CSE 486/586 Distributed Systems
Peer-to-Peer Architectures

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Last Time

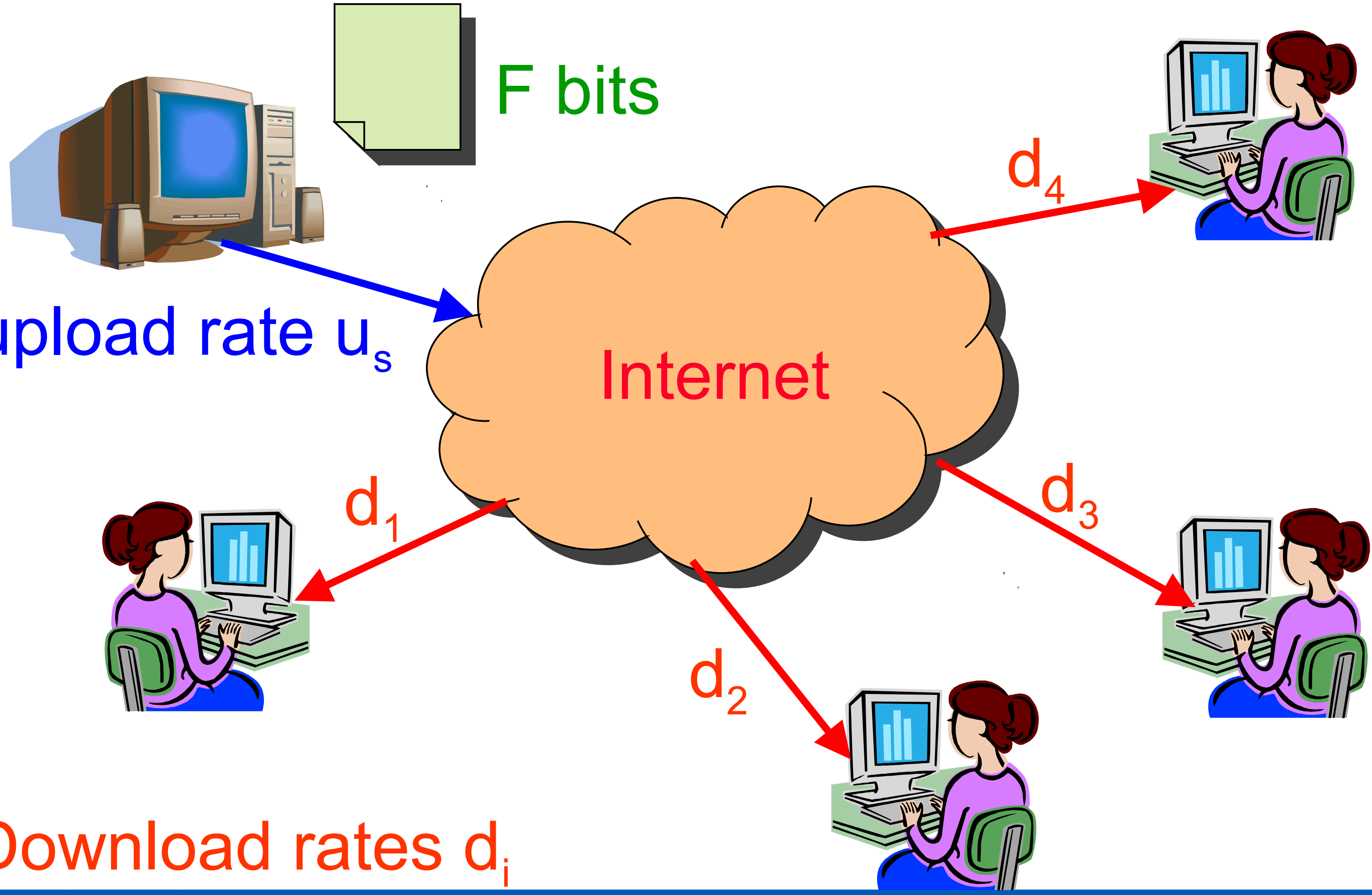
- Gossiping
  - Multicast
  - Failure detection

Today's Question

- How do we organize the nodes in a distributed system?
- Up to the 90's
  - Prevalent architecture: client-server (or master-slave)
  - Unequal responsibilities
- Now
  - Emerged architecture: peer-to-peer
  - Equal responsibilities
- Today: studying peer-to-peer as a paradigm
  - Not just as a file-sharing application
  - We will use file-sharing as the main example
  - Learn the techniques and principles

Motivation: Distributing a Large File

- A client-server architecture can do it…



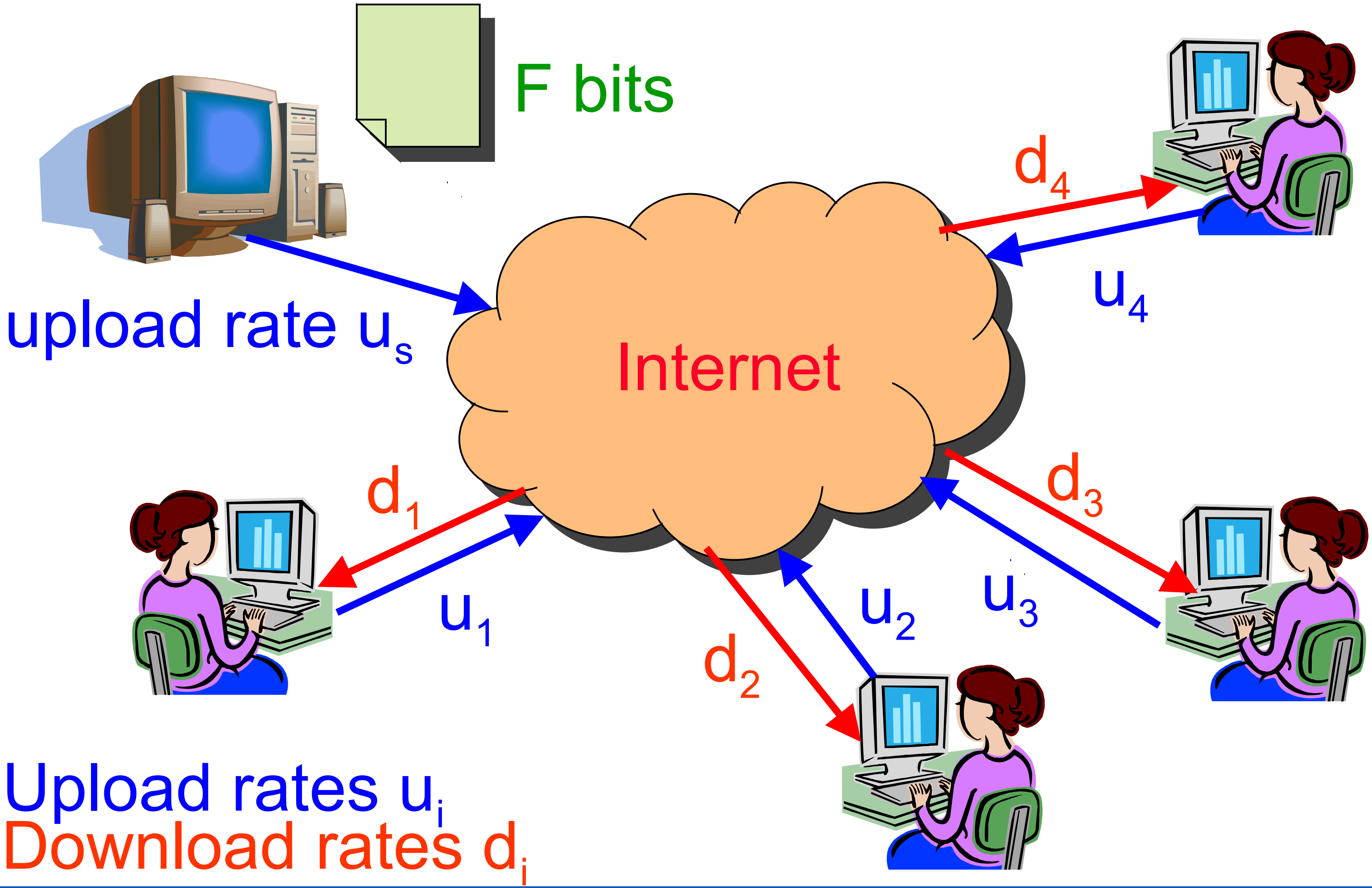
Download rates d_i

Motivation: Distributing a Large File

- …but sometimes not good enough.
  - Limited bandwidth
  - One server can only serve so many clients.
- Can we increase the server upload rate?
  - 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
  - Thus reducing the burden on the server

