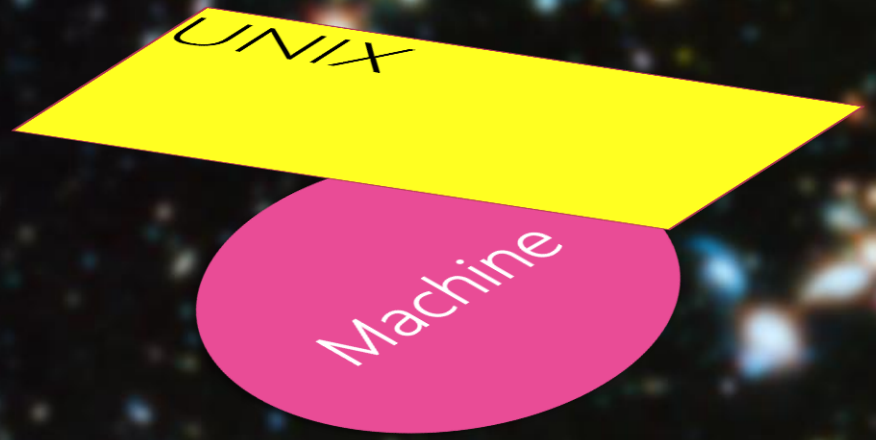
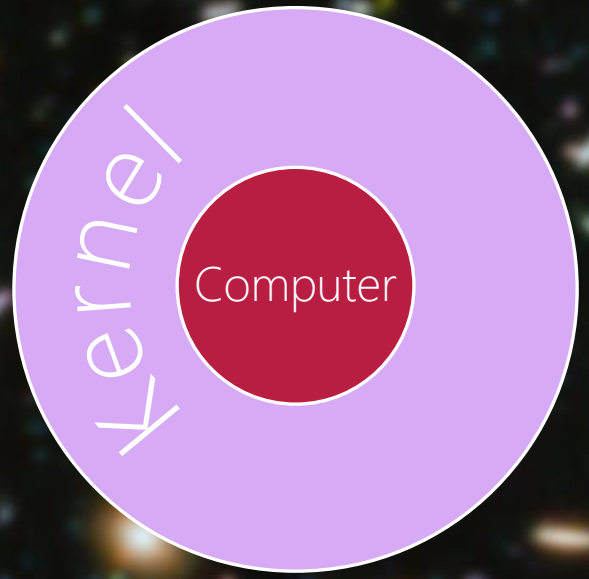
Rosemary’s Baby (1968), Roman Polanski
Based on the novel Rosemary's Baby, Ira Levin





