Operating Systems
Memory Virtualisation
Segmentation + Paging

Nipun Batra

Revision

Revision

1. Base & Bounds for Relocation

Revision

1. Base & Bounds for Relocation
    1. Dynamic

Revision

1. Base & Bounds for Relocation
    1. Dynamic
    2. Base - Start of address space in physical memory

Revision

1. Base & Bounds for Relocation
    1. Dynamic
    2. Base - Start of address space in physical memory
    3. Bounds - Range

Revision

1. Base & Bounds for Relocation
    1. Dynamic
    2. Base - Start of address space in physical memory
    3. Bounds - Range
    4. VA starts with address?

Revision

1. Base & Bounds for Relocation
    1. Dynamic
    2. Base - Start of address space in physical memory
    3. Bounds - Range
    4. VA starts with address?
    5. When is base & bounds setup?

Revision

1. Base & Bounds for Relocation
    1. Dynamic
    2. Base - Start of address space in physical memory
    3. Bounds - Range
    4. VA starts with address?
    5. When is base & bounds setup?
    6. What happens if VA > bounds?

Revision

1. Base & Bounds for Relocation
    1. Dynamic
    2. Base - Start of address space in physical memory
    3. Bounds - Range
    4. VA starts with address?
    5. When is base & bounds setup?
    6. What happens if VA > bounds?
    7. Pros - fast?

Revision

1. Base & Bounds for Relocation
    1. Dynamic
    2. Base - Start of address space in physical memory
    3. Bounds - Range
    4. VA starts with address?
    5. When is base & bounds setup?
    6. What happens if VA > bounds?
    7. Pros - fast?
    8. Cons - need _____ block of memory, leads to:

Revision

1. Base & Bounds for Relocation
    1. Dynamic
    2. Base - Start of address space in physical memory
    3. Bounds - Range
    4. VA starts with address?
    5. When is base & bounds setup?
    6. What happens if VA > bounds?
    7. Pros - fast?
    8. Cons - need _____ block of memory, leads to:
        1. Internal fragmentation

Revision

1. Base & Bounds for Relocation
    1. Dynamic
    2. Base - Start of address space in physical memory
    3. Bounds - Range
    4. VA starts with address?
    5. When is base & bounds setup?
    6. What happens if VA > bounds?
    7. Pros - fast?
    8. Cons - need _____ block of memory, leads to:
        1. Internal fragmentation
        2. External fragmentation

1. Segmentation

1. Segmentation
    1. Motivation - overcome some of the cons of Base & Bounds

1. Segmentation
    1. Motivation - overcome some of the cons of Base & Bounds
    2. Each segment has its own base & bounds

1. Segmentation
    1. Motivation - overcome some of the cons of Base & Bounds
    2. Each segment has its own base & bounds
    3. Total # of registers required = _____

1. Segmentation
    1. Motivation - overcome some of the cons of Base & Bounds
    2. Each segment has its own base & bounds
    3. Total # of registers required = _____
    4. Each segment can have its own :

1. Segmentation
    1. Motivation - overcome some of the cons of Base & Bounds
    2. Each segment has its own base & bounds
    3. Total # of registers required = _____
    4. Each segment can have its own :
        1. _____

1. Segmentation
    1. Motivation - overcome some of the cons of Base & Bounds
    2. Each segment has its own base & bounds
    3. Total # of registers required = _____
    4. Each segment can have its own :
        1. _____
        2. _____

1. Segmentation
    1. Motivation - overcome some of the cons of Base & Bounds
    2. Each segment has its own base & bounds
    3. Total # of registers required = _____
    4. Each segment can have its own :
        1. _____
        2. _____
        3. _____

1. Segmentation
    1. Motivation - overcome some of the cons of Base & Bounds
    2. Each segment has its own base & bounds
    3. Total # of registers required = _____
    4. Each segment can have its own :
        1. _____
        2. _____
        3. _____
5. Check: Segment Start < V.A. < Segment Start + Segment Bound.

1. Segmentation
    1. Motivation - overcome some of the cons of Base & Bounds
    2. Each segment has its own base & bounds
    3. Total # of registers required = _____
    4. Each segment can have its own :
        1. _____
        2. _____
        3. _____
    5. Check: Segment Start < V.A. < Segment Start + Segment Bound.
    6. Translate: Physical Address = (V.A. - Segment Start) + Segment Base Pros - fast?

1. Segmentation
    1. Motivation - overcome some of the cons of Base & Bounds
    2. Each segment has its own base & bounds
    3. Total # of registers required = _____
    4. Each segment can have its own :
        1. _____
        2. _____
        3. _____
    5. Check: Segment Start < V.A. < Segment Start + Segment Bound.
    6. Translate: Physical Address = (V.A. - Segment Start) + Segment Base Pros - fast?
    7. Cons - need _____ block of memory, leads to:

1. Segmentation
    1. Motivation - overcome some of the cons of Base & Bounds
    2. Each segment has its own base & bounds
    3. Total # of registers required = _____
    4. Each segment can have its own :
        1. _____
        2. _____
        3. _____
    5. Check: Segment Start < V.A. < Segment Start + Segment Bound.
    6. Translate: Physical Address = (V.A. - Segment Start) + Segment Base Pros - fast?
    7. Cons - need _____ block of memory, leads to:
        1. Internal fragmentation

1. Segmentation
    1. Motivation - overcome some of the cons of Base & Bounds
    2. Each segment has its own base & bounds
    3. Total # of registers required = _____
    4. Each segment can have its own :
        1. _____
        2. _____
        3. _____
    5. Check: Segment Start < V.A. < Segment Start + Segment Bound.
    6. Translate: Physical Address = (V.A. - Segment Start) + Segment Base Pros - fast?
    7. Cons - need _____ block of memory, leads to:
        1. Internal fragmentation
        2. External fragmentation

Segmentation Example

Kernel

CPU

MMU

Physical Memory

Virtual Address

0x100

Segmentation Example

Kernel

CPU

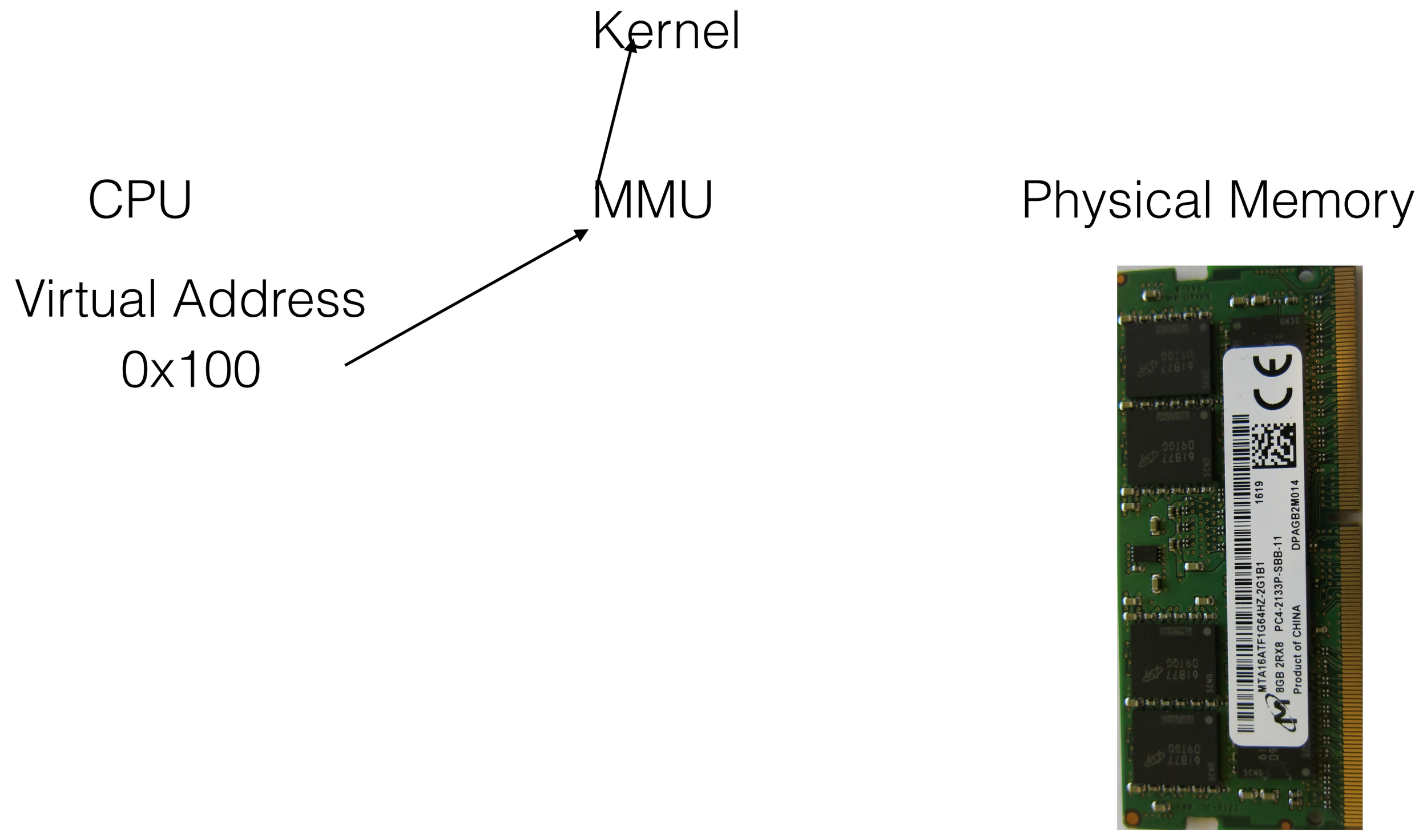
MMU

Physical Memory

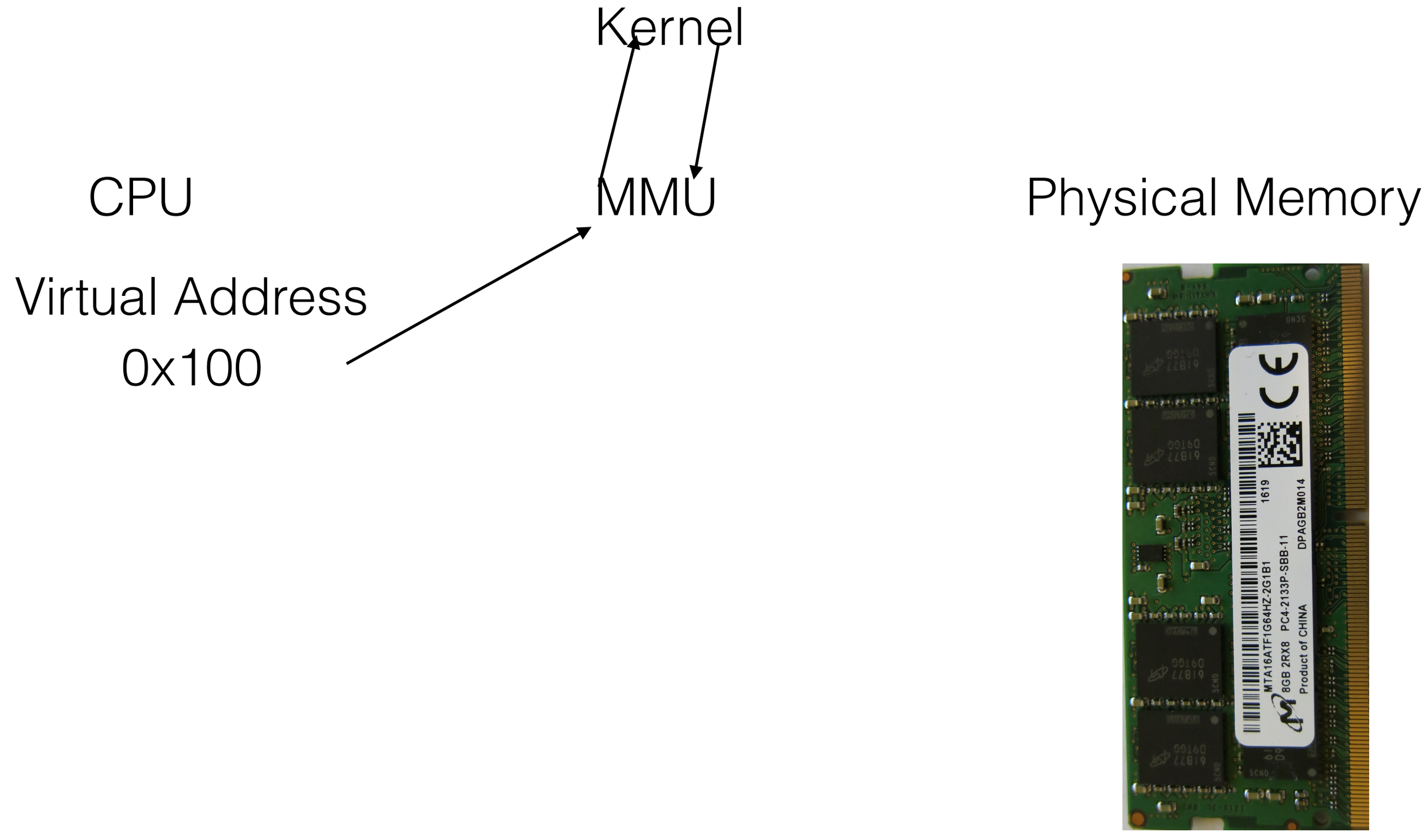
Virtual Address

0x100

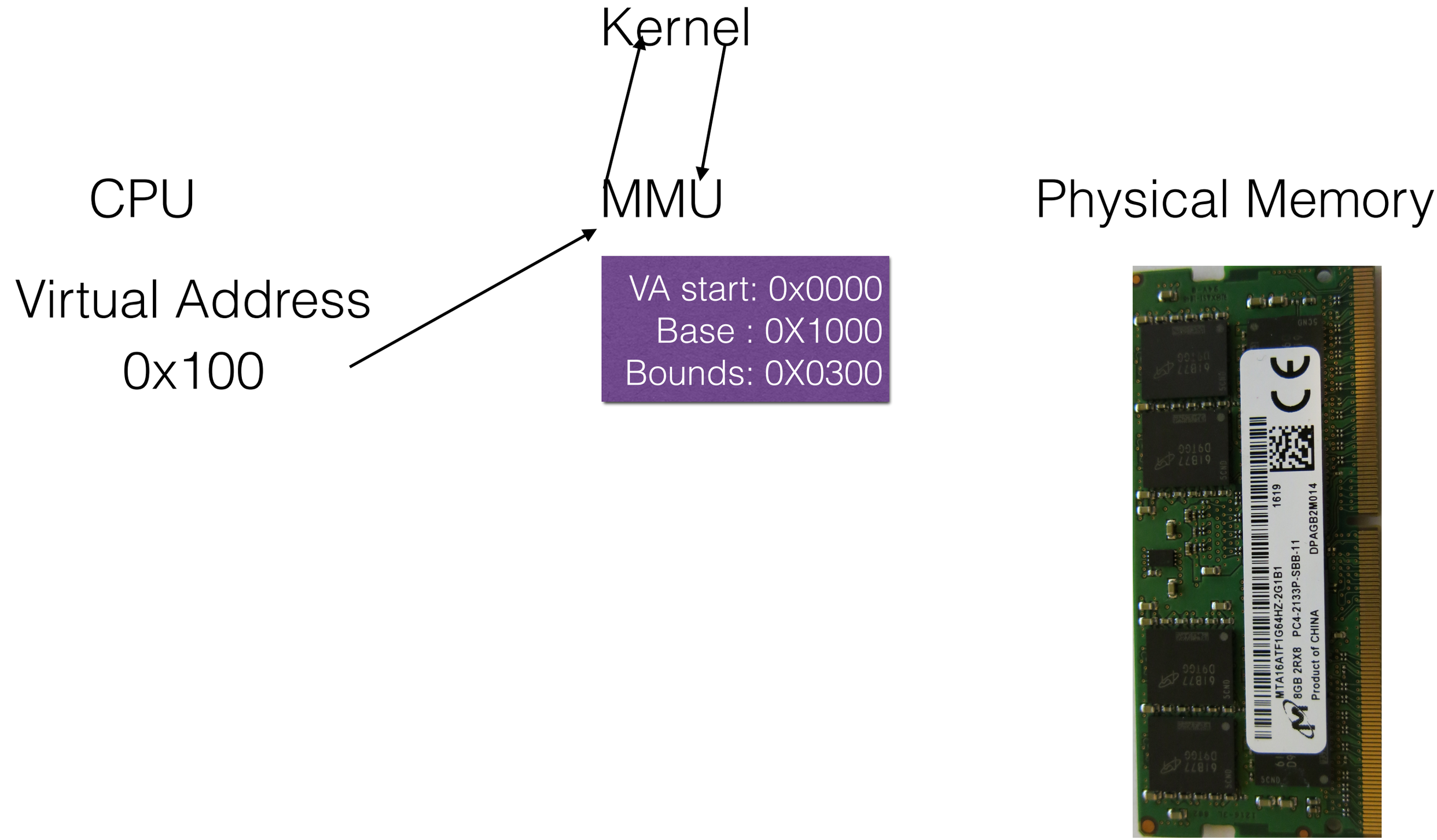
Segmentation Example



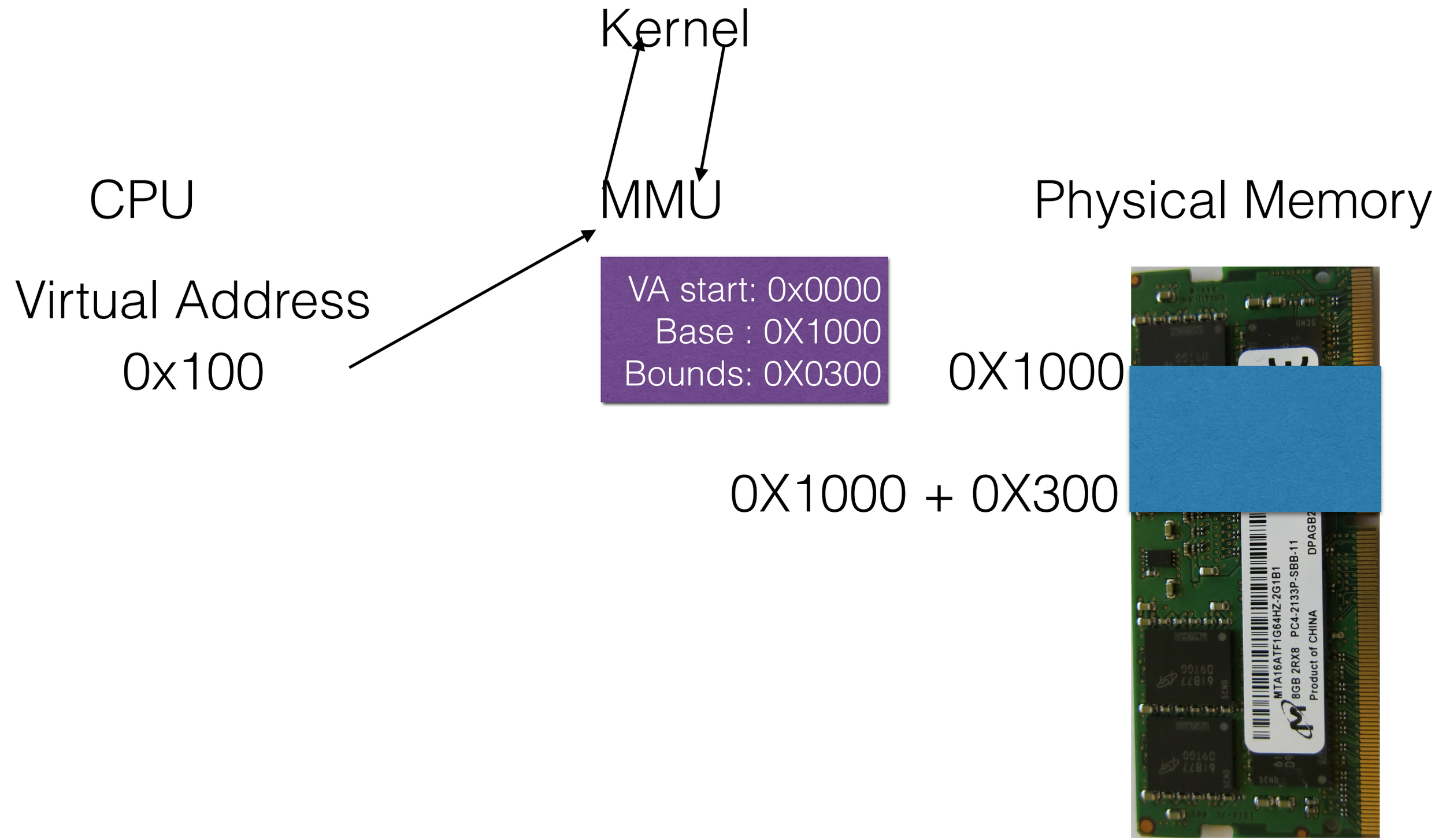
Segmentation Example



Segmentation Example



Segmentation Example



Segmentation Example

Kernel

CPU

MMU

Physical Memory

Virtual Address

0x100

VA start: 0x0000
Base : 0X1000
Bounds: 0X0300

0X1000

0X1000 + 0X300

Check: Segment Start < V.A. < Segment Start + Segment Bound.

Segmentation Example

Kernel

CPU

MMU

Physical Memory

Virtual Address

0x100

VA start: 0x0000
Base : 0X1000
Bounds: 0X0300

0X1000

0X1000 + 0X300

Check: Segment Start < V.A. < Segment Start + Segment Bound.

Check:0X000 <0X0100 <0X300

Segmentation Example

Kernel

CPU

MMU

Physical Memory

Virtual Address

0x100

VA start: 0x0000
Base : 0X1000
Bounds: 0X0300

0X1000

0X1000 + 0X300

Check: Segment Start < V.A. < Segment Start + Segment Bound.

Check:0X000 <0X0100 <0X300

Segmentation Example

Kernel

CPU

MMU

Physical Memory

Virtual Address

0x100

VA start: 0x0000
Base : 0X1000
Bounds: 0X0300

0X1000

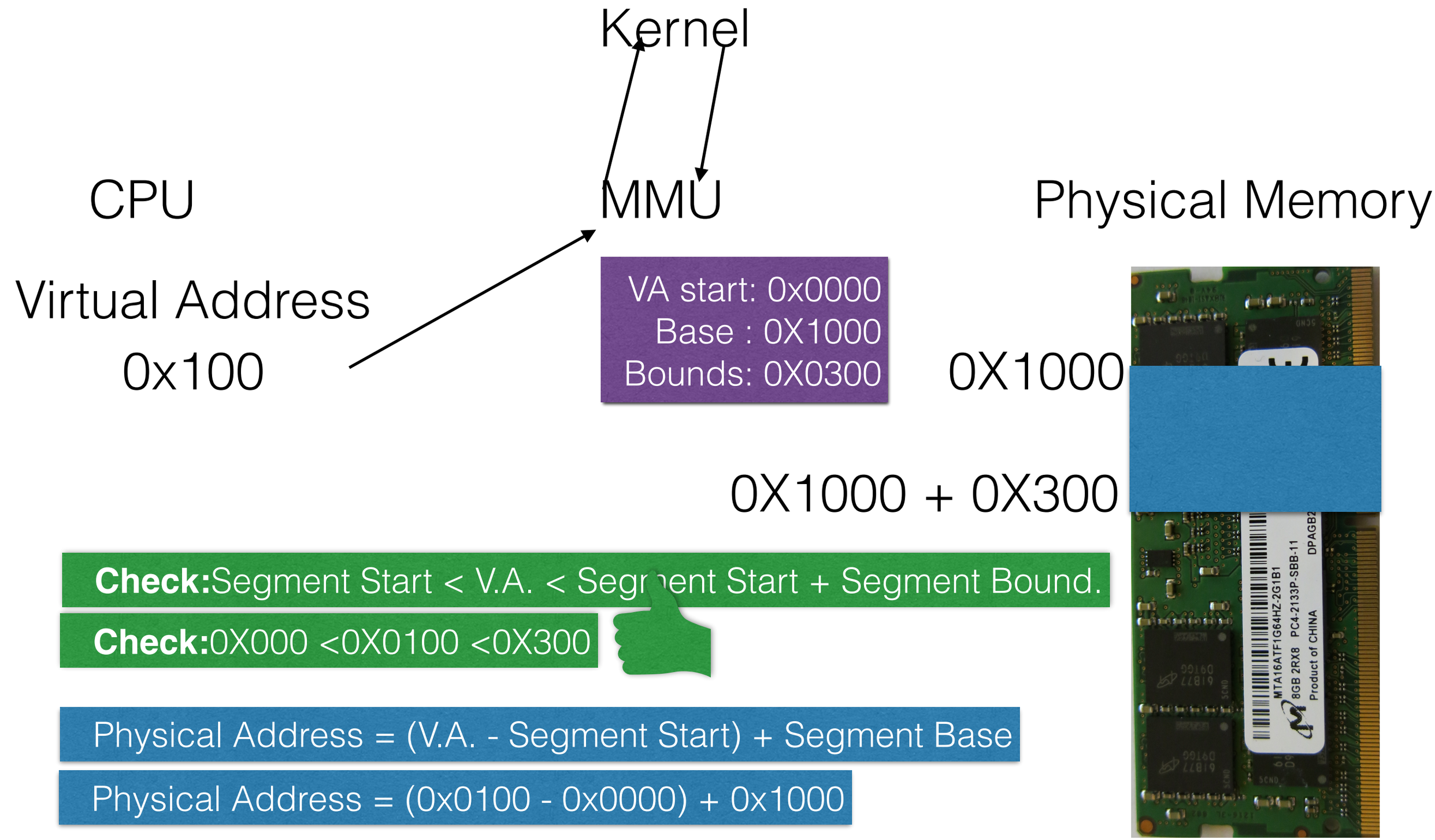
0X1000 + 0X300

Check: Segment Start < V.A. < Segment Start + Segment Bound.

Check:0X000 <0X0100 <0X300

Physical Address = (V.A. - Segment Start) + Segment Base

Segmentation Example



Segmentation Example

Kernel

CPU

MMU

Physical Memory

Virtual Address

0x100

VA start: 0x0000
Base : 0X1000
Bounds: 0X0300

0X1000

0X1000 + 0X300

Check: Segment Start < V.A. < Segment Start + Segment Bound.

