Network Applications: High-performance Server Design

Qiao Xiang, Congming Gao

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

10/24/2023

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

Outline

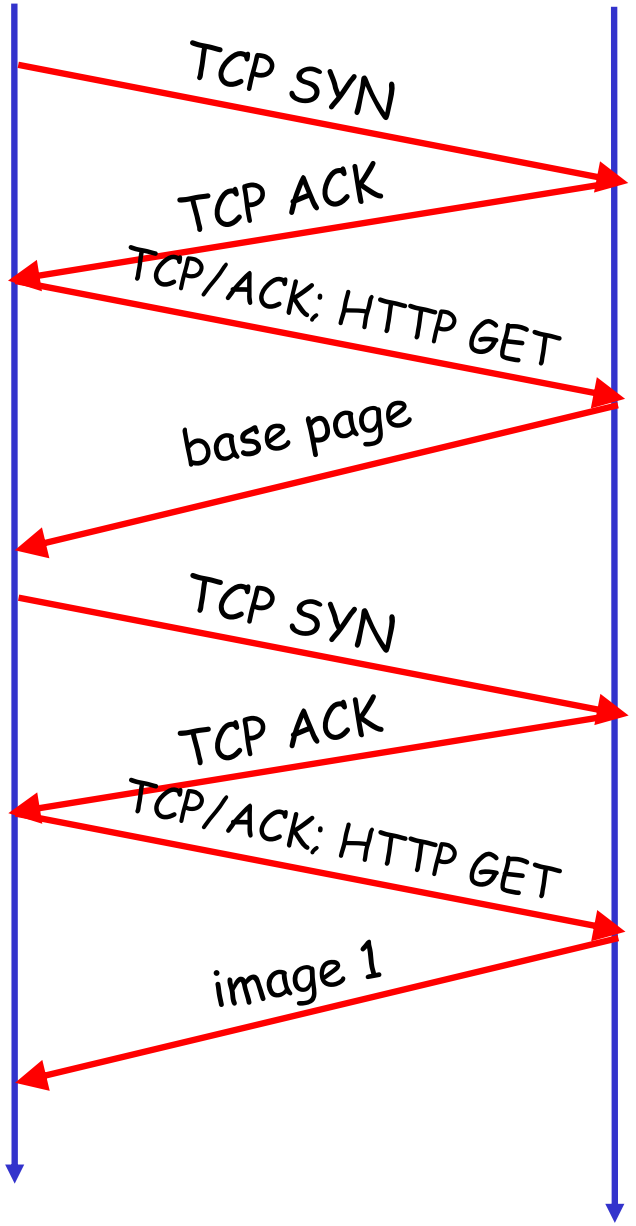
- Admin and recap
- High-performance network server design
  - Overview
  - Threaded servers
    - Per-request thread
      - problem: large # of threads and their creations/deletions may let overhead grow out of control
    - Thread pool
      - Design 1: Service threads compete on the welcome socket
      - Design 2: Service threads and the main thread coordinate on the shared queue
        - » polling (busy wait)
        - » suspension: wait/notify

Admin

- Exam 1 date?

Recap: Latency of Basic HTTP/1.0

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



Recap: Substantial Efforts to Speedup HTTP/1.0

- Reduce the number of objects fetched [Browser cache]

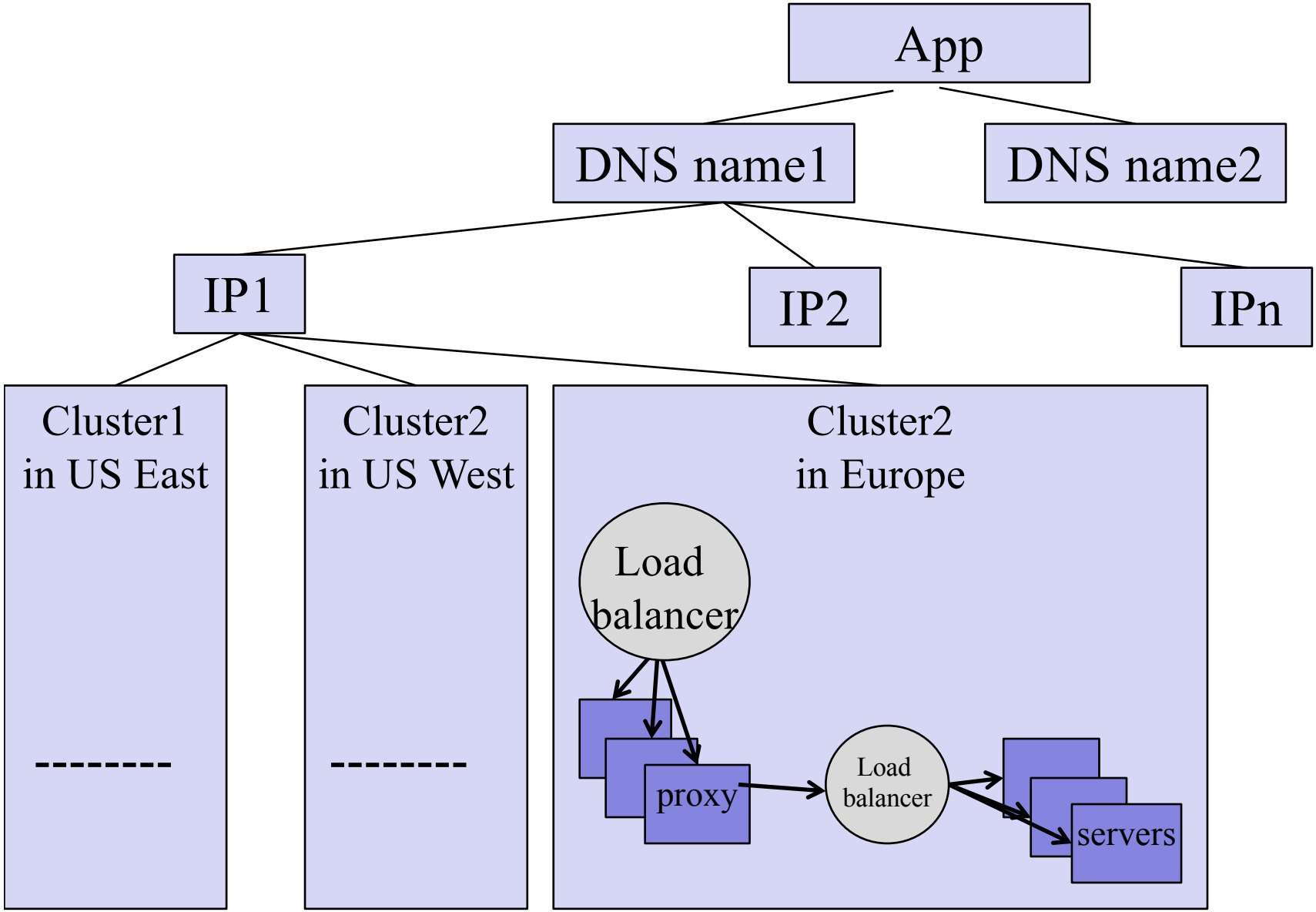
- Reduce data volume [Compression of data]
- Header compression [HTTP/2]

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

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

- Server push [HTTP/2]

