Processes and Pipes

Chris Brown

pluralsight

In This Module …

Processes

What are they?

fork / exec / wait lifecycle

Pipes

— Anonymous

— Named

Demonstrations

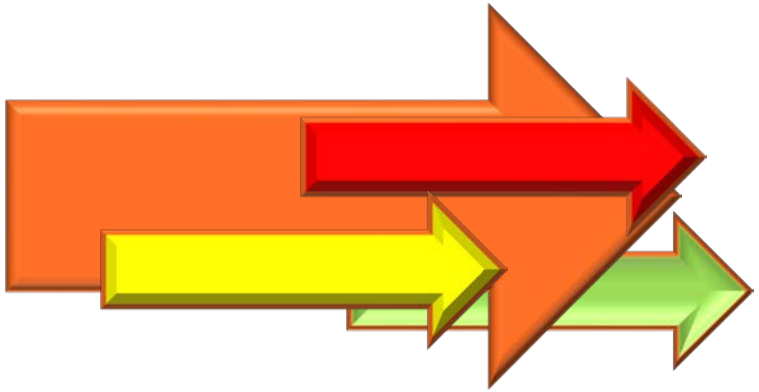
A simple shell

pluralsight

Programs and Processes

A program is a list of instructions to be executed

A process is an instance of a program in execution



pluralsight

Where Are the Processes?

pluralsight

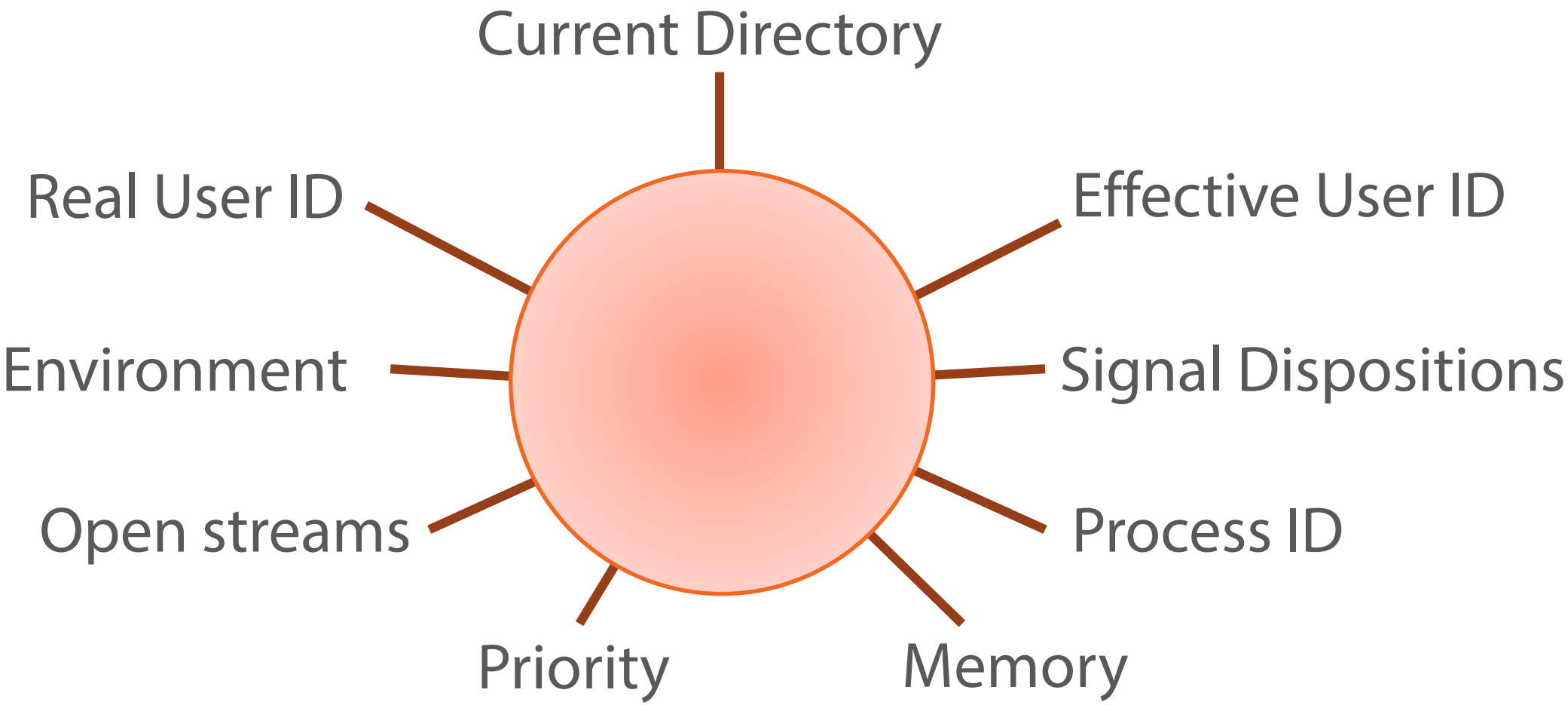
"A process is an instance of a program in execution"

pluralsight

"A process holds the resources a program needs to execute"

