Congestion Control

CS 168

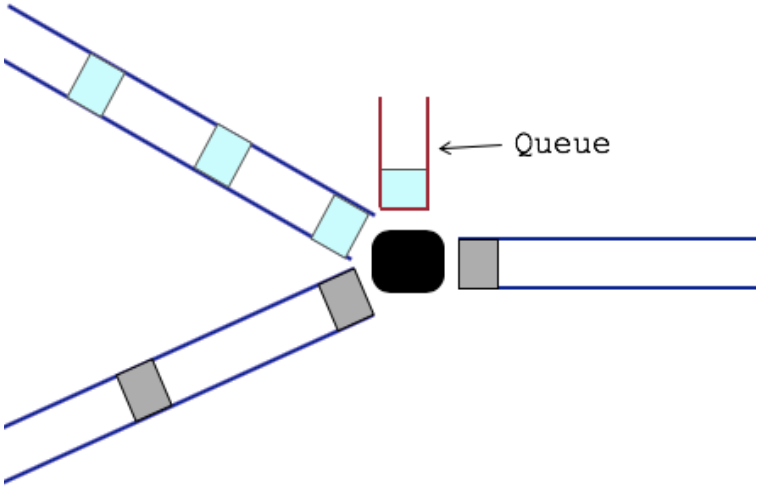
http://cs168.io

Sylvia Ratnasamy

Today: Congestion Control

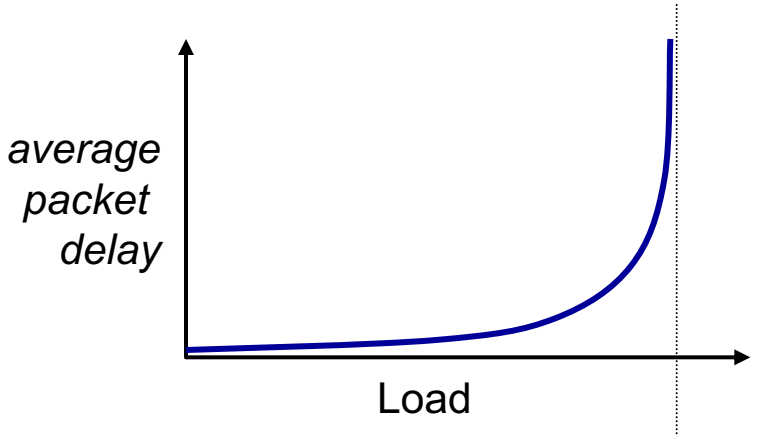
- One of the "core" topics in networking
  - Will occupy us for ~3 lectures
- Today: concepts and design space
  - Thu: CC in TCP
  - Next week: advanced CC

Recall: Lecture 3



- If two packets arrive at a router at the same time, the router will transmit one and buffer the other
- If many packets arrive close in time
  - the router cannot keep up → gets congested
  - causes packet delays and drops

Congestion is harmful



Typical queuing system with bursty arrivals

Some History: TCP in the 1980s

- Sending rate only limited by flow control
  - Dropped packets → senders retransmit, repeatedly!

- Led to "congestion collapse" in Oct. 1986

> In October of '86, the Internet had the first of what became a series of 'congestion collapses'. During this period, the data throughput from LBL to UC Berkeley (sites separated by 400 yards and two IMP hops) dropped from 32 Kbps to 40 bps. We were fascinated by this sudden factor-of-thousand drop in bandwidth and embarked on an investigation of why things had gotten so bad. In particular, we wondered if the 4.3BSD (Berkeley UNIX) TCP was mis-behaving or if it could be tuned to work better under abysmal network conditions. The answer to both of these questions was "yes".

https://ee.lbl.gov/papers/congavoid.pdf

Van Jacobson

- Researcher in the networking group at LBL
- Many contributions to the early TCP/IP stack
- Creator of many widely used network tools
  - traceroute, tcpdump, Berkeley Packet Filter, ...
- Later Chief Scientist at Cisco, now at Google
  - Recently: BBR, a new TCP CC protocol used by Google

Their Approach

- Incremental extension to TCP's existing protocol
  - Source adjusts its window size based on observed packet loss
- A pragmatic and effective solution
  - Required no upgrades to routers or applications!
  - Patch of a few lines of code to BSD's TCP implementation
  - Quickly adopted and has been the de-facto approach since
  - A lesson on wisdom in system design
- Extensively researched and improved upon
  - Countless variants (we will not discuss)
  - As well as radically different approaches (we'll see a few next week)

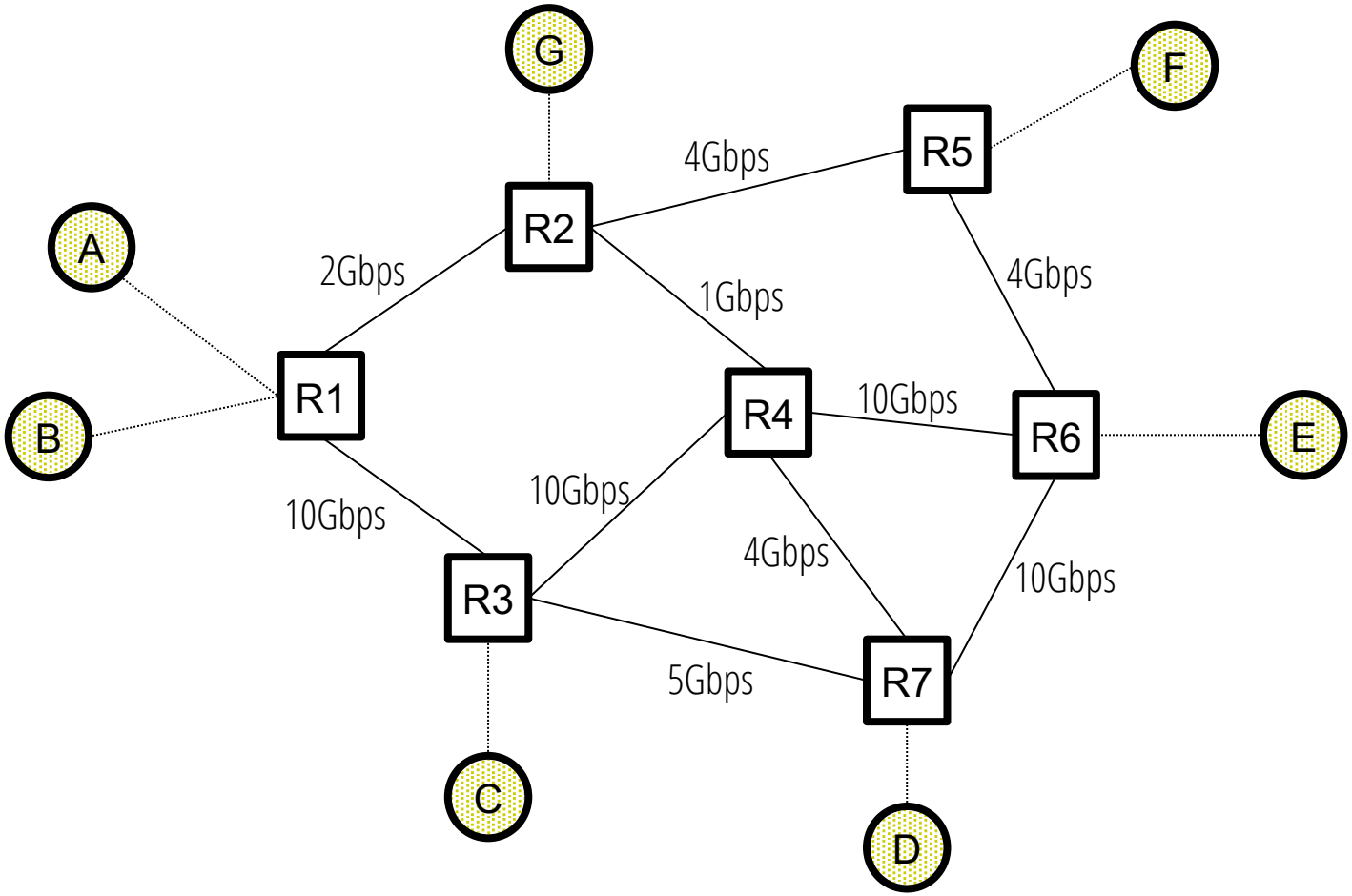
CC more generally...

- Huge literature on the problem
  - In systems, control theory, game theory, stats, econ
- Recent resurgence of interest in industry
  - New pressure for high-performance (cloud services)
  - New context (datacenters, new app workloads)
  - New methods (ML)

Topics for today

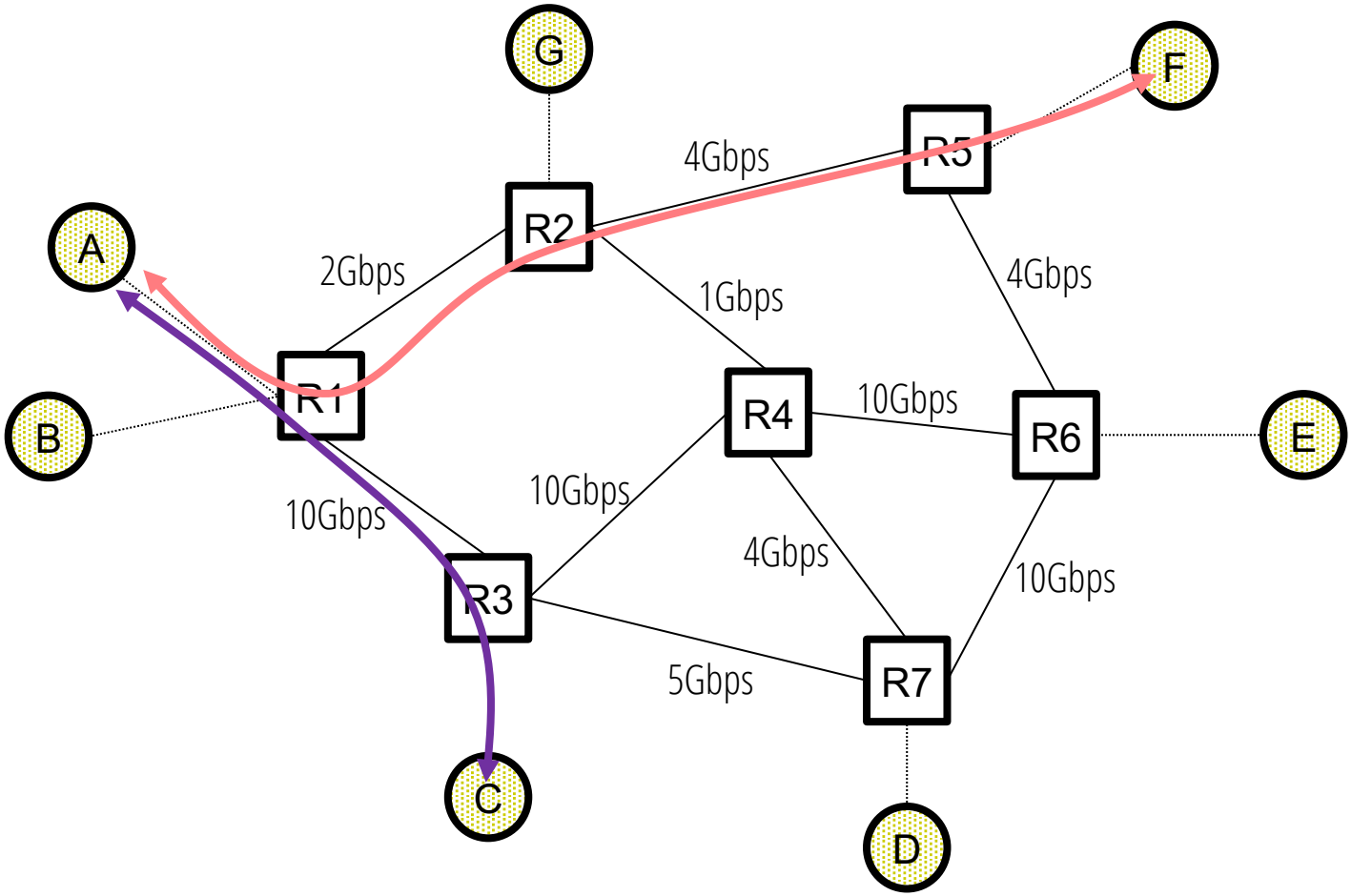
- What makes CC a hard problem?
- Goals for a good solution
- Design space
- Components of a solution
- TCP's approach (high level)

