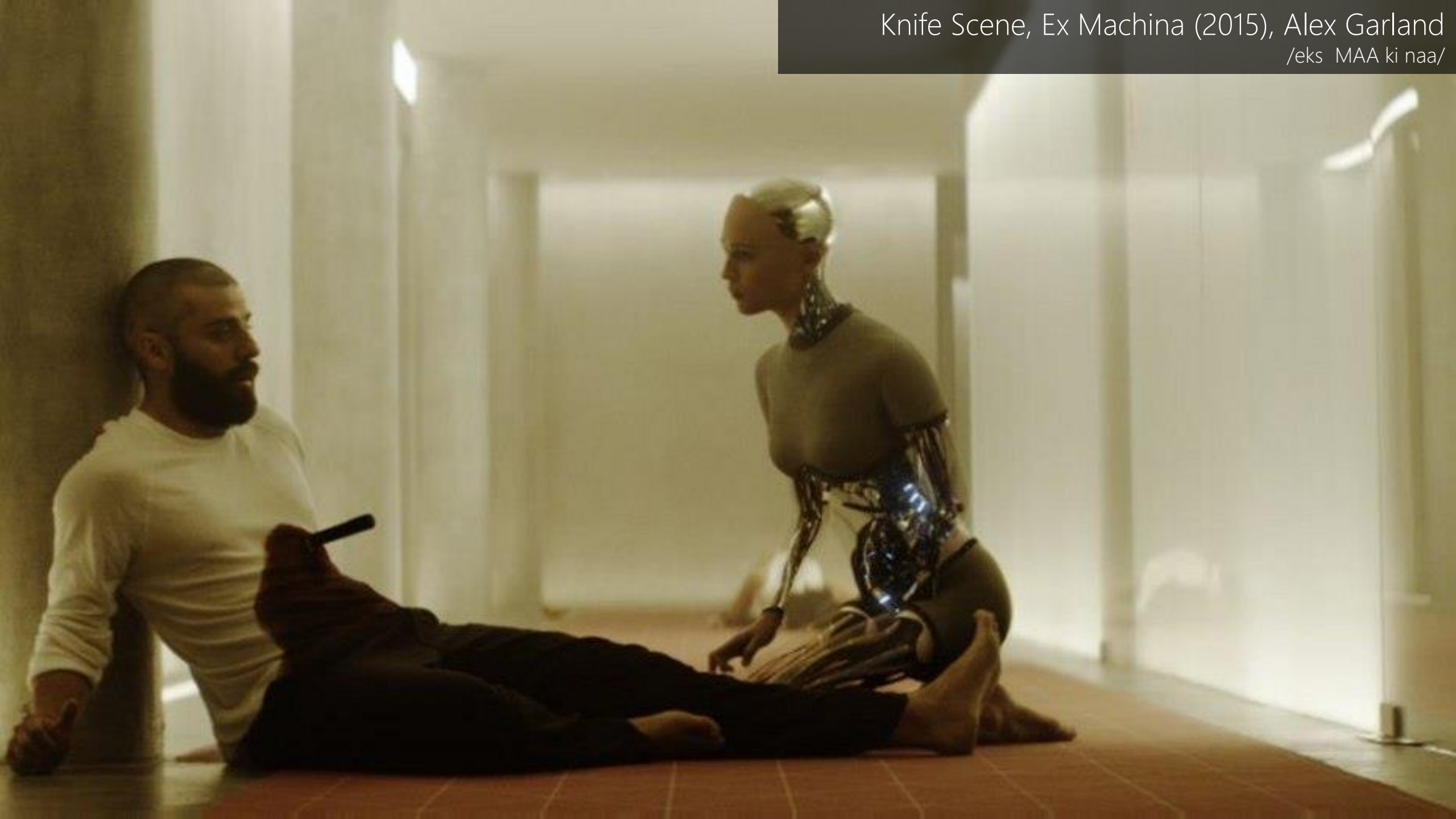
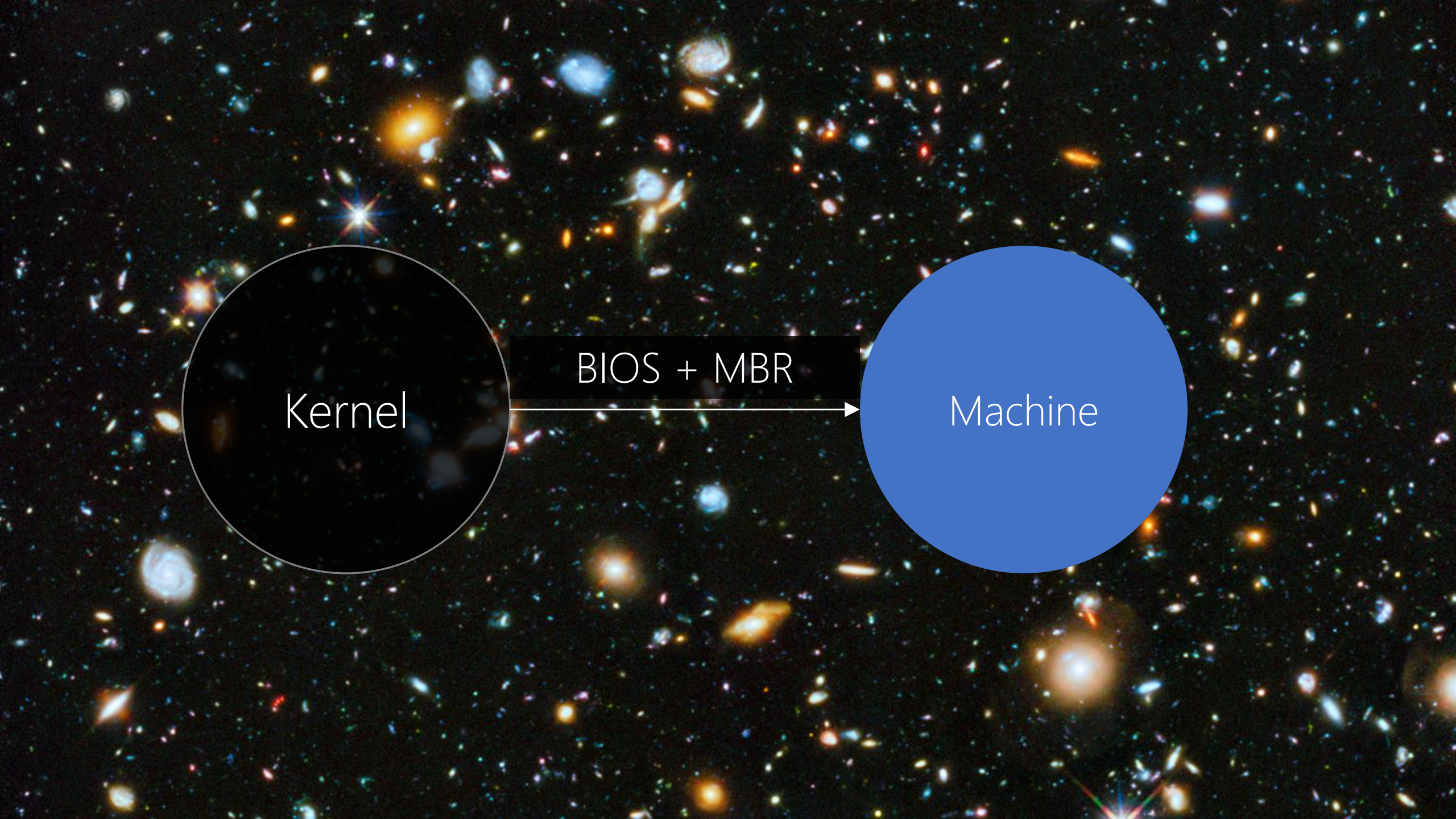
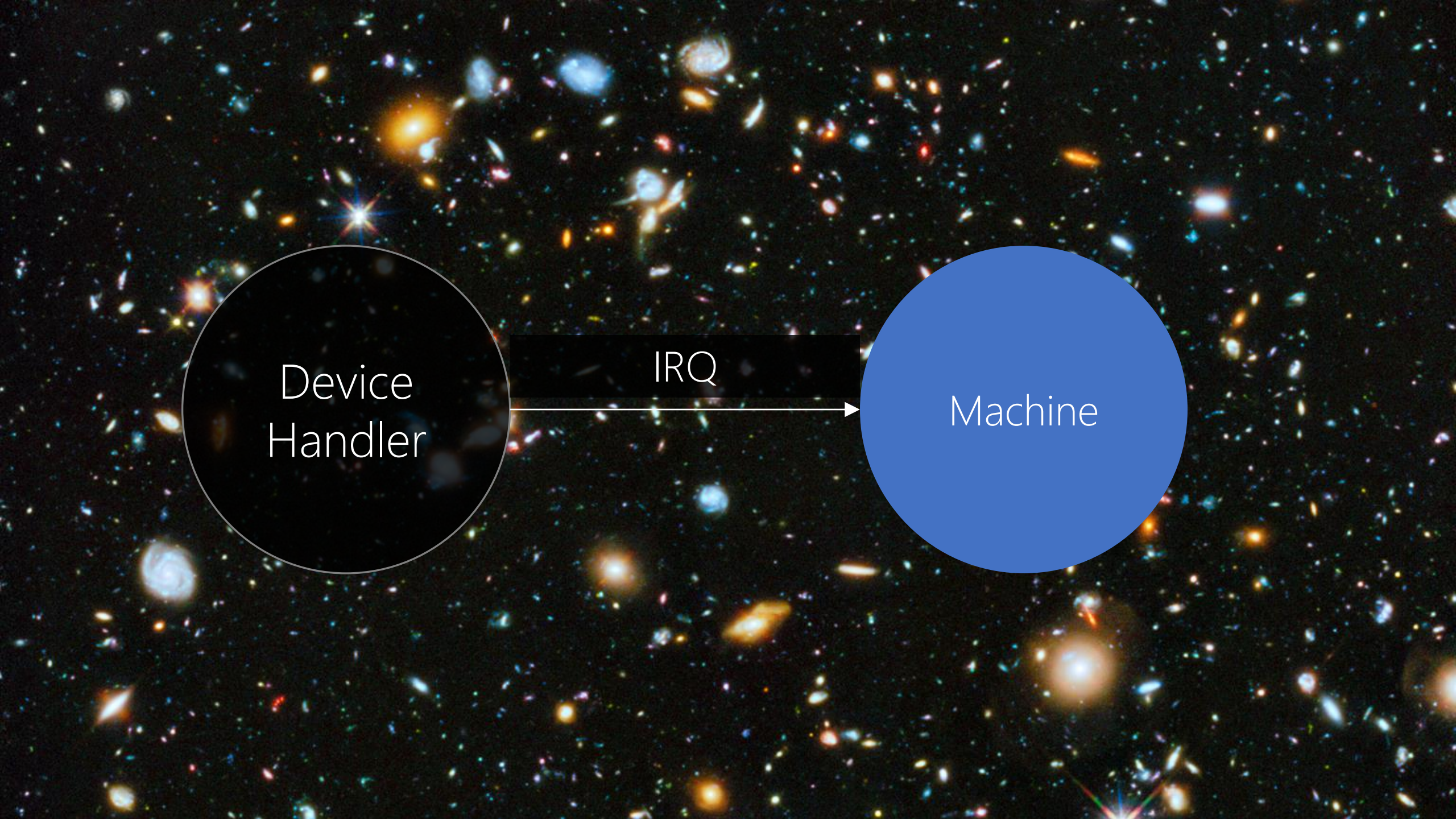


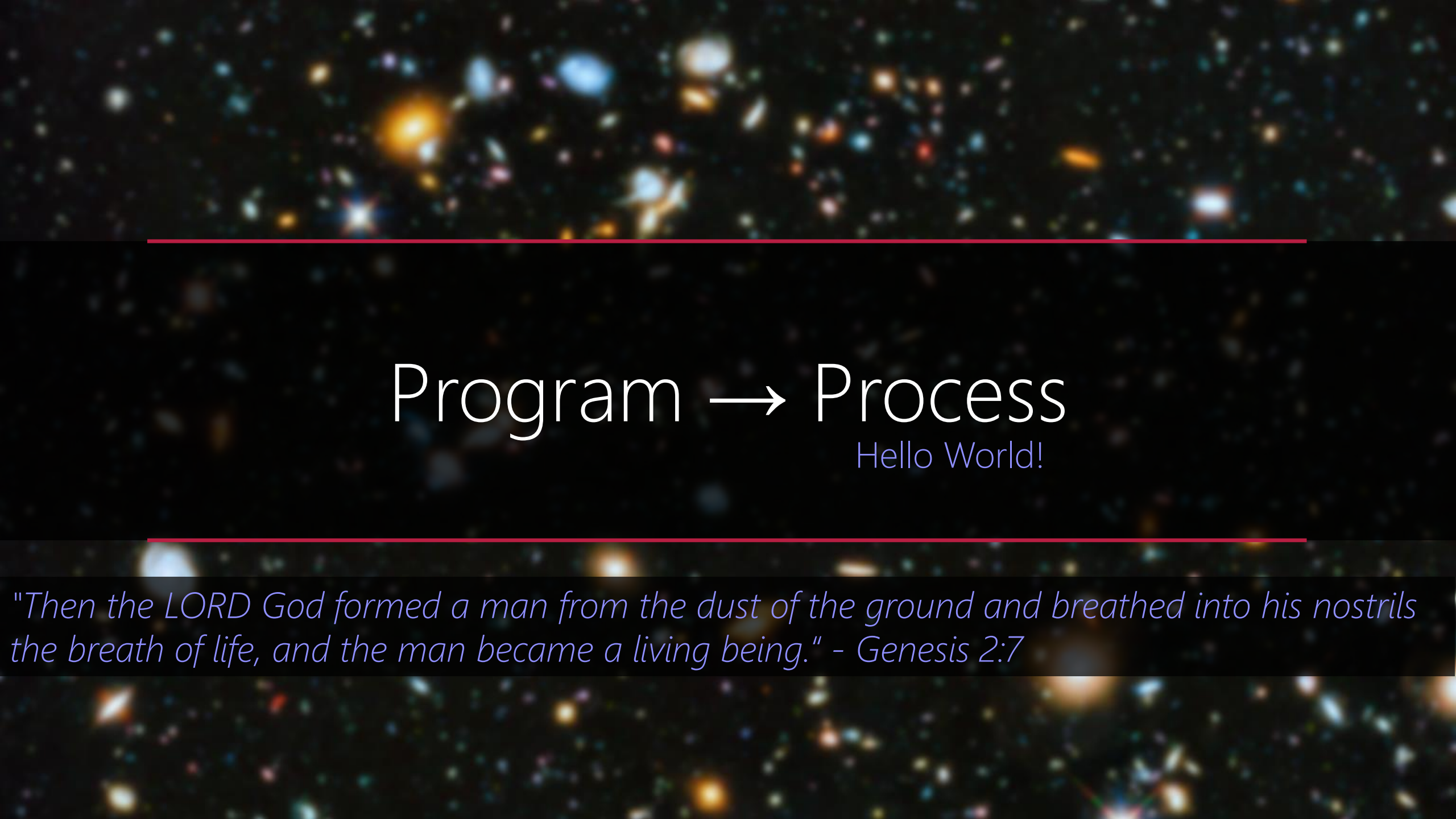
Knife Scene, Ex Machina (2015), Alex Garland

/eks MAA ki naa/


Kernel
BIOS + MBR
Machine


Device
Handler
IRQ
Machine


Ezekiel 25:17, Pulp Fiction (1994), Quentin Tarantino
Well, there's this passage I've got
memorized; sorta fits this occasion.



Program → Process

Hello World!

"Then the LORD God formed a man from the dust of the ground and breathed into his nostrils the breath of life, and the man became a living being." - Genesis 2:7

C

```
hfani@alpha:~$ vi hello.c
hfani@alpha:~$ cc main.c -S
```

```
#include <stdio.h>
void main(){
    printf("hello world!");
}
```

Compiler

Assembly

```
hfani@alpha:~$ cc main.s -c
```

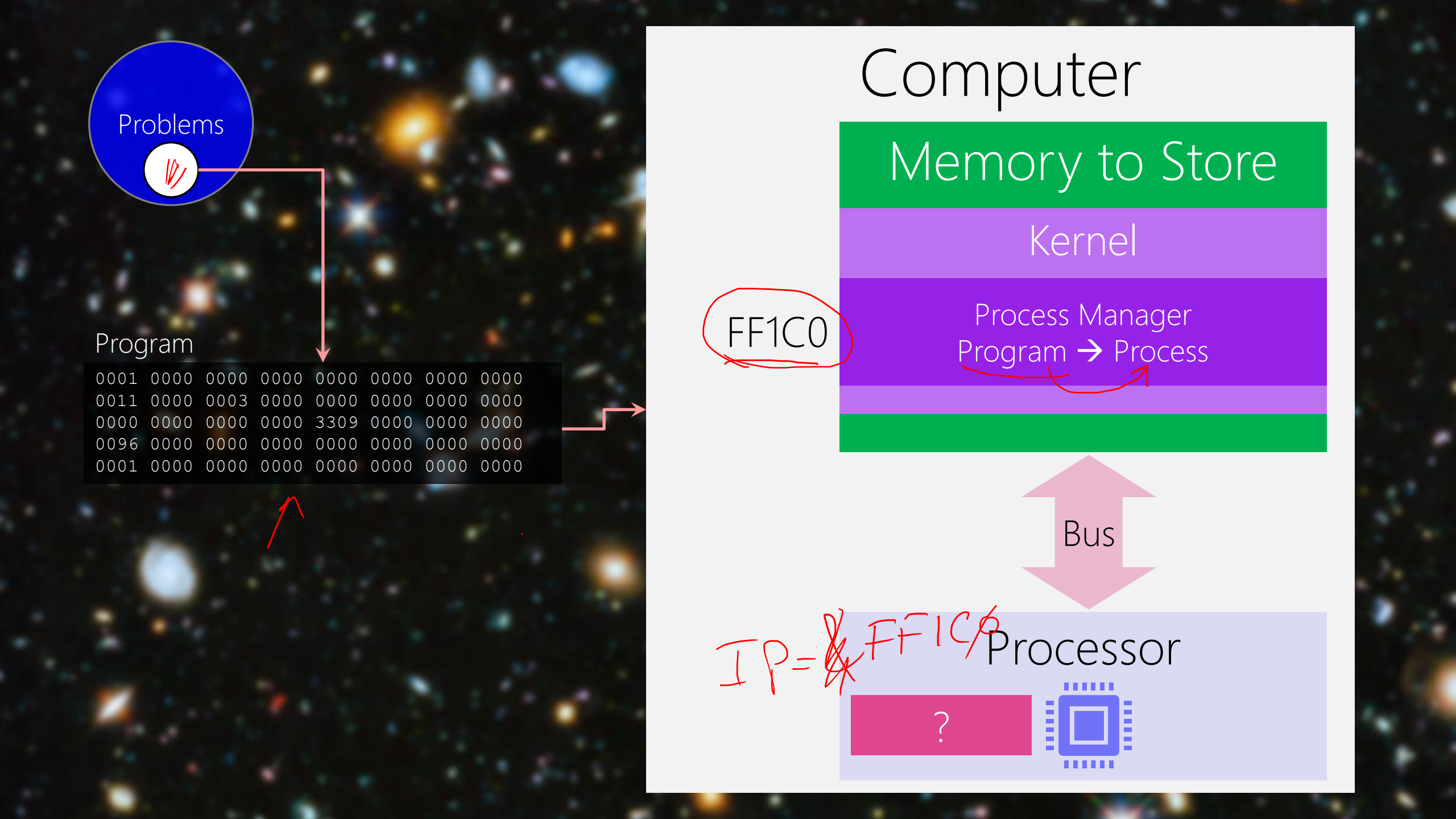
```
<printf@plt>:
jmpq        *0x3002(%rip)
pushq       $0x0
jmpq        401000 <.plt>
<main>:
push        %rbp
mov         %rsp,%rbp
lea         0xfd5(%rip),%rdi
mov         $0x0,%eax
callq       401010 <printf@plt>
nop
pop         %rbp
retq
```

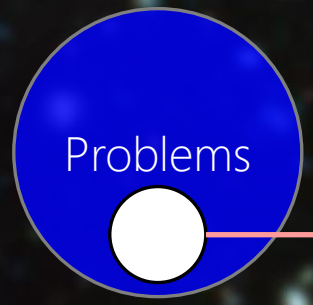
Assembler

OP Code

```
hfani@alpha:~$ cc main.o -o main
```

```
0001 0000 0000 0000 0000 0000 0000 0000
0011 0000 0003 0000 0000 0000 0000 0000
0000 0000 0000 0000 3309 0000 0000 0000
0096 0000 0000 0000 0000 0000 0000 0000
0001 0000 0000 0000 0000 0000 0000 0000
```


Problems
Program
0001 0000 0000 0000 0000 0000 0000 0000
0011 0000 0003 0000 0000 0000 0000 0000
0000 0000 0000 0000 3309 0000 0000 0000
0096 0000 0000 0000 0000 0000 0000 0000
0001 0000 0000 0000 0000 0000 0000 0000
Computer
Memory to Store
Kernel
FF1C0
Process Manager
Program → Process
Bus
IP=FF1C0
Processor
?



Program

```
0001 0000 0000 0000 0000 0000 0000 0000
0011 0000 0003 0000 0000 0000 0000 0000
0000 0000 0000 0000 3309 0000 0000 0000
0096 0000 0000 0000 0000 0000 0000 0000
0001 0000 0000 0000 0000 0000 0000 0000
```

1. Locate the program: /home/hfani/...
2. Call Process Manager for loading the program
   IP = FF1C0
3. Point IP to the first line of the program
   IP = &first opcode

Computer

FF1C0

Memory to Store

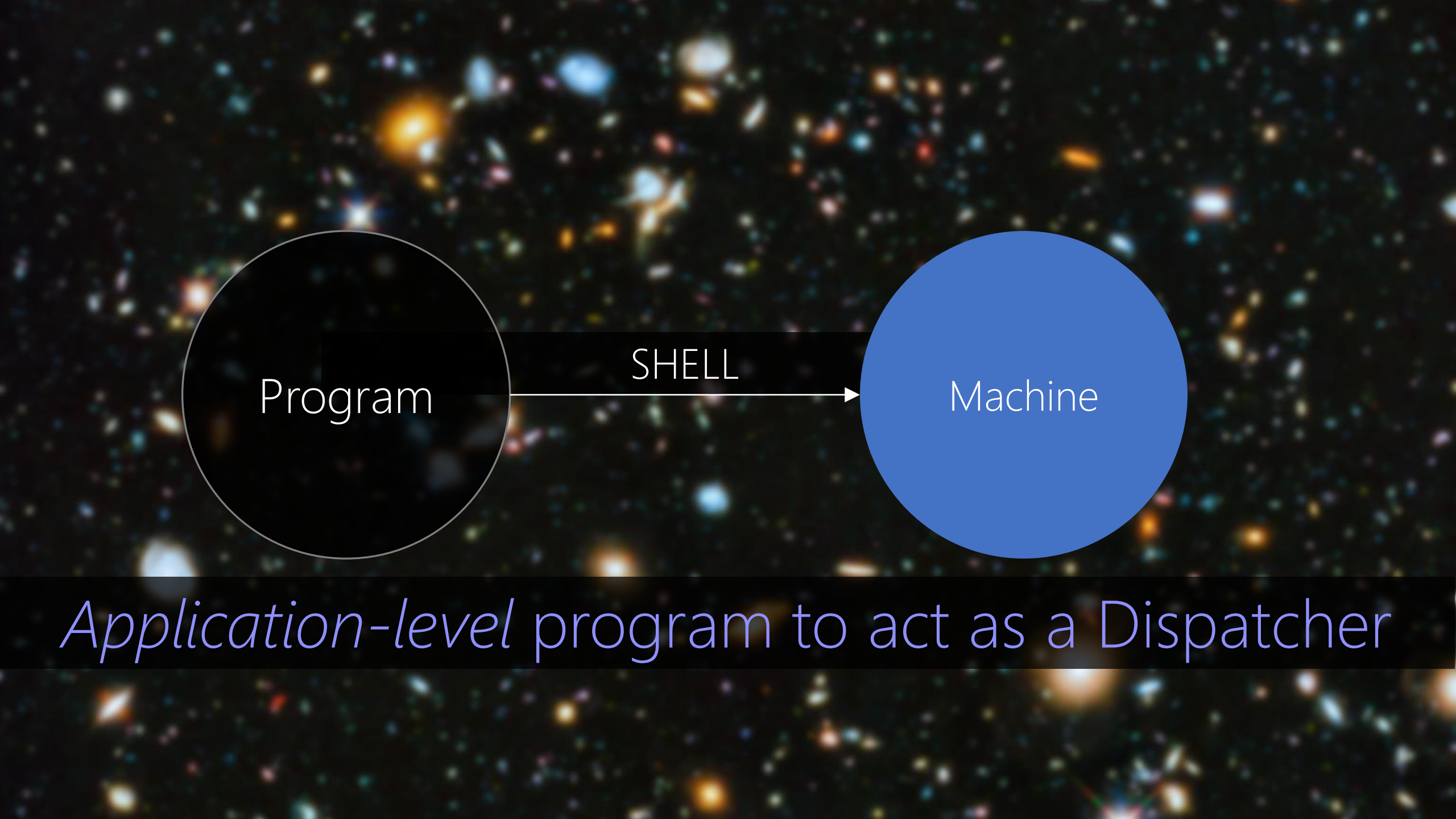
Kernel

Process Manager
Program → Process

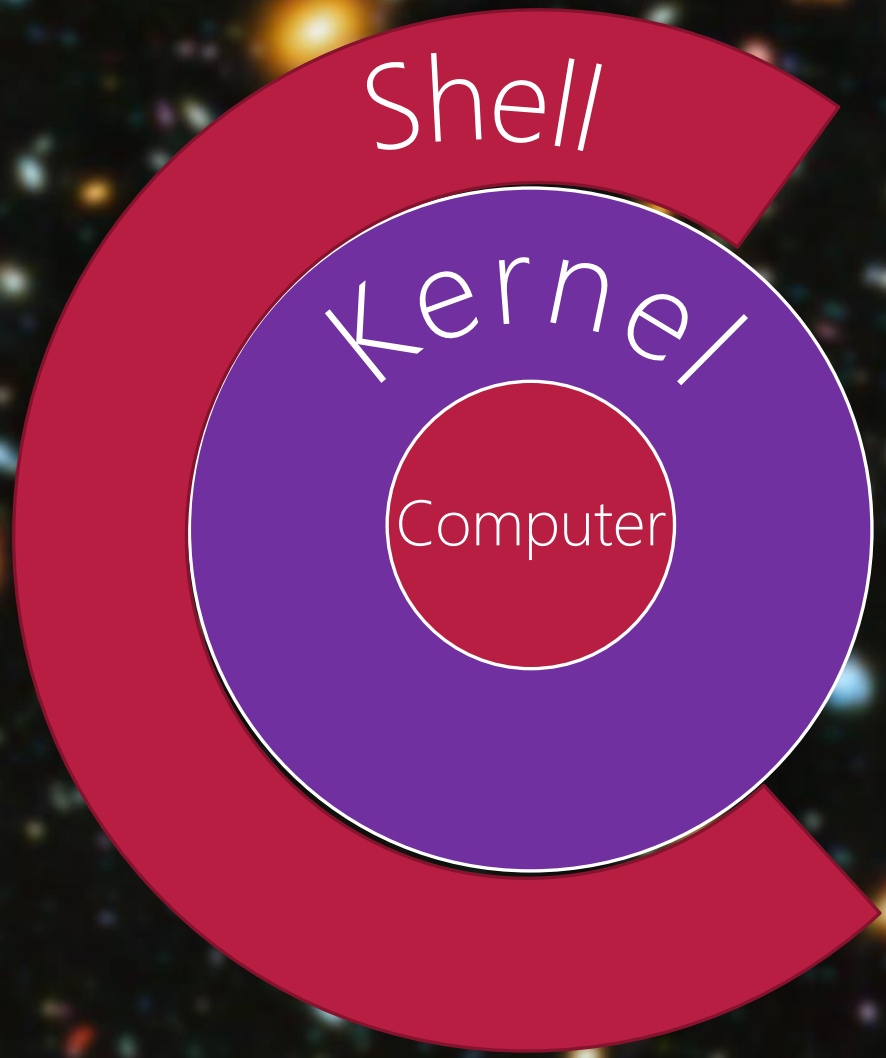
Bus

Processor

?