Motivation: Distributing a Large File

- Peer-to-peer to help



Challenges of Peer-to-Peer

- Peers may come and go
  - Peers are often intermittently connected
  - May come and go at any time
  - Returning peers may have a different IP address
- How do we locate relevant peers?
  - Peers that are online right now
  - Peers that have the content you want
- How can we motivate peers to stay in system?
  - Why not leave as soon as the download ends?
  - Why bother uploading content to anyone else?
- How can we download efficiently?
  - The faster, the better

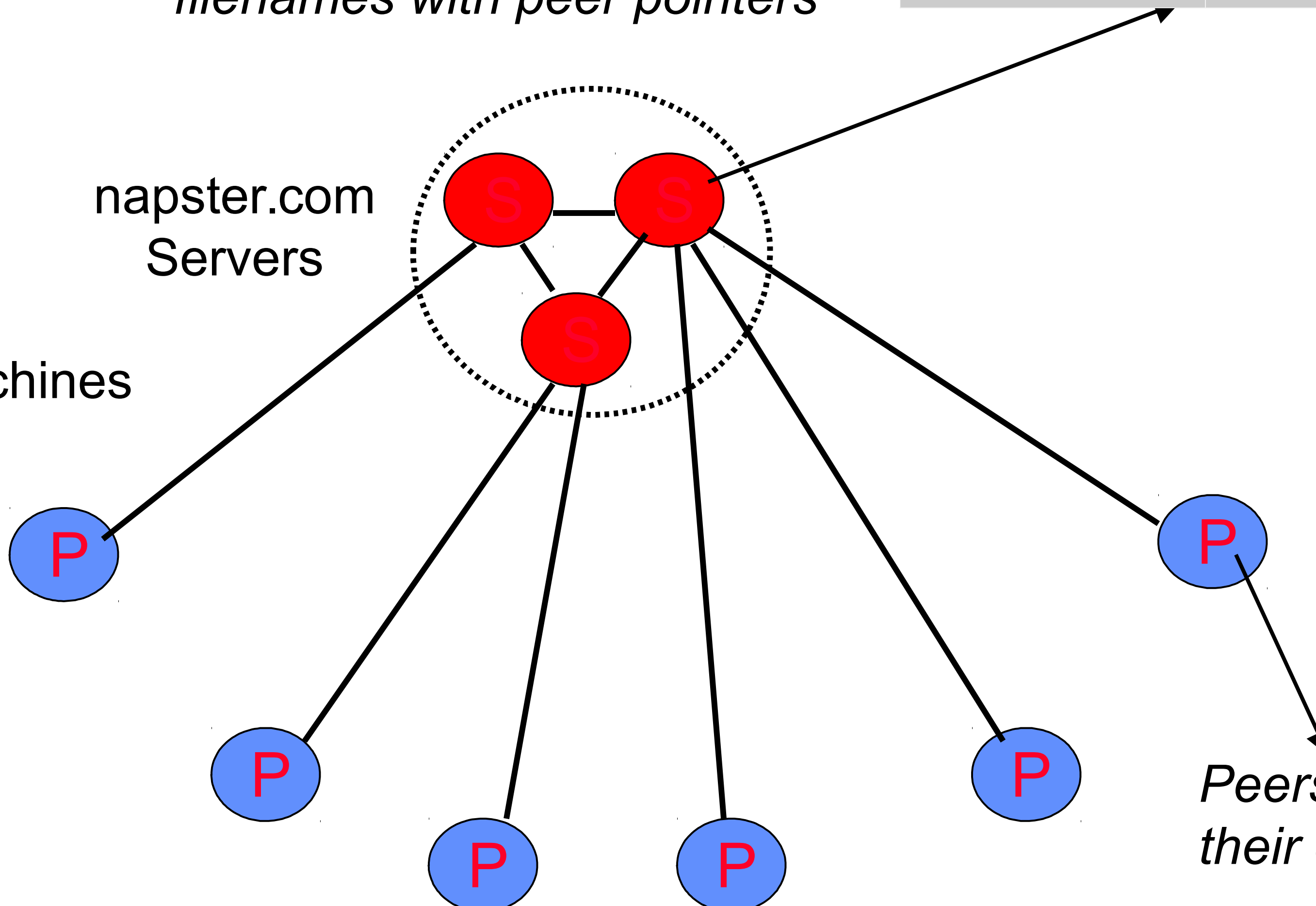
Locating Relevant Peers

- Evolution of peer-to-peer architectures
  - Central directory (Napster)
  - Query flooding (Gnutella)
  - Hierarchical overlay (Kazaa, modern Gnutella)
- Design goals
  - Scalability
  - Simplicity
  - Robustness
  - Plausible deniability

The First: Napster

Filename	File info
...	...
PennyLane.mp3	Beatles, @128.84.92.23:1006
...	...