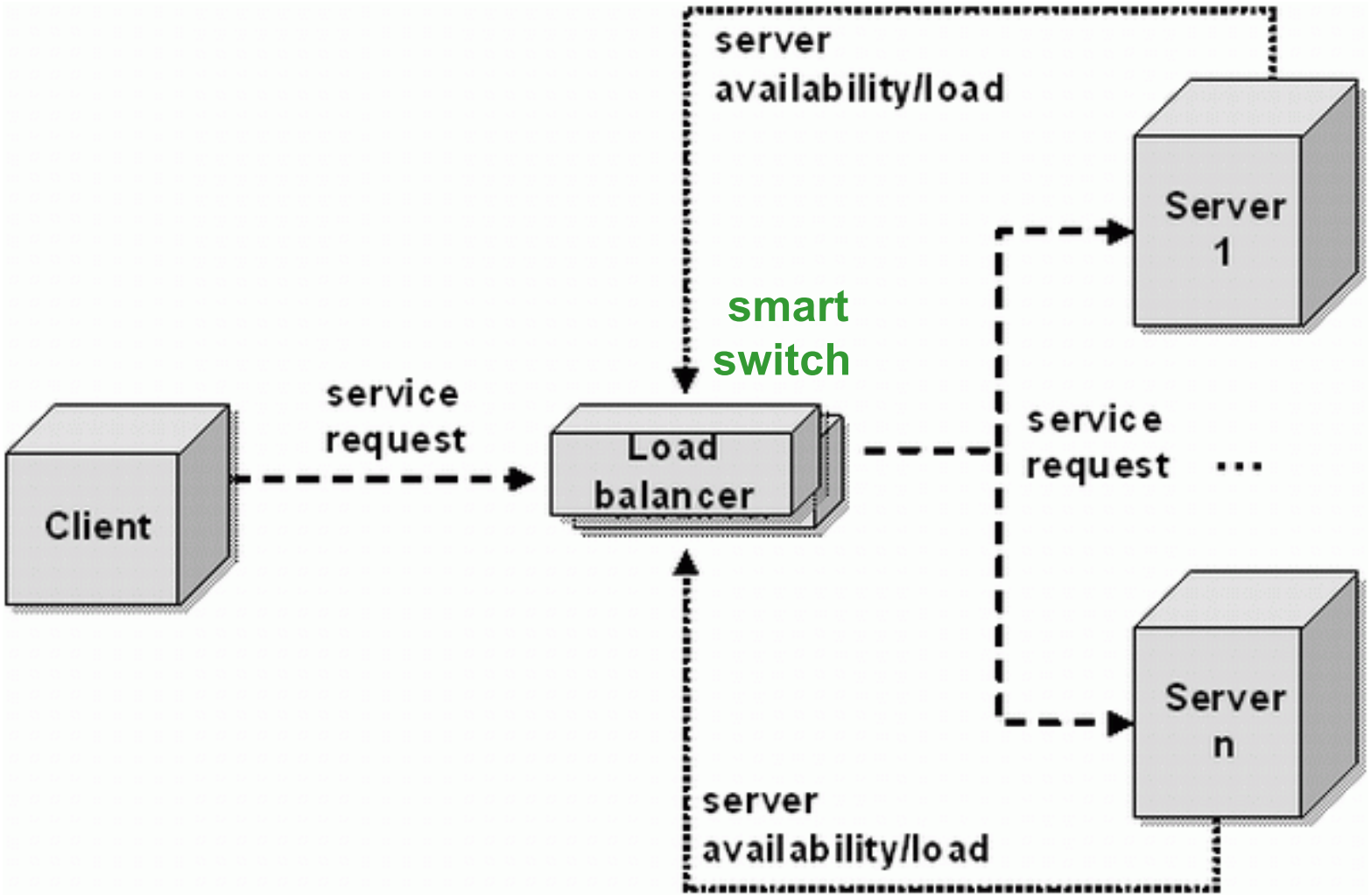
Recap: Direction Mechanisms



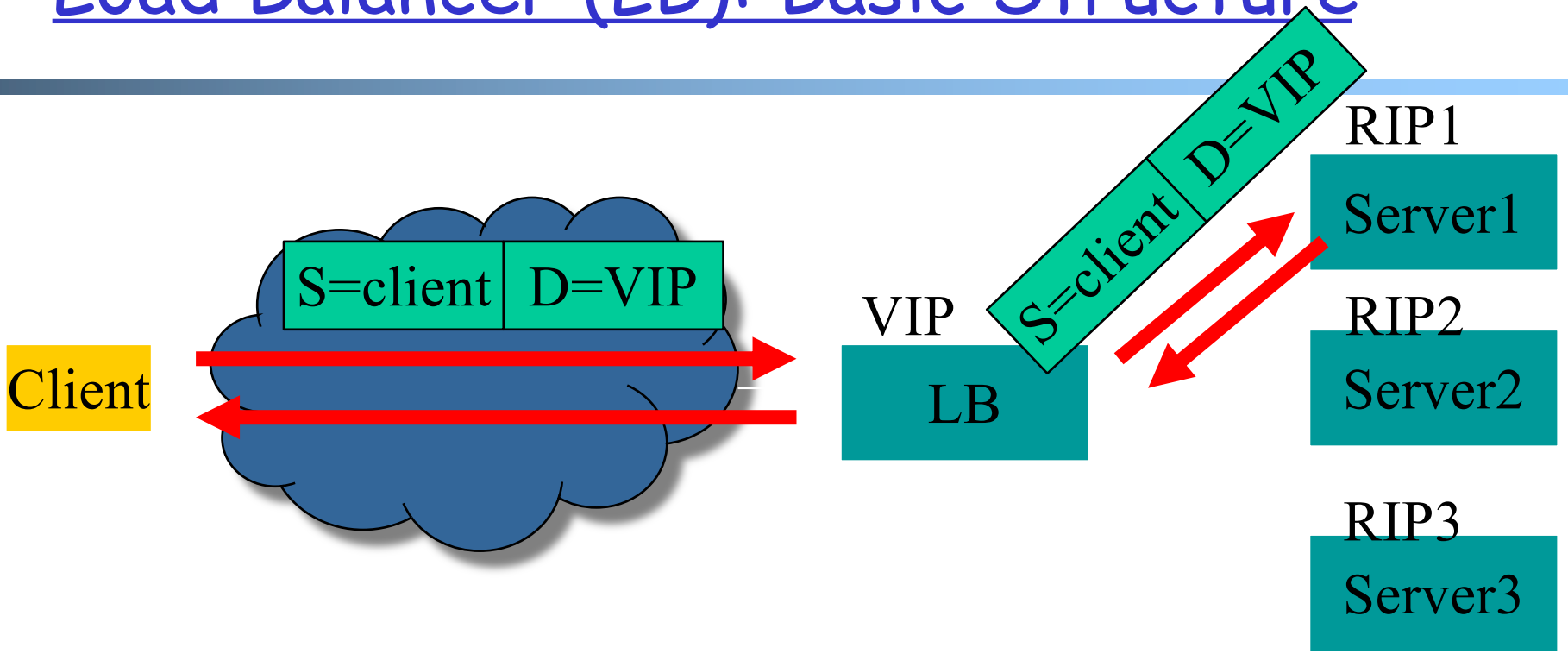
Outline

- Recap
- Single, high-performance network server
- Multiple network servers
  - Basic issues
  - Load direction
    - DNS (IP level)
    - Load balancer/smart switch (sub-IP level)

Smart Switch: Big Picture



Load Balancer (LB): Basic Structure

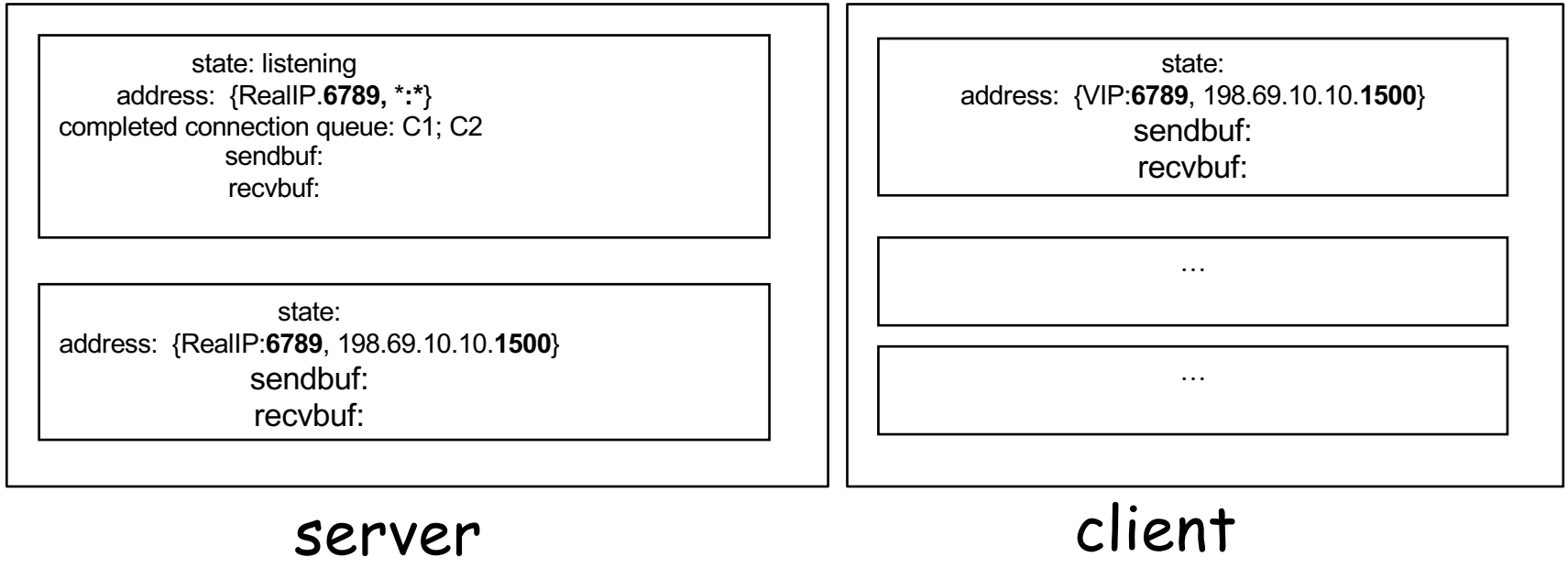


Problem of the basic structure?

Problem

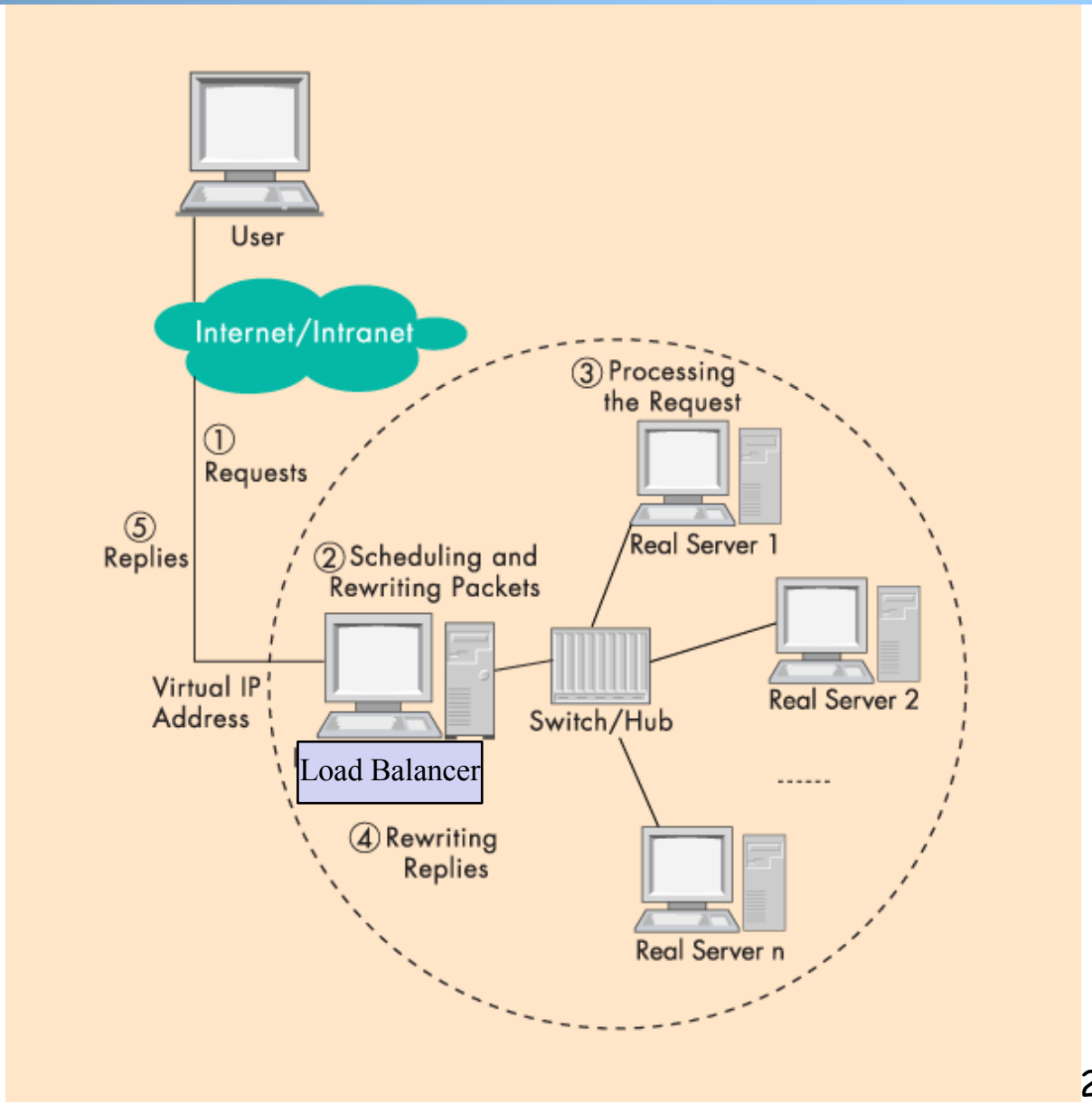
S=client	D=VIP

- Client to server packet has VIP as destination address, but real servers use RIPs
  - 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