Computer

Memory

Kernel: Device Manager

Kernel: Memory Manager

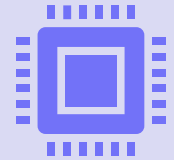
Kernel: File Manager

Kernel: Network Manager

Kernel: Process Manager

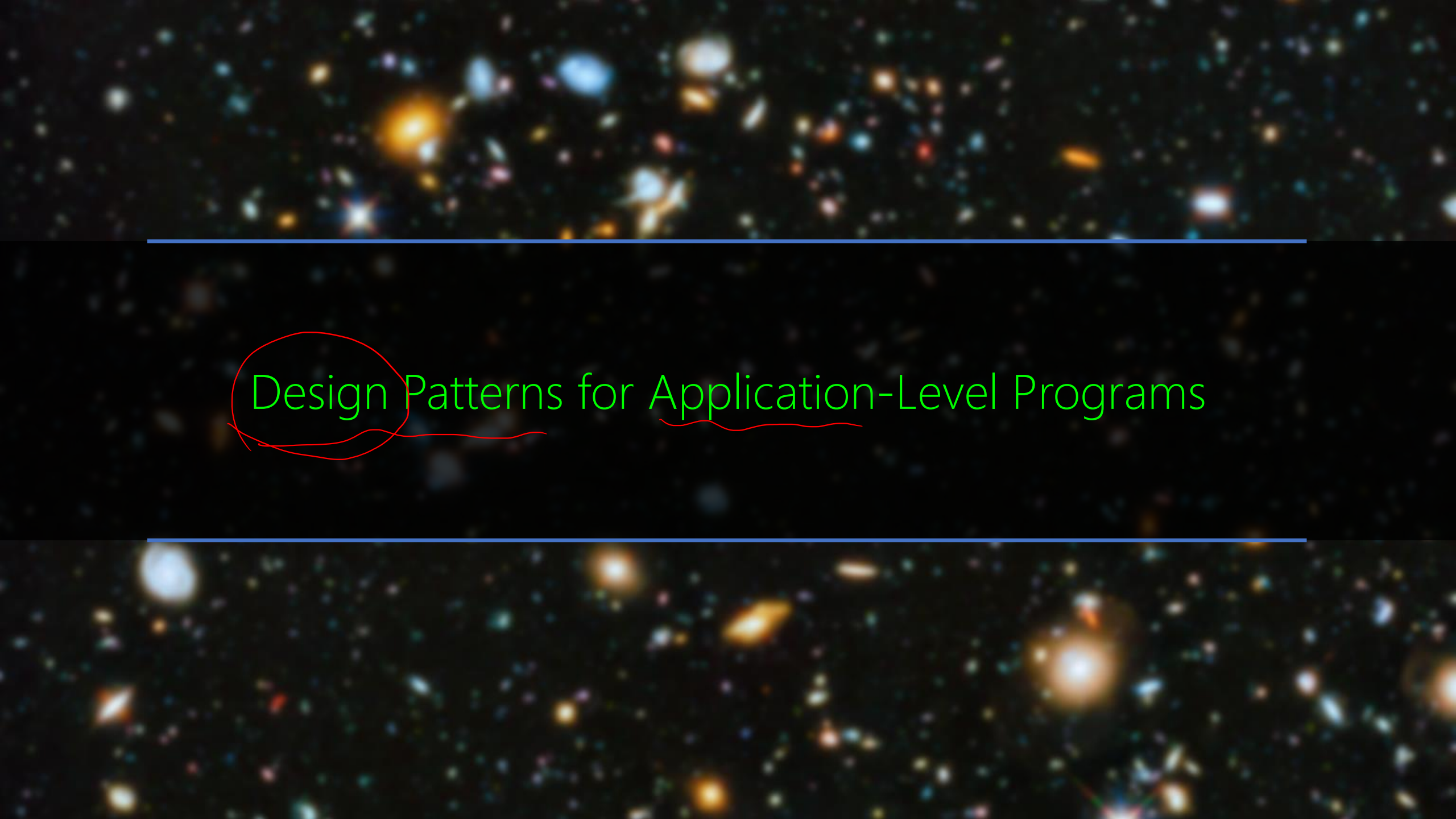
Bus

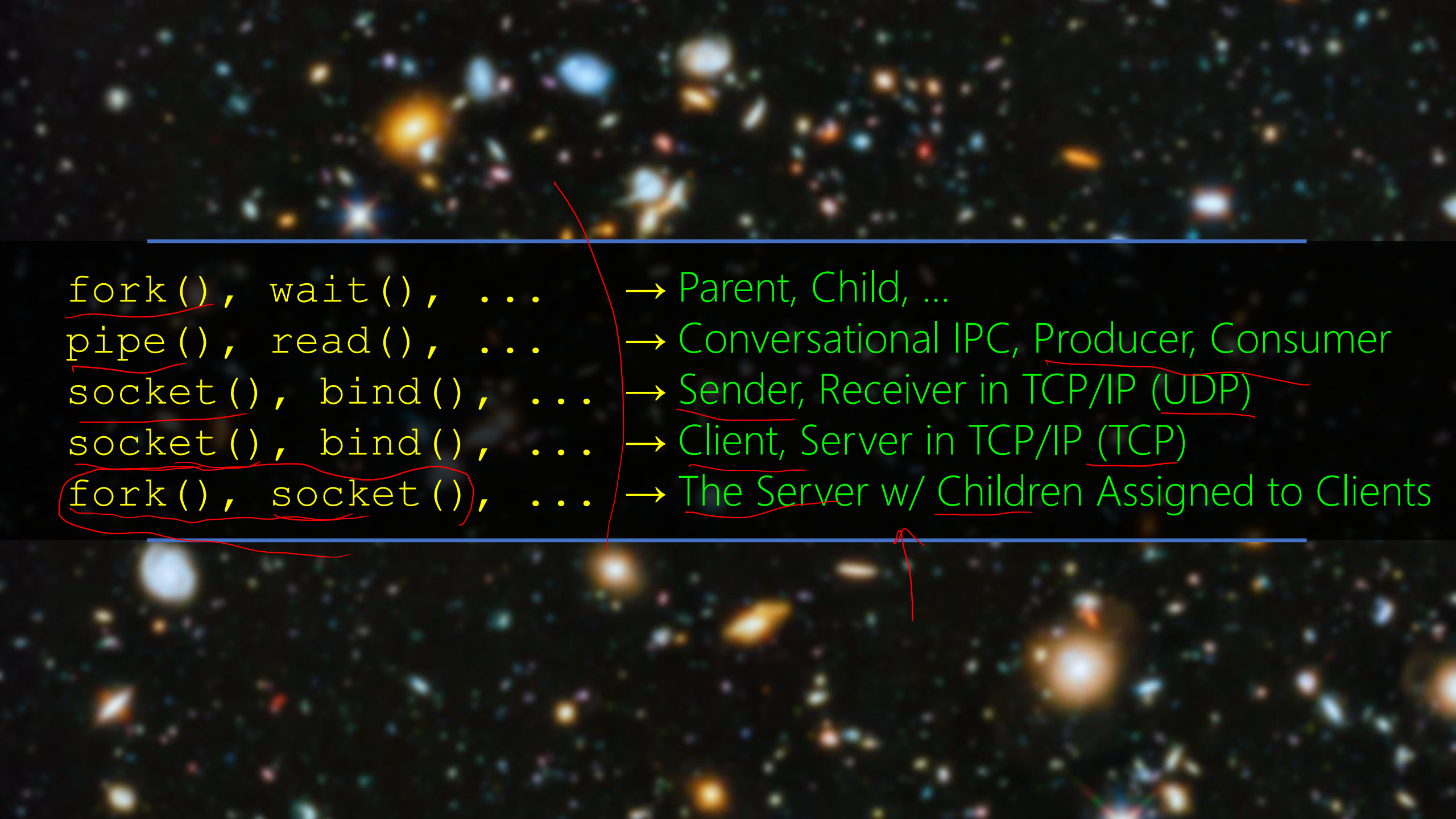
Processor



So far, ...
Technological Services by the Kernel to Application-Level Programs

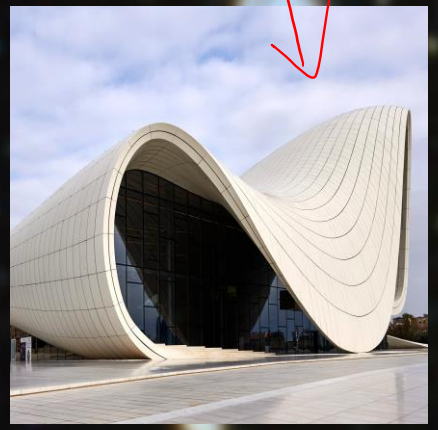
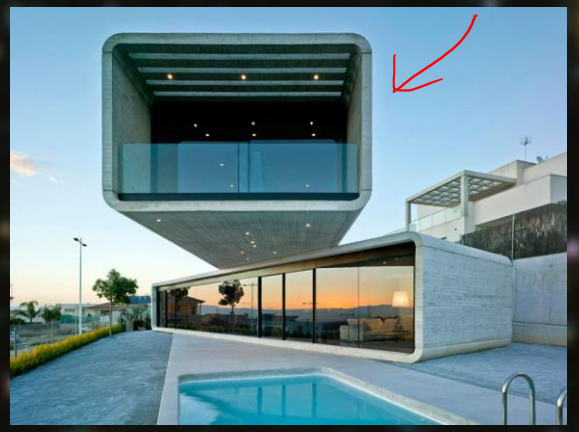
File Manager (File System): Communicating with Devices and Files
Process Manager (Process Control): Bootstrapping, Multiprocessing, New Process, IPC (Signal, Pipe, ...)
Network Manager: Network IPC, Socket, TCP/IP (UDP, TCP)
Memory Manager: Virtual Memory, Paging, ... → COMP3300: Operating Systems
Device Manager: Device Driver, ... → COMP2660: Microprocessor Programming, COMP????: Assembly Language

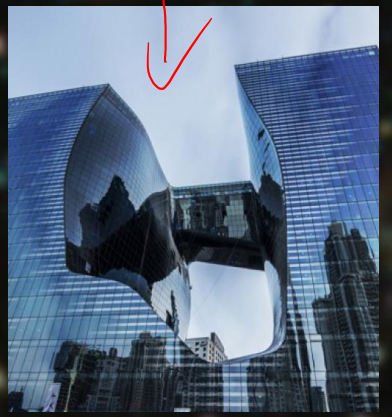

Design Patterns for Application-Level Programs


fork(), wait(), ...
→ Parent, Child, ...
pipe(), read(), ...
→ Conversational IPC, Producer, Consumer
socket(), bind(), ...
→ Sender, Receiver in TCP/IP (UDP)
socket(), bind(), ...
→ Client, Server in TCP/IP (TCP)
fork(), socket(), ...
→ The Server w/ Children Assigned to Clients

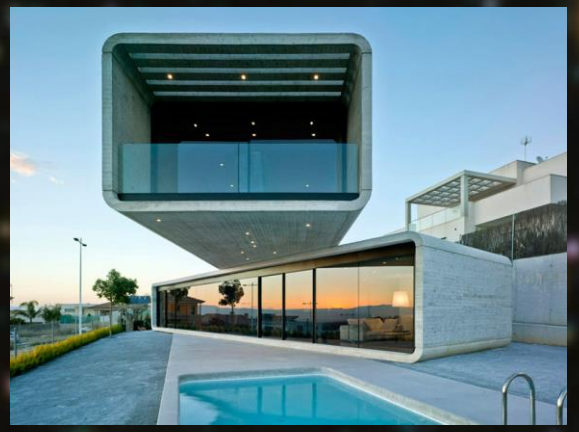
You Know the Technology (Material)
You Know the Design Patterns

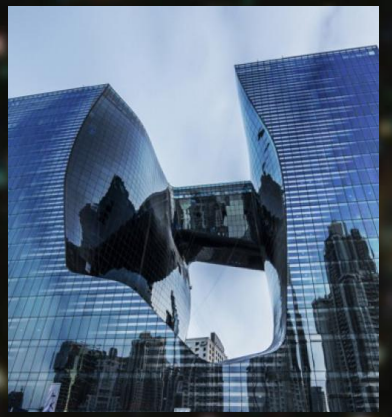
Now, Write a Program

Software Programmer
Software Designer
Software Architect

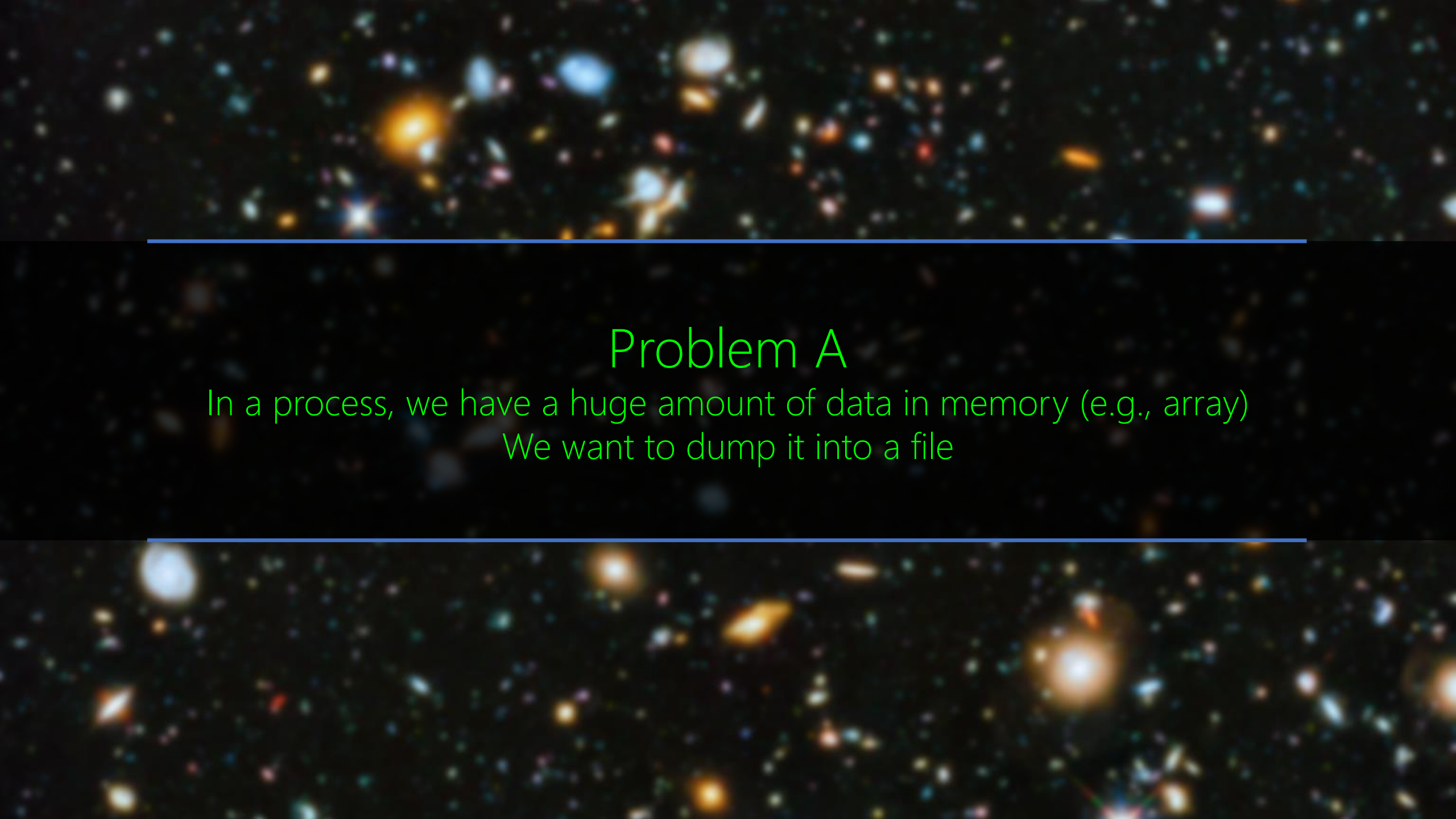




KARIM
RASHID
About Karim

Karim Rashid | I am a Design Pervert

https://www.youtube.com/watch?v=ovjpszjQHN4



Problem A

In a process, we have a huge amount of data in memory (e.g., array)

We want to dump it into a file

Design I (Synchronous)

`open()` or `creat()` → `write()`

+: simple, I get the full mark

-: the program gets stuck in writing to file and cannot do anything else, poor design!

```
Mac:~ hosseinfani$ cp /Users/hosseinfani/Desktop/1968RosemarysBaby.mp4 /Users/hosseinfani/
```

Wait up until finishes!

