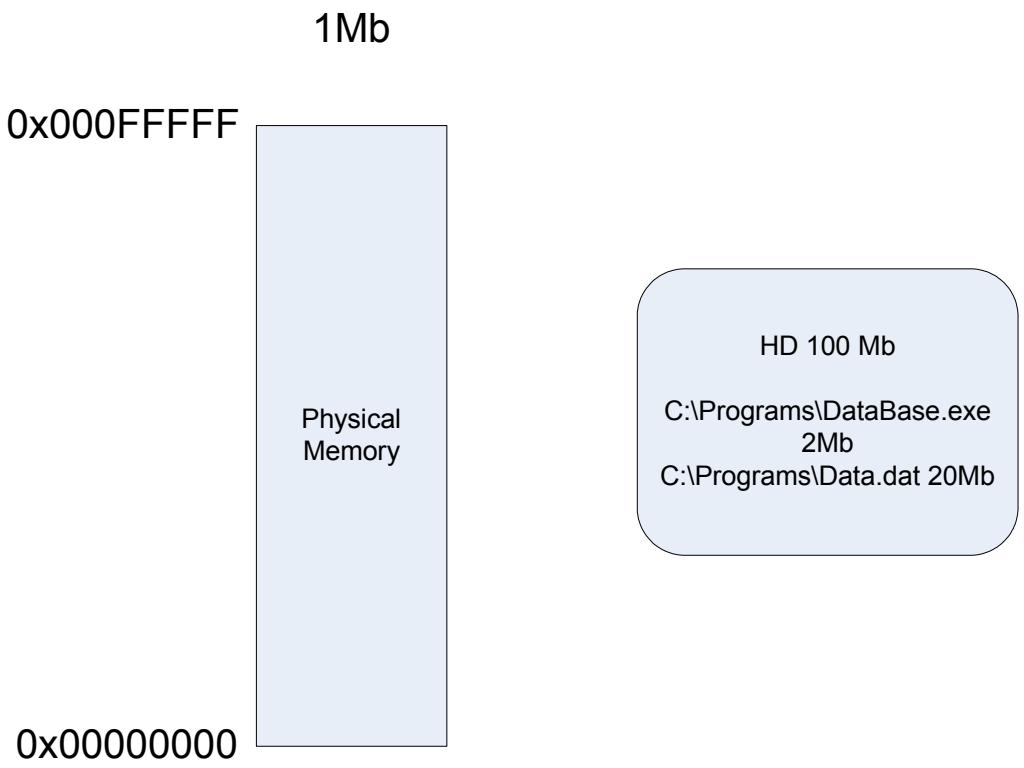
Practical Foundations of Debugging

Chapter 9

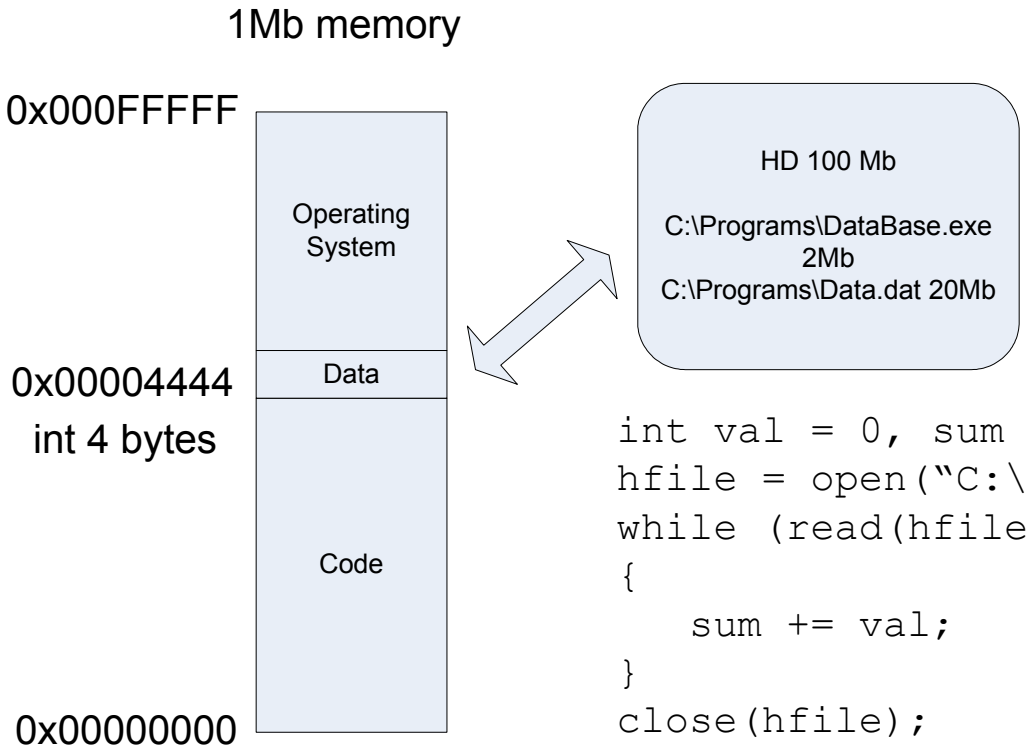
Virtual Memory, Processes and Threads – Part 1

Designed, written and illustrated by Dmitry Vostokov

How to fit?

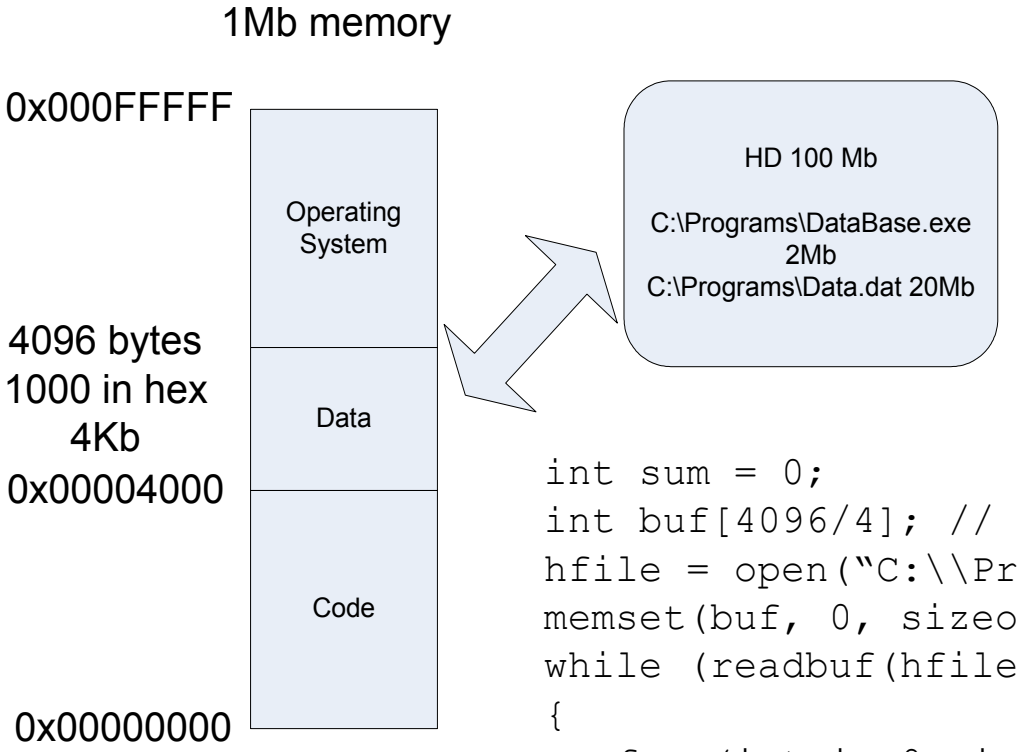

1Mb
0x000FFFFF
Physical Memory
0x00000000
HD 100 Mb
C:\Programs\DataBase.exe 2Mb
C:\Programs\Data.dat 20Mb