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

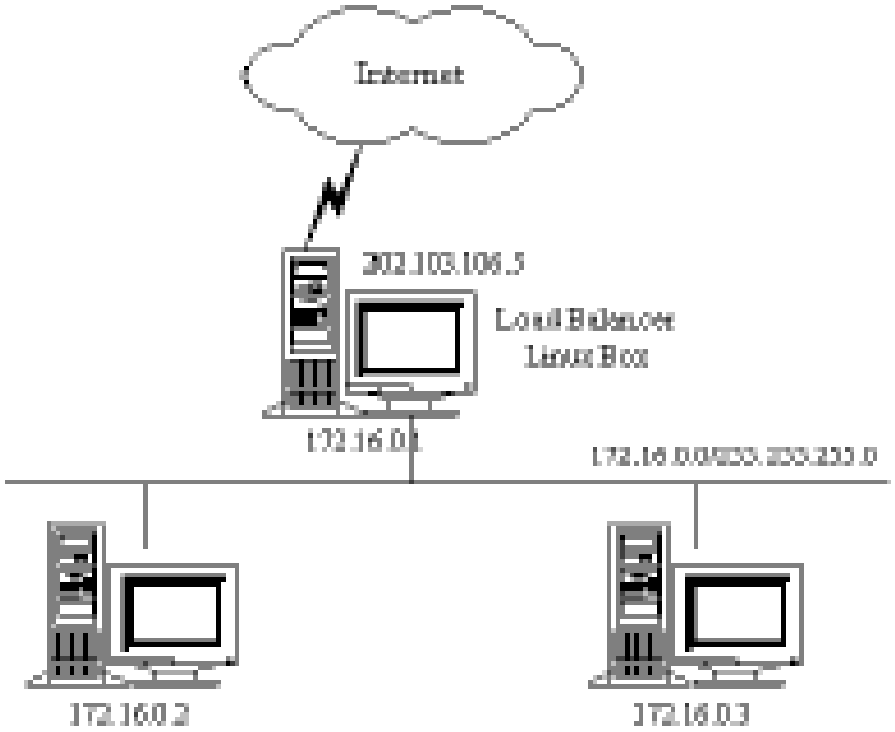
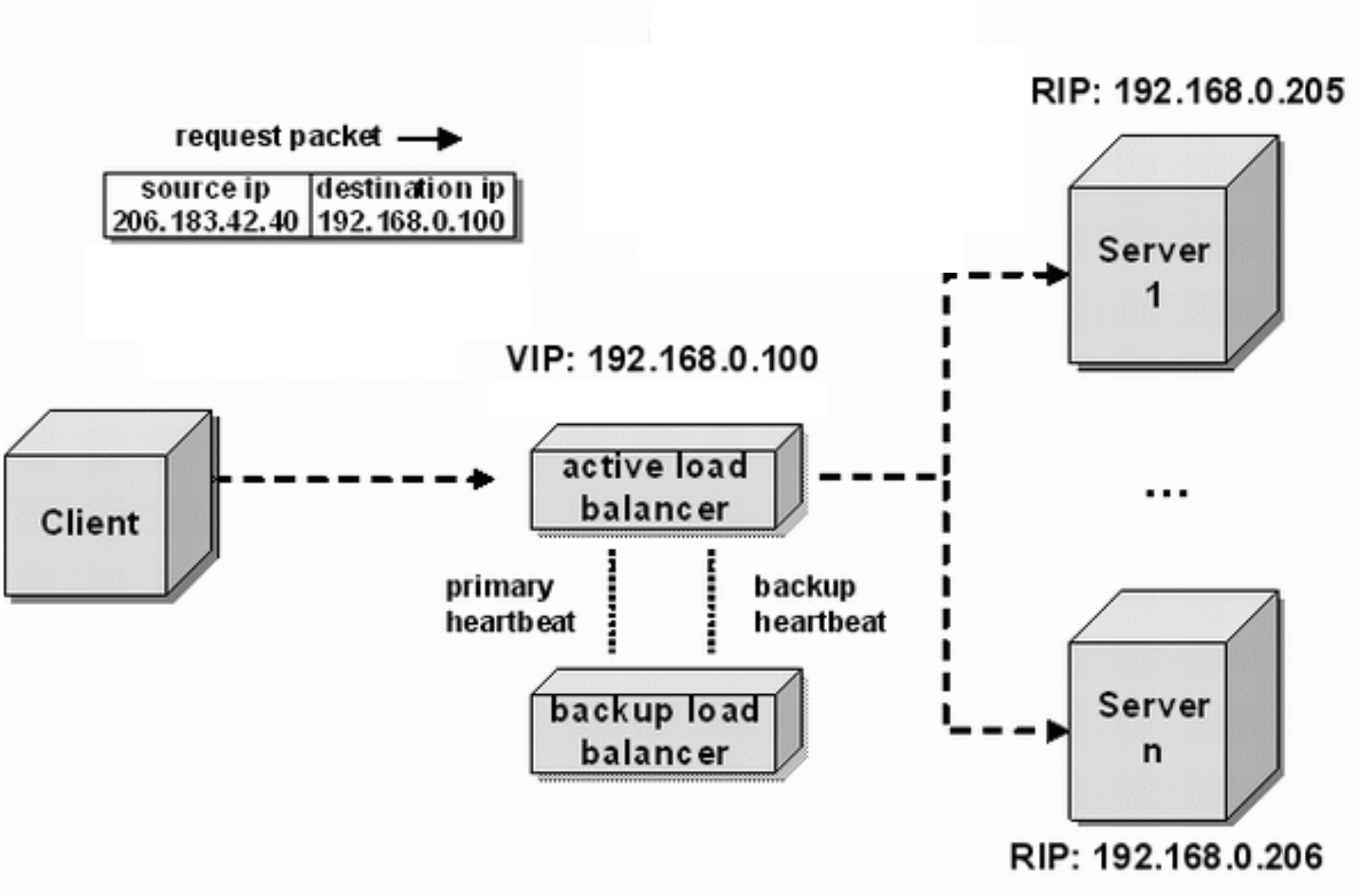


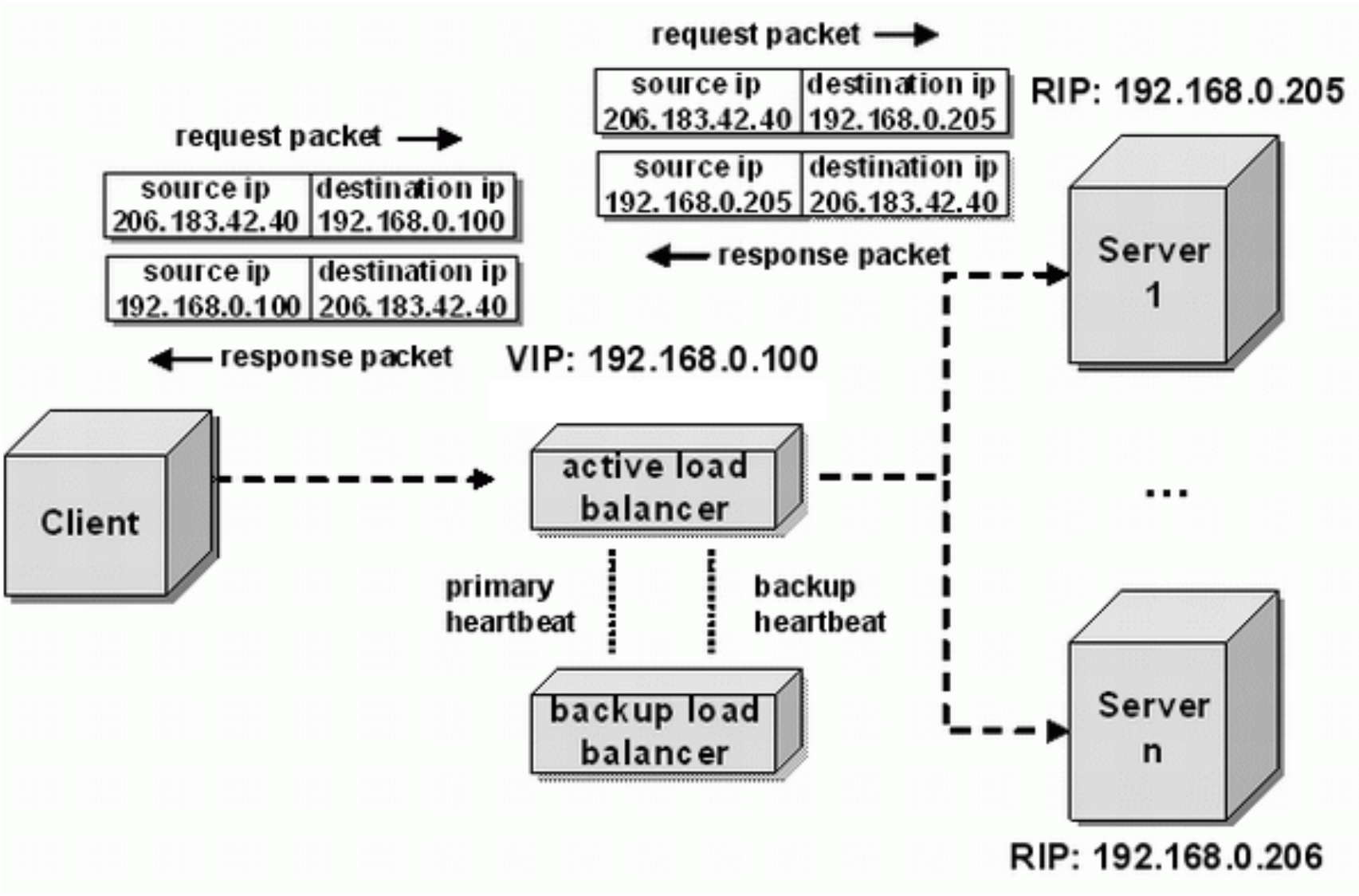
Table 1: an example of virtual server rules

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


RIP: 192.168.0.205
request packet
source ip
206.183.42.40
destination ip
192.168.0.100
Server
1
VIP: 192.168.0.100
Client
active load
balancer
...
primary
heartbeat
backup
heartbeat
backup load
balancer
Server
n
RIP: 192.168.0.206

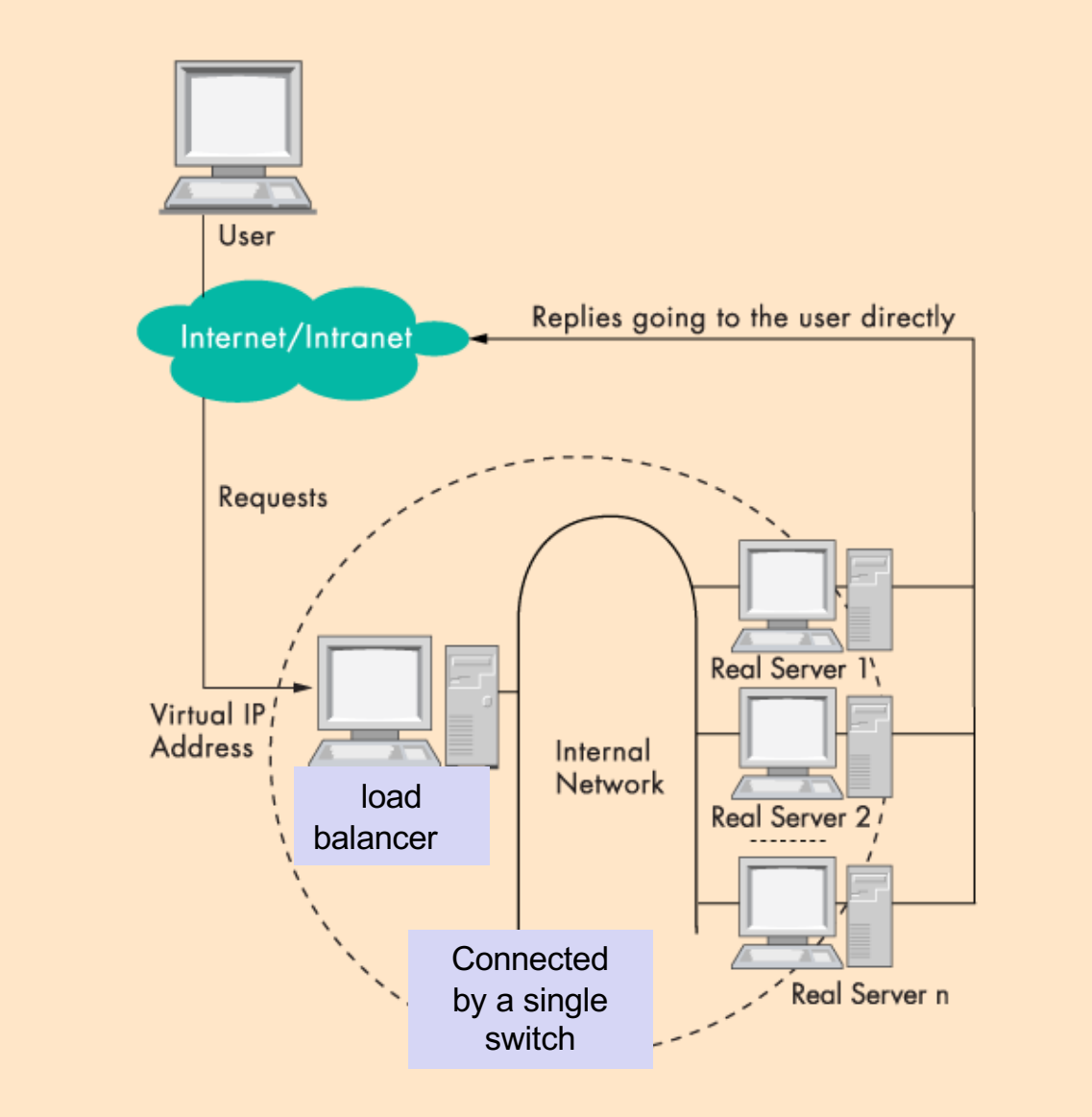
LB/NAT Flow


request packet
source ip
206.183.42.40
destination ip
192.168.0.205
RIP: 192.168.0.205
request packet
source ip
206.183.42.40
destination ip
192.168.0.100
source ip
192.168.0.205
destination ip
206.183.42.40
response packet
Server
1
source ip
192.168.0.100
destination ip
206.183.42.40
response packet
VIP: 192.168.0.100
Client
active load
balancer
...
primary
heartbeat
backup
heartbeat
backup load
balancer
Server
n
RIP: 192.168.0.206