- Next week:
  - TCP CC in detail
  - Advanced topics in CC

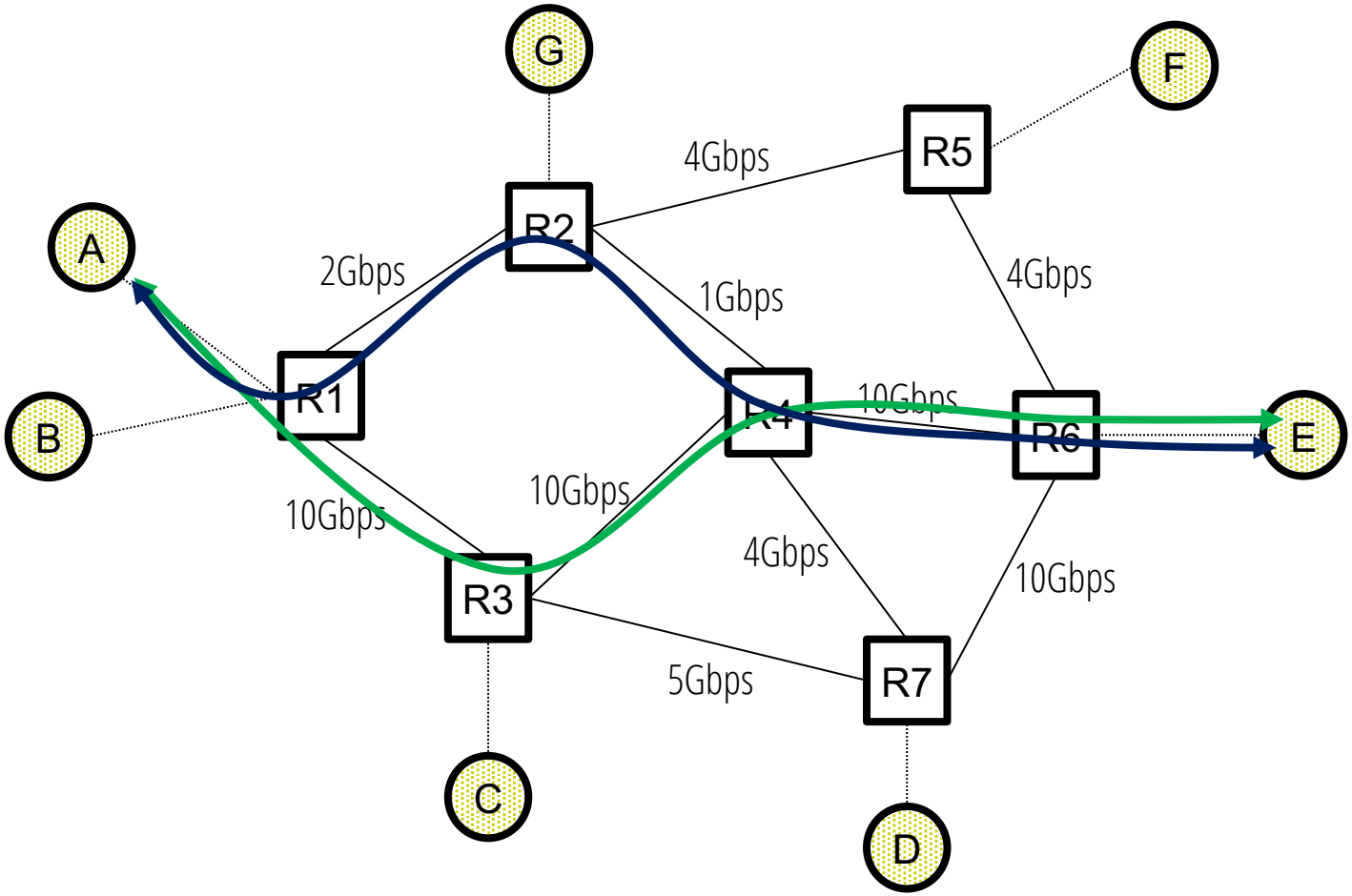


At what rate should Host A send traffic?

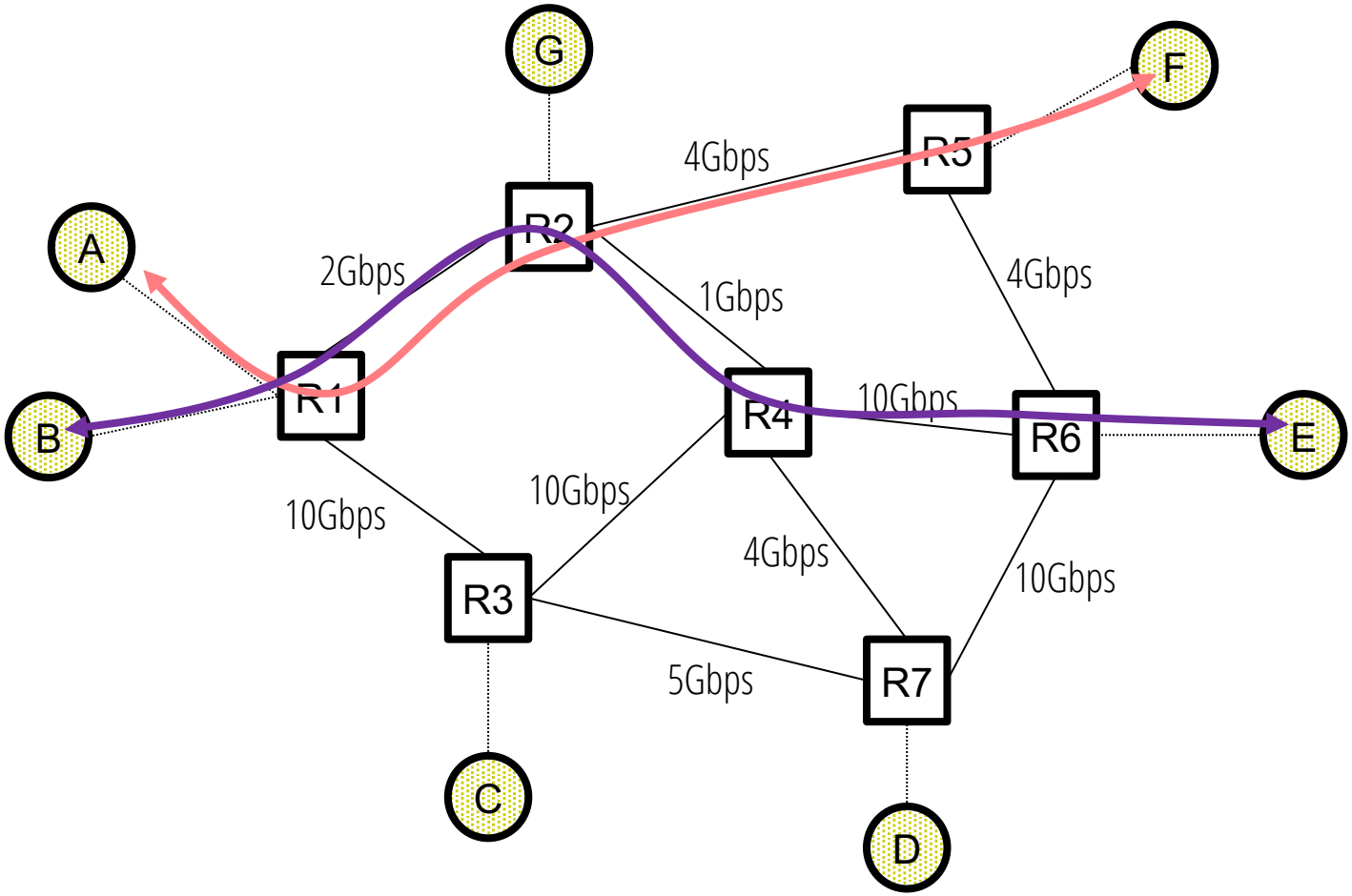
*For this example, we'll ignore the BW of links attaching hosts to routers



