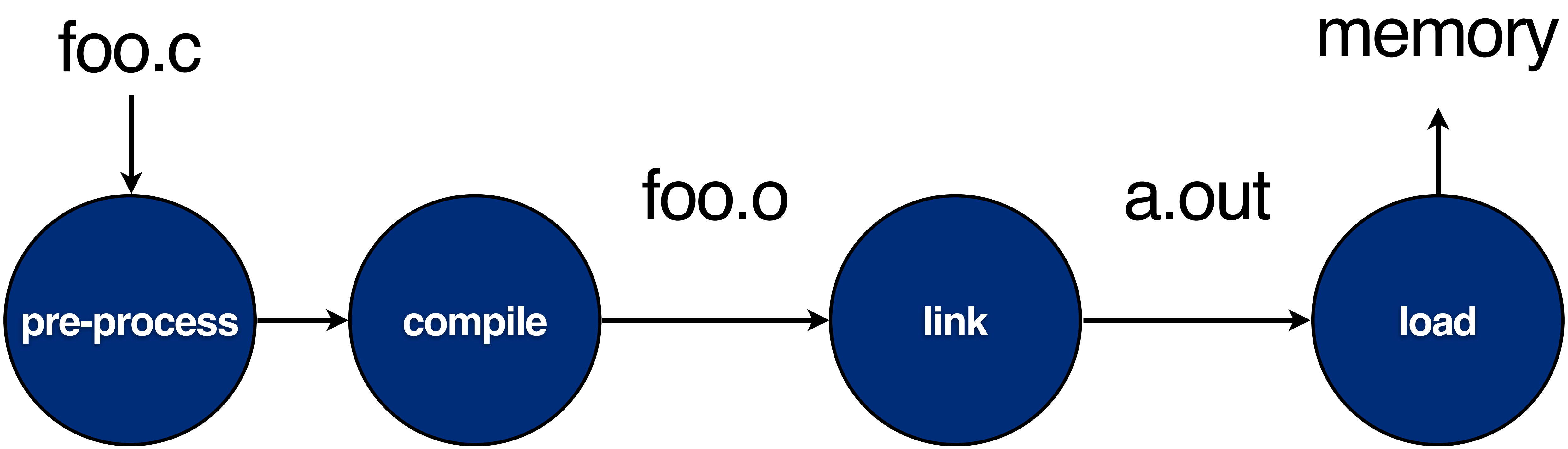
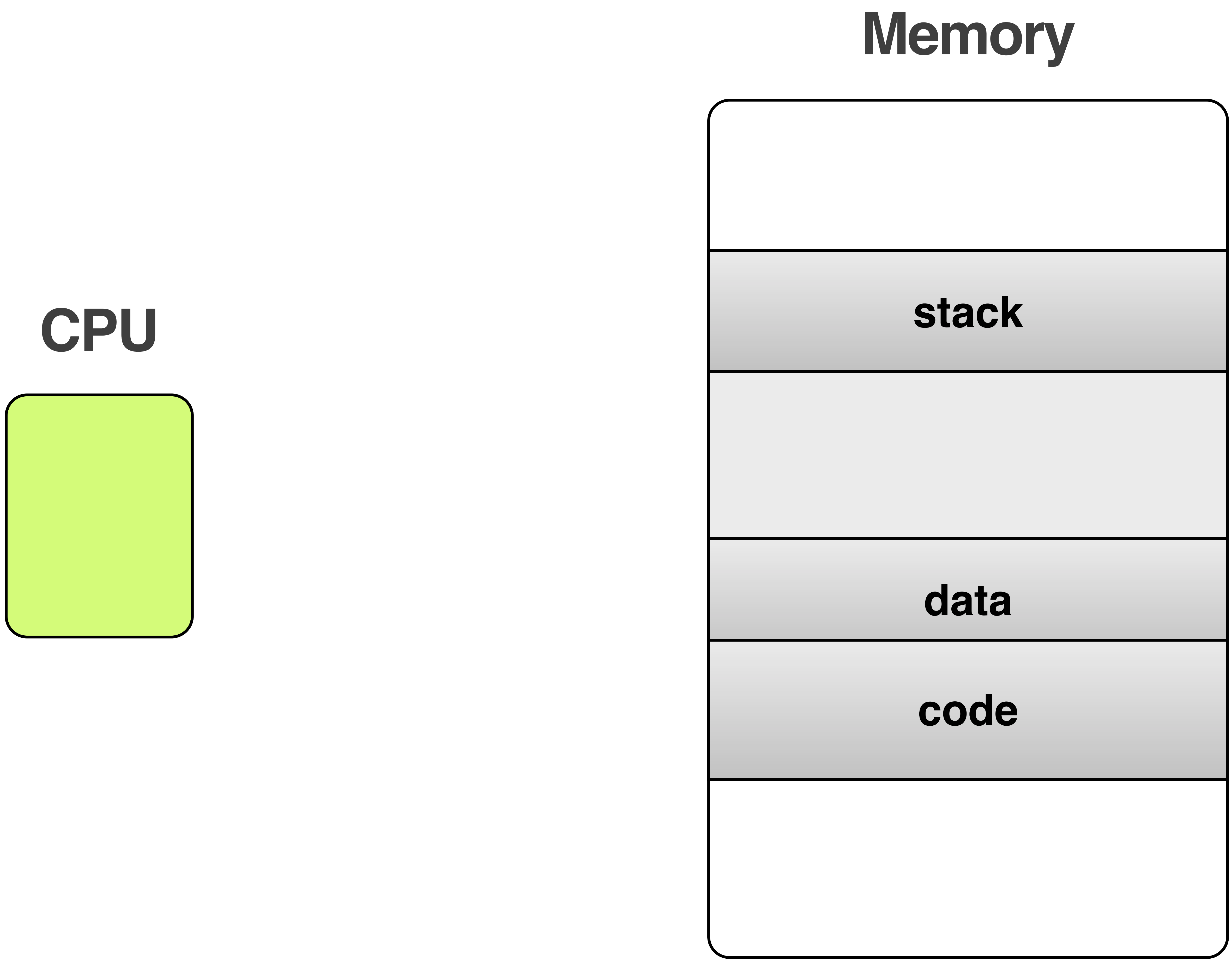
Lecture 8
Memory Management I

14 October, 2011

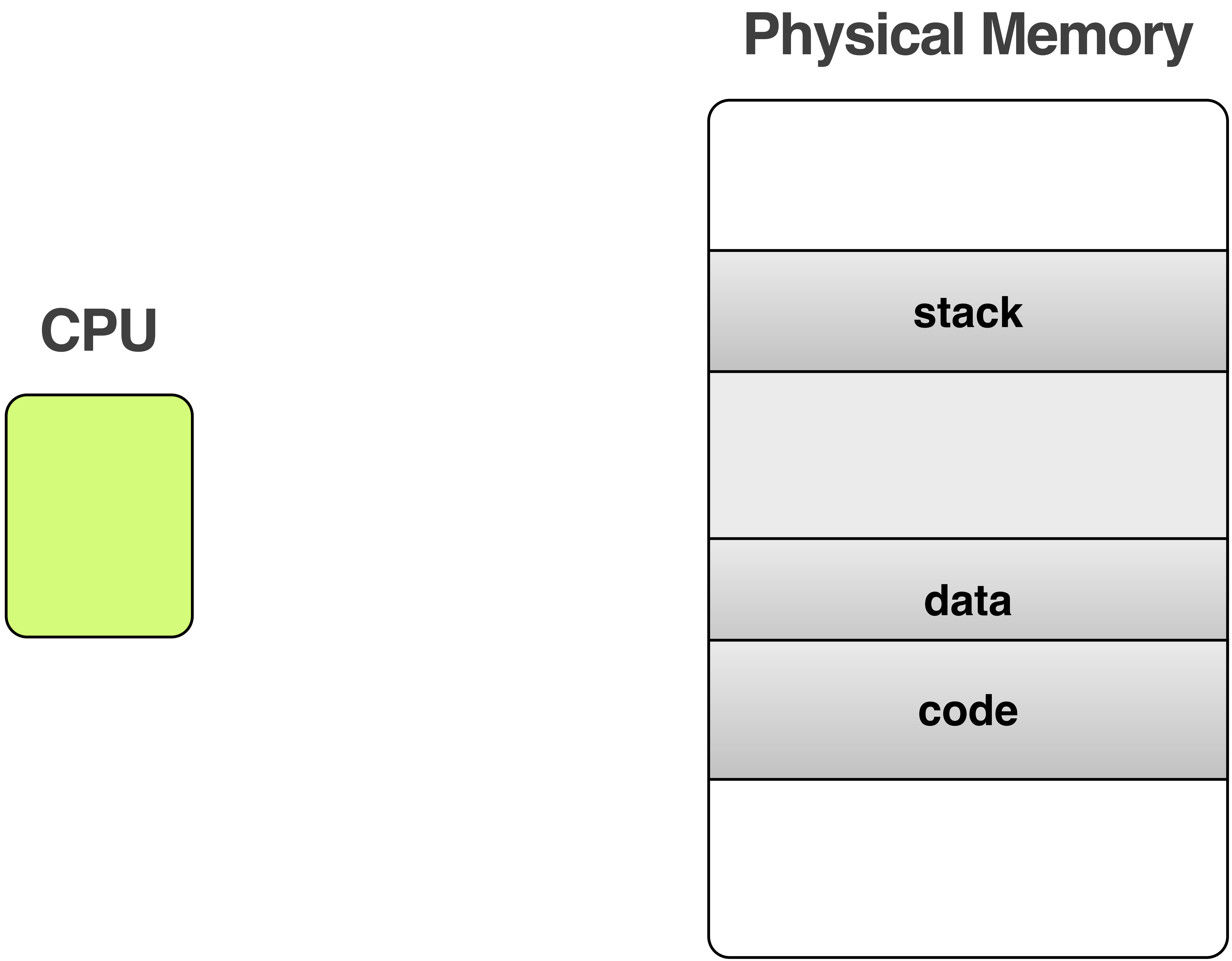
to build and run a program:




Memory
CPU
stack
data
code

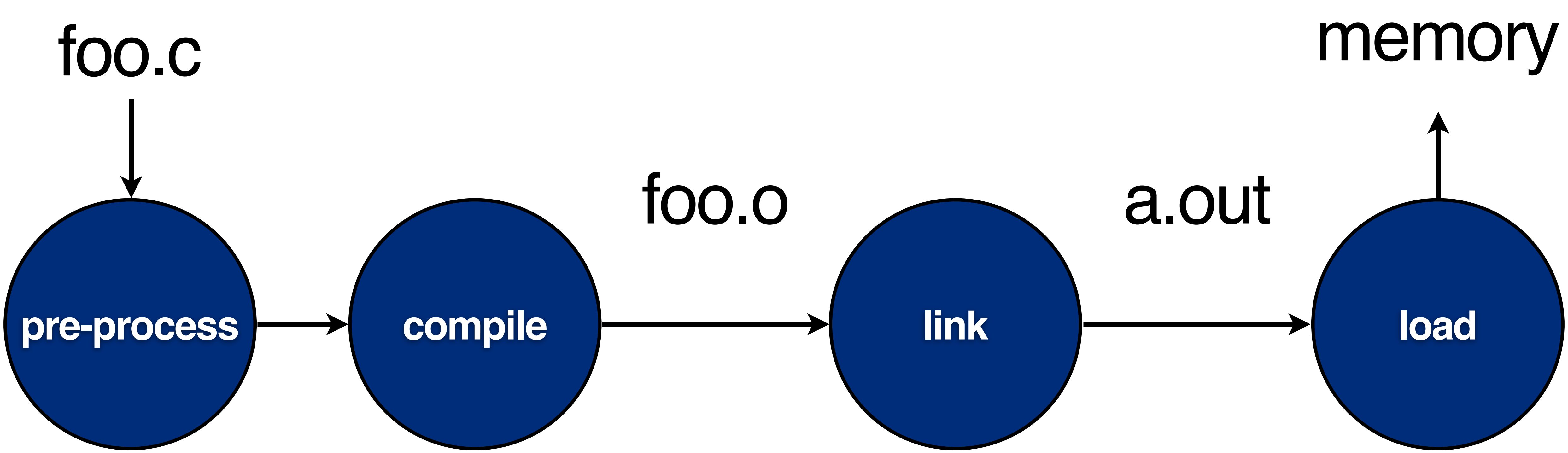
Design 1:
no address space abstraction.

