Multimedia Communications
@CS.NCTU

Lecture 13: Overlay, P2P and CDN

Instructor: Kate Ching-Ju Lin (林靖茹)

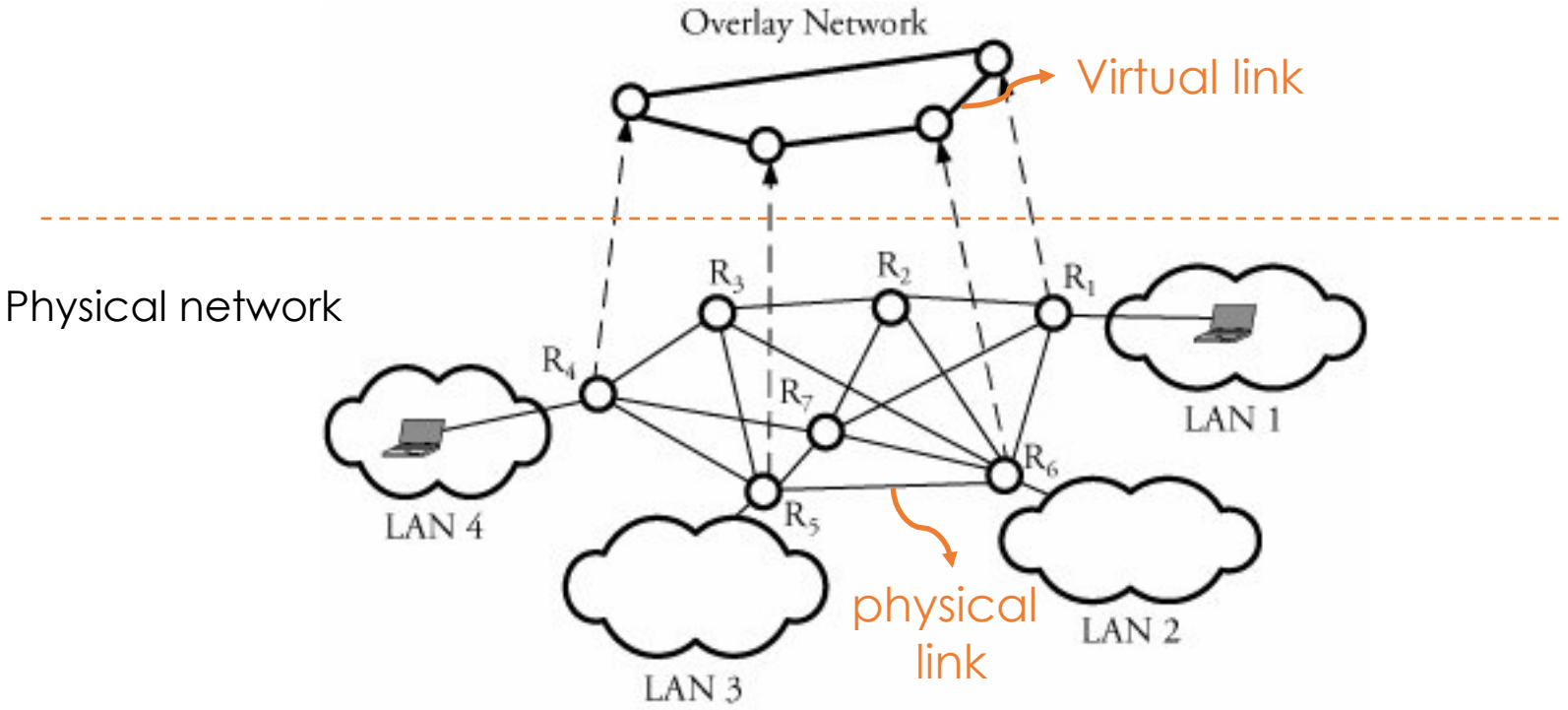
Ch. 2 "Computer Networking: A Top-Down Approach"
http://www-users.cselabs.umn.edu/classes/Spring-2016/csci5221/

Outline

- **Overlay networks**
- P2P System
  - Unstructured P2P
  - Structured P2P
  - Example: P2P streaming
- Content delivery network (CDN)

Overlay Networks

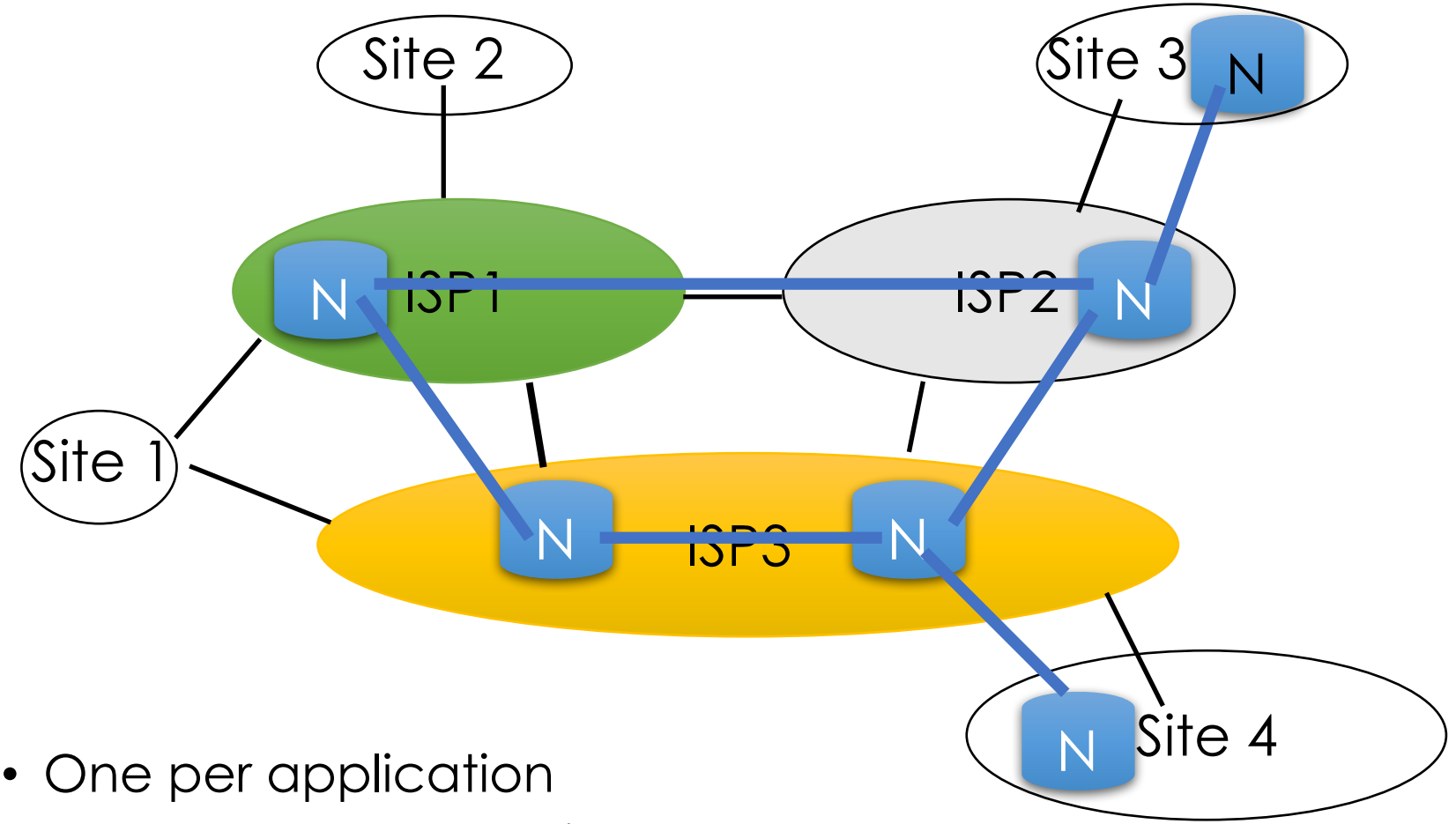
- A virtual network built on top of a physical network
- Nodes in the overlay connected by logical links
- Each logical link might go through many underlying physical links
- Example: Peer-to-peer, CDN



Overlay Networks

- A "logical" network built on top of a physical network
  - Overlay links are tunnels through the underlying network
- Many logical networks may coexist at once
  - Over the same underlying network
  - And providing its own particular service
- Nodes are often end hosts
  - Acting as intermediate nodes that forward traffic
  - Providing a service, such as access to files
- Who controls the nodes providing service?
  - The party providing the service (e.g., Akamai)
  - Distributed collection of end users (e.g., peer-to-peer)