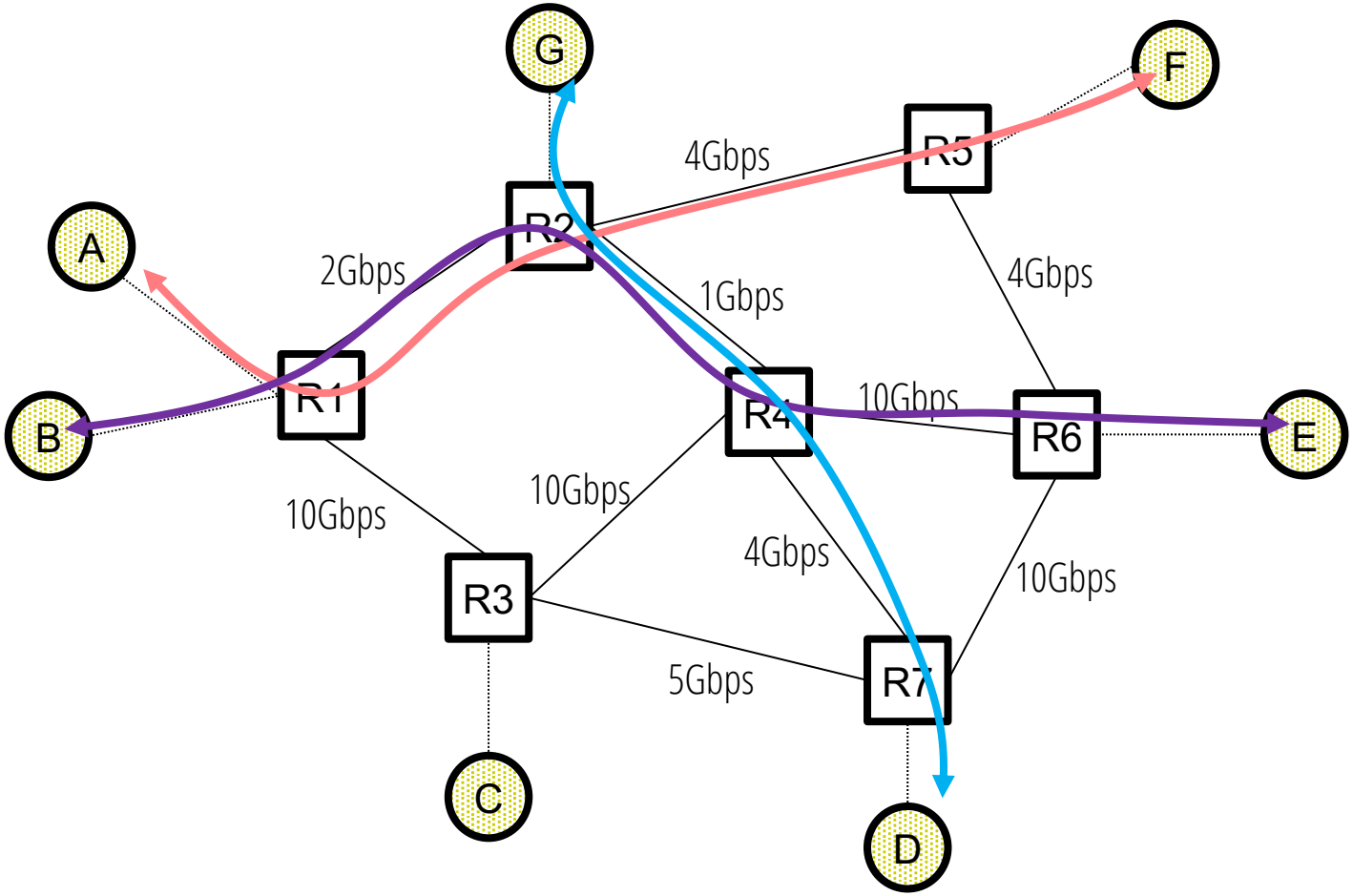
Depends on the destination



Changes with routing dynamics



Depends on “competing” flows



Including "indirect" competition!

Congestion Control

- Fundamentally, a resource allocation problem
  - Flow is assigned a shared of the link BW along a path
- But more complex than traditional resource alloc.
  - Changing one link's allocation can have global impact
  - And we're changing allocations on every flow arrival/exit
  - No single entity has a complete view or complete control!
    - (Exception: within a datacenter)
- Allocations in our context are highly **interdependent**

Outline for today

- What makes CC a hard problem?
- Goals for a good solution
- Design space
- TCP's approach (high level)
- Components of a solution

Goals

- From a resource allocation perspective
  - Low packet delay and loss
  - High link utilization
  - “Fair” sharing across flows

Aim: a good tradeoff between the above goals

Goals

- From a resource allocation perspective
  - Low packet delay and loss
  - High link utilization
  - “Fair” sharing across flows
- From a systems perspective
  - **Practical:** scalable, decentralized, adaptive, *etc.*

Any questions?

Outline for today

- What makes CC a hard problem?
- Goals for a good solution
- Design space
- TCP's approach (high level)
- Components of a solution

Possible Approaches

(0) Send at will

What happens if A sends at 10Gbps?

Possible Approaches

(1) Reservations

- Pre-arrange bandwidth allocations
- Comes with all the problems we've discussed

Possible Approaches

(1) Reservations

(2) Pricing / priorities

- Don’t drop packets for the highest bidders/priority users
- Charge users based on current congestion levels
- Requires payment model

Possible Approaches

(1) Reservations

(2) Pricing / priorities

(3) Dynamic Adjustment

- Hosts dynamically learn current level of congestion
- Adjust their sending rate accordingly
- Many options for how to implement this basic idea

Possible Approaches

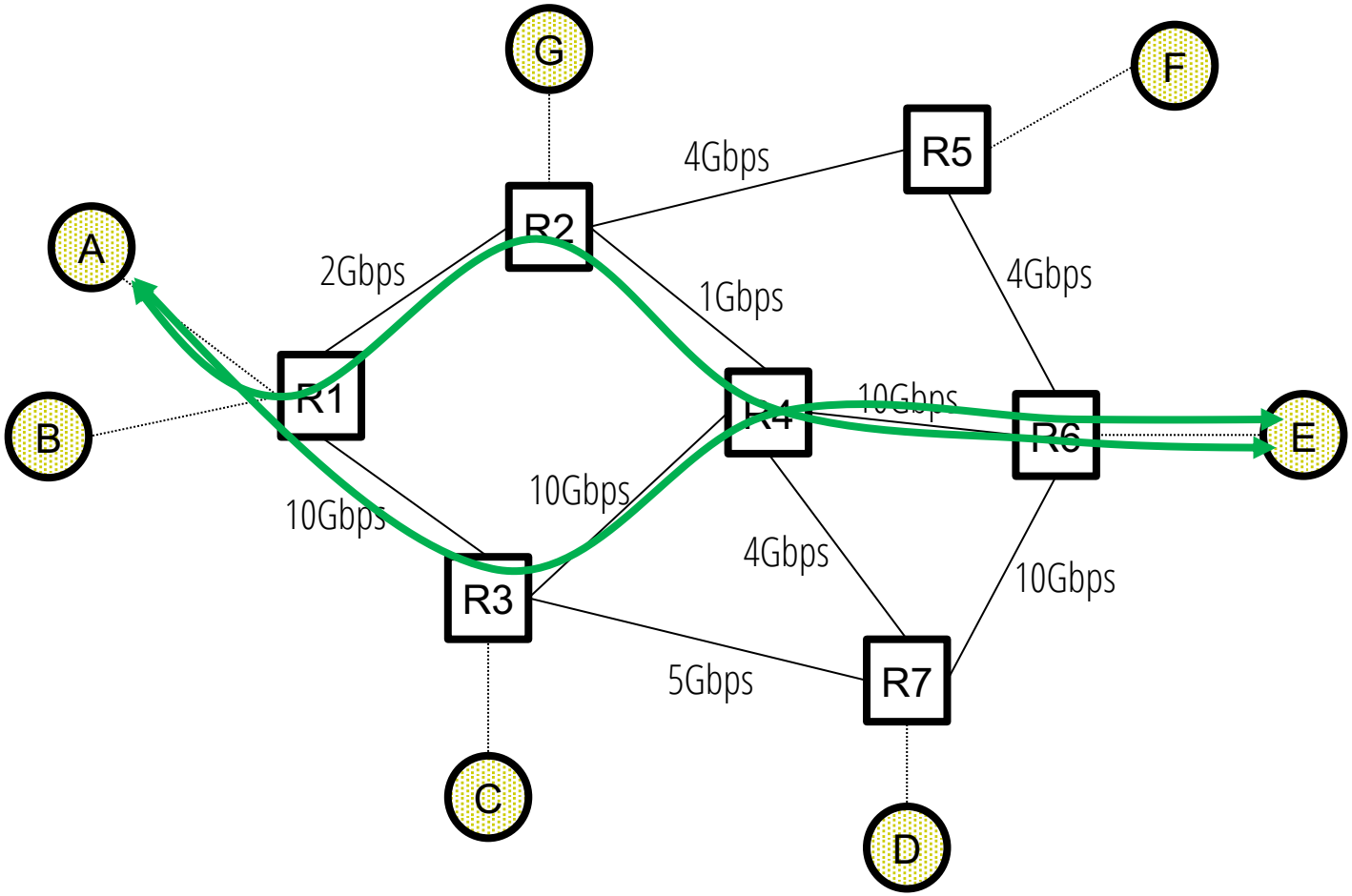
(1) Reservations

(2) Pricing / priorities

(3) Dynamic Adjustment

In practice, the **generality** of dynamic adjustment has proven powerful

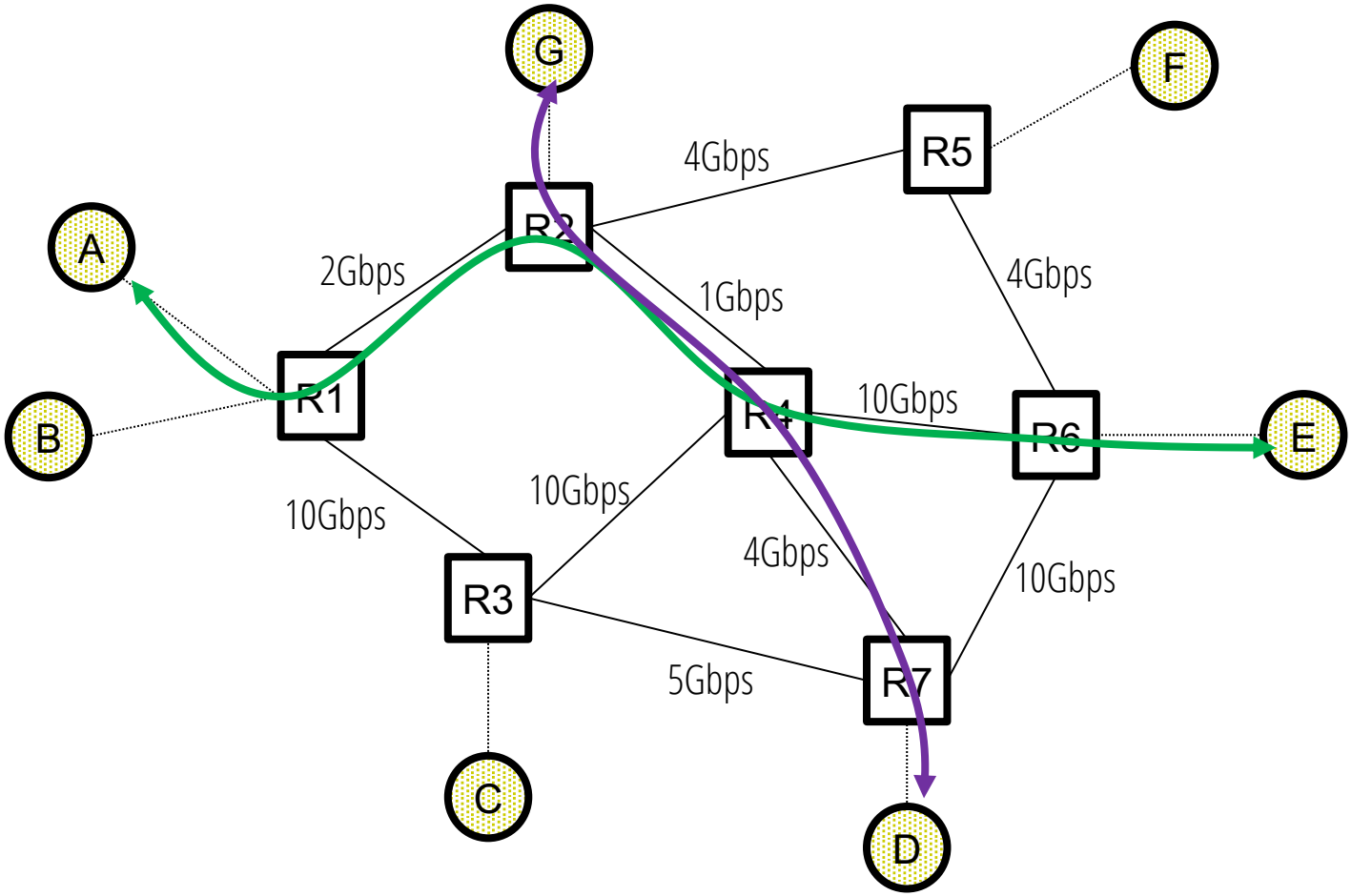
- Doesn't presume business model
- Doesn't assume we know app/user requirements
- But does assume good citizenship!