processes access physical memory **directly**


Physical Memory
CPU
stack
data
code

```
x = x+1;
```

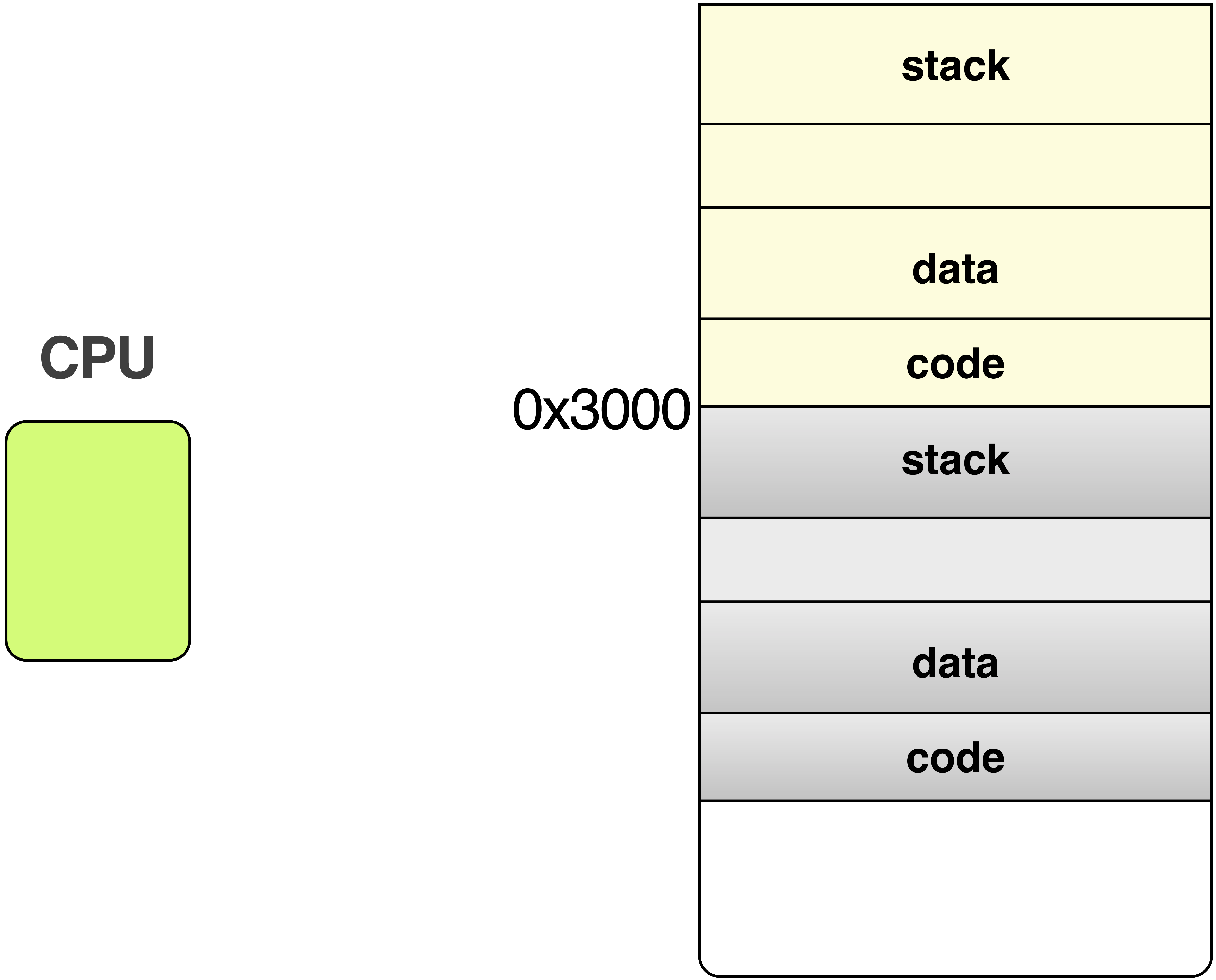
```
MOV R1, 0x001A
ADD R1
MOV 0x001A, R1
```



problem: only one process in memory at a time

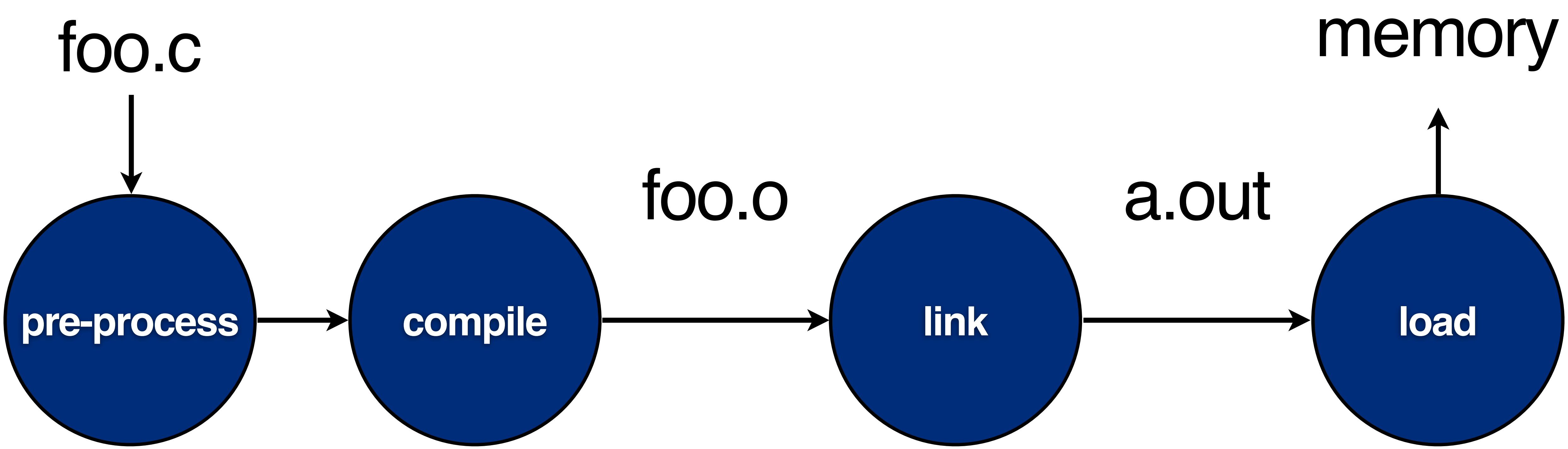
Design 2:
same as Design 1, but

statically relocate
process as it is loaded


CPU
0x3000
stack
data
code
stack
data
code

```
x = x+1;
```

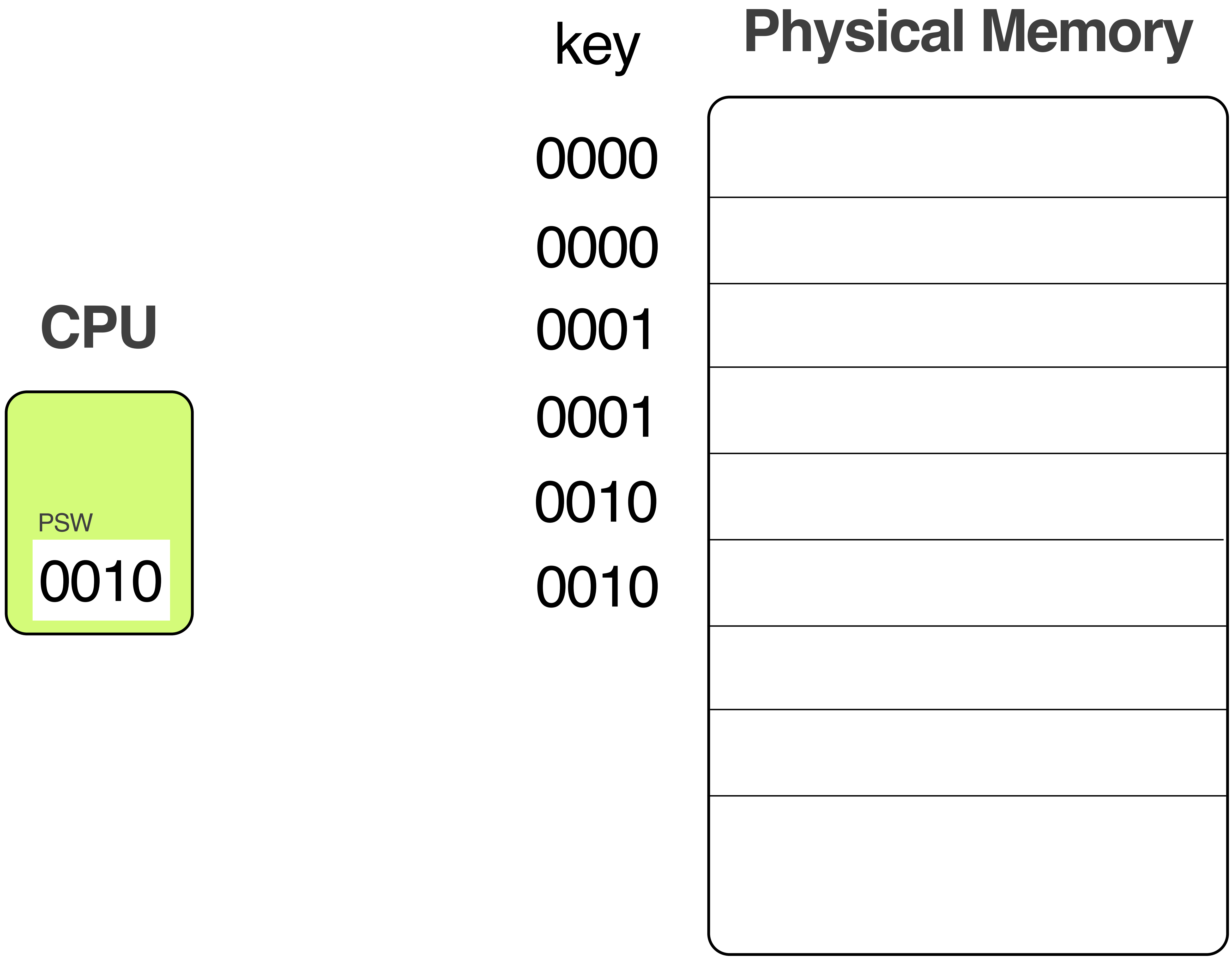
```
MOV R1, 0x001A
ADD R1
MOV 0x001A, R1
```




```
MOV R1, 0x301A
ADD R1
MOV 0x301A, R1
```

problem: no protection among processes

Design 3: memory protection through **key**-based access


key
Physical Memory
0000
0000
0001
0001
0010
0010
CPU
PSW
0010


LG
STEAM
Direct Drive 9kg
*Steam Cycle
Cotton
Cotton Quick
Synthetic
Baby Care
Delicate
Allergy Care
Hand Wash /Wool
Refresh
Quick 30
Spin
1400
Option
Rinse
Temp.