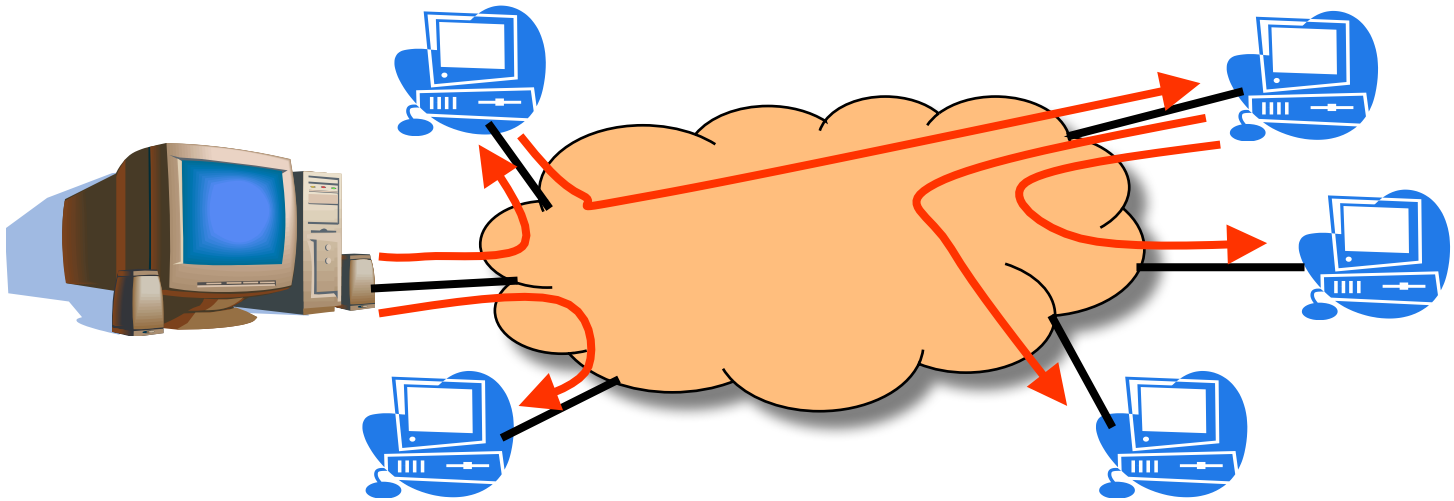
Application-level Overlays



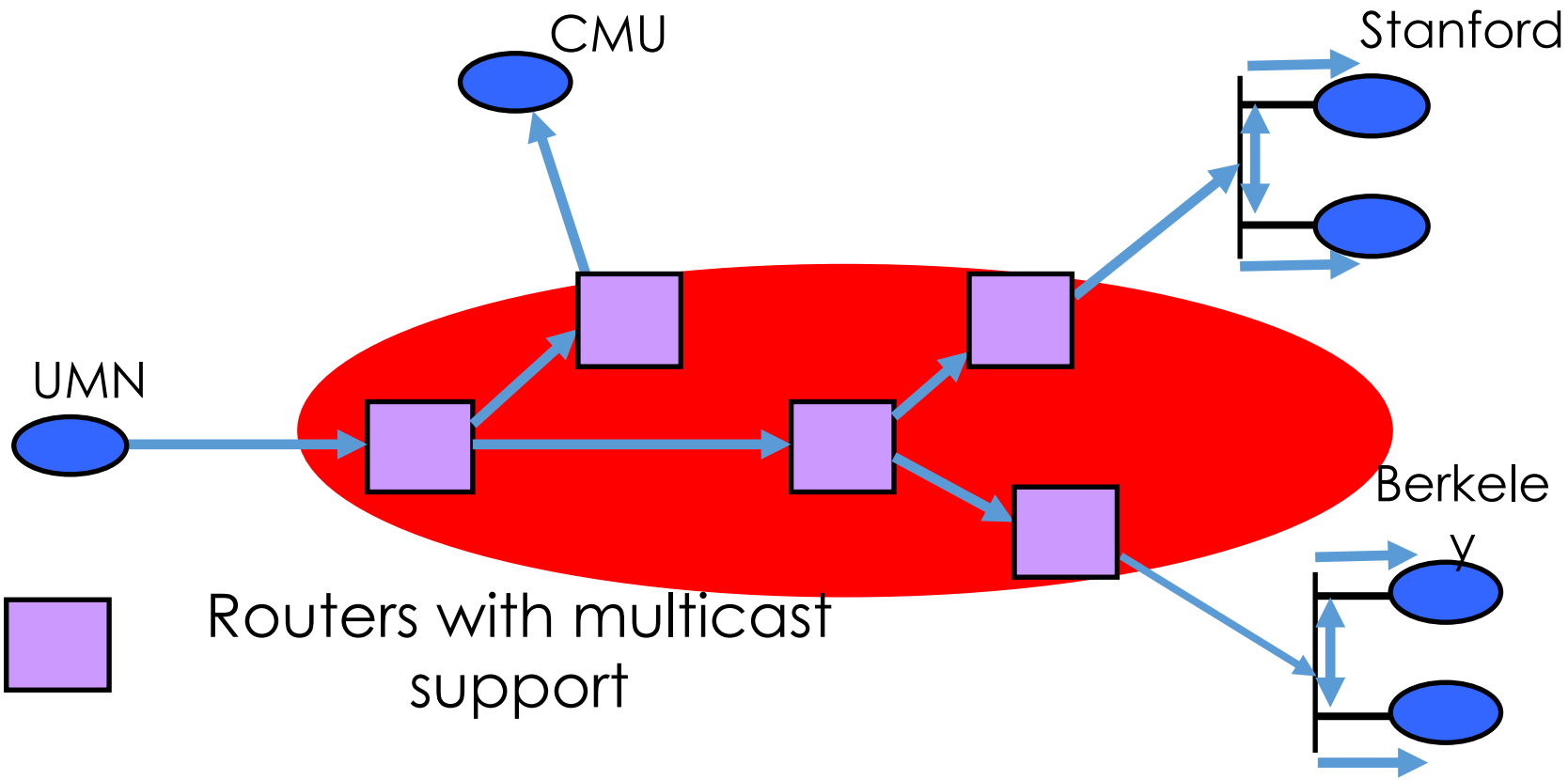
- One per application
- Nodes are decentralized
- Network operations/management may be centralized

Example: End-System Multicast

- IP multicast still is not widely deployed
  - Technical and business challenges
  - Should multicast be a *network*-layer service?
- Multicast tree of end hosts
  - Allow end hosts to form their own multicast tree
  - Hosts receiving the data help forward to others



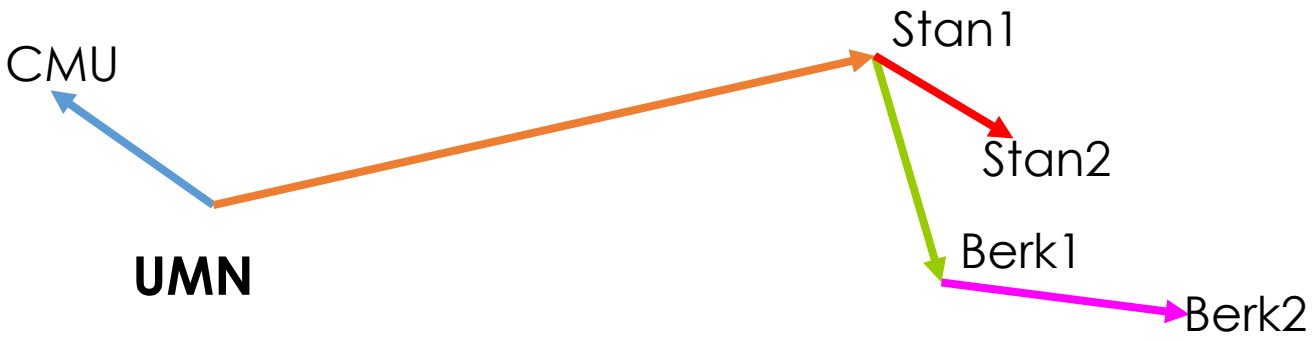
IP Multicast



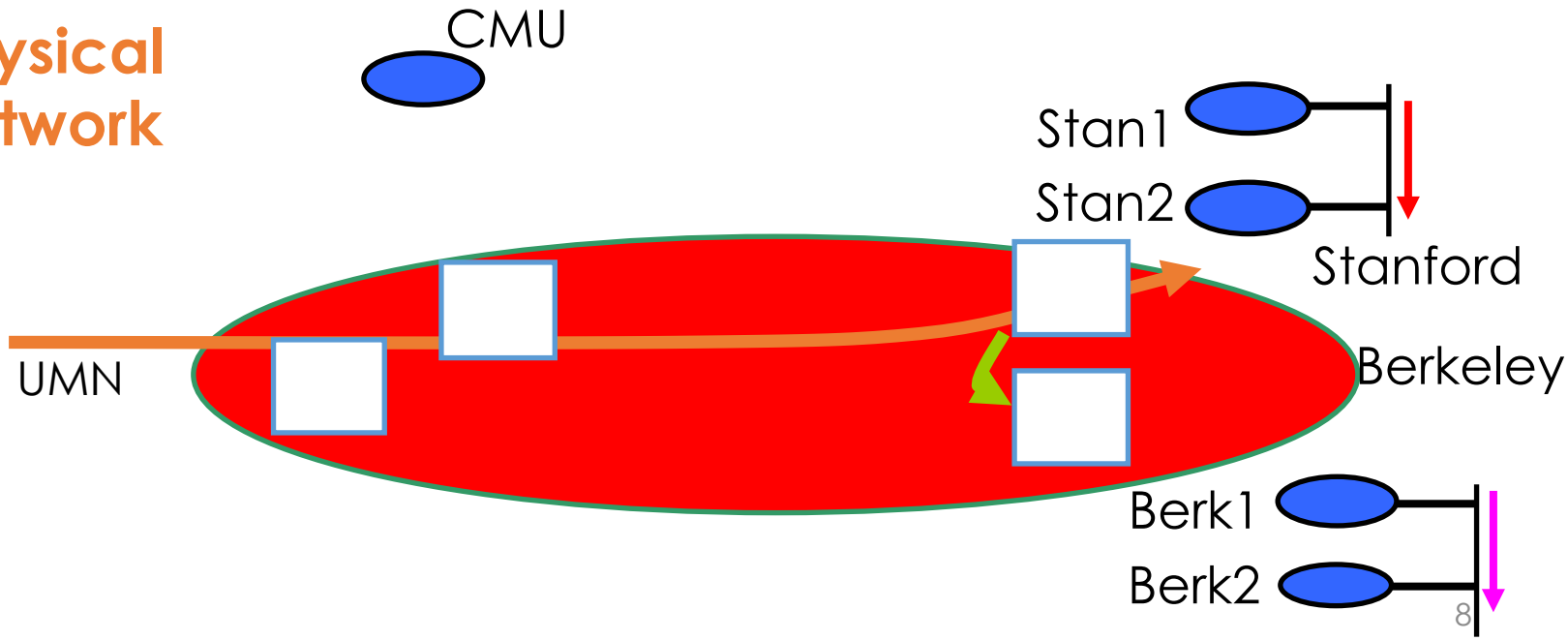
- No duplicate packets
- Highly efficient bandwidth usage

