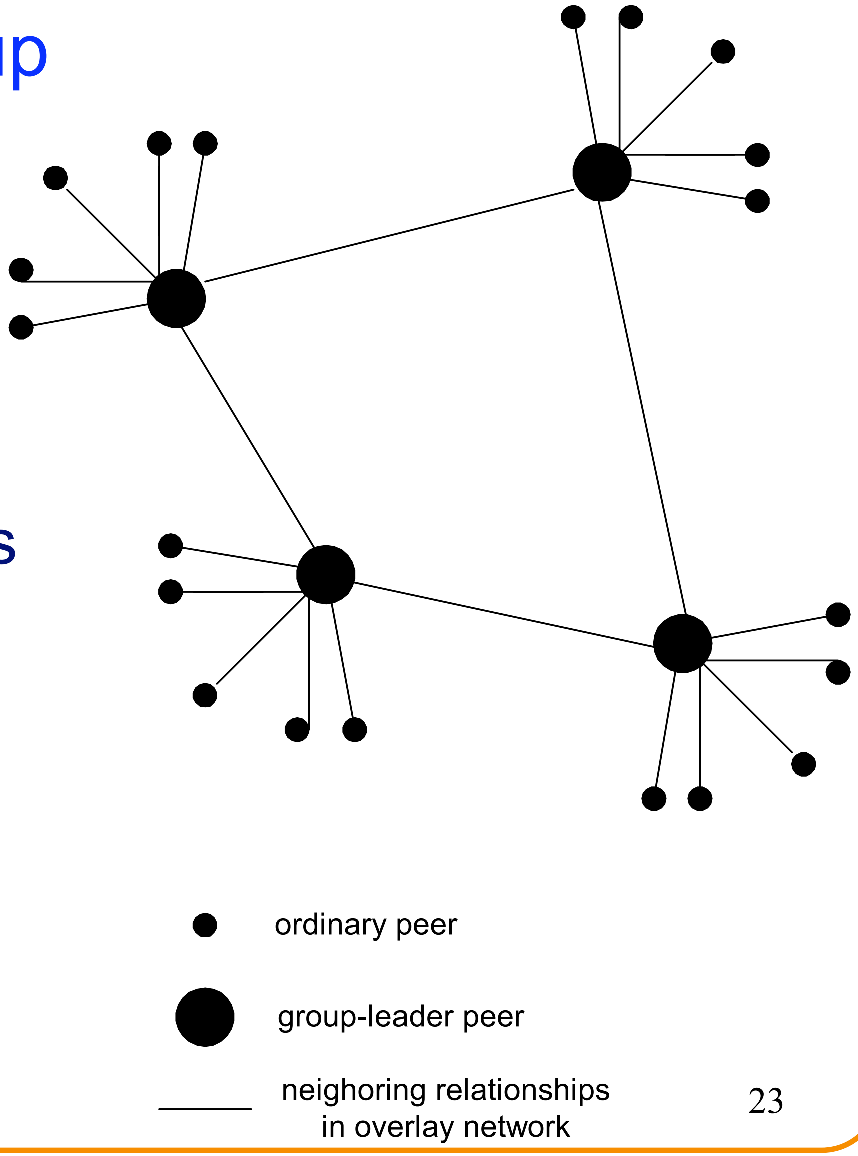
- Advantages
  - Fully decentralized
  - Search cost distributed
  - Processing per node permits powerful search semantics
- Disadvantages
  - Search scope may be quite large
  - Search time may be quite long
  - High overhead, and nodes come and go often

Peer-to-Peer Networks: KaZaA

- KaZaA history
  - 2001: created by Dutch company (Kazaa BV)
  - Single network called FastTrack used by other clients as well
  - Eventually the protocol changed so other clients could no longer talk to it
- Smart query flooding
  - Join: on start, the client contacts a super-node (and may later become one)
  - Publish: client sends list of files to its super-node
  - Search: send query to super-node, and the super-nodes flood queries among themselves
  - Fetch: get file directly from peer(s); can fetch from multiple peers at once

KaZaA: Exploiting Heterogeneity

- Each peer is either a group leader or assigned to a group leader
  - TCP connection between peer and its group leader
  - TCP connections between some pairs of group leaders
- Group leader tracks the content in all its children



KaZaA: Motivation for Super-Nodes

- Query consolidation
  - Many connected nodes may have only a few files
  - Propagating query to a sub-node may take more time than for the super-node to answer itself
- Stability
  - Super-node selection favors nodes with high up-time
  - How long you've been on is a good predictor of how long you'll be around in the future

