Network Applications: Operational Analysis; Load Balancing among Homogeneous Servers

Qiao Xiang, Congming Gao

https://sngroup.org.cn/courses/cnns-xmuf23/index.shtml

10/17/2023

This deck of slides are heavily based on CPSC 433/533 at Yale University, by courtesy of Dr. Y. Richard Yang.

Outline

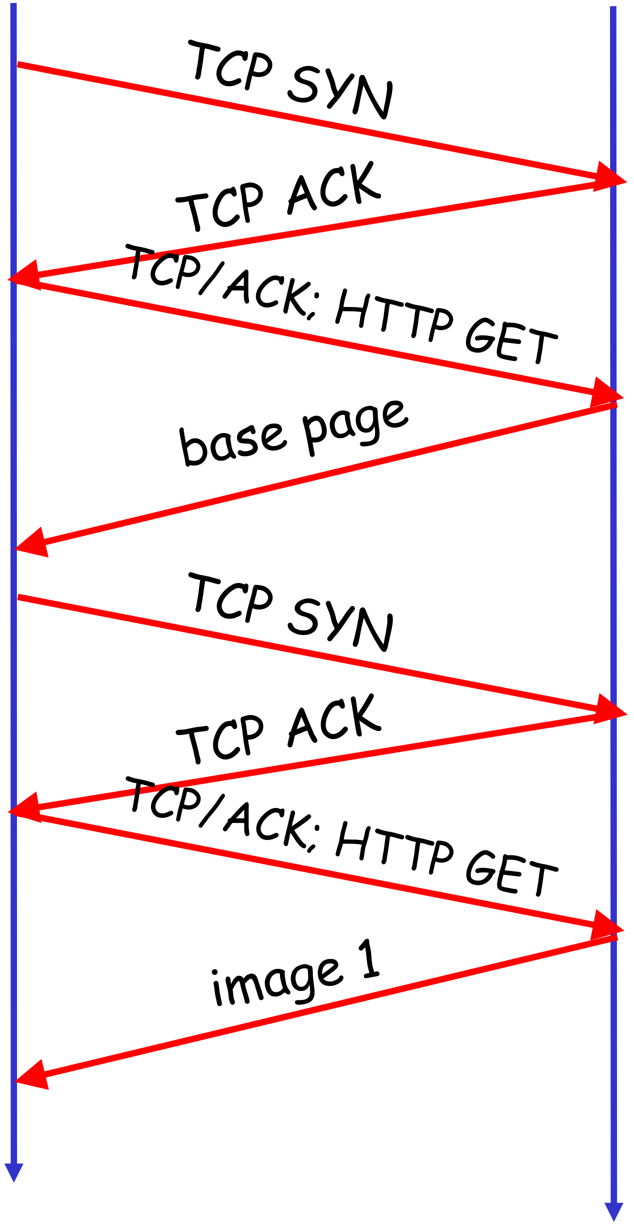
- Admin and recap
- HTTP
  - Basic design: HTTP 1.0
  - HTTP "acceleration"
  - Operational analysis
- Multi-servers
- Application overlays (peer-to-peer networks)

Admin

- Lab assignment 2 due Oct. 19

Recap: Latency of Basic HTTP/1.0

- >= 2 RTTs per object:
  - TCP handshake --- 1 RTT
  - client request and server responds --- at least 1 RTT (if object can be contained in one packet)



Recap: Substantial Efforts to Speedup HTTP/1.0

- Reduce the number of objects fetched [Browser cache]

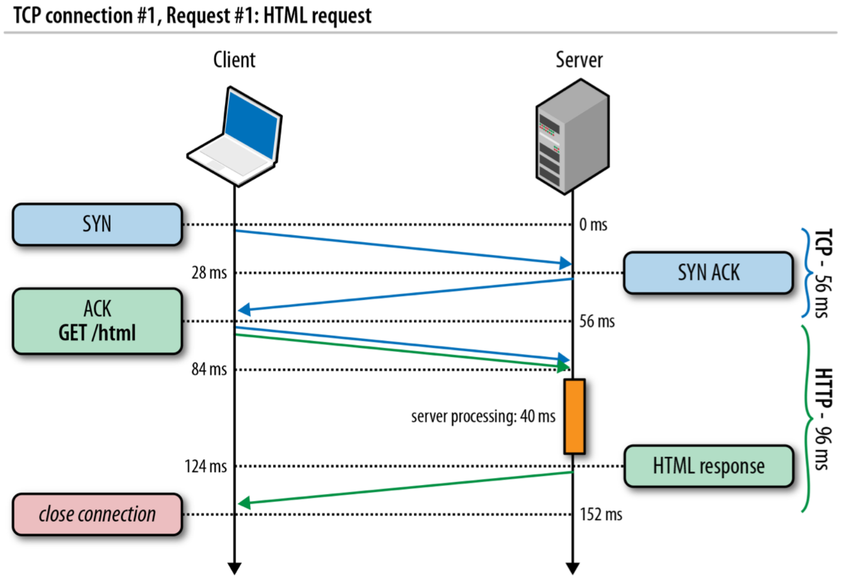
- Reduce data volume [Compression of data]
- Header compression [HTTP/2]