End System Multicast

Overlay multicast tree



Physical network



Outline

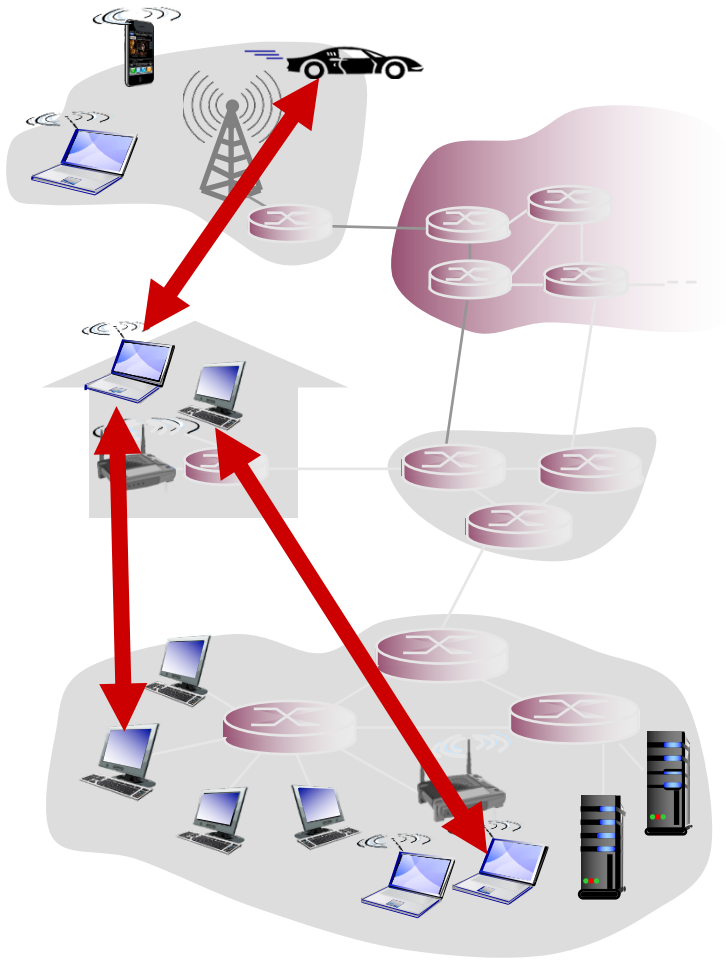
- Overlay networks
- **P2P System**
  - Unstructured P2P
  - Structured P2P
  - Example: P2P streaming
- Content delivery network (CDN)

Pure P2P Architecture

- *No* always-on server
- Arbitrary end systems directly communicate
- Peers are intermittently connected and change IP addresses

Examples:

- File distribution (BitTorrent)
- Streaming (PPstream)
- VoIP (Skype)

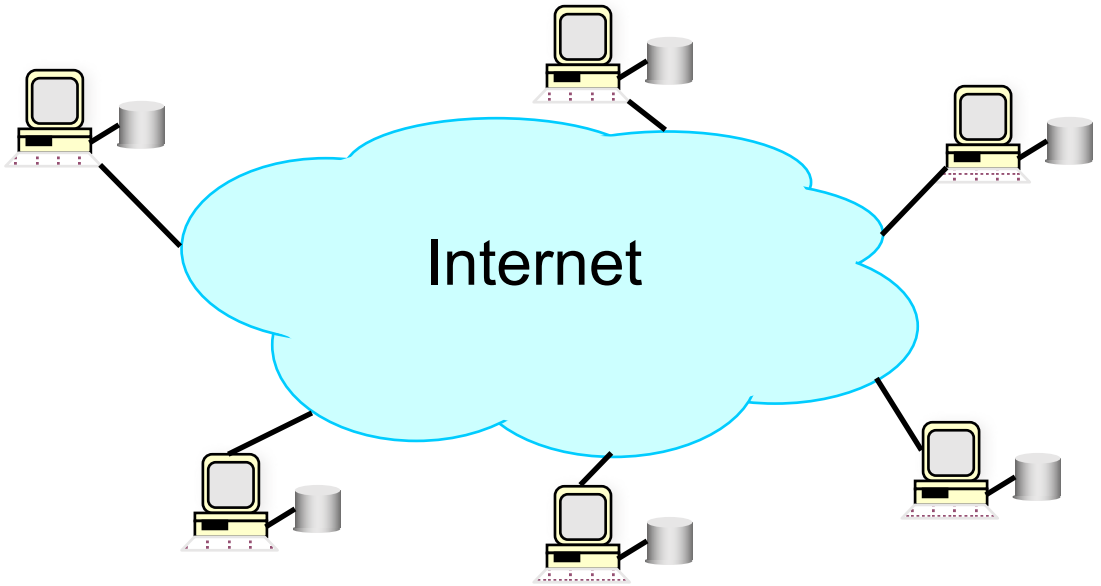


Peer-to-Peer Networks

- Two Types of P2P
  - **Unstructured Peer-to-Peer Networks**
    - Napster, Gnutella, KaZaA, BitTorrent, Skype, pplive, …
  - **Structured Peer-to-Peer Networks**
    - Distributed Hash Tables, DHT
    - Chord, Kadmelia, CAN, …
- What are the Key Challenges in P2P?
- Pros and Cons?

Peer-to-Peer Applications

- Very first killer application: Naptser
  - "free" music over the Internet
- Key idea: share the content, storage *and* bandwidth of individual (home) users

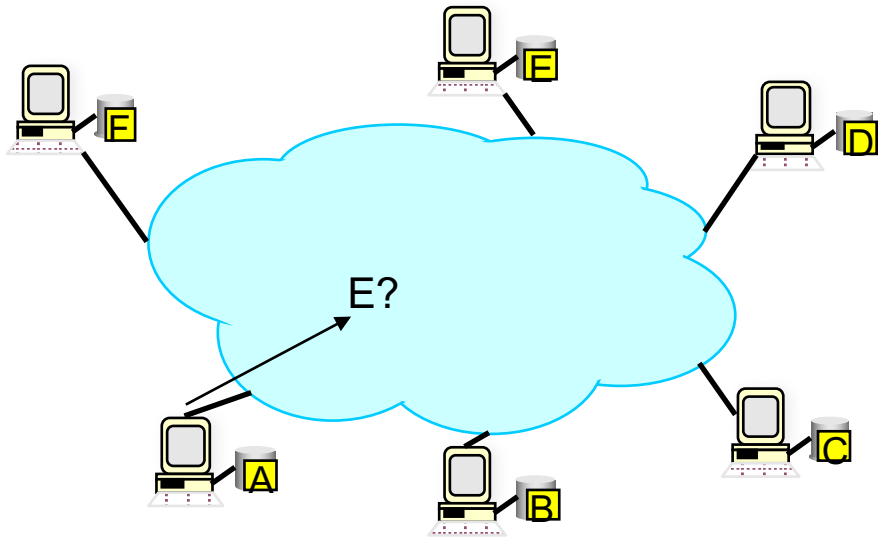


P2P (Application) Model

- Each user stores a subset of files (content)
- Each user has access (can download) files from all users in the system

Key Challenges in "pure" peer-to-peer model

- How to locate your peer & find what you want?
- Need some kind of "directory" or "look-up" service



- centralized
- distributed, using a hierarchal structure
- distributed, using a flat structure
- distributed, with no structure ("flooding" based)
- distributed, using a "hybrid" structured/unstructured approach

