Partial Marks toward Total Marks

Unmarked or yet-to-be-marked submissions are considered 0 for now!

Discussion Board toward Bonus Closes before Final Exam

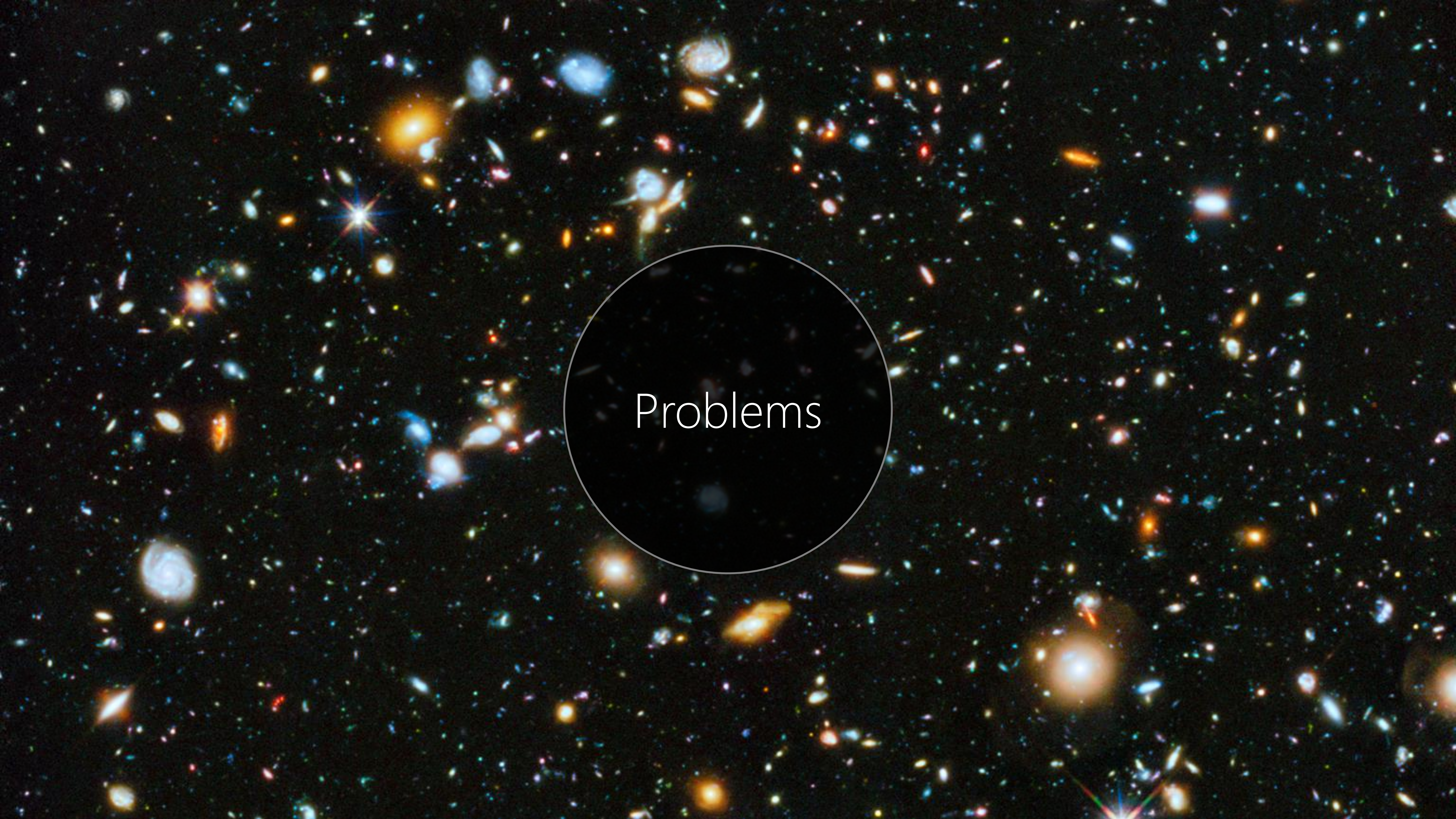
Please do not post spams!

Blue Pill or Red Pill, The Matrix (1999), Lana & Lilly Wachowski
https://youtu.be/zE7PKRjrid4?t=57

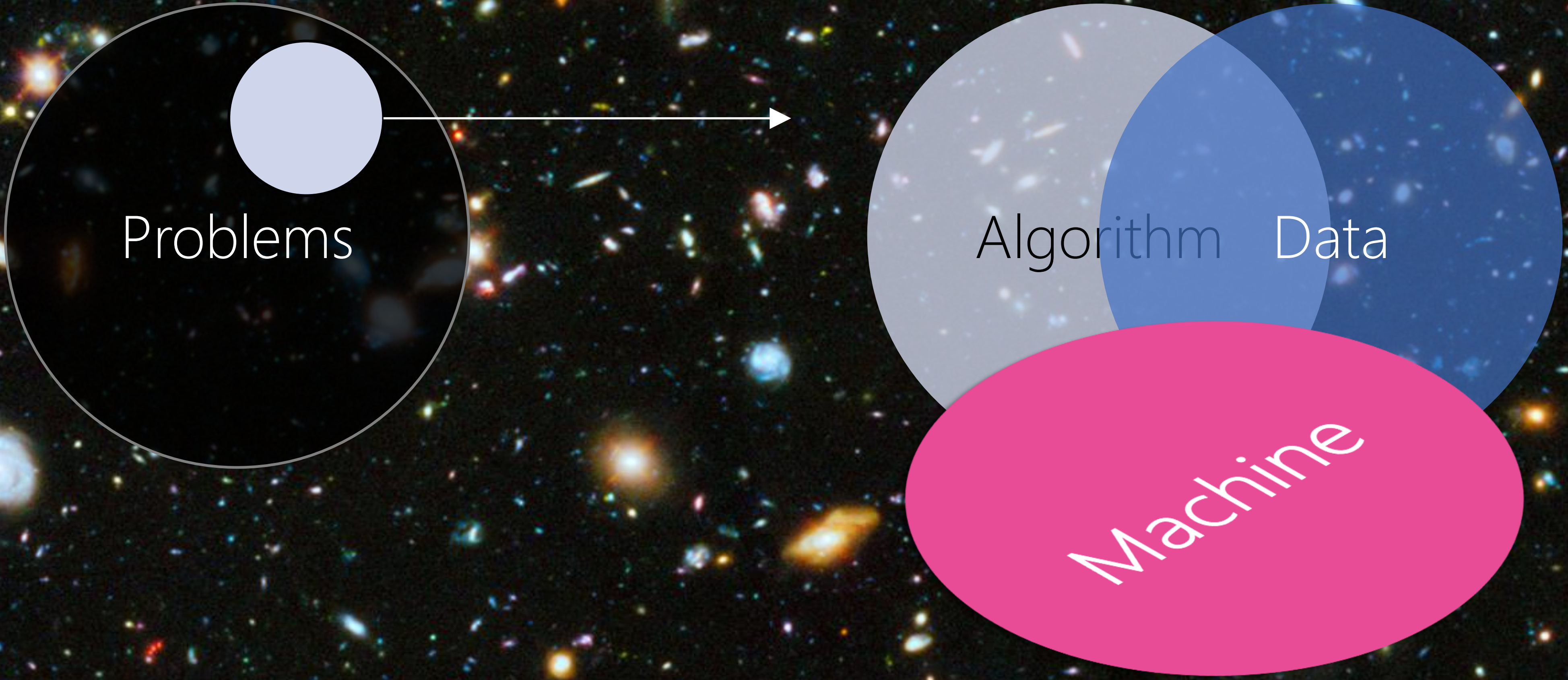
HUBBLE UNVEILS ITS MOST COLORFUL VIEW OF THE UNIVERSE (ZOOM AND PAN)

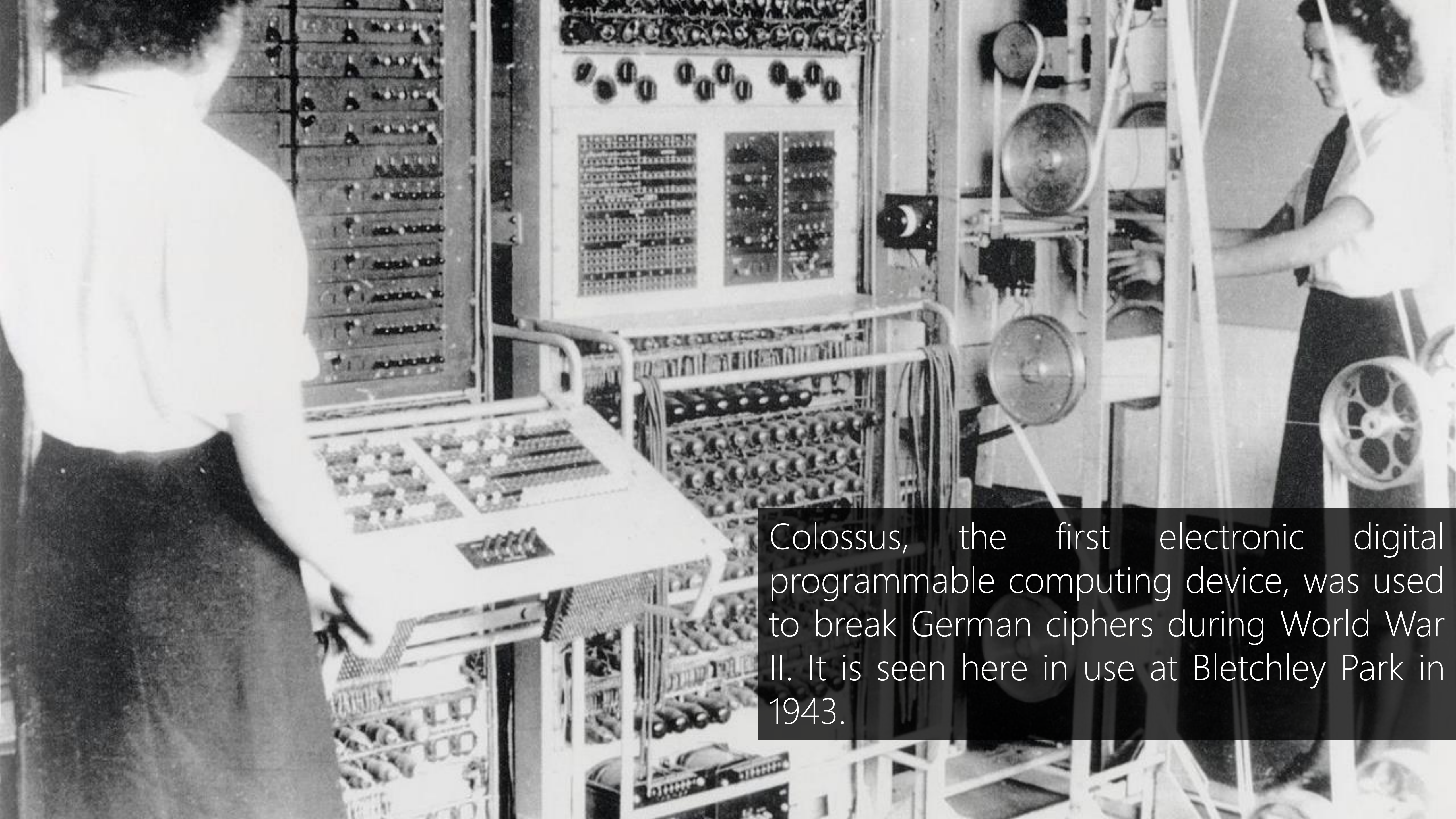
BROWSE THE LINK BELOW AND PLAY!

https://hubblesite.org/contents/media/videos/2014/27/766-Video.html



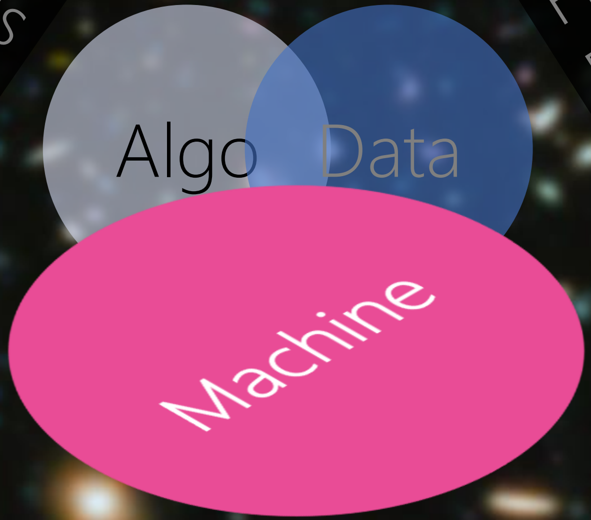
Problems


Problems
Algorithm
Data
Machine


Colossus, the first electronic digital programmable computing device, was used to break German ciphers during World War II. It is seen here in use at Bletchley Park in 1943.

Algorithm Design
Algorithm Analysis
Artificial Intelligence (AI)
Machine Learning
Data Mining

Data Structure
File Structure
Database Management Sys.
Data Warehouse
Big Data
Cloud



Digital Design (Logic Circuits)
Computer Architecture
Assembly Language
Operating Systems

John von Neumann

(/vɒn ˈnɔɪmən/)

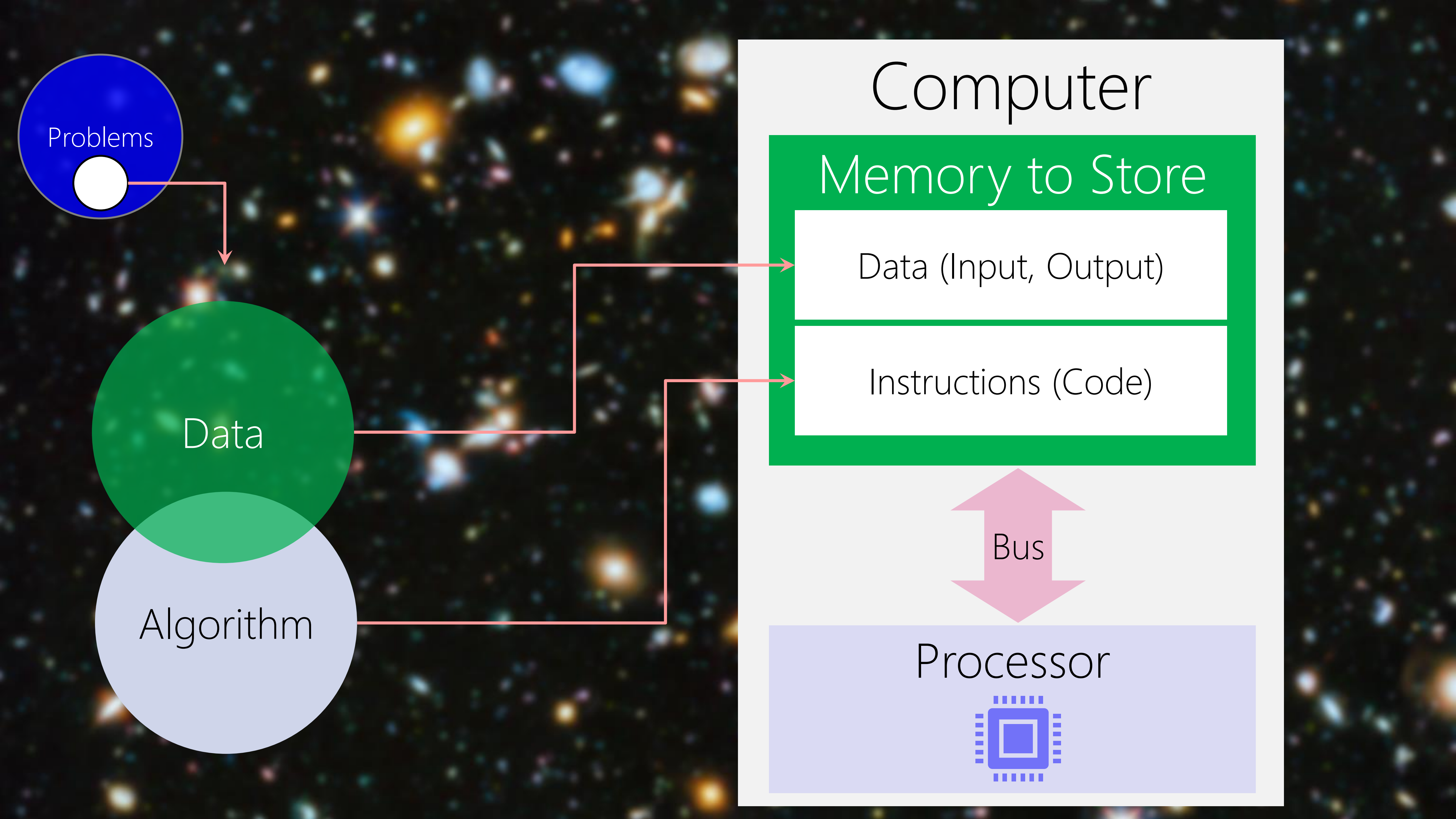
1903 –1957

Mathematician, Physicist, Computer Scientist, Engineer

Polymath

He integrated pure and applied sciences. He made major contributions to many fields, including:

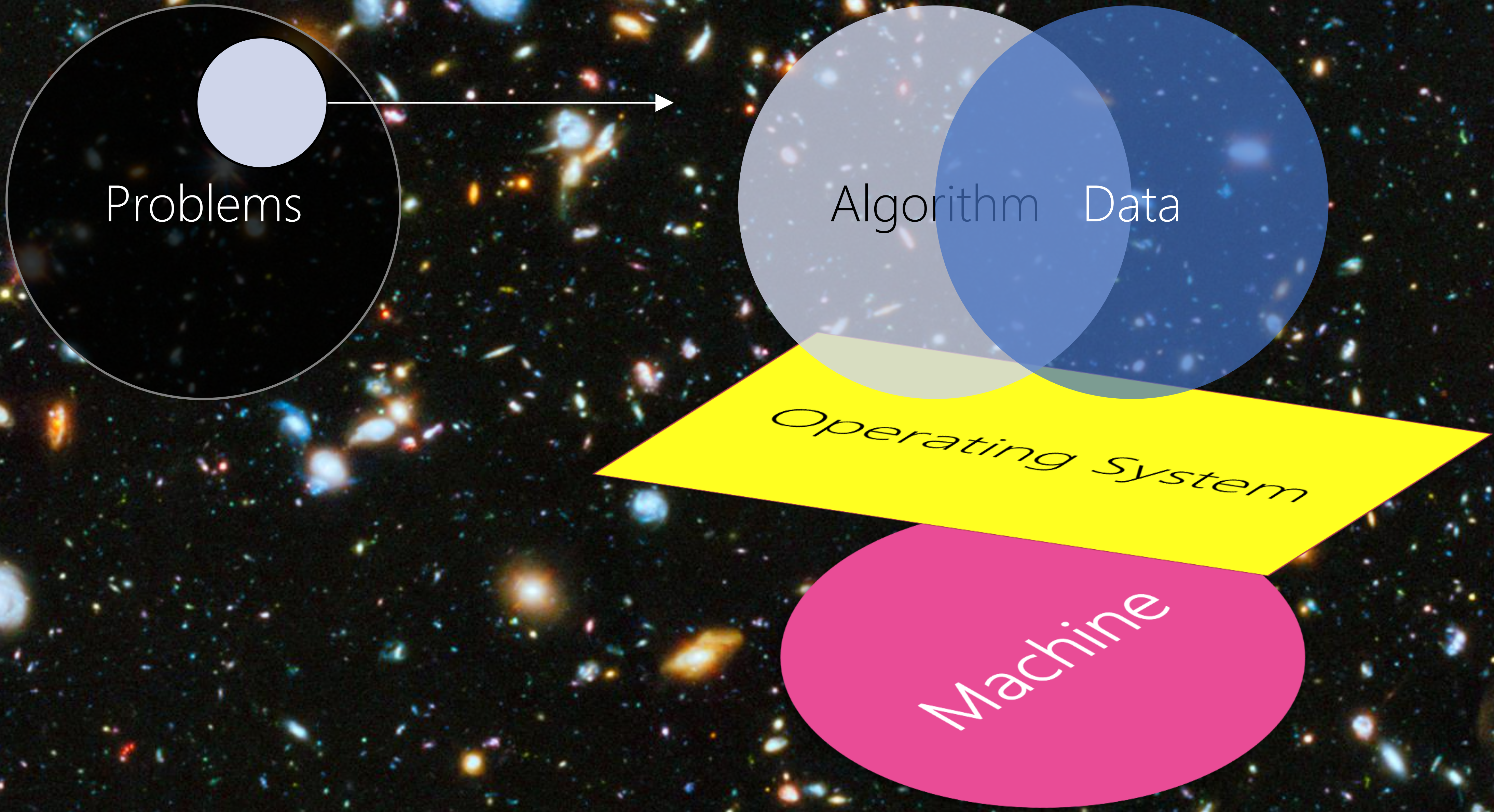
- Mathematics
- Physics
- Economics (game theory)
- Computing
- Statistics

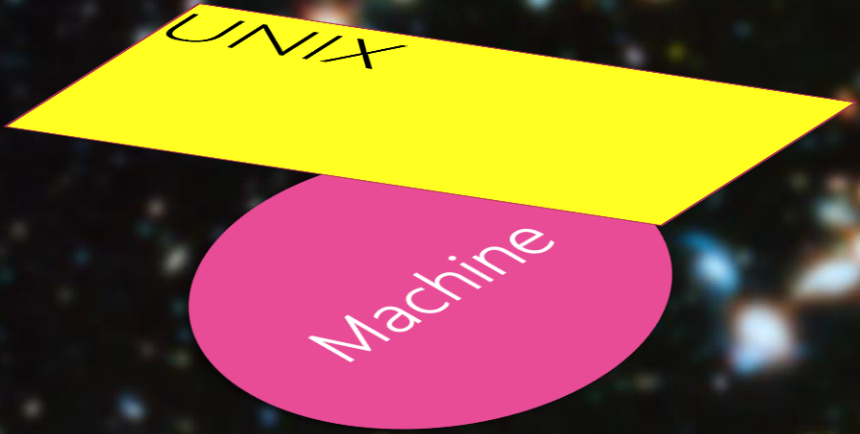

Problems
Data
Algorithm
Computer
Memory to Store
Data (Input, Output)
Instructions (Code)
Bus
Processor

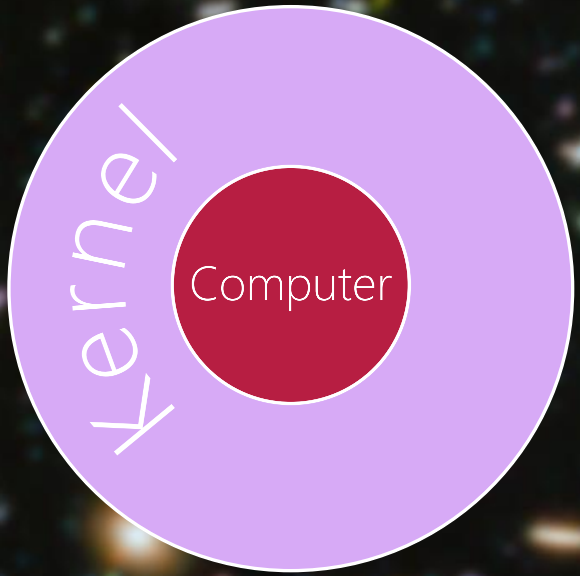
Dennis MacAlistair Ritchie
Kenneth Lane Thompson
AT&T Bell Lab, 1972, GE 645

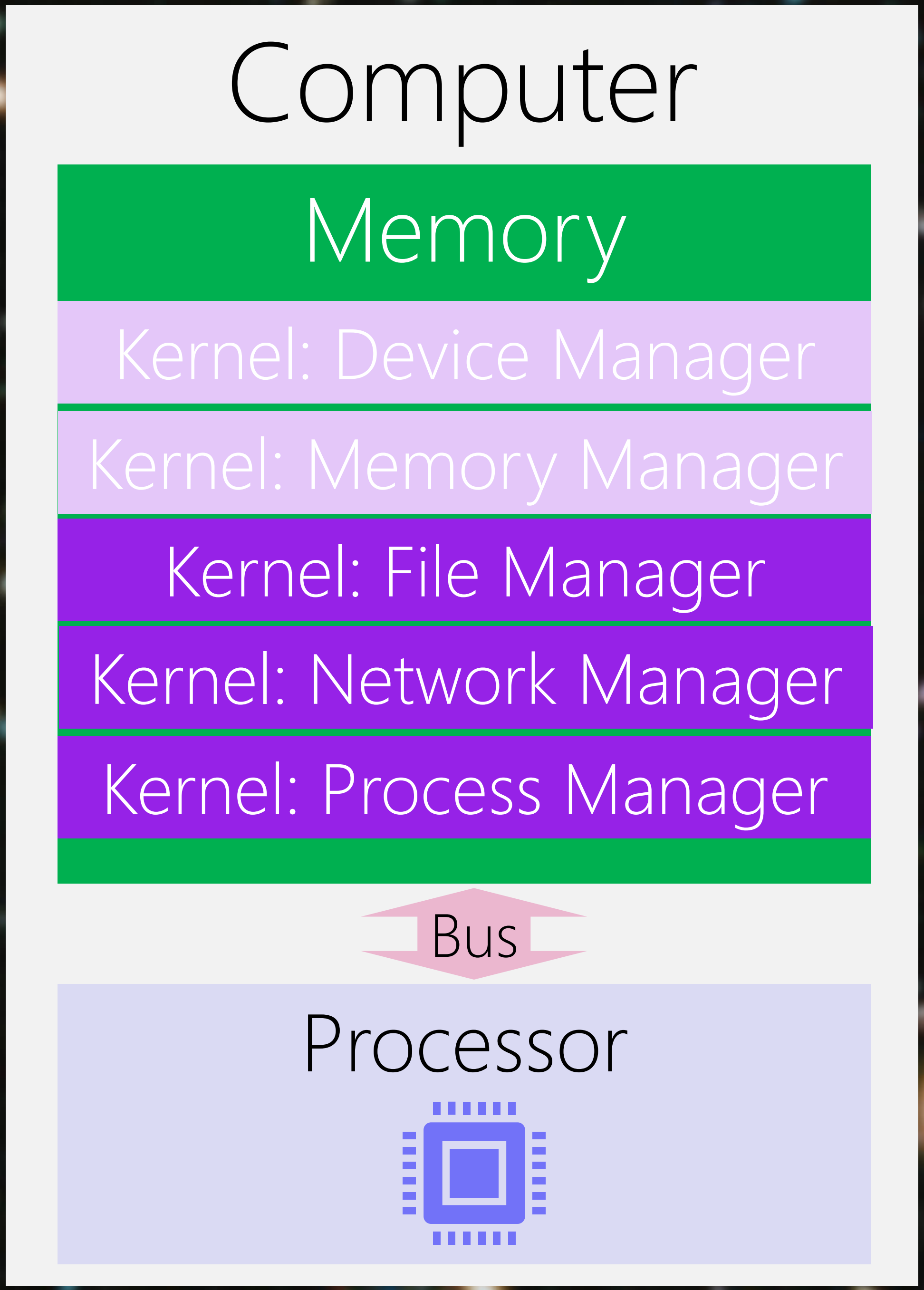
Operating System

A program for programs!
System-level Program


Problems
Algorithm
Data
Operating System
Machine


UNIX
Machine


Kernel
Computer


Computer
Memory
Kernel: Device Manager
Kernel: Memory Manager
Kernel: File Manager
Kernel: Network Manager
Kernel: Process Manager
Bus
Processor

You Can't Handle The Truth, A Few Good Men (1992) - Aaron Sorkin

Trinity escapes from Agents
- The Matrix (1999), Lana & Lilly Wachowski

Storage Device == String of Bytes == File

`fcntl.h: create(), open(): O_CREAT, O_EXCL, O_APPEND, ...`

`unistd.h: read(), write(), lseek(), close()`

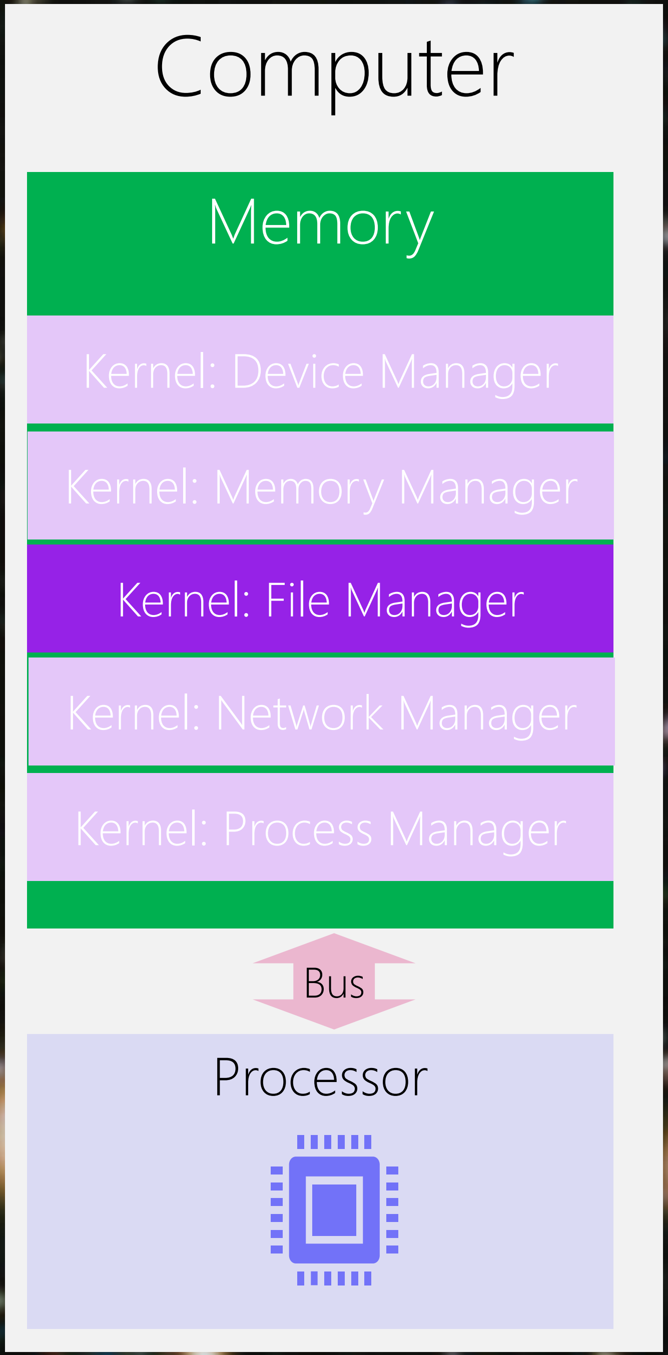
File Descriptor: Standard FDs, `dup()` vs. `open()`

