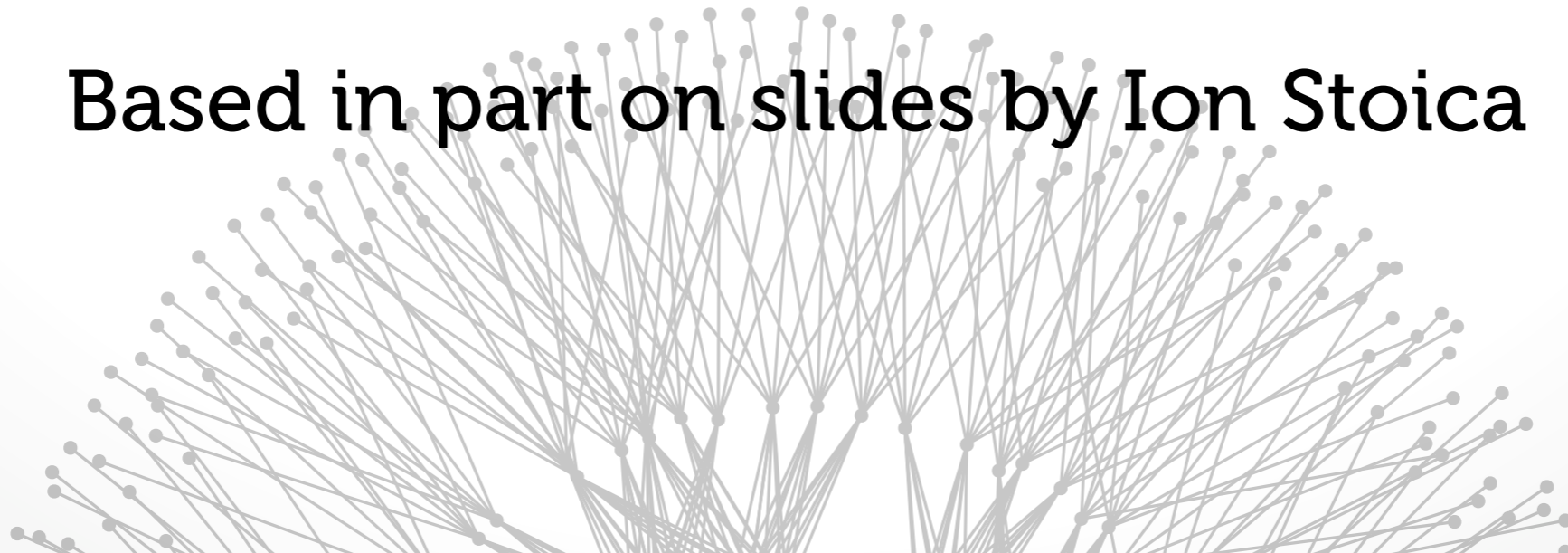


Congestion Control

Brighten Godfrey
CS 538 September 11 2012

Based in part on slides by Ion Stoica



Announcements



Looking good

- in that you all have some great comments
- all that submitted will get credit on Jacobson paper

But do:

- reduce length
- comment on each other's reviews
 - substantive comments on reviews count as reviews!

Don't:

- comment on paper readability etc. unless it's particularly notable
- summarize paper

Laptops in class



No laptops open in class

Assignment #1 released!



Part 1: Internet BGP Routing Traces

Part 2: TCP in a network emulator

Part 3: OpenFlow in a network emulator

Due in one week

**A starting point:
the sliding window protocol**



Make sure receiving end can handle data

Negotiated end-to-end, with no regard to network

Ends must ensure that no more than W packets are in flight if buffer has size W

- Receiver ACKs packets
- When sender gets an ACK, it knows packet has arrived

Sliding window-based flow control



At the sender..



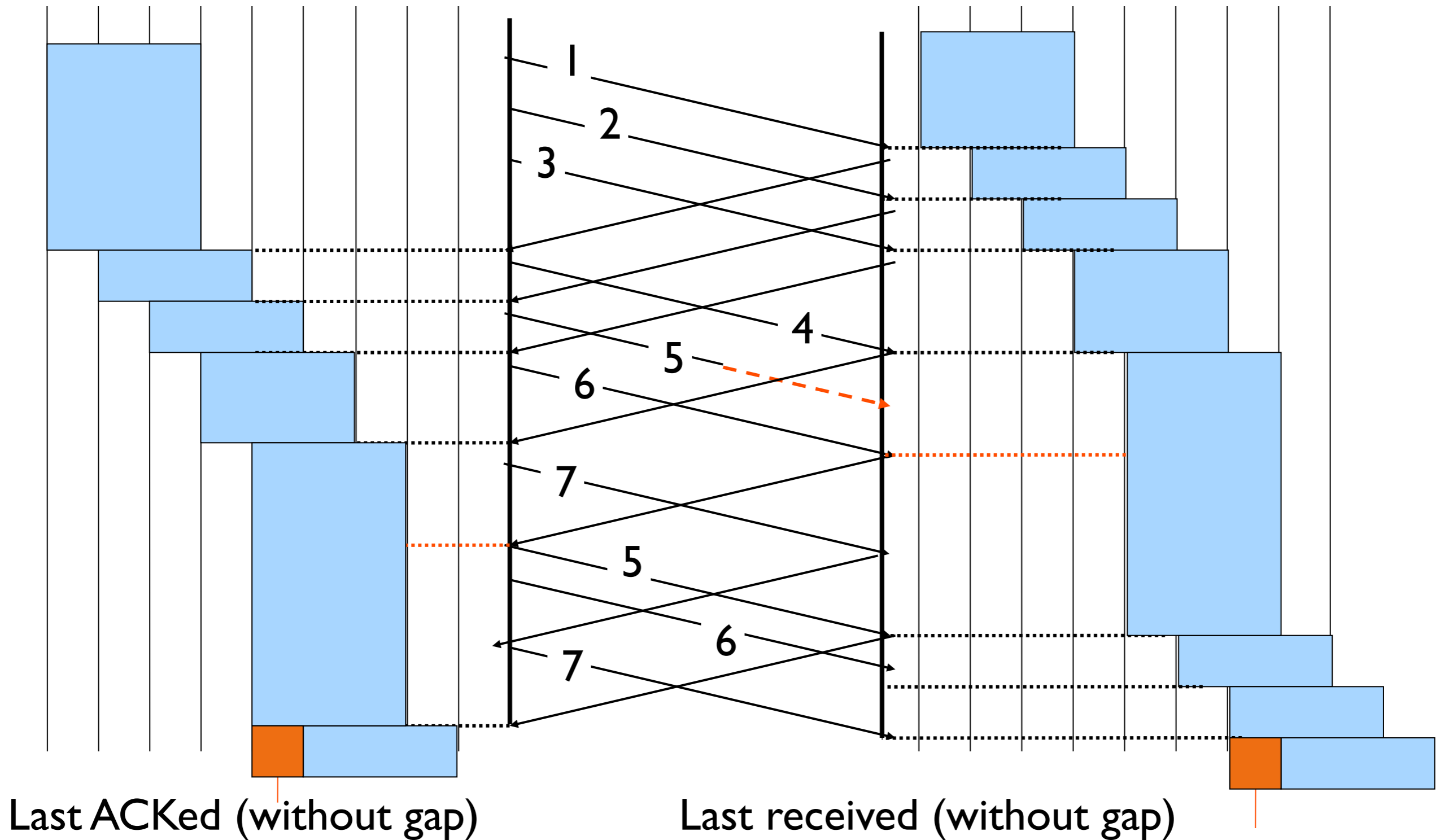
Sliding window-based flow control



At the receiver..



Sliding window





What is the throughput in terms of RTT and window size?

- Throughput is $\sim (w/RTT)$

Sender has to buffer all unacknowledged packets, because they may require retransmission

Receiver may be able to accept out-of-order packets, but only up to its buffer limits

What should the receiver ACK?