Store a directory: i.e., filenames with peer pointers

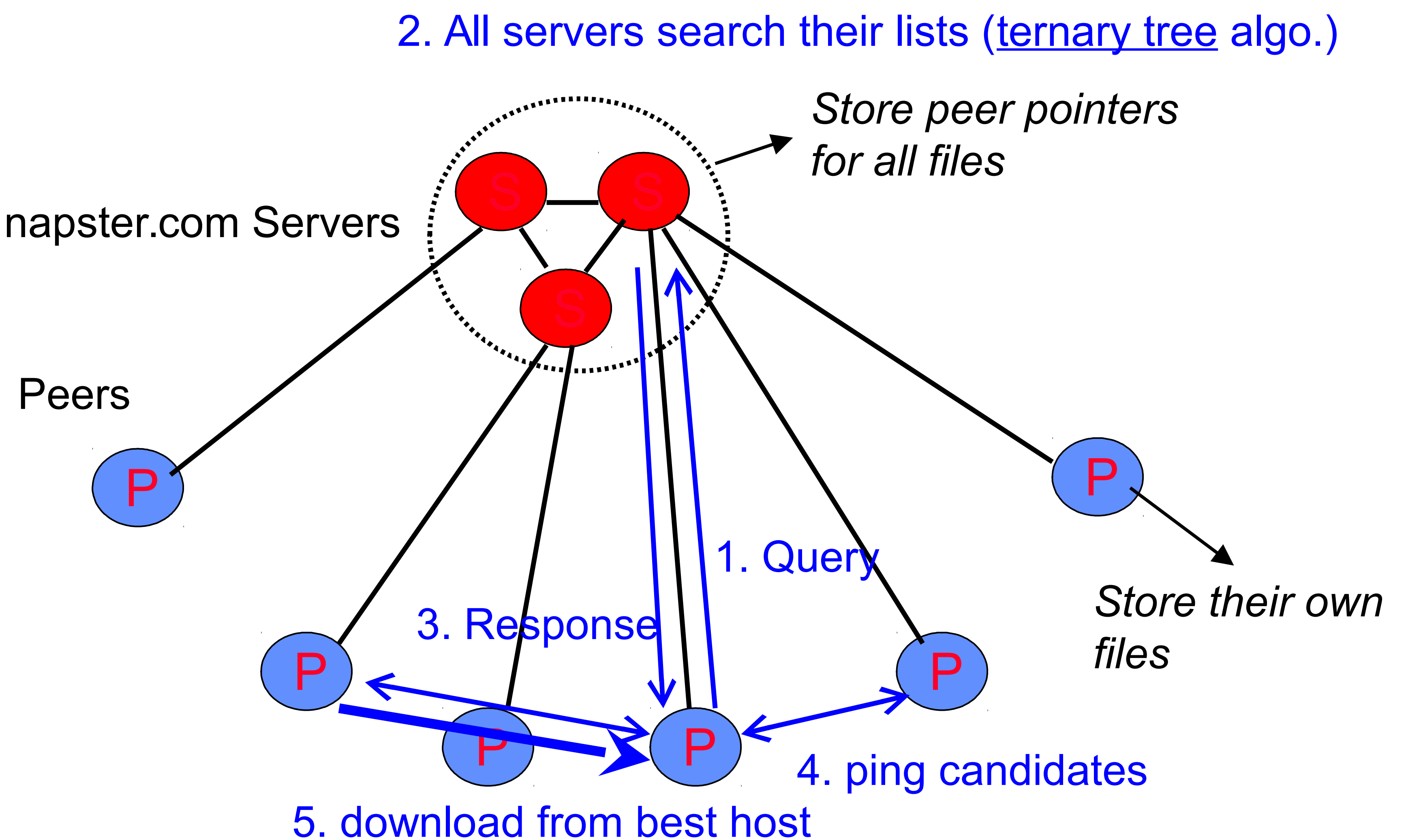


napster.com
Servers

Client machines
("Peers")

Peers store their own files

The First: Napster



The First: Napster

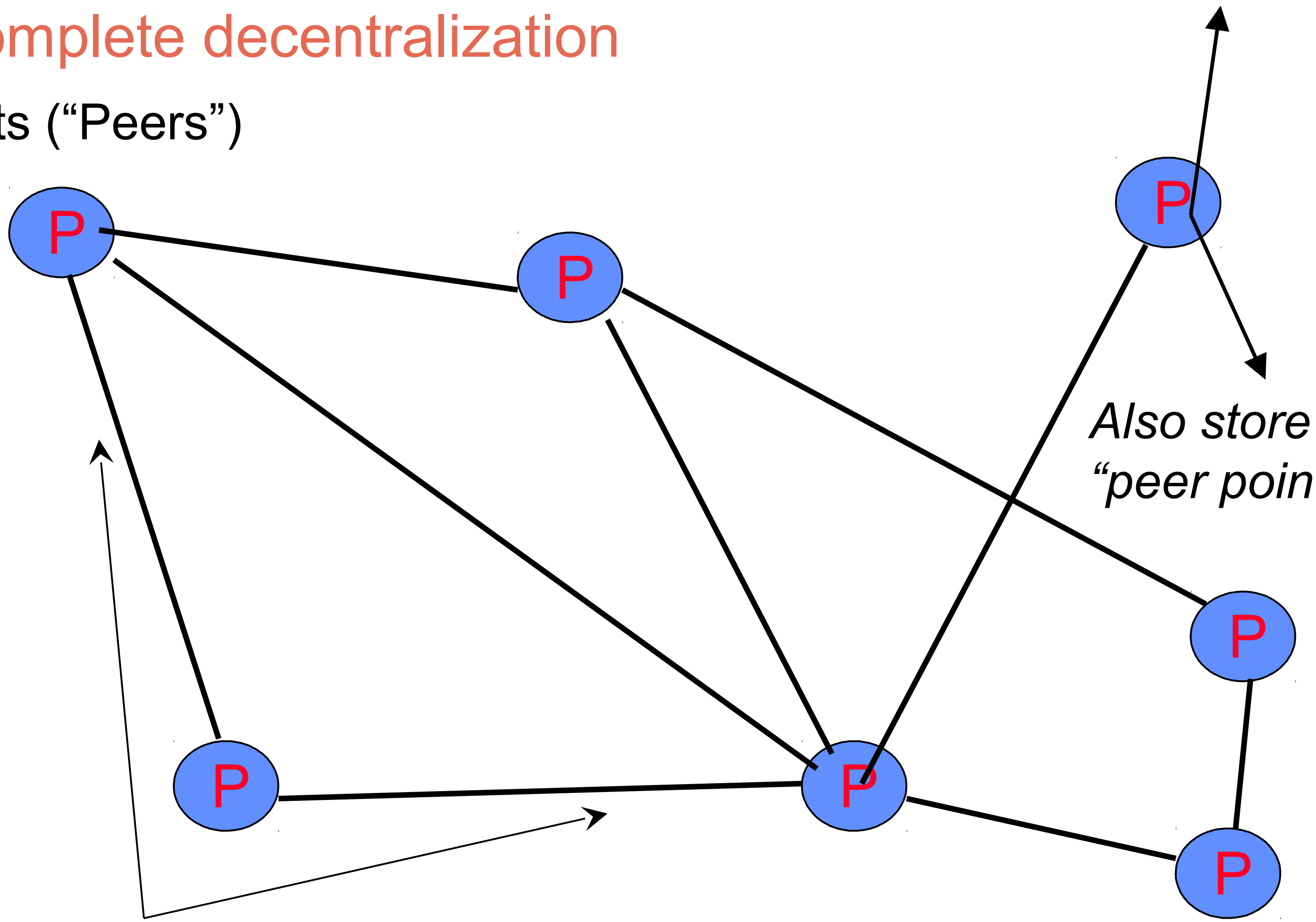
- Server directory is continually updated
  - Always know what files are currently available
  - Point of vulnerability for legal action
- Peer-to-peer file transfer
  - No load on the server
  - Plausible deniability for legal action (turns out not enough)
- Proprietary protocol
  - Login, search, upload,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

The Second: Gnutella

Peers store their own files

- Complete decentralization

Servants ("Peers")