Processing Data (Non-buffered I/O)



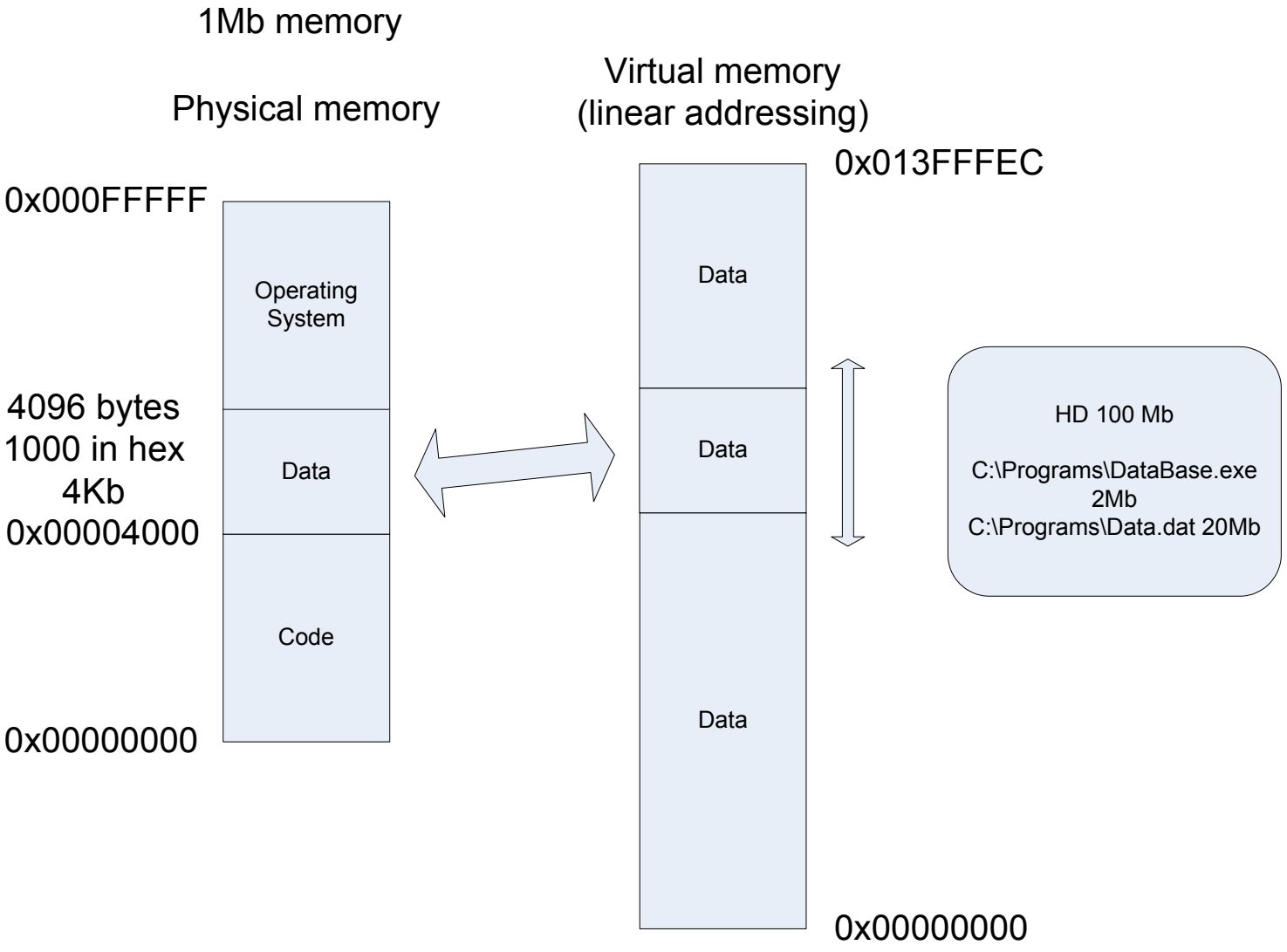

```
int val = 0, sum = 0;
hfile = open("C:\\Programs\\Data.dat");
while (read(hfile, &val))
{
    sum += val;
}
close(hfile);
```

Processing Data (Buffered I/O)

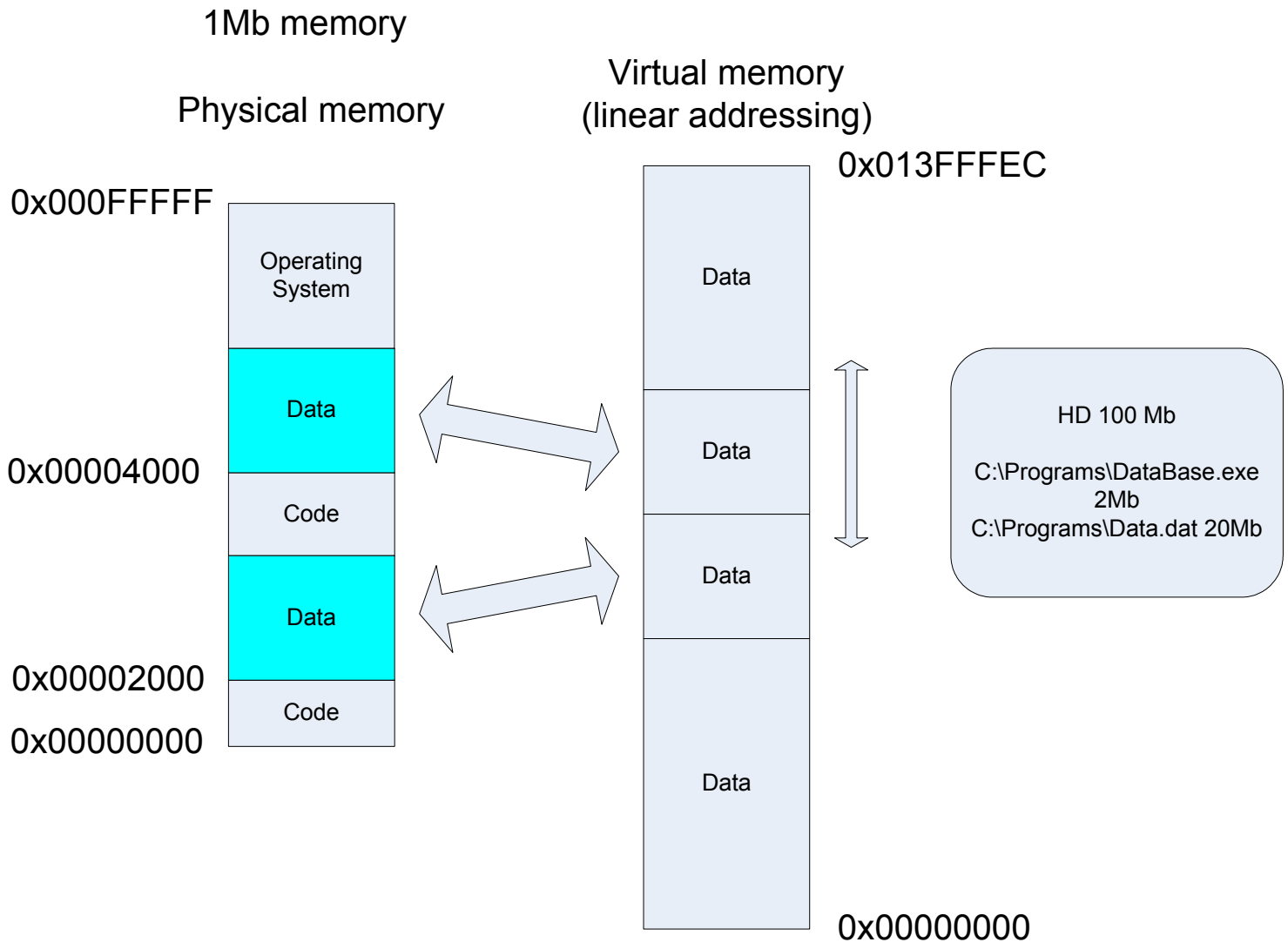



```
int sum = 0;
int buf[4096/4]; // 1024 elements
hfile = open("C:\\Programs\\Data.dat");
memset(buf, 0, sizeof(buf));
while (readbuf(hfile, buf, sizeof(buf)))
{
   for (int i =0; i < sizeof(buf)/sizeof(buf[0]); ++i)
   {
      sum += buf[i];
   }
   memset(buf, 0, sizeof(buf));
}
close(hfile);
```