Check:0X000 <0X0100 <0X300

Physical Address = (V.A. - Segment Start) + Segment Base

Physical Address = (0x0100 - 0x0000) + 0x1000

Segmentation Example

Kernel

CPU

MMU

Physical Memory

Virtual Address

0x2400

Segmentation Example

Kernel

CPU

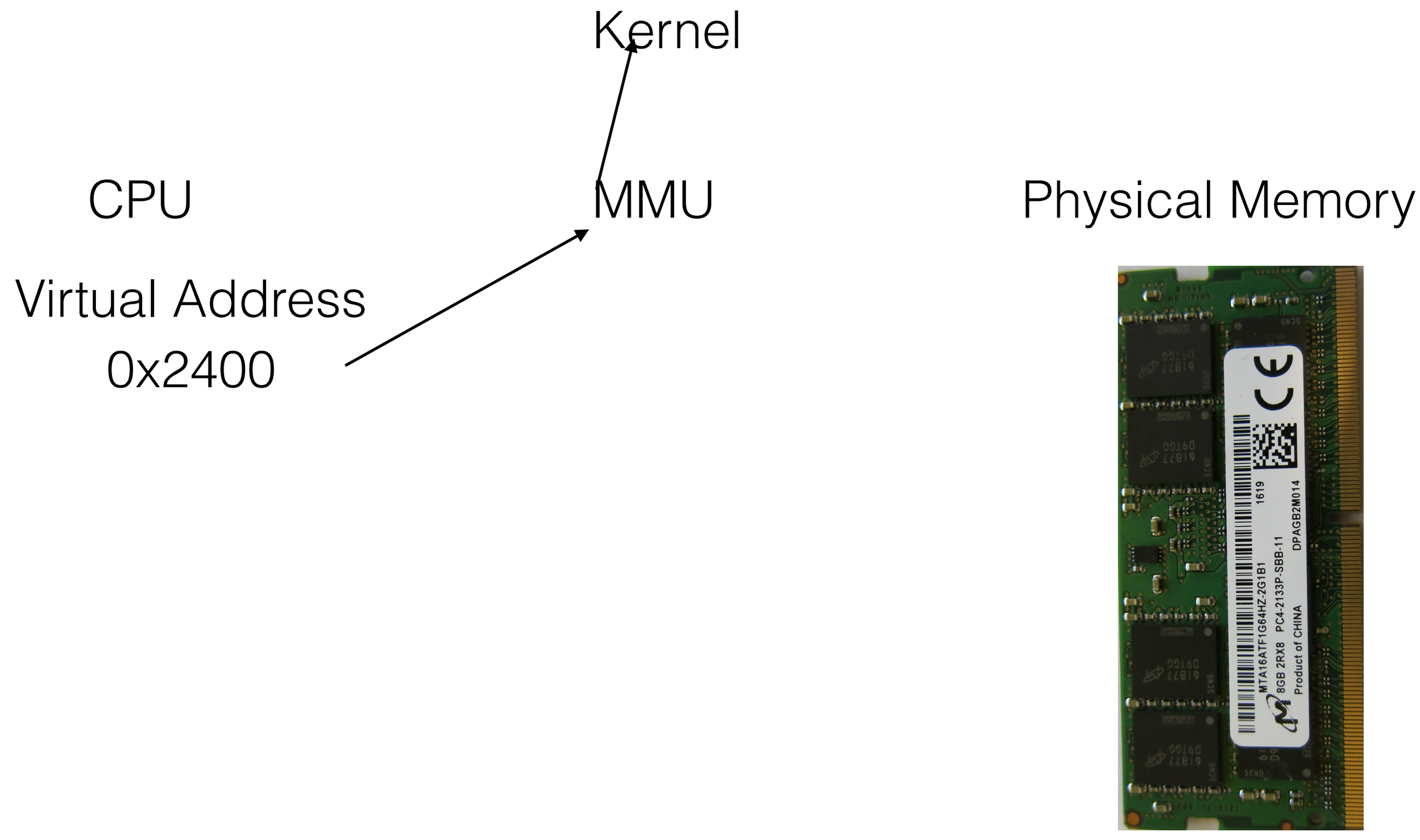
MMU

Physical Memory

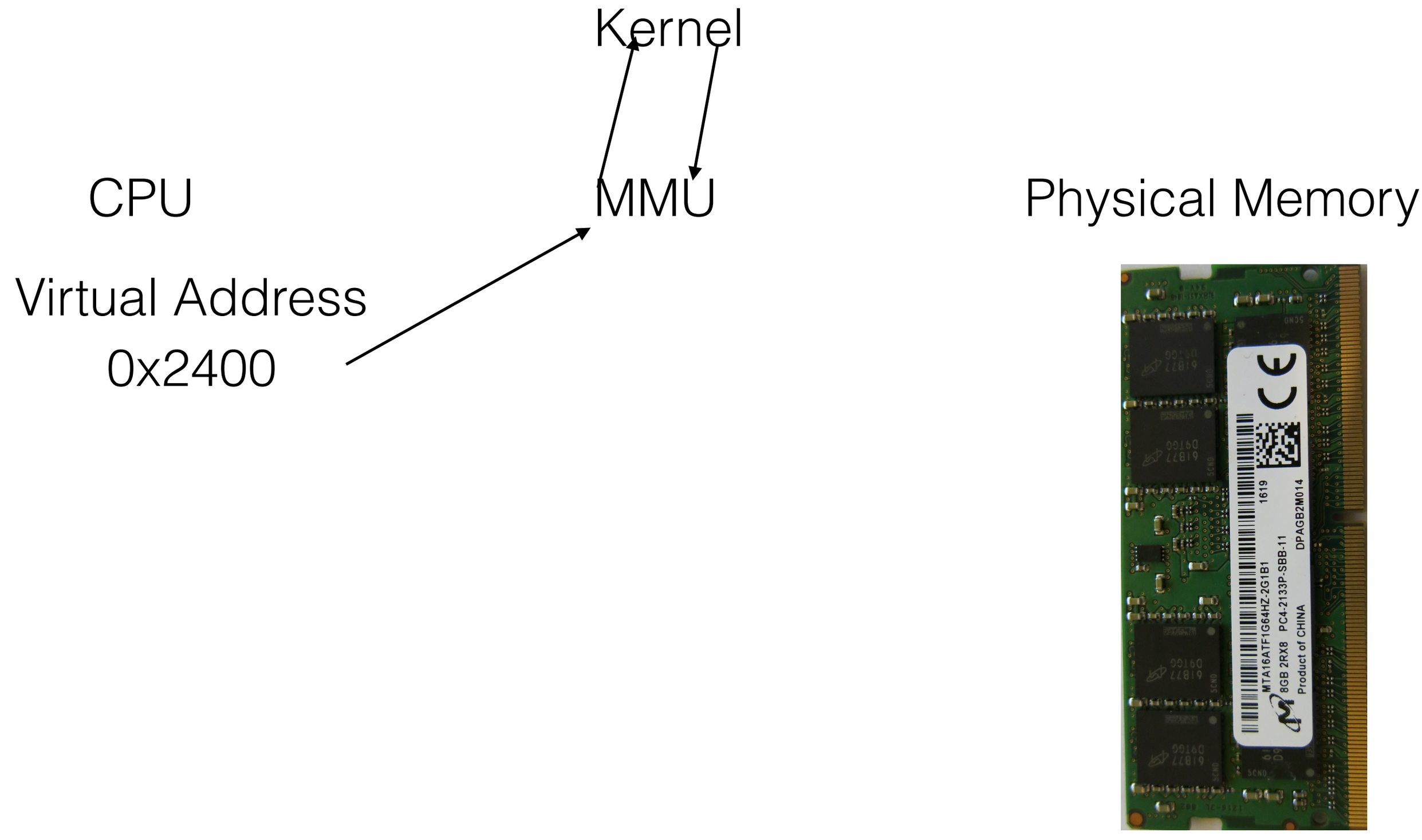
Virtual Address

0x2400

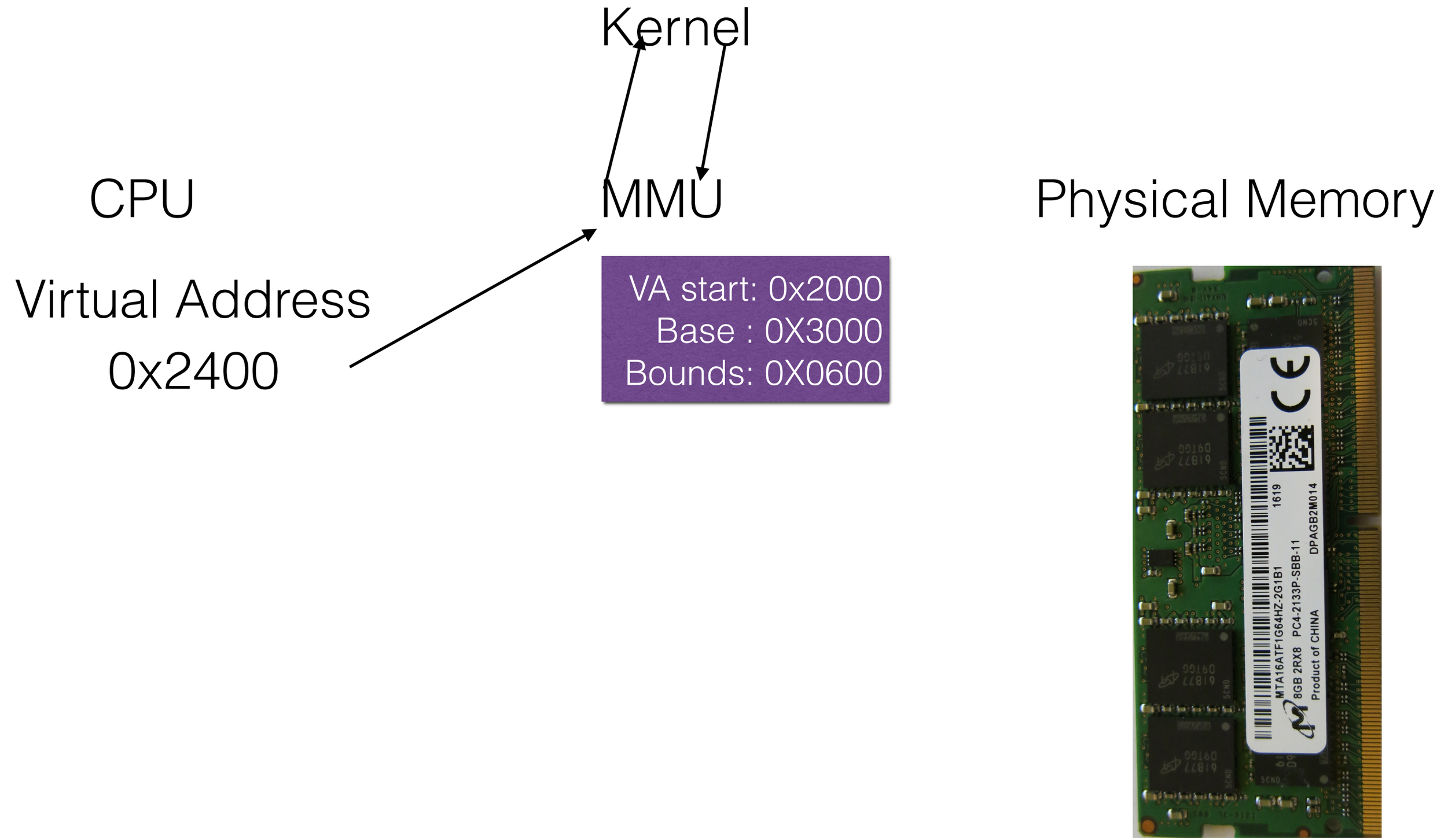
Segmentation Example



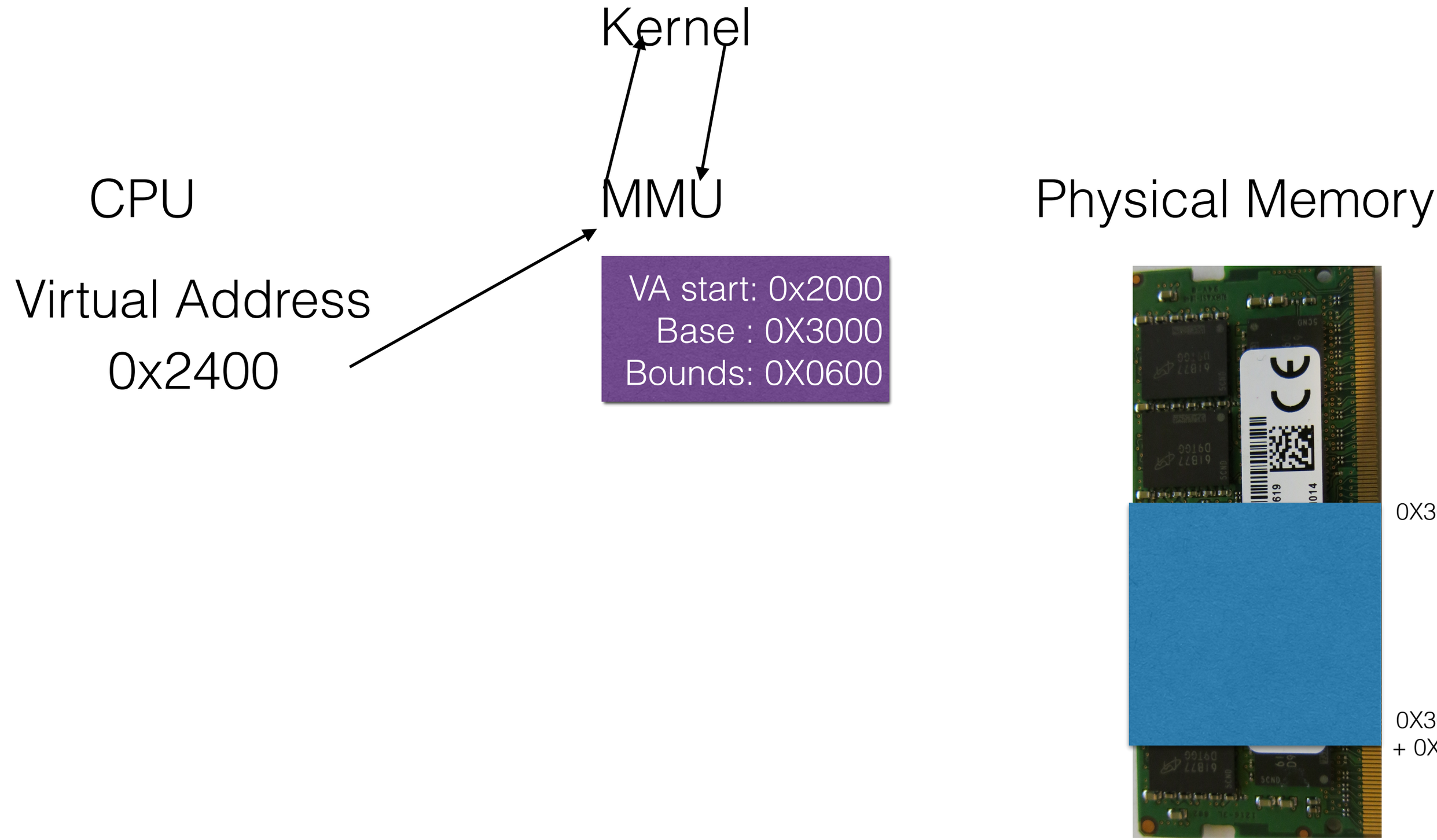
Segmentation Example



Segmentation Example



Segmentation Example



Segmentation Example

Kernel

CPU

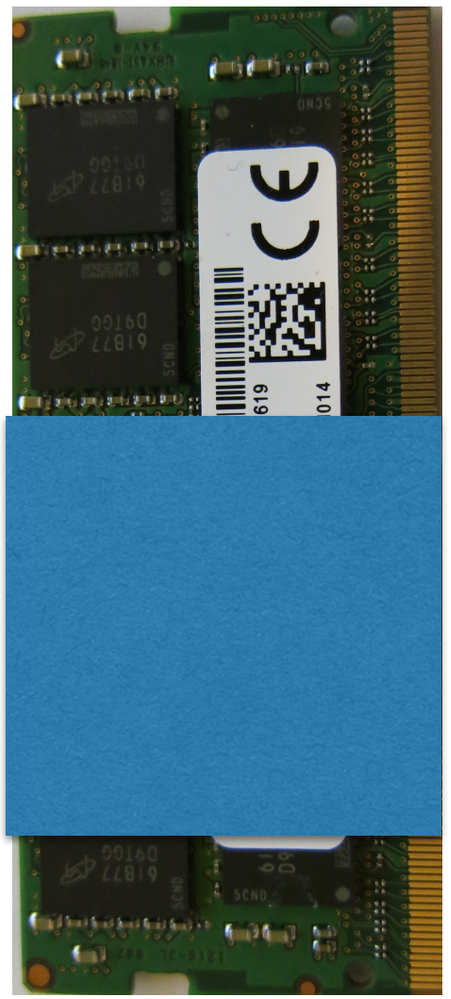
MMU

Physical Memory

Virtual Address

0x2400

VA start: 0x2000
Base : 0X3000
Bounds: 0X0600

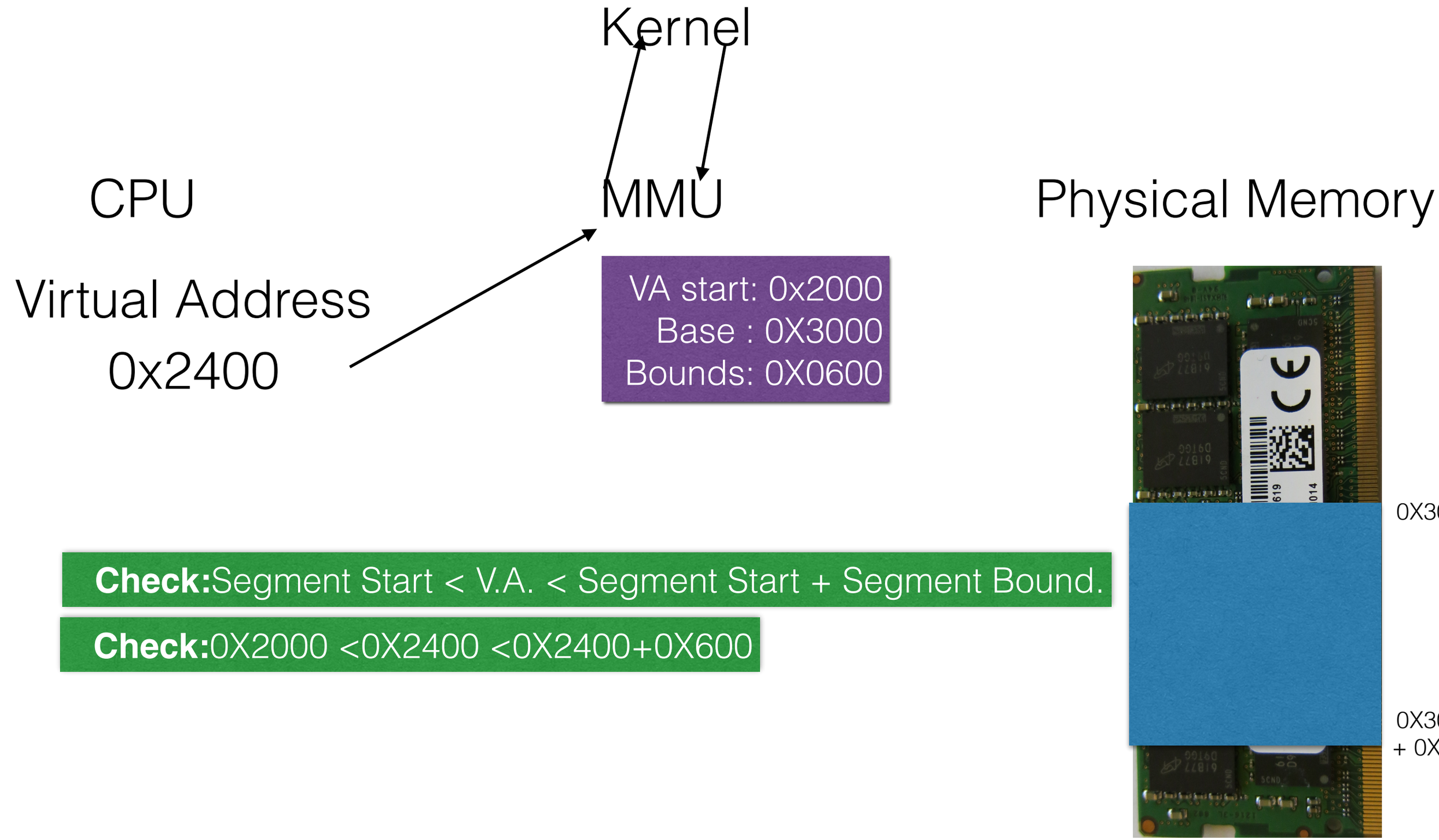


0X3000

0X3000
+ 0X600

Check: Segment Start < V.A. < Segment Start + Segment Bound.

Segmentation Example



Check: Segment Start < V.A. < Segment Start + Segment Bound.

Check:0X2000 <0X2400 <0X2400+0X600

Segmentation Example

Kernel

CPU

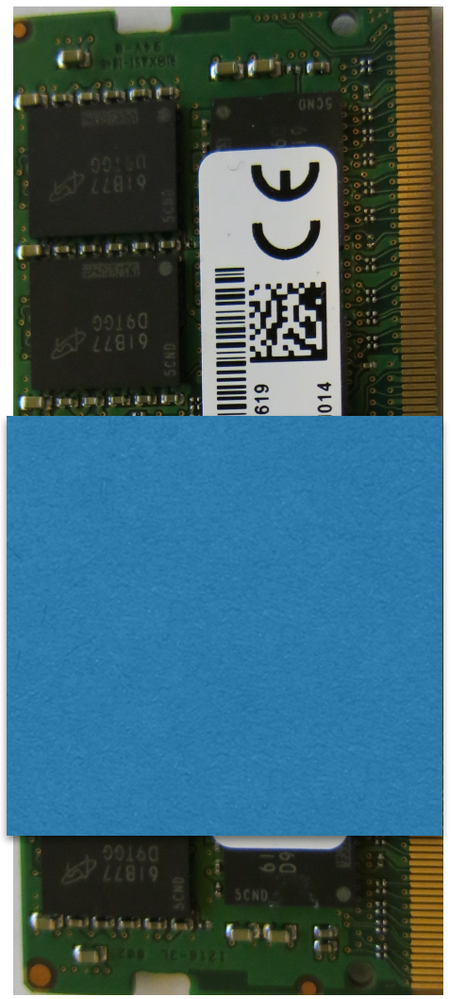
MMU

Physical Memory

Virtual Address

0x2400

VA start: 0x2000
Base : 0X3000
Bounds: 0X0600



0X3000

0X3000
+ 0X600

Check: Segment Start < V.A. < Segment Start + Segment Bound.

Check:0X2000 <0X2400 <0X2400+0X600

Segmentation Example

Kernel

CPU

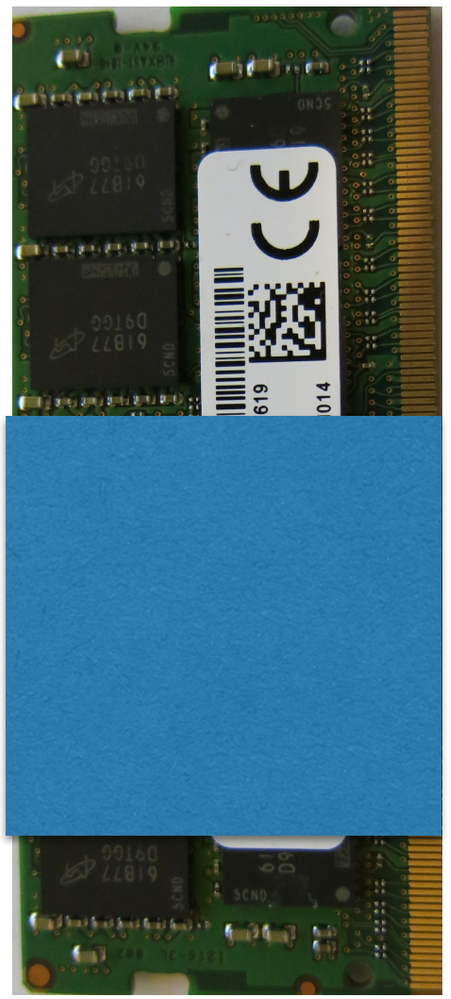
MMU

Physical Memory

Virtual Address

0x2400

VA start: 0x2000
Base : 0X3000
Bounds: 0X0600



0X3000

0X3000
+ 0X600

Check: Segment Start < V.A. < Segment Start + Segment Bound.

Check:0X2000 <0X2400 <0X2400+0X600

Physical Address = (V.A. - Segment Start) + Segment Base

Segmentation Example

Kernel

CPU

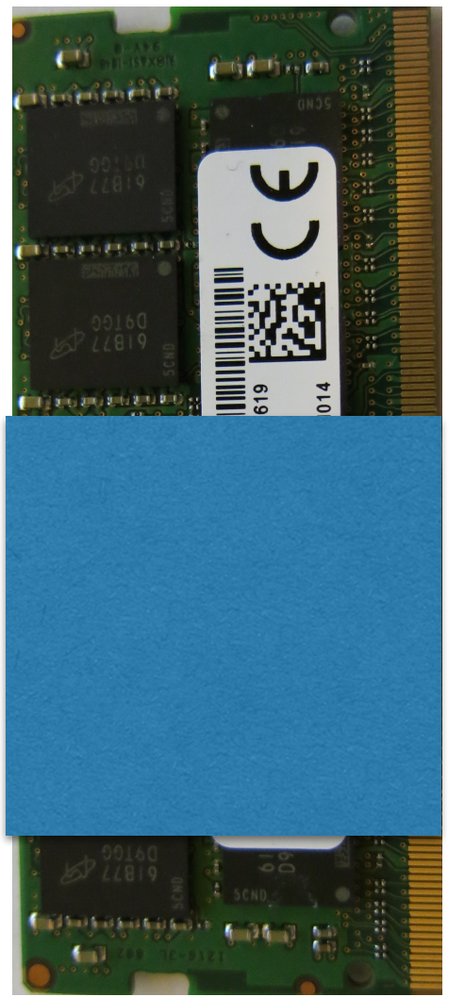
MMU

Physical Memory

Virtual Address

0x2400

VA start: 0x2000
Base : 0X3000
Bounds: 0X0600



0X3000

0X3000
+ 0X600

Check: Segment Start < V.A. < Segment Start + Segment Bound.

Check:0X2000 <0X2400 <0X2400+0X600

Physical Address = (V.A. - Segment Start) + Segment Base

Physical Address = (0x2400 - 0x2000) + 0x3000

Segmentation Example

Kernel

CPU

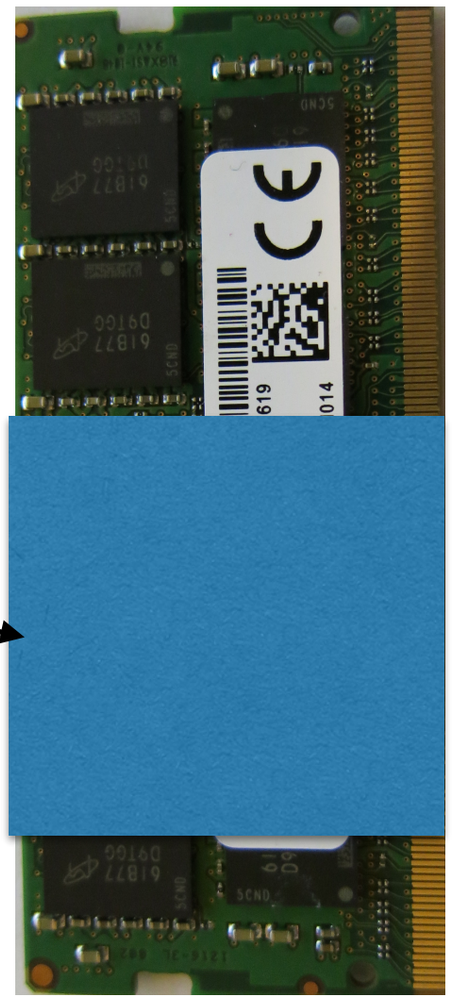
MMU

Physical Memory

Virtual Address

0x2400

VA start: 0x2000
Base : 0X3000
Bounds: 0X0600



0X3000

0X3000
+ 0X600

Check: Segment Start < V.A. < Segment Start + Segment Bound.

Check:0X2000 <0X2400 <0X2400+0X600

Physical Address = (V.A. - Segment Start) + Segment Base

Physical Address = (0x2400 - 0x2000) + 0x3000

Segmentation Example

Kernel

CPU

MMU

Physical Memory

Virtual Address

0x2700

Segmentation Example

Kernel

CPU

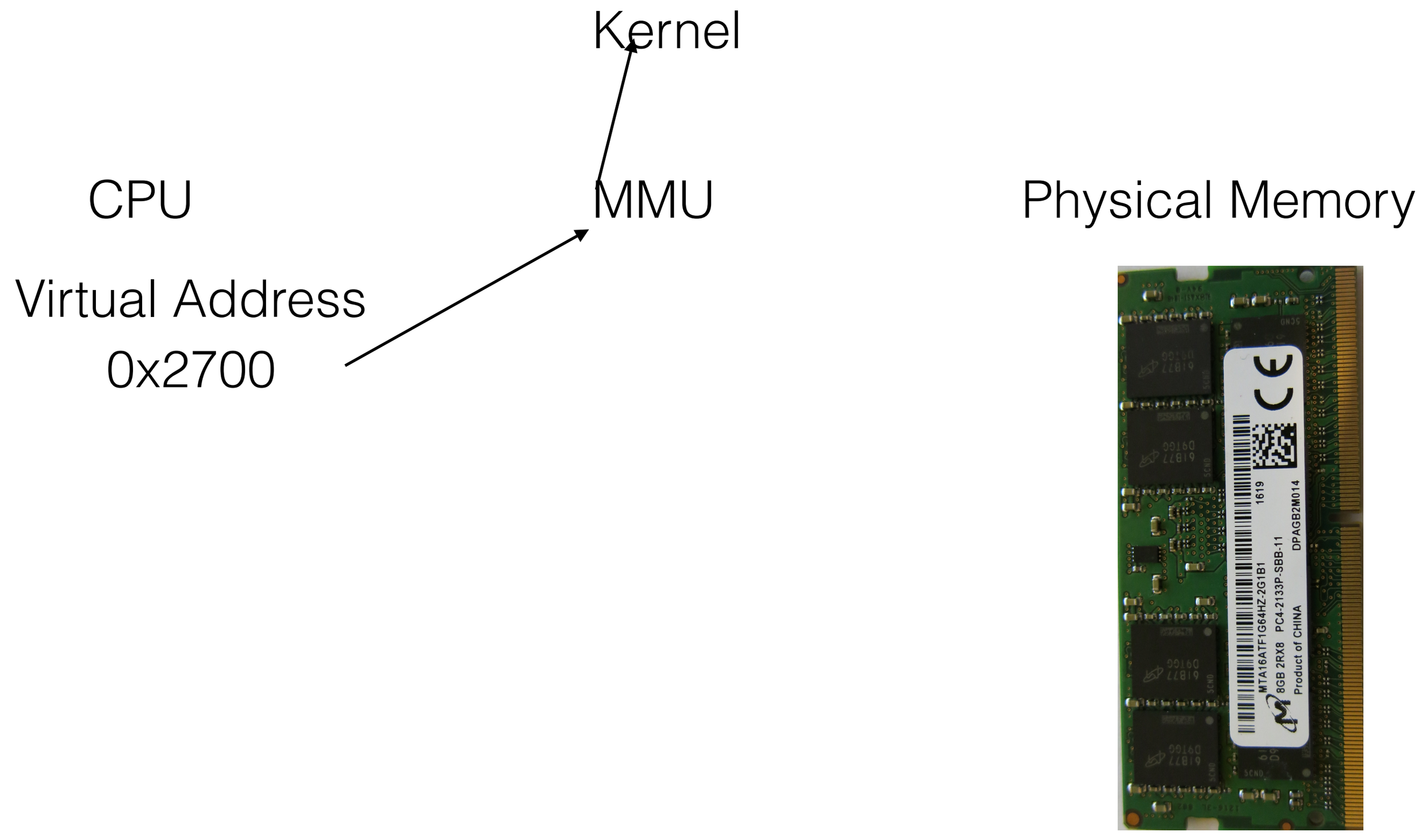
MMU

Physical Memory

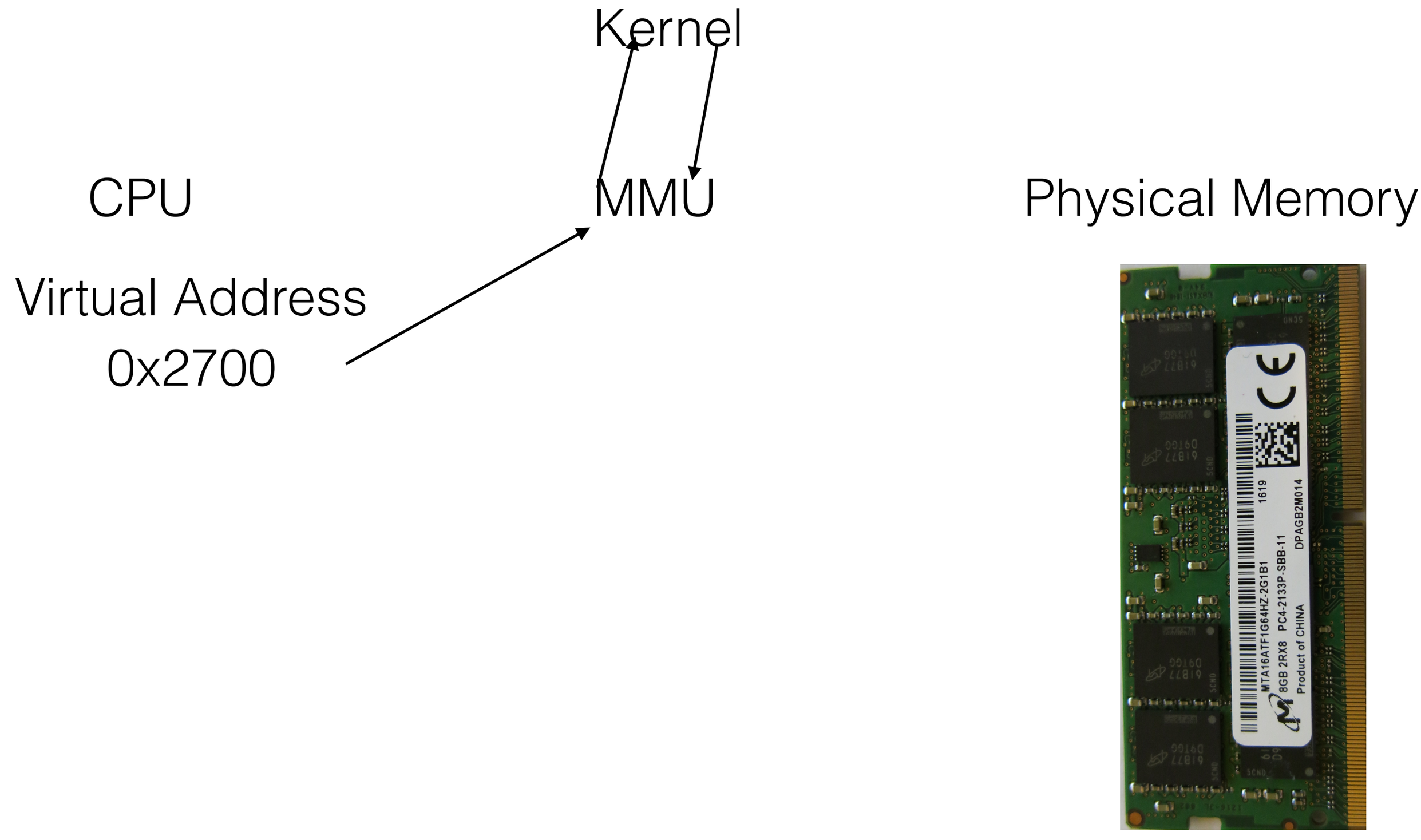
Virtual Address

0x2700

Segmentation Example



Segmentation Example



Segmentation Example

Kernel

VA start: 0x0000
Base : 0X1000
Bounds: 0X0300

CPU

MMU

Physical Memory

Virtual Address
0x2700

Segmentation Example

Kernel

VA start: 0x0000
Base : 0X1000
Bounds: 0X0300

VA start: 0x2000
Base : 0X3000
Bounds: 0X0600

CPU

MMU

Physical Memory

Virtual Address
0x2700

Segmentation Example

Kernel

VA start: 0x0000
Base : 0X1000
Bounds: 0X0300

VA start: 0x2000
Base : 0X3000
Bounds: 0X0600

CPU

MMU

Physical Memory

Virtual Address
0x2700

Check: Segment Start < V.A. < Segment Start + Segment Bound.

Segmentation Example

Kernel

VA start: 0x0000
Base : 0X1000
Bounds: 0X0300

VA start: 0x2000
Base : 0X3000
Bounds: 0X0600

CPU

MMU

Physical Memory

Virtual Address
0x2700

Check: Segment Start < V.A. < Segment Start + Segment Bound.

Segmentation Example

Kernel

VA start: 0x0000
Base : 0X1000
Bounds: 0X0300

VA start: 0x2000
Base : 0X3000
Bounds: 0X0600

CPU

MMU

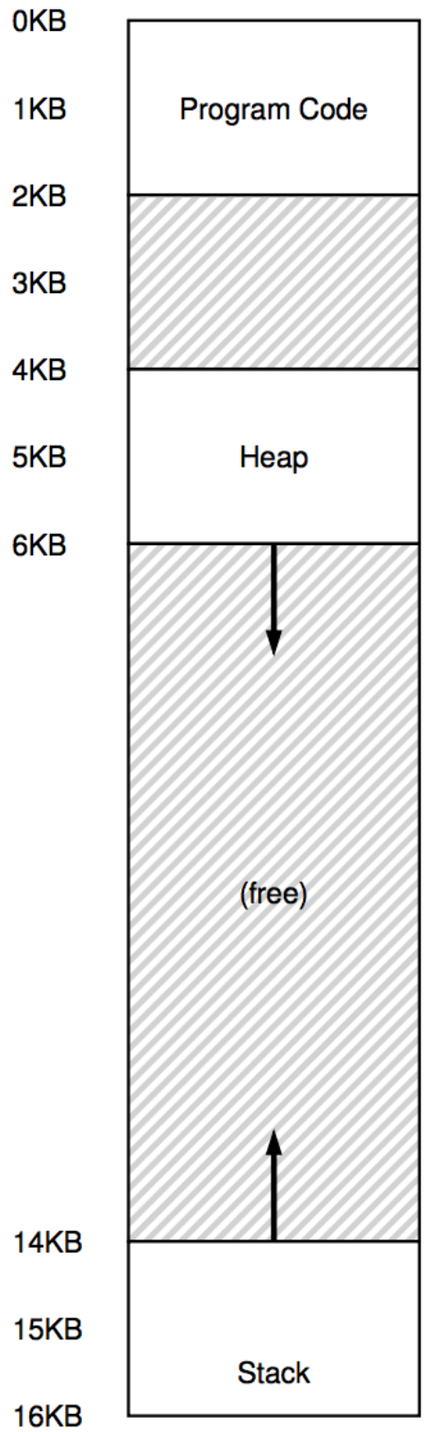
Physical Memory

Virtual Address
0x2700

Check: Segment Start < V.A. < Segment Start + Segment Bound.

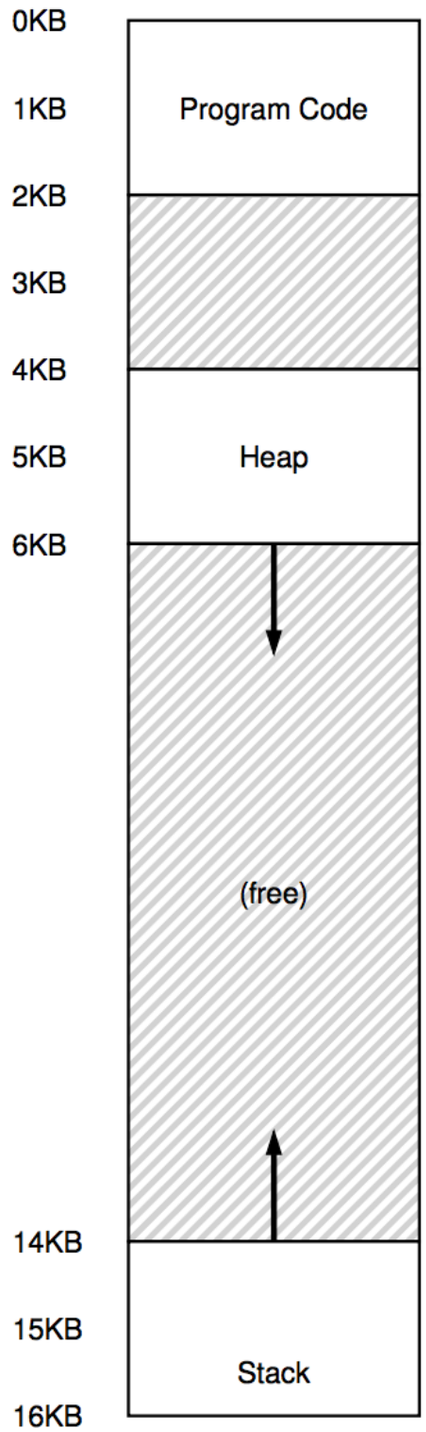
Segment Reference

Virtual Address Space



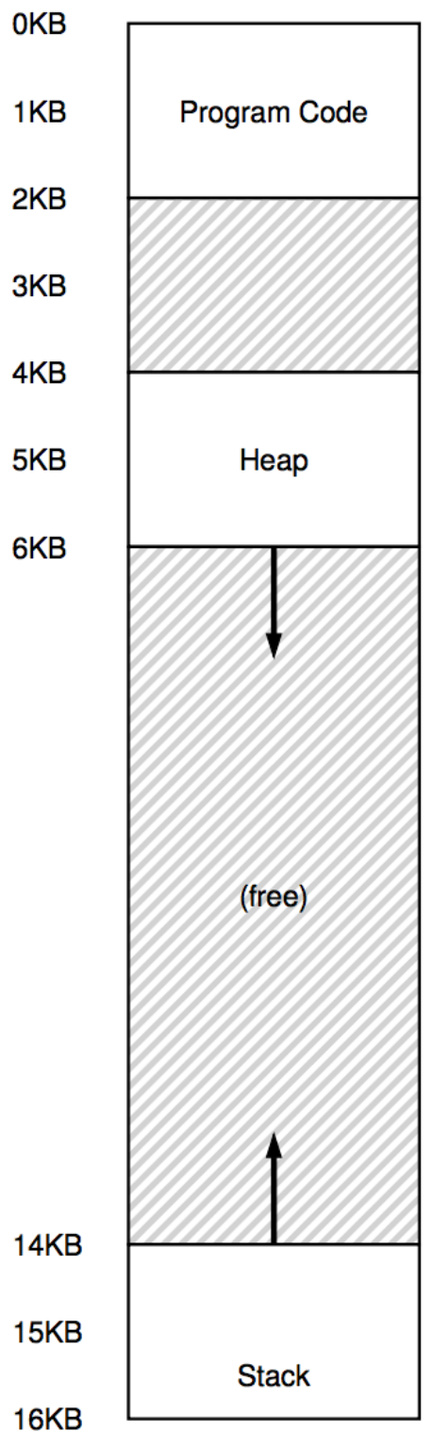
Segment Reference

Virtual Address Space



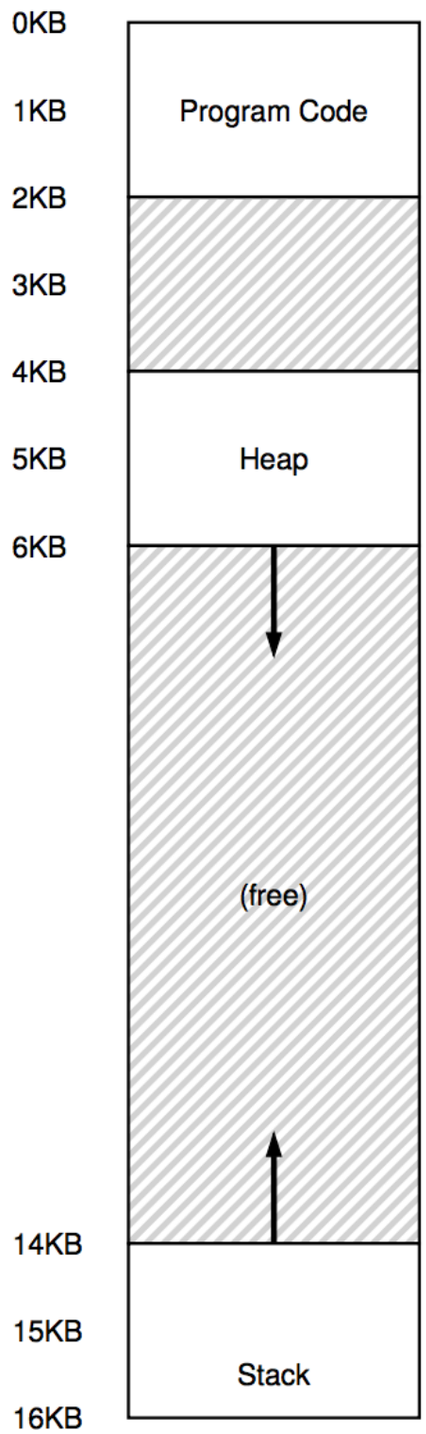
Segment Reference

Virtual Address Space



Segment Reference

Virtual Address Space

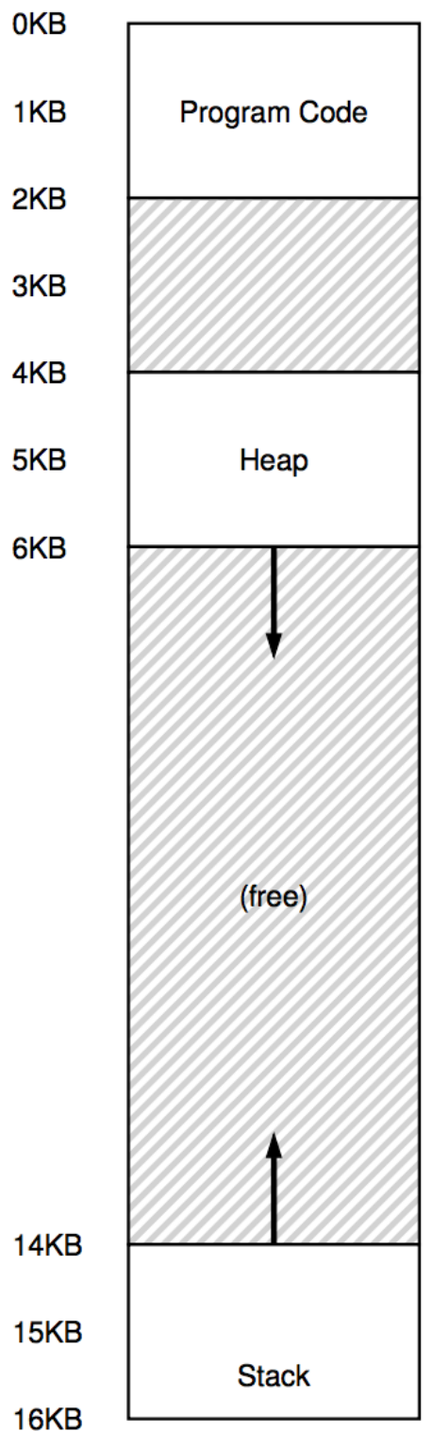


00	0000	0000	0000
00	0111	1111	1111

11	1111	1111	1111

Segment Reference

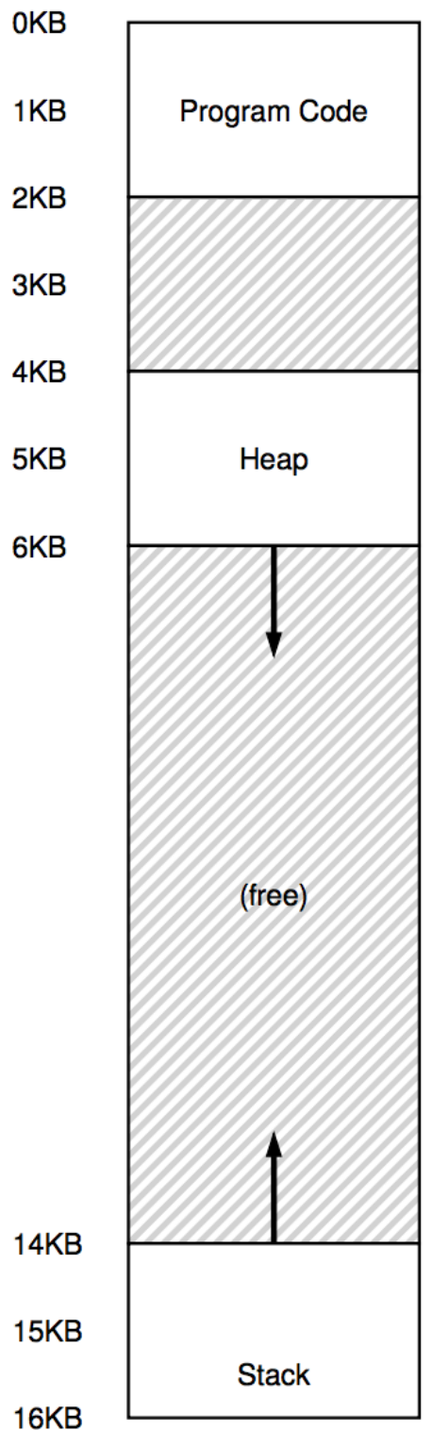
Virtual Address Space



00	0000	0000	0000
00	0111	1111	1111
01	0000	0000	0000
11	1111	1111	1111

Segment Reference

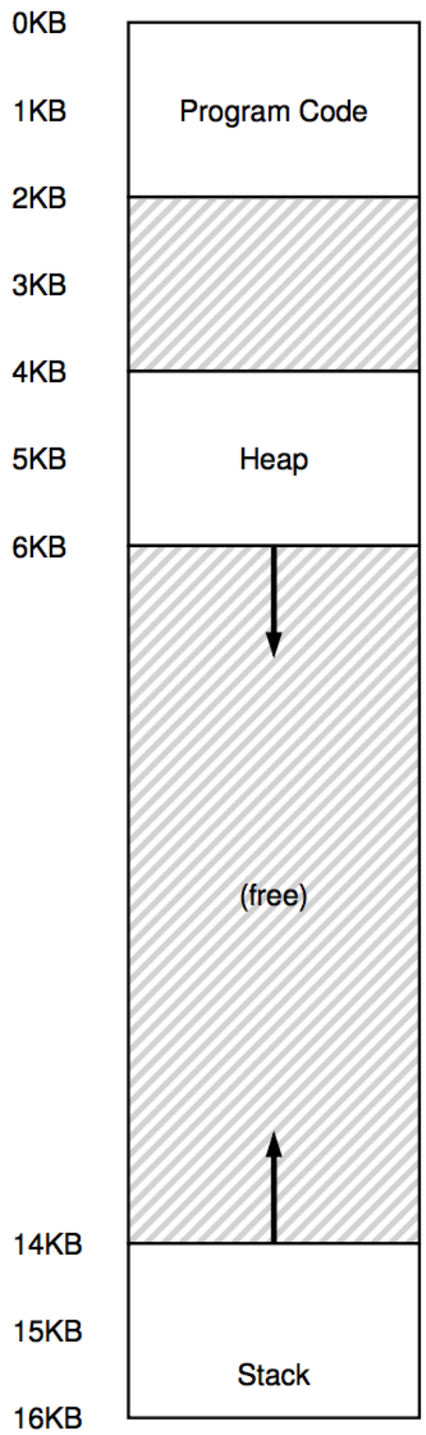
Virtual Address Space



00	0000	0000	0000
00	0111	1111	1111
01	0000	0000	0000
01	0111	1111	1111
11	1111	1111	1111