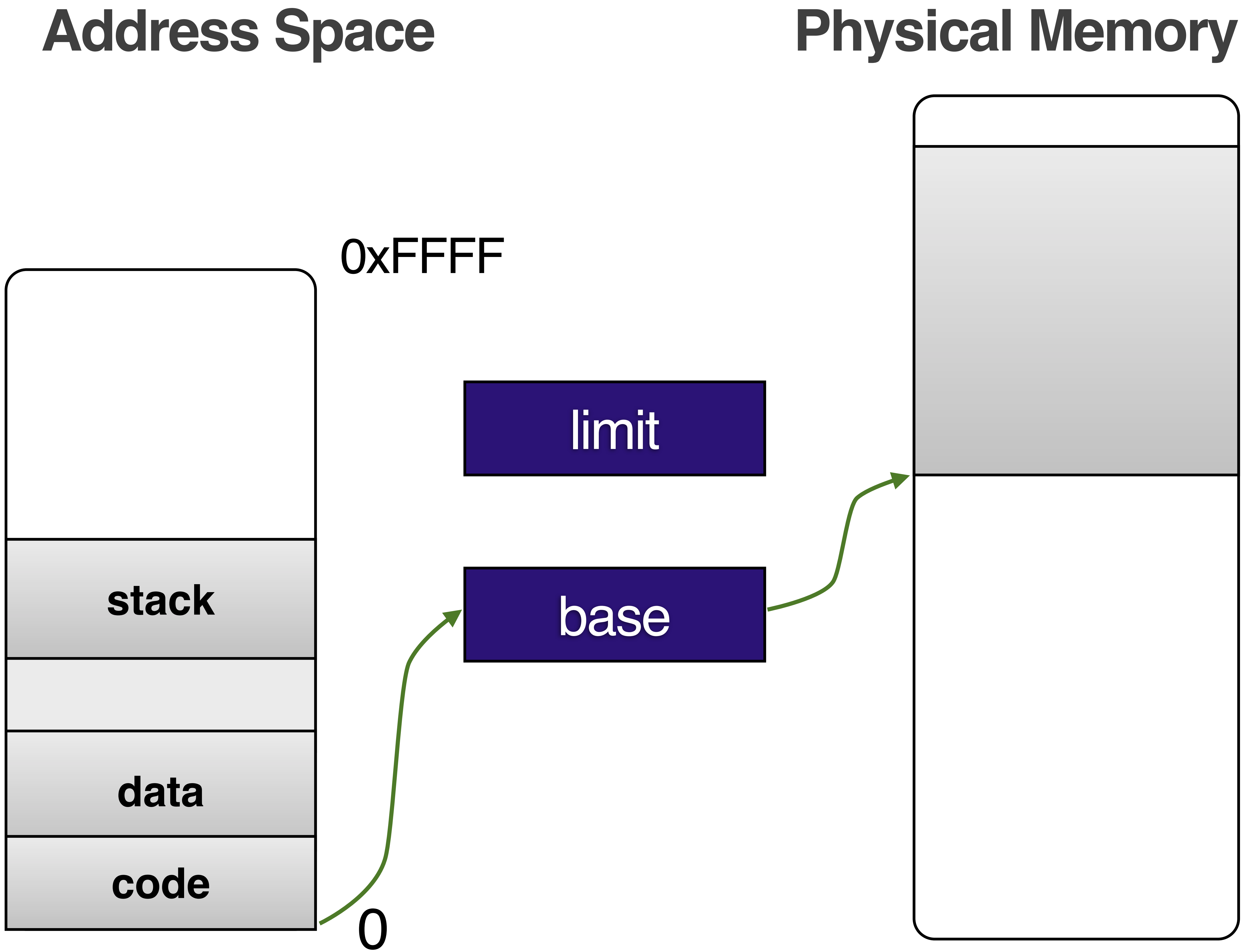

LG

problem:
loading and re-loading
is **slow**

Design 4: use **logical addresses** computed with **base** and **limit**

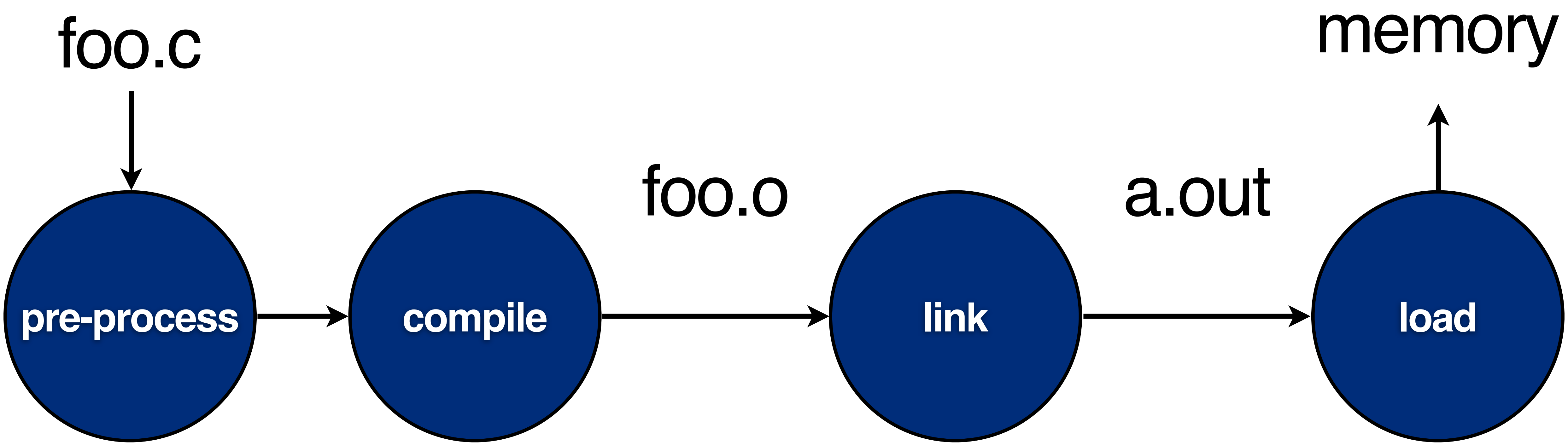
Address Space

Physical Memory


Address Space
Physical Memory
0xFFFF
limit
stack
base
data
code
0

```
x = x+1;
```

```
MOV R1, 0x001A
ADD R1
MOV 0x001A, R1
```




```
MOV base, 0x3000
MOV limit, 0x1000
```

problem:
add and compare for
every memory reference

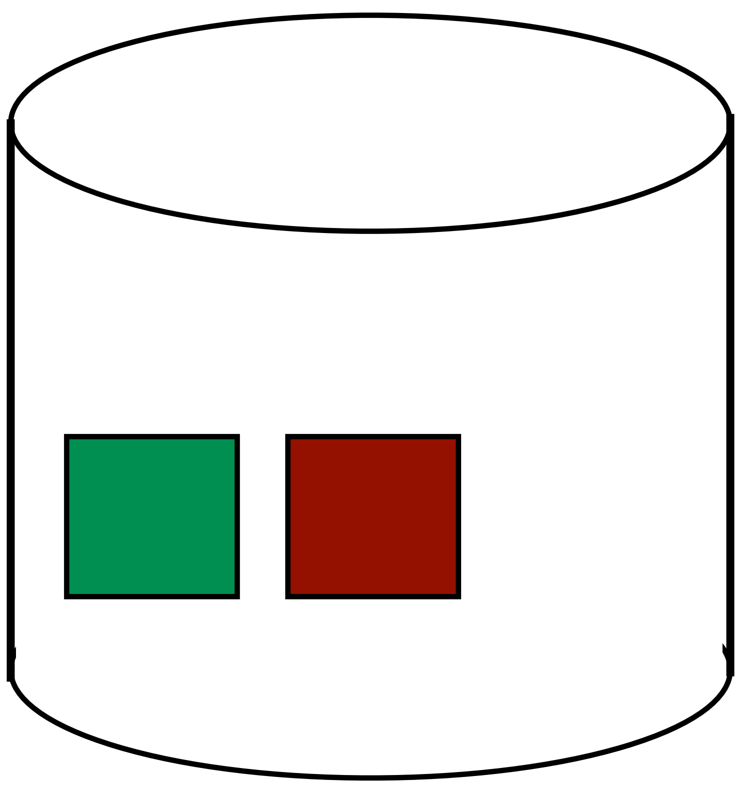
what if there is not enough memory to hold all processes?

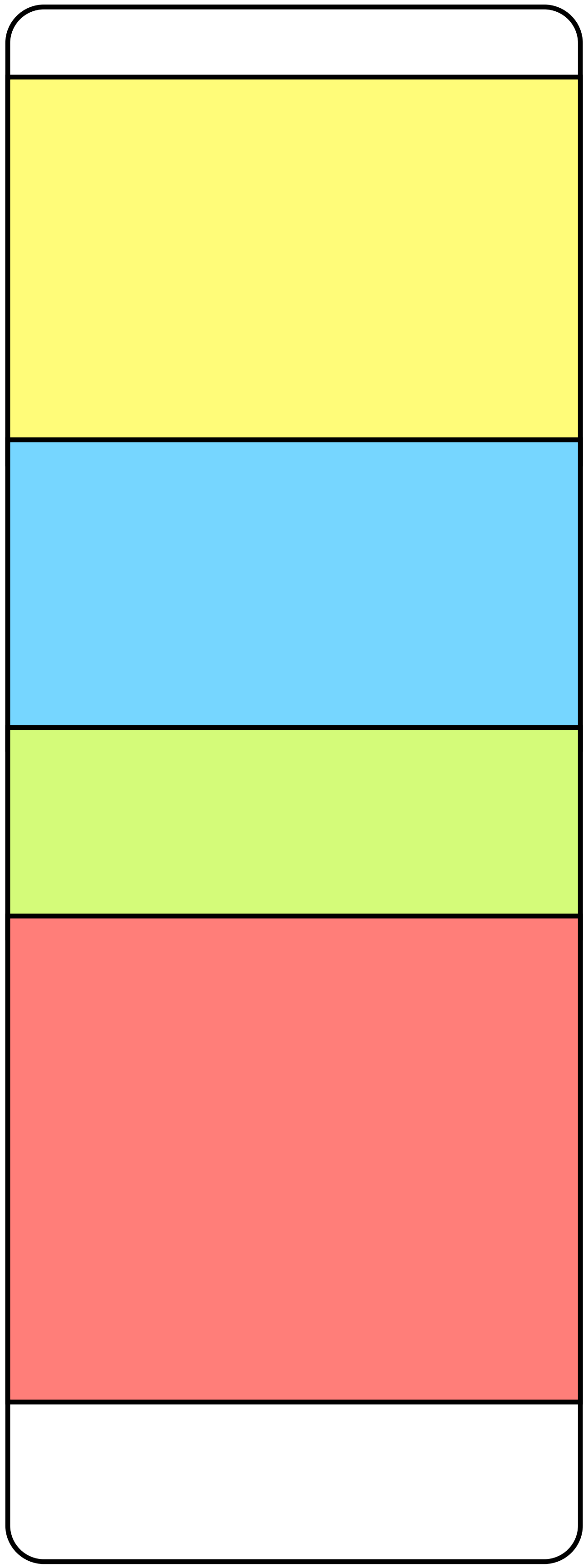
swapping

memory allocation to a process must be **contiguous**

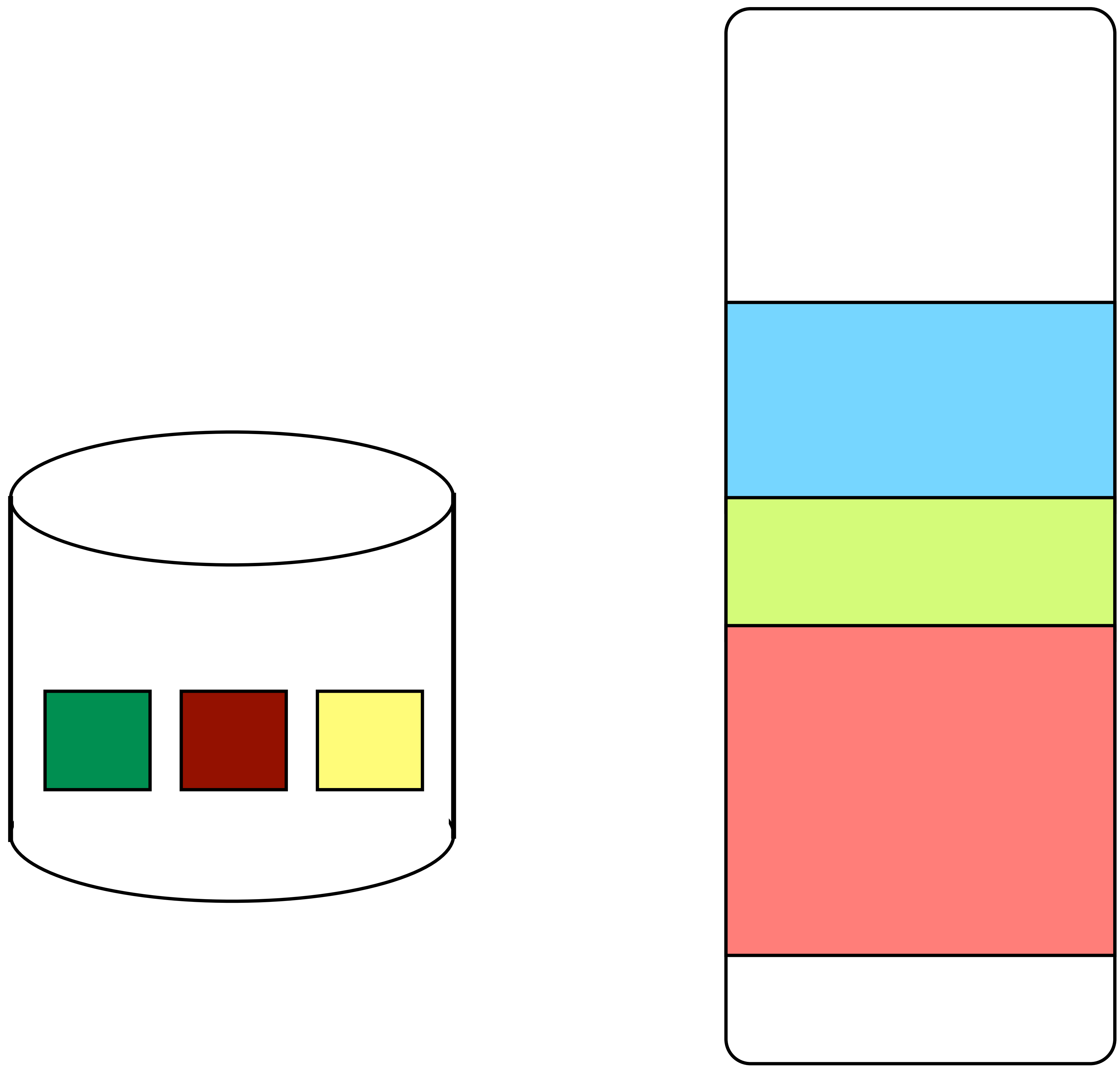
the **whole** process core image must be in memory