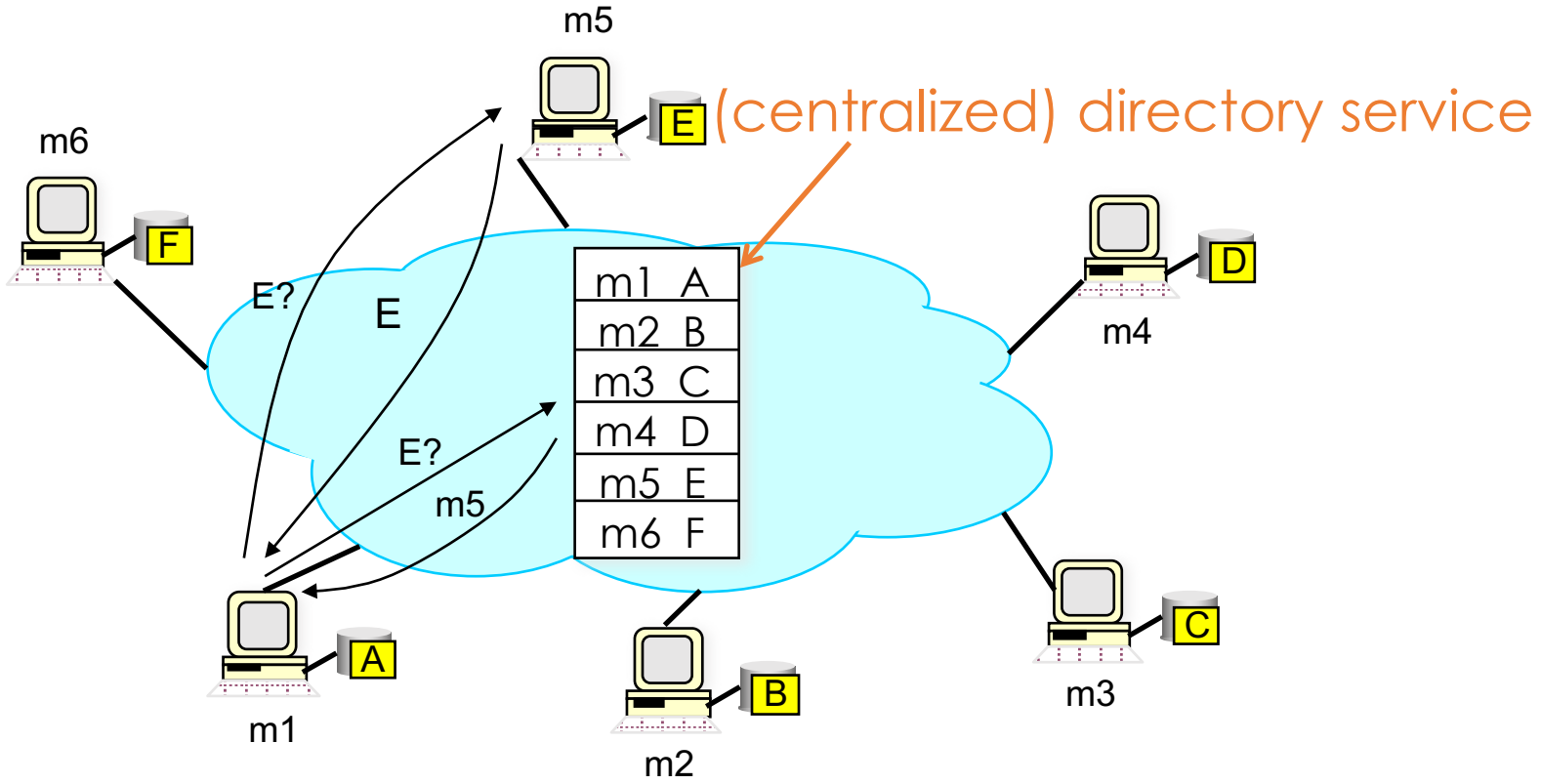
Other Challenges

- **Technical**
  - Scale: up to hundred of thousands or millions of machines
  - Dynamics: machines can come and go any time
- **Social, economic and legal**
  - Incentive Issues: free-loader problem
  - Vast majority of users are free-riders
  - Most share no files and answer no queries
  - A few individuals contributing to the "public good"
  - Copyrighted content and piracy
  - Trust & security issues

Unstructured P2P Applications

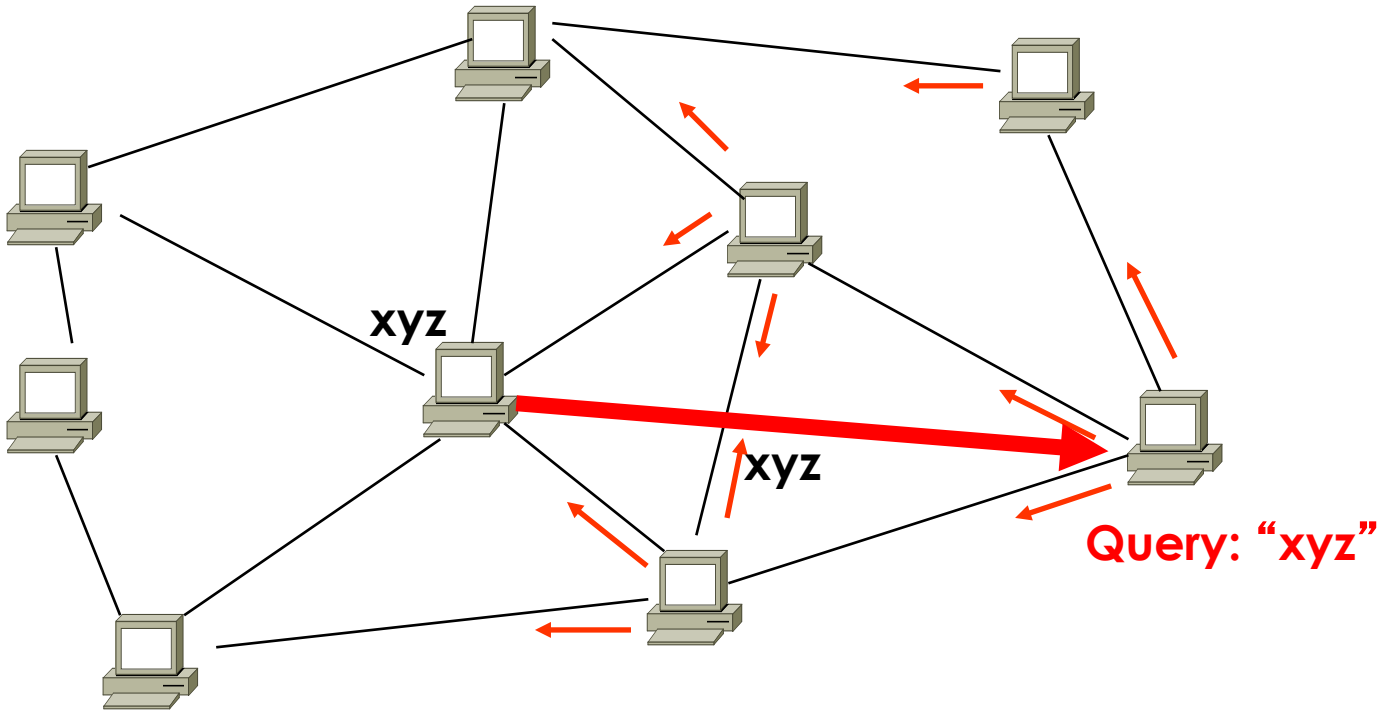
- Napster
  - a *centralized* directory service
  - peers directly download from other peers
- Gnutella
  - fully distributed directory service
  - discover & maintain neighbors, *ad hoc* topology
  - flood & forward queries to neighbors (with bounded hops)
- KaZaA
  - exploit heterogeneity, certain peers as "super nodes"
  - two-tier hierarchy: when join, contact a super-node
  - smart query flooding
  - peer may fetch data from multiple peers at once
  - used by Skype (for directory service)

Napster Architecture



Gnutella

- Ad-hoc topology
- Queries are flooded for bounded number of hops
- No guarantees on recall



BitTorrent & Video Distribution

- Designed for large file (e.g., video) downloads
  - esp. for popular content, e.g. flash crowds
- Focused on efficient *fetching*, not search
  - Distribute same file to many peers
  - Single publisher, many downloaders
- 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
- Also includes mechanisms for preventing "free loading"

BitTorrent Components

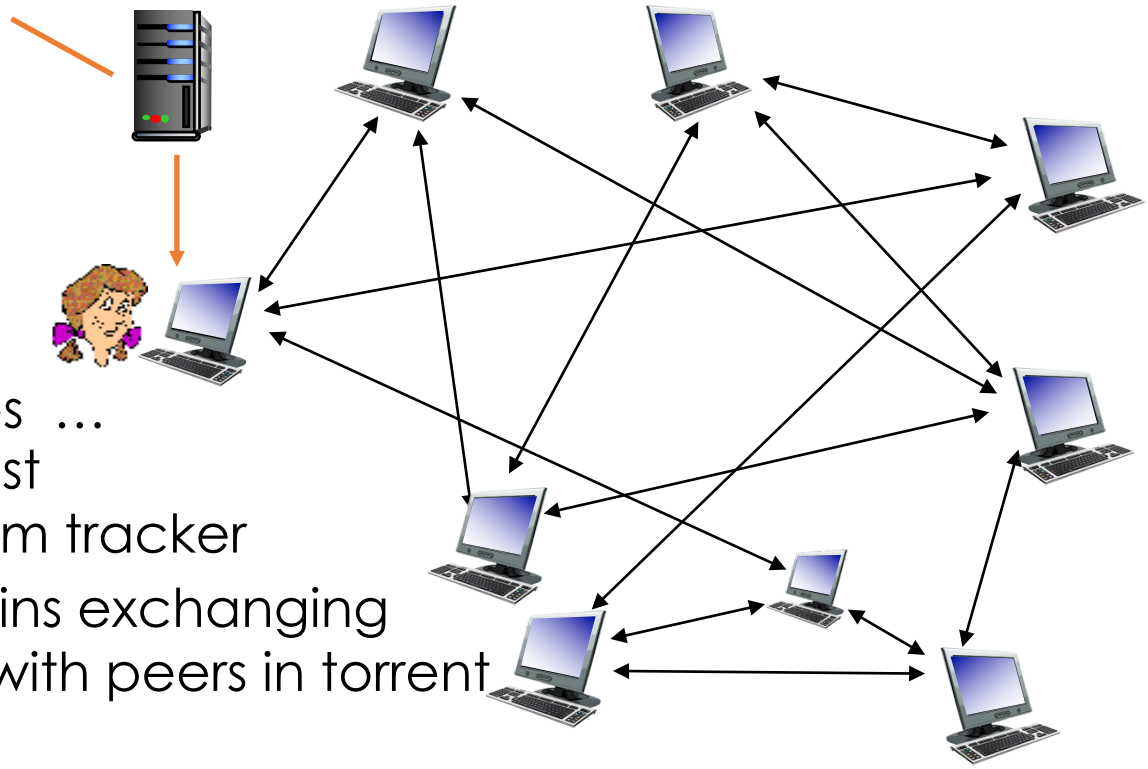
- Seed
  - Peer with entire file
  - Fragmented in pieces
- Leacher
  - Peer with an incomplete copy of the file
- Torrent file
  - Passive component
  - Store summaries of the pieces to allow peers to verify their integrity
- Tracker
  - Allows peers to find each other
  - Returns a list of random peers

