
CS433/533: Computer Networks

Y. Richard Yang

<http://zoo.cs.yale.edu/classes/cs433/>

1/20/2016

Outline

- *Administrative trivia's*
- What is a network protocol?
- A brief introduction to the Internet: past and present
- Challenges of Internet networks and apps
- Summary

Personnel

□ Instructor

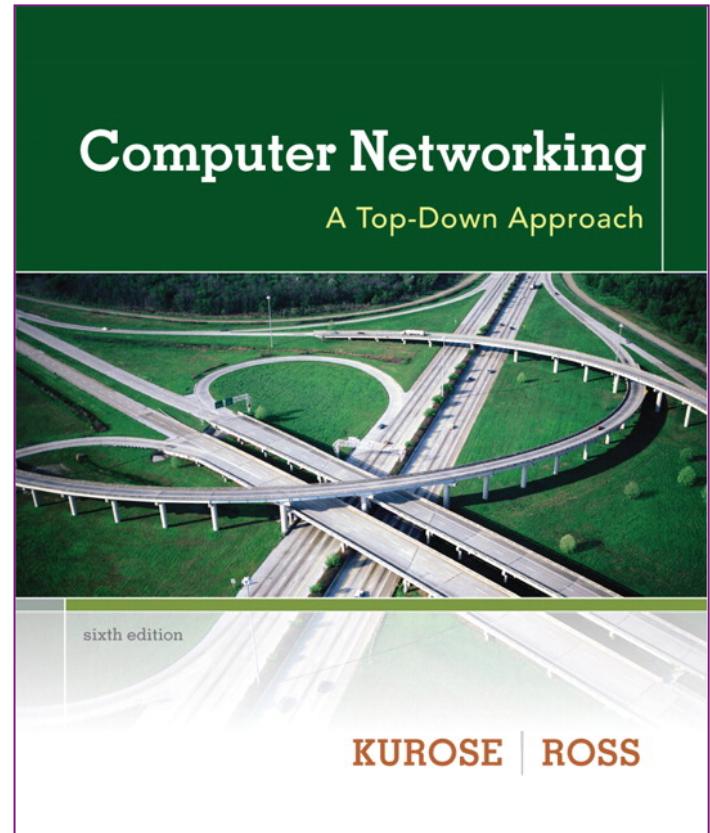
- Y. Richard Yang, ry@cs.yale.edu, AKW 208A
 - office hours
 - WM 2:00-3:00 pm or by appointment
 - please feel free to stop by if you see I am in my office

□ Teaching assistants

- Qiao Xiang, AKW 214
 - office hours TBA
- Dan Peng, AKW 214
 - office hours TBA

Textbook

- Textbook
 - *Computer Networking: A Top-Down Approach, 6/e*
by Jim Kurose and Keith Ross
(7/e will become available in Apr)
- Reference books
 - *Computer Networks*
by Tanenbaum and Wetherall
 - *Computer Networks, A Systems Approach*
by Larry Peterson and Bruce Davie
 - *TCP/IP Illustrated, Volume 1: The Protocols*
by W. Richard Stevens
 - *Java Network Programming,*
by Elliotte Harold
- Resources
 - <http://zoo.cs.yale.edu/classes/cs433>



What are the Goals of this Course?

- Learn design principles and techniques of:
 - the Internet infrastructure (ISP, data center, cloud)
 - large-scale Internet applications

- Focus on how the principles and techniques apply and adapt in real world:
 - real examples from the Internet

What Do You Need To Do?

- Please return the class background survey
 - help us determine your background
 - help us determine the depth, topics, and assignments
 - suggest topics that you want to be covered (if you think of a topic later, please send me email)

- Your workload
 - homework assignments
 - written assignments
 - three programming assignments
 - one HTTP 1.1 server, one TCP, and one OpenStack network orchestrator [still debating]
 - one exam
 - one project

Grading

Exam	20%
Assignments and project	70%
Class Participation	10%

- Subject to change after I know more about your background
- More important is what you realize/learn than the grades !!

Questions?

Outline

- Administrative trivia's
- *What is a network protocol?*

What is a Network Protocol?

- A **network protocol** defines the **format** and the **order** of messages exchanged between two or more communicating entities, as well as the **actions** taken on the transmission and/or receipt of a message or other **events**.

Example Protocol: Simple Mail Transfer Protocol (SMTP)

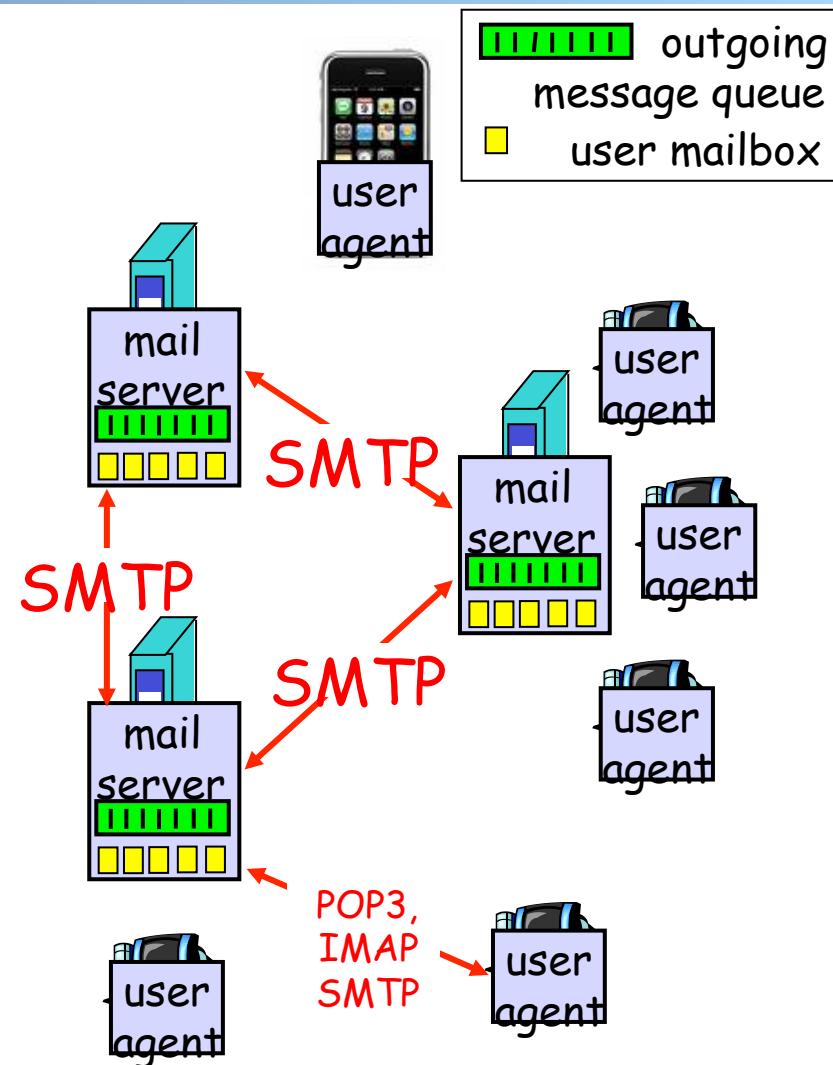
□ Messages from a client to a mail server

- HELO
- MAIL FROM: <address>
- RCPT TO: <address>
- DATA
<This is the text end with a line with a single .>
- QUIT

□ Messages from a mail server to a client

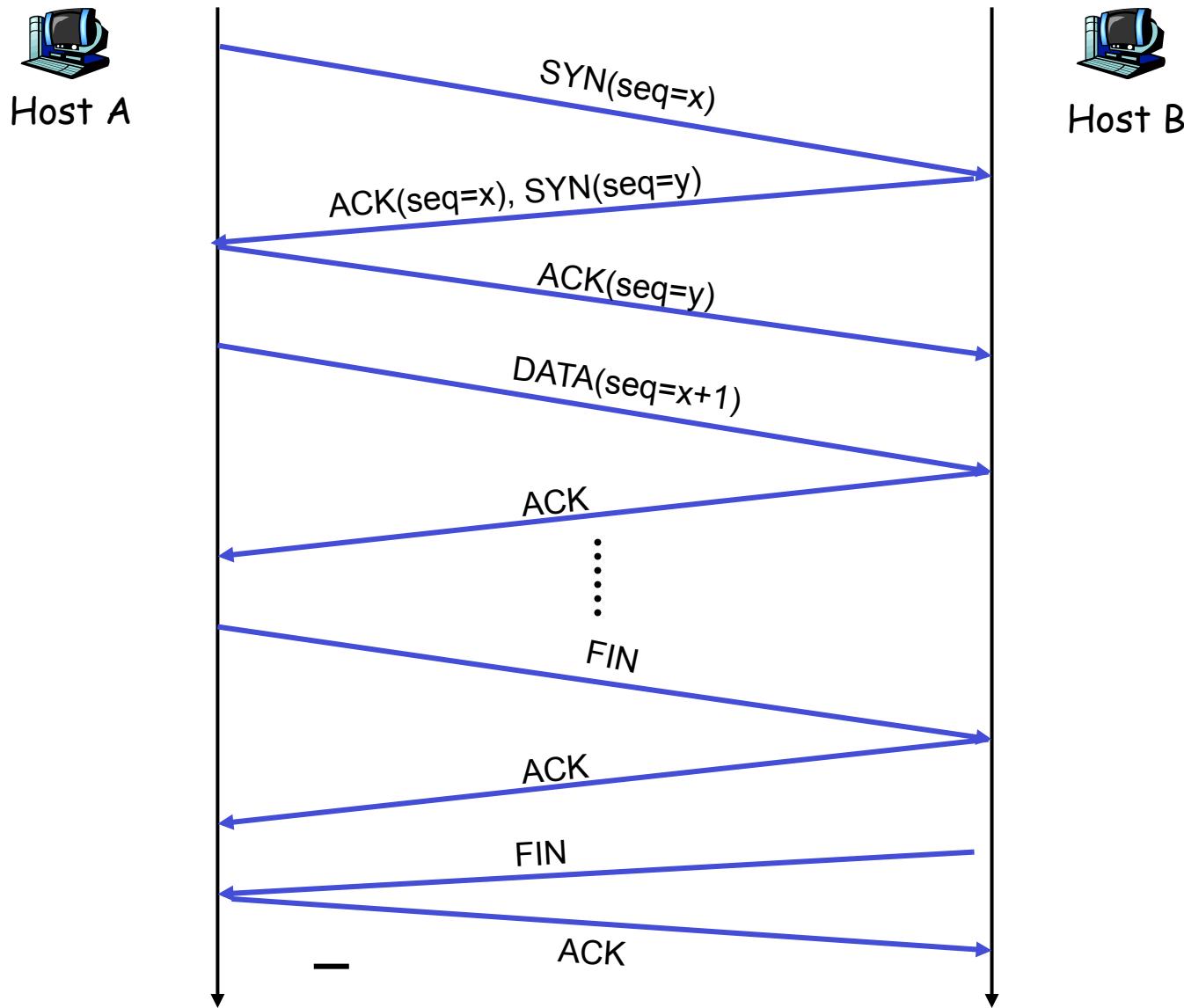
- status code
 - The first digit of the response broadly indicates the success, failure, or progress of the previous command.
 - 1xx - Informative message
 - 2xx - Command ok
 - 3xx - Command ok so far, send the rest of it.
 - 4xx - Command was correct, but couldn't be performed for some reason.
 - 5xx - Command unimplemented, or incorrect, or a serious program error occurred.

- content



Command: %telnet netra.cs.yale.edu smtp

Example: TCP Protocol Handshakes



Protocol Standardization

- Most widely used protocols are defined in standards

- Why standard?

Internet Standardization Process

- All standards of the Internet are published as **RFC** (**Request for Comments**)
 - e.g., the SMTP protocol is specified in RFC821
 - but not all RFCs are Internet Standards: <http://zoo.cs.yale.edu/classes/cs433/cs433-2016-spring/readings/interestingrfcs.html>

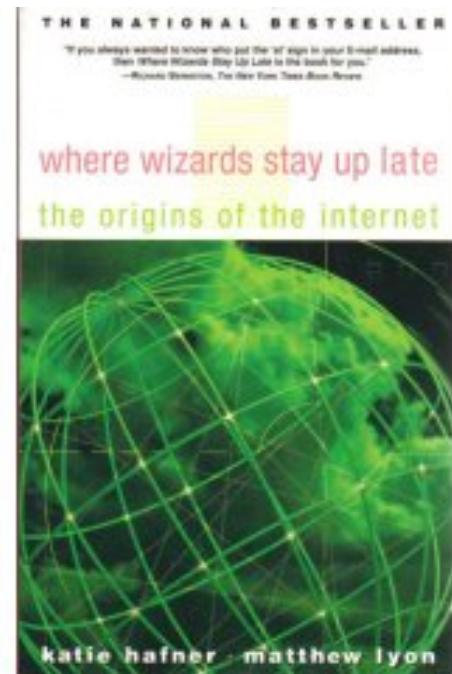
Internet Standardization Process

- All standards of the Internet are published as **RFC** (**Request for Comments**)
 - e.g., the SMTP protocol is specified in RFC821
 - but not all RFCs are Internet Standards: <http://zoo.cs.yale.edu/classes/cs433/cs433-2016-spring/readings/interestingrfcs.html>
- A typical (but not the only) way of standardization:
 - Internet draft
 - RFC
 - proposed standard
 - draft standard (requires 2 working implementations)
 - Internet standard (declared by Internet Architecture Board)
- David Clark, 1992:

We reject: kings, presidents, and voting. We believe in: rough consensus and running code.

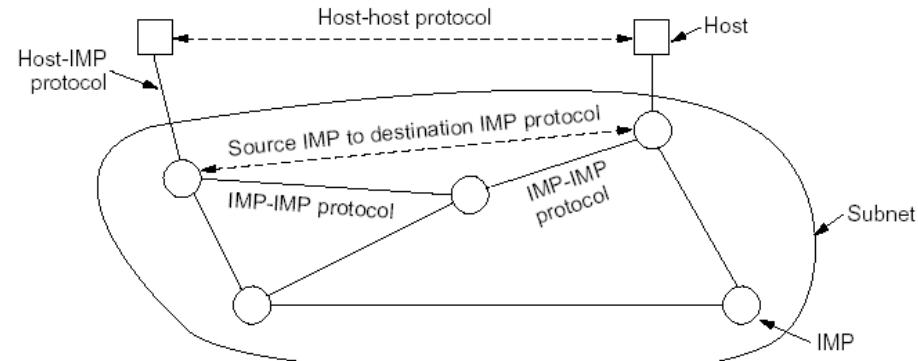
Outline

- Administrative trivia's
- What is a network protocol?
- *A brief introduction to the Internet*
 - *past (a brief history)*
 - present



Prelude: Packet Switching and ARPANET

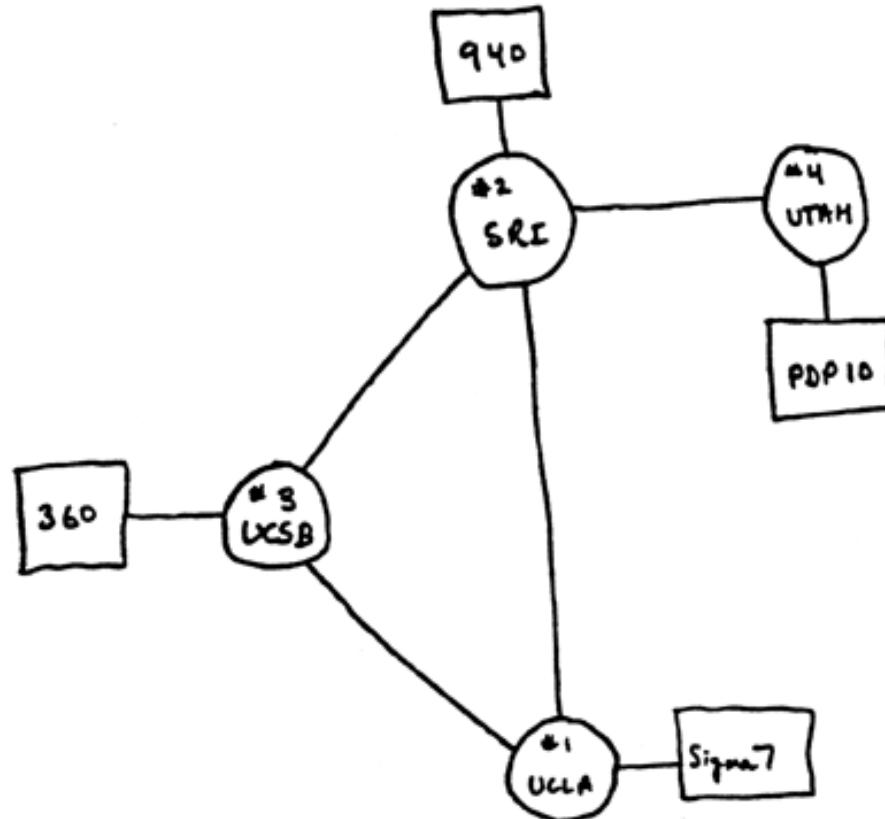
- 1957
 - USSR launched Sputnik; US DoD formed Advanced Research Projects Agency (ARPA)
- 1961
 - First paper by Len Kleinrock on packet switching theory
- 1964
 - Paul Baran from RAND on design of packet switching networks
- 1965-1968
 - **ARPANET** plan
 - Bolt Beranek and Newman, Inc. (BBN), a **small** company, was awarded Packet Switch contract to build Interface Message Processors (IMPs)



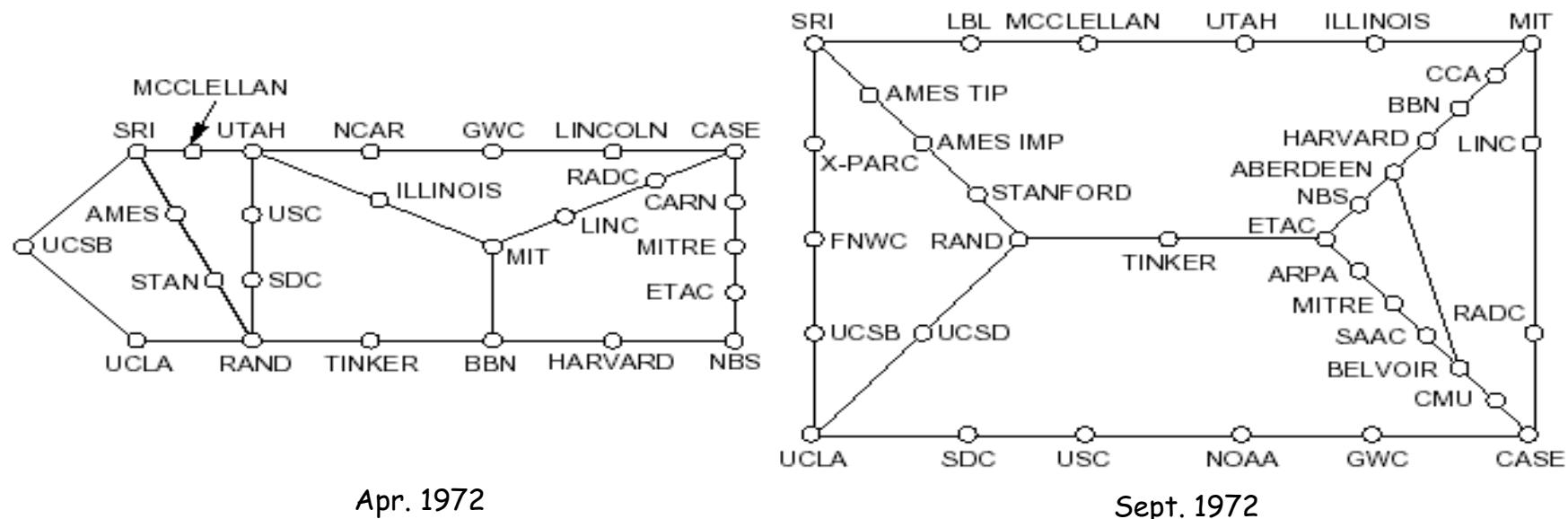
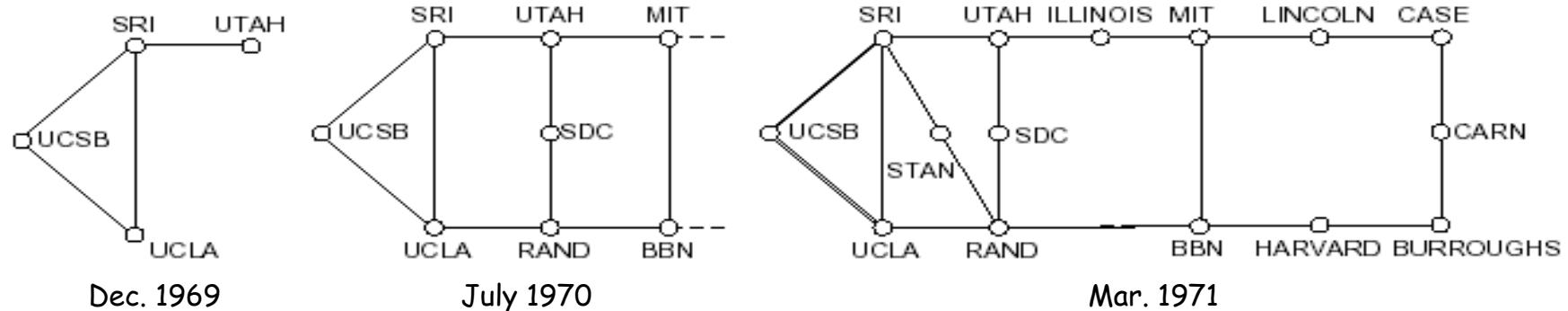
Internet 1.0: Initial ARPANET

□ 1969

- ARPANET commissioned: 4 nodes, 50kbps



Initial Expansion of the ARPANET



RFC 527: ARPAWOCKY; RFC 602: The Stockings Were Hung by the Chimney with Care

The Internet Becomes a Network of Networks

- 1970: ALOHAnet, the first packet radio network, developed by Norman Abramson, Univ of Hawaii, became operational
- 1973: Bob Kahn posed the Internet problem---how to connect ARPANET, packet radio network, and satellite network
- 1974: Vint Cerf, Bob Kahn published initial design of TCP (NCP) to connect multiple networks
 - 1978: TCP (NCP) split to TCP/IP
 - 1983: TCP (NCP) converted to TCP/IP (Jan. 1)

Growth of the Internet

- 1981: BITNET (Because It's Time NETwork) between CUNY and Yale
- 1986: NSF builds NSFNET as backbone, links 6 supercomputer centers, 56 kbps; this allows an explosion of connections, especially from universities
- 1987: 10,000 hosts
- 1988: NSFNET backbone upgrades to 1.5Mbps
- 1988: Internet congestion collapse; TCP congestion control
- 1989: 100,000 hosts

RFC 1121: Act One - The Poem

WELCOME by Leonard Kleinrock

We've gathered here for two days to examine and debate
And reflect on data networks and as well to celebrate.
To recognize the leaders and recount the path we took.

We'll begin with how it happened; for it's time to take a look.
Yes, the history is legend and the pioneers are here.
Listen to the story - it's our job to make it clear.
We'll tell you where we are now and where we'll likely go.
So welcome to ACT ONE, folks.
Sit back - enjoy the show!!

Internet 2.0: Web, Commercialization, Social Networking of the Internet

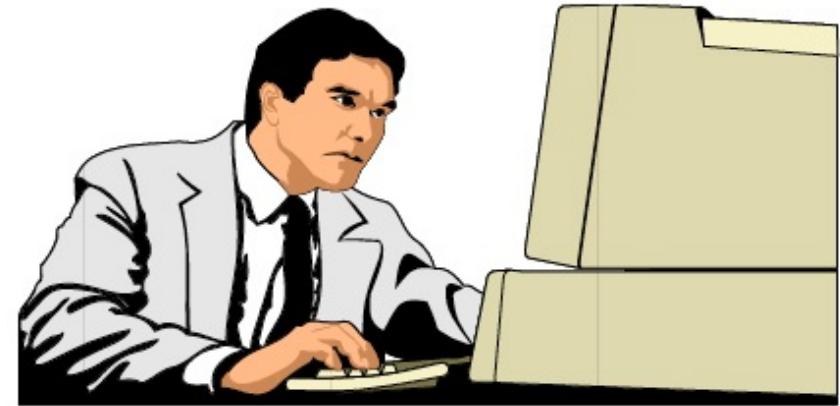
- 1990: ARPANET ceases to exist
- 1991: NSF lifts restrictions on the commercial use of the Net; Berners-Lee of European Organization for Nuclear Research (CERN) released World Wide Web
- 1992: 1 million hosts (RFC 1300: Remembrances of Things Past)
- 1998: Google was founded
- 2004: Facebook was founded
- 2006: Amazon AWS cloud computing

For a link of interesting RFCs, please see

<http://zoo.cs.yale.edu/classes/cs433/cs433-2016-fall/readings/interestingrfcs.html>

For more on Internet history, please see <http://www.zakon.org/robert/internet/timeline/>

Internet 3.0: Always-Connected, Virtualized Life



- Office
- Shopping
- Education
- Entertainment
- Environment

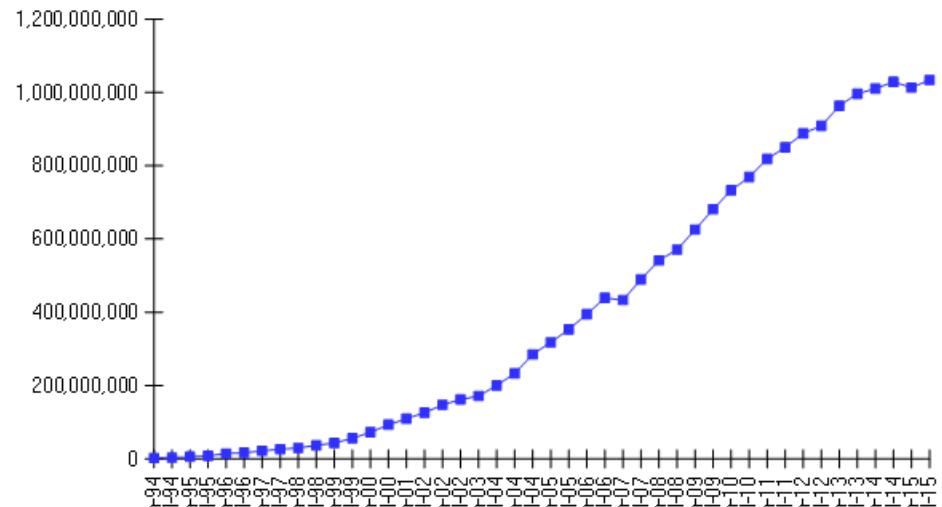
=> Virtual workspace
=> Online shopping
=> Remote education
=> Online media/games
=> Internet of things

Growth of the Internet in Terms of Number of Hosts

Number of Hosts on the Internet:

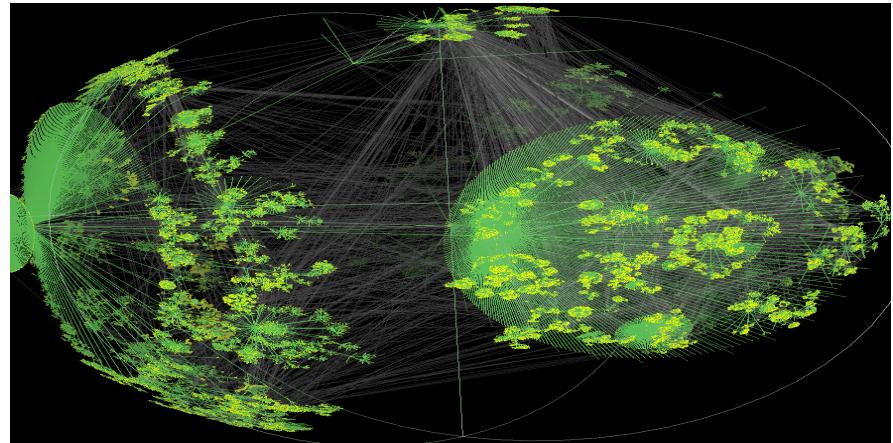
Aug. 1981	213
Oct. 1984	1,024
Dec. 1987	28,174
Oct. 1990	313,000
Jul. 1993	1,776,000
Jul. 1996	12,881,000
Jul. 1999	56,218,000
Jul. 2002	162,128,493
Jul. 2005	353,284,187
Jul. 2008	570,937,778
Jul. 2011	849,869,781
Jul. 2013	996,230,757
Jul. 2015	1,033,836,245

Internet Domain Survey Host Count



Source: Internet Systems Consortium (www.isc.org)

CAIDA router
level view



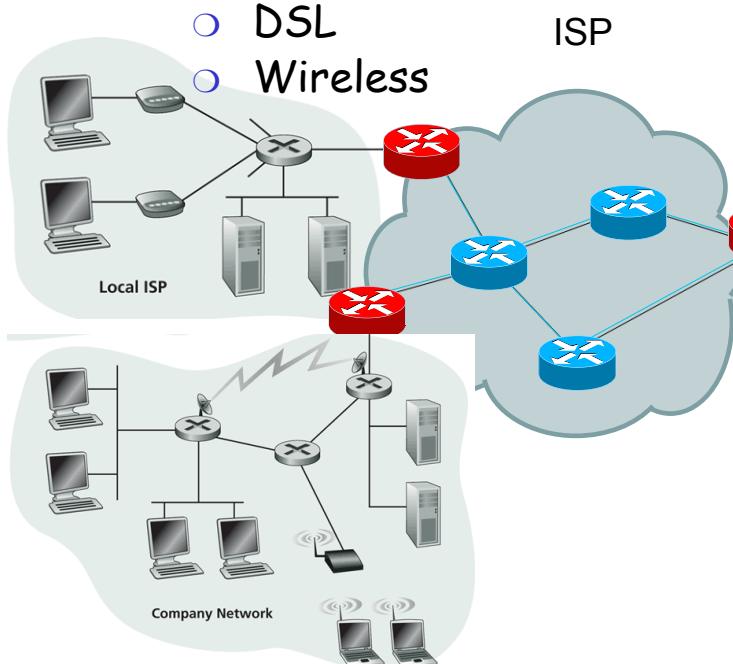
Outline

- Administrative trivia's
- What is a network protocol?
- *A brief introduction to the Internet*
 - past
 - *present*

Internet Physical Infrastructure

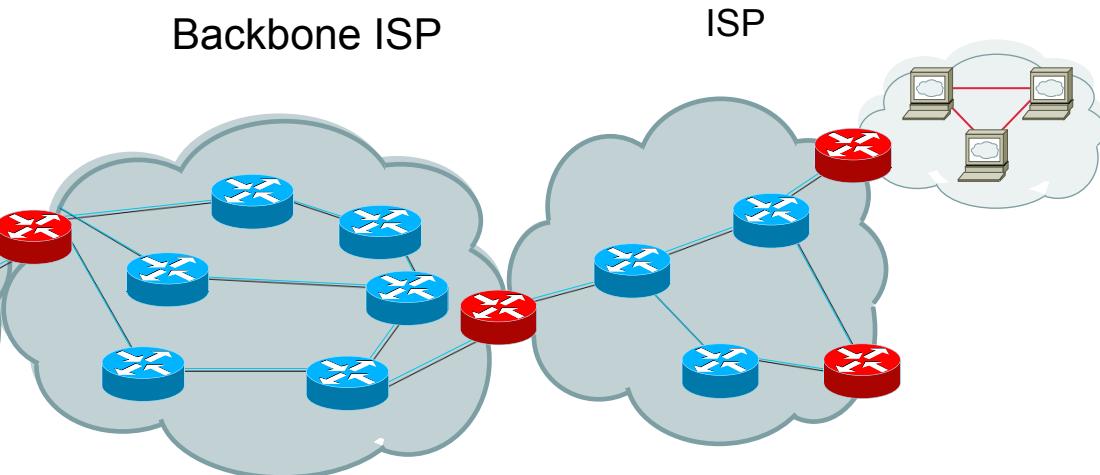
Residential access

- Cable
- Fiber
- DSL
- Wireless



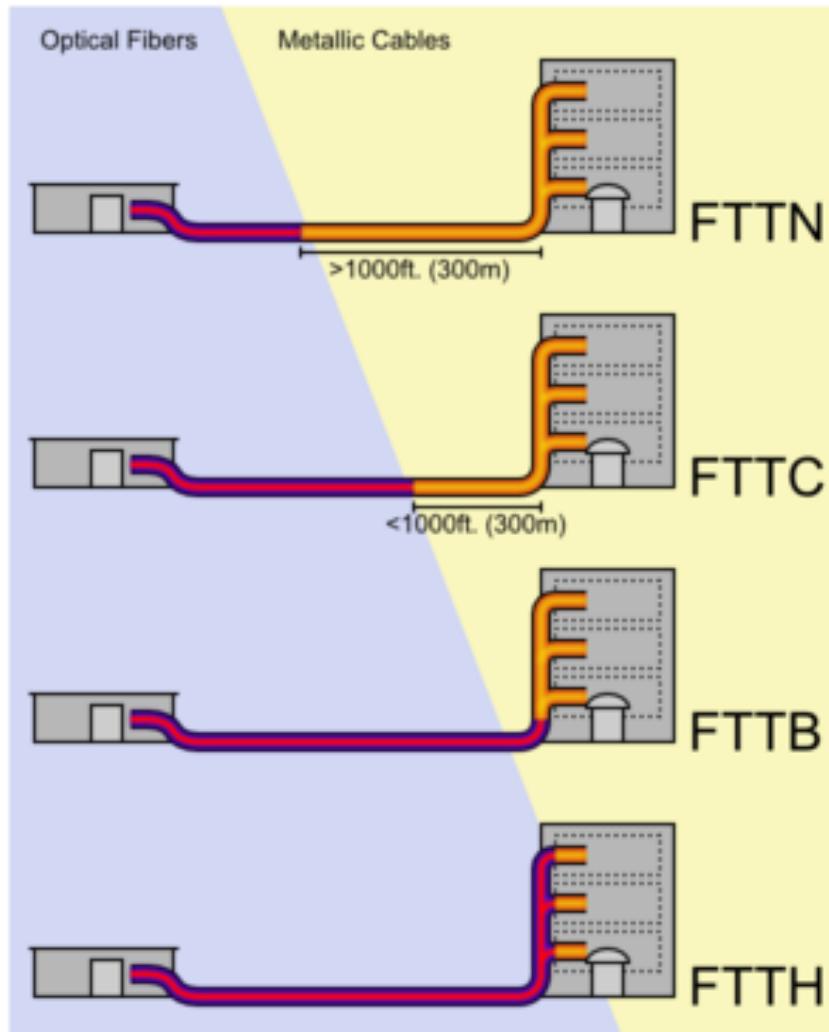
Campus access,
e.g.,

- Ethernet
- Wireless



- The Internet is a network of networks
- Each individually administrated network is called an Autonomous System (AS)

Access: Fiber to the x

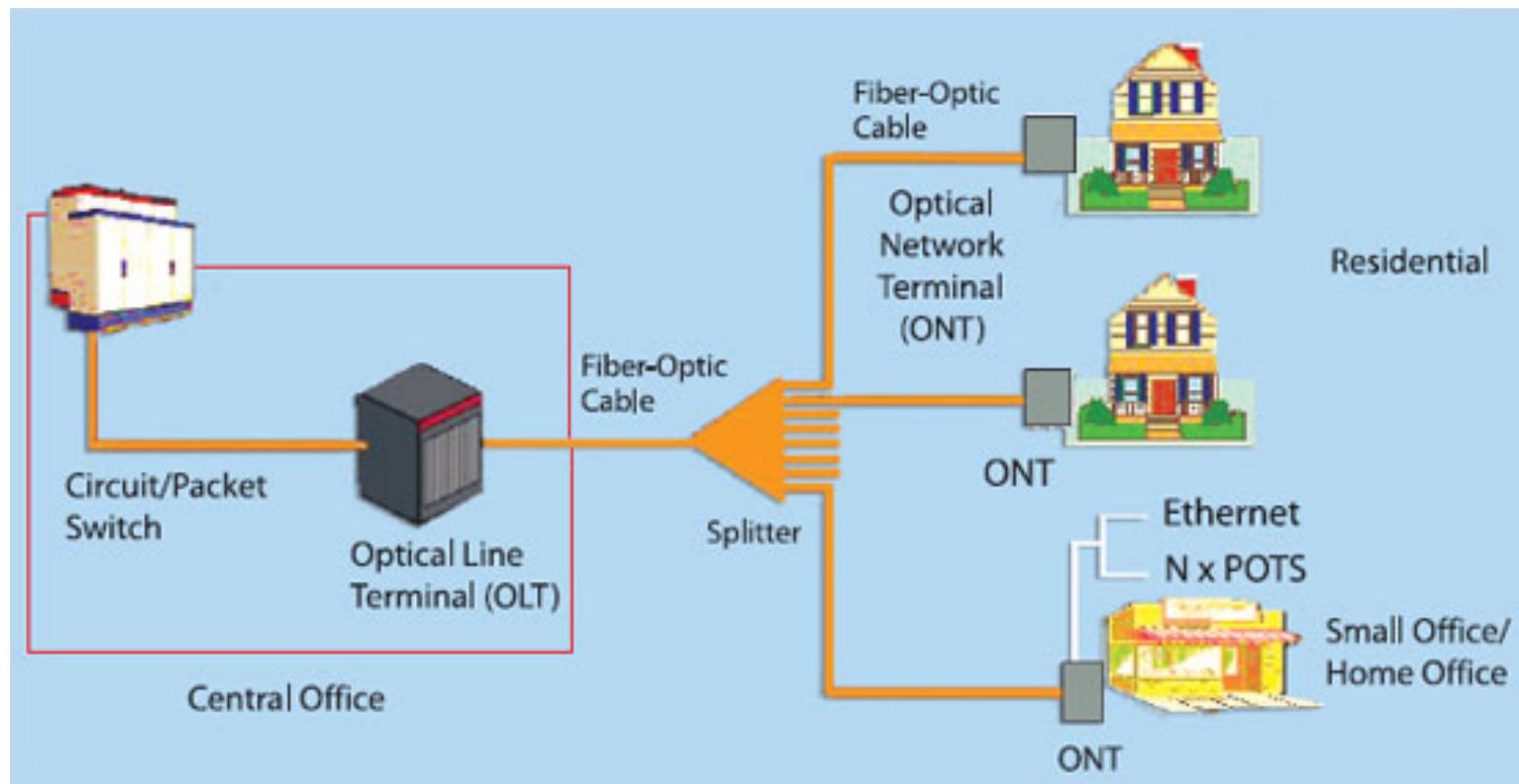


Access: Fiber to the Premises (FTTP)

- Deployed by Verizon, AT&T, Google
- One of the largest comm. construction projects

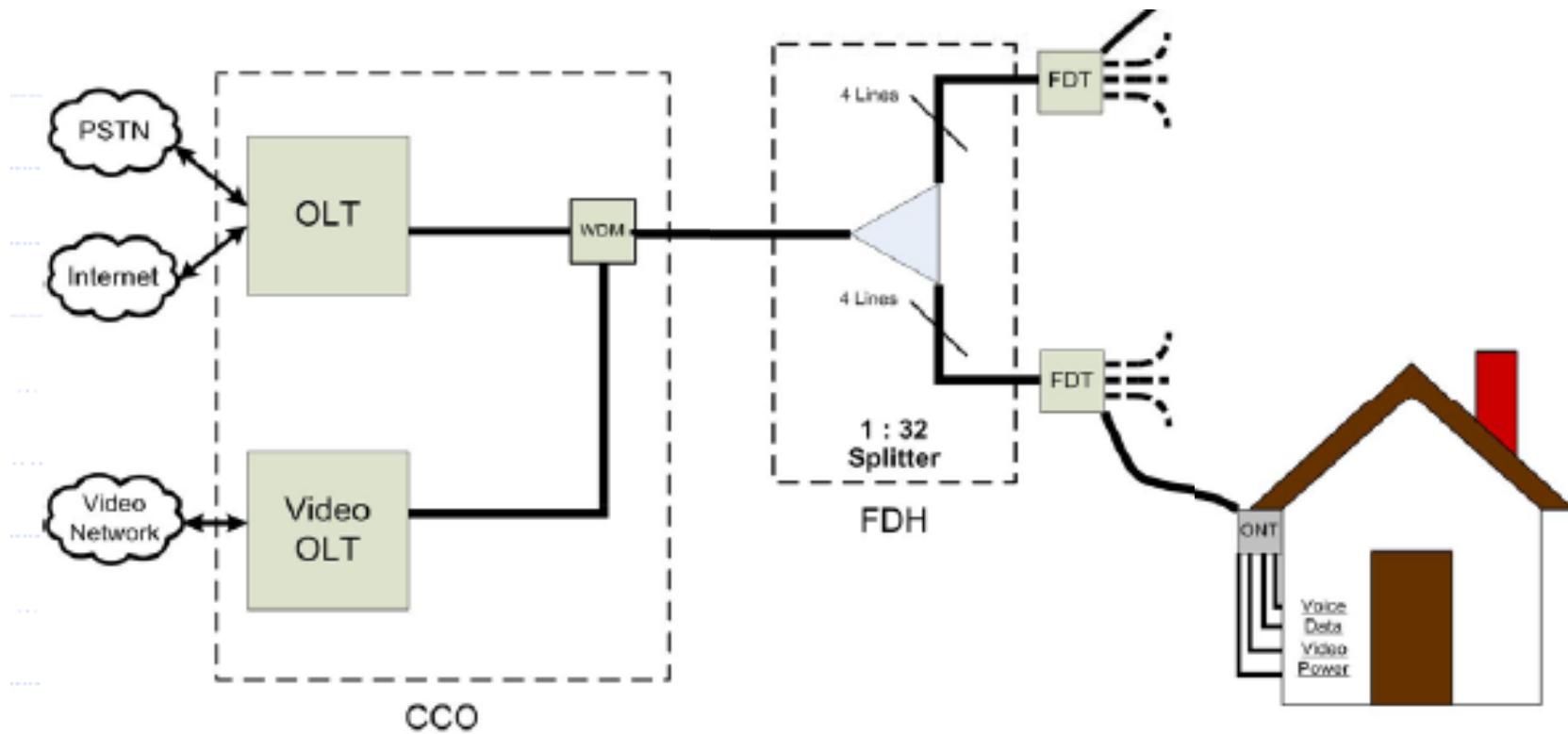


FTTP Architecture

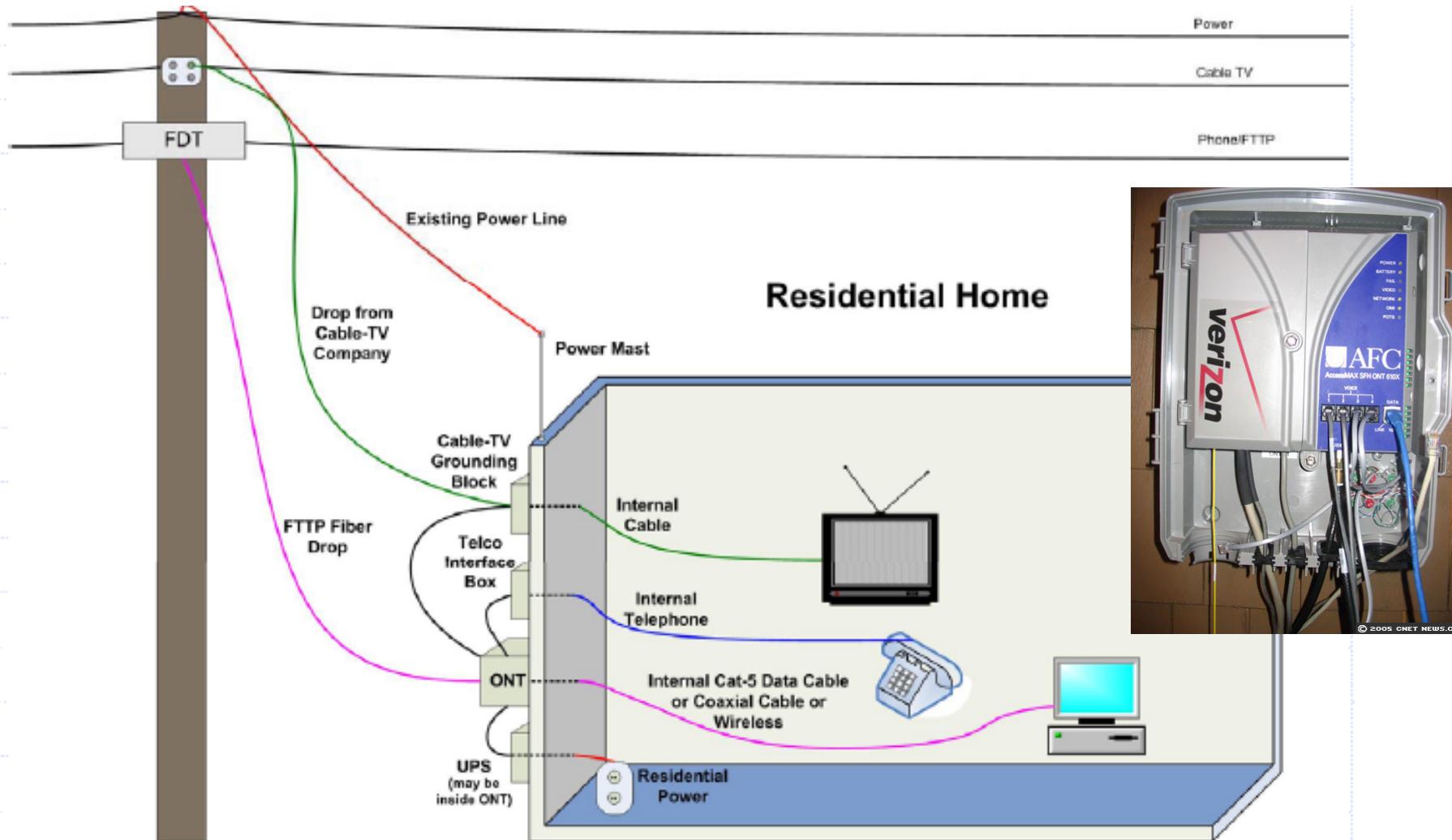


FTTP Architecture

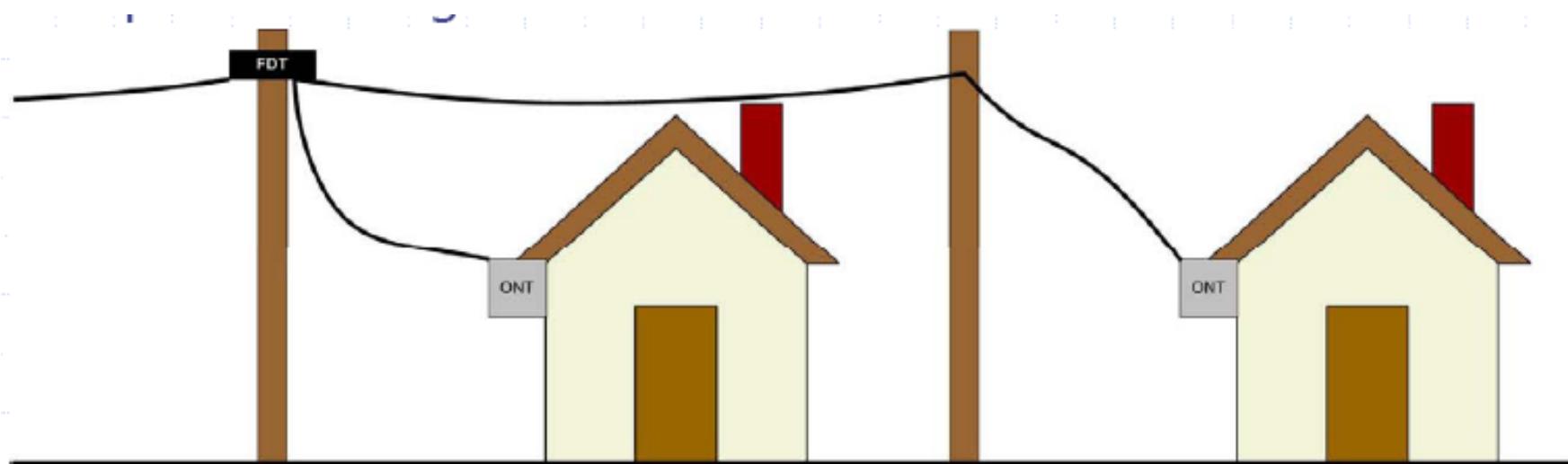
- Optical Network Terminal (ONT) box outside dwelling or business
- Fiber Distribution Terminal (FDT) in poles or pedestals
- Fiber Distribution Hub (FDH) at street cabinet
- Optical Line Terminal (OLT) at central office



FTTP Architecture: To Home



FTTP Architecture: Fiber Distribution Terminal (FDT)



FTTP Architecture: Central to Fiber Distribution Hub (FDH)



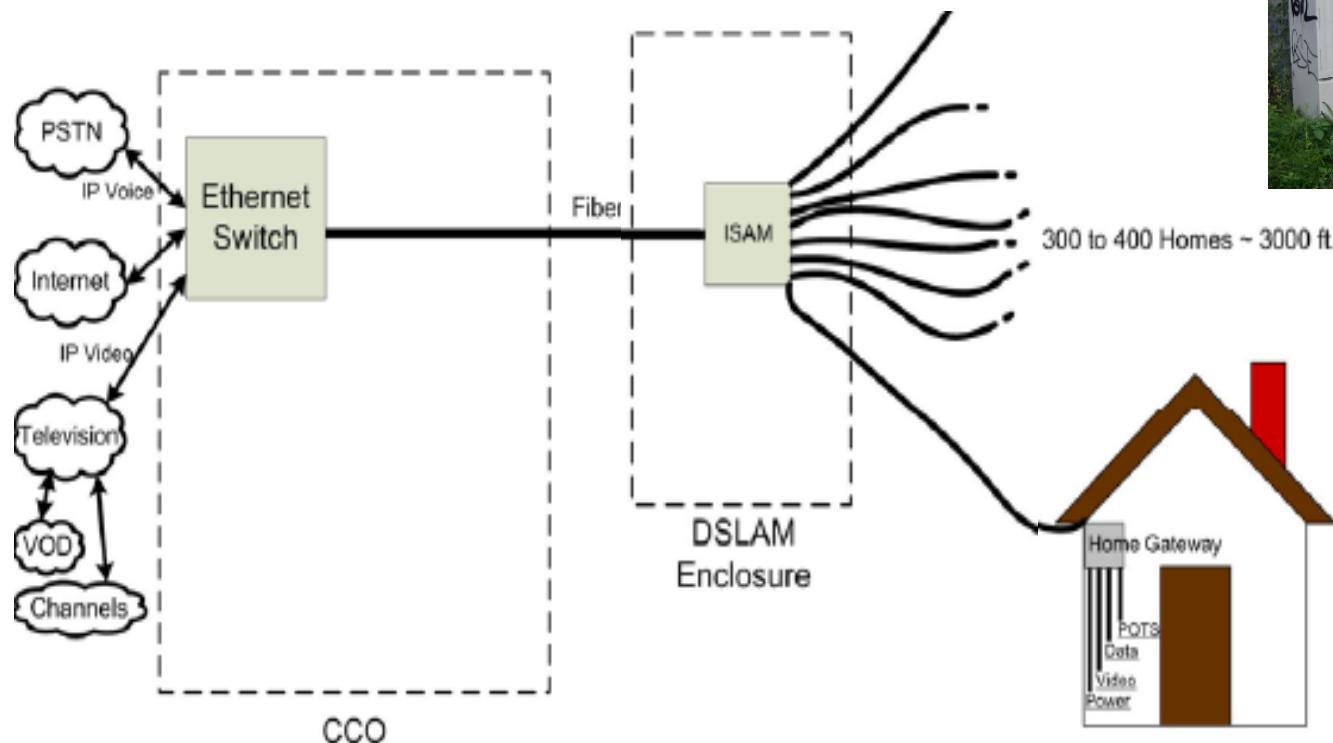
- Backbone fiber ring on primary arterial streets (brown)
- Local distribution fiber plant (red) meets backbone at cabinet



FDH

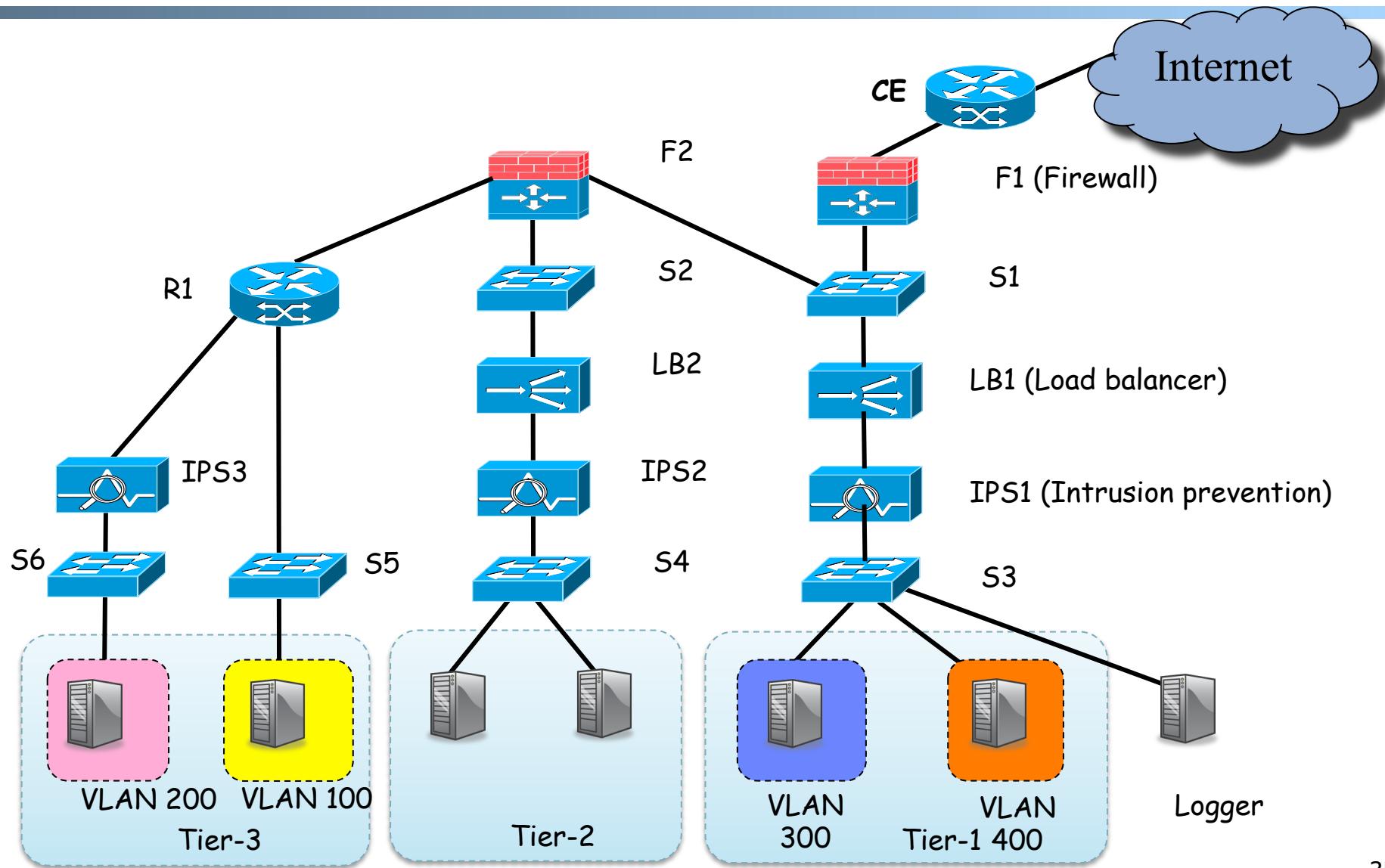
Access: DSL

- Compared with FTTP, copper from cabinet (DSLAM) to home

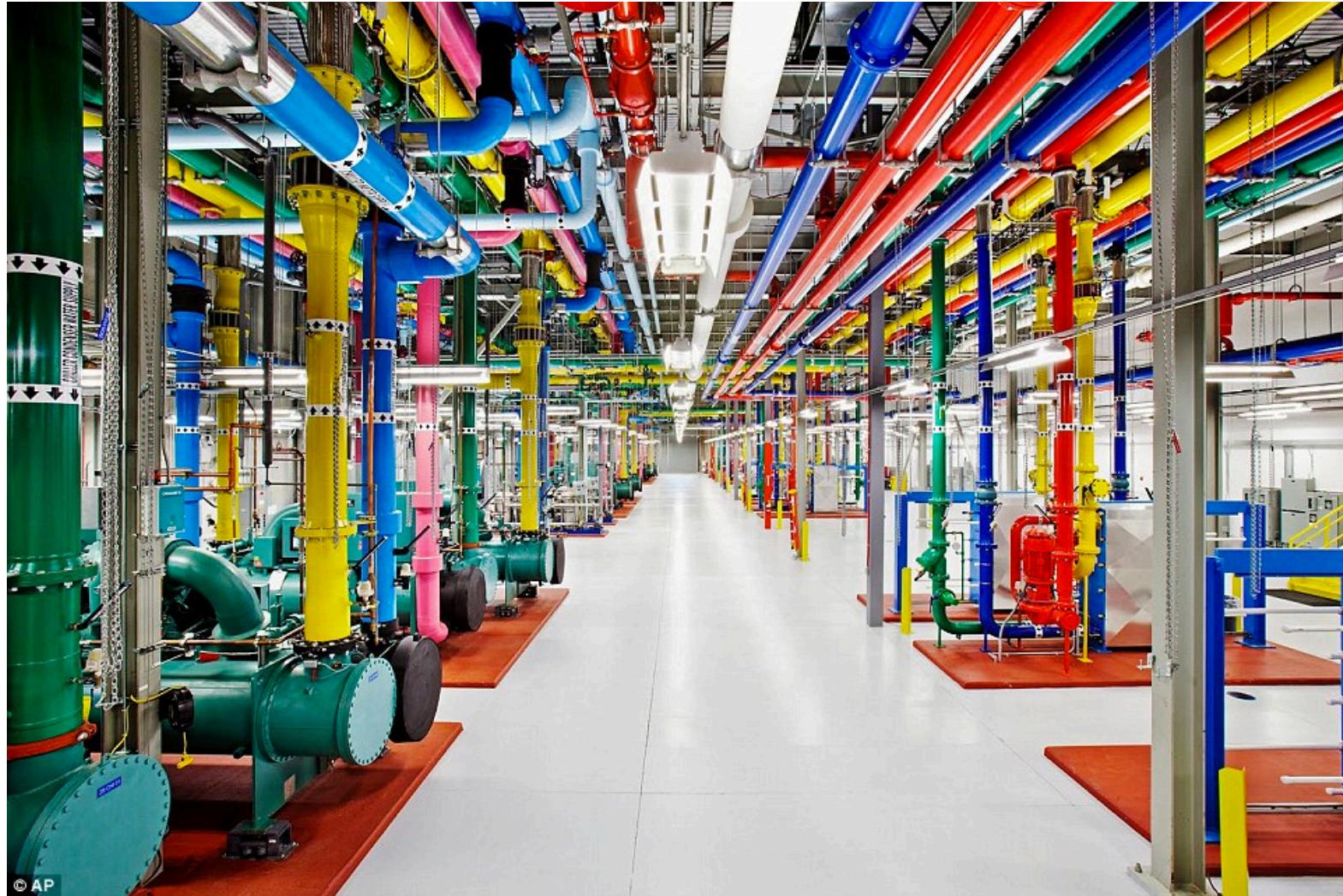


DSLAM

Campus Network

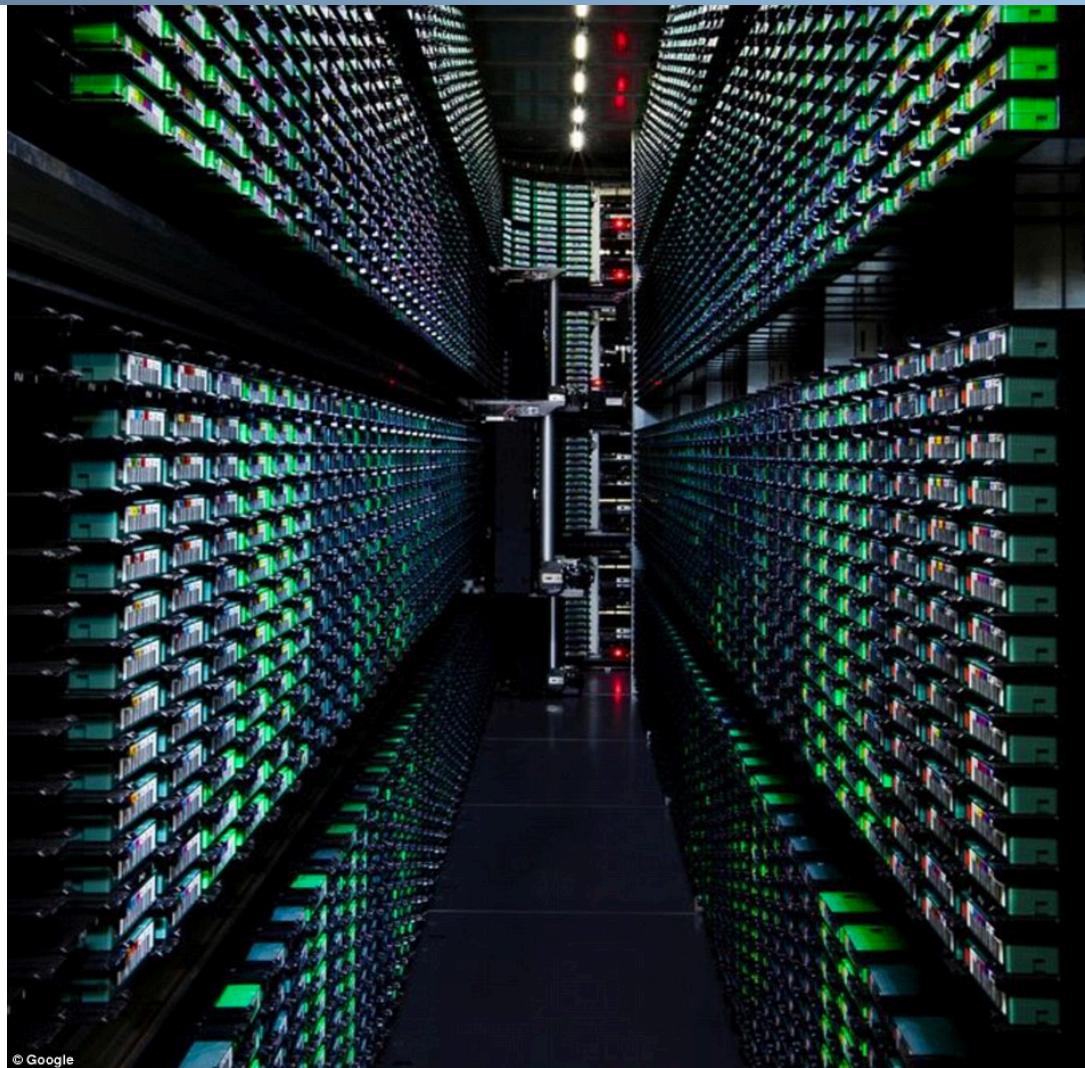


Data Center Networks



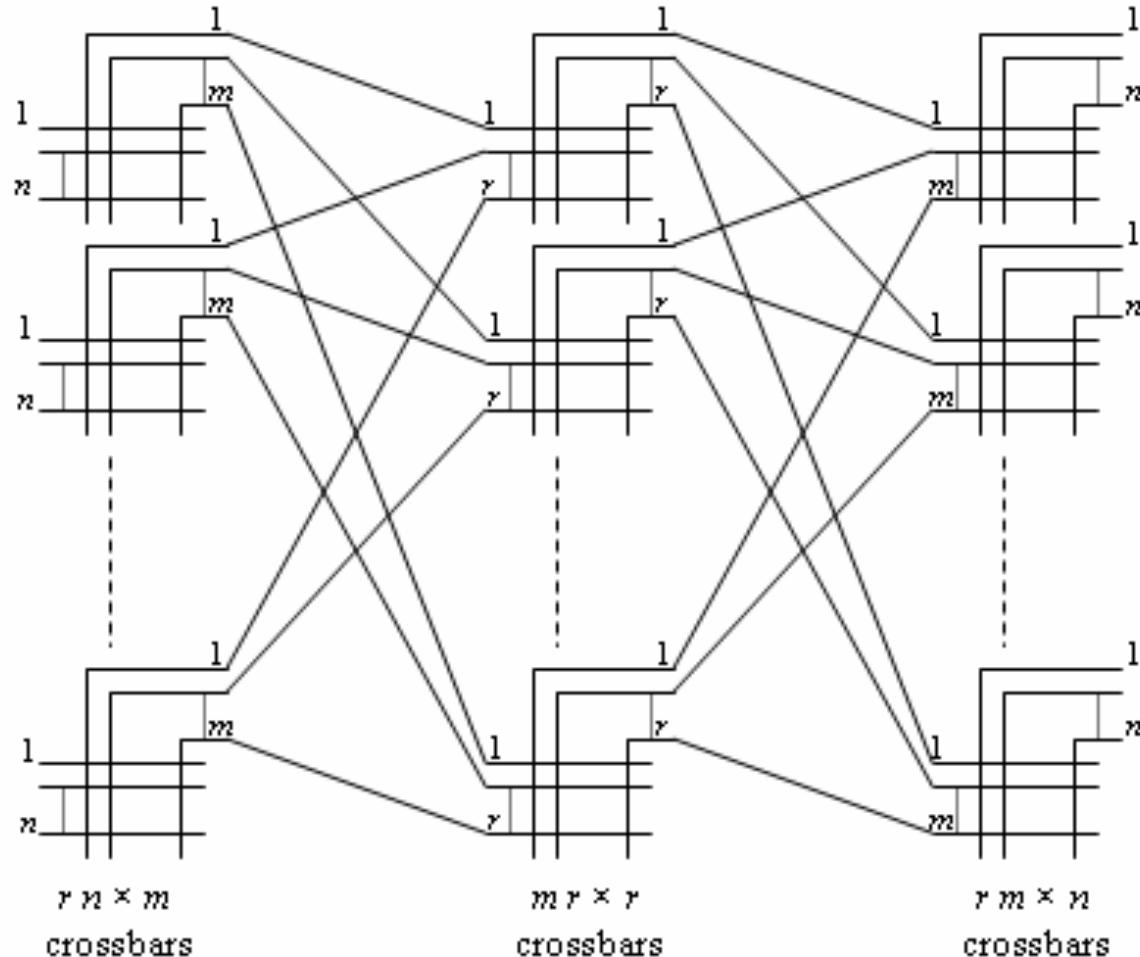
[http://www.dailymail.co.uk/sciencetech/article-3369491/Google's-plan-world-Search-engine-build-half-billion-dollar-data-center-US.html](http://www.dailymail.co.uk/sciencetech/article-3369491/Google-s-plan-world-Search-engine-build-half-billion-dollar-data-center-US.html)

Data Center Networks



<http://www.dailymail.co.uk/sciencetech/article-3369491/Google-s-plan-world-Search-engine-build-half-billion-dollar-data-center-US.html>

Foundation of Data Center Networks: Clos Networks

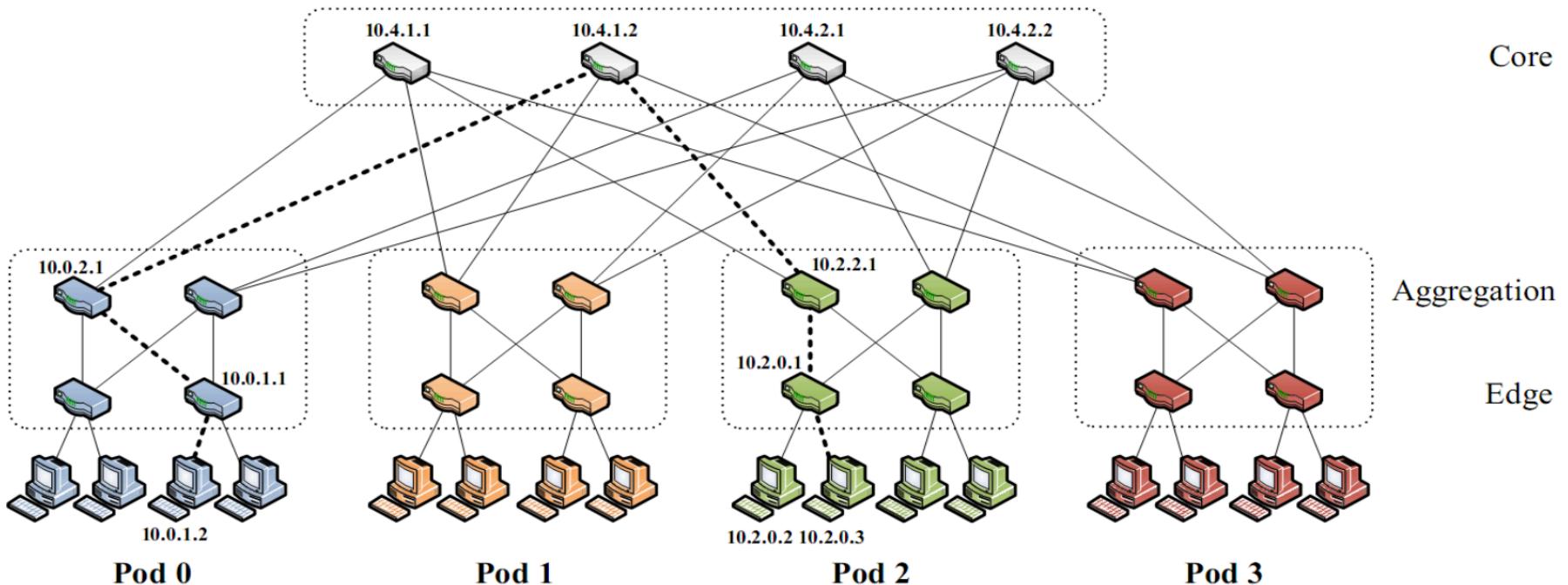


Q: How big is m so that each new call can be established w/o moving current calls?

Homework:
If you can move existing calls, it is only $m \geq n$.

Data Center Networks: Fat-tree Networks

- K-ary fat tree: three-layer topology (edge, aggregation and core)
 - k pods w/ each pod consists of $(k/2)^2$ servers & 2 layers of $k/2$ k-port switches
 - each edge switch connects to $k/2$ servers & $k/2$ aggr. switches
 - each aggr. switch connects to $k/2$ edge & $k/2$ core switches
 - $(k/2)^2$ core switches: each connects to k pods



Data Center Networks

- For example, Google Jupiter at 1 Pbits/sec bisection bw: 100,000 servers at 10G each

Datacenter Generation	First Deployed	Merchant Silicon	ToR Config	Aggregation Block Config	Spine Block Config	Fabric Speed	Host Speed	Bisection BW
Four-Post CRs	2004	vendor	48x1G	-	-	10G	1G	2T
Firehose 1.0	2005	8x10G 4x10G (ToR)	2x10G up 24x1G down	2x32x10G (B)	32x10G (NB)	10G	1G	10T
Firehose 1.1	2006	8x10G	4x10G up 48x1G down	64x10G (B)	32x10G (NB)	10G	1G	10T
Watchtower	2008	16x10G	4x10G up 48x1G down	4x128x10G (NB)	128x10G (NB)	10G	nx1G	82T
Saturn	2009	24x10G	24x10G	4x288x10G (NB)	288x10G (NB)	10G	nx10G	207T
Jupiter	2012	16x40G	16x40G	8x128x40G (B)	128x40G (NB)	10/40G	nx10G/ nx40G	1.3P

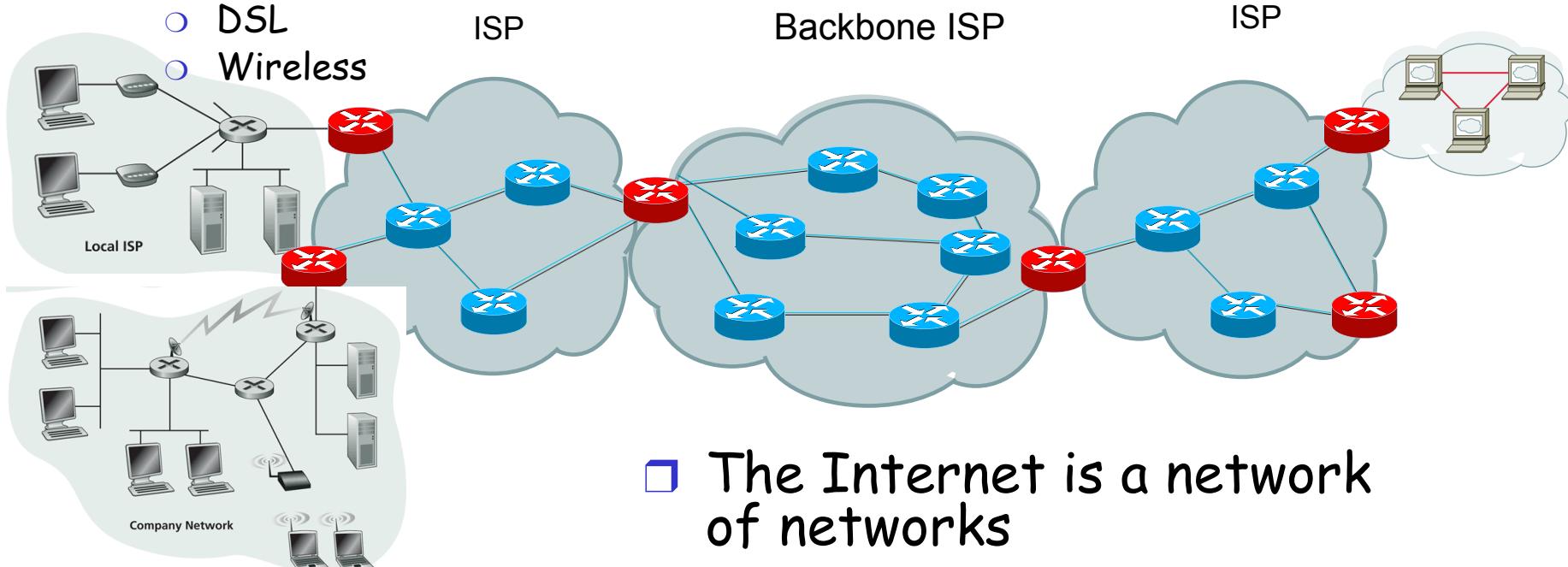
<http://googlecloudplatform.blogspot.com/2015/06/A-Look-Inside-Googles-Data-Center-Networks.html>

<http://conferences.sigcomm.org/sigcomm/2015/pdf/papers/p183.pdf>

Recall: Internet Physical Infrastructure

Residential access

- Cable
- Fiber
- DSL
- Wireless



Campus access,
e.g.,

- Ethernet
- Wireless

- The Internet is a network of networks
- Each individually administrated network is called an Autonomous System (AS)

Yale Internet Connection

```
cicada:~% traceroute www.tsinghua.edu.cn
```

```
traceroute to www.d.tsinghua.edu.cn (166.111.4.100), 64 hops max, 52 byte packets
 1 college.net.yale.internal (172.28.201.1) 10.310 ms 147.298 ms 3.948 ms
 2 10.1.1.13 (10.1.1.13) 3.846 ms 1.721 ms 1.603 ms
 3 level3-10g-asr.net.yale.internal (10.1.4.40) 2.830 ms 3.641 ms 126.830 ms
 4 cen-10g-yale.net.yale.internal (10.1.3.102) 3.116 ms 2.904 ms 2.843 ms
 5 * * *
 6 enrt064hhh-9k-te0-3-0-5.net.cen.ct.gov (67.218.83.254) 43.064 ms 3.999 ms 3.701 ms
 7 198.71.46.215 (198.71.46.215) 10.636 ms 3.819 ms 3.893 ms
 8 et-5-0-0.1180.rtr.newy32aoa.net.internet2.edu (198.71.46.214) 6.510 ms 6.686 ms 9.789 ms
 9 et-10-0-0.118.rtr.atla.net.internet2.edu (198.71.46.174) 26.527 ms 24.777 ms 24.925 ms
10 et-10-2-0.105.rtr.hous.net.internet2.edu (198.71.45.13) 49.217 ms 48.551 ms 48.667 ms
11 et-5-0-0.111.rtr.losa.net.internet2.edu (198.71.45.21) 81.462 ms 81.486 ms 82.688 ms
12 210.25.189.133 (210.25.189.133) 85.624 ms 84.093 ms 83.087 ms
...
```

Yale Internet Connection

cicada:~% traceroute www.tsinghua.edu.cn

Internet2



INTERNET2 NETWORK INFRASTRUCTURE TOPOLOGY

OCTOBER 2014



- Advanced Layer 3 Service
(Research/Education and peering IP)
- Advanced Layer 2 Service
(SDW Ethernet add/drop)
- Advanced Layer 1 Service
(Optical wave add/drop)

INTERNET2 NETWORK BY THE NUMBERS

12	JUNIPER MX400 ROUTERS SUPPORTING LAYER 3 SERVICE
34	BRODCAST AND JUNIPER SWITCHES SUPPORTING LAYER 2 SERVICE
62	CUSTOM COLLOCATION FACILITIES
250+	AMPLIFICATION RACKS
57,7	MILES OF NEWLY ACQUIRED DARK FIBER
0.8	TERRS OF OPTICAL CIRCUITS
103	GIGS OF HYBRID LAYER 3 AND LAYER 2 CAPACITY
500+	GEN4 ACTIVE/FRESH 8900 NETWORK ELEMENTS
1,600	MILES PARTNERED CAPACITY WITH ZAYO COMMUNICATIONS IN SUPPORT OF THE NORTHERN TIER REGION



IN SUPPORT OF
U.S.UCAN

NETWORK
PARTNERS

ciena

CISCO

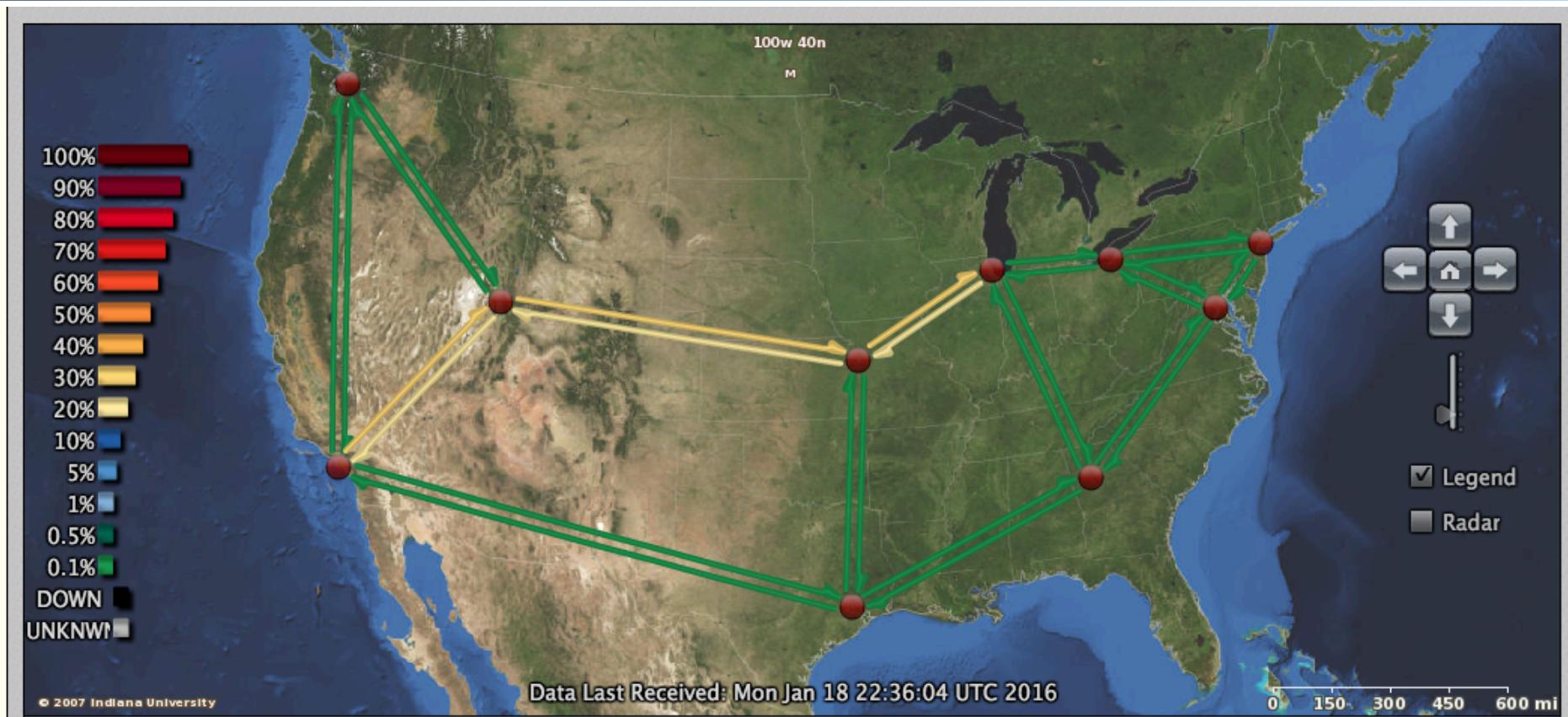
INDIANA UNIVERSITY

infinera

JUNIPER
NETWORKS



Internet2



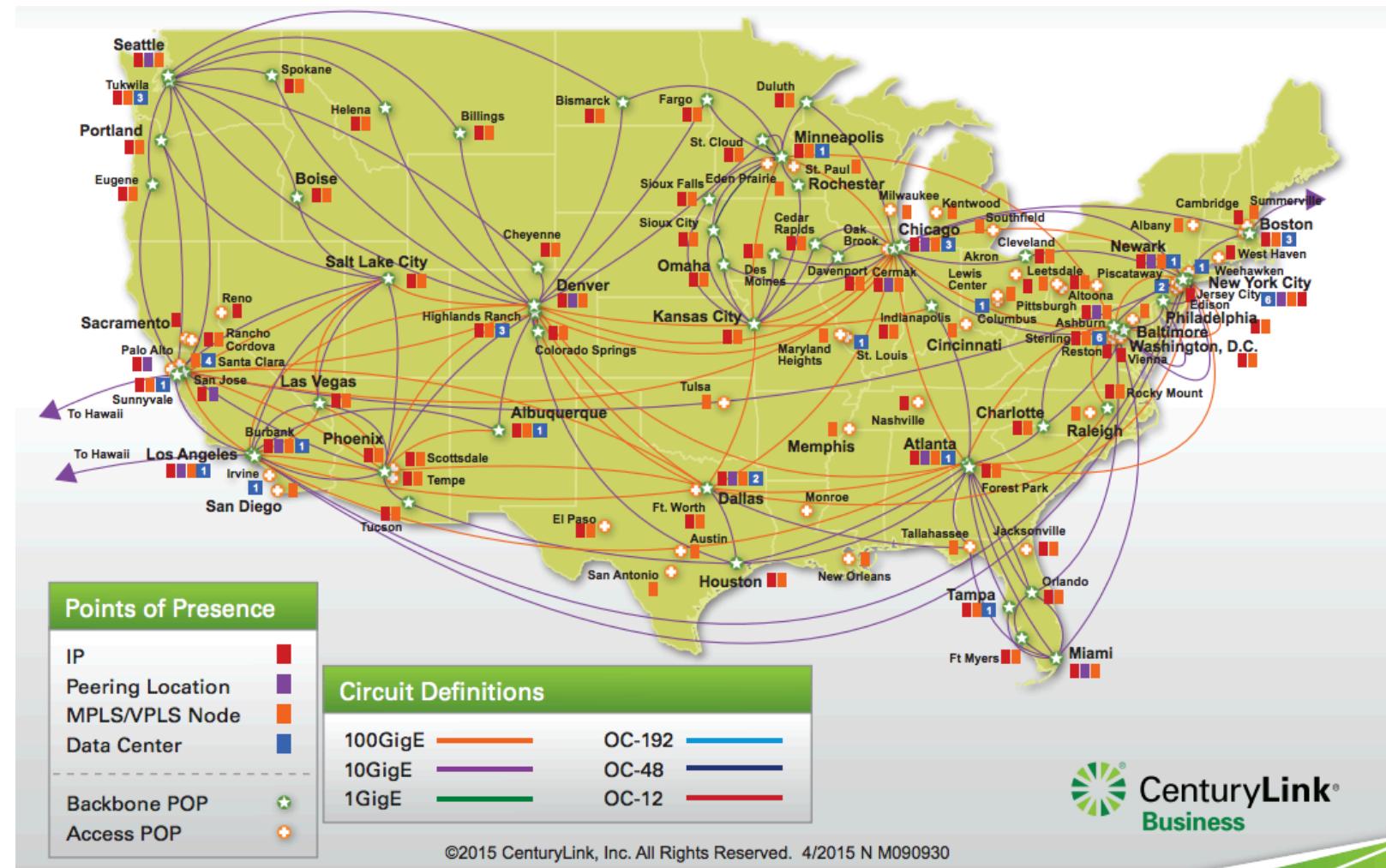
http://atlas.grnoc.iu.edu/atlas.cgi?map_name=Internet2%20IP%20Layer

Yale Internet Connection

Try traceroute from Yale to

- www.microsoft.com
- www.facebook.com
- www.amazon.com
- www.google.com

Qwest (CenturyLink) Network Maps

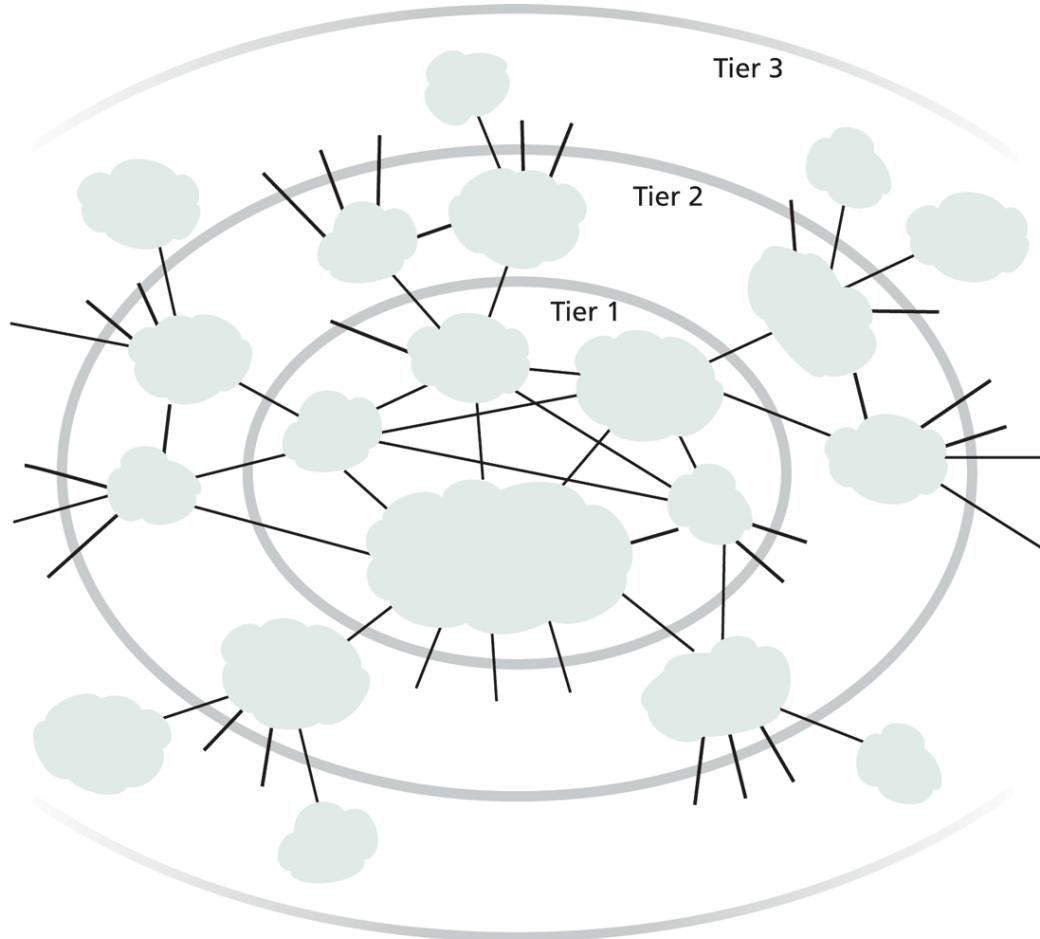


<http://www.centurylink.com/business/asset/network-map/ip-mpls-network-nm090930.pdf>

<http://www.centurylink.com/business/resource-center/network-maps/>

Internet ISP Connectivity

- Roughly hierarchical
 - Divided into tiers
 - Tier-1 ISPs are also called backbone providers, e.g., AT&T, Verizon, Sprint, Level 3, Qwest
- An ISP runs (private) Points of Presence (PoP) where its customers and other ISPs connect to it
- ISPs also connect at (public) Internet Exchange Point (IXP)
 - public peering



Outline

- Administrative trivia's
- What is a network protocol?
- *A brief introduction to the Internet*
 - past
 - *Present*
 - *topology*
 - *traffic*

Internet (Consumer) Traffic

Consumer Internet Traffic, 2012–2017								CAGR 2012–2017
	2012	2013	2014	2015	2016	2017		
By Network (PB per Month)								
Fixed	25,529	32,097	39,206	47,035	56,243	66,842		21%
Mobile	684	1,239	2,223	3,774	6,026	9,131		68%
By Subsegment (PB per Month)								
Internet video	14,818	19,855	25,800	32,962	41,916	52,752		29%
Web, email, and data	5,173	6,336	7,781	9,542	11,828	14,494		23%
File sharing	6,201	7,119	7,816	8,266	8,478	8,667		7%
Online gaming	22	26	32	39	48	59		22%
By Geography (PB per Month)								
Asia Pacific	9,033	11,754	14,887	18,707	23,458	29,440		27%
North America	6,834	8,924	11,312	14,188	17,740	21,764		26%
Western Europe	5,086	5,880	6,804	7,810	9,197	10,953		17%
Central and Eastern Europe	2,194	2,757	3,433	4,182	5,015	5,897		22%
Latin America	2,656	3,382	4,049	4,588	5,045	5,487		16%
Middle East and Africa	410	640	944	1,334	1,816	2,432		43%
Total (PB per Month)								
Consumer Internet traffic	26,213	33,337	41,429	50,809	62,269	75,973		24%

Internet Traffic in Perspective

640K ought to be enough
for anybody.



1 Petabyte
1,000 Terabytes or
250,000 DVDs

1 Exabyte
1,000 Petabytes or
250 million DVDs

1 Zettabyte
1,000 Exabytes or
250 billion DVDs

1 Yottabyte
1,000 Zettabytes or
250 trillion DVDs

480 Terabytes

A digital library of all of the world's catalogued books in all languages

100 Petabytes

The amount of data produced in a single minute by the new particle collider at CERN

5 Exabytes

A text transcript of all words ever spoken †

100 Exabytes

A video recording of all the meetings that took place last year across the world

400 Exabytes

The amount of data that crossed the Internet in 2012 alone

1 Zettabyte

The amount of data that has traversed the Internet since its creation

300 Zettabytes

The amount of visual information conveyed from the eyes to the brain of the entire human race in a single year ‡

20 Yottabytes

A holographic snapshot of the earth's surface

† Roy Williams, "Data Powers of Ten," 2000

‡ Based on a 2006 estimate by the University of Pennsylvania School of Medicine that the retina transmits information to the brain at 10 Mbps.

Outline

- Administrative trivia's
- What is a network protocol?
- A brief introduction to the Internet: past and present
- *Challenges of Internet networks and apps*

Scale



“Developers who have worked at the small scale might be asking themselves why we need to bother when we could just use some kind of out-of-the-box solution. For small-scale applications, this can be a great idea. We save time and money up front and get a working and serviceable application. The problem comes at larger scales—there are no off-the-shelf kits that will allow you to build something like Amazon... There’s a good reason why the largest applications on the Internet are all bespoke creations: no other approach can create massively scalable applications within a reasonable budget.”

Largest Internet Sites in U.S. (Jan. 2016)

RANK	SITE	MONTHLY PEOPLE	DIRECTLY MEASURED	RANK	SITE	MONTHLY PEOPLE	DIRECTLY MEASURED
1	google.com	232,288,144		51	stackexchange.com	21,869,768	✓
2	youtube.com	208,776,272		52	adobe.com	21,779,906	
3	msn.com	165,291,680		53	Hidden profile	—	✓
4	facebook.com	130,319,464		54	Hidden profile	—	✓
5	bing.com	93,572,256		55	healthline.com	20,552,928	✓
6	yahoo.com	86,557,024		56	nytimes.com	20,219,290	
7	amazon.com	82,737,184		57	usmagazine.com	19,856,848	✓
8	yelp.com	78,720,336	✓	58	imgur.com	19,683,136	✓
9	microsoft.com	68,743,296		59	comcast.net	18,676,528	
10	buzzfeed.com	63,977,968	✓	60	Hidden profile	—	✓
11	ebay.com	63,353,208		61	wellsfargo.com	18,444,774	
12	pinterest.com	63,027,644		62	vice.com	17,967,660	✓
13	answers.com	62,870,684	✓	63	rollingstone.com	17,771,256	✓
14	twitter.com	60,161,296		64	fandango.com	17,282,236	✓
15	wikia.com	48,973,188	✓	65	babycenter.com	17,052,306	✓
16	wikipedia.org	47,630,496		66	247sports.com	16,997,532	✓
17	live.com	44,831,308		67	viralands.com	16,939,364	✓
18	aol.com	42,879,008		68	goodreads.com	16,925,204	✓
19	linkedin.com	40,406,232		69	apple.com	16,837,476	
20	paypal.com	40,279,492		70	yourdailydish.com	16,275,180	✓
21	netflix.com	38,836,980		71	Hidden profile	—	✓
22	about.com	37,114,640		72	gizmodo.com	16,140,121	✓

General Complexity

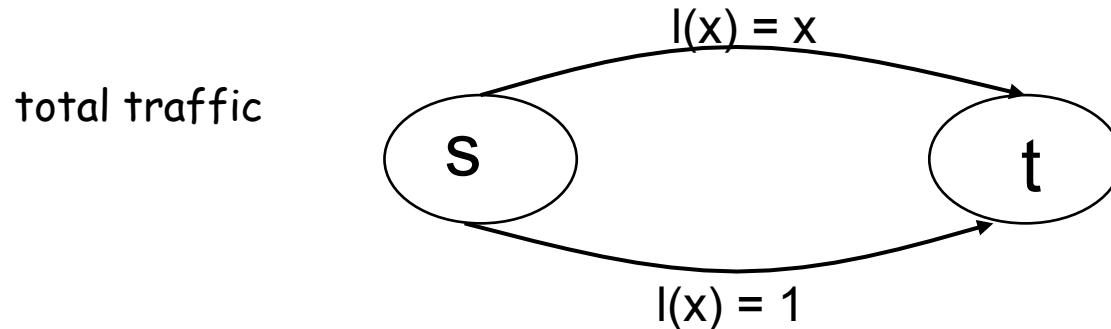


- Complexity in highly organized systems arises primarily from design strategies intended to create robustness to uncertainty in their environments and component parts.
 - Scalability is robustness to changes to the size and complexity of a system as a whole.
 - Evolvability is robustness of lineages to large changes on various (usually long) time scales.
 - Reliability is robustness to component failures.
 - Efficiency is robustness to resource scarcity.
 - Modularity is robustness to component rearrangements.

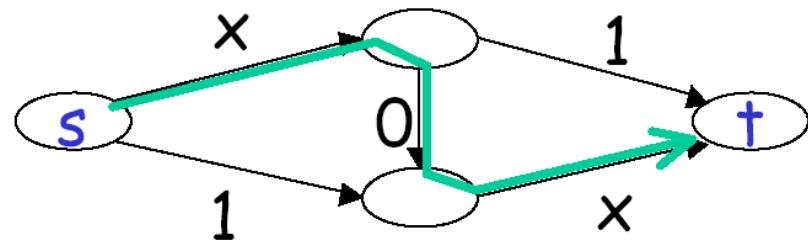
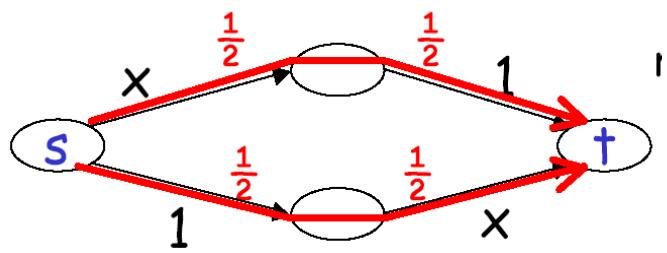
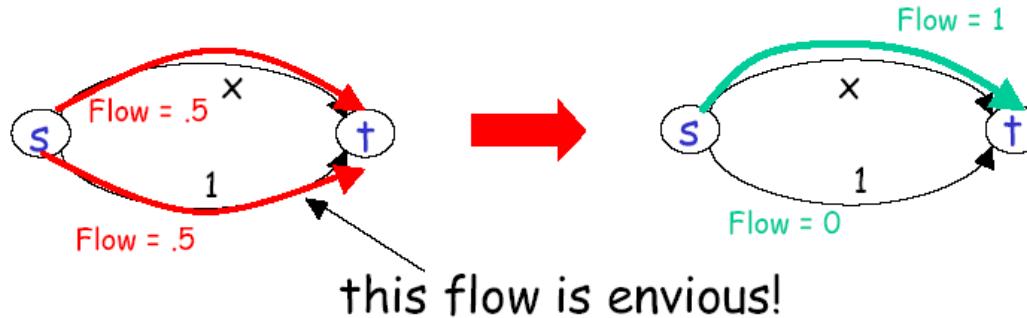
Centralized vs Decentralized (Price of Anarchy)



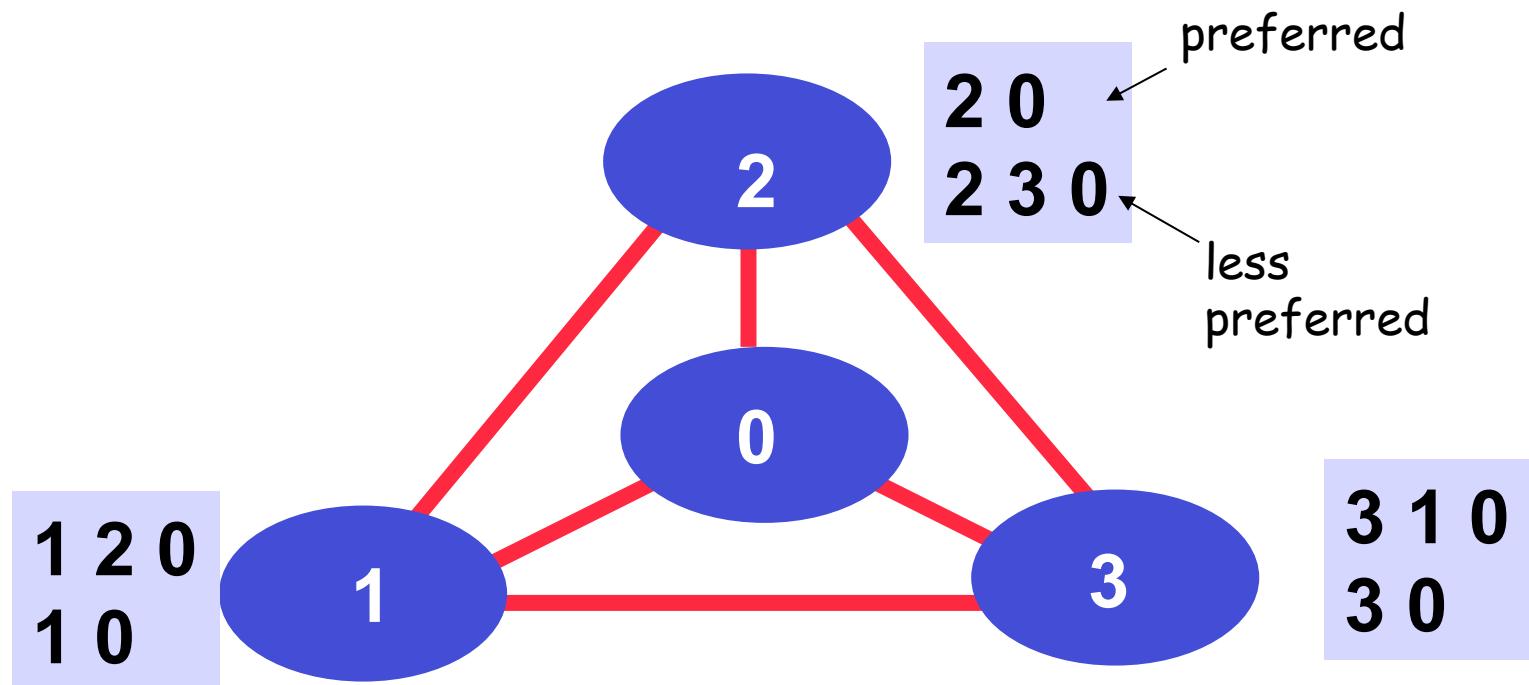
- Autonomous ("Selfish") App: Assume each link has a latency function $l_e(x)$: latency of link e when x amount of traffic goes through e :



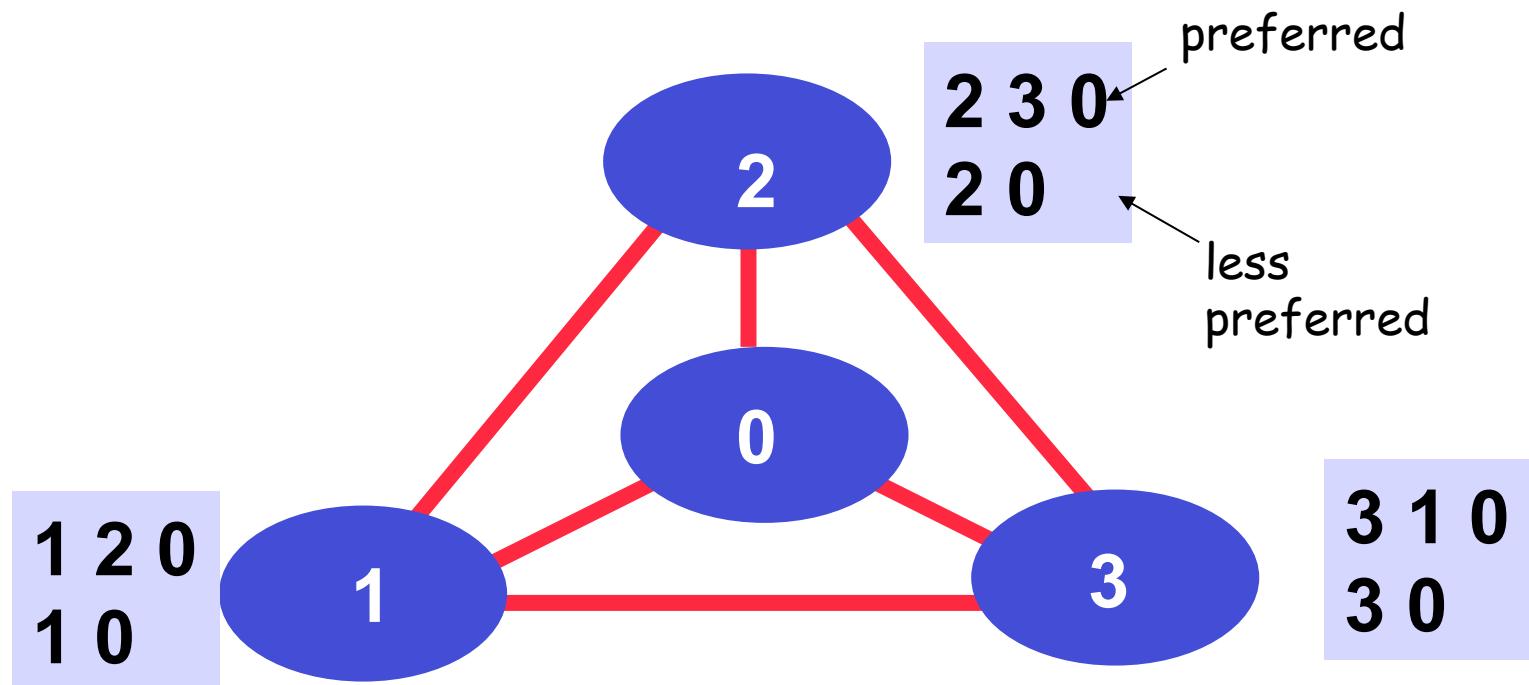
Autonomous ("Selfish") App



Decentralized ("Selfish") Users



Decentralized ("Selfish") Users



Distributed vs Centralized



- Distributed computing is hard, e.g.,
 - FLP Impossibility Theorem
 - Arrow's Impossibility Theorem
- Achieved good design for only few specific tasks (e.g., state distribution, leader election). Hence, a trend in networking is Software Defined Networking, which is a way of moving away from generic distributed computing, by focusing on utilizing the few well-understood primitives, in particular logically centralized state.

What Will We Cover?

- A tentative schedule is posted at class schedule page
- Network architecture and design principles
 - Layered network arch; e2e principle
- Application architecture and design principles
 - application paradigms; high performance network app.
 - HTTP/Web, Email, DNS, Content distribution
- Transport
 - transport services
 - reliability; distributed resource allocation; primal-dual
 - transport protocols: TCP/UDP

What Will We Cover?

- Network
 - network services
 - distributed, asynchronous, autonomous routing algorithms; scalable router design
 - IP/IPv6; mobile IP; cellular networks
- Link and physical
 - multiple access; queueing analysis; capacity analysis
 - Ethernet, 802.11, CDMA, bluetooth
- Cloud and data center design
- Network security
 - security primitives; BAN logic, SSL

Summary

- Course administration
- A protocol defines the format and the order of messages exchanged between two or more communicating entities, as well as the actions taken on the transmission and/or receipt of a message or other events.
- The past:
 - facts:
 - The Internet started as ARPANET in late 1960s
 - The initial link bandwidth was 50 kbps
 - The number of hosts at the end of 1969 was 4
 - some implications of the past:
 - ARPANET is sponsored by ARPA → design should survive failures
 - The initial IMPs were very simple → keep the network simple
 - Many networks → need a network to connect networks
- Current:
 - The number of hosts connected to the Internet is around 1 billions
 - The backbone speed of the current Internet is about 40/100 Gbps
 - The Internet is roughly hierarchical where ISPs interconnect at PoP and IXP
 - Needs to handle scale, complexity, decentralization, security

Preview

- We have only looked at the topology/
connectivity of the Internet
 - a communication network is a mesh of
interconnected devices

- *A fundamental question:* how is data
transferred through a network?

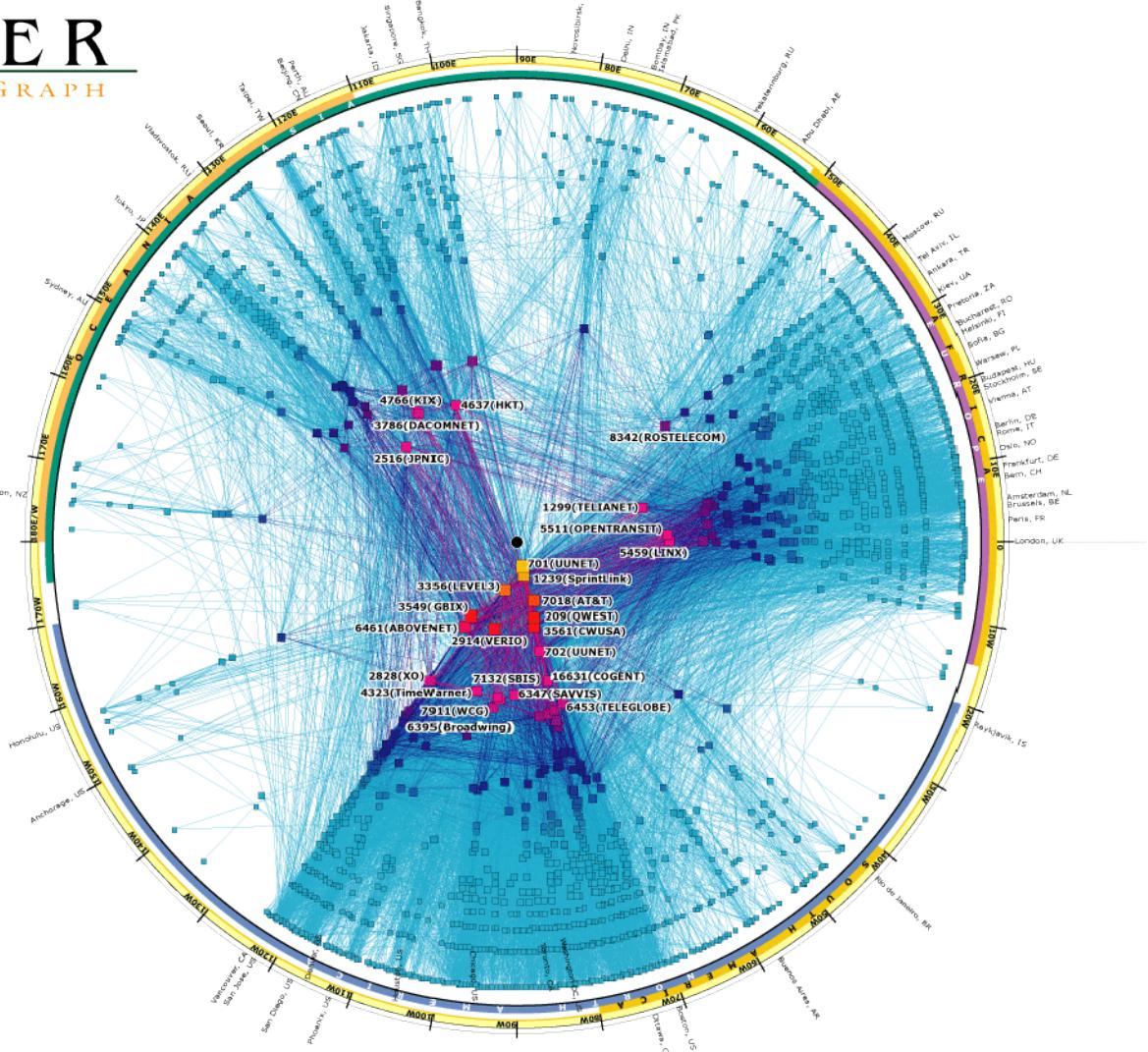
Backup Slides

Challenge of the Internet: Characterizing Internet Topology

copyright ©2003 UC Regents. all rights reserved.

SKITTER AS INTERNET GRAPH

Peering: OutDegree



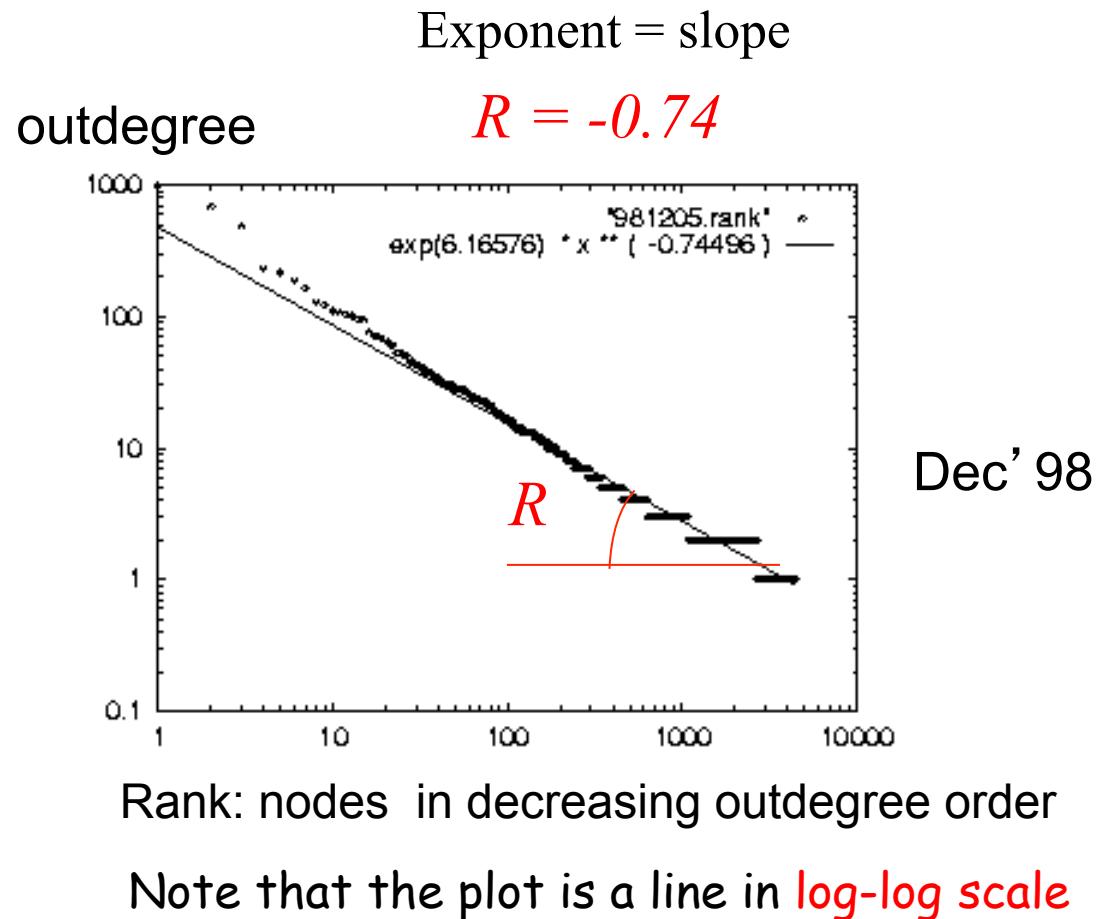
cooperative association for internet data analysis O san diego supercomputer center O university of california, san diego
9500 gilman drive, mc0605 O la jolla, ca 92093-0505 O tel. 858-534-6000 O <http://www.caida.org/>

CAIDA is a program of the University of California's San Diego Supercomputer Center (UCSD/SDSC)

CAIDA's topology mapping projects are supported by DARPA, NCS, NSF, WIDE and CAIDA members

Challenge of the Internet: Power Law?

- Some researchers found that the Internet AS connectivity graph satisfies Power Law
- Does it really satisfy power law? If so, why?



An Example: Network News Transport Protocol (NNTP)

- Messages from a client to a news server
 - help
 - list active <pattern>
 - group <group_name>
 - article <article_number>
 - next
 - post
- Messages from a news server to a client
 - status code
 - The first digit of the response broadly indicates the success, failure, or progress of the previous command.
 - 1xx - Informative message
 - 2xx - Command ok
 - 3xx - Command ok so far, send the rest of it.
 - 4xx - Command was correct, but couldn't be performed for some reason.
 - 5xx - Command unimplemented, or incorrect, or a serious program error occurred.
 - content

Internet (Global) Traffic

IP Traffic, 2012–2017								CAGR 2012–2017
	2012	2013	2014	2015	2016	2017		
By Type (PB per Month)								
Fixed Internet	31,339	39,295	47,987	57,609	68,878	81,818		21%
Managed IP	11,346	14,679	18,107	21,523	24,740	27,668		20%
Mobile data	885	1,578	2,798	4,704	7,437	11,157		66%
By Segment (PB per Month)								
Consumer	35,047	45,023	56,070	68,418	82,683	98,919		23%
Business	8,522	10,530	12,822	15,417	18,372	21,724		21%
By Geography (PB per Month)								
Asia Pacific	13,906	18,121	22,953	28,667	35,417	43,445		26%
North America	14,439	18,788	23,520	28,667	34,457	40,672		23%
Western Europe	7,722	9,072	10,568	12,241	14,323	16,802		17%
Central and Eastern Europe	3,405	4,202	5,167	6,274	7,517	8,844		21%
Latin America	3,397	4,321	5,201	5,975	6,682	7,415		17%
Middle East and Africa	701	1,049	1,483	2,013	2,659	3,465		38%
Total (PB per Month)								
Total IP traffic	43,570	55,553	68,892	83,835	101,055	120,643		23%

Source: Cisco VNI, 2013

Yale Internet Connectivity: Qwest

```
cyndra.cs.yale.edu% /usr/sbin/traceroute www.synopsis.com  
traceroute to www.synopsis.com (198.182.60.11), 30 hops max, 38 byte packets
```

- 1 anger.net.yale.edu (128.36.229.1) 0.767 ms 1.740 ms 1.452 ms
- 2 bifrost.net.yale.edu (130.132.1.100) 0.680 ms 0.597 ms 0.567 ms
- 3 bos-edge-02.inet.qwest.net (63.145.0.13) 4.897 ms 5.257 ms 5.294 ms
- 4 bos-core-01.inet.qwest.net (205.171.28.13) 4.918 ms 5.405 ms 4.898 ms
- 5 ewr-core-02.inet.qwest.net (205.171.8.114) 11.998 ms 11.688 ms 11.647 ms
- 6 ewr-brdr-02.inet.qwest.net (205.171.17.130) 11.432 ms 12.036 ms 11.474 ms
- 7 205.171.1.98 (205.171.1.98) 7.547 ms 7.727 ms 7.632 ms
- 8 ae-1-56.bbr2.NewYork1.Level3.net (4.68.97.161) 7.513 ms 7.466 ms
ae-1-54.bbr2.NewYork1.Level3.net (4.68.97.97) 7.585 ms
- 9 ge-0-1-0.bbr2.SanJose1.Level3.net (64.159.1.130) 75.468 ms
so-0-0-0.bbr1.SanJose1.Level3.net (64.159.1.133) 75.630 ms
ge-0-1-0.bbr2.SanJose1.Level3.net (64.159.1.130) 75.126 ms
- 10 ge-9-0.hsa1.SanJose1.Level3.net (4.68.123.40) 75.499 ms
ge-8-0.hsa1.SanJose1.Level3.net (4.68.123.8) 76.429 ms 76.431 ms
- 11 h1.synopsysmv.bbnplanet.net (4.25.120.46) 86.414 ms 85.996 ms 85.896 ms
- 12 198.182.56.45 (198.182.56.45) 88.705 ms 92.585 ms 90.412 ms

Note: which link Yale will use depends on its current load balancing. It may not be qwest.

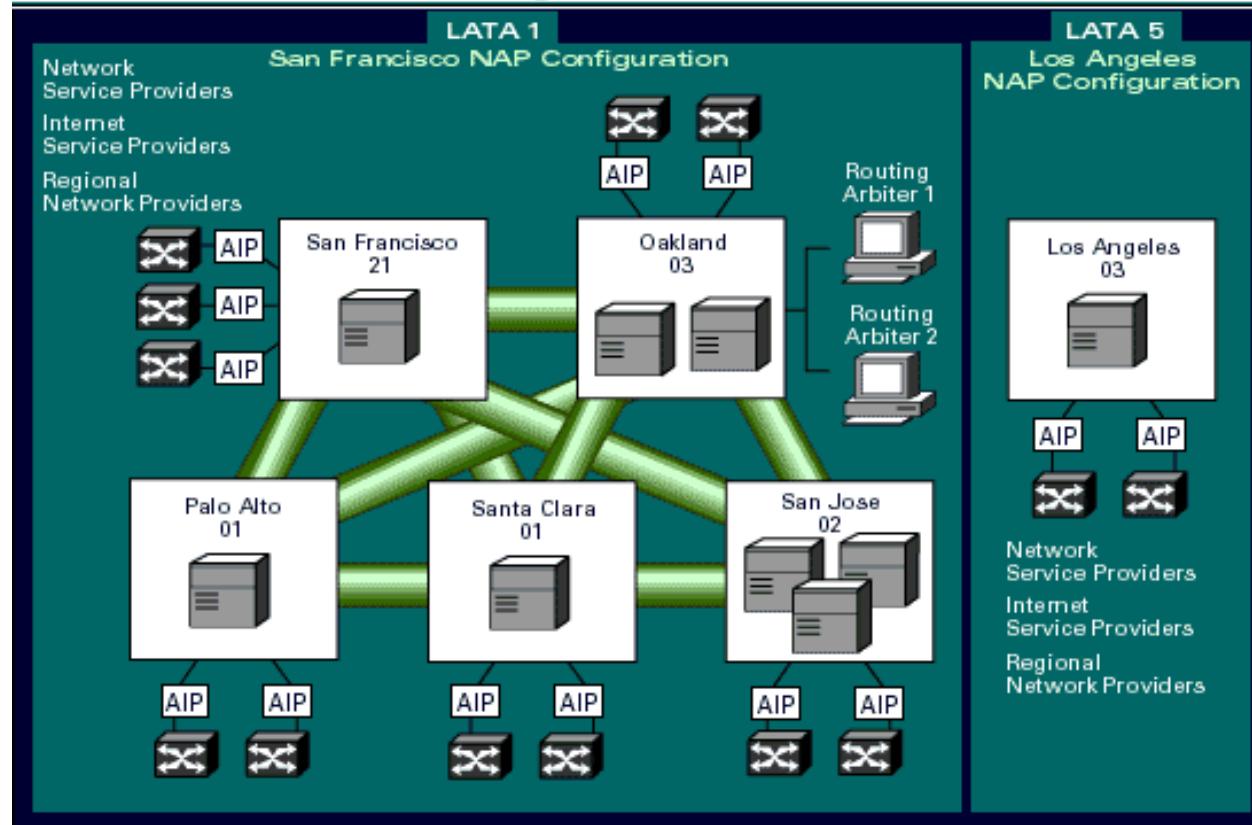
Yale Internet Connectivity: AT&T

```
cicada.cs.yale.edu% /usr/sbin/traceroute www.amazon.com
```

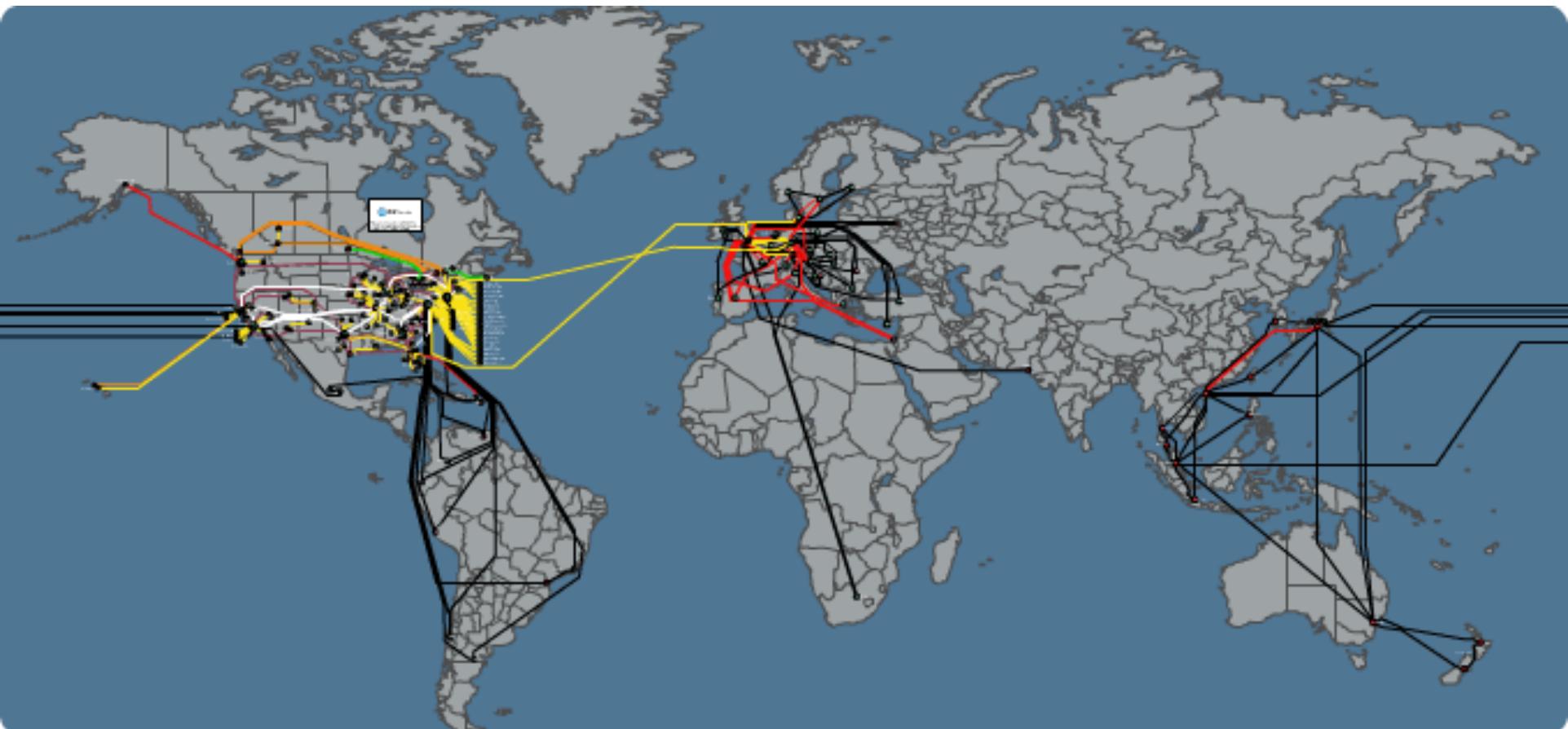
- 1 anger.net.yale.edu (128.36.229.1) 0.906 ms 1.028 ms 0.784 ms
- 2 bifrost.net.yale.edu (130.132.1.100) 0.798 ms 0.722 ms 0.836 ms
- 3 12.175.96.1 (12.175.96.1) 0.861 ms 0.869 ms 0.804 ms
- 4 12.124.179.65 (12.124.179.65) 2.278 ms 2.276 ms 2.223 ms
- 5 gbr5-p80.n54ny.ip.att.net (12.123.1.202) 2.524 ms 2.314 ms 2.169 ms
- 6 tbr1-p013201.n54ny.ip.att.net (12.122.11.9) 3.212 ms 3.203 ms 3.560 ms
- 7 ggr2-p310.n54ny.ip.att.net (12.123.3.105) 3.045 ms 2.468 ms 2.419 ms
- 8 sl-bb20-nyc-12-0.sprintlink.net (144.232.8.49) 3.518 ms 2.748 ms 2.951 ms
- 9 sl-bb26-nyc-6-0.sprintlink.net (144.232.13.9) 4.690 ms 4.460 ms 49.531 ms
- 10 sl-bb23-pen-12-0.sprintlink.net (144.232.20.95) 7.191 ms 7.202 ms 7.033 ms
- 11 sl-bb22-pen-14-0.sprintlink.net (144.232.8.178) 7.131 ms 7.245 ms 7.096 ms
- 12 sl-bb21-pen-15-0.sprintlink.net (144.232.16.29) 7.100 ms 7.423 ms 8.049 ms
- 13 sl-bb23-rly-0-0.sprintlink.net (144.232.20.32) 10.777 ms 10.826 ms 11.049 ms
- 14 sl-st20-ash-11-0.sprintlink.net (144.232.20.150) 11.281 ms 10.948 ms 10.730 ms
- 15 sl-amazon-4-0.sprintlink.net (144.223.246.18) 10.562 ms 10.572 ms 11.381 ms

Network Access Point

Pacific Bell NAP Configuration

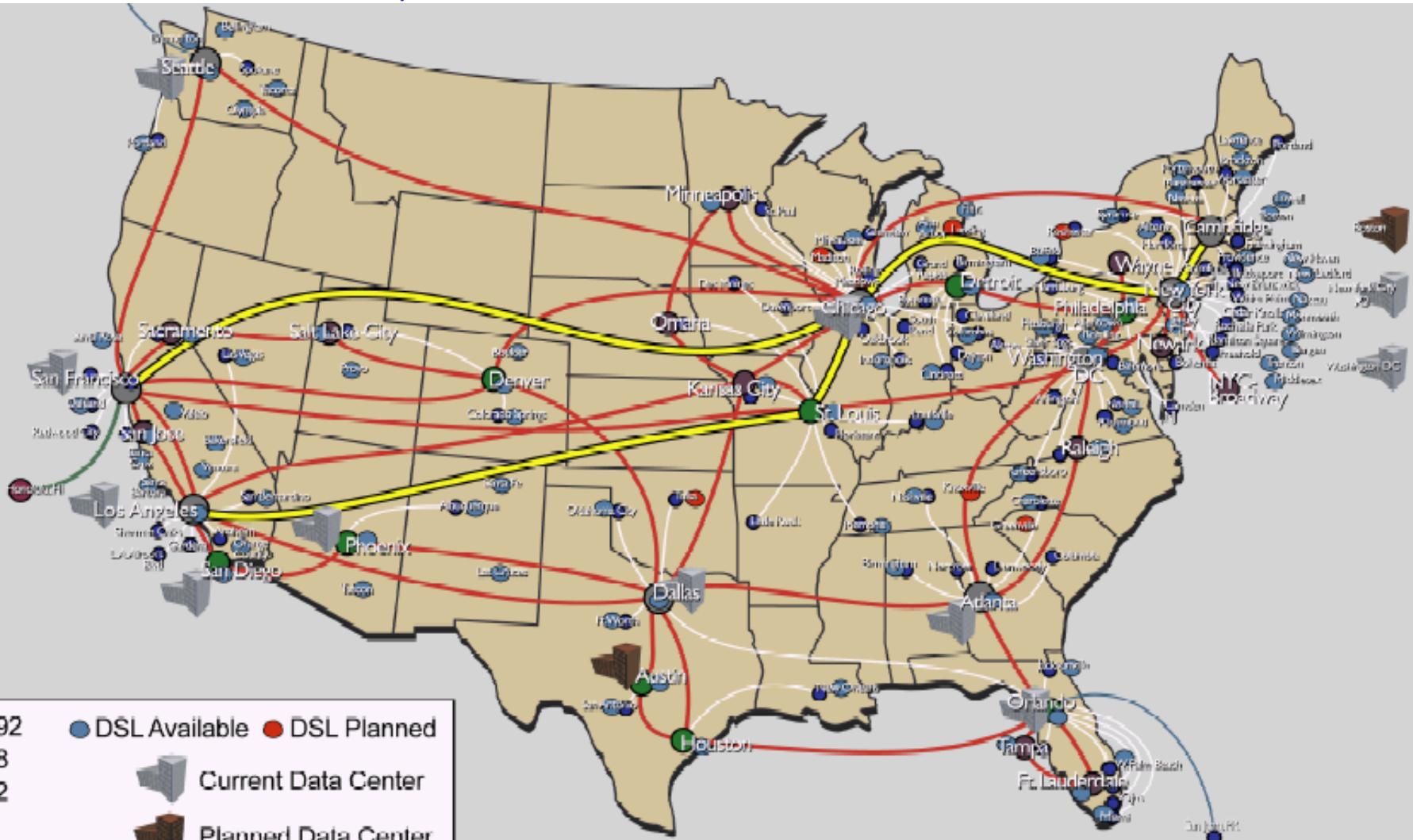


ATT Global Backbone IP Network



From <http://www.business.att.com>

AT&T USA Backbone Map



Present Internet: Likely Web-based

- The Internet infrastructure has better support for HTTP than other protocols
- A trend of software applications:
 - From the desktop to the browser
 - SaaS == Web-based applications
 - Examples: Google Maps/Doc, Facebook
- How do we deliver highly-interactive Web-based applications?
 - AJAX (asynchronous JavaScript and XML)
 - For better, or for worse...

Evolving Computing Models

- Do it yourself (build your own data centers)
- Utility computing
 - Why buy machines when you can rent cycles?
 - Examples: Amazon's EC2, GoGrid, AppNexus
- Platform as a Service (PaaS)
 - Give me nice API and take care of the implementation
 - Example: Google App Engine
- Software as a Service (SaaS)
 - Just run it for me!
 - Example: Gmail; MS Exchange; MS Office Online

Data centers

- [http://www.youtube.com/watch?
v=WBIl0curTxU](http://www.youtube.com/watch?v=WBIl0curTxU)

- [http://www.youtube.com/watch?
v=PBx7rgqeGG8](http://www.youtube.com/watch?v=PBx7rgqeGG8)

- Google
- [http://www.youtube.com/watch?
v=zRwPSFpLX8I](http://www.youtube.com/watch?v=zRwPSFpLX8I)

Yale Internet Connection

cicada:~% traceroute www.cs.utexas.edu

traceroute to www.cs.utexas.edu (128.83.139.21), 30 hops max, 60 byte packets

```
1 anger.net.yale.edu (128.36.232.1) 3.735 ms 3.731 ms 3.729 ms
2 10.1.2.81 (10.1.2.81) 0.359 ms 0.370 ms 0.416 ms
3 10.1.2.113 (10.1.2.113) 2.584 ms 2.603 ms 2.657 ms
4 CEN10G-ASR.net.yale.internal (10.1.4.30) 1.841 ms 2.316 ms 2.124 ms
5 level3-10g-yale.net.yale.internal (10.1.3.103) 1.850 ms 2.265 ms 2.137 ms
6 te-4-1.car1.Stamford1.Level3.net (4.26.48.81) 40.900 ms 40.651 ms 40.723 ms
7 * * *
8 4.71.198.54 (4.71.198.54) 37.201 ms 37.202 ms 37.184 ms
9 aust-utnoc-core-ae2-765.tx-bb.net (192.124.227.126) 45.004 ms 45.049 ms 45.045 ms
10 ser2-v60.gw.utexas.edu (192.12.10.2) 49.800 ms 49.751 ms 49.770 ms
11 nocb10-e10-2-noca2.gw.utexas.edu (128.83.8.9) 50.359 ms 50.686 ms 50.644 ms
12 cs-nocb10-v690.gw.utexas.edu (146.6.10.34) 50.533 ms 52.657 ms 52.546 ms
13 cs65k-cs45k-po1-p2p.gw.utexas.edu (128.83.37.66) 50.552 ms 49.816 ms 49.813 ms
14 net9.cs.utexas.edu (128.83.139.21) 49.755 ms 49.792 ms 49.851 ms
```

Yale Internet Connection

August 2013

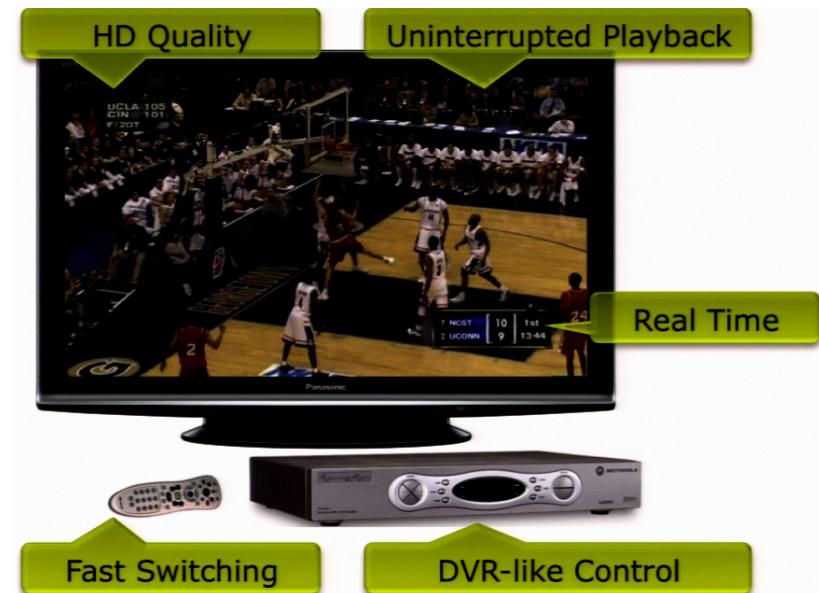
cicada:~% traceroute www.cs.utexas.edu

traceroute to net6.cs.utexas.edu (128.83.120.139), 64 hops max, 52 byte packets

```
1 arubacentral-vlan30-router.net.yale.internal (172.28.204.129) 1.540 ms 1.200 ms 1.344 ms
2 10.1.1.13 (10.1.1.13) 2.854 ms 1.072 ms 1.237 ms
3 qwest-asr.net.yale.internal (10.1.4.5) 1.139 ms 1.327 ms 1.281 ms
4 10.1.3.99 (10.1.3.99) 2.120 ms 1.343 ms 1.874 ms
5 cen-yale.net.yale.edu (130.132.251.74) 1.558 ms 1.634 ms 1.592 ms
6 nox300gw1-vl-706-nox-yale.nox.org (207.210.143.89) 5.570 ms 6.367 ms 5.208 ms
7 nox300gw1-vl-706-nox-yale.nox.org (207.210.143.89) 5.000 ms 5.008 ms 5.663 ms
8 nox1sumgw1-vl-803-nox.nox.org (192.5.89.237) 5.765 ms 5.909 ms 5.145 ms
9 nox1sumgw1-peer-nox-internet2-192-5-89-18.nox.org (192.5.89.18) 27.455 ms 27.232 ms
   27.344 ms
10 64.57.28.36 (64.57.28.36) 38.111 ms 126.638 ms 37.985 ms
11 xe-1-1-0.0.rtr.hous.net.internet2.edu (64.57.28.57) 51.982 ms 106.096 ms 51.817 ms
12 rt1-hardy-hstn-xe-0-1-0-3018.tx-learn.net (74.200.187.6) 52.988 ms 52.937 ms 53.307 ms
13 tx-bb-i2-hstn.tx-learn.net (74.200.187.26) 53.444 ms 53.515 ms 53.288 ms
14 aust-utnoc-core-ge-5-0-0-706.tx-bb.net (192.88.12.50) 54.636 ms 54.703 ms 55.054 ms
15 192.88.12.26 (192.88.12.26) 55.056 ms 74.044 ms 54.926 ms
16 ser10-v702.gw.utexas.edu (128.83.10.1) 55.208 ms 54.803 ms 55.117 ms
17 cs-nocb10-v690.gw.utexas.edu (146.6.10.34) 55.013 ms 55.099 ms 55.045 ms
18 cs65k-cs45k-po1-p2p.aces.utexas.edu (128.83.37.66) 54.960 ms 55.005 ms 55.551 ms
19 net6.cs.utexas.edu (128.83.120.139) 55.015 ms 54.956 ms 54.847 ms
```

Increasing QoE Demand

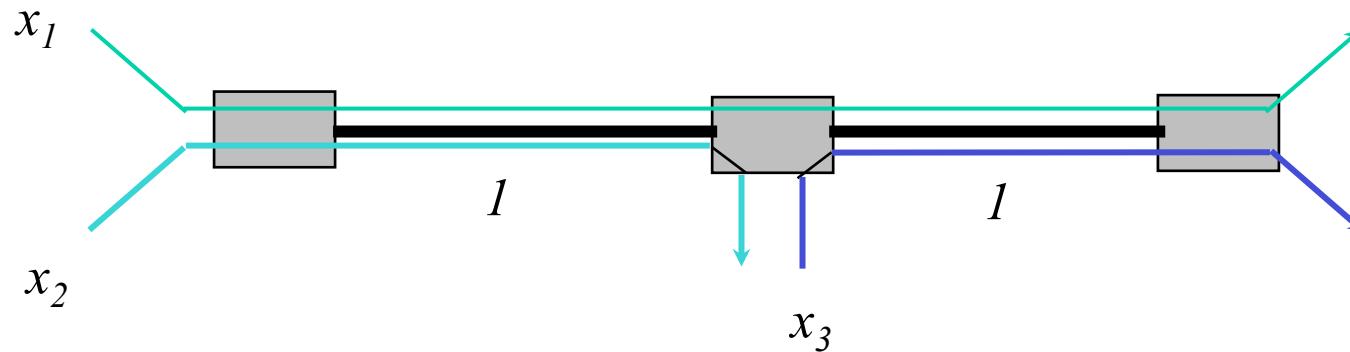
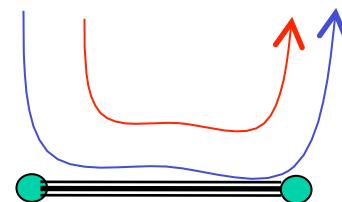
- 20 ms increase in latency
=>
1% drop in click-through rate



Politics: Sharing a Shared Infrastructure



- ☐ question: how to allocate network resources among users?

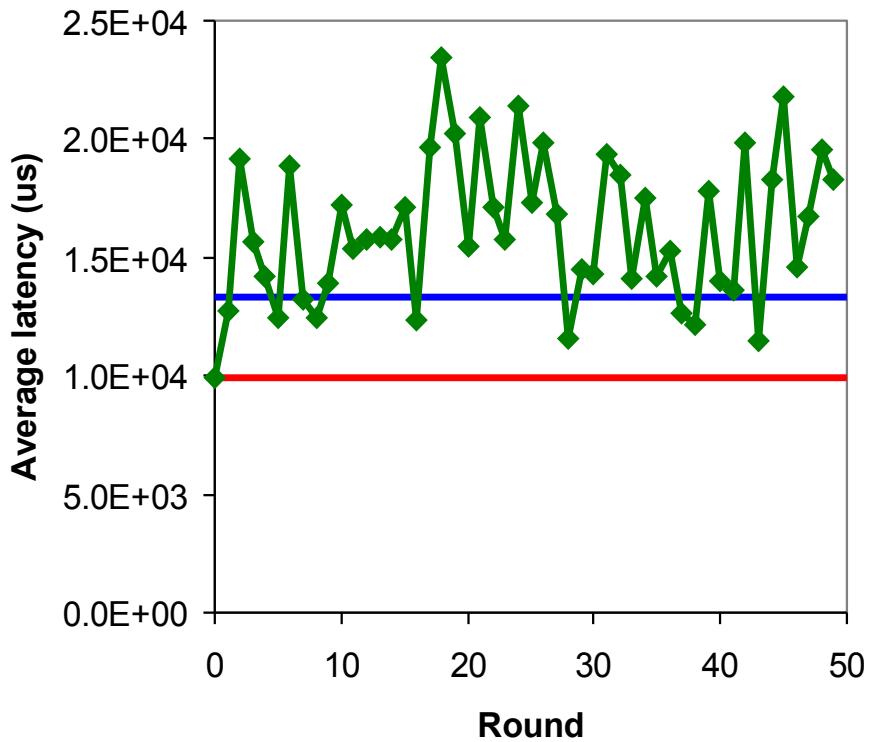


Poor App and Network Interaction

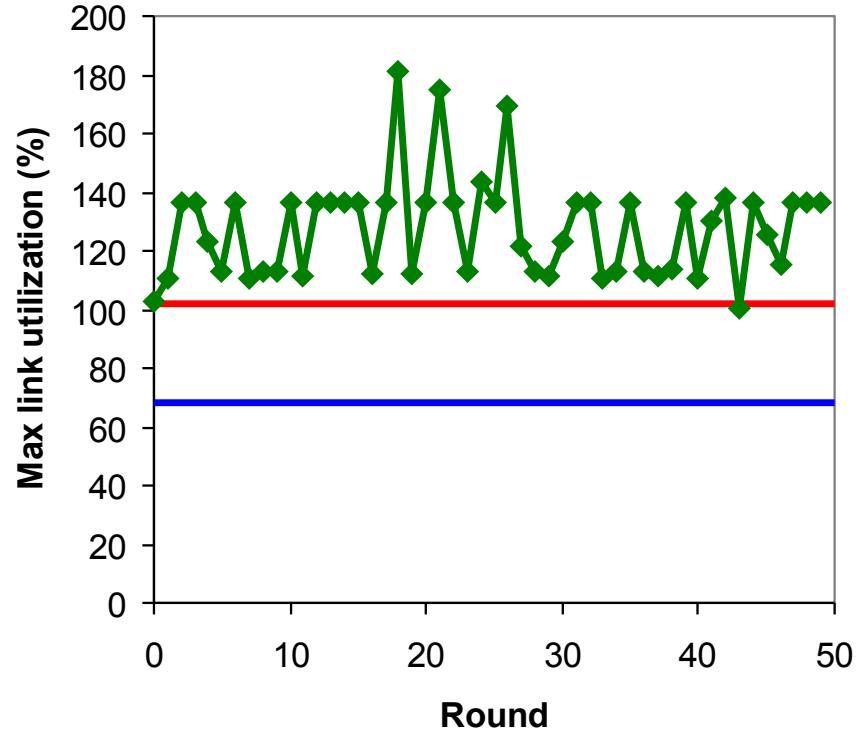
- Network Providers change routing to shift traffic away from highly utilized links
- Adaptive/decentralized apps direct traffic to lower latency paths
- ▶ Equilibrium points can be inefficient



App and Network Interaction



— selfish alone — TE alone
◆ selfish + TE (OSPF)

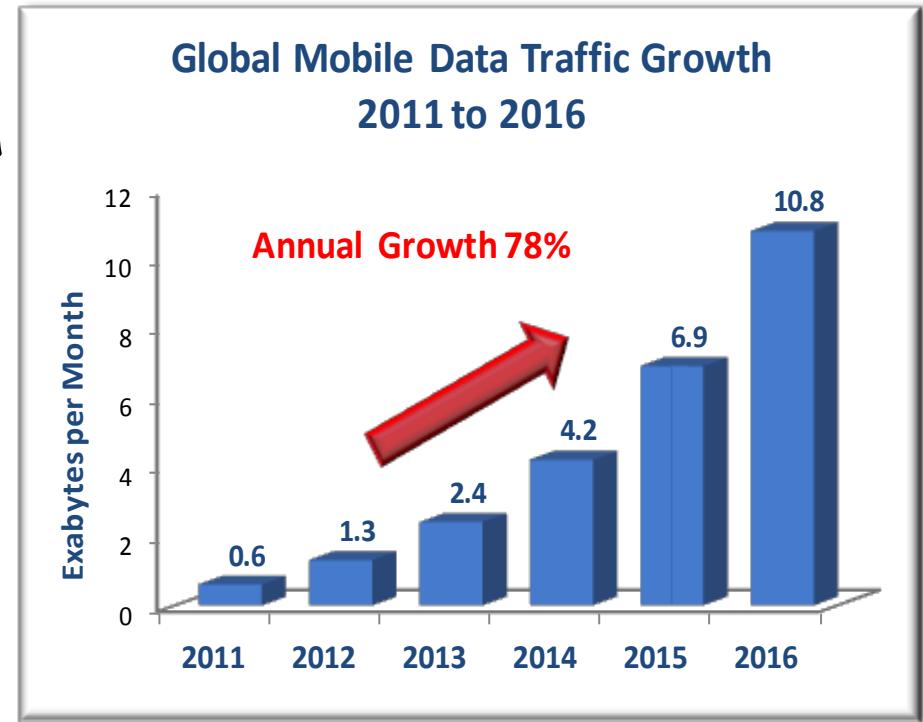


— selfish alone — TE alone
◆ selfish + TE (OSPF)

Fast Wireless Data Growth

□ AT&T

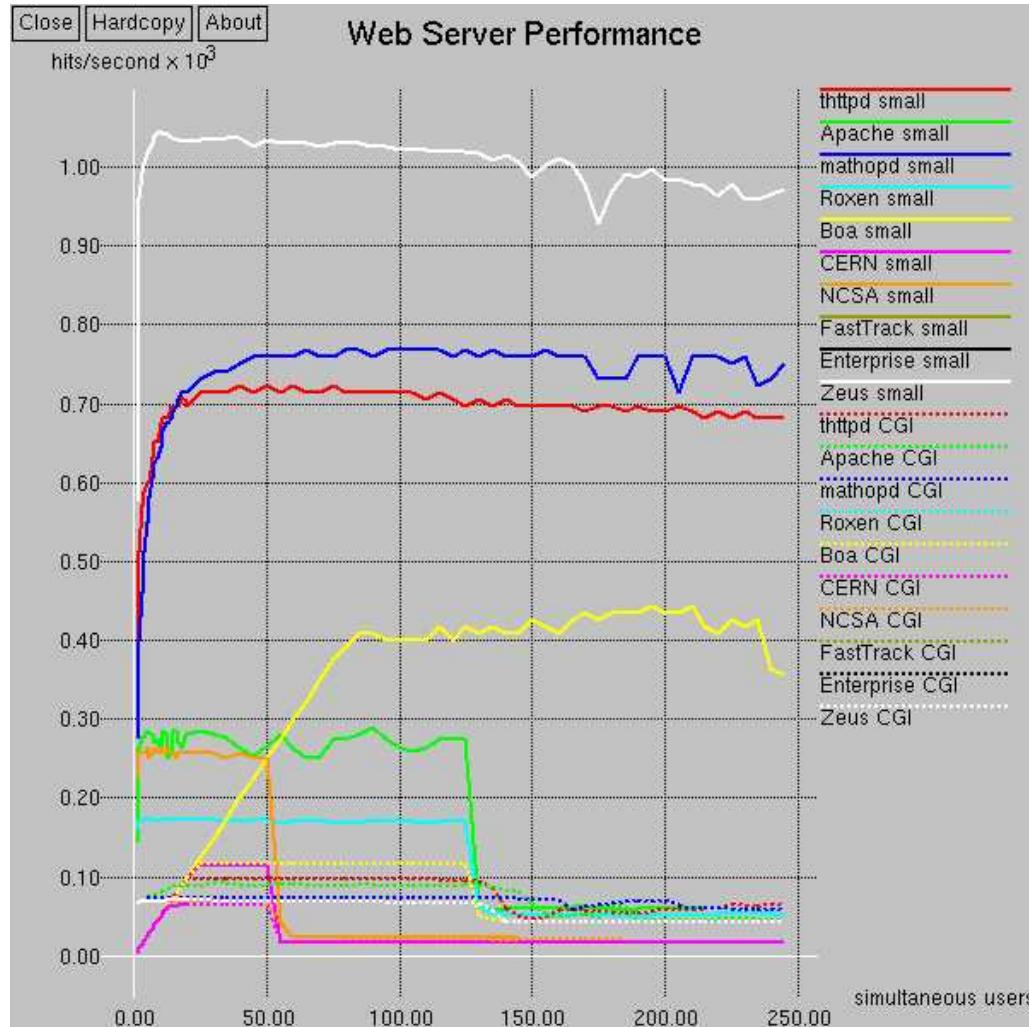
- Wireless data growth
20,000% in the past
5 years



Problems: Bandwidth limitations and poor TCP performance.

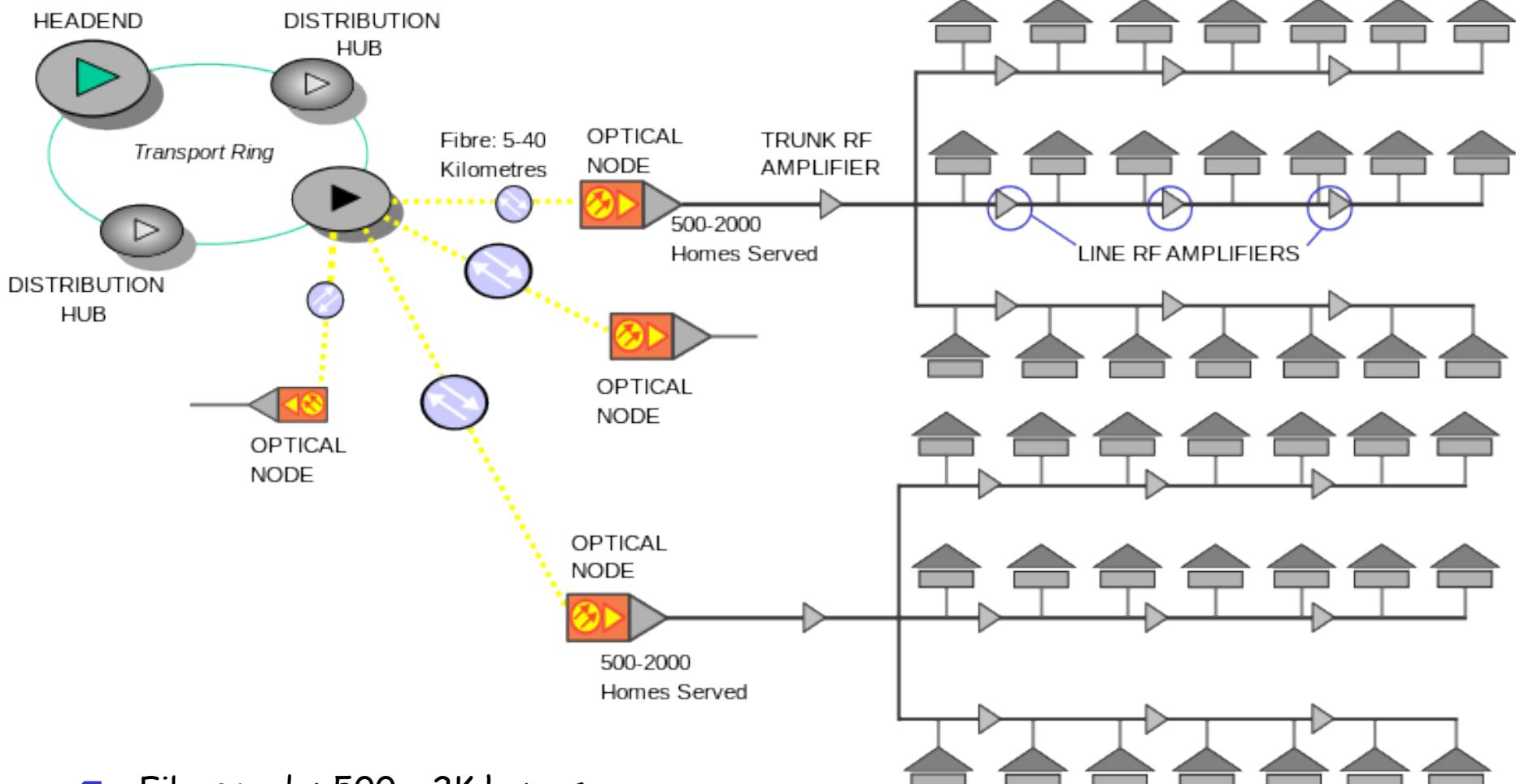
Source: CISCO Visual Networking Index (VNI) Global Mobil Data Traffic Forecast 2011 to 2016

Flexibility vs Performance



Access: Cable

Also called
Hybrid
Fiber-coaxial
Cable (HFC)



- Fiber node: 500 - 2K homes
- Distribution hub: 20K - 40 K homes
- Regional headend: 200 K - 400 K homes

A Taxonomy of Communication Networks

Y. Richard Yang

<http://zoo.cs.yale.edu/classes/cs433/>

01/22/2016

Outline

- *Admin and recap*
- A taxonomy of communication networks

Admin

- Please check the Schedule page for links to related readings

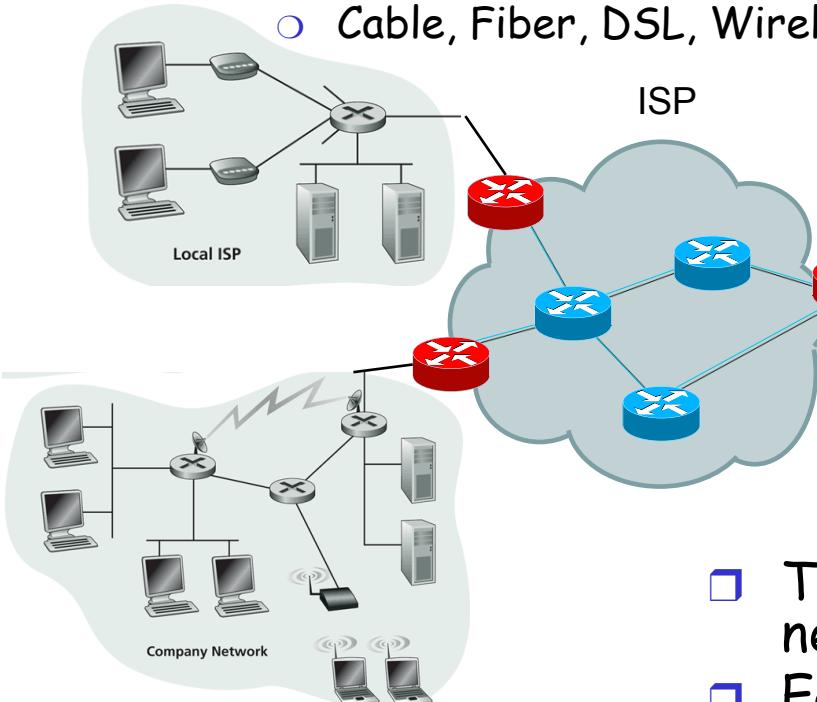
Recap

- A protocol defines the format and the order of messages exchanged between two or more communicating entities, as well as the actions taken on the transmission or receipt of a message or other events.
- Some implications of the past:
 - ARPANET is sponsored by ARPA → design should survive failures
 - The initial IMPs (routers) were made by a small company → keep the network simple
 - Many networks → internetworking: need a network to connect networks
 - Commercialization → architecture supporting decentralized, autonomous systems

Recall: Internet Physical Infrastructure

Residential access, e.g.,

- Cable, Fiber, DSL, Wireless



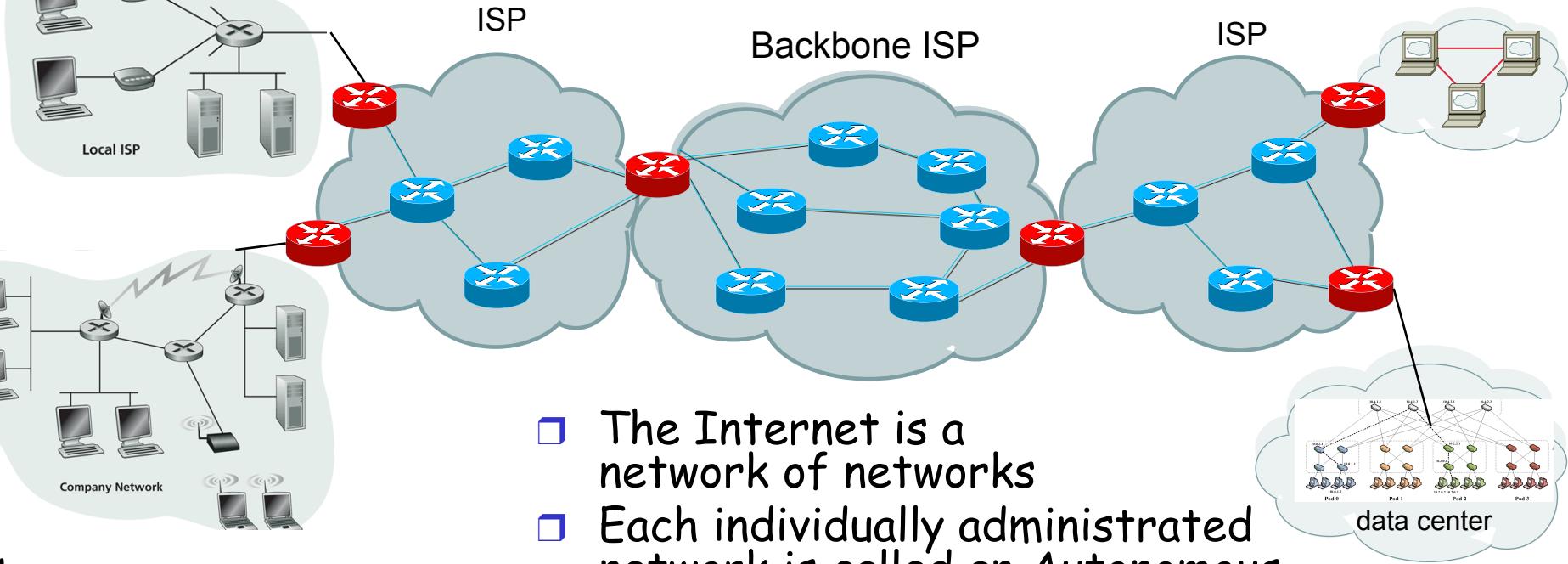
Campus access, e.g.,

- Ethernet, Wireless

<https://www.google.com/loon/how/>

- The Internet is a network of networks
- Each individually administrated network is called an Autonomous System (AS)

~ 52000 ASes (<http://bgp.potaroo.net/as2.0/bgp-active.html>)

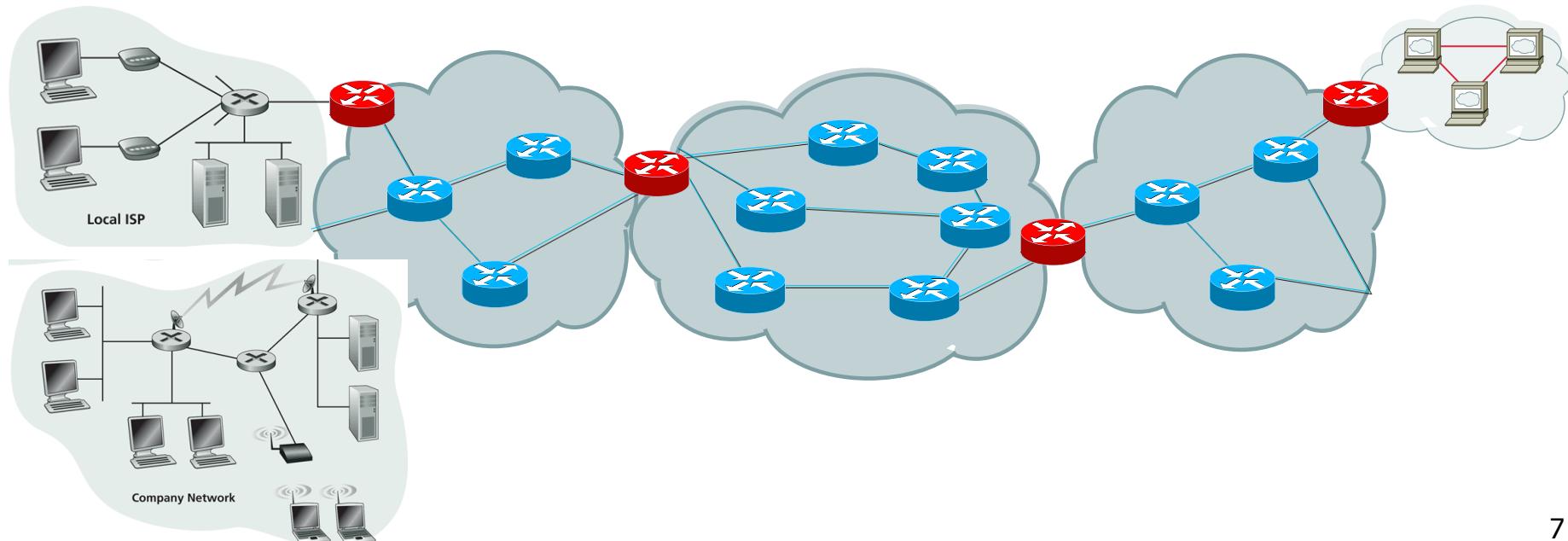


Observing the Internet State

- Read the manual of traceroute, and try it on a zoo machine
% /usr/sbin/traceroute <machine_name>
- Look at the web sites of the routers you see through traceroute
- Lookup ASN info
<https://www.ultratools.com/tools/asnInfo>
- fixedorbit to look for info (e.g., neighbors) about a network:
 - <http://www.fixedorbit.com/search.htm>

Roadmap

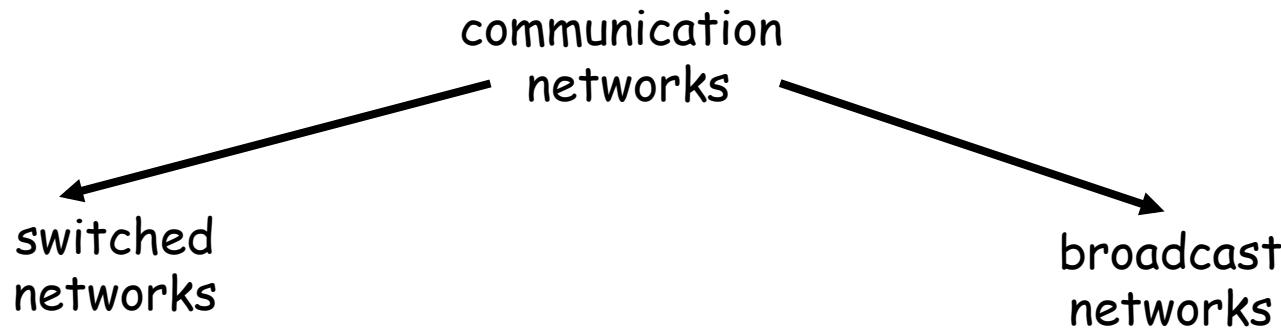
- So far we have looked at only the topology and physical connectivity of the Internet: a mesh of computers interconnected via various physical media
- **A fundamental question:** how are data (the bits) transferred through communication networks?



