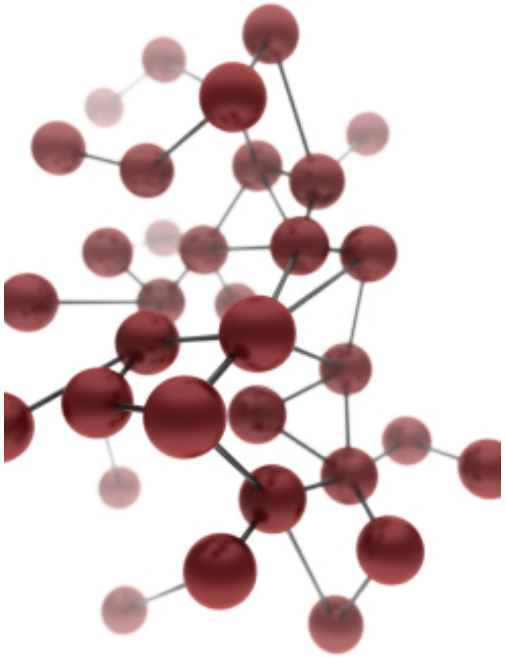


Lecture 9:

FTP (Project 2) and DNS

Today: Project 2 and DNS

- FTP: Project 2 overview

- Domain Name System (DNS)
  - What's behind (e.g.) people.cs.uchicago.edu?

Project 2

- Efficient Parallel FTP client using socket API
  - Use C/C++ to implement
  - Individual work, no collaboration

- Two parts
  - Basic ftp operation (65%)
  - Multi-threaded downloads (35%)

What you need to do for Project 2

- Analyze user input
  - Command line options
  - Parallel configuration file
- Use socket API to interact with FTP
  - FTP protocol: RFC959, http://rfc.net/rfc959.html
  - Basic socket programming
  - Analyze response from server
- Use parallel sessions for different servers

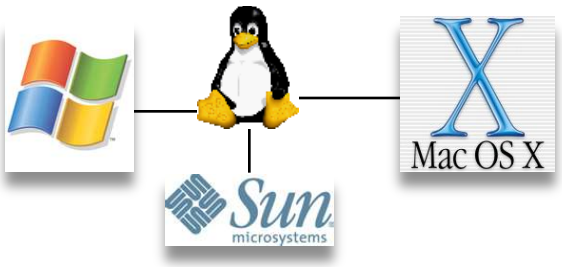
Why do we need a FTP Service?

- Purpose: To Transfer files between two computers

- Goals of FTP Service
  - Promote sharing of files (programs and/or data)
  - Encourage indirect/implicit use of remote computers
  - Shield users from variations in file storage among hosts
  - Transfer data reliably and efficiently

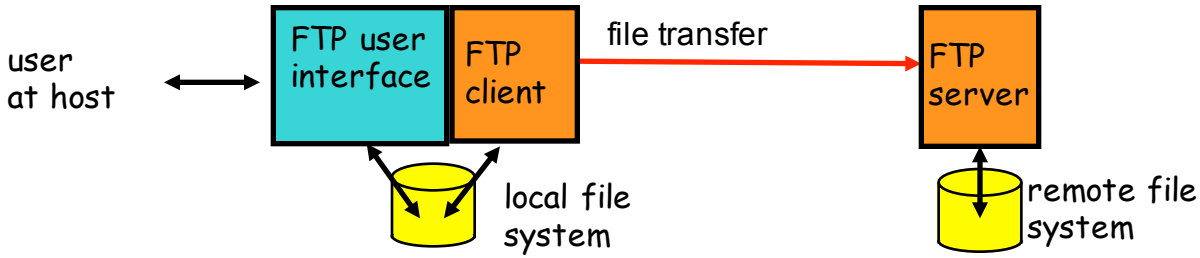
Problems of File Transfer

- At first, file transfer may seem simple
- Heterogeneous systems use different:
  - Operating Systems
  - Character Sets
  - Naming Conventions
  - Directory Structures
  - File Structures and Formats



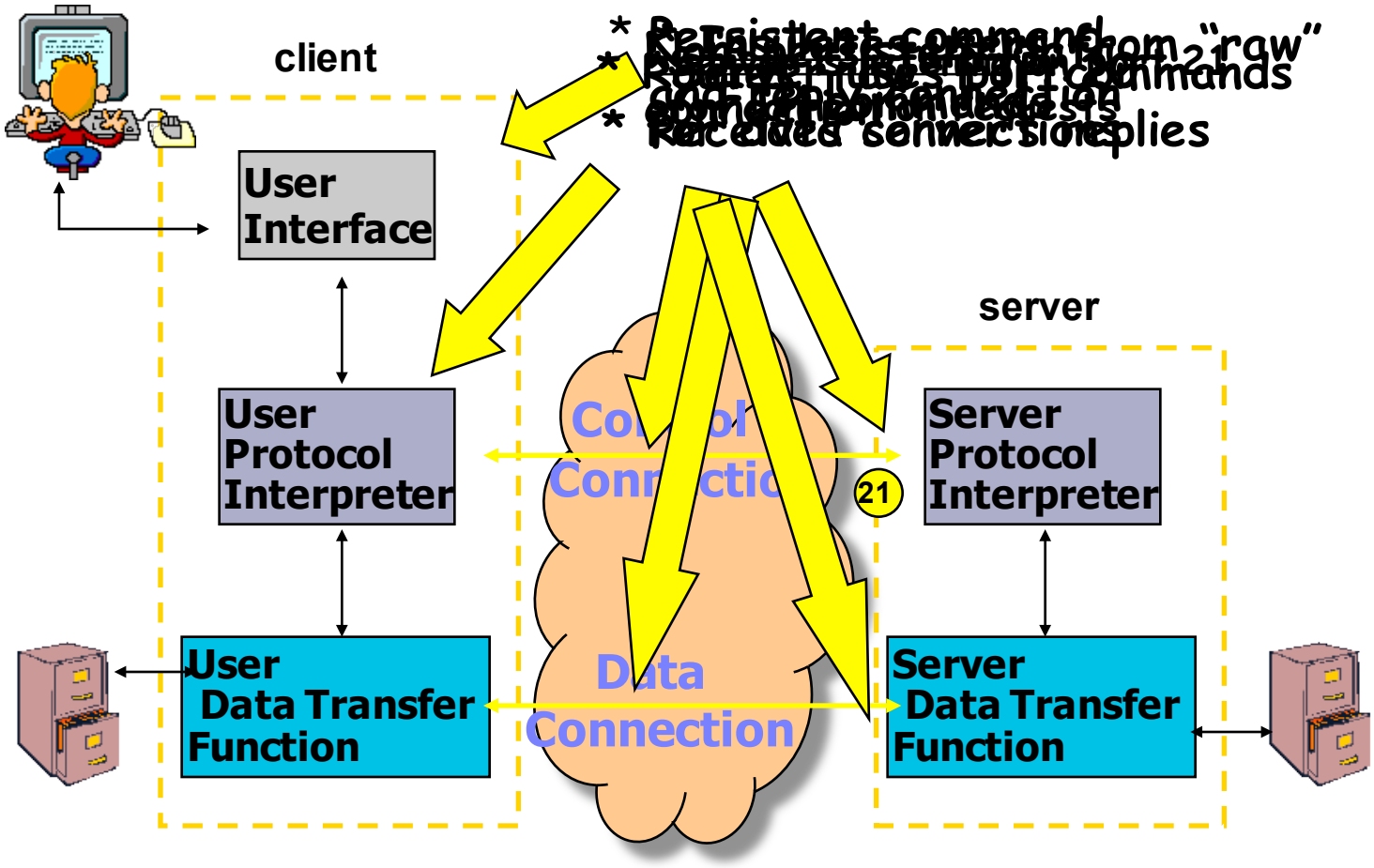
- FTP needed to address and resolve these problems