I/O Redirection

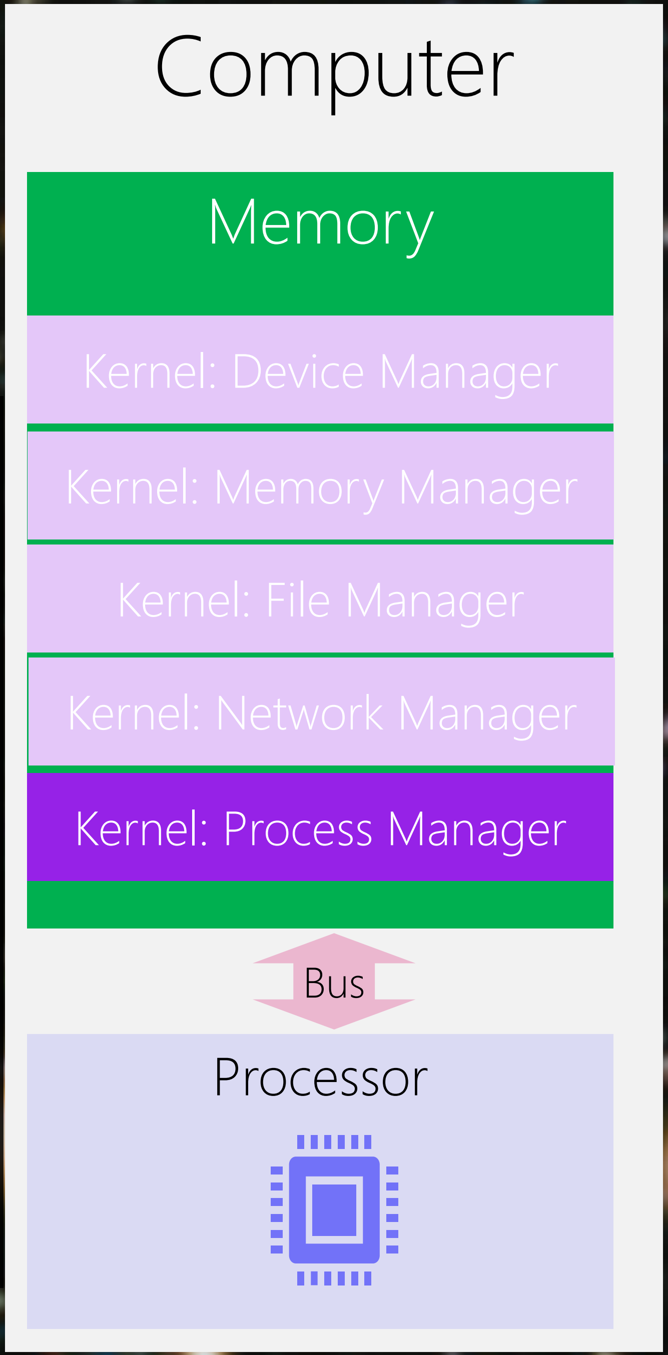
Storage File System:

i-Node, Data Blocks, Groups, Partitions, Files, Directories, Deleting/Moving/Copying

Fragmentation



"Normal people should see Naples before they die, but the great chess masters have to win the Wijk aan Zee tournament first of all"-Bent Larsen
Sharing 1 Chessboard



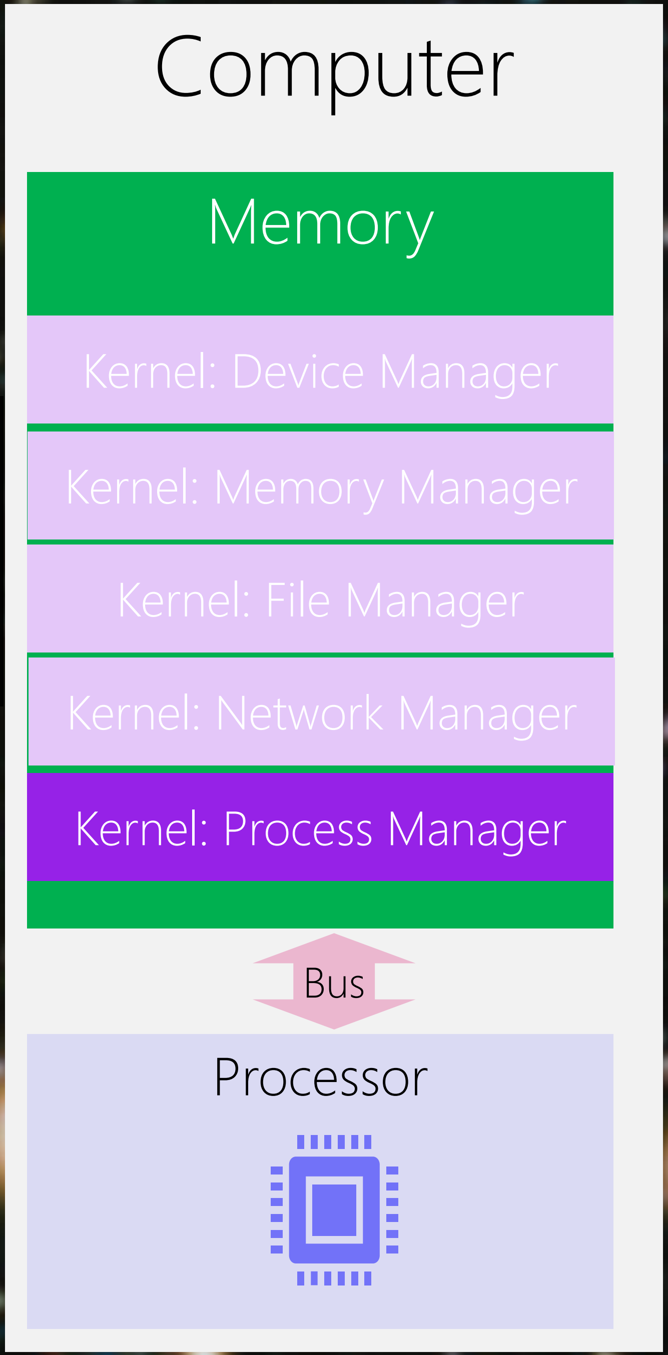
Bootstrapping a program: `main()`, memory layout and the segments: cs, ds, bss, ss,
Stack and return addresses
Heap and dynamic allocation
Process Identifier: `getpid()`
Process Termination, Exit Status, Normal vs. Abnormal Exit

Multiprocessing:
Why? Processor Sharing/Scheduling, `fork()`, parent vs. child, Orphan, Zombie, `wait()`
Child generation models
Grandchild, Multiple children

`fork()` vs. `exec()`

Process Life Cycle: Program, Ready, Run, Blocked, Terminated

Inter-Process Communication (IPC)
One way parent to child
Two way but harsh: signaling, `kill()`, signal handling, default action, ignoring, ...
Two way but normal: Sharing a single file, unnamed file: `pipe()`, named file: `FIFO`,
Synchronized: Producer-Consumer

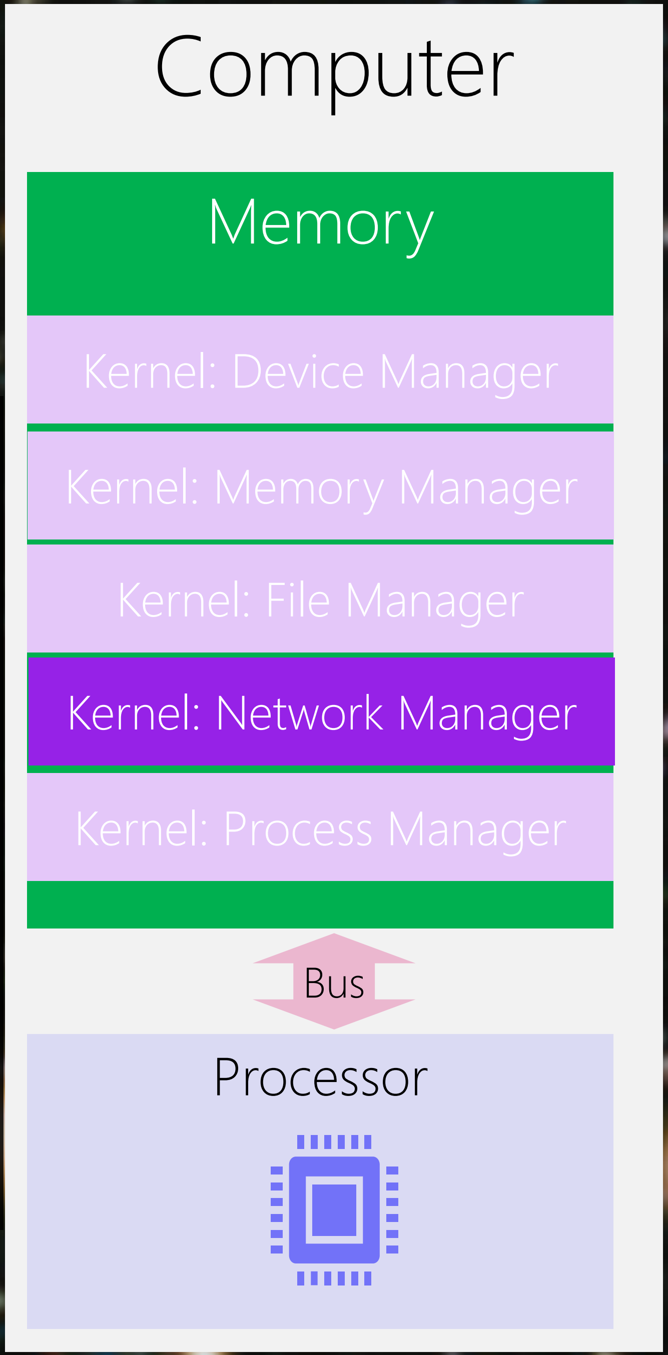


Network Inter-Process Communication (IPC)
Network Protocol: TCP/IP, UDP, TCP, IP, PORT

UDP: shoot a mail
Sender-Receiver: `socket(), bind(), sendto(), recvfrom()`

TCP: a phone call
The Server: `socket(), bind(), listen(), accept(), recv(), send()`
Clients: `socket(), bind(), connect(), send(), recv()`

signal handling, default action, ignoring, ...
Two way but normal: Sharing a single file, unnamed file: `pipe(),` named file: `FIFO,`
Synchronized: Producer-Consumer

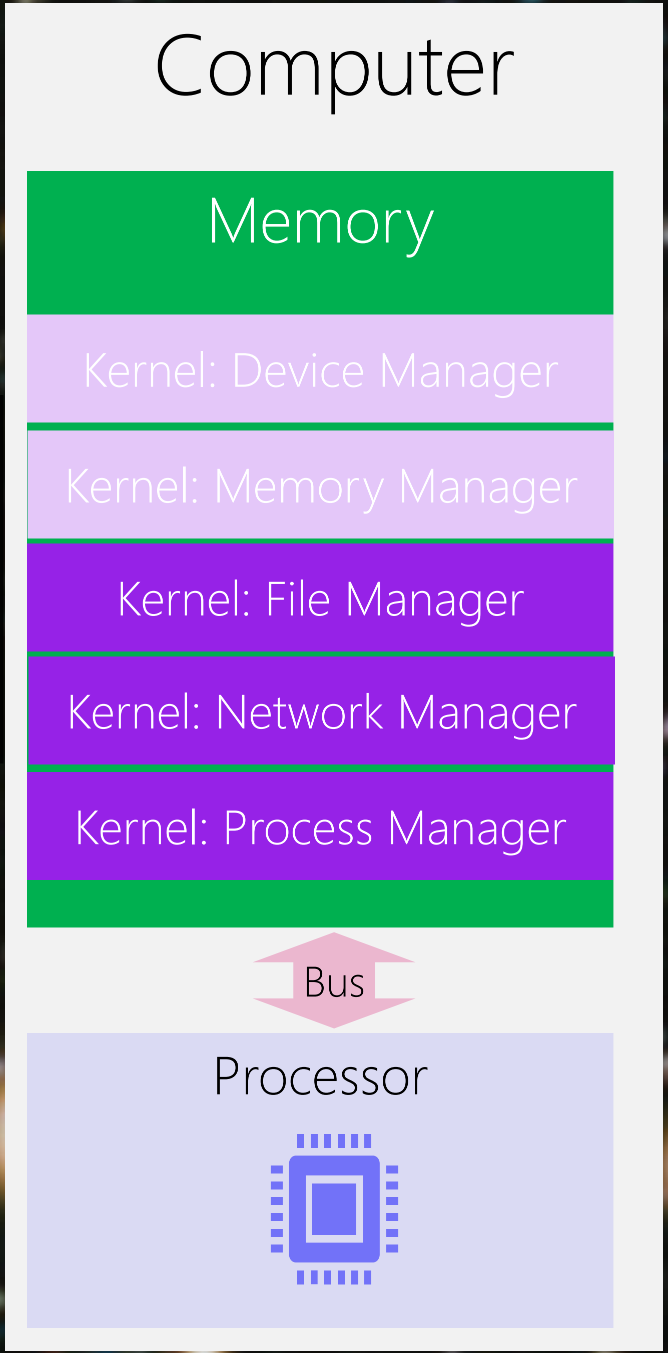


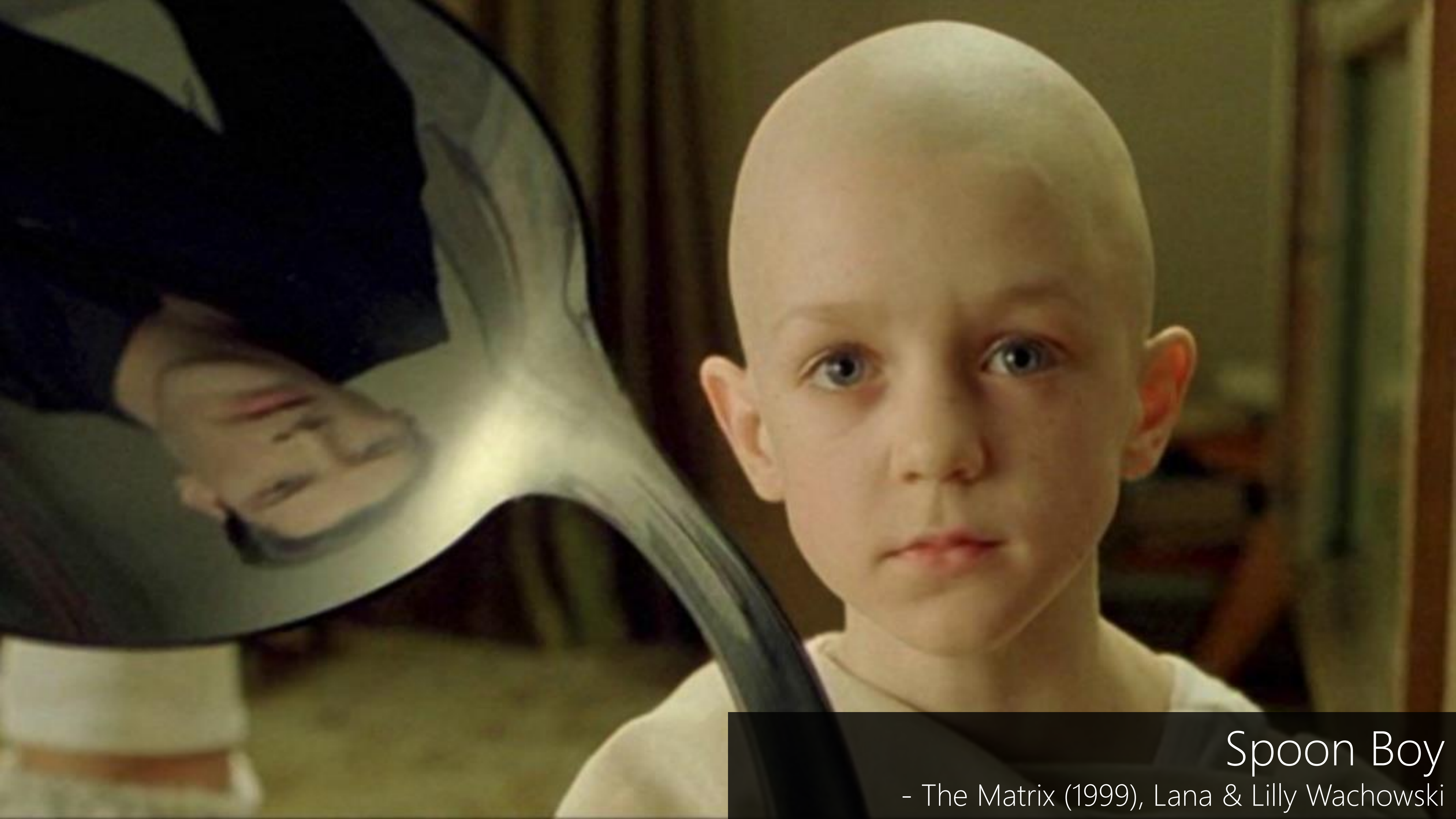
Rosemary's Baby (1968), Roman Polanski
Based on the novel Rosemary's Baby, Ira Levin

Design Patterns

Daemon: Active/Passive, Background, Memory-resident
Daemonize: Orphan child with logfile

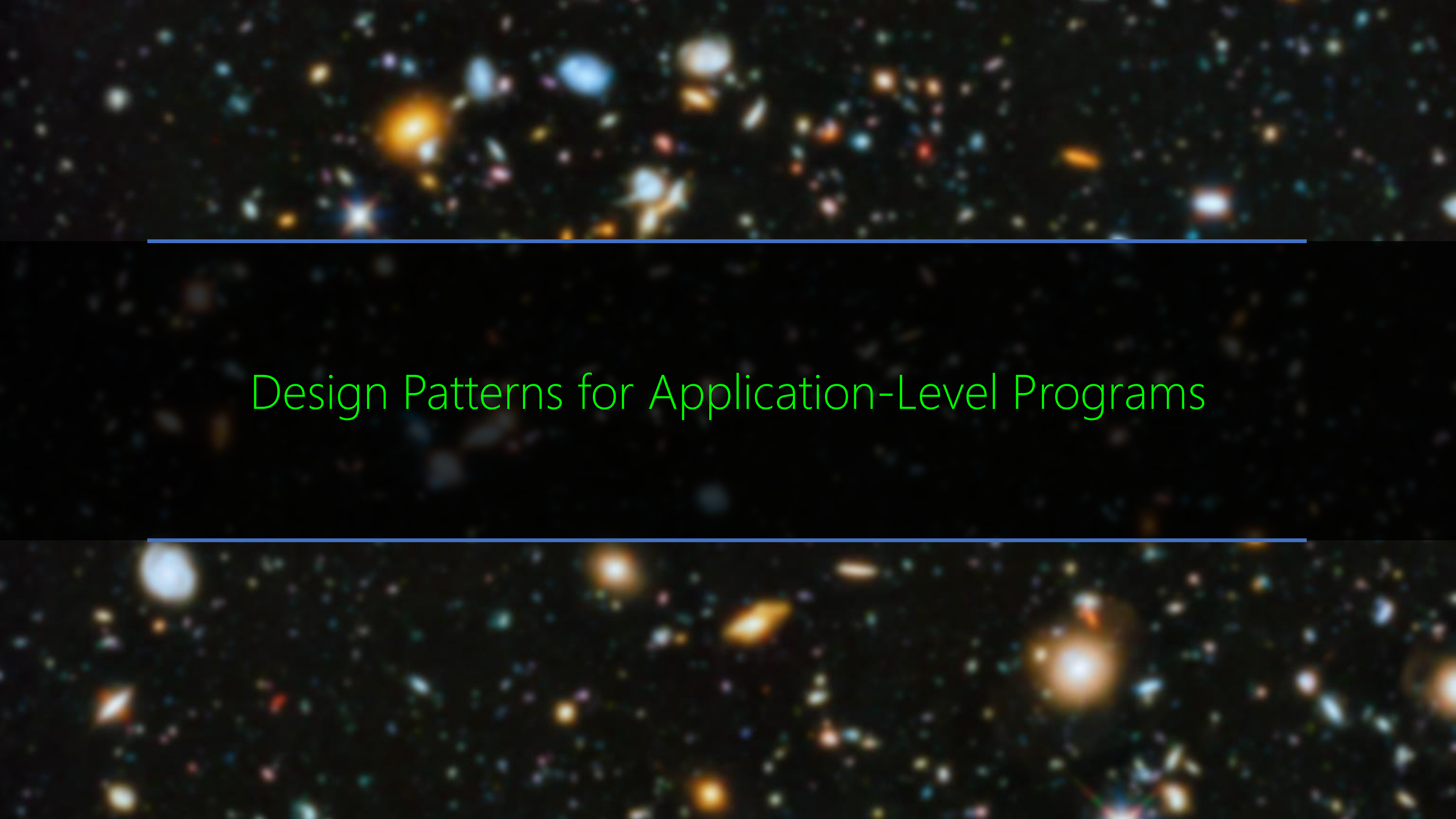
Shared Library: Daemon Library, Web Service





Spoon Boy

- The Matrix (1999), Lana & Lilly Wachowski



Design Patterns for Application-Level Programs

Shared Libraries

Is it possible to share standard library routines for all processes, inside or outside a computer?

https://cs-fundamentals.com/c-programming/static-and-dynamic-linking-in-c

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

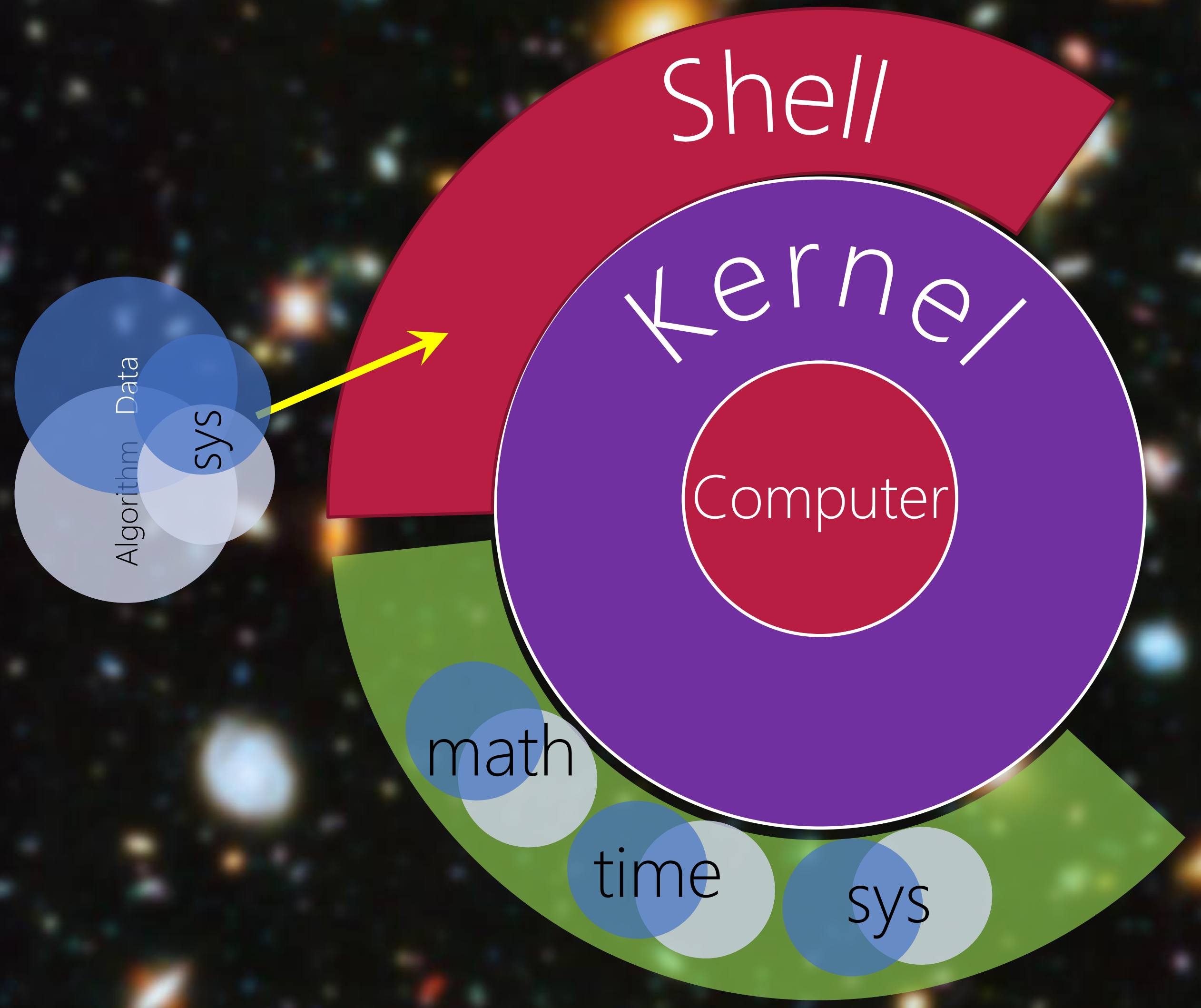
```
0001 0000 0000 0010 0000 0000 0000 1100
00f1 0000 0003 0000 0000 0000 0000 0000
0000 0000 0000 0000 3309 0110 0111 0000
0046 0000 0220 0000 0000 0000 0000 2200
0111 0000 0000 0000 e432 0000 0000 0000
```

