Operating Systems

Lecture 9: Limited Direct Execution + Memory Virtualisation

Nipun Batra

Aug 23, 2018

Administrative

1. Next Wednesday answer sheets in lab session
2. Projects - list would be available on Monday
    1. Project 5 -> 8% (3% reduced from homework)
    2. More details on Tuesday…

Space v/s Time Multiplexing

Time multiplexing : Share resource by dividing over time

Space v/s Time Multiplexing

Time multiplexing : Share resource by dividing over time

1. CPU scheduling on single core

Space v/s Time Multiplexing

Time multiplexing : Share resource by dividing over time

1. CPU scheduling on single core
2. Think more?!

Space v/s Time Multiplexing

Time multiplexing : Share resource by dividing over time

1. CPU scheduling on single core
2. Think more?!
3. Class room scheduling - single class runs in at any given point of time

Space v/s Time Multiplexing

Time multiplexing : Share resource by dividing over time

1. CPU scheduling on single core
2. Think more?!
3. Class room scheduling - single class runs in at any given point of time
4. TDMA??

Space v/s Time Multiplexing

Space multiplexing : Share resource by dividing into smaller pieces

Space v/s Time Multiplexing

Space multiplexing : Share resource by dividing into smaller pieces

1. CPU scheduling on multiple cores?

Space v/s Time Multiplexing

Space multiplexing : Share resource by dividing into smaller pieces

1. CPU scheduling on multiple cores?
2. Cake sharing

Space v/s Time Multiplexing

Space multiplexing : Share resource by dividing into smaller pieces

1. CPU scheduling on multiple cores?
2. Cake sharing
3. Think more?

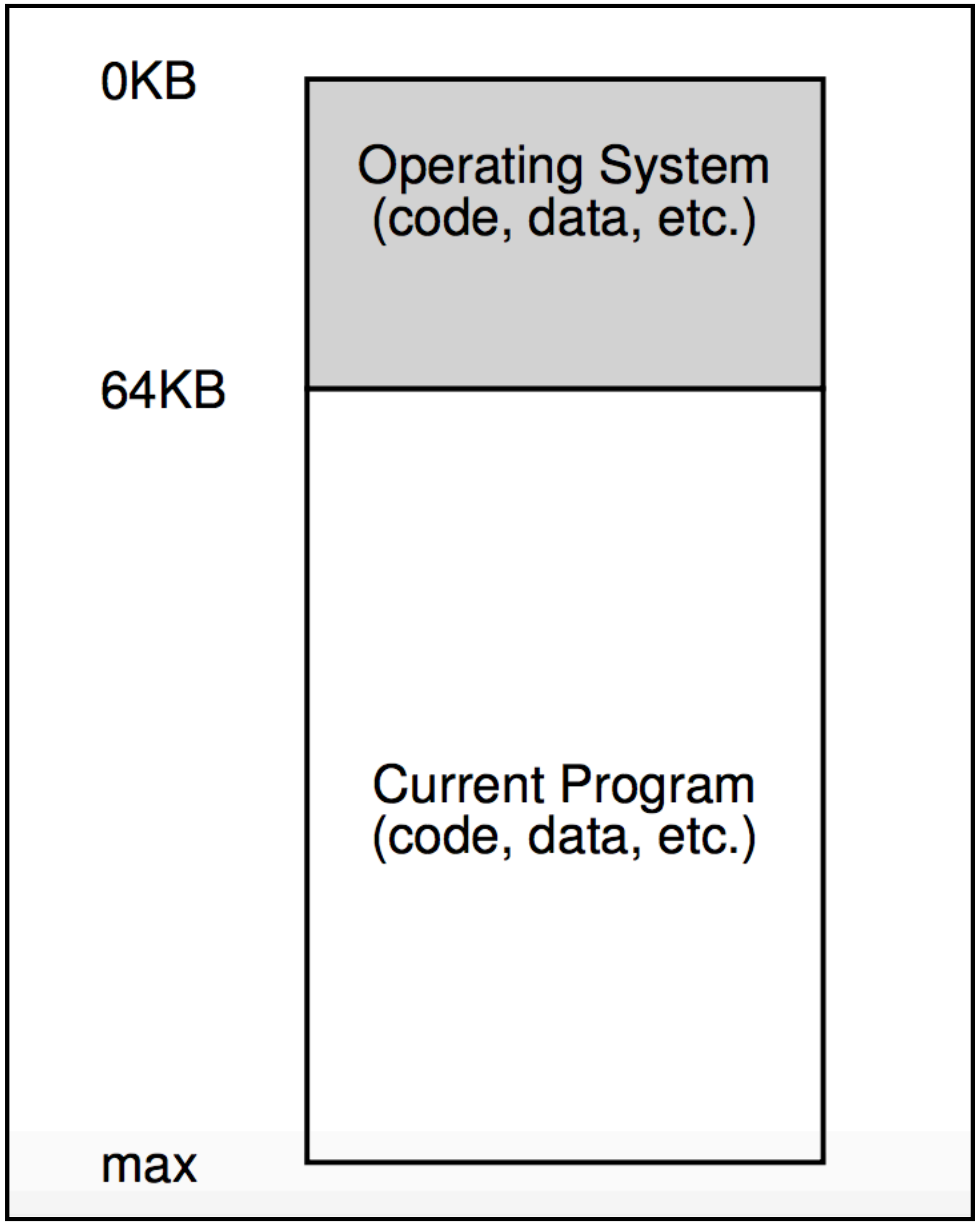
Space v/s Time Multiplexing

Space multiplexing : Share resource by dividing into smaller pieces

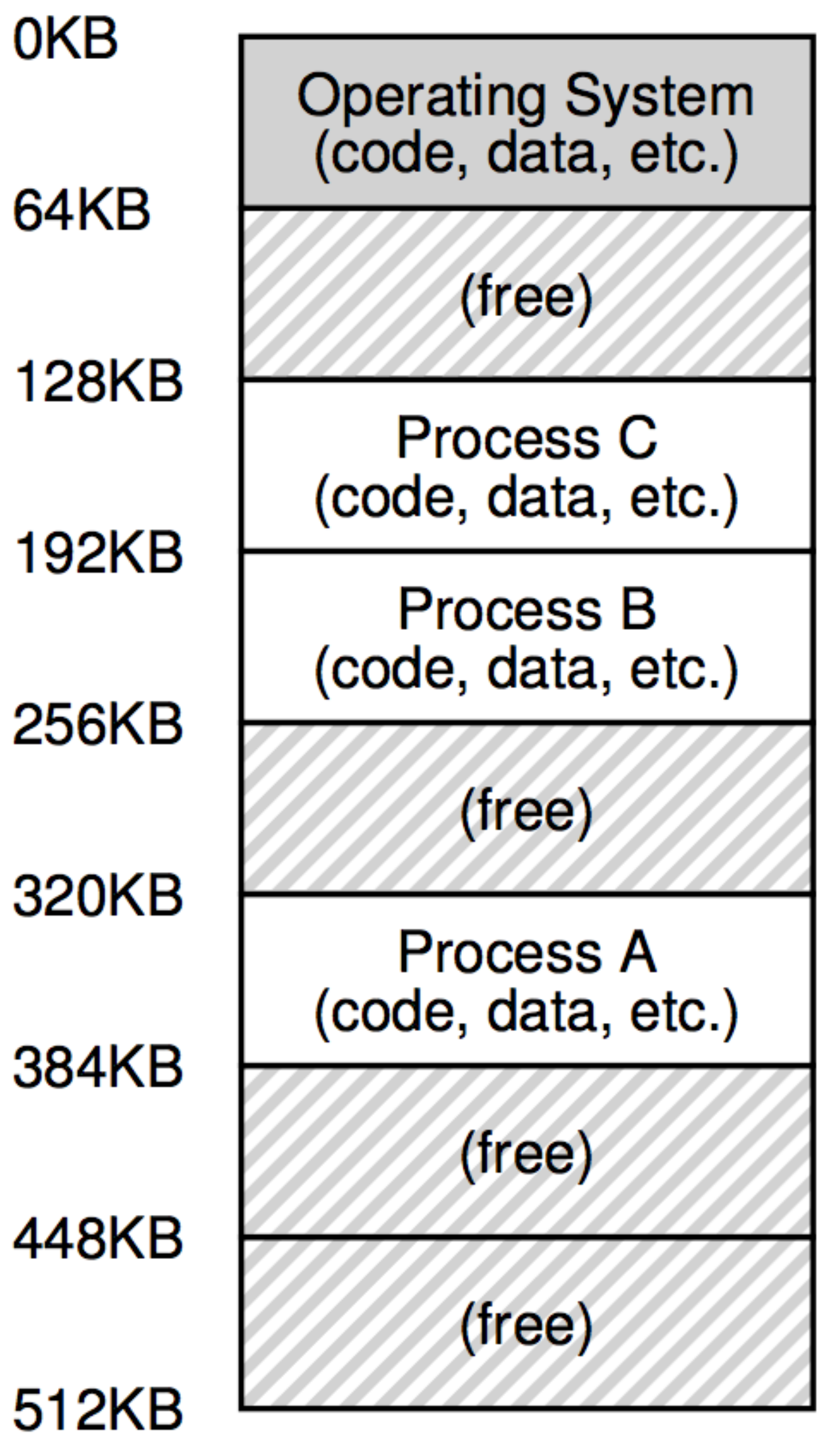
1. CPU scheduling on multiple cores?
2. Cake sharing
3. Think more?
4. Memory management

Memory Virtualisation

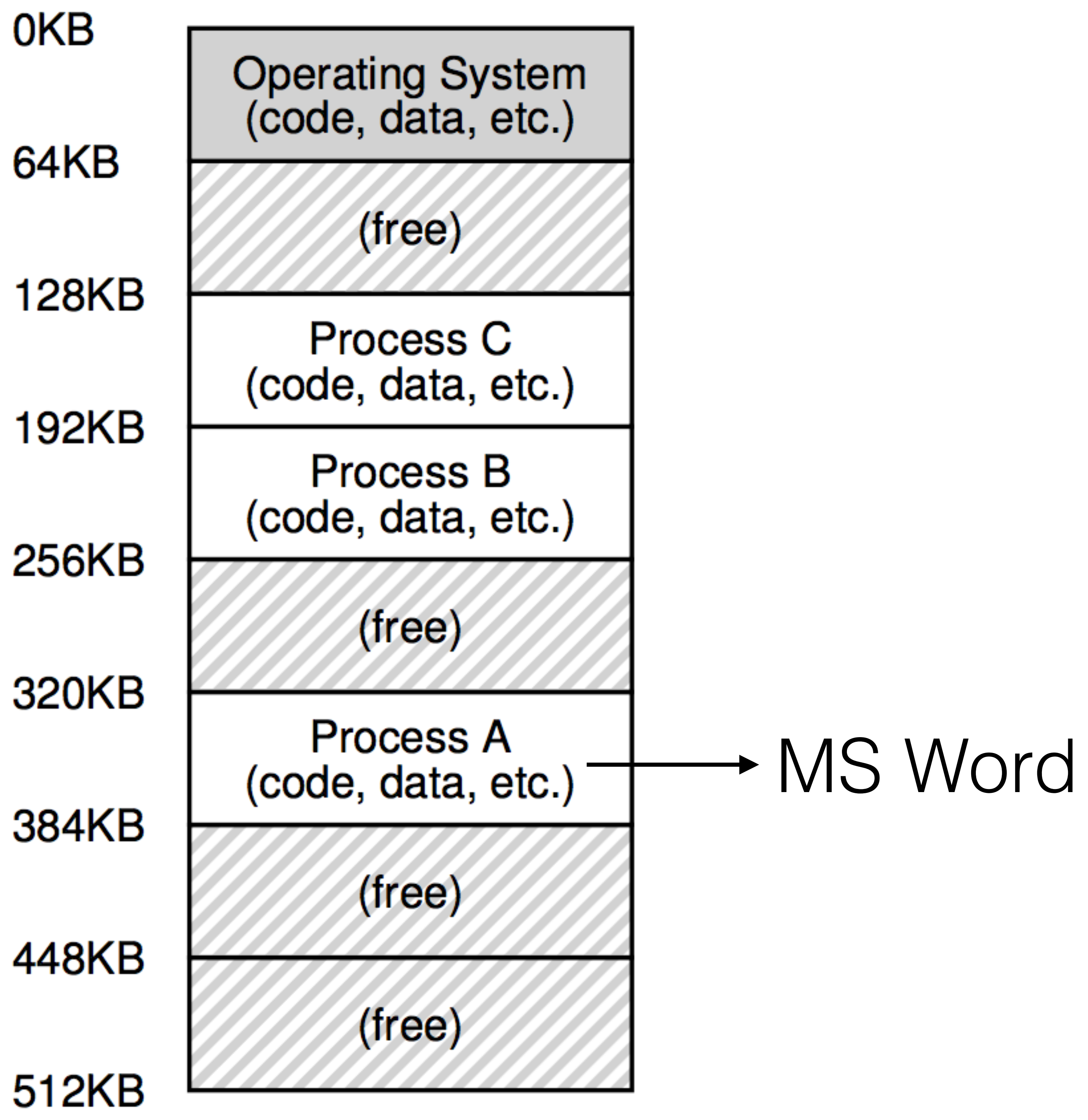
Early days
Single program



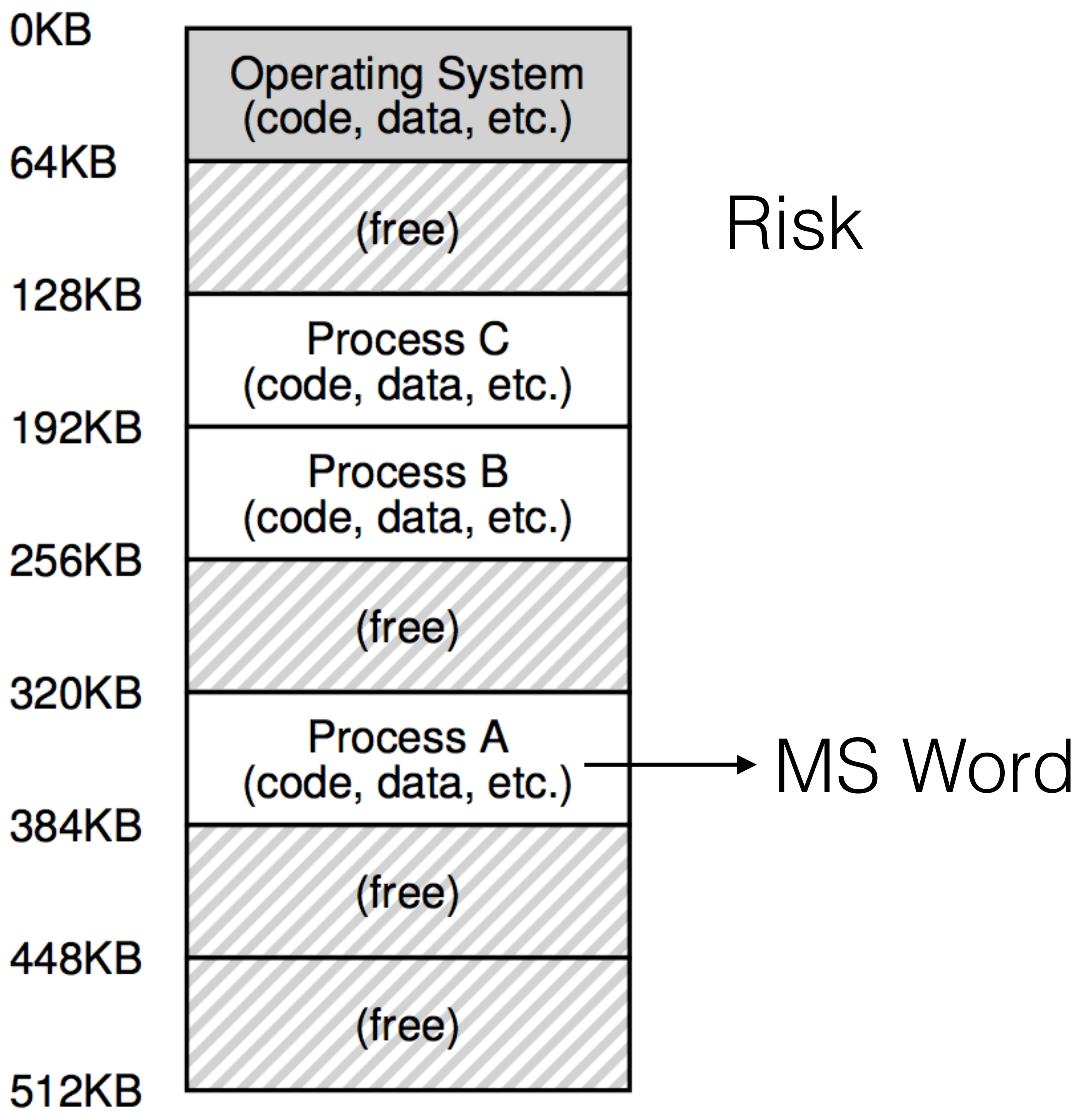
Shared Memory


0KB
Operating System (code, data, etc.)
64KB
(free)
128KB
Process C (code, data, etc.)
192KB
Process B (code, data, etc.)
256KB
(free)
320KB
Process A (code, data, etc.)
384KB
(free)
448KB
(free)
512KB

Shared Memory

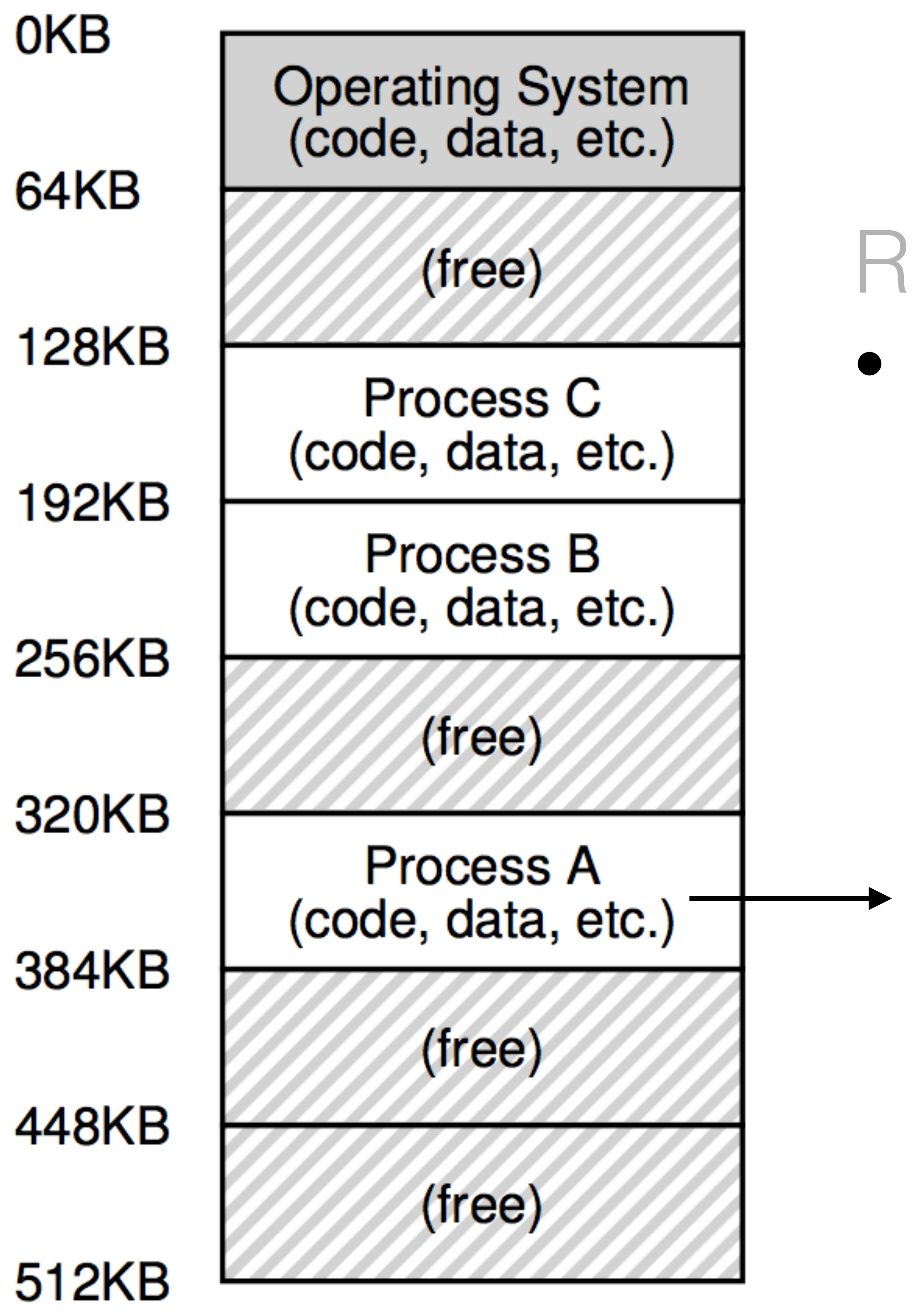

0KB
64KB
128KB
192KB
256KB
320KB
384KB
448KB
512KB
Operating System (code, data, etc.)
(free)
Process C (code, data, etc.)
Process B (code, data, etc.)
(free)
Process A (code, data, etc.)
MS Word
(free)
(free)

Shared Memory


0KB
64KB
128KB
192KB
256KB
320KB
384KB
448KB
512KB
Operating System (code, data, etc.)
(free)
Process C (code, data, etc.)
Process B (code, data, etc.)
(free)
Process A (code, data, etc.)
MS Word
(free)
(free)

Risk

Shared Memory



0KB
64KB
128KB
192KB
256KB
320KB
384KB
448KB
512KB
Operating System (code, data, etc.)
(free)
Process C (code, data, etc.)
Process B (code, data, etc.)
(free)
Process A (code, data, etc.)
(free)
(free)

Risk

- Programs accessing others' memory

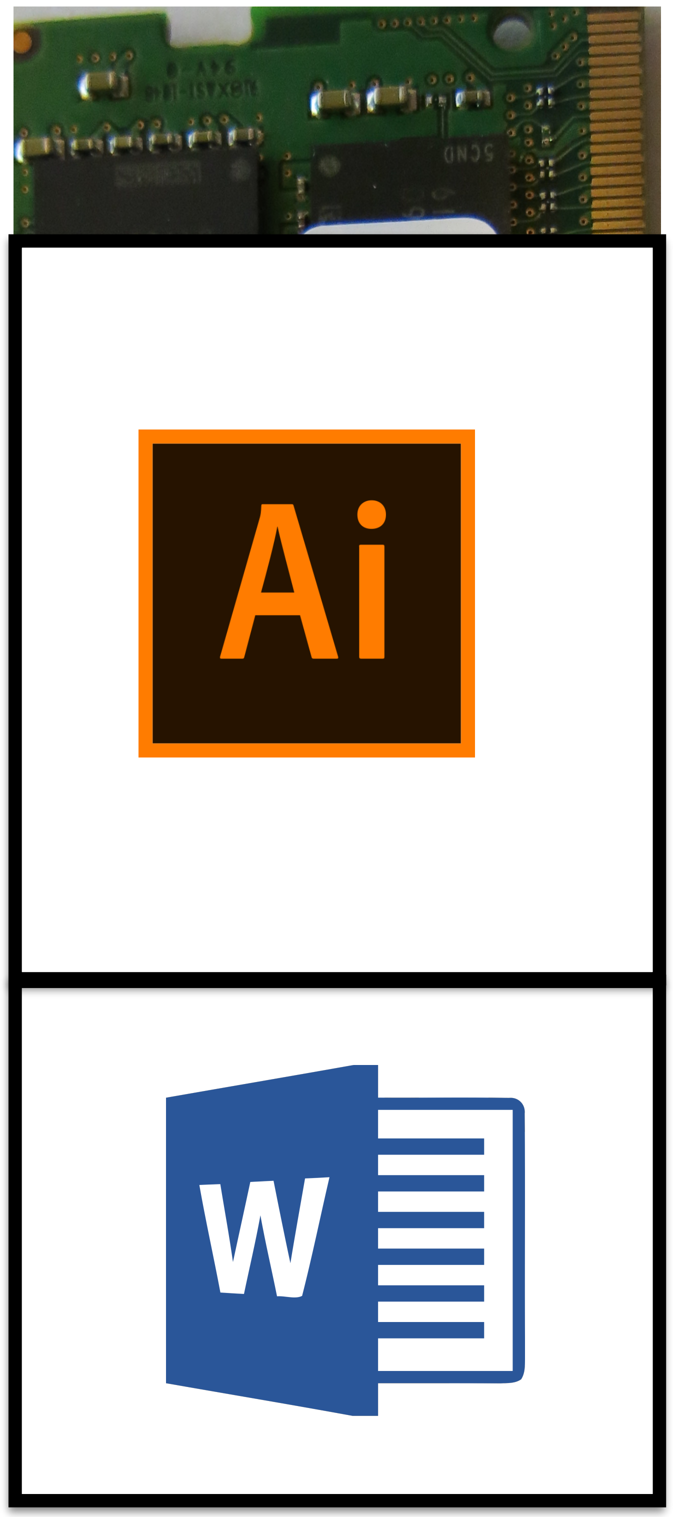
MS Word

Direct Physical Memory Multiplexing

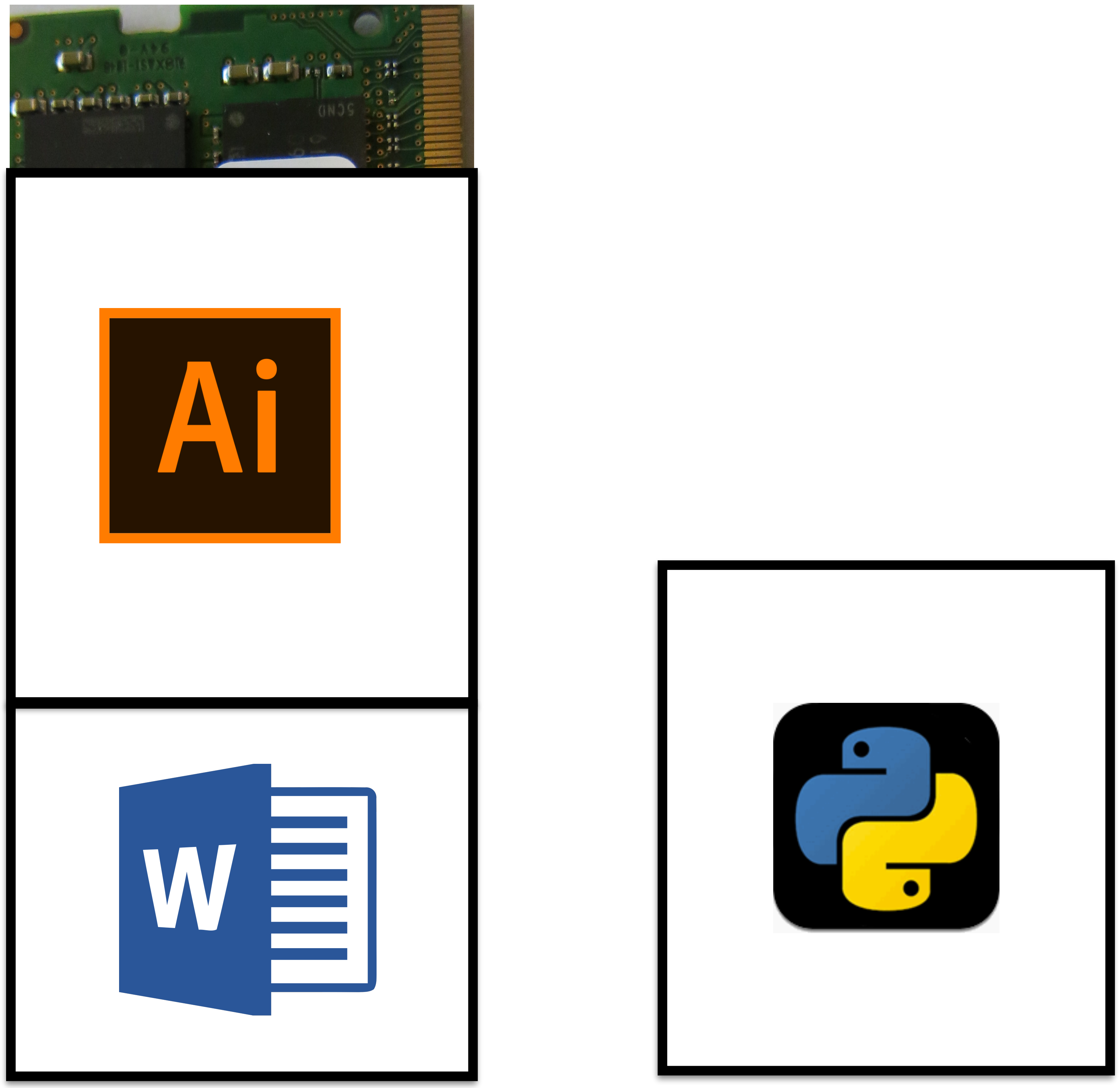
CE
MTA16ATF1G64HZ-2G1B1 1619
8GB 2RX8 PC4-2133P-SBB-11
Product of CHINA DPAGB2M014