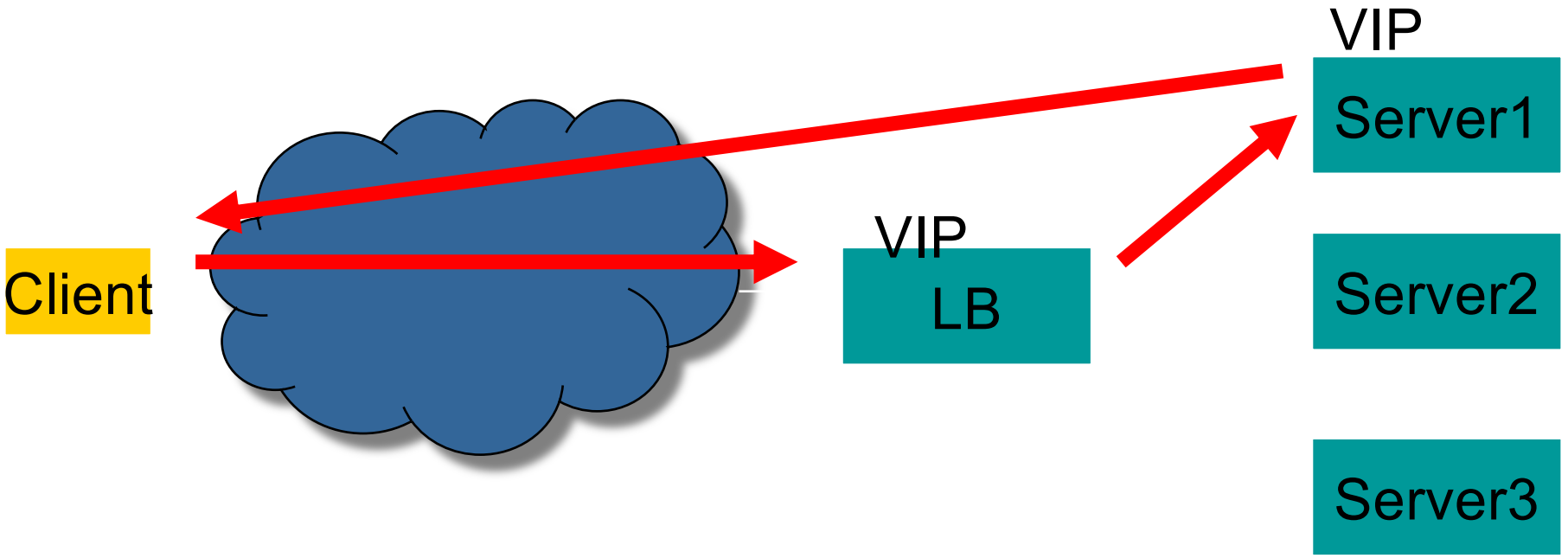
LB/NAT Advantages and Disadvantages

- 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

Goal: LB w/ Direct Reply



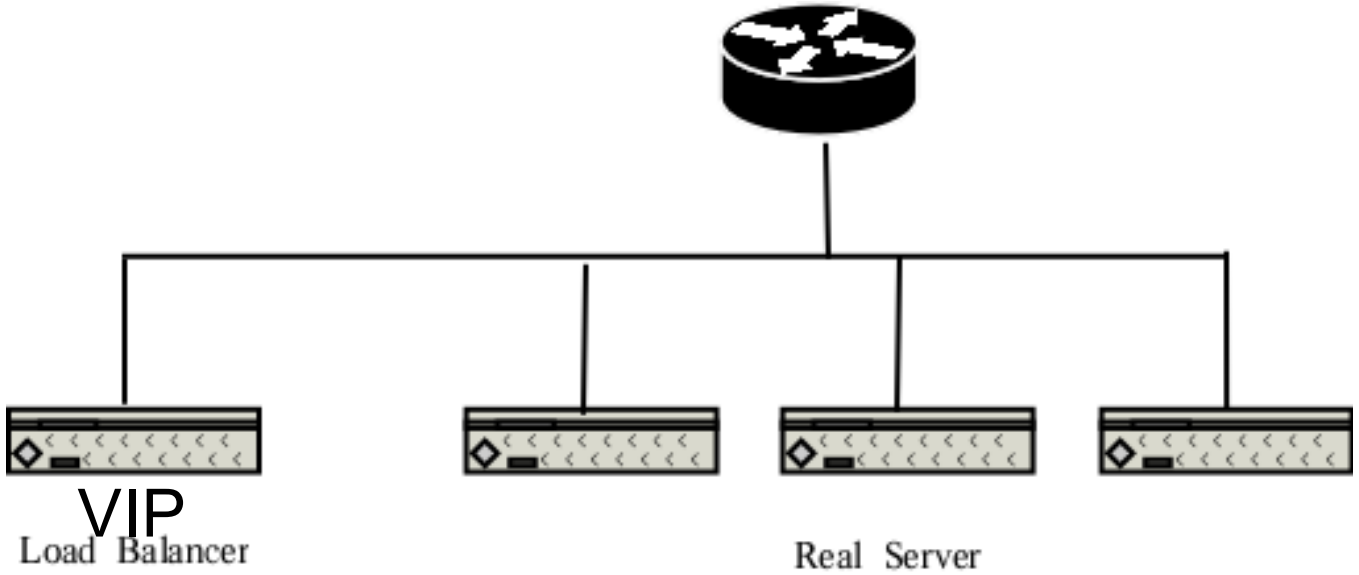
LB with Direct Reply: Implication



Direct reply → Each real server uses VIP as its IP address

→ Address conflict: multiple devices w/ the same IP addr

Why IP Address Matters?



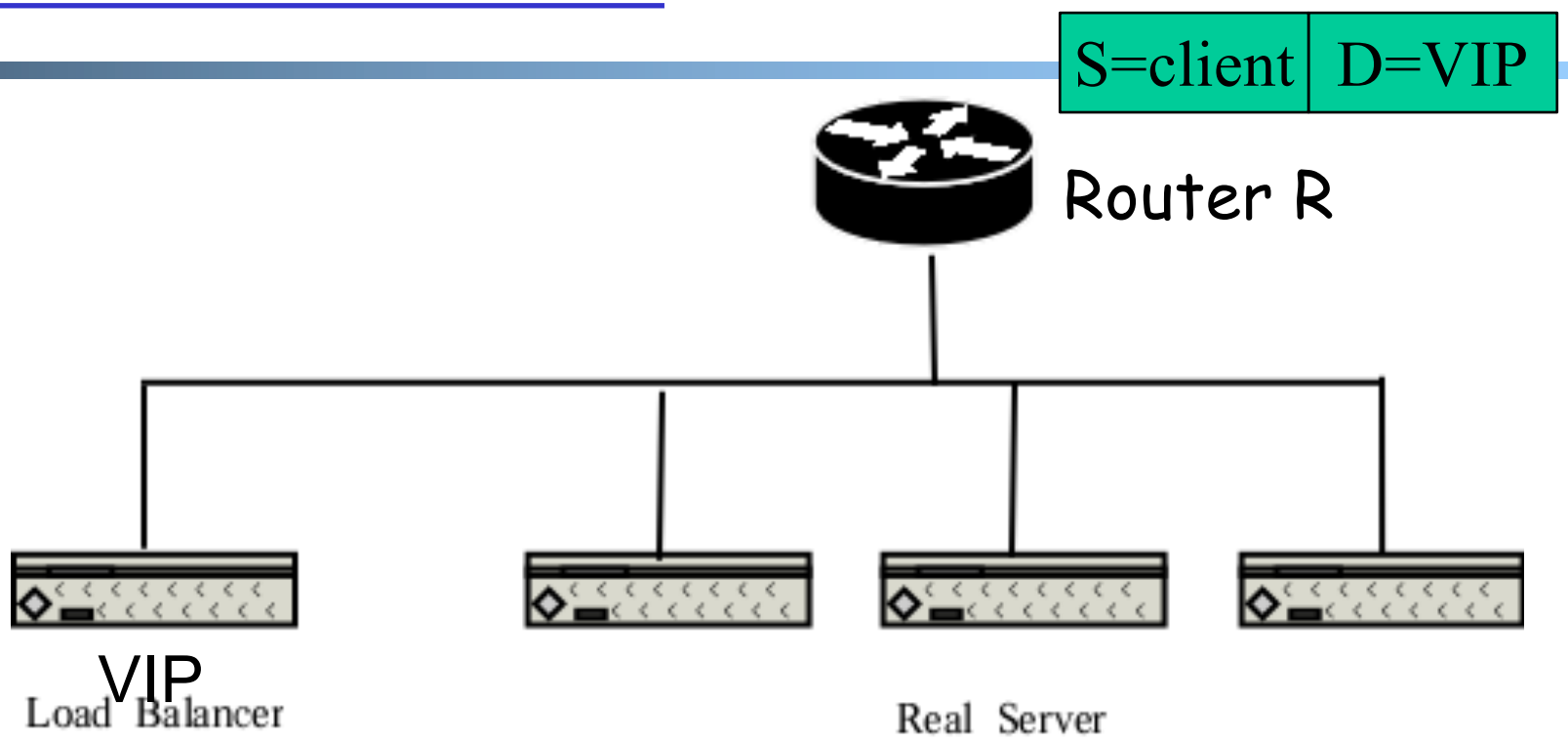
- Each network interface card listens to an assigned MAC address
- A router is configured with the range of IP addresses connected to each interface (NIC)
- To send to a device with a given IP, the router needs to translate IP to MAC (device) address
- The translation is done by the Address Resolution Protocol (ARP)

ARP Protocol

- ARP is "plug-and-play":
  - nodes create their ARP tables without intervention from net administrator

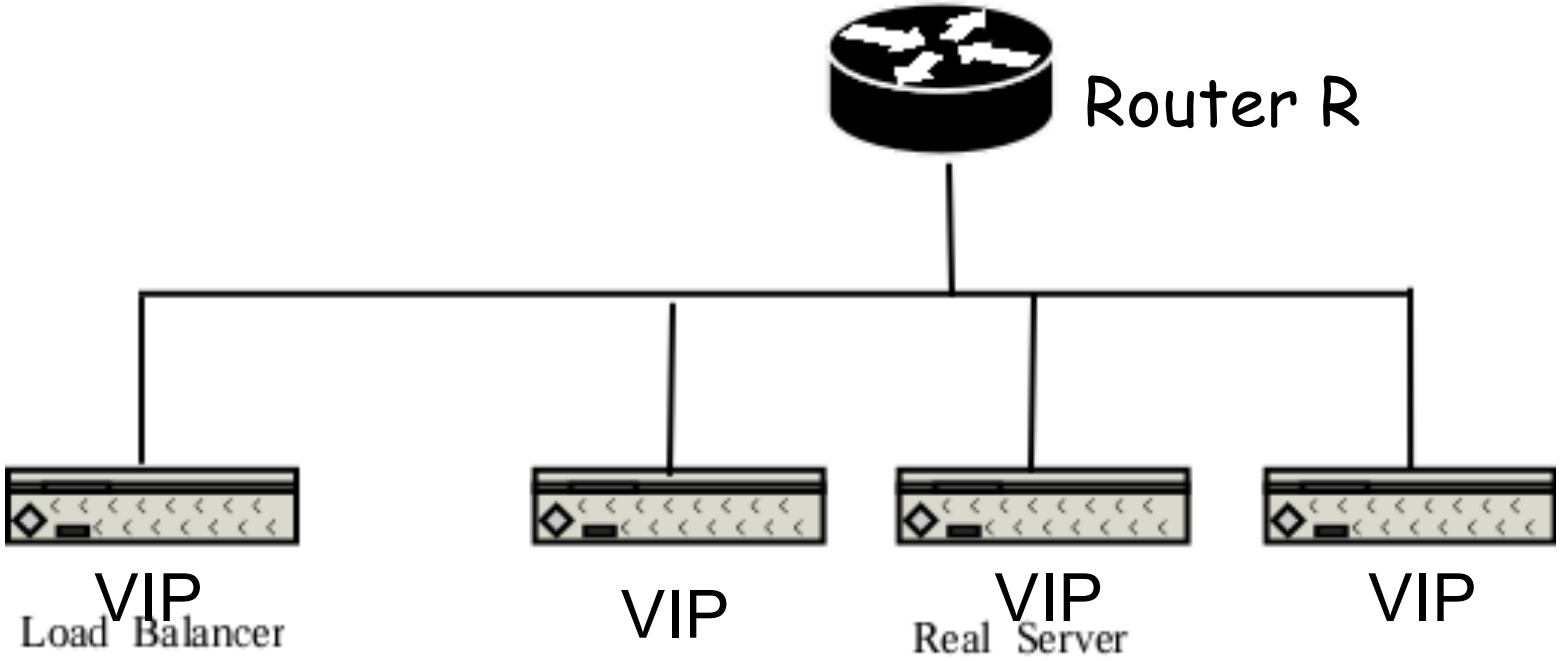
- A broadcast protocol:
  - Router broadcasts query frame, containing queried IP address
    - all machines on LAN receive ARP query
  - Node with queried IP receives ARP frame, replies its MAC address