Physical Memory

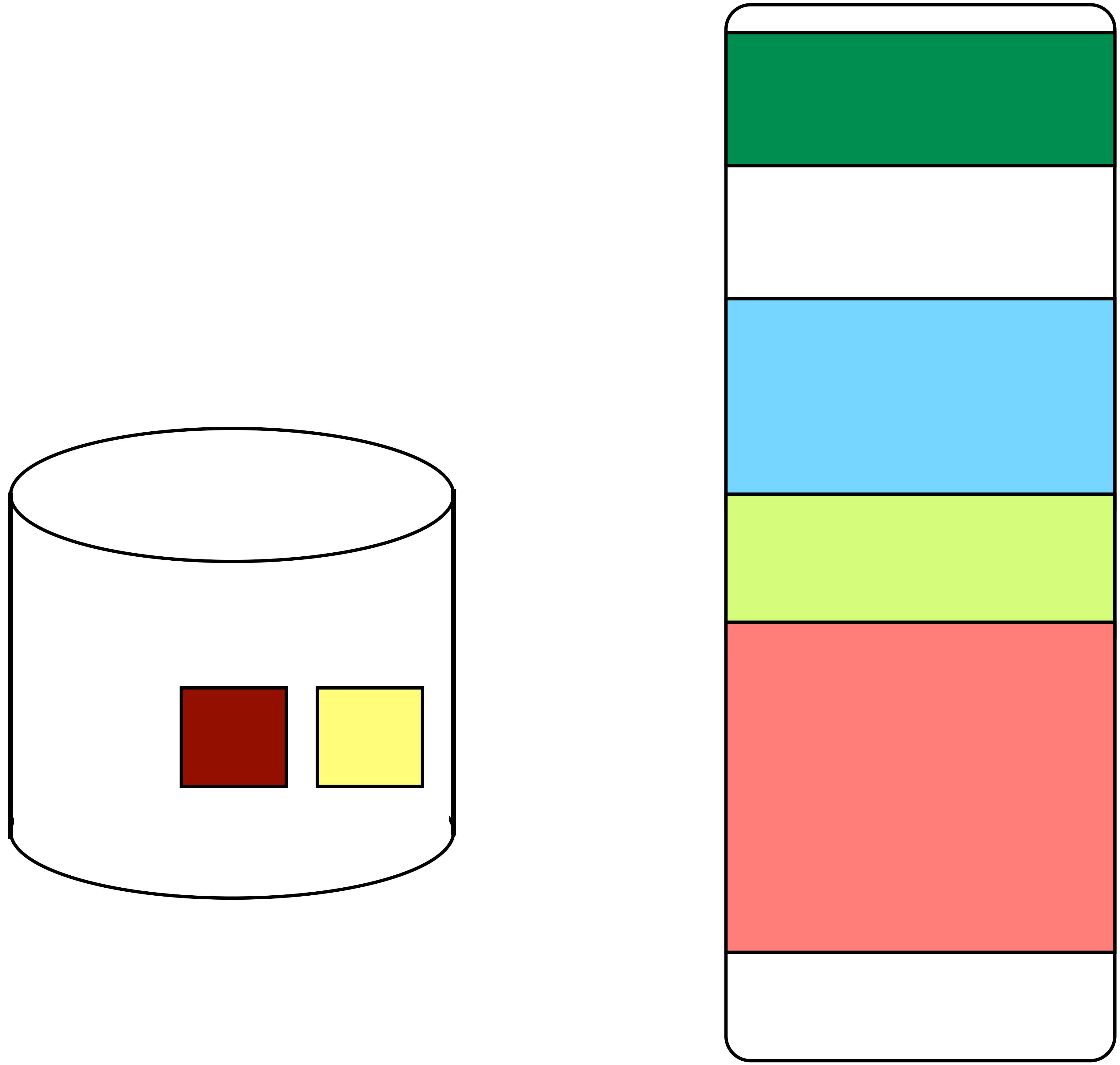




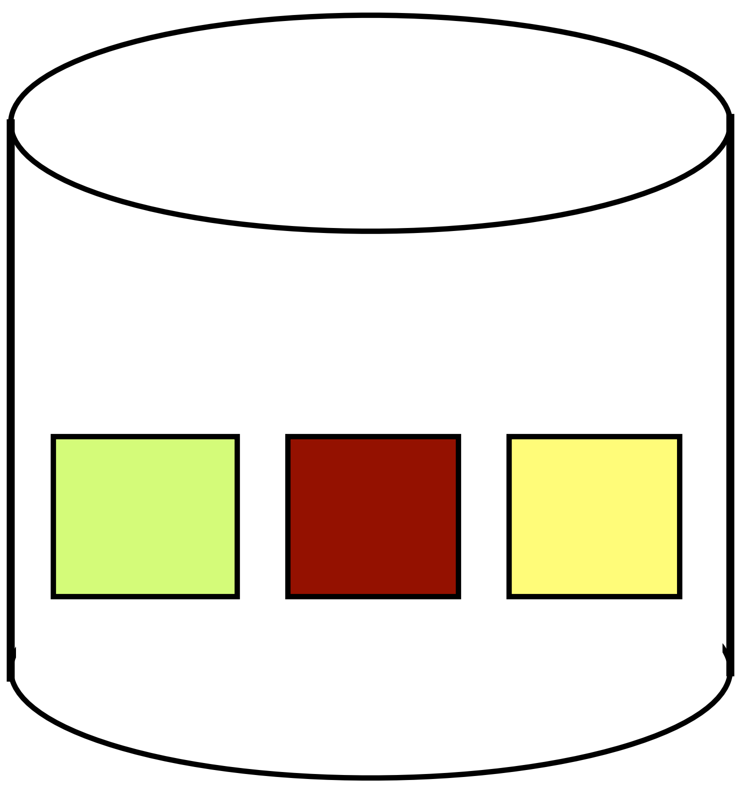
Physical Memory

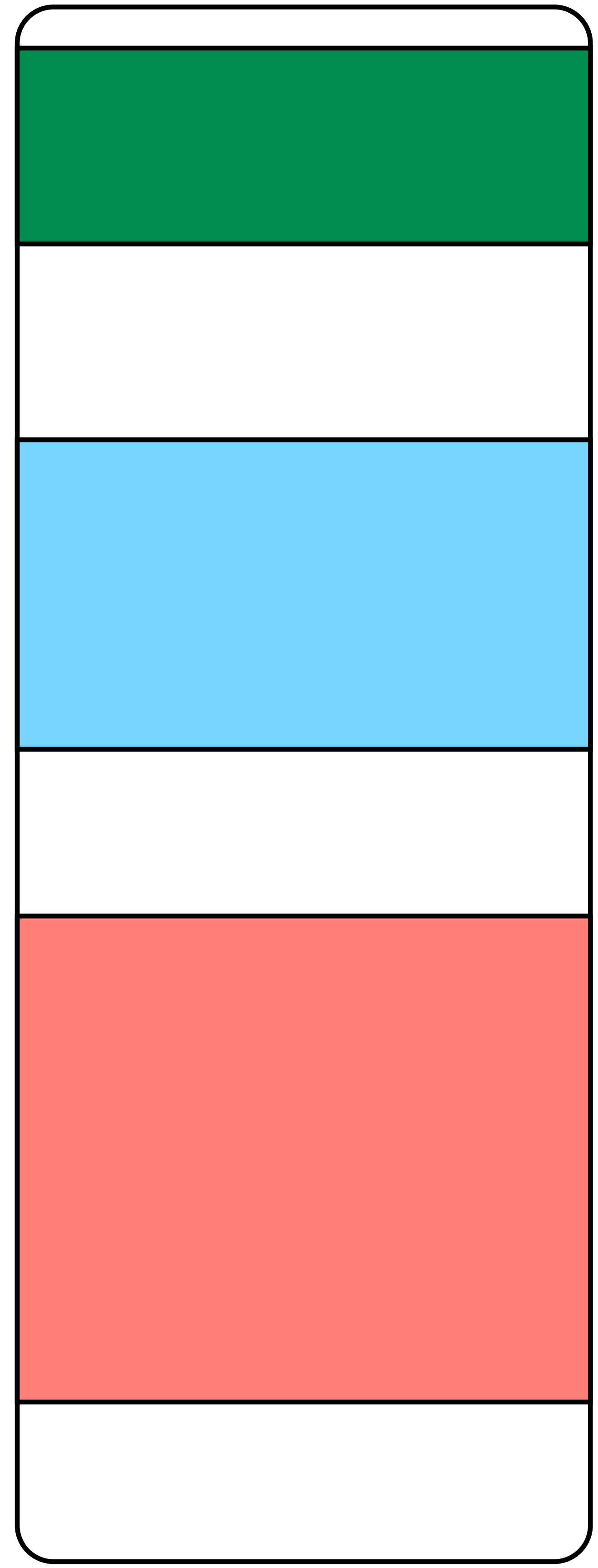


Physical Memory



Physical Memory

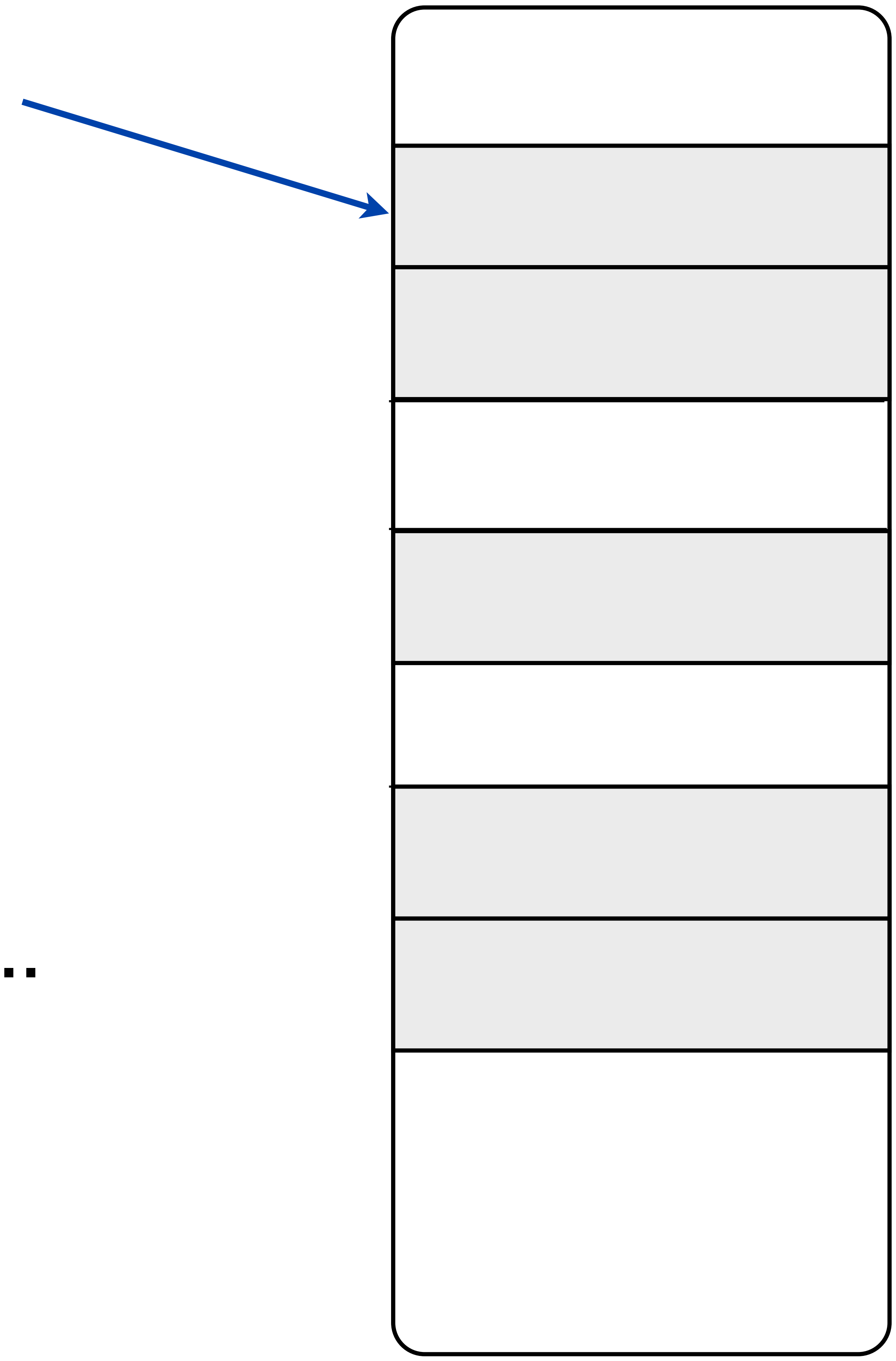




keep track of free/
occupied memory

Physical Memory

allocation unit



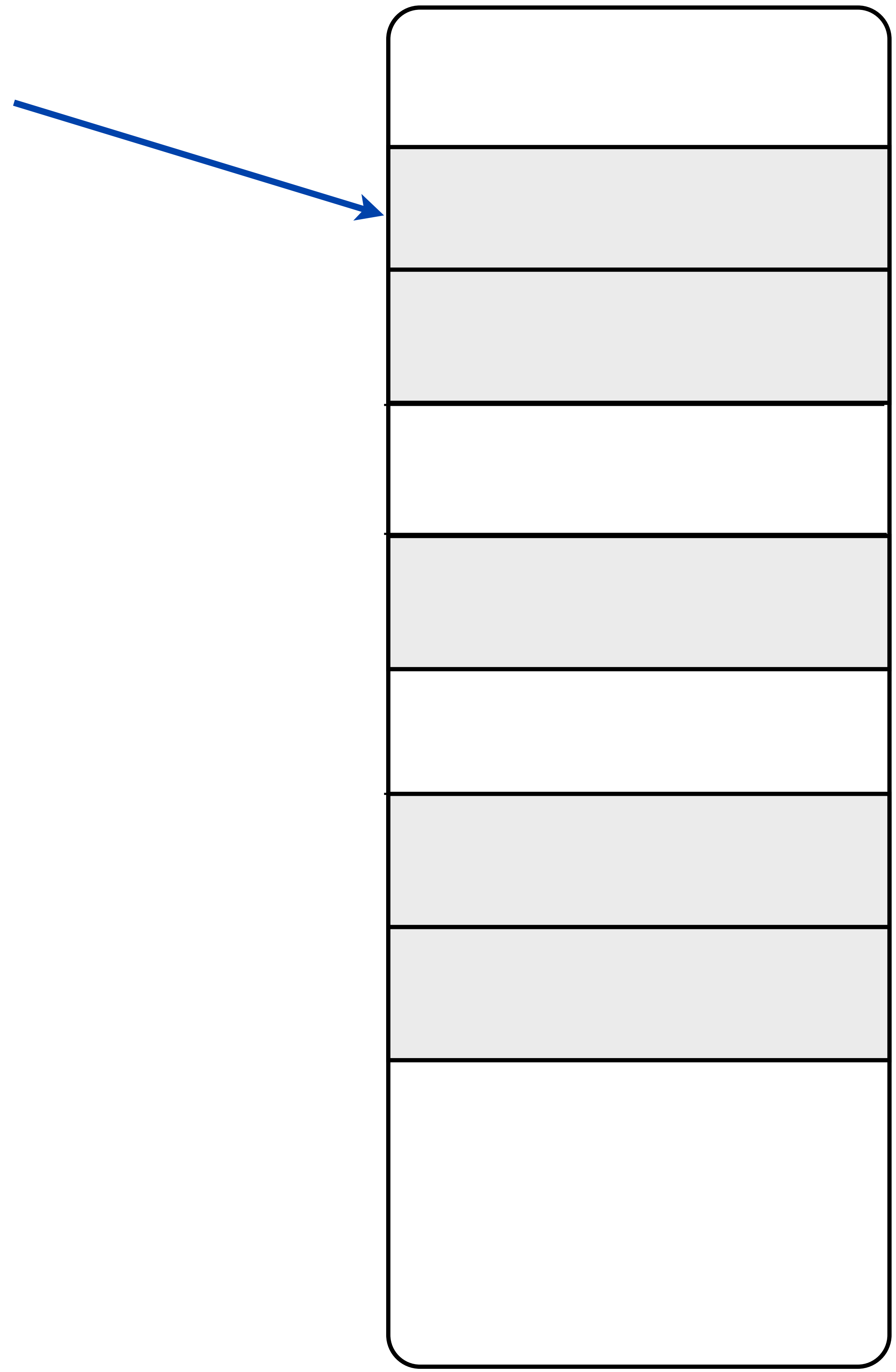
bitmap

01101011 ...

