Mid-point discussion lecture …

TDTS06 Computer Networking

Niklas Carlsson, Associate Professor
http://www.ida.liu.se/~nikca/

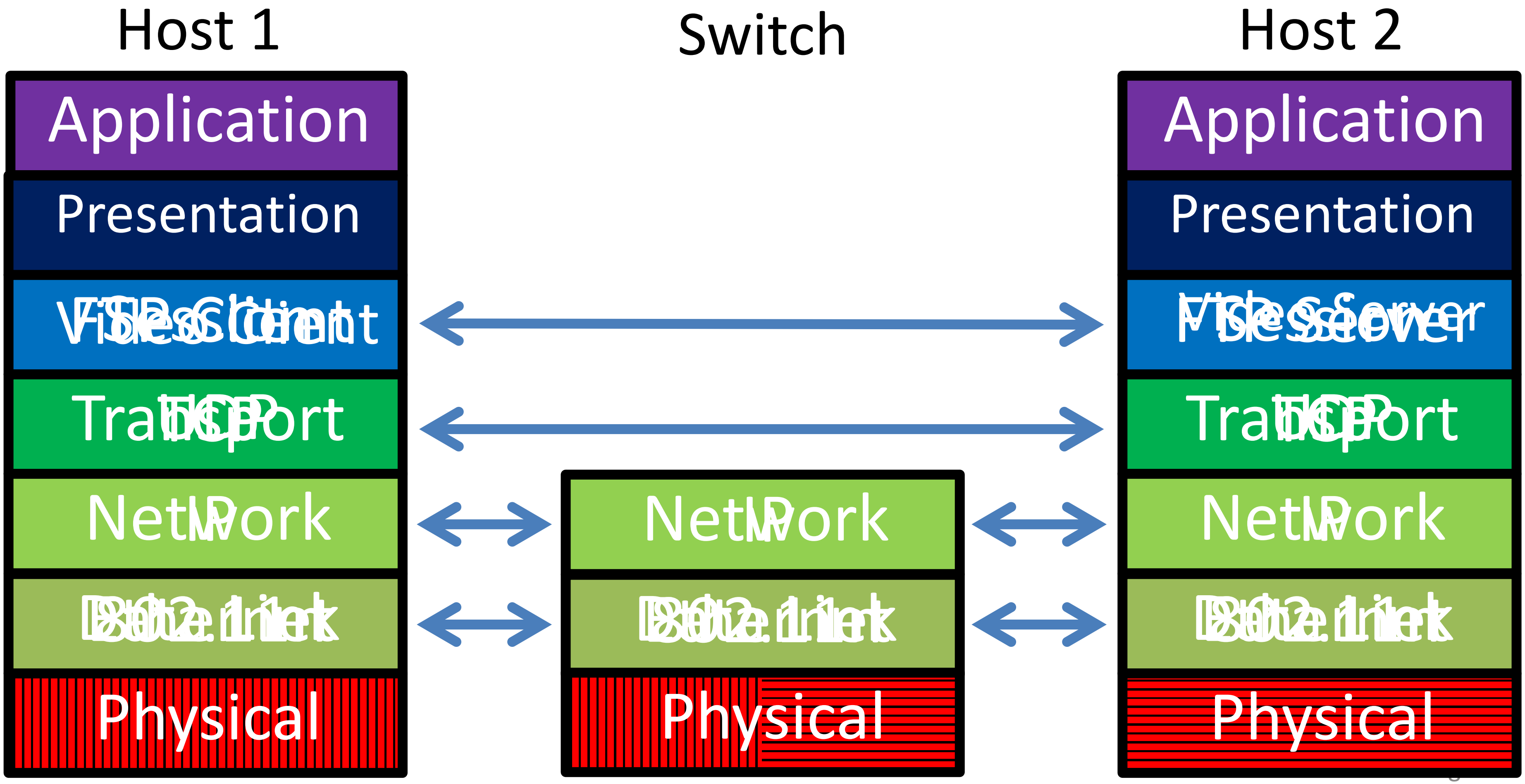
Kick starting science …

… well, cable into wall …

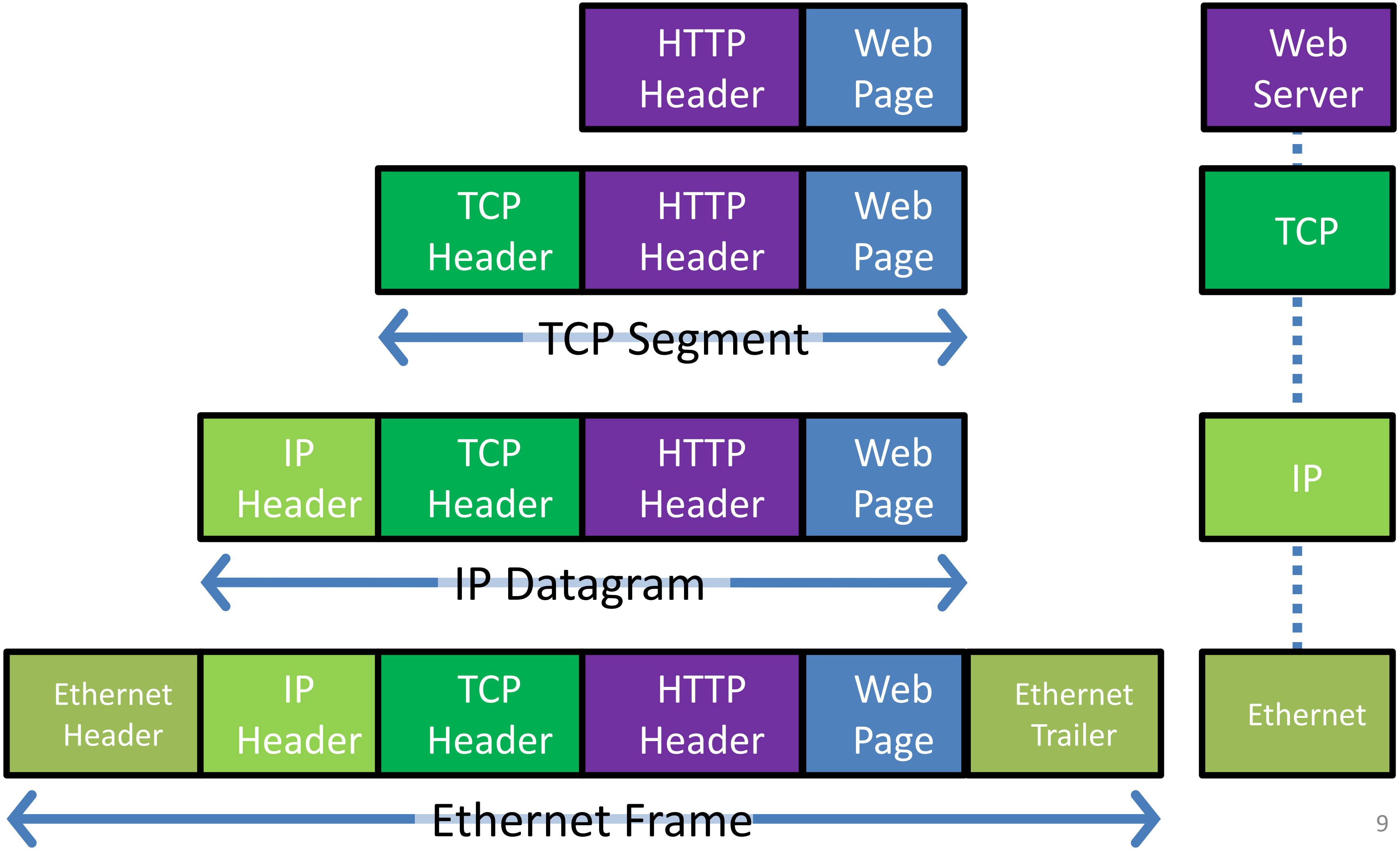
What happens there?