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

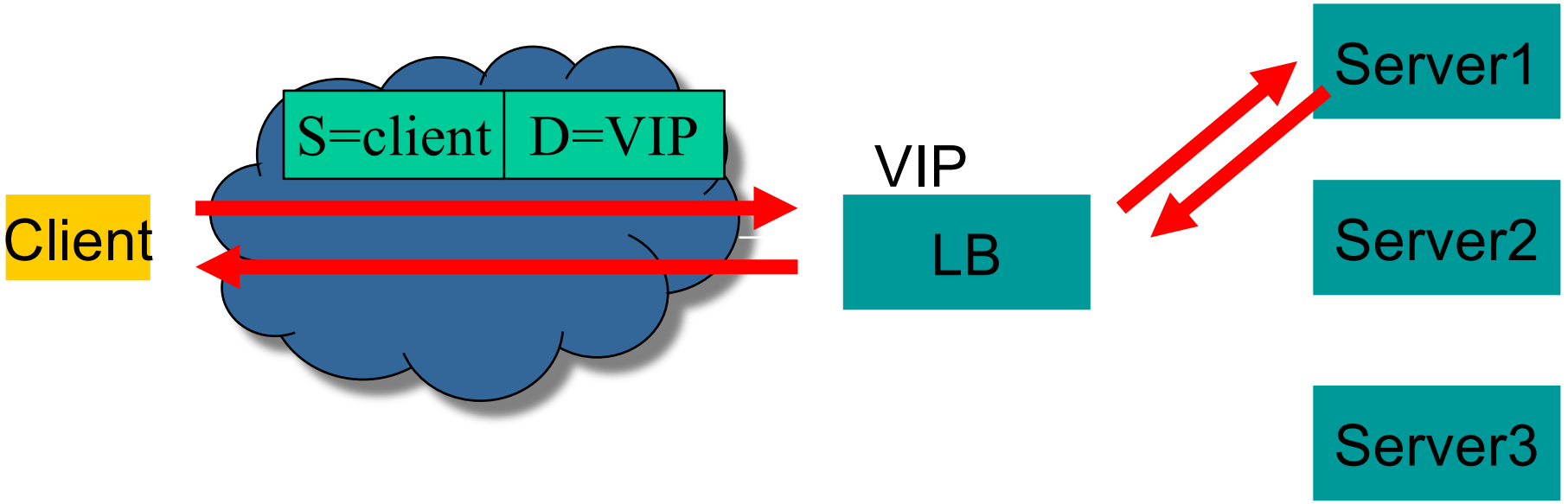
LB/DR Advantages and Disadvantages

- 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
  - Considered by some as a hack, not a clean architecture

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

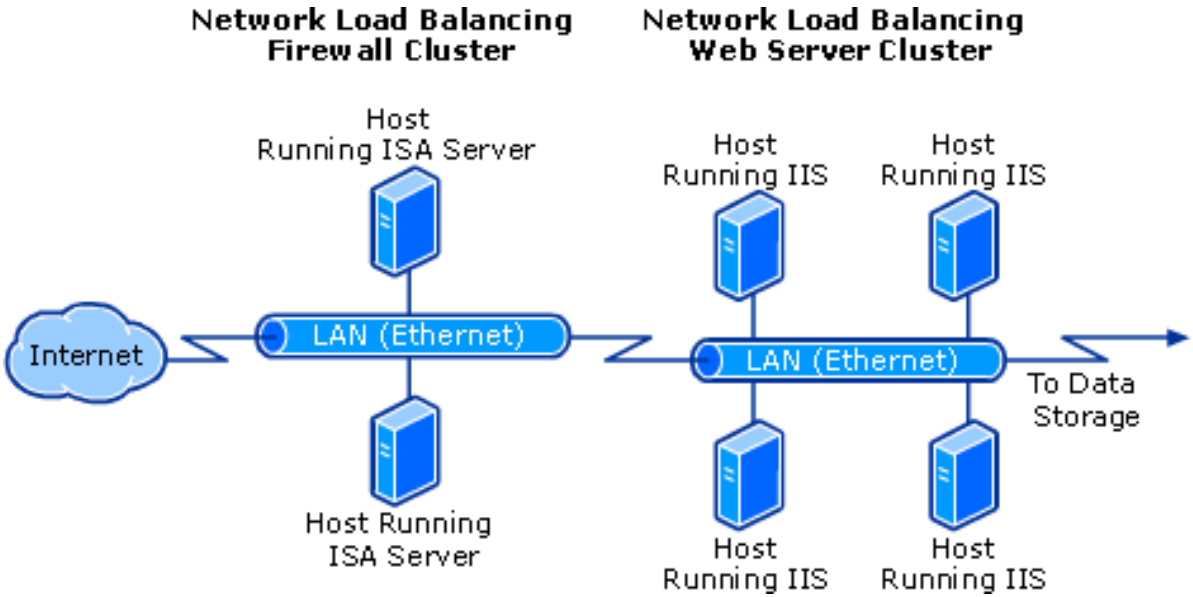
Problem of the Load Balancer Architecture



One major problem is that the LB becomes a single point of failure (SPOF).

Solutions

- Redundant load balancers
  - E.g., two load balancers (a good question to think offline)
- Fully distributed load balancing
  - e.g., Microsoft Network Load Balancing (NLB)



Microsoft NLB

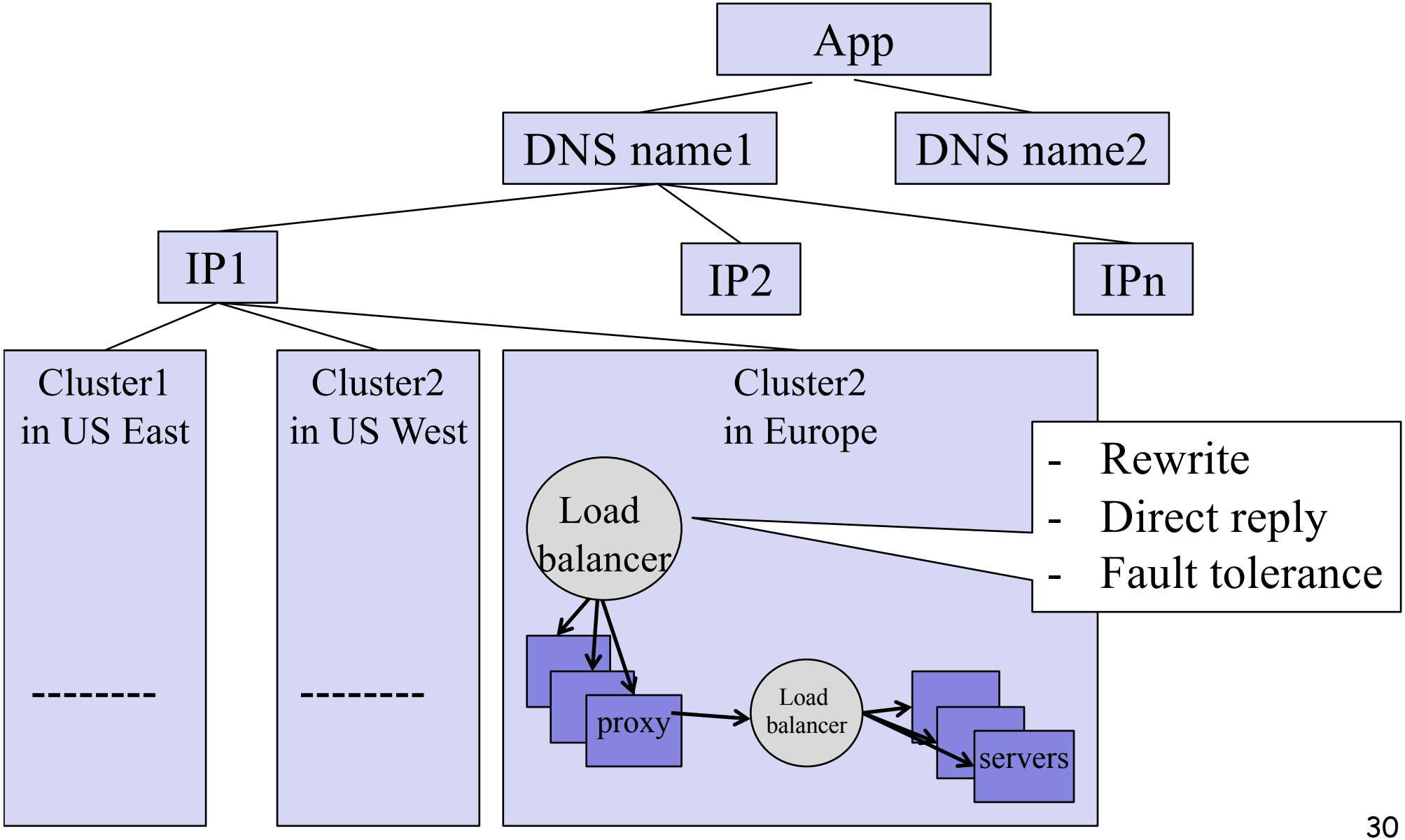
- No dedicated load balancer
- All servers in the cluster receive all packets
- Key issue: one and only one server processes each packet
  - All servers within the cluster simultaneously run a mapping algorithm to determine which server should handle the packet. Those servers not required to service 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

http://technet.microsoft.com/en-us/library/cc739506%28WS.10%29.aspx

Discussion

- Compare the design of using Load Balancer vs Microsoft NLB

Recap: Direction Mechanisms



Outline

- Admin and recap
- Single, high-performance network server
- Multiple servers
  - Overview
  - Basic mechanisms
  - Example: YouTube (offline read)