(1) First, host A discovers it can send at ~10Gbps

(2) A notices that ~10Gbps is congesting the network

(3) A figures out it should cut its rate to ~1Gbps



(4) A notices that 1Gbps is congesting the network

(5) A figures out it should cut its rate to (say) ½ Gbps

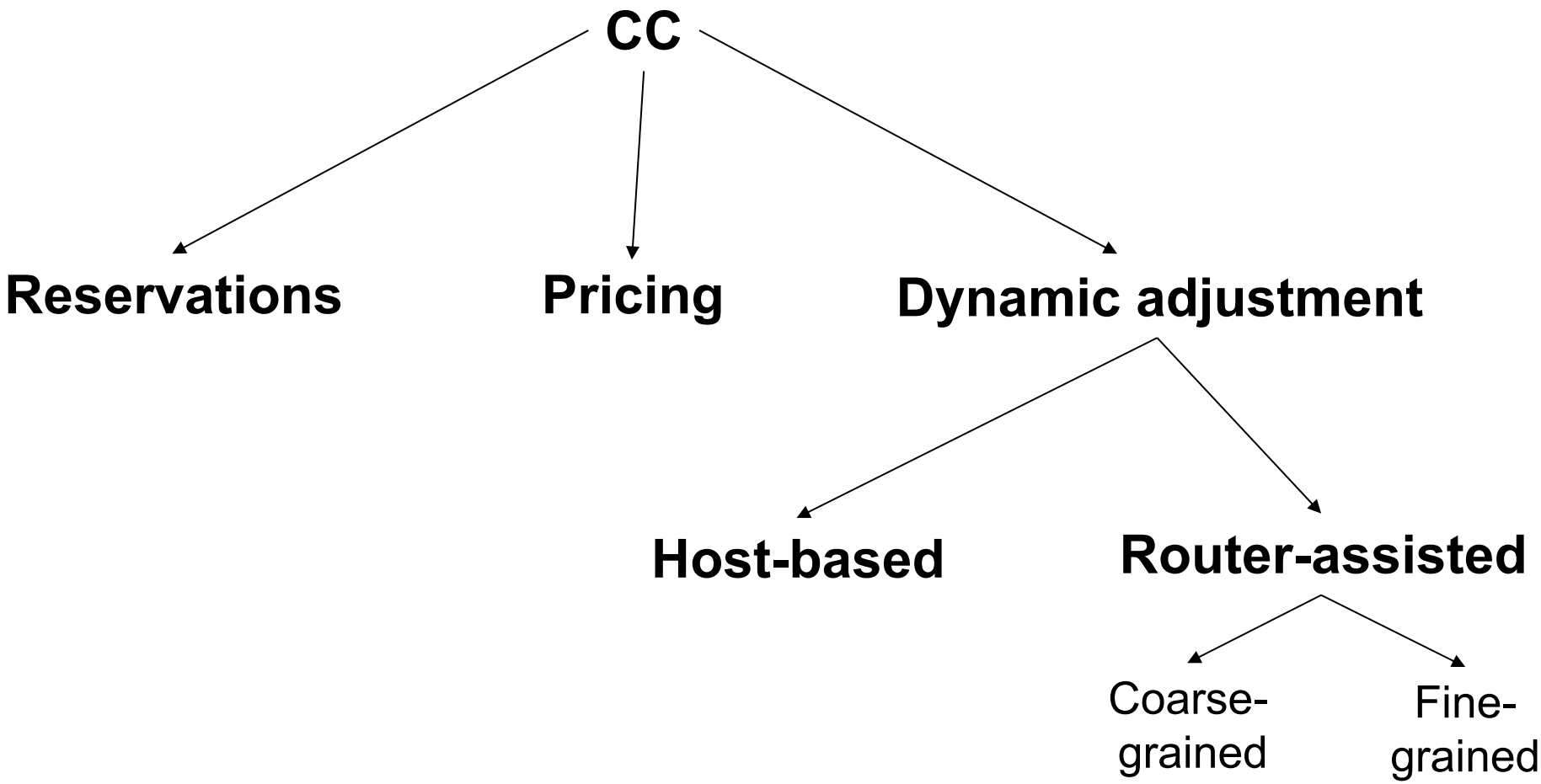
Two broad classes of solutions

- **Host-based CC** → **Jacobson's original TCP approach**
  - No special support from routers
  - Hosts adjust rate based on implicit feedback from routers

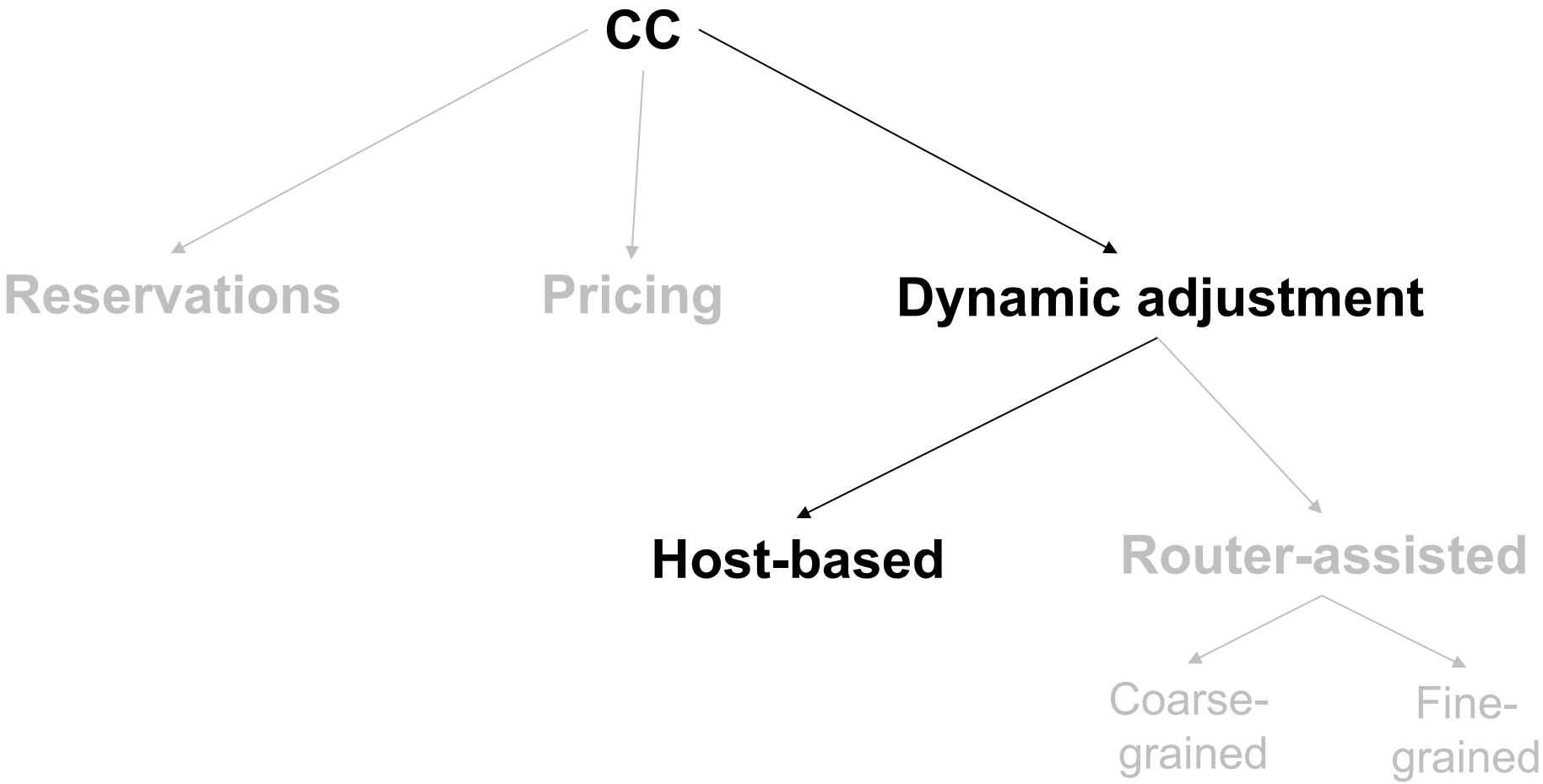
- **Router-assisted CC**
  - Routers signal congestion back to hosts
  - Hosts pick rate based on explicit feedback from routers

- We'll study TCP's host-based approach in detail and a bit of router-assisted CC

Taking stock:
where we are in the design space



Taking stock:
where we are in the design space



Sketch of a (host-based) solution

runs the following:

How do we pick the initial rate?

How do we detect congestion

- Pick initial rate R
- Try sending at a rate R f…ome period of time
  - Did I experience congestion in this time period?
    - If yes, reduce R
    - If no, increase R
  - Repeat

By how much should we increase/decrease

Components of a Solution

- Discovering an initial rate
- Detecting congestion
- Reacting to congestion (or lack thereof)
  - Increase/decrease rules

Detecting Congestion?

