Practical Foundations of Debugging

Chapter 9

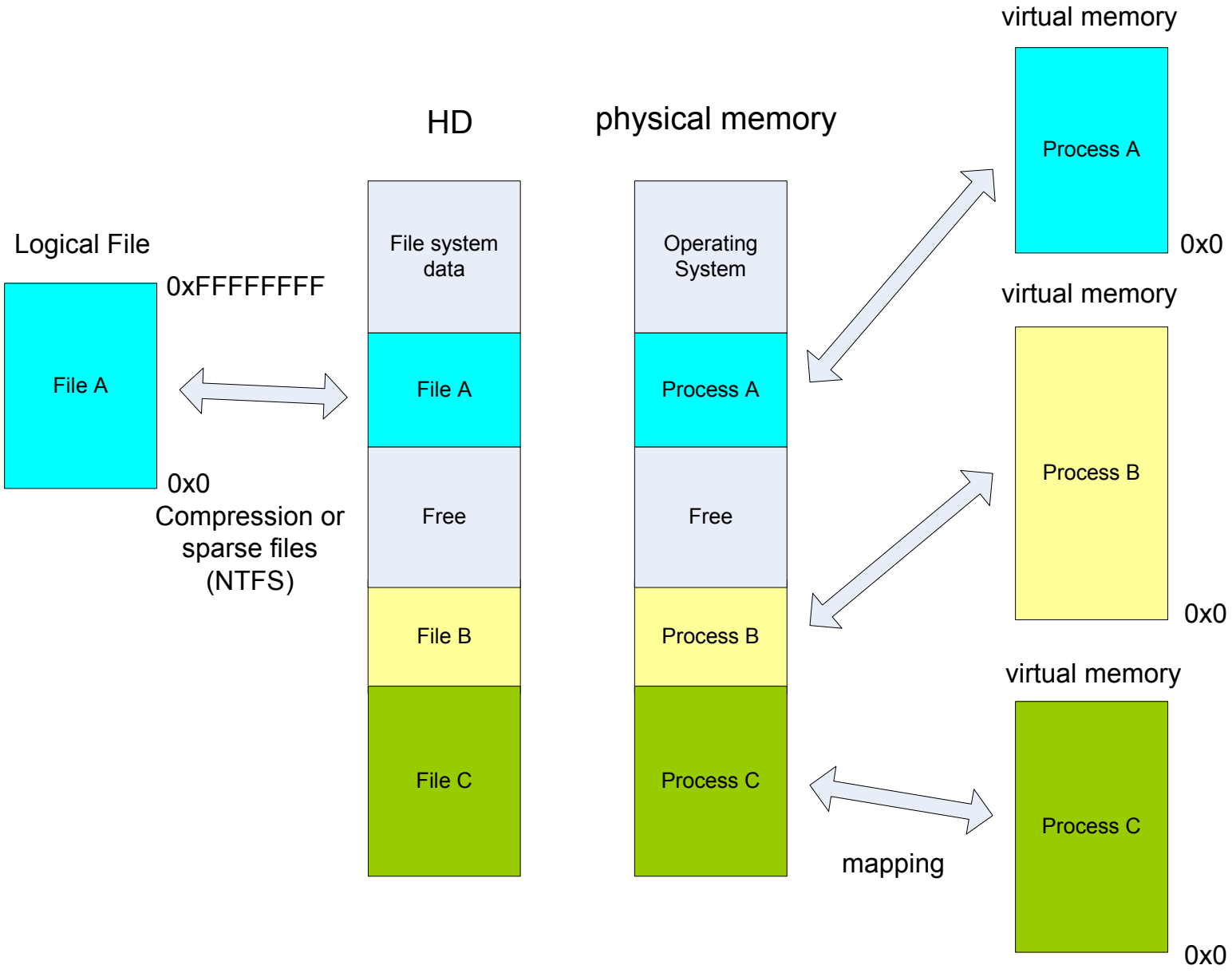
Virtual Memory, Processes and Threads – Part 2

Designed, written and illustrated by Dmitry Vostokov

File System analogy (Picture 1)