Outline

- Admin. and recap
- *A taxonomy of communication networks*

Taxonomy of Communication Networks



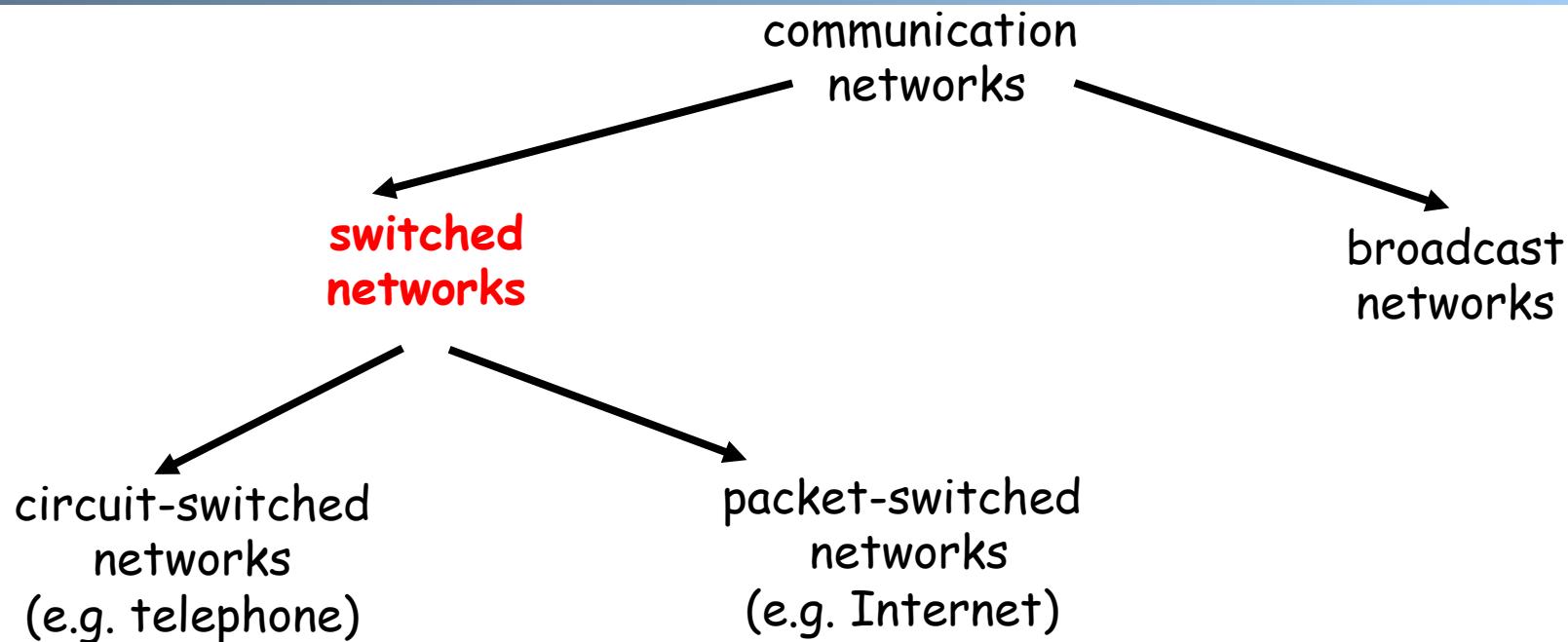
□ Broadcast networks

- nodes share a common channel; information transmitted by a node is received by all other nodes in the network
- examples: TV, radio

□ Switched networks

- information is transmitted to a small sub-set (usually only one) of the nodes

A Taxonomy of Switched Networks



- **Circuit switching:** dedicated circuit per call/session:
 - e.g., telephone, cellular voice
- **Packet switching:** data sent thru network in discrete “chunks”
 - e.g., Internet, cellular data

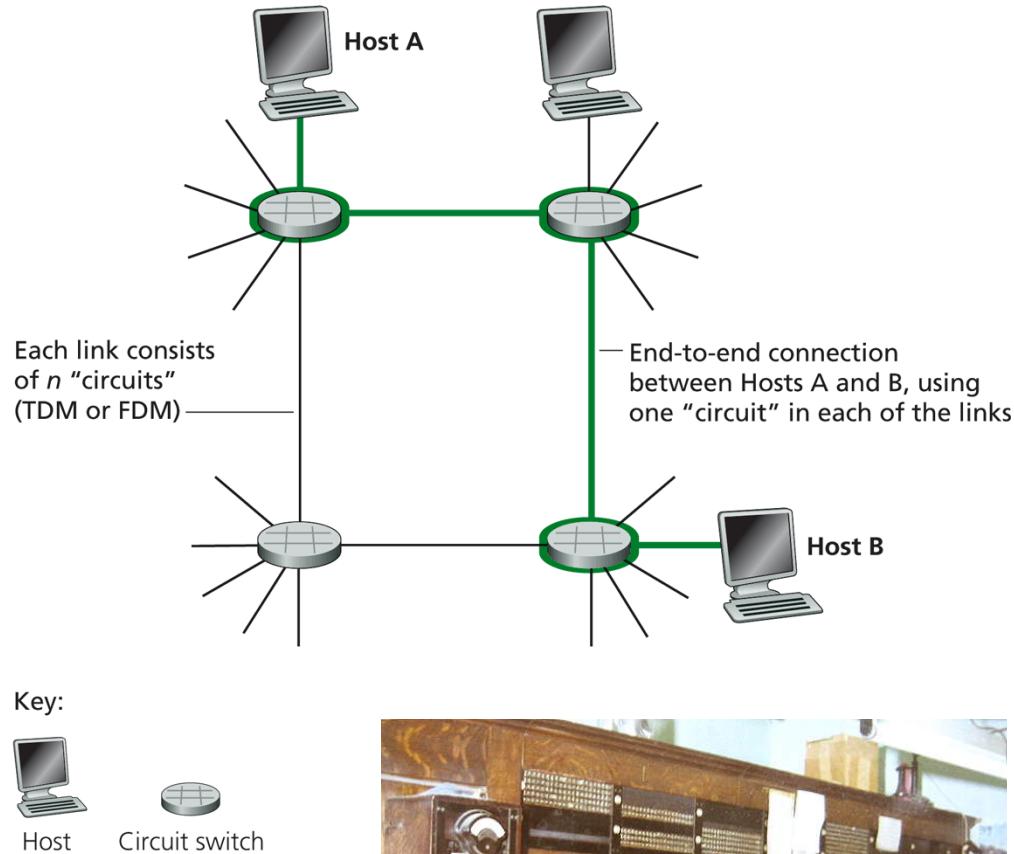
Outline

- Admin. and review
- *A taxonomy of communication networks*
 - *circuit switched networks*

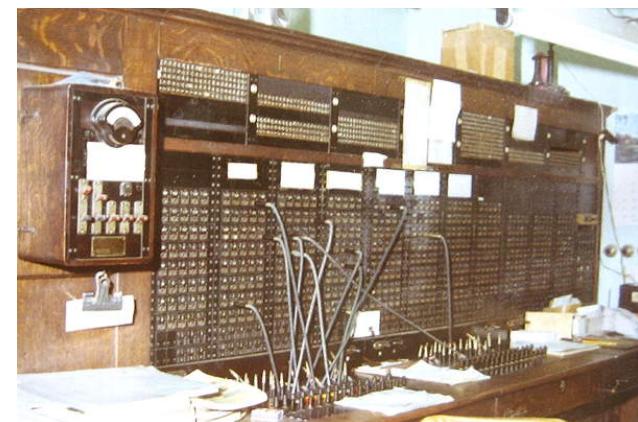
Circuit Switching

- Each link has a number of “circuits”
 - sometime we refer to a “circuit” as a channel or a line

- An end-to-end connection reserves one “circuit” at each link

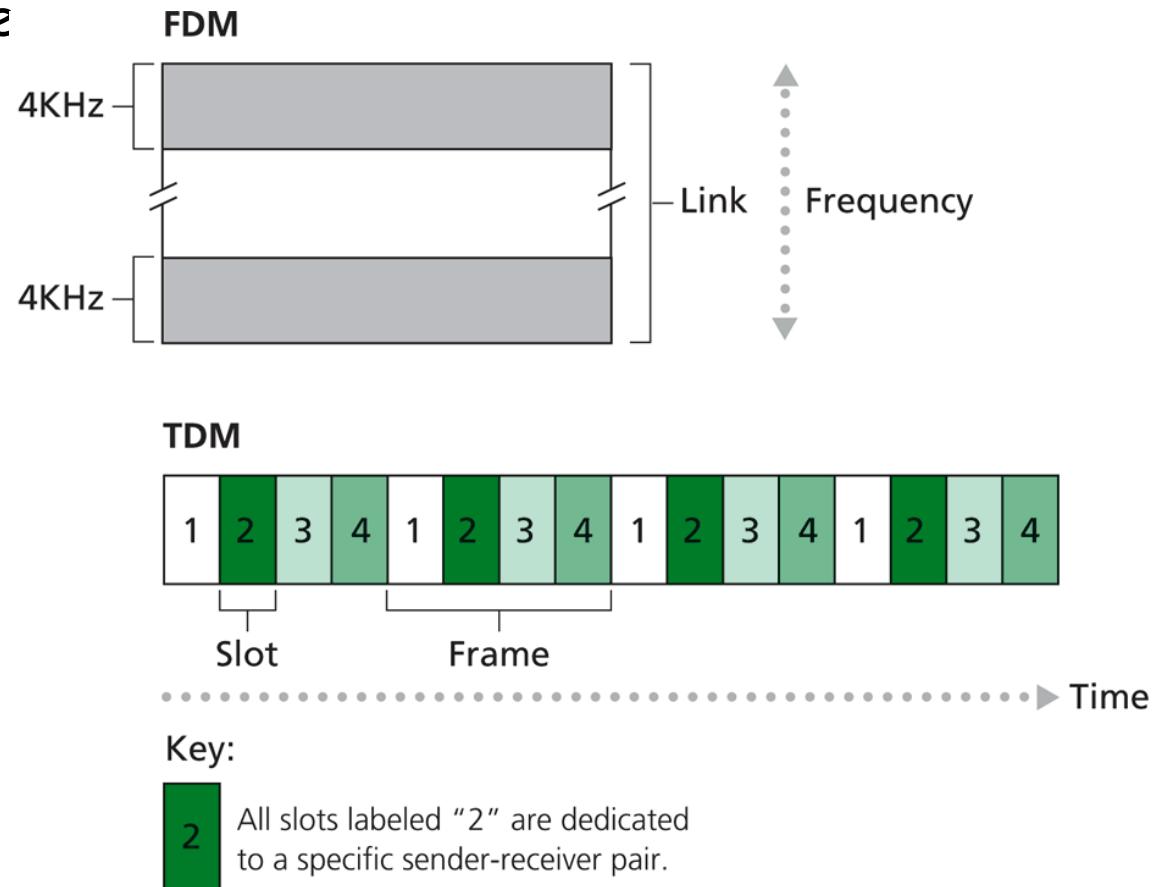


First commercial telephone switchboard was opened in 1878 to serve the 21 telephone customers in New Haven



Circuit Switching: Resources/Circuits (Frequency, Time and others)

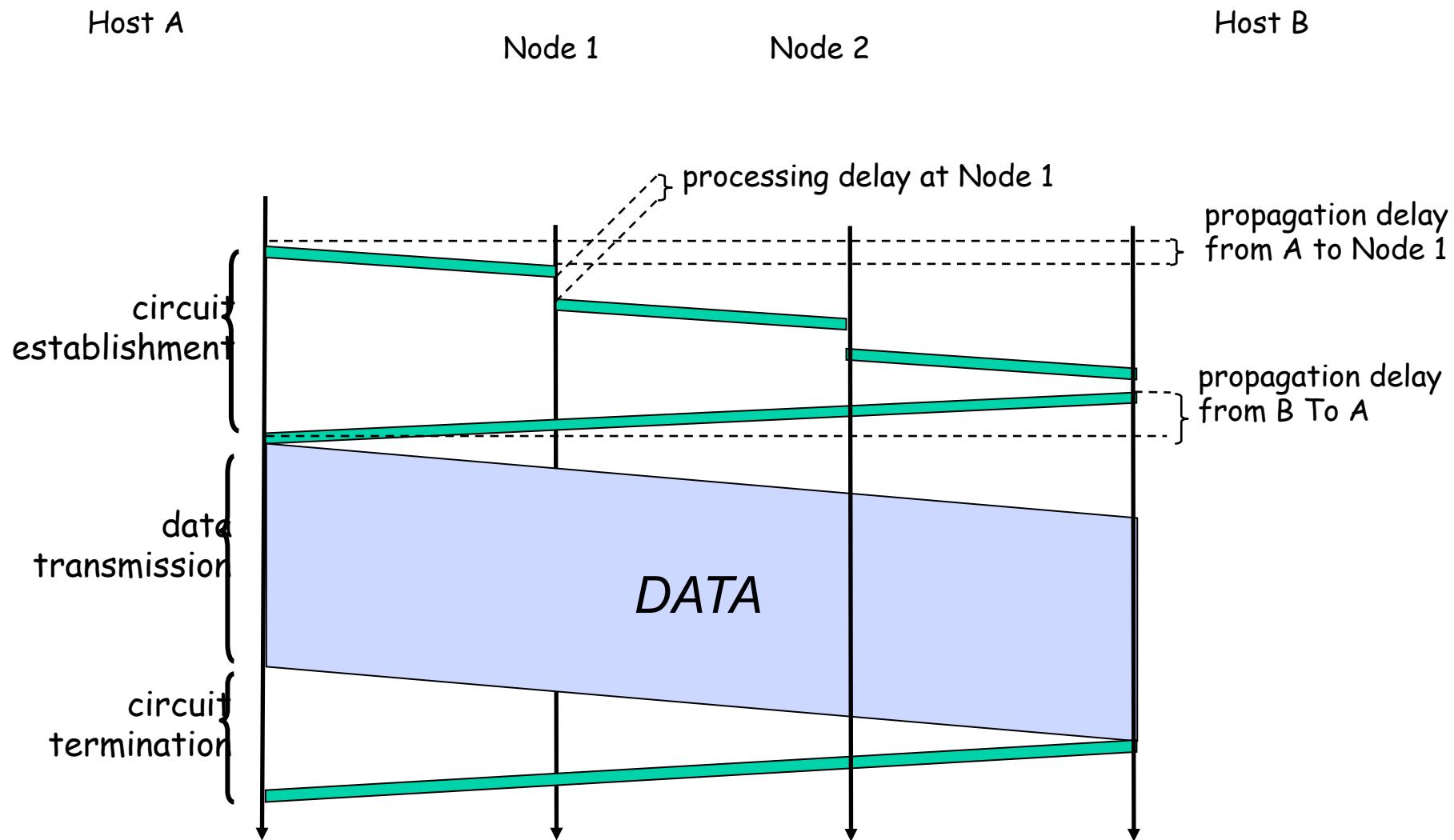
- Divide link resource into “circuits”
 - frequency division multiplexing (FDM)
 - time division multiplexing (TDM)
 - others such as code division multiplexing (CDM), color/lambda division



Circuit Switching: The Process

- Three phases
 1. circuit establishment
 2. data transfer
 3. circuit termination

Timing Diagram of Circuit Switching

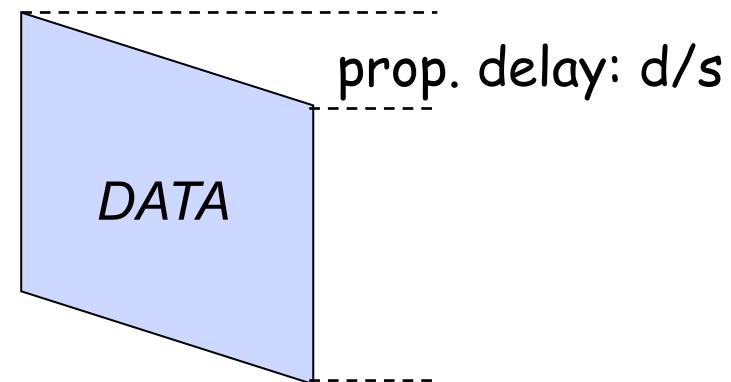


Delay Calculation in Circuit Switched Networks

- ❑ Propagation delay: delay for the first bit to go from a source to a destination

Propagation delay:

- ❑ d = length of physical link
- ❑ s = propagation speed in medium ($\sim 2 \times 10^5$ km/sec)

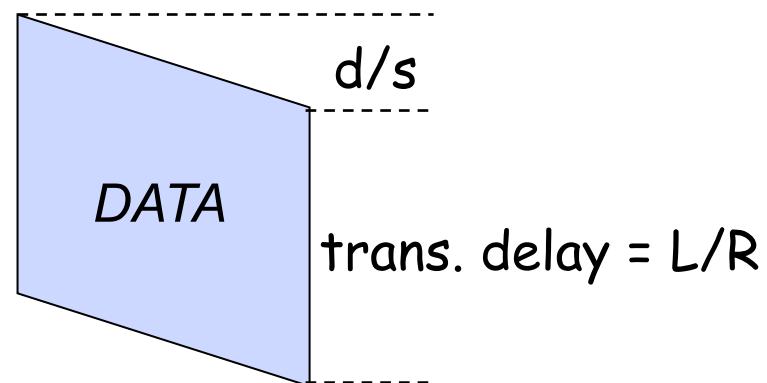


Delay Calculation in Circuit Switched Networks

- **Transmission delay**: time to pump data onto link at *line rate*

Transmission delay:

- R = reserved bandwidth (bps)
- L = message length (bits)



An Example

□ Propagation delay

- suppose the distance between A and B is 4000 km, then one-way propagation delay is:

$$\frac{4000 \text{ km}}{200,000 \text{ km/s}} = 20 \text{ ms}$$

□ Transmission delay

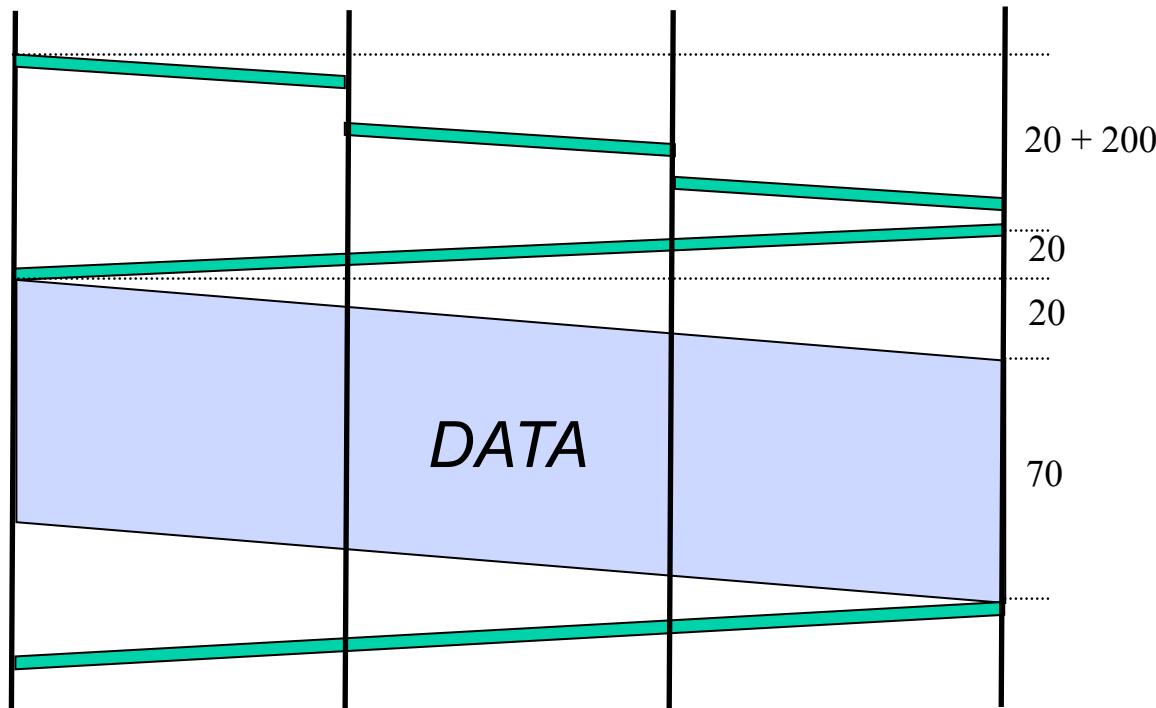
- suppose your iphone reserves a one-slot HSCSD channel
 - each HSCSD frame can transmit about 115 kbps
 - a frame is divided into 8 slots
- then the transmission delay of using one reserved slot for a message of 1 Kbits:

$$\frac{1 \text{ kbit}}{14 \text{ kbps}} \approx 70 \text{ ms}$$

An Example (cont.)

- ❑ Suppose the setup message is very small, and the total setup processing delay is 200 ms
- ❑ Then the delay to transfer a message of 1 Kbits from A to B (from the beginning until host receives last bit) is:

$$20 + 200 + 20 + 20 + 70 = 330ms$$



Outline

- Admin. and review
- *A taxonomy of communication networks*
 - circuit switched networks
 - *packet switched networks*

Packet Switching

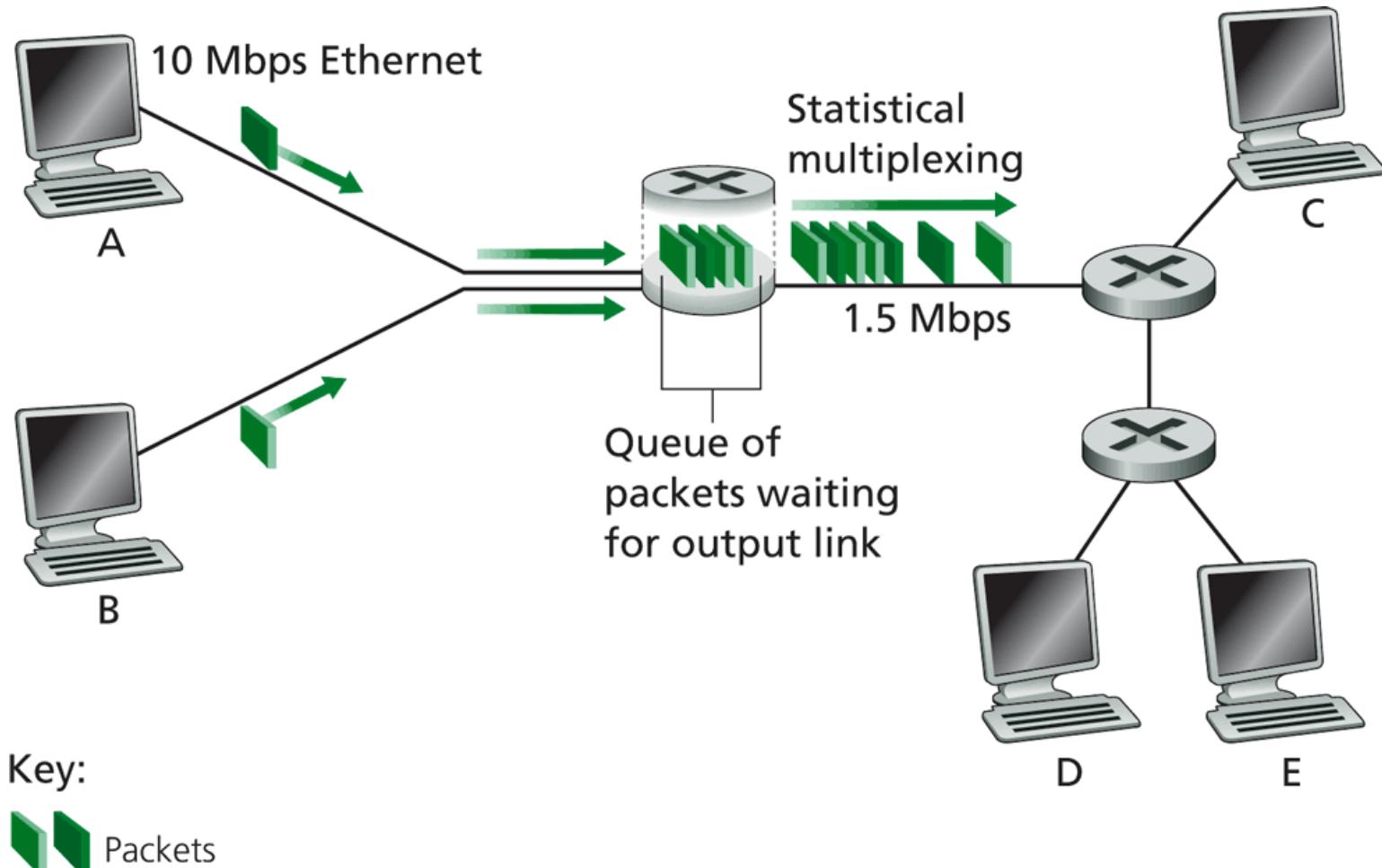
Each end-to-end data **flow** (i.e., a sender-receiver pair) divided into **packets**

- Packets have the following structure:



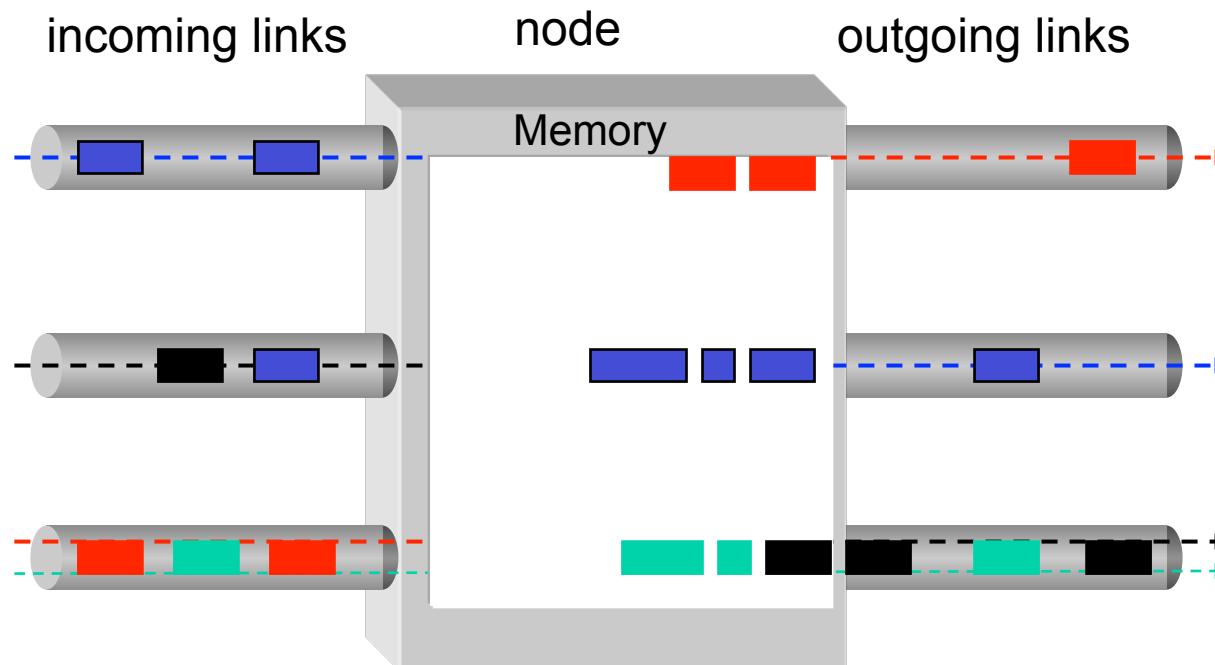
- header and trailer carry control information (e.g., destination address, check sum)
- where is the control information for circuit switching?
- At each node the entire packet is received, processed (e.g., routing), stored briefly, and then forwarded to the next node; thus packet-switched networks are also called **store-and-forward networks**. On its turn, a packet uses **full link bandwidth**

Packet Switching



Inside a Packet Switching Router

An output queueing switch



Outline

- Admin. and review
- *A taxonomy of communication networks*
 - circuit switched networks
 - packet switched networks
 - *circuit switching vs. packet switching*

Packet Switching vs. Circuit Switching

- The early history of the Internet was a heated debate between Packet Switching and Circuit Switching
 - the telephone network was the dominant network
- Need to compare packet switching with circuit switching



Circuit Switching vs. Packet Switching

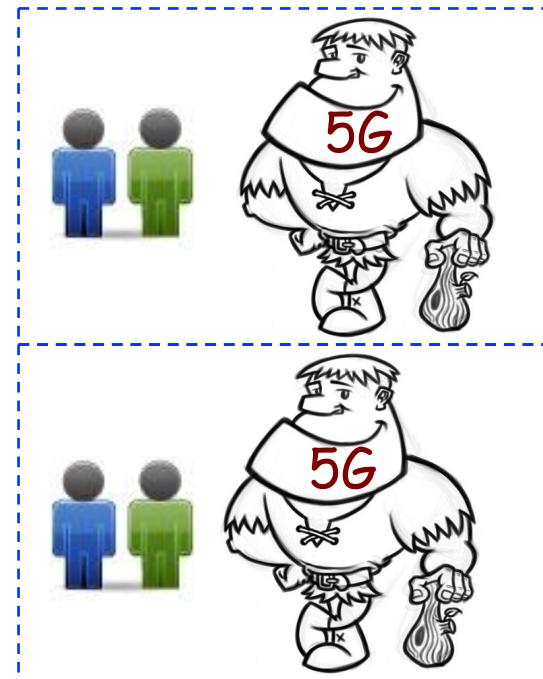
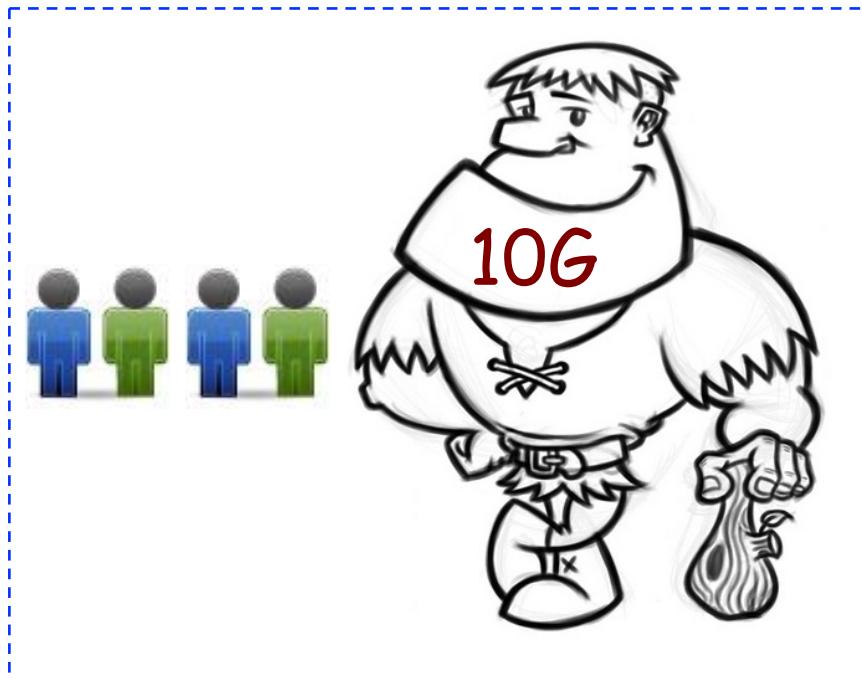
	circuit switching	packet switching
resource usage		
reservation/setup		
resource contention		
charging		
header		
fast path processing		

Circuit Switching vs. Packet Switching

	circuit switching	packet switching
resource usage	use a single partition bandwidth	use whole link bandwidth
reservation/setup	need reservation (setup delay)	no reservation
resource contention	busy signal (session loss)	congestion (long delay and packet losses)
charging	time	packet
header	no per-pkt header	per packet header
fast path processing	fast	per packet processing

Key Issue to be Settled

- A key issue: what is the efficiency of resource partition



- Tool used to analyze the issue: queueing theory
 - Some basic results of queueing theory can be quite useful in many systems settings

Outline

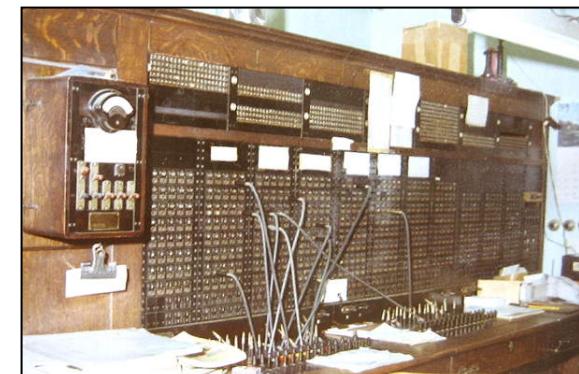
- Admin. and review
- *A taxonomy of communication networks*
 - circuit switched networks
 - packet switched networks
 - circuit switching vs. packet switching
 - *M/M queues and statistical multiplexing*

Queueing Theory

- We are not interested in extremely precise modeling, but want quantitative intuition
- Strategy:
 - model **system state**
 - if we know the fraction of time that the system spends at each state, we can get answers to many basic questions: how long does a new request need to wait before being served?
- System state changes upon events:
 - introduce **state transition** diagram
 - focus on **equilibrium**: state trend neither growing nor shrinking (key issue: how to define equilibrium)

Warm up: Analysis of Circuit-Switching Blocking (Busy) Time

- Assume a link has only a finite number of N circuits
- Objective: compute the percentage of time that a new session (call) is blocked
- Analogy in a more typical real-life scenario?
- Key parameters?



Analysis of Circuit-Switching

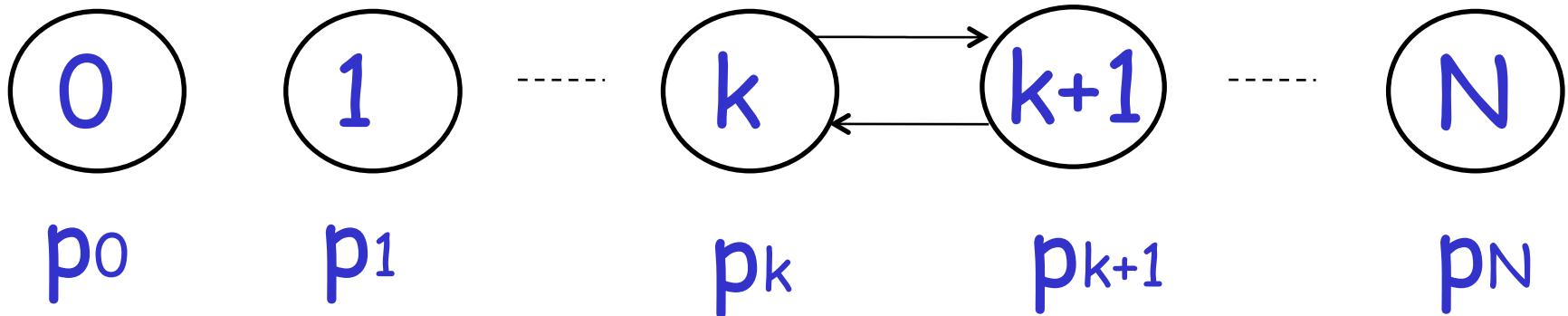
Blocking (Busy) Time

- Consider a simple arrival pattern
 - client requests arrive at a rate of λ (lambda/second)
 - service rate: each call takes on average $1/\mu$ second
- Arrival and service patterns: memoryless (Markovian)
 - During a small interval Δt , the number of expected new arrivals is: $\lambda \Delta t$
 - During a small interval Δt , the chance (fraction) of a current call finishes is: $\mu \Delta t$
- This model is also called an M/M/N model

Analysis of Circuit-Switching

Blocking (Busy) Time: State

system state: # of busy lines



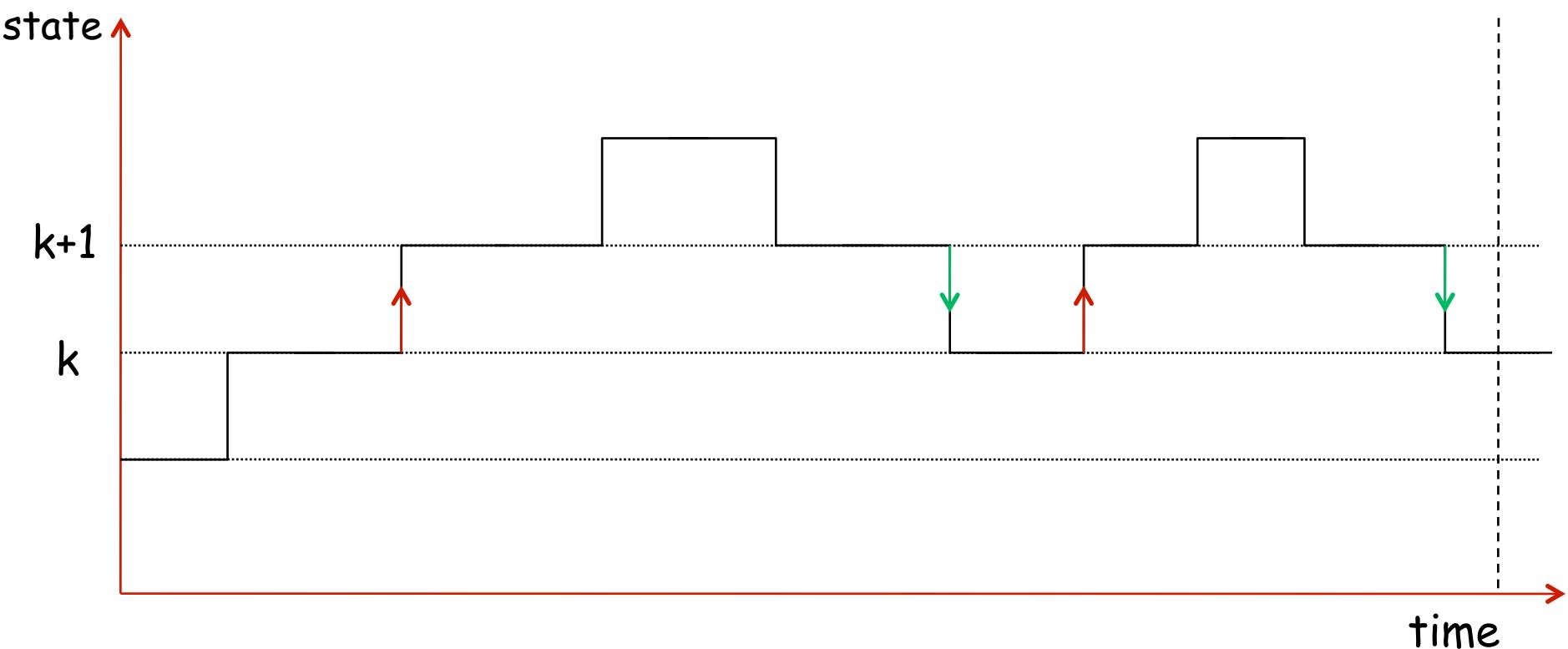
Q: How to characterize equilibrium?

Equilibrium = Time Reversibility [Frank Kelly]

- Statistically cannot distinguish

$$\# f_{k \rightarrow k+1}, \quad \# f_{k+1 \rightarrow k}$$

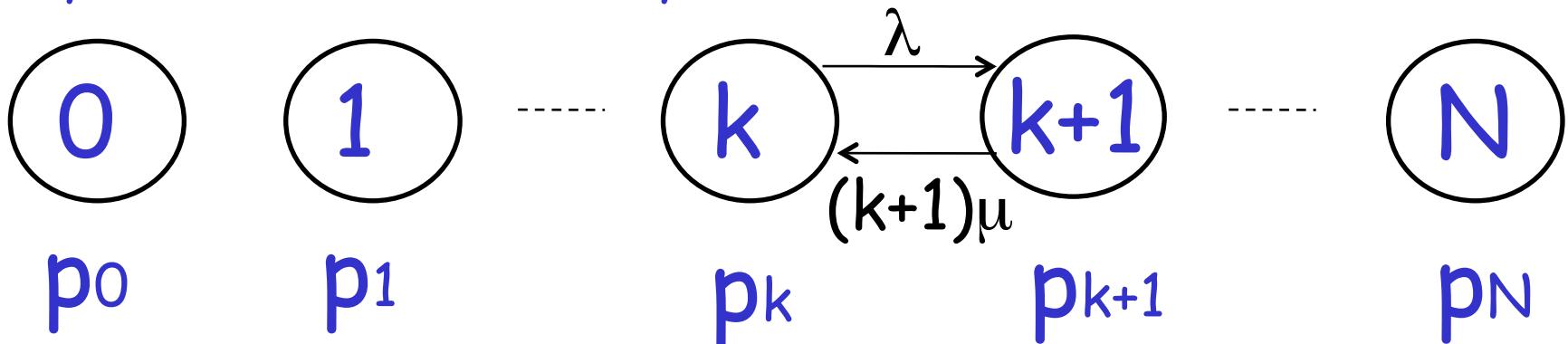
$$\# b_{k \rightarrow k+1}, \quad \# b_{k+1 \rightarrow k}$$



Analysis of Circuit-Switching

Blocking (Busy) Time: Sketch

system state: # of busy lines



at equilibrium (time reversibility) in one unit time:
#(transitions $k \rightarrow k+1$) = #(transitions $k+1 \rightarrow k$)

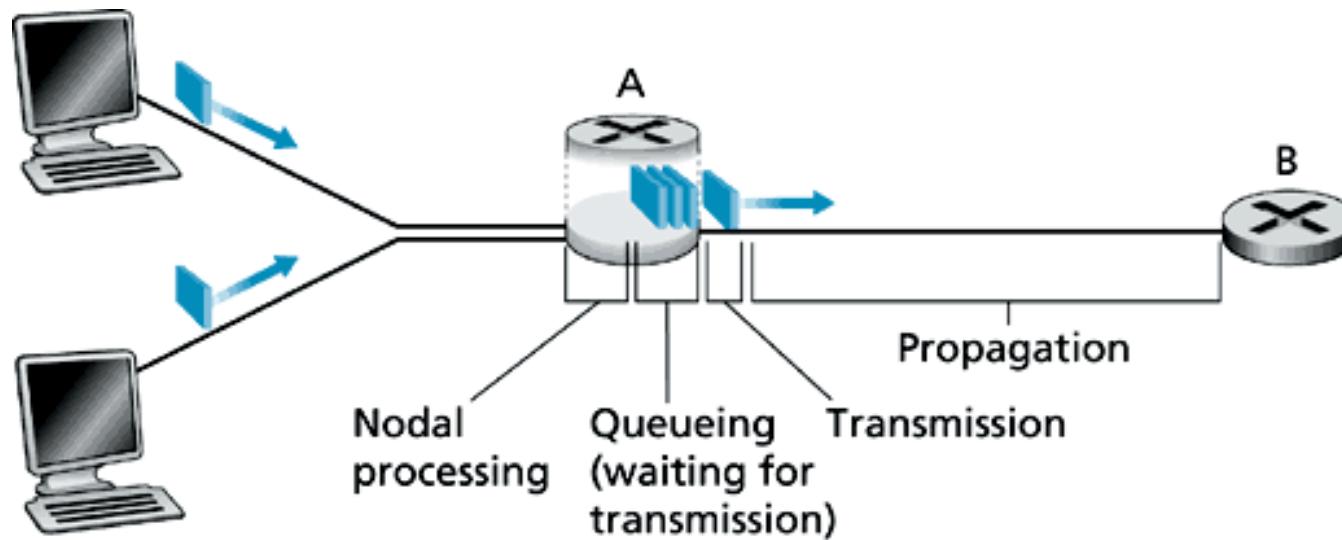
$$p_k \lambda = p_{k+1} (k+1) \mu$$

$$p_{k+1} = \frac{1}{k+1} \frac{\lambda}{\mu} p_k = \frac{1}{(k+1)!} \left(\frac{\lambda}{\mu}\right)^{k+1} p_0$$

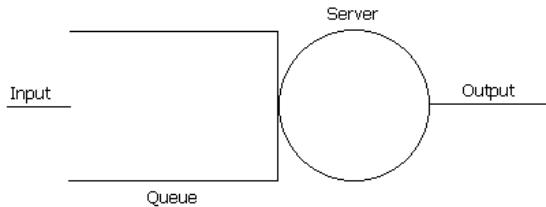
$$p_0 = \frac{1}{1 + \frac{1}{1!} \frac{\lambda}{\mu} + \frac{1}{2!} \left(\frac{\lambda}{\mu}\right)^2 + \dots + \frac{1}{N!} \left(\frac{\lambda}{\mu}\right)^N}$$

Queueing Analysis: Packet Switching Delay

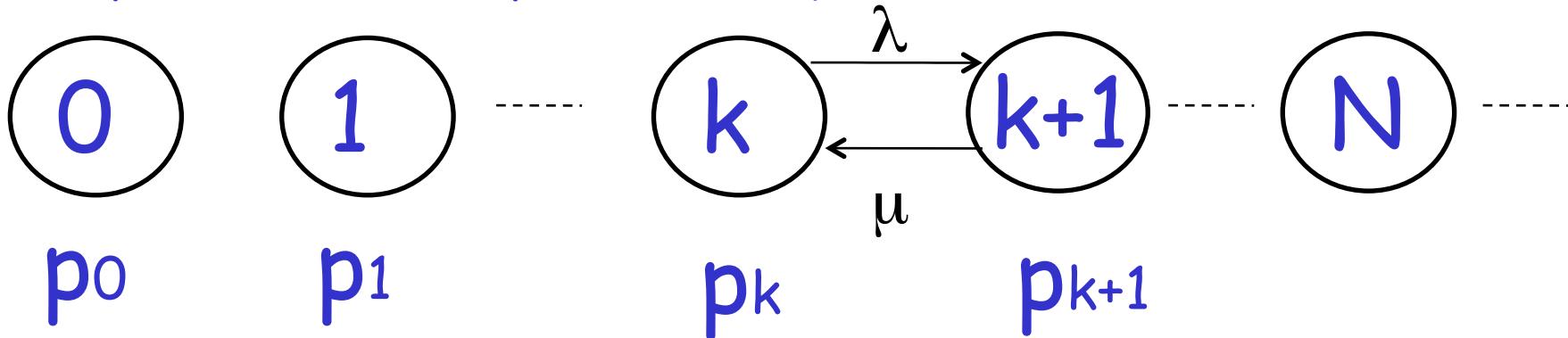
- Four types of delay at each hop
 - nodal processing delay: check errors & routing
 - queueing: time waiting for its turn at output link
 - transmission delay: time to pump packet onto a link at link speed
 - propagation delay: router to router propagation
- The focus is on **queueing and transmission delay**



Packet Switching Delay



system state: #packets in queue



at equilibrium (time reversibility) in one unit time:

$$\#(\text{transitions } k \rightarrow k+1) = \#(\text{transitions } k+1 \rightarrow k)$$

Statistical Multiplexing; Layered Network Architecture; End-to-end Arguments

Y. Richard Yang

<http://zoo.cs.yale.edu/classes/cs433/>

01/25/2016

Outline

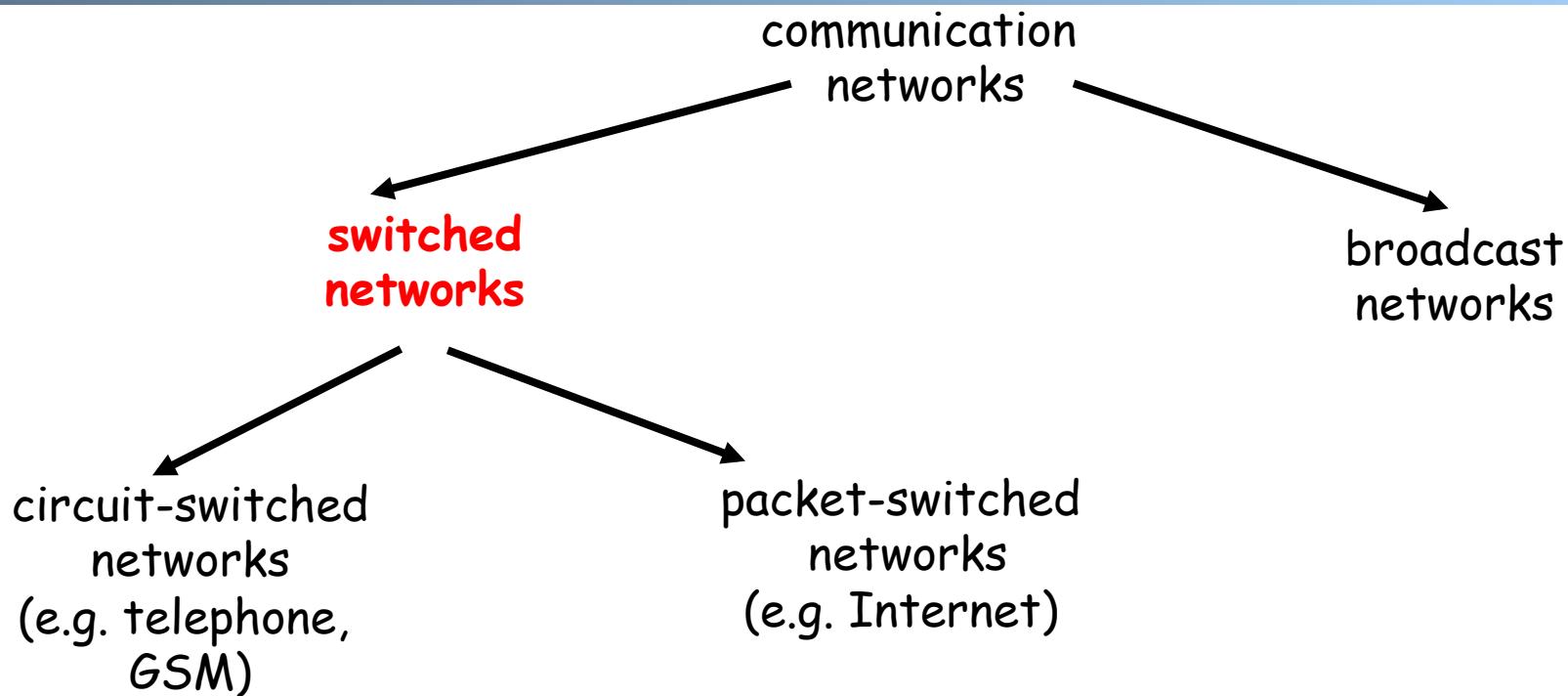
- Admin. and recap
- A taxonomy of communication networks
- Layered network architecture

Admin.

- Readings from the textbook and additional suggested readings
 - All are highly recommended
 - Some are marked as required

- Assignment one is linked on the schedule page
 - Due Feb. 1, 2016, in class
 - if you type it in, you can upload to classes V2

Recap: A Taxonomy of Comm. Networks



- **circuit switching:** dedicated circuit per call/session:
 - e.g., telephone, cellular voice
- **packet switching:** data sent thru network in discrete “chunks”
 - e.g., Internet, cellular data

Recap: Circuit Switching vs. Packet Switching

	circuit switching	packet switching
resource usage	use a single partition bandwidth	use whole link bandwidth
reservation/setup	need reservation (setup delay)	no reservation
resource contention	busy signal (session loss)	congestion (long delay and packet losses)
charging	time	packet
header	no per-pkt header	per packet header
fast path processing	fast	per packet processing

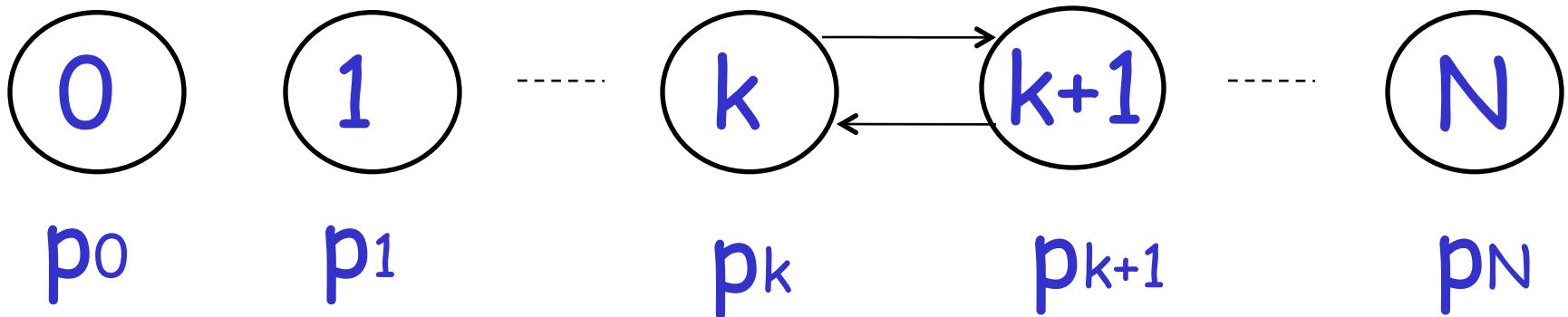
Recap: Queueing Theory

- We are not interested in extremely precise modeling, but want quantitative intuition
- Strategy:
 - model **system state**
 - if we know the fraction of time the system spends at each state, we can get answers to some basic questions: how long does a new request need to wait before being served?
- System state changes upon events:
 - introduce **state transition** diagram
 - focus on **equilibrium**: state trend neither growing nor shrinking
 - One intuitive way to define equilibrium is time reversibility

Recap: Circuit-Switching

System State

system state: # of busy lines

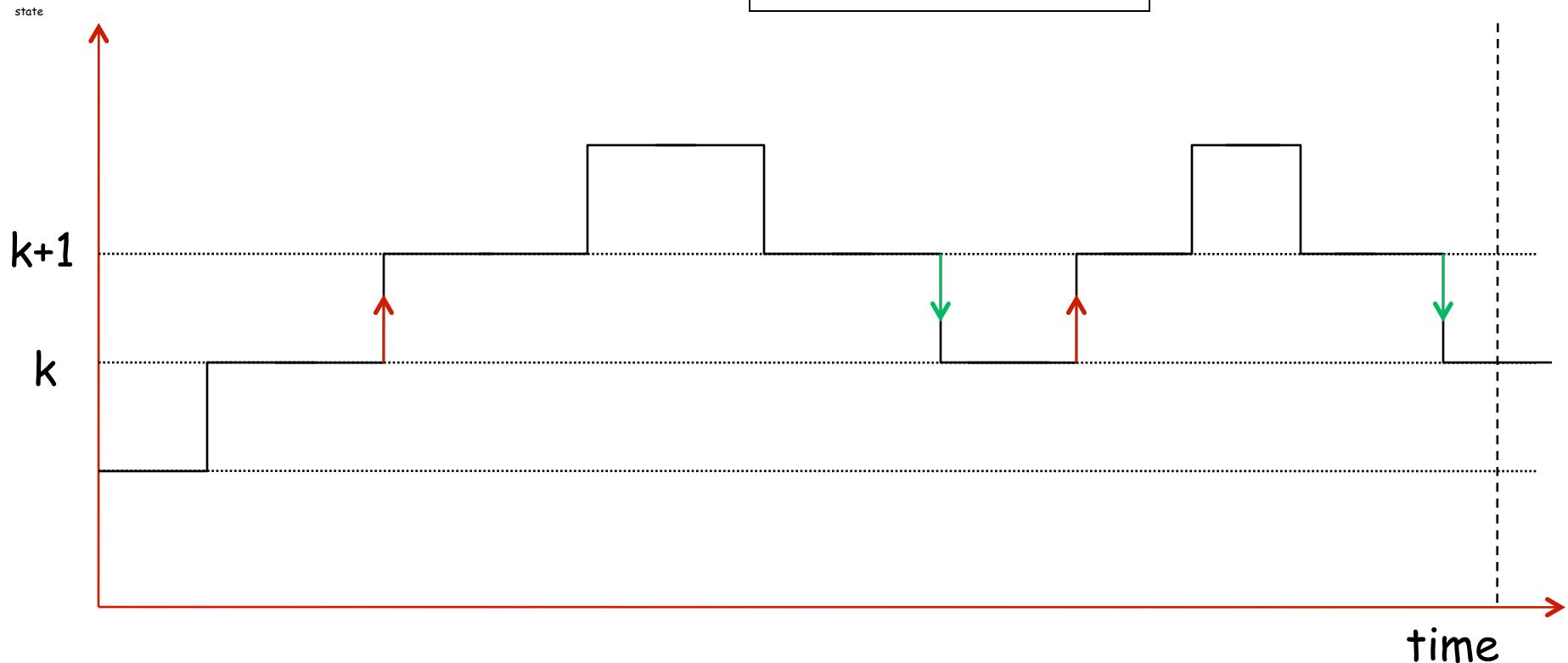


Equilibrium = Time Reversibility

□ Cannot distinguish

$$\# f_{k \rightarrow k+1}, \quad \# f_{k+1 \rightarrow k}$$

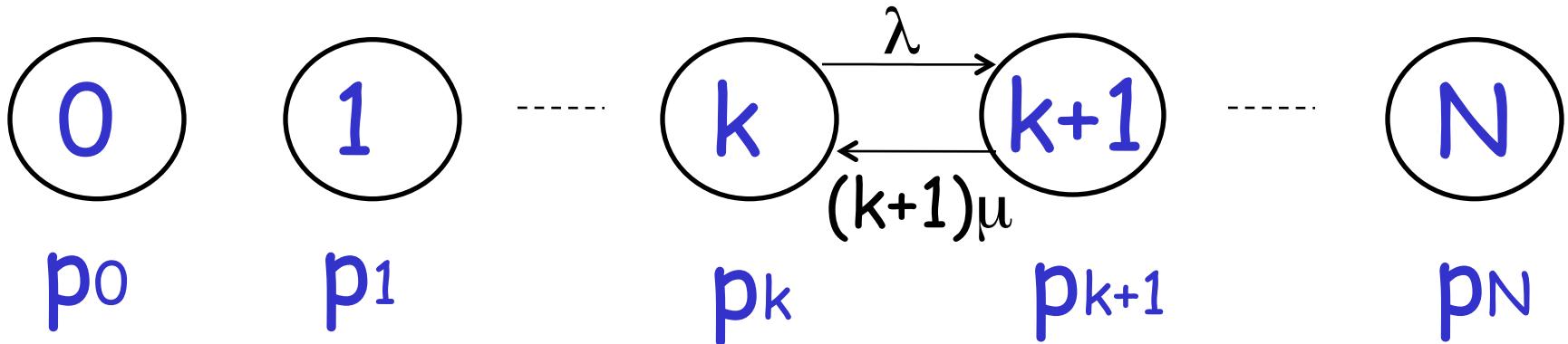
$$\# b_{k \rightarrow k+1}, \quad \# b_{k+1 \rightarrow k}$$



Recap: Circuit-Switching

Blocking (Busy) Time: Sketch

system state: # of busy lines



at equilibrium (time reversibility) in one unit time:

$$\#(\text{transitions } k \rightarrow k+1) = \#(\text{transitions } k+1 \rightarrow k)$$

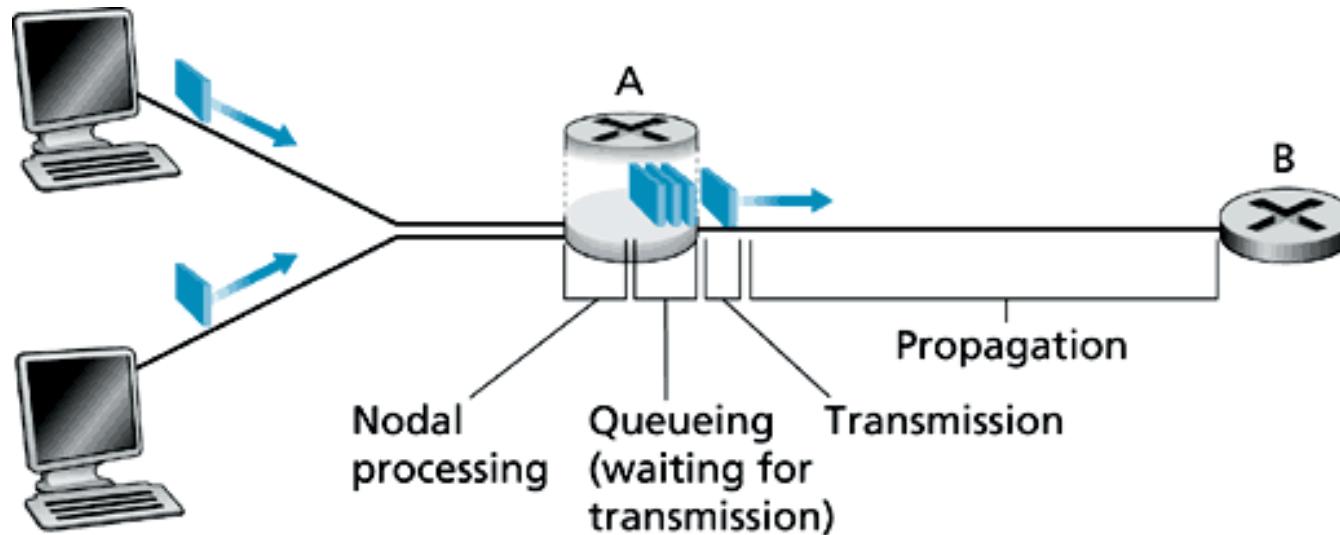
$$p_k \lambda = p_{k+1} (k+1) \mu$$

$$p_{k+1} = \frac{1}{k+1} \frac{\lambda}{\mu} p_k = \frac{1}{(k+1)!} \left(\frac{\lambda}{\mu}\right)^{k+1} p_0$$

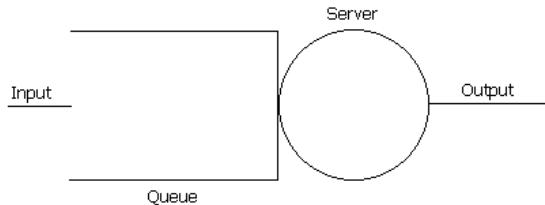
$$p_0 = \frac{1}{1 + \frac{1}{1!} \frac{\lambda}{\mu} + \frac{1}{2!} \left(\frac{\lambda}{\mu}\right)^2 + \dots + \frac{1}{N!} \left(\frac{\lambda}{\mu}\right)^N}$$

Recap: Packet Switching Delay

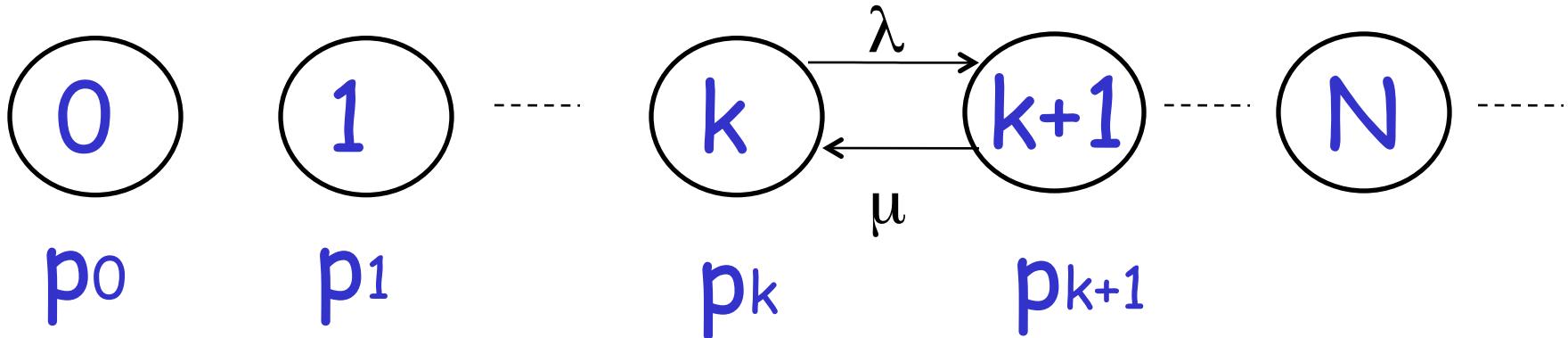
- Four types of delay at each hop
 - nodal processing delay: check errors & routing
 - queueing: time waiting for its turn at output link
 - transmission delay: time to pump packet onto a link at link speed
 - propagation delay: router to router propagation
- The focus is on **queueing and transmission delay**



Packet Switching Delay



system state: #packets in queue



at equilibrium (time reversibility) in one unit time:

$$\#(\text{transitions } k \rightarrow k+1) = \#(\text{transitions } k+1 \rightarrow k)$$

$$p_k \lambda = p_{k+1} \mu$$

$$p_{k+1} = \frac{\lambda}{\mu} p_k = \left(\frac{\lambda}{\mu}\right)^{k+1} p_0 = \rho^{k+1} p_0$$

$$p_0 = 1 - \rho$$

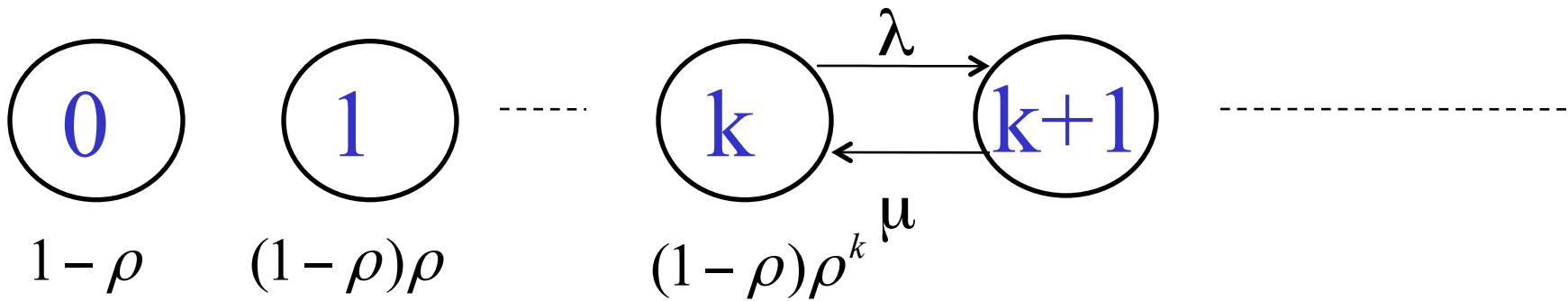
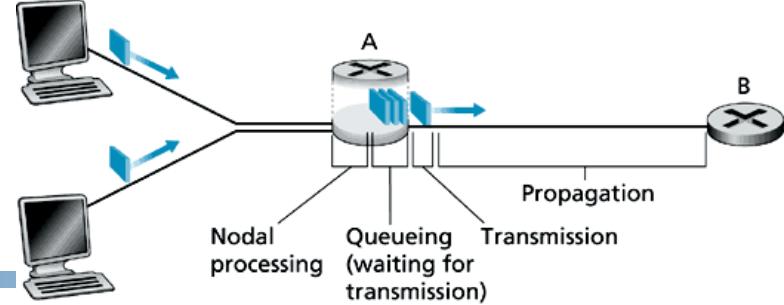
$$\rho = \frac{\lambda}{\mu}$$

Example

- Assume requests (packets) come in at a rate of one request per 30 ms
- Each request (packet) takes on average 20 ms

- What is the fraction of time that the system is empty?
- What is the chance that a packet newly arrived needs to wait for 3 early packets?

Analysis of Delay (cont')



- Average queueing delay:
- Transmission delay:
- Queueing + transmission:

Delay

$$\rho = \frac{\lambda}{\mu}$$

$$S = \frac{1}{\mu}$$

average queueing delay: $w = S \frac{\rho}{1 - \rho}$

$$queueing + trans = S \frac{\rho}{1 - \rho} + S = S \frac{1}{1 - \rho}$$

For a demo of M/M/1, see:

http://www.dcs.ed.ac.uk/home/jeh/Simjava/queueing/mm1_q/mm1_q.html

Queueing Delay as a Function of Utilization

Assume:

R = link bandwidth (bps)

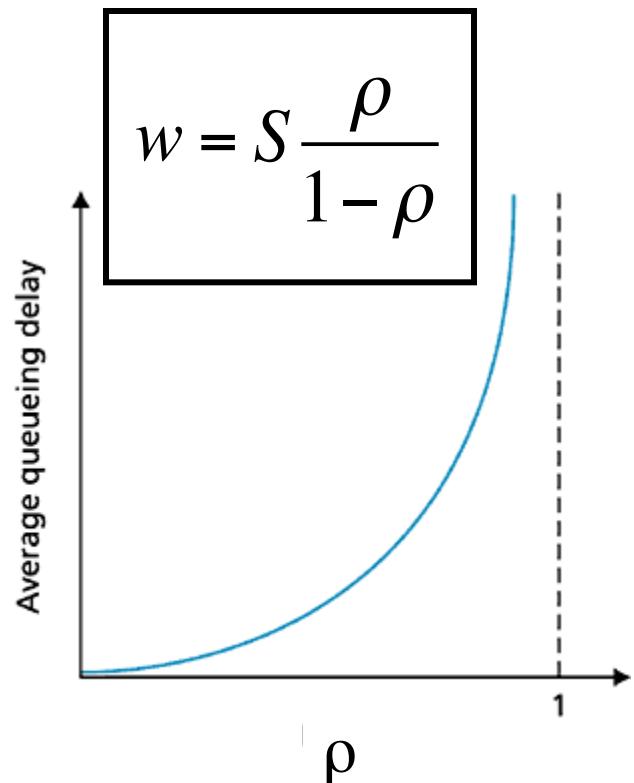
L = packet length (bits)

$S = L / R$

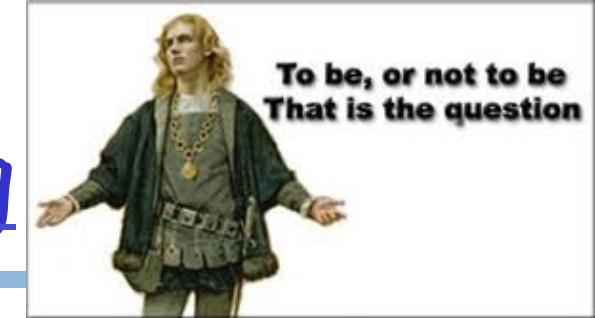
a = average packet arrival rate (pkt/sec)

$$\text{utilization: } \rho = \frac{a}{1/S} = aS$$

$$w = S \frac{\rho}{1 - \rho}$$



- $\rho \sim 0$: average queueing delay small
- $\rho \rightarrow 1$: delay becomes large
- $\rho > 1$: more “work” arriving than can be serviced, average delay infinite !



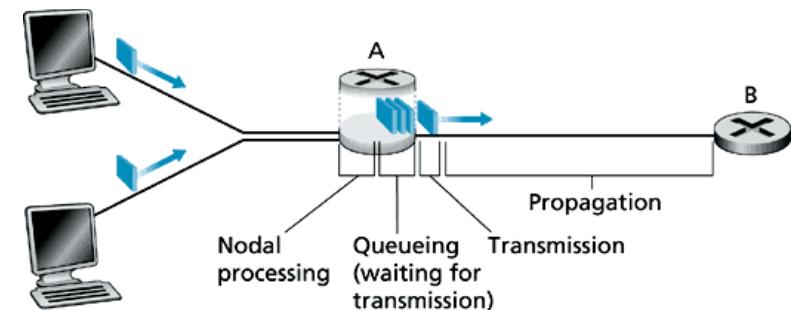
Statistical Multiplexing

A simple model to compare bandwidth efficiency of

- reservation/dedication (aka circuit-switching) and
- no reservation (aka packet switching)

setup

- a single bottleneck link
- n flows; each flow has an arrival rate of a/n



- no reservation: all arrivals into the single link, the queueing delay + transmission delay:

$$S \frac{1}{1 - \rho}$$

- reservation: each flow uses its own reserved (sub)link with rate u/n , the queueing delay + transmission delay:

$$\textcircled{n} S \frac{1}{1 - \rho}$$

Summary:

Packet Switching vs. Circuit Switching

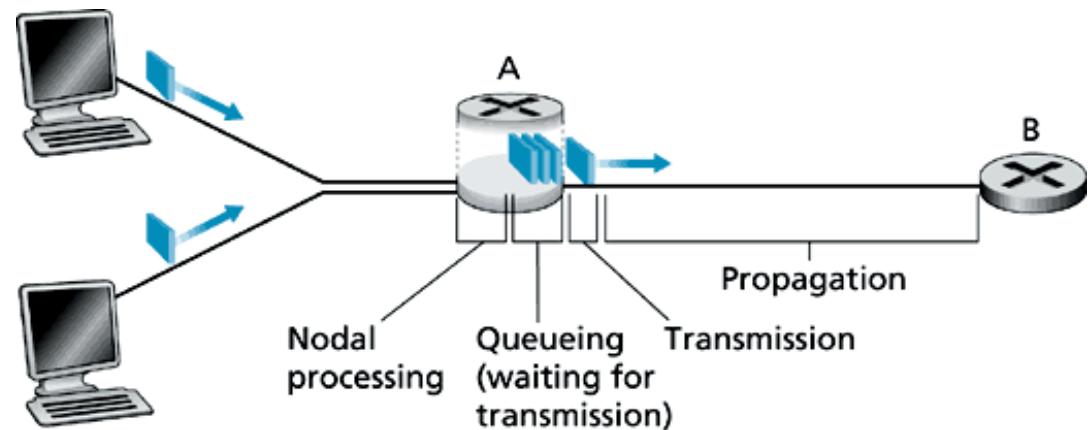
- Advantages of packet switching over circuit switching
 - most important advantage of packet-switching over circuit switching is **statistical multiplexing**, and therefore more efficient bandwidth usage
- Disadvantages of packet switching
 - **potential congestion:** packet delay and high loss
 - protocols needed for reliable data transfer, congestion control
 - it is possible to guarantee quality of service (QoS) in packet-switched networks and still gain statistical multiplexing, but it adds much complexity
 - **packet header overhead**
 - **per packet processing overhead**

Outline

- Admin. and recap
- *A taxonomy of communication networks*
 - circuit switched networks
 - packet switched networks
 - circuit switching vs. packet switching
 - *datagram and virtual circuit packet switched networks*

A Taxonomy of Packet-Switched Networks According to Routing

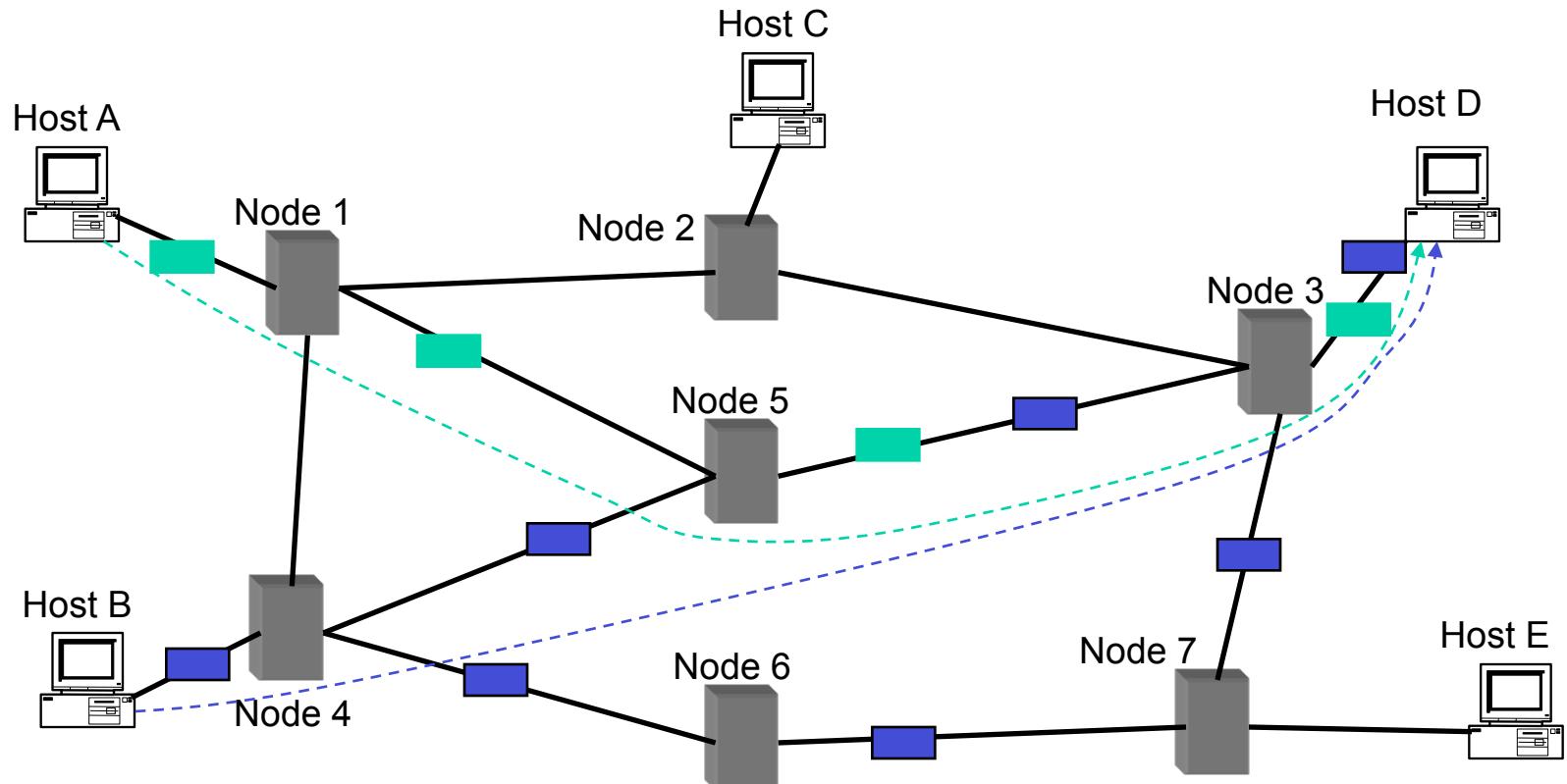
- Two types of packet switching
 - datagram network
 - each packet of a flow is switched **independently**
 - virtual circuit network:
 - all packets from one flow are sent along a **pre-established path** (= virtual circuit)



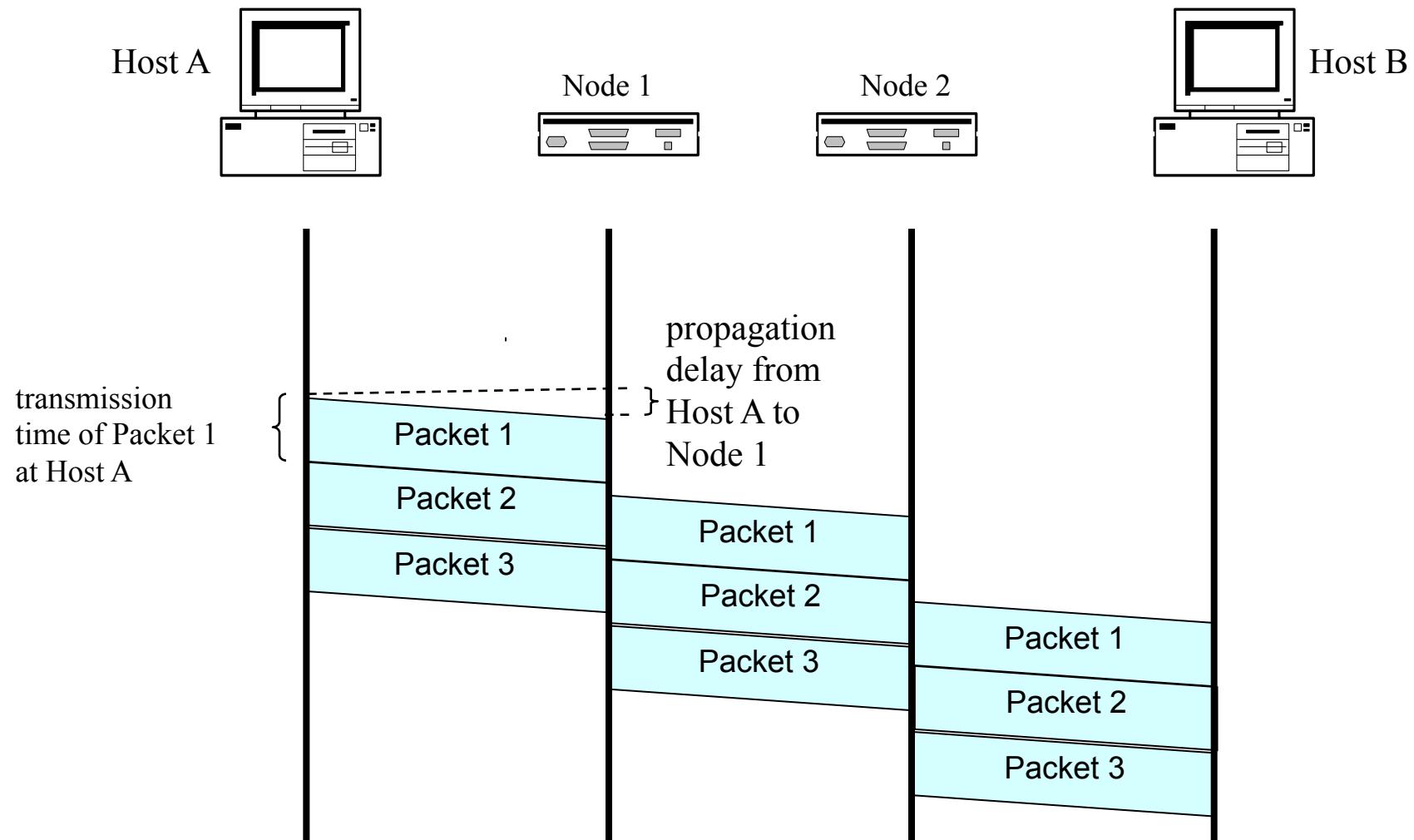
Datagram Packet Switching

- Commonly when we say packet switching we mean datagram switching
- Example: IP networks
- Each packet is independently switched
 - each packet header contains *complete destination address*
 - receiving a packet, a router looks at the packet's destination address and *searches* its current routing table to determine the possible next hops, and pick one
- Analogy: postal mail system

Datagram Packet Switching



Timing Diagram of Datagram Switching

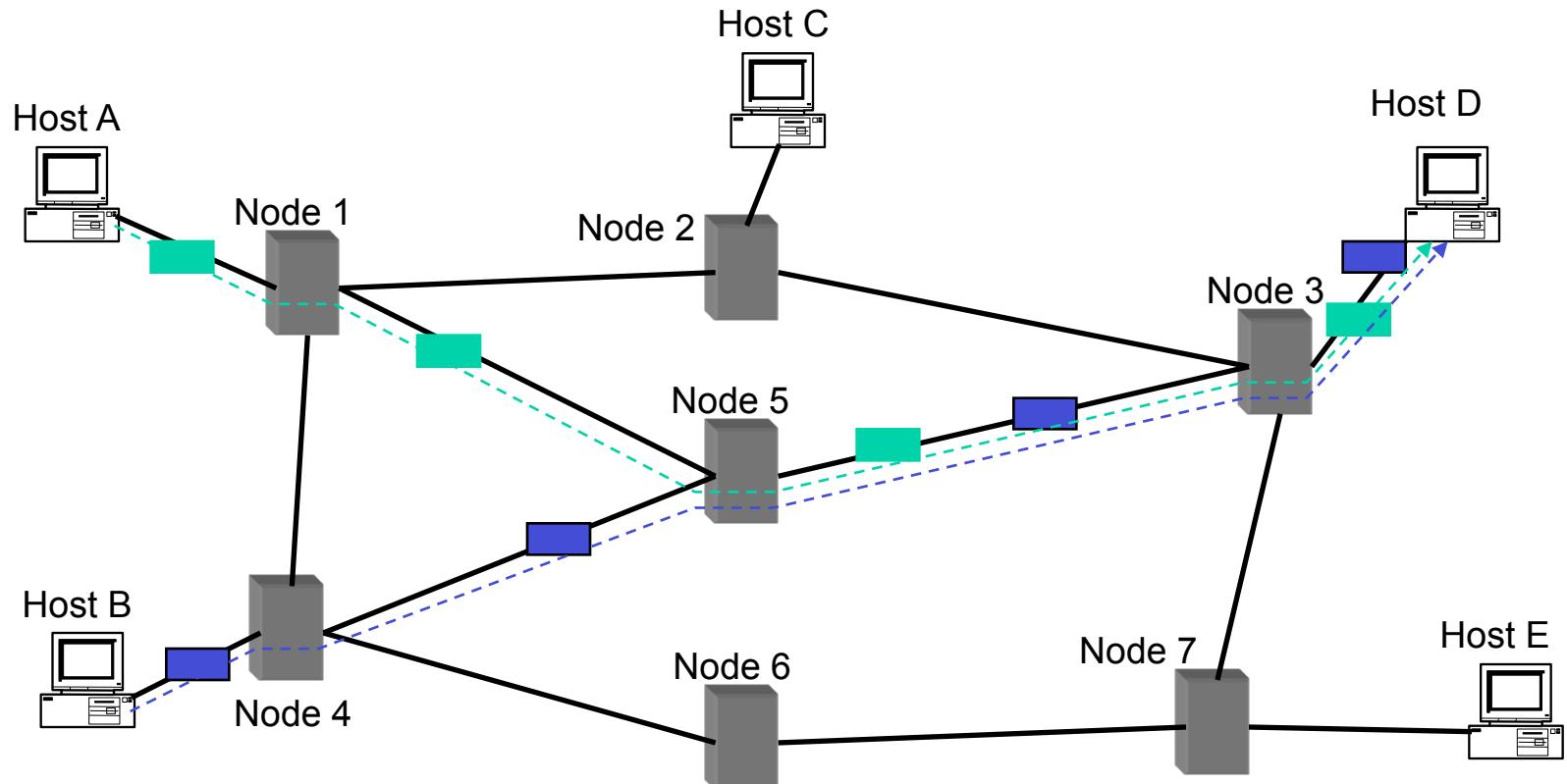


Virtual-Circuit Packet Switching

- Example: Multiple Label Packet Switching (MPLS) in IP networks
- Hybrid of circuit switching and datagram switching
 - fixed path determined at *virtual circuit setup time*, remains fixed thru flow
 - Two implementations:
 1. each packet carries a short **tag** (virtual-circuit (VC) #); tag determines next hop
 2. Match on packet attributes to find entries

Incoming VC#	Outgoing Interface	QoS
12	2	
16	3	
20	3	
...		

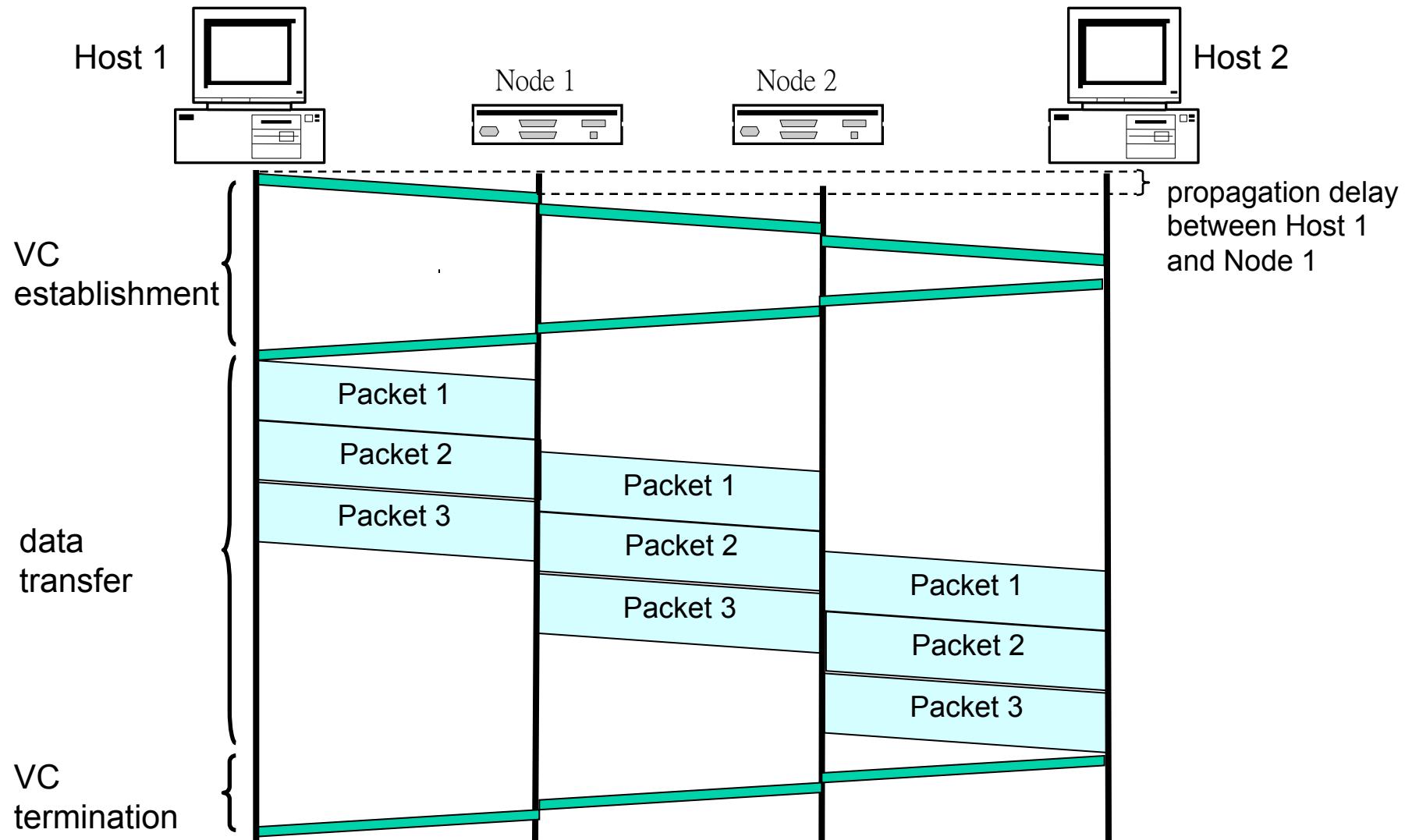
Virtual-Circuit Switching



Virtual-Circuit Packet Switching

- Three phases
 1. VC establishment
 2. Data transfer
 3. VC disconnect

Timing Diagram of Virtual-Circuit Switching

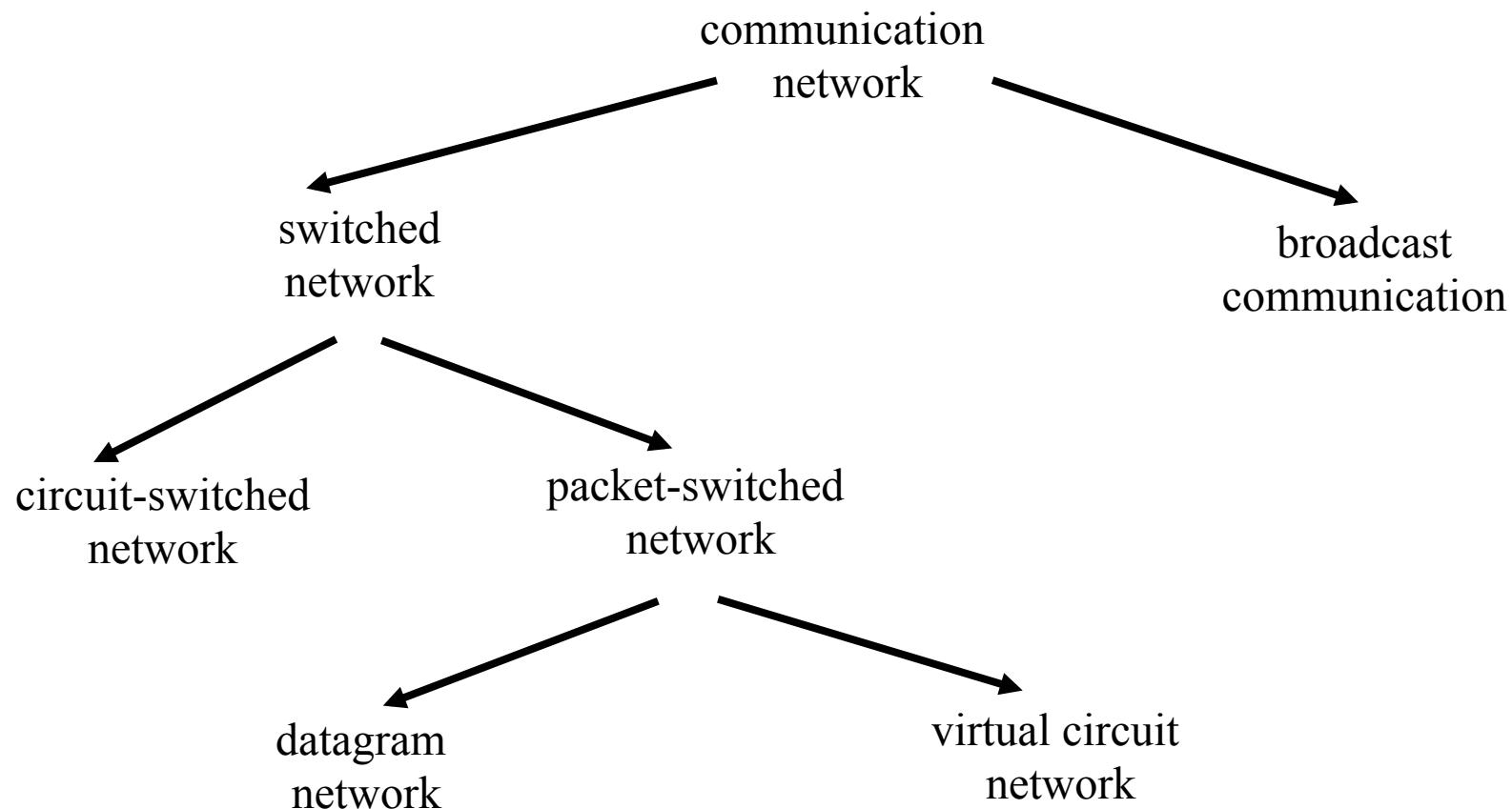


Discussion: Datagram Switching vs. Virtual Circuit Switching

- What are the benefits of datagram switching over virtual circuit switching?

- What are the benefits of virtual circuit switching over datagram switching?

Summary of the Taxonomy of Communication Networks



Summary of Progress

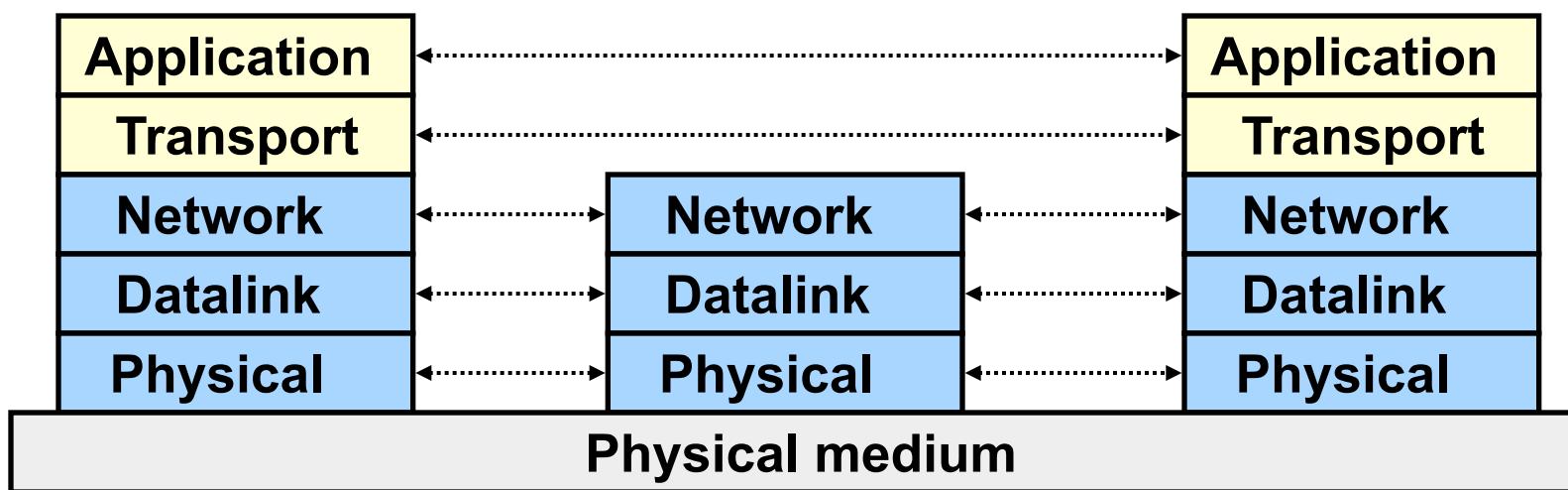
- We have seen the hardware infrastructure, the basic communication scheme, a next key question is how to develop the software system.

Outline

- Admin. and recap
- A taxonomy of communication networks
- Layered network architecture
 - *what is layering?*
 - why layering?
 - how to determine the layers?
 - ISO/OSI layering and Internet layering

What is Layering?

- A technique to organize a networked system into a **succession** of logically distinct entities, such that the service provided by one entity is **solely** based on the service provided by the previous (lower level) entity.



Outline

- Admin. and recap
- A taxonomy of communication networks
- Layered network architecture
 - what is layering?
 - *why layering?*

Why Layering?

Networks are complex !

- many “pieces”:
 - hardware
 - hosts
 - routers
 - links of various media
 - software
 - applications
 - protocols

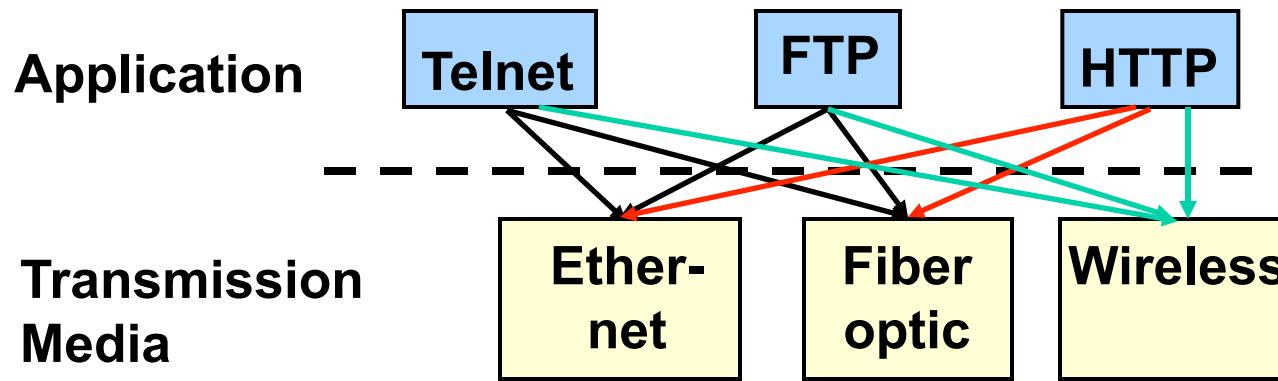
Dealing with complex systems:
explicit structure allows
identification of the relationship
among a complex system's pieces

- layered reference model for discussion

Modularization eases maintenance,
updating of system:

- change of implementation of a layer's service transparent to rest of system, e.g., changes in routing protocol doesn't affect rest of system

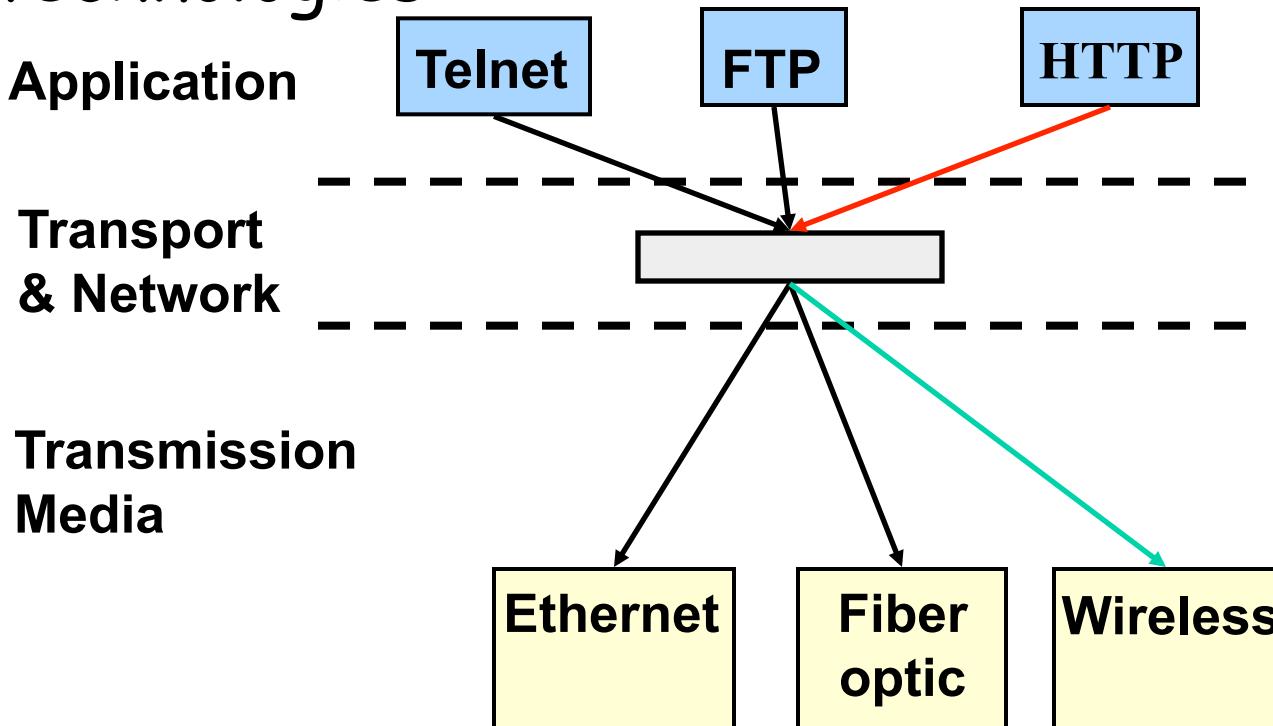
An Example: No Layering



- ❑ No layering: each new application has to be **re-implemented** for every network technology !

An Example: Benefit of Layering

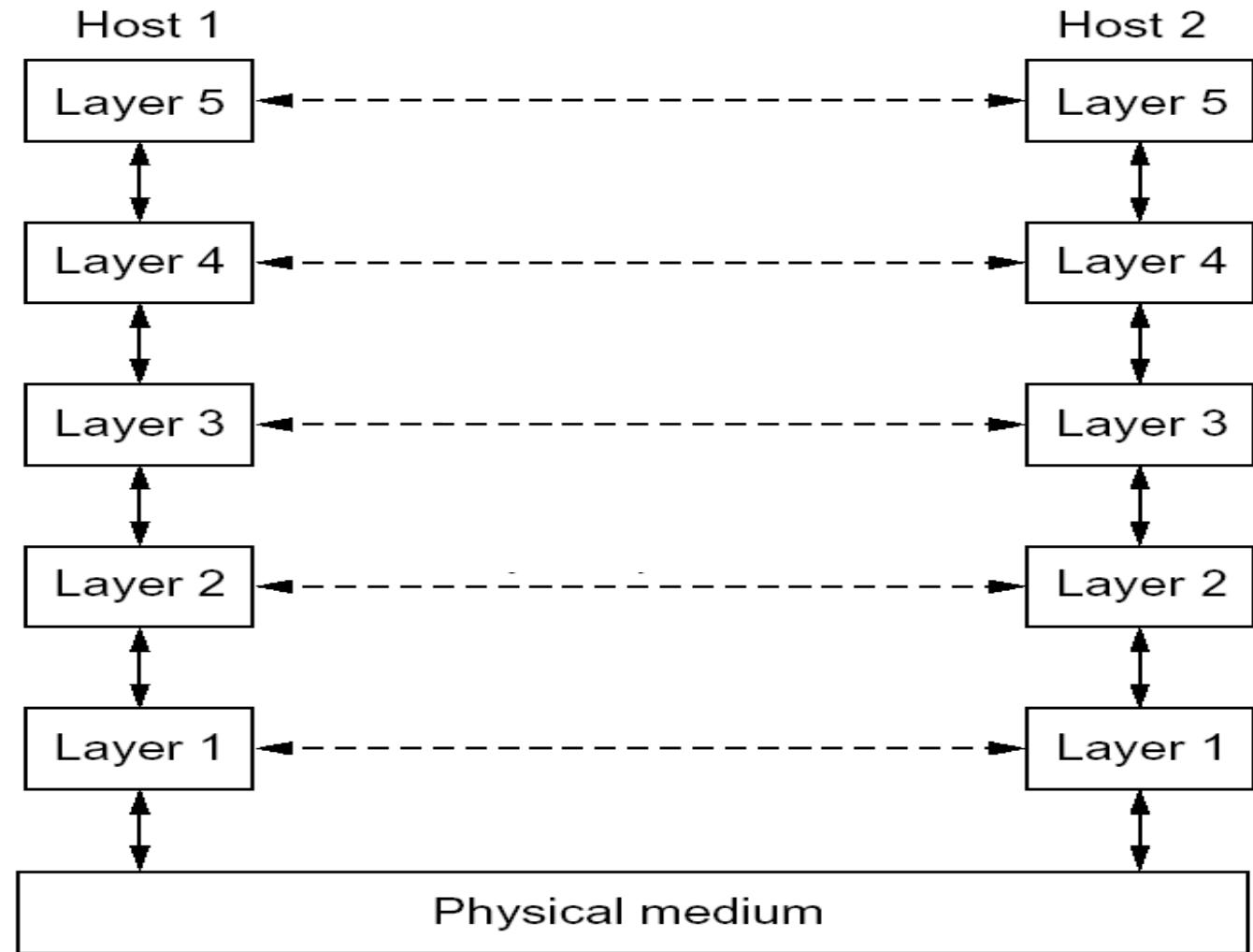
- Introducing an intermediate layer provides a **common abstraction** for various network technologies



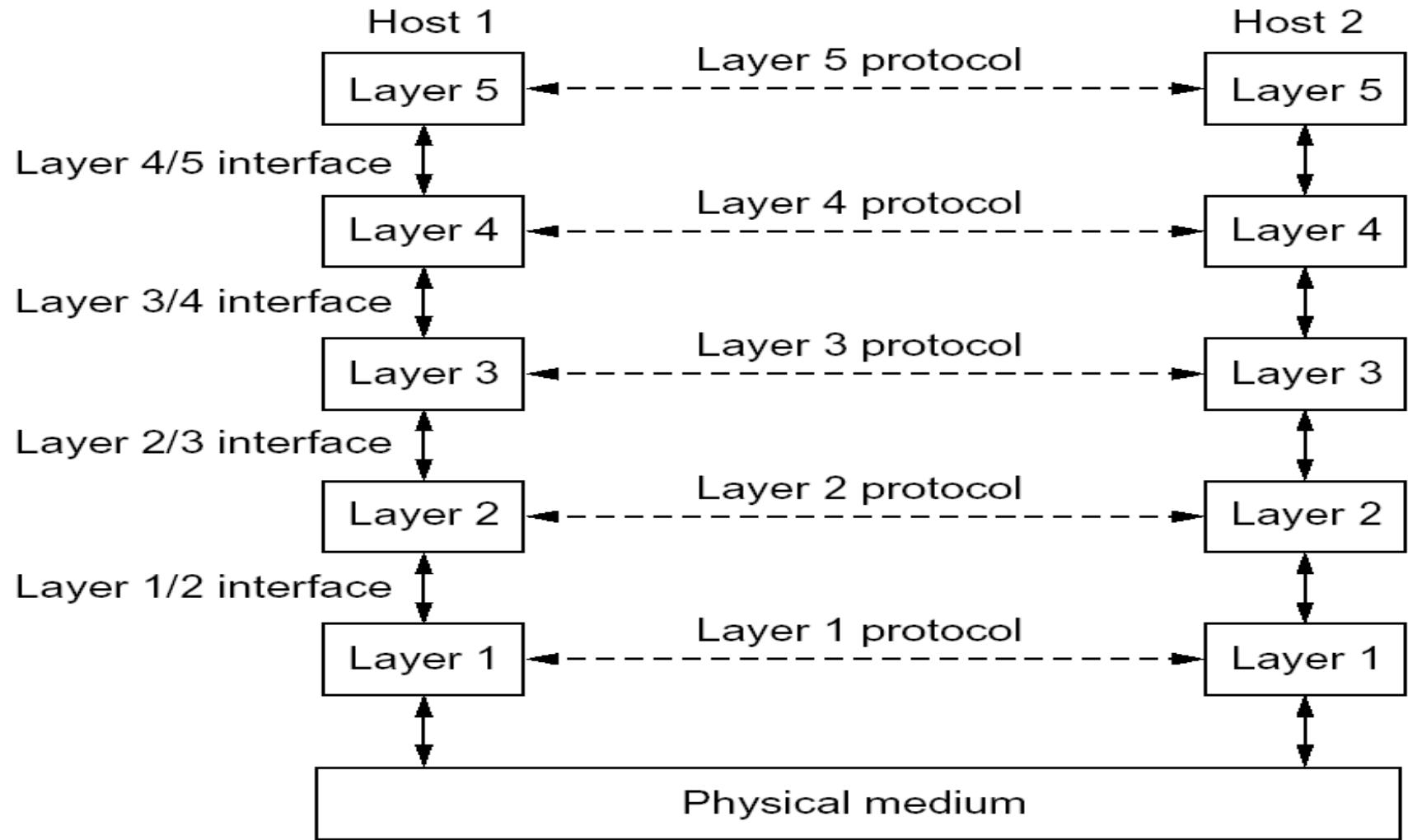
ISO/OSI Concepts

- ISO - International Standard Organization
 - OSI - Open System Interconnection
-
- Service - says **what** a layer does
 - Interface - says **how** to **access** the service
 - Protocol - specifies **how** the service is **implemented**
 - a set of rules and formats that govern the communications between two or more **peers**

An Example of Layering



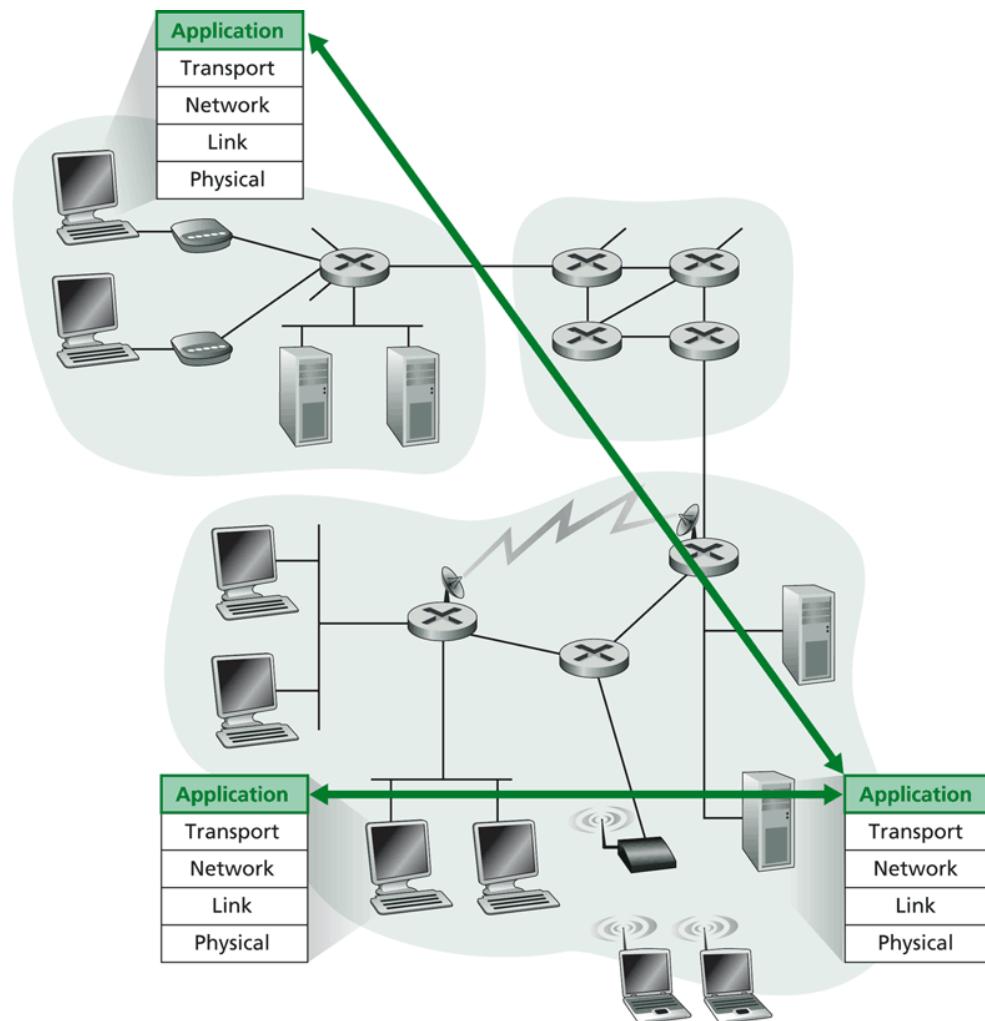
An Example of Layering



Layering -> Logical Communication

E.g.: application

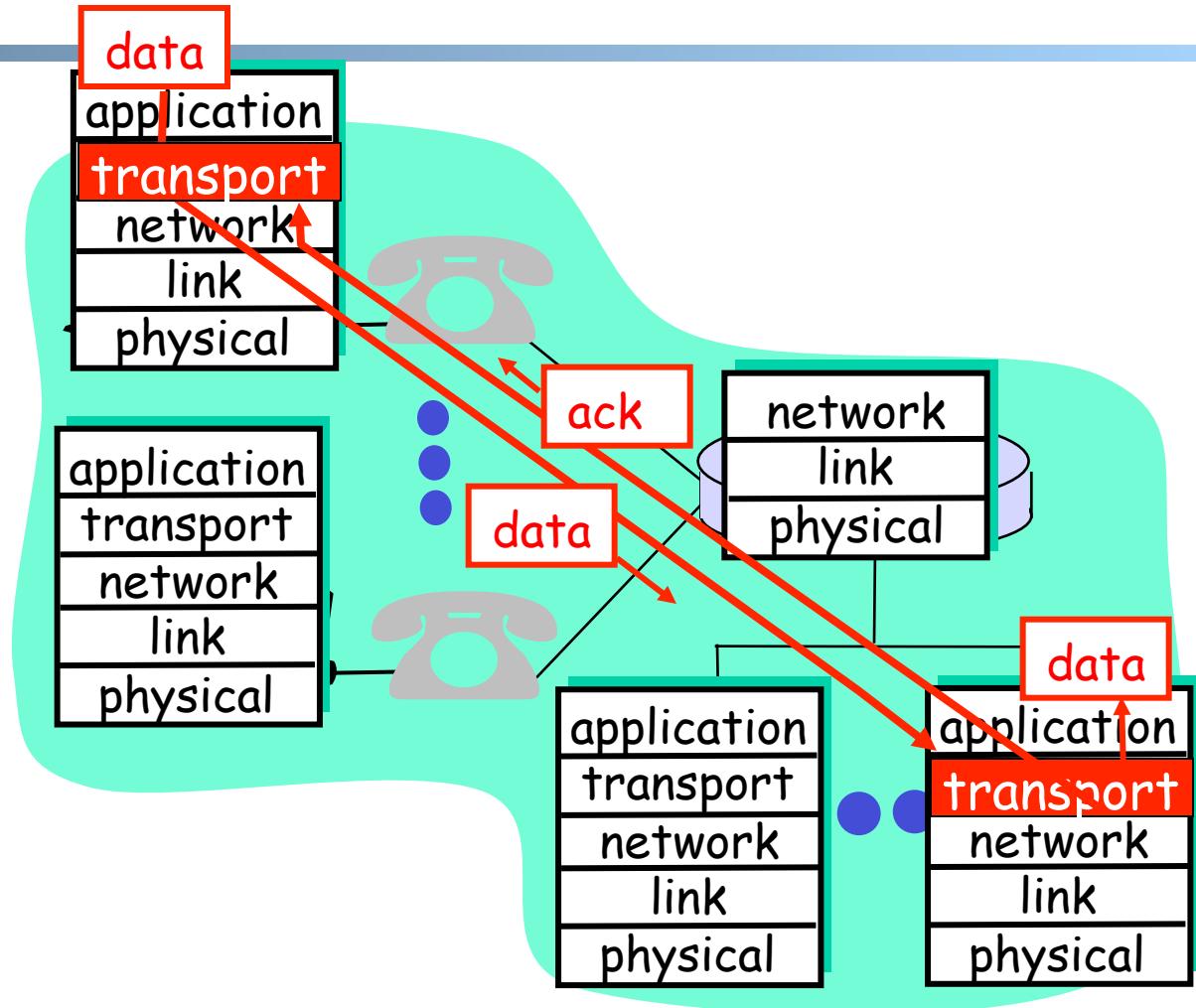
- provide services to users
- application protocol:
 - send messages to peer
 - for example, HELO, MAIL FROM, RCPT TO are messages between two SMTP peers



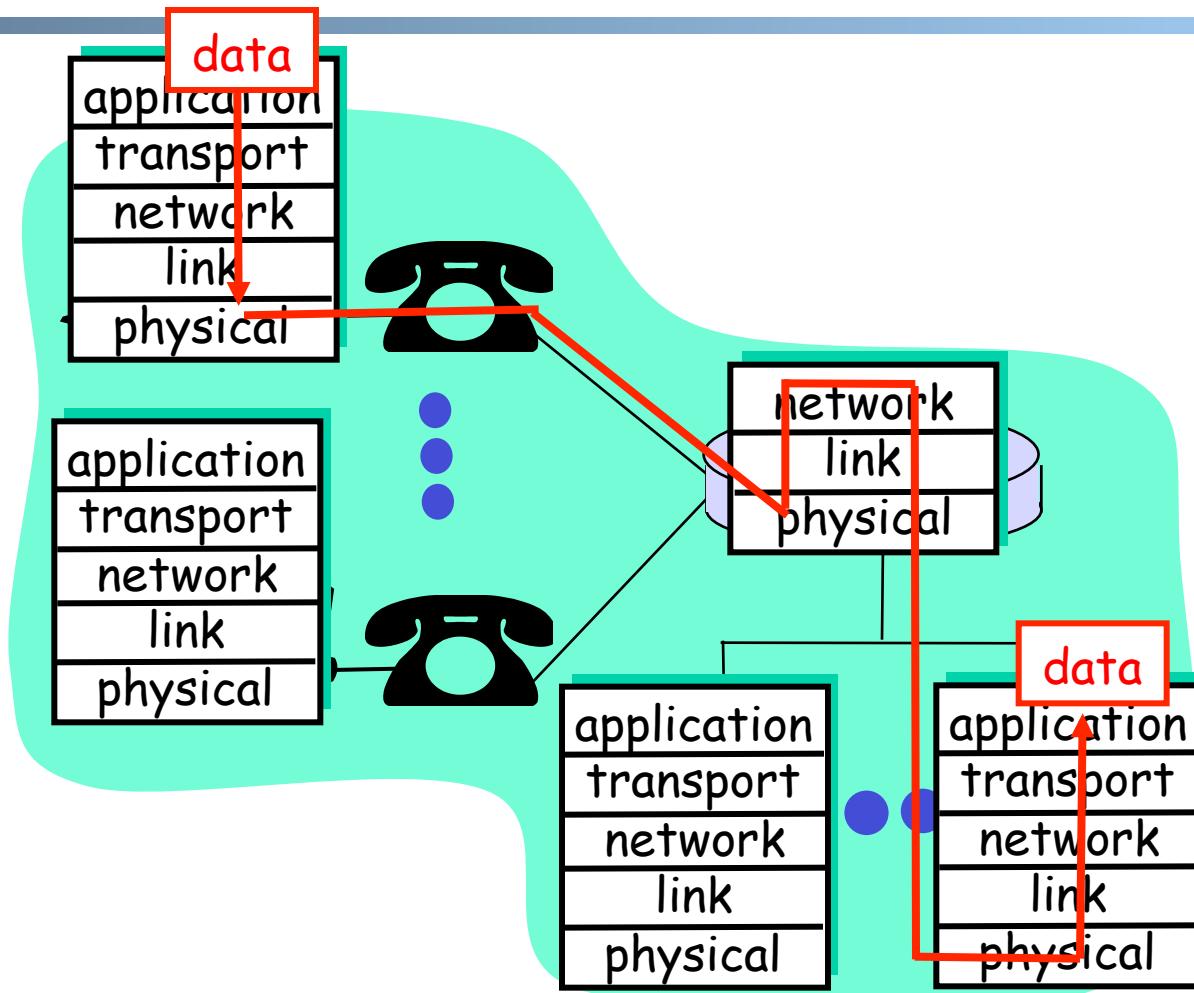
Layering: Logical Communication

E.g.: transport

- take msg from app
- Transport protocol
 - add control info to form "datagram"
 - send datagram to peer
 - wait for peer to ack receipt; if no ack, retransmit



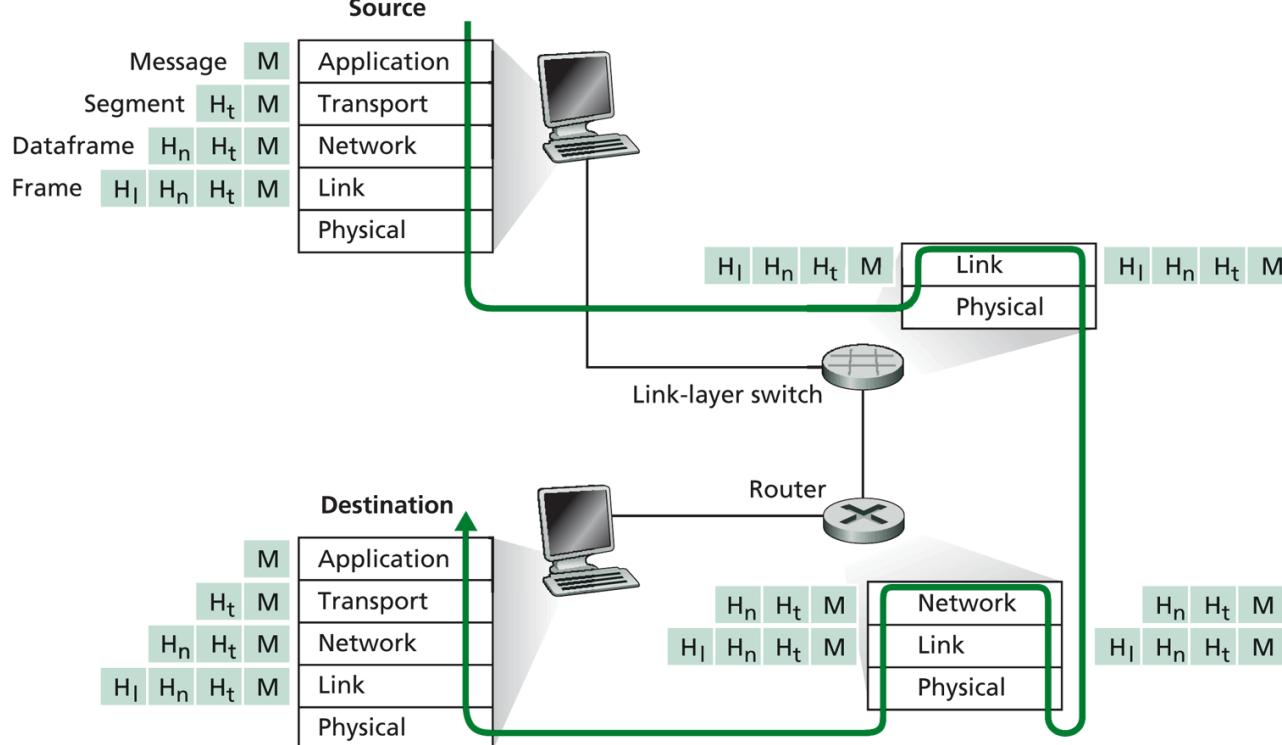
Layering: Physical Communication



Protocol Layering and Meta Data

Each layer takes data from above

- adds header (meta) information to its peer
create new data unit
- passes new data unit to layer below



Key design issue:
How do you *divide* functionalities
among the layers?

Outline

- Review
- A taxonomy of communication networks
- Layered network architecture
 - what is layering?
 - why layering?
 - *how to determine the layers?*

The End-to-End Arguments

The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the endpoints of the communication systems. Therefore, providing that questioned function as a feature of the communications systems itself is not possible.

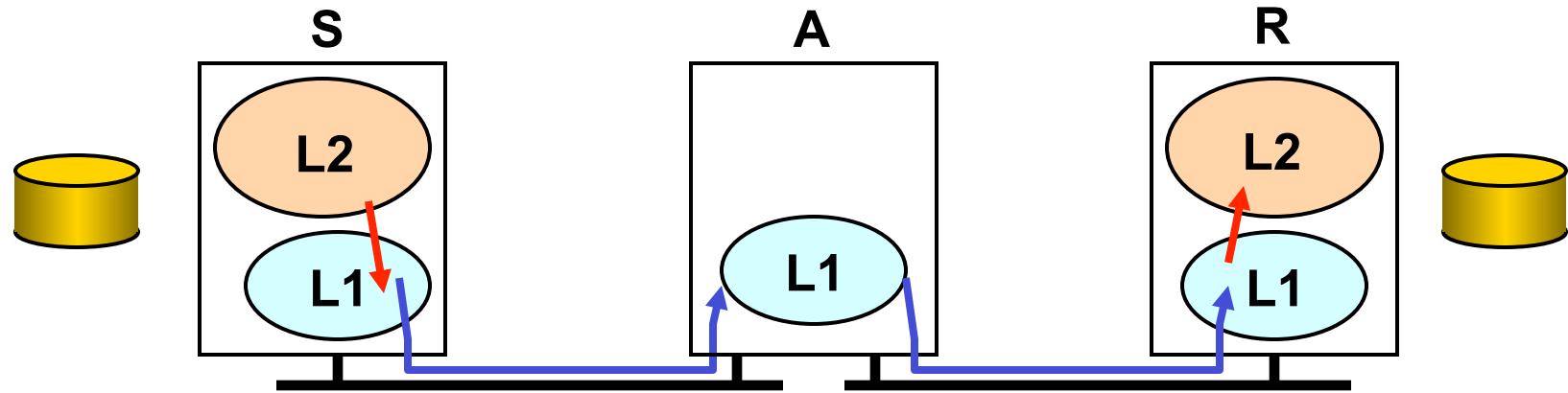
J. Saltzer, D. Reed, and D. Clark, 1984

What does the End-to-End Arguments Mean?

- The application knows the requirements best, place functionalities as high in the layer as possible

- Think twice before implementing a functionality at a lower layer, even when you believe it will be useful to an application

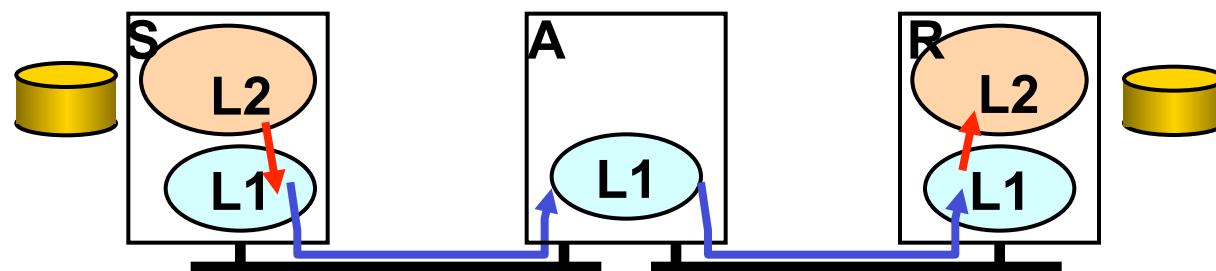
Example: Where to Provide Reliability ?



- Solution 1: the network (lower layer L1) provides reliability, i.e., each hop provides reliability
- Solution 2: the end host (higher layer L2) provides reliability, i.e., end-to-end check and retry

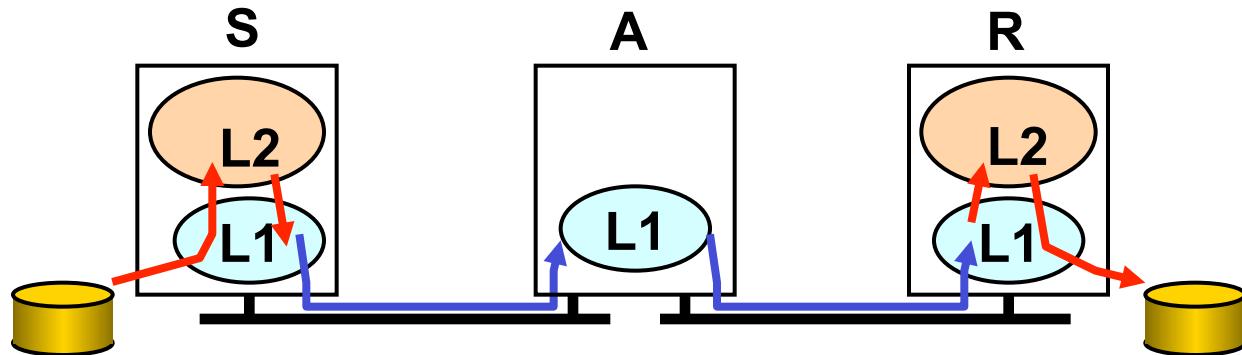
What are Reasons for Implementing Reliability at Higher Layer ?

- The lower layer cannot completely provide the functionality
 - the receiver has to do the check anyway !
- Implementing it at lower layer increases complexity, cost and overhead at lower layer
 - shared by all upper layer applications → everyone pays for it, even if you do not need it
- The upper layer
 - knows the requirements better and thus may choose a better approach to implement it



Are There Reasons Implementing Reliability at Lower Layer ?

- Improve performance, e.g., if high cost/delay/... on a local link
 - improves efficiency
 - reduces delay
- Share common code, e.g., reliability is required by multiple applications

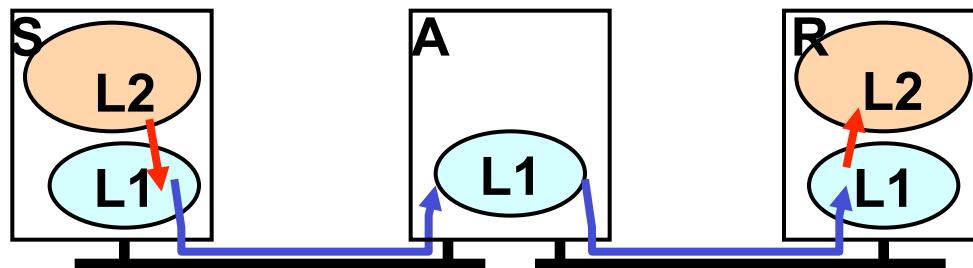


Summary: End-to-End Arguments

- If a higher layer can do it, don't do it at a lower layer -- the higher the layer, the more it knows about the best what it needs
- Add functionality in lower layers iff it
 - (1) is used by and improves performance of a large number of (current and potential future) applications,
 - (2) does not hurt (too much) other applications, and
 - (3) does not increase (too much) complexity/overhead
- Practical tradeoff, e.g.,
 - allow multiple interfaces at a lower layer (one provides the function; one does not)

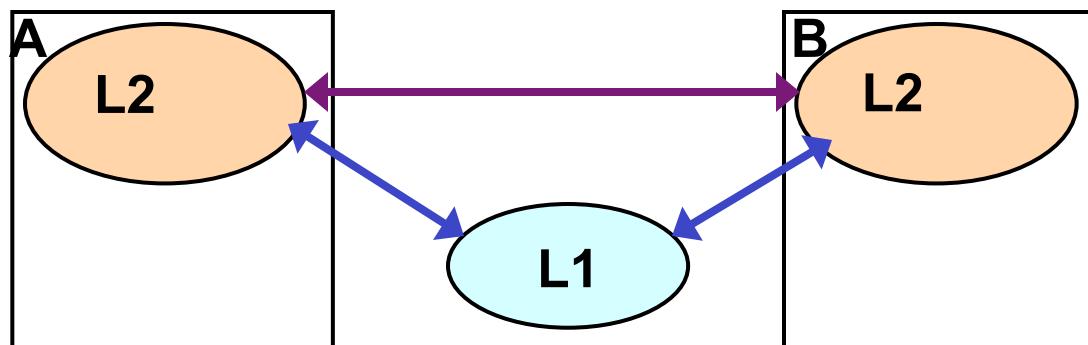
Examples

- We used reliability as an example
- Assume two layers (L1: network; L2: end-to-end). Where may you implement the following functions?
 - security (privacy of traffic)
 - quality of service (e.g., delay/bandwidth guarantee)
 - congestion control (e.g., not to overwhelm network links or receiver)



Example

- Consider the presence service in a social networking system: shows which contacts are online (e.g., skype, MSN)
 - implementing by end user's host app or through a third party service?



Challenges



- Challenges to build a good (networking) system: find the right balance between:

end-to-end arguments

performance



reuse, interoperability,
implementation effort
(apply layering concepts)

No universal answer: the answer depends on the goals and assumptions!

Backup Slides

The Design Philosophy of the DARPA Internet

Goals

0. **Connect different networks**
1. Survivability in the face of failure
2. Support multiple types of services
3. Accommodate a variety of networks
4. Permit distributed management of resources
5. Be cost effective
6. Permit host attachment with a low level of effort
7. Be accountable

Survivability in the Face of Failure: Questions

- What does the goal mean?
- Why is the goal important?
- How does the Internet achieve this goal?
- Does the Internet achieve this goal (or in what degree does the Internet achieve this goal)?

Survivability in the Face of Failure

- Continue to operate even in the presence of network failures (e.g., link and router failures)
 - as long as the network is not partitioned, two endpoints should be able to communicate...moreover, any other failure (excepting network partition) should be **transparent** to endpoints
- Decision: maintain state only at end-points (fate-sharing)
 - eliminate the problem of handling state inconsistency and performing state restoration when router fails
- Internet: **stateless** network architecture

Support Multiple Types of Service: Questions

- What does this goal mean?
- Why is the goal important?
- How does the Internet achieve this goal?
- Does the Internet achieve this goal (or in what degree does the Internet achieve this goal)?

Support Multiple Types of Service

- ❑ Add UDP to TCP to better support other types of applications
 - e.g., “real-time” applications
- ❑ This was arguably the main reason for separating TCP and IP
- ❑ Provide datagram abstraction: lower common denominator on which other services can be built: everything over IP
 - service differentiation was considered (remember ToS?), but this has never happened on the large scale (Why?)

Support a Variety of Networks: Questions

- What does the goal mean?
- Why is this goal important?
- How does the Internet achieve this goal?
- Does the Internet achieve this goal (or in what degree does the Internet achieve this goal)?

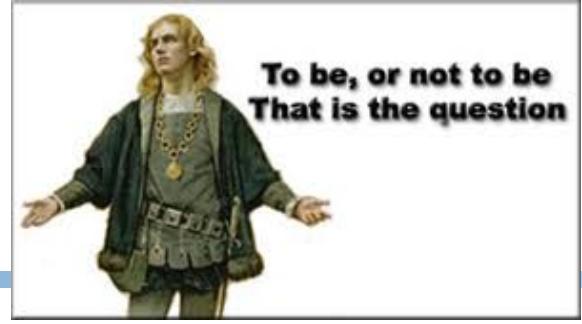
Support a Variety of Networks

- Very successful
 - because the minimalist service; it requires from underlying network only to deliver a packet with a “reasonable” probability of success
- ...does not require:
 - reliability
 - in-order delivery
- The mantra: IP over everything
 - Then: ARPANET, X.25, DARPA satellite network..
 - Now: ATM, SONET, WDM...

Other Goals

- Permit distributed management of resources
- Be cost effective
- Permit host attachment with a low level of effort
- Be accountable

To Partition, or not to Partition: This is the Question.



Assume:

R = link bandwidth (bps)

L = packet length (bits)

$S = L / R$

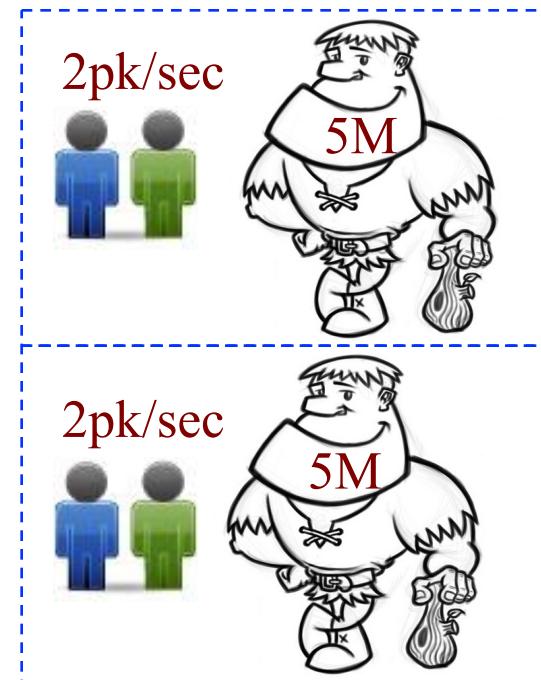
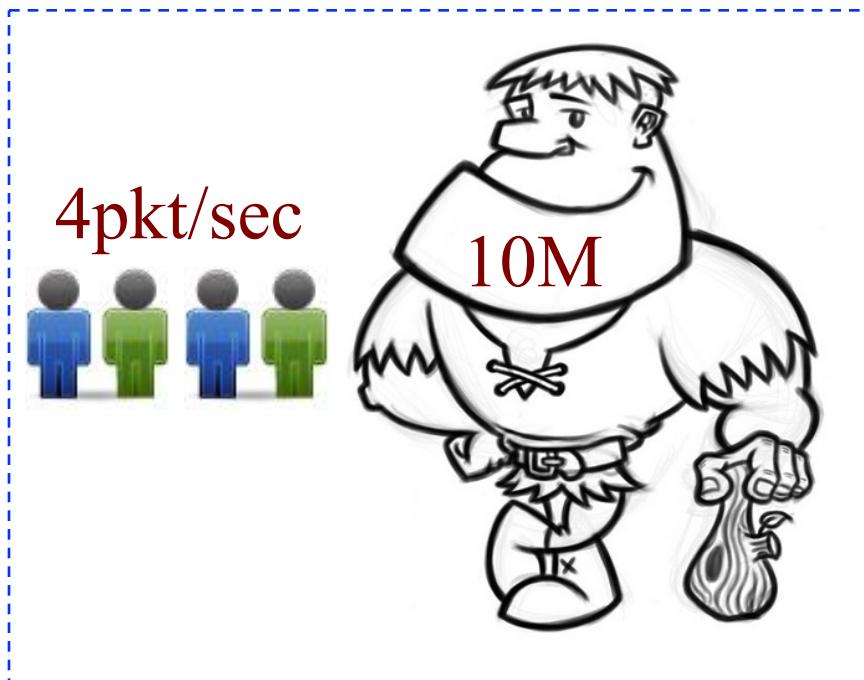
a = average packet arrival rate (pkt/sec)

Setup: n streams; each stream has an arrival rate of a/n

Comparison: each stream reserves $1/n$ bandwidth or not

- Case 1 (not reserve):** all arrivals into a single queue serving with rate R
- Case 2 (reserve):** the arrivals are divided into n links with rate R/n each

Partition or Not



Network Applications: Overview, EMail

Y. Richard Yang

<http://zoo.cs.yale.edu/classes/cs433/>

1/27/2016

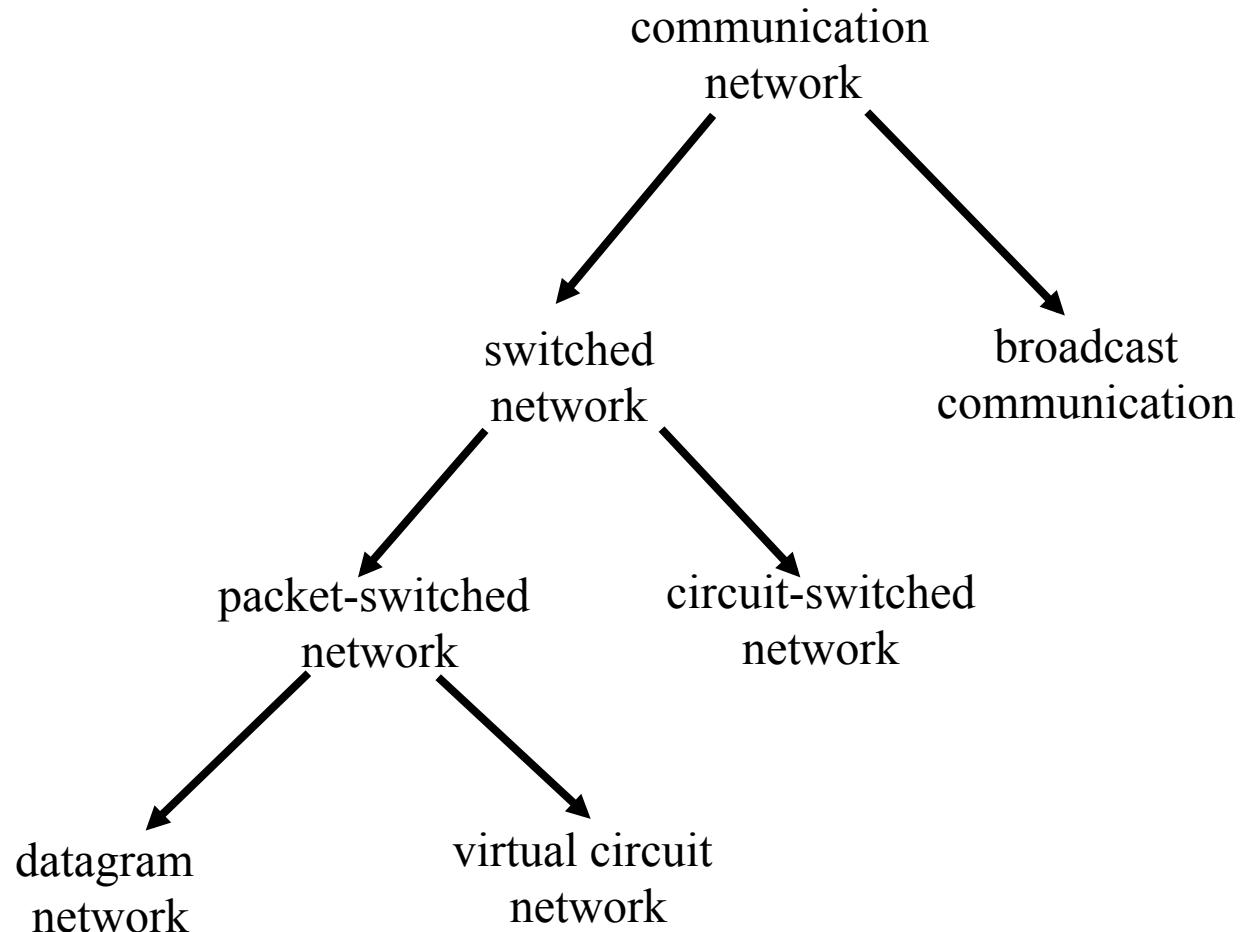
Outline

- Admin and recap
- ISO/OSI Layering and Internet Layering
- Application layer overview
- Network applications
 - Email

Admin

Questions on Assignment One

Recap: Summary of the Taxonomy of Communication Networks



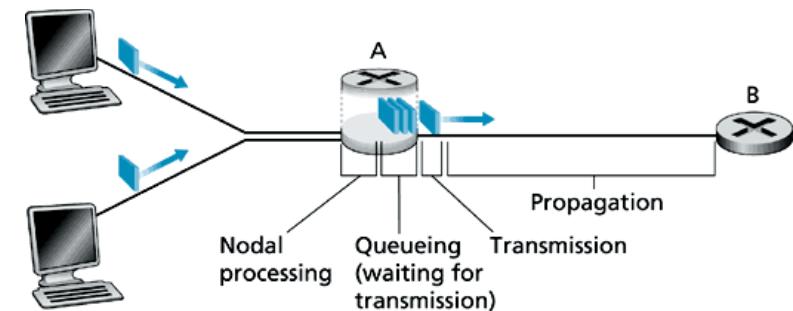
Recap: Statistical Multiplexing

A simple model to compare bandwidth efficiency of

- reservation/dedication (aka circuit-switching) and
- no reservation (aka packet switching)

setup

- a single bottleneck link with rate R
- n flows; each flow has an arrival rate of a/n



- no reservation: all arrivals into the single link with rate R , the queueing delay + transmission delay:

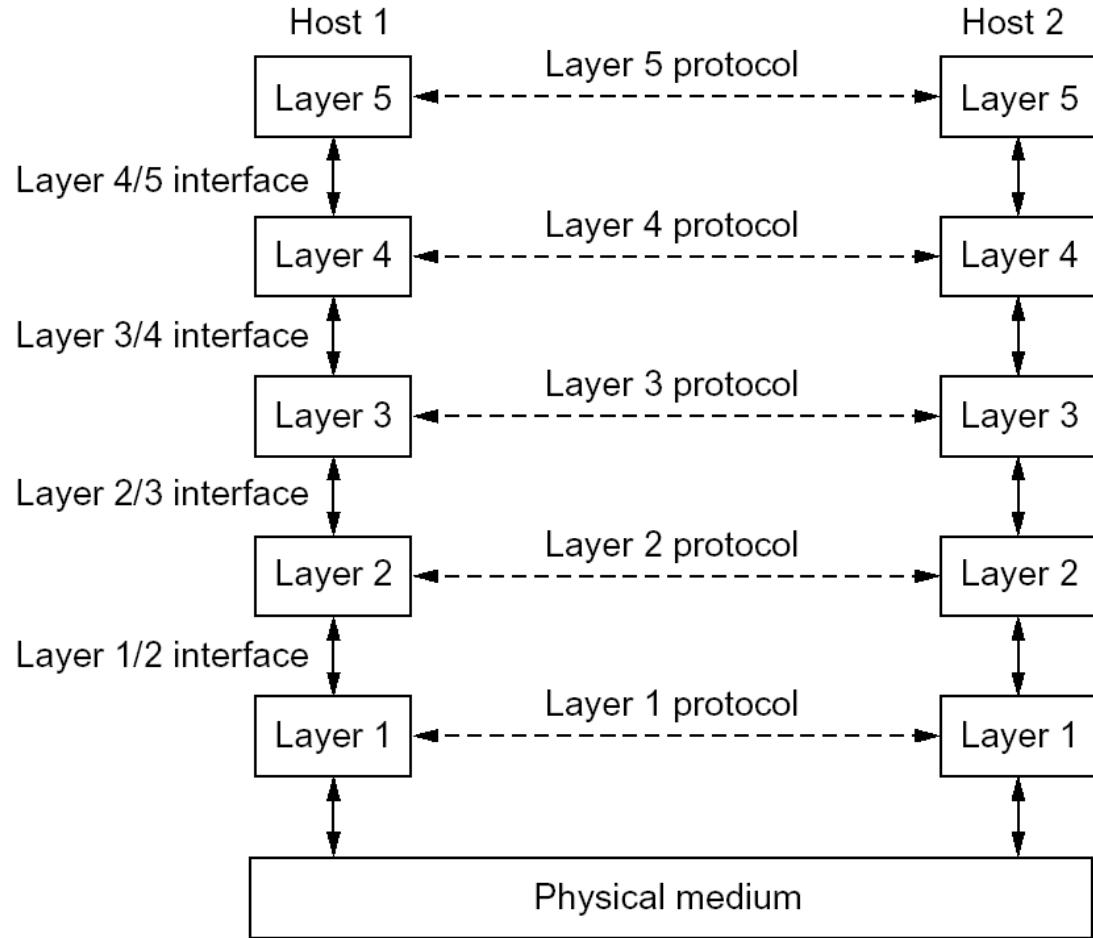
$$\frac{L}{R} \frac{1}{1 - \rho}$$

- reservation: each flow uses its own reserved (sub)link with rate R/n , the queueing delay + transmission delay:

$$n \frac{L}{R} \frac{1}{1 - \rho}$$

Recap: Layering

- Why layering
 - reference model
 - modularization
- Concepts
 - service, interface, and protocol
 - physical vs logical communication
- Key design decision:
what functionalities to put in each layer:
End-to-end arguements

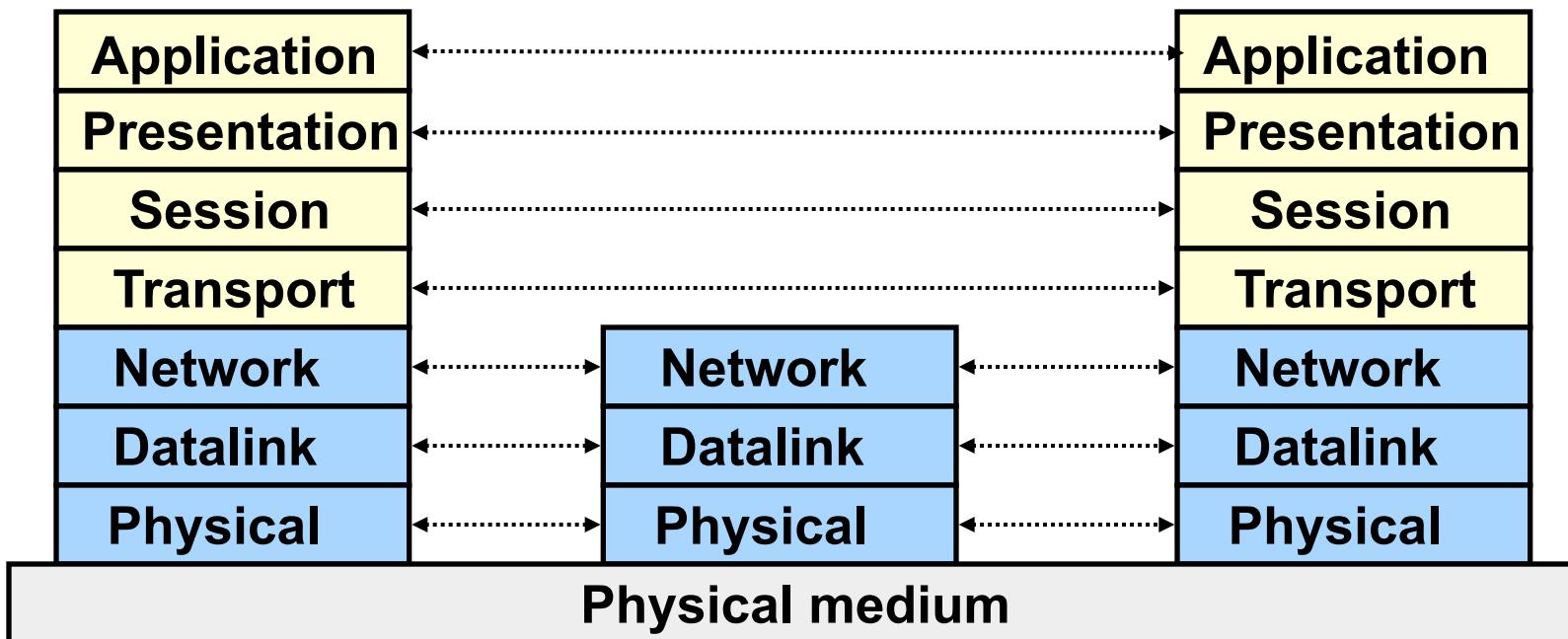


Outline

- Recap
- *ISO/OSI Layering and Internet Layering*
- Application layer overview

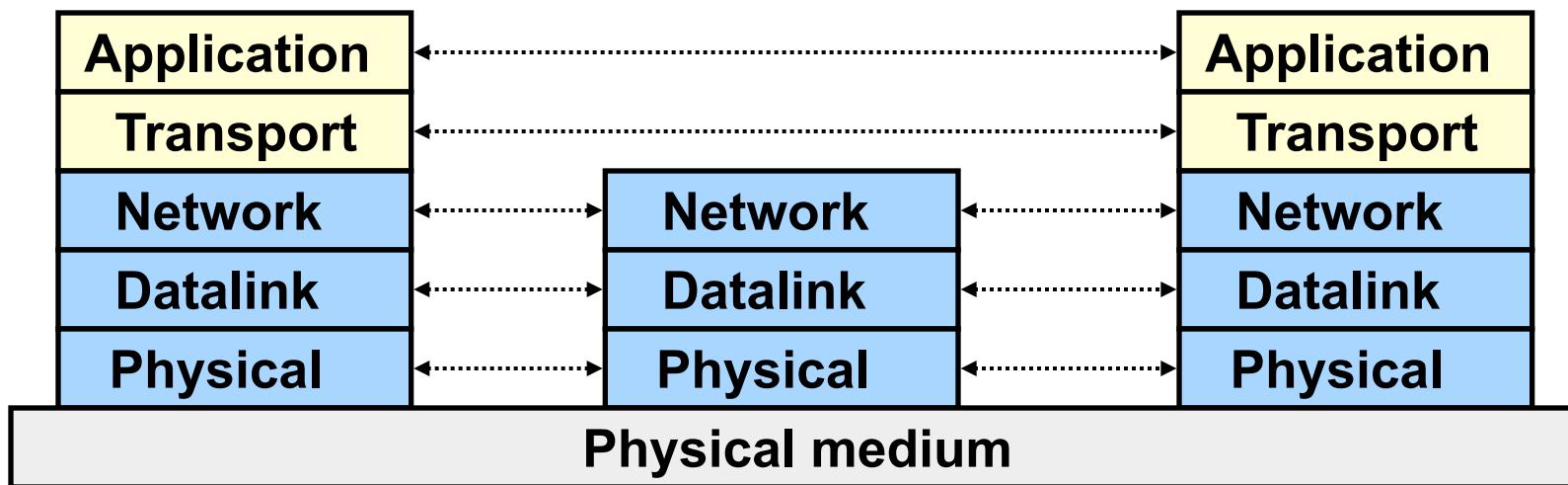
ISO/OSI Reference Model

- Seven layers
 - lower three layers are hop-by-hop
 - next four layers are end-to-end (host-to-host)



Internet Layering

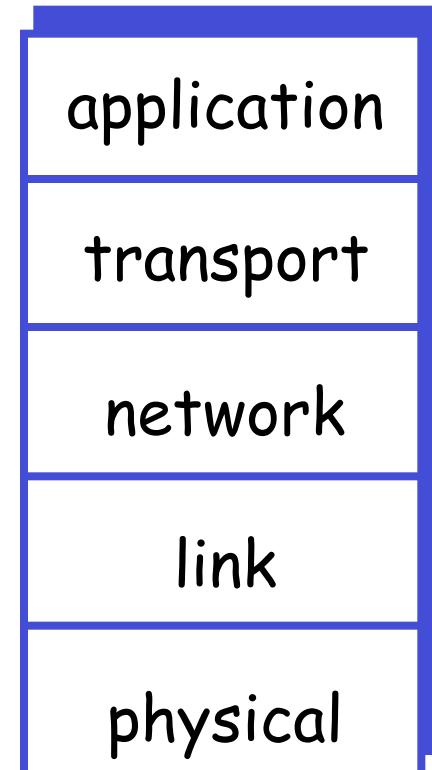
- Lower three layers are hop-by-hop
- Next two layers are end-to-end



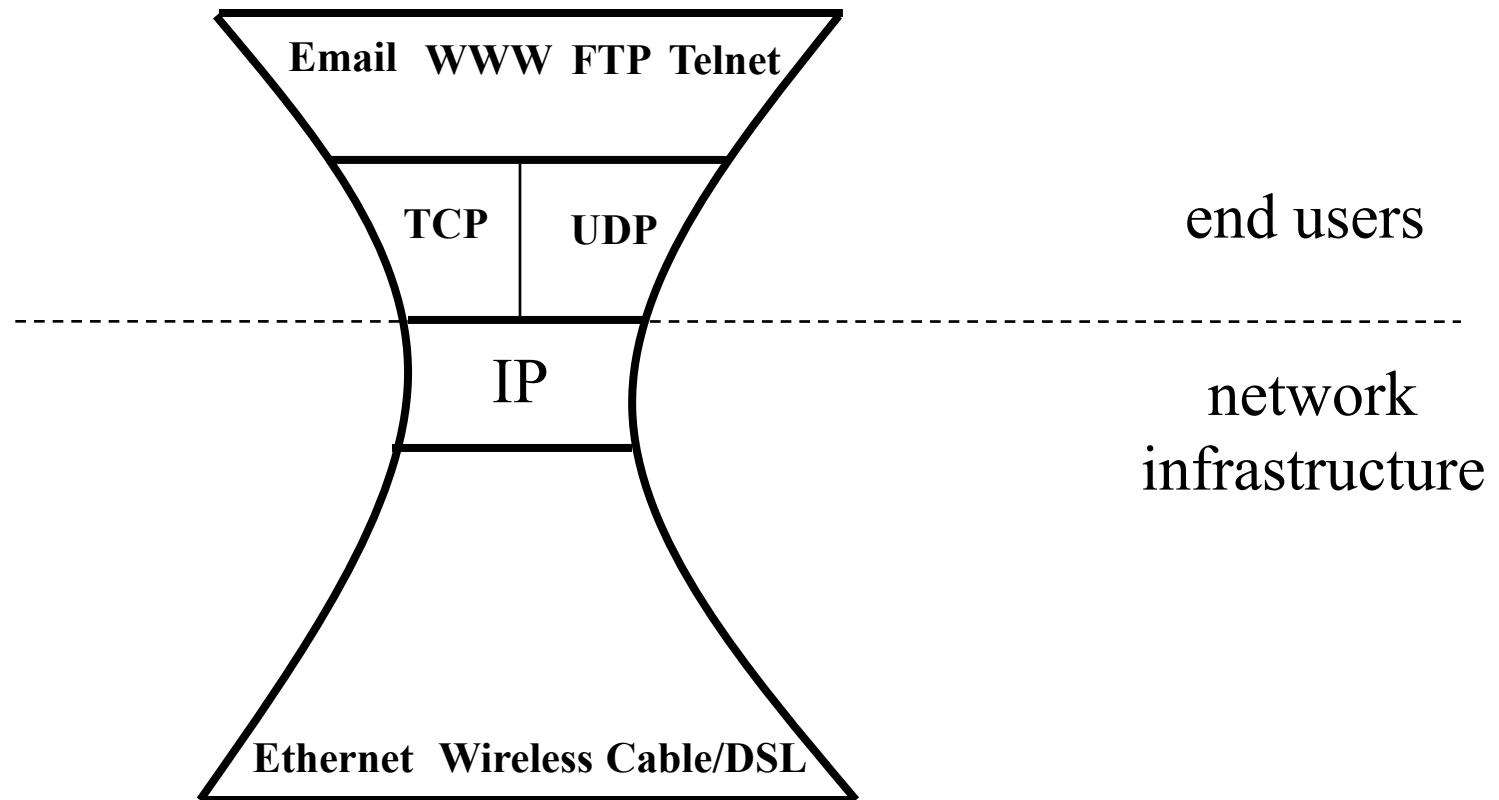
Internet Protocol Layers

□ Five layers

- **Application:** specific network applications
 - ftp, smtp, http, p2p, IP telephony, ...
- **Transport:** host-host data transfer
 - tcp (reliable), udp (not reliable)
- **Network:** routing of datagram from source to destination
 - ipv4, ipv6
- **Link:** data transfer between neighboring network elements
 - ethernet, 802.11, cable, DSL, ...
- **Physical:** bits “on the wire”
 - cable, wireless, optical fiber



The Hourglass Architecture of the Internet



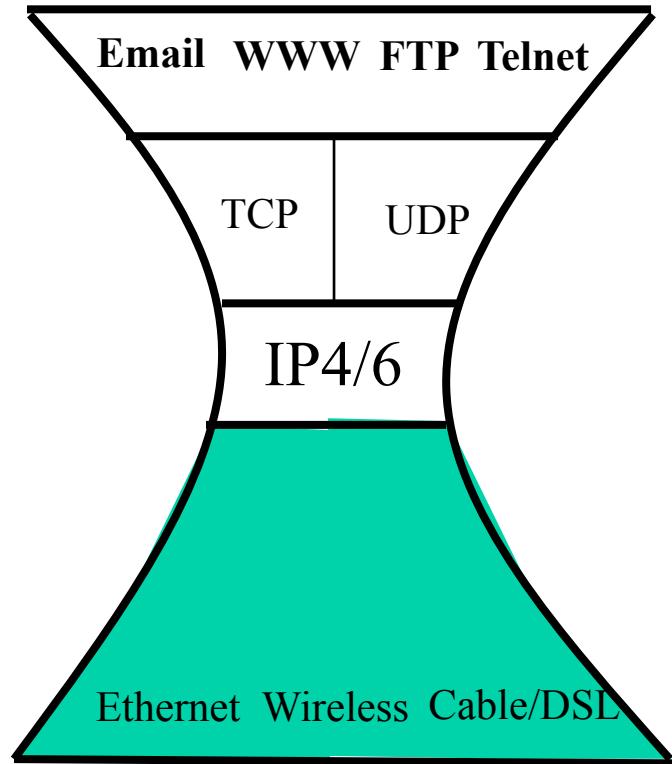
Link Layer (Ethernet)

□ Services

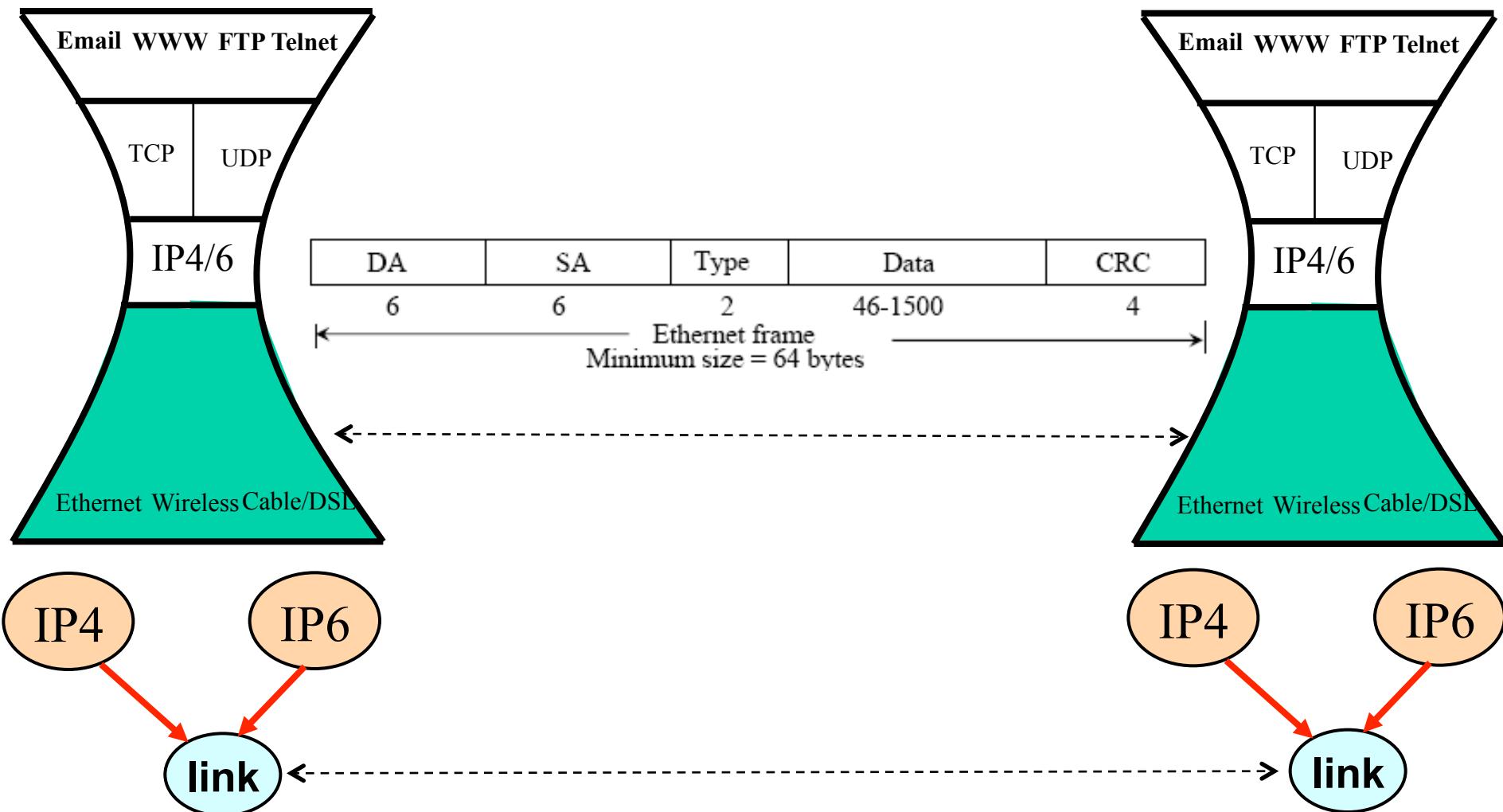
- multiple access control
 - arbitrate access to shared medium
- multiplexing/
demultiplexing
 - from/to the network layer
- error detection

□ Interface

- send frames to a directly
reachable peer



Link Layer: Protocol Header (Ethernet)



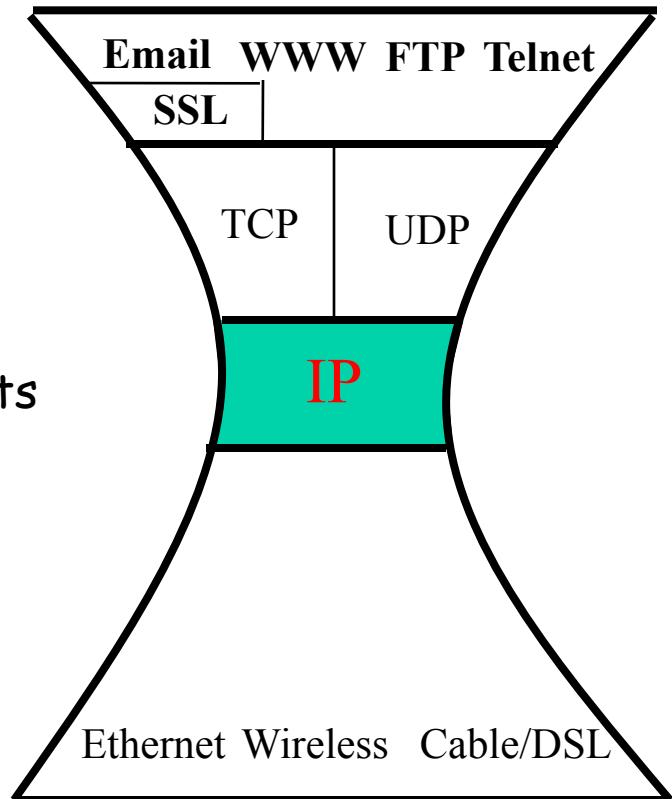
Network Layer: IP

□ Services

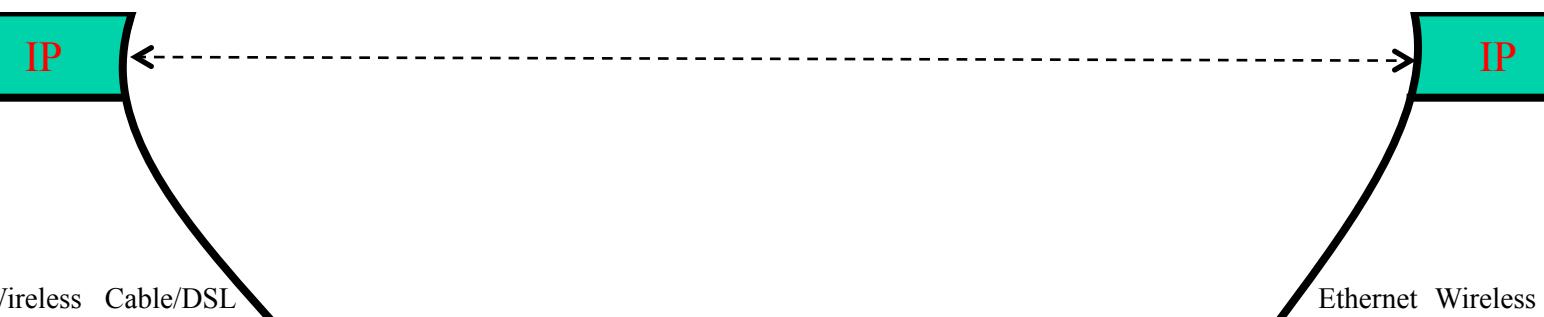
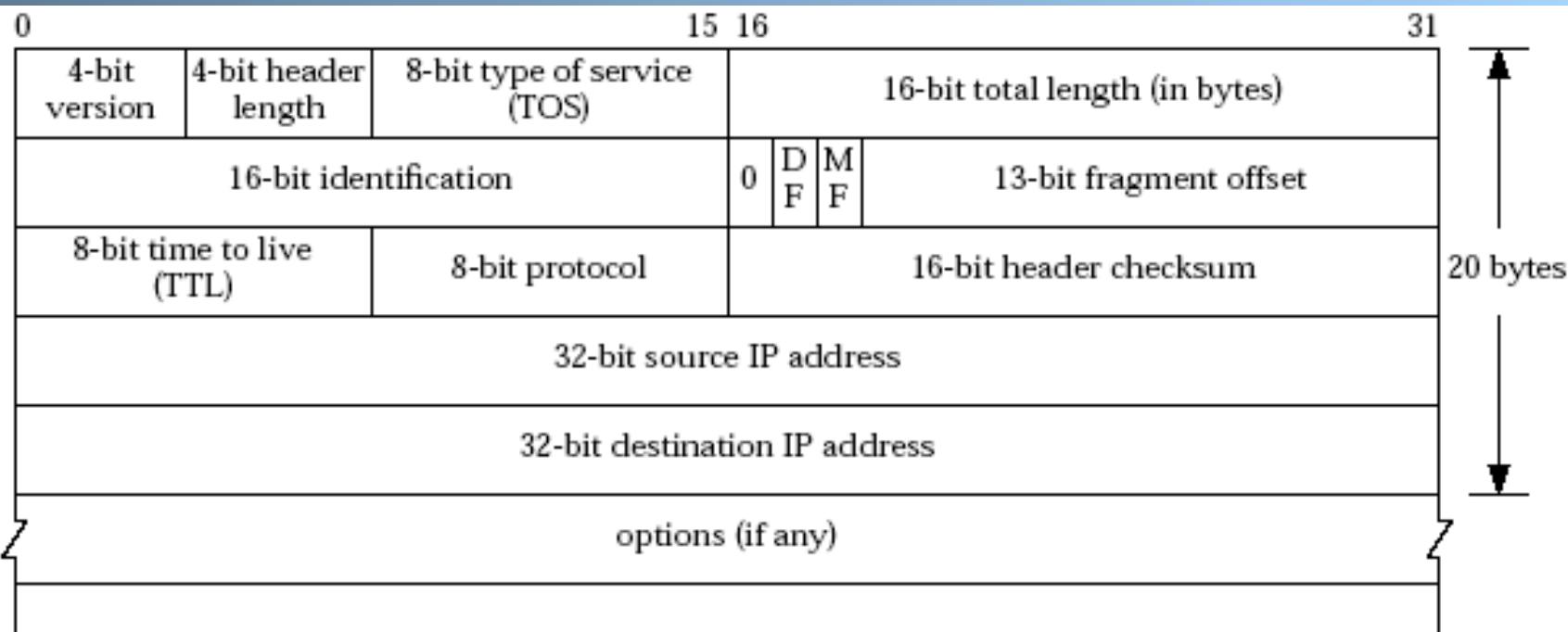
- **routing:** best-effort to send packets from source to destination
- **multiplexing/demultiplexing** from/to the transport
- **fragmentation and reassembling:** partition a fragment into smaller packets
 - removed in IPv6
- **error detection**
- **certain QoS/CoS**
- **does not provide** reliability or reservation

□ Interface:

- send a packet to a (transport-layer) peer at a specified global destination, with certain QoS/CoS

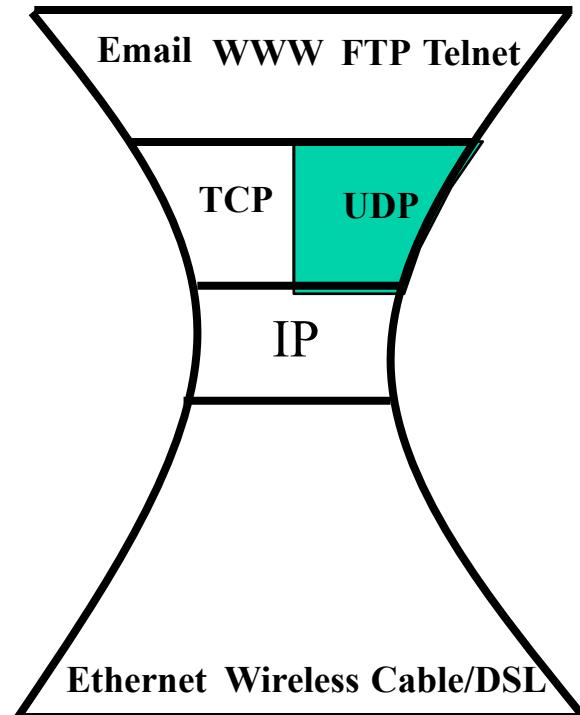


Network Layer: IPv4 Header

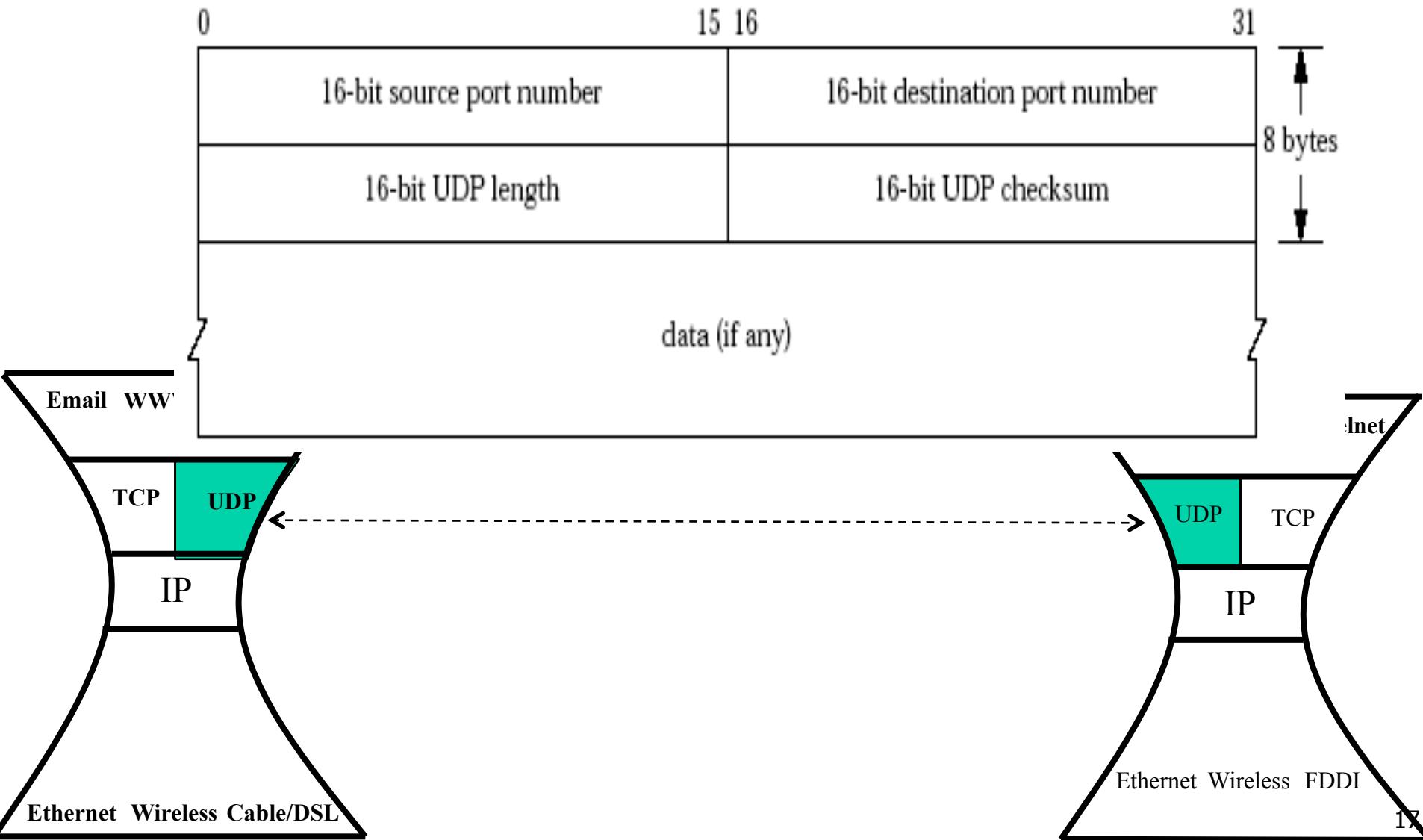


Services Provided by UDP

- A connectionless service
- Does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee
 - why is there a UDP?