FTP: the file transfer protocol

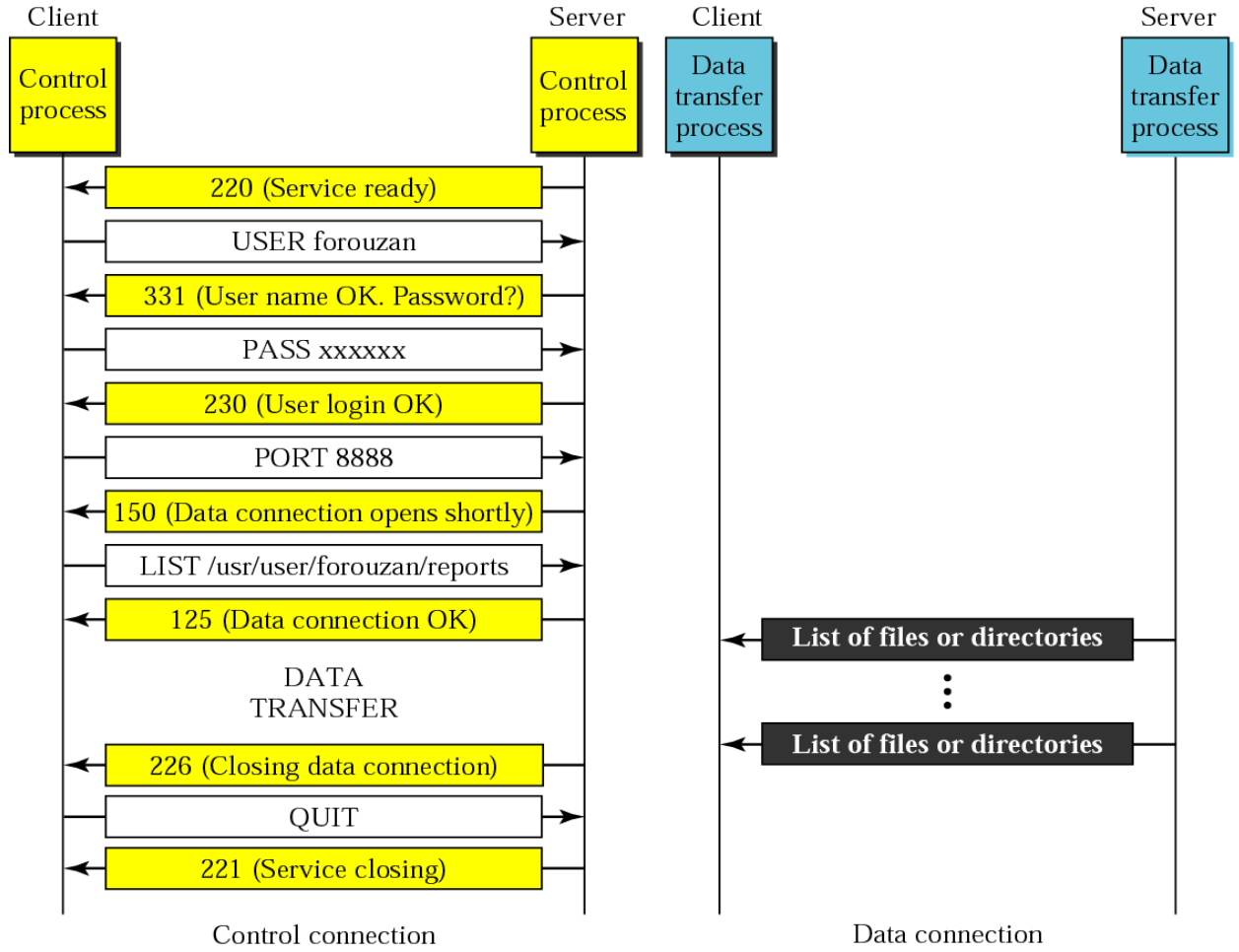


- Transfer file to/from remote host
- client/server model
  - client: side that initiates transfer (either to/from remote)
  - server: remote host
- ftp: RFC 959
- ftp server: port 21

FTP's Two Connections



FTP Connections



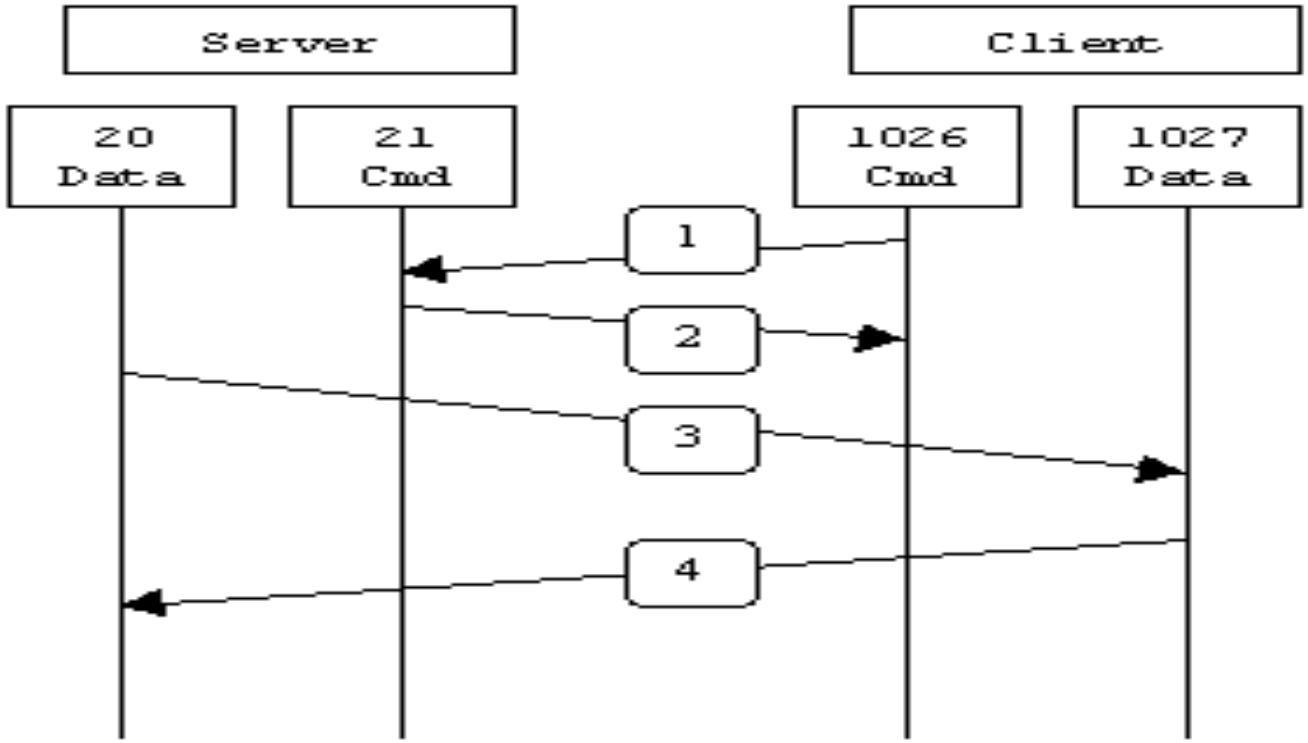
Summary of FTP connections

- FTP has 2 connections
- Control (persistent connection)
  - *Server issues a passive open on well-known 21*
  - *Client uses an ephemeral port to issue active open*
  - *Server ultimately closes control connection*
- Data (ephemeral connection)
  - *Client issues passive open on an ephemeral port*
  - *Client sends this port to server via PORT command*
  - *Server receives the port number and issues active open using its well-known 20 to the received ephemeral port*

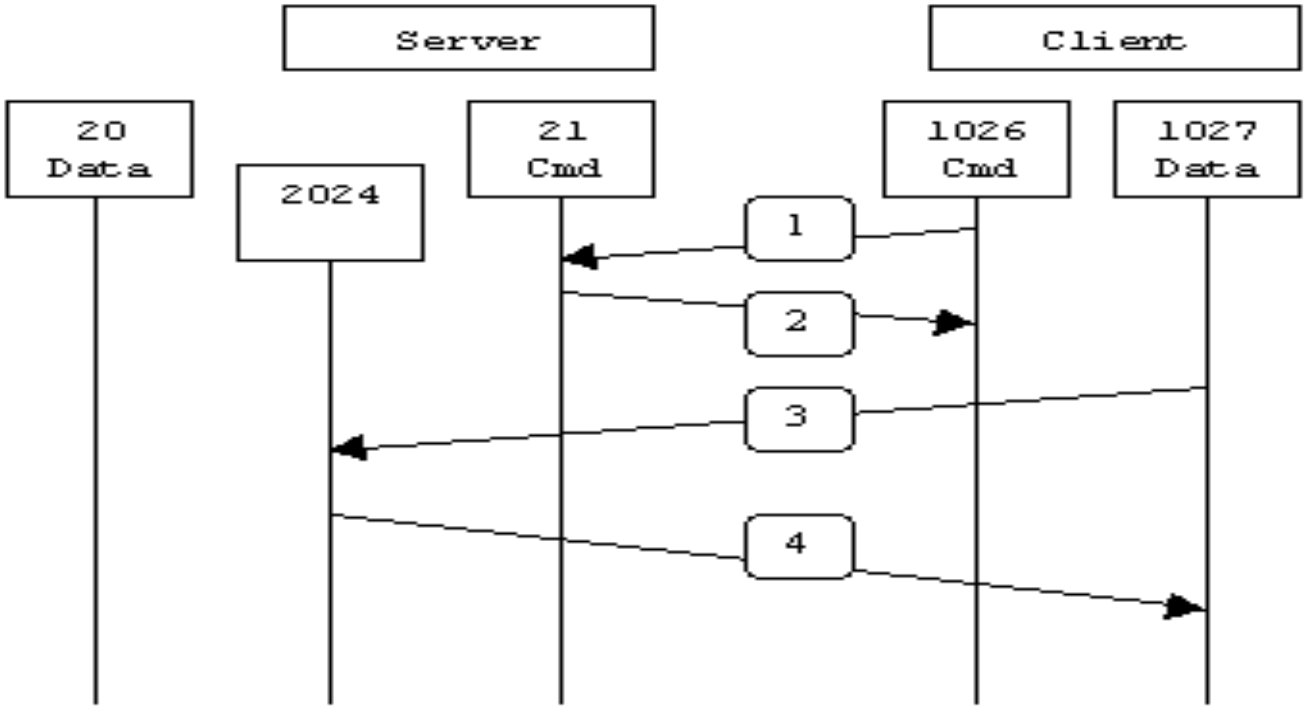
Data Connection continued

- This does not always work…why?
- Instead, use PASV command
  - *Client sends PASV command to server*
  - *Server chooses ephemeral port: passive open*
  - *Server responds with IP, Port in reply (227)*
  - *Client issues active open to server's port*
  - *Don't use server port 20*
- Ultimately, the data sender closes connection

Active Mode (PORT)



Passive Mode (PASV)



Passive vs. Active Connections

- Passive is the default connect type of most FTP clients
  - *It can be changed by user settings*