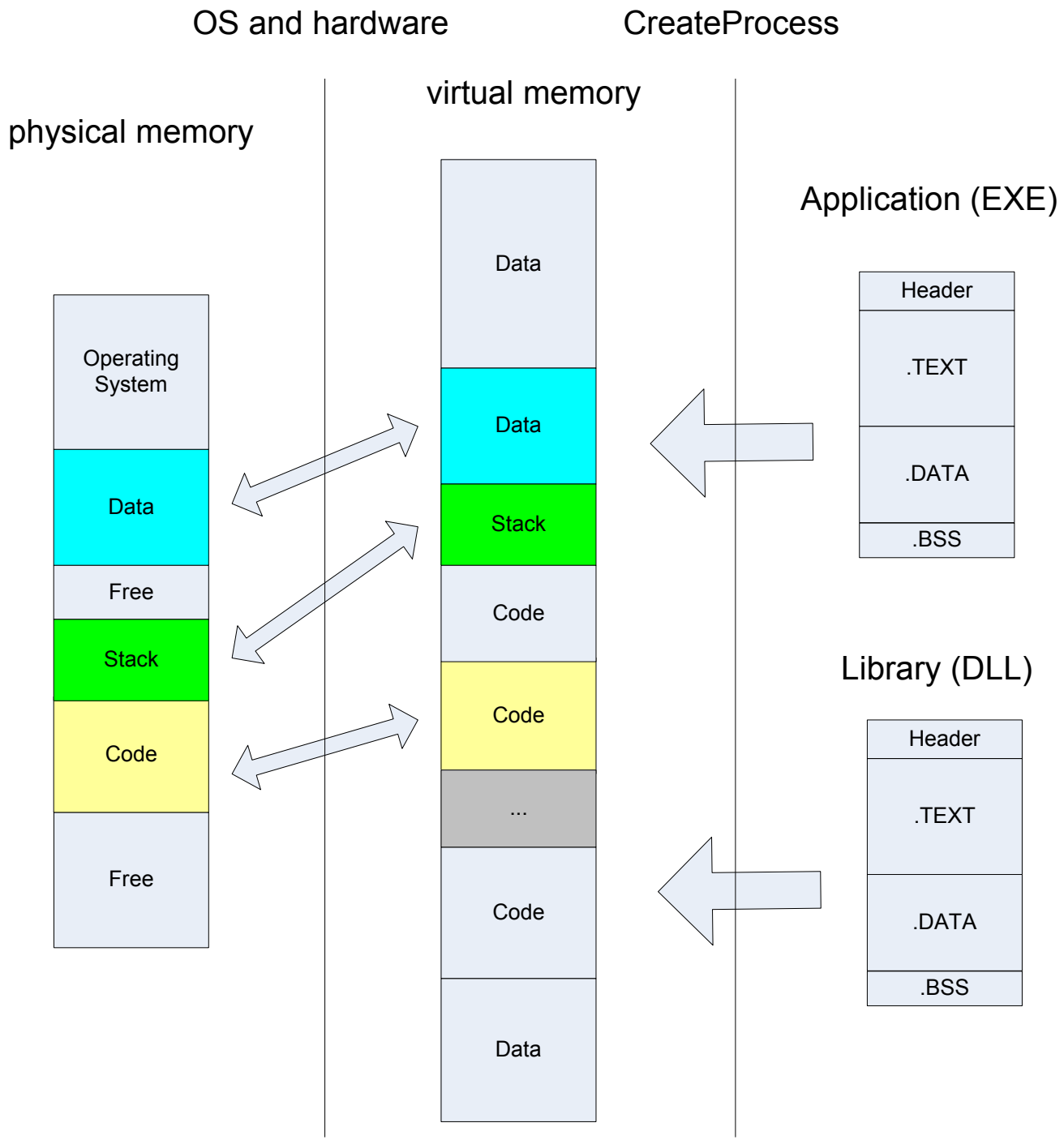
- Hard disk – Physical memory
- Individual files – Processes
- File system data – OS support for VM
- File compression – VM mapping

