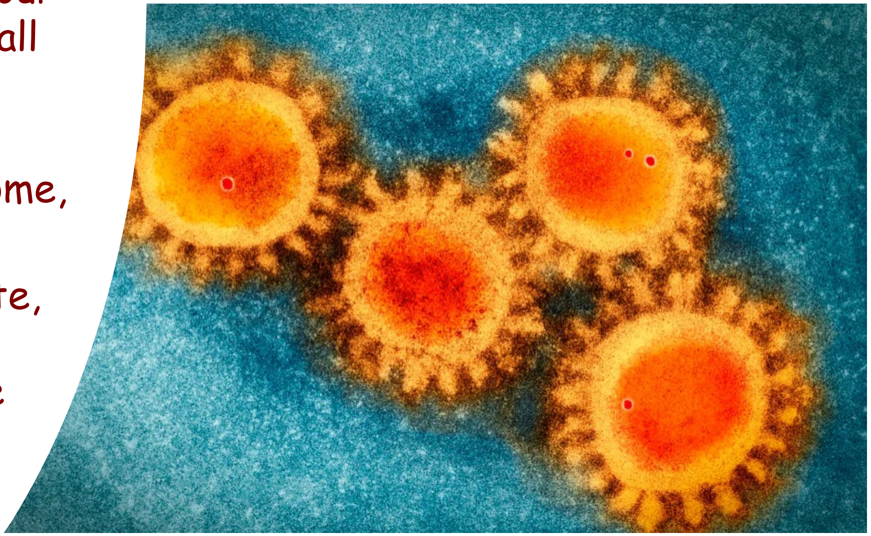
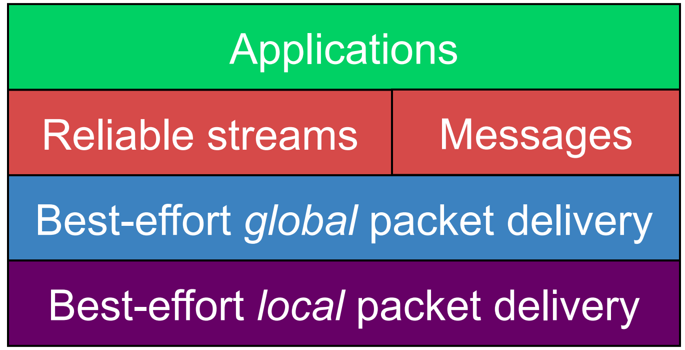
Classroom Protocol: Masks Required

- Wear your mask correctly, over your nose and mouth. **Extras are available from course staff**
- Lifting the mask to take sips of a beverage is permitted. Please keep your mask on over your nose and mouth at all other times.
- If you don't feel well or have a runny nose, sore throat, etc., please stay home, we will work with you.
- If you test positive and need to isolate, please contact me to confirm arrangements for keeping up with the class.





Class Meeting, Lectures 5 & 6:
Transport Layer & Congestion Control

Kyle Jamieson
COS 461: Computer Networks

[Parts adapted from material by M. Freedman (Princeton), B. Karp (UCL), D. Katabi, (MIT), S. Shenker (UCB)]

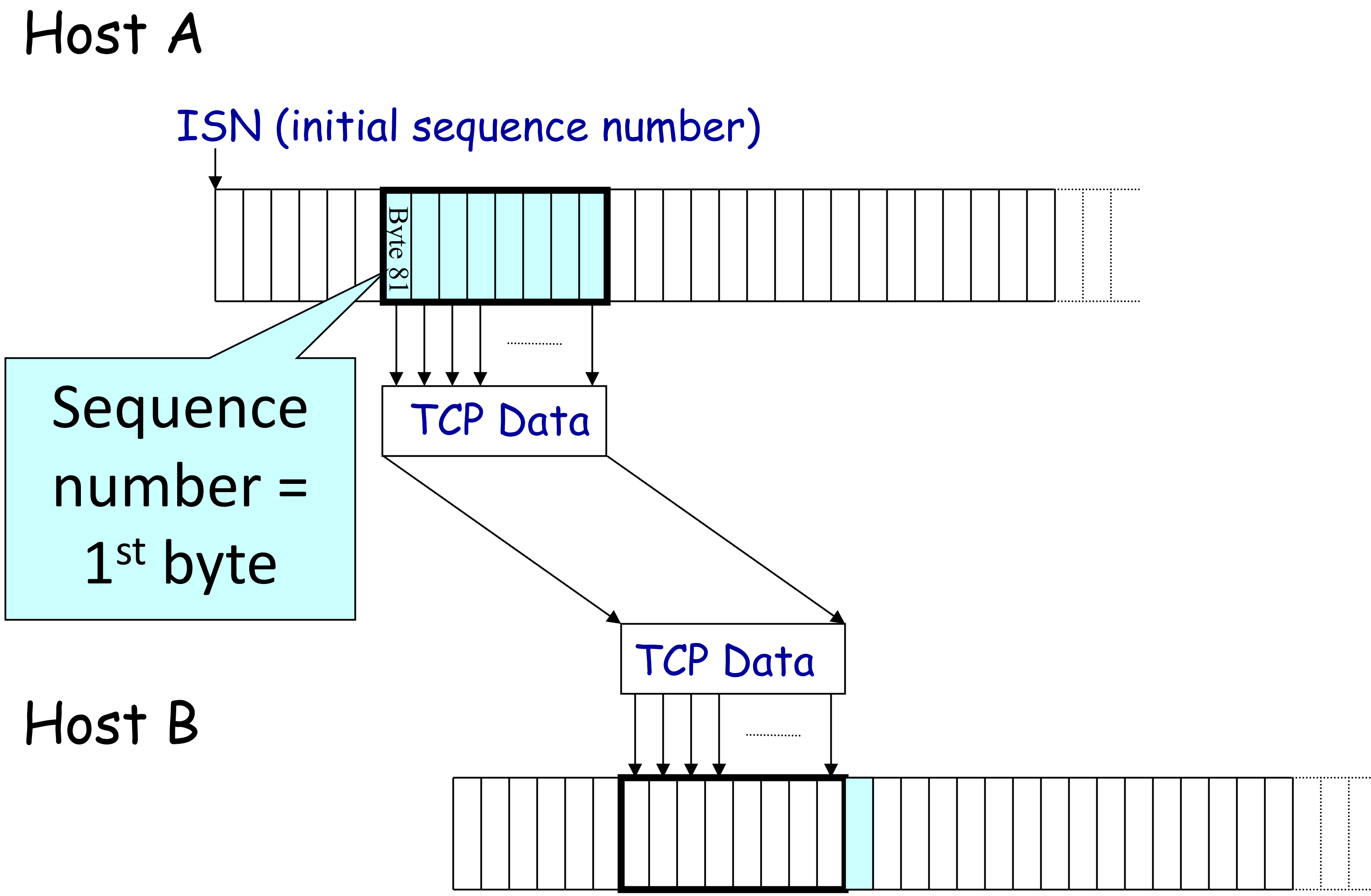
Context: Transport Layer

- Best-effort network layer
  - drops packets
  - delays packets
  - reorders packets
  - corrupts packet contents
- Many applications want reliable transport
  - all data reach receiver, in order they were sent
  - no data corrupted
  - "reliable byte stream"
- Need a transport protocol, *e.g.*, Internet's Transmission Control Protocol (TCP)

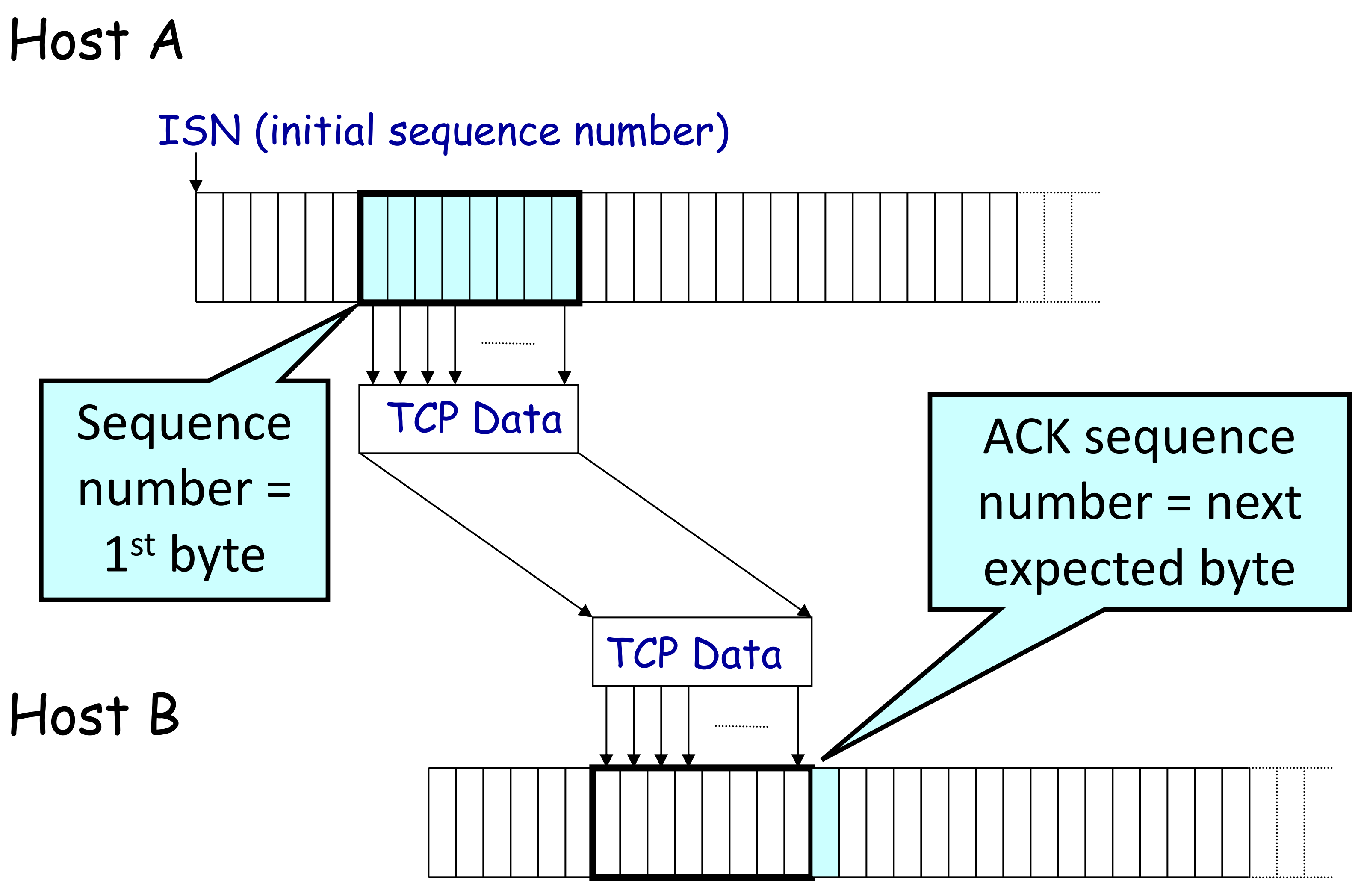
TCP: Connection-Oriented, Reliable Byte Stream Transport