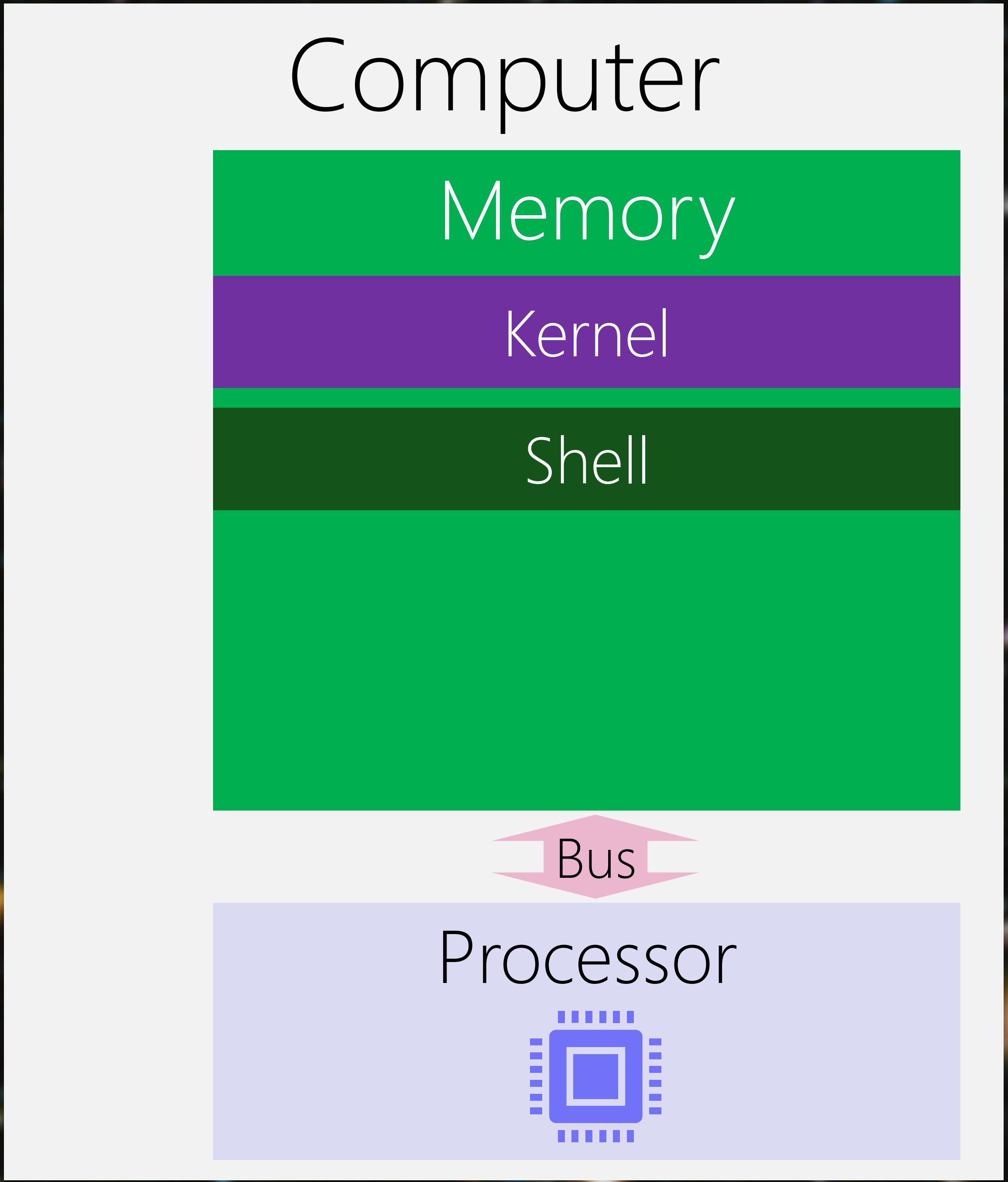
OP Code

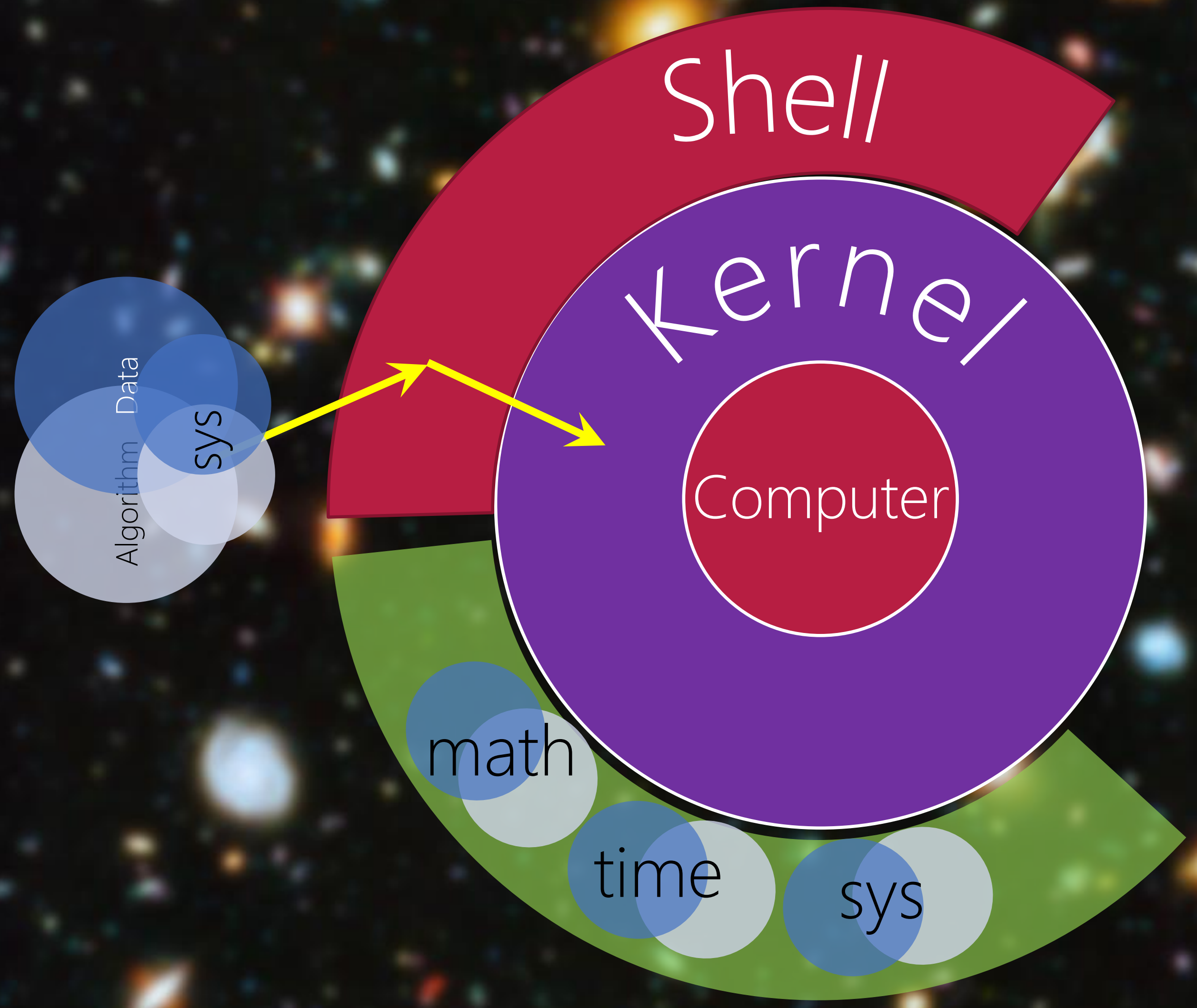
```
0001 0000 0000 0000 0000 0000 0000 0000
0011 0000 0003 0000 0000 0000 0000 0000
0000 0000 0000 0000 3309 0000 0000 0000
0096 0000 0000 0000 0000 0000 0000 0000
0001 0000 0000 0000 0000 0000 0000 0000
```

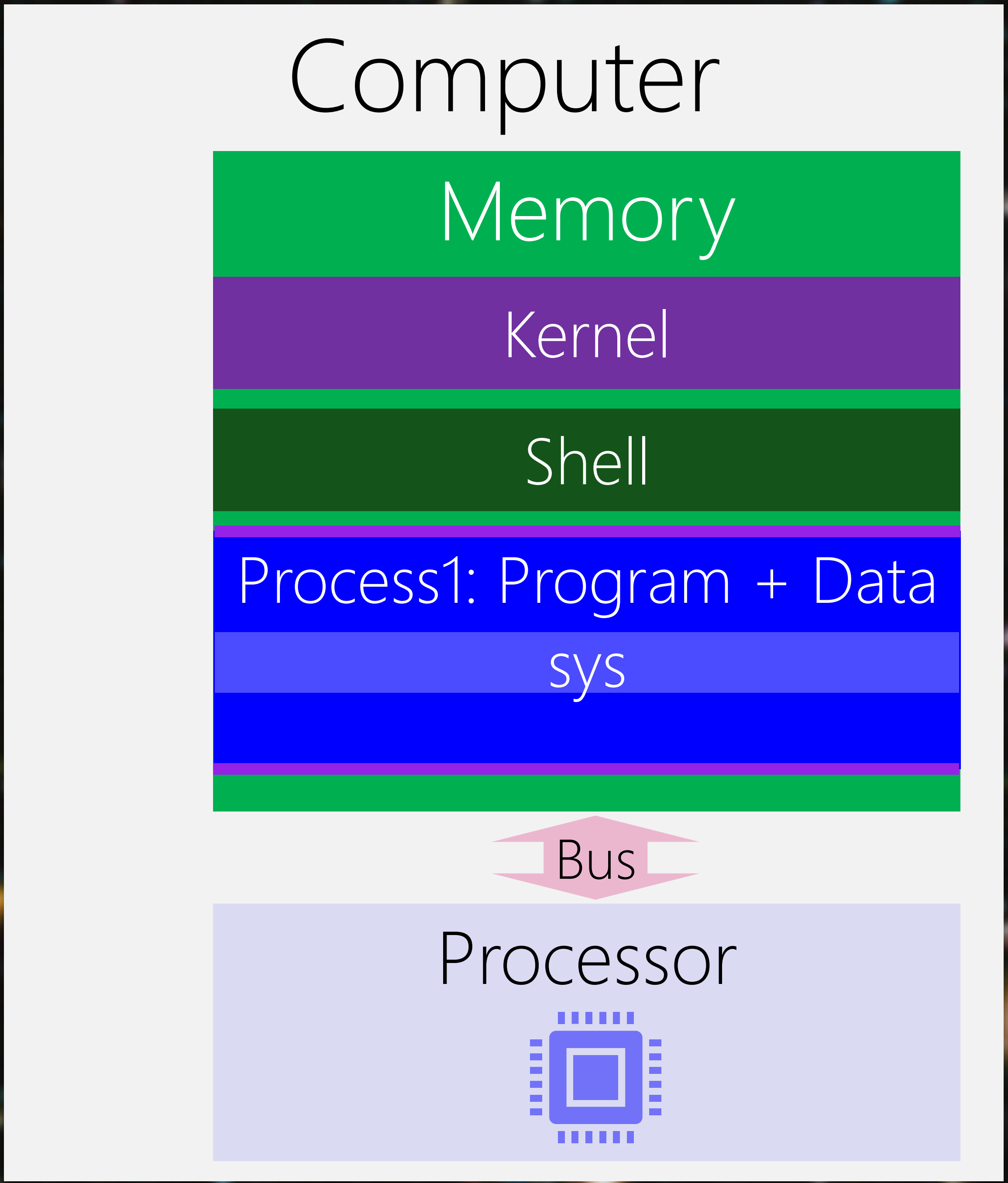
Linker

```
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
```


Shell
Kernel
Computer
Data
sys
Algorithm
math
time
sys


Computer
Memory
Kernel
Shell
Bus
Processor

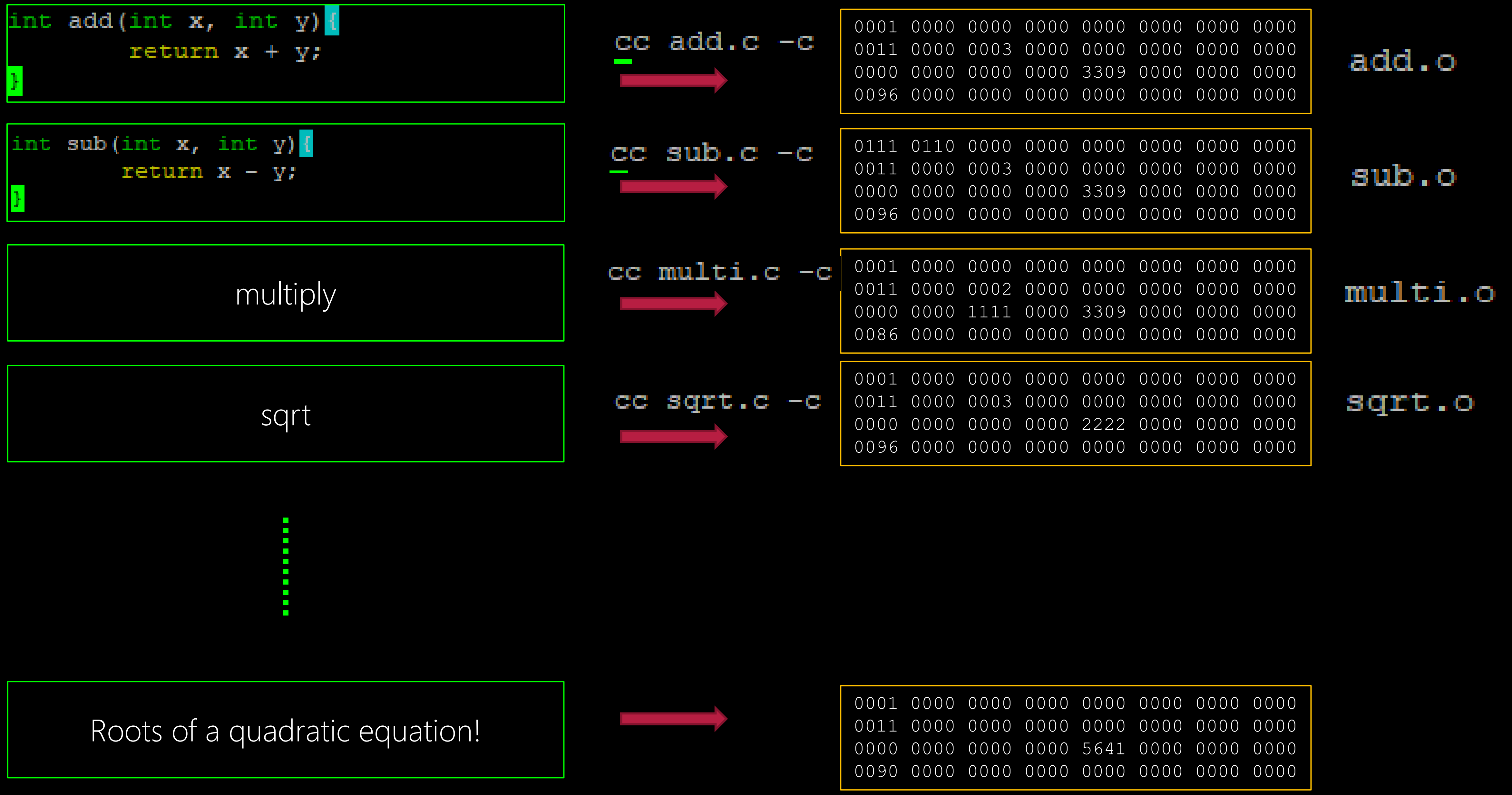

Shell
Kernel
Computer
Data
Algorithm
sys
math
time
sys


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

STATIC LINK

From Kernel's POV, everything is the same!

1) Building the Static Library



```
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

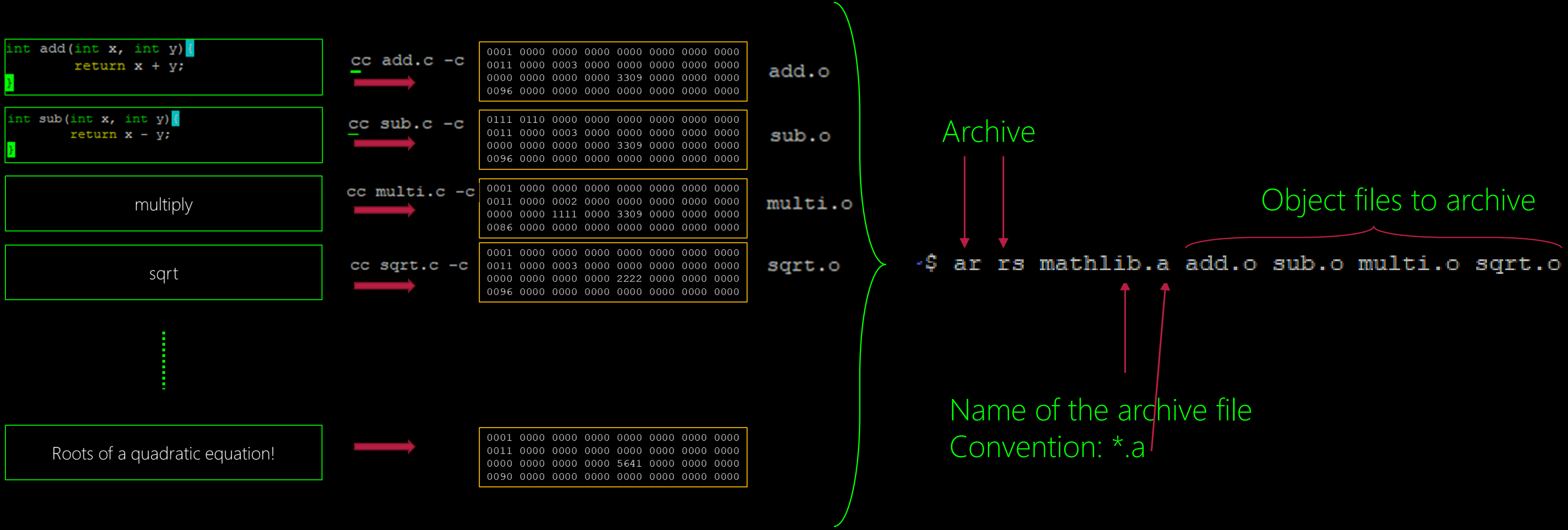
```
cc sqrt.c -c
```

```
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
```



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

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

```
cc add.c -c
cc sub.c -c
cc multi.c -c
cc sqrt.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

```
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

```
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

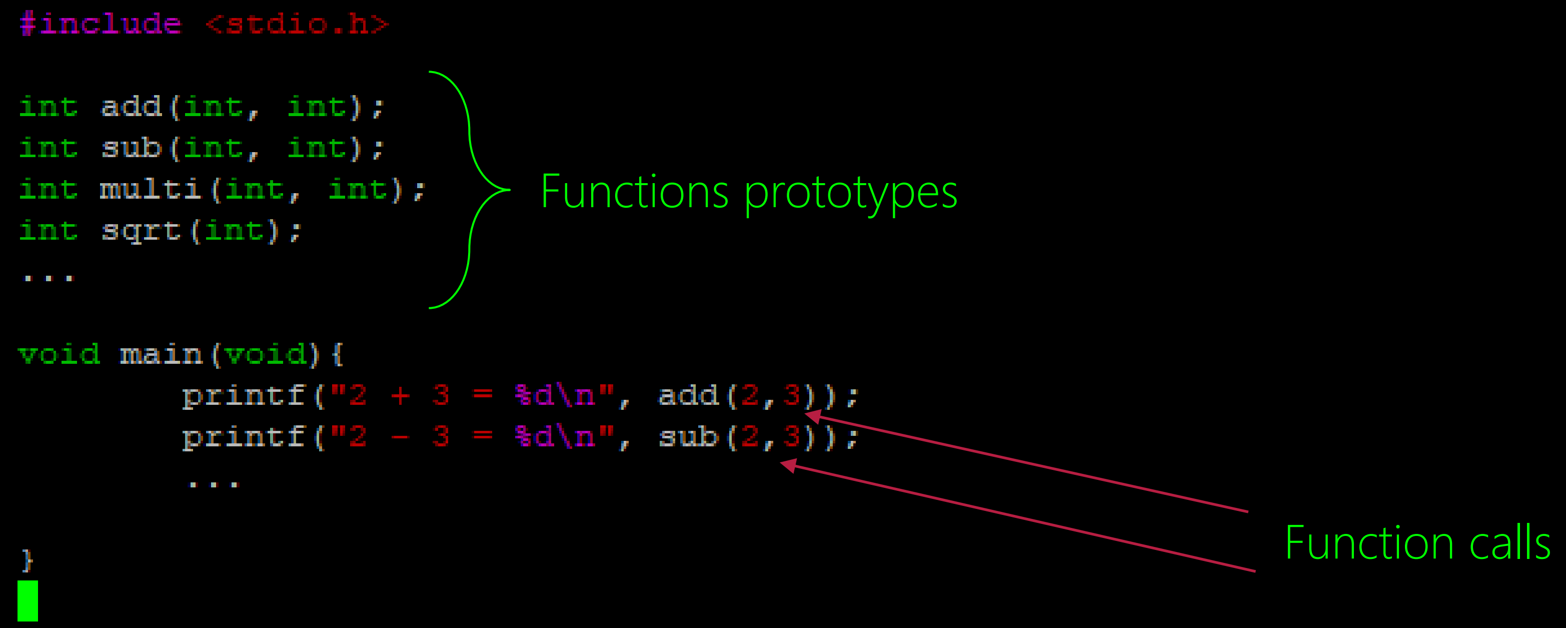
```
0001 0000 0000 0000 0000 0000 0000 0000
0011 0000 0003 0000 0000 0000 0000 0000
0000 0000 0000 0000 2222 0000 0000 0000
0096 0000 0000 0000 0000 0000 0000 0000
```

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
```

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

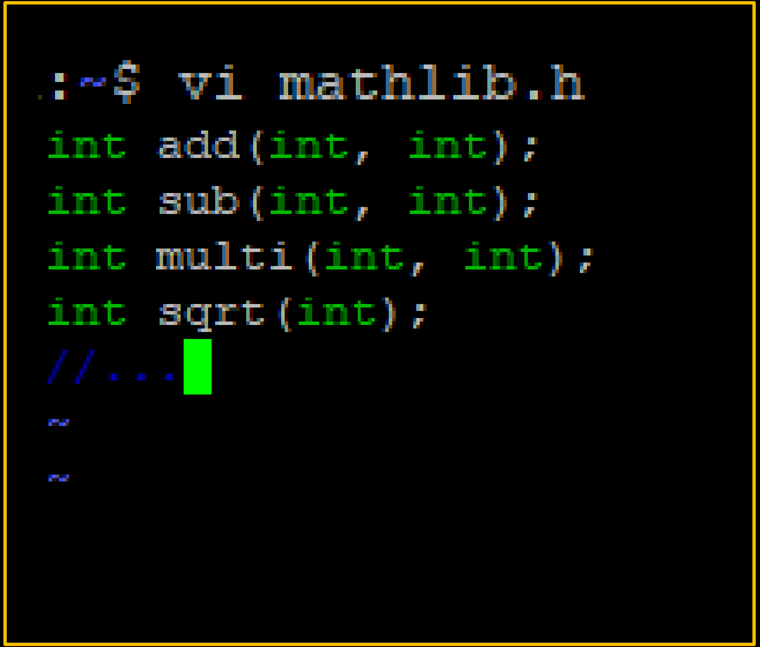
2) Using the Static Library



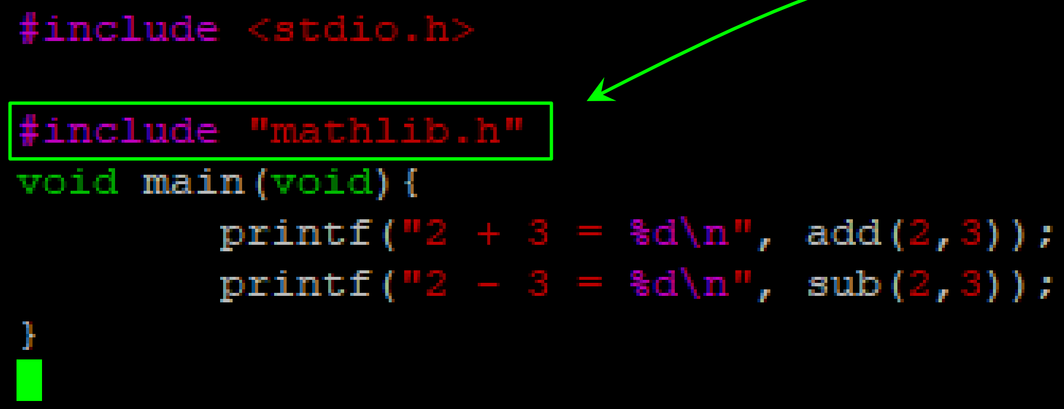
```c
#include <stdio.h>

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));
        ...

}
```

```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));
        ...

}
```

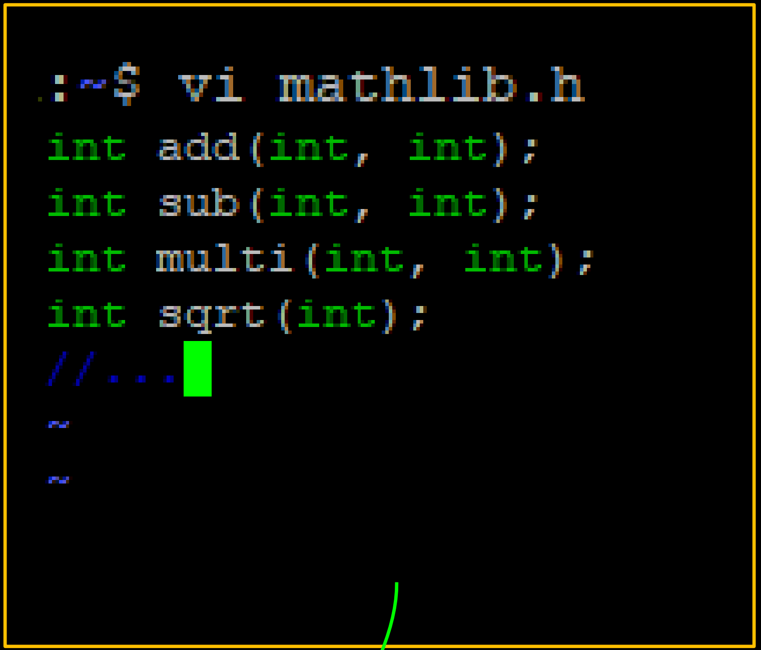
Functions prototypes into another file → Library Header File → mathlib.h



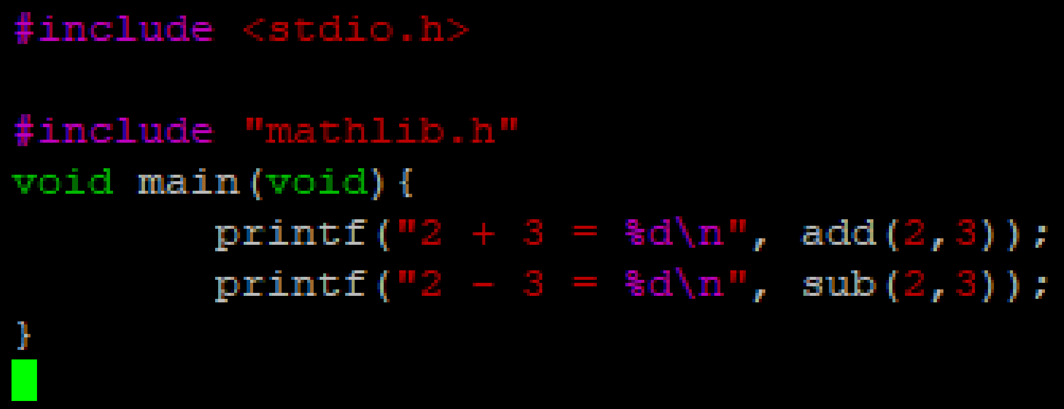
```
:~$ vi mathlib.h
int add(int, int);
int sub(int, int);
int multi(int, int);
int sqrt(int);
//...
~
~
```



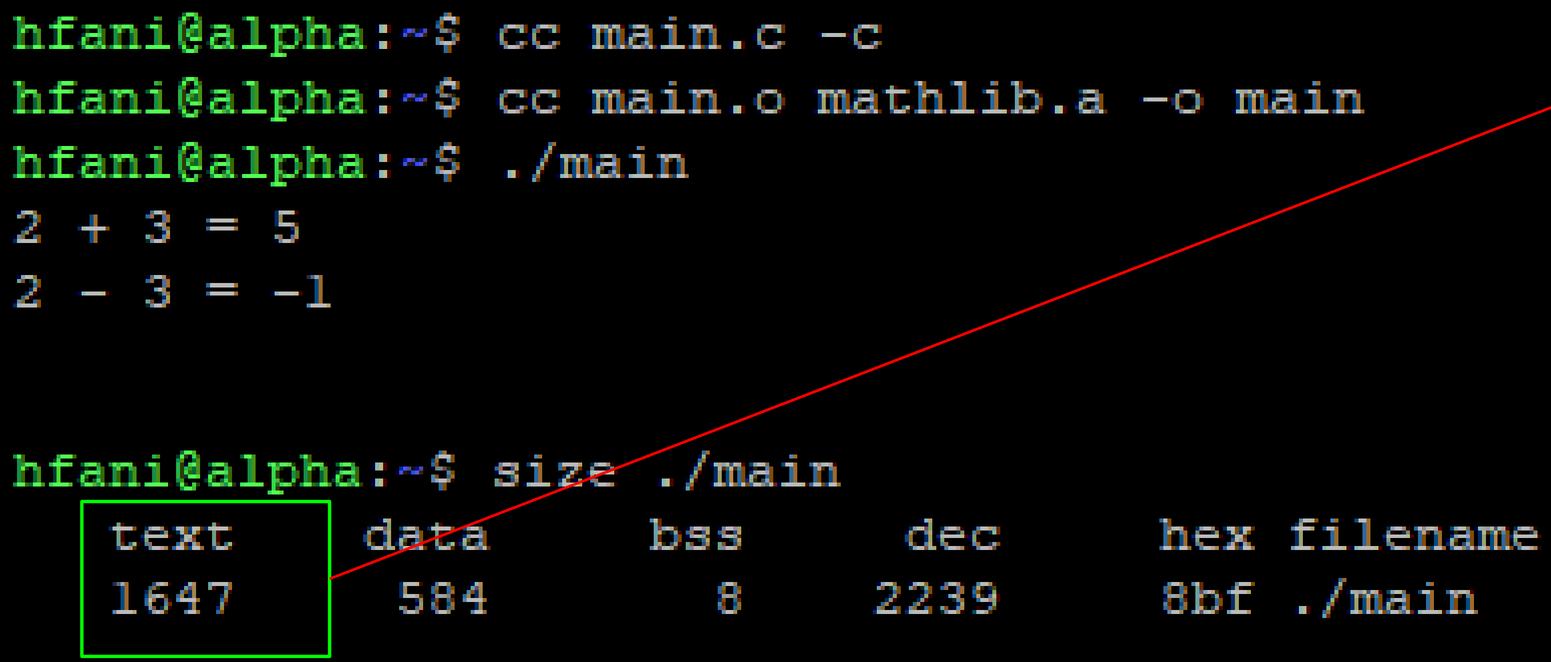
```c
#include <stdio.h>

#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);
//...
~
~
```



```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));
}
```



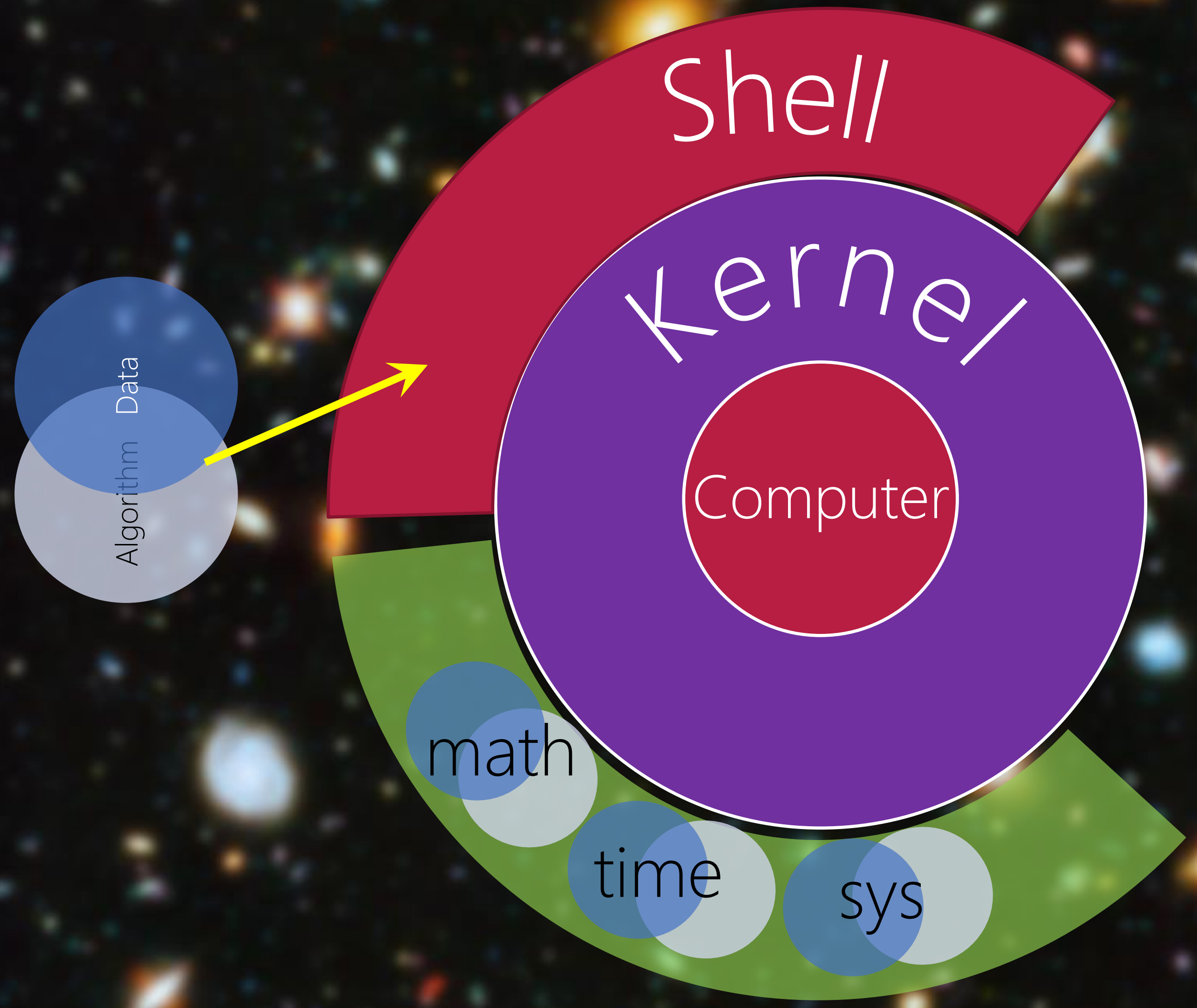
```
hfani@alpha:~$ cc main.c -c
hfani@alpha:~$ cc main.o mathlib.a -o main
hfani@alpha:~$ ./main
2 + 3 = 5
2 - 3 = -1