Application-level program to act as a Dispatcher



Computer

Memory

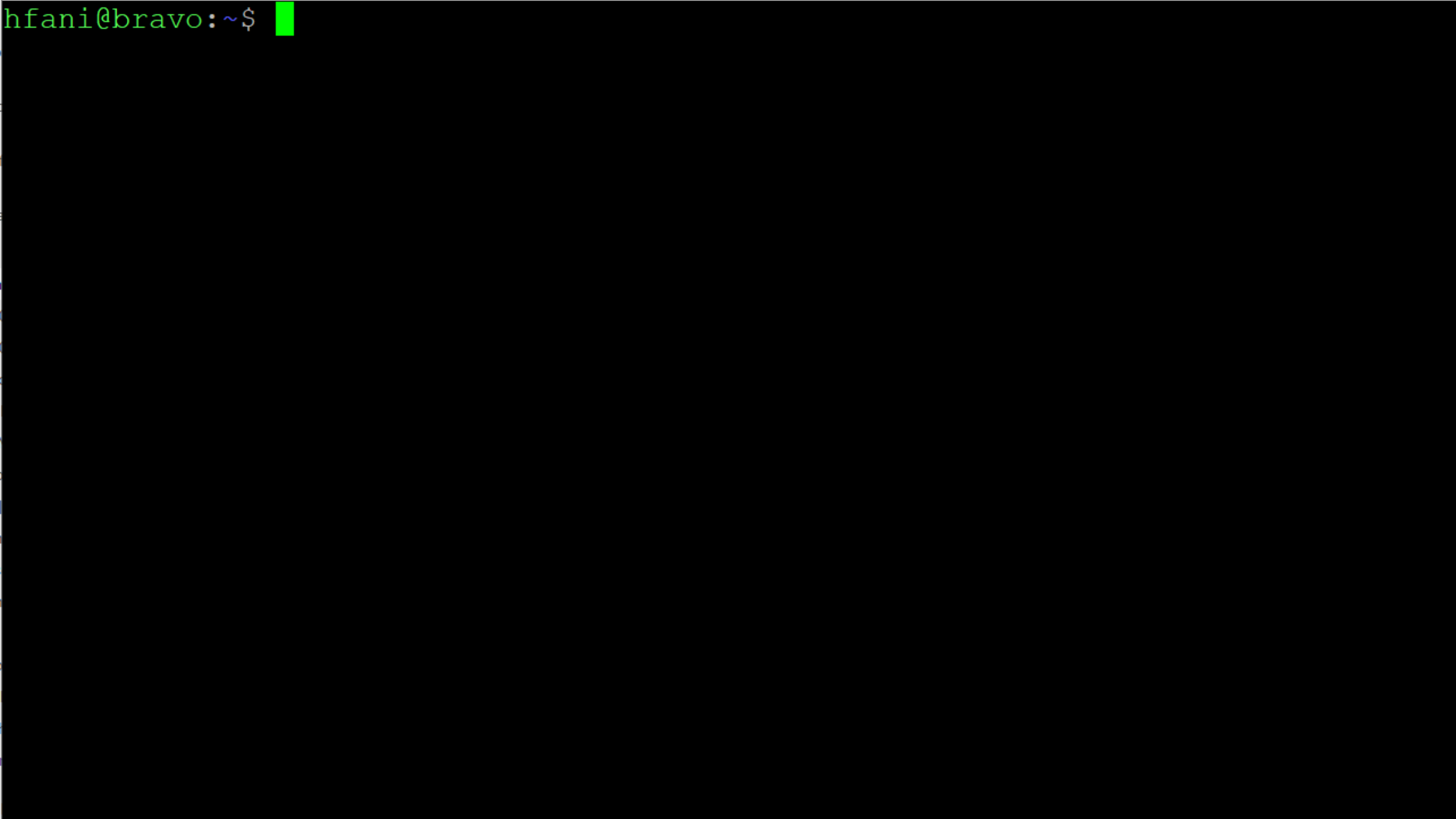
Kernel

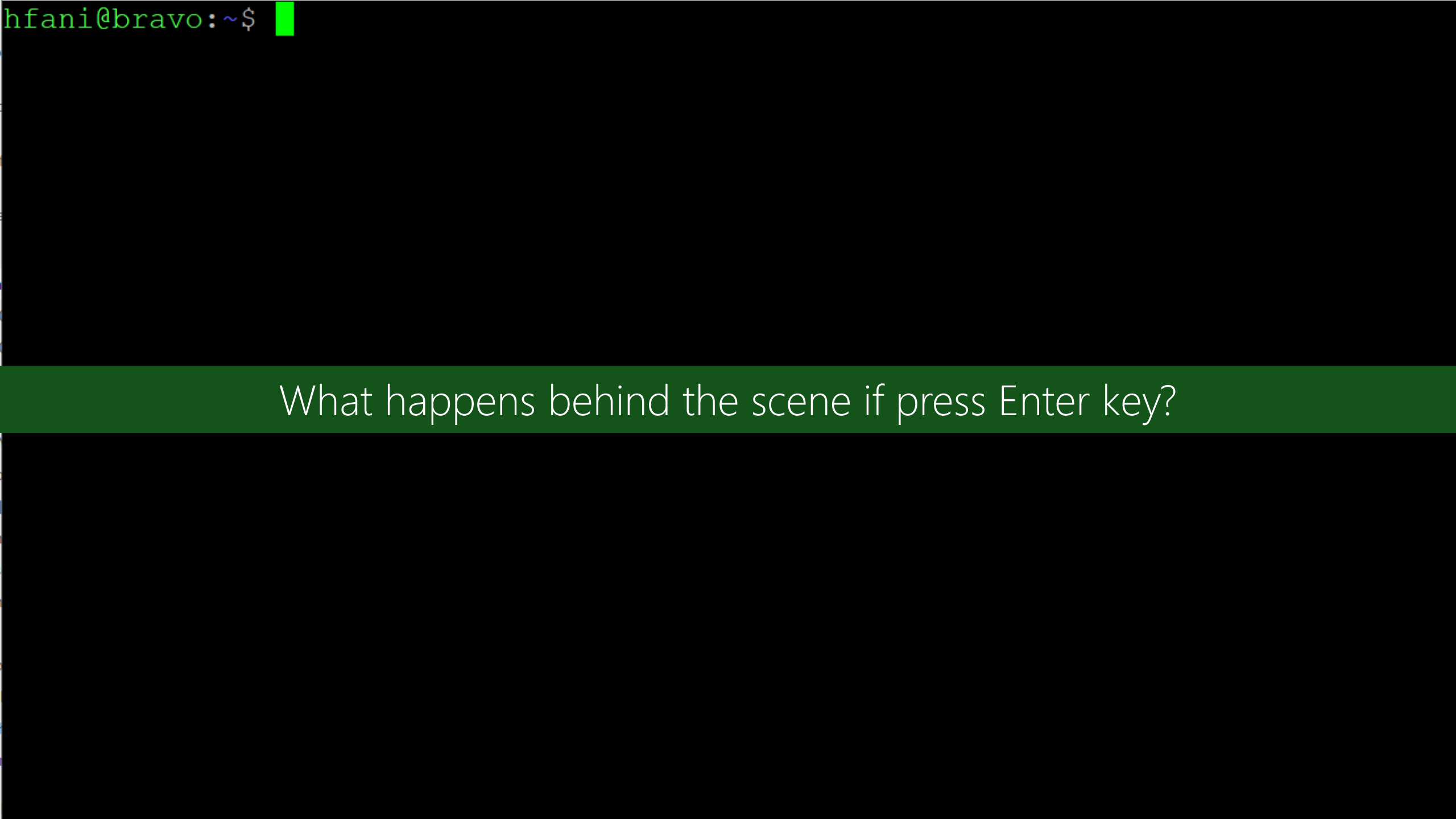
Shell: Busy-Waiting

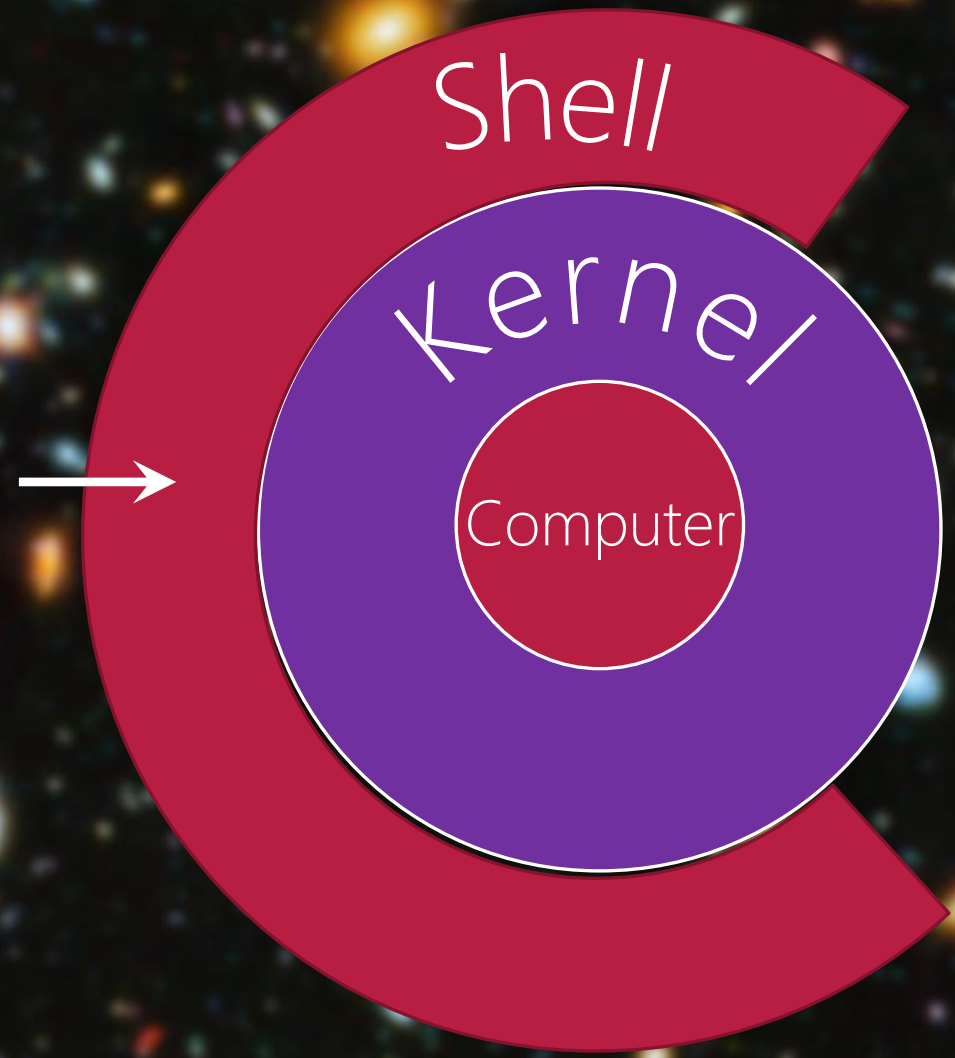
Bus

Processor

&Shell


hfani@bravo:~$


hfani@bravo:~$
What happens behind the scene if press Enter key?



Computer

Memory

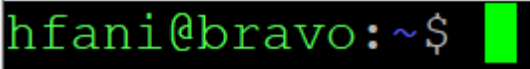
Kernel

Shell: Busy-Waiting

Bus

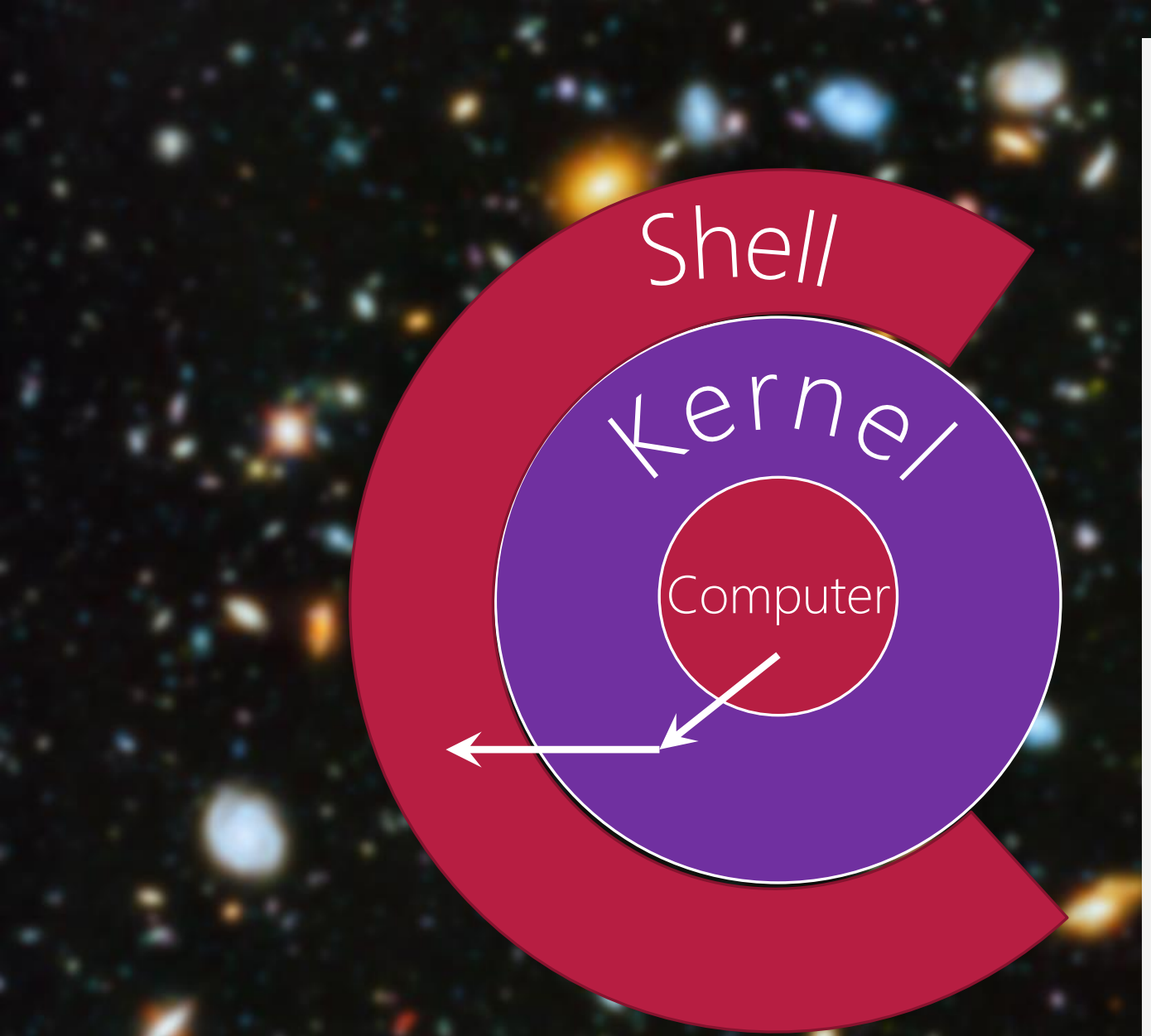
Processor

&Shell



What happens behind the scene if press Enter key?

1. Processor receives IRQ for keyboard
2. Processor sets the IP to the first OpCode of IRQ Handler for keyboard inside Device Manager of the Kernel
3. IRQ Handler interpret the keystroke: Enter
4. IRQ Handler gives it to the program that had the processor: Shell
5. Shell receives Enter: No commands … Reprints the prompt!