Hosts, the Internet architecture, and the E2E arguments …

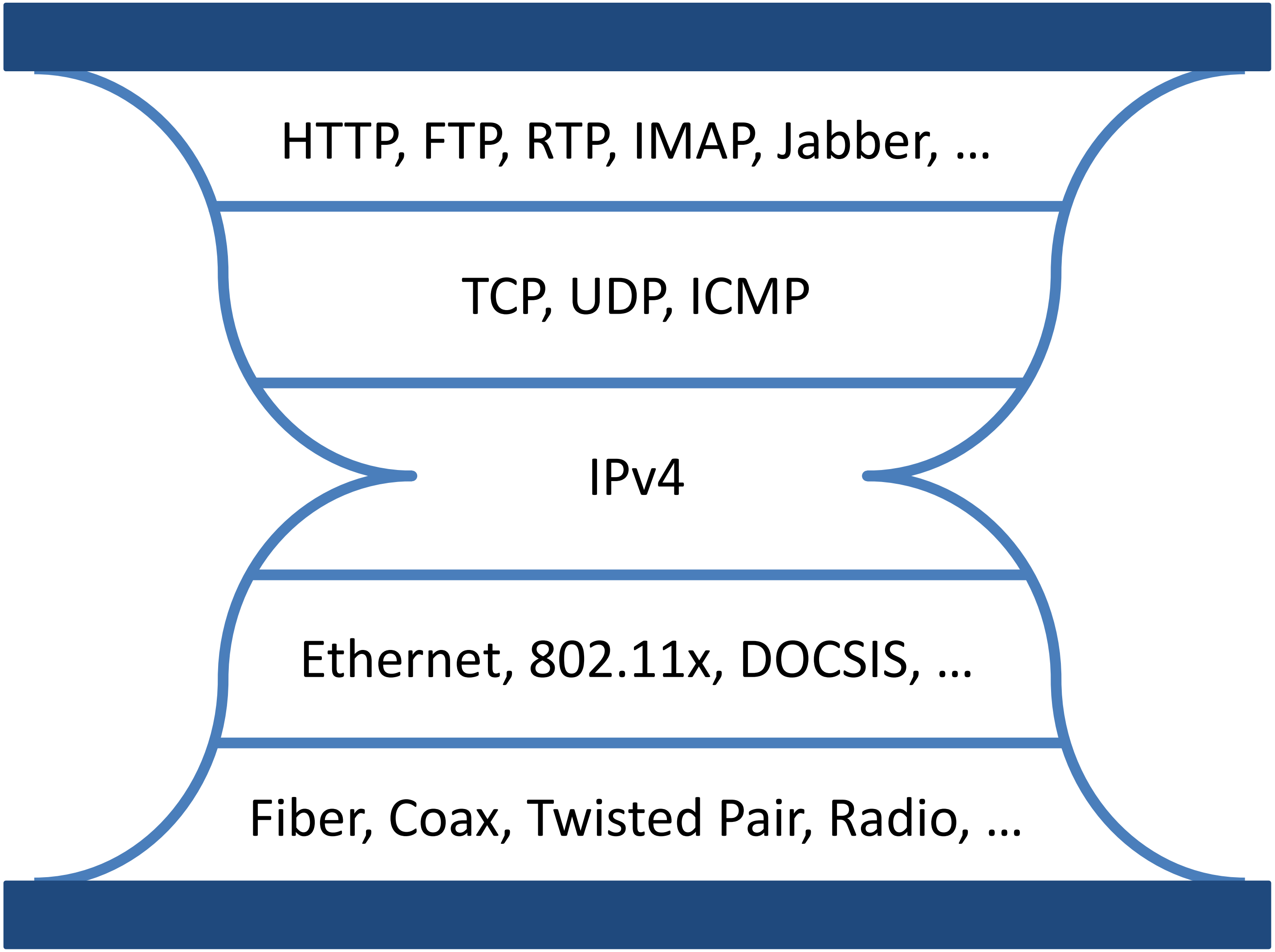
Network Stack in Practice



Encapsulation, Revisited



The Hourglass



HTTP, FTP, RTP, IMAP, Jabber, …

TCP, UDP, ICMP

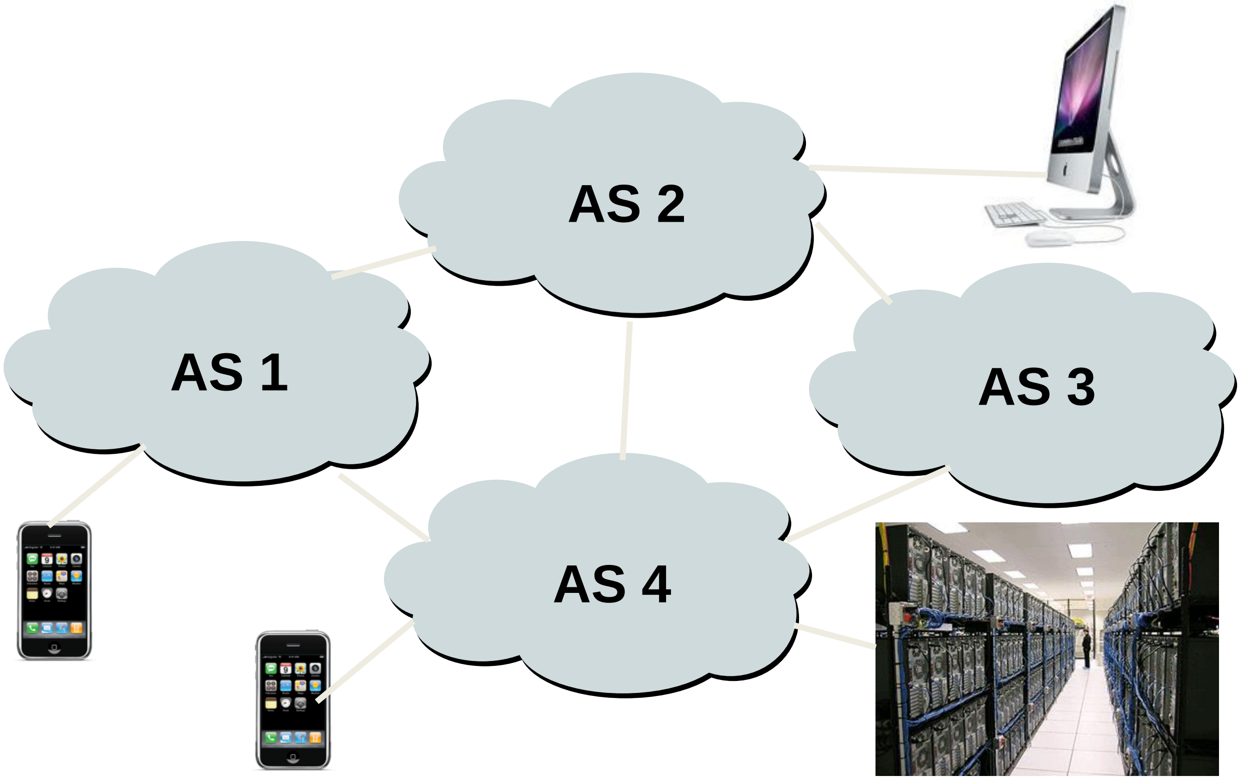
IPv4

Ethernet, 802.11x, DOCSIS, …

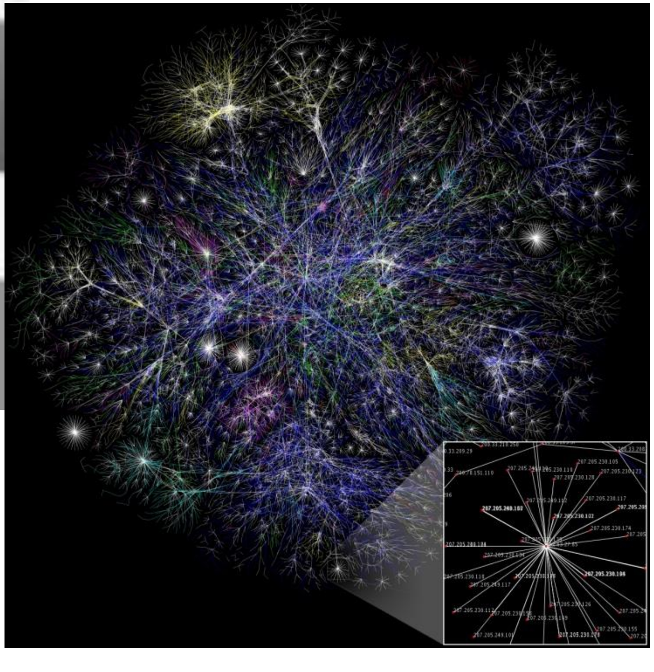
Fiber, Coax, Twisted Pair, Radio, …

Holding the Internet Together

- Distributed cooperation for resource allocation
  - BGP: what end-to-end *paths* to take (for ~50K ASes)
  - TCP: what *rate* to send over each path (for ~3B hosts)

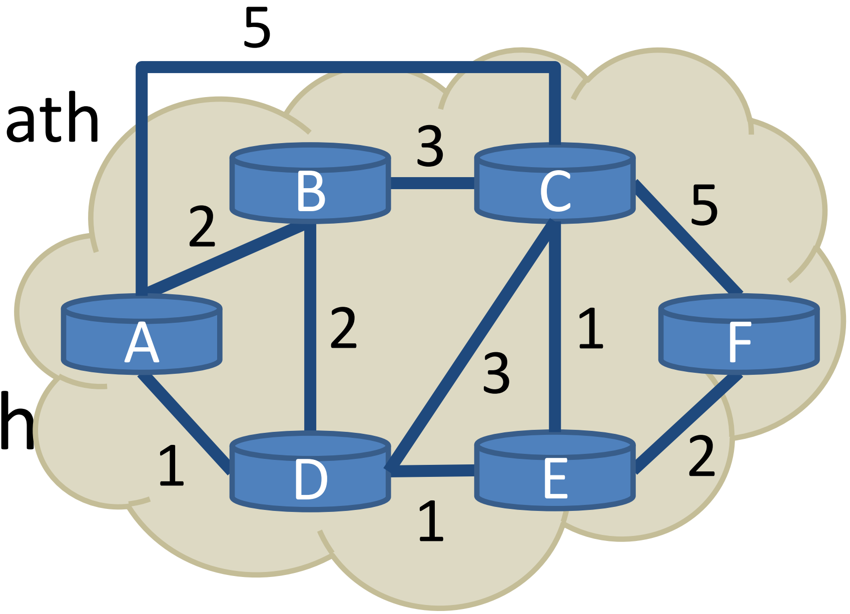


How do we find a path?



Routing on a Graph

- Goal: determine a "good" path through the network from source to destination
- What is a good path?
  - Usually means the shortest path
  - Load balanced
  - Lowest $$$ cost
- Network modeled as a graph
  - Routers → nodes
  - Link → edges
    - Edge cost: delay, congestion level, etc.

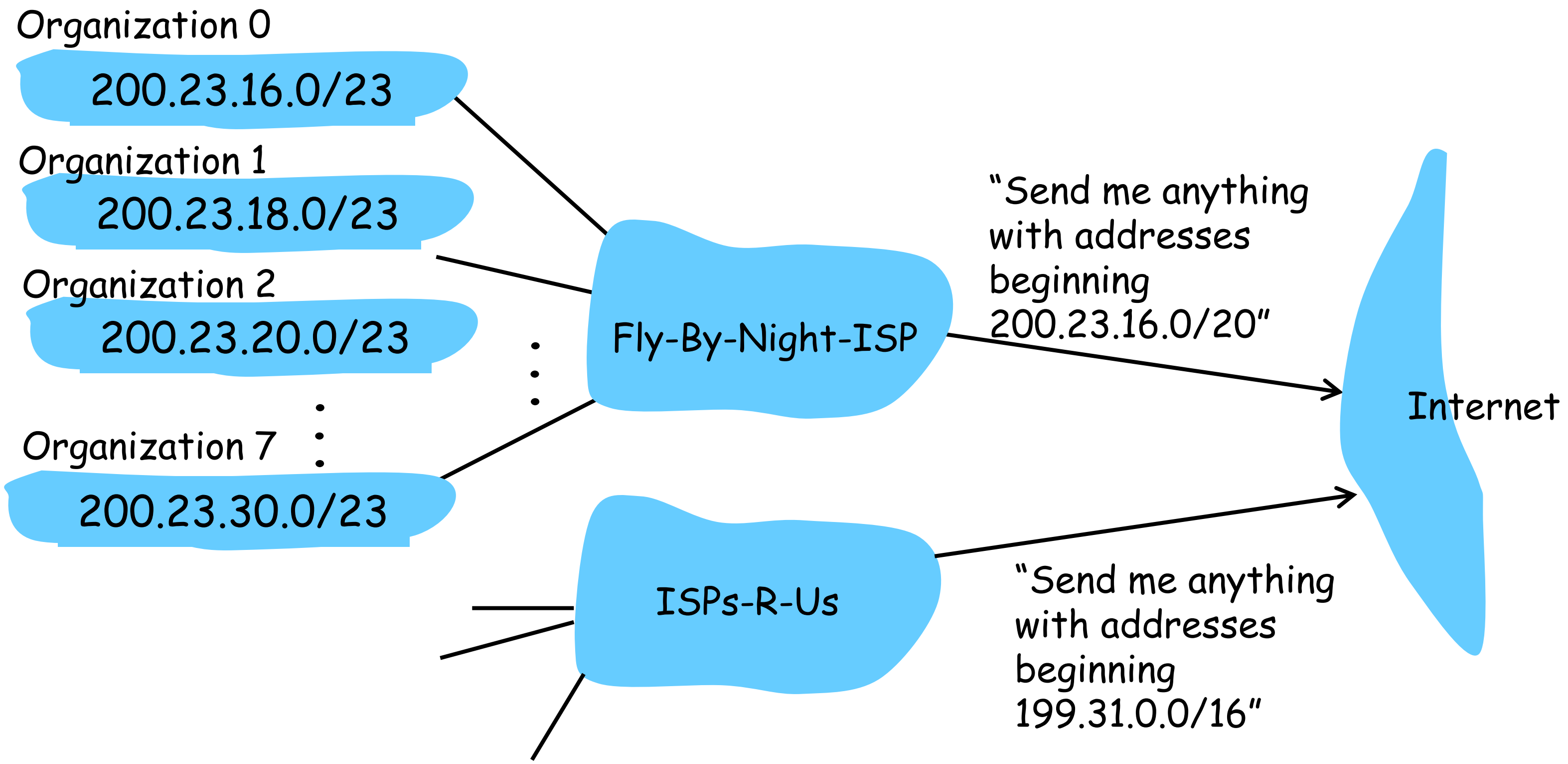


Intra-domain Routing Protocols

- Distance vector
  - Routing Information Protocol (RIP), based on Bellman-Ford
  - Routers periodically exchange reachability info with neighbors
- Link state
  - Open Shortest Path First (OSPF), based on Dijkstra
  - Each network periodically floods neighbor information to all routers
  - Routers locally compute routes

Hierarchical addressing: route aggregation

ISP has an address block; it can further divide this block into sub blocks and assign them to subscriber organizations.