Direct Physical Memory Multiplexing

Direct Physical Memory Multiplexing

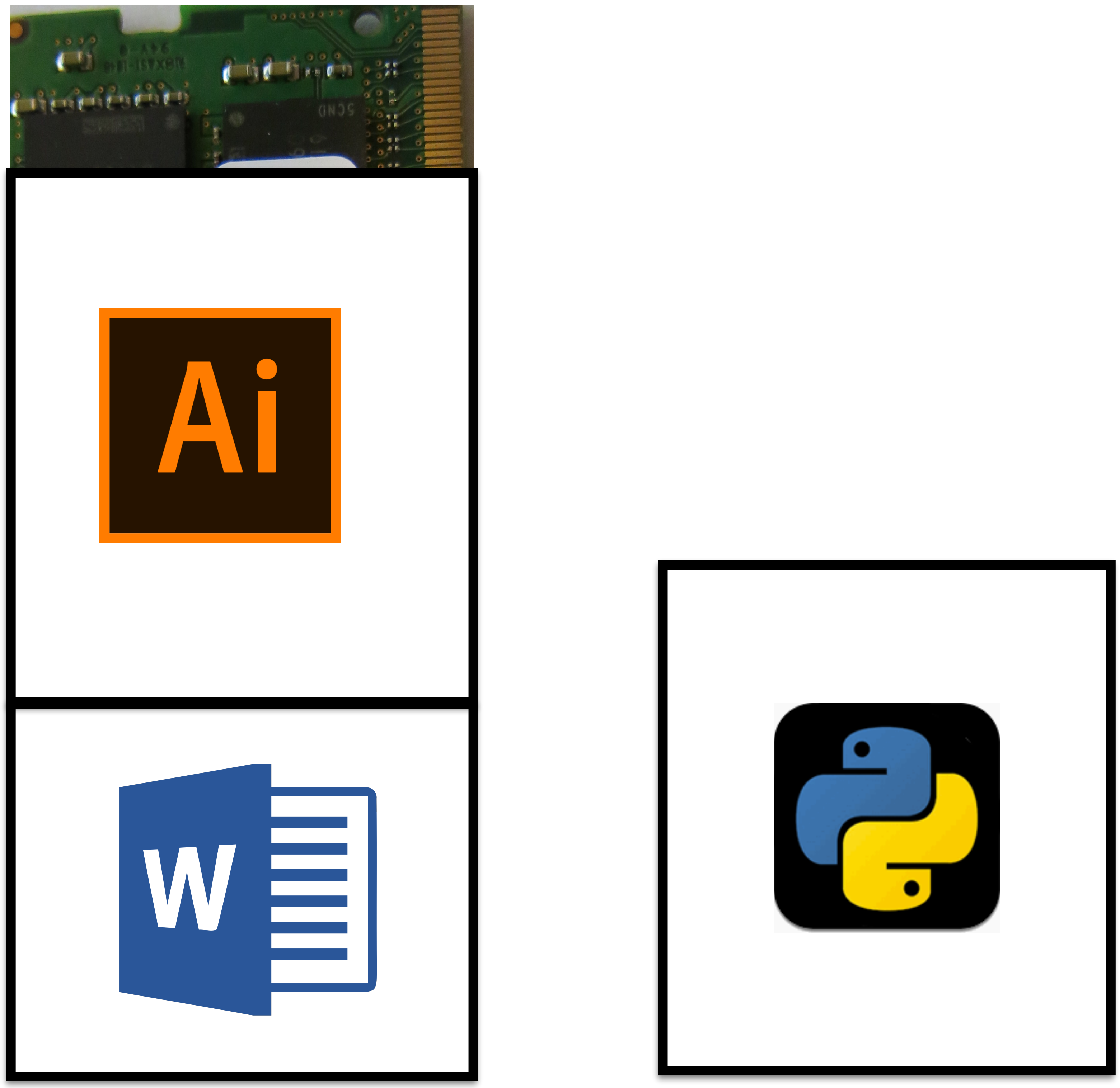


Direct Physical Memory Multiplexing



Direct Physical Memory Multiplexing

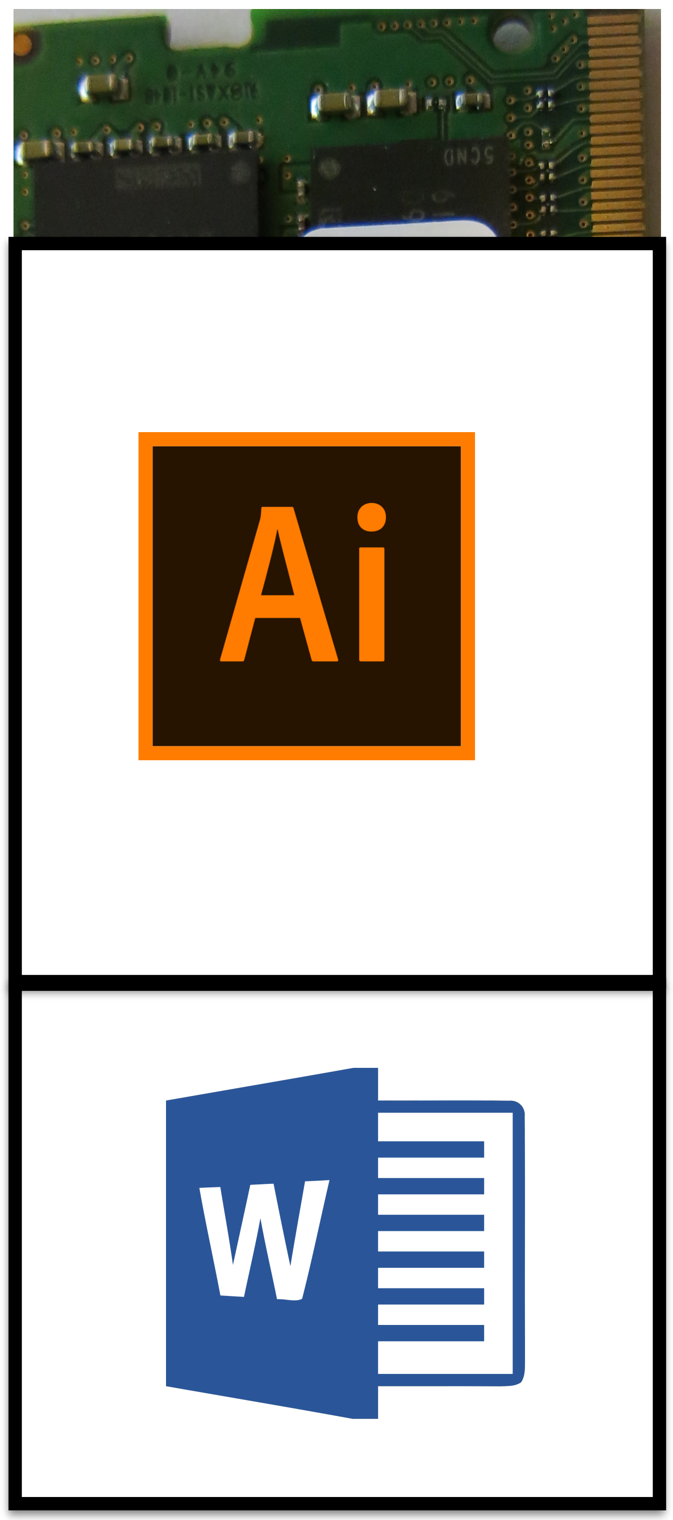
Limited to physical memory on the system



