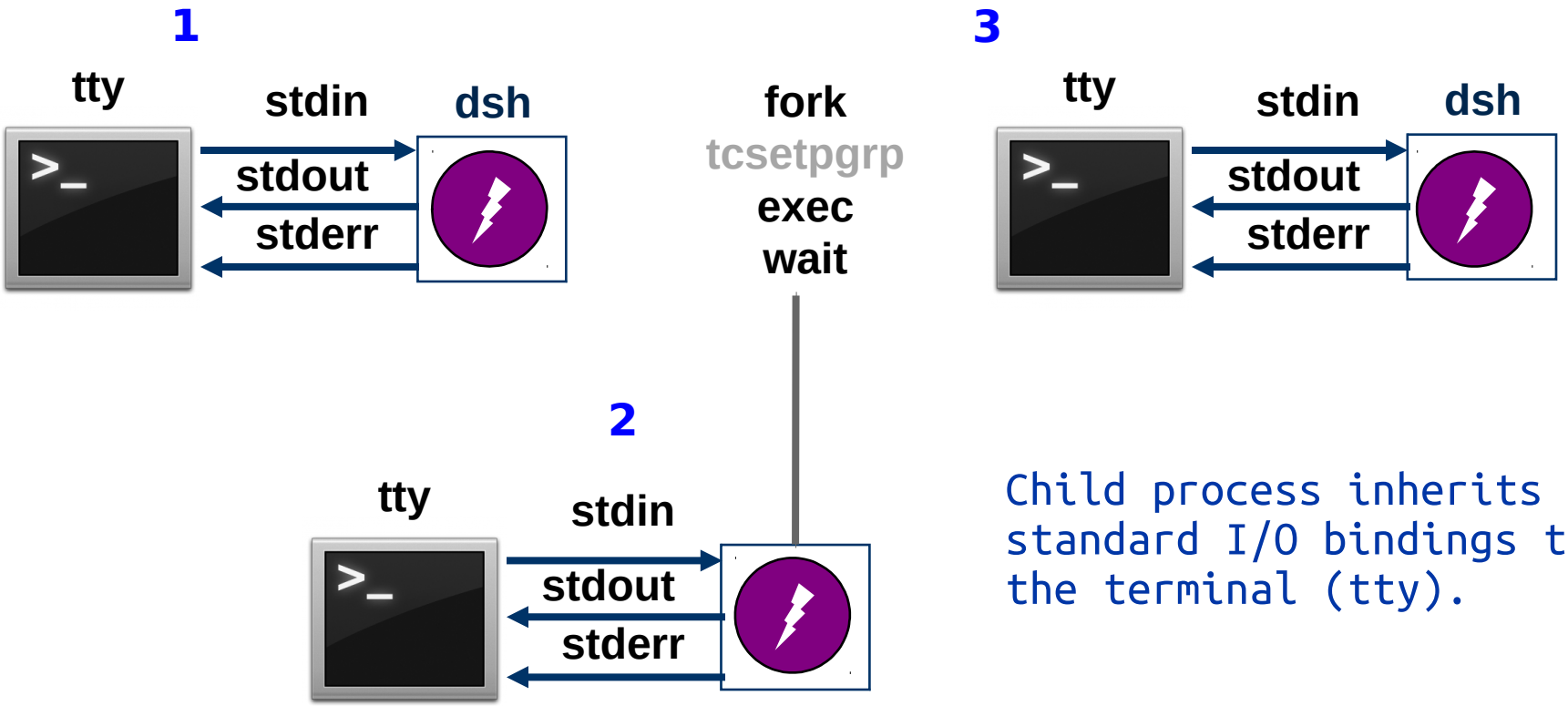
The devil shell (dsh) - Continued

COMPSCI210 Recitation

11th Feb 2013

Vamsi Thummala

Shell and child: bg, fg, jobs



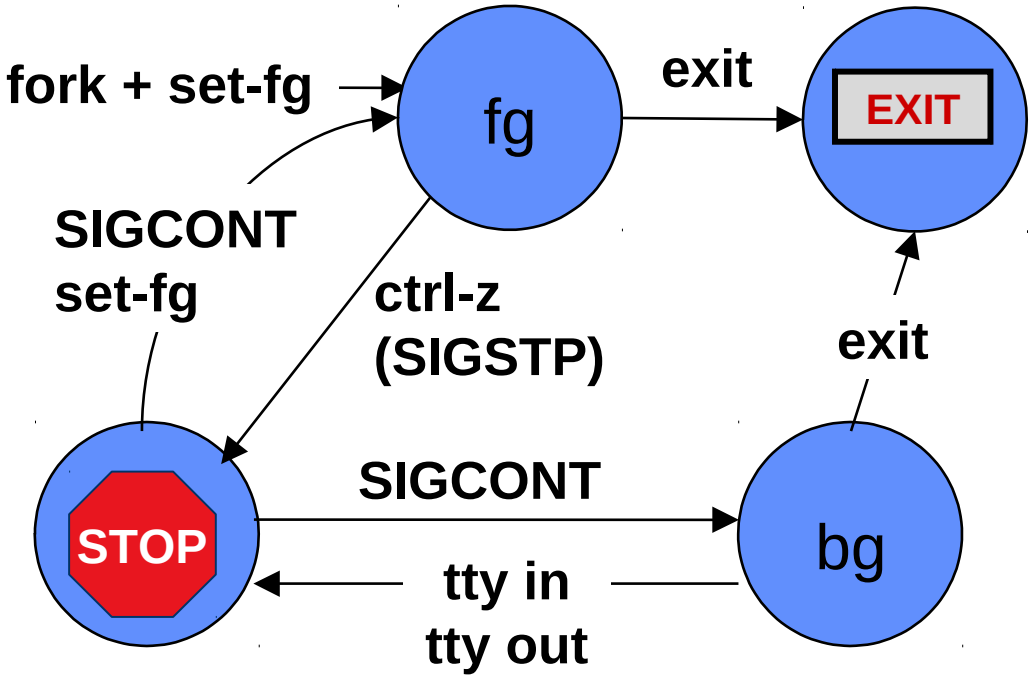
Child process inherits standard I/O bindings to the terminal (tty).

If child is to run in the **foreground**:
Child takes control of the terminal (tty) input (tcsetpgrp).
The foreground process receives all tty input until it stops or exits.
At most one process can control the tty input (others may write to tty).

Job states and transitions

User can send a STOP signal to a foreground process/job by typing **ctrl-z** on tty.

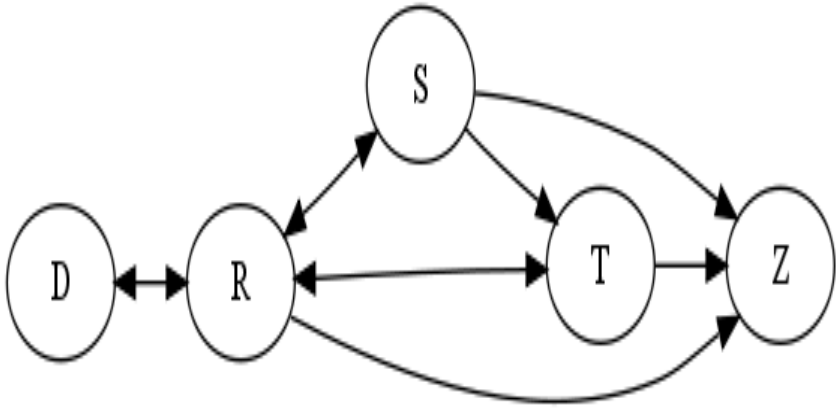
Continue a stopped process by sending it a SIGCONT signal with "**kill***" syscall.



Kernel (tty driver) sends signal to process **P** if **P** attempts to read from tty and p is in background, and (optionally) if **P** attempts to write to tty. Default action of these signals is **STOP**.

Process states

- R: Running or runnable (on run queue)
- D: Uninterruptible sleep (waiting for some event)
- S: Interruptible sleep (waiting for some event or signal)
- T: Stopped, either by a job control signal or because it is being traced by a debugger
- Z: Zombie process, terminated but not yet reaped by its parent



- s This process is a session leader.
- + This process is part of a foreground process group.