P2P File Distribution: BitTorrent

- File divided into 256Kb chunks
- Peers in torrent send/receive file chunks

tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file



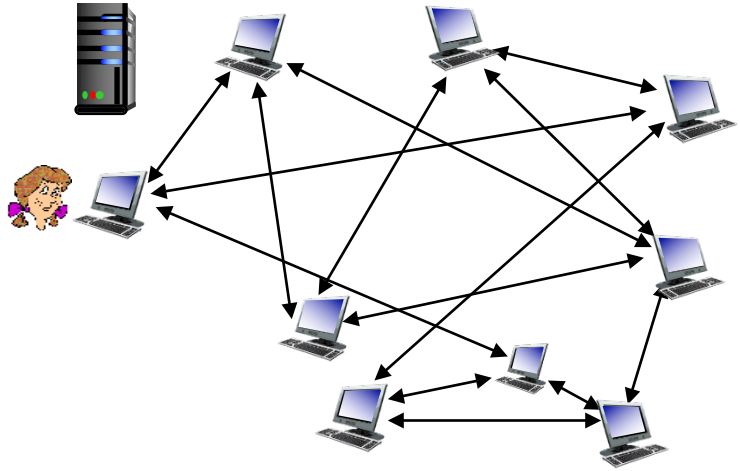
Alice arrives …

… obtains list of peers from tracker

… and begins exchanging file chunks with peers in torrent

P2P File Distribution: BitTorrent

- Peer joining torrent:
  - has no chunks, but will accumulate them over time from other peers
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")



- While downloading, peer uploads chunks to other peers
- Peer may change peers with whom it exchanges chunks
- Churn: peers may come and go
- Once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

BT: requesting, sending file chunks

requesting chunks:

- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, rarest first

sending chunks: tit-for-tat

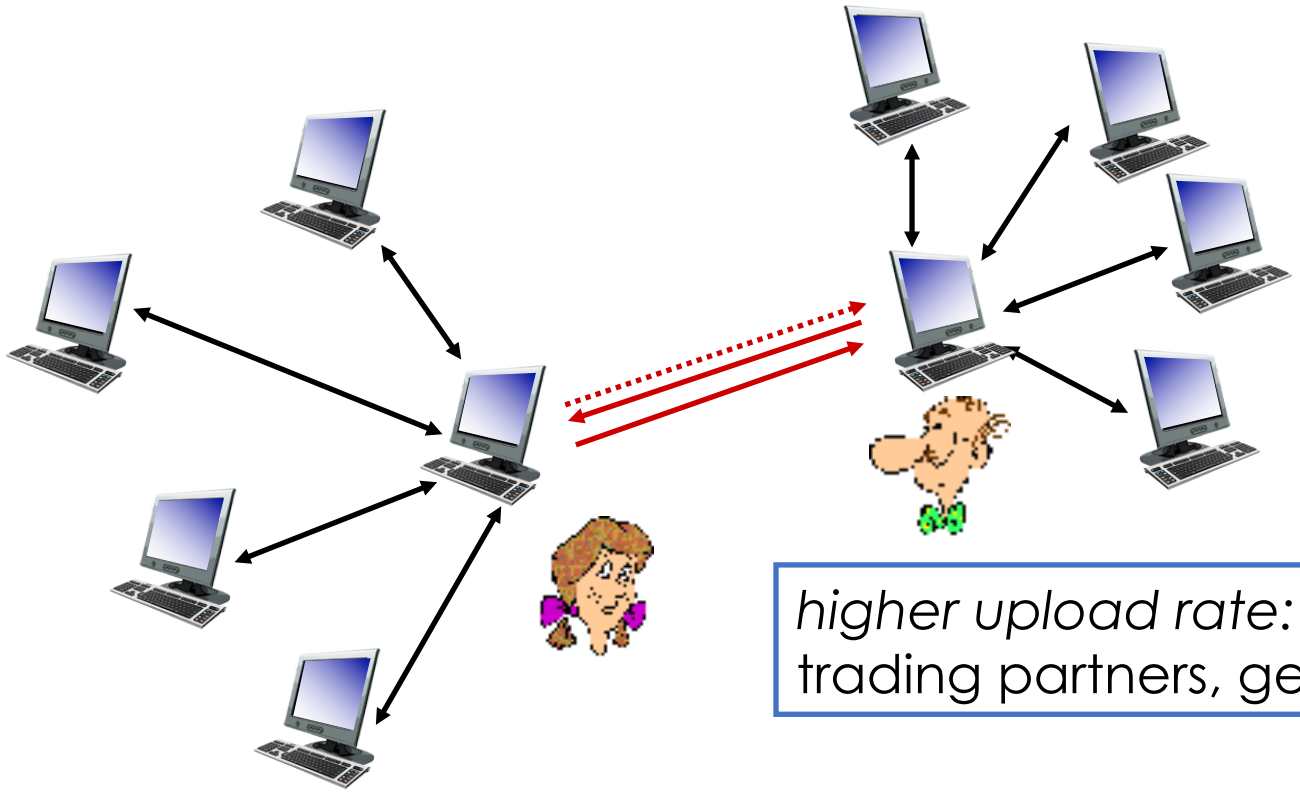
- Alice sends chunks to those peers currently sending her chunks at highest rate
  - other peers do not receive chunks from Alice
  - re-evaluate top 4 every10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - "optimistically unchoke" this peer
  - newly chosen peer may join top 4

BT: Tit-for-Tat

(1) Alice "optimistically unchokes" Bob

(2) Alice becomes one of Bob's top-four providers; Bob reciprocates

(3) Bob becomes one of Alice's top-four providers



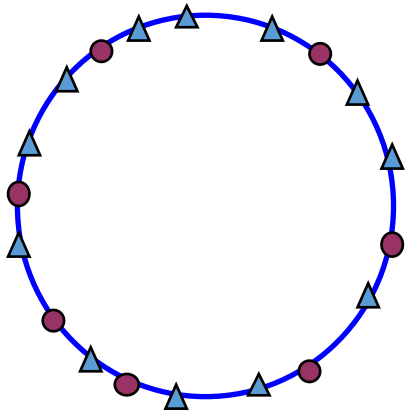
higher upload rate: find better trading partners, get file faster !

Structured P2P Networks

- Introduce a structured logical topology
- Abstraction: a distributed hash table data structure
  - put (key, object); get (key)
  - key: identifier of an object
  - *object* can be anything: a data item, a node (host), a document/file, pointer to a file, …
- Design Goals: guarantee on recall
  - i.e., ensure that an item (file) identified is always found
  - Also scale to hundreds of thousands of (or more) nodes
  - handle rapid join/leave and failure of nodes
- Proposals
  - Chord, CAN, Kademlia, Pastry, Tapestry, etc

Key Ideas (Concepts & Features)

- Keys and node IDs map to the same "flat" ID space
  - node IDs are thus also (special) keys!
- Management (organization, storage, lookup, etc) of keys using consistent hashing
  - distributed, maintained by all nodes in the network
- (Logical) distance defined on the ID space: structured!
  - different DHTs use different distances/structures



Key Ideas (Concepts & Features)

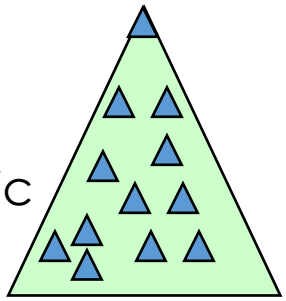
- Look-up/Routing Tables ("finger table" in Chord)
  - each node typically maintains O(log n) routing entries
  - organizing using structured ID space: more information about nodes closer-by; less about nodes farther away
- Bootstrap, handling node joins/leaves or failures
  - when node joins: needs to know at least one node in the system
- Robustness/resiliency through redundancy

Lookup Service using DHT

DHT : distributed hash table

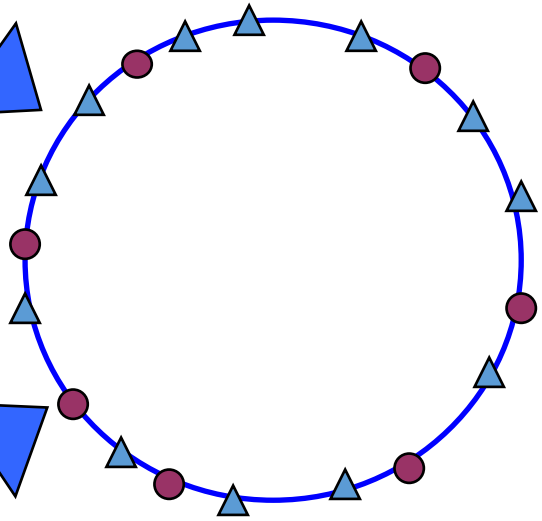
object (name) space

often with its own semantic structure, e.g., domain names



flat ("semantic-free"), circular (a ring) id (identifier) space

m bits (e.g., m=160), M = 2^m id's



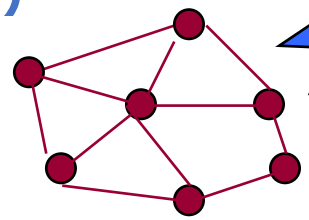
map to ID space via hashing

H(obj_name) = id_k

H(node_addr) = id_n

node (name/address) space

with its own physical topological structure



Why hashing?