"Send me anything with addresses beginning 200.23.16.0/20"

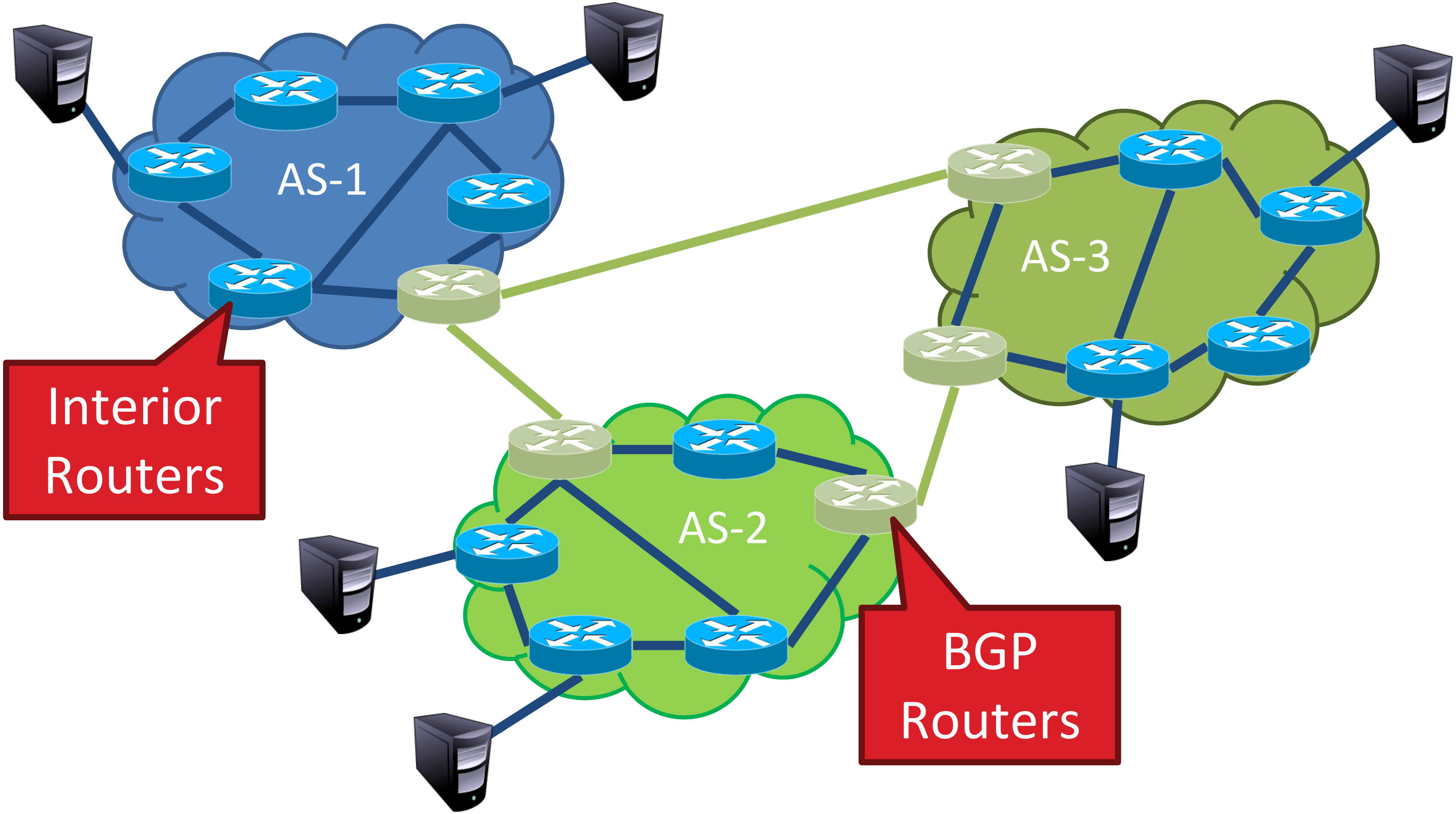
"Send me anything with addresses beginning 199.31.0.0/16"

Example CIDR Routing Table

Address	Netmask	Third Byte	Byte Range
207.46.0.0	19	000xxxxx	0 – 31
207.46.32.0	19	001xxxxx	32 – 63
207.46.64.0	19	010xxxxx	64 – 95
207.46.128.0	18	10xxxxxx	128 – 191
207.46.192.0	18	11xxxxxx	192 – 255

