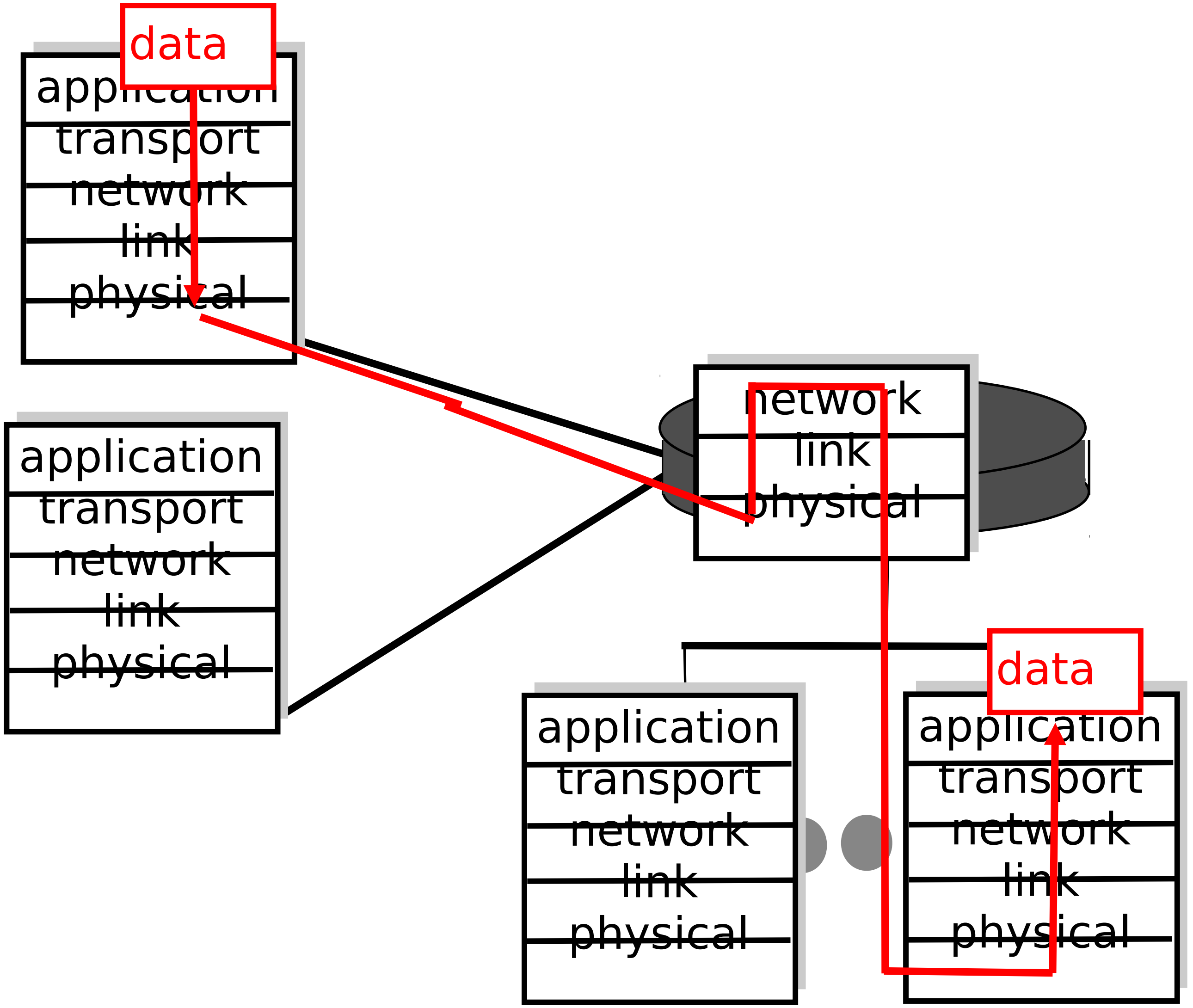
CS 716: Introduction to communication networks