Peer-to-Peer Networks: BitTorrent

- BitTorrent history and motivation
  - 2002: B. Cohen debuted BitTorrent
  - Key motivation: popular content
    - Popularity exhibits temporal locality (Flash Crowds)
    - E.g., Slashdot/Digg effect, CNN Web site on 9/11, release of a new movie or game
  - Focused on efficient *fetching*, not searching
    - Distribute same file to many peers
    - Single publisher, many downloaders
  - Preventing free-loading

BitTorrent: Simultaneous Downloading

- Divide large file into many pieces
  - Replicate different pieces on different peers
  - A peer with a complete piece can trade with other peers
  - Peer can (hopefully) assemble the entire file
- Allows simultaneous downloading
  - Retrieving different parts of the file from different peers at the same time
  - And uploading parts of the file to peers
  - Important for very large files

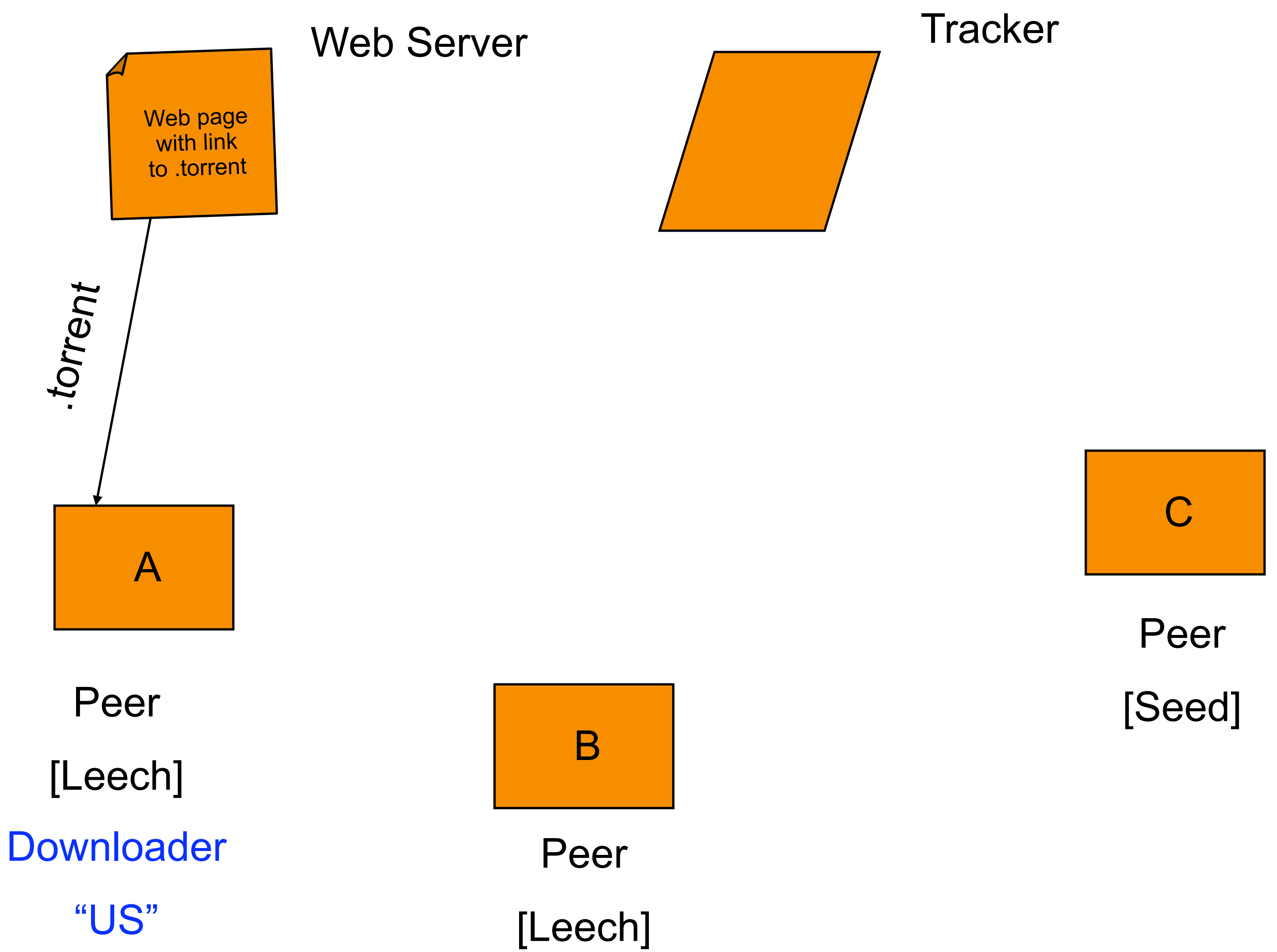
BitTorrent: Tracker

- Infrastructure node
  - Keeps track of peers participating in the torrent
- Peers register with the tracker
  - Peer registers when it arrives
  - Peer periodically informs tracker it is still there
- Tracker selects peers for downloading
  - Returns a random set of peers
  - Including their IP addresses
  - So the new peer knows who to contact for data
- Can have "trackerless" system using DHT