Computer

Memory

Kernel

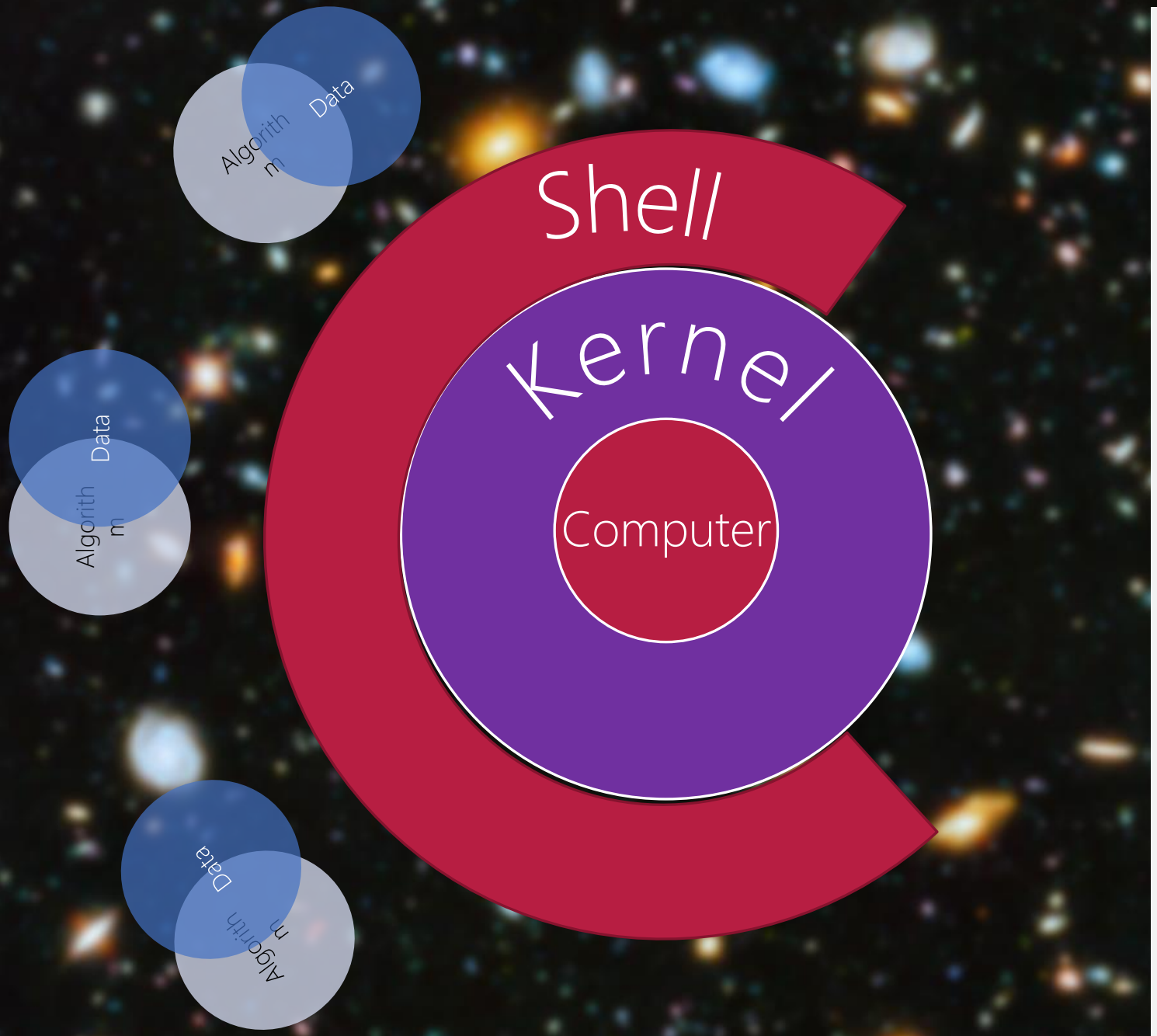
Shell: Busy-Waiting

Bus

Processor

&IRQ-H

```
hfani@bravo:~$ ./hello
```



Computer

Memory

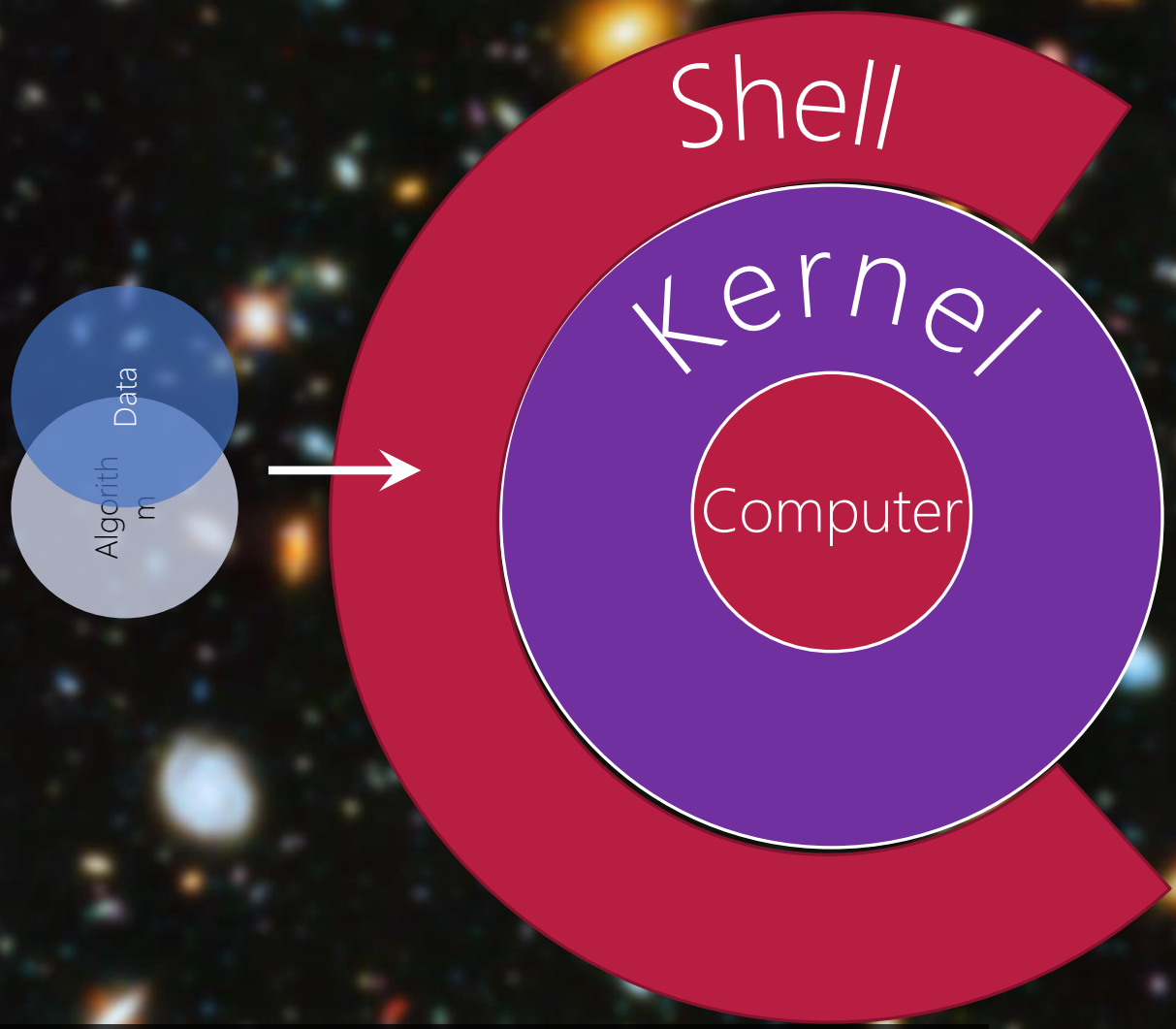
Kernel

Shell: Busy-Waiting

Bus

Processor

&Shell



1. Locate the program: /home/hfani/...

Computer

Memory

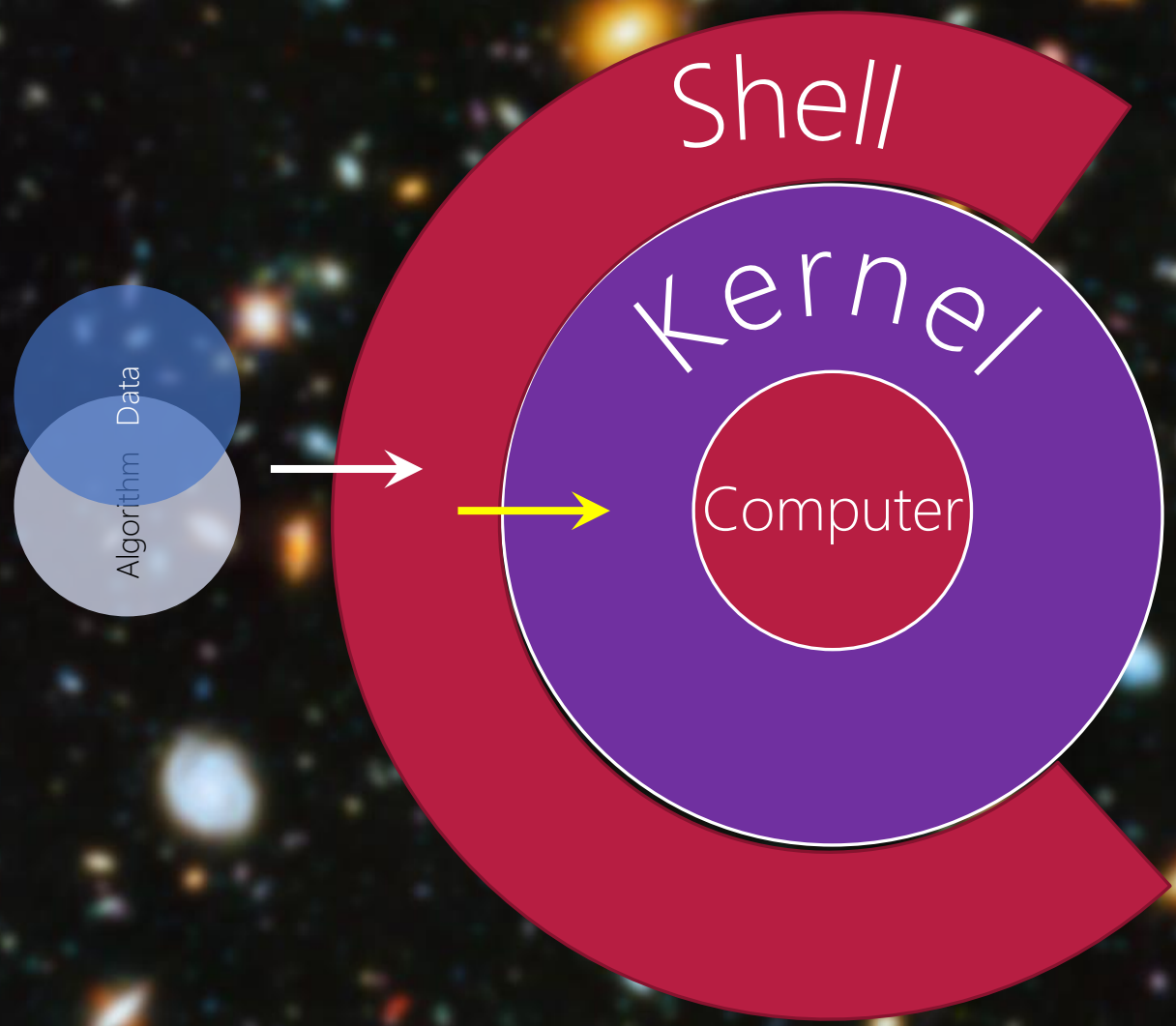
Kernel

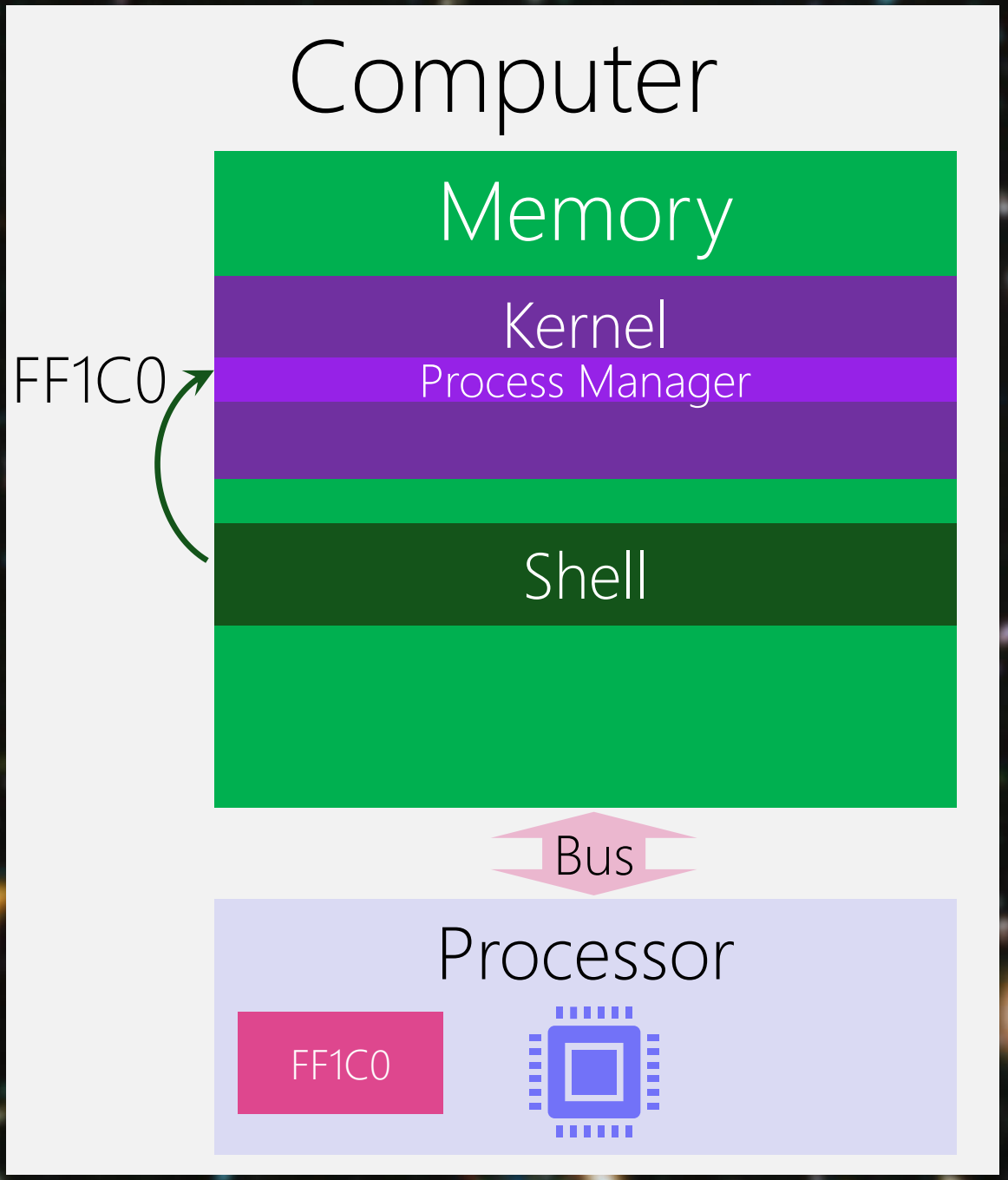
Shell: Busy-Waiting

Bus

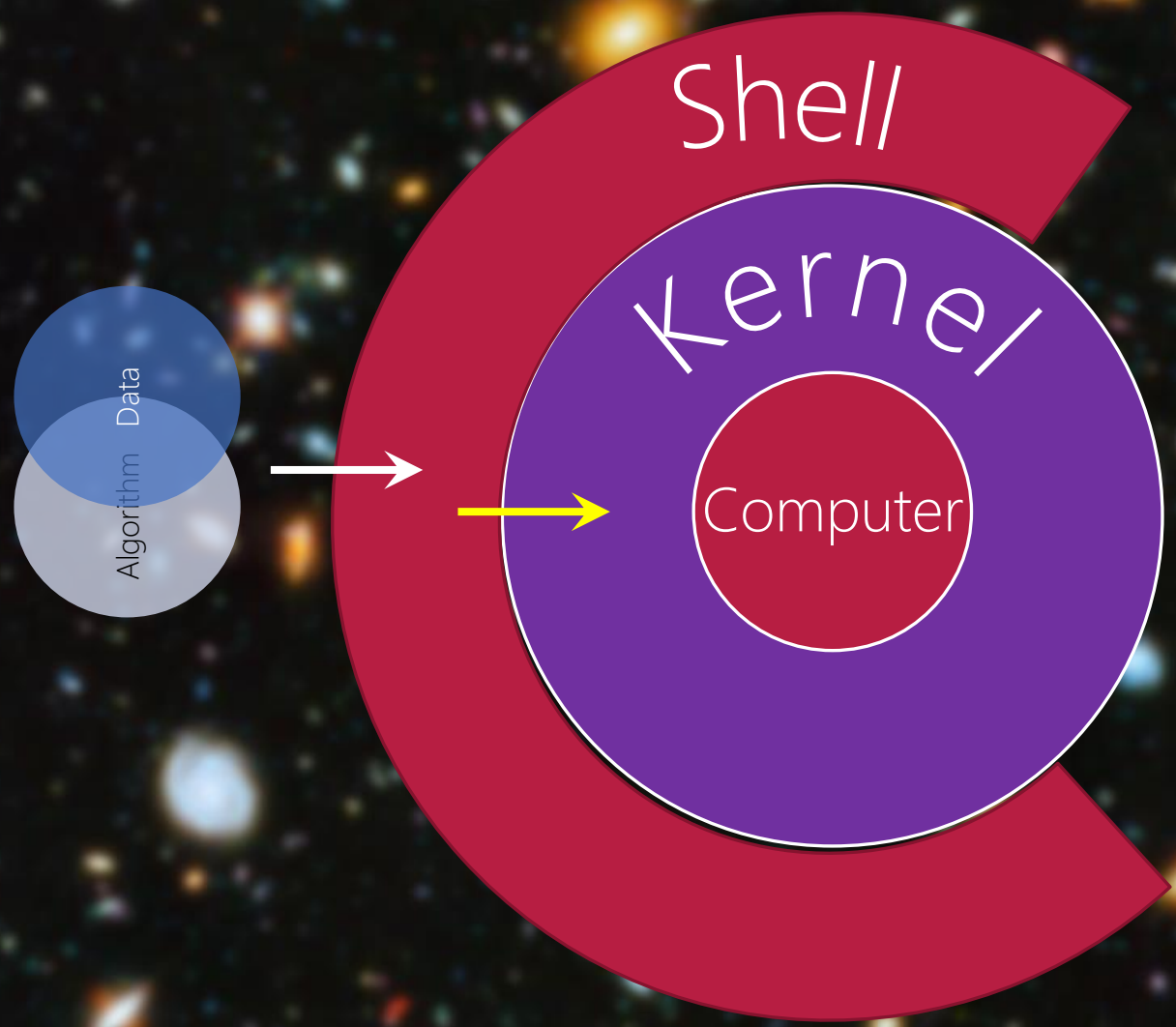
Processor

&Shell





2. Call Process Manager for loading the program: IP = FF1C0



Computer

Memory

Kernel

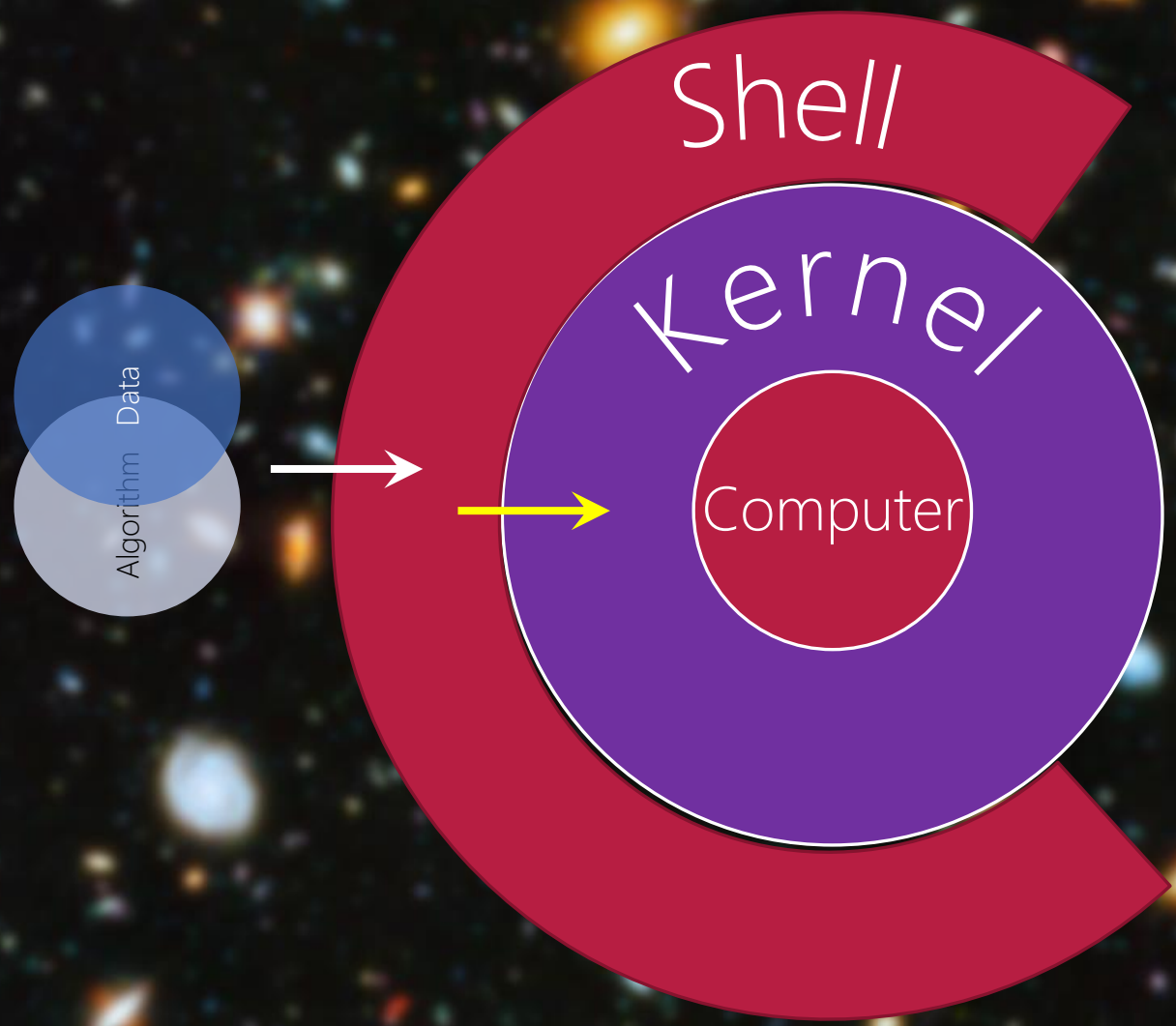
Process Manager

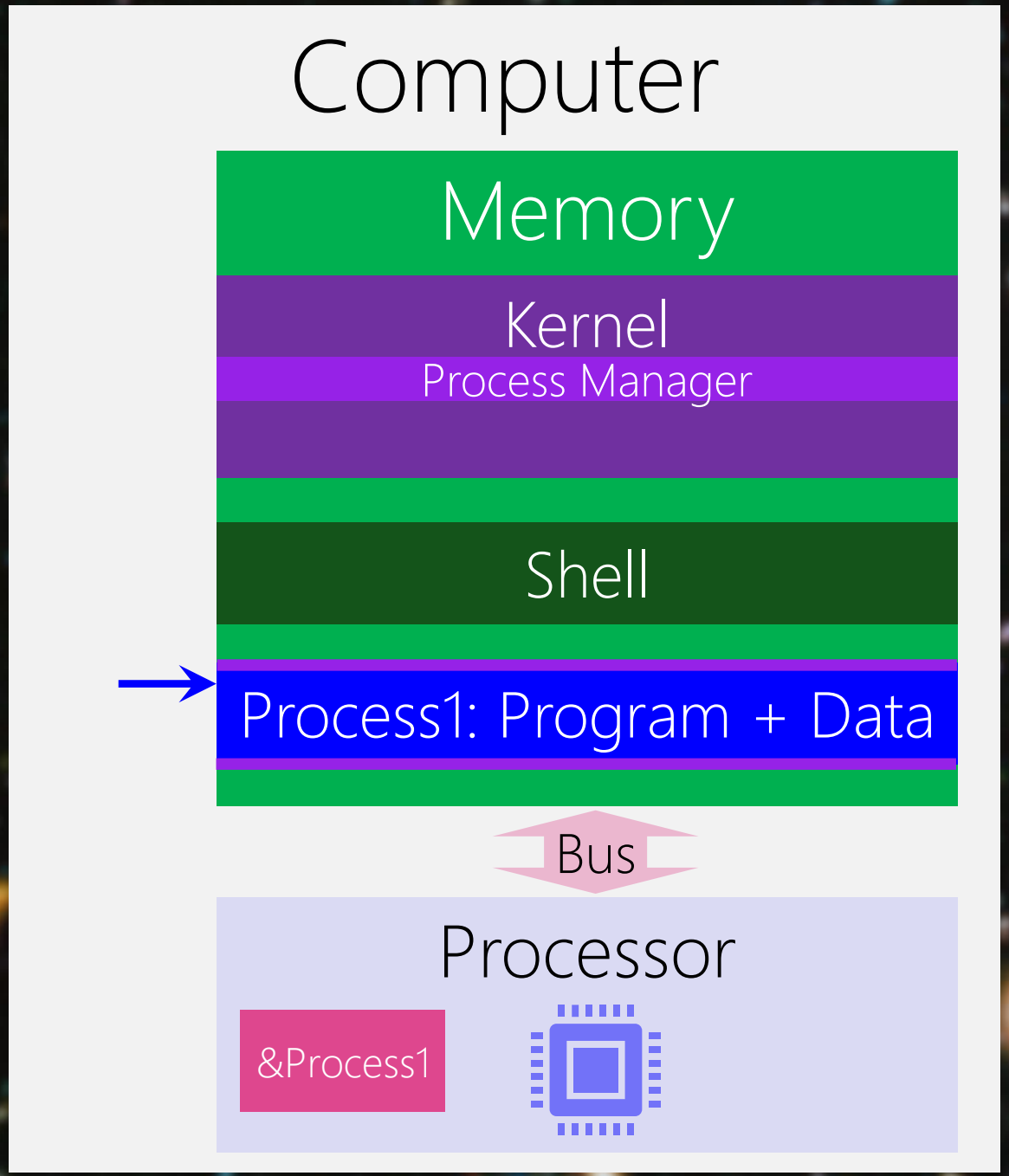
Shell

Process1: Program + Data

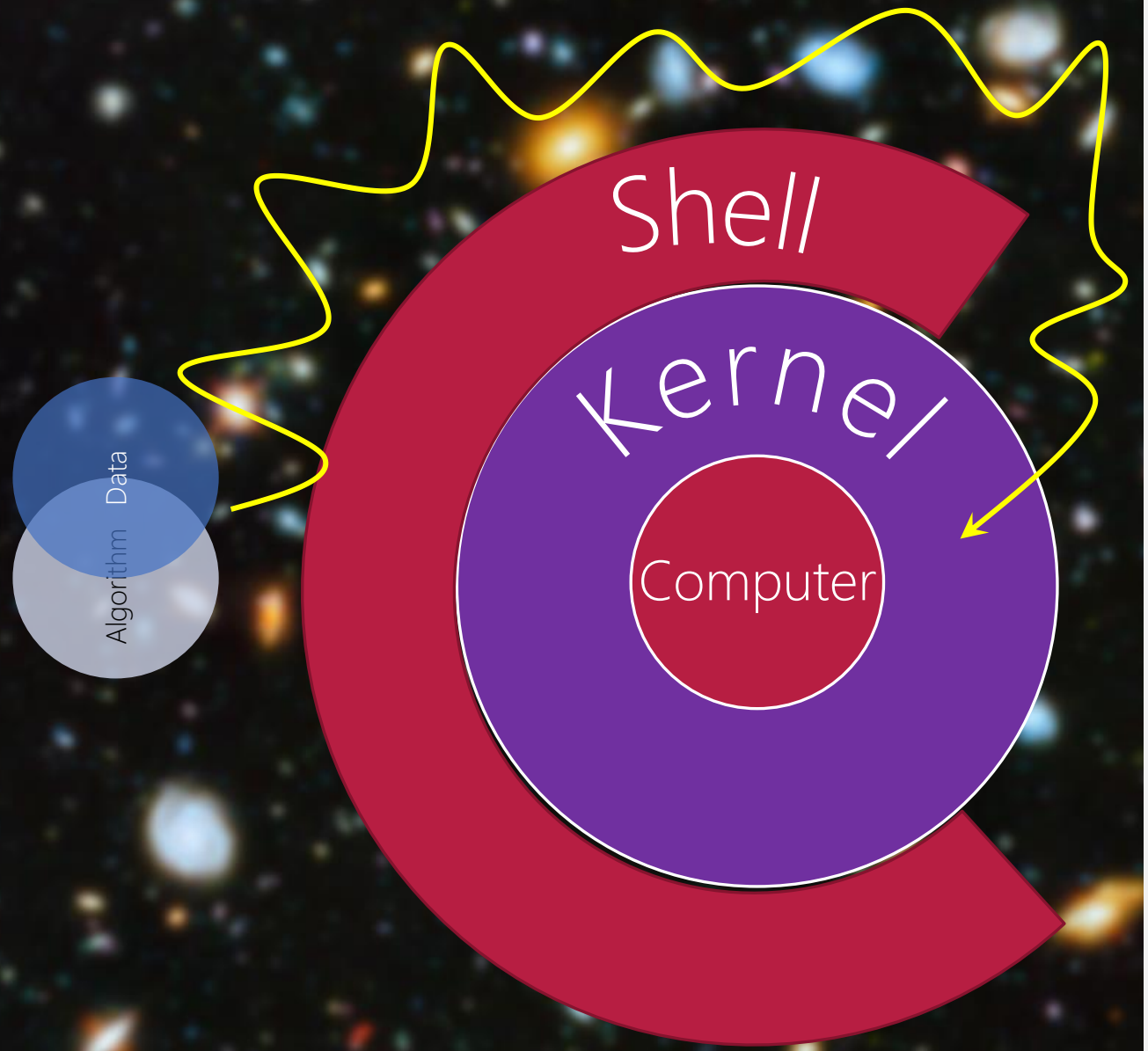
Bus

Processor

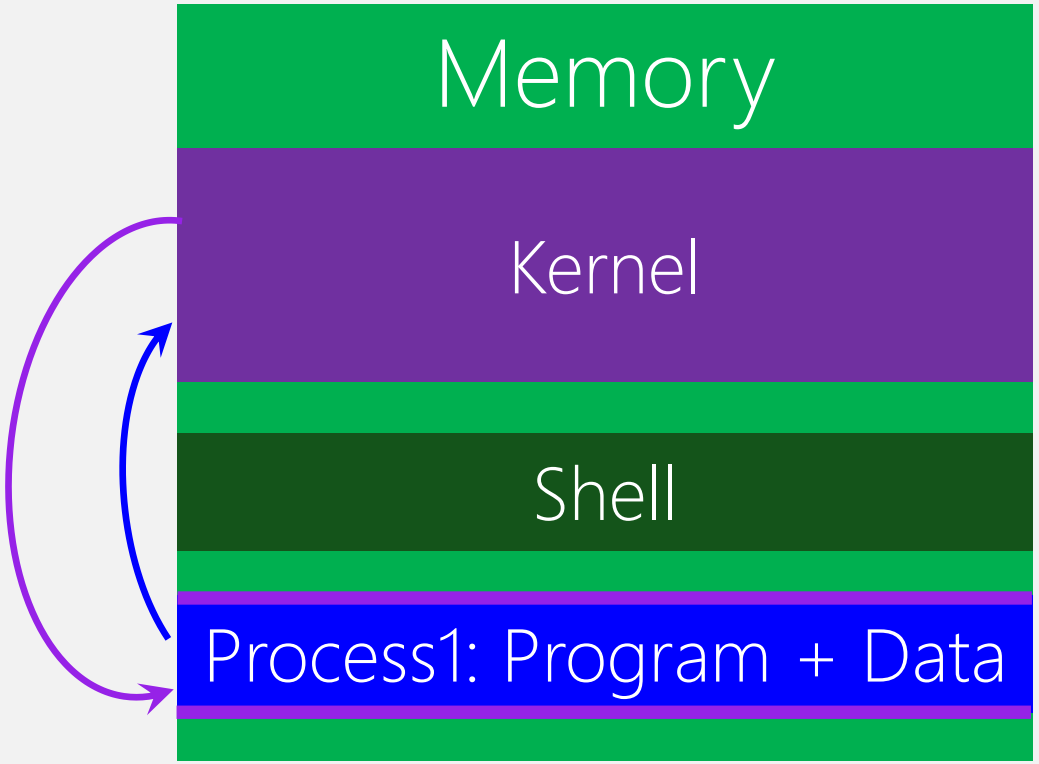




3. Point IP to the first line of the program: IP = &first opcode

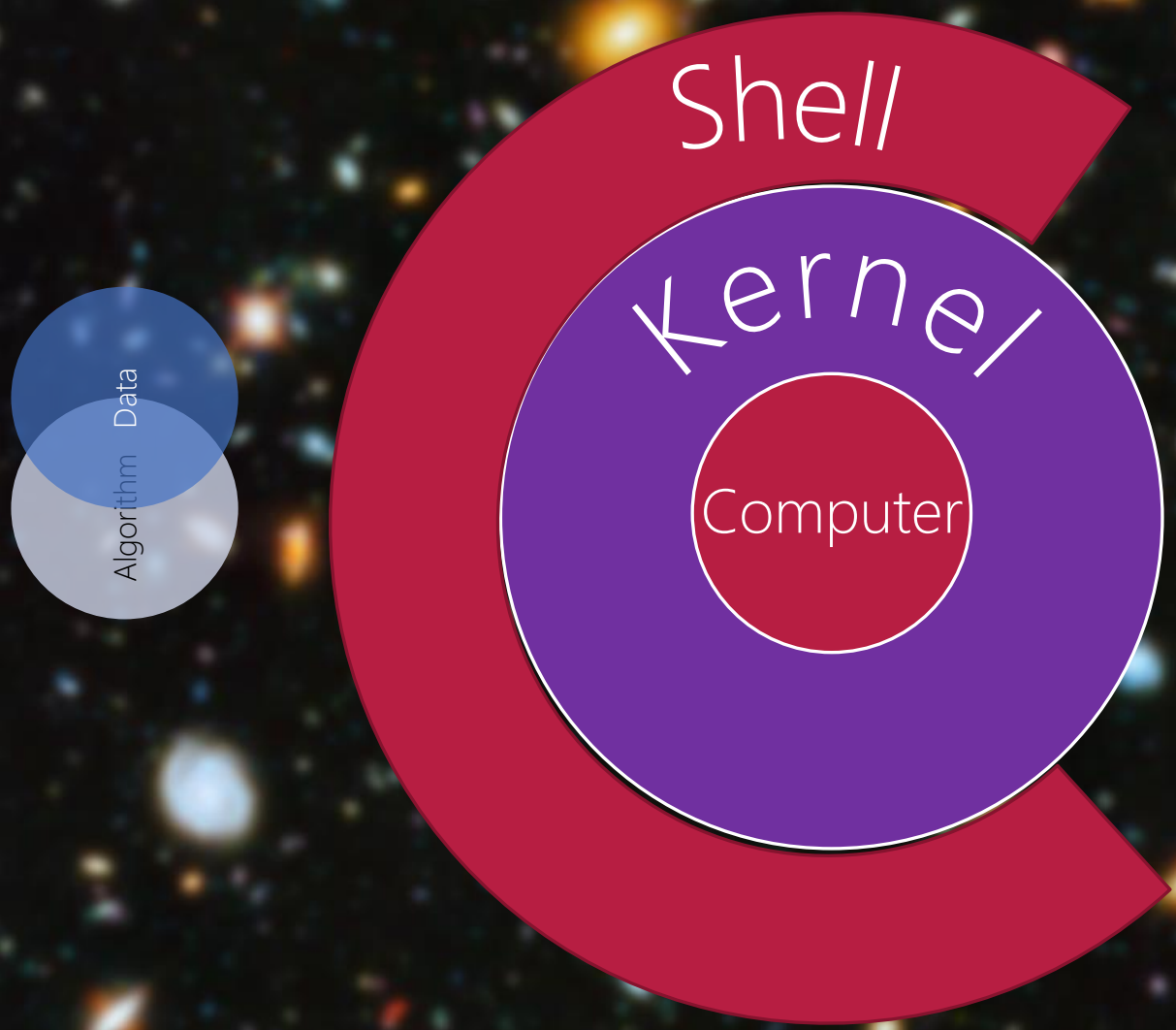


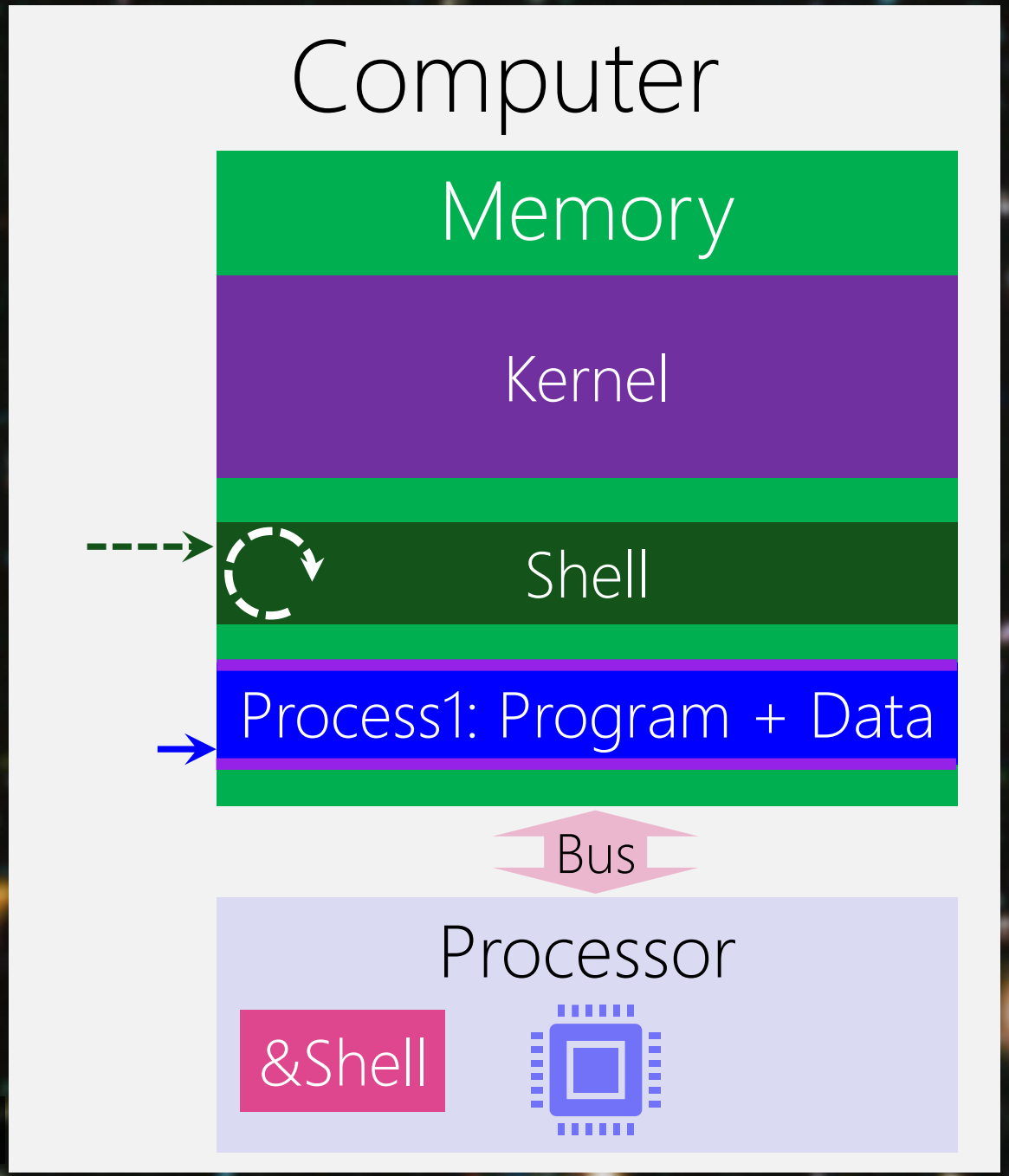
Computer



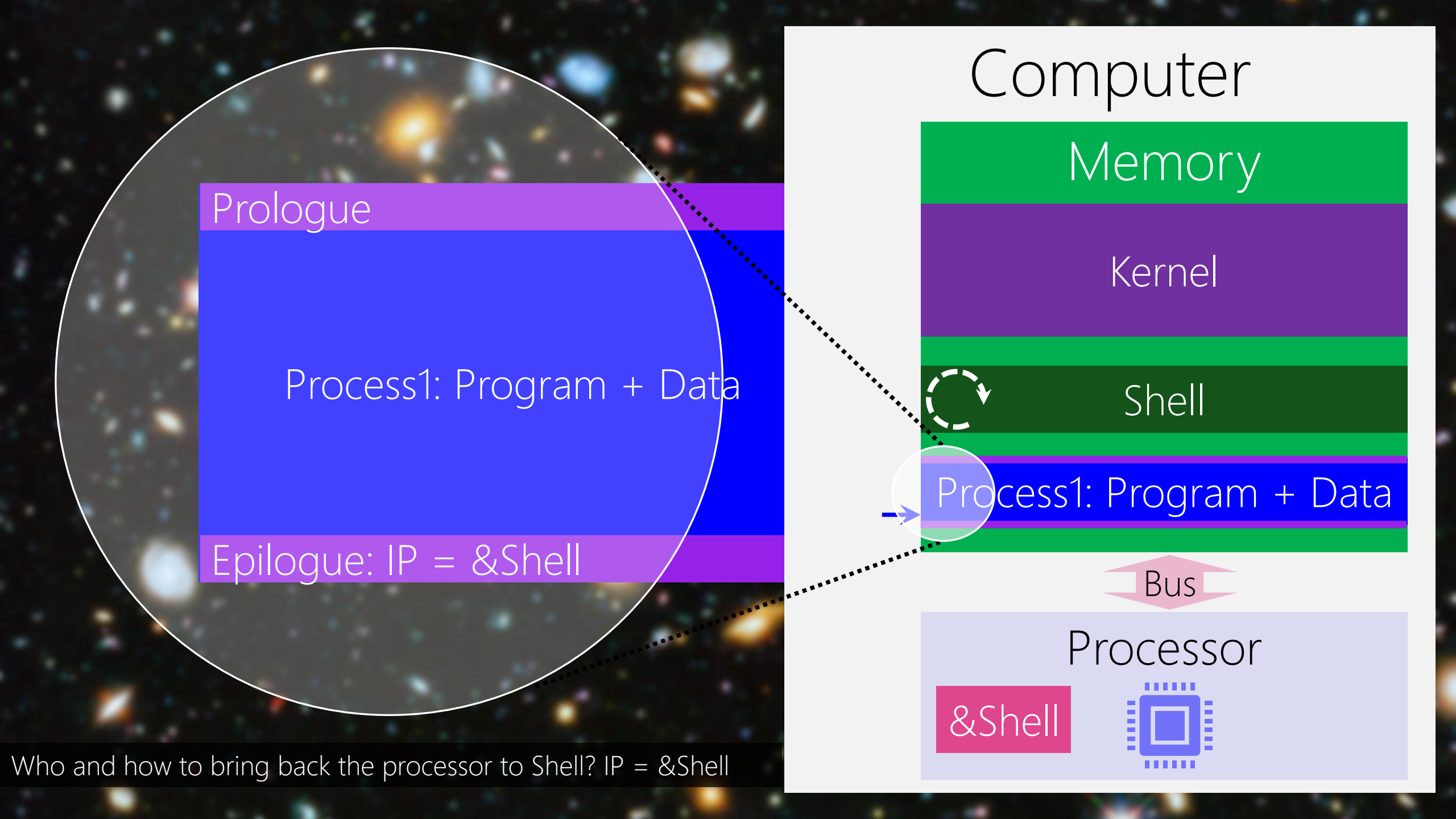
Bus

Processor





Who and how to brings back the processor to Shell? IP = &Shell

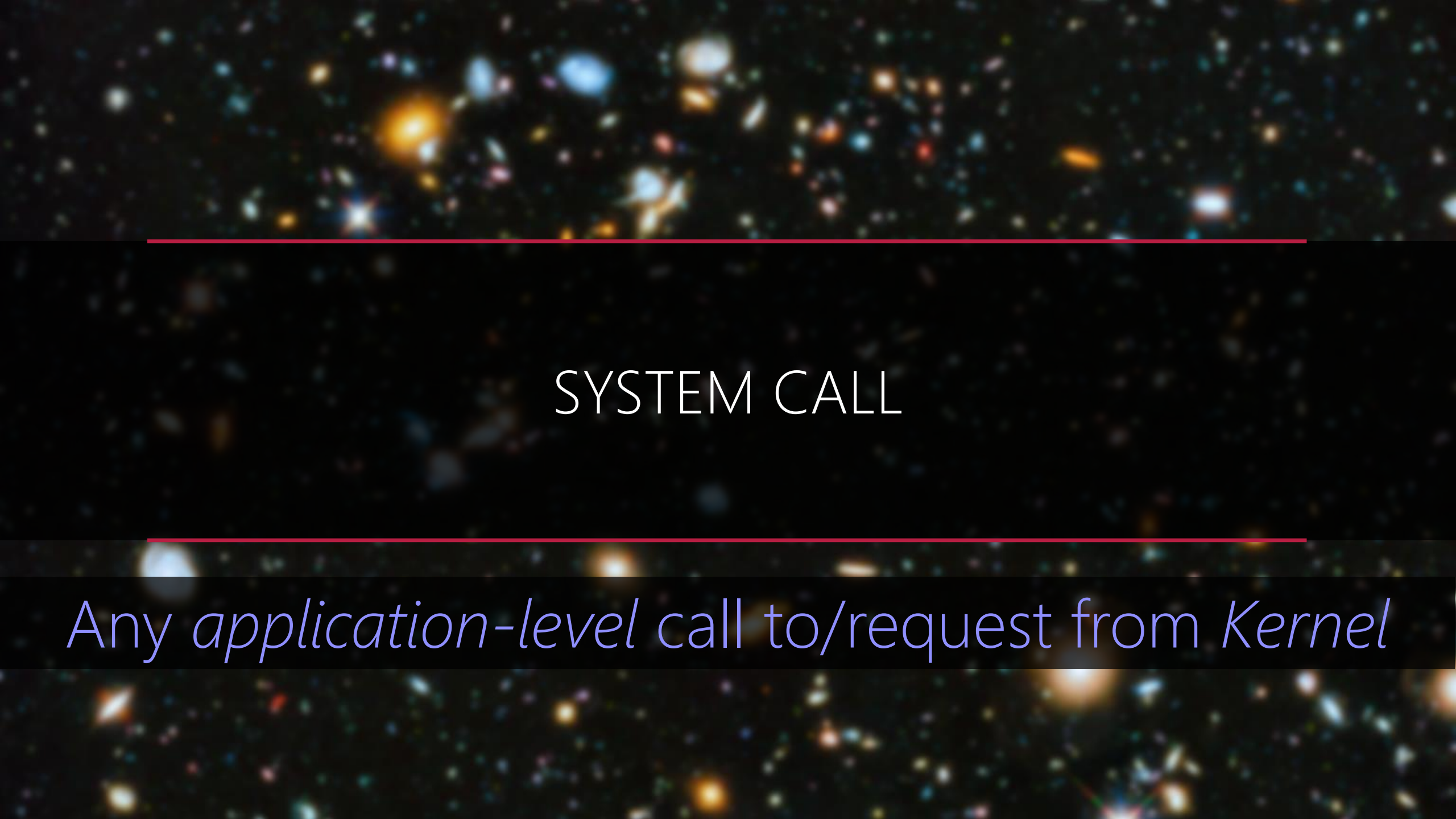


Who and how to bring back the processor to Shell? IP = &Shell

Knife Scene, Ex Machina (2015), Alex Garland
https://www.youtube.com/watch?v=67VATPxULPk

