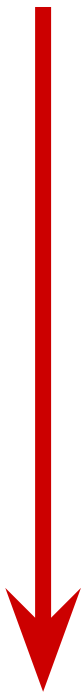
Pelican eel

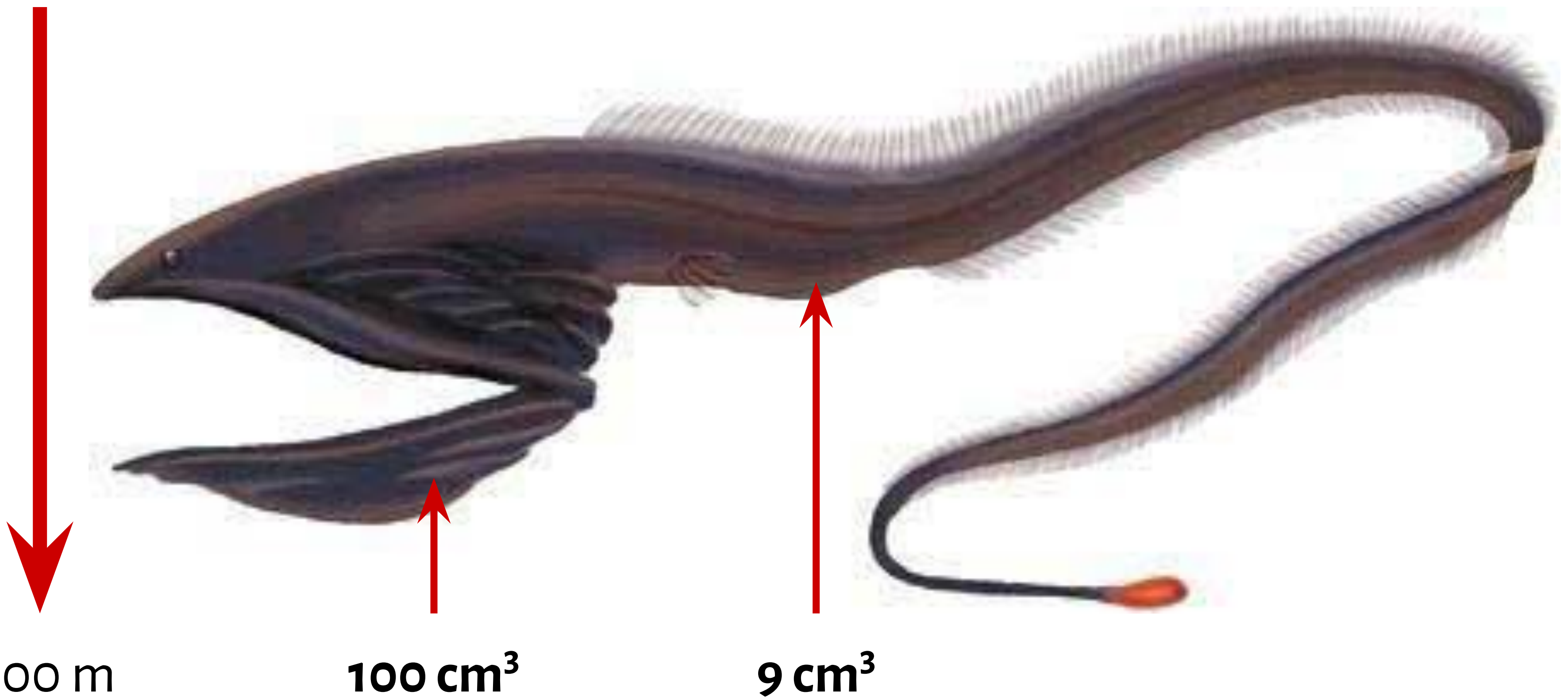
Połykacz z rodziny gardzielcokształtnych



3000 m

Pelican eel

Połykacz z rodziny gardzielcokształtnych



Pelican eel

Połykacz z rodziny gardzielcokształtnych

Pelican eel

Połykacz z rodziny gardzielcokształtnych

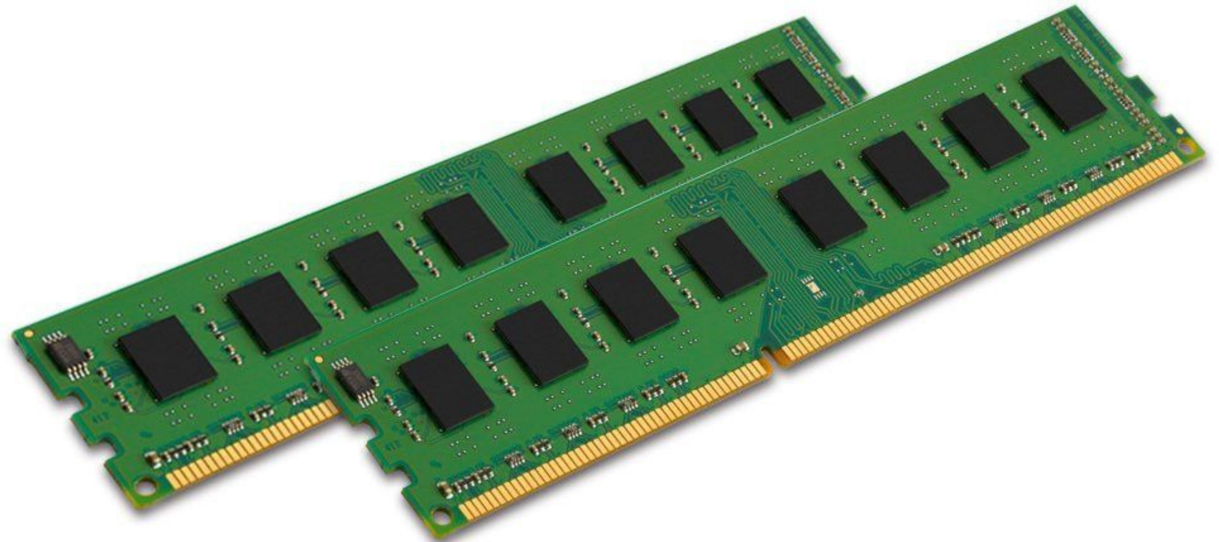
Virtual memory

Addresses

```
mov di, 0x500
```

Addresses

```
mov di, 0x500
```



Memory address (i386)

logical
address

physical
address

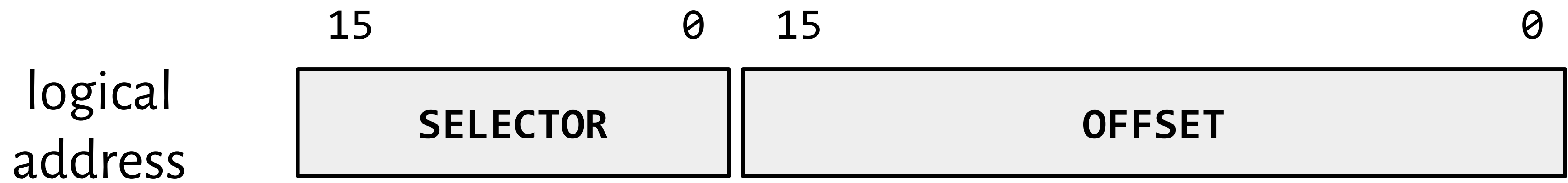
Memory address (i386)

logical
address

linear
address

physical
address

Logical address

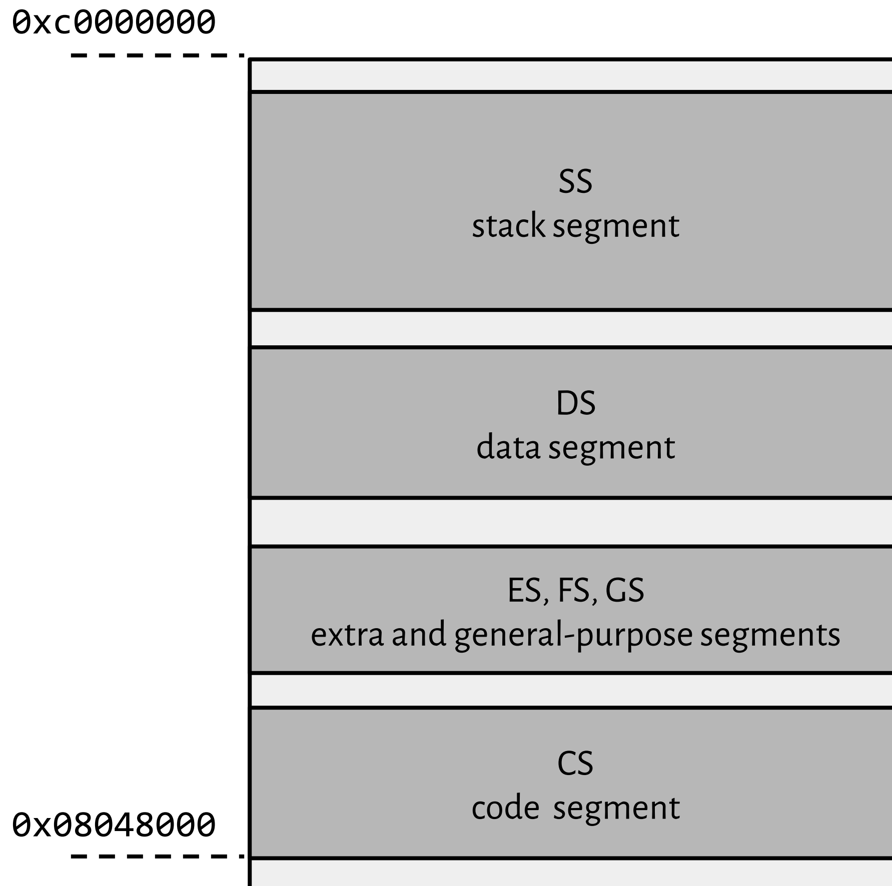


Memory

What processes *think* memory is like:

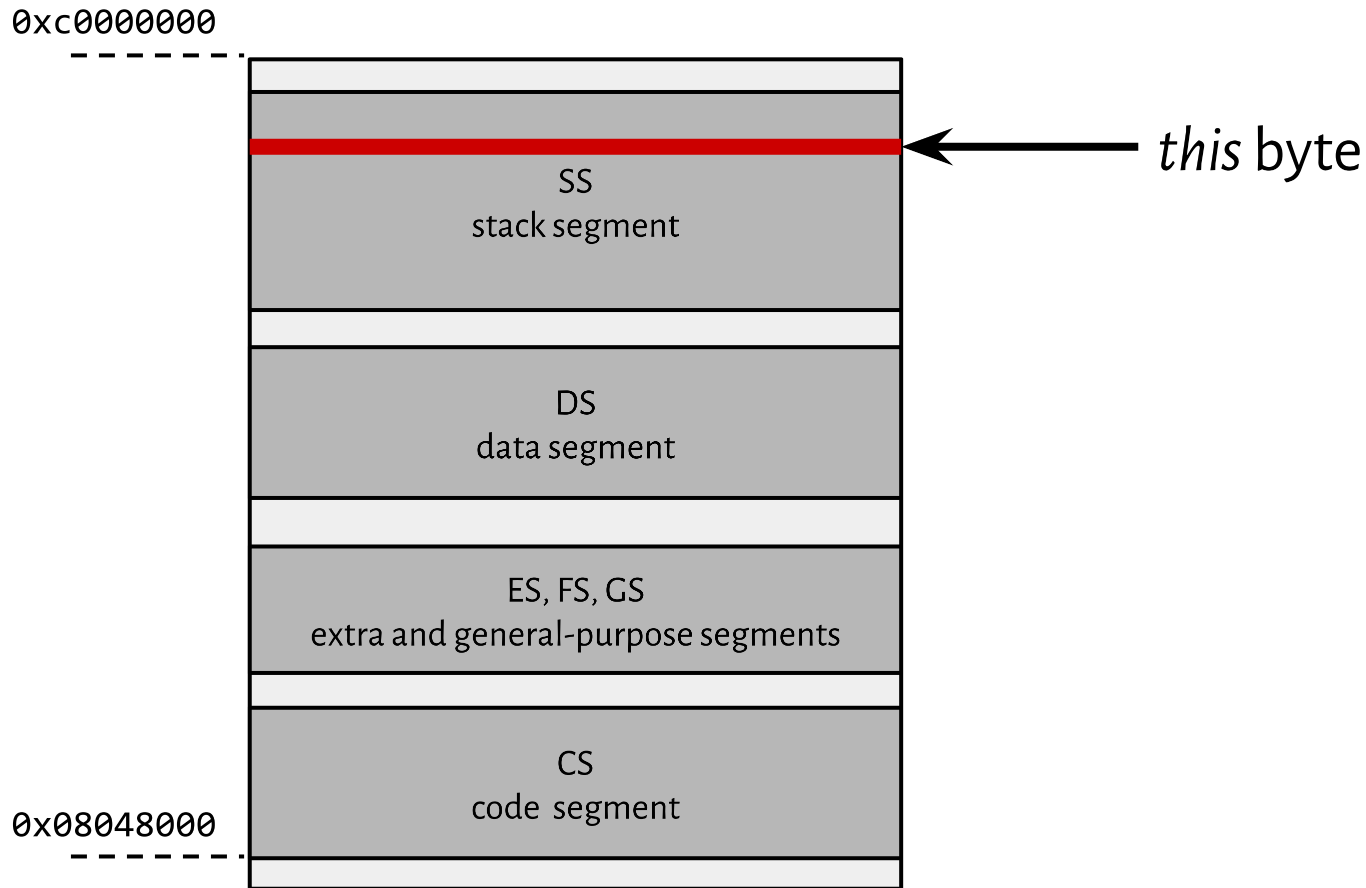
Memory

What processes *think* memory is like:



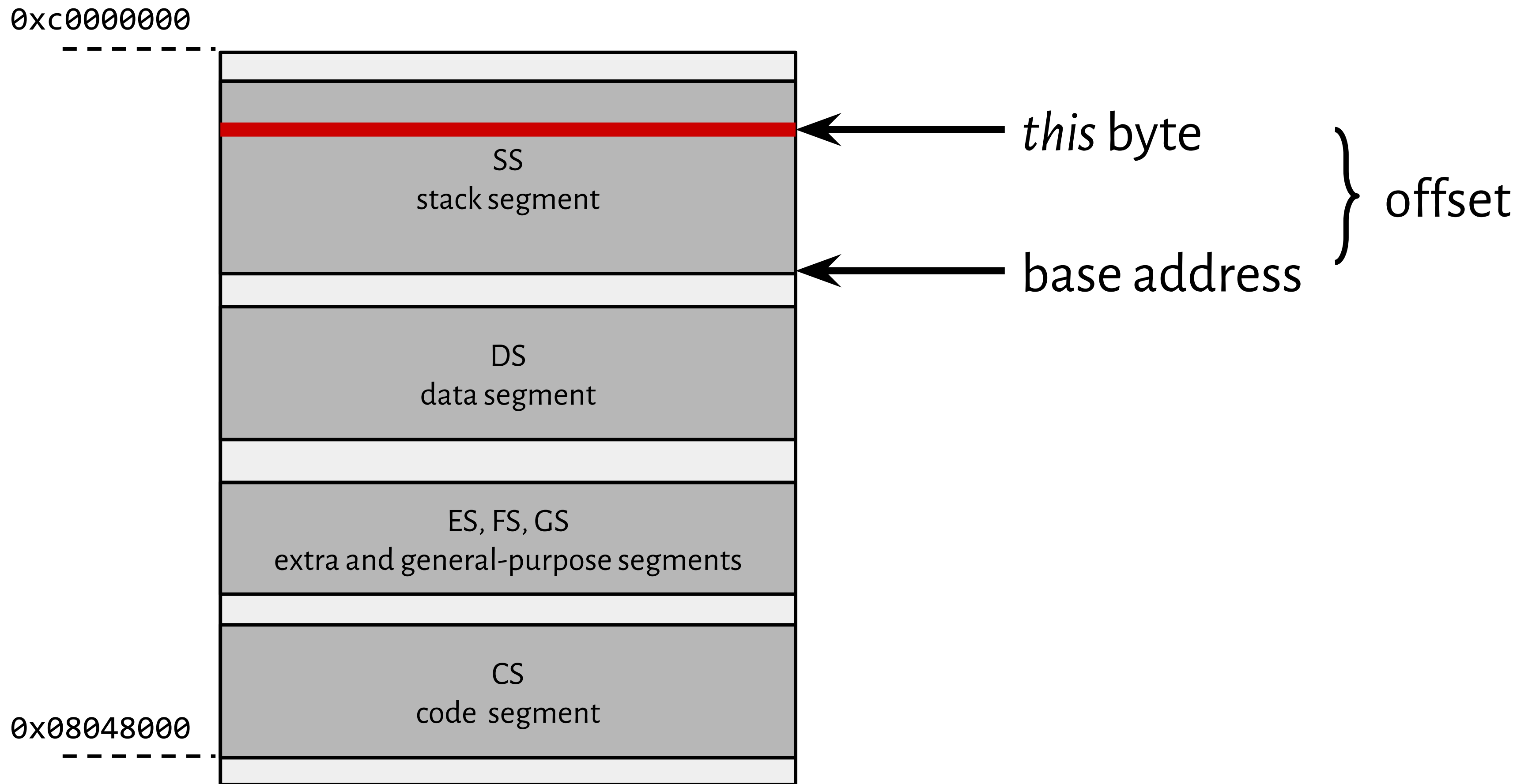
Memory

What processes *think* memory is like:

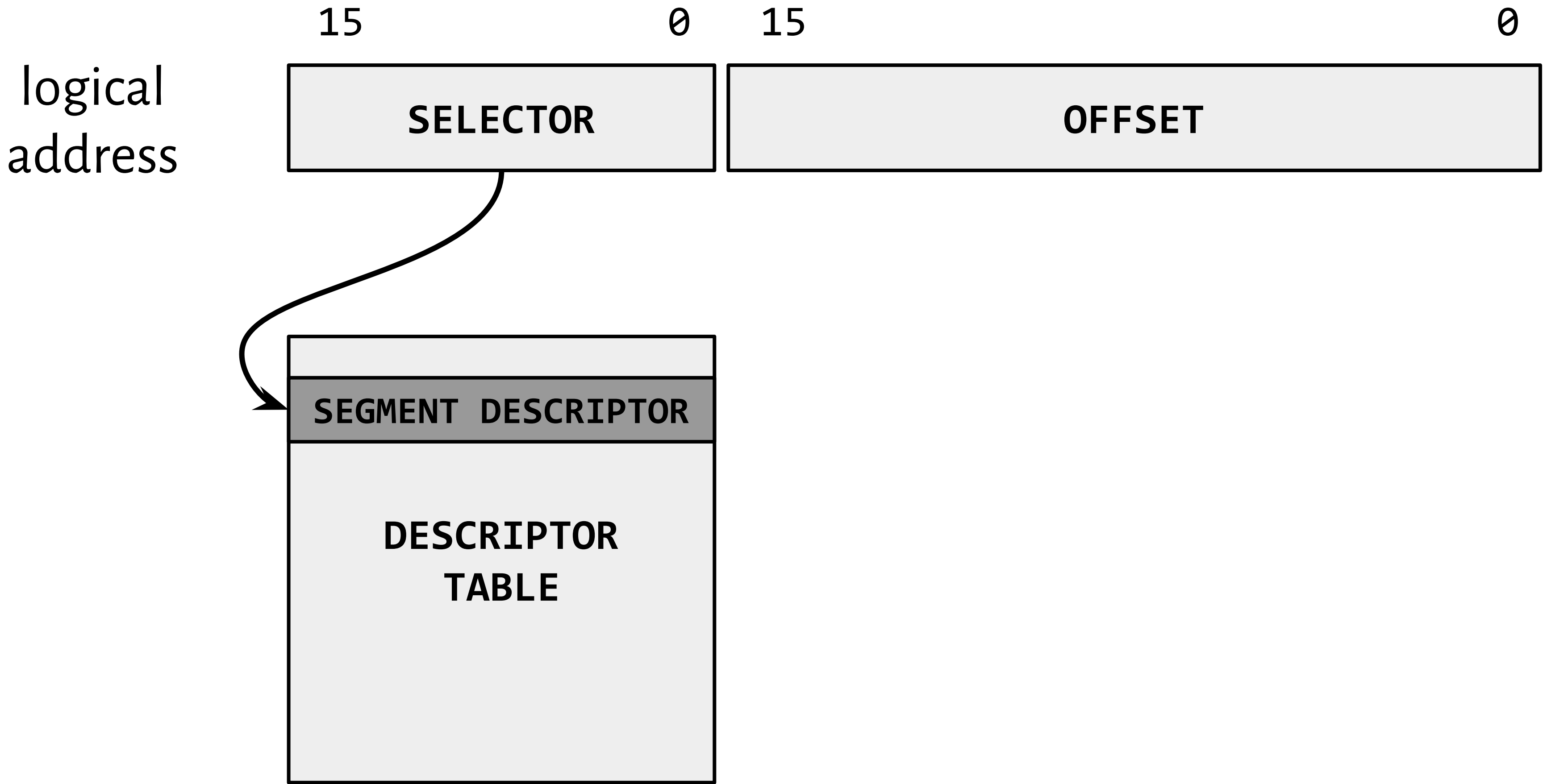


Memory

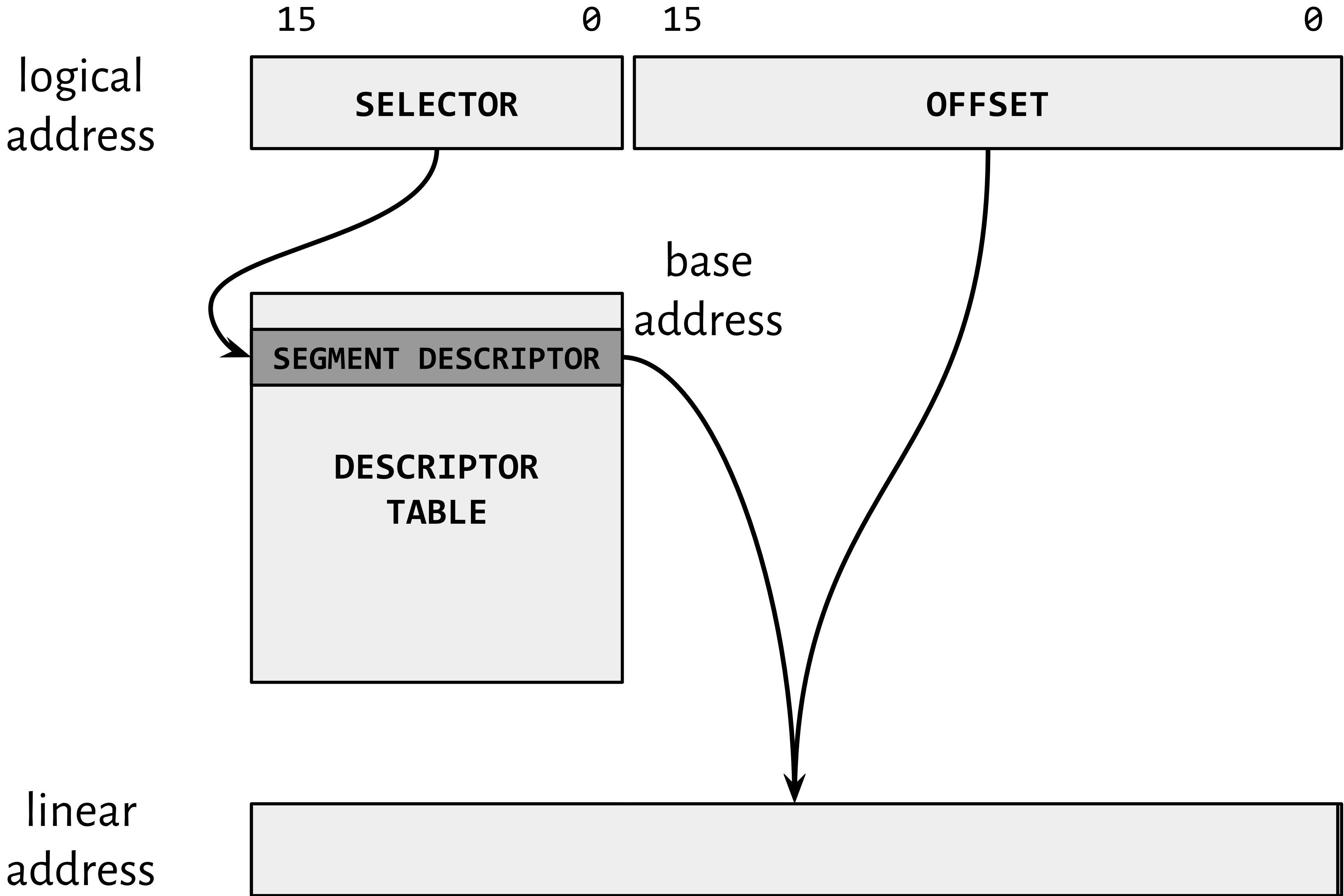
What processes *think* memory is like:



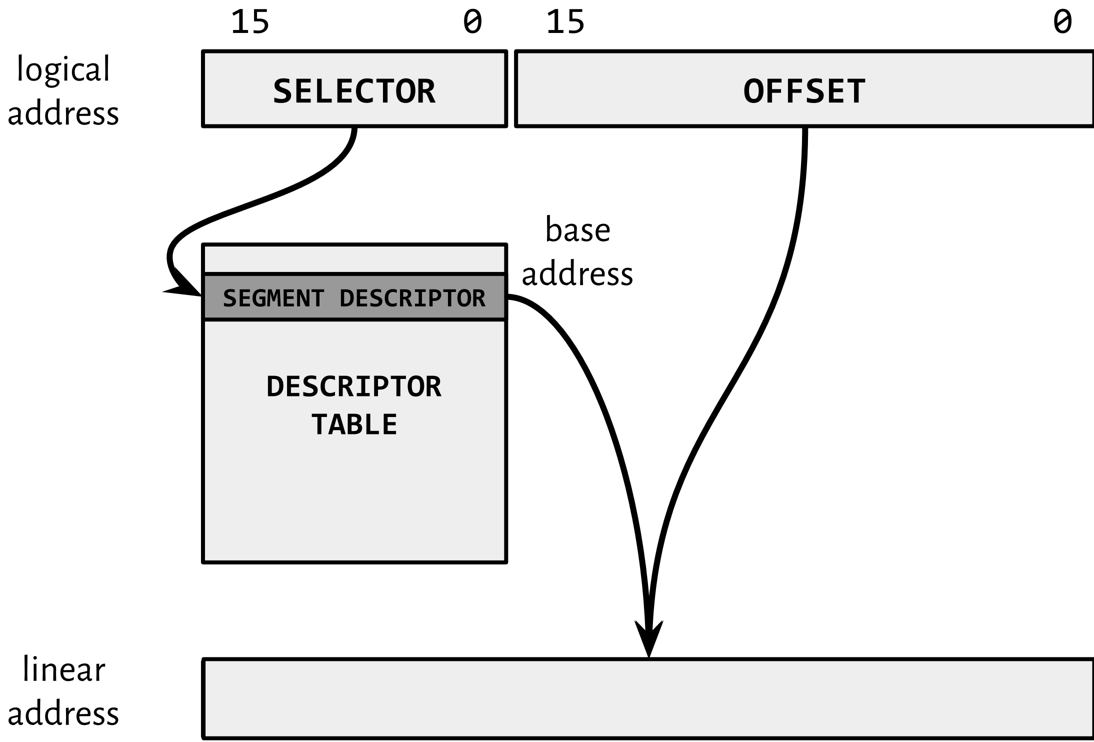
Segments



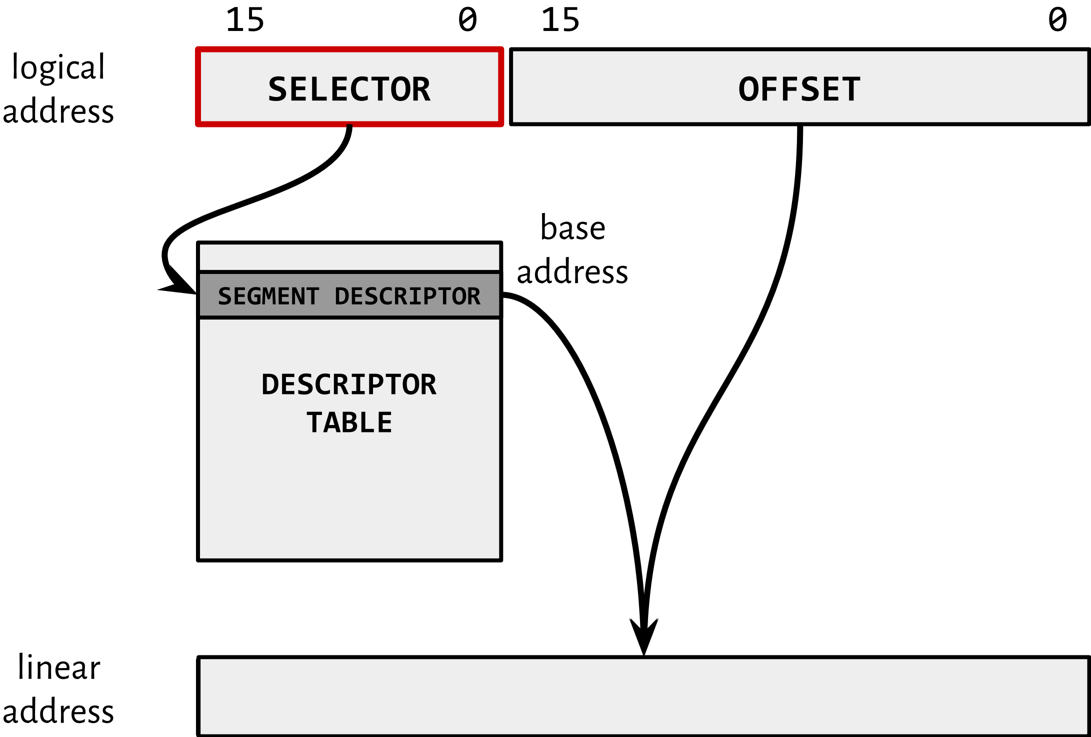
Segments



Segments

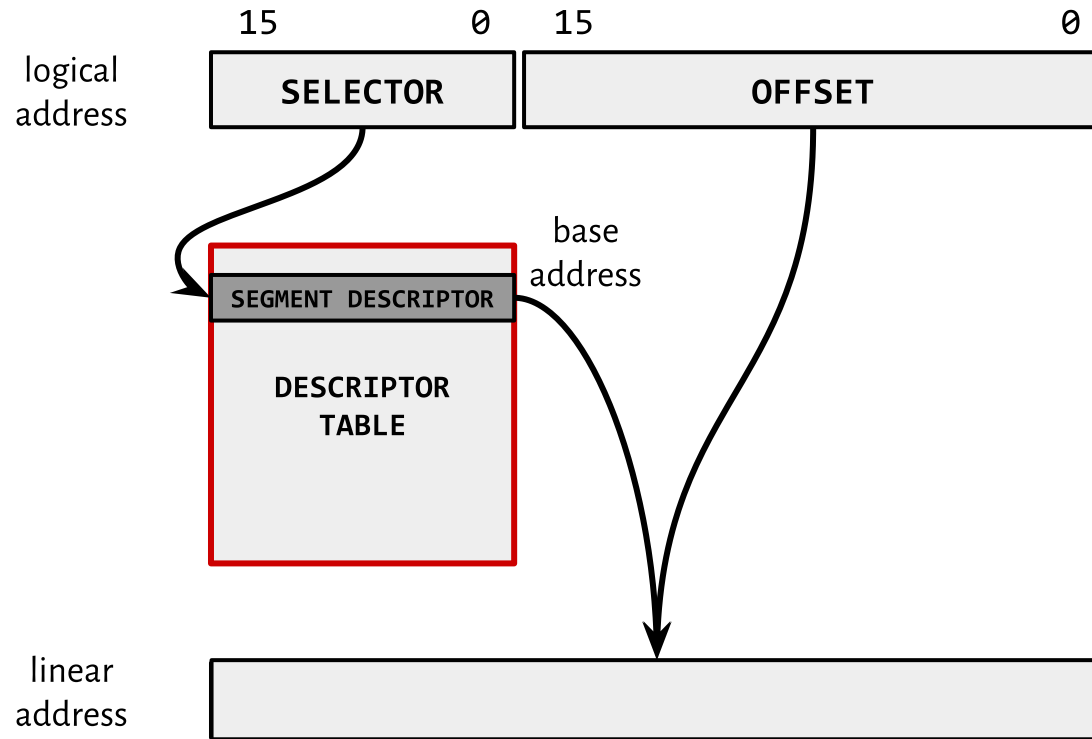


Selector



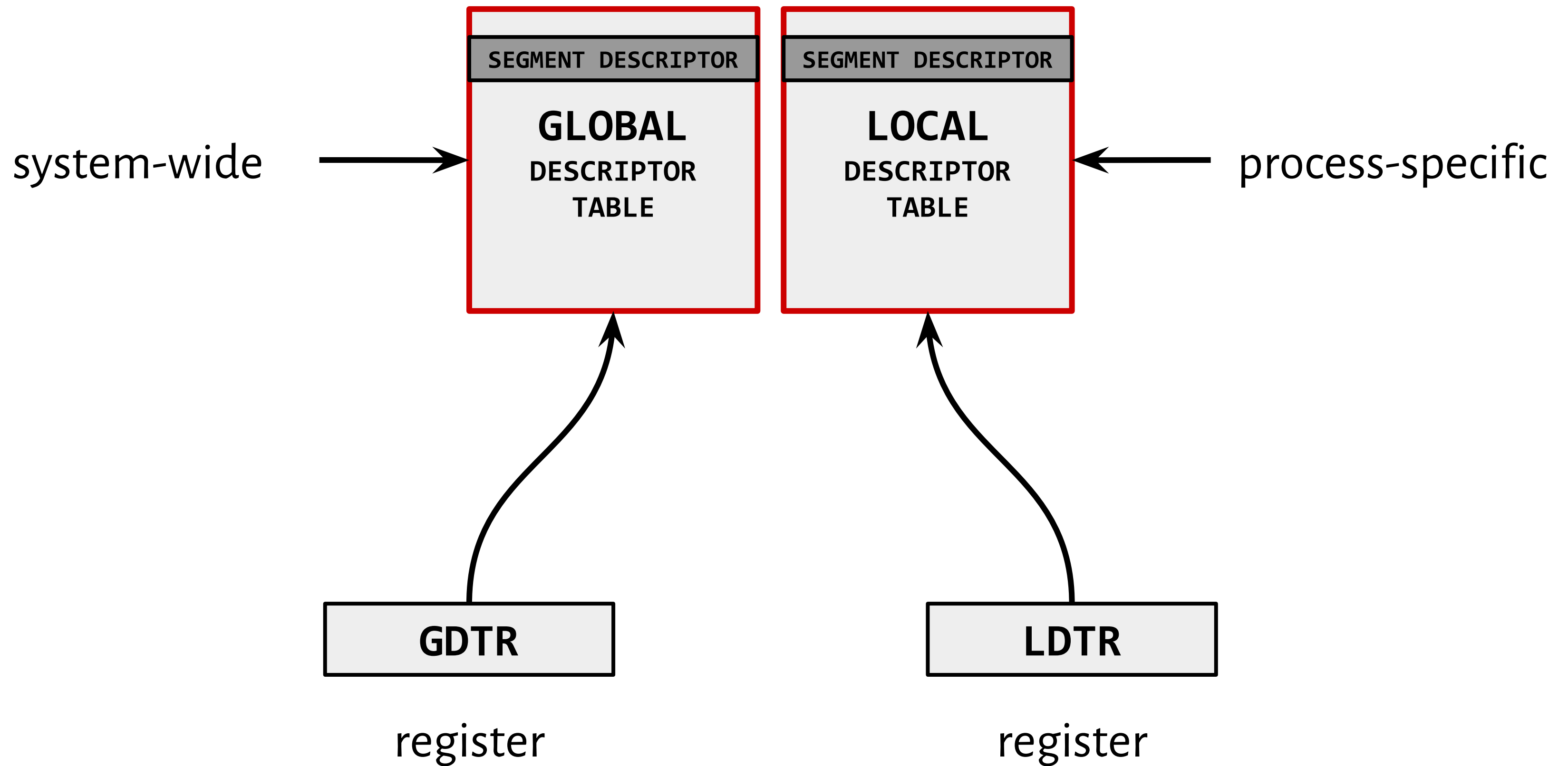
http://www.scs.stanford.edu/05au-cs240c/lab/i386/s05_01.htm

Descriptor table

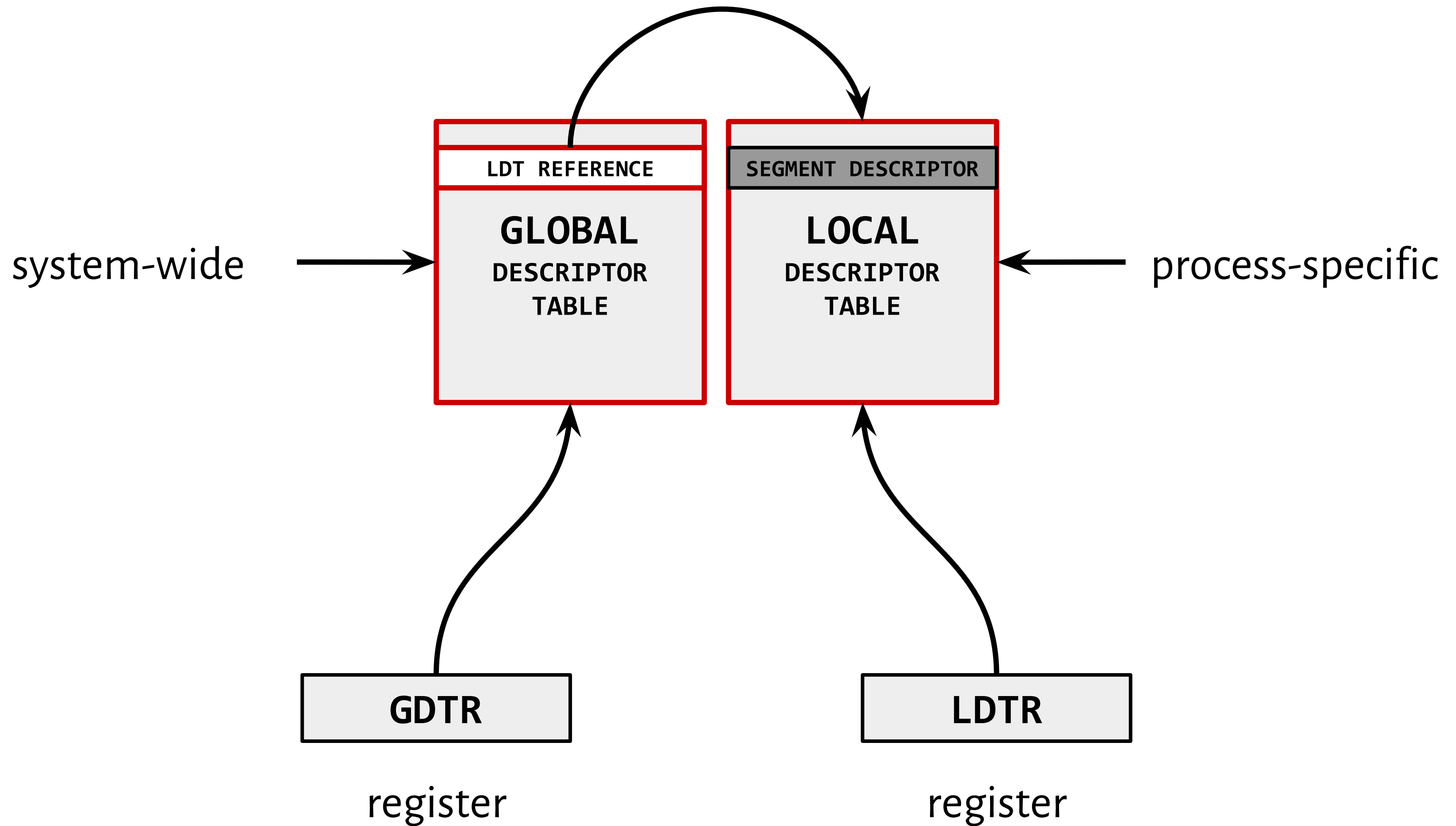


http://www.scs.stanford.edu/05au-cs240c/lab/i386/s05_01.htm

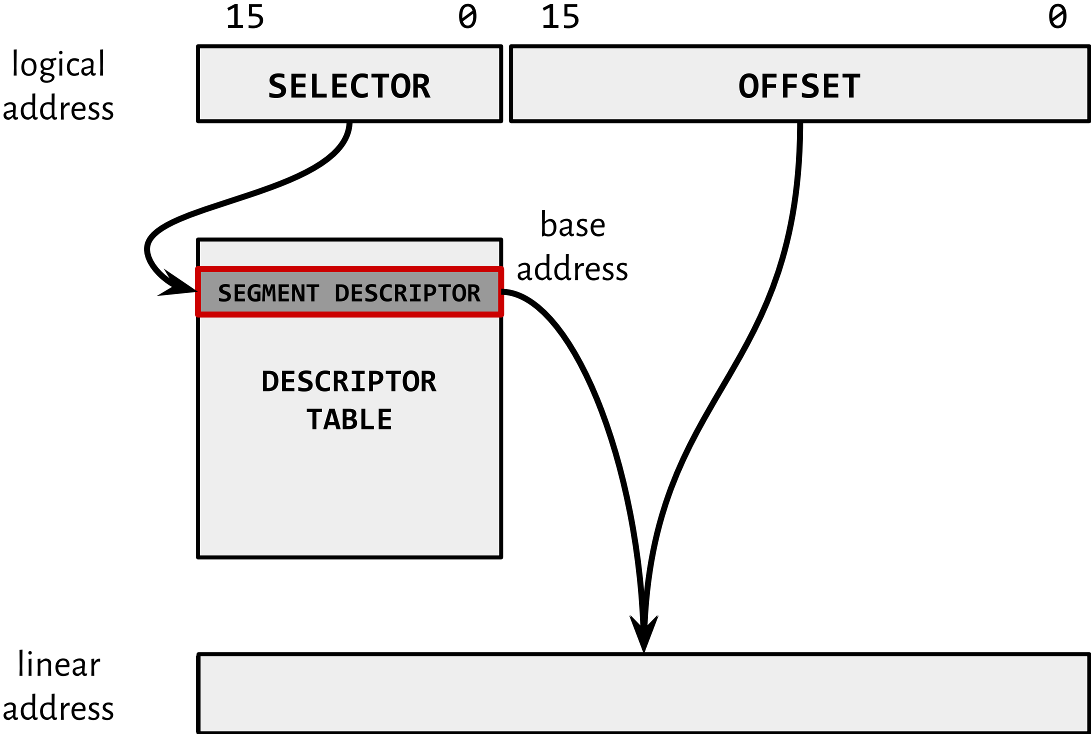
Descriptor tables



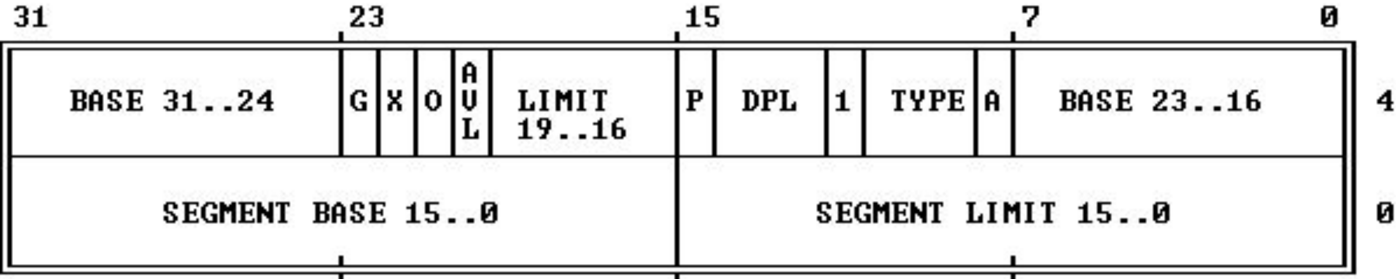
Descriptor tables



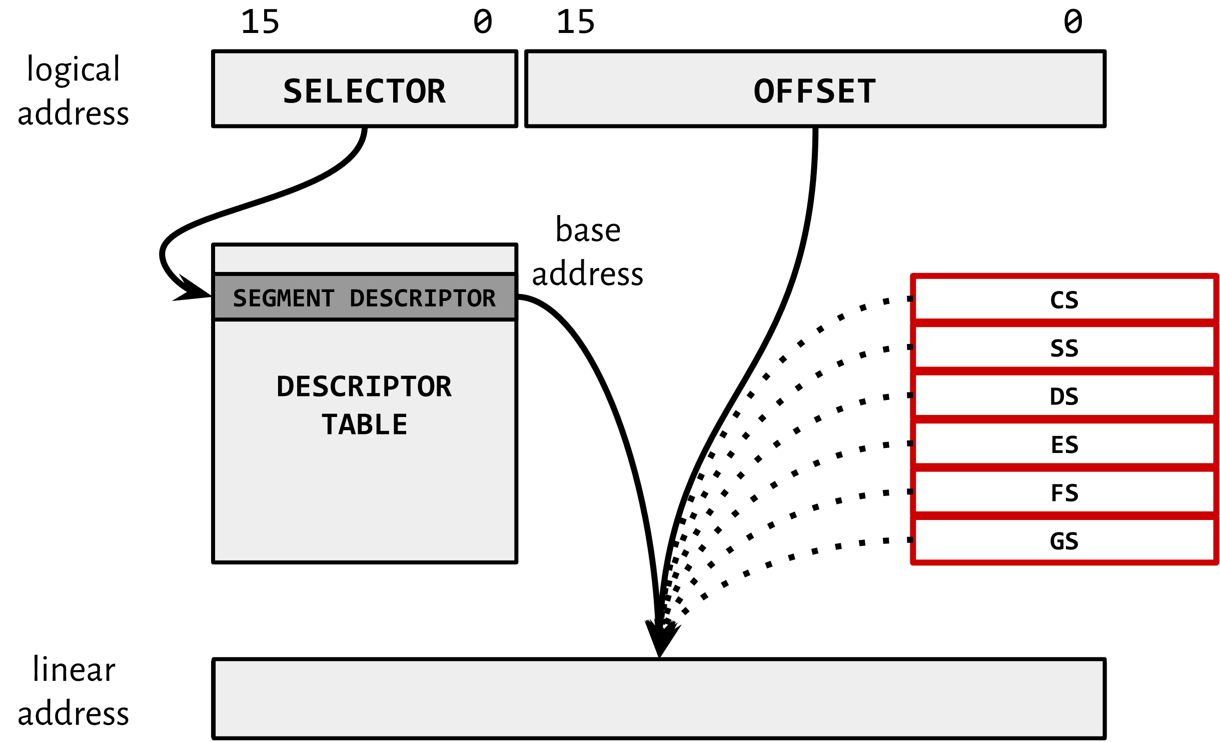
Segment descriptor



http://www.scs.stanford.edu/05au-cs240c/lab/i386/s05_01.htm



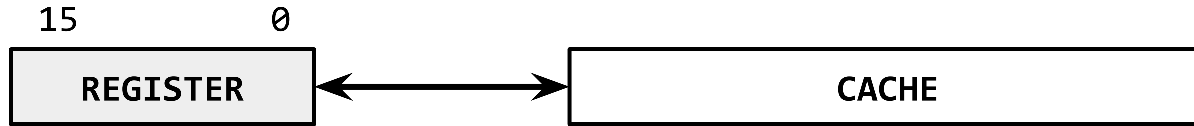
Segment registers



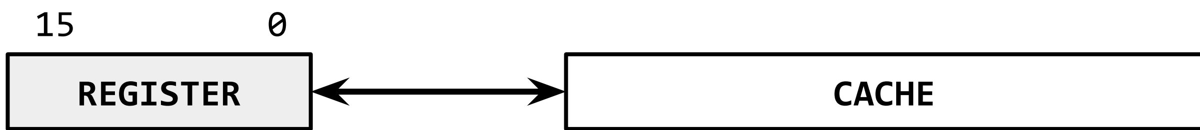
http://www.scs.stanford.edu/05au-cs240c/lab/i386/s05_01.htm