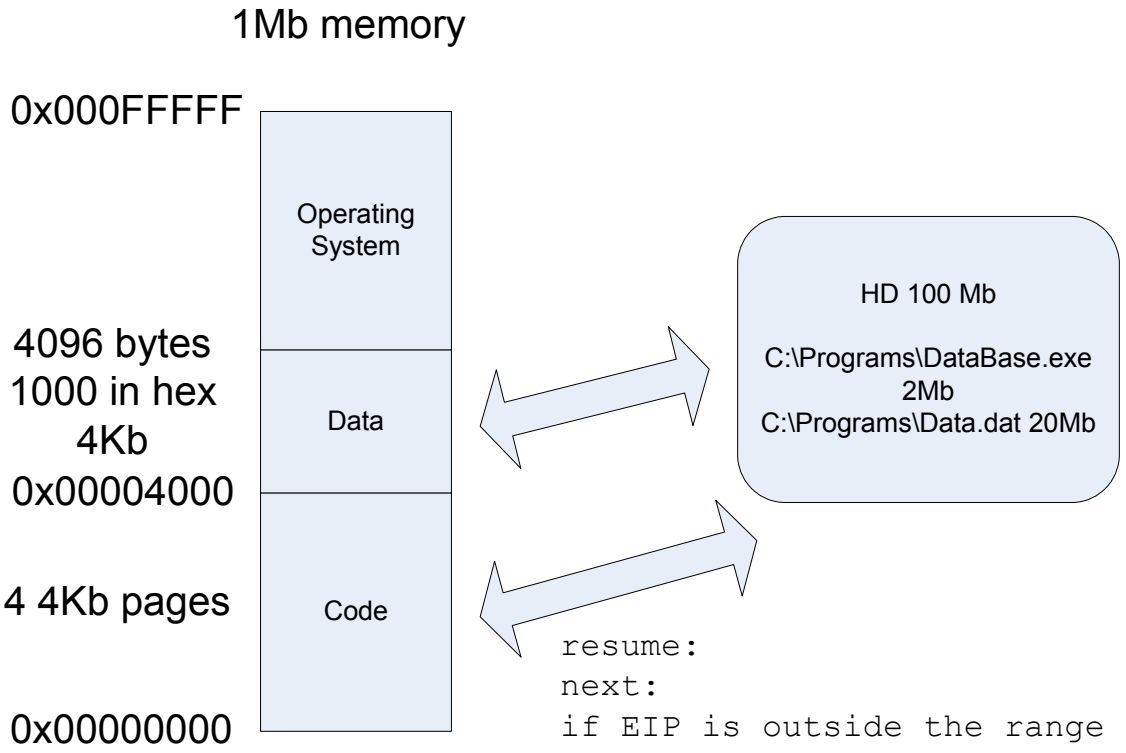
Virtual memory



Physical vs. Linear addresses

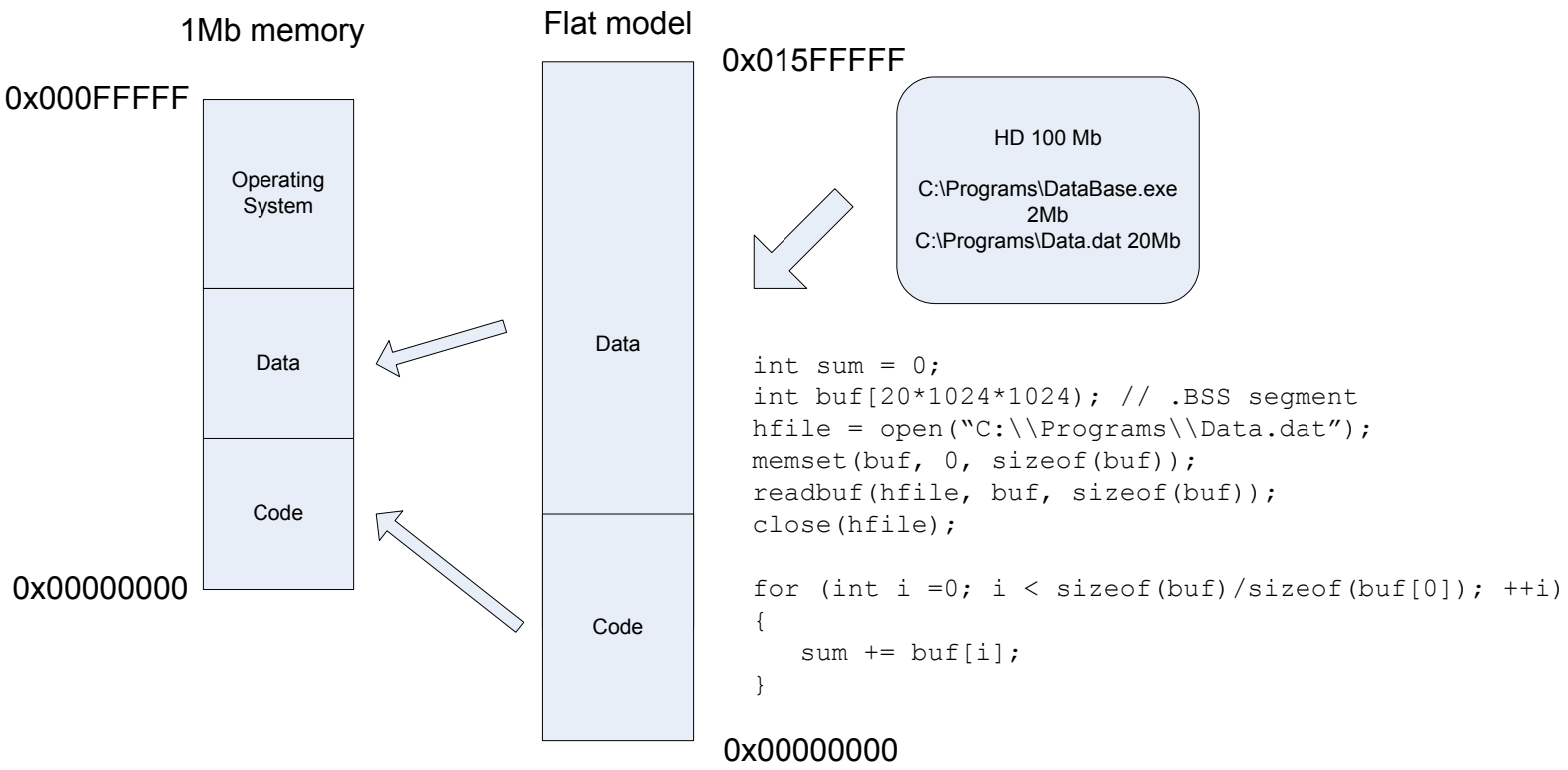


Executing Code (OS is involved)