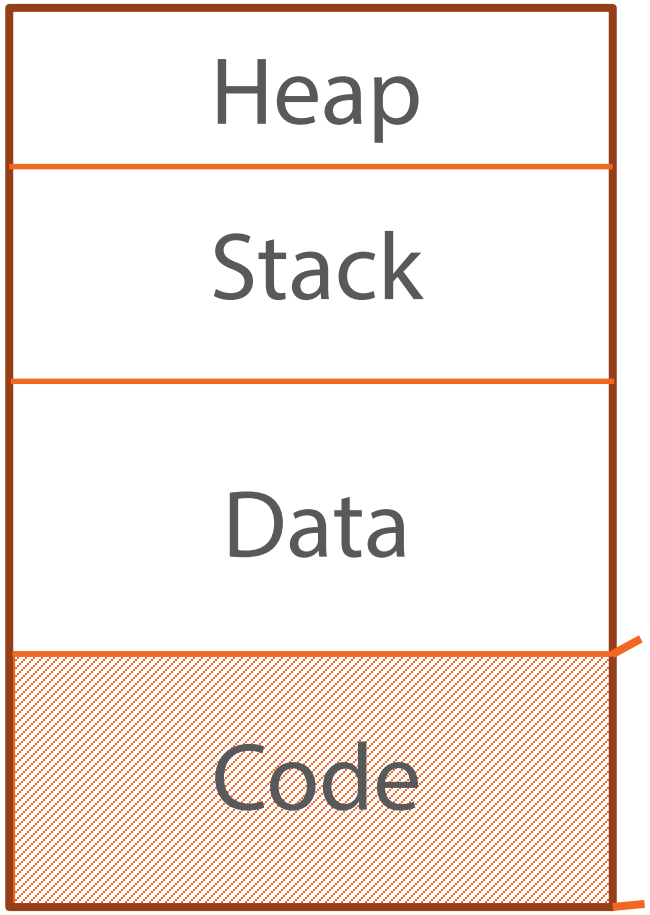
pluralsight

The Context of a Process



pluralsight

Process Memory



Also called the *text segment*

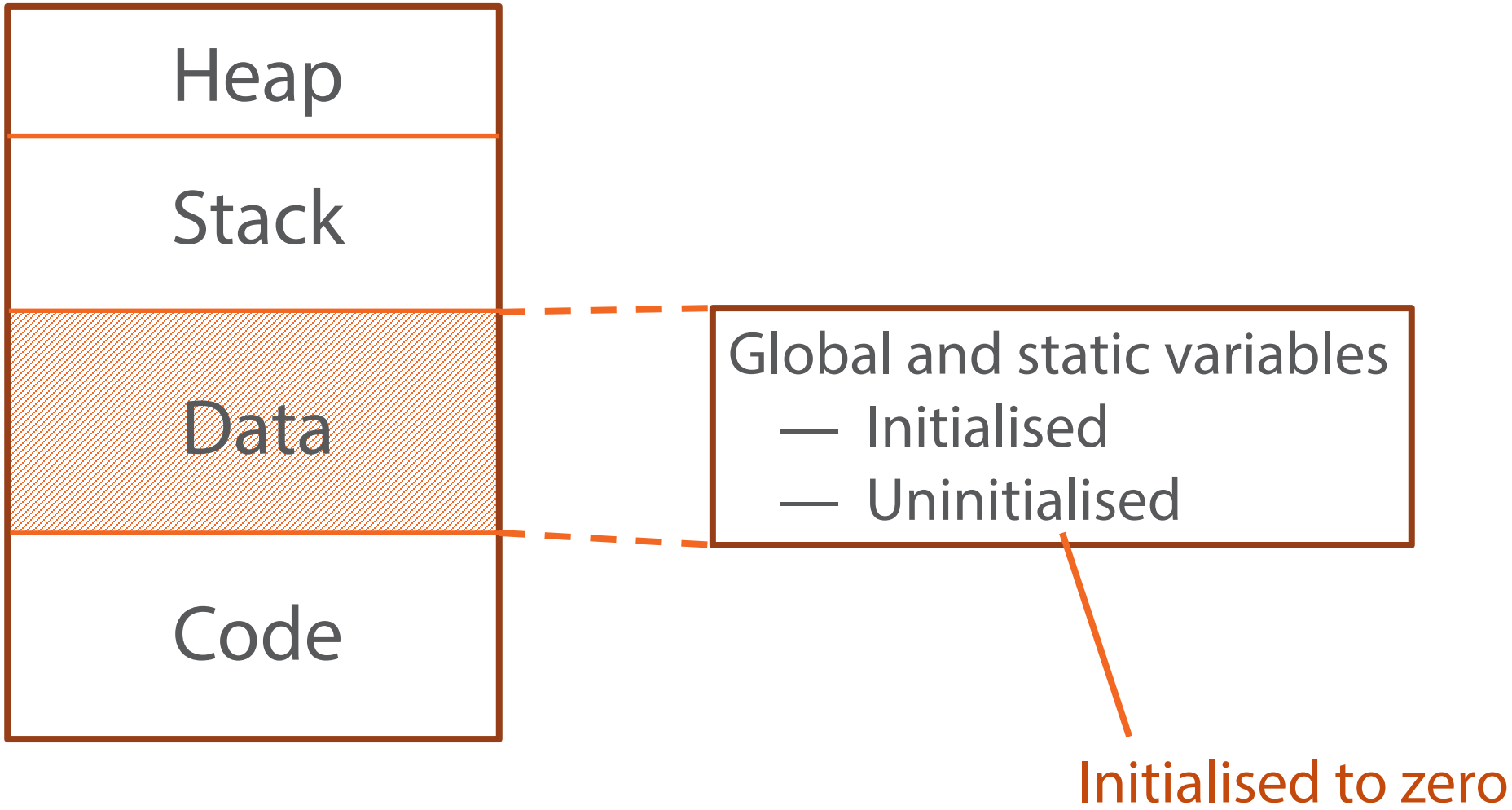
— Holds the program's code

— Read-only

— May be shared with other processes

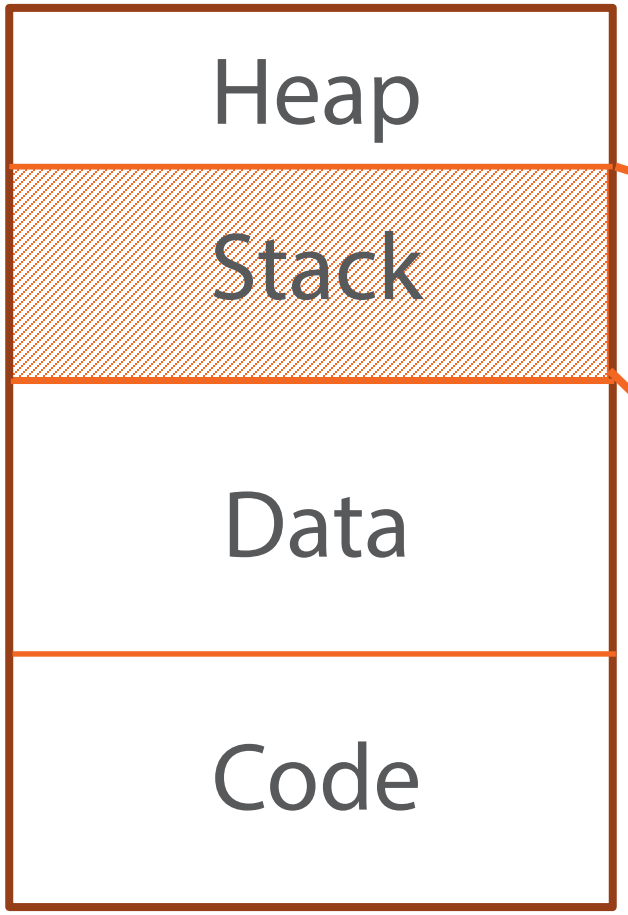
pluralsight

Process Memory



pluralsight

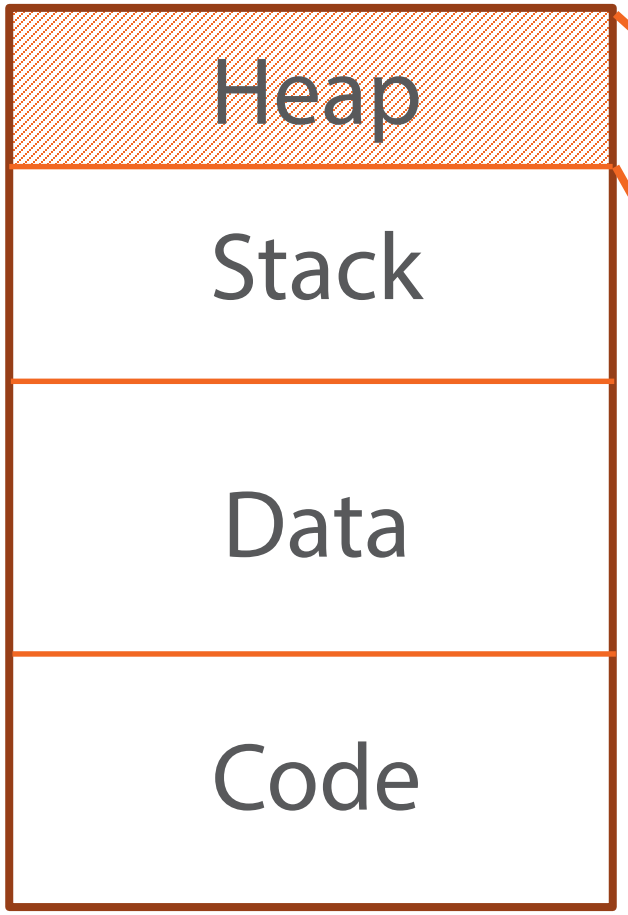
Process Memory



Holds a function's local variables
A function's "stack frame" is allocated when the function is called, and de-allocated when it returns