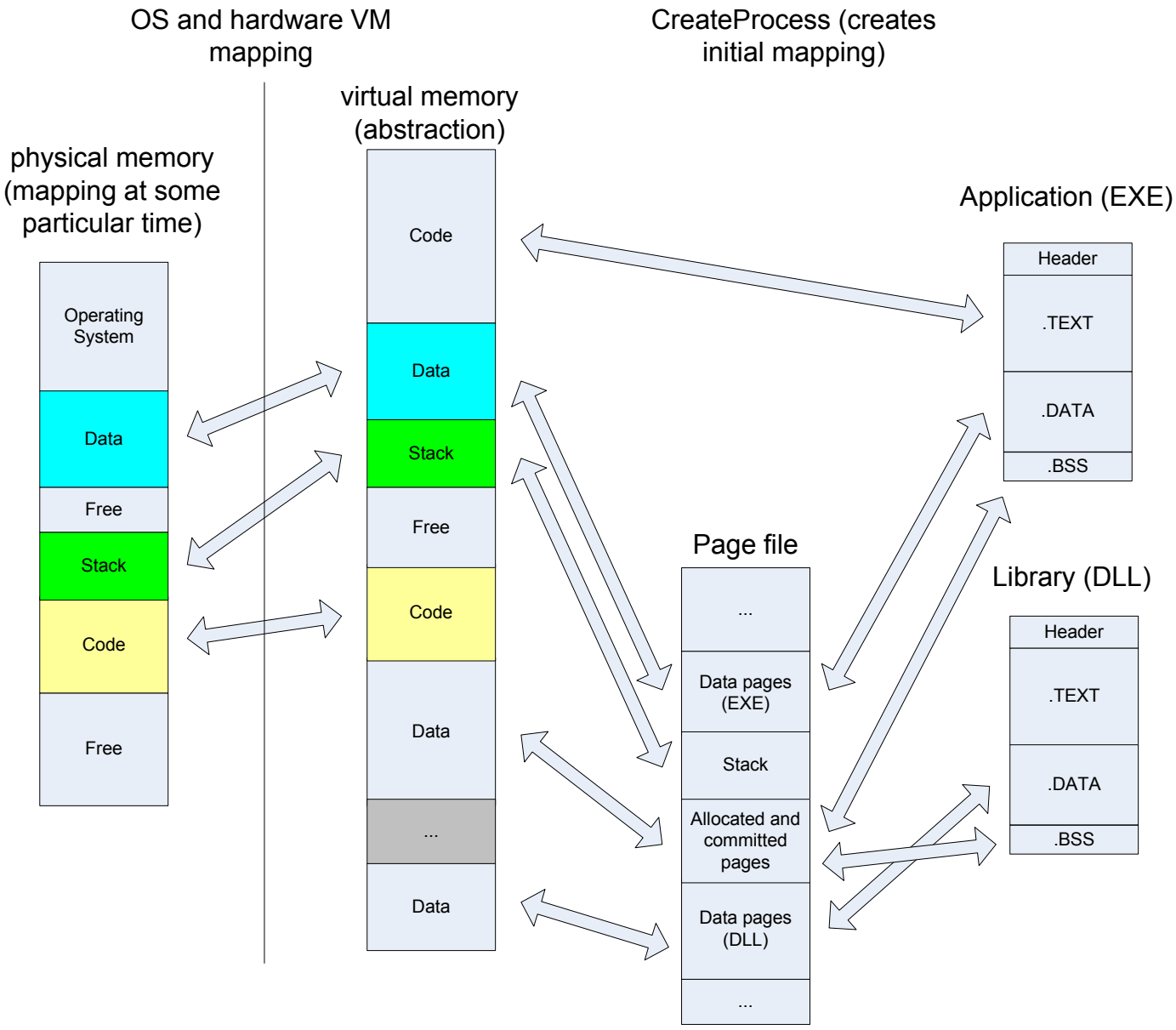
Picture 1


virtual memory
HD
physical memory
Process A
Logical File
File system data
Operating System
0xFFFFFFFF
File A
File A
Process A
0x0
virtual memory
Free
Free
Process B
0x0
Compression or sparse files (NTFS)
File B
Process B
0x0
virtual memory
File C
Process C
Process C
mapping
0x0

Where all is stored?

- Code pages are mapped from application and DLL .TEXT sections stored on secondary storage (can be read-only). This is so called code sharing.
- Initialized data pages are copied and stored in page files. Therefore mapping is between page files and physical memory.
- Uninitialized data and stack pages are reserved in page files. Mapping is between page files and physical memory.
- All additionally allocated memory pages (dynamic memory, heap, etc) must be reserved in page files (**committed**). Mapping is between page files and physical memory.


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


OS and hardware VM mapping
CreateProcess (creates initial mapping)
physical memory (mapping at some particular time)
virtual memory (abstraction)
Application (EXE)
Operating System
Data
Free
Stack
Code
Free
Code
Data
Stack
Free
Code
Data
...
Data
Header
.TEXT
.DATA
.BSS
Page file
...
Data pages (EXE)
Stack
Allocated and committed pages
Data pages (DLL)
...
Library (DLL)
Header
.TEXT
.DATA
.BSS

Committed memory

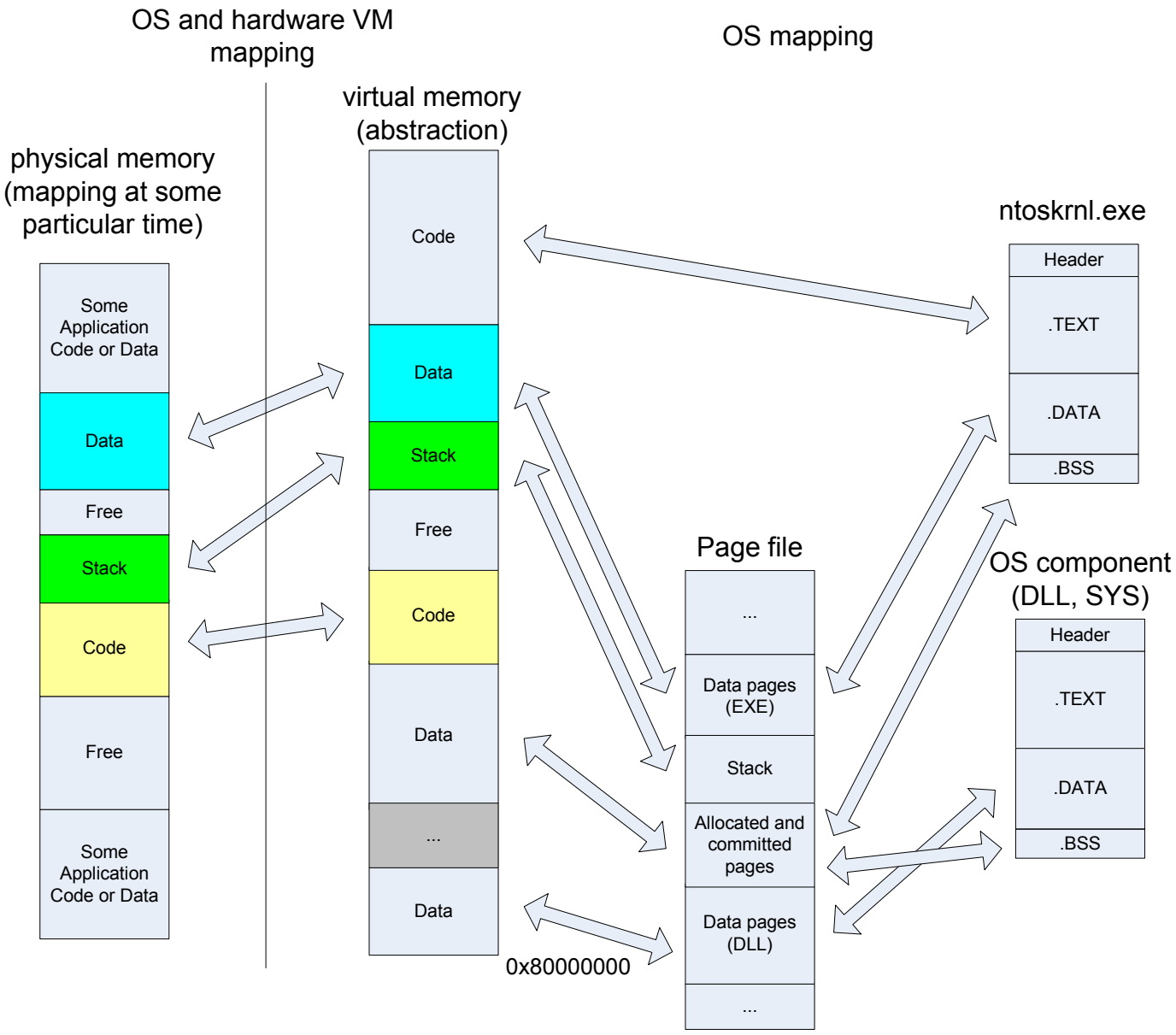
- Total size of virtual memory per process is 4Gb (32-bit registry size limitation)
- The size of virtual memory available for application is 2Gb (3Gb if /3GB switch is present in boot.ini file). Virtual addresses start from 0.
- To reserve an additional region of virtual memory a process calls VirtualAlloc Win32 API function. Then it calls VirtualAlloc again to commit it (to allocate page file space)