- Sending app offers stream of bytes: d0, d1, d2, …
- Receiving app sees all bytes in same order: d0, d1, d2…
  - Result: reliable byte stream transport
    - But: not all applications need in-order behavior
- Each byte stream: *connection*, or *flow*
- Each connection uniquely identified by:
  - <sender IP, sender port, receiver IP, receiver port>

Sequence Numbers in TCP: Data

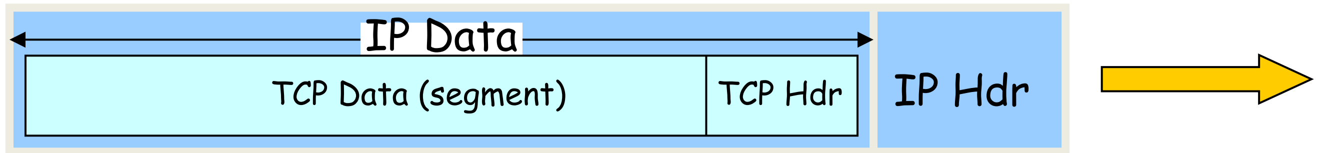


Sequence Numbers in TCP: ACKs



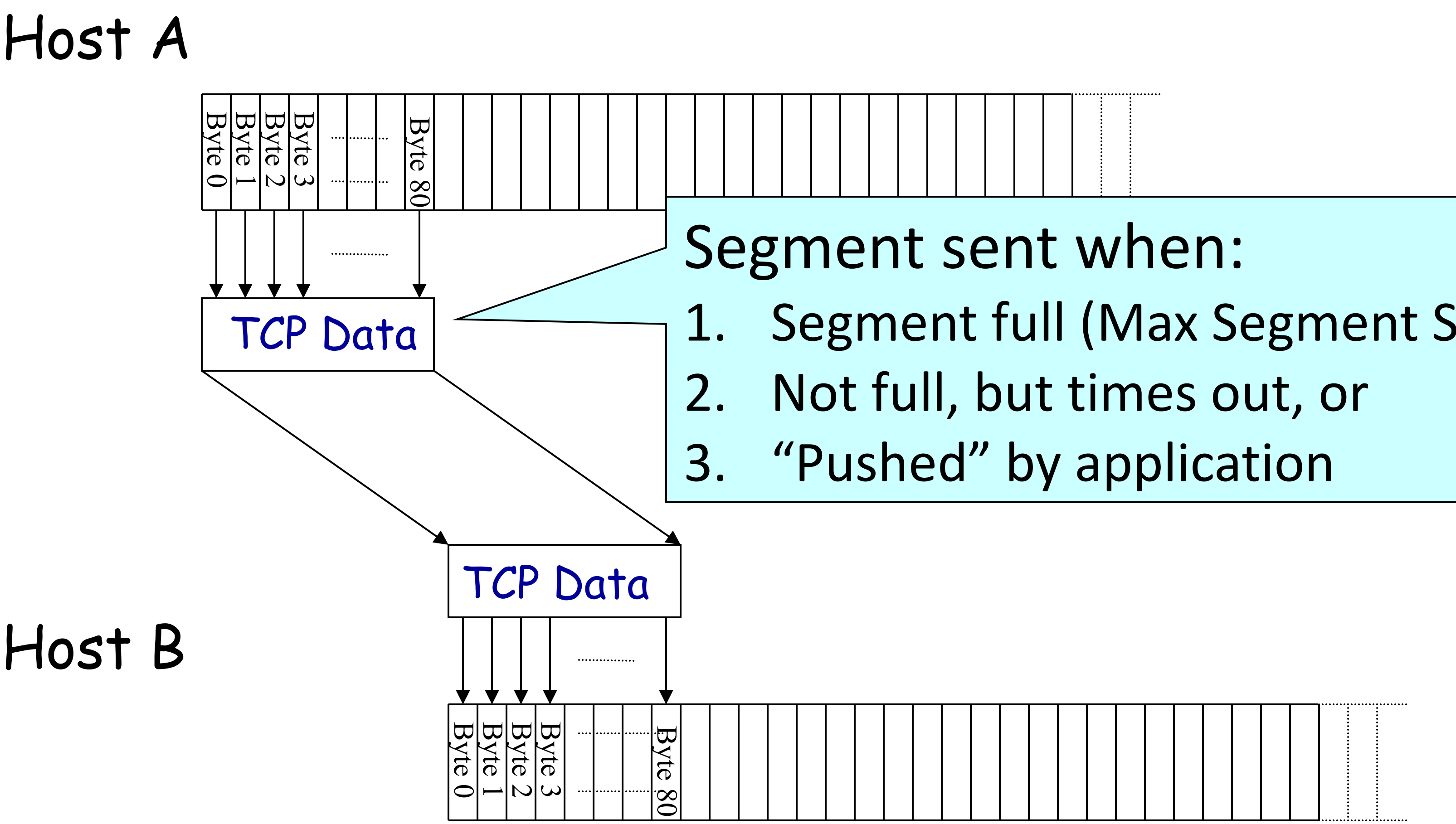
TCP Segment

- IP packet



  - No bigger than Maximum Transmission Unit (MTU)
  - E.g., up to 1500 bytes on an Ethernet link
- TCP packet
  - IP packet with a TCP header and data inside
  - TCP header is typically 20 bytes long
- TCP packet contents (i.e. *segment*)
  - No more than Maximum Segment Size (MSS) bytes
  - E.g., up to 1460 consecutive bytes from the stream: MTU (1500) - IP header (20) - TCP header (20)

…Emulated Using TCP "Segments"



Segment sent when:

1. Segment full (Max Segment Size),
2. Not full, but times out, or
3. "Pushed" by application

Quick TCP Math

- Initial Seq No = 501. Sender sends 4500 bytes successfully acknowledged. Next sequence number to send is:

  (Y) 5000 (M) 5001 (C) 5002

- Next 1000 byte TCP segment received. Receiver acknowledges with ACK number:

  (Y) 5001 (M) 6000 (C) 6001

Quick TCP Math

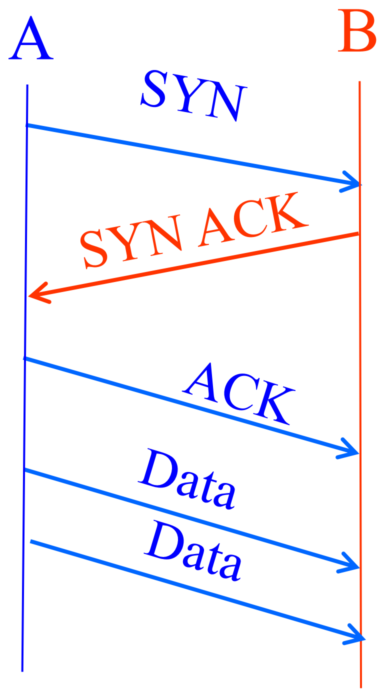
- Initial Seq No = 501. Sender sends 4500 bytes successfully acknowledged. Next sequence number to send is:

  (Y) 5000 $\boxed{\text{(M) 5001}}$ (C) 5002

- Next 1000 byte TCP segment received. Receiver acknowledges with ACK number:

  (Y) 5001 (M) 6000 $\boxed{\text{(C) 6001}}$

Establishing a TCP Connection



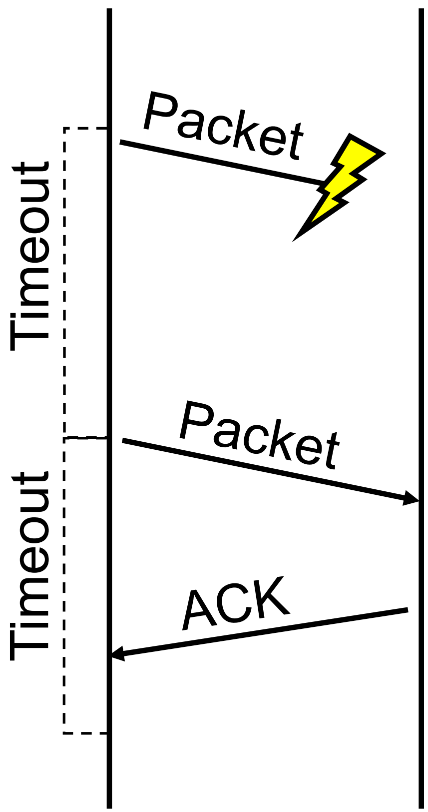
Each host tells its ISN to the other host.

- **Three-way handshake** to establish connection
  - Host A sends a SYN (open) to the host B
  - Host B returns a SYN acknowledgment (SYN ACK)
  - Host A sends an ACK to acknowledge the SYN ACK

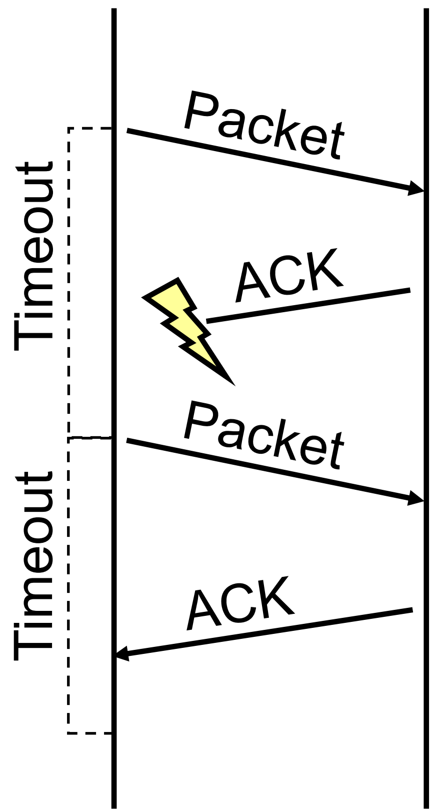
SYN Loss and Web Browsing

- Upon sending SYN, sender sets a timer
  - If SYN lost, timer expires before SYN-ACK received, sender retransmits SYN
- How should the TCP sender set the timer?
  - No idea how far away the receiver is
  - Some TCPs use default of 3 or 6 seconds
- Implications for loading a web page
  - User gets impatient and hits reload
  - … Users aborts connection, initiates new socket
  - Essentially, forces a fast send of a new SYN!