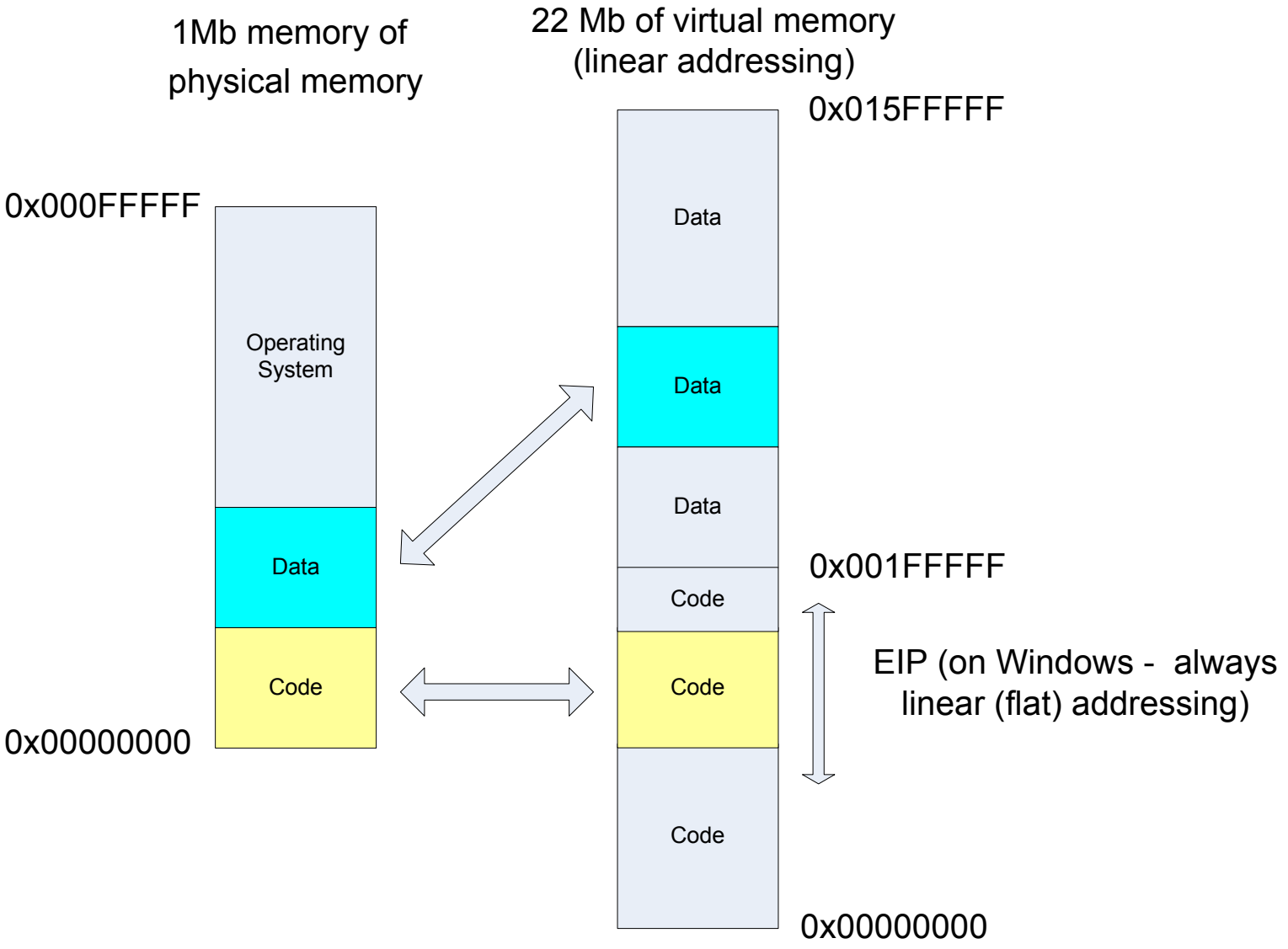



```
resume:
next:
if EIP is outside the range then goto interrupt;
fetch instruction;
advance EIP;
execute instruction;
goto next;
interrupt:
load next 4 pages from disk;
goto resume;
```