Hole in the Routing Table: No coverage for 96 – 127
207.46.96.0/19

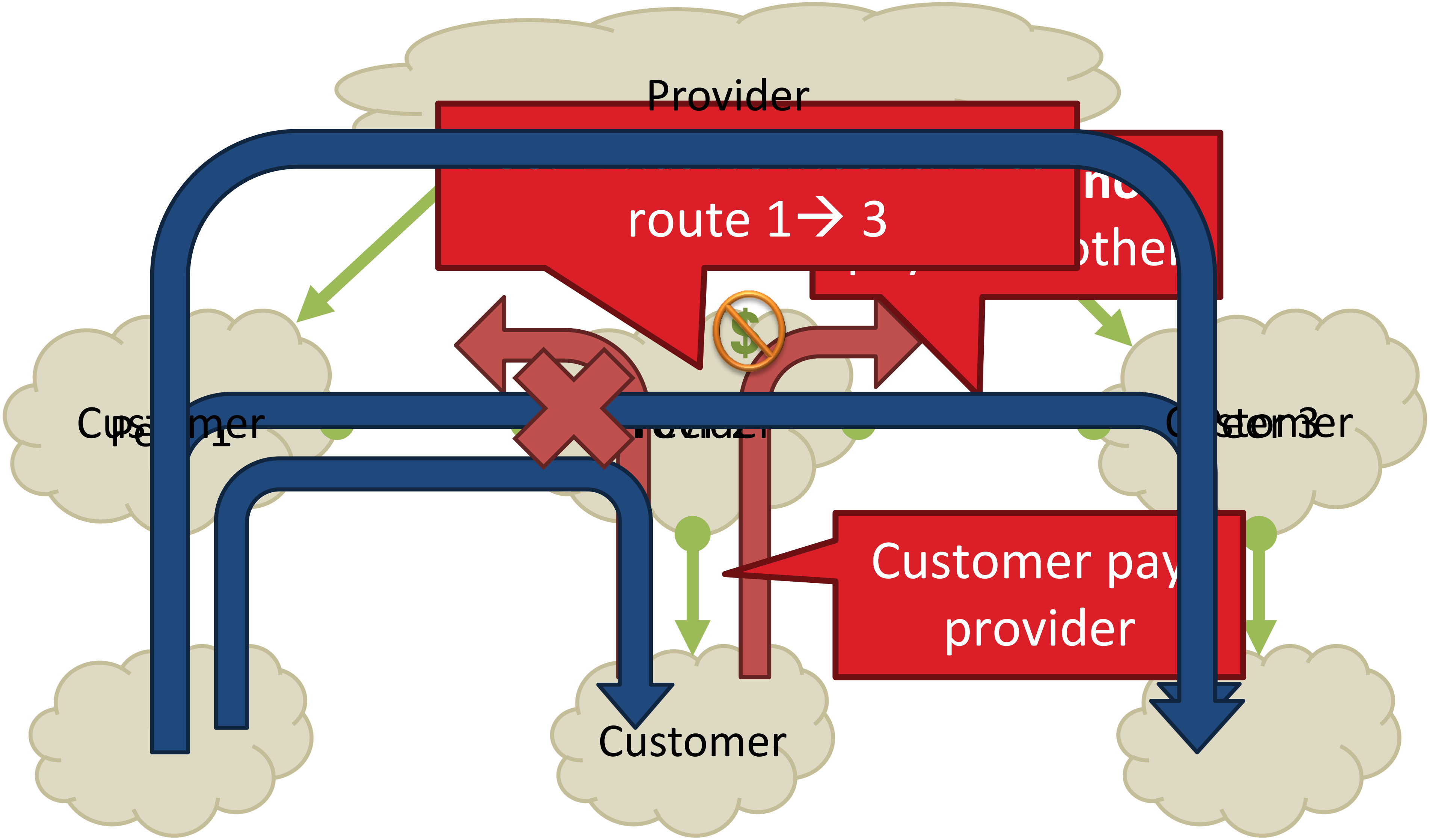
Network of networks: BGP and ASes



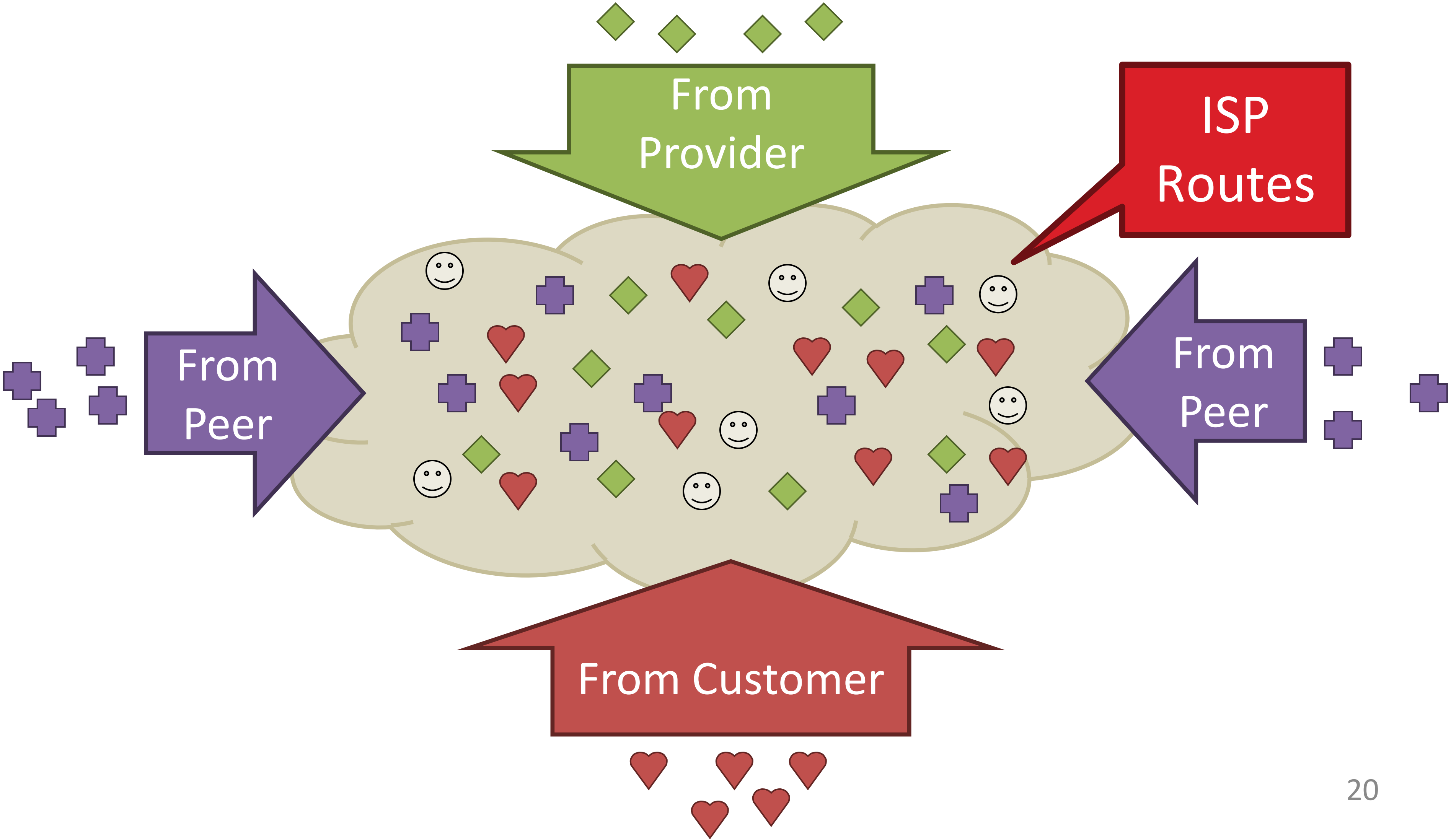
Interior Routers

BGP Routers

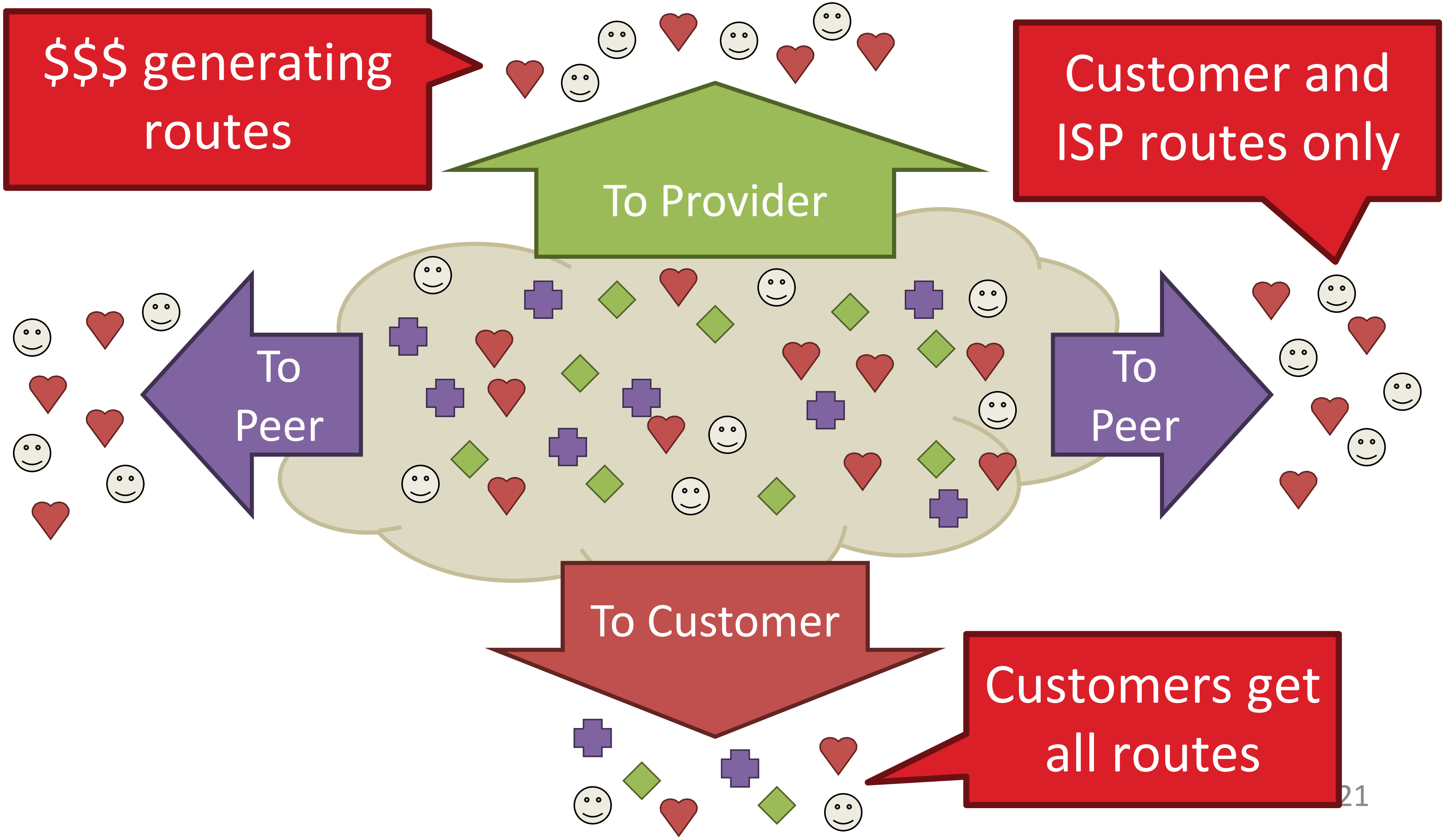
BGP Relationships



Importing Routes



Exporting Routes

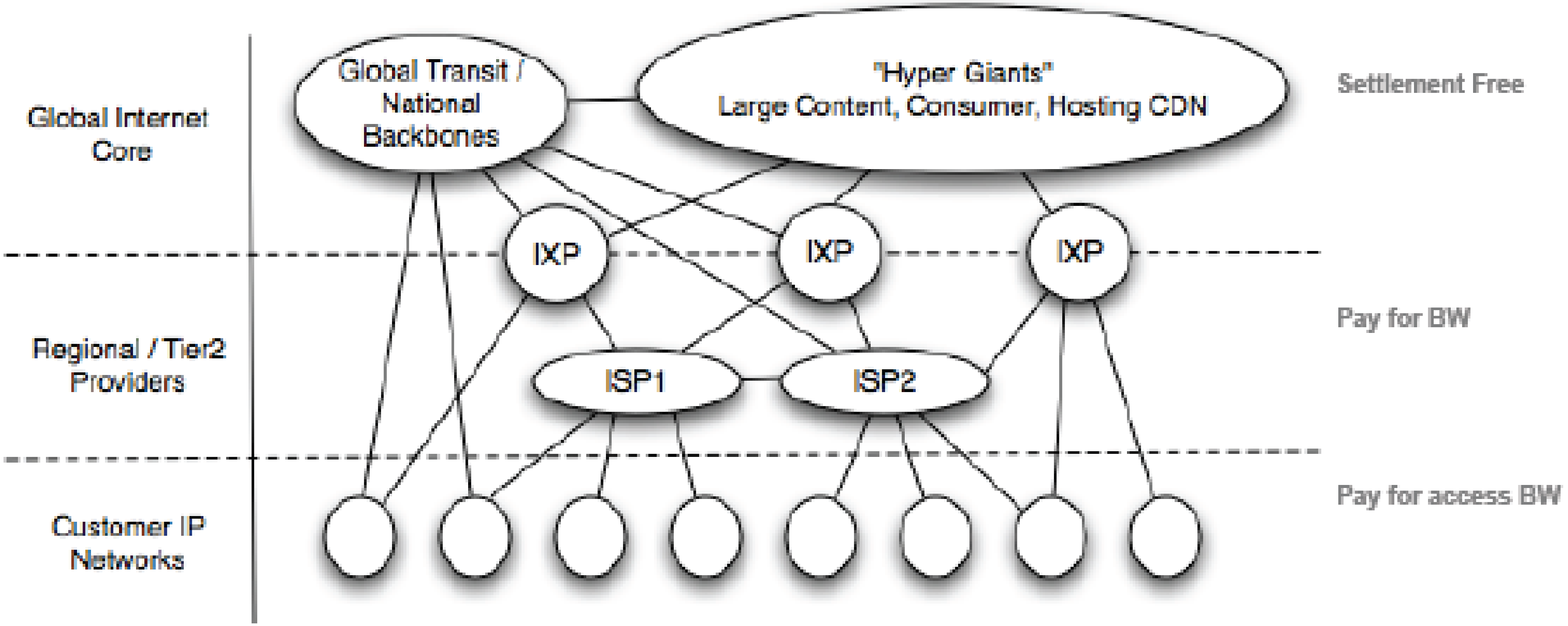


$$$ generating routes

Customer and ISP routes only

Customers get all routes

A new Internet model


Global Internet Core
Regional / Tier2 Providers
Customer IP Networks
Global Transit / National Backbones
"Hyper Giants" Large Content, Consumer, Hosting CDN
IXP
IXP
IXP
ISP1
ISP2
Settlement Free
Pay for BW
Pay for access BW

- Flatter and much more densely interconnected Internet
- Disintermediation between content and “eyeball” networks
- New commercial models between content, consumer and transit

How do we **avoid sending too much** for the receiver and network to handle?