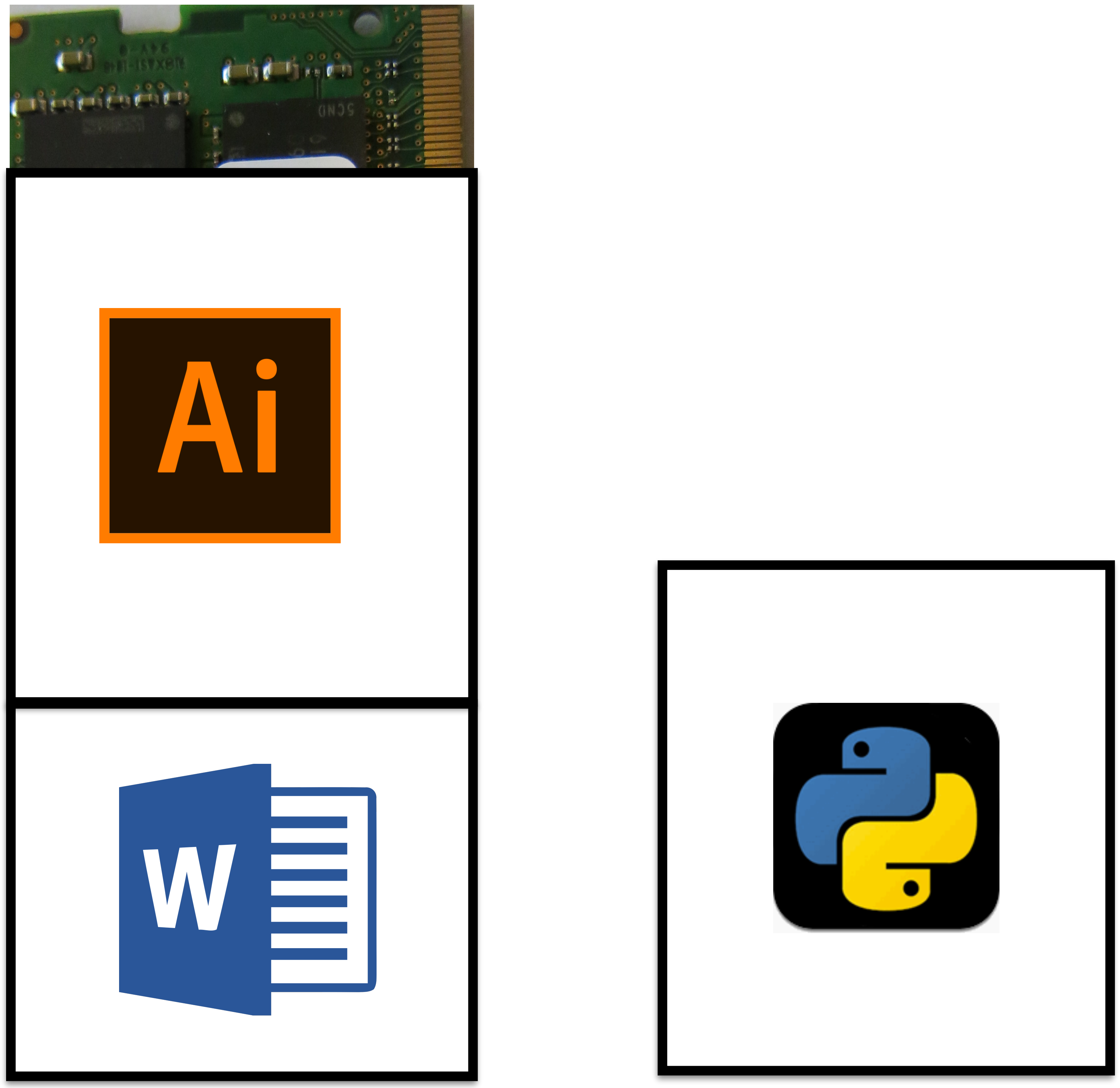
Direct Physical Memory Multiplexing

Direct Physical Memory Multiplexing

Direct Physical Memory Multiplexing

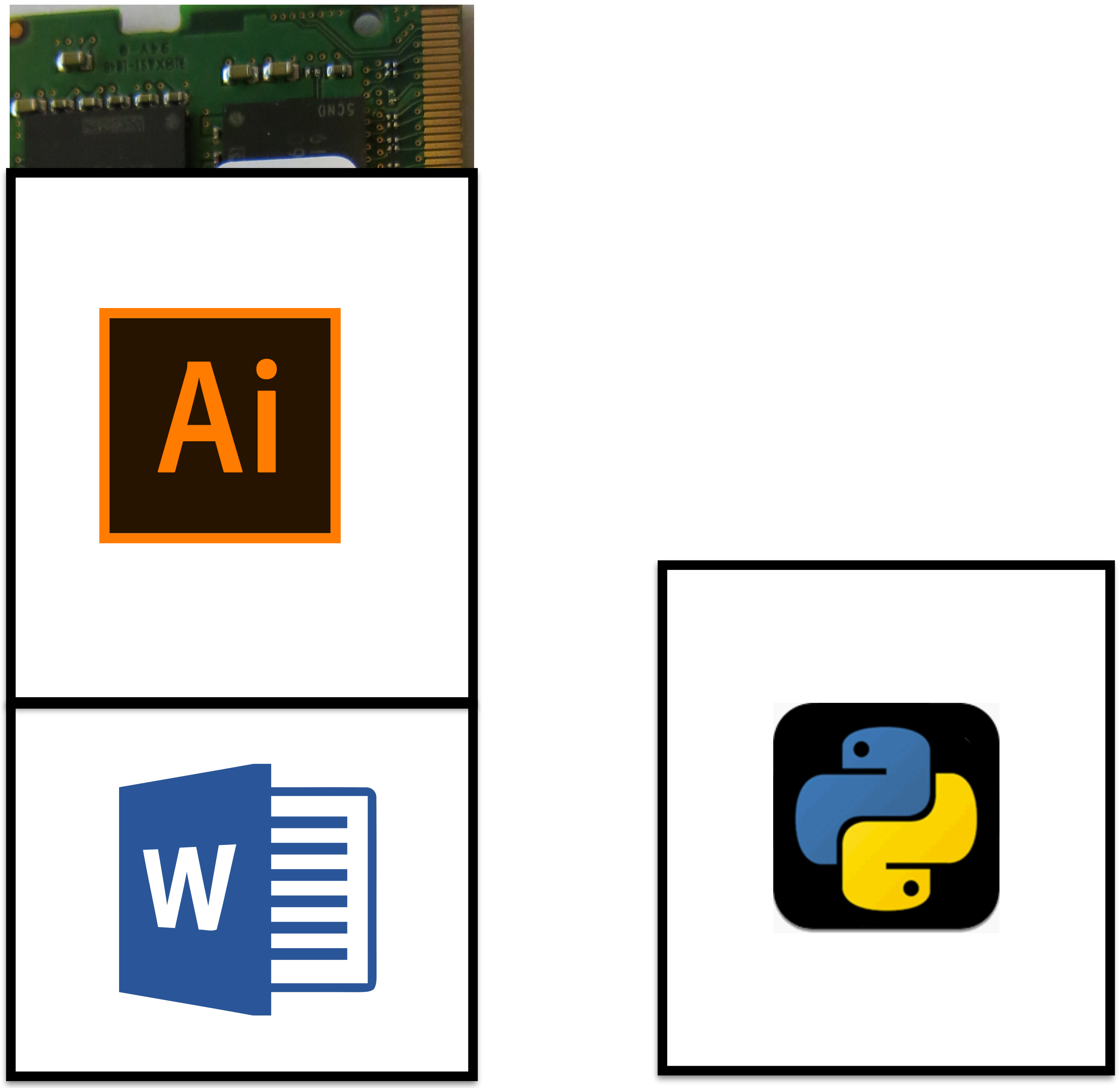


Direct Physical Memory Multiplexing

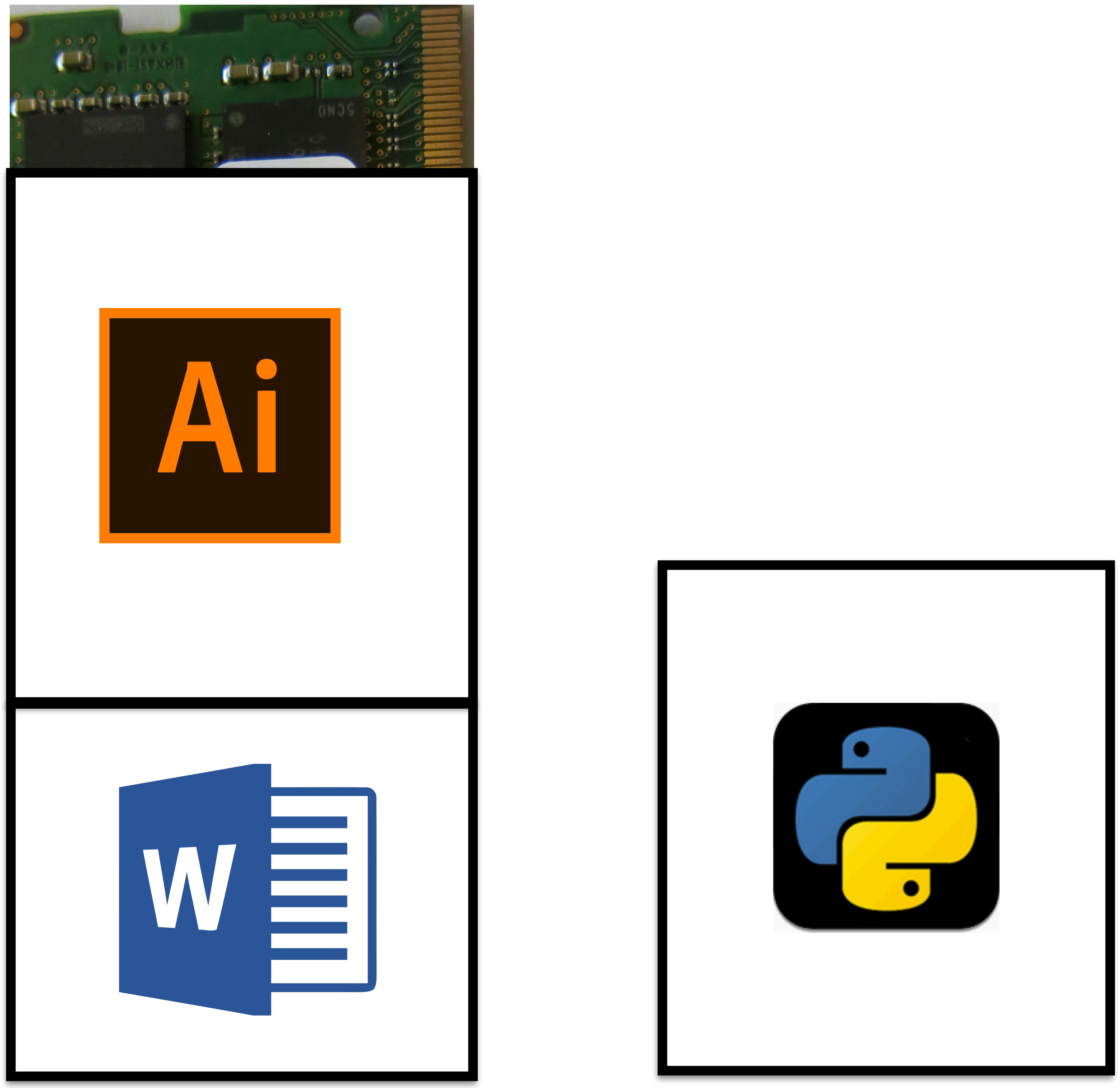


Direct Physical Memory Multiplexing

Limited to physical memory on the system

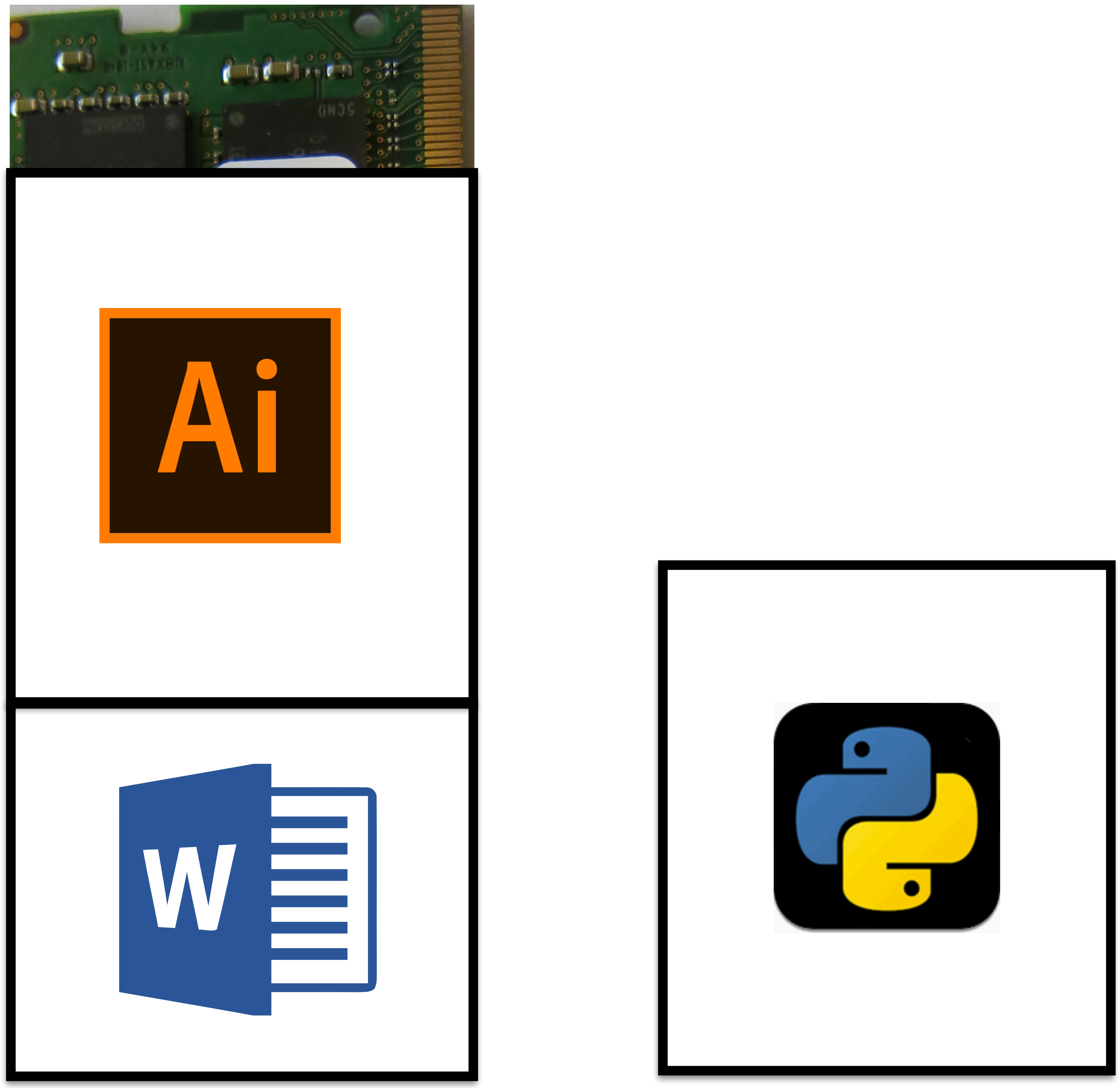


Direct Physical Memory Multiplexing

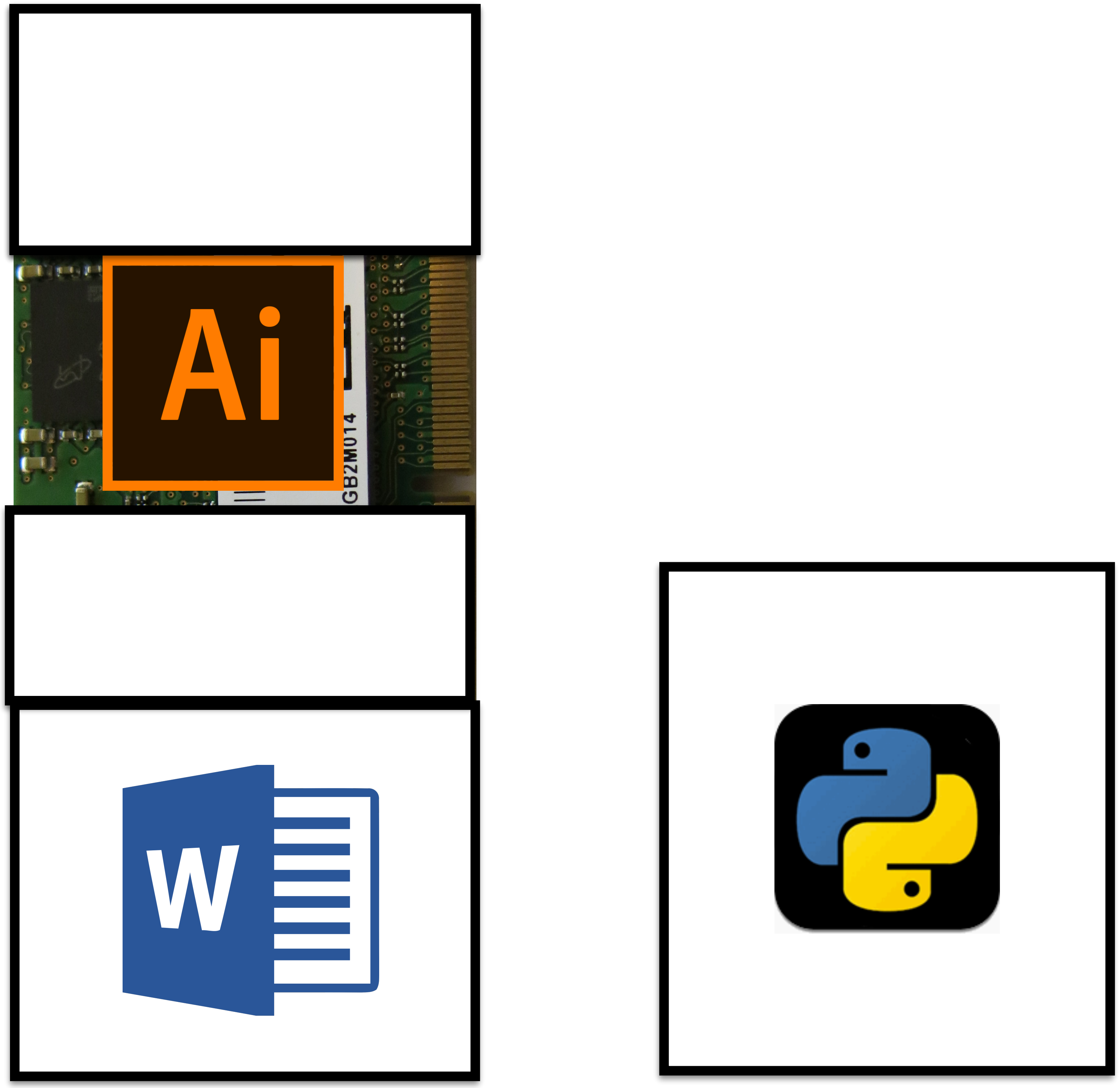


Direct Physical Memory Multiplexing

What if process says it wants full memory?

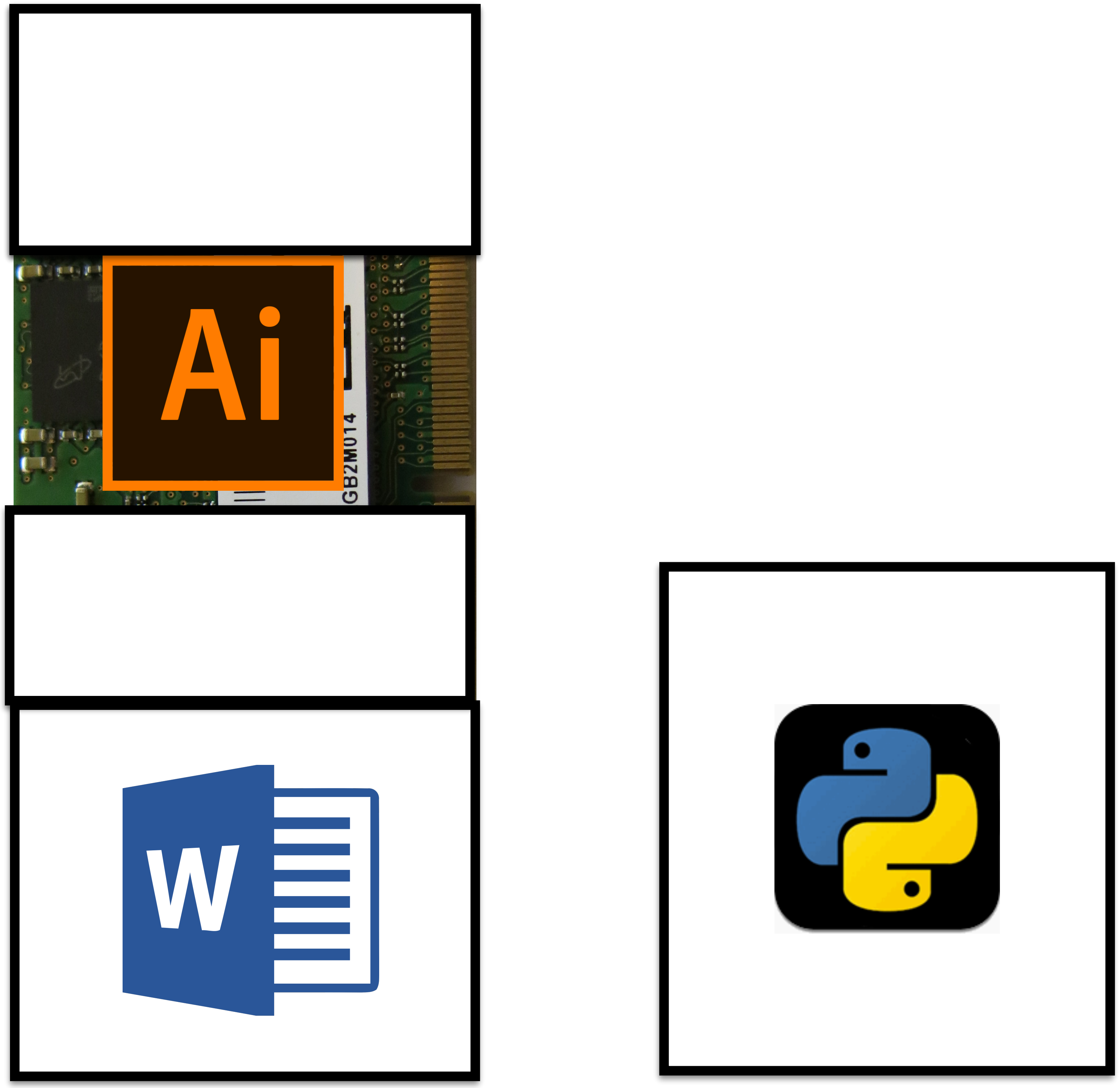


Direct Physical Memory Multiplexing



Direct Physical Memory Multiplexing

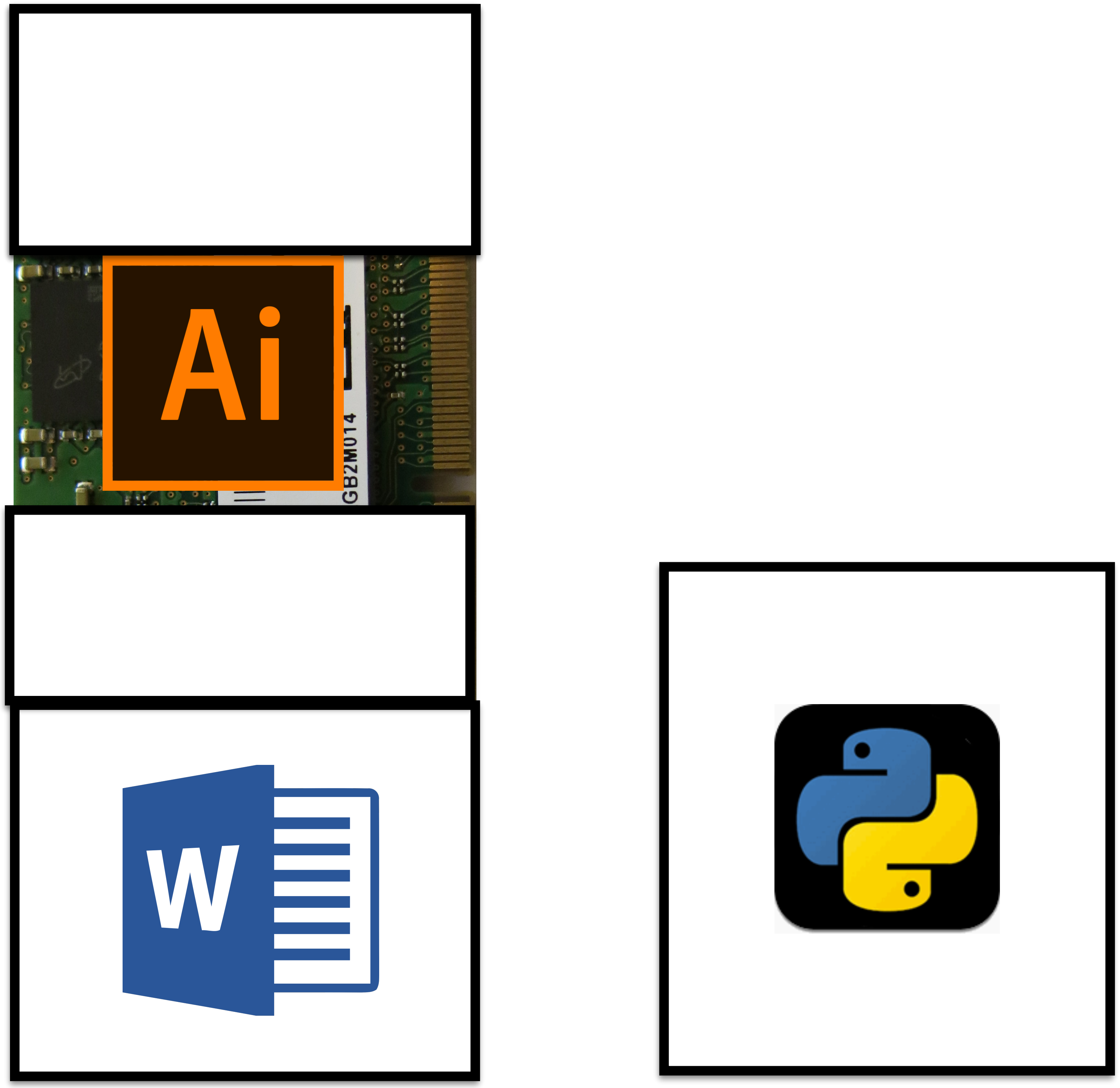
What if process says it wants full memory?



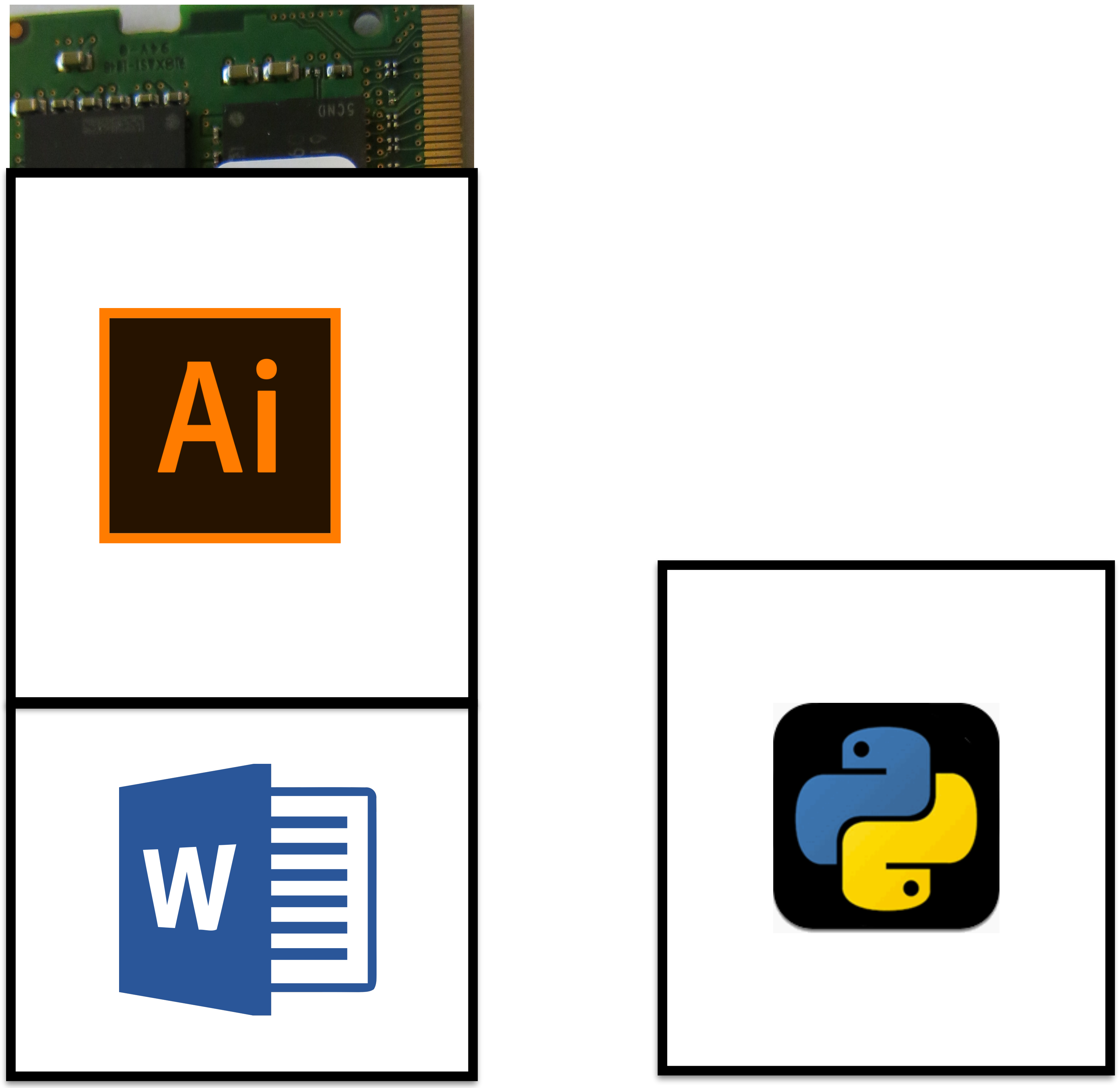
Direct Physical Memory Multiplexing

Internal Fragmentation

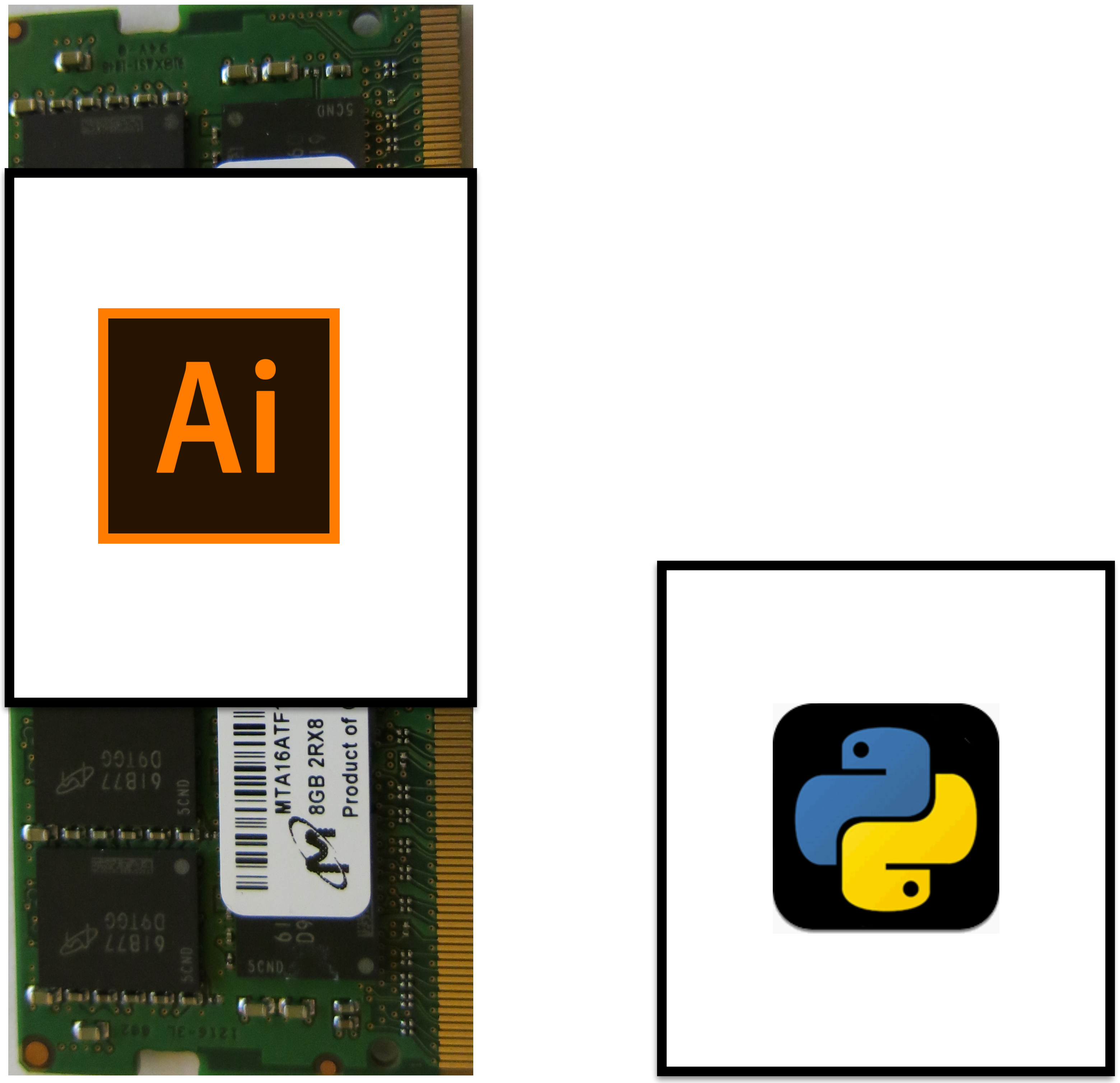
What if process says it wants full memory?



Direct Physical Memory Multiplexing



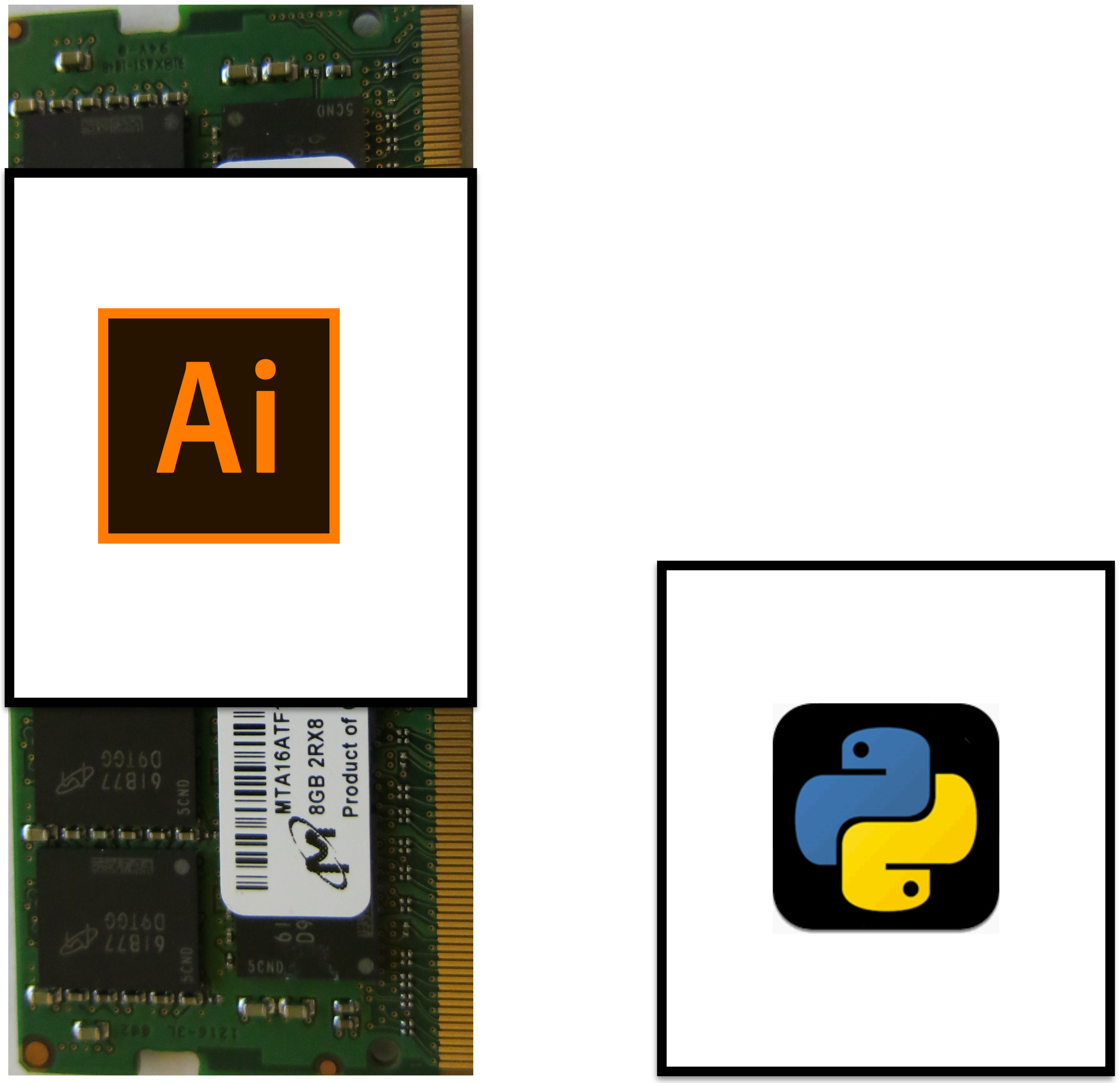
Direct Physical Memory Multiplexing



Direct Physical Memory Multiplexing

Can Python run now?