http://video.google.com/videoplay?docid=-6304964351441328559#

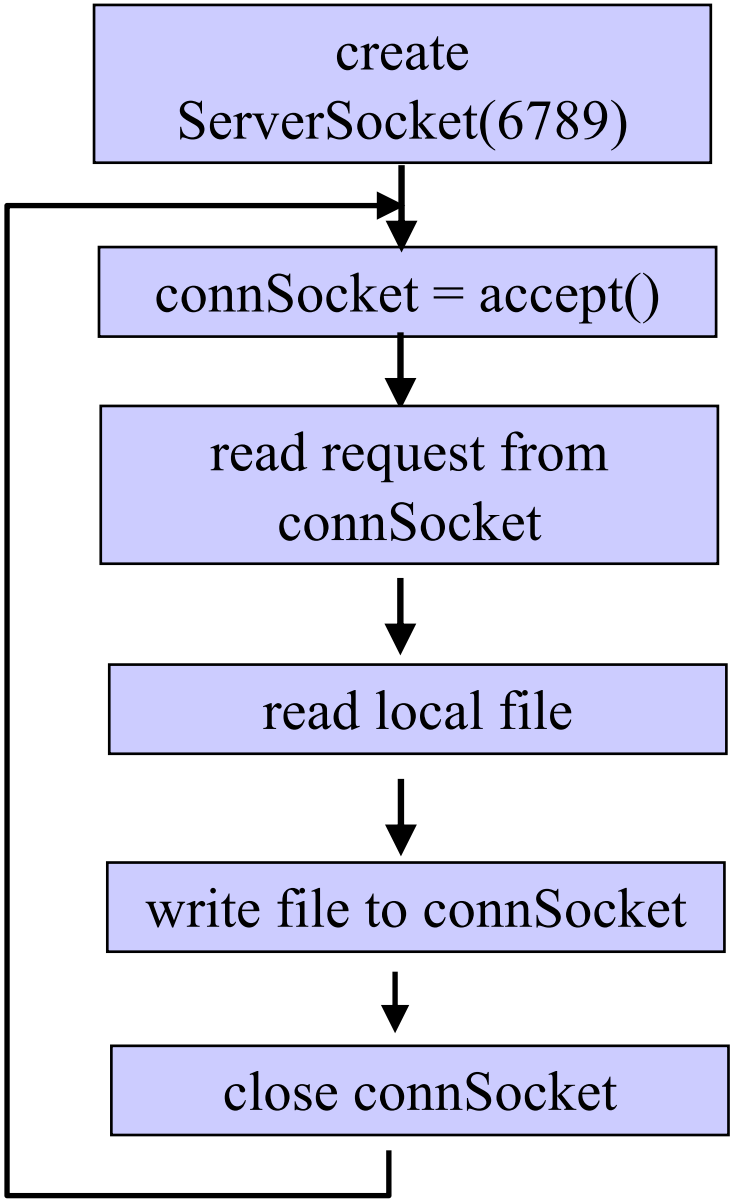
You Tube

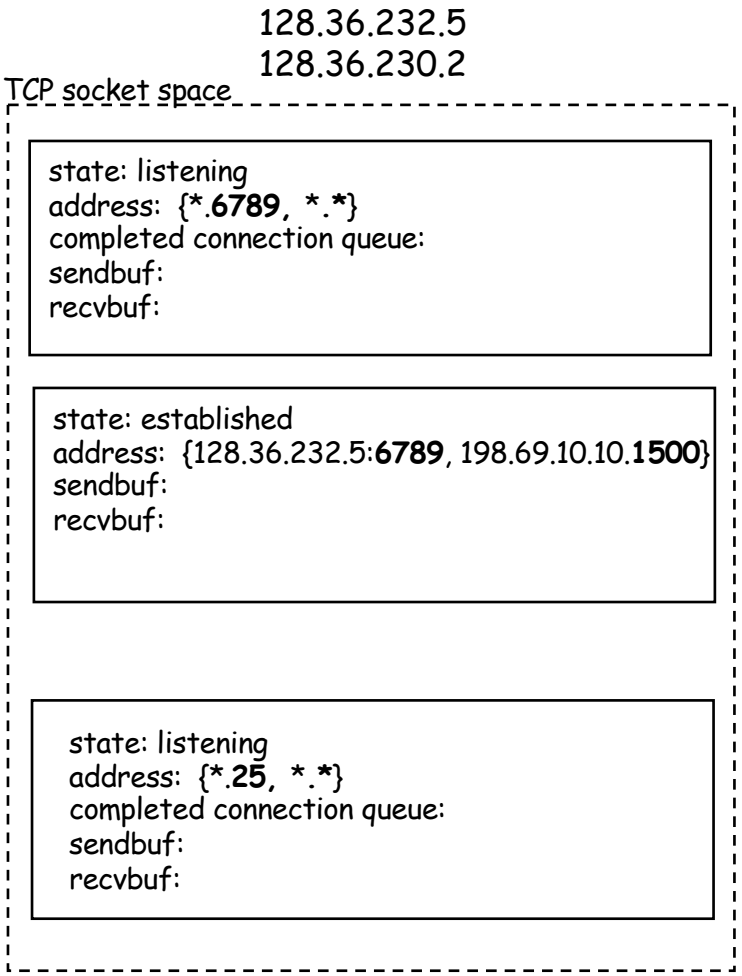
- 02/2005: Founded by Chad Hurley, Steve Chen and Jawed Karim, who were all early employees of PayPal.
- 10/2005: First round of funding ($11.5 M)
- 03/2006: 30 M video views/day
- 07/2006: 100 M video views/day
- 11/2006: acquired by Google
- 10/2009: Chad Hurley announced in a blog that YouTube serving 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

WebServer Implementation



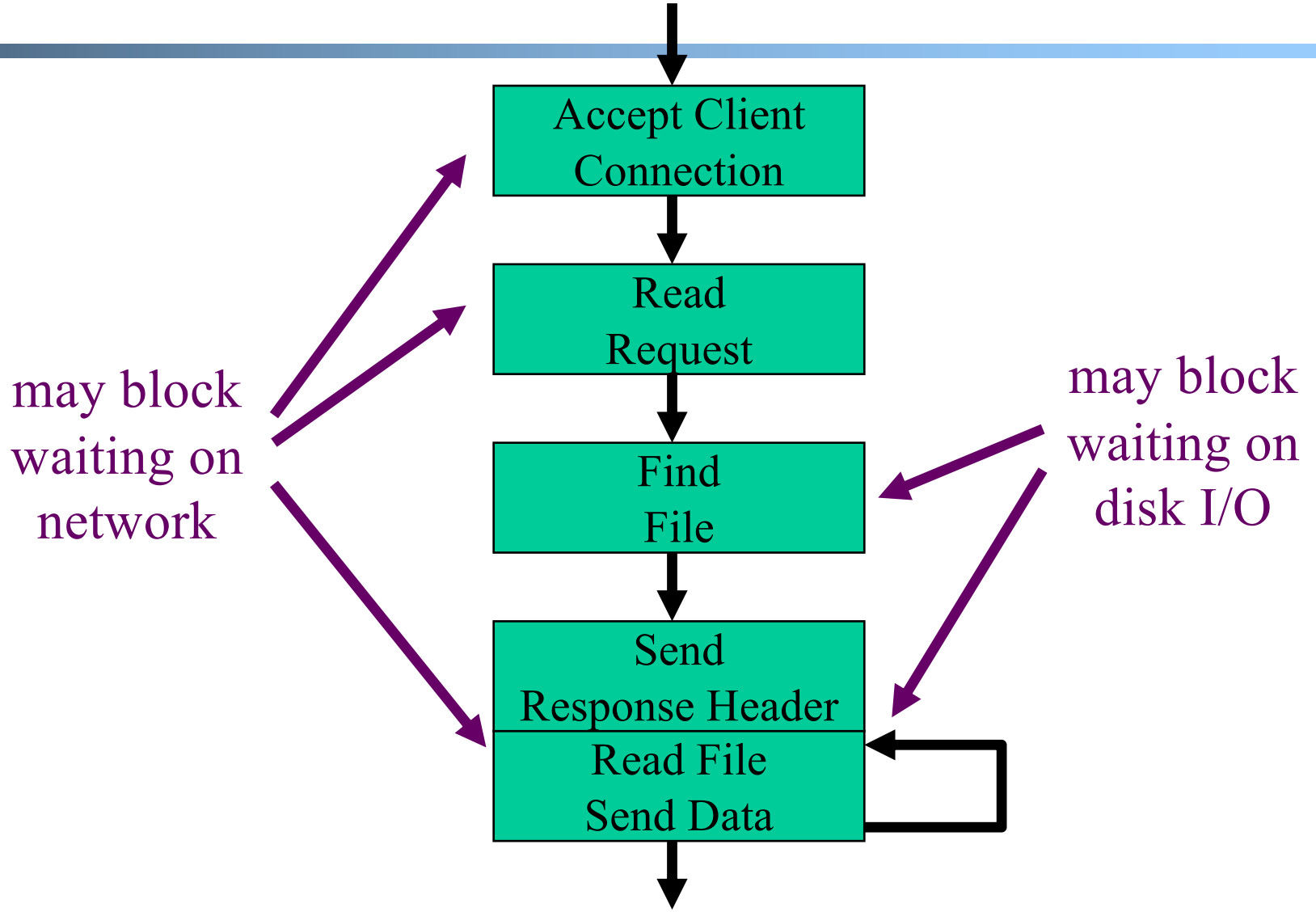


Discussion: what does each step do and how long does it take?

Demo

- Try TCPServer
- Start two TCPClient
  - Client 1 starts early but stops
  - Client 2 starts later but inputs first

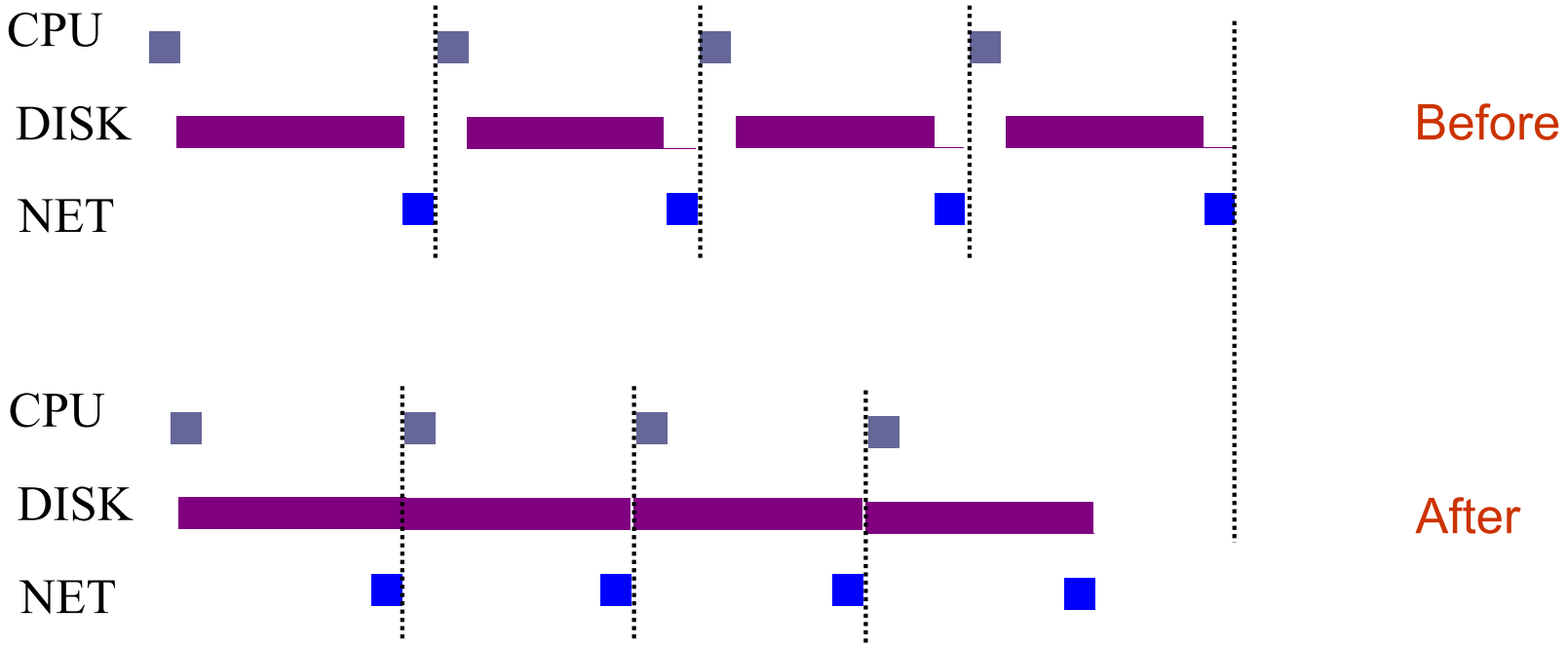
Server Processing Steps

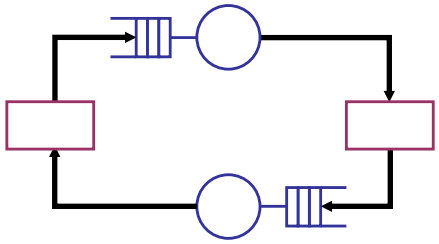


Writing High Performance Servers: Major Issues

- 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 Resource Bottleneck



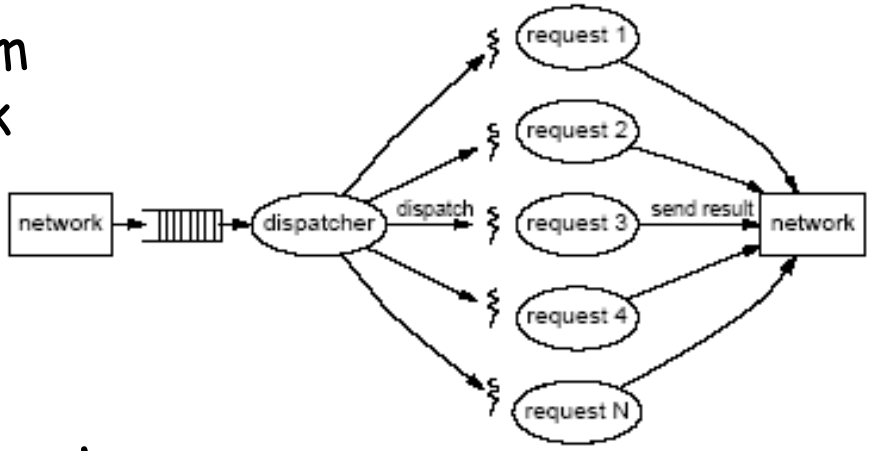


Outline

- Admin and recap
- Network server design
  - Overview
  - *Multi-thread network servers*