- How to choose Passive or Active?
  - *Client has firewall: Passive Mode*
  - *Server has firewall: Active Mode*

FTP Commands

Command	Description
LIST [filelist]	List files or directories (ls / dir)
USER username	Send username to server
PASS password	Password on server
PORT h1,h2,h3,h4,p1,p2	Active Mode, Client IP and port number
PASV	Passive mode
RETR filename	Retrieve (get) filename
REST rest	Restart file transfer
CWD dir	Change directory
STOR filename	Store (put) filename
TYPE type	Set representation type

FTP Client Commands (issued by user interface)

Command	Description
get filename	Retrieve file from server
mget filename*	Retrieve multiple files from server*
put filename	Copy local file to server
mput filename*	Copy multiple local files to server*
open server	Begin login to server
bye / close / exit	Logoff server
ls / dir	List files in current remote dir on server
lcd	Change local directory
cd	Change remote directory
rhelp / remotehelp	Lists commands the server accepts

* Sent to server as multiple command by User Protocol Interpreter

Example FTP Responses

- 120 Service will be ready shortly
- 200 Command OK
- 230 User login OK
- 331 User name OK; password is needed
- 421 Service not available
- 530 User not logged in
- 552 Requested action aborted; exceeded storage allocation

NetCat Test: Control Connection

- Connect to port 21: default control port

-bash-3.00$ nc ftp1.cs.uchicago.edu 21

220 Authorized Users Only

USER xxxx

331 Please specify the password.

PASS xxx

230 Login successful.

TYPE A

200 Switching to ASCII mode.

CWD

250 Directory successfully changed.

LIST

425 Use PORT or PASV first.

QUIT

221 Goodbye.

NetCat Test: Data Connection

- Passive Mode: Send PASV
  - *Server's response to this command includes the host and port address this server is listening on (h1,h2,h3,h4,p1,p2)*
  - *h1.h2.h3.h4 is server IP address, P1*256+p2 is server port number*
- Active Mode
  - *Choose a local port no = p1*256+p2*
  - *Send PORT h1,h2,h3,h4,p1,p2*
- Need another NC process
  - *Passive: nc server serverportno*
  - *Active: nc –l localportno*

NetCat Test: Passive Mode

- NC process 1: Control connection

  -bash-3.00$ nc ftp1.cs.uchicago.edu 21

  …

  LIST

  425 Use PORT or PASV first.

  PASV

  227 Entering Passive Mode (128,135,164,133,248,82)

  LIST

- NC process 2: Data connection

  server port no = 248*256 + 82 = 63570

  -bash-3.00$ nc ftp1.cs.uchicago.edu 63570

NC process 2 should start to connect after receiving PASV response

NetCat Test: Active Mode

- NC process 1: Control connection

  -bash-3.00$ nc ftp1.cs.uchicago.edu 21

  …

  LIST

  425 Use PORT or PASV first.

  PORT 128,111,41,12,100,0

  200 PORT command successful. Consider using PASV.

  LIST

- NC process 2: Data connection

  *100*256+0=25600*

  Local IP address: 128.111.41.12

  -bash-3.00$ nc -l ***25600***

  ```
  -rw-r--r--   1 14      50      10485760 Mar 13  2004 10.MB
  -rw-r--r--   1 14      50      52428800 May 31  2004 50.MB
  lrwxrwxrwx   1 0       0             38 Mar 11  2006 base-9 ->
      mirrors/fedoralegacy/redhat/9/os/i386/
  ```

NC process 2 should start to listen before sending PORT

FTP Raw Commands Useful for Project 2

- USER, PASS
- TYPE, PASV
- REST, RETR
- QUIT

- FTP control log
  - *https://en.wikipedia.org/wiki/List_of_FTP_server_return_codes*
  - *Find those that are relevant to pftp*

Exit code	Explanation
0	Operation successfully completed
1	Can't connect to server
2	Authentication failed
3	File not found
4	Syntax error in client request
5	Command not implemented by server
6	Operation not allowed by server
7	Generic error

Project #2: A Parallel FTP Client

The project is due Monday, November 19th at 11:59pm. All assignments are to be carried out individually. No collaboration is allowed. The detailed project overview was presented in Lecture 9.

The Assignment

In this assignment, you will implement a parallel FTP **client** application using the socket API. The application is called ***pftp***, and must be implemented in C or Python. The FTP protocol is specified in RFC 959. The assignment can be broken down into two components: basic ftp operation and parallel download. Both components must be completed by the due date.

Part 1: Basic FTP Operation (65%)

The client application will take an FTP server and a file name as command-line parameters. When executed, the client will simply retrieve the file specified following the FTP protocol and save it to a local file with the same name.
Note: you will not need to implement the put operation; and each FTP session operates in the default passive mode, which we have introduced in Lecture 9.

Usage:
pftp [-s hostname] [-f file] [options]
pftp -h | --help
pftp -v | --version

For example, by executing:

% pftp -s ftp://mirror.keystealth.org/gnu/ -f /ProgramIndex

The file named ProgramIndex will be created in the current directory.

On success, the ***pftp*** application exits with code 0. If an error occurs, the application exits with the appropriate code and prints a human-readable error message on standard error.

More details on the command line:

-h or --help
: Prints a synopsis of the application usage and exits with return code 0.

-v or --version
: Prints the name of the application, the version number (in this case the version has to be 0.1), the author, and exits, returning 0.

[-f *file*] or [--file *file*]
: Specifies the file to download.

[-s *hostname*] or [--server *hostname*]
: Specifies the server to download the file from

Options:

[-p *port*] or [--port *port*]
: Specifies the port to be used when contacting the server. (default value: 21).

[-n *user*] or [--username *user*]
: Uses the username *user* when logging into the FTP server (default value: anonymous).

[-P *password*] or [--password *password*]
: Uses the password *password* when logging into the FTP server (default value:user@localhost.localnet).

[-m *mode*] or [--mode *mode*]
: Specifies the mode to be used for the transfer (ASCII or binary) (default value: binary).

[-l *logfile*] or [--log *logfile*]
: Logs all the FTP commands exchanged with the server and the corresponding replies to file *logfile*. If the filename is "-" then the commands are printed to the standard output. The lines must be prepended by C->S: or S->C: for client-to-server and server-to-client lines, respectively. For example:

  S->C: 220 Server (Uchicago) [128.135.164.133]
  C->S: USER anonymous
  S->C: 331 Anonymous login ok, send your complete email address as your password.
  C->S: PASS user@localhost.localnet

Part 2: Parallel FTP Download (35%)

The next part of the assignment is to implement a parallel (multi-track) ftp download. The idea is that the ***pftp*** client uses separate threads to simultaneously connect to multiple servers, and download the same file from different positions in the same file. On the server side, this functionality is already supported by current ftp servers through the "restart marker" option, i.e. REST. A REST request sets the start position.

There's no intuitive user interface for this type of functionality. So we will use something very simple and straightforward. ***pftp*** takes in an additional command line argument:
[-t para-config-file] or [--thread config-file]

Usage:
pftp [-t para-config_file] [options]

Each line in the config-file specifies the login, password, hostname and absolute path to the file. Each line should be separated by a Line Feed (LF) character.

When ***pftp*** reads in the para-config-file, it parses the N servers on the list. For each of the N servers, ***pftp*** spawns a separate thread, connects to the server, and moves to the specified directory. To download a file of S bytes, ***pftp*** performs a joint download, where each thread downloads a segment of S/N bytes. The downloaded segments should be put together to form

the entire file. Note: ***pftp*** can use the --thread option simultaneously with the -l option. And ***pftp*** *only supports binary model not ASCII mode.*

For each line of the para-config file, the format should be:
ftp://username:password@servername/file-path.

An example of the para-configuration file is as follows (which will be used as a test for your program)
ftp://cs23300:youcandoit@ftp1.cs.uchicago.edu/rfc959.pdf
ftp://socketprogramming:rocks@ftp2.cs.uchicago.edu/rfc959.pdf

Note that using a para-config_file, any arguments specified for username and password through options will be overridden by the values specified in the file.

Discussion
This application must be *extremely* resilient to wrong/unexpected input data, repeated values, etc. Part of your assignment is to understand how the ***pftp*** application could be abused by a user and to provide meaningful error messages if something goes wrong. You have to program defensively and with user-friendliness in mind. Segmentation fault is not an option!

You **CANNOT** use any **existing library** to retrieve the files from the server. You have to write your own code that will open connections, play the protocol, parse the data received from the server, etc.

The application should return meaningful error messages if an error occurs. In addition, the exit value should give some information about what happened. By returning different exit values in association with different errors the application becomes easily scriptable (that is, an external script can invoke the application and manage error conditions). Obviously, it is impossible to foresee all possible error conditions, therefore the following list is obviously incomplete, but this is expected from your program.