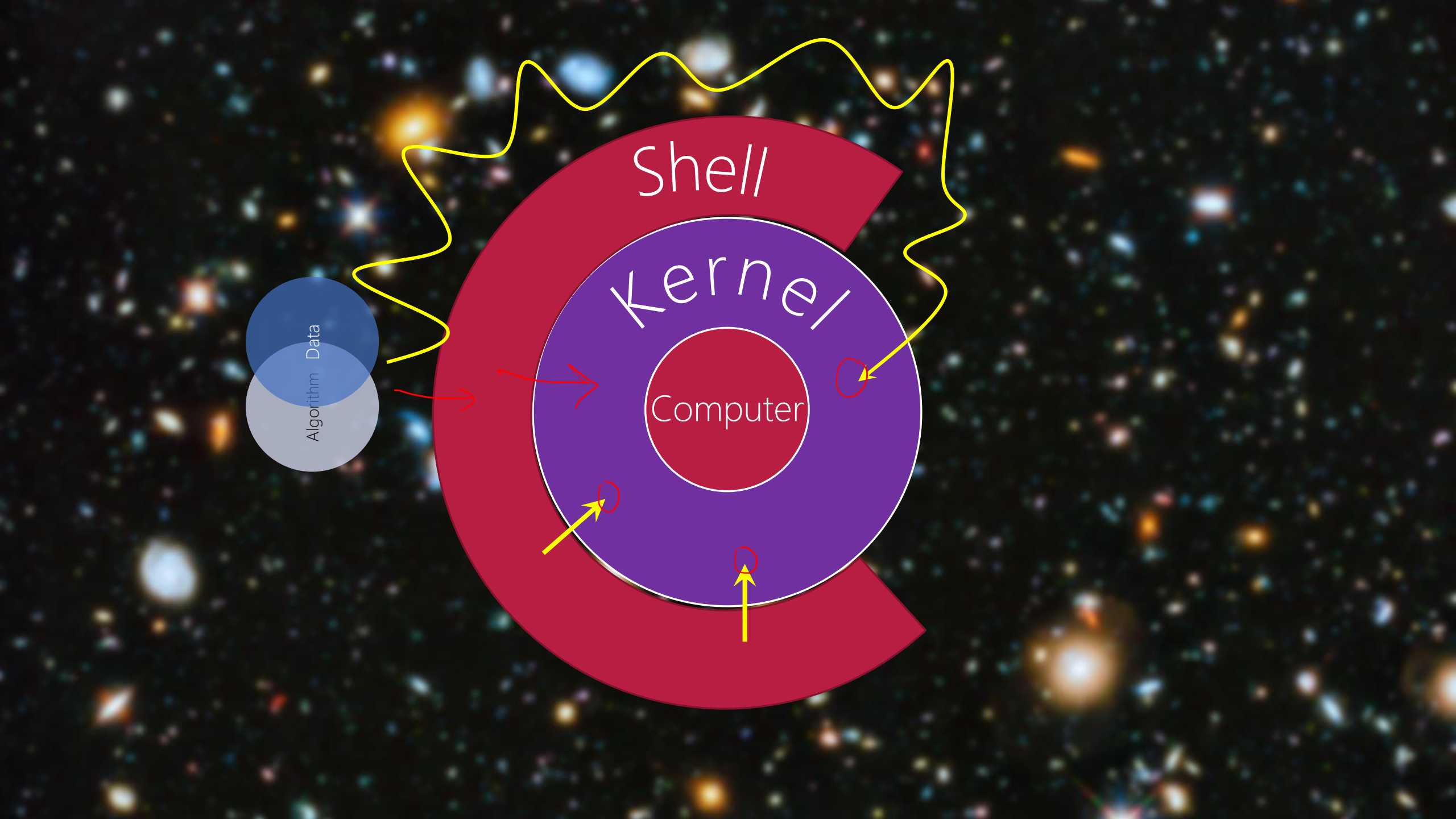
/DAY uus + eks + MAA ki naa/

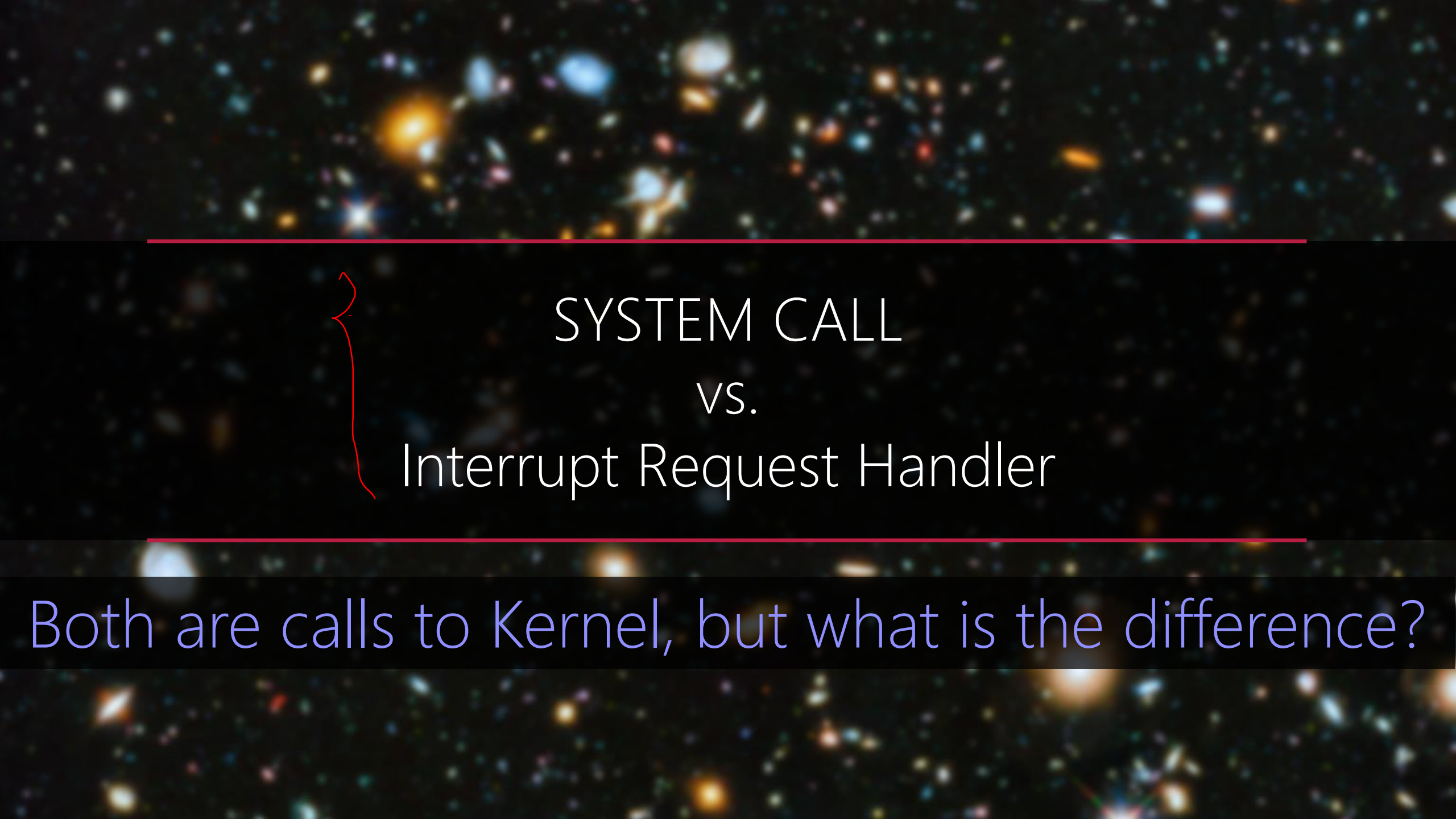
Deus Ex Machina means "god from the machine." In ancient Greek theater, when actors playing gods carried onto stage by a machine. These gods would then serve as the ultimate arbiters of right and wrong and decide how the story ends. But this film is just called "Ex Machina" without the "Deus." A machine without a god.
https://www.looper.com/148401/the-ending-of-ex-machina-finally-explained/

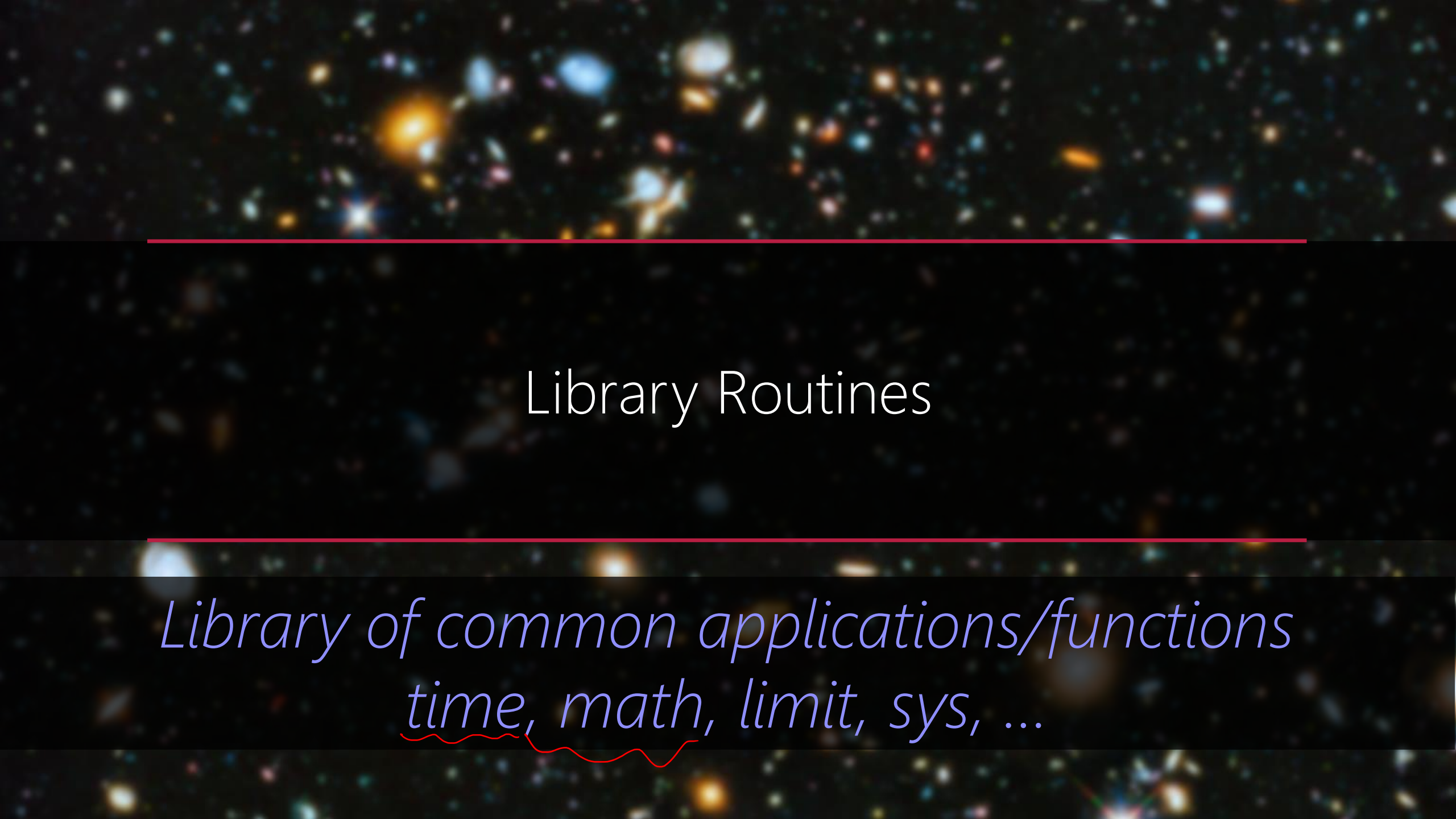


SYSTEM CALL

Any *application-level* call to/request from *Kernel*


Shell
Kernel
Computer
Algorithm
Data

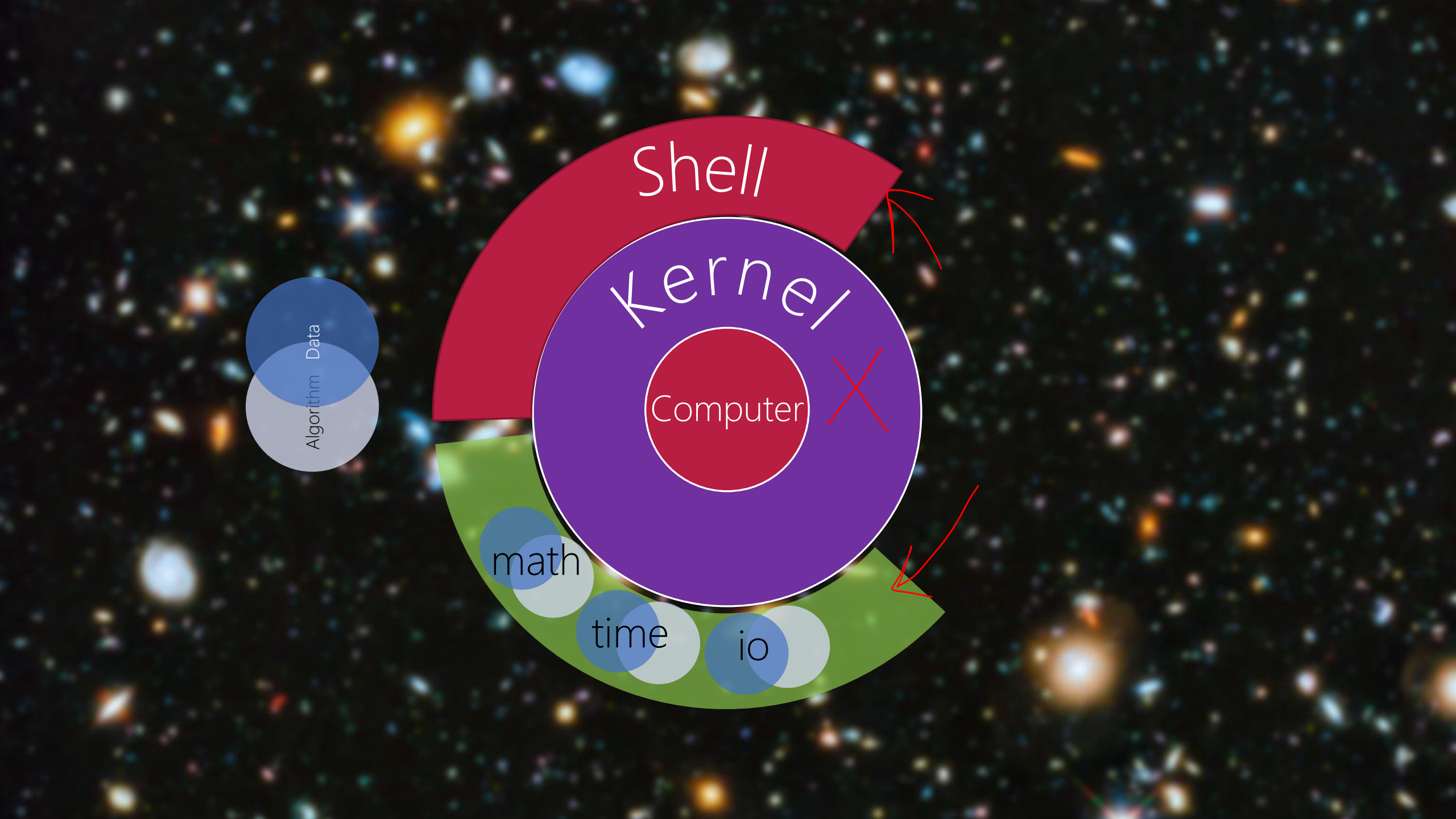

SYSTEM CALL
vs.
Interrupt Request Handler
Both are calls to Kernel, but what is the difference?