Total memory - Memory req for Illustrator > = Memory req for Python

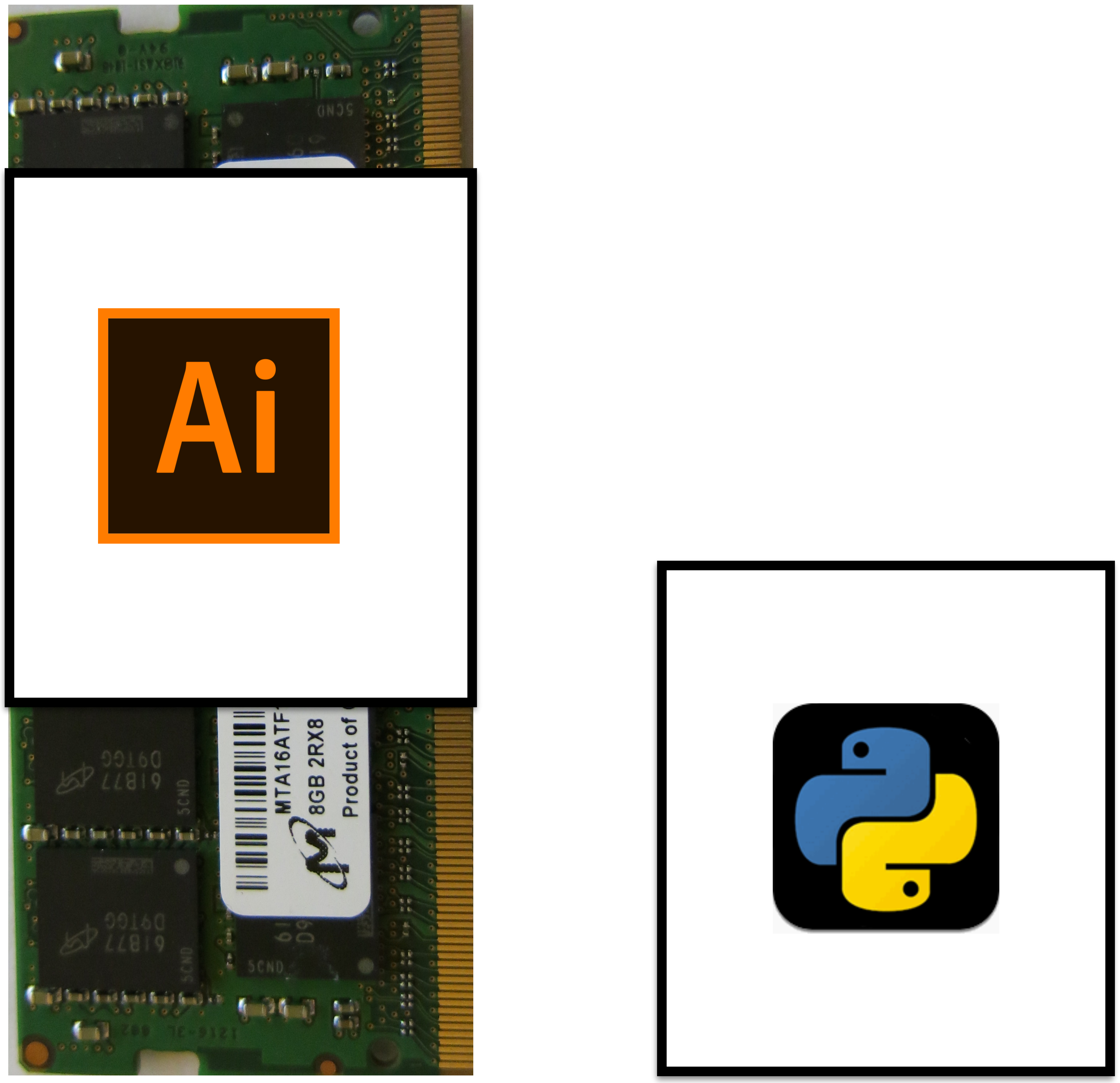


Direct Physical Memory Multiplexing

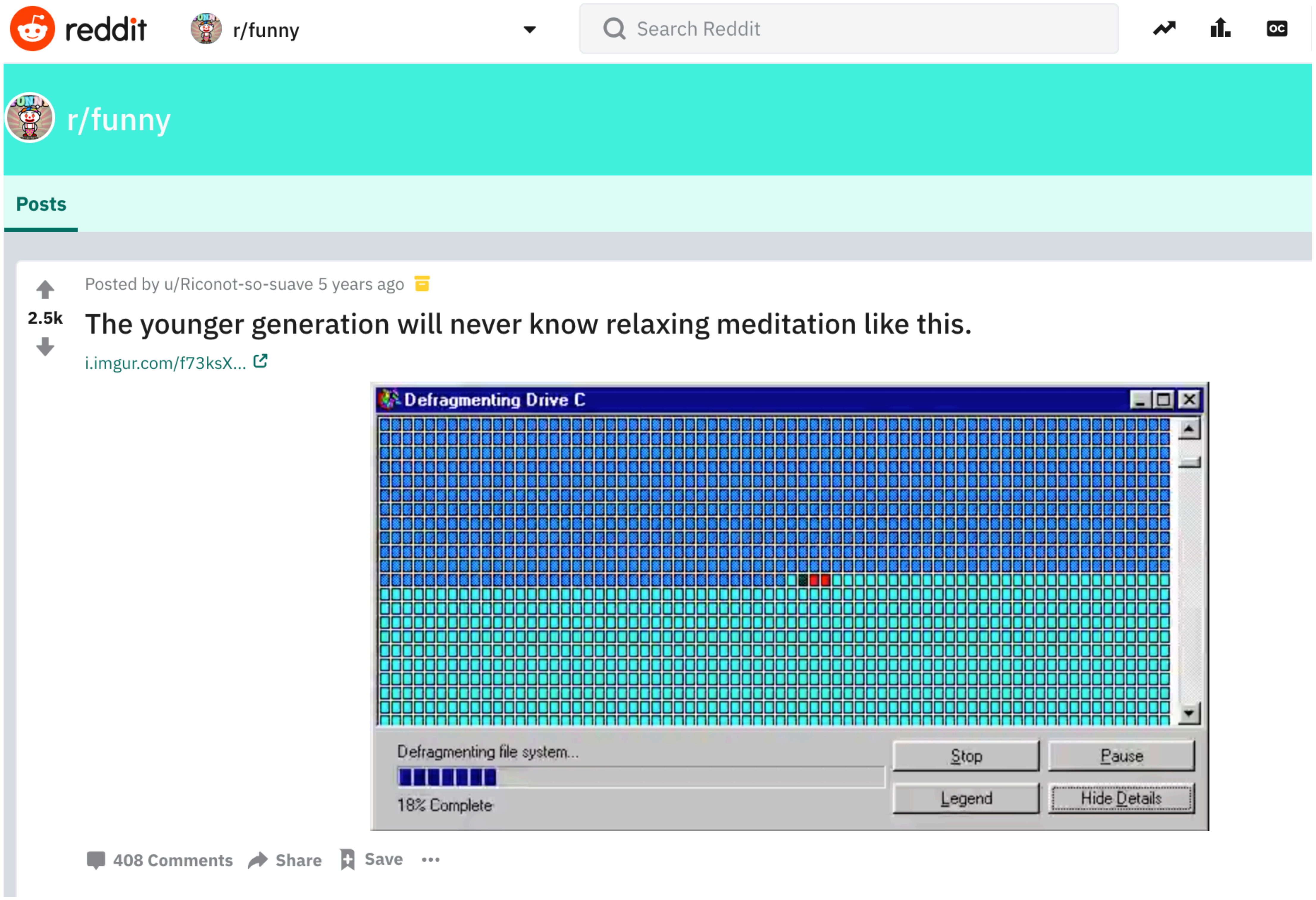
External Fragmentation

Can Python run now?

Total memory - Memory req for Illustrator > = Memory req for Python



Defragmentation Memories …



reddit r/funny Search Reddit

r/funny

Posts

2.5k Posted by u/Riconot-so-suave 5 years ago

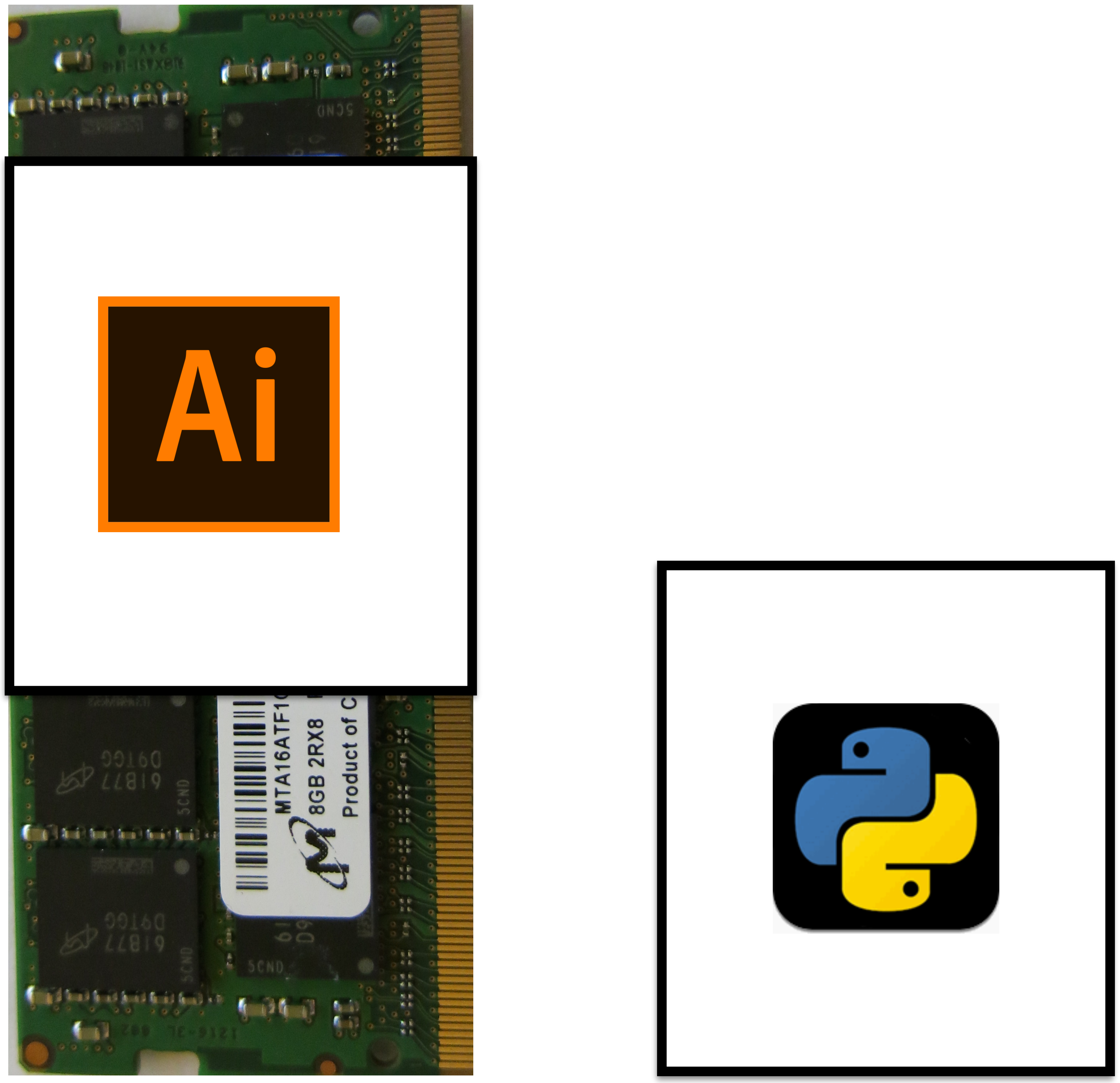
The younger generation will never know relaxing meditation like this.

i.imgur.com/f73ksX...

408 Comments Share Save ...

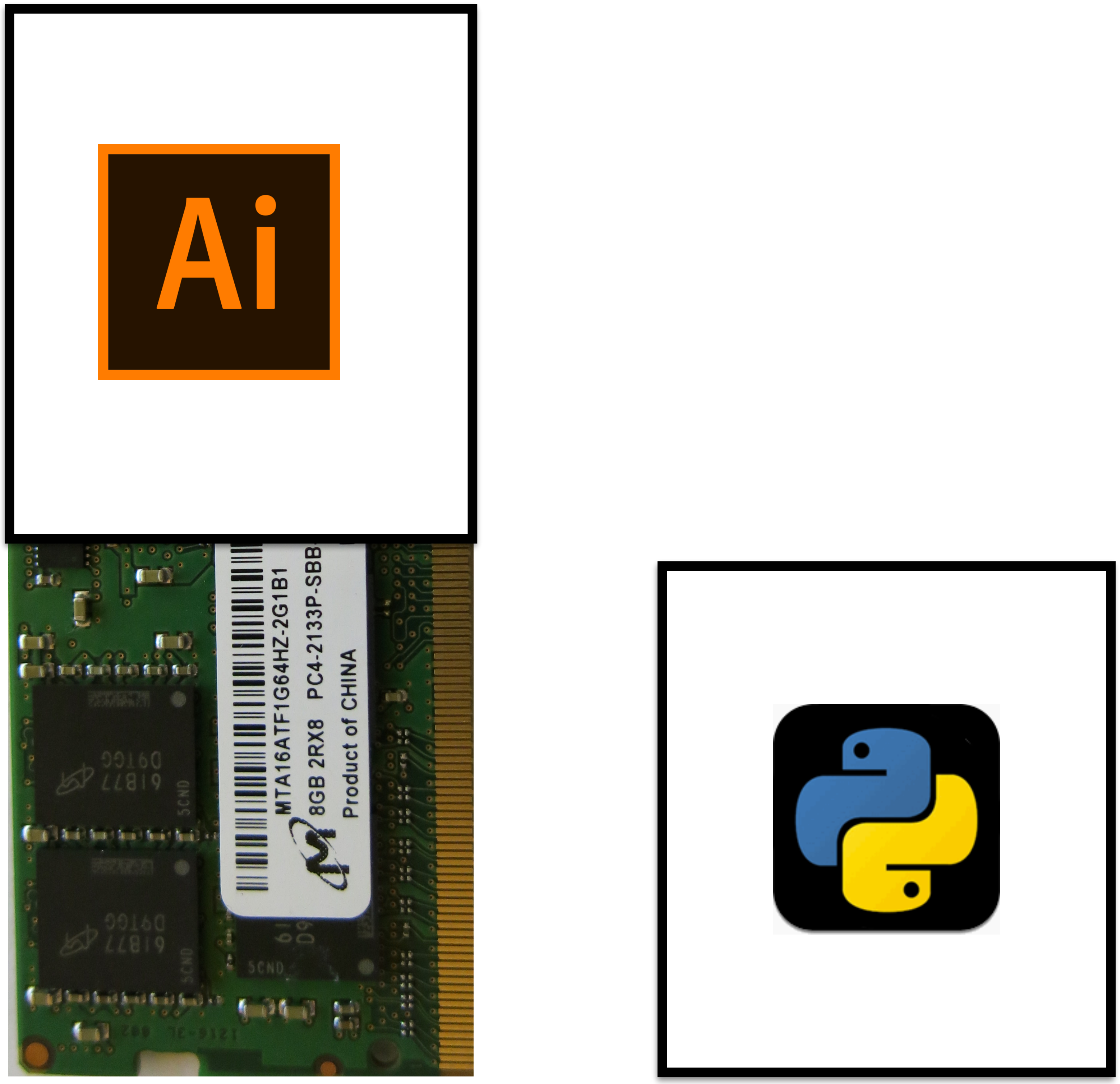
Direct Physical Memory Multiplexing

Defragmentation



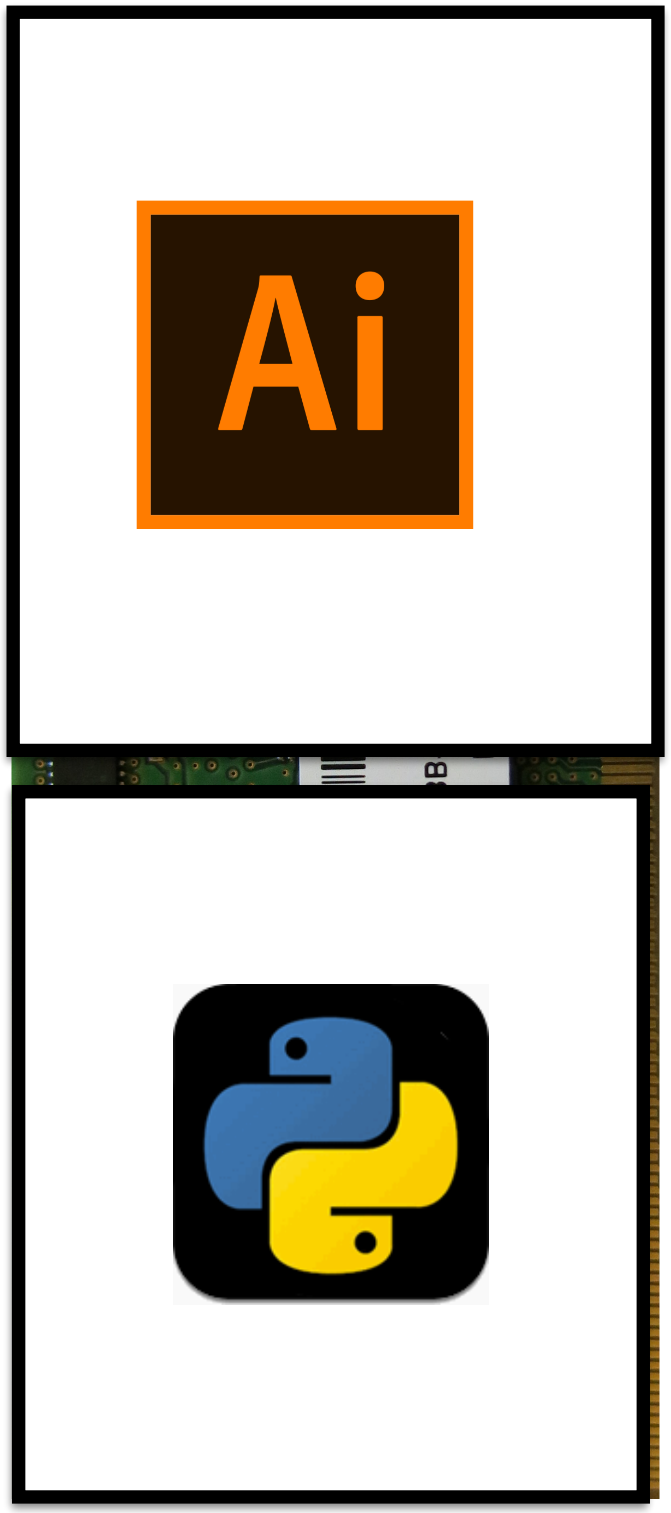
Direct Physical Memory Multiplexing

Defragmentation

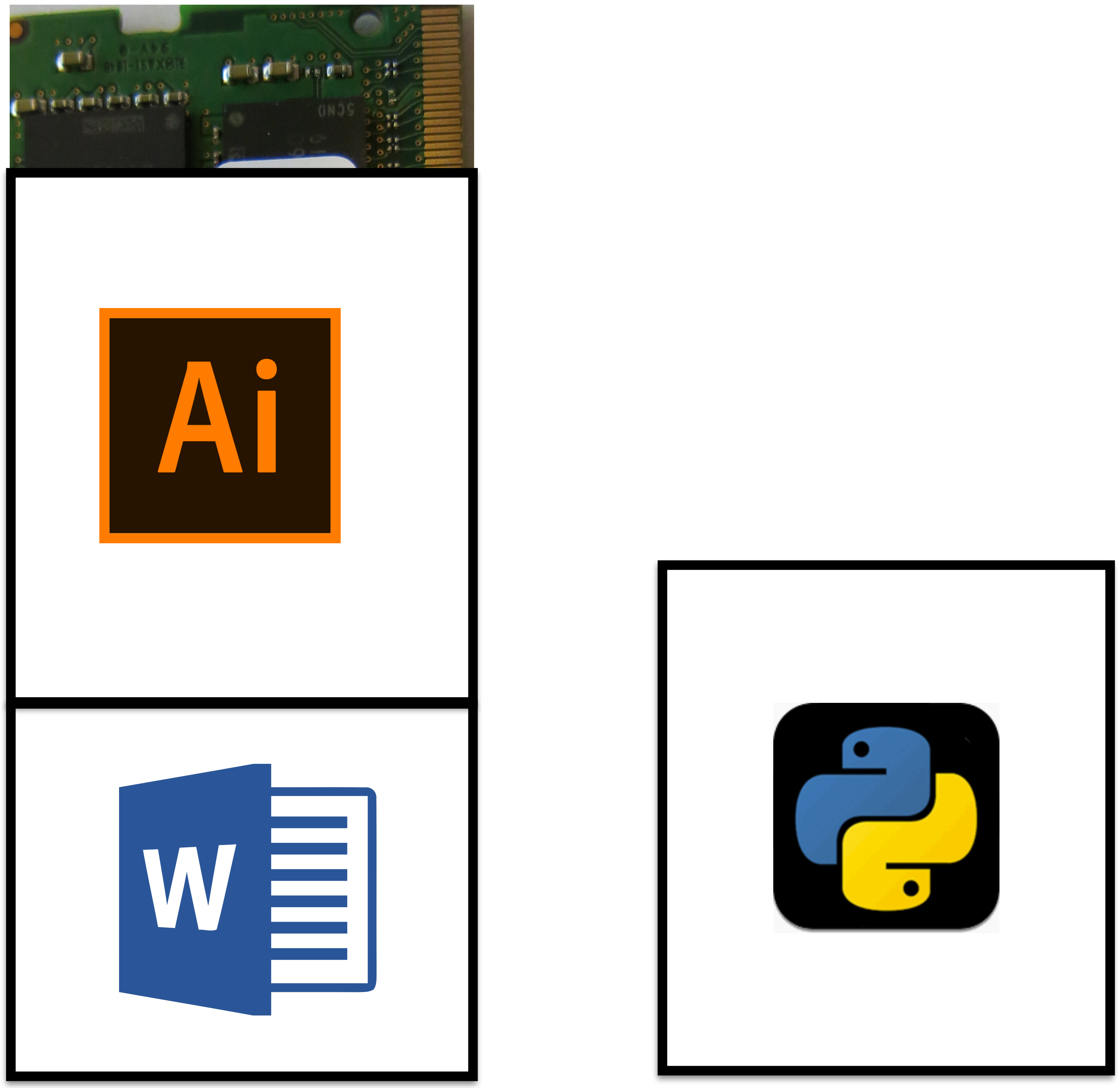


Direct Physical Memory Multiplexing

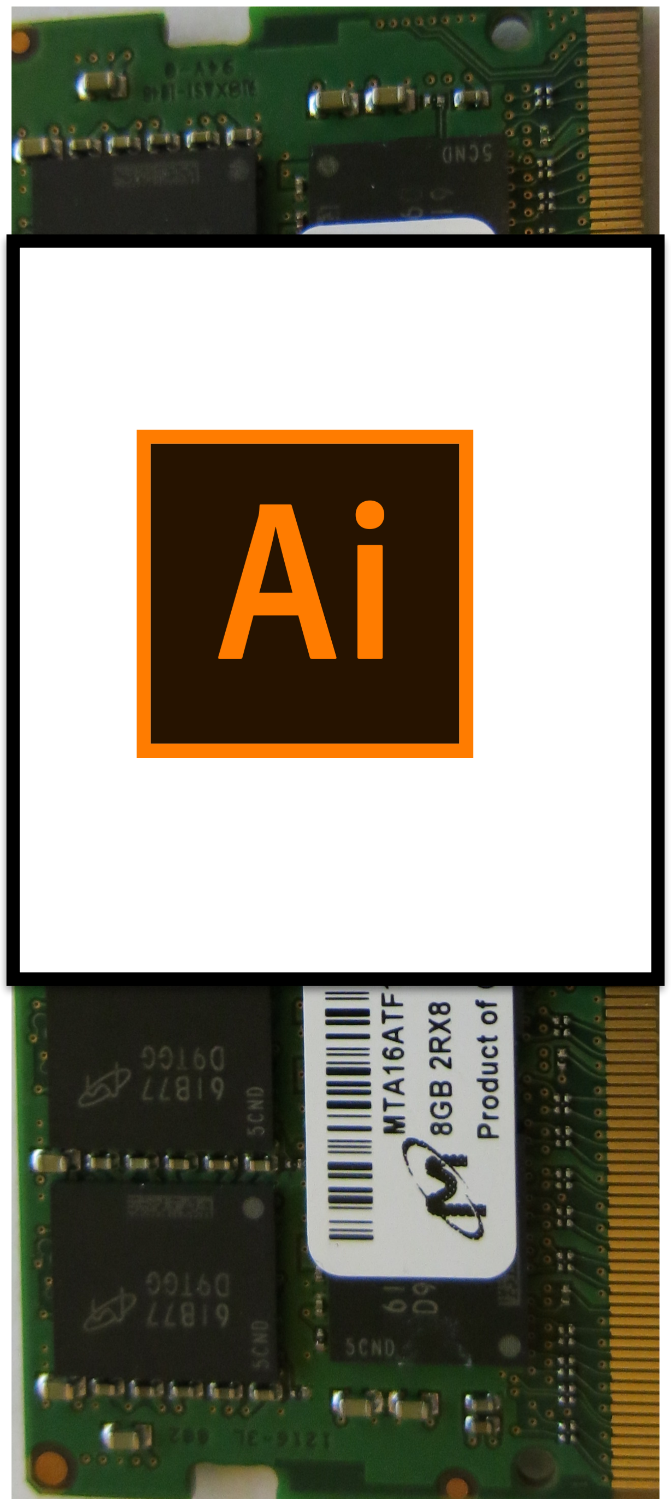
Defragmentation

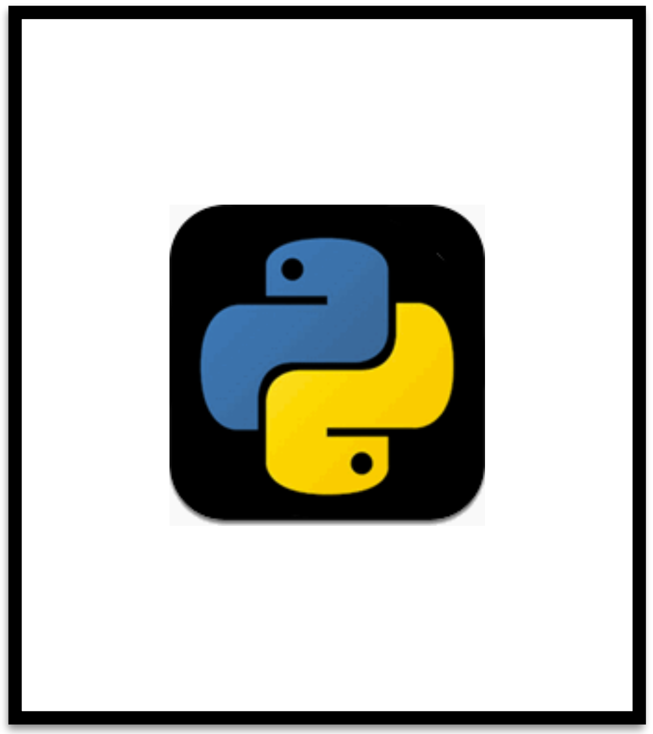


Direct Physical Memory Multiplexing



Direct Physical Memory Multiplexing

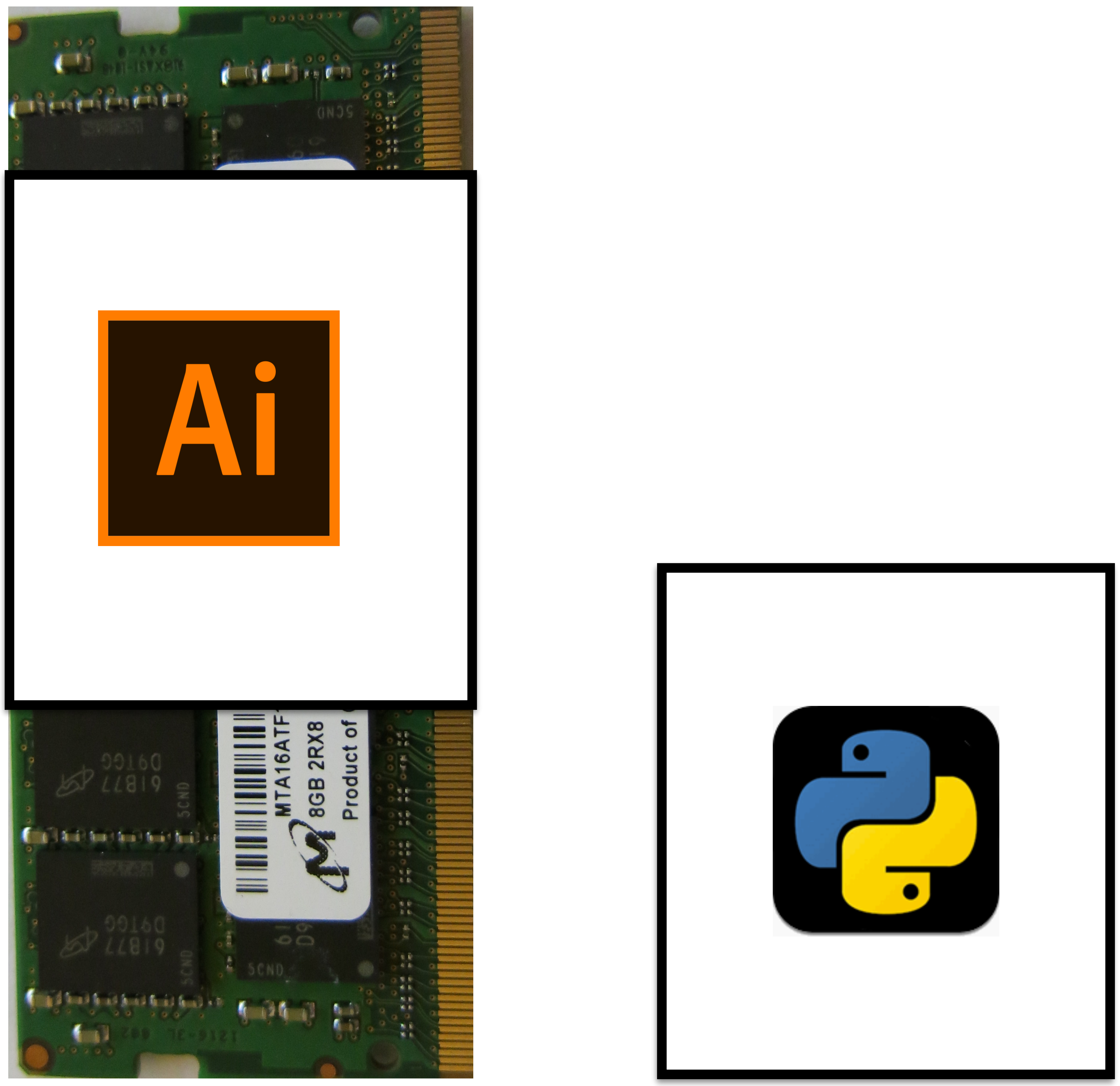




Direct Physical Memory Multiplexing

Can Python run now?