ACK every packet, giving its sequence number

Use negative ACKs (NACKs), indicating which packet did not arrive

Use cumulative ACK, where an ACK for number n implies ACKS for all $k < n$

Use selective ACKs (SACKs), indicating those that did arrive, even if not in order



Getting to equilibrium: Slow Start

- Initially sends packets slowly: very conservative
- Accelerates quickly
- Maybe too conservative now
 - <http://research.google.com/pubs/pub36640.html>

Conservation: Round-Trip Timing

Congestion Avoidance

Round-trip timing



Must retransmit packets that were dropped

To do this efficiently

- Keep transmitting whenever possible
- Detect dropped packets and retransmit quickly

Requires:

- Timeouts (with good timers)
- Other hints that packet were dropped

A bad timer algorithm



$T(n)$ = measured RTT of
this packet

mean:

$$A(n) = b * A(n-1) + (1 - b) * T(n)$$

$$\text{Timeout}(n) = 2 * A(n)$$

Is twice the mean what we really want?

- No: want outliers
- $2A(n)$ a poor estimate of outliers
- Idea: measure **deviation** from mean

Better timer [Jacobson]



$T(n)$ = measured RTT of
this packet

mean: $A(n) = b * A(n-1) + (1 - b) * T(n)$
deviation: $D(n) = b * D(n-1) + (1 - b) * (T(n) - A(n))$
Timeout(n) = $A(n) + 4D(n)$

Questions:

- Measure $T(n)$ only for original transmissions. Why?
- Double Timeout after timeout. Why?
- Is deviation what we *really* want? Really?



What do we REALLY want?

- Estimate whether $\text{Pr}[\text{packet lost}]$ is high
- Is timing the only way?

Another way: Duplicate ACKs

- Receiver sends an ACK whenever a packet arrives
- ACK has seq. # of last **consecutively** received packet
- Duplicate ACKs suggest missing packet (assumptions?)
- Modern TCPs: Fast Retransmit after 3 dup-ACKs

Does this eliminate need for timers?

- No: What if we get no packets from receiver?
- But, makes them less important

Congestion

TCP congestion control



Can the network handle the rate of data?

Determined end-to-end, but TCP is making guesses about the state of the network

Two papers:

- Good science vs great engineering

Dangers of increasing load



Knee – point after which

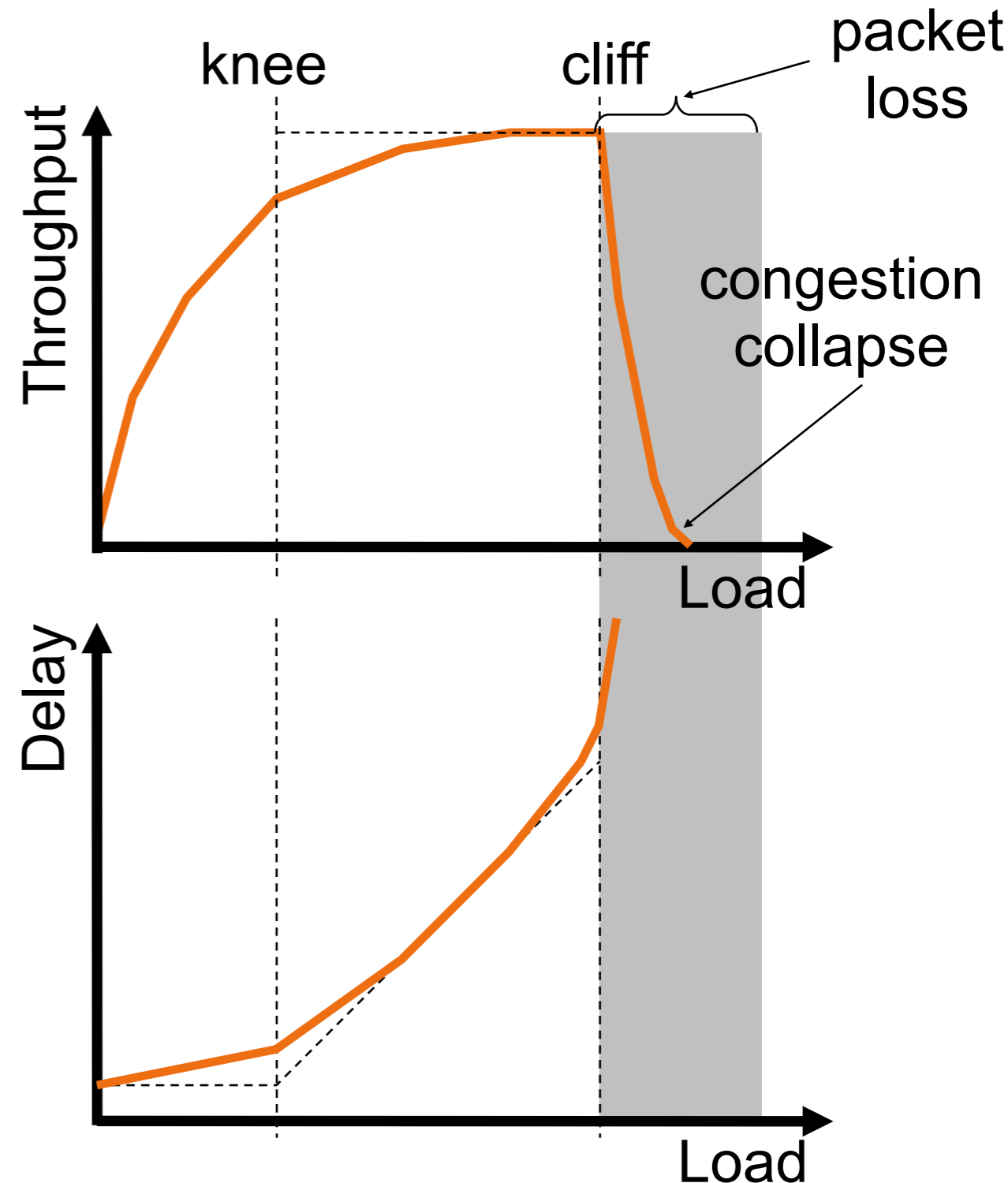
- Throughput increases very slow
- Delay increases fast

Cliff – point after which

- Throughput starts to decrease very fast to zero (congestion collapse)
- Delay approaches infinity