Process States: Unix shell

- ps j or ps -l or ps -jl

```
vamsi@COMPSCI210$ ps j | cat | sort -n
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
 2021  2146  2146  2146 pts/0    24837 Ss    1000   0:01 bash
 2146 24808 24808  2146 pts/0    24808 R+    1000   0:00 ps j
 2146 24809 24808  2146 pts/0    24808 S+    1000   0:00 cat
 2146 24810 24808  2146 pts/0    24808 S+    1000   0:00 sort -n

vamsi@COMPSCI210$ jobs
[1]+  Stopped                 vim
[2]   Running                 sleep 50 &

vamsi@COMPSCI210$ ps -l | cat | sort -k3 -n
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 R  1000 25023  2146  0  80   0 -  1177 -      pts/0    00:00:00 ps
0 S  1000  2146  2021  0  80   0 -  2180 wait   pts/0    00:00:01 bash
0 S  1000 25021  2146  0  80   0 -  1051 hrtime pts/0    00:00:00 sleep
0 S  1000 25024  2146  0  80   0 -  1057 pipe_w pts/0    00:00:00 cat
0 S  1000 25025  2146  0  80   0 -  2154 pipe_w pts/0    00:00:00 sort
0 T  1000 25012  2146  0  80   0 -  3163 signal pts/0    00:00:00 vim
```

Pipeline: Chaining processes

- One-way communication channel
- Symbol: |

```
int fdarray[2]; char buffer[100];
pipe(fdarray);
write(fdarray[1], “hello world”, 11);
read(fdarray[0], buffer, sizeof(buffer));
printf(“Received string: %s\n”, buffer);
```

Pipe between parent/child

```
int fdarray[2];
char buffer[100];
pipe(fdarray);
    switch (pid = fork()) {
          case -1: perror("fork failed"); exit(1);
          case 0:  write(fdarray[1], "hello world", 5);
          default: n = read(fdarray[0], buffer,  sizeof(buffer));
                   //block until data is available
    }
```

How does the pipes work in shell, i.e, "ls | wc"?

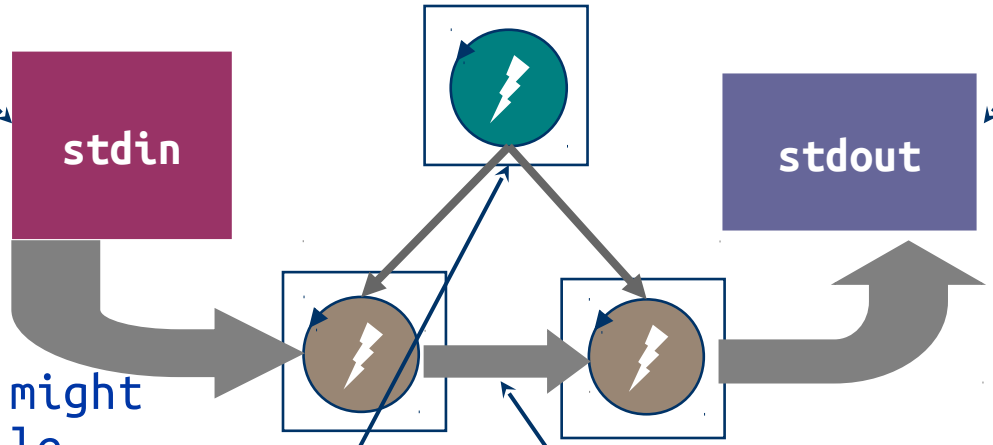
Need to duplicate the child descriptors to stdin/stdout

```
dup2(oldfd, newfd); // duplicates fd; closes and copies at one shot
```

Pipes are core to Unix programming environment

Standard unix programs read a byte stream from standard input (fd==0).

They write their output to standard output (fd==1).