- **Packet loss**
  - Approach commonly used by TCP
- **Benefits**
  - Fail-safe signal
  - Already something TCP detects to implement reliability
- **Cons**
  - Complication: non-congestive loss (e.g., checksum err.)
  - Complication: reordering (e.g., with cumulative ACKs)
  - Detection occurs after packets have experienced delay

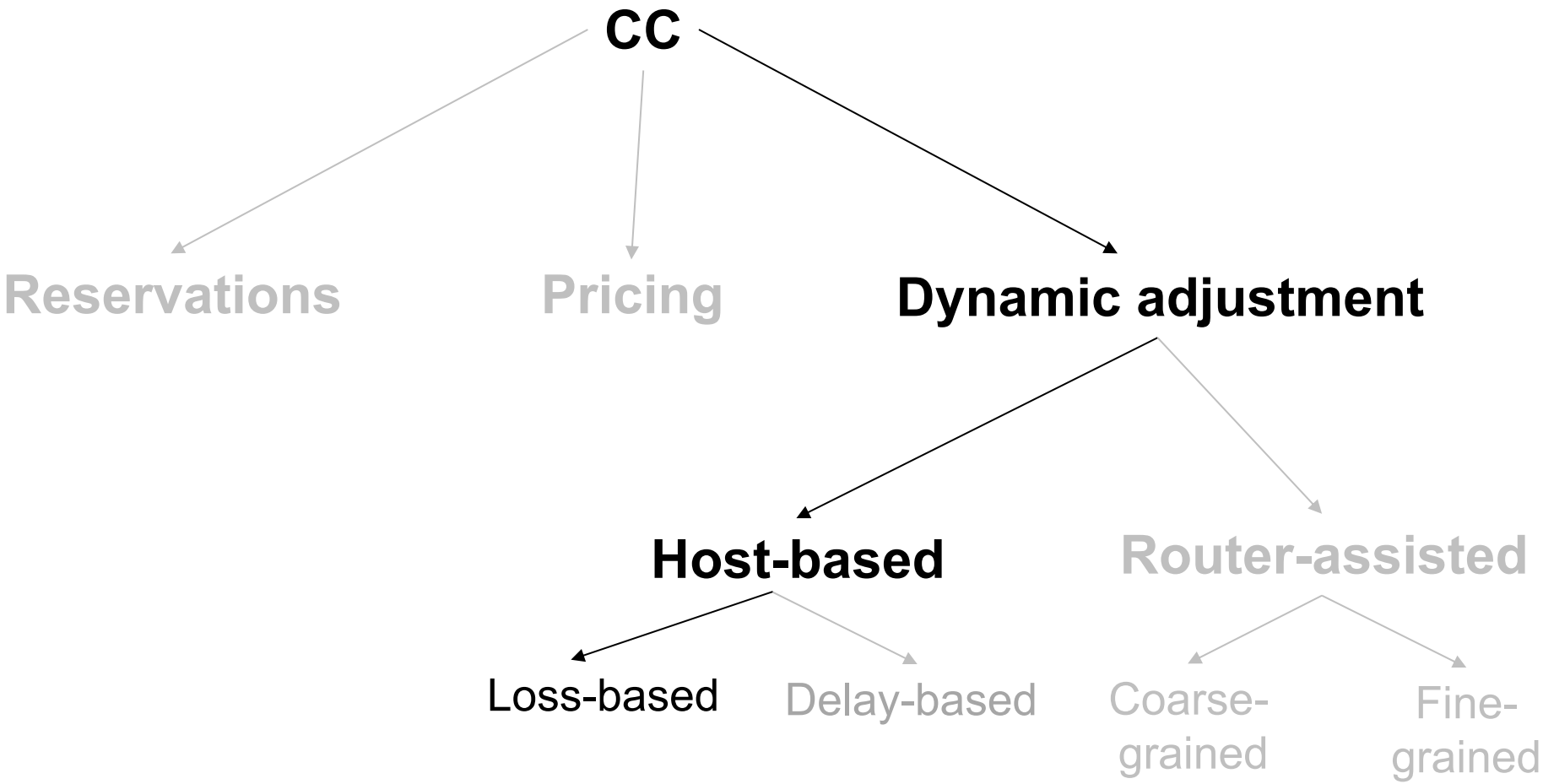
Detecting Congestion?

- **Increase in packet delay**
  - Long considered tricky to get right: packet delay varies with queue size and competing traffic
  - Google's new BBR protocol is now challenging this assumption (next week)

Note: Not All Losses the Same

- **Duplicate ACKs**: isolated loss
  - Packets and ACKs still getting through
  - Suggests mild congestion levels
- **Timeout**: much more serious
  - Not enough packets/dupACKs getting through
  - Must have suffered several losses
- We'll see that TCP reacts differently in each case

Taking stock:
where we are in the design space



Discovering an initial rate?

- Goal: estimate available bandwidth
  - Start slow (for safety)
  - But ramp up quickly (for efficiency)

- Toy example (of an inefficient solution)
  - Add ½ Mbps every 100ms until we detect loss
  - If available BW is 1Mbps, will discover rate in 200ms
  - If available BW is 1Gbps, will take 200 seconds
  - Either is possible!

Solution: “Slow Start”

- Start with a small rate (hence the name)
  - Might be much less than actual bandwidth
  - Linear increase takes too long to ramp up
- Increase exponentially until first loss
  - E.g., double rate until first loss
- A ”safe” rate is half of that when first loss occurred
  - I.e., if first loss occurred at rate R, then R/2 is safe rate

Components of a Solution

- Discovering an initial rate
- Detecting congestion
- Reacting to congestion (or lack thereof)
  - Increase/decrease rules

Sketch of a solution

Each source independently runs the following:

- Pick initial rate R
- Try sending at a rate R for some time period
  - Did I experience congestion in this time period?
    - If yes, reduce R
    - If no, increase R
  - Repeat

By how much should we increase/decrease?

Rate adjustment

- This is a critical part of a CC design!
- Determines how quickly a host adapts to changes in available bandwidth
- Determines how effectively BW is consumed
- Determines how BW is shared (fairness)

Goals for rate adjustment

- **Efficiency**: High utilization of link bandwidth
- **Fairness**: Each flow gets equal share

How should we adjust rate?

- Infinite options...
- At the highest level: fast or slow
- Fast: **multiplicative** increase/decrease
  - E.g., increase/decrease by 2x (R → 2R or R/2)
- Slow: **additive** increase/decrease
  - E.g., increase/decrease by +1 (R→ R+1 or R-1)

Leads to four alternatives

- **AIAD**: gentle increase, gentle decrease
- **AIMD**: gentle increase, rapid decrease
- **MIAD**: rapid increase, gentle decrease
- **MIMD**: rapid increase, rapid decrease

Leads to four alternatives

- **AIAD**: gentle increase, gentle decrease
- **AIMD**: gentle increase, rapid decrease
- **MIAD**: rapid increase, gentle decrease
- **MIMD**: rapid increase, rapid decrease

Why AIMD? Intuition

- Consequences of sending too much are worse than sending too little
  - Too much: packets dropped and retransmitted
  - Too little: somewhat lower throughput

- General approach:
  - Gentle increase when uncongested (exploration)
  - Rapid decrease when congested

Why AIMD? In more detail...

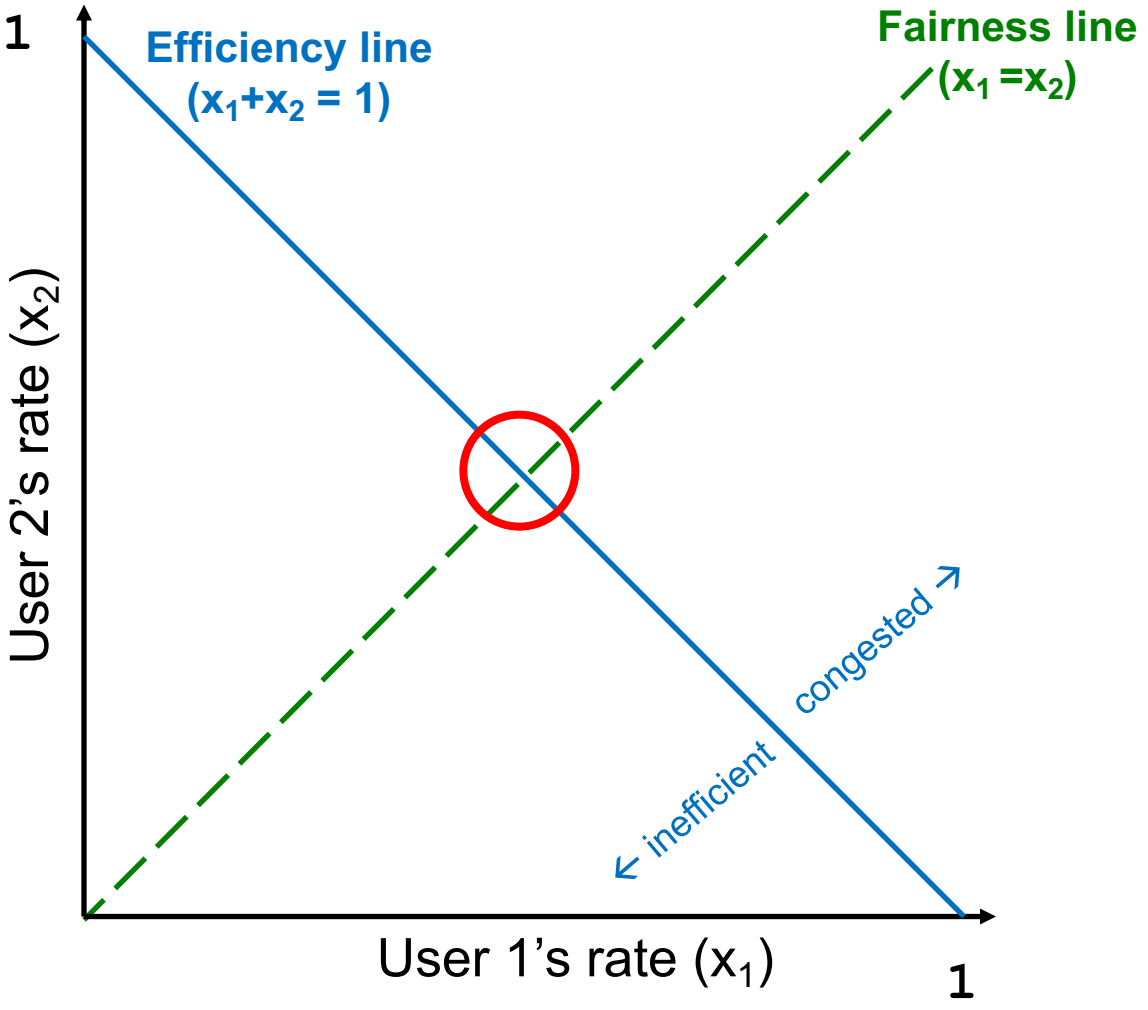
- Consider a simple model
  - Two flows going over single link of capacity C
  - Sending at rates X1 and X2 respectively