pluralsight

Process Memory

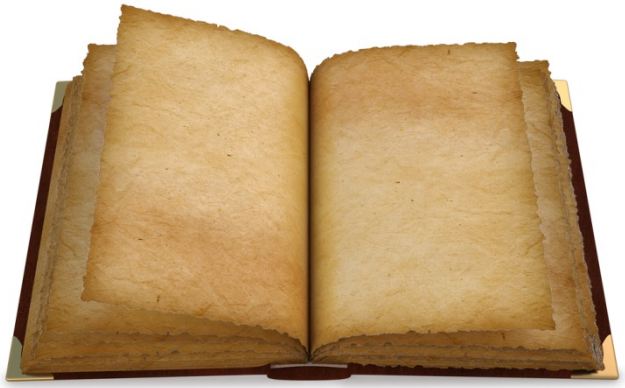


Area used to dynamically allocate storage for data at run time

- `malloc()` and `free()`
- `sbrk()`

pluralsight

Analogy



The script of a play is like a program

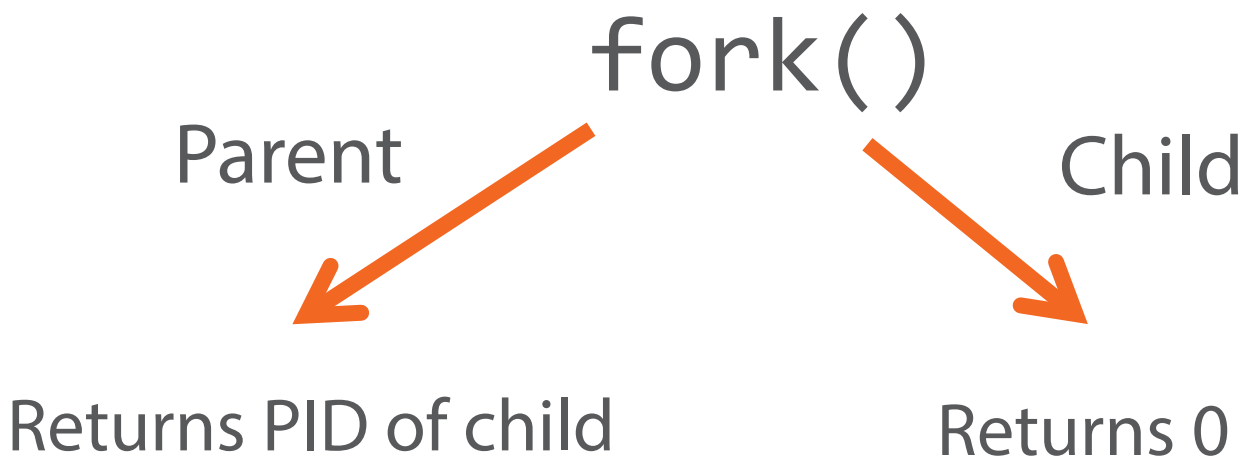
— A list of instructions of what to say and do

An actor is like a process

— The agent responsible for obeying the instructions

pluralsight

Creating a Process



The child inherits copies of most things from its parent, except:

- it shares a copy of the code
- it gets a new PID

pluralsight

fork() Example

```
#include <stdio.h>

void main()
{
    if (fork())
      printf("I am the parent\n");
    else
      printf("I am the child\n");
}
```

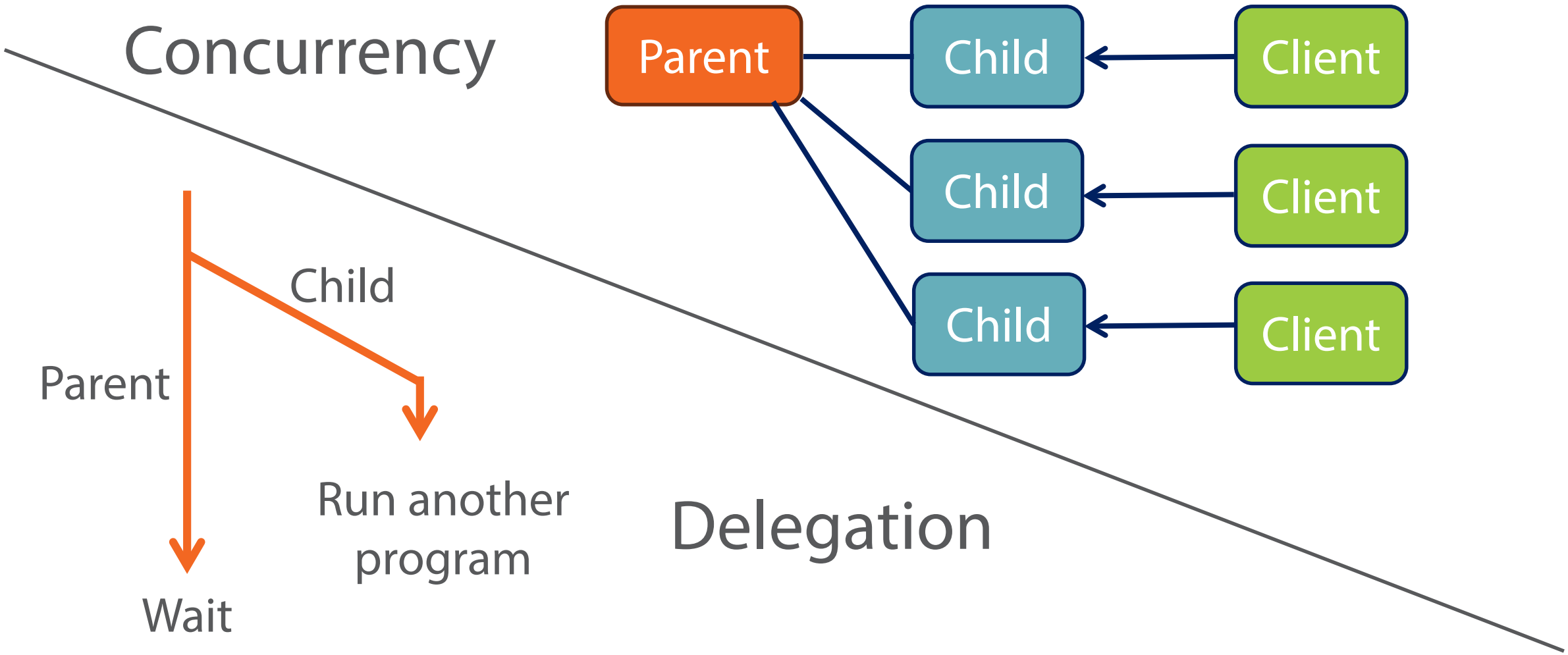
```
Output:

I am the parent
I am the child
```

```
Or …

I am the child
I am the parent
```

```
Or …

I am thI am thee pchild
arent
```

pluralsight

Why fork() ?



pluralsight

Program Execution Using exec()

`exec()` causes the current execution image of a process to be replaced by the image of a new program

```
execl("/bin/ls", "ls", "-a", 0)
```