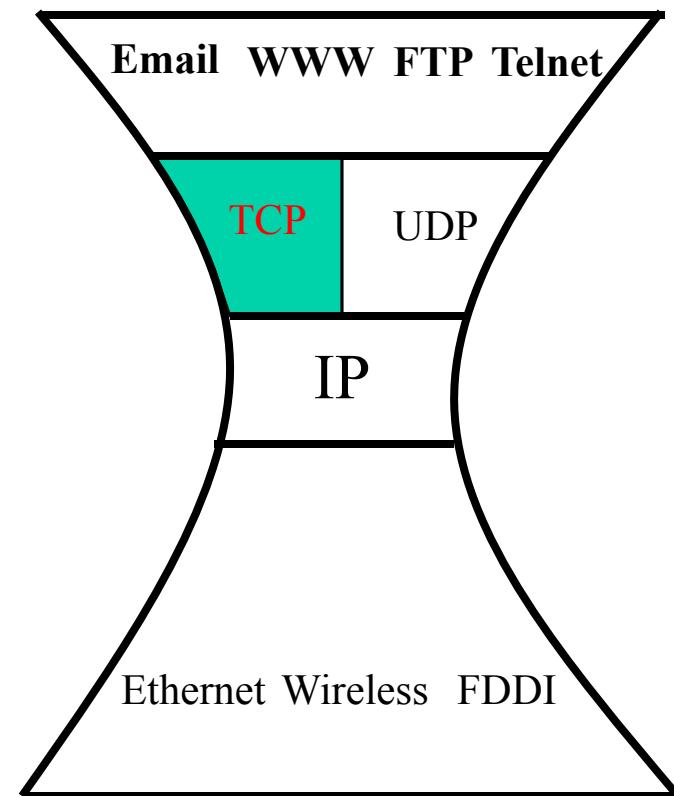
Transport Layer: UDP Header



Transport Layer: TCP

□ Services

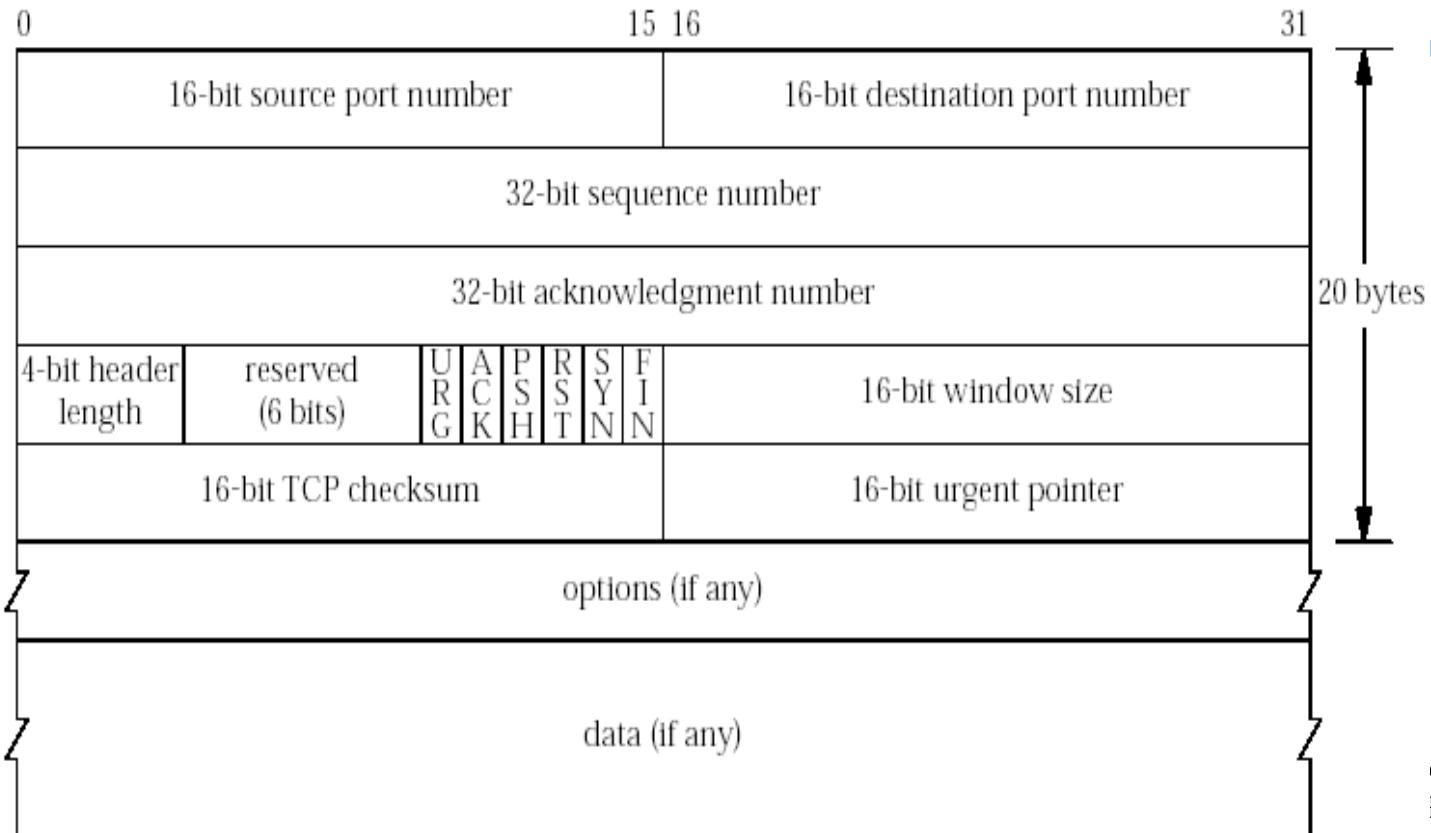
- multiplexing/demultiplexing
- reliable transport
 - between sending and receiving processes
 - setup required between sender and receiver: a **connection-oriented service**
- flow control: sender won't overwhelm receiver
- congestion control: throttle sender when network overloaded
- error detection
- **does not provide timing, minimum bandwidth guarantees**



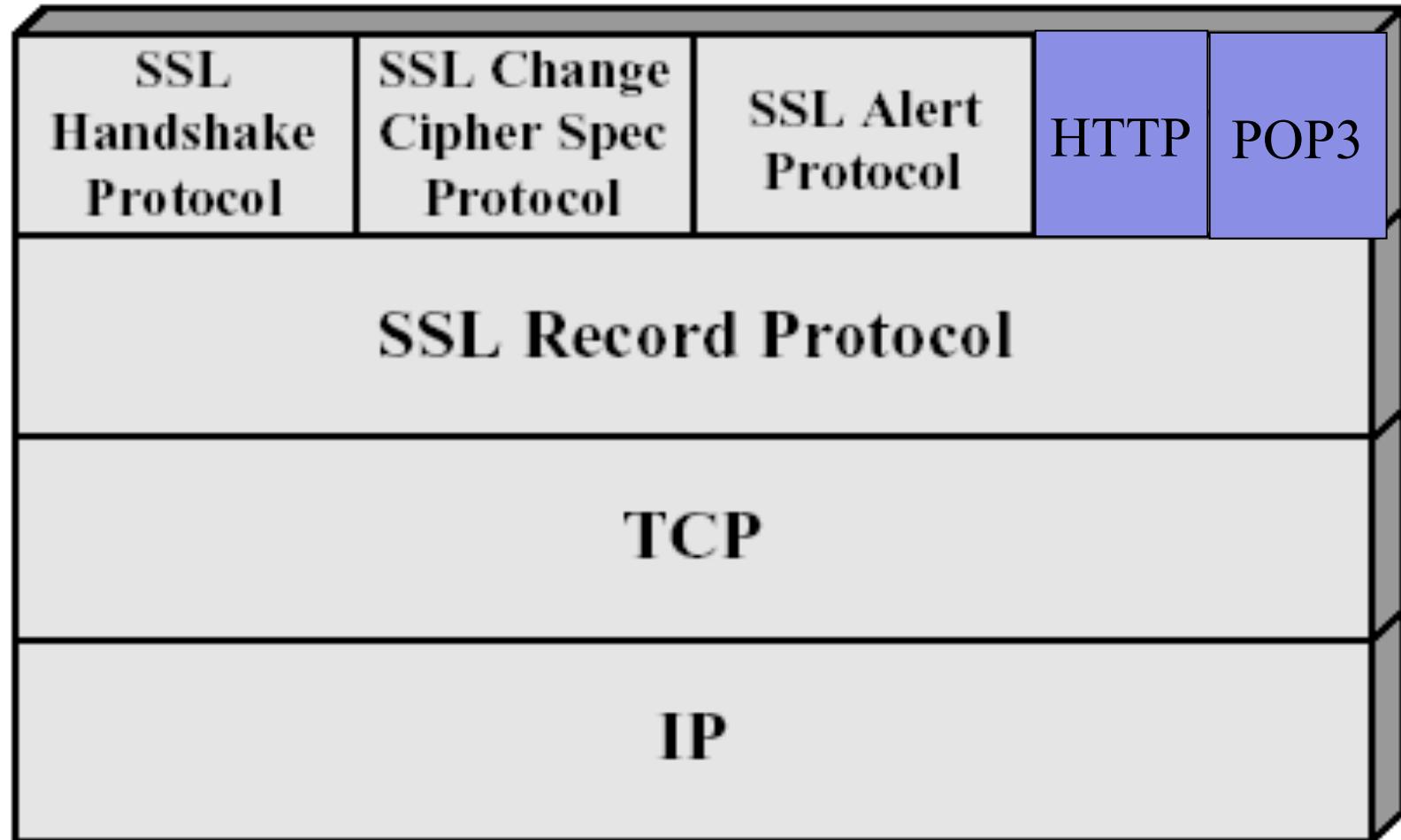
□ Interface:

- send a packet to a (app-layer) peer

Transport Layer: TCP Header

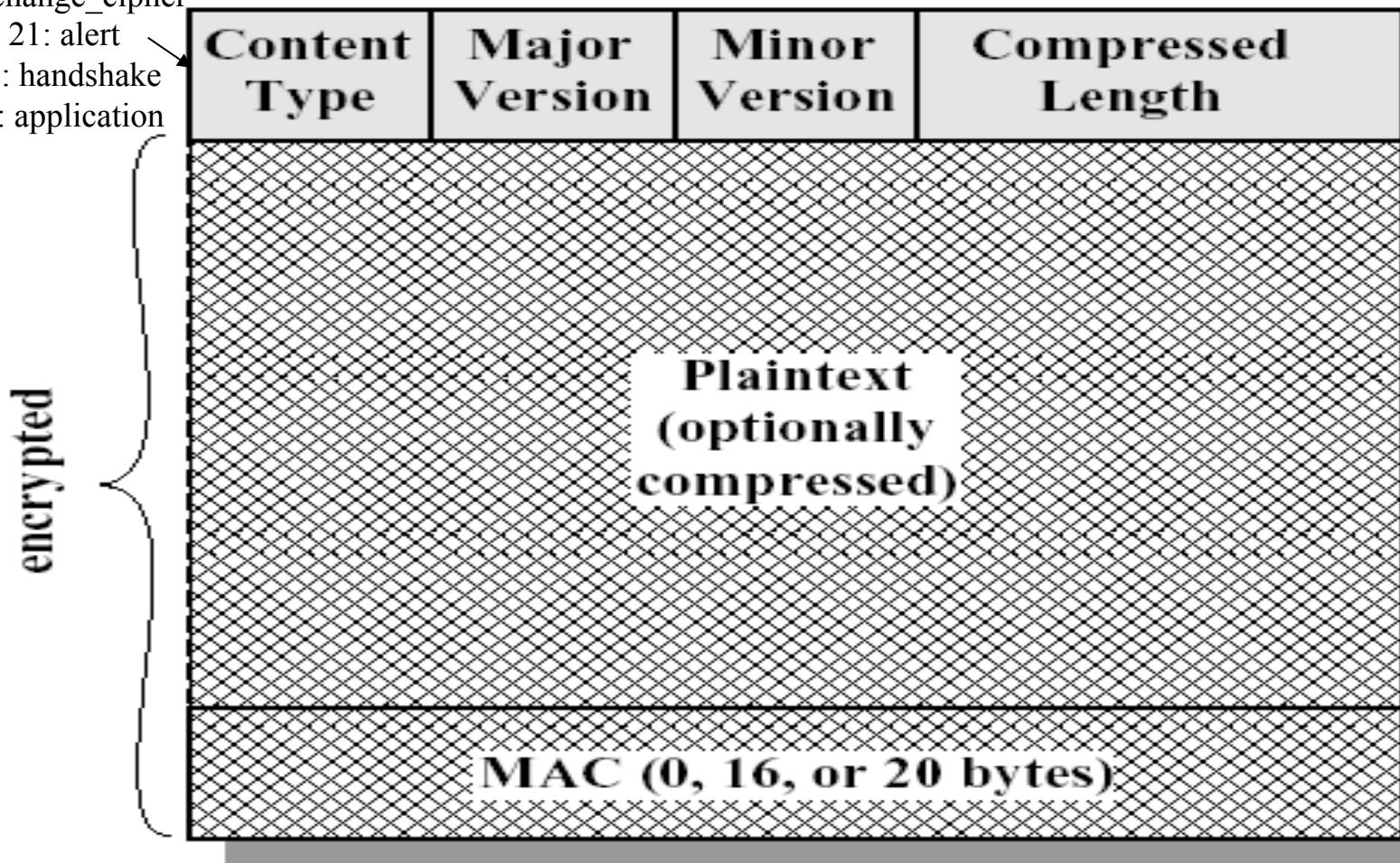


Secure Socket Layer Architecture



SSL Record-Layer Packet Format

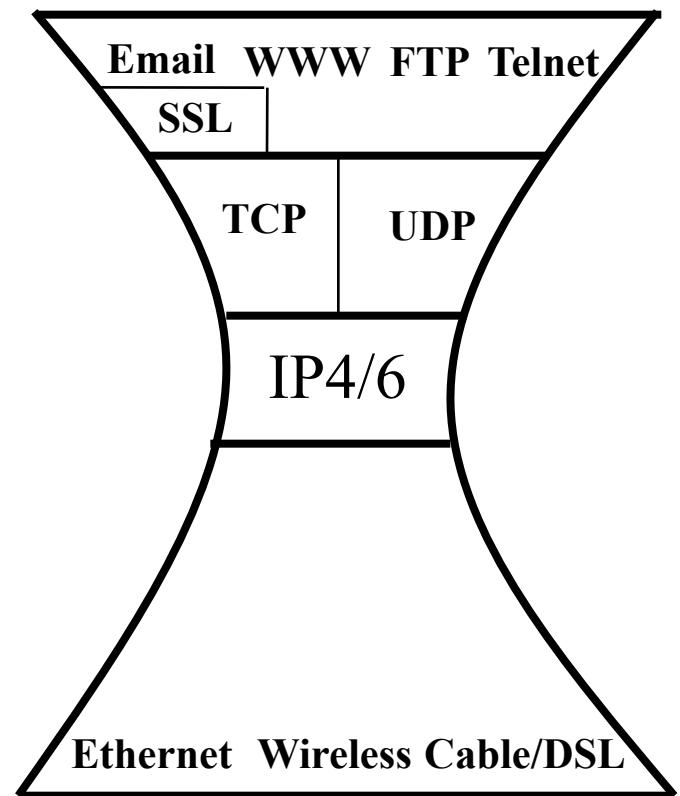
20: change_cipher
21: alert
22: handshake
23: application



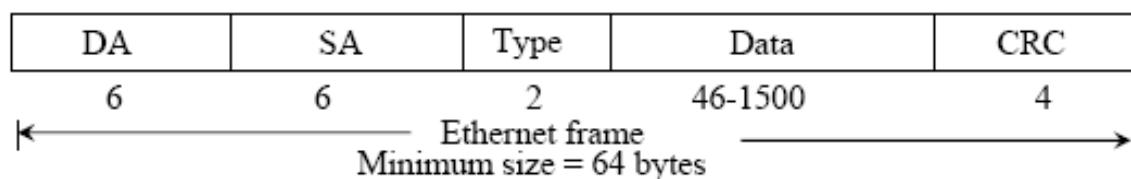
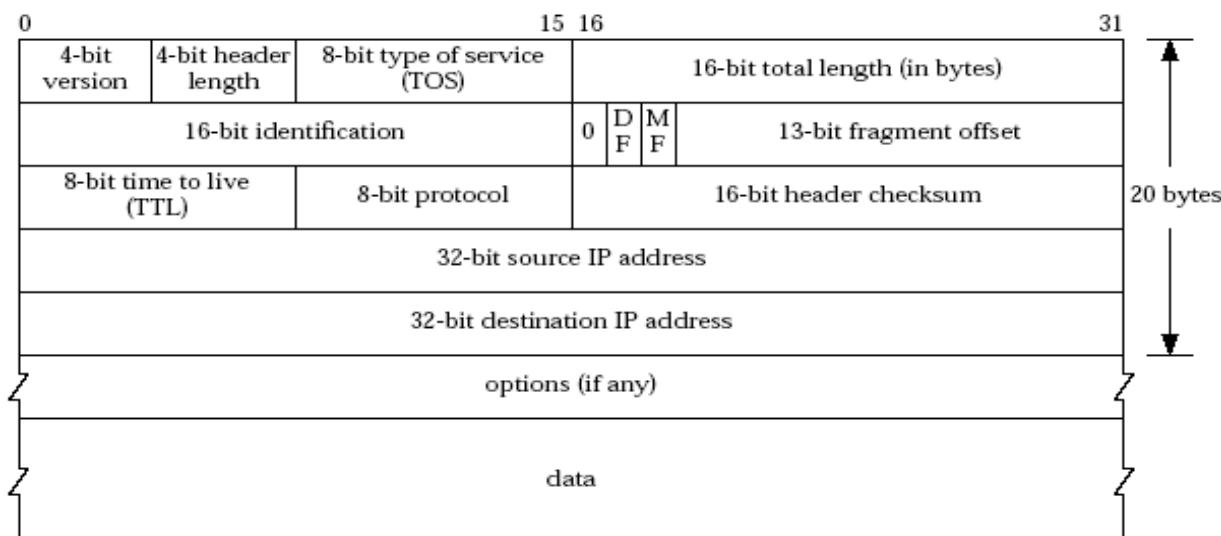
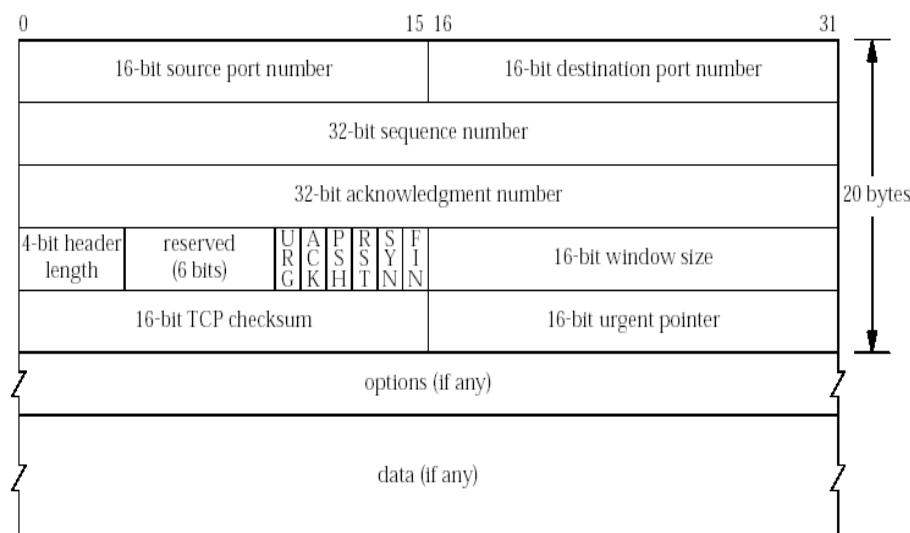
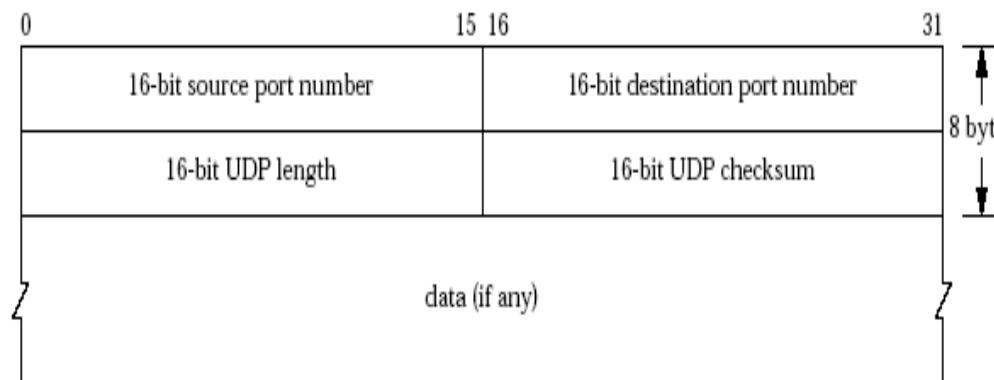
Summary: The Big Picture of the Internet

- Hosts and routers:
 - ~ 1 bil. hosts (July 2015)
 - autonomous systems organized roughly hierarchical
 - backbone links at 100 Gbps

- Software:
 - datagram switching with virtual circuit support at backbone
 - layered network architecture
 - use end-to-end arguments to determine the services provided by each layer
 - the hourglass architecture of the Internet



Protocol Formats



Outline

- Recap
- ISO/OSI Layering and Internet Layering
- *Application layer overview*

Application Layer: Goals

- Conceptual + implementation aspects of network application protocols
 - client server paradigm
 - peer to peer paradigm
 - network app. programming

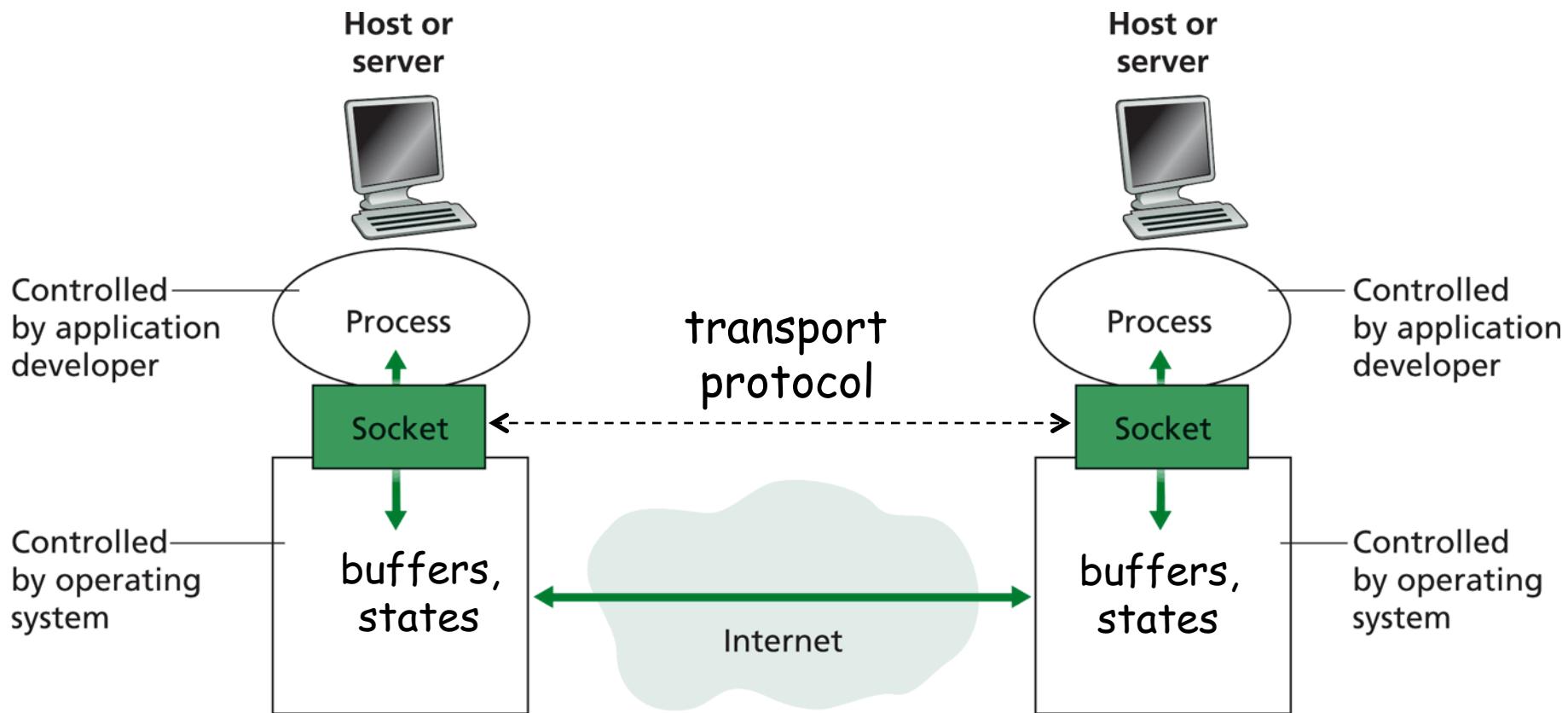
- Learn about applications by examining common applications
 - smtp/pop
 - dns
 - http
 - content distribution

How does an Application Access the Transport Service?

API: application programming interface

- Defines interface between application and transport layer
- Multiple APIs proposed in history
 - XTI (X/Open Transport Interface), a slight modification of the Transport Layer Interface (TLI) developed by AT&T.
- Commonly used: Socket API
 - sometimes called "Berkeley sockets" acknowledging their heritage from Berkeley Unix
 - a socket has a **network-layer host IP address** and a **transport-layer local port number**
 - e.g., email (SMTP) port number 25, web port number 80
 - an application process binds to a socket
 - %netstat or lsof
 - two processes communicate by sending data into socket, reading data out of socket

Socket API



App. and Trans.: App. Protocols and their Transport Protocols

- An application needs to choose the transport protocol

Application	Application layer protocol	Underlying transport protocol
e-mail	smtp [RFC 821]	TCP/SSL
remote terminal access	telnet [RFC 854]	TCP
Web	http [RFC 2068]	TCP/SSL
file transfer	ftp [RFC 959]	TCP
Internet telephony	proprietary (e.g., Vocaltec)	typically UDP
remote file server	NFS	TCP or UDP
streaming multimedia	proprietary	typically UDP but moving to http

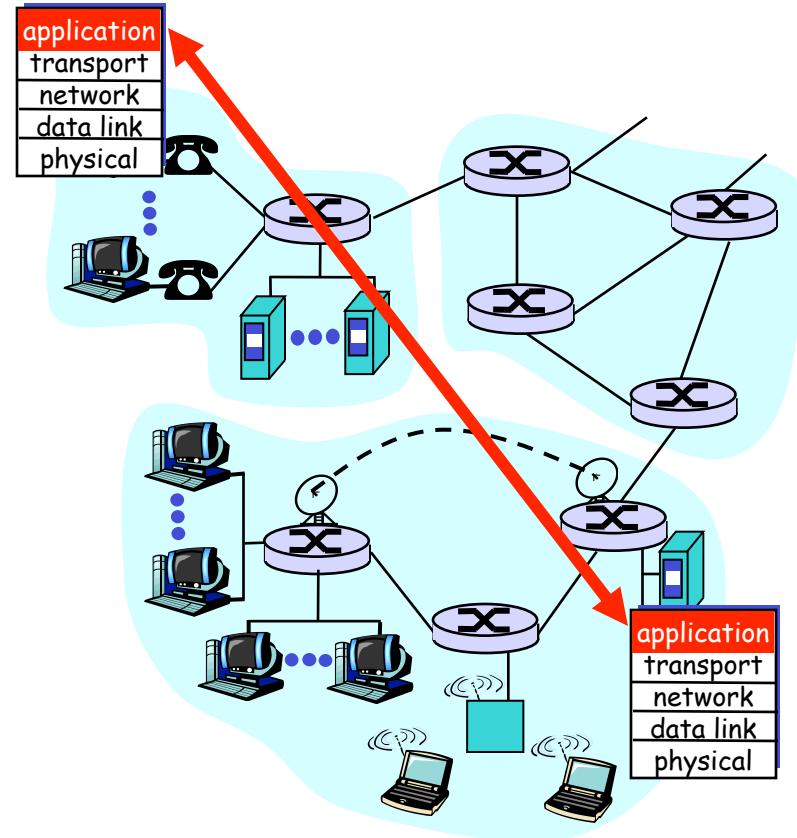
Network Applications vs. Application-layer Protocols

Network application: communicating, distributed processes

- a process is a program that is running within a host
 - a user agent is a process serving as an interface to the user
 - web: browser
 - streaming audio/video: media player
- processes communicate by an application-layer protocol
 - e.g., email, Web

Application-layer protocols

- one “piece” of an app
- define messages exchanged by apps and actions taken
- implementing services by using the service provided by the lower layer, i.e., the transport layer



Client-Server Paradigm

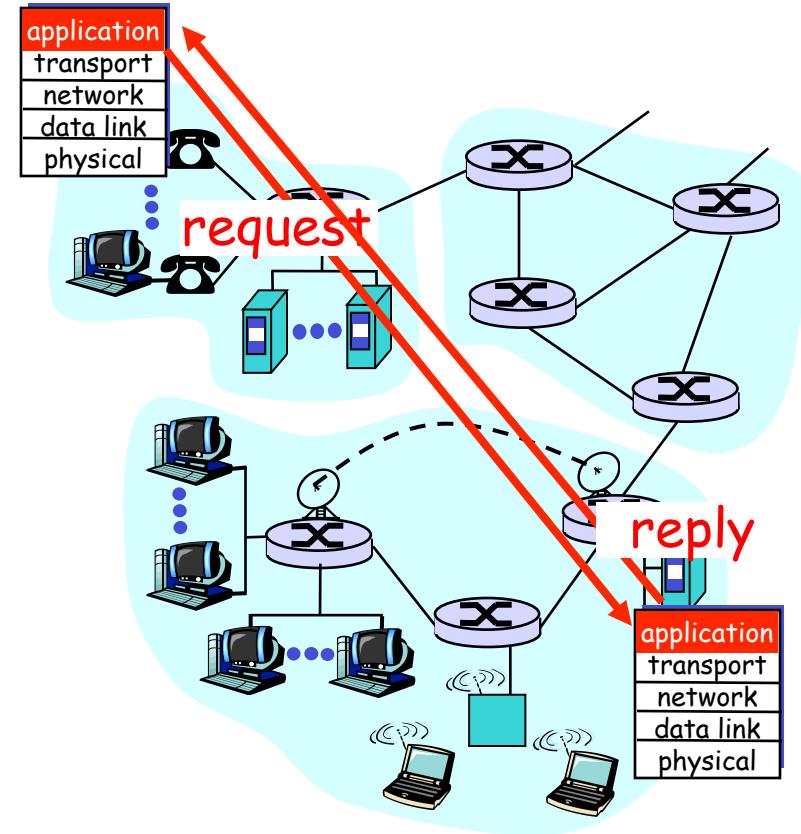
Typical network app has two pieces: *client* and *server*

Client (C):

- initiates contact with server (“speaks first”)
- typically requests service from server
- for Web, client is implemented in browser; for e-mail, in mail reader

Server (S):

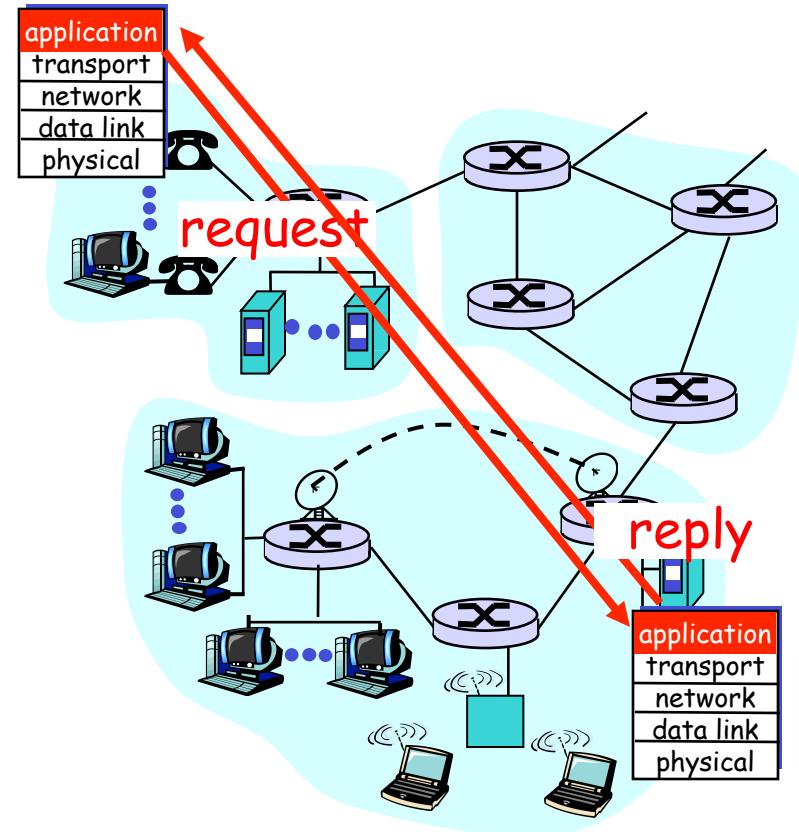
- provides requested service to client
- e.g., Web server sends requested Web page; mail server delivers e-mail



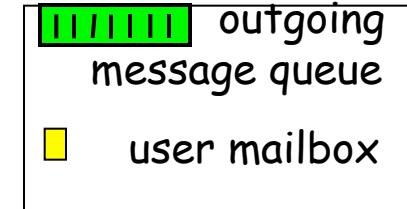
Client-Server Paradigm: Key Questions

Key questions to ask about a C-S application

- Is the application **extensible**?
- Is the application **scalable**?
- How does the application handle server failures (being **robust**)?
- How does the application provide **security**?



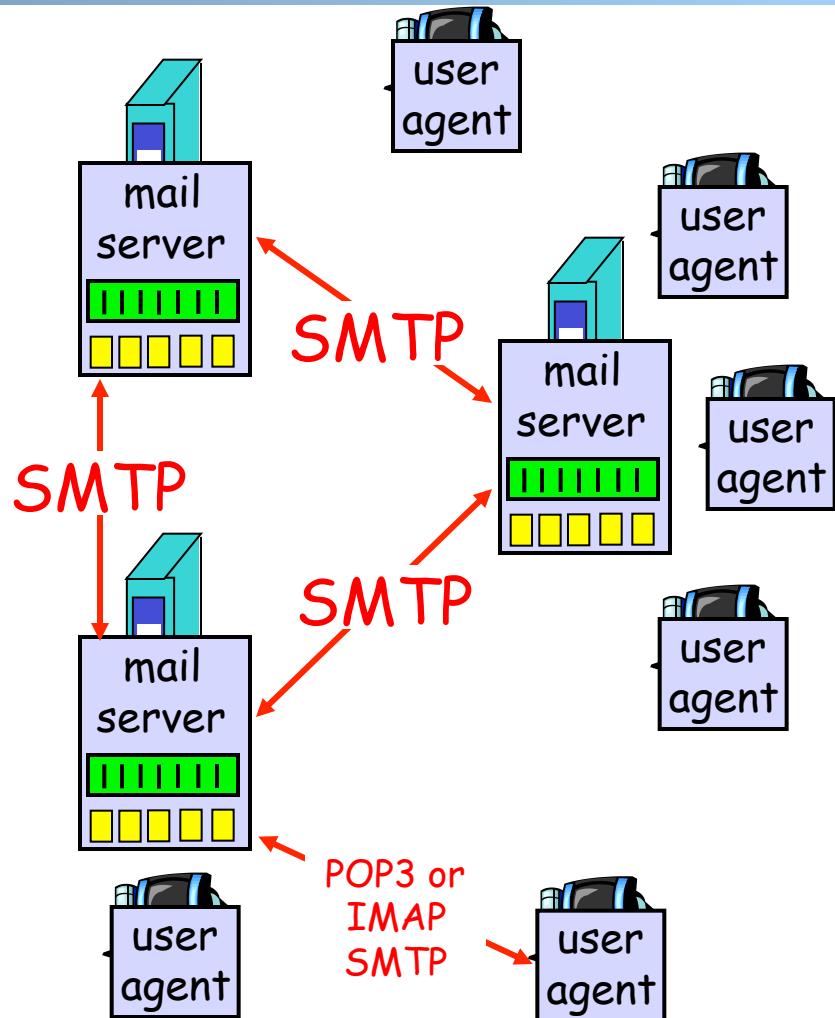
Electronic Mail



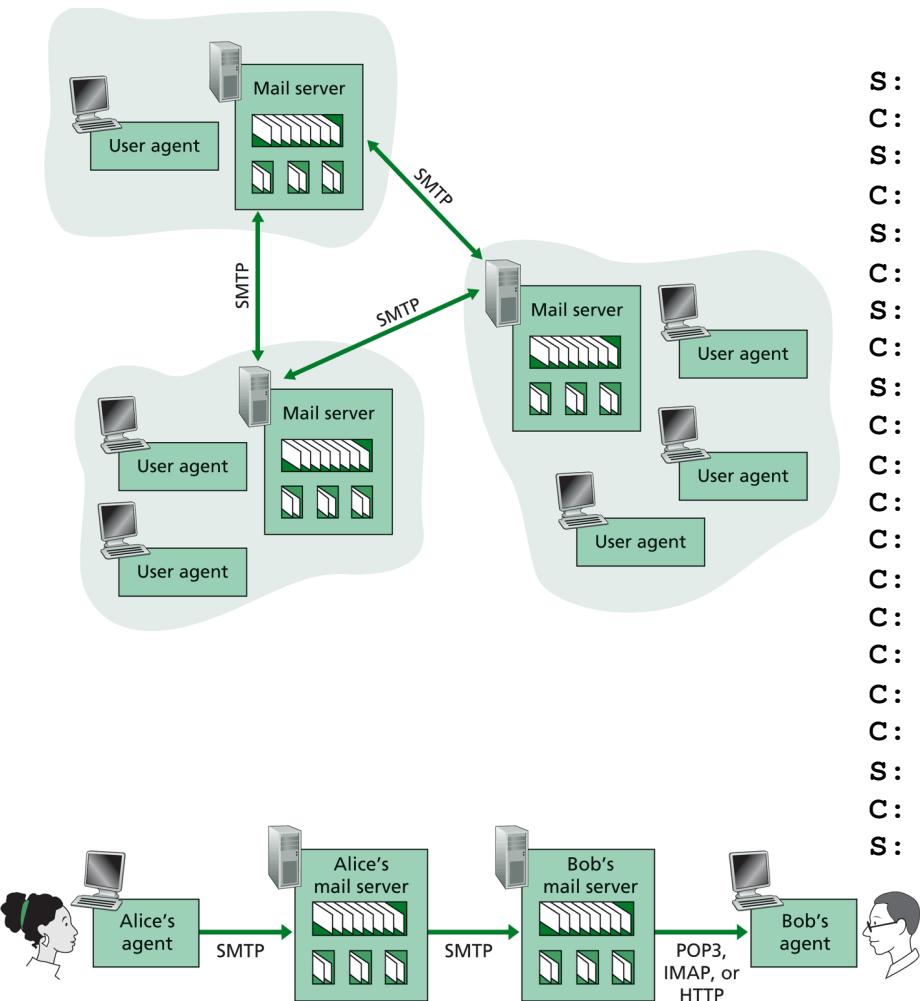
- Still active
 - 80B emails/day
 - 3.9B active email boxes

Three major components:

- User agents
- Mail servers
- Protocols
 - Outgoing email
 - SMTP
 - Retrieving email
 - POP3: Post Office Protocol [RFC 1939]
 - IMAP: Internet Mail Access Protocol [RFC 1730]



SMTP: Outgoing Email as a Client-Server Application



```
S: 220 mr1.its.yale.edu
C: HELO cyndra.yale.edu
S: 250 Hello cyndra.cs.yale.edu, pleased to meet you
C: MAIL FROM: <spoof@cs.yale.edu>
S: 250 spoof@cs.yale.edu... Sender ok
C: RCPT TO: <yry@yale.edu>
S: 250 yry@yale.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Date: Wed, 23 Jan 2008 11:20:27 -0500 (EST)
C: From: "Y. R. Yang" <yry@cs.yale.edu>
C: To: "Y. R. Yang" <yry@cs.yale.edu>
C: Subject: This is subject
C:
C: This is the message body!
C: Please don't spoof!
C:
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 mr1.its.yale.edu closing connection
```

Email Transport Architecture

MUA: User Agent

Mediator: User-level Relay

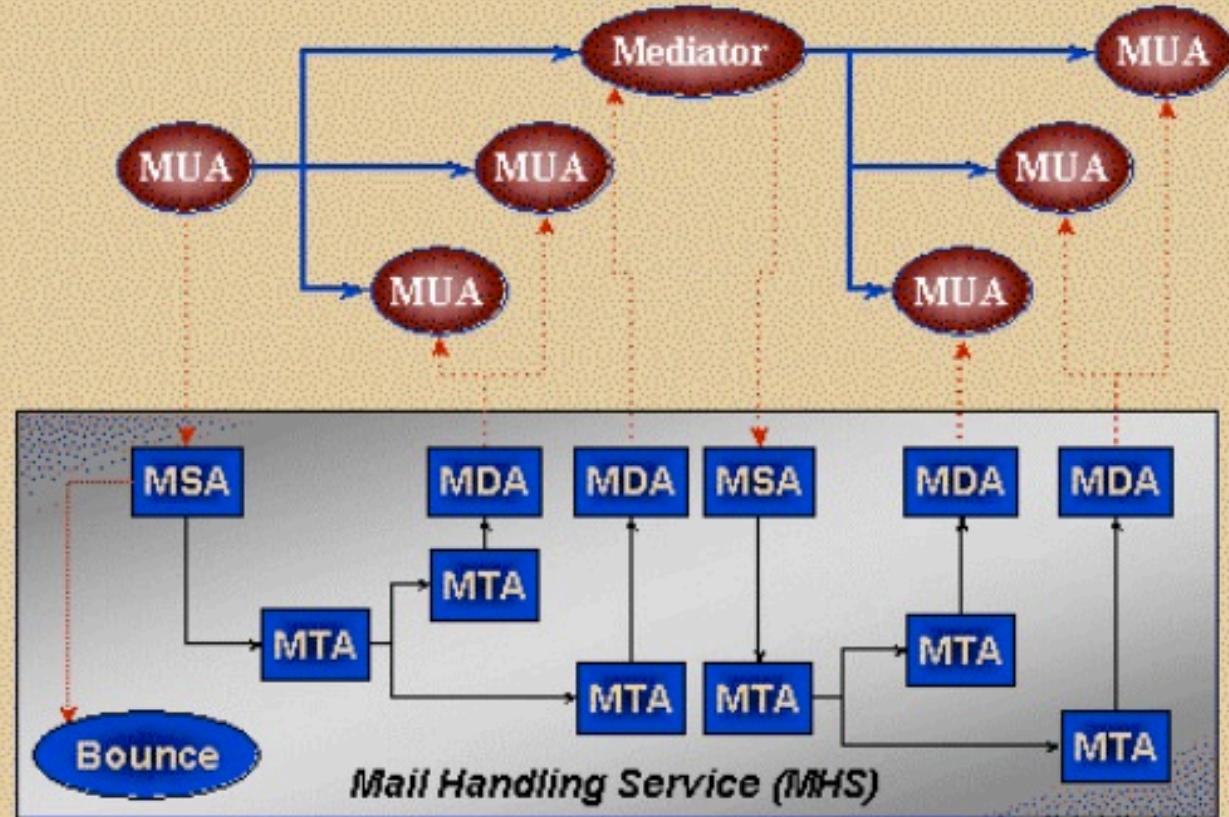
MHS: Mail Handling (transit) Service

MSA: Submission

MTA: Transfer

MDA: Delivery

Bounce: Returns

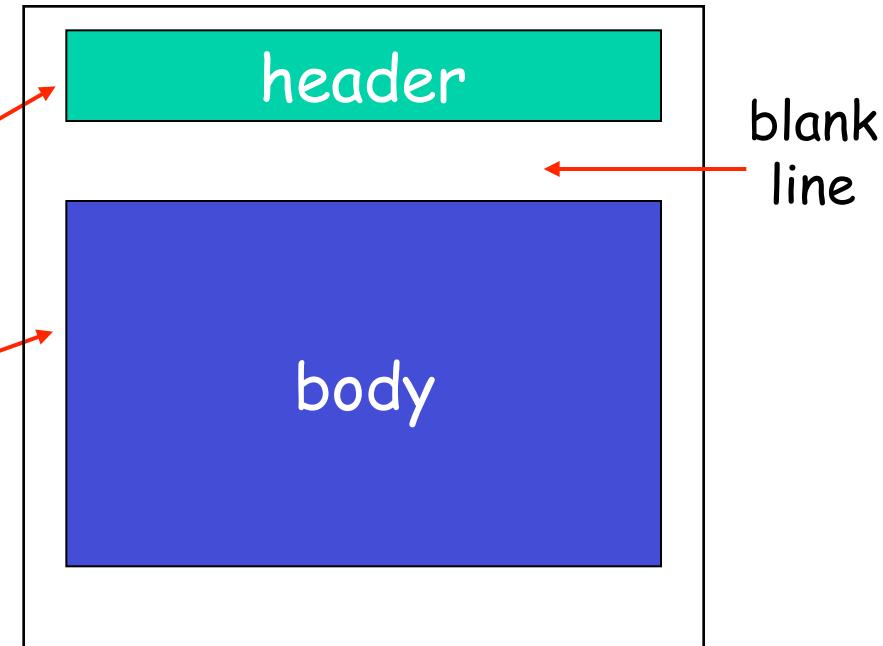


Mail Message Data Format

SMTP: protocol for
exchanging email msgs

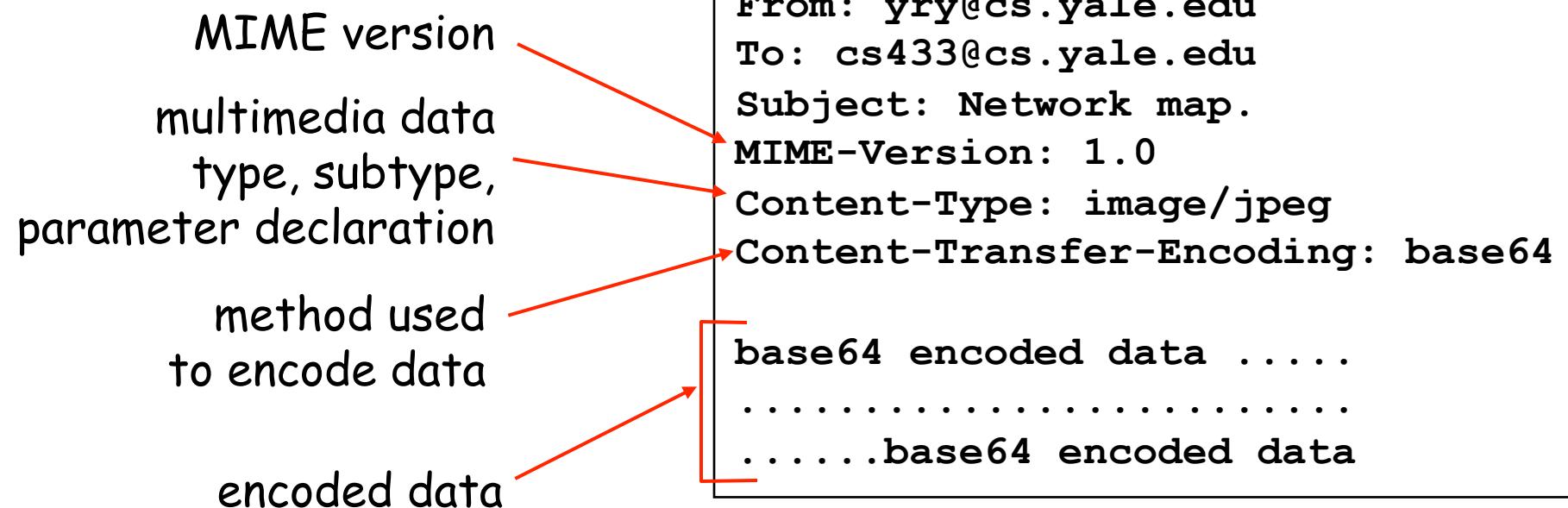
RFC 822: standard for text
message format:

- Header lines, e.g.,
 - To:
 - From:
 - Subject:
- Body
 - the “message”, ASCII
characters only (any
problem?)



Message Format: Multimedia Extensions

- ❑ MIME: multimedia mail extension, RFC 2045, 2056
- ❑ Additional lines in msg header declare MIME content type



Multipart Type: How Attachment Works

From: yry@cs.yale.edu
To: cs433@cs.yale.edu
Subject: Network map.
MIME-Version: 1.0

Content-Type: multipart/mixed; boundary=98766789

--98766789

Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain

Hi,

Attached is network topology map.

--98766789

Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data

.....

.....base64 encoded data

--98766789--

Design Review

```
S: 220 mr1.its.yale.edu
C: HELO cyndra.yale.edu
S: 250 Hello cyndra.cs.yale.edu, pleased to meet you
C: MAIL FROM: <spoof@cs.yale.edu>
S: 250 spoof@cs.yale.edu... Sender ok
C: RCPT TO: <yry@yale.edu>
S: 250 yry@yale.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: From: yry@cs.yale.edu
C: To: cs433@cs.yale.edu
C: Subject: Network map.
C: MIME-Version: 1.0
C: Content-Type: image/jpeg
C: Content-Transfer-Encoding: base64
C:
C: base64 encoded data .....
C: .....
C: .....base64 encoded data
C:
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 mr1.its.yale.edu closing connection
```

Why not make the
msg headers smtp
headers?

POP3 Protocol: Mail Retrieval

Authorization phase

- ❑ client commands:
 - **user**: declare username
 - **pass**: password
- ❑ server responses
 - +OK
 - -ERR

```
S: +OK POP3 server ready
C: user alice
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

Transaction phase, client:

- ❑ **list**: list message numbers
- ❑ **retr**: retrieve message by number
- ❑ **dele**: delete
- ❑ **quit**

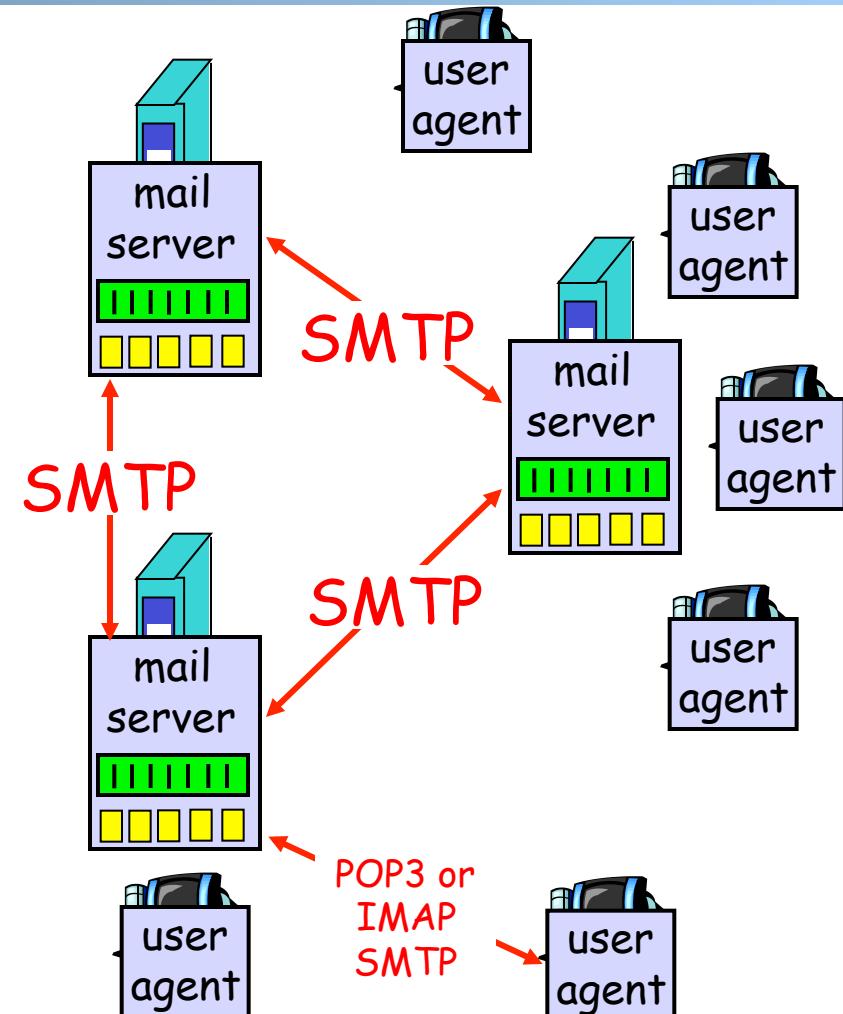
```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

```
%openssl s_client -connect pop.gmail.com:995
```

Evaluation of SMTP/POP/IMAP

Key questions to ask about a C-S application

- extensible?
- scalable?
- robust?
- security?



Email: Positive

- Some nice design features we can learn from the design of the email
 - separate protocols for different functions
 - email retrieval (e.g., POP3, IMAP)
 - email transmission (SMTP)
 - simple/basic requests to implement basic control; fine-grain control through ASCII header and message body
 - make the protocol easy to read/debug/extend (analogy with end-to-end layered design?)
 - status code in response makes message easy to parse

Email: Challenge

□ Spam (Google)

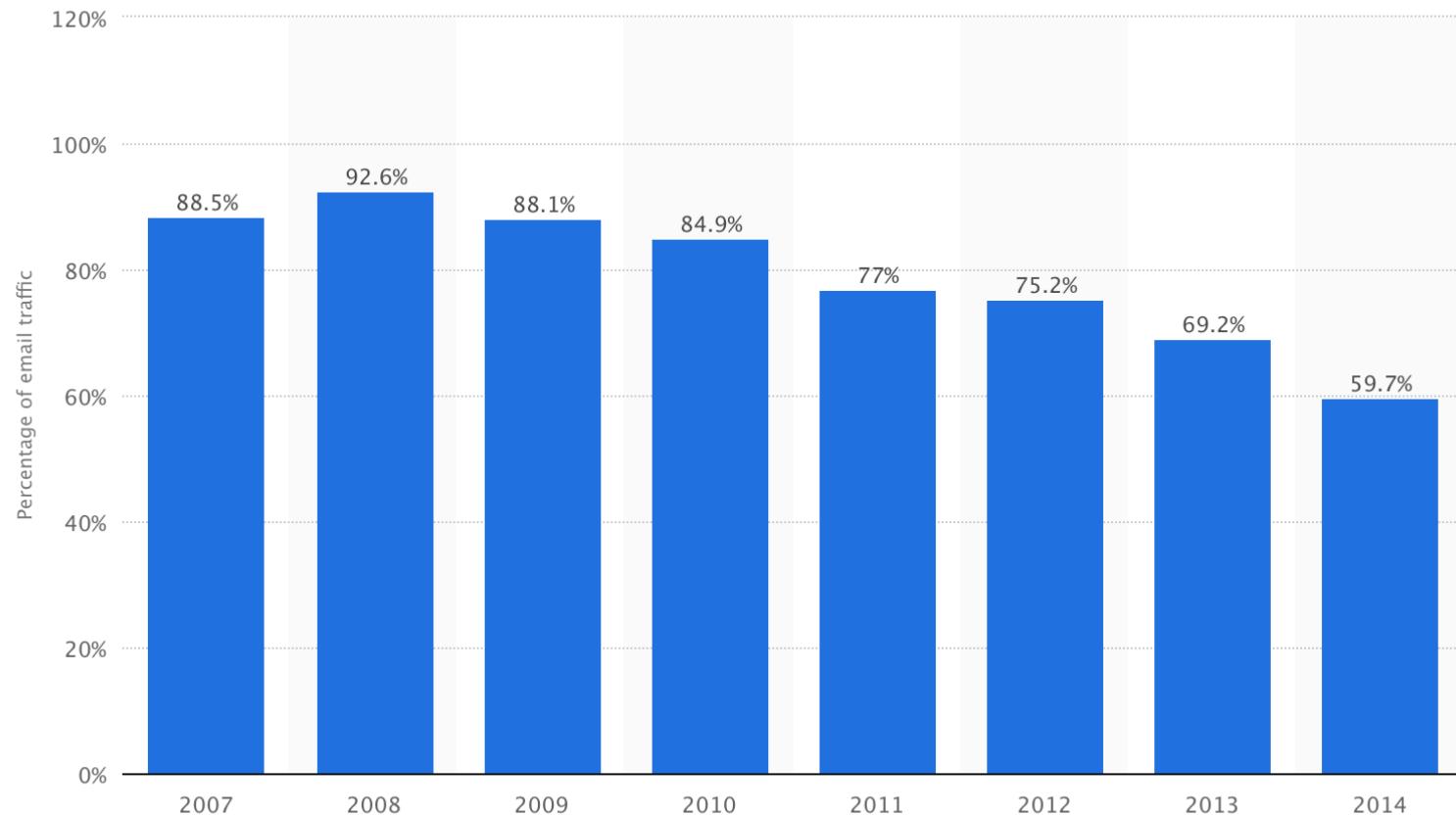


<https://mail.google.com/intl/en/mail/help/fightspam/spamexplained.html>

Email: Challenge

- A large percentage of spam/phish

Global spam volume as percentage of total e-mail traffic from 2007 to 2014

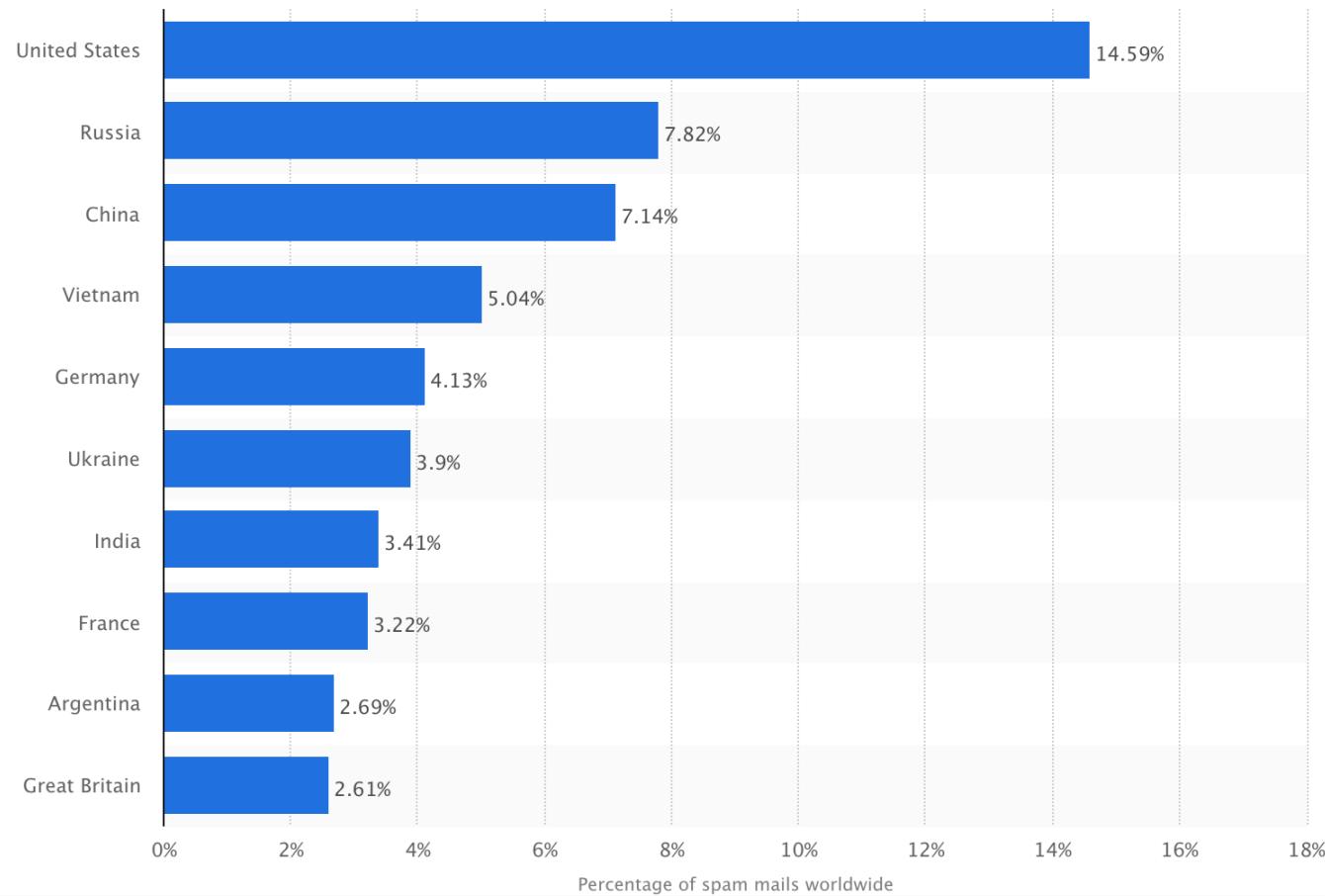


Source: <http://www.statista.com/statistics/420400/spam-email-traffic-share-annual/>

© Statista 2016

Email: Challenge

Leading countries of origin for unsolicited spam emails as of 2nd quarter 2015, by share of worldwide spam volume



Source: <http://www.statista.com/statistics/263086/countries-of-origin-of-spam/>

Discussion: How May Email Spams Be Detected?

Detection Methods Used by GMail

- Known phishing scams
- Message from unconfirmed sender identity
- Message you sent to Spam/similarity to suspicious messages
- Administrator-set policies
- Empty message content

<https://support.google.com/mail/answer/1366858?hl=en>

Network Applications: DNS

Y. Richard Yang

<http://zoo.cs.yale.edu/classes/cs433/>

2/1/2016

Outline

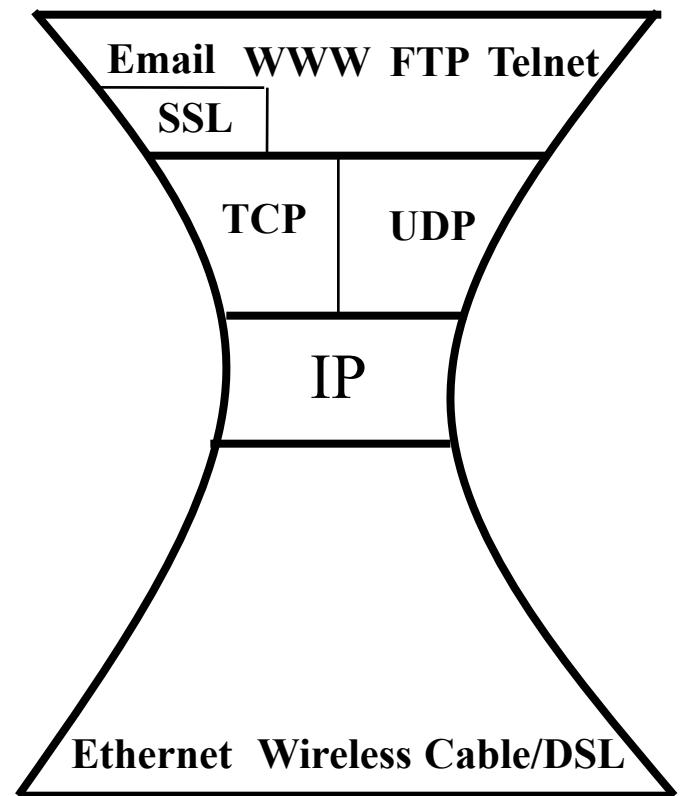
- Admin and recap
- DNS

Admin

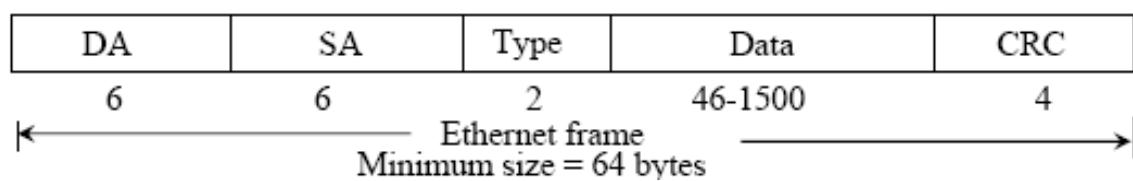
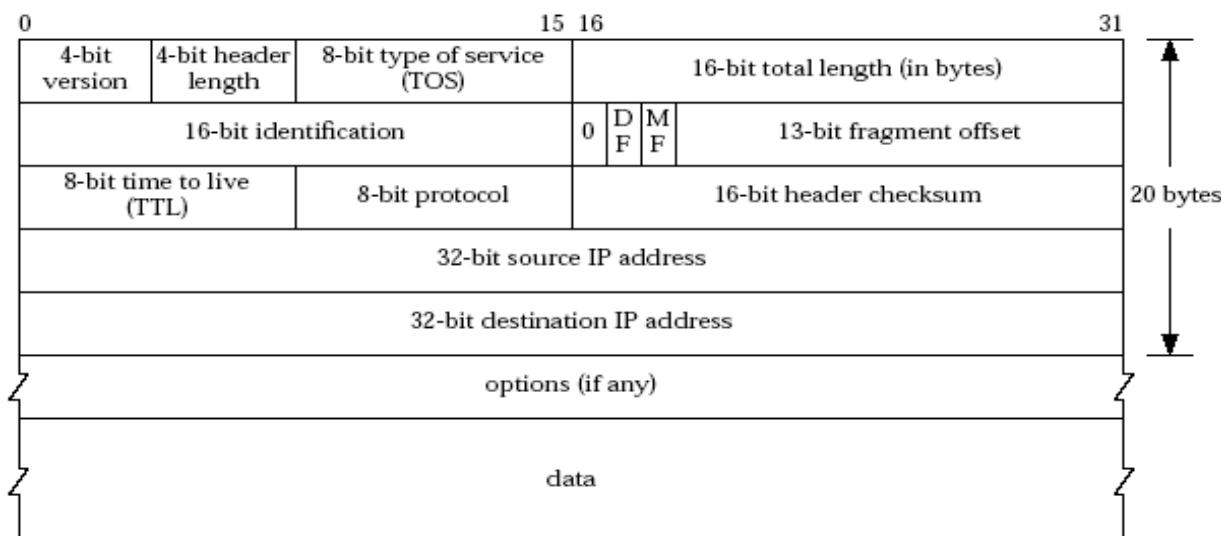
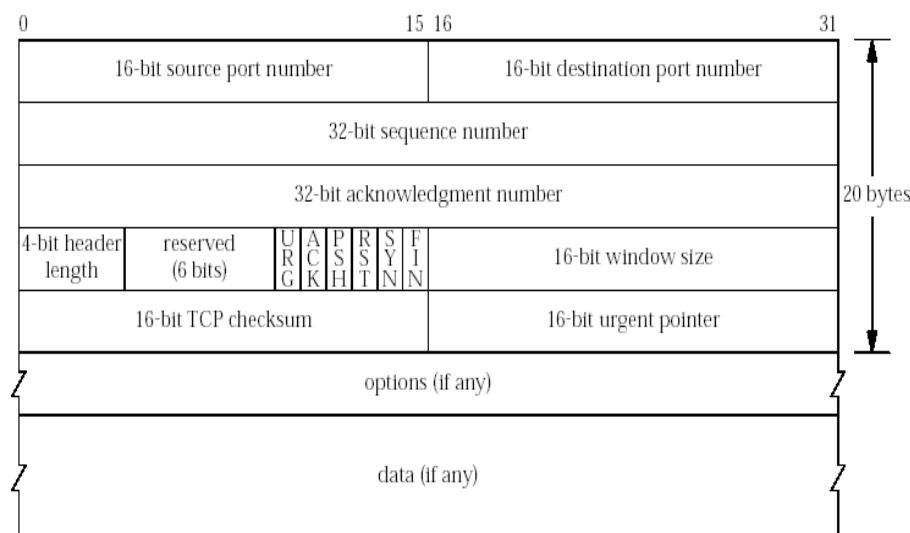
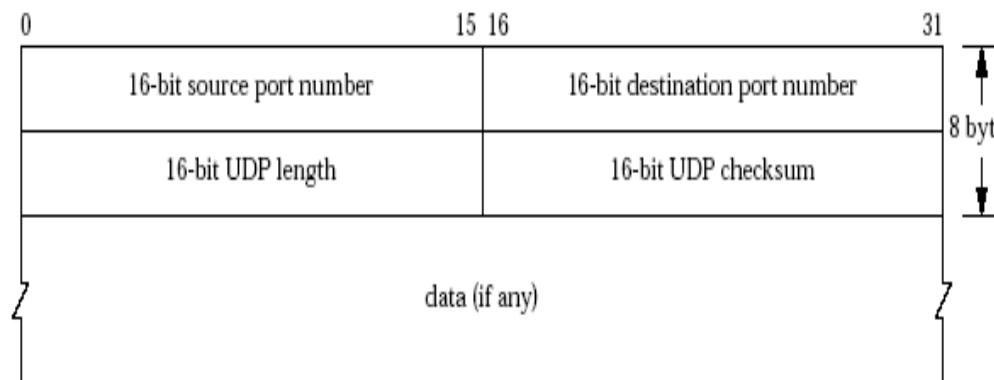
- 72 discretionary late hours for assignments across the semester

Recap: The Big Picture of the Internet

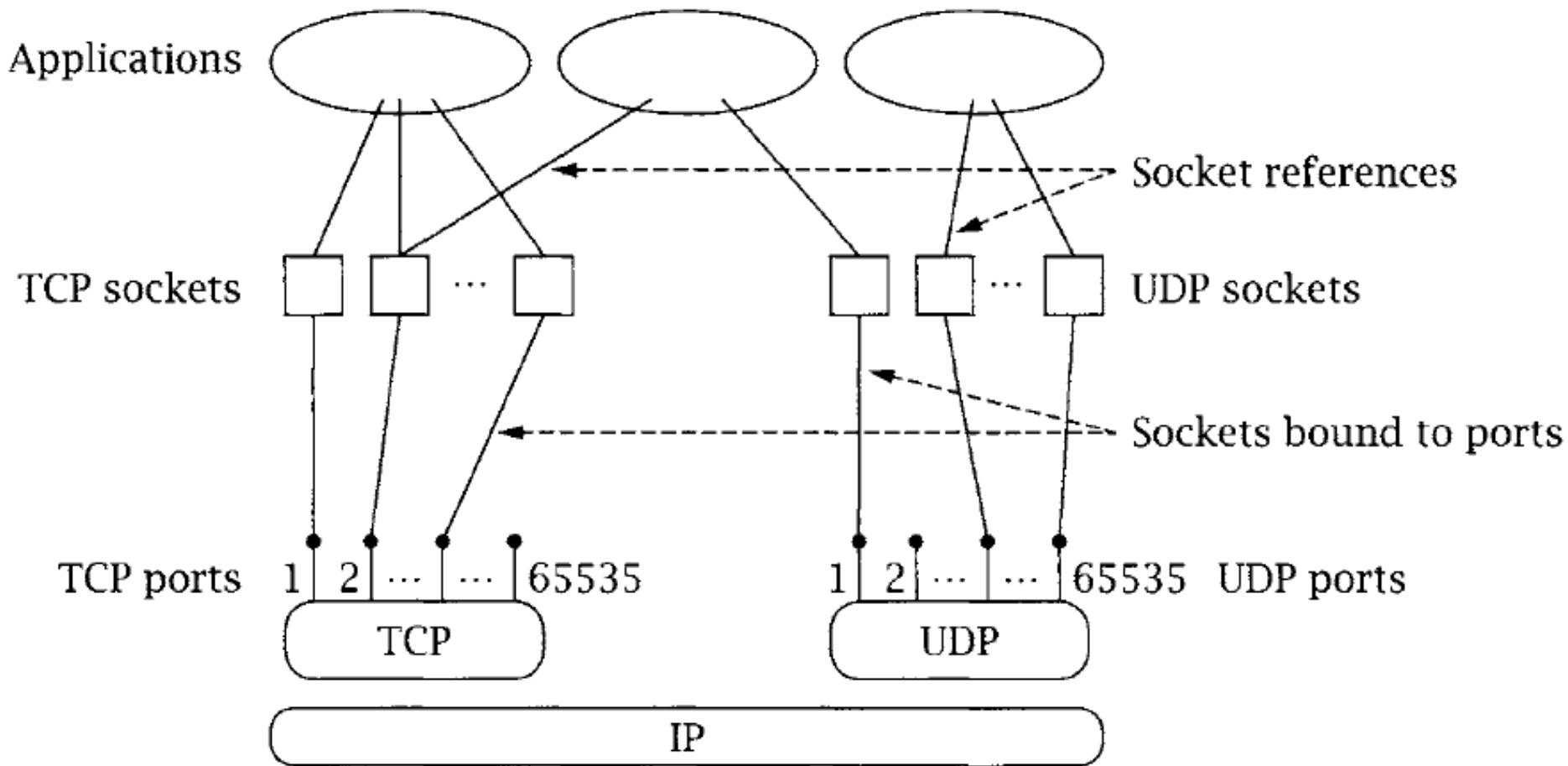
- Hosts and routers:
 - ~ 1 bill. hosts (2015)
 - organized into ~50K networks
 - backbone links 100 Gbps
- Software:
 - datagram switching with virtual circuit support
 - layered network architecture
 - use end-to-end arguments to determine the services provided by each layer
 - the hourglass architecture of the Internet



Protocol Formats



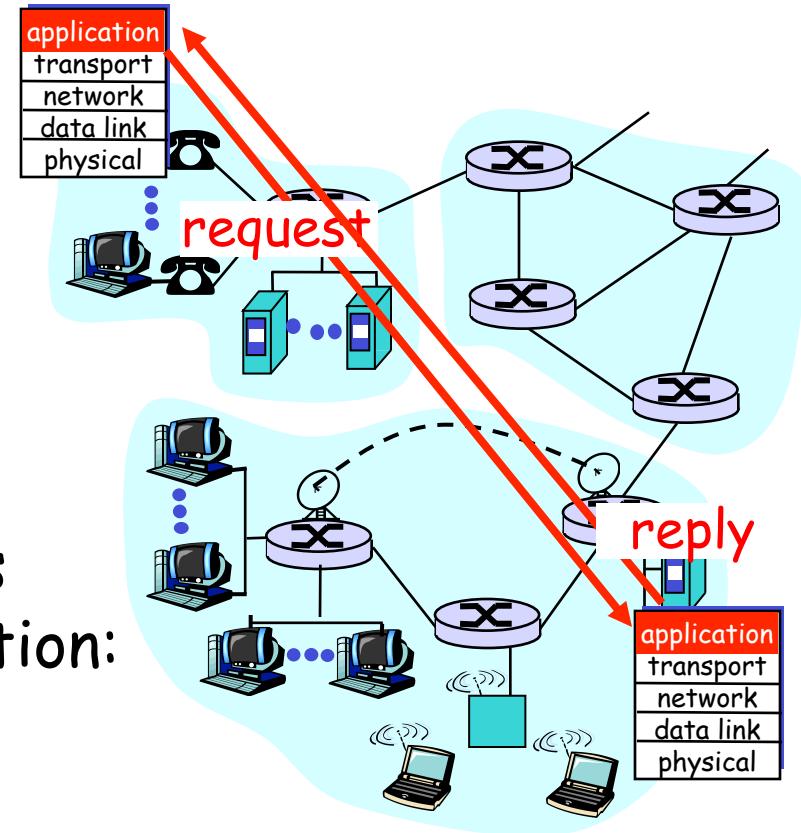
Multiplexing/Demultiplexing



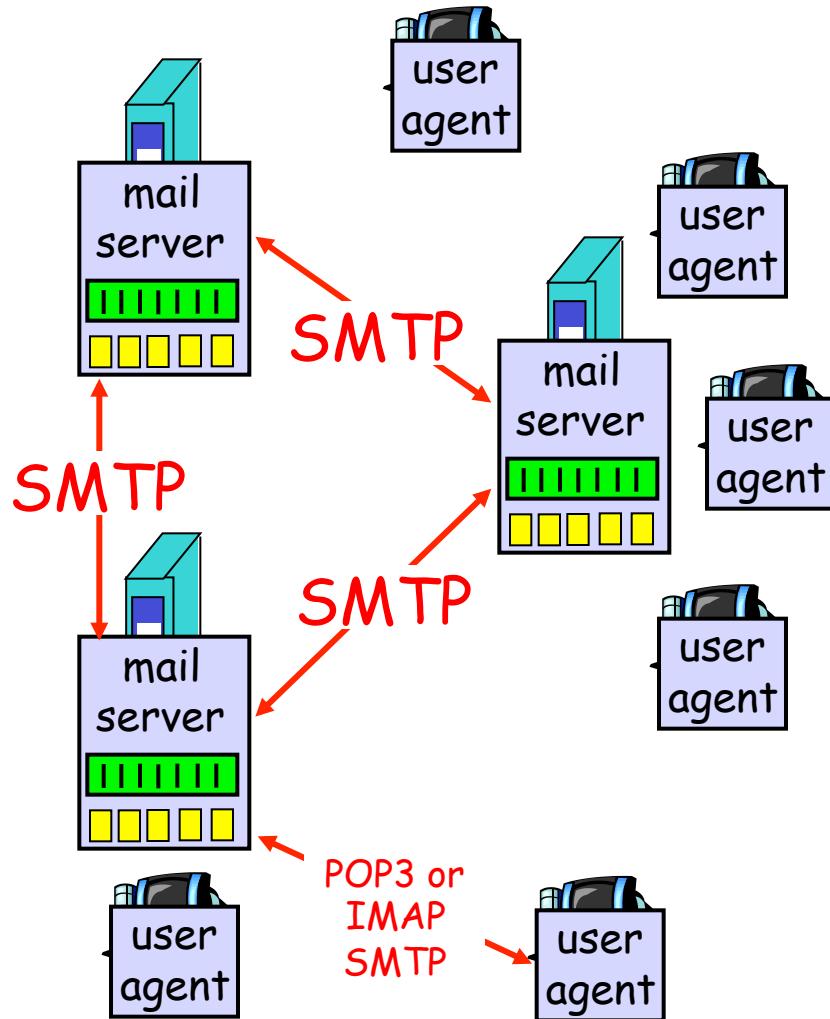
Recap: Client-Server Paradigm

- ❑ The basic paradigm of network applications is the client-server (C-S) paradigm

- ❑ Some key design questions to ask about a C-S application:
 - extensibility
 - scalability
 - robustness
 - security



Recap: Email App



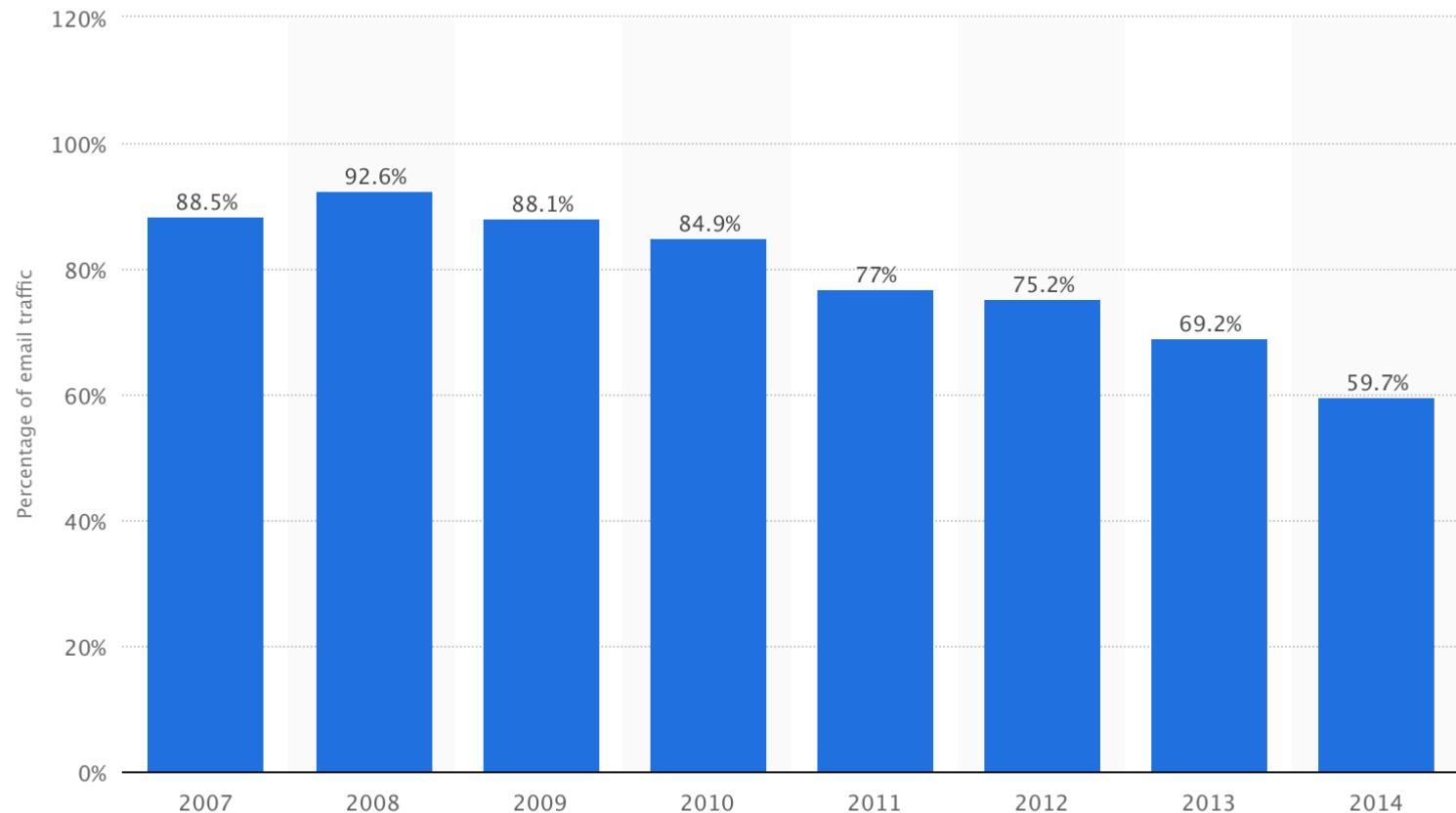
Some nice **protocol extensibility** design features

- separate protocols for different functions
- simple/basic (smtp) requests to implement basic control; fine-grain control through ASCII header and message body
- status code in response makes message easy to parse

Email: Challenge

- A large percentage of spam/phish

Global spam volume as percentage of total e-mail traffic from 2007 to 2014



Source: <http://www.statista.com/statistics/420400/spam-email-traffic-share-annual/>

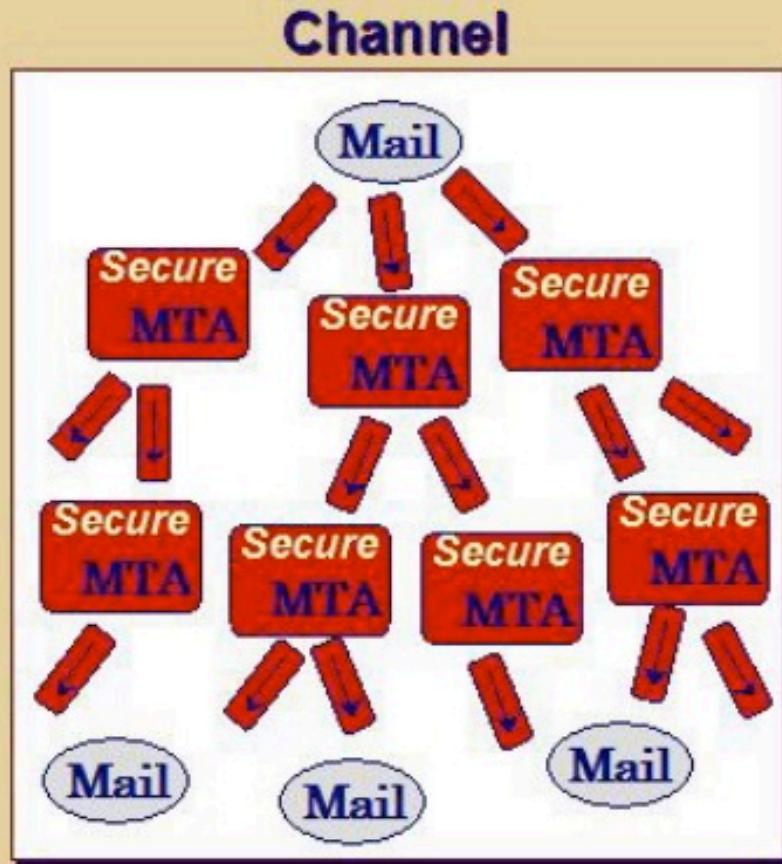
© Statista 2016

Recap: Spam Detection Methods by GMail

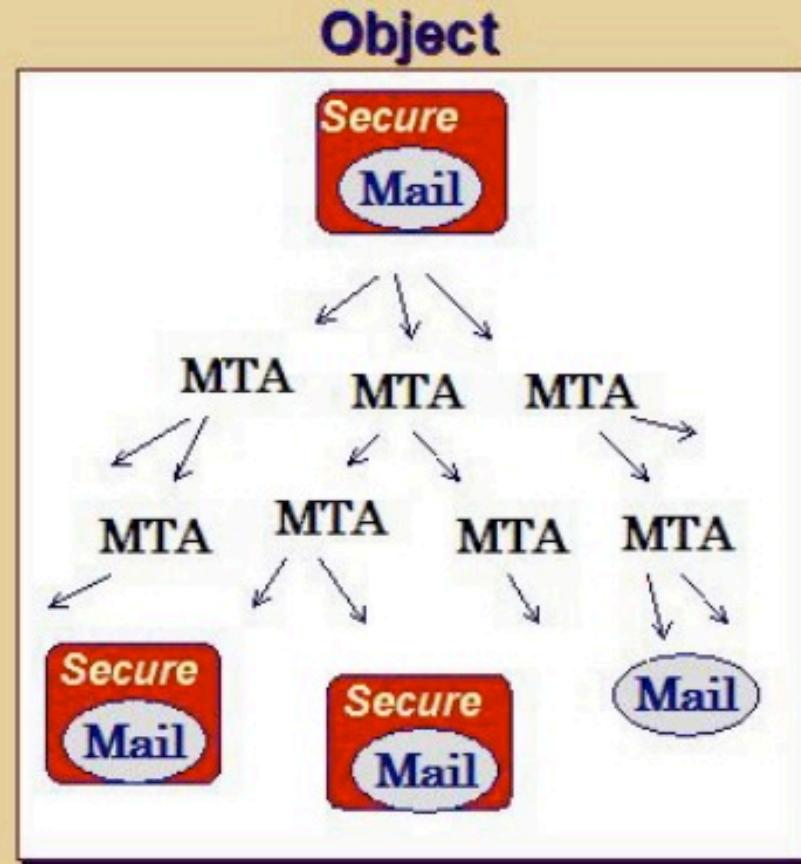
- Known phishing scams
- Message from unconfirmed sender identity
- Message you sent to Spam/similarity to suspicious messages
- Administrator-set policies
- Empty message content

<https://support.google.com/mail/answer/1366858?hl=en>

Current Email Authentication Approaches

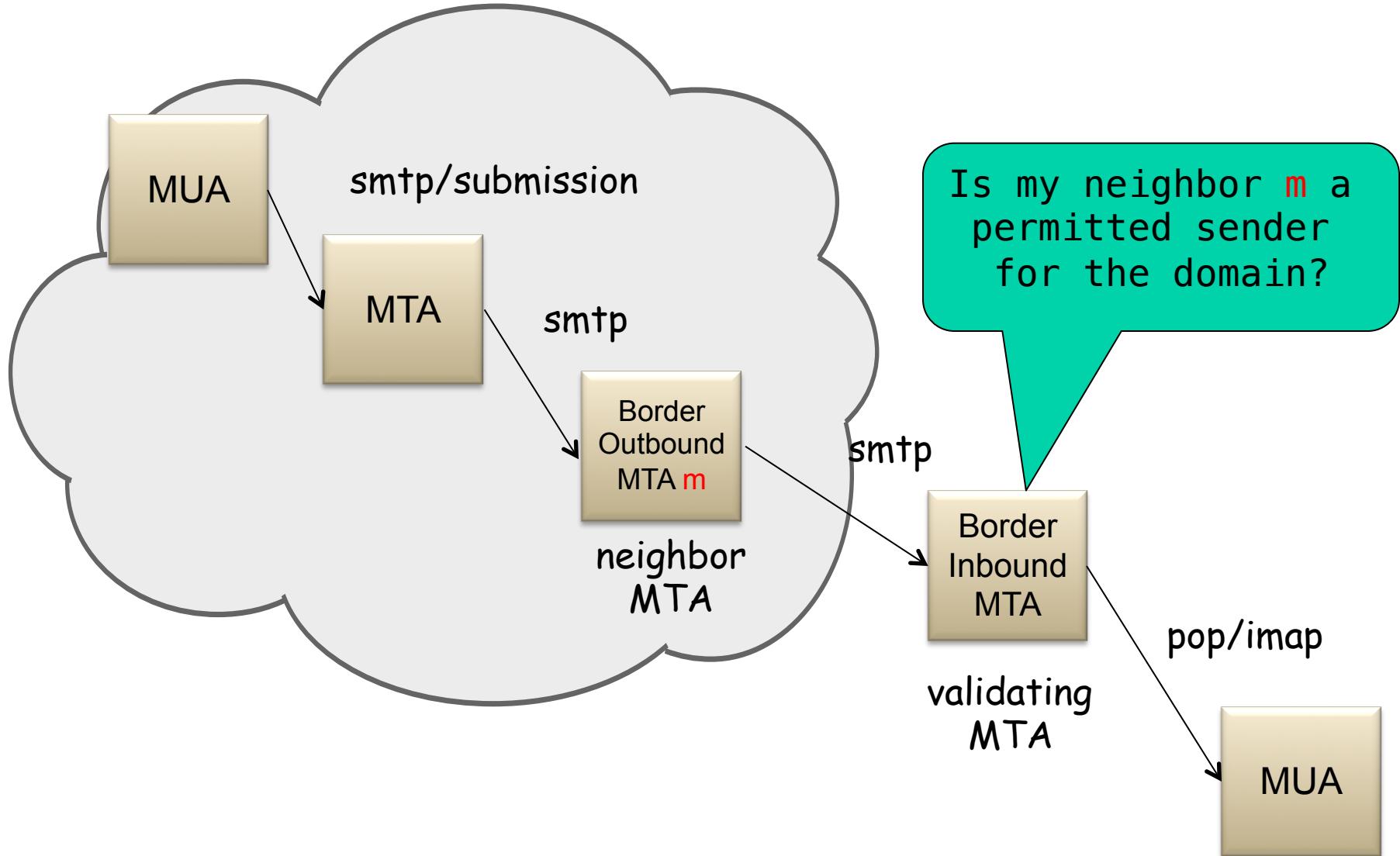


Sender Policy Frame (SPF)



DomainKeys Identified Mail (DKIM)

Sender Policy Framework (SPF RFC7208)



SPF Exercise

- Test 1

- Send real email by gmail
- POP retr

- Test 2

- Send using telnet
- POP retr

Key Remaining Question for SPF?

- How does SPF know if its neighbor MTA is a permitted sender of the domain?

DomainKeys Identified Mail (DKIM; RFC 5585)

- A domain-level digital signature authentication framework for email, using public key crypto
 - E.g., gmail.com signs that the message is sent by gmail server
- Basic idea of public key signature
 - Owner has both public and private keys
 - Owner uses private key to sign a message to generate a signature
 - Others with public key can verify signature

Example: RSA

1. Choose two large prime numbers p, q .
(e.g., 1024 bits each)
2. Compute $n = pq$, $z = (p-1)(q-1)$
3. Choose e (with $e < n$) that has no common factors with z . (e, z are "relatively prime").
4. Choose d such that $ed-1$ is exactly divisible by z .
(in other words: $ed \bmod z = 1$).
5. Public key is (n, e) . Private key is (n, d) .

RSA: Signing/Verification

0. Given (n,e) and (n,d) as computed above
1. To sign message, m , compute $h = \text{hash}(m)$, then sign with private key

$s = h^d \text{ mod } n$ (i.e., remainder when h^d is divided by n)

2. To verify signature s , compute

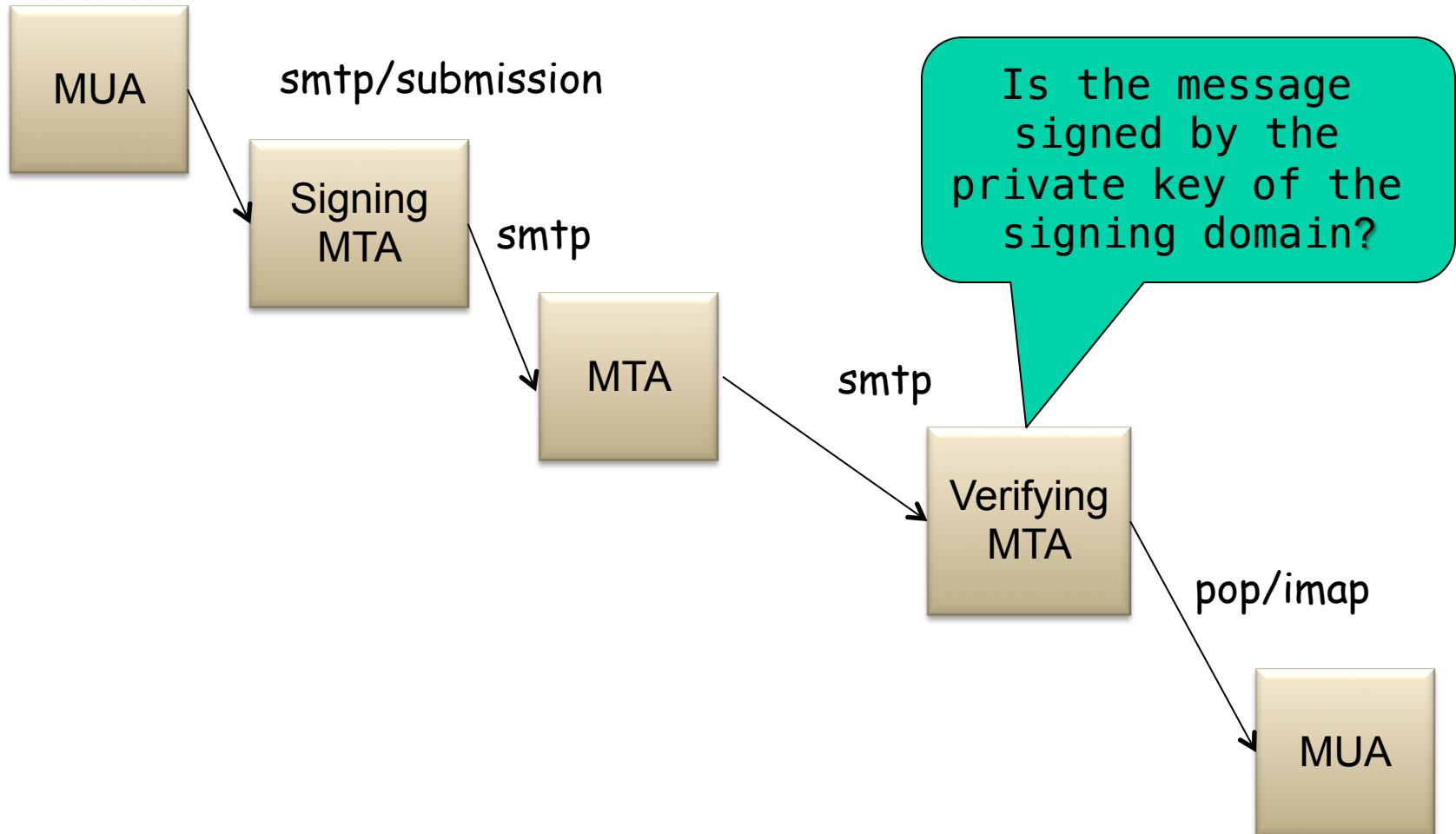
$h' = s^e \text{ mod } n$ (i.e., remainder when s^e is divided by n)

Magic happens!

$$h = (h^d \text{ mod } n)^e \text{ mod } n$$

The magic is a simple application of Euler's generalization of Fermat's little theorem

DomainKeys Identified Mail (DKIM)



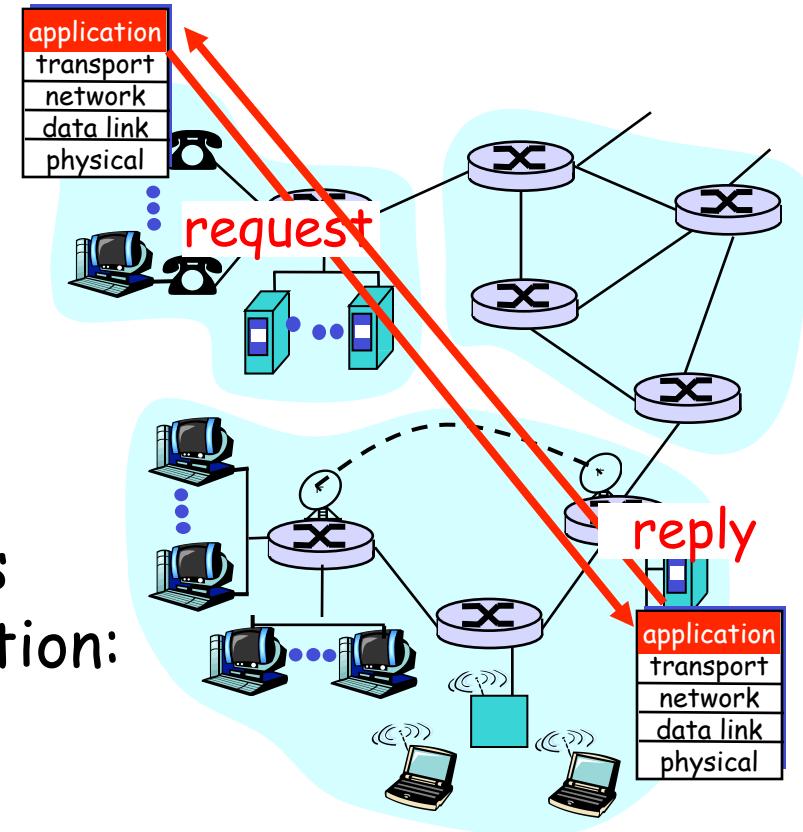
Key Remaining Question about DKIM?

- How does DKIM retrieve the public key of the author domain?

Summary: Client-Server Paradigm

- ❑ The basic paradigm of network applications is the client-server (C-S) paradigm

- ❑ Some key design questions to ask about a C-S application:
 - ✓ extensibility
 - scalability
 - robustness
 - ✓ security

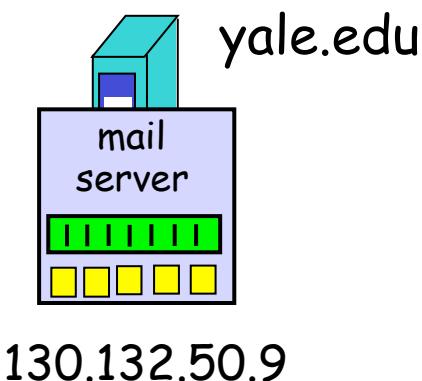
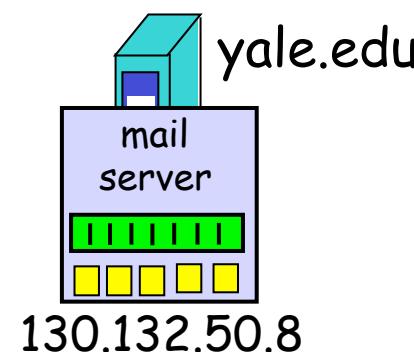
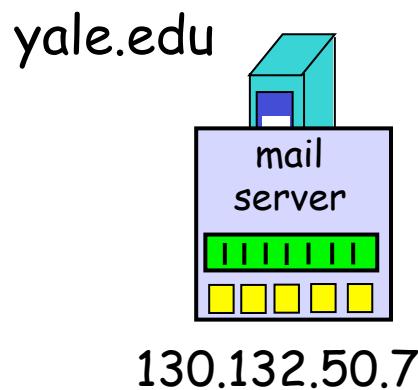


Scalability/R robustness

- High scalability and robustness fundamentally require that multiple email servers serve the same email address

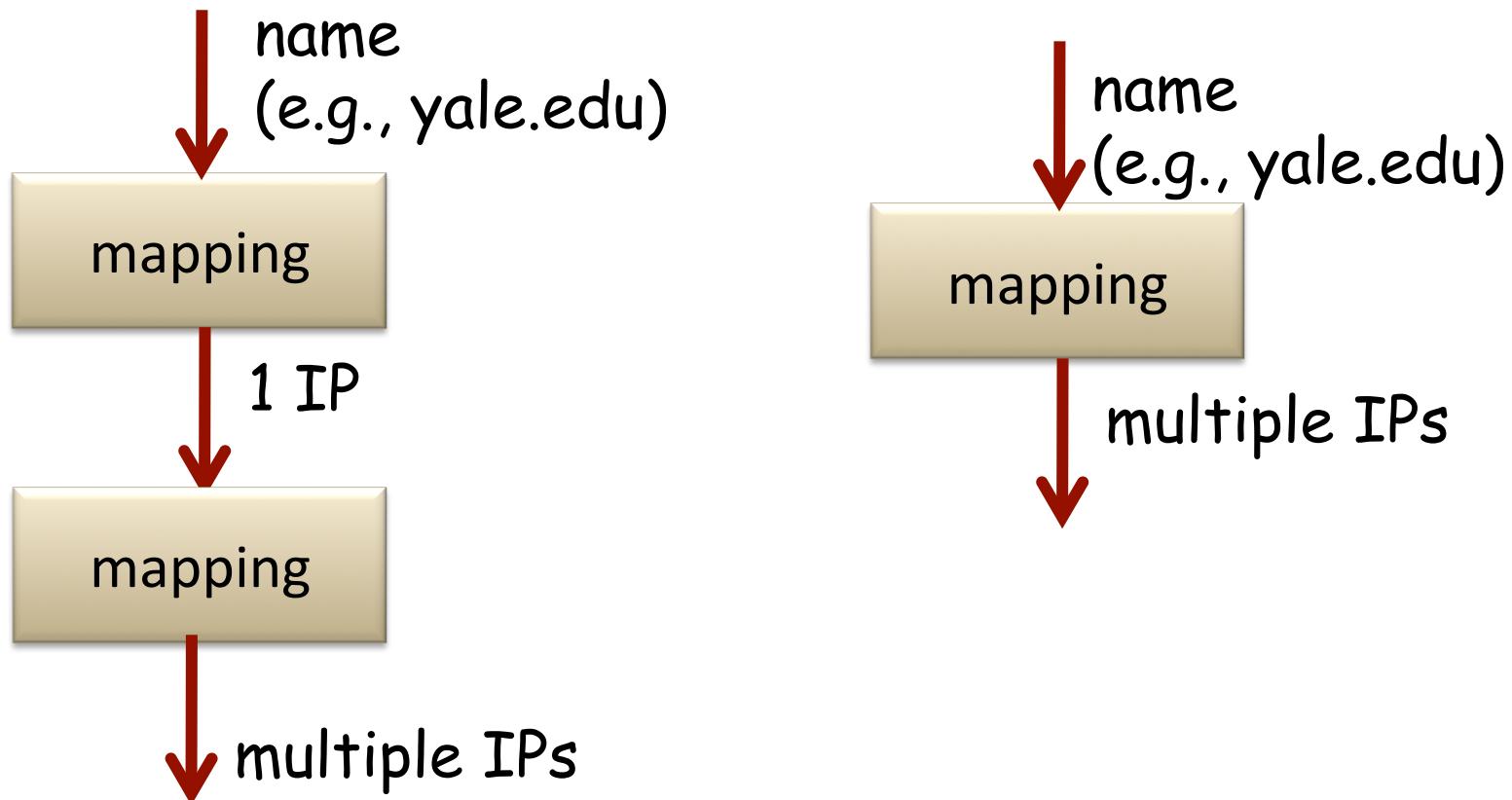


mapping

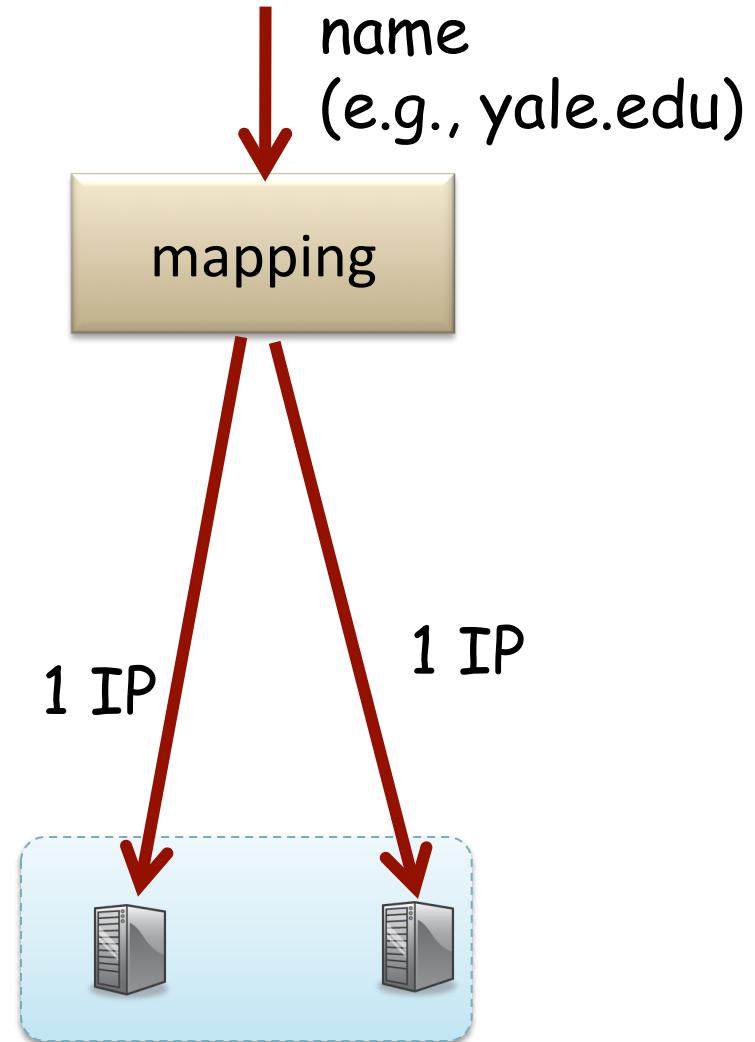
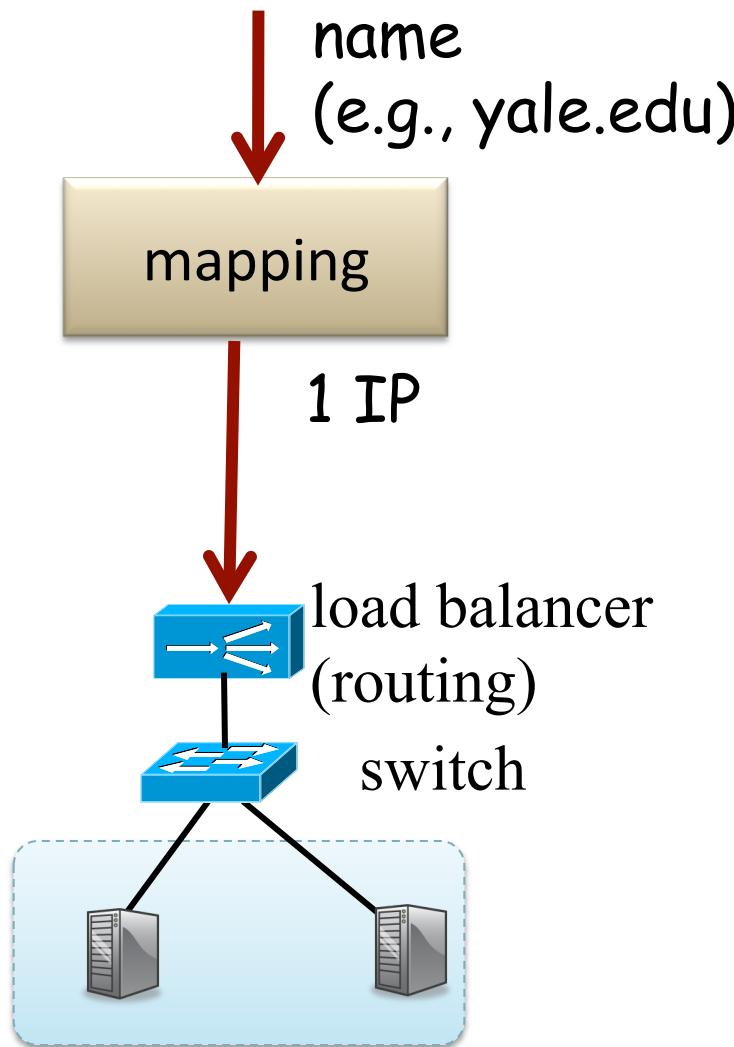


Mapping Functions Design Alternatives

- Map from an email address server name to IP address of email server



Mapping Functions Design Alternatives



Summary: Some Key Remaining Issues about Email

- Basic: How to find the email server of a domain?
- Scalability/robustness: how to find multiple servers for the email domain?
- Security
 - SPF: How does SPF know if its neighbor MTA is a permitted sender of the domain?
 - DKIM: How does DKIM retrieve the public key of the author domain?

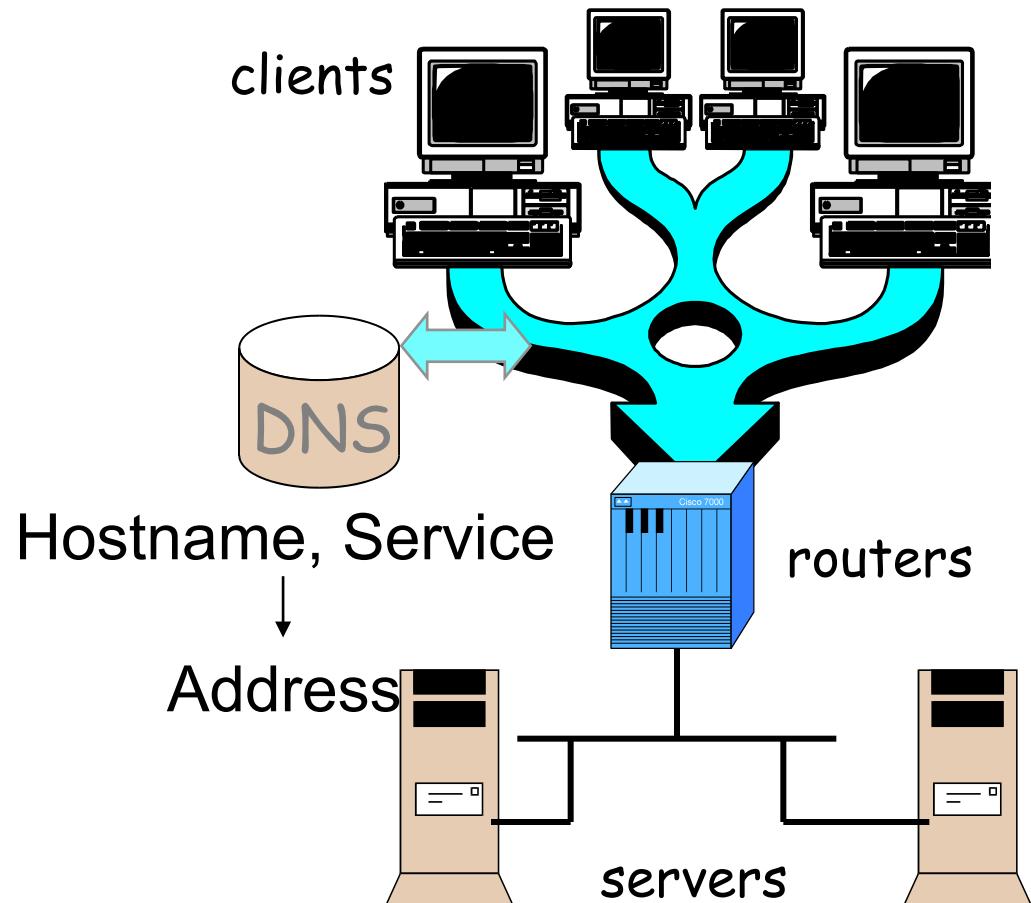
Outline

- Recap
- Email security (authentication)
- DNS

DNS: Domain Name System

□ Function

- map between (domain name, service) to value, e.g.,
 - (www.cs.yale.edu, Addr)
→ 128.36.229.30
 - (cs.yale.edu, Email)
→ netra.cs.yale.edu



DNS Records

<http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml>

DNS: stores resource records (**RR**)

RR format: (**name**, **type**, **value**, **ttl**)

- Type=A
 - name** is hostname
 - value** is IP address
- Type=NS
 - name** is domain (e.g. yale.edu)
 - value** is the name of the authoritative name server for this domain
- Type=TXT
 - general txt
- Type=CNAME
 - name** is an alias name for some “canonical” (the real) name
 - value** is canonical name
- Type=MX
 - value** is hostname of mail server associated with **name**
- Type=SRV
 - general extension for services

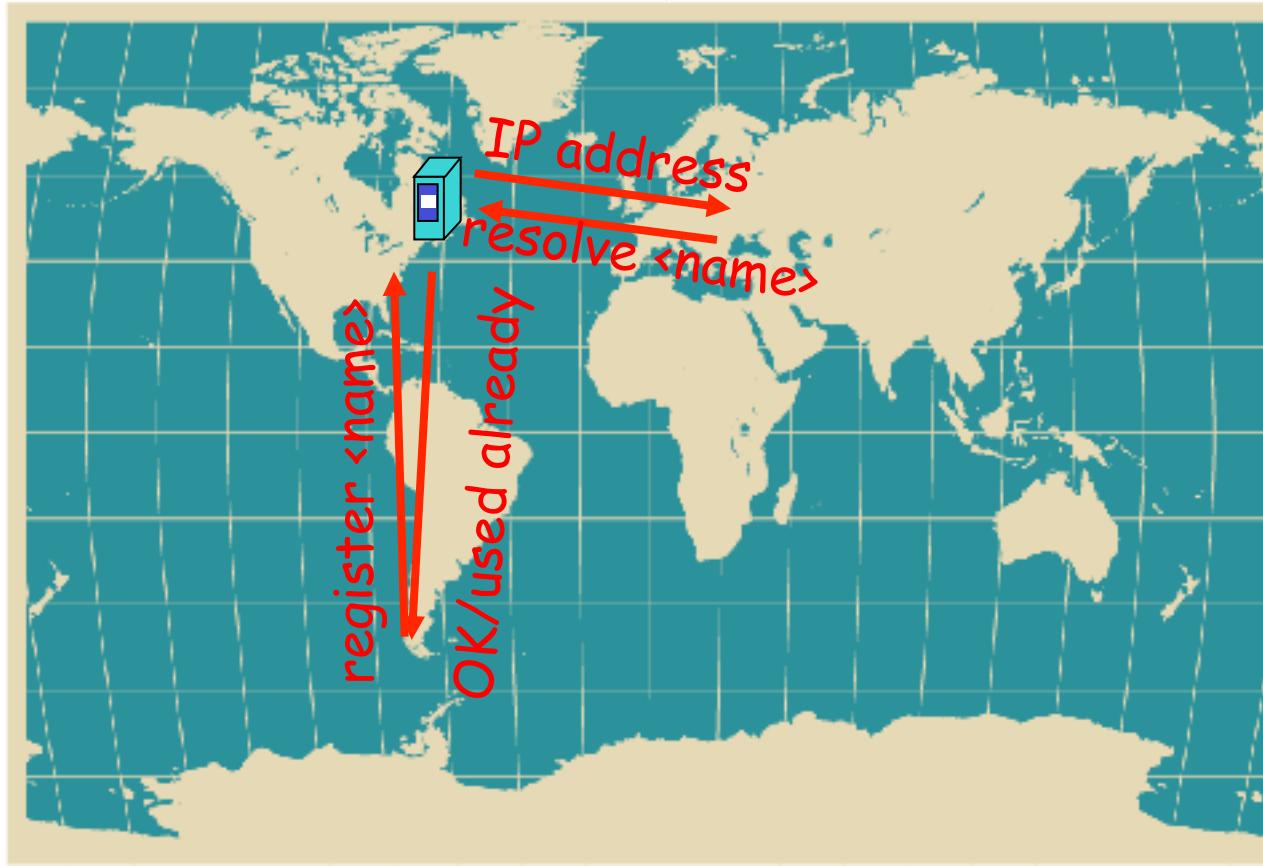
Try DNS: Examples

- dig <type> <domain>
 - type=MX
 - gmail.com
 - type=A
 - type=TXT
 - gmail.com
 - 20120113._domainkey.gmail.com

DNS Design: Dummy Design

- DNS itself can be considered as a client-server system as well
- How about a dummy design: introducing one super Internet DNS server?

THE DNS server of the Internet

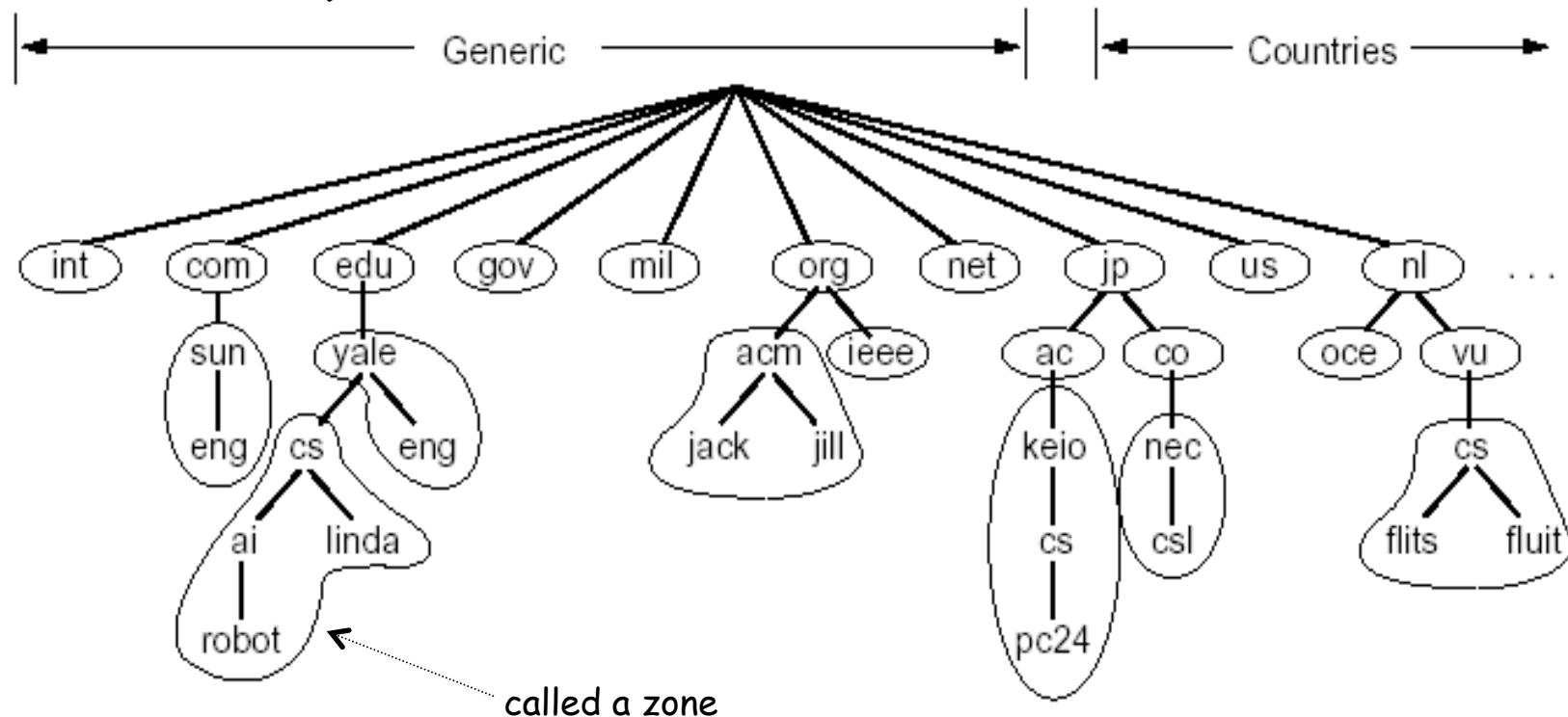


Problems of a Single DNS Server

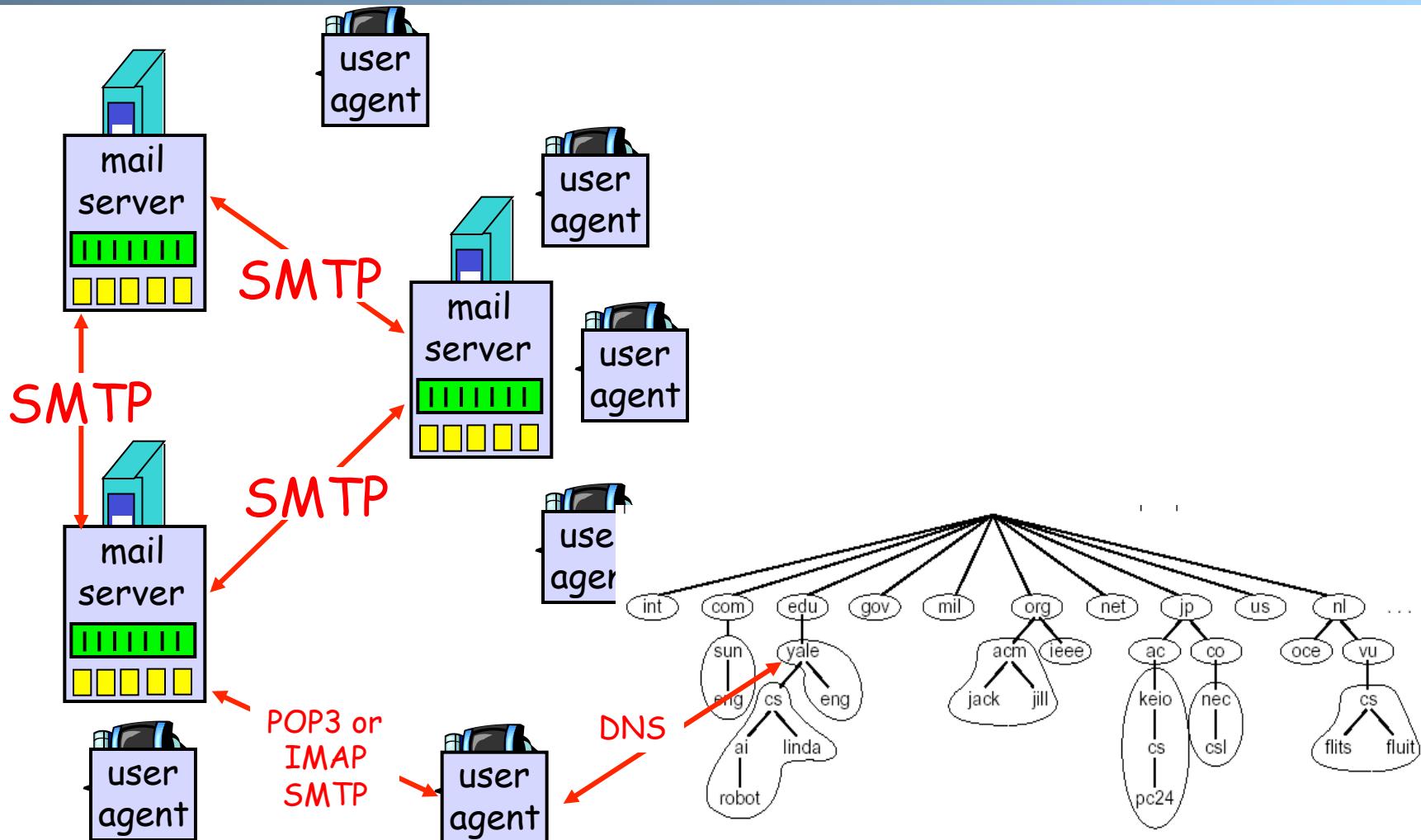
- Scalability and robustness bottleneck
- Administrative bottleneck

DNS: Distributed Management of the Domain Name Space

- A distributed database managed by authoritative name servers
 - divided into zones, where each zone is a sub-tree of the global tree
 - each zone has its own **authoritative name servers**
 - an authoritative name server of a zone may **delegate** a subset (i.e. a sub-tree) of its zone to another name server



Email Architecture + DNS



Root Zone and Root Servers

- The root zone is managed by the root name servers
 - 13 root name servers worldwide

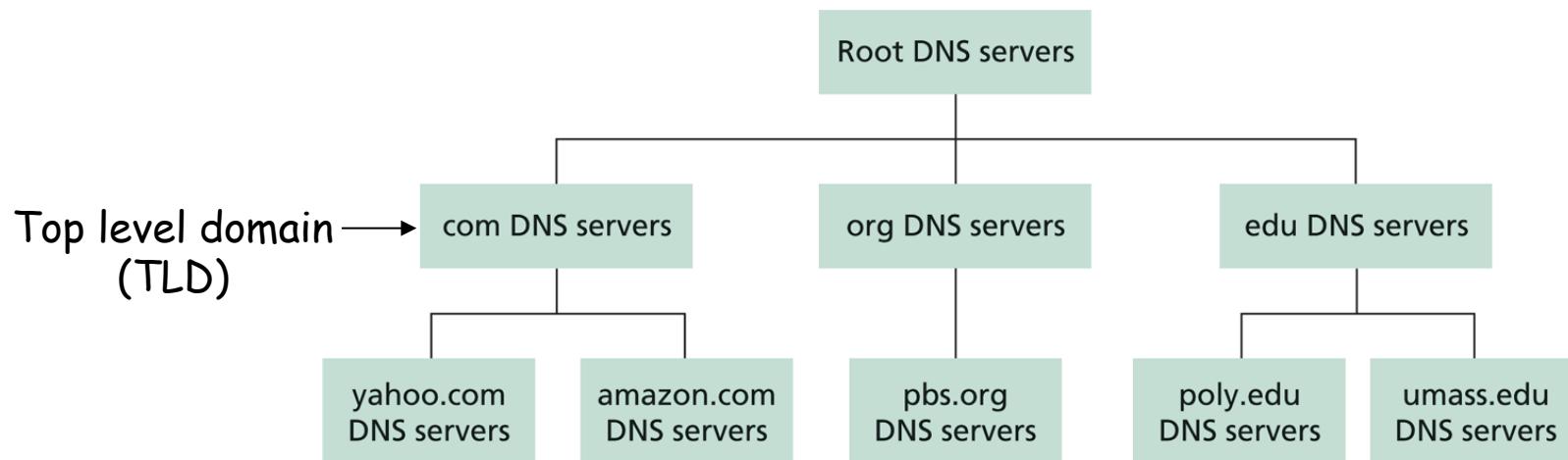
- a. Verisign, Dulles, VA
 - c. Cogent, Herndon, VA (also Los Angeles)
 - d. U Maryland College Park, MD
 - g. US DoD Vienna, VA
 - h. ARL Aberdeen, MD
 - j. Verisign, (11 locations)
 - i. Autonomica, Stockholm (plus 3 other locations)
 - k. RIPE London (also Amsterdam, Frankfurt)
 - m. WIDE Tokyo
- e. NASA Mt View, CA
- f. Internet Software C.
Palo Alto, CA
(and 17 other locations)
- b. USC-ISI Marina del Rey, CA
- l. ICANN Los Angeles, CA



See <http://root-servers.org/> for more details

Linking the Name Servers

- Each name server knows the addresses of the root servers
- Each name server knows the addresses of its immediate children (i.e., those it delegates)



Q: how to query a hierarchy?

DNS Message Flow: Two Types of Queries

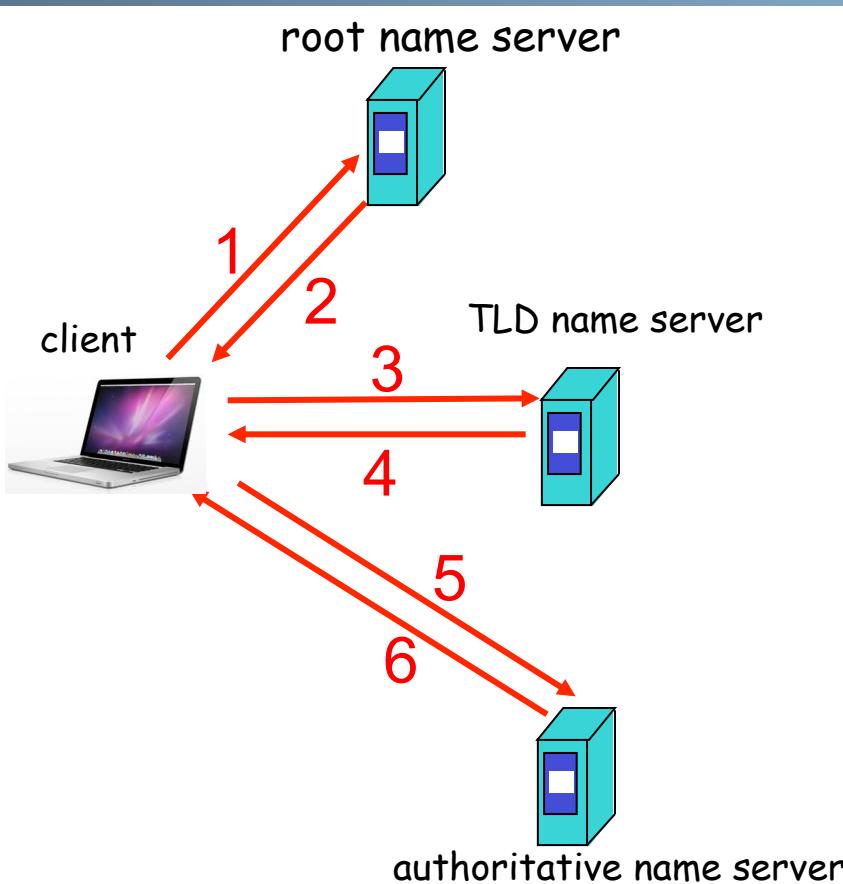
Recursive query:

- ❑ The contacted name server resolves the name completely

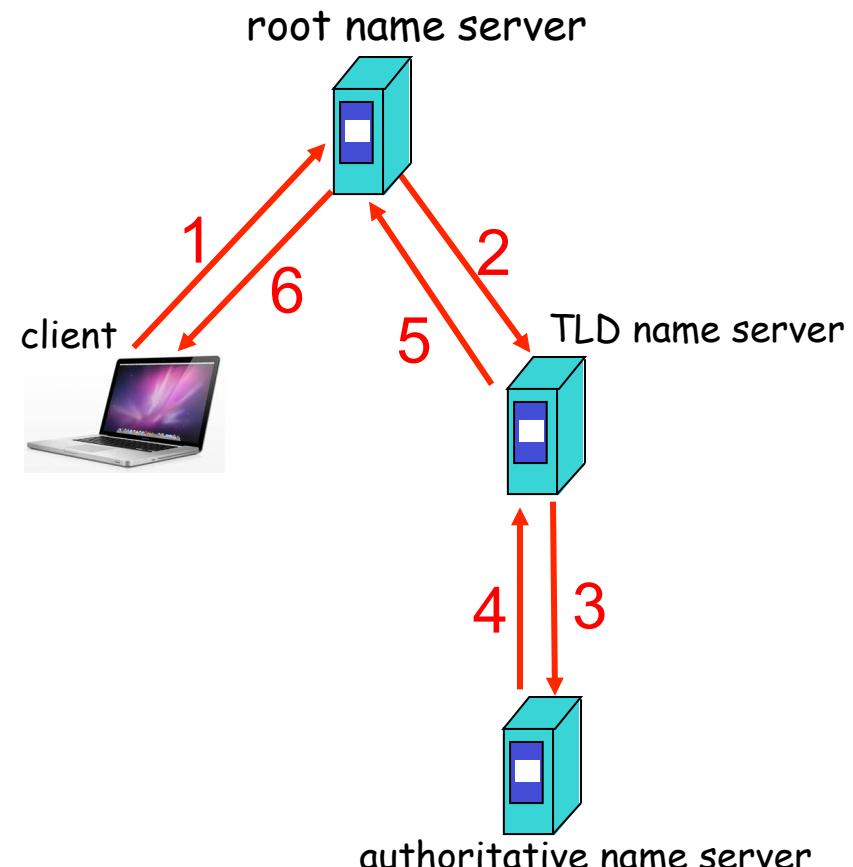
Iterated query:

- ❑ Contacted server replies with name of server to contact
 - “I don’t know this name, but ask this server”

Two Extreme DNS Message Flows

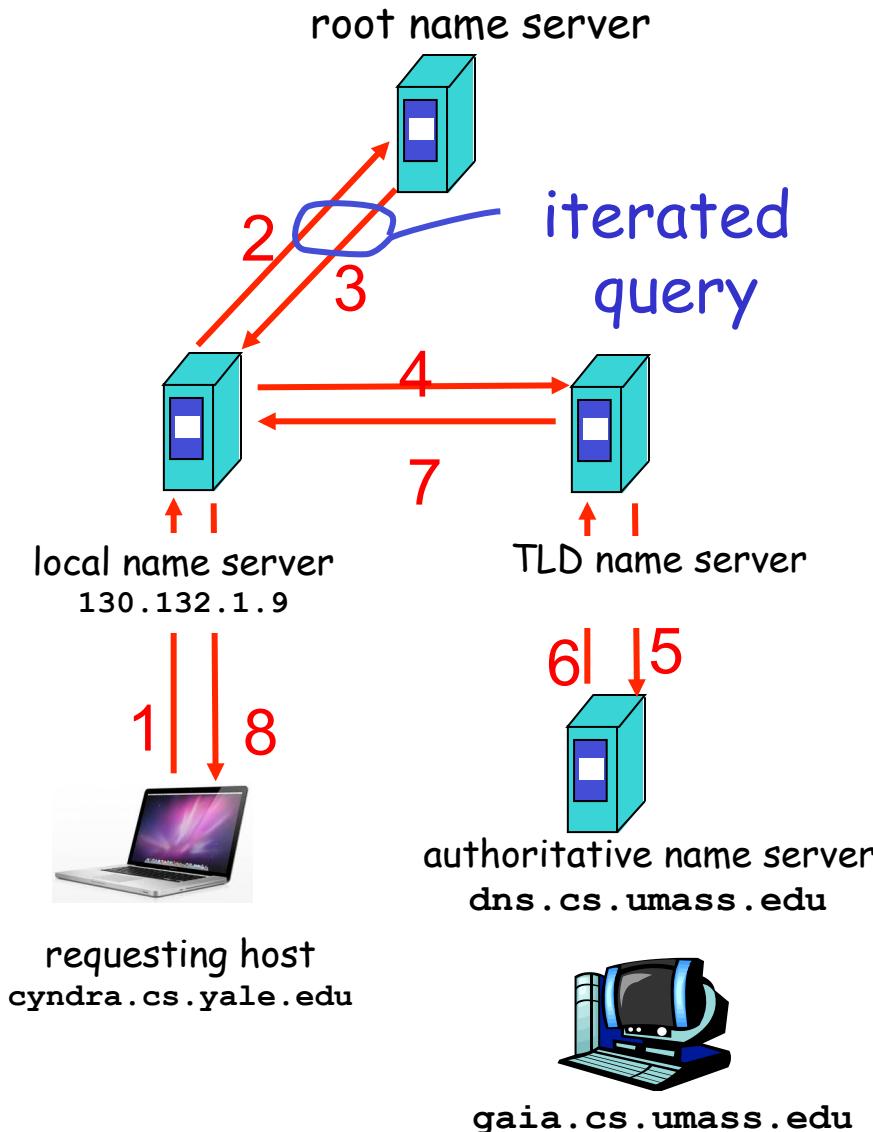


Issues of the
two approaches?



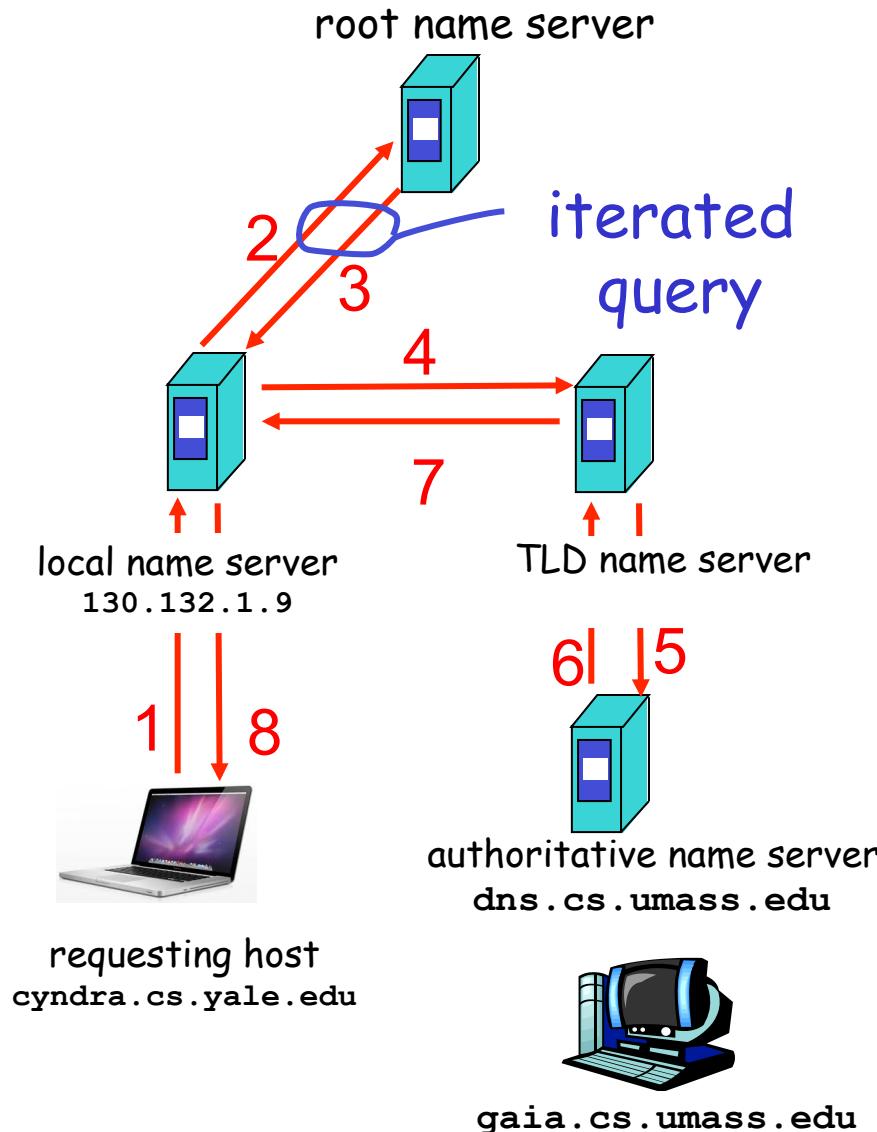
Typical DNS Message Flow: The Hybrid Case

- Host knows only local name server
- Local name server is learned from DHCP, or configured, e.g. /etc/resolv.conf
- Local DNS server helps clients resolve DNS names



Typical DNS Message Flow: The Hybrid Case

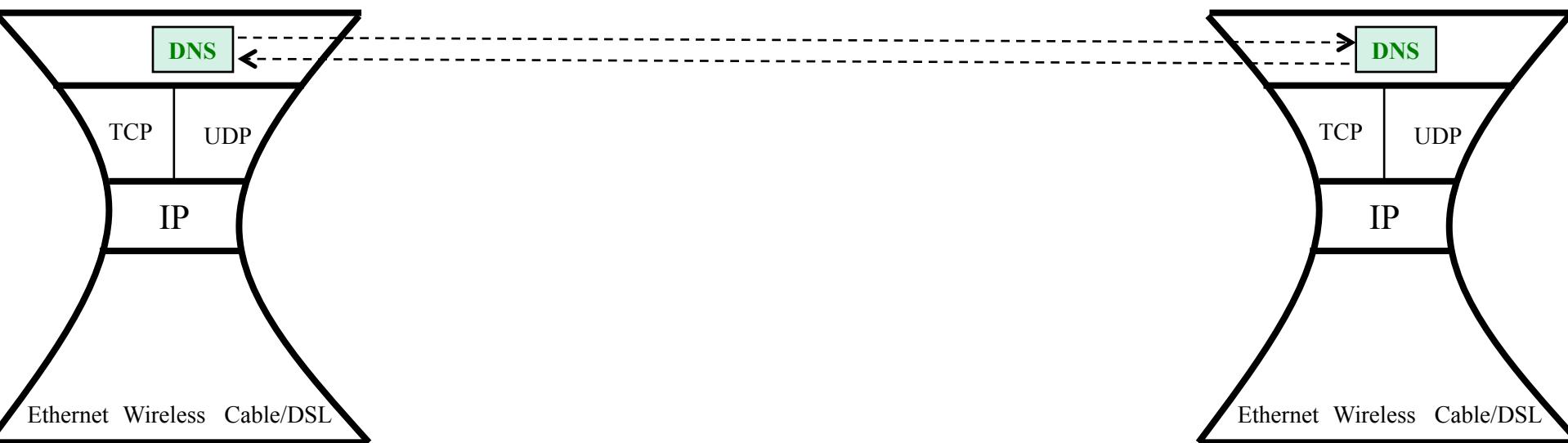
- Host knows only local name server
- Local name server is learned from DHCP, or configured, e.g. /etc/resolv.conf
- Local DNS server helps clients resolve DNS names
- Benefits of local name servers
 - simplifies client
 - Caches/reuses results



Outline

- Recap
- Email security (authentication)
- DNS
 - High-level design
 - Details

DNS Message Format?



Network Applications: DNS Details; UDP Network App Programming

Y. Richard Yang

<http://zoo.cs.yale.edu/classes/cs433/>

2/3/2016

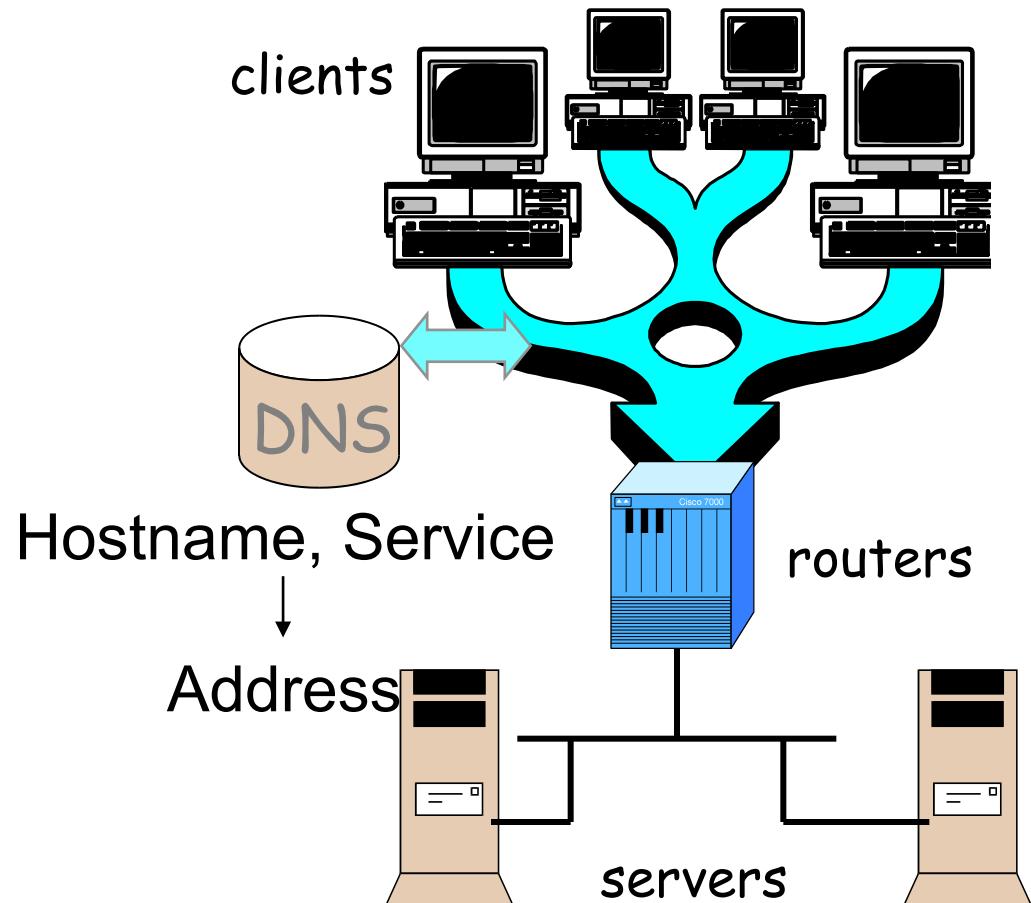
Outline

- Admin and recap
- Network app programming

Recap: Domain Name System (DNS)

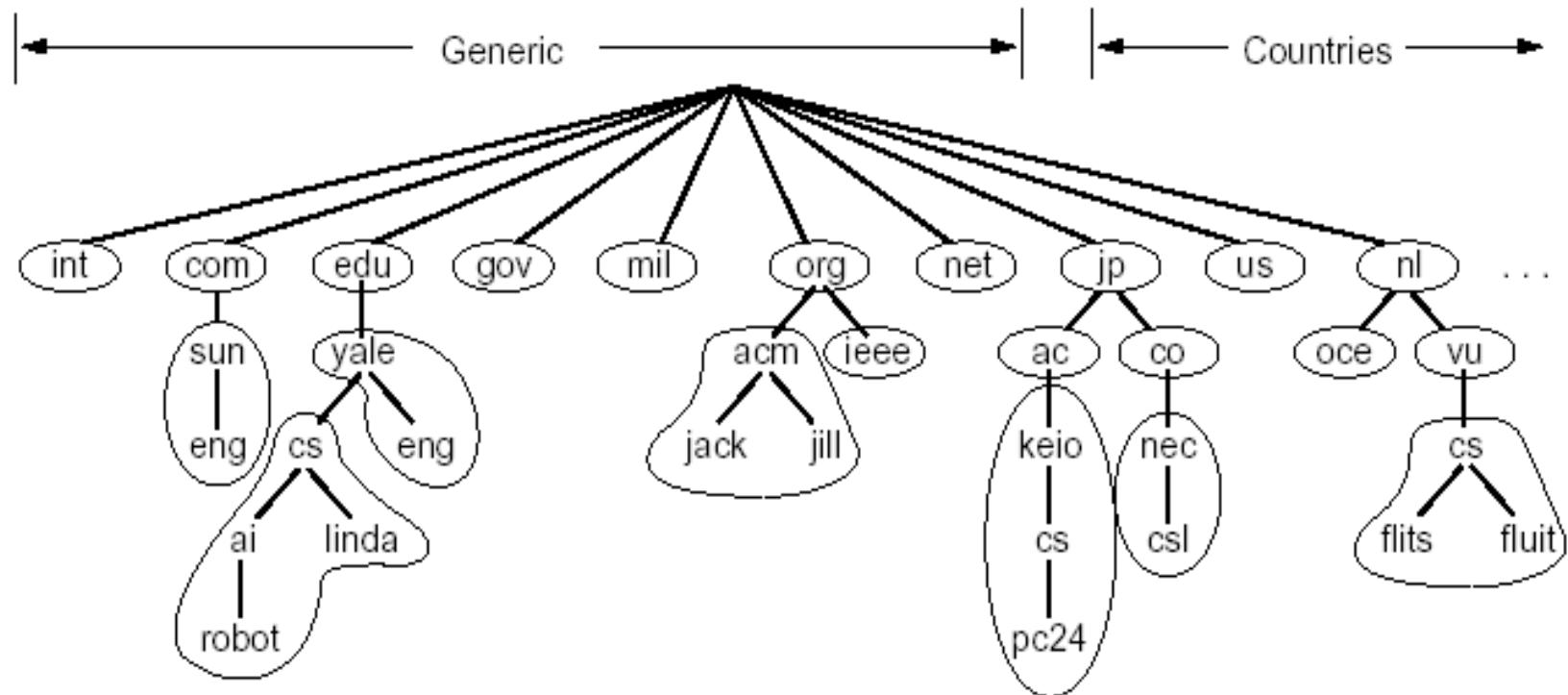
□ Function

- map between (domain name, service) to value, e.g.,
 - (www.cs.yale.edu, Addr)
→ 128.36.229.30
 - (cs.yale.edu, Email)
→ netra.cs.yale.edu



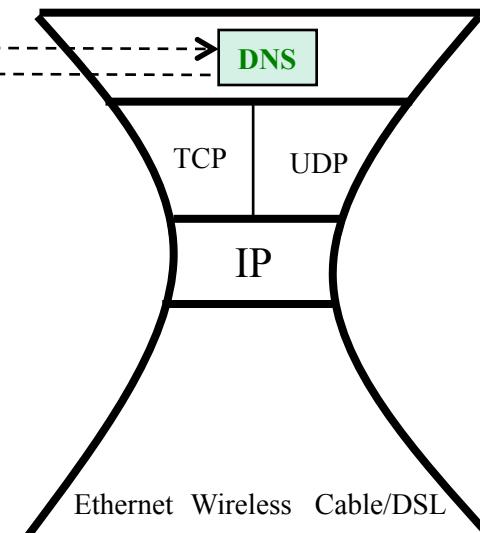
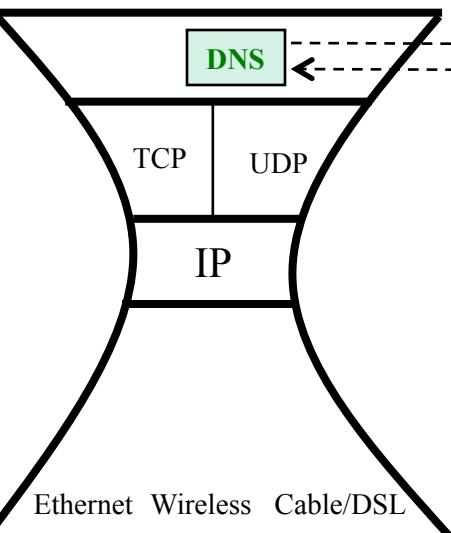
Recap: DNS

- ❑ Key design features of DNS
 - Hierarchical domain name space allowing delegation
 - Recursive or iterative queries



DNS Message Format?

Basic encoding decisions: UDP/TCP,
how to encode domain name, how to
encode answers...