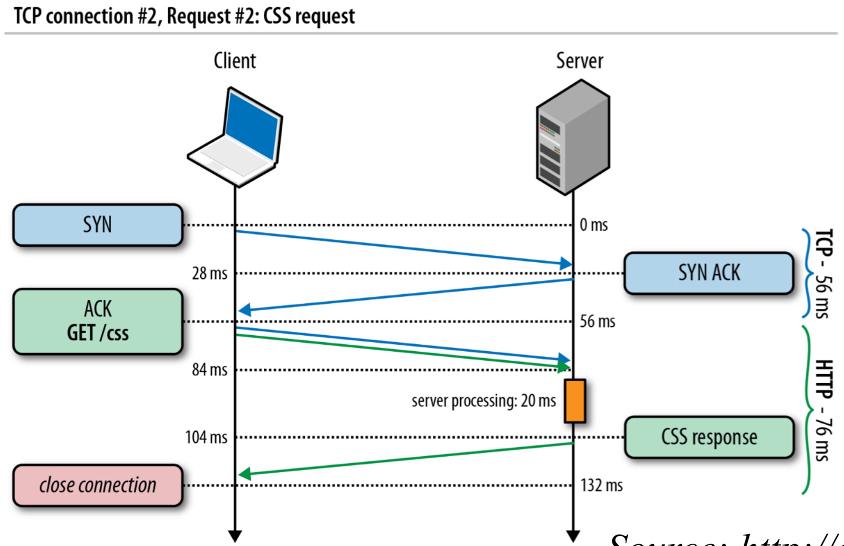
- Reduce the latency to the server to fetch the content [Proxy cache]
- Remove the extra RTTs to fetch an object [Persistent HTTP, aka HTTP/1.1]

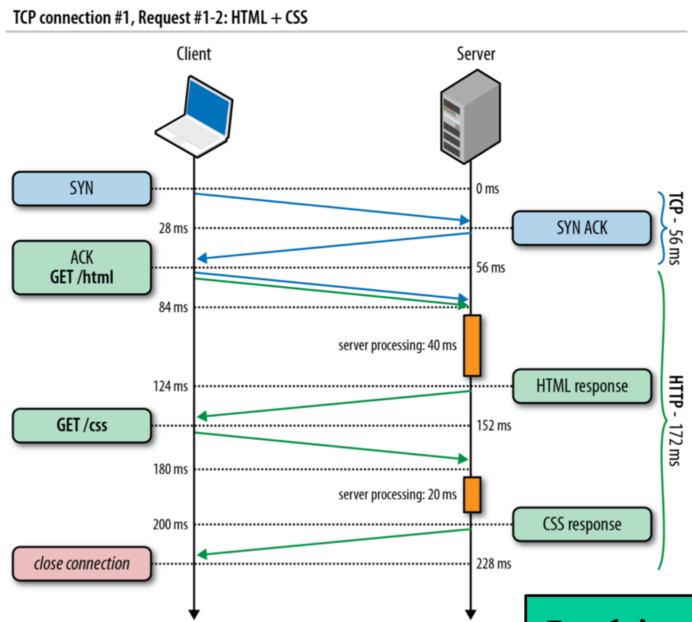
- Increase concurrency [Multiple TCP connections]
- Asynchronous fetch (multiple streams) using a single TCP [HTTP/2]

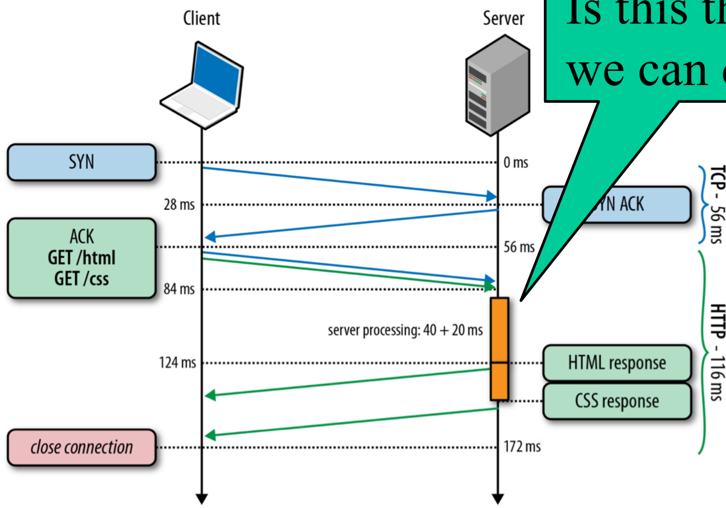
- Server push [HTTP/2]