No progress report.

Can you pause/resume it?

Design II (Asynchronous)

`open()` or `creat()` → `fork()` → `write()` by child

Is it worth it?

+: the program continues with other stuff while the child does the writing, I get the full mark plus "good job!"

-: you need to know what forking is, handle parent/child part of codes.

```
-Mac:~ hosseinfani$ pv /Users/hosseinfani/Desktop/1968RosemarysBaby.mp4 > /Users/hosseinfani/copy.mp4
:04 [ 404MiB/s] [=========================================================>] 100%
```

Progress report!

https://www.ivarch.com/programs/pv.shtml

Design III (Asynchronous)

`open()` or `creat()` → N × `fork()` → `write(part_i)` by child_i

Is it worth it?

+: the program continues with other stuff while the children do the writing, very fast, I get the full mark plus "WoW!"

-: you need to know what forking is, handle parent/children part of codes, splitting the data, …

https://www.internetdownloadmanager.com/video02/idm.ogv

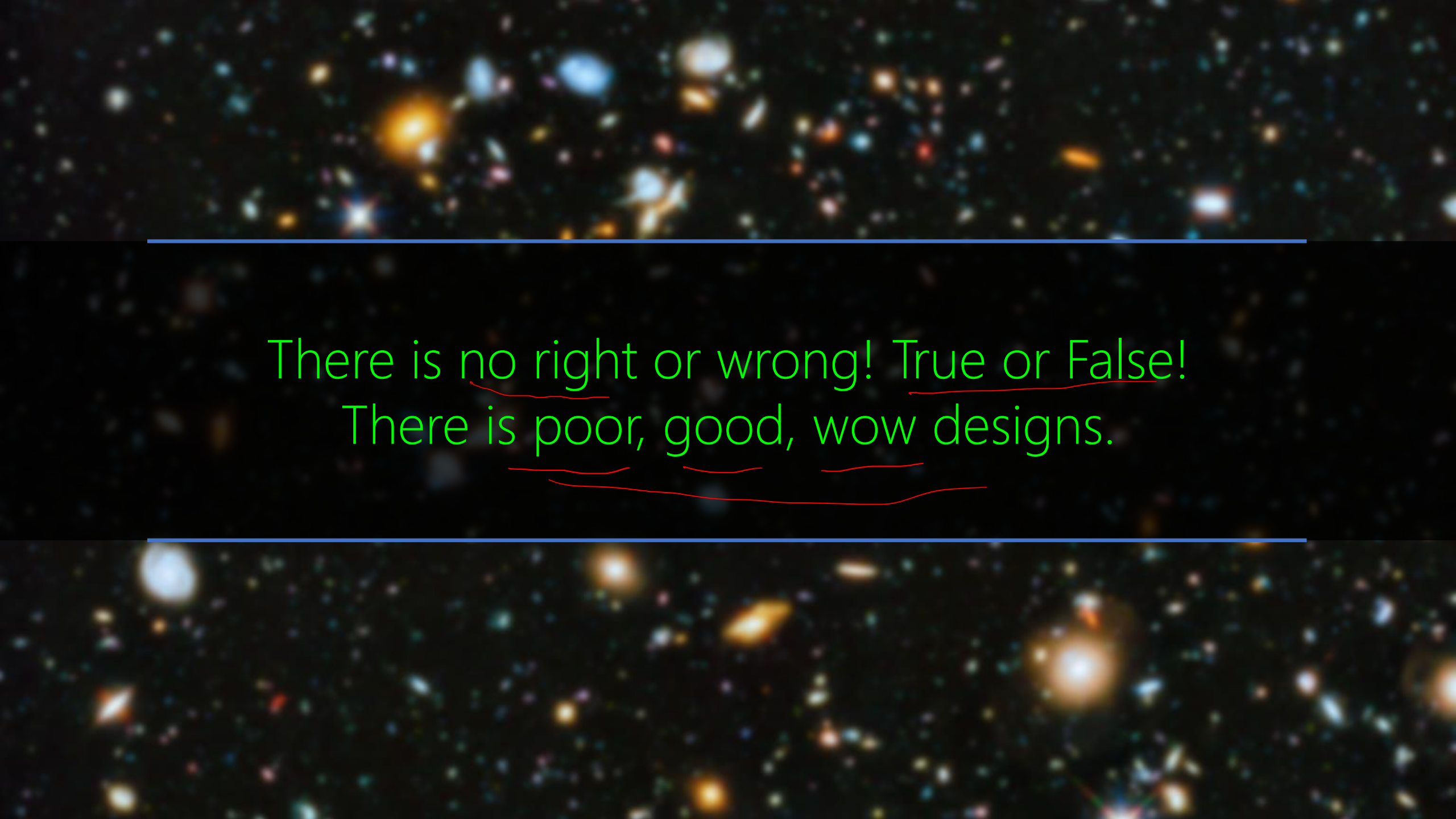
Status Receiving data...

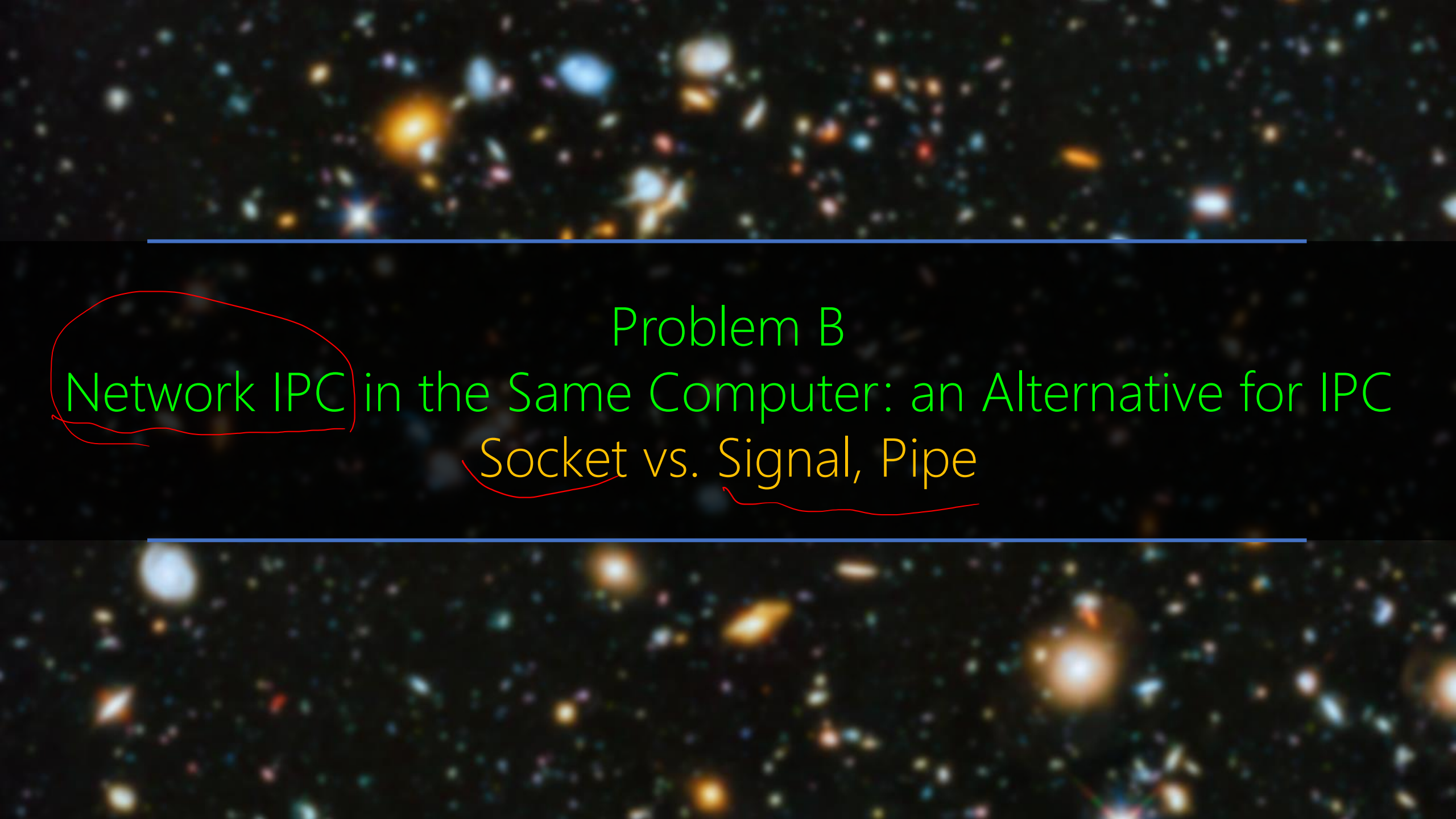
File size	17,745 MB
Downloaded	4,360 MB (24,57 %)
Transfer rate	1,916 MB/sec
Time left	6 sec
Resume capability	Yes