Exit code	Explanation
0	Operation successfully completed
1	Can't connect to server
2	Authentication failed
3	File not found
4	Syntax error in client request
5	Command not implemented by server
6	Operation not allowed by server
7	Generic error

Hints and Additional Notes

- Note that when invoked without any parameters the application should behave as if it were invoked with the --help option.
- All error message must be printed to standard error.
- If a parameter is specified multiple times, then the last value assigned to the parameter is the value accepted.
- List of FTP raw commands: http://www.nsftools.com/tips/RawFTP.htm
  Hint: the key commands that you should be using for this project include: PASV, REST, RETR, TYPE, USER, PASS, QUIT, etc.
- **Additional info on FTP can also be found in the lecture9.pdf**
- Tutorial for Multi-thread programming in C:
  https://www.geeksforgeeks.org/multithreading-c-2/

FTP Servers for testing your program:
We have set up two ftp servers:

ftp1.cs.uchicago.edu username: cs23300. Password: youcandoit
ftp2.cs.uchicago.edu username: socketprogramming Password:rocks

Two files were placed on both servers: rfc959.pdf, lecture8.pdf

Submission

Use the following dropbox upload link. You can submit multiple times but only the last submission will be used for grading.

https://www.dropbox.com/request/9CvMbG9PvJyzX4aq448G

Domain Name System (DNS)

Host Names & Addresses

- Host addresses: e.g., *128.135.250.222*
  - a number used by protocols
  - conforms to network structure (the "where")
- Host names: e.g., *groot.uchicago.edu*
  - mnemonic name usable by humans
  - conforms to organizational structure (the "who")
- The Domain Name System (DNS) is how we map from one to the other
  - a directory service for hosts on the Internet

Why bother?

- Convenience
  - Easier to remember www.google.com than 108.177.119.105

- Provides a level of indirection!
  - Decoupled names from addresses
  - Many uses beyond just naming a specific host

DNS: Early days

- Mappings stored in a hosts.txt file (in /etc/hosts)
  - maintained by the Stanford Research Institute (SRI)
  - new versions periodically copied from SRI (via FTP)

- As Internet grew, this system broke down
  - SRI couldn't handle the load
  - conflicts in selecting names
  - hosts had inaccurate copies of hosts.txt

- Domain Name System (DNS) invented to fix this
  - First name server implementation done by 4 Berkeley students!

Goals?

- Scalable
  - many names
  - many updates
  - many users creating names
  - many users looking up names
- Highly available
- Correct
  - no naming conflicts (uniqueness)
  - consistency
- Lookups are fast

How?

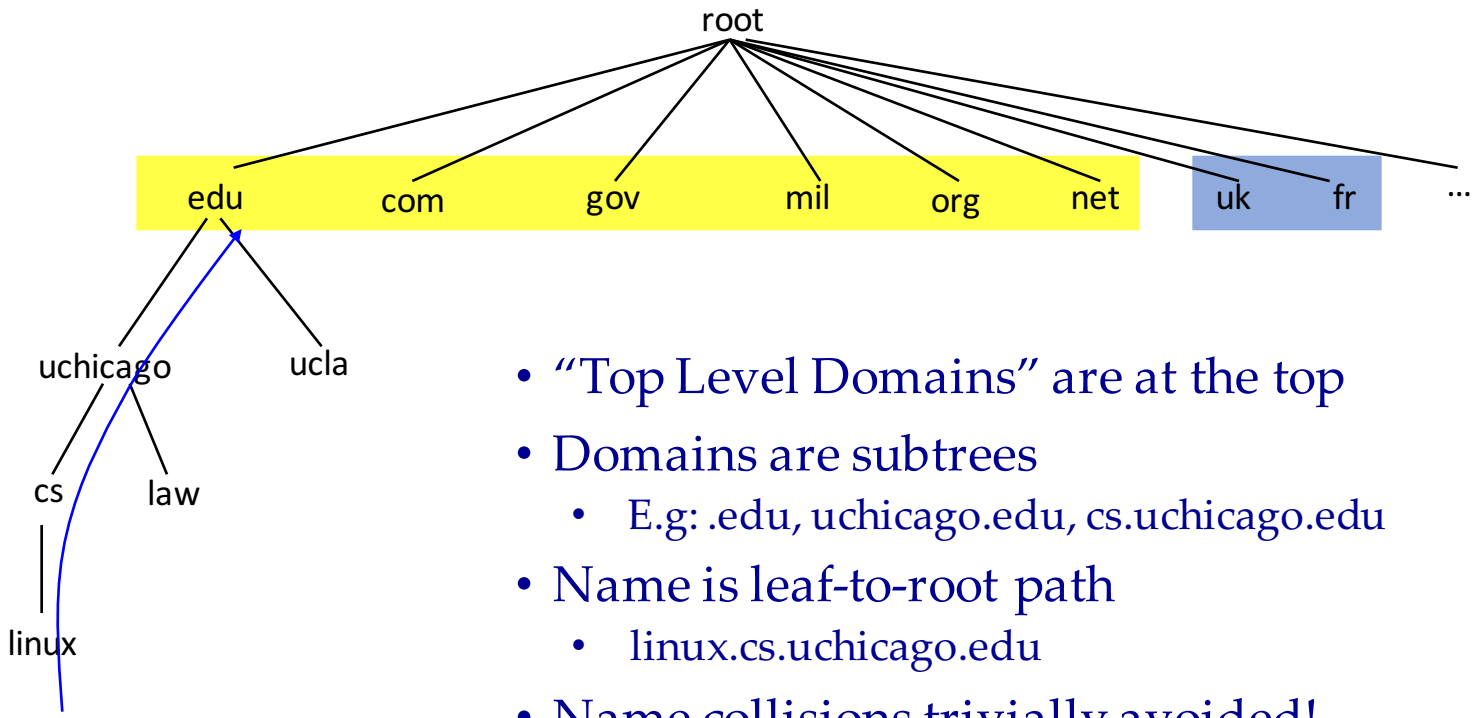
- Partition the namespace
- Distribute administration of each partition
  - Autonomy to update my own (machines') names
  - Don't have to track everybody's updates
- Distribute name resolution for each partition
- *How should we partition things?*

Key idea: hierarchical distribution

Three intertwined hierarchies

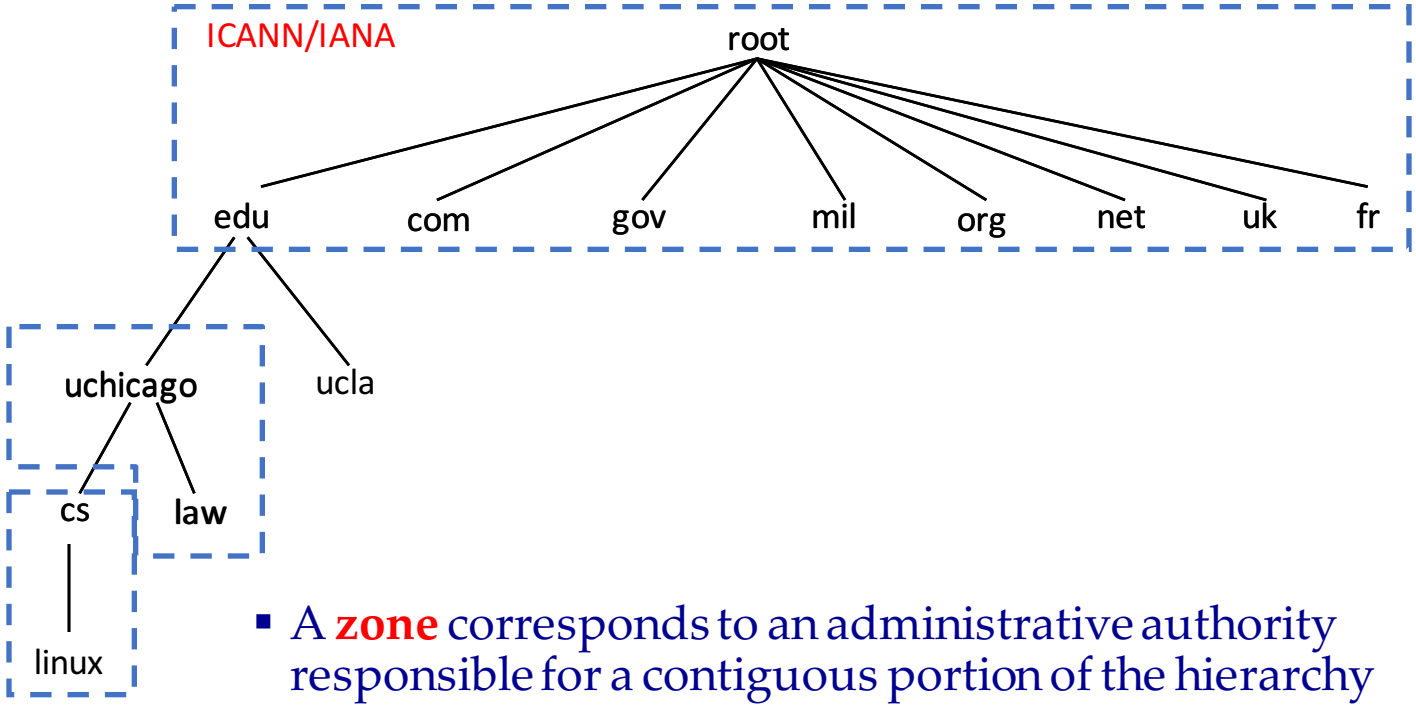
- Hierarchical namespace
  - As opposed to original flat namespace
- Hierarchically administered
  - As opposed to centralized administrator
- Hierarchy of servers
  - As opposed to centralized storage

Hierarchical Namespace



- "Top Level Domains" are at the top
- Domains are subtrees
  - E.g: .edu, uchicago.edu, cs.uchicago.edu
- Name is leaf-to-root path
  - linux.cs.uchicago.edu
- Name collisions trivially avoided!
  - each domain's responsibility

Hierarchical Administration



- A **zone** corresponds to an administrative authority responsible for a contiguous portion of the hierarchy
- E.g.: UChicago controls *law.uchicago.edu* and **.cs.uchicago.edu* while CS controls **.cs.uchicago.edu*