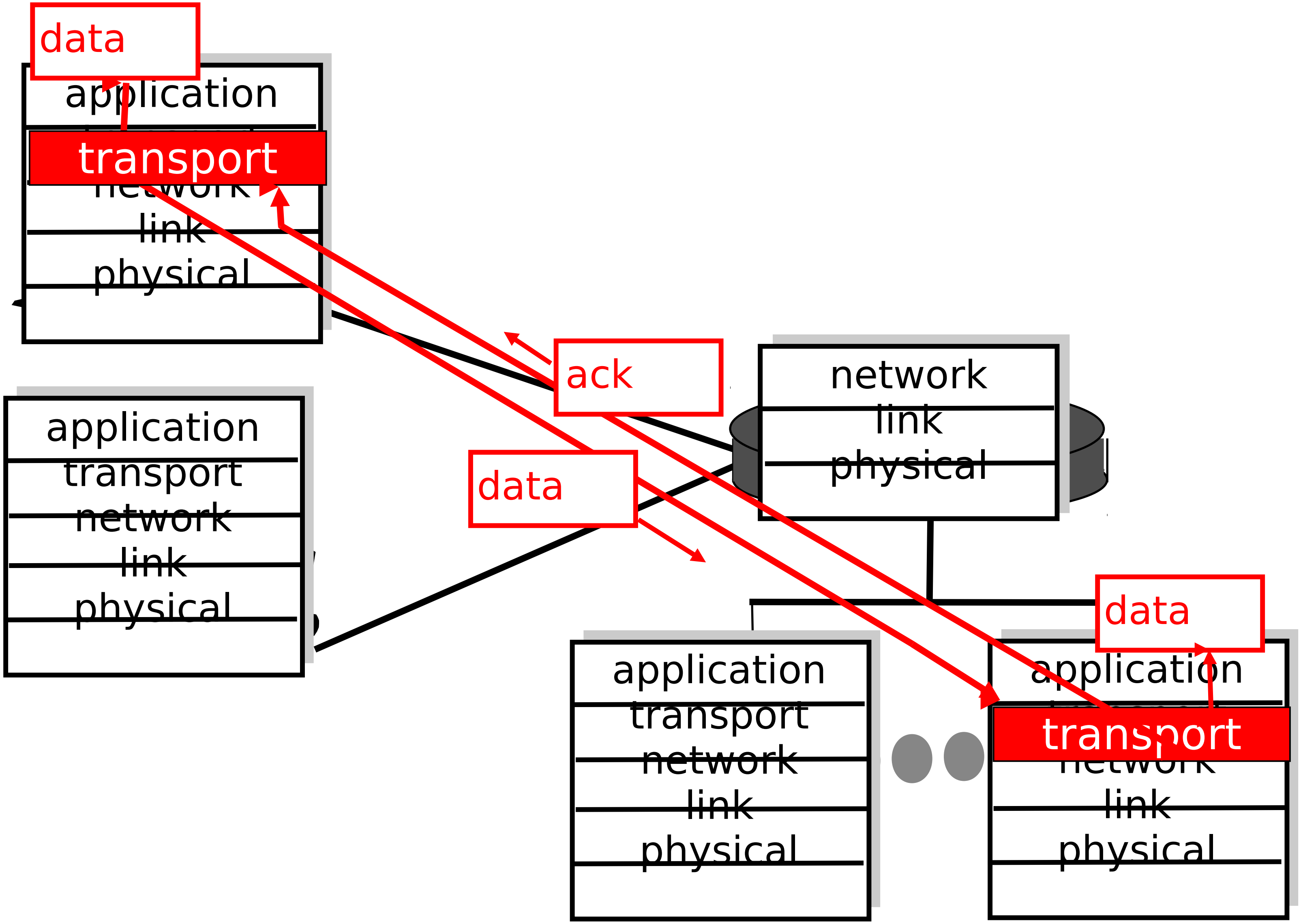
- 24th class; 11th Nov 2011

Instructor: Sridhar Iyer
IIT Bombay

Layering: physical communication



Layering: logical communication

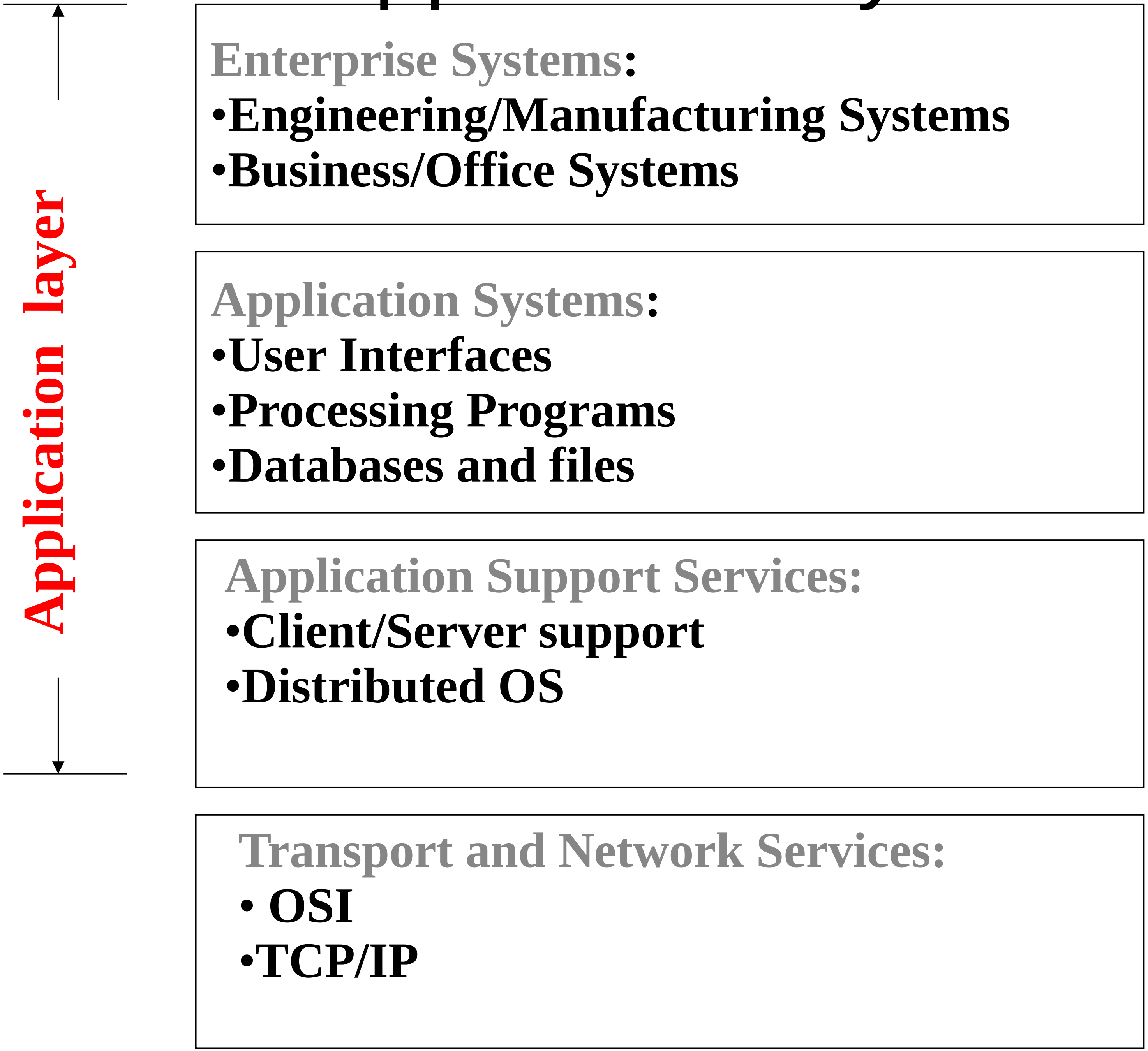


Application layer

- Application:
  - communicating, distributed processes
  - running in network hosts, in “user space”
  - exchange messages to implement application