DHT-based Schemes

- Chord
- CAN (content addressable network)
- Pastry
- Tapestry
- Viceroy
- Leopard (locality-aware DHT)
- ……

(e.g., look up & read “DHT” in wikipedia)

Outline

- Overlay networks
- P2P System
  - Unstructured P2P
  - Structured P2P
  - Example: P2P streaming
- **Content delivery network (CDN)**

Demands of Video Streaming

- Video traffic: major consumer of Internet bandwidth
  - Netflix, YouTube: 37%, 16% of downstream residential ISP traffic
  - ~1B YouTube users, ~75M Netflix users
- Challenge: scale - how to reach ~1B users?
  - single mega-video server won't work (why?)
- Challenge: heterogeneity
  - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)
- **Solution:** distributed, application-level infrastructure

  → **Content Distributed Networks (CDN)**

Why CDNs?

- **Challenge:** how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?
- **Option 1:** single, large "mega-server"
  - Reliability: single point of failure
  - Not enough bandwidth: point of network congestion
  - Far from users: long path to distant clients
  - multiple copies of video sent over outgoing link

…. quite simply: this solution *doesn't scale*

Why CDNs?

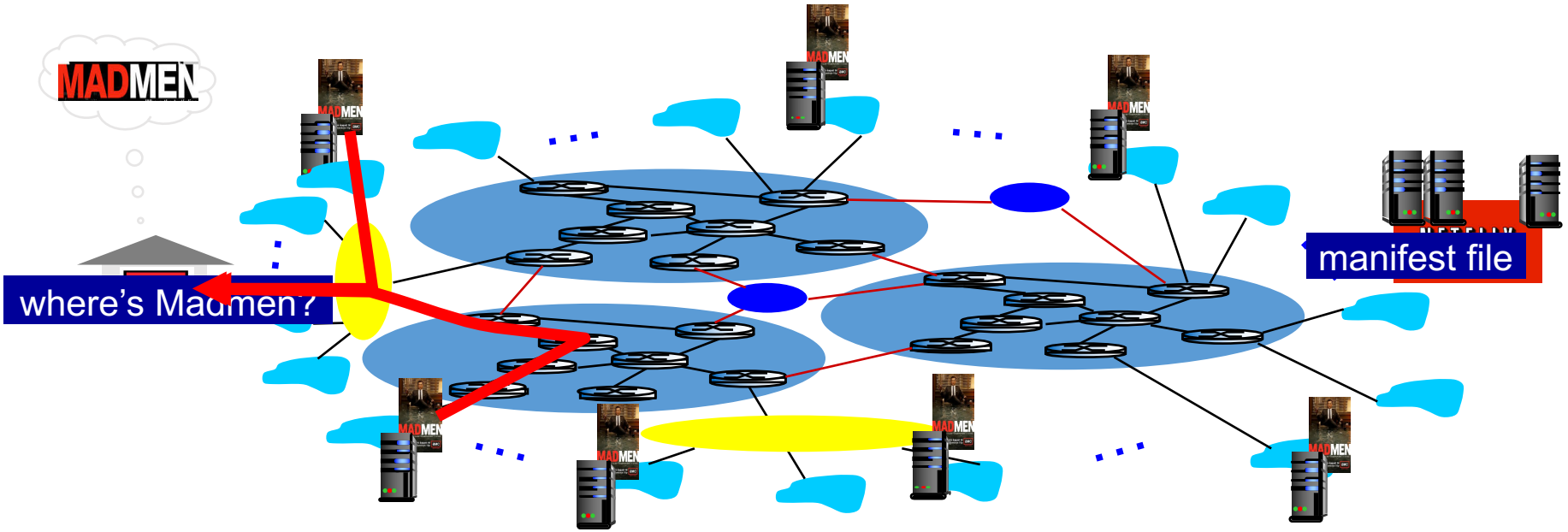
- **Challenge:** how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?
- **Option 2:** store/serve multiple copies of videos at multiple geographically distributed sites (CDN)
  - enter deep: push CDN servers deep into many access networks
    - close to users
    - used by Akamai, 1700 locations
  - bring home: smaller number (10's) of larger clusters in POPs near (but not within) access networks
    - used by Limelight

Content Distribution Networks

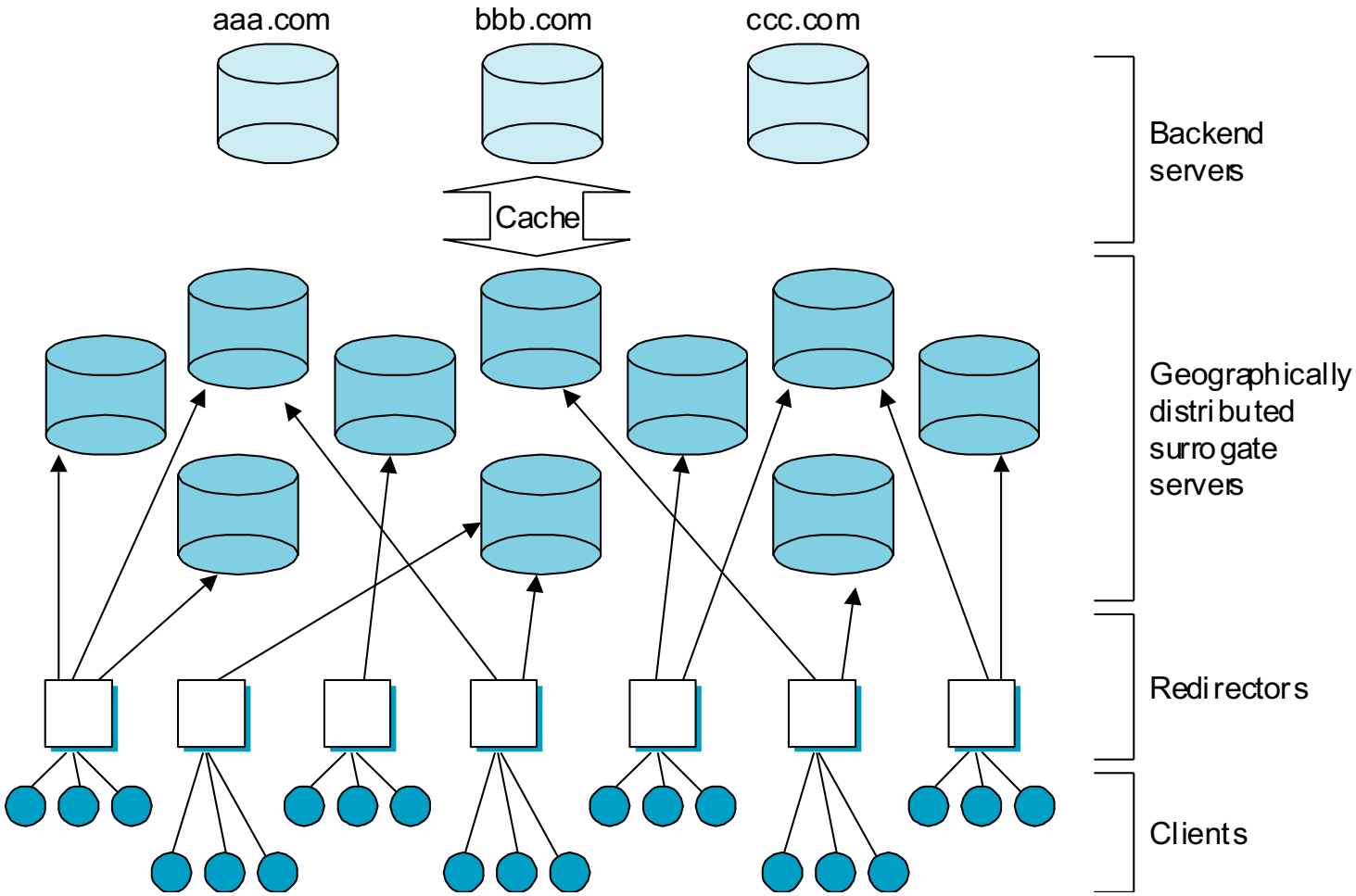
- **CDN:** an application overlay (e.g., Akamai)
- Design Space
  - Caching (data-driven, passive)
    - explicit
    - transparent (hijacking connections)
  - Replication (pro-active)
    - server farms
    - geographically dispersed (CDN)
- Three Main CDN Providers (in North America, Europe):
  - Akamai, Limelight, Level 3 CDN