Recap: HTTP/1.0, Keep-Alive, Pipelining





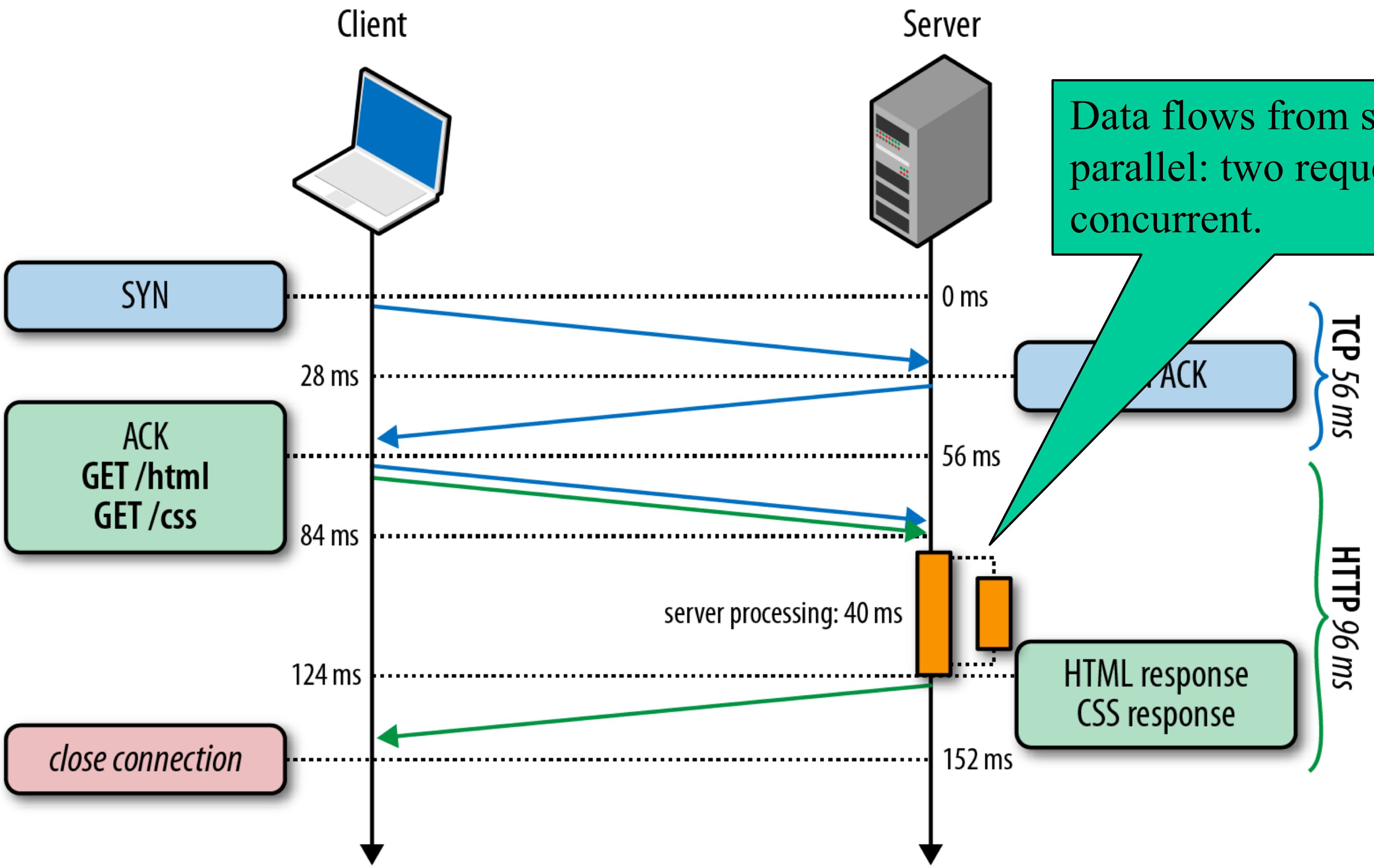




Is this the best we can do?

Source: http://chimera.labs.oreilly.com/books/1230000000545/ch11.html

HTTP/2 Basic Idea: Remove Head-of-Line Blocking in HTTP/1.1



Data flows from sequential to parallel: two requests must be concurrent.