Server Hierarchy

- Top of hierarchy: Root servers
  - Location hardwired into other servers
- Next Level: Top-level domain (TLD) servers
  - .com, .edu, etc.
  - Managed professionally
- Bottom Level: Authoritative DNS servers
  - Actually store the name-to-address mapping
  - Maintained by the corresponding administrative authority

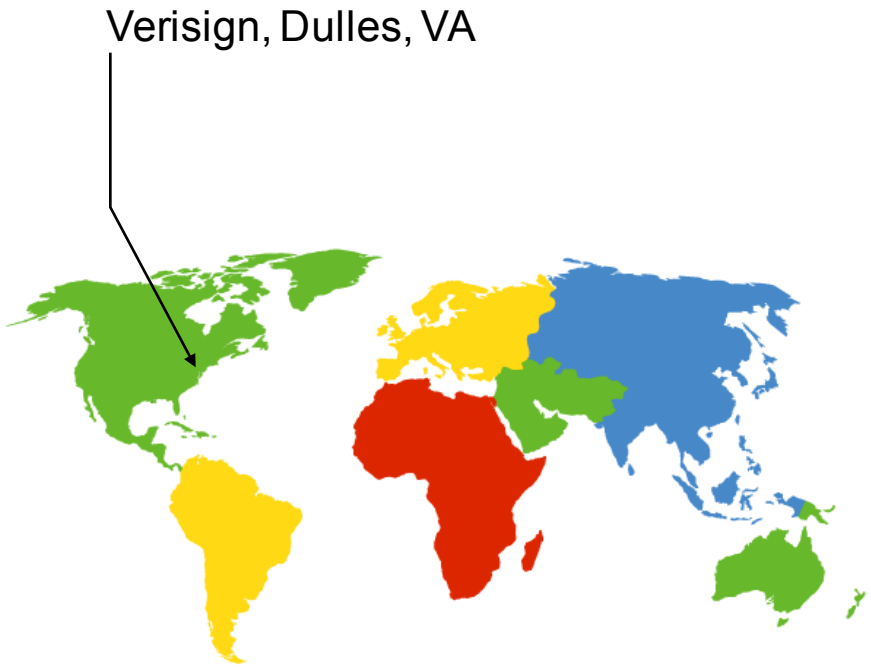
Server Hierarchy

- Every server knows the address of the root name server
  - Root servers know the address of all TLD servers
  - …
- An authoritative DNS server stores name-to-address mappings ("resource records") for all DNS names in the domain that it has authority for

- → Each server stores a subset of the total DNS database
- → Each server can discover the server(s) responsible for any portion of the hierarchy

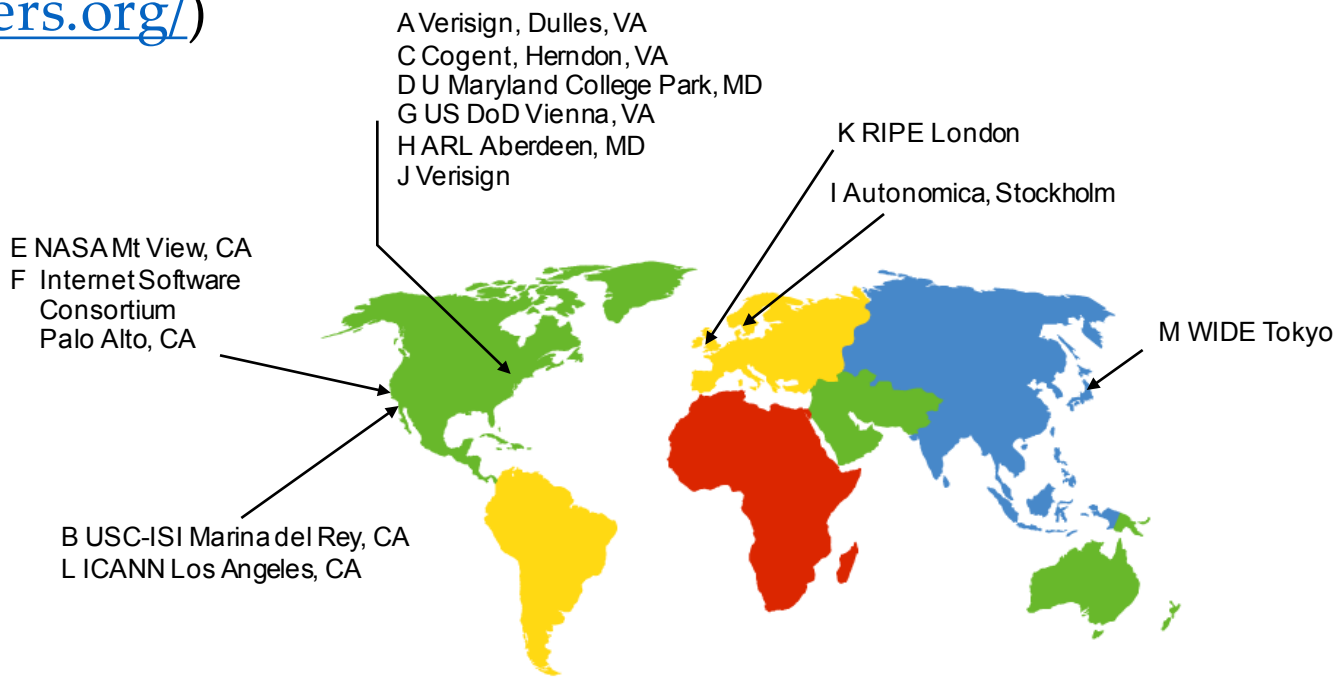
DNS Root

- Located in Virginia, USA



DNS Root Servers

- 13 root servers (labeled A-M; see http://www.root-servers.org/)



DNS Root Servers

- 13 root servers (labeled A-M; see http://www.root-servers.org/)
- Replicated

Anycast in a nutshell

- Routing finds shortest paths to destination
- What happens if multiple machines advertise the same address?
- The network will deliver the packet to the closest machine with that address
- This is called "anycast"
  - Very robust
  - Requires no modification to routing algorithms

DNS Records

- DNS servers store resource records (RRs)
  - RR is (name, value, type, TTL)

- Type = A: *(→ Address)*
  - name = hostname
  - value = IP address

- Type = NS: *(→ Name Server)*
  - name = domain
  - value = name of dns server for domain

DNS Records (cont'd)

- Type = MX: *(→ Mail eXchanger)*
  - name = domain in email address
  - value = name(s) of mail server(s)

Inserting Resource Records into DNS

- Example: you just created company "FooBar"
- You get a block of IP addresses from your ISP
  - say 212.44.9.128/25

- Register foobar.com at registrar (e.g., Go Daddy)
  - Provide registrar with names and IP addresses of your authoritative name server(s)
  - Registrar inserts RR pairs into the .com TLD server:
    - **(foobar.com, dns1.foobar.com, NS)**
    - **(dns1.foobar.com, 212.44.9.129, A)**

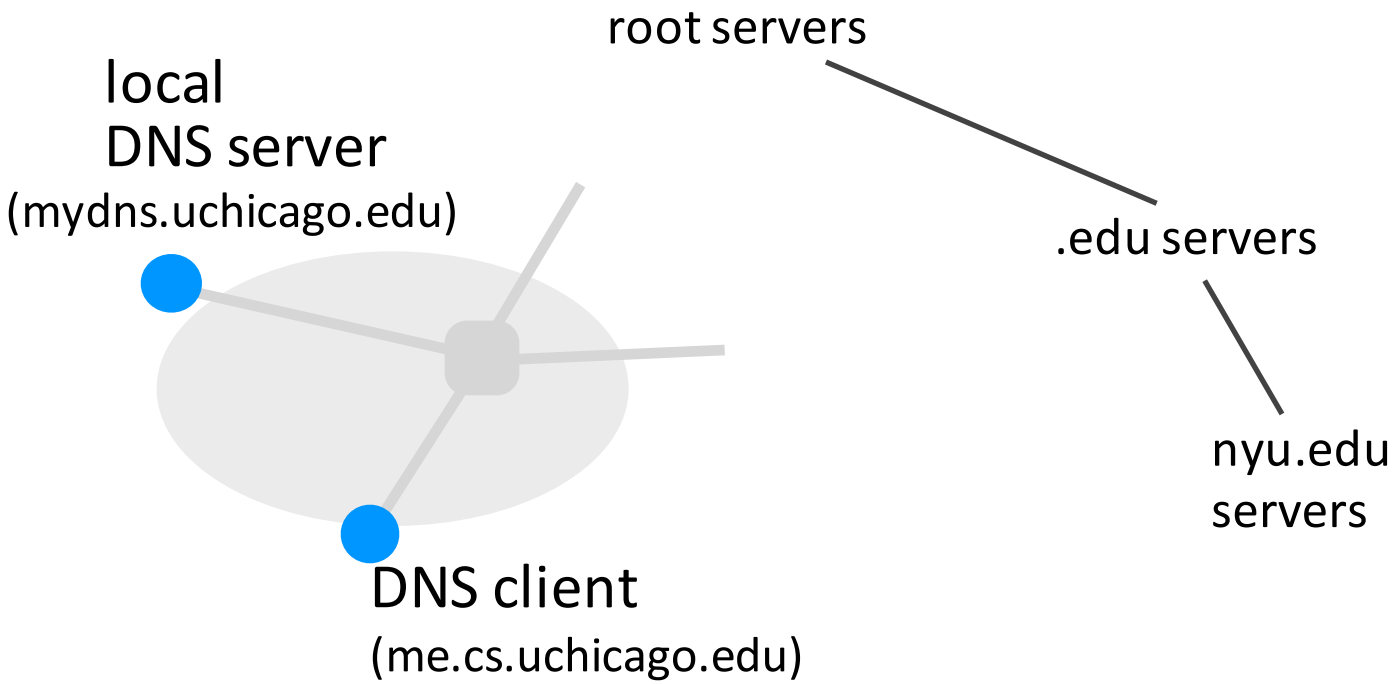
- Store resource records in your server dns1.foobar.com
  - e.g., type A record for www.foobar.com
  - e.g., type MX record for foobar.com