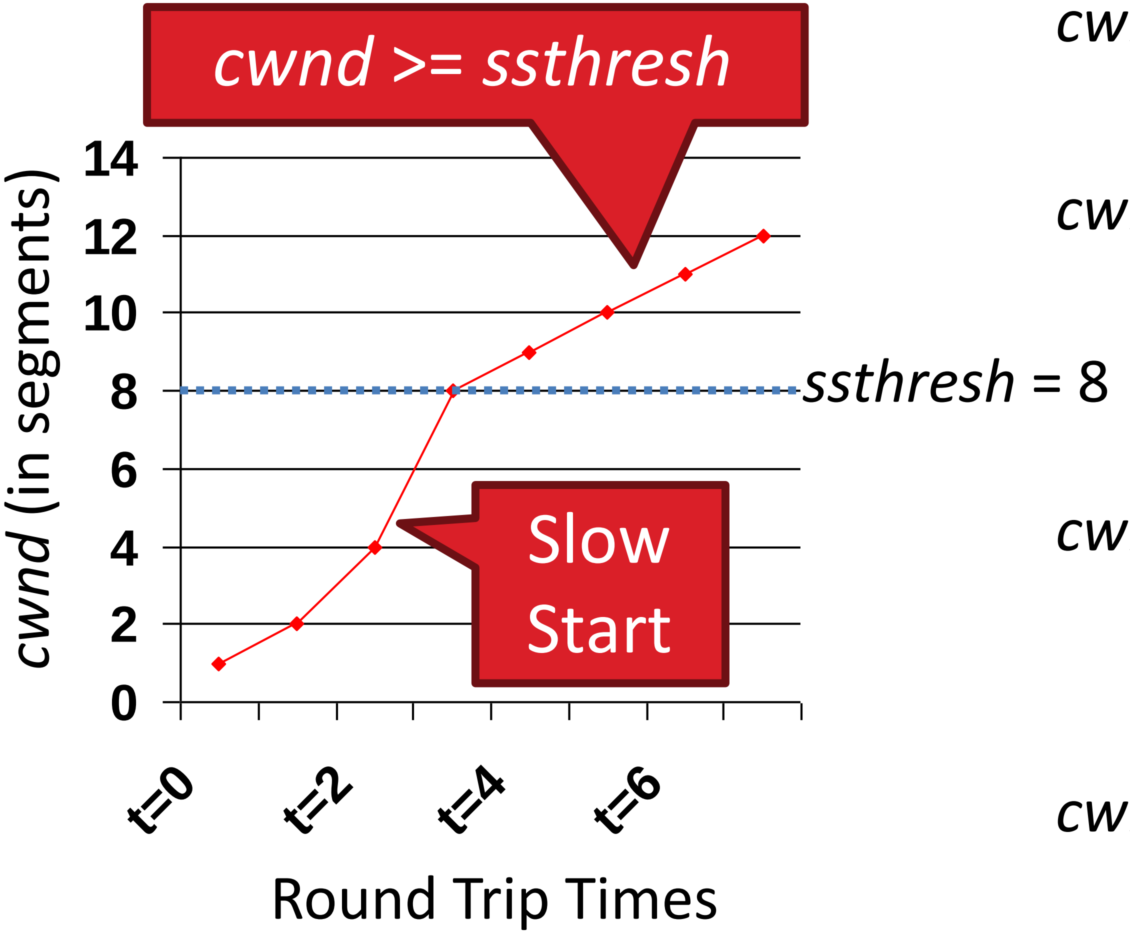
TCP Tahoe: Summary

- Basic ideas
  - Gently probe network for spare capacity
  - Drastically reduce rate on congestion
  - Windowing: self-clocking
  - Other functions: round trip time estimation, error recovery

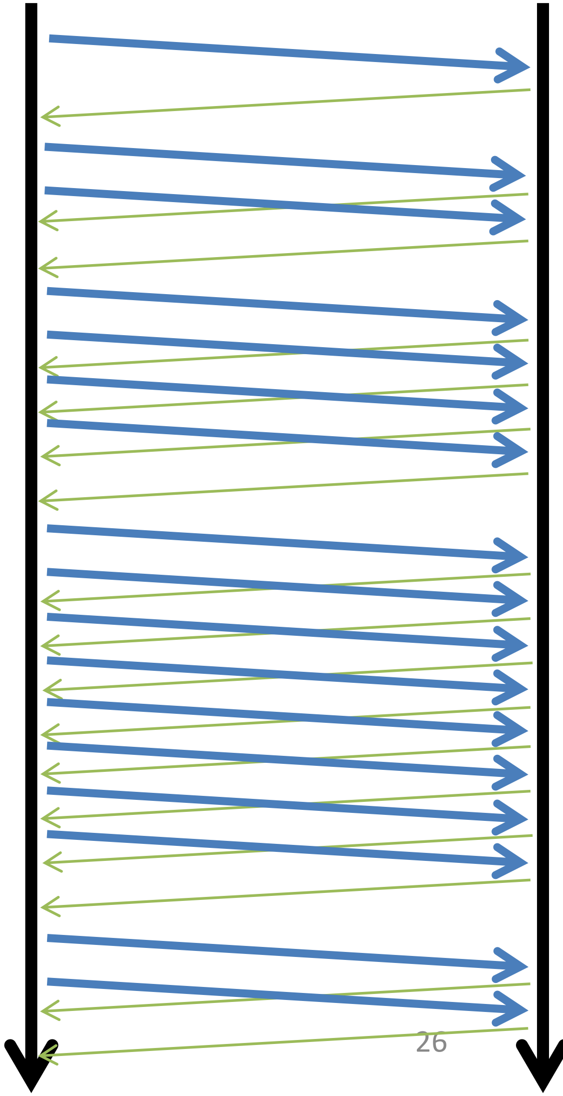
```
for every ACK {
        if (W < ssthresh) then W++          (SS)
        else            W += 1/W            (CA)

      }
      for every loss {
            ssthresh = W/2
            W  = 1
      }
```