Example:

```
int matrix[40000][40000];  // very sparse, 1.6Gb
```

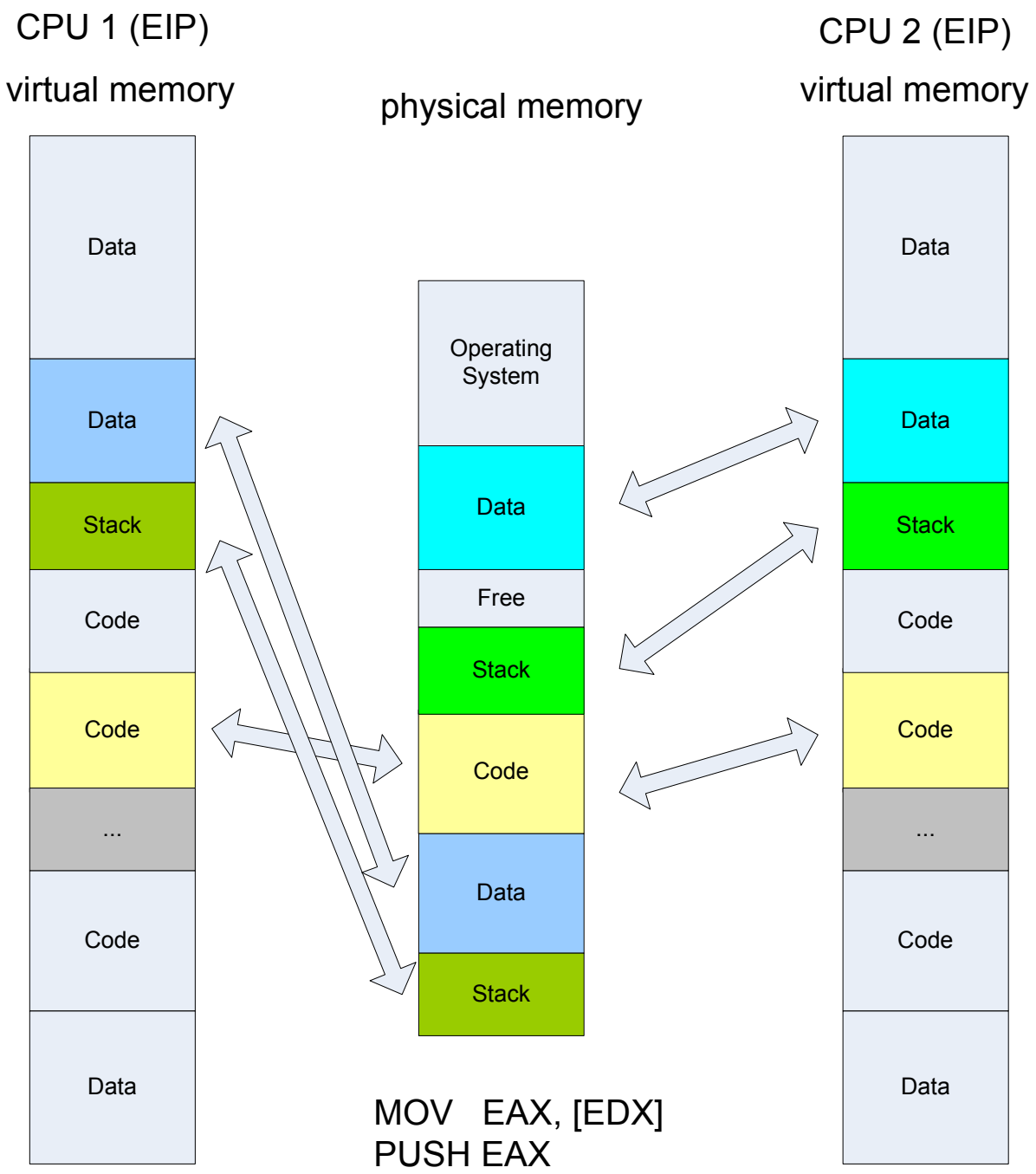
How to fit OS into physical memory?

- Consider OS as a one big application having it’s own virtual address space starting at 0x80000000 (2Gb).
- All storage rules for processes (page file, code sharing, etc.) are applied here, except dynamic memory is not allocated using Win32 API calls
- OS Drivers are just some kind of DLLs preloaded or loaded on demand (PNP).


OS and hardware VM mapping
OS mapping
physical memory (mapping at some particular time)
virtual memory (abstraction)
ntoskrnl.exe
Some Application Code or Data
Data
Free
Stack
Code
Free
Some Application Code or Data
Code
Data
Stack
Free
Code
Data
...
Data
0x80000000
Header
.TEXT
.DATA
.BSS
Page file
...
Data pages (EXE)
Stack
Allocated and committed pages
Data pages (DLL)
...
OS component (DLL, SYS)
Header
.TEXT
.DATA
.BSS

Virtual to physical address mapping

- EIP points to virtual address
- OS + hardware provide mapping mechanism
- Switching to another process requires new mapping because virtual addresses are independent (the same virtual address must point to different physical address - process isolation)