In an M/M/1 queue

- Delay = $1/(1 - \text{utilization})$



Cong. control vs. cong. avoidance

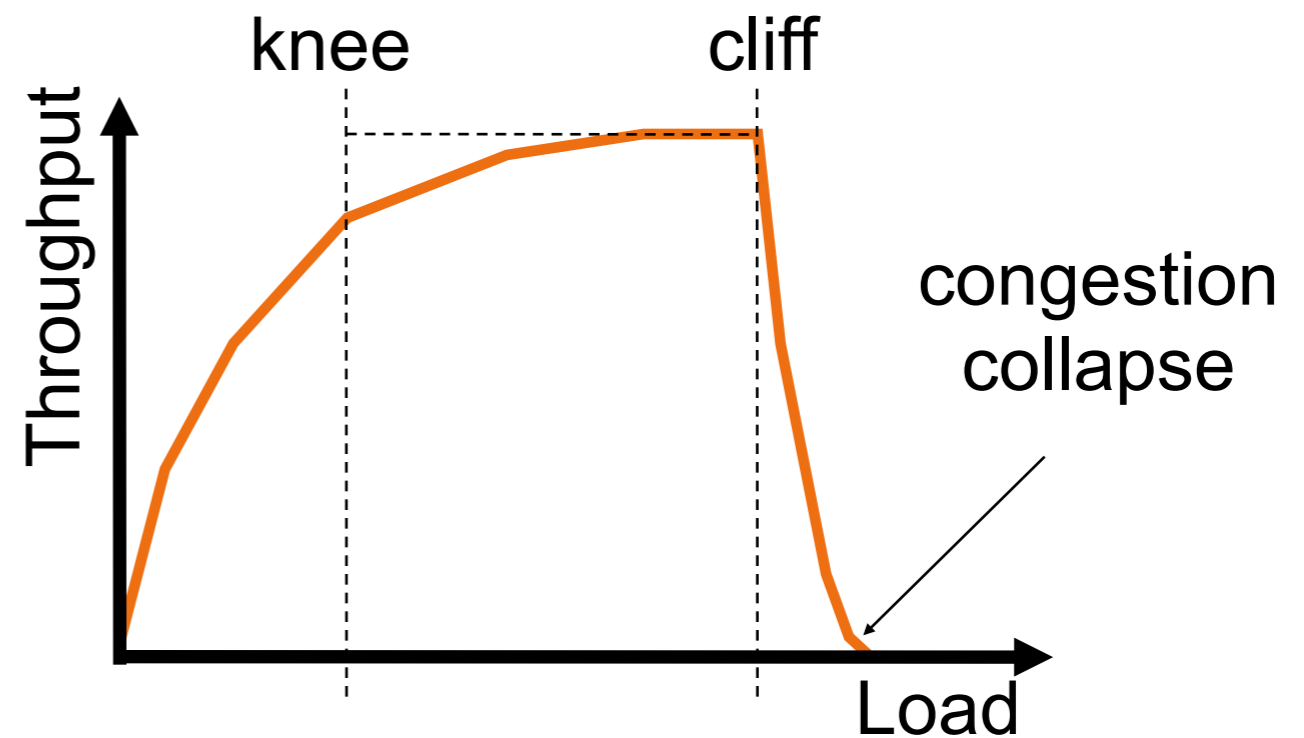


Congestion control goal

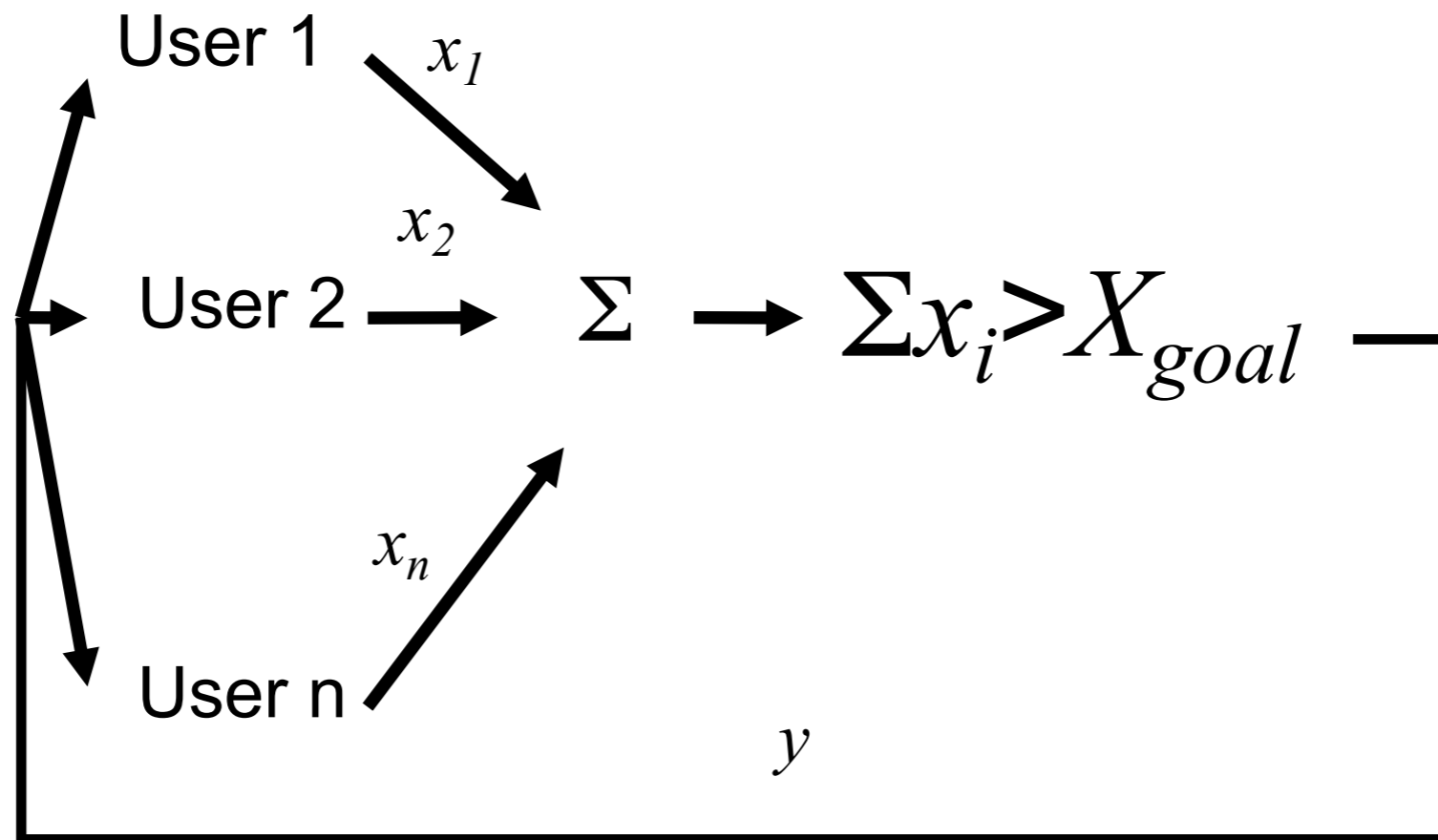
- Stay left of cliff

Congestion avoidance goal

- Stay left of knee



Control system model [CJ89]



Simple, yet powerful model

Explicit binary signal of congestion

Possible choices



$$x_i(t+1) = \begin{cases} a_I + b_I x_i(t) & \text{increase} \\ a_D + b_D x_i(t) & \text{decrease} \end{cases}$$

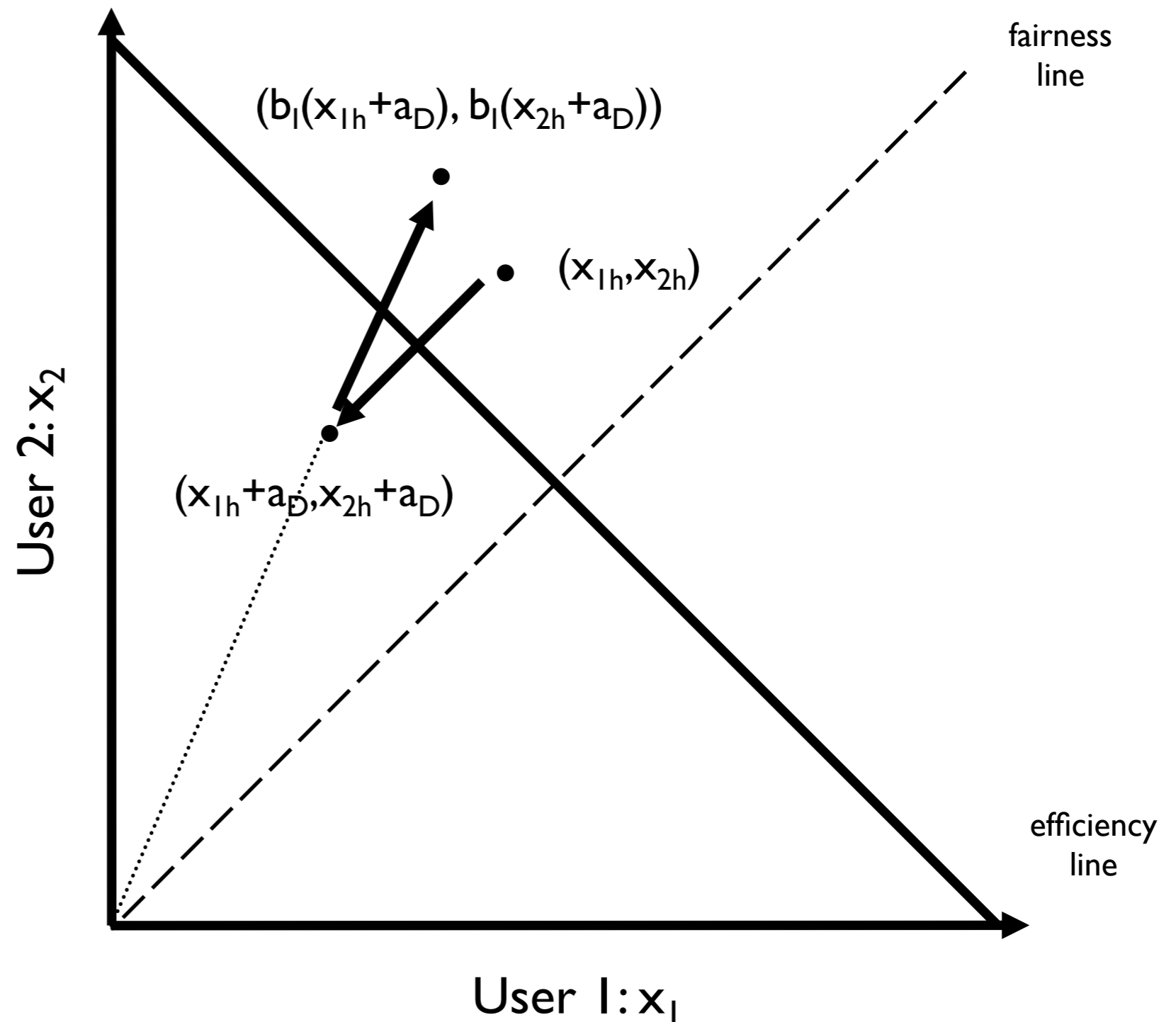
- Multiplicative increase, additive decrease
 - $a_I=0, b_I>1, a_D<0, b_D=1$
- Additive increase, additive decrease
 - $a_I>0, b_I=1, a_D<0, b_D=1$
- Multiplicative increase, multiplicative decrease
 - $a_I=0, b_I>1, a_D=0, 0<b_D<1$
- Additive increase, multiplicative decrease
 - $a_I>0, b_I=1, a_D=0, 0<b_D<1$