  - e.g., email, file transfer, the Web

- Application layer protocols define messages exchanged by application components and actions taken. They use services provided by lower layer protocols.

Application layer



Application layer

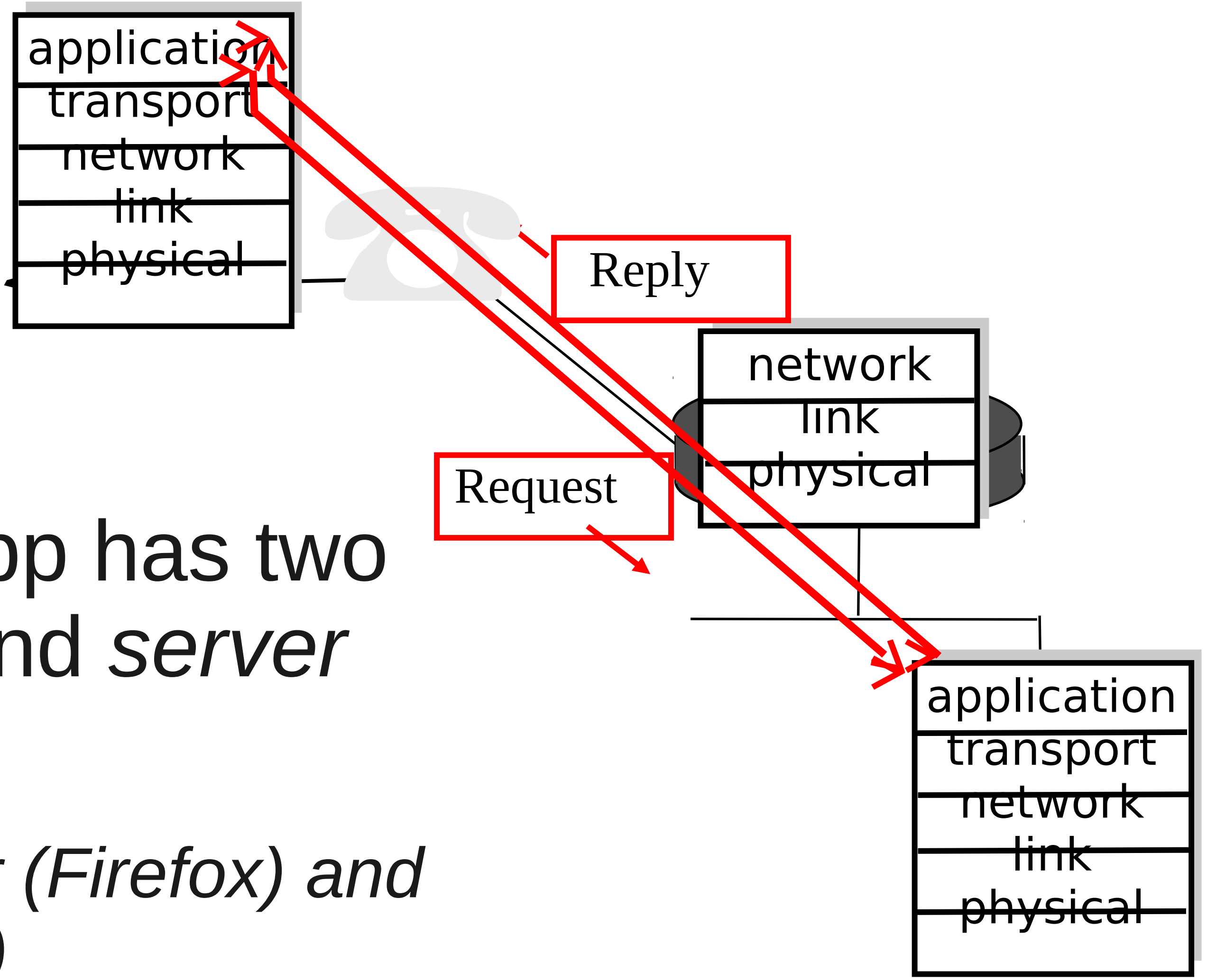
Enterprise Systems:
- **Engineering/Manufacturing Systems**
- **Business/Office Systems**