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

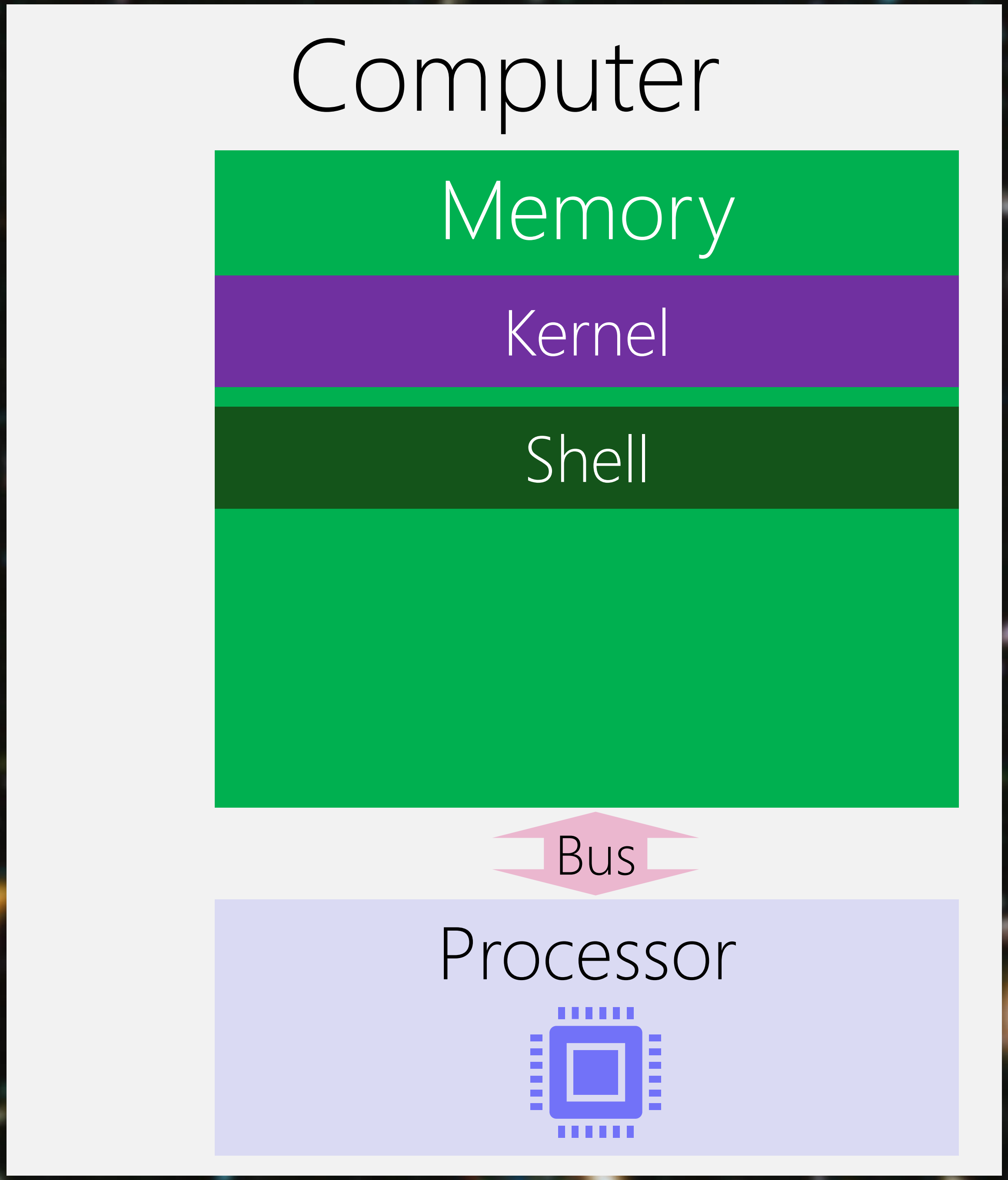
Includes main.o as well as
object files of mathlib.a <u>that are used</u>

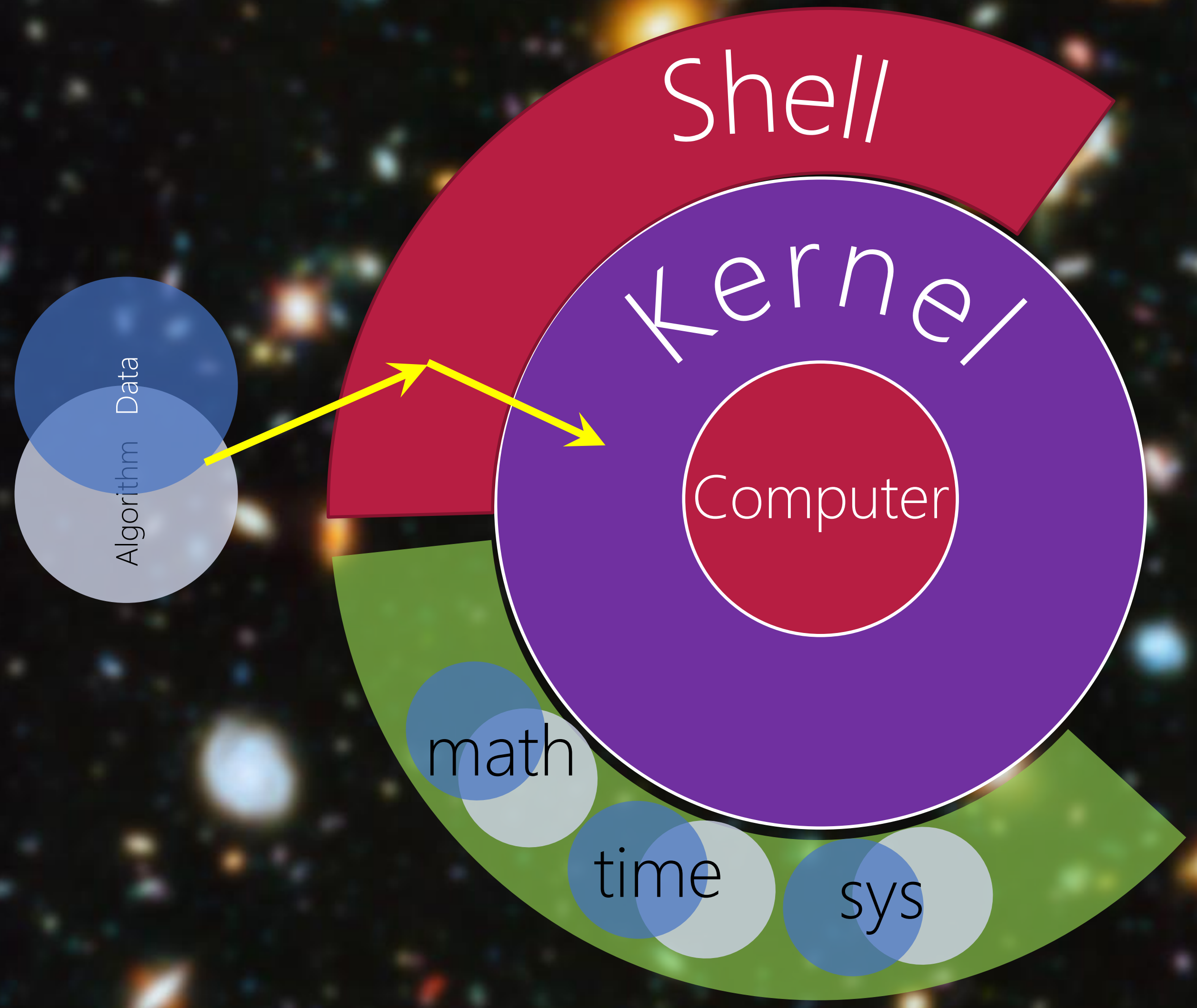
main.o
add.o
sub.o

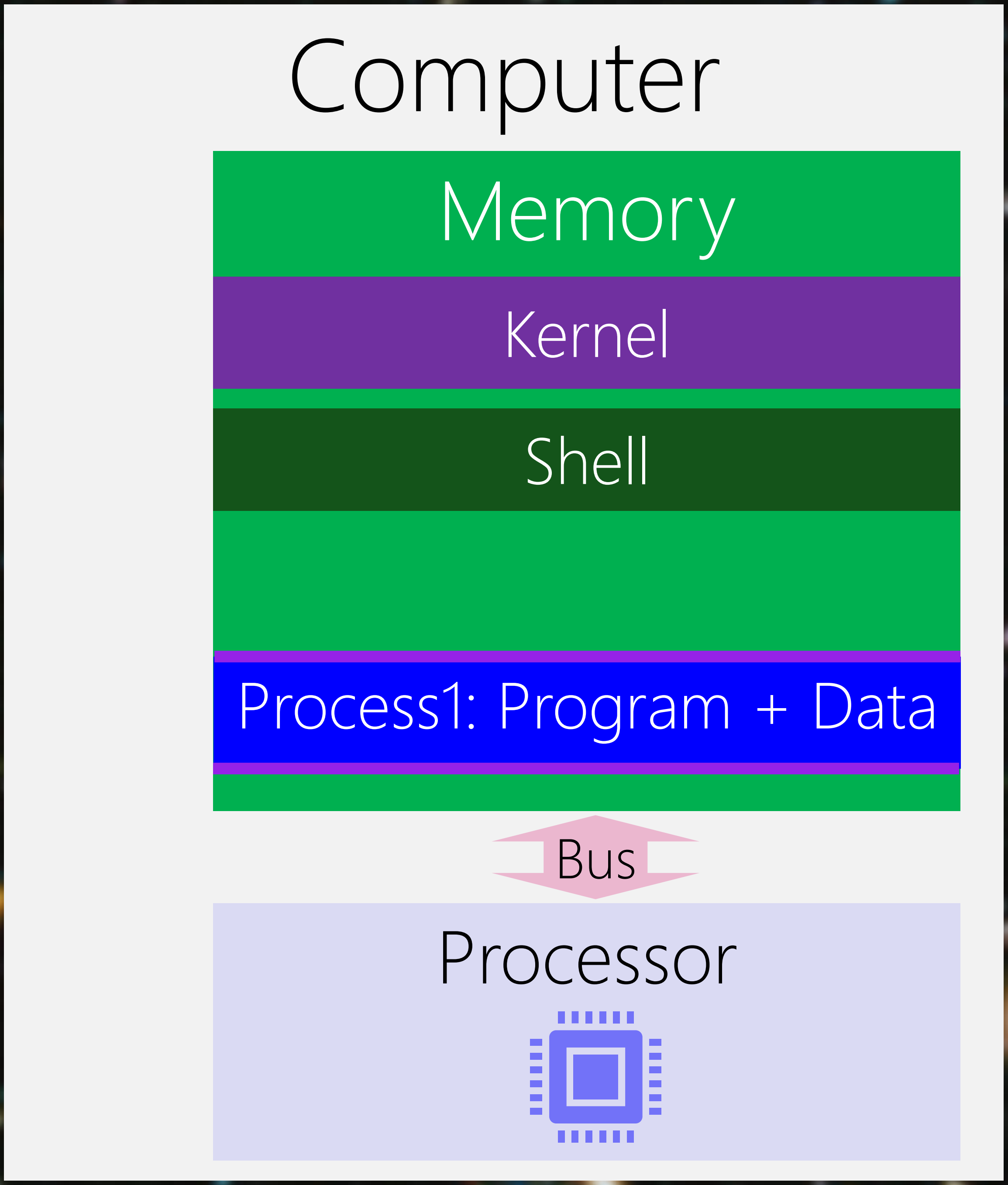
DYNAMIC LINK

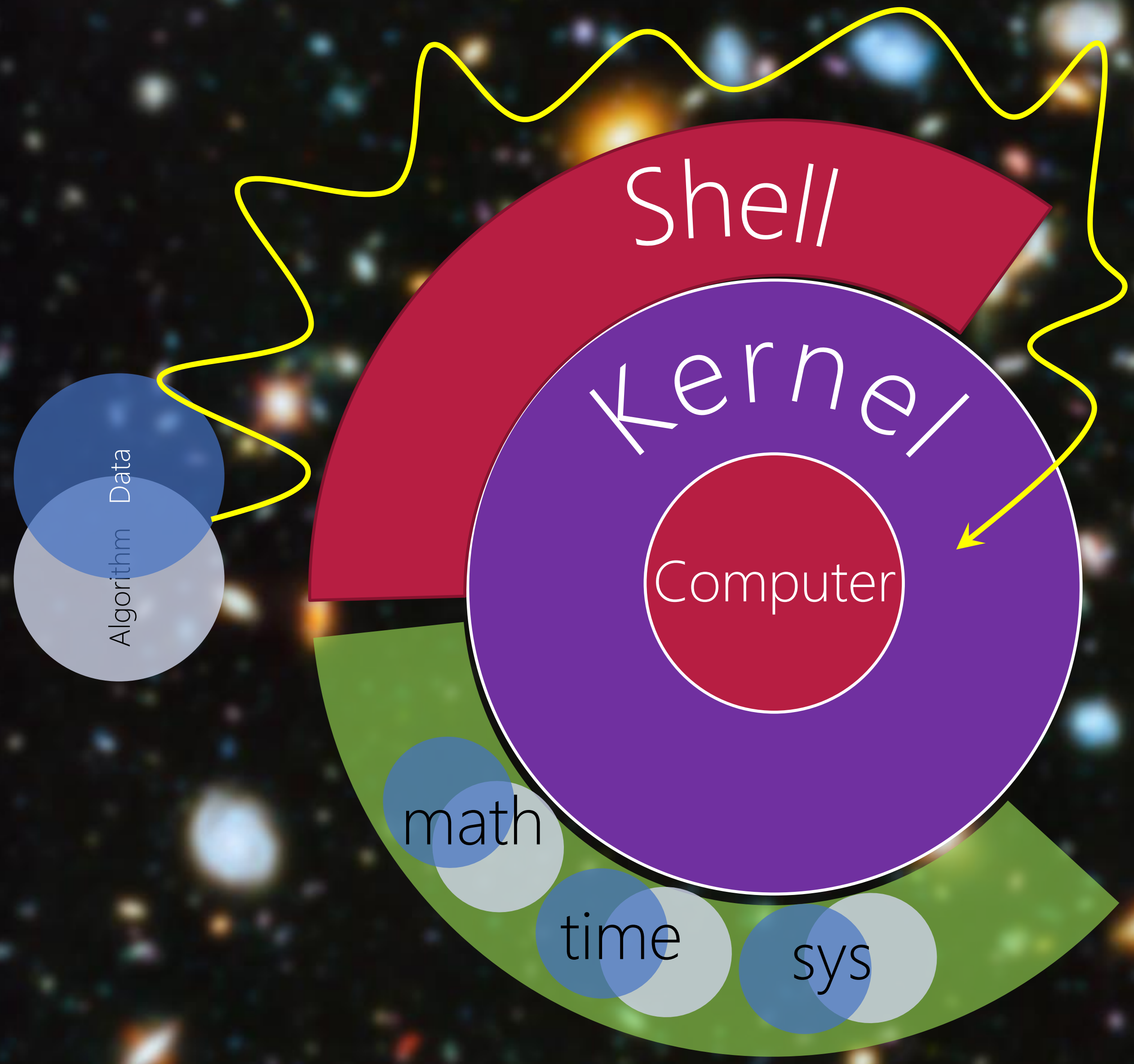
At run time, using kernel call!