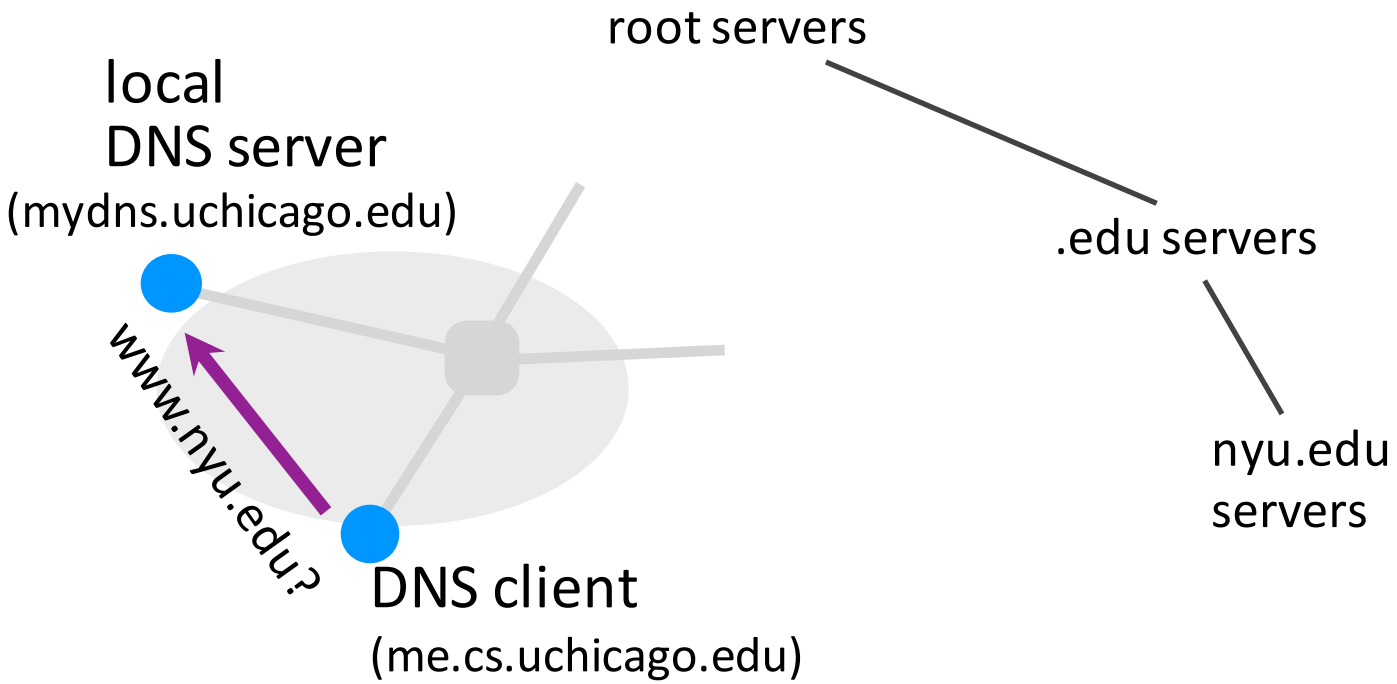
Using DNS (Client/App View)

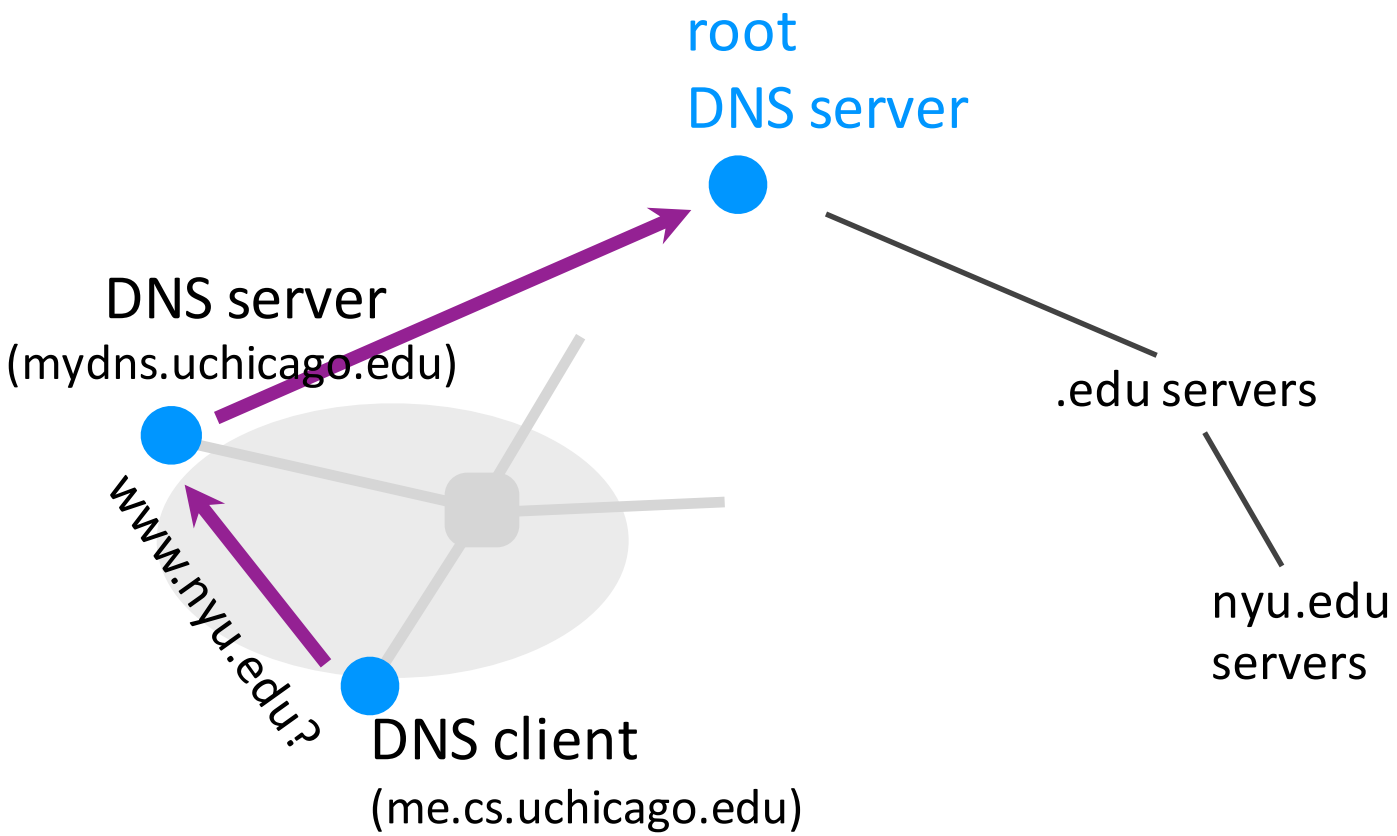
- Two components
  - Local DNS servers
  - Resolver software on hosts