Multi-Threaded Servers

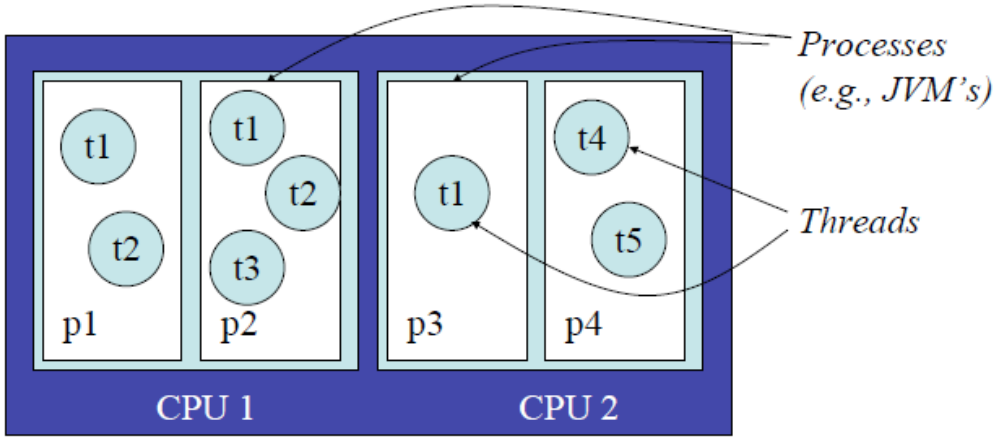
- Motivation:
  - Avoid blocking the whole program (so that we can reach bottleneck throughput)
- Idea: introduce threads
  - A thread is a sequence of instructions which may execute in parallel with other threads
  - When a blocking operation happens, only the flow (thread) performing the operation is blocked



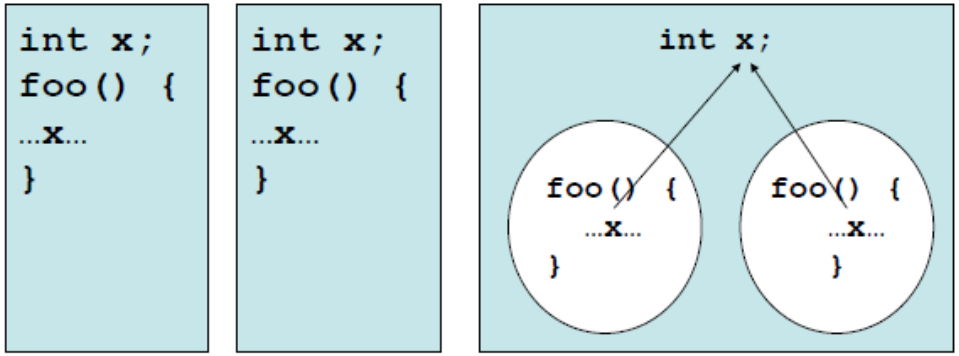
Background: Java Thread Model

- Every Java application has at least one thread
  - The "main" thread, started by the JVM to run the application's main() method
  - Most JVMs 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

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

http://java.sun.com/javase/6/docs/api/java/lang/Thread.html

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();
```

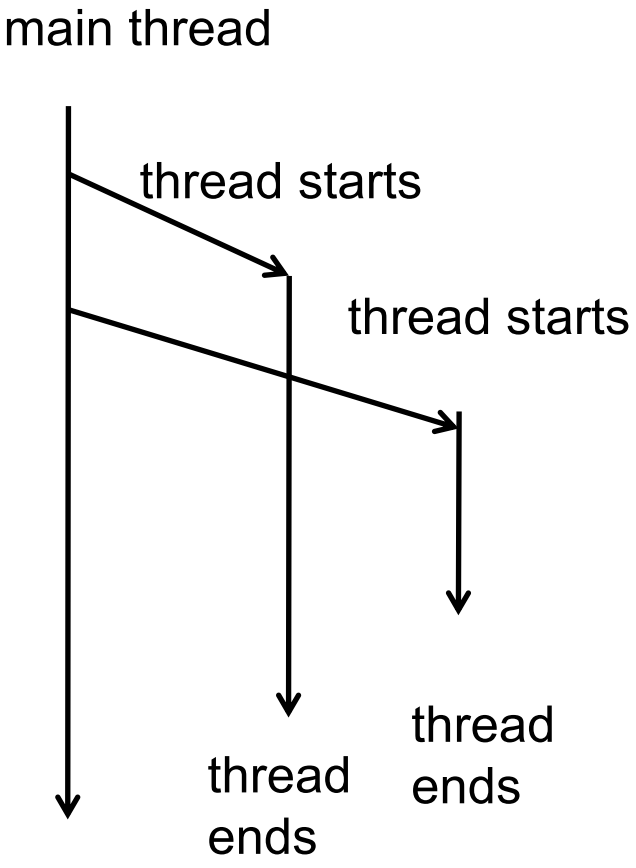
Example: a Multi-threaded TCPServer

- 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();
   }
```

```
class RequestHandler implements Runnable {
   RequestHandler(Socket connSocket) { … }
   public void run() {
     //
   } }
```



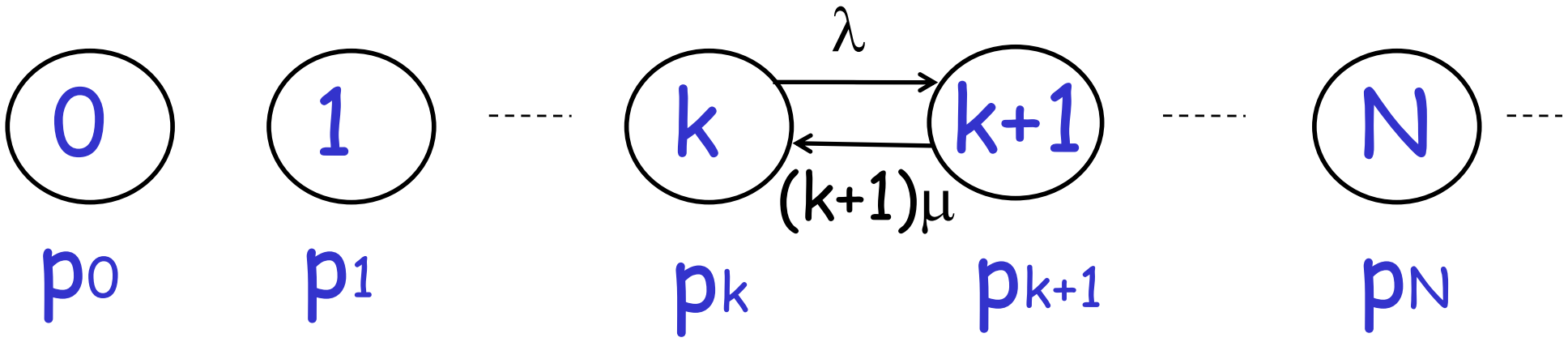
Try the per-request-thread TCP server: TCPServerMT.java