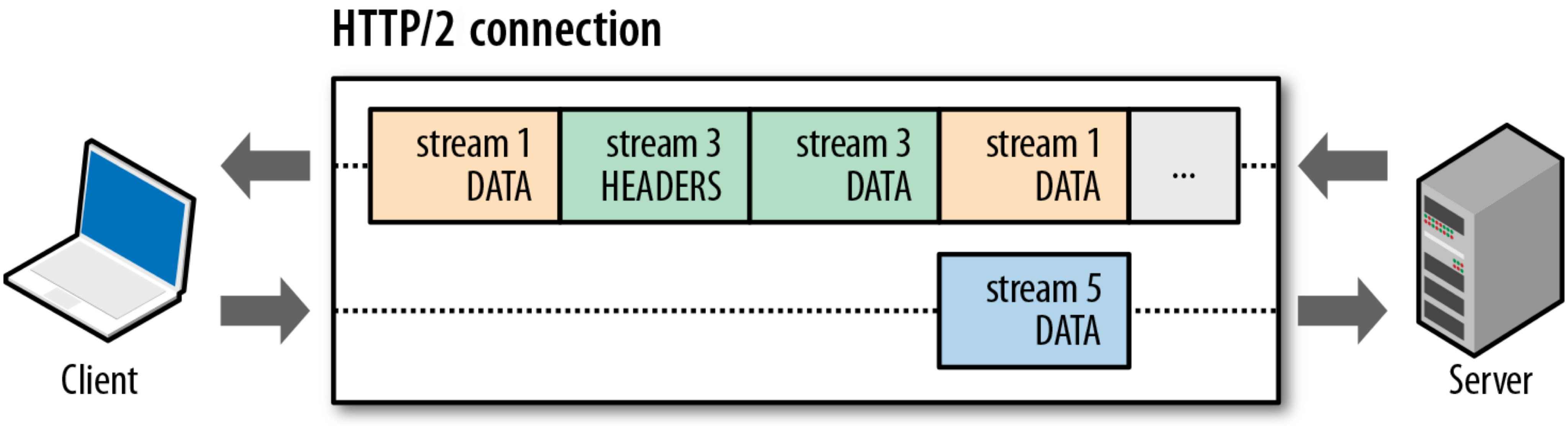
Demo: https://http2.akamai.com/demo

Source: http://chimera.labs.oreilly.com/books/1230000000545/ch11.html

Observing HTTP/2

- export SSLKEYLOGFILE=/tmp/keylog.txt
- Start Chrome, e.g.,
  - Mac: /Applications/Google Chrome.app/Contents/MacOS/Google Chrome
  - Ubuntu: firefox
- Visit HTTP/2 pages, such as https://www.tmall.com
- Wireshark:
  - Mac: Wireshark -> preferences -> protocol -> TSL
    (pre)-master-secret log file name
  - Ubuntu: edit -> perferences -> protocol -> SSL
    (pre)-master-secret log file name

HTTP/2 Design: Multi-Streams



Bit	+0..7	+8..15	+16..23	+24..31
0	Length			Type
32	Flags			
40	R Stream Identifier			
...	*Frame Payload*			

HTTP/2 Binary Framing

https://hpbn.co/http2/ https://tools.ietf.org/html/rfc7540

HTTP/2 Header Compression

Request headers

:method	GET
:scheme	https
:host	example.com
:path	/resource
user-agent	Mozilla/5.0 …
custom-hdr	some-value

Static table

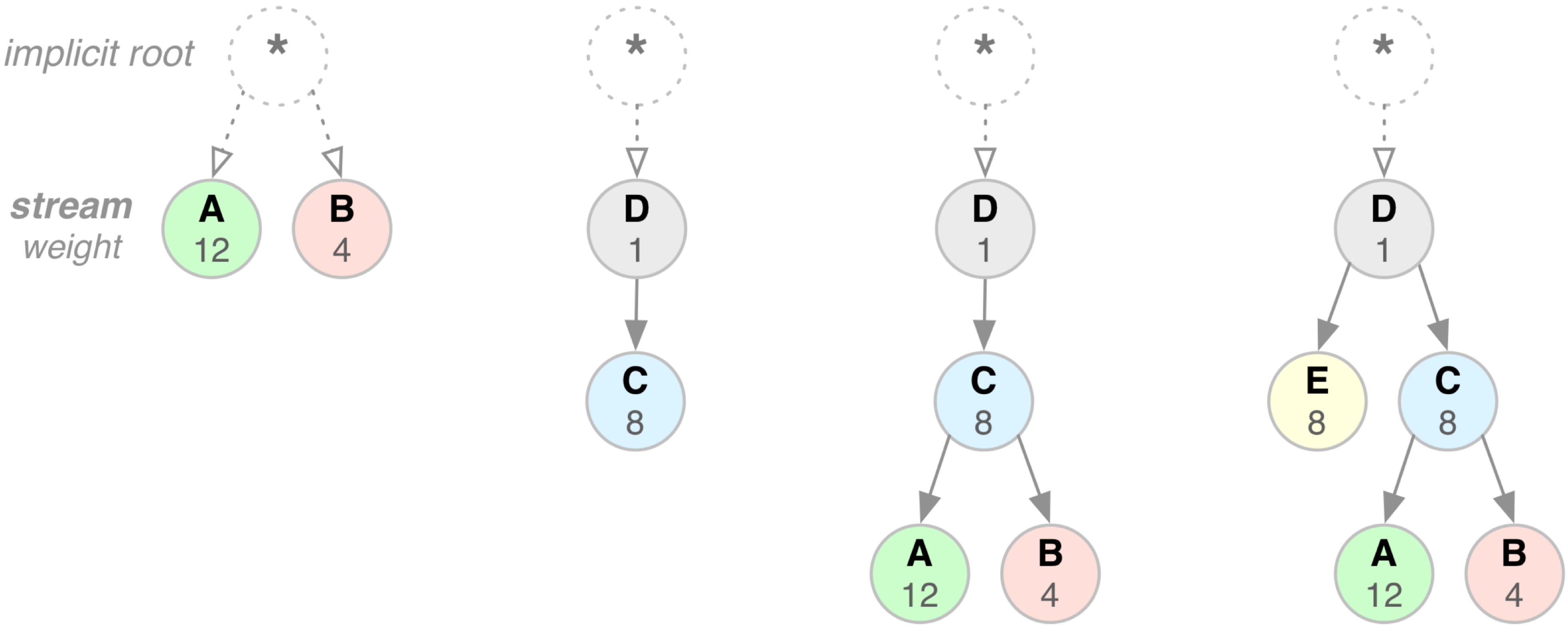
1	:authority	
2	:method	GET
…	…	…
51	referer	
…	…	…
62	user-agent	Mozilla/5.0 …
63	:host	example.com
…	…	…