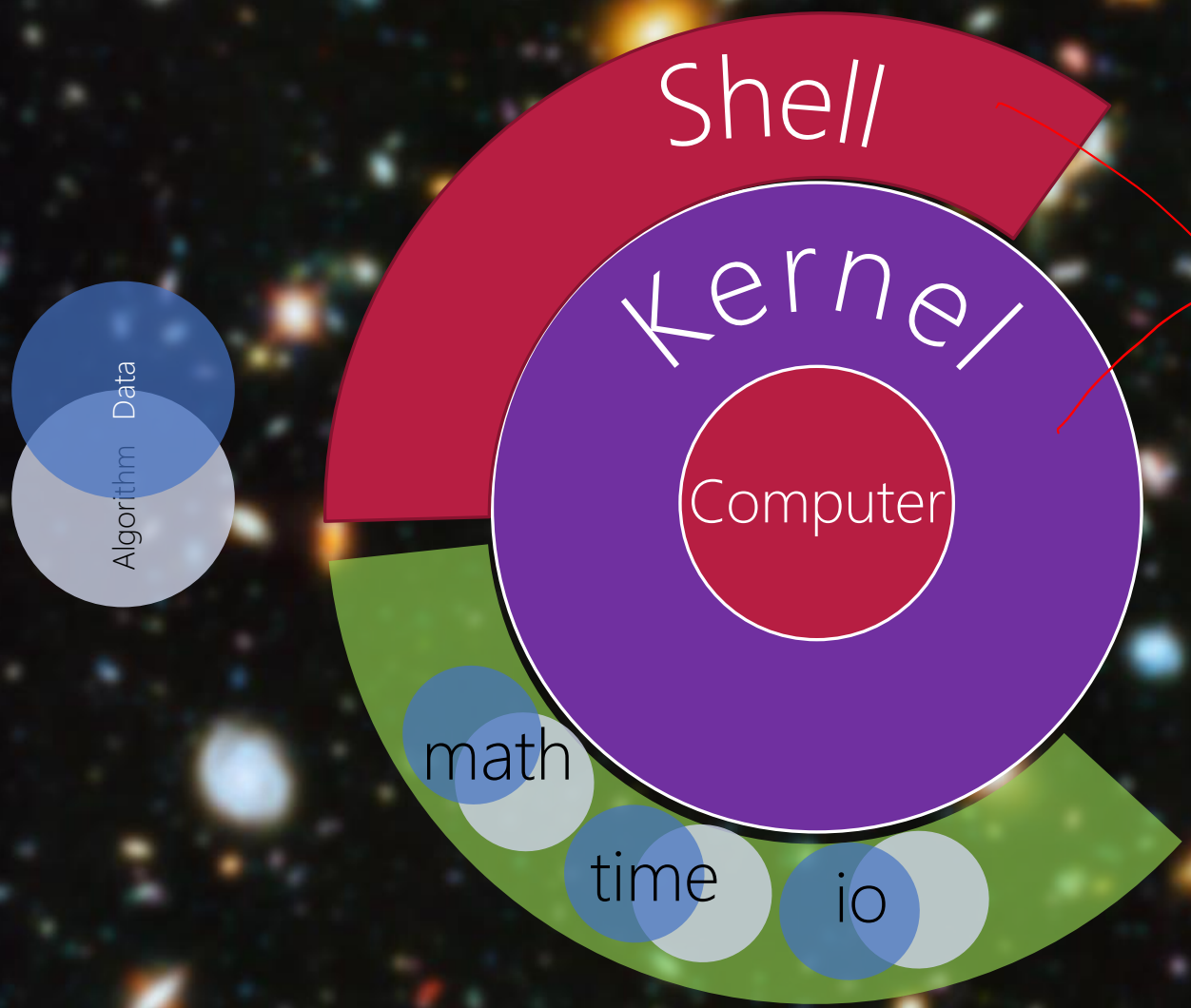


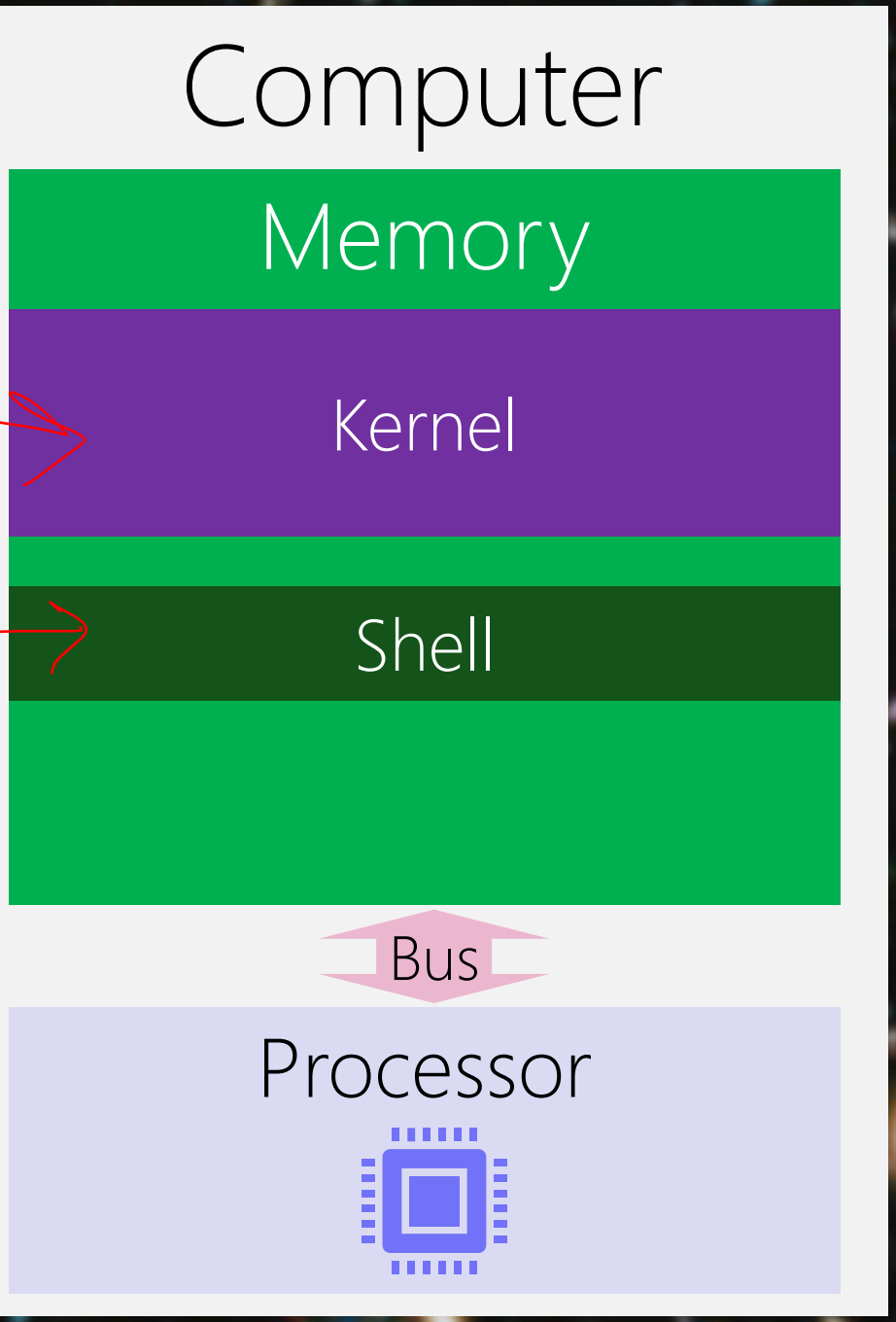
Library Routines

Library of common applications/functions

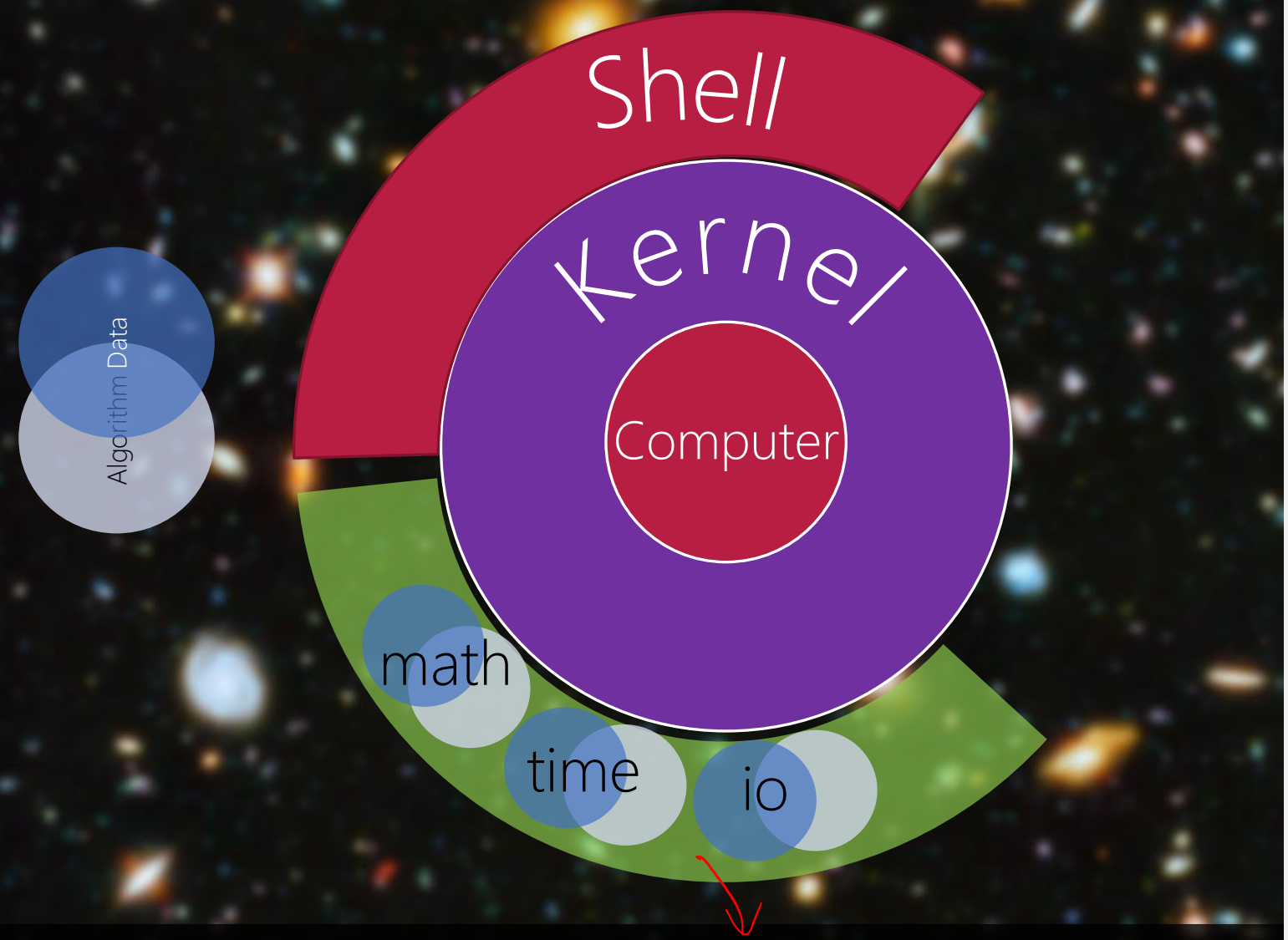
time, math, limit, sys, …


Shell
Kernel
Computer
Algorithm
Data
math
time
io

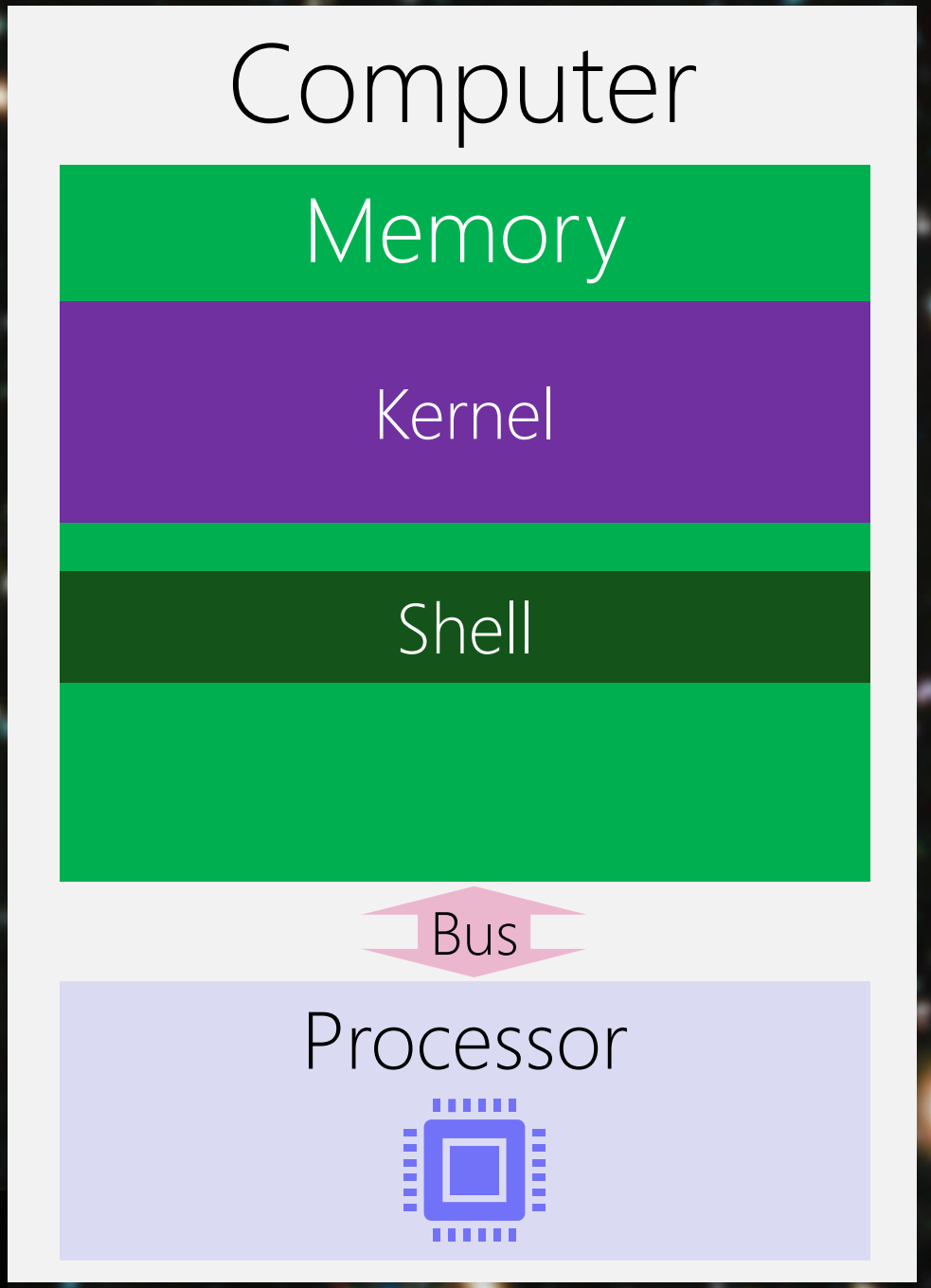


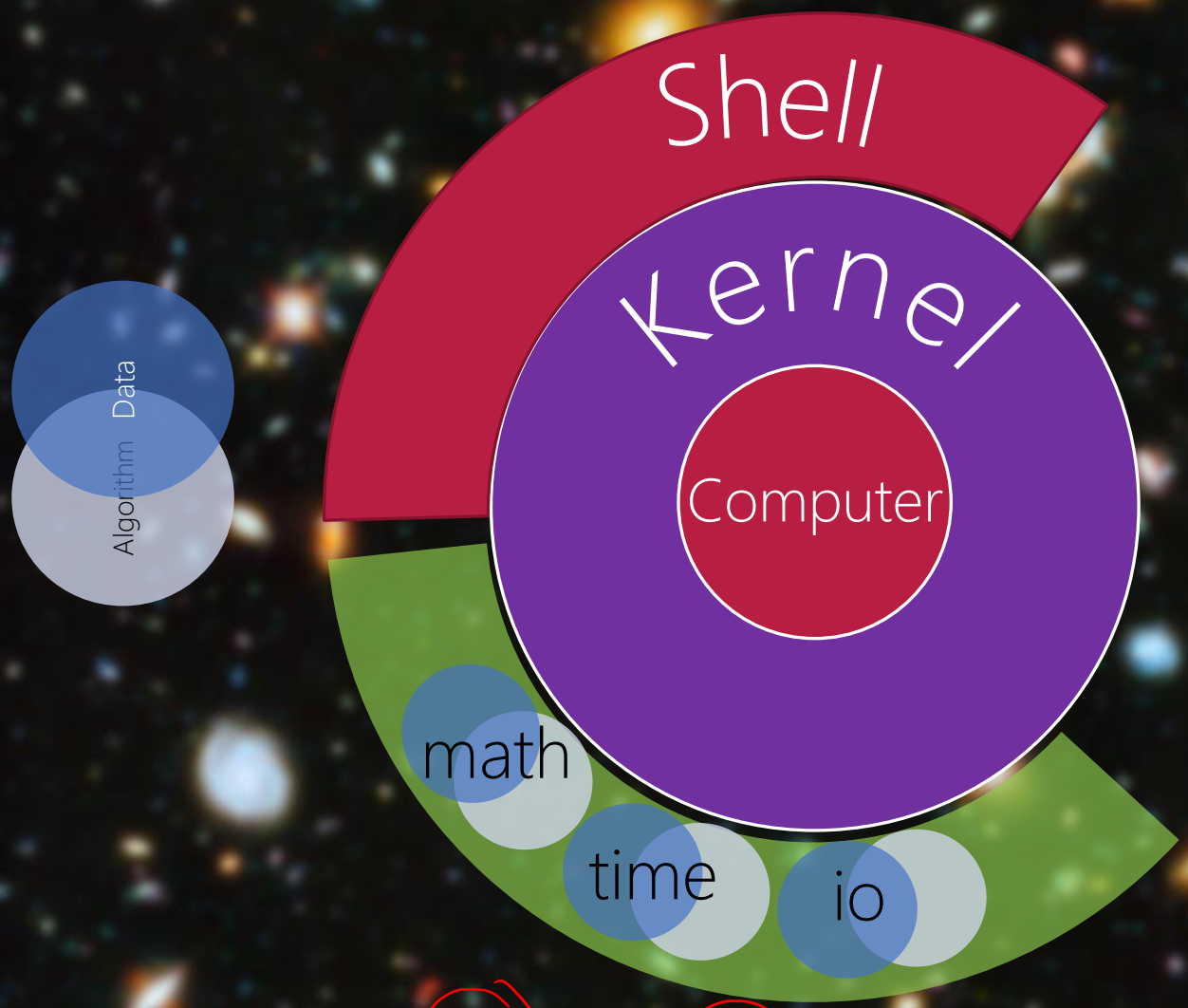


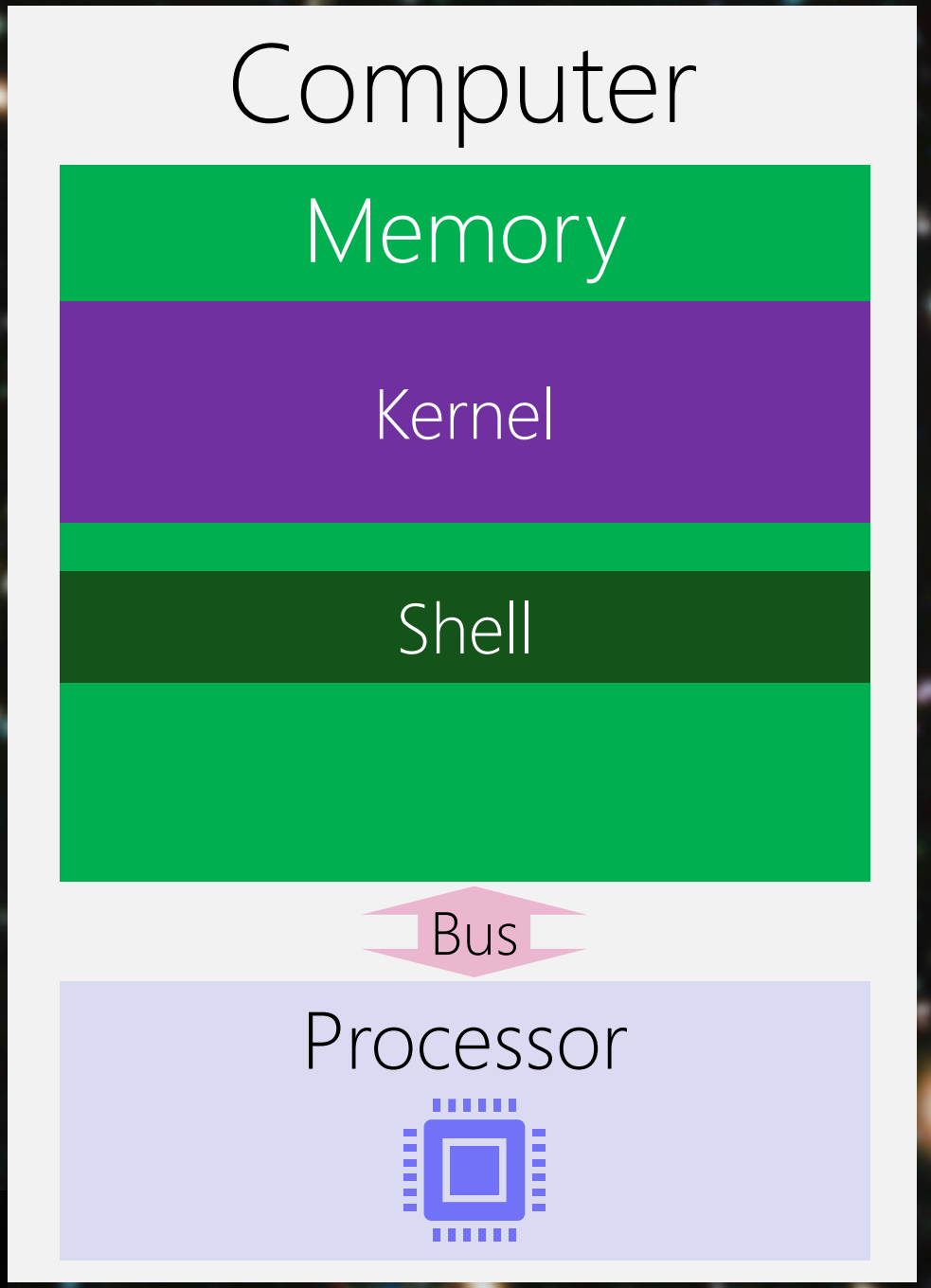
Library routines are NOT in memory!



They are all in OP Code!







How to link (include vs. call) to library routines?

STATIC LINK

At compile time, using linker

`#include`

C

Compiler

```
#include <stdio.h>
void main(){
    printf("hello world!");
}
```

Assembly

Assembler

```
<printf@plt>:
jmpq        *0x3002(%rip)
pushq       $0x0
jmpq        401000 <.plt>
<main>:
push        %rbp
…
```

OP Code

stdio.h

```
0001 0000 0000 0010 0000 0000 0000 1100
00f1 0000 0003 0000 0000 0000 0000 0000
0000 0000 0000 0000 3309 0110 0111 0000
0046 0000 0220 0000 0000 0000 0000 2200
0111 0000 0000 0000 e432 0000 0000 0000
```

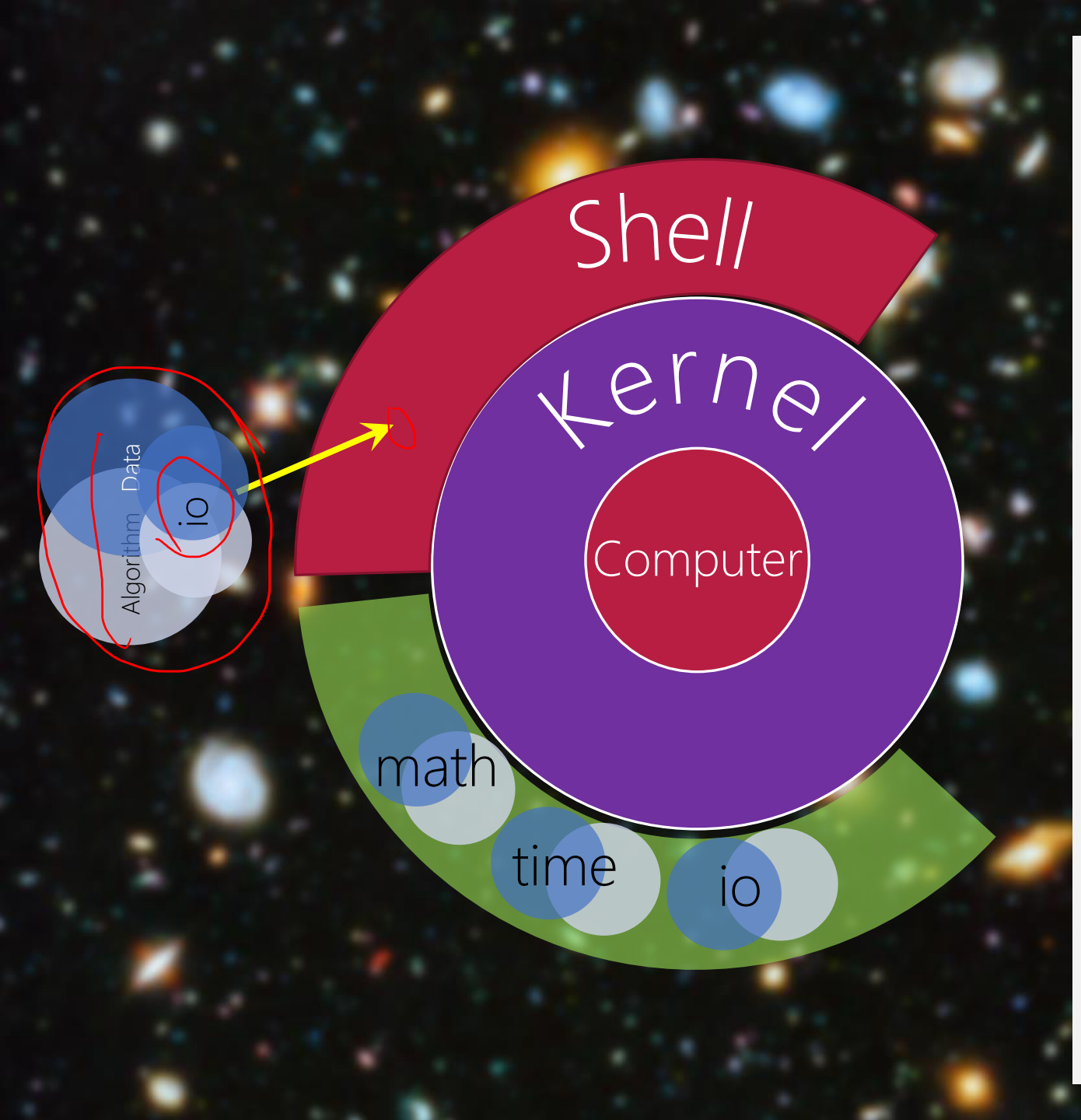
OP Code

```
0001 0000 0000 0000 0000 0000 0000 0000
0011 0000 0003 0000 0000 0000 0000 0000
0000 0000 0000 0000 3309 0000 0000 0000
0096 0000 0000 0000 0000 0000 0000 0000
0001 0000 0000 0000 0000 0000 0000 0000
```

Linker

cc

```
0001 0000 0000 0010 0000 0000 0000 1100
00f1 0000 0003 0000 0000 0000 0000 0000
0000 0000 0000 0000 3309 0110 0111 0000
0046 0000 0220 0000 0000 0000 0000 2200
0111 0000 0000 0000 e432 0000 0000 0000
0001 0000 0000 0000 0000 0000 0000 0000
0011 0000 0003 0000 0000 0000 0000 0000
0000 0000 0000 0000 3309 0000 0000 0000
0096 0000 0000 0000 0000 0000 0000 0000
0001 0000 0000 0000 0000 0000 0000 0000
```