Application Systems:
- **User Interfaces**
- **Processing Programs**
- **Databases and files**

Application Support Services:
- **Client/Server support**
- **Distributed OS**

Transport and Network Services:
- **OSI**
- **TCP/IP**

Client-Server paradigm



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

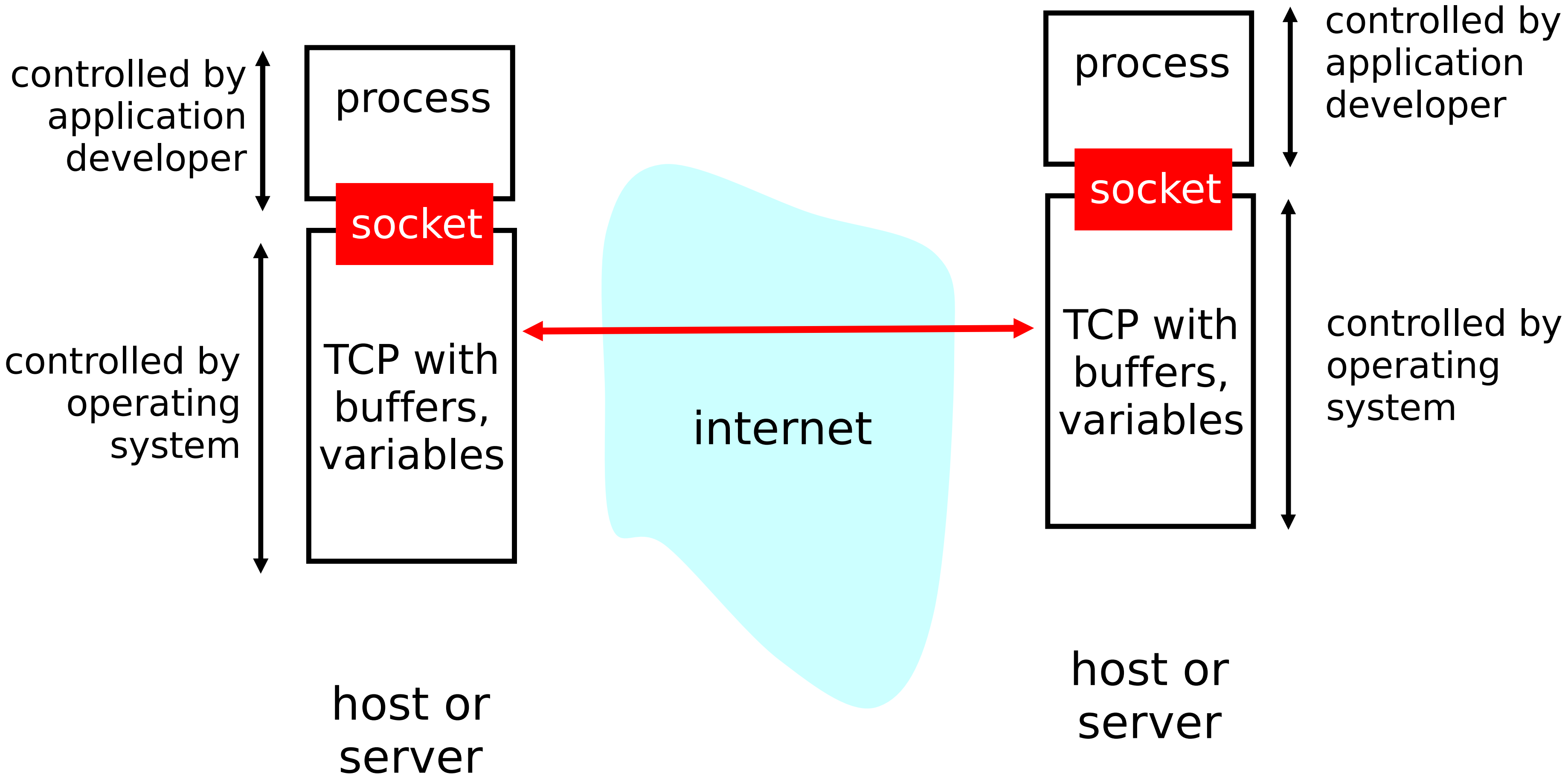
Example: Web browser (Firefox) and Web server (Apache)