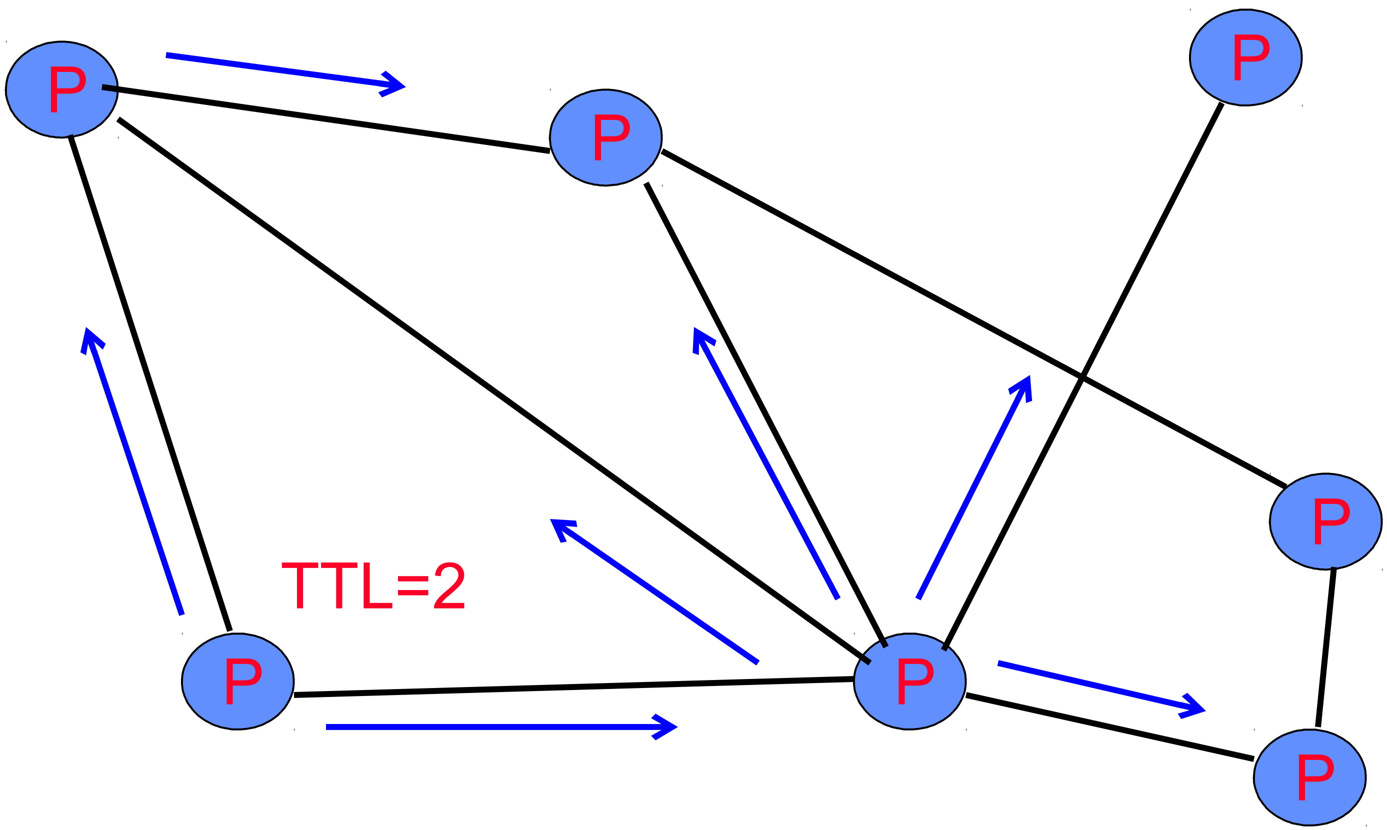


Also store "peer pointers"

Connected in an overlay graph (== each link is an implicit Internet path)

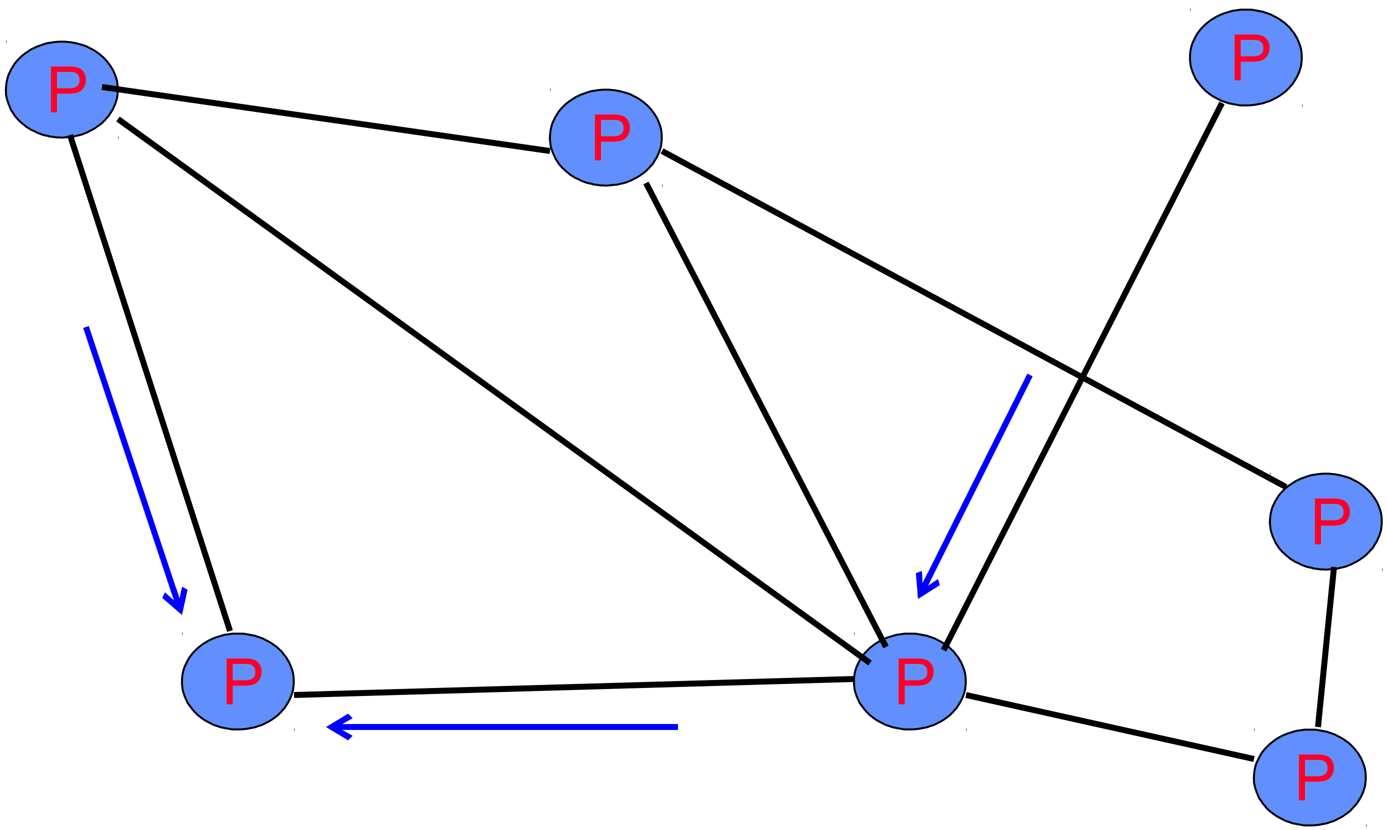
The Second: Gnutella

Queries flooded out, TTL-restricted, forwarded only once



The Second: Gnutella

Successful results (QueryHits) routed on reverse path

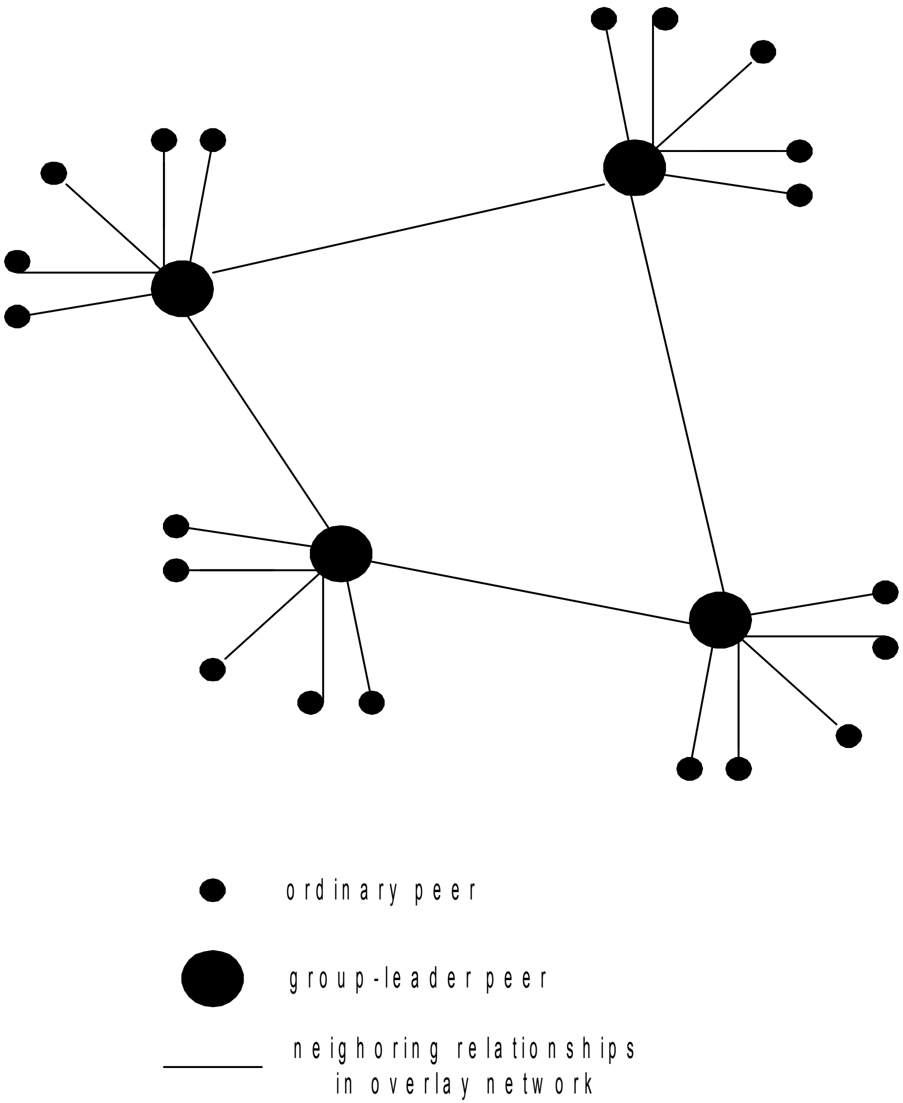


The Second: Gnutella

- Advantages
  - Fully decentralized
  - Distributed search cost
  - Processing at nodes 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