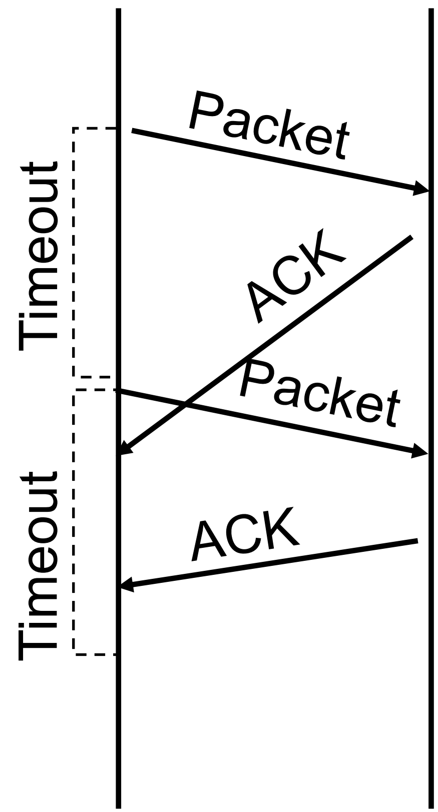
Reasons for Retransmission



Packet lost



ACK lost
DUPLICATE PACKET



Early timeout
DUPLICATE PACKETS

How Long Should Sender Wait?

- Sender sets a timeout to wait for an ACK
  - Too short: wasted retransmissions
  - Too long: excessive delays when packet lost
- TCP sets timeout as a function of the RTT
  - Expect ACK to arrive after an "round-trip time"
  - … plus a fudge factor to account for queuing
- But, how does the sender know the RTT?
  - Running average of delay to receive an ACK

Still, timeouts are slow (≈RTT)

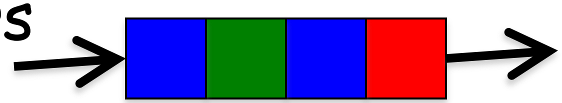
- When packet n is lost…
  - … packets n+1, n+2, and so on may get through
- Exploit the ACKs of these packets
  - ACK says receiver is still awaiting nth packet
  - Duplicate ACKs suggest later packets arrived
  - **Sender uses "duplicate ACKs" as a hint**
- **Fast retransmission**
  - **Retransmit after "triple duplicate ACK"**

Effectiveness of Fast Retransmit

- When does Fast Retransmit work best?
  - High likelihood of many packets in flight
  - Long data transfers, large window size, …
- Implications for Web traffic
  - Many Web transfers are short (e.g., 10 packets)
    - So, often there aren't many packets in flight
  - Making fast retransmit is less likely to "kick in"
    - Forcing users to click "reload" more often…

Network Congestion: Context

- Best-effort network does not "block" calls
  - So, they can easily become overloaded
  - Congestion == "Load higher than capacity"
- Examples of congestion
  - Link layer: Ethernet frame collisions
  - Network layer: full IP packet buffers

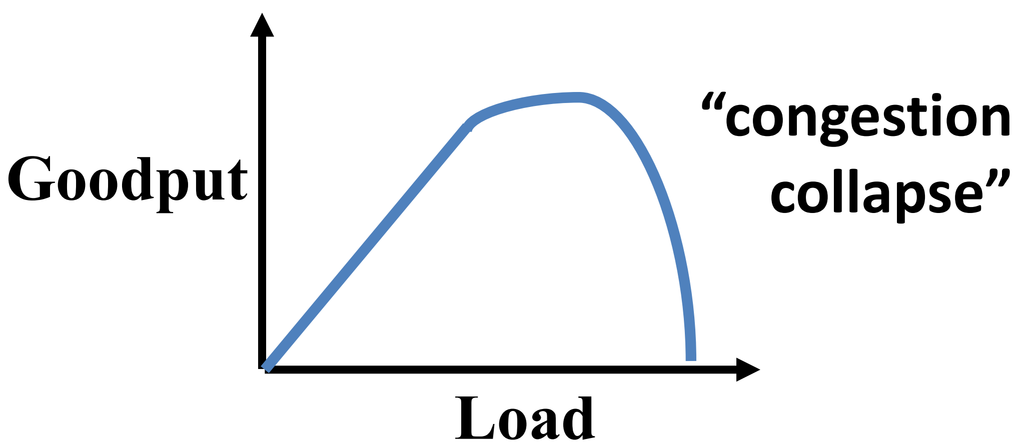


- Excess packets are simply dropped
  - And the sender can simply retransmit

queue

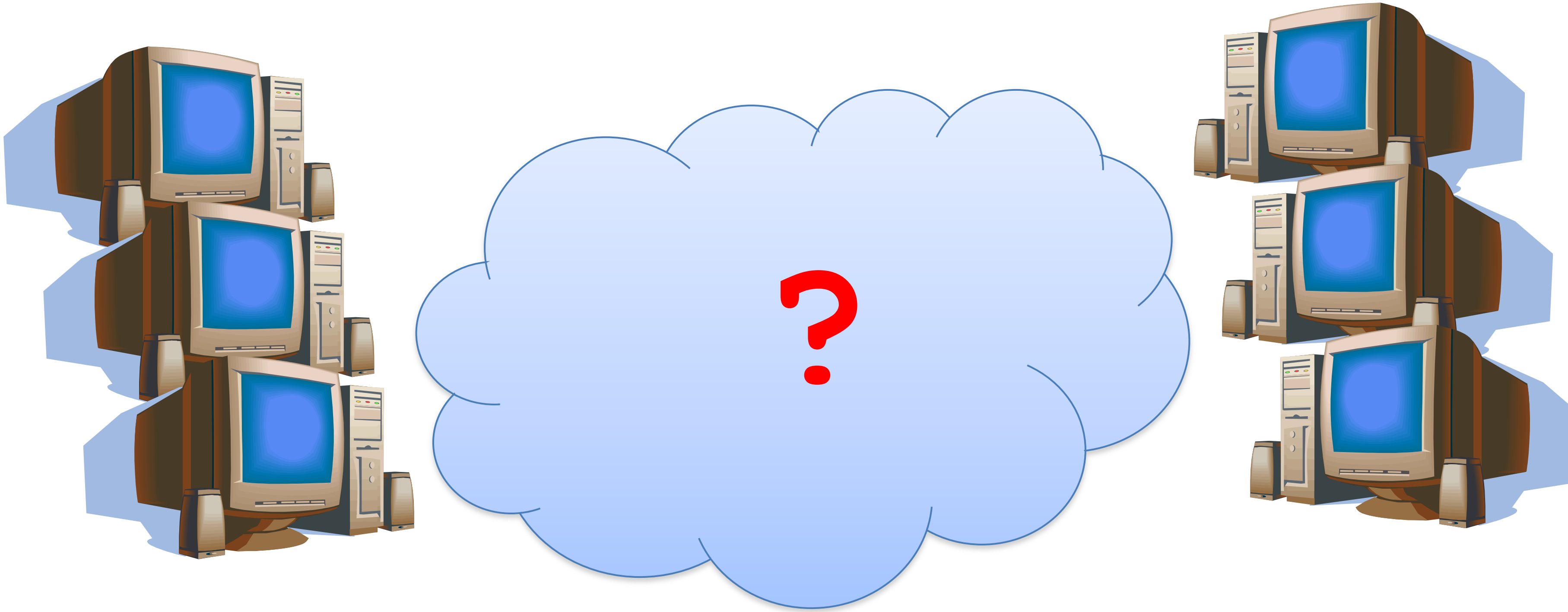
Problem: Congestion Collapse

- Network can undergo *congestion collapse*
  - Senders retransmit the lost packets
  - Leading to even *greater* load
  - … and even *more* packet loss



Increase in load that results in a *decrease* in useful work done.

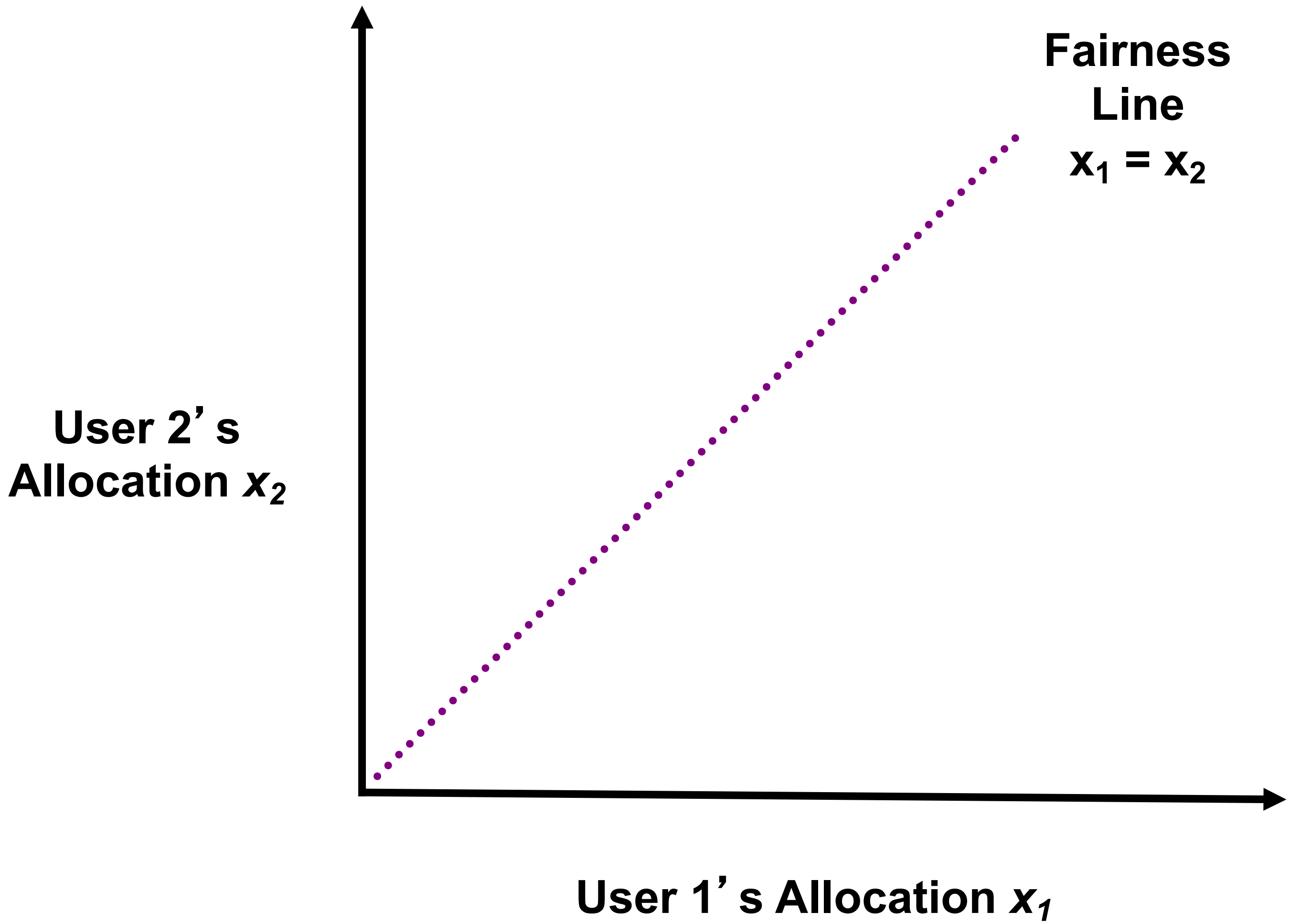
Detect and Respond to Congestion



- *What does the end host see?*
- *What can the end host change?*
- Distributed Resource Sharing

