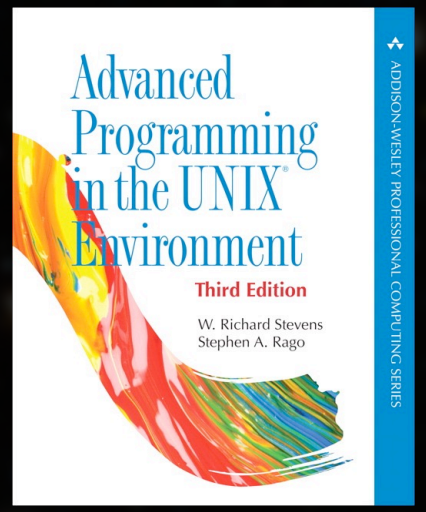
The Prestige (2006), Christopher Nolan
Robert Angier(Hugh Jackman) and Alfred Borden (Christian Bale)



Chapter 15: Inter-Process Communication

Continuous Communication → Conversation

In previous examples, there exist a single communication.

Example III: Solution A

```
void main(void){
        while(1){
                Example II's Code with some minor change
        }
}
```

```
hfani@charlie:~$ vi parent_child_conv_a.c
int main(int argc, char *argv[]){
    while(1){
        int fd = open("child_results_conv.txt", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
        printf("parent opens the file for R/W with fd: %d\n", fd);
        //int X = atoi(argv[1]);
        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!\n");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("I am the child, pid=%d and given the fd %d\n", getpid(), fd);

                        int Y[1] = {-1};
                        int X;
                        printf("enter a positive number:\n");
                        scanf("%d", &X);
                        if(X == -1){
                                printf("child: the user wants to end the program.\n");
                                write(fd, Y, sizeof(Y));
                                exit(0);
                        }
                        Y[0] = X * X;
                        int byte_write = write(fd, Y, sizeof(Y));
                        printf("child write %d bytes.\n", byte_write);

                        printf("I brought the number to the power 2 and wrote the result: %d.\n", Y[0]);
                        exit(0);
                }
        }
        int child_exit;
        wait(&child_exit);//wait for the child to X^2

        int Y[1];
        lseek(fd, 0, SEEK_SET);
        int byte_read = read(fd, Y, sizeof(Y));
        printf("parent read %d bytes\n", byte_read);
        close(fd);

        if(Y[0] == -1){
                printf("child exits on user -1. I exit too.\n");
                exit(0);
        }

        int result = Y[0] + 5;
        printf("here is the result: %d\n", result);
    }
    //exit(0);
```

1) Child: Ask the user for a positive number
2) Child: If it's -1, write it down to the file and exit
3) Child: Otherwise, do the task

1) Parent: Read the value written by the child
2) Parent: If it's -1, exit
3) Parent: Otherwise, do the task

Example III: ~~Solution A~~

Very Bad Solution, Indeed Wrong!

Why?

```
void main(void){
        while(1){
                Example II's Code with some minor change
        }
}
```

```
hfani@charlie:~$ ./parent_child_conv
parent opens the file for R/W with fd: 3
I am the parent, pid=739728
I am the child, pid=739729 and given the fd 3
enter a positive number:
2
child write 4 bytes.
I brought the number to the power 2 and wrote the result: 4.
parent read 4 bytes
here is the result: 9
parent opens the file for R/W with fd: 3
I am the parent, pid=739728
I am the child, pid=739760 and given the fd 3
enter a positive number:
4
child write 4 bytes.
I brought the number to the power 2 and wrote the result: 16.
parent read 4 bytes
here is the result: 21
parent opens the file for R/W with fd: 3
I am the parent, pid=739728
I am the child, pid=739971 and given the fd 3
enter a positive number:
41
child write 4 bytes.
I brought the number to the power 2 and wrote the result: 1681.
parent read 4 bytes
here is the result: 1686
parent opens the file for R/W with fd: 3
I am the parent, pid=739728
I am the child, pid=740147 and given the fd 3
enter a positive number:
-1
child: the user wants to end the program.
parent read 4 bytes
child exits on user -1. I exit too.
```

The parent is the same, but each time we give birth to a new child!

Example III: Solution B
Same Child

Parent

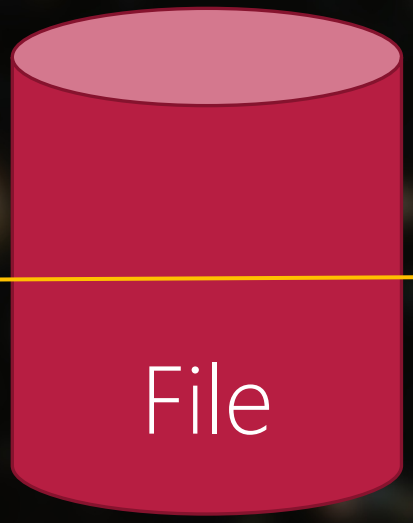
There is nothing for me yet. I sleep.

Ok, Read X*X

X*X + 5

Print out the final result

Wake up child! I'm done.



Child

I'm waiting for the user …

User entered X

Write X * X

Wake up ma! There is sth for you.

It's my turn to sleep.

Ok, let's start again …

Example III: Solution B
Same Child

IMPORTANT: the parent does NOT `wait()` for the child to `exit()`!
But `pause()` for the child for another round of conversation.

`sleep(int second)` cannot work because we depend on other process to wake up

```
hfani@charlie:~$ vi parent_child_conv_b.c
int main(int argc, char *argv[]){

        signal(SIGUSR1, parent_signal_handler);

        child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!\n");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("parent: I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("child: I am the child, pid=%d\n", getpid());
                        signal(SIGUSR2, child_signal_handler);
                        printf("child: I sleep until parent starts the work...\n");
                        pause();
                }
        }

        printf("parent: wake up child. It's time to work...\n");
        kill(child_pid, SIGUSR2);
        printf("parent: I sleep till you wake me up, child.\n");
        pause();
}
```

```
hfani@charlie:~$ vi parent_child_conv_b.c
int main(int argc, char *argv[]){

        signal(SIGUSR1, parent_signal_handler);

        child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!\n");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("parent: I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("child: I am the child, pid=%d\n", getpid());
                        signal(SIGUSR2, child_signal_handler);
                        printf("child: I sleep until parent starts the work...\n");
                        pause();
                }
        }

        printf("parent: wake up child. It's time to work...\n");
        kill(child_pid, SIGUSR2);
        printf("parent: I sleep till you wake me up, child.\n");
        pause();
}
```

```
hfani@charlie:~$ vi parent_child_conv_b.c
```

```c
void child_signal_handler(int signal){
        printf("child: I received a wake up signal from my parrent. The signal is %d\n", signal);
        int Y[1] = {-1};
        int X;
        printf("child: enter a positive number:\n");
        scanf("%d", &X);
        int fd = open(filename_2_share, O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
        printf("child opens the file with fd: %d\n", fd);
        if(X == -1){
                printf("child: the user wants to end the program.\n");
                write(fd, Y, sizeof(Y));
                exit(0);
        }
        Y[0] = X * X;
        int byte_write = write(fd, Y, sizeof(Y));
        close(fd);
        printf("child: write %d bytes.\n", byte_write);
        printf("child: I brought the number to the power 2 and wrote the result: %d.\n", Y[0]);
        printf("child: Ma, wake up ...\n");
        kill(getppid(), SIGUSR1);
        pause();
}
```

```
hfani@charlie:~$ vi parent_child_conv_b.c
```

```
void parent_signal_handler(int signal){
        printf("parent: I received a wake up signal from my child. The signal is %d\n", signal);
        int fd = open(filename_2_share, O_RDONLY);
        printf("parent: I opened the file with fd: %d\n", fd);
        int Y[1];
        int byte_read = read(fd, Y, sizeof(Y));
        printf("parent: I read %d bytes\n", byte_read);
        close(fd);
        int result = Y[0] + 5;
        printf("parent: here is the final result: %d\n", result);
        printf("parent: wake up child for another round of work ...");
        kill(SIGUSR2, child_pid);
        printf("parent: I sleep.");
        pause();
}
```

Example III: Solution B
Same Child

Synchronization! Collaboration! Cooperation!


The Mirror
Alexandre Desplat

Example III: Solution B

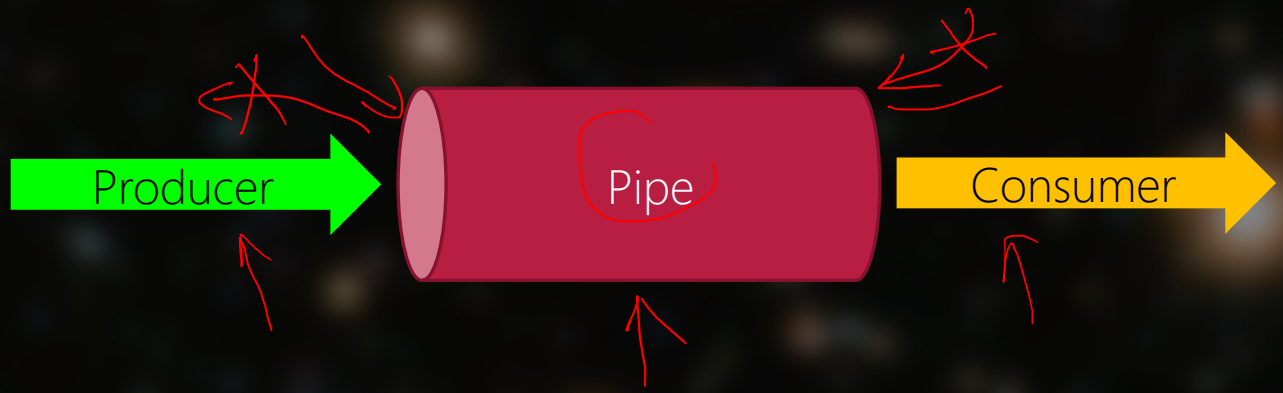
Does not work! Deadlock! Why?

```
hfani@charlie:~$ ./parent_child_conv_b
parent: I am the parent, pid=1023596
parent: wake up child. It's time to work...
parent: I sleep till you wake me up, child.
child: I am the child, pid=1023597
child: I sleep until parent starts the work...
```

Unnamed File → Pipe

Handles all opening, closing, seeking, pauses, wakeups, ….