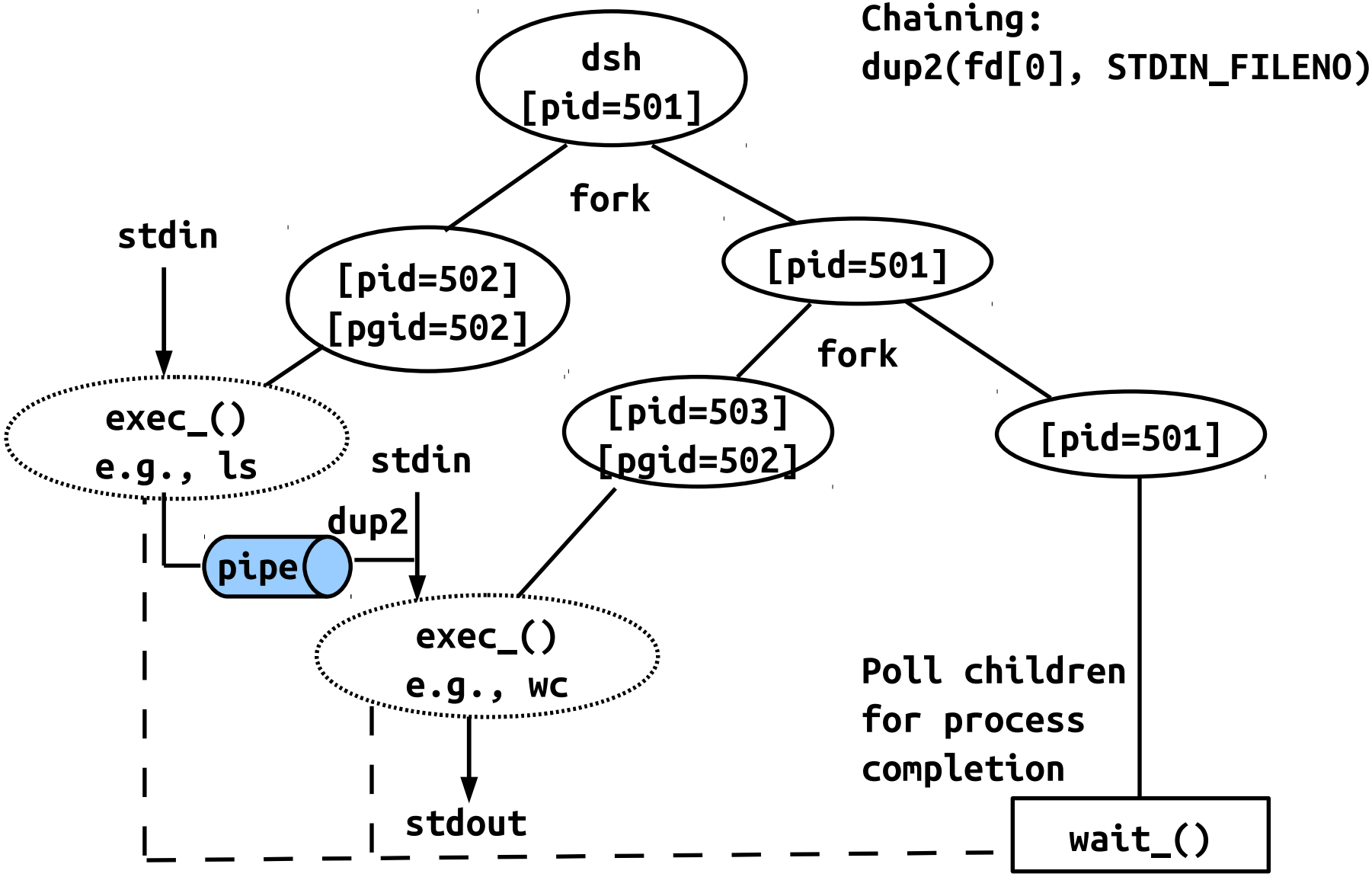


Stdin or stdout might be bound to a file, pipe, device, or network socket.

If the parent sets it up, the program doesn't even have to know.

That style makes it easy to combine simple programs using pipes or files.

Pipeline implementation



dsh additional requirements

- Auto compilation and execution of C programs
  - How to execute two processes sequentially?
- Error handling and logging
  - dup2(stderr, ...)
- Batch mode
  - $./dsh < batchFile
  - Batch mode is used for partial grading
  - It is important that you should test in batch mode before submission