<< Hide details | Pause | Cancel

Start positions and download progress by connections

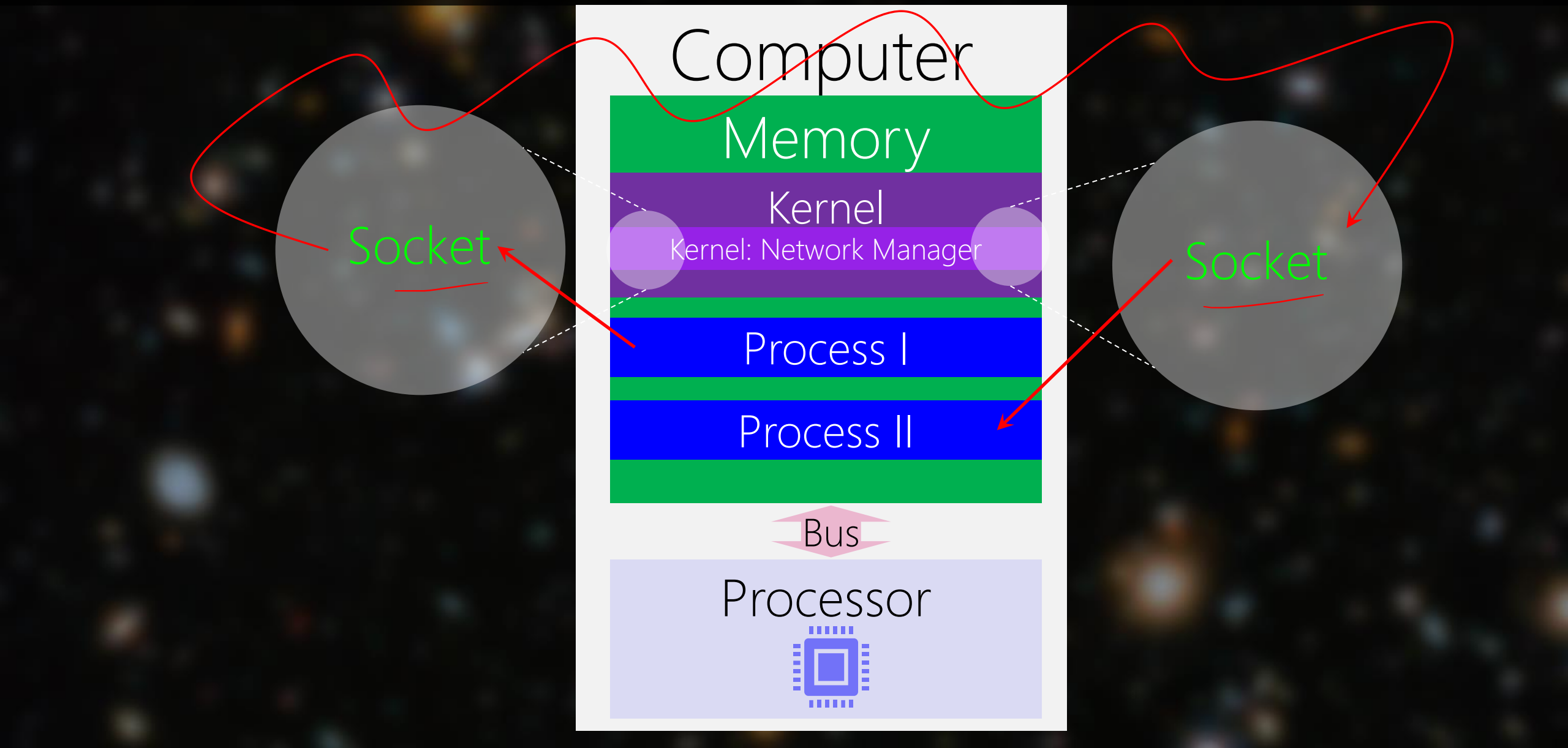
N.	Downloaded	Info
1	656,000 KB	Send GET...
2	789,937 KB	Receiving data...
3	640,625 KB	Receiving data...


There is no right or wrong! True or False!
There is poor, good, wow designs.



Problem B
Network IPC in the Same Computer: an Alternative for IPC
Socket vs. Signal, Pipe

Network IPC on the Same Computer → IPC



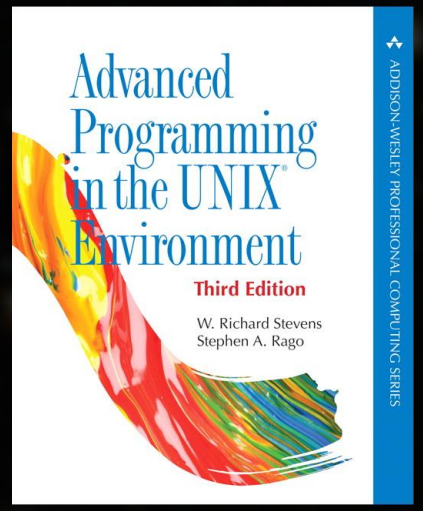
Problem

Network IPC in the Same Computer: an Alternative for IPC

Authority vs. Responsibility

+: The parent should not carry the tasks of child,
what else?

-: The parent doesn't control giving birth to/kill the child(ren), no shared file/pipe/mmap,
what else?



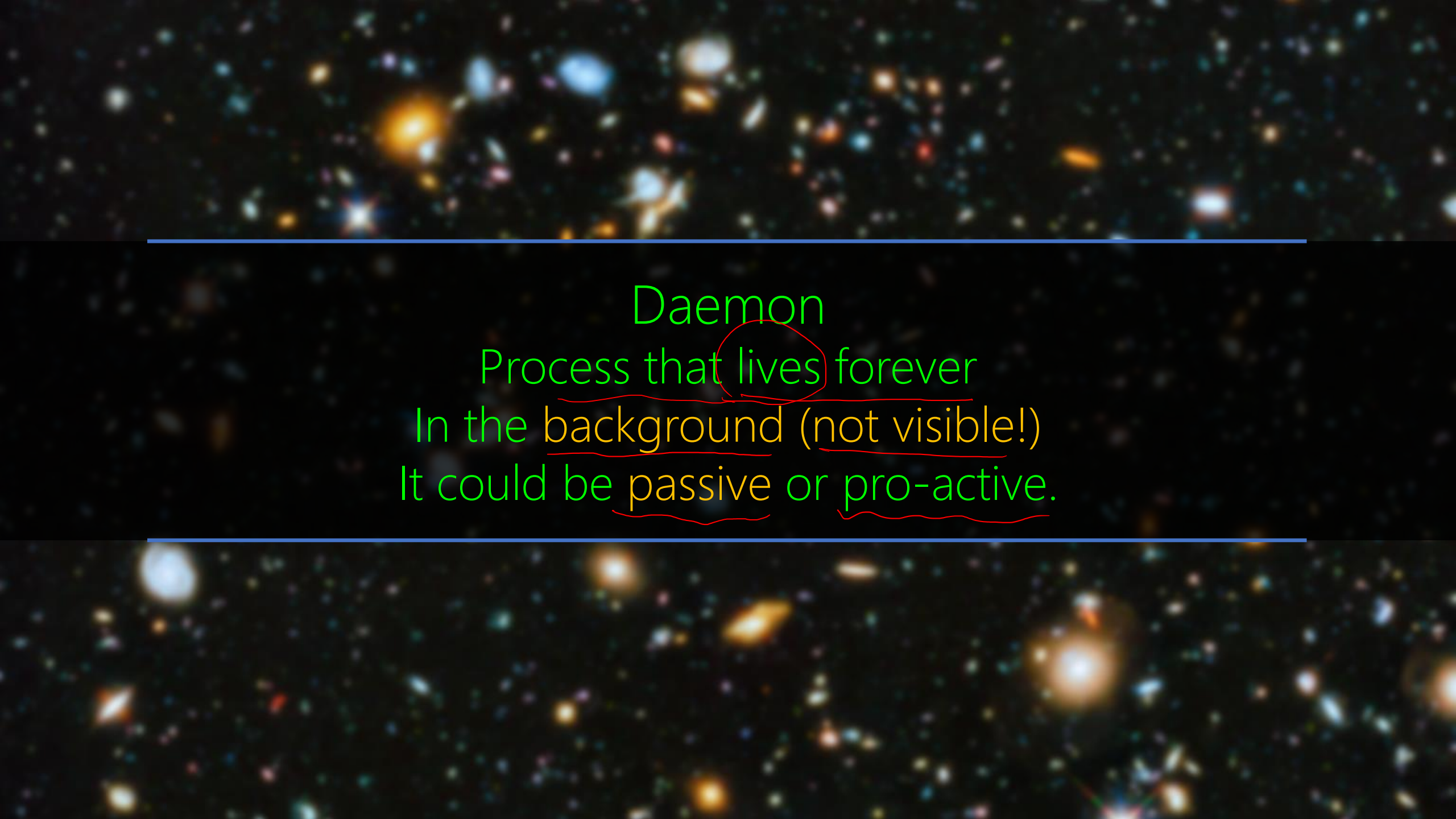
Chapter 13: Daemon Processes

Rosemary's Baby (1968), Roman Polanski
https://www.youtube.com/watch?v=I8cTs2s4dI0