Temporary File, Memory, Device, …. (We don't know)

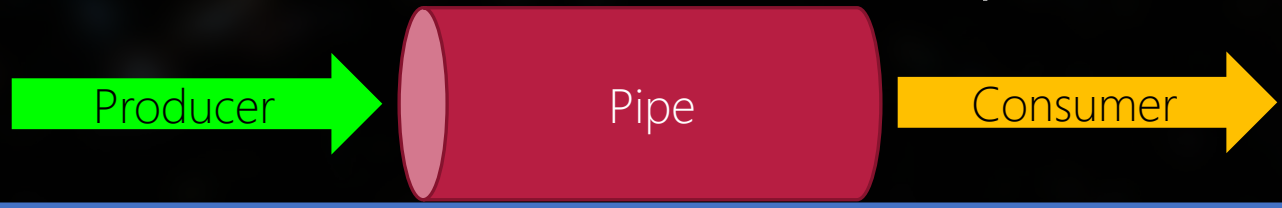


Unnamed File → Pipe

Half Duplex, Unidirectional, Forward Only

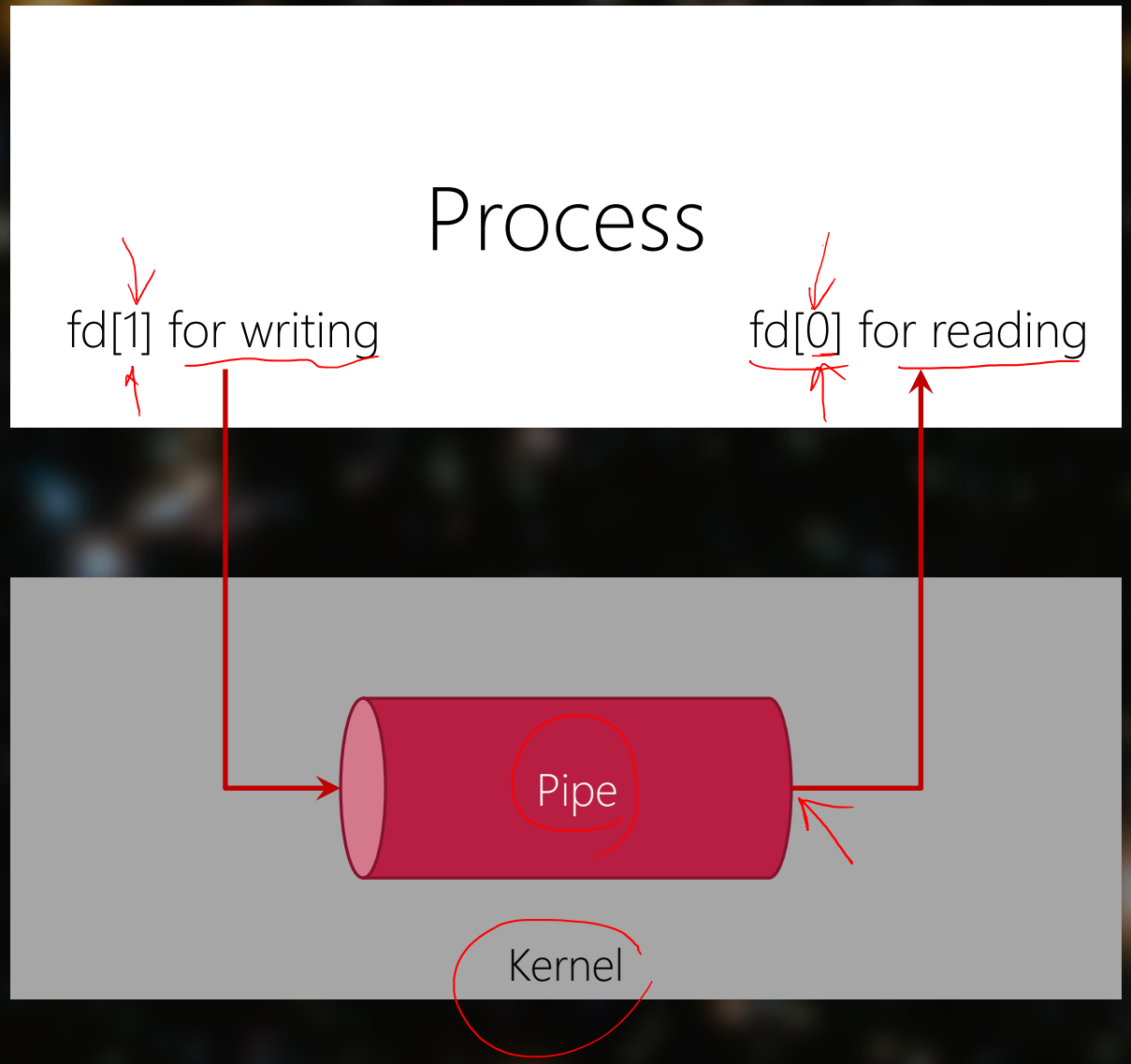
No `lseek()` or rewind!