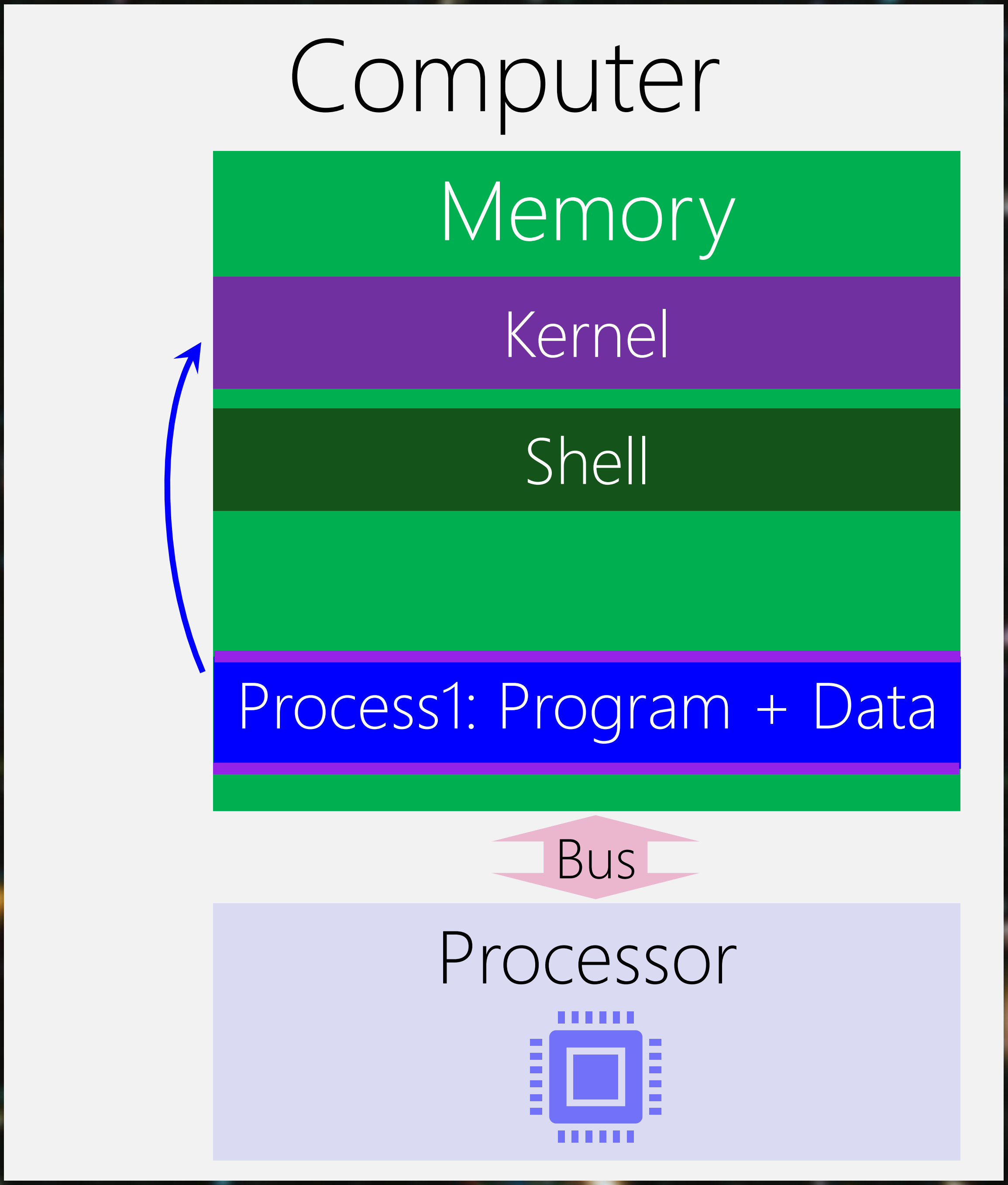

Shell
Kernel
Computer
Data
Algorithm
math
time
sys

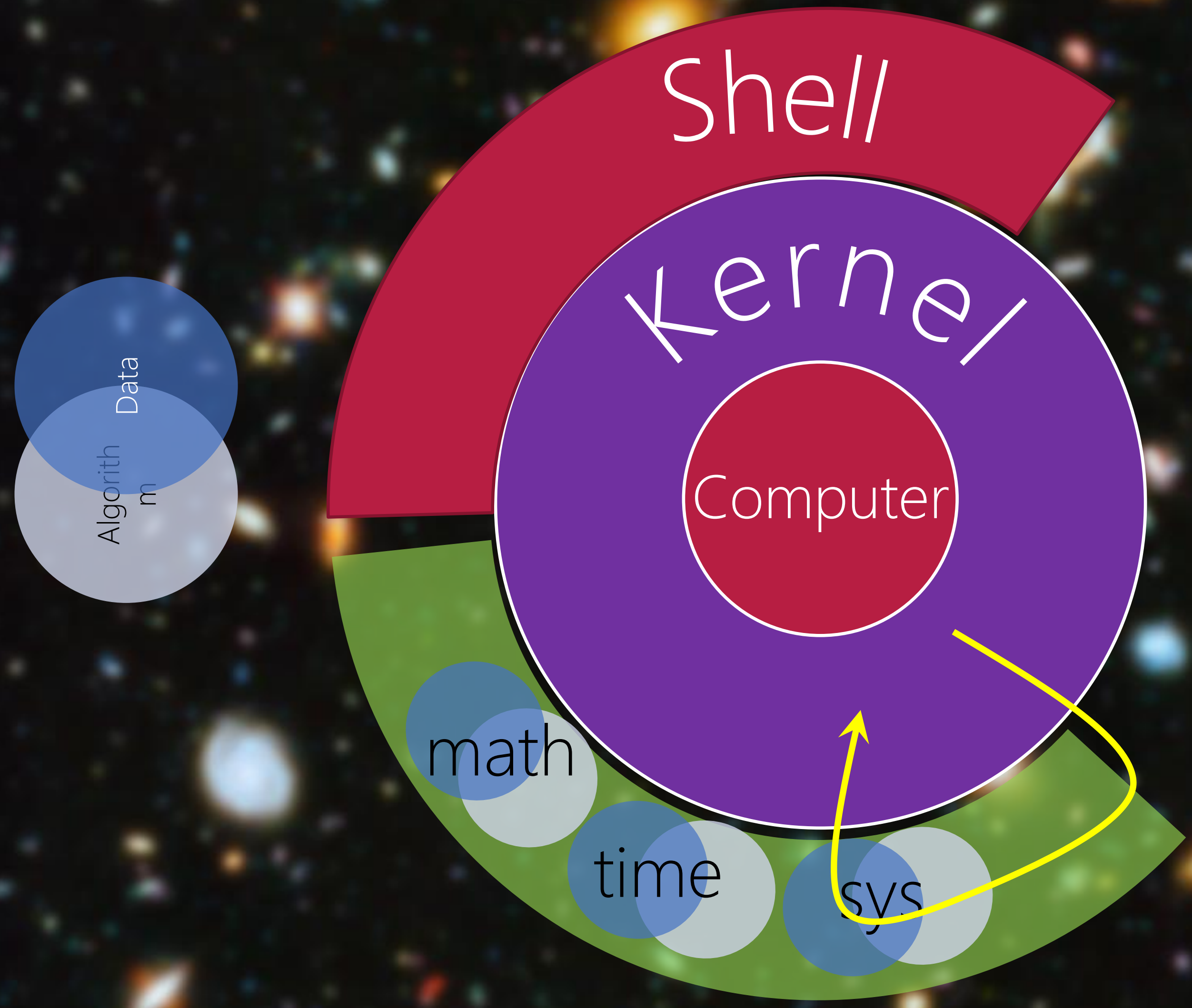

Computer
Memory
Kernel
Shell
Bus
Processor


Shell
Kernel
Computer
Data
Algorithm
math
time
sys

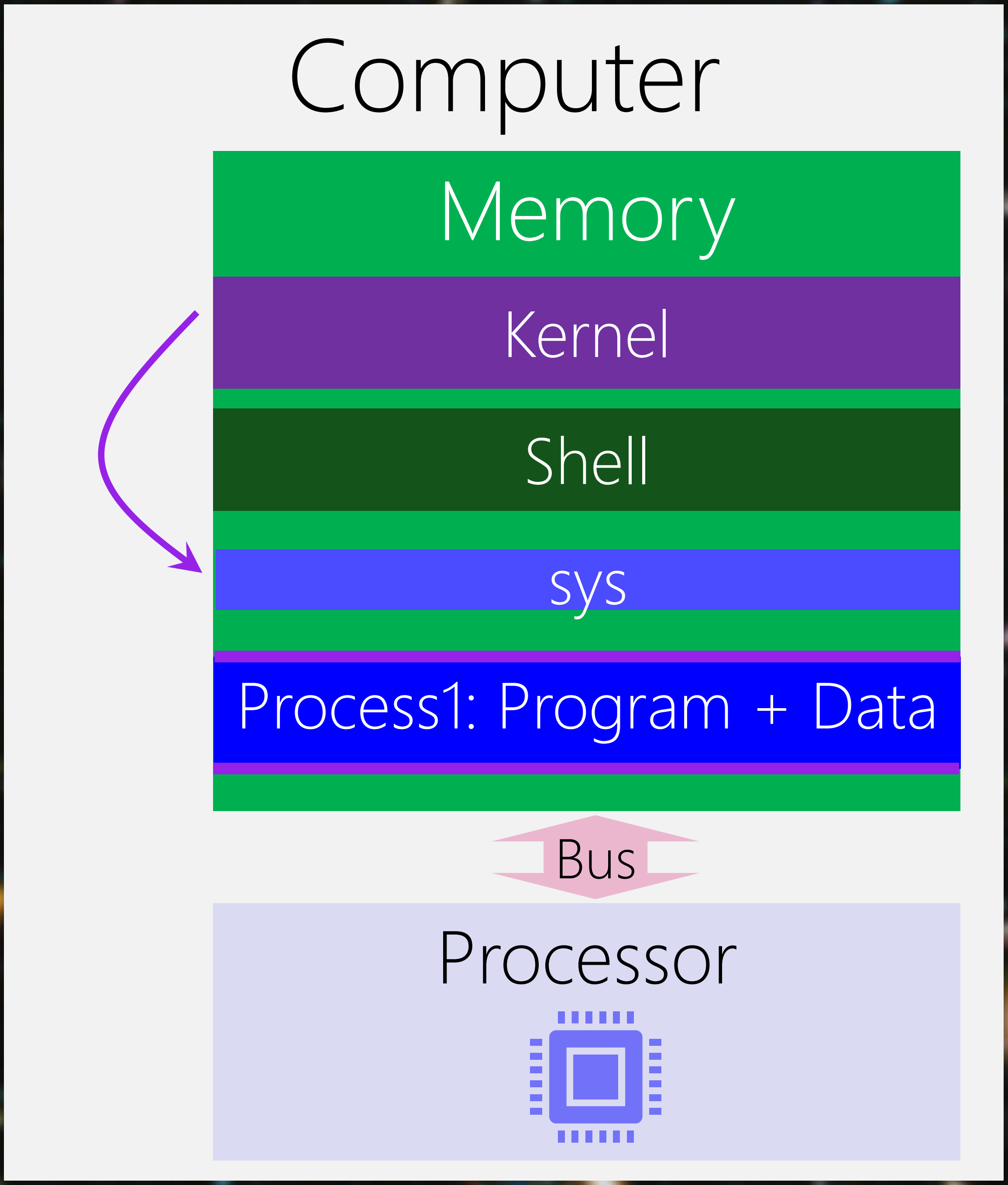

Computer
Memory
Kernel
Shell
Process1: Program + Data
Bus
Processor


Shell
Kernel
Computer
Data
Algorithm
math
time
sys

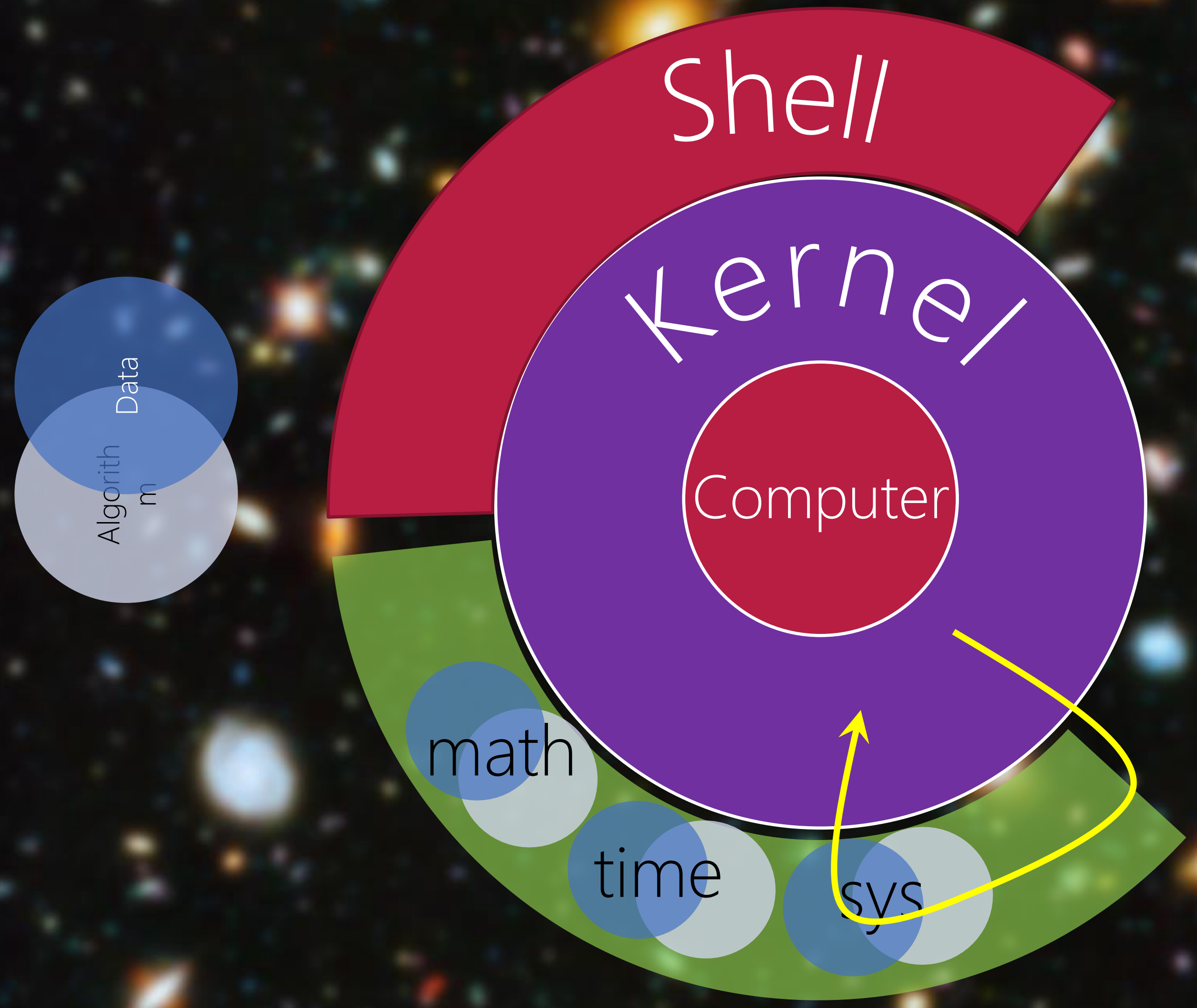

Computer
Memory
Kernel
Shell
Process1: Program + Data
Bus
Processor



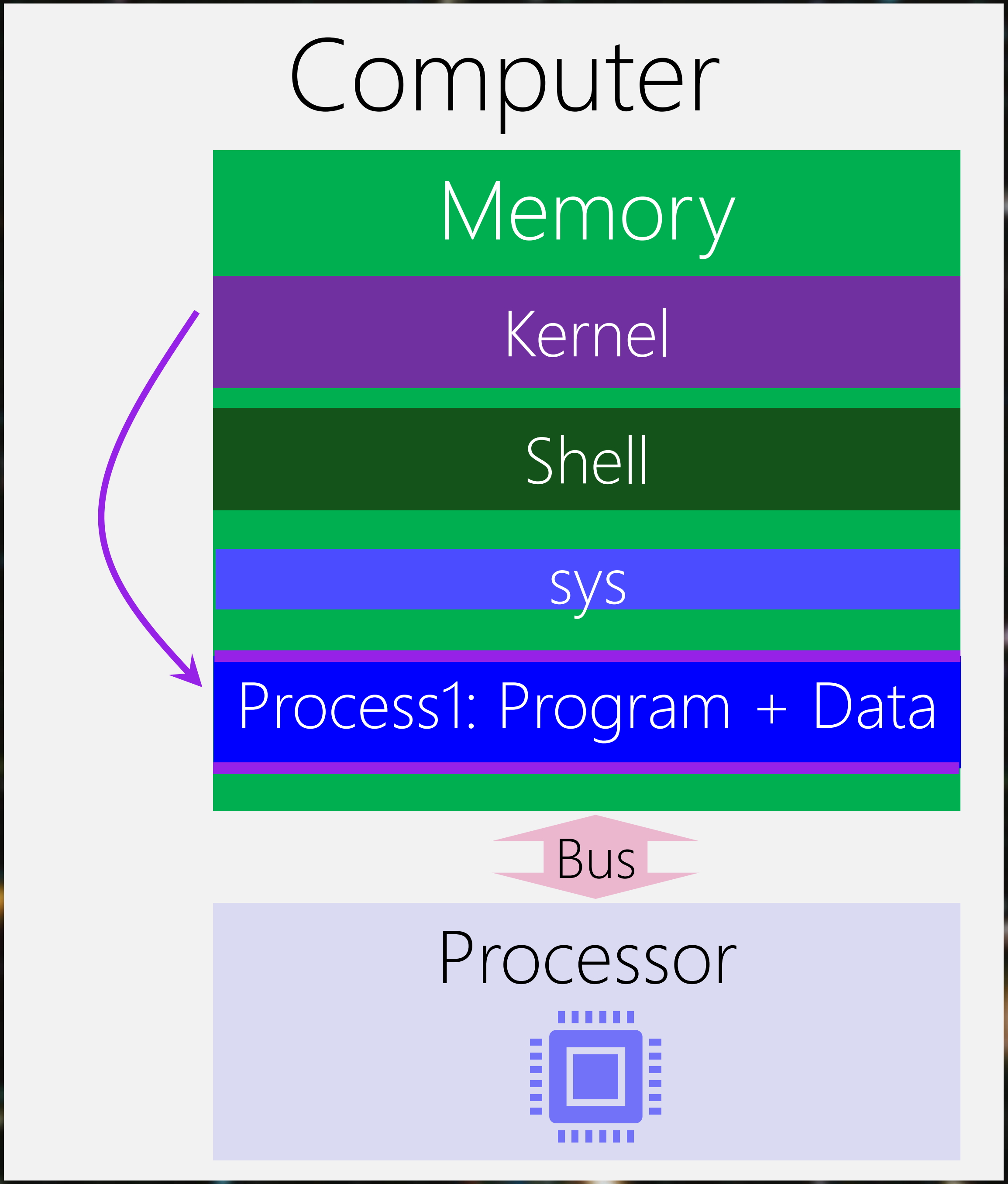
Data
Algorith m
Shell
Kernel
Computer
math
time
sys



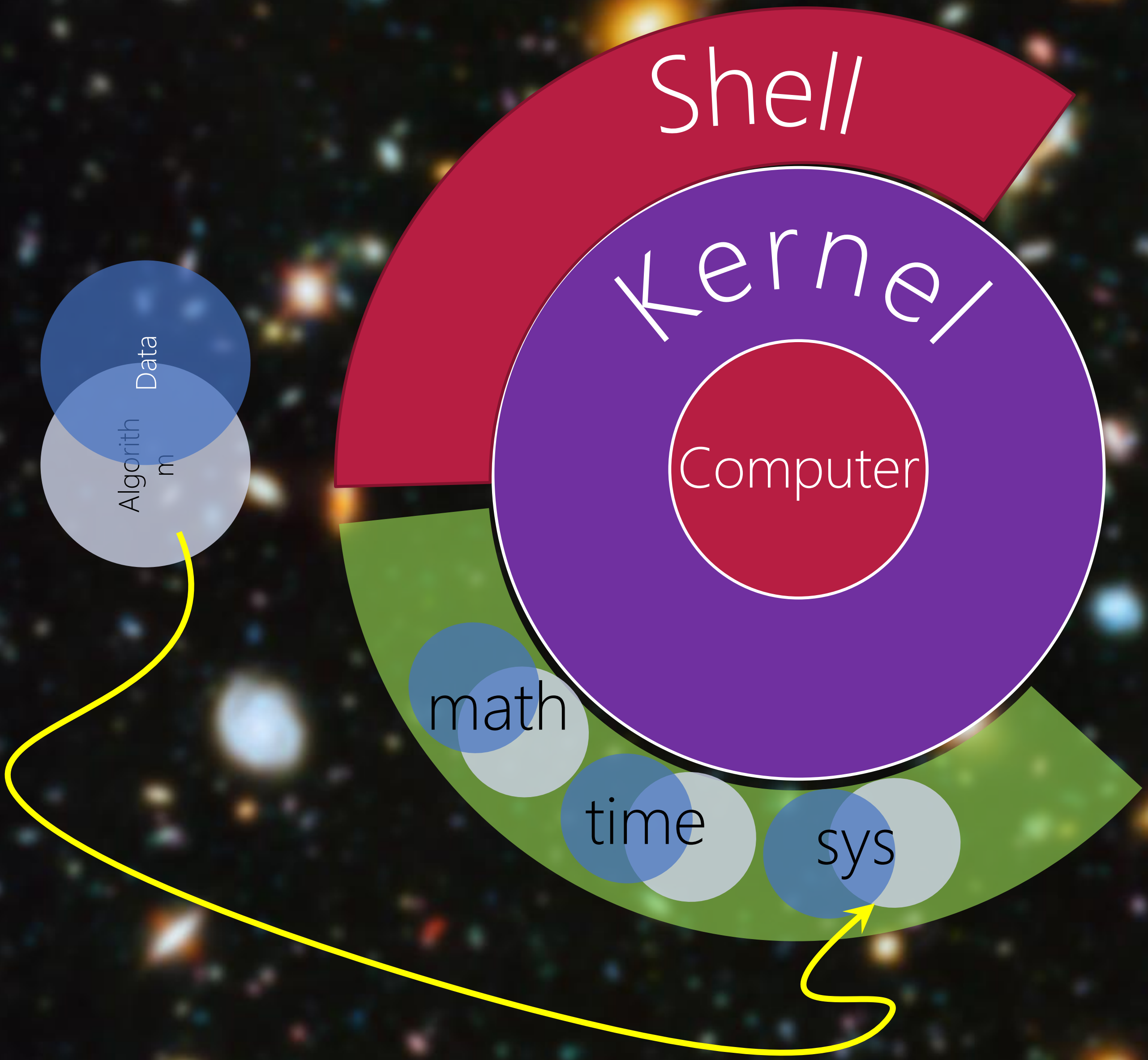
Computer
Memory
Kernel
Shell
sys
Process1: Program + Data
Bus
Processor

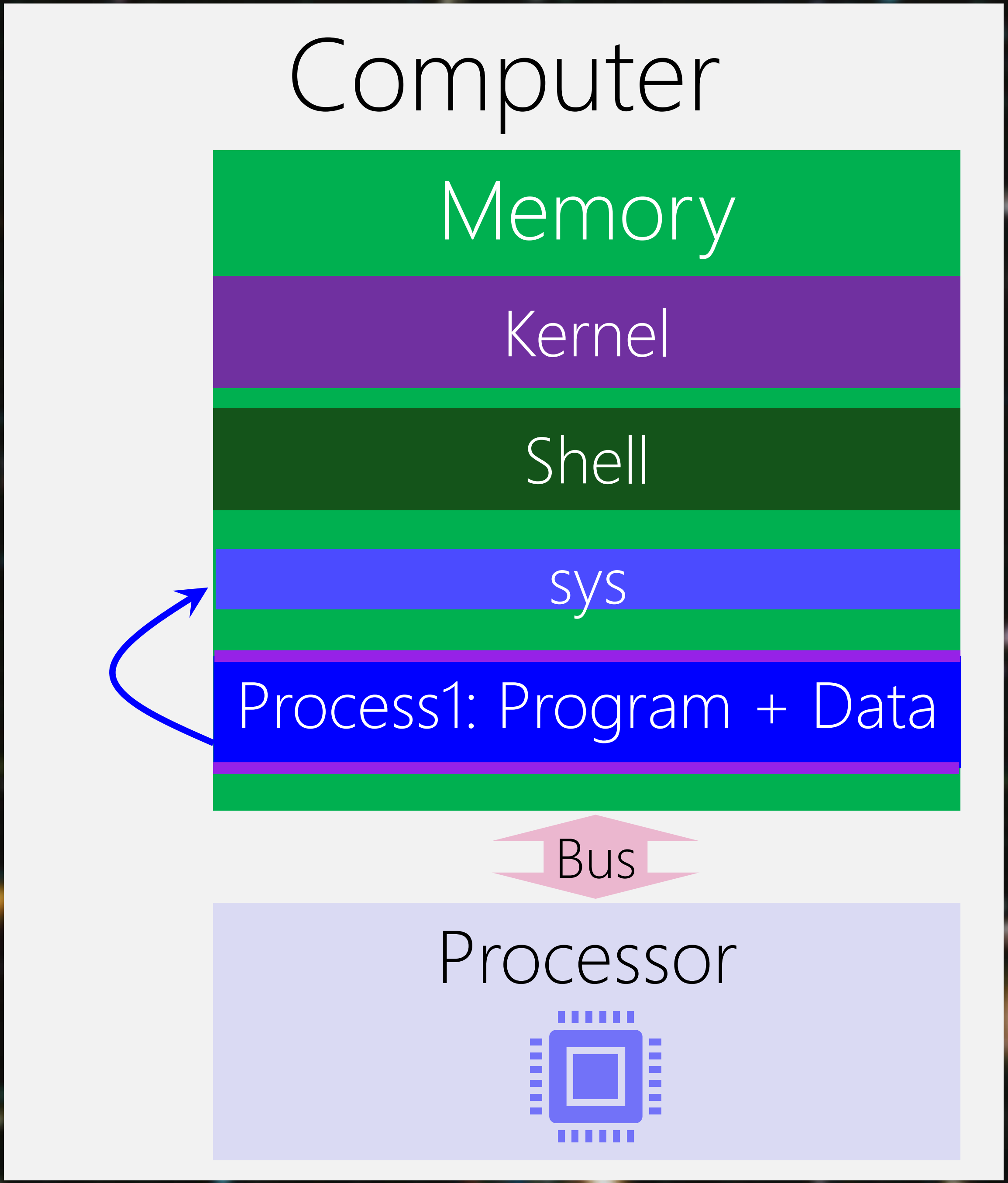


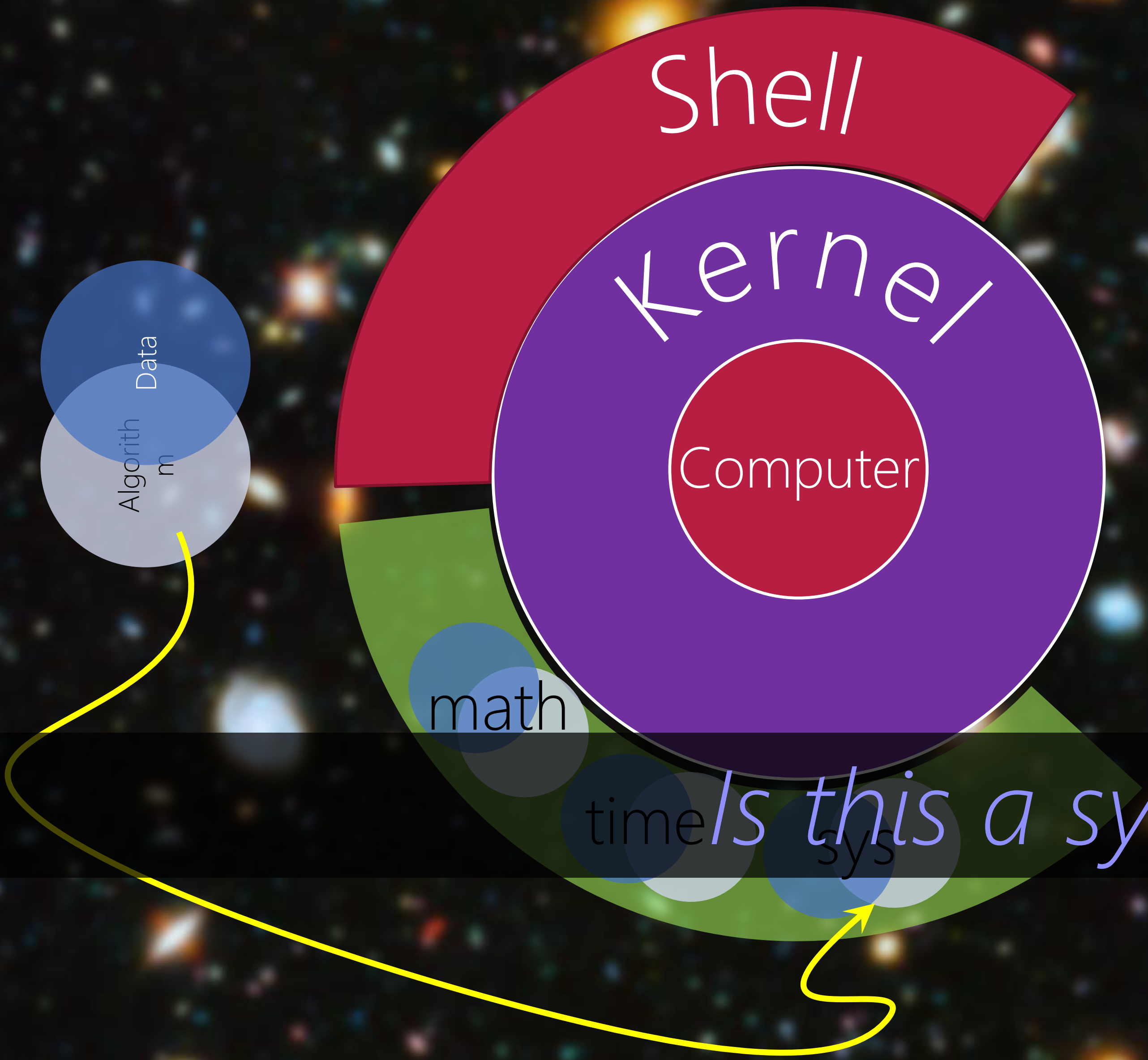
Data
Algorithm
Shell
Kernel
Computer
math
time
sys

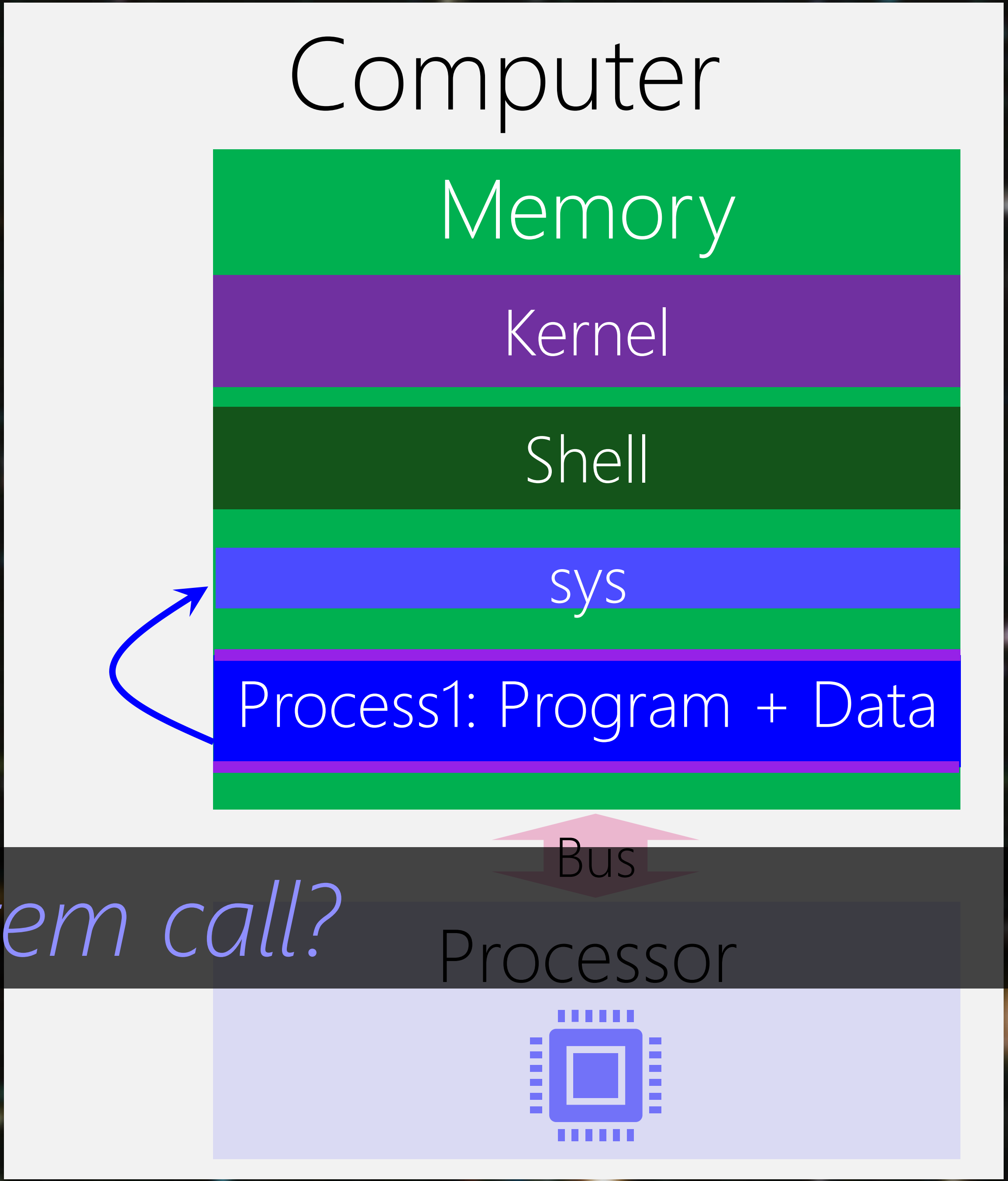


Computer
Memory
Kernel
Shell
sys
Process1: Program + Data
Bus
Processor

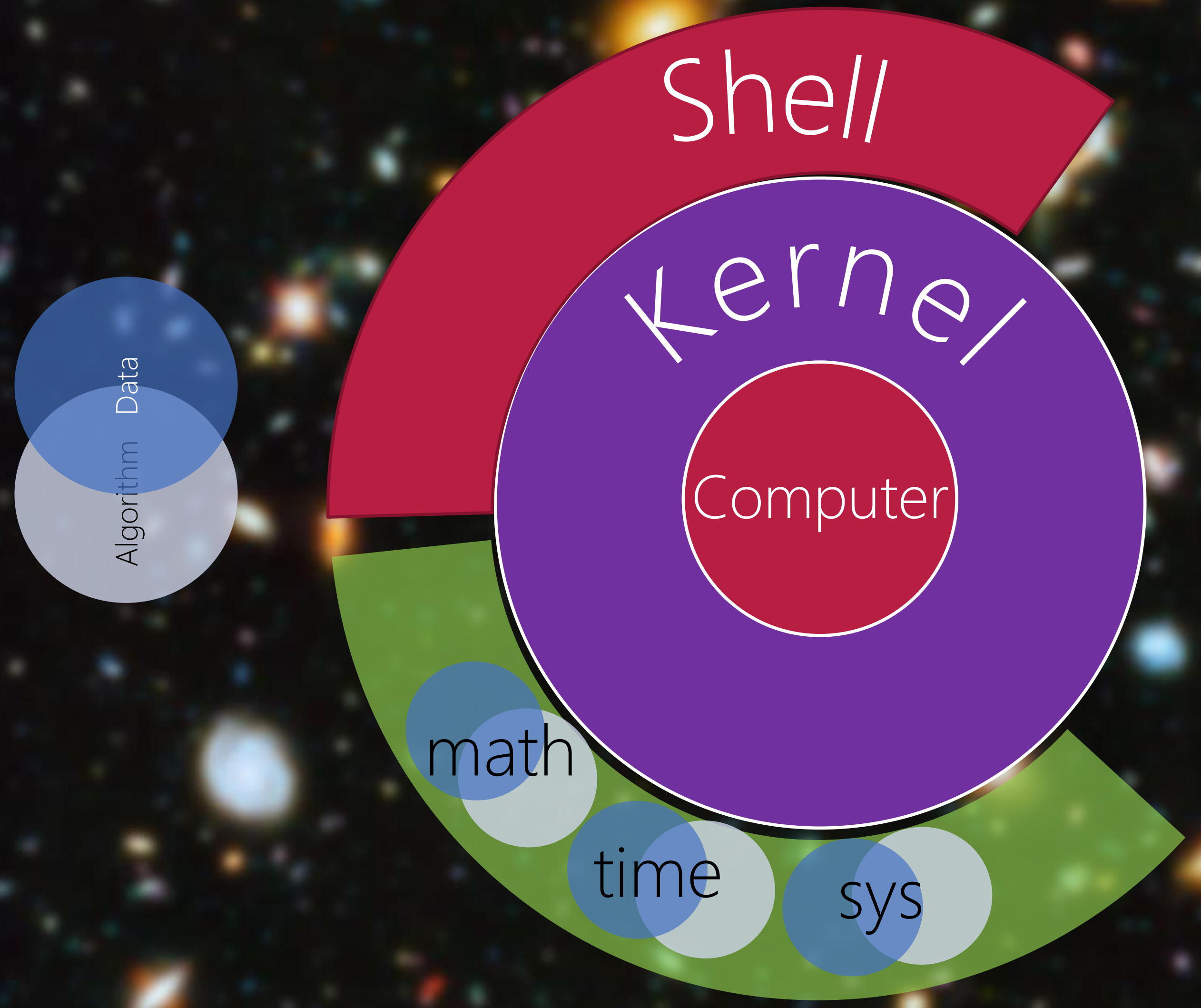

Shell
Kernel
Computer
Data
Algorithm
math
time
sys

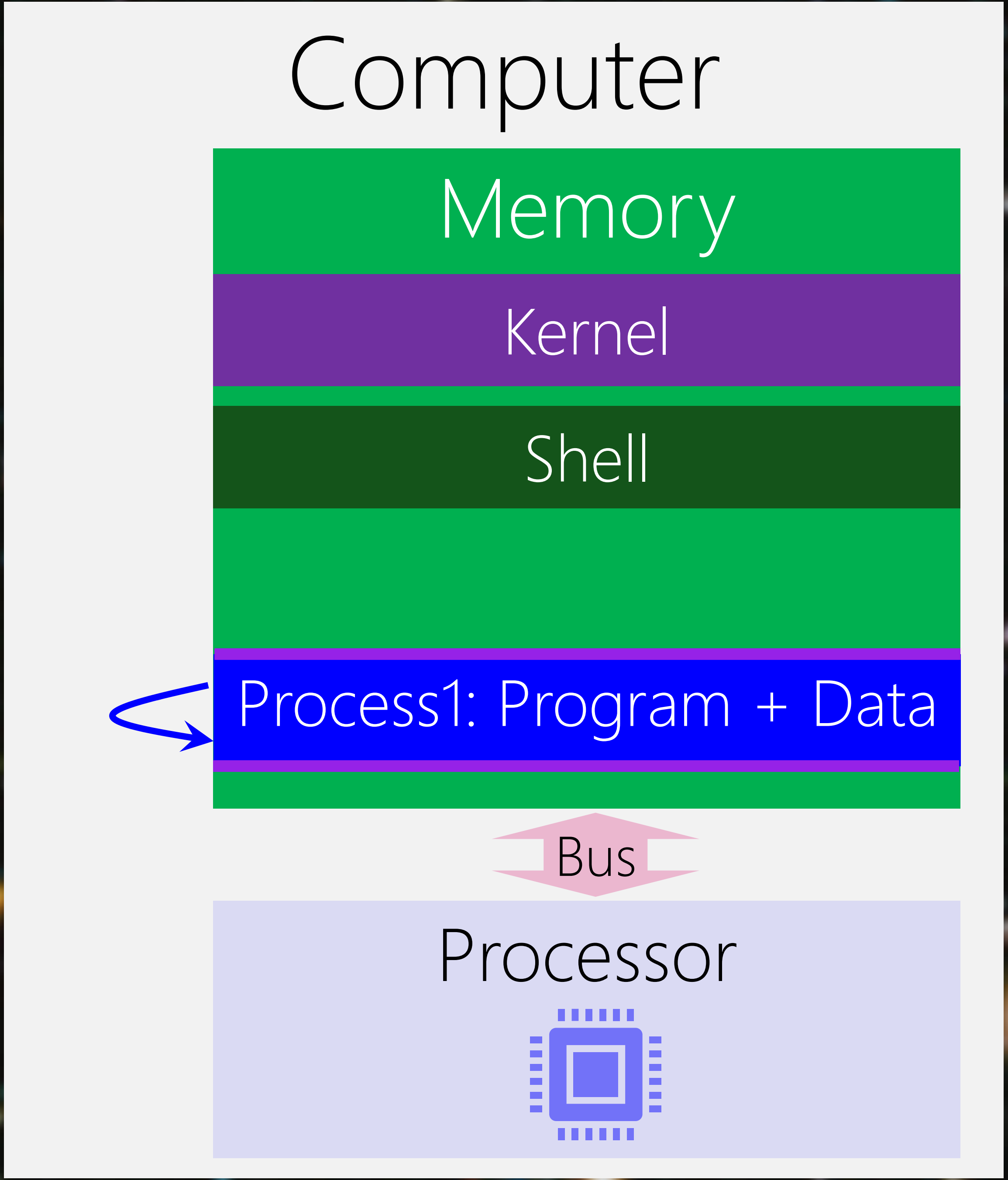

Computer
Memory
Kernel
Shell
sys
Process1: Program + Data
Bus
Processor

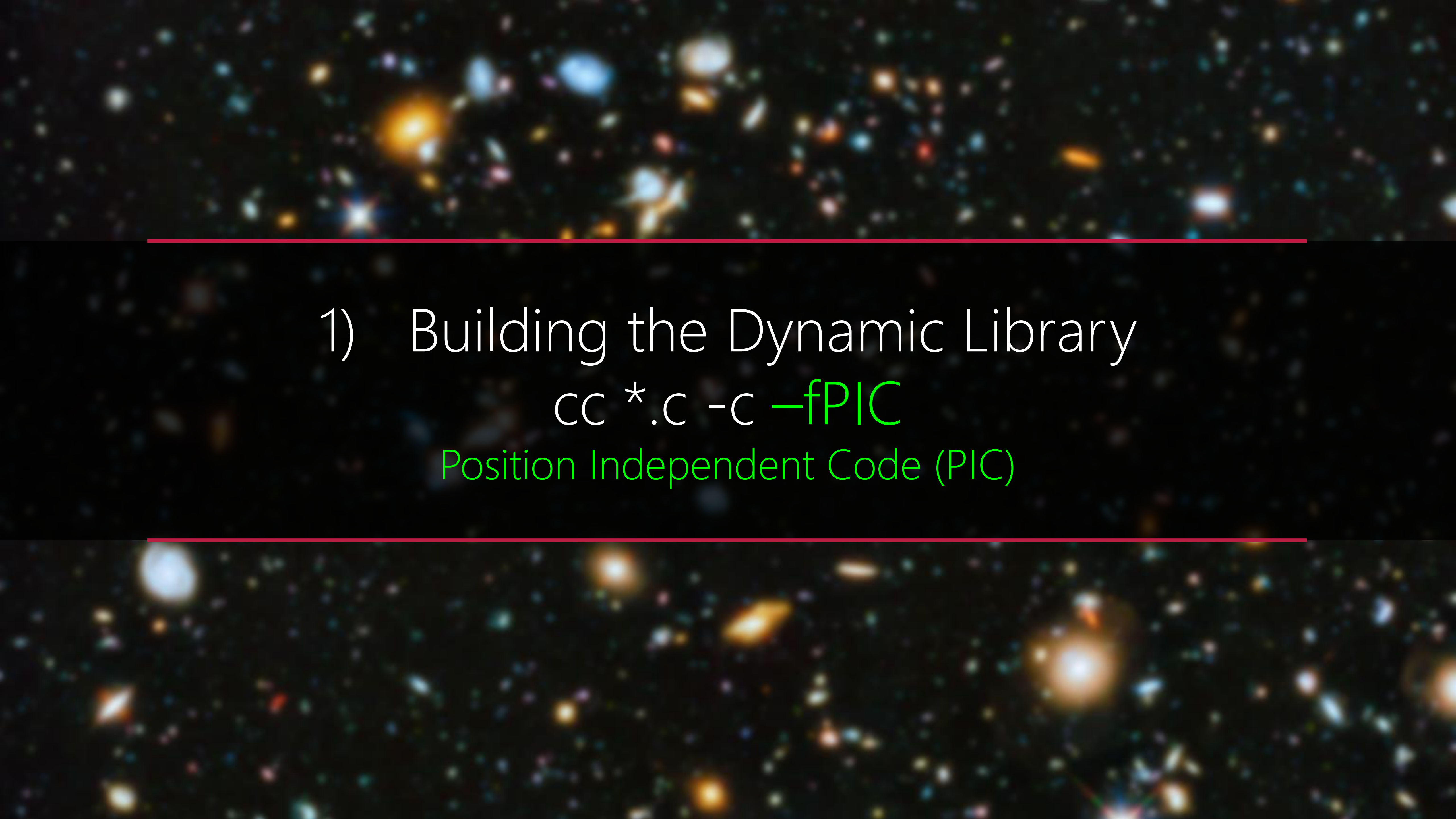




Is this a system call?