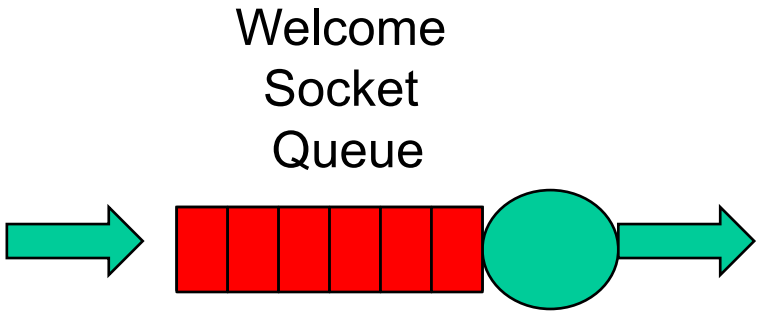
Summary: Implementing Threads

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



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

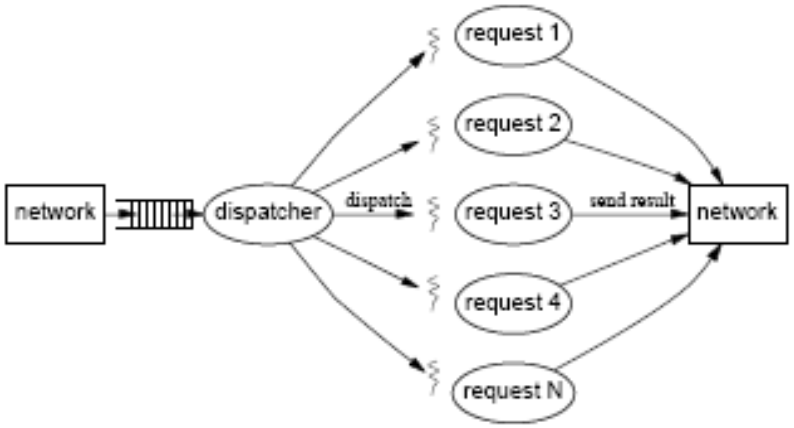
```
class RequestHandler
          implements Runnable {
   RequestHandler(Socket connSocket)
   {
      …
   }
   public void run() {
      // process request
   }
  …
}

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

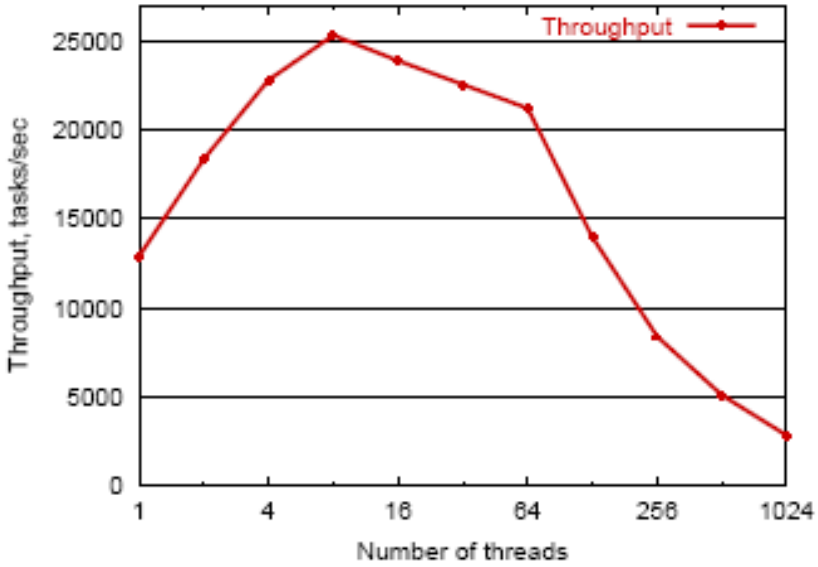
Modeling Per-Request Thread Server: Theory





Problem of Per-Request Thread: Reality

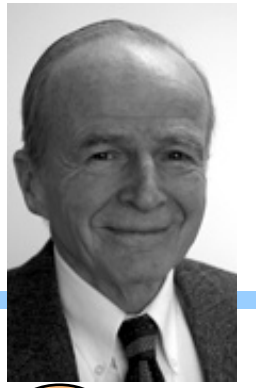




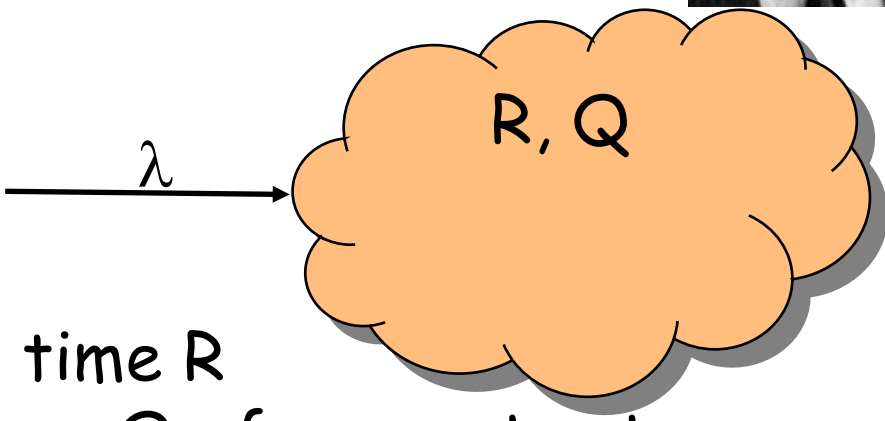
(937 MHz x86, Linux 2.2.14, each thread reading 8KB file)

- 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?

Recall: Little's Law (1961)



- For any system with no or (low) loss.
- Assume
  - mean arrival rate λ, mean time R at system, and mean number Q of requests at system

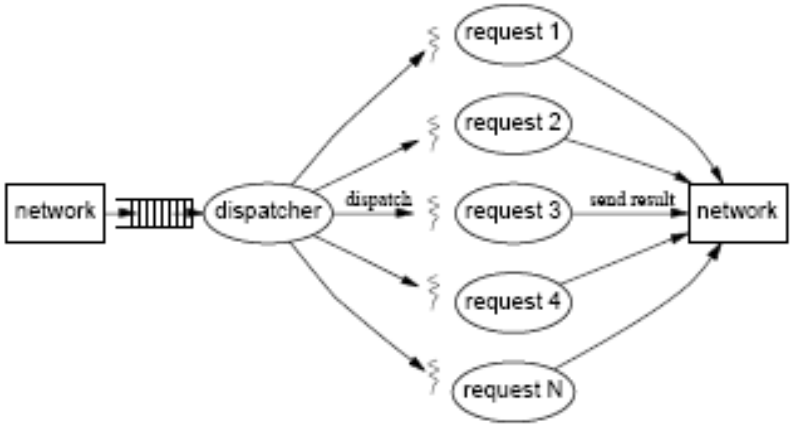


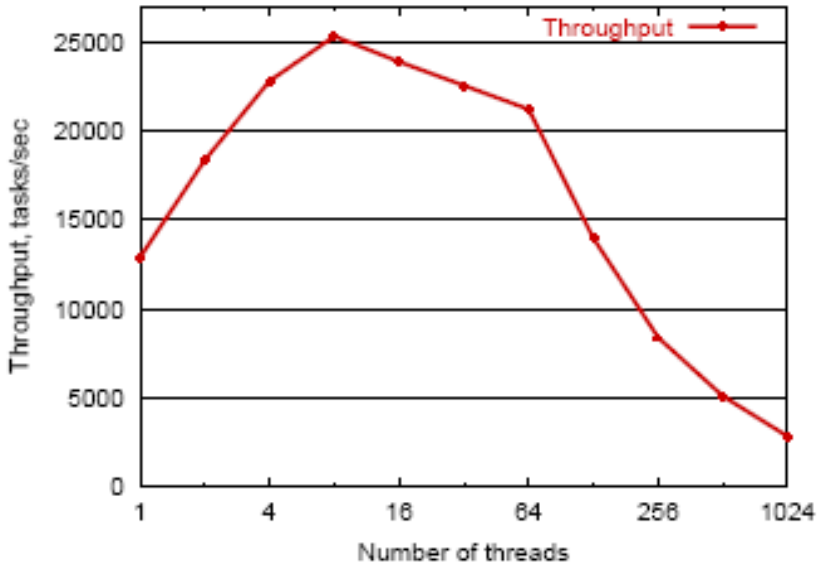
- Then relationship between Q, λ, and R:

$$Q = \lambda R$$

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

Discussion: How to Address the Issue





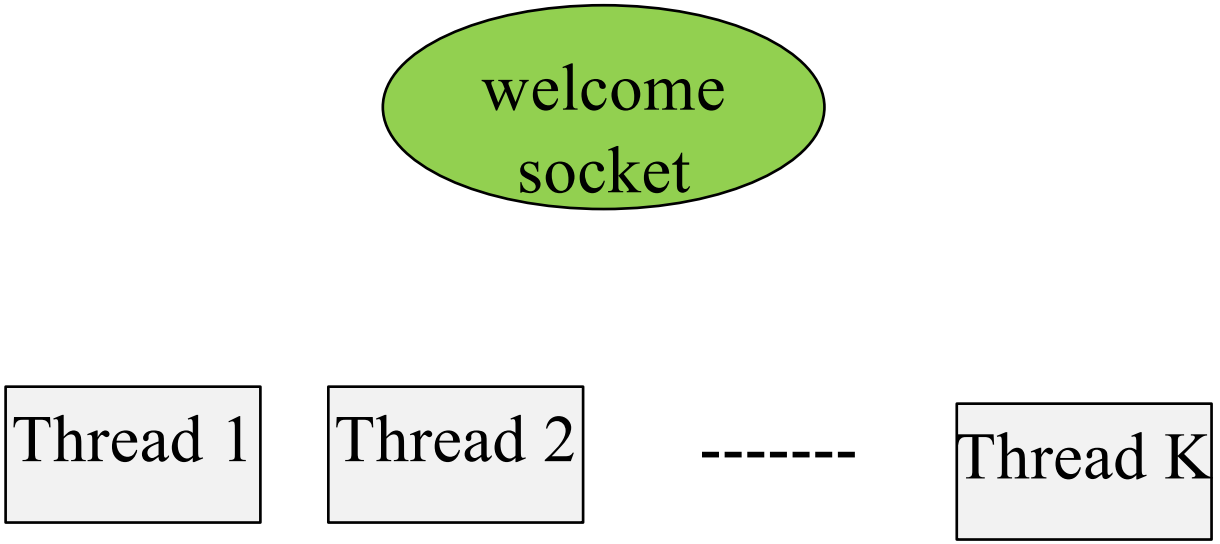
(937 MHz x86, Linux 2.2.14, each thread reading 8KB file)

Outline

- Admin and recap
- High-performance network server design
  - Overview
  - Threaded servers
    - Per-request thread
      - problem: large # of threads and their creations/deletions may let overhead grow out of control
    - *Thread pool*

Using a Fixed Set of Threads (Thread Pool)

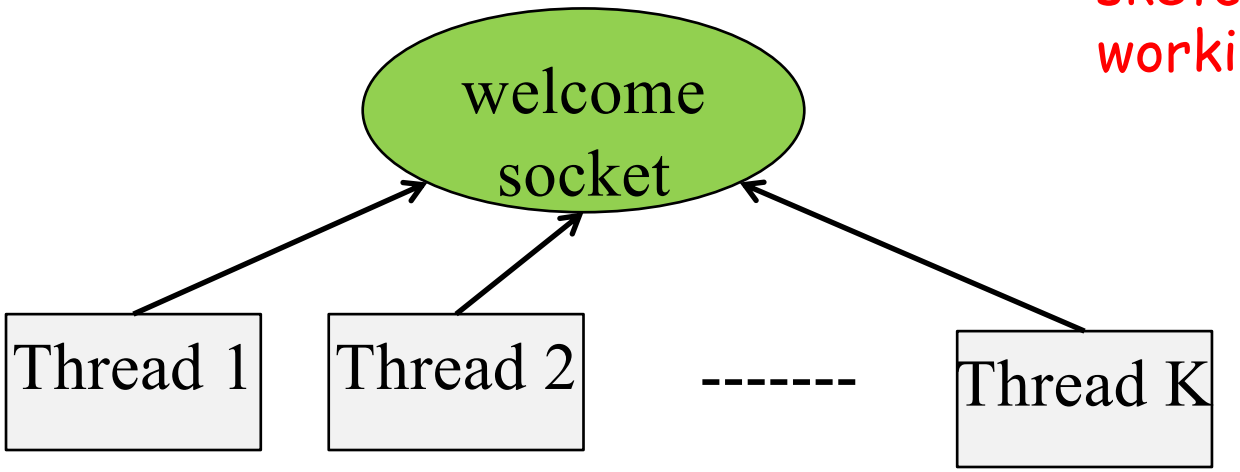
- Design issue: how to distribute the requests from the welcome socket to the thread workers



Design 1: Threads Share Access to the welcomeSocket

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

sketch; not working code

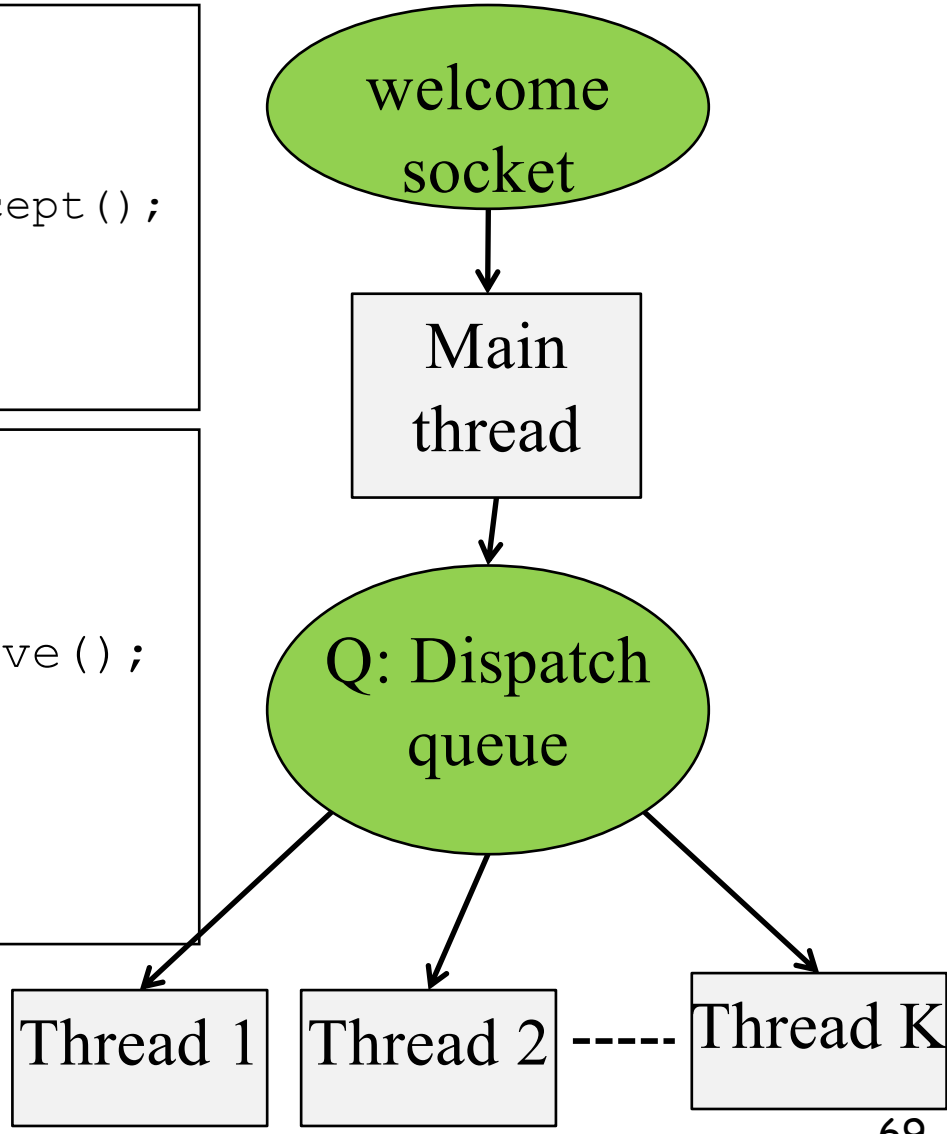


Design 2: Producer/Consumer

```
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?