The Third: KaAzA

- Middle ground between Napster & Gnutella
- Each peer is either a group leader (supernode) 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 of its children



The Third: KaZaA

- A supernode stores a directory listing
  - <filename,peer pointer>, similar to Napster servers
- Supernode membership changes over time
- Any peer can become a supernode, provided it has earned enough reputation
  - Kazaalite:
    - Participation level (reputation) of a user between 0 and 1000
    - Initially 10, then affected by length of periods of connectivity and total number of uploads
  - More sophisticated reputation schemes invented, especially based on economics
- A peer searches by contacting a nearby supernode

Now: 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
- Bram Cohen (the inventor) attended UB.
- Focused on efficient fetching, not searching
  - Distribute same file to many peers
  - Single publisher, many downloaders
- Preventing free-loading

Key Feature: Parallel Downloading

- Divide large file into many pieces (chunks)
  - Replicate different chunks on different peers
  - A peer with any complete chunk can trade with other peers
  - Peer can (hopefully) assemble the entire file
- Allows simultaneous downloading
  - Retrieve chunks from many peers simultaneously
  - Upload chunks to other (maybe overlapping) peers
  - Important for very large files
- System Components
  - Web server
  - Tracker
  - Peers

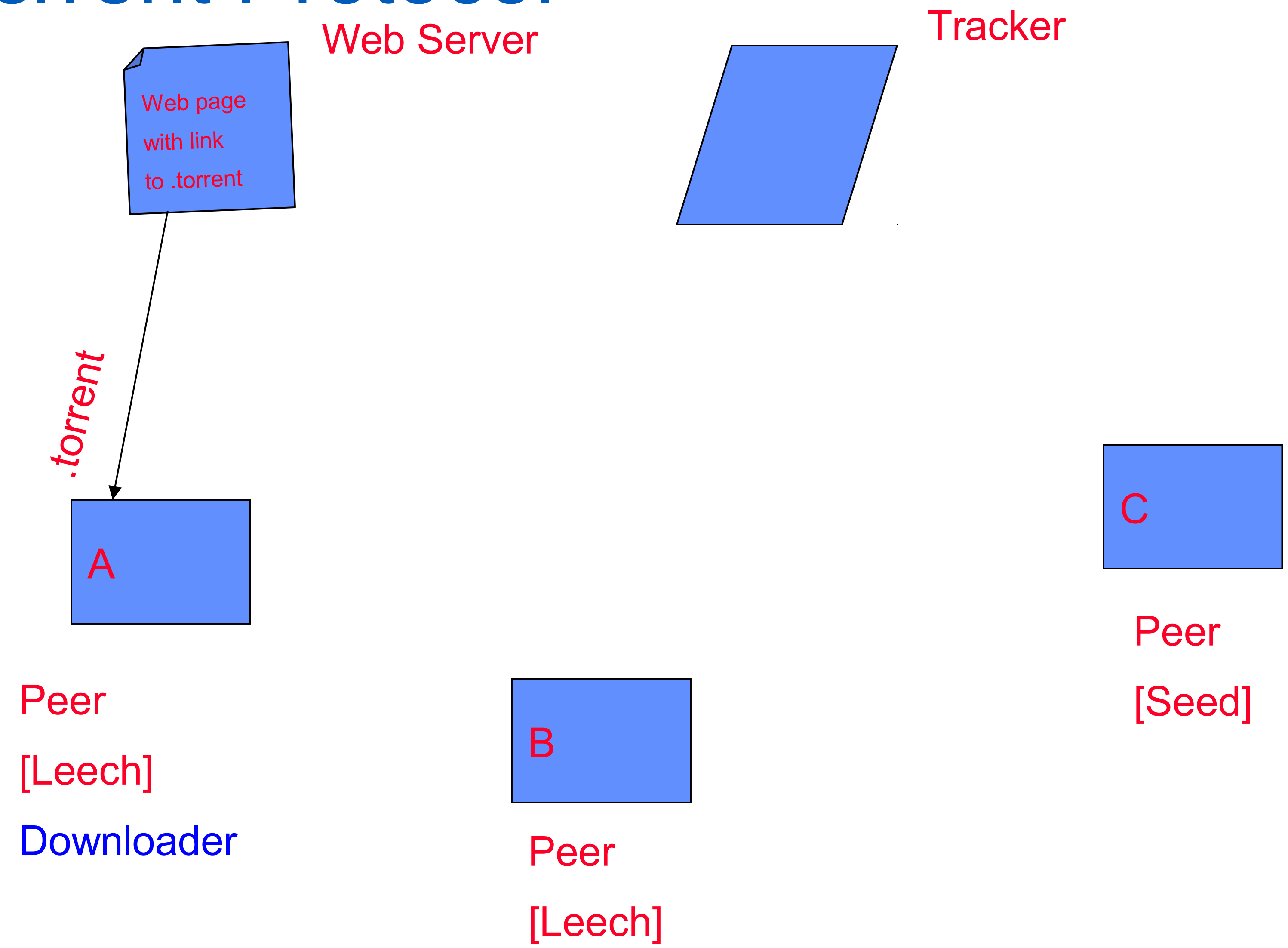
Tracker

- 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 be "trackerless" using DHT