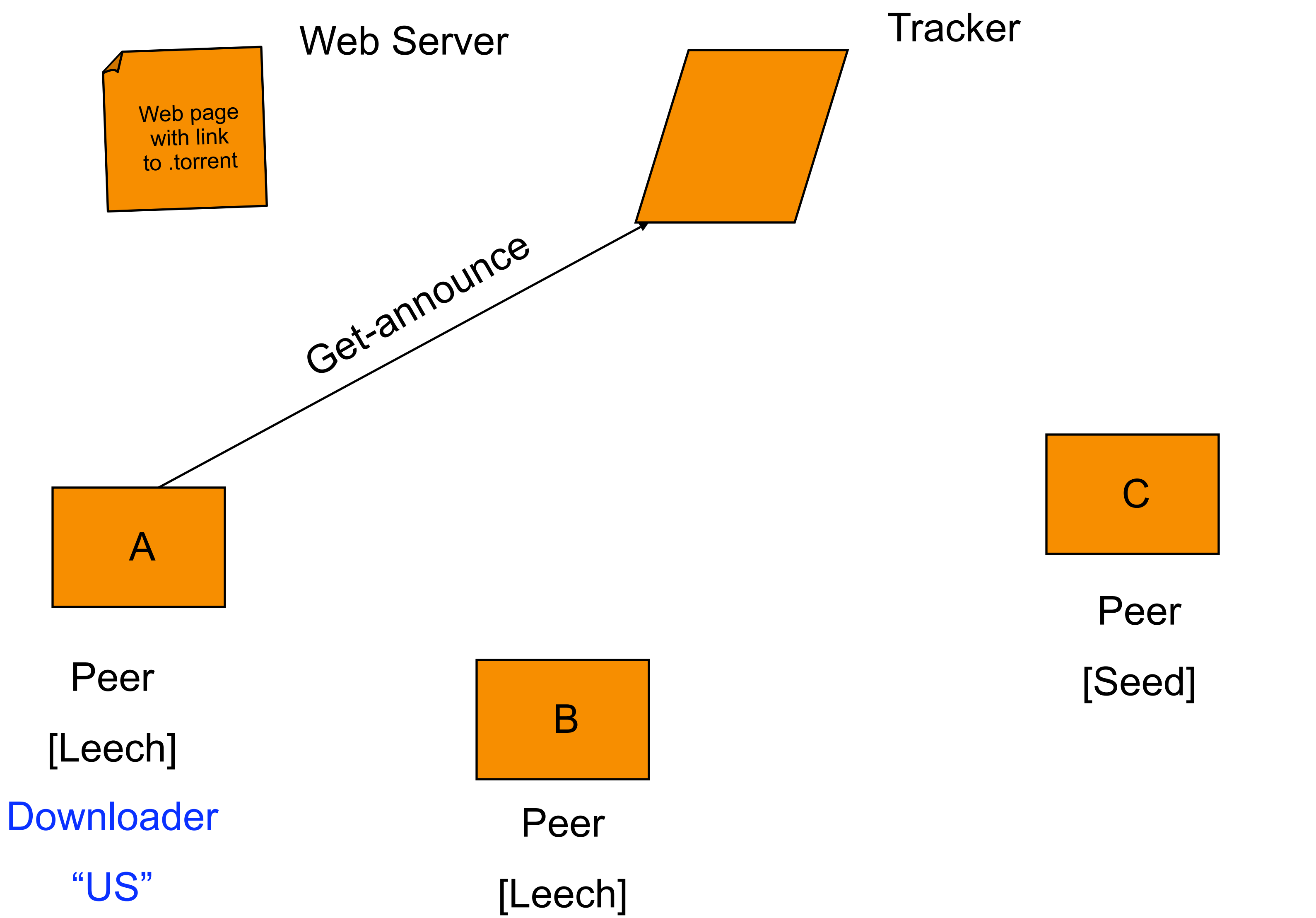
BitTorrent: Chunks

- Large file divided into smaller pieces
  - Fixed-sized chunks
  - Typical chunk size of 256 Kbytes
- Allows simultaneous transfers
  - Downloading chunks from different neighbors
  - Uploading chunks to other neighbors
- Learning what chunks your neighbors have
  - Periodically asking them for a list
- File done when all chunks are downloaded