Computer

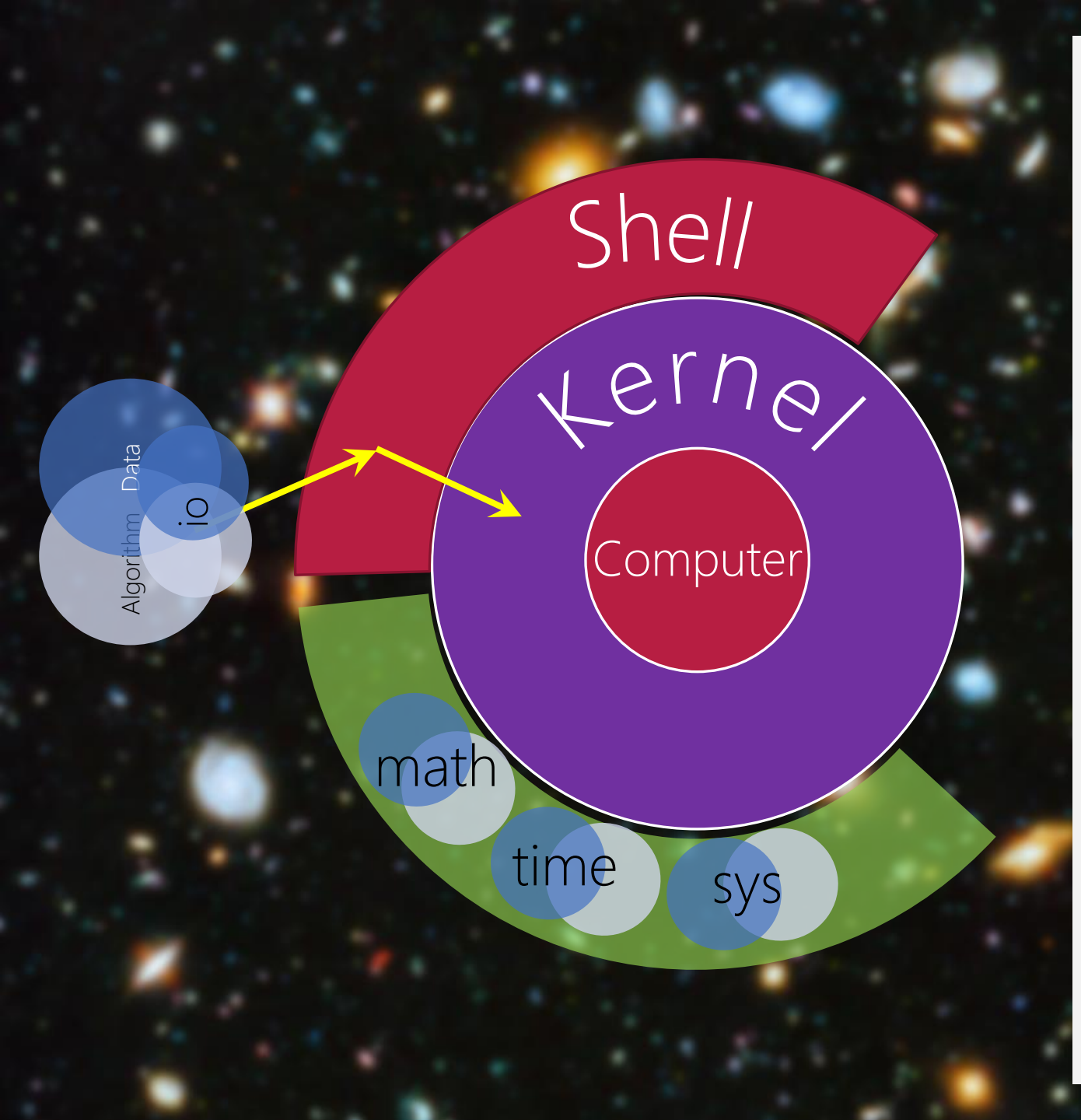
Memory

Kernel

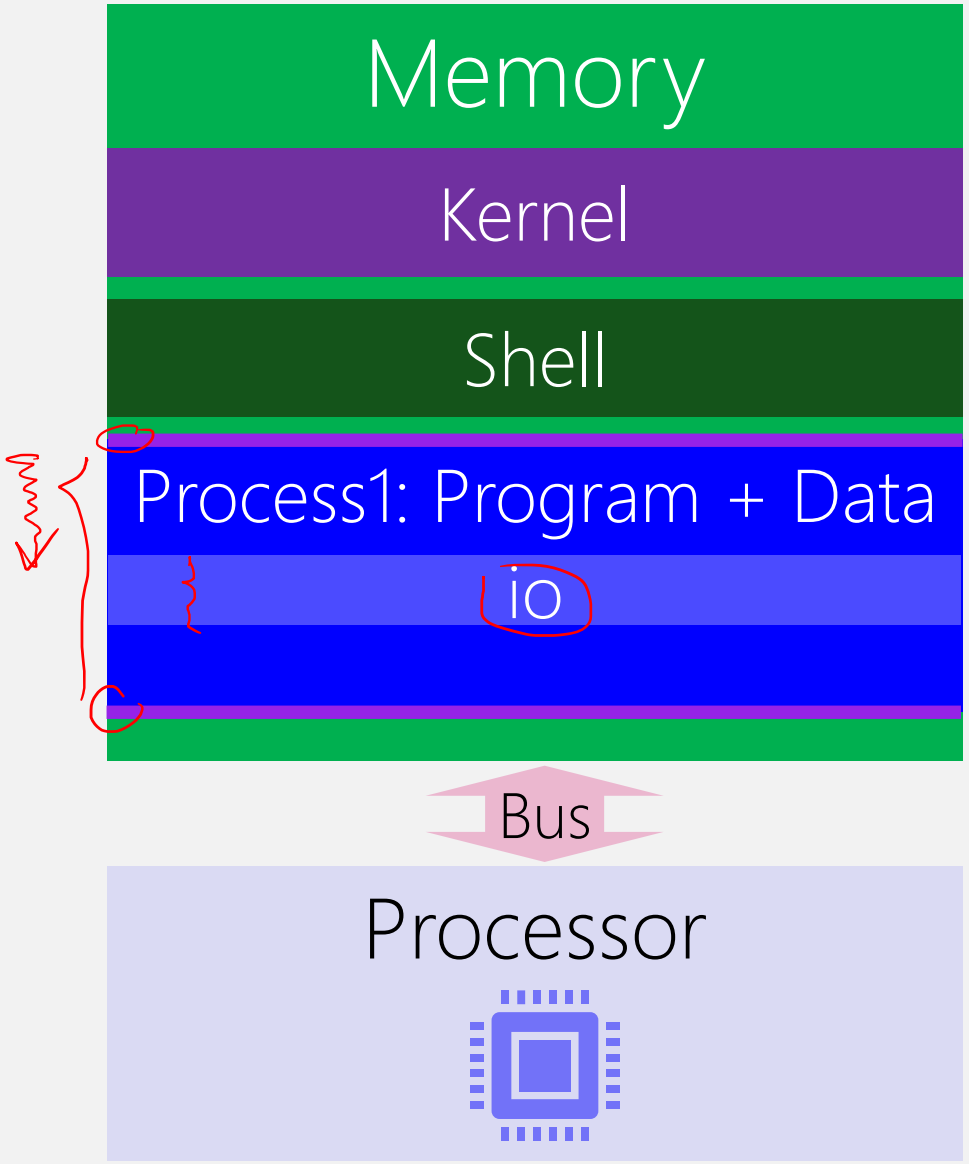
Shell

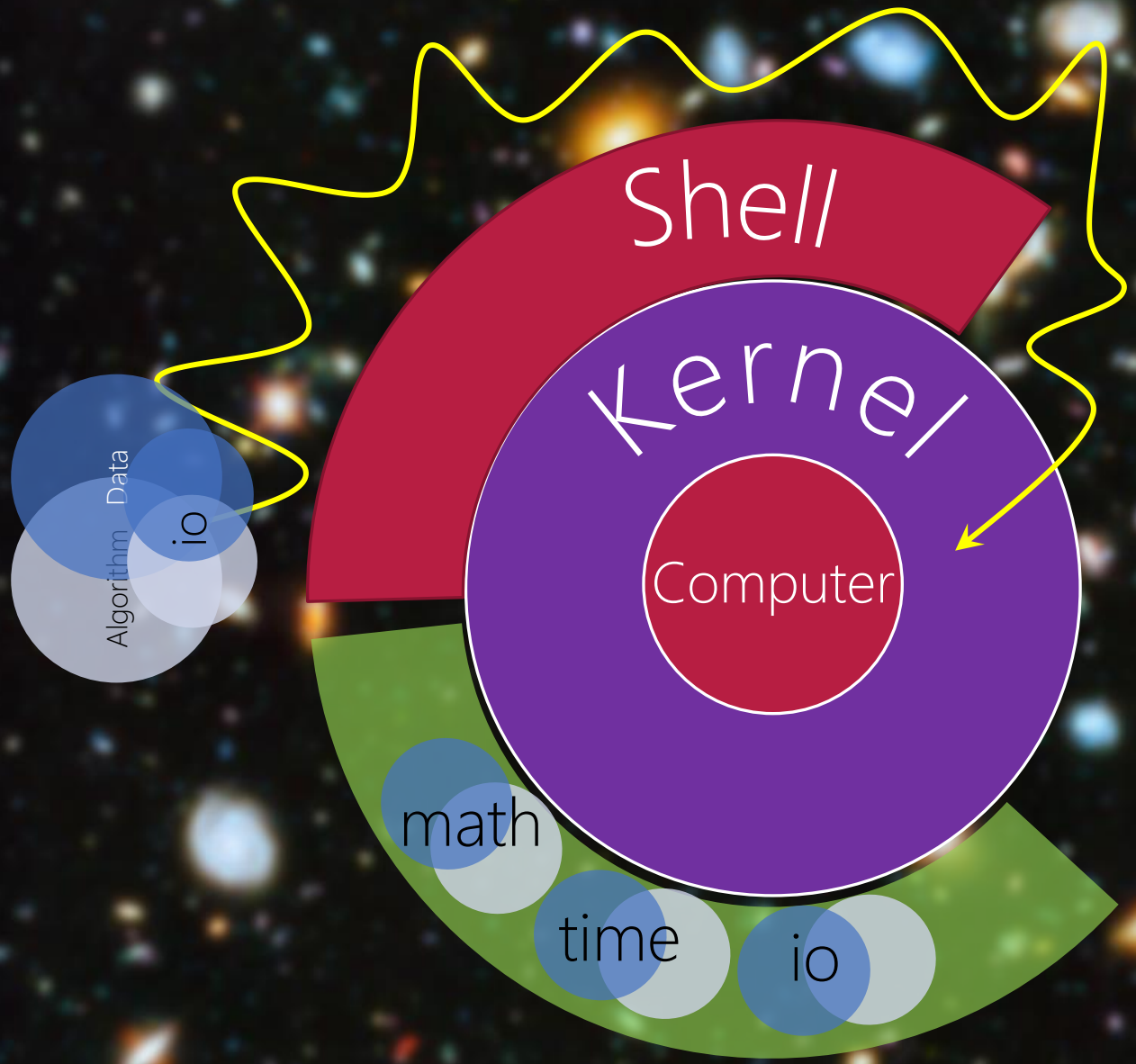
Bus

Processor

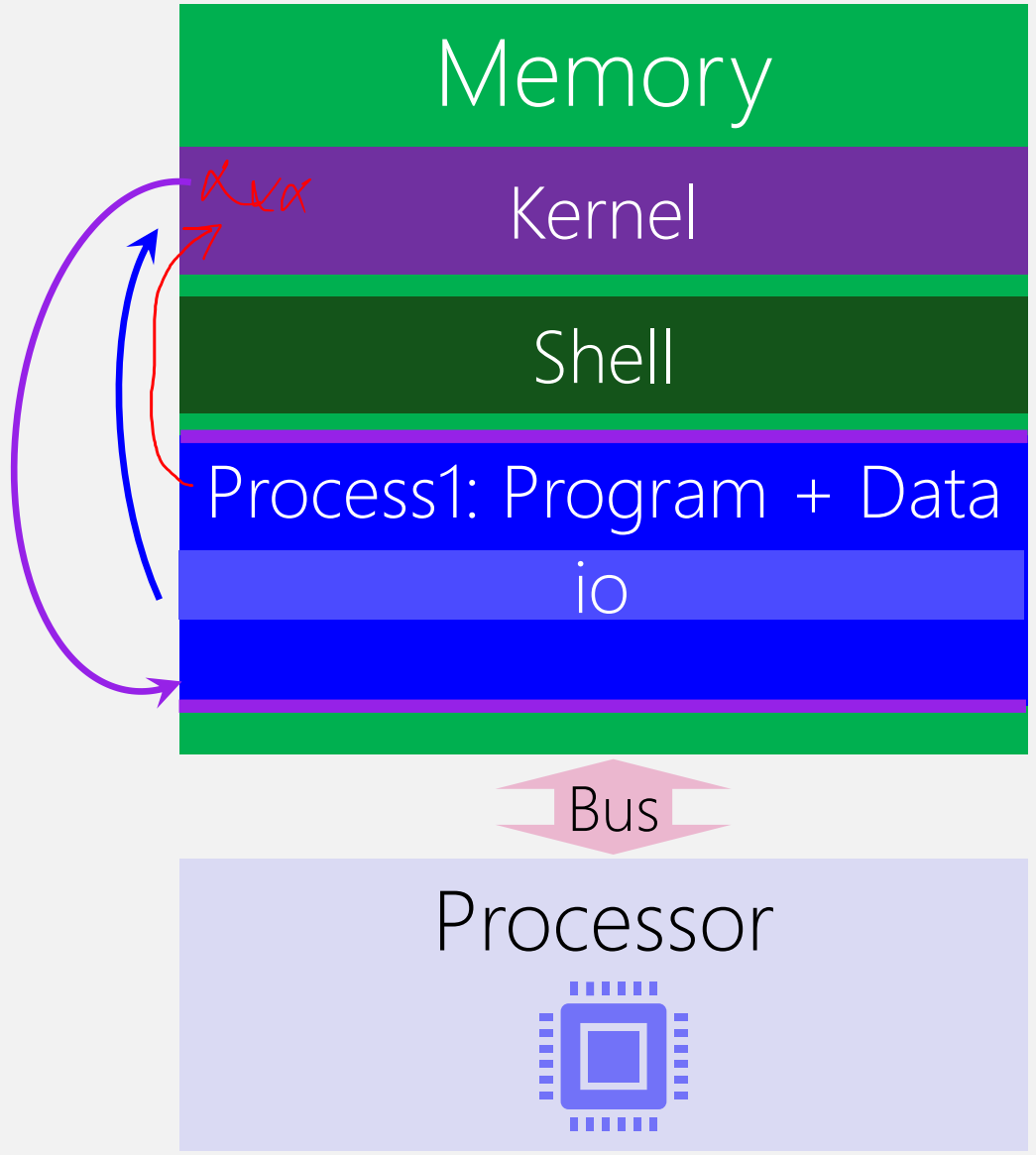

Shell
Kernel
Computer
Algorithm
Data
io
math
time
sys

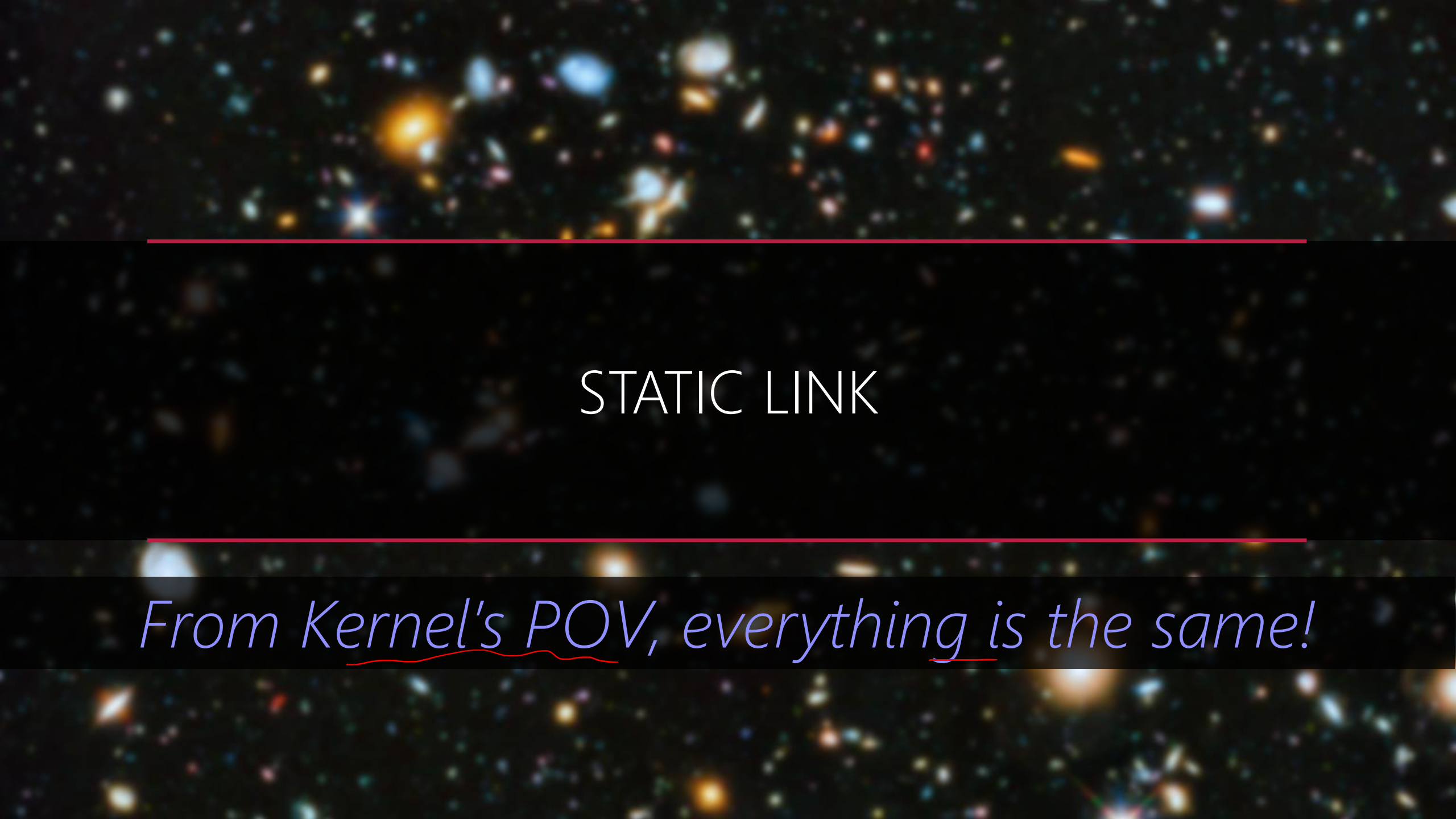
Computer

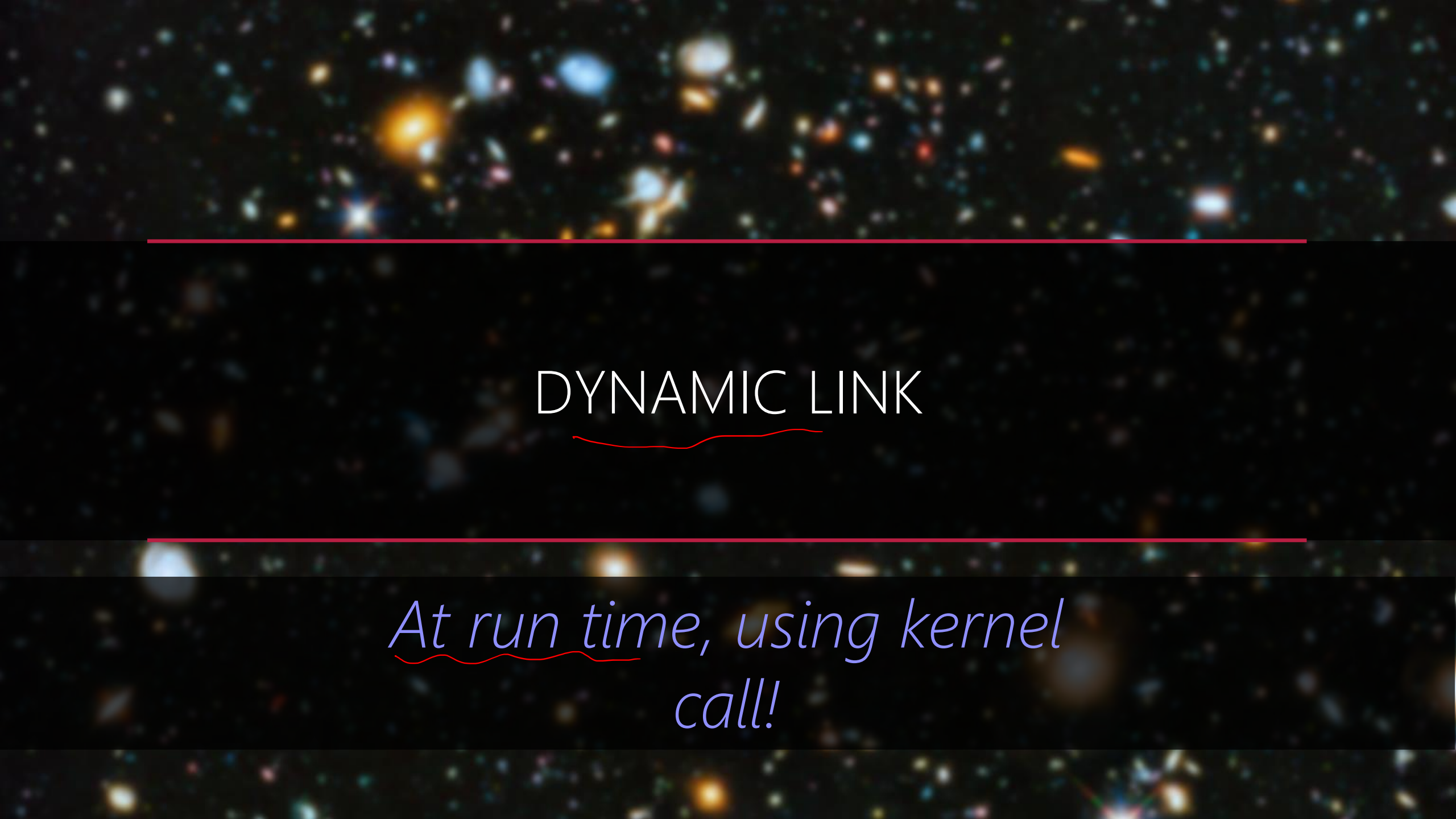

Memory
Kernel
Shell
Process1: Program + Data
io
Bus
Processor

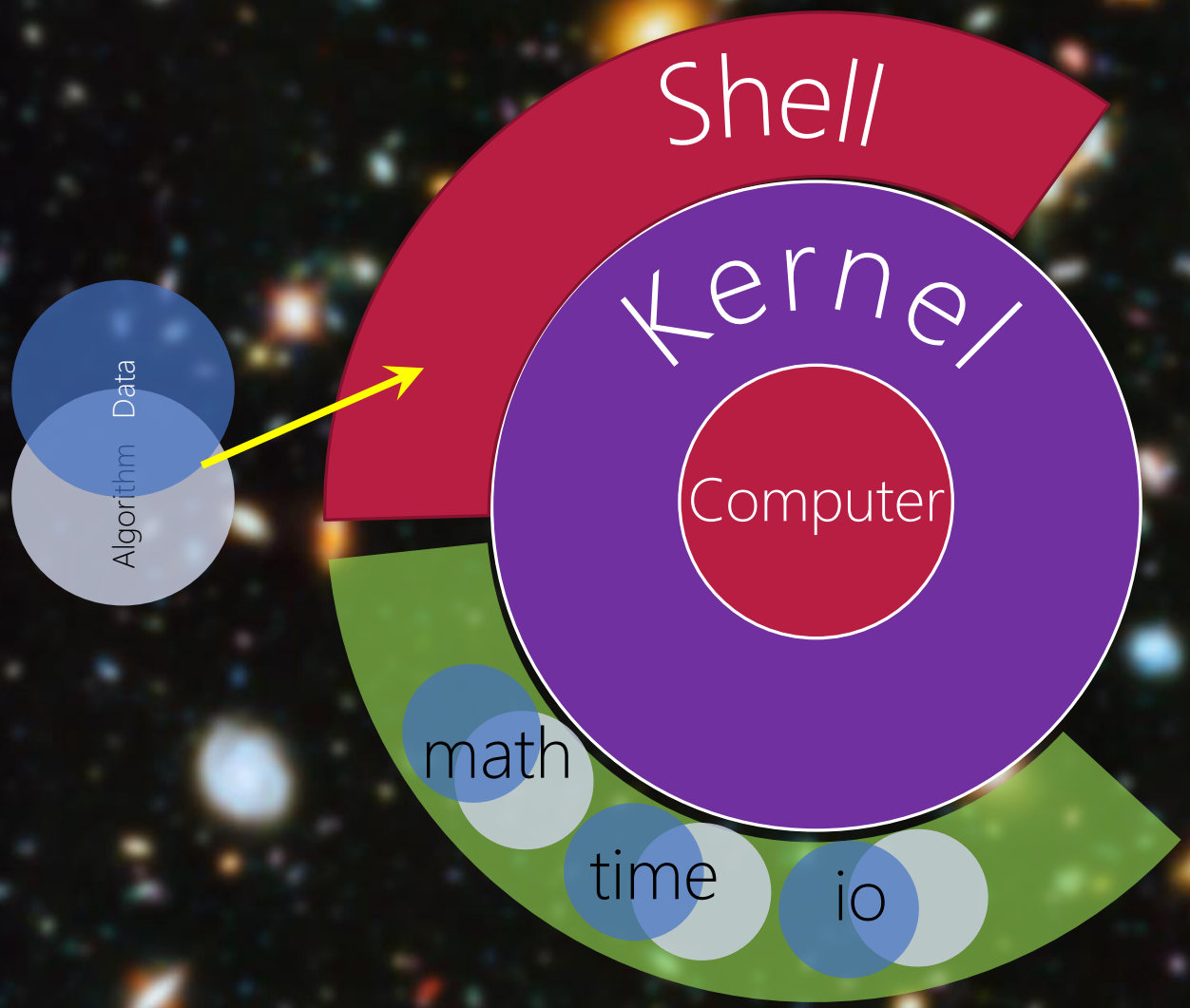

Shell
Kernel
Computer
Algorithm
Data
io
math
time
io

Computer

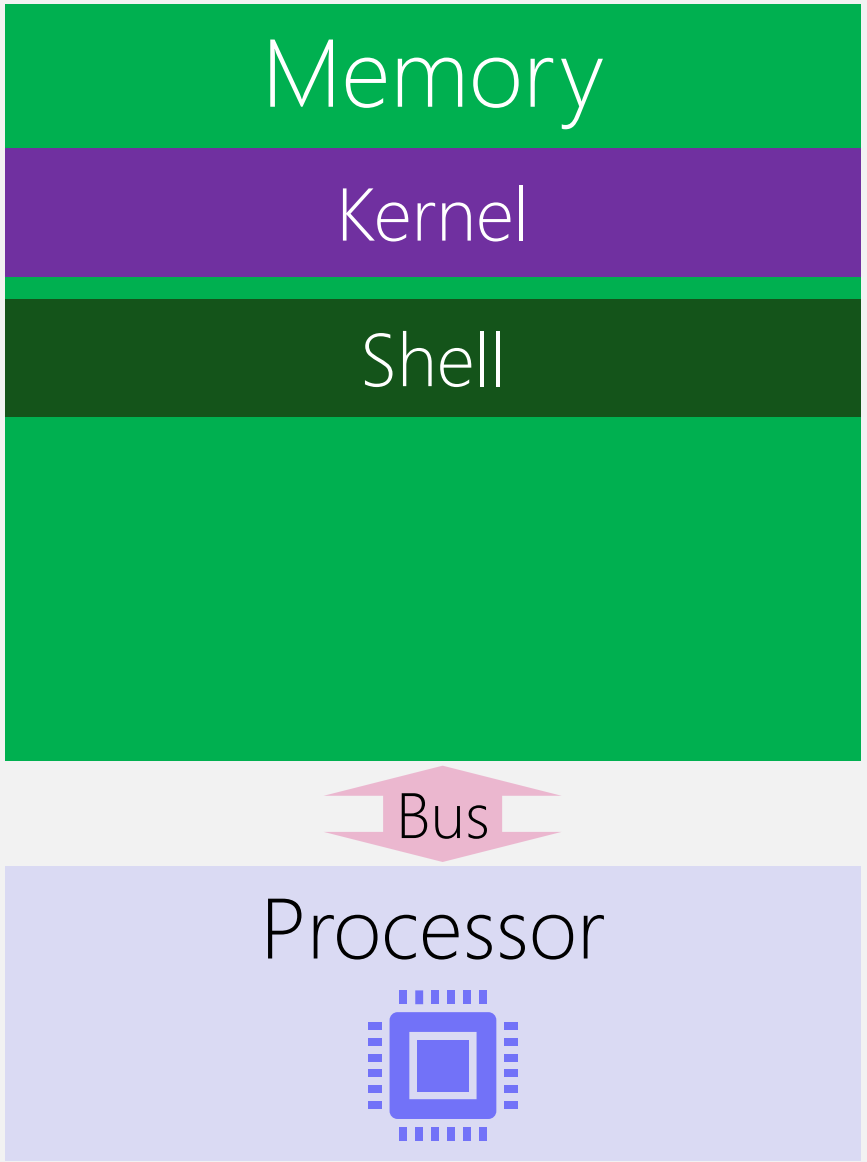

Memory
Kernel
Shell
Process1: Program + Data
io
Bus
Processor

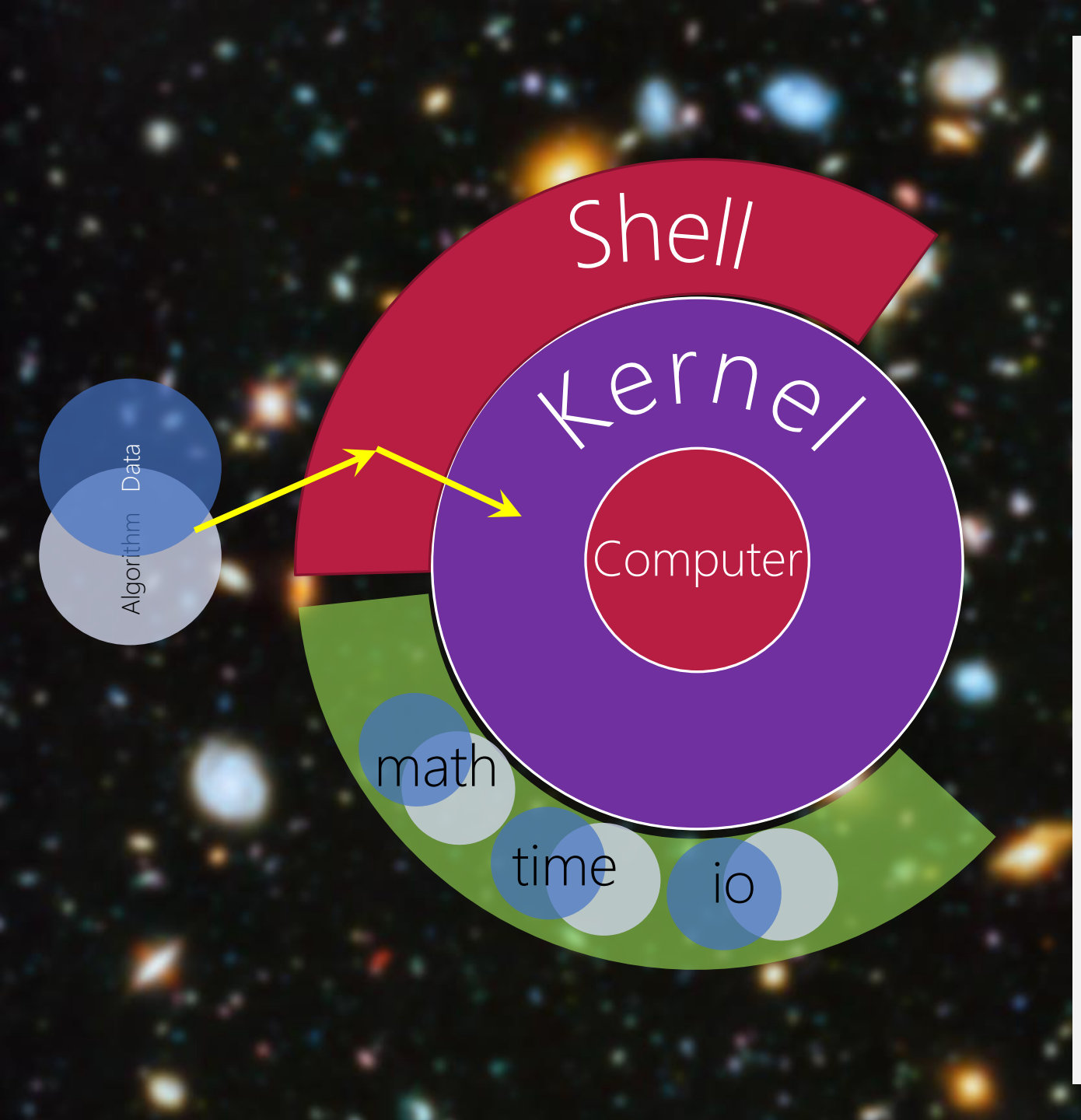

STATIC LINK
From Kernel's POV, everything is the same!


DYNAMIC LINK
At run time, using kernel call!