Observing DNS Messages

- Issue DNS query using the command dig:
 - force iterated query to see the trace:
%dig +trace www.cnn.com
 - see the manual for more details

- Capture the messages
 - DNS server is at port 53
 - Display and clear DNS cache
 - <https://support.apple.com/en-us/HT202516> (e.g., MAC)
 - Try to load the dns-capture file from class Schedule page, if you do not want live capture

DNS Protocol, Messages

DNS protocol : typically over UDP (can use TCP);
query and *reply* messages, both with the **same** message format

Identification	Flags	12 bytes
Number of questions	Number of answer RRs	
Number of authority RRs	Number of additional RRs	
Questions (variable number of questions)		Name, type fields for a query
Answers (variable number of resource records)		RRs in response to query
Authority (variable number of resource records)		Records for authoritative servers
Additional information (variable number of resource records)		Additional “helpful” info that may be used

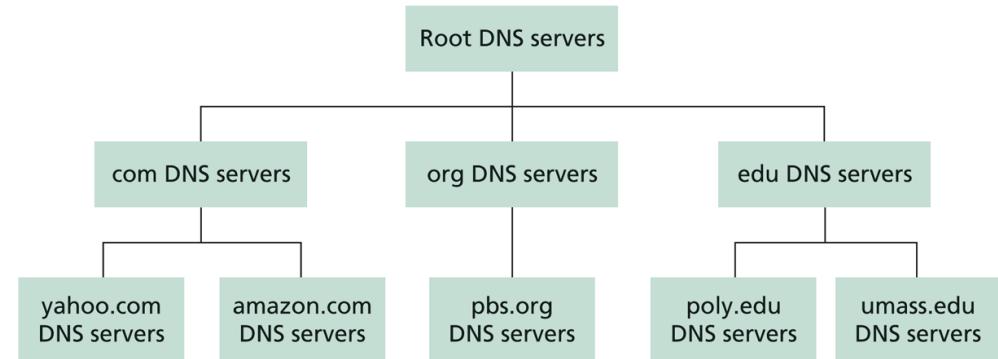
DNS Details

- Header (Sec. 4.1.1 of <https://www.ietf.org/rfc/rfc1035.txt>)
- Encoding of questions (Sec. 4.1.2):
 - [Label-length label-chars]
- Encoding of answers (Sec. 4.1.3)
 - Pointer format (<http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml>)
- See example DNS packets

Evaluation of DNS

Key questions to ask about a C-S application

- **extensible?**
- **scalable?**
- **robust?**
- **security?**



What DNS did Right?

- Hierarchical delegation avoids central control, improving manageability and scalability
- Redundant servers improve robustness
 - see
<http://www.internetnews.com/dev-news/article.php/1486981> for DDoS attack on root servers in Oct. 2002 (9 of the 13 root servers were crippled, but only slowed the network)
- Caching reduces workload and improves robustness

Problems of DNS

- Domain names may not be the best way to name other resources, e.g. files
- Simple query model makes it hard to implement advanced query
- Relatively static resource types make it hard to introduce new services or handle mobility
- Although theoretically you can update the values of the records, it is rarely enabled
- Early binding (separation of DNS query from application query) does not work well in mobile, dynamic environments
 - e.g., load balancing, locate the nearest printer

Outline

- Recap
- Network app programming

Socket Programming

Socket API

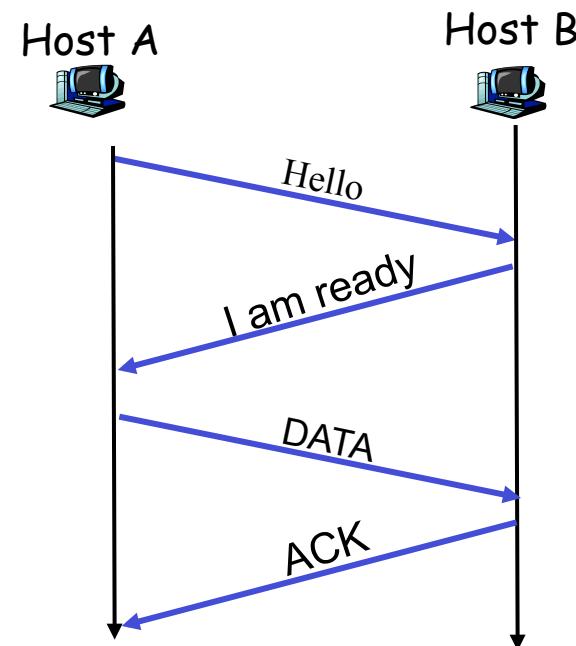
- introduced in
BSD4.1 UNIX, 1981
- Two types of
sockets
 - Connectionless (UDP)
 - connection-oriented
(TCP)

socket

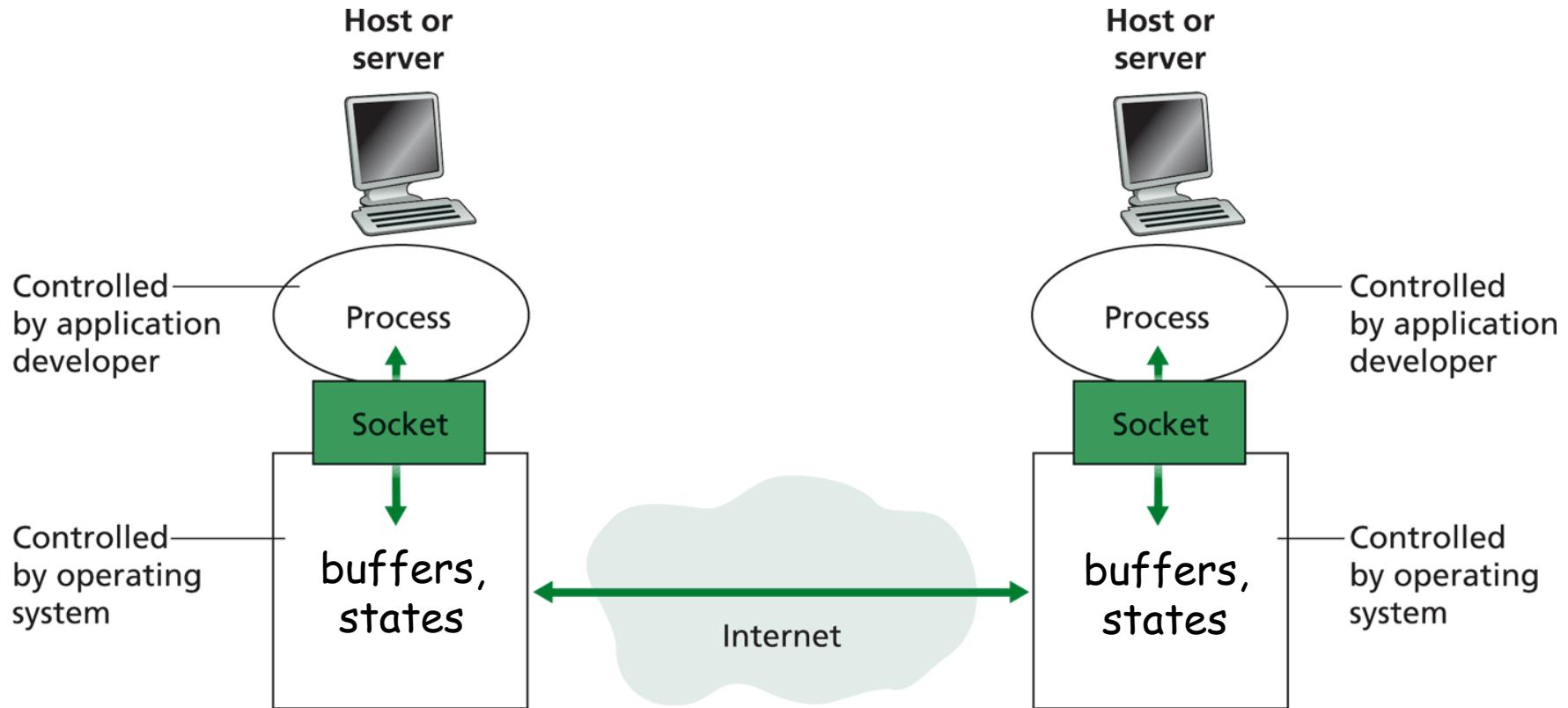
an interface (a “door”)
into which one
application process can
both send and
receive messages to/from
another (remote or
local) application process

Services Provided by Transport

- User data protocol (UDP)
 - multiplexing/demultiplexing
- Transmission control protocol (TCP)
 - multiplexing/demultiplexing
 - reliable data transfer
 - rate control: flow control and congestion control



Big Picture: Socket



Outline

- Recap
- Basic network application programming
 - Overview
 - UDP (Datagram Socket)

DatagramSocket (Java) (Basic)

- **DatagramSocket()**
constructs a datagram socket and binds it to any available port on the local host
- **DatagramSocket(int lport)**
constructs a datagram socket and binds it to the specified port on the local host machine.

- **DatagramPacket(byte[] buf, int length)**
constructs a DatagramPacket for receiving packets of length length.
- **DatagramPacket(byte[] buf, int length, InetAddress address, int port)**
constructs a datagram packet for sending packets of length length to the specified port number on the specified host.

- **receive(DatagramPacket p)**
receives a datagram packet from this socket.
- **send(DatagramPacket p)**
sends a datagram packet from this socket.

- **close()**
closes this datagram socket.

Connectionless UDP: Big Picture (Java version)

Server (running on serv)

create socket,
port=x, for
incoming request:
`serverSocket =
DatagramSocket(x)`

```
read request from  
serverSocket  
generate reply, create  
datagram using client  
host address, port number
```

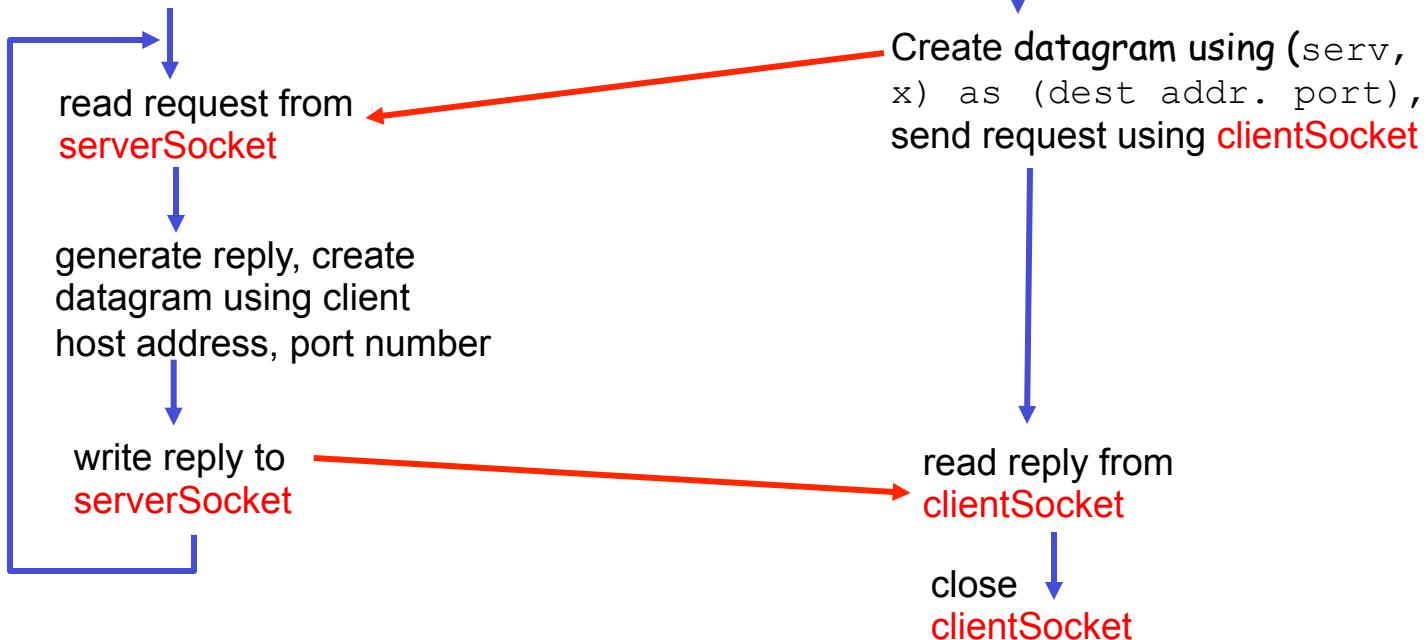
write reply to
`serverSocket`

Client

create socket,
`clientSocket =
DatagramSocket()`

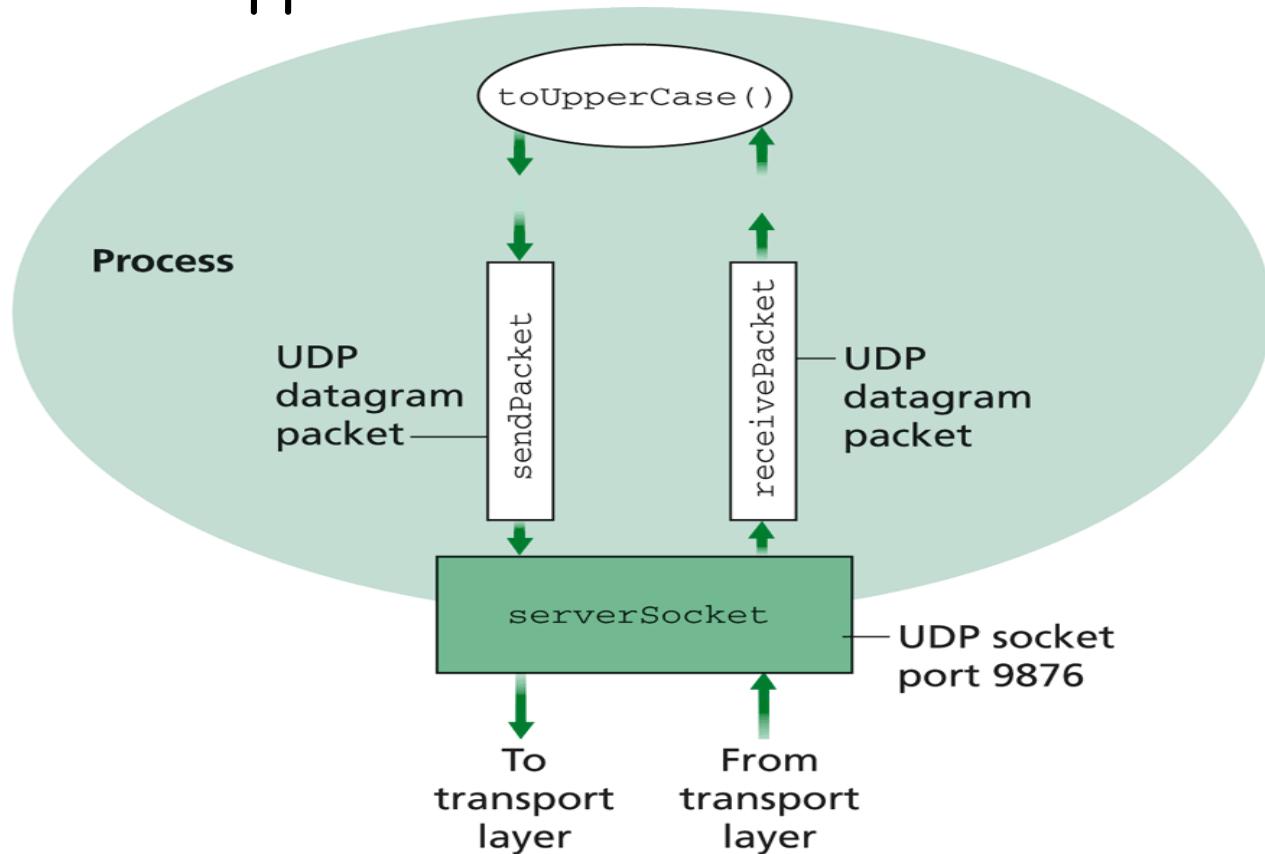
Create datagram using (serv,
x) as (dest addr. port),
send request using `clientSocket`

```
read reply from  
clientSocket  
close  
clientSocket
```



Example: UDPServer.java

- A simple UDP server which changes any received sentence to upper case.



Java Server (UDP): Create Socket

```
import java.io.*;
import java.net.*;

class UDPServer {
    public static void main(String args[]) throws Exception
    {
        DatagramSocket serverSocket = new DatagramSocket(9876);
    }
}
```

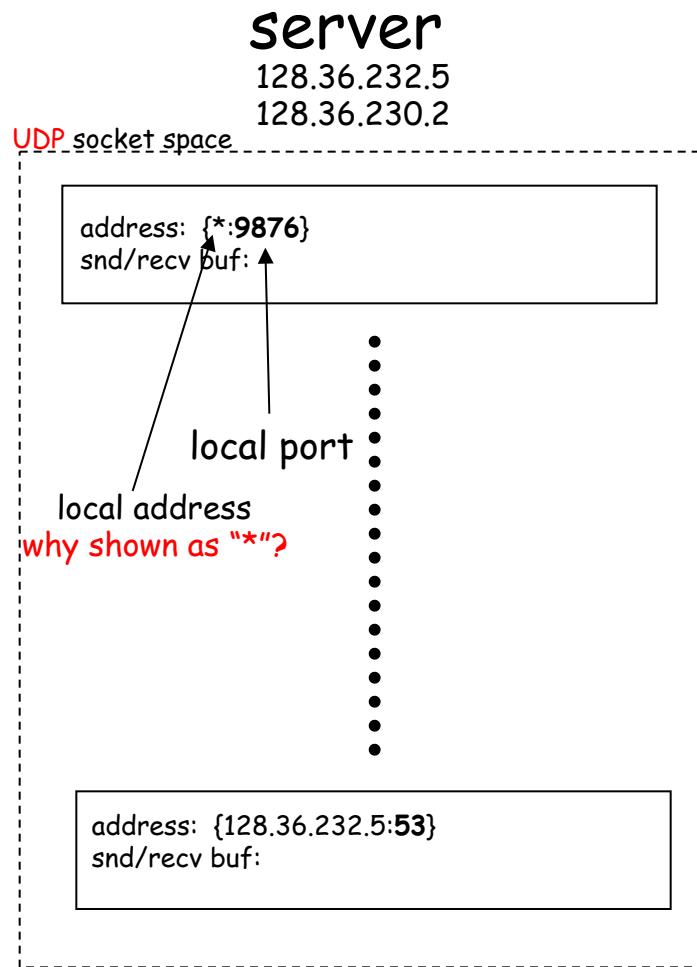
*Create
datagram socket
bind at port 9876*



Check socket state:

%netstat -a -p udp -n

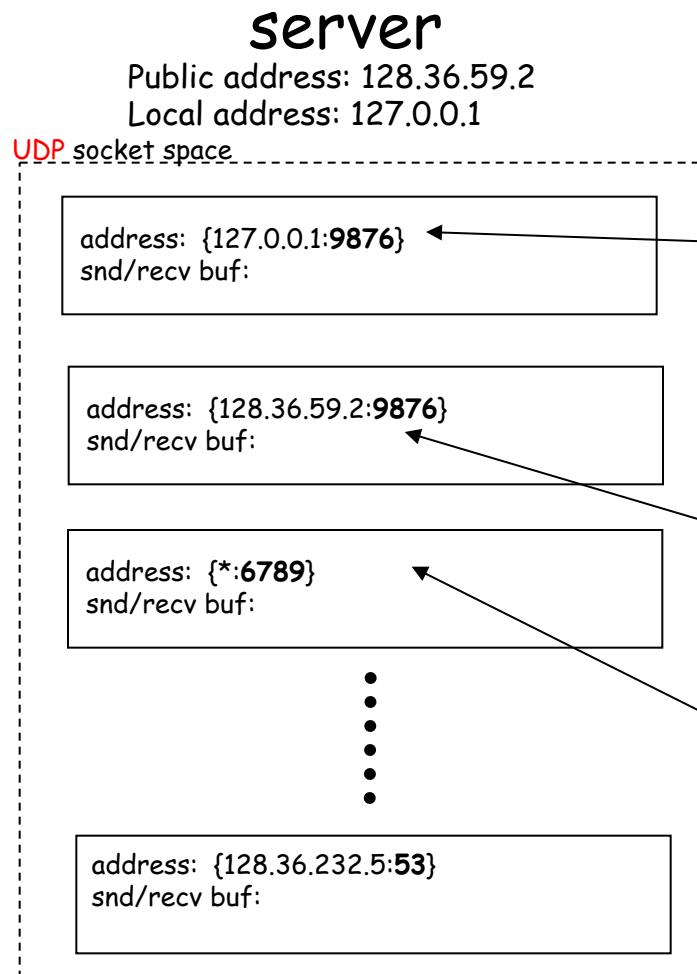
System State after the Call



"*" indicates that the socket binds to **all** IP addresses of the machine:

% ifconfig -a

Binding to Specific IP Addresses



```
InetAddress sIP1 =  
    InetAddress.getByName("localhost");  
DatagramSocket ssock1 = new  
    DatagramSocket(9876, sIP1);
```

```
InetAddress sIP2 =  
    InetAddress.getByName("128.36.59.2");  
DatagramSocket ssock2 = new  
    DatagramSocket(9876, sIP2);
```

```
DatagramSocket serverSocket = new  
    DatagramSocket(6789);
```

UDP Demultiplexing

server

Public address: 128.36.59.2

Local address: 127.0.0.1

UDP socket space

address: {127.0.0.1:9876}

snd/recv buf:

address: {128.36.59.2:9876}

snd/recv buf:

•
•
•
•
•

address: {128.36.232.5:53}

snd/recv buf:

SP: x

DP: 9876

S-IP: A

D-IP: 127.0.0.1

P1

P2

client
on server

client
IP: B

UDP demultiplexing is based on matching (dst address, dst port)

UDP Demultiplexing

server

Public address: 128.36.59.2

Local address: 127.0.0.1

UDP socket space

address: {127.0.0.1:9876}

snd/recv buf:

address: {128.36.59.2:9876}

snd/recv buf:

address: {*:6789}

snd/recv buf:

⋮
⋮

address: {128.36.232.5:53}

snd/recv buf:

SP: x

DP: 9876

S-IP: A

D-IP: 127.0.0.1

Client
on server

P1

SP: y

DP: 6789

S-IP: C

D-IP: 128.36.59.2

P3

client
IP: C

UDP demultiplexing is based on matching (dst address, dst port)

Per Socket State

- Each Datagram socket has a set of states:
 - local address
 - send buffer size
 - receive buffer size
 - timeout
 - traffic class

See <http://download.java.net/jdk7/archive/b123/docs/api/java/net/DatagramSocket.html>

Example: socket state after clients sent msgs to the server

Java Server (UDP): Receiving

```
import java.io.*;
import java.net.*;

class UDPServer {
    public static void main(String args[]) throws Exception
    {

        DatagramSocket serverSocket = new DatagramSocket(9876);

        byte[] receiveData = new byte[1024];
        byte[] sendData = null;

        while(true)
        {
            Create space for
            received datagram] → DatagramPacket receivePacket =
                new DatagramPacket(receiveData, receiveData.length);

            Receive
            datagram] → serverSocket.receive(receivePacket);
        }
    }
}
```

Create space for
received datagram

Receive
datagram

DatagramPacket

□ Receiving

- **DatagramPacket(byte[] buf, int length)**
constructs a DatagramPacket for receiving packets of length length.
- **DatagramPacket(byte[] buf, int offset, int length)**
constructs a DatagramPacket for receiving packets starting at offset, length length.

□ Sending

- **DatagramPacket(byte[] buf, int length,
InetAddress address, int port)**
constructs a datagram packet for sending packets of length length to the specified port number on the specified host.
- **DatagramPacket(byte[] buf, int offset, int length,
InetAddress address, int port)**

Java Server (UDP): Processing

```
import java.io.*;
import java.net.*;

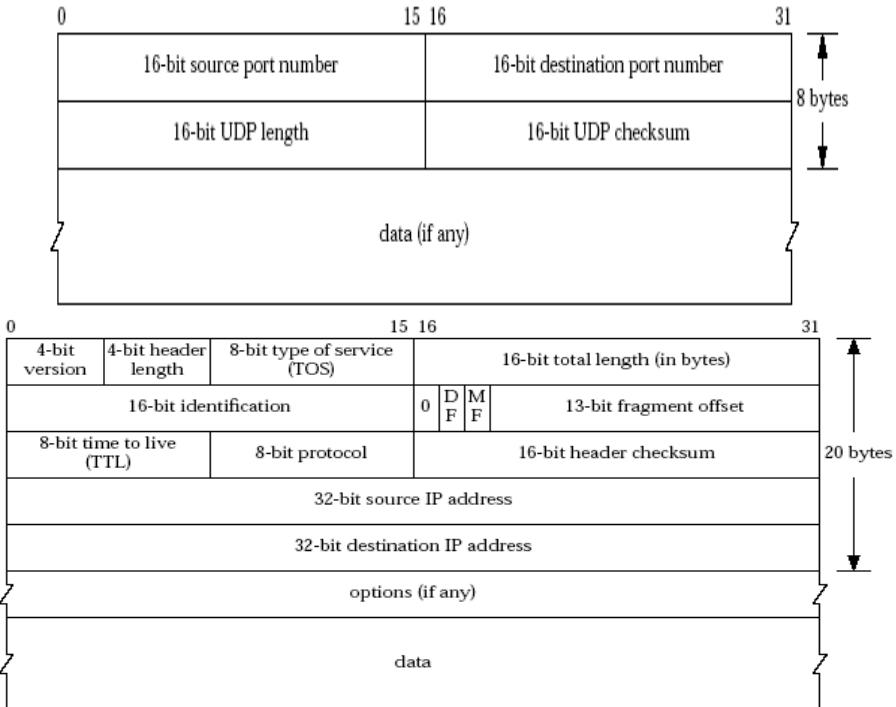
class UDPServer {
    public static void main(String args[]) throws Exception {
        ...
        // process data
        String sentence = new String(receivePacket.getData(),
            0, receivePacket.getLength());
        String capitalizedSentence = sentence.toUpperCase();
        sendData = capitalizedSentence.getBytes();
    }
}
```

getData() returns a pointer to an underlying buffer array;
for efficiency, don't assume receive() will reset the rest of the array

getLength() returns how much data is valid.

Java Server (UDP): Response

- Java DatagramPacket:
 - getAddress() /
getPort() returns the **source address/port**



Java server (UDP): Reply

Get IP addr
port #, of
sender

```
InetAddress IPAddress = receivePacket.getAddress();  
int port = receivePacket.getPort();
```

Create datagram
to send to client

```
DatagramPacket sendPacket =  
    new DatagramPacket(sendData, sendData.length,  
                      IPAddress, port);
```

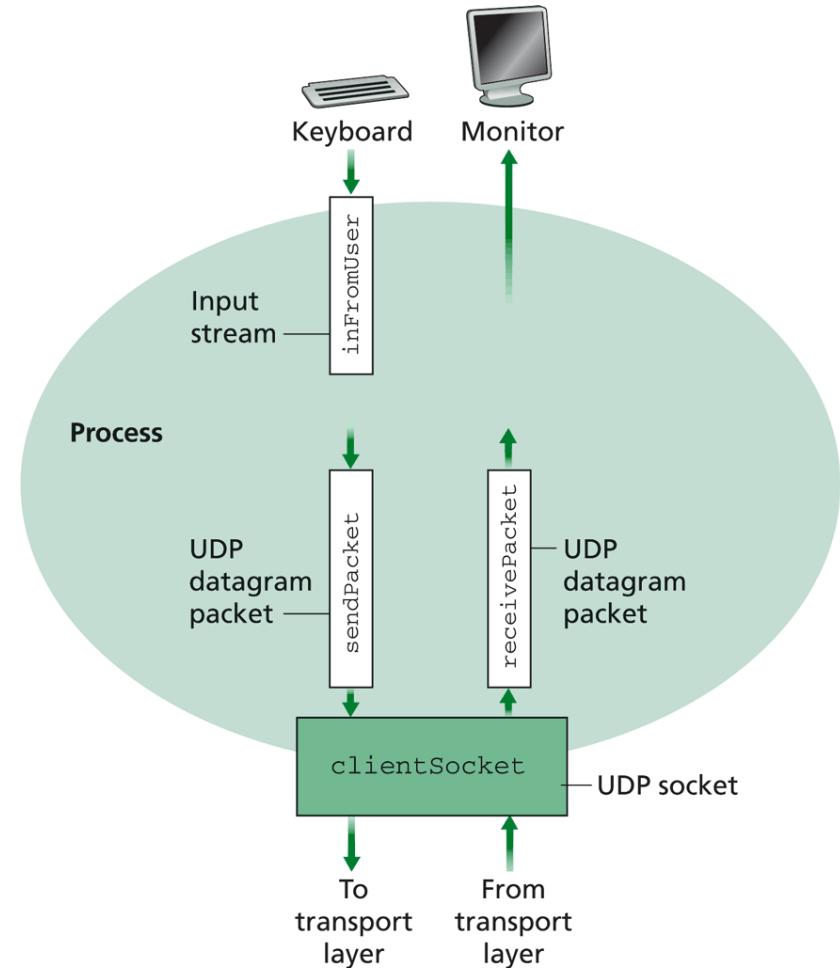
Write out
datagram
to socket

```
serverSocket.send(sendPacket);
```

End of while loop,
loop back and wait for
another datagram

Example: UDPClient.java

- A simple UDP client which reads input from keyboard, sends the input to server, and reads the reply back from the server.



Example: Java client (UDP)

```
import java.io.*;
import java.net.*;

class UDPClient {
    public static void main(String args[]) throws Exception
    {
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));
        String sentence = inFromUser.readLine();
        byte[] sendData = sentence.getBytes();

        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress sIPAddress = InetAddress.getByName("servname");
    }
}
```

Create input stream → BufferedReader inFromUser =
new BufferedReader(new InputStreamReader(System.in));
String sentence = inFromUser.readLine();
byte[] sendData = sentence.getBytes();

Create client socket → DatagramSocket clientSocket = new DatagramSocket();

Translate hostname to IP address using DNS → InetAddress sIPAddress = InetAddress.getByName("servname");

Example: Java client (UDP), cont.

```
Create datagram  
with data-to-send,  
length, IP addr, port → DatagramPacket sendPacket =  
new DatagramPacket(sendData, sendData.length, sIPAddress, 9876);  
  
Send datagram  
to server → clientSocket.send(sendPacket);  
  
byte[] receiveData = new byte[1024];  
DatagramPacket receivePacket =  
new DatagramPacket(receiveData, receiveData.length);  
  
Read datagram  
from server → clientSocket.receive(receivePacket);  
  
String modifiedSentence =  
new String(receivePacket.getData());  
  
System.out.println("FROM SERVER:" + modifiedSentence);  
clientSocket.close();  
}  
}
```

Demo

%mac: java UDPServer

% netstat to see buffer

%cicada: java UDPClient <server>

% wireshark to capture traffic

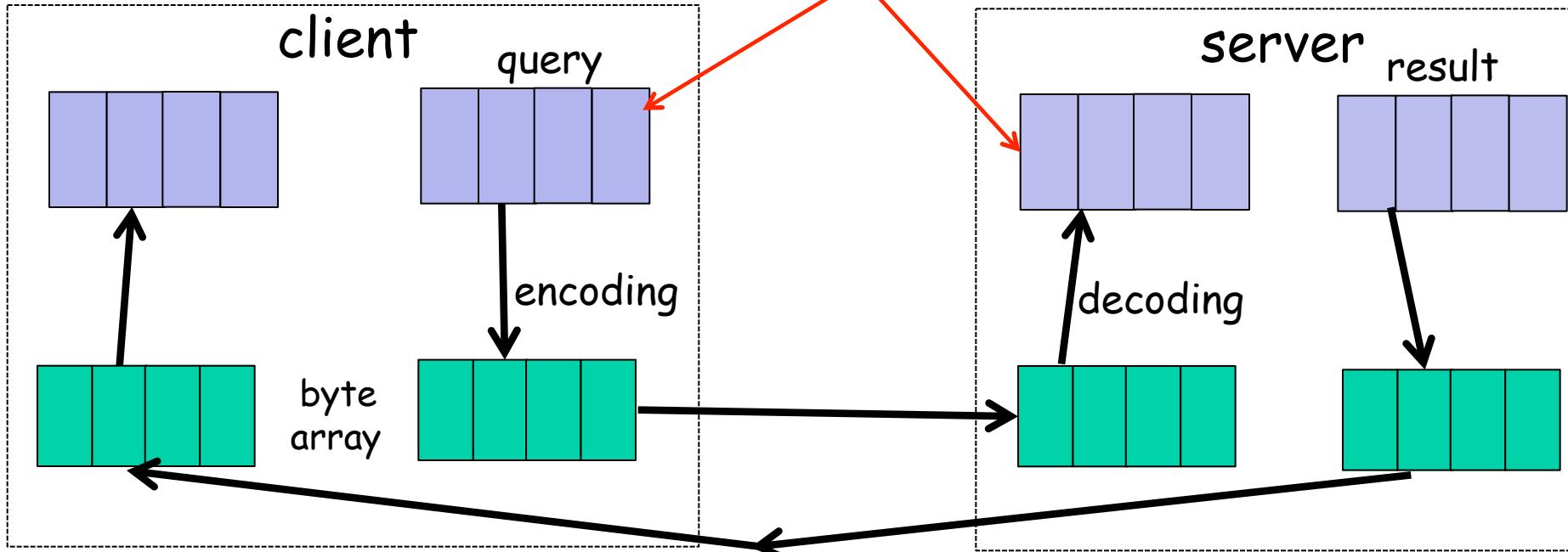
Discussion on Example Code

- A simple upper-case UDP echo service is among the simplest network service.
- Are there any problems with the program?

Data Encoding/Decoding

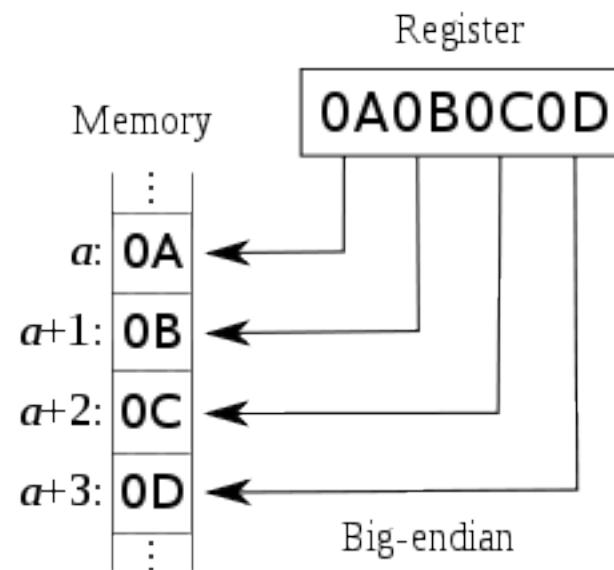
- Pay attention to encoding/decoding of data:
transport layer handles only a sequence of bytes

if not careful, query sent !=
query received (how?)

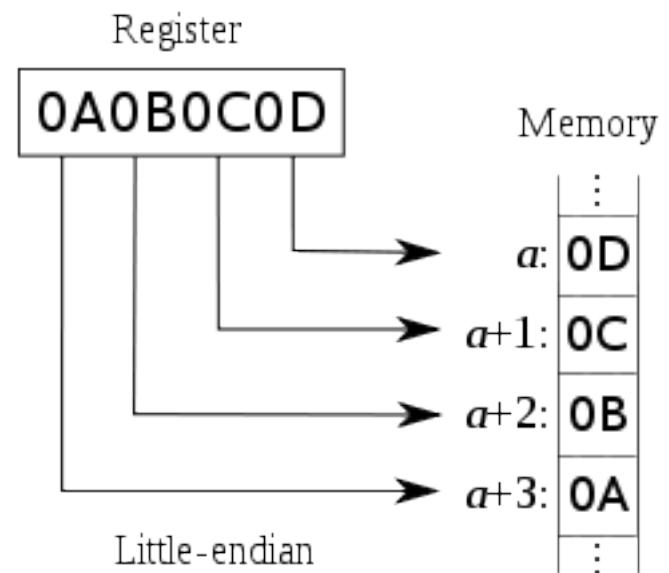


Example: Endianness of Numbers

- $\text{int var} = 0x0A0B0C0D$



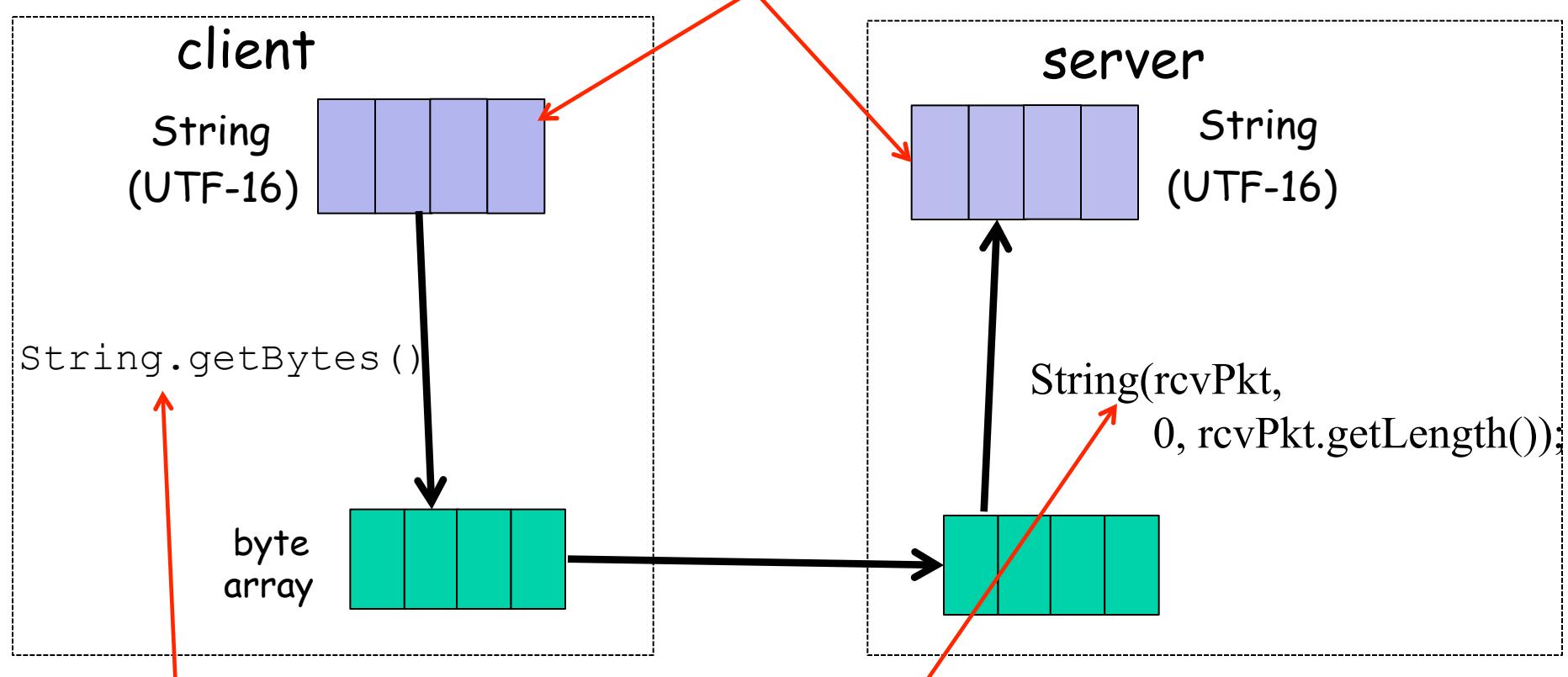
ARM, Power PC, Motorola 68k, IA-64



Intel x86

Example: String and Chars

Will we always get back the same string?



Depends on default local platform char set (why?) :
`java.nio.charset.Charset.defaultCharset()`

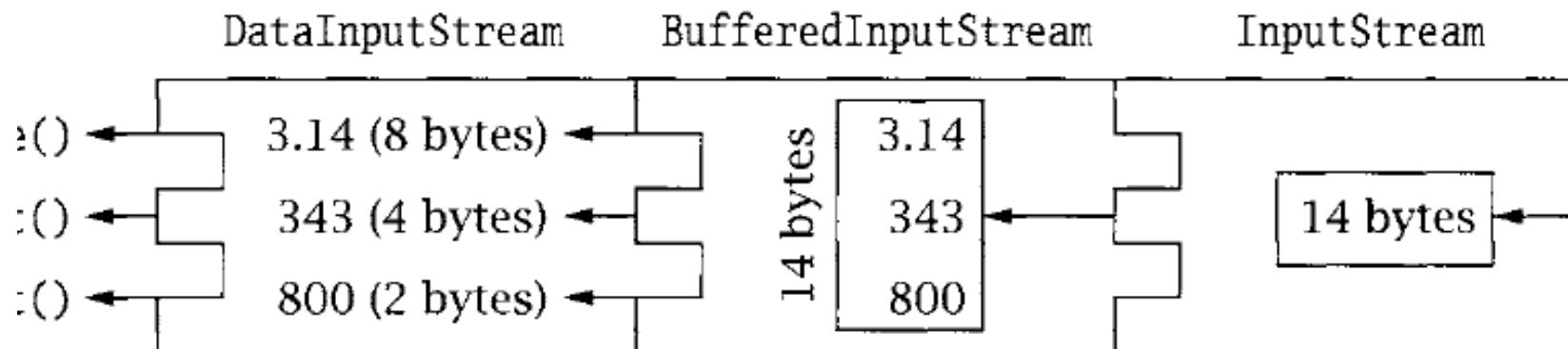
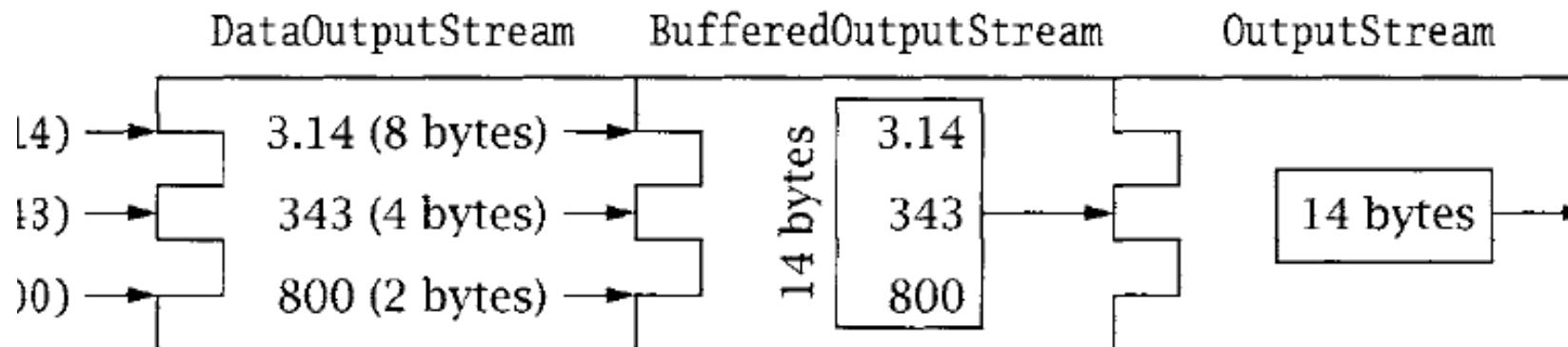
Example: Charset Troubles

- Try
 - java EncodingDecoding US-ASCII UTF-8

Encoding/Decoding as a Common Source of Errors

- Please read chapter 4 of Java Network Programming for more details
- Common mistake even in many (textbook) examples:
 - [http://www.java2s.com/Code/Java/Network-Protocol/
UseDatagramSockettosendoutandreceiveDatagramPacket.htm](http://www.java2s.com/Code/Java/Network-Protocol/UseDatagramSockettosendoutandreceiveDatagramPacket.htm)

DataStream



Network Applications:

Network App Programming: TCP

FTP

Y. Richard Yang

<http://zoo.cs.yale.edu/classes/cs433/>

2/8/2016

Outline

- Admin and recap
- Network app programming
- FTP

Recap: DNS Protocol, Messages

Many features: typically over **UDP** (can use **TCP**); **query** and **reply** messages with the **same message format**; **length/content encoding of names**; **simple compression**; **additional info as server push**

Identification	Flags	12 bytes
Number of questions	Number of answer RRs	
Number of authority RRs	Number of additional RRs	
Questions (variable number of questions)		Name, type fields for a query
Answers (variable number of resource records)		RRs in response to query
Authority (variable number of resource records)		Records for authoritative servers
Additional information (variable number of resource records)		Additional “helpful” info that may be used

Recap: Connectionless UDP: Big Picture (Java version)

Server (running on serv)

create socket,
port=x, for
incoming request:
`serverSocket =
DatagramSocket(x)`

```
read request from  
serverSocket  
generate reply, create  
datagram using client  
host address, port number
```

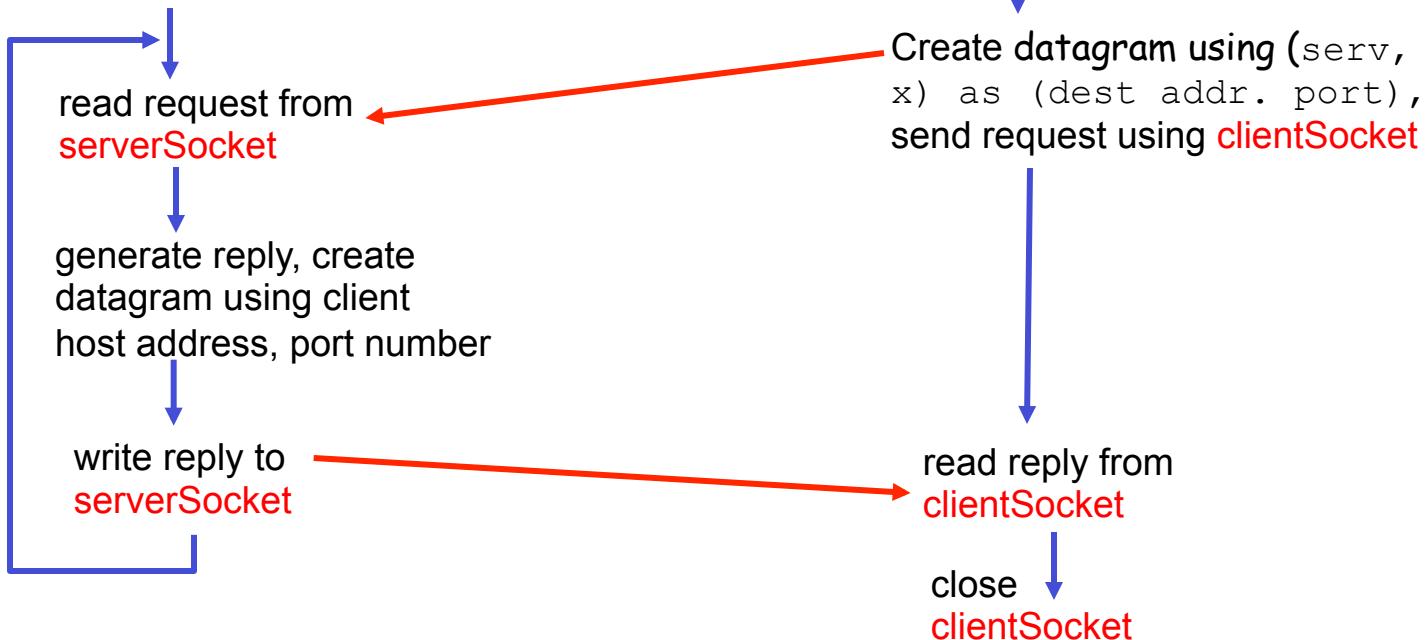
write reply to
`serverSocket`

Client

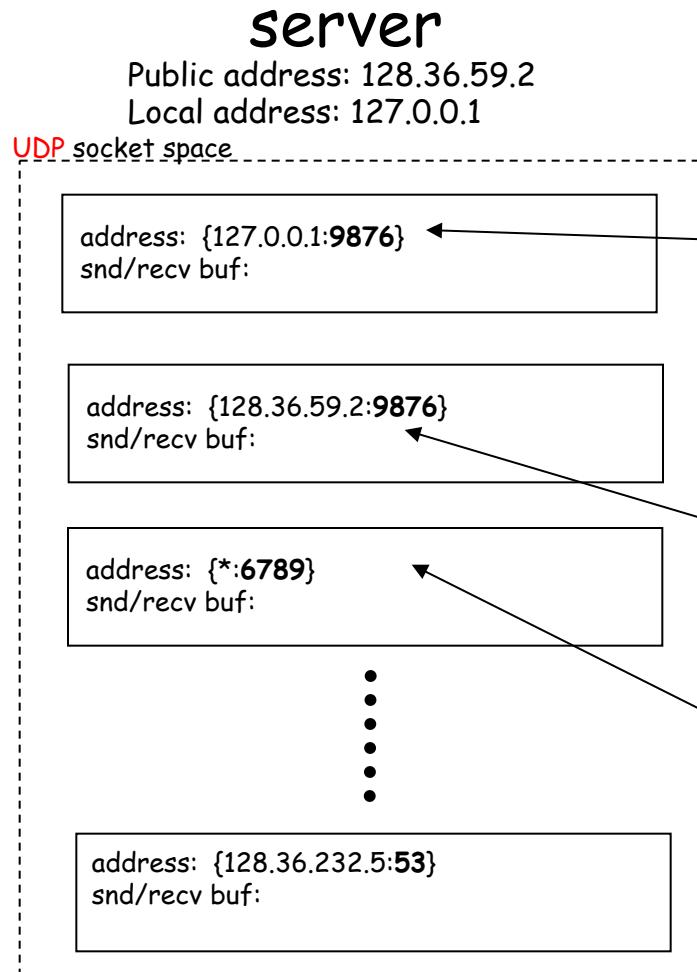
create socket,
`clientSocket =
DatagramSocket()`

Create datagram using (serv,
x) as (dest addr. port),
send request using `clientSocket`

```
read reply from  
clientSocket  
close  
clientSocket
```



Recap: UDP Sockets



```
InetAddress sIP1 =  
InetAddress.getByName("localhost");  
DatagramSocket ssock1 =  
new DatagramSocket(9876, sIP1);
```

```
InetAddress sIP2 =  
InetAddress.getByName("128.36.59.2");  
DatagramSocket ssock2 =  
new DatagramSocket(9876, sIP2);
```

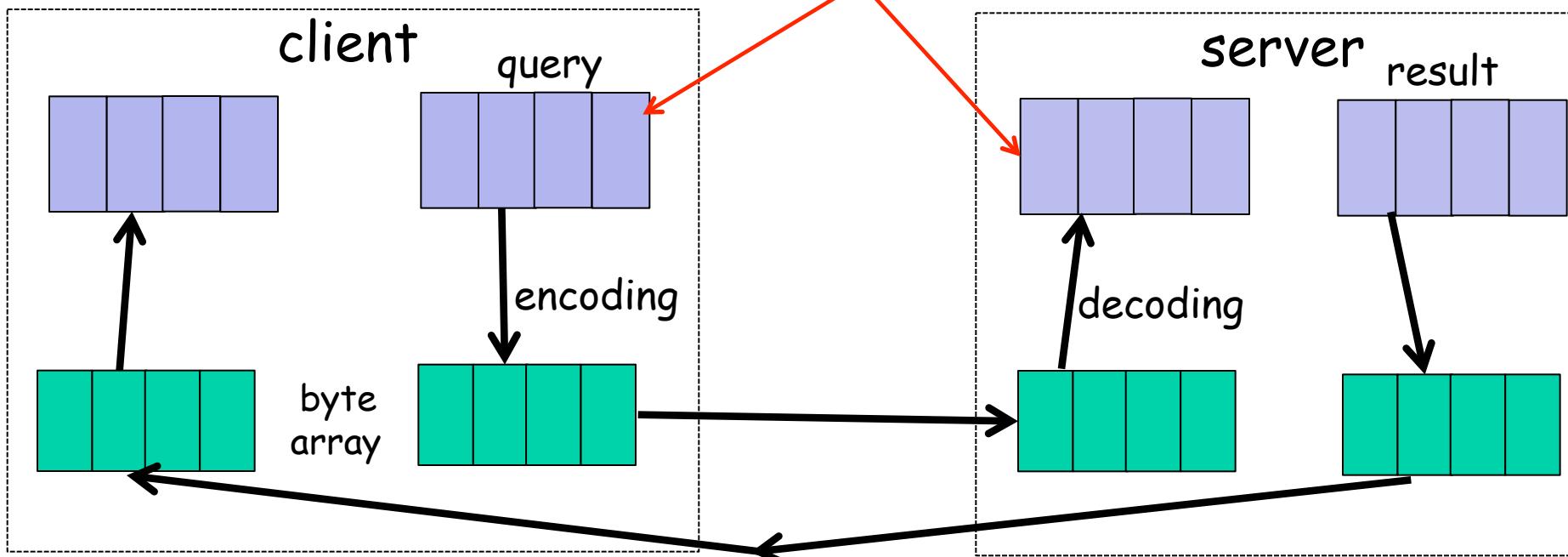
```
DatagramSocket serverSocket =  
new DatagramSocket(6789);
```

UDP demultiplexing is based on matching (dst address, dst port)

Recap: Data Encoding/Decoding

- Pay attention to encoding/decoding of data:
transport layer handles only a sequence of bytes

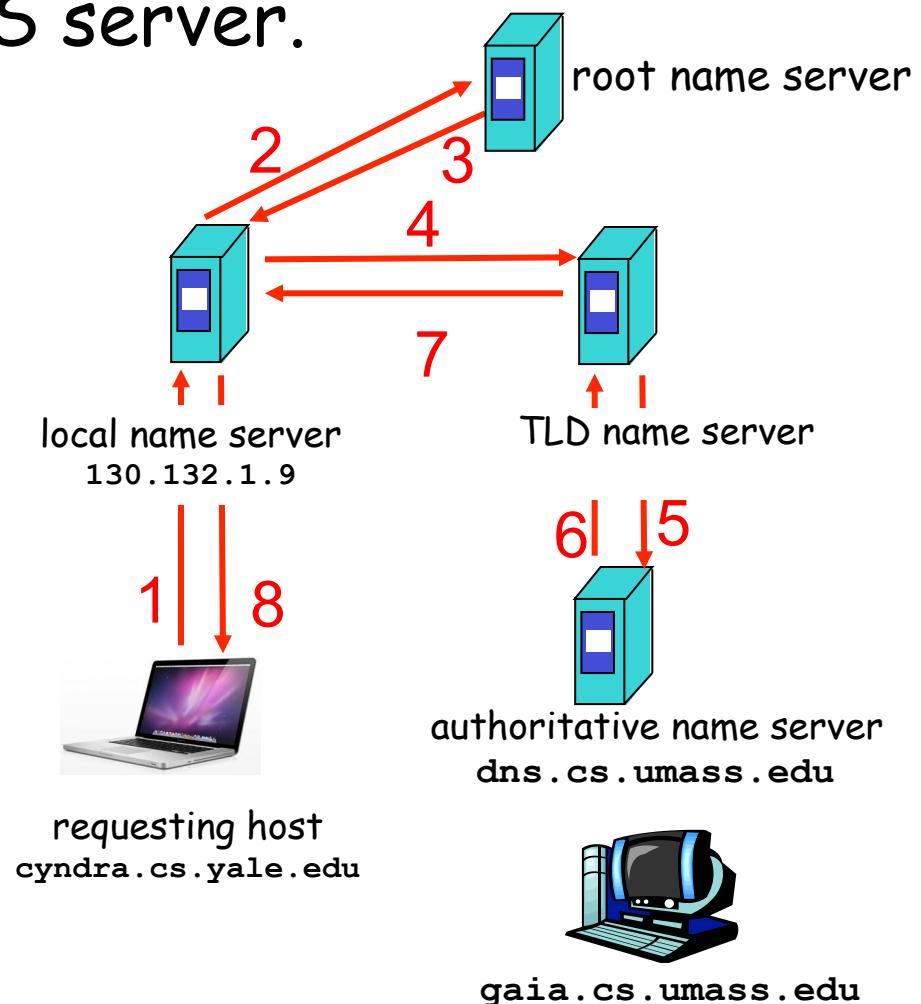
if not careful, query sent !=
query received



Discussion: UDP/DNS Server Pseudocode

- Modify the example UDP server code to implement a local DNS server.

Identification	Flags	12 bytes
Number of questions	Number of answer RRs	
Number of authority RRs	Number of additional RRs	
Questions (variable number of questions)		Name, type fields for a query
Answers (variable number of resource records)		RRs in response to query
Authority (variable number of resource records)		Records for authoritative servers
Additional information (variable number of resource records)		Additional "helpful" info that may be used



UDP/DNS Implementation

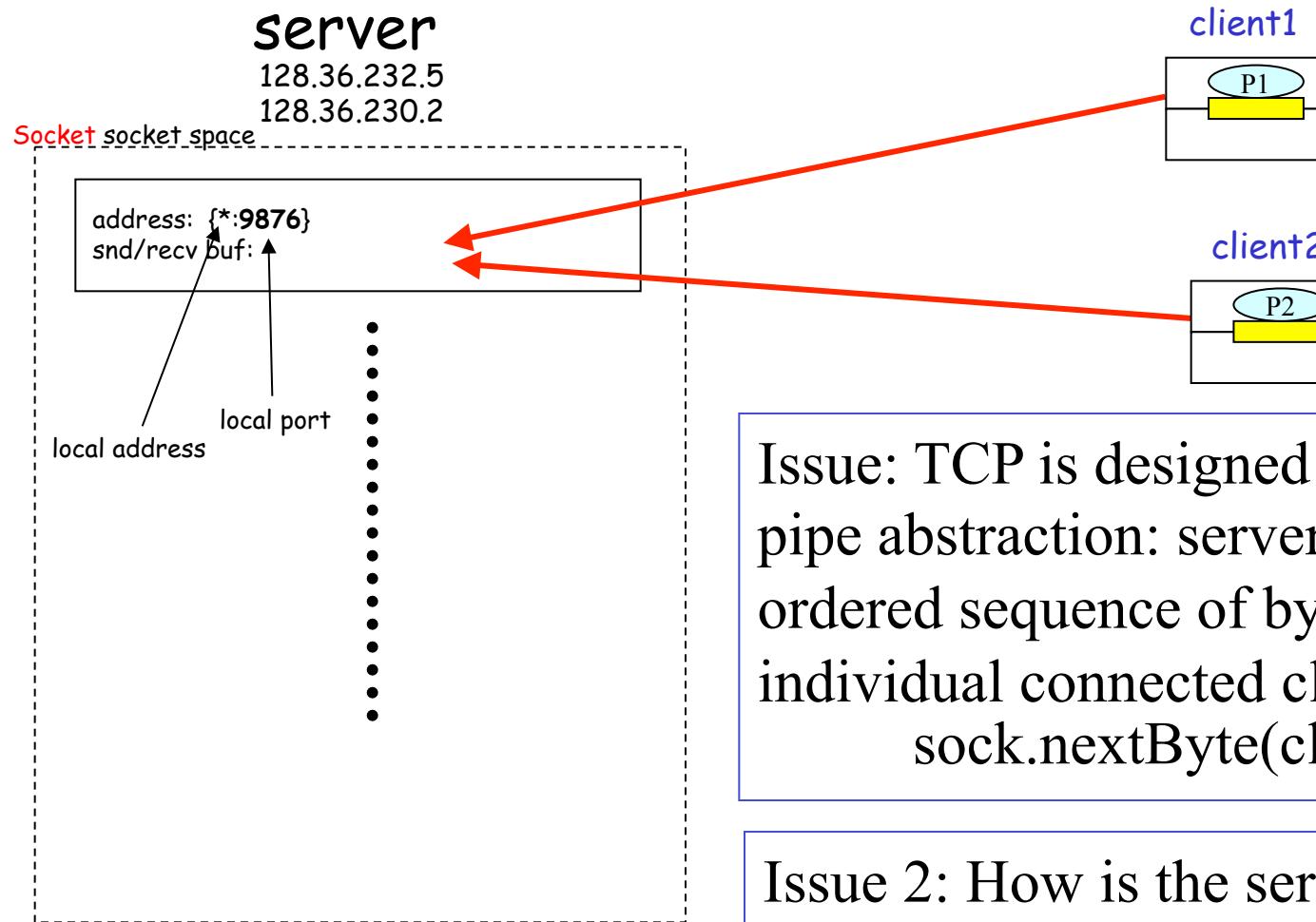
- Standard UDP demultiplexing (find out return address by src.addr/src.port of UDP packet) does not always work
- DNS solution:
identification: remember the mapping

Identification	Flags	12 bytes
Number of questions	Number of answer RRs	
Number of authority RRs	Number of additional RRs	
Questions (variable number of questions)		Name, type fields for a query
Answers (variable number of resource records)		RRs in response to query
Authority (variable number of resource records)		Records for authoritative servers
Additional information (variable number of resource records)		Additional "helpful" info that may be used

Outline

- Recap
- Network application programming
 - Overview
 - UDP
 - Basic TCP

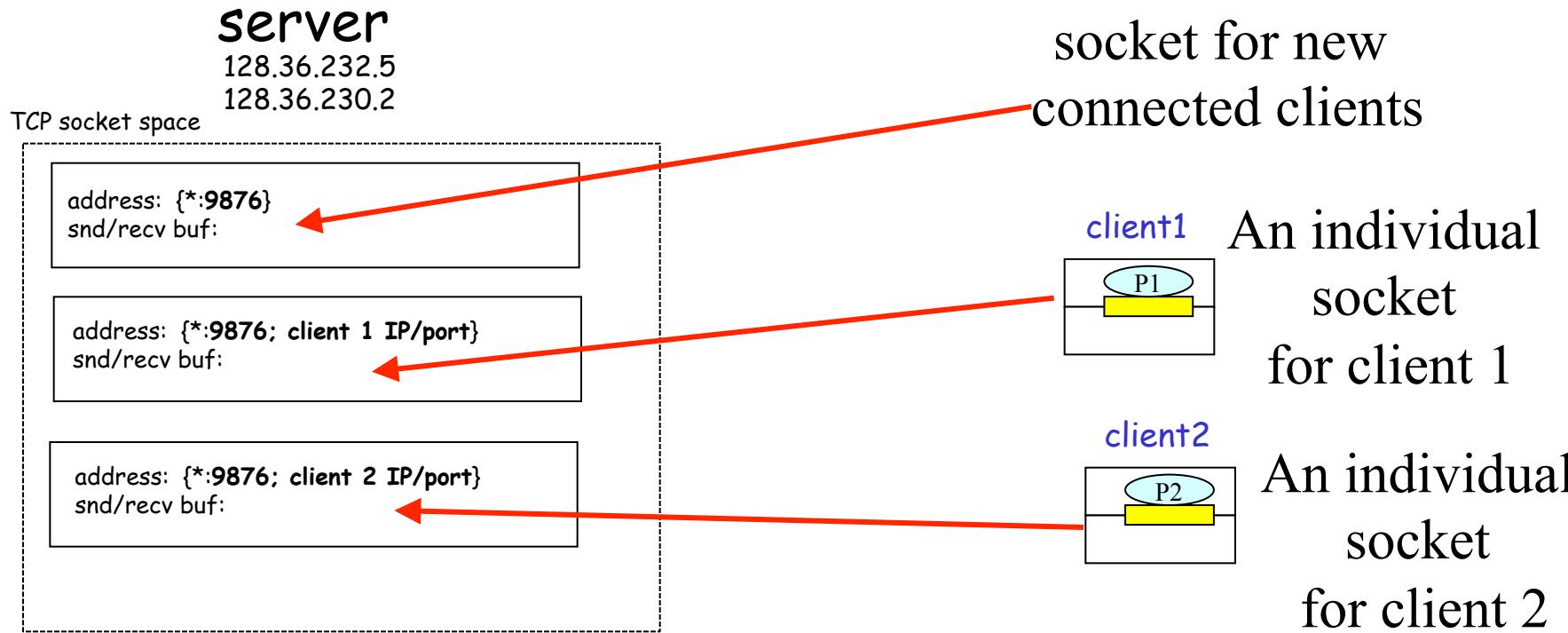
TCP Socket Design: Starting w/ UDP



Issue: TCP is designed to provide a pipe abstraction: server reads an ordered sequence of bytes from each individual connected client
`sock.nextByte(client1)?`

Issue 2: How is the server notified that a new client is connected?
`newClient = sock.getNewClient()?`

BSD TCP Socket API Design



Q: How to decide where to put a new packet?

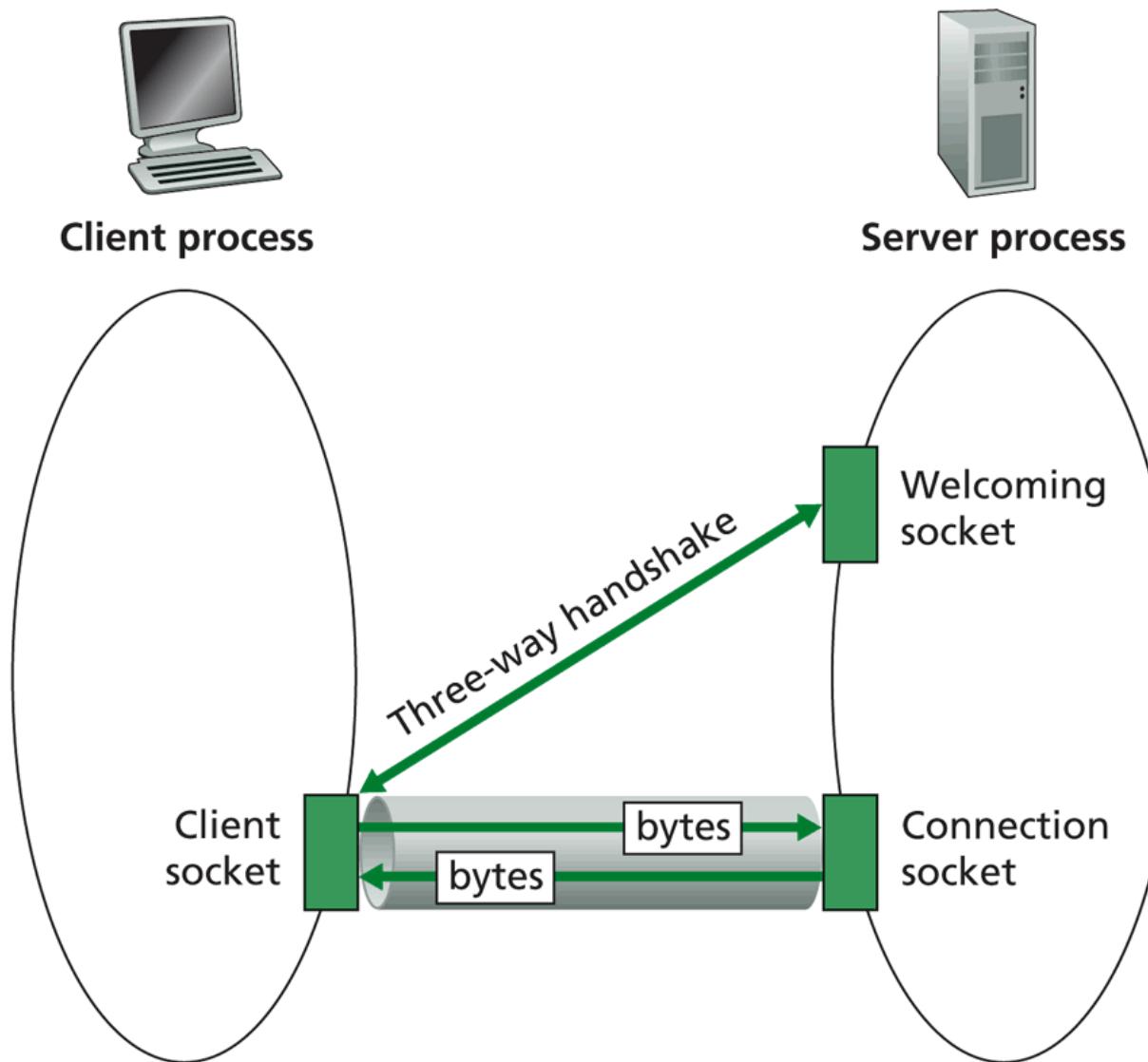
A: Packet demultiplexing is based on four tuples:
(dst addr, dst port, src addr, src port)

TCP Connection-Oriented Demux

- TCP socket identified by 4-tuple:
 - source IP address
 - source port number
 - dest IP address
 - dest port number
- recv host uses all four values to direct segment to appropriate socket
 - different connections/sessions are automatically separated into different sockets

-Welcome socket: the waiting room
-connSocket: the operation room

TCP Socket Big Picture



Client/server Socket Workflow: TCP

Server (running on hostid)

```
create socket,  
port=x, for  
incoming request:  
welcomeSocket =  
ServerSocket(x)
```

```
wait for incoming  
connection request  
connectionSocket =  
welcomeSocket.accept()
```

```
read request from  
connectionSocket
```

```
write reply to  
connectionSocket
```

```
close  
connectionSocket
```

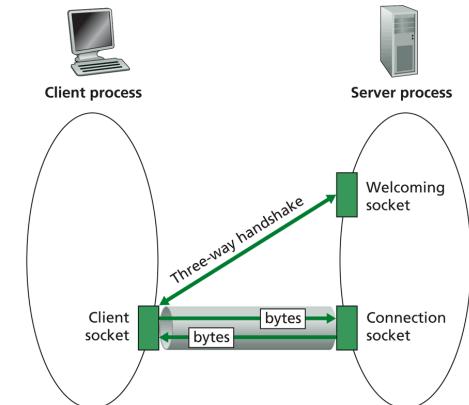
Client

```
create socket,  
connect to hostid, port=x  
clientSocket =  
Socket()
```

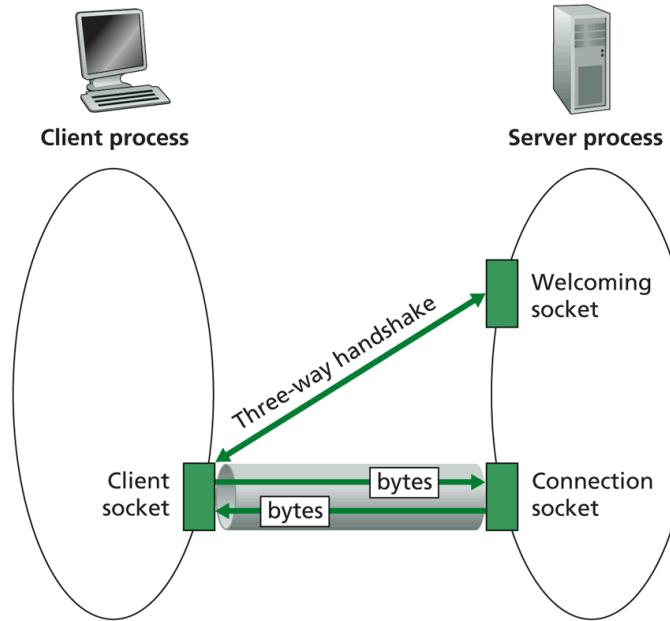
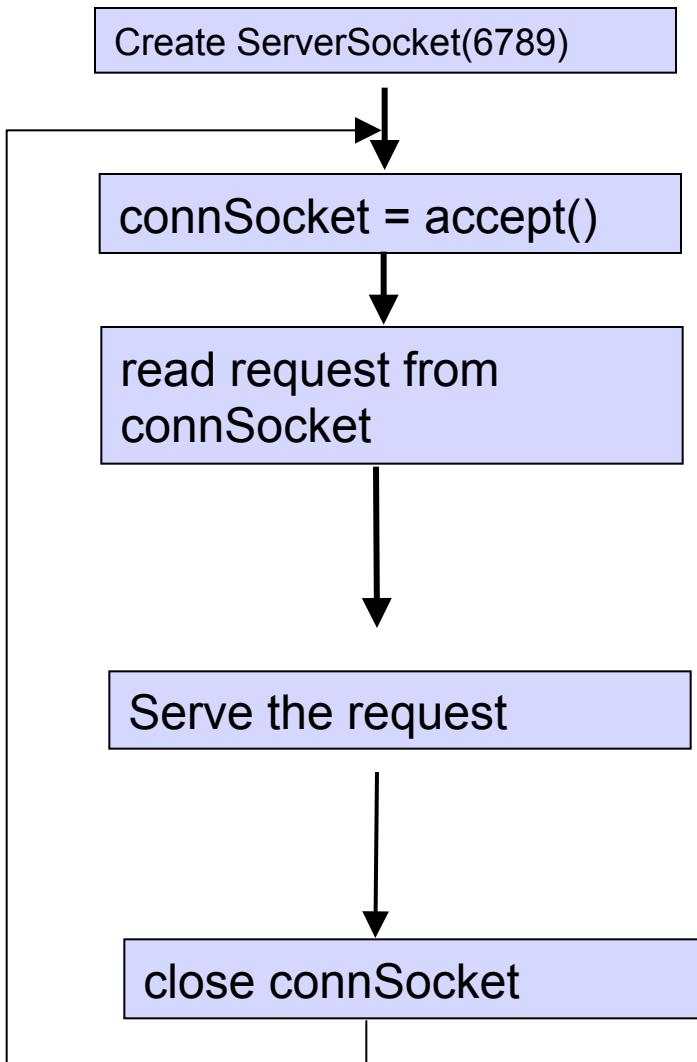
```
send request using  
clientSocket
```

```
read reply from  
clientSocket
```

```
close  
clientSocket
```



Server Flow



-Welcome socket: the waiting room
-connSocket: the operation room

ServerSocket

- **ServerSocket()**
 - creates an unbound server socket.
- **ServerSocket(int port)**
 - creates a server socket, bound to the specified port.
- **ServerSocket(int port, int backlog)**
 - creates a server socket and binds it to the specified local port number, with the specified backlog.
- **ServerSocket(int port, int backlog, InetAddress bindAddr)**
 - creates a server with the specified port, listen backlog, and local IP address to bind to.

- **bind(SocketAddress endpoint)**
 - binds the ServerSocket to a specific address (IP address and port number).
- **bind(SocketAddress endpoint, int backlog)**
 - binds the ServerSocket to a specific address (IP address and port number).

- **Socket accept()**
 - listens for a connection to be made to this socket and accepts it.

- **close()**
 - closes this socket.

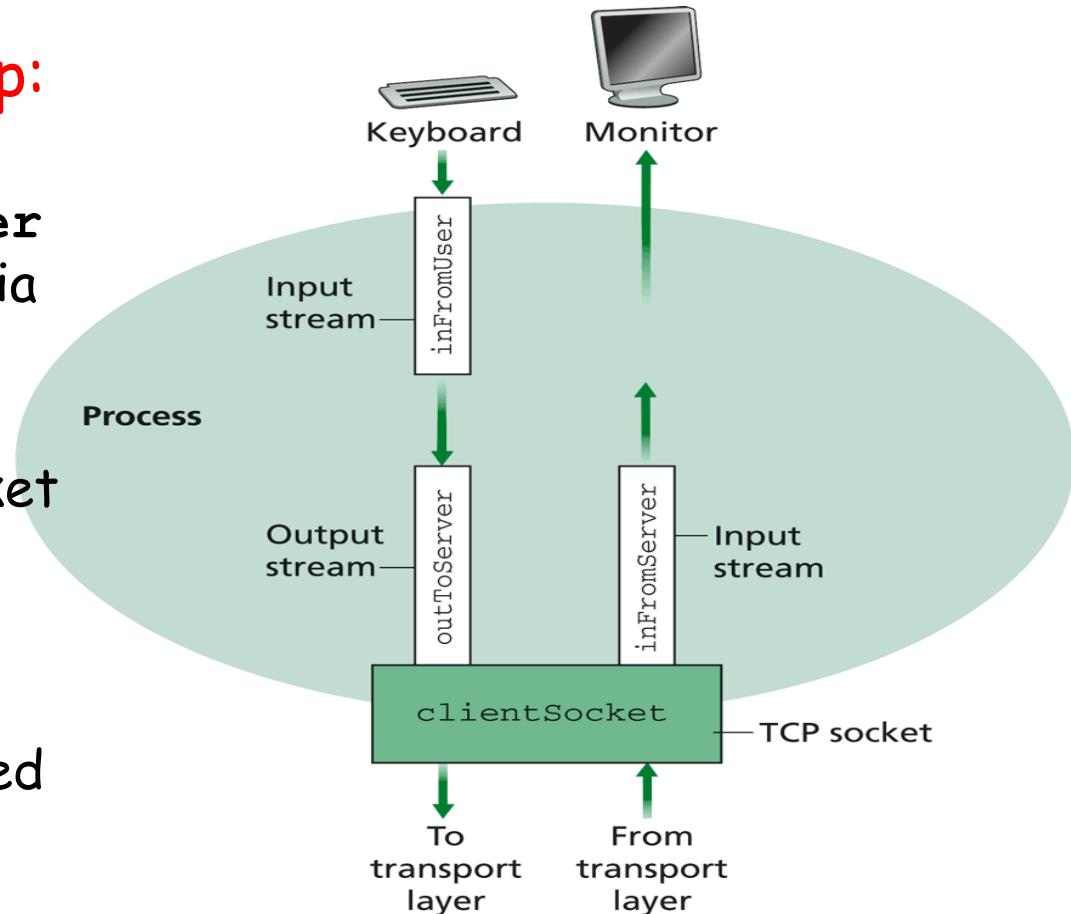
(Client) Socket

- **Socket(InetAddress address, int port)**
creates a stream socket and connects it to the specified port number at the specified IP address.
- **Socket(InetAddress address, int port, InetAddress localAddr, int localPort)**
creates a socket and connects it to the specified remote address on the specified remote port.
- **Socket(String host, int port)**
creates a stream socket and connects it to the specified port number on the named host.
- **bind(SocketAddress bindpoint)**
binds the socket to a local address.
- **connect(SocketAddress endpoint)**
connects this socket to the server.
- **connect(SocketAddress endpoint, int timeout)**
connects this socket to the server with a specified timeout value.
- **InputStream get InputStream()**
returns an input stream for this socket.
- **OutputStream get OutputStream()**
returns an output stream for this socket.
- **close()**
closes this socket.

Simple TCP Example

Example client-server app:

- 1) client reads line from standard input (`inFromUser` stream), sends to server via socket (`outToServer` stream)
- 2) server reads line from socket
- 3) server converts line to uppercase, sends back to client
- 4) client reads, prints modified line from socket (`inFromServer` stream)



Example: Java client (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        Create input stream → BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));
        sentence = inFromUser.readLine();

        Create client socket, connect to server → Socket clientSocket = new Socket("server.name", 6789);

        Create output stream attached to socket → DataOutputStream outToServer =
            new DataOutputStream(clientSocket.getOutputStream());
    }
}
```

OutputStream

- **public abstract class OutputStream**
 - **public abstract void write(int b) throws IOException**
 - **public void write(byte[] data) throws IOException**
 - **public void write(byte[] data, int offset, int length) throws IOException**
 - **public void flush() throws IOException**
 - **public void close() throws IOException**

InputStream

- public abstract class InputStream
 - public abstract int read() throws IOException
 - public int read(byte[] input) throws IOException
 - public int read(byte[] input, int offset, int length) throws IOException
 - public long skip(long n) throws IOException
 - public int available() throws IOException
 - public void close() throws IOException

Example: Java client (TCP), cont.

```
Send line  
to server ]→ outToServer.writeBytes(sentence + '\n');

Create  
input stream  
attached to socket ]→ BufferedReader inFromServer =  
new BufferedReader(new  
InputStreamReader(clientSocket.getInputStream()));

modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();

}

}
```

Example: Java server (TCP)

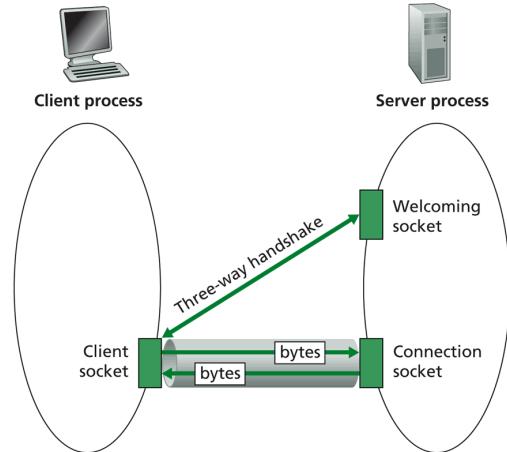
```
import java.io.*;
import java.net.*;

class TCPServer {

    public static void main(String argv[]) throws Exception
    {
        String clientSentence;
        String capitalizedSentence;

        ServerSocket welcomeSocket = new ServerSocket(6789);
```

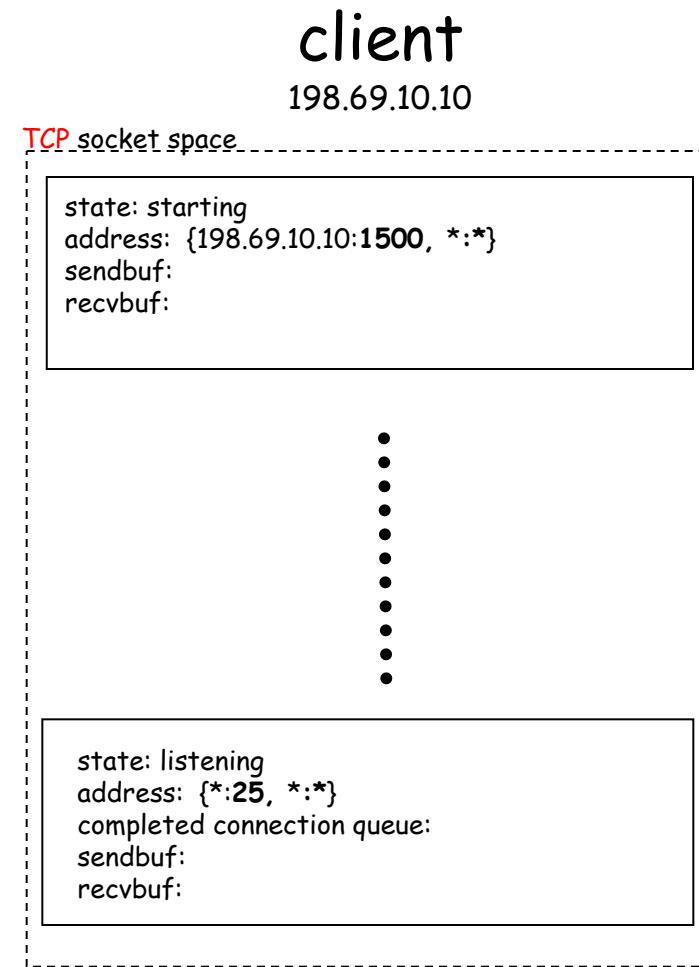
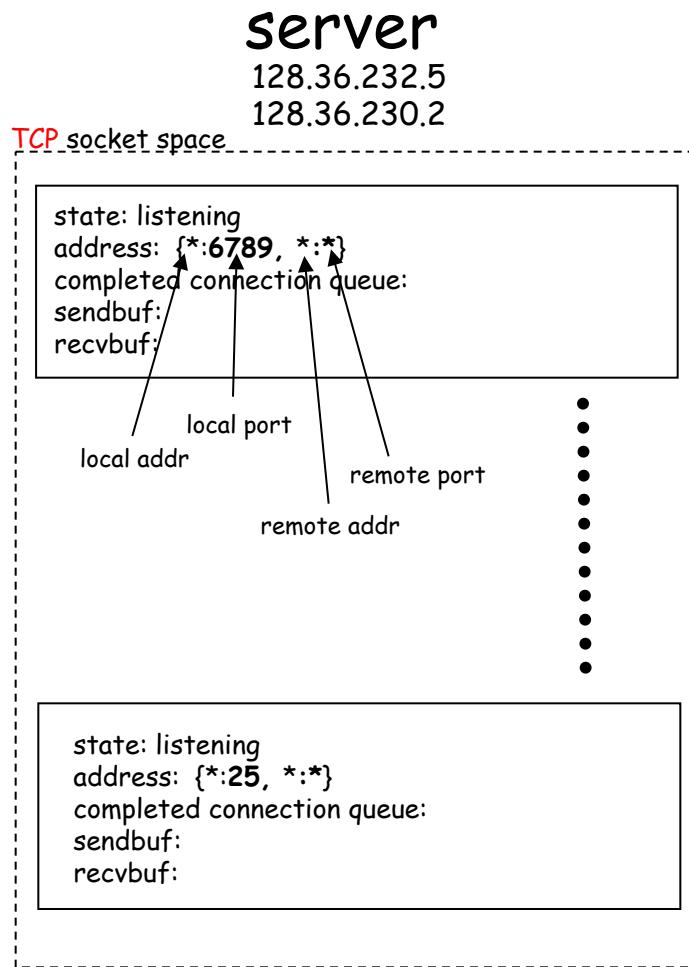
Create welcoming socket at port 6789



Demo

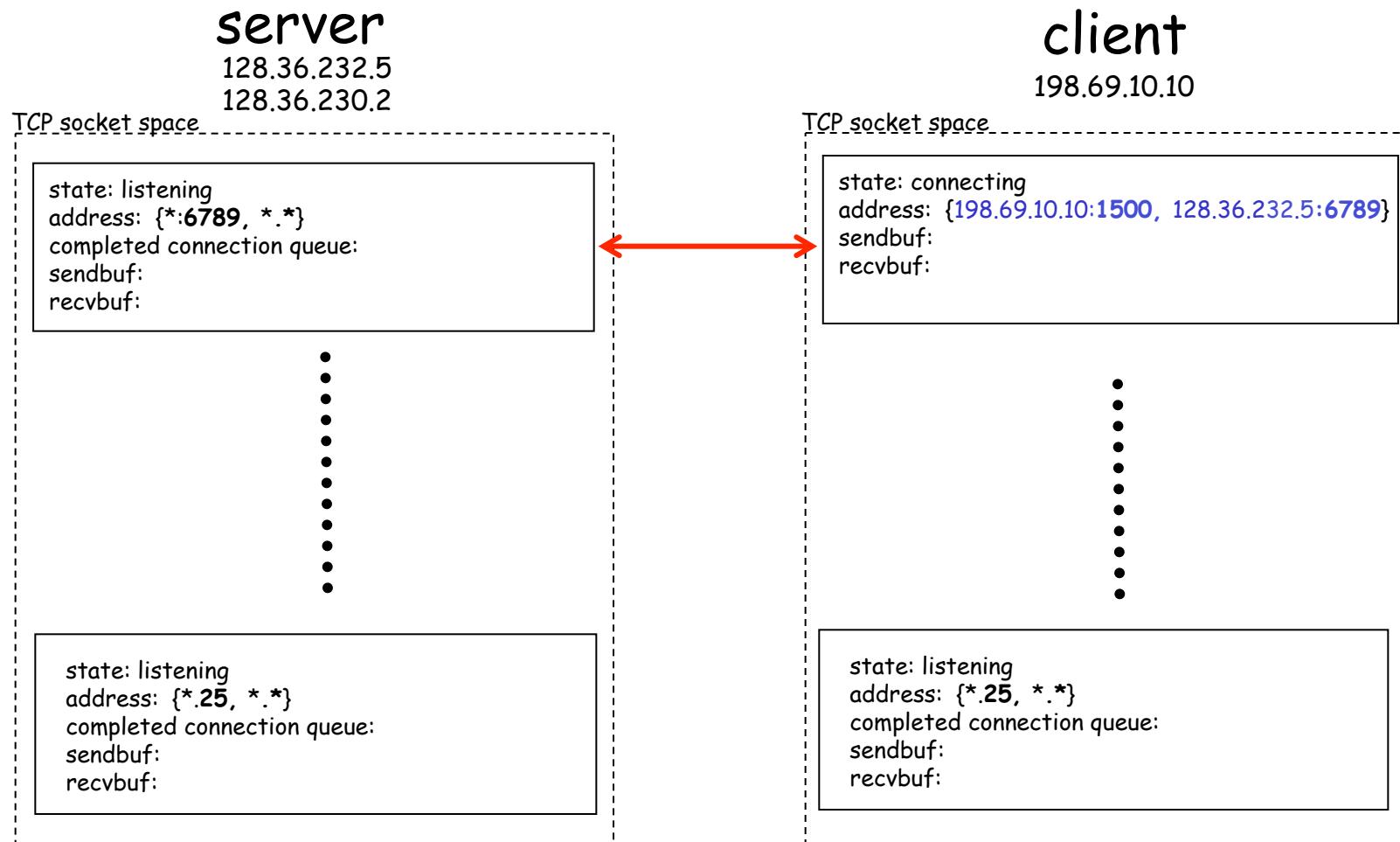
```
% wireshark to capture our TCP traffic  
tcp.srcport==6789 or tcp.dstport==6789
```

Under the Hood: After Welcome (Server) Socket



%netstat -p tcp -n -a

Client Initiates Connection



Example: Client Connection

Handshake Done

server

128.36.232.5
128.36.230.2

TCP socket space

```
state: listening
address: {*:6789, *:*}
completed connection queue:
{128.36.232.5:6789, 198.69.10.10.1500}
sendbuf:
recvbuf:
```

•
•
•
•
•
•
•

```
state: listening
address: {*:25, *:*}
completed connection queue:
sendbuf:
recvbuf:
```

client

198.69.10.10

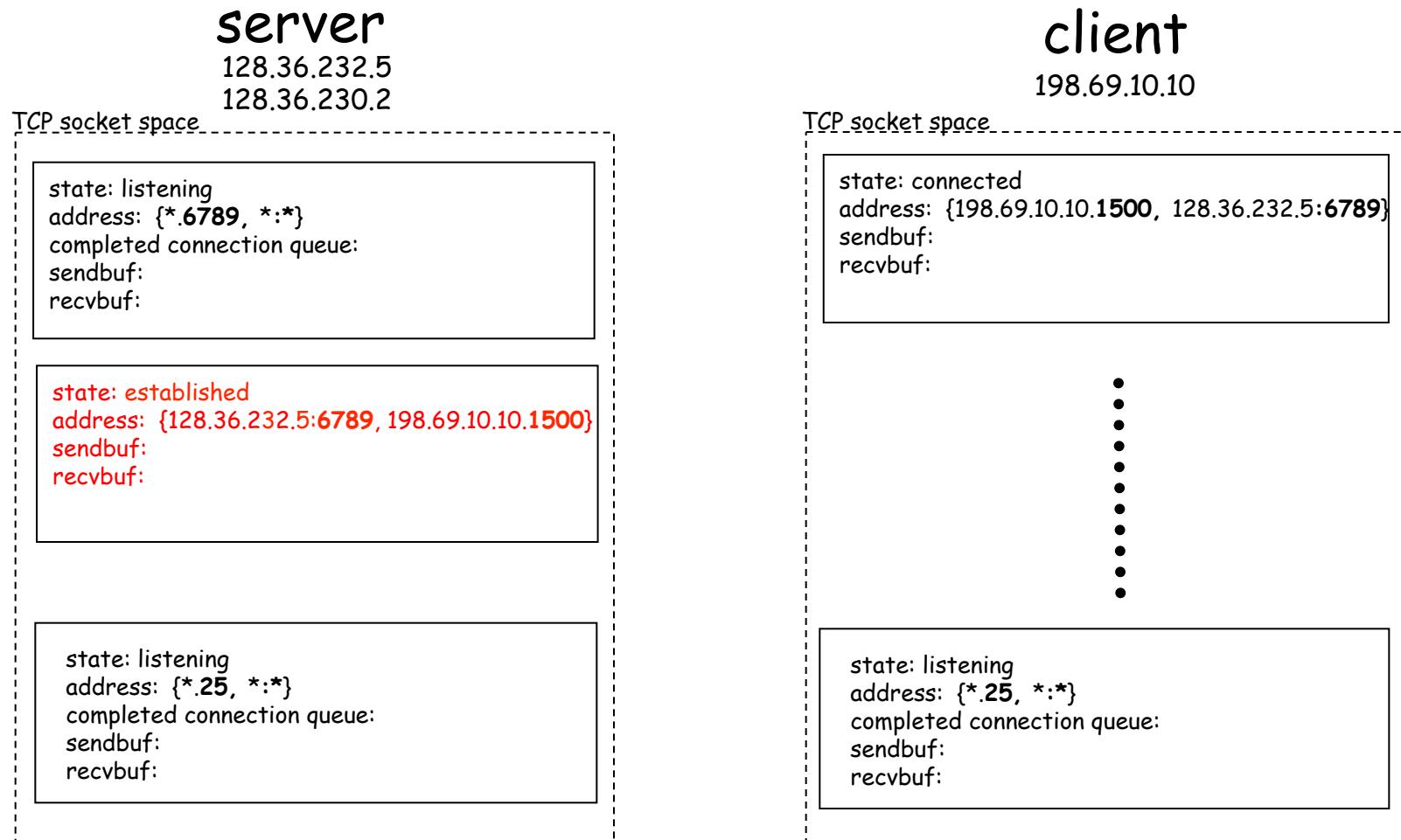
TCP socket space

```
state: connected
address: {198.69.10.10:1500, 128.36.232.5:6789}
sendbuf:
recvbuf:
```

•
•
•
•
•
•

```
state: listening
address: {*:25, *:*}
completed connection queue:
sendbuf:
recvbuf:
```

Example: Client Connection Handshake Done



Packet demultiplexing is based on (dst addr, dst port, src addr, src port)
Packet sent to the socket with **the best match!**

Demo

- ❑ What if more client connections than backlog allowed?

Example: Java server (TCP)

```
import java.io.*;
import java.net.*;

class TCPServer {

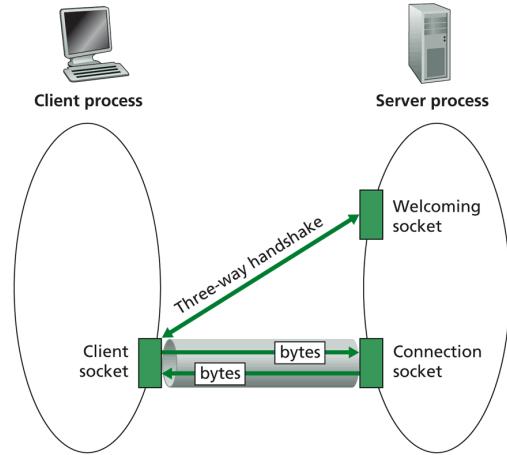
    public static void main(String argv[]) throws Exception
    {
        String clientSentence;
        String capitalizedSentence;

        ServerSocket welcomeSocket = new ServerSocket(6789);

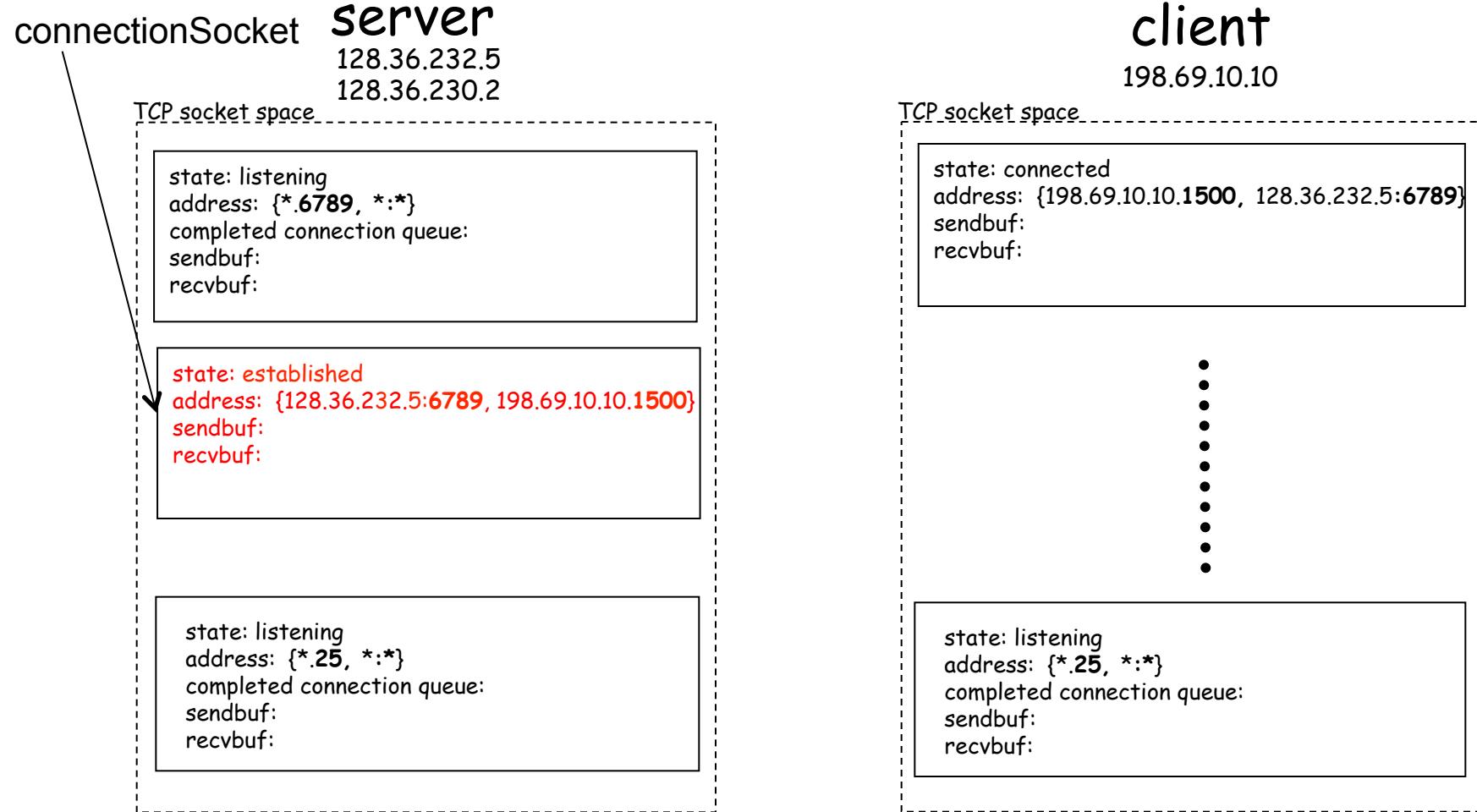
        while(true) {
            Socket connectionSocket = welcomeSocket.accept();

```

Wait, on welcoming socket for contact by client



Example: Server accept()



Example: Java server (TCP):

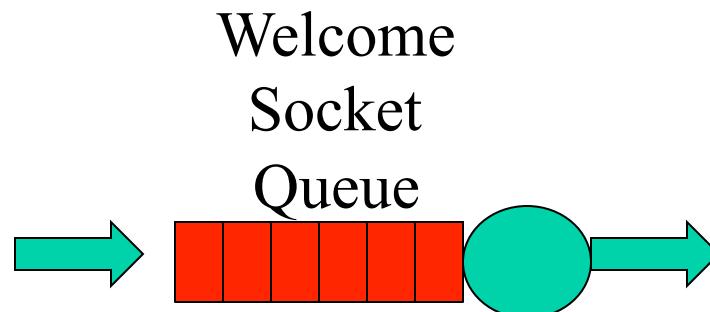
Processing

```
Create input stream, attached to socket → BufferedReader inFromClient =  
new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));  
  
Read in line from socket → clientSentence = inFromClient.readLine();  
capitalizedSentence = clientSentence.toUpperCase() + '\n';  
  
}  
}  
}
```

Example: Java server (TCP): Output

Analysis

- Assume that client requests arrive at a rate of λ /second
- Assume that each request takes $1/\mu$ seconds
- Some basic questions
 - How big is the backlog (welcome queue)



Analysis

- ❑ Is there any interop issue in the sample program?

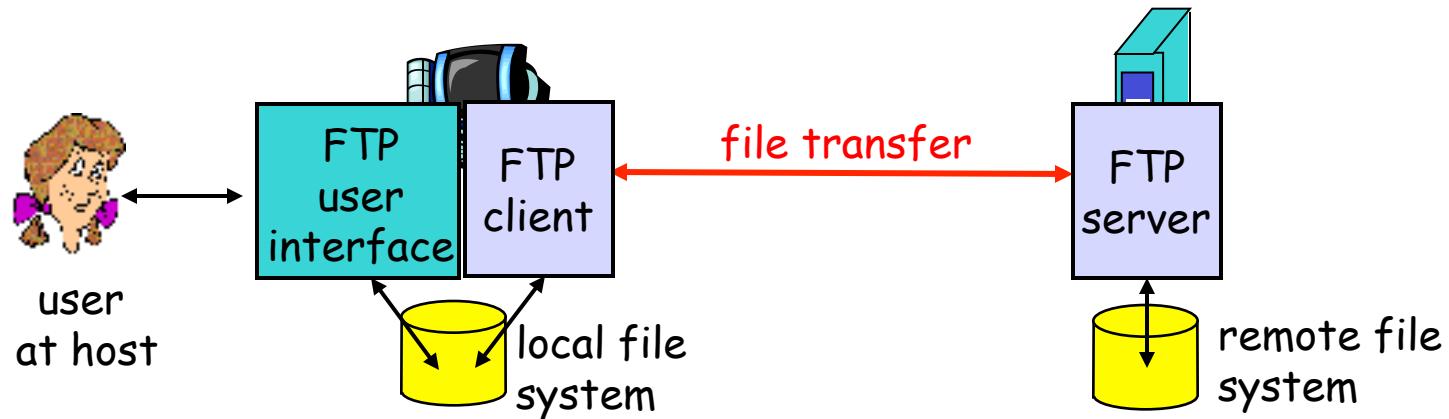
Analysis

- Is there any interop issue in the sample program?
 - DataOutputStream writeBytes(String) truncates
 - [http://docs.oracle.com/javase/1.4.2/docs/api/java/io/DataOutputStream.html#writeBytes\(java.lang.String\)](http://docs.oracle.com/javase/1.4.2/docs/api/java/io/DataOutputStream.html#writeBytes(java.lang.String))

Outline

- Recap
- Network application programming
- FTP

FTP: the File Transfer Protocol



- Transfer files to/from remote host
- Client/server model
 - client*: side that initiates transfer (either to/from remote)
 - server*: remote host
- ftp: RFC 959
- ftp server: port 21/20 (smtp 25, http 80)

FTP Commands, Responses

Sample commands:

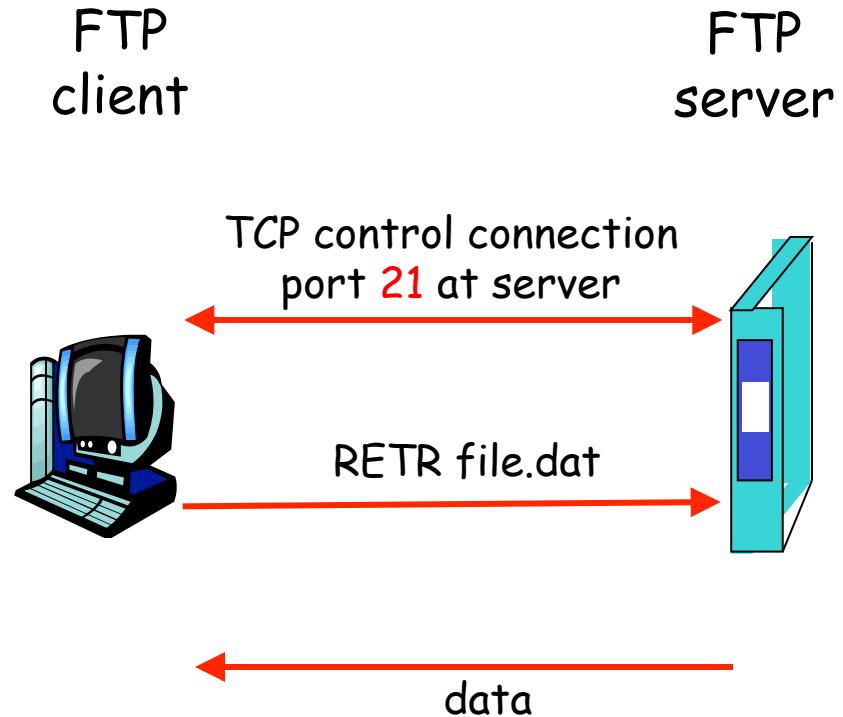
- sent as ASCII text over control channel
- USER *username***
- PASS *password***
- PWD** returns current dir
- STAT** shows server status
- LIST** returns list of file in current directory
- RETR *filename*** retrieves (gets) file
- STOR *filename*** stores file

Sample return codes

- status code and phrase
- 331 Username OK, password required**
- 125 data connection already open; transfer starting**
- 425 Can't open data connection**
- 452 Error writing file**

FTP Protocol Design

- What is the simplest design of data transfer?



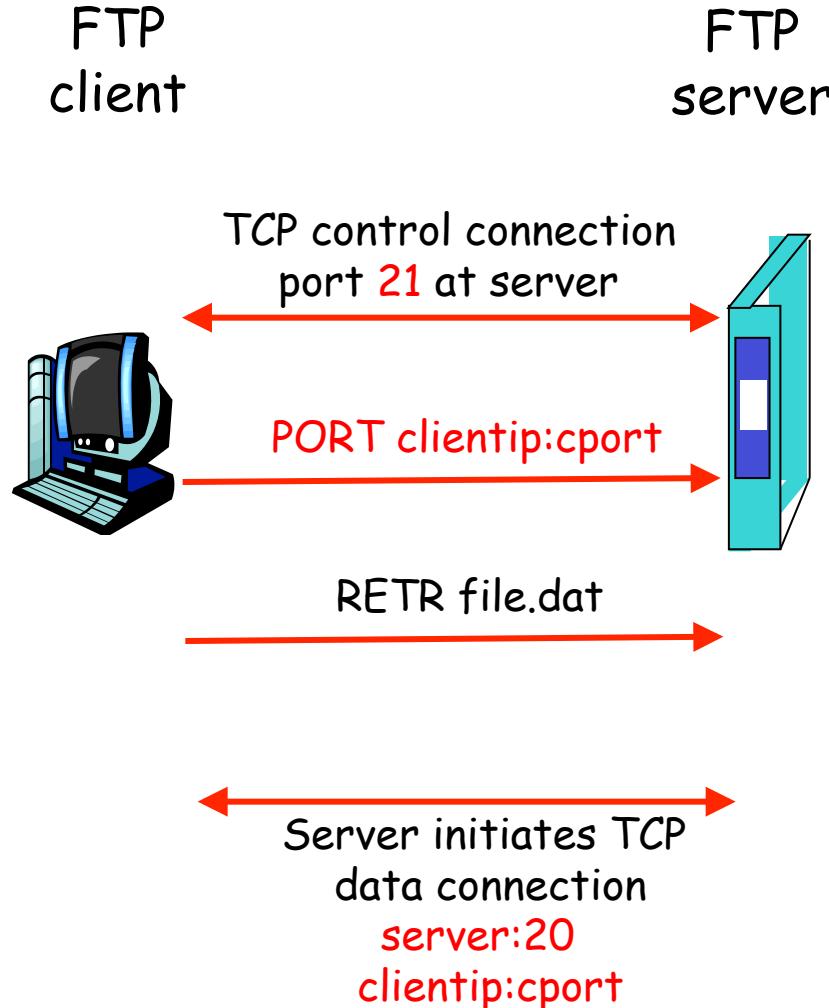
FTP: A Client-Server Application with Separate Control, Data Connections

- Two types of TCP connections opened:
 - A control connection: exchange commands, responses between client, server.
“out of band control”
 - Data connections: each for file data to/
from server

Discussion: why does FTP separate control/data connections?

Q: How to create a new data connection?

Traditional FTP: Client Specifies Port for Data Connection



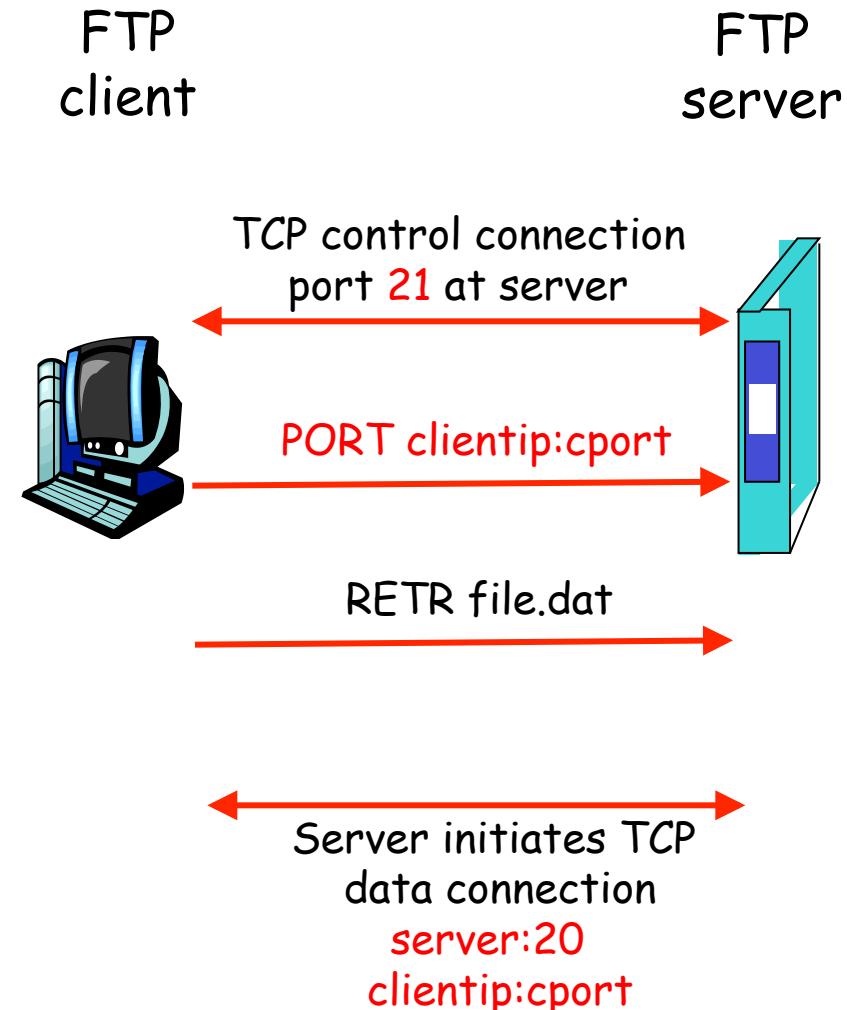
Example using telnet/nc

- Use telnet for the control channel
 - telnet ftp.freebsd.org 21
 - user, pass
 - port 172,27,5,145,4,1
 - list

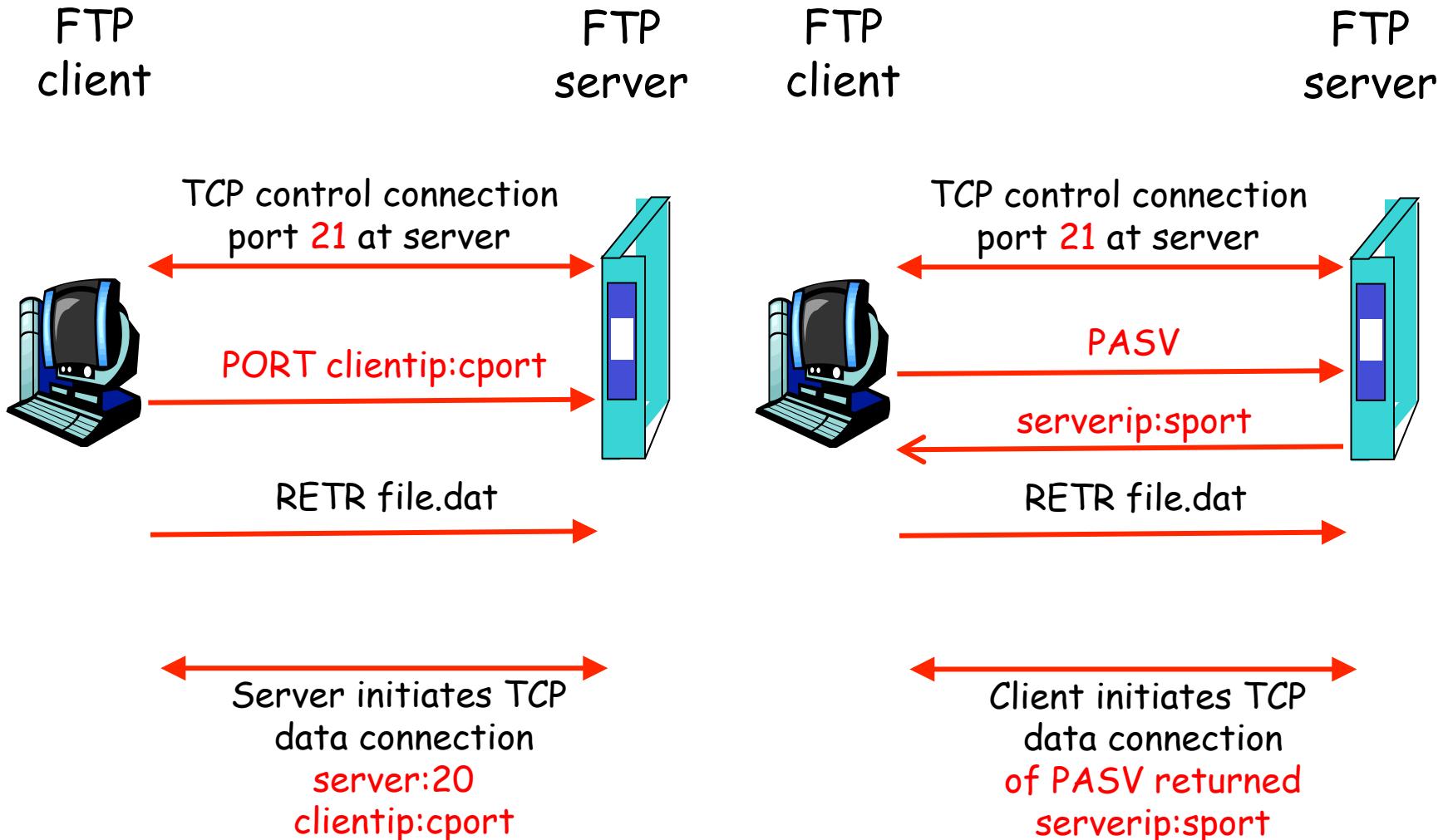
- use nc (NetCat) to receive/send data with server
 - nc -v -l 1025

Problem of the Client PORT Approach

- Many Internet hosts are behind **NAT/firewalls** that block connections initiated from outside



FTP PASV: Server Specifies Data Port, Client Initiates Connection



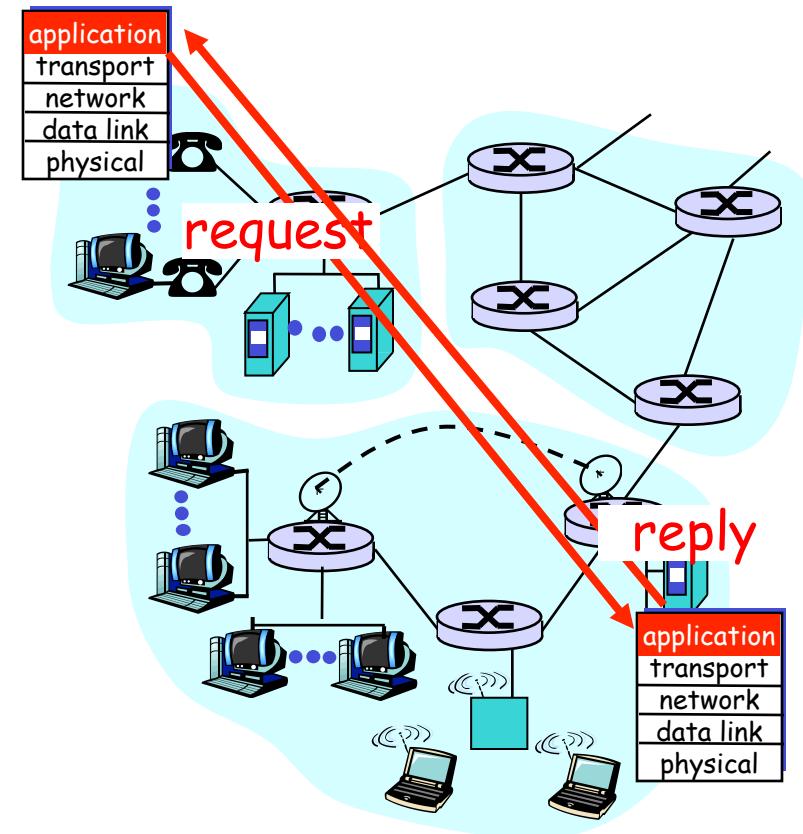
Example

- Use Wireshark to capture traffic
 - Using chrome to visit
<ftp://ftp.freebsd.org/pub/FreeBSD/README.TXT>

FTP Evaluation

Key questions to ask about a C-S application

- Is the application **extensible**?
- Is the application **scalable**?
- How does the application handle server failures (being **robust**)?
- How does the application provide **security**?

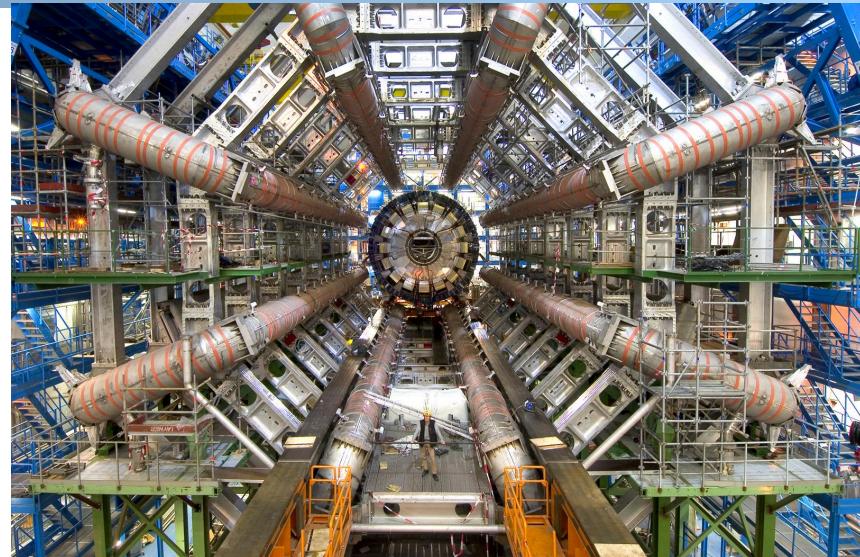


What are some interesting design features of the FTP protocol?

FTP Extensions

- ❑ FTP extensions are still being used extensively in large data set transfers (e.g., LHC)

- ❑ See GridFTP to FTP extensions
 - <https://www.ogf.org/documents/GFD.20.pdf>

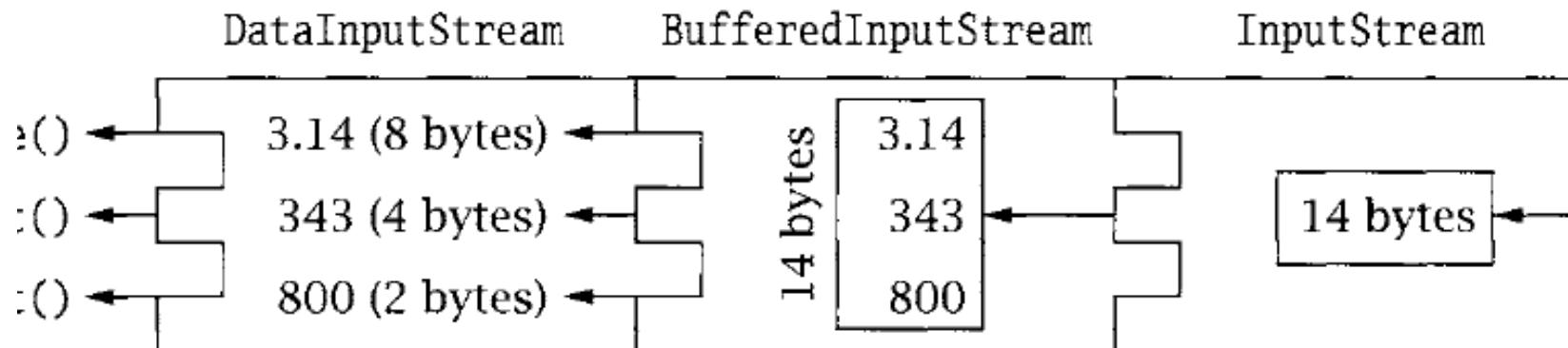
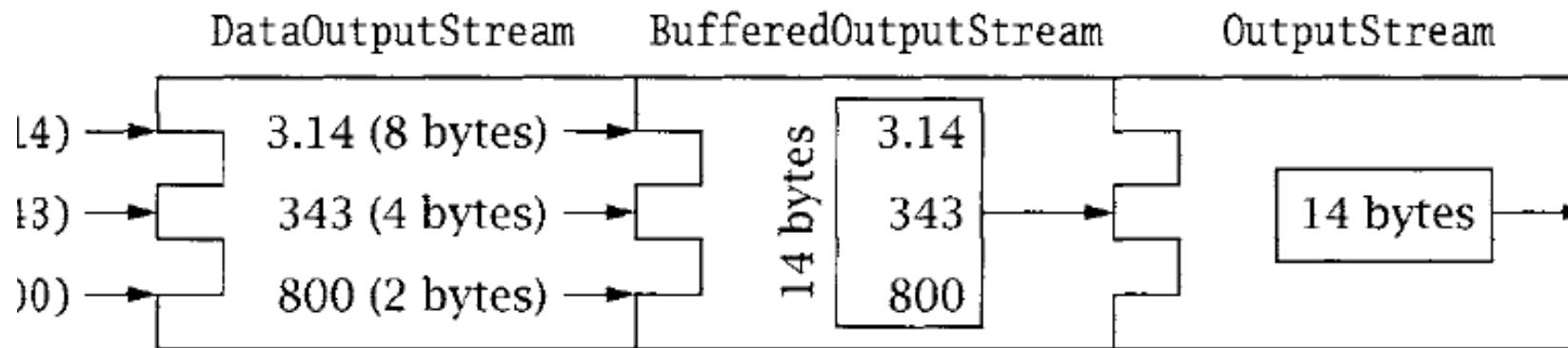


Outline

- Recap
- Network application programming
- FTP
- HTTP

Backup Slides

DataStream



Demo

```
% wireshark to capture traffic  
tcp.srcport==6789 or tcp.dstport==6789
```

Network Applications:

HTTP/1.0

Y. Richard Yang

<http://zoo.cs.yale.edu/classes/cs433/>

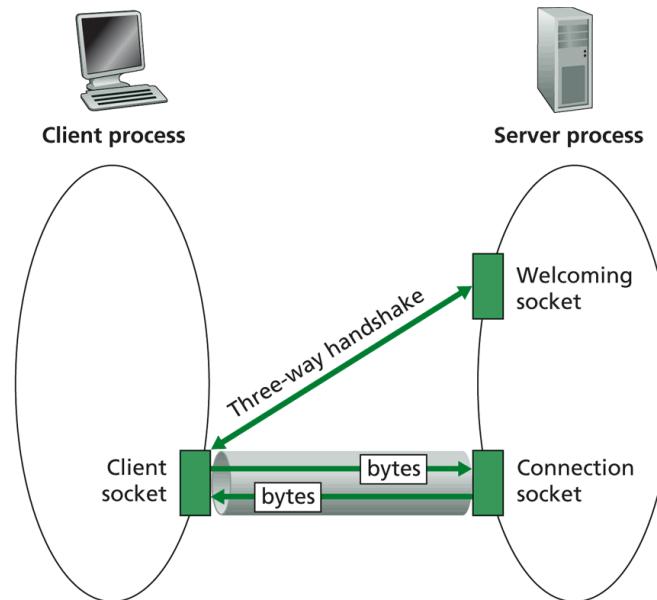
2/10/2016

Admin

- Assignment 2 posted

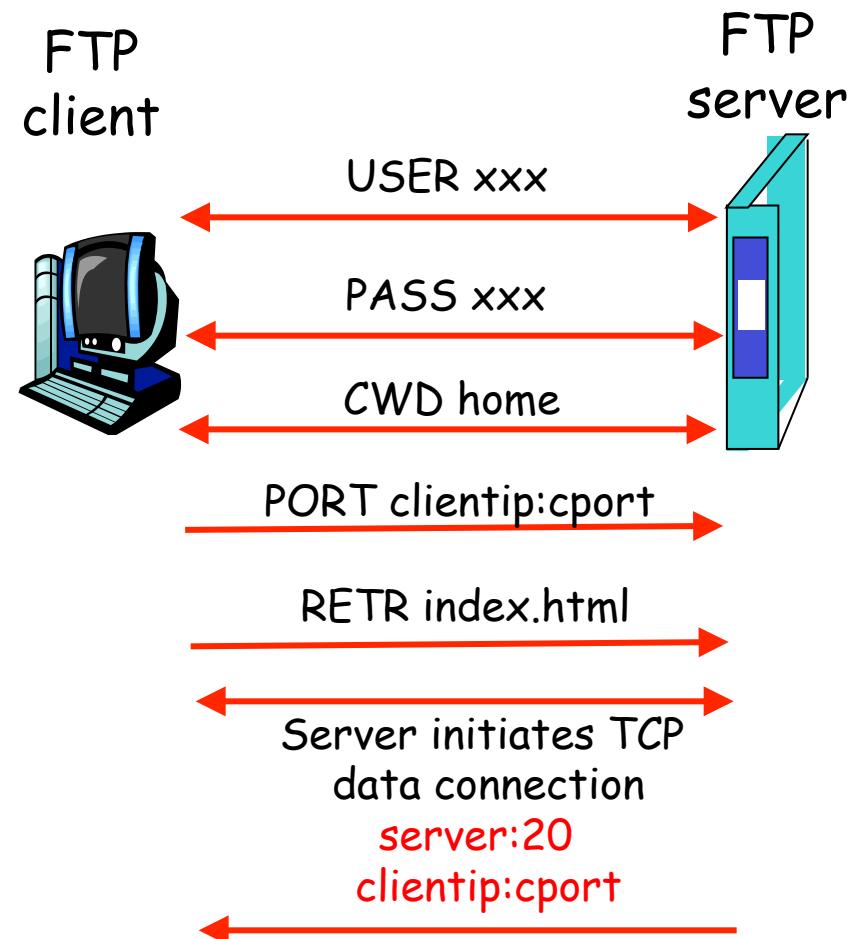
Recap: TCP Sockets

- TCP server socket demux by 4-tuple:
 - source IP address
 - source port number
 - dest IP address
 - dest port number



Recap: FTP

- A stateful protocol
 - state established by commands such as
 - USER/PASS, CWD, TYPE
- Multiple TCP connections
 - A control connection
 - Data connections
 - Two approaches: PORT vs PASV
 - GridFTP: concurrent data connections; block data transfer mode



Outline

- Recap
- HTTP

From Opaque Files to Web Pages

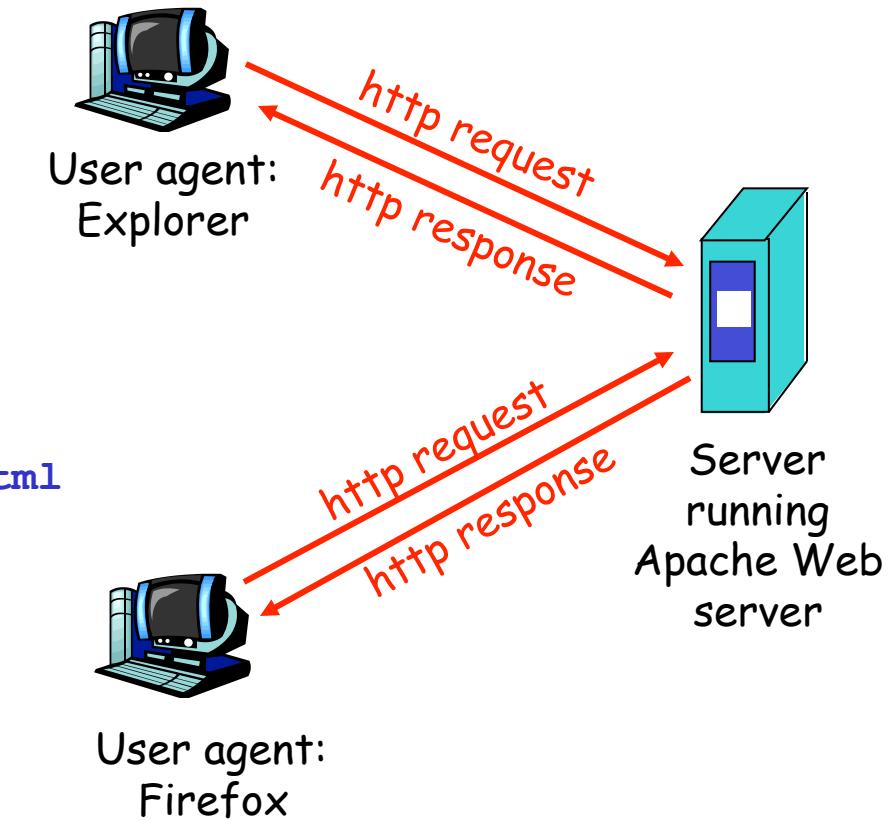
□ Web page:

- authored in HTML
- addressed by a URL
 - URL has two components:
 - host name, port number and
 - path name

`http://www.cs.yale.edu:80/index.html`

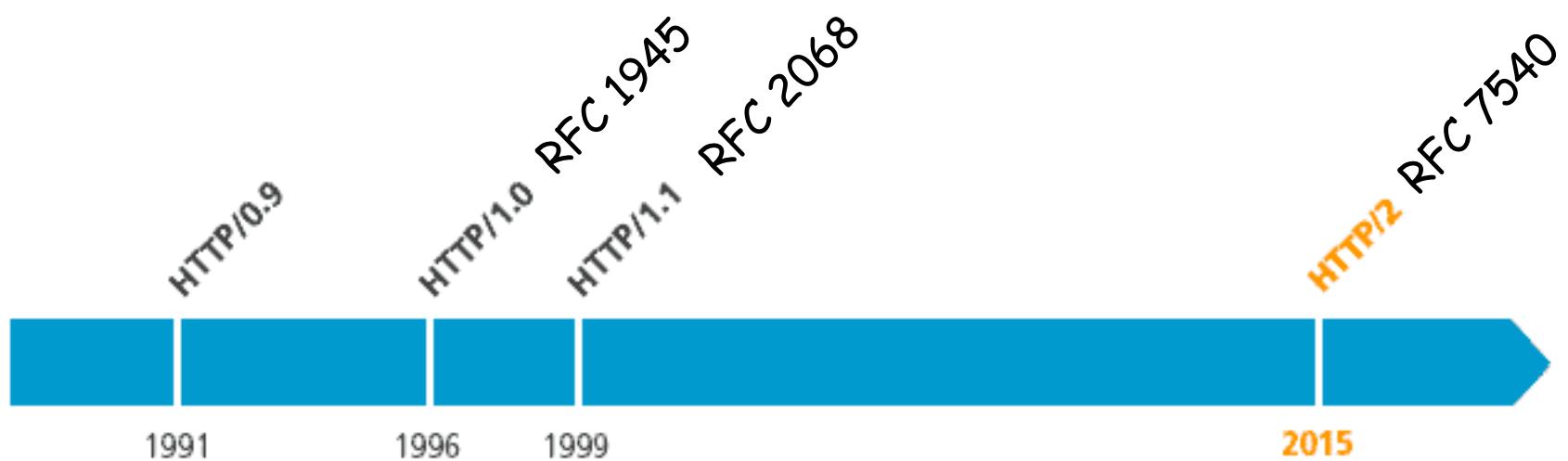
□ Most Web pages consist of:

- base HTML page, and
- several referenced objects



The Web pages are requested through
HTTP: hypertext transfer protocol

HTTP is Still Evolving



HTTP 1.0 Message Flow

- Server waits for requests from clients
- Client initiates TCP connection (creates socket) to server, port 80
- Client sends request for a document
- Web server sends back the document
- TCP connection closed
- Client parses the document to find embedded objects (images)
 - repeat above for each image

HTTP 1.0 Message Flow (more detail)

Suppose user enters URL
www.cs.yale.edu/index.html

1a. http client initiates TCP connection to http server (process) at www.cs.yale.edu. Port 80 is default for http server.

2. http client sends http *request message* (containing URL) into TCP connection socket

0. http server at host www.cs.yale.edu waiting for TCP connection at port 80.

1b. server “accepts” connection, ack. client

3. http server receives request message, forms *response message* containing requested object (index.html), sends message into socket (the sending speed increases slowly, which is called slow-start)

time ↓

HTTP 1.0 Message Flow (cont.)

time ↓

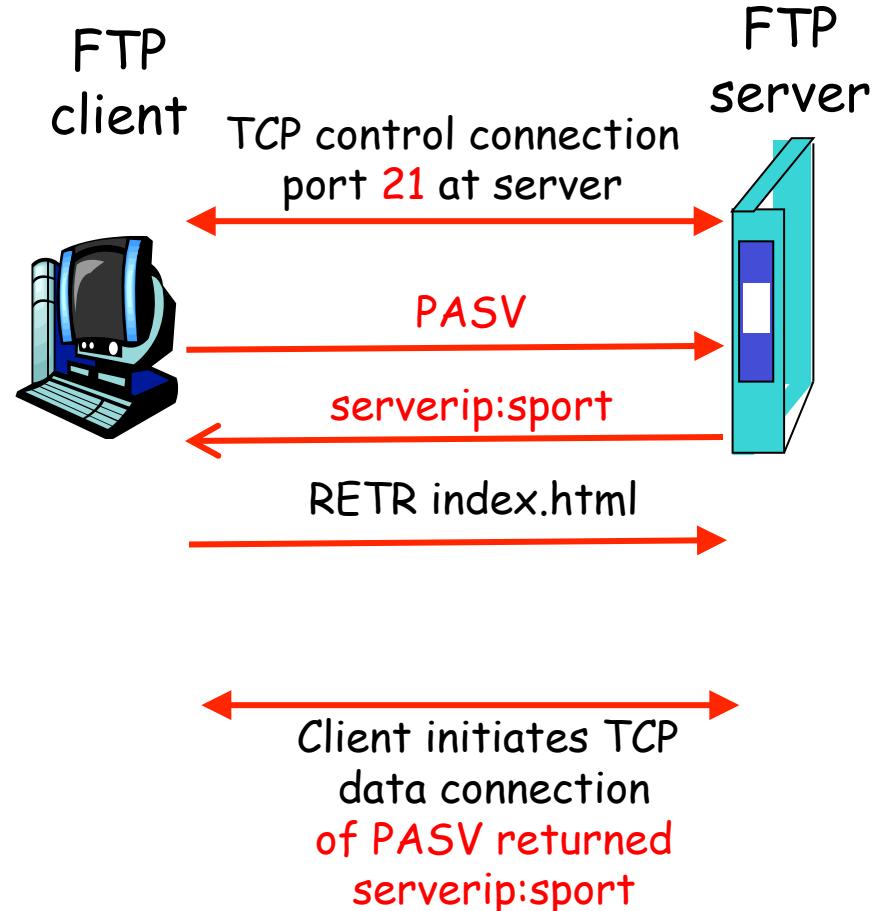
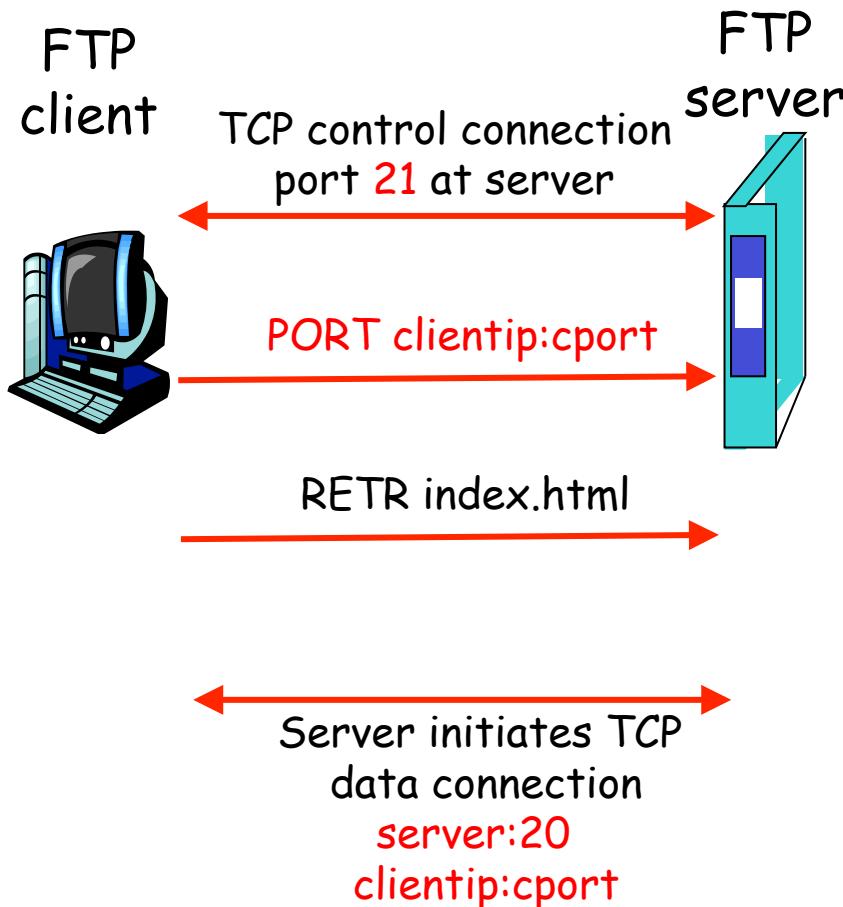
4. http server closes TCP connection.

5. http client receives response message containing html file, parses html file, finds embedded image

6. Steps 1-5 repeated for each of the embedded images

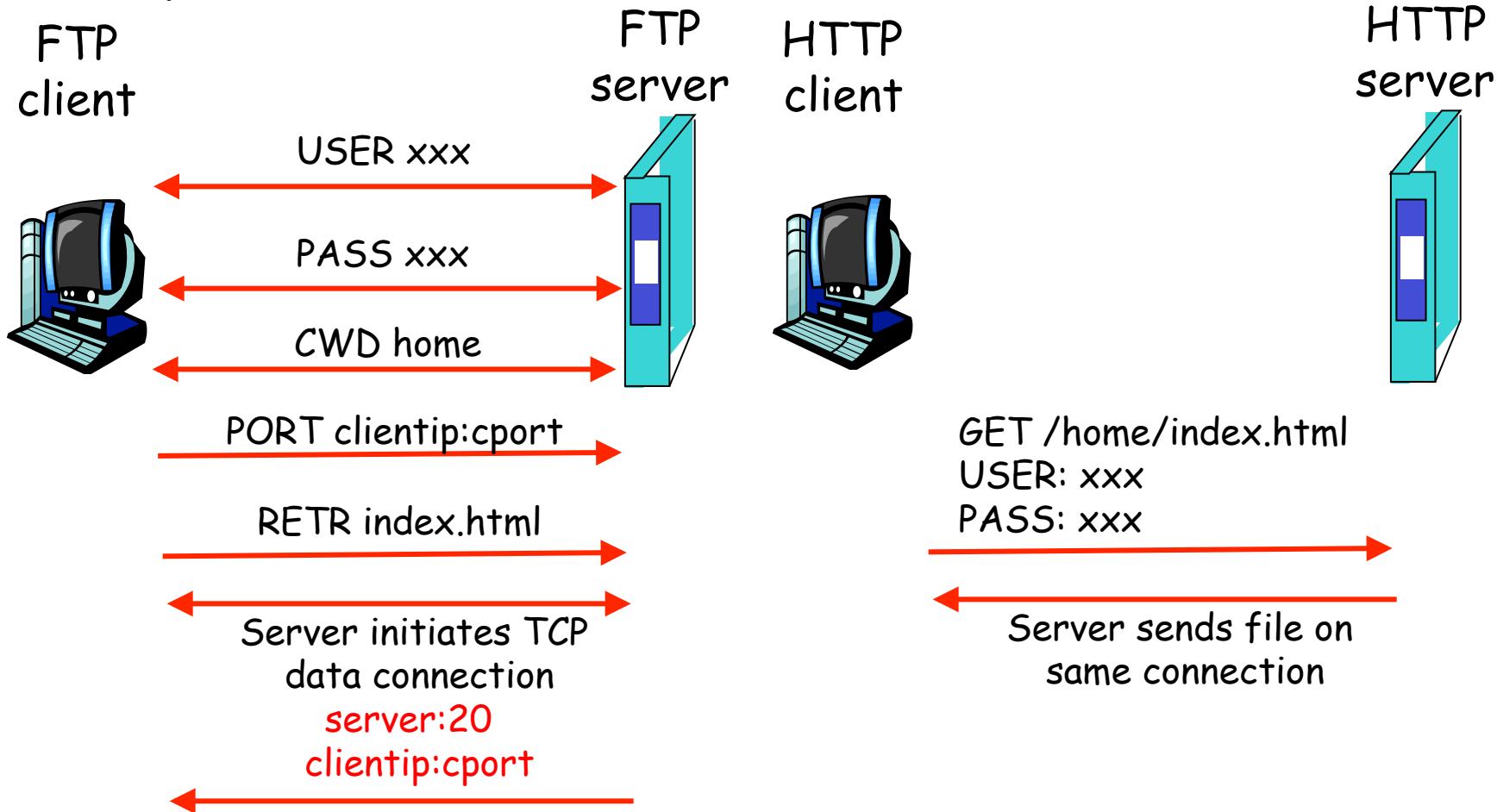
Discussion

- How about we use FTP as HTTP?



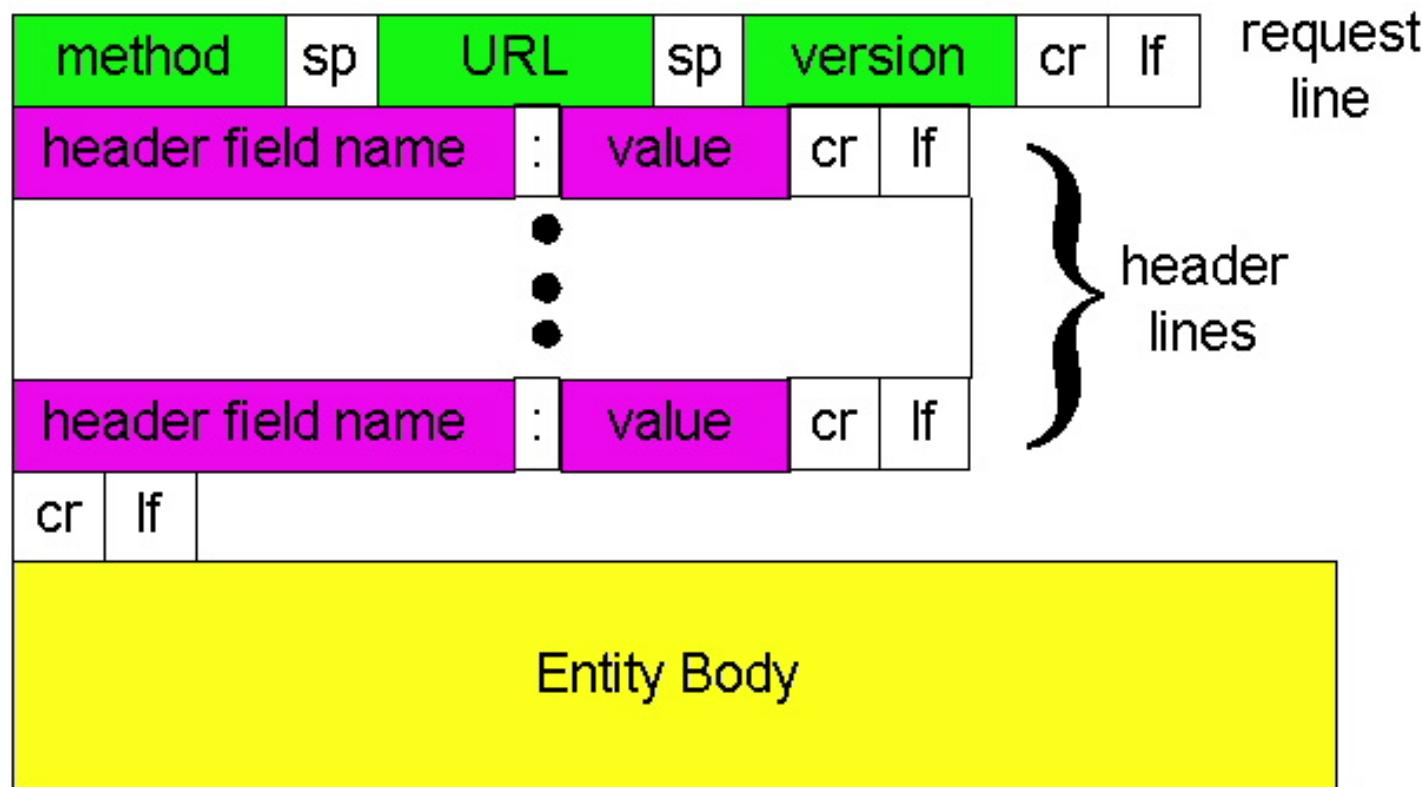
HTTP1.0 Message Flow

- **HTTP1.0 servers are stateless** servers: each request is self-contained



HTTP Request Message: General Format

- ASCII (human-readable format)



Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet www.cs.yale.edu 80
```

Opens TCP connection to port 80
(default http server port) at www.cs.yale.edu.
Anything typed in sent
to port 80 at www.cs.yale.edu

2. Type in a GET http request:

```
GET /index.html HTTP/1.0
```

By typing this in (hit carriage
return twice), you send
this minimal (but complete)
GET request to http server

3. Look at response message sent by the http server.

Trying out HTTP (client side) for yourself

- Try telnet GET on www.yale.edu

HTTP Request Message Example: GET

request line
(GET, POST,
HEAD, PUT,
DELETE,
TRACE ... commands)

header lines

Carriage return,
line feed
indicates end
of message

Virtual host multiplexing

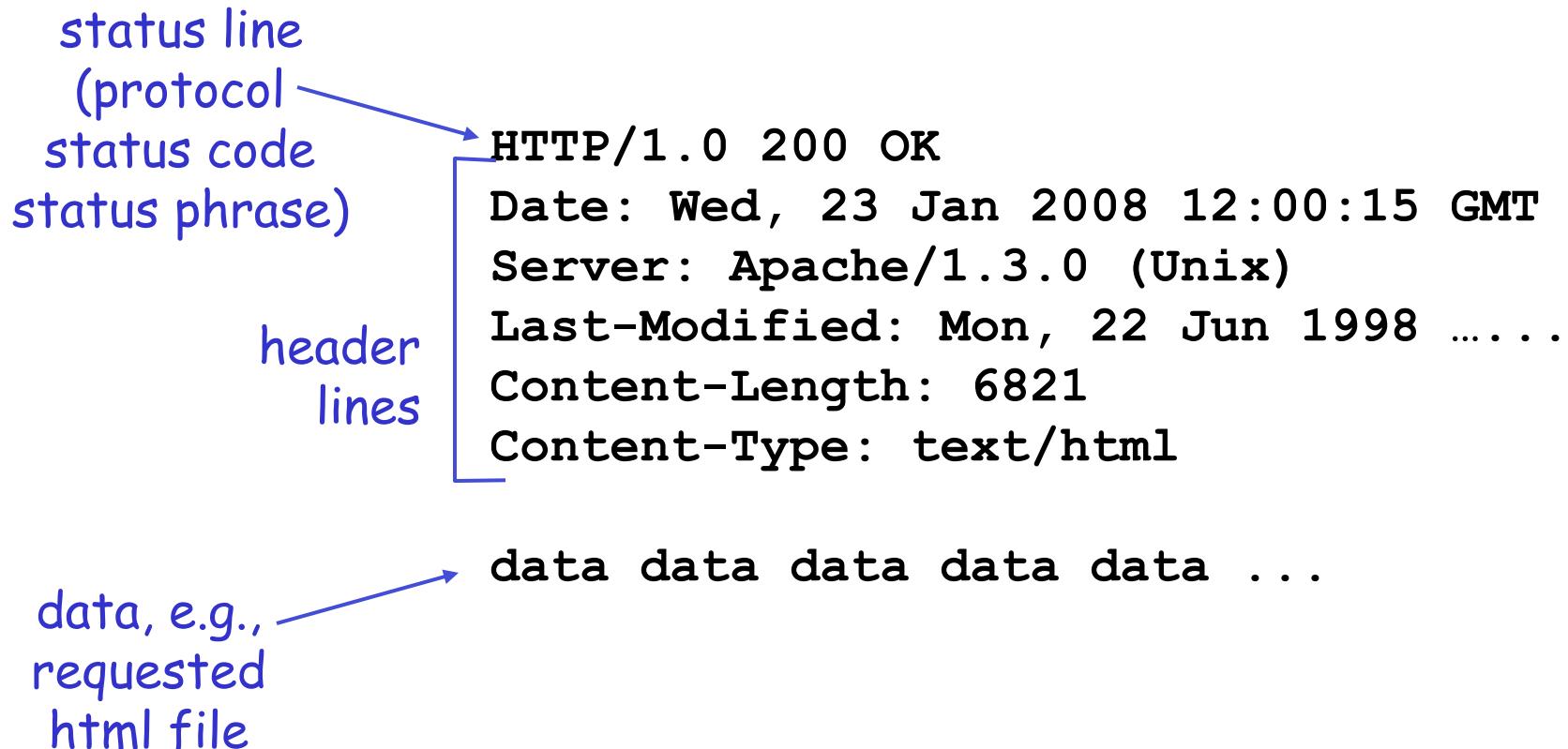
Connection management

Content negotiation

```
GET /somedir/page.html HTTP/1.0
Host: www.somechool.edu
Connection: close
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: en
```

(extra carriage return, line feed)

HTTP Response Message



HTTP Response Status Codes

In the first line of the server->client response message. A few sample codes:

200 OK

- request succeeded, requested object later in this message

301 Moved Permanently

- requested object moved, new location specified later in this message (Location:)

400 Bad Request

- request message not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

Trying Use Chrome to visit Course Page

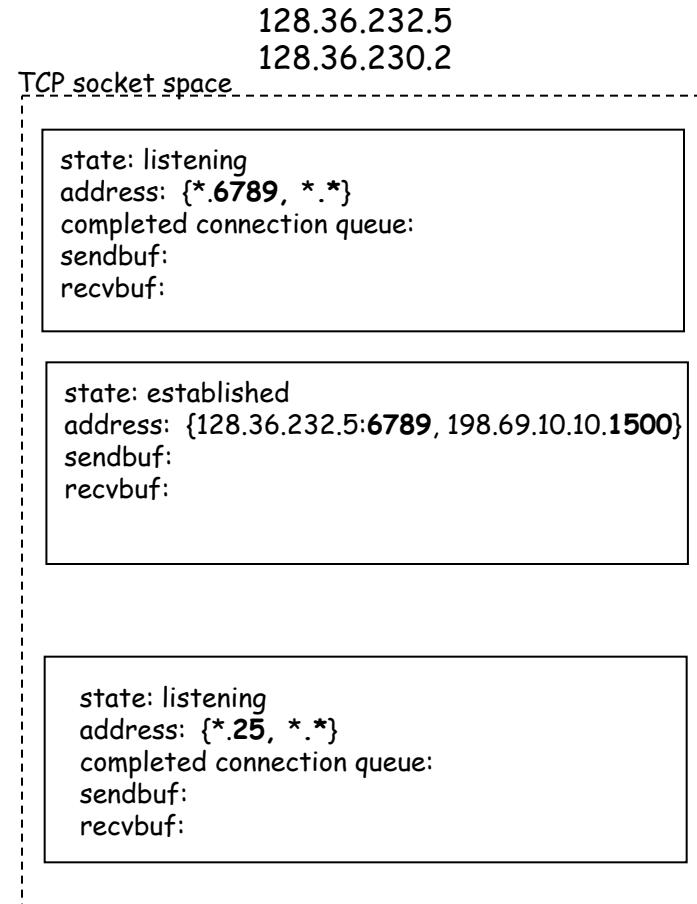
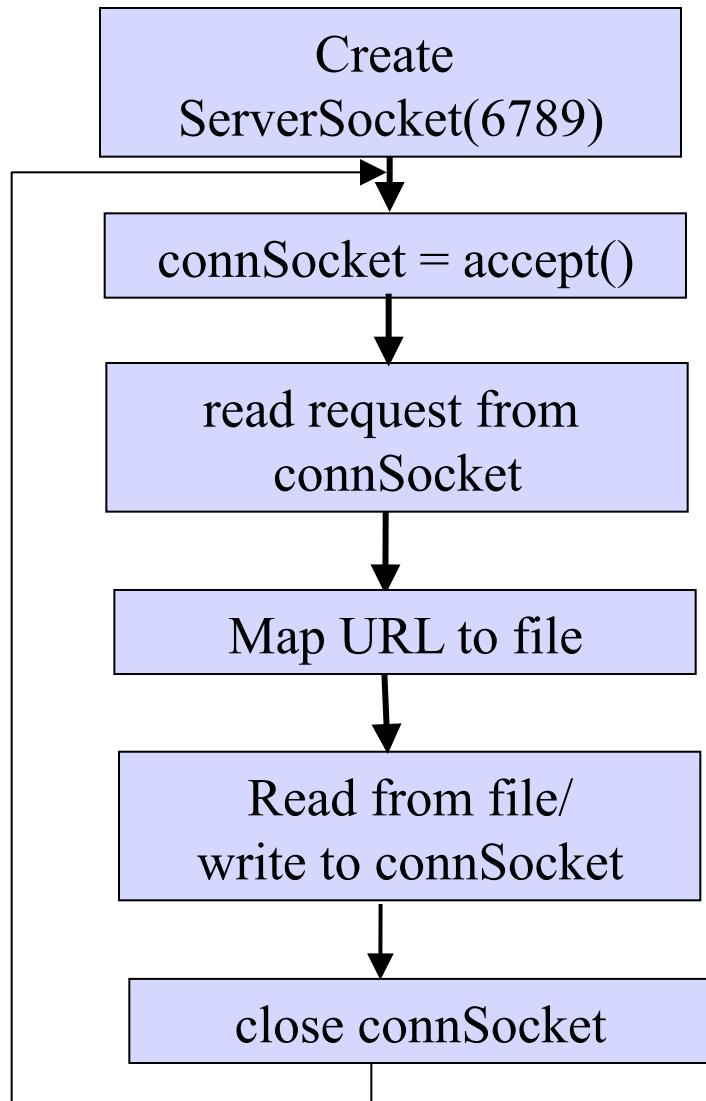
Design Exercise

- ❑ Workflow of an HTTP server processing a GET request that maps to a file:

GET /somedir/page.html HTTP/1.0

Host: www.somechool.edu

Basic HTTP Server Workflow

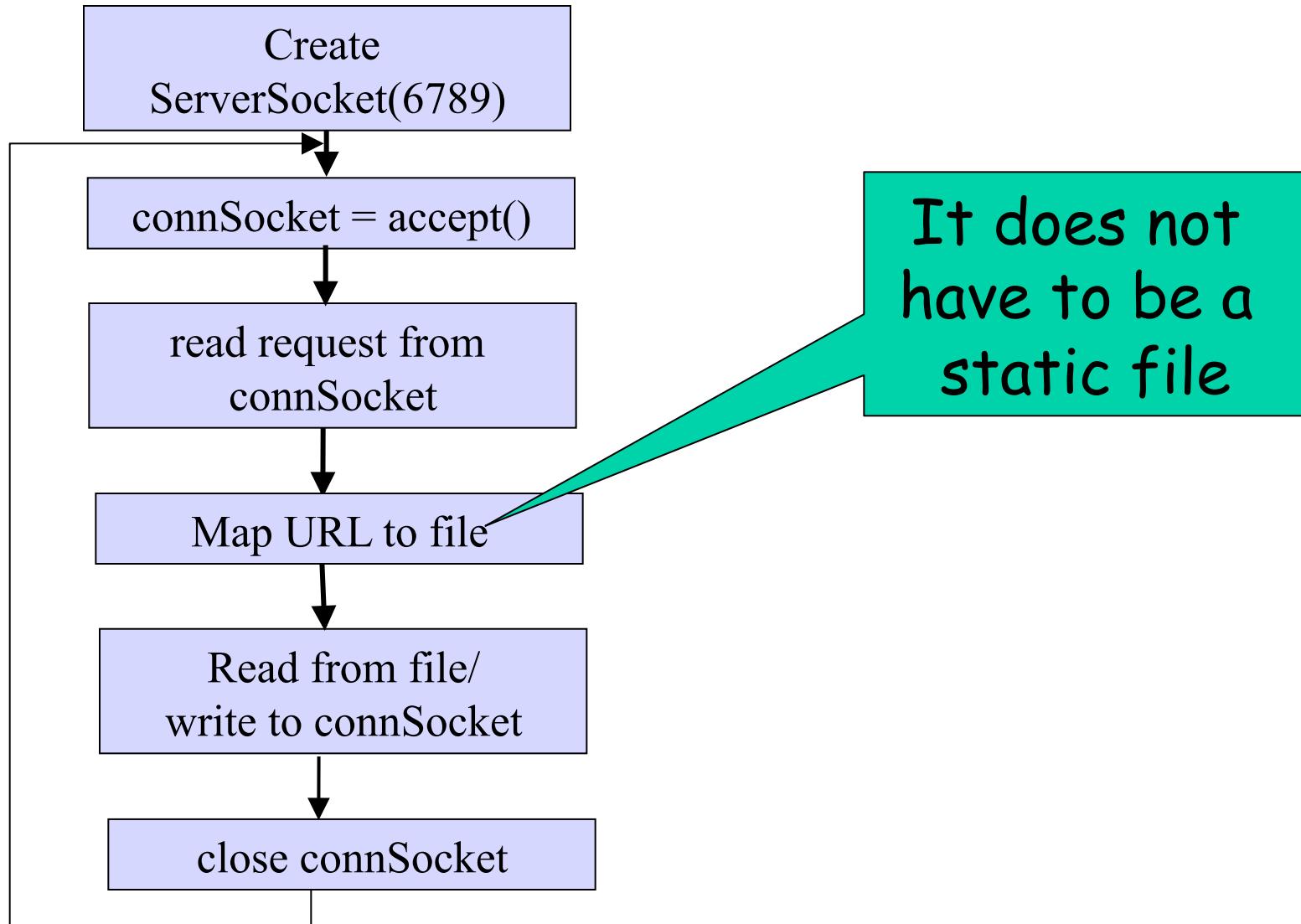


Example Code

- See BasicWebServer.java

- Try using telnet and real browser, and fetch
 - file1.html
 - index.htmlwhat difference in behavior?

Static -> Dynamic Content



Dynamic Content Pages

- There are multiple approaches to make dynamic web pages:
 - Embed code into pages (server side include)
 - http server includes an interpreter for the type of pages
 - Invoke external programs (http server is agnostic to the external program execution)

`http://www.cs.yale.edu/index.shtml`

`http://www.cs.yale.edu/cgi-bin/ureserve.pl`

`http://www.google.com/search?q=Yale&sourceid=chrome`

Example SSI

- See `programming/examples-java-socket/BasicWebServer/ssi/index.shtml`,
`header.shtml`, ...

Example SSI

- See programming/examples-java-socket/
BasicWebServer/ssi/index.shtml,
header.shtml, ...

- To enable ssi, need configuration to tell the
web server (see conf/apache-htaccess)
 - <https://httpd.apache.org/docs/2.2/howto/htaccess.html> (Server Side Includes example)

CGI: Invoking External Programs

□ Two issues

- Input: Pass HTTP request parameters to the external program
- Output: Redirect external program output to socket

Example: Typical CGI Implementation

- Starts the executable as a child process
 - Passes HTTP request as environment variables
 - <http://httpd.apache.org/docs/2.2/env.html>
 - CGI standard: <http://www.ietf.org/rfc/rfc3875>
 - Redirects input/output of the child process to the socket

Example: CGI

□ Example:

- GET /search?q=Yale&sourceid=chrome HTTP/1.0
- setup environment variables, in particular
\$QUERY_STRING=q=Yale&sourceid=chrome
- start search and redirect its input/output

<https://docs.oracle.com/javase/7/docs/api/java/lang/ProcessBuilder.html>

Example

- <http://zoo.cs.yale.edu/classes/cs433/cs433-2016-spring/programming/examples-java-socket/BasicWebServer/cgi/price.cgi?appl>

```
#!/usr/bin/perl -w

$company = $ENV{'QUERY_STRING'};
print "Content-Type: text/html\r\n";
print "\r\n";

print "<html>";
print "<h1>Hello! The price is ";

if ($company =~ /appl/) {
    my $var_rand = rand();
    print 450 + 10 * $var_rand;
} else {
    print "150";
}

print "</h1>";
print "</html>";
```

Client Using Dynamic Pages

- See ajax.html and wireshark for client code example

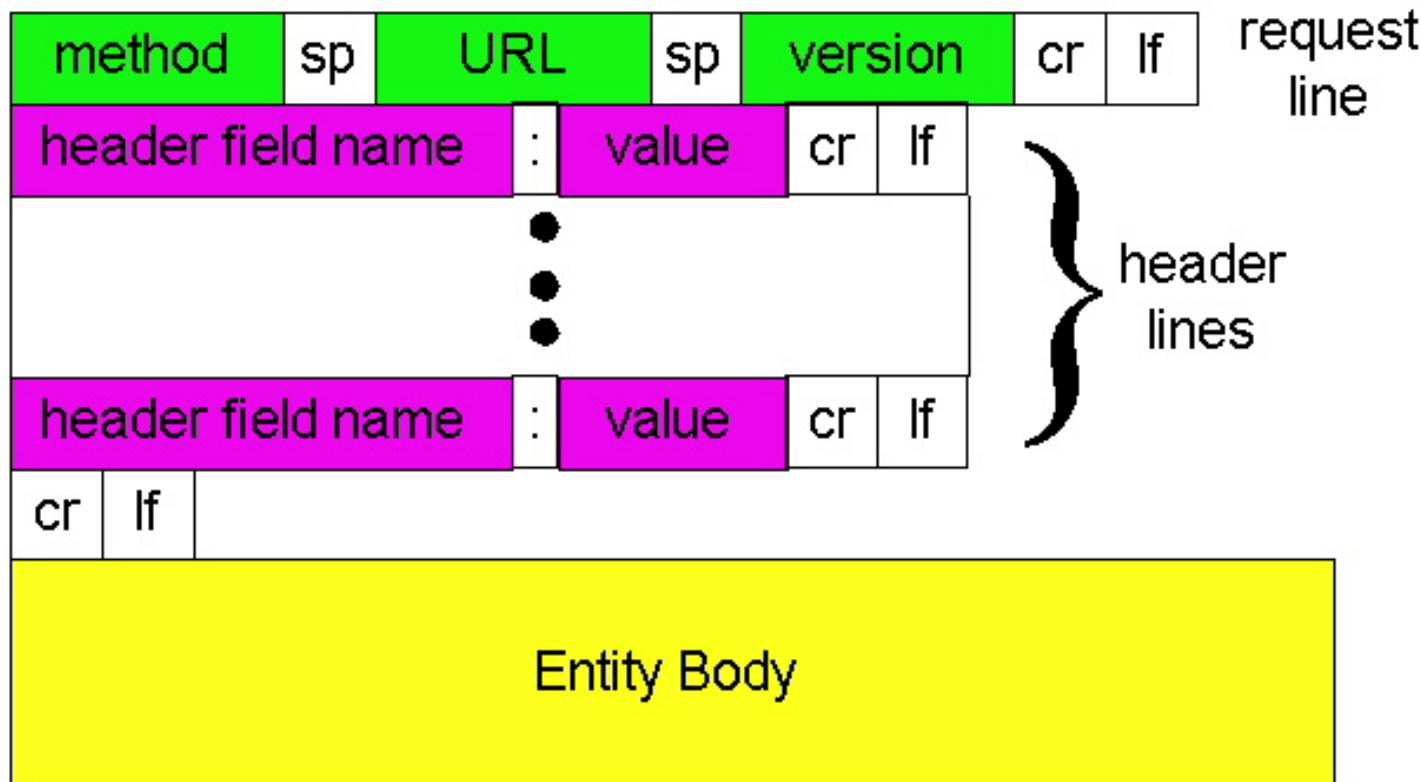
<http://zoo.cs.yale.edu/classes/cs433/cs433-2016-spring/programming/examples-java-socket/BasicWebServer/cgi/ajax.html>

Discussions

- ❑ What features are missing in HTTP that we have covered so far?