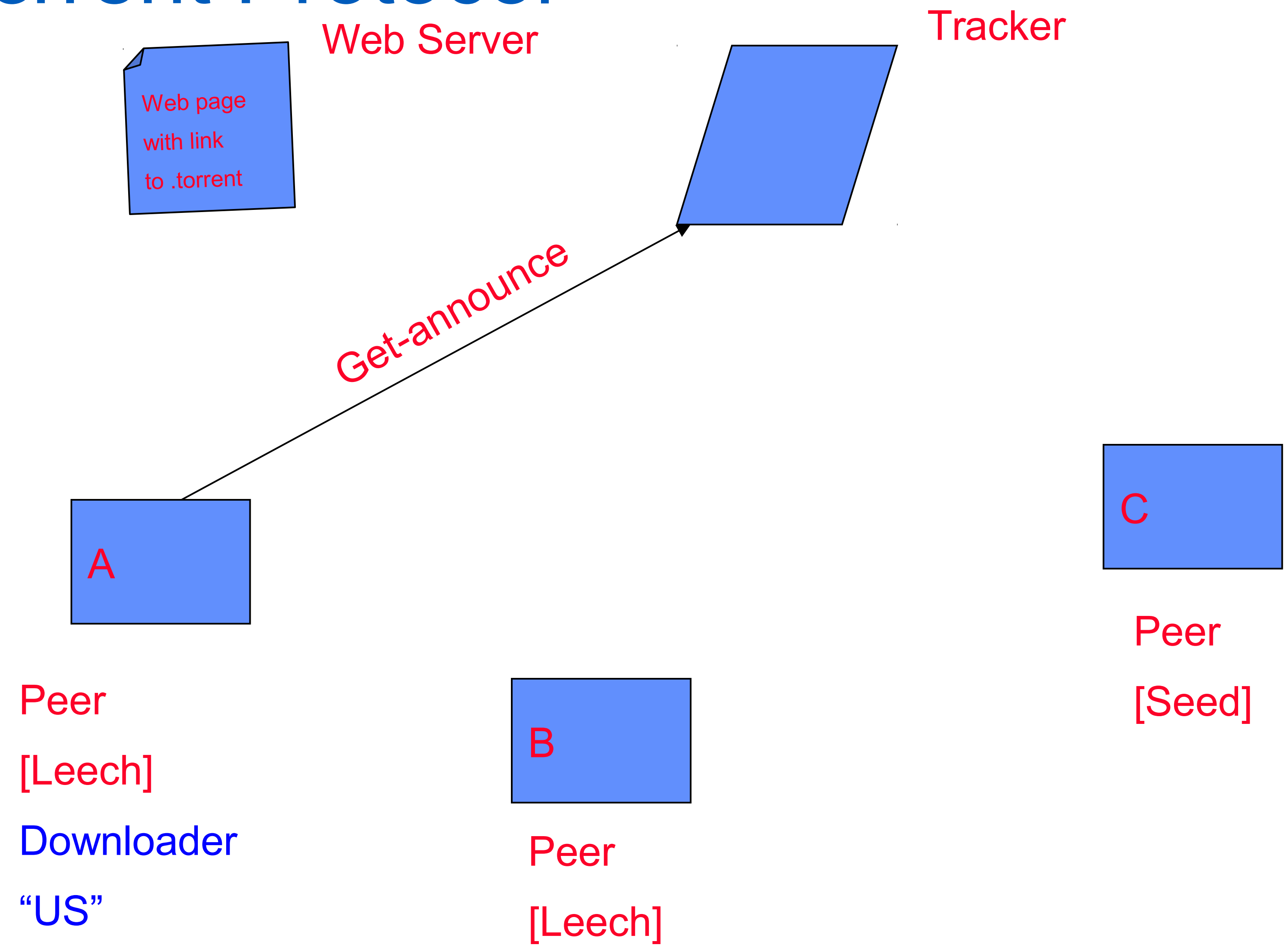
Chunks

- Large file divided into smaller pieces
  - Fixed-sized chunks
  - Typical chunk size of 256 Kbytes
  - Each chunk is checksummed (using SHA-1)
- Allows simultaneous transfers
  - Downloading different chunks from different neighbors
  - Uploading complete chunks to other neighbors
- Peers learn what chunks their neighbors have
  - Periodically ask them for a list
- File done when all chunks are downloaded