Dynamic table

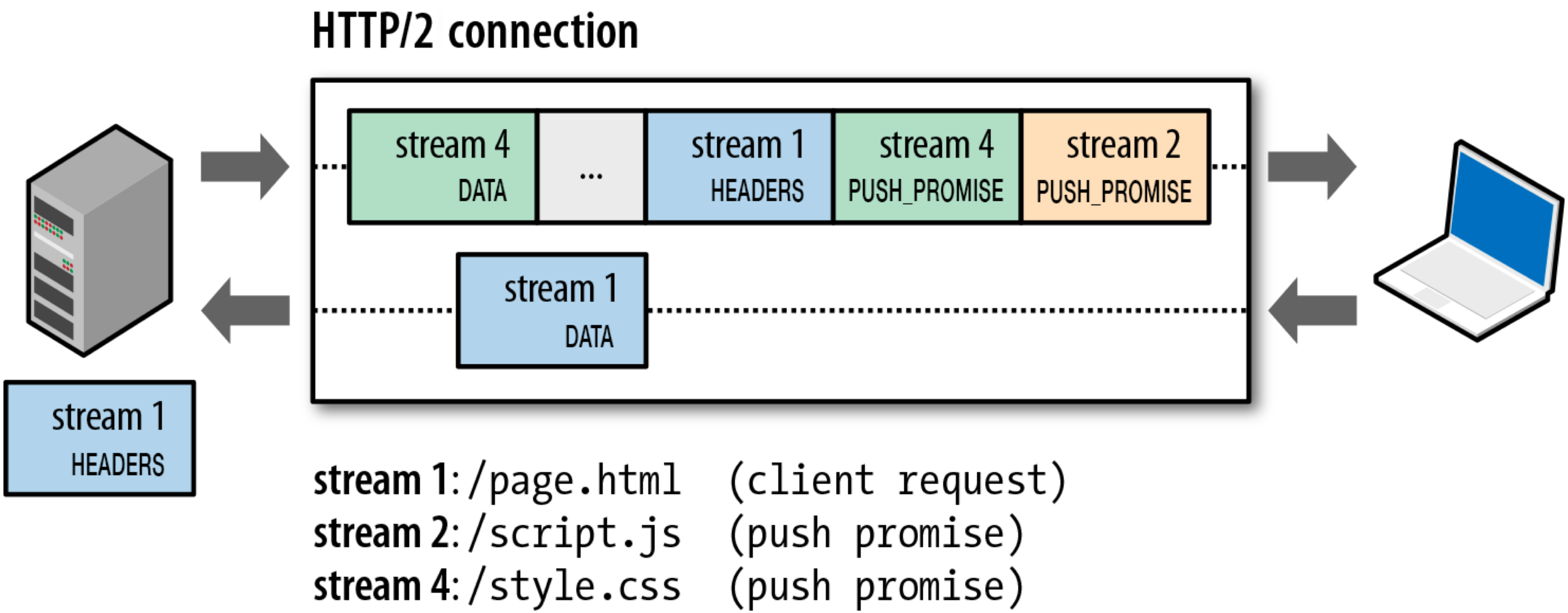
Encoded headers

2	
7	
63	
19	Huffman("/resource")
62	
Huffman("custom-hdr")	
Huffman("some-value")	