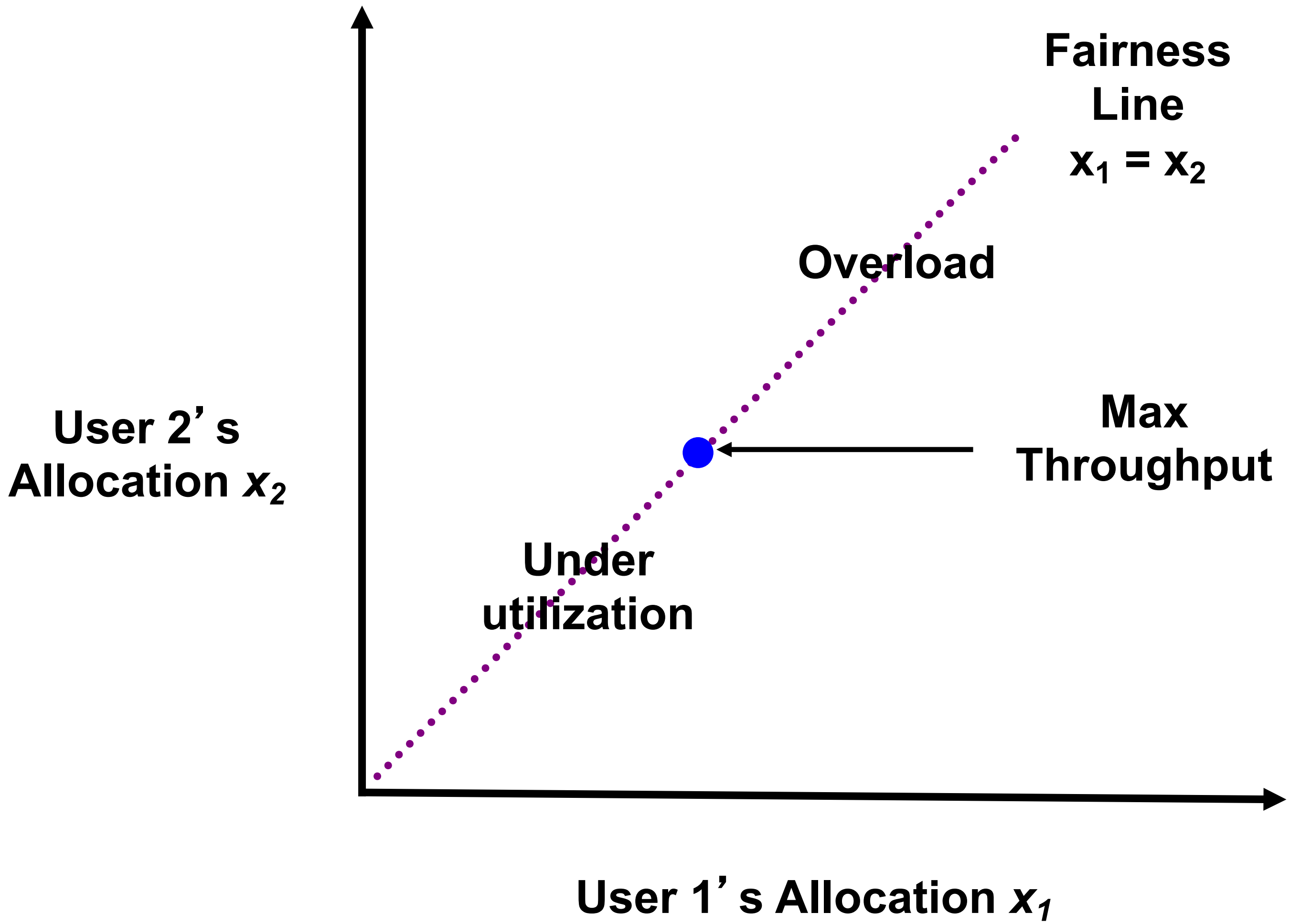
TCP seeks “Fairness”

Phase Plots



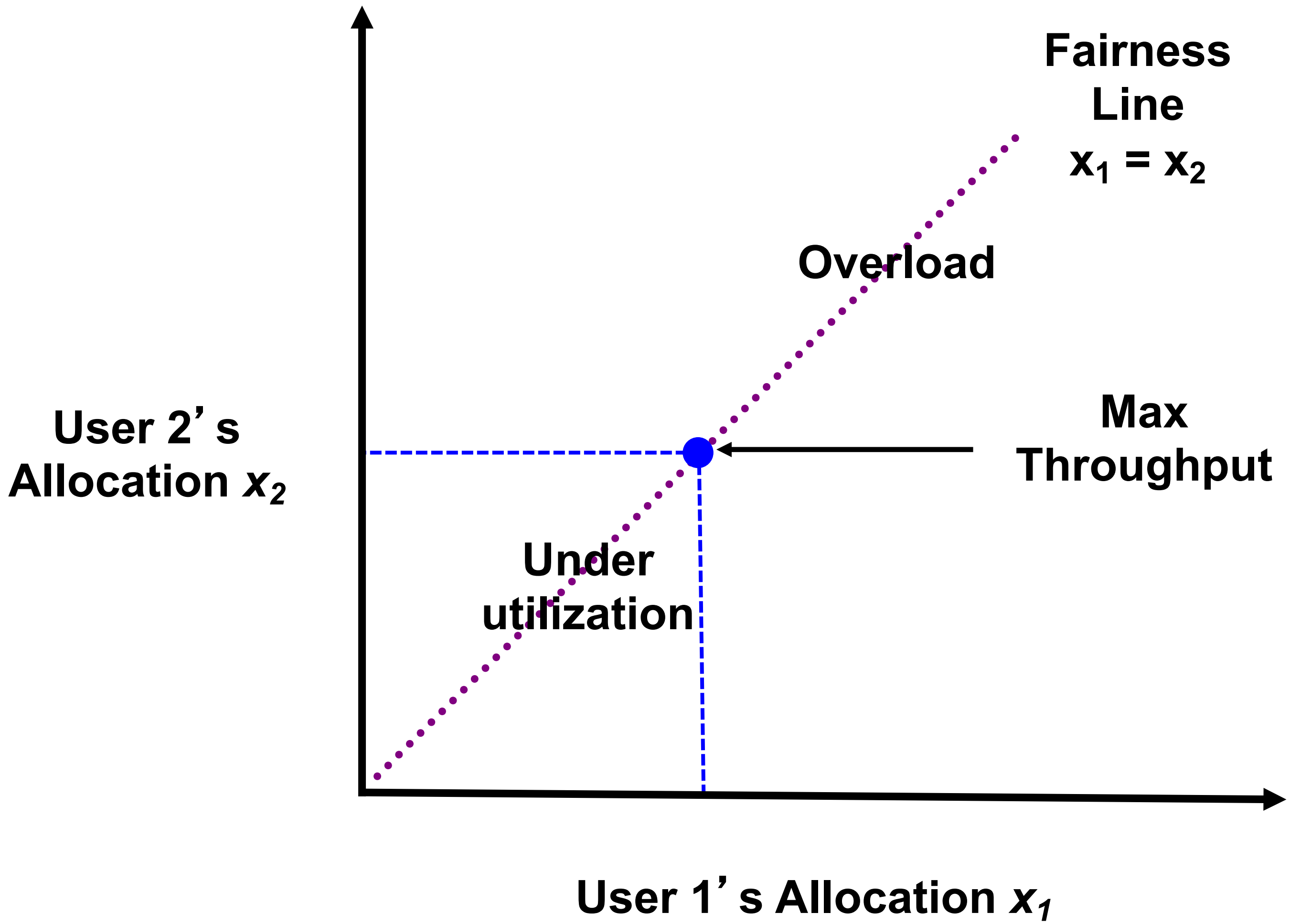
Fairness
Line
$x_1 = x_2$
User 2's
Allocation x_2
User 1's Allocation x_1

Phase Plots



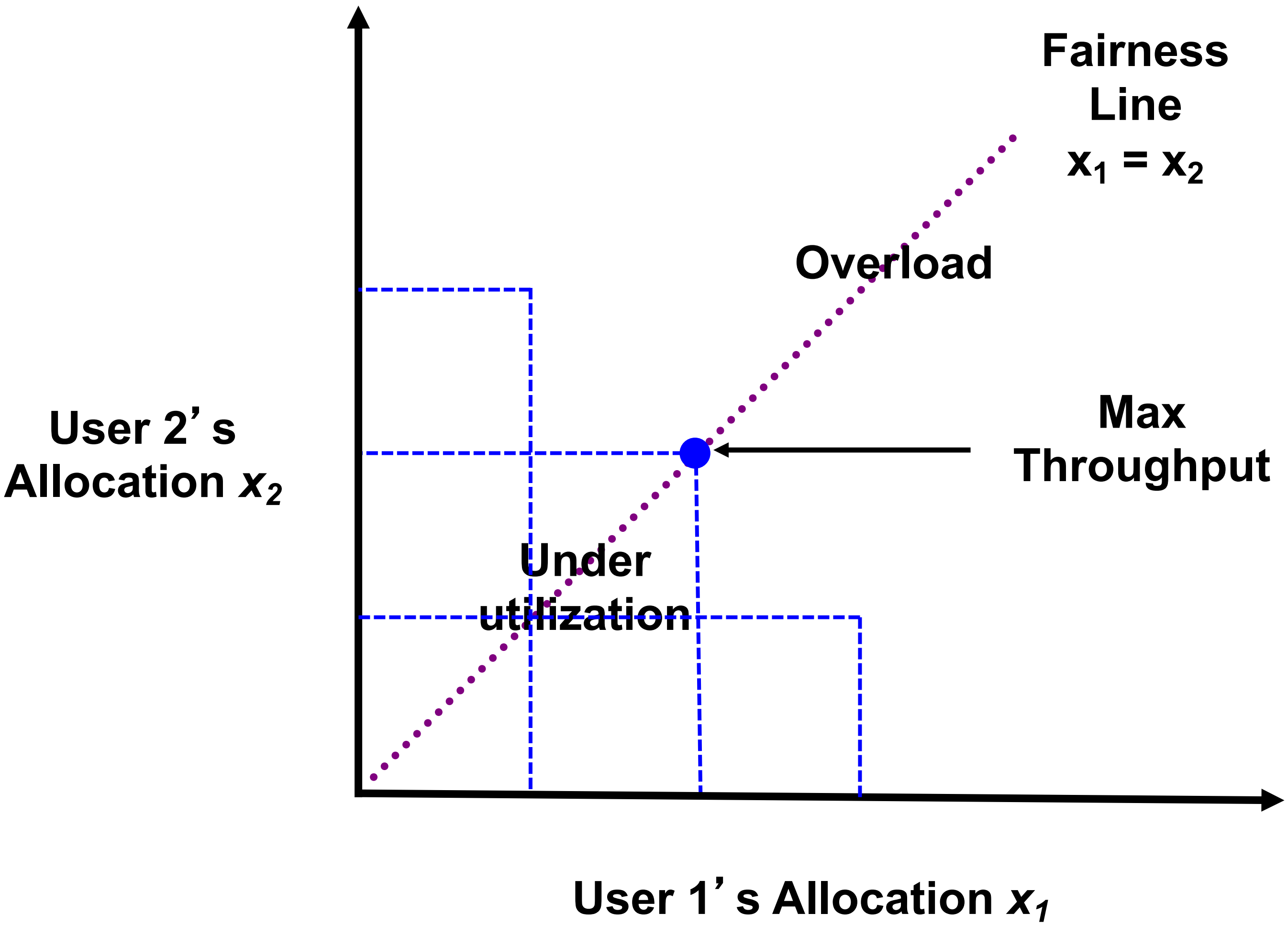
Fairness
Line
$x_1 = x_2$
Overload
User 2's
Allocation x_2
Max
Throughput
Under
utilization
User 1's Allocation x_1