Physical Memory

allocation unit

linked list



which hole to assign
to a process?

first fit

next fit

best fit

worst fit

quick fit

large

vs.

small

allocation units

internal
and
external
fragmentation

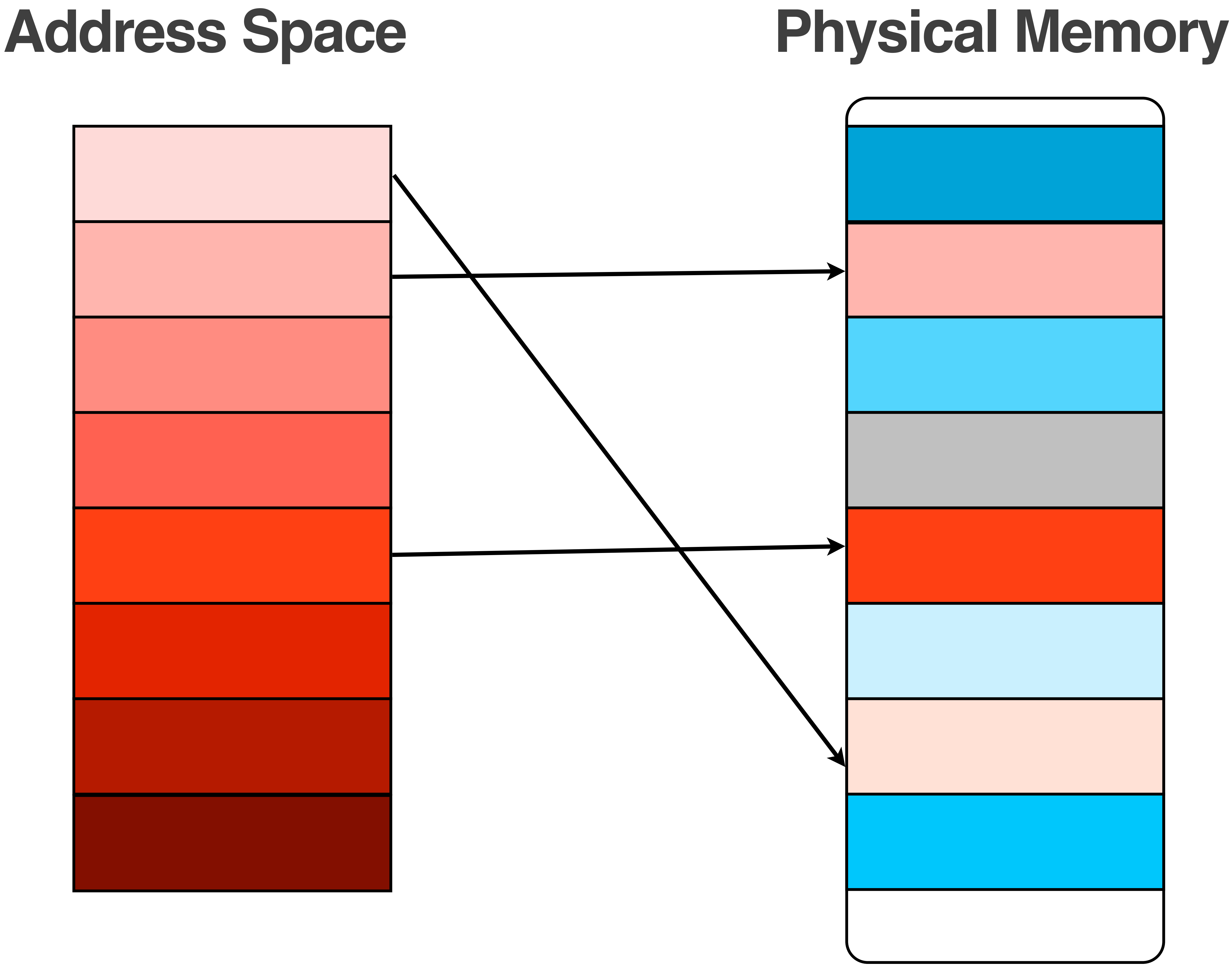
memory compaction
removes holes,
but is slow

Design 5:
virtual memory

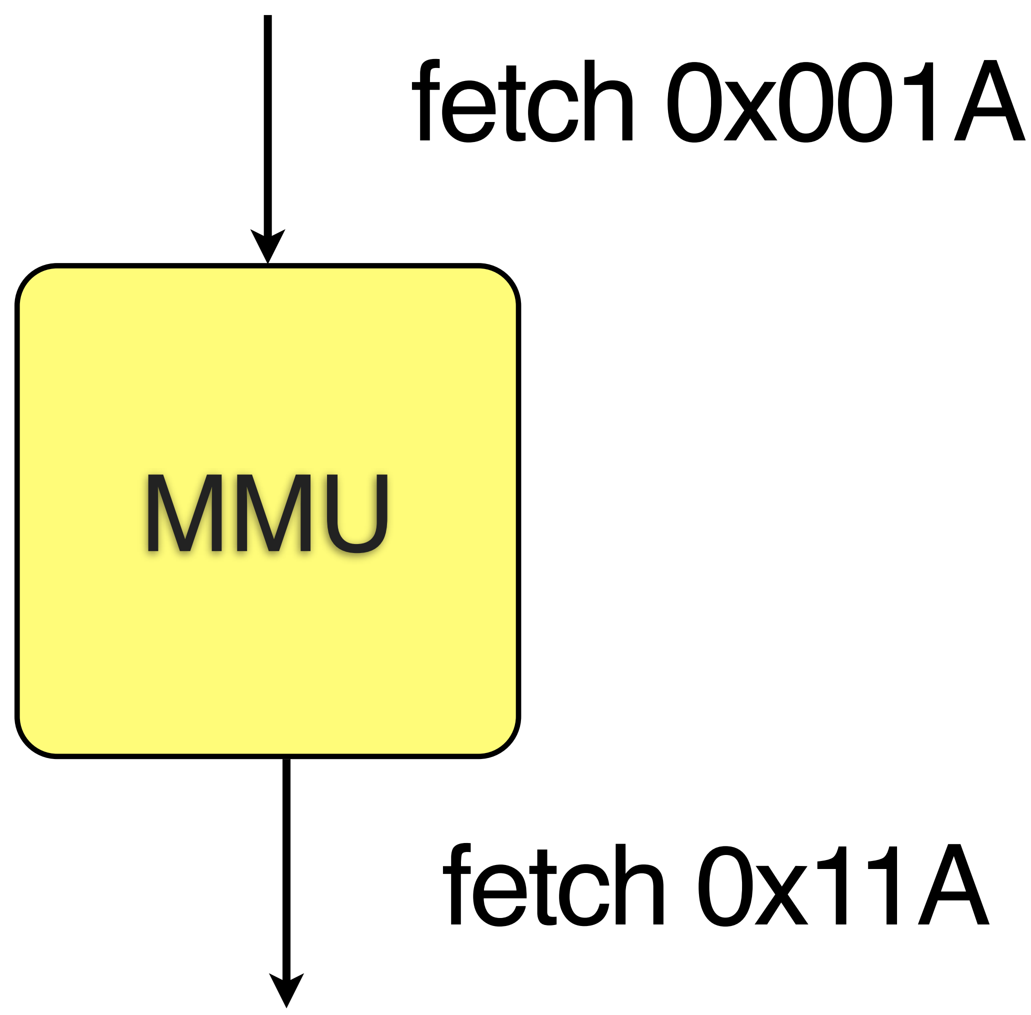
organize:

address space into **pages**

phy. memory into **frames**


Address Space
Physical Memory

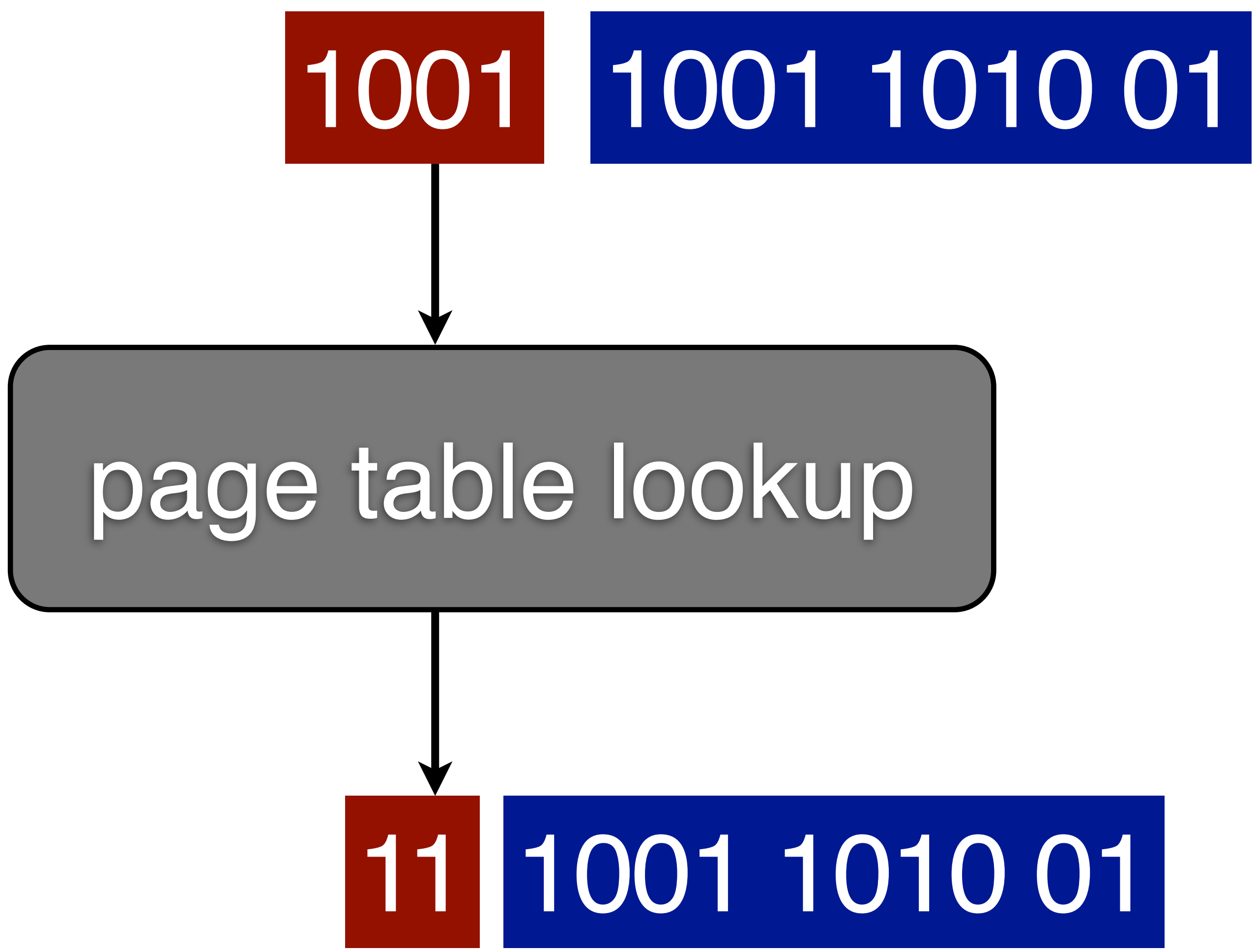
MOV R1, 0x001A



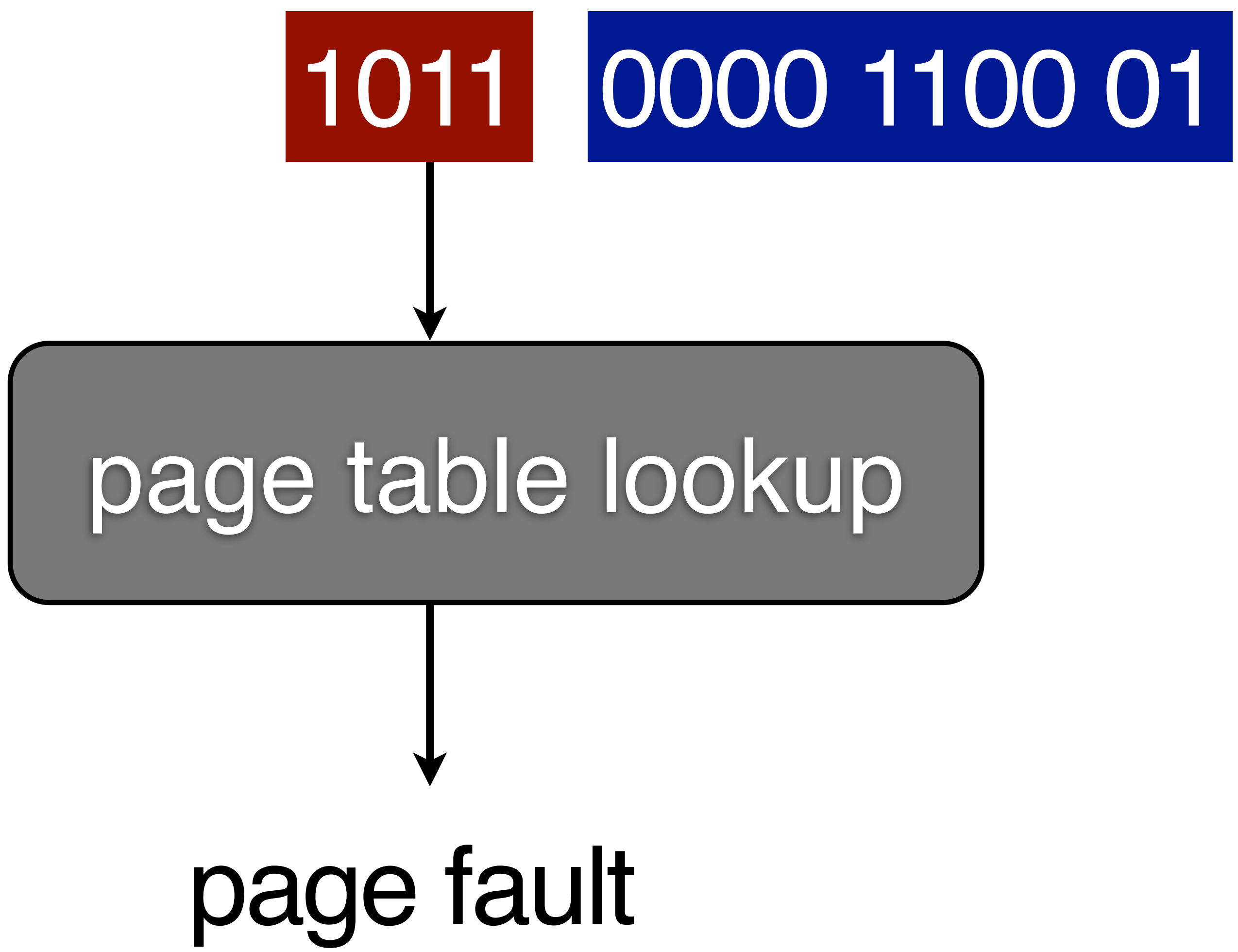
page table

page	frame	present?	dirty?	can write?	...
1	2	1	1	1	
2	7	1	0	0	
3	-	0	0	0	
:	:	:			

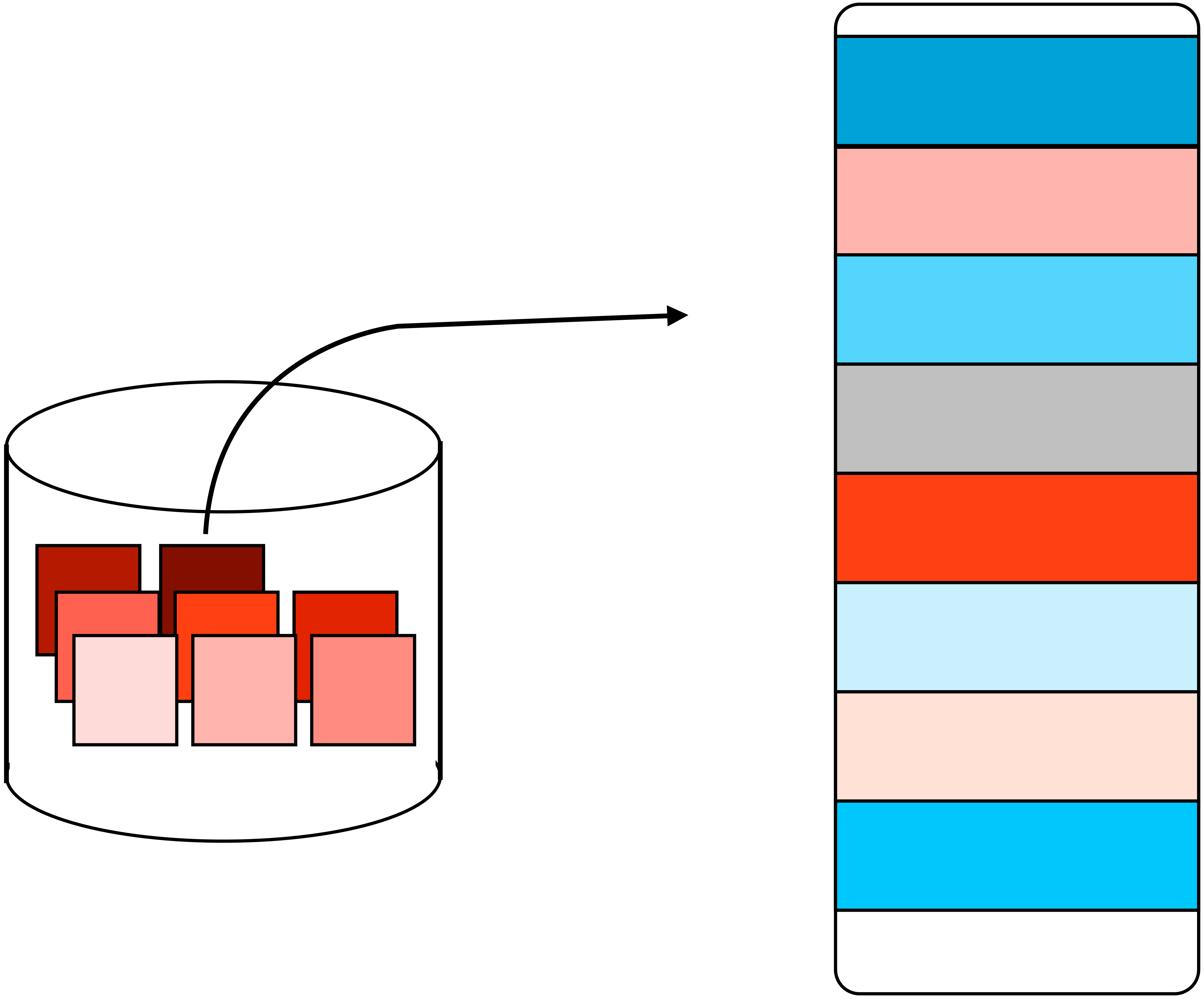
address translation



address translation



Physical Memory



64 bit addresses
4 MB page size
4 GB RAM
8 byte page table entry

1. address translation is **slow**
2. page table can be **huge**

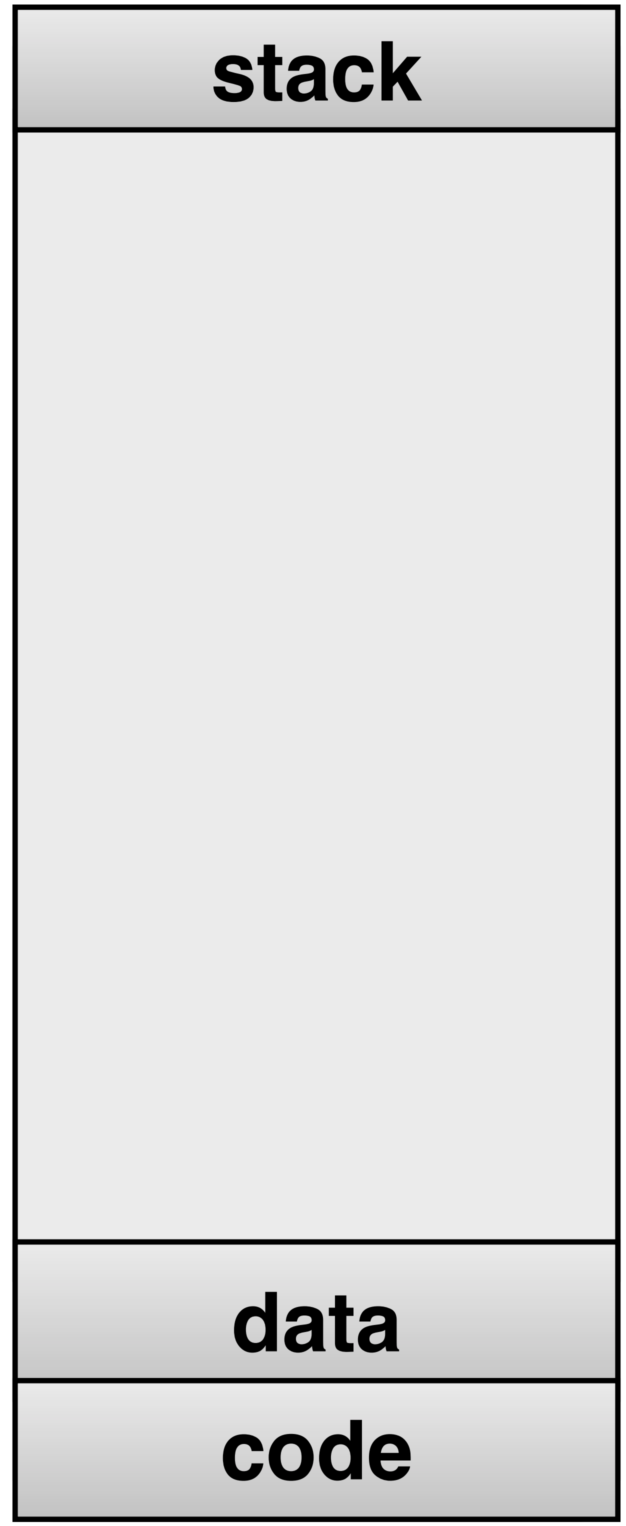
Translation Lookaside Buffer (TLB)

