The Web
+
Random Access

CS168, Fall 2014

Sylvia Ratnasamy

http://inst.eecs.berkeley.edu/~cs168/

Material thanks to Ion Stoica, Scott Shenker, Jennifer Rexford, Nick McKeown, and many other colleagues

Quick Poll

- CS168 is getting too popular. We need advice.
- Two identical versions: Fall and Spring
- One regular 168 (Fall) and one “honors” (Spring)
- How many of you would have taken CS168H?
- How many of you have taken CS 162?
- How many of you would take an “advanced and applied networking” course (not 268, but a 194)?
- How many of you are planning to vote?

The Web

The Web – Precursor

Ted Nelson

- **1967**, Ted Nelson, Xanadu:
  - A world-wide publishing network that would allow information to be stored not as separate files but as connected literature
  - Owners of documents would be automatically paid via electronic means for the virtual copying of their documents
- Coined the term “Hypertext”

The Web – History

Tim Berners-Lee

- World Wide Web (WWW): a distributed database of “pages” linked through Hypertext Transport Protocol (HTTP)
  - First HTTP implementation - 1990
    - Tim Berners-Lee at CERN
  - HTTP/0.9 – 1991
    - Simple GET command for the Web
  - HTTP/1.0 –1992
    - Client/Server information, simple caching
  - HTTP/1.1 - 1996

On inventing a “killer app”

HTML is precisely what we were trying to PREVENT— ever-breaking links, links going outward only, quotes you can't follow to their origins, no version management, no rights management.

– Ted Nelson

Web Components

- Infrastructure:
  - Clients
  - Servers
- Content:
  - URL: naming content
  - HTML: formatting content
- Protocol for exchanging information: HTTP

Uniform Record Locator (URL)

```
protocol://host-name[:port]/directory-path/resource
```

- Extend the idea of hierarchical hostnames to include anything in a file system
  - http://www.cs.berkeley.edu/~sylvia/cs268/lecture1.ppt
- Extend to program executions as well…
  - http://us.f413.mail.yahoo.com/ym/ShowLetter?box=%40B%40Bulk&MsgId=2604_1744106_29699_1123_1261_0_28917_3552_1289957100&Search=&Nhead=f&YY=31454&order=down&sort=date&pos=0&view=a&head=b
  - Server side processing can be incorporated in the name

Uniform Record Locator (URL)

```
protocol://host-name[:port]/directory-path/resource
```

- *protocol*: http, ftp, https, smtp, rtsp, *etc*.
- *hostname*: DNS name, IP address
- *port:* defaults to protocol's standard port; *e.g.* http: 80 https: 443
- *directory path*: hierarchical, reflecting file system
- *resource*: Identifies the desired resource

Web and DNS

- URLs use hostnames
- Thus, content names are tied to specific hosts
- Why is this a problem?
- Makes persistence of names problematic…

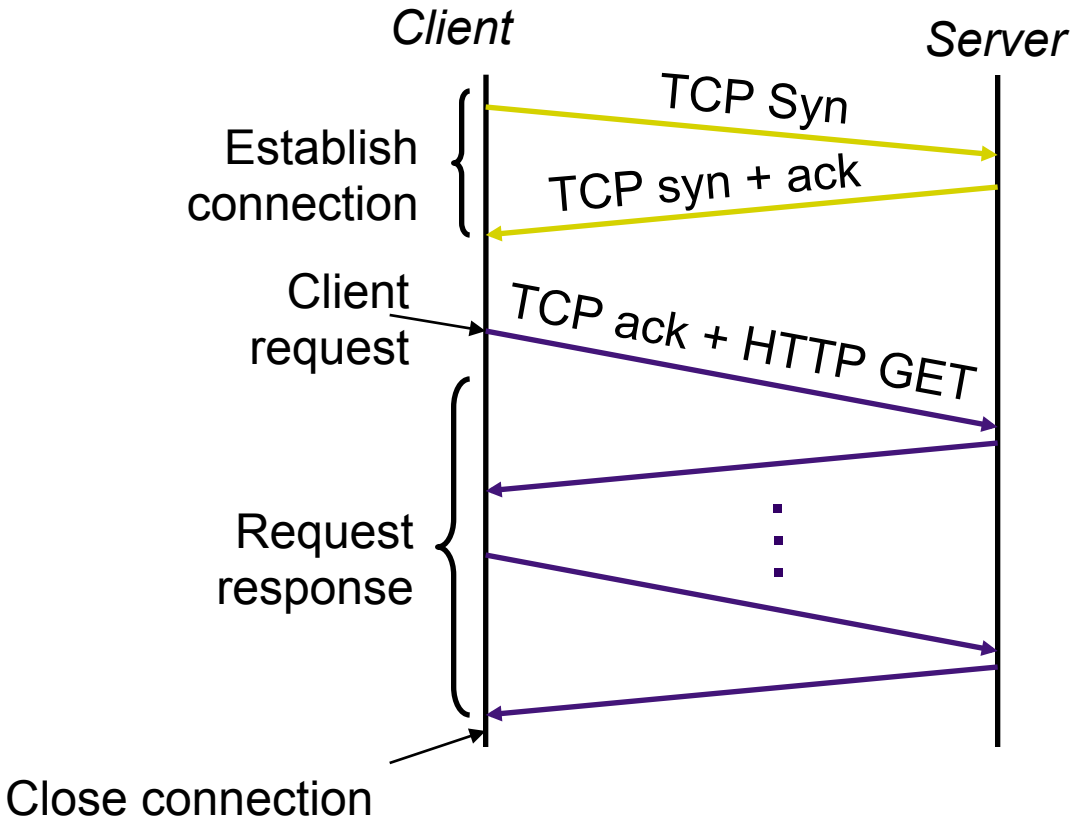
Why not name content directly?

- How do you know where to send the request?
- How do you scale?
- How do you trust the response?
  - Requesting host
  - Network
- How would you design it?

Hyper Text Transfer Protocol (HTTP)

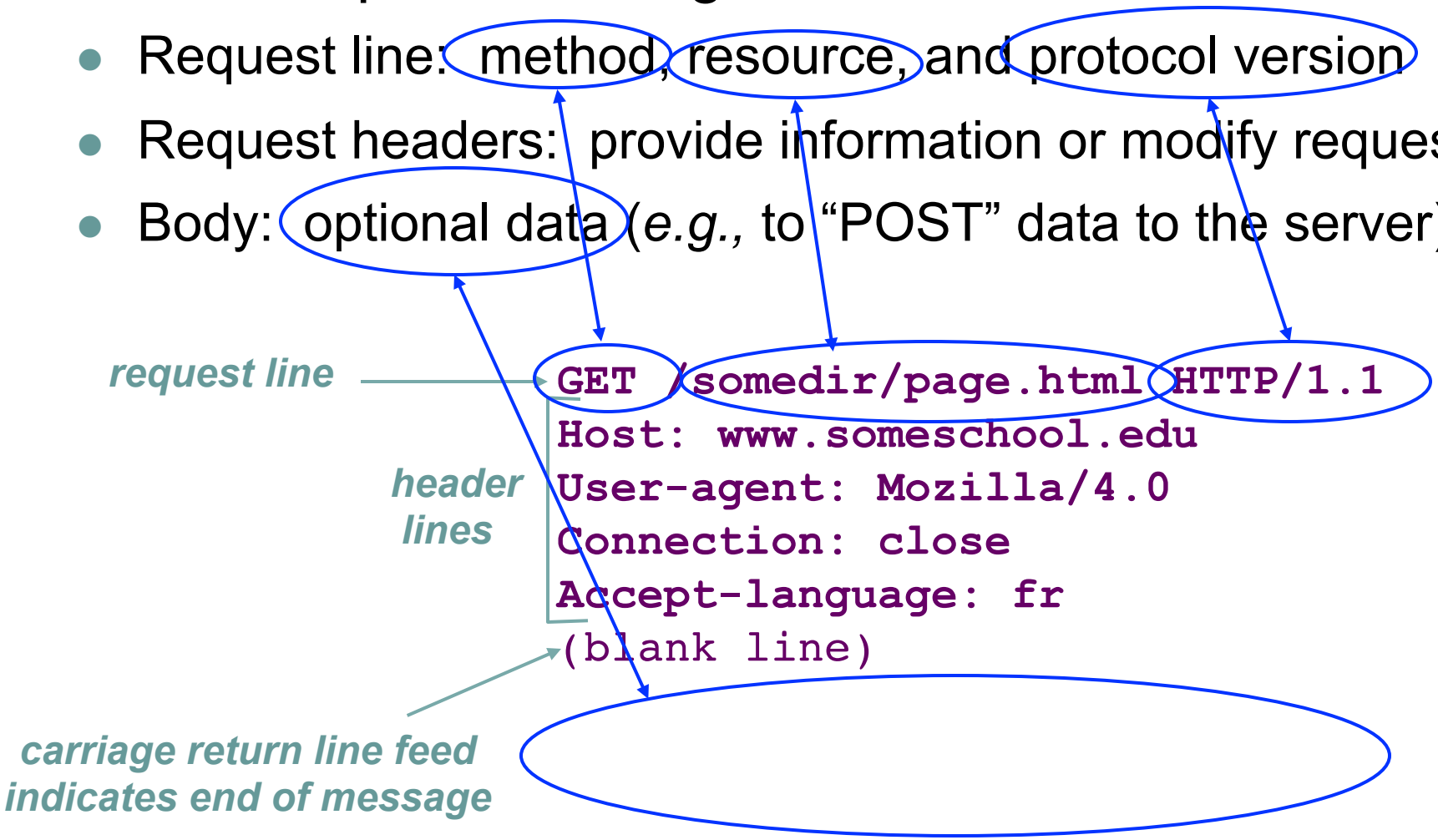
- Client-server architecture
  - server is “always on” and “well known”
  - clients initiate contact to server
- Synchronous request/reply protocol
  - Runs over TCP, Port 80
- Stateless
- ASCII format