Phase Plots

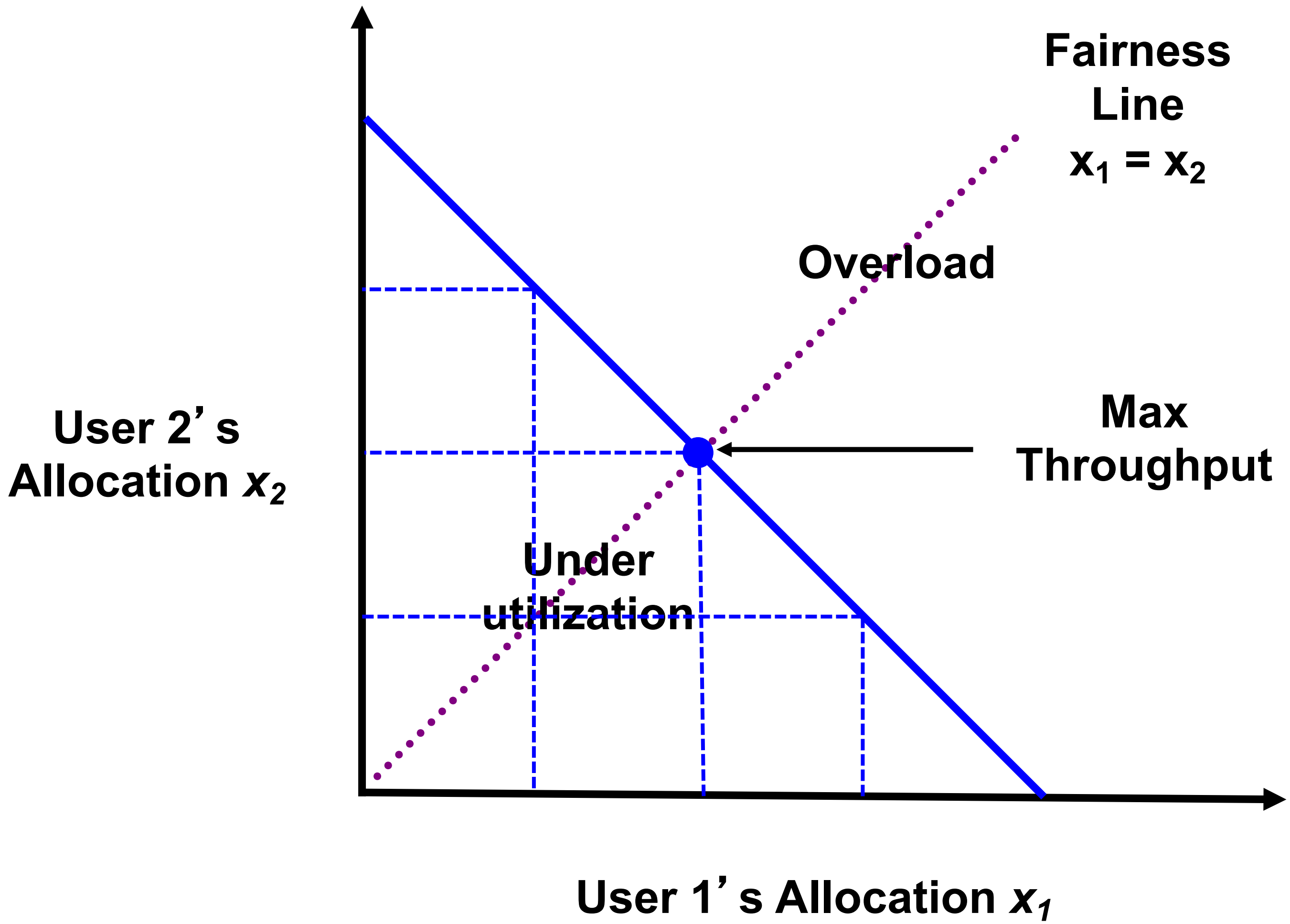


Fairness Line
$x_1 = x_2$
Overload
Max Throughput
User 2's Allocation x_2
Under utilization
User 1's Allocation x_1

Phase Plots

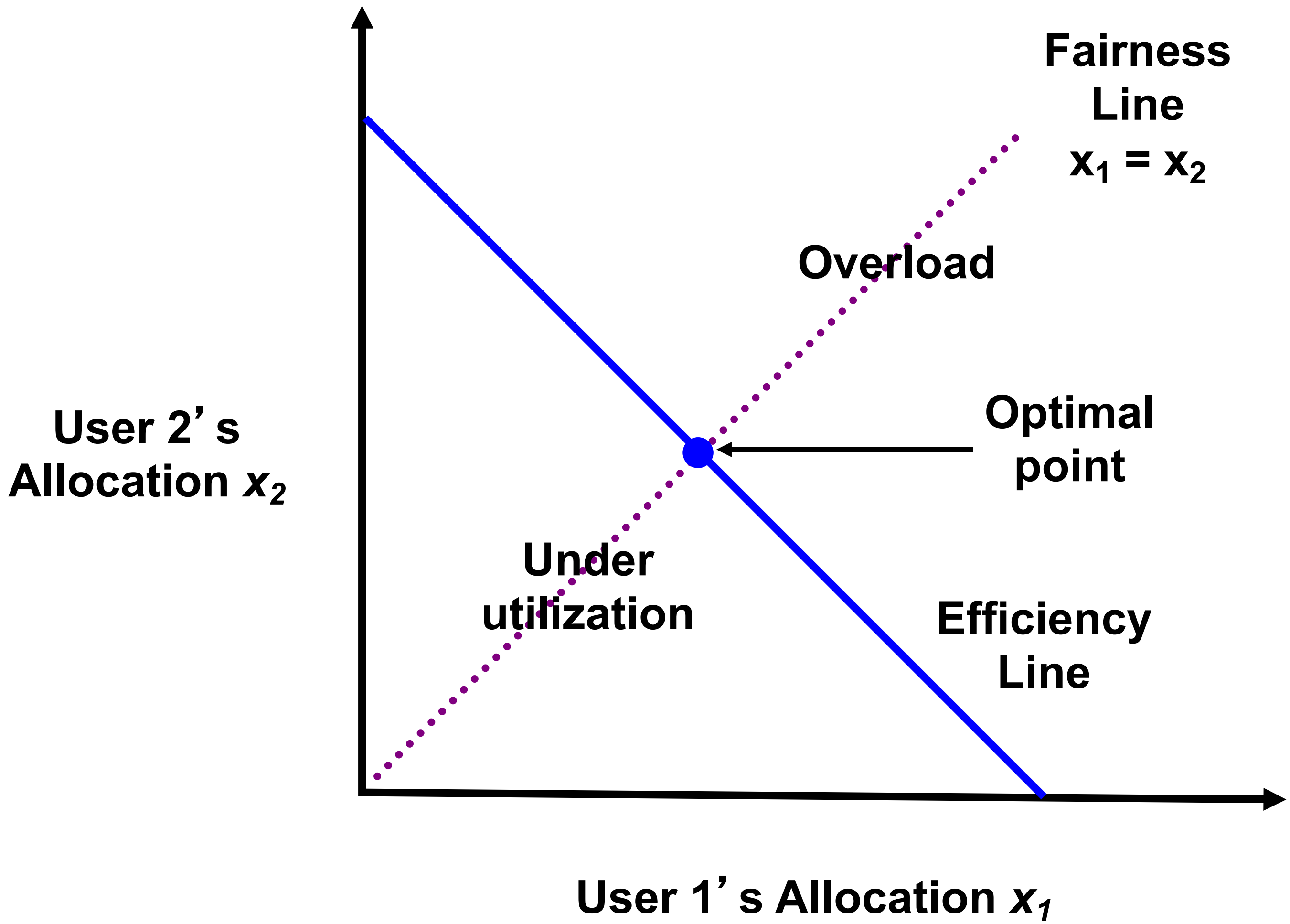

Fairness Line $x_1 = x_2$
Overload
User 2's Allocation x_2
Max Throughput
Under utilization
User 1's Allocation x_1