Segment Reference

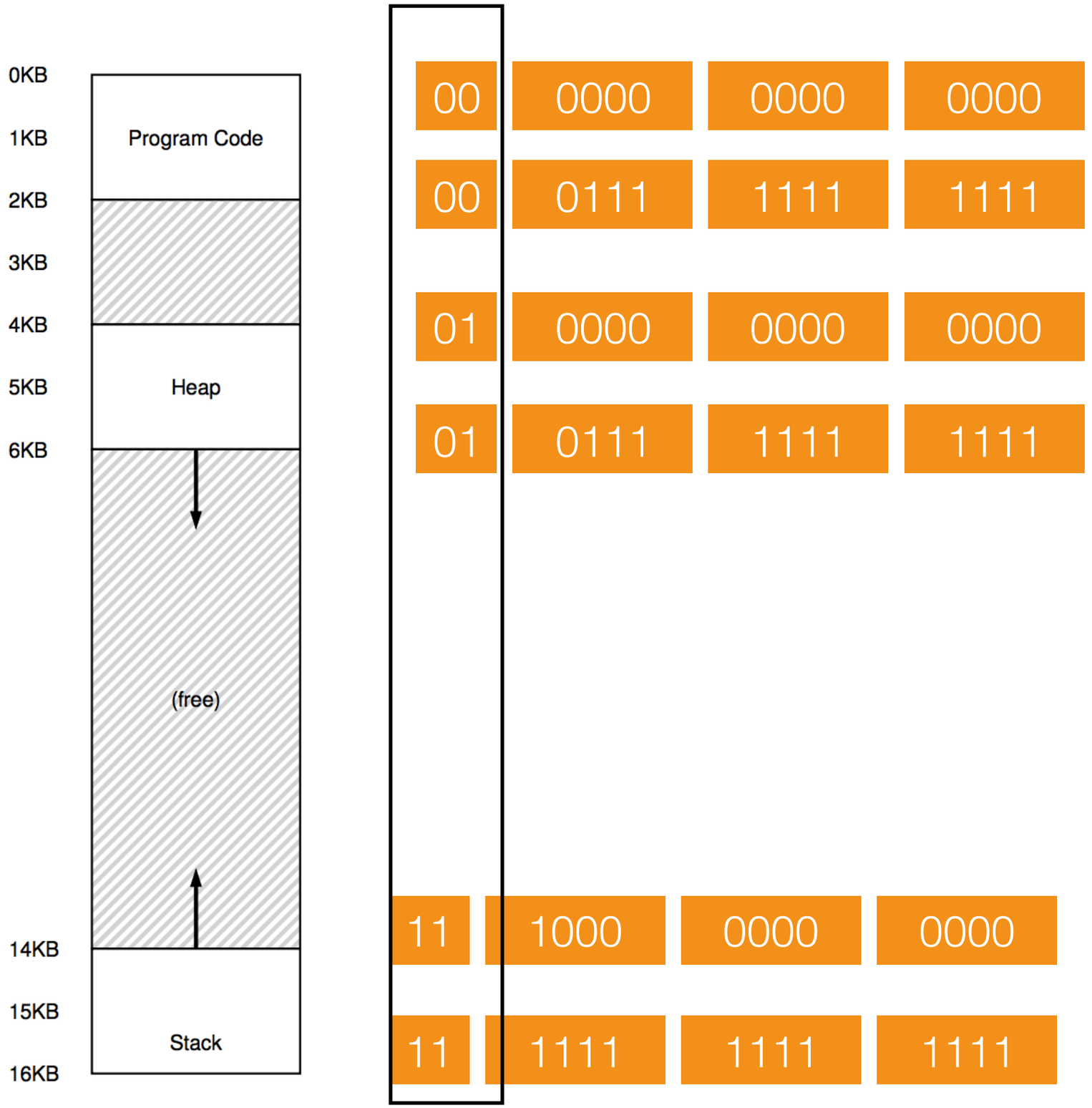
Virtual Address Space



00	0000	0000	0000
00	0111	1111	1111
01	0000	0000	0000
01	0111	1111	1111
11	1000	0000	0000
11	1111	1111	1111

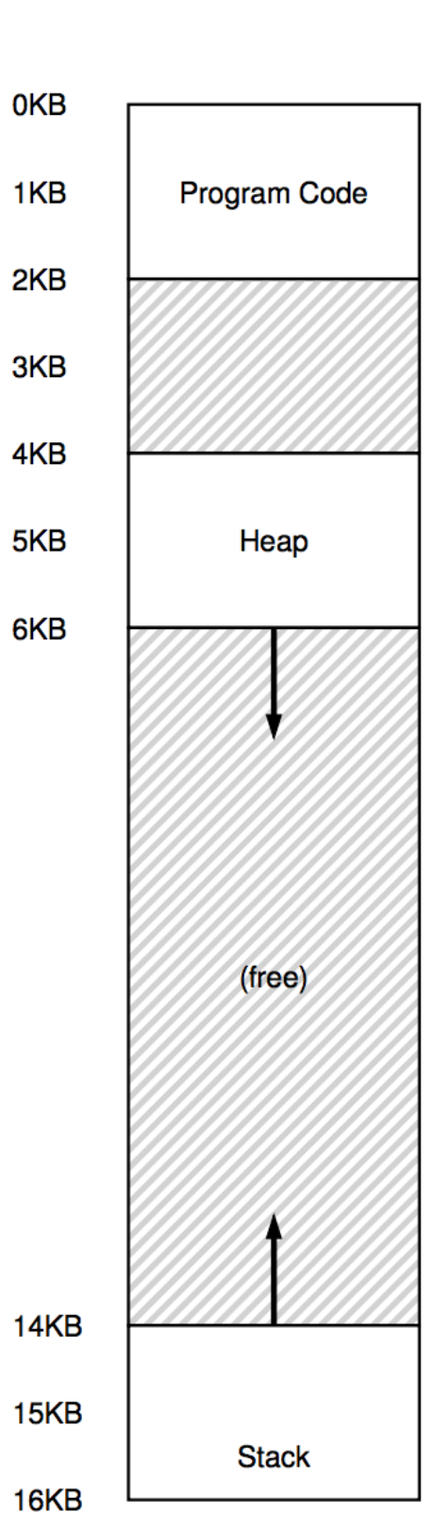
Segment Reference

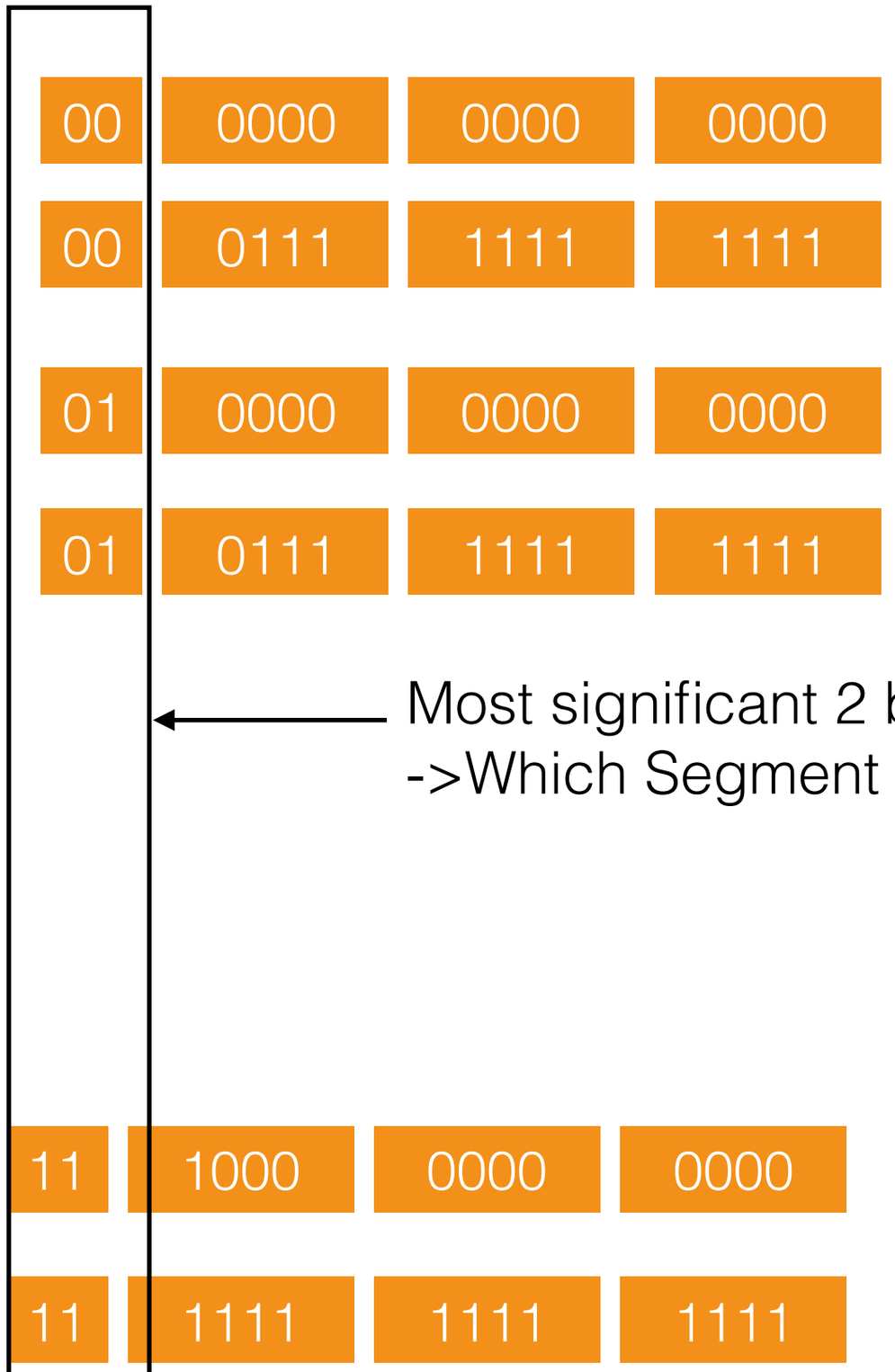
Virtual Address Space



Segment Reference

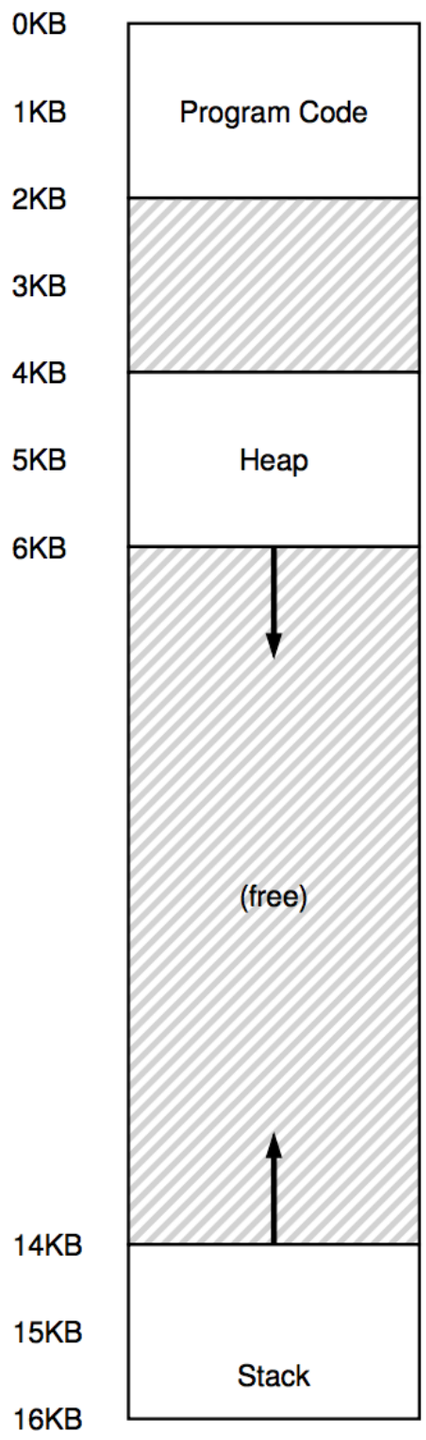
Virtual Address Space

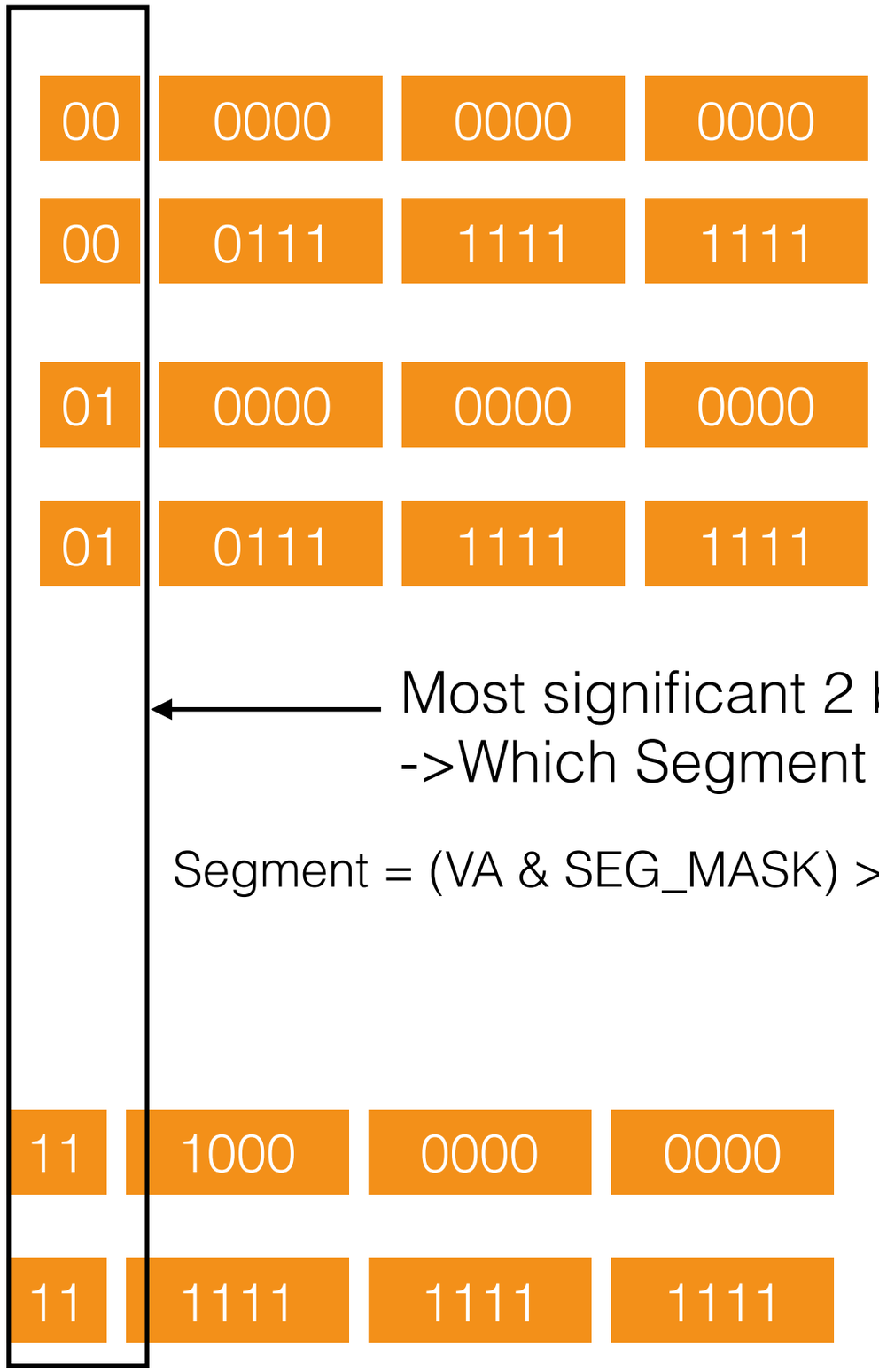




Segment Reference

Virtual Address Space



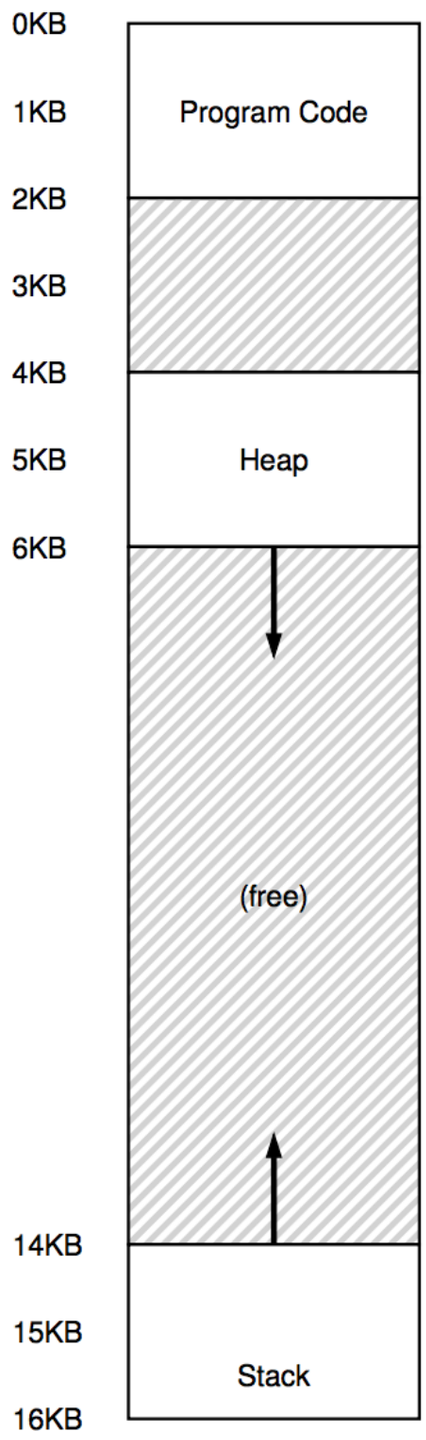


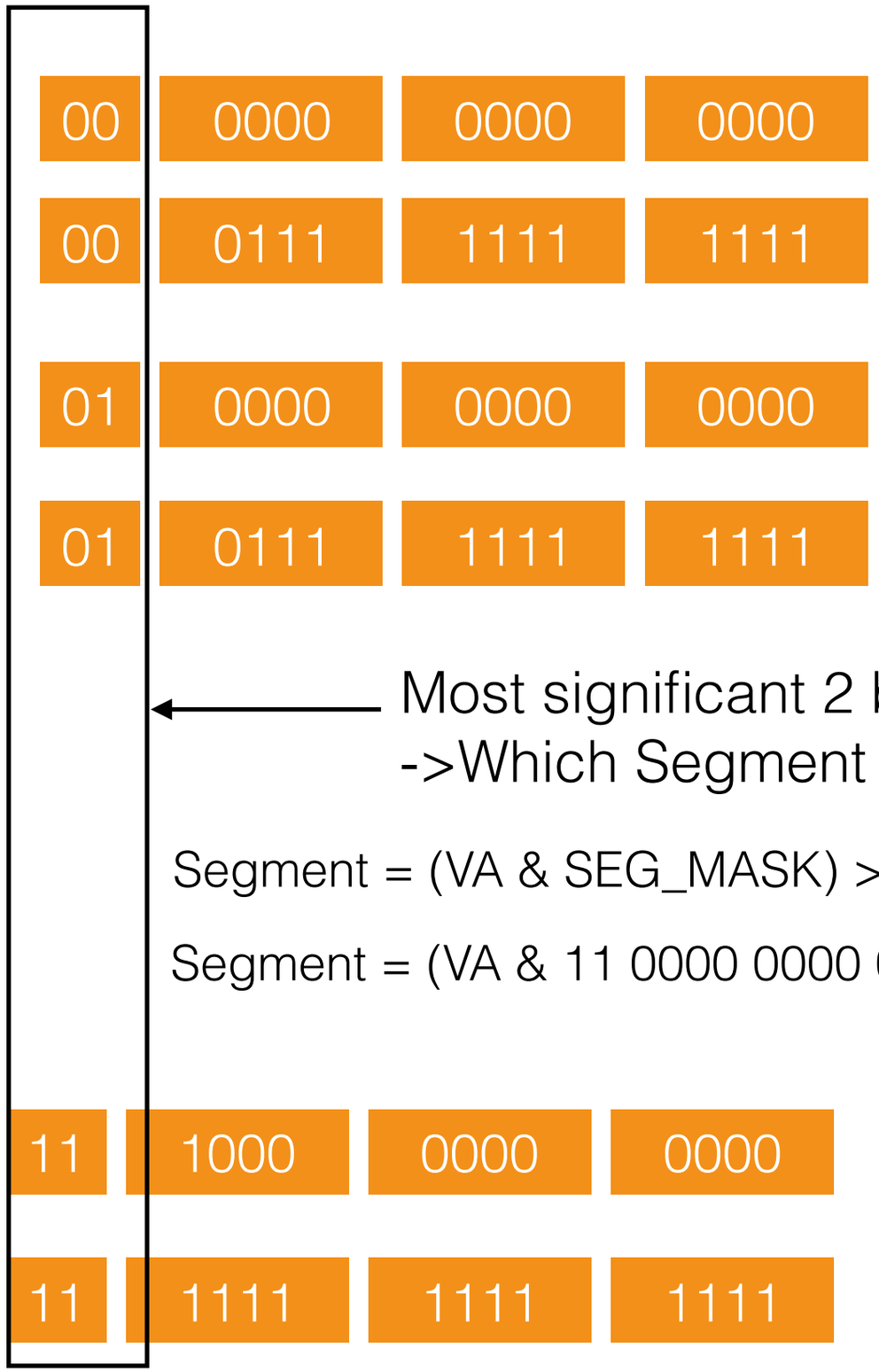
Most significant 2 bits
->Which Segment

Segment = (VA & SEG_MASK) >>SEG_SHIFT

Segment Reference

Virtual Address Space



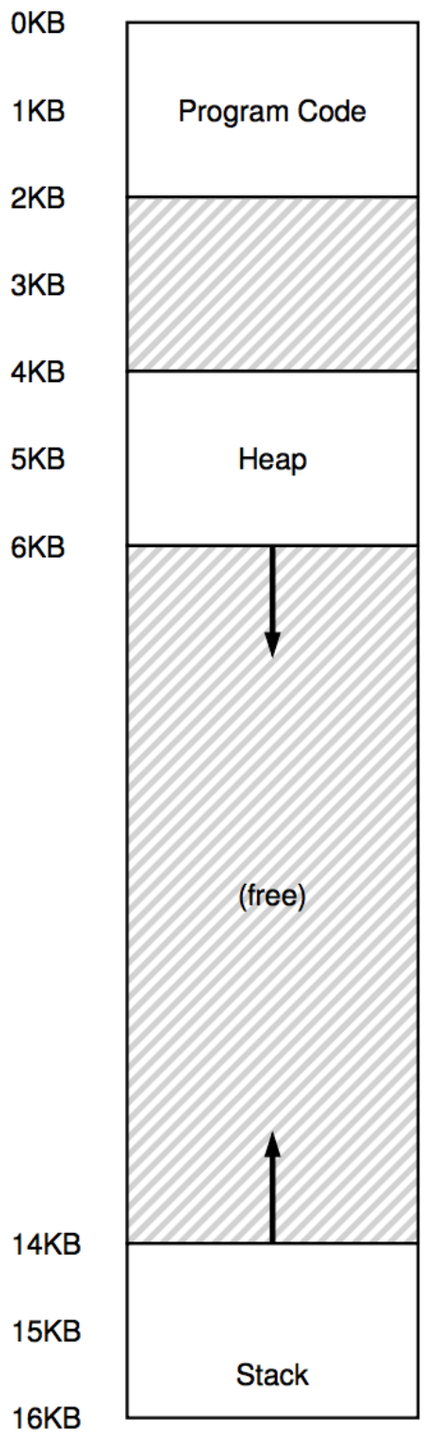


Most significant 2 bits
->Which Segment

Segment = (VA & SEG_MASK) >>SEG_SHIFT

Segment = (VA & 11 0000 0000 0000) >>12

Segment Reference



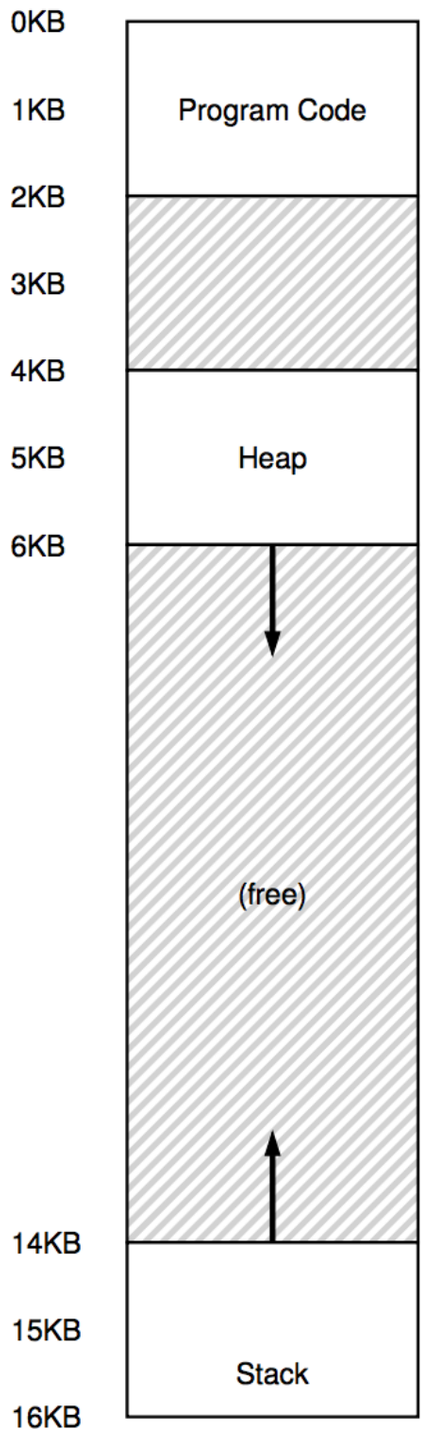
00	0000	0000	0000
00	0111	1111	1111
01	0000	0000	0000
01	0111	1111	1111

Segment = (VA & 11 0000 0000 0000) >>12

11	1000	0000	0000
11	1111	1111	1111

Segment Reference

00	0000	0000	1010



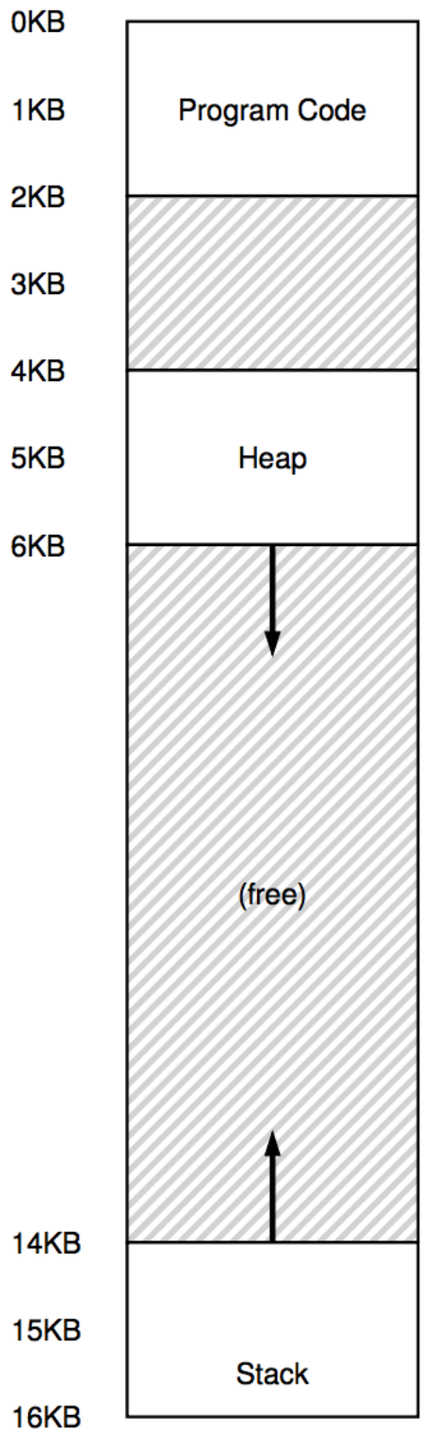
00	0000	0000	0000
00	0111	1111	1111
01	0000	0000	0000
01	0111	1111	1111