Steps in HTTP Request/Response



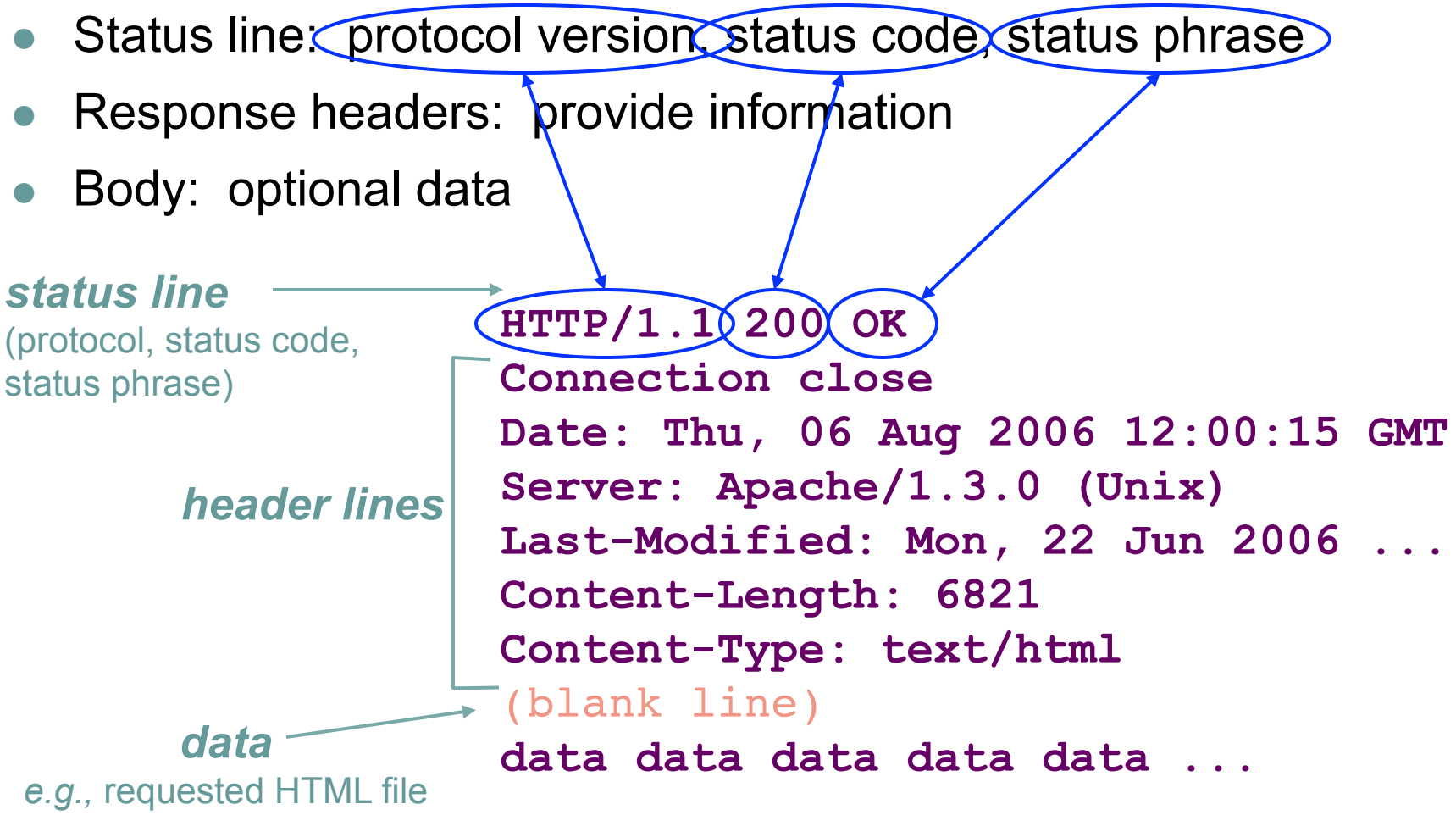
Client-to-Server Communication

- HTTP Request Message
  - Request line: method, resource, and protocol version
  - Request headers: provide information or modify request
  - Body: optional data (*e.g.*, to "POST" data to the server)



Server-to-Client Communication

- HTTP Response Message
  - Status line: protocol version, status code, status phrase
  - Response headers: provide information
  - Body: optional data



HTTP is *Stateless*

- Each request-response treated independently
  - Servers *not* required to retain state
- **Good**: Improves scalability on the server-side
  - Failure handling is easier
  - Can handle higher rate of requests
  - Order of requests doesn't matter
- **Bad**: Some applications need persistent state
  - Need to uniquely identify user or store temporary info
  - *e.g.,* Shopping cart, user profiles, usage tracking, …