Which should
we pick?

Mult. increase, additive decrease



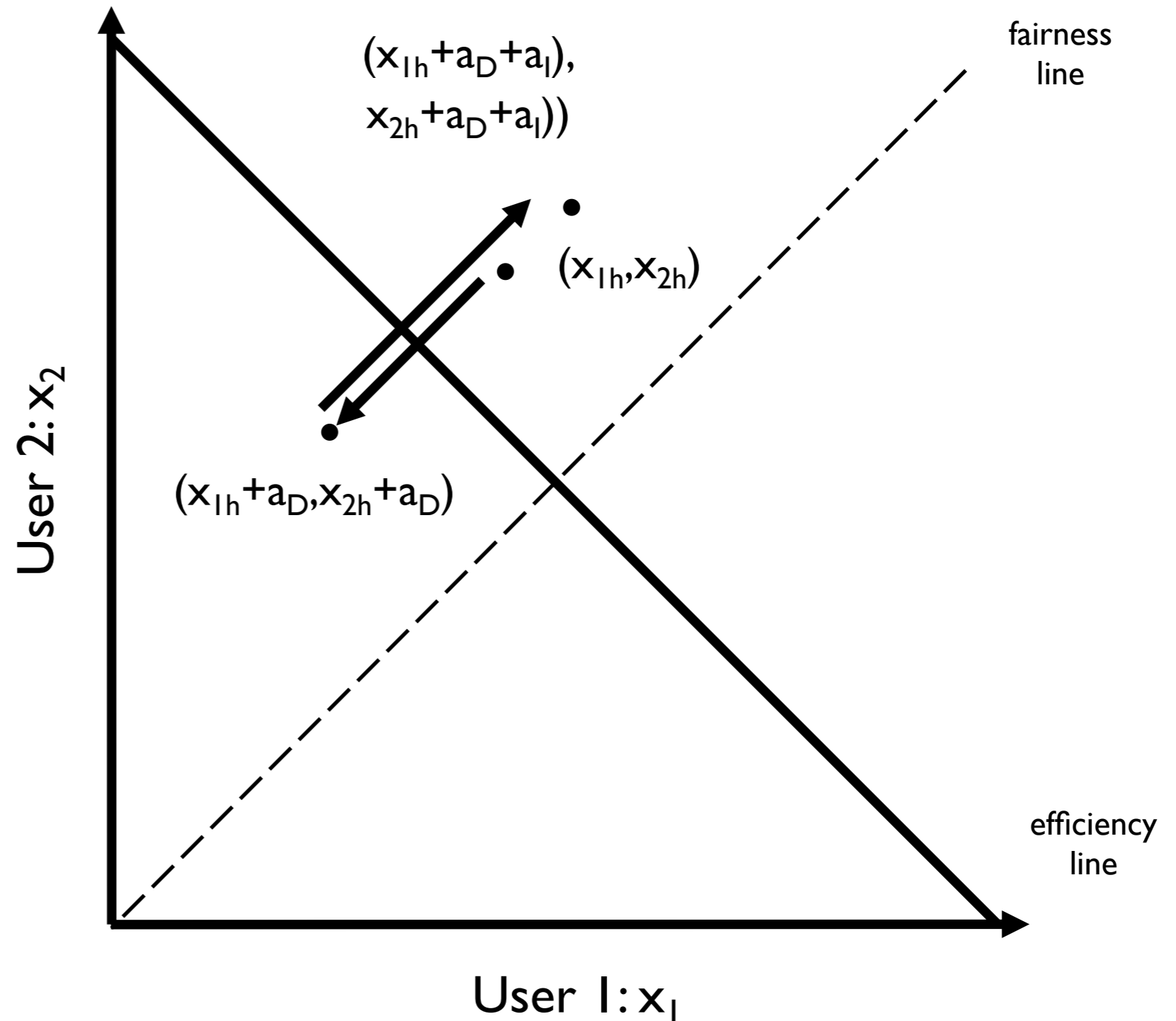
- Does not converge to fairness
- (Additive decrease worsens fairness)



Additive increase, add. decrease



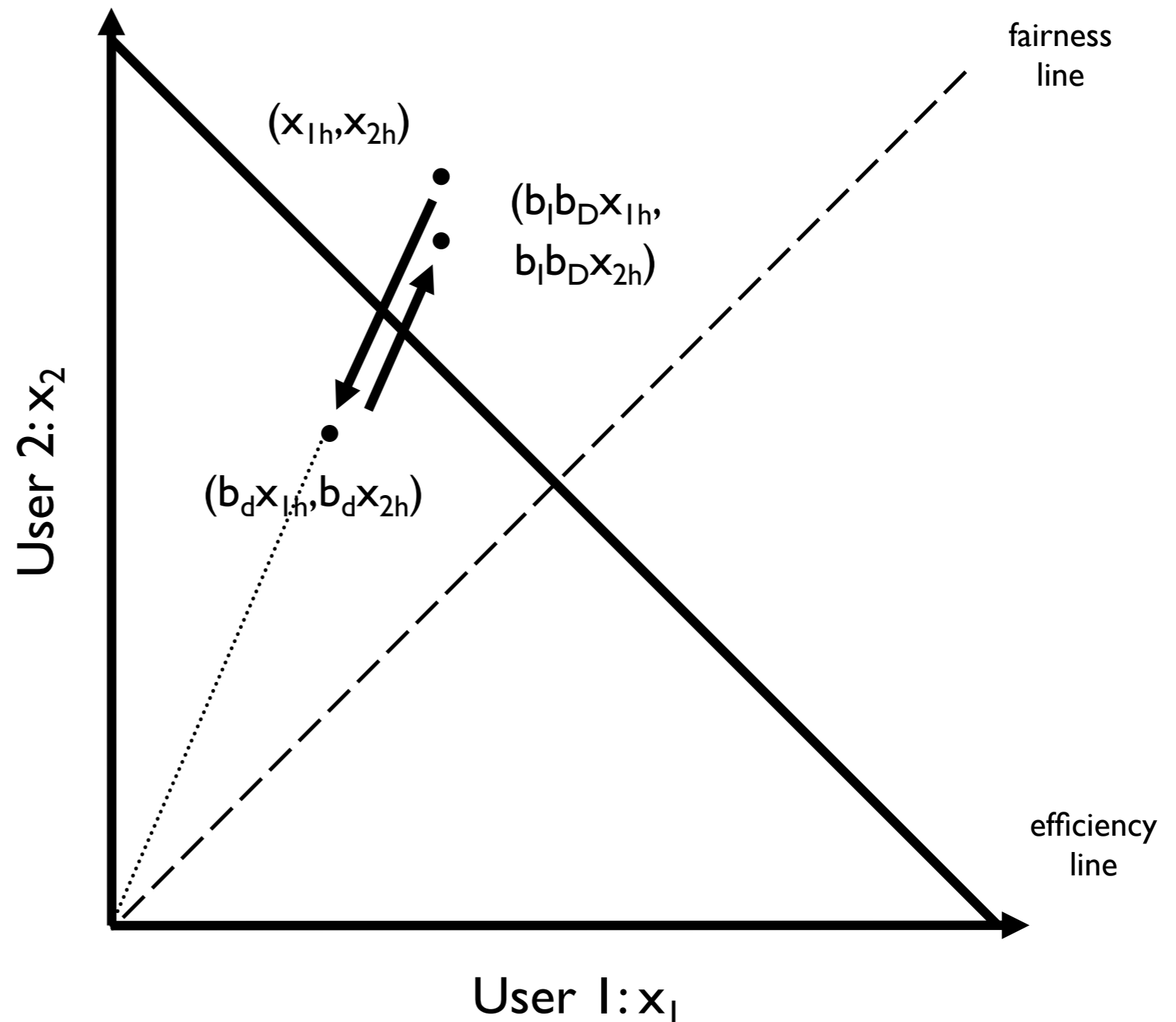
- Reaches stable cycle, but does not converge to fairness