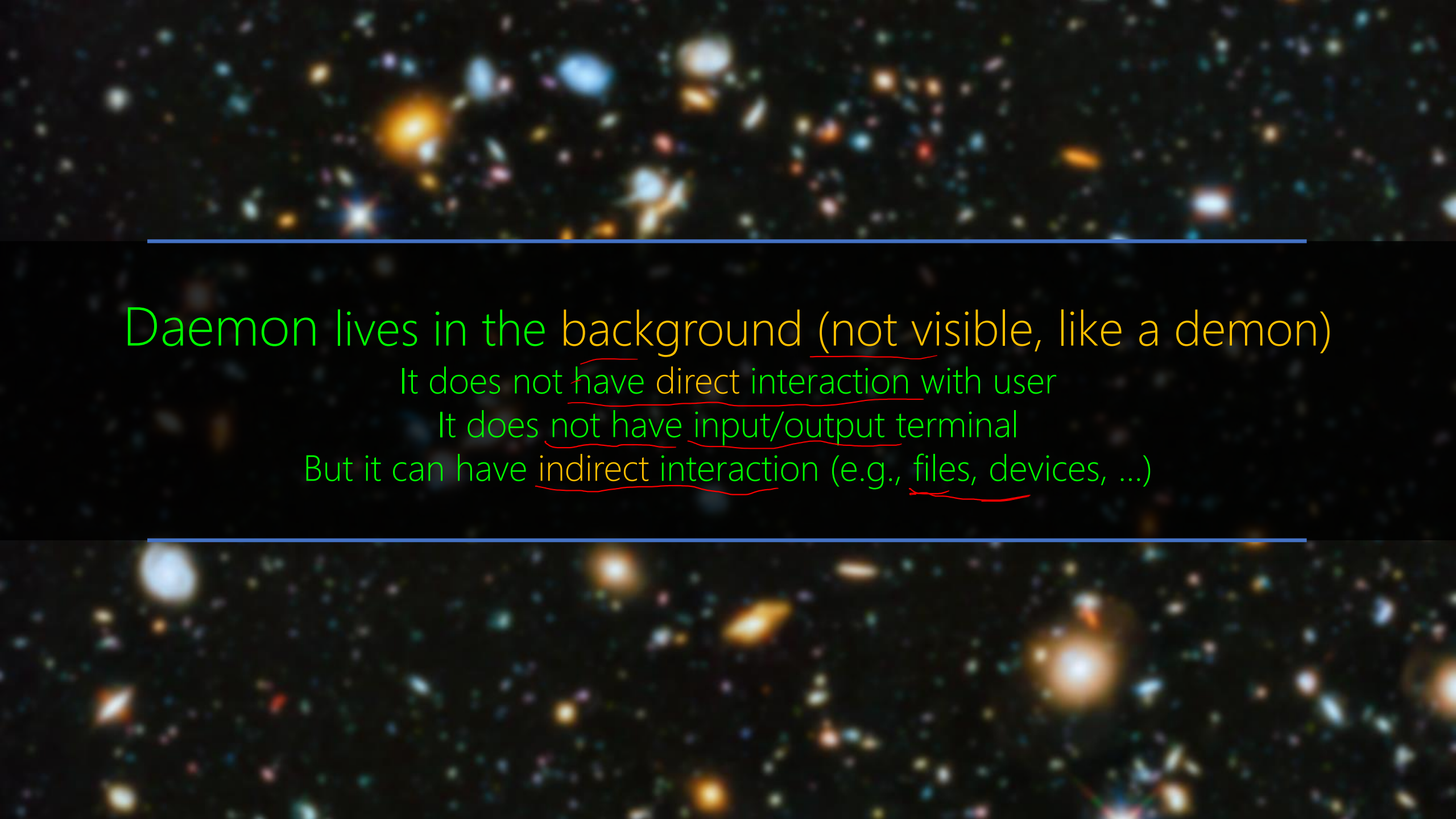
Daemon

/ˈdiːmən/ DEE-mən: Evil Spirit
/ˈdeɪmən/ DAY-mən: in Greek mythology's a supernatural being working in the background

https://en.wikipedia.org/wiki/Daemon_(computing)


Daemon
Process that lives forever
In the background (not visible!)
It could be passive or pro-active.


Daemon lives forever
It resides in memory till the computer is off
Memory-resident process

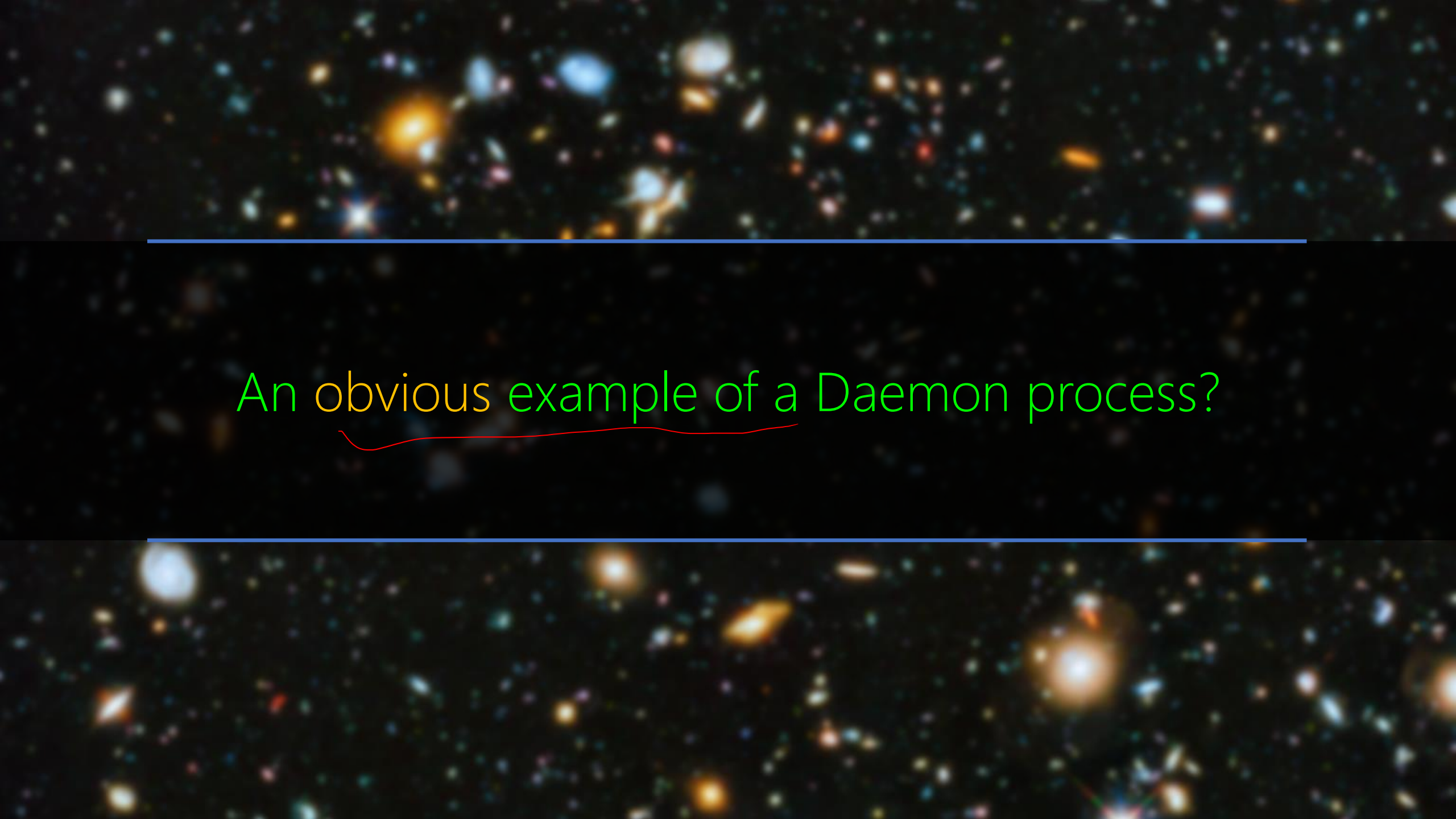

Daemon lives in the background (not visible, like a demon)
It does not have direct interaction with user
It does not have input/output terminal
But it can have indirect interaction (e.g., files, devices, ...)

Daemon can be passive

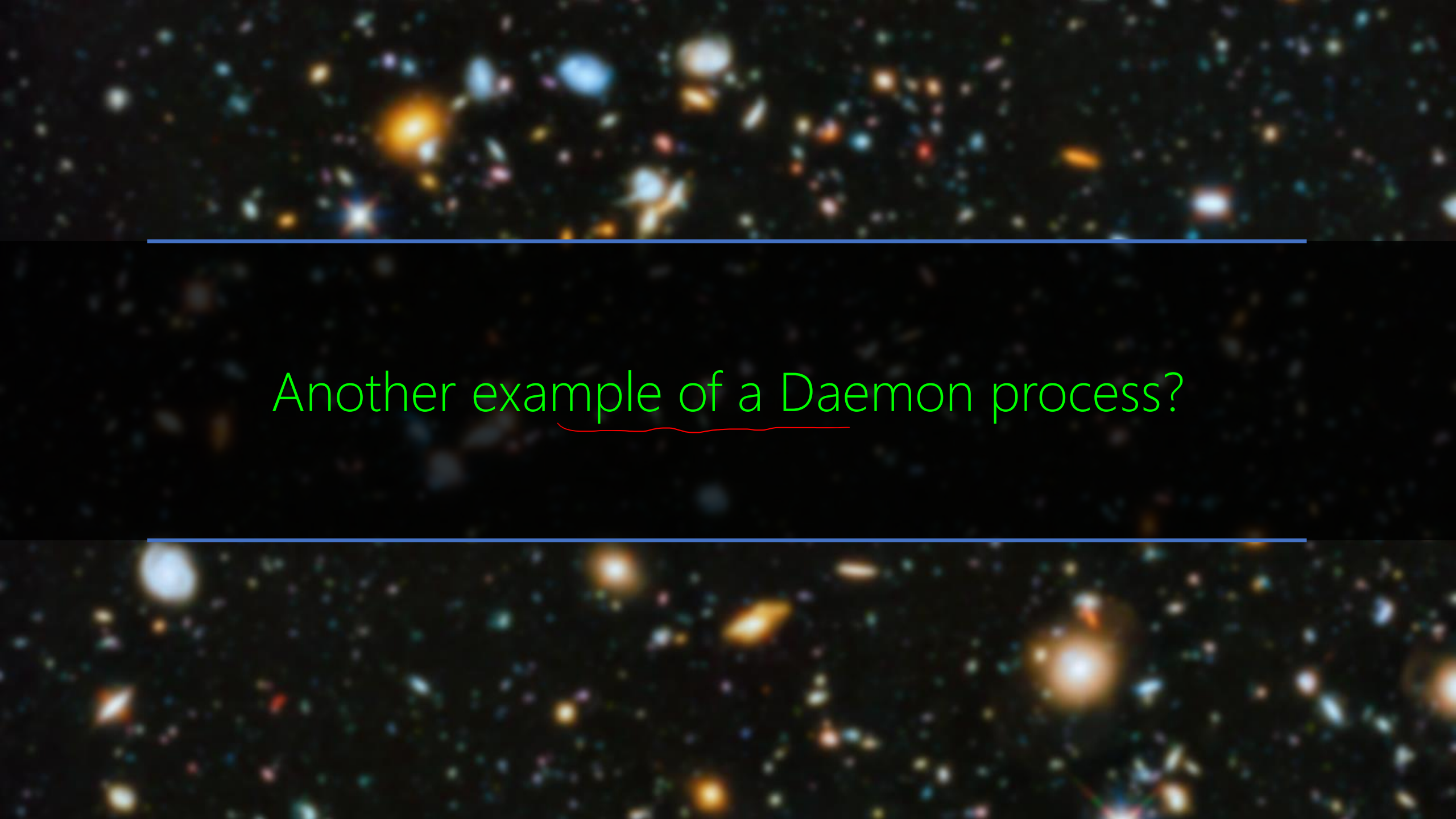
Waits for other processes to initiate a communication