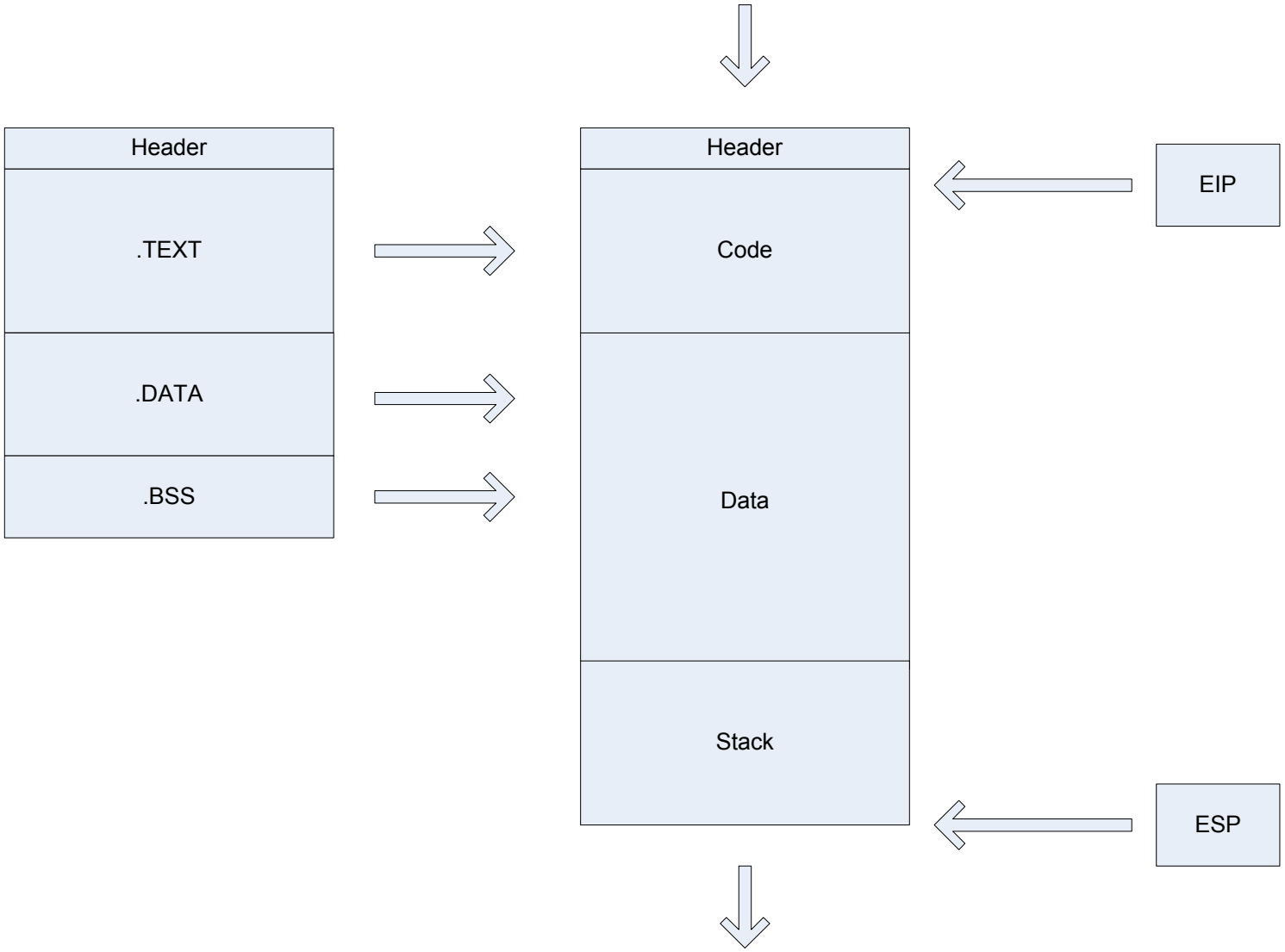
“Flat model” abstraction

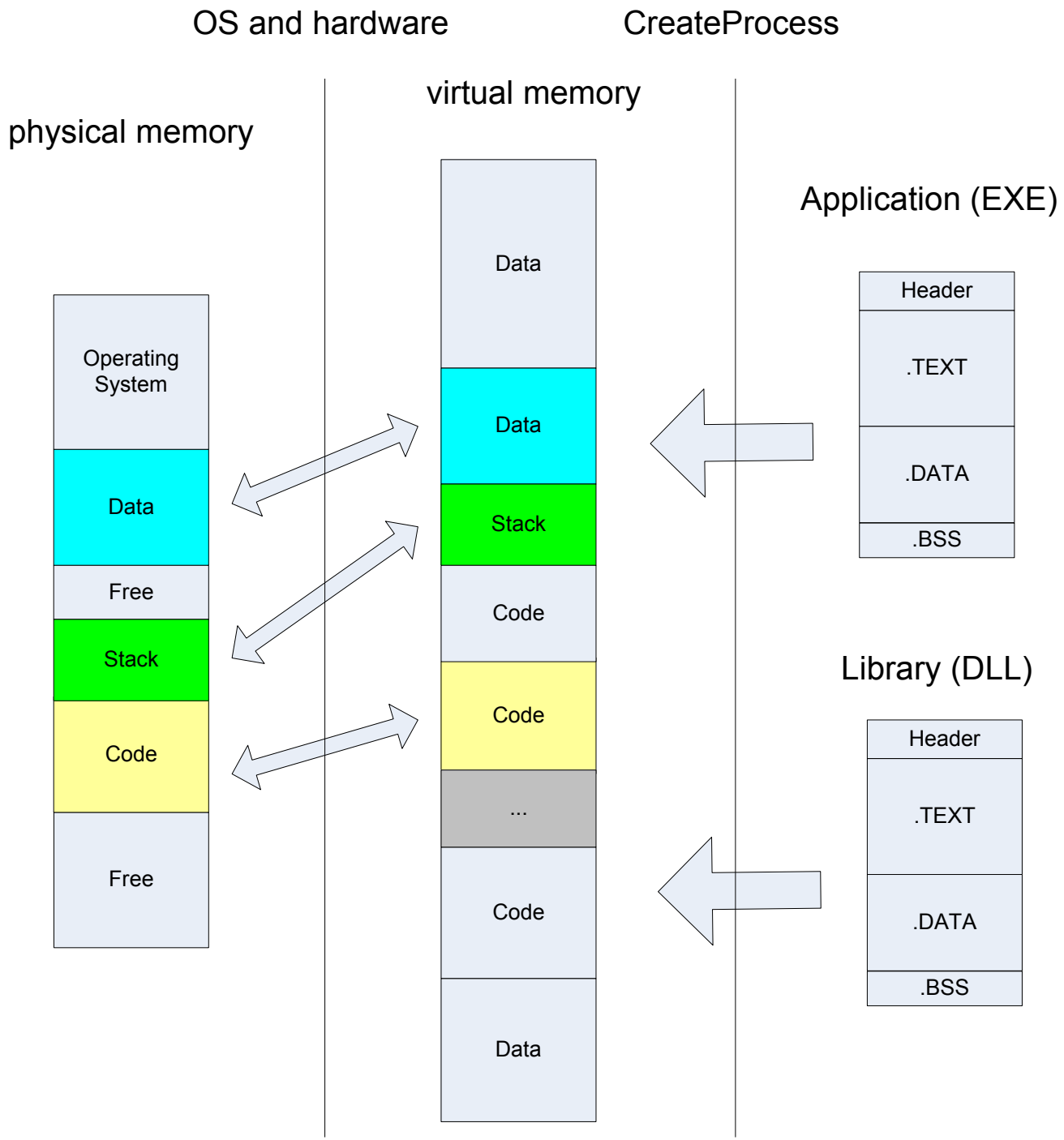


Now Operating System takes responsibility for managing both data and code



Application memory (simplified picture from memory and stacks presentation)