- When X1+X2 > C, network is congested
- When X1+X2 < C, network is underloaded

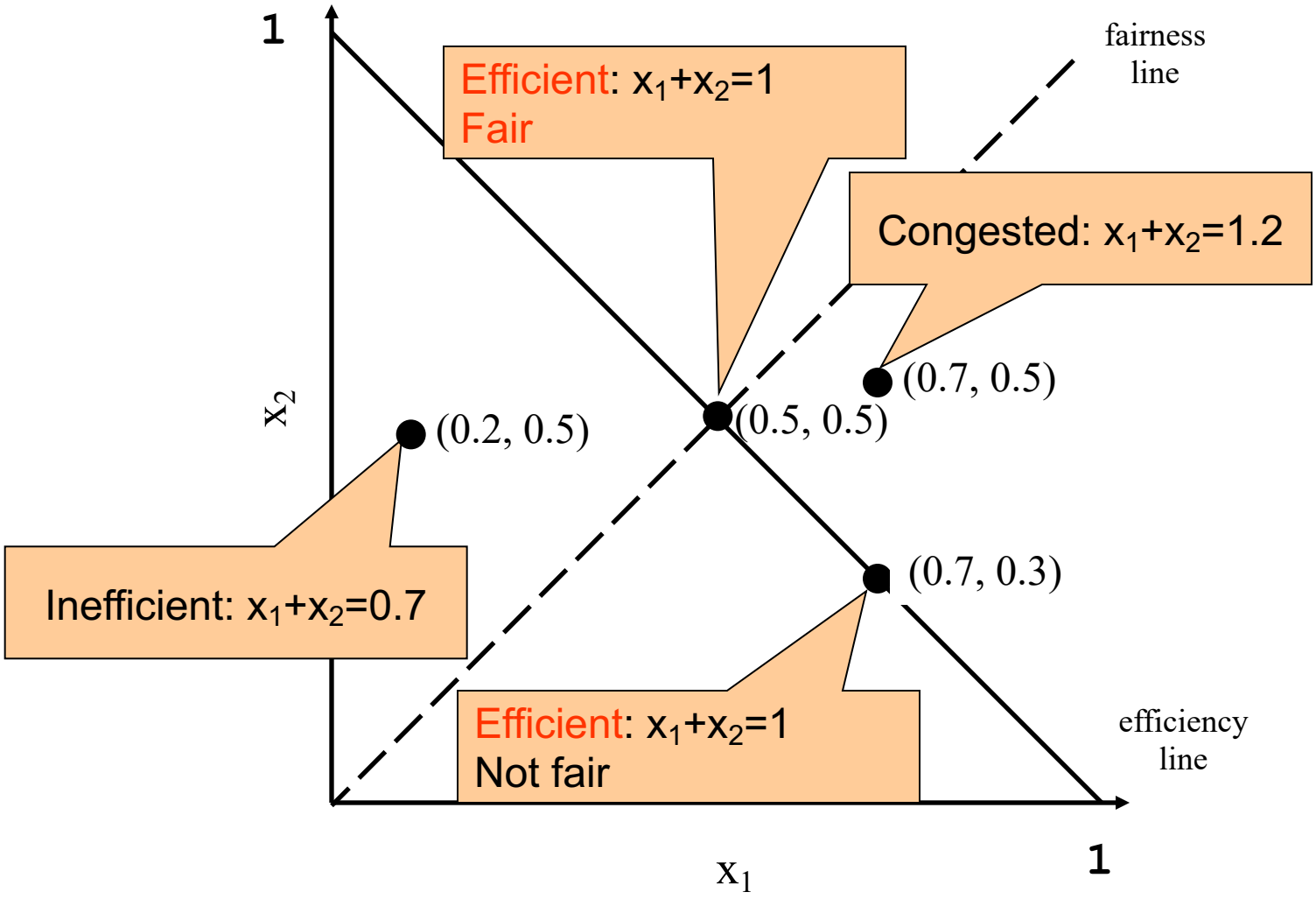
- Would like *both*:
  - X1 + X2 = C → link is fully utilized with no congestion
  - X1 = X2 → sharing is “fair”

Simple Model, C=1

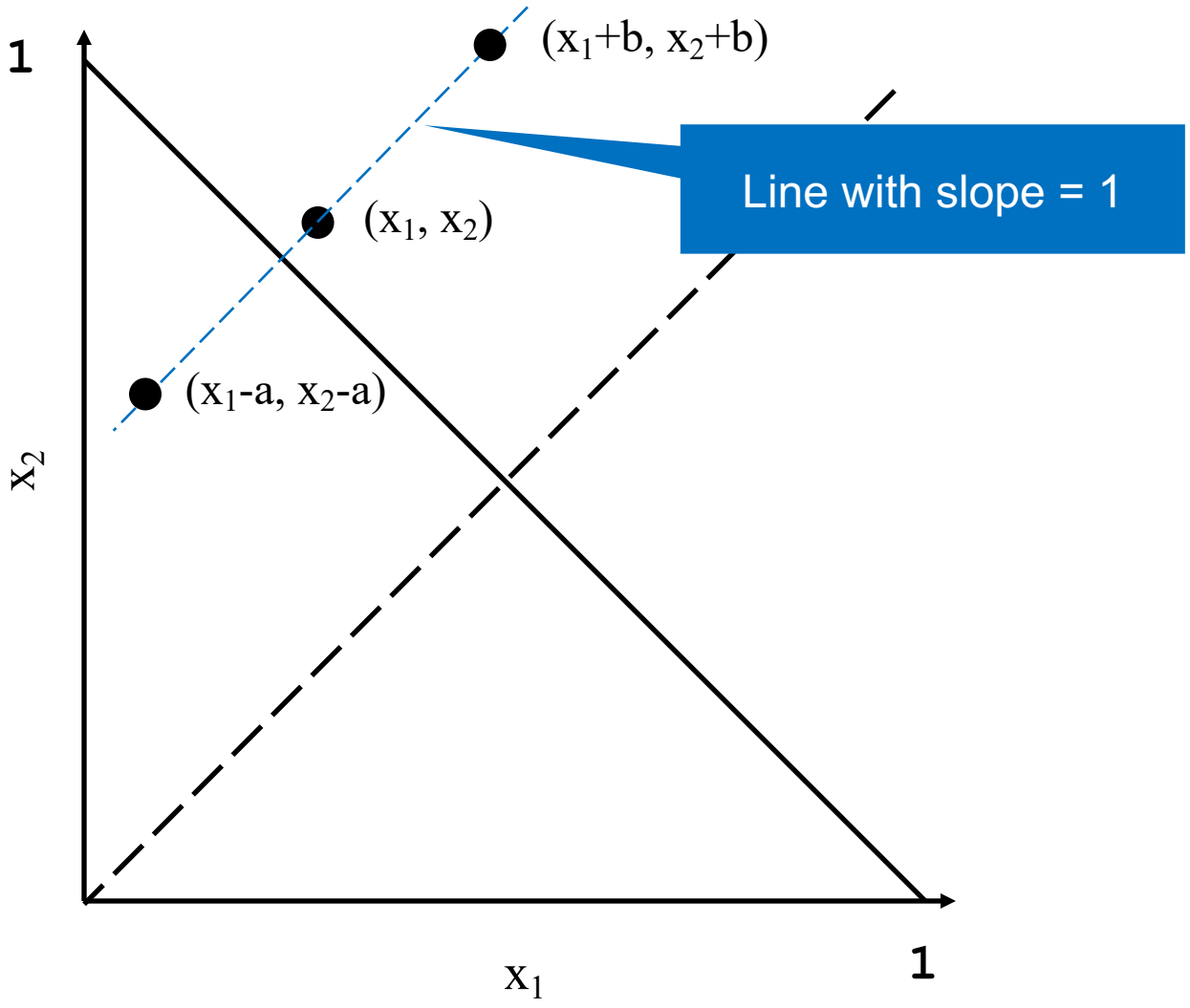
- Two users with rates x_1 and x_2
- Congestion when $x_1+x_2 > 1$
- Unused capacity when $x_1+x_2 < 1$
- Fair when $x_1 = x_2$