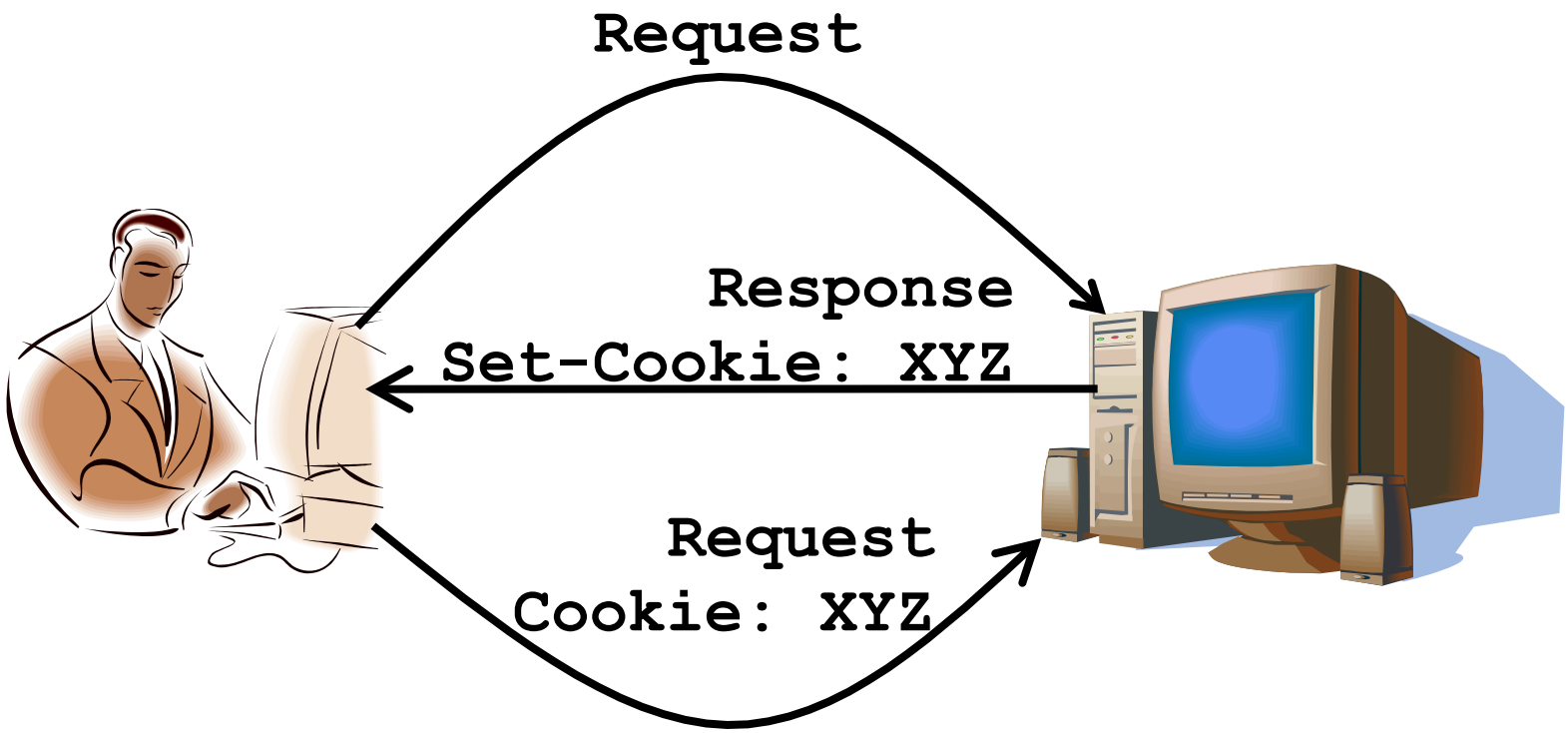
Question

- How does a stateless protocol keep state?

State in a Stateless Protocol:

Cookies

- *Client-side* state maintenance
  - Client stores small state on behalf of server
  - Client sends state in future requests to the server
- Can provide authentication

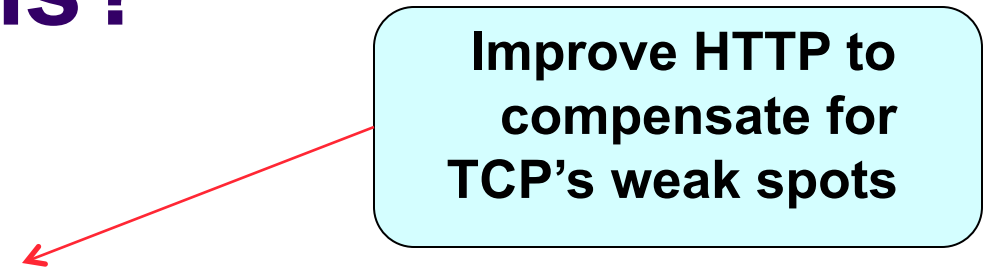


HTTP Performance Issues

Performance Goals

- User
  - fast downloads (not identical to low-latency commn.!)
  - high availability
- Content provider
  - happy users (hence, above)
  - cost-effective infrastructure
- Network (secondary)
  - avoid overload

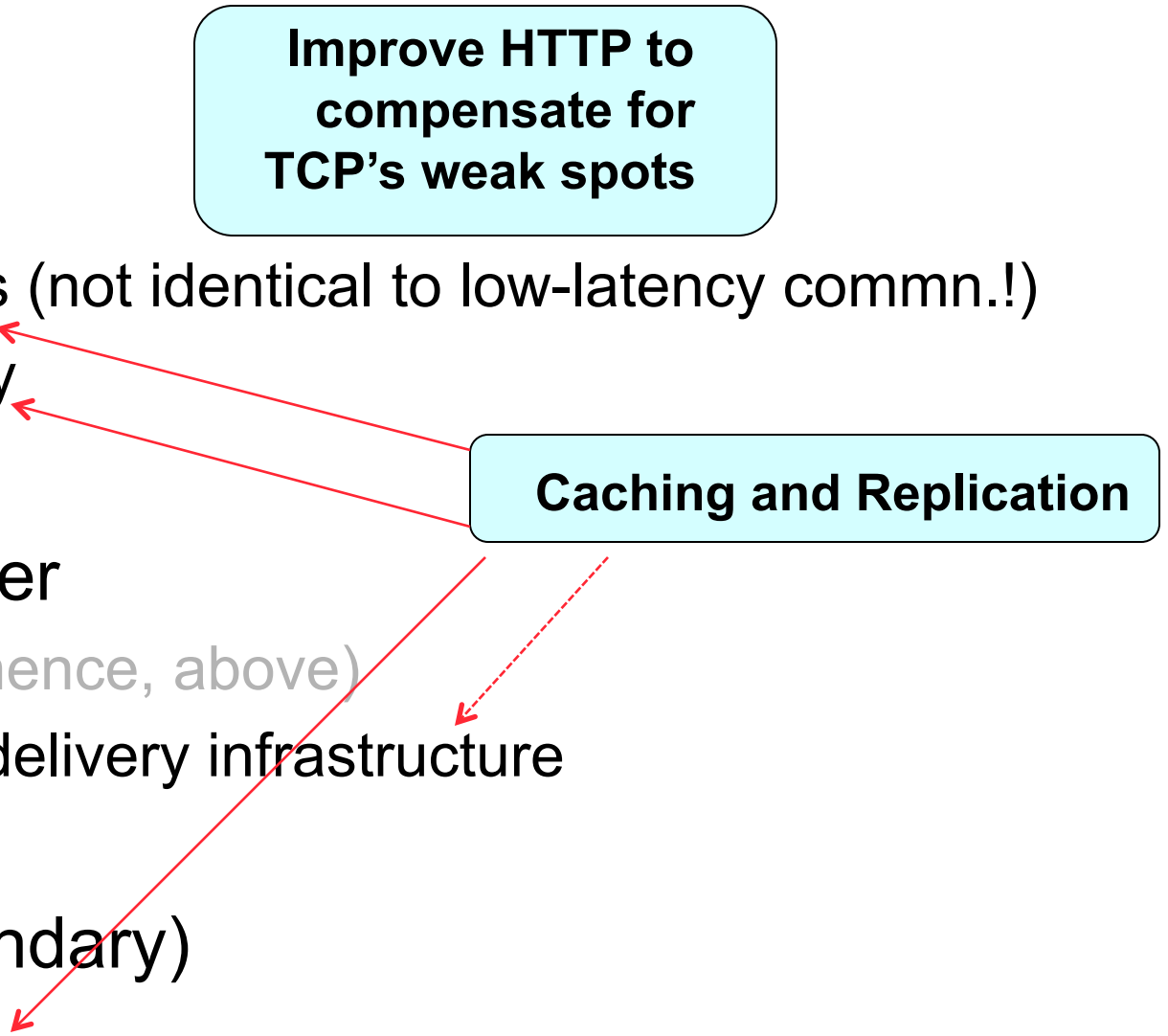
Solutions?



- User
  - fast downloads (not identical to low-latency commn.!)
  - high availability
- Content provider
  - happy users (hence, above)
  - cost-effective infrastructure
- Network (secondary)
  - avoid overload

Solutions?

- User
  - fast downloads (not identical to low-latency commn.!)
  - high availability
- Content provider
  - happy users (hence, above)
  - cost-effective delivery infrastructure
- Network (secondary)
  - avoid overload



Solutions?