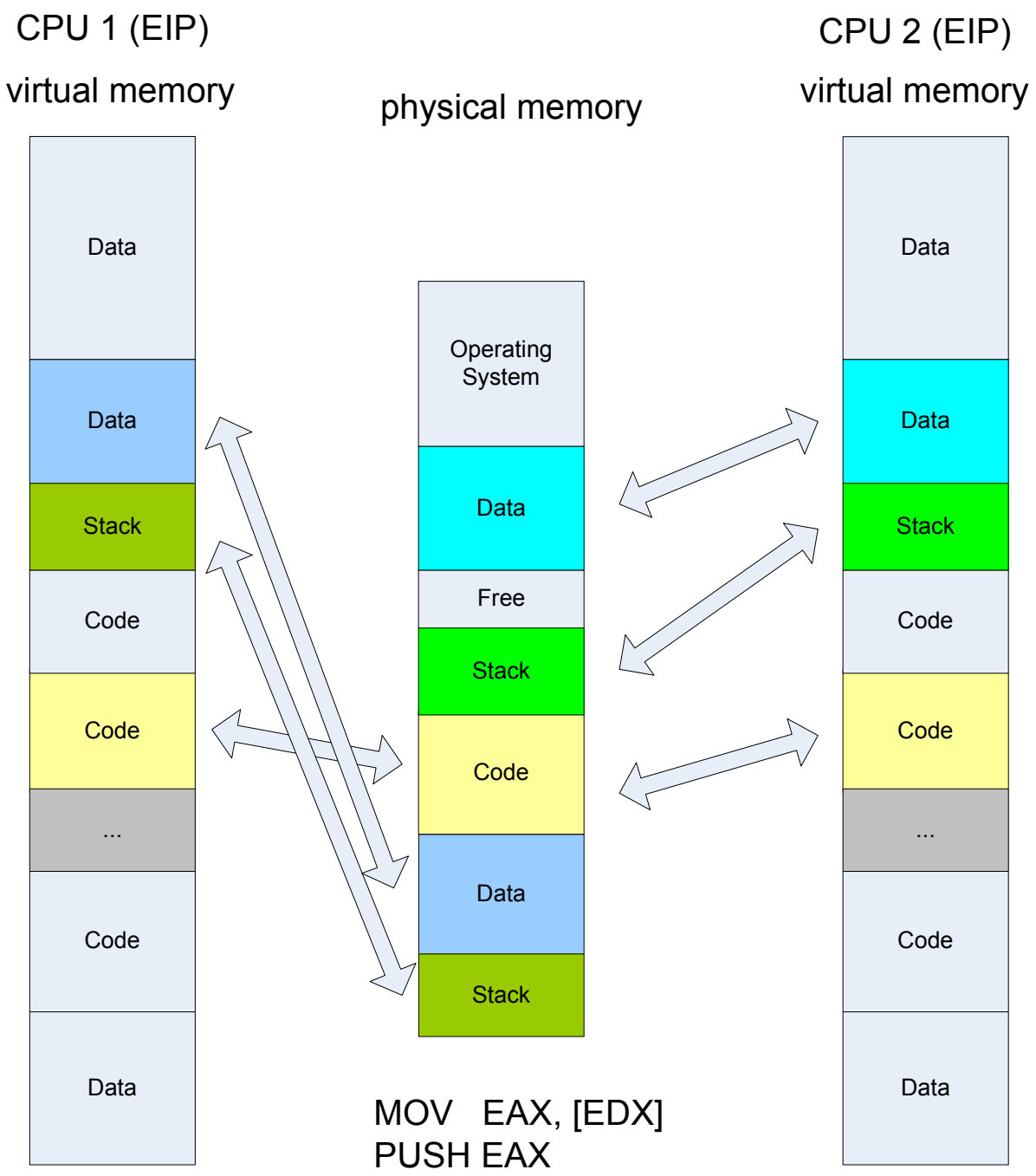

OS and hardware
CreateProcess
physical memory
virtual memory
Application (EXE)
Operating System
Data
Free
Stack
Code
Free
Data
Stack
Code
Code
...
Code
Data
Header
.TEXT
.DATA
.BSS
Library (DLL)
Header
.TEXT
.DATA
.BSS

Notes

- Virtual memory range is 4Gb (32-bit)
- Addresses in a user dump are linear addresses from virtual memory

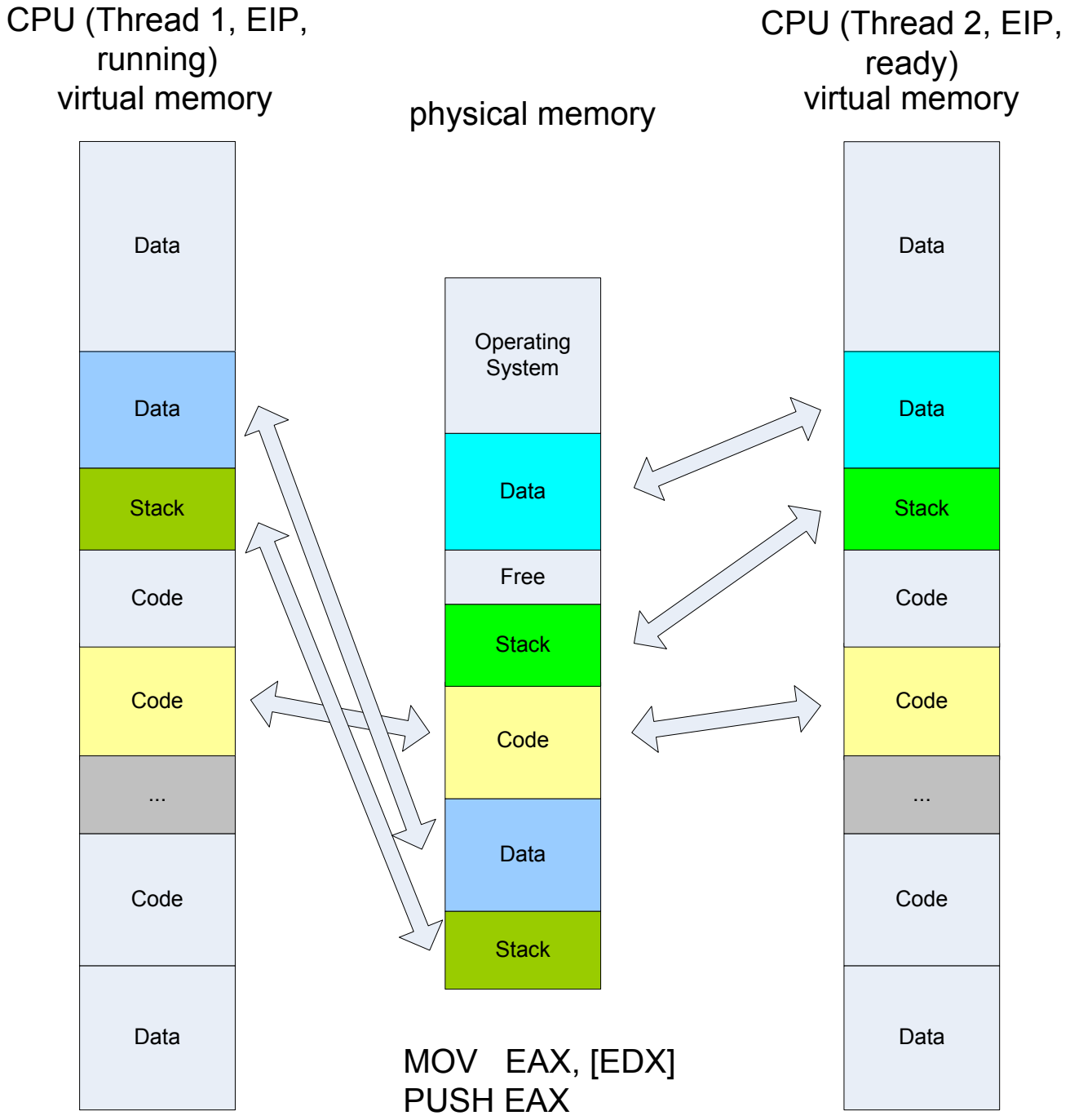
Process

- Operating system structure describing regions of virtual memory and other allocated resources (files, synchronization primitives, USER and GDI objects, etc.)
- Application and other components (DLLs, files) are mapped into linear virtual address space
- Owns resources


CPU 1 (EIP)
virtual memory
physical memory
CPU 2 (EIP)
virtual memory
Data
Data
Stack
Code
Code
...
Code
Data
Operating System
Data
Free
Stack
Code
Data
Stack
Data
Data
Stack
Code
Code
...
Code
Data
MOV EAX, [EDX]
PUSH EAX

Why do we need threads?

- To improve application and system performance by creating pseudo-parallelism
- Application performance: internet browser - if we click on a link a thread is created to download a new page, but we can still browse the current page
- System performance: parallel access to hardware inside OS


CPU (Thread 1, EIP, running)
virtual memory
physical memory
CPU (Thread 2, EIP, ready)
virtual memory
Data
Data
Stack
Code
Code
...
Code
Data
Operating System
Data
Free
Stack
Code
Data
Stack
Data
Data
Stack
Code
Code
...
Code
Data
MOV EAX, [EDX]
PUSH EAX