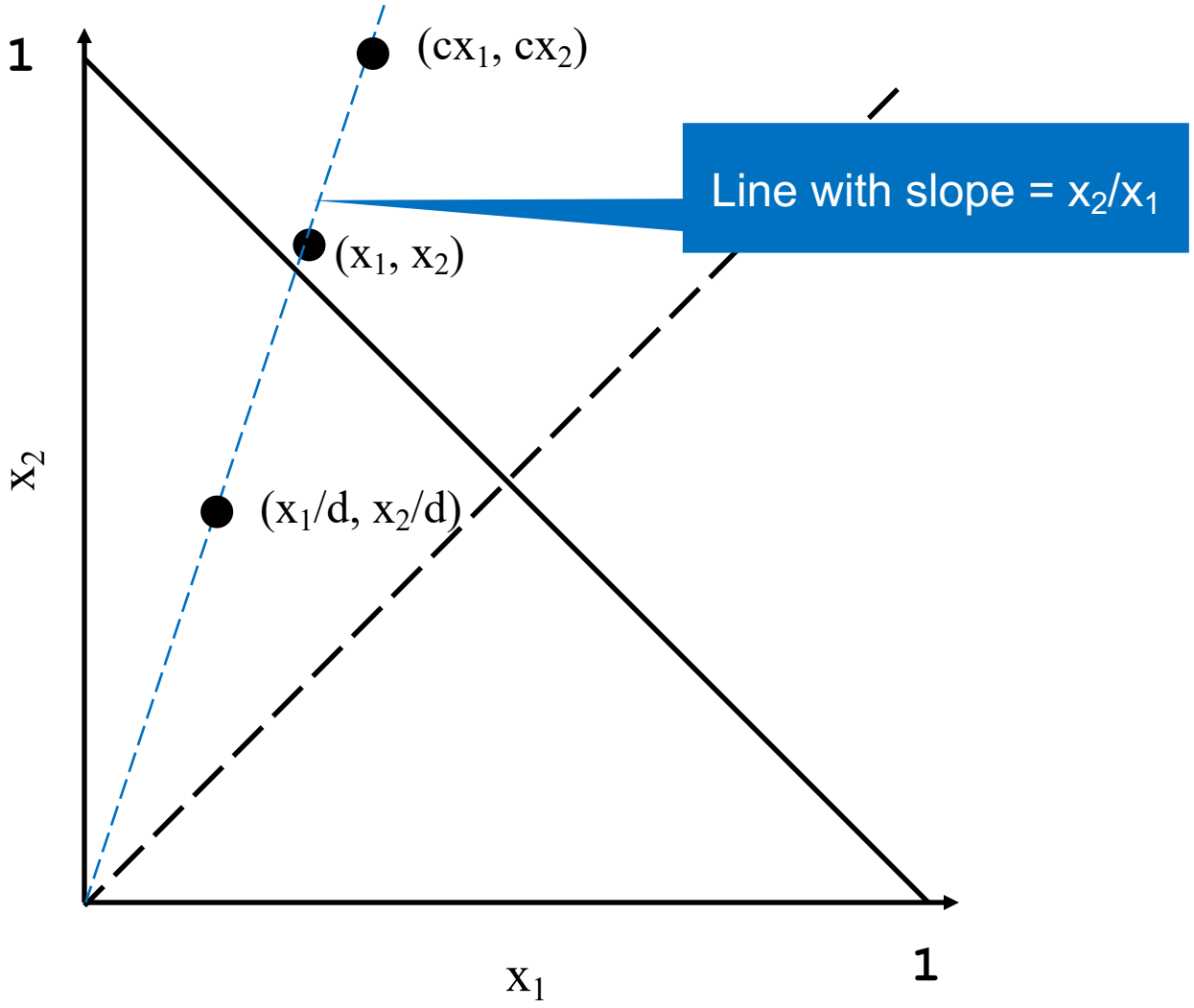
Example Allocations, C=1



Example Adjustments



Example Adjustments



Our Four Options

- AIAD: gentle increase, gentle decrease
- AIMD: gentle increase, rapid decrease
- MIAD: rapid increase, gentle decrease
- MIMD: rapid increase, rapid decrease

- **And now apply our simple model!**

AIAD Dynamics

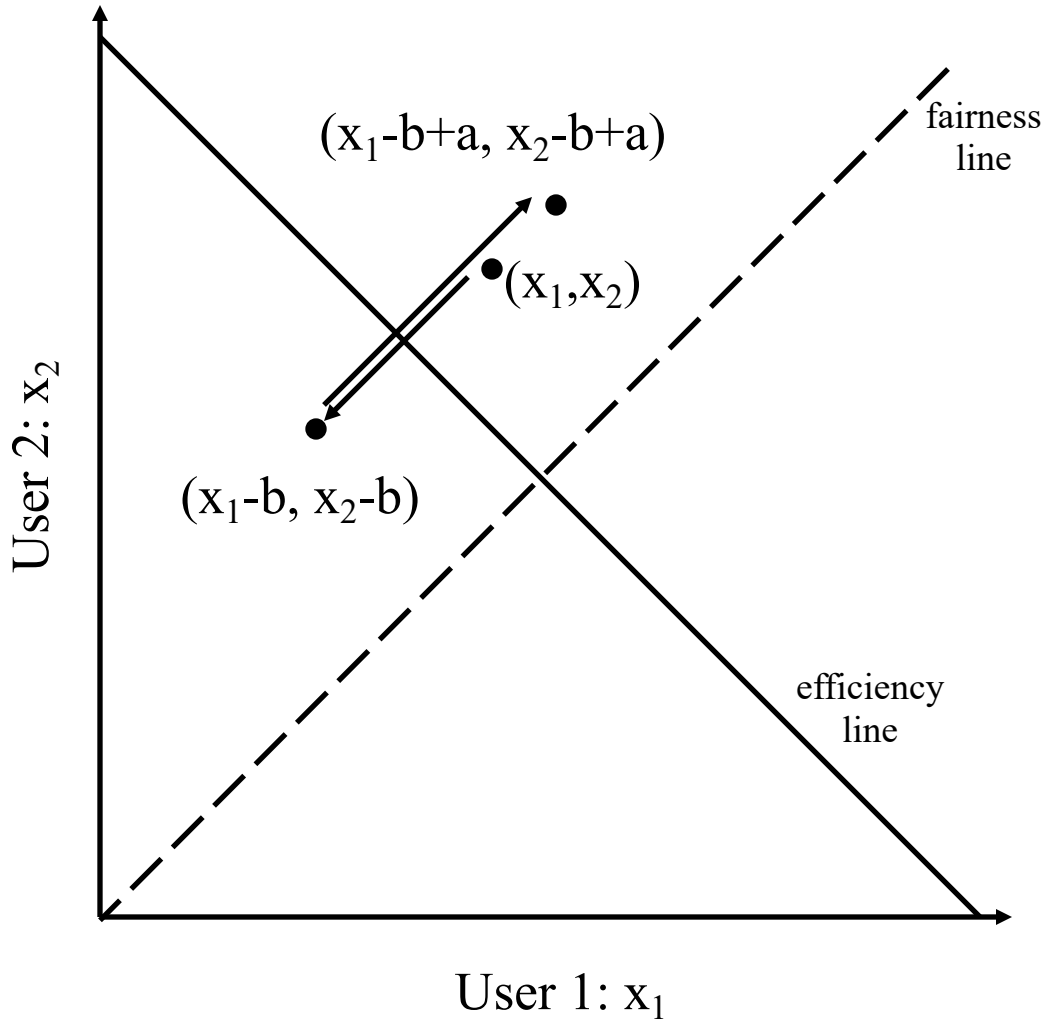
- Consider: Increase: +1 Decrease: -2
- Start at X1 = 1, X2 = 3, with C = 5
- First iteration: no congestion
  - X1 → 2, X2 → 4
- Second iteration: congestion
  - X1 → 0, X2 → 2
- Third iteration: no congestion
  - X1 → 1, X2 → 3
- …

Back where we started!
→ Gap between X1 and X2 didn't change at all

AIAD

- Increase: x + a
- Decrease: x - b

- Does not converge to fairness



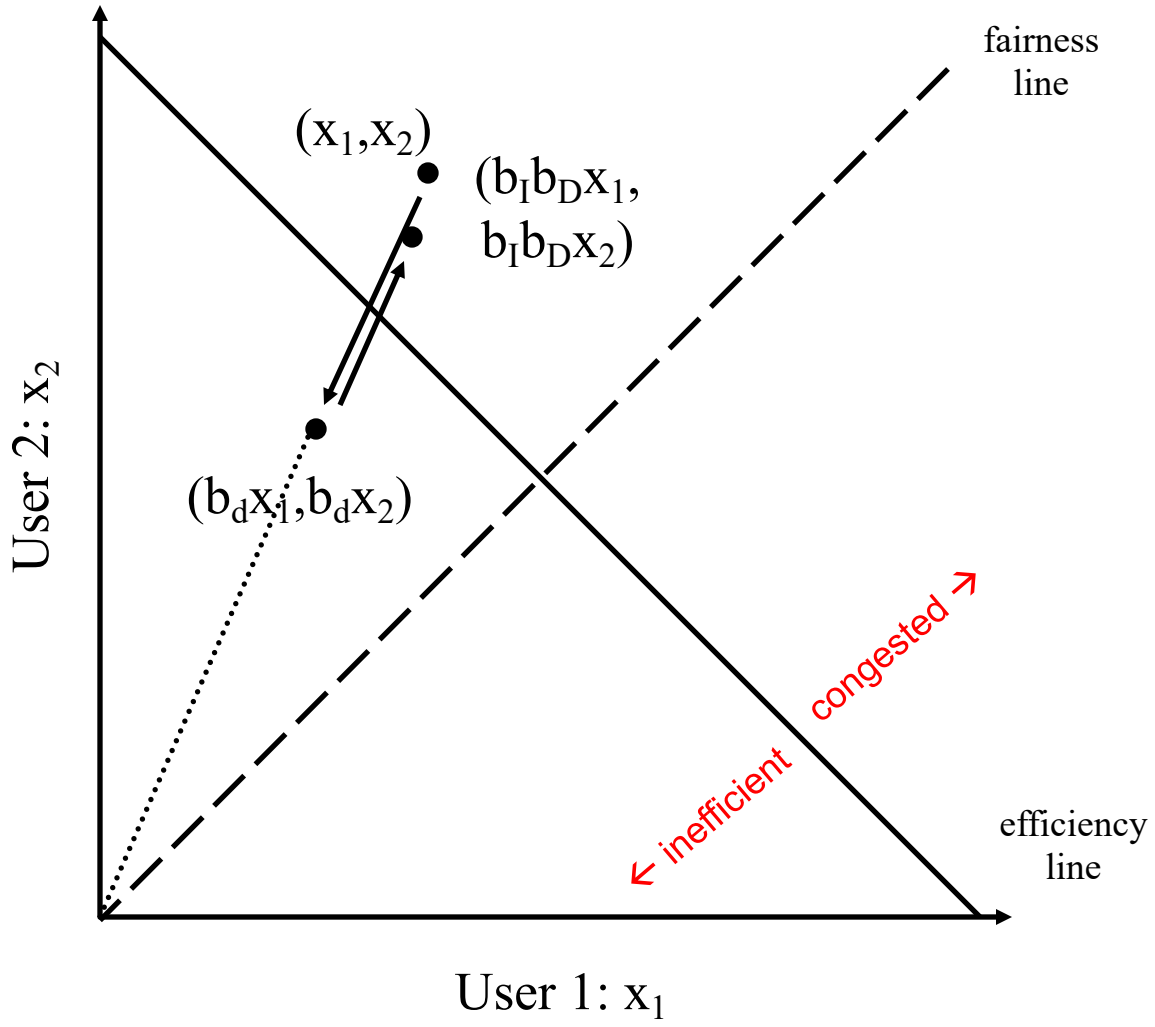
MIMD Dynamics

- Consider: Increase: $\times 2$ Decrease: $\div 4$
- Start at X1 = ½, X2 = 1, with C = 5
- First iteration: no congestion
  - X1 → 1, X2 → 2
- Second iteration: no congestion
  - X1 → 2, X2 → 4
- Third iteration: congestion
  - X1 → ½ , X2 → 1
- … **Again, no improvement in fairness**

MIMD

- Increase: $x \times b_I$
- Decrease: $x \times b_D$

- Does not converge to fairness



MIAD Dynamics

- Consider: Increase: $\times 2$ Decrease: -1
- Start at X1 = 1, X2 = 3, with C = 5

- First iteration: no congestion; X1 $\rightarrow$ 2, X2 $\rightarrow$ 6
- Second iteration: congestion; X1 $\rightarrow$ 1, X2 $\rightarrow$ 5
- Third iteration: congestion; X1 $\rightarrow$ 0, X2 $\rightarrow$ 4
- Fourth iteration: no congestion; X1 $\rightarrow$ 0, X2 $\rightarrow$ 8

X1 pegged at 0; MIAD is maximally unfair!