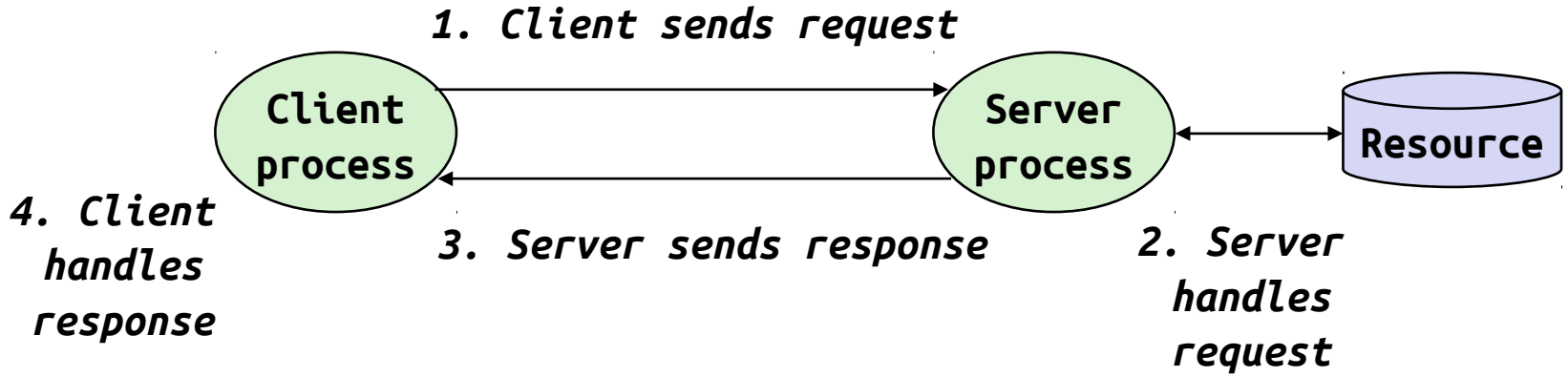
IPC: Beyond pipes

- Named pipes

```
dsh$ mkfifo namedPipe
dsh$ cat < namedPipe > out &
dsh$ jobs
[1]+  Running cat < namedPipe > out &
dsh$ echo "Communicating to other process via name  pipe" >
namedPipe
dsh$ cat out
Communicating to other process via name pipe
```

- Sockets
  - Named bidirectional pipe
  - To the kernel, an endpoint of communication
  - Can be used to communicate across a network
  - Underlying basis for all Internet applications

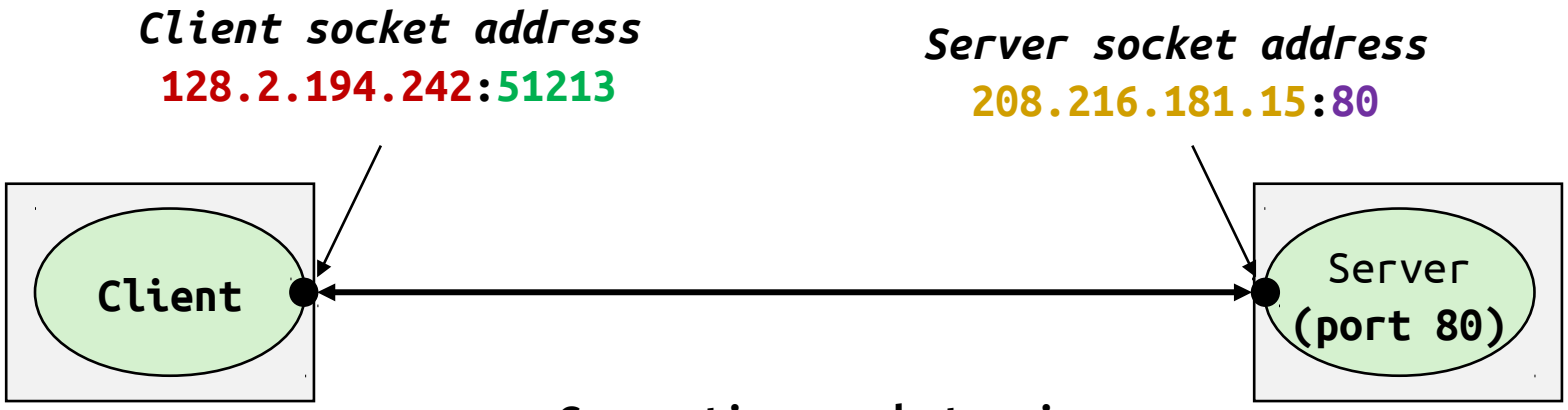
Client Server communication



- Create a socket with the **socket()** system call
- Connect the socket to the address of the server using the **connect()** system call
- Send and receive data using the **read()** and **write()** system calls

- Create a socket with the **socket()** system call
- Bind the socket to an address using the **bind()** system call. For a server socket on the Internet, an address consists of a port number on the host machine.
- Listen for connections with the **listen()** system call
- Accept a connection with the **accept()** system call. This call typically blocks until a client connects with the server.
- Send and receive data