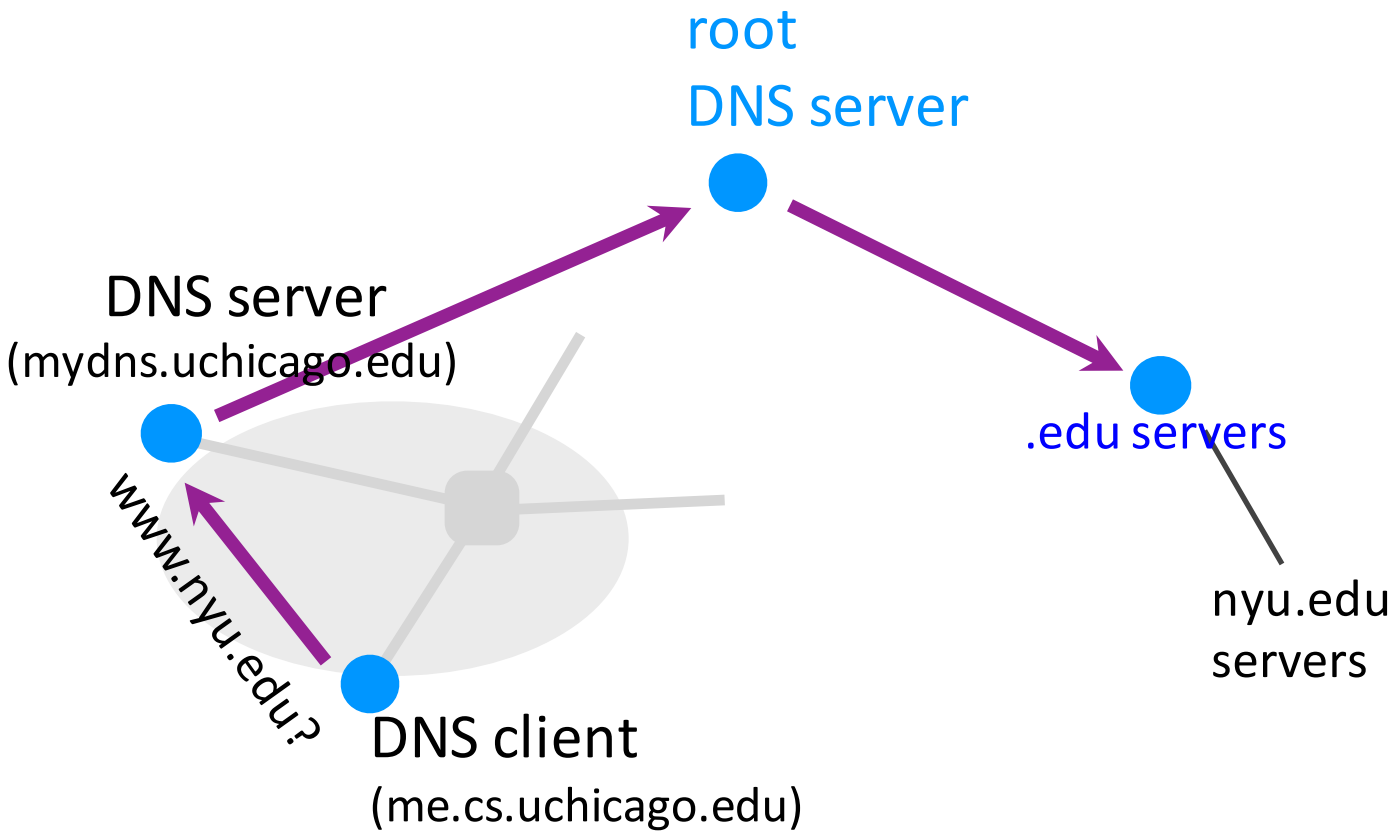
- Local DNS server ("default name server")
  - Clients configured with the default server's address or learn it via a host configuration protocol (e.g., DHCP – future lecture)

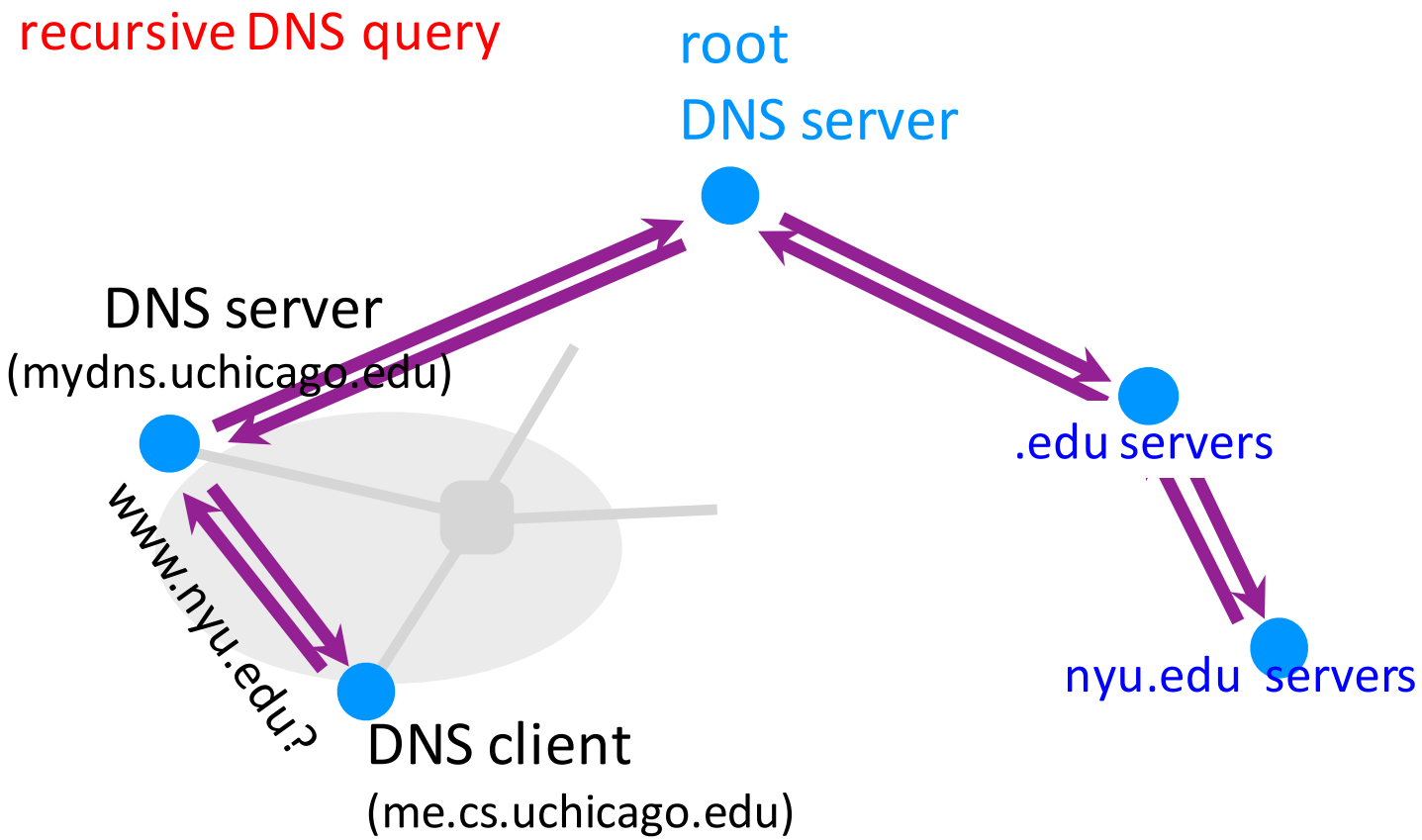
- Client application
  - Obtain DNS name (e.g., from URL)
  - Triggers DNS request to its local DNS server

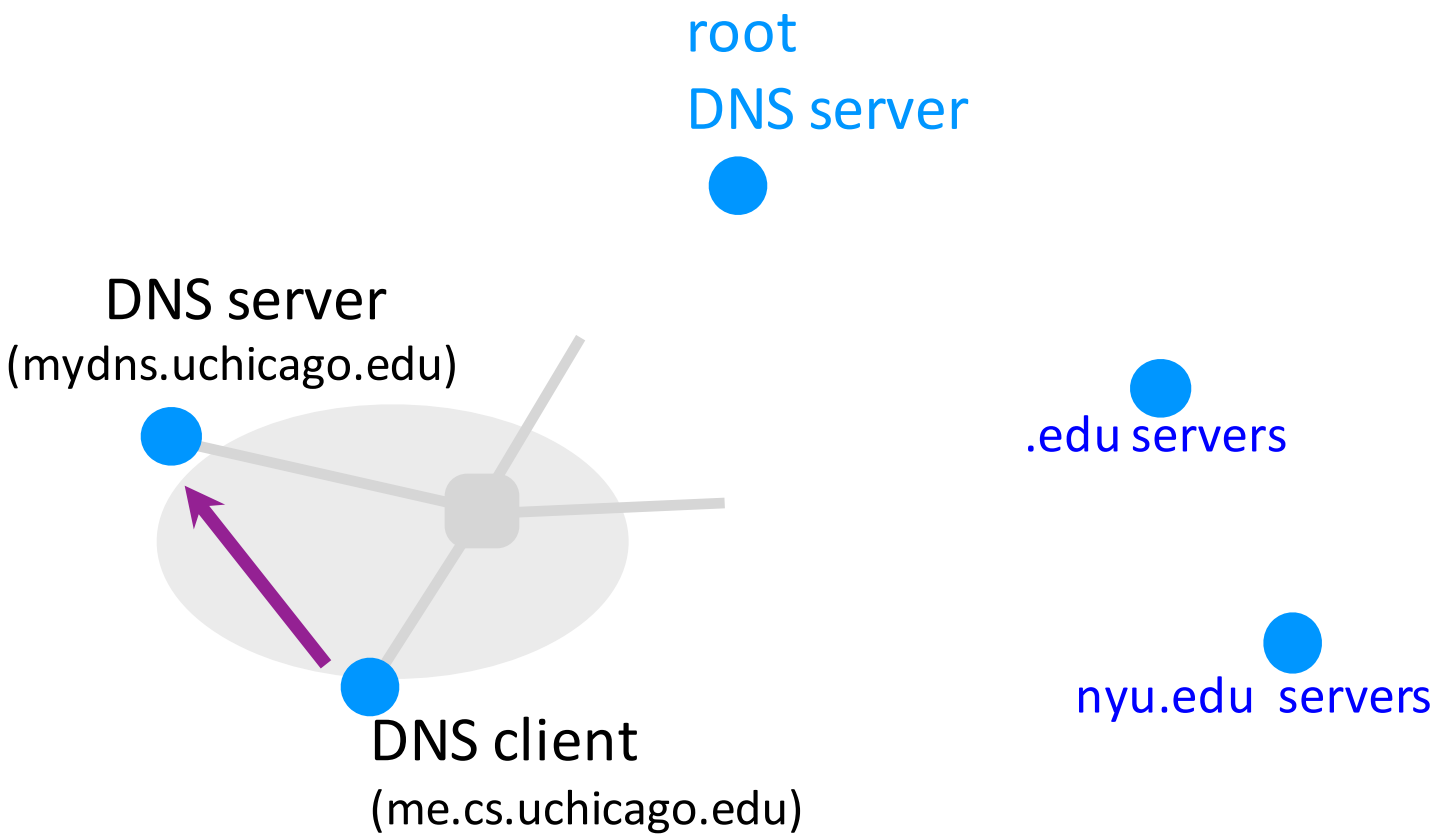

root servers
local
DNS server
(mydns.uchicago.edu)
.edu servers
nyu.edu
servers
DNS client
(me.cs.uchicago.edu)