Segment = (VA & 11 0000 0000 0000) >>12

11	1000	0000	0000
11	1111	1111	1111

Segment Reference

00	0000	0000	1010
11	0000	0000	0000



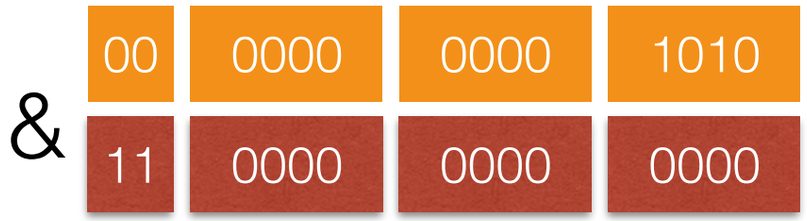
00	0000	0000	0000
00	0111	1111	1111

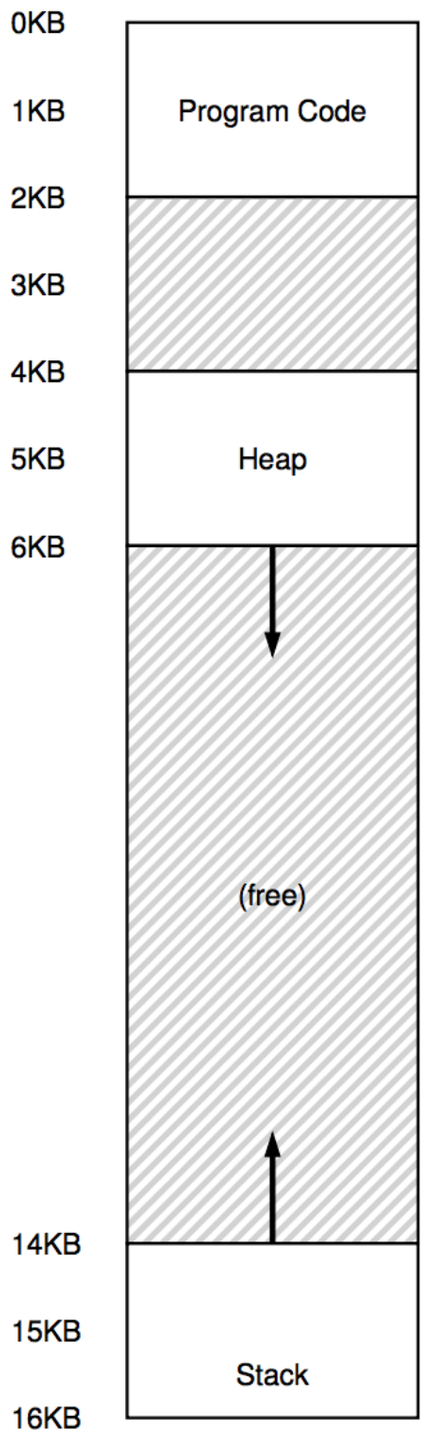
01	0000	0000	0000
01	0111	1111	1111

Segment = (VA & 11 0000 0000 0000) >>12

11	1000	0000	0000
11	1111	1111	1111

Segment Reference





00 0000 0000 0000

00 0111 1111 1111

01 0000 0000 0000

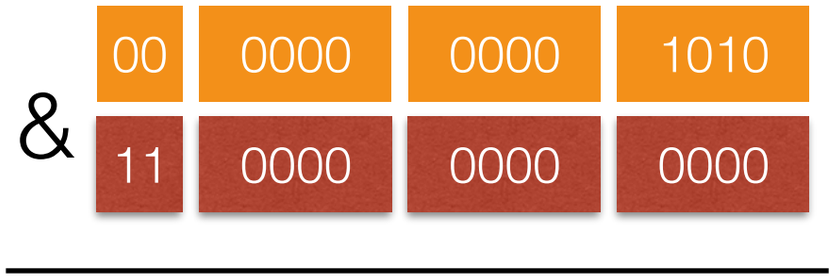
01 0111 1111 1111

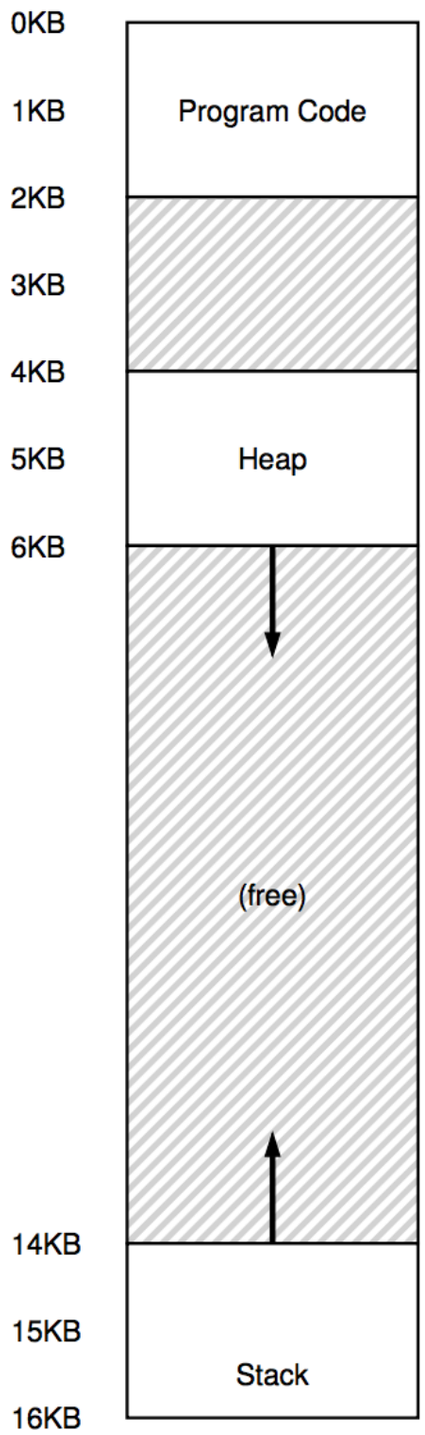
Segment = (VA & 11 0000 0000 0000) >>12

11 1000 0000 0000

11 1111 1111 1111

Segment Reference





00 0000 0000 0000

00 0111 1111 1111

01 0000 0000 0000

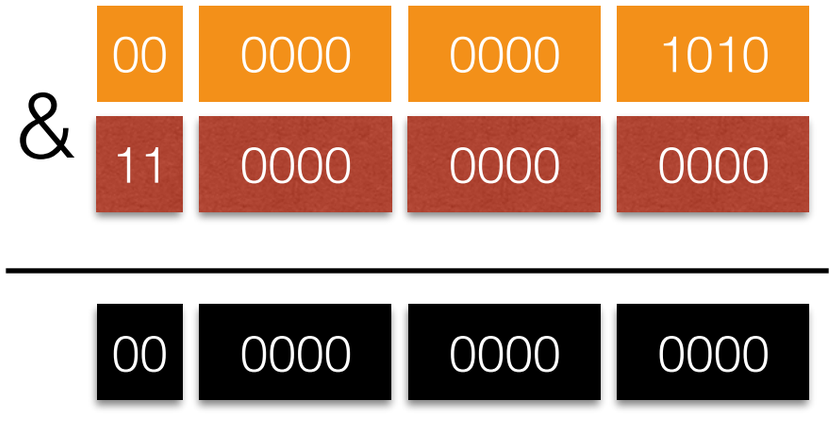
01 0111 1111 1111

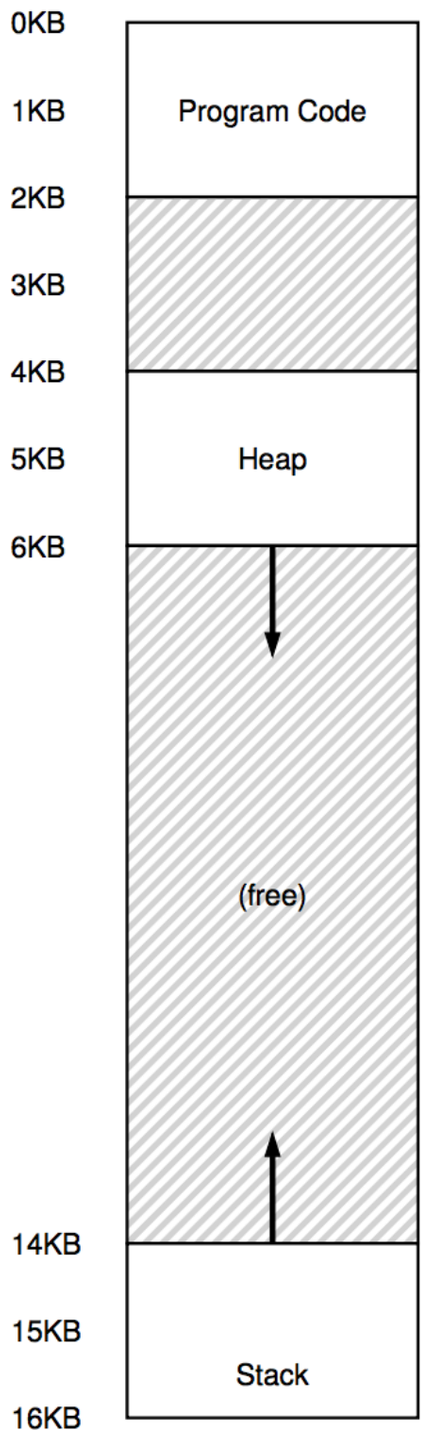
Segment = (VA & 11 0000 0000 0000) >>12

11 1000 0000 0000

11 1111 1111 1111

Segment Reference





00 0000 0000 0000

00 0111 1111 1111

01 0000 0000 0000

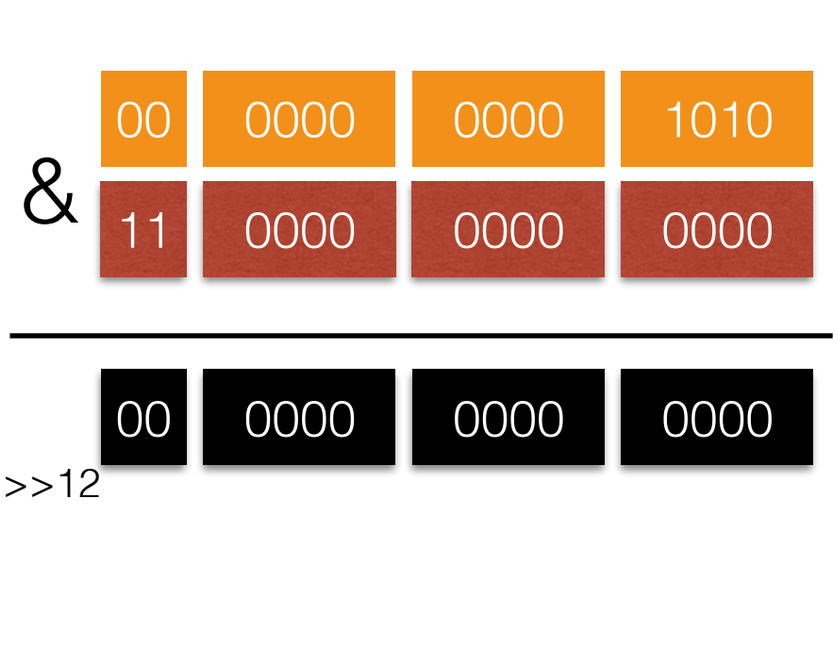
01 0111 1111 1111

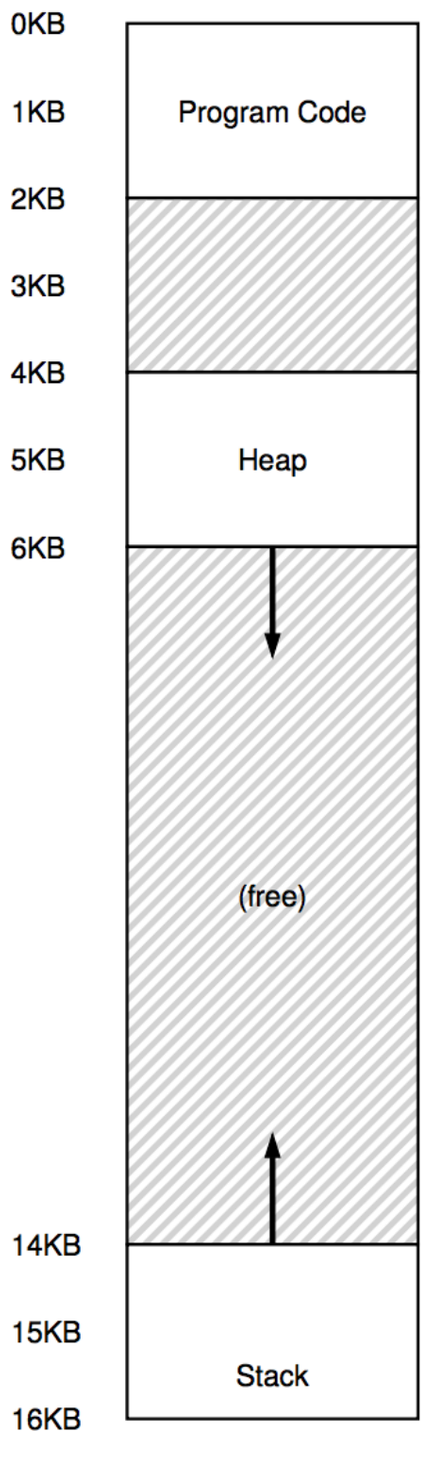
Segment = (VA & 11 0000 0000 0000) >>12

11 1000 0000 0000

11 1111 1111 1111

Segment Reference





00 0000 0000 0000

00 0111 1111 1111

01 0000 0000 0000

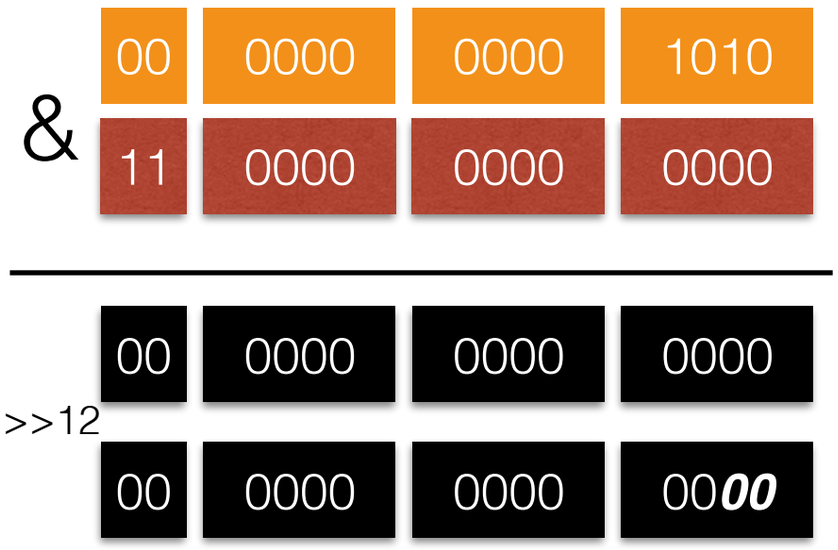
01 0111 1111 1111

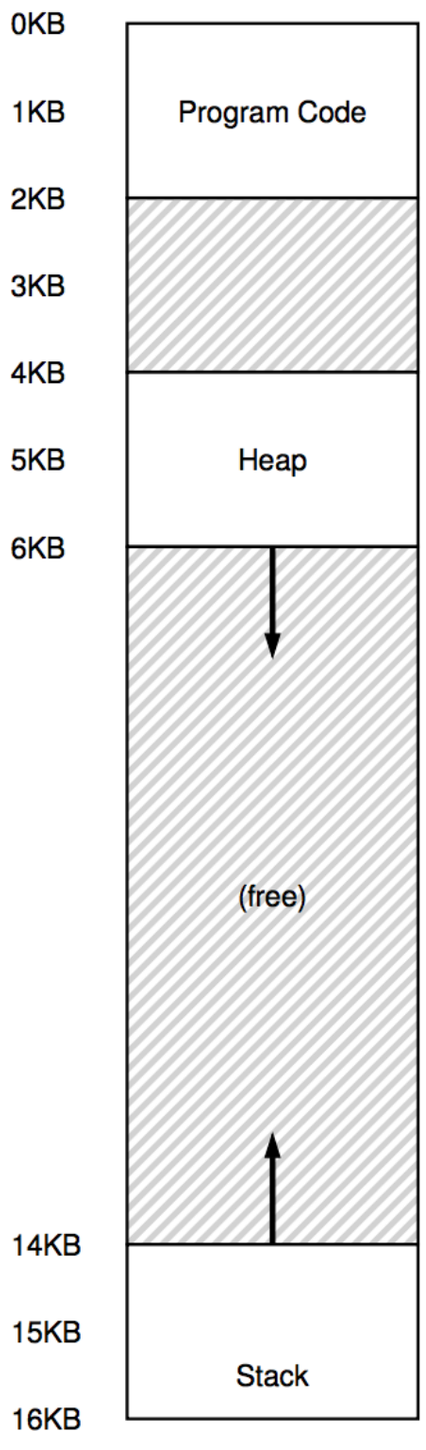
Segment = (VA & 11 0000 0000 0000) >>12

11 1000 0000 0000

11 1111 1111 1111

Segment Reference





00 0000 0000 0000

00 0111 1111 1111

01 0000 0000 0000

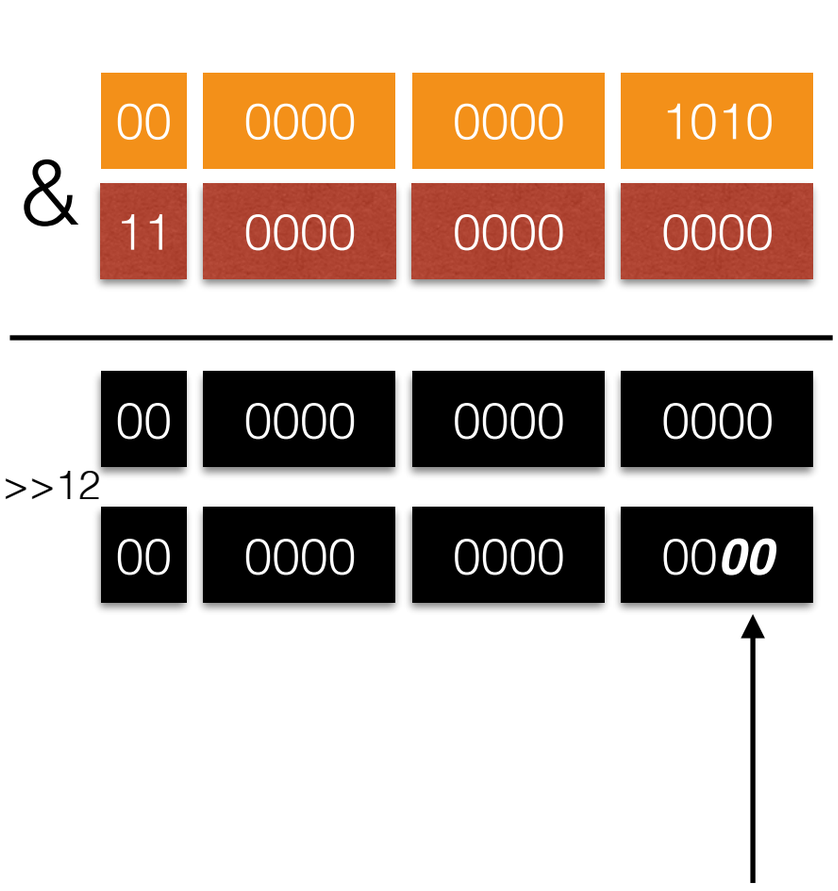
01 0111 1111 1111

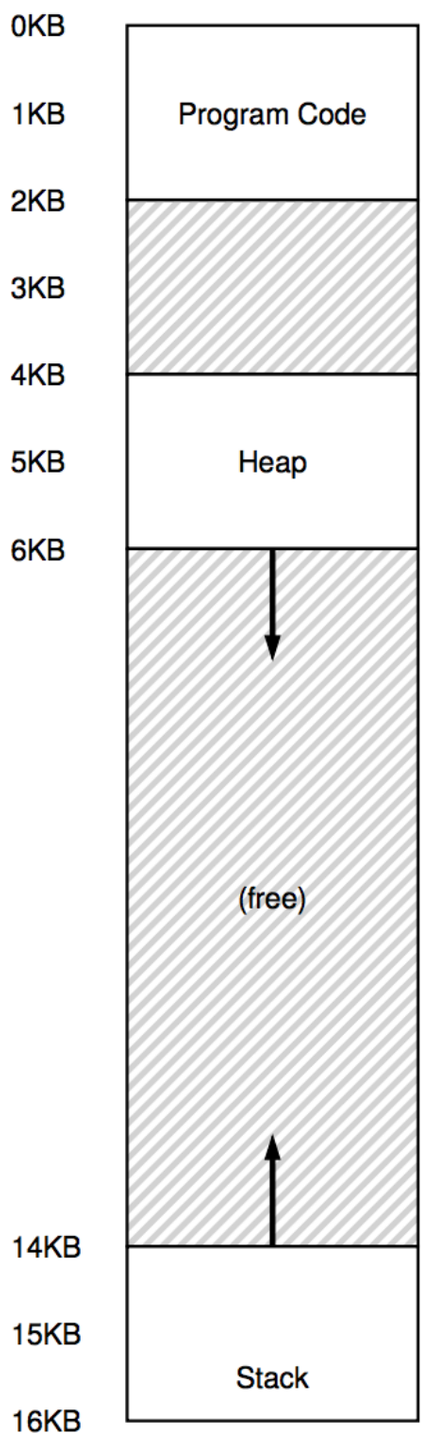
Segment = (VA & 11 0000 0000 0000) >>12

11 1000 0000 0000

11 1111 1111 1111

Segment Reference





00 0000 0000 0000

00 0111 1111 1111

01 0000 0000 0000

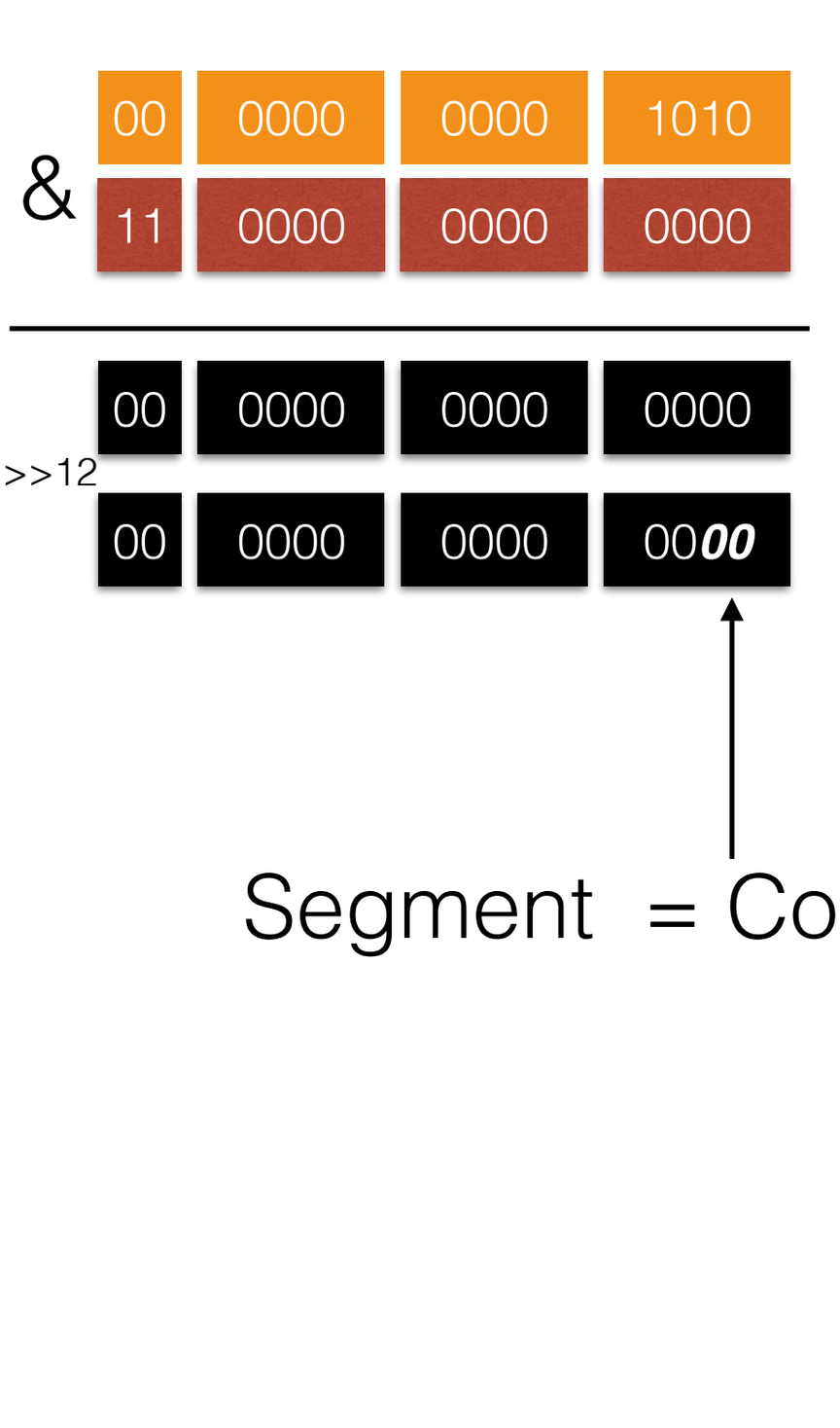
01 0111 1111 1111

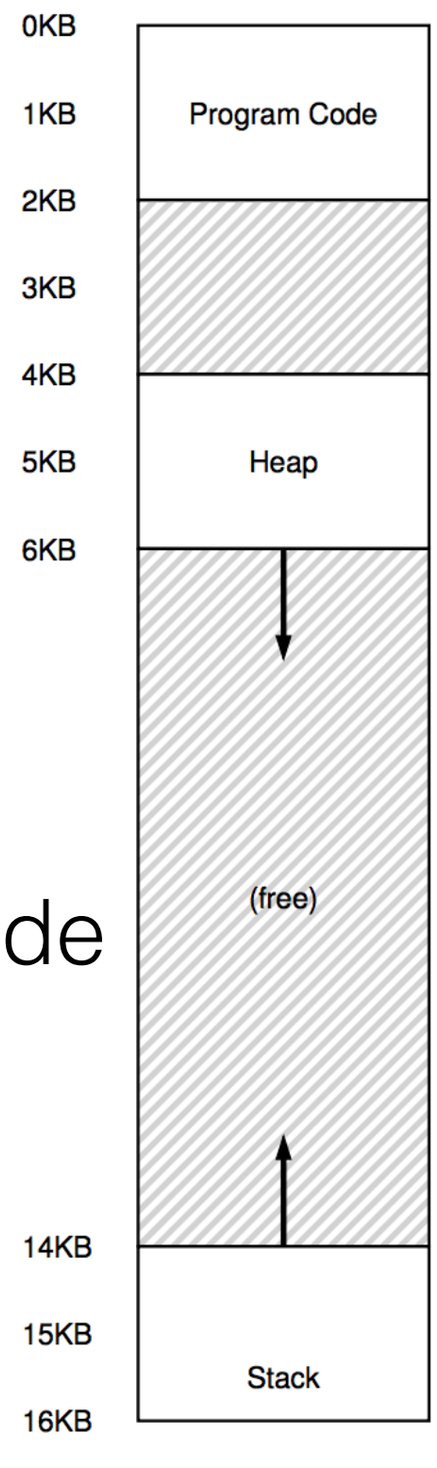
Segment = (VA & 11 0000 0000 0000) >>12

11 1000 0000 0000

11 1111 1111 1111

Segment Reference





00 0000 0000 0000

00 0111 1111 1111

01 0000 0000 0000

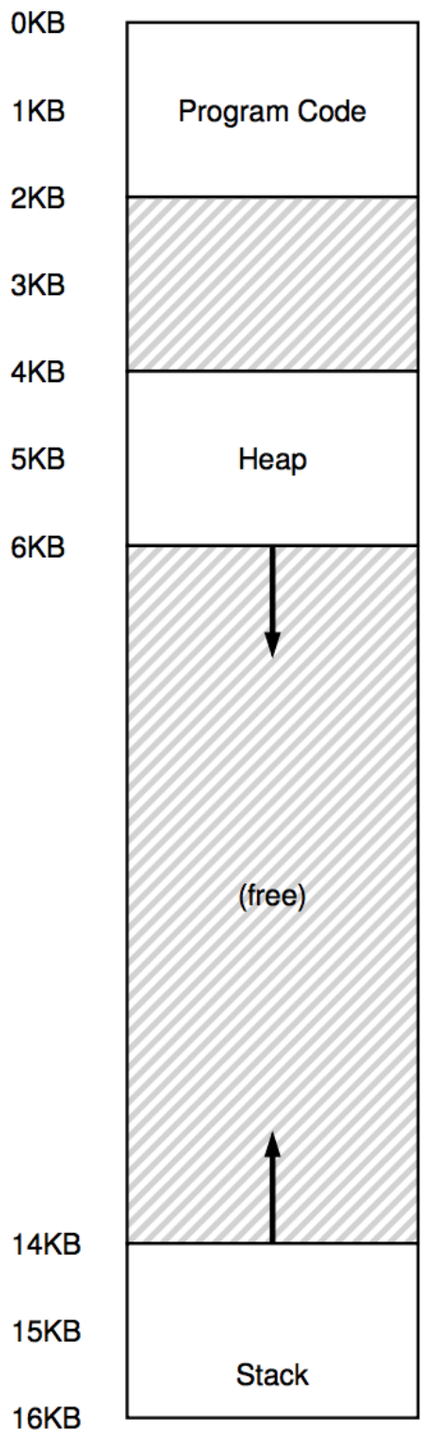
01 0111 1111 1111

Segment = (VA & 11 0000 0000 0000) >>12

11 1000 0000 0000

11 1111 1111 1111

Segment Reference



00 0000 0000 0000

00 0111 1111 1111

01 0000 0000 0000

01 0111 1111 1111

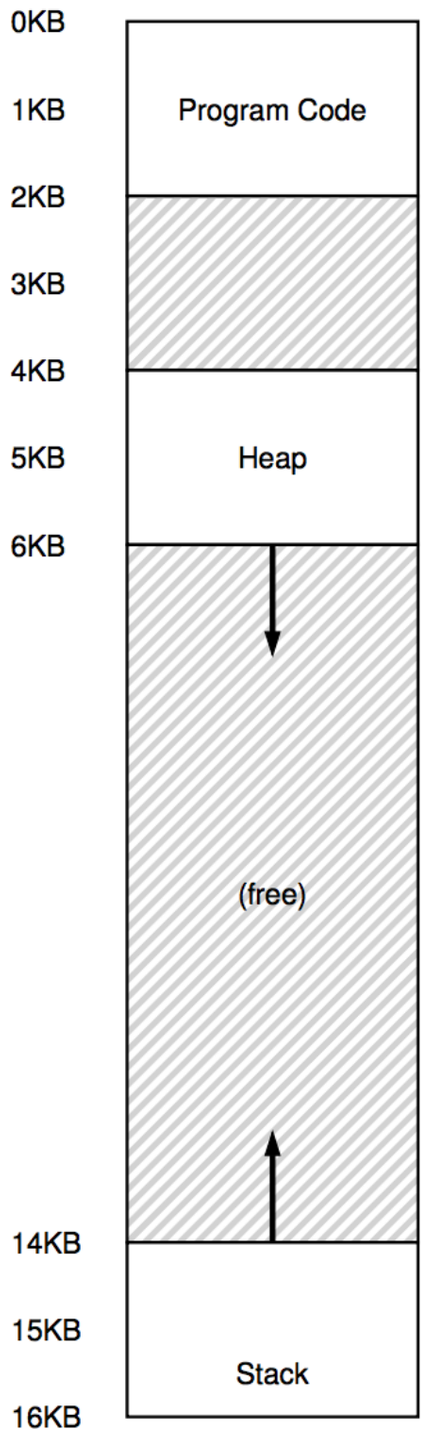
Segment = (VA & 11 0000 0000 0000) >>12

11 1000 0000 0000

11 1111 1111 1111

Segment Reference

01 | 0000 | 1010 | 1010



00 | 0000 | 0000 | 0000

00 | 0111 | 1111 | 1111

01 | 0000 | 0000 | 0000

01 | 0111 | 1111 | 1111

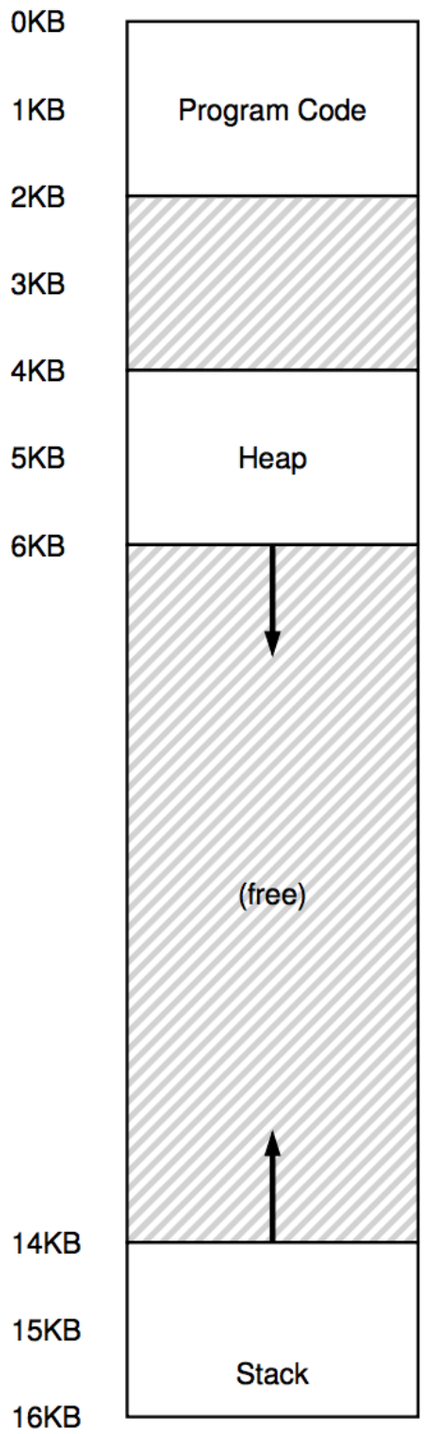
Segment = (VA & 11 0000 0000 0000) >>12

11 | 1000 | 0000 | 0000

11 | 1111 | 1111 | 1111

Segment Reference

01	0000	1010	1010
11	0000	0000	0000

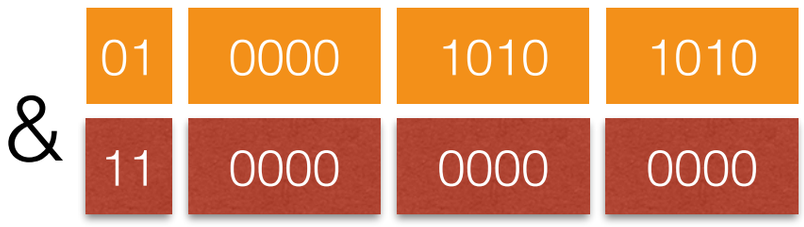


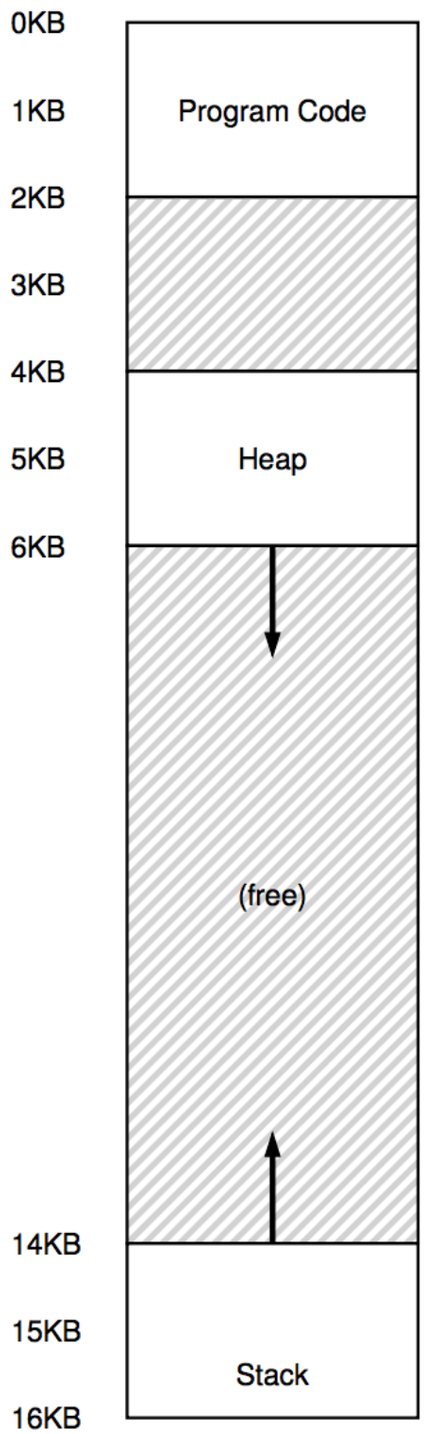
00	0000	0000	0000
00	0111	1111	1111
01	0000	0000	0000
01	0111	1111	1111