HTTP: POST

- If an HTML page contains forms or parameter too large, they are sent using POST and encoded in message body



HTTP: POST Example

POST /path/script.cgi HTTP/1.0

User-Agent: MyAgent

Content-Type: application/x-www-form-urlencoded

Content-Length: 15

item1=A&item2=B

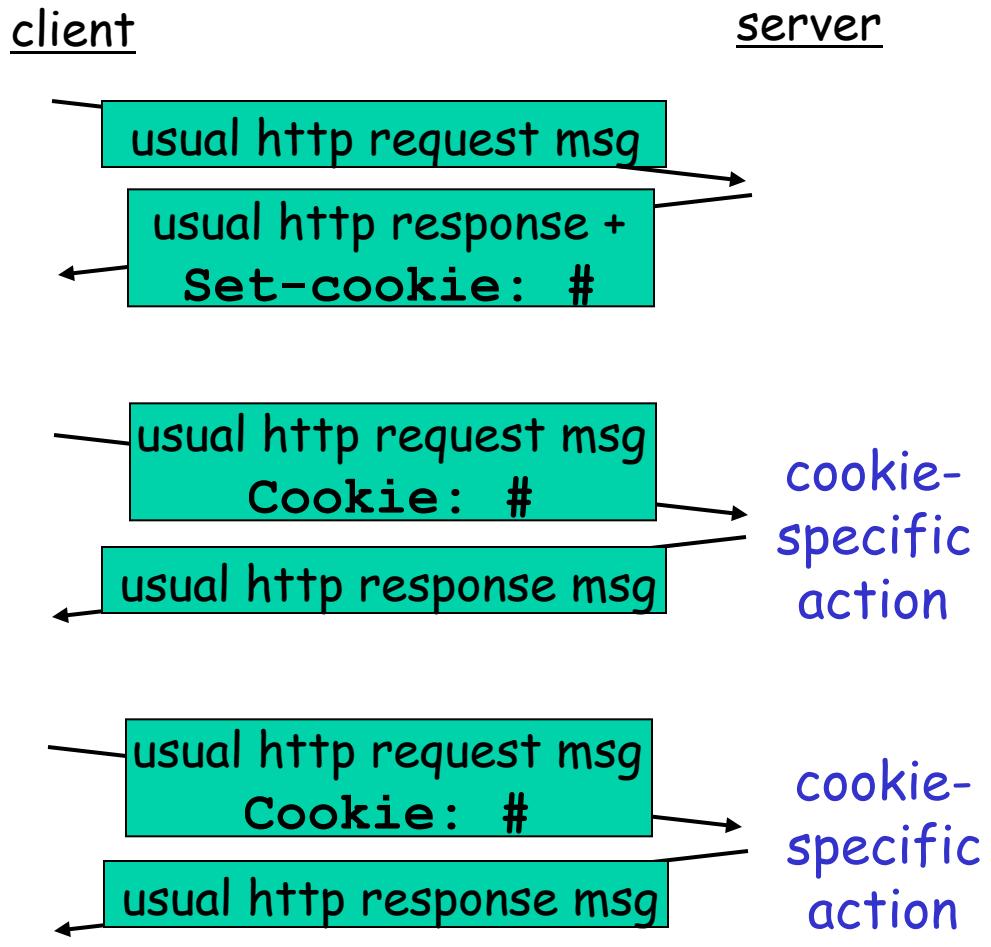
Example using nc:

programming/examples-java-socket/BasicWebServer/nc/

Stateful User-server Interaction: Cookies

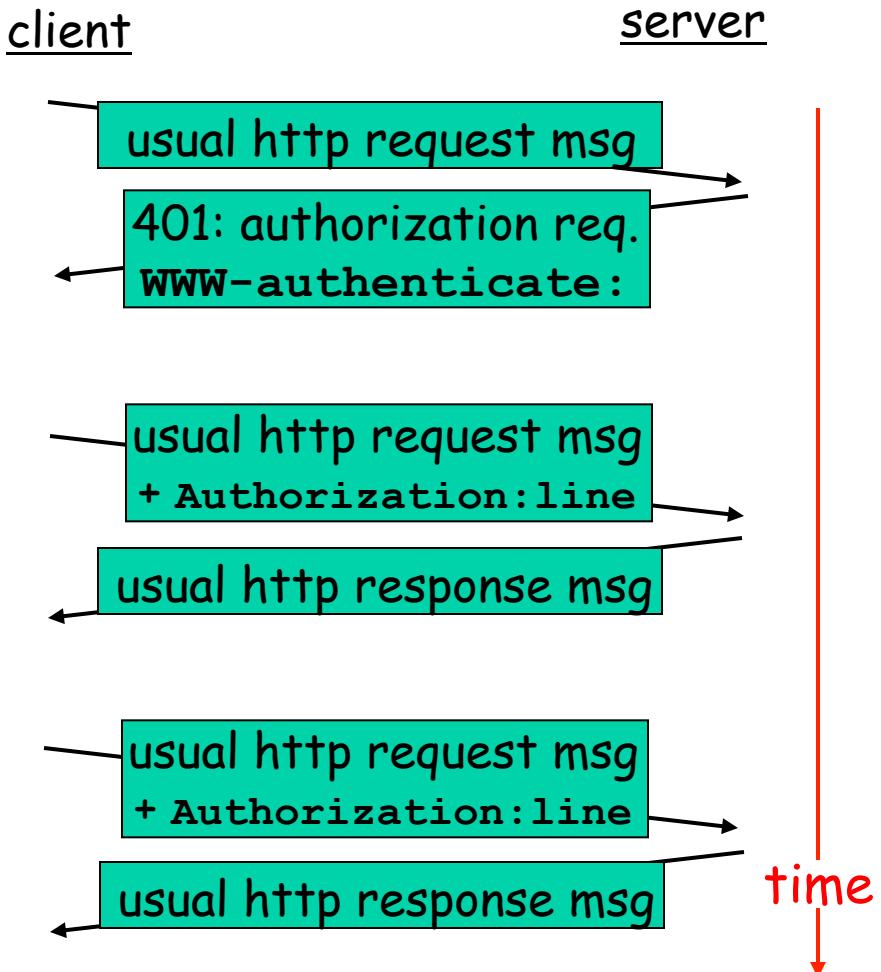
Goal: no explicit application level session

- Server sends “cookie” to client in response msg
`Set-cookie: 1678453`
- Client presents cookie in later requests
`Cookie: 1678453`
- Server matches presented-cookie with server-stored info
 - authentication
 - remembering user preferences, previous choices



Authentication of Client Request

- Authentication goal:** control access to server documents
- **stateless:** client must present authorization in each request
 - authorization: typically name, password
 - Authorization: header line in request
 - if no authorization presented, server refuses access, sends **WWW-authenticate:** header line in response



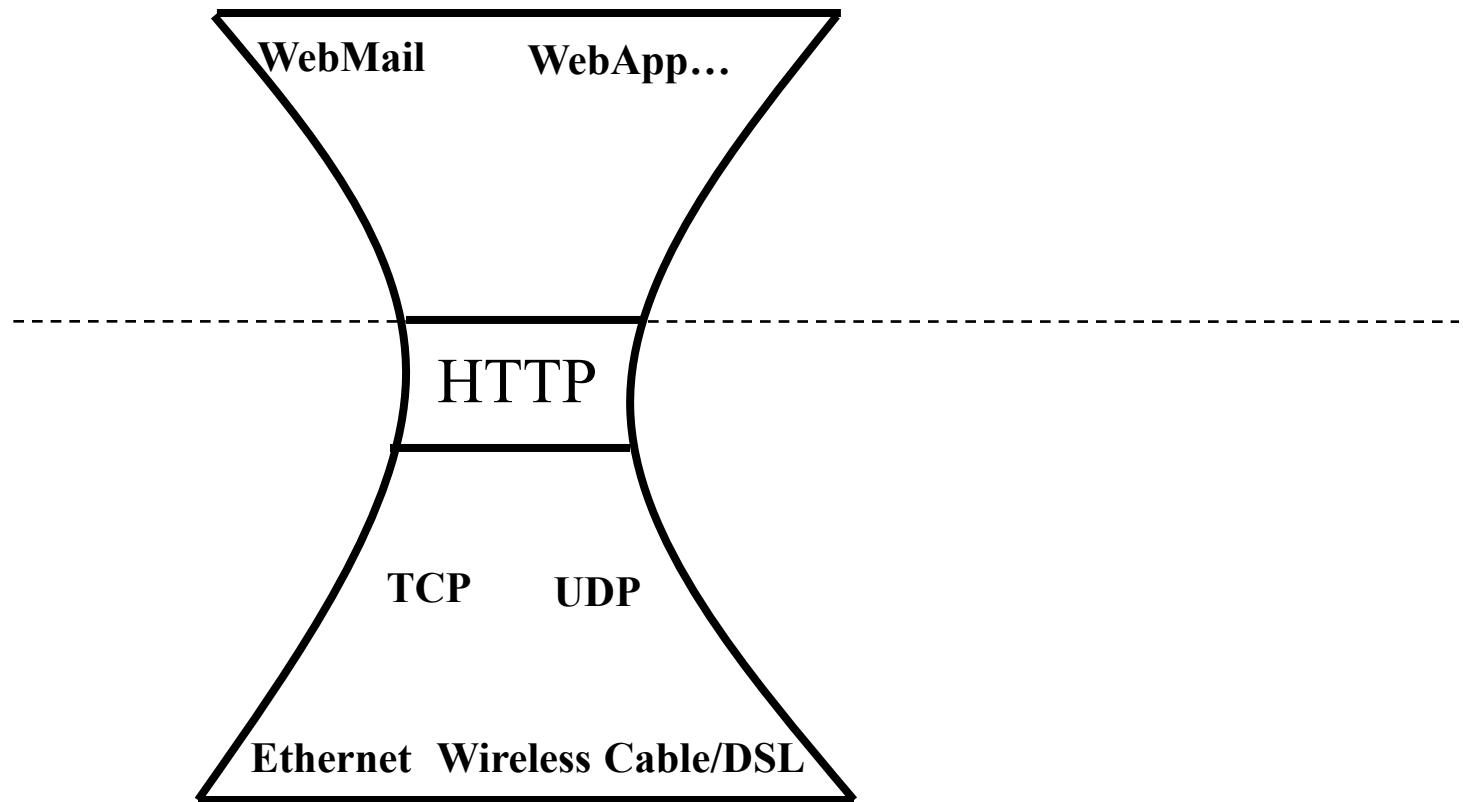
Browser caches name & password so that user does not have to repeatedly enter it.

Example: Amazon S3

- Amazon S3 API

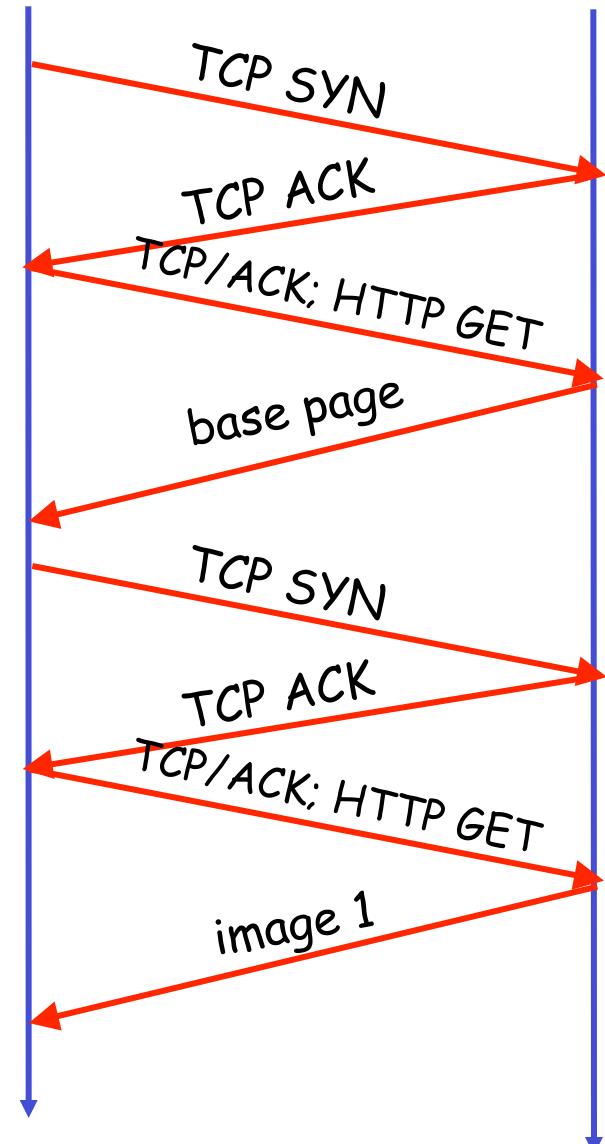
- <http://docs.aws.amazon.com/AmazonS3/latest/API/APIRest.html>

HTTP as the Thin Waist



Protocol Flow 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)



Discussion: How to Speedup HTTP/1.0

Network Applications:
Network App Programming: HTTP/1.1/2;
High-performance Server Design

Y. Richard Yang

<http://zoo.cs.yale.edu/classes/cs433/>

2/15/2016

Outline

- Admin and recap
- HTTP “acceleration”
- Network server design

Recap: HTTP

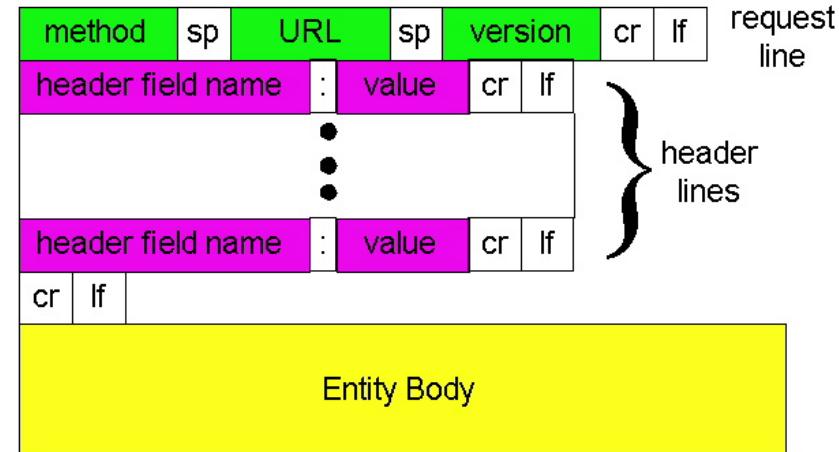
□ C-S app serving Web pages

○ message format

- request/response line, header lines, entity body
- simple methods, rich headers

○ message flow

- stateless server, thus states such as cookie and authentication are needed in each message

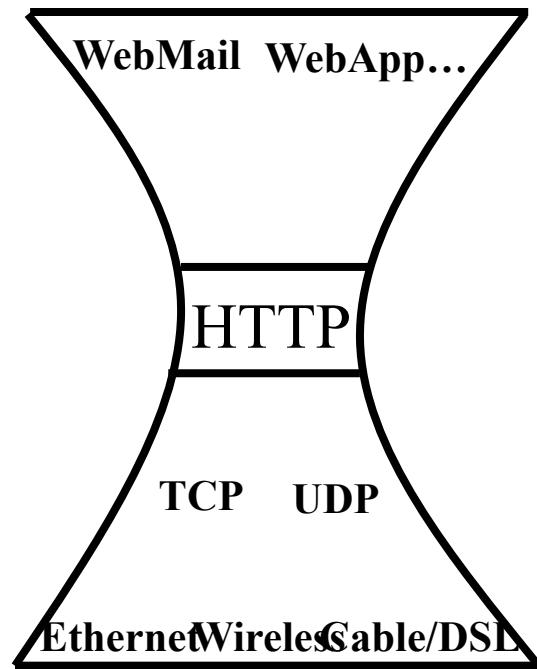
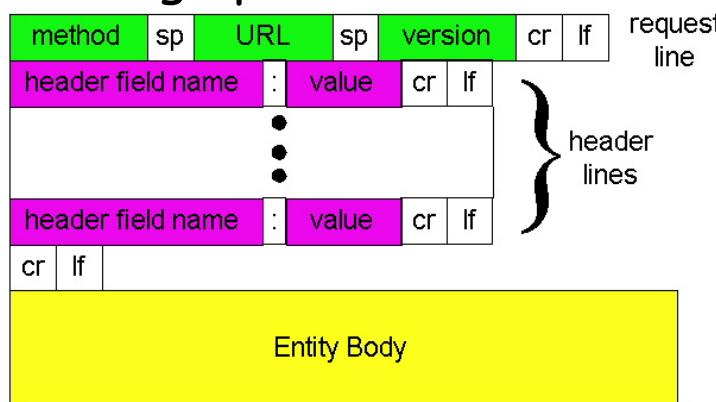


Recap: HTTP

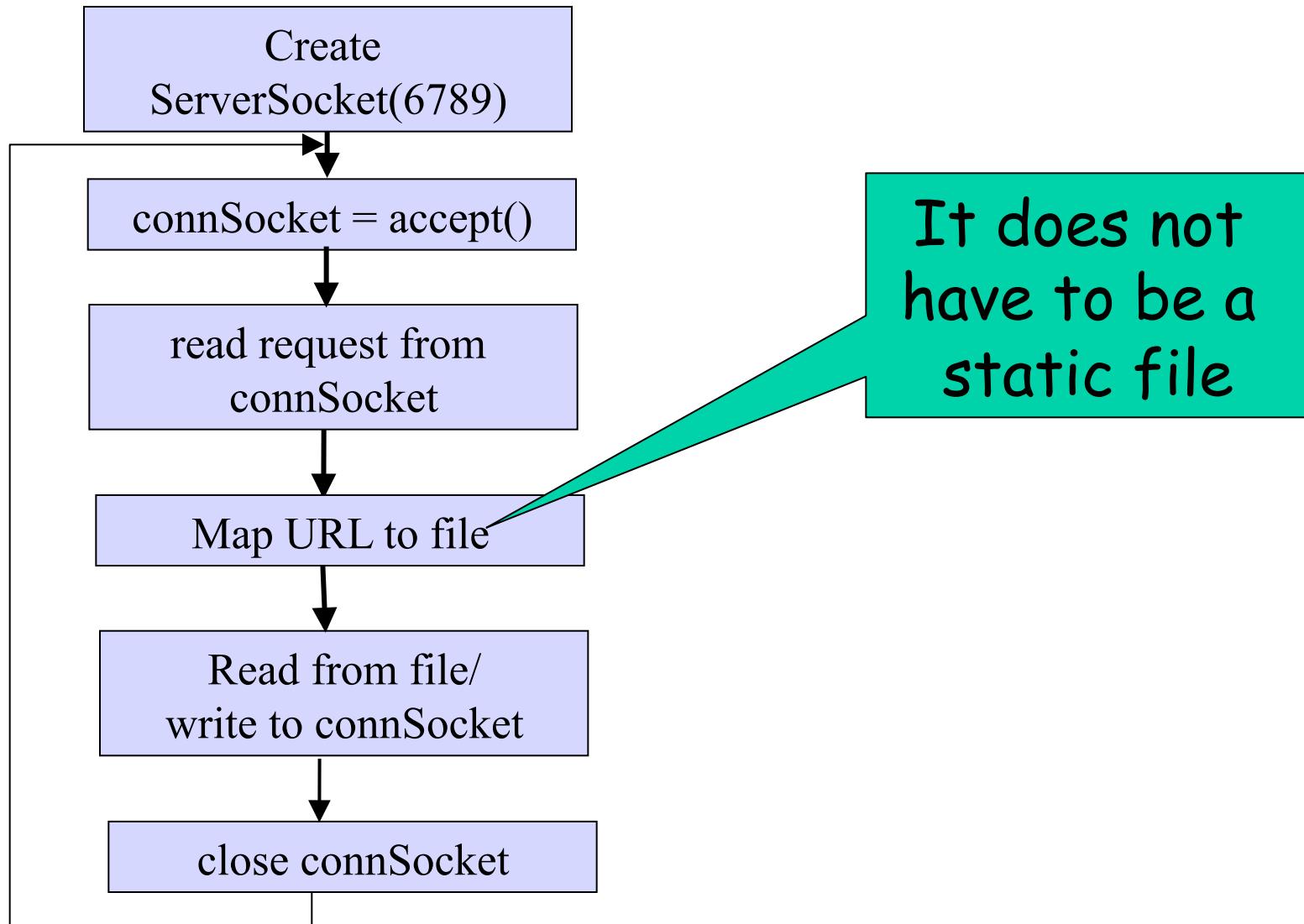
- ❑ Wide use of HTTP for Web applications

- ❑ Example: RESTful API
 - RESTful design

- http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
- <http://docs.oracle.com/javaee/6/tutorial/doc/giepu.html>

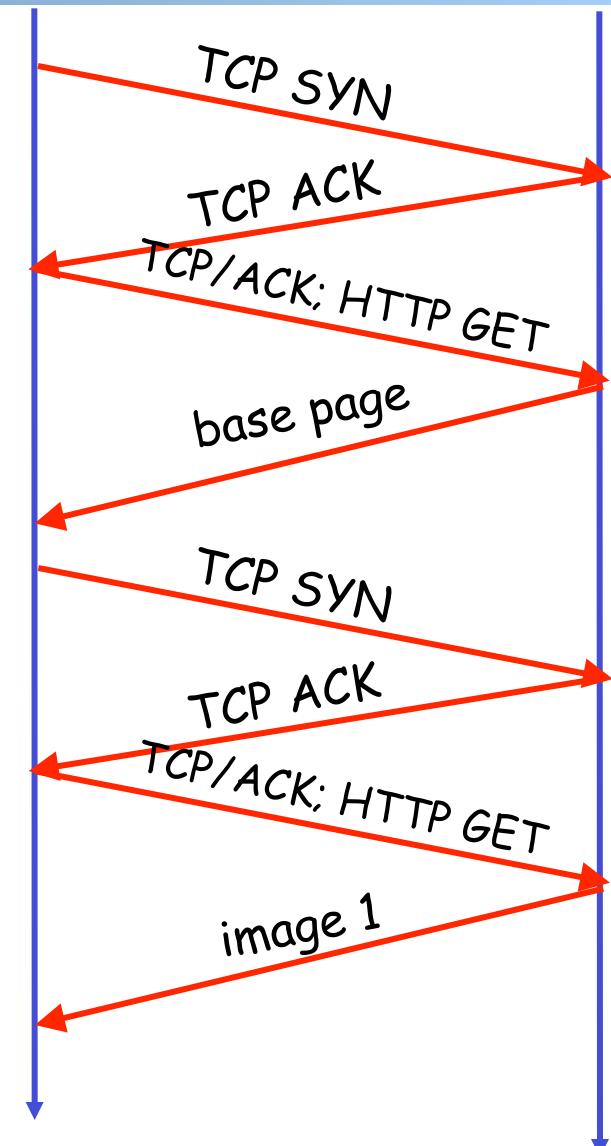


Recap: Basic HTTP/1.0 Server



Recap: Protocol Flow 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)



Outline

- Admin and recap
- HTTP "acceleration"

Substantial Efforts to Speedup Basic HTTP/1.0

- Reduce the number of objects fetched [Browser cache]
- Reduce data volume [Compression of data]
- Reduce the latency to the server to fetch the content [Proxy cache]
- Increase concurrency [Multiple TCP connections]
- Remove the extra RTTs to fetch an object [Persistent HTTP, aka HTTP/1.1]
- Asynchronous fetch (multiple streams) using a single TCP [HTTP/2]
- Server push [HTTP/2]
- Header compression [HTTP/2]



Browser Cache and Conditional GET

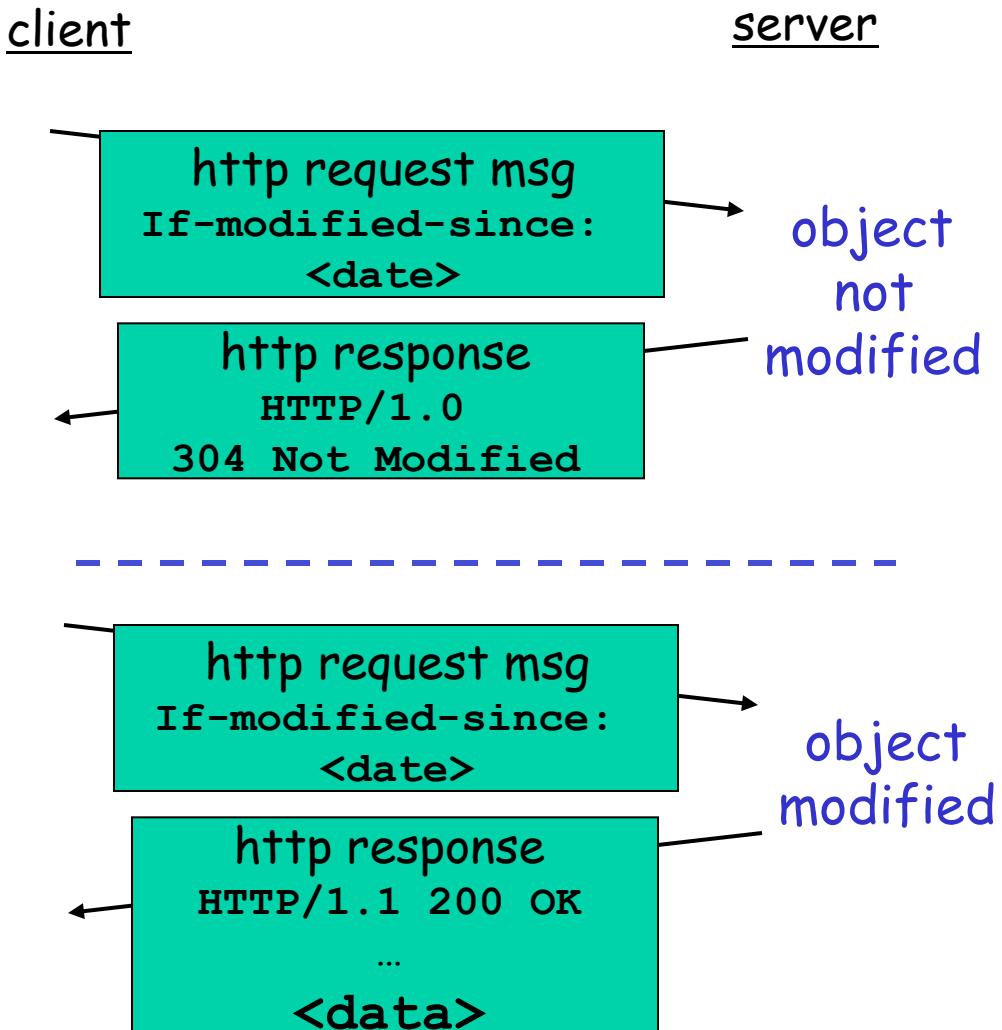
- **Goal:** don't send object if client has up-to-date stored (cached) version

- client: specify date of cached copy in http request

If-modified-since:
 <date>

- server: response contains no object if cached copy up-to-date:

HTTP/1.0 304 Not Modified

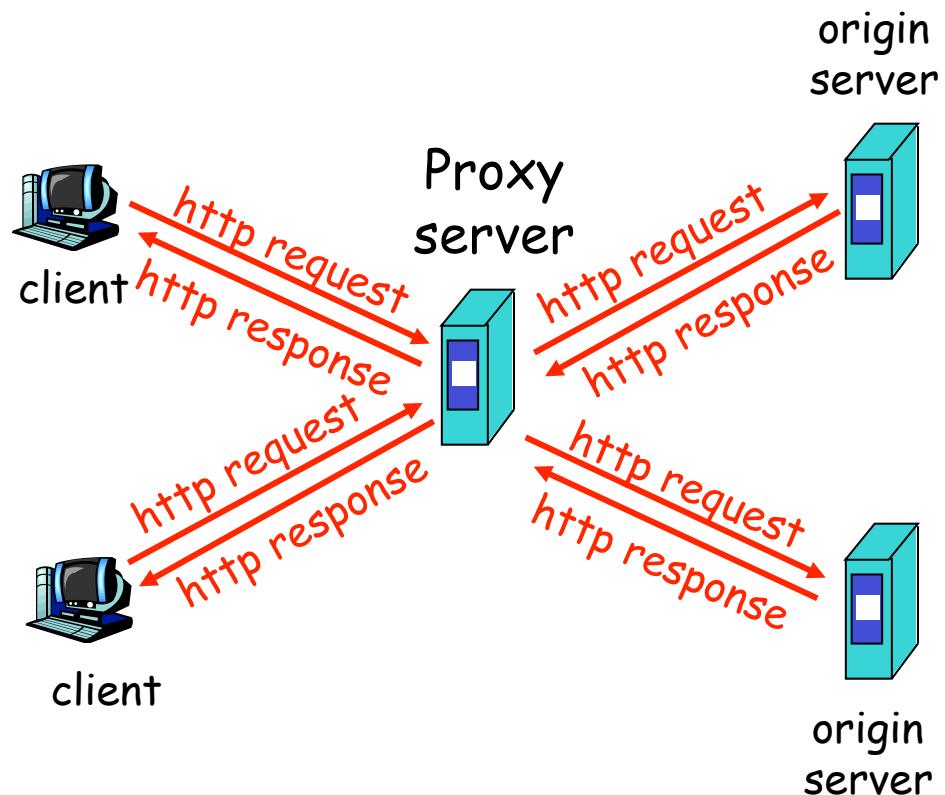


Web Caches (Proxy)

Goal: satisfy client request without involving origin server

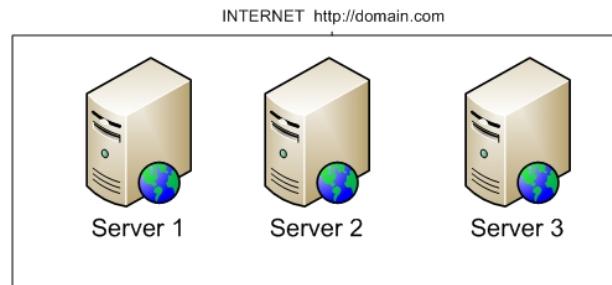
- Two types of proxies

- Forward proxy
 - Typically in the same network as the client
 - Reverse proxy
 - Typically in the same network as the server

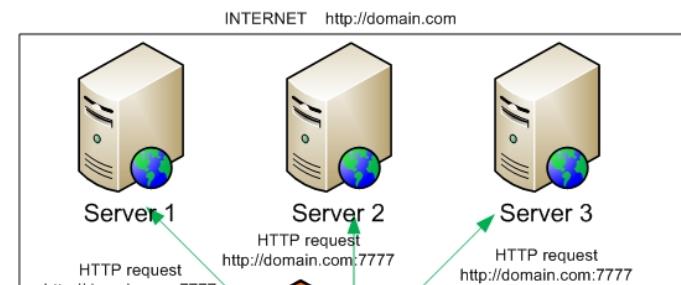


Two Types of Proxies

(FORWARD) PROXY server

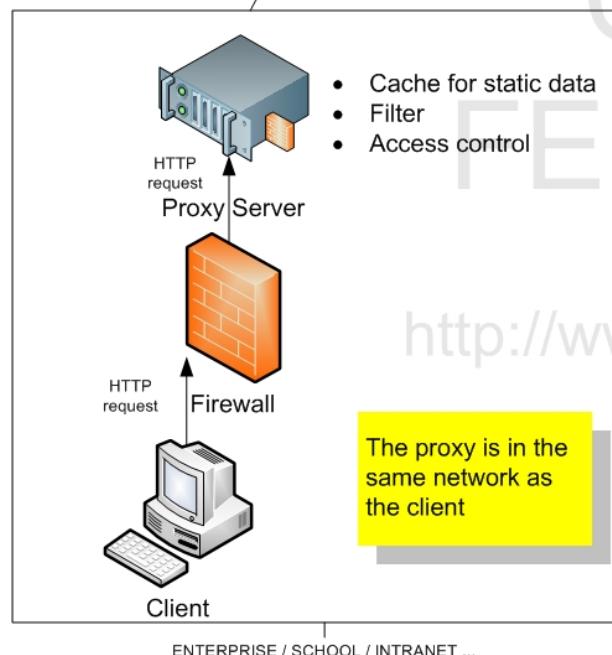


REVERSE-PROXY server



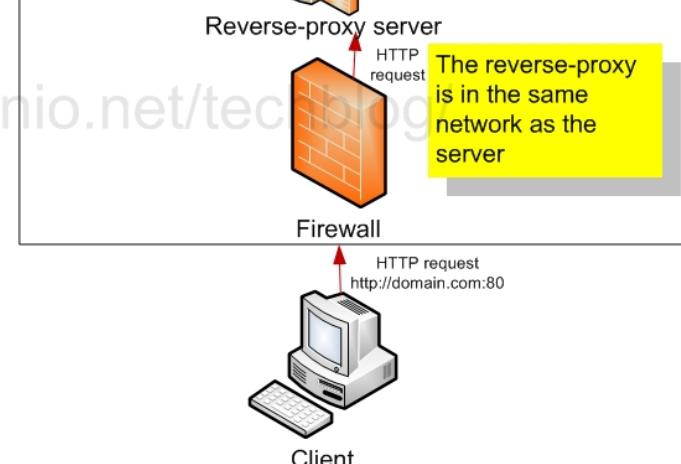
- Cache for static data
- Filter
- Access control

The proxy is in the same network as the client



- Load-balancing
- Filter
- Cache for static data

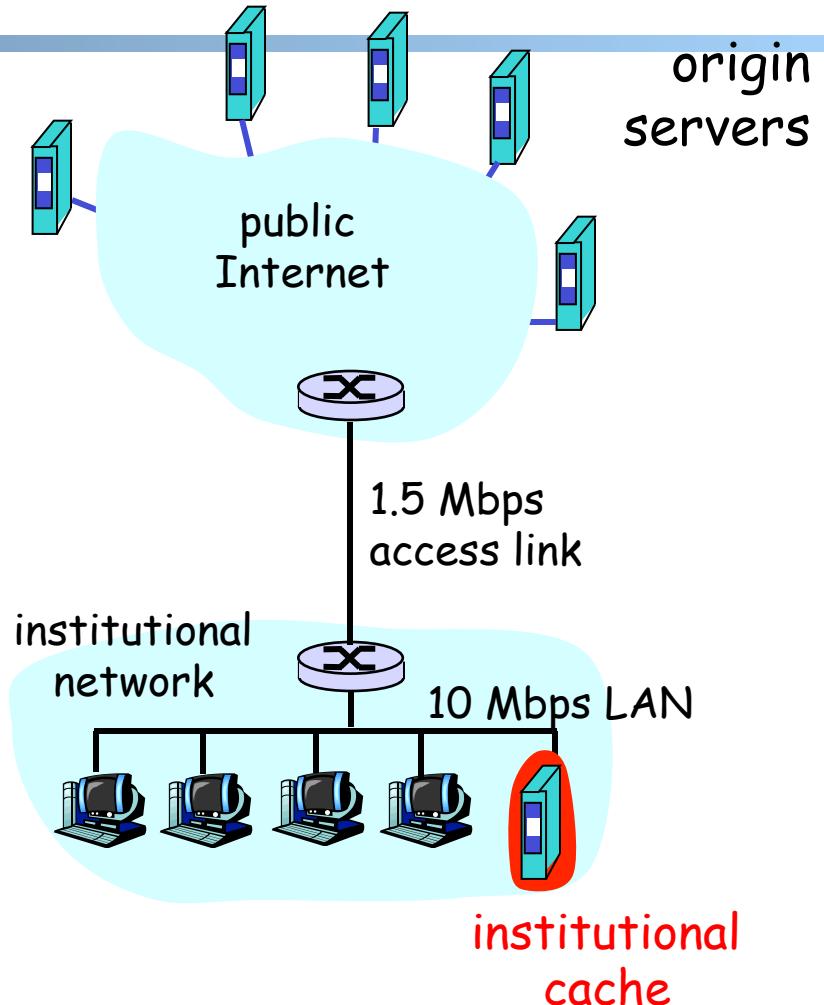
The reverse-proxy is in the same network as the server



Benefits of Forward Proxy

Assume: cache is “close” to client (e.g., in same network)

- smaller response time: cache “closer” to client
- decrease traffic to distant servers
 - link out of institutional/local ISP network often bottleneck



No Free Lunch: Problems of Web Caching

- The major issue of web caching is how to maintain consistency
- Two ways
 - pull
 - Web caches periodically pull the web server to see if a document is modified
 - push
 - whenever a server gives a copy of a web page to a web cache, they sign a lease with an expiration time; if the web page is modified before the lease, the server notifies the cache

HTTP/1.1: Persistent (keepalive/pipelining)

HTTP

- On same TCP connection: server parses request, responds, parses new request, ...
- Client sends requests for all referenced objects as soon as it receives base HTML
- Fewer RTTs

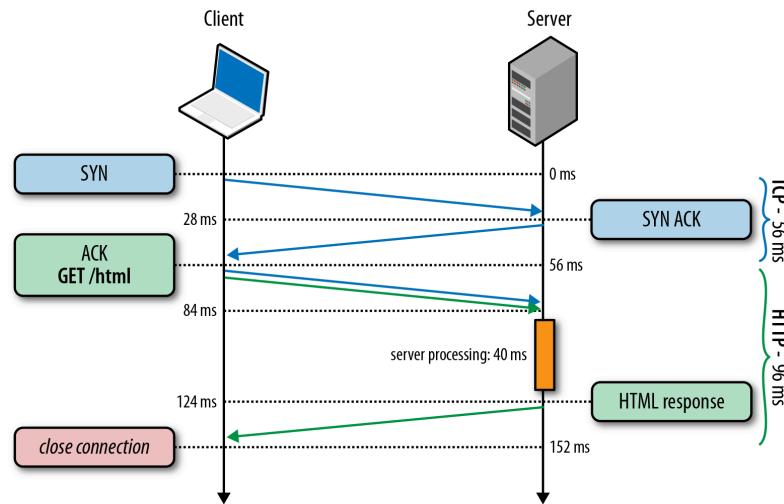
See Joshua Graessley WWDC 2012 talk: 3x within iTunes

Example

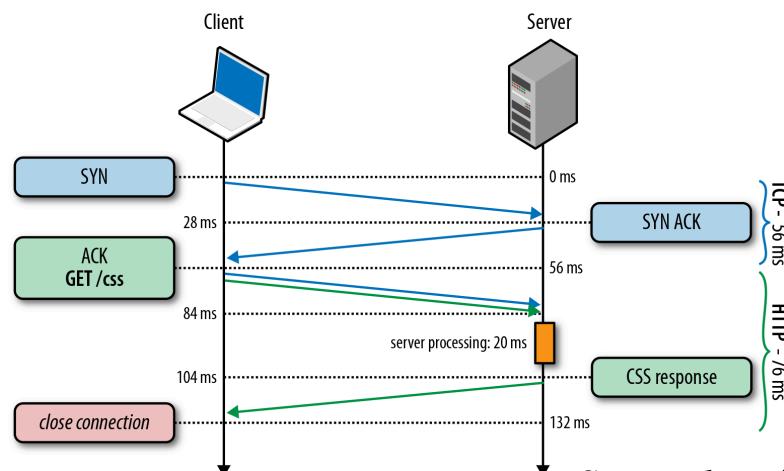
- Visit cs home page using Chrome

HTTP/1.0, Keep-Alive, Pipelining

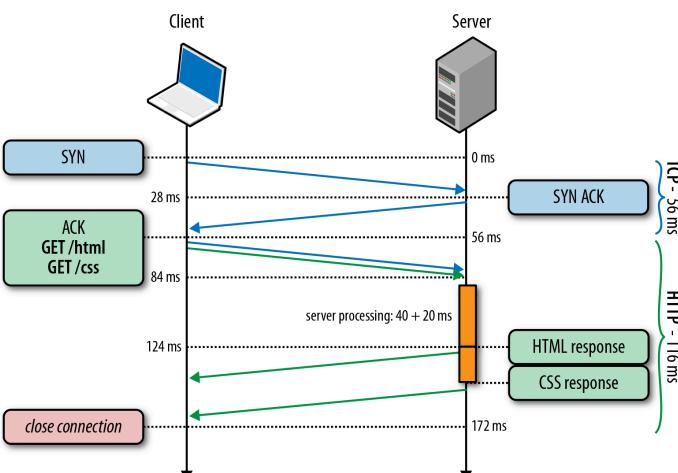
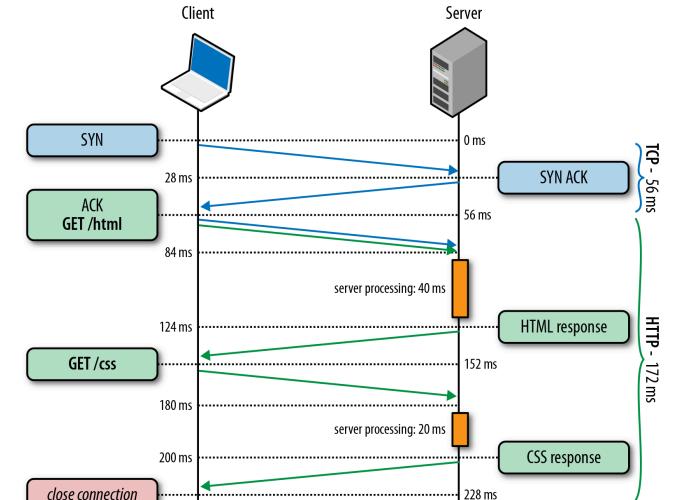
TCP connection #1, Request #1: HTML request



TCP connection #2, Request #2: CSS request

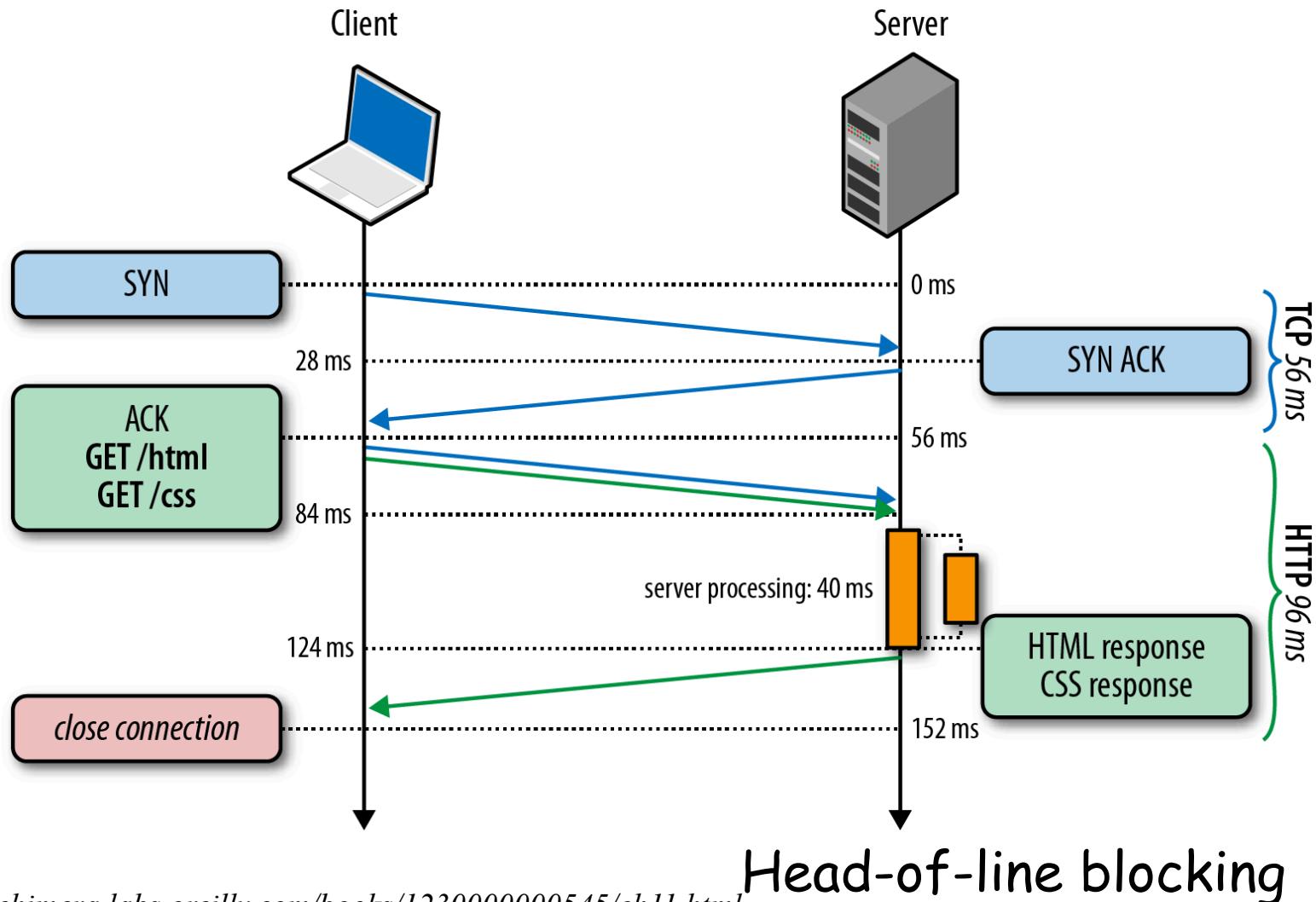


TCP connection #1, Request #1-2: HTML + CSS

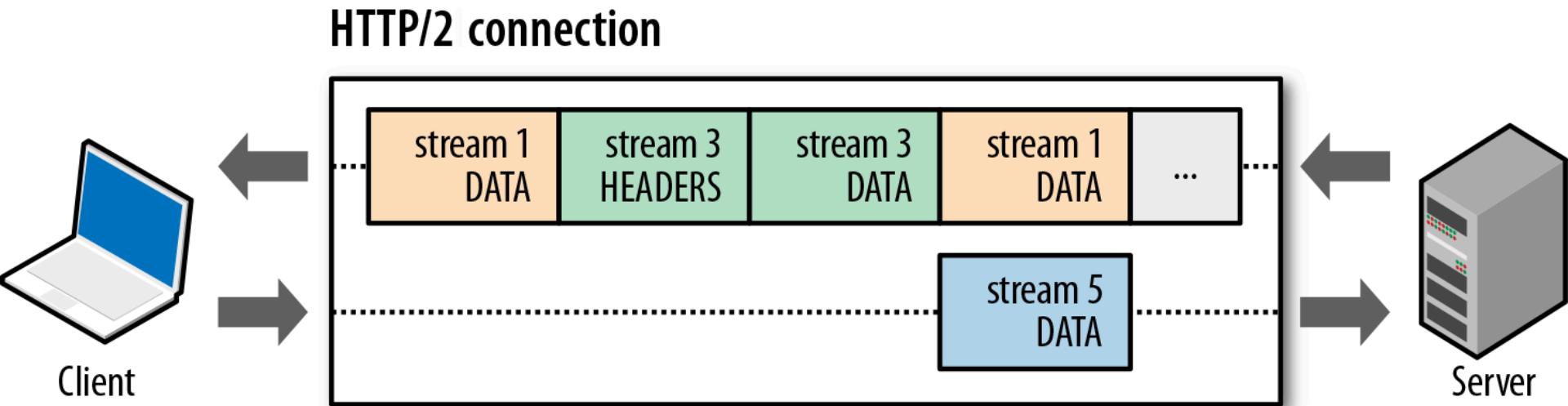


Source: <http://chimera.labs.oreilly.com/books/1230000000545/ch11.html>

Remaining Problem of Pipelining



HTTP/2 Multi-Streams Multiplexing

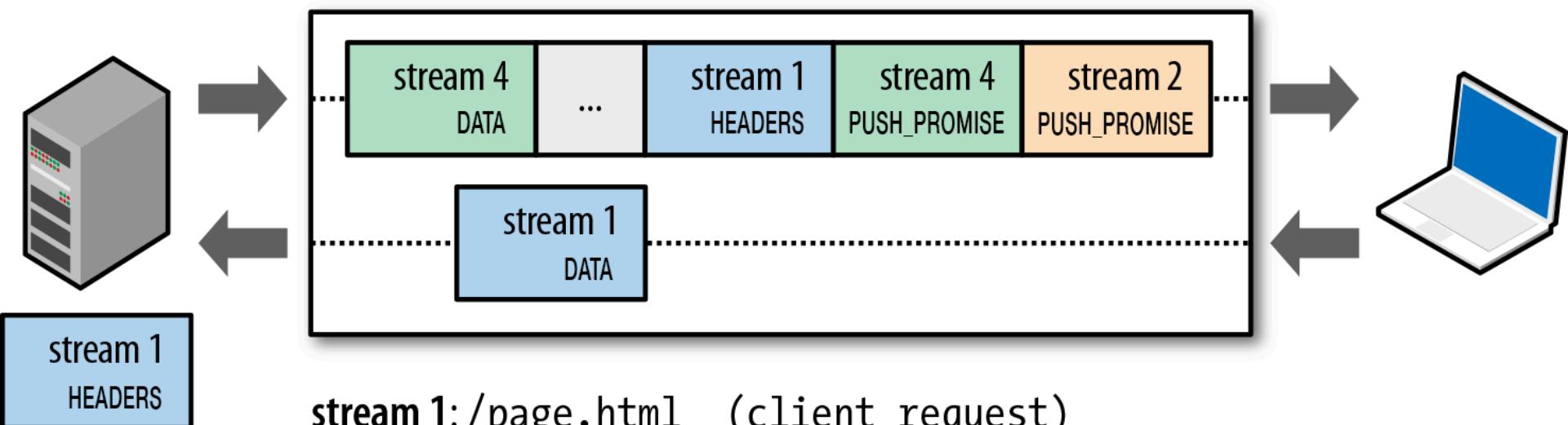


Bit	+0..7	+8..15	+16..23	+24..31
0		Length		Type
32	Flags			
40	R		Stream Identifier	
...			<i>Frame Payload</i>	

HTTP/2 Binary Framing

HTTP/2 Server Push

HTTP/2 connection

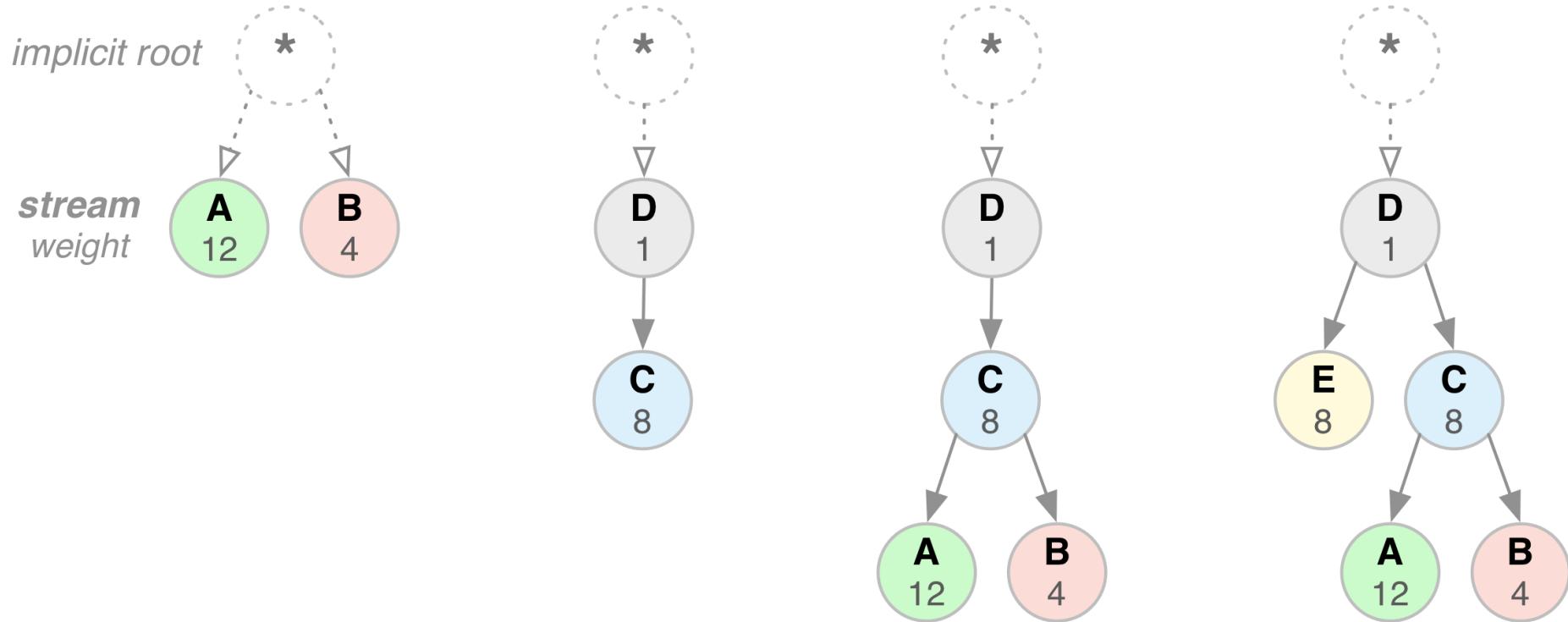


stream 1: /page.html (client request)

stream 2: /script.js (push promise)

stream 4: /style.css (push promise)

HTTP/2 Stream Dependency and Weights



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
...



Encoded headers

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

Dynamic table



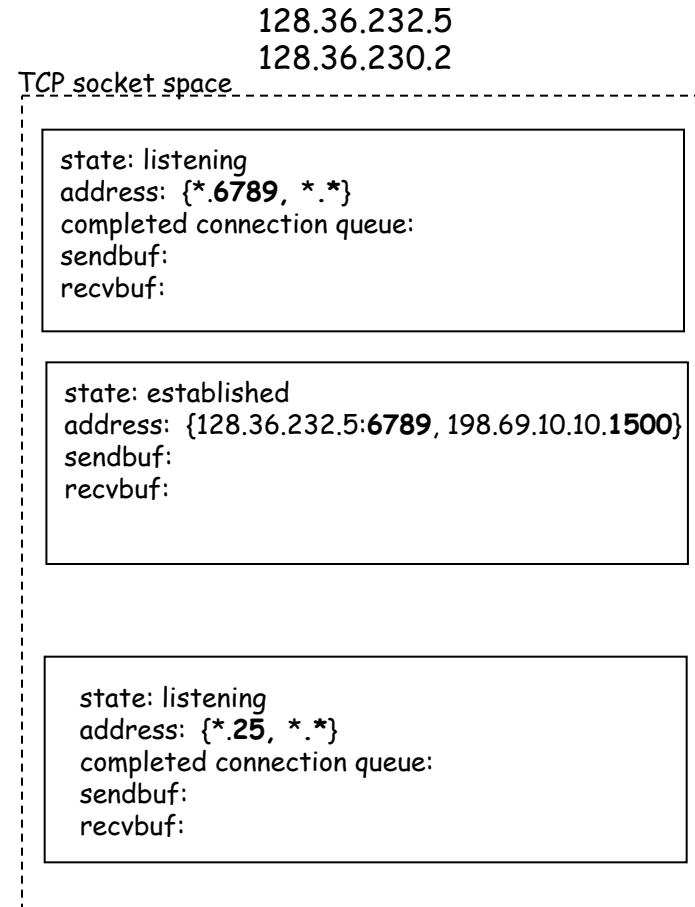
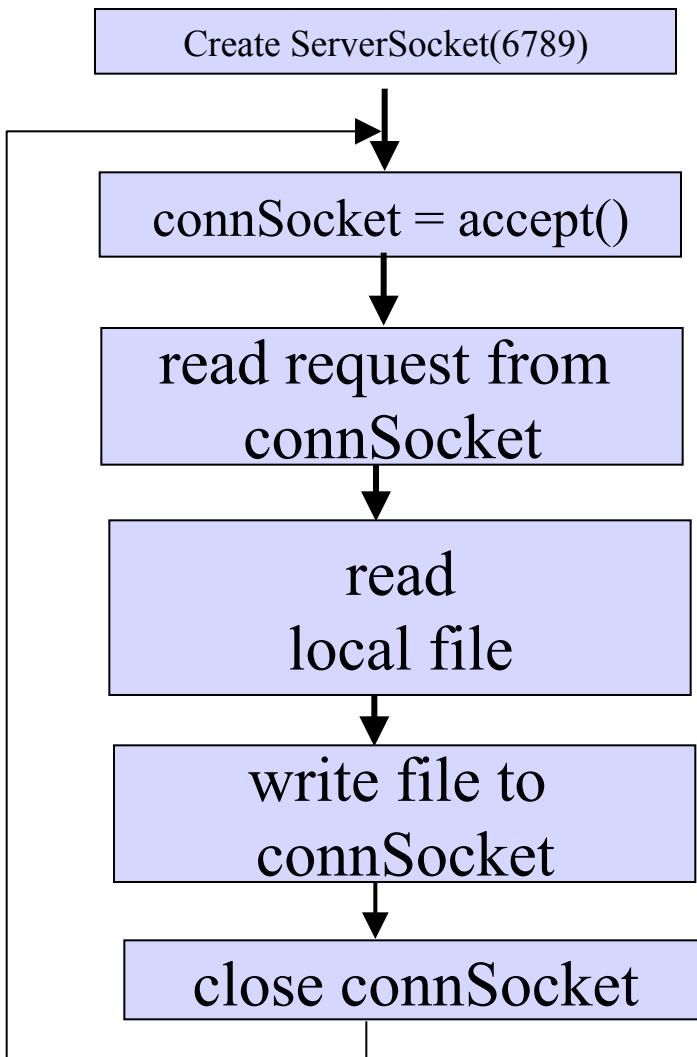
Example

- Visit HTTP/2 pages, such as
<https://http2.akamai.com>
- See chrome://net-internals/#http2

Outline

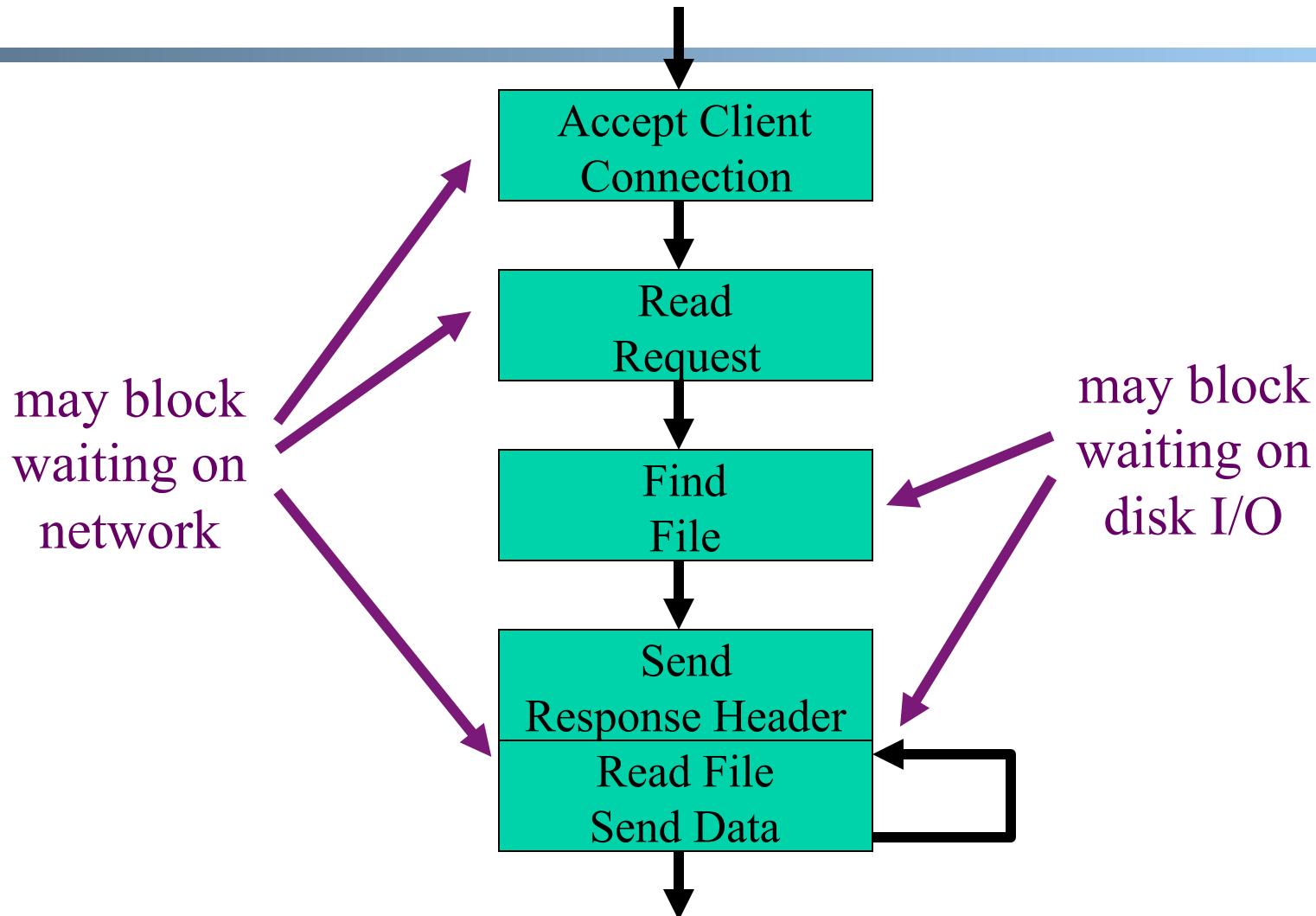
- Admin and recap
- HTTP “acceleration”
- Network server design

WebServer Implementation



Discussion: what does each step do and how long does it take?

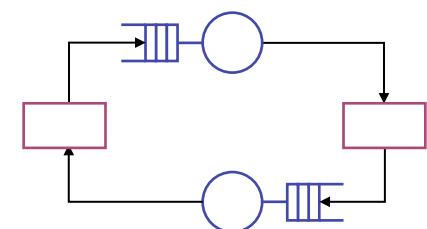
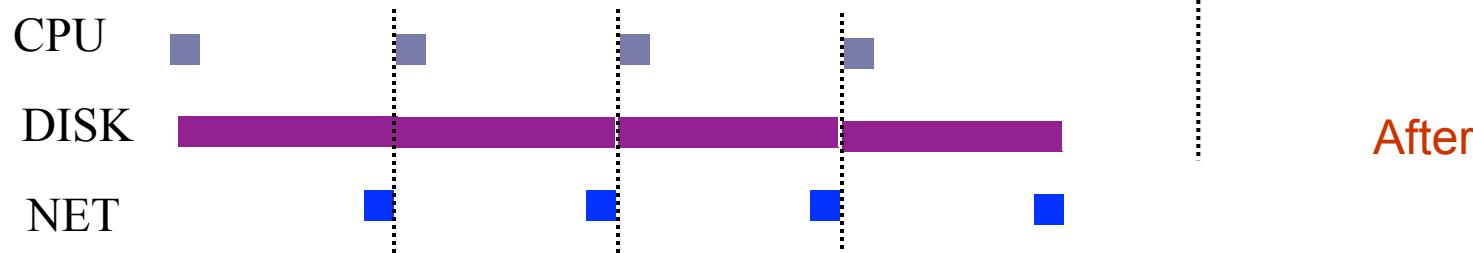
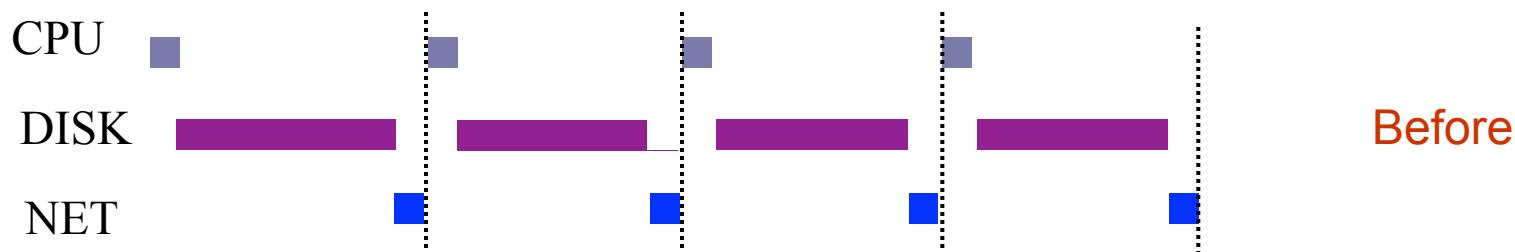
Server Processing Steps



Writing High Performance Servers: Major Issues

- Many socket and IO operations can cause a process to block, e.g.,
 - accept: waiting for new connection;
 - read a socket waiting for data or close;
 - write a socket waiting for buffer space;
 - I/O read/write for disk to finish

Goal: Limited Only by the Bottleneck



Outline

- Admin and recap
- HTTP "acceleration"
- Network server design
 - Overview
 - Multi-thread network servers

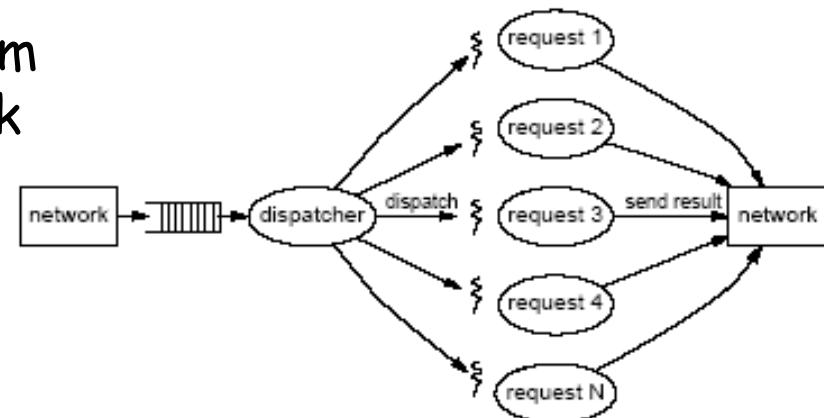
Multi-Threaded Servers

□ Motivation:

- Avoid blocking the whole program
(so that we can reach bottleneck
throughput)

□ Idea: introduce threads

- A thread is a sequence of
instructions which may execute
in parallel with other threads
- When a blocking operation
happens, only the flow (thread)
performing the operation is
blocked

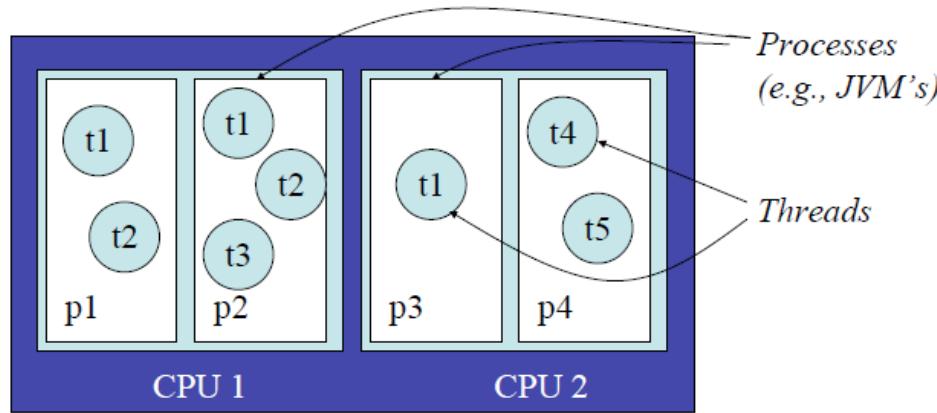


Background: Java Thread Model

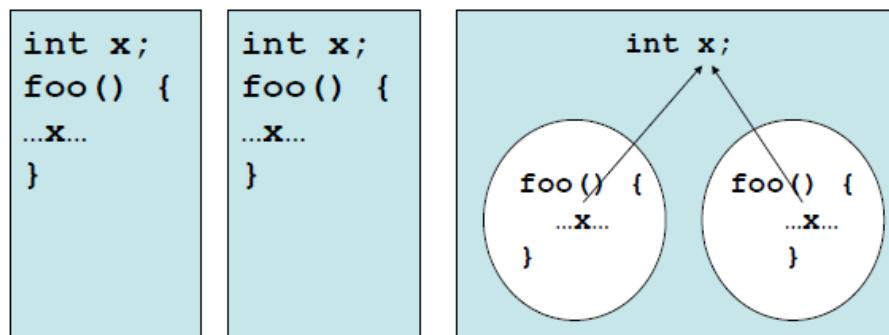
- Every Java application has at least one thread
 - The “main” thread, started by the JVM to run the application’s main() method
 - Most JVM’s use POSIX threads to implement Java threads

- main() can create other threads
 - Explicitly, using the Thread class
 - Implicitly, by calling libraries that create threads as a consequence (RMI, AWT/Swing, Applets, etc.)

Thread vs Process



A computer



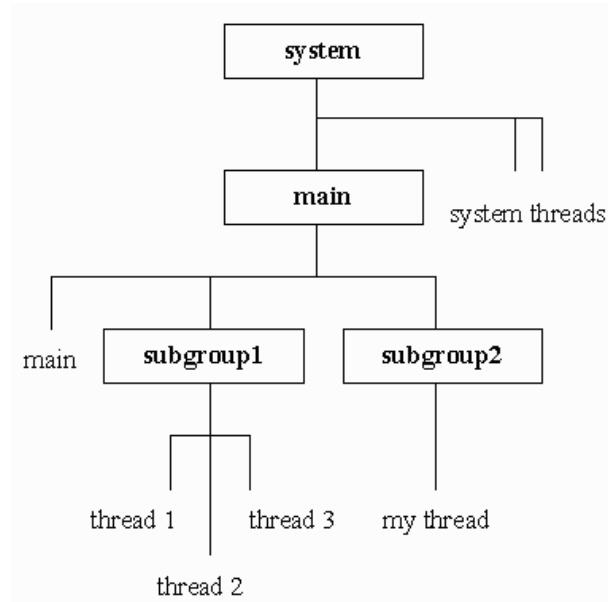
*Processes do not
share data*

*Threads share data
within a process*

Background: Java Thread Class

- Threads are organized into thread groups
 - A thread group represents a set of threads

```
activeGroupCount();
```
 - A thread group can also include other thread groups to form a tree
 - Why thread group?



Creating Java Thread

- Two ways to implement Java thread
 1. Extend the Thread class
 - Overwrite the `run()` method of the Thread class
 2. Create a class C implementing the Runnable interface, and create an object of type C, then use a Thread object to wrap up C
- A thread starts execution after its `start()` method is called, which will start executing the thread's (or the Runnable object's) `run()` method
- A thread terminates when the `run()` method returns

Option 1: Extending Java Thread

```
class PrimeThread extends Thread {  
    long minPrime;  
  
    PrimeThread(long minPrime) {  
        this.minPrime = minPrime;  
    }  
  
    public void run() {  
        // compute primes larger than minPrime . . .  
    }  
}  
  
PrimeThread p = new PrimeThread(143);  
p.start();
```

Option 1: Extending Java Thread

```
class RequestHandler extends Thread {  
    RequestHandler(Socket connSocket) {  
        // ...  
    }  
    public void run() {  
        // process request  
    }  
    ...  
}
```

```
Thread t = new RequestHandler(connSocket);  
t.start();
```

Option 2: Implement the Runnable Interface

```
class PrimeRun implements Runnable {  
    long minPrime;  
    PrimeRun(long minPrime) {  
        this.minPrime = minPrime;  
    }  
  
    public void run() {  
        // compute primes larger than minPrime . . .  
    }  
}  
  
PrimeRun p = new PrimeRun(143);  
  
new Thread(p).start();
```

Option 2: Implement the Runnable Interface

```
class RequestHandler implements Runnable {  
    RequestHandler(Socket connSocket) { ... }  
    public void run() {  
        //  
    }  
    ...  
}  
RequestHandler rh = new RequestHandler(connSocket);  
Thread t = new Thread(rh);  
t.start();
```

Summary: Implementing Threads

```
class RequestHandler  
    extends Thread {  
RequestHandler(Socket connSocket)  
{  
    ...  
}  
public void run() {  
    // process request  
}  
...  
}
```

```
Thread t = new RequestHandler(connSocket);  
t.start();
```

```
class RequestHandler  
    implements Runnable {  
RequestHandler(Socket connSocket)  
{  
    ...  
}  
public void run() {  
    // process request  
}  
...  
}
```

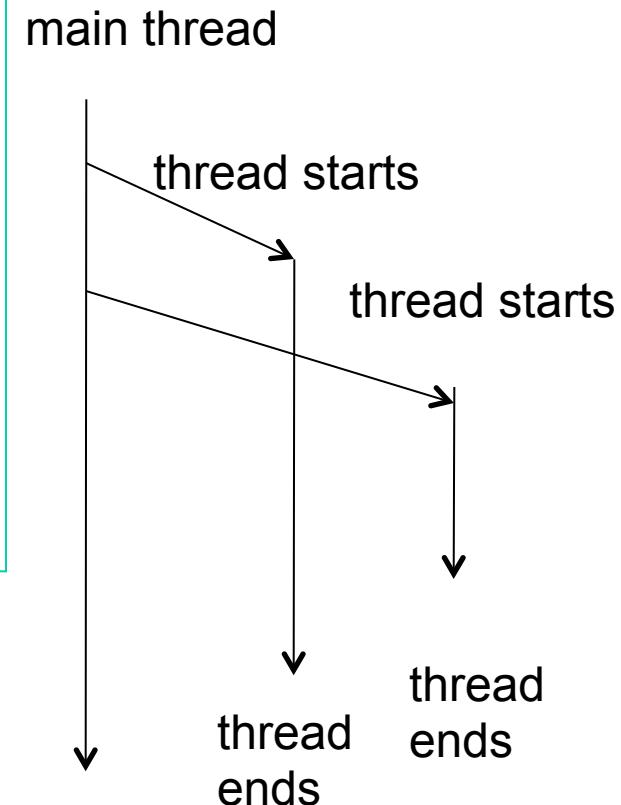
```
RequestHandler rh = new  
    RequestHandler(connSocket);  
Thread t = new Thread(rh);  
t.start();
```

Example: a Multi-threaded TCPServer

- Turn TCPServer into a multithreaded server by creating a thread for each accepted request

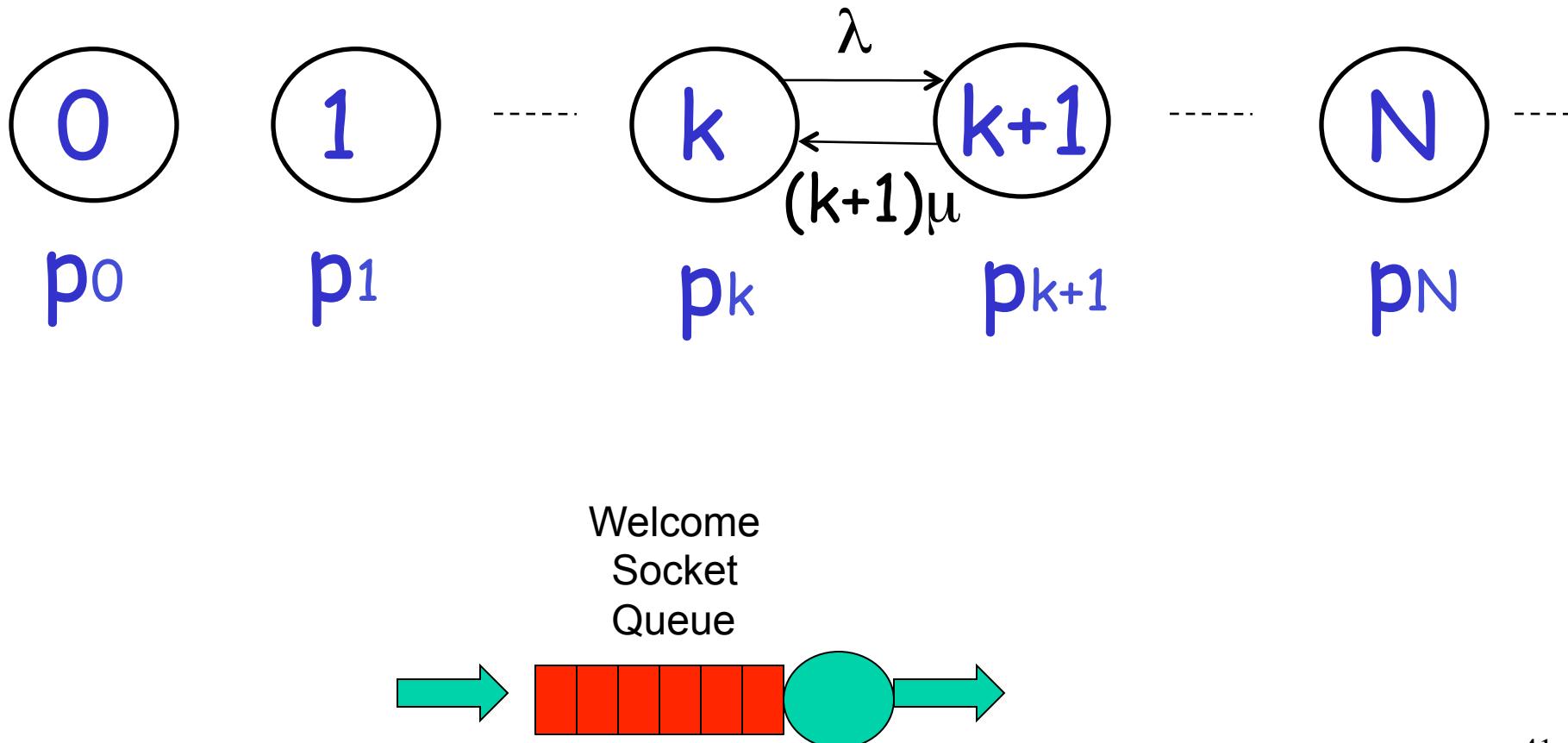
Per-Request Thread Server

```
main() {  
    ServerSocket s = new ServerSocket(port);  
    while (true) {  
        Socket conSocket = s.accept();  
        RequestHandler rh  
            = new RequestHandler(conSocket);  
        Thread t = new Thread (rh);  
        t.start();  
    }  
}
```

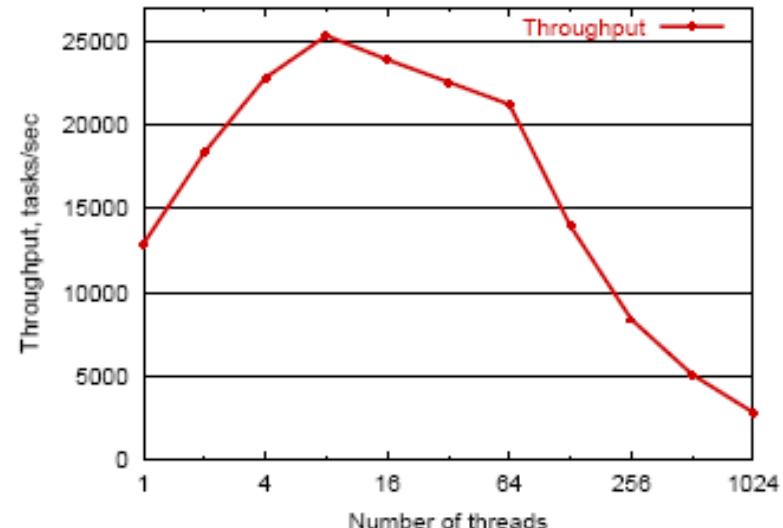
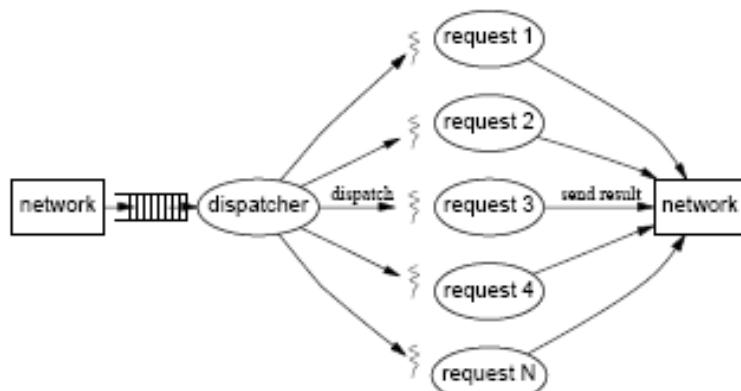


Try the per-request-thread TCP server: [TCPServerMT.java](#)

Modeling Per-Request Thread Server: Theory



Problem of Per-Request Thread: Reality



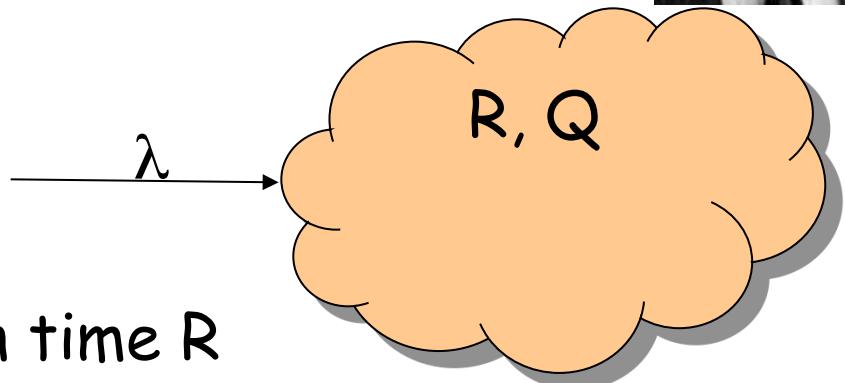
(937 MHz x86, Linux 2.2.14, each thread reading 8KB file)

- High thread creation/deletion overhead
- Too many threads → resource overuse → throughput meltdown → response time explosion
 - Q: given avg response time and connection arrival rate, how many threads active on avg?

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:

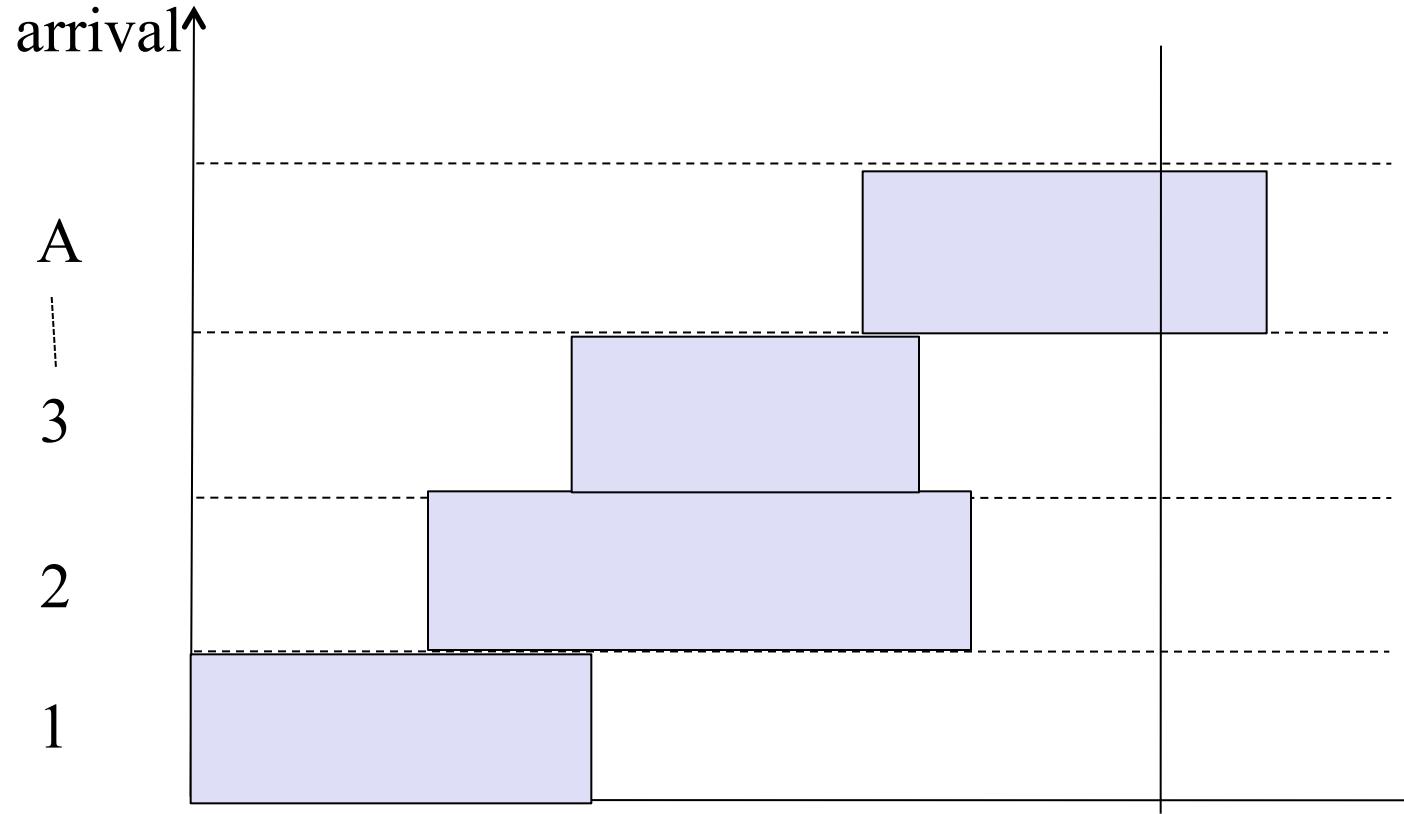


$$Q = \lambda R$$

Example: Yale College admits 1500 students each year, and mean time a student stays is 4 years, how many students are enrolled?

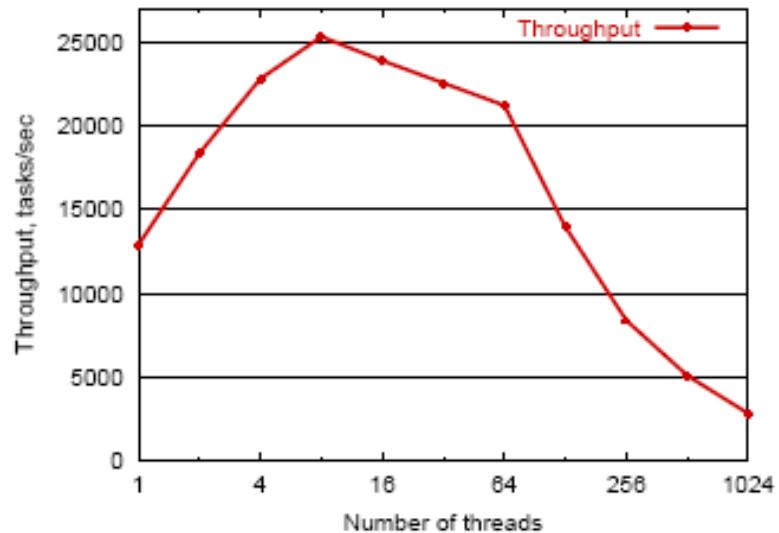
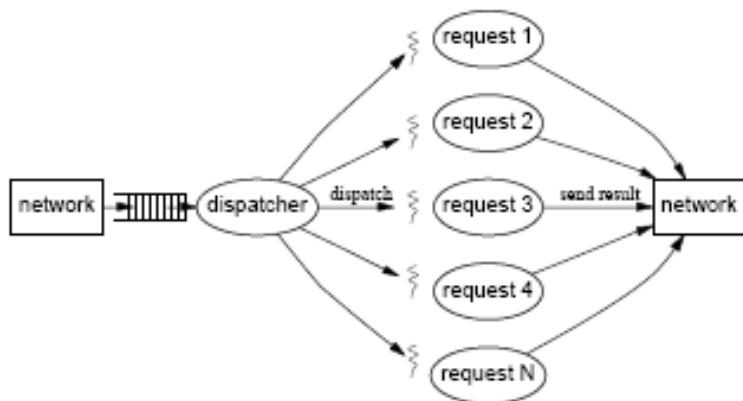
Little's Law

$$Q = \lambda R$$



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

Discussion: How to Address the Issue



(937 MHz x86, Linux 2.2.14, each thread reading 8KB file)

Network Applications: High-performance Server Design

Y. Richard Yang

<http://zoo.cs.yale.edu/classes/cs433/>

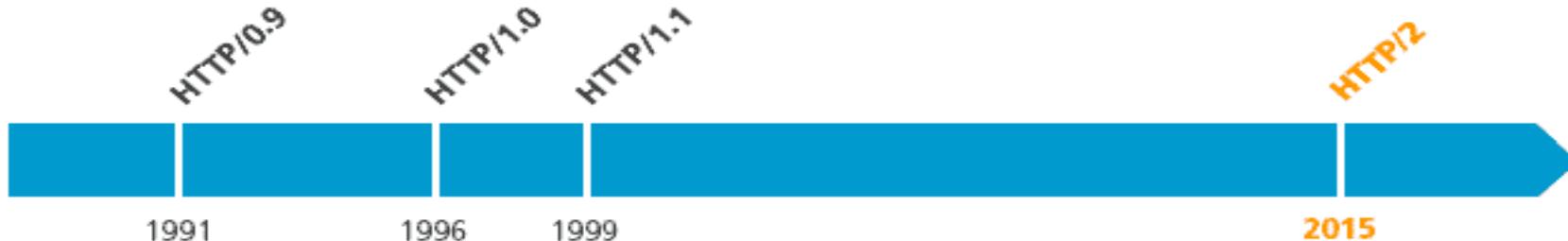
2/17/2016

Outline

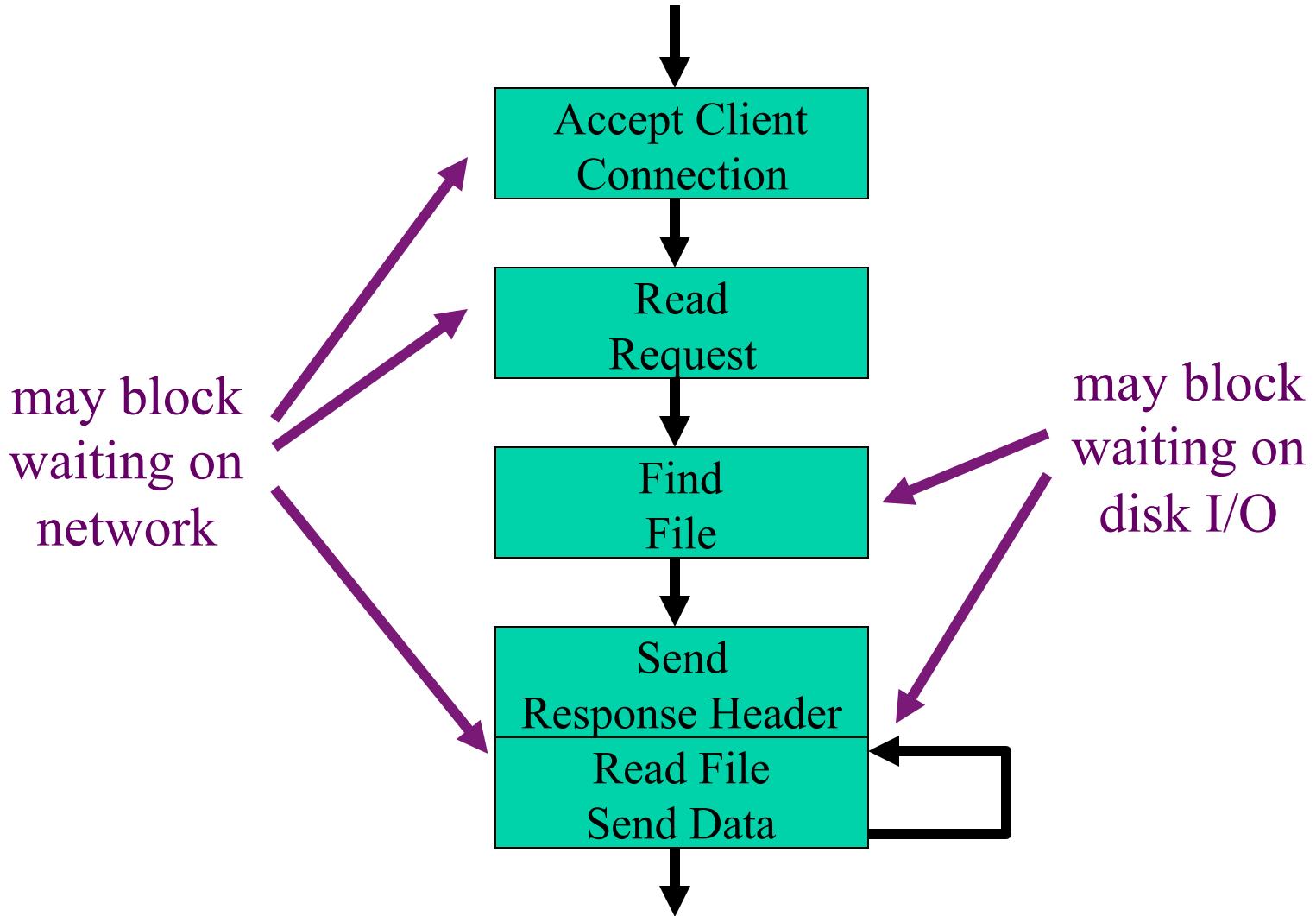
- Admin and recap
- High-performance network server design

Recap: Substantial Efforts to Speedup Basic HTTP/1.0

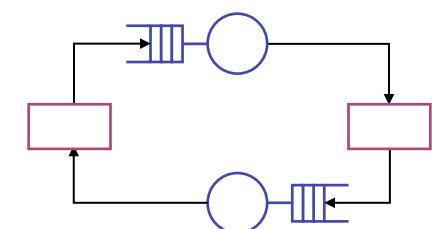
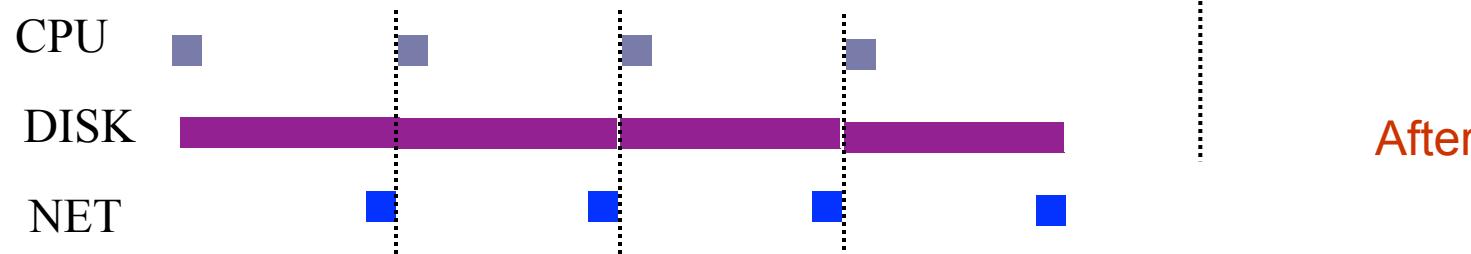
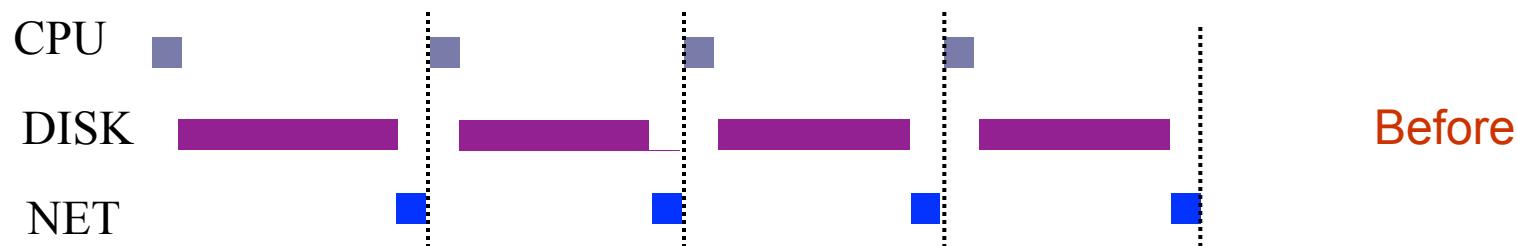
- Reduce the number of objects fetched [Browser cache]
- Reduce data volume [Compression of data]
- Reduce the latency to the server to fetch the content [Proxy cache]
- Increase concurrency [Multiple TCP connections]
- Remove the extra RTTs to fetch an object [Persistent HTTP, aka HTTP/1.1]
- Asynchronous fetch (multiple streams) using a single TCP [HTTP/2]
- Server push [HTTP/2]
- Header compression [HTTP/2]



Recap: Server Processing Steps

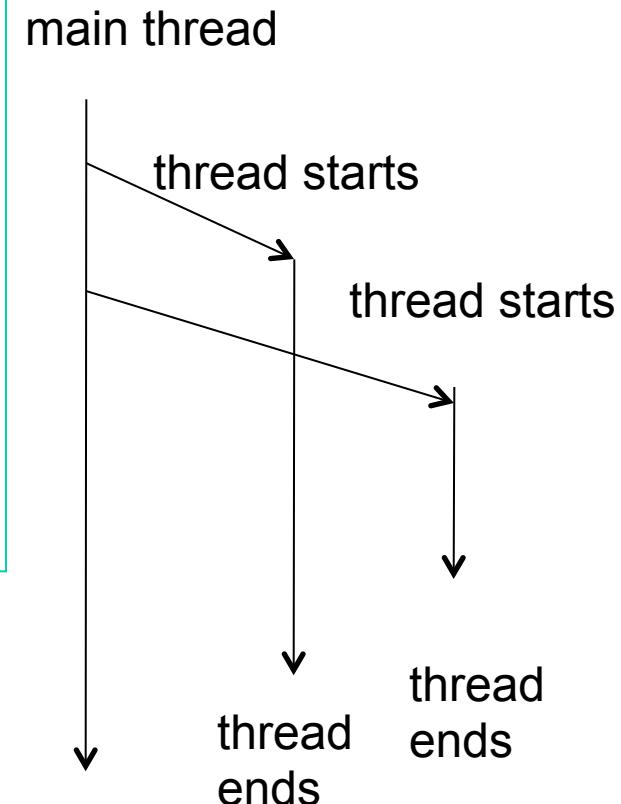


Recap: Design Server Limited Only by the Bottleneck

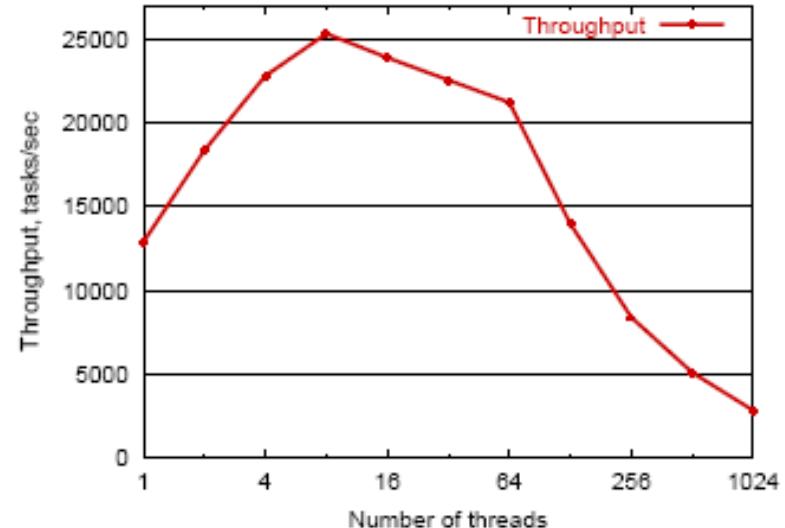
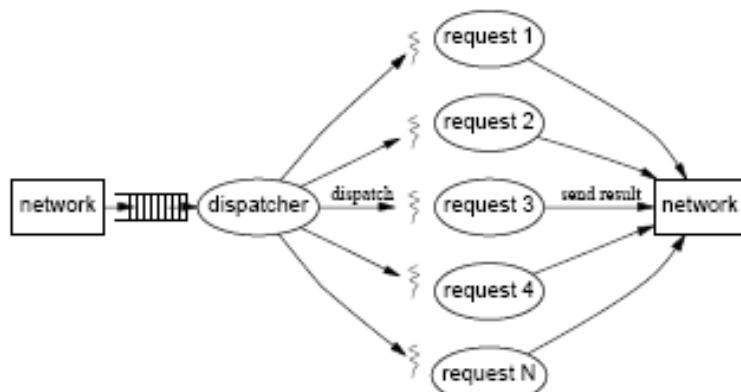


Recap: Per-Request Thread Server

```
main() {  
    ServerSocket s = new ServerSocket(port);  
    while (true) {  
        Socket conSocket = s.accept();  
        RequestHandler rh  
            = new RequestHandler(conSocket);  
        Thread t = new Thread (rh);  
        t.start();  
    }  
}
```



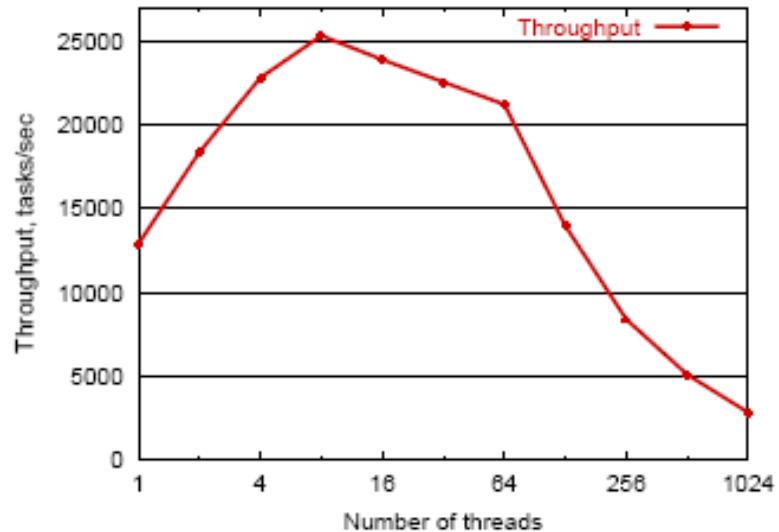
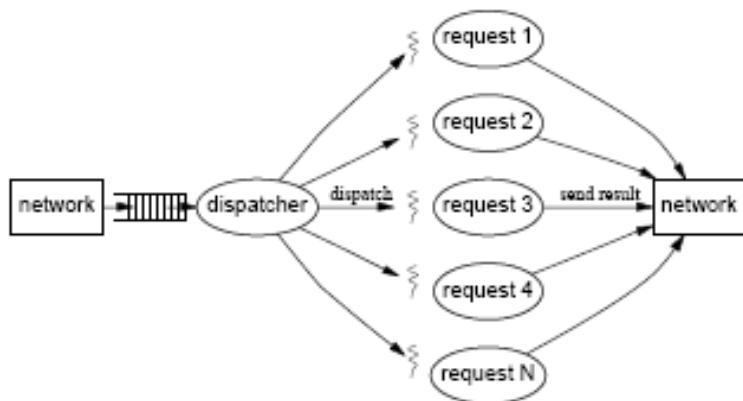
Recap: Problem of Per-Request Thread



(937 MHz x86, Linux 2.2.14, each thread reading 8KB file)

- High thread creation/deletion overhead
- Too many threads → resource overuse → throughput meltdown → response time explosion

Discussion: How to Address the Issue



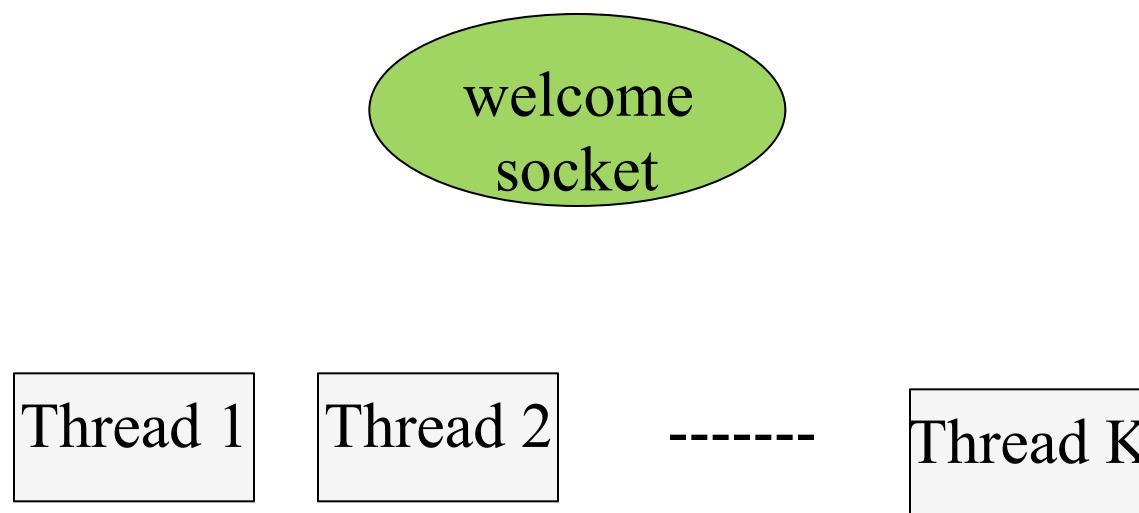
(937 MHz x86, Linux 2.2.14, each thread reading 8KB file)

Outline

- Admin and recap
- High-performance network server design
 - Overview
 - Threaded servers
 - Per-request thread
 - Thread pool

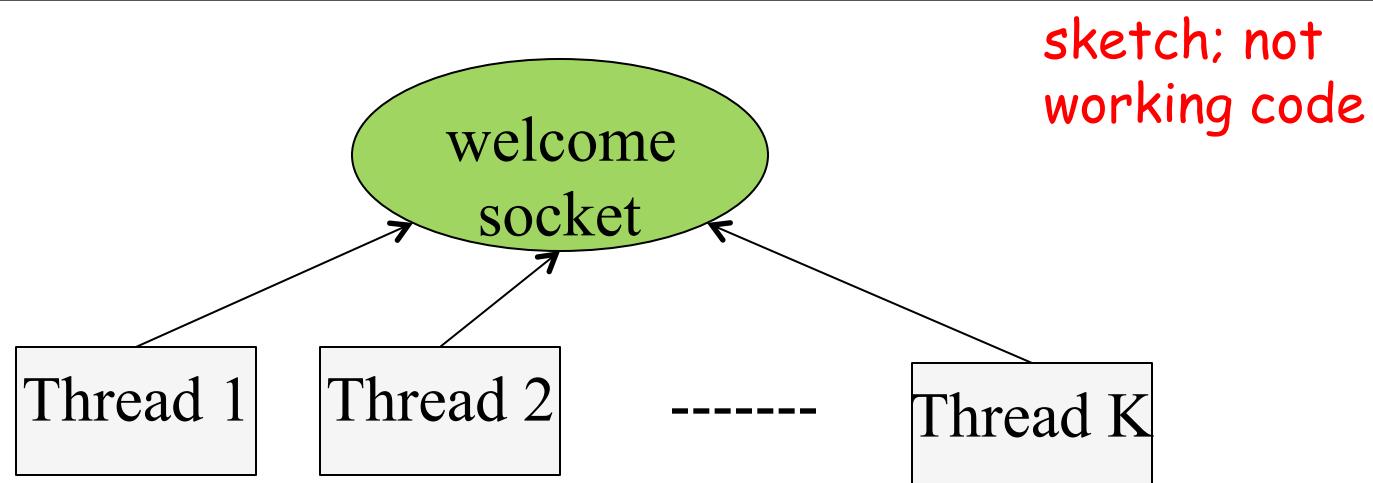
Using a Fixed Set of Threads (Thread Pool)

- Design issue: how to distribute the requests from the welcome socket to the thread workers



Design 1: Threads Share Access to the welcomeSocket

```
WorkerThread {  
    void run {  
        while (true) {  
            Socket myConnSock = welcomeSocket.accept();  
            // process myConnSock  
            myConnSock.close();  
        } // end of while  
    }  
}
```

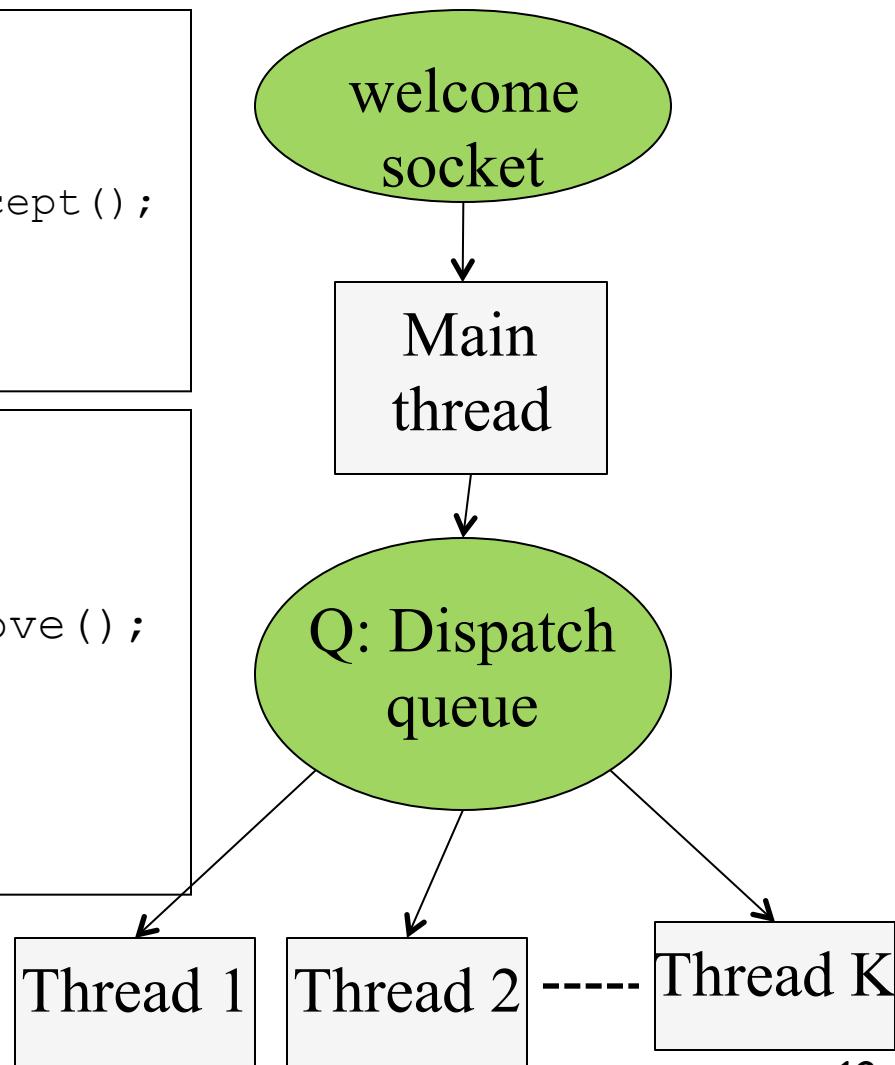


Design 2: Producer/Consumer

```
main {
    void run {
        while (true) {
            Socket con = welcomeSocket.accept();
            Q.add(con);
        } // end of while
    }
}
```

```
WorkerThread {
    void run {
        while (true) {
            Socket myConnSock = Q.remove();
            // process myConnSock
            myConnSock.close();
        } // end of while
    }
}
```

sketch; not working code



Common Issues Facing Designs 1 and 2

- Both designs involve multiple threads modifying the same data concurrently
 - Design 1: welcomeSocket
 - Design 2: Q
- In our original TCPServerMT, do we have multiple threads modifying the same data concurrently?

Concurrency and Shared Data

- Concurrency is easy if threads don't interact
 - Each thread does its own thing, ignoring other threads
 - Typically, however, threads need to communicate/coordinate with each other
 - Communication/coordination among threads is often done by *shared data*

Simple Example

```
public class ShareExample extends Thread {  
    private static int cnt = 0; // shared state, count  
                            // total increases  
  
    public void run() {  
        int y = cnt;  
        cnt = y + 1;  
    }  
  
    public static void main(String args[]) {  
        Thread t1 = new ShareExample();  
        Thread t2 = new ShareExample();  
        t1.start();  
        t2.start();  
        Thread.sleep(1000);  
        System.out.println("cnt = " + cnt);  
    }  
}
```

Q: What is the result of the program?

Simple Example

What if we add a `println`:

```
int y = cnt;  
System.out.println("Calculating...");  
cnt = y + 1;
```

What Happened?

- ❑ A thread was preempted in the middle of an operation
- ❑ The operations from reading to writing `cnt` should be *atomic* with no interference access to `cnt` from other threads
- ❑ But the scheduler interleaves threads and caused a *race condition*

- ❑ Such bugs can be extremely hard to reproduce, and also hard to debug

Synchronization

- Refers to mechanisms allowing a programmer to control the execution order of some operations across different threads in a concurrent program.

- We use Java as an example to see synchronization mechanisms

- We'll look at locks first.

Java Lock (1.5)

```
interface Lock {  
    void lock();  
    void unlock();  
    ... /* Some more stuff, also */  
}  
class ReentrantLock implements Lock { ... }
```

- Only one thread can hold a lock at once
- Other threads that try to acquire it *block* (or become suspended) until the lock becomes available
- *Reentrant lock can be reacquired by same thread*
 - As many times as desired
 - No other thread may acquire a lock until it has been released the same number of times that it has been acquired
 - Do not worry about the reentrant perspective, consider it a lock

Java Lock

□ Fixing the ShareExample.java problem

```
import java.util.concurrent.locks.*;
public class ShareExample extends Thread {
    private static int cnt = 0;
    static Lock lock = new ReentrantLock();

    public void run() {
        lock.lock();
        int y = cnt;
        cnt = y + 1;
        lock.unlock();
    }
    ...
}
```

Java Lock

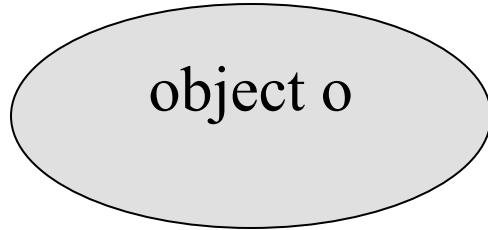
- It is recommended to use the following pattern

```
...
lock.lock();
try {
    // processing body
} finally {
    lock.unlock();
}
```

Java synchronized

- This pattern is really common
 - Acquire lock, do something, release lock after we are done, **under any circumstances, even if exception was raised, the method returned in the middle, etc.**
- Java has a language construct for this
 - `synchronized (obj) { body }`
- Every Java object has an implicitly associated lock
 - Obtains the lock associated with `obj`
 - Executes `body`
 - Release lock when scope is exited
 - Even in cases of exception or method return

Discussion



- An object and its associated lock are different !
- Holding the lock on an object does not affect what you can do with that object in any way
- Examples:
 - `synchronized(o) { ... } // acquires lock named o`
 - `o.f(); // someone else can call o's methods`
 - `o.x = 3; // someone else can read and write o's fields`

Synchronization on this

```
class C {  
    int cnt;  
    void inc() {  
        synchronized (this) {  
            cnt++;  
        } // end of sync  
    } // end of inc  
}
```

```
C c = new C();
```

```
Thread 1  
c.inc();
```

```
Thread 2  
c.inc();
```

- ❑ A program can often use `this` as the object to lock
- ❑ Does the program above have a data race?
 - No, both threads acquire locks on the same object before they access shared data

Synchronization on this

```
class C {  
    static int cnt;  
    void inc() {  
        synchronized (this) {  
            cnt++;  
        } // end of sync  
    } // end of inc  
  
    void dec() {  
        synchronized (this) {  
            cnt--;  
        } // end of sync  
    } // end of dec  
}
```

```
C c = new C();
```

```
Thread 1  
c.inc();
```

```
Thread 2  
c.dec();
```

- ❑ Does the program above have a data race?
 - No, both threads acquire locks on the same object before they access shared data

Example

- See
 - ShareWelcome/Server.java
 - ShareWelcome/Service Thread.java

Discussion

- You would not need the lock for `accept` if Java were to label the call as thread safe (`synchronized`)
- One reason Java does not specify `accept` as thread safe is that one could register your own socket implementation with [ServerSocket.setSocketFactory](#)
- Always consider thread safety in your design
 - If a resource is shared through concurrent read/write, write/write), consider thread-safe issues.

Why not Synchronization

- Synchronized method invocations generally are going to be slower than non-synchronized method invocations

- Synchronization gives rise to the possibility of deadlock, a severe performance problem in which your program appears to hang

Synchronization Overhead

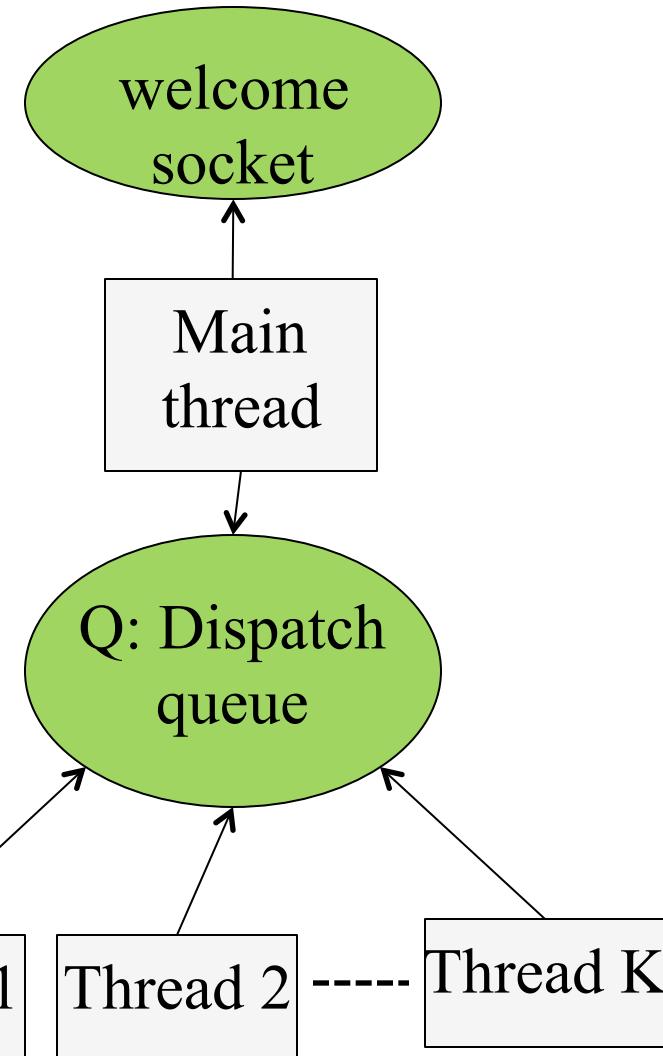
- Try SyncOverhead.java

Method	Time (ms; 5,000,000 exec)
no sync	9 ms
synchronized method	116 ms
synchronized on this	110 ms
lock	117 ms
lock and finally	113 ms

Design 2: Producer/Consumer

```
main {
    void run {
        while (true) {
            Socket con = welcomeSocket.accept();
            Q.add(con);
        } // end of while
    }
}
```

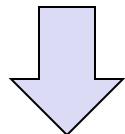
```
WorkerThread {
    void run {
        while (true) {
            Socket myConnSock = Q.remove();
            // process myConnSock
            myConnSock.close();
        } // end of while
    }
}
```



How to turn it into working code?

Main

```
main {
    void run {
        while (true) {
            Socket con = welcomeSocket.accept();
            Q.add(con);
        } // end of while
    }
}
```



```
main {
    void run {
        while (true) {
            Socket con = welcomeSocket.accept();
            synchronized(Q) {
                Q.add(con);
            }
        } // end of while
    }
}
```

Worker

```
WorkerThread {  
    void run {  
        while (true) {  
            Socket myConnSock = Q.remove();  
            // process myConnSock  
            myConnSock.close();  
        } // end of while  
    }  
}
```



```
while (true) {  
    // get next request  
    Socket myConn = null;  
    while (myConn==null) {  
        synchronize(Q) {  
            if (!Q.isEmpty())  
                myConn = (Socket) Q.remove();  
        }  
    } // end of while  
    // process myConn  
}
```

Example

- try
 - ShareQ/Server.java
 - ShareQ/ServiceThread.java

Problem of ShareQ Design

- ❑ Thread continually spins (**busy wait**) until a condition holds

```
while (true) { // spin
    lock;
    if (Q.condition) // {
        // do something
    } else {
        // do nothing
    }
    unlock
} //end while
```

- ❑ Can lead to high utilization and slow response time
- ❑ Q: Does the shared welcomeSock have busy-wait?

Outline

- Admin and recap
- High-performance network server design
 - Overview
 - Threaded servers
 - Per-request thread
 - Thread pool
 - Polling
 - Wait/notify

Solution: Suspension

- Put thread to sleep to avoid busy spin
- Thread life cycle: while a thread executes, it goes through a number of different phases
 - New: created but not yet started
 - Runnable: is running, or can run on a free CPU
 - Blocked: waiting for socket/I/O, a lock, or **suspend** (wait)
 - Sleeping: paused for a user-specified interval
 - Terminated: completed

Solution: Suspension

- Thread stops execution until notified that the condition may be true

```
while (true) {  
    // get next request  
    Socket myConn = null;  
    while (myConn==null) {  
        lock Q;  
        if (Q.isEmpty()) // {  
            // stop and wait ← Hold lock?  
        } else {  
            // get myConn from Q  
        }  
        unlock Q;  
    }  
    // get the next request; process  
}
```

Alternative: Suspension

- Thread stops execution until notified that the condition may be true

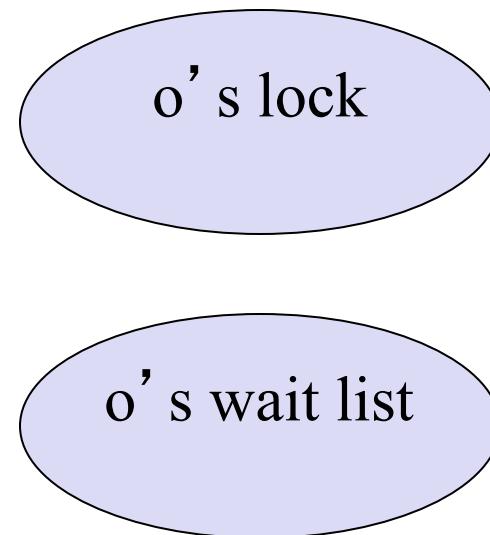
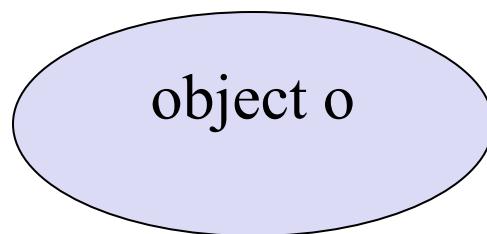
```
while (true) {  
    // get next request  
    Socket myConn = null;  
    while (myConn==null) {  
        lock Q;  
        if (Q.isEmpty()) // {  
            // stop and wait ←  
        } else {  
            // get myConn from Q  
        }  
        unlock Q;  
    }  
    // get the next request; process  
}
```

Design pattern:

- Need to release lock to avoid deadlock (to allow main thread write into Q)
- Typically need to reacquire lock after waking up

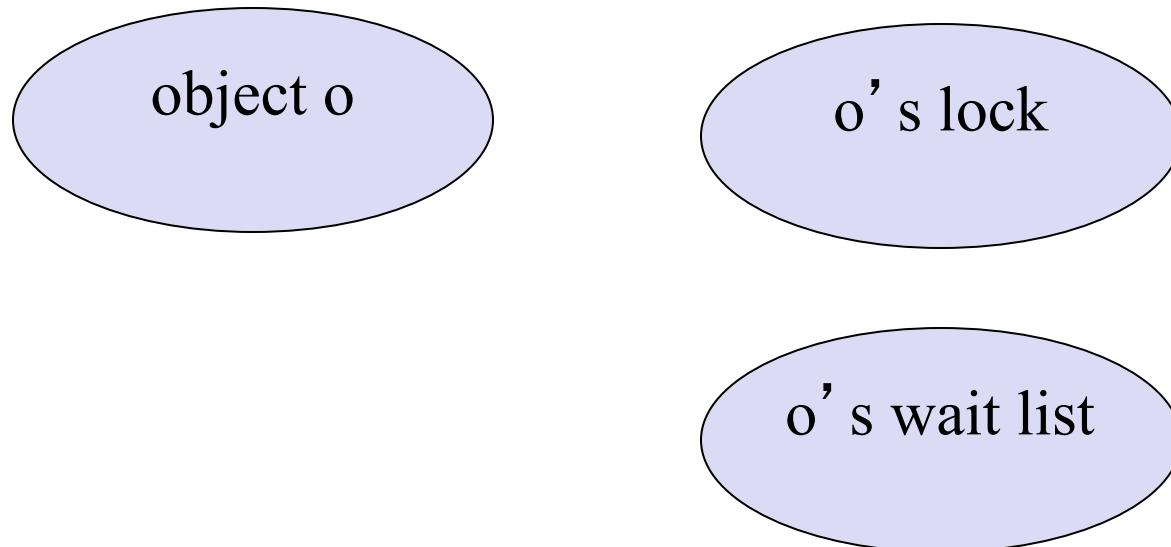
Wait-sets and Notification

- ❑ Every Java Object has an associated wait-set (called wait list) in addition to a lock object



Wait-sets and Notification

- ❑ Wait list object can be manipulated only while the object lock is held
 - Otherwise, `IllegalMonitorStateException` is thrown



Wait-sets

- Thread enters the wait-set by invoking `wait()`
 - `wait()` releases the lock
 - No other held locks are released
 - then the thread is suspended
- Can add optional time `wait(long millis)`
 - `wait()` is equivalent to `wait(0)` - wait forever
 - for robust programs, it is typically a good idea to add a timer

Worker

```
while (true) {
    // get next request
    Socket myConn = null;
    while (myConn==null) {
        lock Q;
        if (! Q.isEmpty()) // {
            myConn = Q.remove();
        }
        unlock Q;
    } // end of while
    // get the next request; process
}
```

```
while (true) {
    // get next request
    Socket myConn = null;
    synchronized(Q) {
        while (Q.isEmpty()) {
            Q.wait();
        }
        myConn = Q.remove();
    } // end of sync
    // process request in myConn
} // end of while
```

Note the while loop; no guarantee that Q is not empty when wake up

Wait-set and Notification (cont)

- Threads are released from the wait-set when:
 - `notifyAll()` is invoked on the object
 - All threads released (typically recommended)
 - `notify()` is invoked on the object
 - One thread selected at ‘random’ for release
 - The specified time-out elapses
 - The thread has its `interrupt()` method invoked
 - `InterruptedException` thrown
 - A spurious wakeup occurs
 - Not (yet!) spec’ ed but an inherited property of underlying synchronization mechanisms e.g., POSIX condition variables

Notification

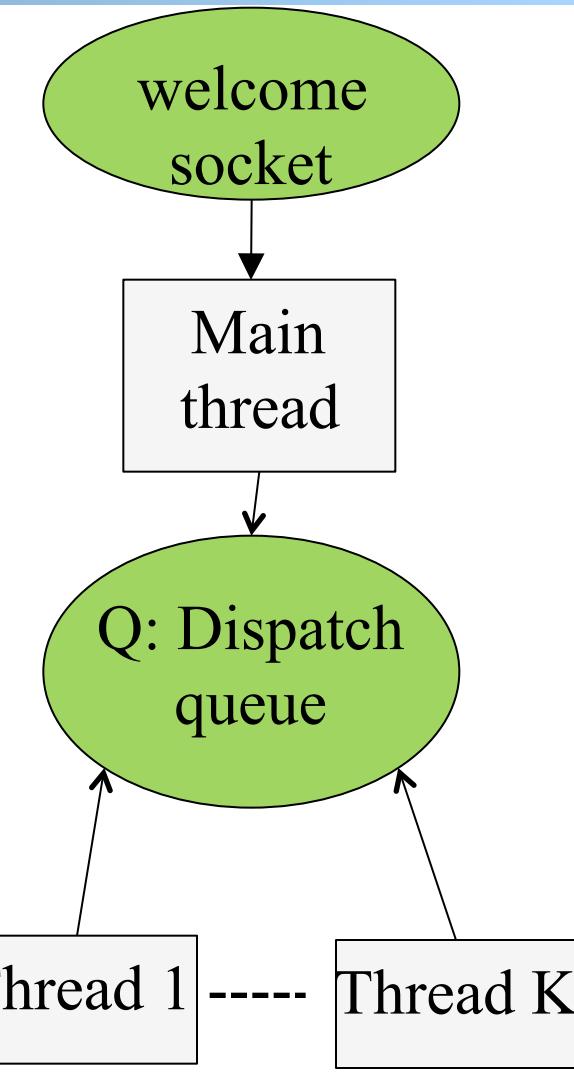
- ❑ Caller of `notify()` must hold lock associated with the object
- ❑ Those threads awoken must reacquire lock before continuing
 - (This is part of the function; you don't need to do it explicitly)
 - Can't be acquired until notifying thread releases it
 - A released thread contends with all other threads for the lock

Main Thread

```
main {
    void run {
        while (true) {
            Socket con = welcomeSocket.accept();
            synchronized(Q) {
                Q.add(con);
            }
        } // end of while
    }
}
```



```
main {
    void run {
        while (true) {
            Socket con = welcomeSocket.accept();
            synchronize(Q) {
                Q.add(con);
                Q.notifyAll();
            }
        } // end of while
    }
}
```

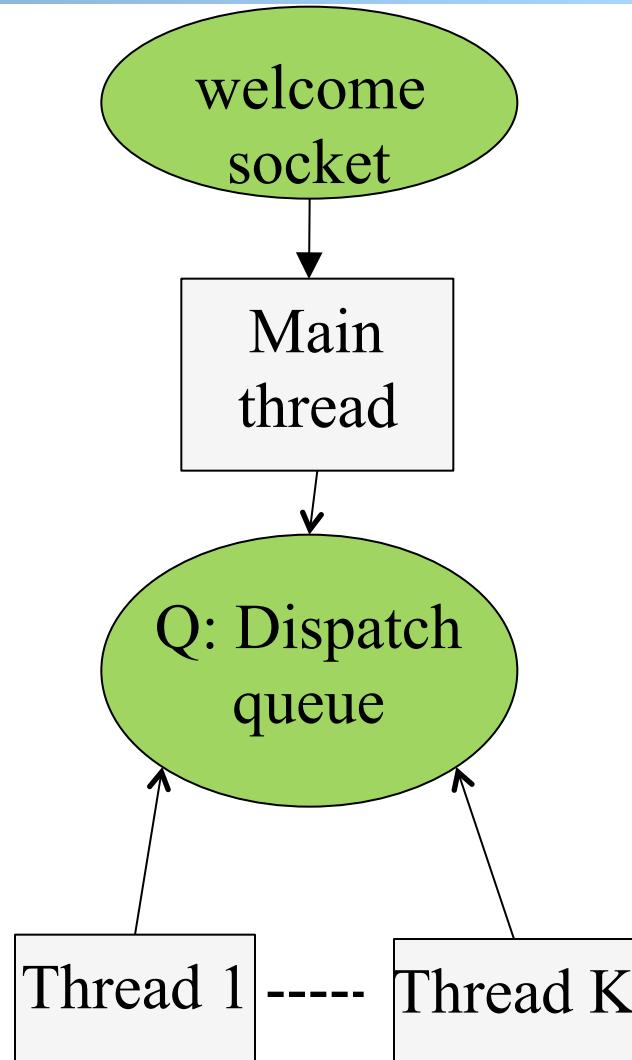


Worker

```
while (true) {  
    // get next request  
    Socket myConn = null;  
    while (myConn==null) {  
        synchronize(Q) {  
            if (! Q.isEmpty()) // {  
                myConn = Q.remove();  
            }  
        }  
    } // end of while  
    // process myConn  
}
```

Busy wait

```
while (true) {  
    // get next request  
    Socket myConn = null;  
    while (myConn==null) {  
        synchronize(Q) {  
            if (! Q.isEmpty()) // {  
                myConn = Q.remove();  
            } else {  
                Q.wait();  
            }  
        }  
    } // end of while  
    // process myConn  
}
```



Worker: Another Format

```
while (true) {  
    // get next request  
    Socket myConn = null;  
    synchronized(Q) {  
        while (Q.isEmpty()) {  
            Q.wait();  
        }  
        myConn = Q.remove();  
    } // end of sync  
    // process request in myConn  
} // end of while
```

Note the while loop; no guarantee that Q is not empty when wake up



Example

- See
 - WaitNotify/Server.java
 - WaitNotify/ServiceThread.java

Summary: Guardian via Suspension: Waiting

```
synchronized (obj) {  
    while (!condition) {  
        try { obj.wait(); }  
        catch (InterruptedException ex)  
        { ... }  
    } // end while  
    // make use of condition  
} // end of sync
```

- **Golden rule:** Always test a condition in a loop
 - Change of state may not be what you need
 - Condition may have changed again
- Break the rule only after you are sure that it is safe to do so

Summary: Guarding via Suspension: Changing a Condition

```
synchronized (obj) {  
    condition = true;  
    obj.notifyAll(); // or obj.notify()  
}
```

- Typically use `notifyAll()`
- There are subtle issues using `notify()`, in particular when there is interrupt

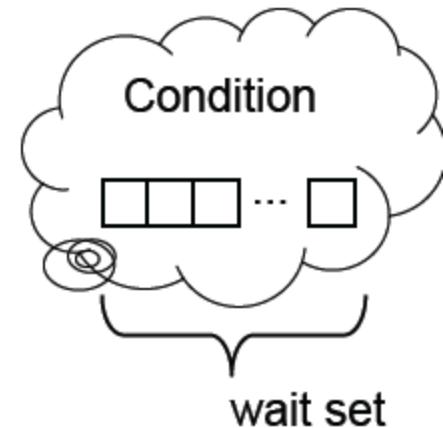
Note

- Use of `wait()`, `notifyAll()` and `notify()` similar to
 - Condition queues of classic Monitors
 - Condition variables of POSIX PThreads API
 - In C# it is called Monitor (
<http://msdn.microsoft.com/en-us/library/ms173179.aspx>)
- Python Thread module in its Standard Library is based on Java Thread model (<https://docs.python.org/3/library/threading.html>)
 - "The design of this module is loosely based on Java's threading model. However, where Java makes locks and condition variables basic behavior of every object, they are separate objects in Python."

Java (1.5)

```
interface Lock { Condition newCondition(); ... }  
interface Condition {  
    void await();  
    void signalAll(); ...  
}
```

- Condition created from a Lock
- await called with lock held
 - Releases the lock
 - But not any other locks held by this thread
 - Adds this thread to wait set for lock
 - Blocks the thread
- signalAll called with lock held
 - Resumes all threads on lock's wait set
 - Those threads must reacquire lock before continuing
 - (This is part of the function; you don't need to do it explicitly)



Producer/Consumer Example

```
Lock lock = new ReentrantLock();
Condition ready = lock.newCondition();
boolean valueReady = false;
Object value;

void produce(Object o) {
    lock.lock();
    while (valueReady)
        ready.await();
    value = o;
    valueReady = true;
    ready.signalAll();
    lock.unlock();
}

Object consume() {
    lock.lock();
    while (!valueReady)
        ready.await();
    Object o = value;
    valueReady = false;
    ready.signalAll();
    lock.unlock();
}
```

Beyond Class: Complete Java Concurrency Framework

Executors

- Executor
- ExecutorService
- ScheduledExecutorService
- Callable
- Future
- ScheduledFuture
- Delayed
- CompletionService
- ThreadPoolExecutor
- ScheduledThreadPoolExecutor
- AbstractExecutorService
- Executors
- FutureTask
- ExecutorCompletionService

Queues

- BlockingQueue
- ConcurrentLinkedQueue
- LinkedBlockingQueue
- ArrayBlockingQueue
- SynchronousQueue
- PriorityBlockingQueue
- DelayQueue

Concurrent Collections

- ConcurrentHashMap
- CopyOnWriteArrayList
- CopyOnWriteArraySet

Synchronizers

- CountDownLatch
- Semaphore
- Exchanger
- CyclicBarrier

Locks: `java.util.concurrent.locks`

- Lock
- Condition
- ReadWriteLock
- AbstractQueuedSynchronizer
- LockSupport
- ReentrantLock
- ReentrantReadWriteLock

Atomics: `java.util.concurrent.atomic`

- Atomic[Type]
- Atomic[Type]Array
- Atomic[Type]FieldUpdater
- Atomic{Markable,Stampable}Reference

See jcf slides for a tutorial.

Correctness

- ❑ What do you analyze to show the server design is correct?

```
while (true) {  
    // get next request  
    Socket myConn = null;  
    synchronized(Q) {  
        while (Q.isEmpty()) {  
            Q.wait();  
        }  
        myConn = Q.remove();  
    } // end of sync  
    // process request in myConn  
} // end of while
```

```
main {  
    void run {  
        while (true) {  
            Socket con = welcomeSocket.accept();  
            synchronized(Q) {  
                Q.add(con);  
                Q.notifyAll();  
            }  
        } // end of while  
    }  
}
```

Key Correctness Properties

- Safety
- Liveness (progress)
- Fairness
 - For example, in some settings, a designer may want the threads to share load equally

Safety Properties

- What safety properties?
 - No read/write; write/write conflicts
 - holding lock Q before reading or modifying shared data Q and Q.wait_list
 - Q.remove() is not on an empty queue
- There are formal techniques to model server programs and analyze their properties, but we will use basic analysis
 - This is enough in many cases

Make Program Explicit

```
main {
    void run {
        while (true) {
            Socket con = welcomeSocket.accept();
            synchronize(Q) {
                Q.add(con);
                Q.notifyAll();
            }
        } // end of while
    }
}
```

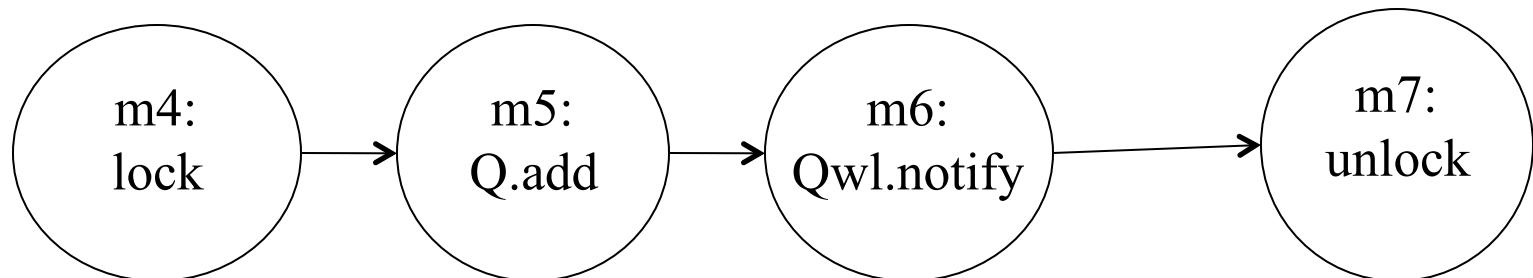
```
1. main {
    void run {
2.     while (true) {
3.         Socket con = welcomeSocket.accept();
4.         lock(Q) {
5.             Q.add(con);
6.             notify Q.wait_list; // Q.notifyAll();
7.             unlock(Q);
8.         } // end of while
9.     }
}
```

```
while (true) {  
    // get next request  
    Socket myConn = null;  
    synchronized(Q) {  
        while (Q.isEmpty()) {  
            Q.wait();  
        }  
        myConn = Q.remove();  
    } // end of sync  
    // process request in myConn  
} // end of while
```

```
1.while (true) {  
    // get next request  
2.    Socket myConn = null;  
3.    lock(Q);  
4.        while (Q.isEmpty()) {  
5.            unlock(Q)  
6.            add to Q.wait_list;  
7.            yield until marked to wake; //wait  
8.            lock(Q);  
9.        }  
10.       myConn = Q.remove();  
11.       unlock(Q);  
        // process request in myConn  
12.    } // end of while
```

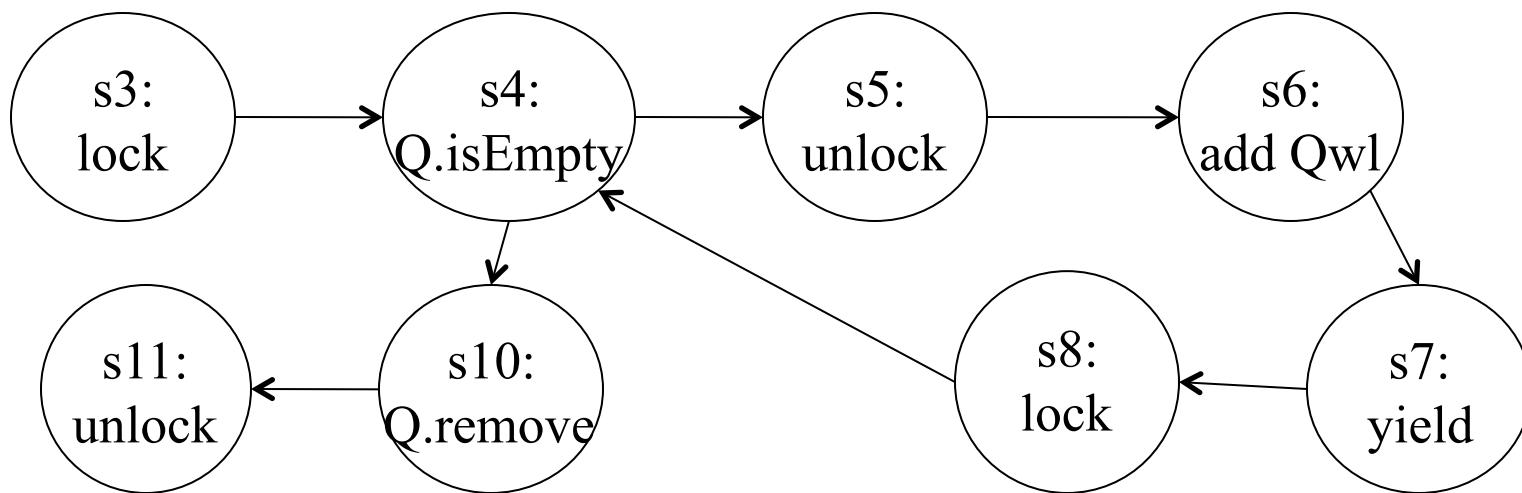
Statements to State

```
1. main {
    void run {
2.     while (true) {
3.         Socket con = welcomeSocket.accept();
4.         lock(Q) {
5.             Q.add(con);
6.             notify Q.wait_list; // Q.notifyAll();
7.             unlock(Q);
8.         } // end of while
9.     }
}
```

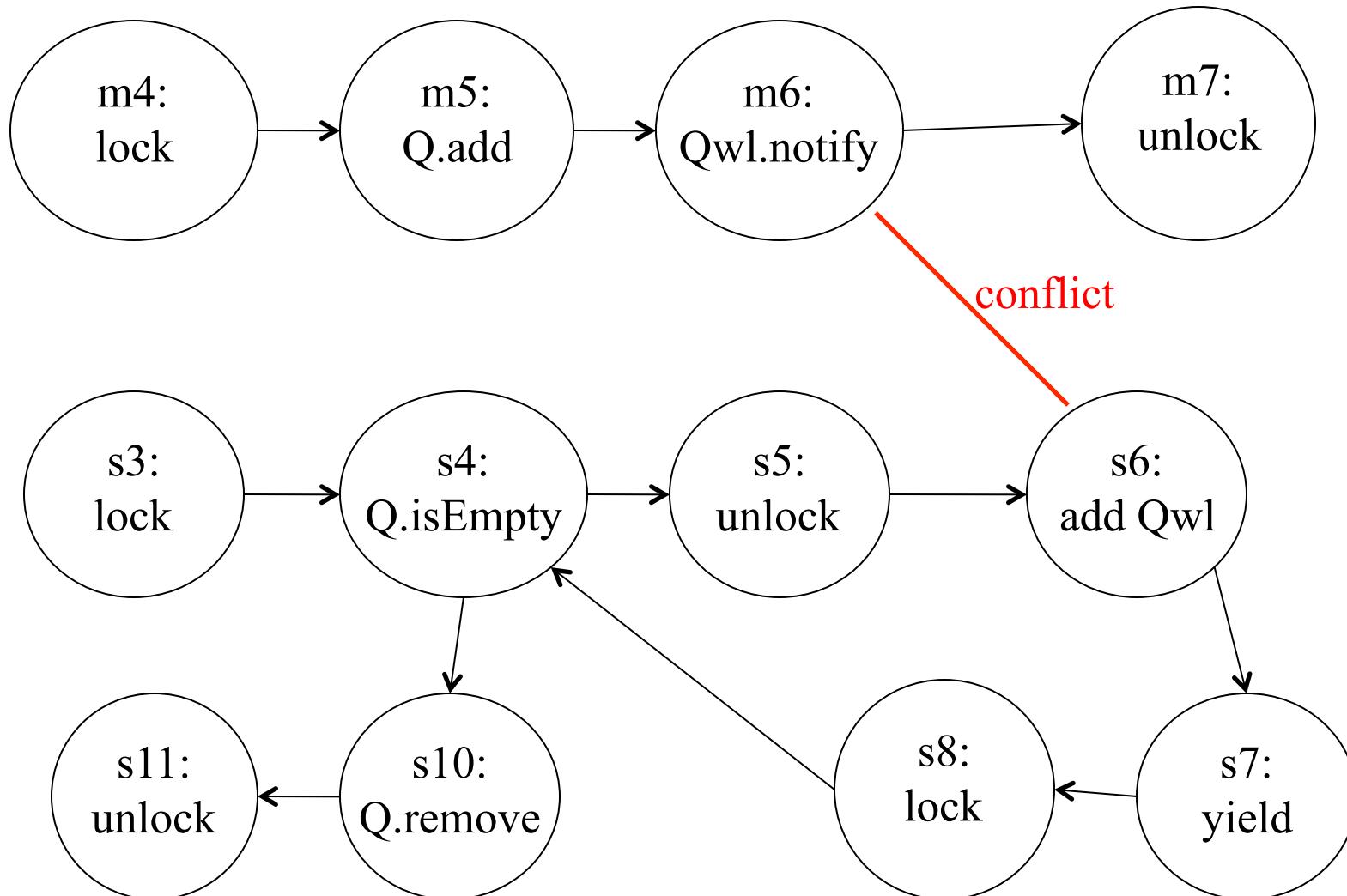


Statements

```
1.while (true) {  
    // get next request  
2.    Socket myConn = null;  
3.    lock(Q);  
4.    while (Q.isEmpty()) {  
5.        unlock(Q)  
6.        add to Q.wait_list;  
7.        yield; //wait  
8.        lock(Q);  
9.    }  
10.   myConn = Q.remove();  
11.   unlock(Q);  
    // process request in myConn  
12.} // end of while
```



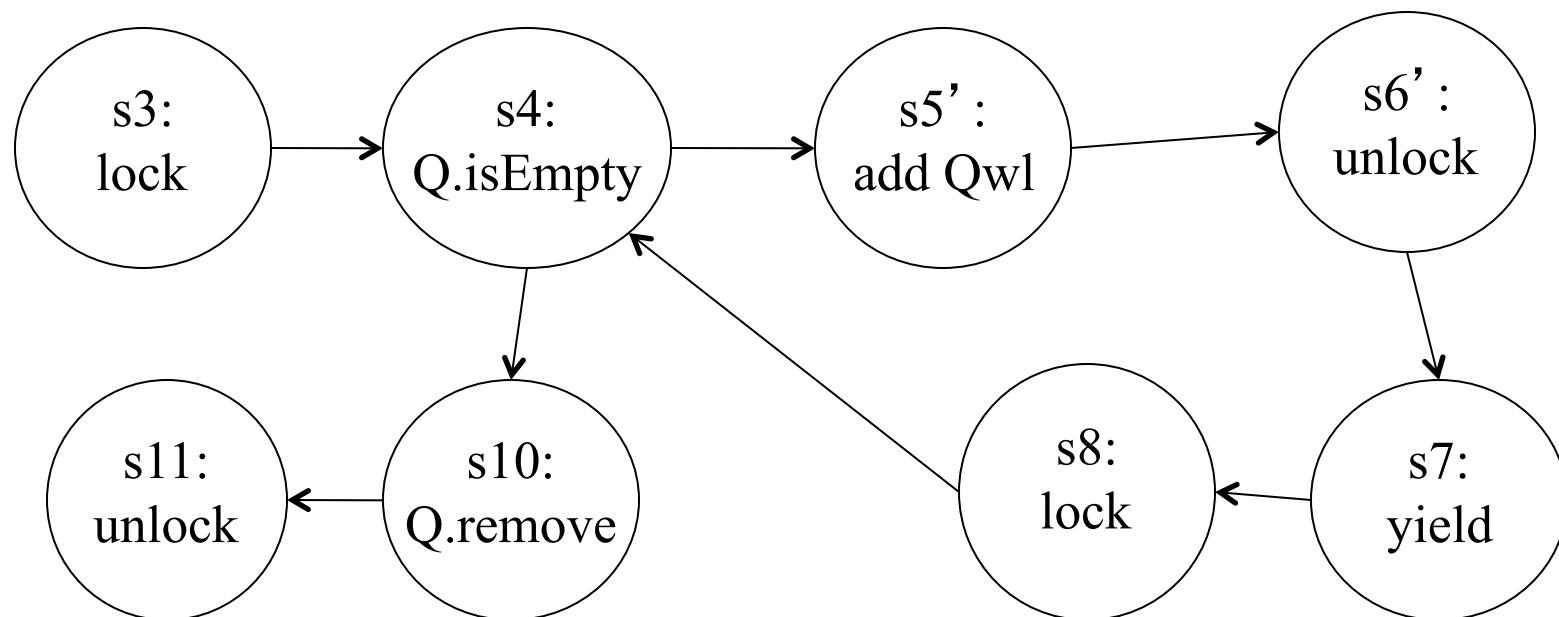
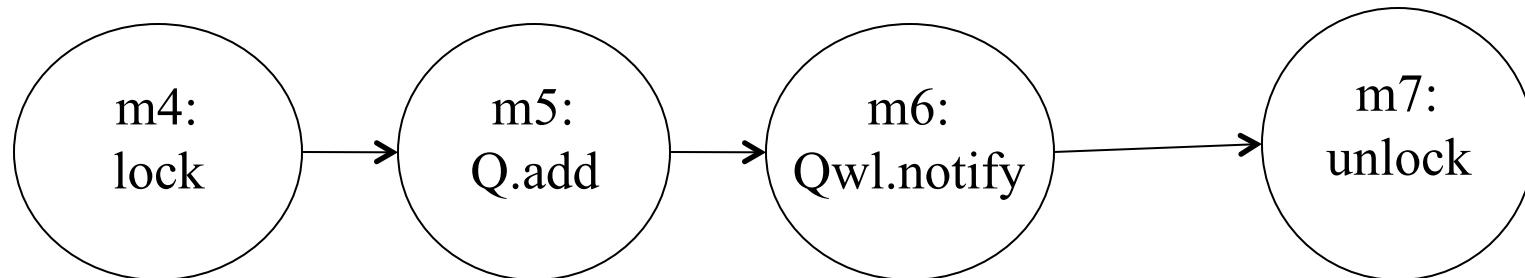
Check Safety



Real Implementation of wait

```
1.while (true) {  
    // get next request  
2.Socket myConn = null;  
3.    lock(Q);  
4.        while (Q.isEmpty()) {  
5.            add to Q.wait_list;  
6.            unlock(Q); after add to wait list  
7.            yield; //wait  
8.            lock(Q);  
9.        }  
10.       myConn = Q.remove();  
11.       unlock(Q);  
        // process request in myConn  
12.} // end of while
```

States

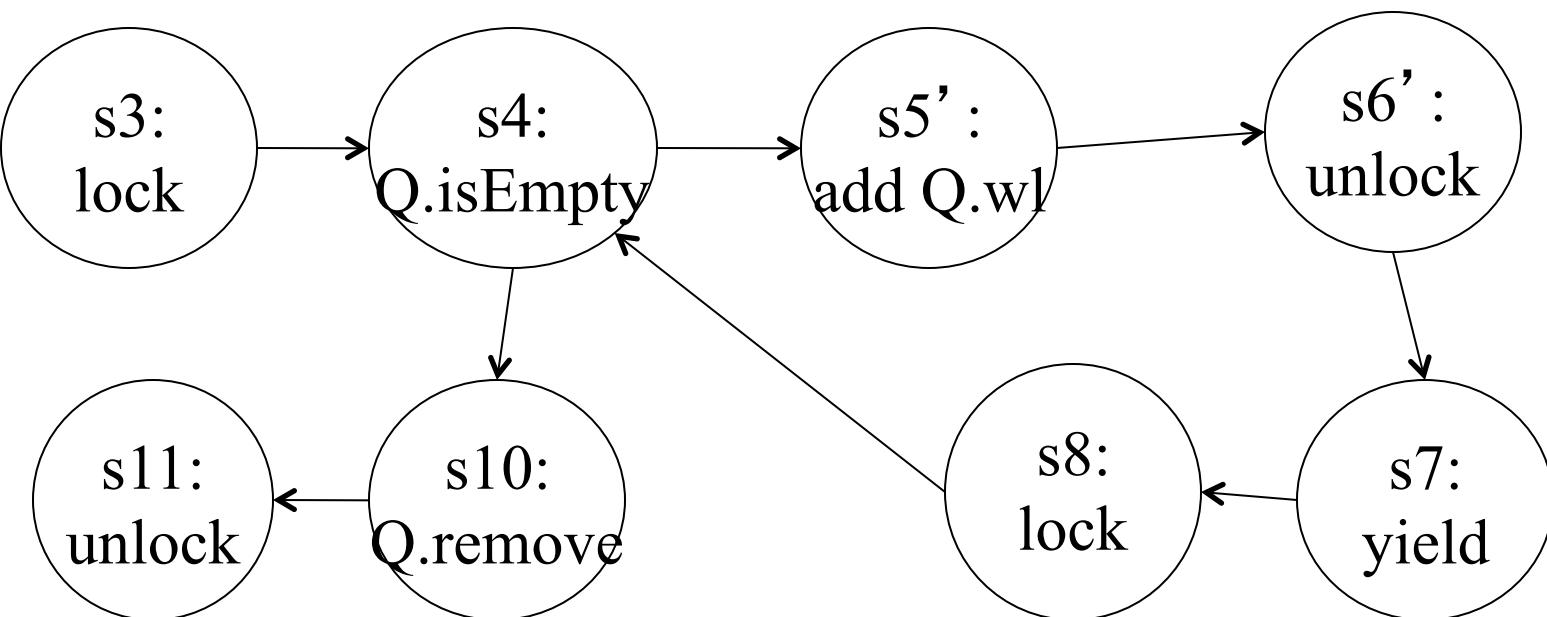


Liveness Properties

- What liveness (progress) properties?
 - main thread can always add to Q
 - every connection in Q will be processed

Main Thread can always add to Q

- Assume main is blocked
- Suppose Q is not empty, then each iteration removes one element from Q
- In finite number of iterations, all elements in Q are removed and all service threads unlock and block



Each connection in Q is processed

- Cannot be guaranteed unless
 - there is fairness in the thread scheduler, or
 - put a limit on Q size to block the main thread

Blocking Queues in Java

- Design Pattern for producer/consumer pattern with blocking, e.g.,
 - put/take
- Two handy implementations
 - LinkedBlockingQueue (FIFO, may be bounded)
 - ArrayBlockingQueue (FIFO, bounded)
 - (plus a couple more)

<https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html>

Network Applications: High-performance Server Design

Y. Richard Yang

<http://zoo.cs.yale.edu/classes/cs433/>

2/22/2016

Admin

- Assignment three will be posted tomorrow.
- Please start to think about projects
- Dates for exam 1?

Recap: Thread-Based Network Servers

- Why: threads (execution sequences) so that only one thread is blocked
- How:
 - Per-request thread
 - problem: large # of threads and their creations/deletions may let overhead grow out of control
 - Thread pool
 - Design 1: Service threads compete on the welcome socket
 - Design 2: Service threads and the main thread coordinate on the shared queue
 - polling (busy wait)
 - suspension: wait/notify

Recap: Program Correctness Analysis

□ Safety

- No read/write; write/write conflicts
 - holding lock Q before reading or modifying shared data Q and Q.wait_list
- Q.remove() is not on an empty queue

□ Liveness (progress)

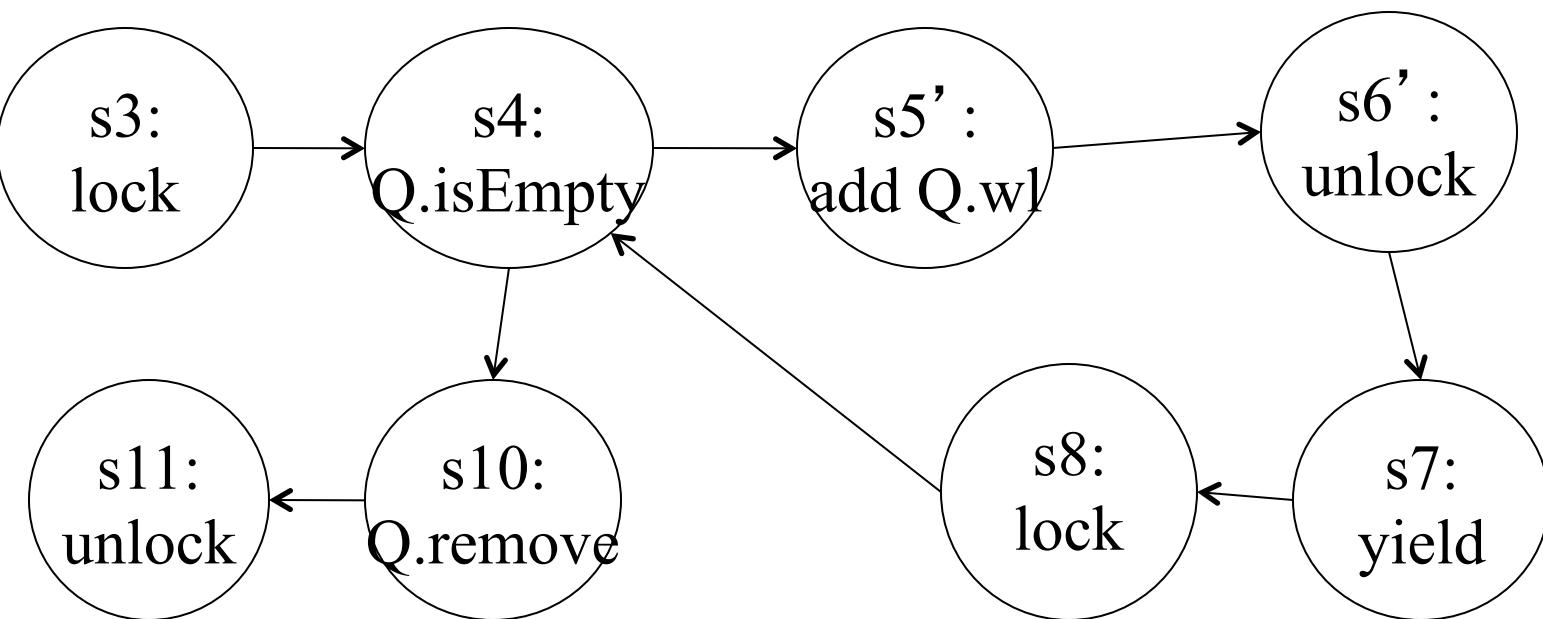
- main thread can always add to Q
- every connection in Q will be processed

□ Fairness

- For example, in some settings, a designer may want the threads to share load equally

Main thread can always add to Q

- Assume main is blocked
- Suppose Q is not empty, then each iteration removes one element from Q
- In finite number of iterations, all elements in Q are removed and all service threads unlock and block



Each connection in Q is processed

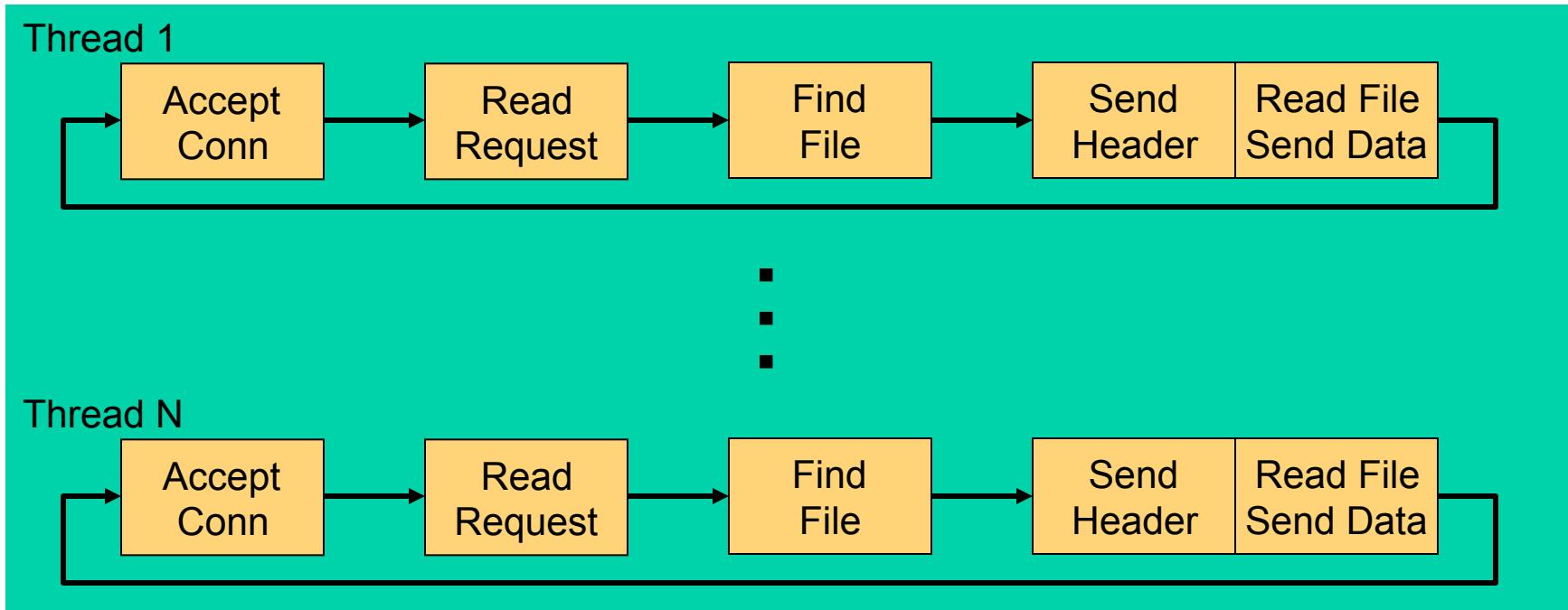
- Cannot be guaranteed unless
 - there is fairness in the thread scheduler, or
 - put a limit on Q size to block the main thread

Blocking Queues in Java

- Design Pattern for producer/consumer pattern with blocking
- Two handy implementations
 - LinkedBlockingQueue (FIFO, may be bounded)
 - ArrayBlockingQueue (FIFO, bounded)
 - (plus a couple more)

<https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html>

Summary: Using Threads

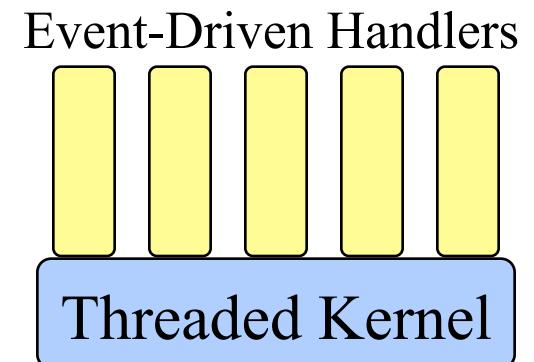


- Advantages
 - Intuitive (sequential) programming model
 - Shared address space simplifies optimizations
- Disadvantages
 - Overhead: thread stacks, synchronization
 - **Thread pool parameter (how many threads) difficult to tune**

Should You Use Threads?