	16-BIT VISIBLE SELECTOR	HIDDEN DESCRIPTOR
CS		
SS		
DS		
ES		

Segment registers

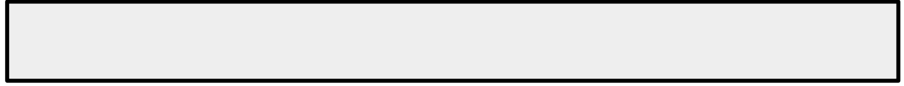


Segment registers

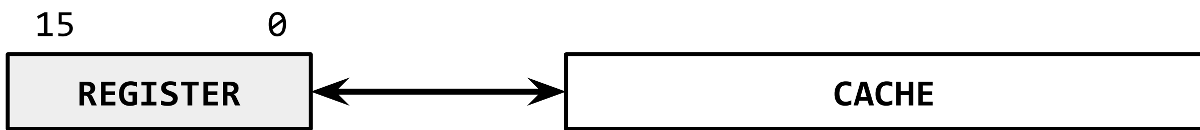


Using the cache:

```
mov [si], ax
```

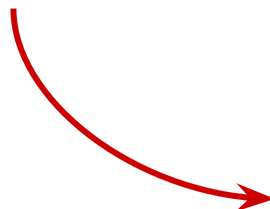


Segment registers



Using the cache:

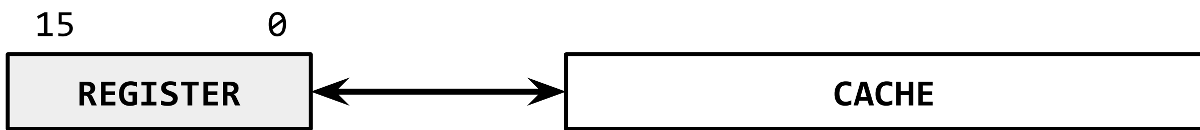
```
mov [si], ax
```



offset

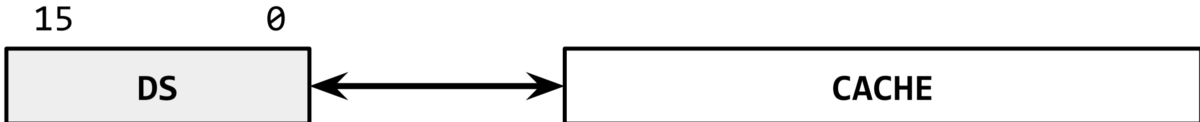
linear address

Segment registers



Using the cache:

```
mov [si], ax
```

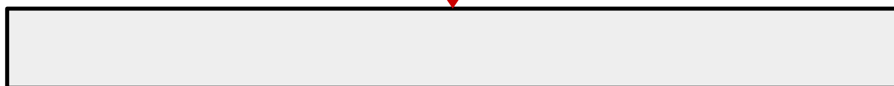


automatically

offset

base address

linear address



Why was segmentation introduced?

It allows addressing physical memory with 20 bits when using 16-bit addresses.

It separates memory used by distinct processes.

Is segmentation still in use?

Daniel P. Bovet, Marco Cesati, *Understanding Linux Kernel, 3rd Edition* (p. 42):

All Linux processes running in User Mode ***use the same pair of segments to address instructions and data****. These segments are called user code segment and user data segment , respectively. Similarly, all Linux processes running in Kernel Mode use the same pair of segments to address instructions and data: they are called kernel code segment and kernel data segment, respectively.*

Is segmentation still in use?

Daniel P. Bovet, Marco Cesati, *Understanding Linux Kernel, 3rd Edition* (p. 42):

All Linux processes running in User Mode ***use the same pair of segments to address instructions and data***. *These segments are called user code segment and user data segment , respectively. Similarly, all Linux processes running in Kernel Mode use the same pair of segments to address instructions and data: they are called kernel code segment and kernel data segment, respectively.*

Table 2-3. Values of the Segment Descriptor fields for the four main Linux segments

Segment	Base	G	Limit	S	Type	DPL	D/B	P
user code	0x00000000	1	0xfffff	1	10	3	1	1
user data	0x00000000	1	0xfffff	1	2	3	1	1
kernel code	0x00000000	1	0xfffff	1	10	0	1	1
kernel data	0x00000000	1	0xfffff	1	2	0	1	1

Is segmentation still in use?

Daniel P. Bovet, Marco Cesati, *Understanding Linux Kernel, 3rd Edition* (p. 42):

All Linux processes running in User Mode ***use the same pair of segments to address instructions and data****. These segments are called user code segment and user data segment , respectively. Similarly, all Linux processes running in Kernel Mode use the same pair of segments to address instructions and data: they are called kernel code segment and kernel data segment, respectively.*

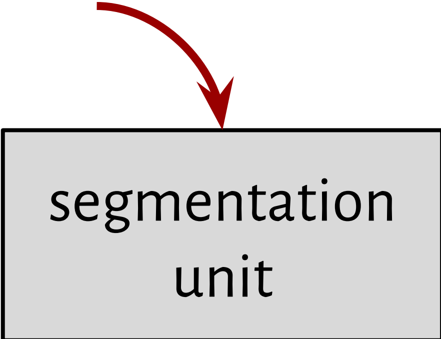
Table 2-3. Values of the Segment Descriptor fields for the four main Linux segments

Segment	Base	G	Limit	S	Type	DPL	D/B	P
user code	0x00000000	1	0xfffff	1	10	3	1	1
user data	0x00000000	1	0xfffff	1	2	3	1	1
kernel code	0x00000000	1	0xfffff	1	10	0	1	1
kernel data	0x00000000	1	0xfffff	1	2	0	1	1

https://softwareengineering.stackexchange.com/questions/100047/why-not-segmentation

https://en.wikipedia.org/wiki/Flat_memory_model

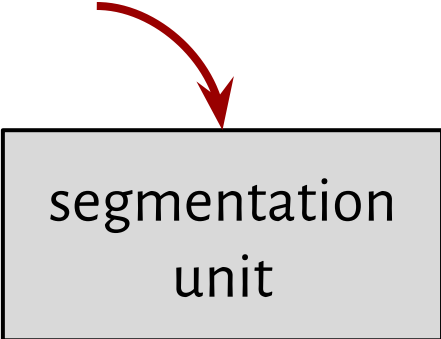
Memory address (i386)



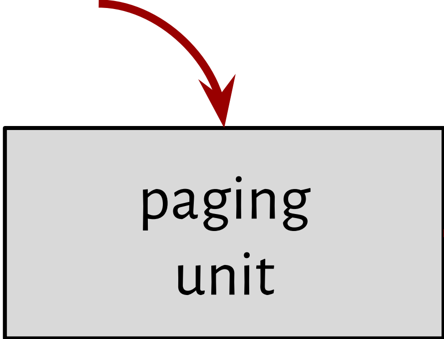
physical
address

Memory address (i386)

logical
address



linear
address



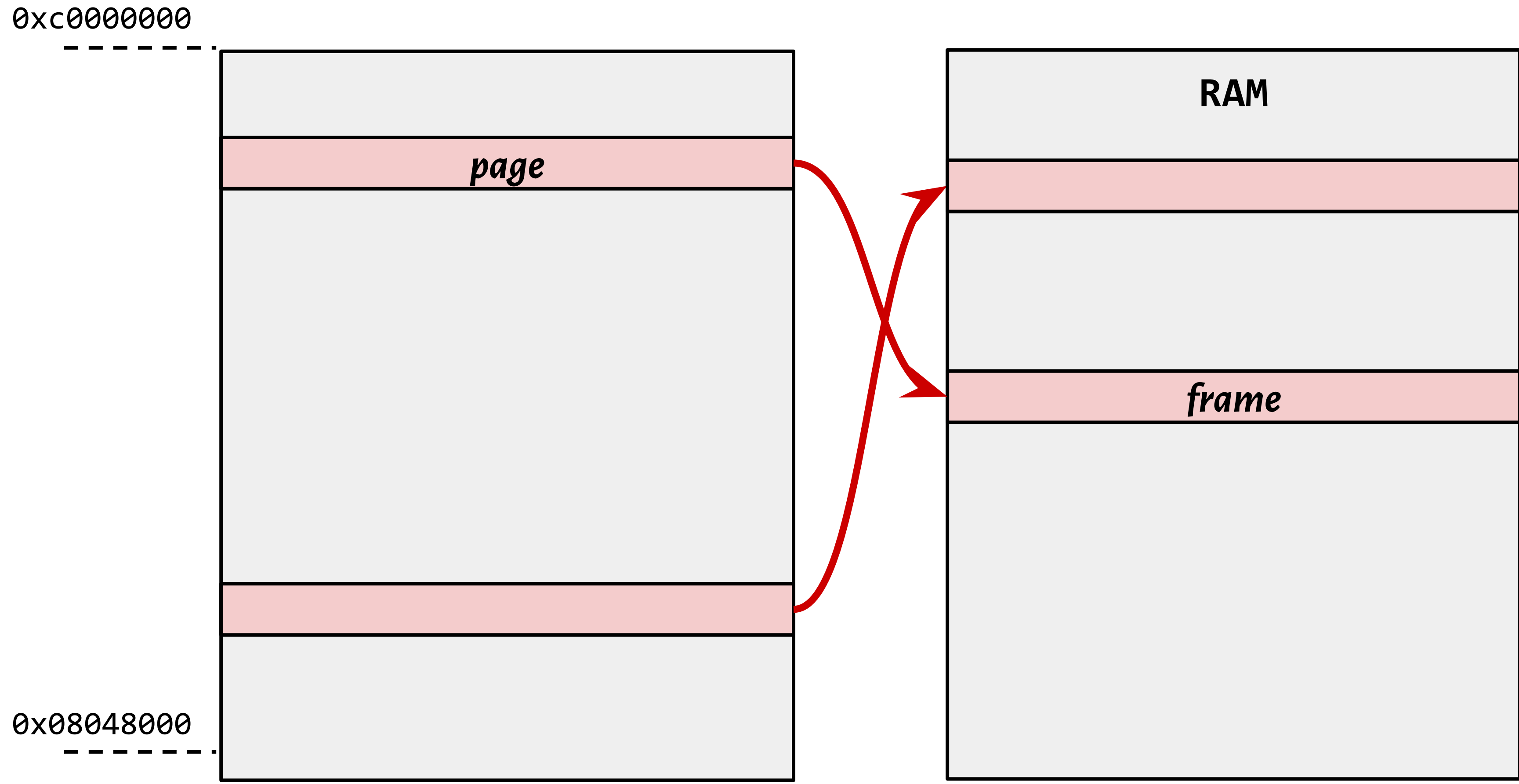
physical
address

Memory

What processes *think* memory is like:

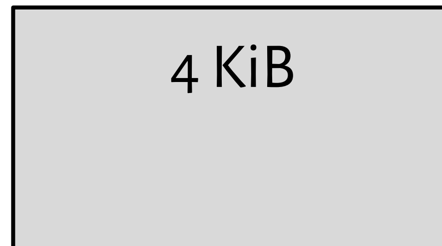
Memory

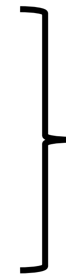
What processes *think* memory is like: While...:



Paging

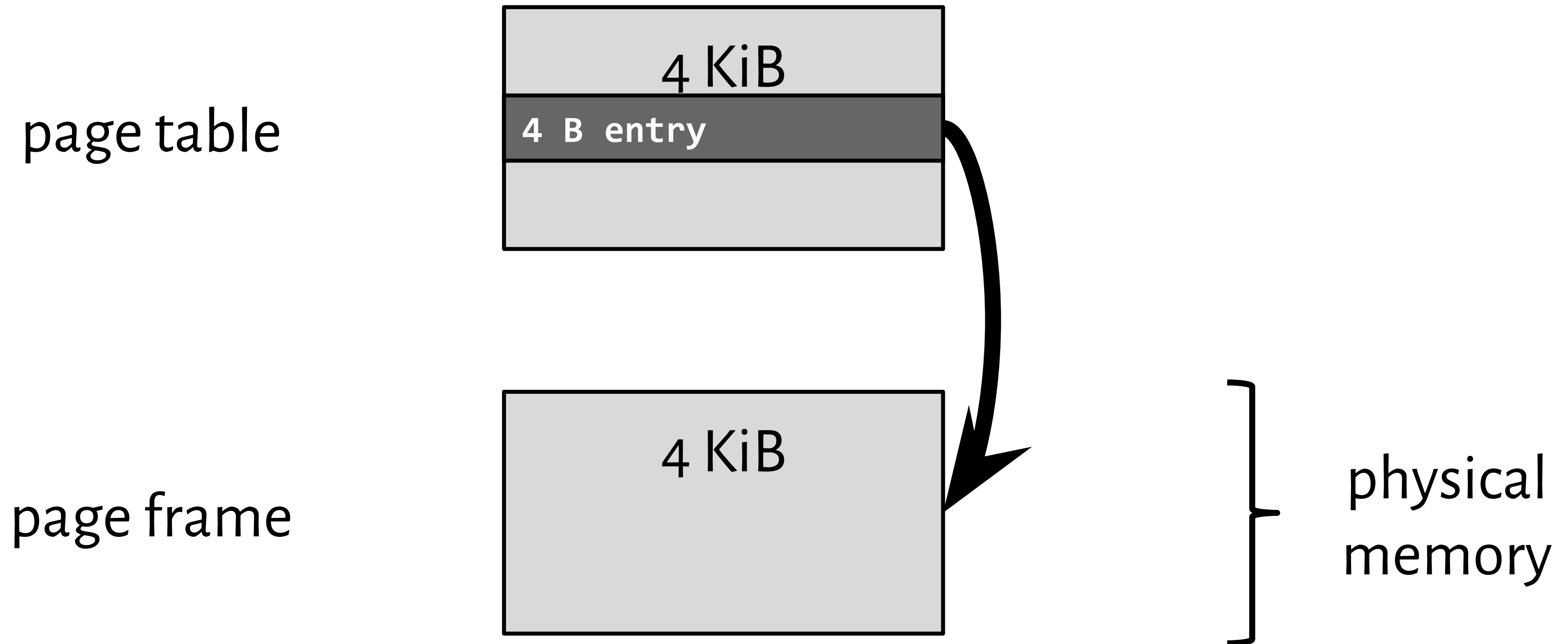
page frame



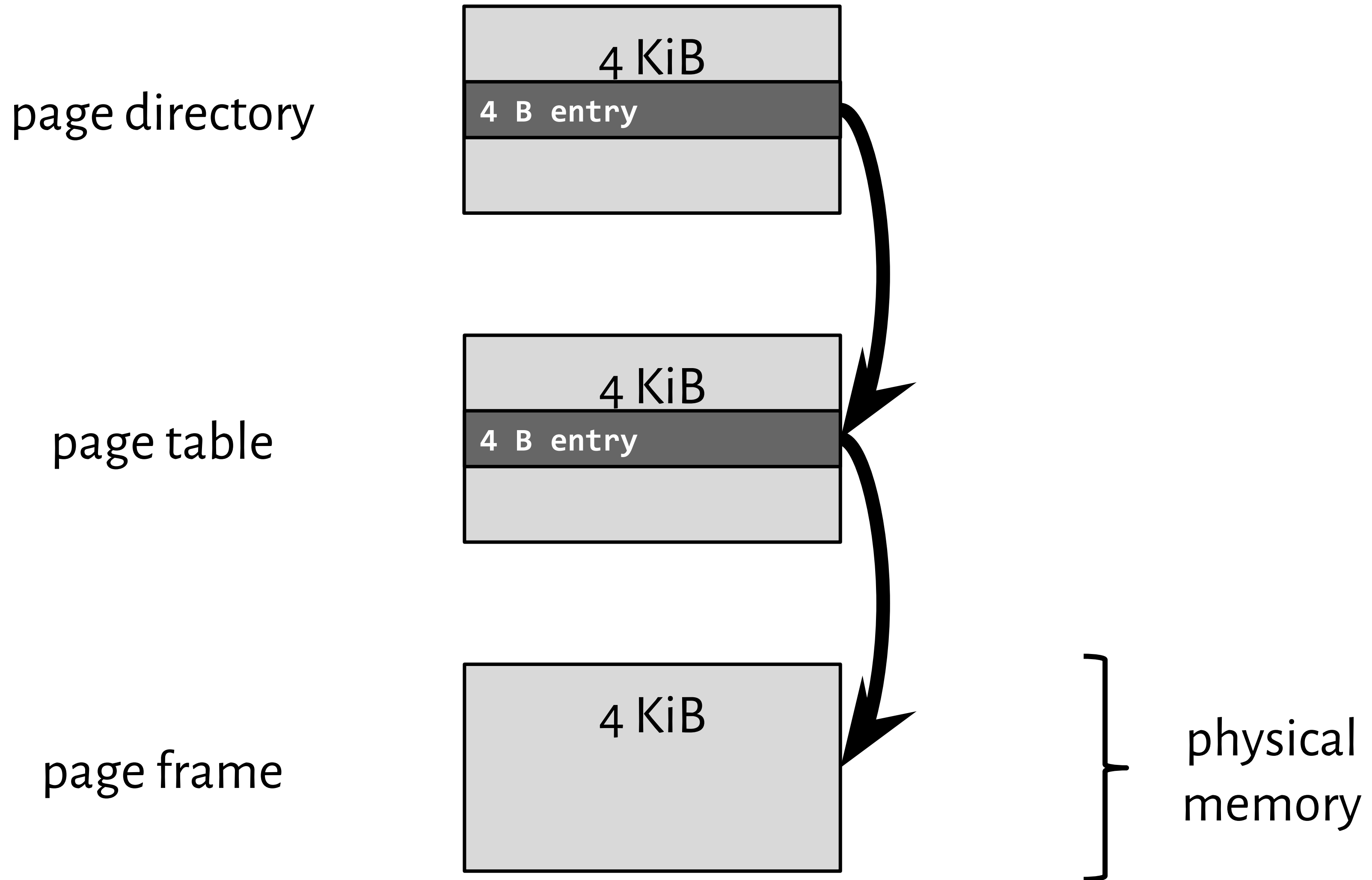


physical memory

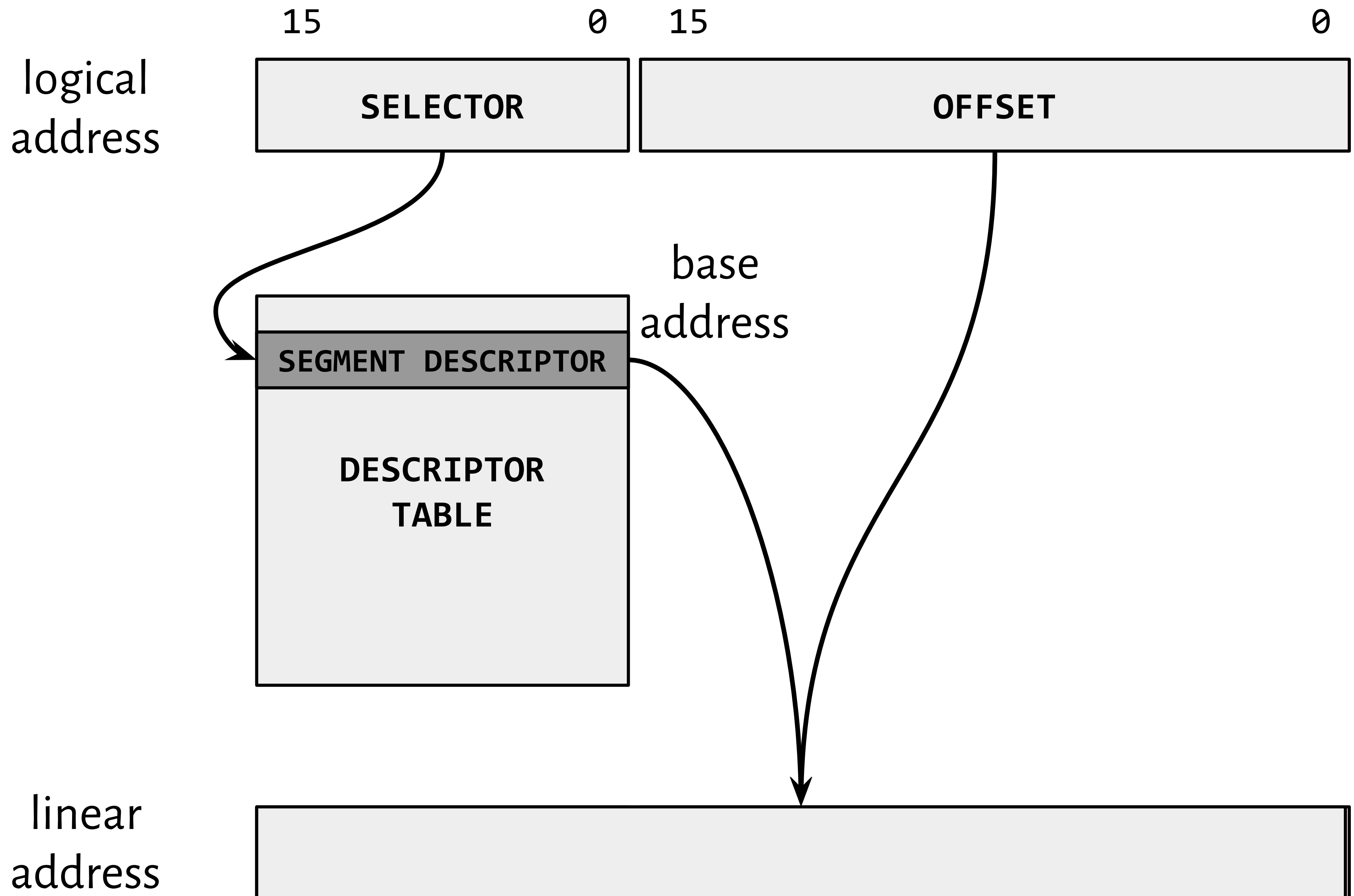
Paging



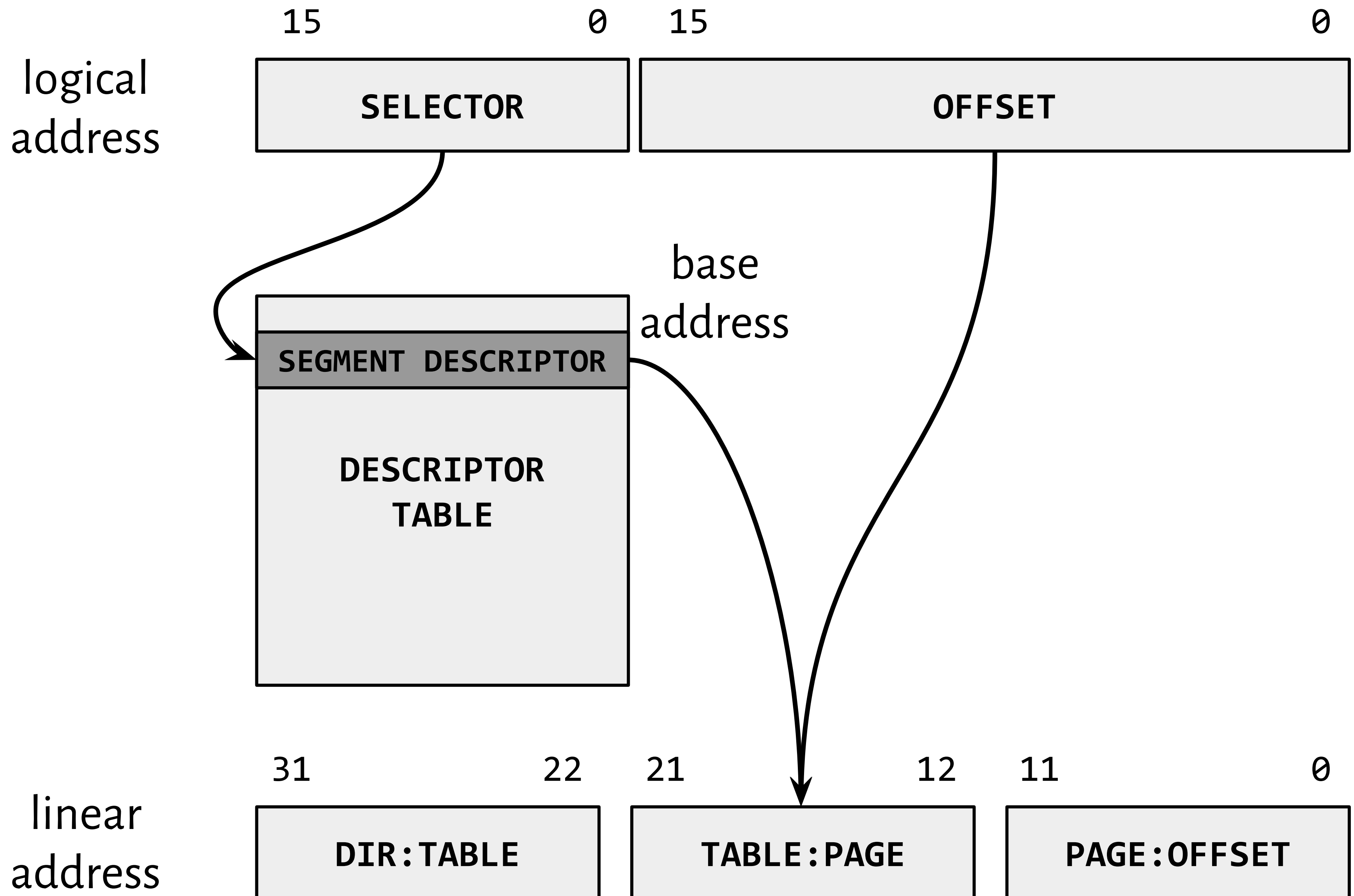
Paging



Linear address

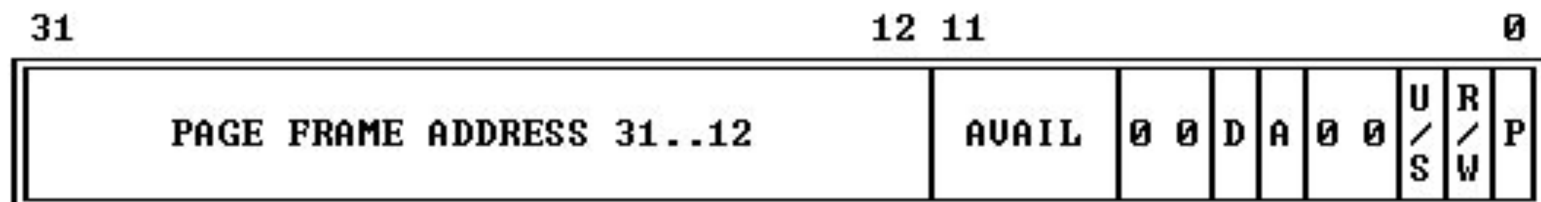


Linear address



Paging

Figure 5-10. Format of a Page Table Entry

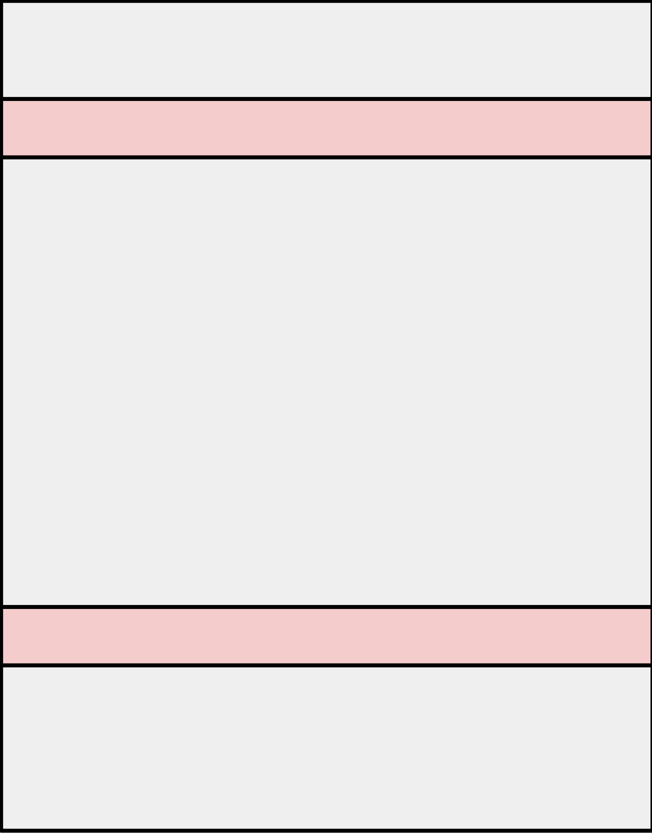


P - PRESENT
R/W - READ/WRITE
U/S - USER/SUPERVISOR
D - DIRTY
AVAIL - AVAILABLE FOR SYSTEMS PROGRAMMER USE

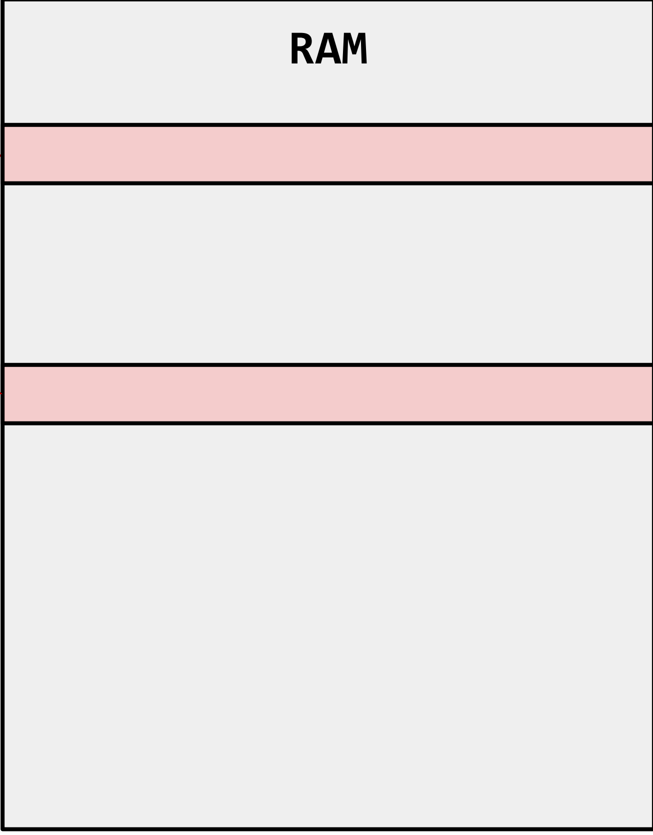
NOTE: 0 INDICATES INTEL RESERVED. DO NOT DEFINE.