Mult. increase, mult. decrease



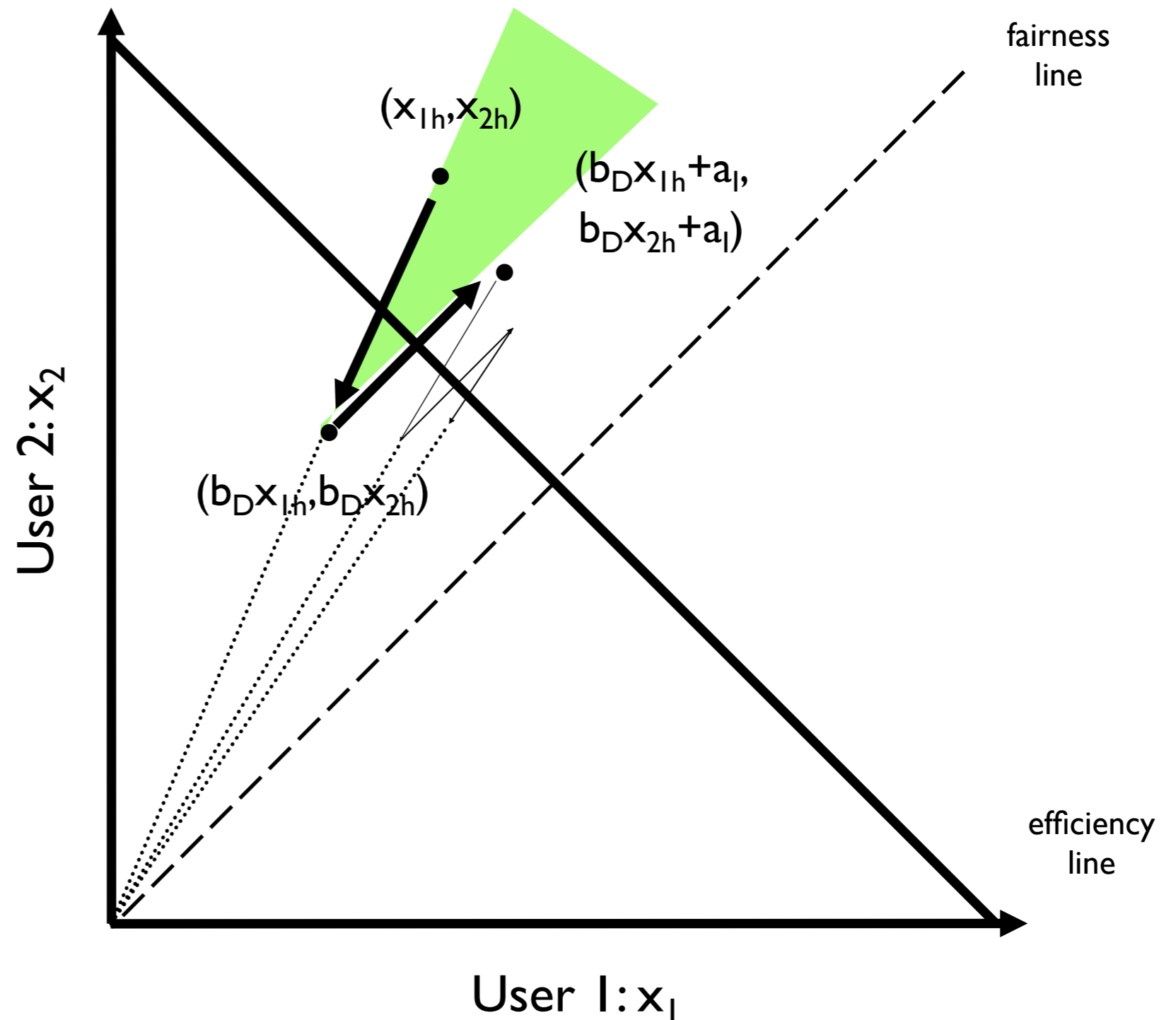
- Converges to stable cycle, but is not fair



Additive increase, mult. decrease



- Converges to stable and fair cycle





Critical to understanding complex systems

- [CJ89] model relevant after >20 years, 10^6 increase of bandwidth, $1000x$ increase in number of users

Criteria for good models

- Two conflicting goals: reality and simplicity
- Realistic, complex model \rightarrow too hard to understand, too limited in applicability
- Unrealistic, simple model \rightarrow can be misleading

Putting the pieces together

TCP congestion control



[CJ89] provides theoretical basis for basic congestion avoidance mechanism

Must turn this into real protocol

TCP congestion control



Maintains three variables:

- cwnd: congestion window
- flow_win: flow window; receiver advertised window
- ssthresh: threshold size (used to update cwnd)

For sending, use: $\text{win} = \min(\text{flow_win}, \text{cwnd})$

TCP: slow start



Goal: reach knee quickly

Upon starting (or restarting):

- Set $\text{cwnd} = 1$
- Each time a segment is acknowledged, increment cwnd by one ($\text{cwnd}++$).