Sockets API

- Interface between application and transport layer
  - two processes communicate by
  - sending data into a socket
  - reading data out of a socket

- Client “identifies” Server process using
- <IP address ; port number>

Sockets interface


controlled by application developer
process
socket
controlled by operating system
TCP with buffers, variables
host or server
internet
process
socket
controlled by application developer
TCP with buffers, variables
controlled by operating system
host or server

Client actions

- Create a socket (*socket*)
- Map server name to IP address (*gethostbyname*)
- Connect to a given port on the server address (*connect*)

- Client must contact server first!

Server actions

- Create a socket (*socket*)
- Bind to one or more port numbers (*bind*)
- Listen on the socket (*listen*)
- Accept client connections (*accept*)
- Server process must be running!

- Homework Question: What is the difference between *bind* and *listen*?

Client architecture

- Simpler than servers
  - Typically do not interact with multiple servers concurrently
  - Typically do not require special ports

- Most client software executes as a conventional program
- Clients, unlike servers, do not require special privileged ports
- Most clients rely on OS for security

Server Architecture

Type of connection:

•Connection-Oriented: reliable but needs OS resources

•Connection-less: needs less resources but application has to handle loss of messages

Server State:

•Stateless: each transaction is independent, crash transparent

•Stateful: server maintains state, faster but expensive for server

Servicing of Requests:

•Iterative: accept requests one at a time

•Concurrent: fork a new process for each client; can service multiple clients but needs more resources

Super server process: inetd

- Common services have dedicated port numbers
- inetd binds to all ports required
- Selects and accepts incoming client calls
- Forks program that provides port-specific service and continues

inetd (Internet daemon)

Lines from /etc/services.conf

Client	Server	Port
Mail	smtpd	25
Telnet	telnetd	23
FTP	ftpd	20, 21
Browser	httpd	80
SNMP	snmpd	161
NFS	nfsd	2049

Lines from /etc/inetd.conf

```
ftp      stream  tcp      nowait  root     /usr/sbin/tcpd in.ftpd -l -a
telnet   stream  tcp      nowait  root     /usr/sbin/tcpd in.telnetd
```

Some Application Protocols

- Activity: Design the protocol (client-server interaction) and the server architecture for these applications.

- DNS
- FTP
- HTTP
- WAP

Domain Name System (DNS)

- Map between host names and IP addresses
  - IP address (32 bit) - used for addressing datagrams
  - “name”, e.g., www.iitb.ac.in - used by humans

- DNS provides logical hierarchical view of the Internet
  - globally *distributed database* implemented in hierarchy of many *name servers*
  - *application-layer protocol* to communicate to *resolve* names (address/name translation)
  - client/server interaction

DNS clients and servers

- clients: query servers to resolve names; nslookup
- servers: name server daemons, reply to queries; BIND, named
- gethostbyname: resolver library call that can be invoked from application program
- Lazily validated cache for performance

- *Should DNS be a centralized server or distributed?*

DNS hierarchy

- Servers are organized in a hierarchy
- Each server has an authority over a part of the naming hierarchy

- Name is a unique domain suffix is assigned by Internet Authority.
- No limit on number of subdomains or number of levels.