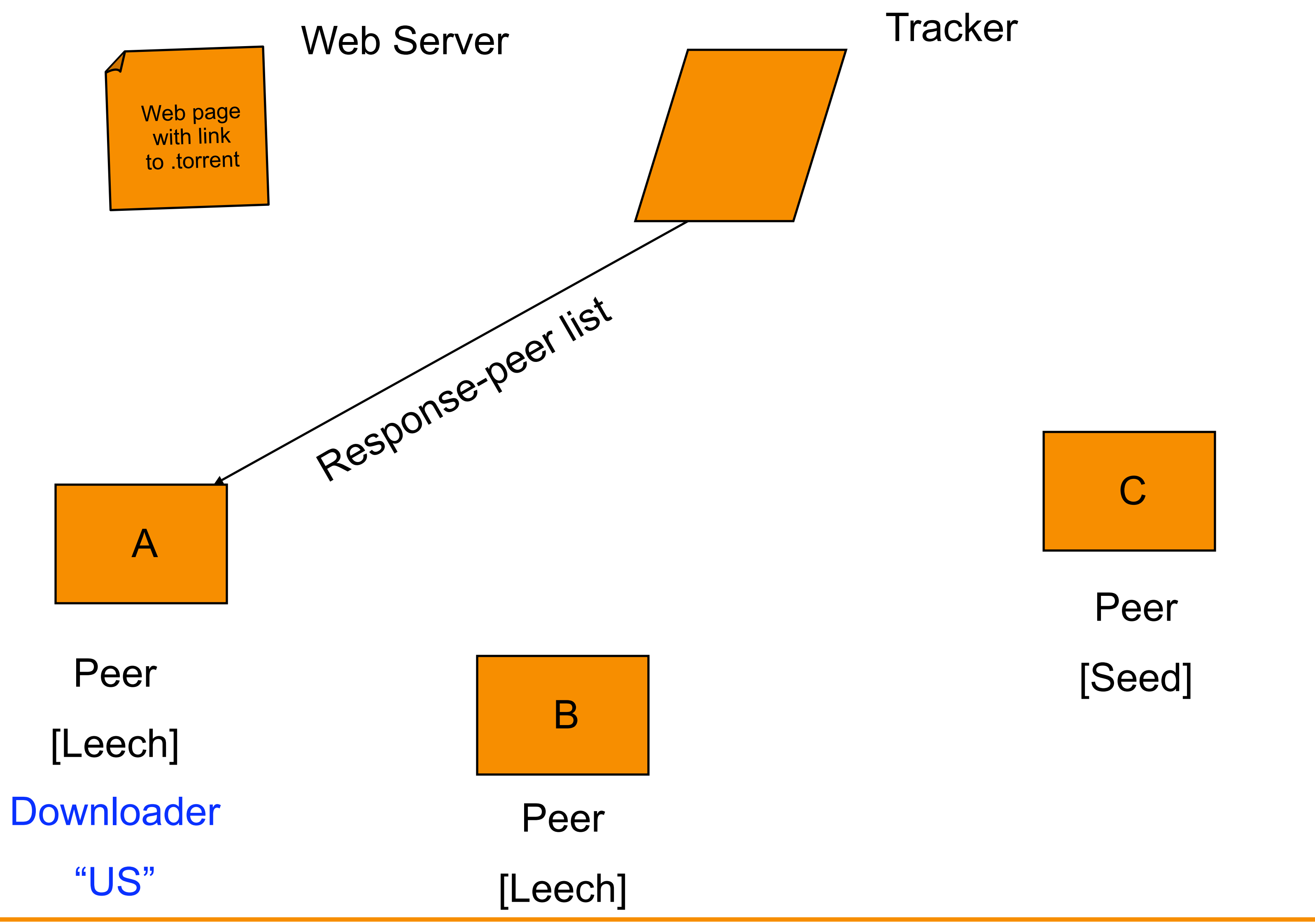
BitTorrent: Overall Architecture



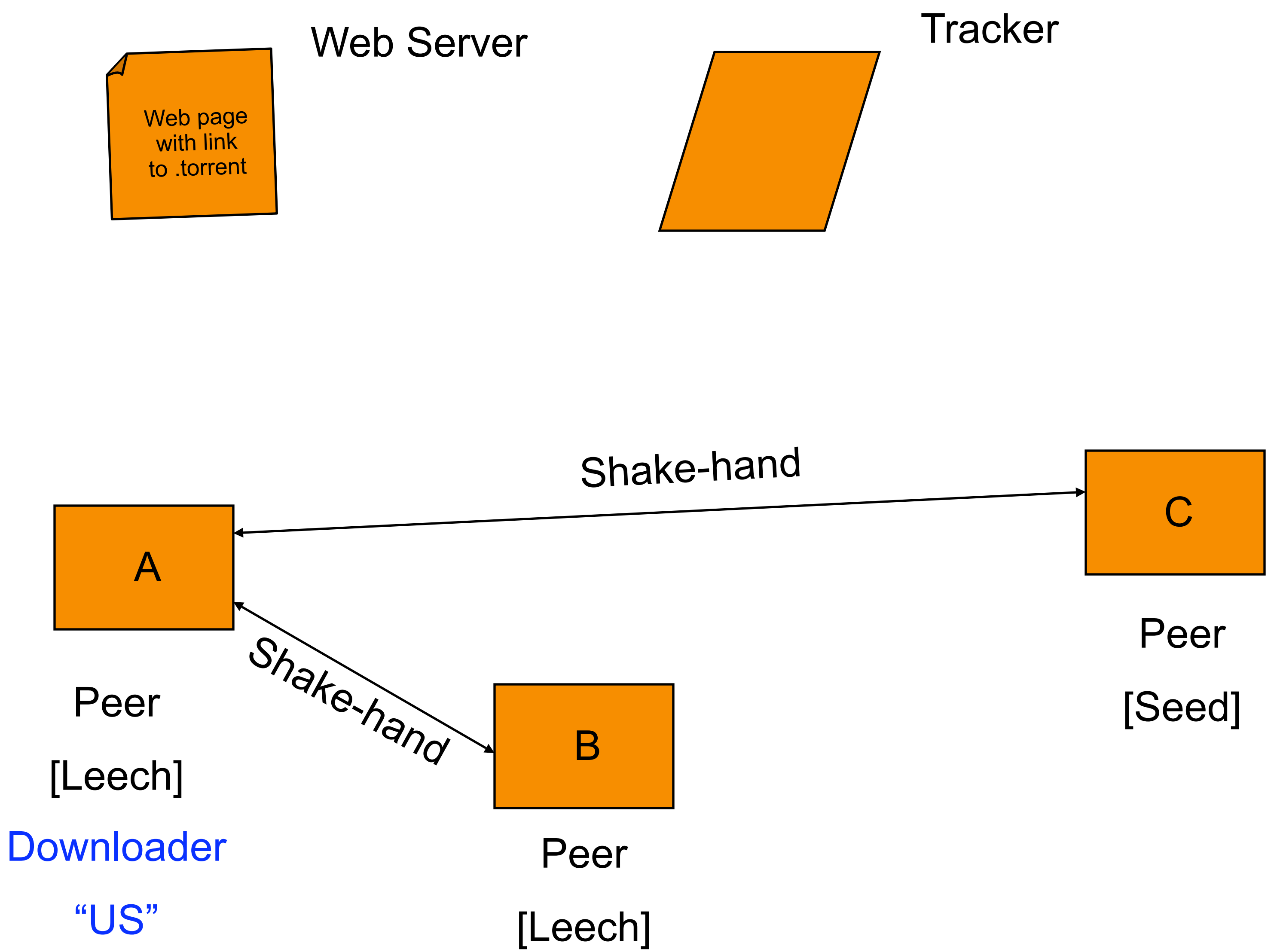
BitTorrent: Overall Architecture



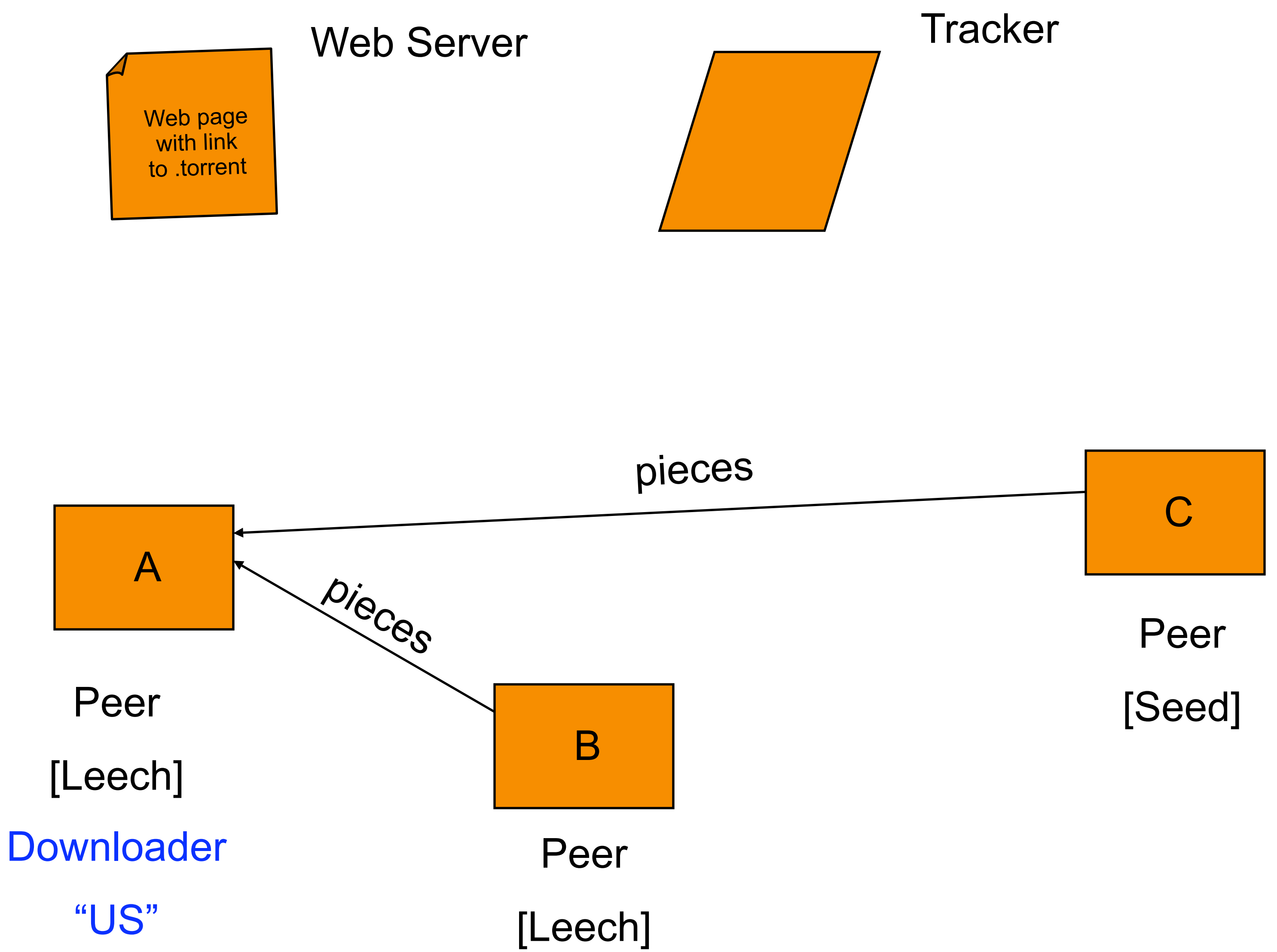
BitTorrent: Overall Architecture



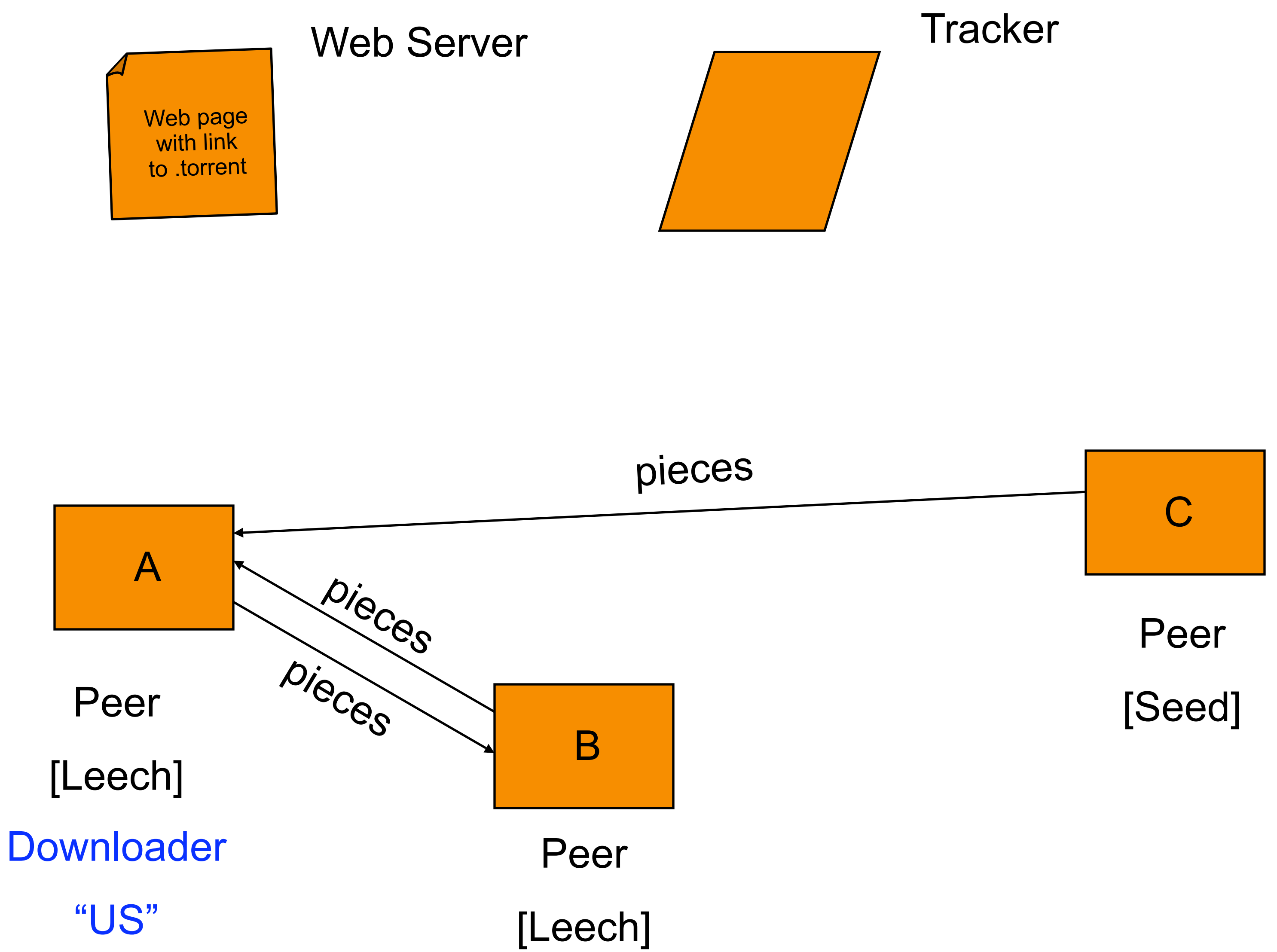
BitTorrent: Overall Architecture



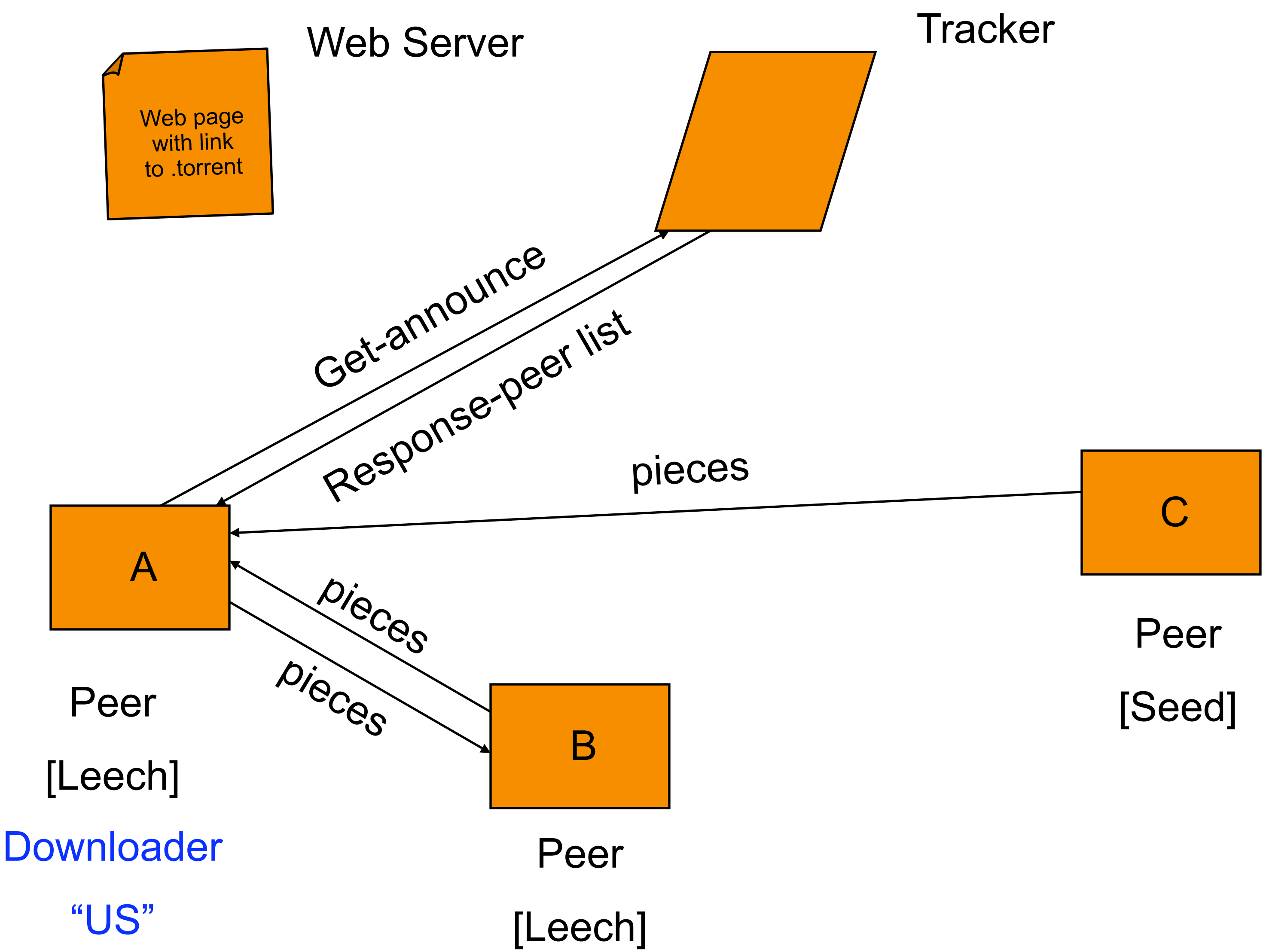
BitTorrent: Overall Architecture



BitTorrent: Overall Architecture



BitTorrent: Overall Architecture



BitTorrent: Chunk Request Order

- Which chunks to request?
  - Could download in order
  - Like an HTTP client does
- Problem: many peers have the early chunks
  - Peers have little to share with each other
  - Limiting the scalability of the system