Congestion Avoidance Example


cwnd >= *ssthresh*

Slow Start


cwnd = 1

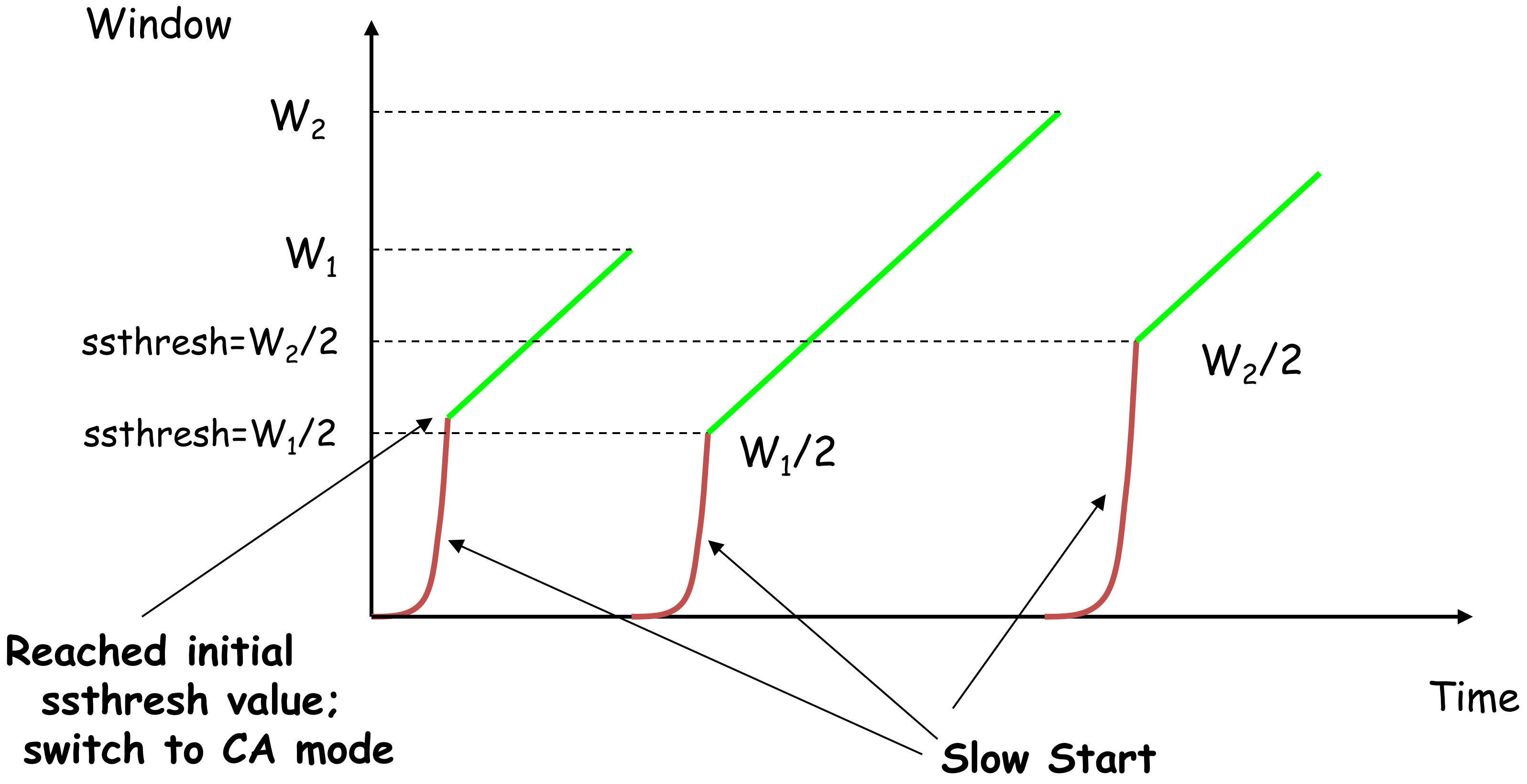
cwnd = 2

cwnd = 4

cwnd = 8

cwnd = 9

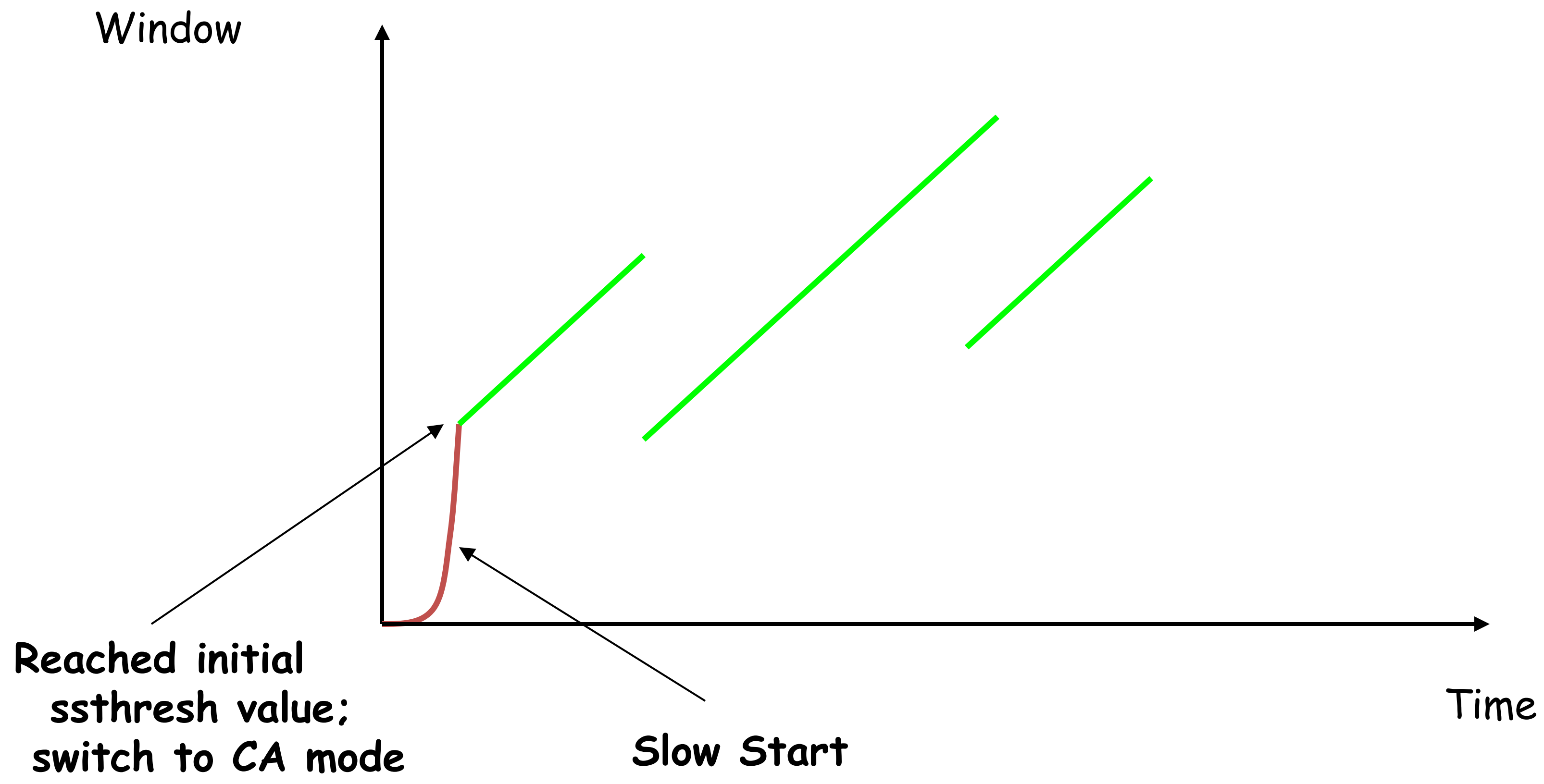
TCP Tahoe


Window
W_2
W_1
ssthresh=$W_2/2$
ssthresh=$W_1/2$
$W_2/2$
$W_1/2$
Time
Reached initial ssthresh value; switch to CA mode
Slow Start

- Can we do better than TCP Tahoe?

TCP Reno

Note how there is "Fast Recovery" after cutting Window in half



TCP Reno: Fast Recovery

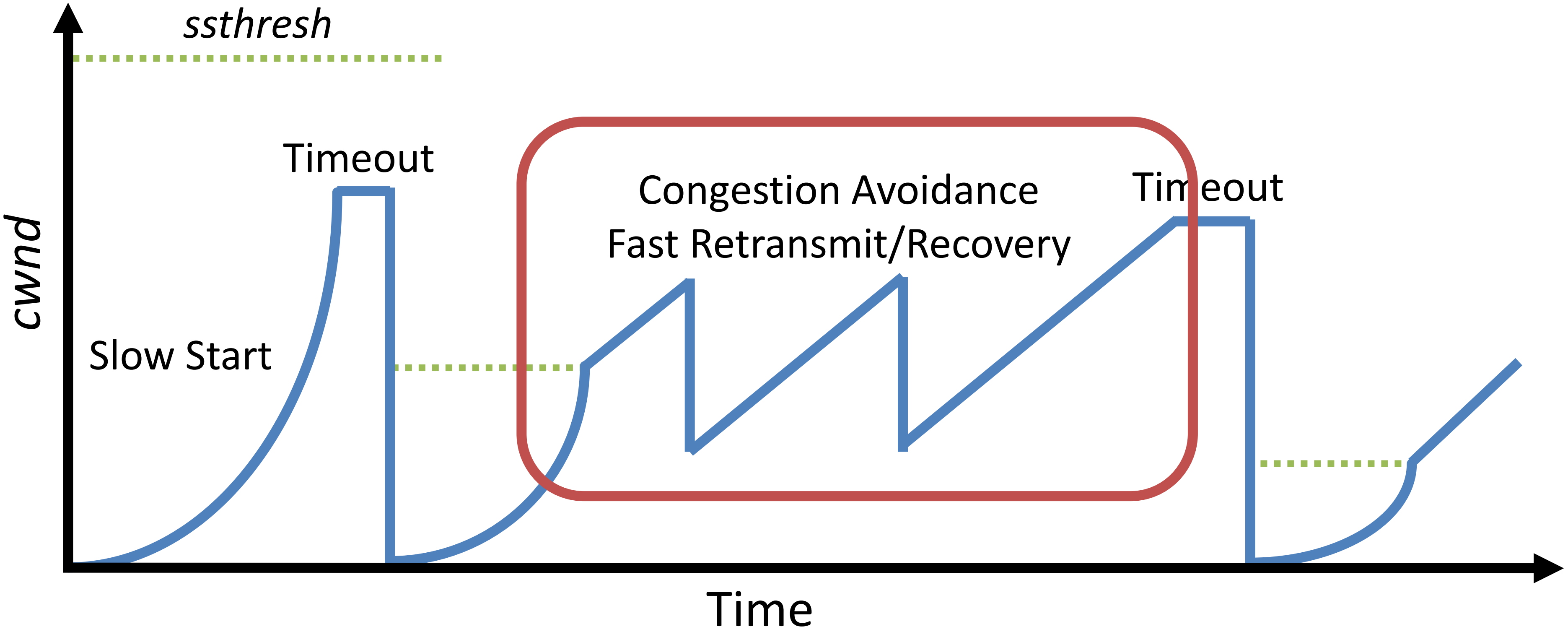
- Objective: prevent `pipe’ from emptying after fast retransmit
  - each dup ACK represents a packet having left the pipe (successfully received)
  - Let’s enter the “FR/FR” mode on 3 dup ACKs

```
ssthresh $\leftarrow$ W/2
retransmit lost packet
W $\leftarrow$ ssthresh + ndup (window inflation)
Wait till W is large enough; transmit new packet(s)
On non-dup ACK (1 RTT later)
    W $\leftarrow$ ssthresh (window deflation)
    enter CA mode
```

TCP Reno: Summary

- Fast Recovery along with Fast Retransmit used to avoid slow start
- On 3 duplicate ACKs
  - Fast retransmit and fast recovery
- On timeout
  - Fast retransmit and slow start

Fast Retransmit and Fast Recovery



- At steady state, *cwnd* oscillates around the optimal window size
- TCP always forces packet drops

TCP Throughput

- What’s the average throughout ot TCP as a function of window size and RTT?
  - Ignore slow start
- Let W be the window size when loss occurs.
- When window is W, throughput is W/RTT
- Just after loss, window drops to W/2, throughput to W/2RTT.
- Average throughout: .75 W/RTT

TCP Futures

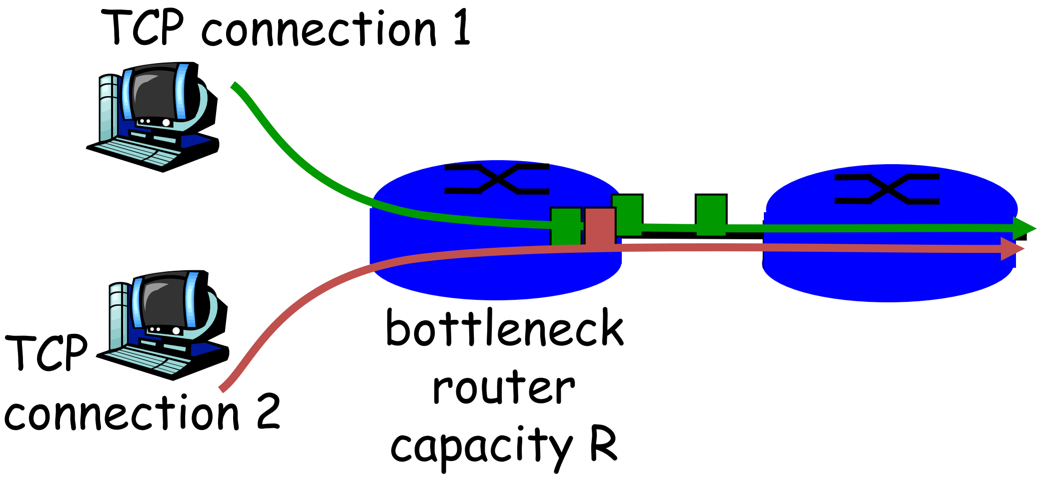
- Example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- Requires window size W = 83,333 in-flight segments
- Throughput in terms of loss rate:

$$\frac{1.22 \cdot MSS}{RTT\sqrt{L}}$$

- ➜ L = $2 \cdot 10^{-10}$ *Wow*
- New versions of TCP for high-speed needed!