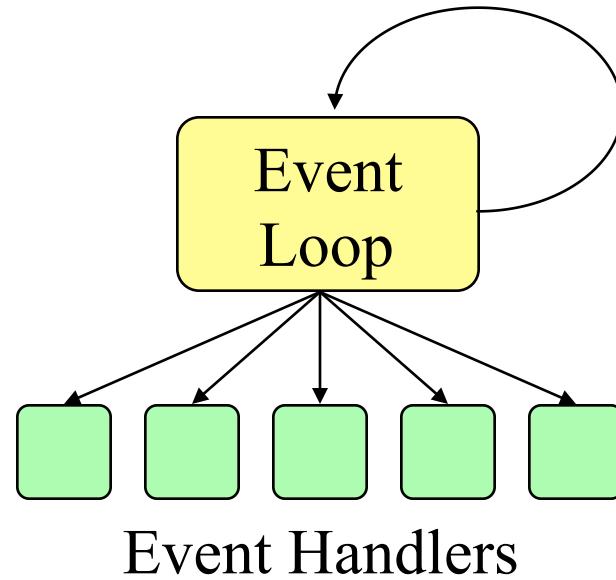
- ❑ Typically avoid threads for io
 - Use event-driven, not threads, for GUIs, distributed systems, low-end servers.

- ❑ Use threads where true CPU concurrency is needed.
 - Where threads needed, isolate usage in threaded application kernel: keep most of code single-threaded.



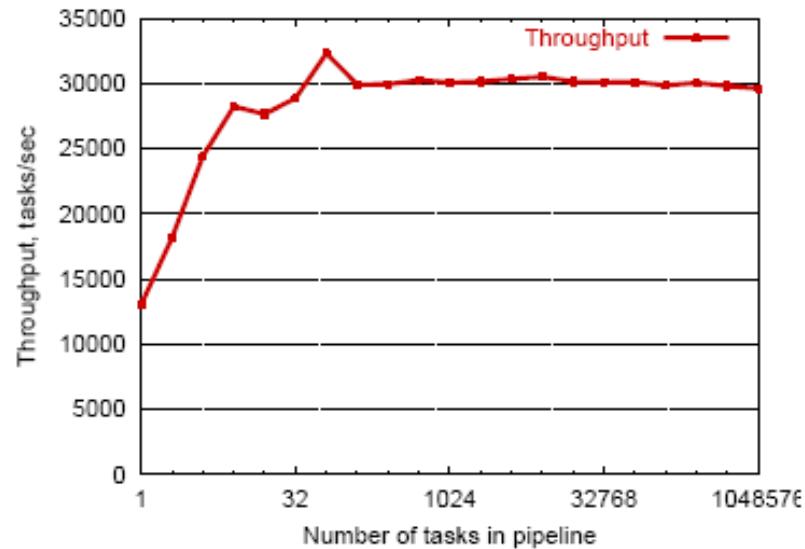
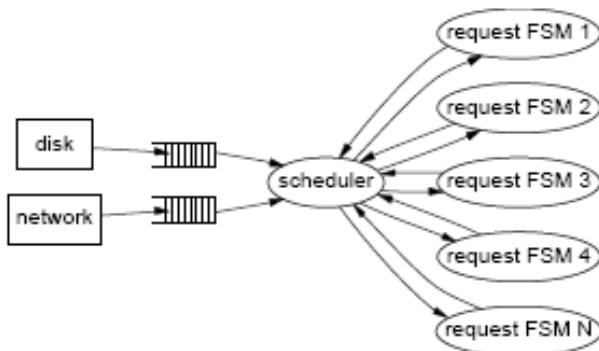
(Extreme) Event-Driven Programming

- One execution stream: no CPU concurrency
- A single-thread event loop issues commands, waits for events, invokes handlers (callbacks)
 - Handlers issue asynchronous (non-blocking) I/O
 - No preemption of event handlers (handlers generally short-lived)



[Ousterhout 1995]

Event-Driven Programming

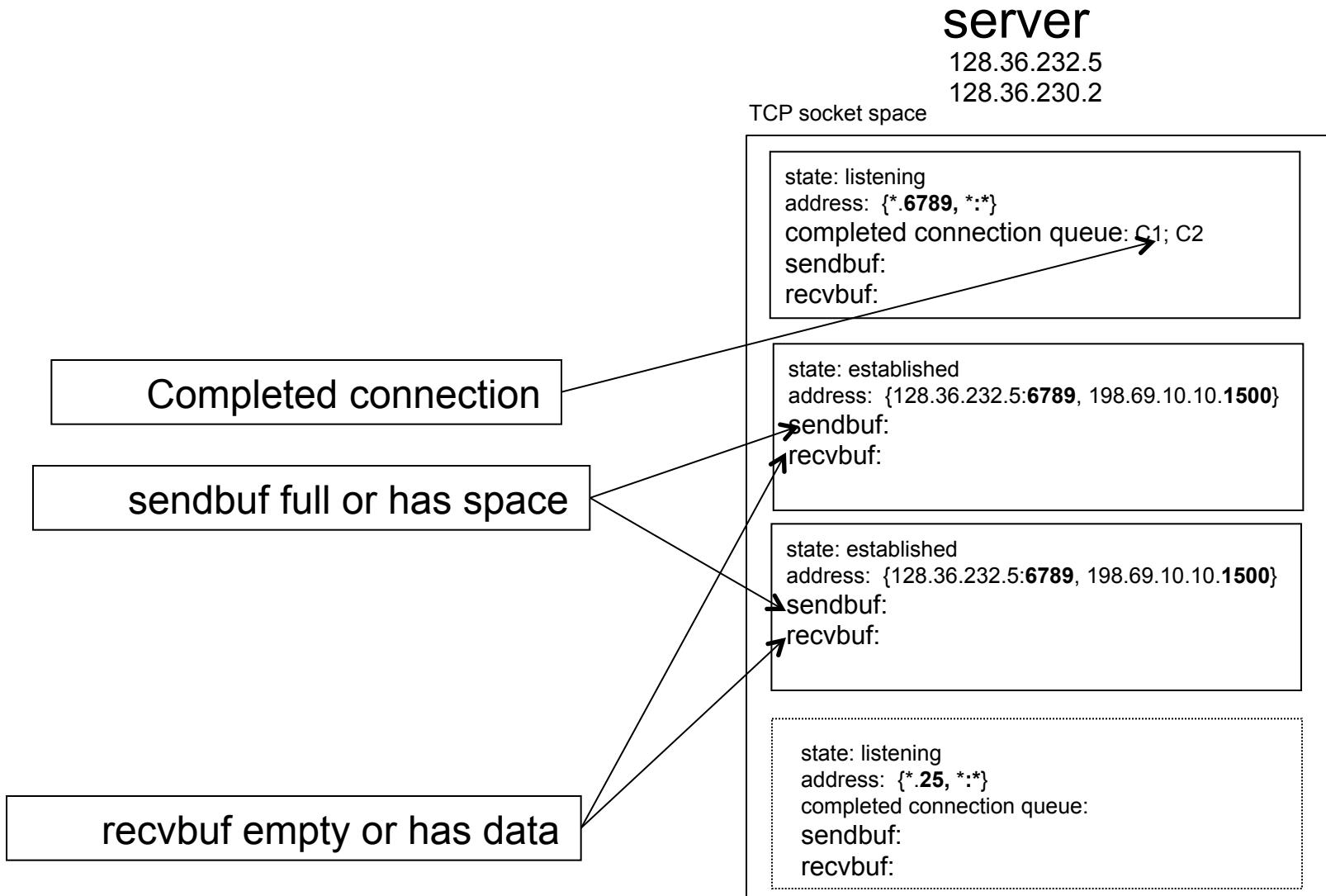


- Advantages
 - Single address space
 - No synchronization
 - Disadvantages
 - Program complexity
 - In practice, disk reads/page fault still block
 - Many examples: Click router, Flash web server, TP Monitors, NOX controller, Google Chrome (libevent), Dropbox (libevent),

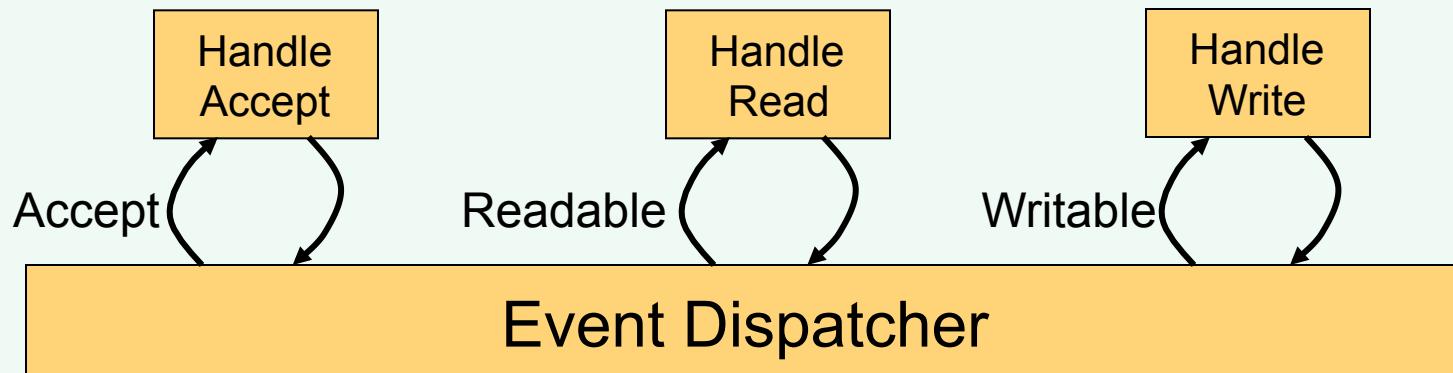
Async Server I/O Basis

- Modern operating systems, such as Windows, Mac and Linux, provide facilities for fast, scalable IO based on the use of **asynchronous initiation** (e.g., `aio_read`) and **notifications of ready IO operations** taking place in the operating system layers.
 - Windows: IO Completion Ports
 - Linux: `select`, `epoll` (2.6)
 - Mac/FreeBSD: `kqueue`
- An Async IO package (e.g., Java Nio, Boost ASOI, Netty) aims to make (**some of**) these facilities available to applications

Async I/O Example: Ready Operations



Async IO with OS Notification



- Software framework on top of OS notification
 - Register handlers with dispatcher on sources (e.g., which sockets) and events (e.g., acceptable, readable, writable) to monitor
 - Dispatcher asks OS to check if any ready source/event
 - Dispatcher calls the registered handler of each ready event/source

Dispatcher Structure

```
//clients register interests/handlers on events/sources
while (true)  {
    - ready events = select() /* or selectNow(), or
                                select(int timeout)
                                to check the
                                ready events from the
                                registered interest
                                events of sources */
    - foreach ready event {
        switch event type:
            accept: call accept handler
            readable: call read handler
            writable: call write handler
    }
}
```

Outline

- Admin and recap
- High performance servers
 - Thread
 - Per-request thread
 - Thread pool
 - Busy wait
 - Wait/notify
 - Asynchronous servers
 - Overview
 - Java async io

Async I/O in Java

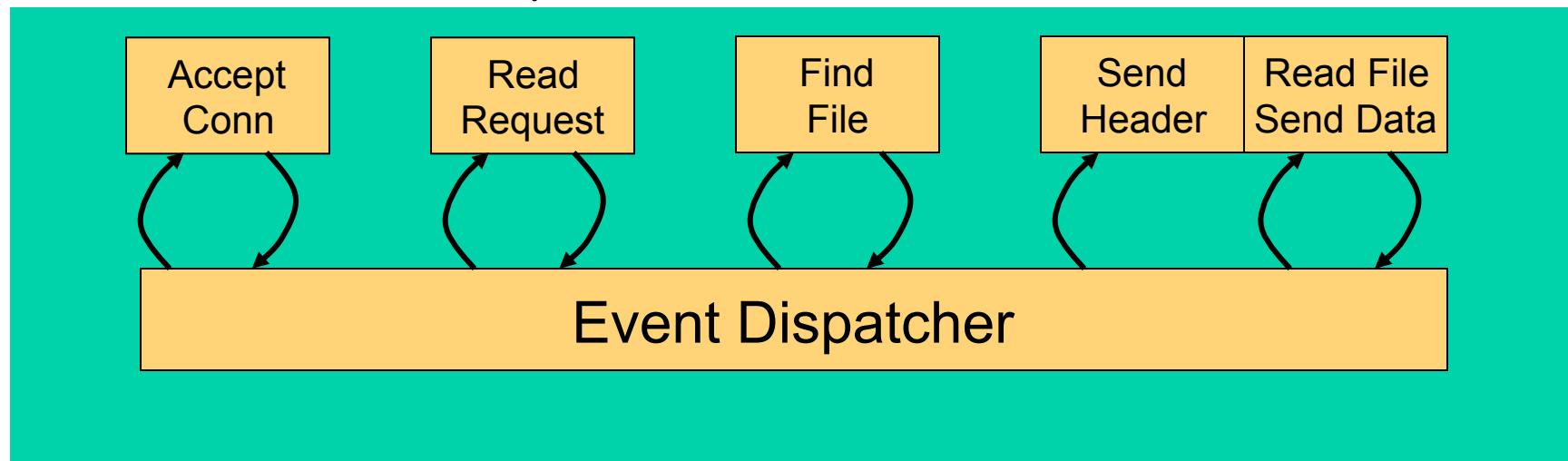
- Java AIO provides some platform-independent abstractions on top of OS notification mechanisms (e.g., select/epoll)
- A typical network server (or package) builds on top of AIO abstractions
 - Sources (channel)
 - Selector
 - Buffers

Async I/O in Java: Sources

- A source that can generate events in Java is called a `SelectableChannel` object:
 - Example `SelectableChannels`:
`DatagramChannel`, `ServerSocketChannel`,
`SocketChannel`, `Pipe.SinkChannel`,
`Pipe.SourceChannel`
 - **use** `configureBlocking(false)` **to make a channel non-blocking**
- Note: Java `SelectableChannel` **does not include file I/O**

Async I/O in Java: Selector

- ❑ An important class is the class `Selector`, which is a base of the multiplexer/dispatcher
- ❑ Constructor of `Selector` is protected; create by invoking the `open` method to get a selector (why?)



Selector and Registration

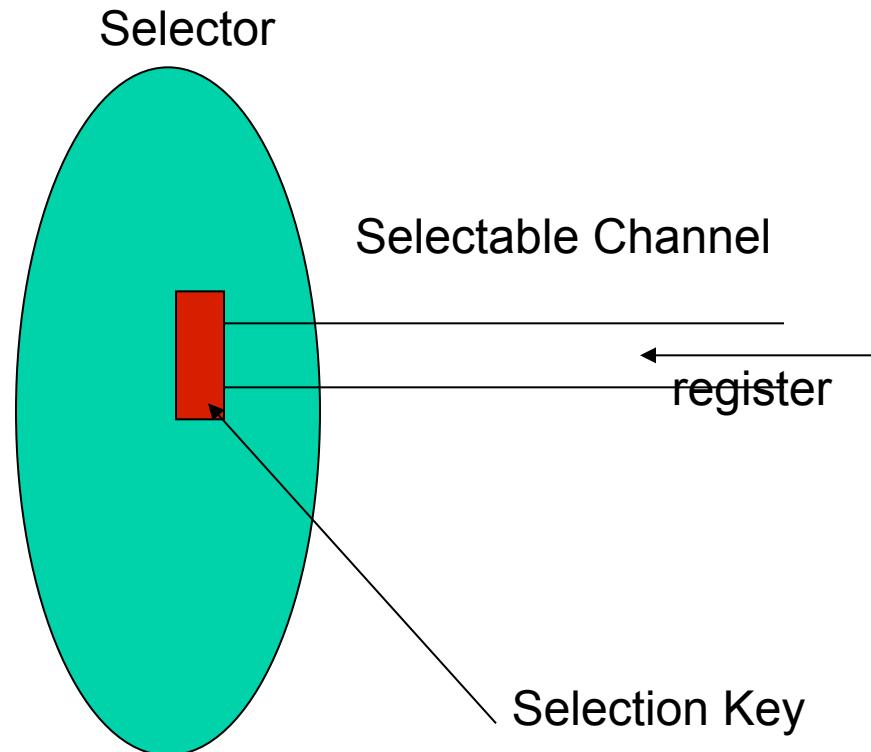
- ❑ A selectable channel registers events to be monitored with a selector with the register method
- ❑ The registration returns an object called a SelectionKey:

```
SelectionKey key =  
    channel.register(selector, ops);
```

Java Async I/O Structure

- ❑ A SelectionKey object stores:

- **interest set**: events to check:
`key.interestOps(ops)`
 - **ready set**: after calling `select`, it contains the events that are ready, e.g.
`key.isReadable()`
 - **an attachment** that you can store anything you want
`key.attach(myObj)`



Checking Events

- A program calls `select` (or `selectNow()`, or `select(int timeout)`) to check for ready events from the registered SelectableChannels
 - Ready events are called the selected key set

```
selector.select();
Set readyKeys = selector.selectedKeys();
```
- The program iterates over the selected key set to process all ready events

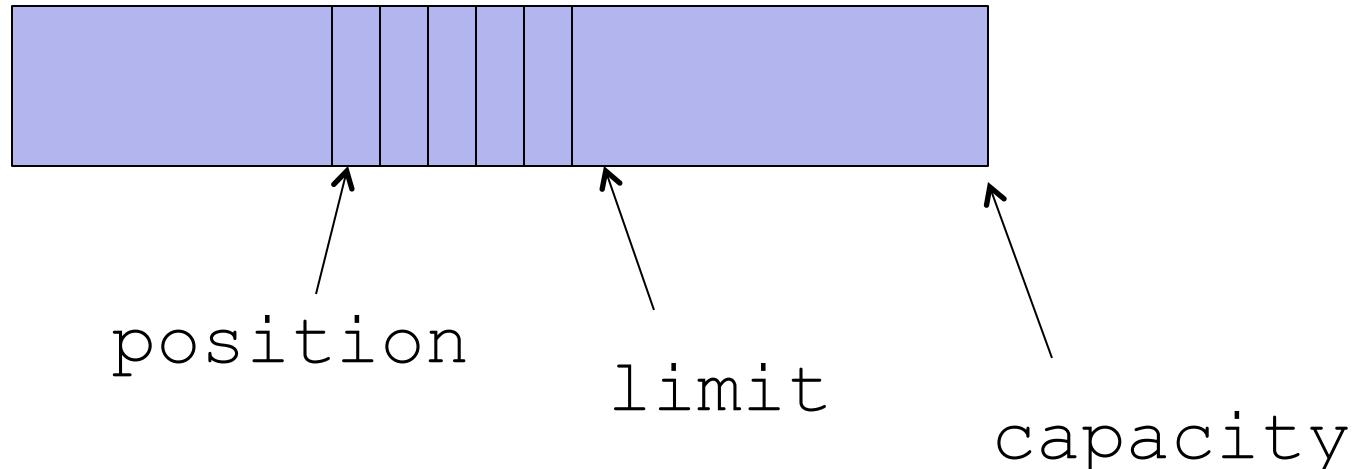
Dispatcher Structure

```
while (true)  {  
    - selector.select()  
    - Set readyKeys = selector.selectedKeys();  
  
    - foreach key in readyKeys {  
        switch event type of key:  
            accept: call accept handler  
            readable: call read handler  
            writable: call write handler  
    }  
}
```

Async I/O in Java: ByteBuffer

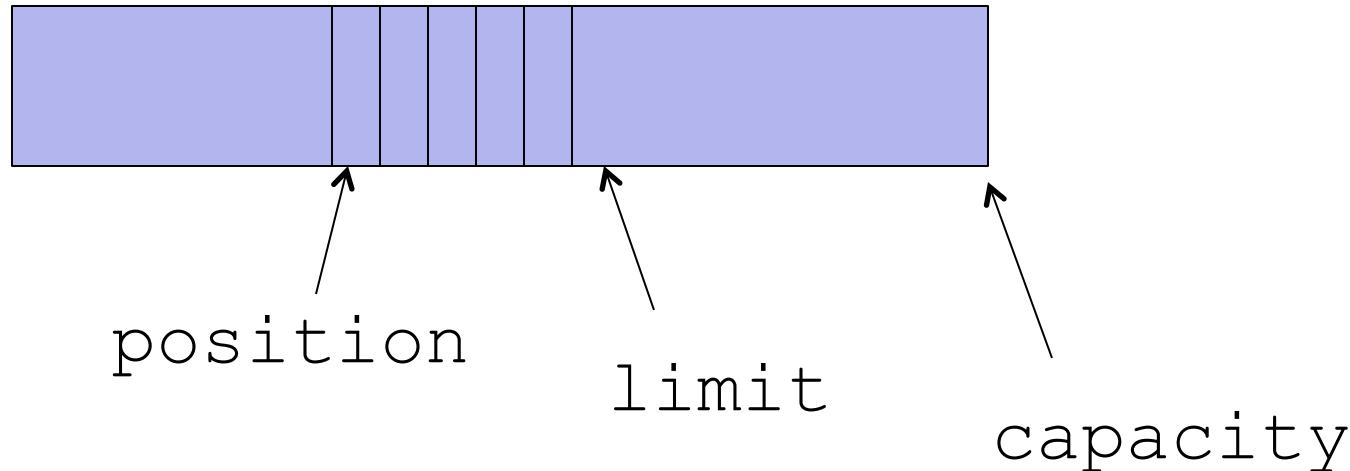
- Java SelectableChannels typically use ByteBuffer for read and write
 - channel.read(byteBuffer);
 - channel.write(byteBuffer);
- ByteBuffer is a powerful class that can be used for both read and write
- It is derived from the class Buffer
- You can use it either as an absolute indexed or relatively indexed buffer

Buffer (relative index)



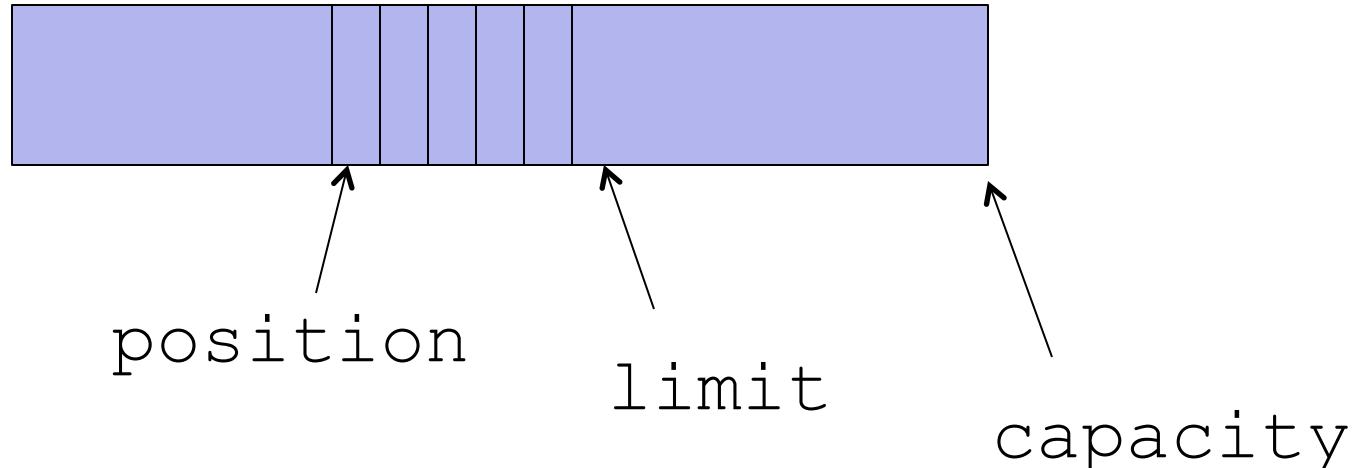
- Each Buffer has three numbers: position, limit, and capacity
 - Invariant: $0 \leq position \leq limit \leq capacity$
- `Buffer.clear()`: $position = 0$; $limit=capacity$

channel.read(Buffer)



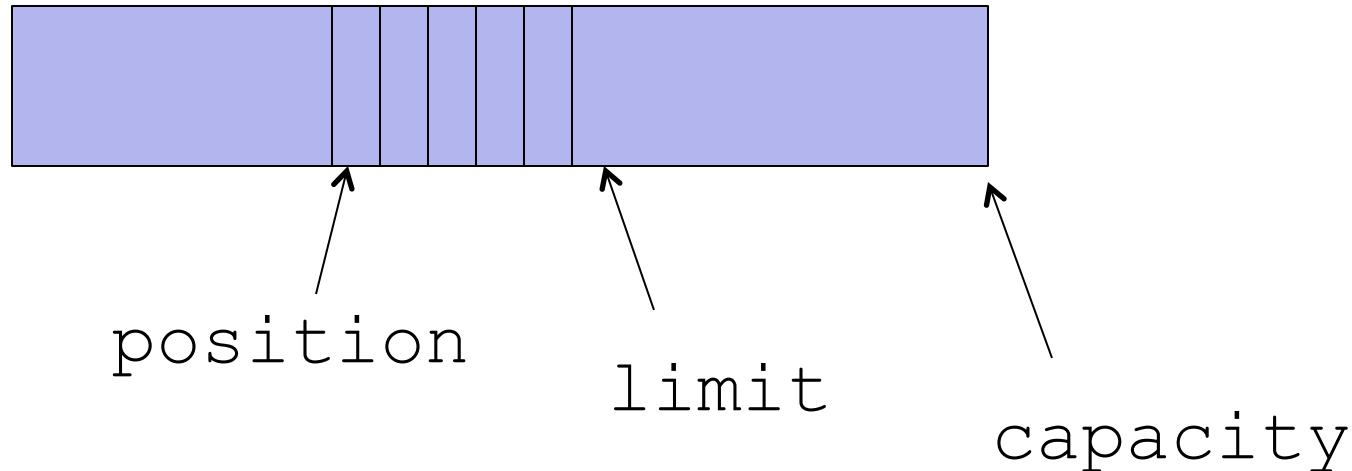
- ❑ Put data into Buffer, starting at position, **not to reach** limit

channel.write(Buffer)



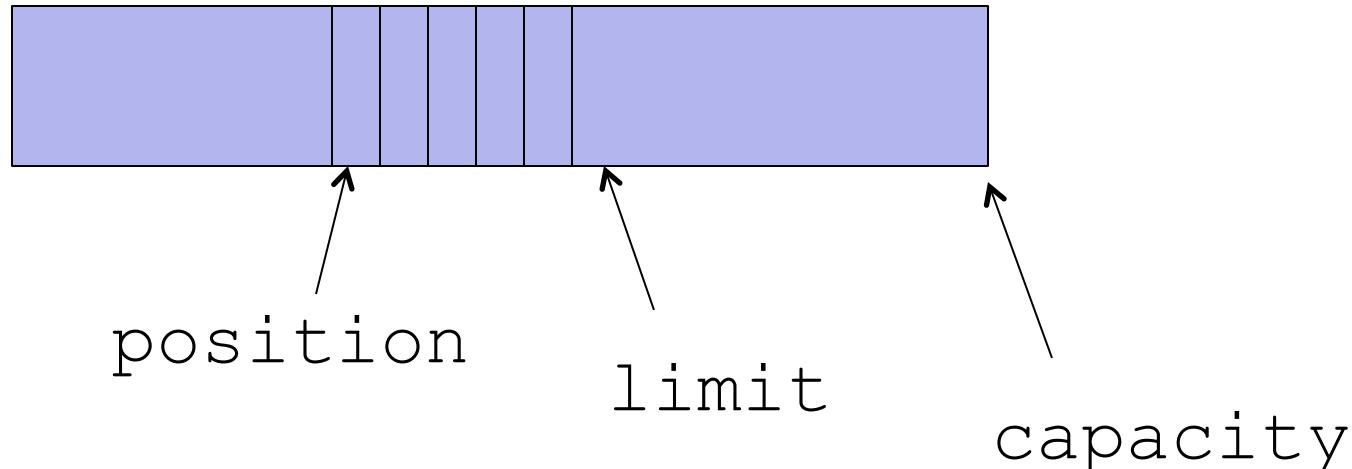
- ❑ Move data from Buffer to channel, starting at position, not to reach limit

Buffer.flip()



- ❑ `Buffer.flip(): limit=position; position=0`
- ❑ Why flip: used to switch from preparing data to output, e.g.,
 - `buf.put(header); // add header data to buf`
 - `in.read(buf); // read in data and add to buf`
 - `buf.flip(); // prepare for write`
 - `out.write(buf);`

Buffer.compact()



- Move [position , limit) to 0
- Set position to limit-position, limit to capacity

```
buf.clear(); // Prepare buffer for use
for (;;) {
    if (in.read(buf) < 0 && !buf.hasRemaining())
        break; // No more bytes to transfer
    buf.flip();
    out.write(buf);
    buf.compact(); // In case of partial write
}
```

Example

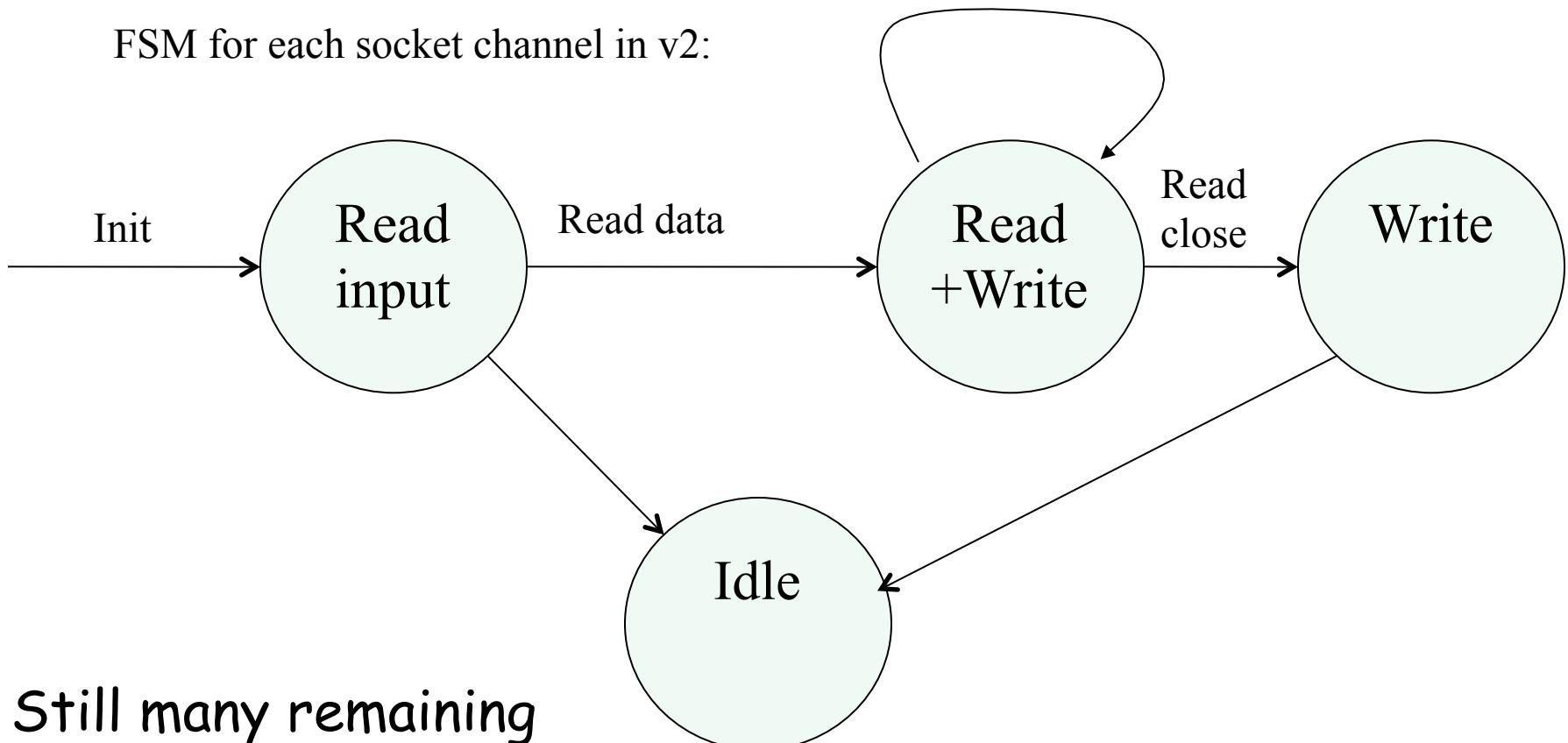
- See AsyncEchoServer/v1-2/
EchoServer.java

Problems of Async Echo Server v1

- Empty write: Callback to `handleWrite()` is unnecessary when nothing to write
 - Imagine empty write with 10,000 sockets
 - Solution: initially read only, later allow write
- `handleRead()` still reads after the client closes
 - Solution: after reading end of stream (read returns -1), deregister read interest for the channel

(Partial) Finite State Machine (FSM)

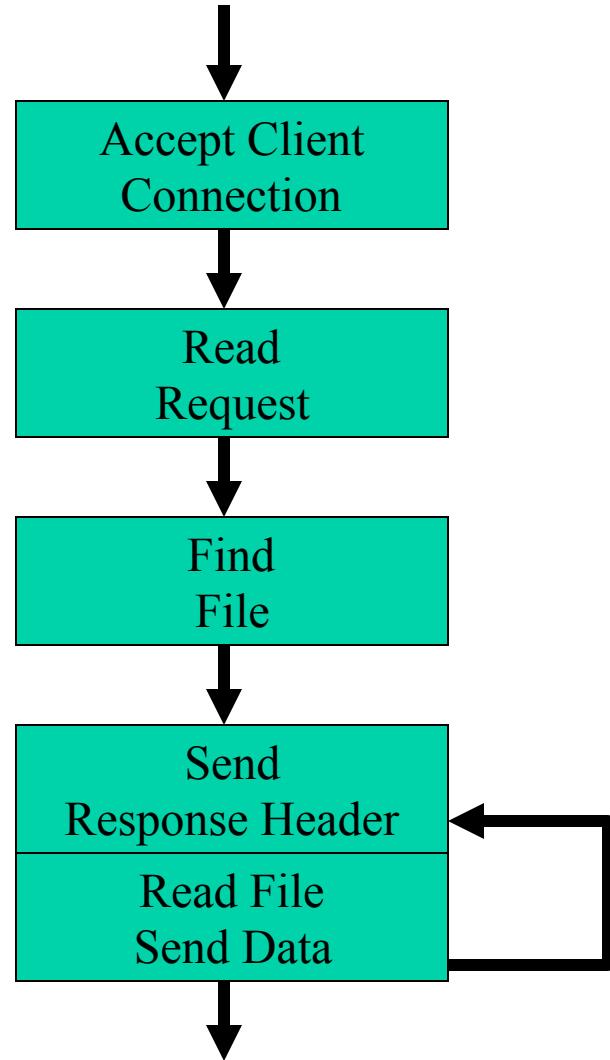
FSM for each socket channel in v2:



Still many remaining issues such as idle instead of close

Finite-State Machine and Thread

- Why no need to introduce FSM for a thread version?



Network Applications:
High-performance Server Design:
Async Servers/Operational Analysis

Y. Richard Yang

<http://zoo.cs.yale.edu/classes/cs433/>

2/24/2016

Admin

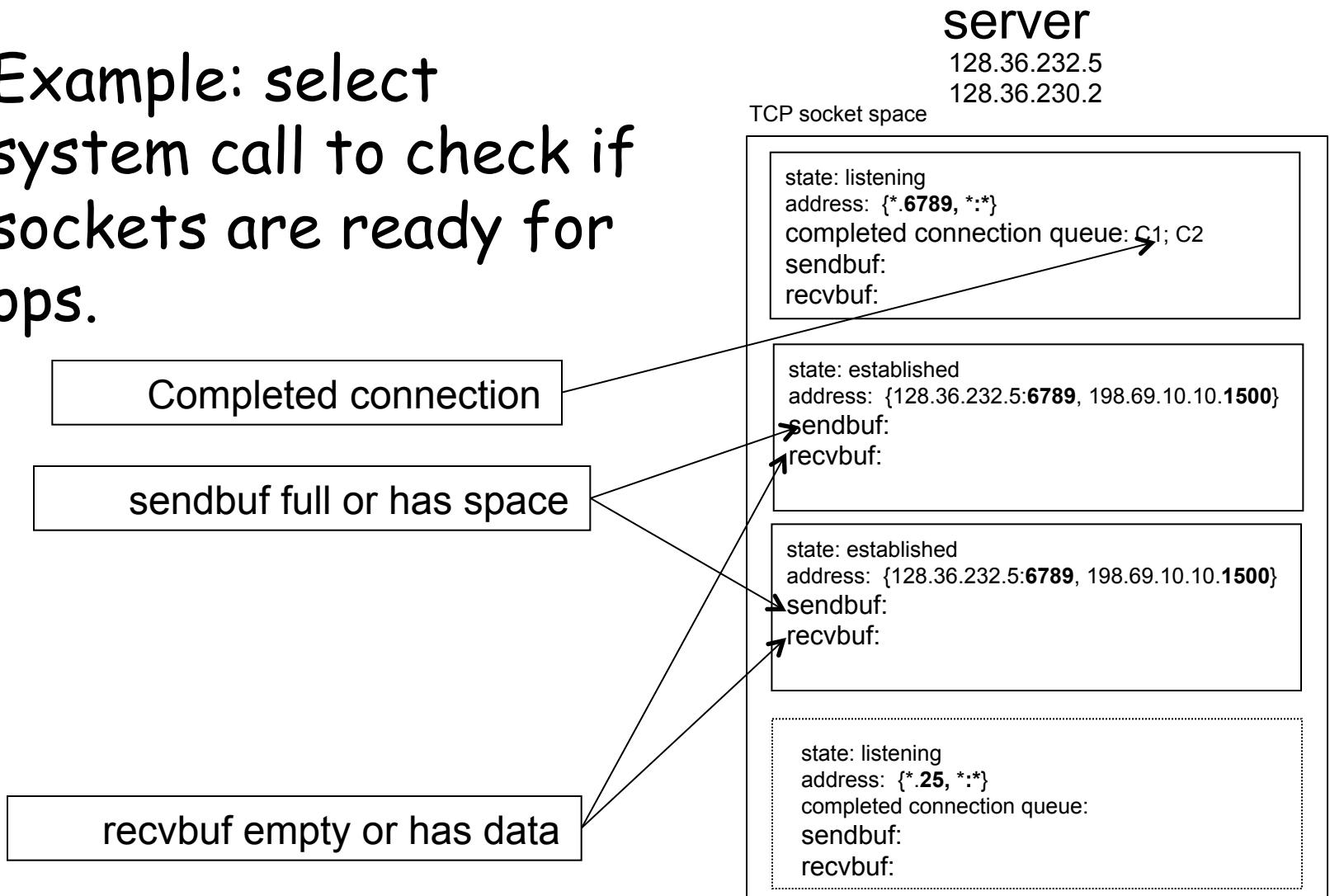
- Assignment three posted.
- Decisions
 - Projects or exam 2
 - Date for exam 1

Recap: Async Network Server

- Basic ideas: non-blocking operations
 1. peek system state (using a select mechanism) to issue only ready operations
 2. asynchronous initiation (e.g., aio_read) and completion notification (callback)

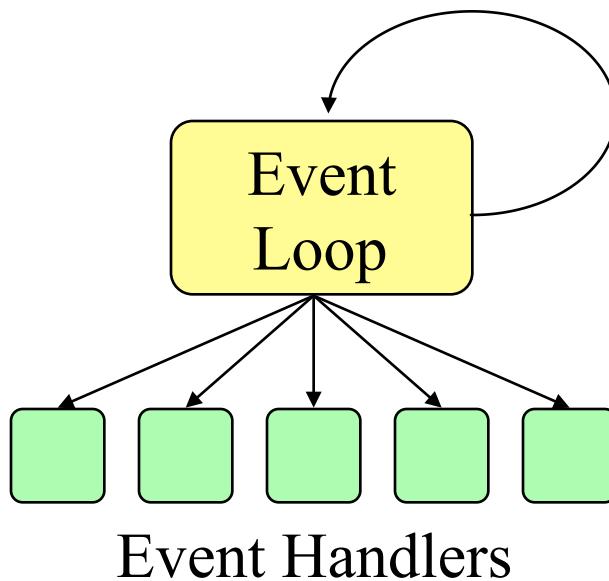
Recap: Async Network Server using Select

- ❑ Example: select system call to check if sockets are ready for ops.



Recap: Async Network Server using Select

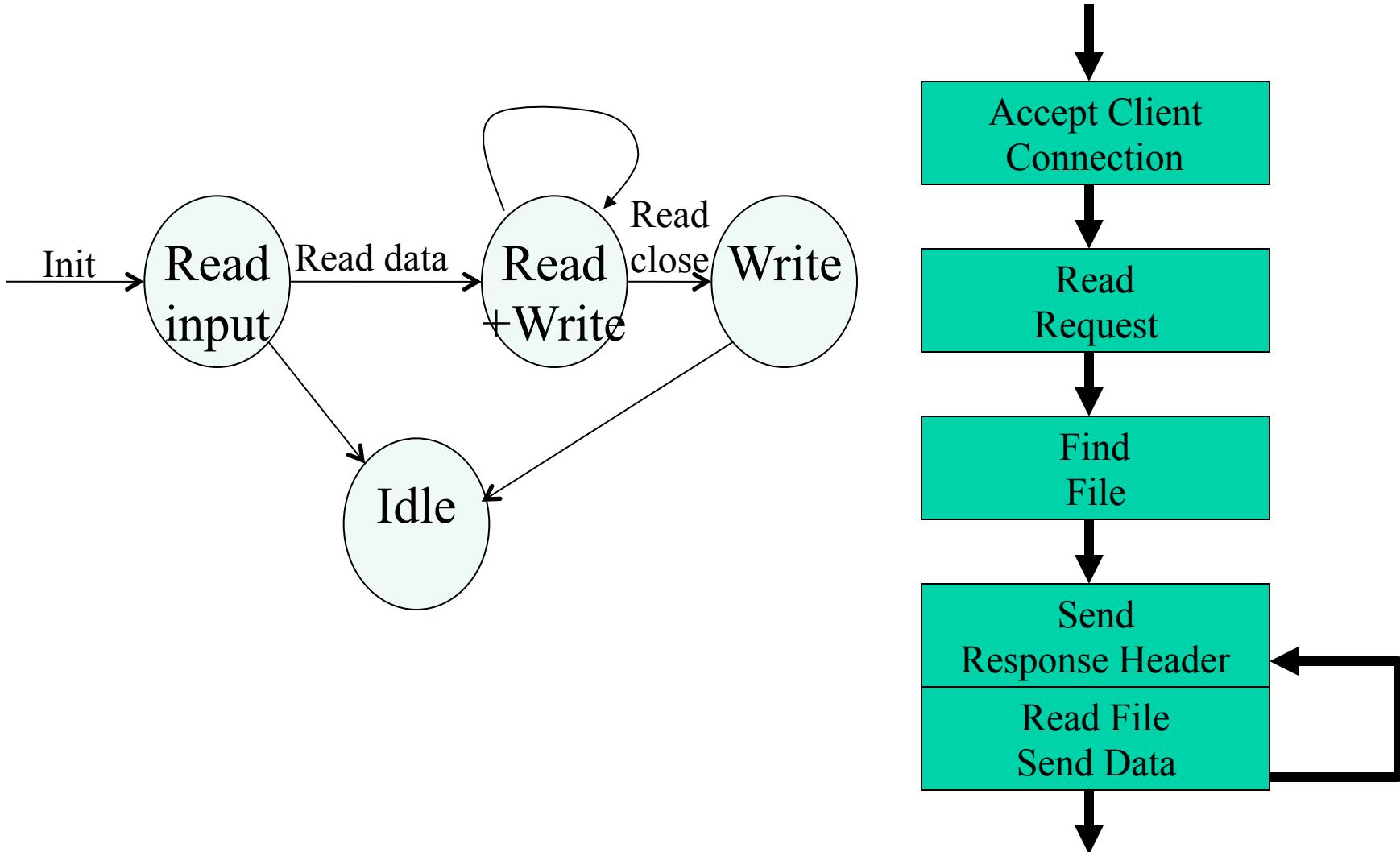
- ❑ A event loop issues commands, waits for events, invokes handlers (callbacks)



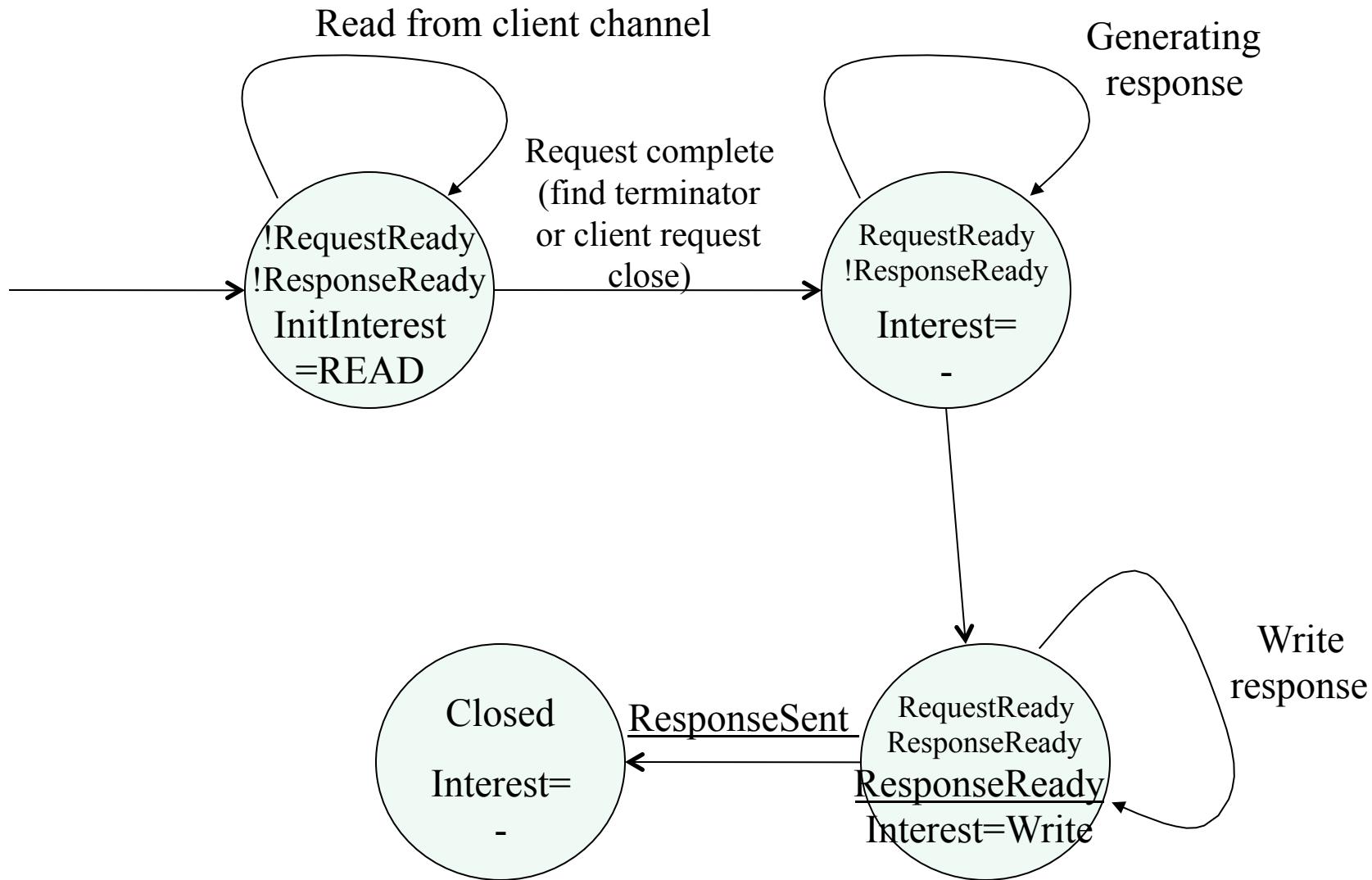
```
// clients register interests/
handlers on events/sources
while (true)  {
    - ready events = select()
    /* or selectNow(),
       or select(int timeout) */

    - foreach ready event {
        switch event type:
        accept: call accept handler
        readable: call read handler
        writable: call write handler
    }
}
```

Recap: Need to Manage Finite State Machine



Another Finite State Machine



Finite State Machine Design

- EchoServerV2:
 - Mixed read and write
- Example last slide: staged
 - First read request and then write response
- Choice depends on protocol and tolerance of complexity, e.g.,
 - HTTP/1.0 channel may use staged
 - HTTP/1.1/2/Chat channel may use mixed

Toward More General Server Framework

- Our example EchoServer is for a specific protocol
- A general async/io programming framework tries to introduce structure to allow substantial reuse
 - Async io programming framework is among the more complex software systems
 - We will see one simple example, using EchoServer as a basis

A More Extensible Dispatcher Design

- Fixed accept/read/write functions are not general design
- Requirement: map from key to handler
- A solution: Using attachment of each channel
 - Attaching a `ByteBuffer` to each channel is a narrow design for simple echo servers
 - A more general design can use the attachment to store a callback that indicates not only data (state) but also the handler (function)

A More Extensible Dispatcher Design

- Attachment stores generic event handler
 - Define interfaces
 - IAcceptHandler and
 - IReadWriteHandler
 - Retrieve handlers at run time

```
if (key.isAcceptable()) { // a new connection is ready
    IAcceptHandler aH = (IAcceptHandler) key.attachment();
    aH.handleAccept(key);
}

if (key.isReadable() || key.isWritable()) {
    IReadWriteHandler rwH = IReadWriteHandler)key.attachment();
    if (key.isReadable()) rwH.handleRead(key);
    if (key.isWritable()) rwH.handleWrite(key);
}
```

Handler Design: Acceptor

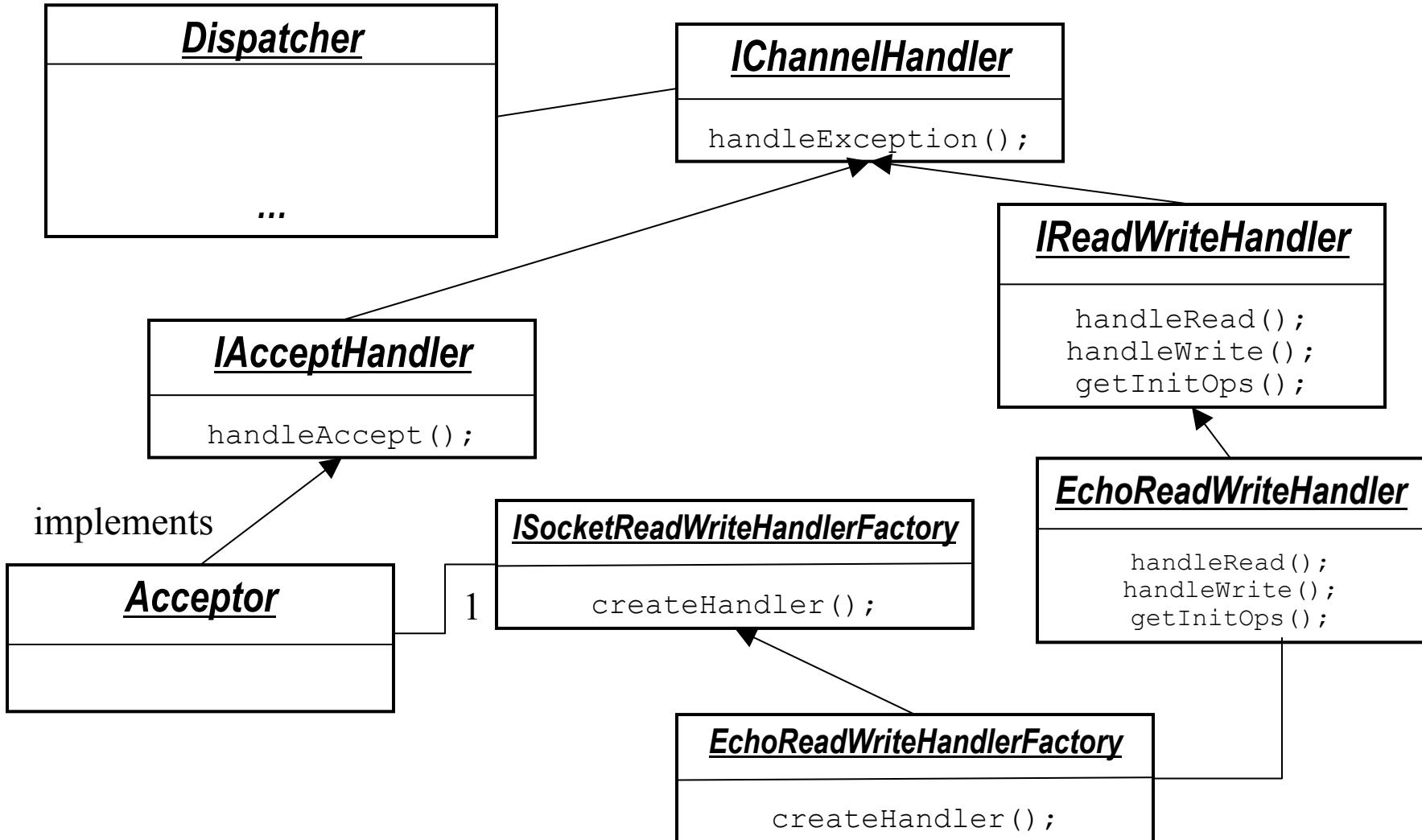
- What should an accept handler object know?
 - ServerSocketChannel (so that it can call accept)
 - Can be derived from SelectionKey in the call back
 - Selector (so that it can register new connections)
 - Can be derived from SelectionKey in the call back
 - What ReadWrite object to create (different protocols may use different ones)?
 - Pass a Factory object: SocketReadWriteHandlerFactory

Handler Design:

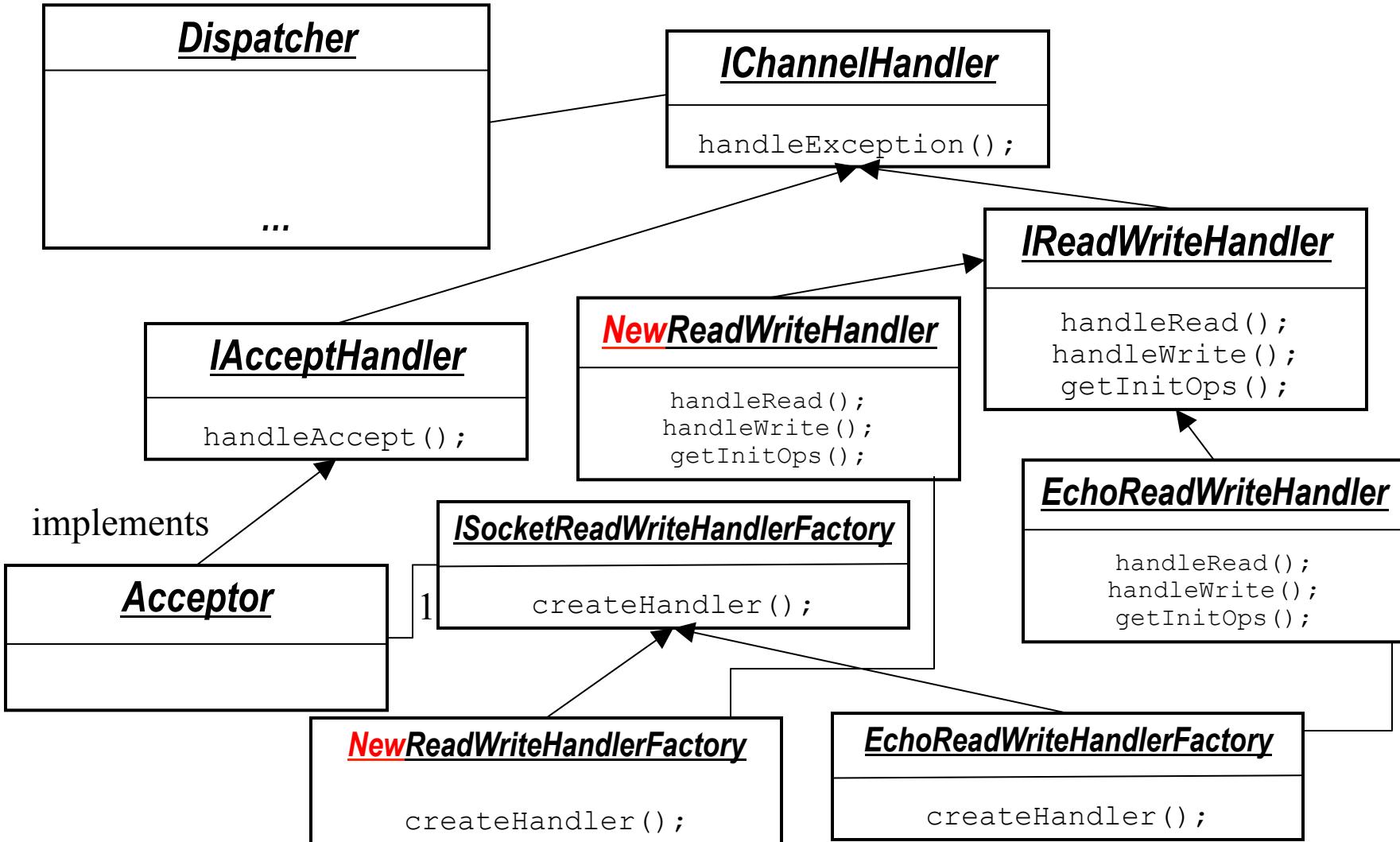
ReadWriteHandler

- What should a ReadWrite handler object know?
 - SocketChannel (so that it can read/write data)
 - Can be derived from SelectionKey in the call back
 - Selector (so that it can change state)
 - Can be derived from SelectionKey in the call back

Class Diagram of SimpleNAIO



Class Diagram of SimpleNAIO



SimpleNAIO

- See AsyncEchoServer/v3/*.java

Discussion on SimpleNAIO

In our current implementation (Server.java)

1. Create dispatcher
2. Create server socket channel and listener
3. Register server socket channel to dispatcher
4. Start dispatcher thread

Can we switch 3 and 4?

Extending SimpleNAIO

- A production network server often closes a connection if it does not receive a complete request in TIMEOUT
 - One way to implement time out is that
 - the read handler registers a timeout event with a timeout watcher thread with a call back
 - the watcher thread invokes the call back upon TIMEOUT
 - the callback closes the connection
- Any problem?

Extending Dispatcher Interface

- Interacting from another thread to the dispatcher thread can be tricky
- Typical solution: `async command queue`

```
while (true) {  
    - process async. command queue  
    - ready events = select (or selectNow(), or  
      select(int timeout)) to check for ready events  
      from the registered interest events of  
      SelectableChannels  
  
    - foreach ready event  
      call handler  
}
```

Question

- How may you implement the `async` command queue to the selector thread?

```
public void invokeLater(Runnable run) {  
    synchronized (pendingInvocations) {  
        pendingInvocations.add(run);  
    }  
    selector.wakeup();  
}
```

Question

- ❑ What if another thread wants to wait until a command is finished by the dispatcher thread?

```
public void invokeAndWait(final Runnable task)
    throws InterruptedException
{
    if (Thread.currentThread() == selectorThread) {
        // We are in the selector's thread. No need to schedule
        // execution
        task.run();
    } else {
        // Used to deliver the notification that the task is executed
        final Object latch = new Object();
        synchronized (latch) {
            // Uses the invokeLater method with a newly created task
            this.invokeLater(new Runnable() {
                public void run() {
                    task.run();
                    // Notifies
                    synchronized(latch) { latch.notify(); }
                }
            });
            // Wait for the task to complete.
            latch.wait();
        }
        // Ok, we are done, the task was executed. Proceed.
    }
}
```

Recap: Async Network Server

- Basic idea: non-blocking operations
 1. peek system state (`select`) to issue only ready operations
 2. asynchronous initiation (e.g., `aio_read`) and completion notification (callback)

Alternative Design: Asynchronous Channel using Future/Listener

- Java 7 introduces
 - ASynchronousServerSocketChannel and ASynchronousSocketChannel beyond ServerSocketChannel and SocketChannel
 - accept, connect, read, write return Futures or have a callback. Selectors are not used

<https://docs.oracle.com/javase/7/docs/api/java/nio/channels/AsynchronousServerSocketChannel.html>

<https://docs.oracle.com/javase/7/docs/api/java/nio/channels/AsynchronousSocketChannel.html>

```
SocketAddress address
    = new InetSocketAddress(args[0], port);
AsynchronousSocketChannel client
    = AsynchronousSocketChannel.open();
Future<Void> connected
    = client.connect(address);

ByteBuffer buffer = ByteBuffer.allocate(100);

// wait for the connection to finish
connected.get();

// read from the connection
Future<Integer> future = client.read(buffer);

// do other things...

// wait for the read to finish...
future.get();

// flip and drain the buffer
buffer.flip();
WritableByteChannel out
    = Channels.newChannel(System.out);
out.write(buffer);
```

```
class LineHandler implements
CompletionHandler<Integer, ByteBuffer> {

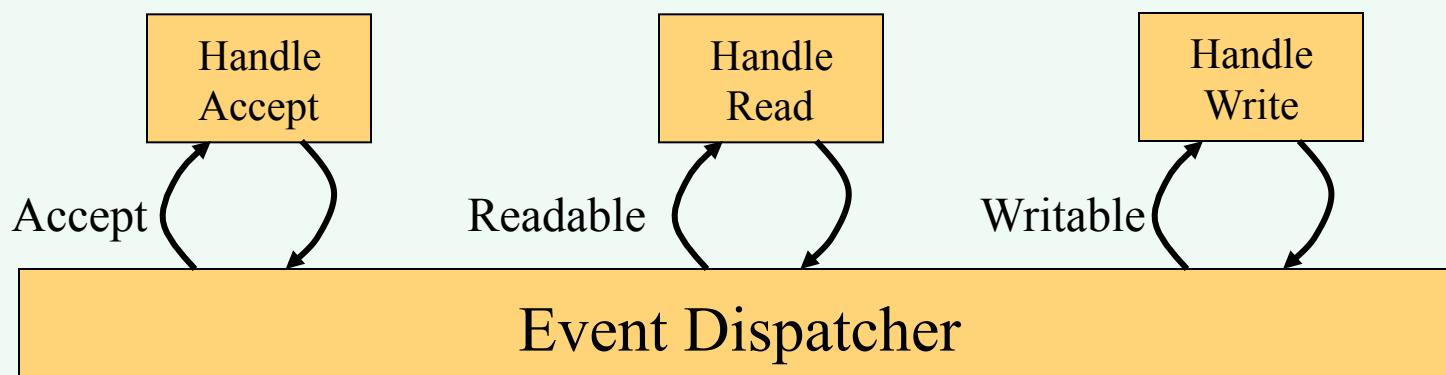
    @Override
    public void completed(Integer result, ByteBuffer buffer)
    {
        buffer.flip();
        WritableByteChannel out
            = Channels.newChannel(System.out);
        try {
            out.write(buffer);
        } catch (IOException ex) {
            System.err.println(ex);
        }
    }

    @Override
    public void failed(Throwable ex,
                      ByteBuffer attachment) {
        System.err.println(ex.getMessage());
    }
}
```

```
ByteBuffer buffer = ByteBuffer.allocate(100);
CompletionHandler<Integer, ByteBuffer>
    handler = new LineHandler();
channel.read(buffer, buffer, handler);
```

Extending FSM

- ❑ In addition to management threads, a system may still need multiple threads for performance (why?)
 - FSM code can never block, but page faults, file io, garbage collection may still force blocking
 - CPU may become the bottleneck and there maybe multiple cores supporting multiple threads (typically 2 n threads)



Summary: Architecture

- Architectures
 - Multi threads
 - Asynchronous
 - Hybrid

- Assigned reading: SEDA

Problems of Event-Driven Server

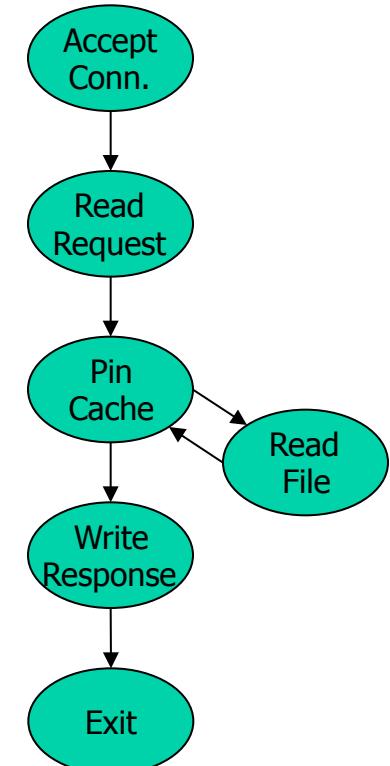
- Obscure control flow for programmers and tools
- Difficult to engineer, modularize, and tune
- Difficult for performance/failure isolation between FSMs

Another view

- Events obscure control flow
 - For programmers and tools

Threads	Events
<pre>thread_main(int sock) { struct session s; accept_conn(sock, &s); read_request(&s); pin_cache(&s); write_response(&s); unpin(&s); } pin_cache(struct session *s) { pin(&s); if(!in_cache(s)) read_file(&s); }</pre>	<pre>AcceptHandler(event e) { struct session *s = new_session(e); RequestHandler.enqueue(s); } RequestHandler(struct session *s) { ...; CacheHandler.enqueue(s); } CacheHandler(struct session *s) { pin(s); if(!in_cache(s)) ReadFileHandler.enqueue(s); else ResponseHandler.enqueue(s); } ... ExitHandler(struct session *s) { ...; unpin(&s); free_session(s); }</pre>

Web Server

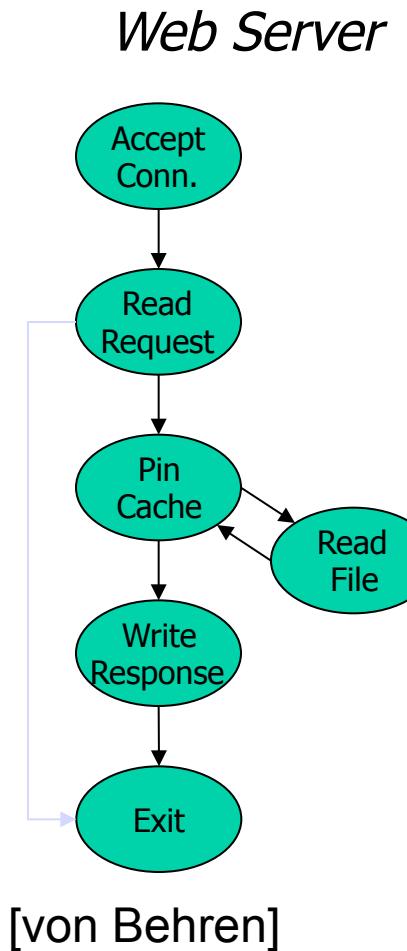


[von Behren]

State Management

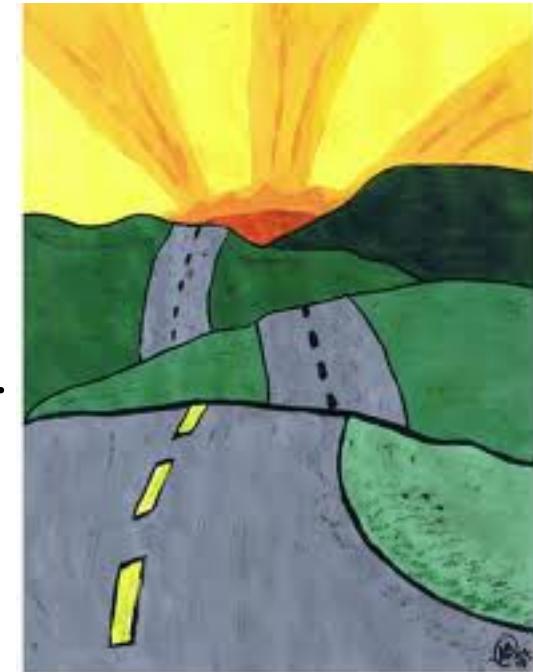
- Events require manual state management
- Hard to know when to free
 - Use GC or risk bugs

Threads	Events
<pre>thread_main(int sock) { struct session s; accept_conn(sock, &s); if(!read_request(&s)) return; pin_cache(&s); write_response(&s); unpin(&s); }</pre>	<pre>CacheHandler(struct session *s) { pin(s); if(!in_cache(s)) ReadFileHandler.enqueue(s); else ResponseHandler.enqueue(s); } RequestHandler(struct session *s) { ...; if(error) return; CacheHandler.enqueue(s); } ... ExitHandler(struct session *s) { ...; unpin(&s); free_session(s); } AcceptHandler(event e) { struct session *s = new_session(e); RequestHandler.enqueue(s); }</pre>
<pre>pin_cache(struct session *s) { pin(&s); if(!in_cache(&s)) read_file(&s); }</pre>	



Summary: The High-Performance Network Servers Journey

- ❑ Avoid blocking (so that we can reach bottleneck throughput)
 - Introduce threads
- ❑ Limit unlimited thread overhead
 - Thread pool, async io
- ❑ Coordinating data access
 - synchronization (lock, synchronized)
- ❑ Coordinating behavior: avoid busy-wait
 - Wait/notify; select FSM, Future/Listener
- ❑ Extensibility/robustness
 - Language support/Design for interfaces



Beyond Class: Design Patterns

- We have seen Java as an example
- C++ and C# can be quite similar. For C++ and general design patterns:
 - <http://www.cs.wustl.edu/~schmidt/PDF/OOCP-tutorial4.pdf>
 - <http://www.stal.de/Downloads/ADC2004/pra03.pdf>

Some Questions

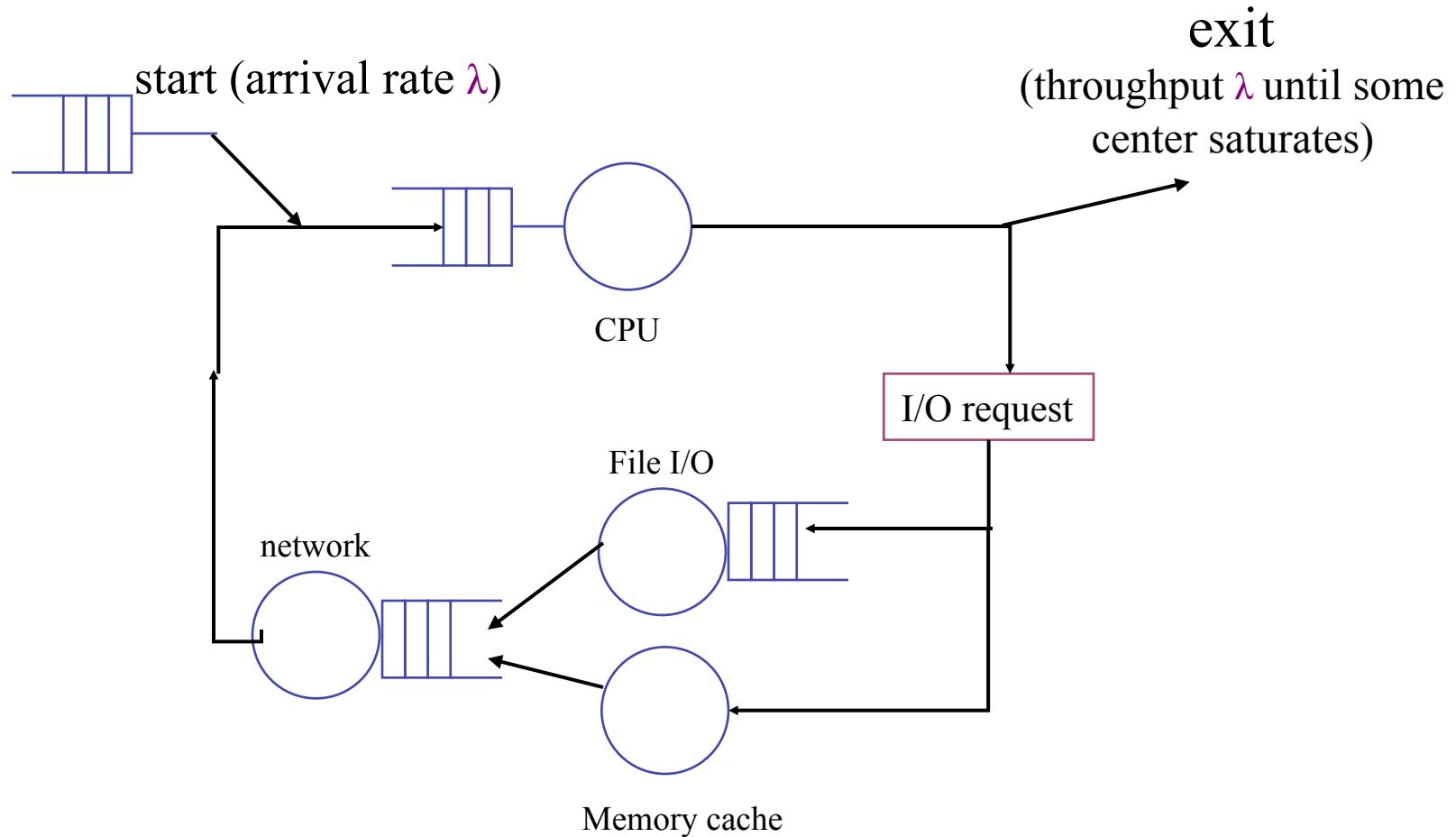
- When is CPU the bottleneck for scalability?
 - So that we need to add helpers
- How do we know that we are reaching the limit of scalability of a single machine?
- These questions drive network server architecture design

Operational Analysis

- Relationships that do not require any assumptions about the distribution of service times or inter-arrival times.
- Identified originally by Buzen (1976) and later extended by Denning and Buzen (1978).

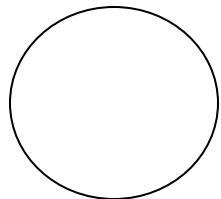
- We touch only some techniques/results
 - In particular, bottleneck analysis
- More details see linked reading

Under the Hood (An example FSM)



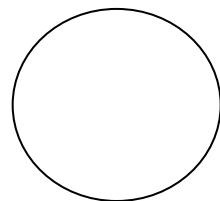
Operational Analysis: Resource Demand of a Request

CPU



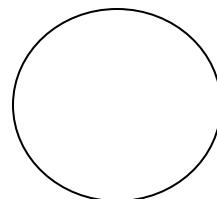
V_{CPU} visits for S_{CPU} units of resource time per visit

Network



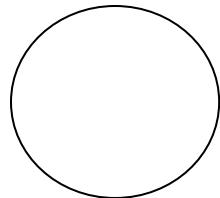
V_{Net} visits for S_{Net} units of resource time per visit

Disk



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

Memory



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

Operational Quantities

- T : observation interval A_i : # arrivals to device i
- B_i : busy time of device i C_i : # completions at device i
- $i = 0$ denotes system

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

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

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

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

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:

$$Q = \lambda R = X R$$

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

$$R = S + Q S = S + X R S$$

- According to utilization law, $U = X S$

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

Network Applications: Operational Analysis; Load Balancing among Multiple Servers

Y. Richard Yang

<http://zoo.cs.yale.edu/classes/cs433/>

2/29/2016

Admin

- Assignment three status and questions.

Recap: Operational Analysis

- Objective: derive relationships among (measured) operational performance metrics

- T : observation interval
- B_i : busy time of device i
- $i = 0$ denotes system
- A_i : # arrivals to device i
- C_i : # completions at device i

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

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

$$\text{Utilization } U_i = \frac{B_i}{T} \quad \text{Mean service time } S_i = \frac{B_i}{C_i}$$

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

Forced Flow Law

- Assume each request visits device i V_i times

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

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

$$= V_i X$$

Bottleneck Device

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

$$= V_i X S_i$$

$$= X V_i S_i$$

- Define $D_i = V_i S_i$ as the total demand of a request on device i
- The device with the highest D_i has the highest utilization, and thus is called the **bottleneck**

Bottleneck vs System Throughput

$$\text{Utilization } U_i = X V_i S_i \leq 1$$

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

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:

- $D_{CPU} = 10 \text{ ms}$ (e.g. 100 requests/s)

□ Network:

- $D_{Net} = 1 \text{ Mbytes} / 100 \text{ Mbps} = 80 \text{ ms}$ (e.g., 12.5 requests/s)

□ Disk I/O:

- $D_{disk} = 0.5 * 1 \text{ ms} * 1M/8K = 62.5 \text{ ms}$
(e.g. = 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

Interactive Response Time Law

□ System setup

- Closed system with N users
- Each user sends in a request, after response, think time, and then sends next request
- Notation
 - Z = user think-time, R = Response time
- The total cycle time of a user request is $R+Z$

In duration T , #requests generated by each user: $T/(R+Z)$ requests

Interactive Response Time Law

- If N users and flow balanced:

System Throughput $X = \text{Total\# req./T}$

$$= \frac{N \frac{T}{R+Z}}{T}$$

$$= \frac{N}{R+Z}$$

$$R = \frac{N}{X} - Z$$

Bottleneck Analysis

$$X(N) \leq \min\left\{\frac{1}{D_{\max}}, \frac{N}{D+Z}\right\}$$

$$R(N) \geq \max\{D, ND_{\max} - Z\}$$

- Here D is the sum of D_i

Proof

$$X(N) \leq \min\left\{\frac{1}{D_{\max}}, \frac{N}{D+Z}\right\}$$

$$R(N) \geq \max\{D, ND_{\max} - Z\}$$

□ We know

$$X \leq \frac{1}{D_{\max}} \quad R(N) \geq D$$

Using interactive response time law:

$$R = \frac{N}{X} - Z \quad \longrightarrow \quad R \geq ND_{\max} - Z$$

$$X = \frac{N}{R+Z} \quad \longrightarrow \quad X \leq \frac{N}{D+Z}$$

Summary: Operational Laws

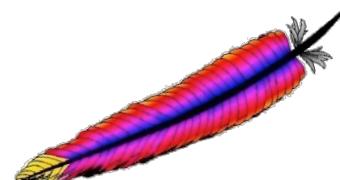
- Utilization law: $U = X_S$
- Forced flow law: $X_i = V_i X$
- Bottleneck device: largest $D_i = V_i S_i$
- Little's Law: $Q_i = X_i R_i$
- Bottleneck bound of interactive response
(for the given closed model):

$$X(N) \leq \min\left\{\frac{1}{D_{\max}}, \frac{N}{D+Z}\right\}$$

$$R(N) \geq \max\{D, ND_{\max} - Z\}$$

In Practice: Common Bottlenecks

- No more file descriptors
- Sockets stuck in TIME_WAIT
- High memory use (swapping)
- CPU overload
- Interrupt (IRQ) overload



[Aaron Bannert]

YouTube Design Alg.

```
while (true)
{
    identify_and_fix_bottlenecks();
    drink();
    sleep();
    notice_new_bottleneck();
}
```

Summary: High-Perf. Network Server

- Avoid blocking (so that we can reach bottleneck throughput)
 - Introduce threads
- Limit unlimited thread overhead
 - Thread pool, async io
- Shared variables
 - Synchronization (lock, synchronized)
- Avoid busy-wait
 - Wait/notify; FSM
- Extensibility/robustness
 - Language support/Design for interfaces
- System modeling and measurements
 - Queueing analysis, operational analysis

Outline

- Recap
- Single network server
- Multiple network servers
 - Why multiple network servers

Why Multiple Servers?

- Scalability
 - Scaling beyond single server capability
 - There is a fundamental limit on what a single server can
 - process (CPU/bw/disk throughput)
 - store (disk/memory)
 - Scaling beyond geographical location capability
 - There is a limit on the speed of light
 - Network detour and delay further increase the delay

Why Multiple Servers?

- Redundancy and fault tolerance
 - Administration/Maintenance (e.g., incremental upgrade)
 - Redundancy (e.g., to handle failures)

Why Multiple Servers?

- System/software architecture
 - Resources may be naturally distributed at different machines (e.g., run a single copy of a database server due to single license; access to resource from third party)
 - Security (e.g., front end, business logic, and database)

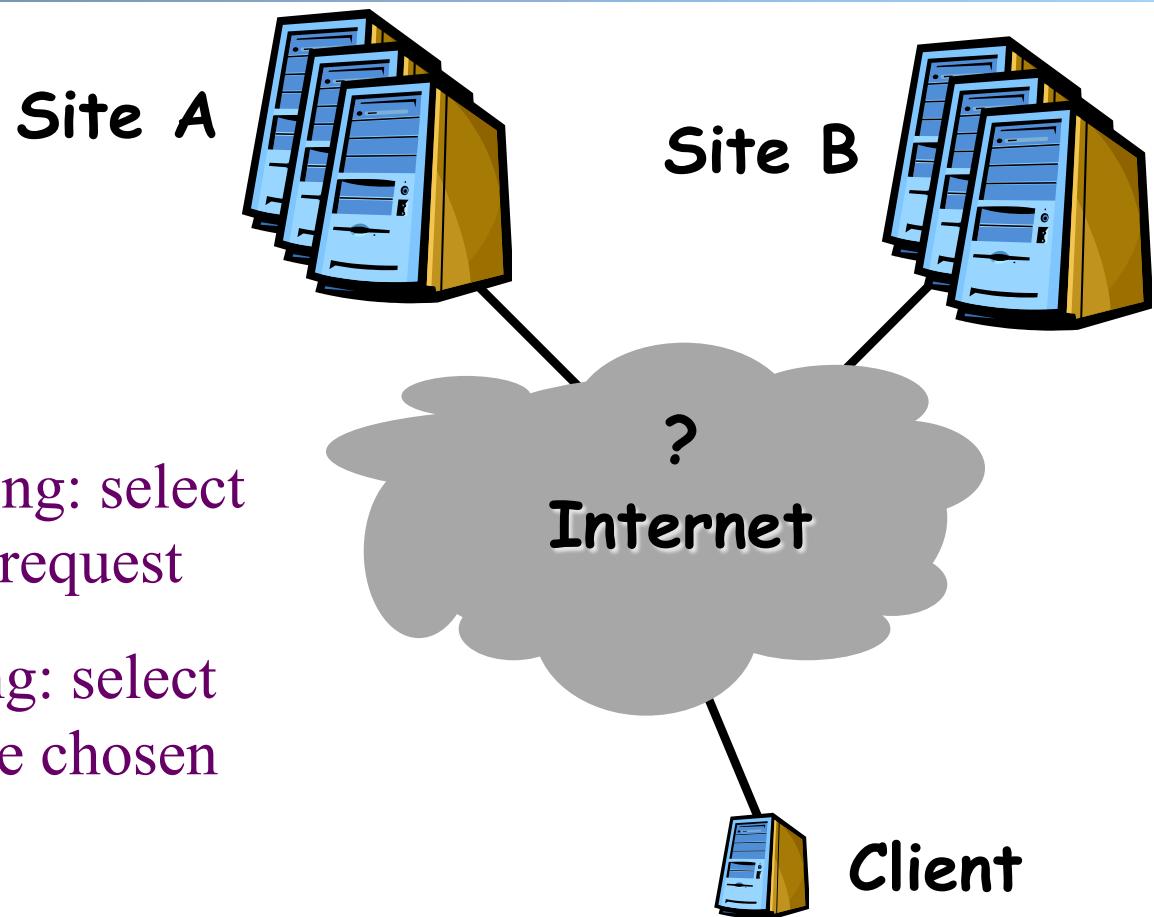
Discussion: Key Technical Challenges in Using Multiple Servers

Outline

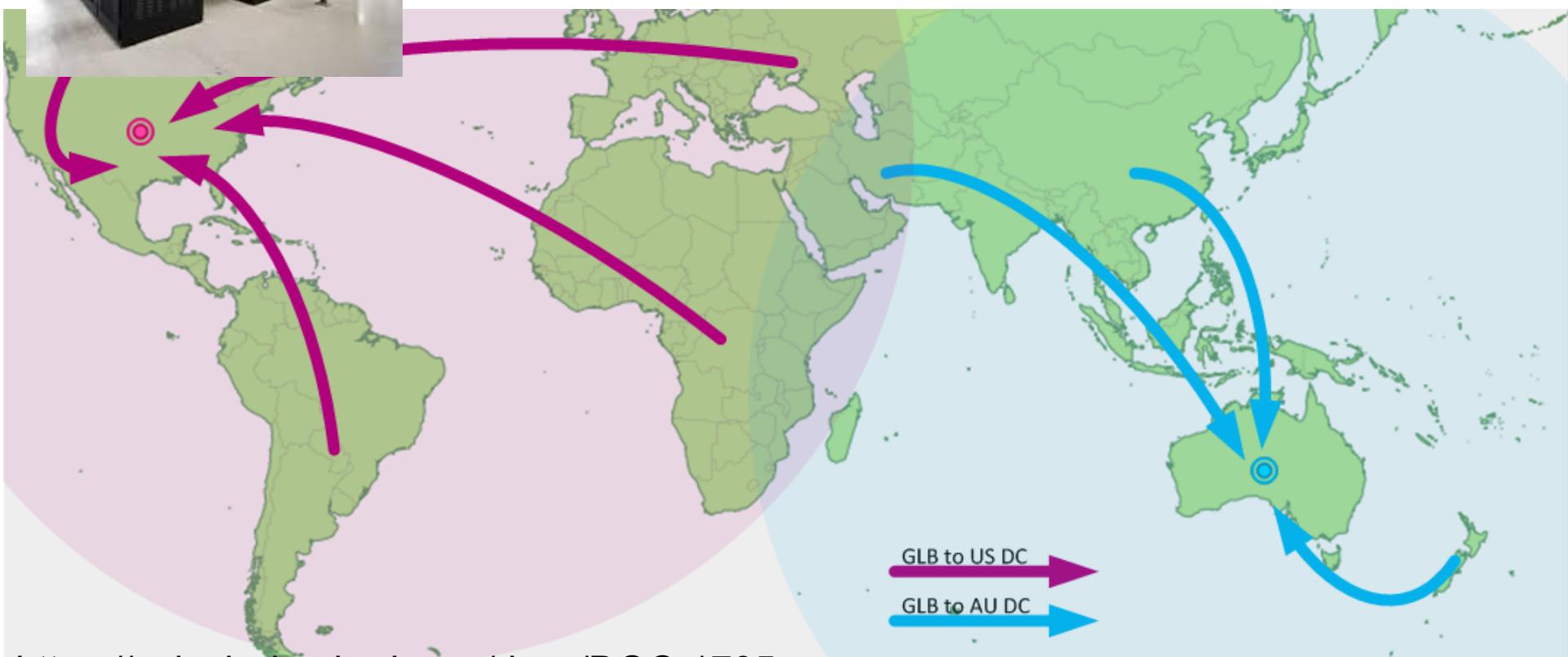
- Recap
- Single network server
- Multiple network servers
 - Why multiple servers
 - Request routing mechanisms

Request Routing: Overview

- Global request routing: select a server site for each request
- Local request routing: select a specific server at the chosen site



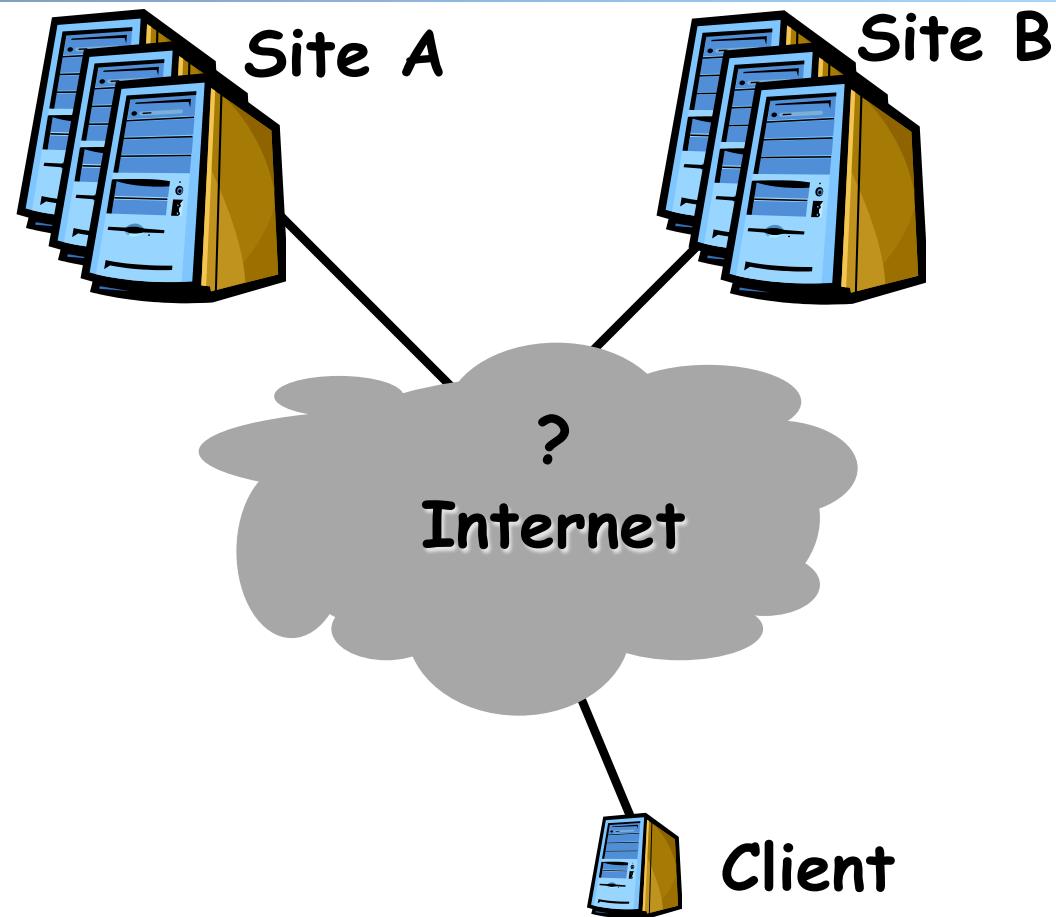
Request Routing: Overview



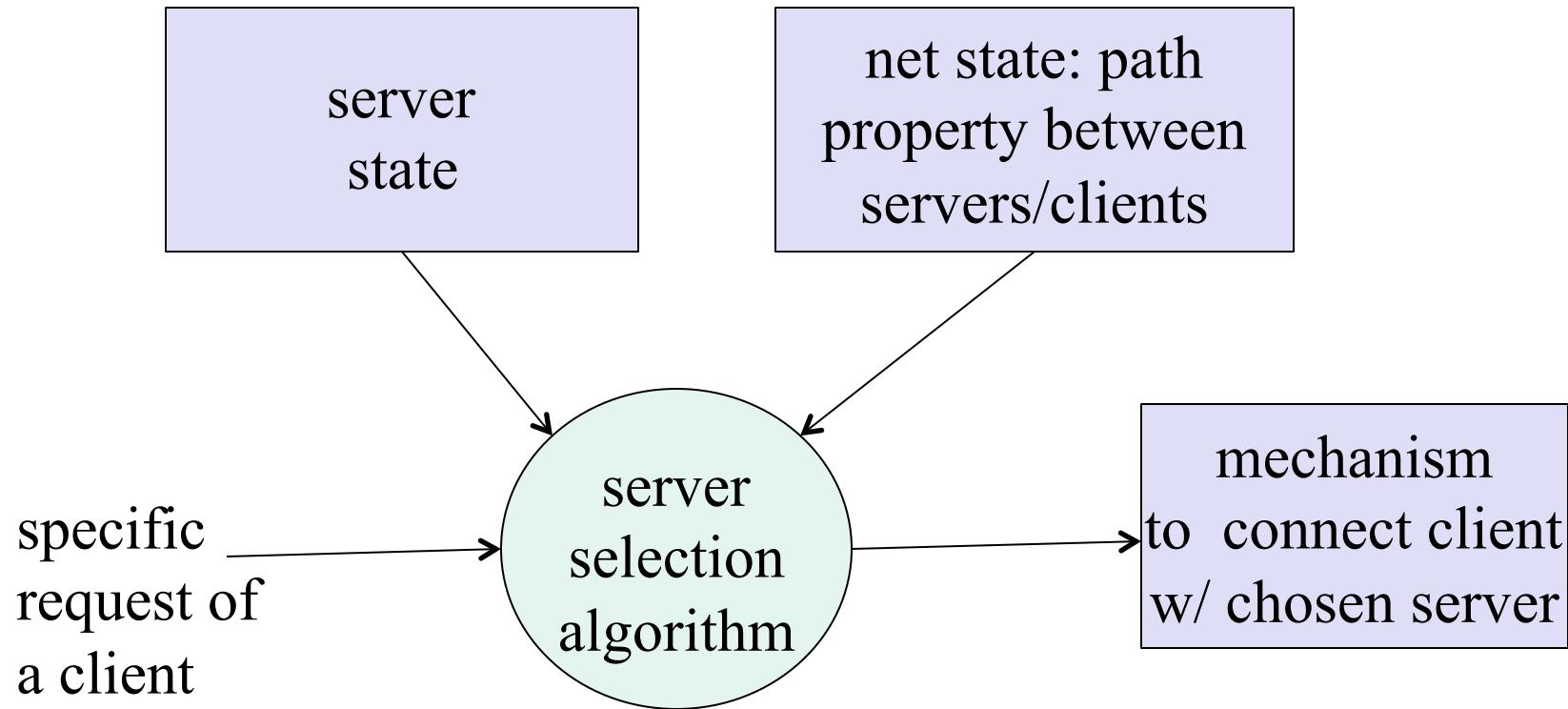
Request Routing: Basic Architecture

□ Major components

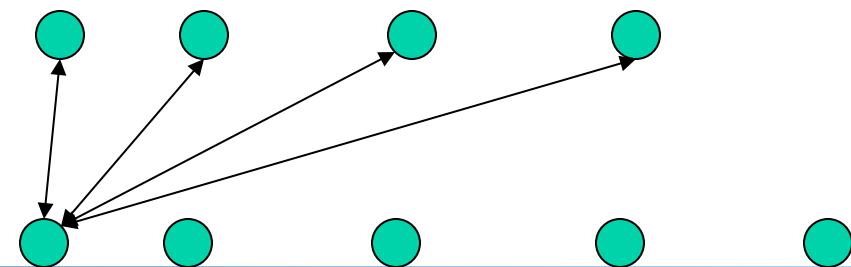
- Server state monitoring
 - Load (incl. failed or not); what requests it can serve
- Network path properties between clients and servers
 - E.g., bw, delay, loss, network cost
- Server selection alg.
- Request direction mechanism



Request Routing: Basic Arch

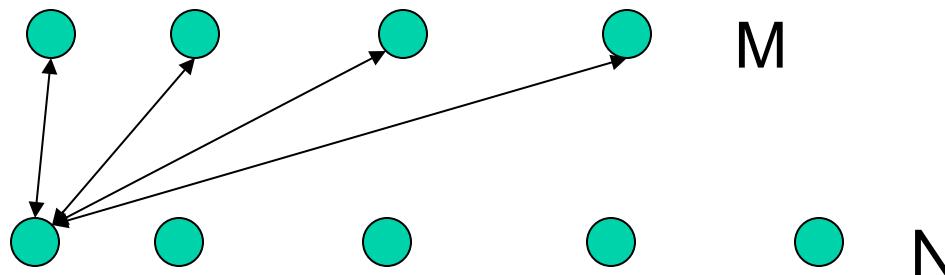


Network Path Properties



□ Why is the problem difficult?

- Scalability: if do measurements, complete measurements grow with $N * M$, where
 - N is # of clients
 - M is # of servers

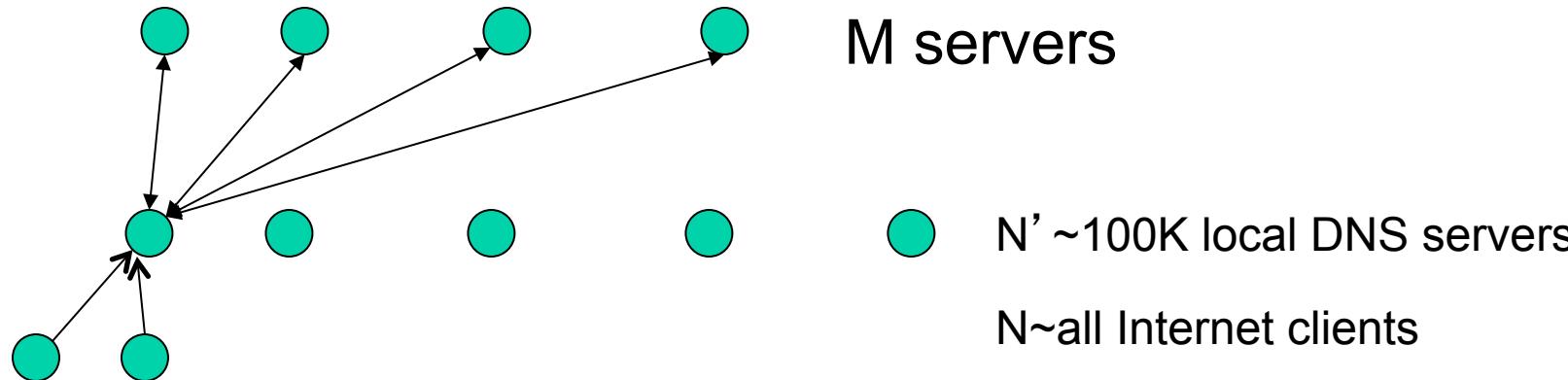


- Complexity/feasibility in computing path metrics

Network Path Properties: Improve Scalability

□ Aggregation:

- merge a set of IP addresses (reduce N and M)
 - E.g., when computing path properties, Akamai aggregates all clients sharing the same local DNS server

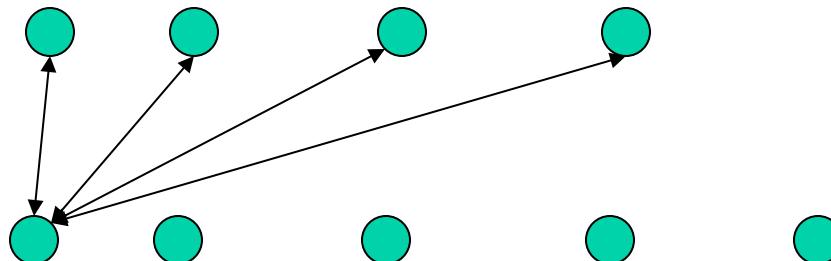


□ Sampling and prediction

- Instead of measuring $N \times M$ entries, we measure a subset and **predict the unmeasured paths**
- We will cover it later in the course

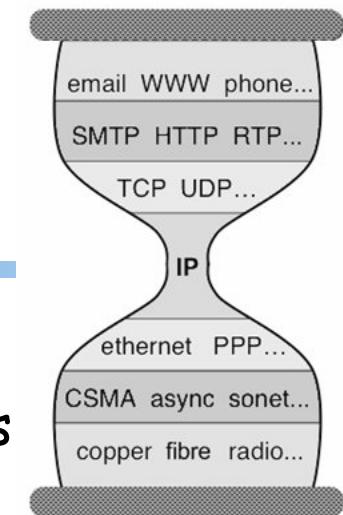
Server Selection

- Why is the problem difficult?
 - What are potential problems of just sending each new client to the lightest load server?

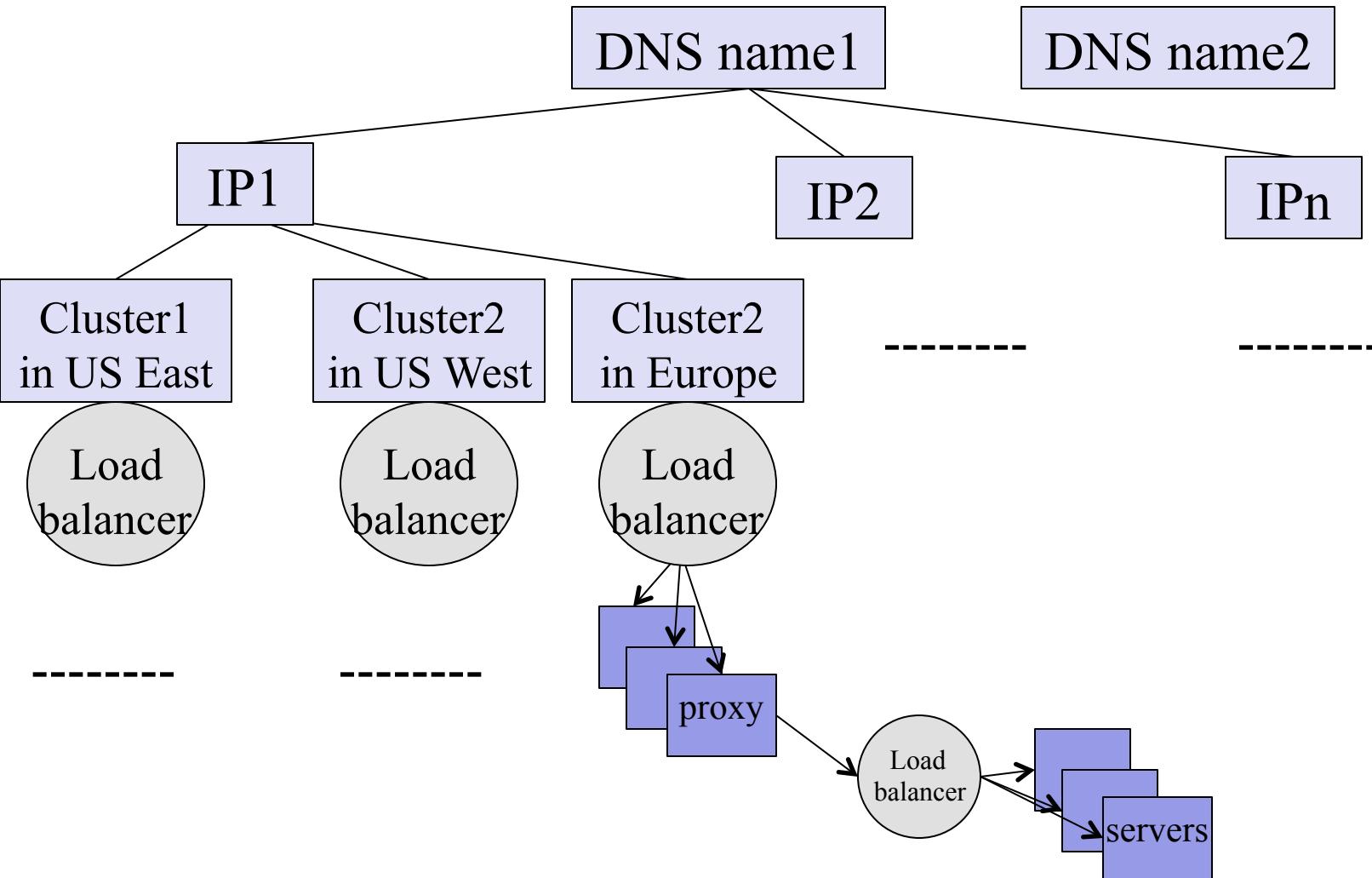


Client Direction Mechanisms

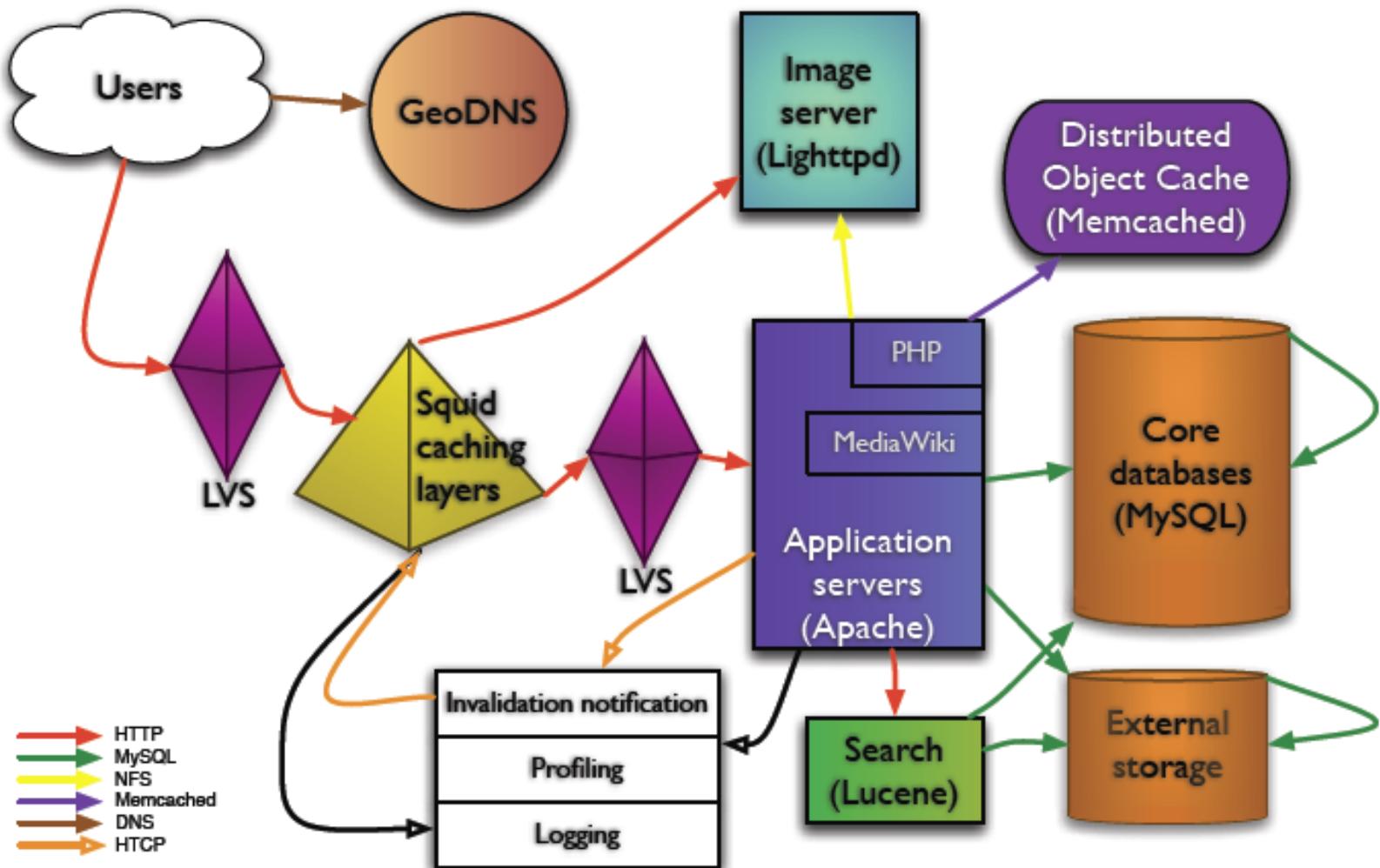
- Application layer
 - App/user is given a list of candidate server names
 - HTTP redirector
- DNS: name resolution gives a list of server addresses
- IP layer: Same IP address represents multiple physical servers
 - IP **anycast**: Same IP address shared by multiple servers and announced at different parts of the Internet. Network directs different clients to different servers
 - Smart-switch indirection: a server IP address may be a **virtual IP** address for a cluster of physical servers



Direction Mechanisms are Often Combined



Example: Wikipedia Architecture



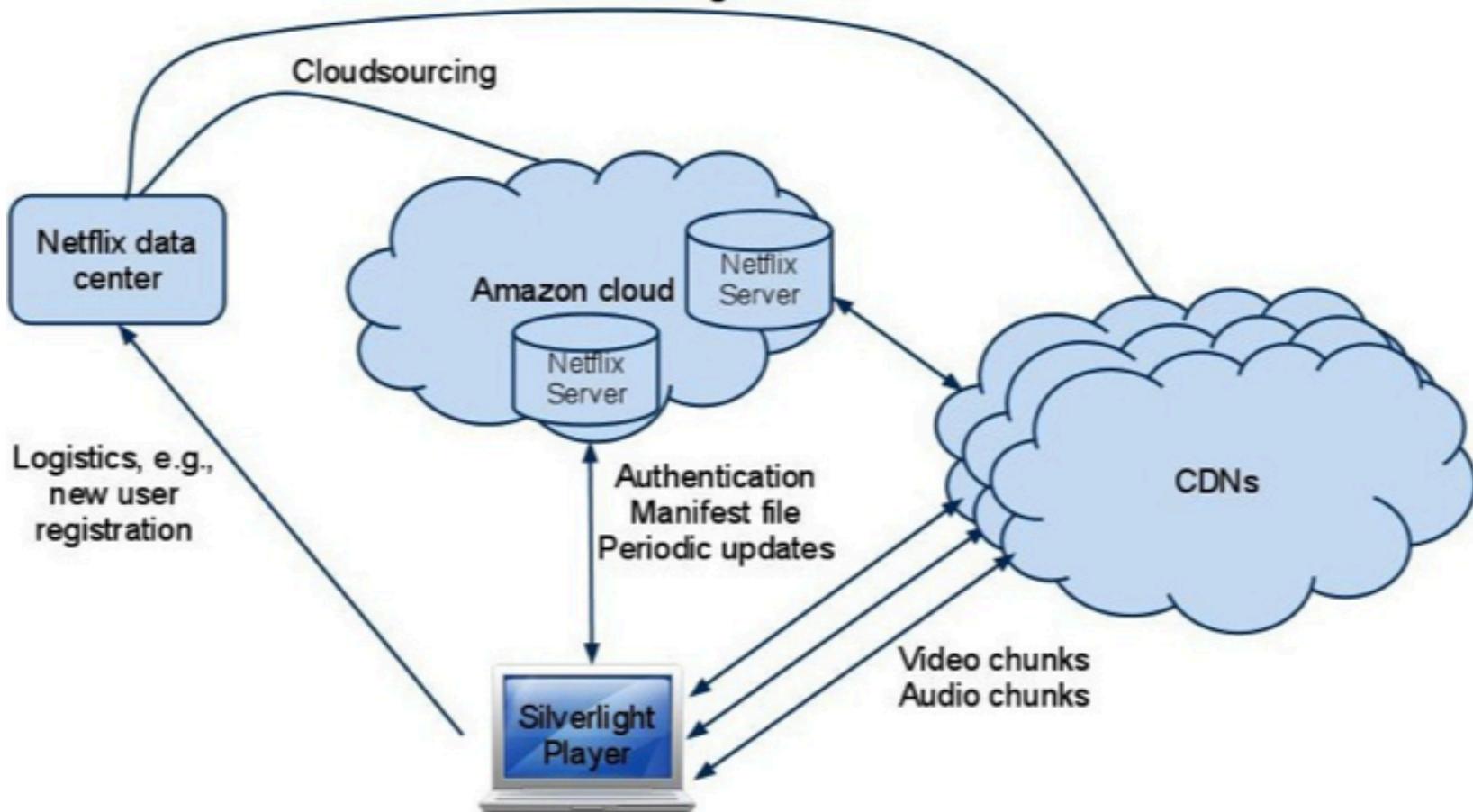
Outline

- Recap
- Single network server
- Multiple network servers
 - Why multiple servers
 - Request routing mechanisms
 - Overview
 - Application-layer

Example: Netflix

Hostname	Organization
www.netflix.com	Netflix
signup.netflix.com	Amazon
movies.netflix.com	Amazon
agmoviecontrol.netflix.com	Amazon
nflx.i.87f50a04.x.lcdn.nflximg.com	Level 3
netflix-753.vo.llnwd.net	Limelight
netflix753.as.nflximg.com.edgesuite.net	Akamai

Outsourcing



Example: Netflix Manifest File

- Client player authenticates and then downloads manifest file from servers at Amazon Cloud

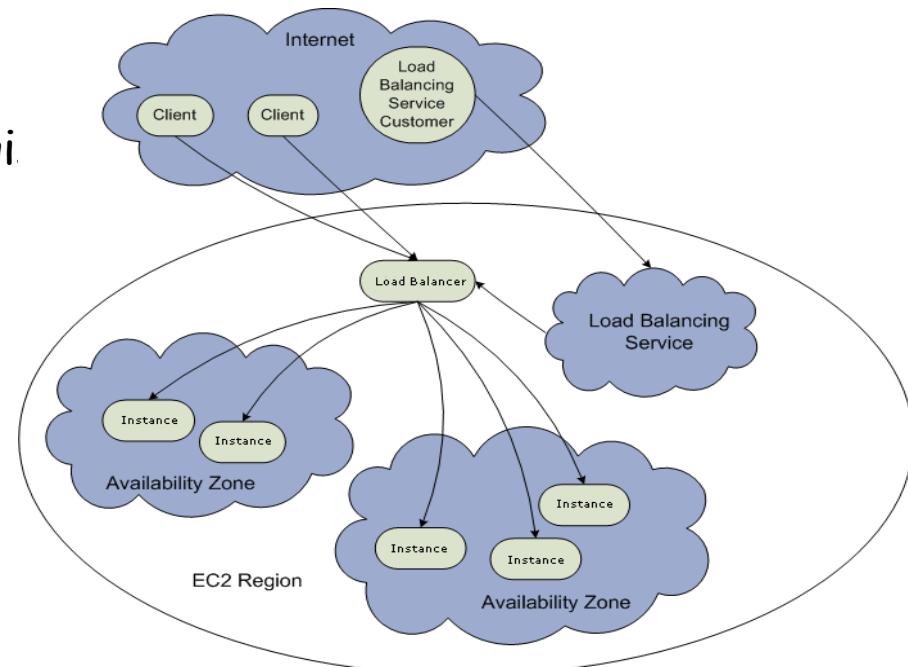
```
<nccp:cdns>
  <nccp:cdn>
    <nccp:name>level3</nccp:name>
    <nccp:cdnid>6</nccp:cdnid>
    <nccp:rank>1</nccp:rank>
    <nccp:weight>140</nccp:weight>
  </nccp:cdn>
  <nccp:cdn>
    <nccp:name>limelight</nccp:name>
    <nccp:cdnid>4</nccp:cdnid>
    <nccp:rank>2</nccp:rank>
    <nccp:weight>120</nccp:weight>
  </nccp:cdn>
  <nccp:cdn>
    <nccp:name>akamai</nccp:name>
    <nccp:cdnid>9</nccp:cdnid>
    <nccp:rank>3</nccp:rank>
    <nccp:weight>100</nccp:weight>
  </nccp:cdn>
</nccp:cdns>
```

Example: Netflix Manifest File

```
<nccp:bitrate>560</nccp:bitrate>
<nccp:videoprofile>
  playready-h264mpl30-dash
</nccp:videoprofile>
<nccp:resolution>
  <nccp:width>512</nccp:width>
  <nccp:height>384</nccp:height>
</nccp:resolution>
<nccp:pixelaspect>
  <nccp:width>4</nccp:width>
  <nccp:height>3</nccp:height>
</nccp:pixelaspect>
<nccp:downloadurls>
  <nccp:downloadurl>
    <nccp:expiration>131xxx</nccp:expiration>
    <nccp:cdnid>6</nccp:cdnid>
    <nccp:url>http://nflx.i..../...</nccp:url>
  </nccp:downloadurl>
  <nccp:downloadurl>
    <nccp:expiration>131xxx</nccp:expiration>
    <nccp:cdnid>4</nccp:cdnid>
    <nccp:url>http://netflix.../...</nccp:url>
  </nccp:downloadurl>
  <nccp:downloadurl>
    <nccp:expiration>131xxx</nccp:expiration>
    <nccp:cdnid>9</nccp:cdnid>
    <nccp:url>http://netflix.../...</nccp:url>
  </nccp:downloadurl>
</nccp:downloadurls>
```

Example: Amazon Elastic Cloud 2 (EC2) Elastic Load Balancing

- Use the *create-load-balancer* command to create an Elastic Load Balancer.
- Use the *register-instances-with-load-balancer* command to register Amazon EC2 instances that you want to load balance with the Elastic Load Balancer.
- Elastic Load Balancing automatically checks the health of your load balancing Amazon EC2 instances. You can optionally customize the health checks by using the *configure-healthcheck* command.
- Traffic to the DNS name provided by the Elastic Load Balancer is automatically distributed across healthy Amazon EC2 instances.



Details: Create Load Balancer

The operation returns the DNS name of your LoadBalancer.
You can then map that to any other domain name (such as
`www.mywebsite.com`)
(how?)

```
%aws elb create-load-balancer --load-
balancer-name my-load-balancer --listeners
"Protocol=HTTP,LoadBalancerPort=80,InstanceP
rotocol=HTTP,InstancePort=80" --
availability-zones us-west-2a us-west-2b
```

Result:

```
{ "DNSName": "my-load-balancer-123456789.us-west-2.elb.amazonaws.com"}
```

Details: Configure Health Check

The operation configures how instances are monitored, e.g.,

```
%aws elb configure-health-check --load-
balancer-name my-load-balancer --health-
check Target=HTTP:80/
png,Interval=30,UnhealthyThreshold=2,Healthy
Threshold=2,Timeout=3
```

Result:

```
{
    "HealthCheck": {
        "HealthyThreshold": 2,
        "Interval": 30,
        "Target": "HTTP:80/png",
        "Timeout": 3,
        "UnhealthyThreshold": 2
    }
}
```

Details: Register Instances

The operation registers instances that can receive traffic,

```
%aws elb register-instances-with-load-
balancer --load-balancer-name my-load-
balancer --instances i-d6f6fae3
```

Result:

```
{  "Instances": [
    {"InstanceId": "i-d6f6fae3"},  

    {"InstanceId": "i-207d9717"},  

    {"InstanceId": "i-afefb49b"}  

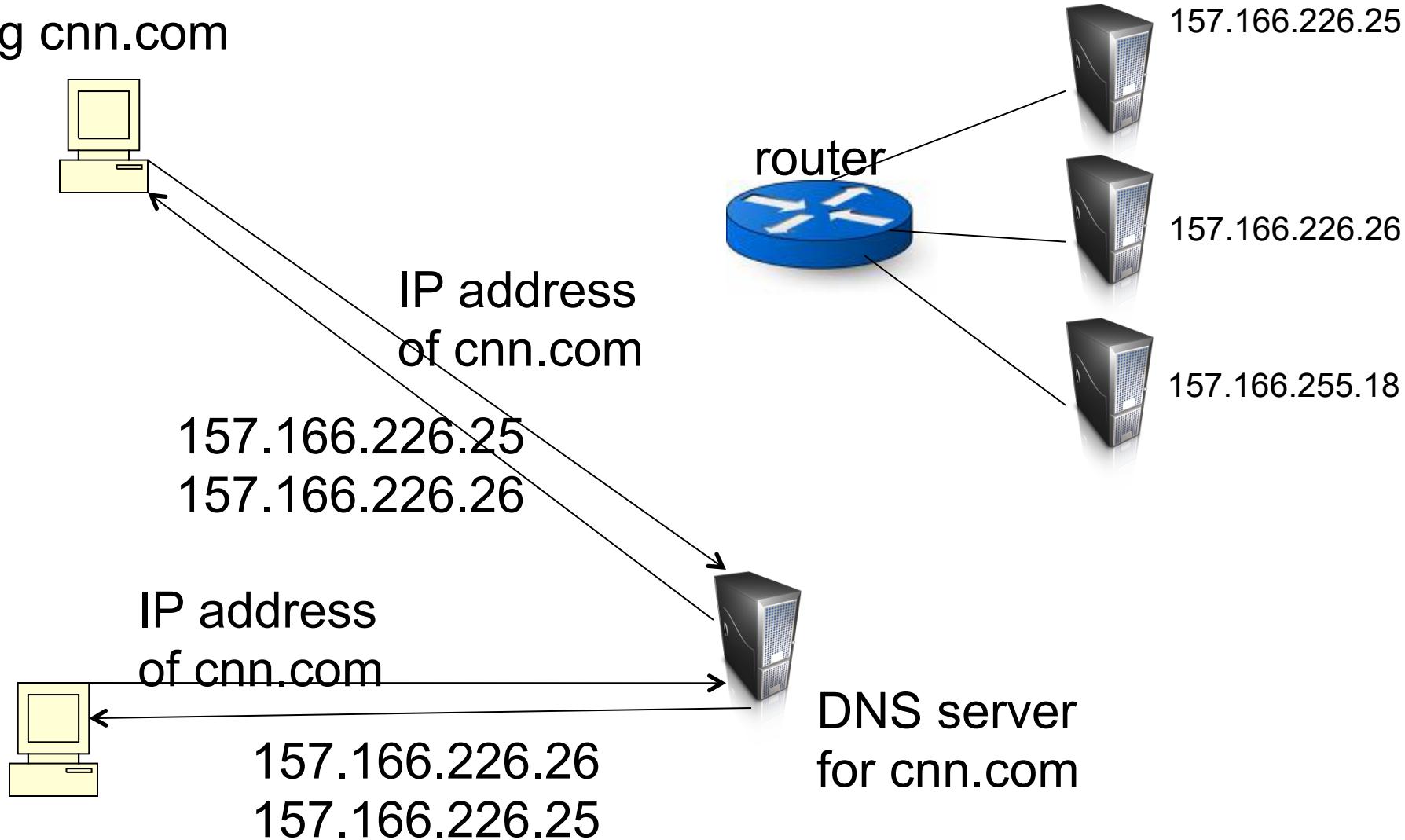
]  
}
```

Outline

- Recap
- Single network server
- Multiple network servers
 - Why multiple servers
 - Request routing mechanisms
 - Overview
 - Application-layer
 - DNS

Basic DNS Indirection and Rotation

%dig cnn.com



CDN Using DNS (Akamai Architecture as an Example)

- Content publisher (e.g., NYTimes)
 - provides base HTML documents
 - runs **origin** server(s)

- Akamai runs
 - (~200,000) **edge** servers for hosting content
 - Deployment into 110 countries and 1400 networks
 - customized **DNS redirection servers** to select edge servers based on
 - closeness to client browser
 - server load

Source: <https://www.akamai.com/us/en/about/facts-figures.jsp>

Linking to Akamai

- ❑ Originally, URL Akamaization of embedded content: e.g.,

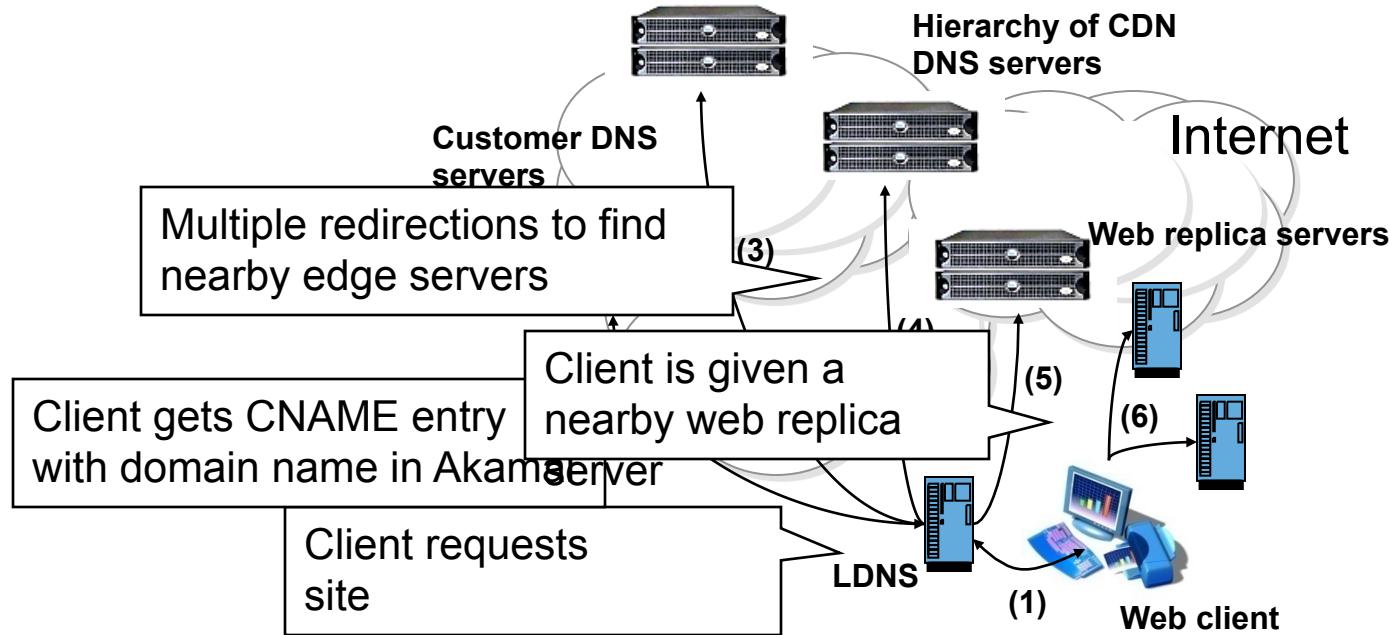
changed to

<IMGSRC = http://a661. g.akamai.net/hash/image.gif>

Note that this DNS redirection unit is per customer, not individual files.

- ❑ URL Akamaization is becoming obsolete and supported mostly for legacy reasons
 - Currently most content publishers prefer to use DNS CNAME to link to Akamai servers

Akamai Load Direction Flow



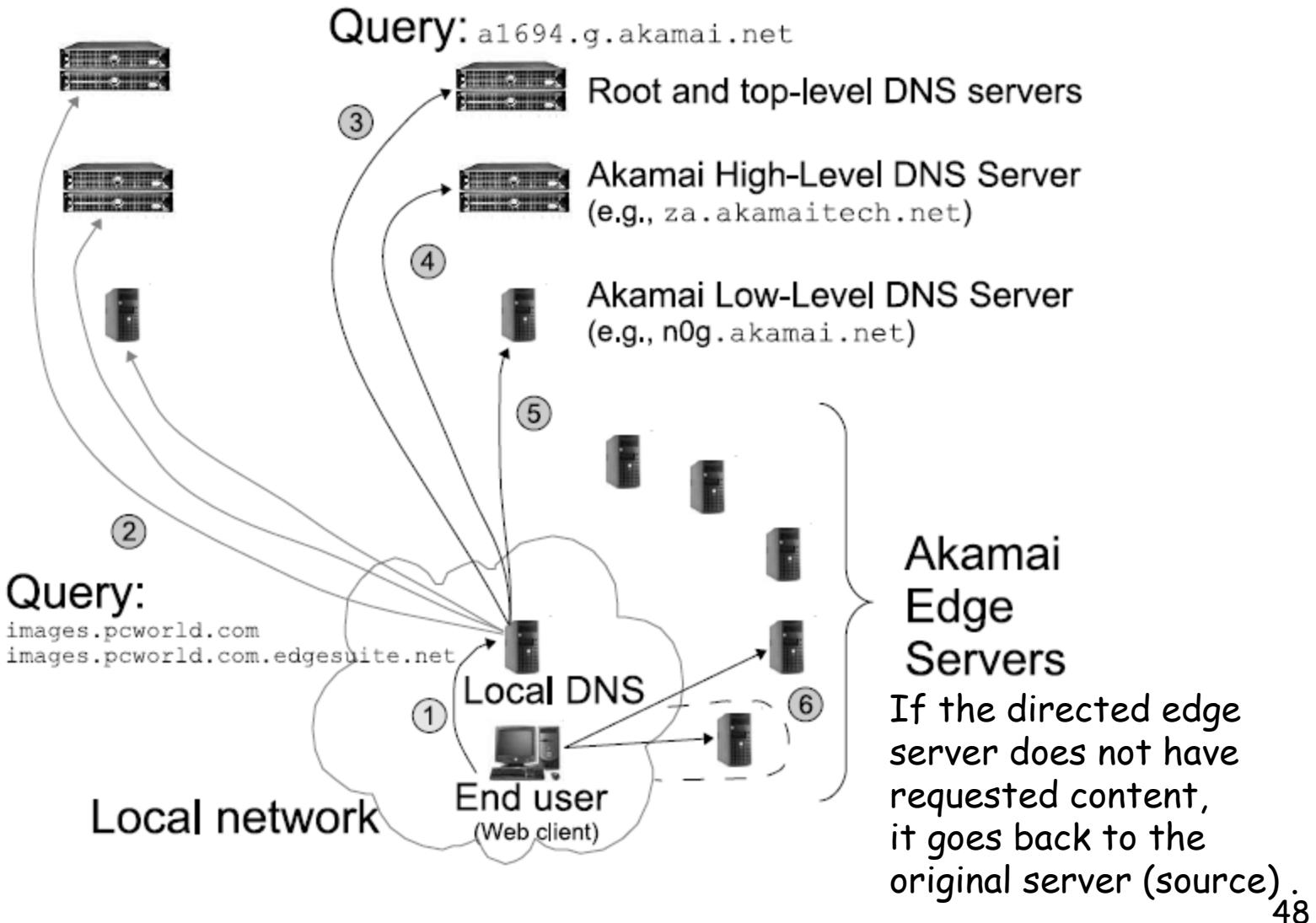
More details see “Global hosting system”: FT Leighton, DM Lewin – US Patent 6,108,703, 2000.

Exercise: Zoo machine

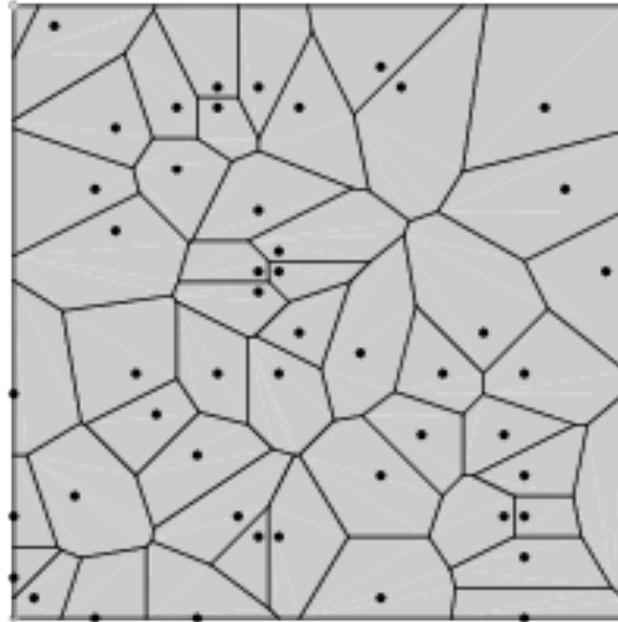
- Check any web page of New York Times and find a page with an image
- Find the URL
- Use

```
%dig +trace +recurse  
to see Akamai load direction
```

Akamai Load Direction



Two-Level Direction



proximity: high-level DNS determines client location; directs to low-level DNS, who manages a close-by cluster

Local DNS Alg: Potential Input

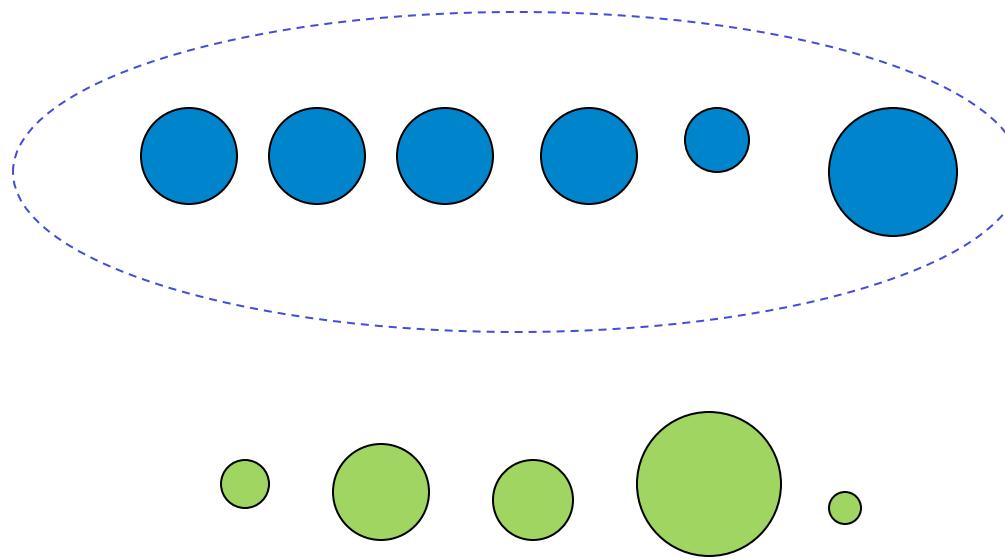
- $p(m, e)$: path properties (from a client site m to an edge sever e)
 - Akamai might use a one-hop detour routing (see [akamai-detour.pdf](#))
- a^k_m : request load from client site m to publisher k

- x_e : load on edge server e
- caching state of a server e

Local DNS Alg

- Details of Akamai algorithms are proprietary
- A Bin-Packing algorithm (column 12 of Akamai Patent) every T second
 - Compute the load to each publisher k (called serial number)
 - Sort the publishers from increasing load
 - For each publisher, associate a list of random servers generated by a hash function
 - Assign the publisher to the first server that does not overload

Hash Bin-Packing

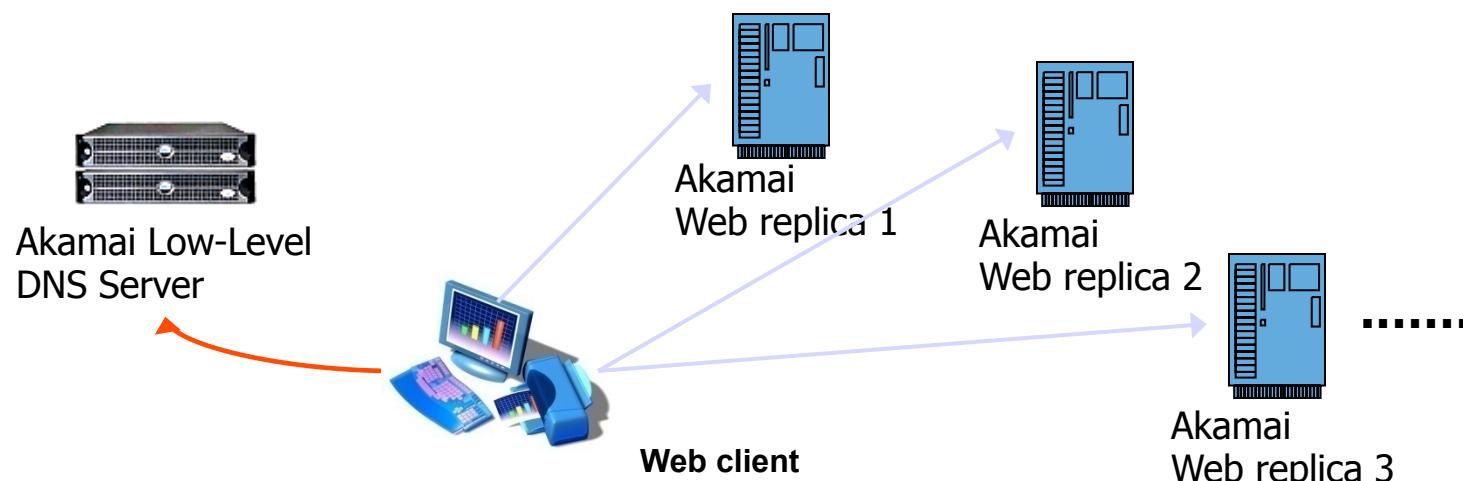


LB: maps request to individual
machines inside cluster

Experimental Study of Akamai Load Balancing

□ Methodology

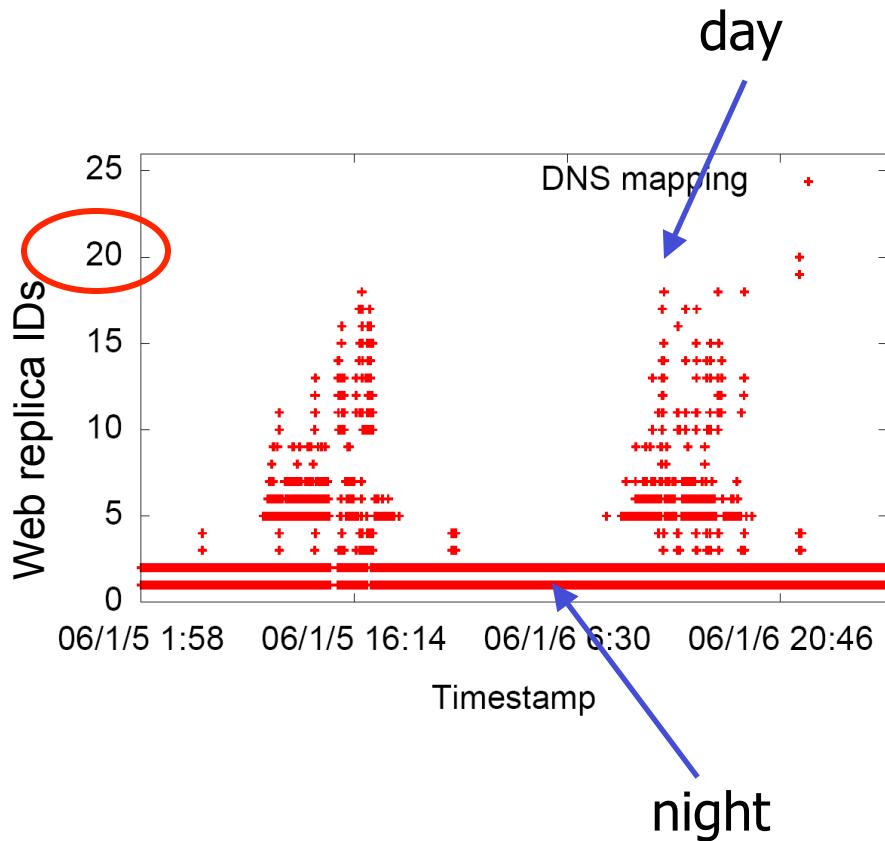
- 2-months long measurement
- 140 PlanetLab nodes (clients)
 - 50 US and Canada, 35 Europe, 18 Asia, 8 South America, the rest randomly scattered
- Every 20 sec, each client queries an appropriate CNAME for Yahoo, CNN, Fox News, NY Times, etc.



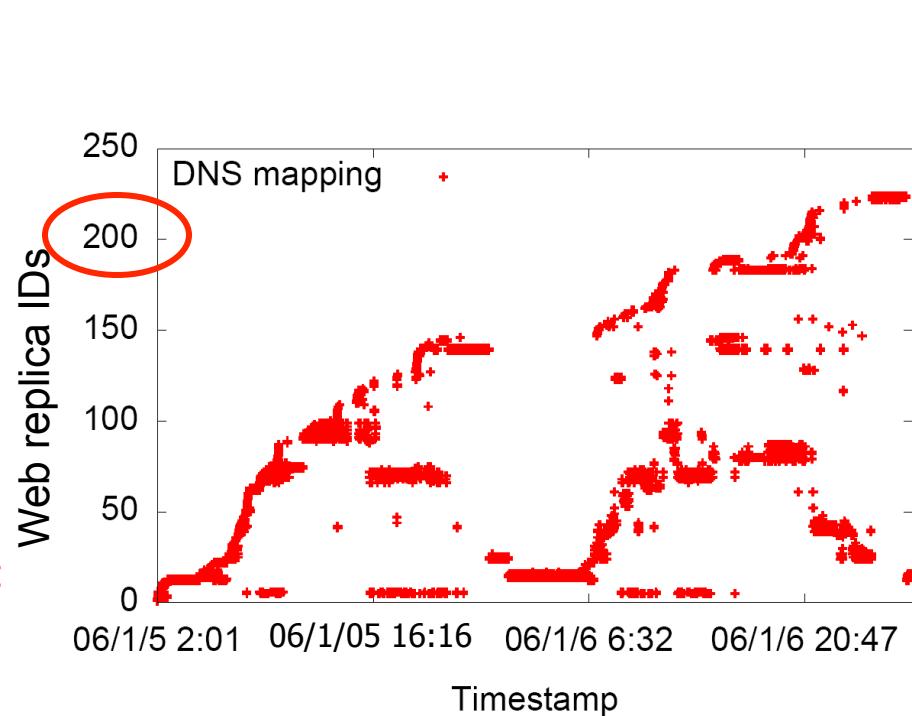
Server Pool: to Yahoo

Target: a943.x.a.yimg.com (Yahoo)

Client 1: Berkeley

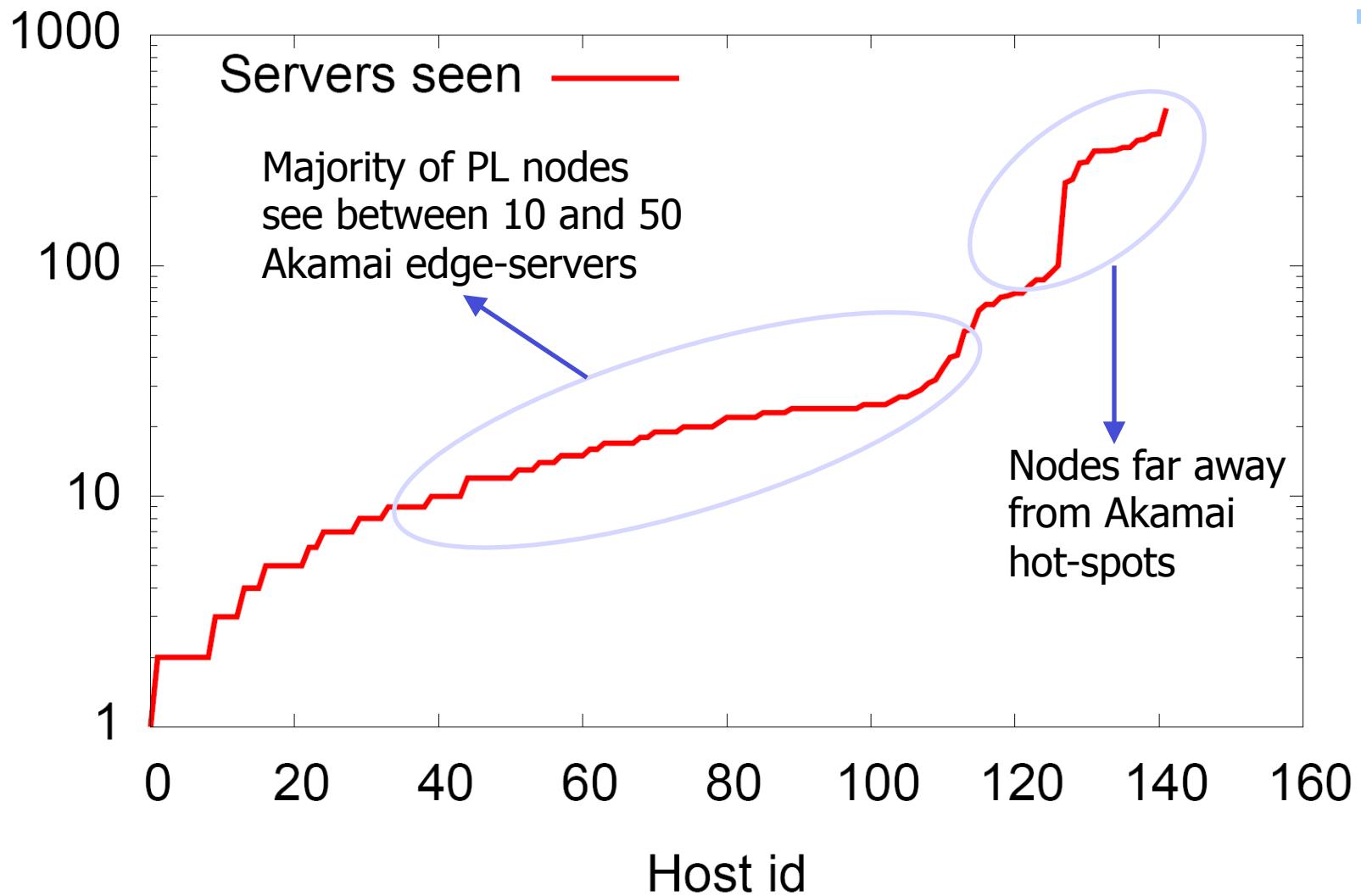


Client 2: Purdue

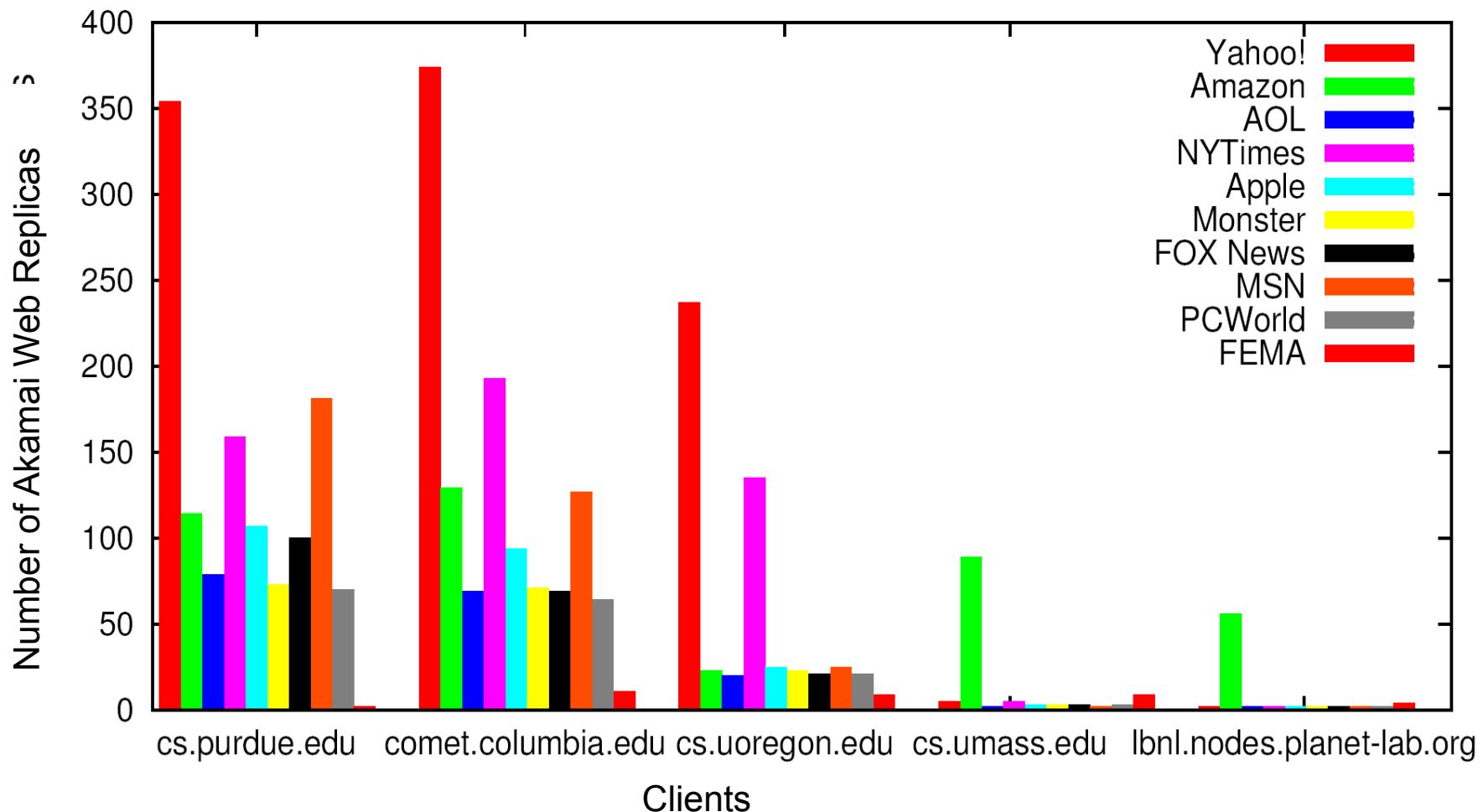


Server Diversity for Yahoo

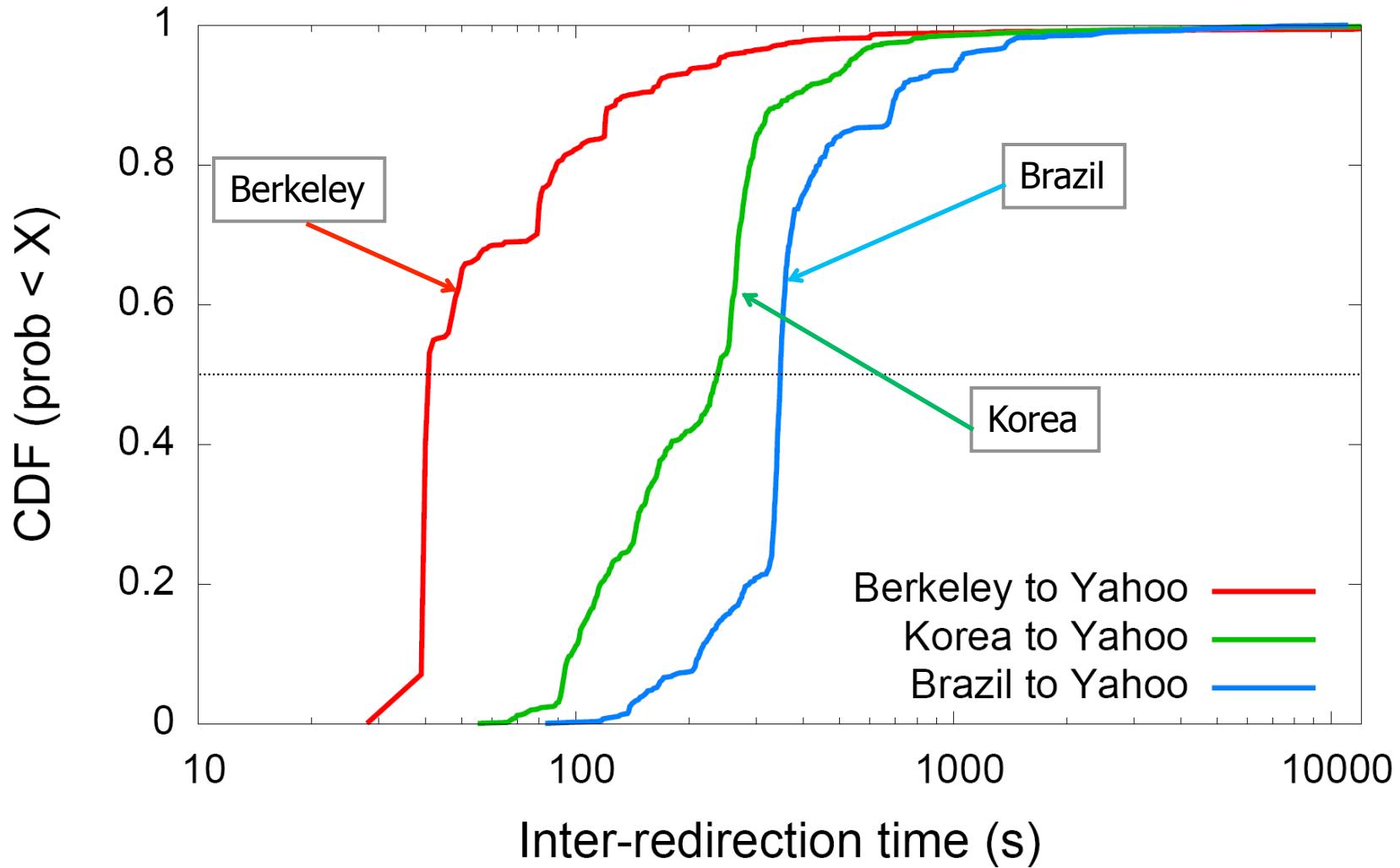
Unique edge server IPs seen



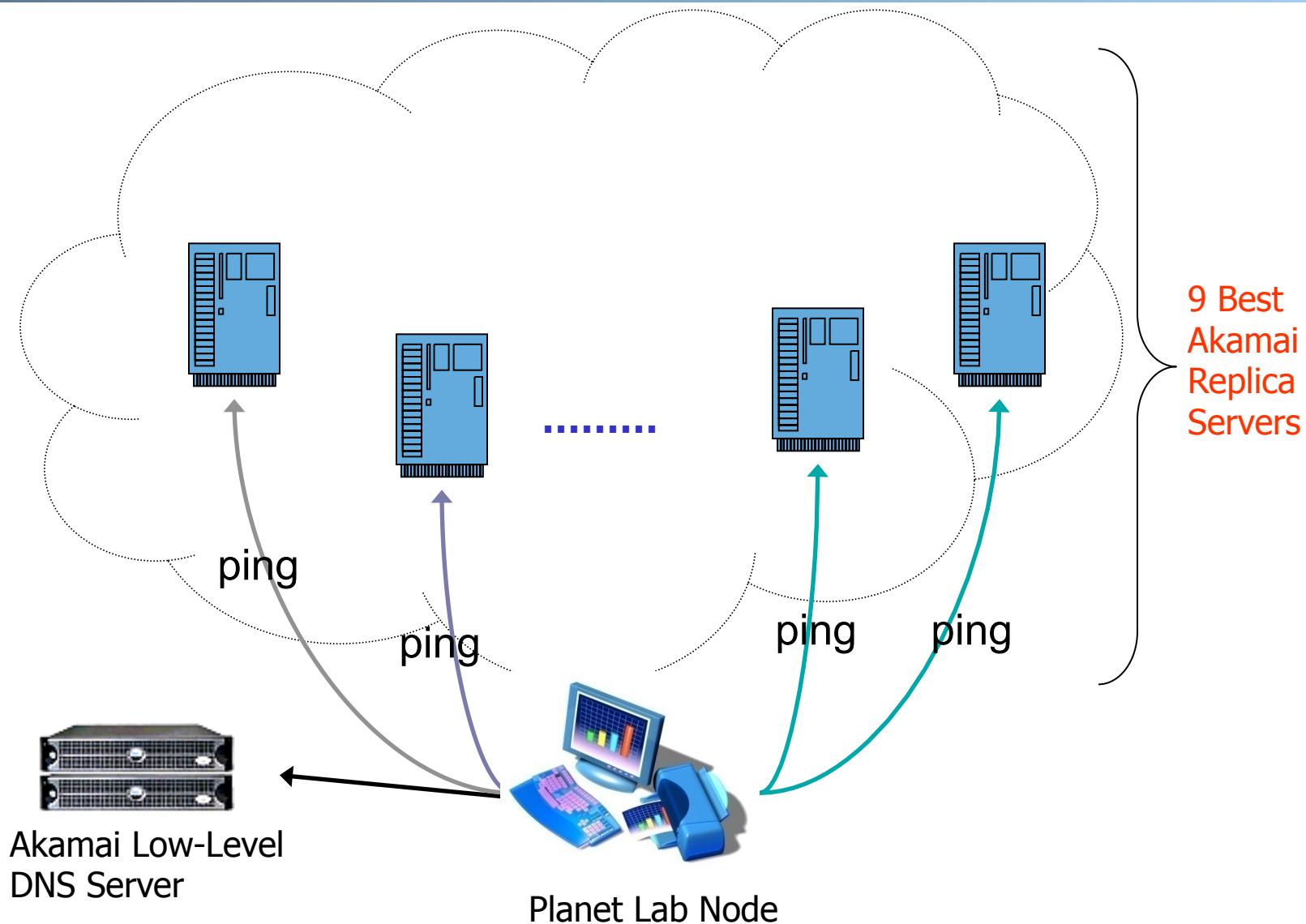
Server Pool: Multiple Akamai Hosted Sites



Load Balancing Dynamics



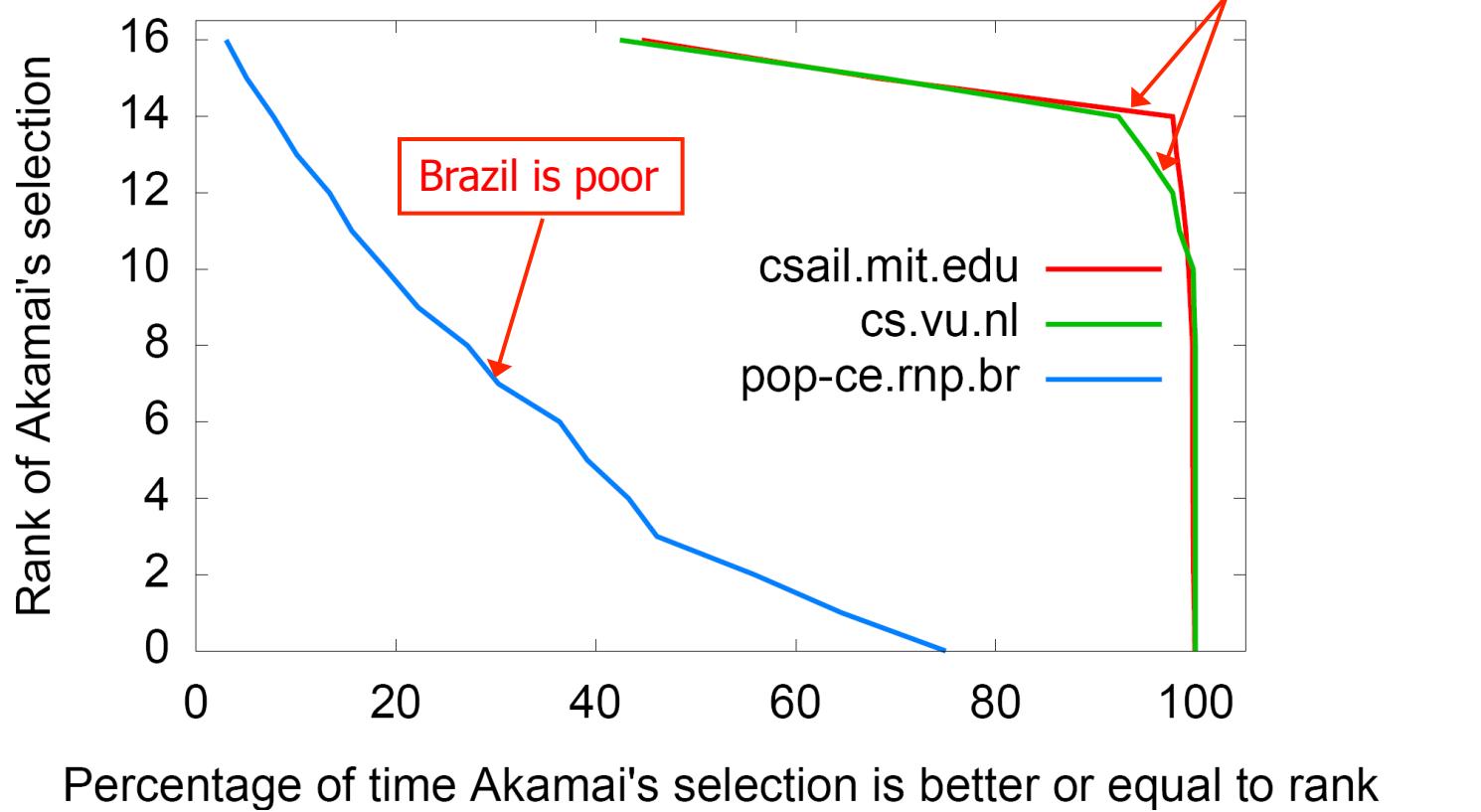
Redirection Effectiveness: Measurement Methodology



Do redirections reveal network conditions?

☐ Rank = $r1+r2-1$

- 16 means perfect correlation



(Offline Read) Facebook DNS Load Direction

- ❑ A system named Cartographer (written in Python) processes measurement data and configures the DNS maps of individual DNS servers (open source tinydns)

Discussion

- ❑ Advantages and disadvantages of using DNS

Network Applications:
LBN using Smart Switch;
Application Overlay Networks

Y. Richard Yang

<http://zoo.cs.yale.edu/classes/cs433/>

3/2/2016

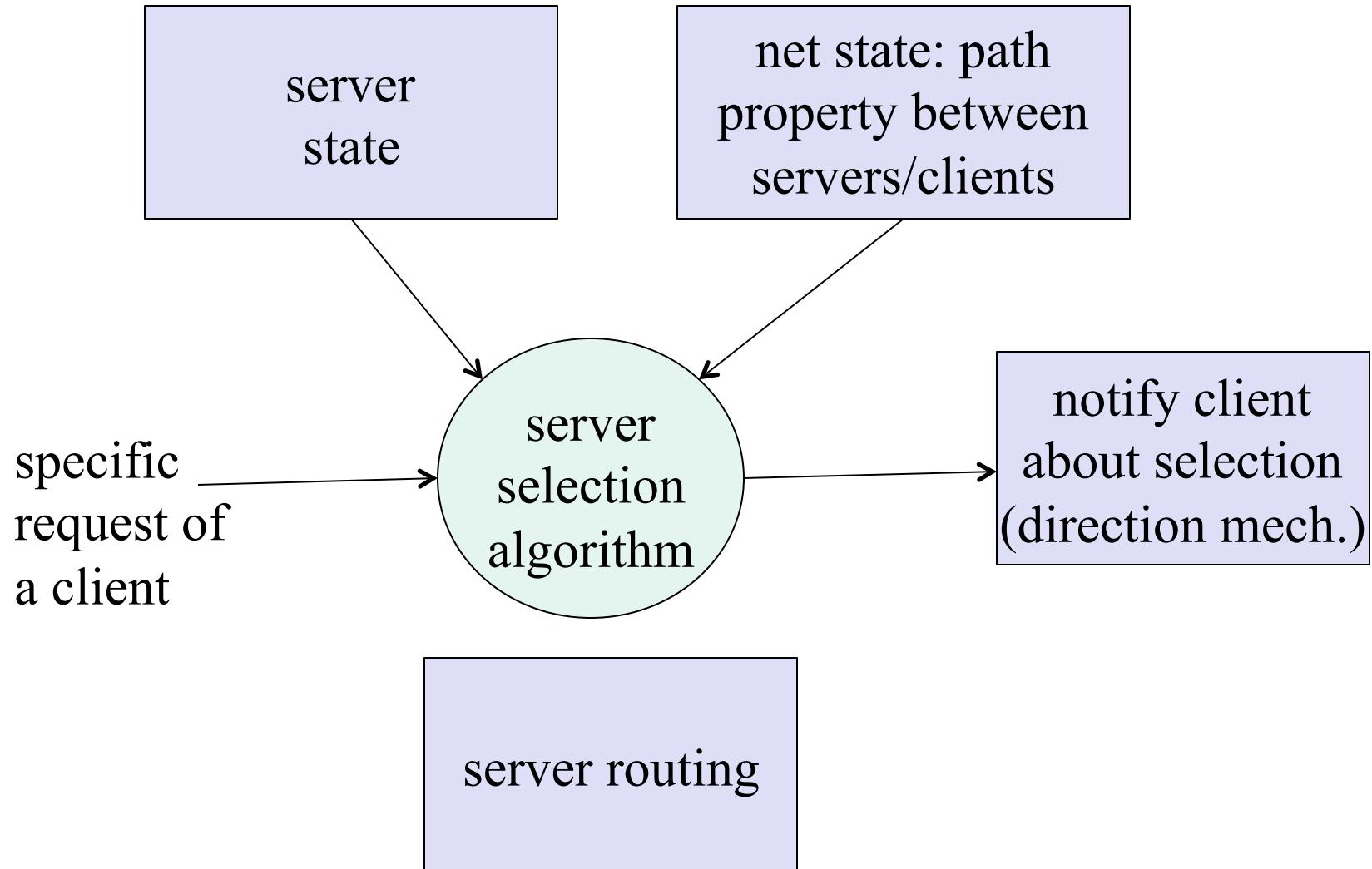
Outline

- Admin and recap
- Multiple servers
- Application overlays
 - Overlays for scalability
 - Overlays for anonymity
 - Overlays for distributed content hosting

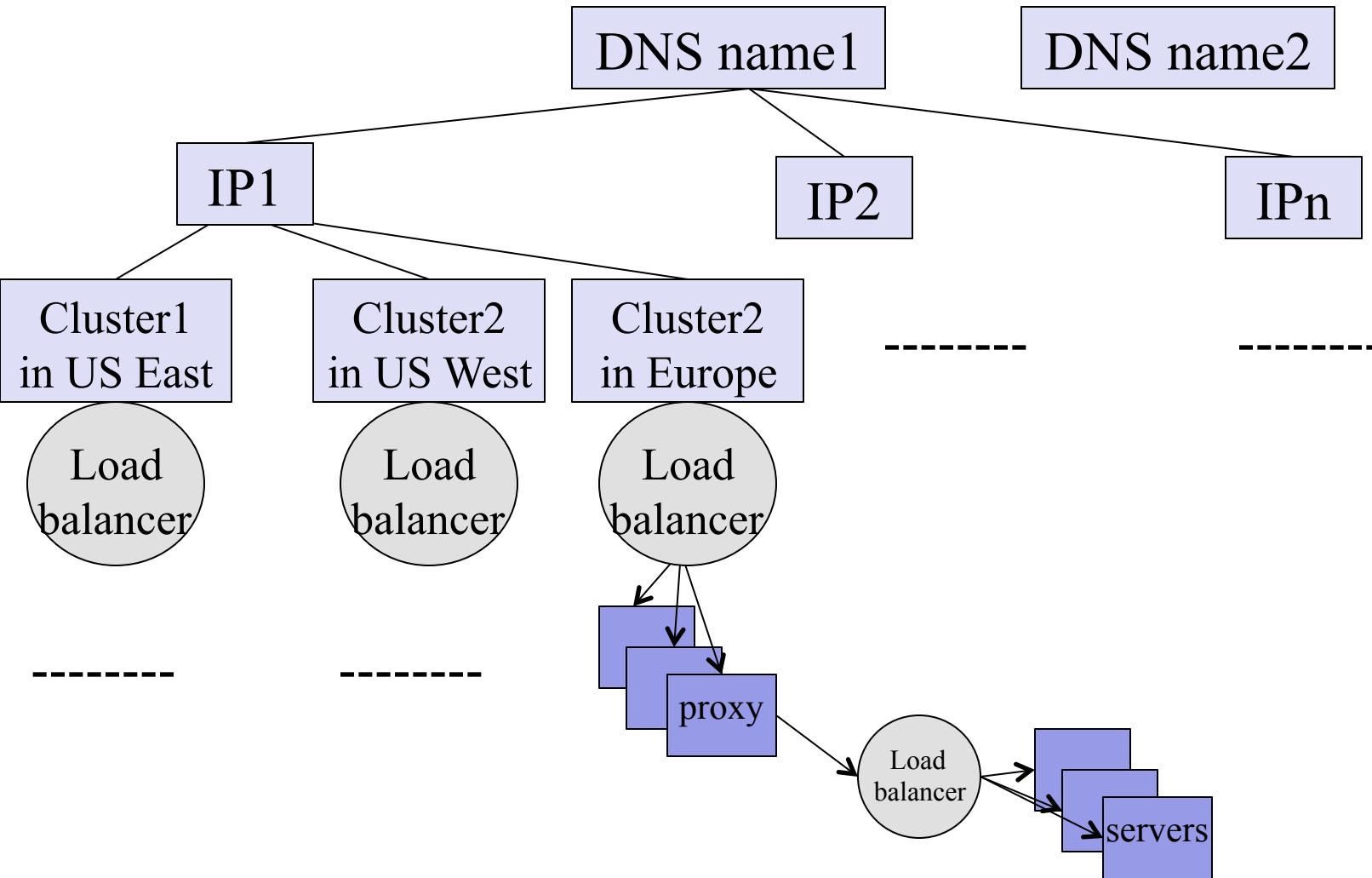
Admin

- Assignment three status and questions.