Slow Start is not actually slow

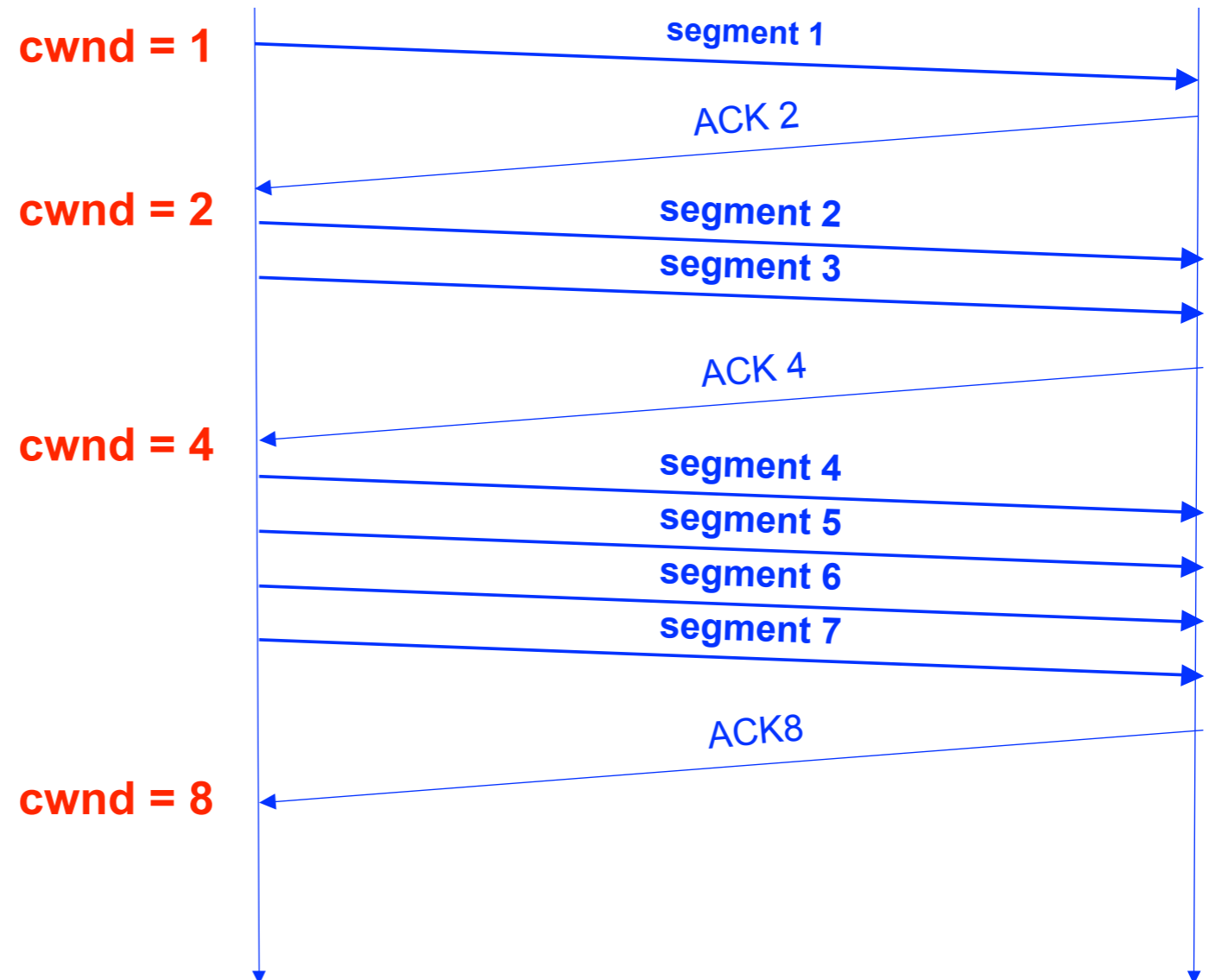
- cwnd increases exponentially

Slow start example



The congestion window size grows very rapidly

TCP slows down the increase of cwnd when $cwnd \geq ssthresh$



Congestion avoidance



Slow down “Slow Start”

ssthresh variable is lower-bound guess about location of knee

If $cwnd > ssthresh$ then

each time a segment is acknowledged

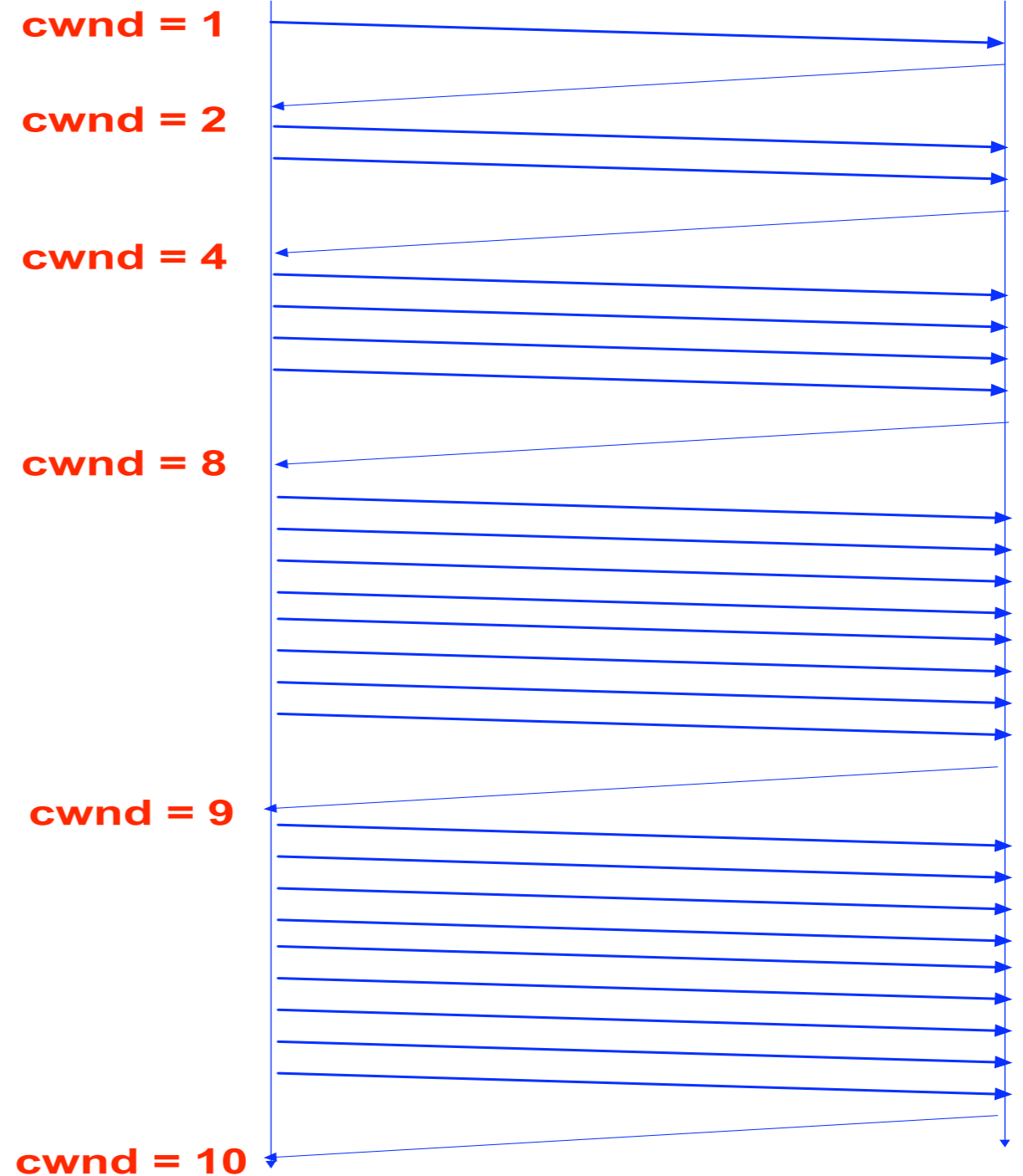
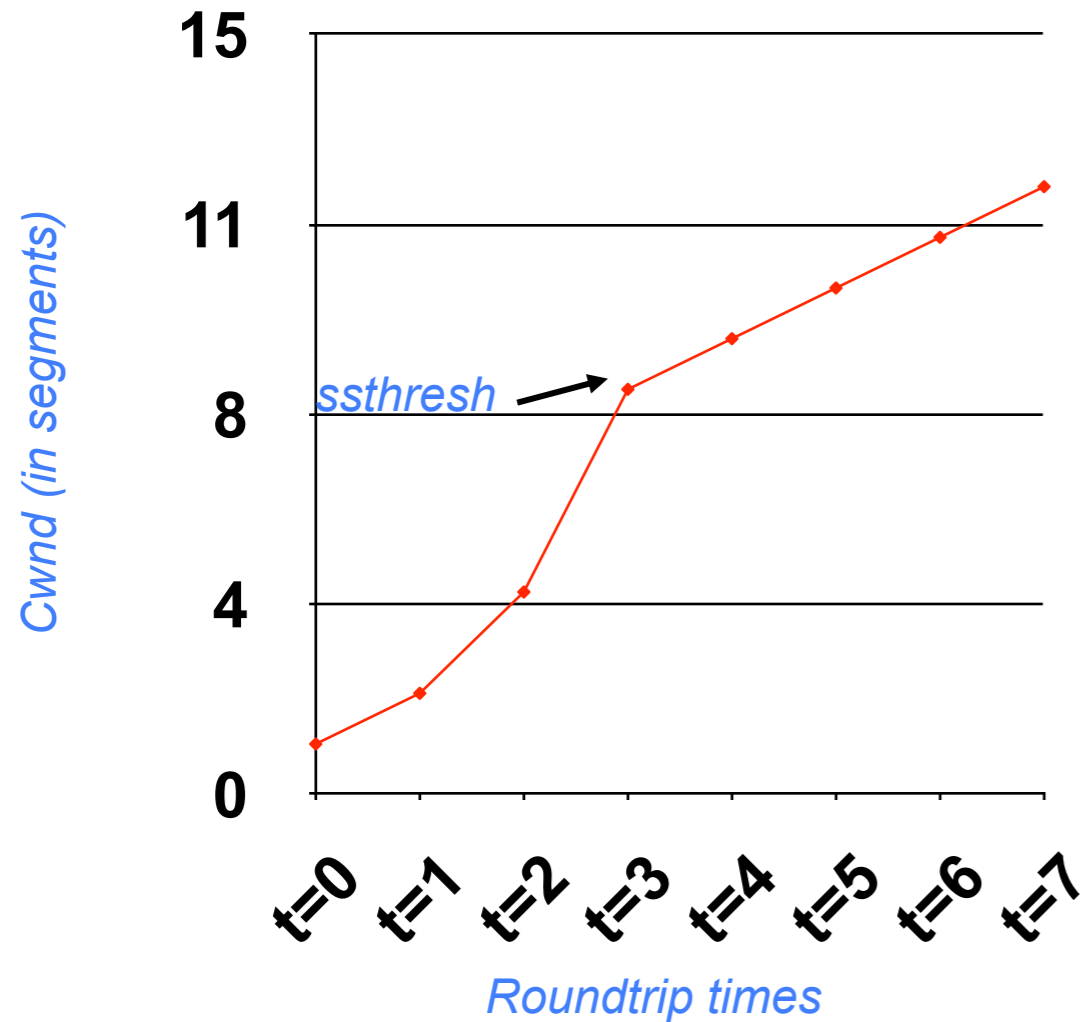
increment $cwnd$ by $1/cwnd$ ($cwnd += 1/cwnd$).