TCP Fairness

Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



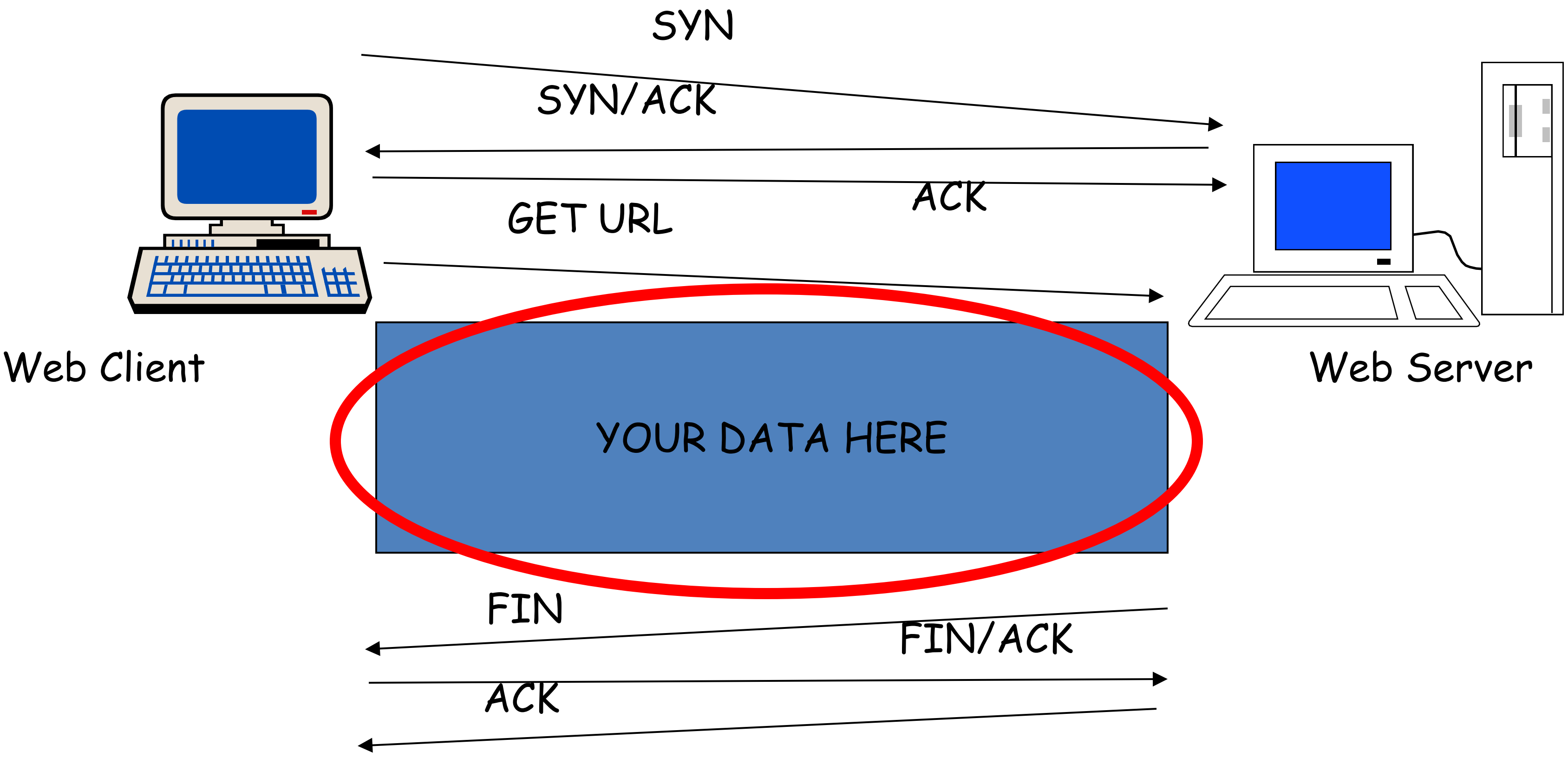
Fairness (more)

- TCP fairness: dependency on RTT
  - Connections with long RTT get less throughput
- Parallel TCP connections
- TCP friendliness for UDP streams
  - Similar throughput (and behavior) as TCP; e.g.,

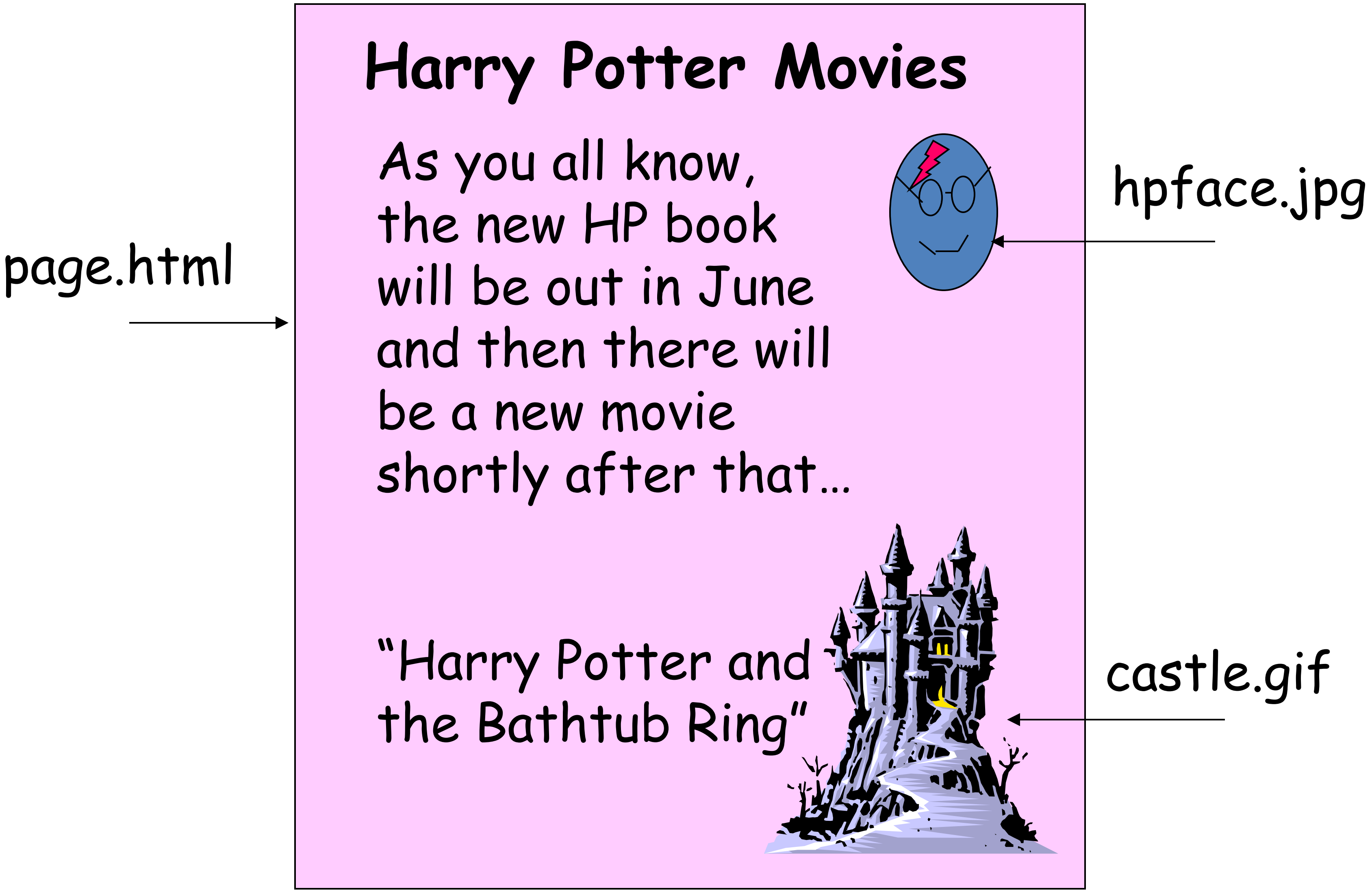
$$throughput \propto \frac{1}{RTT\sqrt{L}}$$

TCP+HTTP refresher

- TCP is a connection-oriented protocol



Example Web Page



Harry Potter Movies

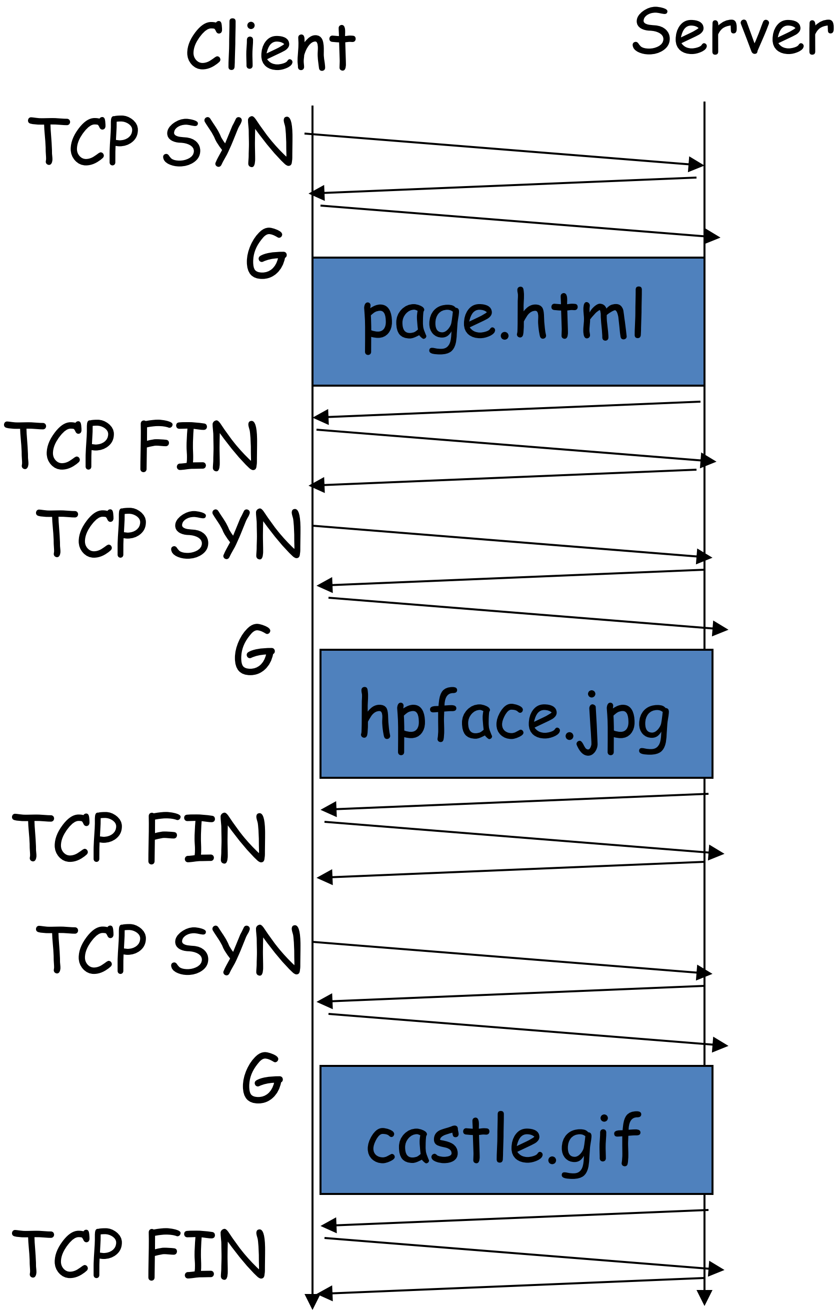
As you all know,
the new HP book
will be out in June
and then there will
be a new movie
shortly after that…