If the call succeeds it never returns

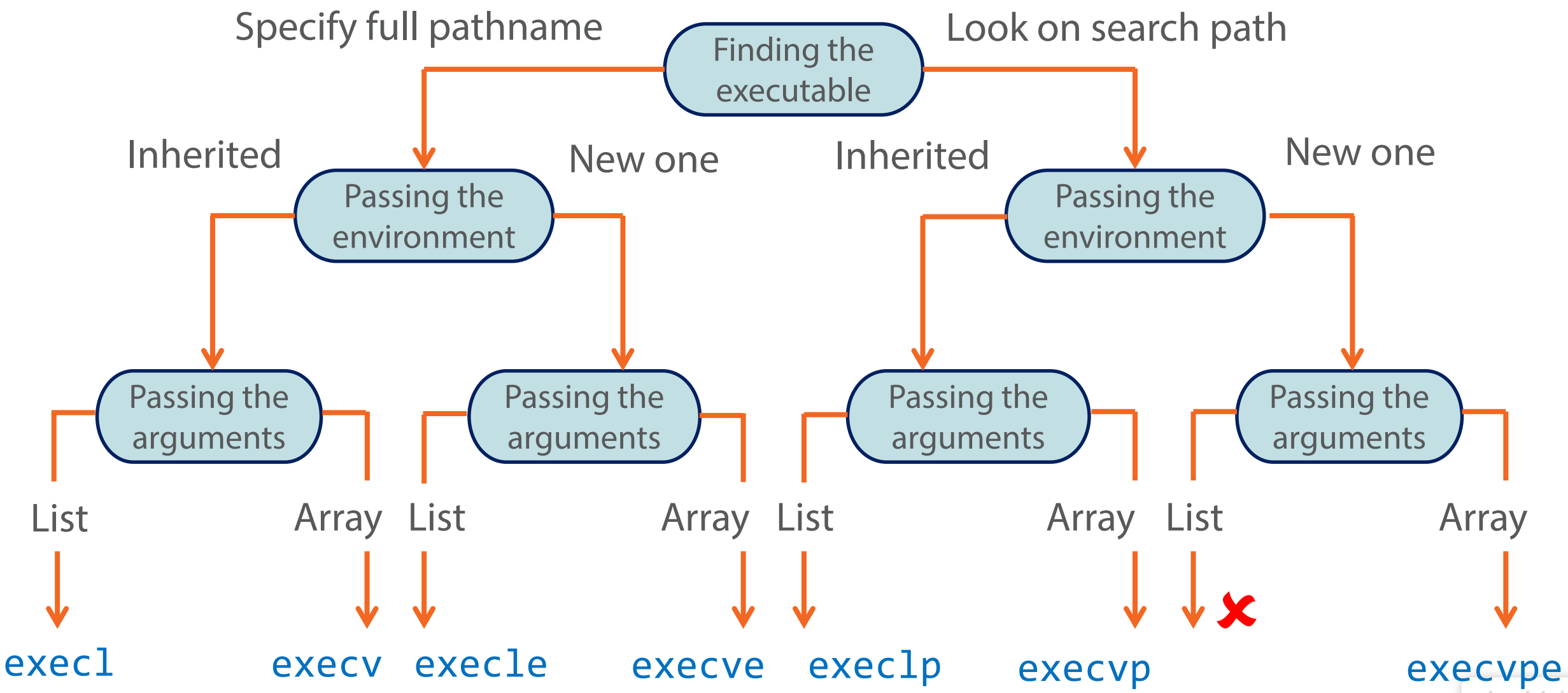
The new executable image

The command-line arguments

There are seven variations of exec!

pluralsight

Choosing the Right Version of exec( )



pluralsight

exec() Examples

```
char *argv[] = {"ls", "-a", 0};
char *envp[] = {"EDITOR=vi", "TZ=:EST", 0};

execl("/bin/ls", "ls", "-a", 0};

execle("/bin/ls", "ls", "-a", 0, envp);

execv("/bin/ls", argv);

execve("/bin/ls", argv, envp);

execlp("ls", "ls", "-a", 0);

execvp("ls", argv);

execvpe("ls", argv, envp);
```

pluralsight

Process Termination

```
exit(n);
```

Exit status
Passed back to parent
0 means success
1-255 means failure

```
int status;
wait(&status);
```

Call waits until a child
process terminates.
Returns PID of the child

The child's exit status is returned here.
Pass 0 (NULL) if not interested

pluralsight

Exit Status – Normal Termination

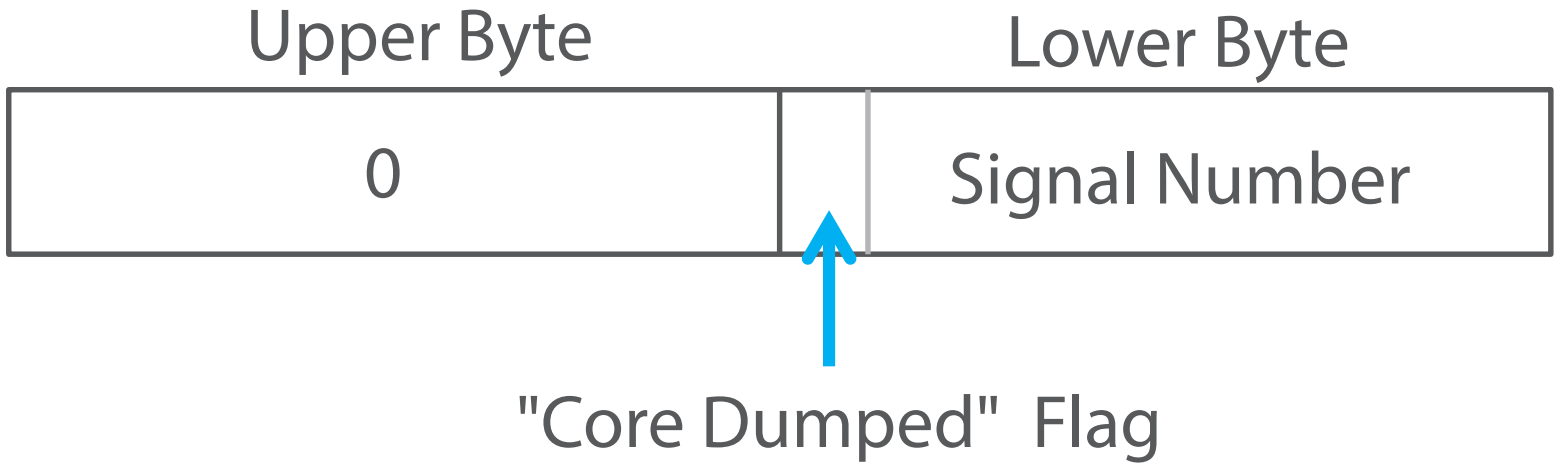
Upper Byte	Lower Byte
Exit status (0-255)	0

Conventionally: zero = success, nonzero = "failure"