Data
Algorithm
Shell
Kernel
Computer
math
time
sys

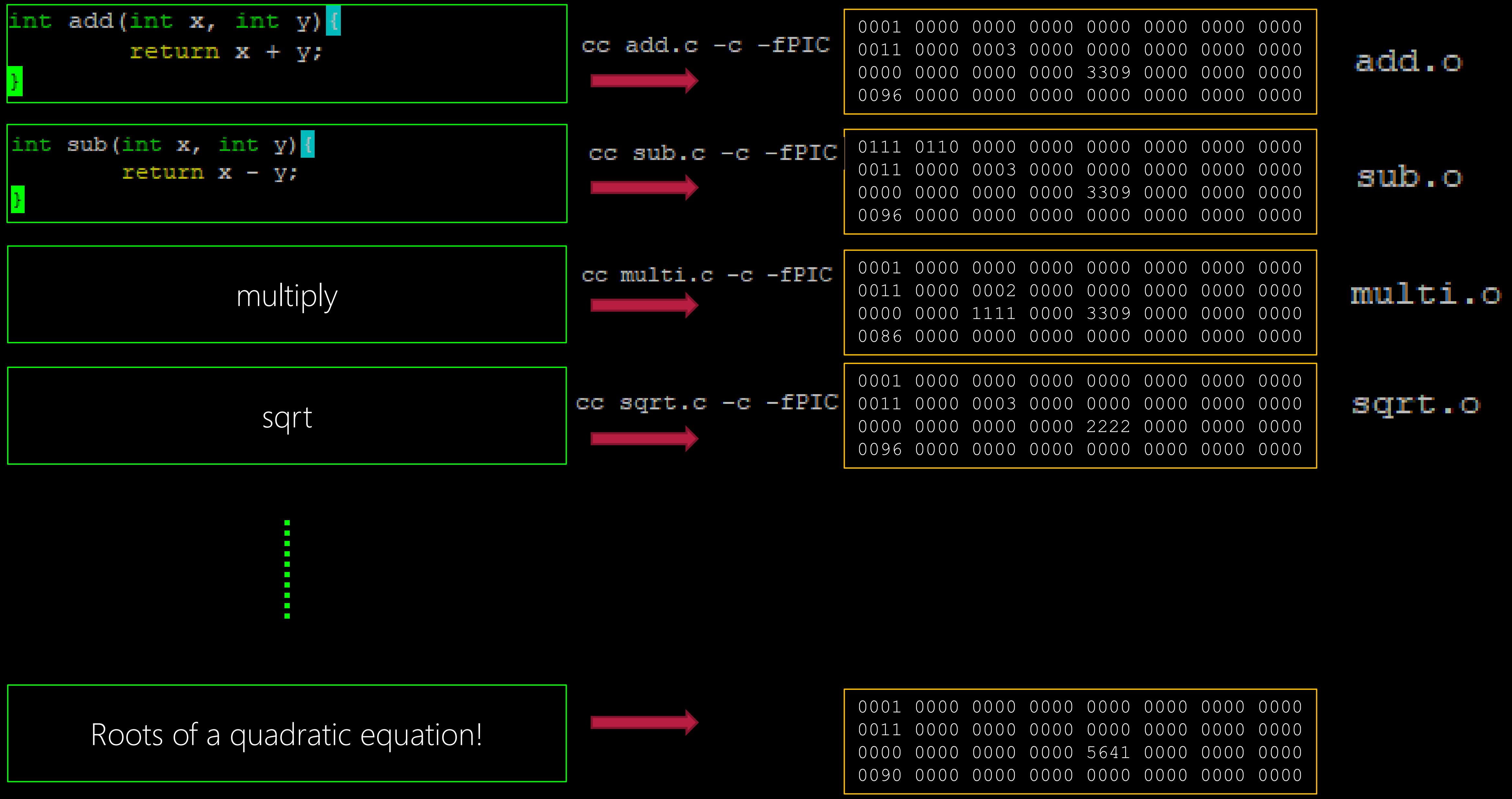

Computer
Memory
Kernel
Shell
Process1: Program + Data
Bus
Processor



1) Building the Dynamic Library

```
cc *.c -c –fPIC
```

Position Independent Code (PIC)



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

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

```
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
```

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

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

```
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
```

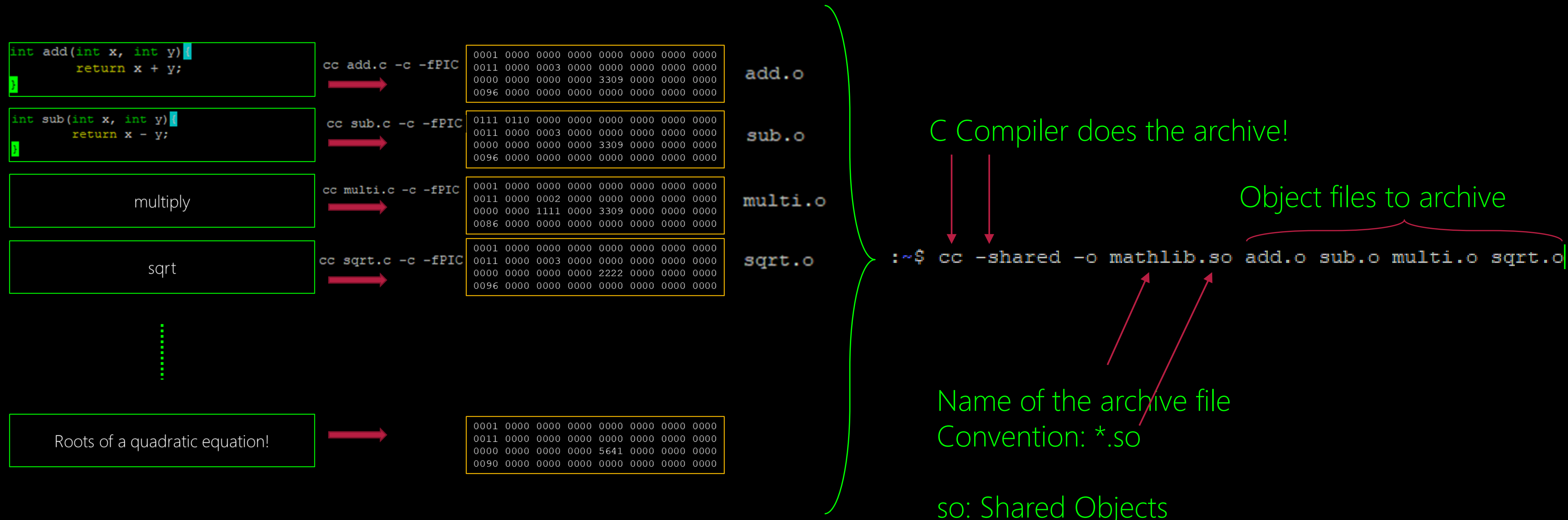
```
cc multi.c -c -fPIC
```

```
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
```

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

```
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
```

```
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
```



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

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

```
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 -fPIC
```

```
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 -fPIC
```

```
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 -fPIC
```

```
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
```

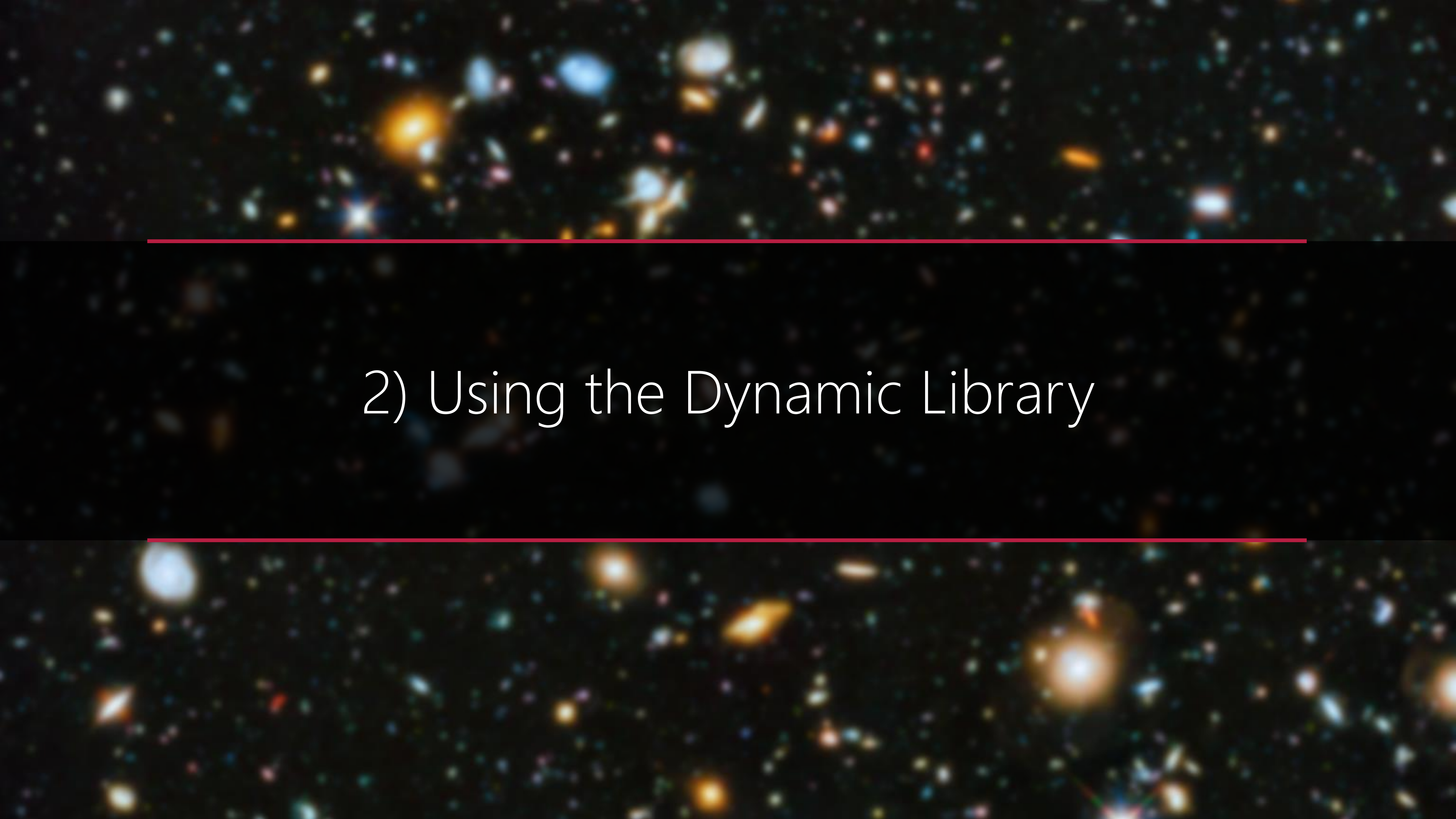
C Compiler does the archive!

Object files to archive

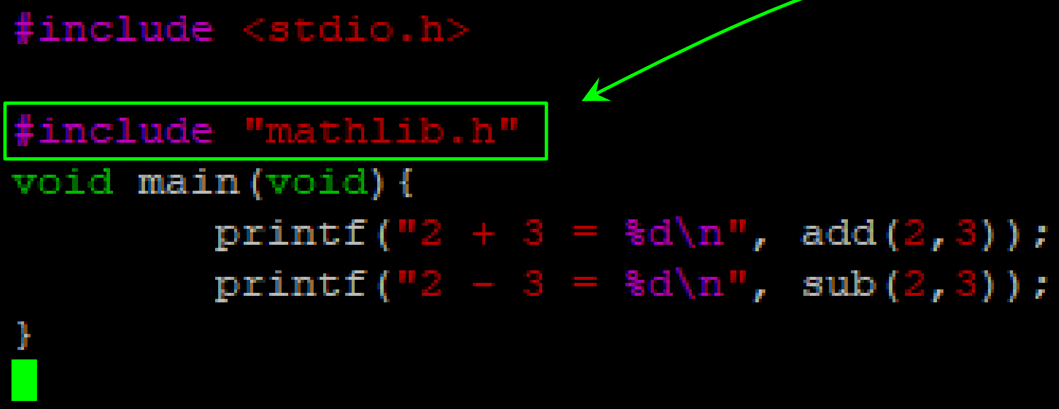
```
:~$ cc -shared -o mathlib.so add.o sub.o multi.o sqrt.o
```

Name of the archive file
Convention: *.so

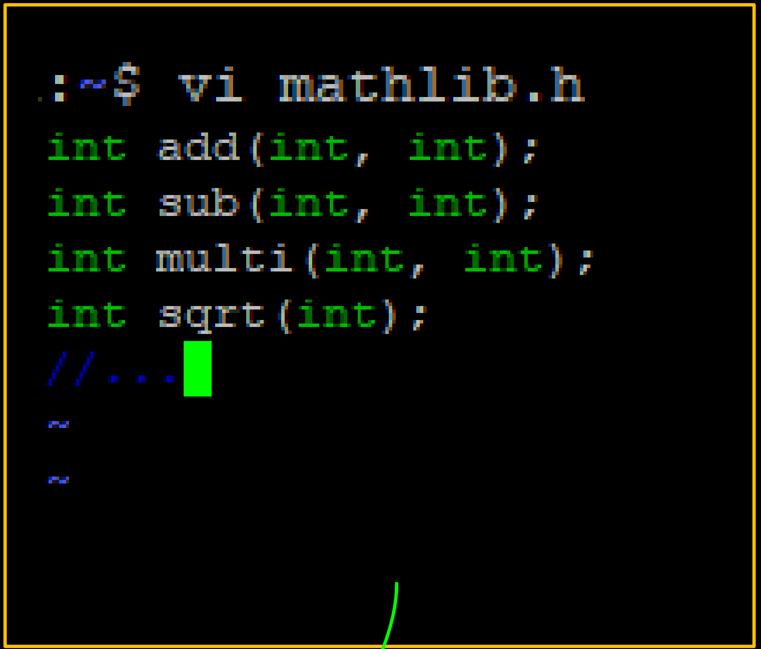
so: Shared Objects



2) Using the Dynamic Library

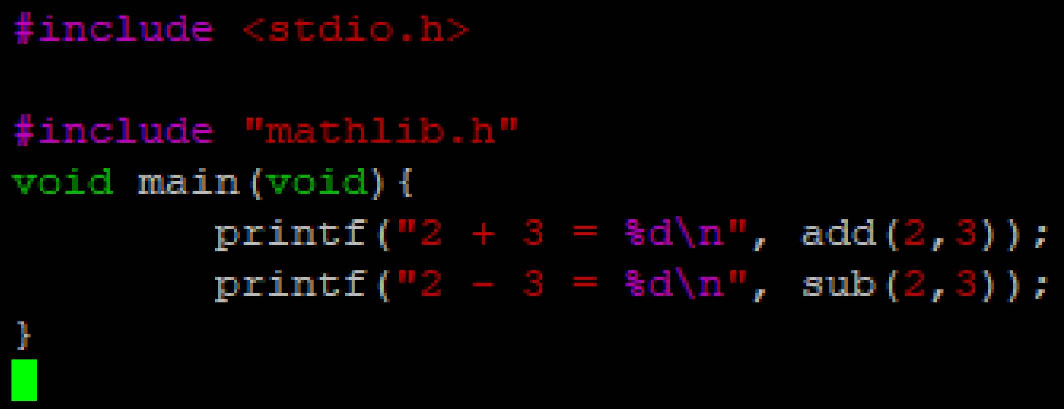


```c
#include <stdio.h>

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

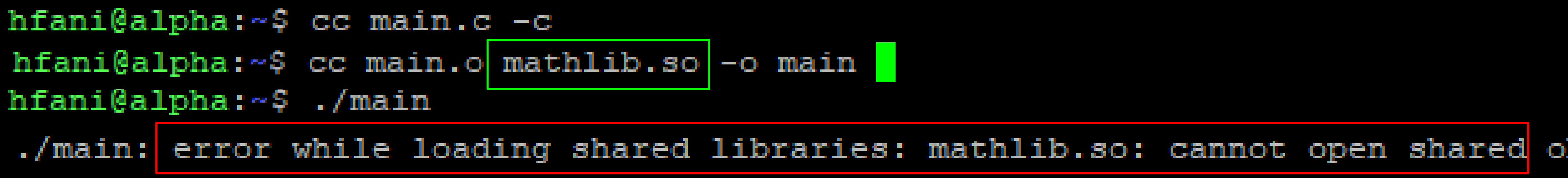


```
:~$ vi mathlib.h
int add(int, int);
int sub(int, int);
int multi(int, int);
int sqrt(int);
//...
~
~
```

The same as in using static library!

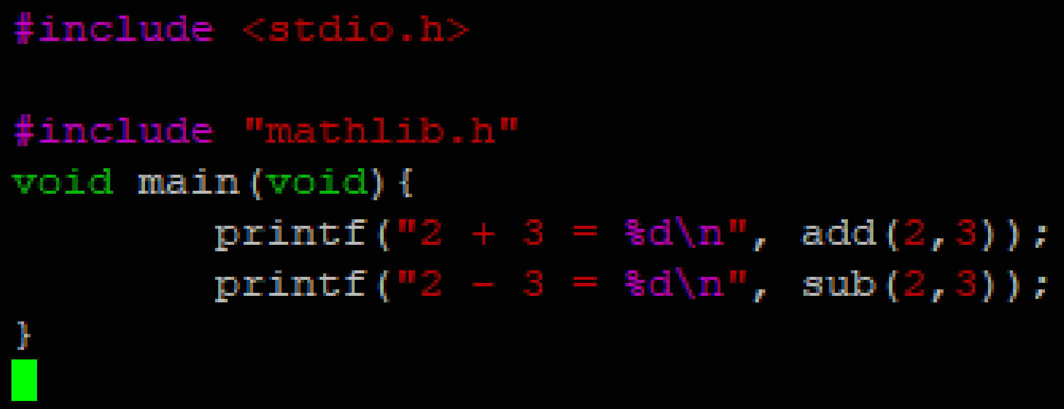


```c
#include <stdio.h>

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

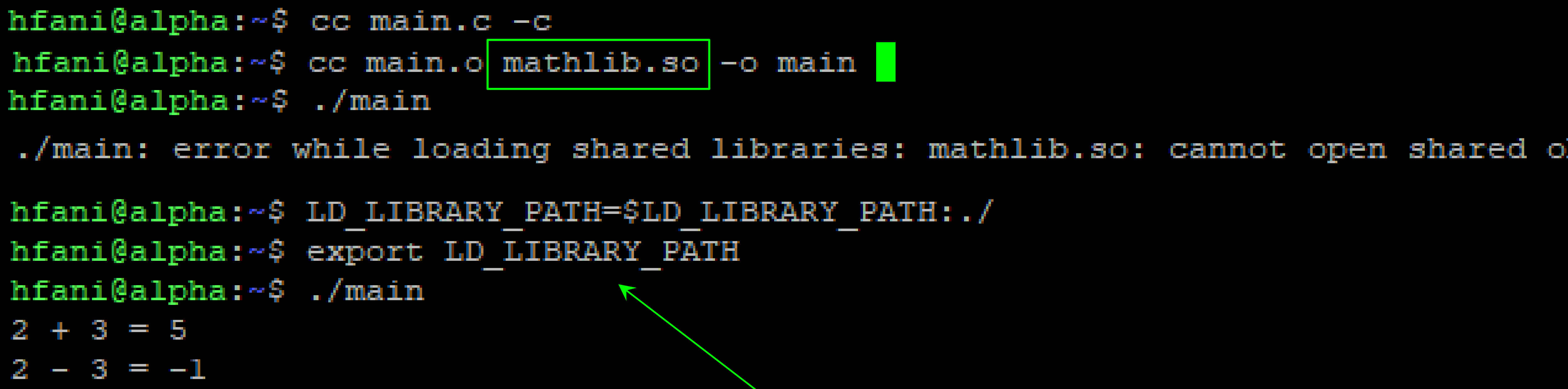


```
hfani@alpha:~$ cc main.c -c
hfani@alpha:~$ cc main.o mathlib.so -o main
hfani@alpha:~$ ./main
./main: error while loading shared libraries: mathlib.so: cannot open shared o
```

Either put your lib in the standard location
Or tell the shell the kernel where to find it

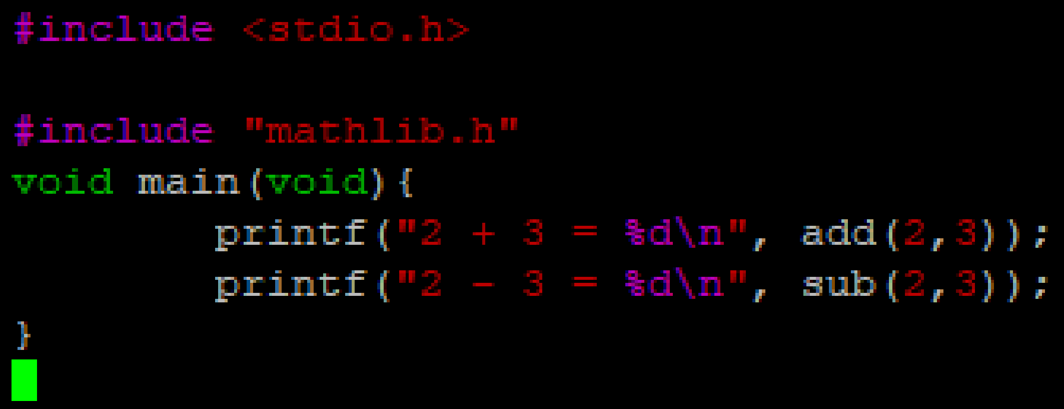


```
#include <stdio.h>

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

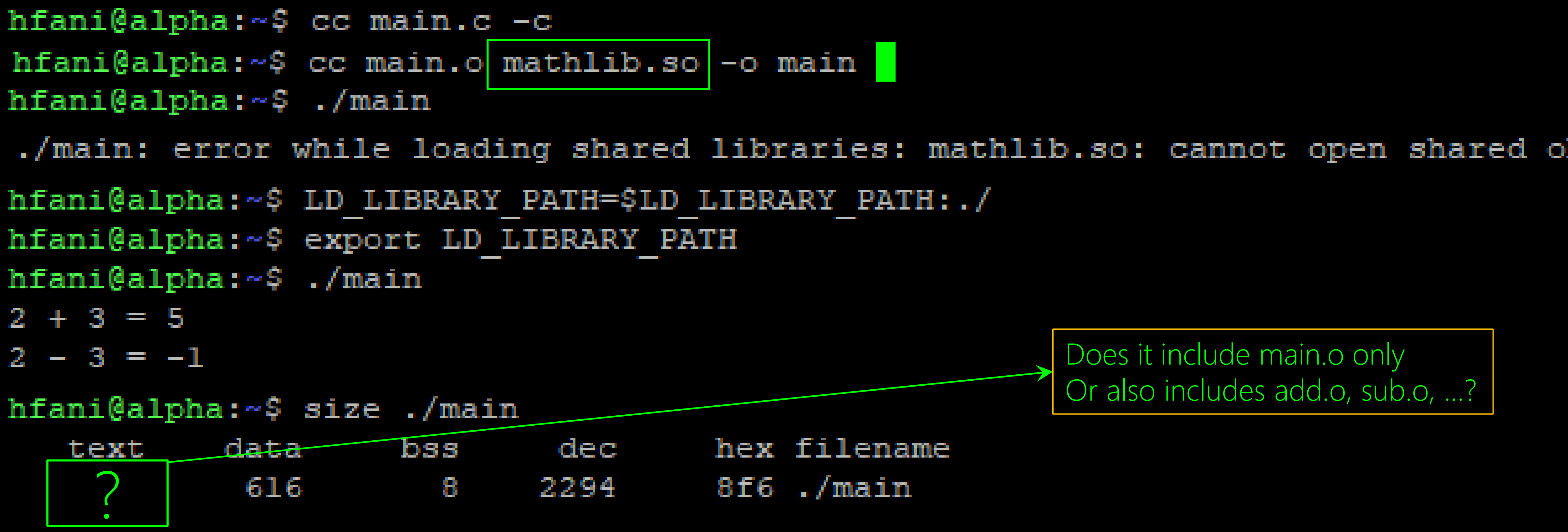


```
hfani@alpha:~$ cc main.c -c
hfani@alpha:~$ cc main.o mathlib.so -o main
hfani@alpha:~$ ./main
./main: error while loading shared libraries: mathlib.so: cannot open shared o
hfani@alpha:~$ LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./
hfani@alpha:~$ export LD_LIBRARY_PATH
hfani@alpha:~$ ./main
2 + 3 = 5
2 - 3 = -1
```

System Variable for the path of shared libs



```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));
}
```

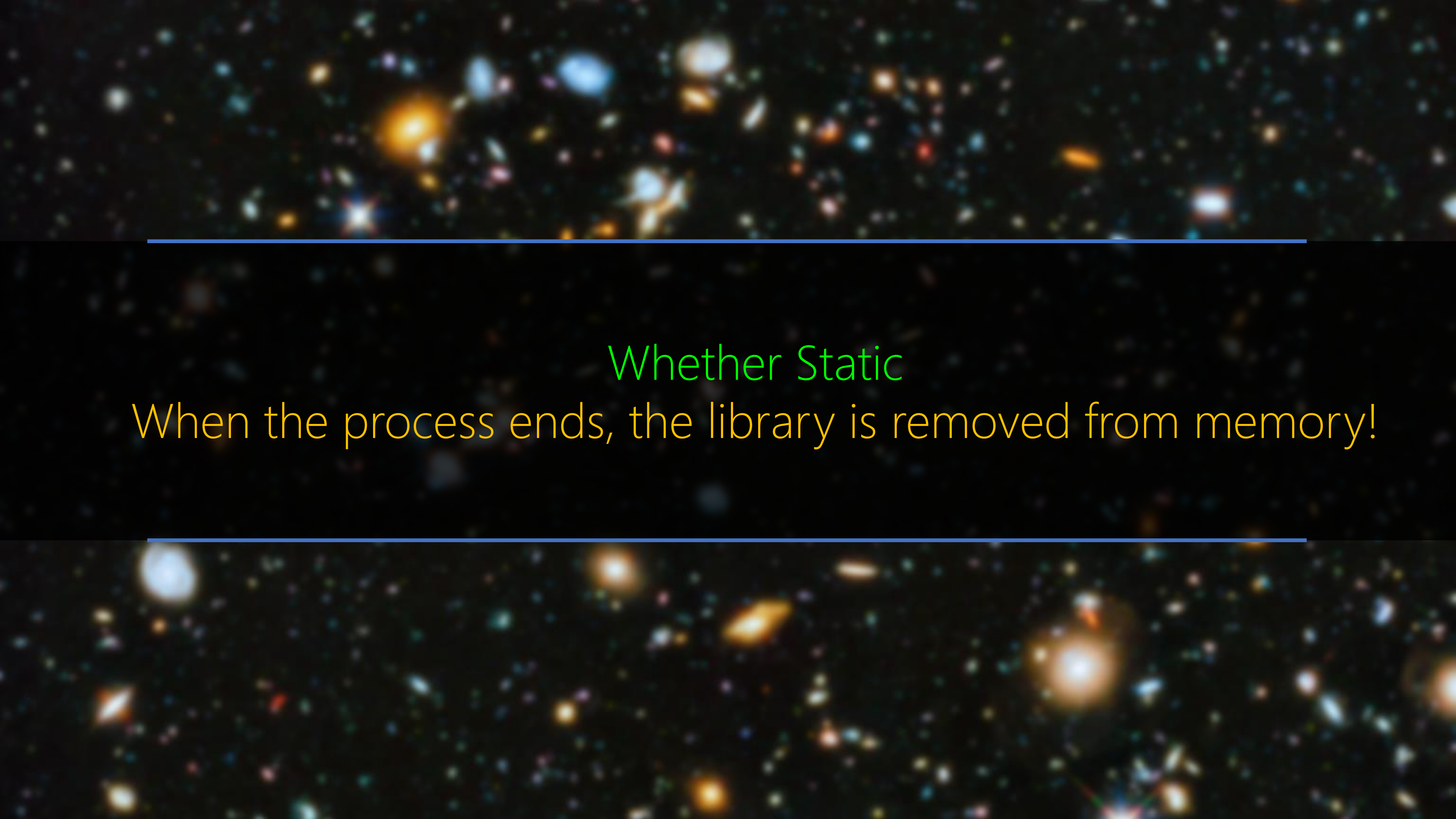