Thread

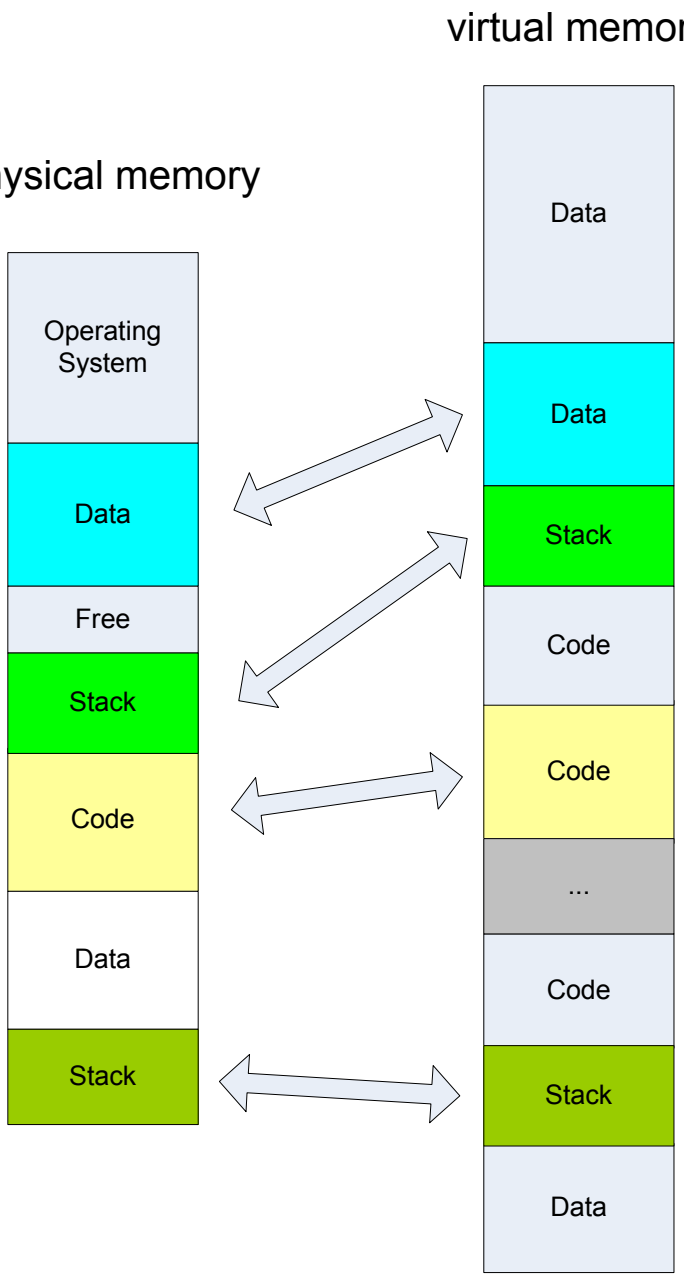
- Unit of execution (current EIP)
- At least one thread in a process
- All resources (owned by a process) are shared among all process threads – synchronization issues

Thread 1 (EIP)

```
MOV   EAX, [EDX]
PUSH EAX
```

Thread 2 (EIP)

```
MOV  ESP, EBP
POP   EBP
RET
```



Pictures vs. Words

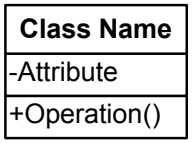
- Better than a thousand words
- Bad example: Flow of CreateProcess (pp. 304-317 Inside Windows 2000)
- Diagramming notations

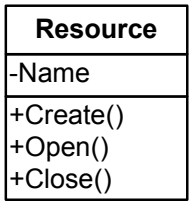
UML (Unified Modeling Language)

- Standard diagramming notation used to describe and communicate application structure and behavior, software architecture and designs (http://www.uml.org/)
- Diagrams for modeling static structure are similar to ER (entity-relationship) and EER (enhanced ER)
- We will use UML diagrams to depict OS structures and interactions

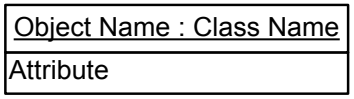
Classes and objects

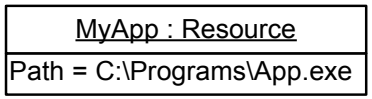
- Class (entity type in ER)





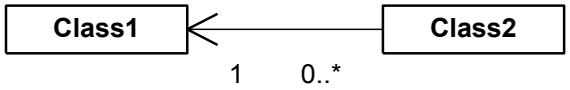
- Object (entity occurrence, class instance)





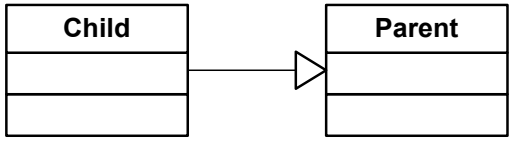
Relationships

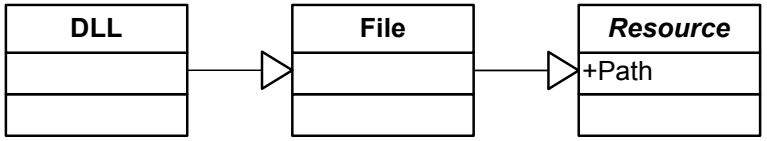
- Binary association



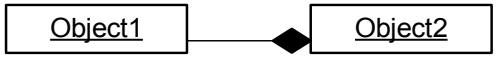
FK, Not NULL constraint

- Generalization (parent – child, class - subclass)

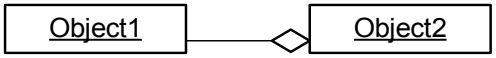




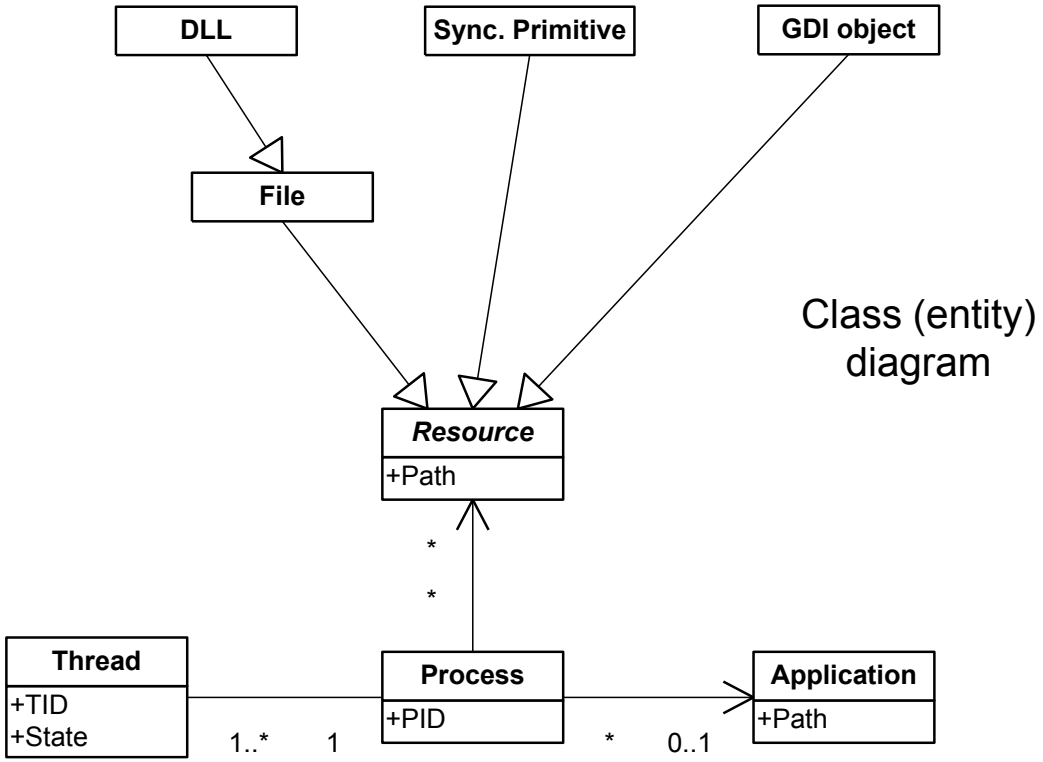
- Containment (full ownership)



- Aggregation (shared ownership)