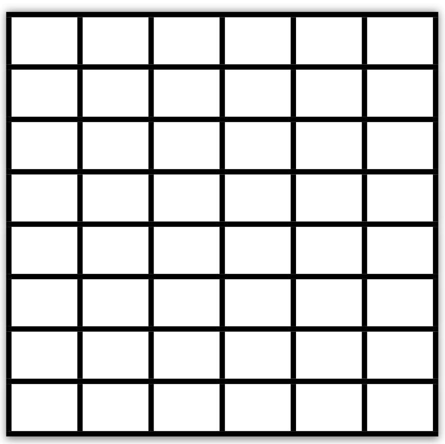
cache for page table entries

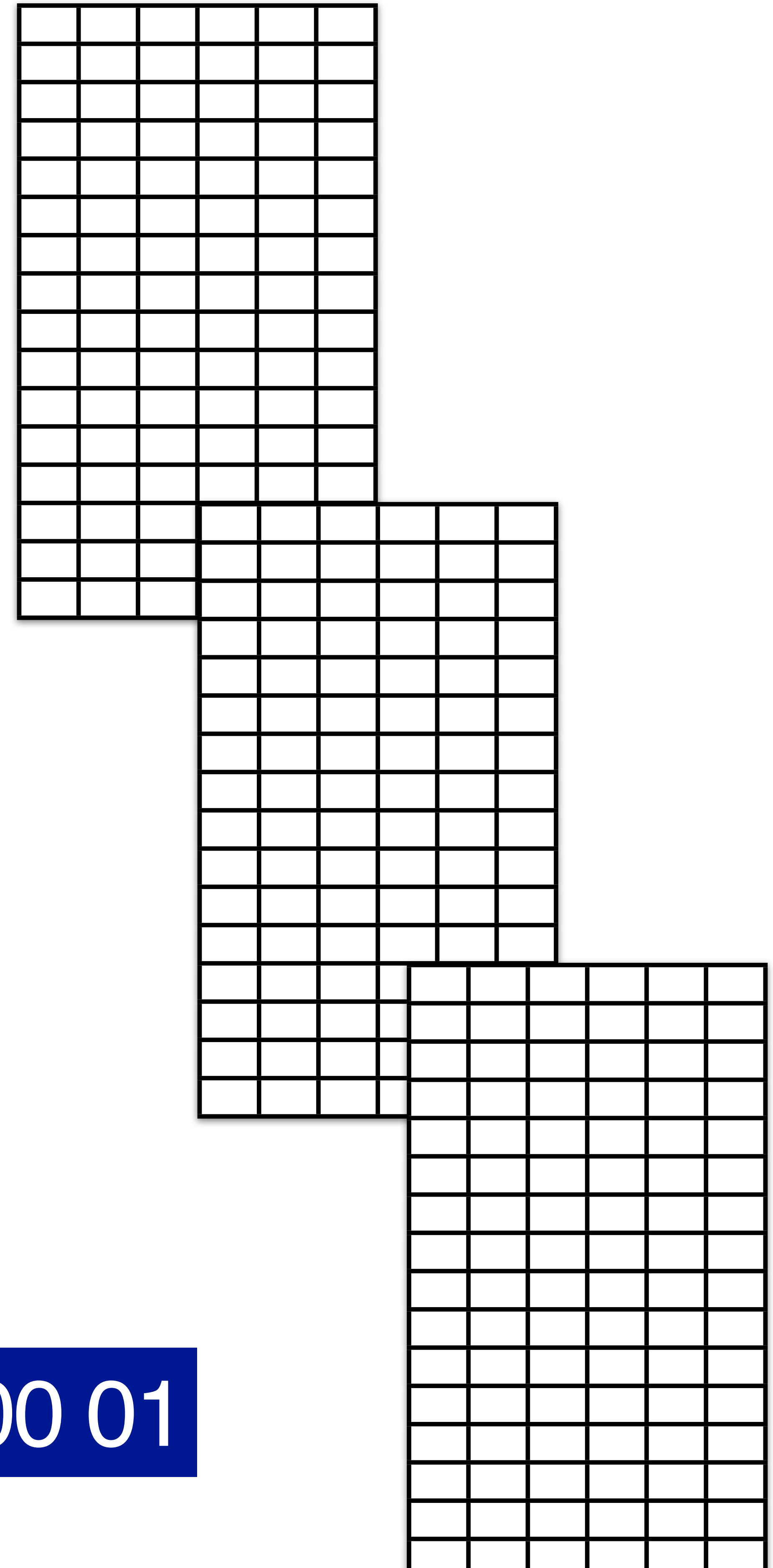
TLB
(hardware)

Page Table
(in memory)

Hierarchical Page Table


stack
data
code





101 0001 0000 1100 01

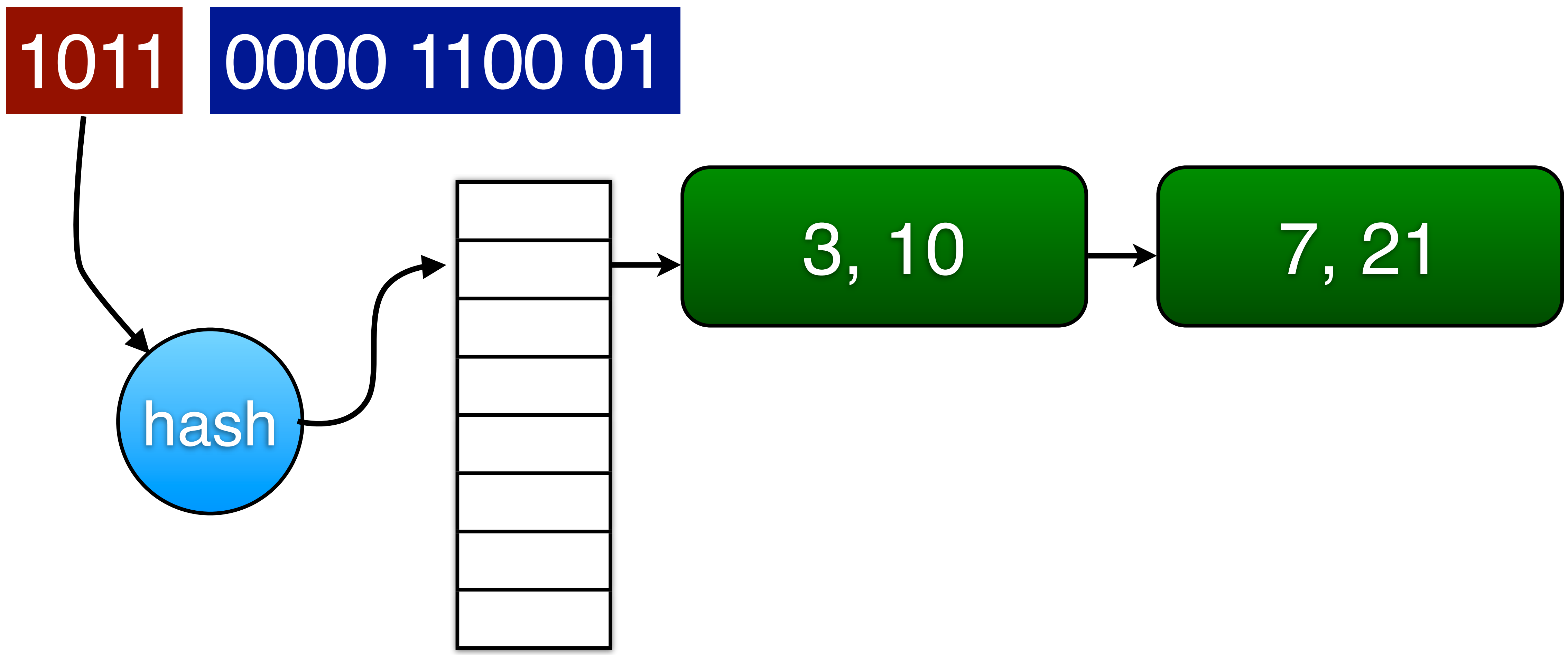
Inverted Page Table

inverted page table

frame	page		...
1	2		
2	7		
3	-		
:	:		

inverted page table

(using hash table)



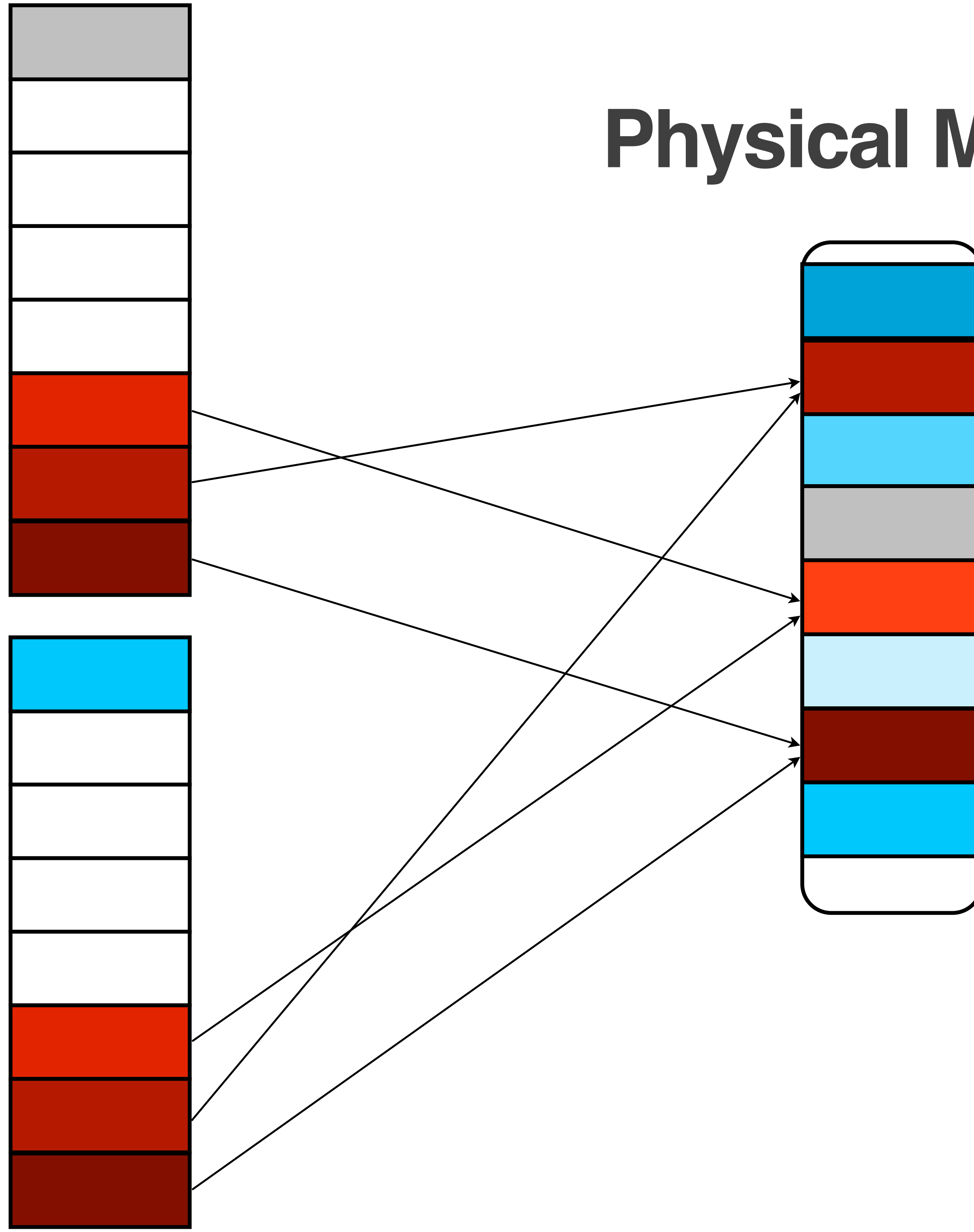
Sharing Pages

Sharing Code

(e.g, when running the same program)