Then it provides a service


Daemon can be pro-active
It does many tasks
E.g., routinely report/log the processor activity in a file
Keylogger!!

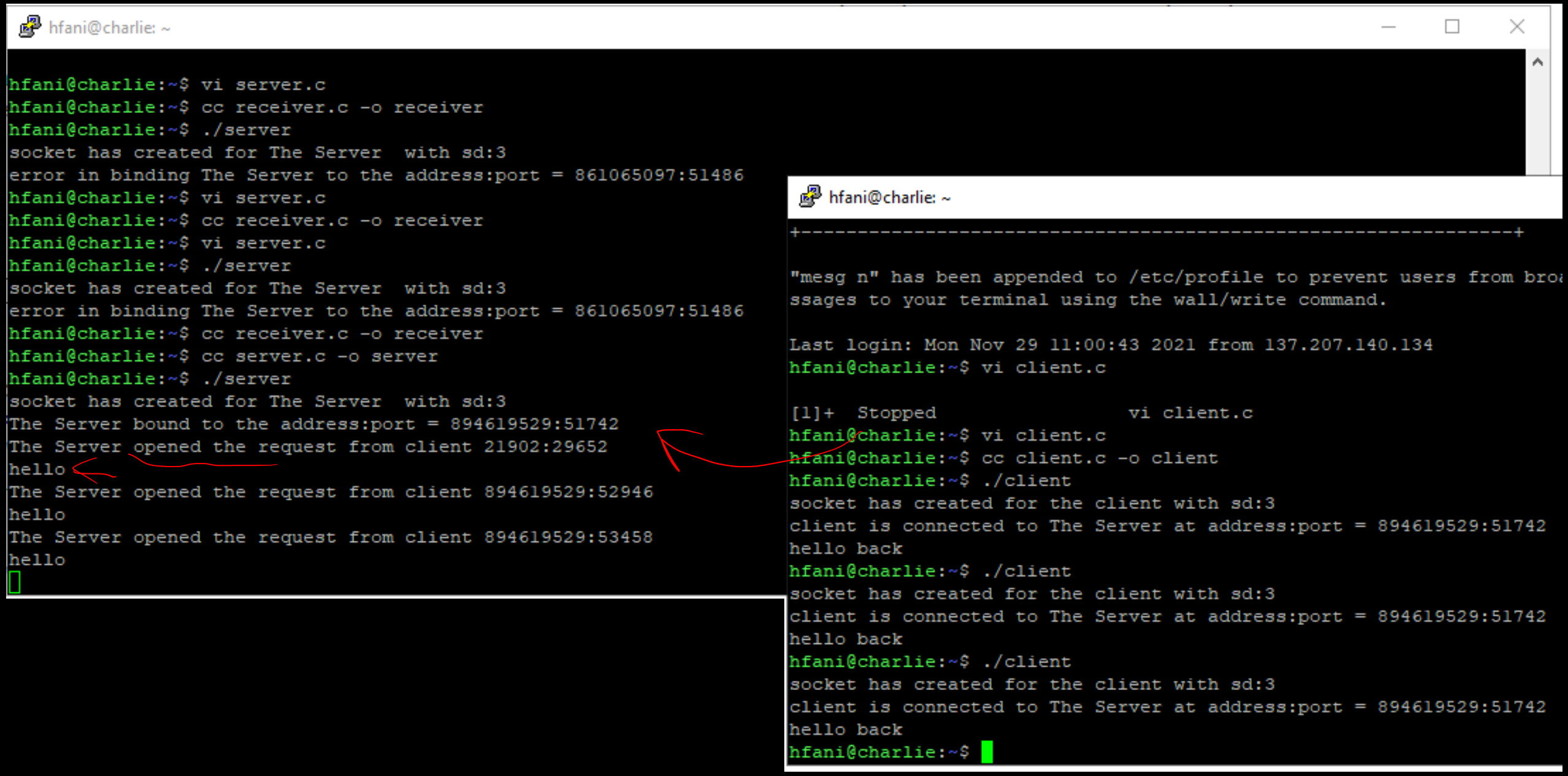

An obvious example of a Daemon process?


The Kernel (Why?)
init()


Another example of a Daemon process?

The Server (Network IPC)

```
hfani@alpha:~$ ./server
socket has created for The Server  with sd:3
The Server bound to the address:port = 861065097:51742
```


hfani@charlie: ~
hfani@charlie:~$ vi server.c
hfani@charlie:~$ cc receiver.c -o receiver
hfani@charlie:~$ ./server
socket has created for The Server with sd:3
error in binding The Server to the address:port = 861065097:51486
hfani@charlie:~$ vi server.c
hfani@charlie:~$ cc receiver.c -o receiver
hfani@charlie:~$ vi server.c
hfani@charlie:~$ ./server
socket has created for The Server with sd:3
error in binding The Server to the address:port = 861065097:51486
hfani@charlie:~$ cc receiver.c -o receiver
hfani@charlie:~$ cc server.c -o server
hfani@charlie:~$ ./server
socket has created for The Server with sd:3
The Server bound to the address:port = 894619529:51742
The Server opened the request from client 21902:29652
hello
The Server opened the request from client 894619529:52946
hello
The Server opened the request from client 894619529:53458
hello
hfani@charlie: ~
+----------+
"mesg n" has been appended to /etc/profile to prevent users from bro
ssages to your terminal using the wall/write command.
Last login: Mon Nov 29 11:00:43 2021 from 137.207.140.134
hfani@charlie:~$ vi client.c
[1]+ Stopped vi client.c
hfani@charlie:~$ vi client.c
hfani@charlie:~$ cc client.c -o client
hfani@charlie:~$ ./client
socket has created for the client with sd:3
client is connected to The Server at address:port = 894619529:51742
hello back
hfani@charlie:~$ ./client
socket has created for the client with sd:3
client is connected to The Server at address:port = 894619529:51742
hello back
hfani@charlie:~$ ./client
socket has created for the client with sd:3
client is connected to The Server at address:port = 894619529:51742
hello back
hfani@charlie:~$

The Server (Network IPC)

There is a small problem tho! What?

```
hfani@alpha:~$ ./server
socket has created for The Server  with sd:3
The Server bound to the address:port = 861065097:51742
```

The Server (Network IPC)

It is not in the background!

It's interacting with user. The user cannot use this terminal anymore.

```
hfani@alpha:~$ ./server
socket has created for The Server  with sd:3
The Server bound to the address:port = 861065097:51742
```

The Server (Network IPC)

Putting it in the background, alive, responding to clients

But not visible

```
hfani@alpha:~$ ./server
socket has created for The Server  with sd:3
The Server bound to the address:port = 861065097:51742
```


Daemonize
Putting it in the background, alive, but not visible
How?
Think about it as a designer, considering all previous technologies, patterns, ...