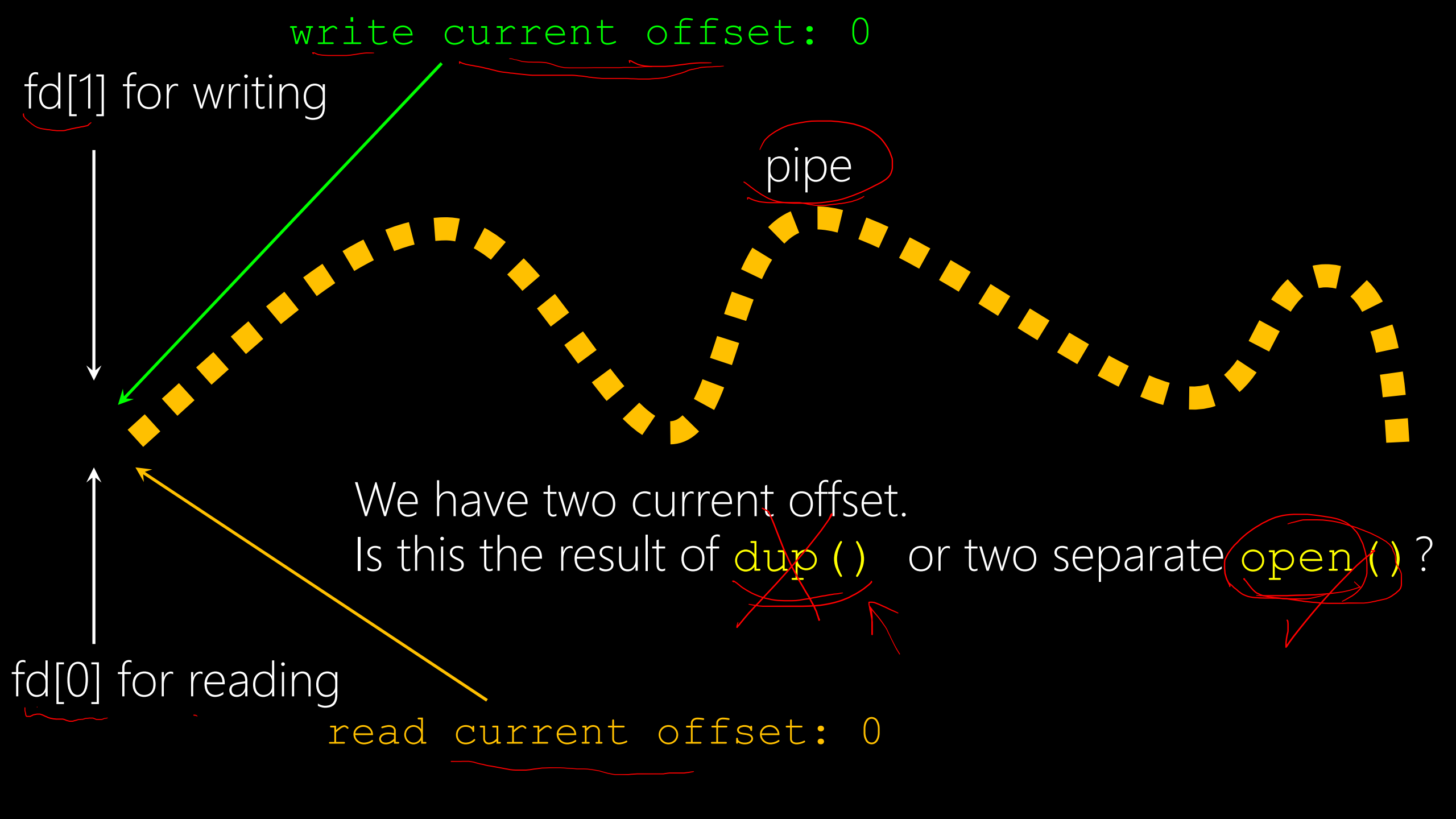
Unnamed File → Pipe

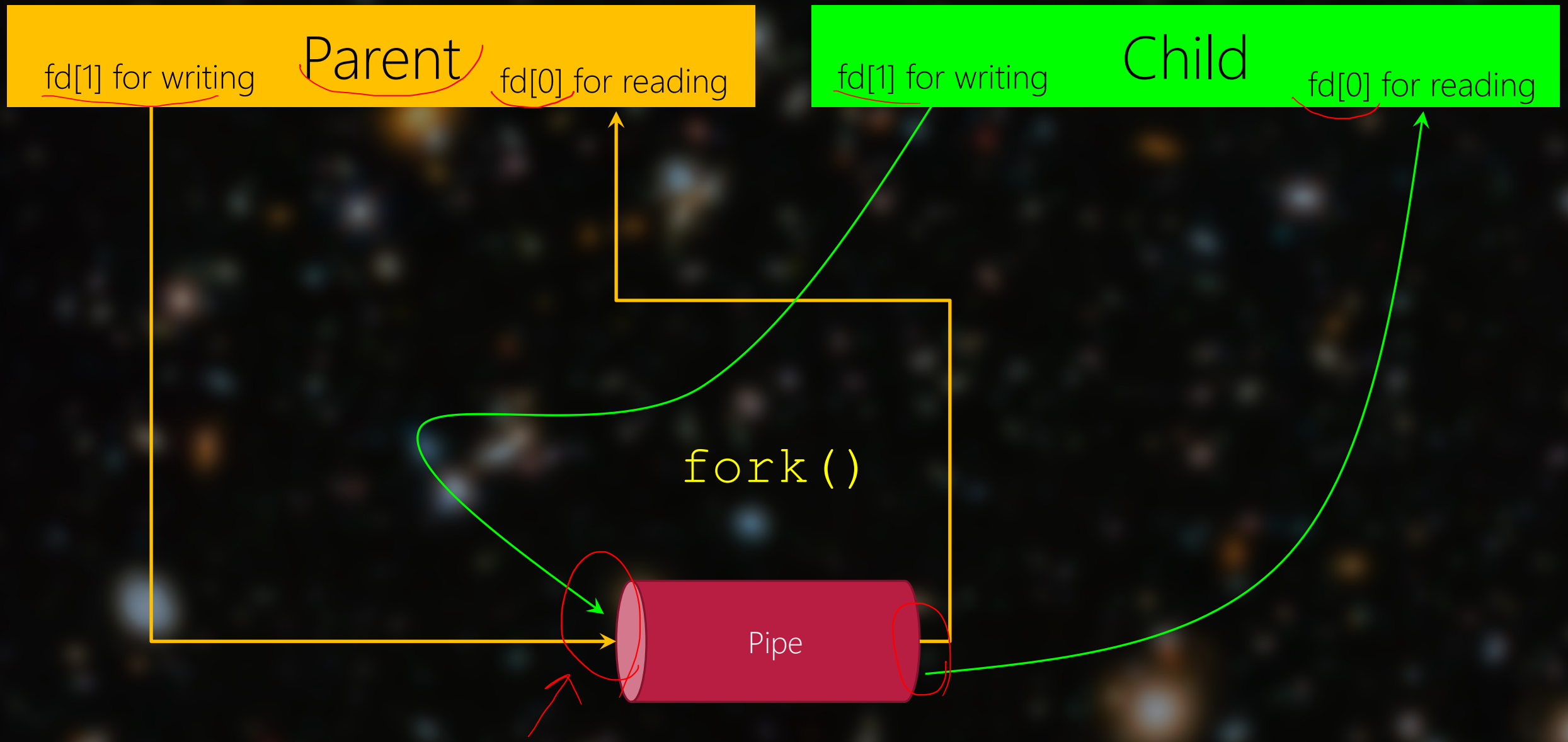


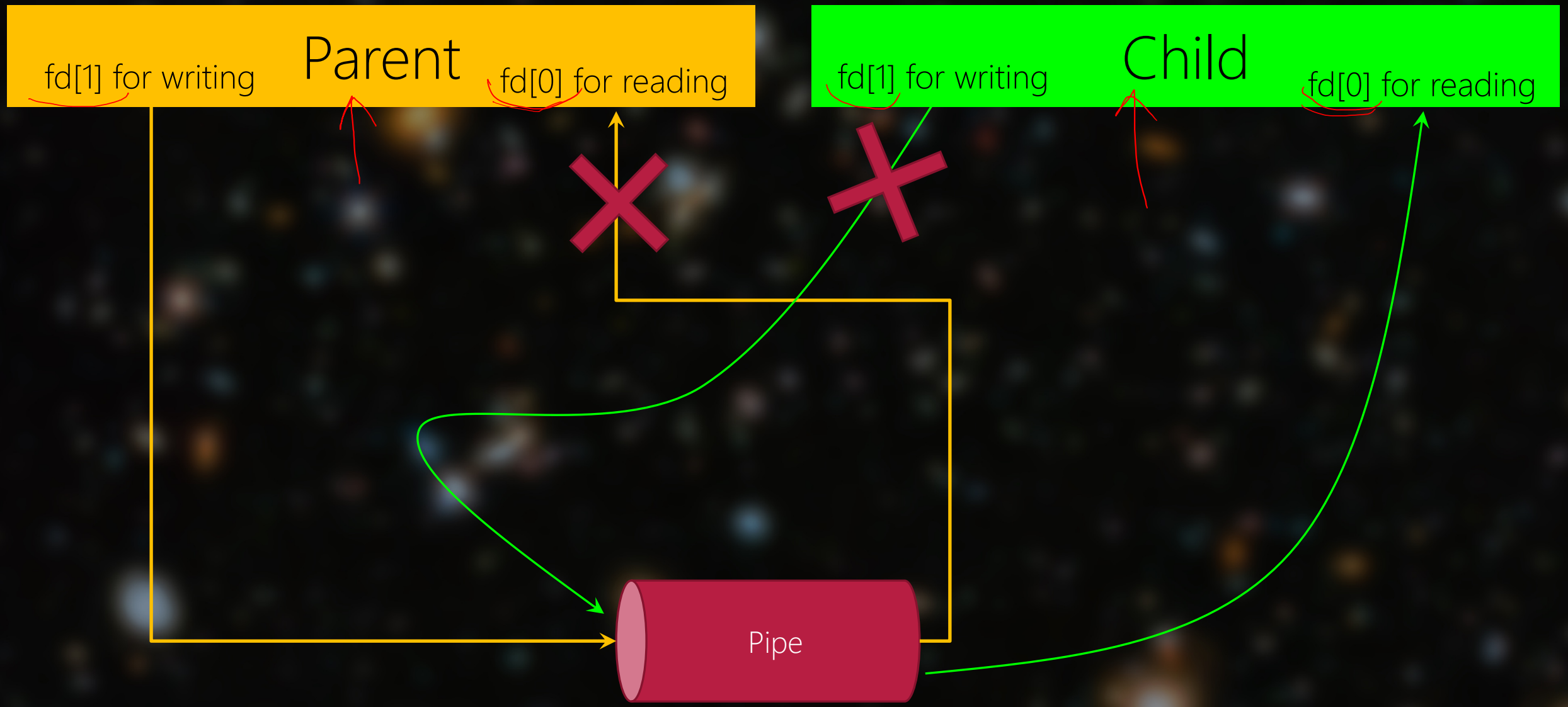

```
#include <unistd.h>
int pipe(int fd[2]);
```