```
hfani@alpha:~$ cc main.c -c
hfani@alpha:~$ cc main.o mathlib.so -o main
hfani@alpha:~$ ./main
./main: error while loading shared libraries: mathlib.so: cannot open shared o
hfani@alpha:~$ LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./
hfani@alpha:~$ export LD_LIBRARY_PATH
hfani@alpha:~$ ./main
2 + 3 = 5
2 - 3 = -1
hfani@alpha:~$ size ./main
   text    data     bss     dec     hex filename
      ?     616       8    2294     8f6 ./main
```

Does it include main.o only
Or also includes add.o, sub.o, …?

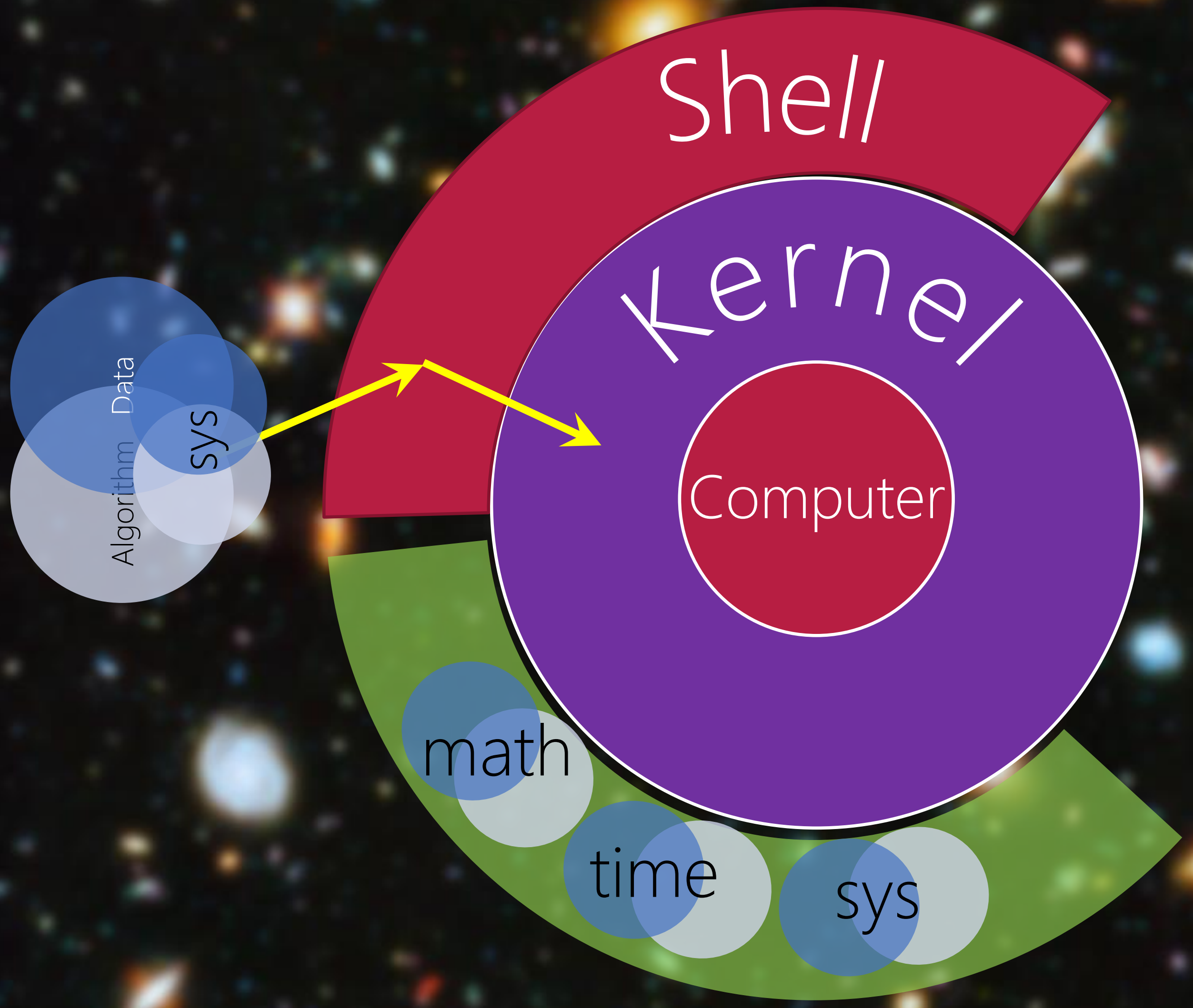
STATIC vs. DYNAMIC

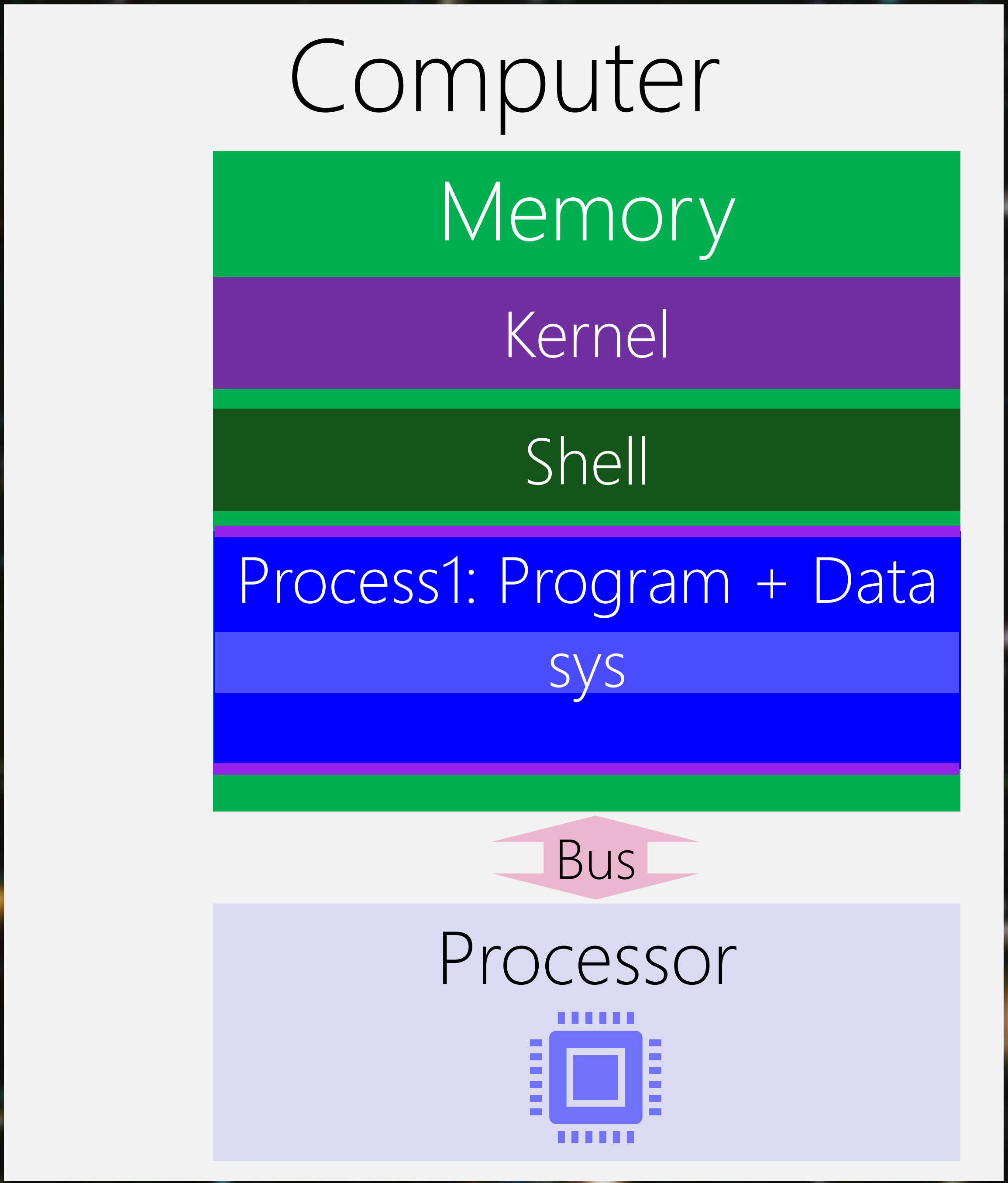
Speed vs. Memory



Whether Static

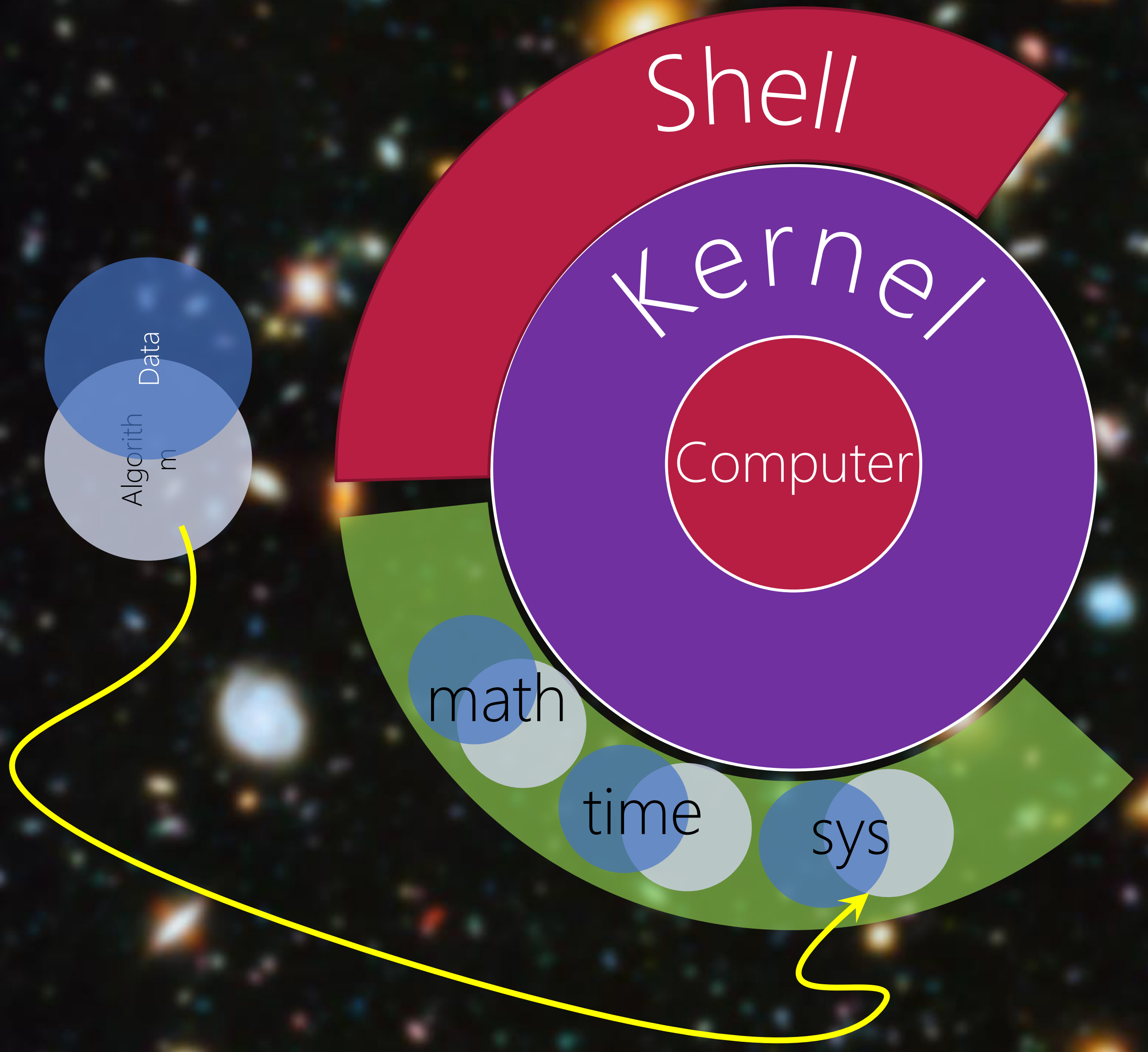
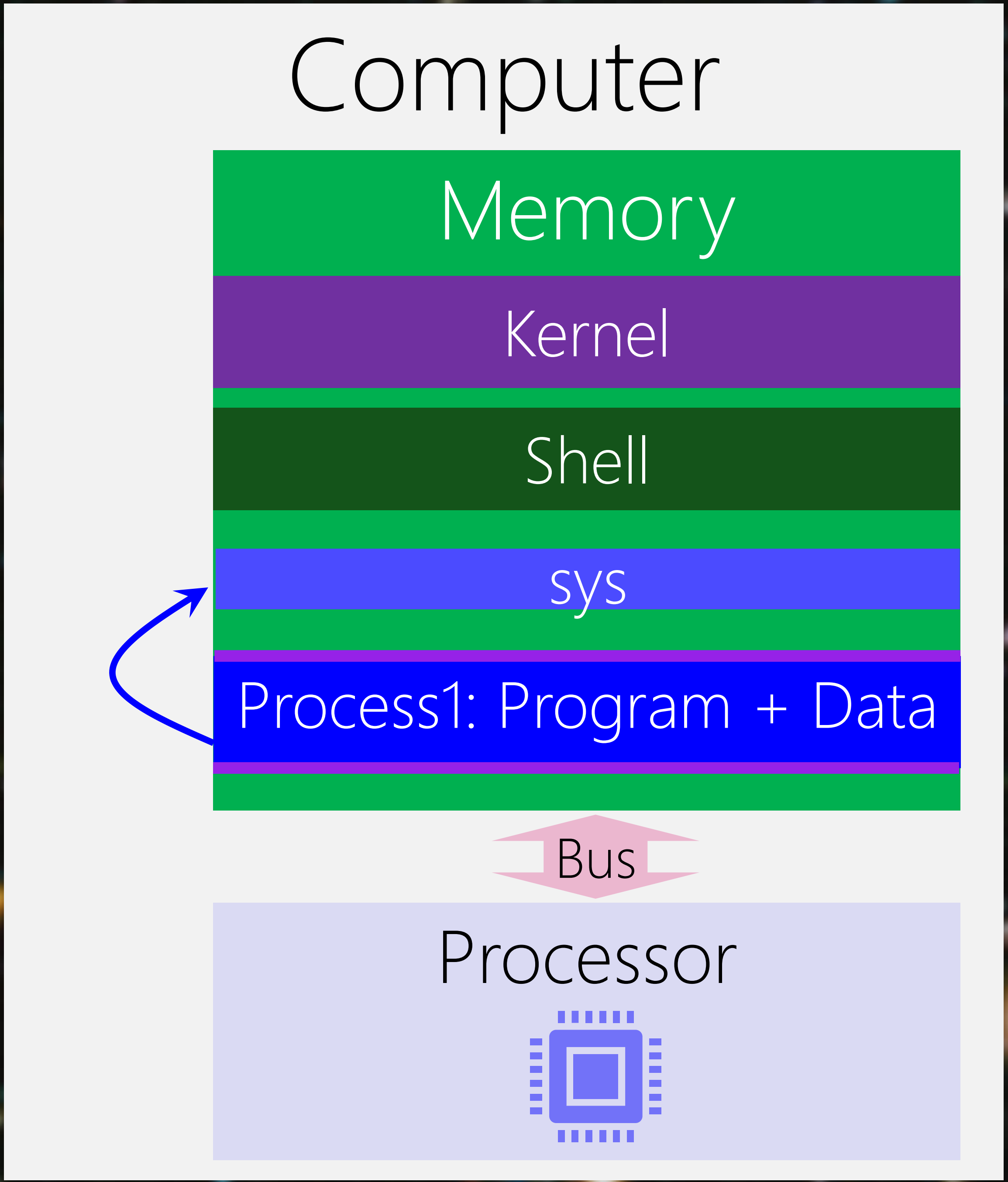
When the process ends, the library is removed from memory!

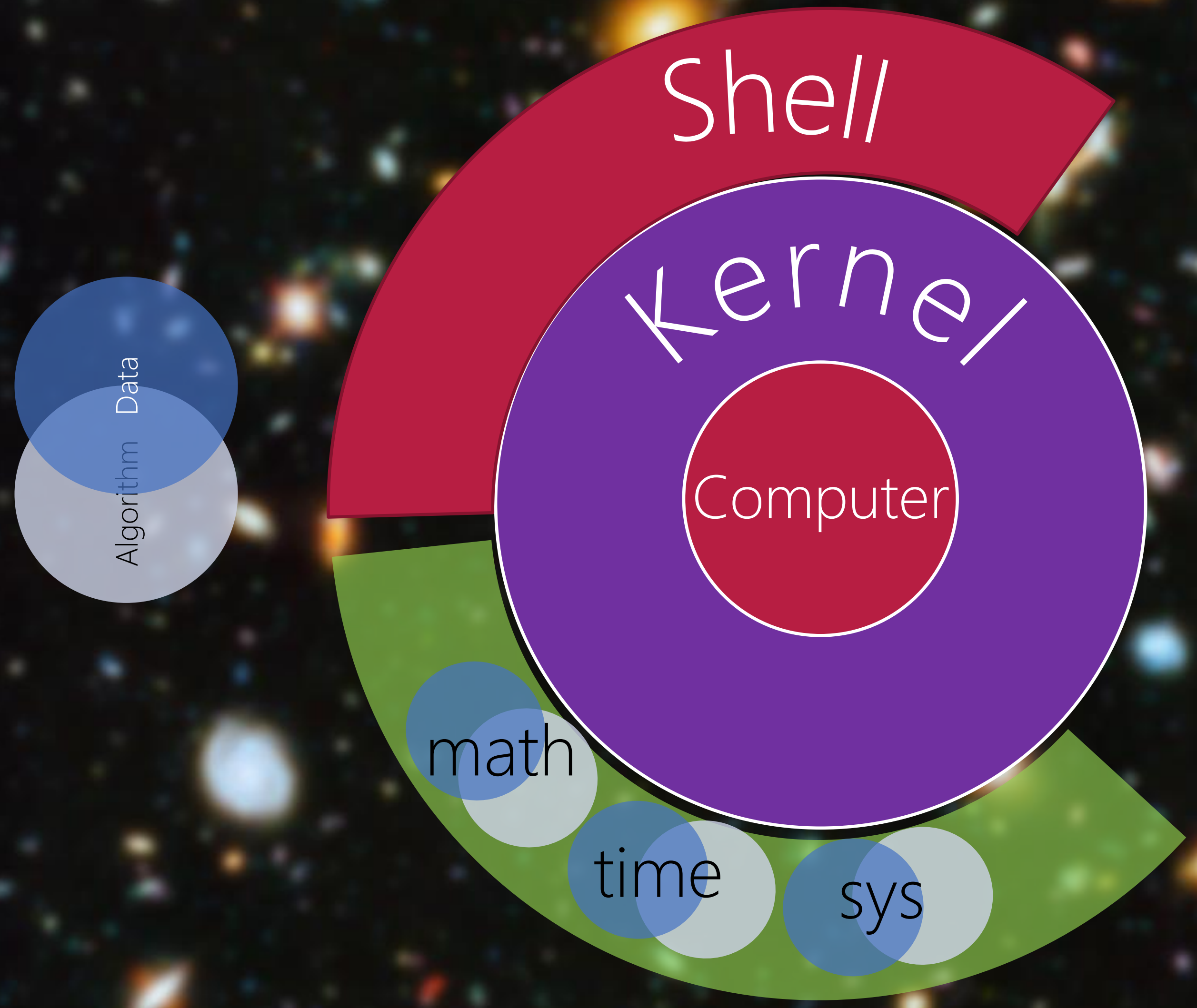

Shell
Kernel
Computer
Data
Algorithm
sys
math
time
sys

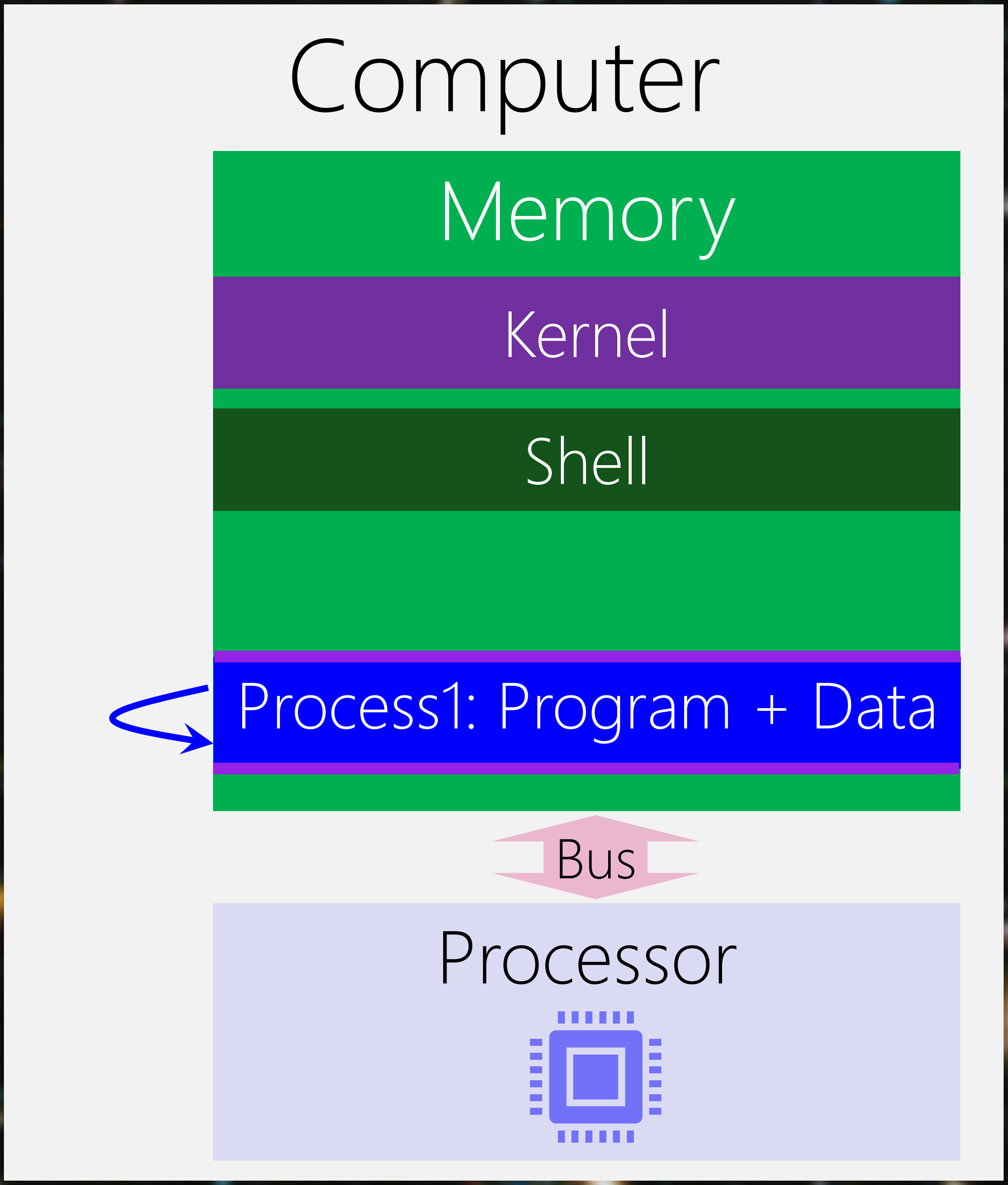

Computer
Memory
Kernel
Shell
Process1: Program + Data
sys
Bus
Processor

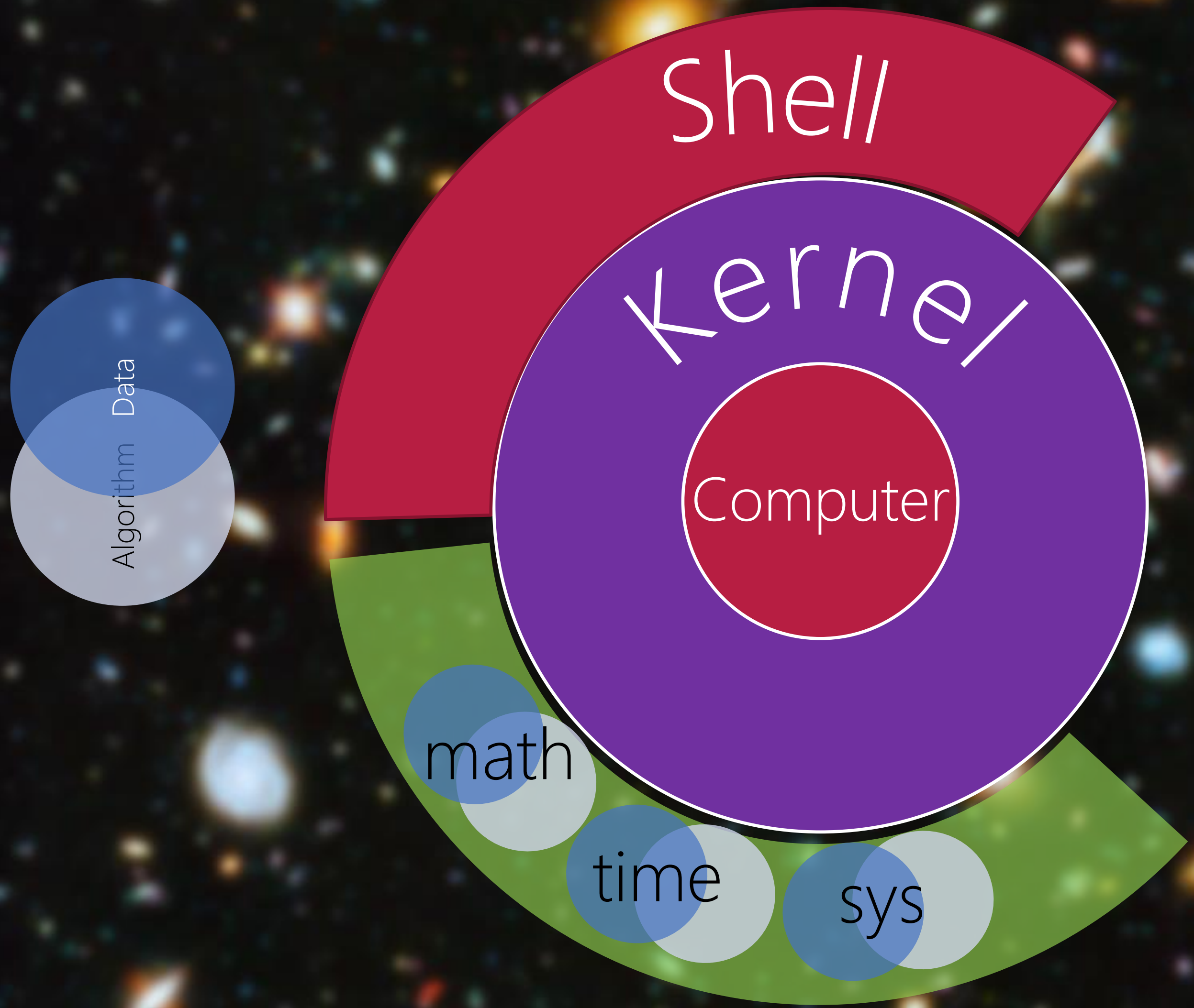
or Dynamic

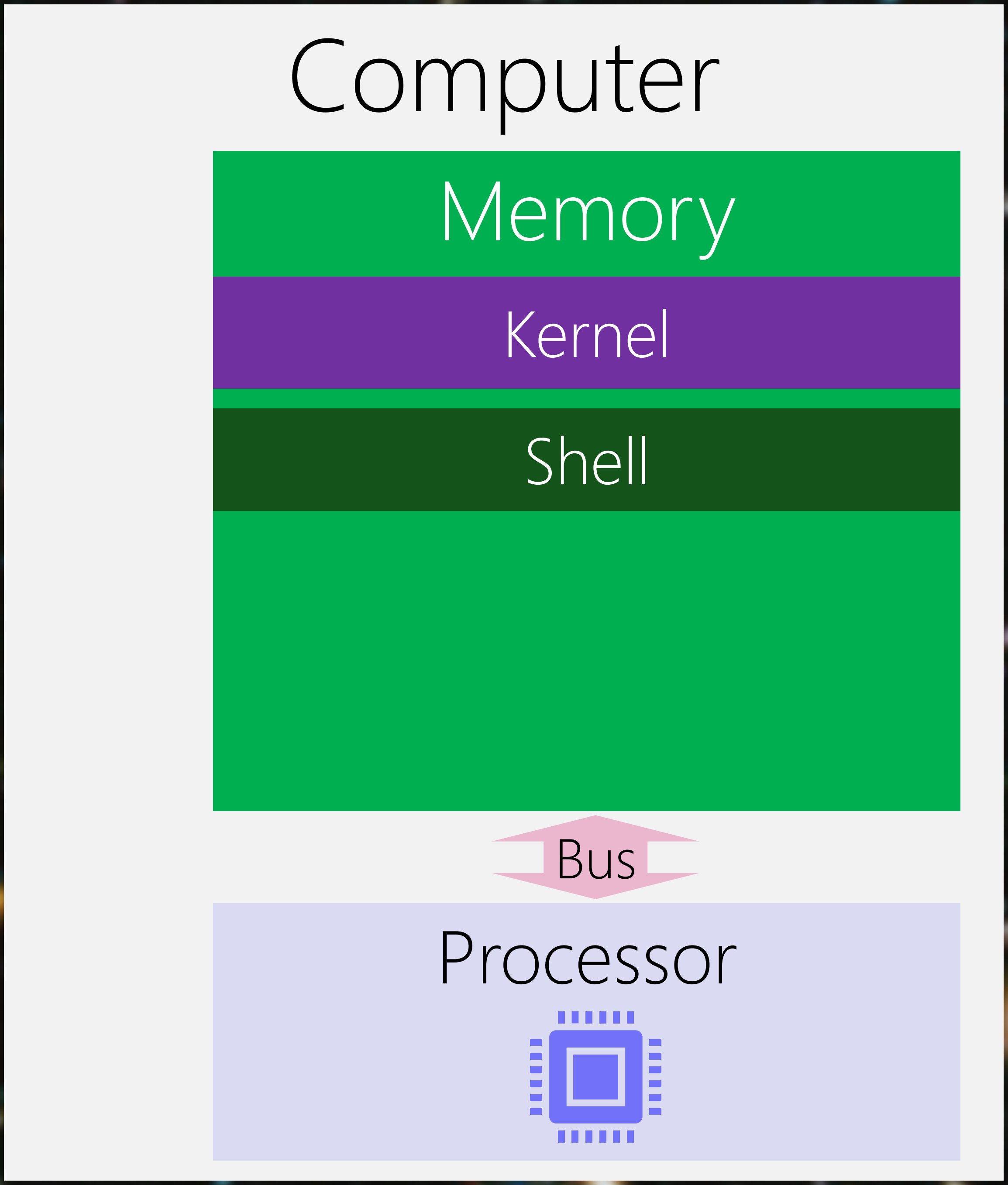
When the process ends, the library is removed from memory!

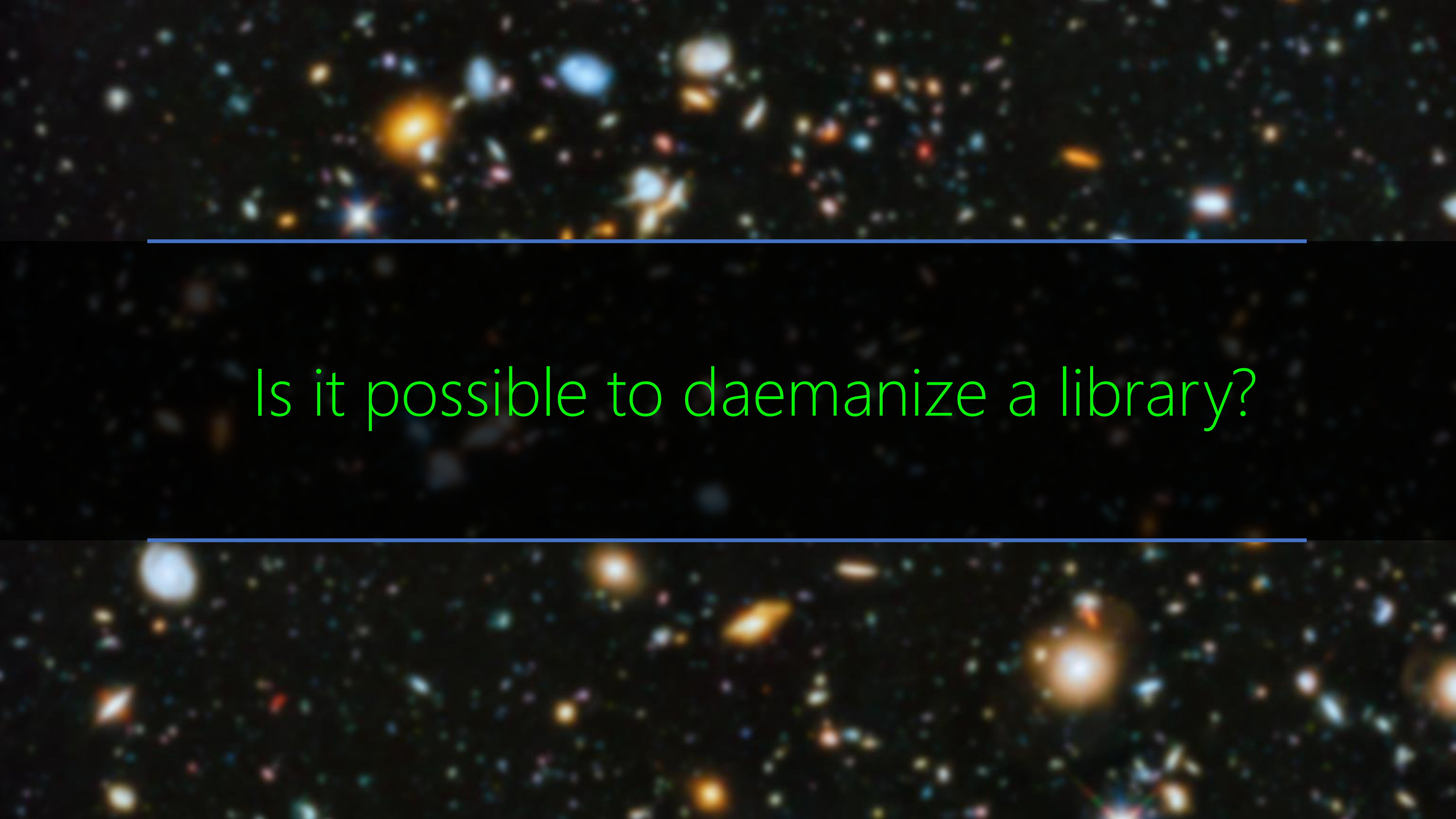

Shell
Kernel
Computer
Data
Algorithm
math
time
sys

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


Data
Algorithm
Shell
Kernel
Computer