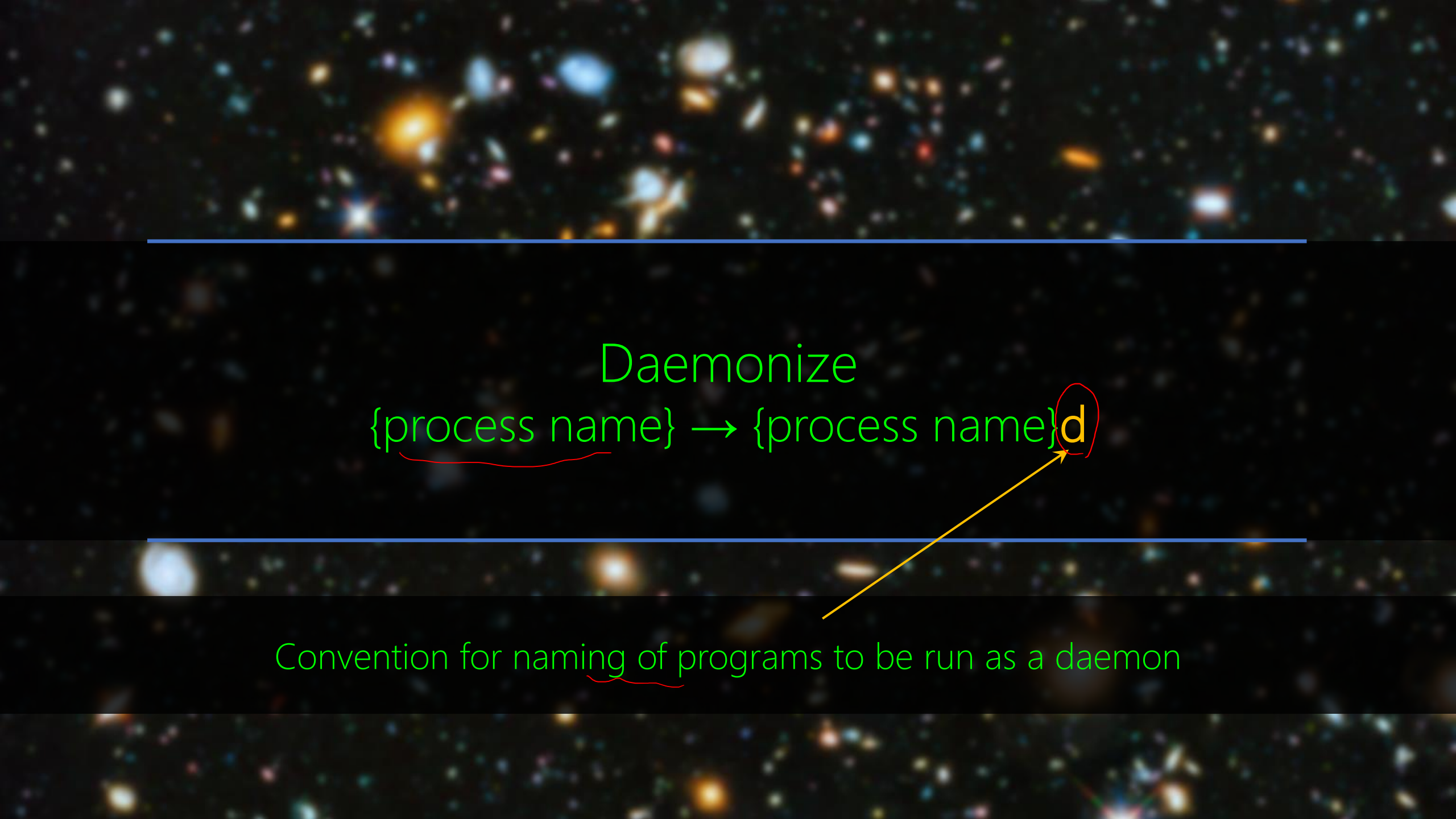
Rosemary's Baby (1968), Roman Polanski

The Server (server.c)

```c
#include <stdlib.h>

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include <stdio.h>
#include <string.h>
int main(void){
        int domain = AF_INET;//Network Protocol: TCP/IP
        int type = SOCK_STREAM;//Connection-Oriented
        int protocol = 0;//Default transport: TCP for Internet connection-oriented

        int server_sd;//socket descriptor ~= file descriptor
        server_sd = socket(domain, type, protocol);
        if (server_sd == -1){
                printf("error in creating socket for The Server!\n");
                exit(1);
        }
        else
                printf("socket has created for The Server  with sd:%d\n", server_sd);

        //Binding to an address is a must for The Server!
        struct in_addr server_sin_address;
        server_sin_address.s_addr = inet_addr("137.207.82.51");//nslookup `hostname`
        int server_sin_port = htons(7882);//larger than 1024

        struct sockaddr_in server_sin;
        server_sin.sin_family = domain;
        server_sin.sin_addr = server_sin_address;
        server_sin.sin_port = server_sin_port;
        int result = bind(server_sd, (struct sockaddr *) &server_sin, sizeof(server_sin));
        if (result == -1){
                printf("error in binding The Server to the address:port = %d:%d\n", server_sin.sin_addr, server_sin.sin_port);
                exit(1);
        }
        else
                printf("The Server bound to the address:port = %d:%d\n", server_sin.sin_addr, server_sin.sin_port);

        //The Server ready to receive calls (up to 5 calls. More are rejected!)
        if (listen(server_sd, 5) < 0) {
                perror("The Server's listening failed!\n");
                exit(1);
        }

        struct sockaddr_in client_sin;//I want to know who send the message
        int client_sin_len;
        while(1)
        {
                result = accept(server_sd, (struct sockaddr *) &client_sin, &client_sin_len);
                if (result == -1){
                        printf("error in opening the request from client %d:%d !\n", client_sin.sin_addr, client_sin.sin_port);
                        //exit(1);Do not exit. Go for the next client call
                }
                else
                        printf("The Server opened the request from client %d:%d\n", client_sin.sin_addr, client_sin.sin_port);

                char msg[10];
                recv(result, msg, 10, 0);
                printf("%s\n", msg);
                send(result, "hello back", 11, 0);
        }
```

The Server (server.c)

```c
#include <stdlib.h>

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include <stdio.h>
#include <string.h>
int main(void){
        int domain = AF_INET;//Network Protocol: TCP/IP
        int type = SOCK_STREAM;//Connection-Oriented
        int protocol = 0;//Default transport: TCP for Internet connection-oriented

        int server_sd;//socket descriptor ~= file descriptor
        server_sd = socket(domain, type, protocol);
        if (server_sd == -1){
                printf("error in creating socket for The Server!\n");
                exit(1);
        }
        else
                printf("socket has created for The Server  with sd:%d\n", server_sd);

        //Binding to an address is a must for The Server!
        struct in_addr server_sin_address;
        server_sin_address.s_addr = inet_addr("137.207.82.51");//nslookup `hostname`
        int server_sin_port = htons(7882);//larger than 1024

        struct sockaddr_in server_sin;
        server_sin.sin_family = domain;
        server_sin.sin_addr = server_sin_address;
        server_sin.sin_port = server_sin_port;
        int result = bind(server_sd, (struct sockaddr *) &server_sin, sizeof(server_sin));
        if (result == -1){
                printf("error in binding The Server to the address:port = %d:%d\n", server_sin.sin_addr, server_sin.sin_port);
                exit(1);
        }
        else
                printf("The Server bound to the address:port = %d:%d\n", server_sin.sin_addr, server_sin.sin_port);

        //The Server ready to receive calls (up to 5 calls. More are rejected!)
        if (listen(server_sd, 5) < 0) {
                perror("The Server's listening failed!\n");
                exit(1);
        }

        struct sockaddr_in client_sin;//I want to know who send the message
        int client_sin_len;
        while(1)
        {
                result = accept(server_sd, (struct sockaddr *) &client_sin, &client_sin_len);
                if (result == -1){
                        printf("error in opening the request from client %d:%d !\n", client_sin.sin_addr, client_sin.sin_port);
                        //exit(1);Do not exit. Go for the next client call
                }
                else
                        printf("The Server opened the request from client %d:%d\n", client_sin.sin_addr, client_sin.sin_port);

                char msg[10];
                recv(result, msg, 10, 0);
                printf("%s\n", msg);
                send(result, "hello back", 11, 0);
        }
```


Daemonize
{process name} → {process name}d
Convention for naming of programs to be run as a daemon

The Server as a daemon (serverd.c)

```c
#include <stdlib.h>

#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include <stdio.h>
#include <string.h>
int main(void){
        int child_pid = fork();
        if(child_pid > 0)
                exit(0);
        else if (child_pid == -1){
                printf("cannot create a child for The Server!\n");
                exit(1);
        }
        //else if child_pid == 0 //the child
```

```c
        int domain = AF_INET;//address protocol family
        int type = SOCK_STREAM;//Connection-Oriented
        int protocol = 0;//Default transport: TCP for Internet connection-oriented

        int server_sd;//socket descriptor == file descriptor
        server_sd = socket(domain, type, protocol);
        if (server_sd == -1){
                printf("error in creating socket for The Server!\n");
                exit(1);
        }
        else
                printf("socket has created for The Server  with sd:%d\n", server_sd);

        //Binding to an address is a must for The Server!
        struct in_addr server_sin_address;
        server_sin_address.s_addr = inet_addr("137.207.82.51");//nslookup `hostname`
        int server_sin_port = htons(7882);//larger than 1024

        struct sockaddr_in server_sin;
        server_sin.sin_family = domain;
        server_sin.sin_addr = server_sin_address;
        server_sin.sin_port = server_sin_port;
        int result = bind(server_sd, (struct sockaddr *) &server_sin, sizeof(server_sin));
        if (result == -1){
                printf("error in binding The Server to the address:port = %d:%d\n", server_sin.sin_addr, server_sin.sin_port);
                exit(1);
        }
        else
                printf("The Server bound to the address:port = %d:%d\n", server_sin.sin_addr, server_sin.sin_port);

        //The Server ready to receive calls (up to 5 calls. More are rejected!)
        if (listen(server_sd, 5) < 0) {
                perror("The Server's listening failed!\n");
                exit(1);
        }

        struct sockaddr_in client_sin;//I want to know who send the message
        int client_sin_len;
        while(1)
        {
                result = accept(server_sd, (struct sockaddr *) &client_sin, &client_sin_len);
                if (result == -1){
                        printf("error in opening the request from client %d:%d !\n", client_sin.sin_addr, client_sin.sin_port);
                        //exit(1);Do not exit. Go for the next client call
                }
                else
                        printf("The Server opened the request from client %d:%d\n", client_sin.sin_addr, client_sin.sin_port);

                char msg[10];
                recv(result, msg, 10, 0);
                printf("%s\n", msg);
                send(result, "hello back", 11, 0);
        }
```

1) Wrap it for a child
2) Parent exits
3) Orphan child lives forever (Adopted by the kernel