MACRO	Meaning
`WIFEXITED(status)`	True if child exited normally
`WEXITSTATUS(status)`	The exit status

pluralsight

Exit Status – Killed by Signal

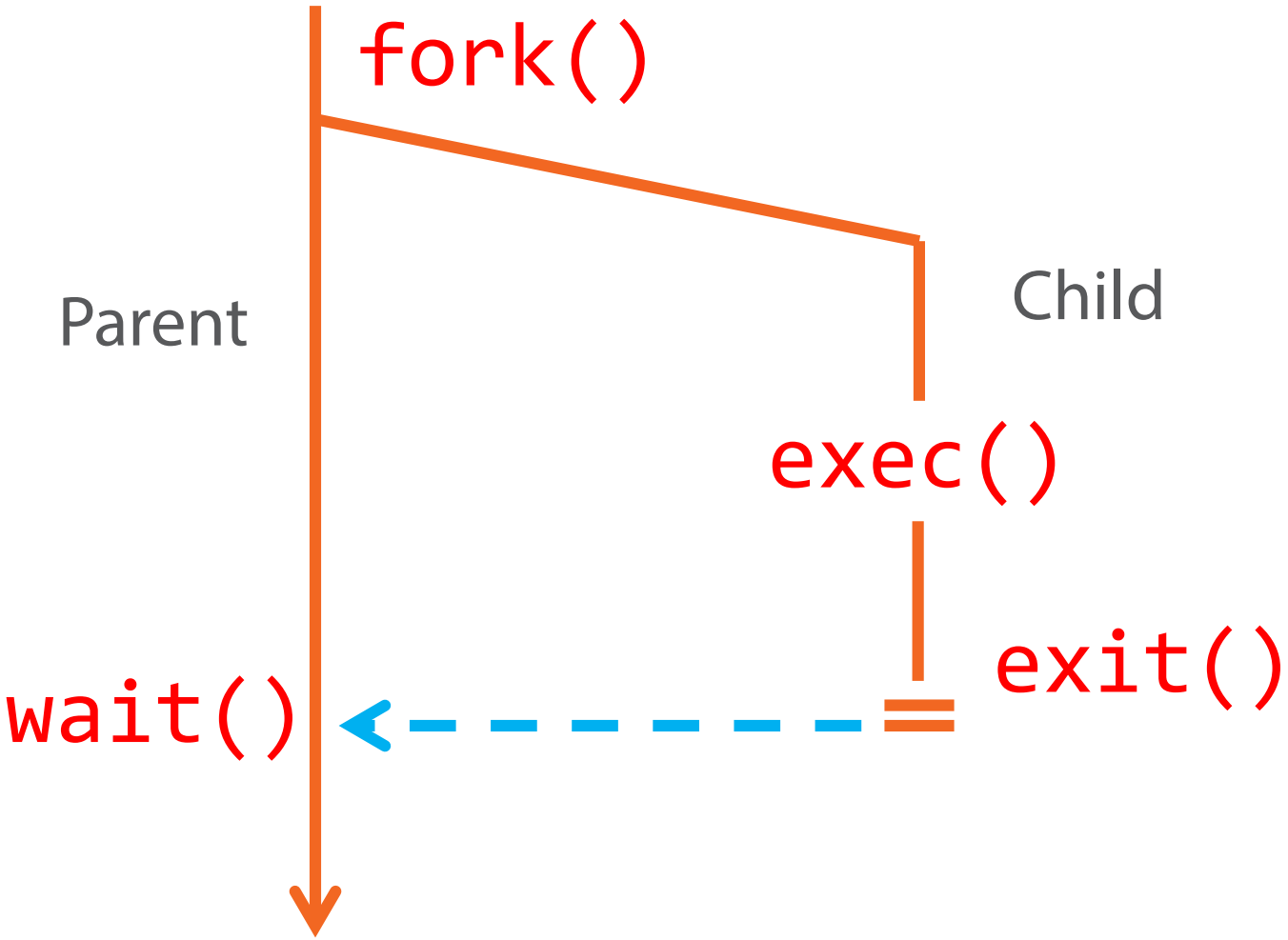


MACRO	Meaning
`WIFSIGNALED(status)`	True if child terminated by signal
`WTERMSIG(status)`	The signal number

pluralsight

Process Life Cycle

```
main()
{   if (fork() == 0) {
        execlp("prog", …);
    }
    else {
        wait(&status);
    }
}
```



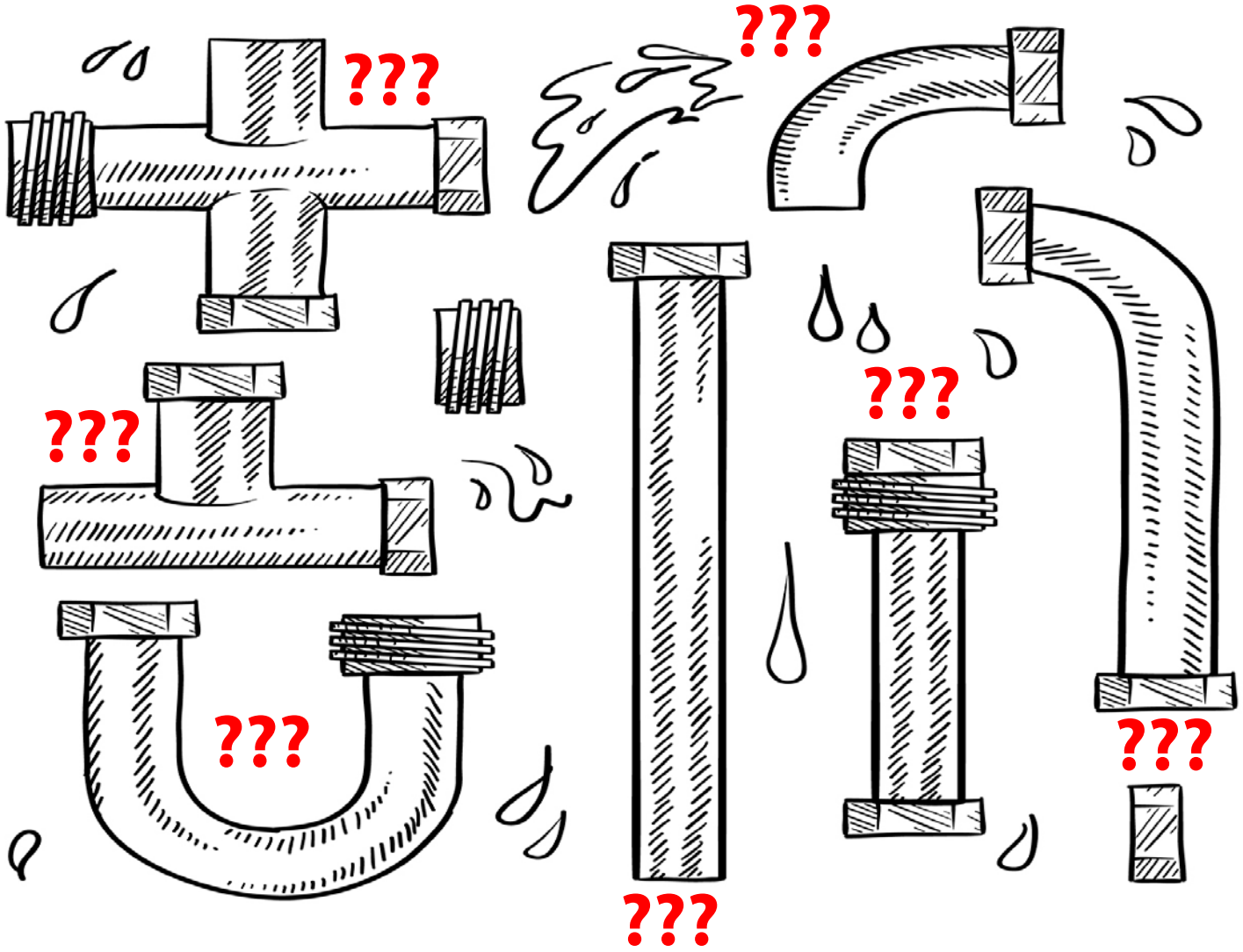
pluralsight

A Tiny Shell

```
main()
{
    char line[100];

    /* Main command loop */
    while (printf("> "), gets(line) != NULL) {
        if (fork() == 0) { /* Child */
            execlp(line, line, (char *)0);
            printf("%s: not found\n", line);
            exit(1);
        }
        else  wait(0); /* Parent */
    }
}
```