math
time
sys


Computer
Memory
Kernel
Shell
Process1: Program + Data
Bus
Processor


Shell
Kernel
Computer
Data
Algorithm
math
time
sys

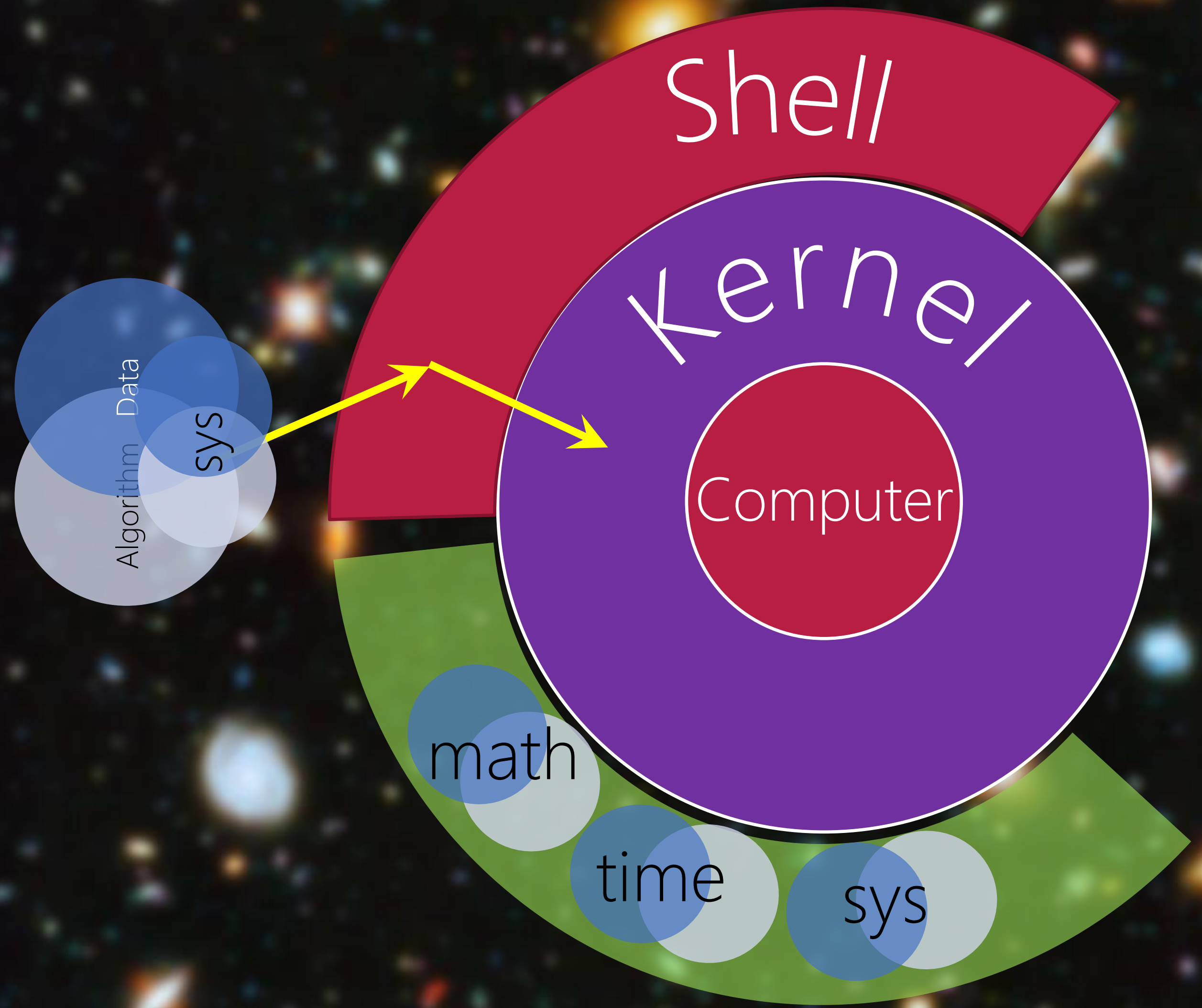

Computer
Memory
Kernel
Shell
Bus
Processor

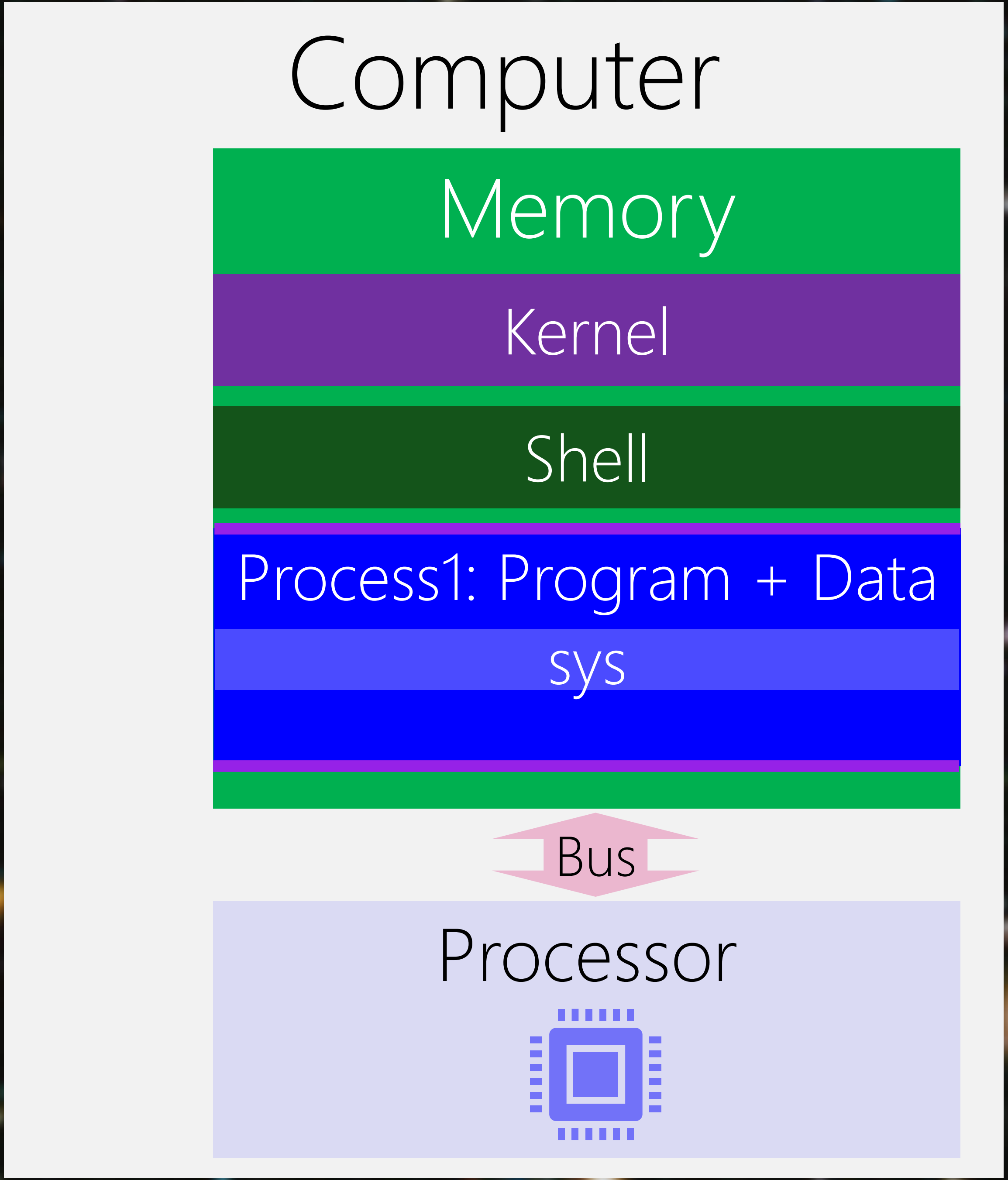


Is it possible to daemanize a library?

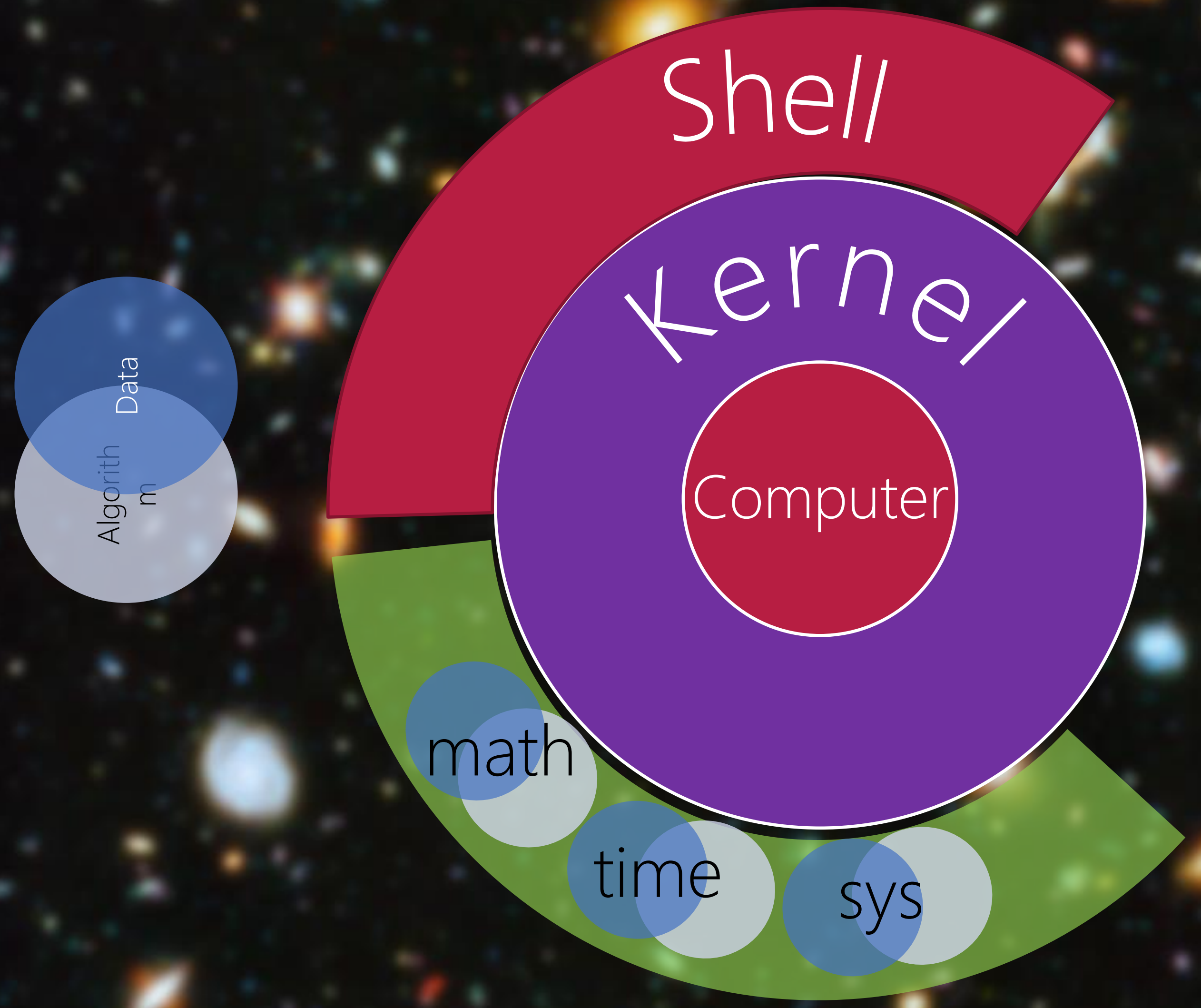
Is it possible to daemanize a library?

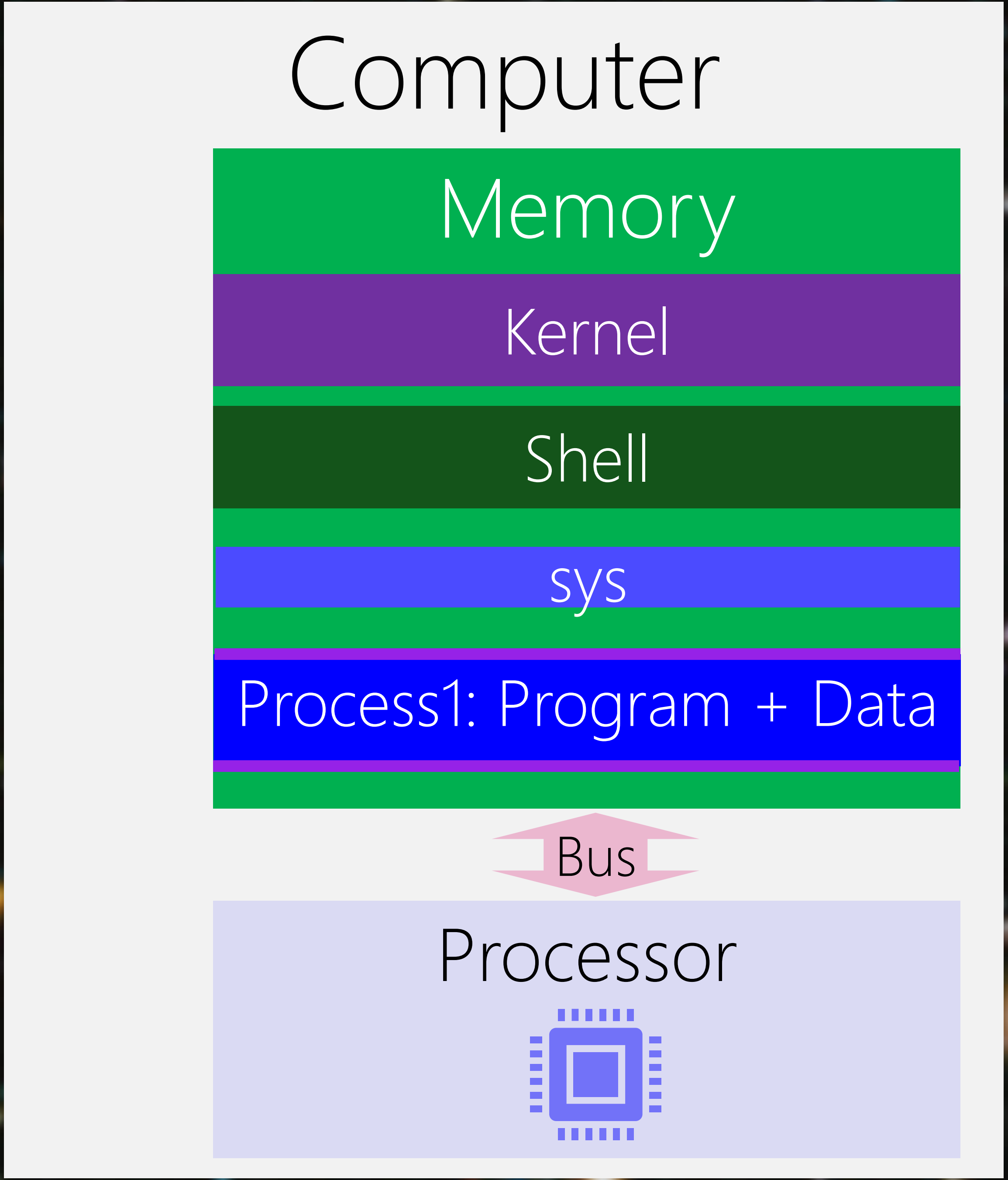
Static Library: No (Why?)


Shell
Kernel
Computer
Data
Algorithm
sys
math
time
sys


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

Is it possible to daemanize a library?
Dynamic Library: Yes (Why?)


Data
Algorithm
Shell
Kernel
Computer
math
time
sys


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

Shared Libraries

Daemon Dynamic Libraries

Shared Libraries

Daemon Dynamic Libraries

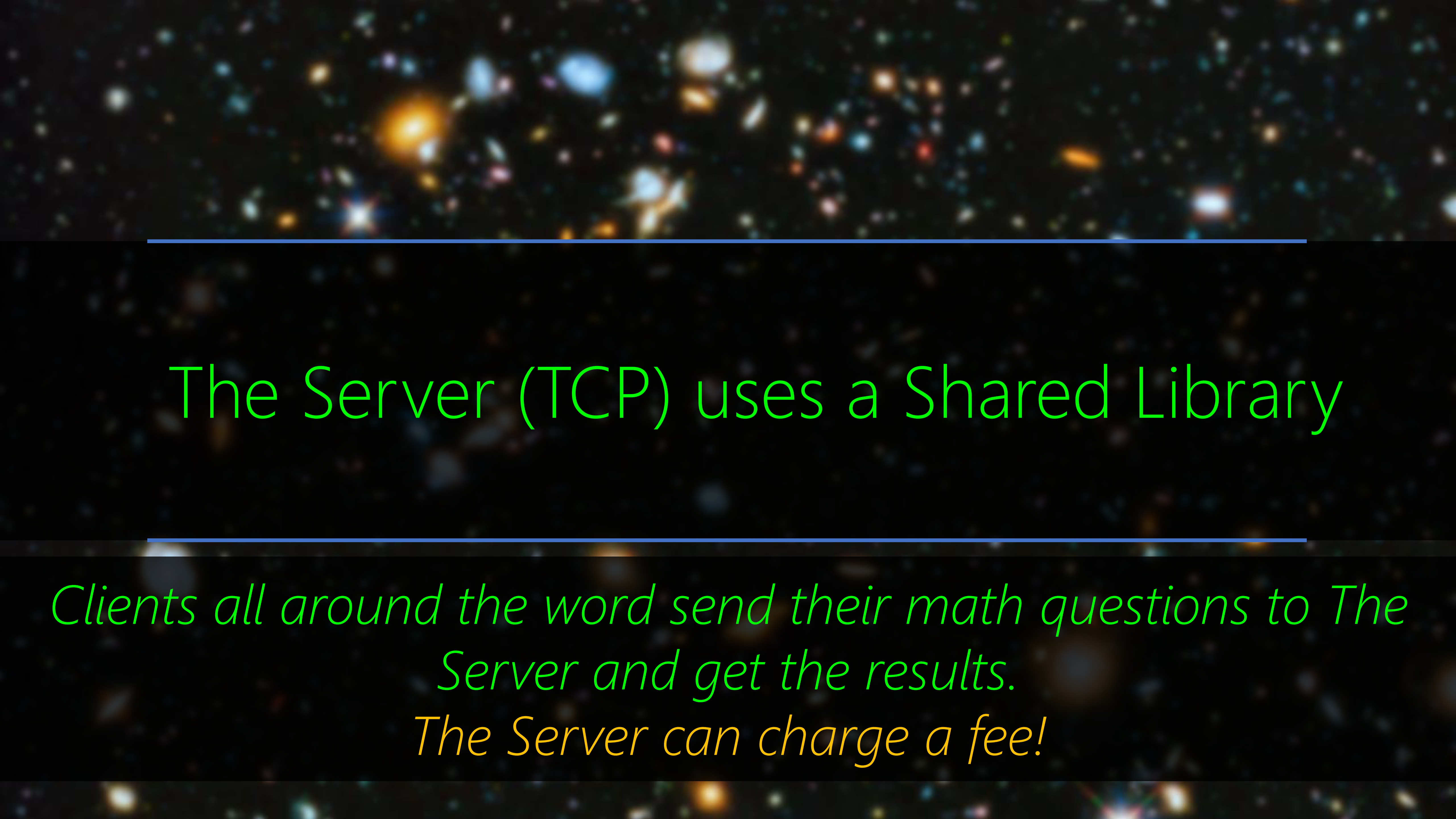
The sharing scope is within the computer!

Globally Shared Libraries

All around the world, processes can use a single shared library

Is it possible to share the library with other computers?

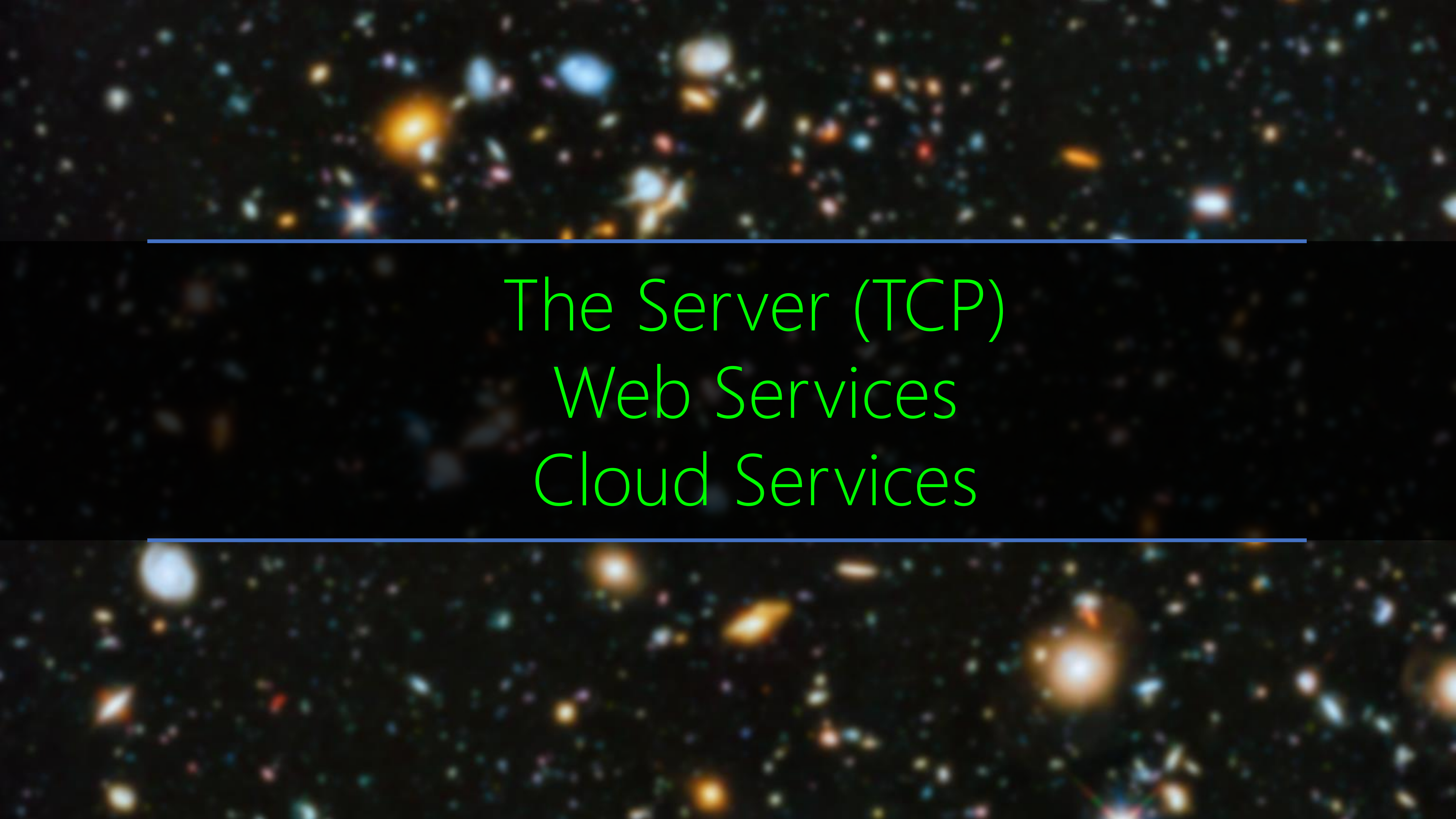
sample final exam question



The Server (TCP) uses a Shared Library

Clients all around the word send their math questions to The Server and get the results.

The Server can charge a fee!



The Server (TCP)

Web Services

Cloud Services