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

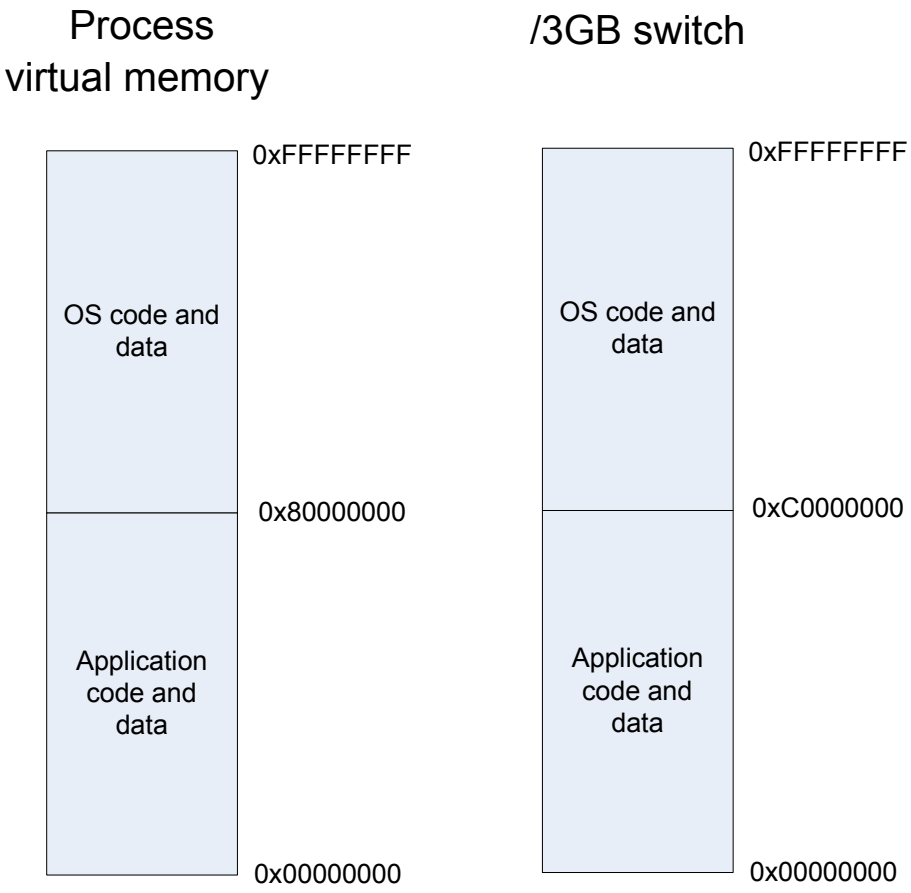
OS architectures

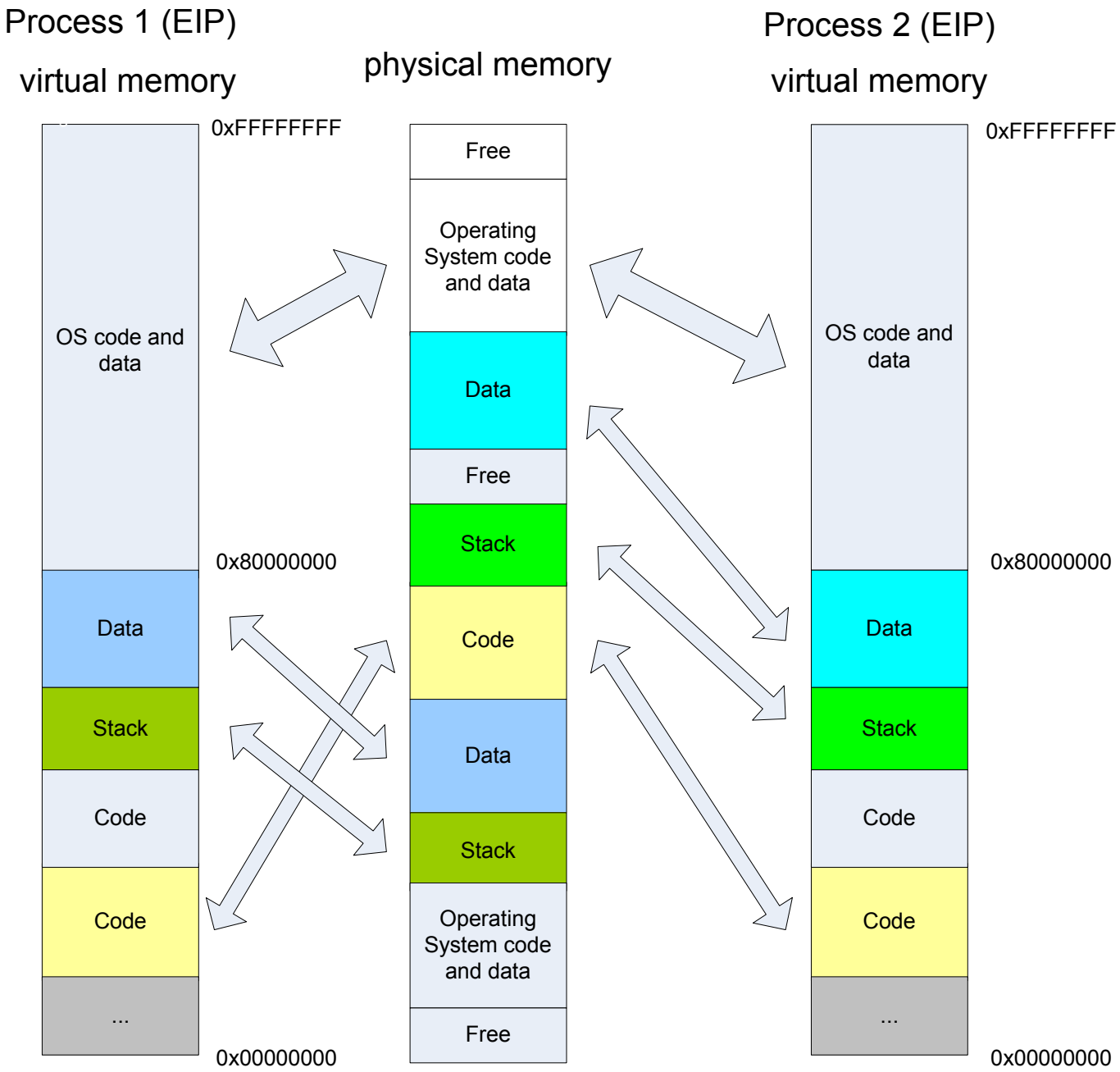
- Although we speak about so called Wintel platform the support Intel processor provides for OS is independent from MSFT NT implementation
- There are other OS where process mappings and calling OS services are implemented differently on Intel processors (Linux, FreeBSD, Solaris port and many other less known OS)