AIMD Dynamics

- Consider: Increase: $+1$ Decrease: $\div 2$
- Start at X1 = 1, X2 = 2, with C = 5 Diff = 1

- First iteration: no congestion: X1 → 2, X2 → 3 Diff = 1
- Second: no congestion: X1 → 3, X2 → 4 Diff = 1
- Third: congestion: X1 → 1.5, X2 → 2 Diff = 0.5
- Fourth: no congestion: X1 → 2.5, X2 → 3 Diff = 0.5
- Fifth: congestion: X1 → 1.25, X2 → 1.5 Diff = 0.25
- Sixth: no congestion: X1 → 2.25, X2 → 2.5 Diff = 0.25
- Seventh: no congestion: X1 → 3.25, X2 → 3.5 Diff = 0.25
- Eighth: congestion: X1 → 1.625, X2 → 1.75 Diff = 0.125
- Ninth: no congestion: X1 →2.625, X2→ 2.75 Diff = 0.125

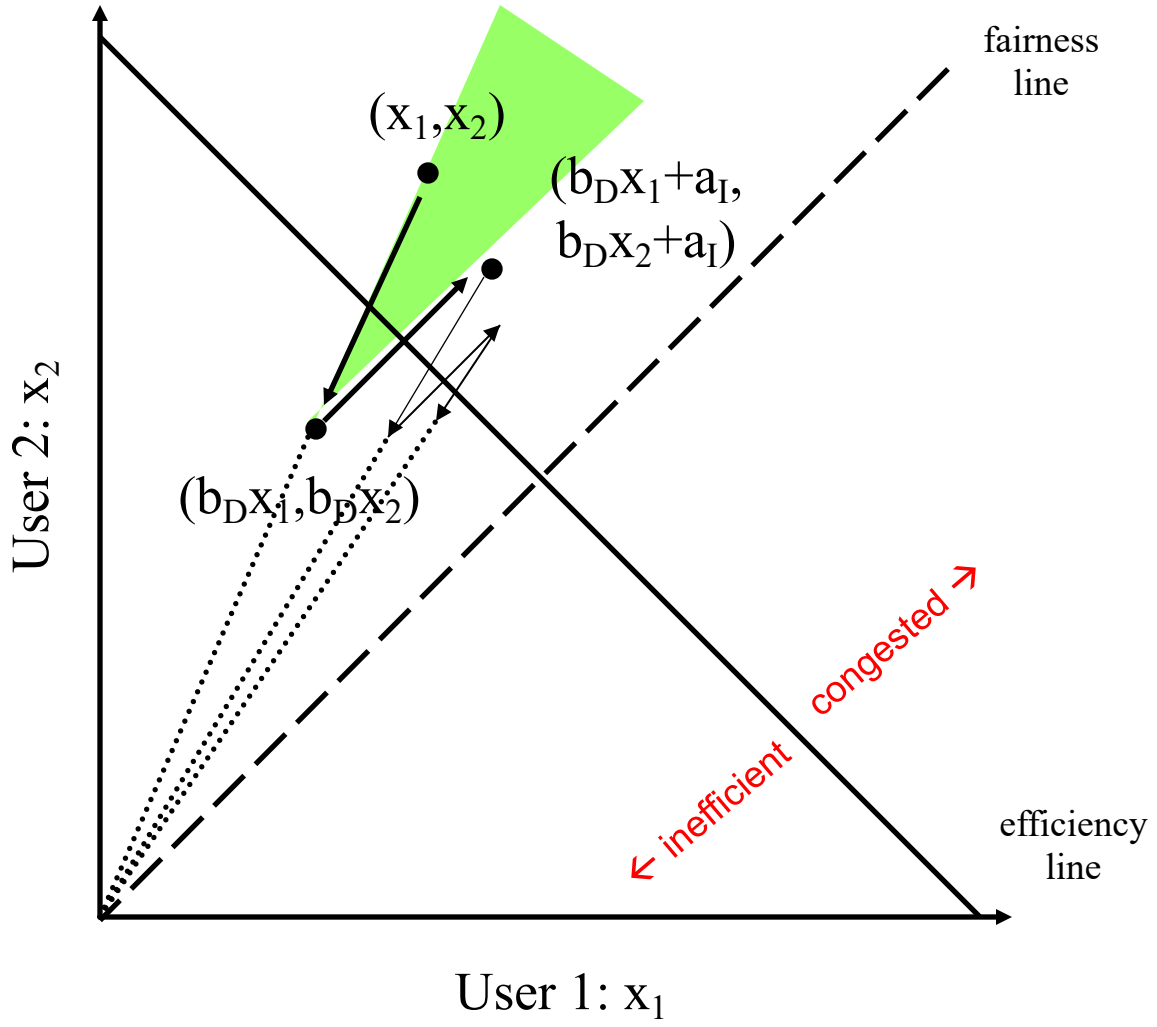
AIMD

- Difference between X1 and X2 decreasing!
  - Difference stays constant when increasing
  - Halves every time there is a decrease

AIMD

- Increase: $x+a_I$
- Decrease: $x*b_D$

- **Converges to fairness**



Answer to Why AIMD?

- AIMD embodies gentle increase, rapid decrease
- AIMD only choice that drives us towards “fairness”
- Out of the four options
  - AIAD, MIMD: retain unfairness
  - MIAD: maximally unfair
  - AIMD: fair and appropriate gentle/rapid actions

Any Questions?

Sketch of a solution

Each source independently runs the following:

- Pick initial rate R
- Try sending at a rate R for some time period
  - Did I experience congestion in this time period?
    - If yes, reduce R
    - If no, increase R
  - Repeat

Sketch of TCP's solution

Each source independently runs the following:

- Pick initial rate R
- Try sending at a rate R for some time period
  - Did I experience congestion in this time period?
    - If yes, reduce R
    - If no, increase R
  - Repeat

Sketch of TCP's solution

Each source independently runs the following:

- Slow-start to find initial rate
- Try sending at a rate R for some time period
  - Did I experience congestion in this time period?
    - If yes, reduce R
    - If no, increase R
  - Repeat

Sketch of TCP's solution

Each source independently runs the following:

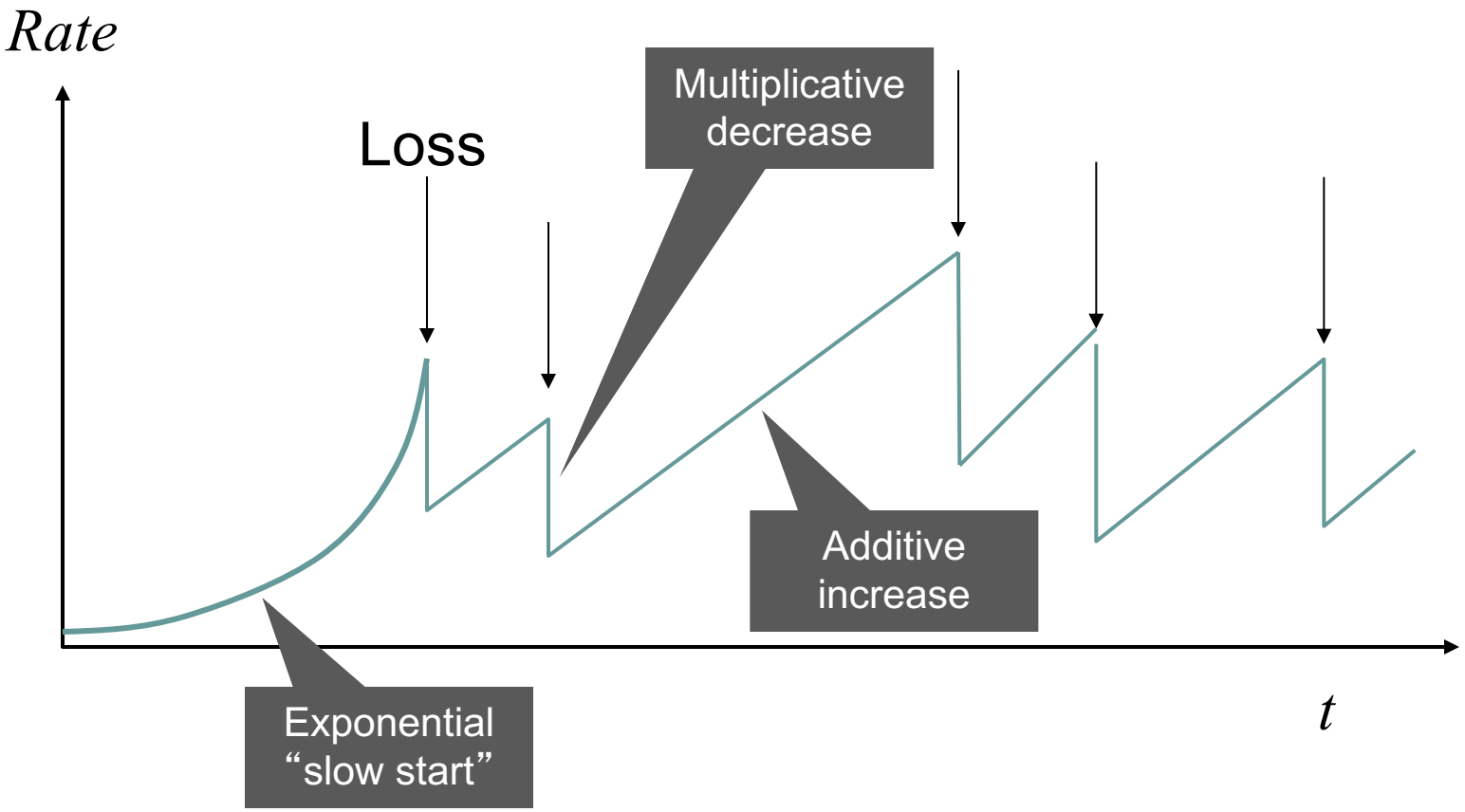
- Slow-start to find initial rate
- Try sending at a rate R for some time period
  - Did I experience ~~congestion~~ loss in this time period?
    - If yes, reduce R
    - If no, increase R
  - Repeat

Sketch of TCP's solution

Each source independently runs the following:

- Slow-start to find initial rate
- Try sending at a rate R for some time period
  - Did I experience ~~congestion~~ loss in this time period?
    - If yes, reduce R multiplicatively (2x)
    - If no, increase R additively (+1)
  - Repeat

Leads to the TCP "Sawtooth"



Next time: details of TCP CC

- Our overall approach with a few key differences
  - Based on adjusting window size on timescale of RTT
  - Different reactions for timeouts (severe loss) vs. duplicate ACKs (isolated loss)
  - Slow-start used on timeout as well as at beginning
  - Optimization for the case of isolated loss