Total memory - Memory req for Illustrator > = Memory req for Python

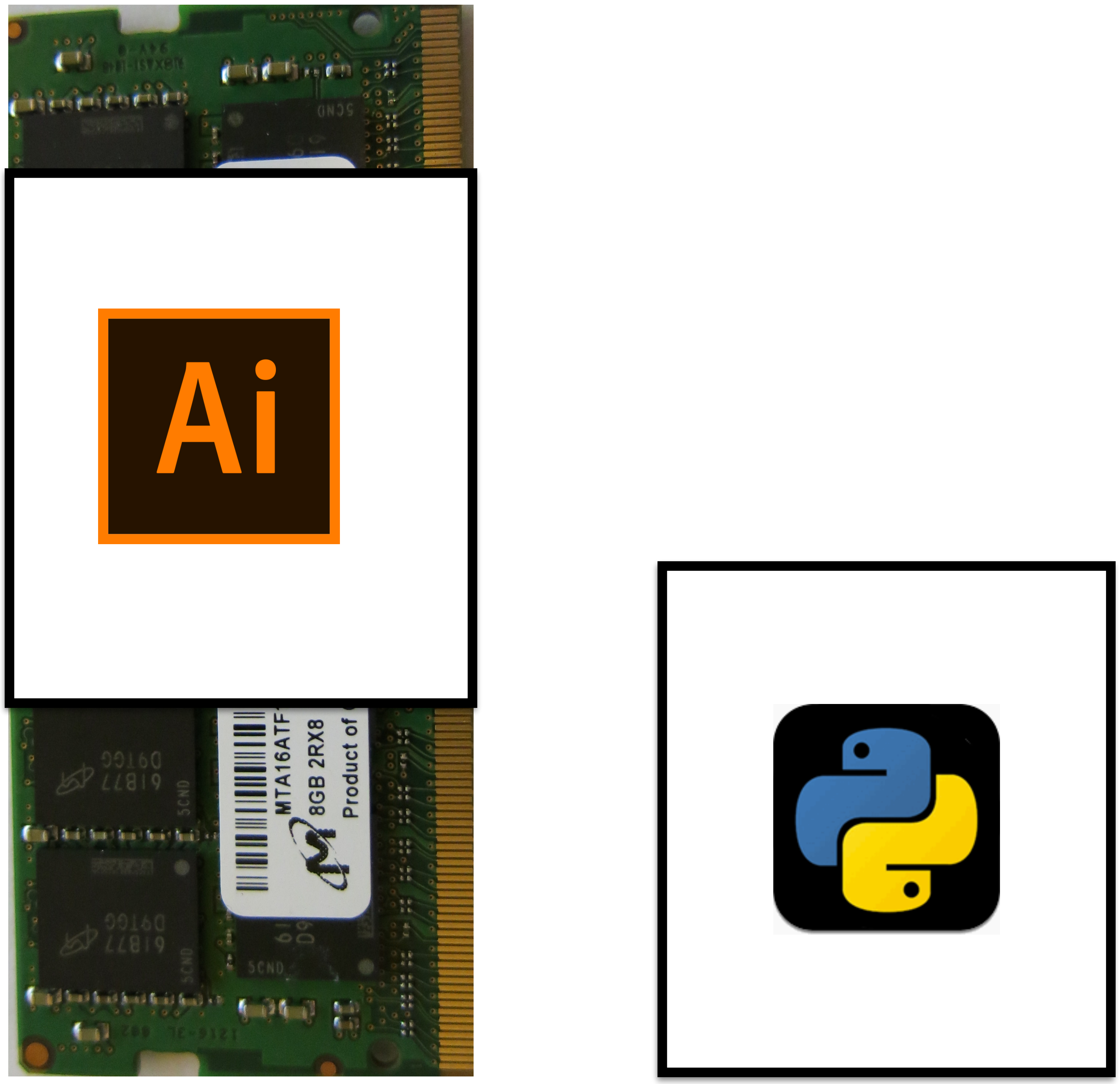


Direct Physical Memory Multiplexing

External Fragmentation

Can Python run now?

Total memory - Memory req for Illustrator > = Memory req for Python



Goals of OS for Memory Virtualisation/ Management

1. Transparency
   1. Physical memory is invisible to user program
   2. Program thinks it has own private large (contiguous + plentiful) memory
2. Efficiency
   1. Not taking very long
   2. Not taking too much space
3. Protection/Isolation
   1. Protect processes from each other

Memory Interface

1. Load (address)
2. Store (address, value)

Physical v/s Virtual Memory

Physical v/s Virtual Memory

1. Abstraction : Break the connection between physical memory and an address

Physical v/s Virtual Memory

1. Abstraction : Break the connection between physical memory and an address
2. Data accessed using memory interface is virtual address

Physical v/s Virtual Memory

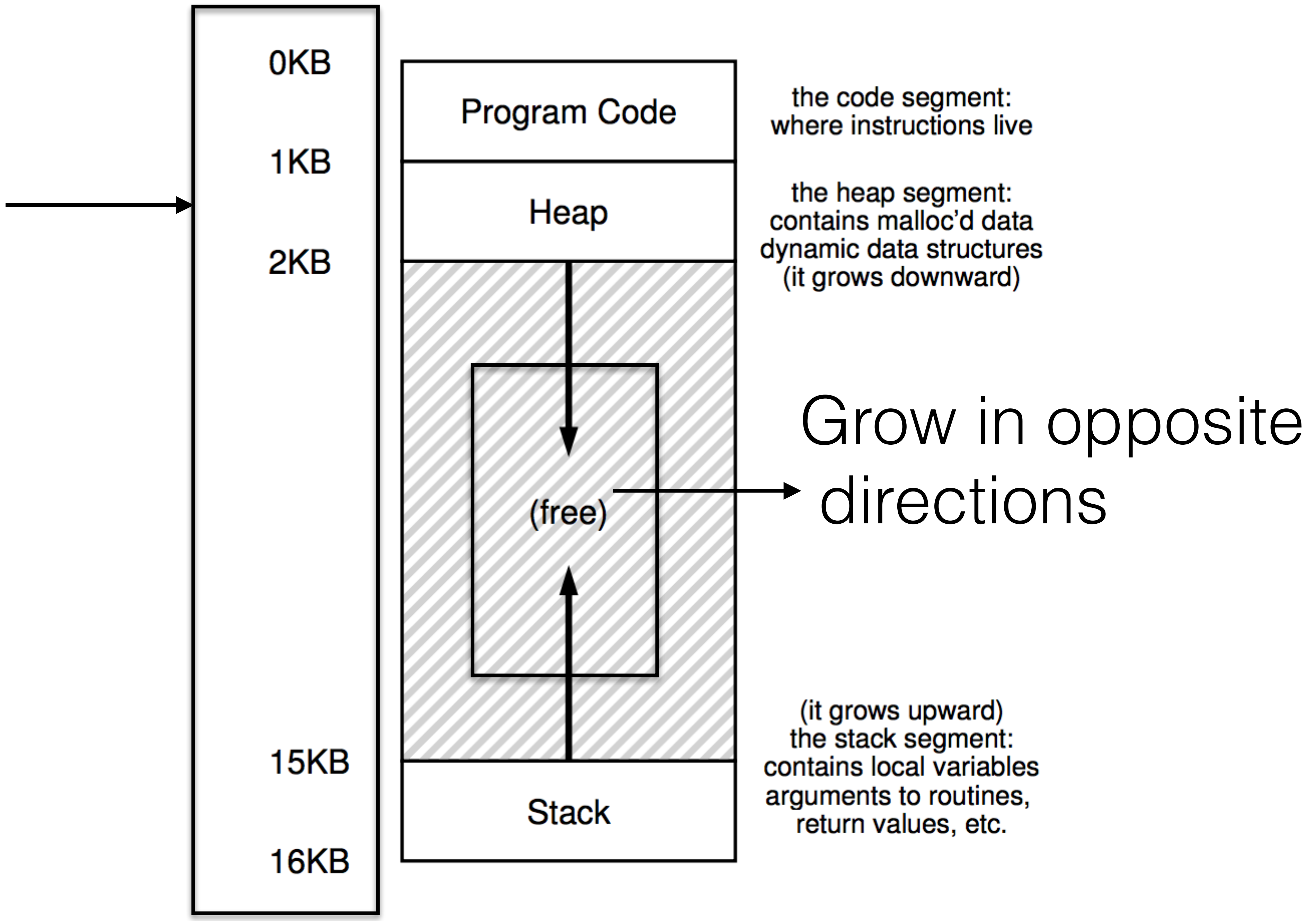
1. Abstraction : Break the connection between physical memory and an address
2. Data accessed using memory interface is virtual address
    1. Physical address points to memory

Physical v/s Virtual Memory

1. Abstraction : Break the connection between physical memory and an address
2. Data accessed using memory interface is virtual address
   1. Physical address points to memory
   2. Virtual address points to something *acting like memory*

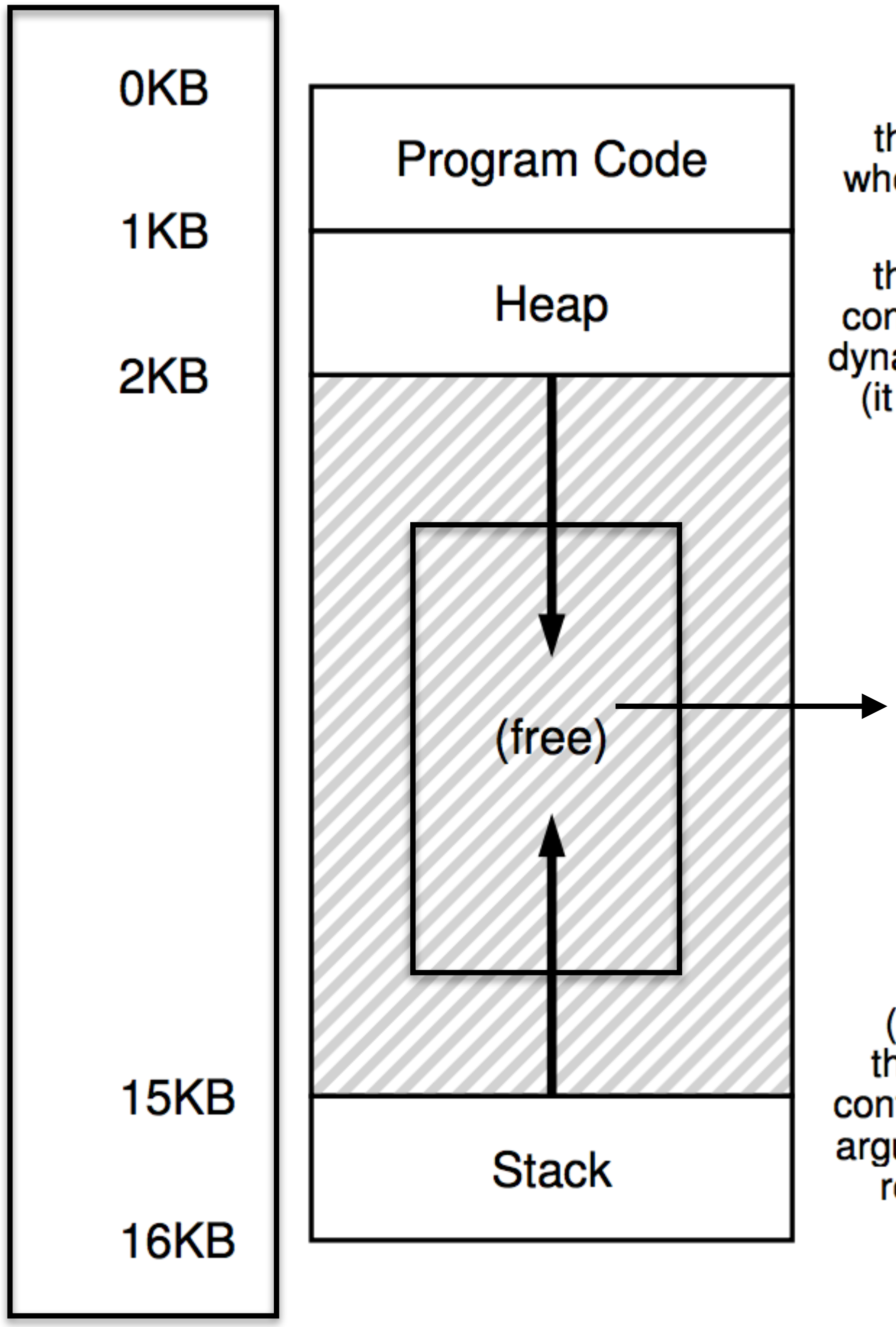
Address Space

Virtual Address


0KB
1KB
2KB
15KB
16KB
Program Code
the code segment: where instructions live
Heap
the heap segment: contains malloc'd data dynamic data structures (it grows downward)
(free)
Stack
(it grows upward) the stack segment: contains local variables arguments to routines, return values, etc.

Grow in opposite directions

Stack Overflow?!


0KB
1KB
2KB
15KB
16KB
Program Code
Heap
(free)
Stack
the code segment: where instructions live
the heap segment: contains malloc'd data dynamic data structures (it grows downward)
(it grows upward) the stack segment: contains local variables arguments to routines, return values, etc.

What if they meet?

Self-study : what is stack overflow?

Stack Overflow?!

```
def fib(n):
    if n==1 or n==0:
        return 1
    else:
        return n*fib(n-1)
```

Exec Revisited

`fork_same_address.c`

Example

```
void func() {
    int x = 3000; //
    x = x + 3;    //
    ...
```

Example

```
void func() {
    int x = 3000; //
    x = x + 3;    //
    ...
```

$\xrightarrow{\text{Compiler}}$

Example

```
void func() {
    int x = 3000; //
    x = x + 3;    //
    ...
```

$\xrightarrow{\text{Compiler}}$

```
128: movl 0x0(%ebx), %eax
132: addl $0x03, %eax
135: movl %eax, 0x0(%ebx)
```

Example

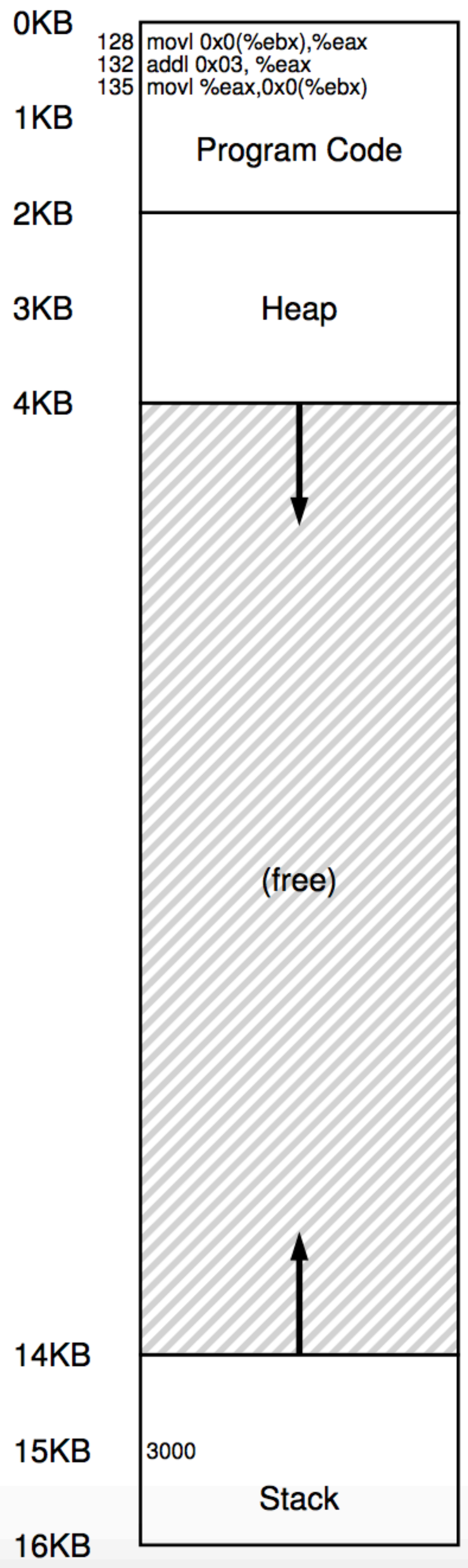
$\xrightarrow{\text{Compiler}}$

```
128: movl 0x0(%ebx), %eax
132: addl $0x03, %eax
135: movl %eax, 0x0(%ebx)
```

Example

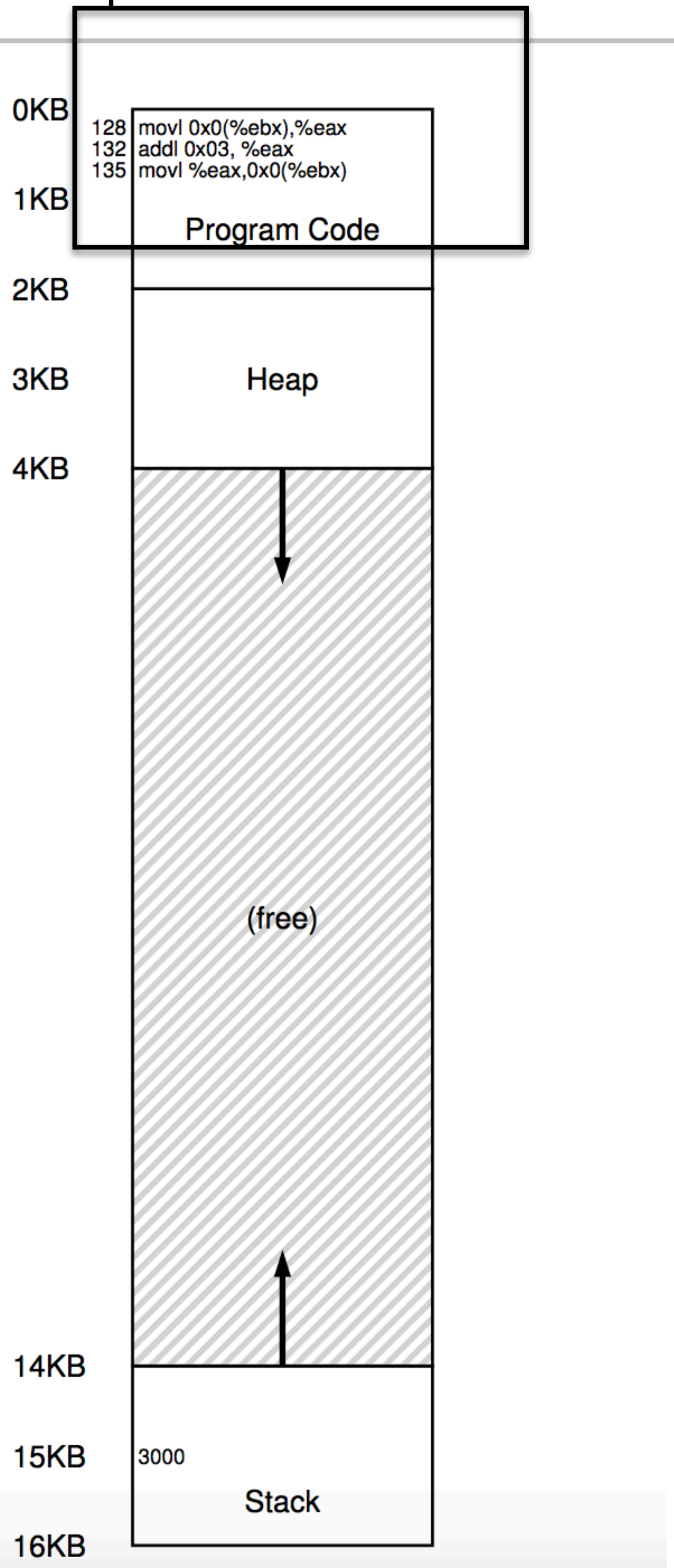
```
128: movl 0x0(%ebx), %eax
132: addl $0x03, %eax
135: movl %eax, 0x0(%ebx)
```

Example



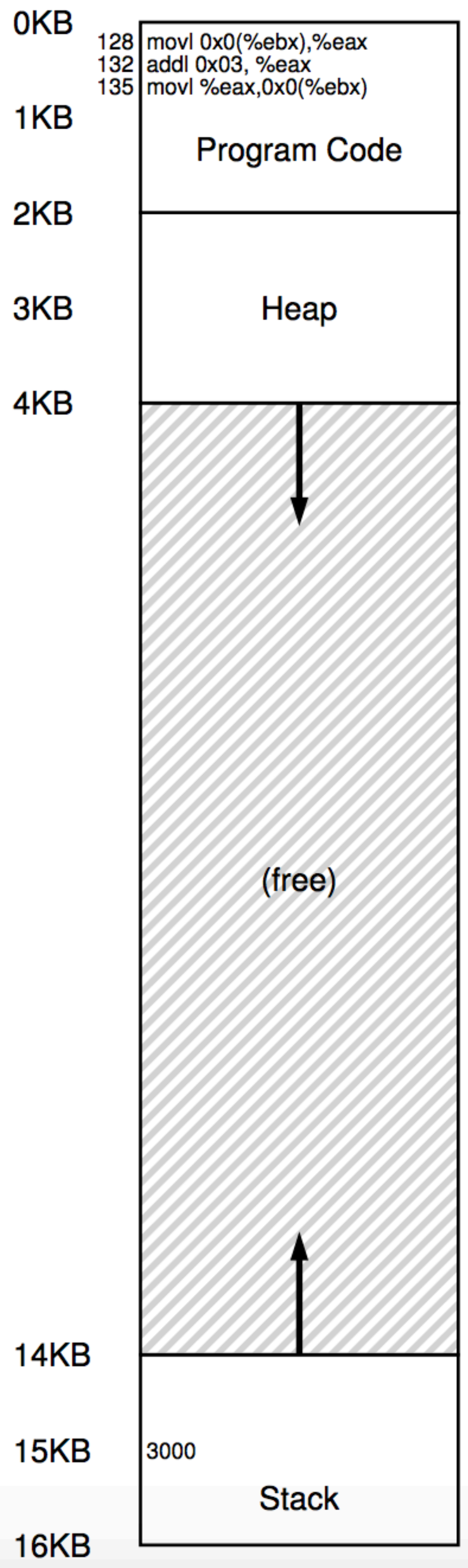
```
128: movl 0x0(%ebx), %eax
132: addl $0x03, %eax
135: movl %eax, 0x0(%ebx)
```