Improve HTTP to compensate for TCP's weak spots

- User
  - fast downloads (not identical to low-latency commn.!)
  - high availability

Caching and Replication

- Content provider
  - happy users (hence, above)
  - cost-effective delivery infrastructure

- Network (secondary)
  - avoid overload

Exploit economies of scale (Webhosting, CDNs, datacenters)

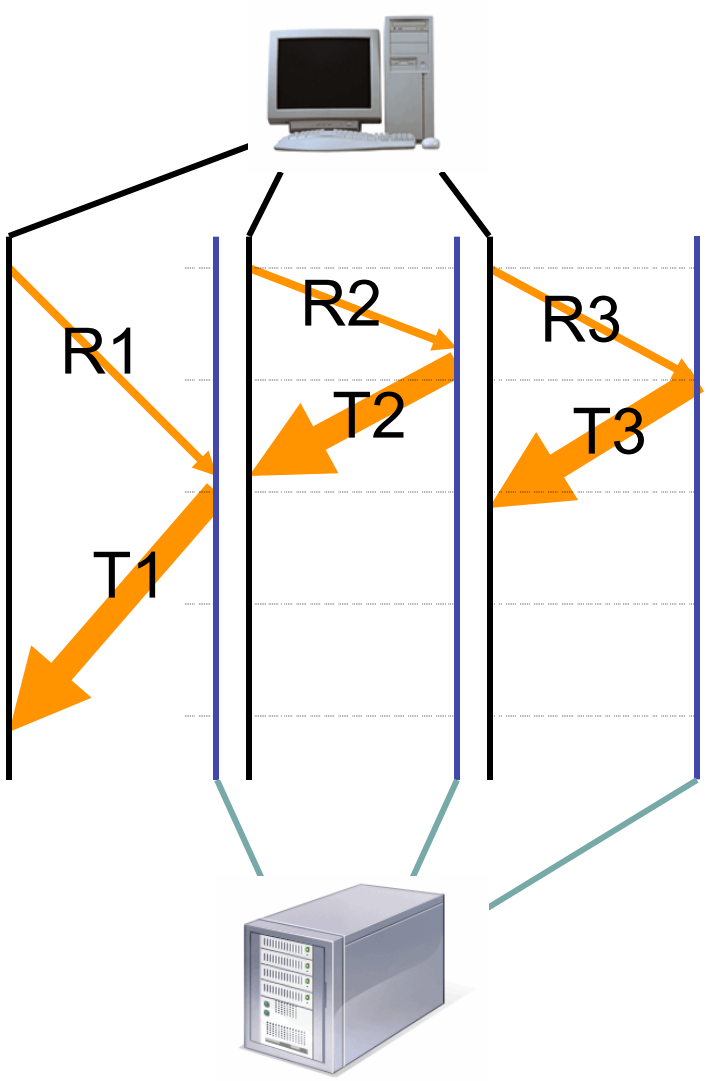
HTTP Performance

- Most Web pages have multiple objects
  - *e.g.,* HTML file and a bunch of embedded images
- How do you retrieve those objects (naively)?
  - *One item at a time*
- New TCP connection per (small) object!

Improving HTTP Performance:

Concurrent Requests & Responses

- Use multiple connections *in parallel*
- Does not necessarily maintain order of responses



- Client = ☺
- Content provider = ☺
- Network = ☹ Why?

Improving HTTP Performance:

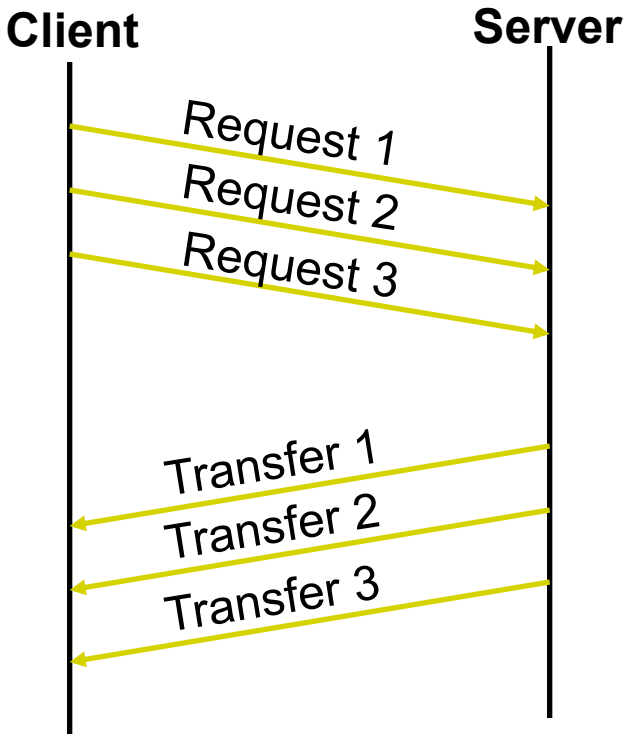
Persistent Connections

- Maintain TCP connection across multiple requests
  - Including transfers subsequent to current page
  - Client or server can tear down connection
- Performance advantages:
  - Avoid overhead of connection set-up and tear-down
  - Allow TCP to learn more accurate RTT estimate
  - Allow TCP congestion window to increase
  - i.e., leverage previously discovered bandwidth
- Default in HTTP/1.1

Improving HTTP Performance:

Pipelined Requests & Responses

- Batch requests and responses to reduce the number of packets
- Multiple requests can be contained in one TCP segment



Scorecard: Getting *n* Small Objects

Time dominated by latency

- One-at-a-time: ~2n RTT
- M concurrent: ~2[n/m] RTT
- Persistent: ~ (n+1)RTT
- Pipelined: ~2 RTT
- Pipelined/Persistent: ~2 RTT first time, RTT later

Scorecard: Getting *n* Large Objects

Time dominated by bandwidth

- One-at-a-time: ~ nF/B
- M concurrent: ~ [n/m] F/B
  - assuming shared with large population of users
  - and each TCP connection gets the same bandwidth
- Pipelined and/or persistent: ~ nF/B
  - The only thing that helps is getting more bandwidth..

Improving HTTP Performance:

Caching

- Why does caching work?
  - Exploits *locality of reference*
- How well does caching work?
  - Very well, up to a limit
  - Large overlap in content
  - But many unique requests
    - A universal story!
    - Effectiveness of caching grows logarithmically with size

Improving HTTP Performance:

Caching: How

- Modifier to GET requests:
  - `If-modified-since` – returns “not modified” if resource not modified since specified time

- Client specifies “if-modified-since” time in request
- Server compares this against “last modified” time of resource
- Server returns “Not Modified” if resource has not changed
- …. or a “OK” with the latest version otherwise

Improving HTTP Performance:

Caching: How

- Modifier to GET requests:
  - `If-modified-since` – returns “not modified” if resource not modified since specified time
- Response header:
  - `Expires` – how long it’s safe to cache the resource
  - `No-cache` – ignore all caches; always get resource directly from server

Improving HTTP Performance:

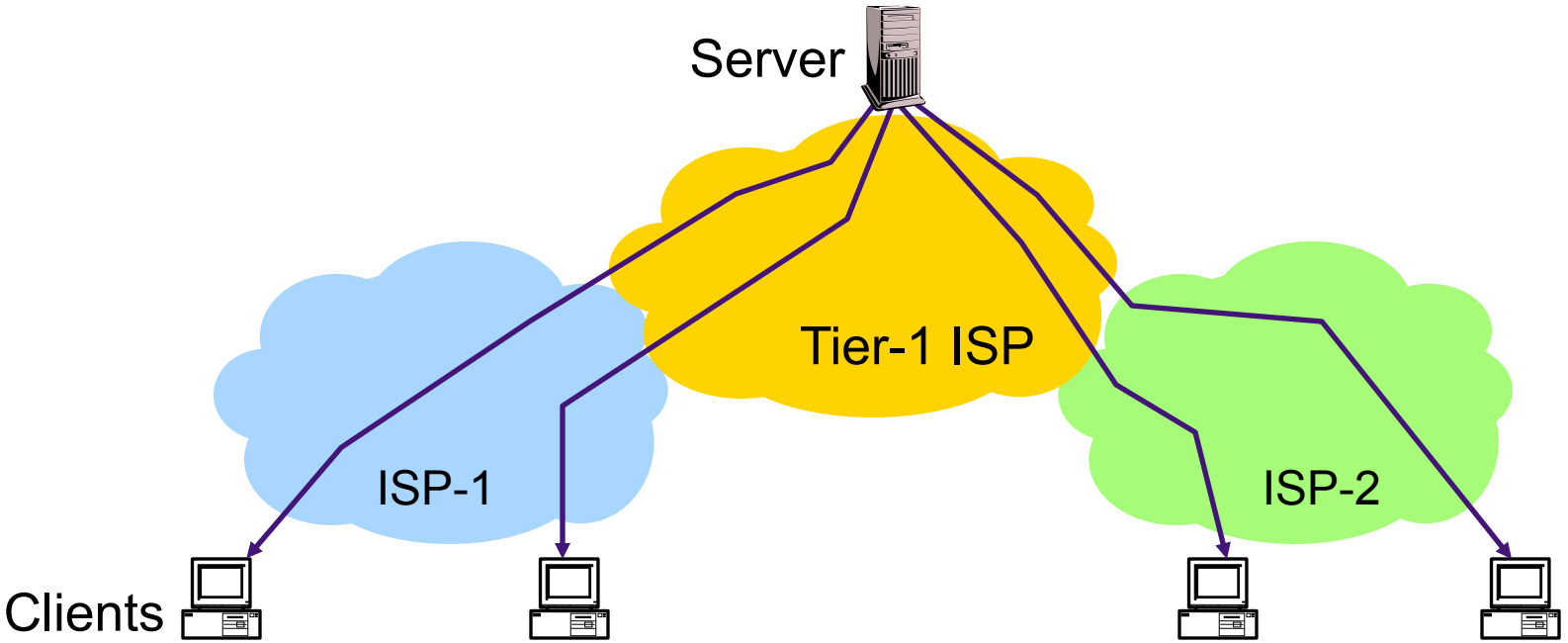
Caching: Where?