Result: $cwnd$ is increased by one after a full window of segments have been acknowledged

Slow start/cong. avoidance example



- Assume that $ssthresh = 8$



All together: TCP pseudocode



Initially:

```
  cwnd = 1;  
  ssthresh = infinite;
```

New ack received:

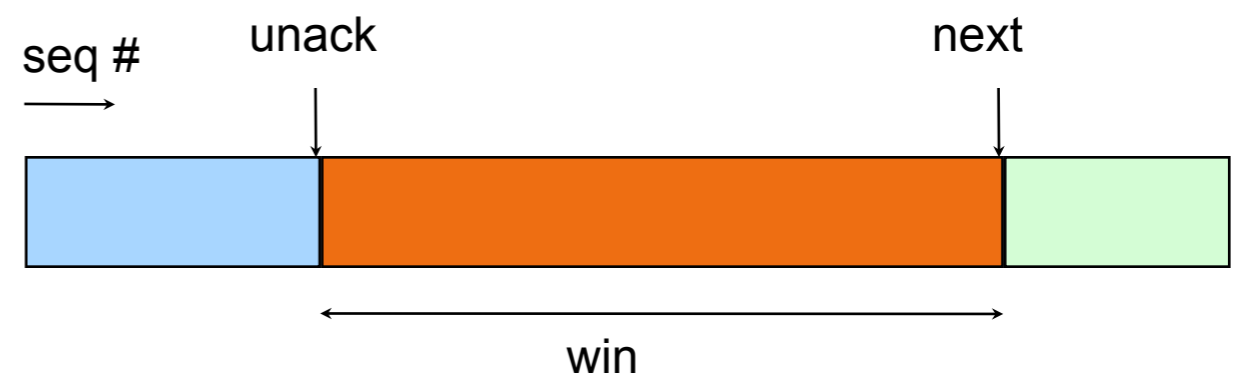
```
  if (cwnd < ssthresh)  
    /* Slow Start */  
    cwnd = cwnd + 1;  
  else  
    /* Additive increase */  
    cwnd = cwnd + 1/cwnd;
```

Timeout:

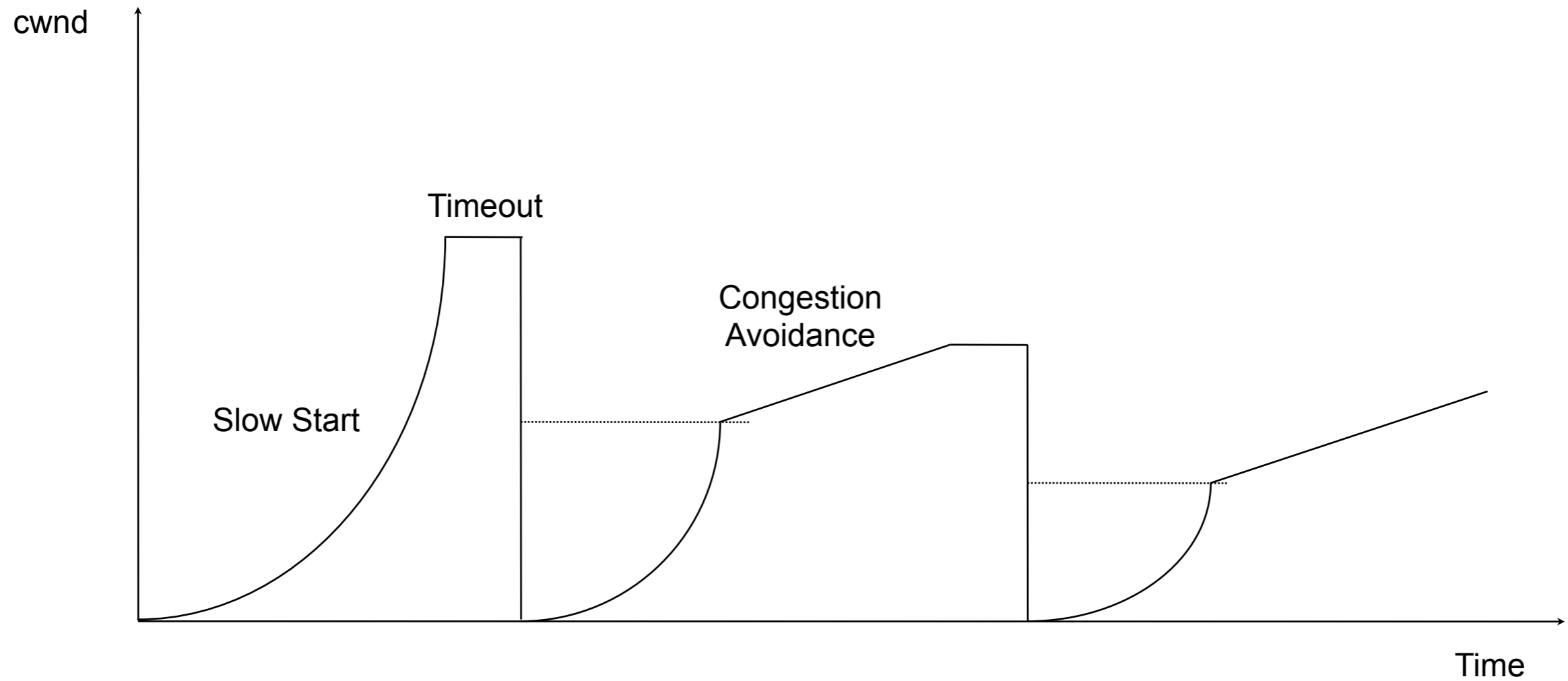
```
  /* Multiplicative decrease */  
  ssthresh = cwnd/2;  
  cwnd = 1;
```

```
while (next < unack + win)  
  transmit next packet;
```

```
where win = min(cwnd,  
                flow_win);
```



The big picture (so far)