Example



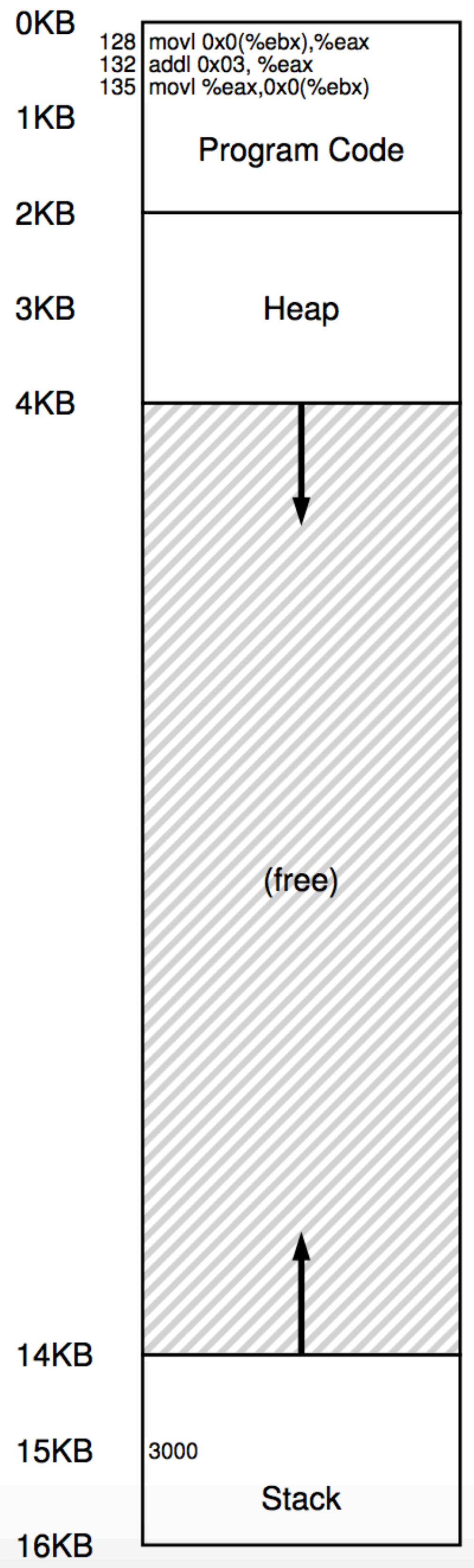
```
128: movl 0x0(%ebx), %eax
132: addl $0x03, %eax
135: movl %eax, 0x0(%ebx)
```

Example

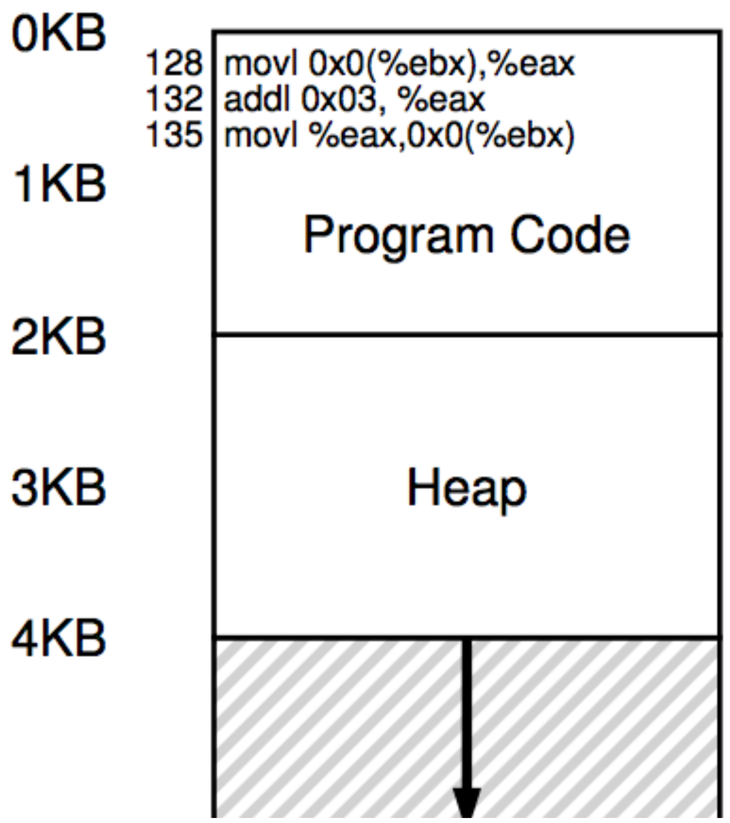


```
128: movl 0x0(%ebx), %eax
132: addl $0x03, %eax
135: movl %eax, 0x0(%ebx)
```

Example


0KB
128 movl 0x0(%ebx),%eax
132 addl 0x03, %eax
135 movl %eax,0x0(%ebx)
1KB
Program Code
2KB
Heap
3KB
4KB
(free)
14KB
15KB
3000
Stack
16KB

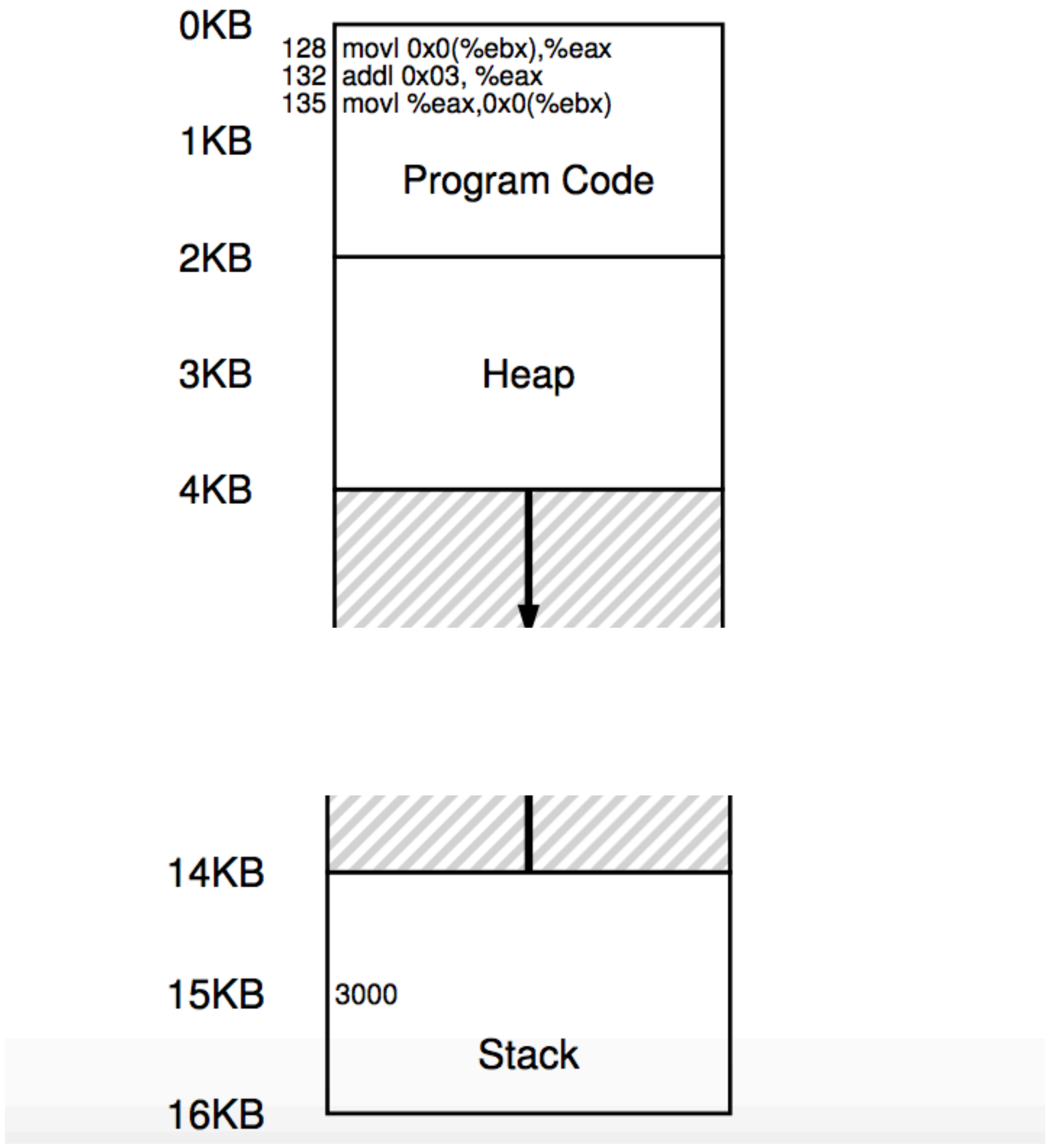
Example



```
128 movl 0x0(%ebx),%eax
132 addl 0x03, %eax
135 movl %eax,0x0(%ebx)
```

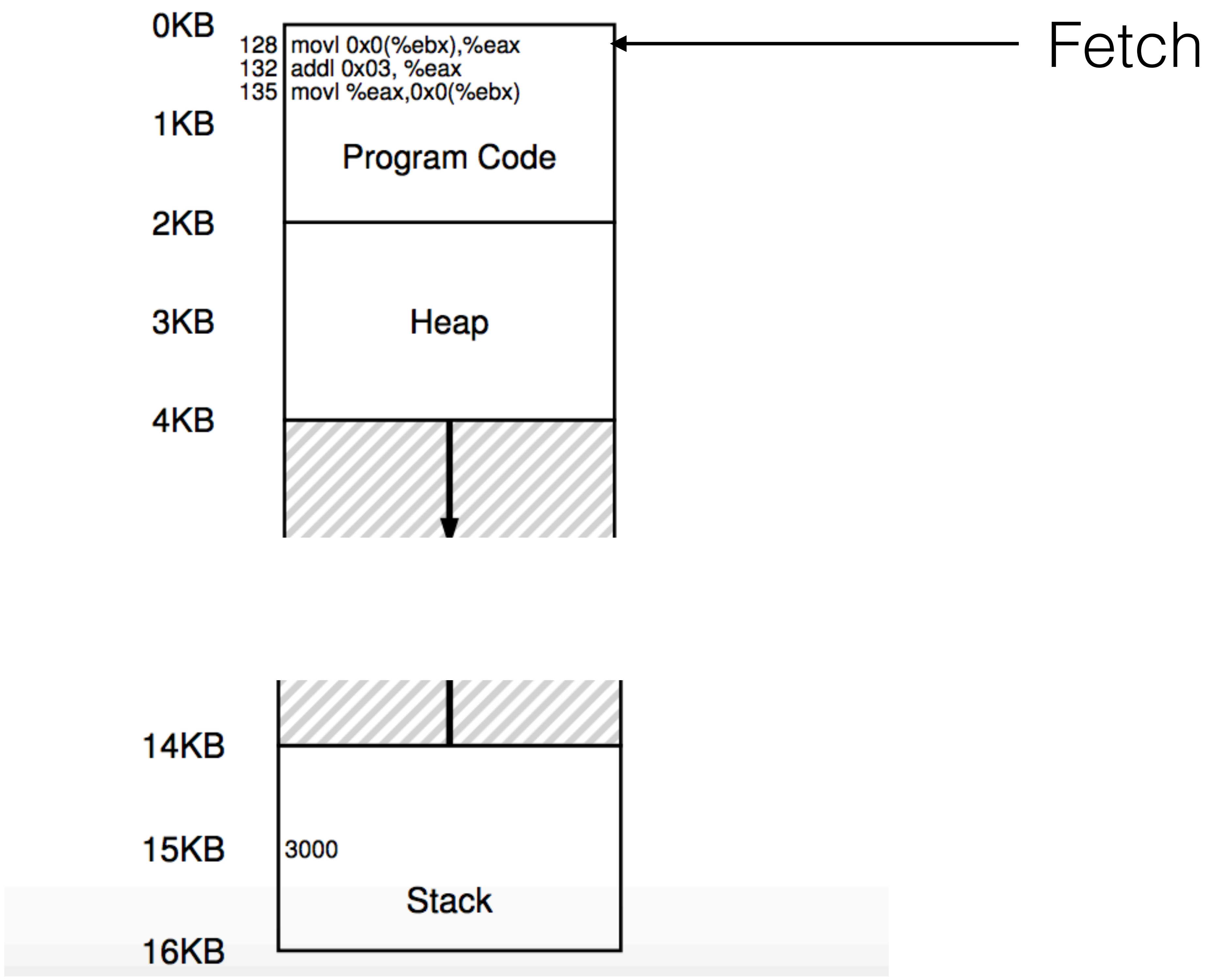
Fetch

Example

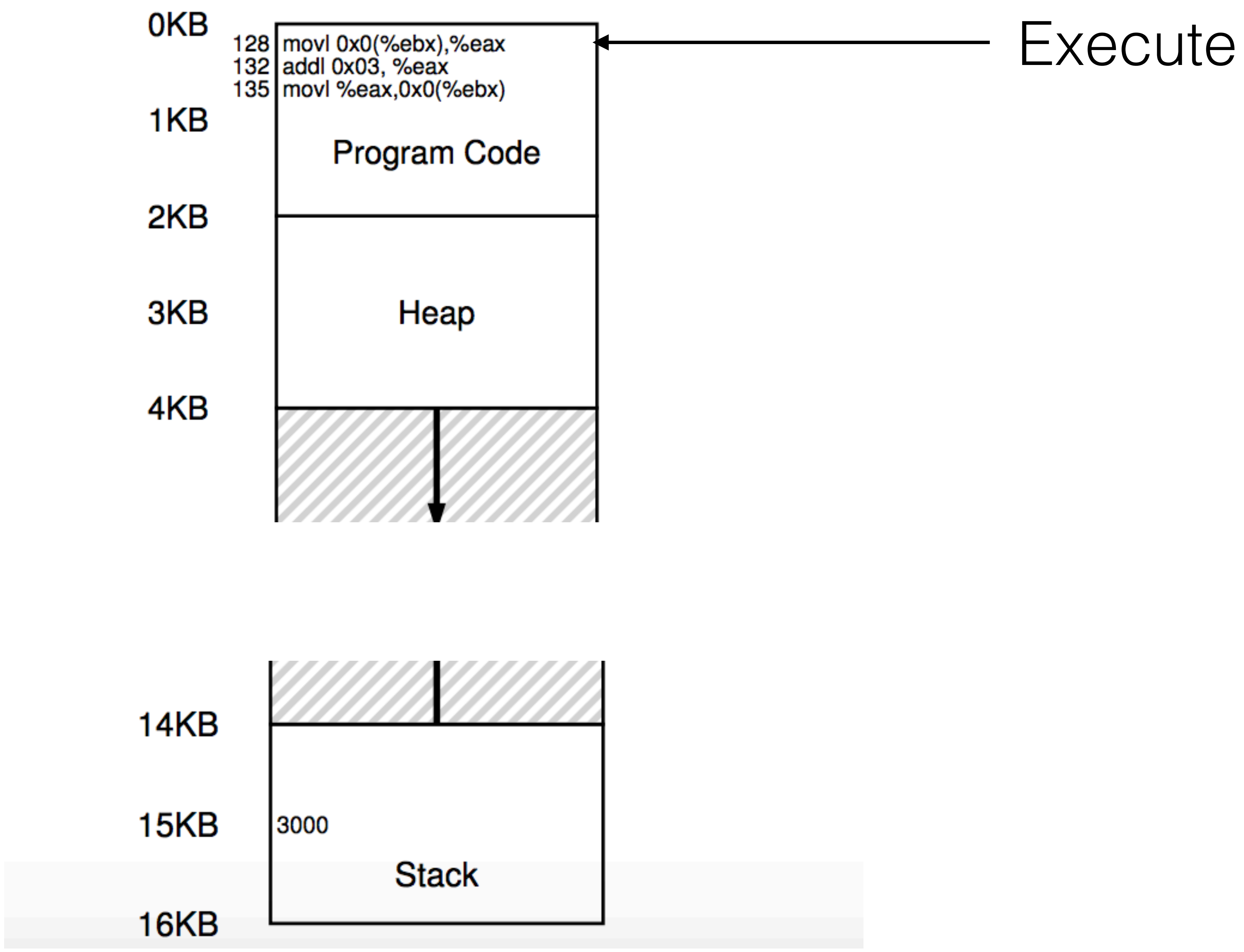


Fetch

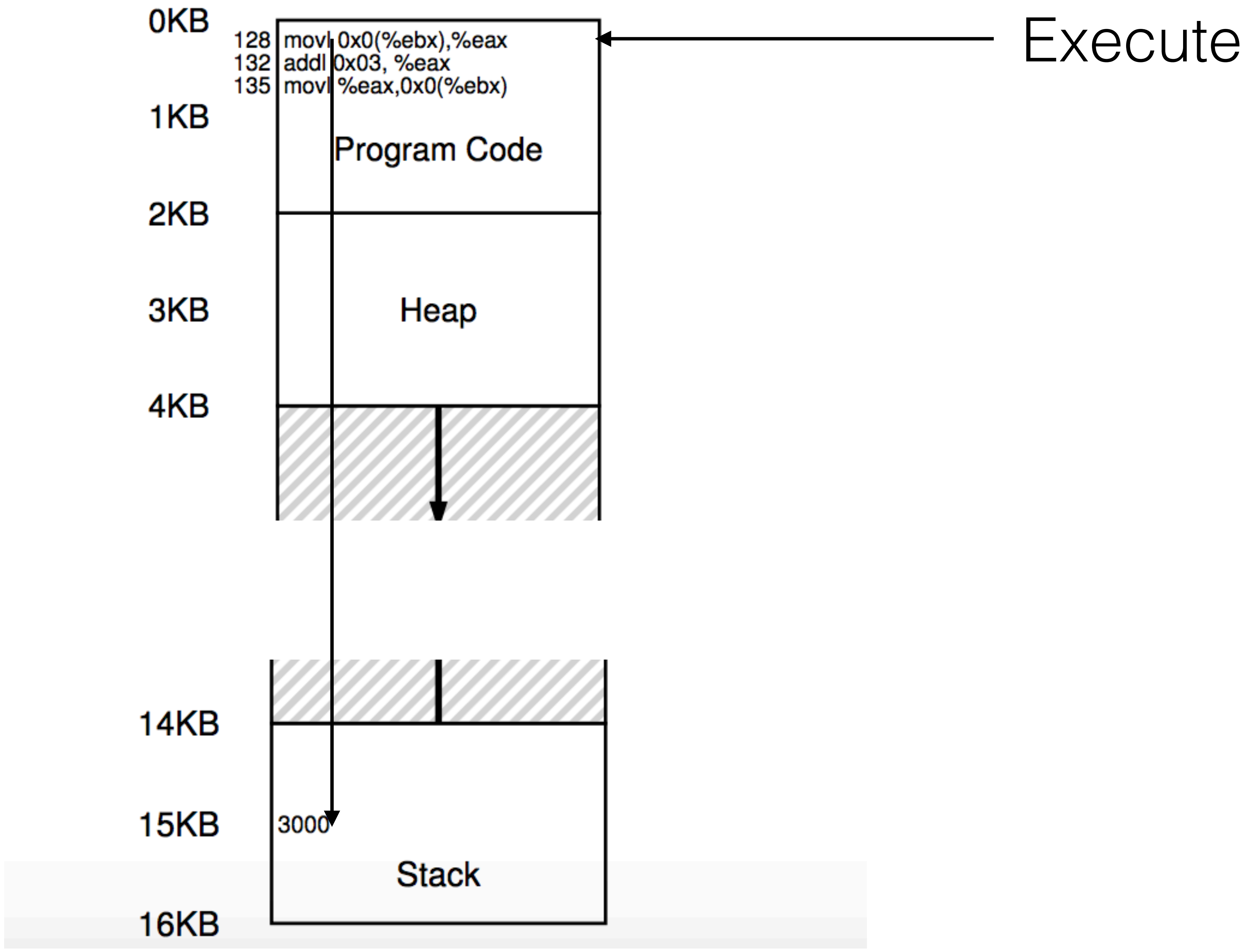
Example


0KB
1KB
2KB
3KB
4KB
14KB
15KB
16KB
128 movl 0x0(%ebx),%eax
132 addl 0x03, %eax
135 movl %eax,0x0(%ebx)
Program Code
Heap
3000
Stack
Fetch

Example

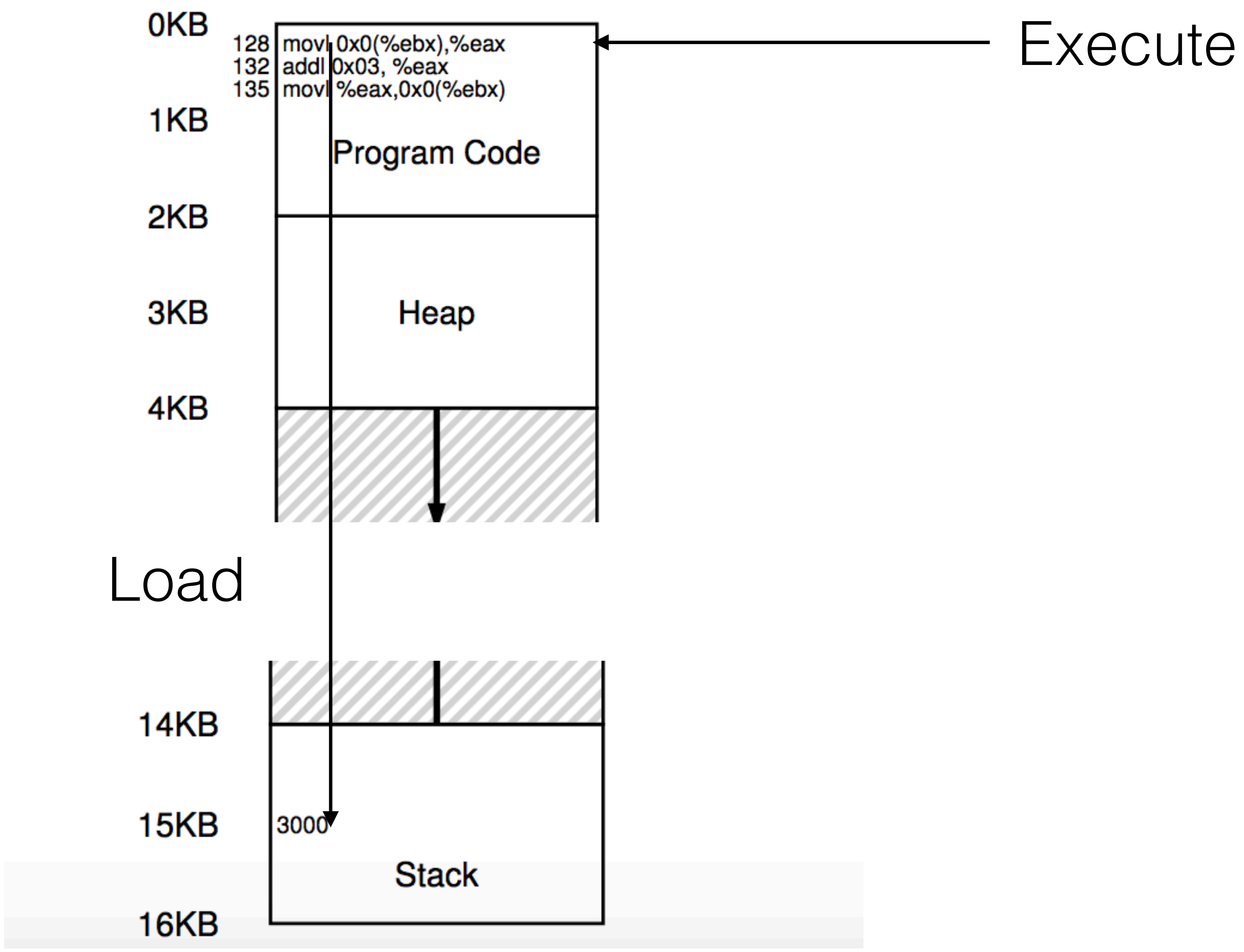

0KB
1KB
2KB
3KB
4KB
14KB
15KB
16KB
128 movl 0x0(%ebx),%eax
132 addl 0x03, %eax
135 movl %eax,0x0(%ebx)
Program Code
Heap
3000
Stack
Execute