Recap: LBN Arch



Recap: LBN Mechanisms



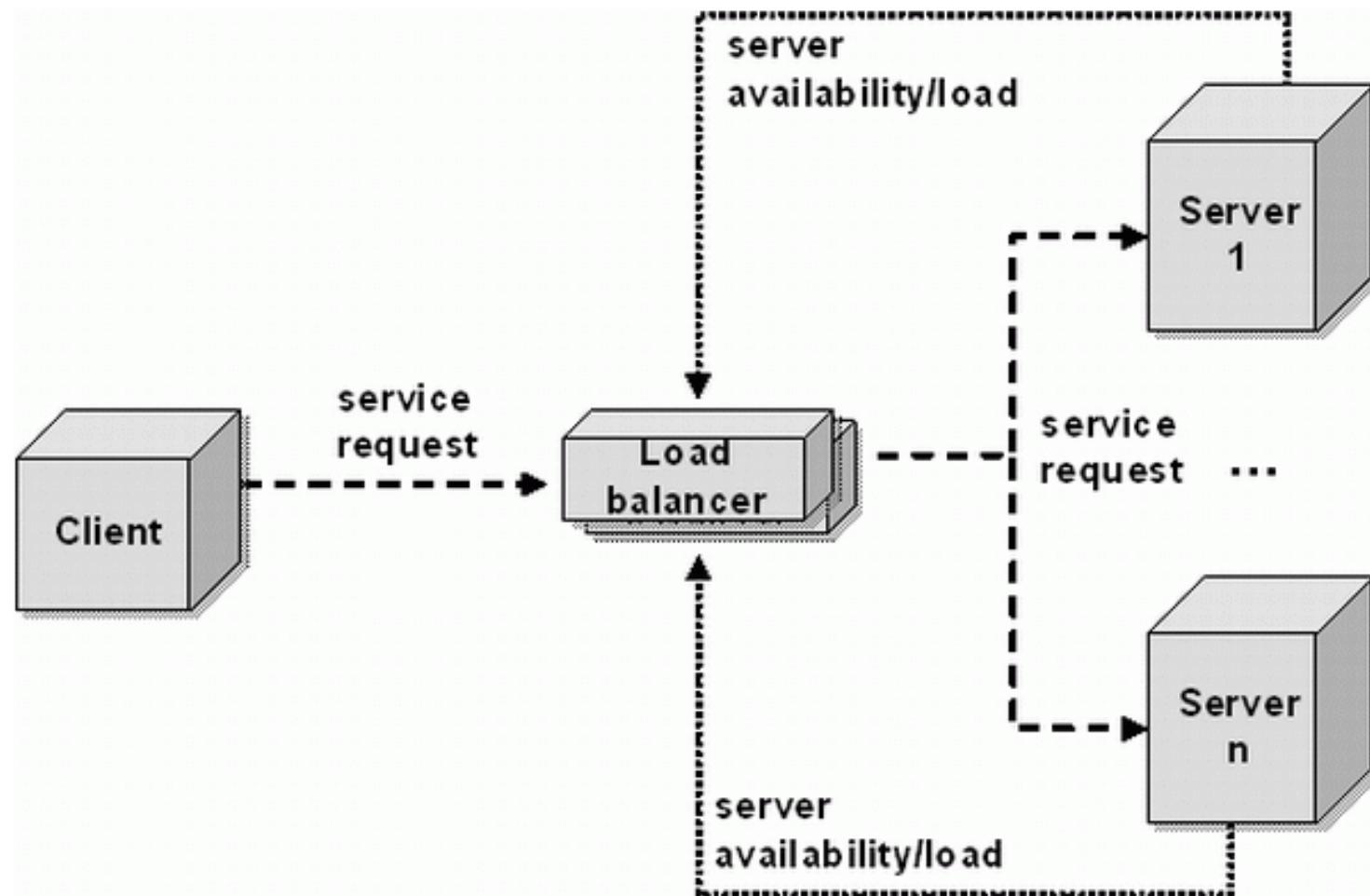
Problem

- Clients get a single service IP address, but we want to use a cluster of (physical) servers
 - The single IP is called a **virtual IP address (VIP)**

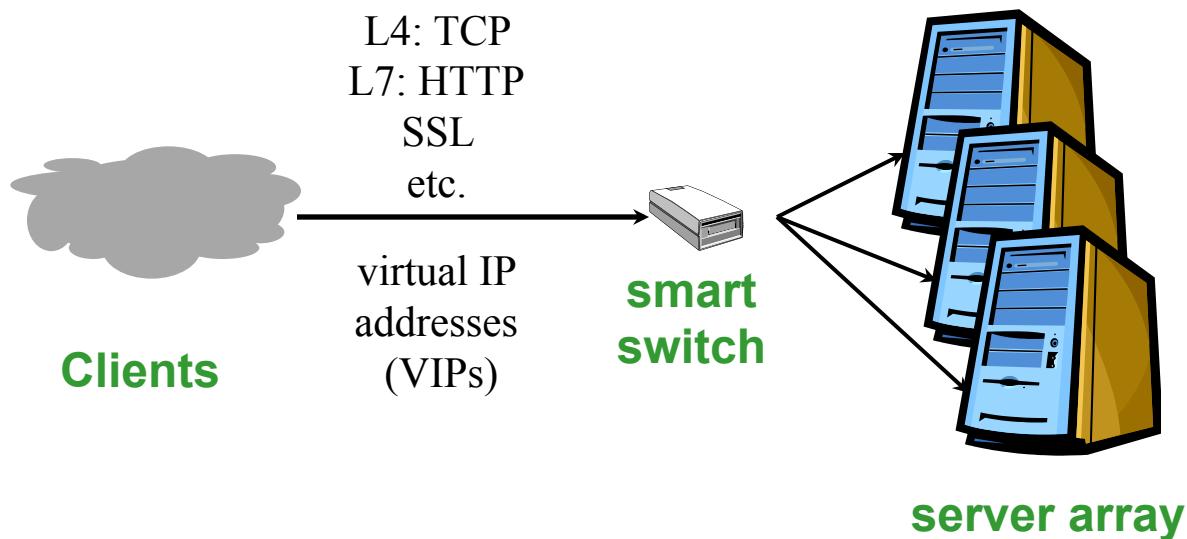
Outline

- Recap
- Multiple network servers
 - Basic issues
 - Load direction
 - DNS
 - Load balancer (smart switch)

Smart Switch: Big Picture



VIP Clustering



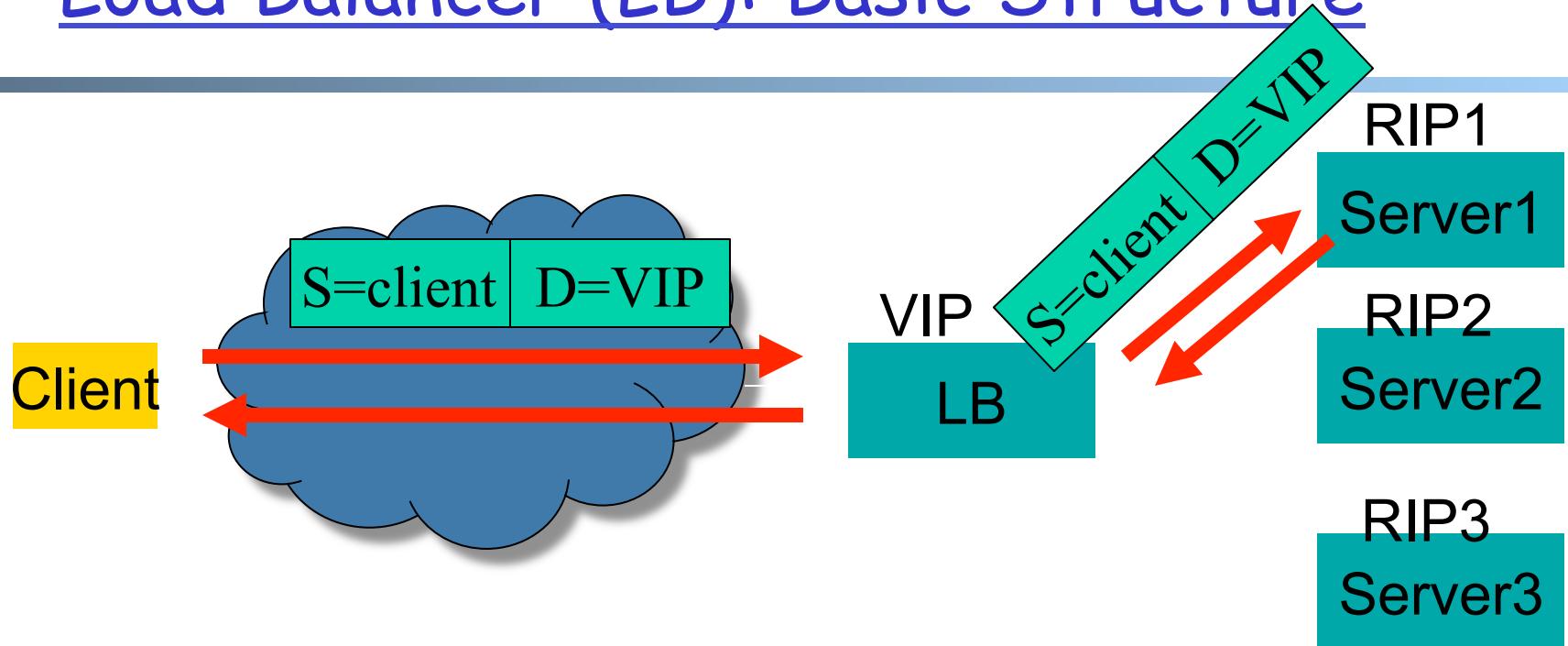
Goals

server load balancing
failure detection
access control/filtering
priorities/QoS
request locality
transparent caching

What to switch/filter on?

L3 source IP and/or VIP
L4 (TCP) ports etc.
L7 URLs and/or cookies
L7 SSL session IDs

Load Balancer (LB): Basic Structure



Problem of the basic structure?

S=client

D=VIP

Problem

- Client to server packet has VIP as destination address, but real servers use RIPv
 - if LB just forwards the packet from client to a real server, the real server drops the packet
 - reply from real server to client has real server IP as source -> client will drop the packet

state: listening
address: {*.**6789**, *:*****}
completed connection queue: C1; C2
sendbuf:
recvbuf:

state: established
address: {128.36.232.5:**6789**, 198.69.10.10.**1500**}
sendbuf:
recvbuf:

state: established
address: {128.36.232.5:**6789**, 198.69.10.10.**1500**}
sendbuf:
recvbuf:

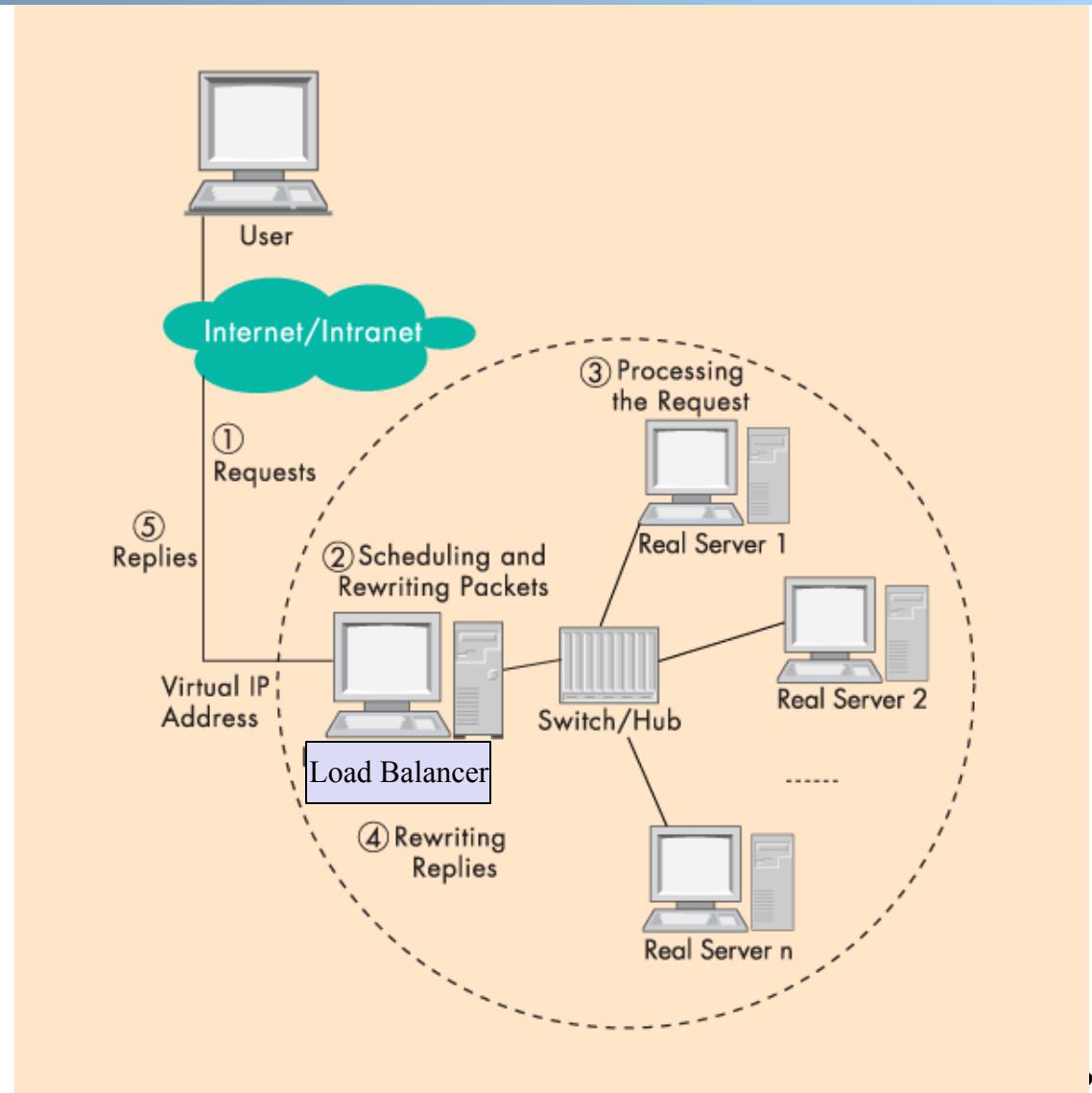
...

...

Real Server TCP socket space

Solution 1: Network Address Translation (NAT)

- ❑ LB does rewriting/translation
- ❑ Thus, the LB is similar to a typical NAT gateway with an additional scheduling function



Example Virtual Server via NAT

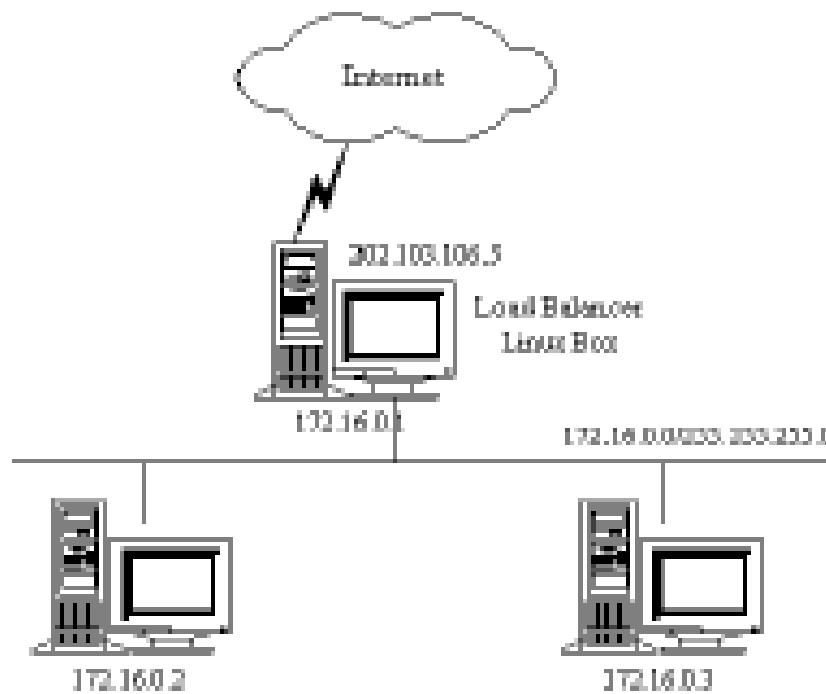
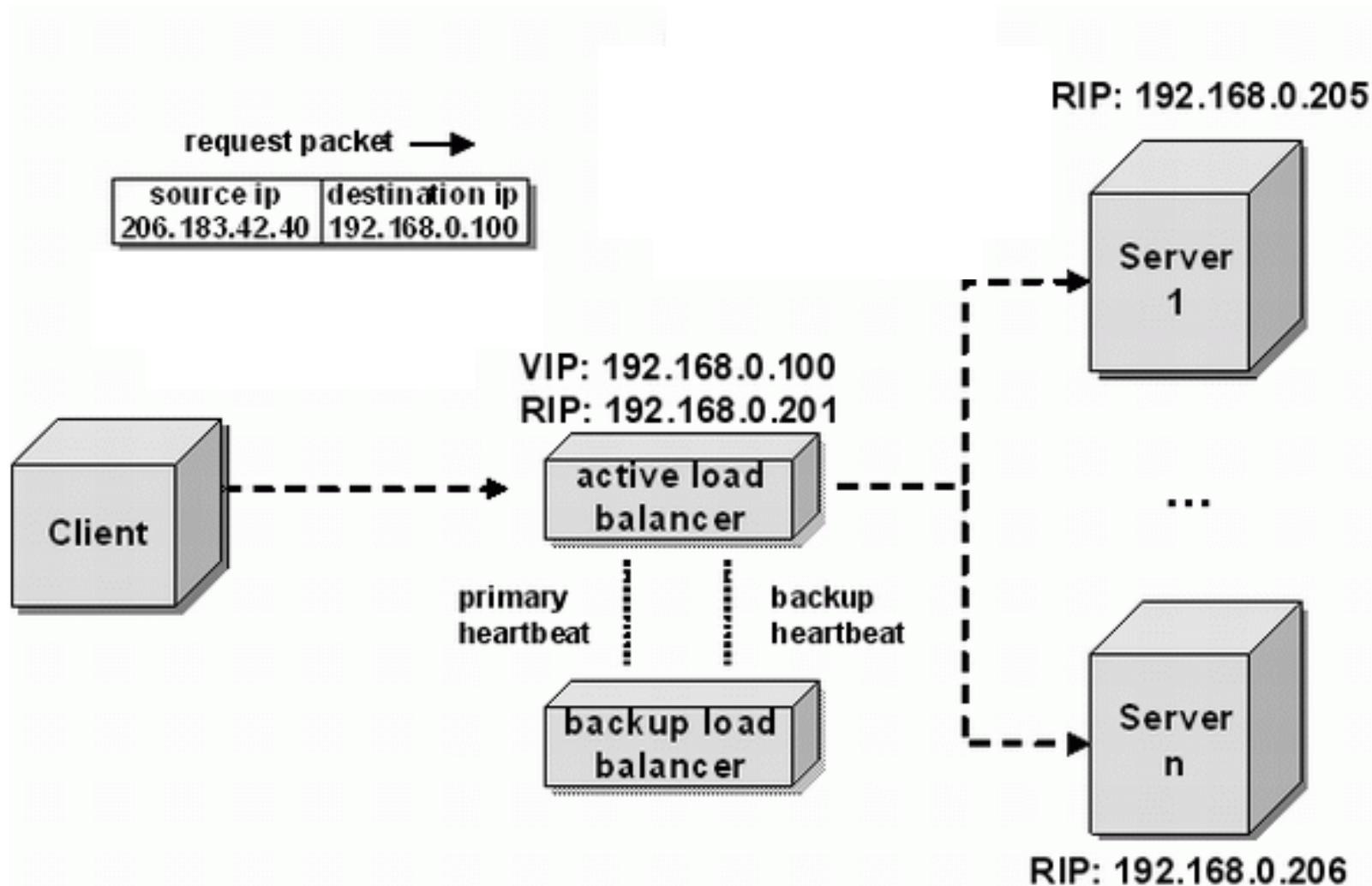


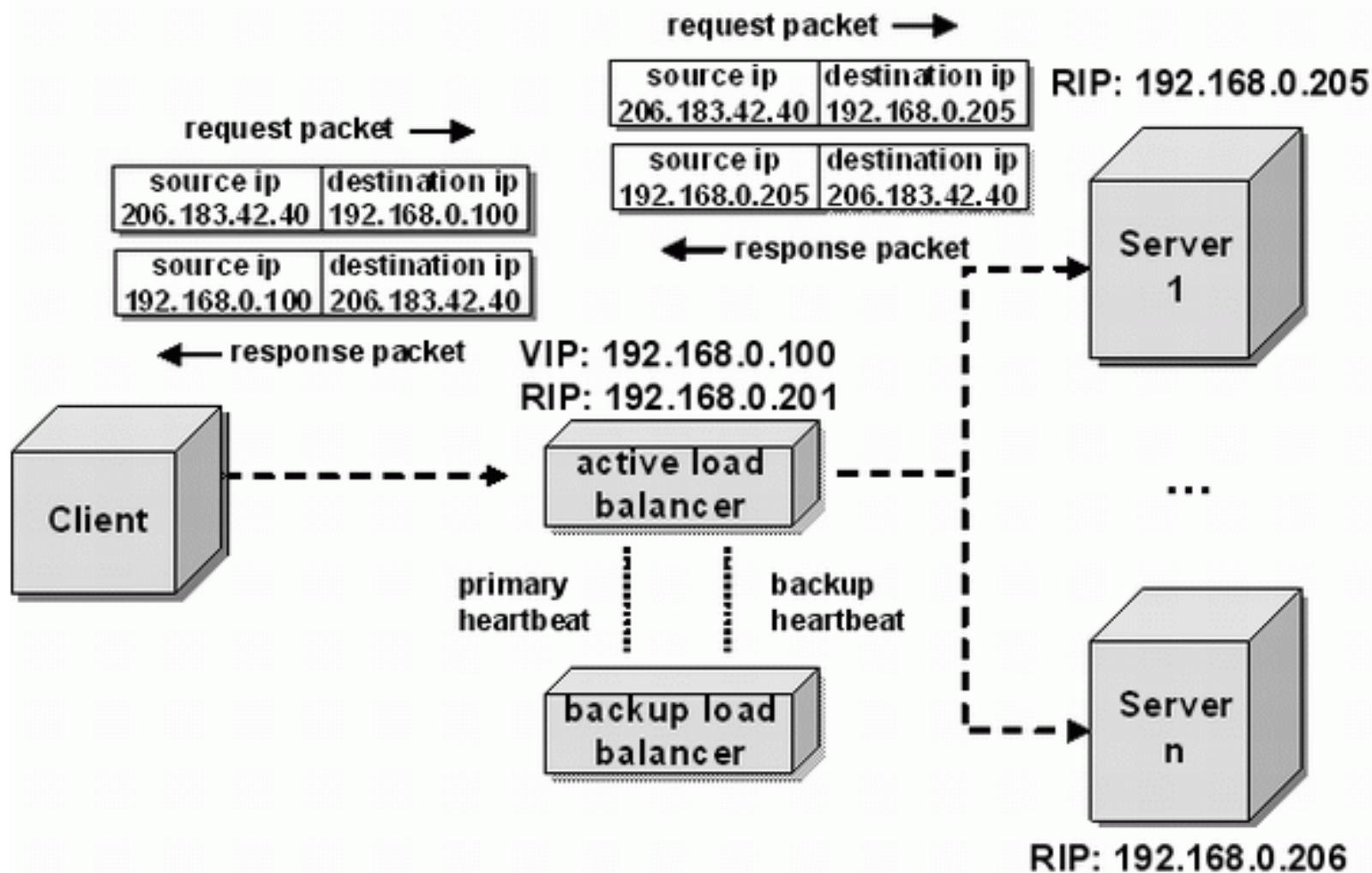
Table 1: an example of virtual server rules

Protocol	Virtual IP Address	Port	Real IP Address	Port	Weight
TCP	202.103.106.5	80	172.16.0.2	80	1
			172.16.0.3	8000	2
TCP	202.103.106.5	21	172.16.0.3	21	1

LB/NAT Flow



LB/NAT Flow



LB/NAT Advantages and Disadvantages

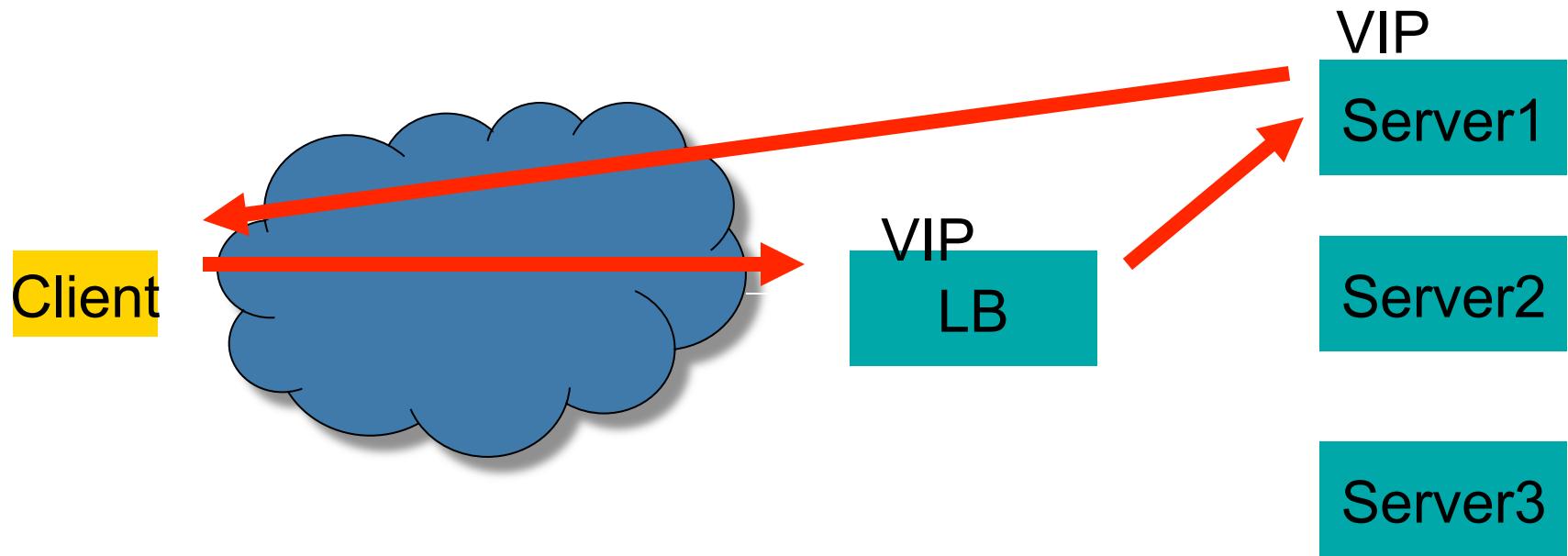
□ Advantages:

- Only one public IP address is needed for the load balancer; real servers can use private IP addresses
- Real servers need no change and are not aware of load balancing

□ Problem

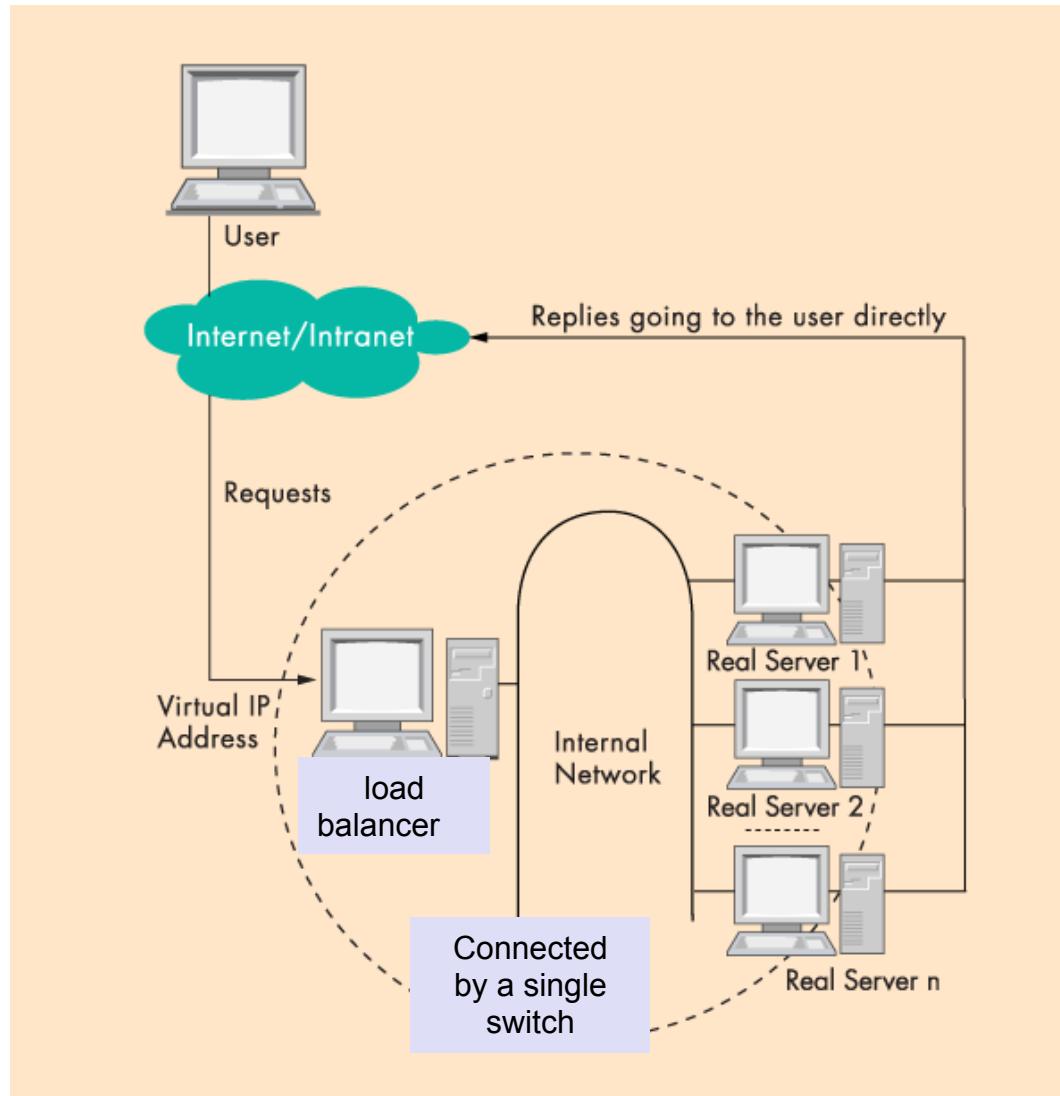
- The load balancer must be on the critical path and hence may become the bottleneck due to load to rewrite request and response packets
 - Typically, rewriting responses has more load because there are more response packets

LB with Direct Reply

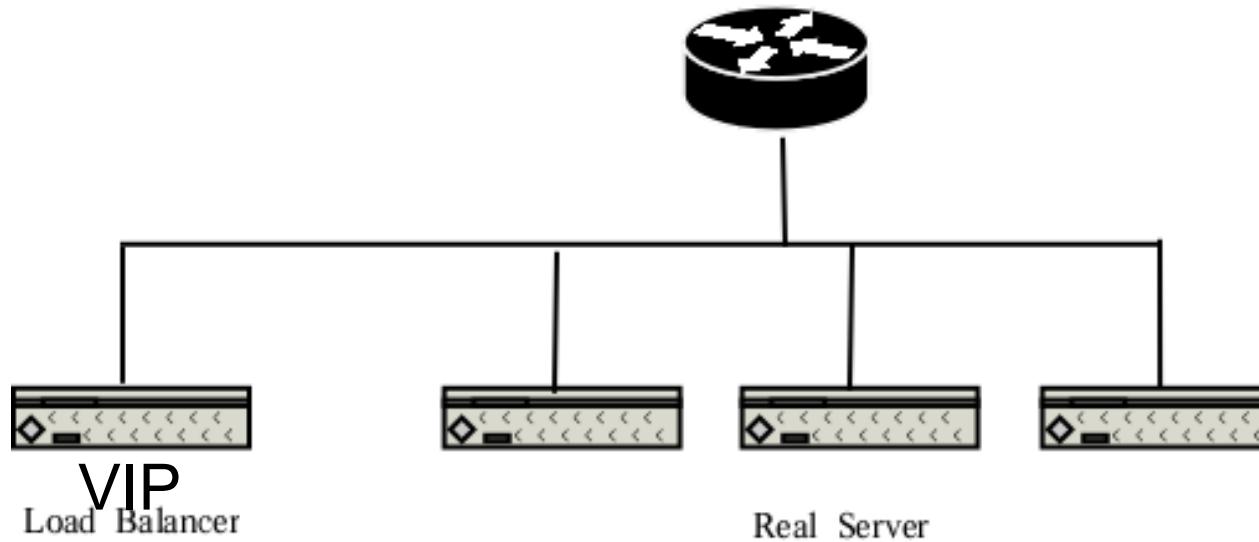


Direct reply → Each real server uses VIP
as its IP address

LB/DR Architecture



Why IP Address Matters?



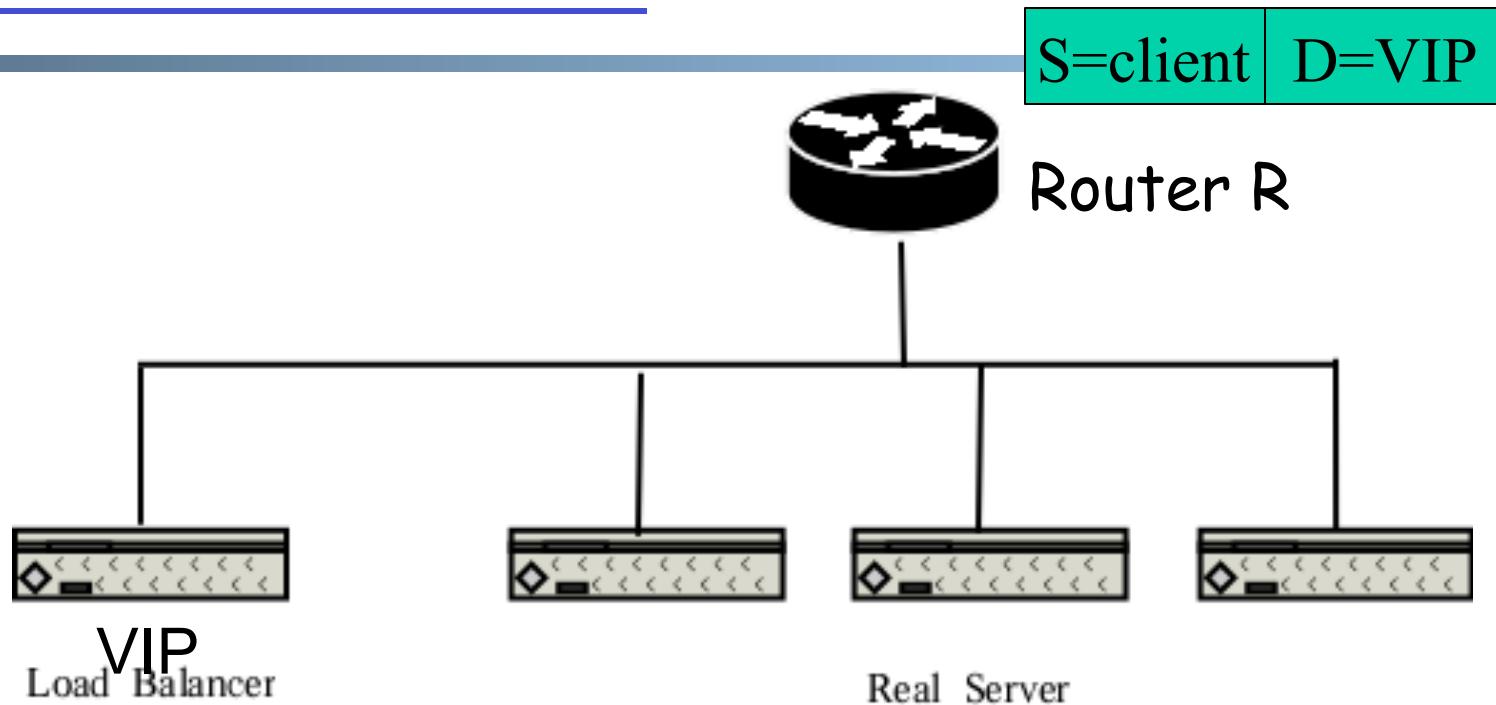
- Each network interface card listens to an assigned MAC address
- A router is configured with the range of IP addresses connected to each interface (NIC)
- To send to a device with a given IP, the router needs to translate IP to MAC (device) address
- The translation is done by the Address Resolution Protocol (ARP)

ARP Protocol

- ARP is “plug-and-play”:
 - nodes create their ARP tables without intervention from net administrator

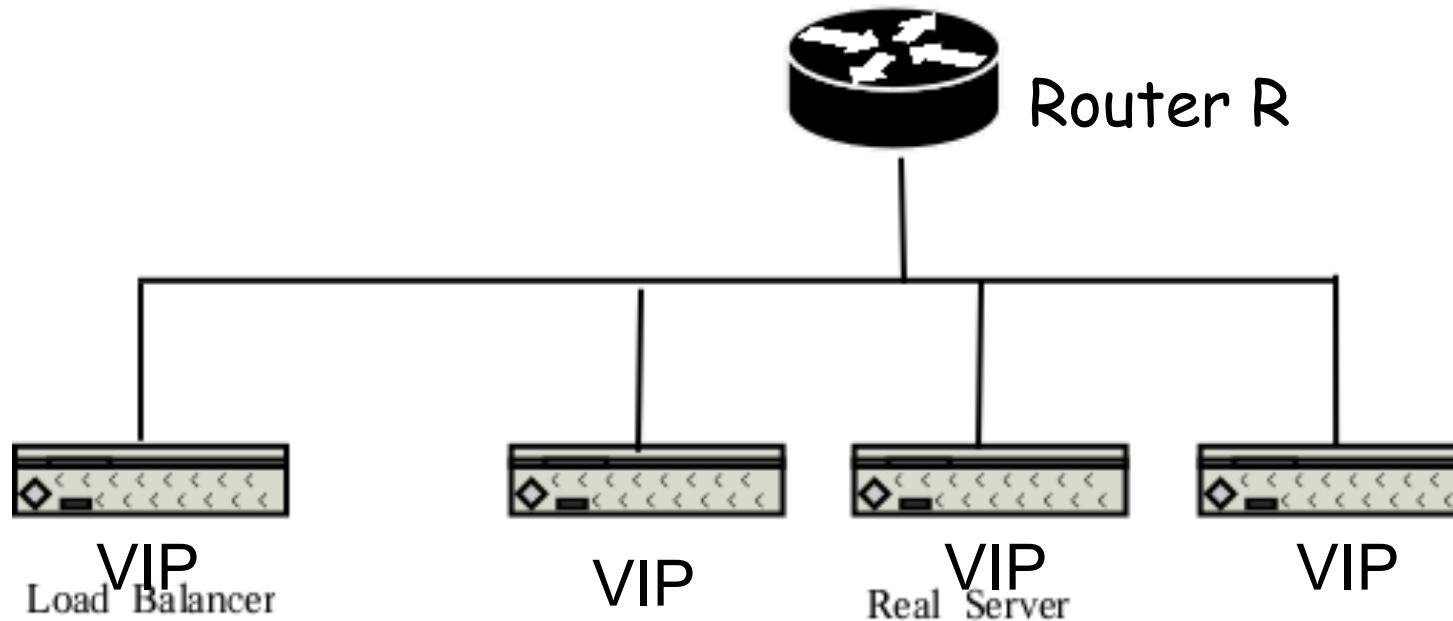
- A **broadcast** protocol:
 - Router broadcasts query frame, containing queried IP address
 - all machines on LAN receive ARP query
 - Node with queried IP receives ARP frame, replies its MAC address

ARP in Action



- Router broadcasts ARP broadcast query: who has VIP?
- ARP reply from LB: I have VIP; my MAC is MAC_{LB}
- Data packet from R to LB: destination MAC = MAC_{LB}

LB/DR Problem



ARP and race condition:

- When router R gets a packet with dest. address VIP, it broadcasts an Address Resolution Protocol (ARP) request: who has VIP?
- One of the real servers may reply before load balancer

Solution: configure real servers to not respond to ARP request

LB via Direct Routing

- The virtual IP address is shared by real servers and the load balancer.
- Each real server has a non-ARPing, loopback alias interface configured with the virtual IP address, and the load balancer has an interface configured with the virtual IP address to accept incoming packets.
- The workflow of LB/DR is similar to that of LB/NAT:
 - the load balancer directly routes a packet to the selected server
 - the load balancer simply changes the MAC address of the data frame to that of the server and retransmits it on the LAN (how to know the real server's MAC?)
 - When the server receives the forwarded packet, the server determines that the packet is for the address on its loopback alias interface, processes the request, and finally returns the result directly to the user

LB/DR Advantages and Disadvantages

□ Advantages:

- Real servers send response packets to clients directly, avoiding LB as bottleneck

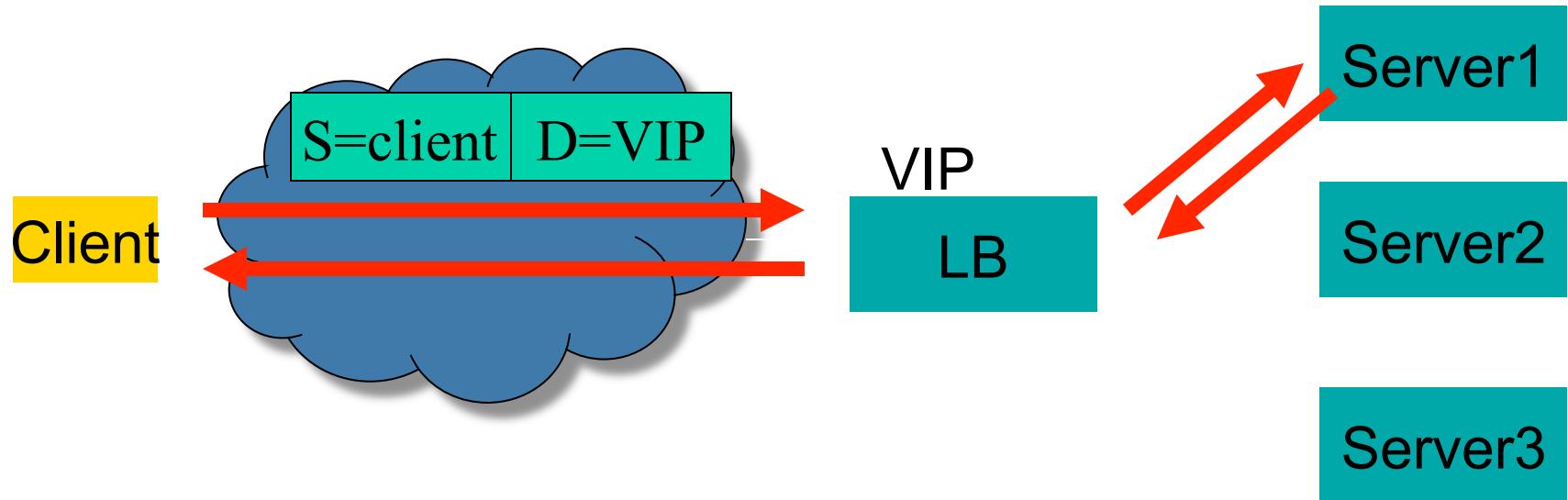
□ Disadvantages:

- Servers must have non-arp alias interface
- The load balancer and server must have one of their interfaces in the same LAN segment

Example Implementation of LB

- An example open source implementation is Linux virtual server (linux-vs.org)
 - Used by
 - www.linux.com
 - sourceforge.net
 - wikipedia.org
 - More details on ARP problem: http://www.austintek.com/LVS/LVS-HOWTO/HOWTO/LVS-HOWTO.arp_problem.html
 - Many commercial LB servers from F5, Cisco, ...
- More details please read chapter 2 of Load Balancing Servers, Firewalls, and Caches

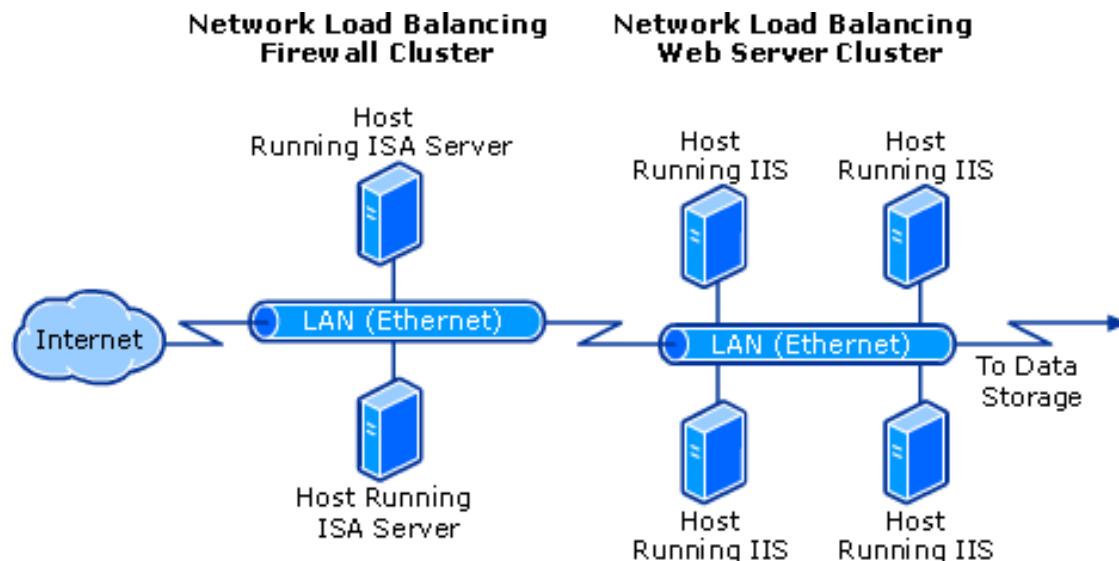
Discussion: Problem of the Load Balancer Architecture



A major remaining problem is that the LB becomes a single point of failure (SPOF).

Solutions

- Redundant load balancers
 - E.g., two load balancers
- Fully distributed load balancing
 - e.g., Microsoft Network Load Balancing (NLB)



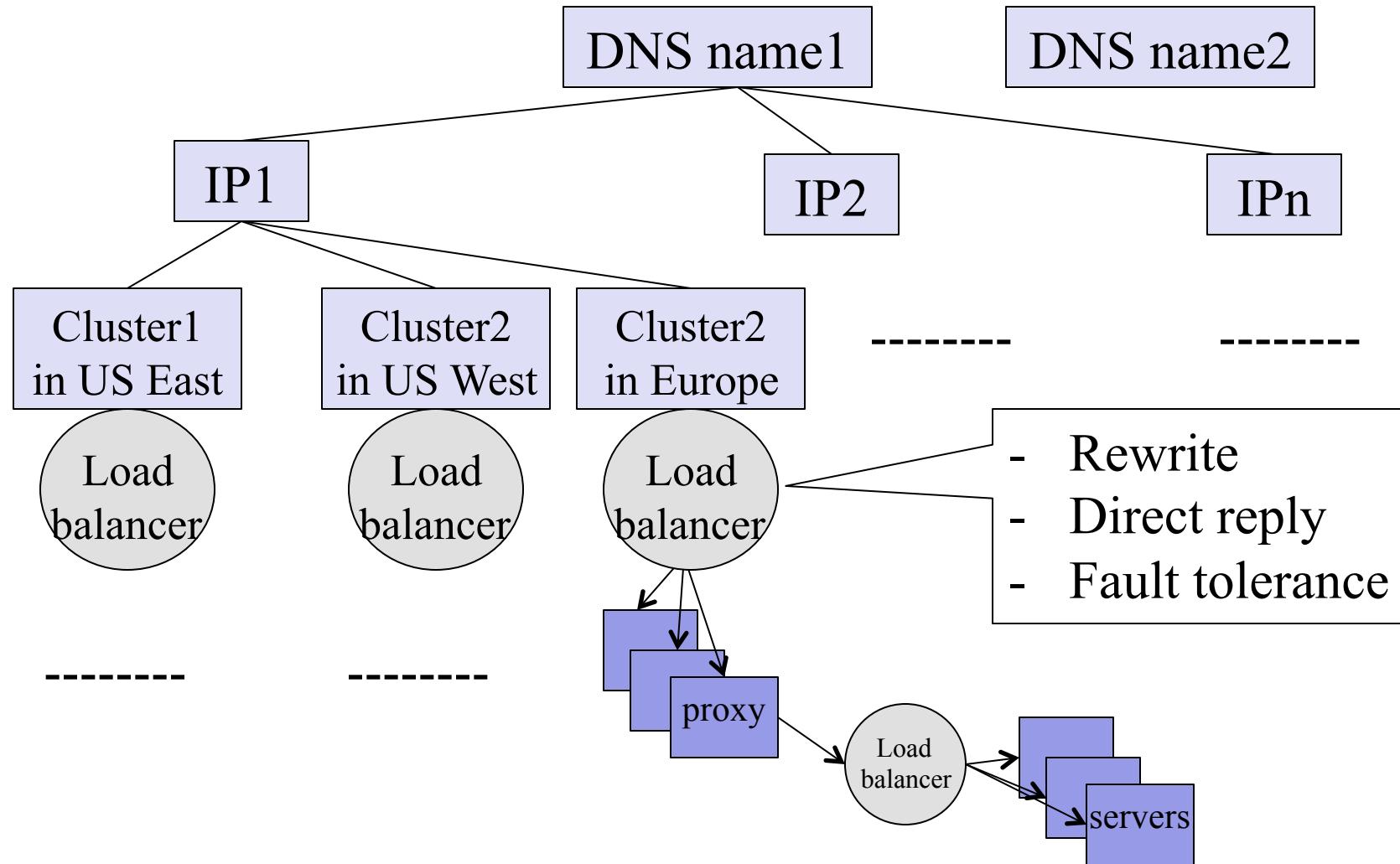
Microsoft NLB

- No dedicated load balancer
- All servers in the cluster receive all packets
- All servers within the cluster simultaneously run a mapping algorithm to determine which server should handle the packet. Those servers not required to service the packet simply discard it.
 - Mapping (ranking) algorithm: computing the “winning” server according to host priorities, multicast or unicast mode, port rules, affinity, load percentage distribution, client IP address, client port number, other internal load information

Discussion

- Compare the design of using Load Balancer vs Microsoft NLB

Summary: Example Direction Mechanisms



Outline

- Admin and recap
- Multiple servers
 - Overview
 - Basic mechanisms
 - Example: YouTube (offline read)

You Tube

- 02/2005: Founded by Chad Hurley, Steve Chen and Jawed Karim, who were all early employees of PayPal.
- 10/2005: First round of funding (\$11.5 M)
- 03/2006: 30 M video views/day
- 07/2006: 100 M video views/day
- 11/2006: acquired by Google
- 10/2009: Chad Hurley announced in a blog that YouTube serving well over 1 B video views/day (avg = 11,574 video views /sec)

Pre-Google Team Size

- 2 Sysadmins
- 2 Scalability software architects
- 2 feature developers
- 2 network engineers
- 1 DBA
- 0 chefs

YouTube Design Alg.

```
while (true)
{
    identify_and_fix_bottlenecks();
    drink();
    sleep();
    notice_new_bottleneck();
}
```

YouTube Major Components

- Web servers
- Video servers
- Thumbnail servers
- Database servers

YouTube: Web Servers

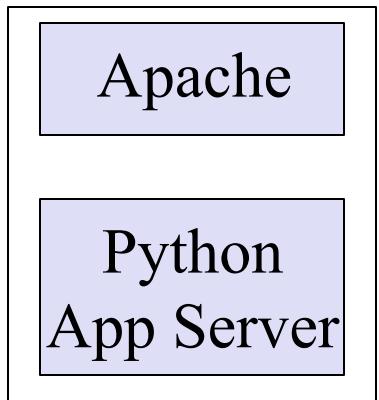
❑ Components

- ❑ Netscaler load balancer; Apache; Python App Servers; Databases

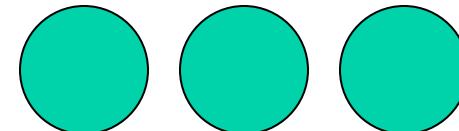
❑ Python

- ❑ Web code (CPU) is not bottleneck
 - ❑ JIT to C to speedup
 - ❑ C extensions
 - ❑ Pre-generate HTML responses
- ❑ Development speed more important

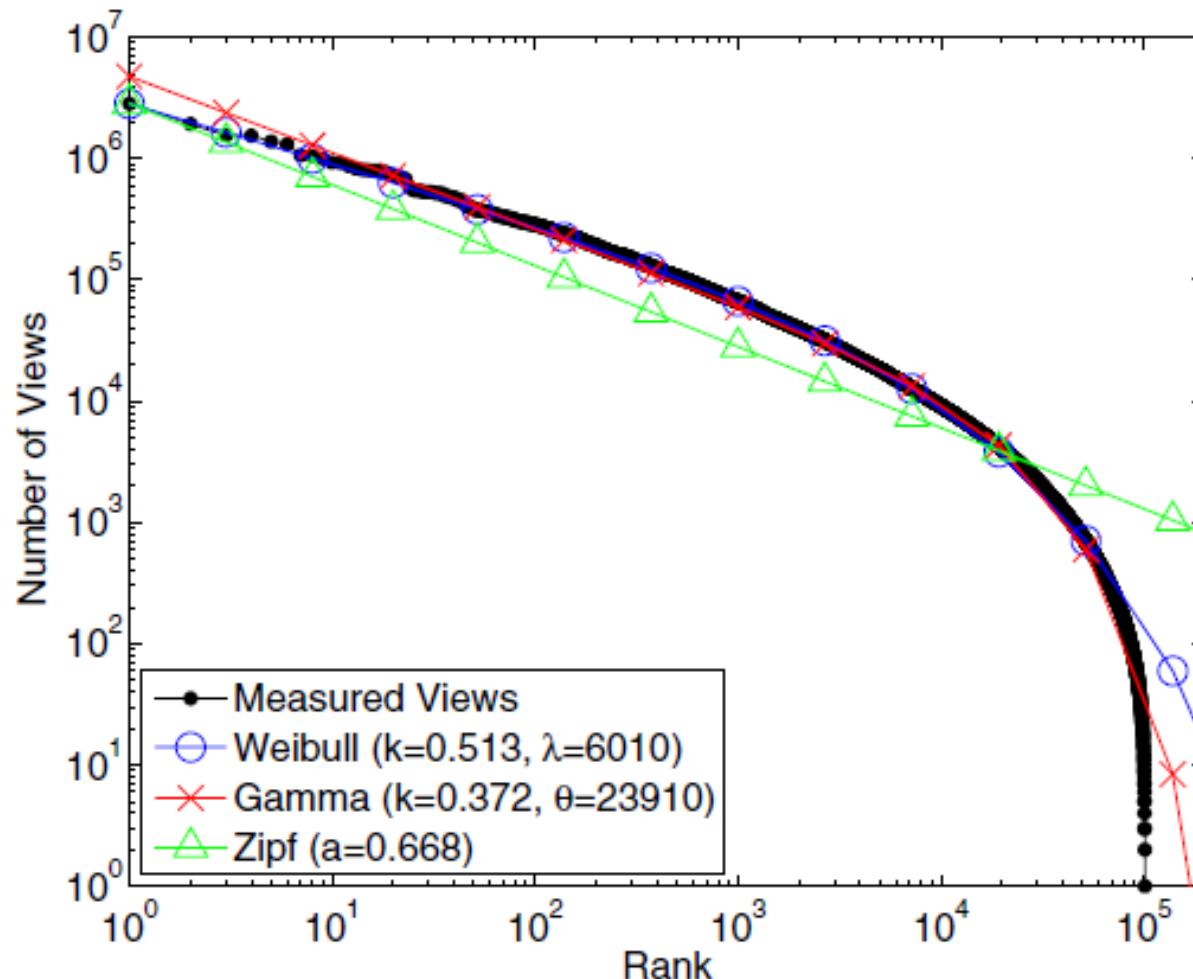
Web servers



Databases



YouTube: Video Popularity



See “Statistics and Social Network of YouTube Videos”, 2008.

YouTube: Video Popularity

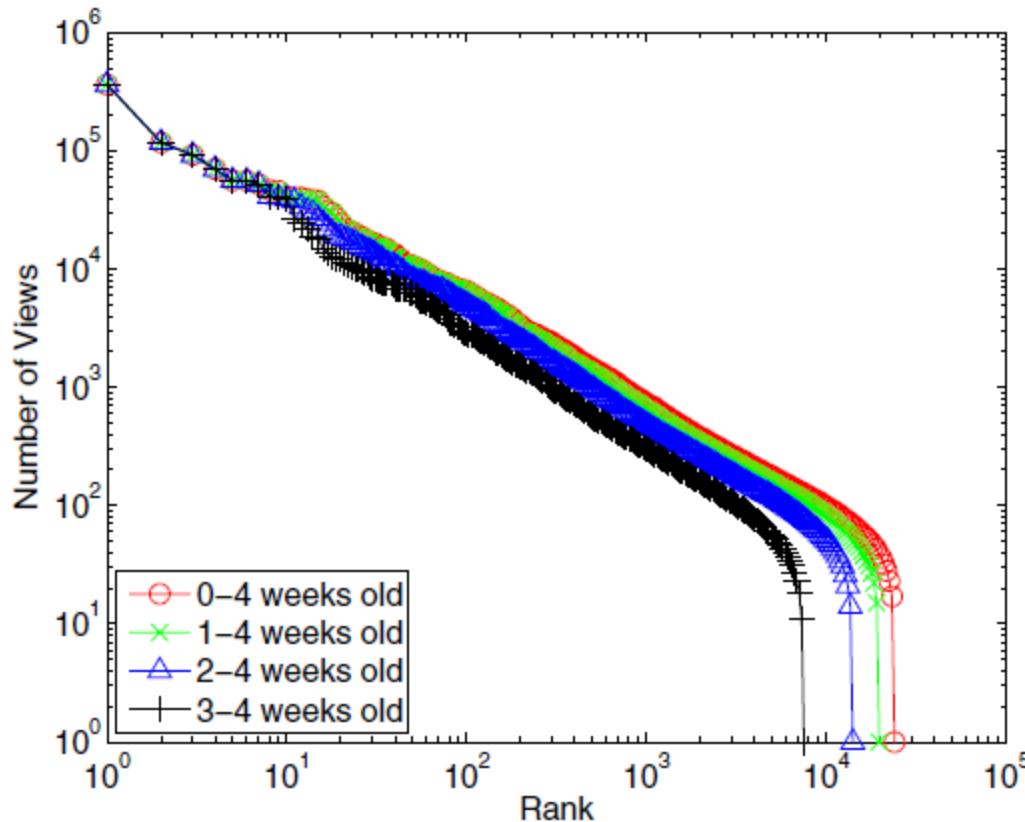


Fig. 8. Recently added YouTube videos rank by popularity

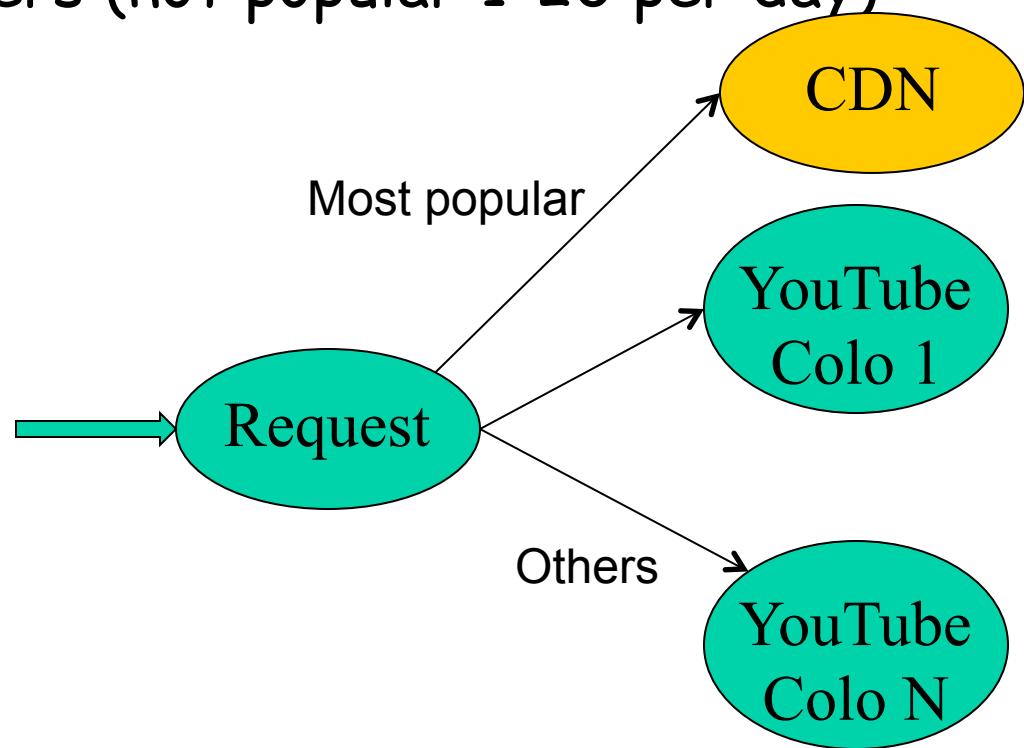
How to design
a system to handle
highly skewed
distribution?

See “Statistics and Social Network of YouTube Videos”, 2008.

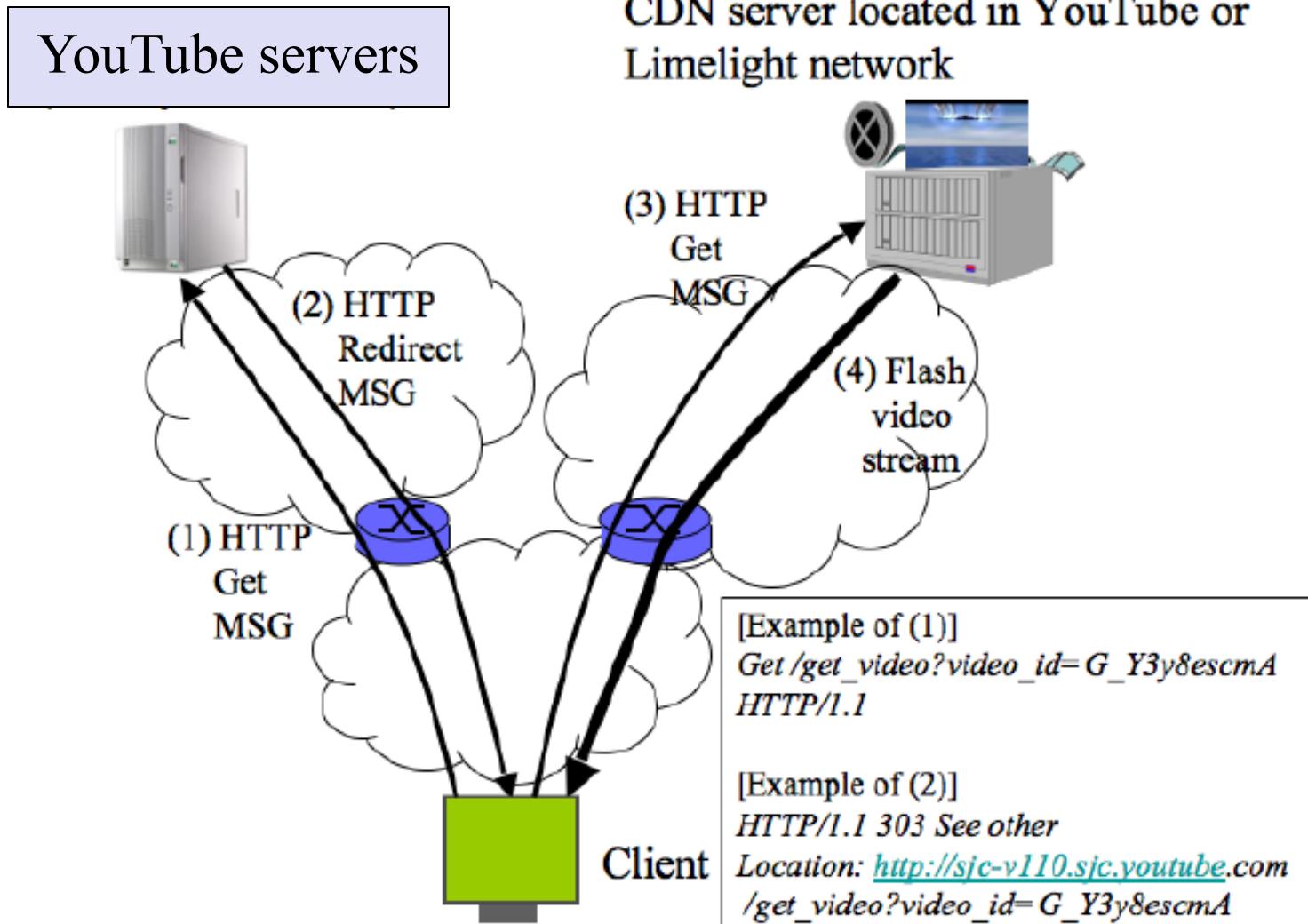
YouTube: Video Server Architecture

□ Tiered architecture

- CDN servers (for popular videos)
 - Low delay; mostly in-memory operation
- YouTube servers (not popular 1-20 per day)



YouTube Redirection Architecture



YouTube Video Servers

- Each video hosted by a mini-cluster consisting of multiple machines
- Video servers use the lighttpd web server for video transmission:
 - Apache had too much overhead (used in the first few months and then dropped)
 - Async io: uses epoll to wait on multiple fds
 - Switched from single process to multiple process configuration to handle more connections

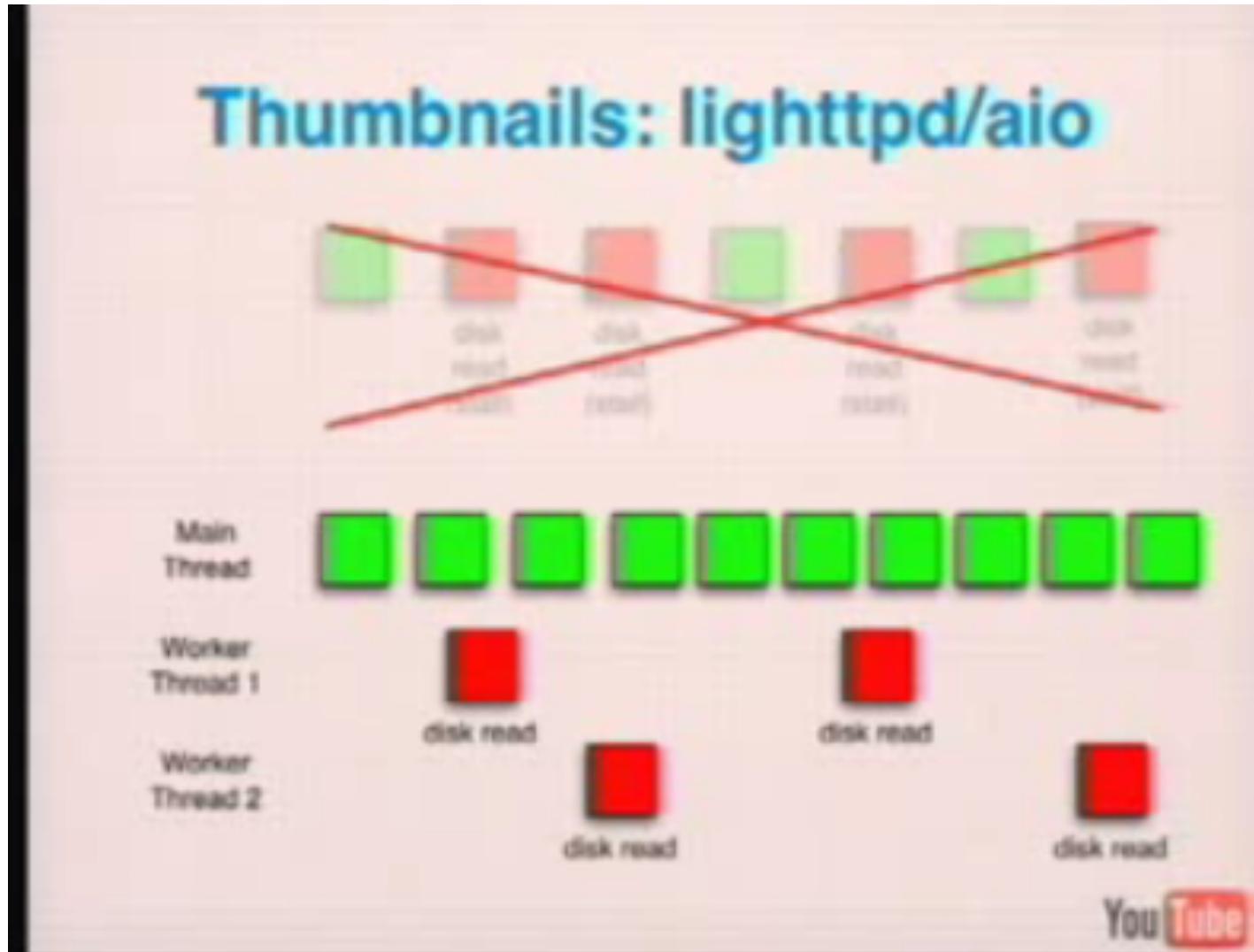
Thumbnail Servers

- Thumbnails are served by a few machines
- Problems running thumbnail servers
 - A high number of requests/sec as web pages can display 60 thumbnails on page
 - Serving a lot of small objects implies
 - lots of disk seeks and problems with file systems inode and page caches
 - may ran into per directory file limit
 - Solution: storage switched to Google BigTable

Thumbnail Server Software Architecture

- Design 1: Squid in front of Apache
 - Problems
 - Squid worked for a while, but as load increased performance eventually decreased: Went from 300 requests/second to 20
 - under high loads Apache performed badly, changed to lighttpd
- Design 2: lighttpd default: By default lighttpd uses a single thread
 - Problem: often stalled due to I/O
- Design 3: switched to multiple processes contending on shared accept
 - Problems: high contention overhead/individual caches

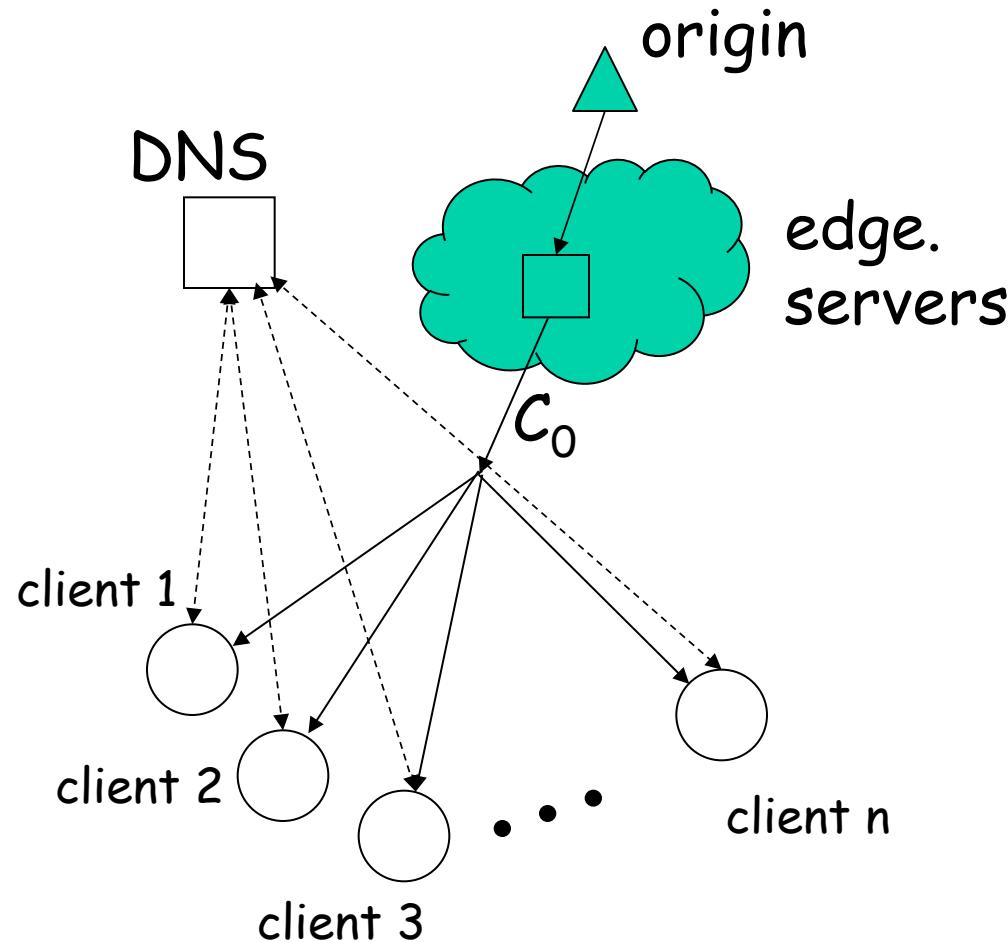
Thumbnails Server: lighttpd/aio



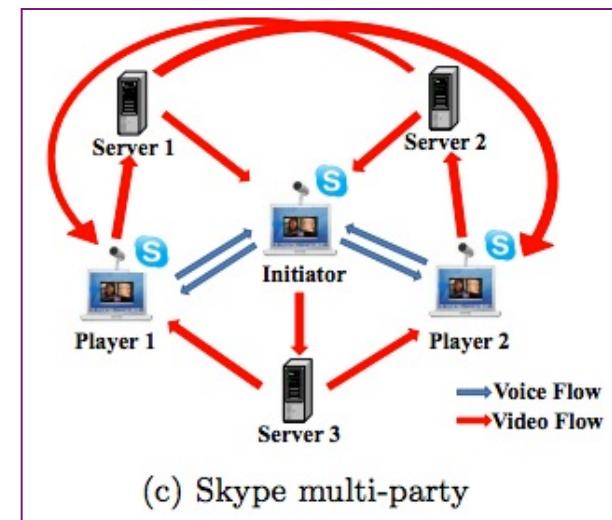
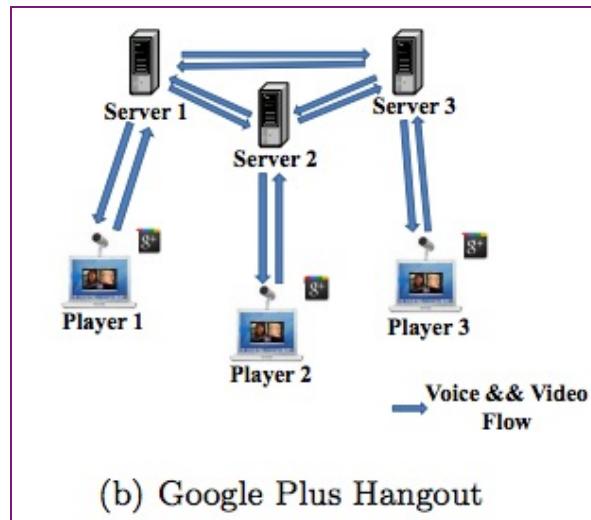
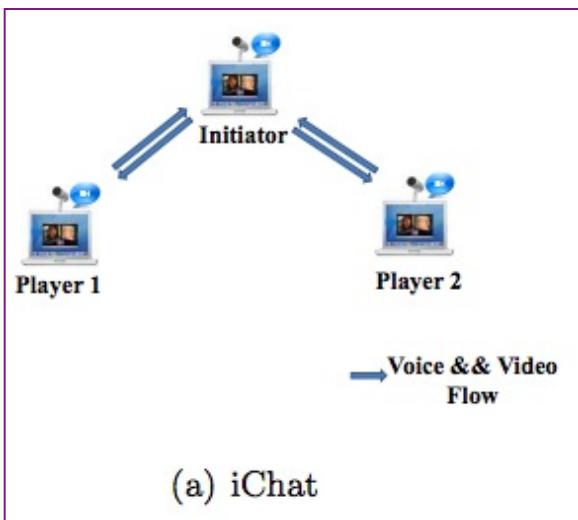
Outline

- Admin and recap
- Multiple servers
- Application overlays

Scalability of Server-Only Approaches



Server + Hosts Systems

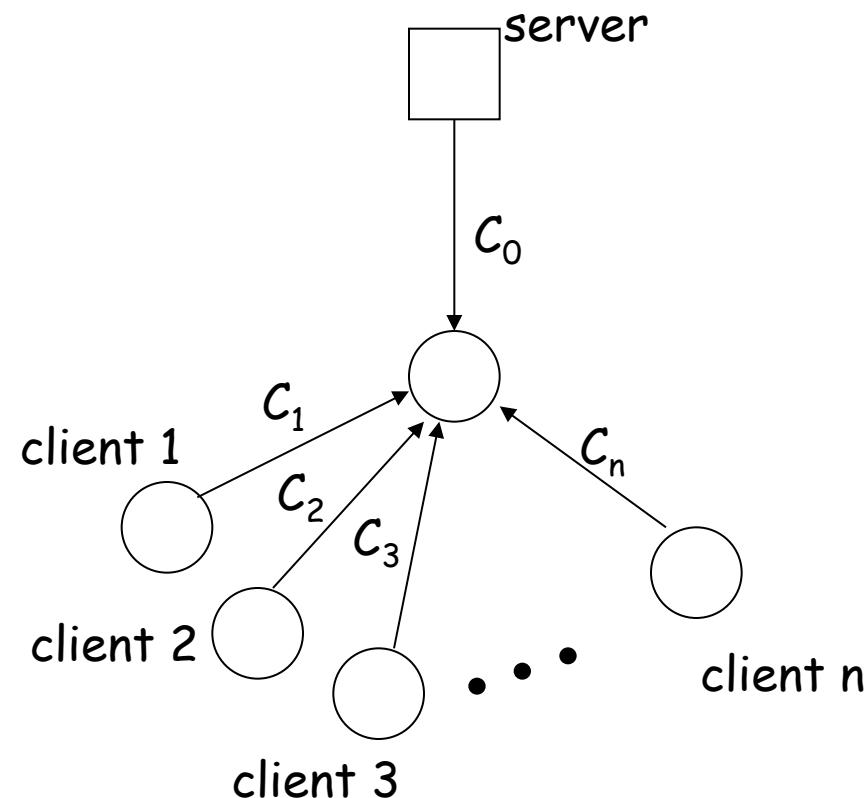


An Upper Bound on Scalability

❑ Assume

- need to achieve same rate to all clients
- only uplinks can be bottlenecks

❑ What is an upper bound on scalability?

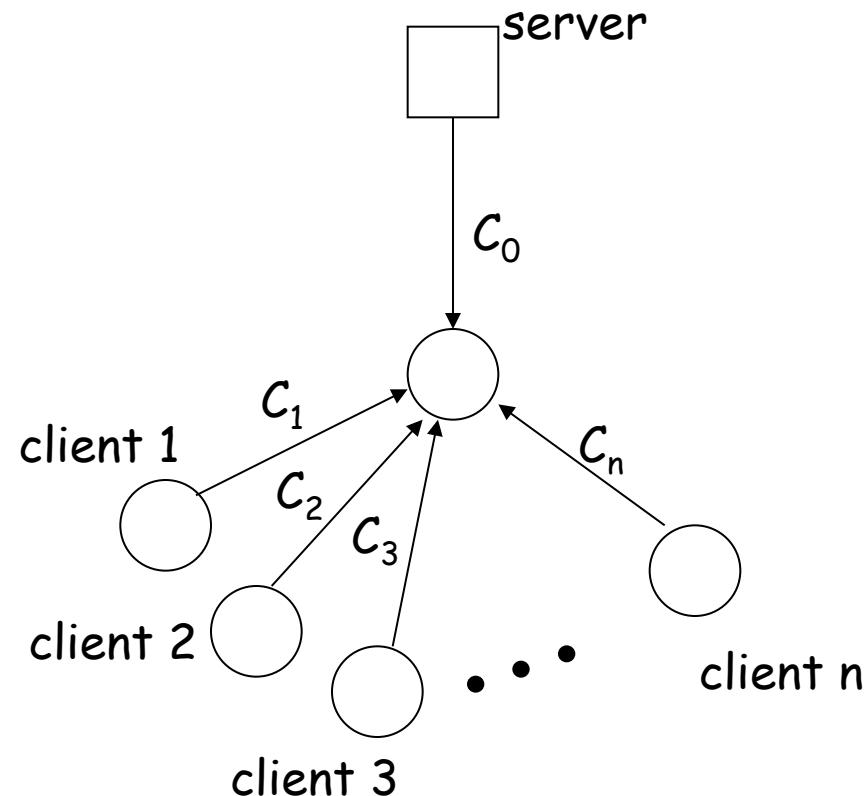


The Scalability Problem

- Maximum throughput

$$R = \min\{C_0, (C_0 + \sum C_i)/n\}$$

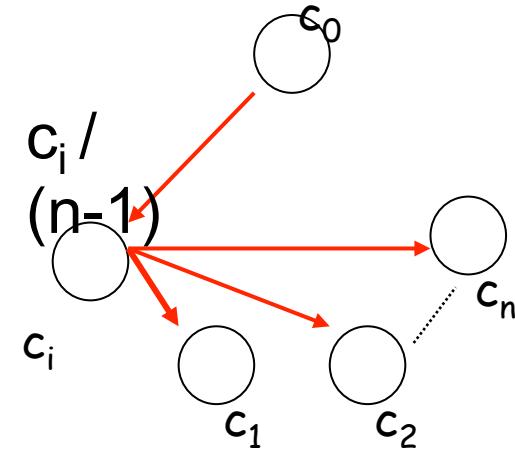
- The bound is theoretically approachable



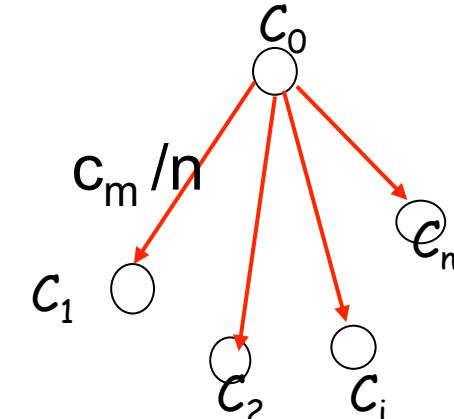
Theoretical Capacity: upload is bottleneck

$$R = \min\{C_0, (C_0 + \sum C_i)/n\}$$

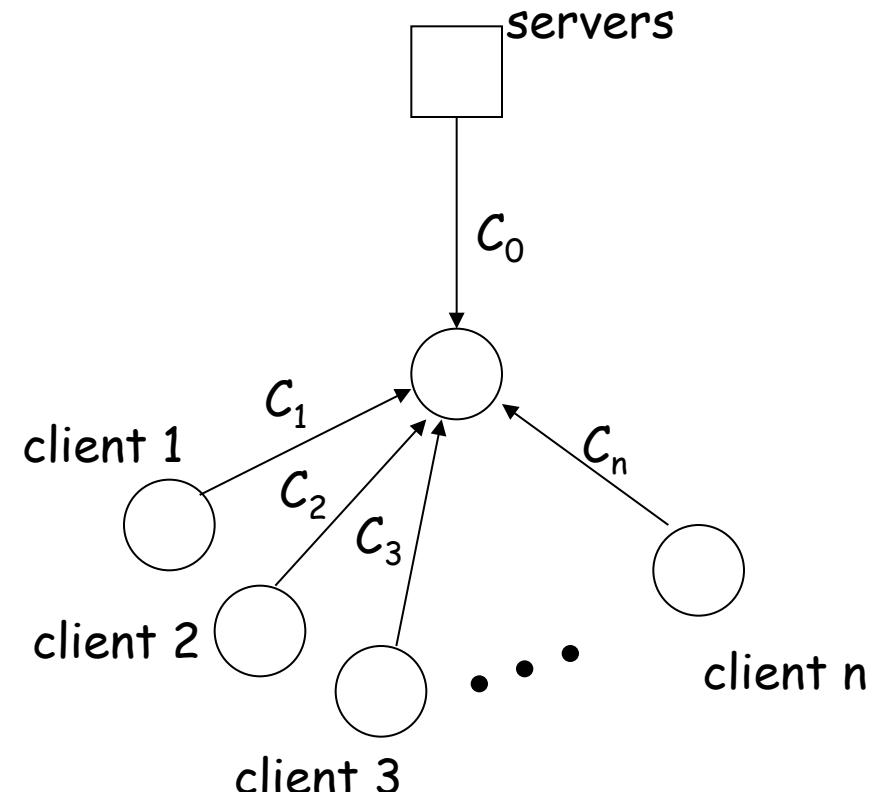
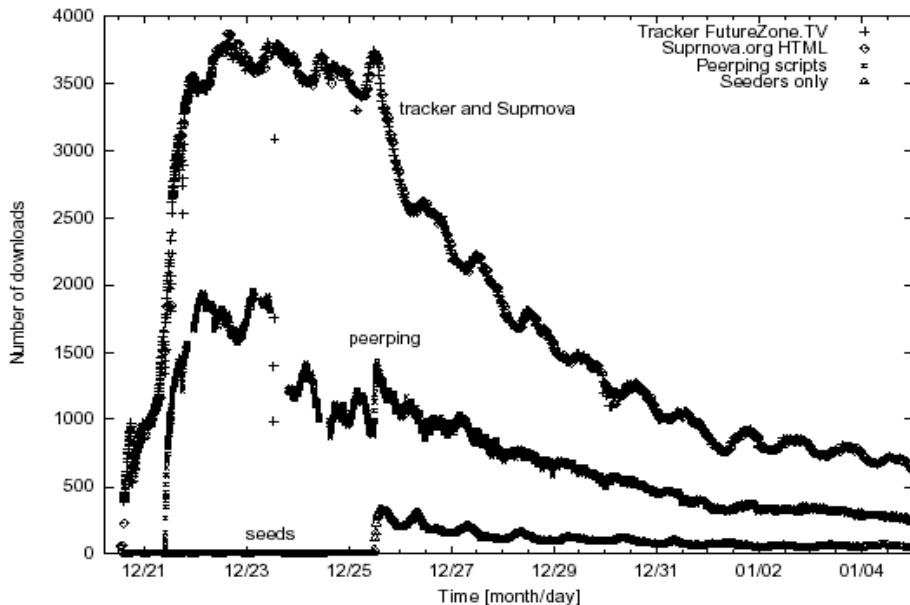
- Assume $C_0 > (C_0 + \sum C_i)/n$
- Tree i:
 - server \rightarrow client i: $c_i/(n-1)$
 - client i \rightarrow other $n-1$ clients



- Tree 0:
 - server has remaining $C_m = c_0 - (c_1 + c_2 + \dots + c_n)/(n-1)$
 - send to client i: C_m/n



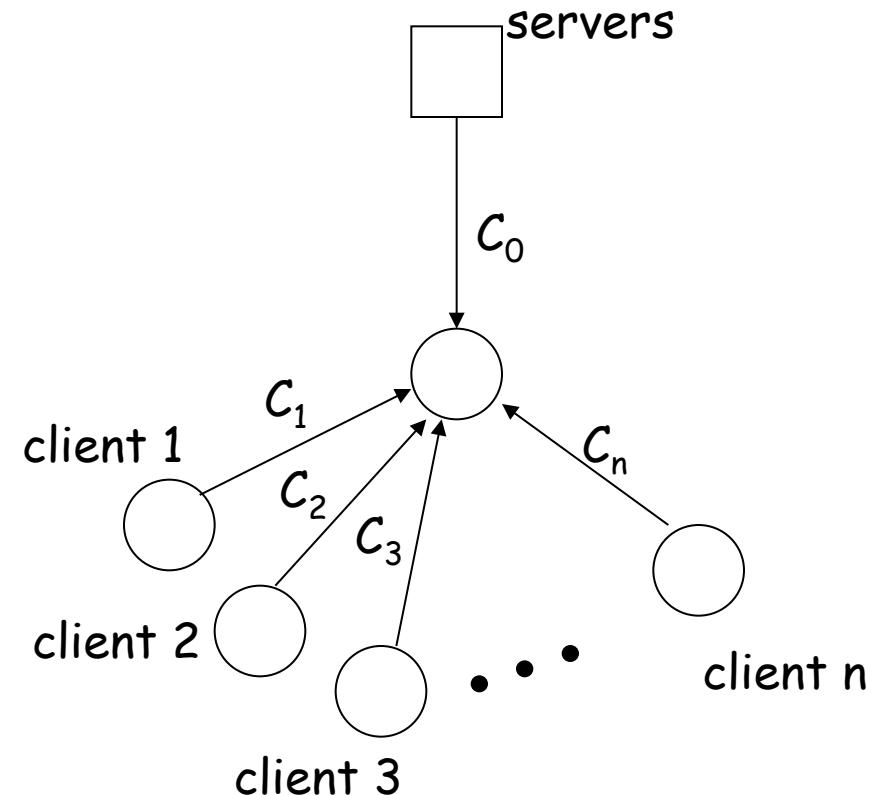
Why not Building the Trees?



- ❑ Clients come and go (churns): maintaining the trees is too expensive
- ❑ Each client needs N connections

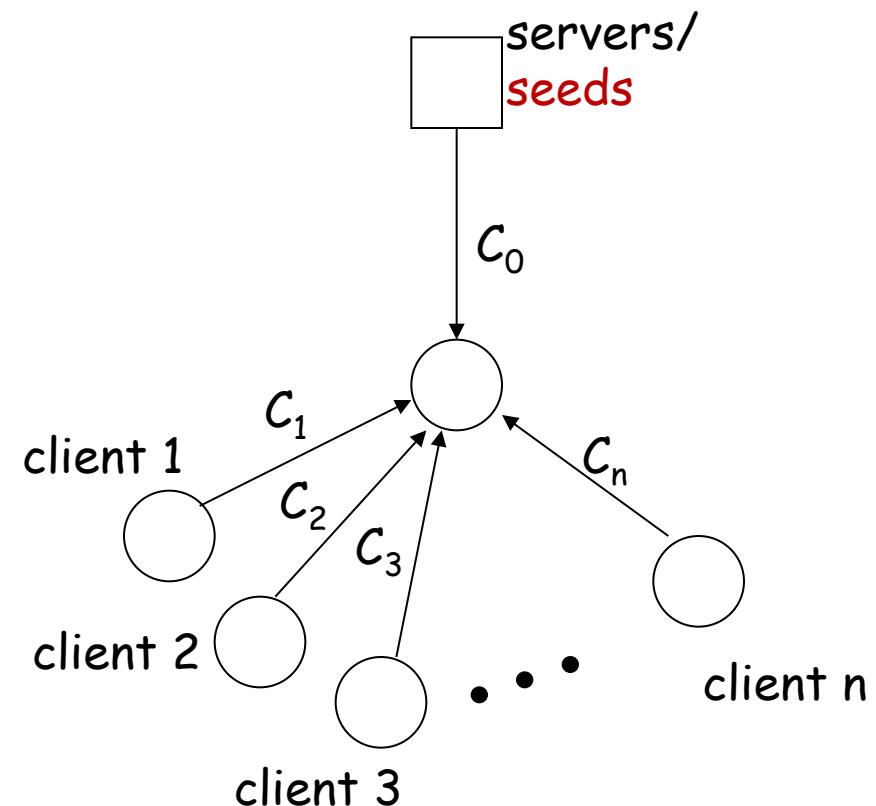
Server+Host (P2P) Content Distribution: Key Design Issues

- Robustness
 - Resistant to churns and failures
- Efficiency
 - A client has content that others need; otherwise, its upload capacity may not be utilized
- Incentive: clients are willing to upload
 - Some real systems nearly 50% of all responses are returned by the top 1% of sharing hosts



Discussion: How to handle the issues?

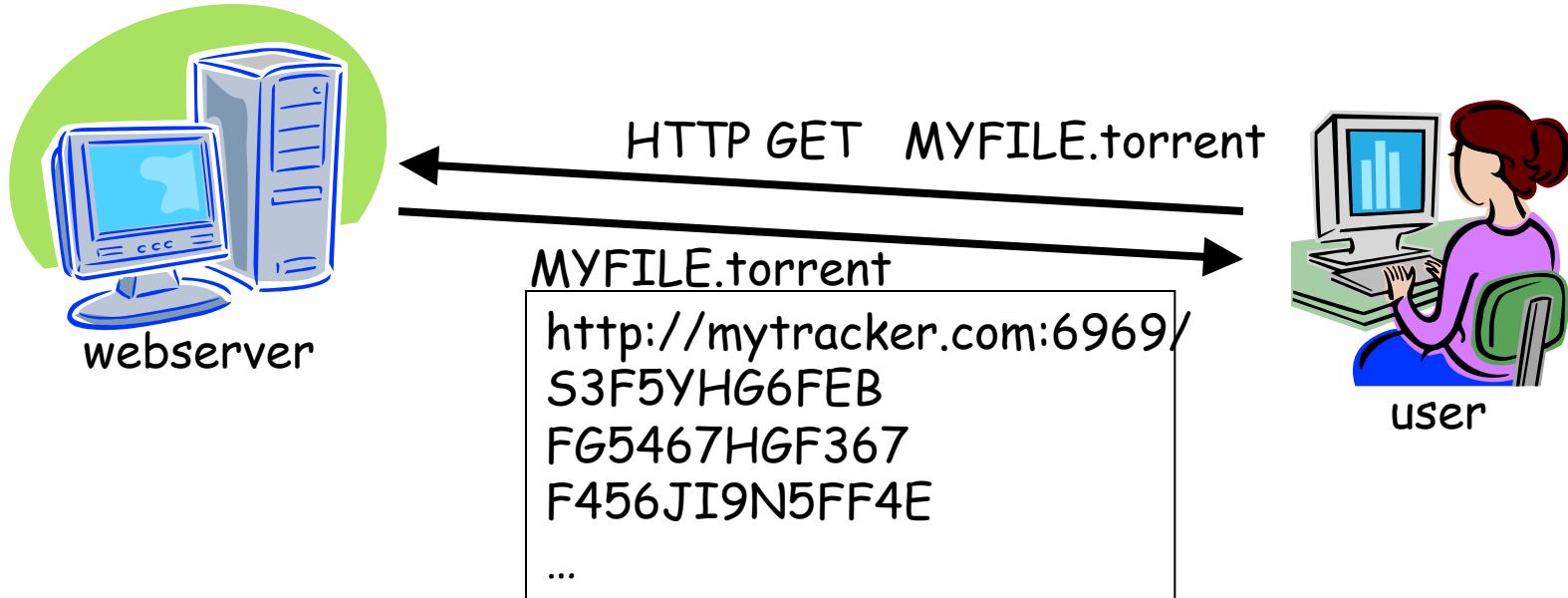
- Robustness
- Efficiency
- Incentive



Example: BitTorrent

- A P2P file sharing protocol
- Created by Bram Cohen in 2004
 - Spec at bep_0003: http://www.bittorrent.org/beps/bep_0003.html

BitTorrent: Lookup



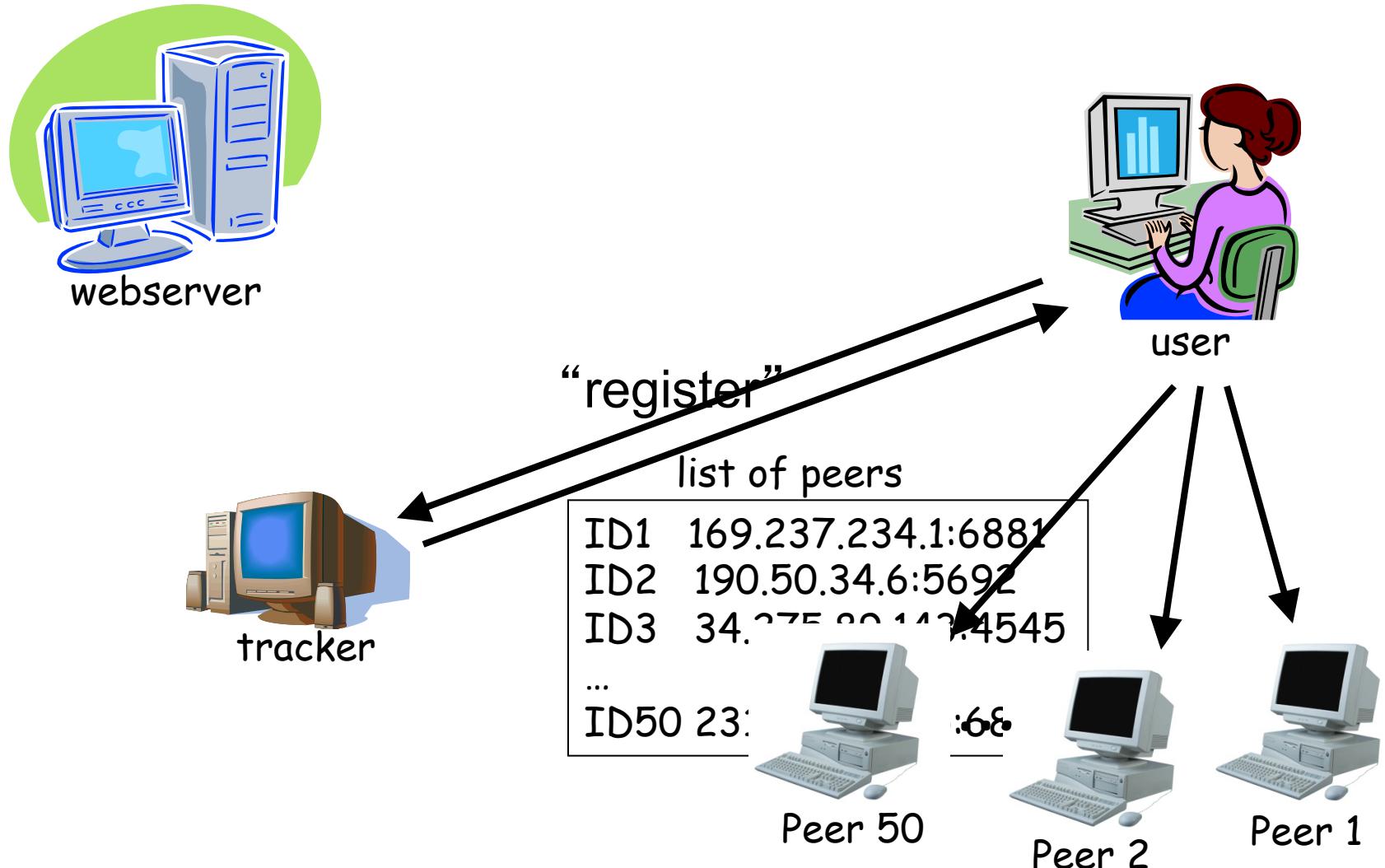
Metadata (.torrent) File Structure

- Meta info contains information necessary to contact the tracker and describes the files in the torrent
 - URL of tracker
 - file name
 - file length
 - piece length (typically 256KB)
 - SHA-1 hashes of pieces for verification
 - also creation date, comment, creator, ...

Tracker Protocol

- Communicates with clients via HTTP/HTTPS
- Client GET request
 - info_hash: uniquely identifies the file
 - peer_id: chosen by and uniquely identifies the client
 - client IP and port
 - numwant: how many peers to return (defaults to 50)
 - stats: e.g., bytes uploaded, downloaded
- Tracker GET response
 - interval: how often to contact the tracker
 - list of peers, containing peer id, IP and port
 - stats

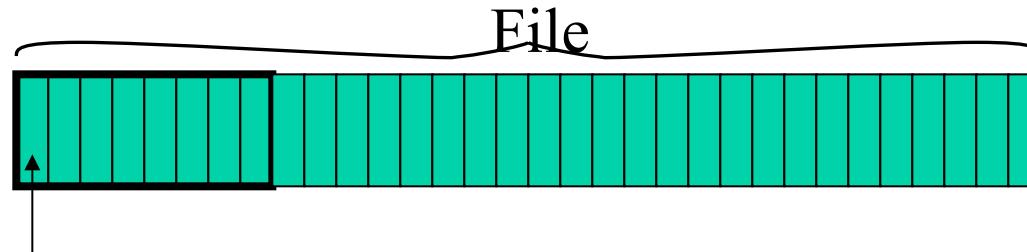
Tracker Protocol



Robustness and efficiency: Piece-based Swarming

- Divide a large file into small blocks and request block-size content from different peers

Block: unit of download

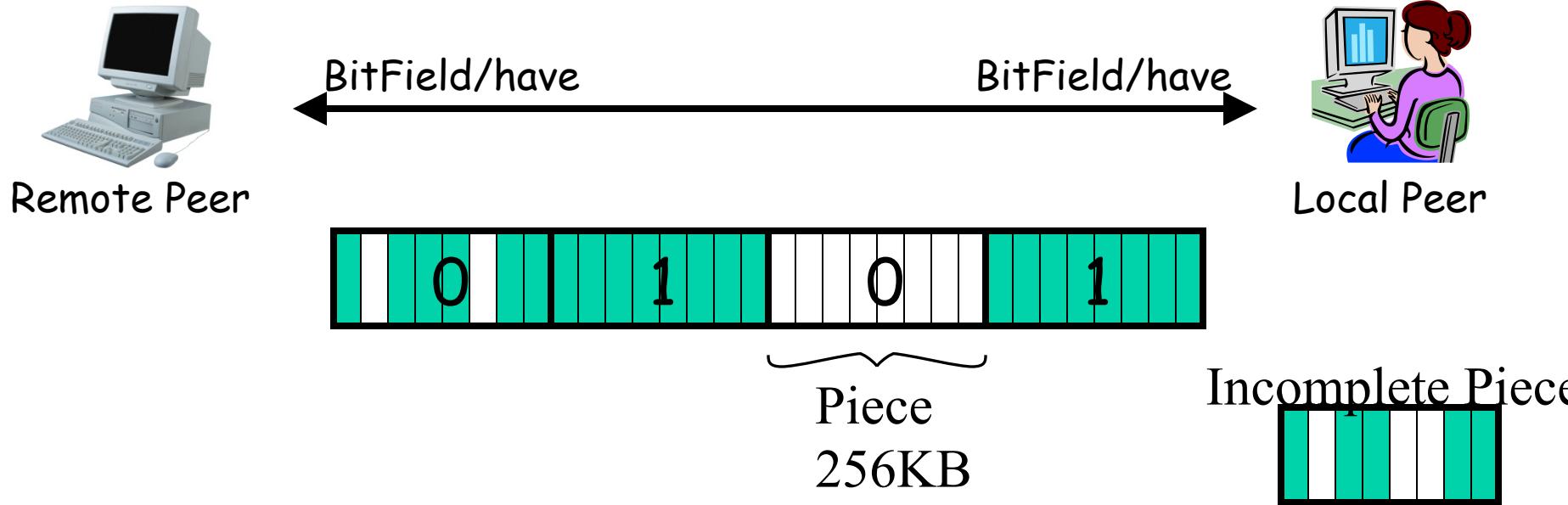


Block: 16KB

- If do not finish downloading a block from one peer within timeout (say due to churns), switch to requesting the block from another peer

Detail: Peer Protocol

(Over TCP)



- Peers exchange bitmap representing content availability
 - bitfield msg during initial connection
 - have msg to notify updates to bitmap
 - to reduce bitmap size, aggregate multiple blocks as a piece

Peer Request

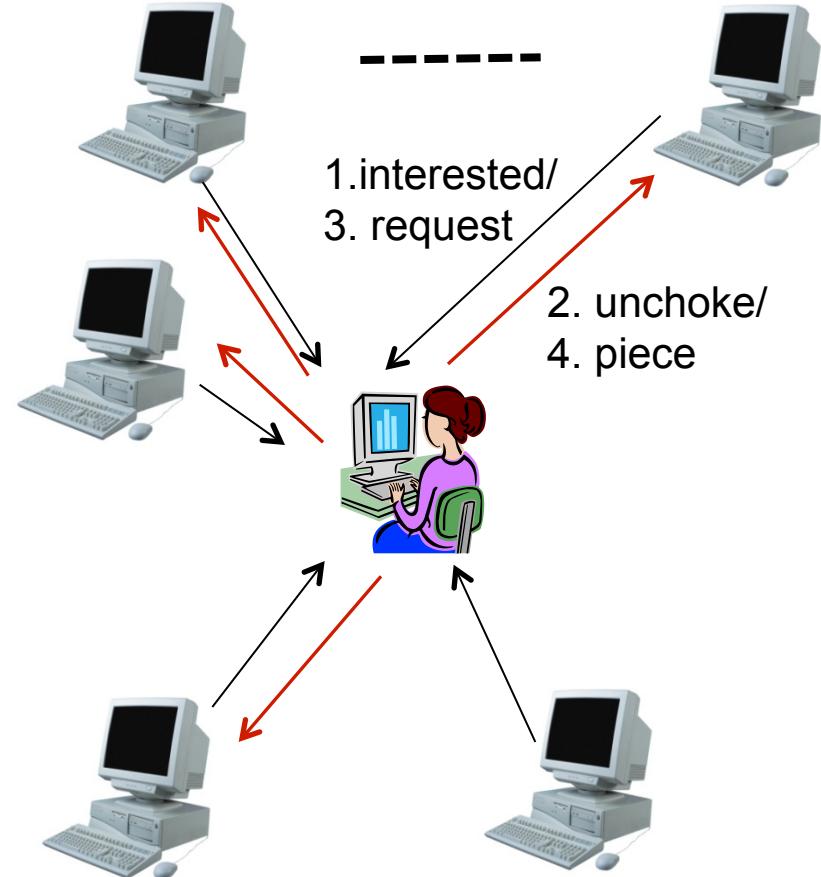
http://www.bittorrent.org/beps/bep_0003.html

- If peer A has a piece that peer B needs, peer B sends interested to A

- unchoke: indicate that A allows B to request

- request: B requests a specific block from A

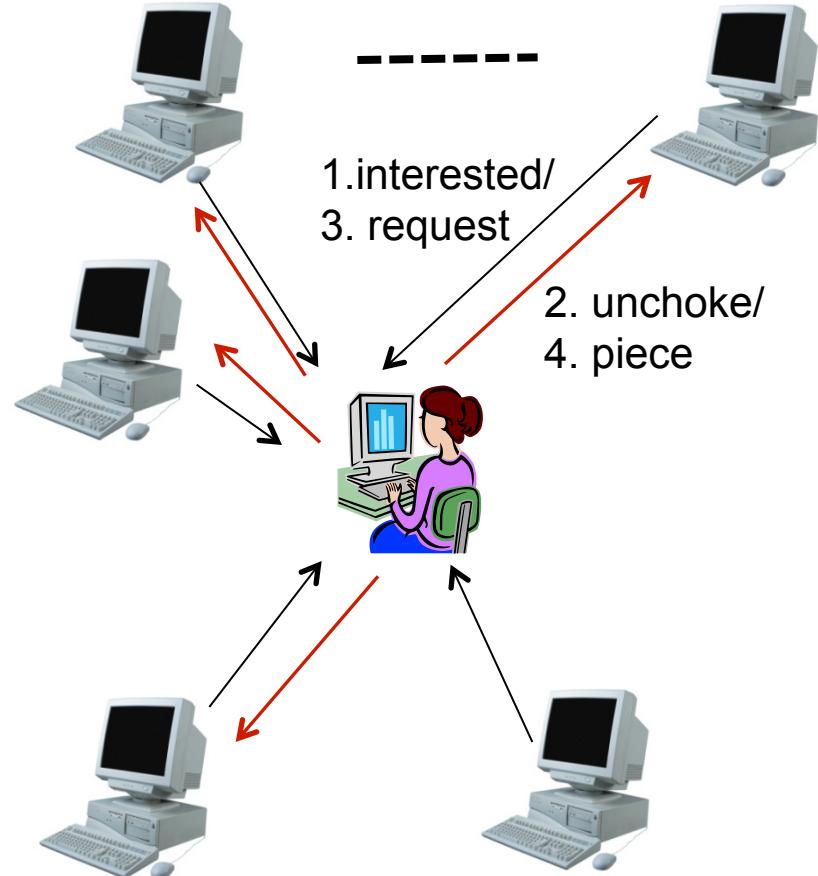
- piece: specific data



Key Design Points

- request:
 - which data blocks to request?

- unchoke:
 - which peers to serve?



Request: Block Availability

- Request (local) **rarest first**
 - achieves the fastest replication of rare pieces
 - obtain something of value

Block Availability: Revisions

- When downloading starts (first 4 pieces): choose at random and request them from the peers
 - get pieces as quickly as possible
 - obtain something to offer to others
- Endgame mode
 - defense against the “last-block problem”: cannot finish because missing a few last pieces
 - send requests for missing pieces to all peers in our peer list
 - send cancel messages upon receipt of a piece

BitTorrent: Unchoke

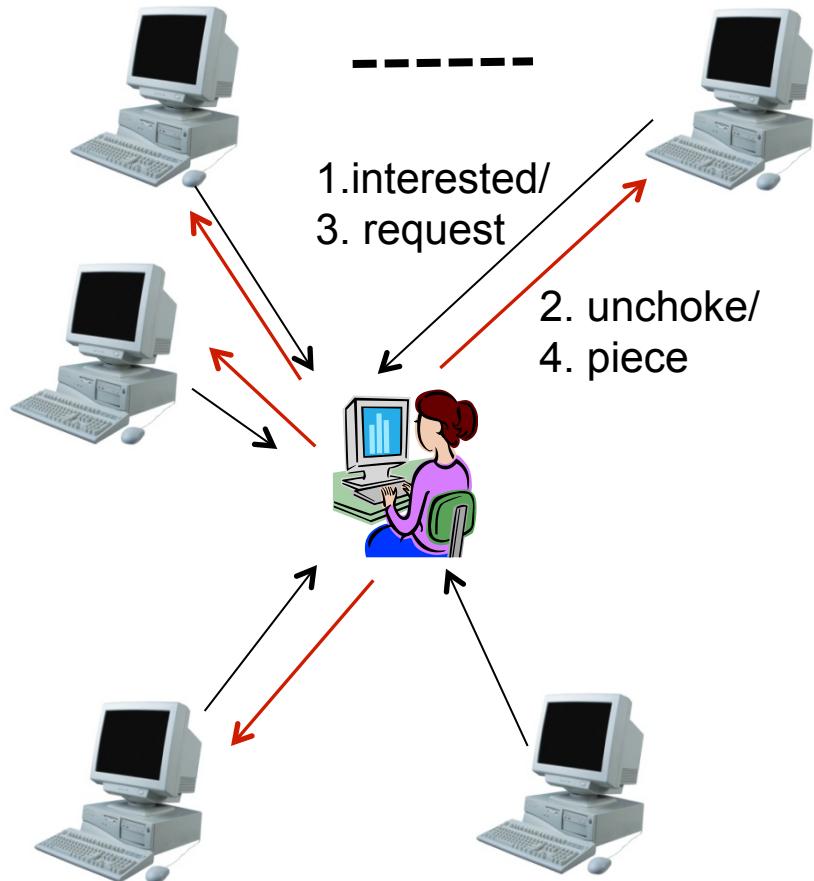
- ❑ Periodically (typically every 10 seconds) calculate data-receiving rates from all peers

- ❑ Upload to (*unchoke*) the fastest

- constant number (4) of unchoking slots

- partition upload bw equally among unchoked

commonly referred to as “**tit-for-tat**” strategy



Optimistic Unchoking

- Periodically select a peer at random and upload to it
 - typically every 3 unchoking rounds (30 seconds)

- Multi-purpose mechanism
 - allow bootstrapping of new clients
 - continuously look for the fastest peers (exploitation vs exploration)

BitTorrent Fluid Analysis

- Normalize file size to 1
- $x(t)$: number of downloaders (also known as leechers) who do not have all pieces at time t .
- $y(t)$: number of seeds in the system at time t .
- λ : the arrival rate of new requests.
- μ : the uploading bandwidth of a given peer.
- c : the downloading bandwidth of a given peer, assume $c \geq \mu$.
- θ : the rate at which downloaders abort download.
- γ : the rate at which seeds leave the system.
- η : indicates the effectiveness of downloader sharing, η takes values in $[0, 1]$.

System Evolution

$$\begin{aligned}\frac{dx}{dt} &= \lambda - \theta x(t) - \min\{cx(t), \mu(\eta x(t) + y(t))\}, \\ \frac{dy}{dt} &= \min\{cx(t), \mu(\eta x(t) + y(t))\} - \gamma y(t),\end{aligned}$$

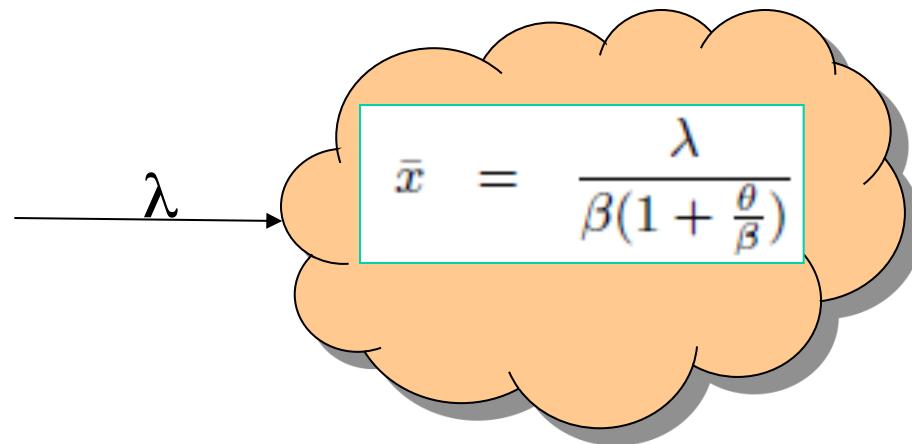
Solving steady state: $\frac{dx(t)}{dt} = \frac{dy(t)}{dt} = 0$

Define $\frac{1}{\beta} = \max\left\{\frac{1}{c}, \frac{1}{\eta}\left(\frac{1}{\mu} - \frac{1}{\gamma}\right)\right\}$

$$\boxed{\begin{aligned}\bar{x} &= \frac{\lambda}{\beta(1 + \frac{\theta}{\beta})} \\ \bar{y} &= \frac{\lambda}{\gamma(1 + \frac{\theta}{\beta})}.\end{aligned}}$$

System State

Q: How long does each downloader stay as a downloader?



$$T = \frac{1}{\theta + \beta}$$

$$\frac{1}{\beta} = \max\left\{\frac{1}{c}, \frac{1}{\eta}\left(\frac{1}{\mu} - \frac{1}{\gamma}\right)\right\}$$

Outline

- Admin and recap
- Multiple servers
- Application overlays
 - Overlays for scalability
 - Overlays for anonymity

Problem

- Internet surveillance like traffic analysis reveals users privacy
- Encryption does not hide packet headers, which still reveal a great deal about users
- End-to-end anonymity is needed

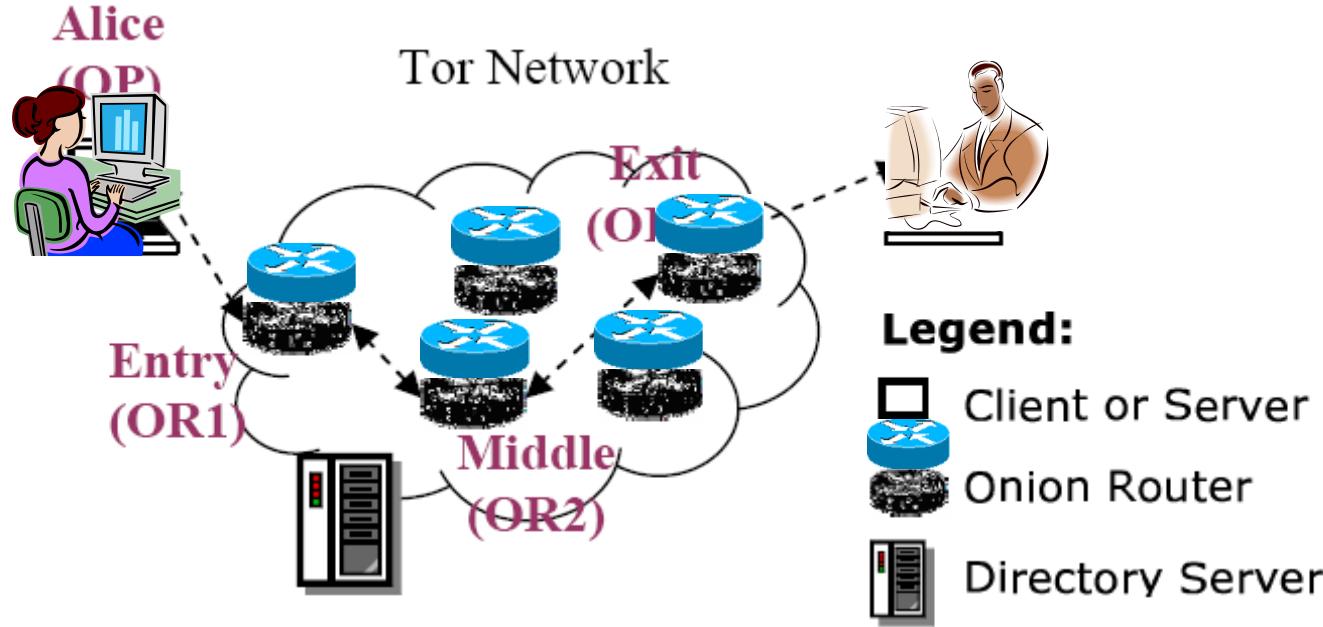
- Solution: distributed, anonymous networks

Example: Tor Networks

- Basic idea
 - Overlay network using Onion Routers (OR)
 - Onion routing so that each onion router knows only the previous and next hop

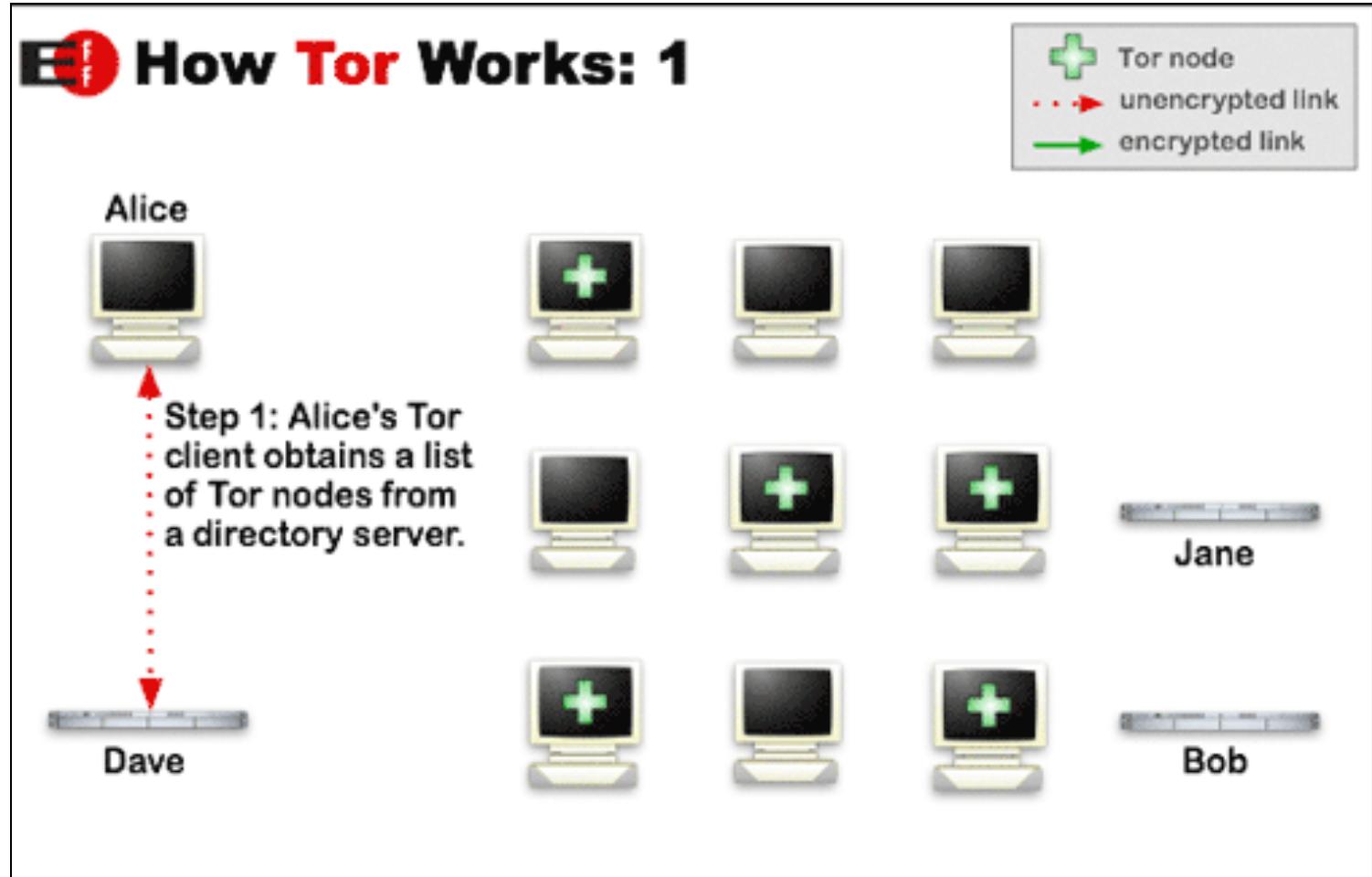
<https://torstatus.blutmagie.de/>

Tor Components

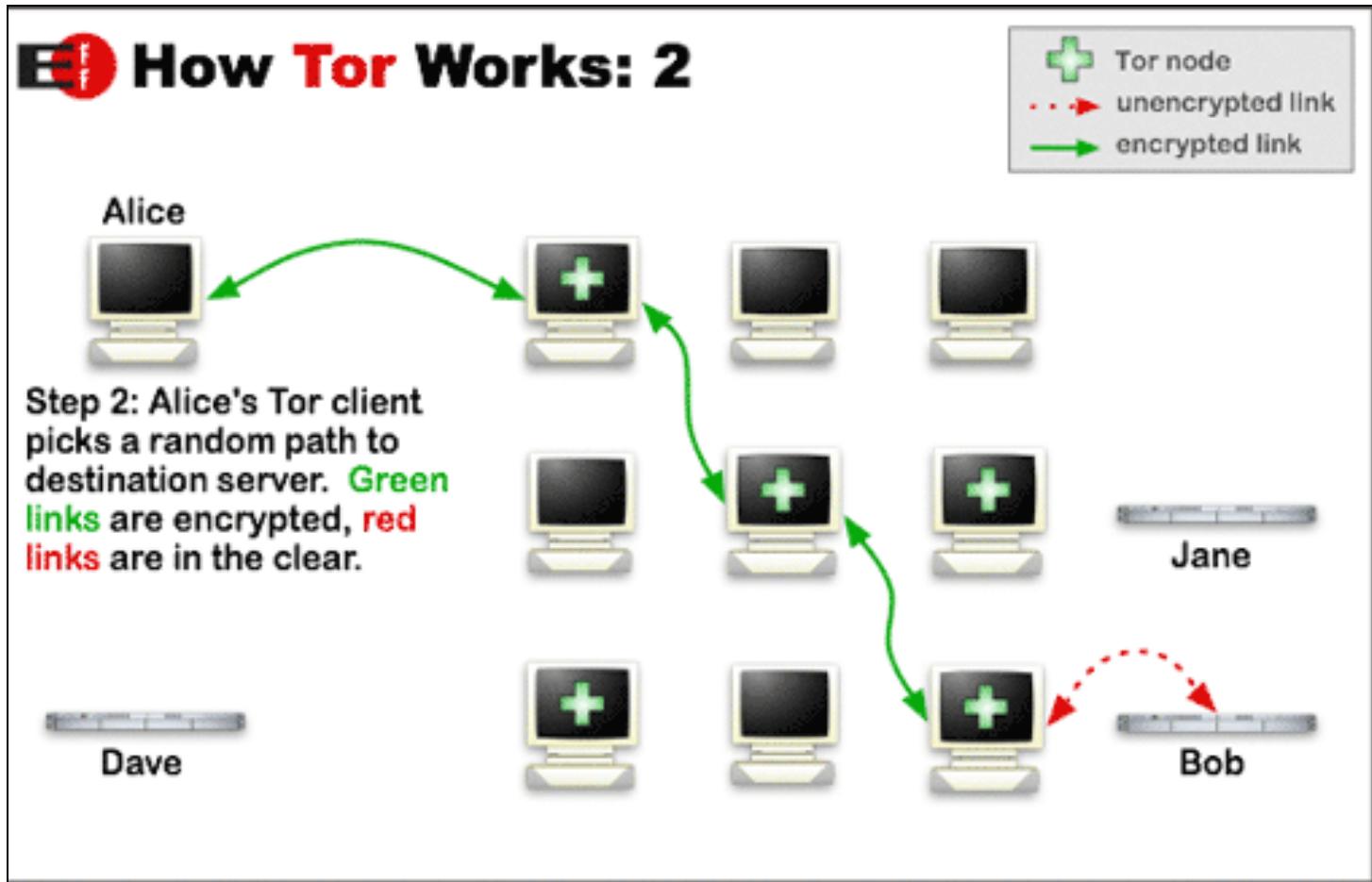


- **Client:** the user of the Tor network
- **Server:** the target TCP applications such as web servers
- **Tor (onion) router:** the special proxy relays the application data
- **Directory server:** servers holding Tor router information

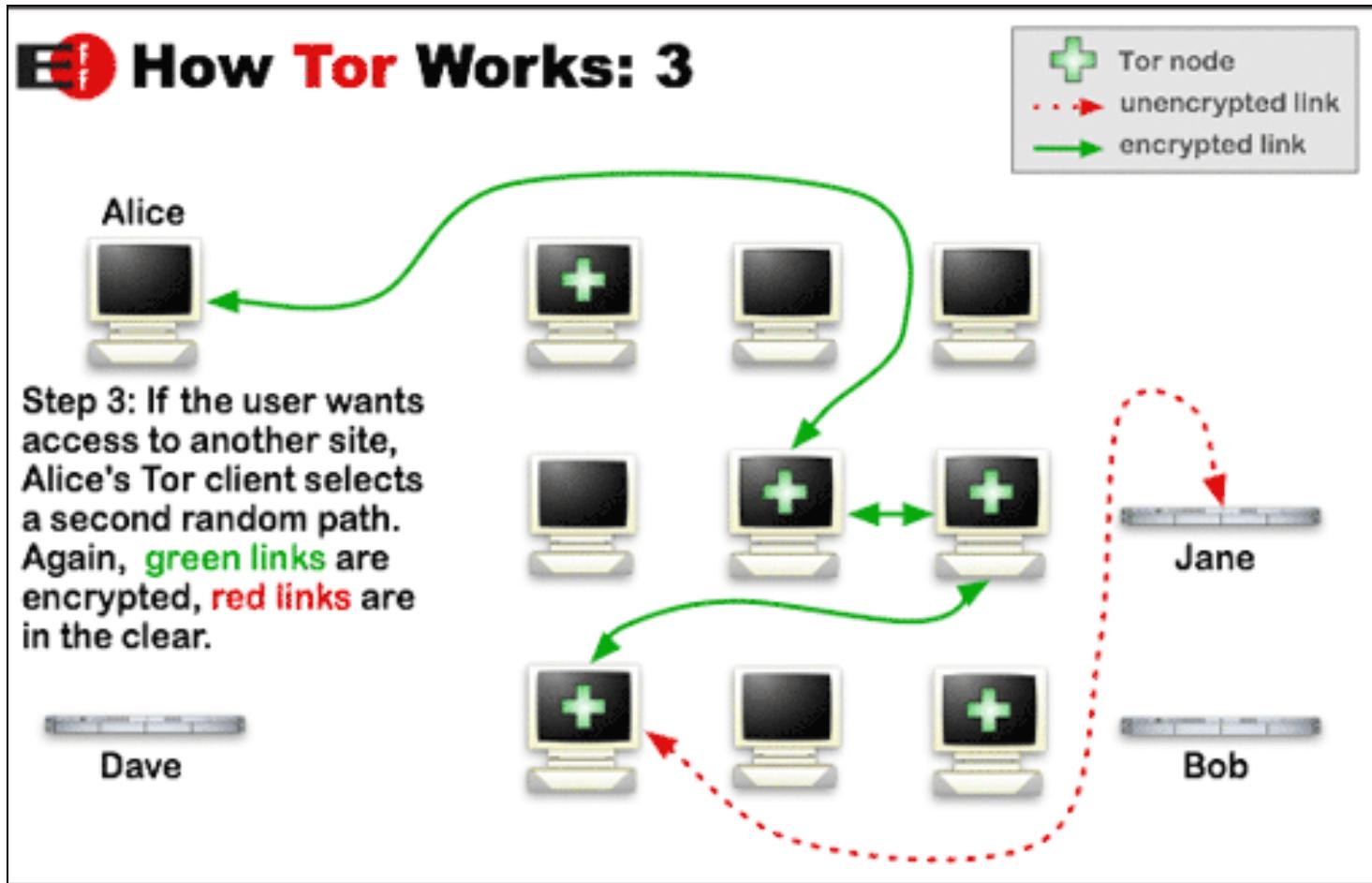
Tor Steps



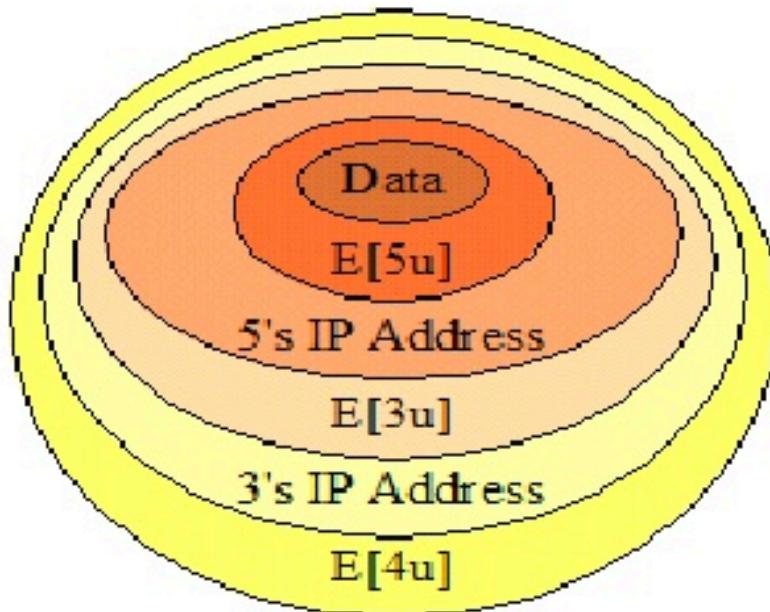
Tor Steps



Tor Steps

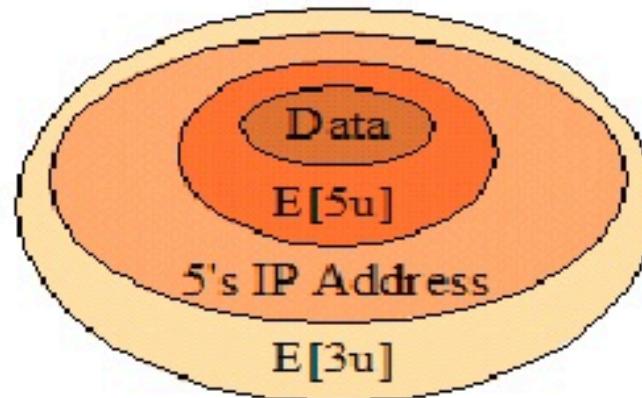


Onions using public key



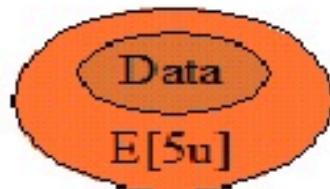
Onion Sent by Client to 4

Router 4 will decrypt the E[4u] layer using its private key, to find the next router's IP address, and encrypted data.



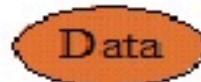
Onion Sent by 4 to 3

Router 3 will decrypt the E[3u] layer using its private key, to find the next router's IP address, and encrypted data.



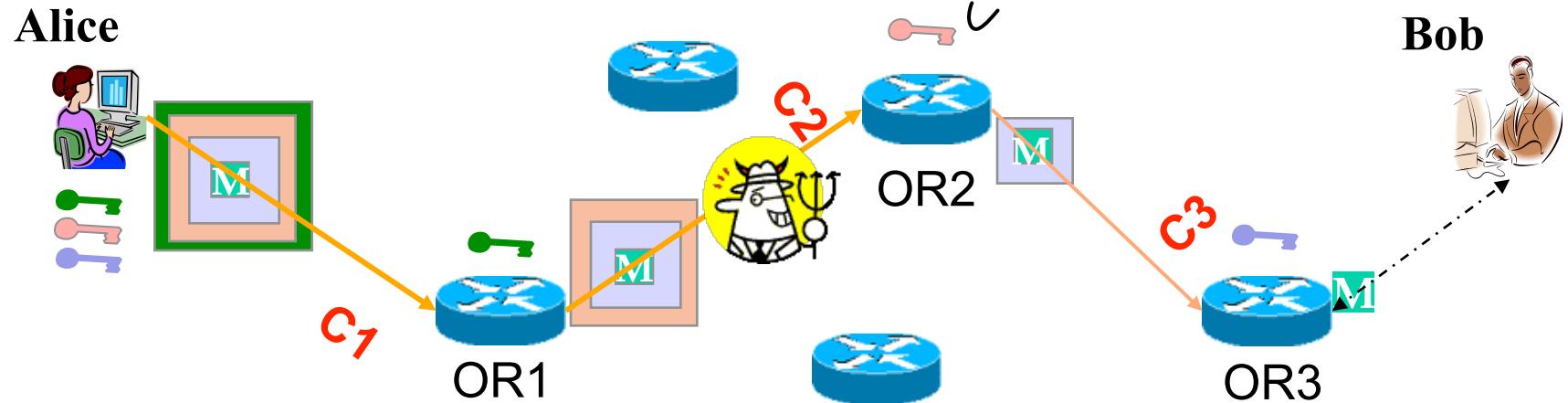
Onion Sent by 3 to 5

Router 5 will decrypt the E[5u] layer using its private key, to find just the unencrypted data packet.



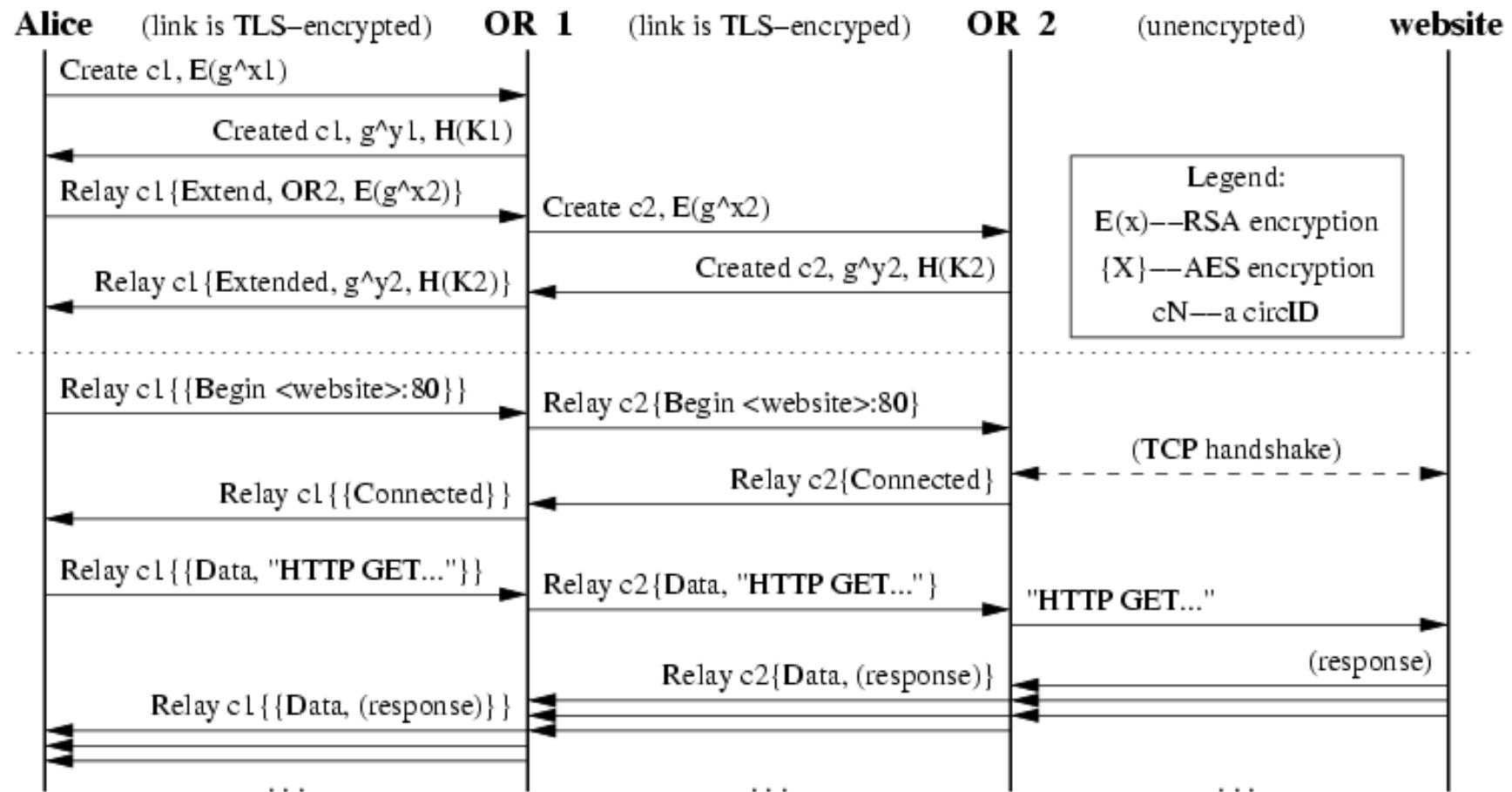
Data Sent by 5 to Target

Tor Optimization: Symmetric Key



- A circuit is built incrementally one hop by one hop
- Onion-like encryption
 - Alice negotiates an AES key with each router
 - Messages are divided into equal sized **cells**
 - Each router knows only its predecessor and successor
 - Only the Exit router (OR3) can see the message

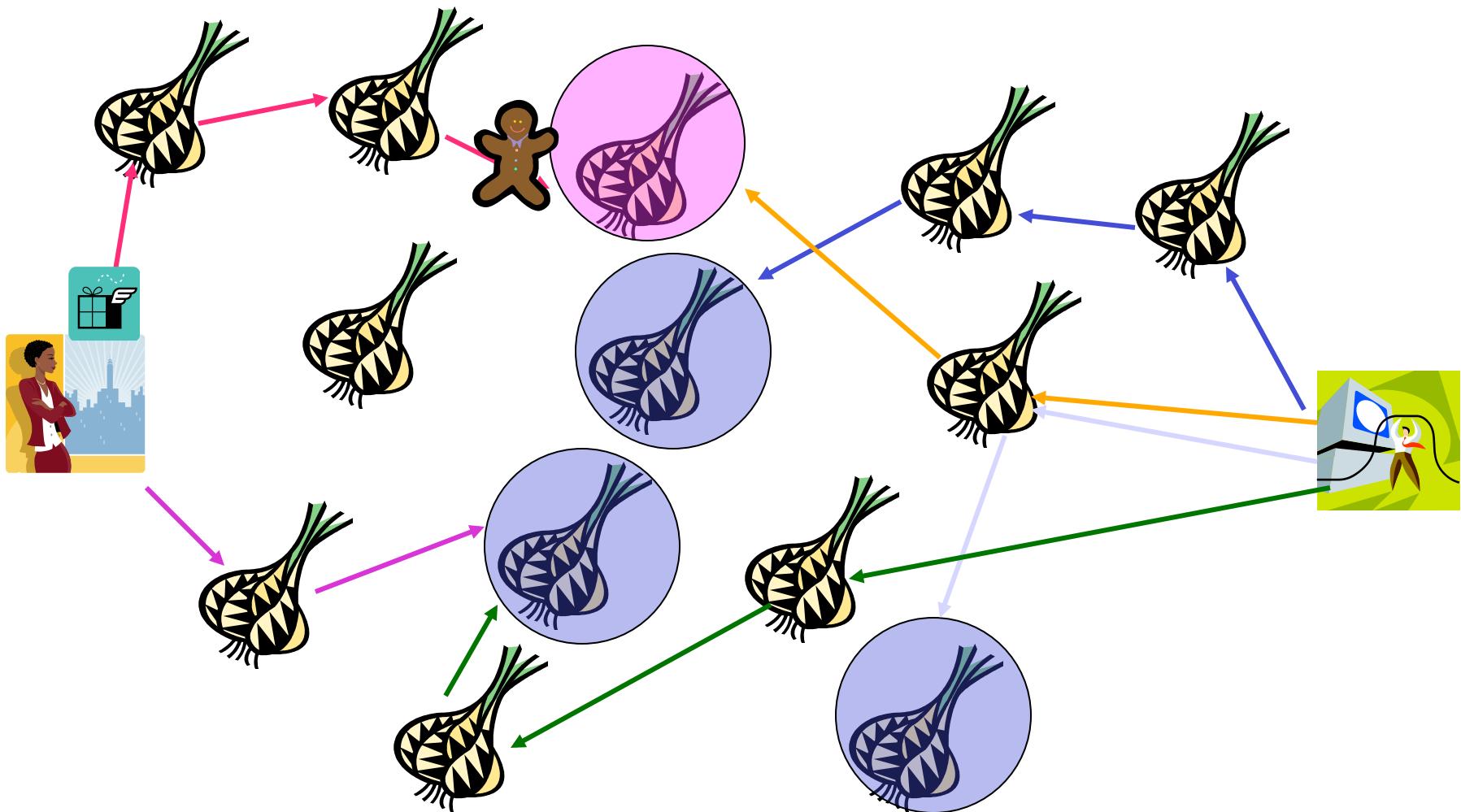
Example Tor Commands



Tor Location-Hidden Service

- Goal: allow one to offer a TCP service without revealing his IP address
 - Access Control: filtering incoming requests
 - Robustness: maintain a long-term pseudonymous identity
 - Smear-resistance: against socially disapproved acts

Tor Location-Hidden Service: Creating and connecting to a Location hidden service

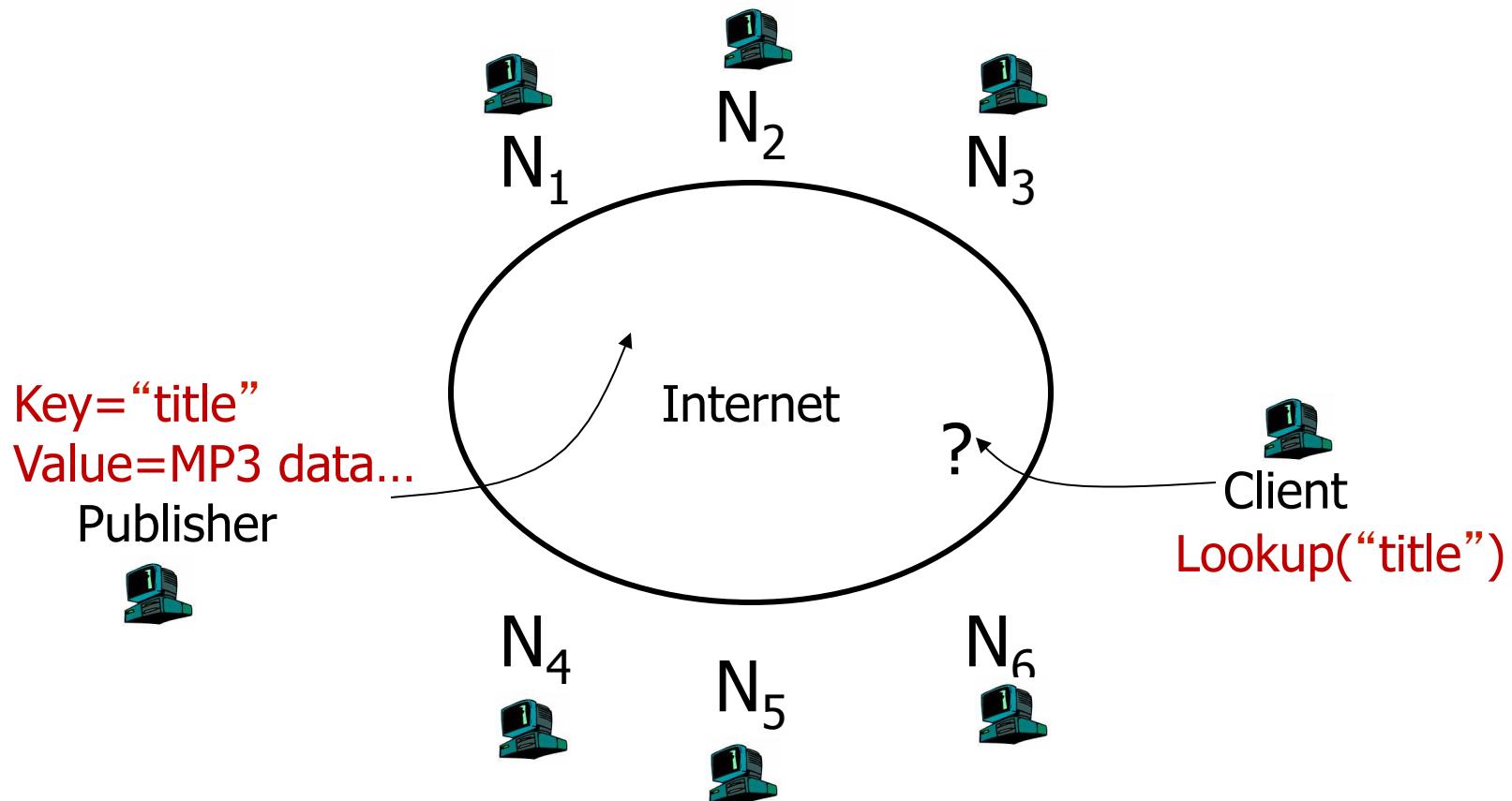


Optional Slides

Outline

- Admin and recap
- Multiple servers
- Application overlays
 - Overlays for scalability
 - Overlays for anonymity
 - Overlays for distributed content hosting

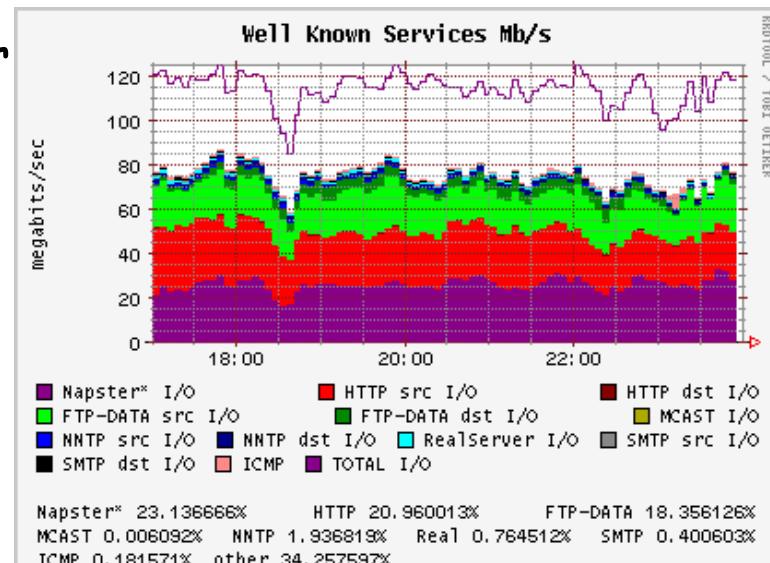
The Content Lookup Problem



find where a particular file is stored
pay particular attention to see its equivalence of DNS

Napster

- Program for sharing **music** over the Internet
- History:
 - **5/99:** Shawn Fanning (freshman, Northeastern U.) founded Napster Online music service, wrote the program in 60 hours
 - **12/99:** first lawsuit
 - **3/00:** 25% UWisc traffic Napster
 - **2000:** est. 60M users
 - **2/01:** US Circuit Court of Appeals: Napster knew users violating copyright laws
 - **7/01:** # simultaneous online users: Napster 160K
 - **9/02:** bankruptcy



03/2000

We are referring to the Napster before closure.

Napster: How Does it Work?

Application-level, client-server protocol over TCP

A centralized index system that maps files (songs) to machines that are alive and with files

Steps:

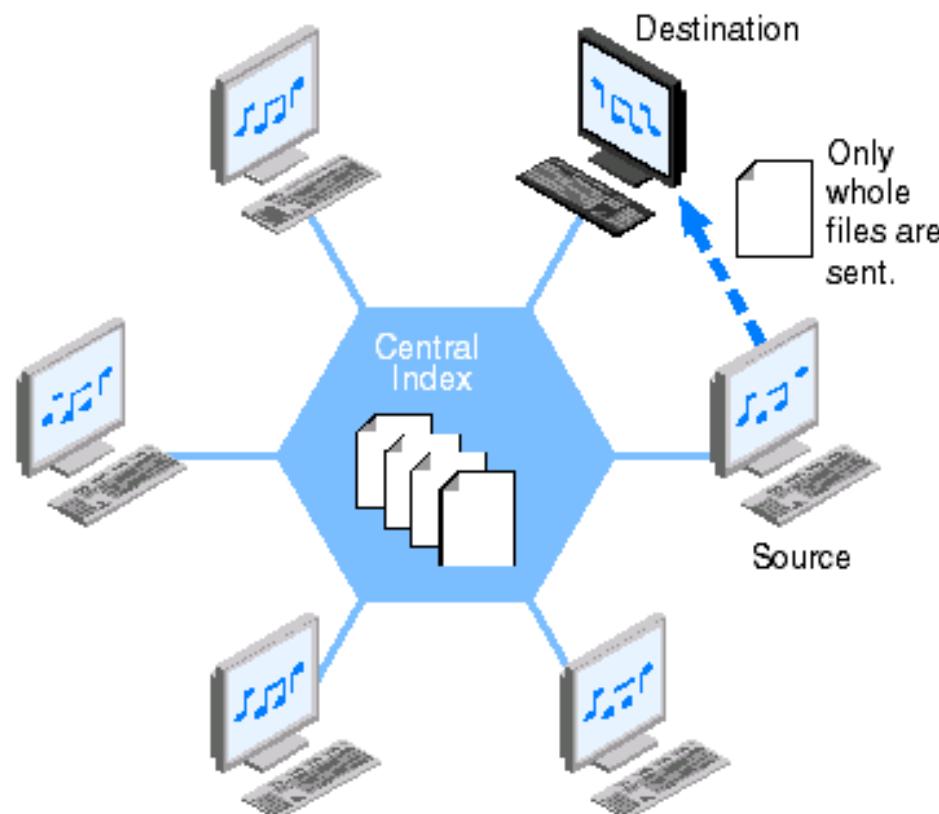
- Connect to Napster server
- Upload your list of files (push) to server
- Give server keywords to search the full list
- Select “best” of hosts with answers

Napster Architecture

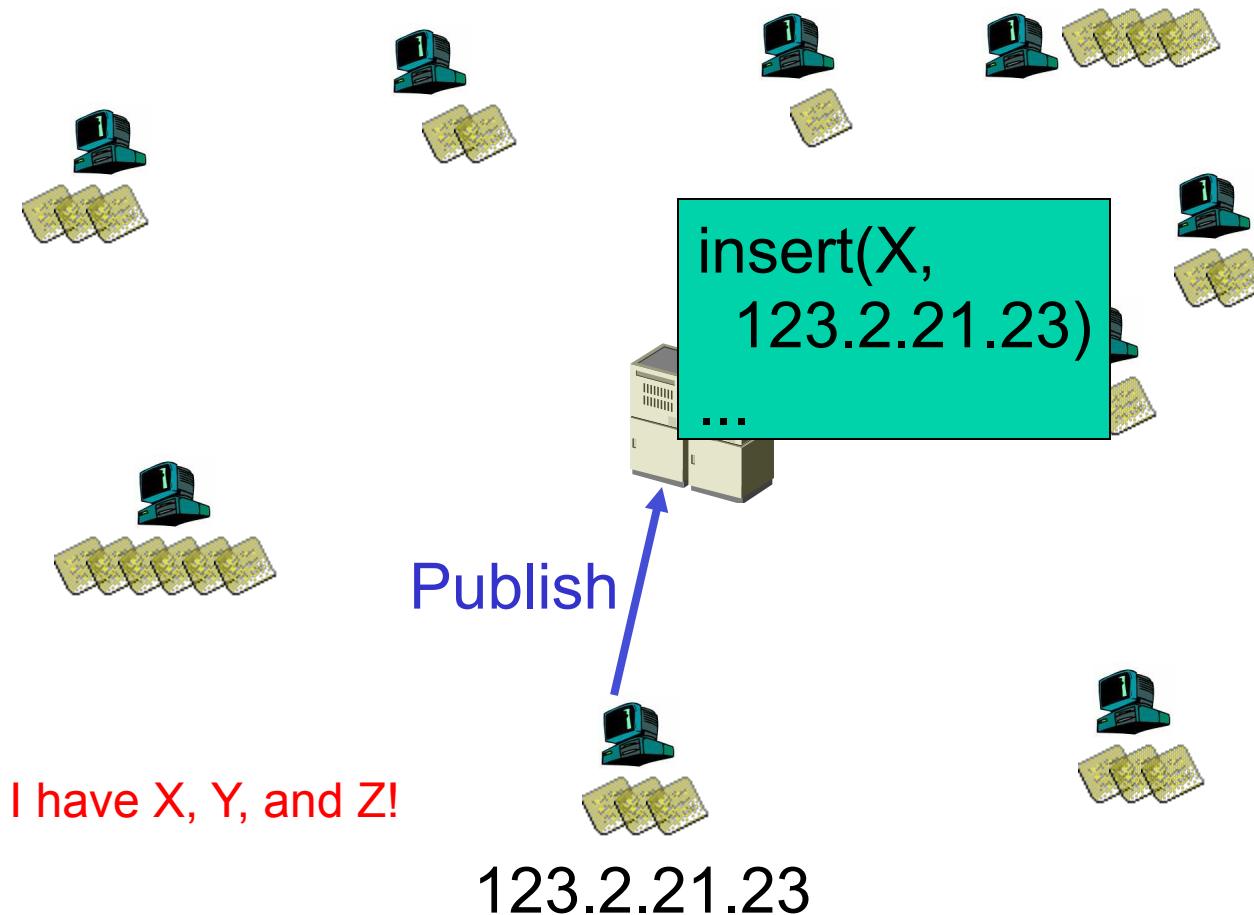
From Computer Desktop Encyclopedia
© 2004 The Computer Language Co. Inc.

THE ORIGINAL NAPSTER

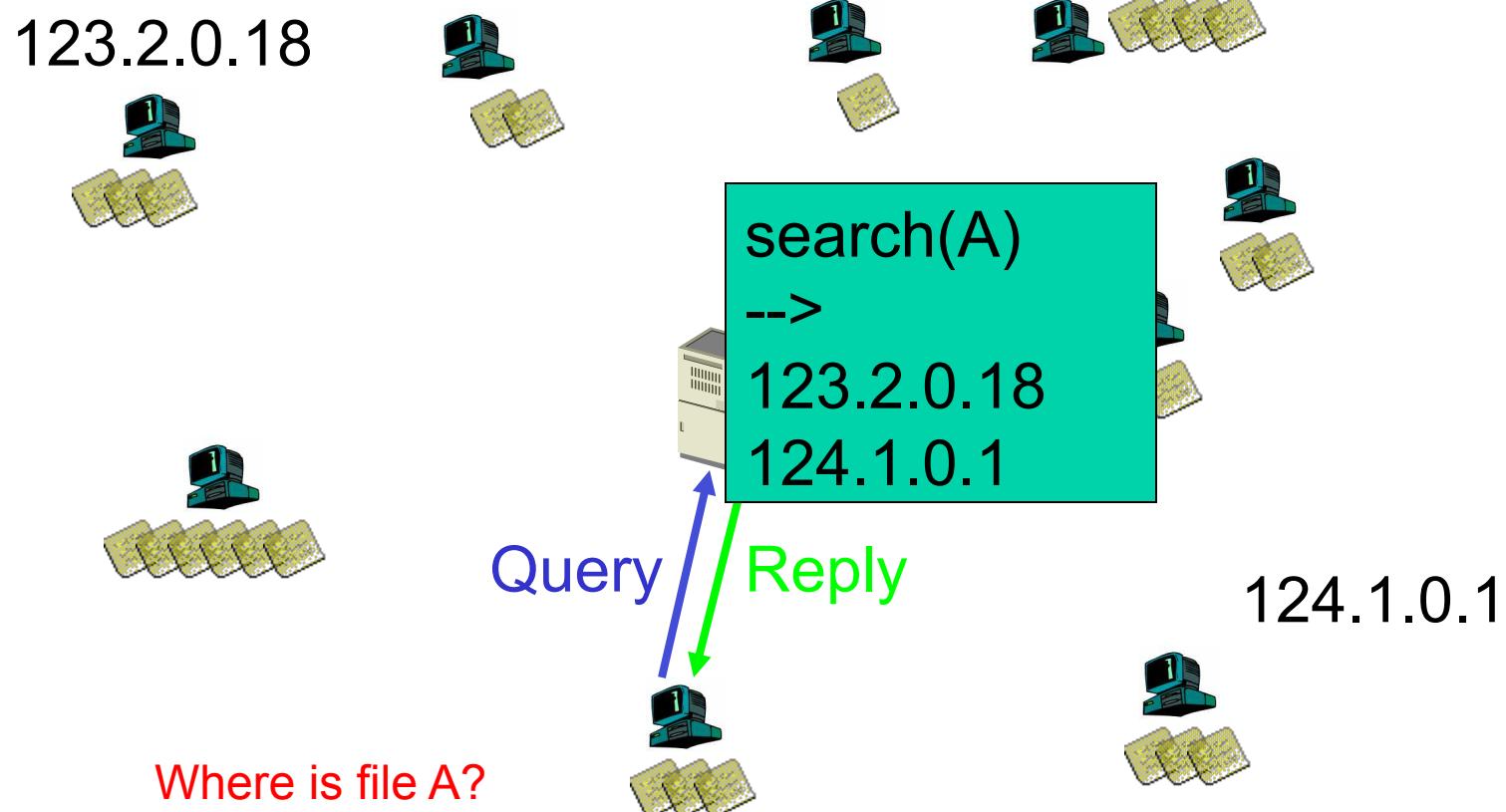
Napster provided a central directory of users who had files to share.



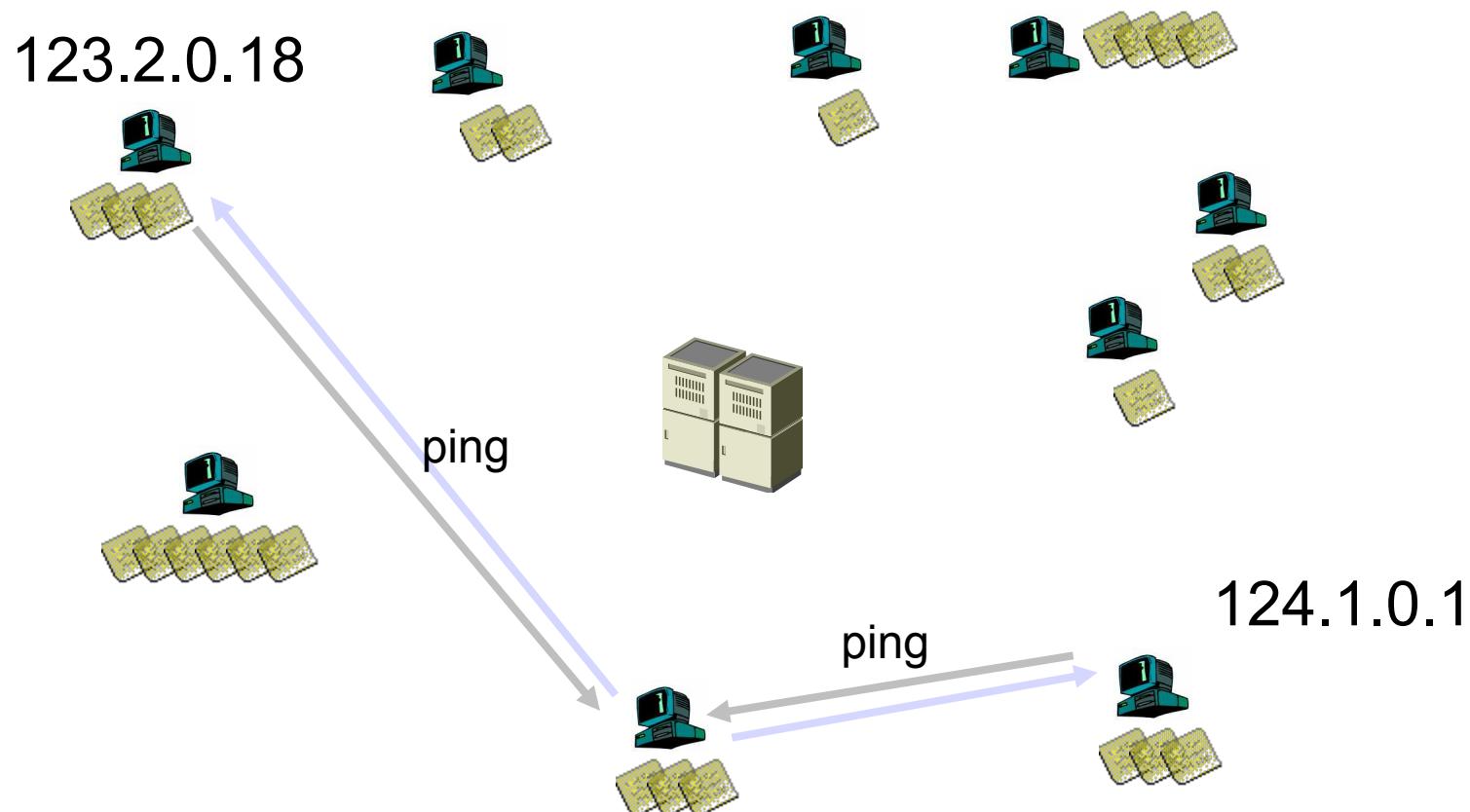
Napster: Publish



Napster: Search

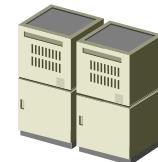
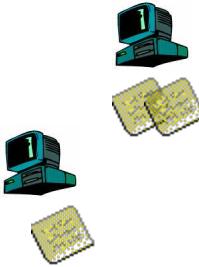
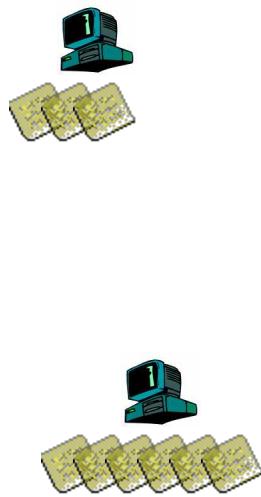


Napster: Ping

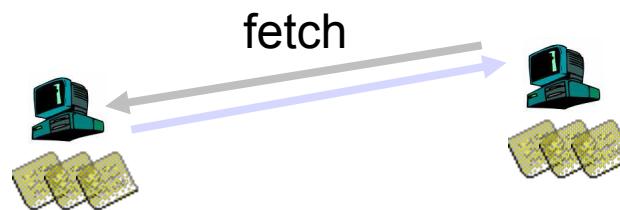


Napster: Fetch

123.2.0.18



124.1.0.1



Napster Messages

General Packet Format

[chunksize] [chunkinfo] [data...]