- Options
  - Client
  - Forward proxies
  - Reverse proxies
  - Content Distribution Network

Improving HTTP Performance:

Caching: Where?

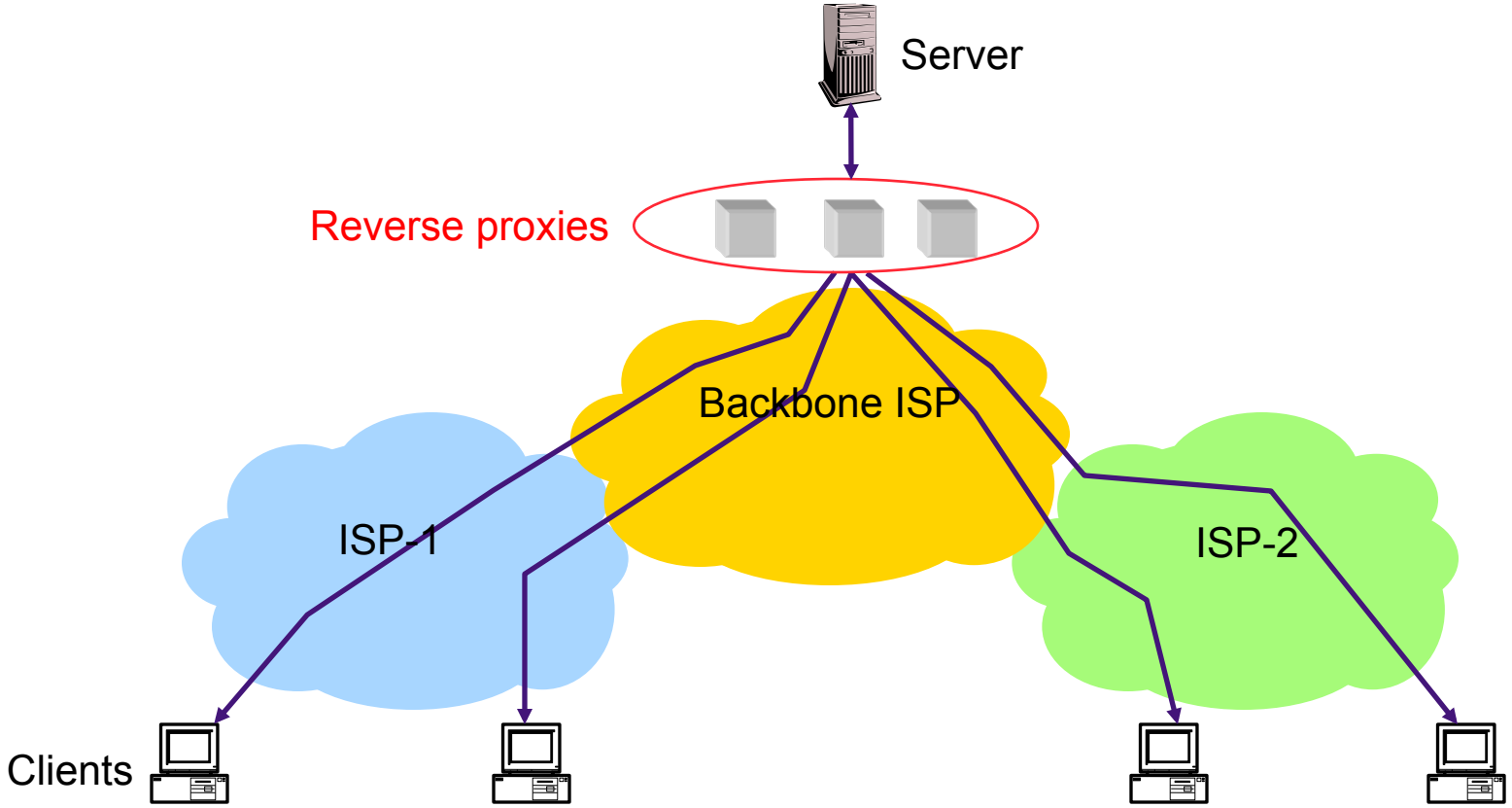
- Baseline: Many clients transfer same information
  - Generate unnecessary server and network load
  - Clients experience unnecessary latency



Improving HTTP Performance:

Caching with Reverse Proxies

- Cache documents close to **server**
  → decrease server load
- Typically done by content provider



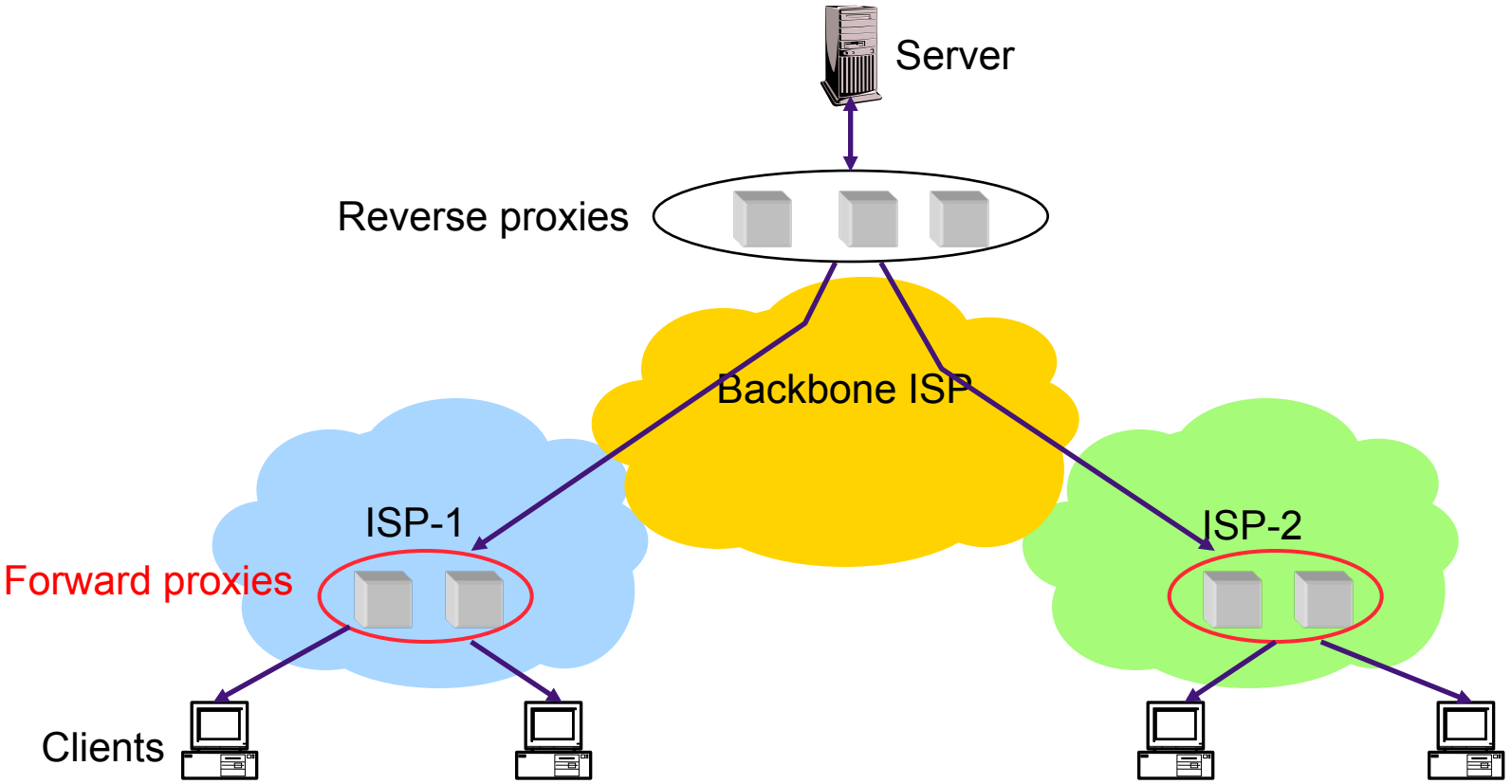
Improving HTTP Performance:

Caching with Forward Proxies

- Cache documents close to **clients**

  → reduce network traffic and decrease latency

- Typically done by ISPs or enterprises



Improving HTTP Performance:

Replication

- Replicate popular Web site across many machines
  - Spreads load on servers
  - Places content closer to clients
  - Helps when content isn't cacheable
- Problem: Want to direct client to particular replica
  - Balance load across server replicas
  - Pair clients with nearby servers
- Common solution:
  - DNS returns different addresses based on client's geo location, server load, *etc.*

Improving HTTP Performance:

Content Distribution Networks

- Caching and replication as a service
- Large-scale distributed storage infrastructure (usually) administered by one entity
  - *e.g.,* Akamai has servers in 20,000+ locations
- Combination of (pull) caching and (push) replication
  - **Pull:** Direct result of clients' requests
  - **Push:** Expectation of high access rate
- Also do some processing
  - Handle *dynamic* web pages
  - *Transcoding*

Improving HTTP Performance:

CDN Example – Akamai

- Akamai creates new domain names for each client
  - e.g., *a128.g.akamai.net* for *cnn.com*
- The CDN's DNS servers are authoritative for the new domains
- The client content provider modifies its content so that embedded URLs reference the new domains.
  - "Akamaize" content
  - e.g.: *http://www.cnn.com/image-of-the-day.gif* becomes *http://a128.g.akamai.net/image-of-the-day.gif*
- Requests now sent to CDN's infrastructure…

Cost-Effective Content Delivery

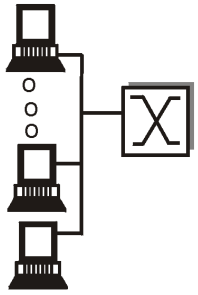
- General theme: multiple sites hosted on shared physical infrastructure
  - efficiency of statistical multiplexing
  - economies of scale (volume pricing, *etc.*)
  - amortization of human operator costs

- Examples:
  - Web hosting companies
  - CDNs
  - Cloud infrastructure

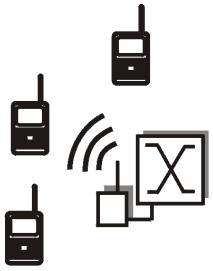
Data Link Layer

Point-to-Point vs. Broadcast Media

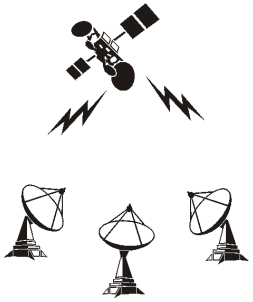
- Point-to-point: dedicated pairwise communication
  - E.g., long-distance fiber link
  - E.g., Point-to-point link between Ethernet switch and host
- Broadcast: shared wire or medium
  - Traditional Ethernet (pre ~2000)
  - 802.11 wireless LAN



shared wire
(e.g. Ethernet)



shared wireless
(e.g. Wavelan)



satellite

cocktail party

Multiple Access Algorithm

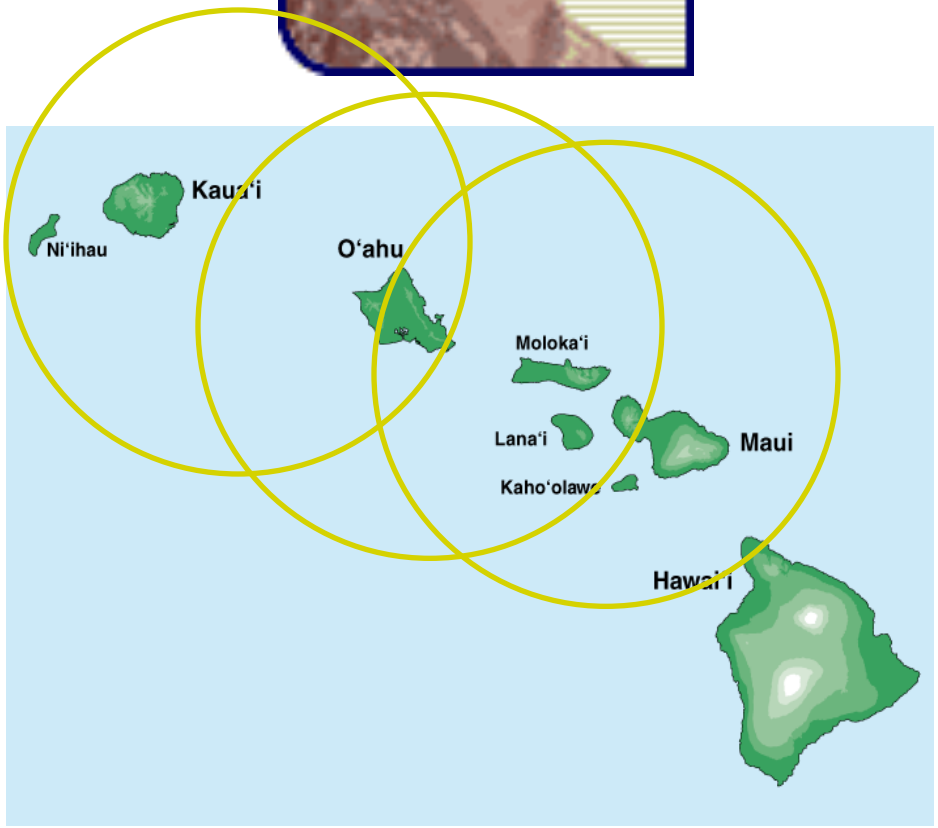
- Context: a shared broadcast channel
  - Must avoid having multiple nodes speaking at once
  - Otherwise, collisions lead to garbled data
  - Need distributed algorithm for sharing the channel
  - Algorithm determines which node can transmit

- Three classes of techniques
  - Channel partitioning: divide channel into pieces
  - Taking turns: scheme for trading off who gets to transmit
  - Random access: allow collisions, and then recover
    - More in the Internet style!

Random Access MAC Protocols

- When node has packet to send
  - Transmit at full channel data rate
  - No *a priori* coordination among nodes
- Two or more transmitting nodes ⇒ collision
  - Data lost
- Random access MAC protocol specifies:
  - How to detect collisions
  - How to recover from collisions
- Examples
  - ALOHA and Slotted ALOHA
  - CSMA, CSMA/CD, CSMA/CA (wireless, covered later)

Where it all Started: AlohaNet

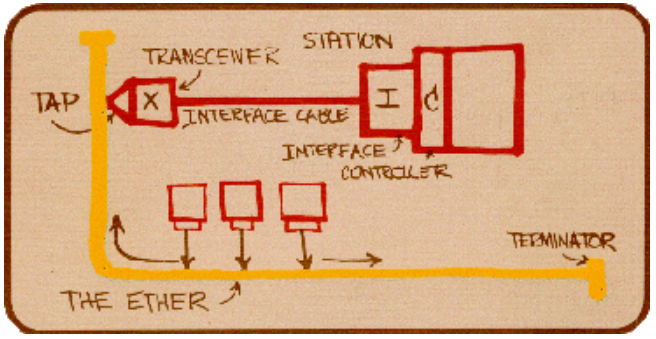


- Norm Abramson left Stanford in 1970 (***so he could surf!)***
- Set up first data communication system for Hawaiian islands
- Central hub at U. Hawaii, Oahu