Example

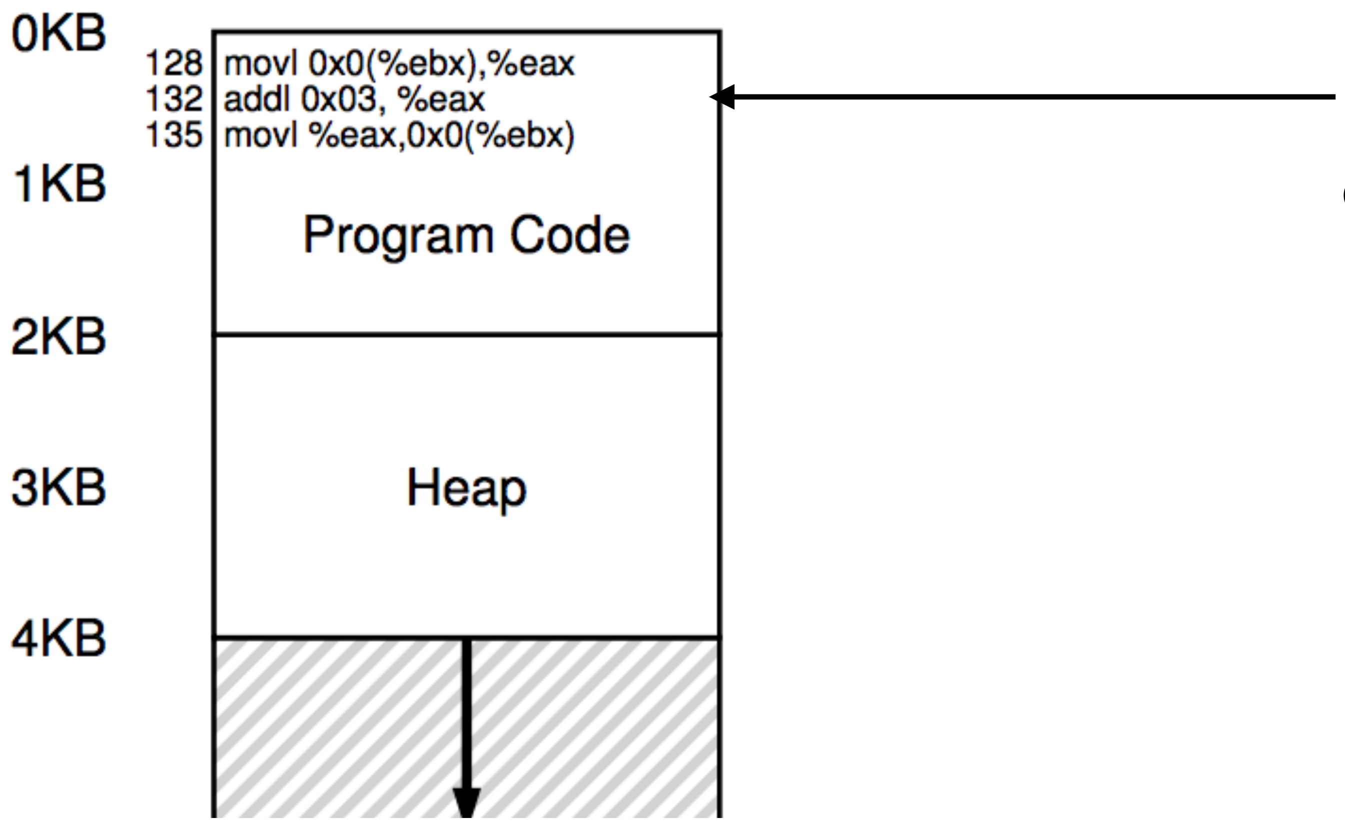


```
128 movl 0x0(%ebx),%eax
132 addl 0x03, %eax
135 movl %eax,0x0(%ebx)
```

Example

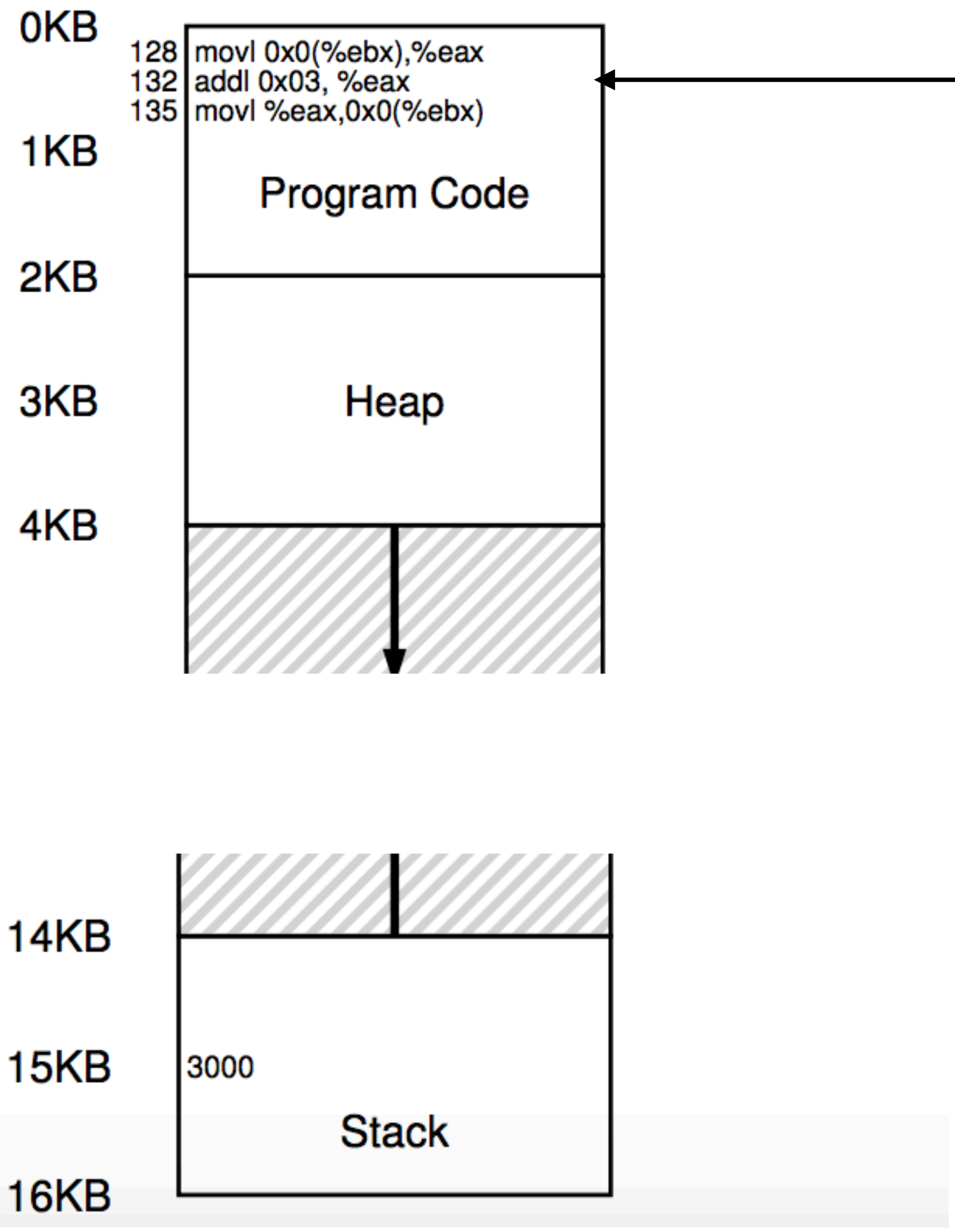

0KB
1KB
2KB
3KB
4KB
14KB
15KB
16KB
128 movl 0x0(%ebx),%eax
132 addl 0x03, %eax
135 movl %eax,0x0(%ebx)
Program Code
Heap
Stack
3000
Execute
Load

Example



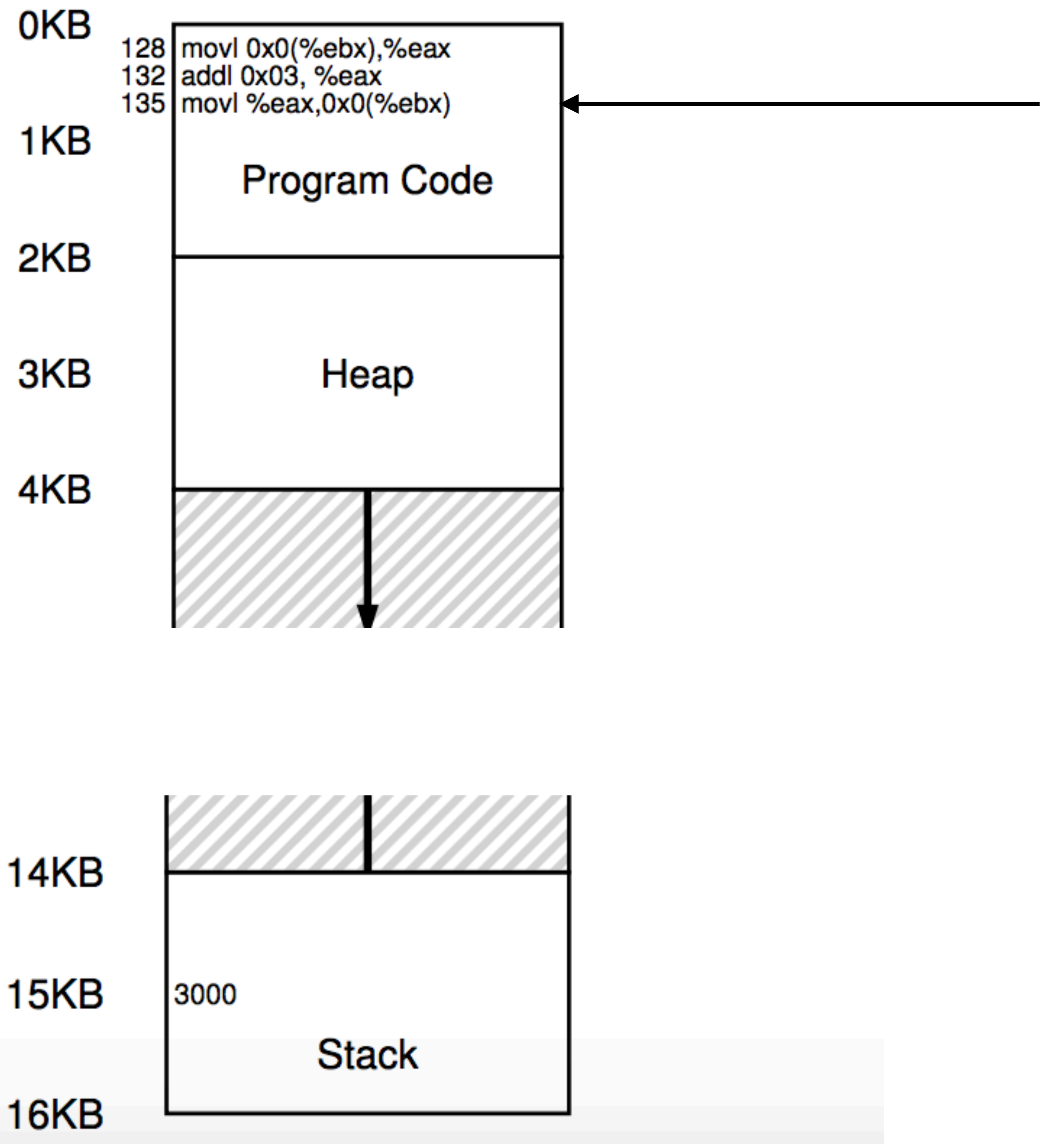
Fetch
& Execute

Example



Fetch
& Execute

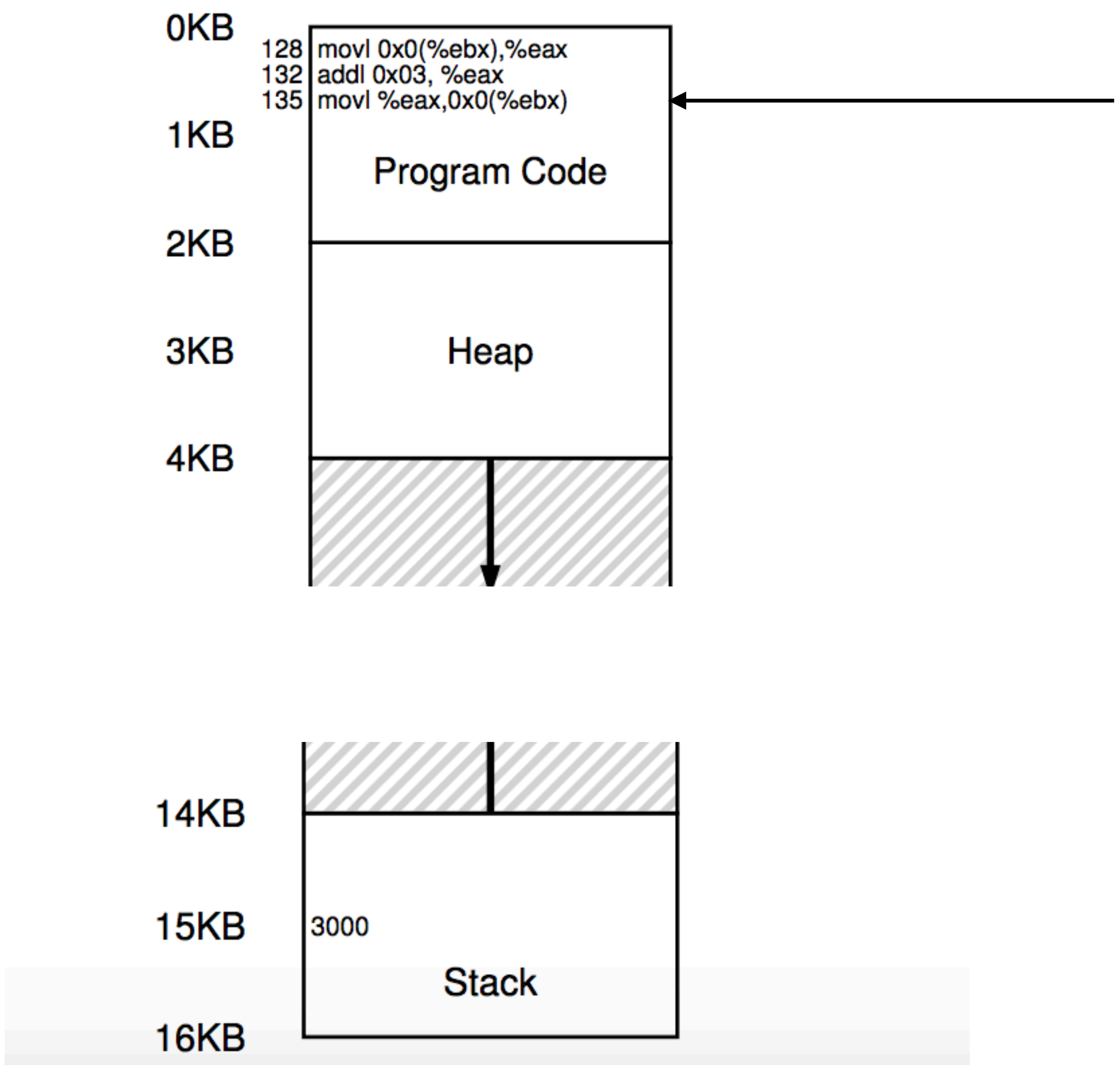
Example



```
128 movl 0x0(%ebx),%eax
132 addl 0x03, %eax
135 movl %eax,0x0(%ebx)
```

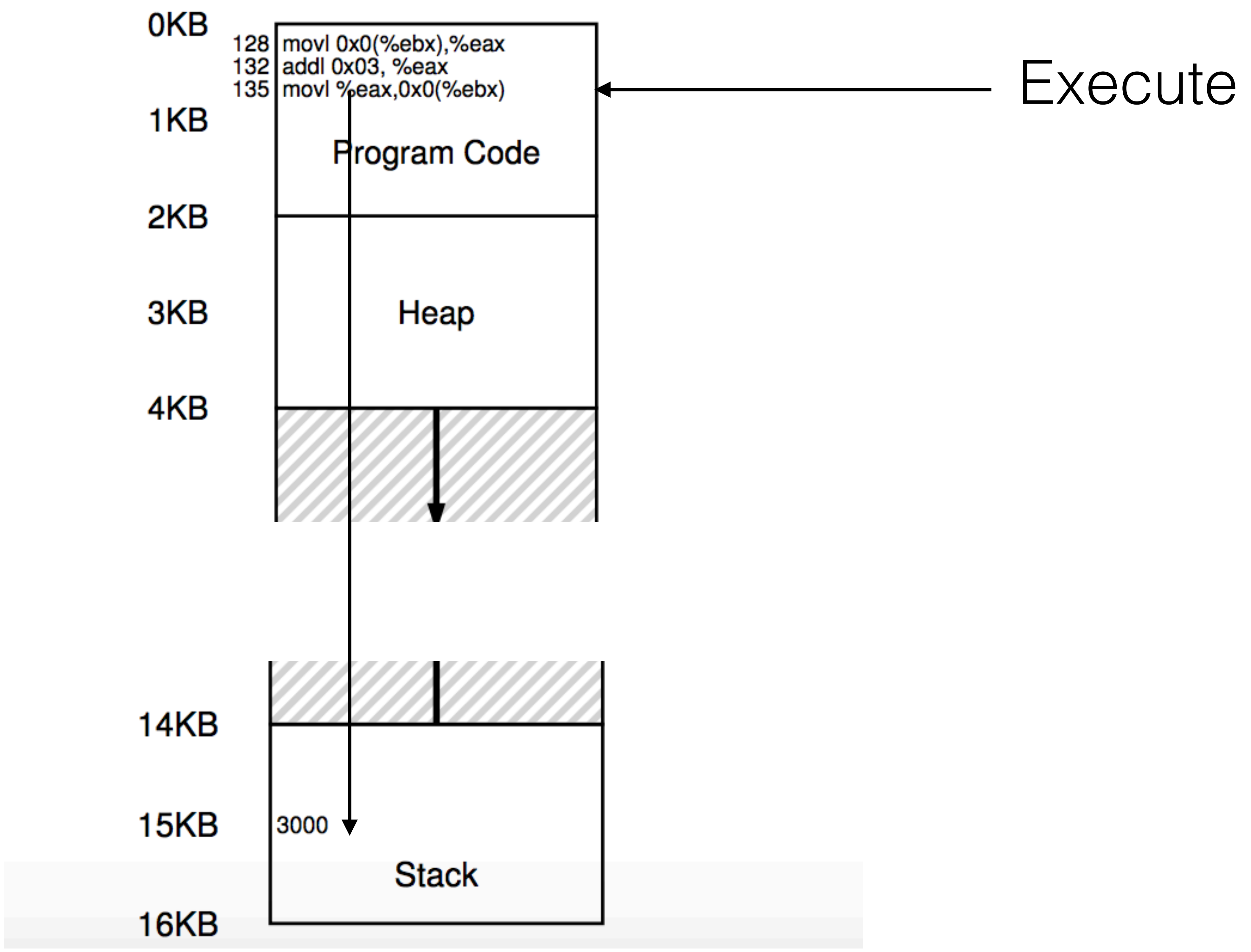
Fetch

Example

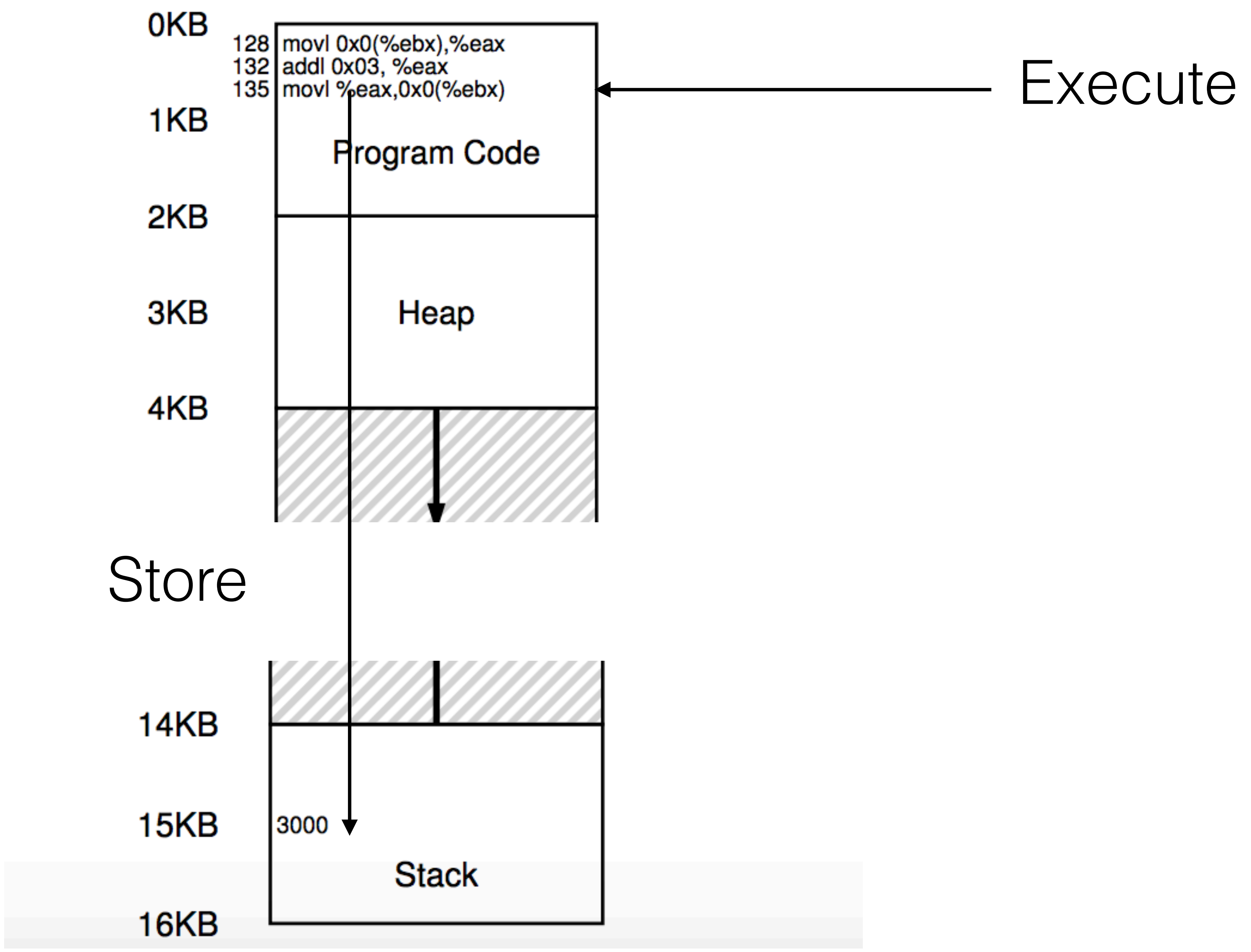

```
128 movl 0x0(%ebx),%eax
132 addl 0x03, %eax
135 movl %eax,0x0(%ebx)
```

Execute

Example

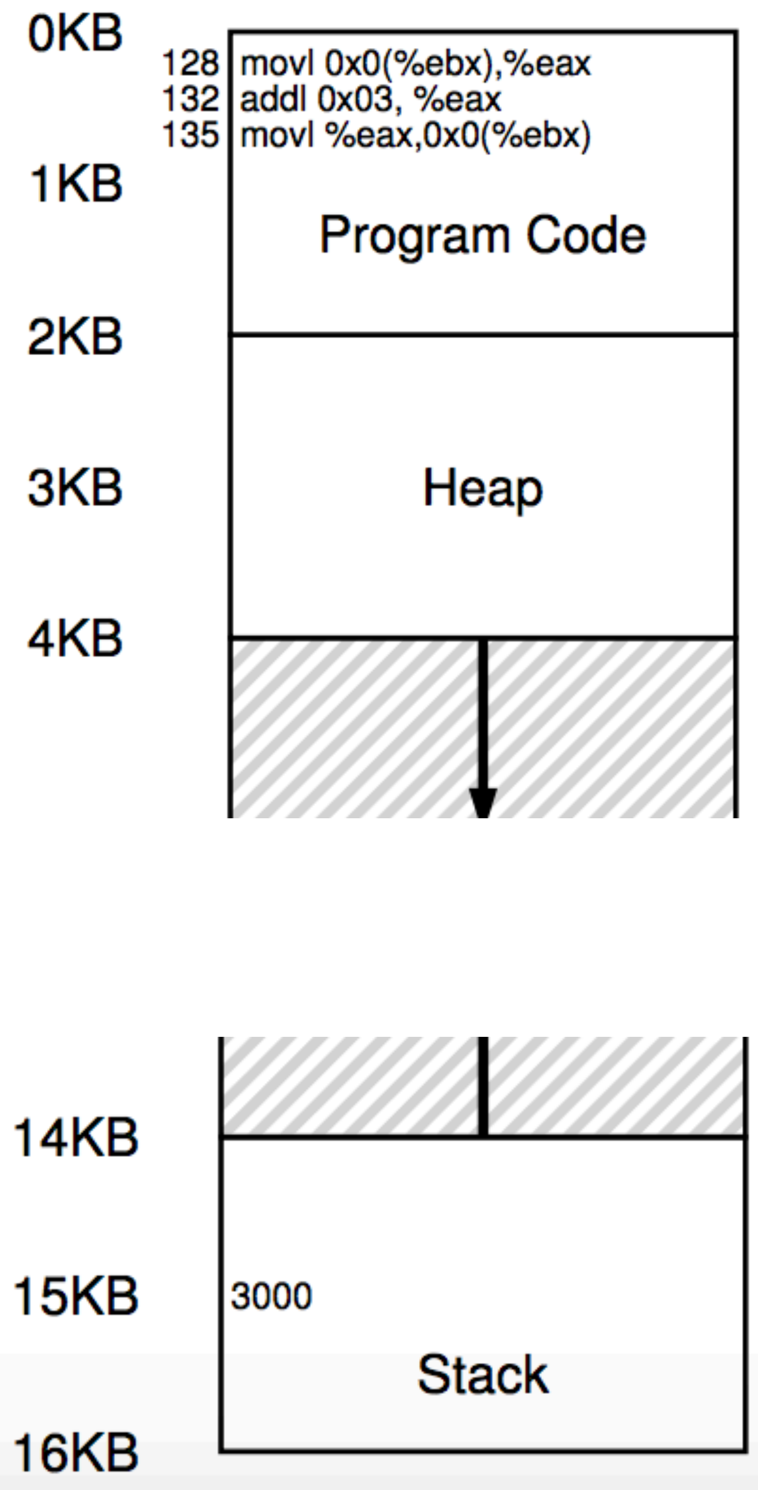

0KB
1KB
2KB
3KB
4KB
14KB
15KB
16KB
128 movl 0x0(%ebx),%eax
132 addl 0x03, %eax
135 movl %eax,0x0(%ebx)
Program Code
Heap
3000
Stack
Execute