Memory

What processes *think* memory is like:



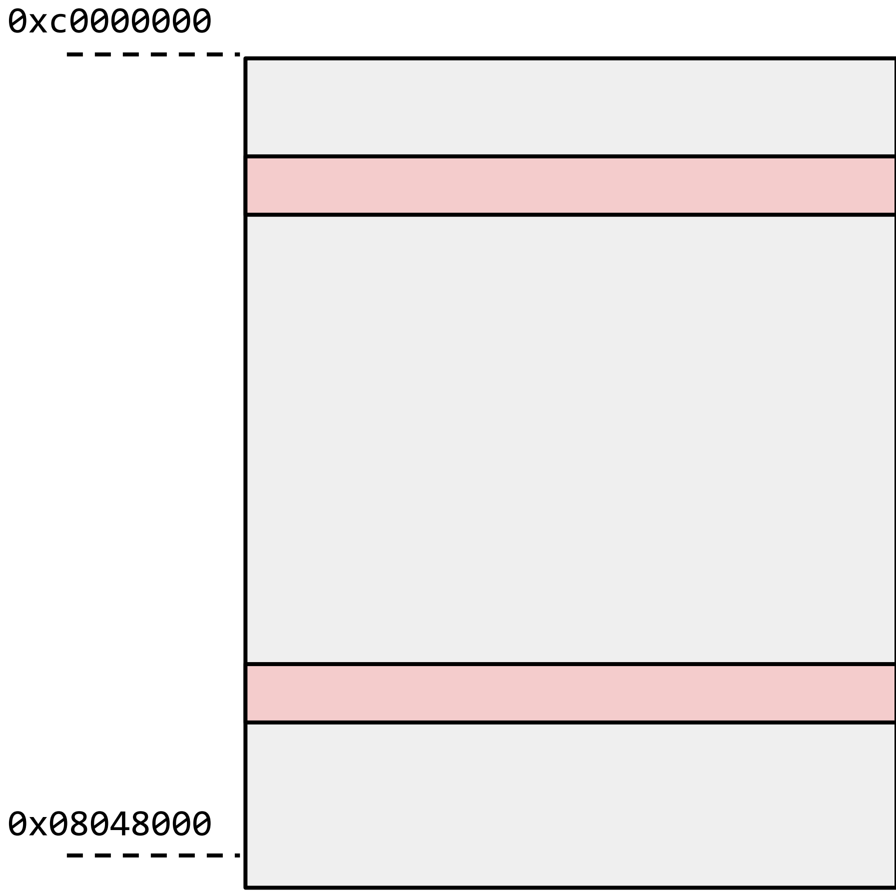
While...:

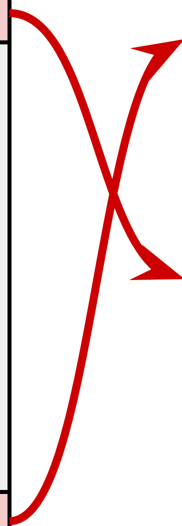


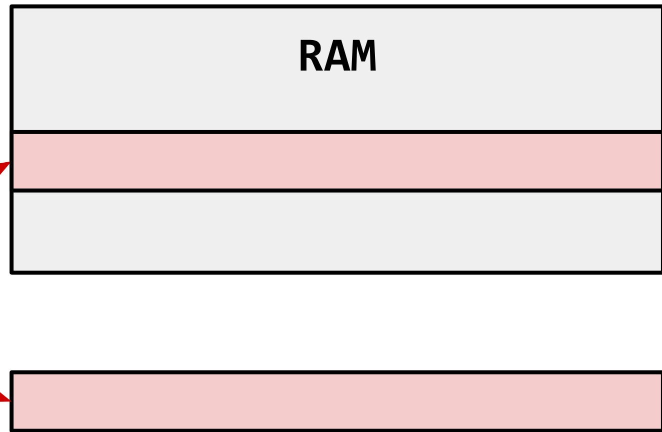
Memory

What processes *think* memory is like:

While...:

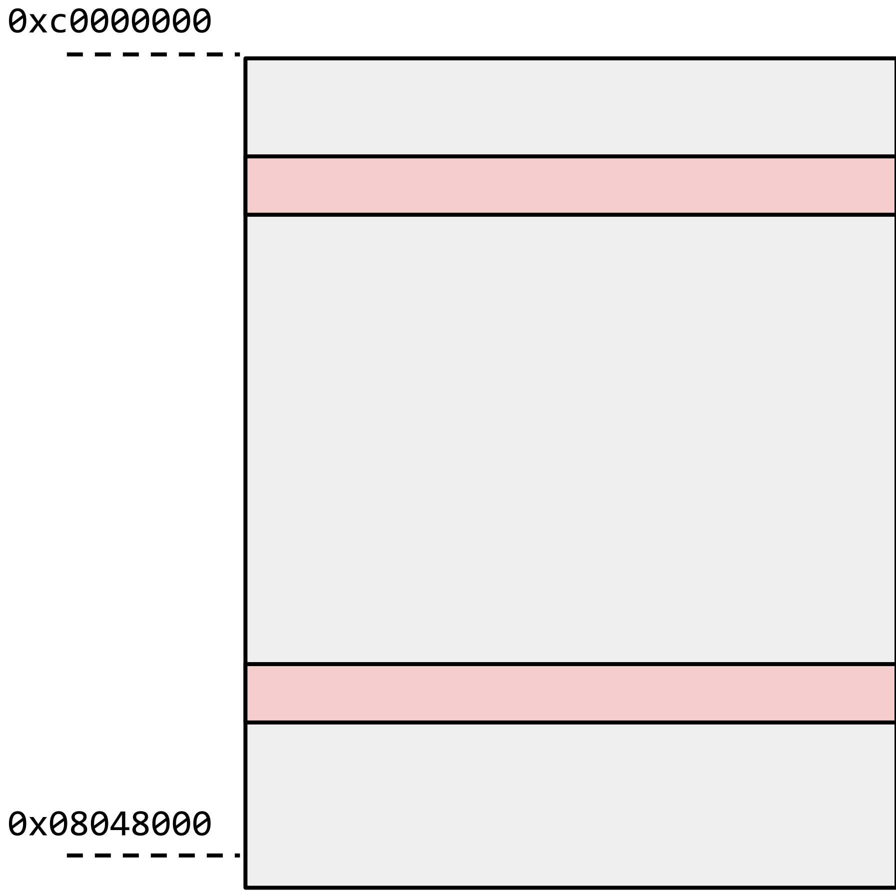


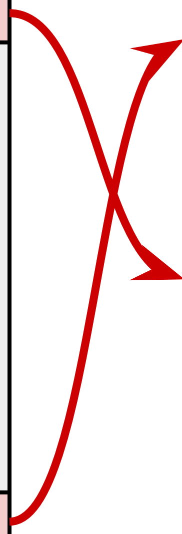




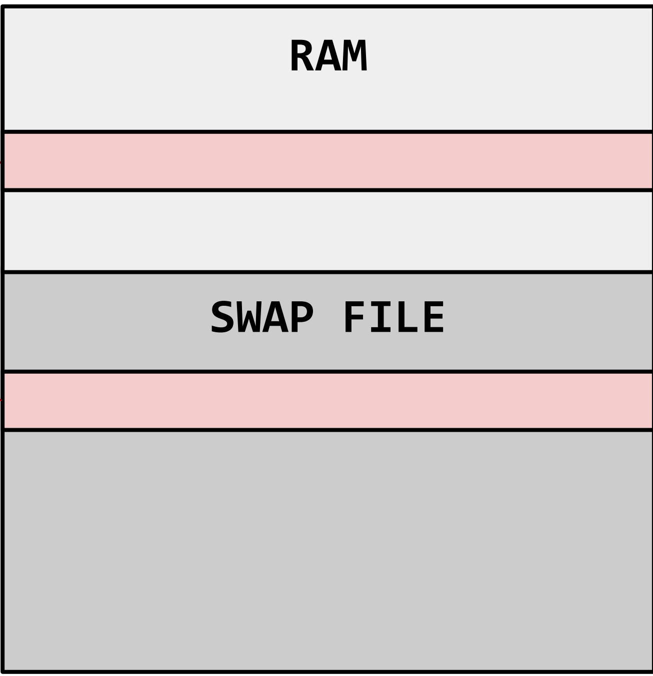
Memory

What processes *think* memory is like:





While...:



Is paging sufficient?

Segmentation

It allows addressing physical memory with 20 bits when using 16-bit addresses.

It separates memory used by distinct processes.

Is paging sufficient?

It separates memory used by distinct processes.

Is paging sufficient?

It allows allocating more memory than physically available RAM.

It separates memory used by distinct processes.

Is paging efficient?

Yes: it is performed by hardware (**Management Memory Unit**).

The process gets to be executed.

The OS kernel informs **MMU** on the process' page table.

MMU takes care of the subsequent translations.

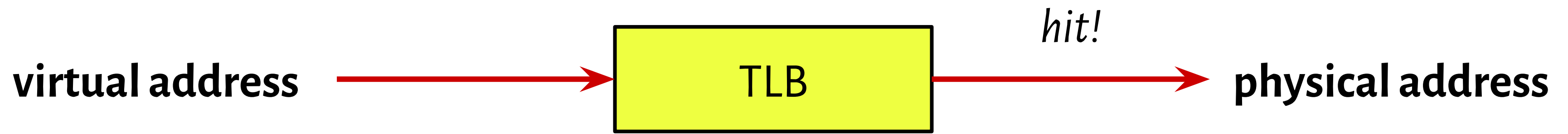
Is paging efficient?

On the other hand: MMU refers to page tables.

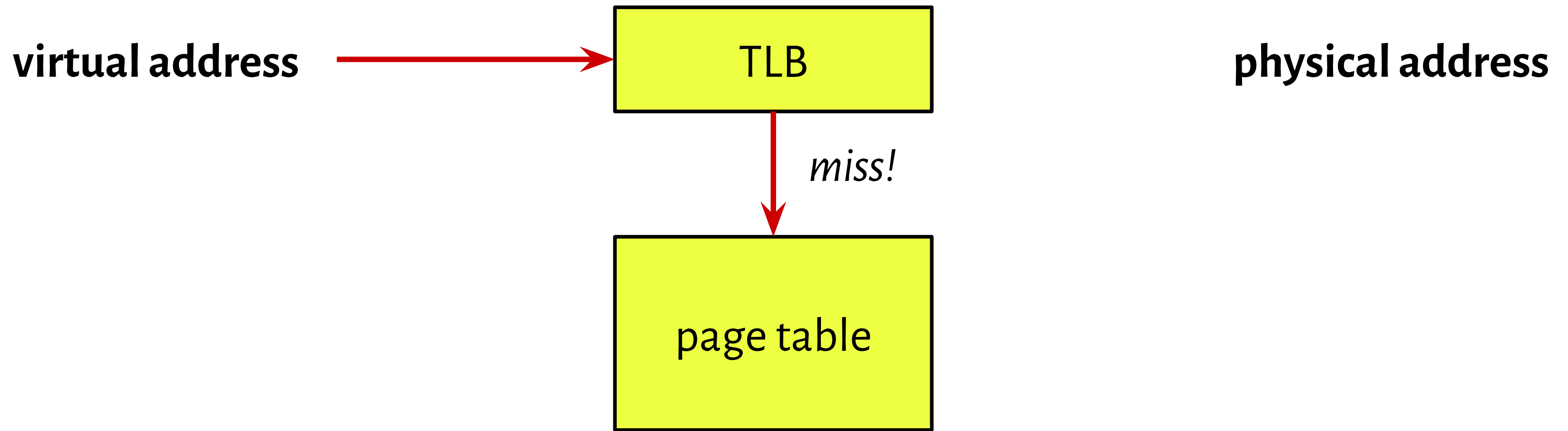
No, we're safe: there is a Translation Lookaside Buffer (TLB).