Segment = (VA & 11 0000 0000 0000) >>12

11	1000	0000	0000
11	1111	1111	1111

Segment Reference





00	0000	0000	0000
00	0111	1111	1111
01	0000	0000	0000
01	0111	1111	1111

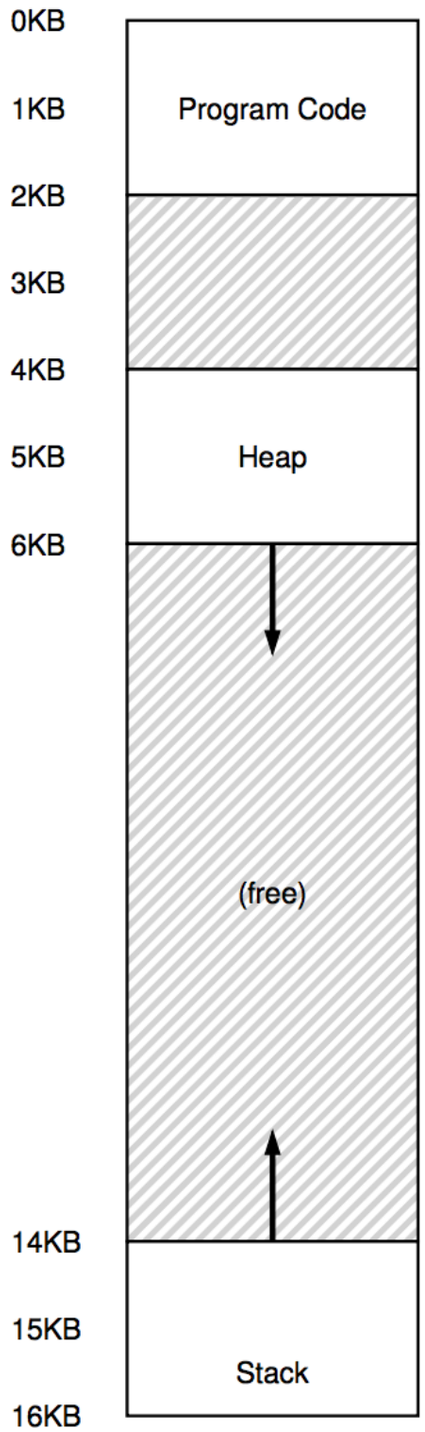
Segment = (VA & 11 0000 0000 0000) >>12

11	1000	0000	0000
11	1111	1111	1111

Segment Reference

01 0000 1010 1010

& 11 0000 0000 0000



00 0000 0000 0000

00 0111 1111 1111

01 0000 0000 0000

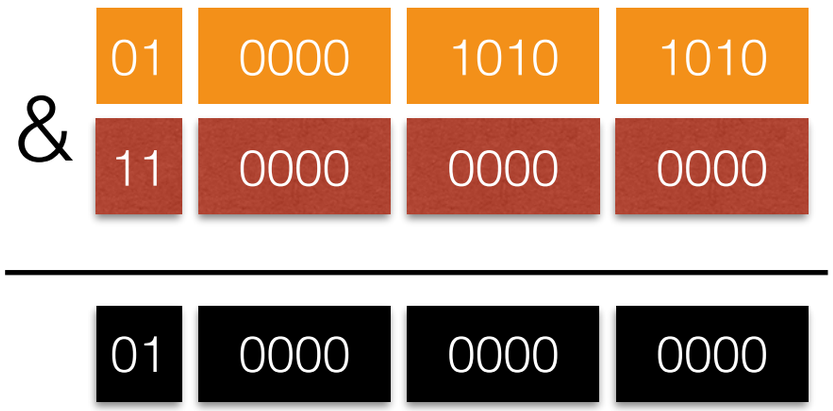
01 0111 1111 1111

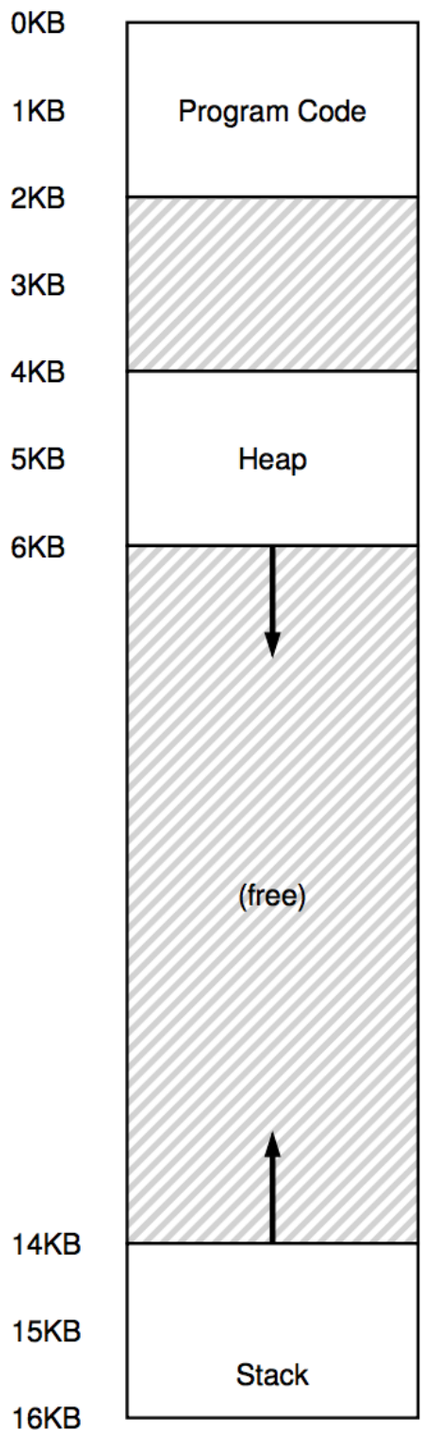
Segment = (VA & 11 0000 0000 0000) >>12

11 1000 0000 0000

11 1111 1111 1111

Segment Reference





00 0000 0000 0000

00 0111 1111 1111

01 0000 0000 0000

01 0111 1111 1111

Segment = (VA & 11 0000 0000 0000) >>12

11 1000 0000 0000

11 1111 1111 1111

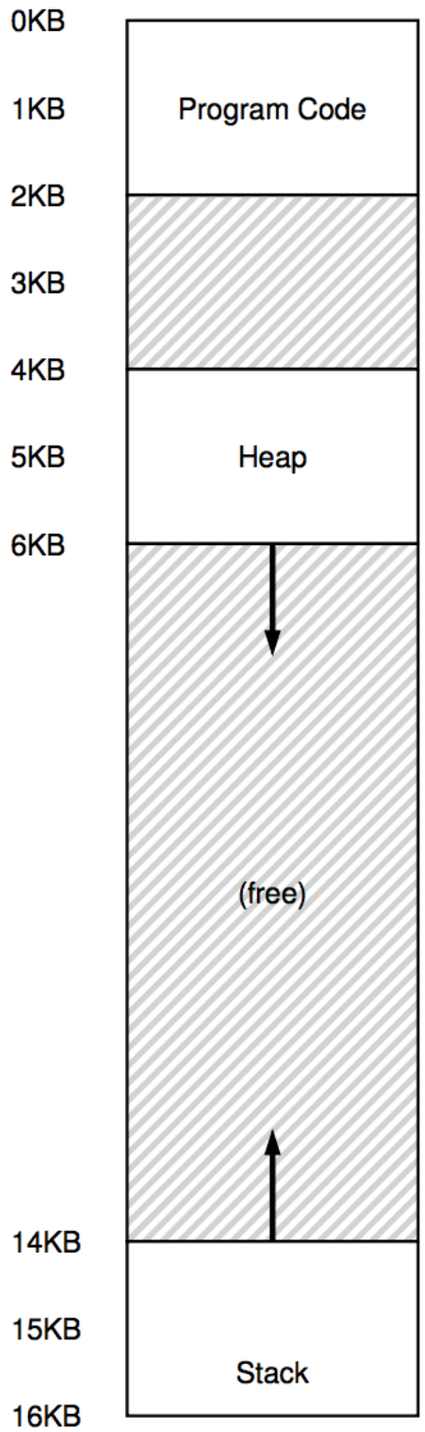
Segment Reference

01 0000 1010 1010

& 11 0000 0000 0000

01 0000 0000 0000

>>12



00 0000 0000 0000

00 0111 1111 1111

01 0000 0000 0000

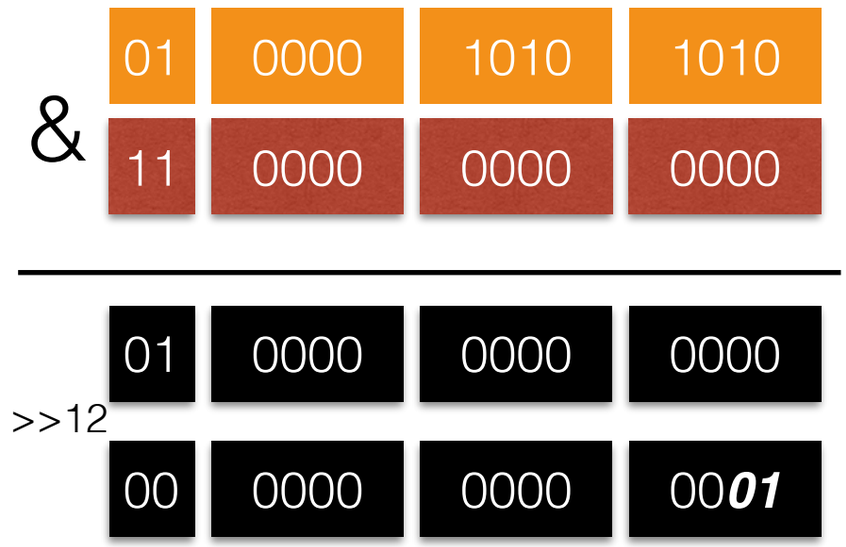
01 0111 1111 1111

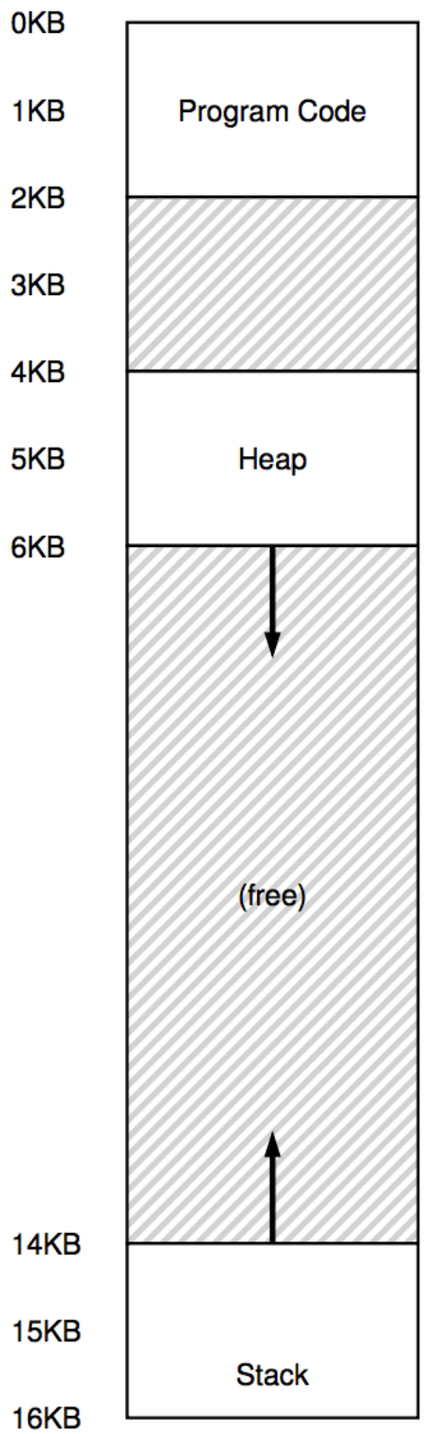
Segment = (VA & 11 0000 0000 0000) >>12

11 1000 0000 0000

11 1111 1111 1111

Segment Reference





00 0000 0000 0000

00 0111 1111 1111

01 0000 0000 0000

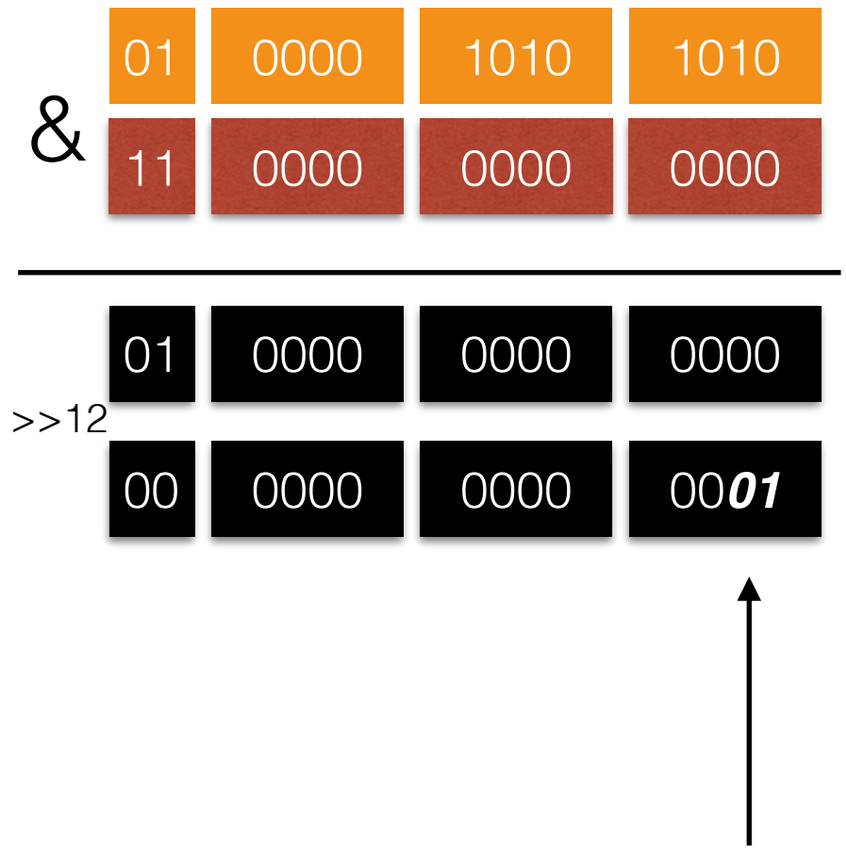
01 0111 1111 1111

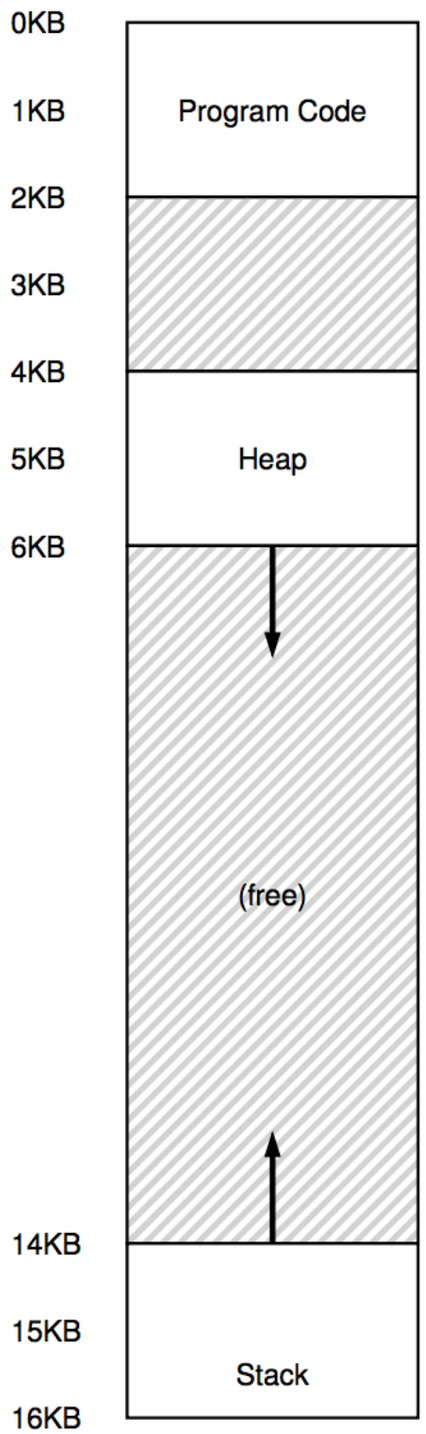
Segment = (VA & 11 0000 0000 0000) >>12

11 1000 0000 0000

11 1111 1111 1111

Segment Reference



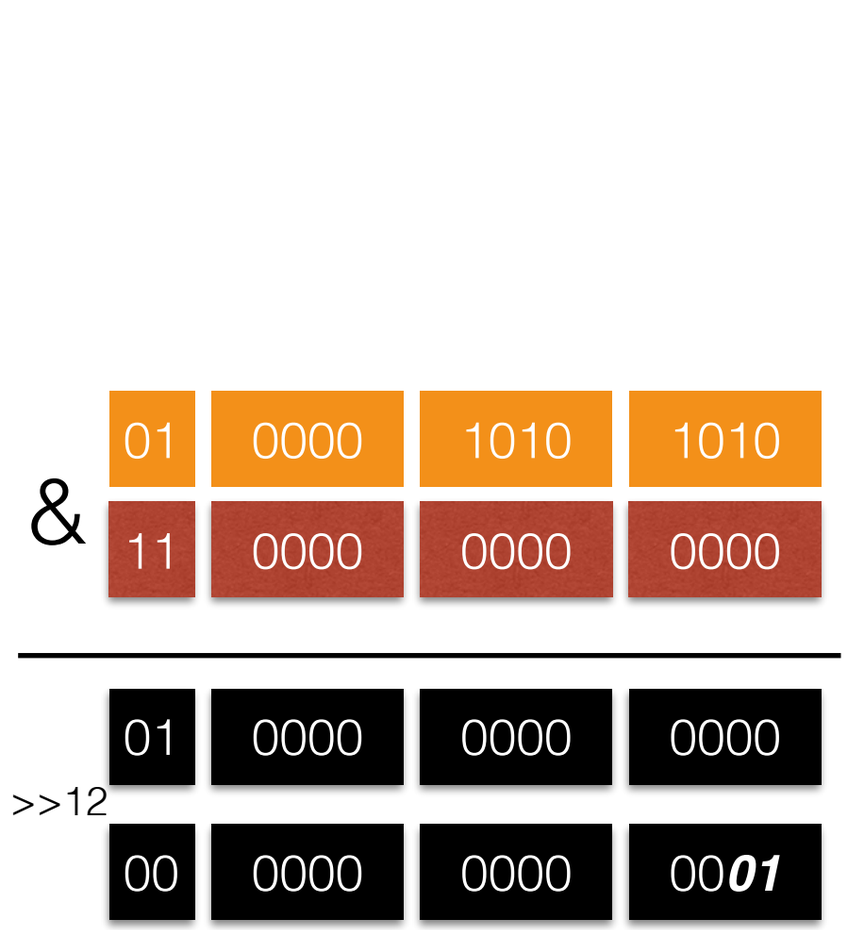


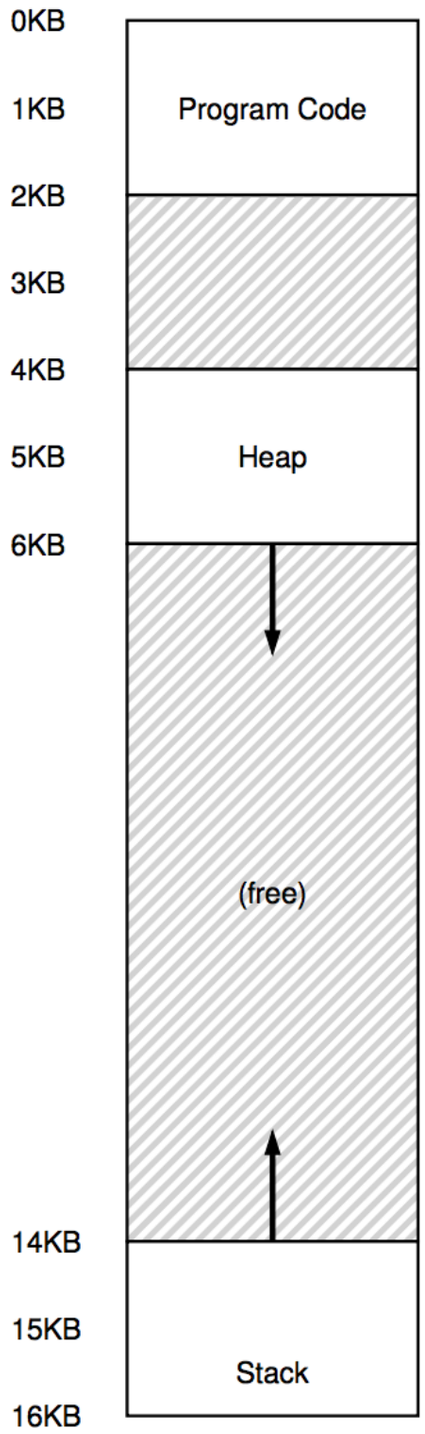
00	0000	0000	0000
00	0111	1111	1111
01	0000	0000	0000
01	0111	1111	1111

Segment = (VA & 11 0000 0000 0000) >>12

11	1000	0000	0000
11	1111	1111	1111

Segment Reference





00 0000 0000 0000

00 0111 1111 1111

01 0000 0000 0000

01 0111 1111 1111

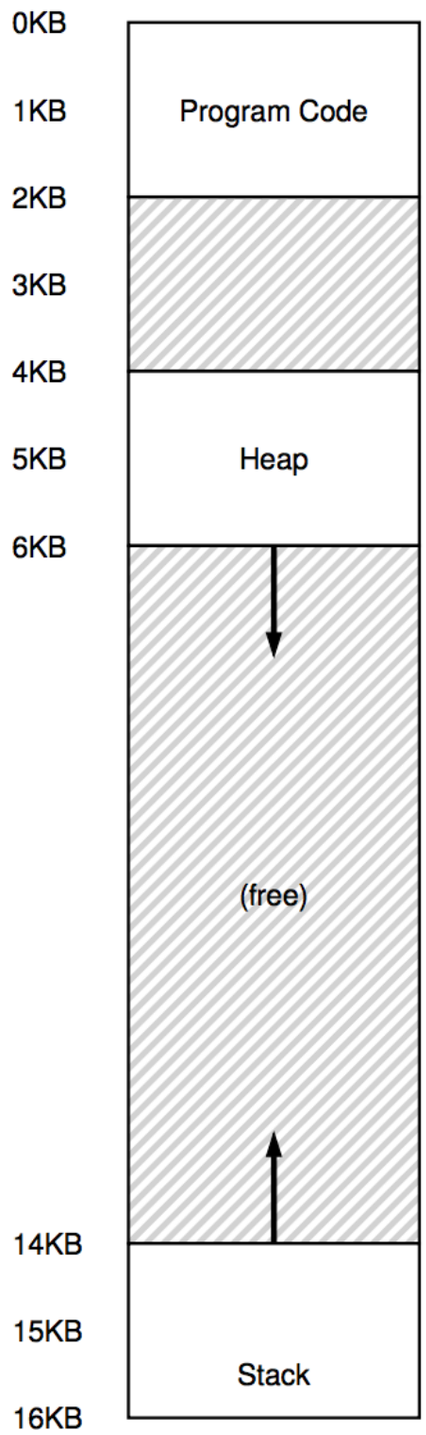
Segment = (VA & 11 0000 0000 0000) >>12

11 1000 0000 0000

11 1111 1111 1111

Segment Reference

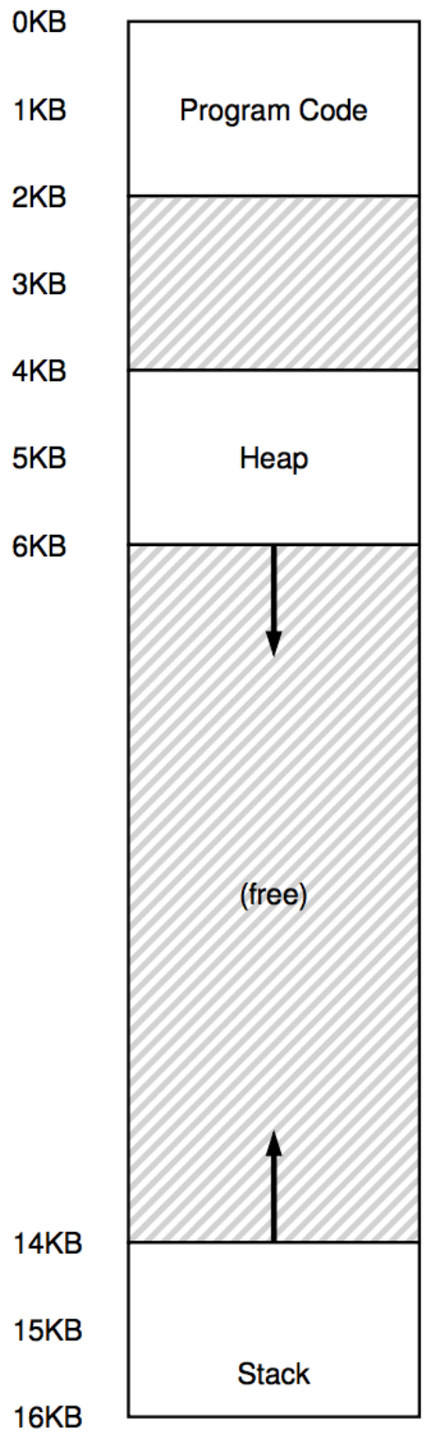
Virtual Address Space

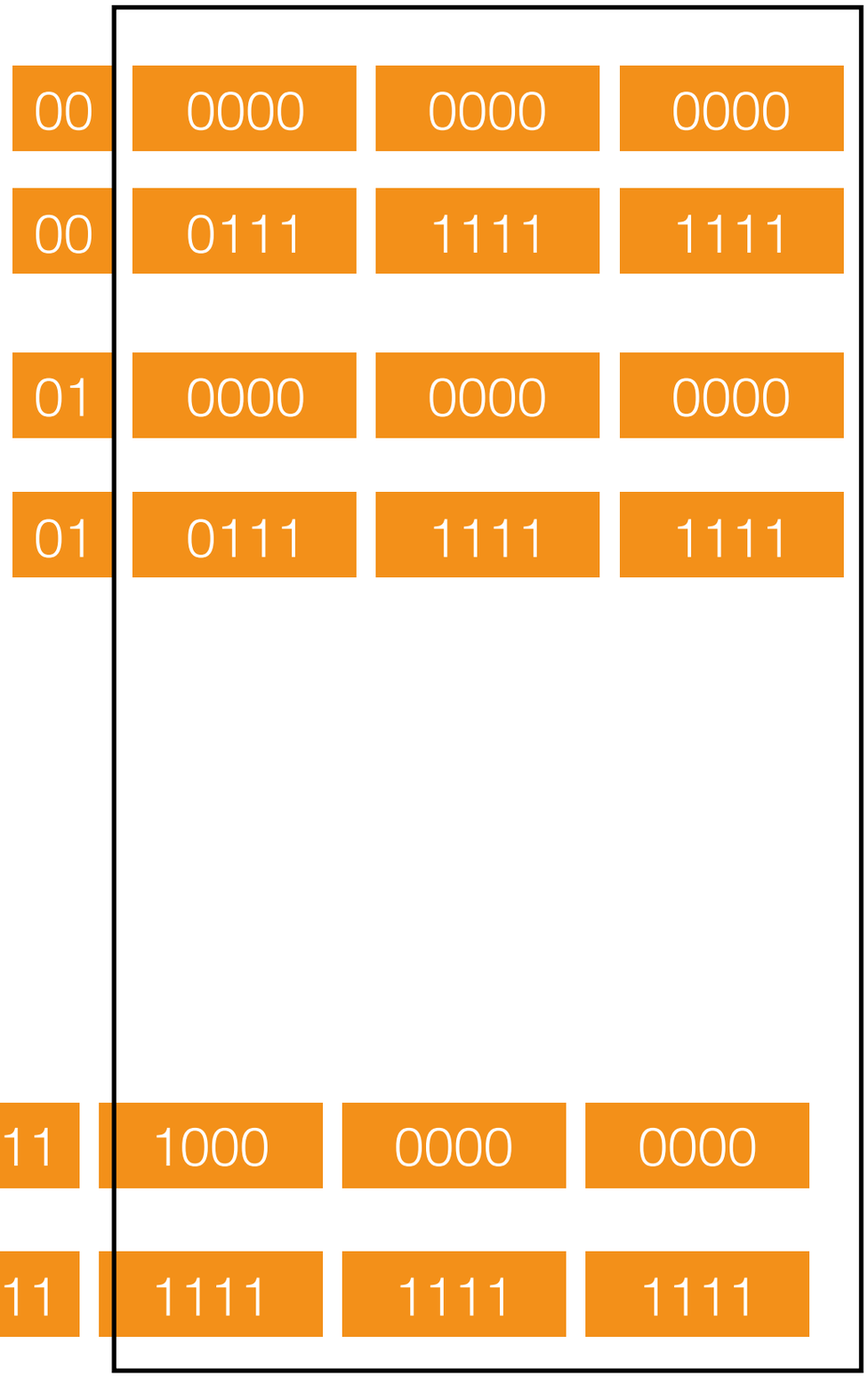


00	0000	0000	0000
00	0111	1111	1111
01	0000	0000	0000
01	0111	1111	1111
11	1000	0000	0000
11	1111	1111	1111