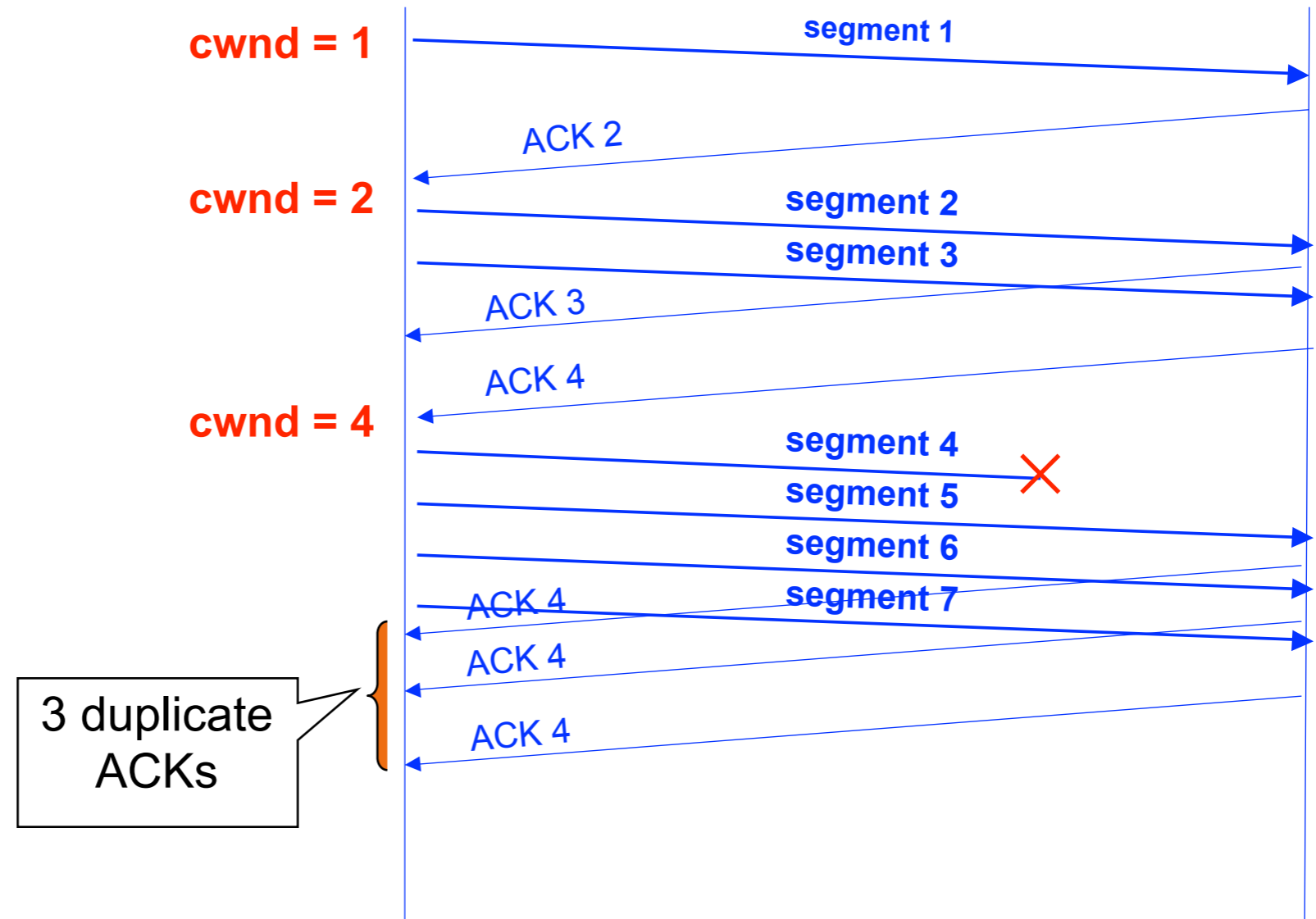


Fast retransmit



Resend a segment
after 3 duplicate
ACKs

Avoids waiting for
timeout to
discover loss





After a fast-retransmit set *cwnd* to *ssthresh/2*

- i.e., don't reset *cwnd* to 1

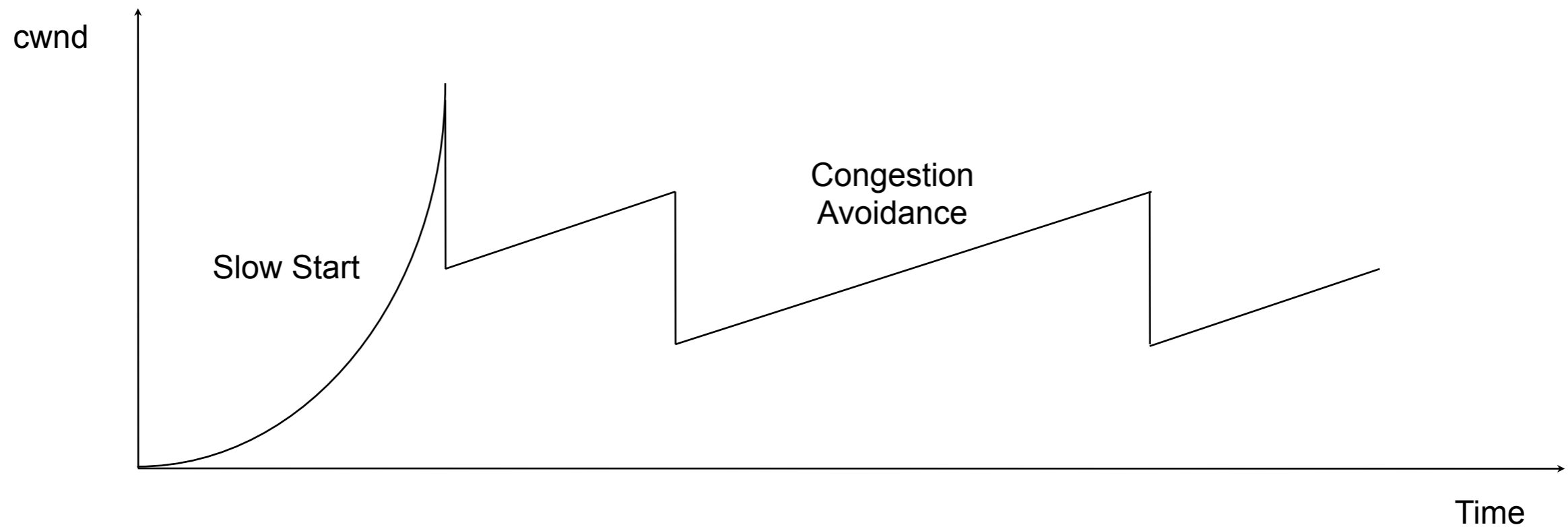
But when RTO expires still do *cwnd* = 1

Fast Retransmit and Fast Recovery

- Implemented by TCP Reno
- Most widely used version of TCP today

Lesson: avoid RTOs at all costs!

Picture with fast retransmit & recov.



Retransmit after 3 duplicated acks

- prevent expensive timeouts

No need to slow start again

At steady state, cwnd oscillates around the optimal window size

Discussion

Engineering vs. Science in CC



Great engineering by Jacobson and others built useful protocol

- TCP Reno, etc.

Good science by Chiu, Jain and others

- Basis for understanding why it works so well

Limitations of TCP CC



In what ways is TCP congestion control broken or suboptimal?

A partial list...



Efficiency

Tends to fill queues (adding latency)

Slow to converge (for short flows or links with high bandwidth•delay product)

Loss \neq congestion

May not fully utilize bandwidth



Fairness

Unfair to large-RTT flows (less throughput)

Unfair to short flows if ssthresh starts small

Equal rates isn't necessarily "fair" or best

Vulnerable to selfish & malicious behavior

- TCP assumes everyone is running TCP!

More announcements



Check out Jack Dorsey talk tonight

- 7pm, 1310 DCL

Next time: “Fixing” TCP

- Efficiency
- Fairness

Reading:

- Briscoe: Flow Rate Fairness: Dismantling a Religion

Presentation topic scheduling

- Do this by Thursday 11:59 pm