Phase Plots

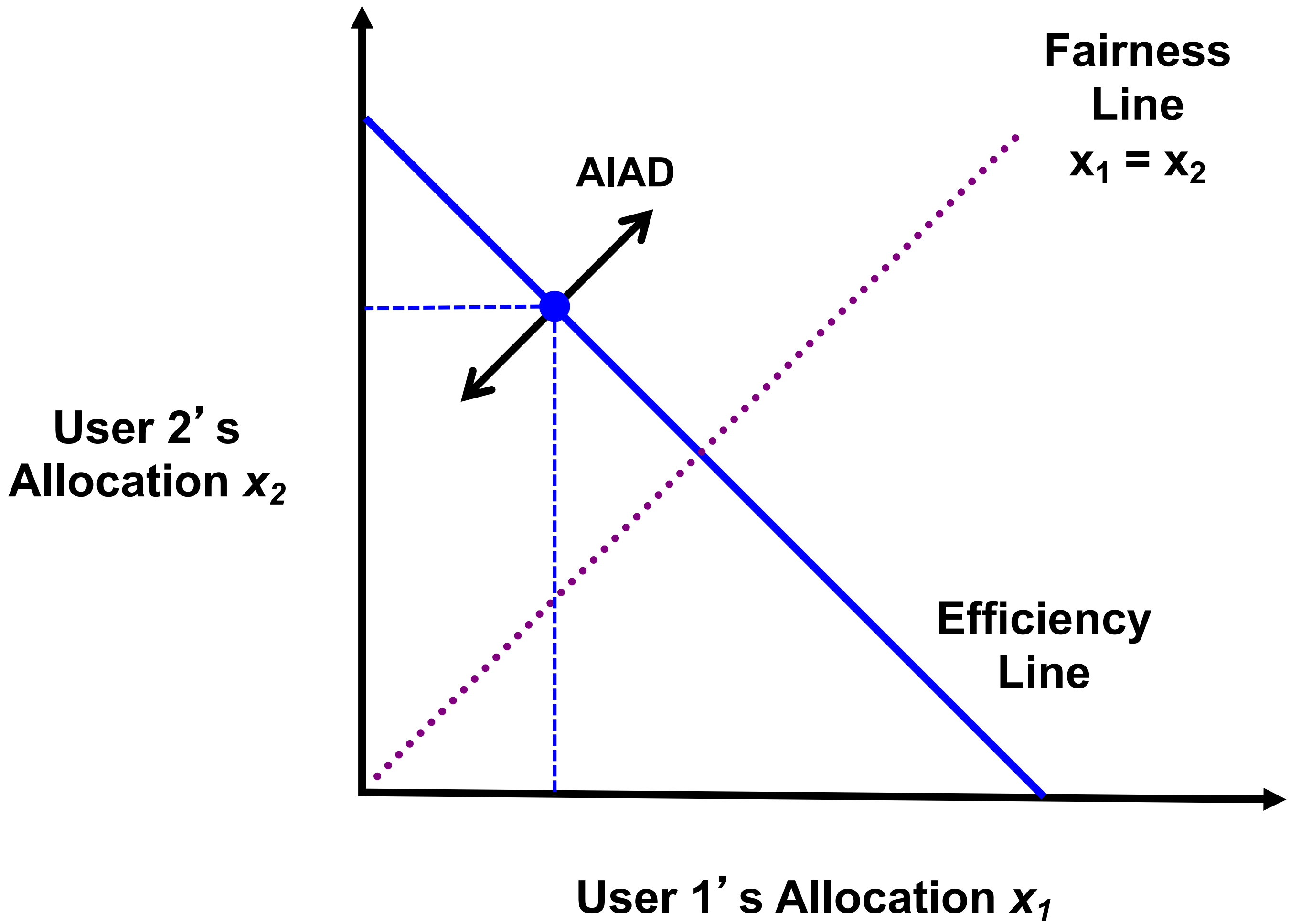


Fairness Line
$x_1 = x_2$
Overload
Max Throughput
Under utilization
User 2's Allocation x_2
User 1's Allocation x_1

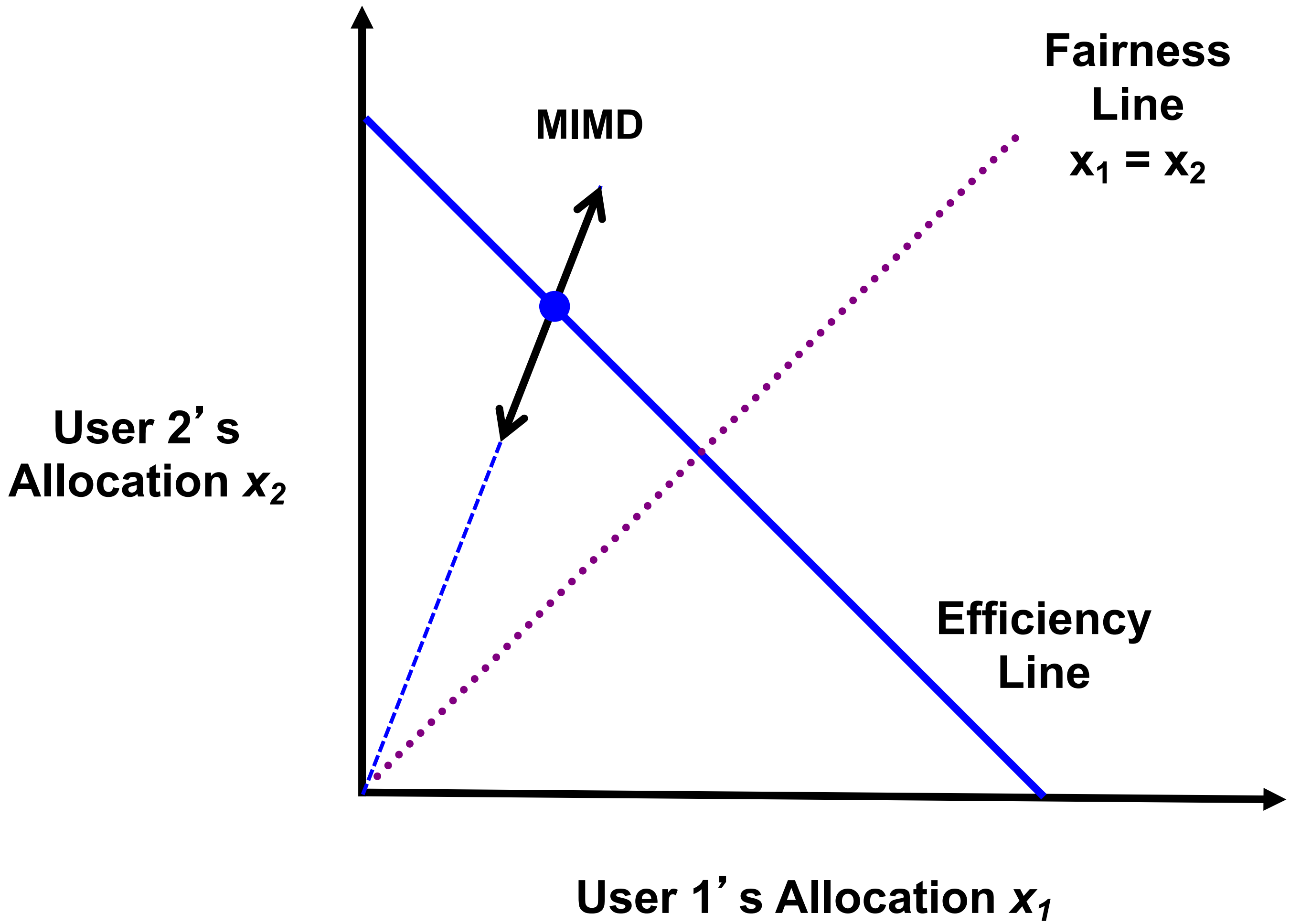
Phase Plots


Fairness
Line
$x_1 = x_2$
Overload
Optimal
point
User 2's
Allocation x_2
Under
utilization
Efficiency
Line
User 1's Allocation x_1