HTTP/2 Stream Dependency and Weights



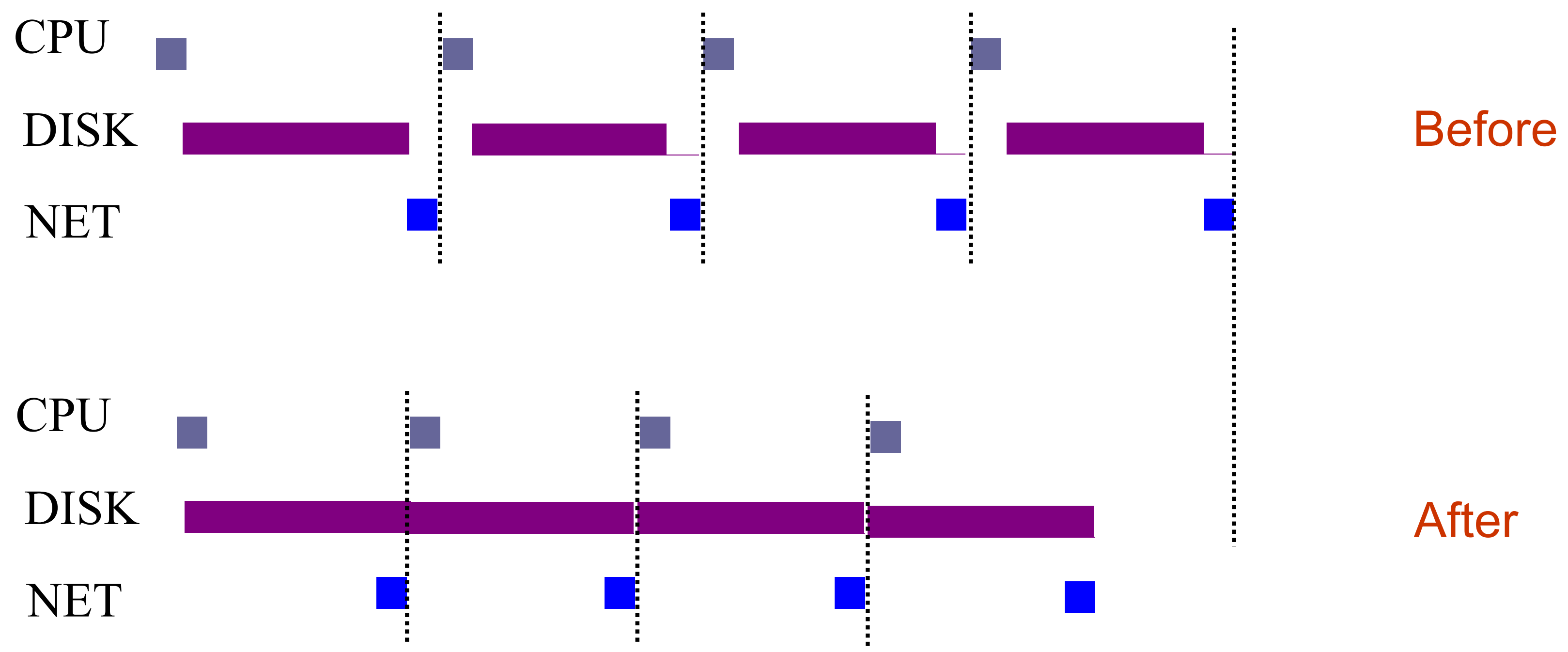
HTTP/2 Server Push

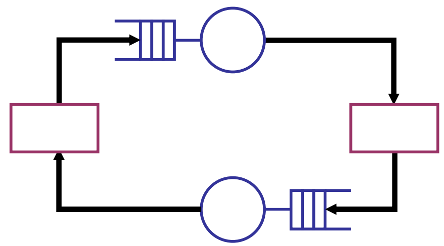


Outline

- Admin and recap
- HTTP
  - HTTP "acceleration"
  - *Operational analysis*

Goal: Best Server Design Limited Only by Resource Bottleneck





Some Questions

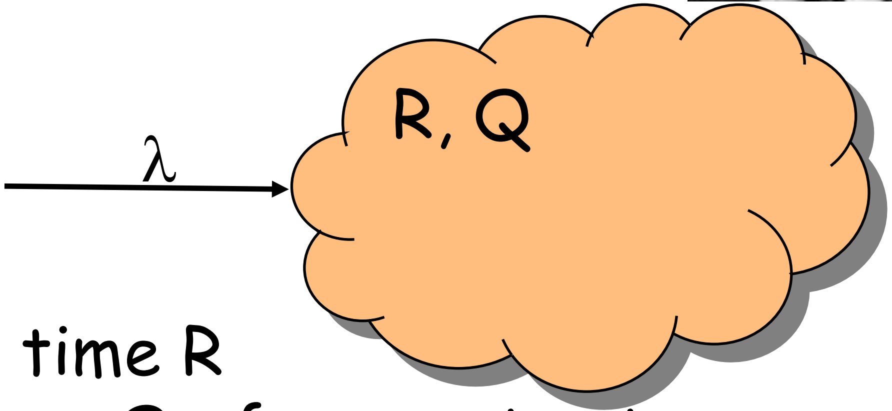
- When is CPU the bottleneck for scalability?
  - So that we need to add helper threads
- How do we know that we are reaching the limit of scalability of a single machine?

- These questions drive network server architecture design

- Some basic performance analysis techniques are good to have

Background: Little's Law (1961)

- For any system with no or (low) loss.