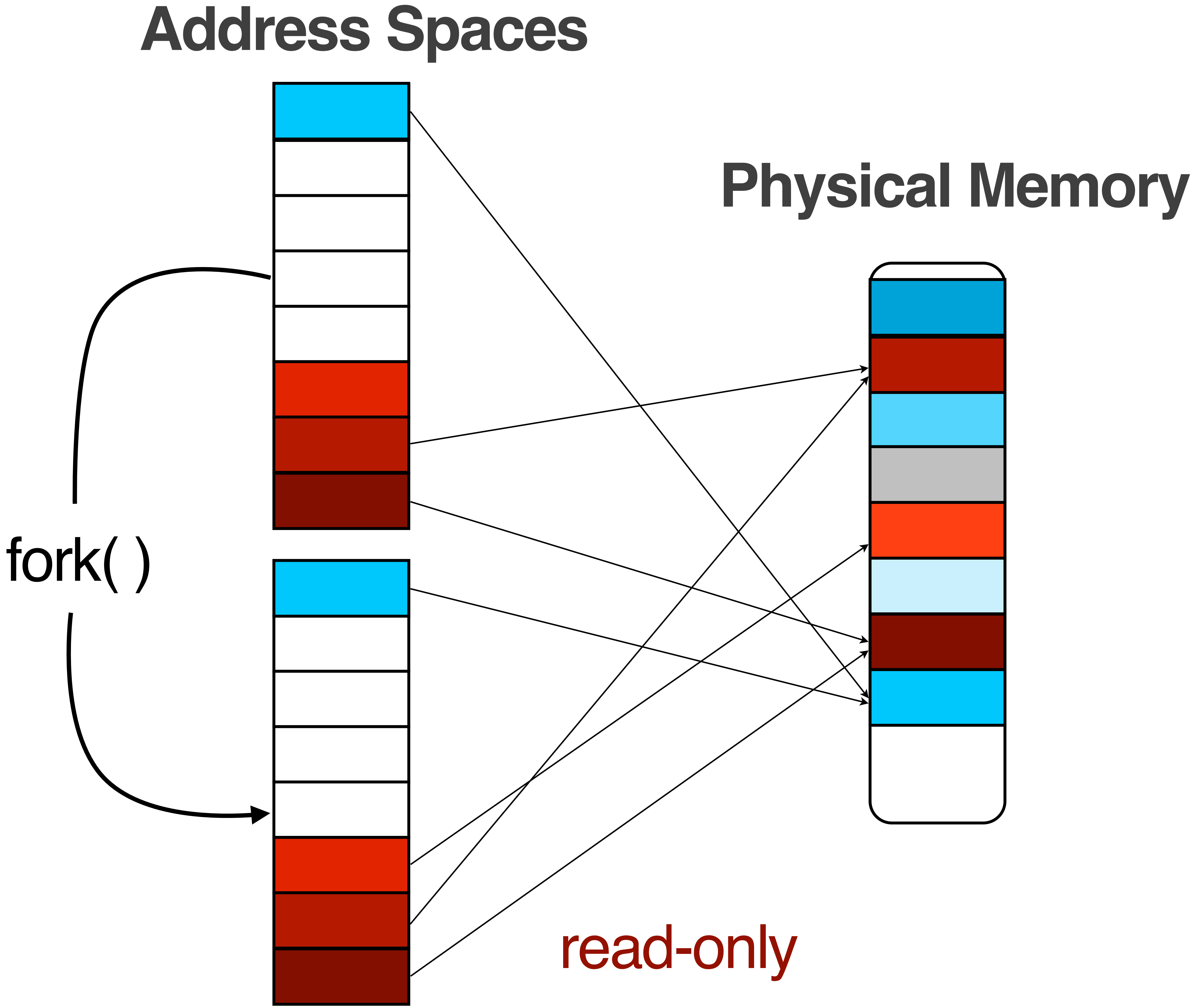
Address Spaces

Physical Memory

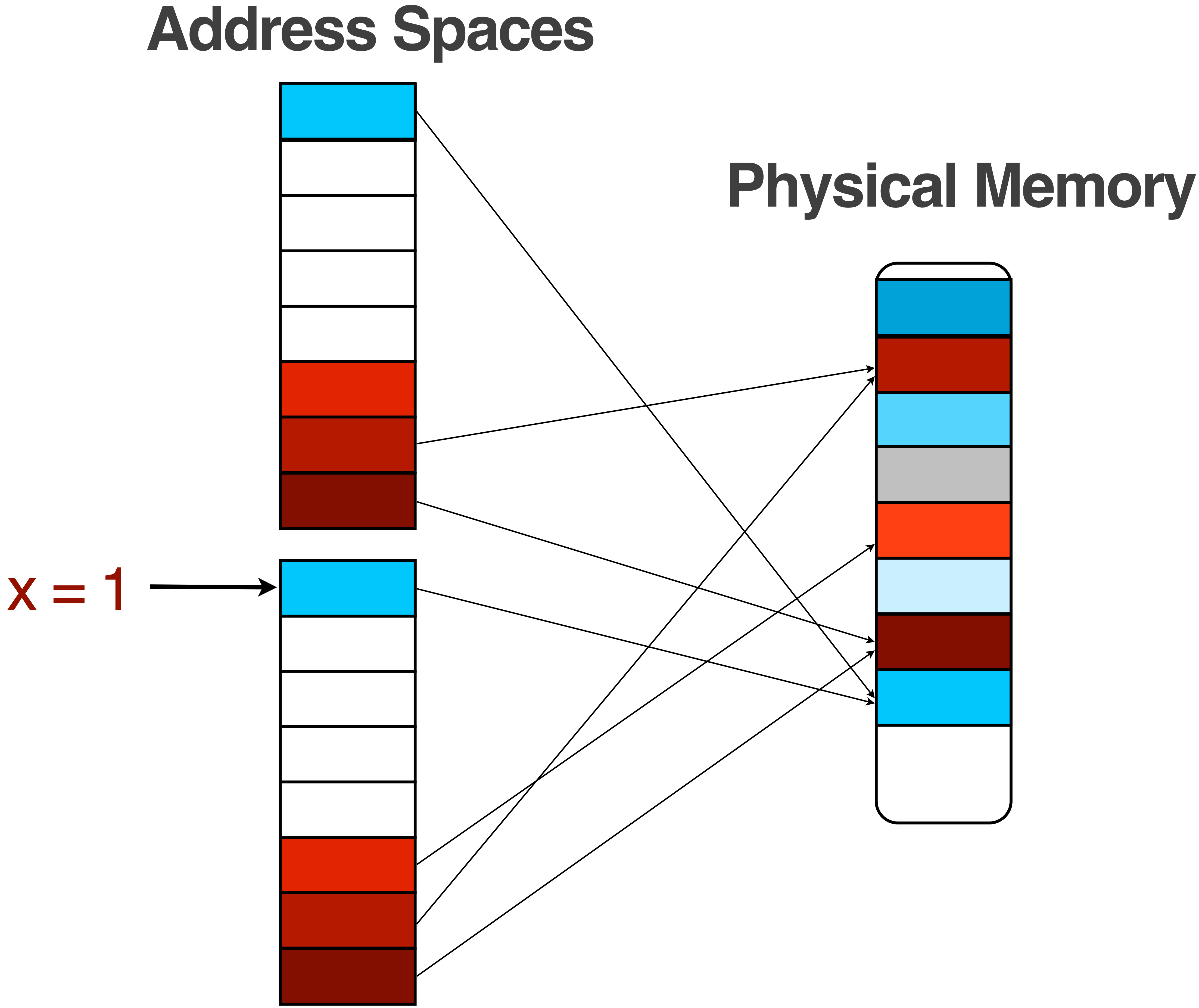


Sharing Data

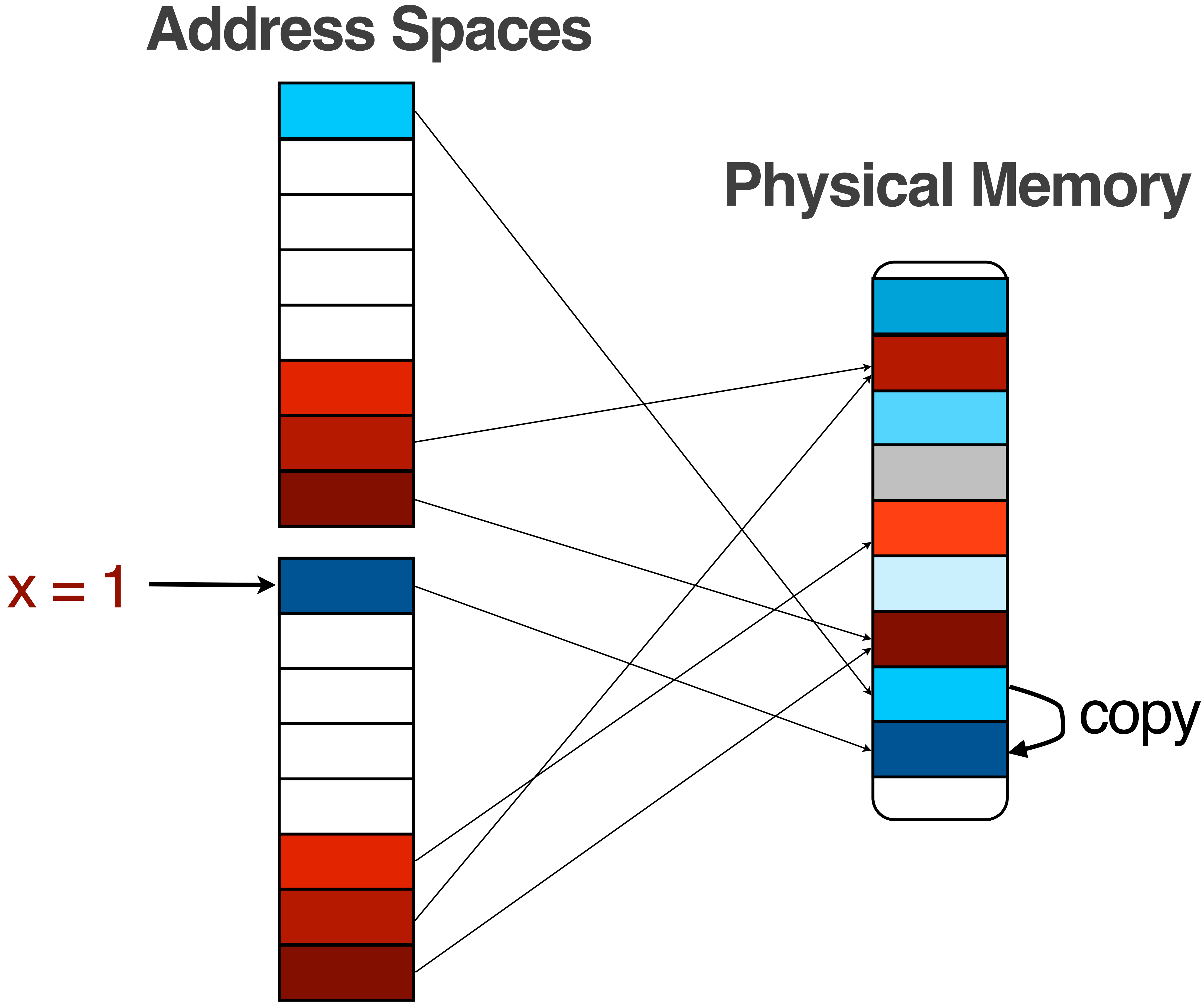
(e.g., shared memory, copy-on-write)



Address Spaces
Physical Memory
fork( )
read-only



Address Spaces
Physical Memory
x = 1


Address Spaces
Physical Memory
x = 1
copy