Paging: the scheme

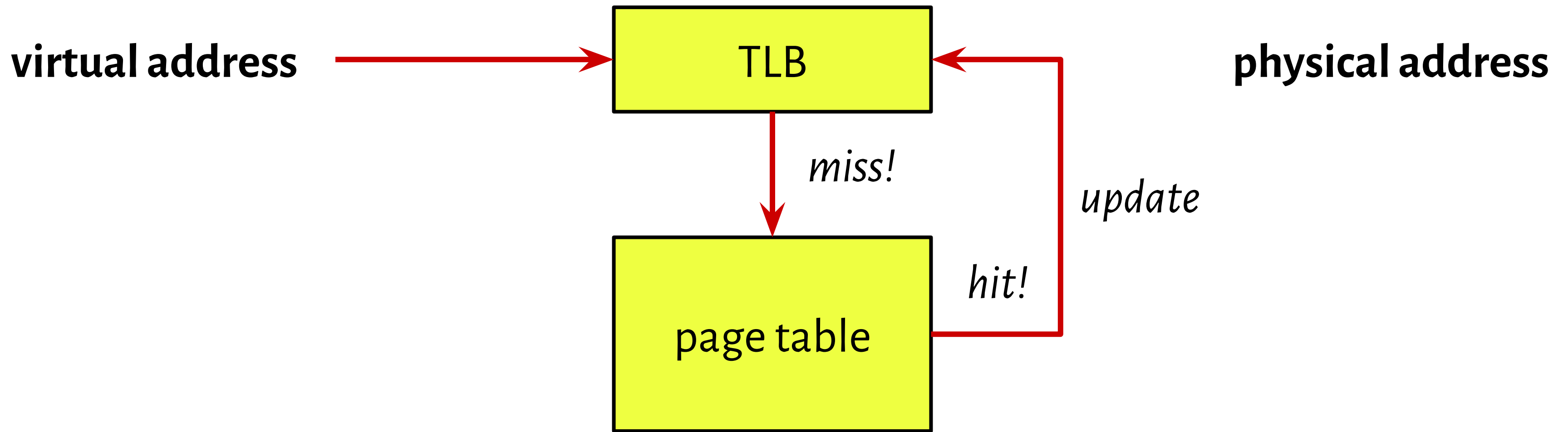
Paging: the scheme



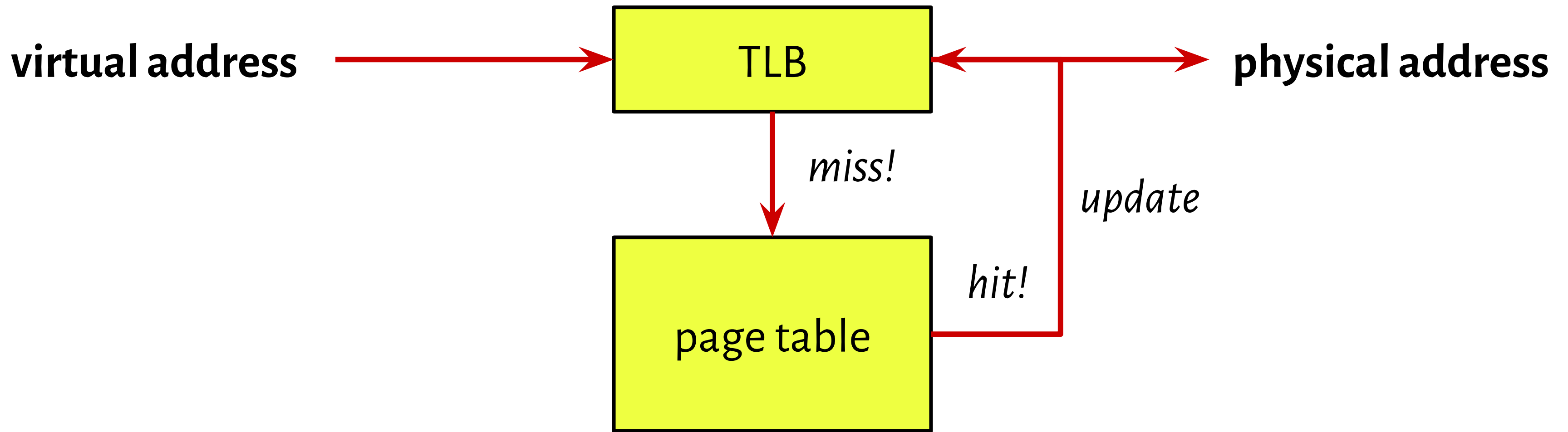
Paging: the scheme



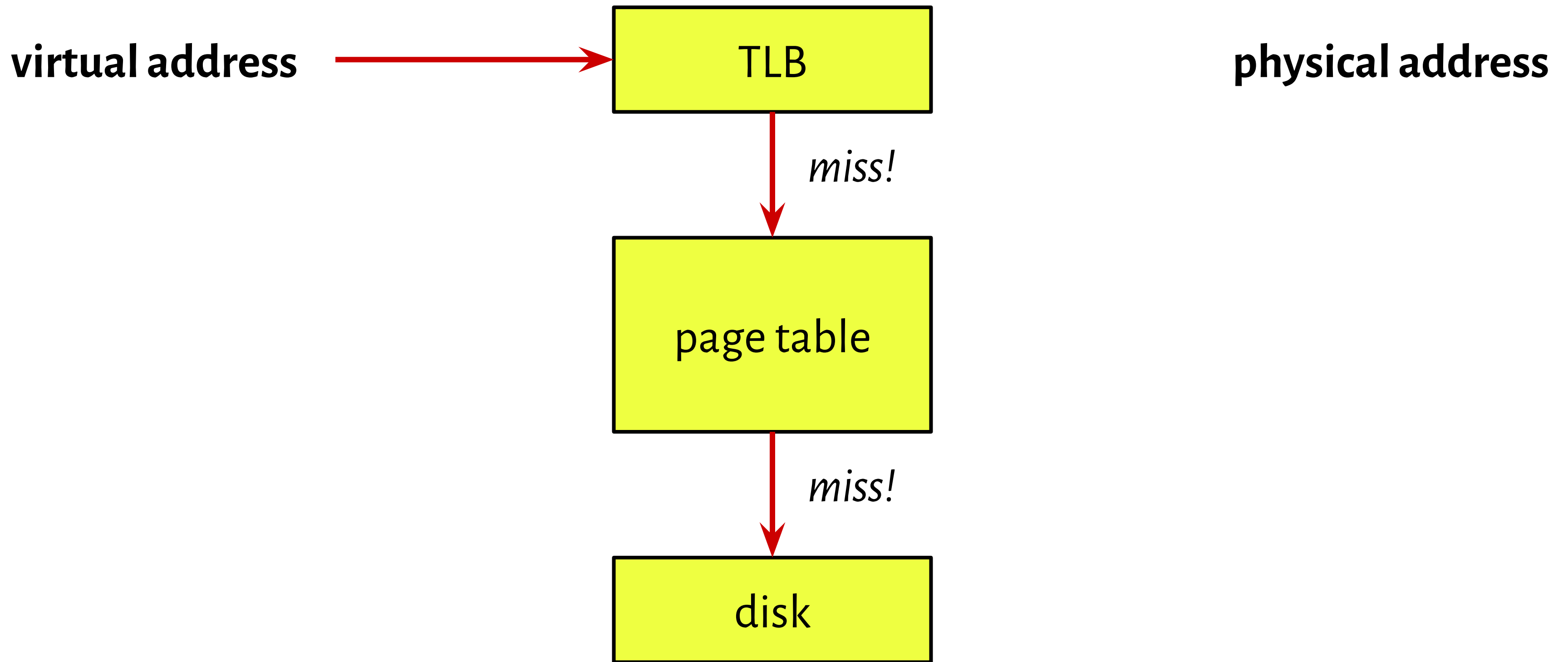
Paging: the scheme



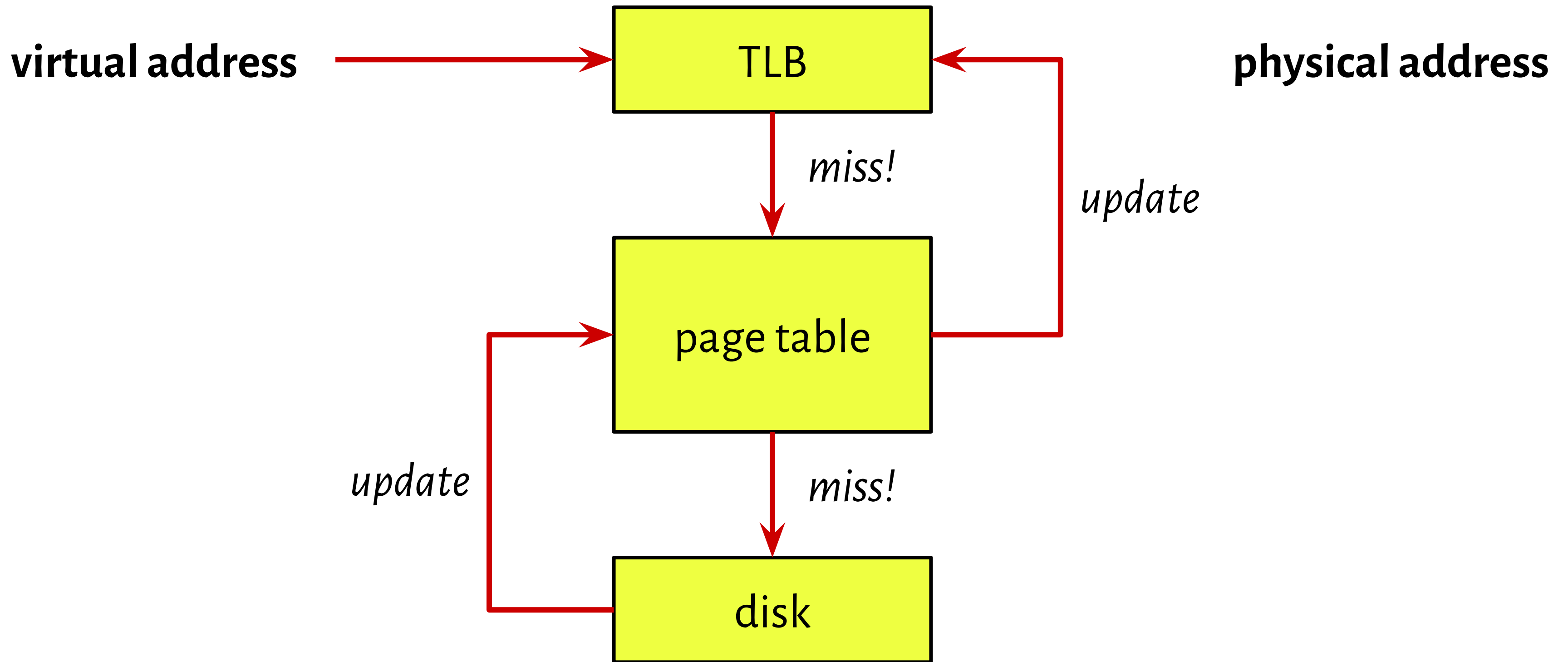
Paging: the scheme



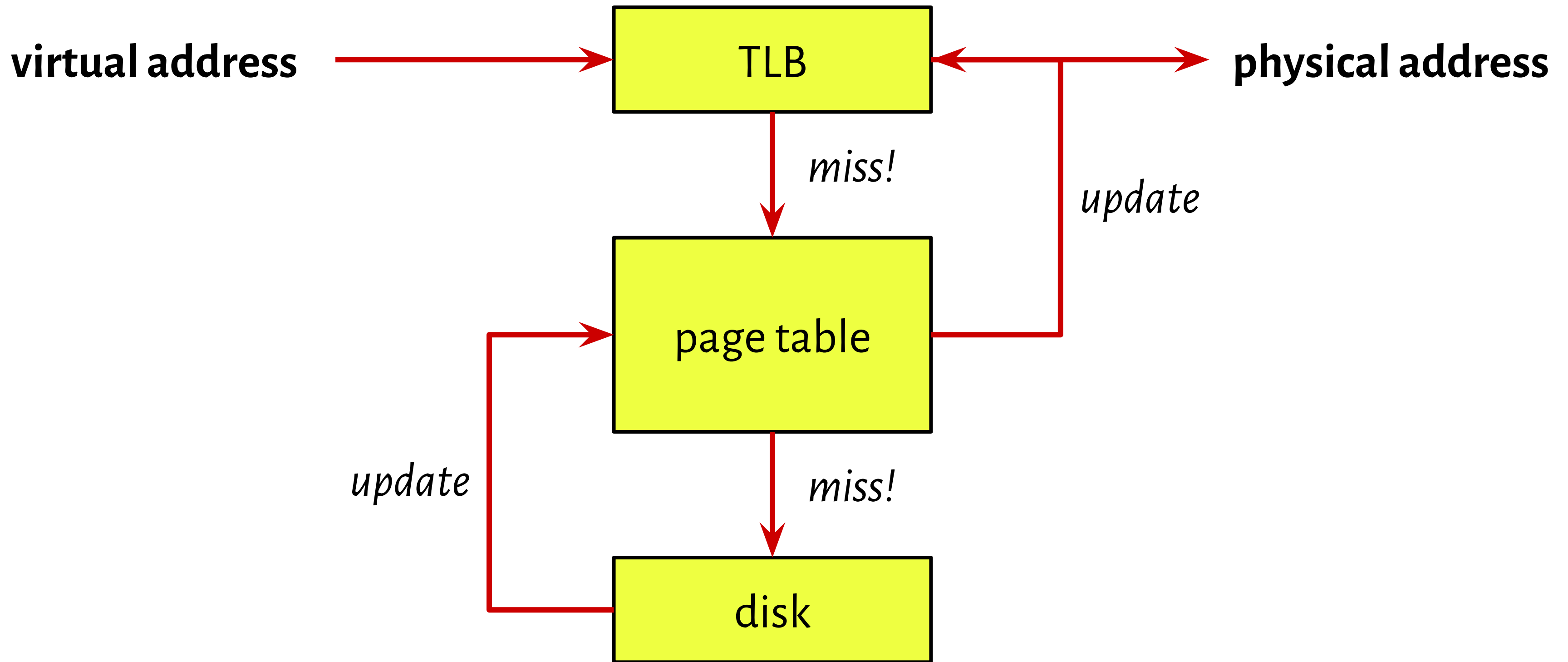
Paging: the scheme



Paging: the scheme



Paging: the scheme



Page replacement algorithms

Extra frames do not solve the problem:

https://en.wikipedia.org/wiki/Bélády's_anomaly

Page replacement algorithms

Extra frames do not solve the problem:

https://en.wikipedia.org/wiki/Bélády's_anomaly

You will have to solve exam problems...

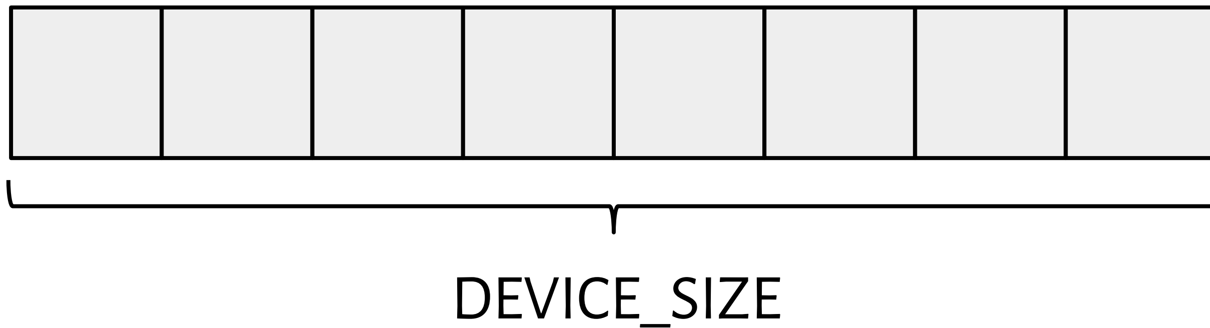
http://students.mimuw.edu.pl/SO/Wyklady-html/06_pamiec/6_cw-pam2.html

http://students.mimuw.edu.pl/SO/Wyklady-html/05_pamiec/5_pamiec.html

Assignment #6

Implement a stack-driver.

1. Allocate memory for the buffer. Dynamically.

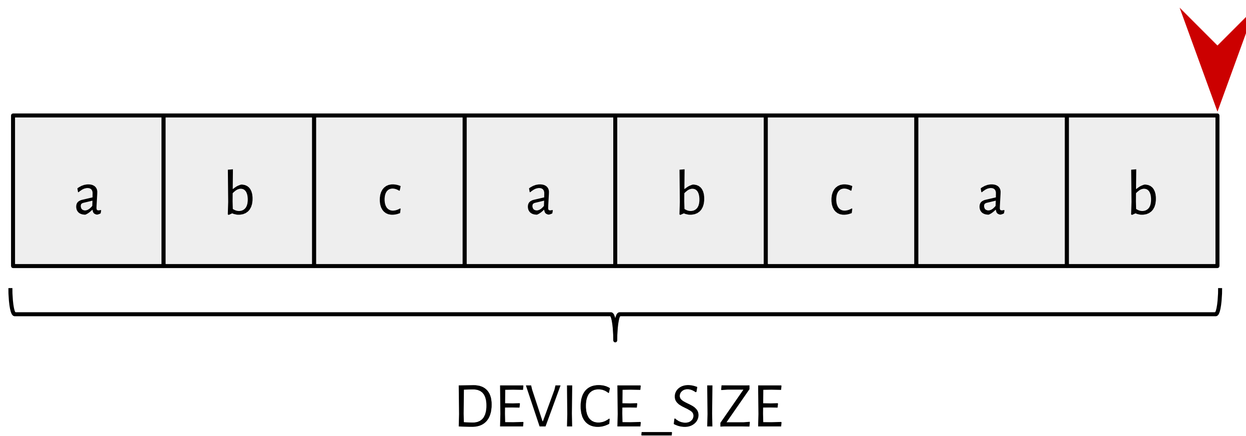



```
# service up /service/hello_stack
```

Assignment #6

Implement a stack-driver.

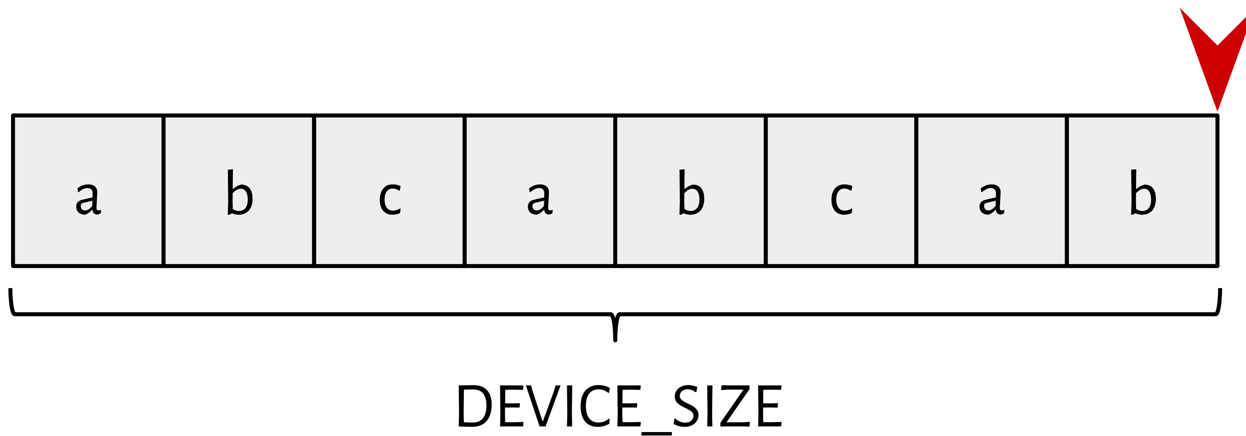
2. Initialize the device.