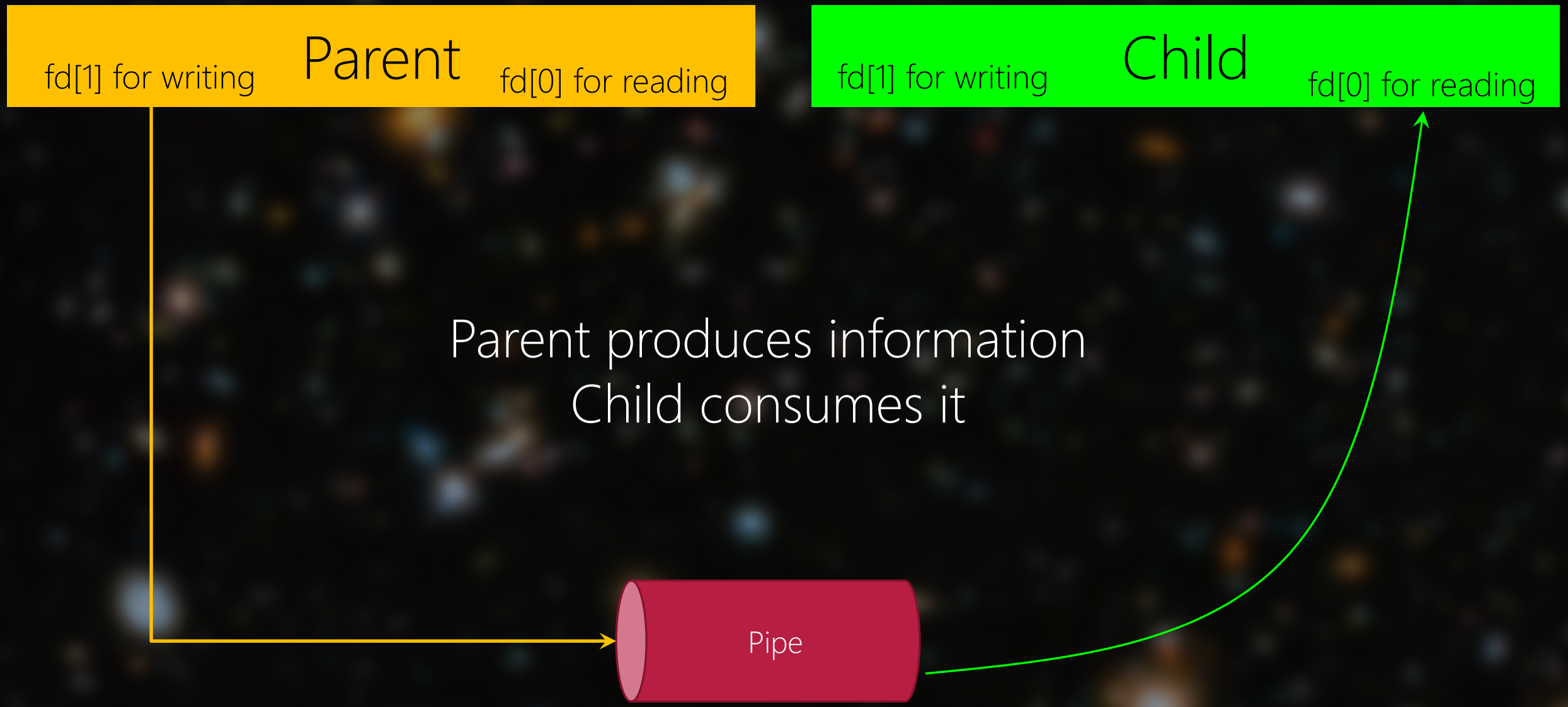
Returns 0 if OK, -1 on error

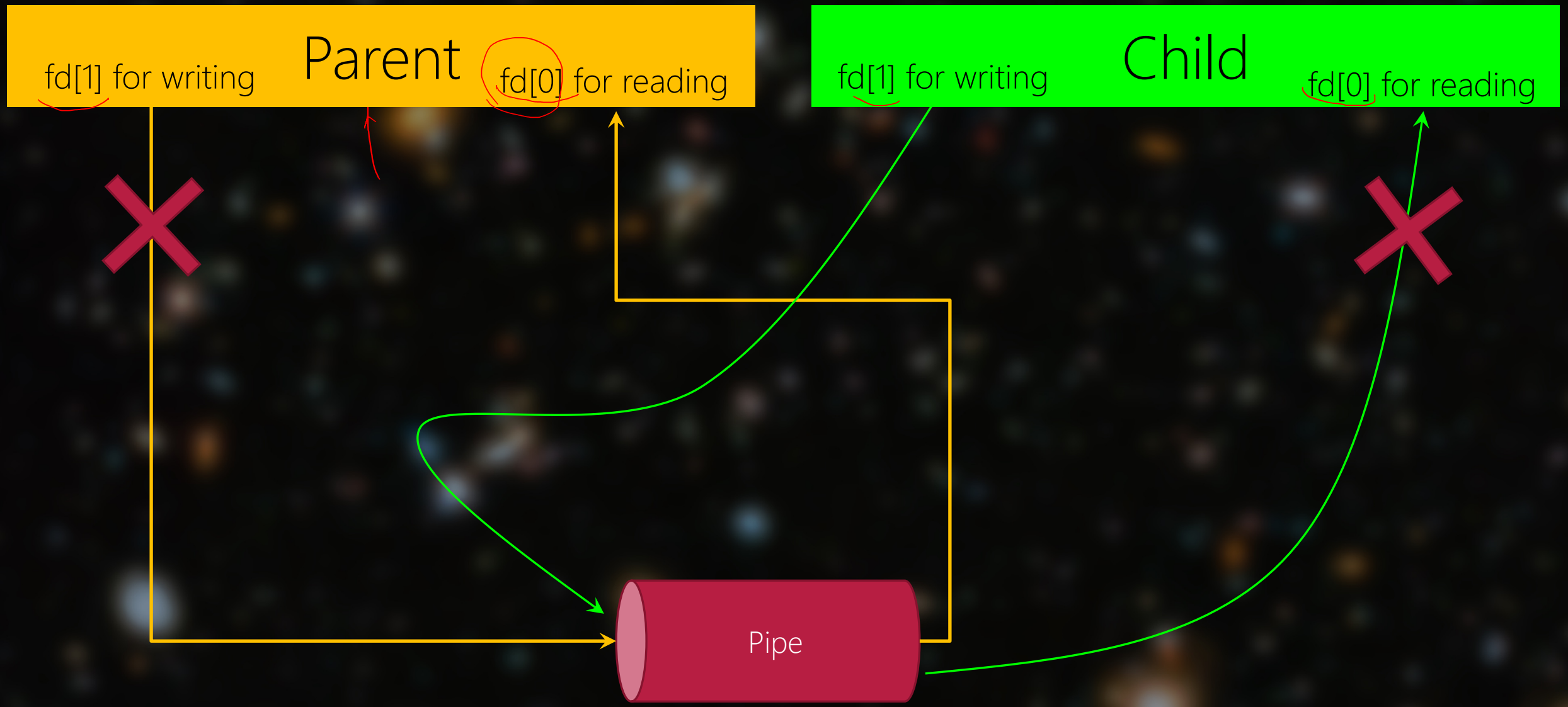

Process
fd[1] for writing
fd[0] for reading
Pipe
Kernel


write current offset: 0
fd[1] for writing
pipe
We have two current offset.
Is this the result of dup() or two separate open()?
fd[0] for reading
read current offset: 0


fd[1] for writing
Parent
fd[0] for reading
fd[1] for writing
Child
fd[0] for reading
fork()
Pipe

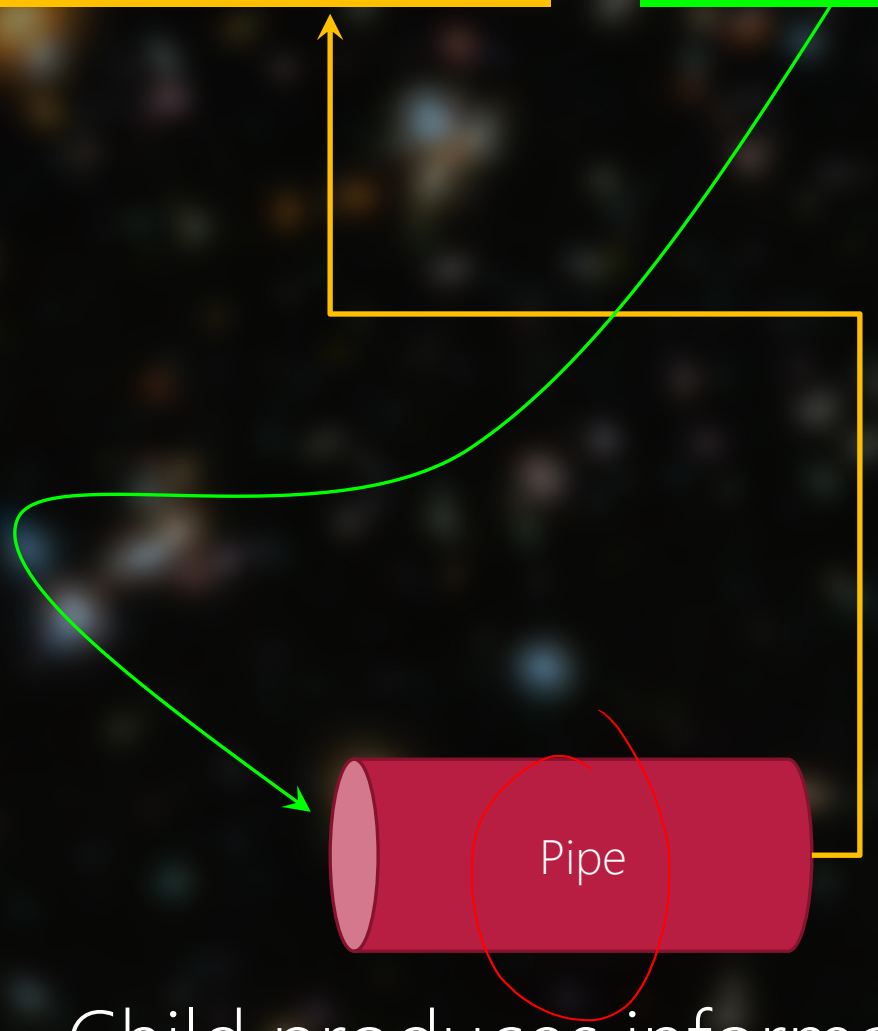

Parent
fd[1] for writing
fd[0] for reading
Child
fd[1] for writing
fd[0] for reading
Pipe


fd[1] for writing
Parent
fd[0] for reading
fd[1] for writing
Child
fd[0] for reading
Parent produces information
Child consumes it
Pipe

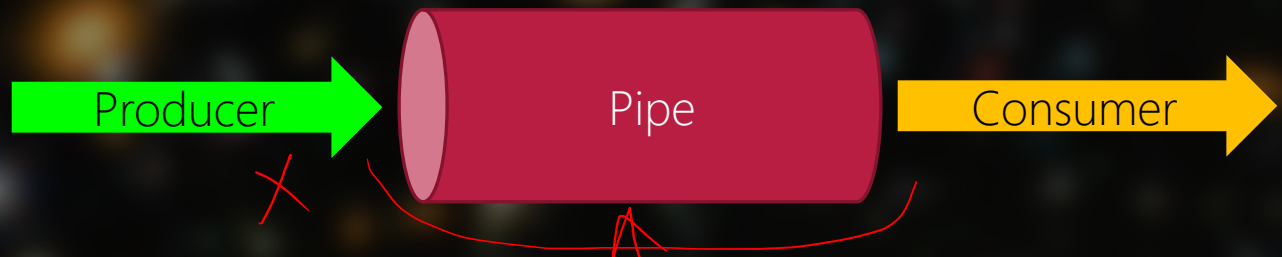

Parent
fd[1] for writing
fd[0] for reading
Child
fd[1] for writing
fd[0] for reading
Pipe

fd[1] for writing Parent fd[0] for reading

fd[1] for writing Child fd[0] for reading

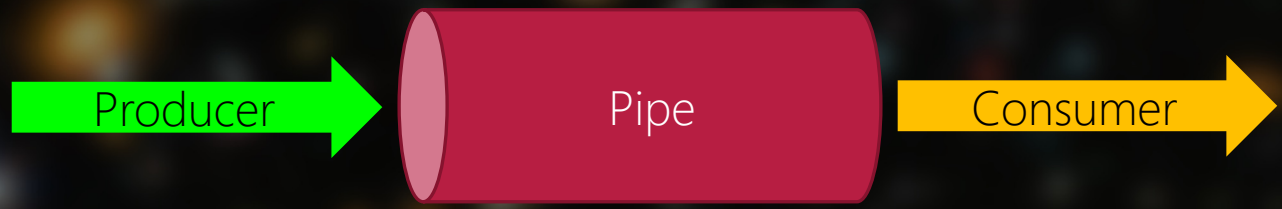


Child produces information
Parent consumes it



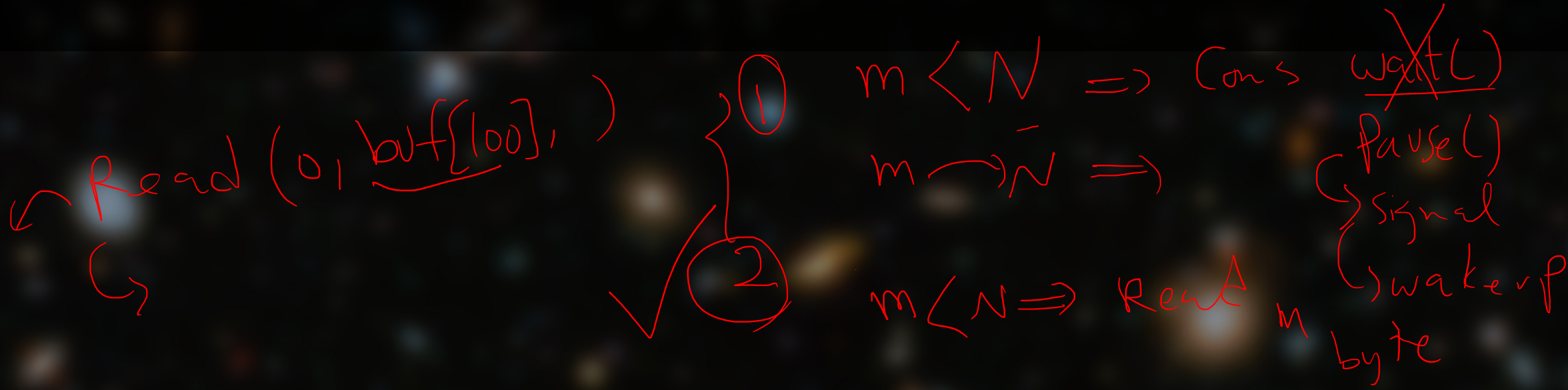
Situations:

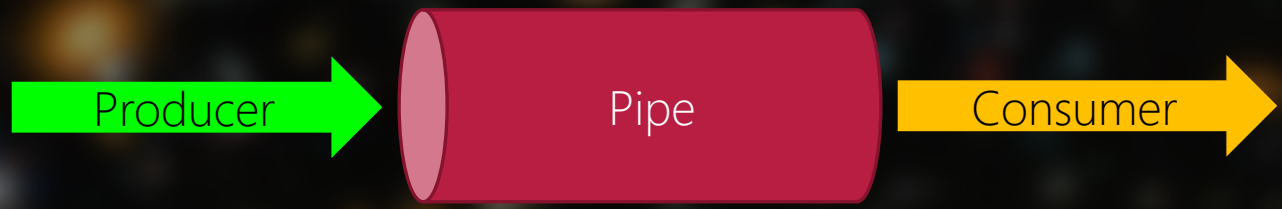
1) If the consumer wants to `read()` N bytes but there less data
2) If the consumer wants to `read()` but there is no data (empty pipe)
3) If the consumer wants to `read()` but there is no producer anymore
4) If the producer wants to `write()` but there is no consumer
5) If the producer wants to `write()` but pipe is full



Situations:

1) If the consumer wants to `read()` N bytes but there less data

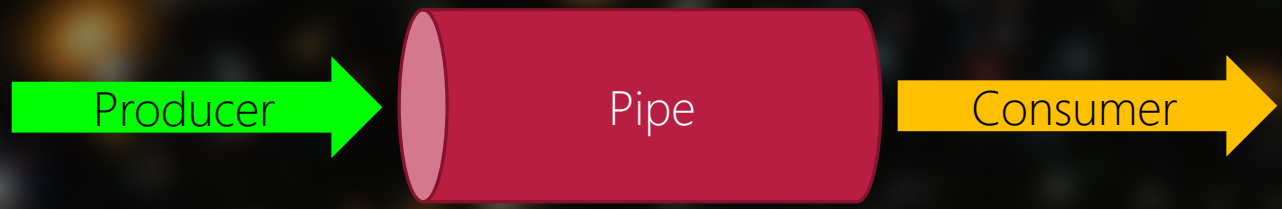




Situations:

1) If the consumer wants to `read()` N bytes but there less data

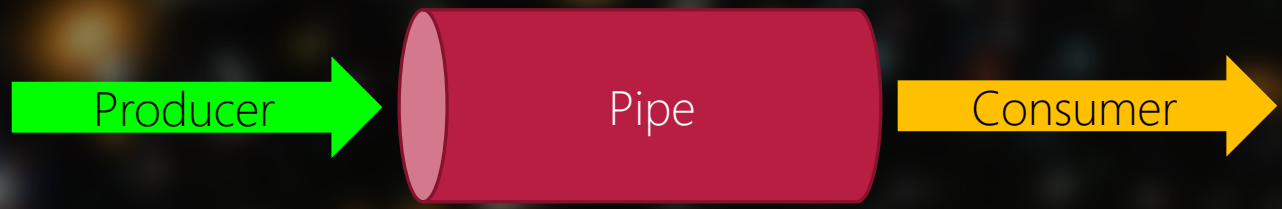
We already saw this when reading from a file while giving large buffer
Only the available data will be read



Situations:

2) If the consumer wants to `read()` but there is no data (empty pipe)

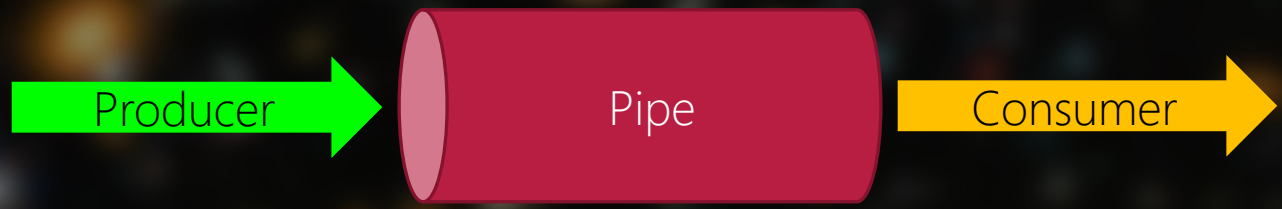
pause()



Situations:

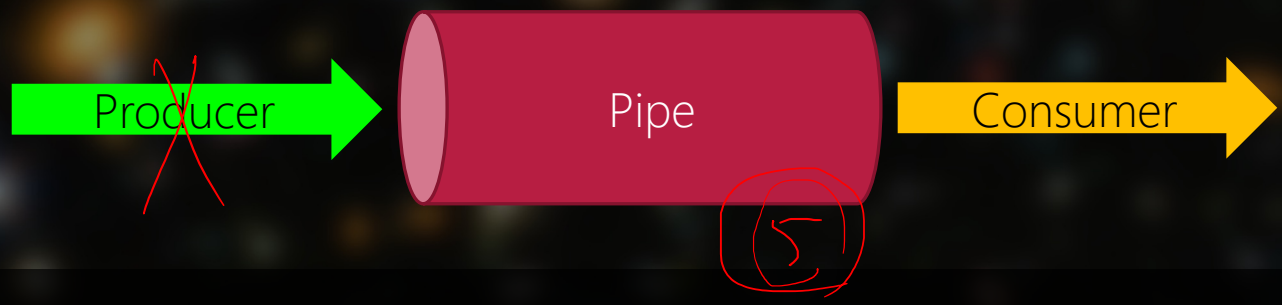
2) If the consumer wants to `read()` but there is no data (empty pipe)

If a producer exists, the consumer `pause()` till the kernel SIGNALs it when at least 1 byte become available



Situations:

3) If the consumer wants to `read()` but there is no producer anymore



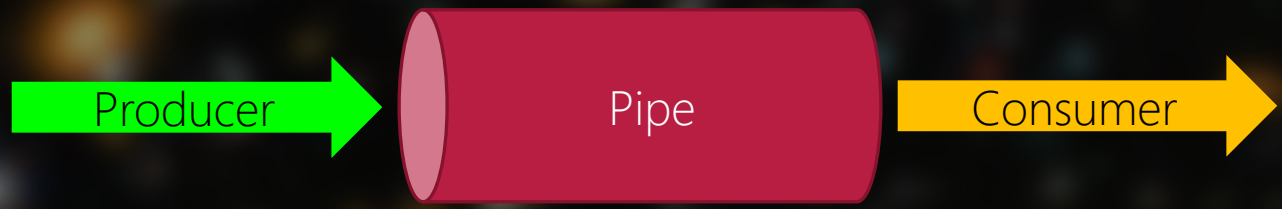
Situations:

3) If the consumer wants to `read()` but there is no producer anymore

The consumer can continue to read until there is no information left

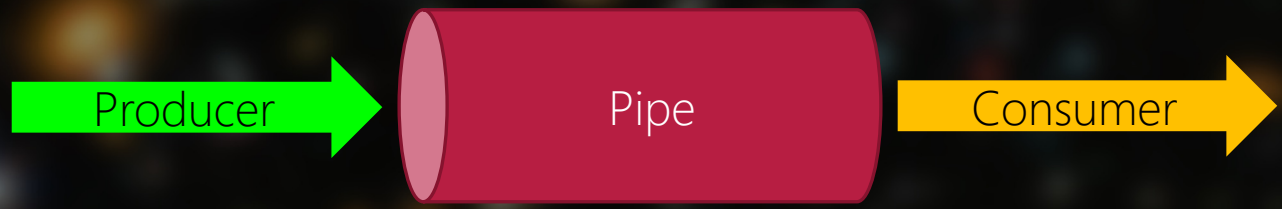
The consumer does NOT `pause()`

The last read returns 0 (EOF) and consumer decides to exit



Situations:

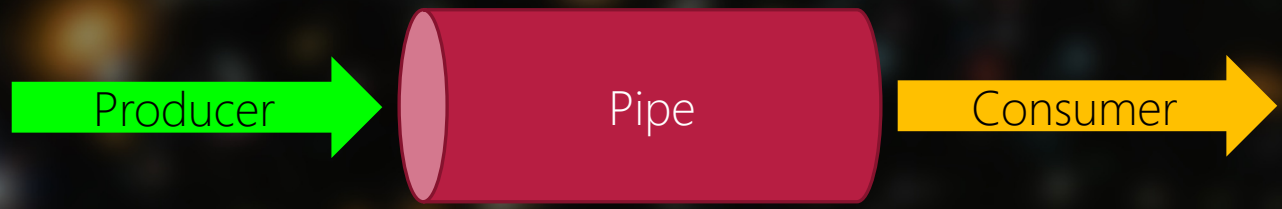
4) If the producer wants to `write()` but there is no consumer