Accessing OS code

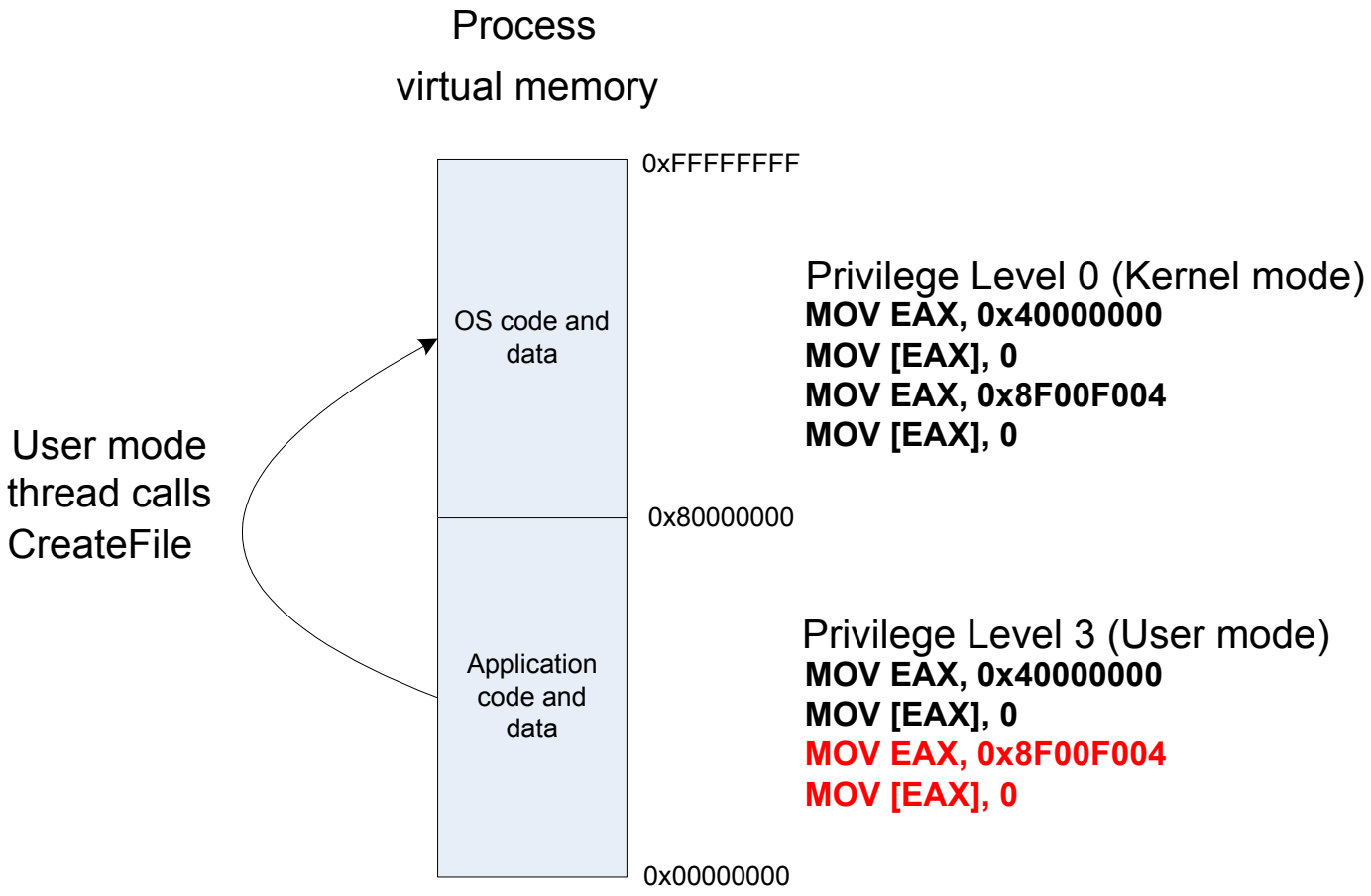
- OS code has its own virtual address range
- If OS has the same virtual address range (0 – 0xFFFFFFFF) than every system call has to trigger the change in virtual to physical mapping (very expensive procedure)
- MSFT has decided to allocate upper 2Gb of virtual address space to OS code (0x80000000 – 0xFFFFFFFF) – default configuration
- To support application that require more virtual addresses (for example, DB applications) you can specify /3GB switch in boot.ini
- Now every OS call does not require change in mapping because mapping for virtual addreses 0x80000000 – 0xFFFFFFFF is the same for every process