Assignment #6

Implement a stack-driver.

2. Define reading.

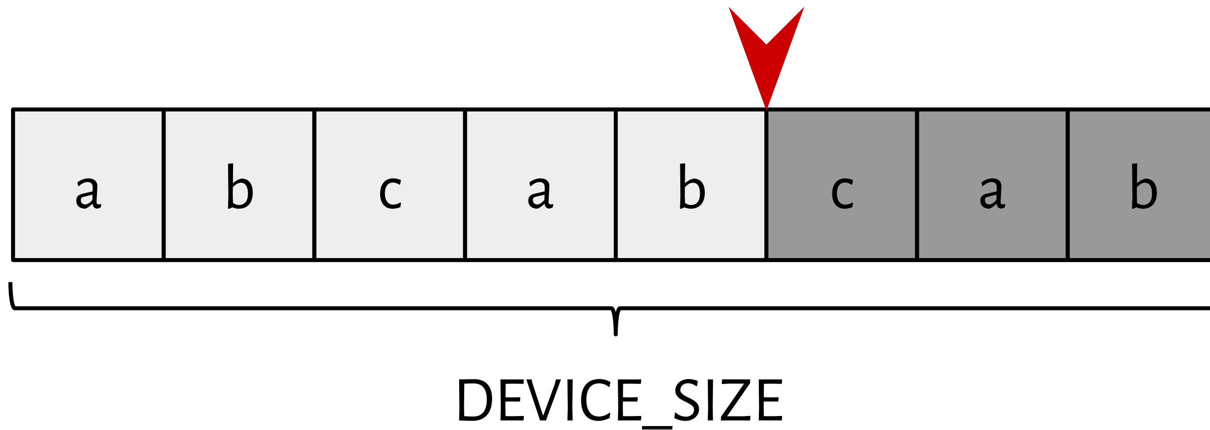



```
# head -c 3 /dev/hello_stack
```

Assignment #6

Implement a stack-driver.

2. Define reading.

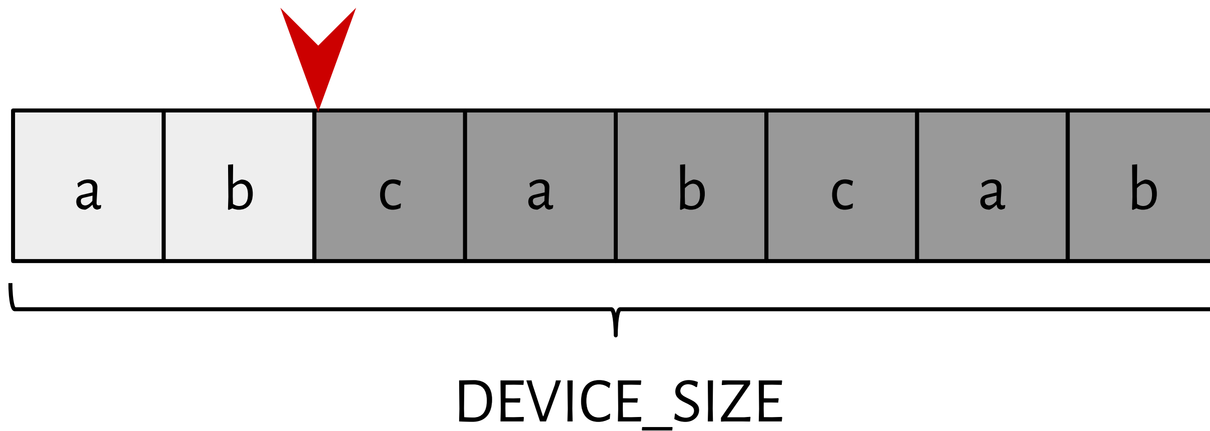



```
# head -c 3 /dev/hello_stack
# cab
```

Assignment #6

Implement a stack-driver.

2. Define reading.

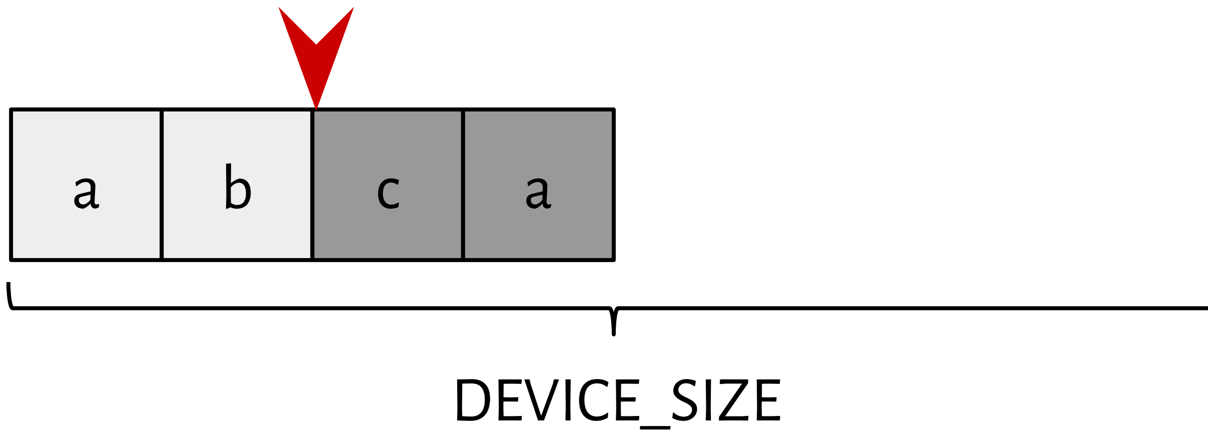



```
# head -c 3 /dev/hello_stack
# cab
```

Assignment #6

Implement a stack-driver.

2. Define reading.

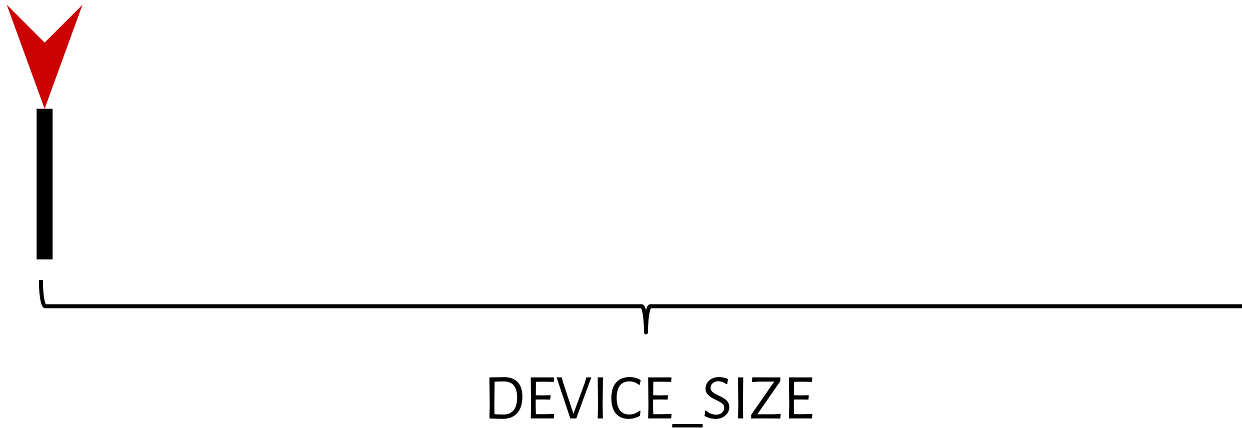



```
# head -c 3 /dev/hello_stack
# cab
```

Assignment #6

Implement a stack-driver.

2. Define reading.

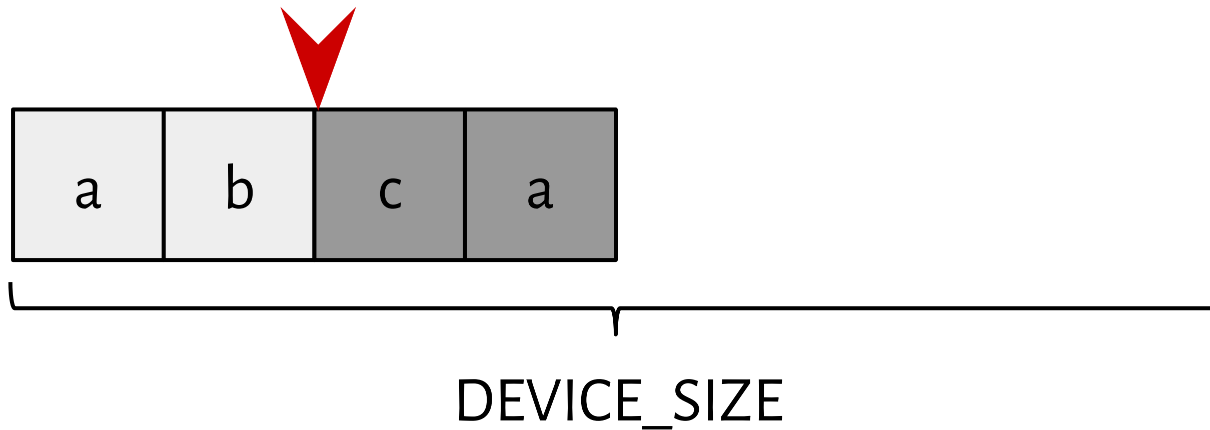



```
# head -c 3 /dev/hello_stack
# ab
```

Assignment #6

Implement a stack-driver.

3. Define writing.

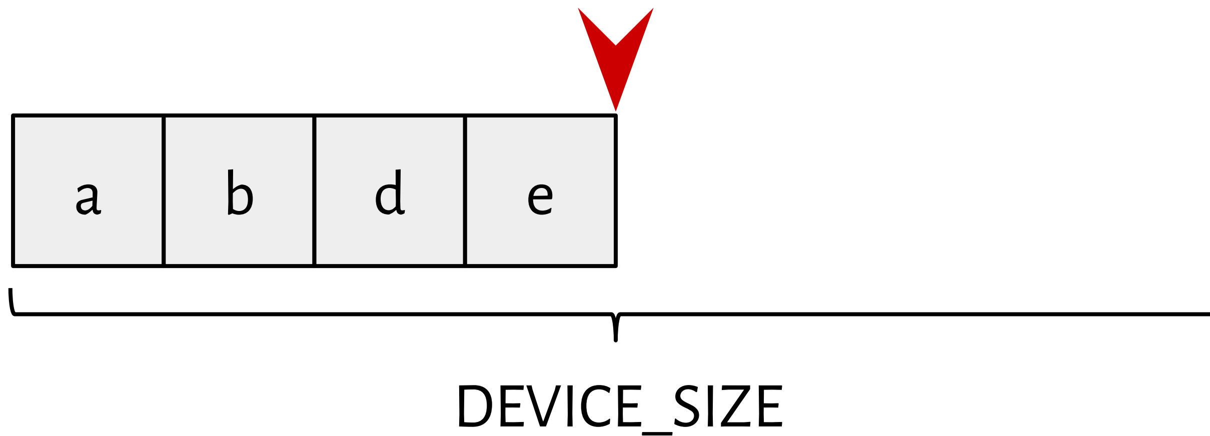



```
# echo def > /dev/hello_stack
```

Assignment #6

Implement a stack-driver.

3. Define writing.

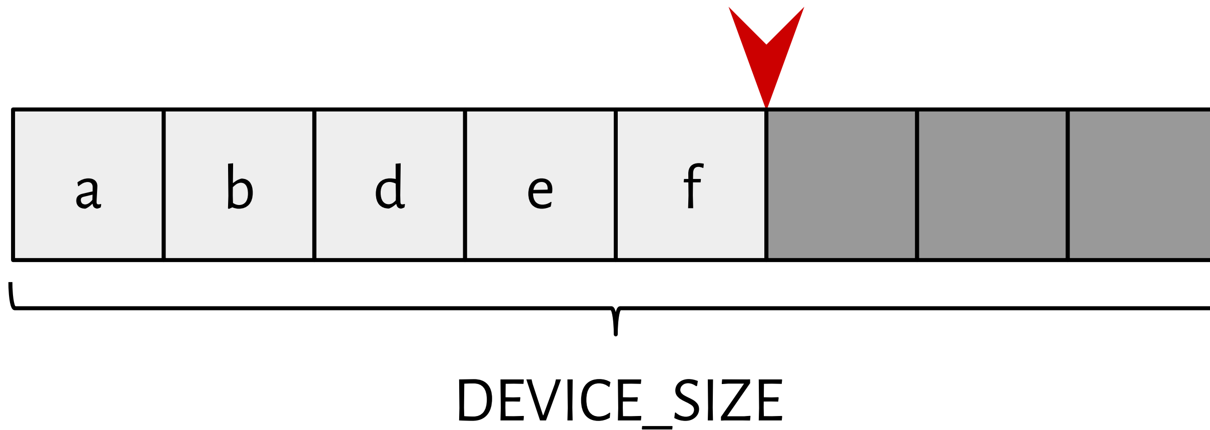



```
# echo def > /dev/hello_stack
```

Assignment #6

Implement a stack-driver.

3. Define writing.

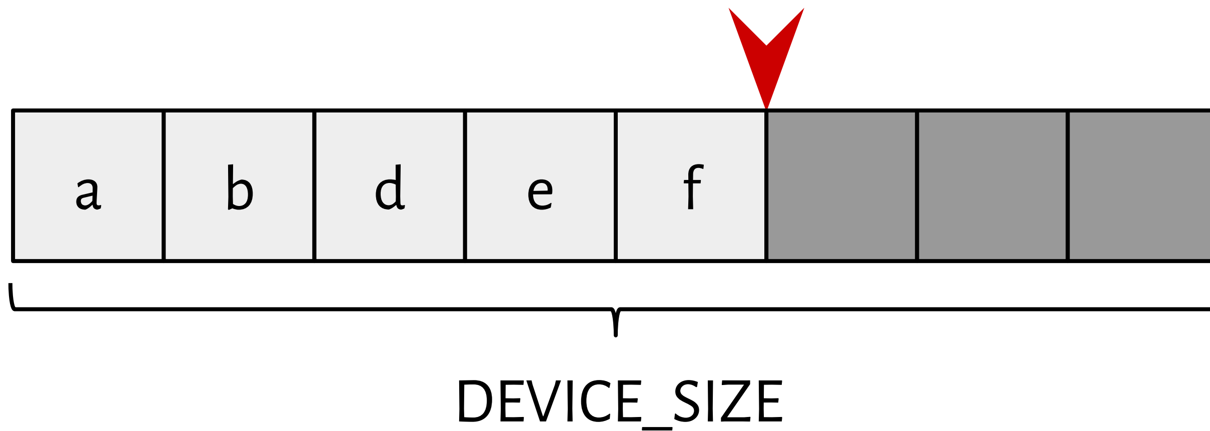



```
# echo def > /dev/hello_stack
```

Assignment #6

Implement a stack-driver.

4. Keep the device's state intact during updates.




```
# service update /service/hello_stack
```