Key Idea of CDN

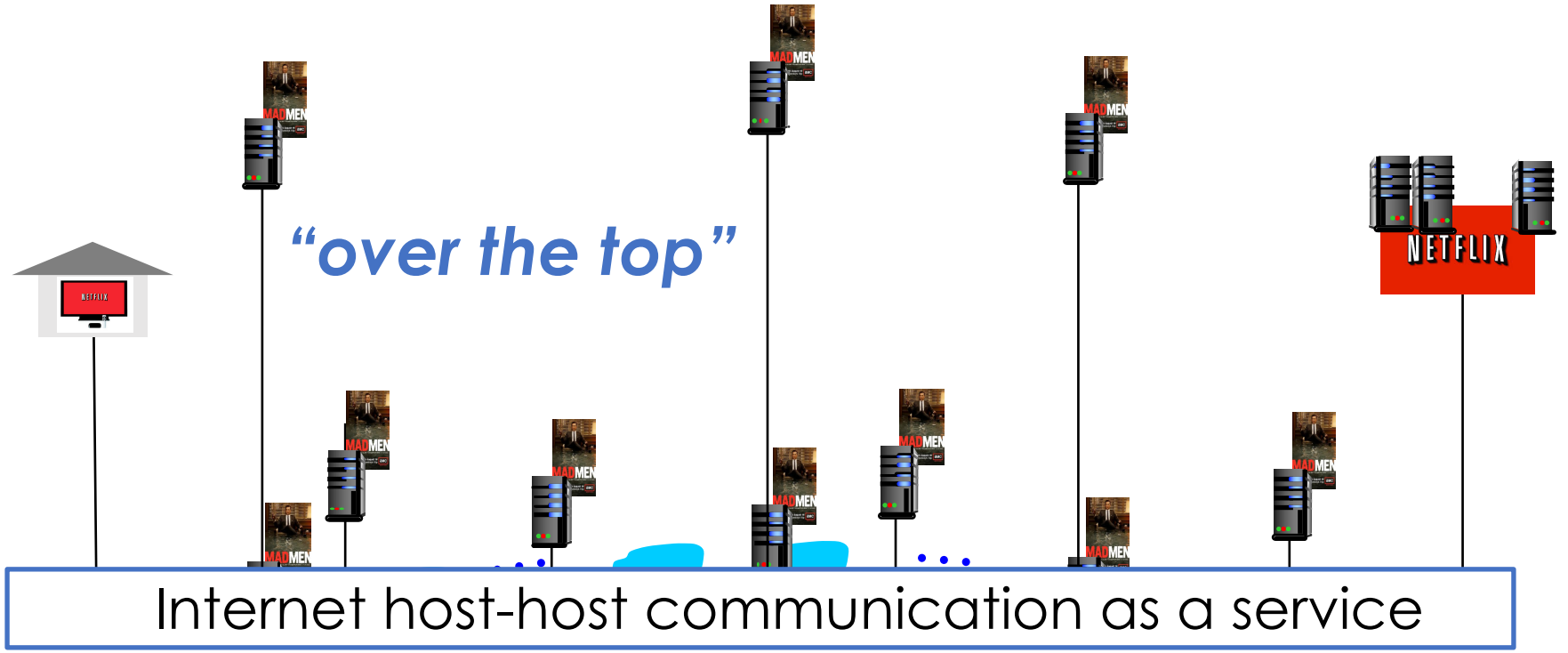
- CDN: stores copies of content at CDN nodes
  - e.g. Netflix stores copies of MadMen
- Subscriber requests content from CDN
  - directed to nearby copy, retrieves content
  - may choose different copy if network path congested



CDN: Schematic Illustration



Framework of CDN

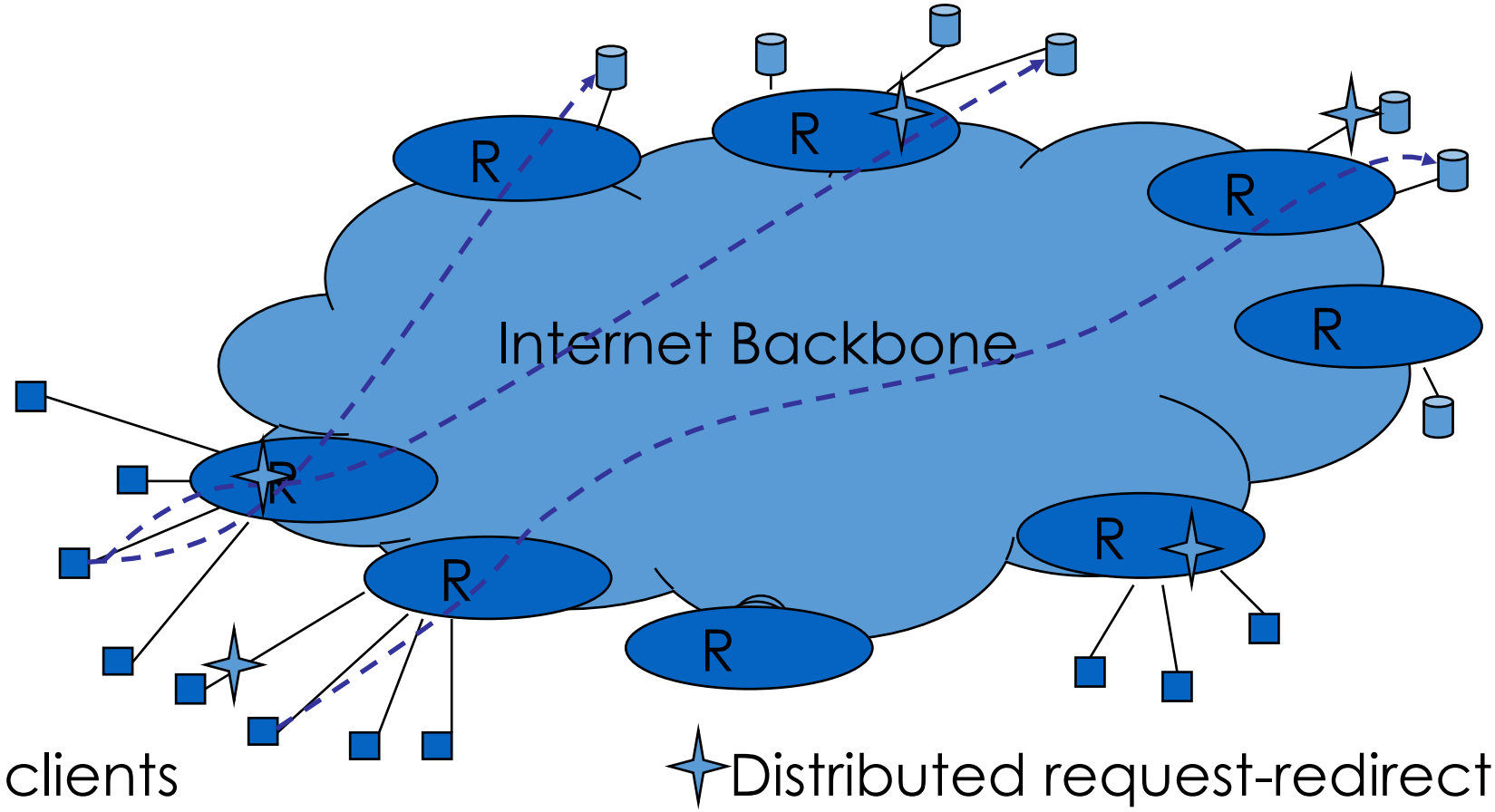


OTT challenges: coping with a congested Internet

- from which CDN node to retrieve content?
- viewer behavior in presence of congestion?
- what content to place in which CDN node?