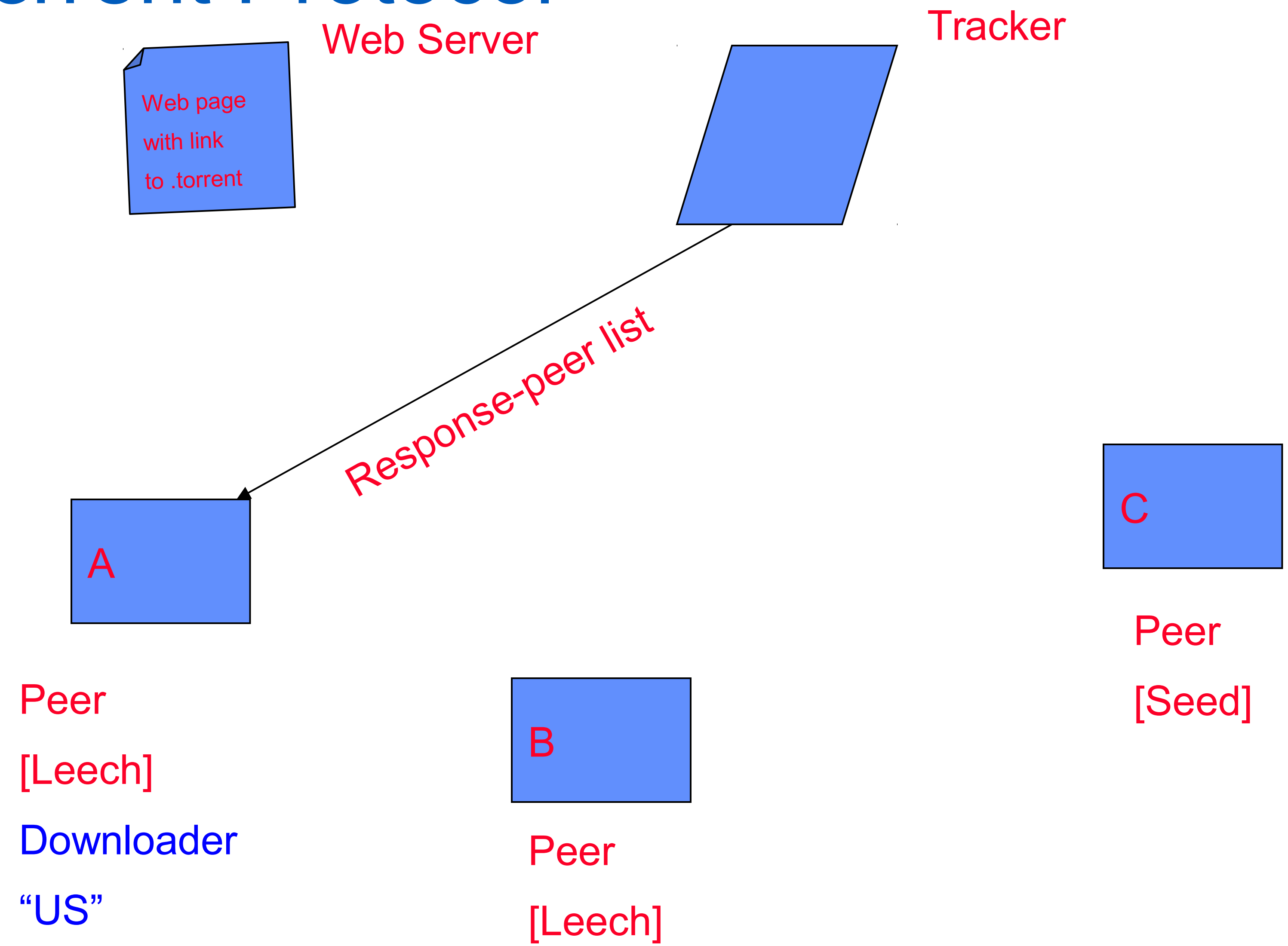
BitTorrent Protocol



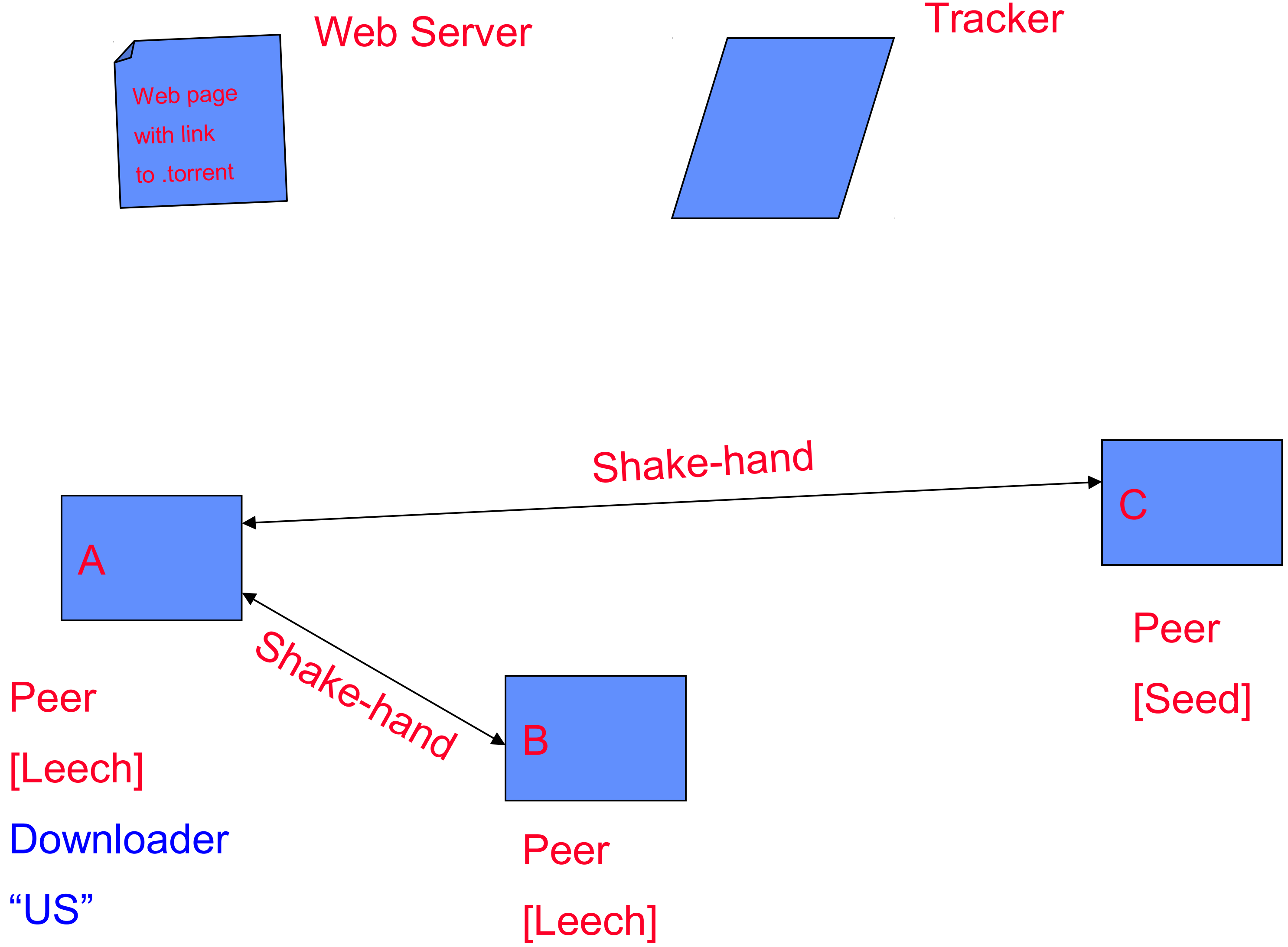
BitTorrent Protocol



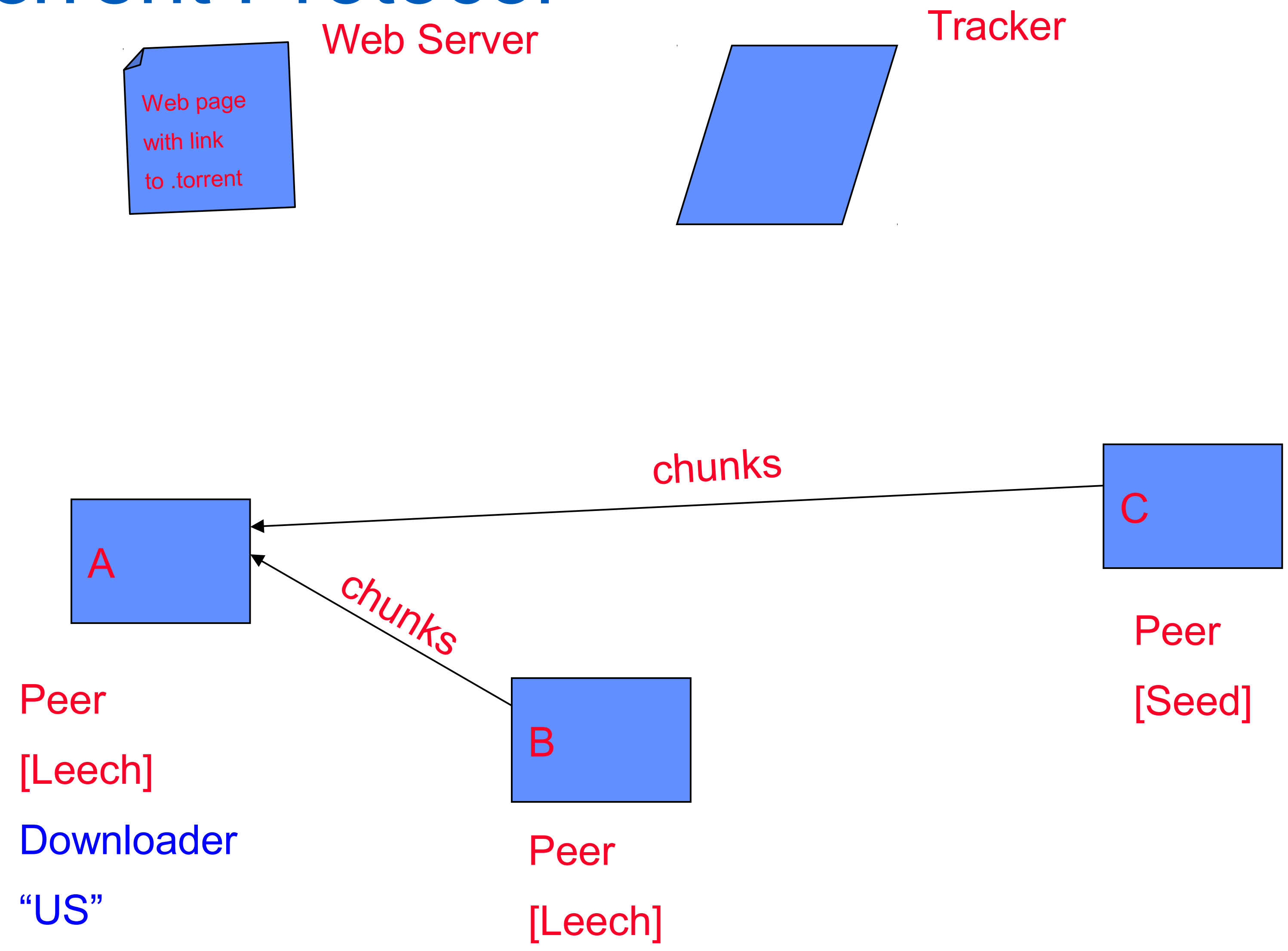
BitTorrent Protocol



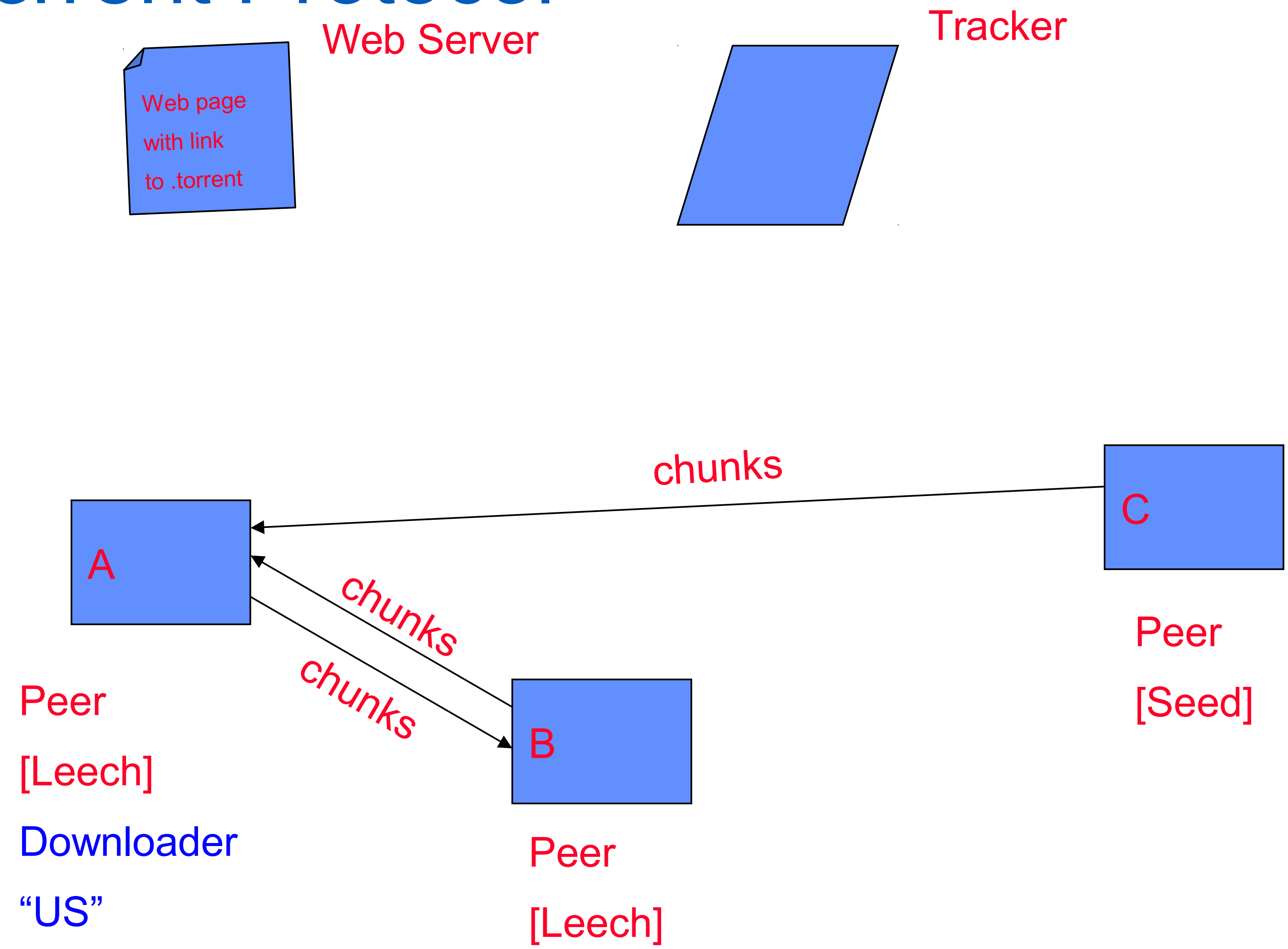
BitTorrent Protocol



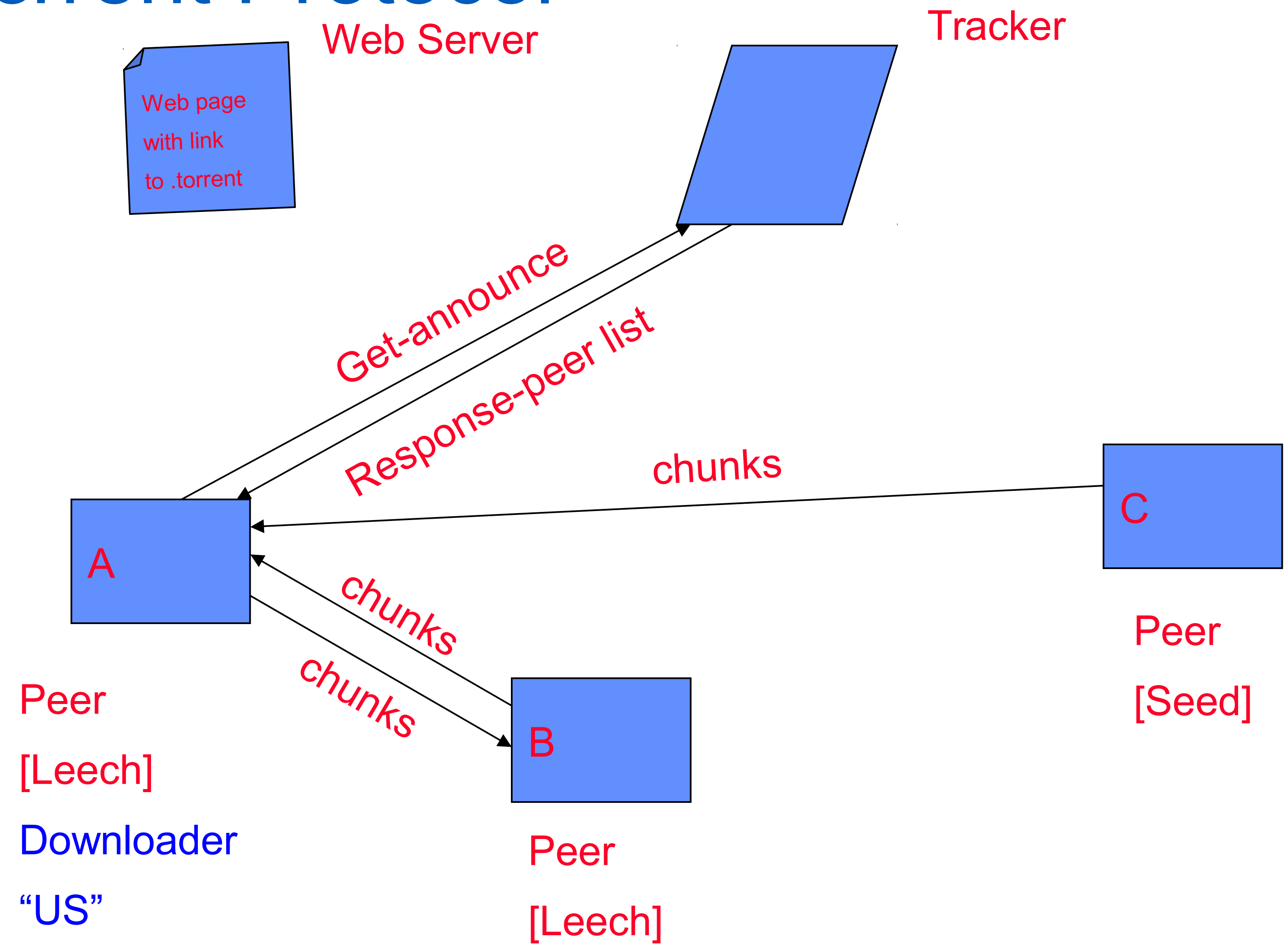
BitTorrent Protocol



BitTorrent Protocol



BitTorrent Protocol



Chunk Request Order

- Which chunks should a peer 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
  - The scalability of the system is limited
- 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

Rarest Chunk First

- Which chunks should a peer 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

Preventing Free-Riding

- 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 prevents free riding
  - Allow the fastest peers to download from you first
  - Occasionally let some free-riders download

Preventing Free-Riding

- Peer has limited upload bandwidth
  - This bandwidth must be shared among multiple peers
- Prioritize upload bandwidth: tit for tat
  - Favor neighbors that are uploading at the highest rate
- Reward the top four neighbors
  - Measure download bit rates from each neighbor
  - Reciprocate by sending to the top four peers
  - Recompute and reallocate every 10 seconds
- Optimistic un-choking
  - Randomly try a new neighbor every 30 seconds
  - Give new neighbors a chance to become a better partner

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
  - Uploads at higher rates to higher-bandwidth peers
  - 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
- Lots of legal questions remain
- Further need for incentives

Summary

- Evolution of peer-to-peer
  - Central directory (Napster)
  - Query flooding (Gnutella)
  - Hierarchical overlay (Kazaa, modern Gnutella)
- BitTorrent
  - Focuses on parallel download
  - Prevents free-riding
- Next: Distributed Hash Tables

References

- Textbook sections 10.1-10.3, 10.5.3. **Required Reading.**

Acknowledgements

- These slides originally by Steve Ko, with light modification and used by permission by Ethan Blanton
- These slides contain material developed and copyrighted by Indranil Gupta (UIUC), Michael Freedman (Princeton), and Jennifer Rexford (Princeton).