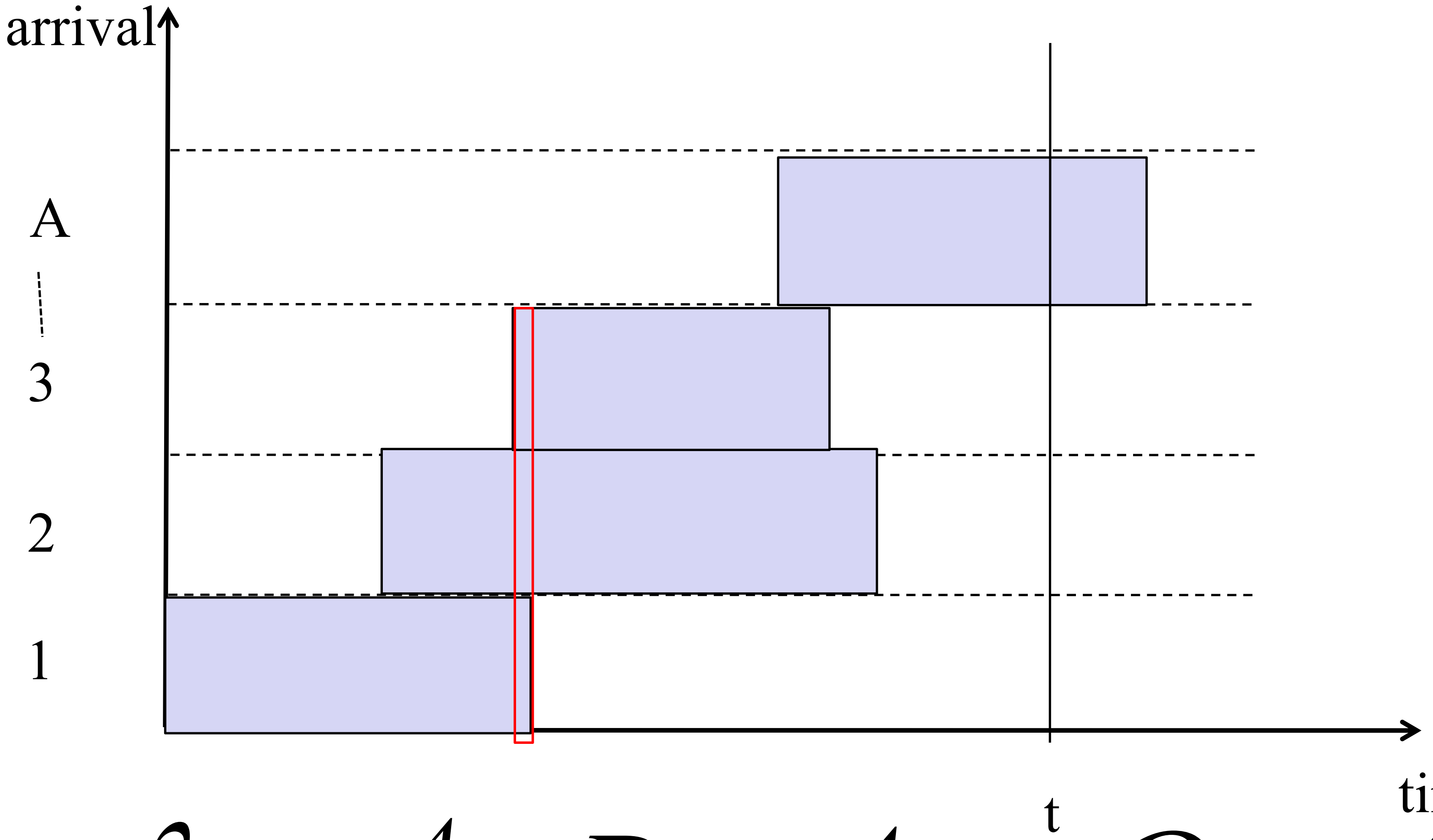


- Assume
  - mean arrival rate λ, mean time R at system, and mean number Q of requests at system
- Then relationship between Q, λ, and R:

$$Q = \lambda R$$

Example: XMU admits 3000 students each year, and mean time a student stays is 4 years, how many students are enrolled?

Little's Law: Proof $Q = \lambda R$



$$\lambda = \frac{A}{t} \quad R = \frac{Area}{A} \quad Q = \frac{Area}{t}$$

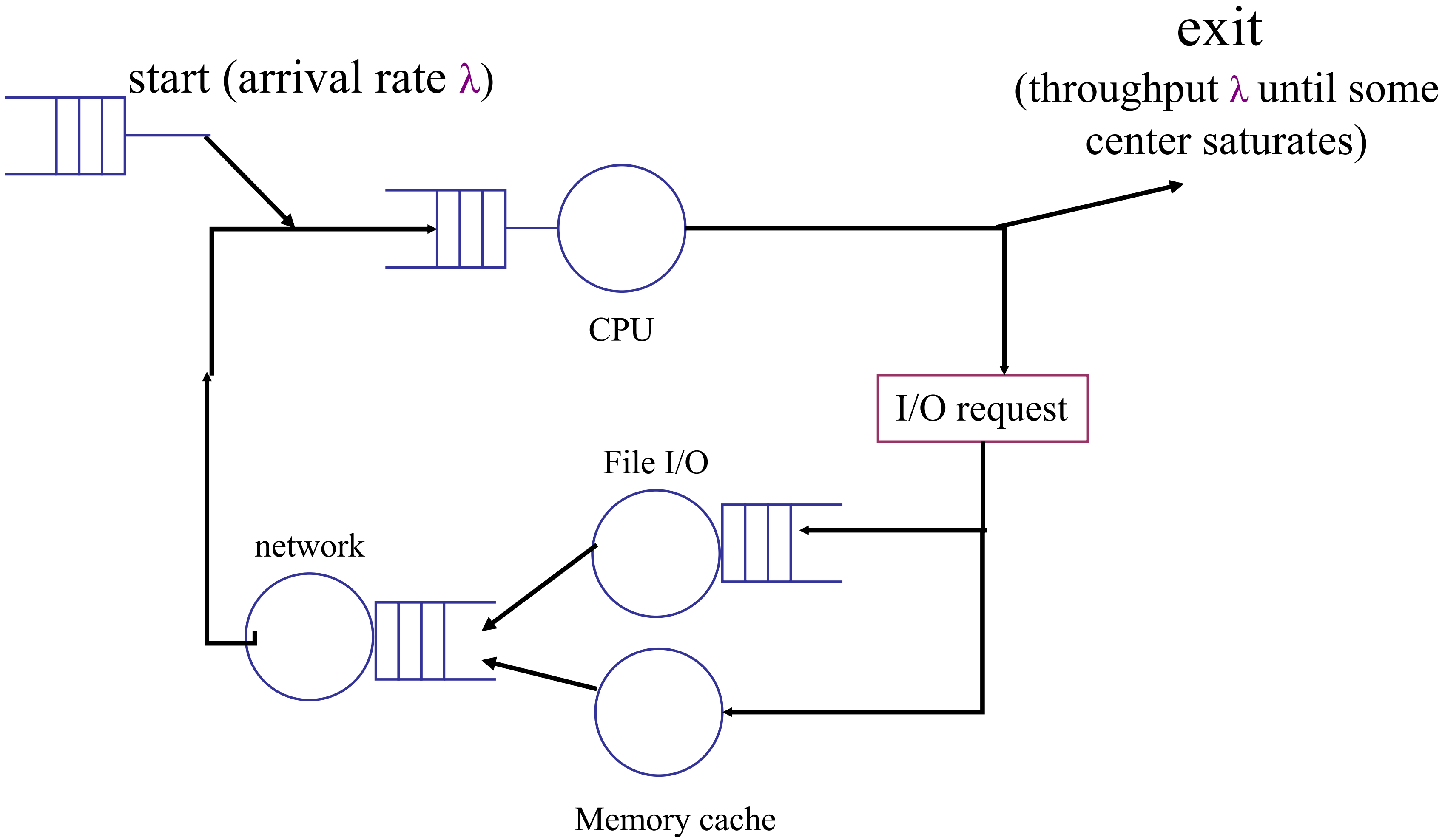
Operational Analysis

- Relationships that do not require any assumptions about the distribution of service times or inter-arrival times
  - Hence focus on measurements