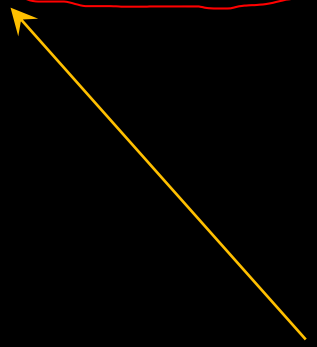
```
}
```

```
hfani@alpha:~$ ./serverd 127.0.0.1 2022
socket has created for The Server  with sd:3
The Server bound to the address:port = 861065097:51742
hfani@alpha:~$
```

It returns to the shell, ready for other commands or processes

}

```
hfani@alpha:~$ ./serverd 127.0.0.1 2022
socket has created for The Server  with sd:3
The Server bound to the address:port = 861065097:51742
hfani@alpha:~$
hfani@alpha:~$ ps
    PID TTY          TIME CMD
1568863 pts/6    00:00:00 bash
1586249 pts/6    00:00:00 serverd
1586307 pts/6    00:00:00 ps
```



It returns to the shell, ready for other commands or processes

But it is in memory, passive, waiting for clients

}

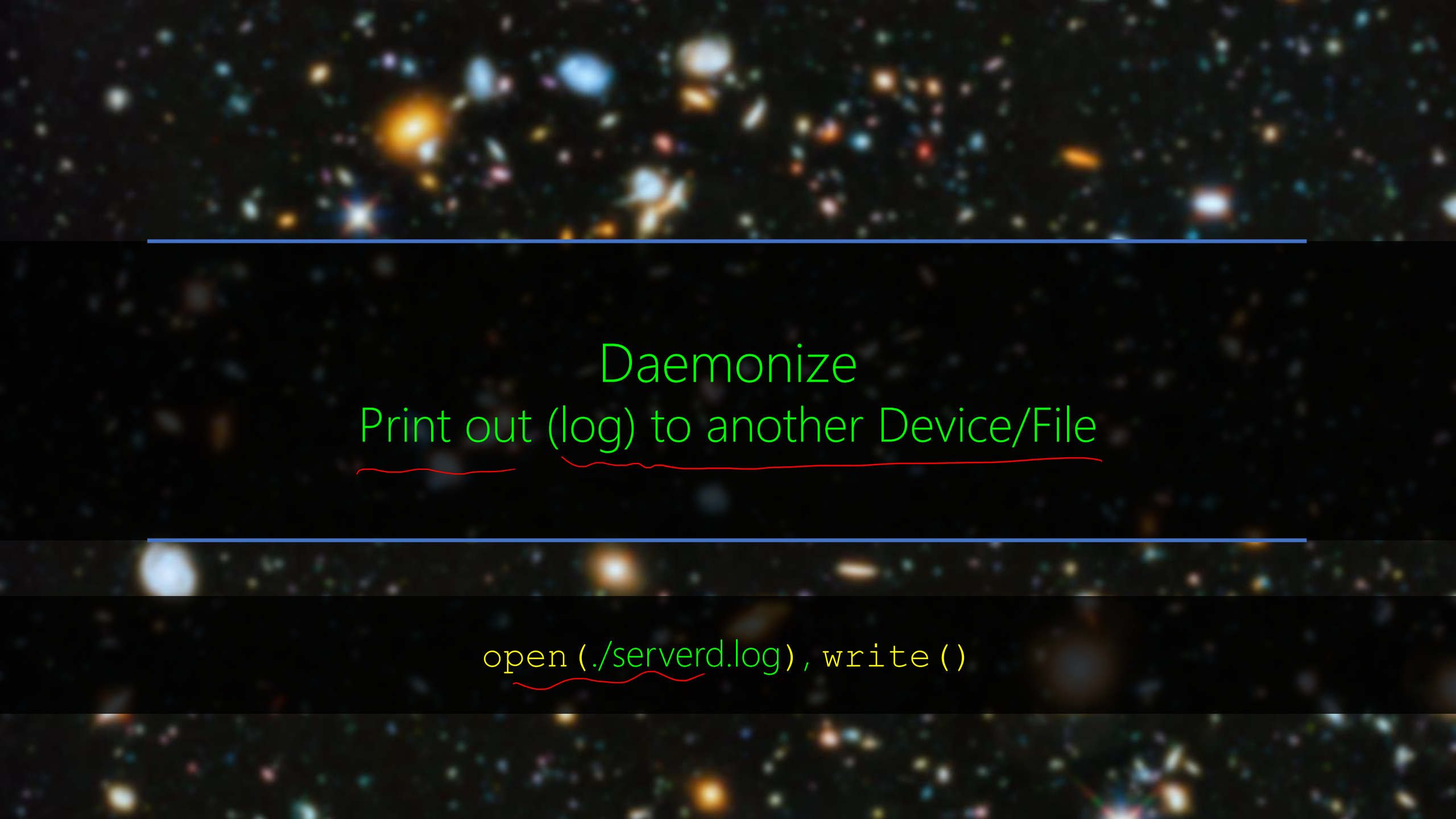
```
hfani@alpha:~$ ./client 127.0.0.1 2022
socket has created for the client with sd:3
client is connected to The Server at address:port = 861065097:51742
The Server opened the request from client 0:0
Child pid 9655: I The Server's child to handle the communication
with the client 0:0
Hey!
0I will look into Hey!
1I will look into Hey!
2I will look into Hey!
3I will look into Hey!
4I will look into Hey!
```

These lines are not printed by the client program

They are printed by The Server!

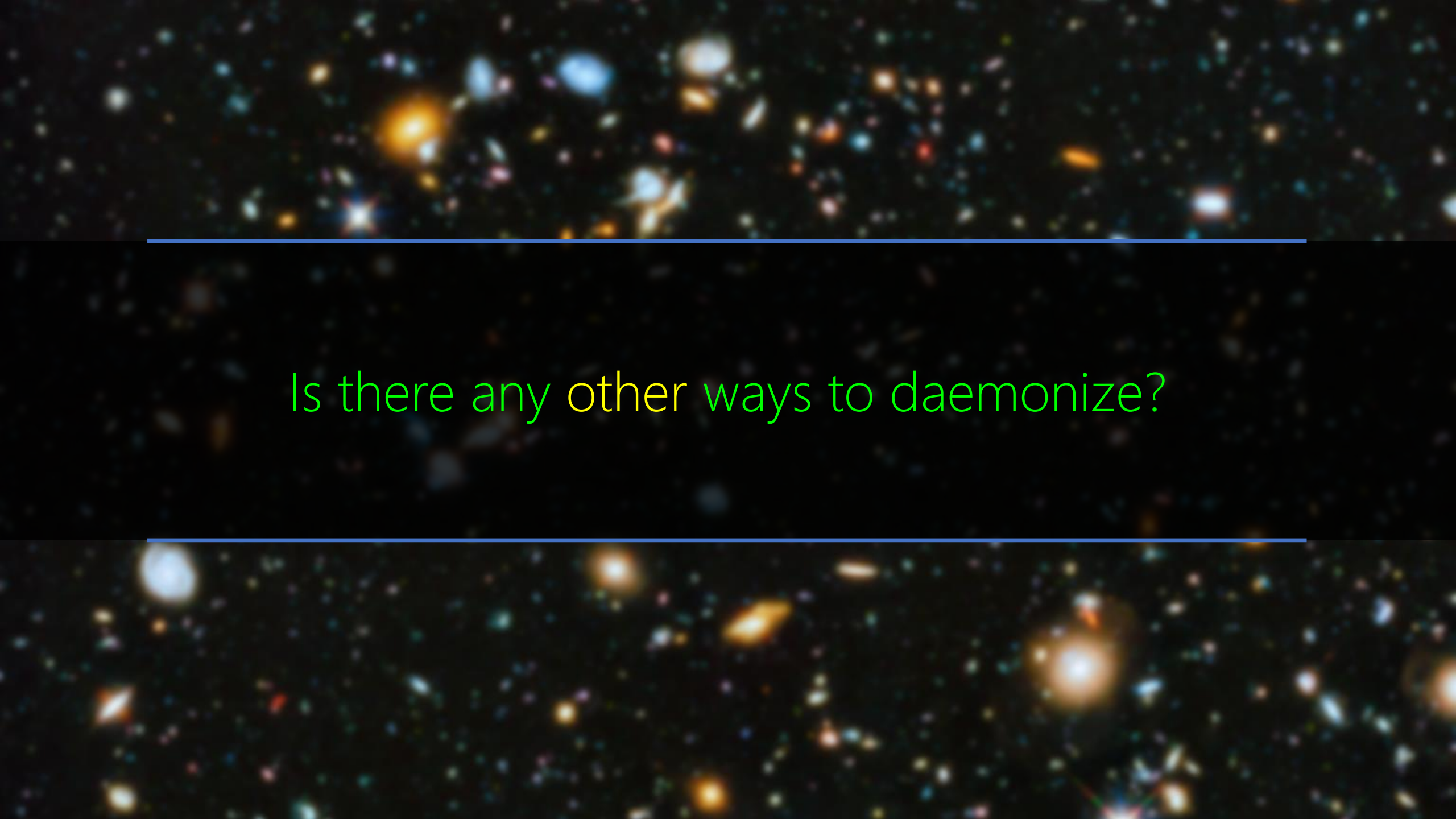
Because both are using same standard output fd=1

}


Daemonize
Print out (log) to another Device/File
open(./serverd.log), write()

Daemonize

1) Create a child and give all task to the child
2) Exit the parent
3) Any printing to a log file
4) By convention, add "d" to the program file name
5) Other advanced steps (Advanced System Programming)


Is there any other ways to daemonize?


How to end a daemon?

Rosemary's Baby (1968), Roman Polanski