- Problem: eventually nobody has rare chunks
  - E.g., the chunks need the end of the file
  - Limiting the ability to complete a download
- Solutions: random selection and rarest first

BitTorrent: Rarest Chunk First

- Which chunks to request first?
  - The chunk with the fewest available copies
  - I.e., the rarest chunk first
- Benefits to the peer
  - Avoid starvation when some peers depart
- Benefits to the system
  - Avoid starvation across all peers wanting a file
  - Balance load by equalizing # of copies of chunks

Free-Riding Problem in P2P Networks

- Vast majority of users are free-riders
  - Most share no files and answer no queries
  - Others limit # of connections or upload speed
- A few "peers" essentially act as servers
  - A few individuals contributing to the public good
  - Making them hubs that basically act as a server
- BitTorrent prevent free riding
  - Allow the fastest peers to download from you
  - Occasionally let some free loaders download

Bit-Torrent: Preventing Free-Riding

- Peer has limited upload bandwidth
  - And must share it among multiple peers
- Prioritizing the upload bandwidth: tit for tat
  - Favor neighbors that are uploading at highest rate
- Rewarding the top four neighbors
  - Measure download bit rates from each neighbor
  - Reciprocates by sending to the top four peers
  - Recompute and reallocate every 10 seconds
- Optimistic unchoking
  - Randomly try a new neighbor every 30 seconds
  - So new neighbor has a chance to be a better partner

BitTyrant: Gaming BitTorrent

- BitTorrent can be gamed, too
  - Peer uploads to top N peers at rate 1/N
  - E.g., if N=4 and peers upload at 15, 12, 10, 9, 8, 3
  - … then peer uploading at rate 9 gets treated quite well
- Best to be the Nth peer in the list, rather than 1st
  - Offer just a bit more bandwidth than the low-rate peers
  - But not as much as the higher-rate peers
  - And you'll still be treated well by others
- BitTyrant software
  - http://bittyrant.cs.washington.edu/

BitTorrent Today

- Significant fraction of Internet traffic
  - Estimated at 30%
  - Though this is hard to measure
- Problem of incomplete downloads
  - Peers leave the system when done
  - Many file downloads never complete
  - Especially a problem for less popular content
- Still lots of legal questions remains
- Further need for incentives

Conclusions

- Peer-to-peer networks
  - Nodes are end hosts
  - Primarily for file sharing, and recently telephony
- Finding the appropriate peers
  - Centralized directory (Napster)
  - Query flooding (Gnutella)
  - Super-nodes (KaZaA)
- BitTorrent
  - Distributed download of large files
  - Anti-free-riding techniques
- Great example of how change can happen so quickly in application-level protocols