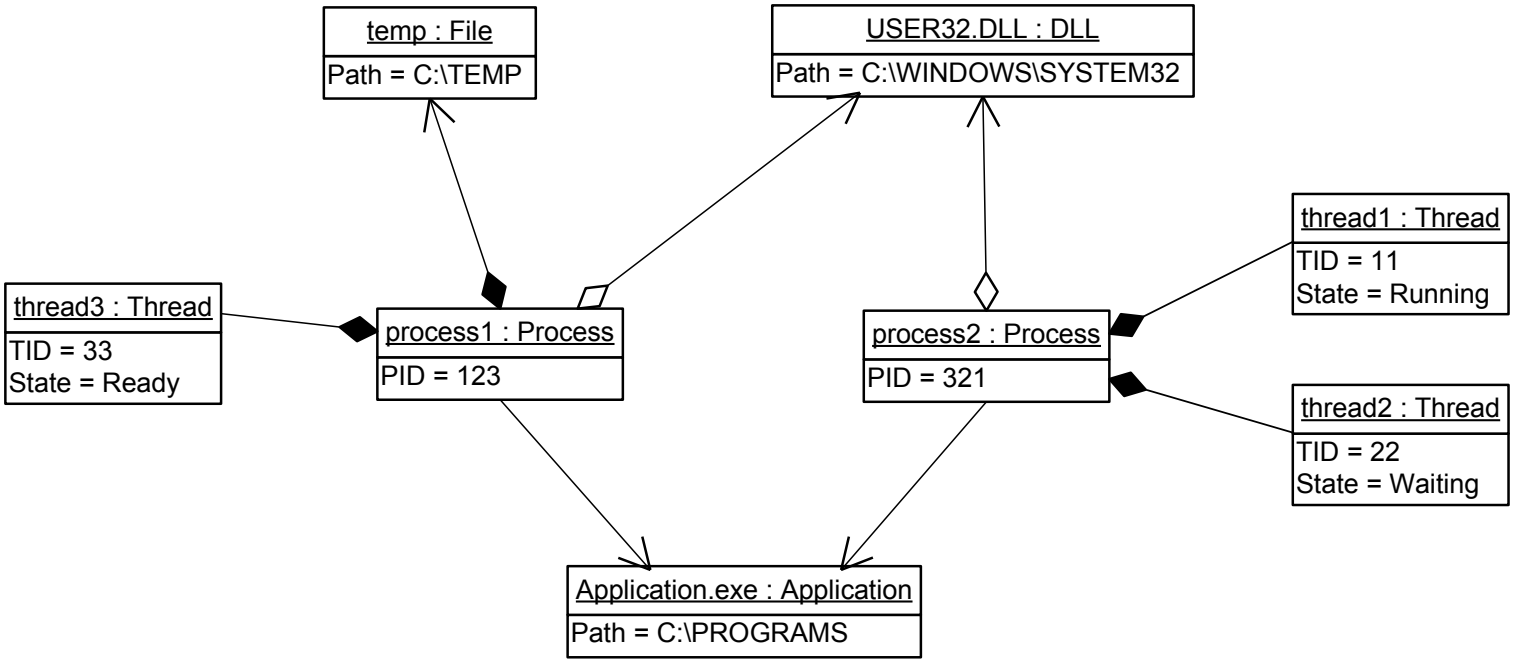


Simplified class UML diagram



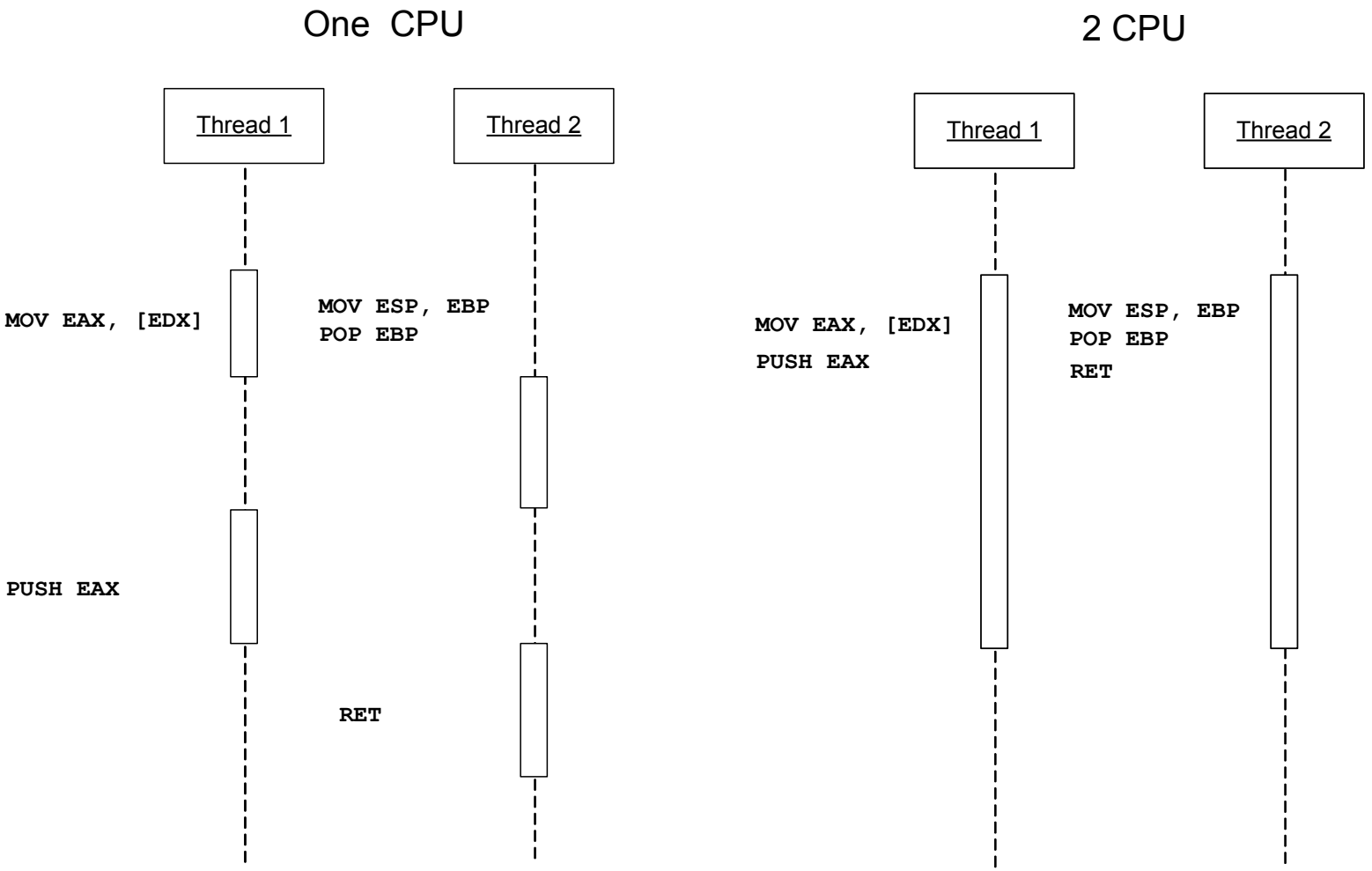
Class (entity) diagram

Relationship example



Object (instance) diagram

UML sequence diagram (2 threads)



What's next?

- Virtual memory and processes
- Multithreading, memory and stacks
- Calling Windows functions
  (stdcall vs. cdecl)
- Strings
- Pointers to pointers (LPSTR *)
- Structures in memory