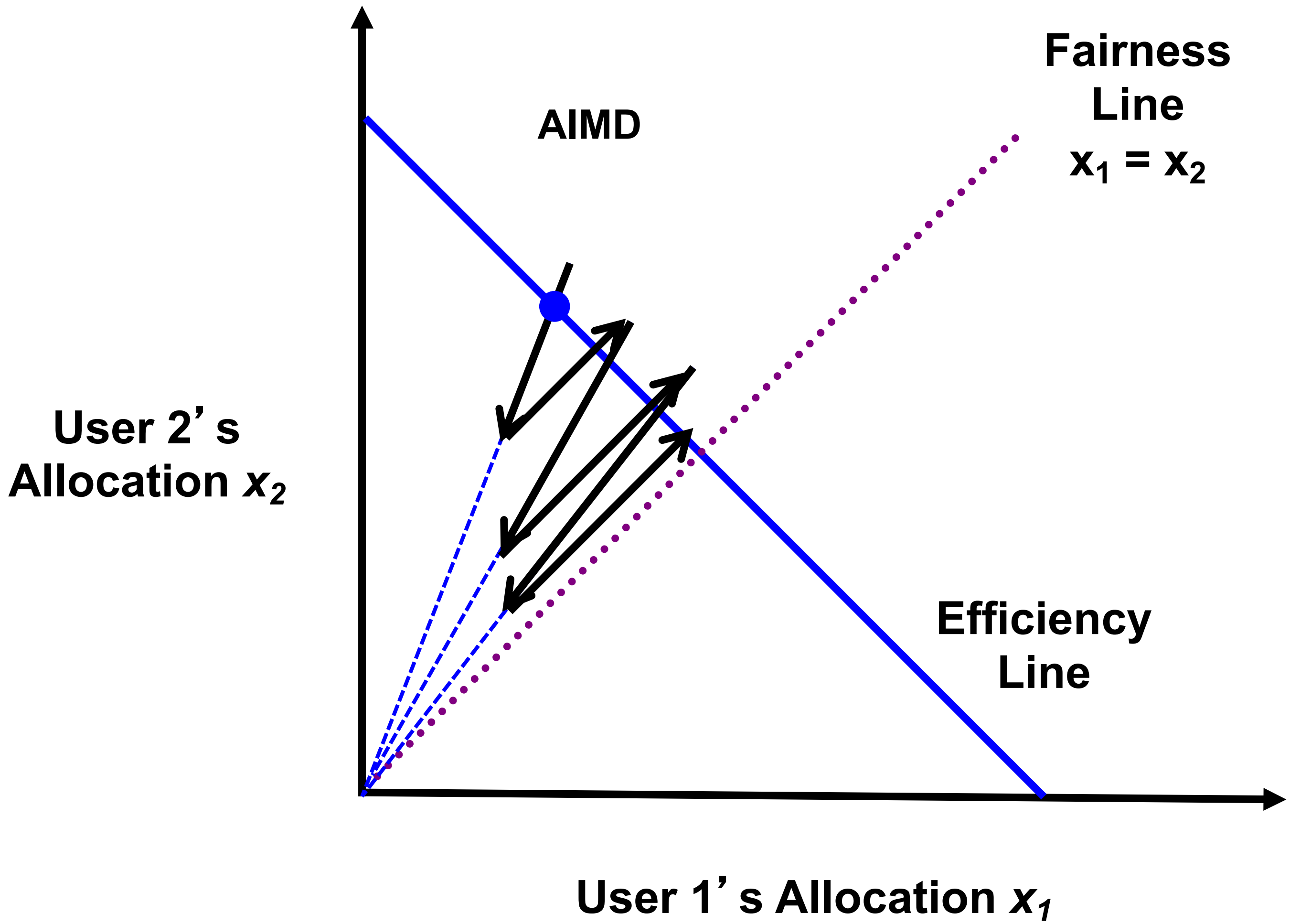
Additive Increase/Decrease



Multiplicative Increase/Decrease

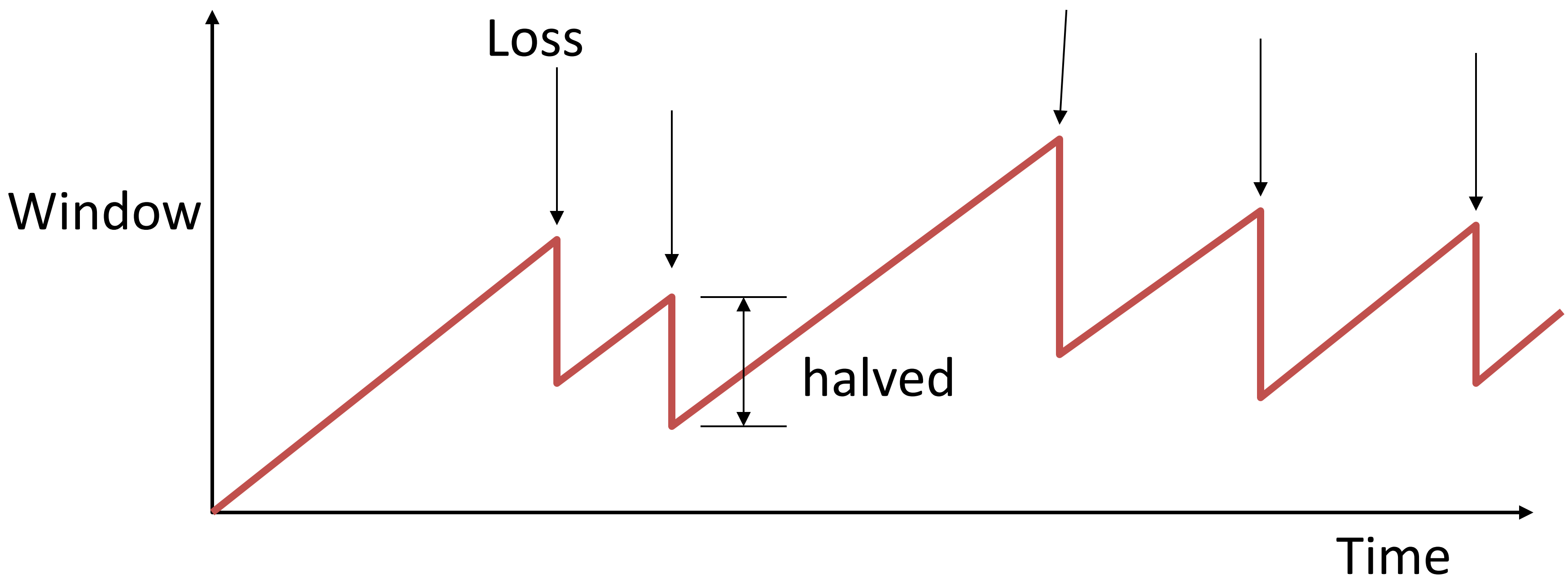


Additive Increase / Multiplicative Decrease



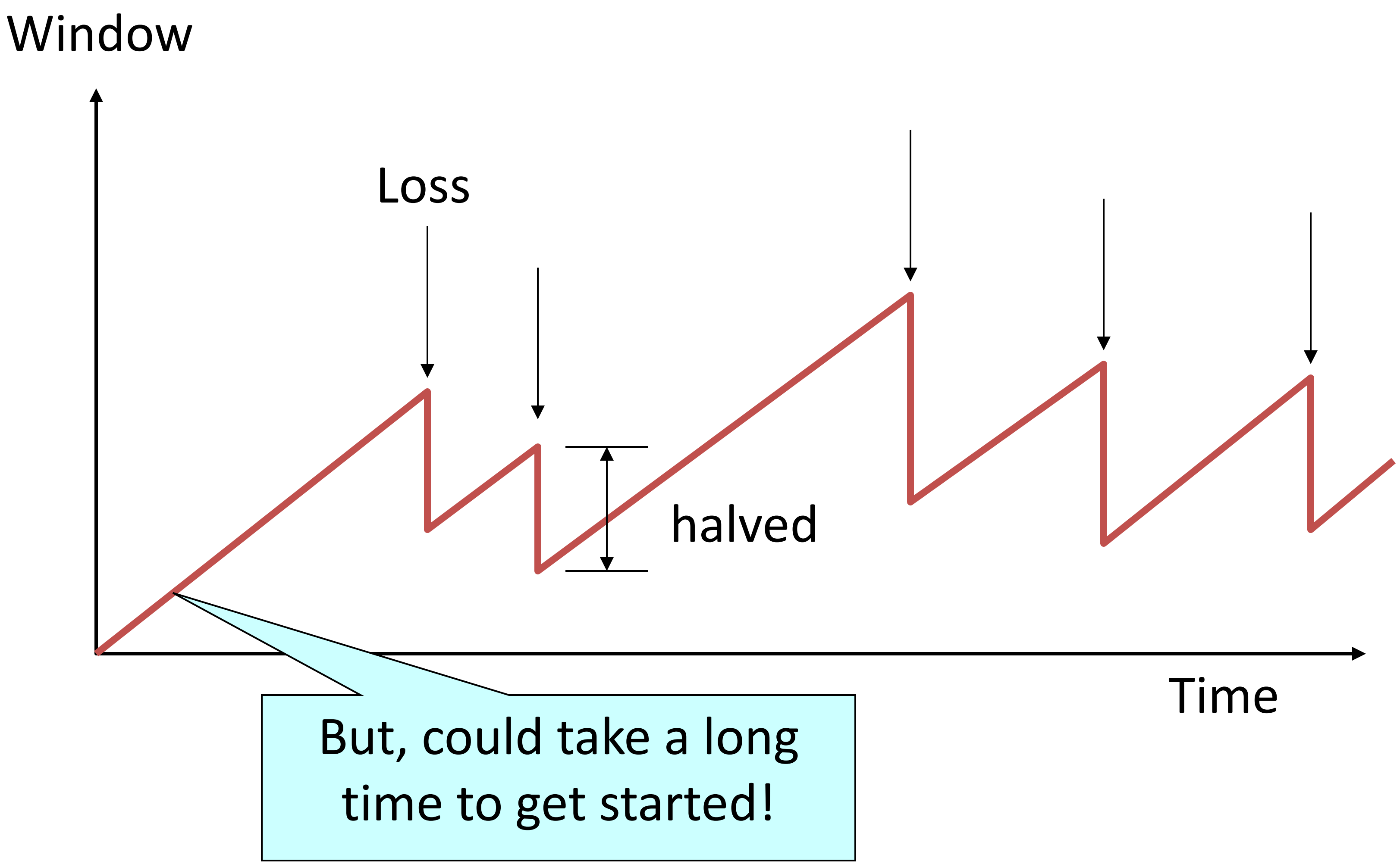
TCP Congestion Control

- Additive increase, multiplicative decrease
  - On packet loss, divide congestion window in half
  - On success for last window, increase window linearly



How Should a New Flow Start?