Kernel and User space

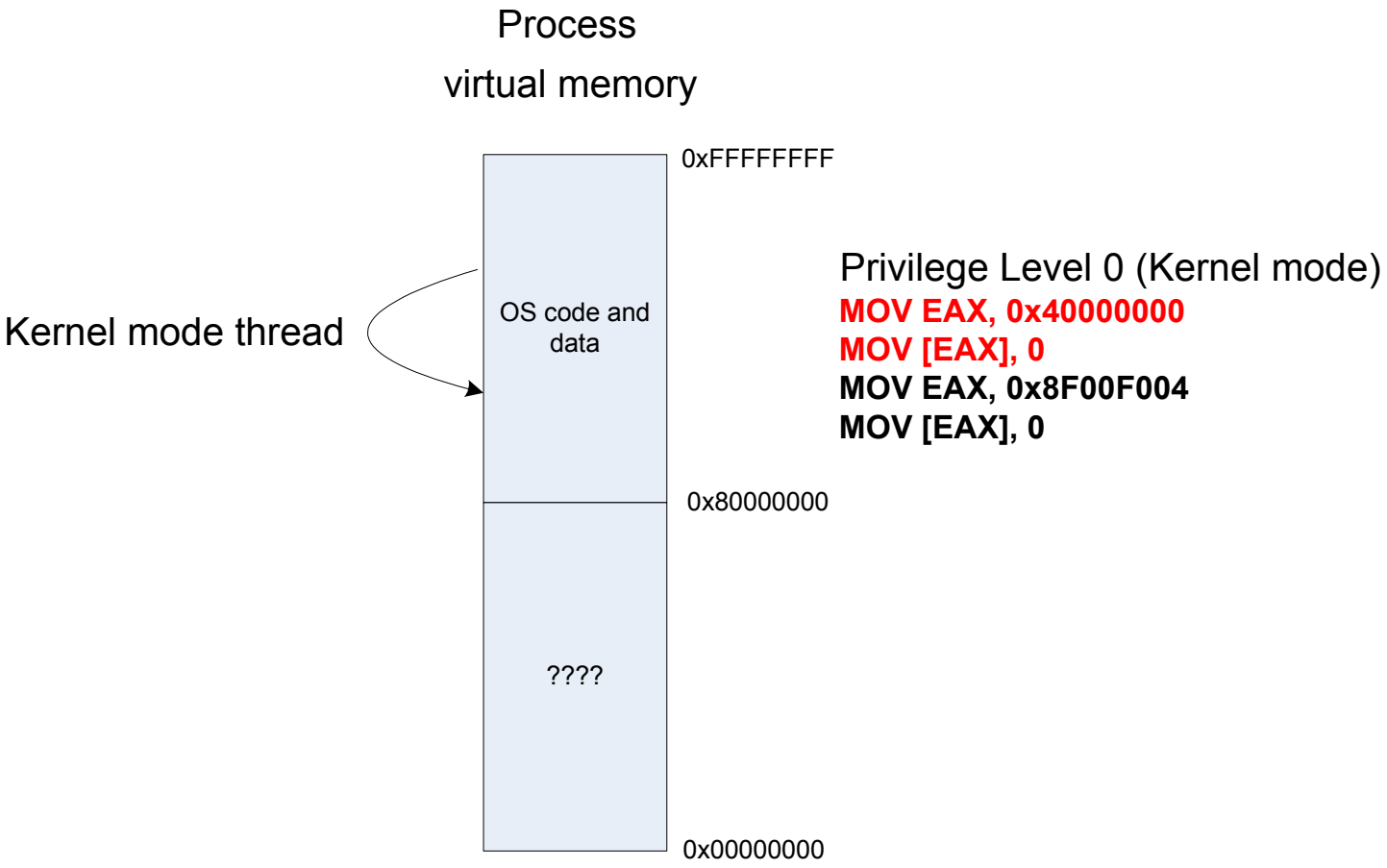



Process 1 (EIP)
virtual memory
physical memory
Process 2 (EIP)
virtual memory
0xFFFFFFFF
OS code and data
Data
Stack
Code
...
0x80000000
0x00000000
Free
Operating System code and data
Data
Free
Stack
Code
Data
Stack
Operating System code and data
Free
0xFFFFFFFF
OS code and data
Data
Stack
Code
...
0x80000000
0x00000000