pluralsight

Anonymous Pipes



pluralsight

Anonymous Pipes

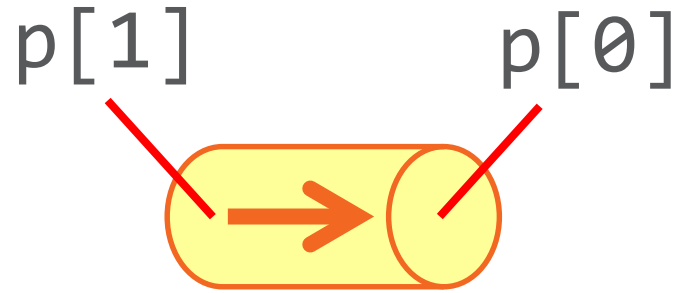
- Widely used inter-process communication mechanism
  - At the heart of the UNIX "tool building" philosophy
- Easy to create at the command line:
  - `du -s /home/* | sort -n`
- Unidirectional
- Provide buffering and loose synchronisation between a producer (upstream) and a consumer (downstream)
  - Producer blocks when writing if pipe is full
  - Consumer blocks when reading if pipe is empty

pluralsight

Creating a Pipe

```
int p[2];
pipe(p);
```

Returns
0 on success
-1 on error



pluralsight

Copying File Descriptors

`dup(fd)`

Copies fd onto the lowest available descriptor

`dup2(fd1, fd2)`

Copies fd1 onto fd2 (fd2 is closed first)

Returns the new descriptor or -1 on error

pluralsight

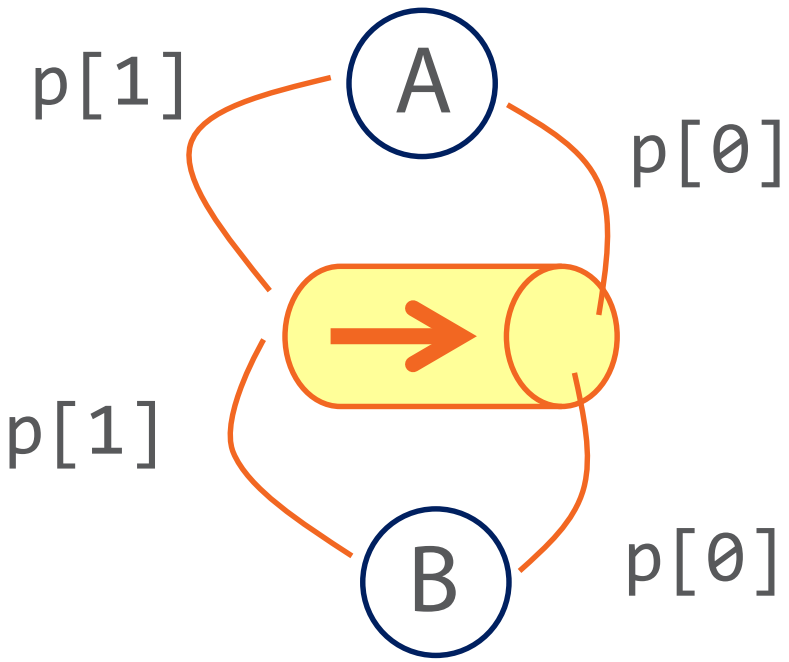
Redirecting Standard Input

```
int fd;
fd = open("foo", ...);
close(0);
dup(fd);
```

```
int fd;
fd = open("foo", ...);
dup2(fd, 0);
```

pluralsight

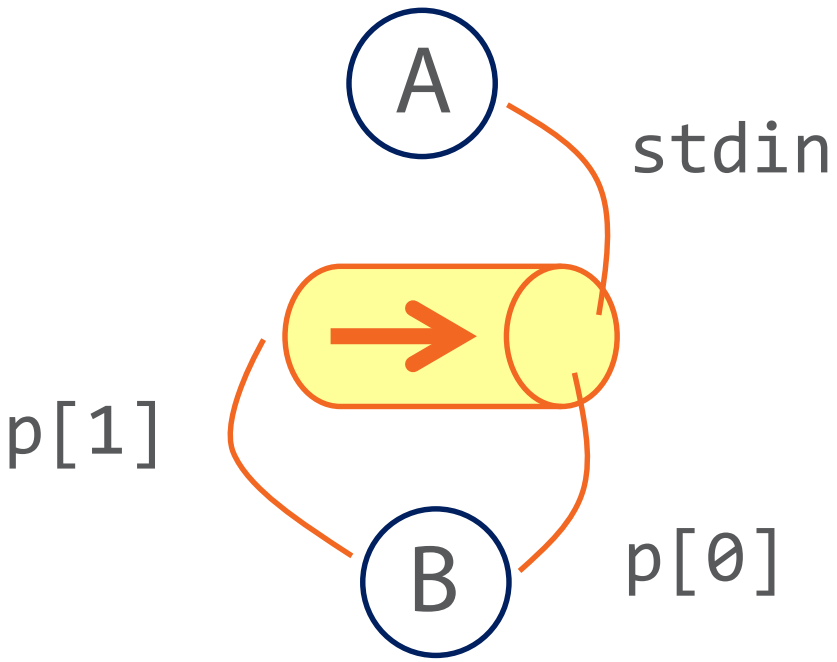
Anonymous Pipes




```
int p[2];
pipe(p);
fork();
```

pluralsight

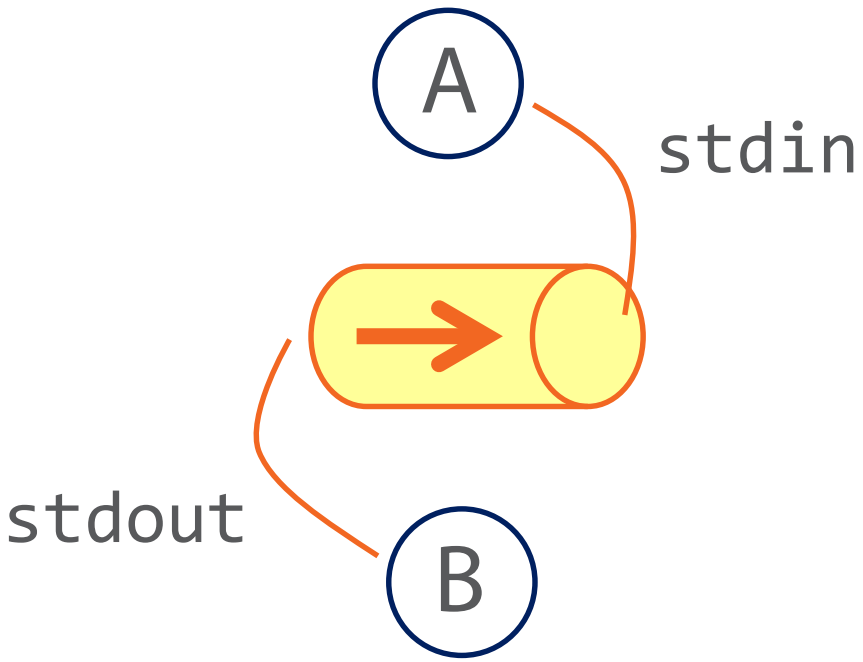
Anonymous Pipes




```
int p[2];
pipe(p);
fork();
/* Parent */
dup2(p[0], 0);
close(p[1]);
exec(.. downstream ..);
```