- The server does not need to keep all names
- It needs to know other servers who are responsible for other subdomains

DNS: Local name servers

Local Name Servers:

- each organization/ISP has *local (default) name server*
- host DNS query first goes to local name server

Authoritative Name Server:

- for a host: stores that host’s IP address, name
- can perform name/address translation for that host’s name

DNS: Root name servers

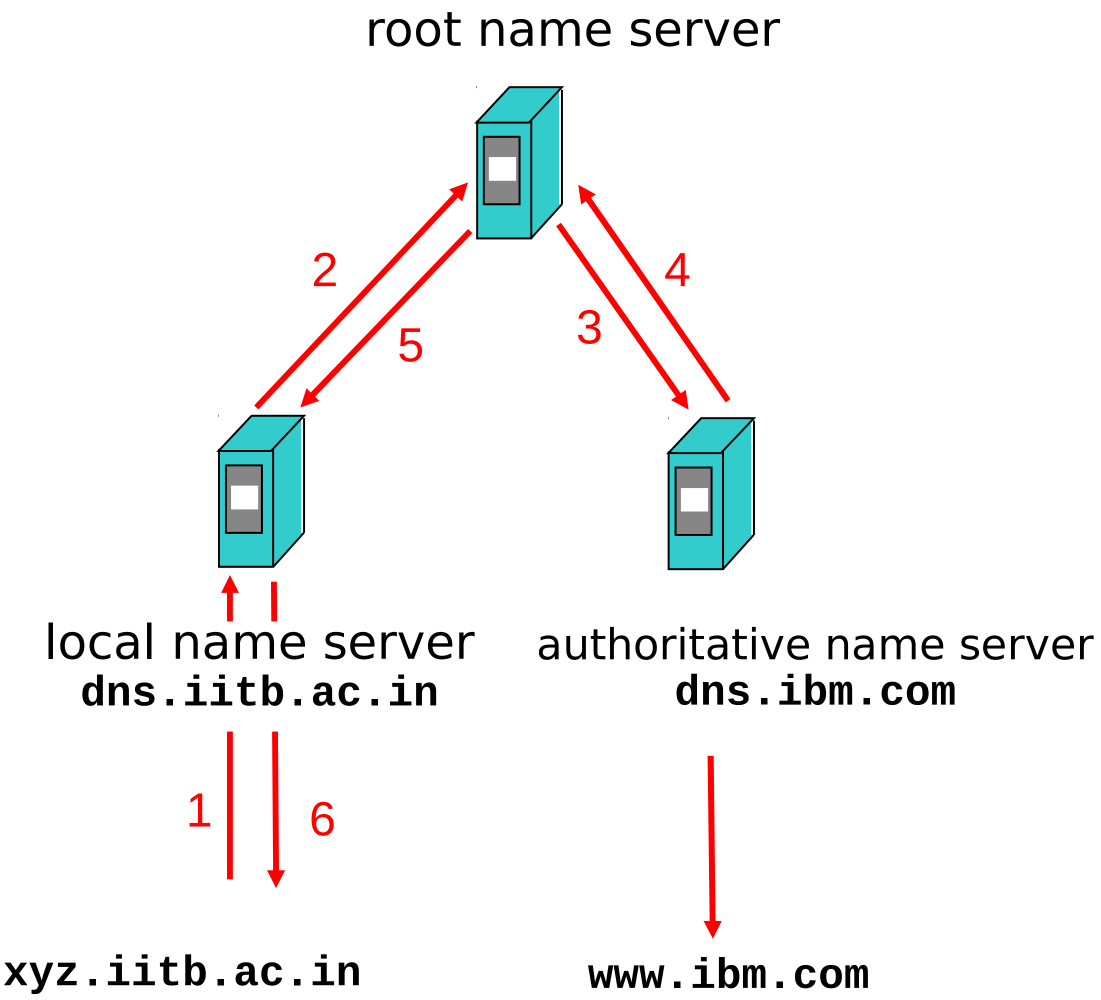
- Contacted by local name server that cannot resolve name
- Root Name Server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server

Several root name servers worldwide

DNS: Example

host xyz.iitb.ac.in wants IP address of www.ibm.com

1. Contacts its local DNS server, dns.iitb.ac.in
2. dns.iitb.ac.in contacts root name server, if necessary
3. root name server contacts authoritative name server, dns.ibm.com, if necessary