Situations:

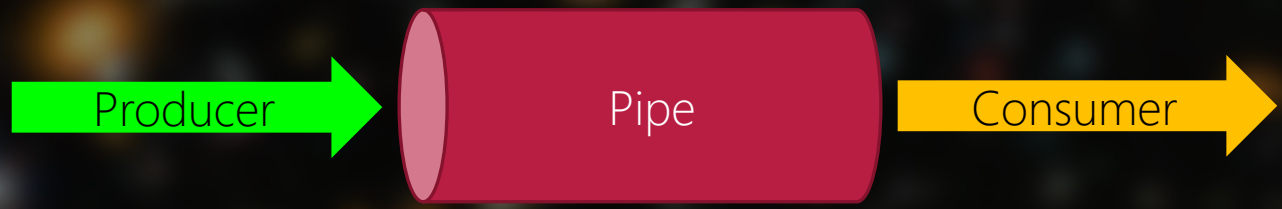
4) If the producer wants to `write()` but there is no consumer

The producer fails and receives SIGPIPE by the kernel



Situations:

5) If the producer wants to `write()` but pipe is full



Situations:

5) If the producer wants to `write()` but pipe is full

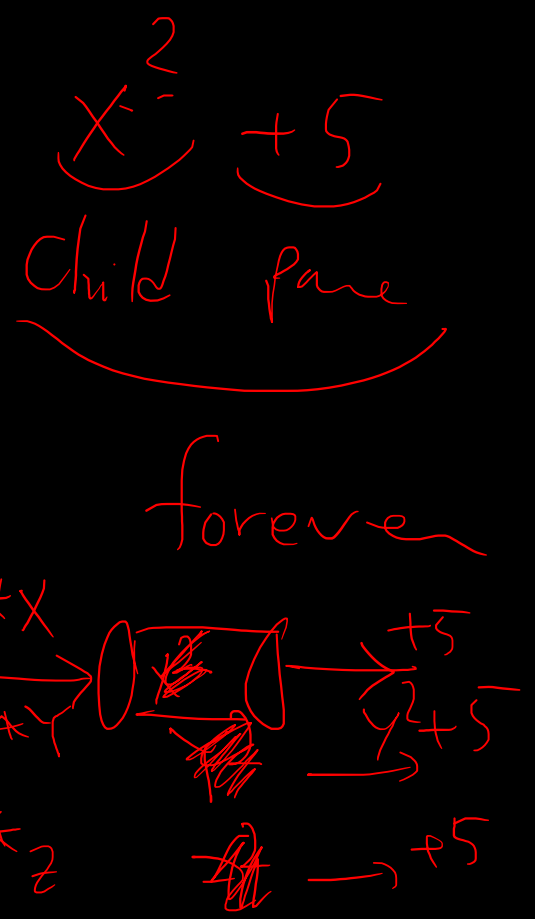
It `pause()` until consumer reads some and make some space!

```
hfani@charlie:~$ vi pipe.c
int main(void)
{
        int fd[2];

        if (pipe(fd) < 0){
                printf("pipe error.\n");
                exit(1);
        }

        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!\n");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("chile: I am the child, pid=%d and given the fd %d\n", getpid(), fd);
                        printf("child: I want to be the producer.\n");
                        close(fd[0]);
                        int Y[1] = {-1};
                        int X;
                        while(1){
                                printf("child: enter a positive number:\n");
                                scanf("%d", &X);
                                if(X == -1){
                                        printf("child: the user wants to end the program.\n");
                                        exit(0);
                                }
                                Y[0] = X * X;
                                int byte_write = write(fd[1], Y, sizeof(Y));
                                printf("child write %d bytes.\n", byte_write);
                                printf("child: I brought the number to the power 2 and wrote the result: %d.\n", Y[0]);
                        }
                }
        }
        printf("parent: I want to be the consumer.\n");
        close(fd[1]);
        while(1){
                int Y[1];
                int byte_read = read(fd[0], Y, sizeof(Y));
                if (byte_read == 0){
                        printf("parent: there is no more data and no producer. I exit.\n");
                        exit(0);
                }
                printf("parent read %d bytes\n", byte_read);
                int result = Y[0] + 5;
                printf("here is the result: %d\n", result);
        }
}
```

Passing an array of `fd[2]` to `pipe()` and receiving separate read and write file descriptors.



```
hfani@charlie:~$ vi pipe.c
int main(void)
{
        int fd[2];

        if (pipe(fd) < 0){
                printf("pipe error.\n");
                exit(1);
        }

        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!\n");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("chile: I am the child, pid=%d and given the fd %d\n", getpid(), fd);
                        printf("child: I want to be the producer.\n");
                        close(fd[0]);
                        int Y[1] = {-1};
                        int X;
                        while(1){
                                printf("child: enter a positive number:\n");
                                scanf("%d", &X);
                                if(X == -1){
                                        printf("child: the user wants to end the program.\n");
                                        exit(0);
                                }
                                Y[0] = X * X;
                                int byte_write = write(fd[1], Y, sizeof(Y));
                                printf("child write %d bytes.\n", byte_write);
                                printf("child: I brought the number to the power 2 and wrote the result: %d.\n", Y[0]);
                        }
                }
        }
        printf("parent: I want to be the consumer.\n");
        close(fd[1]);
        while(1){
                int Y[1];
                int byte_read = read(fd[0], Y, sizeof(Y));
                if (byte_read == 0){
                        printf("parent: there is no more data and no producer. I exit.\n");
                        exit(0);
                }
                printf("parent read %d bytes\n", byte_read);
                int result = Y[0] + 5;
                printf("here is the result: %d\n", result);
        }
}
```

Child is the producer.
So, it closes the read descriptor `fd[0]`

Parent is the consumer.
So, it closes the write descriptor `fd[1]`

```
hfani@charlie:~$ vi pipe.c
int main(void)
{
        int fd[2];

        if (pipe(fd) < 0){
                printf("pipe error.\n");
                exit(1);
        }

        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!\n");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("chile: I am the child, pid=%d and given the fd %d\n", getpid(), fd);
                        printf("child: I want to be the producer.\n");
                        close(fd[0]);
                        int Y[1] = {-1};
                        int X;
                        while(1){
                                printf("child: enter a positive number:\n");
                                scanf("%d", &X);
                                if(X == -1){
                                        printf("child: the user wants to end the program.\n");
                                        exit(0);
                                }
                                Y[0] = X * X;
                                int byte_write = write(fd[1], Y, sizeof(Y));
                                printf("child write %d bytes.\n", byte_write);
                                printf("child: I brought the number to the power 2 and wrote the result: %d.\n", Y[0]);
                        }
                }
        }
        printf("parent: I want to be the consumer.\n");
        close(fd[1]);
        while(1){
                int Y[1];
                int byte_read = read(fd[0], Y, sizeof(Y));
                if (byte_read == 0){
                        printf("parent: there is no more data and no producer. I exit.\n");
                        exit(0);
                }
                printf("parent read %d bytes\n", byte_read);
                int result = Y[0] + 5;
                printf("here is the result: %d\n", result);
        }
}
```

Child produces forever until the user enters -1

Parent consumes forever until the child is working

If the child exits, the parent make sure to consume all the data first and then exits.

```
hfani@charlie:~$ vi pipe.c
int main(void)
{
        int fd[2];

        if (pipe(fd) < 0){
                printf("pipe error.\n");
                exit(1);
        }

        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!\n");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("chile: I am the child, pid=%d and given the fd %d\n", getpid(), fd);
                        printf("child: I want to be the producer.\n");
                        close(fd[0]);
                        int Y[1] = {-1};
                        int X;
                        while(1){
                                printf("child: enter a positive number:\n");
                                scanf("%d", &X);
                                if(X == -1){
                                        printf("child: the user wants to end the program.\n");
                                        exit(0);
                                }
                                Y[0] = X * X;
                                int byte_write = write(fd[1], Y, sizeof(Y));
                                printf("child write %d bytes.\n", byte_write);
                                printf("child: I brought the number to the power 2 and wrote the result: %d.\n", Y[0]);
                        }
                }
        }
        printf("parent: I want to be the consumer.\n");
        close(fd[1]);
        while(1){
                int Y[1];
                int byte_read = read(fd[0], Y, sizeof(Y));
                if (byte_read == 0){
                        printf("parent: there is no more data and no producer. I exit.\n");
                        exit(0);
                }
                printf("parent read %d bytes\n", byte_read);
                int result = Y[0] + 5;
                printf("here is the result: %d\n", result);
        }
}
```

If child wants to write but the pipe is full, it pauses

Synchronization

If parent wants to consume but there is no data, it pauses

```
hfani@charlie:~$ vi pipe.c
int main(void)
{
        int fd[2];

        if (pipe(fd) < 0){
                printf("pipe error.\n");
                exit(1);
        }

        int child_pid = fork();
        if(child_pid == -1){
                perror("impossible to have a child!\n");
                exit(1);
        }
        if(child_pid >= 0){//(child_pid != -1)
                if(child_pid > 0)
                        printf("I am the parent, pid=%d\n", getpid());
                else{//(child_pid == 0)
                        printf("chile: I am the child, pid=%d and given the fd %d\n", getpid(), fd);
                        printf("child: I want to be the producer.\n");
                        close(fd[0]);
                        int Y[1] = {-1};
                        int X;
                        while(1){
                                printf("child: enter a positive number:\n");
                                scanf("%d", &X);
                                if(X == -1){
                                        printf("child: the user wants to end the program.\n");
                                        exit(0);
                                }
                                Y[0] = X * X;
                                int byte_write = write(fd[1], Y, sizeof(Y));
                                printf("child write %d bytes.\n", byte_write);
                                printf("child: I brought the number to the power 2 and wrote the result, %d.\n", Y[0]);
                        }
                }
        }
        printf("parent: I want to be the consumer.\n");
        close(fd[1]);
        while(1){
                int Y[1];
                int byte_read = read(fd[0], Y, sizeof(Y));
                if (byte_read == 0){
                        printf("parent: there is no more data and no producer. I exit.\n");
                        exit(0);
                }
                printf("parent read %d bytes\n", byte_read);
                int result = Y[0] + 5;
                printf("here is the result: %d\n", result);
        }
}
```