Segment Reference

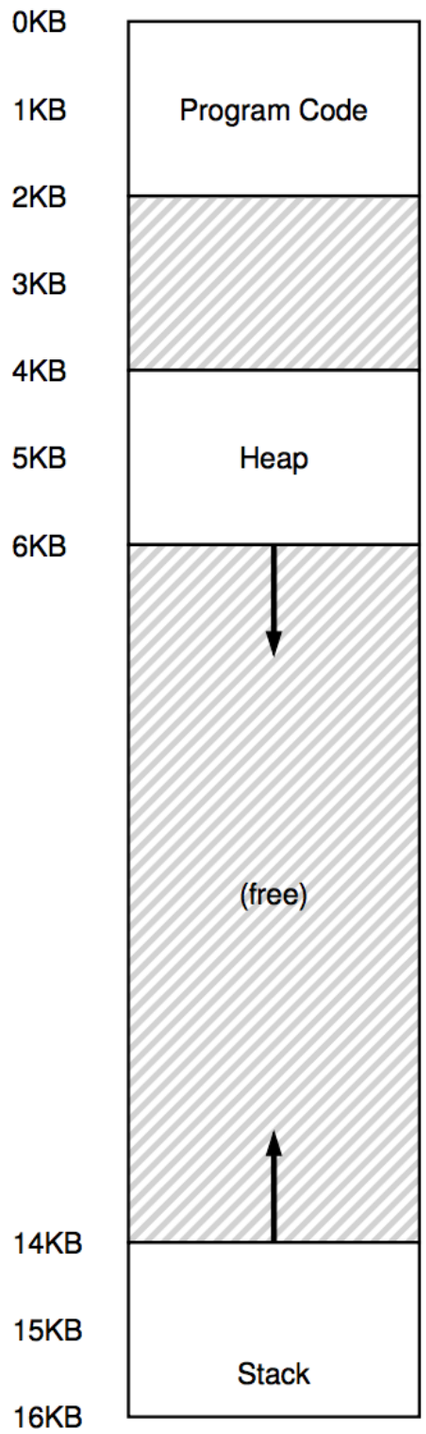
Virtual Address Space

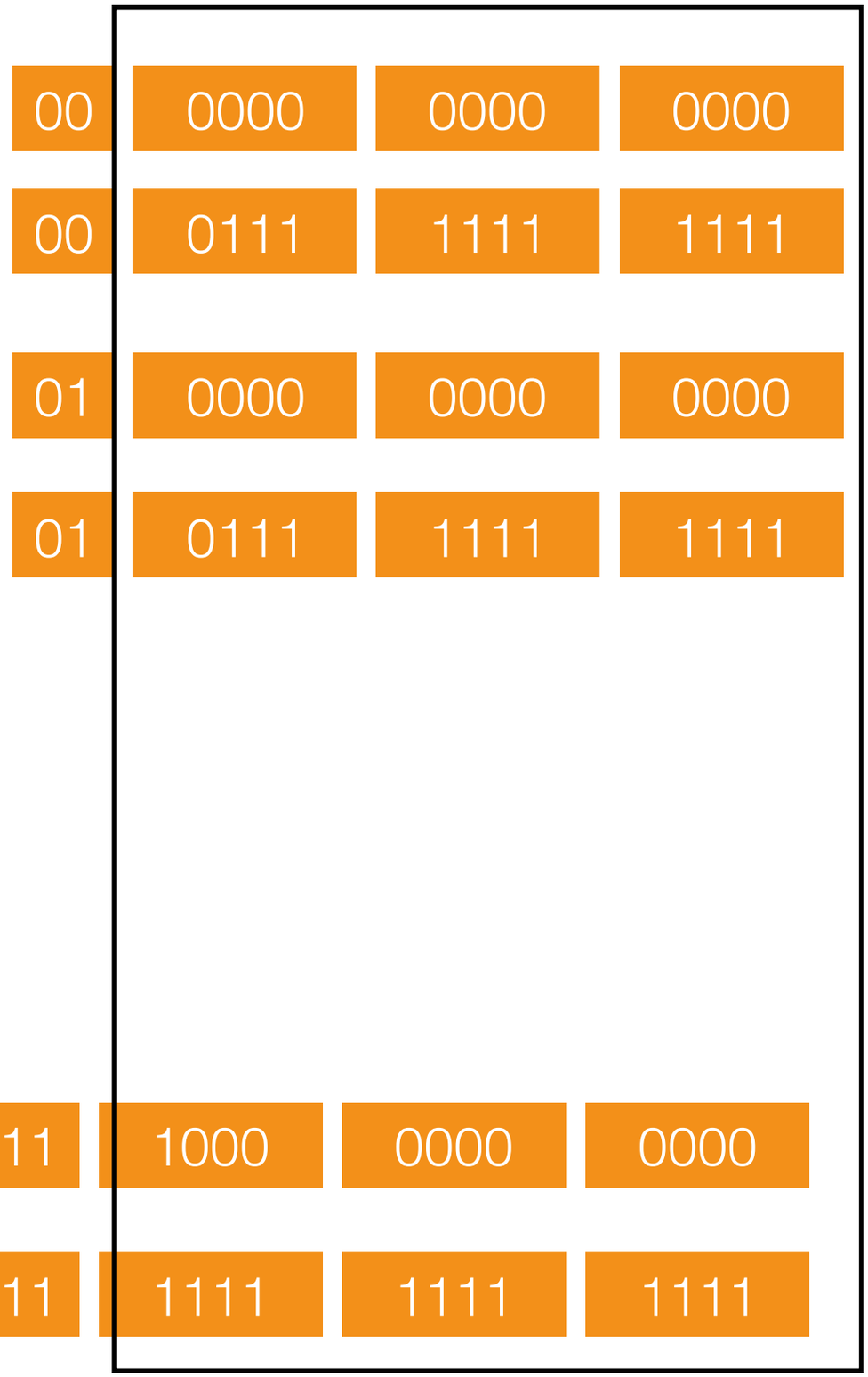




Segment Reference

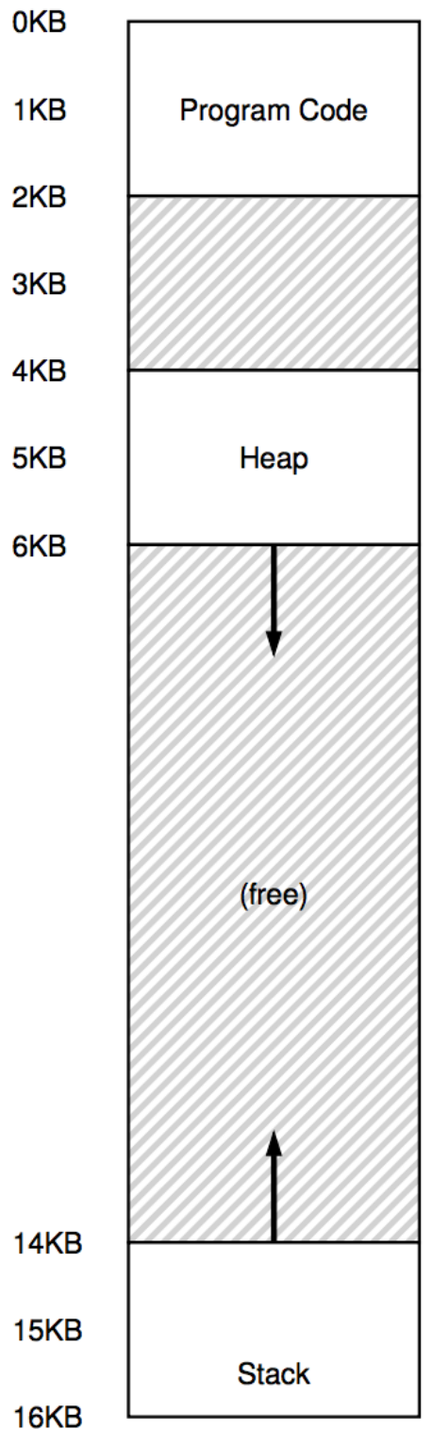
Virtual Address Space

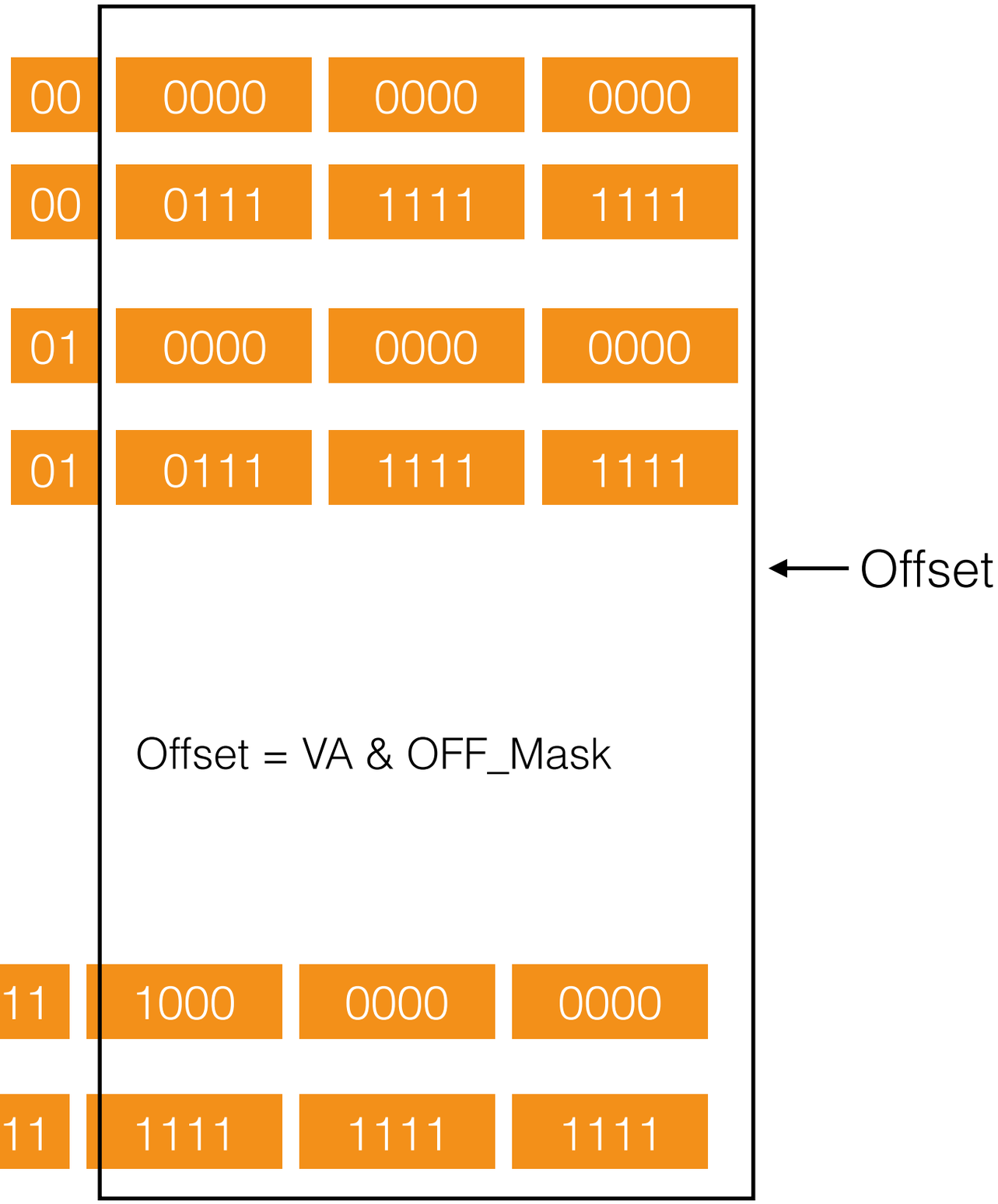




Segment Reference

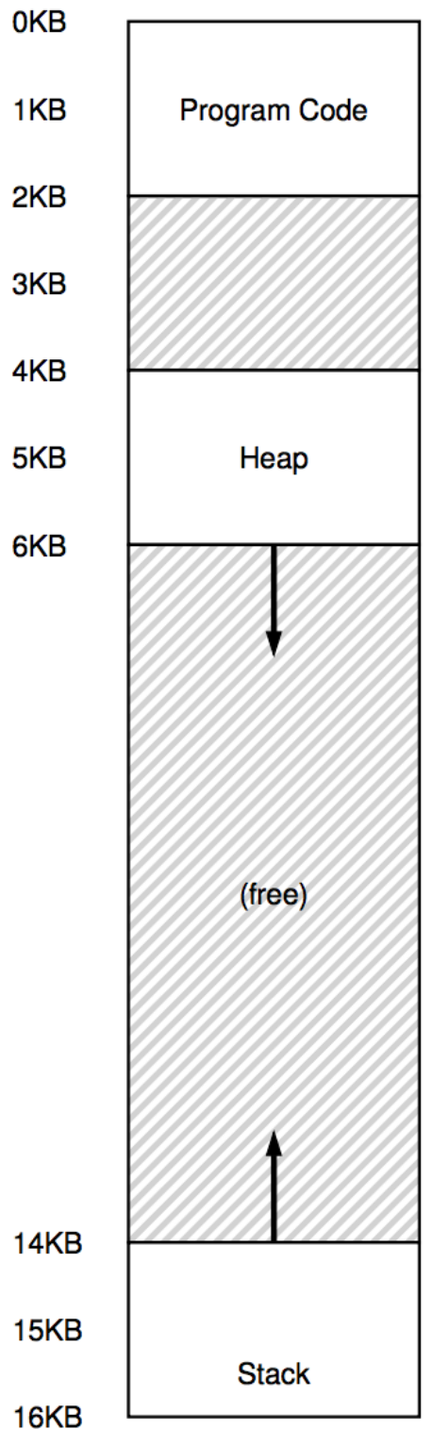
Virtual Address Space

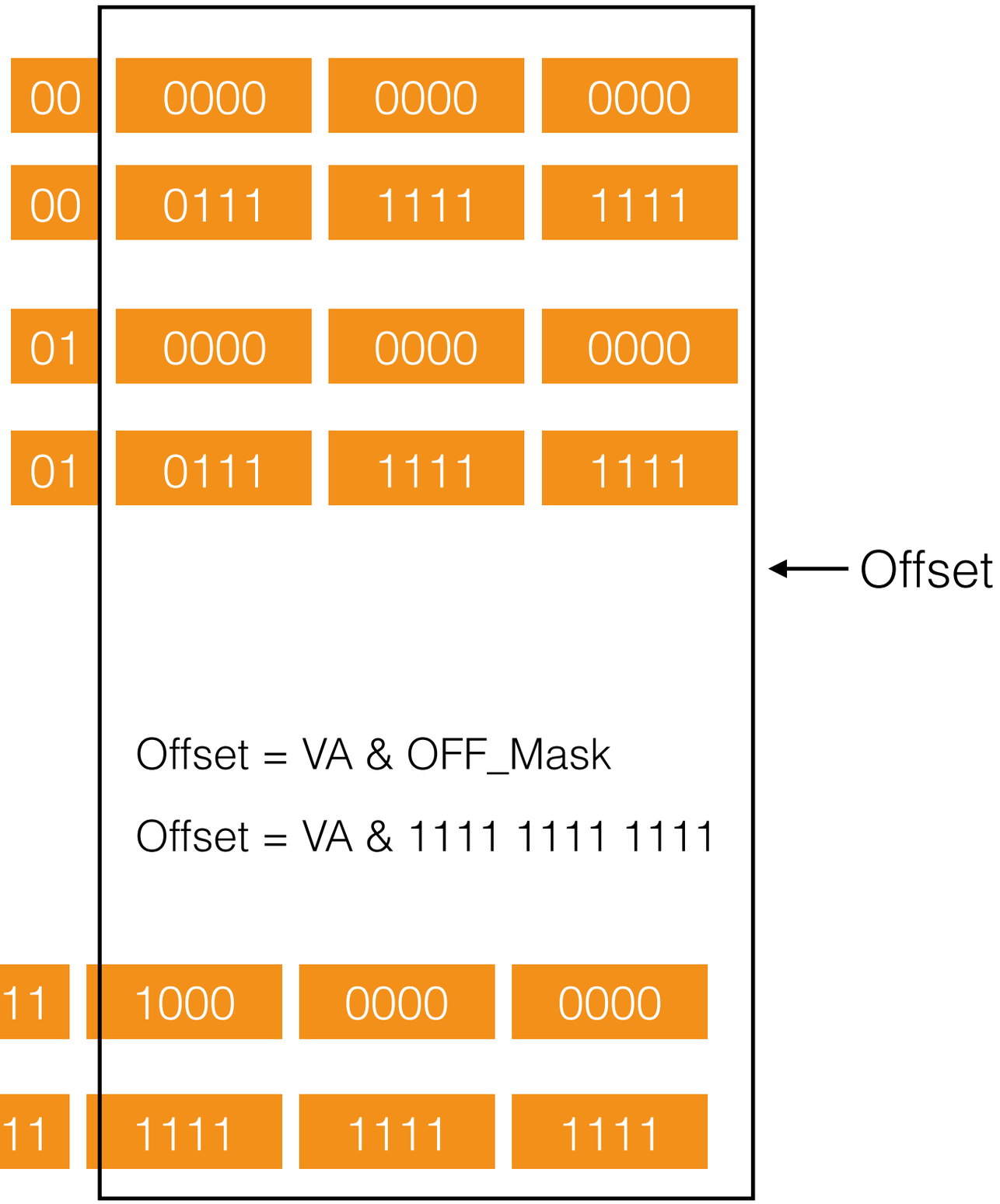




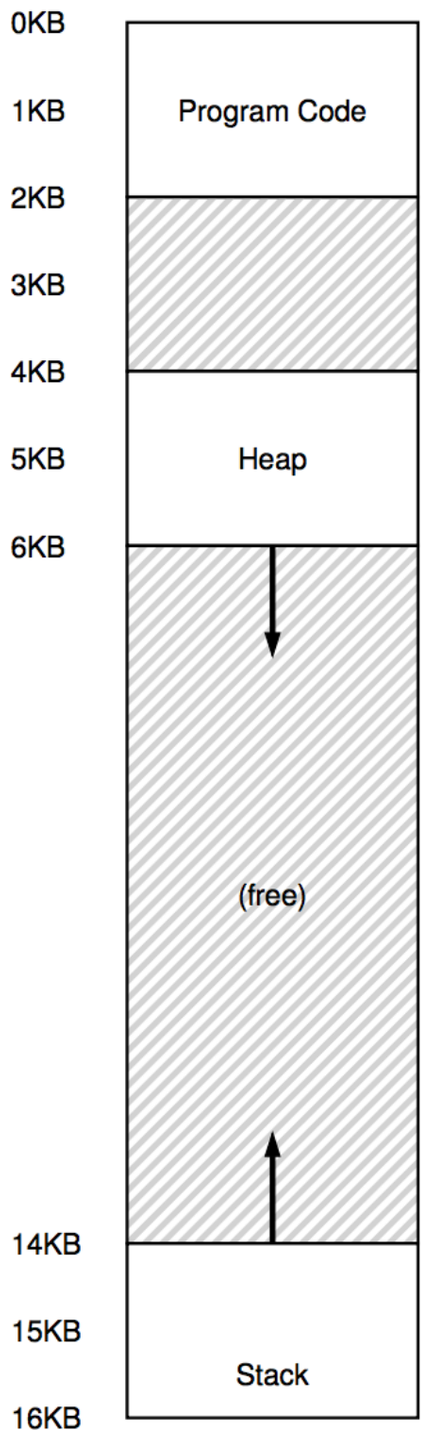
Segment Reference

Virtual Address Space





Segment Reference



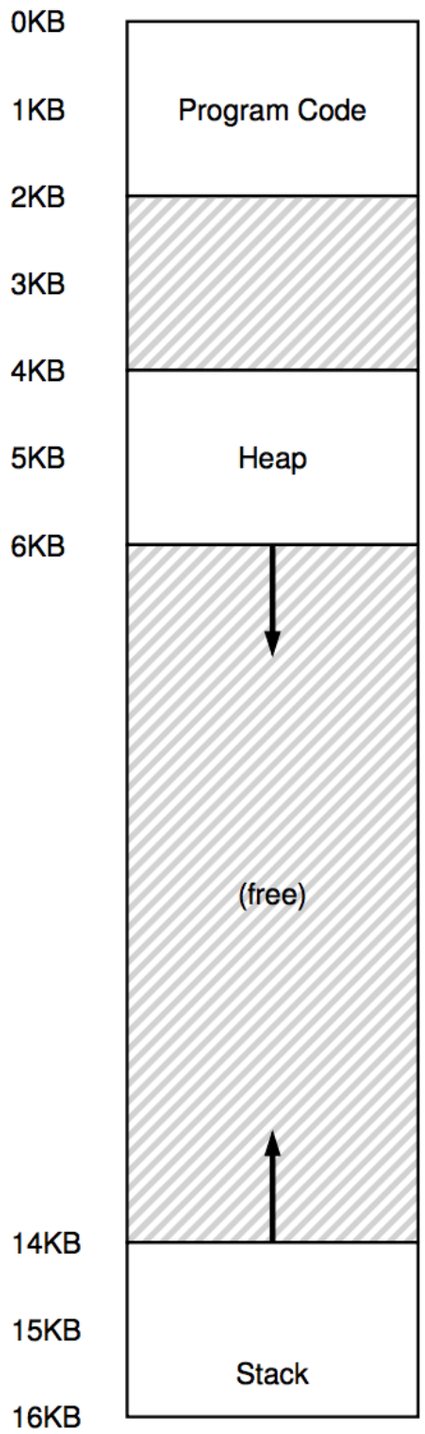
00	0000	0000	0000
00	0111	1111	1111
01	0000	0000	0000
01	0111	1111	1111

Offset = VA & OFF_Mask

11	1000	0000	0000
11	1111	1111	1111

Segment Reference

00 | 0000 | 0000 | 1010



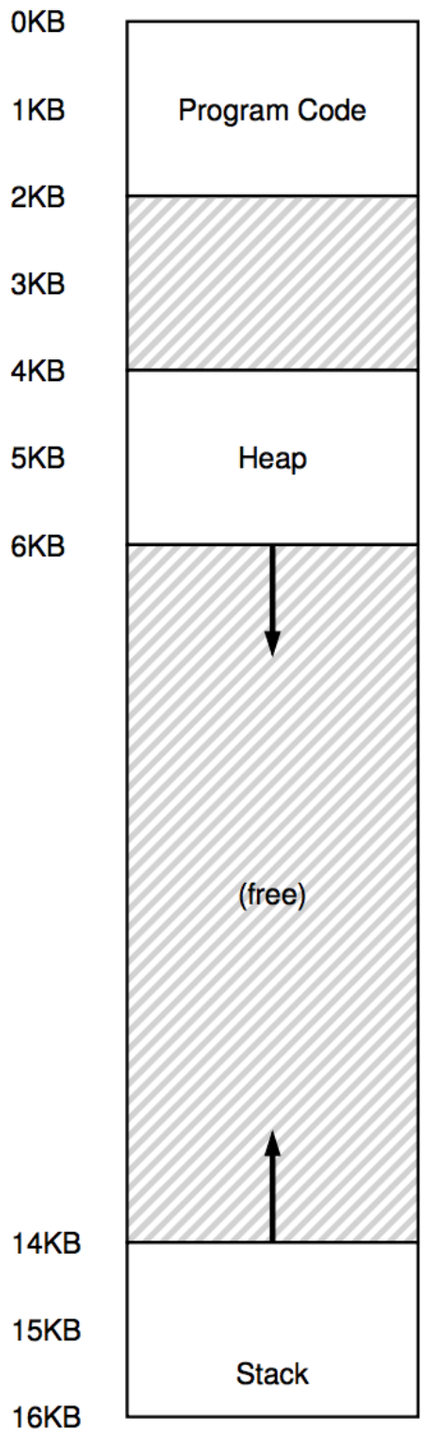
00	0000	0000	0000
00	0111	1111	1111
01	0000	0000	0000
01	0111	1111	1111