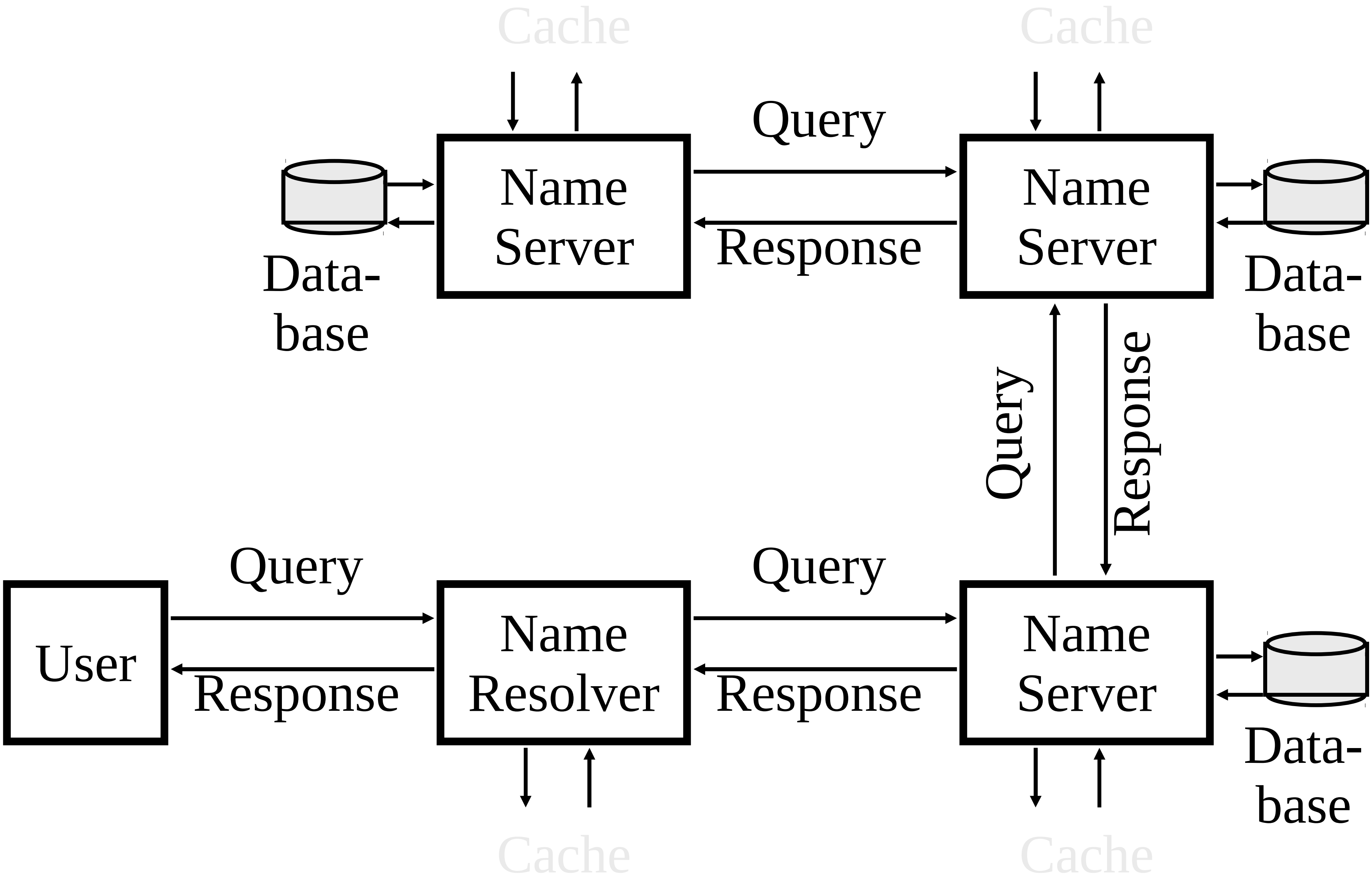


DNS: Name resolution

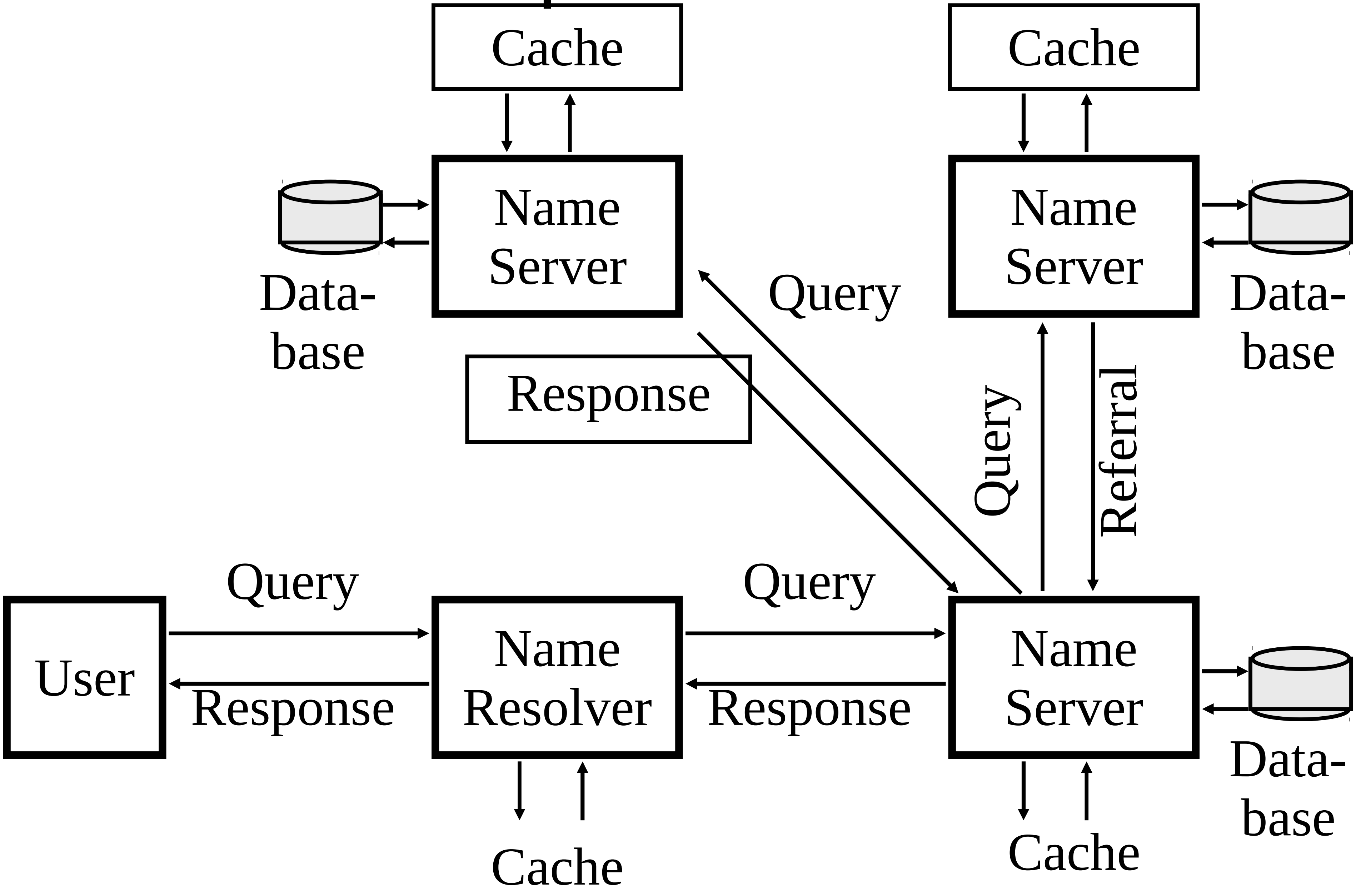
- Recursive queries:
  - puts burden of name resolution on contacted name server
  - not scalable under heavy load

- Iterated queries:
  - contacted server replies with name of server to contact. Ex: root name server may know *intermediate name server* to contact to find authoritative name server

Recursive queries



Iterated queries



Self Study: File Transfer Protocol

- FTP: Transfer file to/from remote host
- client/server model
  - *client:* side that initiates transfer (either to/from remote)
  - *server:* remote host
  - Uses TCP as transport protocol

SMTP: Simple Mail Transfer Protocol

- Three Components:
  - user agents: “mail readers”: For composing, editing, reading mail messages.

  - mail servers: “mailbox”for incoming messages ; “mail queue” for outgoing messages, are stored on server.

  - smtp: protocol between mail servers
    - client: sending mail server
    - server: receiving mail sever

SMTP functioning

- Three phases of transfer:
  - handshaking: Connection establishment
  - transfer: direct transfer from sending server (client) to receiving server (server); push-based: client sends data instead of server