pluralsight

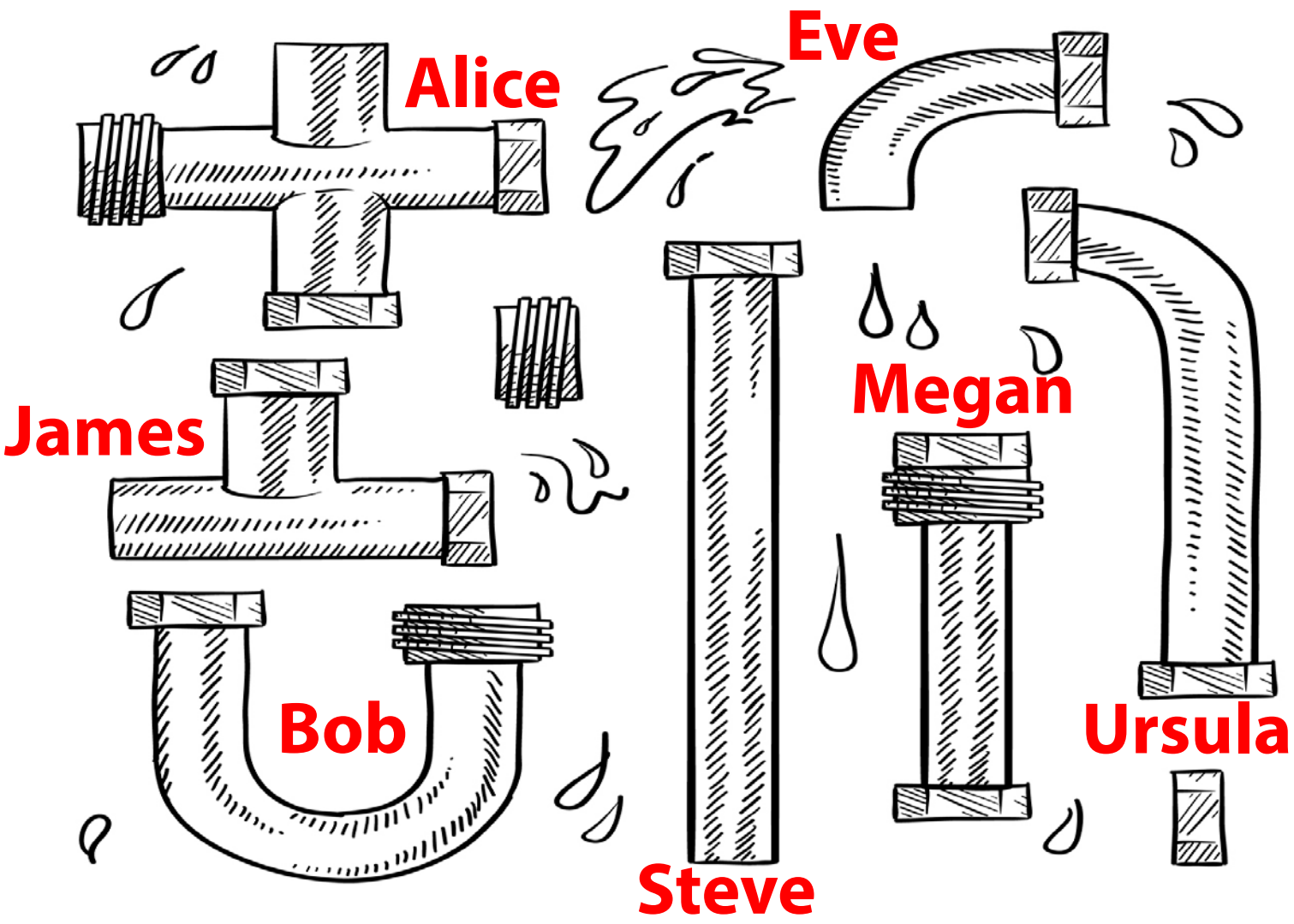
Anonymous Pipes




```
int p[2];
pipe(p);
fork();
/* Parent */
dup2(p[0], 0);
close(p[1]);
exec(.. downstream ..);
/* Child */
dup2[p[1], 1);
close(p[0]);
exec(.. upstream ..);
```

pluralsight

Named Pipes



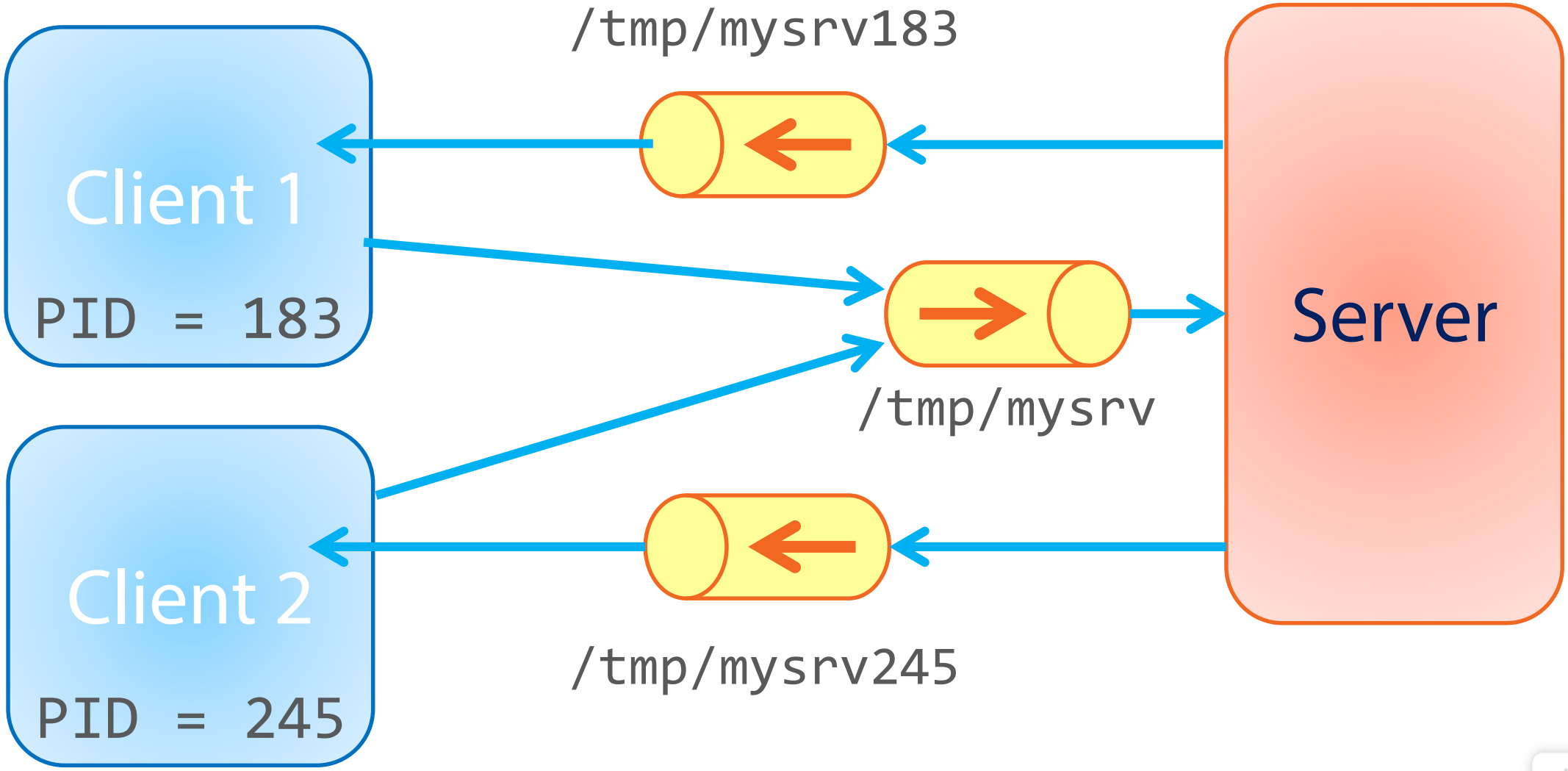
pluralsight

Creating Named Pipes

- Named pipes have an entry in the file system
  - Have ownership and permissions like any file
- On the command line, created with `mkfifo /tmp/mypipe`
- In a program, use `mkfifo("/tmp/mypipe", 0644);`
- Unidirectional
- Loose synchronisation
  - Opening either end blocks until the other end is opened
- Uses normal `read()` and `write()` calls