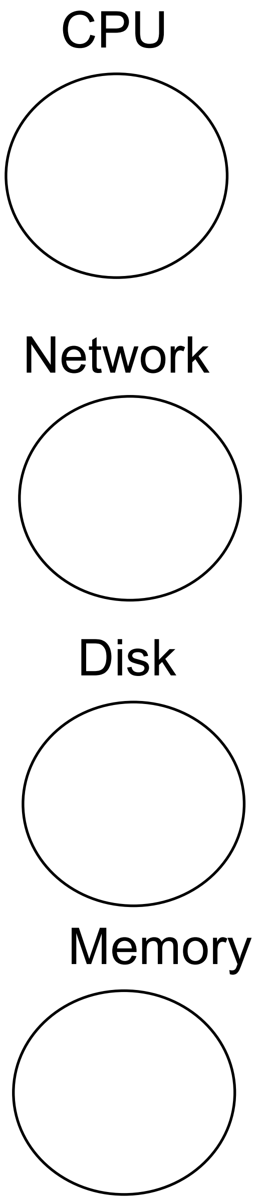
- Identified originally by Buzen (1976) and later extended by Denning and Buzen (1978).

- We touch only some techniques/results
  - In particular, bottleneck analysis
- More details see linked reading

Under the Hood (An example FSM)



Operational Analysis: Resource Demand of a Request



V_{CPU} visits for S_{CPU} units of resource time per visit

V_{Net} visits for S_{Net} units of resource time per visit

V_{Disk} visits for S_{Disk} units of resource time per visit

V_{Mem} visits for S_{Mem} units of resource time per visit

Operational Quantities

- T: observation interval
- Bi: busy time of device i
- i = 0 denotes system

Ai: # arrivals to device i

Ci: # completions at device i

$$\text{arrival rate } \lambda_i = \frac{A_i}{T}$$

$$\text{Throughput } X_i = \frac{C_i}{T}$$

$$\text{Utilization } U_i = \frac{B_i}{T}$$

$$\text{Mean service time } S_i = \frac{B_i}{C_i}$$

Utilization Law

$$\text{Utilization } U_i = \frac{B_i}{T}$$

$$= \frac{C_i}{T}\frac{B_i}{C_i}$$

$$= X_i S_i$$

- The law is independent of any assumption on arrival/service process
- Example: Suppose NIC processes 125 pkts/sec, and each pkt takes 2 ms. What is utilization of the network NIC?

Deriving Relationship Between R, U, and S for one Device

- Assume flow balanced (arrival=throughput), Little's Law:

$$Q = \lambda R = XR$$

- Assume PASTA (Poisson arrival--memory-less arrival--sees time average), a new request sees Q ahead of it, and FIFO

$$R = S + QS = S + XRS$$

- According to utilization law, U = XS

$$R = S + UR \longrightarrow R = \frac{S}{1-U}$$

Forced Flow Law

- Assume each request visits device i Vi times

$$\text{Throughput } \mathrm{X}_i = \frac{C_i}{T}$$

$$= \frac{C_i}{C_0} \frac{C_0}{T}$$

$$= V_i X$$

Bottleneck Device

$$\text{Utilization } U_i = X_i S_i$$

$$= V_i X S_i$$

$$= X V_i S_i$$

- Define Di = Vi Si as the total demand of a request on device i
- The device with the highest Di has the highest utilization, and thus is called the bottleneck

Bottleneck vs System Throughput

$$\text{Utilization } \mathrm{U}_i = XV_iS_i \leq 1$$

$$\rightarrow \; X \leq \frac{1}{D_{\max}}$$

Example 1

- A request may need
  - 10 ms CPU execution time
  - 1 Mbytes network bw
  - 1 Mbytes file access where
    - 50% hit in memory cache
- Suppose network bw is 100 Mbps, disk I/O rate is 1 ms per 8 Kbytes (assuming the program reads 8 KB each time)
- Where is the bottleneck?

Example 1 (cont.)

- CPU:
  - D_{CPU}= 10 ms (e.q. 100 requests/s)
- Network:
  - D_{Net} = 1 Mbytes / 100 Mbps = 80 ms (e.q., 12.5 requests/s)
- Disk I/O:
  - Ddisk = 0.5 * 1 ms * 1M/8K = 62.5 ms (e.q. = 16 requests/s)

Example 2

- A request may need
  - 150 ms CPU execution time (e.g., dynamic content)
  - 1 Mbytes network bw
  - 1 Mbytes file access where
    - 50% hit in memory cache
- Suppose network bw is 100 Mbps, disk I/O rate is 1 ms per 8 Kbytes (assuming the program reads 8 KB each time)

- Bottleneck: CPU -> use multiple threads to use more CPUs, if available, to avoid CPU as bottleneck