Start slow (a small CWND) to avoid overloading network

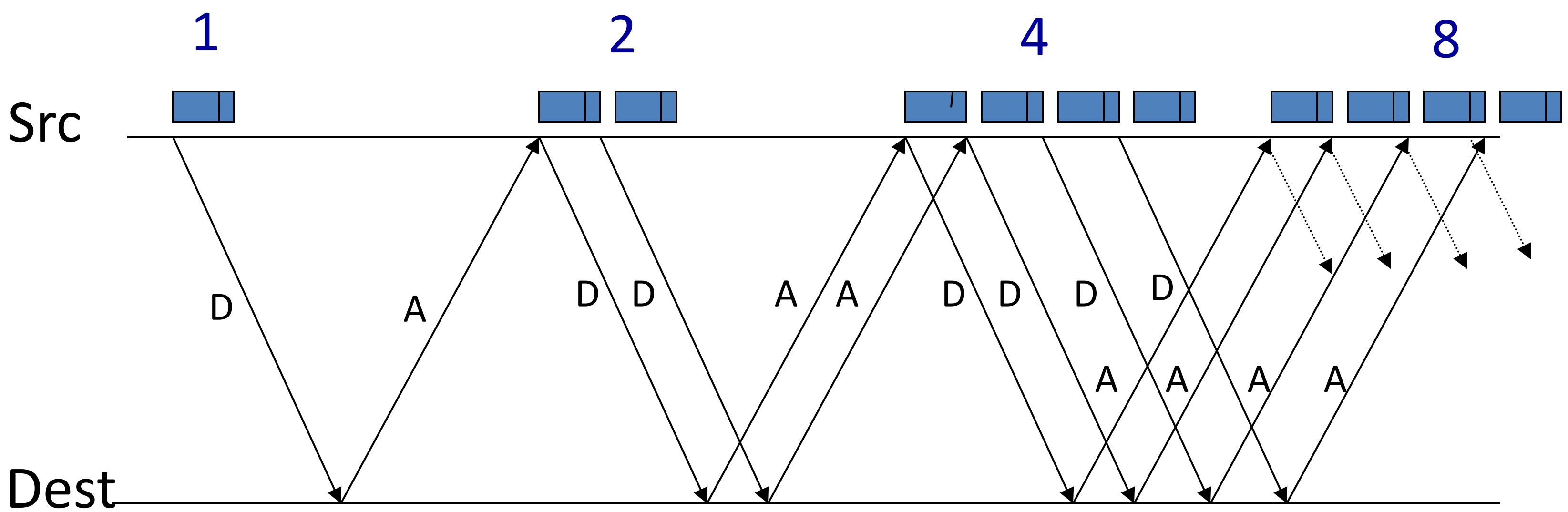


"Slow Start" Phase

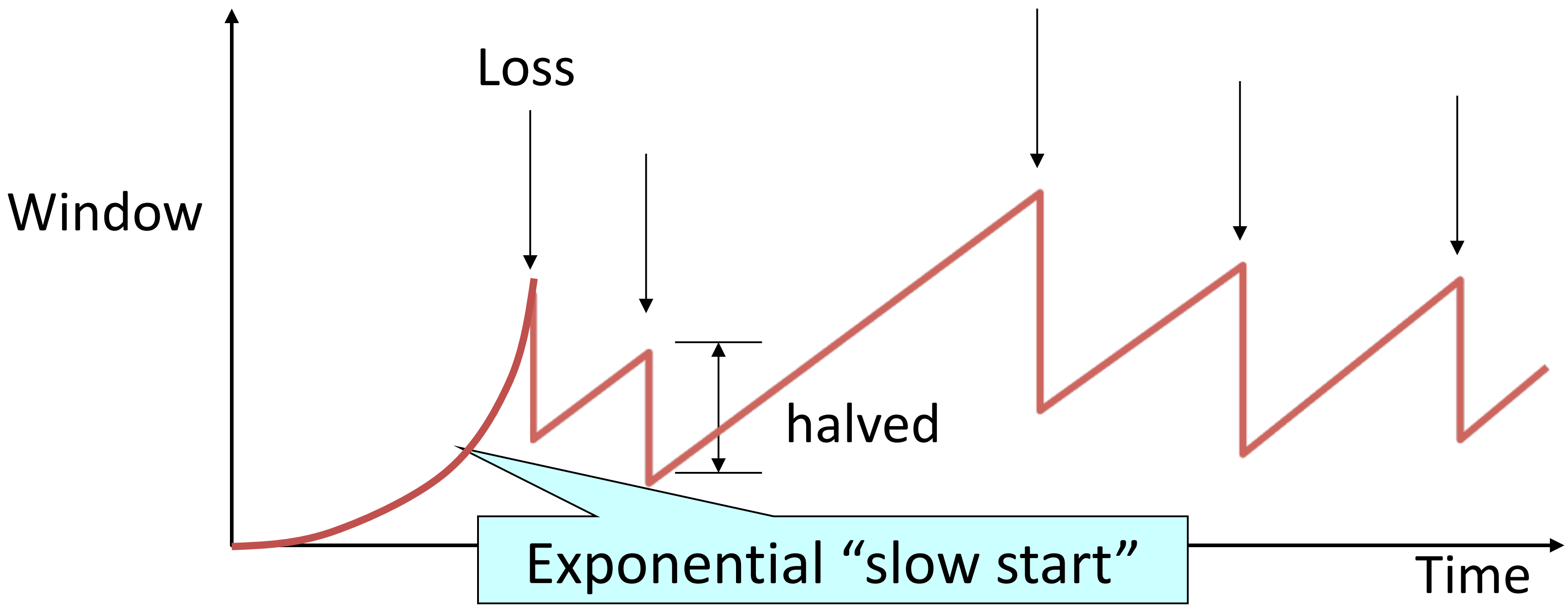
- Start with a small congestion window
  - Initially, CWND is 1 MSS
  - So, initial sending rate is MSS / RTT
- Could be pretty wasteful
  - Might be much less than actual bandwidth
  - Linear increase takes a long time to accelerate
- Slow-start phase (really "fast start")
  - Sender starts at a slow rate (hence the name)
  - … but increases rate exponentially until the first loss

Slow Start in Action

Double CWND per round-trip time



Slow Start and the TCP Sawtooth

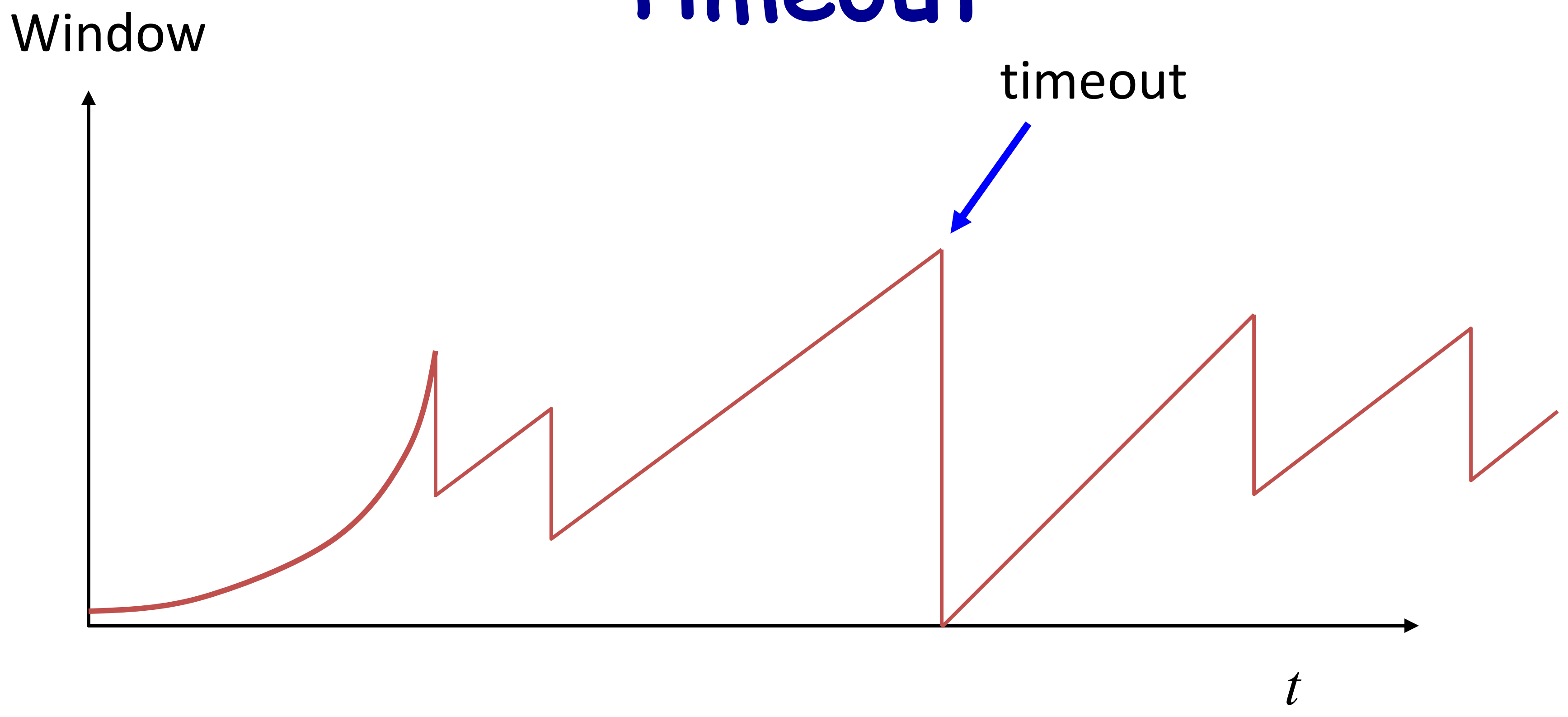


- TCP originally had *no* congestion control
  - Source would start by sending entire receiver window
  - Led to congestion collapse!
  - "Slow start" is, comparatively, slower

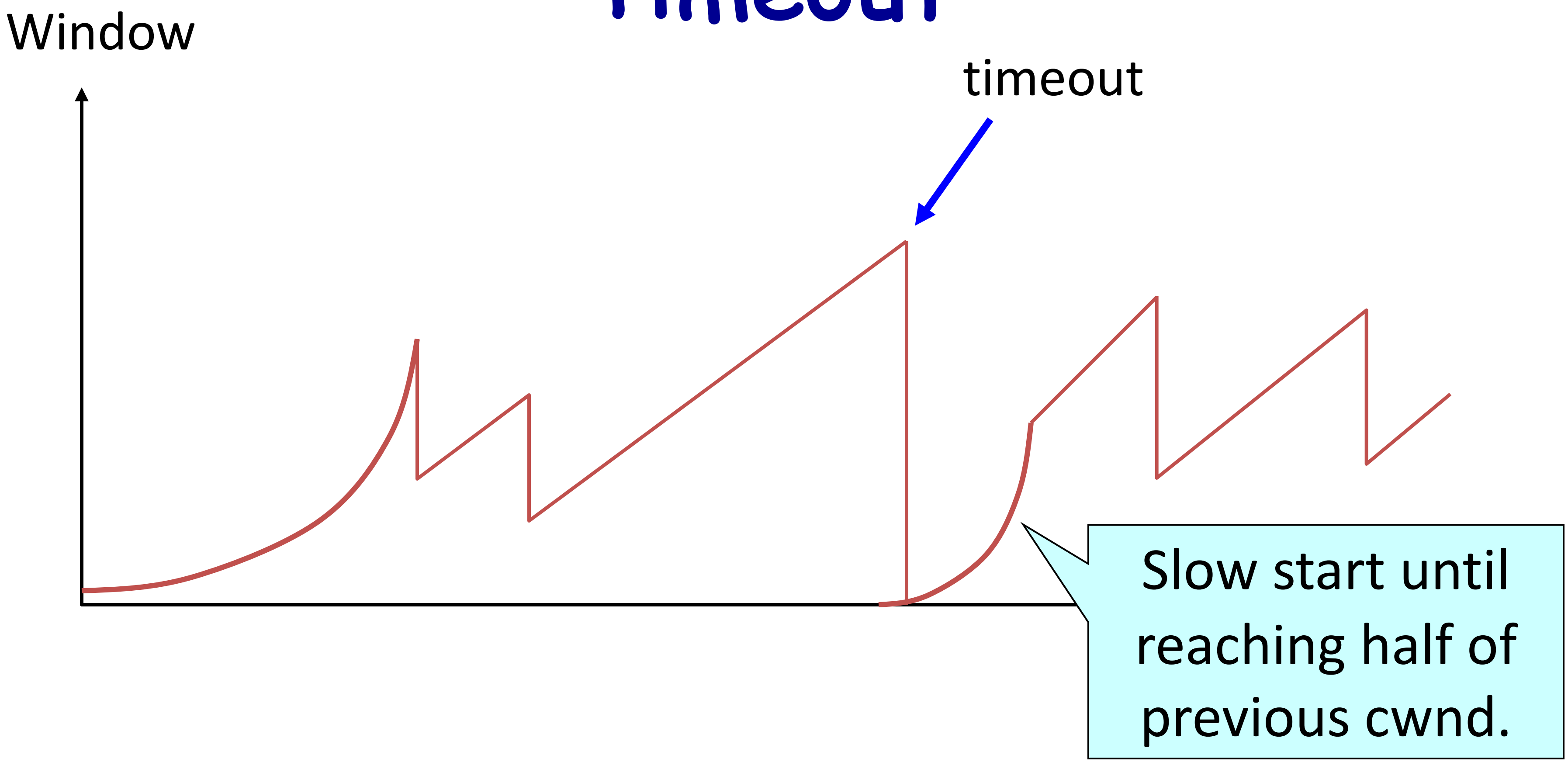
Two Kinds of Loss in TCP

- Timeout vs. Triple Duplicate ACK
  - Which suggests network is in worse shape?
- Timeout
  - If entire window was lost, buffers may be full
  - ...blasting entire CWND would cause another burst
  - ...be aggressive: start over with a low CWND
- Triple duplicate ACK
  - Might be do to bit errors, or "micro" congestion
  - ...react less aggressively (halve CWND)

Repeating Slow Start After Timeout



Repeating Slow Start After Timeout



Slow-start restart: Go back to CWND of 1, but take advantage of knowing the previous value of CWND.

Conclusions

- Congestion is inevitable
  - Internet does not reserve resources in advance
  - TCP actively tries to push the envelope
- Congestion can be handled
  - Additive increase, multiplicative decrease
  - Slow start and slow-start restart
- Fundamental tensions
  - Feedback from the network?
  - Enforcement of "TCP friendly" behavior?

Next Up in 461

Next Class Meeting:

Lectures 7 (Queue Management) &

8 (Middleboxes, Tunneling)

Precepts this Thursday and Friday:

Hamming Codes & Cyclic Redundancy Check

Heads-up: Assignment 2 due in 10 days, 2/24!