(I want(0))

There is no `wait()` system call for parent!

```
hfani@charlie:~$ cc pipe.c -o pipe
hfani@charlie:~$ ./pipe
I am the parent, pid=1041949
parent: I want to be the consumer.
chile: I am the child, pid=1041950 and given the fd 318395608
child: I want to be the producer.
child: enter a positive number:
2
child write 4 bytes.
child: I brought the number to the power 2 and wrote the result: 4.
child: enter a positive number:
parent read 4 bytes
here is the result: 9
3
child write 4 bytes.
child: I brought the number to the power 2 and wrote the result: 9.
child: enter a positive number:
parent read 4 bytes
here is the result: 14
-1
child: the user wants to end the program.
parent: there is no more data and no producer. I exit.
```

Week#2

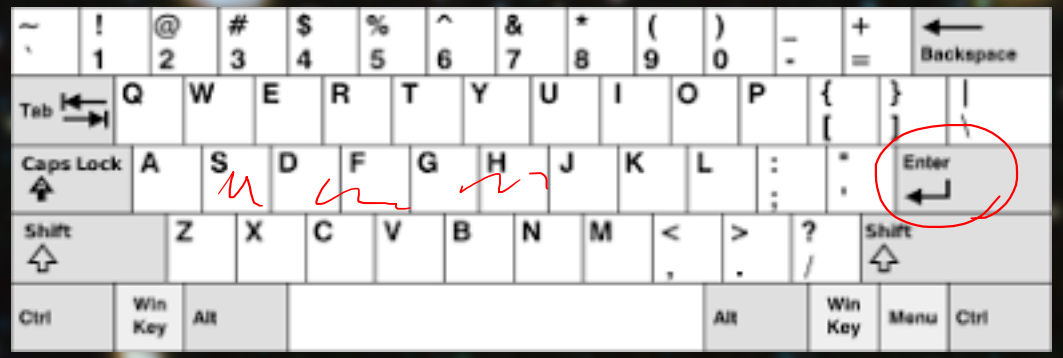
Interrupt Request (IRQ)

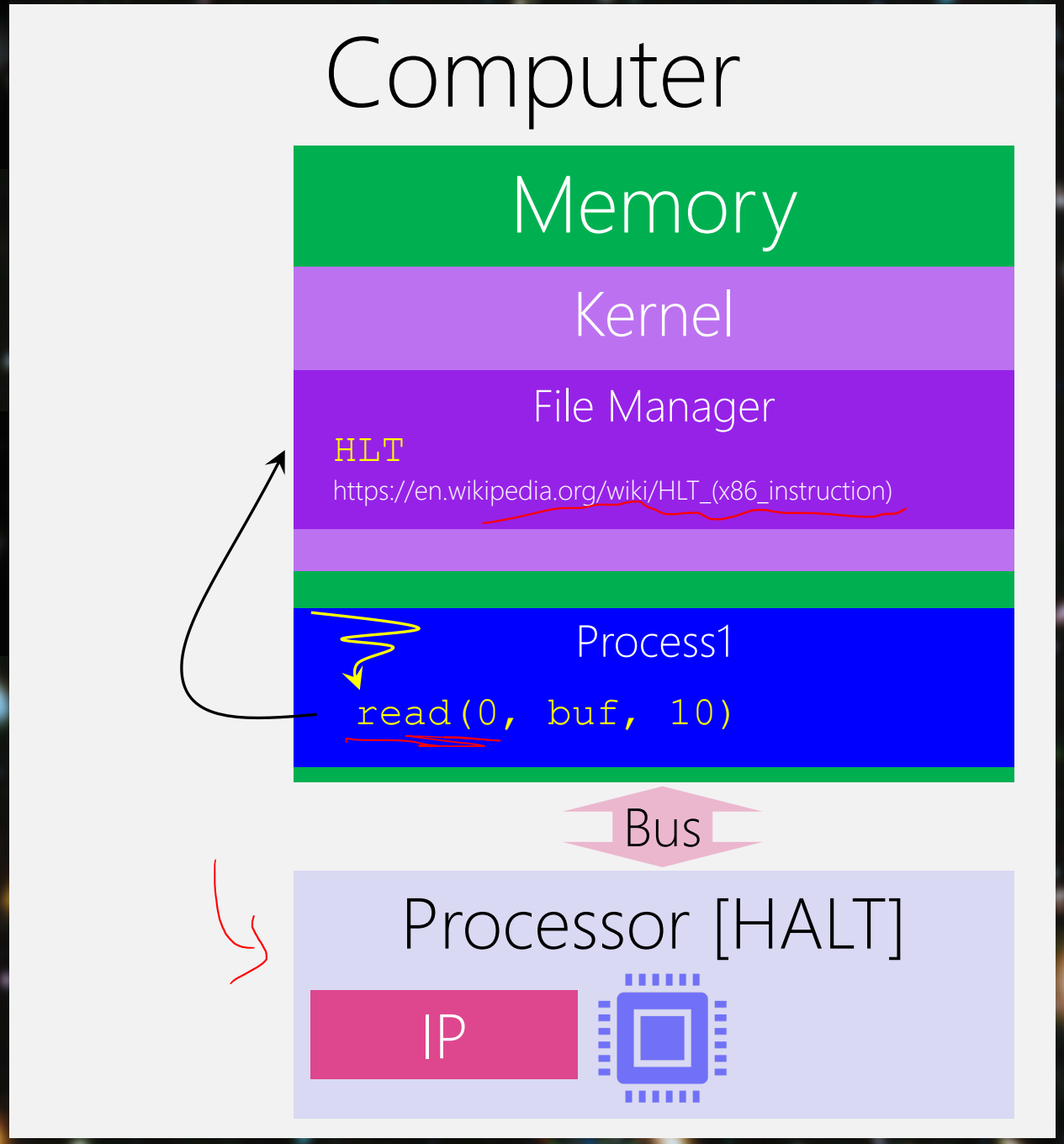
Interrupt Request Handler

What is happening next?

What is the processor doing?

B) HALT State





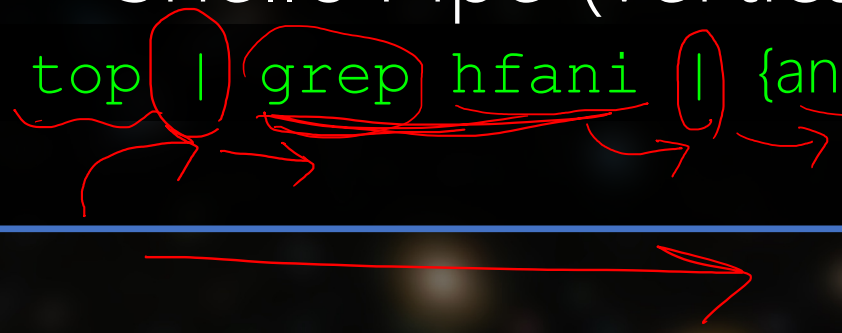
```
hfani@charlie:~$ cc pipe.c -o pipe
hfani@charlie:~$ ./pipe
I am the parent, pid=1041949
parent: I want to be the consumer.
chile: I am the child, pid=1041950 and given the fd 318395608
child: I want to be the producer.
child: enter a positive number:
2
child write 4 bytes.
child: I brought the number to the power 2 and wrote the result: 4.
child: enter a positive number:
parent read 4 bytes
here is the result: 9
3
child write 4 bytes.
child: I brought the number to the power 2 and wrote the result: 9
child: enter a positive number:
parent read 4 bytes
here is the result: 14
-1
child: the user wants to end the program.
parent: there is no more data and no producer. I exit.
```

Appreciate the benefit of processor sharing:
While the child is waiting for user new input, the parent does the addition with 5 for previous one!

How big is the pipe?

Shell's Pipe (vertical bar `|')

```
top | grep hfani | {another program}
```



Named Pipe → FIFO

Like Pipe but ...

```
mkfifo(const char *path, mode_t mode)
```

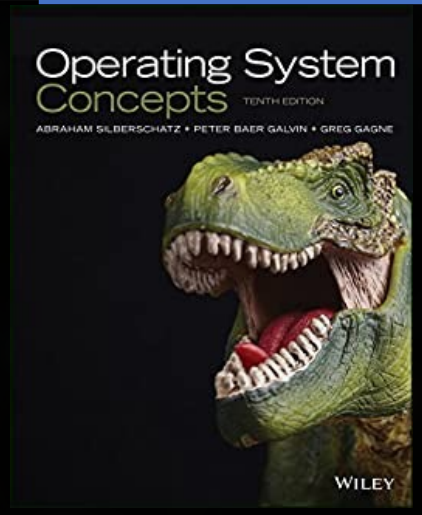
Pipe	FIFO
Unamed File, cannot be found in File System	Named File, should be **`open()`** like a regular to read or write
Between processes with the *same ancestor*	Between *any* processes
It is *deleted* after processes are terminated.	It *exists* even after processes termination. Should be explicitly deleted.

Half Duplex Conversation: One Talks, One Listens

Full Duplex Conversation: Both Talk, Both Listen

if both talk at the same time? if both listen at the same time?

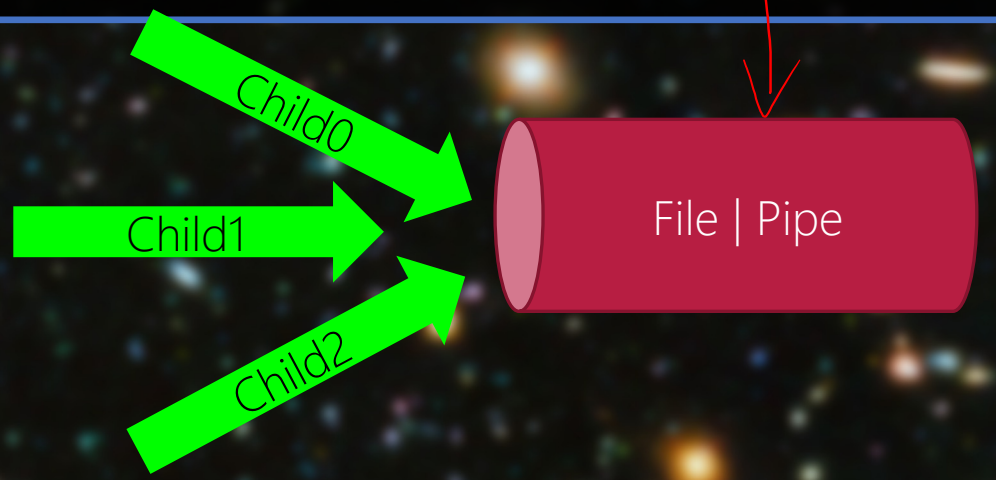
Advanced Synchronization: semaphore, mutex, ...