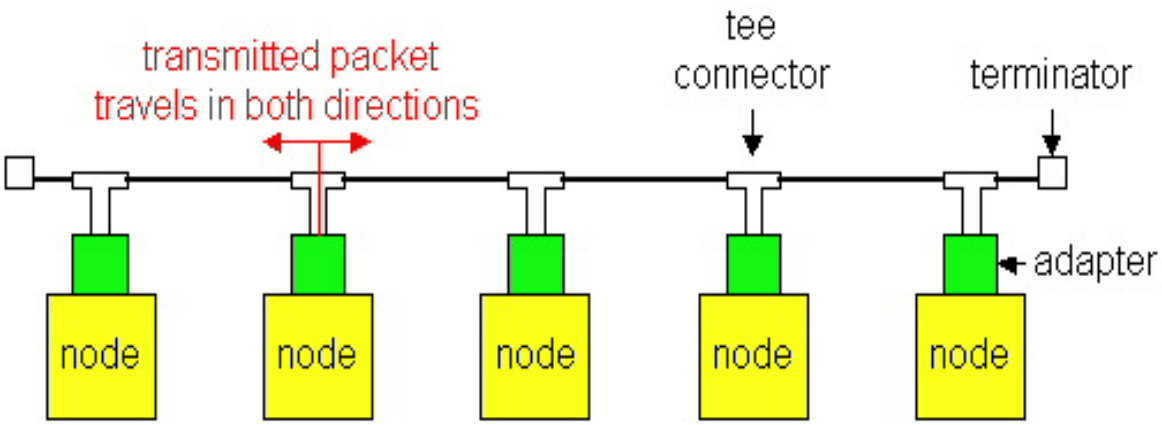
Aloha Signaling

- Two channels: random access, broadcast
- Sites send packets to hub (random-access channel)
  - If not received (due to collision), site resends
- Hub sends packets to all sites (broadcast channel)
  - Sites can receive even if they are also sending

Ethernet:



- Bob Metcalfe, Xerox PARC, visits Hawaii and gets an idea!
- Shared wired medium
  - coax cable



Evolution

- Ethernet was invented as a broadcast technology
  - Hosts share channel
  - Each packet received by all attached hosts
  - CSMA/CD for media access control

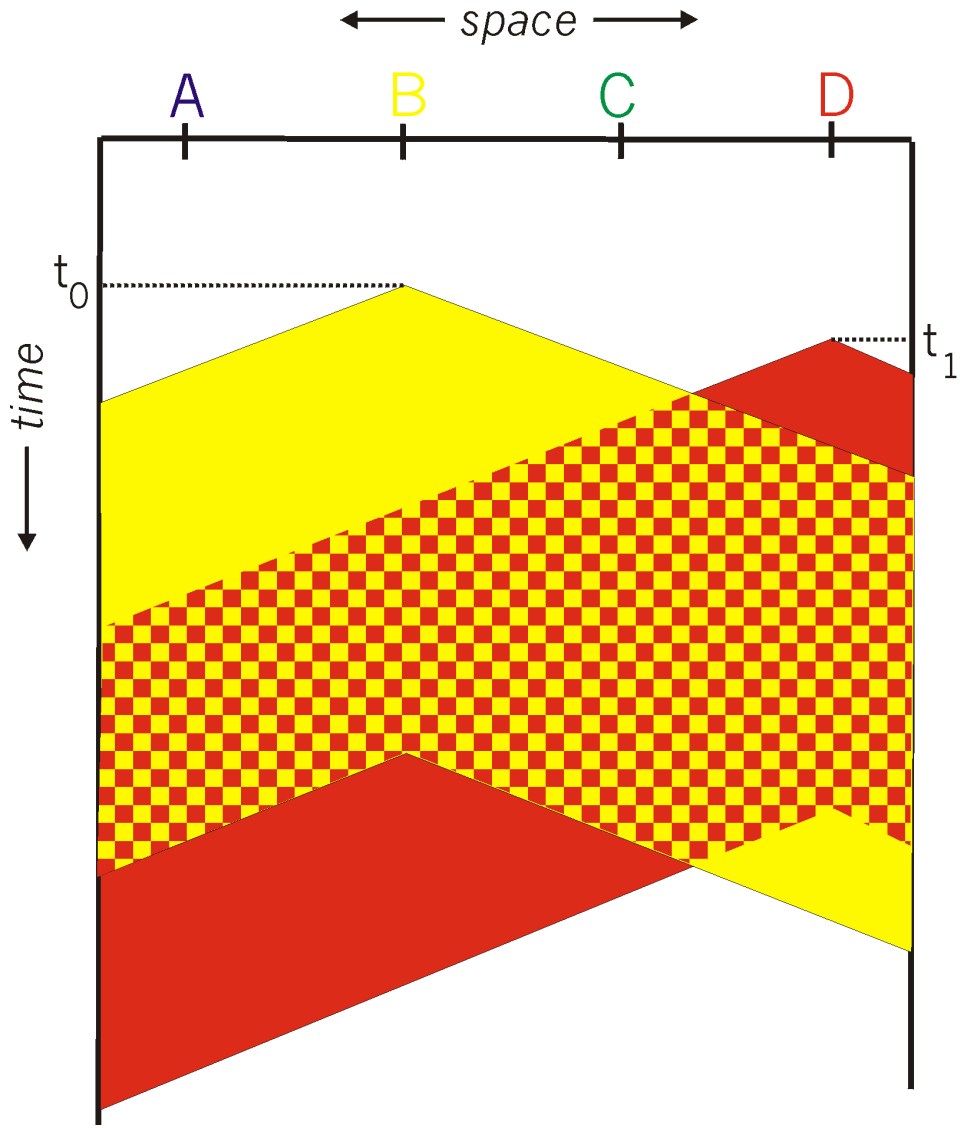
- Current Ethernets are “switched” (next lecture)
  - Point-to-point links between switches; between a host and switch
  - No sharing, no CSMA/CD
  - Uses “self learning” and “spanning tree” algorithms for routing

CSMA (Carrier Sense Multiple Access)

- CSMA: listen before transmit
  - If channel sensed idle: transmit entire frame
  - If channel sensed busy, defer transmission
- Human analogy: don’t interrupt others!
- Does this eliminate all collisions?
  - No, because of nonzero propagation delay

CSMA Collisions

Propagation delay: two nodes may not hear each other's before sending.



CSMA reduces but does not eliminate collisions

Biggest remaining problem?

Collisions still take the full transmission slot!

CSMA/CD (Collision Detection)