“Harry Potter and
the Bathtub Ring”

page.html

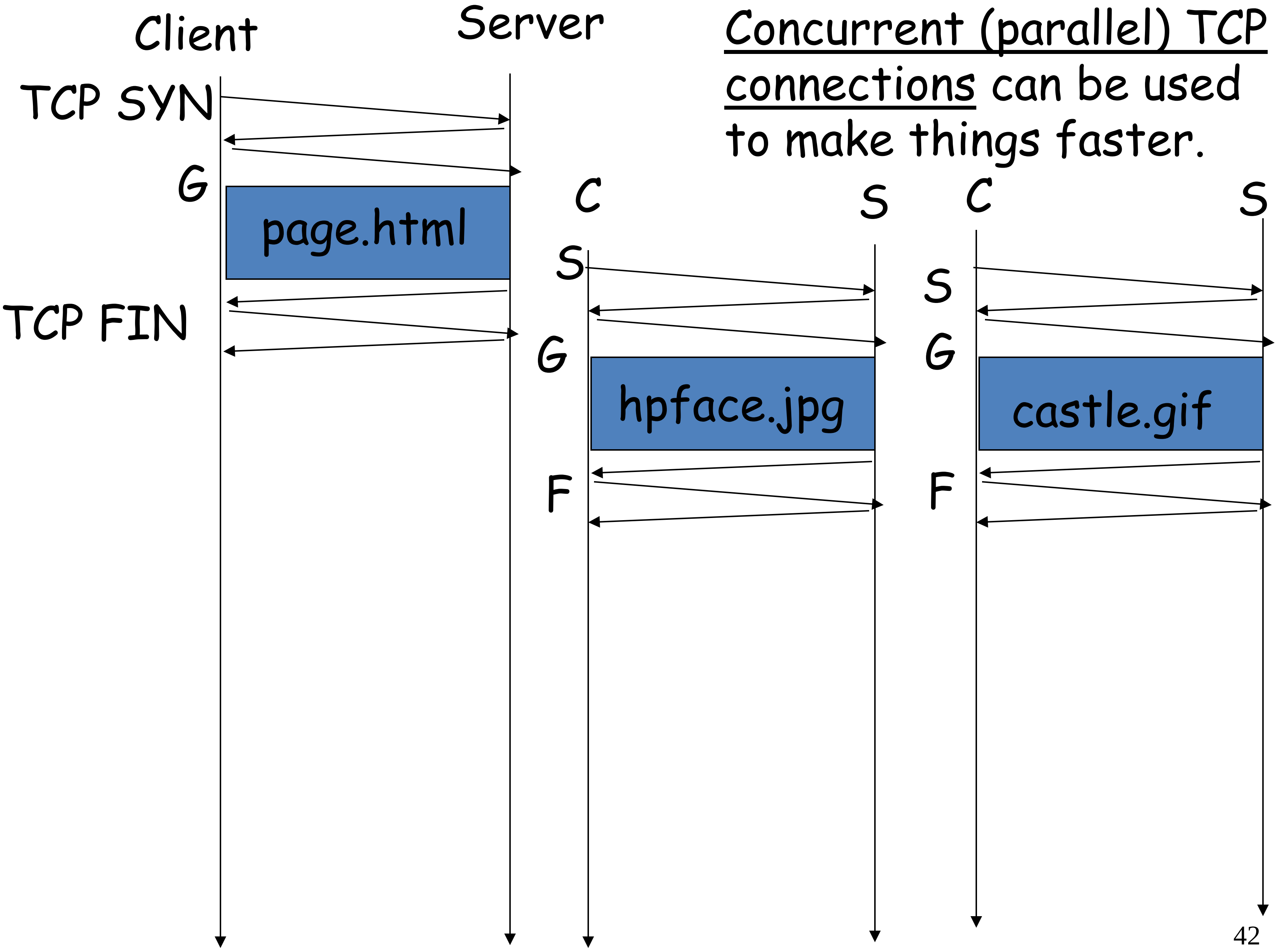
hpface.jpg

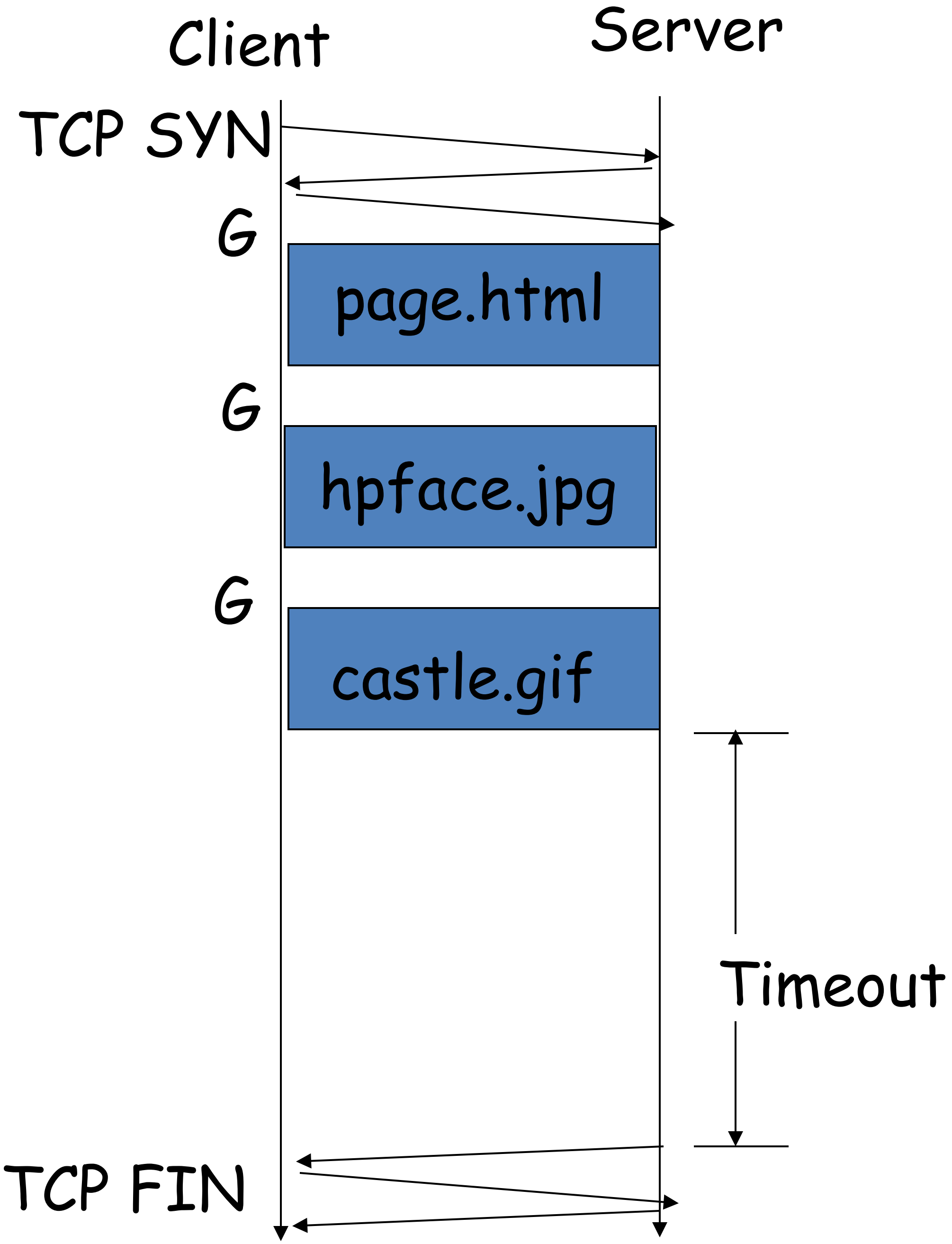
castle.gif



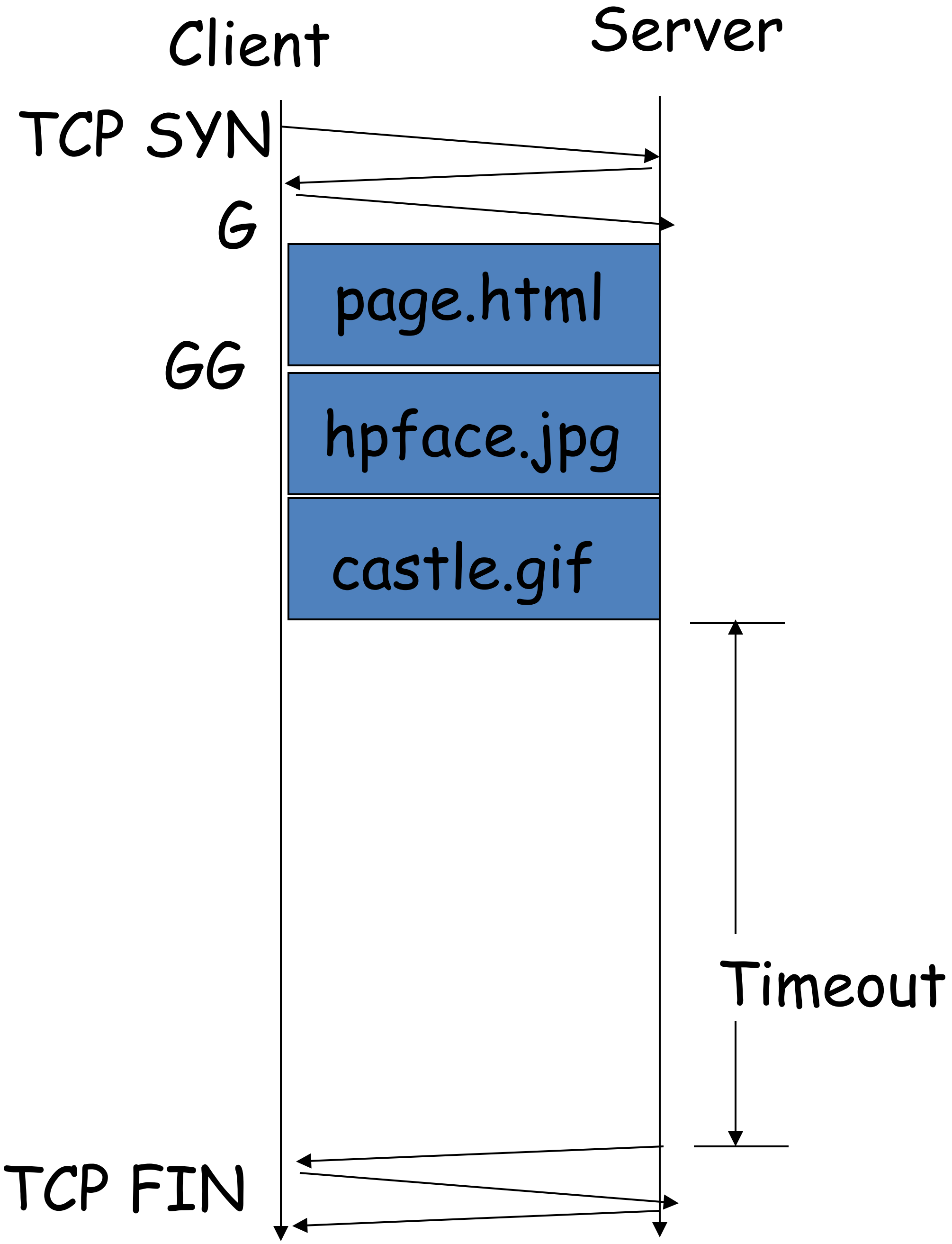
The “classic” approach in HTTP/1.0 is to use one HTTP request per TCP connection, serially.

Concurrent (parallel) TCP connections can be used to make things faster.





The "persistent HTTP" approach can re-use the same TCP connection for Multiple HTTP transfers, one after another, serially Amortizes TCP overhead, but maintains TCP state longer at server.



The "pipelining" feature in HTTP/1.1 allows requests to be issued asynchronously on a persistent connection. Requests must be processed in proper order. Can do clever packaging.