  - closure: Connection termination

SMTP: Example

- C: telnet mailServer.iitb.edu 25

  S: 220 mailServer.iitb.edu
- C: Helo myServer.edu

  S: 250 Hello myServer.edu, pleased to meet you
- C: MAIL FROM: <xyz@myServer.edu>

  S: 250 xyz@myServer.edu... Sender ok
- C: RCPT TO: <abc@mailServer.iitb.edu>

  S: 250 abc@mailServer.edu ... Recipient ok
- C: DATA

  S: 354 Enter mail, end with "." on a line by itself
- C: Hi abc,

  C: This is uvw pretending to be xyz.

  C: .

  S: 250 Message accepted for delivery
- C: QUIT

  S: 221 mailServer.iitb.edu closing connection

Self study: HTTP

- client initiates TCP connection (creates socket) to server
- server accepts TCP connection from client

  - http messages (application-layer protocol messages) exchanged between browser (http client) and WWW server (http server)

- http is “stateless”: server maintains no information about past client requests

Closure

- Application protocols animations
  - http://oscar.iitb.ac.in/
  - Find others on your own – Google search.
- Tutorial Questions:
  - How does a HTTP server demultiplex incoming requests?
  - Explain the difference between stateful servers and stateless servers.
- Topics NOT covered
  - Socket programming; Other application designs – p2p, soa.