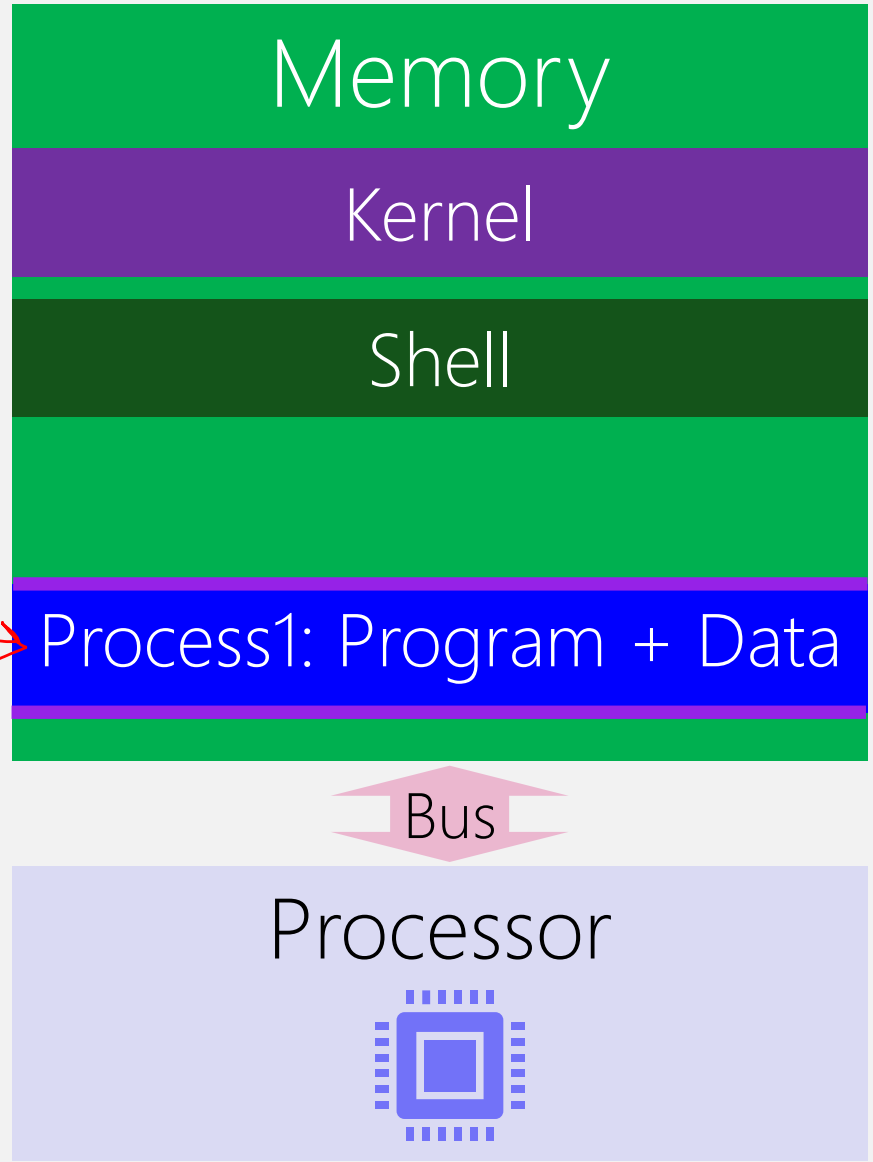

Shell
Kernel
Computer
Algorithm
Data
math
time
io

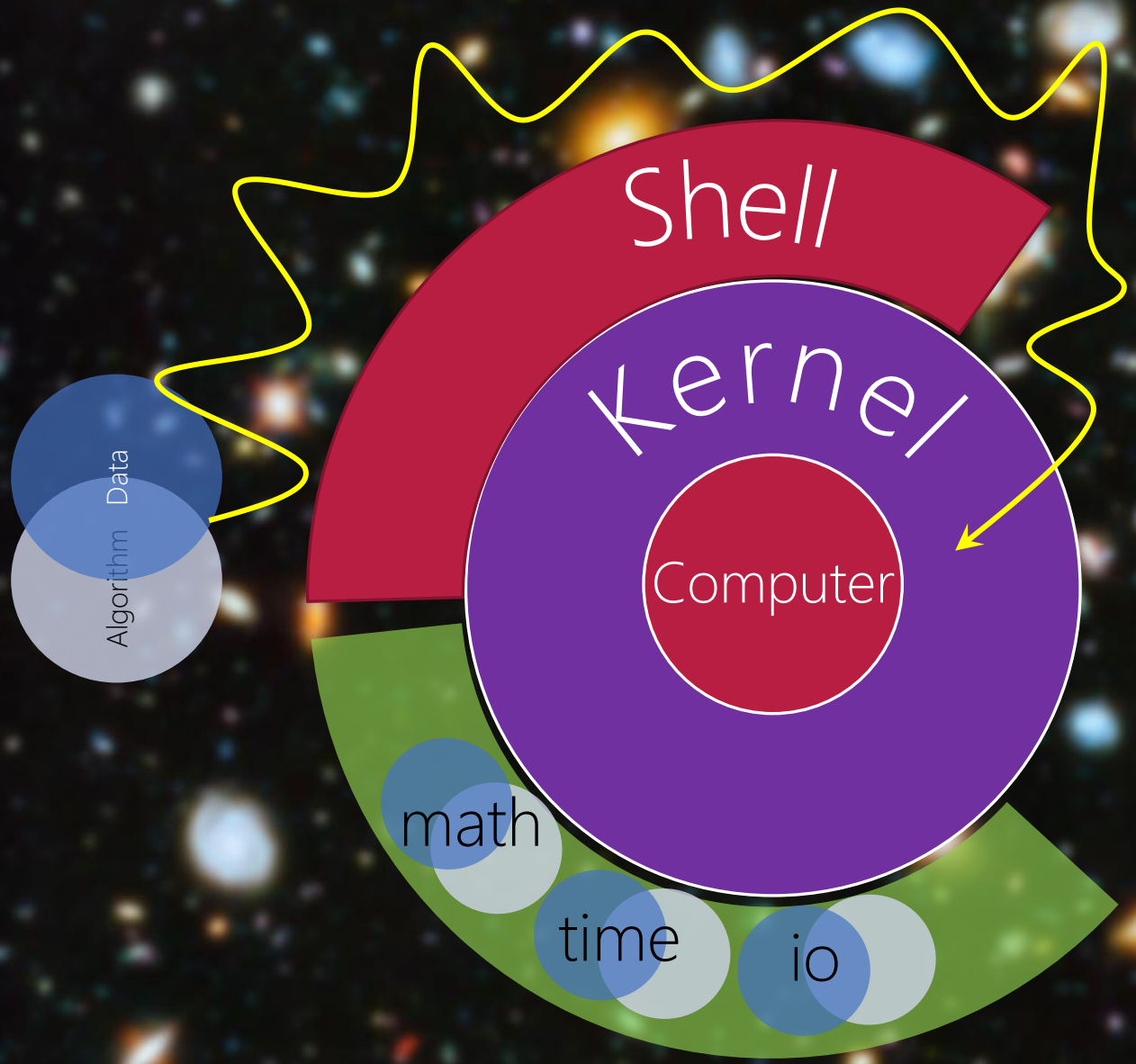
Computer

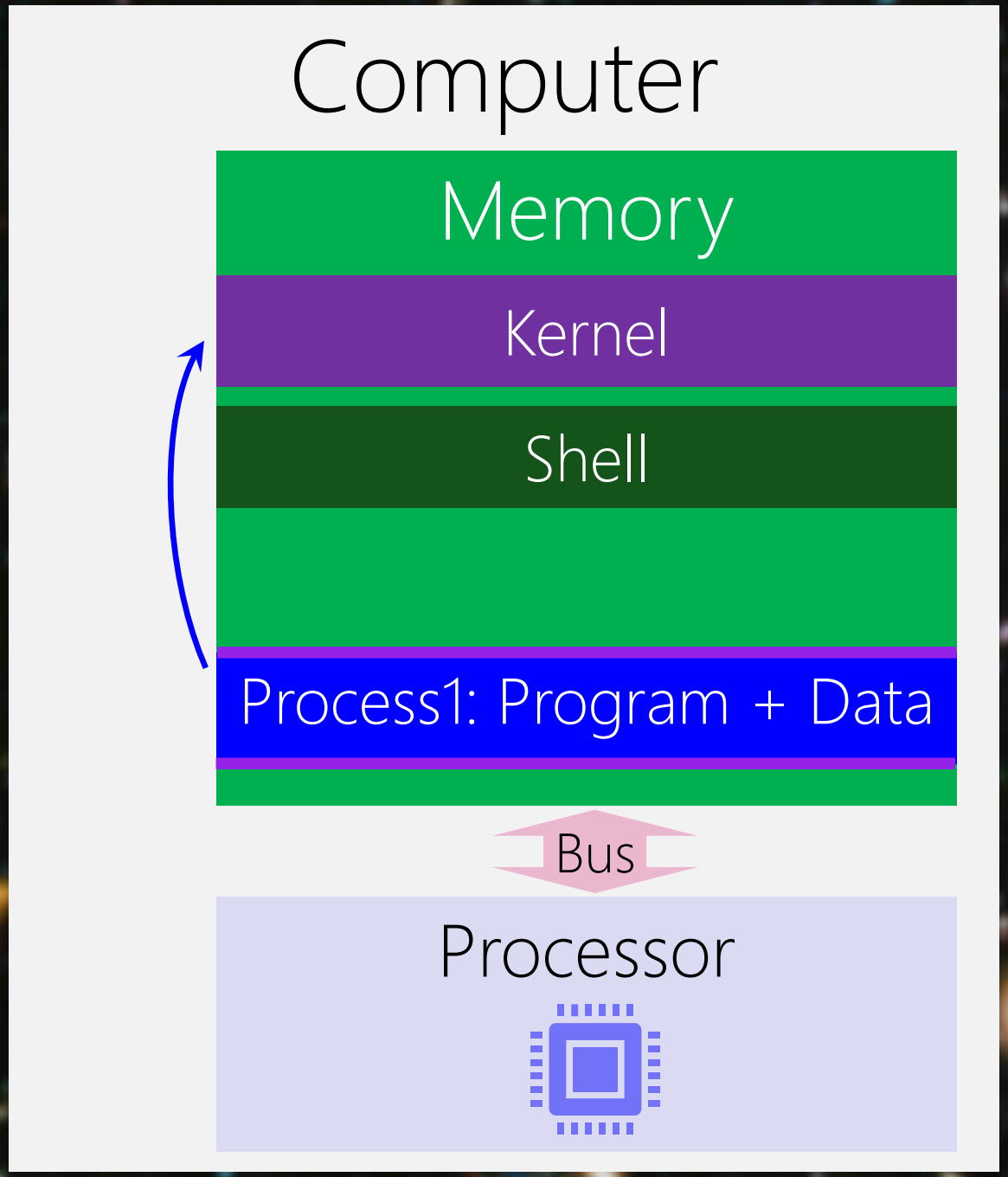

Memory
Kernel
Shell
Bus
Processor

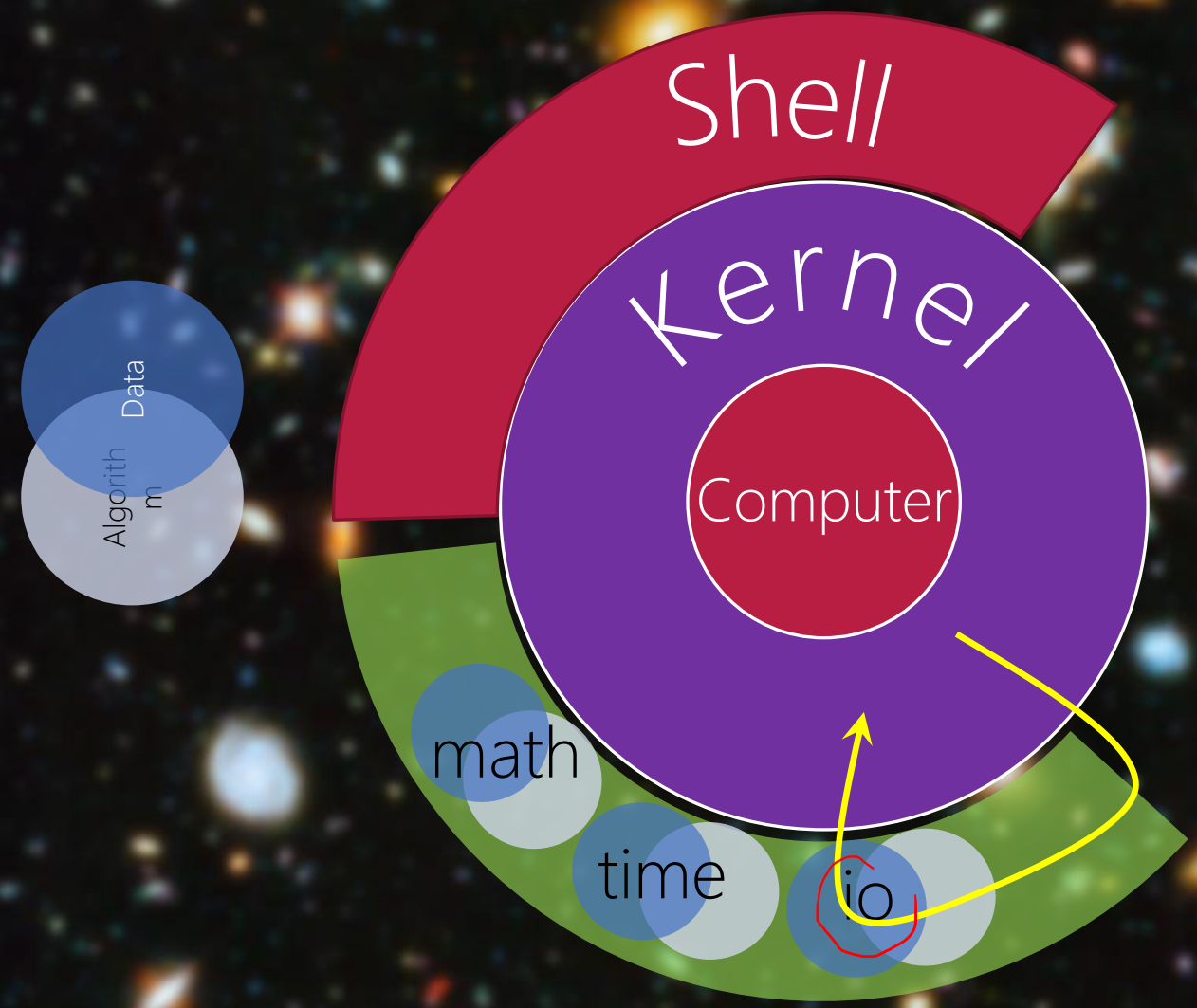

Shell
Kernel
Computer
Algorithm
Data
math
time
io

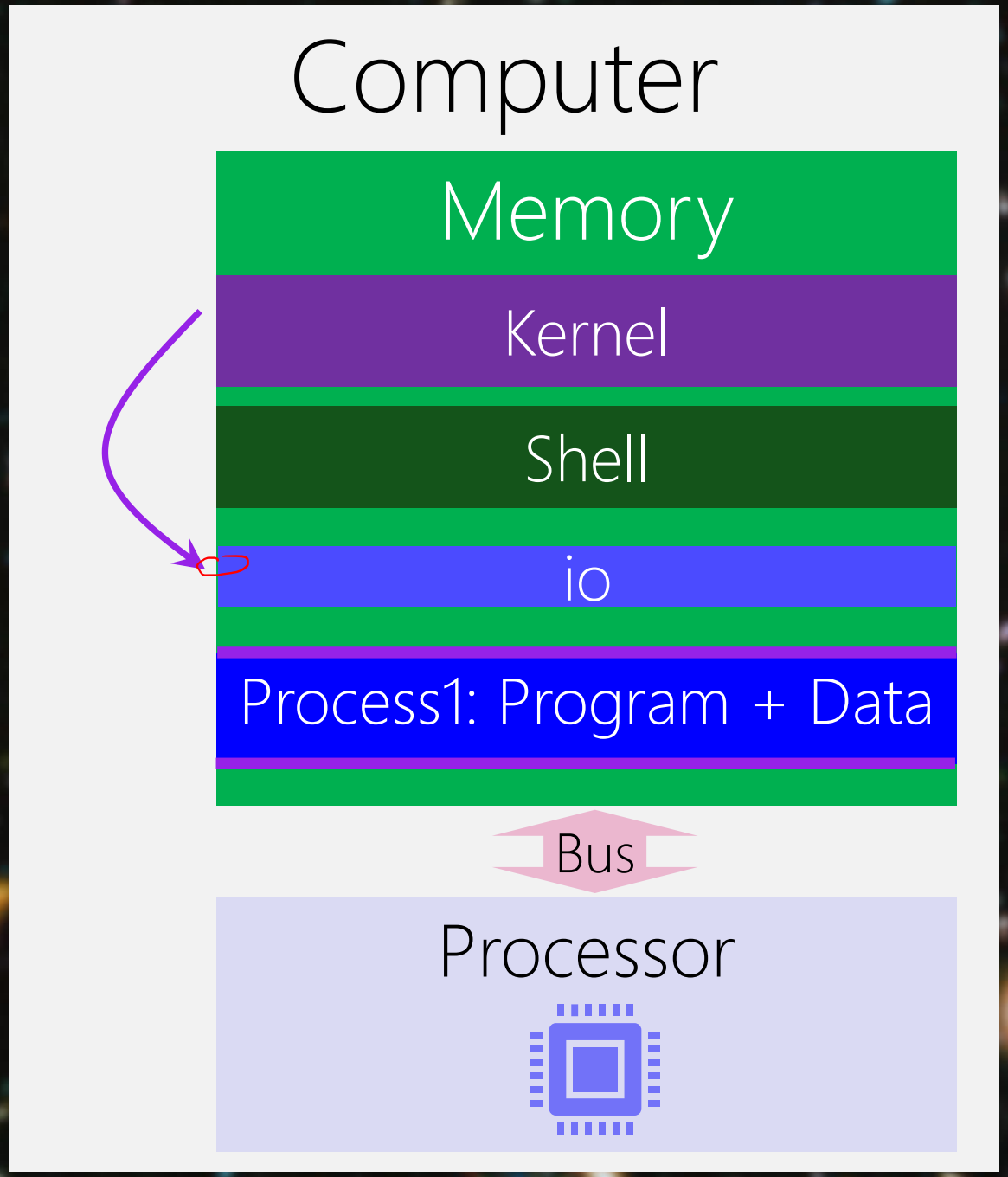
Computer

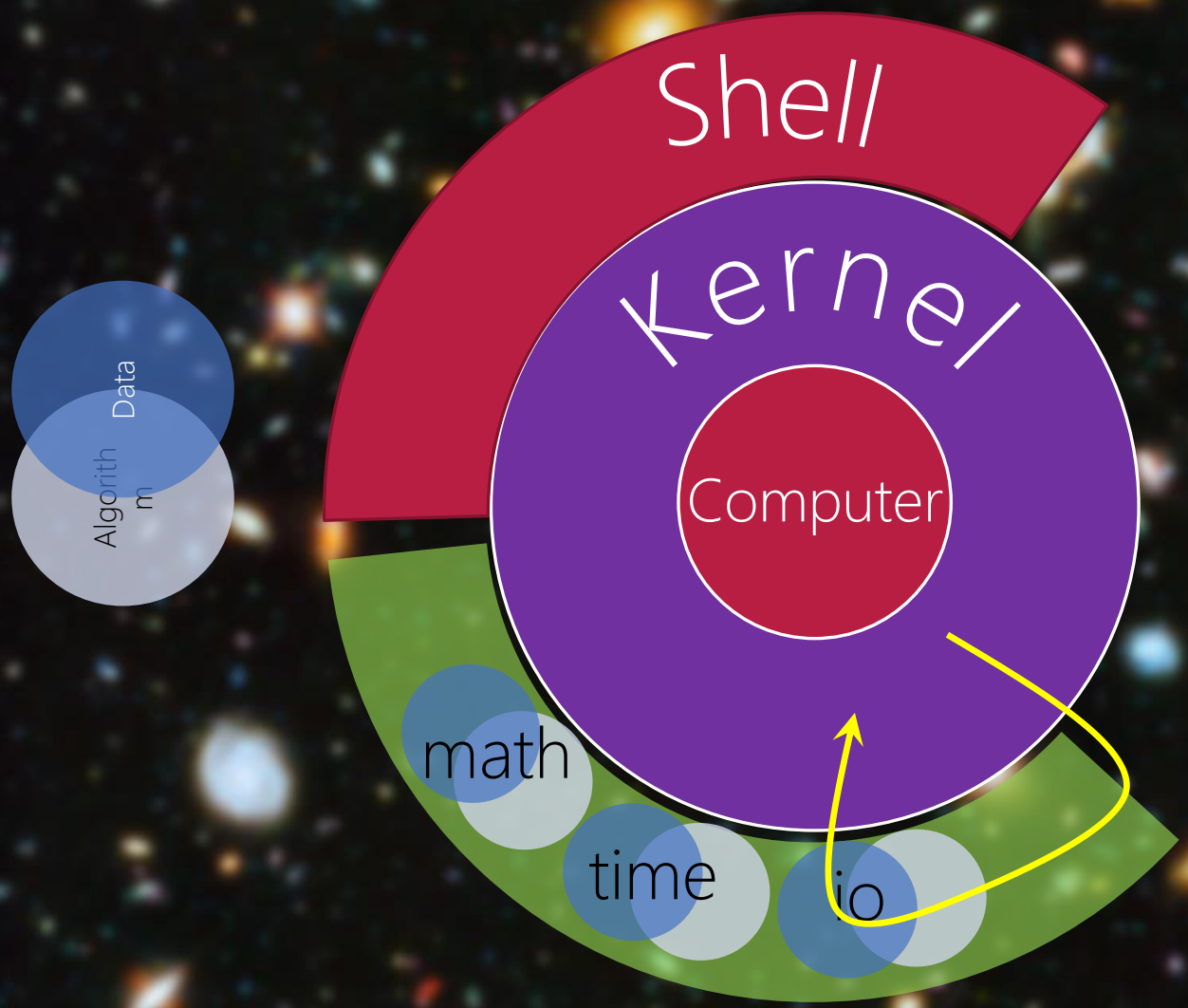

Memory
Kernel
Shell
Process1: Program + Data
Bus
Processor


Shell
Kernel
Computer
Algorithm
Data
math
time
io

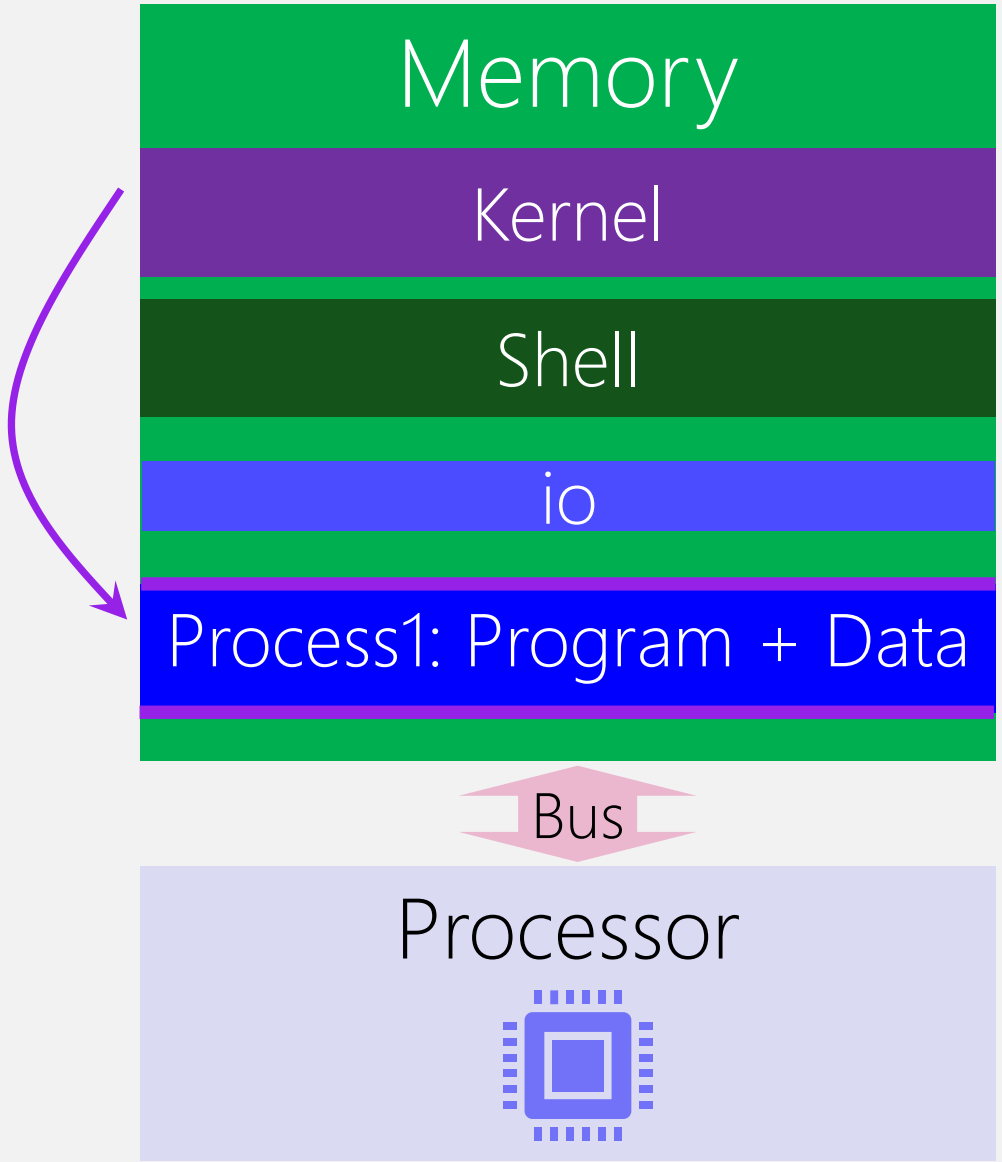

Computer
Memory
Kernel
Shell
Process1: Program + Data
Bus
Processor

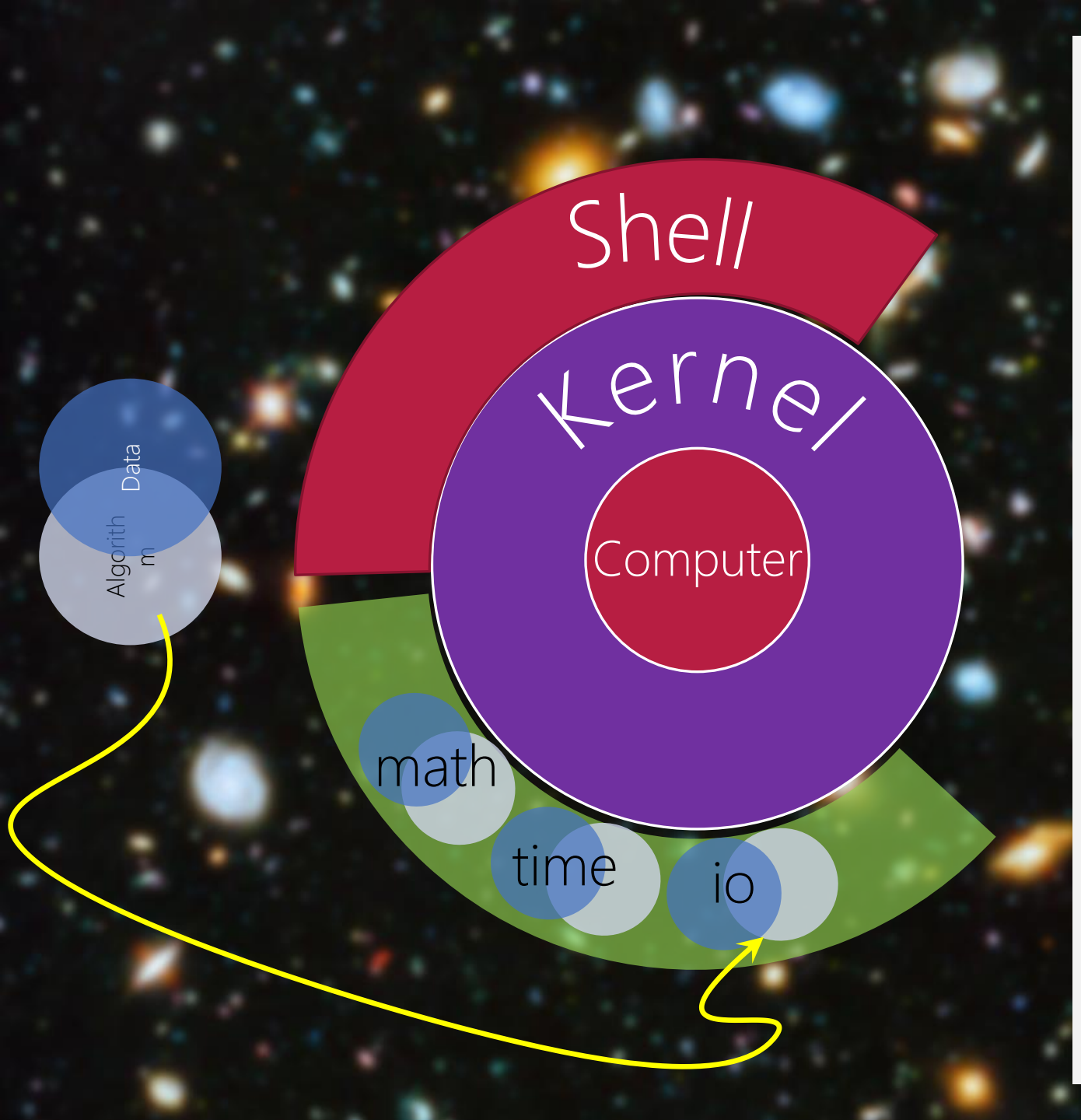

Shell
Kernel
Computer
Data
Algorithm
math
time
io


Computer
Memory
Kernel
Shell
io
Process1: Program + Data
Bus
Processor

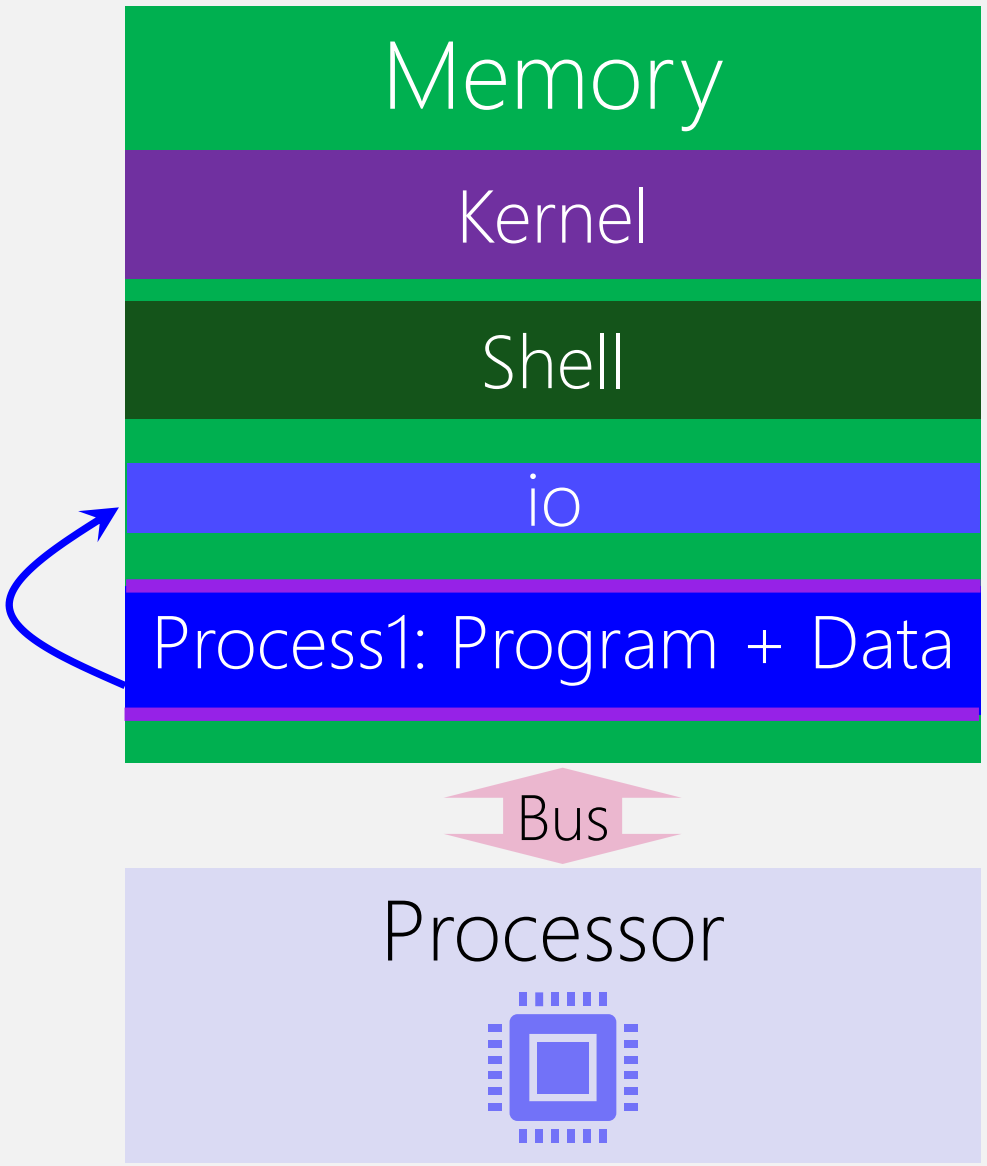

Shell
Kernel
Computer
math
time
io
Data
Algorithm

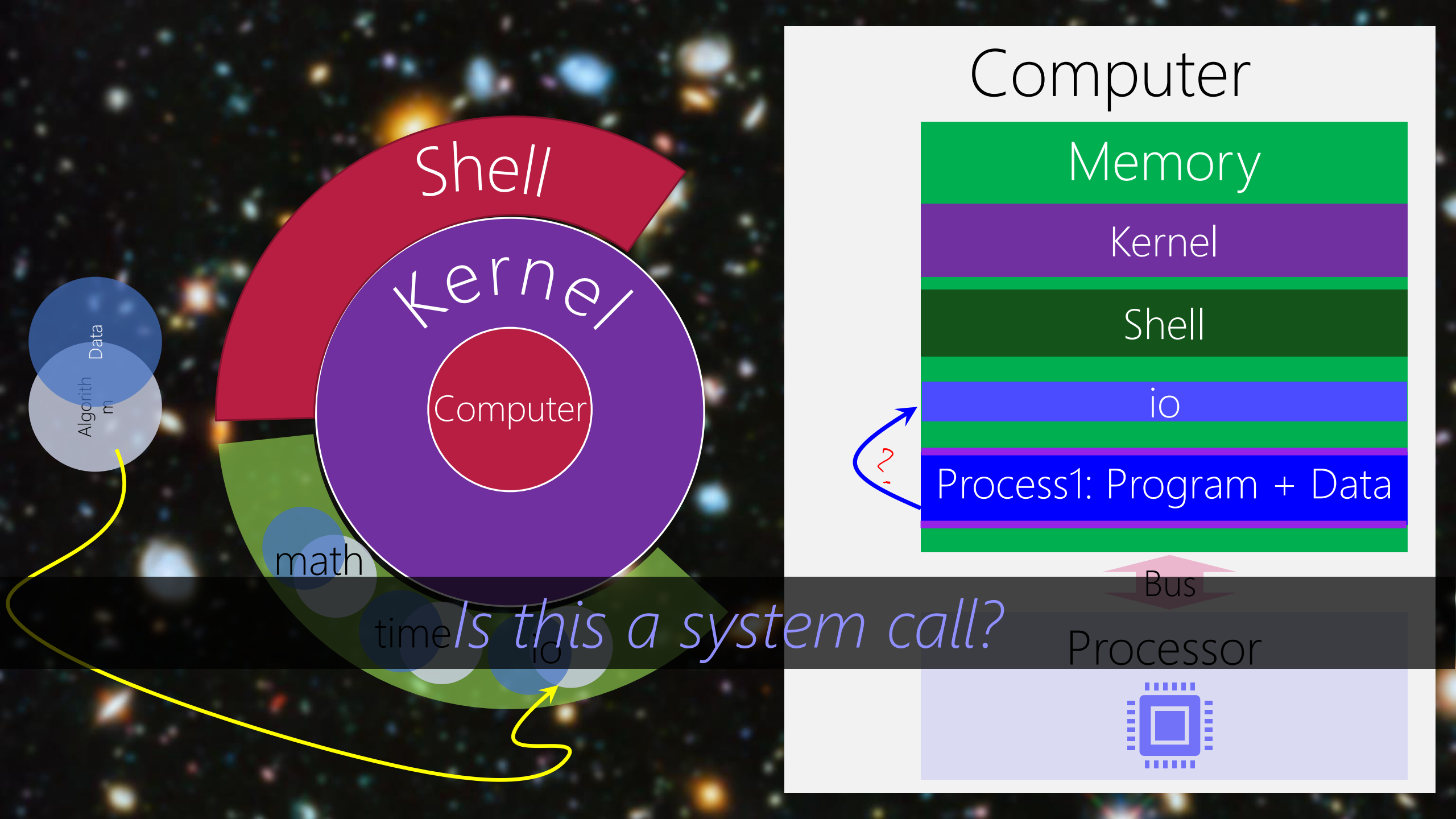
Computer

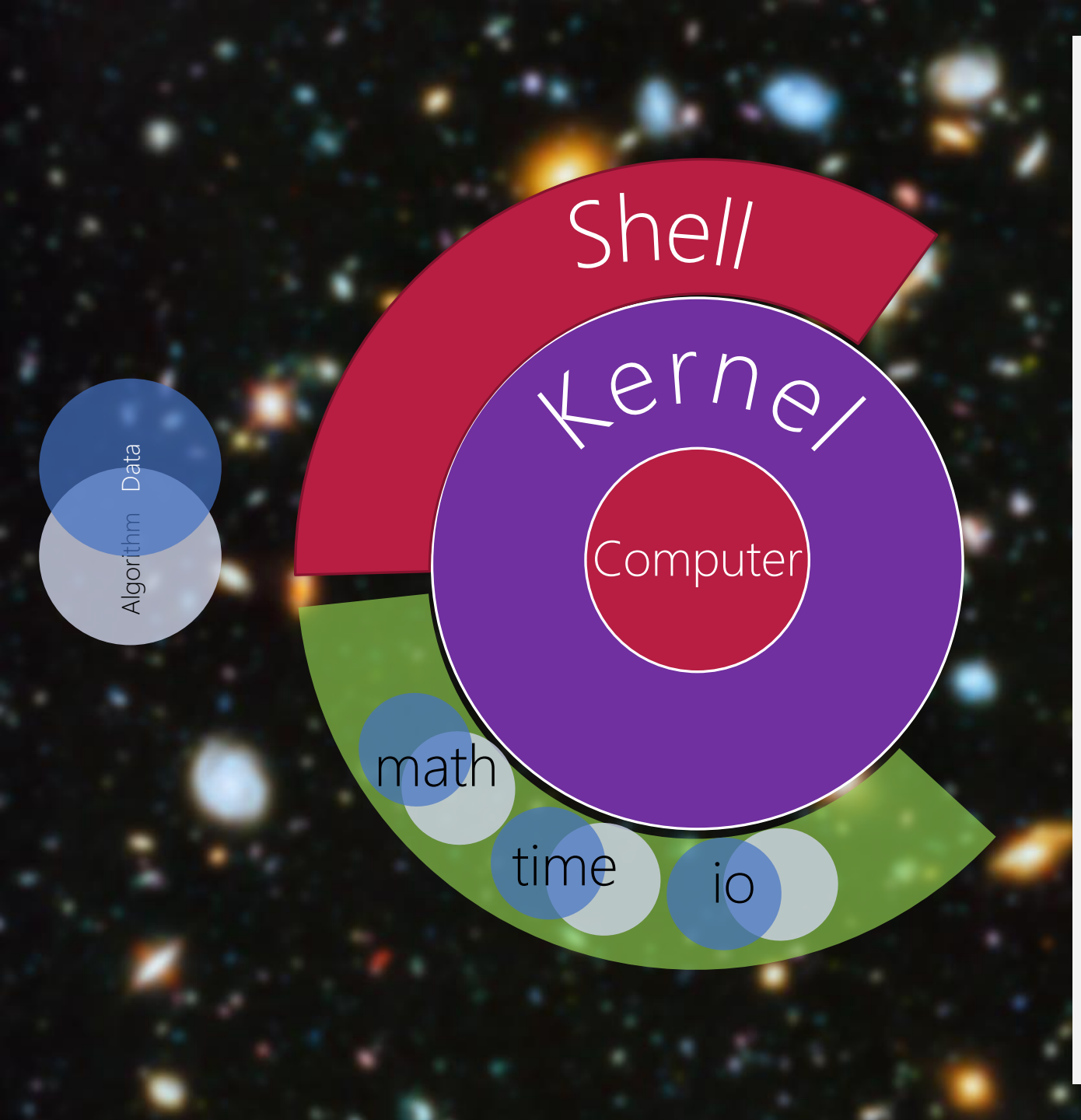

Memory
Kernel
Shell
io
Process1: Program + Data
Bus
Processor

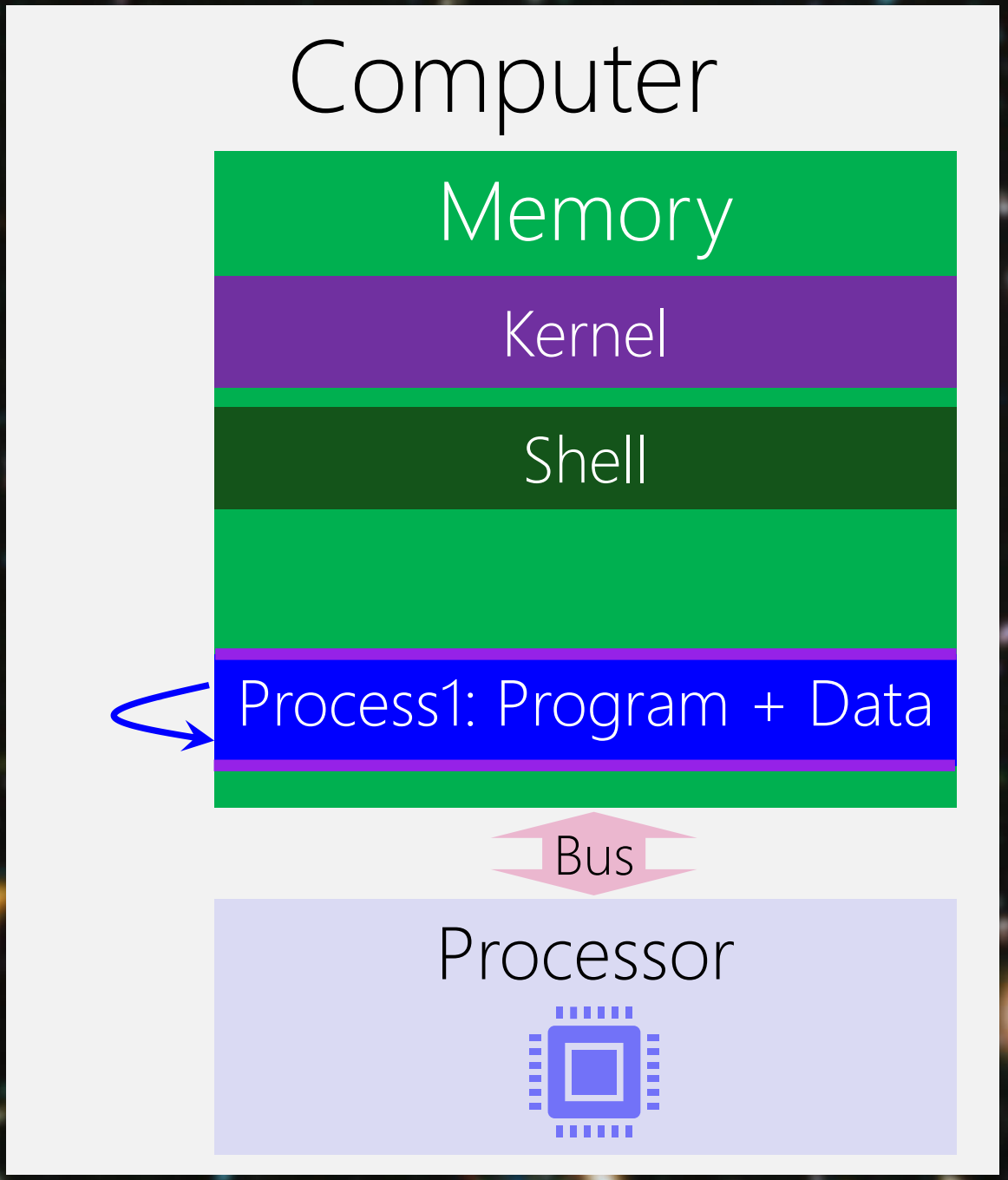

Shell
Kernel
Computer
Data
Algorithm
math
time
io

Computer


Memory
Kernel
Shell
io
Process1: Program + Data
Bus
Processor

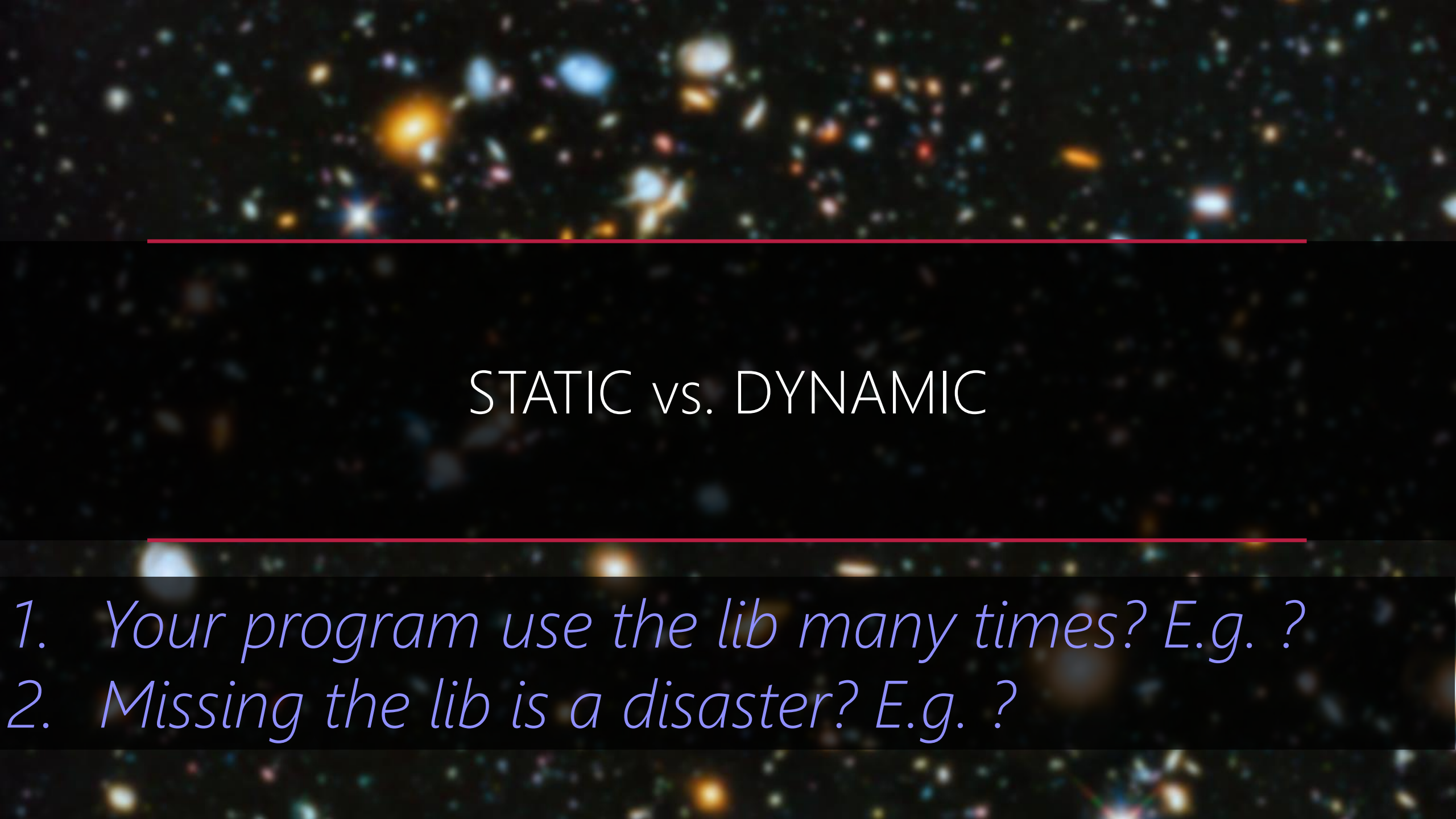

Shell
Kernel
Computer
Data
Algorithm
math
time
io
Computer
Memory
Kernel
Shell
io
Process1: Program + Data
Bus
Processor
Is this a system call?


Shell
Kernel
Computer
Algorithm
Data
math
time
io


Computer
Memory
Kernel
Shell
Process1: Program + Data
Bus
Processor

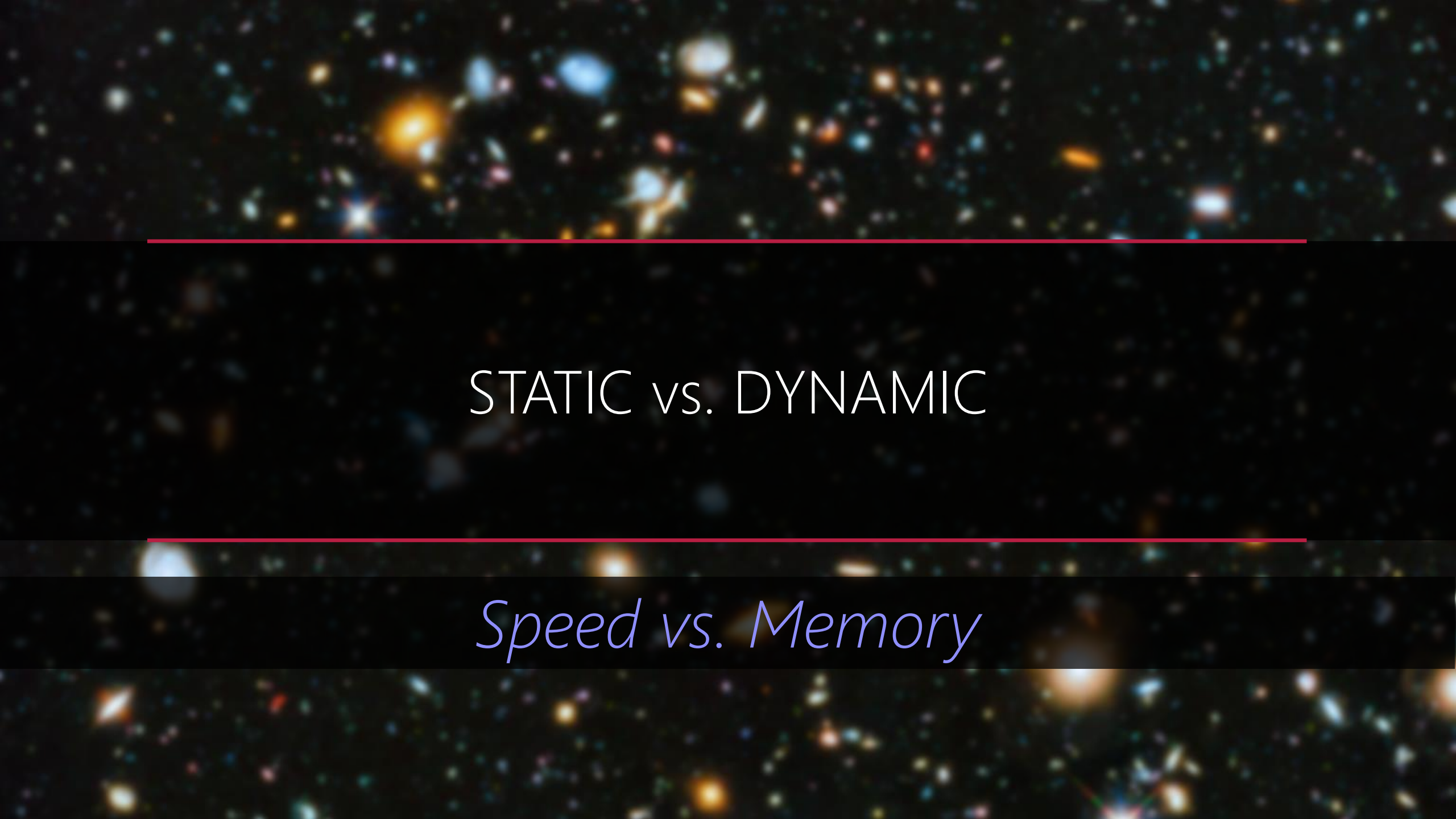
STATIC vs. DYNAMIC


Into the Wild (2007) - Sean Penn



STATIC vs. DYNAMIC

1. *Your program use the lib many times? E.g. ?*
2. *Missing the lib is a disaster? E.g. ?*


STATIC vs. DYNAMIC
Speed vs. Memory

1) Building the Static Library

```c
int add(int x, int y){
        return x + y;
}
```

```c
int sub(int x, int y){
        return x - y;
}
```

multiply

sqrt

Roots of a quadratic equation!

```
cc add.c -c
```

```
0001 0000 0000 0000 0000 0000 0000 0000
0011 0000 0003 0000 0000 0000 0000 0000
0000 0000 0000 0000 3309 0000 0000 0000
0096 0000 0000 0000 0000 0000 0000 0000
```

add.o

```
cc sub.c -c
```

```
0111 0110 0000 0000 0000 0000 0000 0000
0011 0000 0003 0000 0000 0000 0000 0000
0000 0000 0000 0000 3309 0000 0000 0000
0096 0000 0000 0000 0000 0000 0000 0000
```

sub.o

```
cc multi.c -c
```

```
0001 0000 0000 0000 0000 0000 0000 0000
0011 0000 0002 0000 0000 0000 0000 0000
0000 0000 1111 0000 3309 0000 0000 0000
0086 0000 0000 0000 0000 0000 0000 0000
```

multi.o

```
cc sqrt.c -c
```

```
0001 0000 0000 0000 0000 0000 0000 0000
0011 0000 0003 0000 0000 0000 0000 0000
0000 0000 0000 0000 2222 0000 0000 0000
0096 0000 0000 0000 0000 0000 0000 0000
```

sqrt.o

```
0001 0000 0000 0000 0000 0000 0000 0000
0011 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 5641 0000 0000 0000
0090 0000 0000 0000 0000 0000 0000 0000
```

Optionally

```
int add(int x, int y){