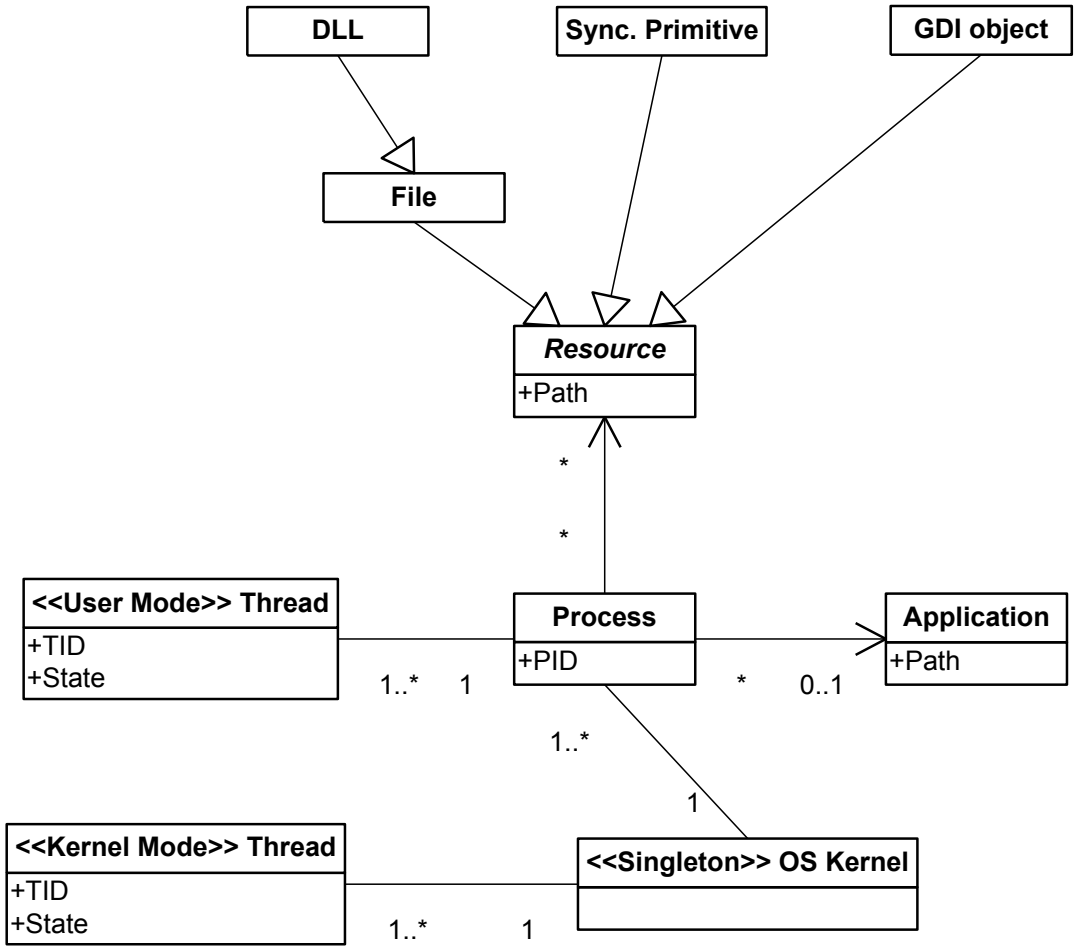
Privilege levels and user-mode threads



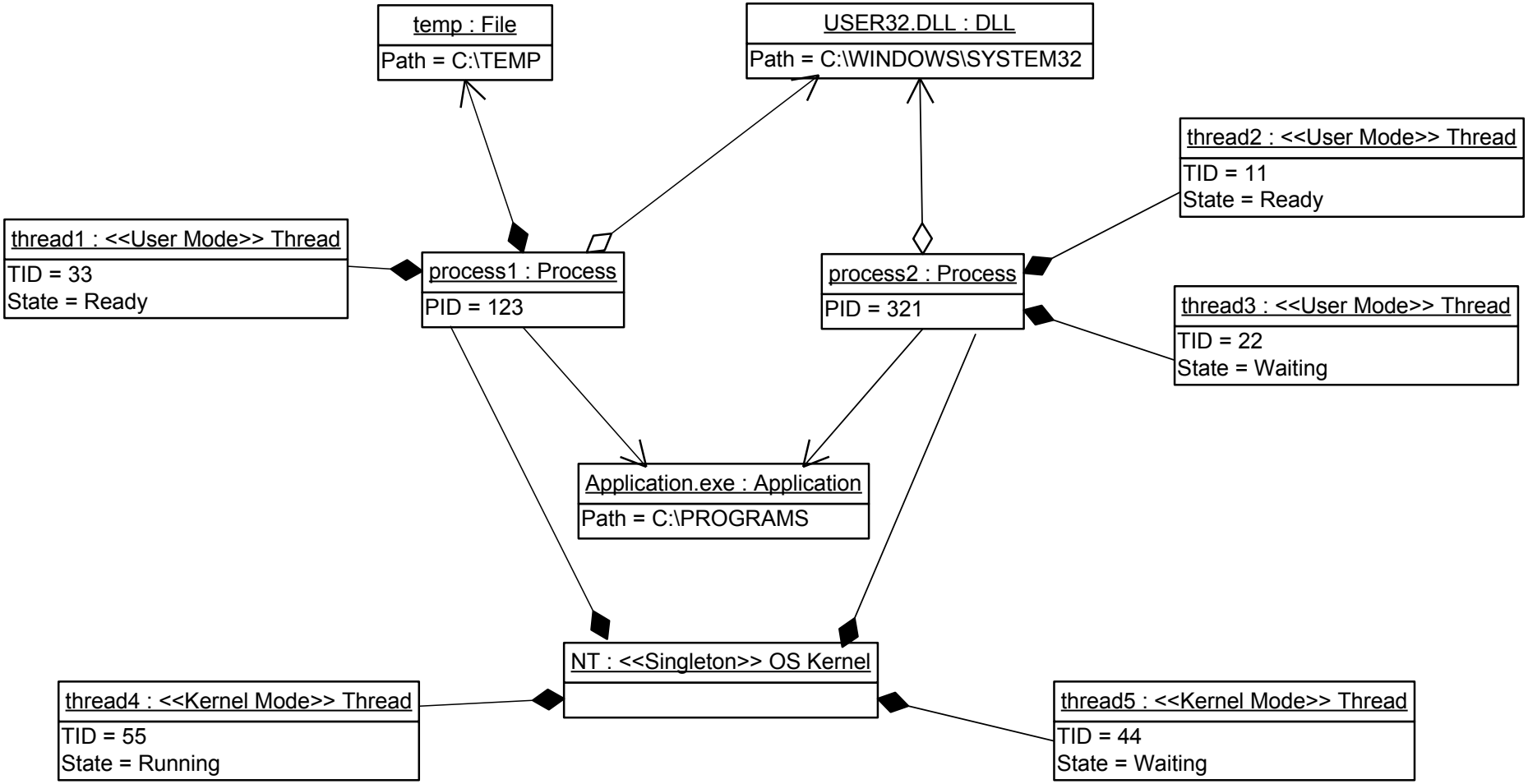
Kernel mode threads



Class diagram refinement



Object diagram



What's next?

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