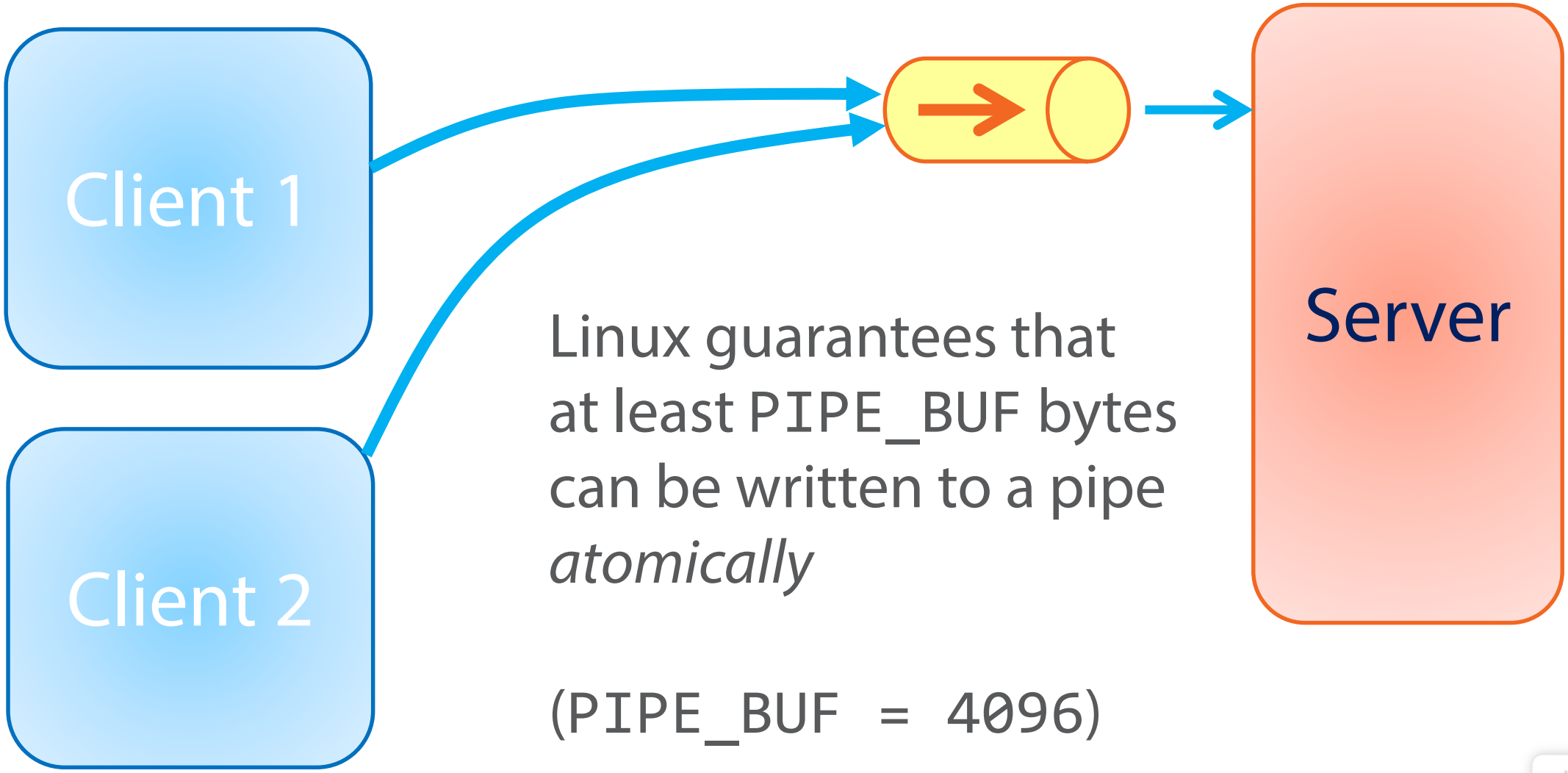
pluralsight

Named Pipes for Client/Server



pluralsight

Multiple Writers to a Pipe

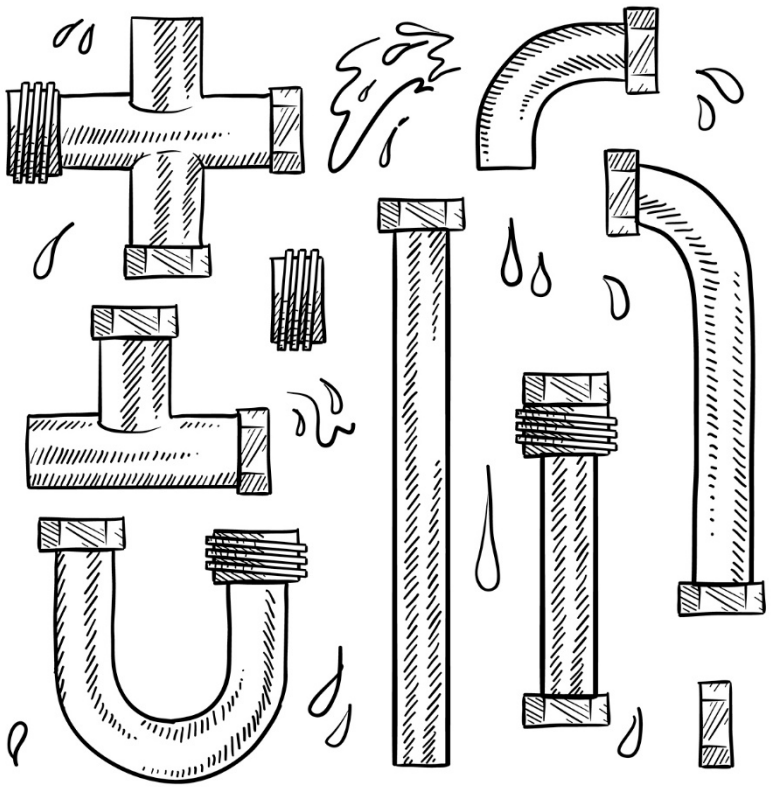


Linux guarantees that at least `PIPE_BUF` bytes can be written to a pipe *atomically*

(`PIPE_BUF = 4096`)

pluralsight

Module Summary



Processes

`fork()`, `exec()` and `wait()`

Process life cycle

Pipes

— Anonymous

— Named

pluralsight

Coming up in the Next Module

- Access Control
- User identity
- Process identity
- File permissions and ownership

pluralsight