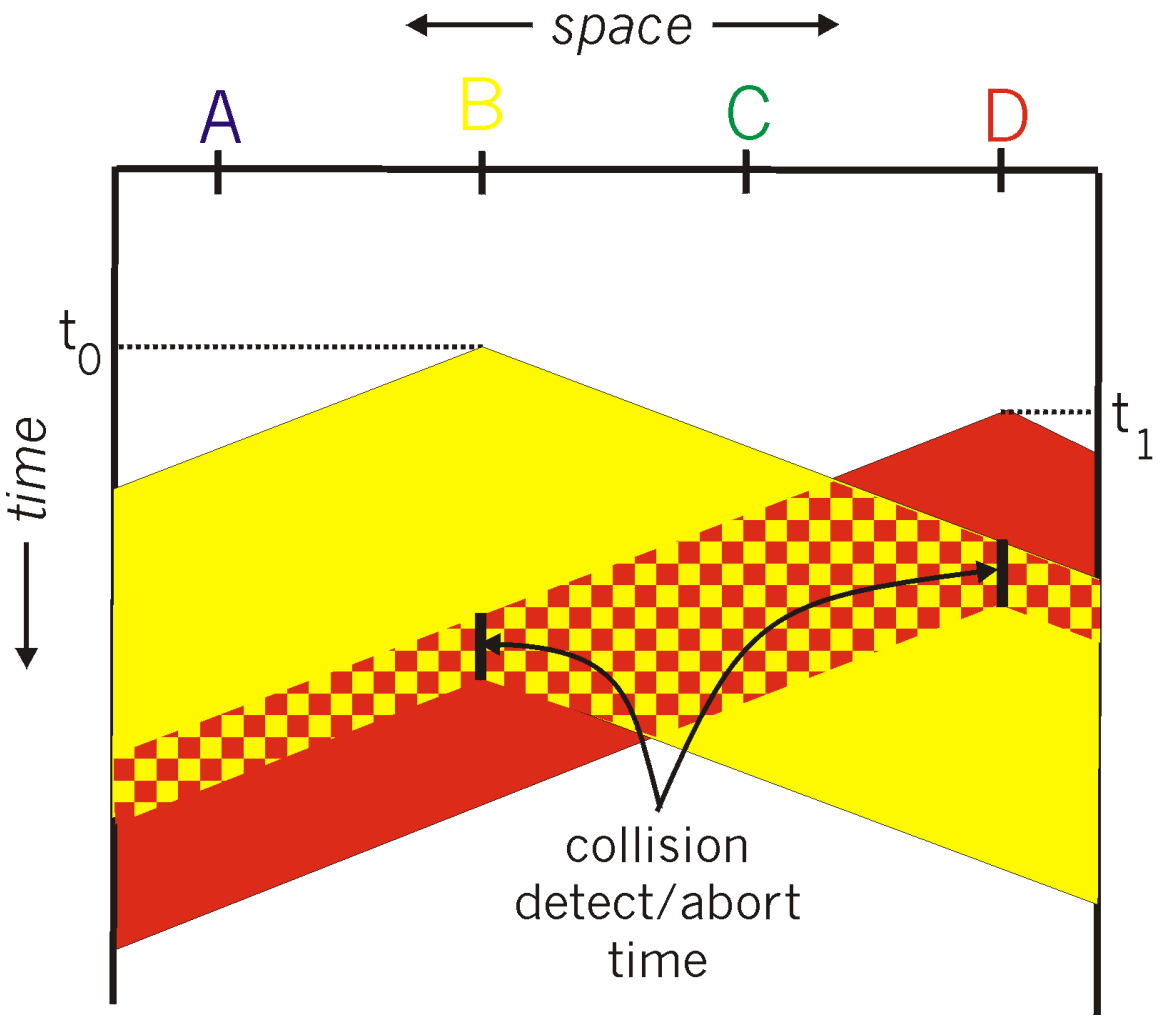
- CSMA/CD: carrier sensing, deferral as in CSMA
  - **Collisions detected within short time**
  - Colliding transmissions aborted, reducing wastage
- Collision detection easy in wired (broadcast) LANs
  - Compare transmitted, received signals
- Collision detection difficult in wireless LANs
  - *Lecture on wireless*

CSMA/CD Collision Detection

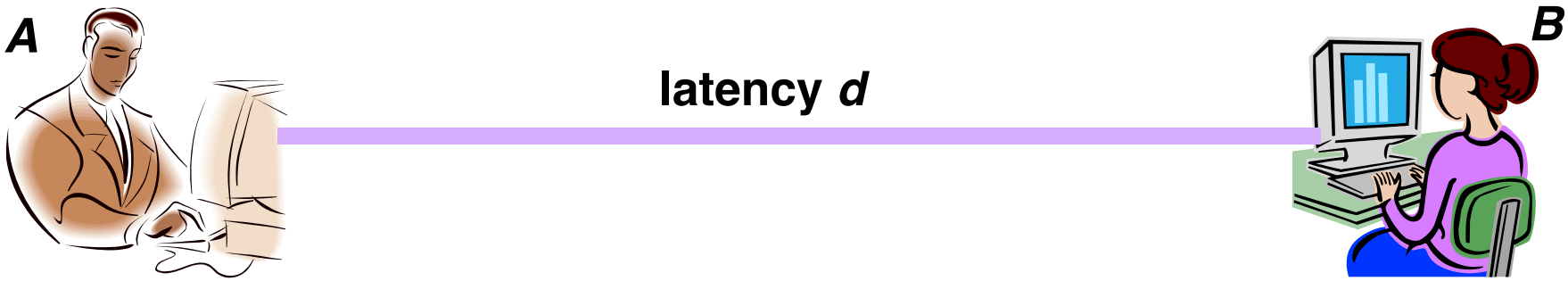
B and **D** can tell that collision occurred.

Note: for this to work, need restrictions on **minimum frame size** and **maximum distance.**

Why?

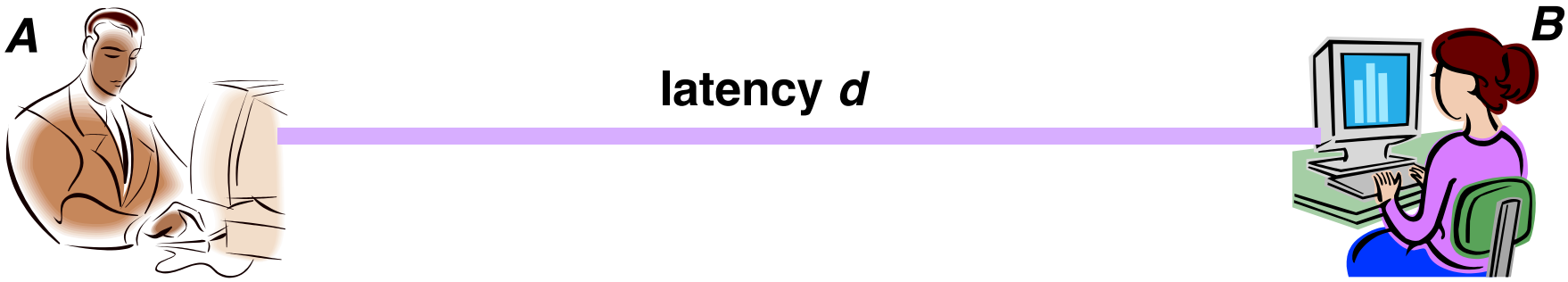


Limits on CSMA/CD Network Length



- Latency depends on physical length of link
  - Time to propagate a packet from one end to the other
- Suppose *A* sends a packet at time ***t***
  - And *B* sees an idle line at a time just before ***t+d***
  - … so *B* happily starts transmitting a packet
- *B* detects a collision, and sends jamming signal
  - But *A* can't see collision until ***t+2d***

Limits on CSMA/CD Network Length



- *A* needs to wait for time ***2d*** to detect collision
  - So, *A* should keep transmitting during this period
  - … and keep an eye out for a possible collision
- Imposes restrictions. E.g., for 10 Mbps Ethernet:
  - Maximum length of the wire: 2,500 meters
  - Minimum length of a frame: 512 bits (64 bytes)
    - 512 bits = 51.2 μsec (at 10 Mbit/sec)
    - For light in vacuum, 51.2 μsec ≈ 15,000 meters
      vs. 5,000 meters "round trip" to wait for collision
  - What about 10Gbps Ethernet?

Key Ideas of Random Access

1. Carrier sense
   - Listen before speaking, and don't interrupt
   - Checking if someone else is already sending data
   - … and waiting till the other node is done
2. Collision detection
   - If someone else starts talking at the same time, stop
     - *But make sure everyone knows there was a collision!*
   - Realizing when two nodes are transmitting at once
   - …by detecting that the data on the wire is garbled
3. Randomness
   - *Don't start talking again right away*
   - Waiting for a random time before trying again

How long should you wait?

- After collision, when should you resend?
- Should it be immediate?
- Should it be a random number with a fixed distribution?

Ethernet: CSMA/CD Protocol

- **Carrier sense**: wait for link to be idle
- **Collision detection**: listen while transmitting
  - No collision: transmission is complete
  - Collision: abort transmission & send **jam** signal
- **Random access**: binary exponential back-off
  - After collision, wait a random time before trying again
  - After m^{th} collision, choose K randomly from {0, …, 2^m-1}
  - … and wait for K*512 bit times before trying again
    - If transmission occurring when ready to send, wait until end of transmission (CSMA)

Performance of CSMA/CD

- Time wasted in collisions
  - Proportional to distance d
- Time spend transmitting a packet
  - Packet length p divided by bandwidth b
- Rough estimate for efficiency (K some constant)

$$E \sim \frac{\frac{p}{b}}{\frac{p}{b} + Kd}$$

- Note:
  - For large packets, small distances, E ~ 1
  - As bandwidth increases, E decreases
  - That is why high-speed LANs are all switched