Offset = VA & OFF_Mask

11	1000	0000	0000
11	1111	1111	1111

Segment Reference

00	0000	0000	1010
00	1111	1111	1111



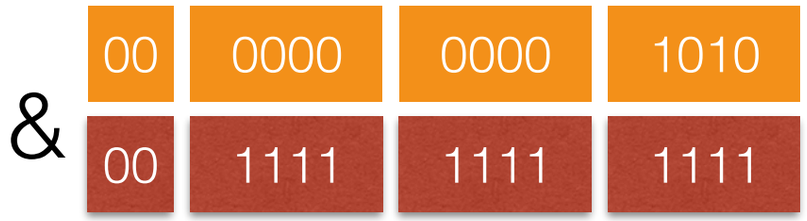
00	0000	0000	0000
00	0111	1111	1111

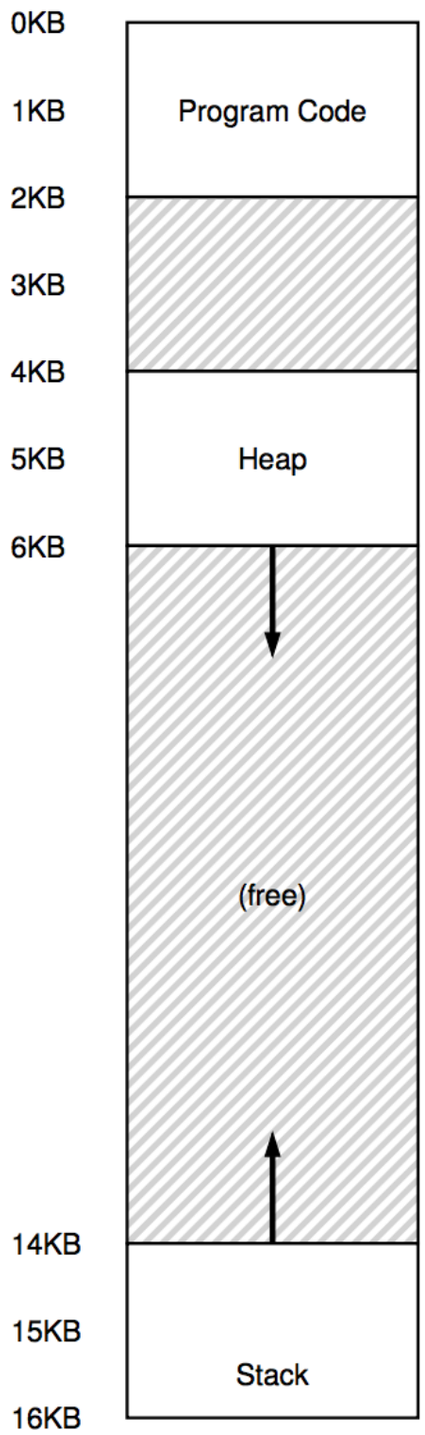
01	0000	0000	0000
01	0111	1111	1111

Offset = VA & OFF_Mask

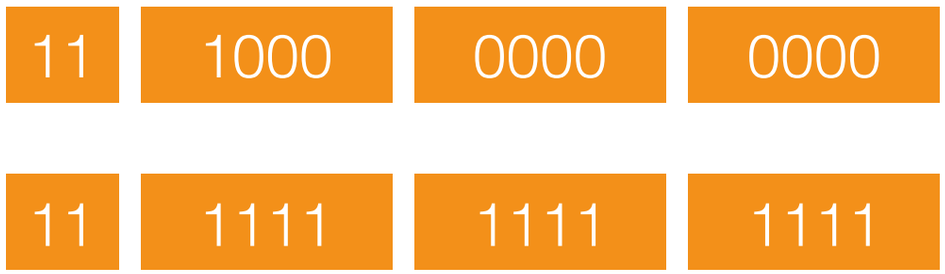
11	1000	0000	0000
11	1111	1111	1111

Segment Reference

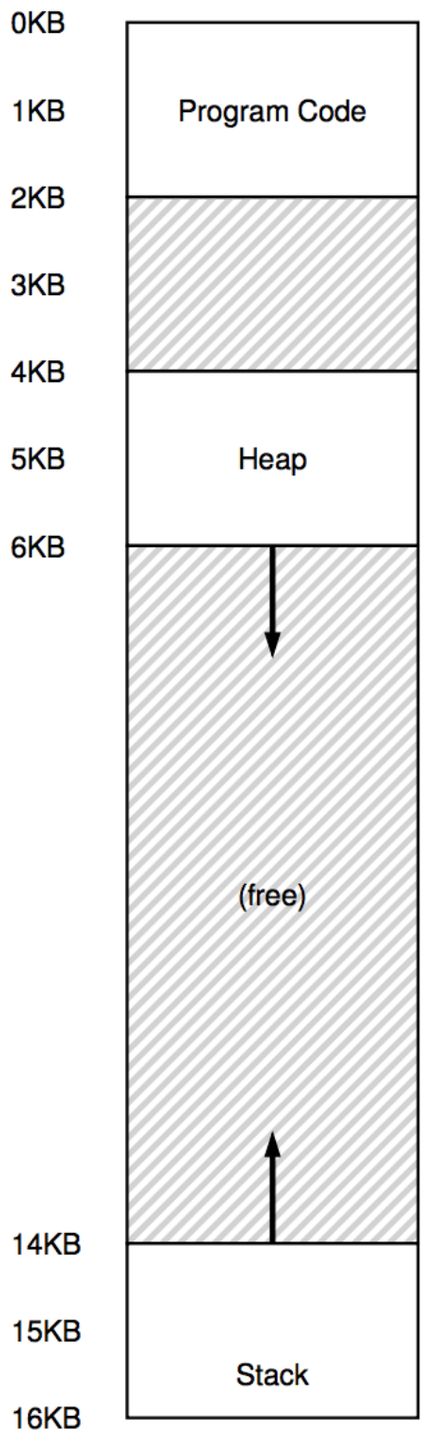




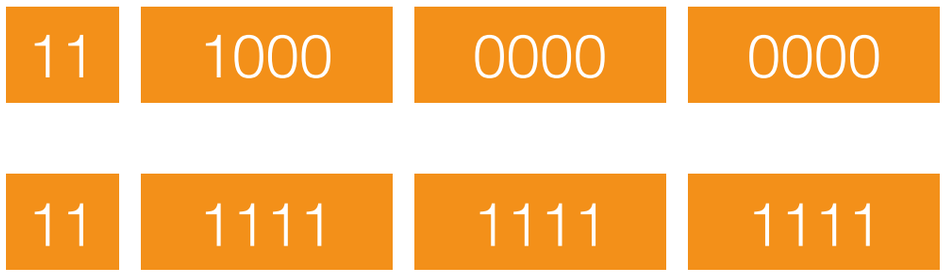
Offset = VA & OFF_Mask



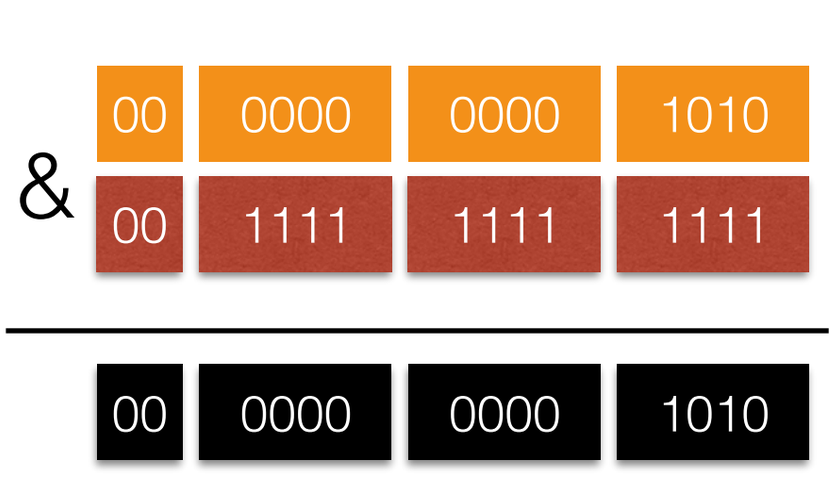
Segment Reference

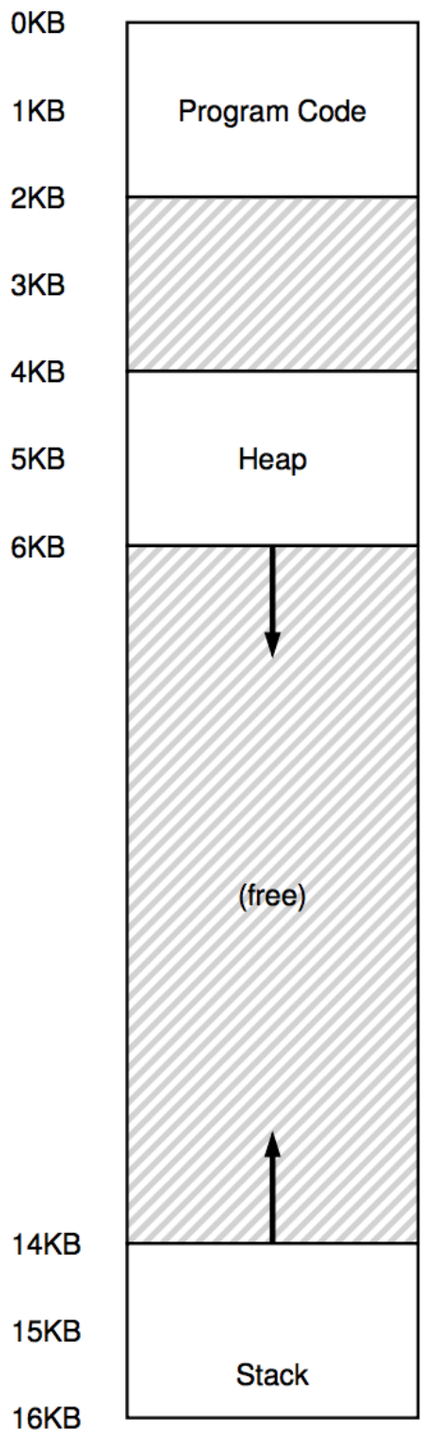


Offset = VA & OFF_Mask



Segment Reference





00 0000 0000 0000

00 0111 1111 1111

01 0000 0000 0000

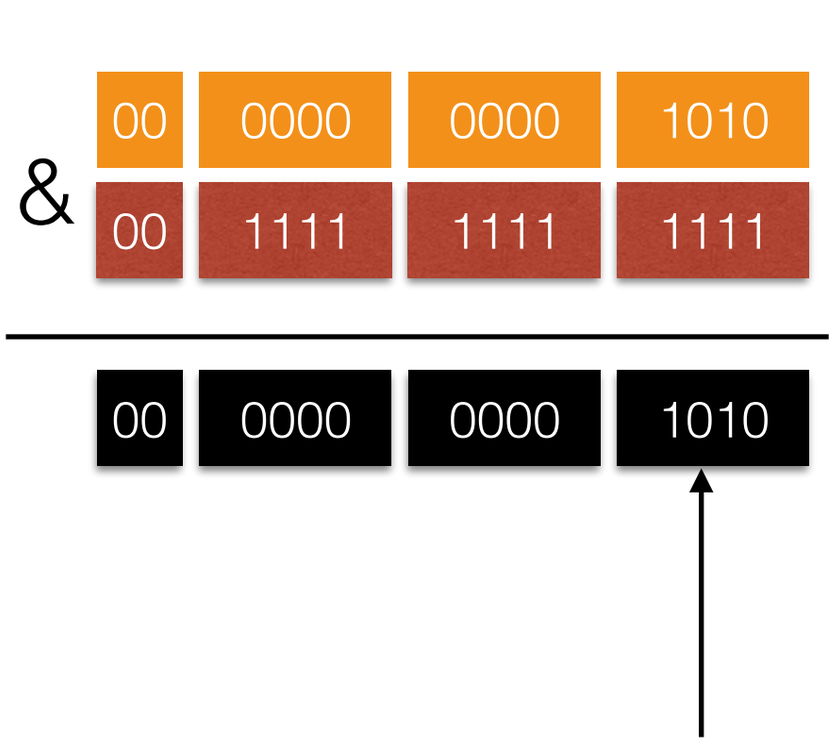
01 0111 1111 1111

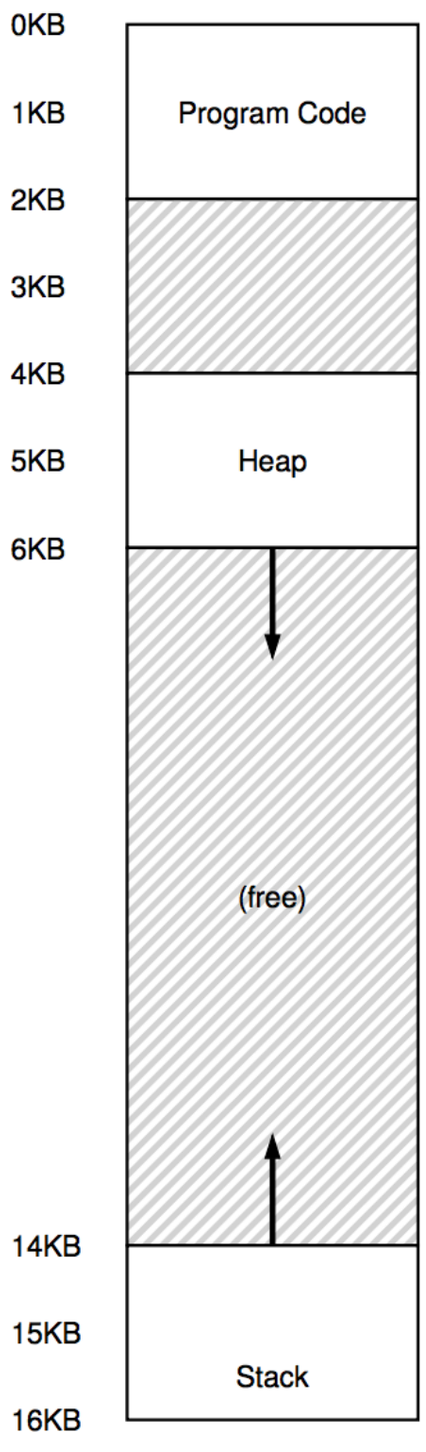
Offset = VA & OFF_Mask

11 1000 0000 0000

11 1111 1111 1111

Segment Reference





00 0000 0000 0000

00 0111 1111 1111

01 0000 0000 0000

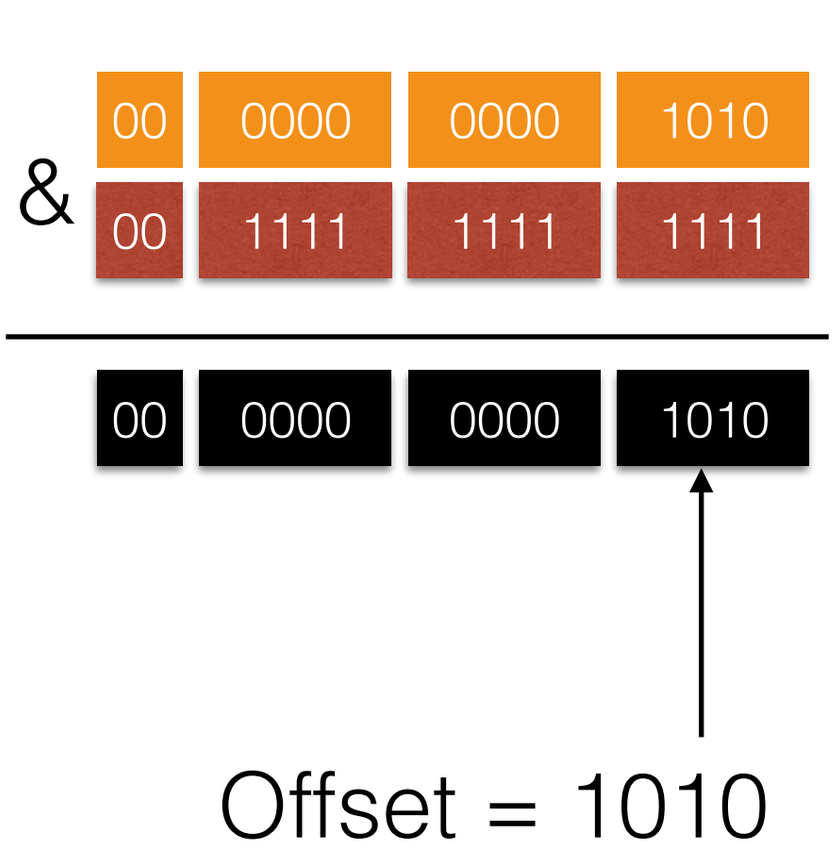
01 0111 1111 1111

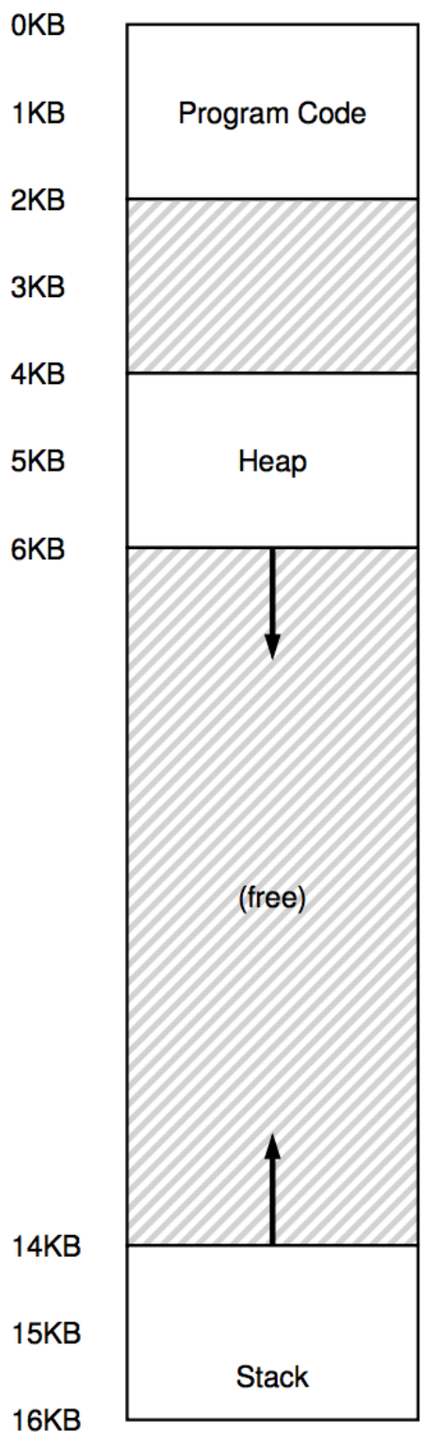
Offset = VA & OFF_Mask

11 1000 0000 0000

11 1111 1111 1111

Segment Reference





00 0000 0000 0000

00 0111 1111 1111

01 0000 0000 0000

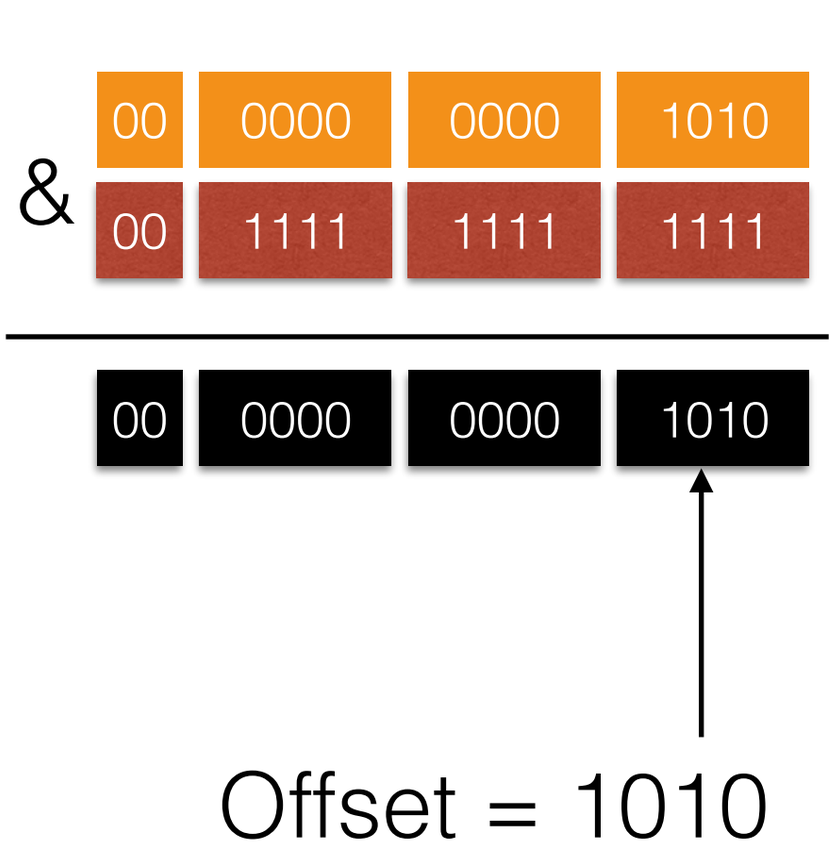
01 0111 1111 1111

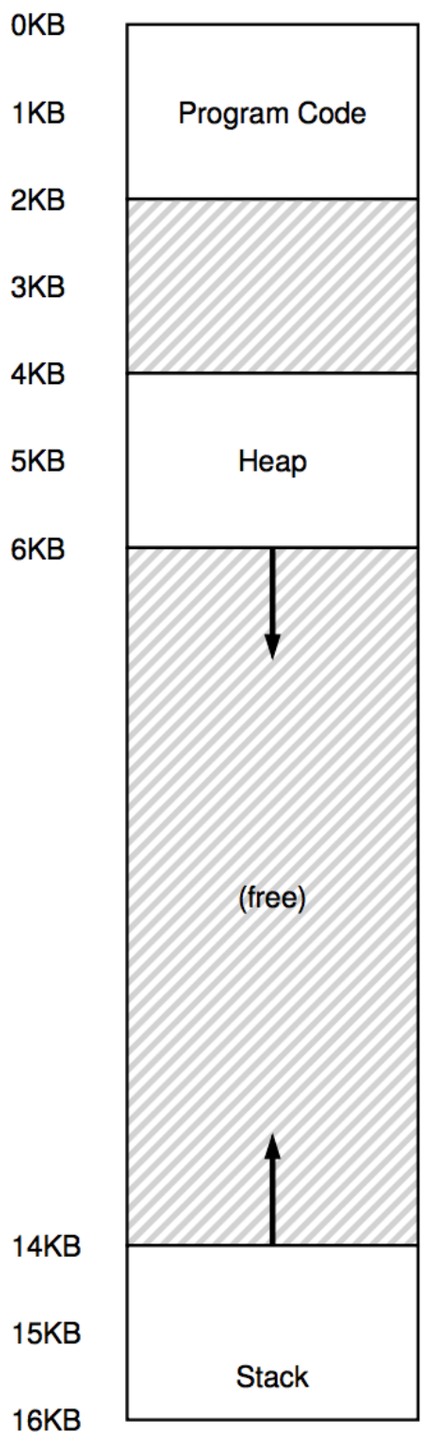
Offset = VA & OFF_Mask

11 1000 0000 0000

11 1111 1111 1111

Segment Reference





00 0000 0000 0000

00 0111 1111 1111

01 0000 0000 0000

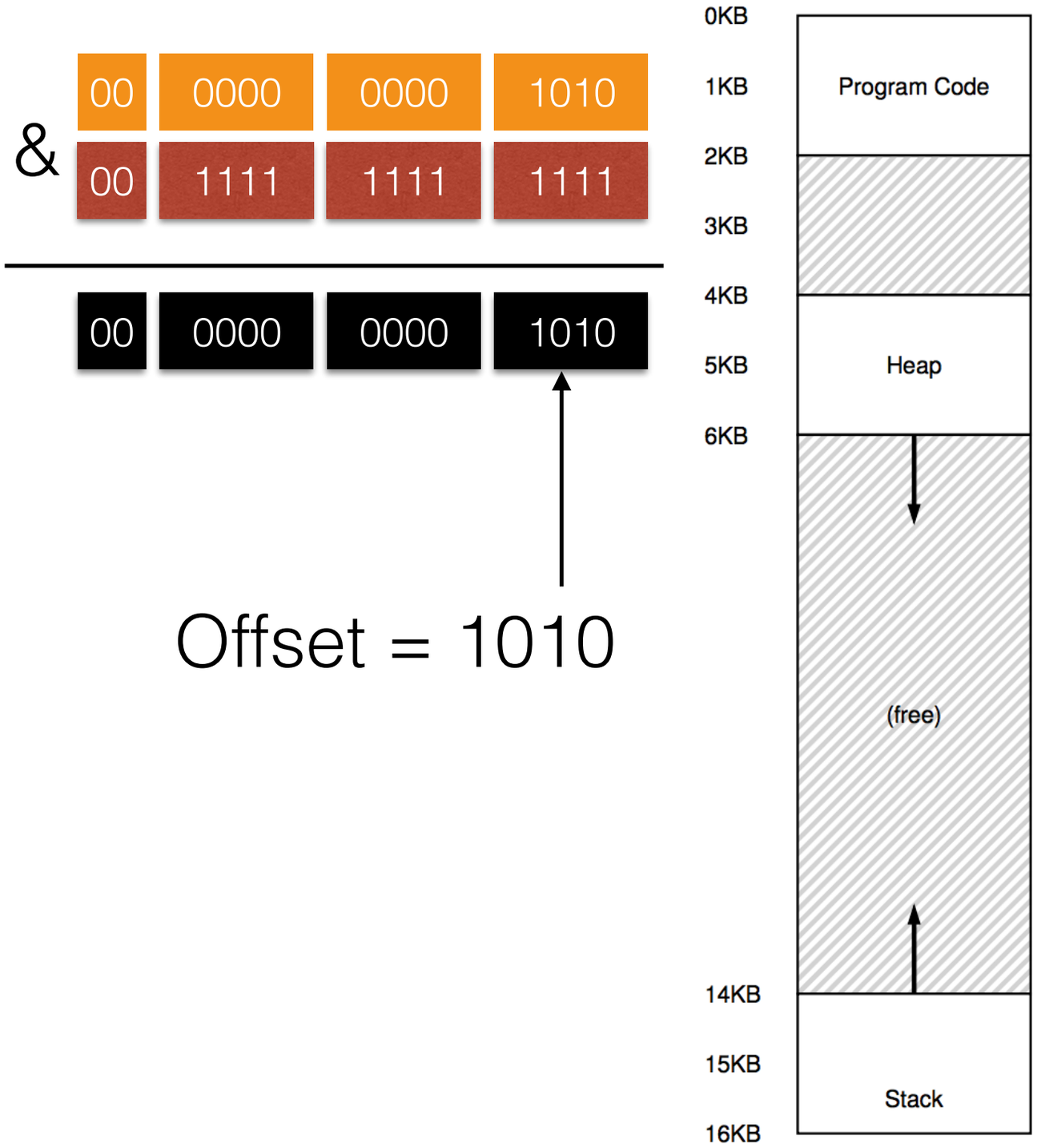
01 0111 1111 1111

Offset = VA & OFF_Mask

11 1000 0000 0000

11 1111 1111 1111

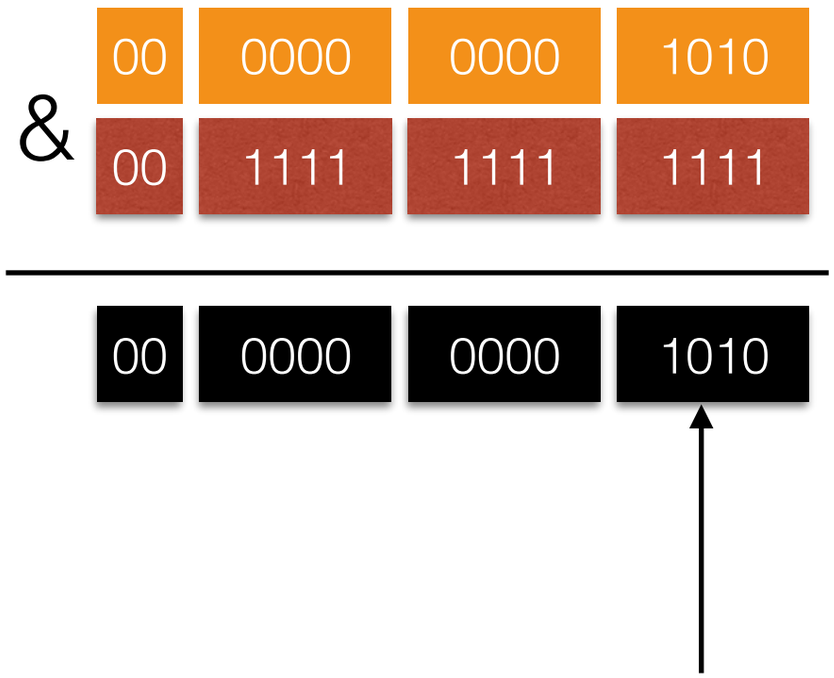
Segment Reference



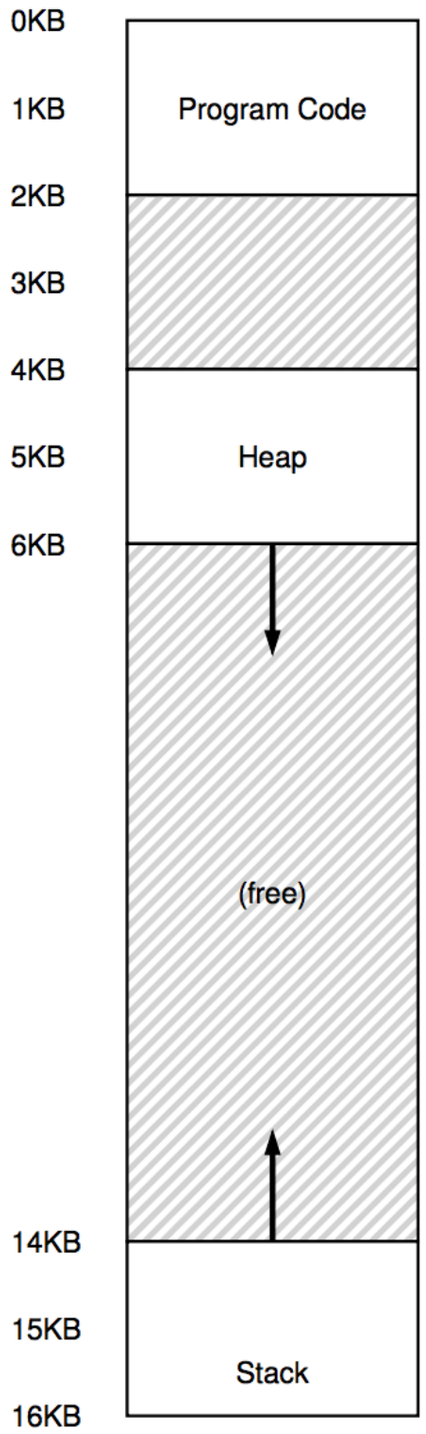
11 1000 0000 0000

11 1111 1111 1111

Segment Reference



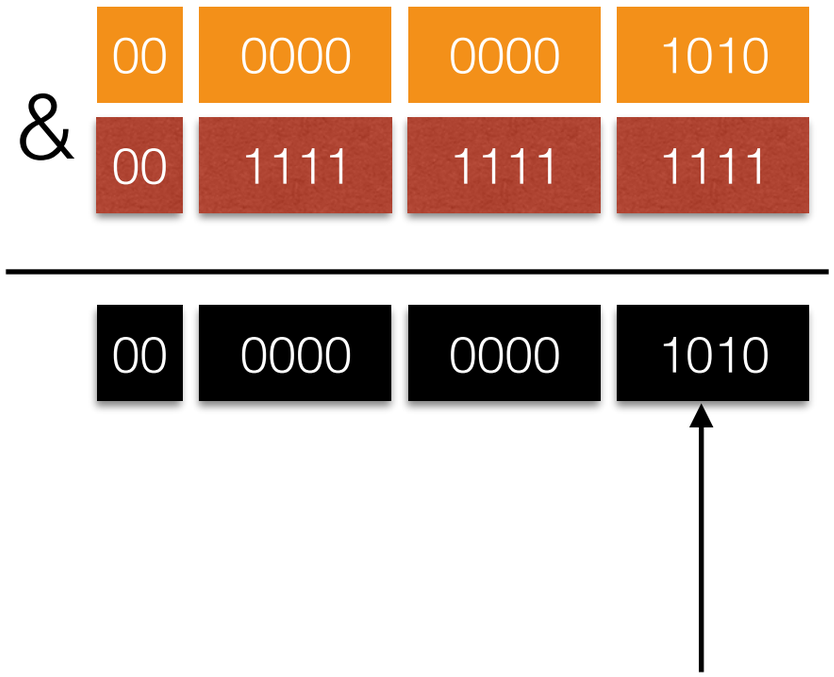
Offset = 1010



Segment Register

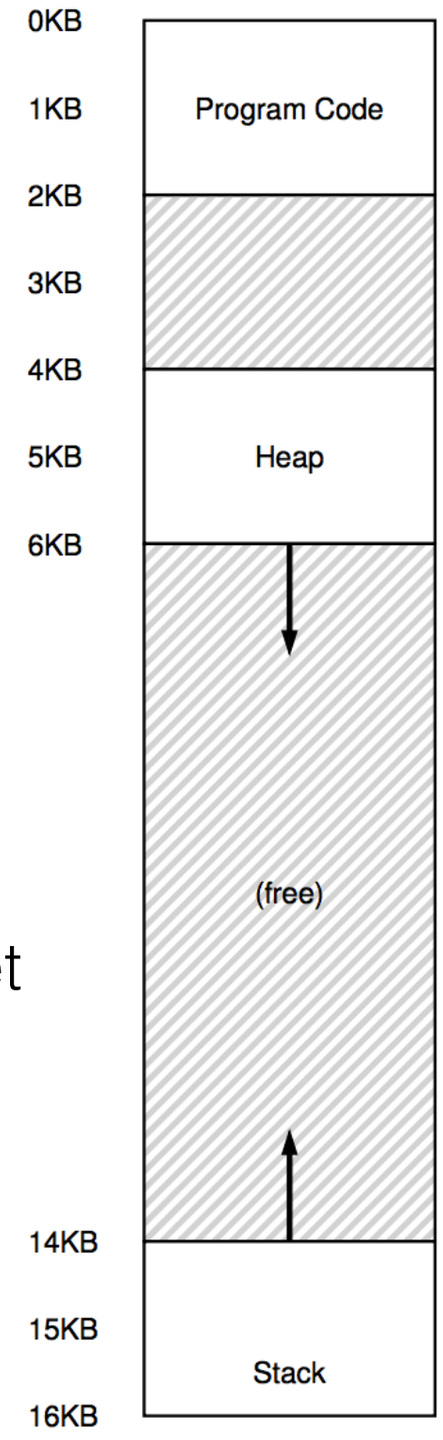
Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

Segment Reference



Offset = 1010

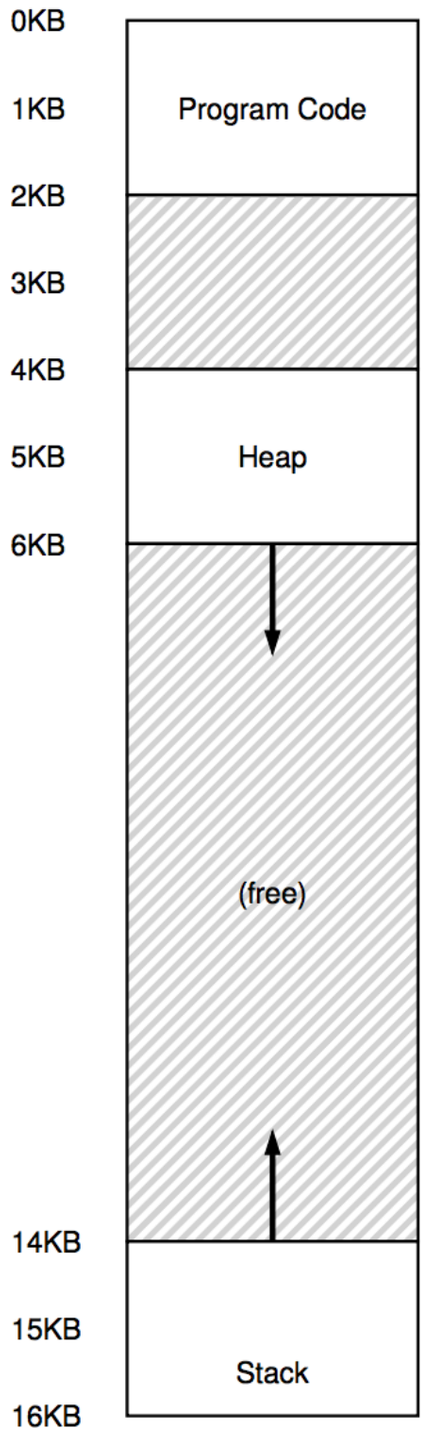
PA = Base[Segment] + Offset
= 32K + 1010b
= (32768 + 10)b



Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

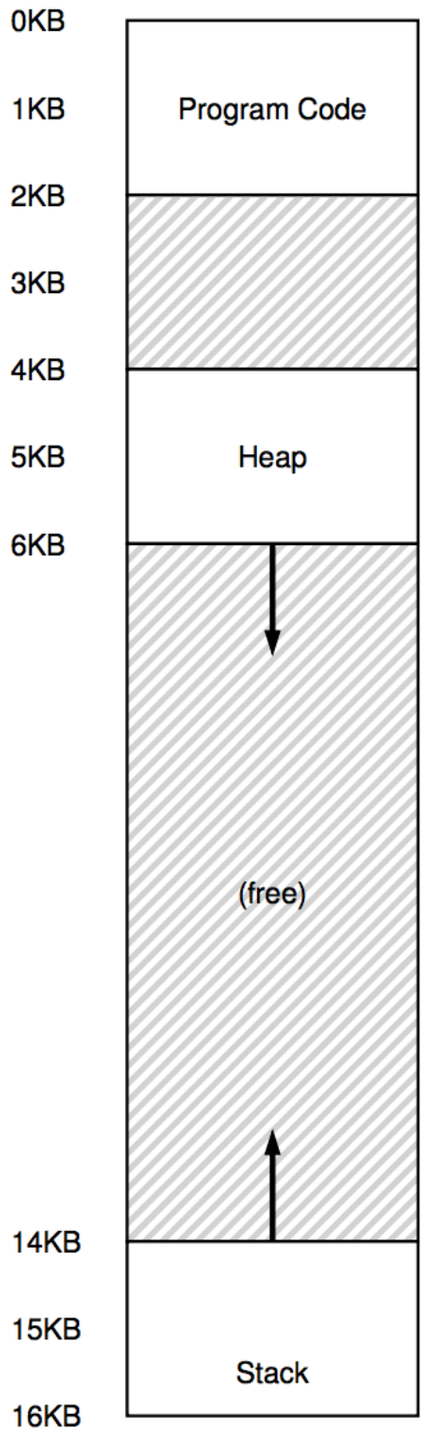
Segment Reference



Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

Segment Reference



Segment Register

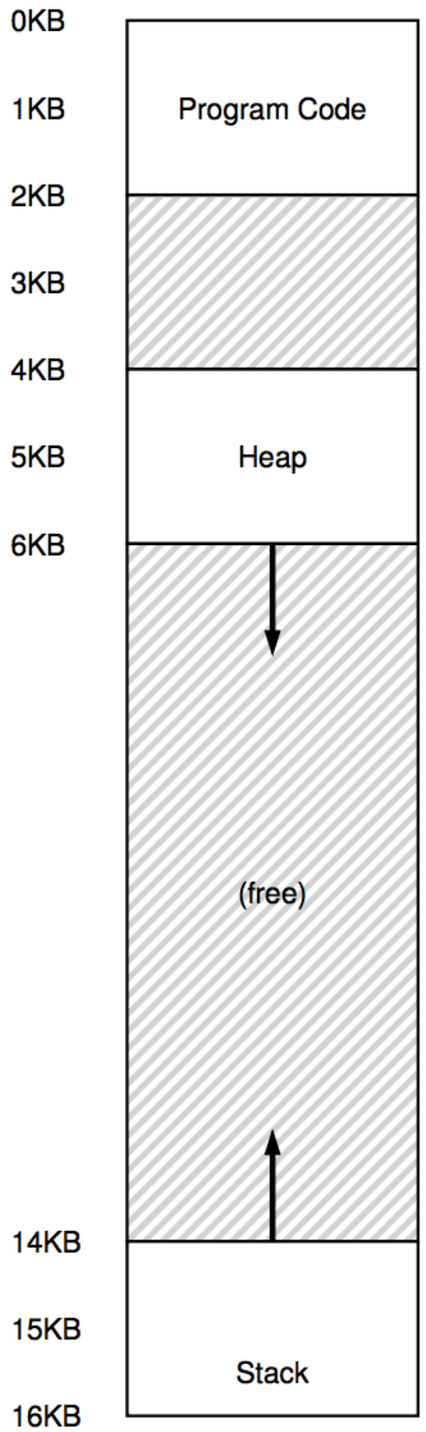
Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

11 1100 0000 0000

11 1000 0000 0000

11 1111 1111 1111

Segment Reference



Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

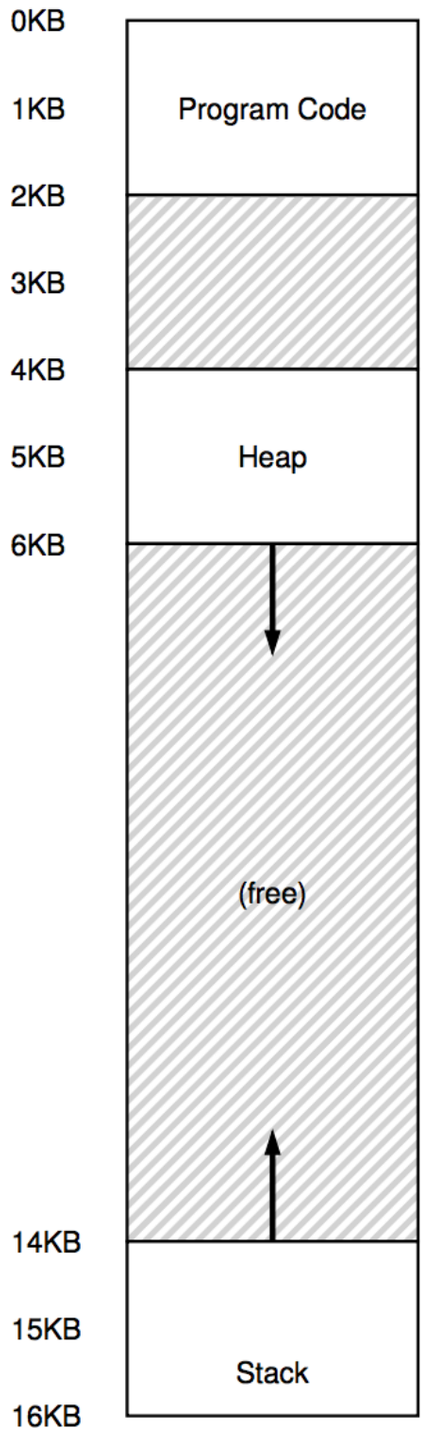
15K VA -> 27K PA

11	1100	0000	0000

11	1000	0000	0000
11	1111	1111	1111

Segment Reference

PA = Base[Segment] + Offset



Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

15K VA -> 27K PA

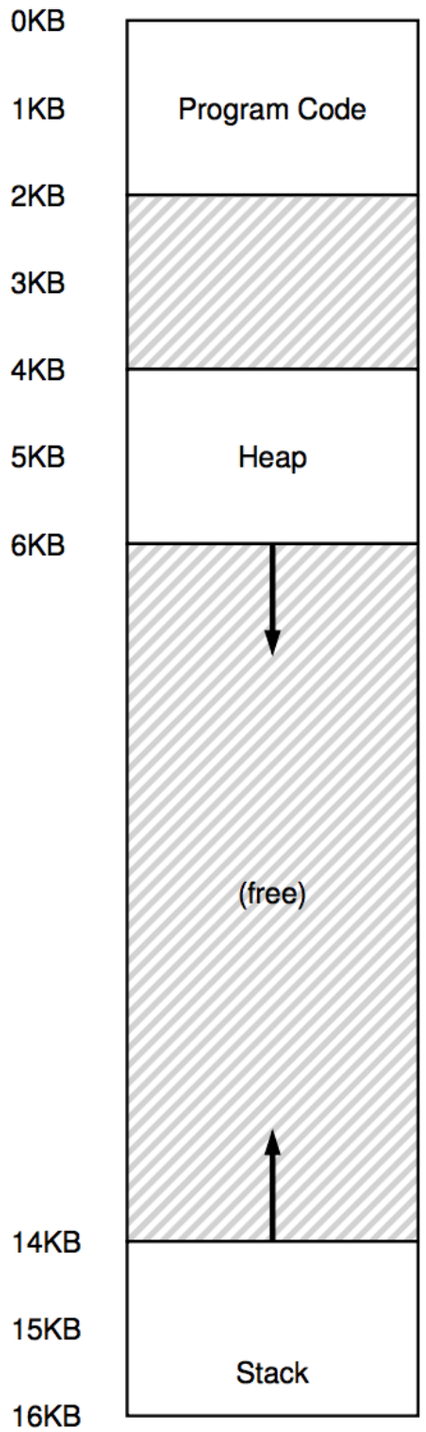
11	1100	0000	0000

11	1000	0000	0000
11	1111	1111	1111

Segment Reference

PA = Base[Segment] + Offset

= 28K + 1100 0000 0000b



Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

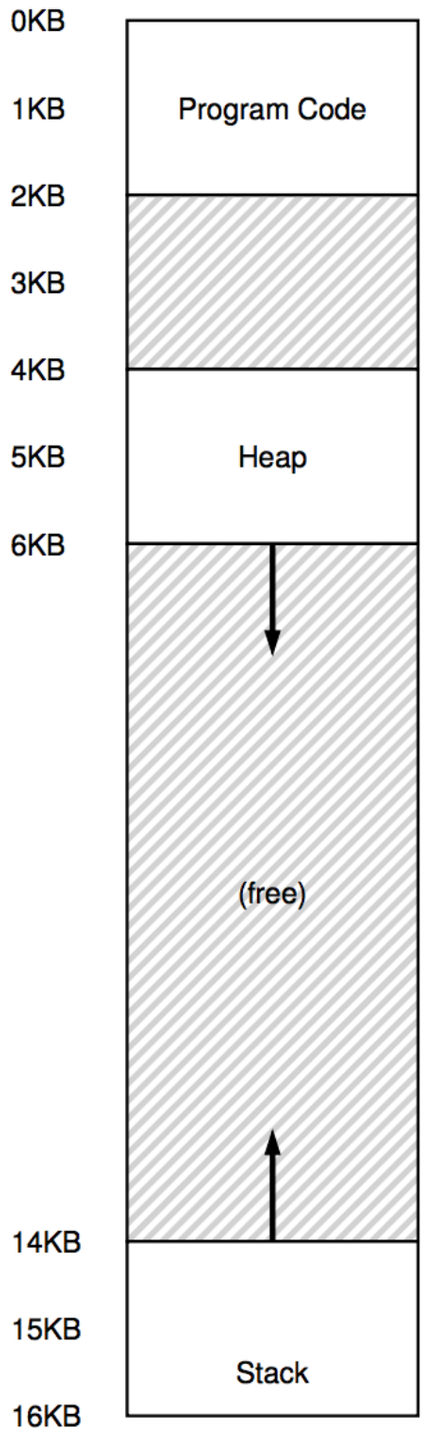
15K VA -> 27K PA

11	1100	0000	0000

11	1000	0000	0000
11	1111	1111	1111

Segment Reference

PA = Base[Segment] + Offset
= 28K + 1100 0000 0000b
= 28K + 3 K = 31K



Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

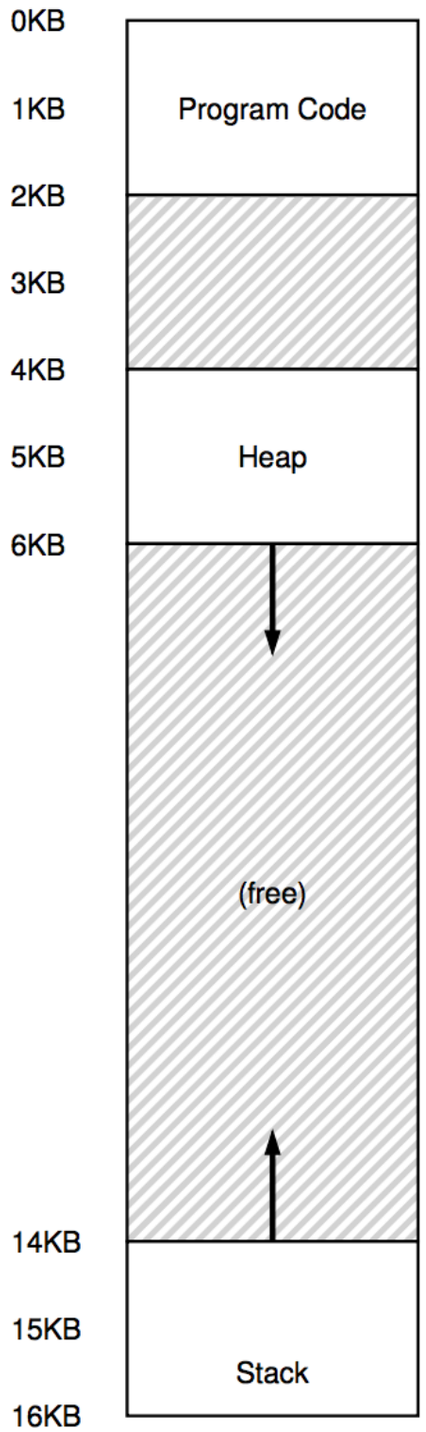
15K VA -> 27K PA

11	1100	0000	0000

11	1000	0000	0000
11	1111	1111	1111

Segment Reference

PA = Base[Segment] + Offset
= 28K + 1100 0000 0000b
= 28K + 3 K = 31K
!=27K



Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

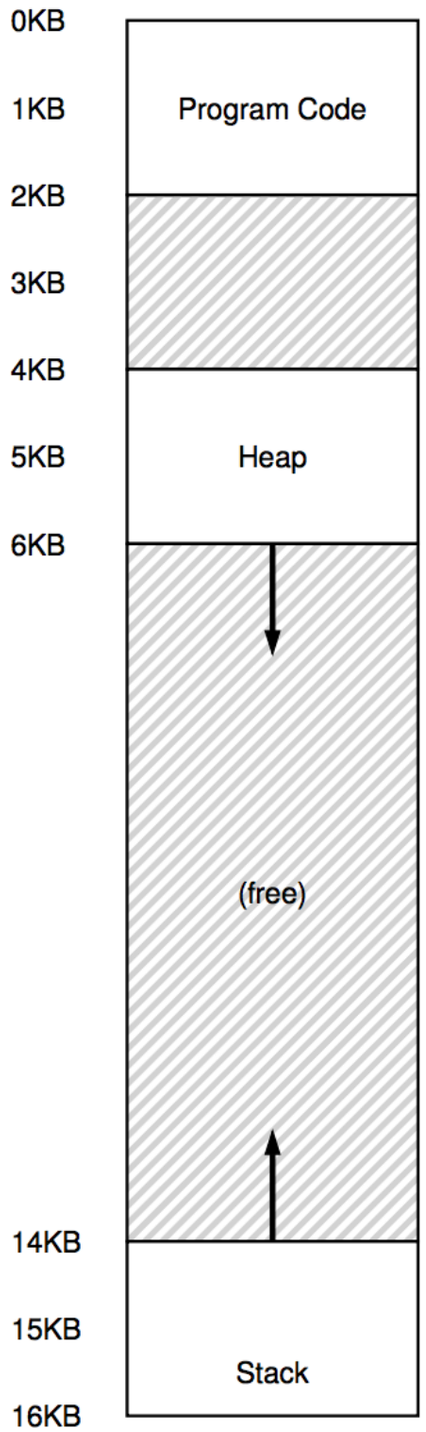
15K VA -> 27K PA

11	1100	0000	0000

11	1000	0000	0000
11	1111	1111	1111

Segment Reference

PA = Base[Segment] + Offset
= 28K + 1100 0000 0000b
= 28K + 3 K = 31K
!=27K



Segment Register

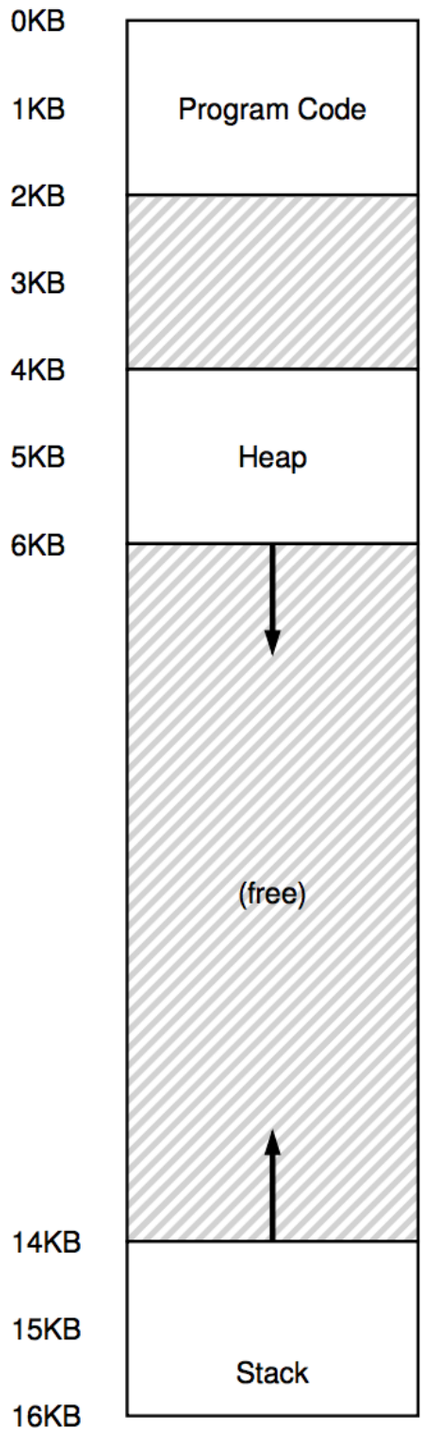
Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