Example



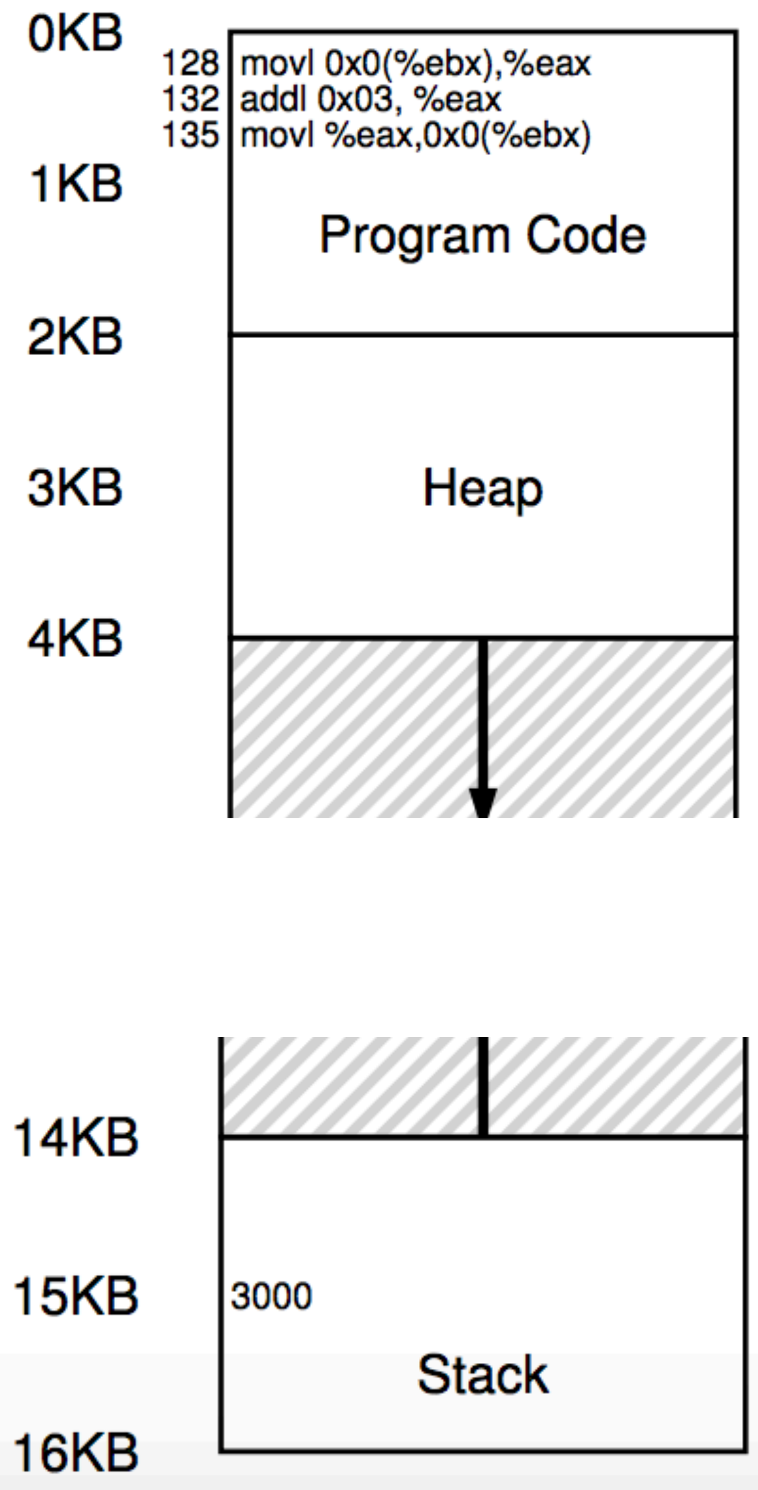
0KB
1KB
2KB
3KB
4KB
14KB
15KB
16KB
128 movl 0x0(%ebx),%eax
132 addl 0x03, %eax
135 movl %eax,0x0(%ebx)
Program Code
Heap
Execute
Store
3000
Stack

Pop Quiz



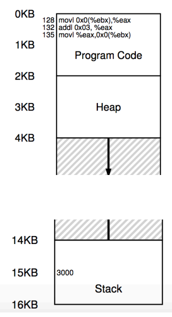
Do all process start and end from 0 KB and 16 KB?

Relocation



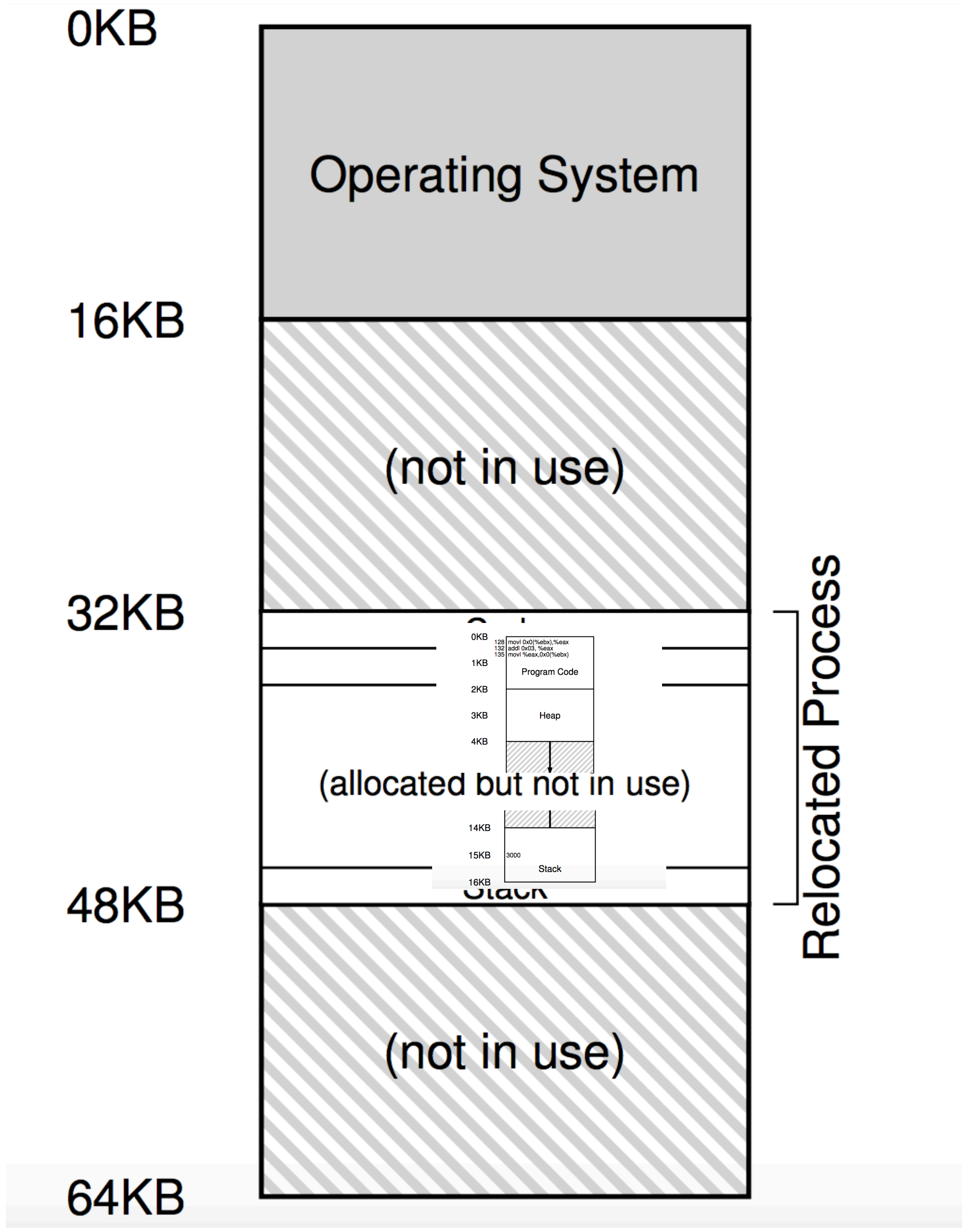
0KB
128 movl 0x0(%ebx),%eax
132 addl 0x03, %eax
135 movl %eax,0x0(%ebx)
1KB
Program Code
2KB
3KB
Heap
4KB
14KB
15KB
3000
Stack
16KB

Relocation



```
128 movl 0x0(%ebx),%eax
132 addl 0x03, %eax
135 movl %eax,0x0(%ebx)
```

Relocation



0KB
Operating System
16KB
(not in use)
32KB
0KB
128 movl 0x0(%ebx),%eax
132 addl 0x03, %eax
135 movl %eax,0x0(%ebx)
1KB
Program Code
2KB
3KB
Heap
4KB
(allocated but not in use)
14KB
15KB
3000
Stack
16KB
Stack
48KB
(not in use)
64KB
Relocated Process

General Address Translation

Kernel

CPU

MMU

Physical Memory

General Address Translation

Kernel

CPU

MMU

Physical Memory

General Address Translation

Kernel

CPU

MMU

Physical Memory

Virtual Address

General Address Translation

Kernel

CPU

MMU

Physical Memory

Virtual Address

0x10102030

General Address Translation

Kernel

CPU

MMU

Physical Memory

Virtual Address
0x10102030

General Address Translation

Kernel

CPU

MMU

Physical Memory

Virtual Address

0x10102030

$$\begin{tikzcd} & \text{Kernel} & \\ \text{CPU} & \text{MMU} \arrow[u] & \text{Physical Memory} \\ \text{Virtual Address 0x10102030} \arrow[ur] & & \end{tikzcd}$$

General Address Translation

$$\begin{tikzcd}
 & \text{Kernel} \arrow[d, bend left] & \\
\text{CPU} & \text{MMU} \arrow[u, bend left] & \text{Physical Memory} \\
\text{Virtual Address } \texttt{0x10102030} \arrow[ur] & &
\end{tikzcd}$$

General Address Translation

$$\begin{tikzcd}
 & \text{Kernel} \arrow[d, bend left] & \\
\text{CPU} & \text{MMU} \arrow[u, bend left] & \text{Physical Memory} \\
\text{Virtual Address 0x10102030} \arrow[ur] & \text{Physical Address} &
\end{tikzcd}$$

General Address Translation

Kernel

CPU

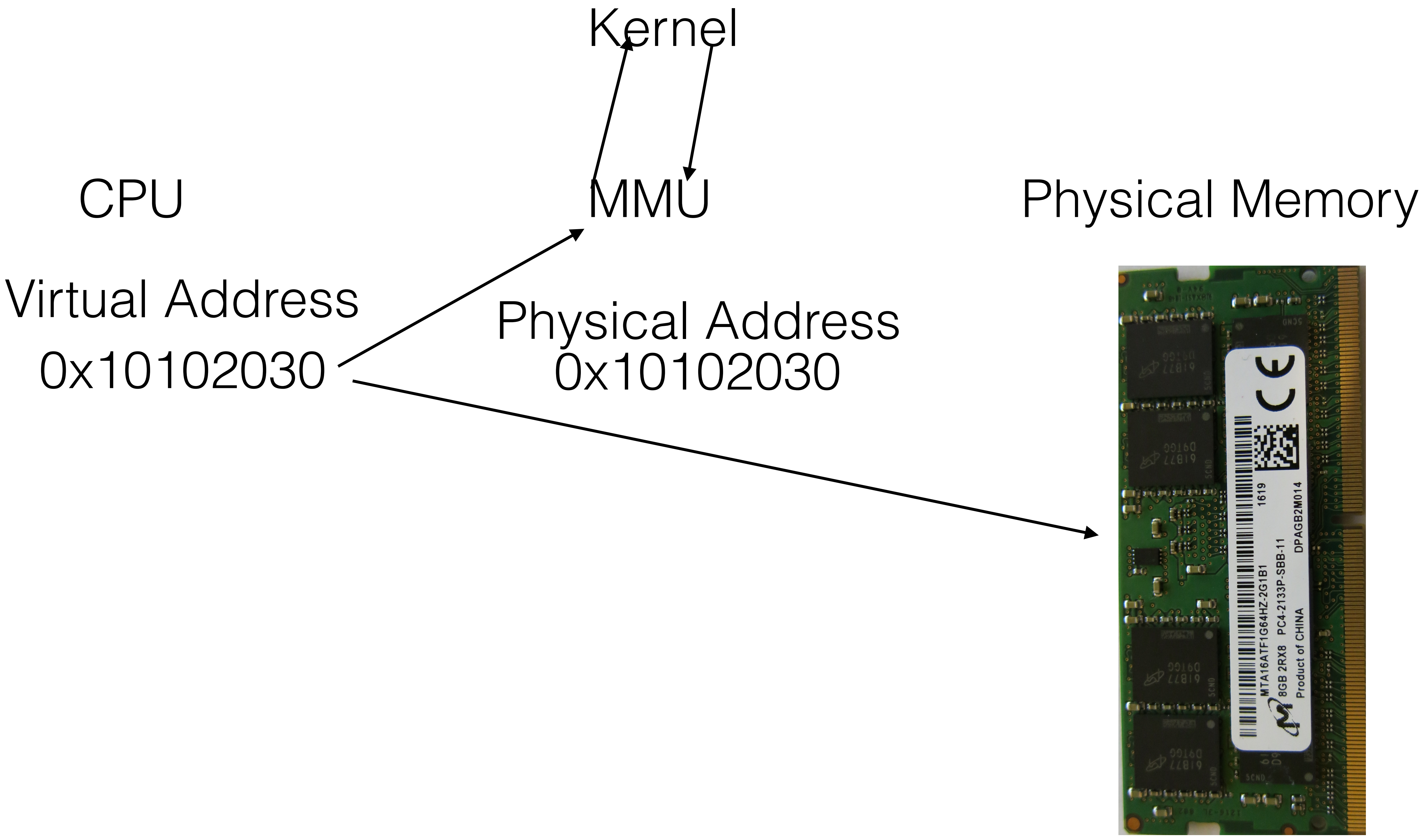
MMU

Physical Memory

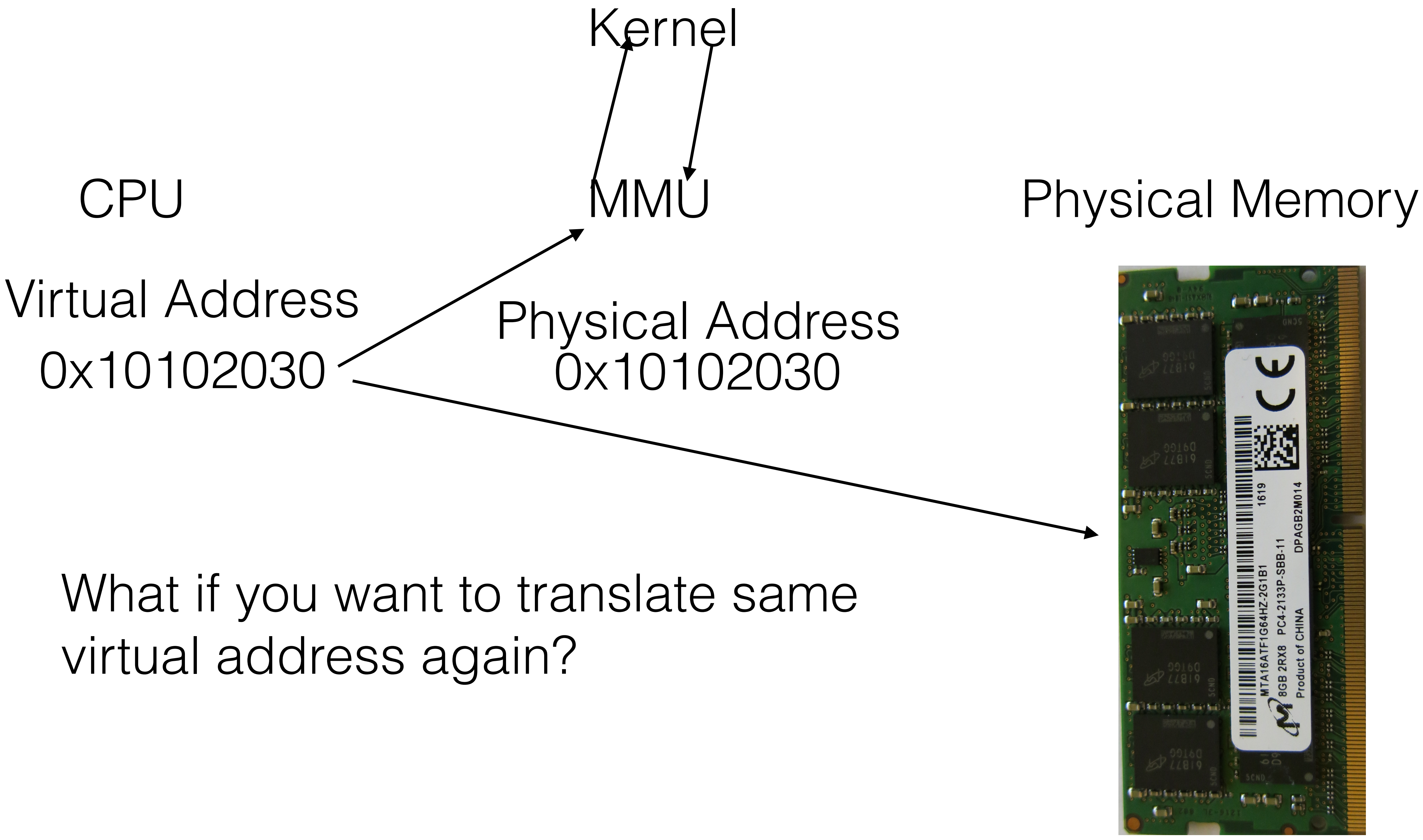
Virtual Address
0x10102030

Physical Address
0x10102030

General Address Translation

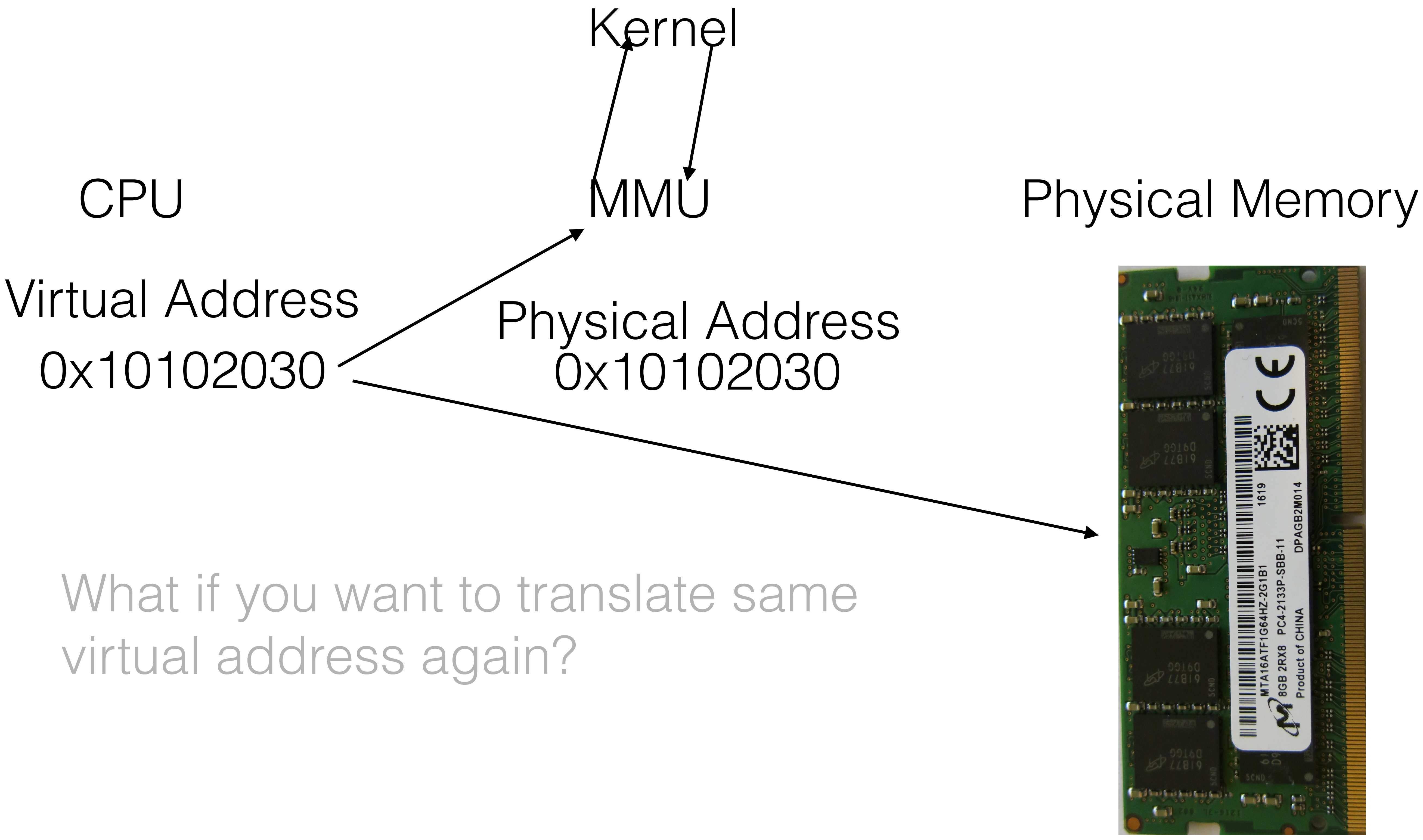


General Address Translation



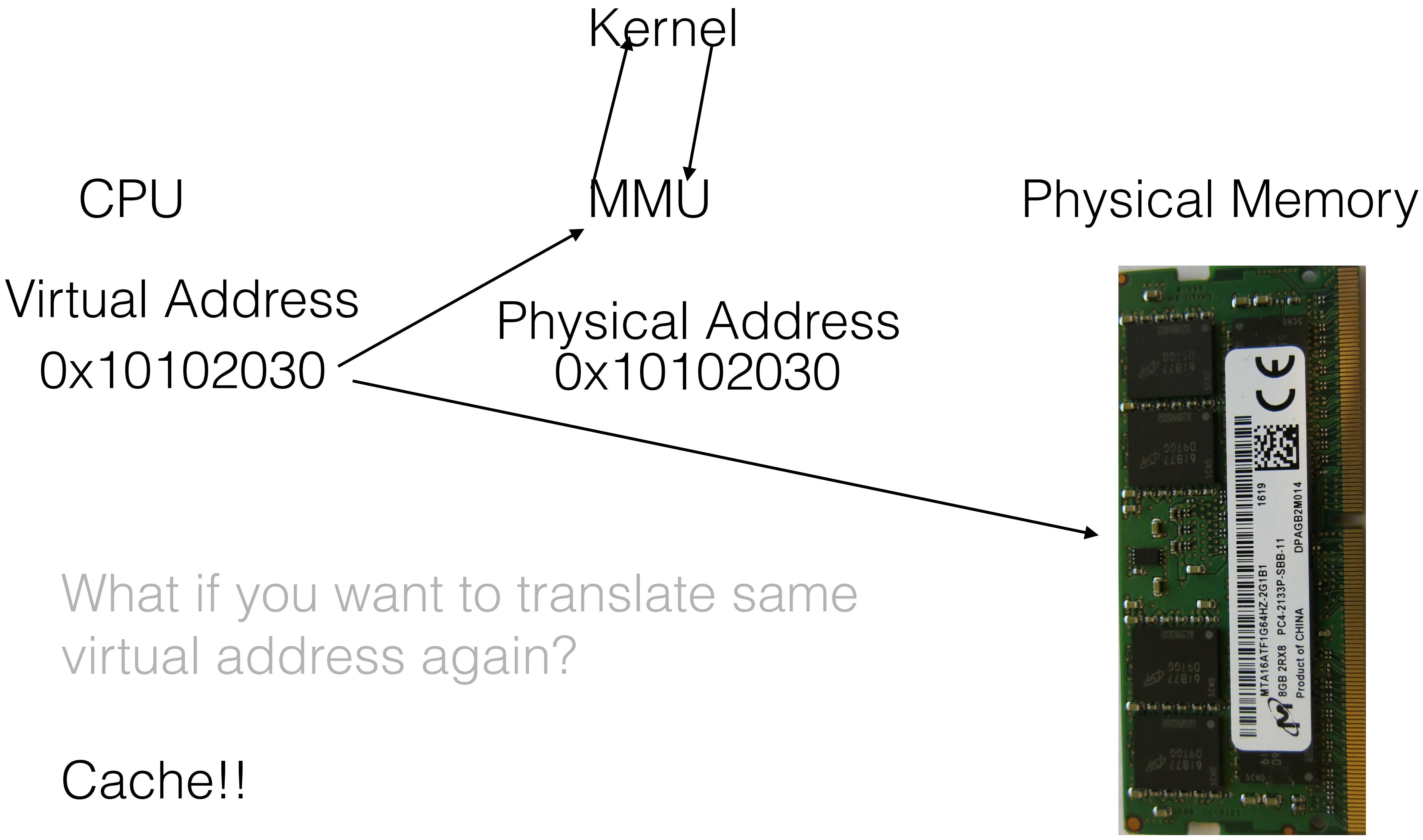
What if you want to translate same virtual address again?

General Address Translation



What if you want to translate same virtual address again?

General Address Translation



What if you want to translate same virtual address again?

Cache!!

General Address Translation

Kernel

CPU

MMU

Physical Memory

Virtual Address
0x10102030

Physical Address
0x10102030

What do you do with cache if there is a context switch?