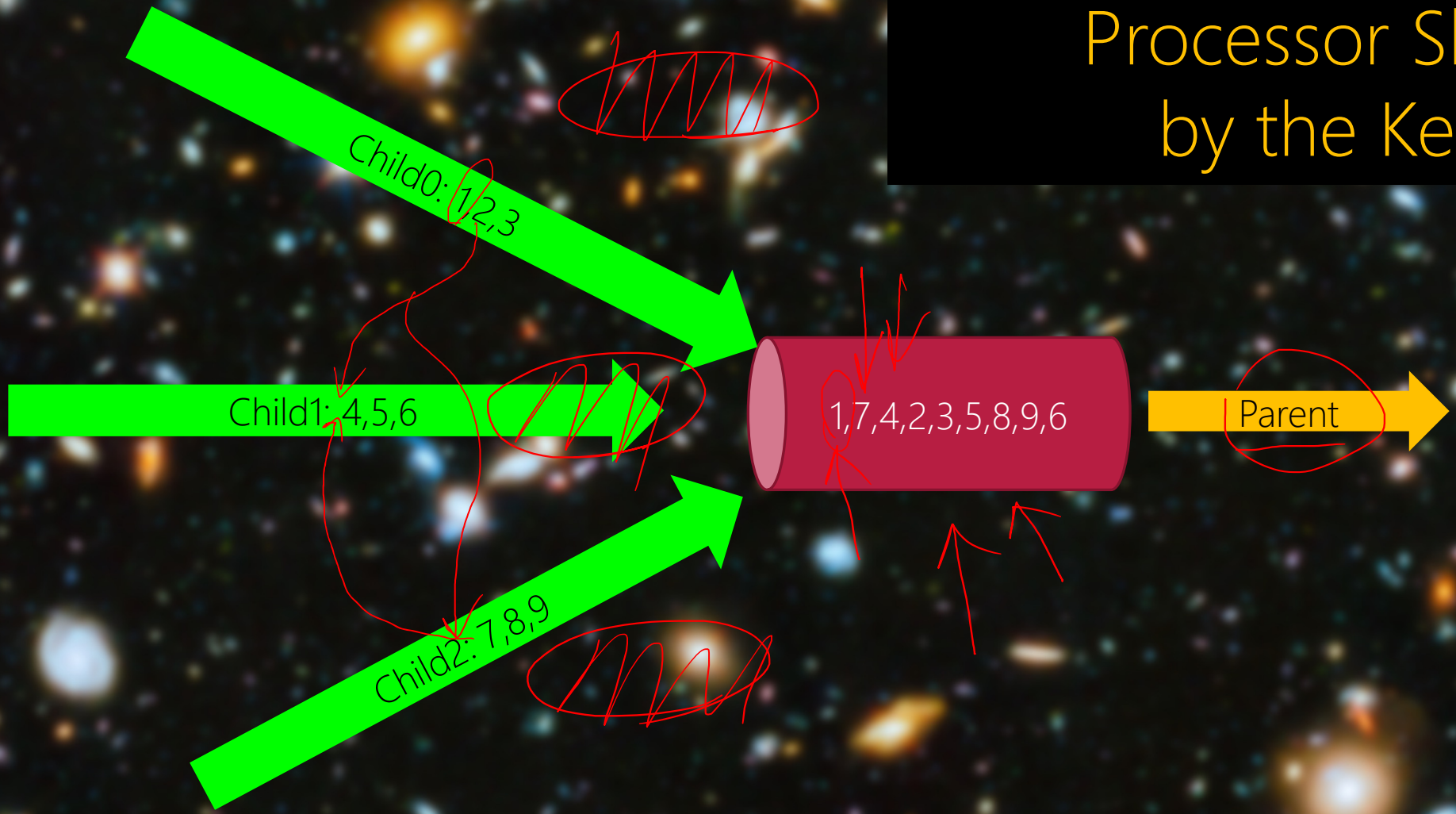
COMP3300: Operating Systems Fundamentals

Race Condition, Mutual Access
Semaphores, Mutex, ...

Lab09: Shared File | Pipe between Children

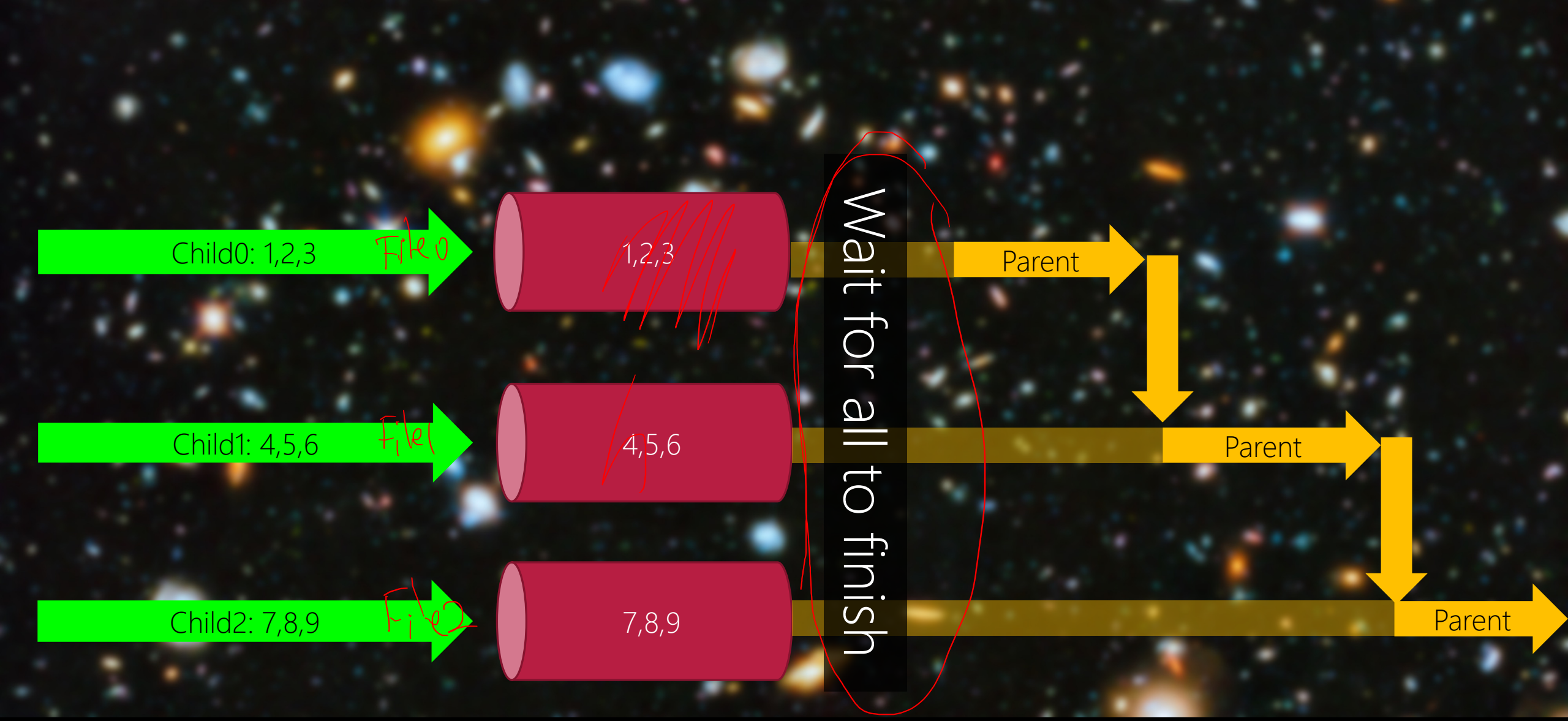


No Assumption about Processor Sharing by the Kernel



Sleep

Still, There is a Problem.
Why?



Always correct regardless of processor scheduling.

What We Talk About When We Talk About Love!?
Do you want a family?
Single Processor, Single Parent, Many Children

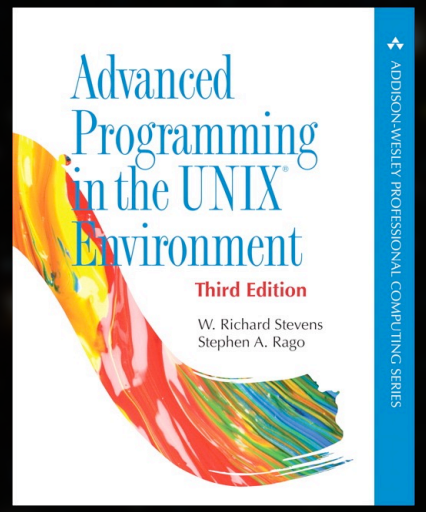
Canada child benefit

Education Savings Week is November 13-19, 2022! It's never too early to start saving for your child's education. Open a registered education savings plan today! For more information visit canada.ca/education-savings.

The CRA will not deduct your Canada child benefit (CCB) payments if you have amounts owing due to being ineligible for COVID-19 Canada Emergency or Recovery Benefit payments.

The Canada child benefit (CCB) is administered by the Canada Revenue Agency (CRA). It is a tax-free monthly payment made to eligible families to help with the cost of raising children under 18 years of age. The CCB may include the child disability benefit and any related provincial and territorial programs.

pats0n.livejournal.com



Chapter 16: Network IPC (Sockets)