CHUNKSIZE:

Intel-endian 16-bit integer
size of [data...] in bytes

CHUNKINFO: (hex)

Intel-endian 16-bit integer.

00 - login rejected	5B - whois query
02 - login requested	5C - whois result
03 - login accepted	5D - whois: user is offline!
0D - challenge? (nuprin1715)	69 - list all channels
2D - added to hotlist	6A - channel info
2E - browse error (user isn't online!)	90 - join channel
2F - user offline	91 - leave channel

Centralized Database: Napster

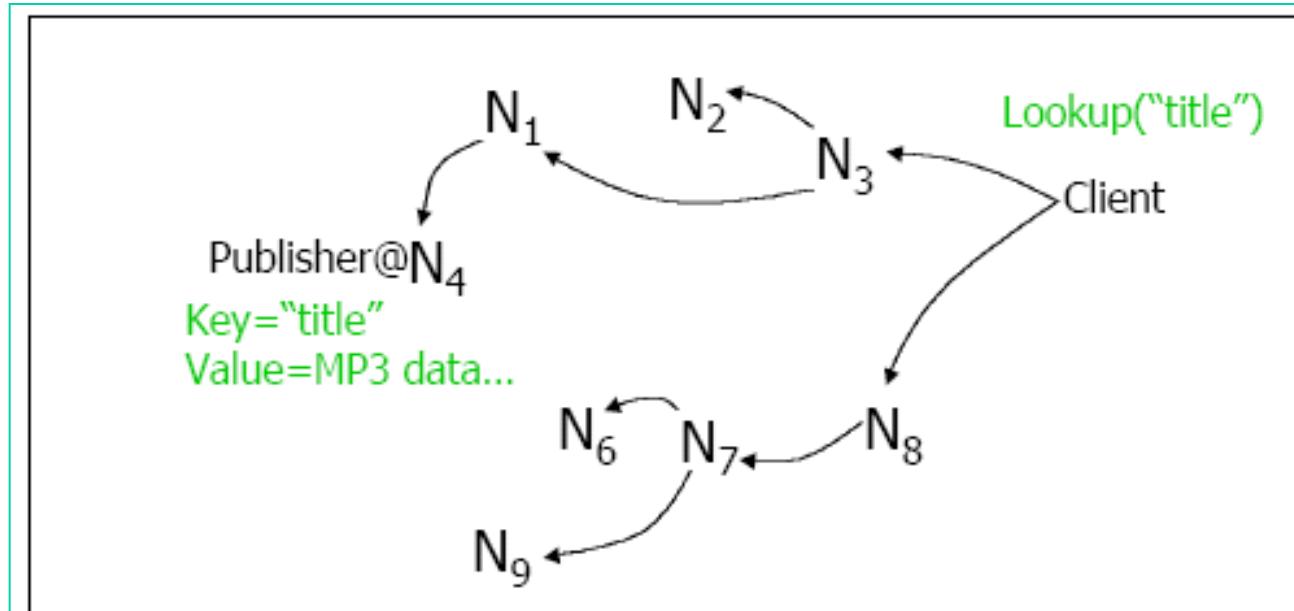
- Summary of features: a hybrid design
 - control: client-server (aka special DNS) for files
 - data: peer to peer
- Advantages
 - simplicity, easy to implement sophisticated search engines (boolean exp) on top of the index system
- Disadvantages
 - application specific (compared with DNS)
 - lack of robustness, scalability: central search server single point of bottleneck/failure
 - easy to take down !

Gnutella

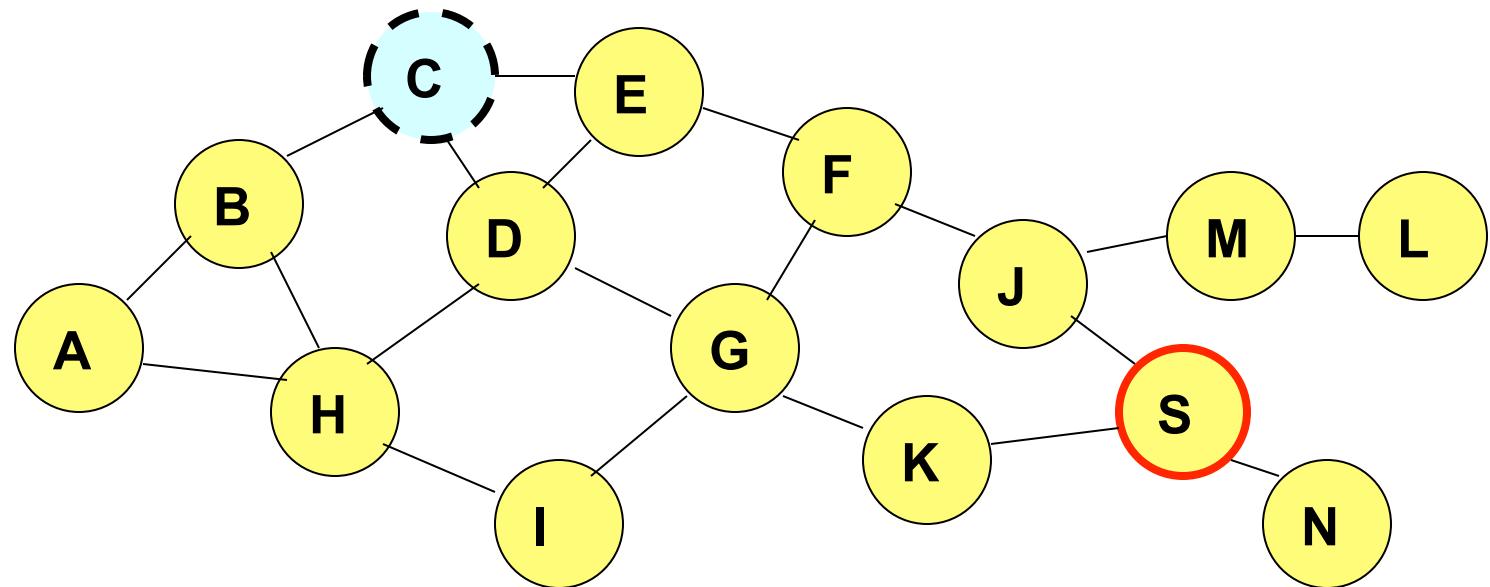
- On March 14th 2000, J. Frankel and T. Pepper from AOL's Nullsoft division (also the developers of the popular Winamp mp3 player) released Gnutella
- Within hours, AOL pulled the plug on it
- Quickly reverse-engineered and soon many other clients became available: Bearshare, Morpheus, LimeWire, etc.

Decentralized Flooding: Gnutella

- On startup, client contacts other servents (server + client) in network to form interconnection/peering relationships
 - servent interconnection used to forward control (queries, hits, etc)
- How to find a resource record: decentralized flooding
 - send requests to neighbors
 - neighbors recursively forward the requests

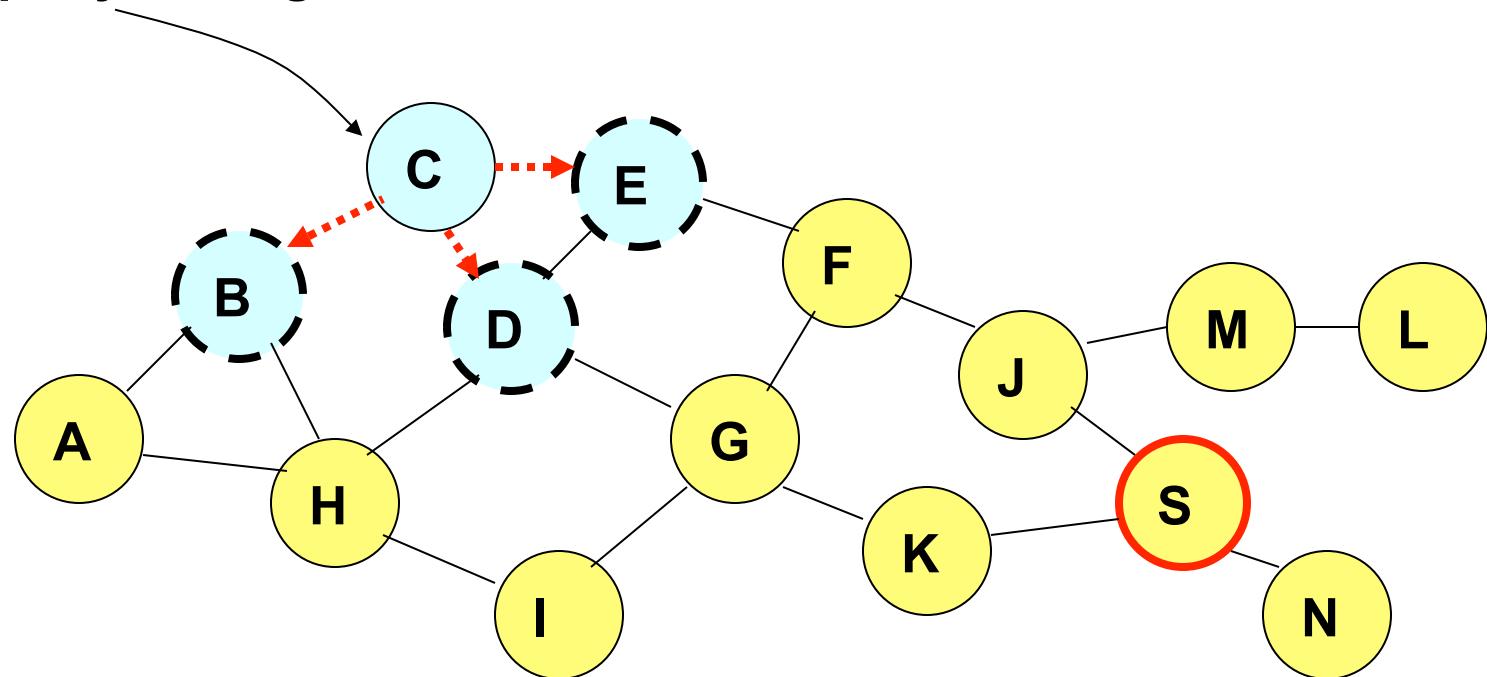


Decentralized Flooding



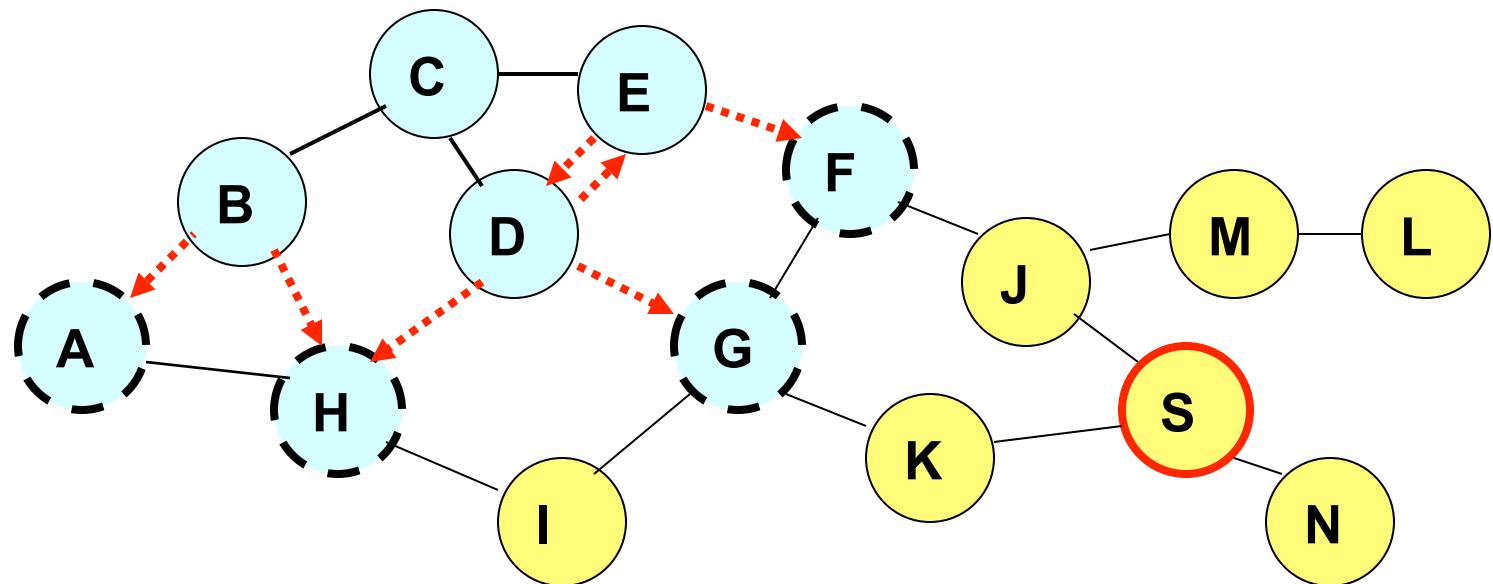
Decentralized Flooding

send query to neighbors



- Each node forwards the query to its neighbors other than the one who forwards it the query

Background: Decentralized Flooding



- Each node should keep track of forwarded queries to avoid loop !
 - node state: nodes keep state (which will time out---soft state)
 - packet state: carry the state in the query, i.e. carry a list of visited nodes

Decentralized Flooding: Gnutella

- Basic message header
 - Unique ID, TTL, Hops
- Message types
 - Ping - probes network for other servents
 - Pong - response to ping, contains IP addr, # of files, etc.
 - Query - search criteria + speed requirement of servent
 - QueryHit - successful response to Query, contains addr + port to transfer from, speed of servent, etc.
 - Ping, Queries are flooded
 - QueryHit, Pong: reverse path of previous message

Advantages and Disadvantages of Gnutella

- Advantages:
 - totally decentralized, highly robust

- Disadvantages:
 - not scalable; the entire network can be swamped with flood requests
 - especially hard on slow clients; at some point broadcast traffic on Gnutella exceeded 56 kbps
 - to alleviate this problem, each request has a TTL to limit the scope
 - each query has an initial TTL, and each node forwarding it reduces it by one; if TTL reaches 0, the query is dropped (consequence?)

Freenet

- History
 - final year project [Ian Clarke](#) , [Edinburgh University](#), Scotland, June, 1999
- Goals:
 - totally distributed system without using centralized index or broadcast (flooding), instead search by **routing**
 - routing/storing system responds adaptively to usage patterns, transparently moving, replicating files as necessary to provide efficient service
 - provide publisher anonymity, security
 - free speech : resistant to attacks - a third party shouldn't be able to deny (e.g., deleting) the access to a particular file (data item, object)

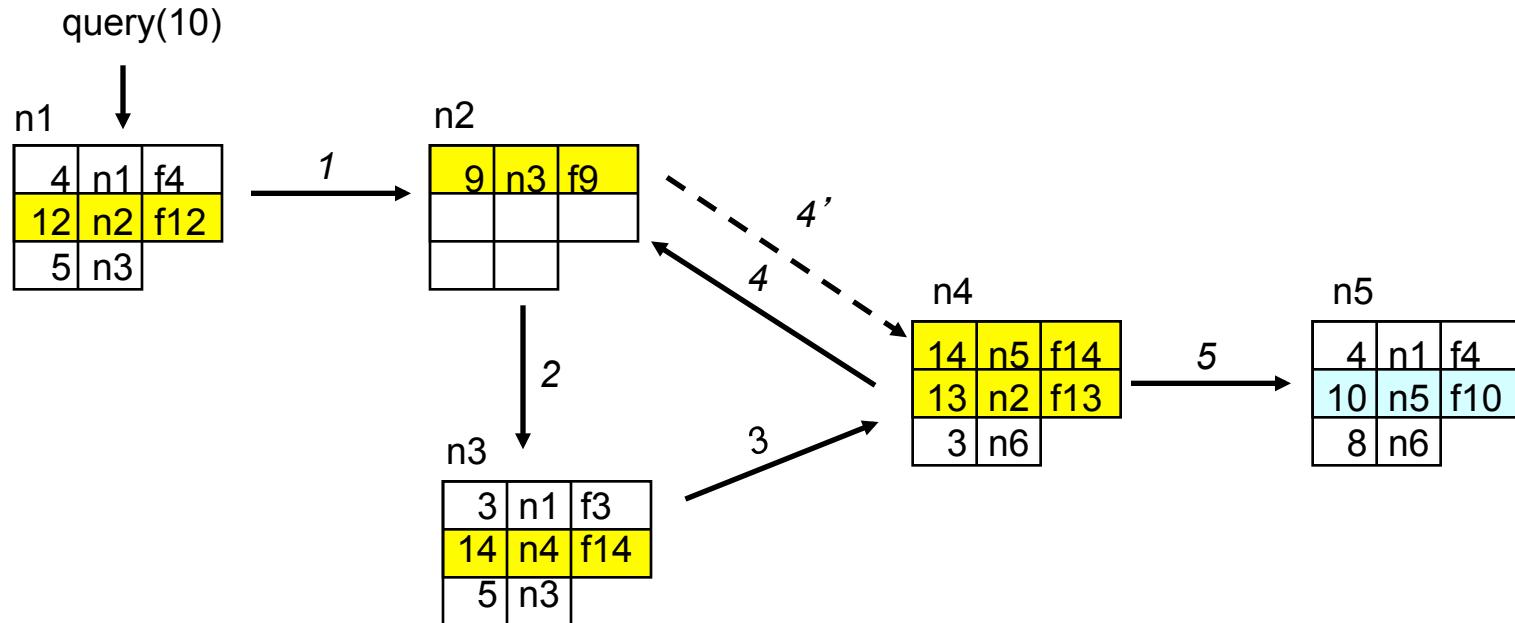
Basic Structure of Freenet

- Each machine stores a set of files; each file is identified by a unique identifier (called key or id)
 - Each node maintains a “routing table”
 - *id* - file id, key
 - *next_hop node* - where to search for a file with *id'* that is similar to *id*
 - *file* - local copy, if exists, of file with *id*

Freenet Query

- API: $\text{file} = \text{query}(id)$
 - Upon receiving a query for file id
 - check whether the queried file is stored locally
 - check TTL to limit the search scope
 - each query is associated a TTL that is decremented each time the query message is forwarded
 - when TTL=1, the query is forwarded with a probability
 - TTL can be initiated to a random value (why random value?)
 - look for the “**closest**” id in the table with an unvisited $next_hop$ node
 - if found one, forward the query to the corresponding $next_hop$
 - otherwise, backtrack
 - ends up performing a Depth First Search (DFS)-like traversal
 - search direction ordered by closeness to target
 - When file is returned it is cached along the reverse path (any advantage?)

Query Example



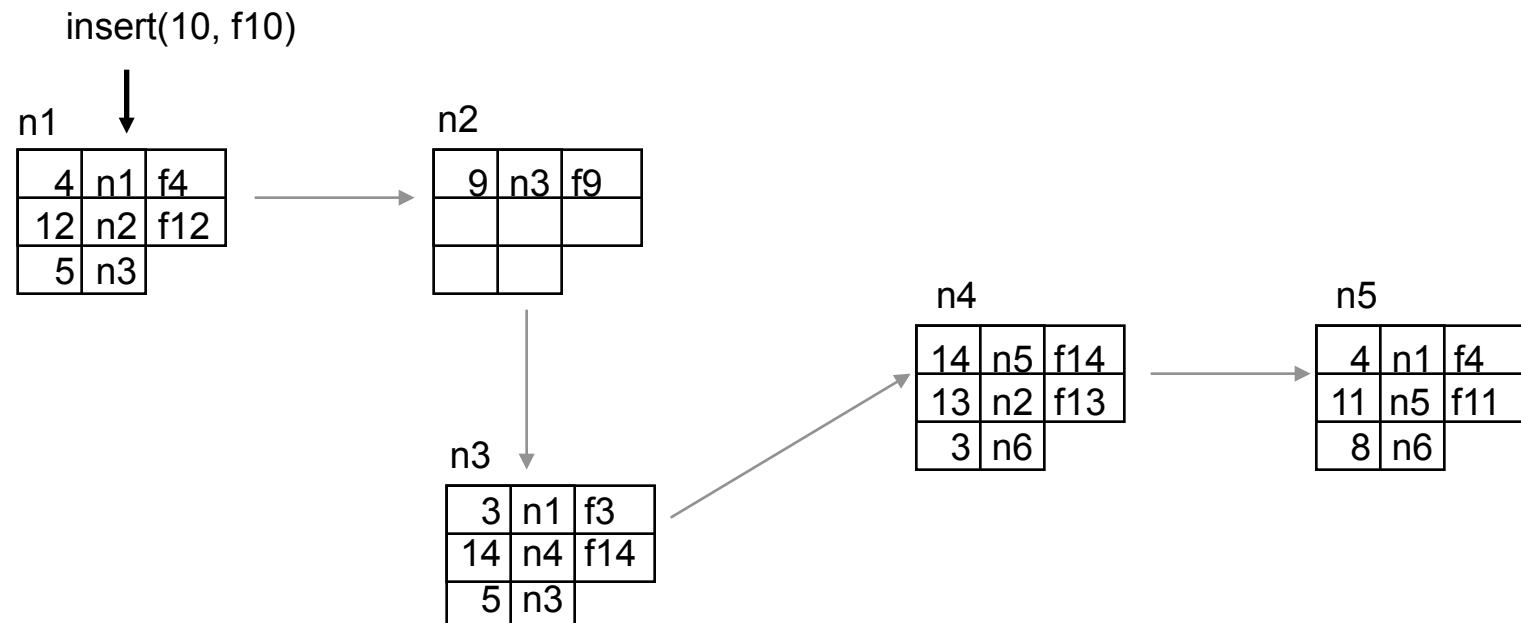
Beside the routing table, each node also maintains a query table containing the state of all outstanding queries that have traversed it → to backtrack

Insert

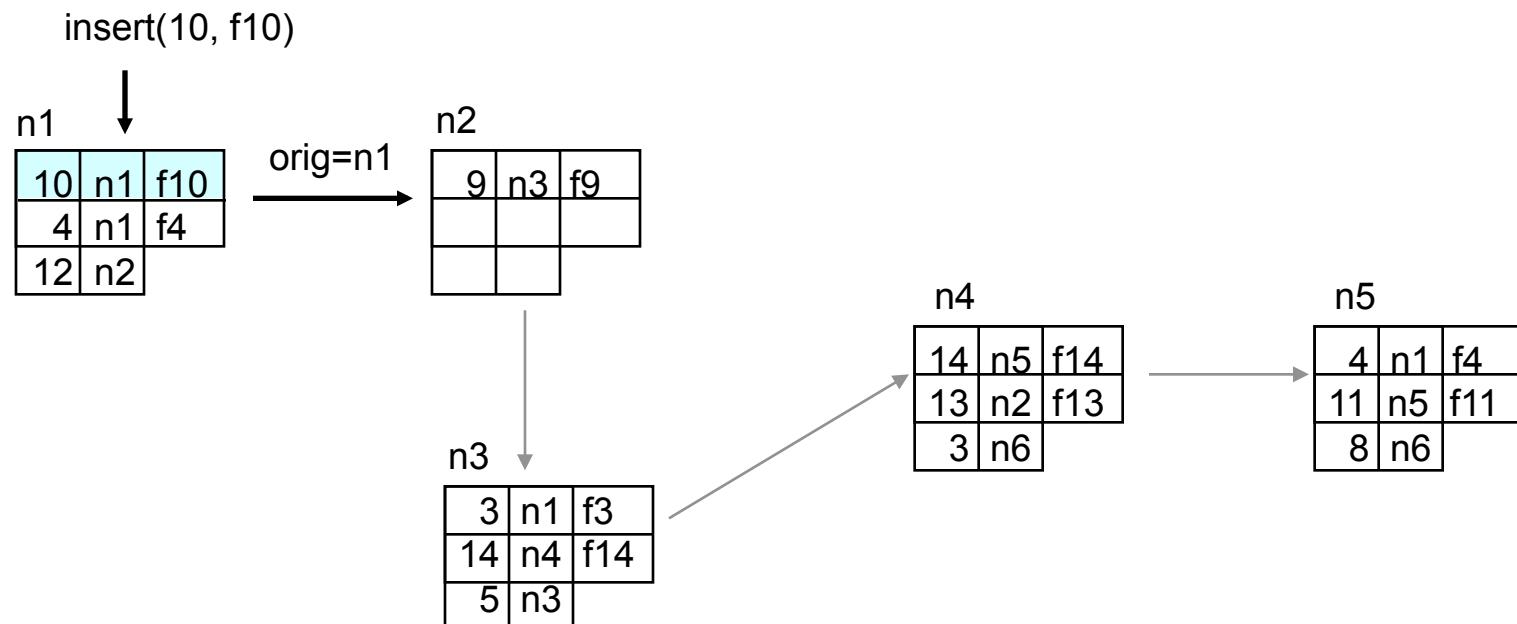
- API: `insert(id, file);`
- Two steps
 - first attempt a “search” for the file to be inserted
 - if found, report collision
 - if not found, insert the file by sending it along the query path (why?)
 - a node probabilistically replaces the originator with itself (why?)

Insert Example

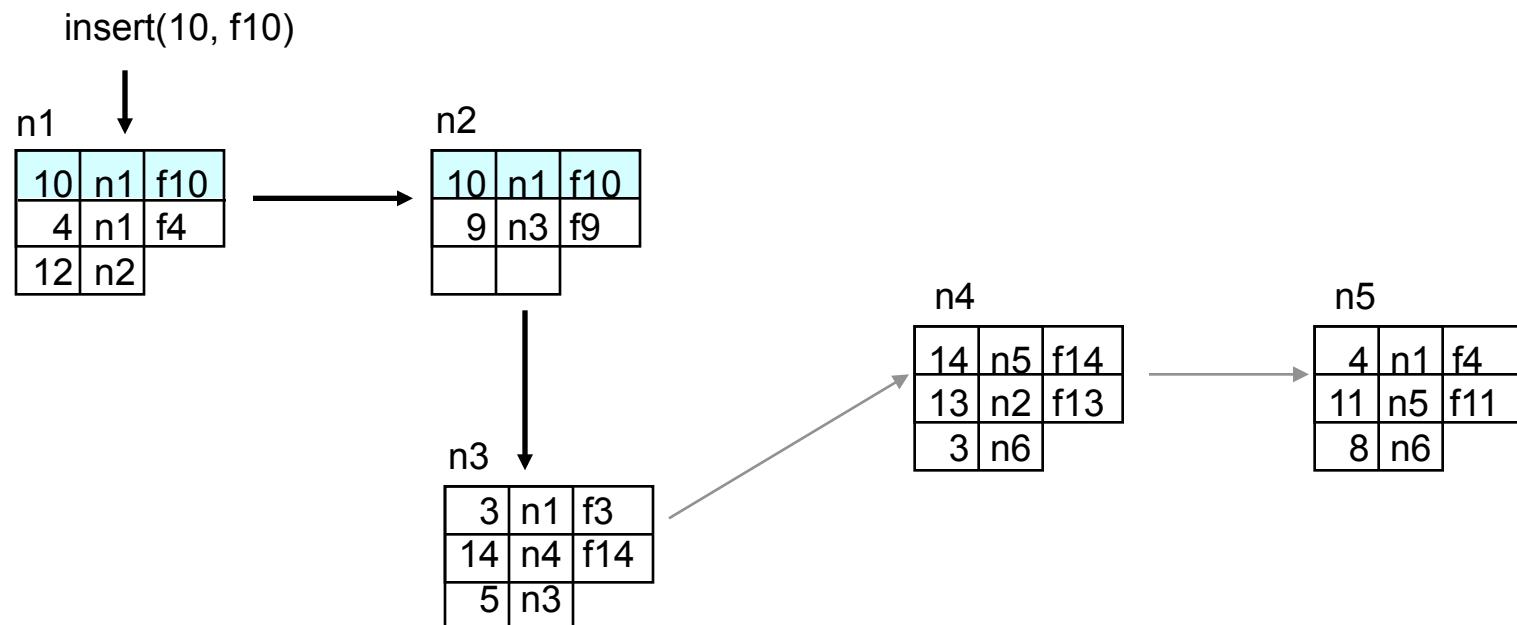
- Assume query returned failure along the shown path (backtrack slightly complicates things); insert f10



Insert Example

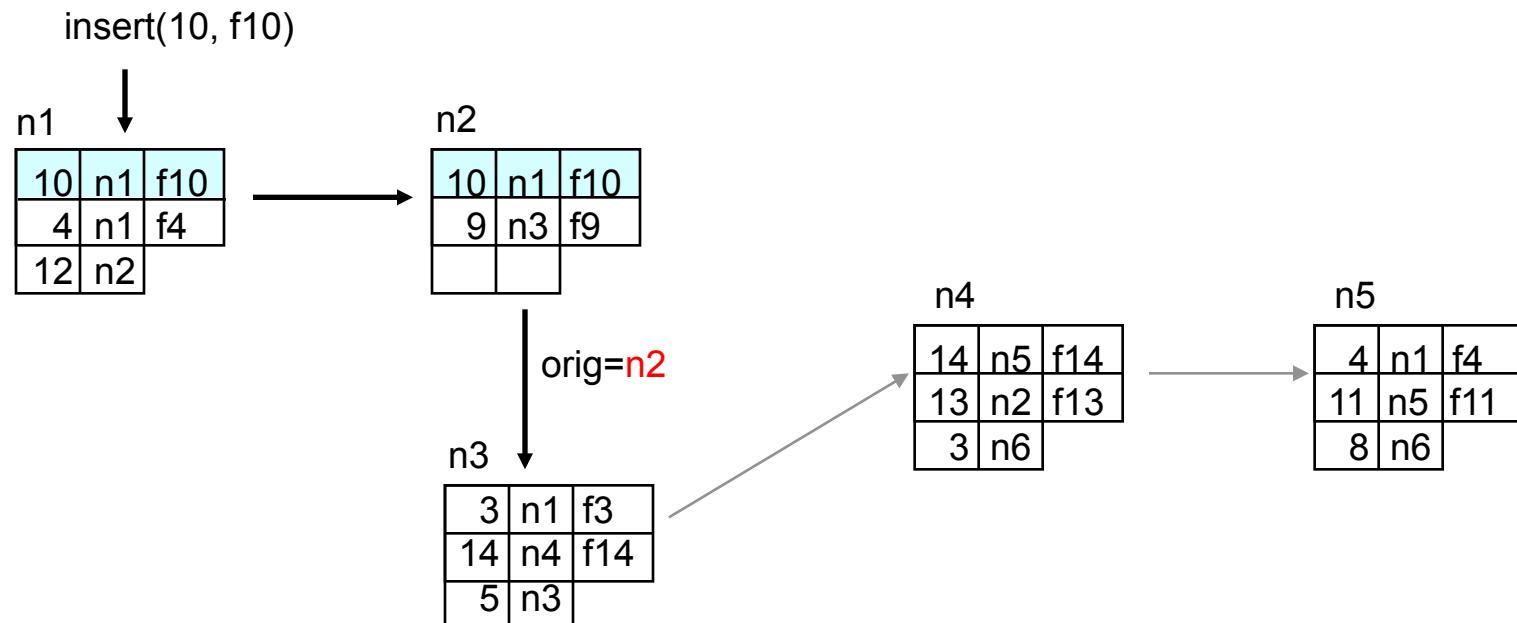


Insert Example

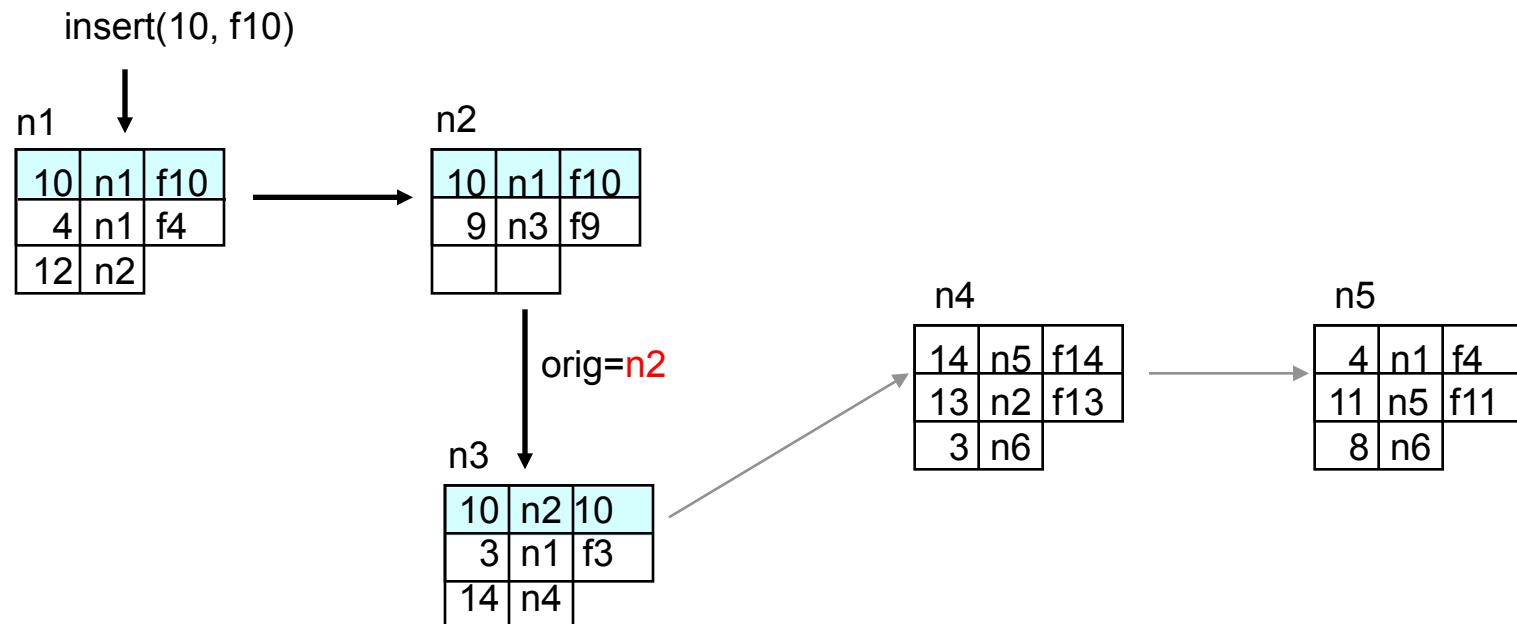


Insert Example

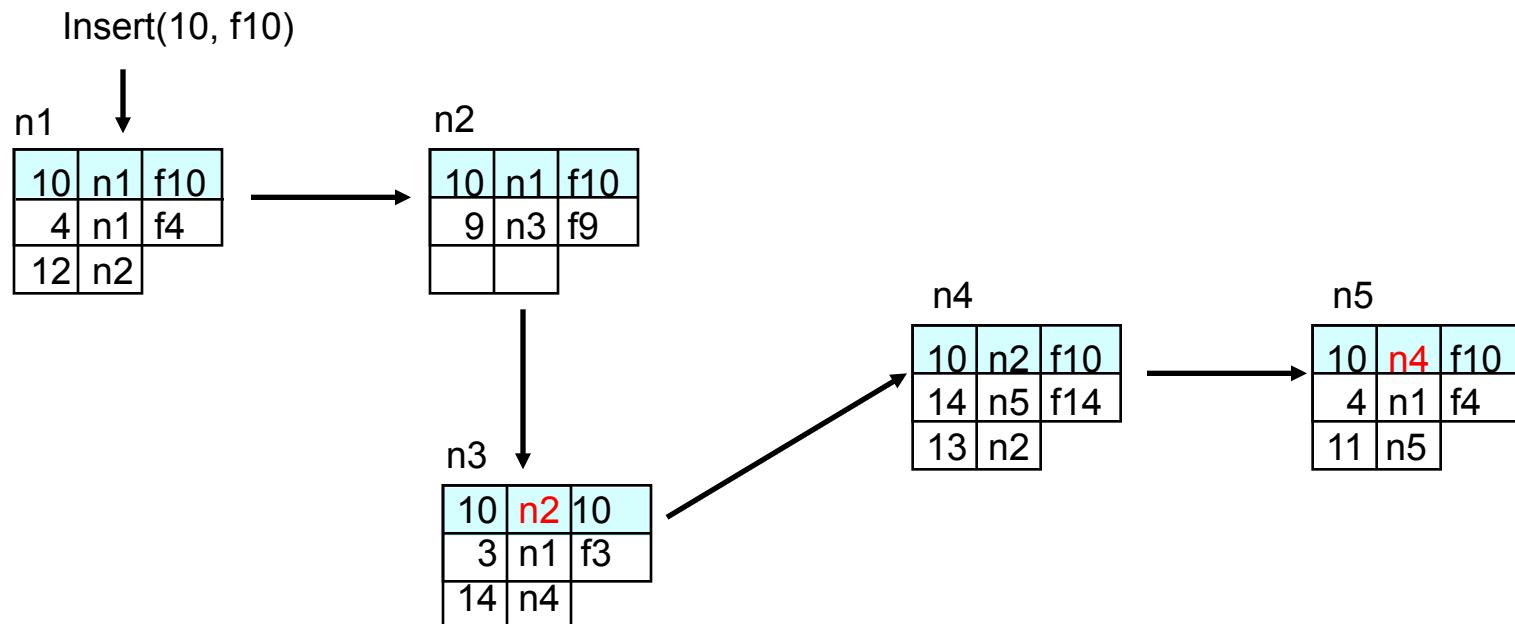
- n2 replaces the originator (n1) with itself



Insert Example



Insert Example

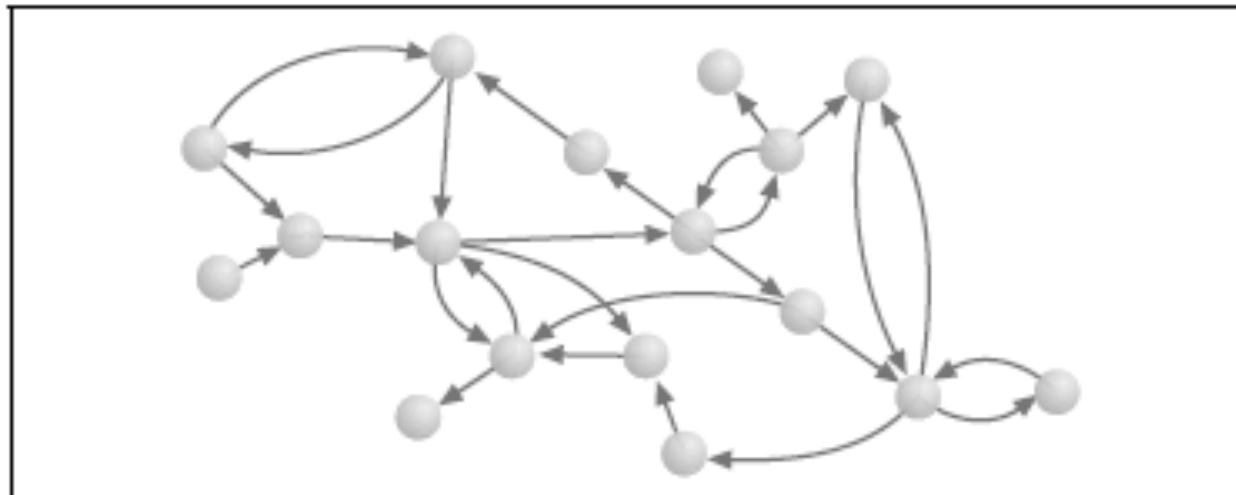


Freenet Analysis

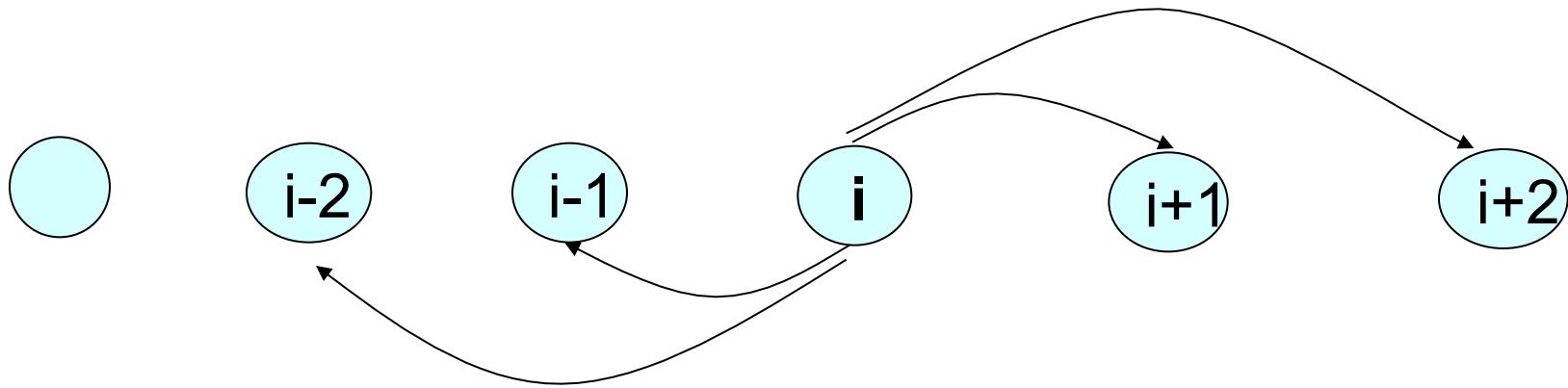
- Authors claim the following effects:
 - nodes eventually specialize in locating similar keys
 - if a node is listed in a routing table, it will get queries for related keys
 - thus will gain “experience” answering those queries
 - popular data will be transparently replicated and will exist closer to requestors
 - as nodes process queries, connectivity increases
 - nodes will discover other nodes in the network
- Caveat: lexicographic closeness of file names/keys may not imply content similarity

Understanding Freenet Self-Organization: Freenet Graph

- We create a Freenet reference graph
 - creating a vertex for each Freenet node
 - adding a directed link from A to B if A refers to an item stored at B

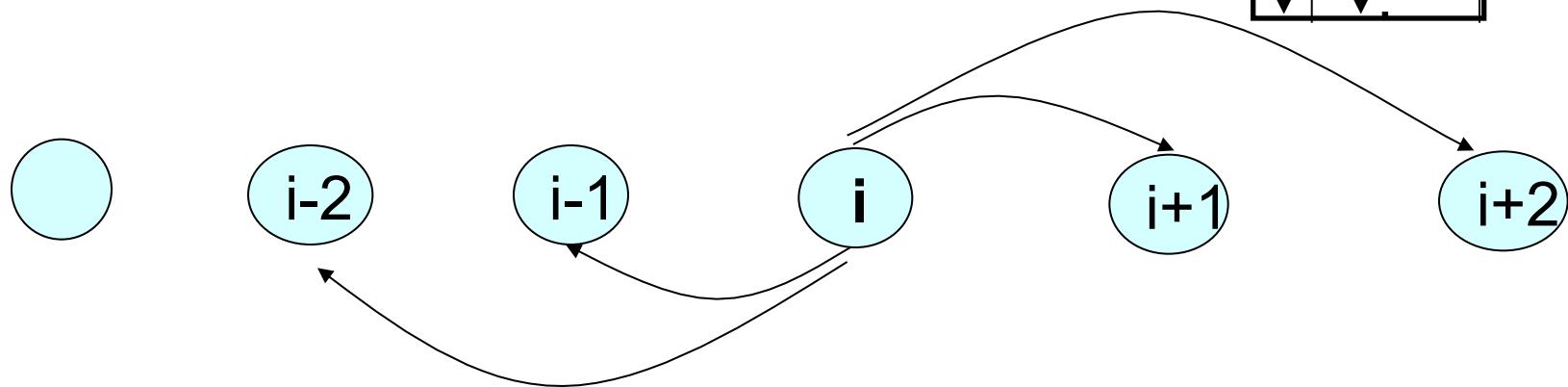


Experiment: Freenet Graph: Init



- Assume a network of 1000 nodes, with node id 0 to 999
 - Each node can store 50 data items, and 200 references
 - Assume initially each node i has i , and knows the storage of $i - 2, -1, i + 1, i + 2$ (all mod 1000)
 - thus a regular, locally-clustered graph with avg path length $\sim 1000 / 8 = 125$

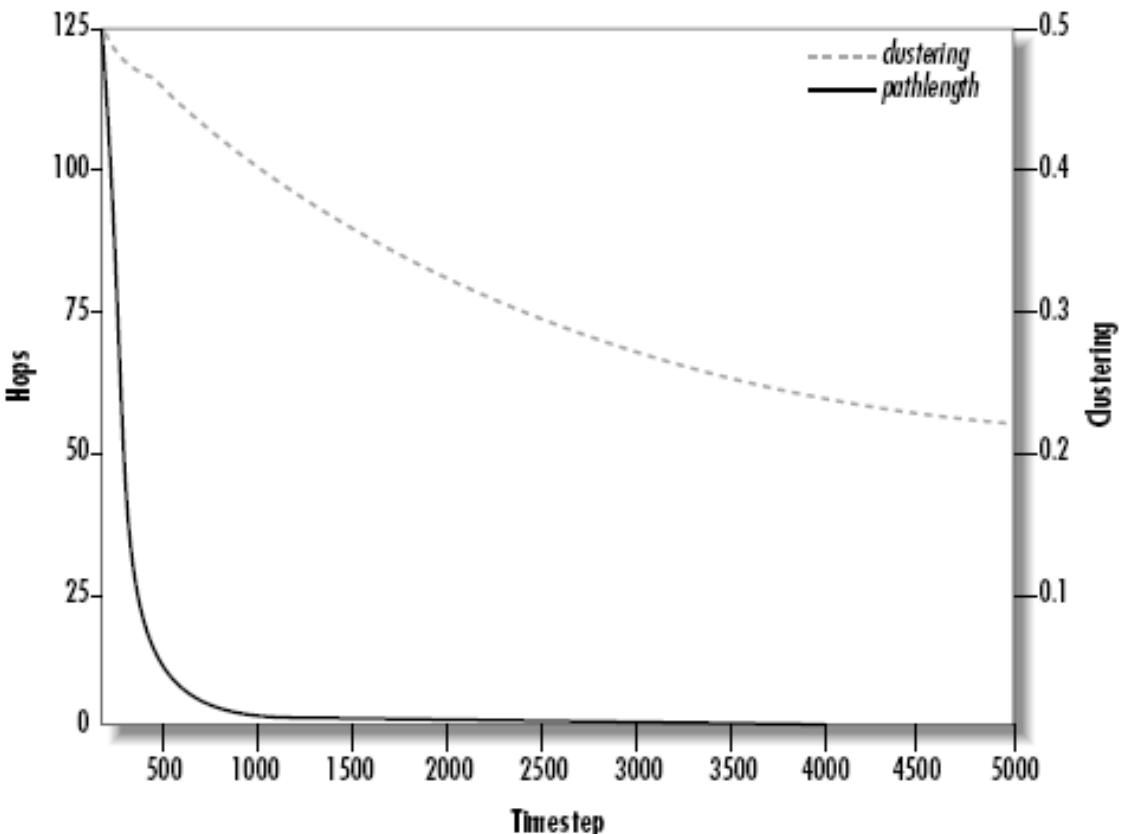
Example: Search Effect



- What is the effect that if the first search is node 0 searching for item 490 (assume no probabilistic replacement to hide origin)?
 - Nodes 0, 2, 4, 6, ..., 488 all cache item 490, and has a pointer to node 490
 - The search forms many **long-distance** links

Experiment: Evolution of Freenet Graph

- At each step
 - pick a node randomly
 - flip a coin to determine search or insert
 - if search, randomly pick a key in the network
 - if insert, pick a random key

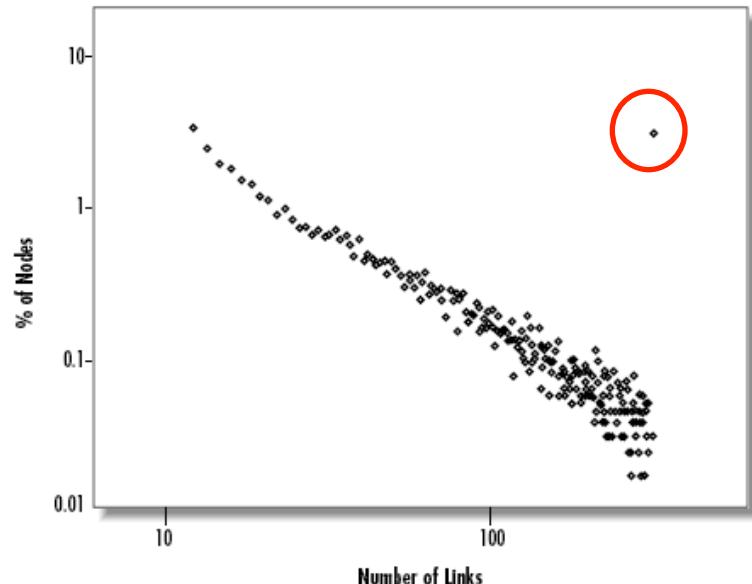
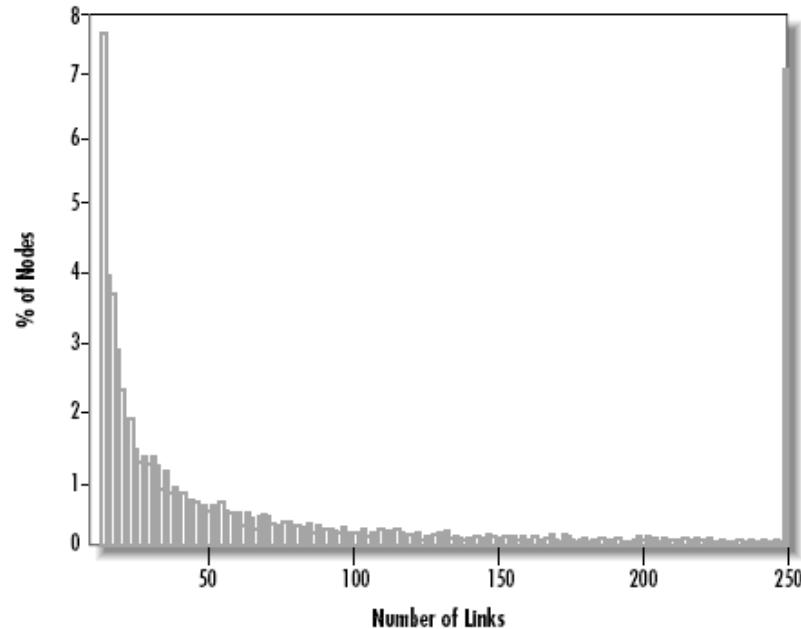


Evolution of path length and clustering;
Clustering is defined as percentage of local links

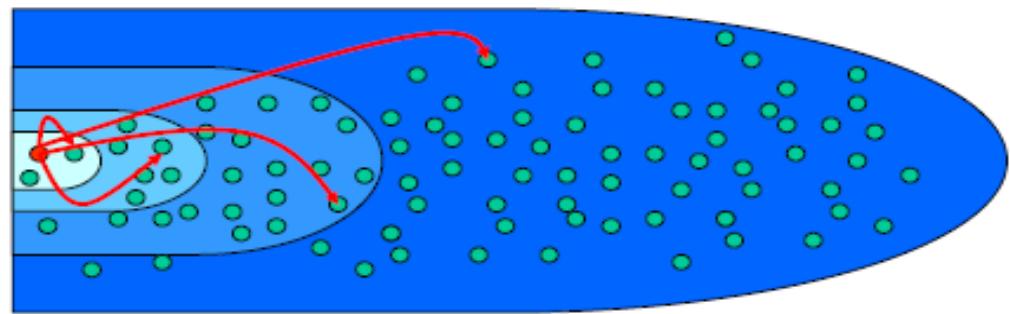
Freenet Evolves to Small-World Network

- With usage, the regular, highly localized Freenet network evolved into one irregular graph

- High percentage of highly connected nodes provide shortcuts/bridges
 - make the world a “small world”
 - most queries only traverse a small number of hops to find the file



Small-World



- First discovered by Milgram
 - in 1967, Milgram mailed 160 letters to a set of randomly chosen people in Omaha, Nebraska
 - goal: pass the letters to a given person in Boston
 - each person can only pass the letter to an intermediary known on a first-name basis
 - pick the person who may make the best progress
 - result: 42 letters made it through !
 - median intermediaries was 5.5---thus six degree of separation
 - a potential explanation: **highly connected** people with **non-local links** in mostly locally connected communities improve search performance !

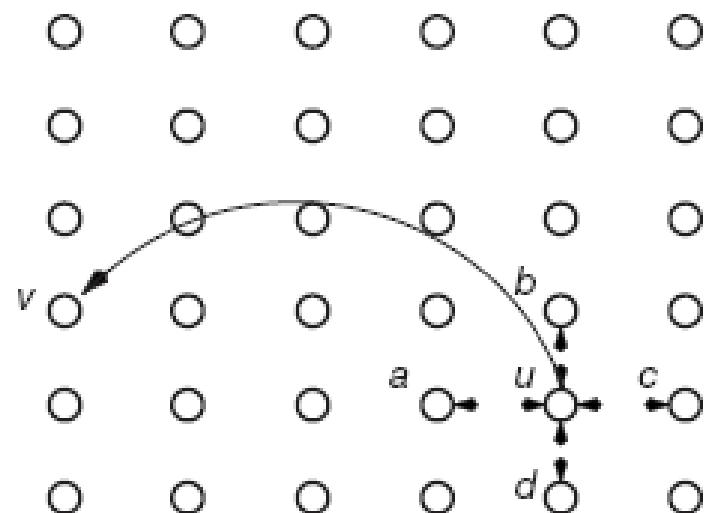
The Computer Networks

Distributed Search Question

- Question: what kind of long distance links to maintain so that distributed network search is effective?
- Assume that each node has
 - a fixed # (say p distance away) local links
 - a small # (say a total of q) long-distance links s.t. the probability of a link between nodes x and y is some (α) inverse-power of the distance $d(x, y)$ of x and y

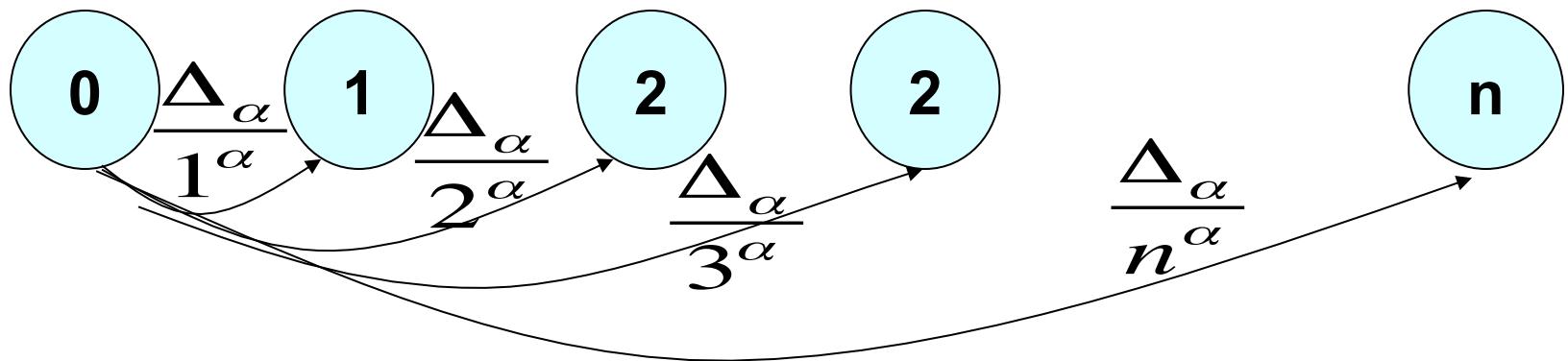
$$\frac{d(x, y)^{-\alpha}}{\Delta_\alpha}$$

- Different alpha's give diff types of links.
- Q: what is a good alpha?



What Should the Long-Distance Links Look?

- Consider the simple case of one dimensional space.



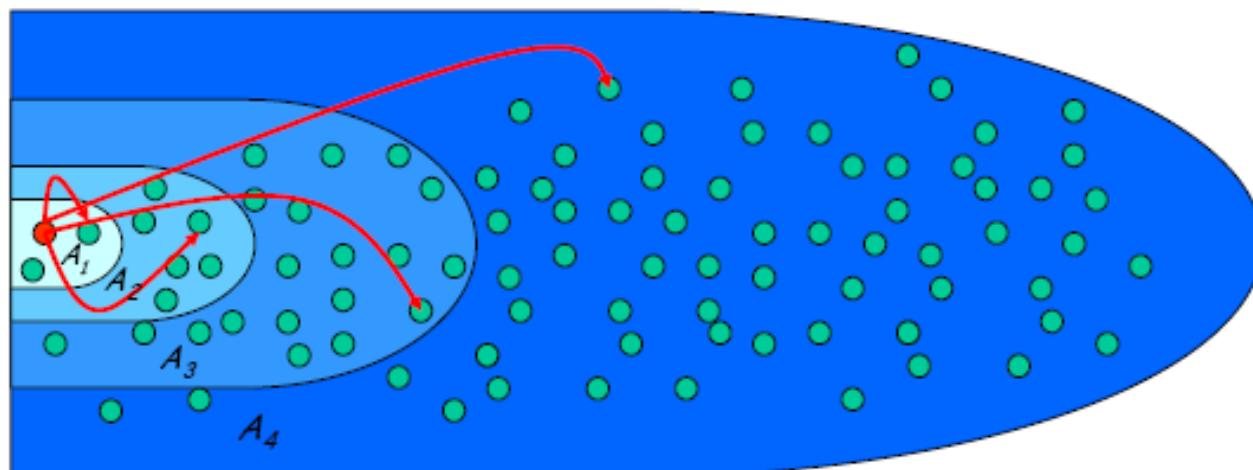
- Which alpha leads to best performing distributed search alg?

What Should the Long-Distance Links Look?

- For 1-d space, for any distributed algorithm, the expected # of search steps, for different α 's:
 - $0 \leq \alpha < 1$: $\geq k_1 n^{(1-\alpha)/2}$
 - $\alpha > 1$: $\geq k_1 n^{(\alpha-1)/\alpha}$
 - $\alpha = 1$: $O(\log^2 n)$ greedy search

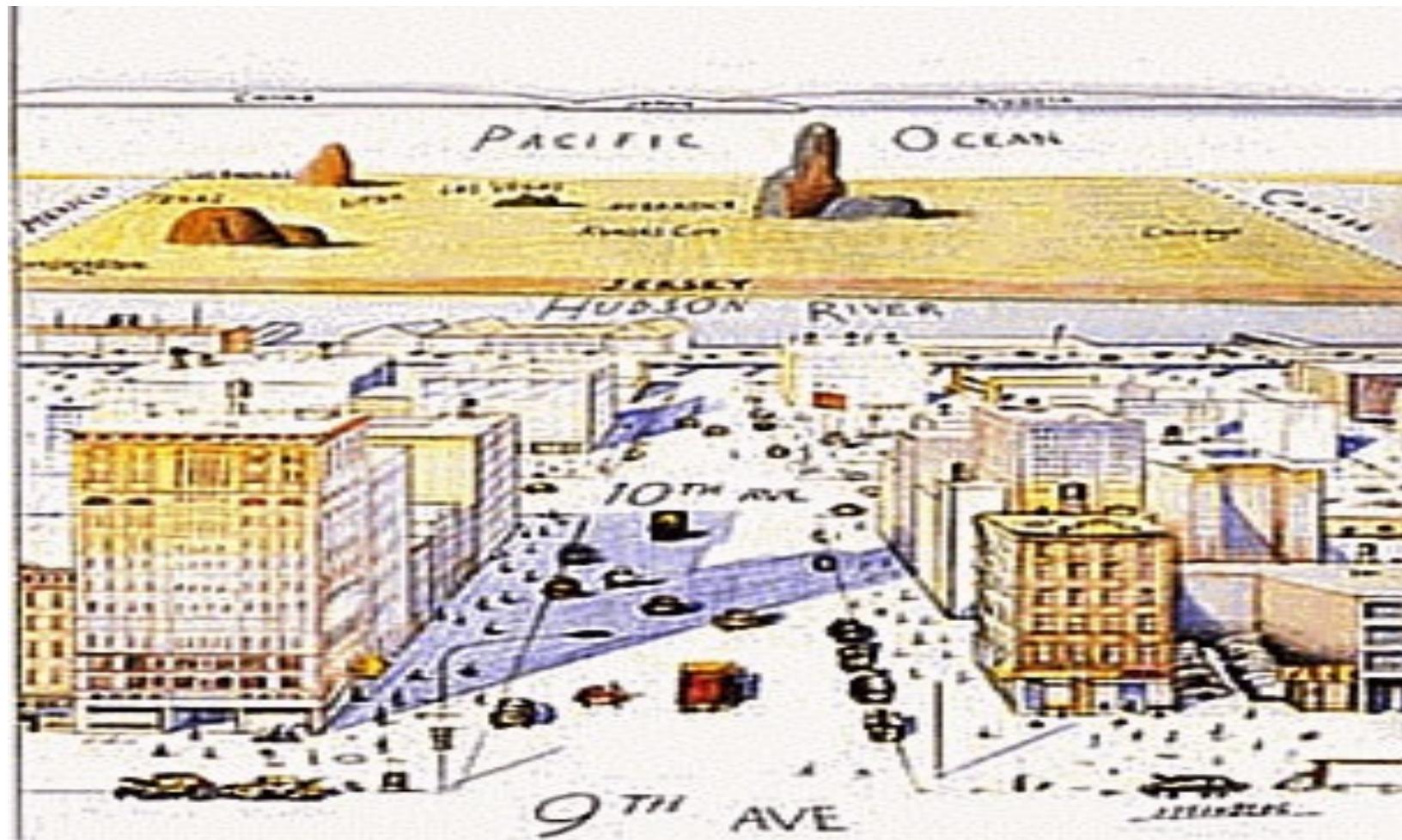
Distributed Search

- In the general case, α should be the dimension
- In other words, a guideline on long-distance links: roughly the same number of nodes in each region A_1 , A_2 , A_4 , A_8 , where A_1 is the set of nodes who are one lattice step away, A_2 is those two steps away, A_4 is four steps away...



probability is proportional to $(\text{lattice steps})^{-d}$

Small World



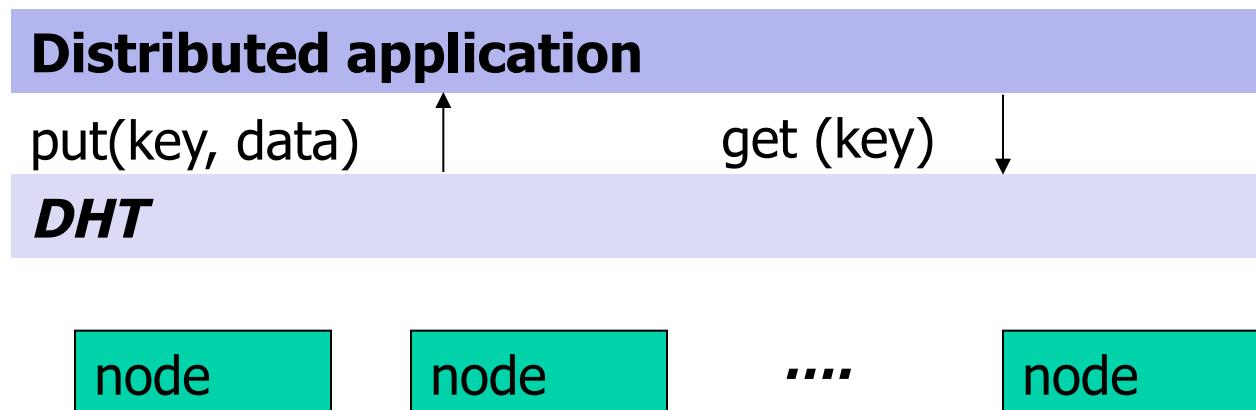
Saul Steinberg; View of World from 9th Ave

Freenet: Issues

- Does **not** always guarantee that a file is found, even if the file is in the network
- Good average-case performance, but a potentially **long search path** in a large network
 - approaching small-world...

DHT: Overview

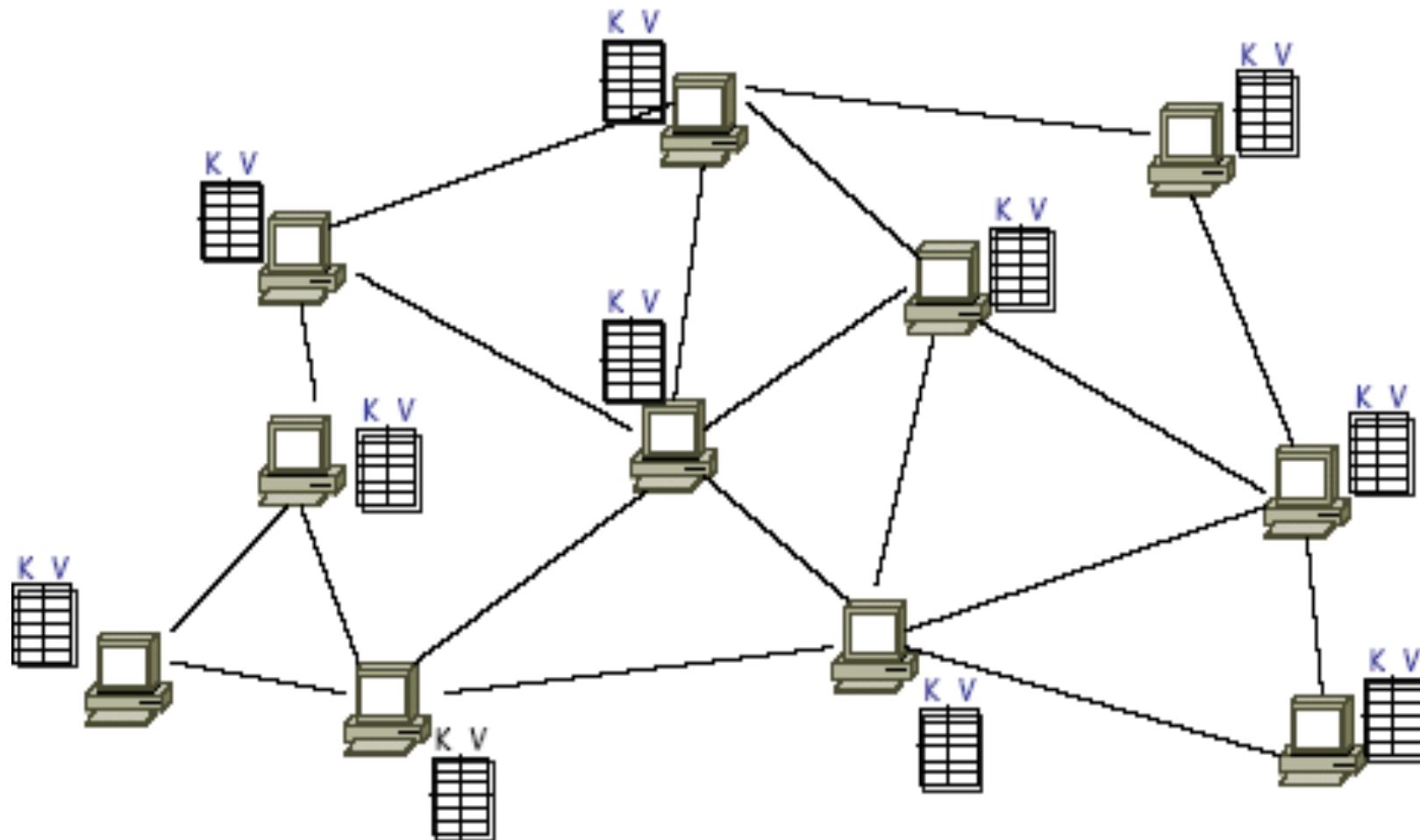
- Abstraction: a distributed “hash-table” (DHT) data structure
 - `put(key, value)` and `get(key) → value`
 - DHT imposes no structure/meaning on keys
 - one can build complex data structures using DHT
- Implementation:
 - nodes in system form an interconnection network: ring, zone, tree, hypercube, butterfly network, ...



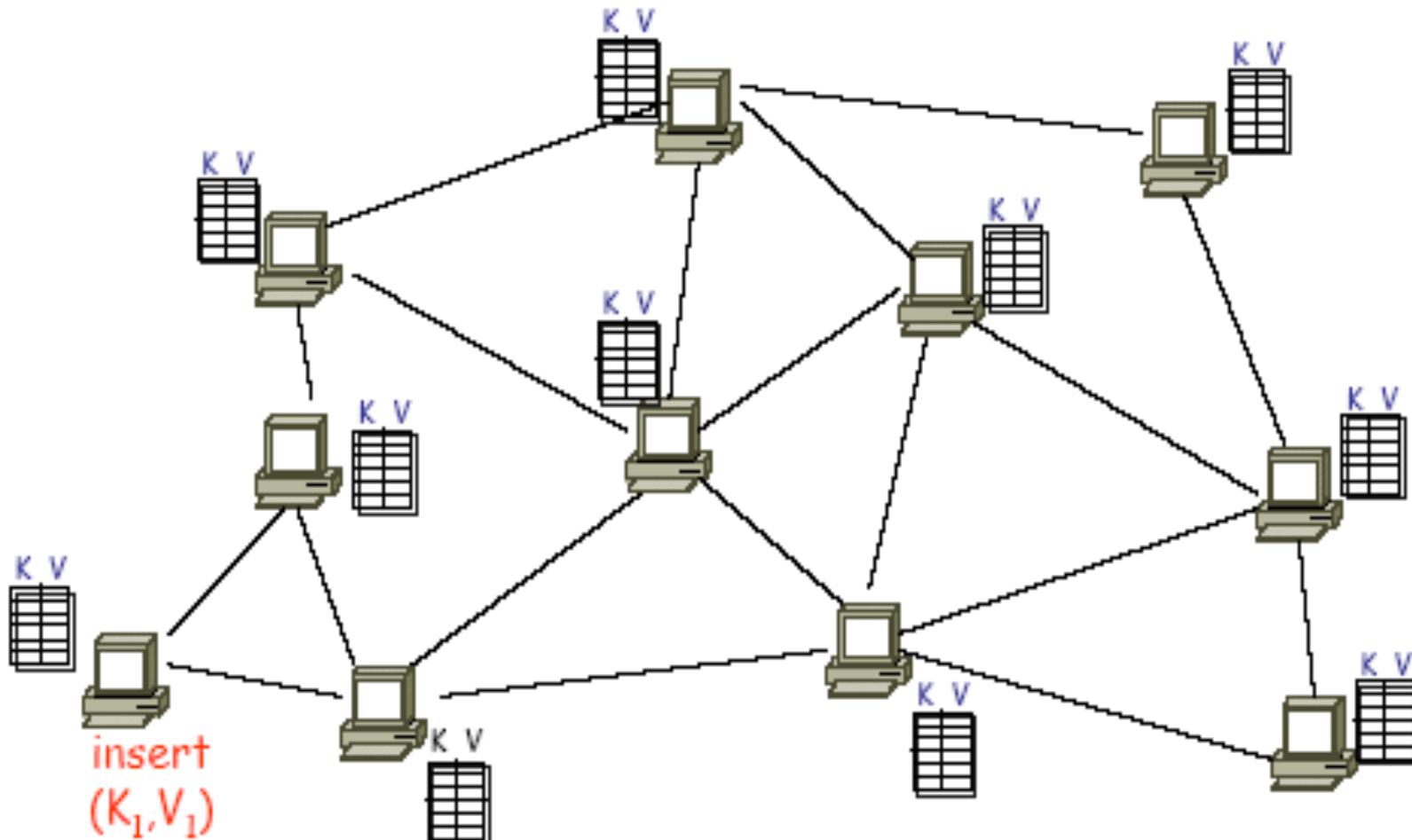
DHT Applications

- File sharing and backup [CFS, Ivy, OceanStore, PAST, Pastiche ...]
- Web cache and replica [Squirrel, Croquet Media Player]
- Censor-resistant stores [Eternity]
- DB query and indexing [PIER, Place Lab, VPN Index]
- Event notification [Scribe]
- Naming systems [ChordDNS, Twine, INS, HIP]
- Communication primitives [I3, ...]
- Host mobility [DTN Tetherless Architecture]

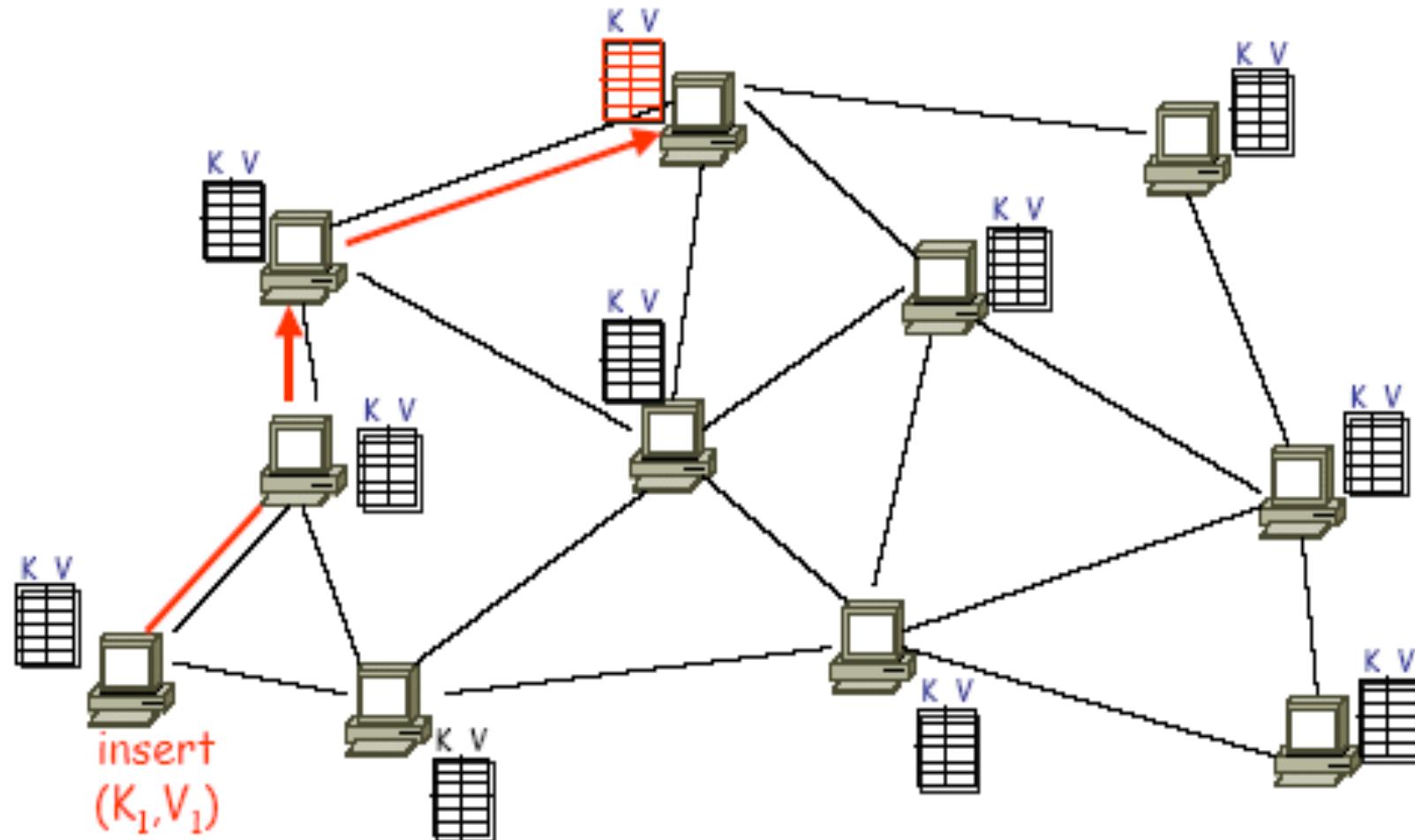
DHT: Basic Idea



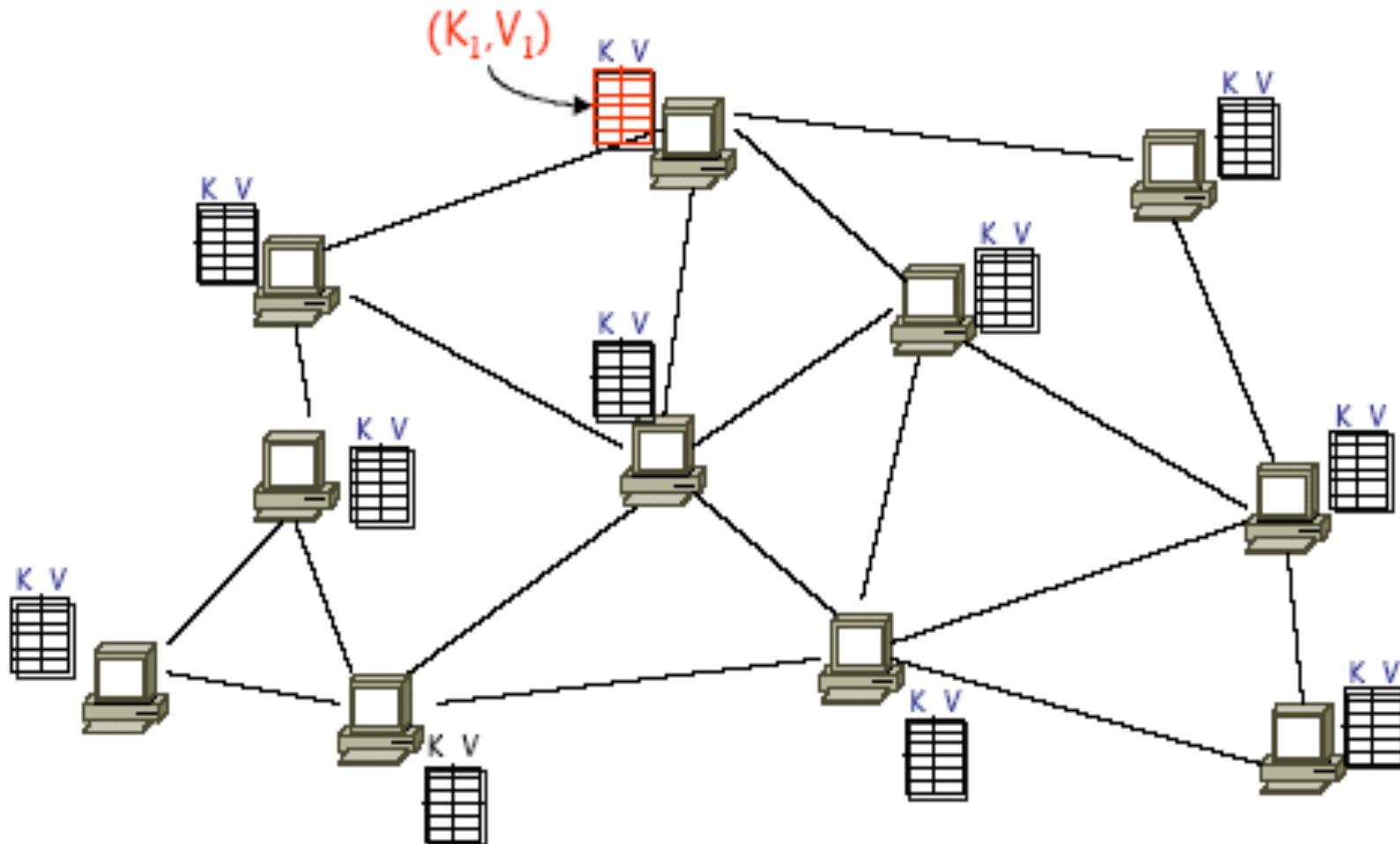
DHT: Basic Idea (2)



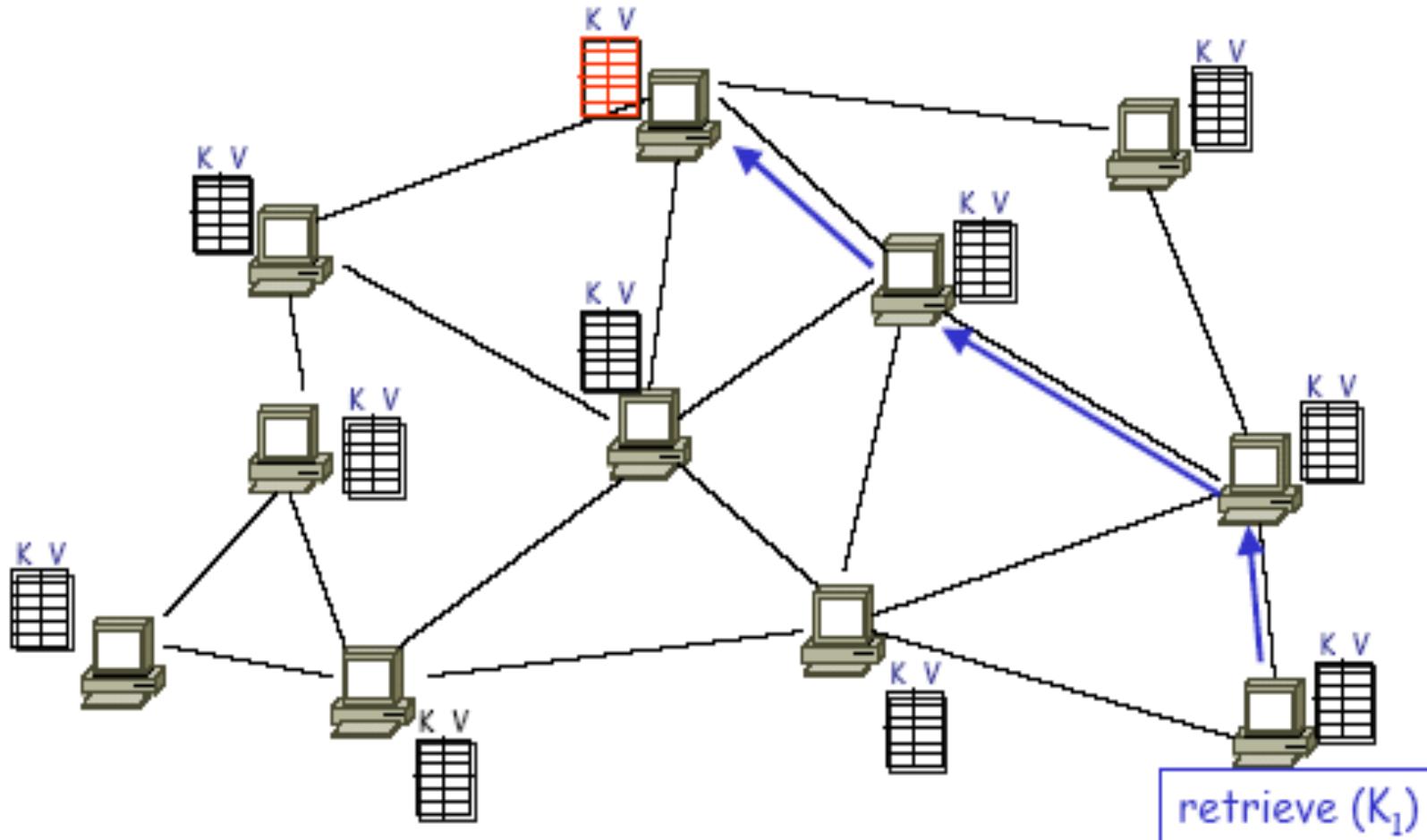
DHT: Basic Idea (3)



DHT: Basic Idea (4)



DHT: Basic Idea (5)



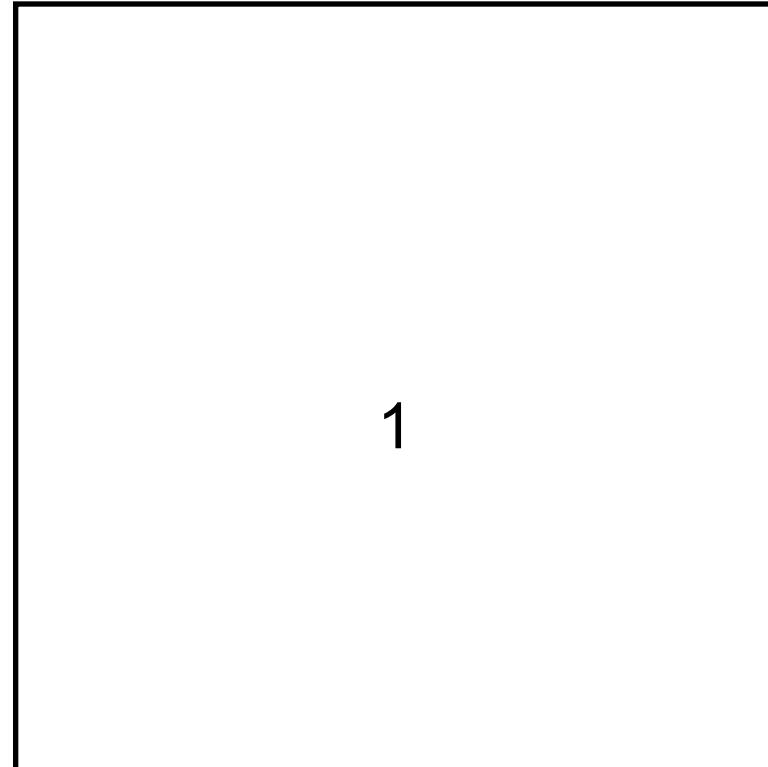
CAN

□ Abstraction

- map a key to a “point” in a **multi-dimensional Cartesian space**
- a node “owns” a **zone** in the overall space
- route from one “point” to another

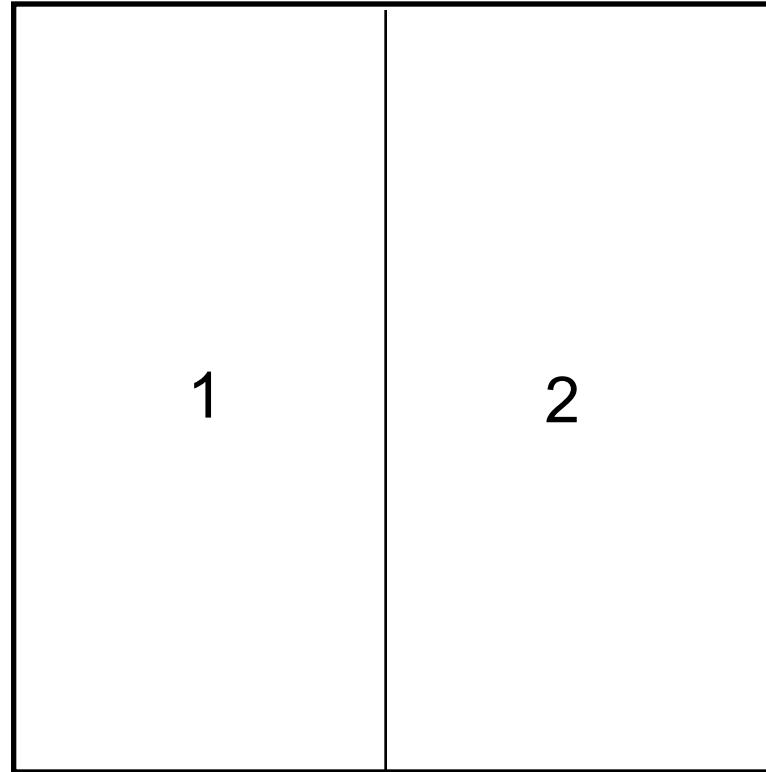
CAN Example: Two Dimensional Space

- Space divided among nodes
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1



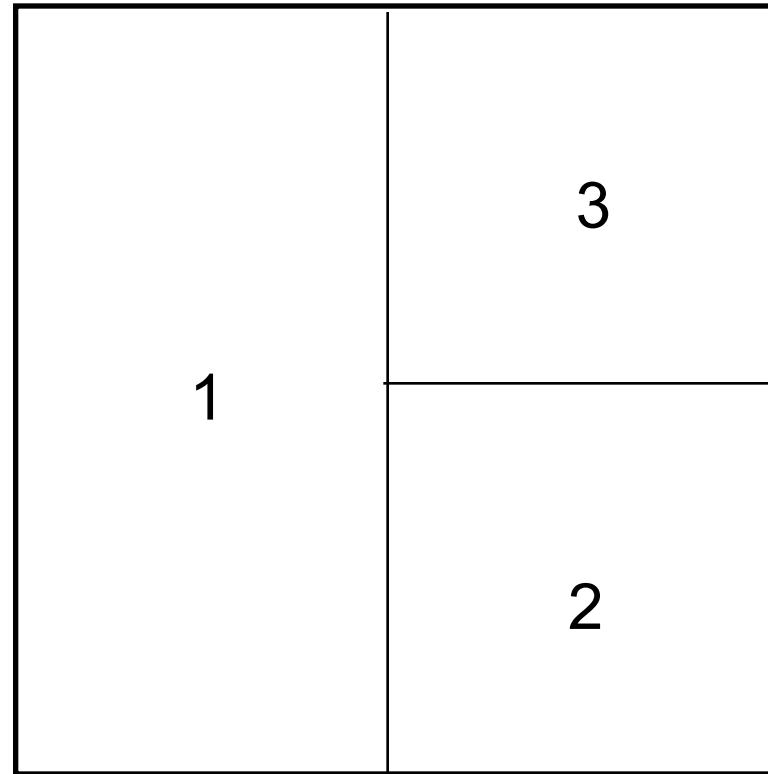
CAN Example: Two Dimensional Space

- Space divided among nodes
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1



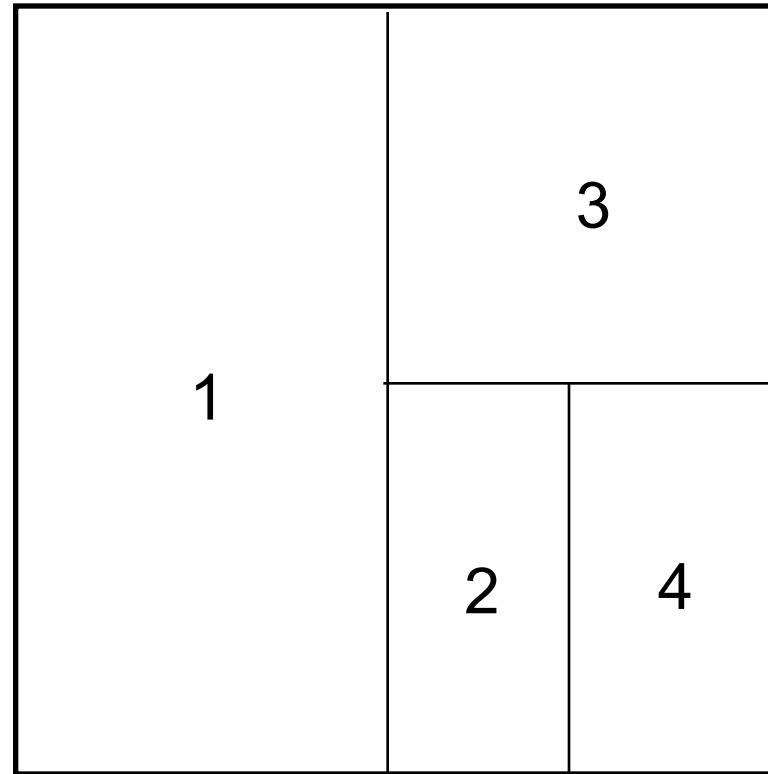
CAN Example: Two Dimensional Space

- Space divided among nodes
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1



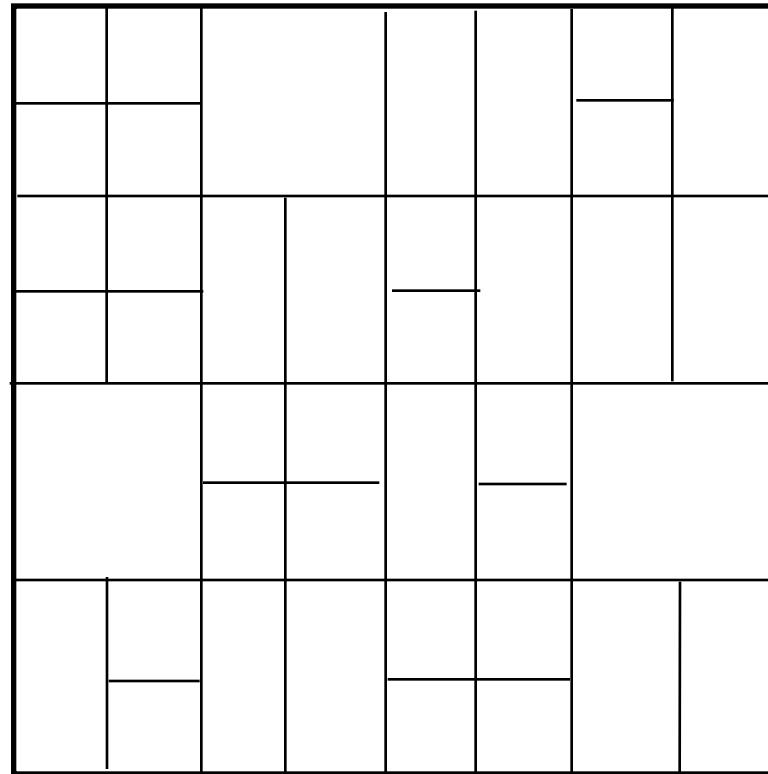
CAN Example: Two Dimensional Space

- Space divided among nodes
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1



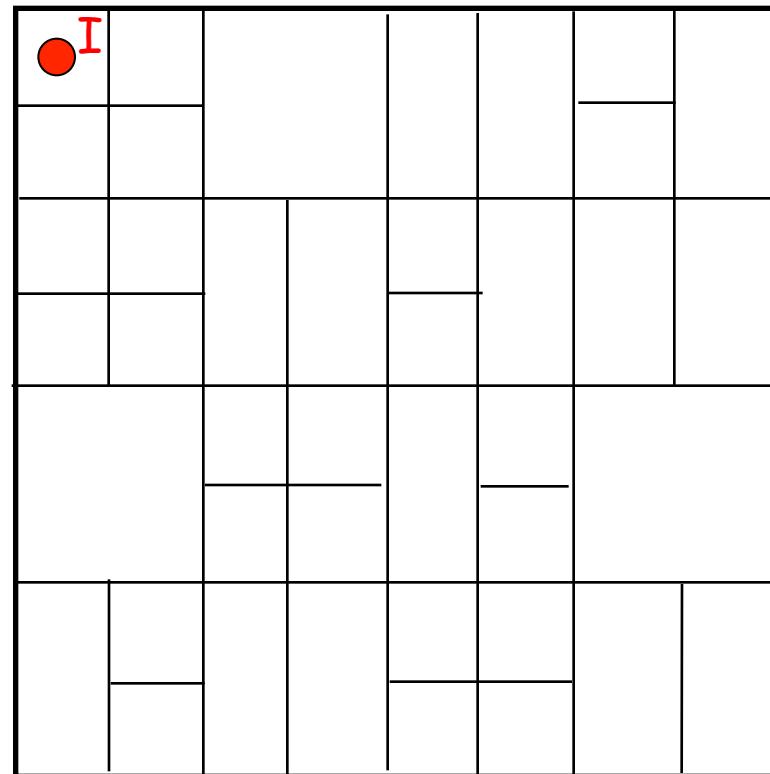
CAN Example: Two Dimensional Space

- Space divided among nodes
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1



CAN Insert: Example (1)

node I::insert(K,V)

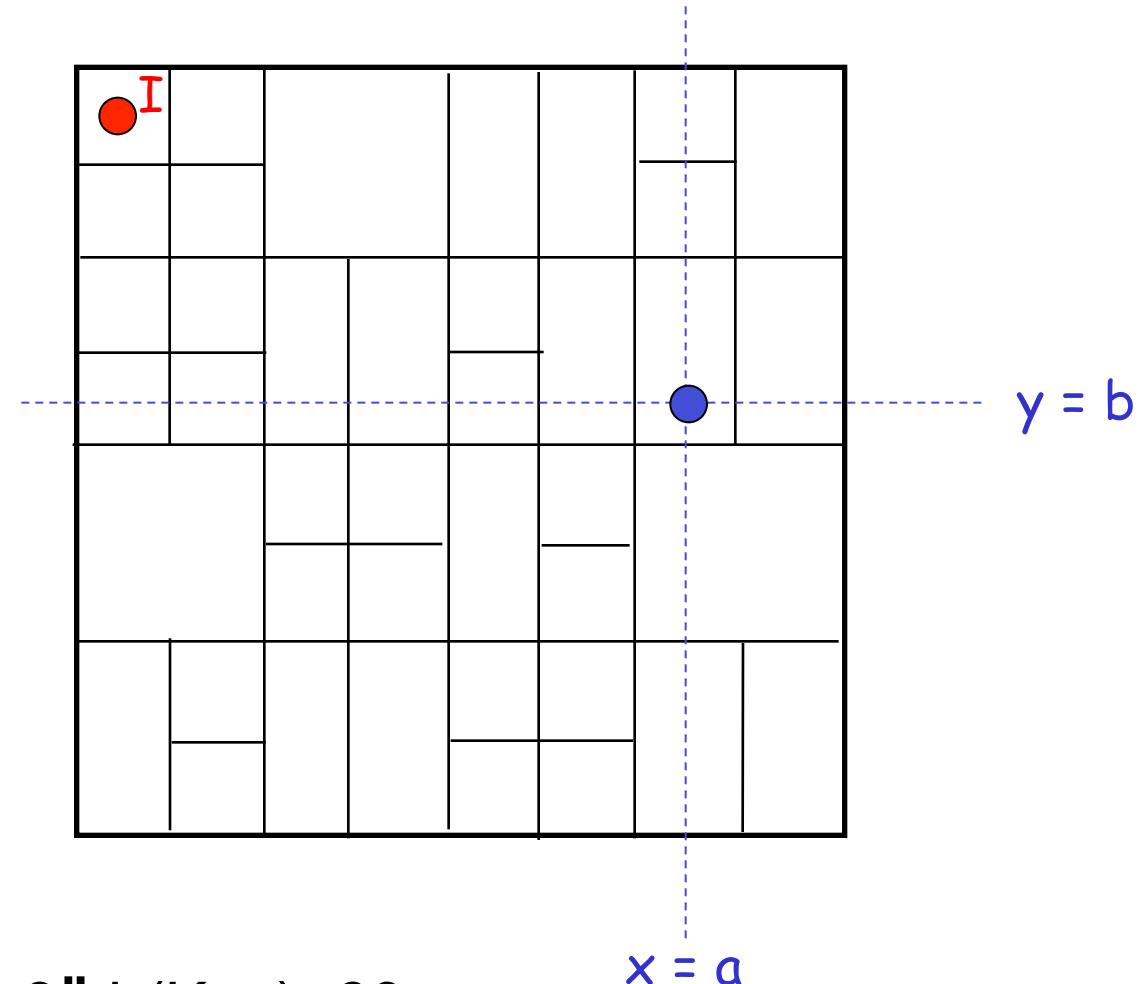


CAN Insert: Example (2)

node I::insert(K,V)

(1) $a = h_x(K)$

$b = h_y(K)$



Example: Key=“Matrix3” $h(\text{Key})=60$

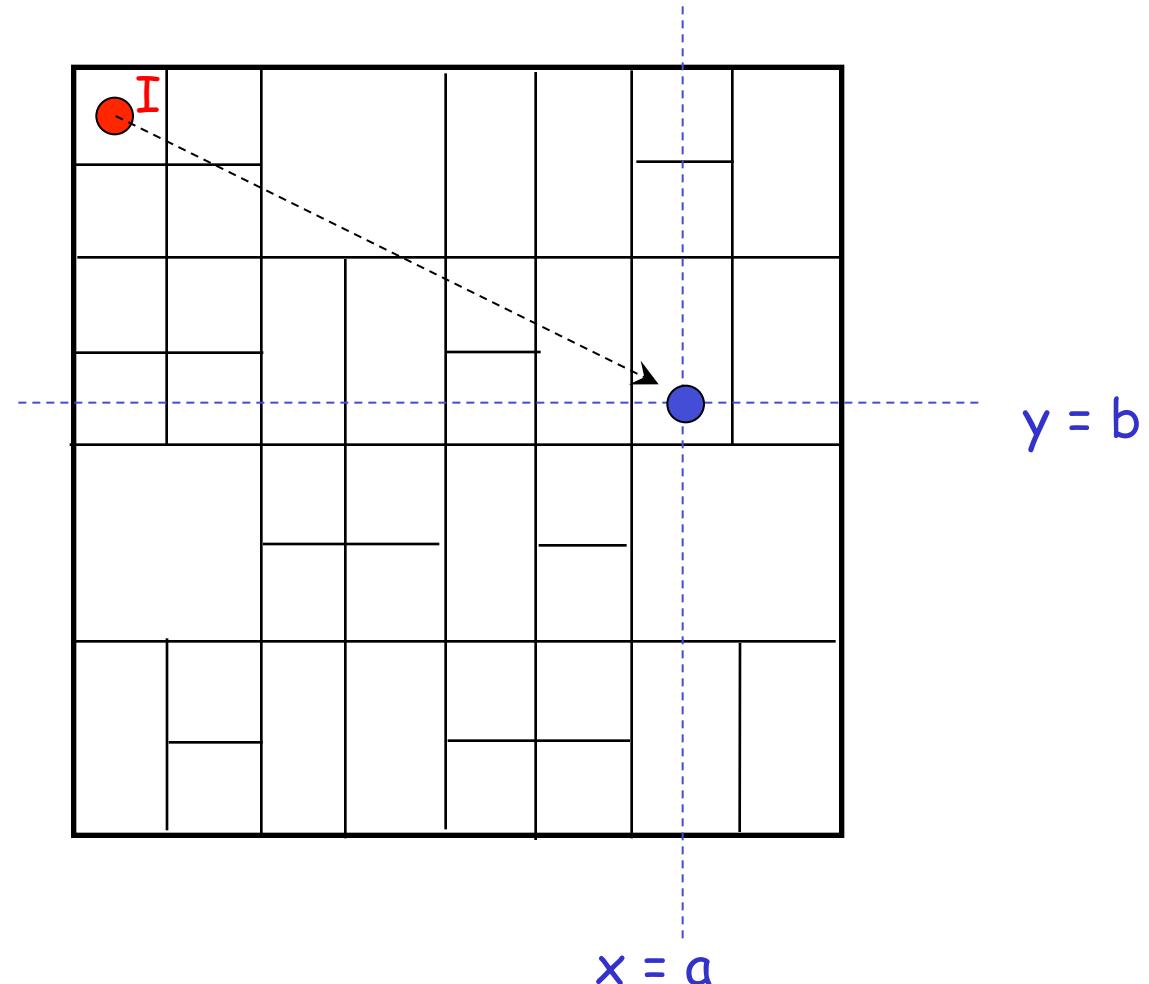
CAN Insert: Example (3)

node I::insert(K,V)

(1) $a = h_x(K)$

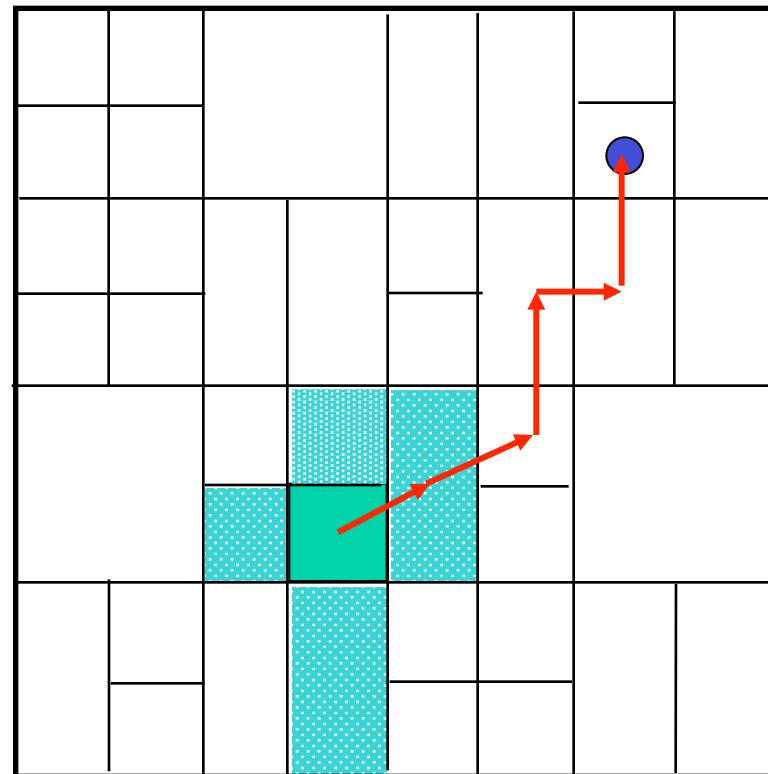
$b = h_y(K)$

(2) route(K,V) $\rightarrow (a,b)$



CAN Insert: Routing

- A node maintains state only for its immediate neighboring nodes
- Forward to neighbor which is closest to the target point
 - a type of greedy, local routing scheme



CAN Insert: Example (4)

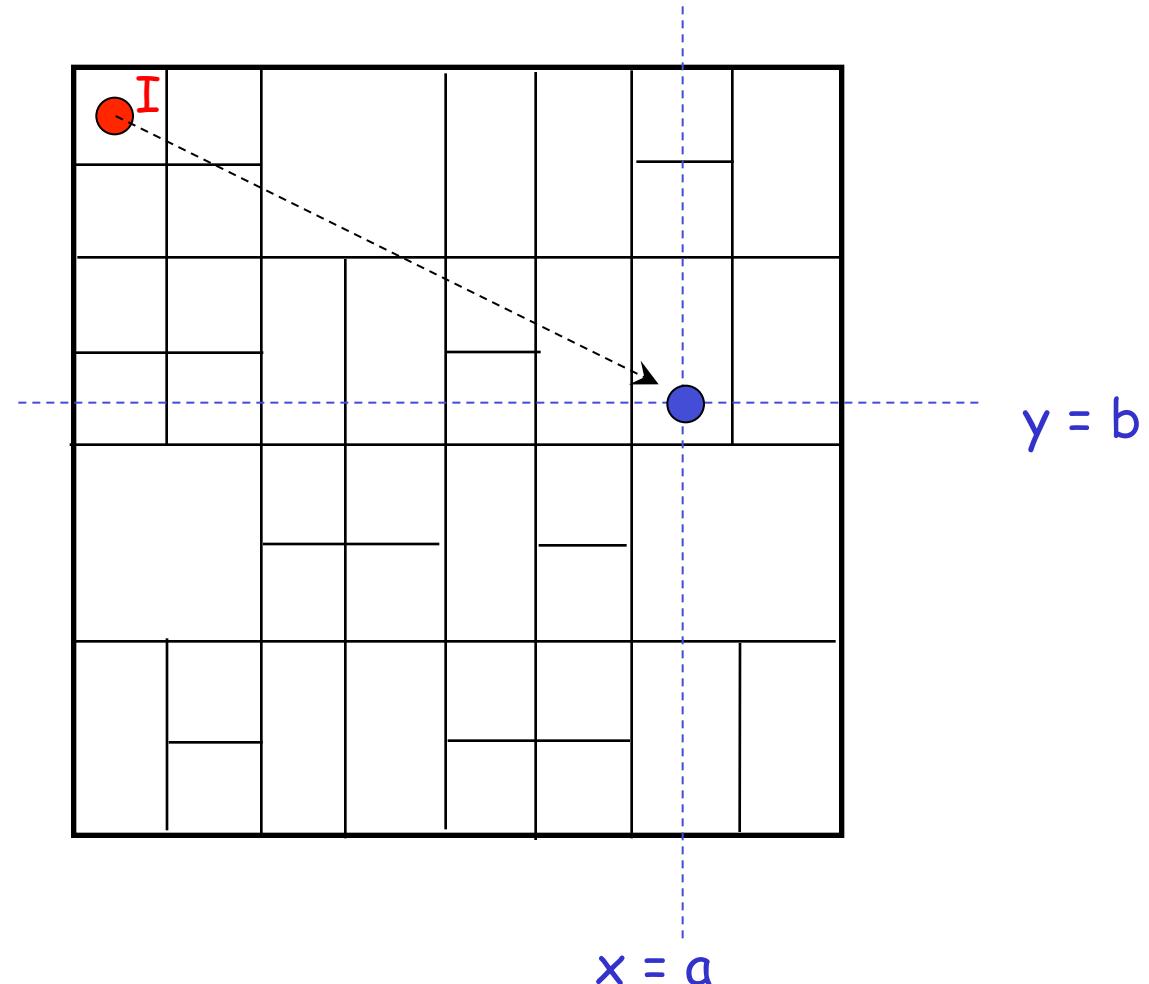
node I::insert(K,V)

(1) $a = h_x(K)$

$b = h_y(K)$

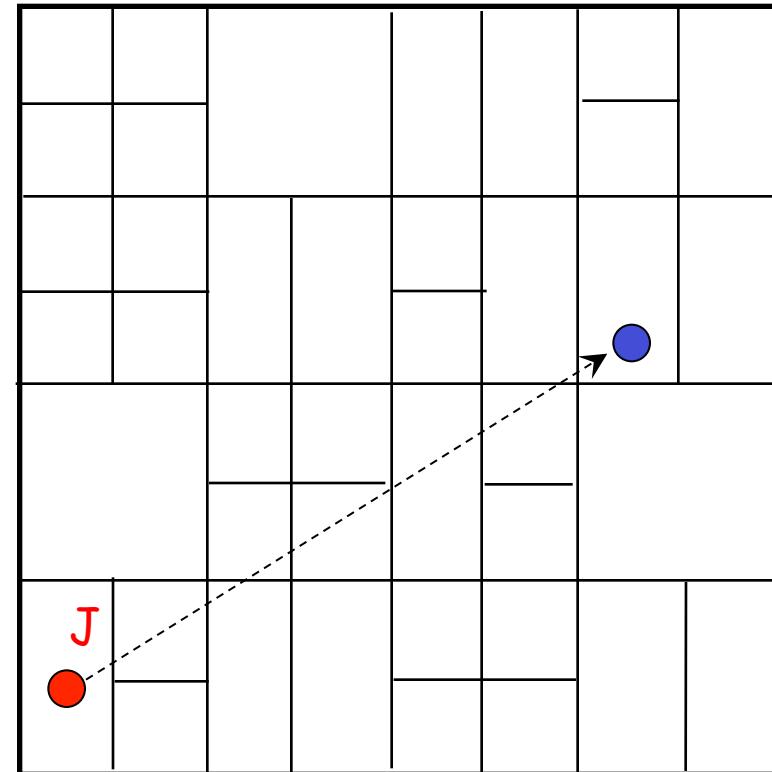
(2) route(K,V) $\rightarrow (a,b)$

(3) (K,V) is stored at
 (a,b)



CAN Retrieve: Example

node J::retrieve(K)



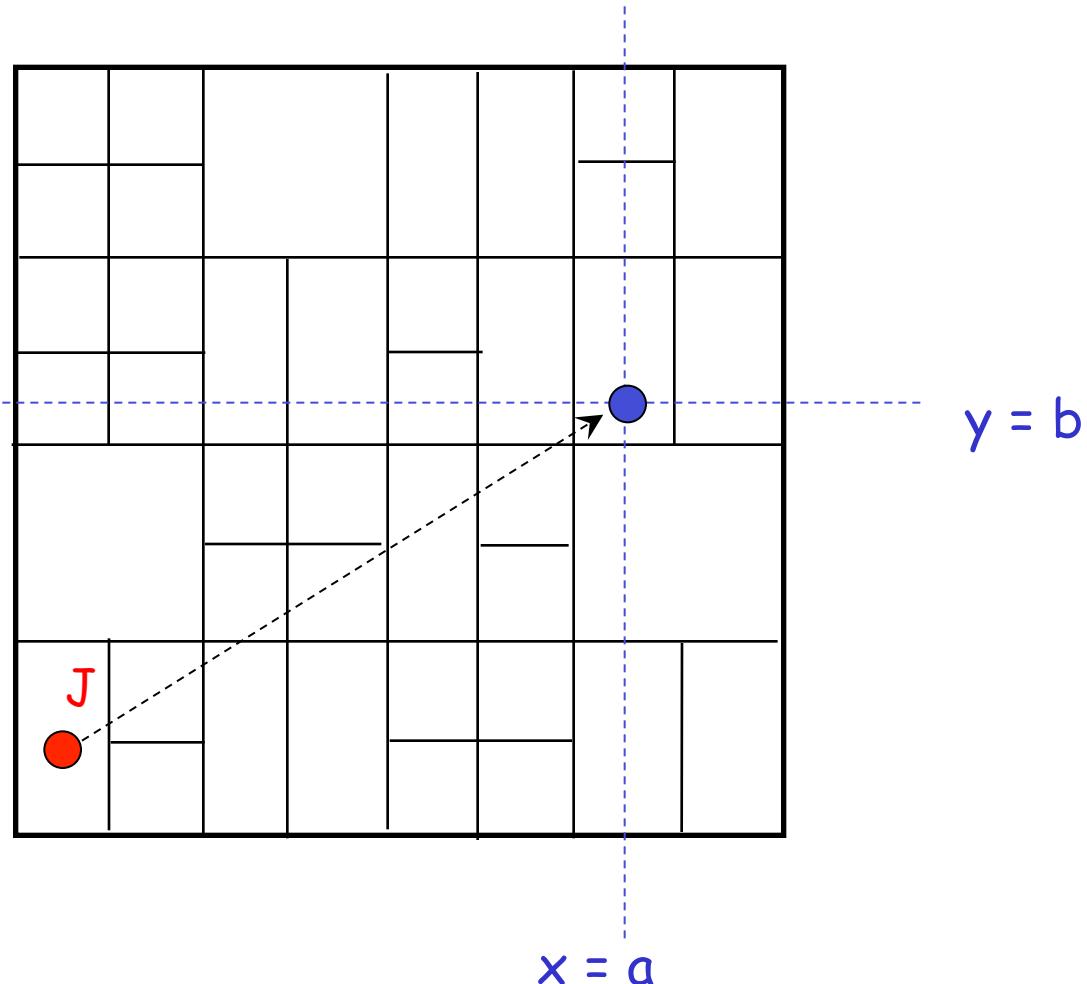
CAN Retrieve: Example

node J::retrieve(K)

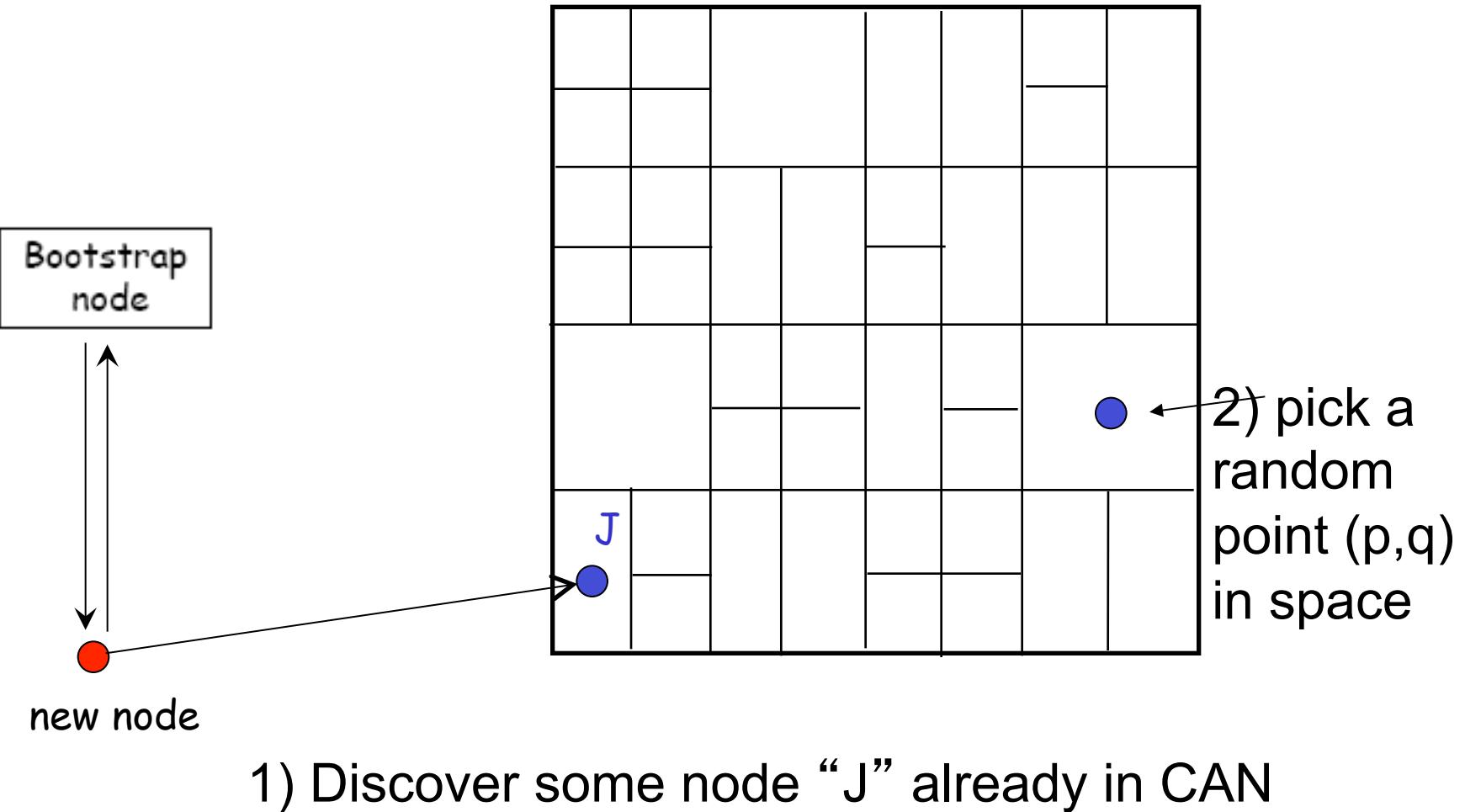
(1) $a = h_x(K)$

$b = h_y(K)$

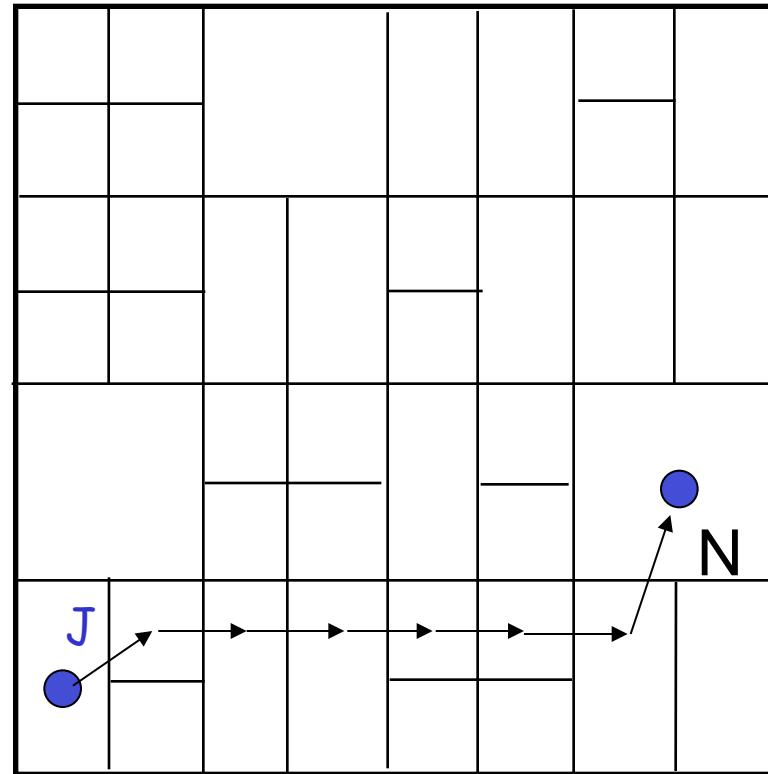
(2) route "retrieve(K)" to
(a,b)



CAN Insert: Join (1)



CAN Insert: Join (2)

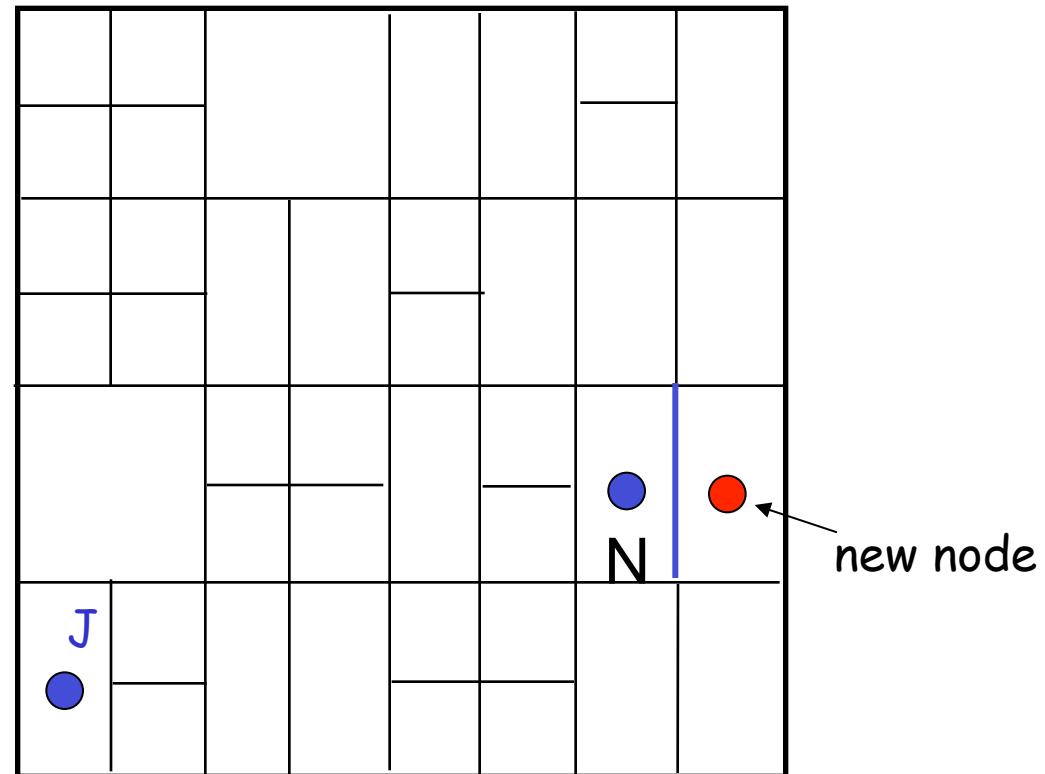


new node

3) J routes to (p, q) , discovers node N

CAN Insert: Join (3)

Inserting a new node affects only a single other node and its immediate neighbors



4) split N's zone in half... new node owns one half

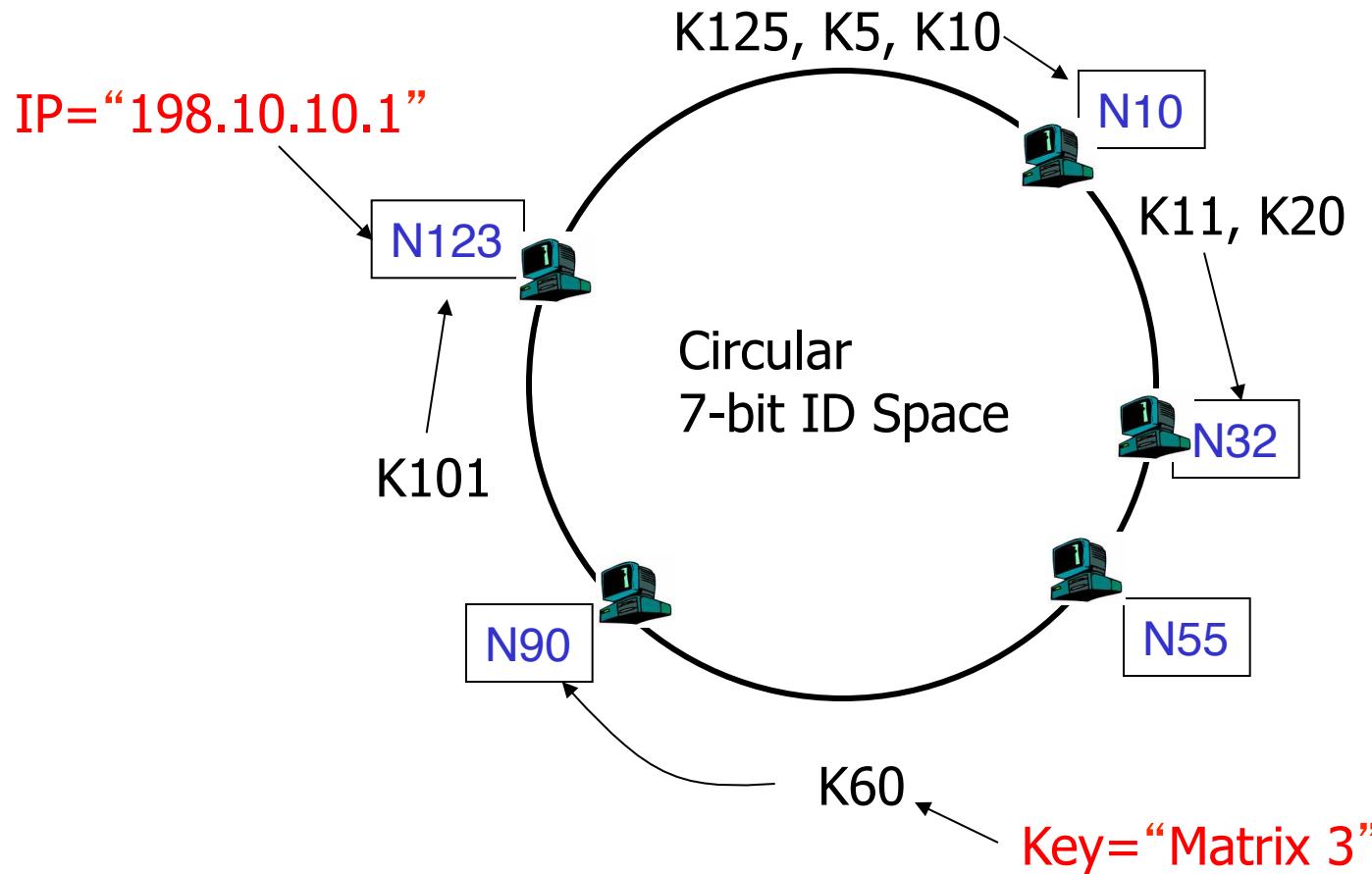
CAN Evaluations

- Guarantee to find an item if in the network
- Load balancing
 - hashing achieves some load balancing
 - overloaded node replicates popular entries at neighbors
- Scalability
 - for a uniform (regularly) partitioned space with **n** nodes and **d** dimensions
 - storage:
 - per node, number of neighbors is **2d**
 - routing
 - average routing path is $(dn^{1/d})/3$ hops (due to Manhattan distance routing, expected hops in each dimension is dimension length * 1/3)
 - a fixed d can scale the network without increasing per-node state

Chord

- Space is a ring
- Consistent hashing: m bit identifier space for both keys and nodes
 - key identifier = SHA-1(key), where SHA-1() is a popular hash function,
Key=“Matrix3” → ID=60
 - node identifier = SHA-1(IP address)
 - IP=“198.10.10.1” → ID=123

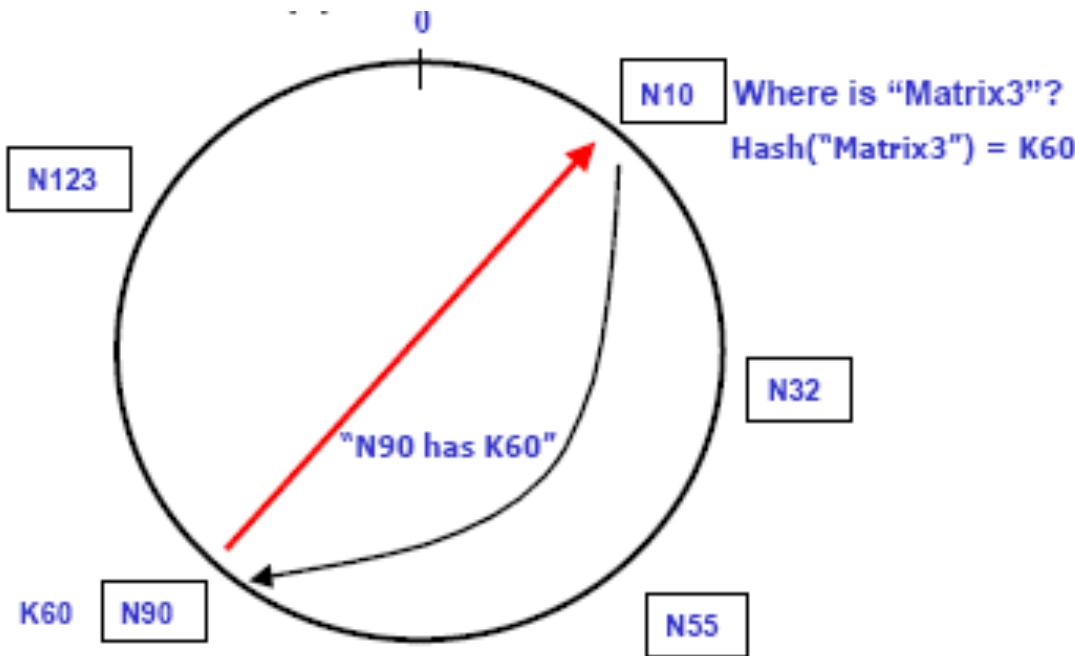
Chord: Storage using a Ring



- A key is stored at its successor: node with next higher or equal ID

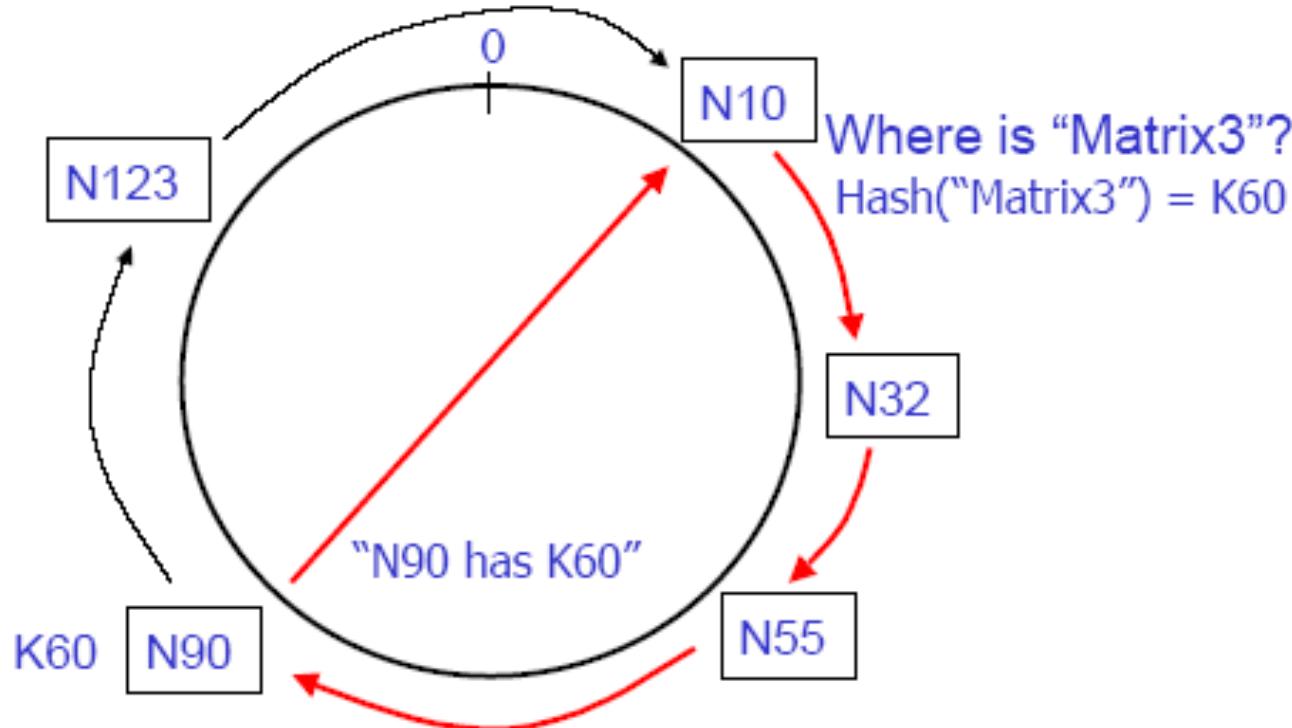
How to Search: One Extreme

- ❑ Every node knows of every other node
- ❑ Routing tables are large $O(N)$
- ❑ Lookups are fast $O(1)$



How to Search: the Other Extreme

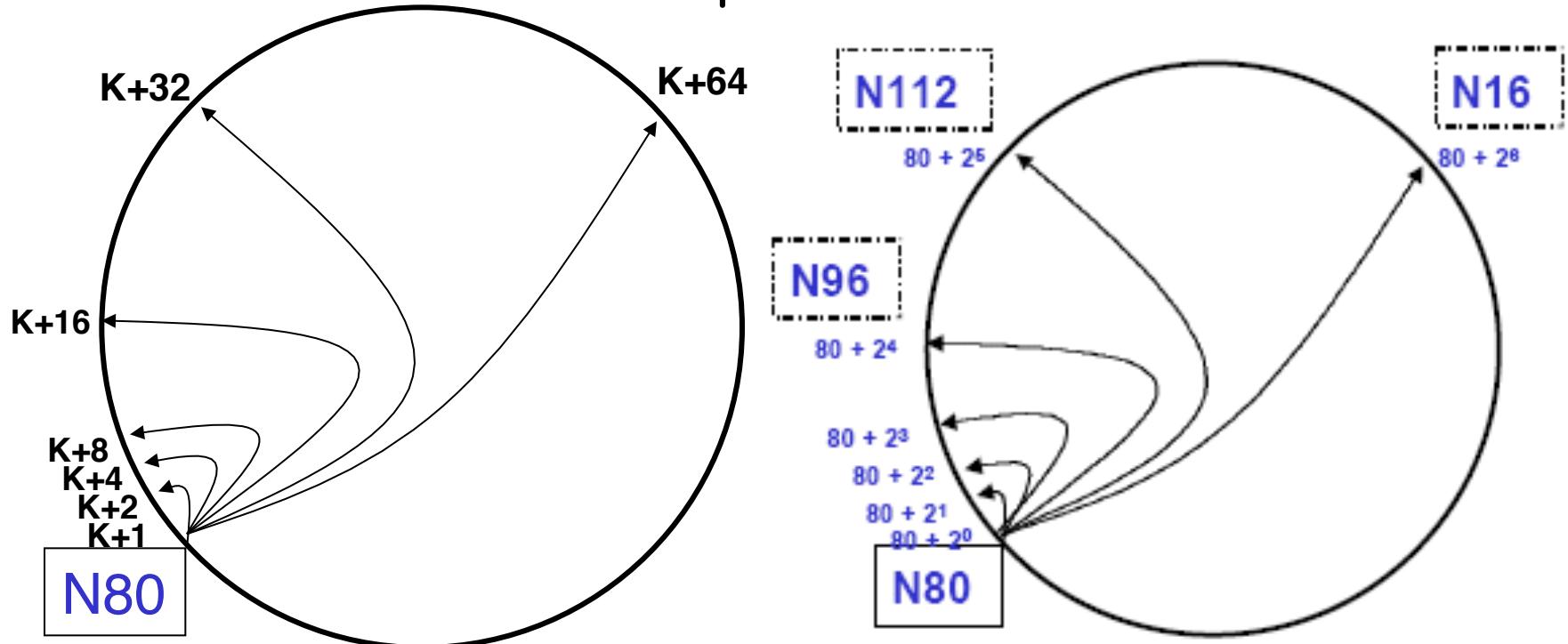
- ❑ Every node knows its successor in the ring



- ❑ Routing tables are small $O(1)$
- ❑ Lookups are slow $O(N)$

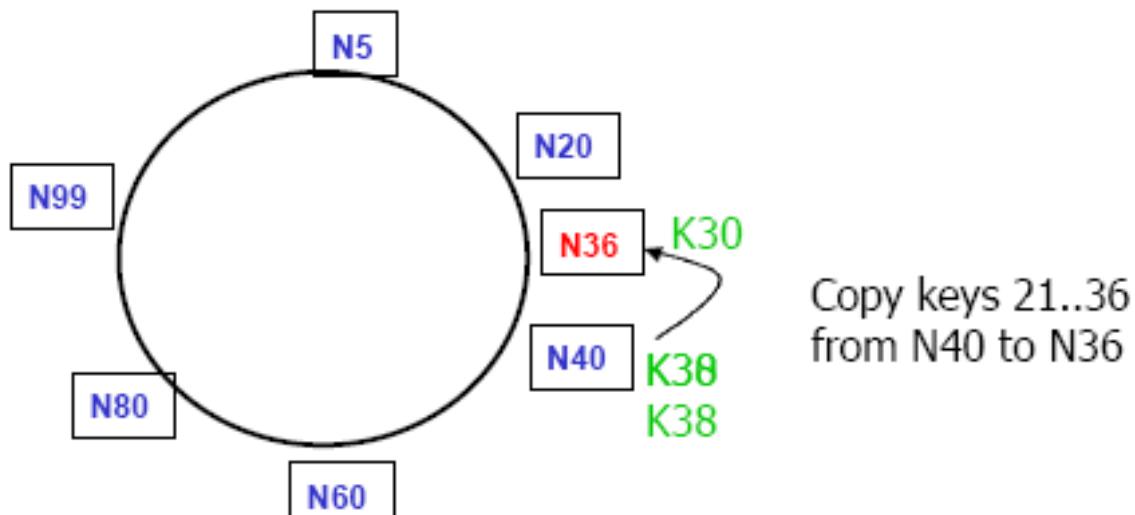
Chord Solution: "finger tables"

- Node K knows the node that is maintaining $K + 2^i$, where K is mapped id of current node
 - increase distance exponentially



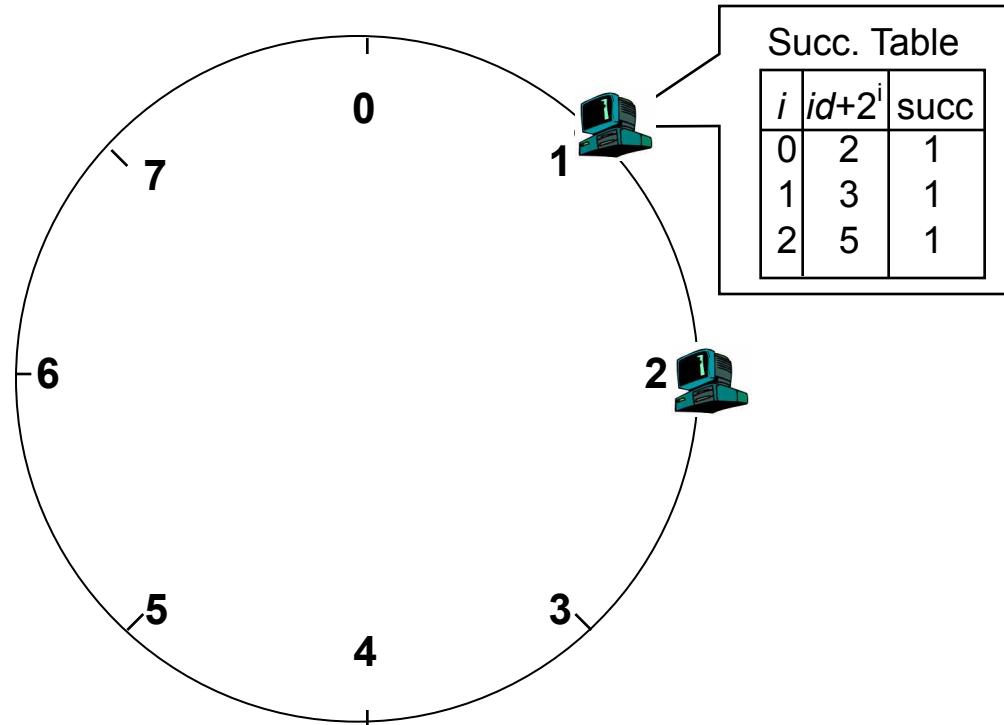
Joining the Ring

- ❑ use a contact node to obtain info
- ❑ transfer keys from successor node to new node
- ❑ updating fingers of existing nodes



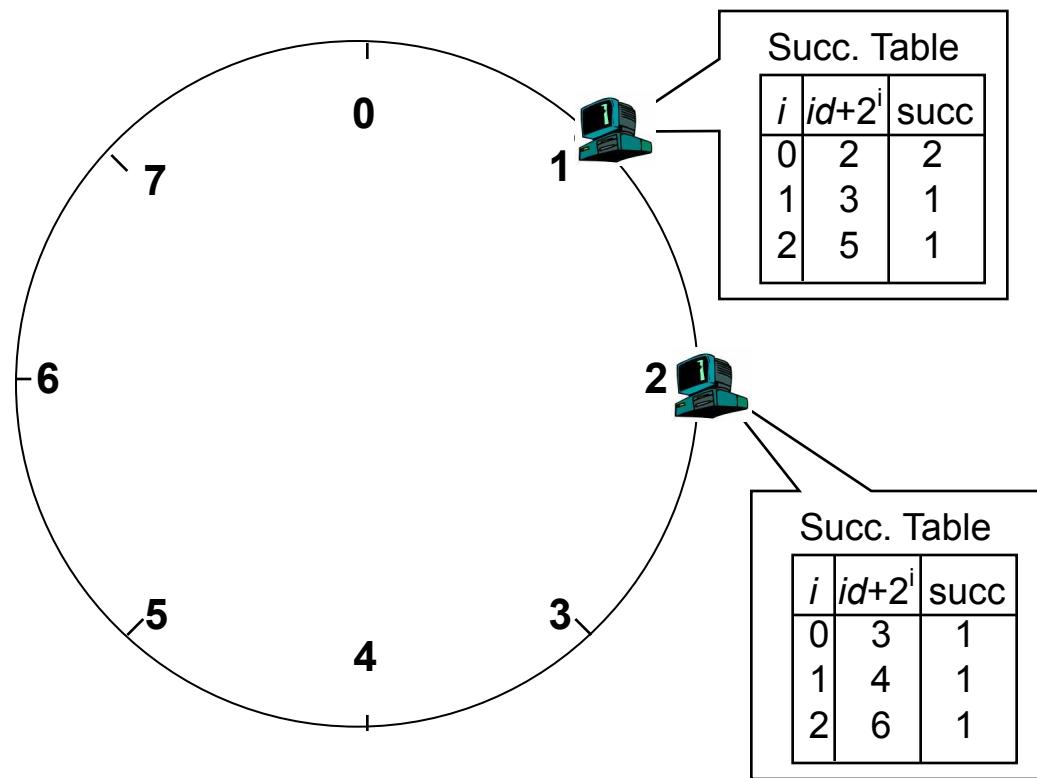
DHT: Chord Node Join

- Assume an identifier space [0..8]
- Node n1 joins



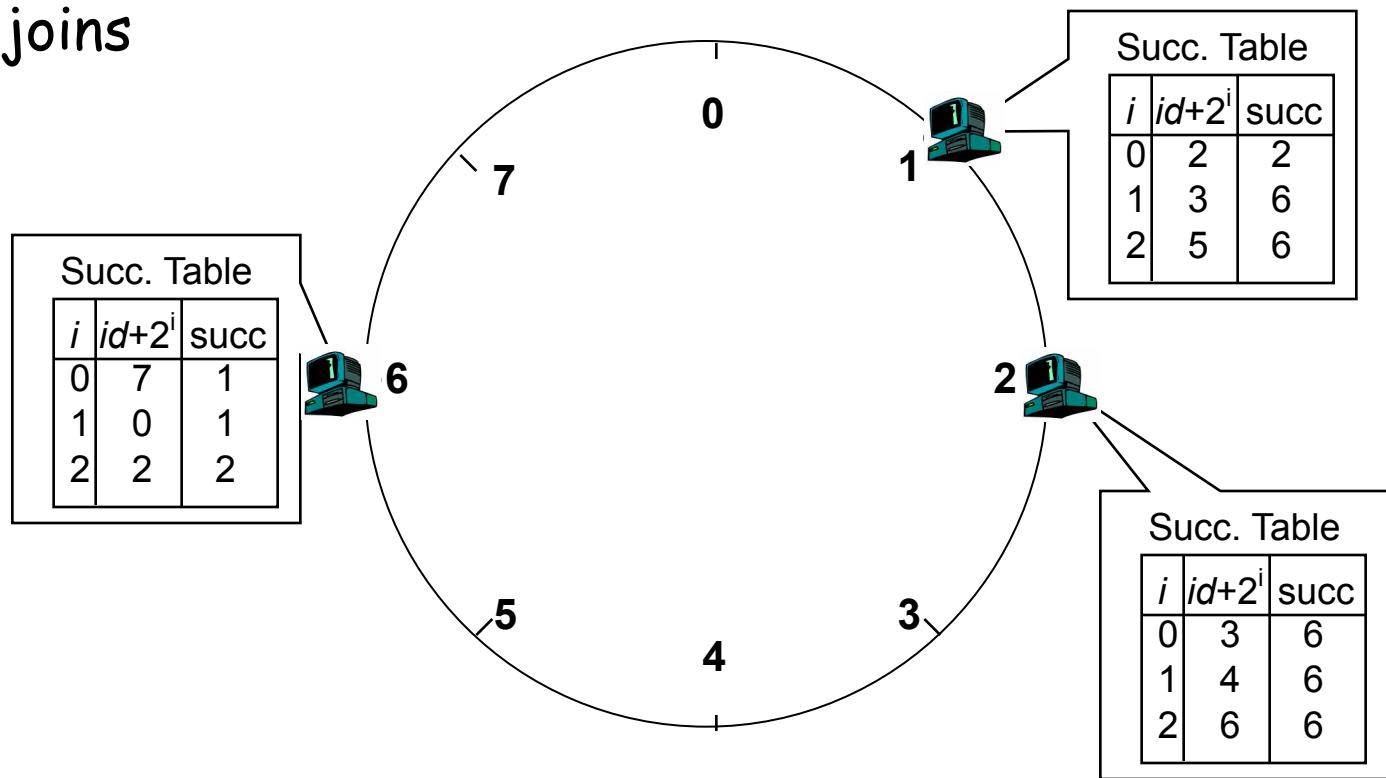
DHT: Chord Node Join

- Node n2 joins



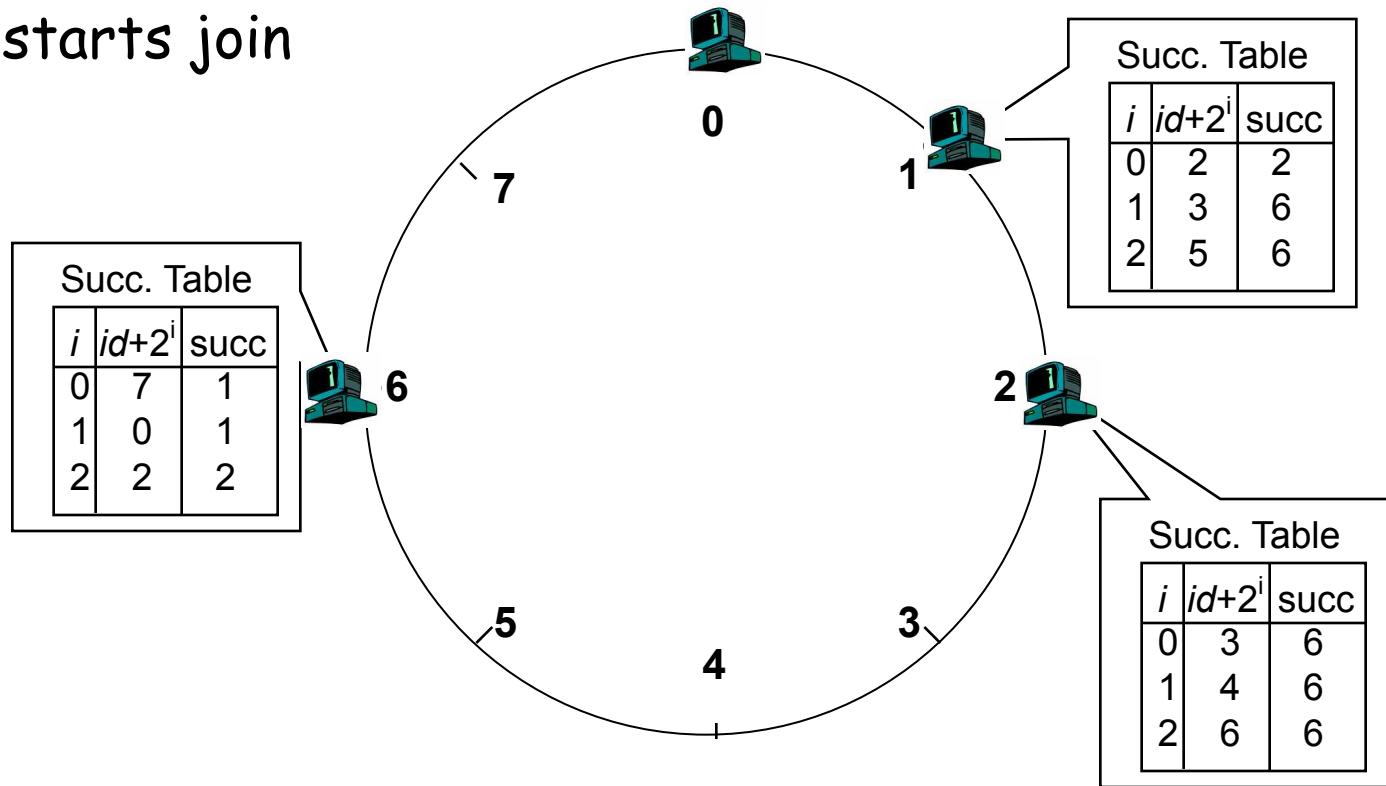
DHT: Chord Node Join

- Node n6 joins



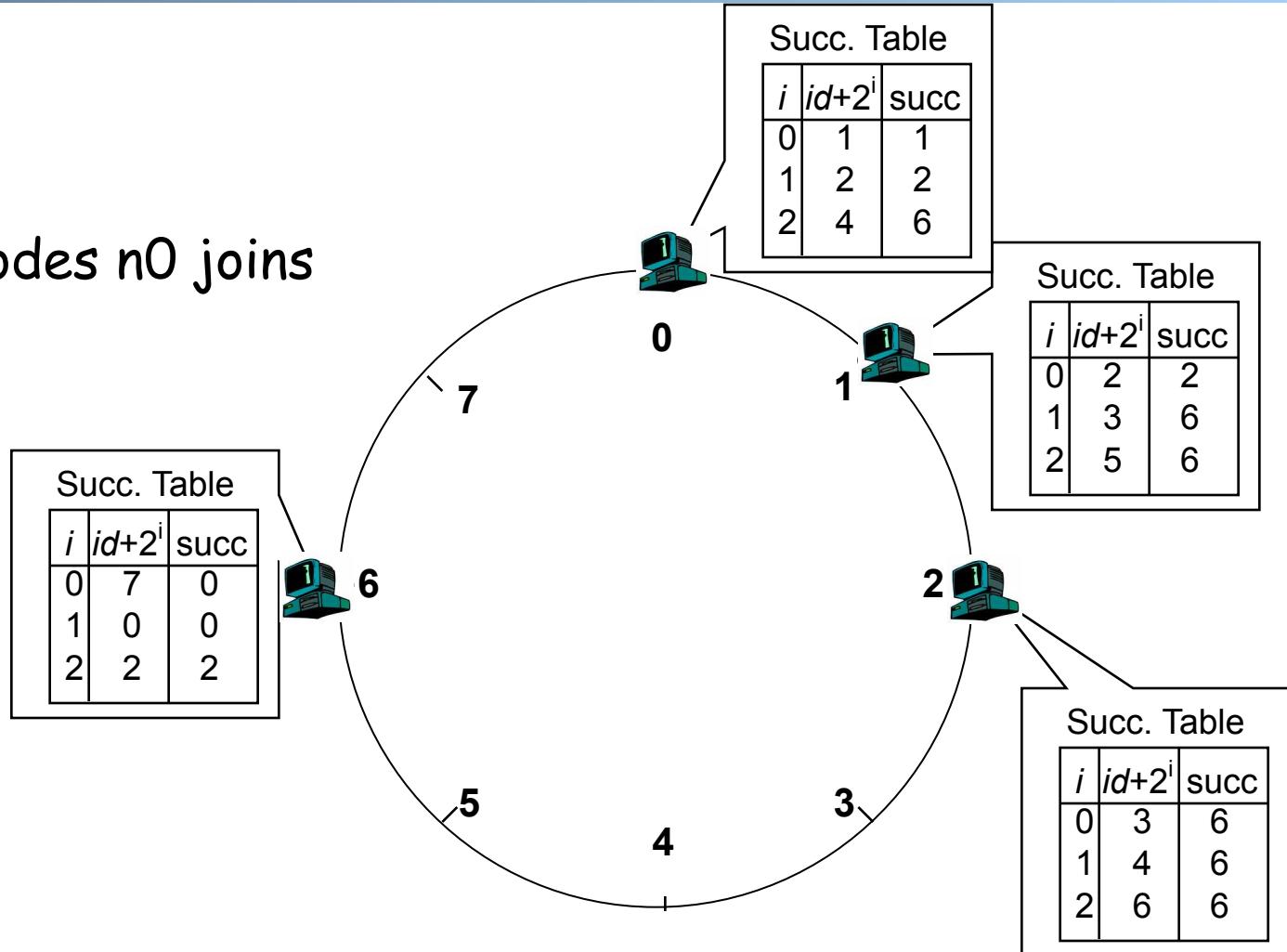
DHT: Chord Node Join

- Node n0 starts join



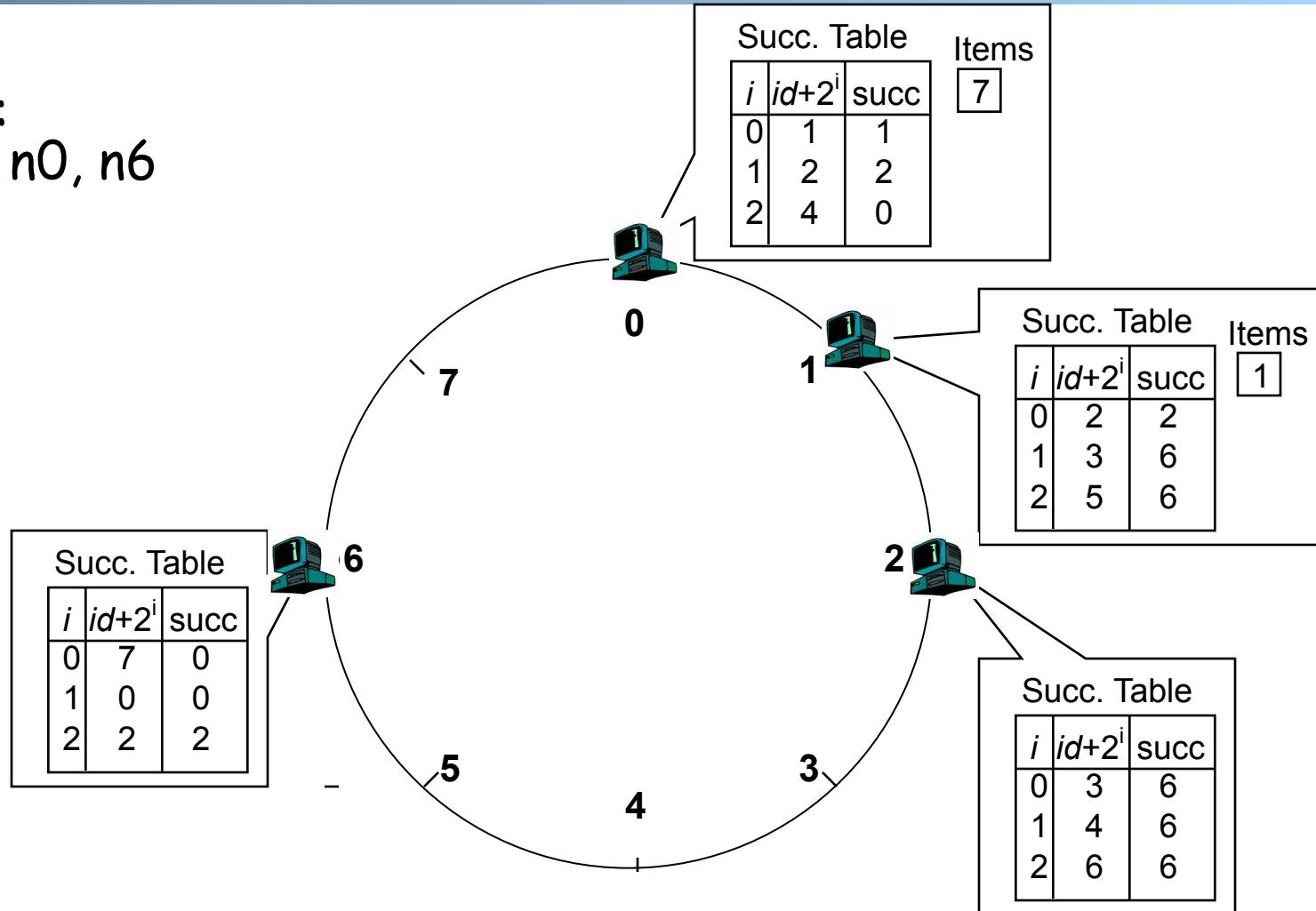
DHT: Chord Node Join

- After Nodes n0 joins



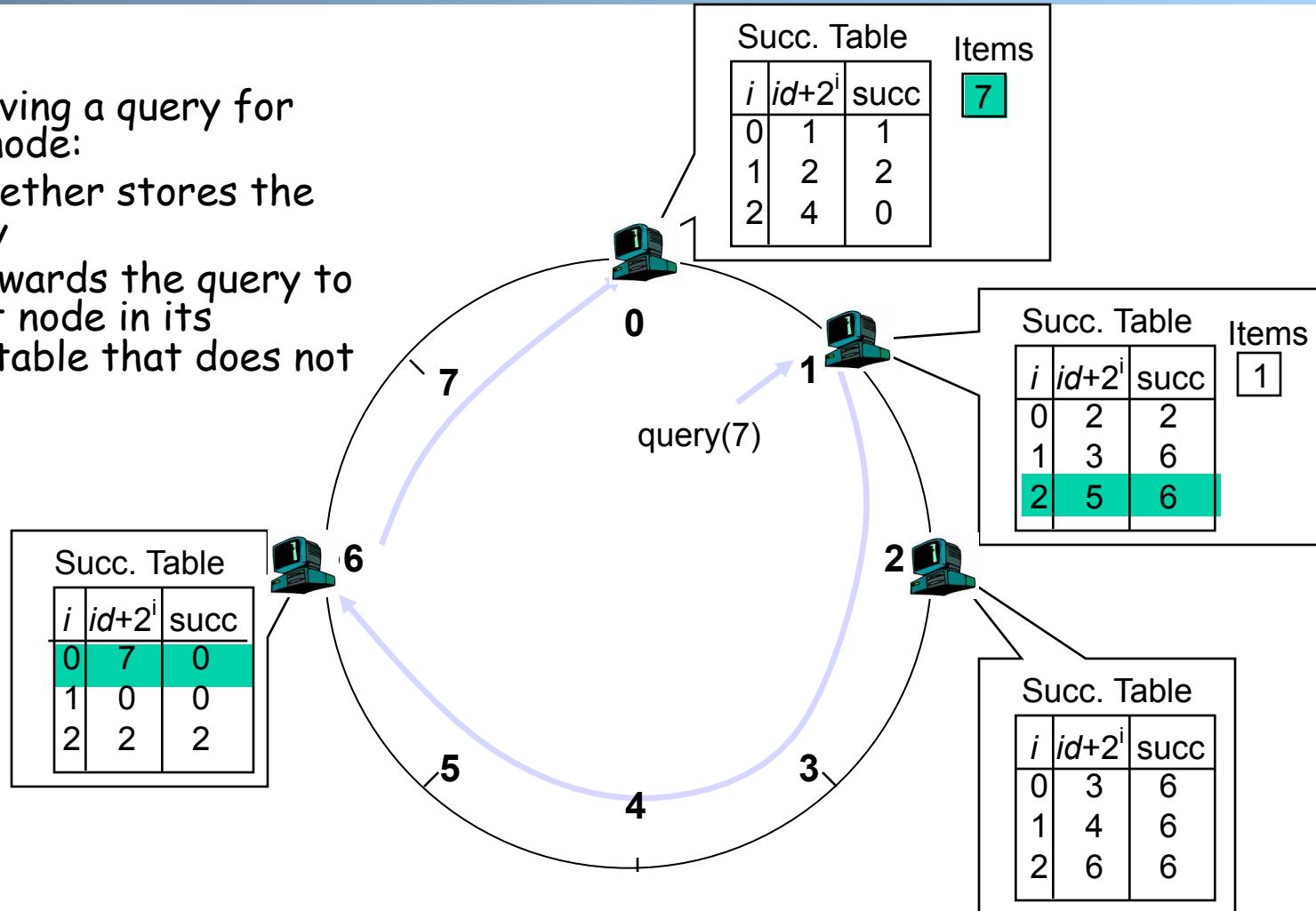
DHT: Chord Insert Items

- Nodes:
 n_1, n_2, n_0, n_6
- Items:
 f_7, f_1



DHT: Chord Routing

- Upon receiving a query for item id , a node:
 - checks whether stores the item locally
 - if not, forwards the query to the largest node in its successor table that does not exceed id



Chord/CAN Summary

- Each node “owns” some portion of the key-space
 - in CAN, it is a multi-dimensional “zone”
 - in Chord, it is the key-id-space between two nodes in 1-D ring
- Files and nodes are assigned random locations in key-space
 - provides some load balancing
 - probabilistically equal division of keys to nodes
- Routing/search is local (distributed) and greedy
 - node X does not know of a path to a key Z
 - but if it appears that node Y is the closest to Z among all of the nodes known to X
 - so route to Y