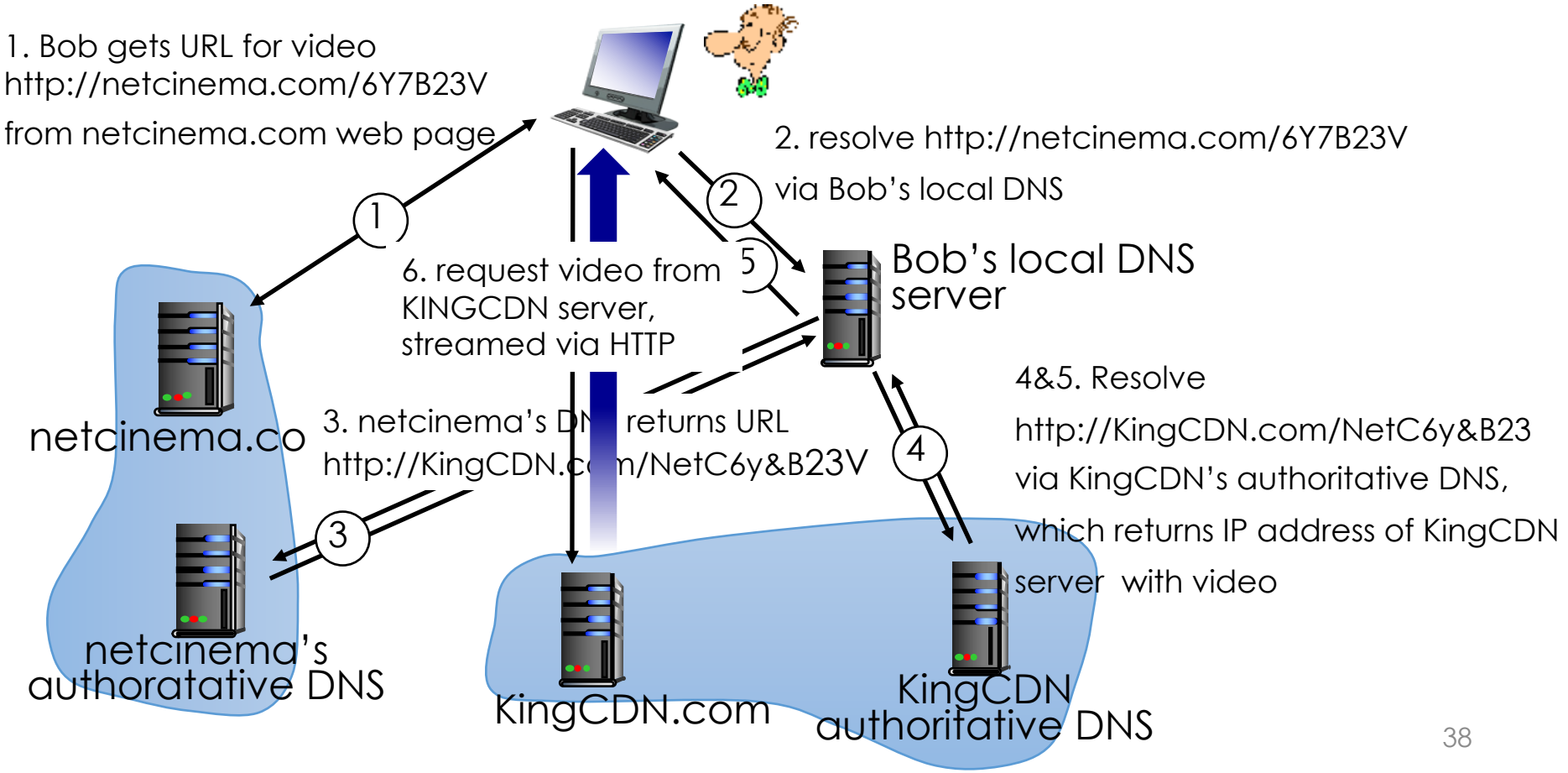
Redirection Overlay

Geographically distributed server clusters

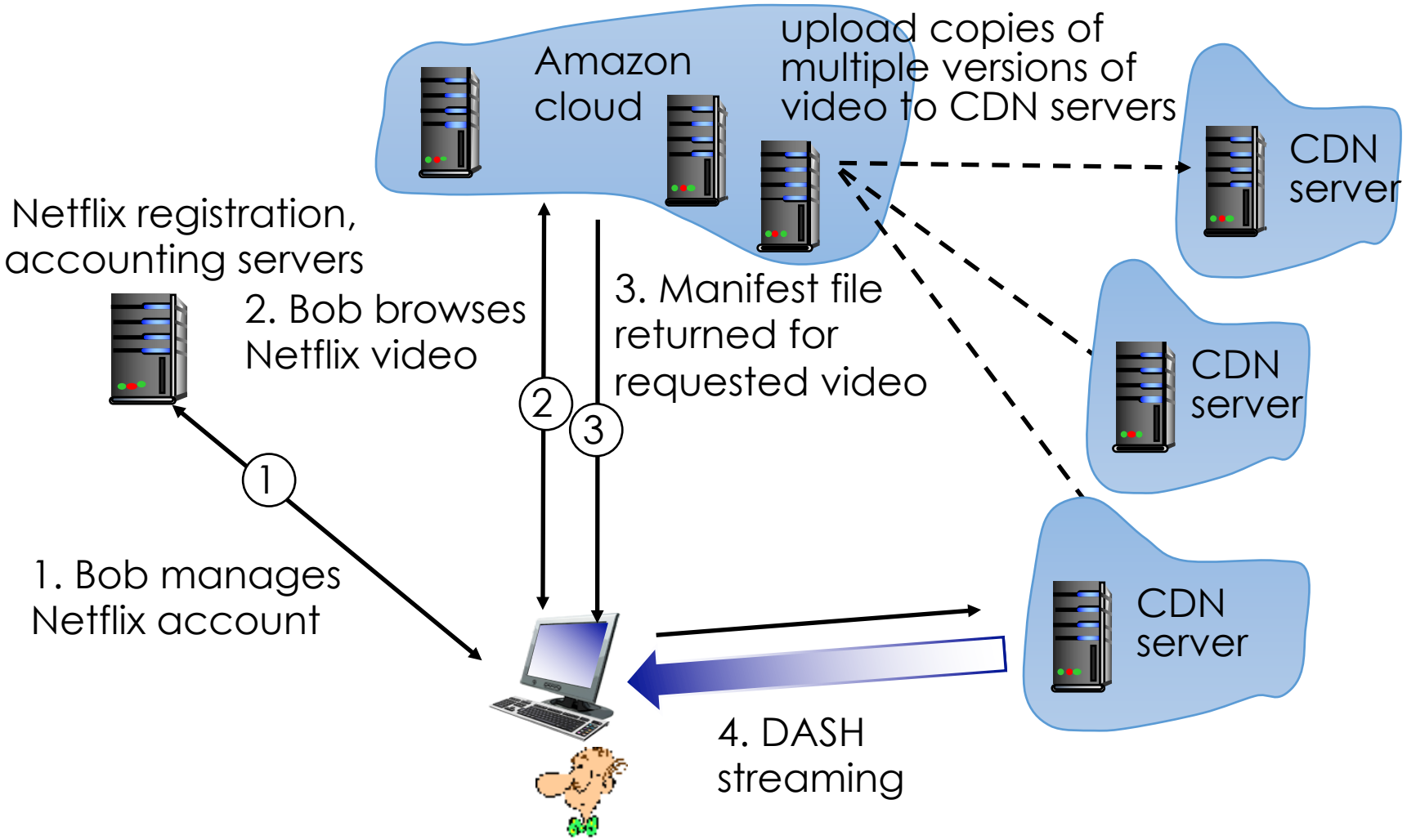


CDN Content Access

- Bob (client) requests video http://netcinema.com/6Y7B23V
  - video stored in CDN at http://KingCDN.com/NetC6y&B23V



Case Study: Netflix



Redirection Techniques

- URL Rewriting: embedded links
- HTTP redirection: requires an extra round trip
- DNS (anycast): one name maps onto many addresses
  - esp. useful for finding nearby servers & (coarse-grained) locality-based load balancing
  - Question: how to figure out geo-location of users (at DNS query time)?
  - works for both servers and reverse proxies
- Router: IP Anycast
  - announce (via BGP) the same IP address prefixes at multiple locations
- "Layer 4/7" (application) switches
  - one address, select a server (reverse proxy)
  - content-based routing (near client)

CDN Cluster Selection Strategy

- **Challenge:** how does CDN DNS select "good" CDN node to stream to client
  - pick CDN node geographically closest to client
  - pick CDN node with shortest delay (or min # hops) to client (CDN nodes periodically ping access ISPs, reporting results to CDN DNS)

- **Alternative:** let client decide - give client a list of several CDN servers
  - client pings servers, picks "best"

Akamai CDN: quickie

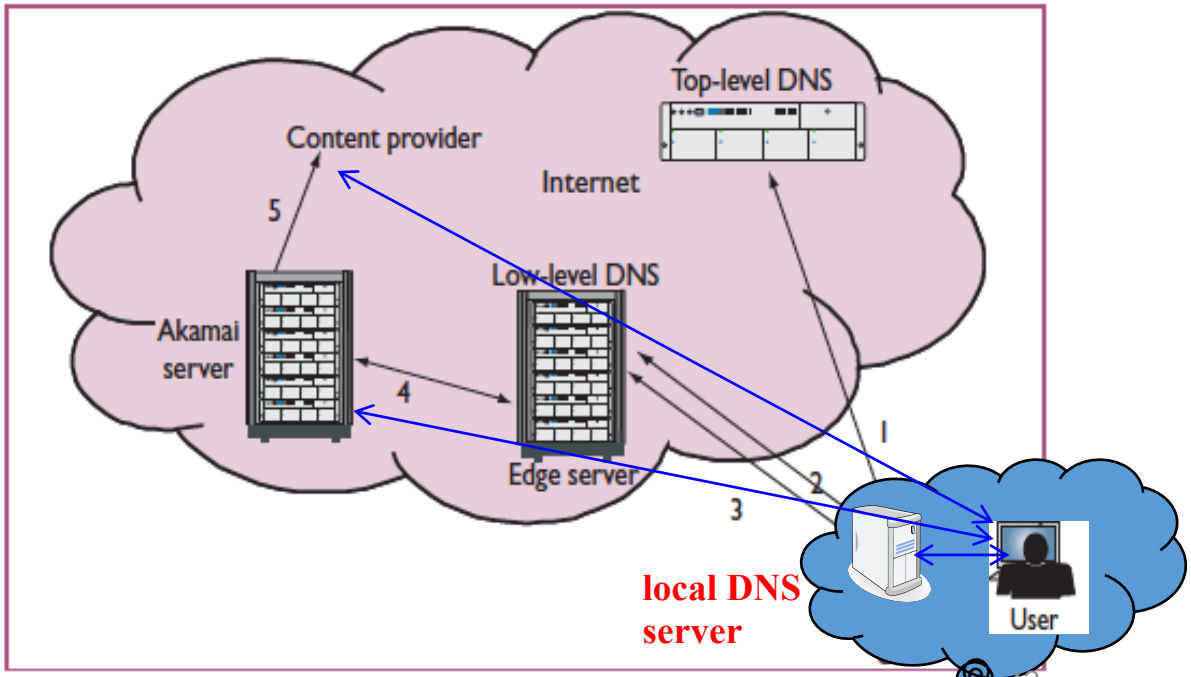
- pioneered creation of CDNs circa 2000
- now: 61,000 servers in 1,000 networks in 70 countries
- delivers est 15-20% of all Internet traffic
- runs its own DNS service (alternative to public root, TLD, hierarchy)
- hundreds of billions of Internet interactions daily
- more shortly….

Akamai CDN Overview

- More than 10s of thousands servers in more 1000s networks globally
- Support a variety of services
  - DNS resolution, web content delivery, web search, large software update, media content distribution (music, video, etc), …

Basic operations

- User, local DNS & Akamai DNS
- User, Akamai servers & content providers



Akamai CDN Architecture