Client Server communication: A detailed example



Client host address
128.2.194.242

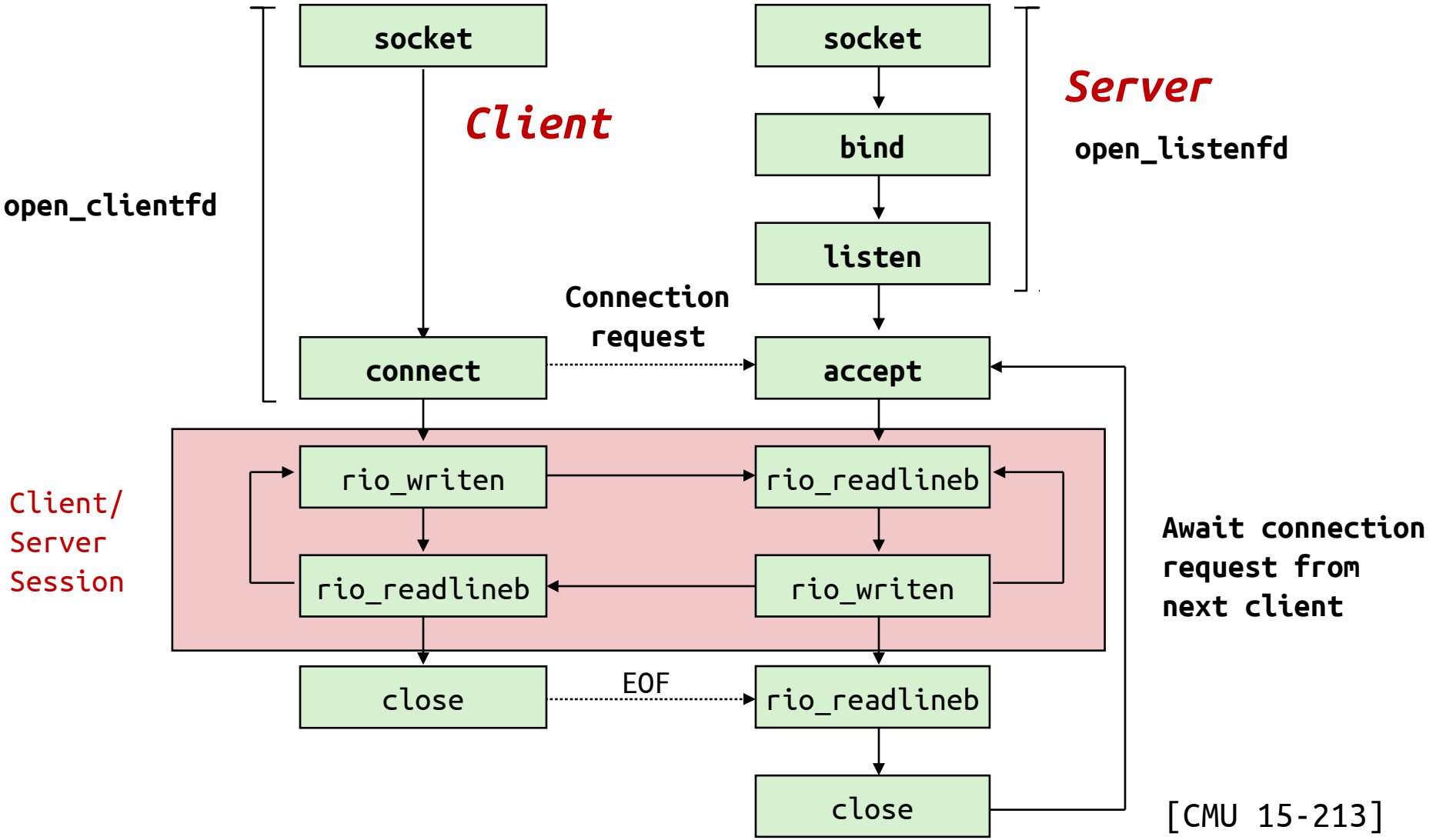
51213 is an ephemeral port allocated by the kernel

Server host address
208.216.181.15

80 is a well-known port associated with Web servers

[Demo]

Socket interface



java.net

- Low level API
  - Addresses
  - Sockets
  - Interfaces
- High level API
  - URIs
  - URLs
  - Connections

Concept checkers for midterm

- Basic: address space, process, thread, event
- Kernel: syscall, context switch, and exceptions (trap, fault, interrupt)
- Protection: reference monitor, access control list, capability
- Execution: process vs. thread
- Concurrency: event-driven vs. threading
- Fragmentation: internal vs. external
- IPC: pipes vs. sockets

Fall 2012 midterm paper

- With solutions
  - http://www.cs.duke.edu/courses/compsci210/fall12/midterm-210-12f-sol.pdf