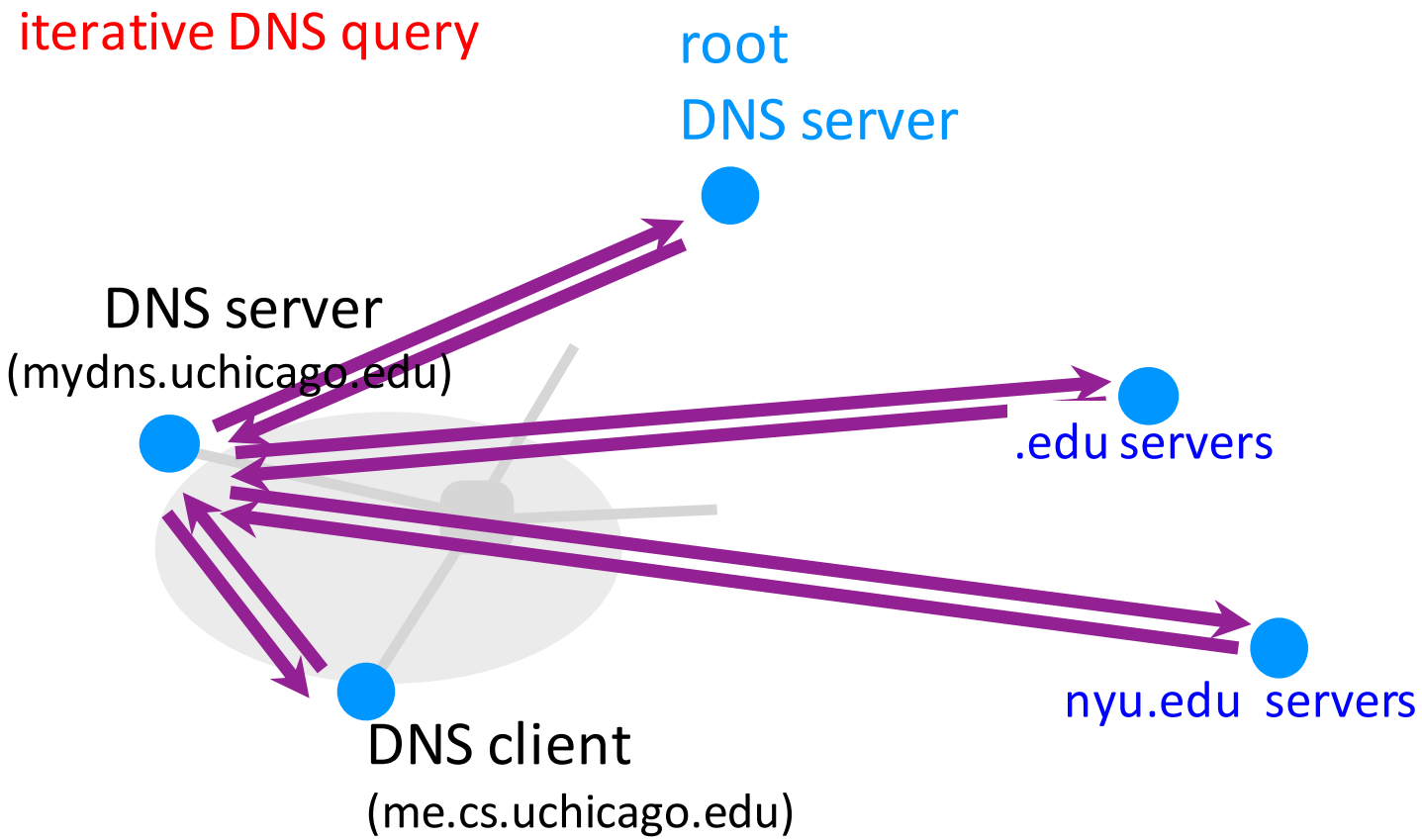

root servers
local
DNS server
(mydns.uchicago.edu)
.edu servers
www.nyu.edu?
nyu.edu
servers
DNS client
(me.cs.uchicago.edu)


root
DNS server
DNS server
(mydns.uchicago.edu)
.edu servers
www.nyu.edu?
nyu.edu
servers
DNS client
(me.cs.uchicago.edu)


root
DNS server
DNS server
(mydns.uchicago.edu)
.edu servers
www.nyu.edu?
nyu.edu
servers
DNS client
(me.cs.uchicago.edu)


recursive DNS query
root
DNS server
DNS server
(mydns.uchicago.edu)
.edu servers
www.nyu.edu?
nyu.edu servers
DNS client
(me.cs.uchicago.edu)


root
DNS server
DNS server
(mydns.uchicago.edu)
.edu servers
nyu.edu servers
DNS client
(me.cs.uchicago.edu)


iterative DNS query
root
DNS server
DNS server
(mydns.uchicago.edu)
.edu servers
nyu.edu servers
DNS client
(me.cs.uchicago.edu)

DNS Protocol

- Query and Reply messages; both with the same message format

- Client-Server interaction on UDP Port 53
  - Spec supports TCP too, but not always implemented

Goals – how are we doing?

- Scalable
  - many names
  - many updates
  - many users creating names
  - many users looking up names
- Highly available

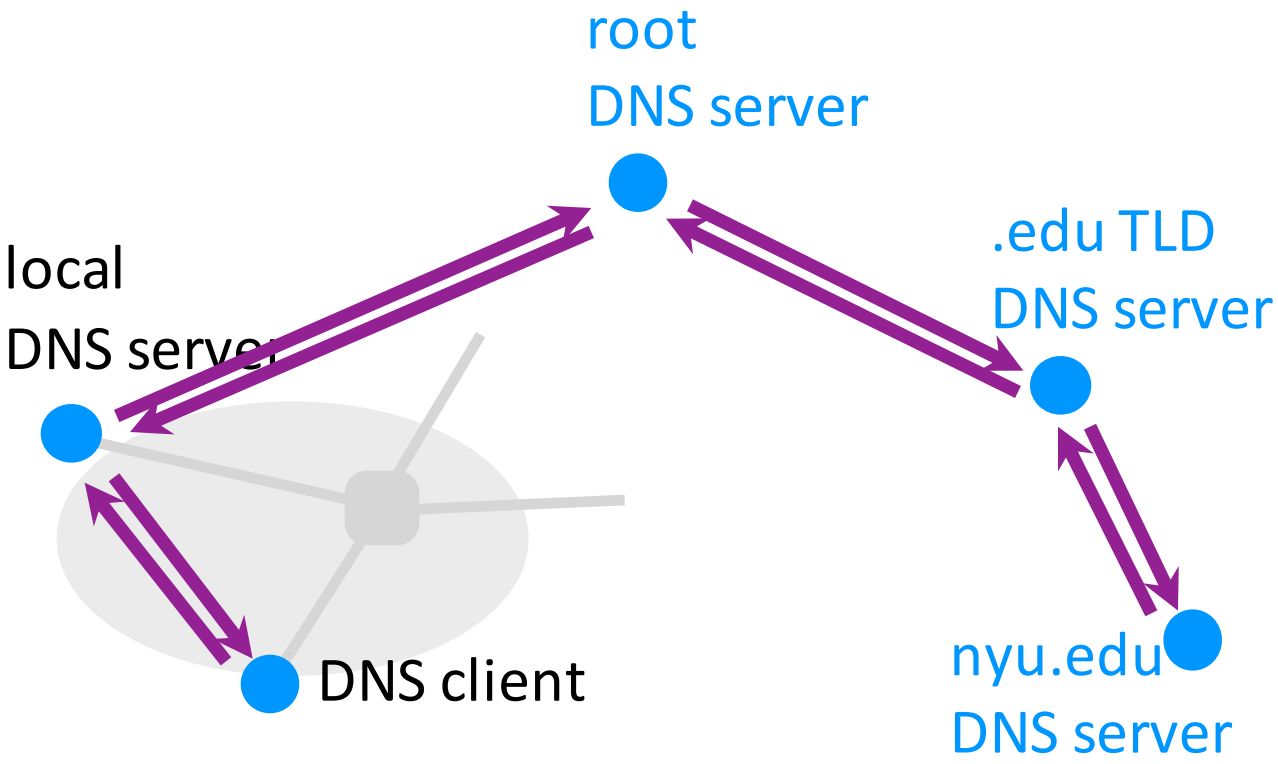
Per-domain availability

- DNS servers are replicated
  - Primary and secondary name servers required
  - Name service available if at least one replica is up
  - Queries can be load-balanced between replicas

- Try alternate servers on timeout
  - Exponential backoff when retrying same server

Goals – how are we doing?

- Scalable
  - many names
  - many updates
  - many users creating names
  - many users looking up names
- Highly available
- Correct
  - no naming conflicts (uniqueness)
  - consistency
- Lookups are fast


root
DNS server
.edu TLD
DNS server
local
DNS server
DNS client
nyu.edu
DNS server