        return x + y;
}
```

```
cc add.c -c
```

```
0001 0000 0000 0000 0000 0000 0000 0000
0011 0000 0003 0000 0000 0000 0000 0000
0000 0000 0000 0000 3309 0000 0000 0000
0096 0000 0000 0000 0000 0000 0000 0000
```

add.o

```
int sub(int x, int y){
        return x - y;
}
```

```
cc sub.c -c
```

```
0111 0110 0000 0000 0000 0000 0000 0000
0011 0000 0003 0000 0000 0000 0000 0000
0000 0000 0000 0000 3309 0000 0000 0000
0096 0000 0000 0000 0000 0000 0000 0000
```

sub.o

multiply

```
cc multi.c -c
```

```
0001 0000 0000 0000 0000 0000 0000 0000
0011 0000 0002 0000 0000 0000 0000 0000
0000 0000 1111 0000 3309 0000 0000 0000
0086 0000 0000 0000 0000 0000 0000 0000
```

multi.o

sqrt

```
cc sqrt.c -c
```

```
0001 0000 0000 0000 0000 0000 0000 0000
0011 0000 0003 0000 0000 0000 0000 0000
0000 0000 0000 0000 2222 0000 0000 0000
0096 0000 0000 0000 0000 0000 0000 0000
```

sqrt.o

Roots of a quadratic equation!

```
0001 0000 0000 0000 0000 0000 0000 0000
0011 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 5641 0000 0000 0000
0090 0000 0000 0000 0000 0000 0000 0000
```

Archive

Object files to archive

```
-$ ar rs mathlib.a add.o sub.o multi.o sqrt.o
```

Name of the archive file
Convention: *.a

2) Using the Static Library

```c
#include <stdio.h>

int add(int, int);
int sub(int, int);
int multi(int, int);
int sqrt(int);
...

void main(void){
        printf("2 + 3 = %d\n", add(2,3));
        printf("2 - 3 = %d\n", sub(2,3));
        ...

}
```

Functions prototypes

Function calls

```c
#include <stdio.h>

int add(int, int);
int sub(int, int);
int multi(int, int);
int sqrt(int);
...

void main(void){
        printf("2 + 3 = %d\n", add(2,3));
        printf("2 - 3 = %d\n", sub(2,3));
        ...

}
```

Functions prototypes into another file → Library Header File → mathlib.h

```
:~$ vi mathlib.h
int add(int, int);
int sub(int, int);
int multi(int, int);
int sqrt(int);
//...
~
~
```

Optionally

```c
#include <stdio.h>

#include "mathlib.h"
void main(void){
        printf("2 + 3 = %d\n", add(2,3));
        printf("2 - 3 = %d\n", sub(2,3));
}
```

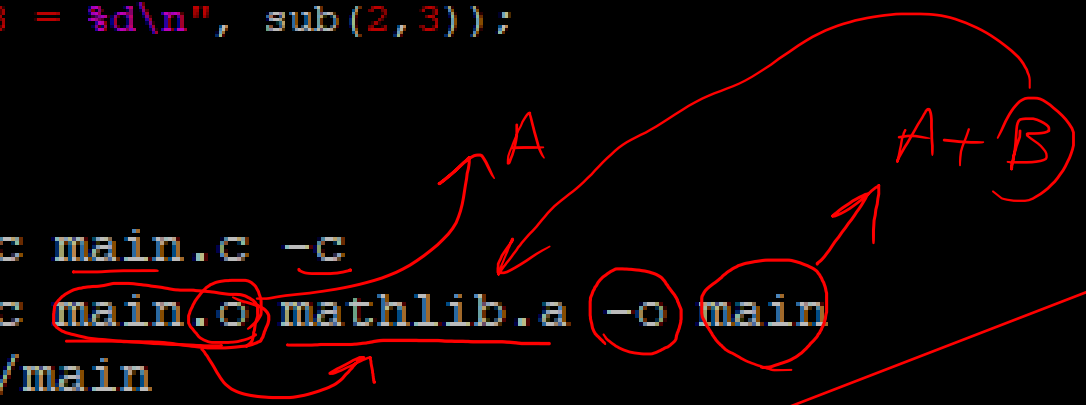
```
:~$ vi mathlib.h
int add(int, int);
int sub(int, int);
int multi(int, int);
int sqrt(int);
//...
~
~
```

Optionally

```c
#include <stdio.h>

#include "mathlib.h"
void main(void){
        printf("2 + 3 = %d\n", add(2,3));
        printf("2 - 3 = %d\n", sub(2,3));
}
```

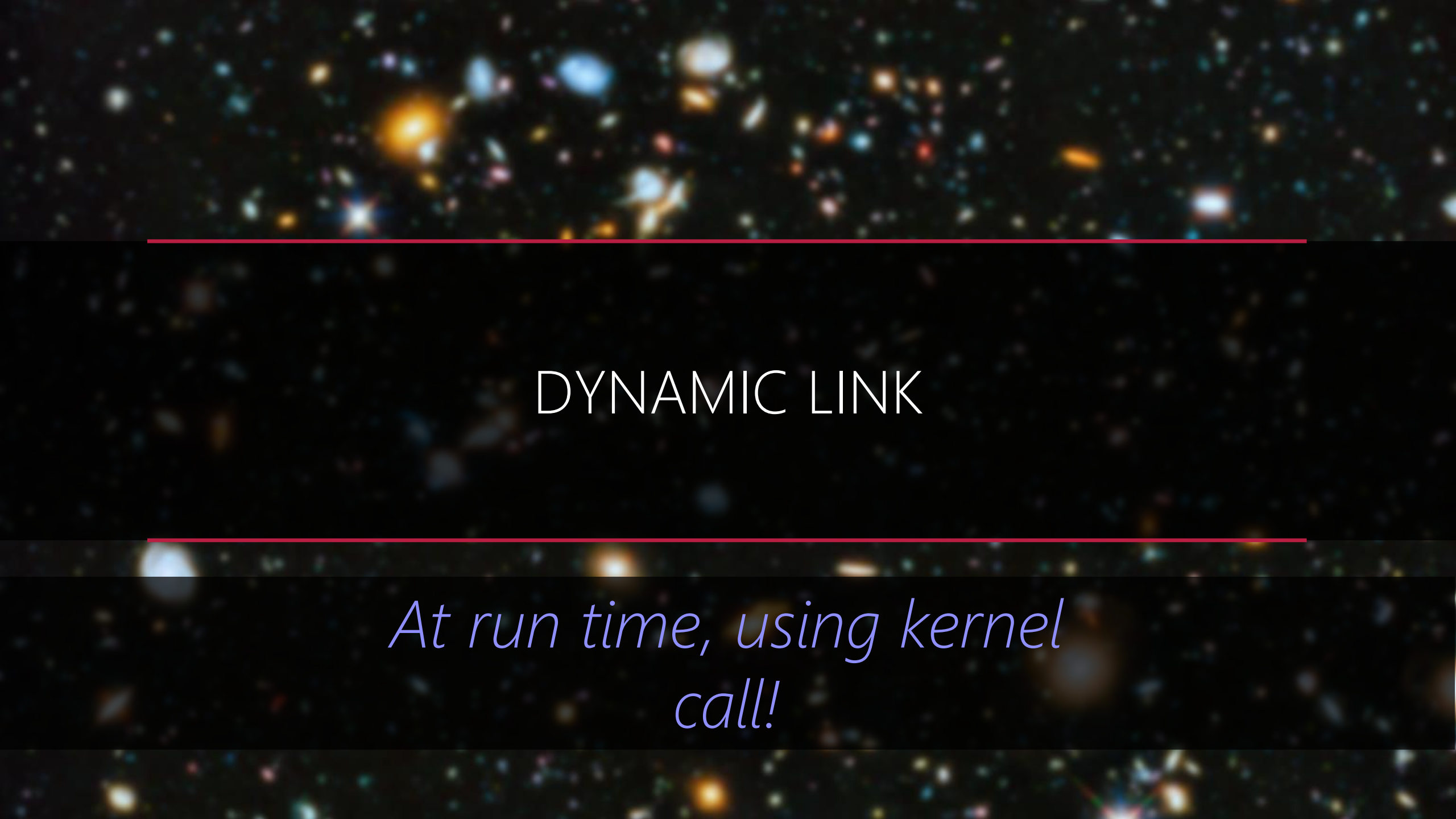
```
hfani@alpha:~$ cc main.c -c
hfani@alpha:~$ cc main.o mathlib.a -o main
hfani@alpha:~$ ./main
2 + 3 = 5
2 - 3 = -1
```



Includes main.o as well as
object files of mathlib.a that are used

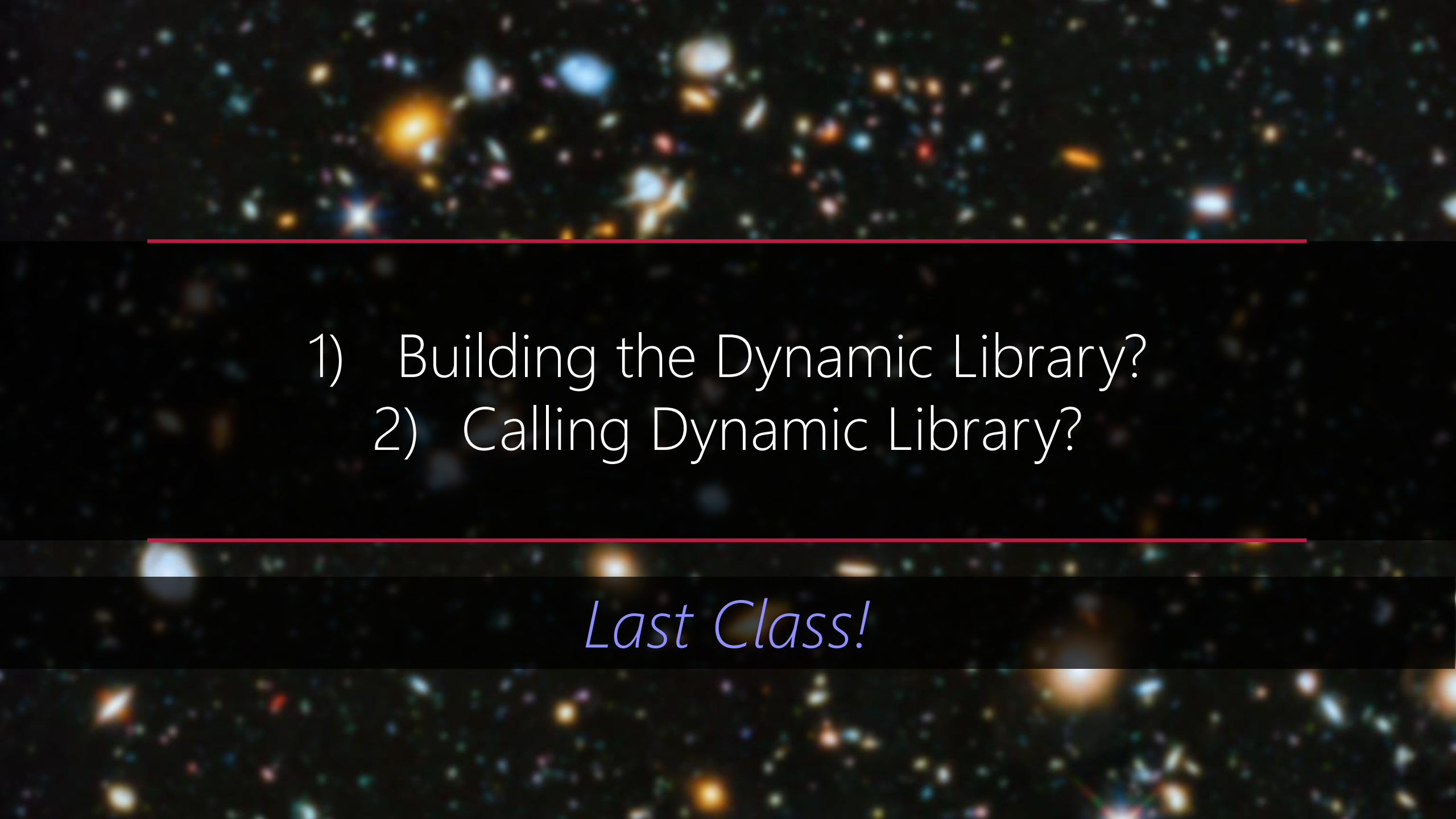
main.o
add.o
sub.o

```
hfani@alpha:~$ size ./main
   text    data     bss     dec     hex filename
   1647     584       8    2239     8bf ./main
```



DYNAMIC LINK

At run time, using kernel call!



1) Building the Dynamic Library?
2) Calling Dynamic Library?

Last Class!