15K VA -> 27K PA

11	1100	0000	0000

11	1000	0000	0000
11	1111	1111	1111

Segment Reference



Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

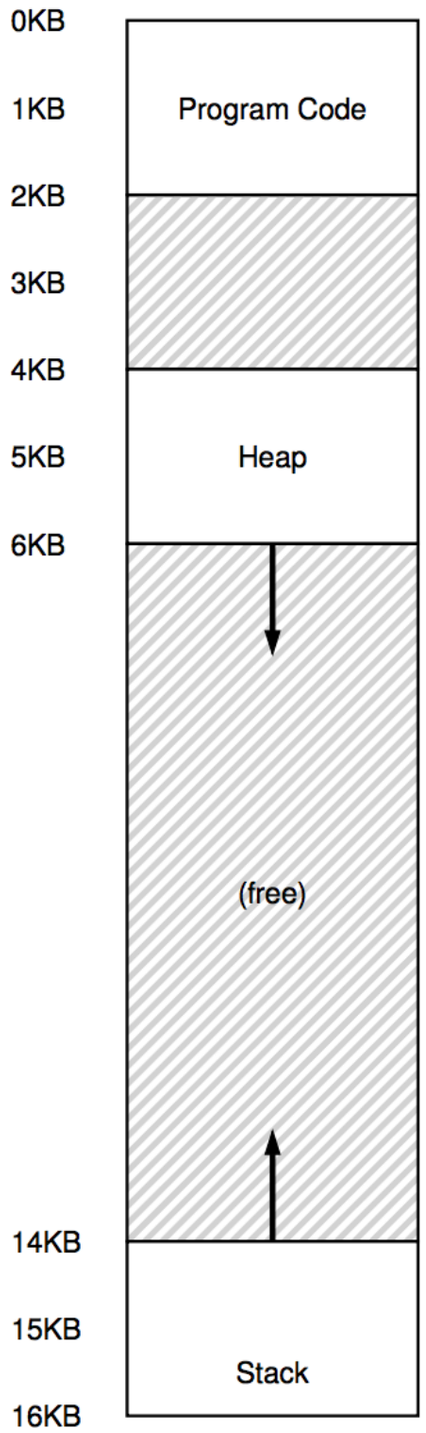
15K VA -> 27K PA

11	1100	0000	0000

11	1000	0000	0000
11	1111	1111	1111

Segment Reference

Take 2



Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

15K VA -> 27K PA

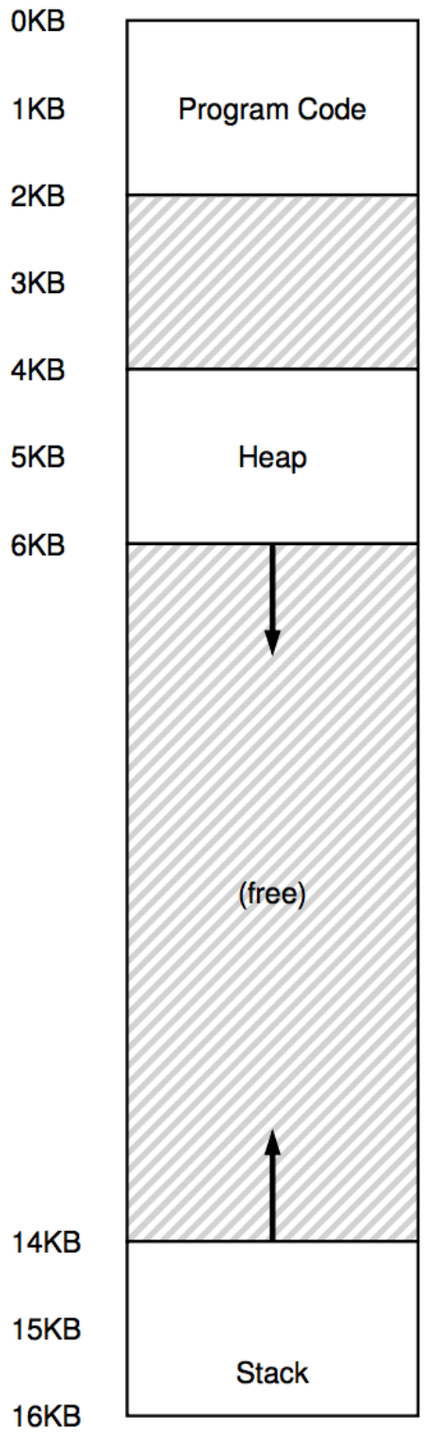
11	1100	0000	0000

11	1000	0000	0000
11	1111	1111	1111

Segment Reference

Take 2

PA = Base[Segment] - Offset



Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

15K VA -> 27K PA

11	1100	0000	0000

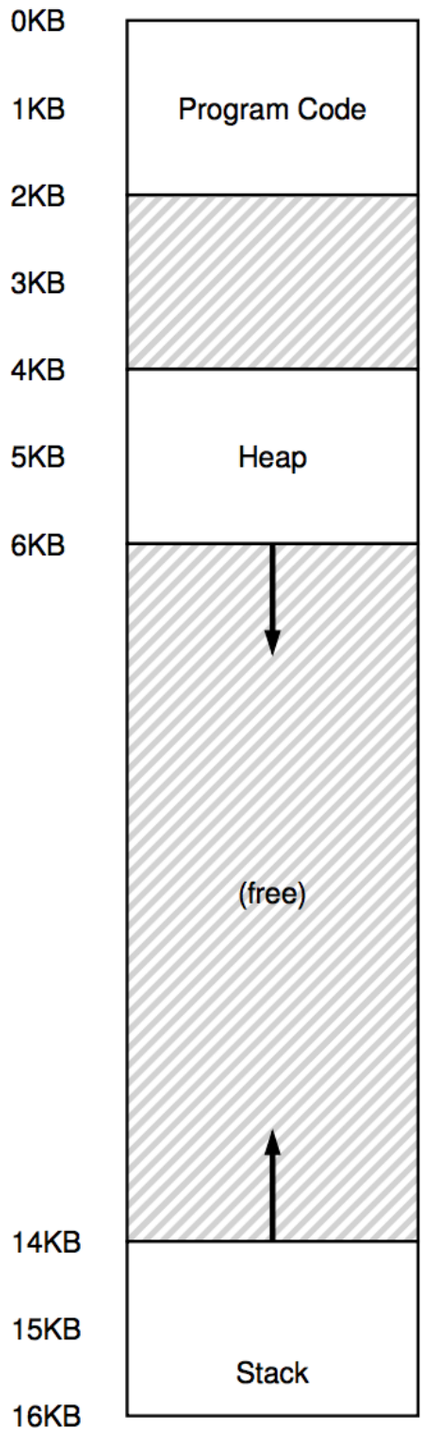
11	1000	0000	0000
11	1111	1111	1111

Segment Reference

Take 2

PA = Base[Segment] - Offset

= 28K - 1100 0000 0000b



Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

15K VA -> 27K PA

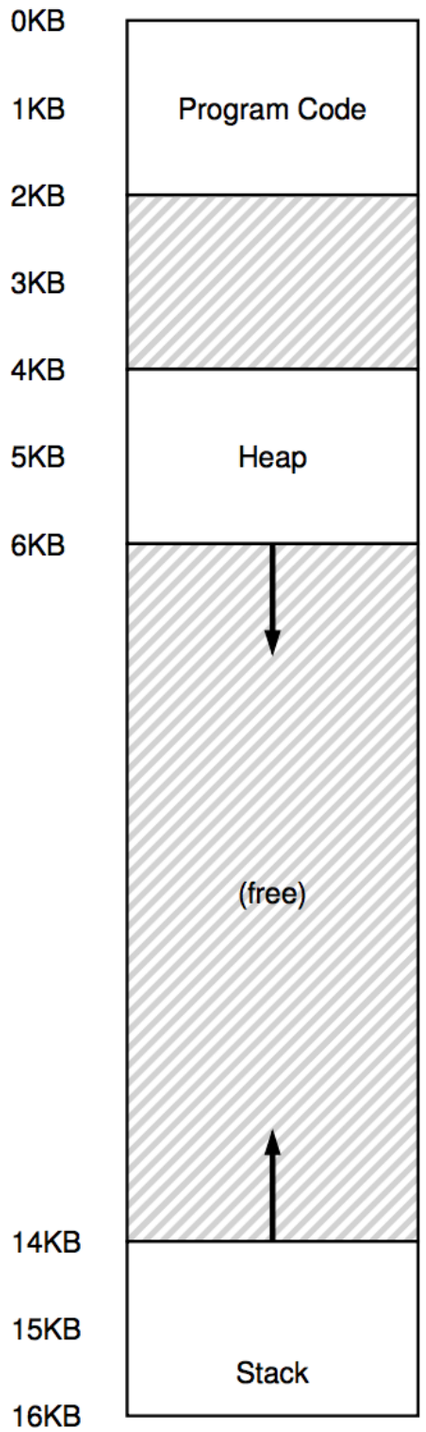
11	1100	0000	0000

11	1000	0000	0000
11	1111	1111	1111

Segment Reference

Take 2

PA = Base[Segment] - Offset
= 28K - 1100 0000 0000b
= 28K - 3 K = 25 K



Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

15K VA -> 27K PA

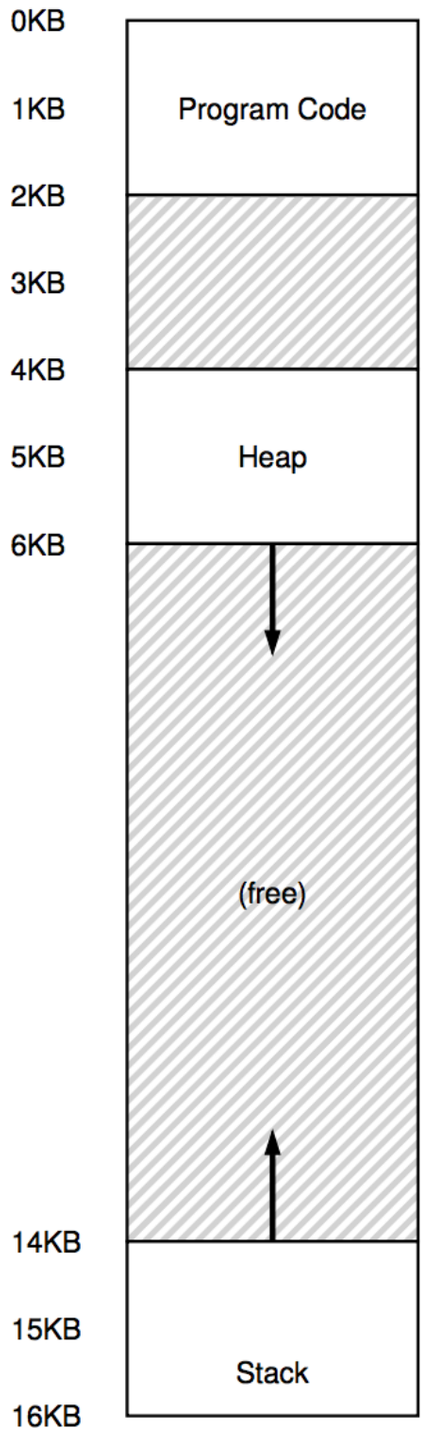
11	1100	0000	0000

11	1000	0000	0000
11	1111	1111	1111

Segment Reference

Take 2

PA = Base[Segment] - Offset
= 28K - 1100 0000 0000b
= 28K - 3 K = 25 K
!=27K



Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

15K VA -> 27K PA

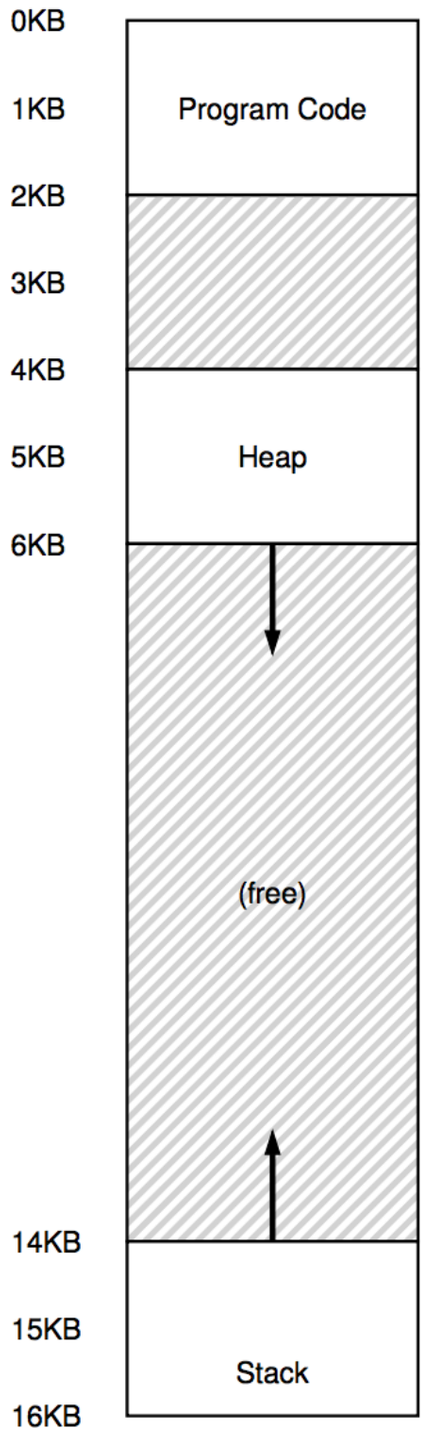
11	1100	0000	0000

11	1000	0000	0000
11	1111	1111	1111

Segment Reference

Take 2

PA = Base[Segment] - Offset
= 28K - 1100 0000 0000b
= 28K - 3 K = 25 K
!=27K



Segment Register

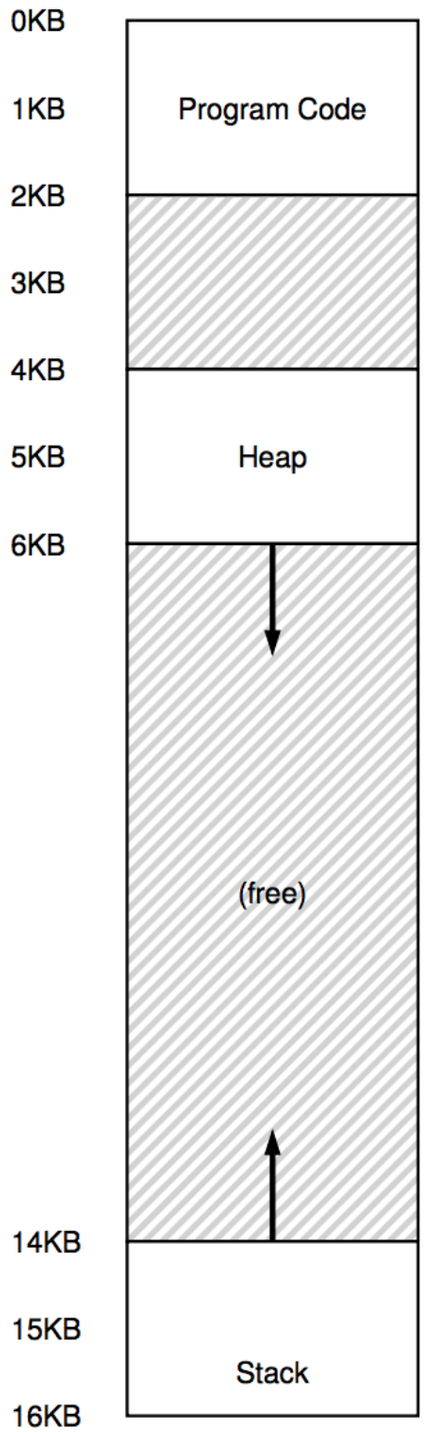
Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

15K VA -> 27K PA

11	1100	0000	0000

11	1000	0000	0000
11	1111	1111	1111

Segment Reference



Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

15K VA -> 27K PA

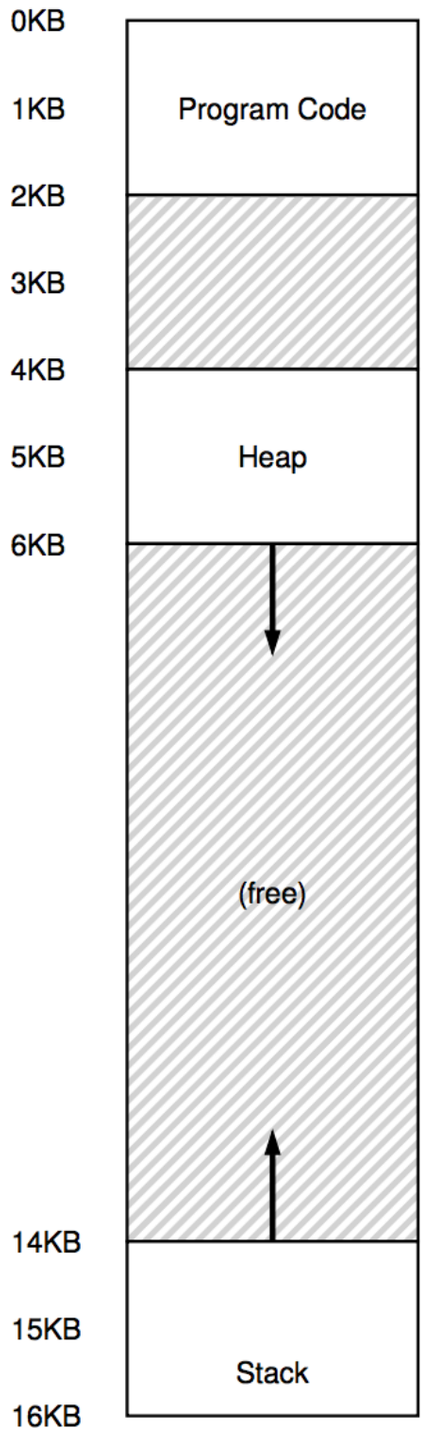
1100 0000 0000

11 1000 0000 0000

11 1111 1111 1111

Segment Reference

Take 3



Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

15K VA -> 27K PA

1100 0000 0000

11 1000 0000 0000

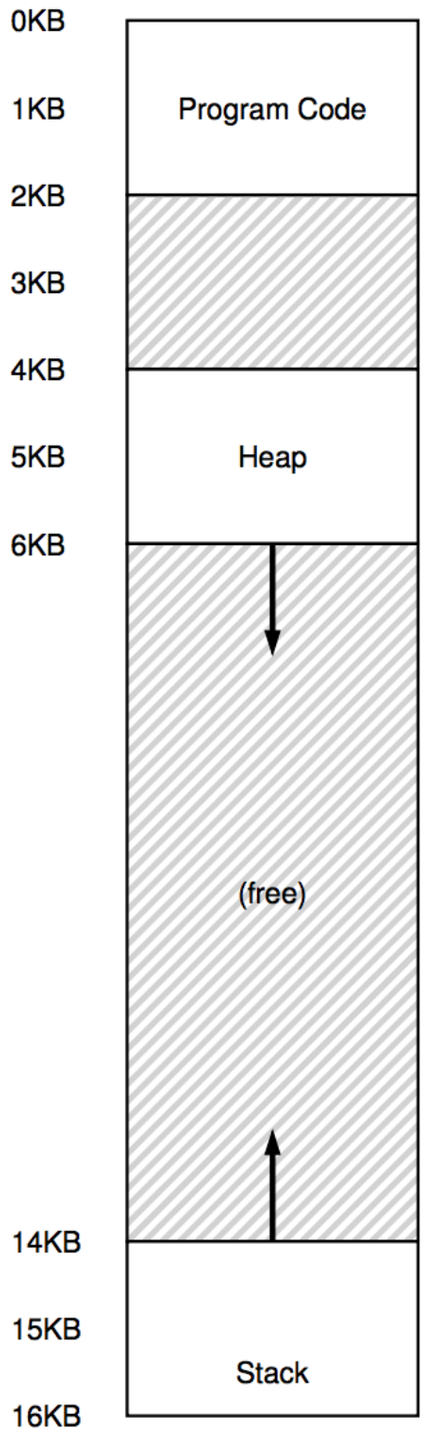
11 1111 1111 1111

Segment Reference

Take 3

PA = Base[Segment] + Offset - Offset (for first address)

15K VA -> 27K PA



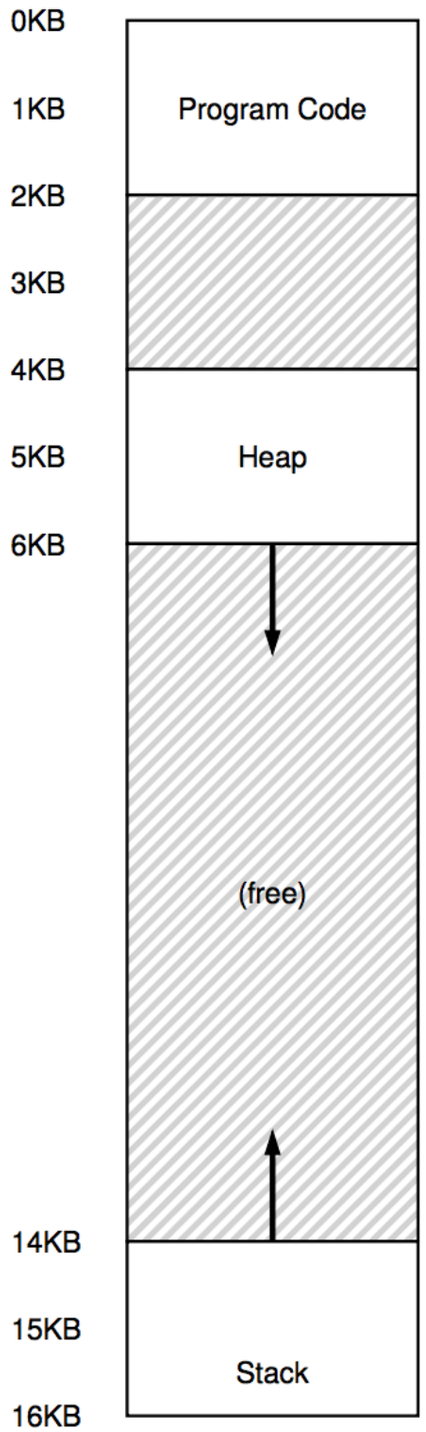
Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

Segment Reference

Take 3

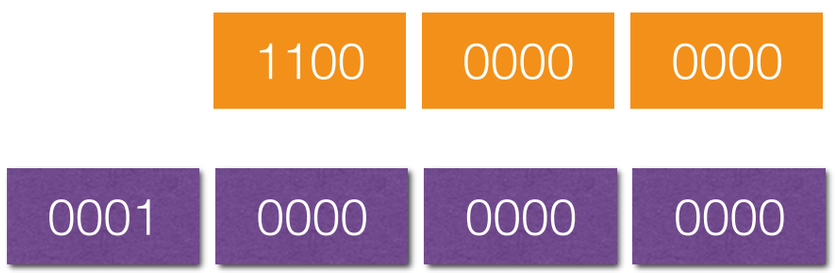
PA = Base[Segment] + Offset - Offset (for first address)



Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

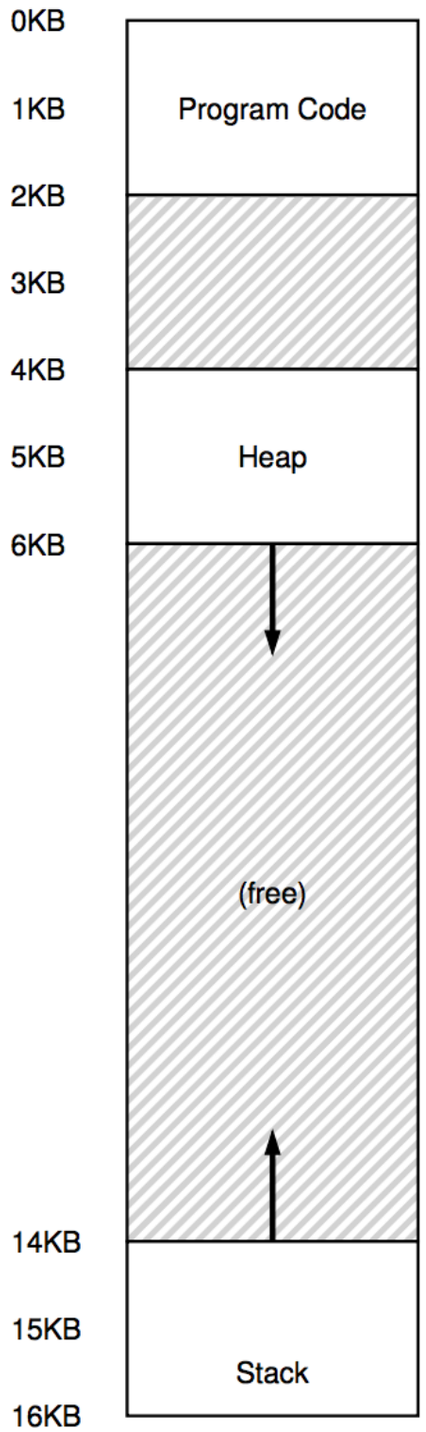
15K VA -> 27K PA



Segment Reference

Take 3

PA = Base[Segment] + Offset - Offset (for first address)



Segment Register

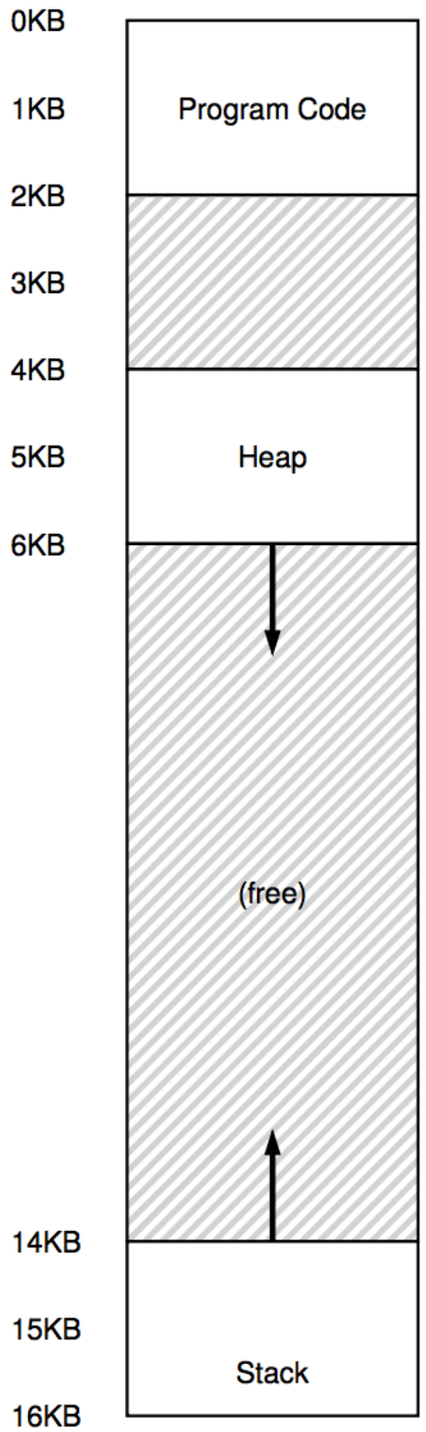
Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

15K VA -> 27K PA

Segment Reference

Take 3

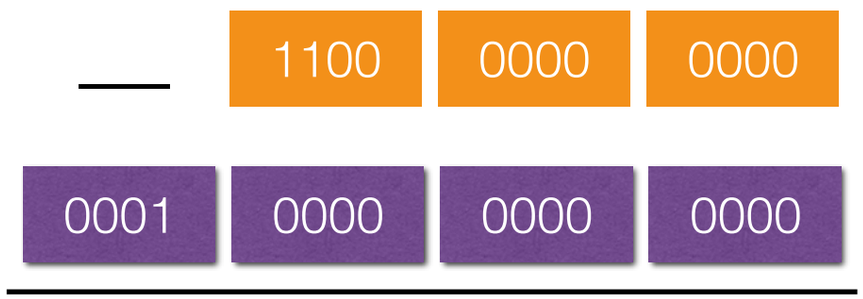
PA = Base[Segment] + Offset - Offset (for first address)



Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

15K VA -> 27K PA



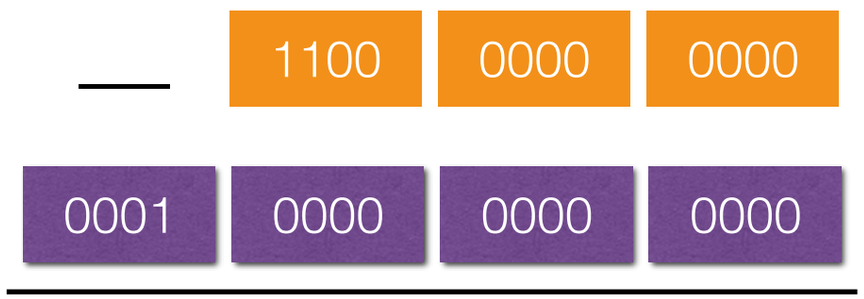
Segment Reference

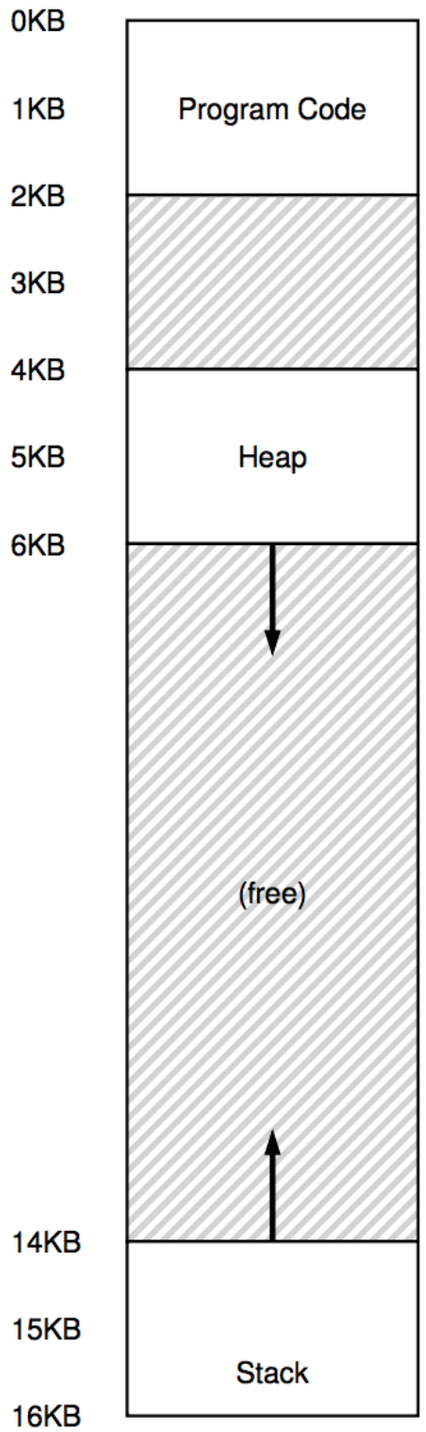
Take 3

PA = Base[Segment] + Offset - Offset (for first address)

= 28K + (-01 0000 0000)b

15K VA -> 27K PA





Segment Register

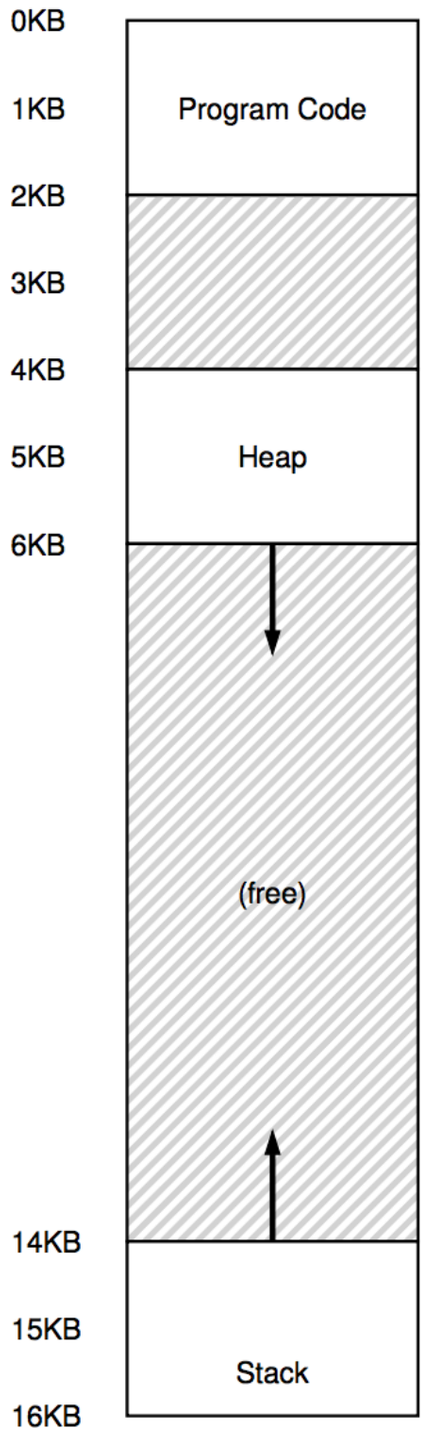
Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

Segment Reference

Take 3

PA = Base[Segment] + Offset - Offset (for first address)

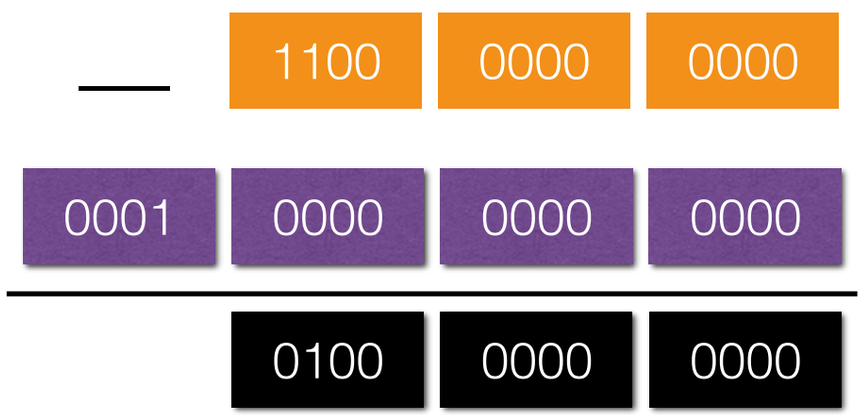
= 28K + (-01 0000 0000)b



Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

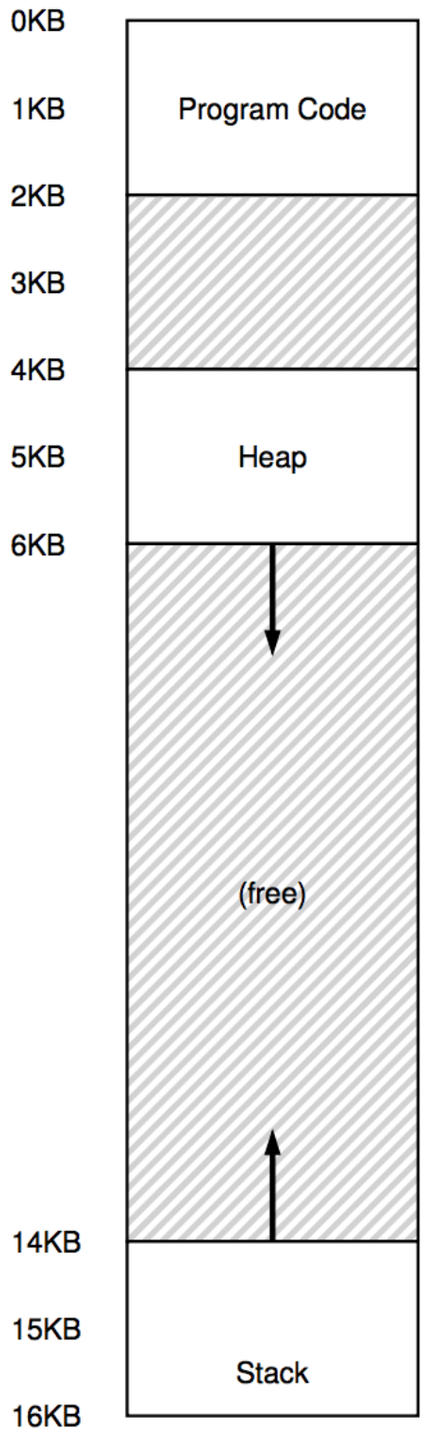
15K VA -> 27K PA



Segment Reference

Take 3

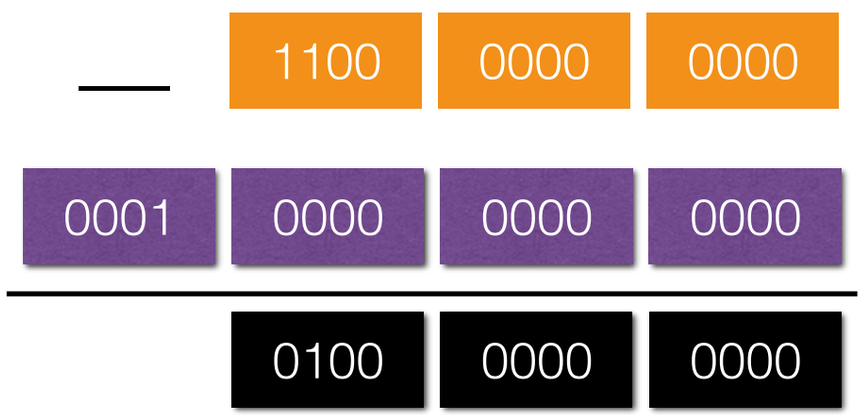
PA = Base[Segment] + Offset - Offset (for first address)
= 28K + (-01 0000 0000)b
= 28K - 1 K = 27 K



Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

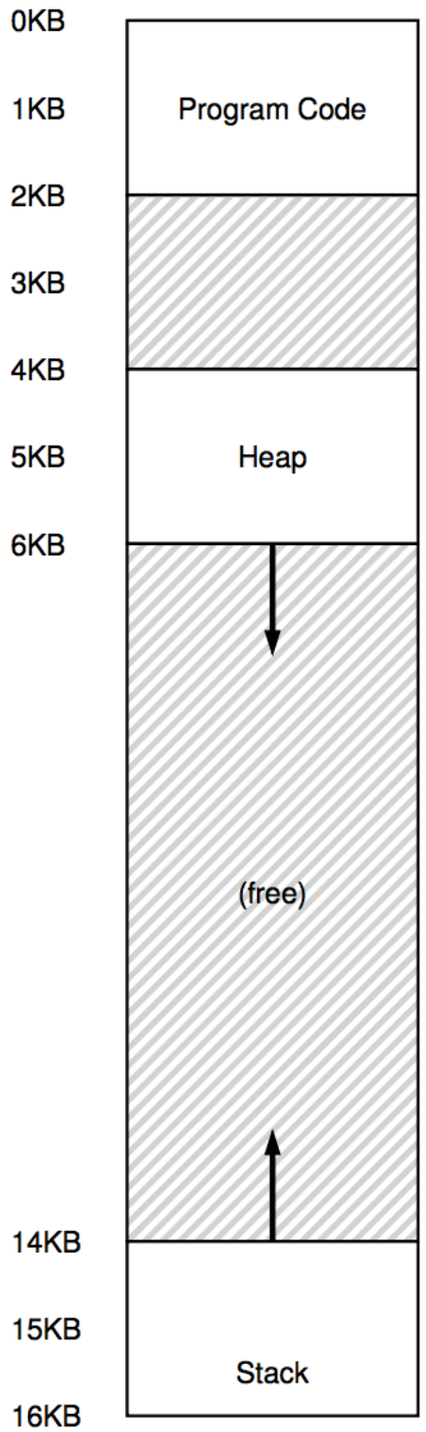
15K VA -> 27K PA



Segment Reference

Take 3

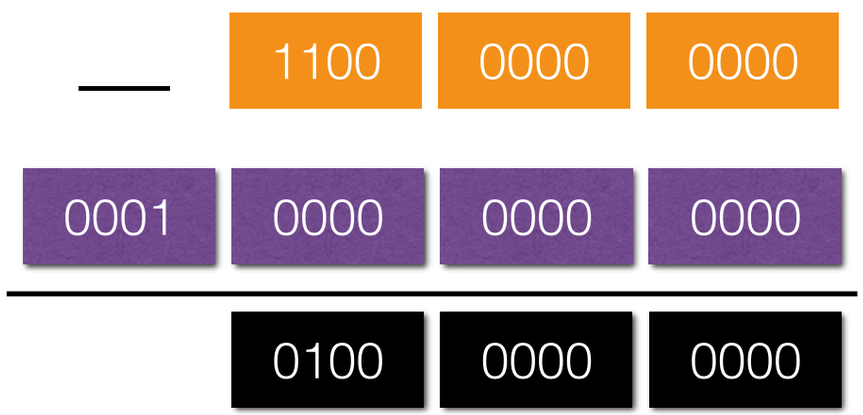
PA = Base[Segment] + Offset - Offset (for first address)
= 28K + (-01 0000 0000)b
= 28K - 1 K = 27 K



Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

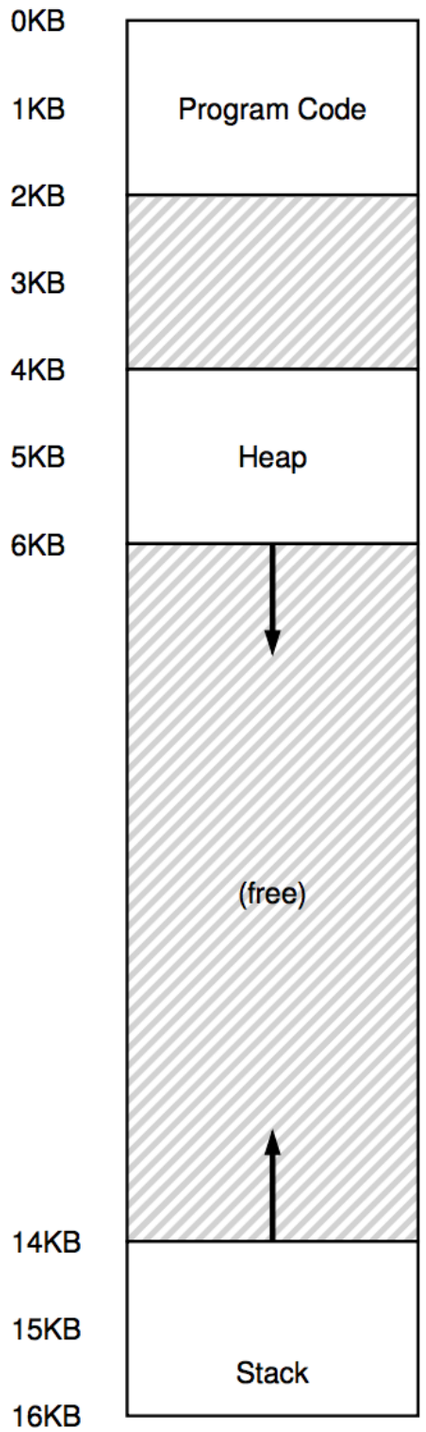
15K VA -> 27K PA



Segment Reference

Take 3

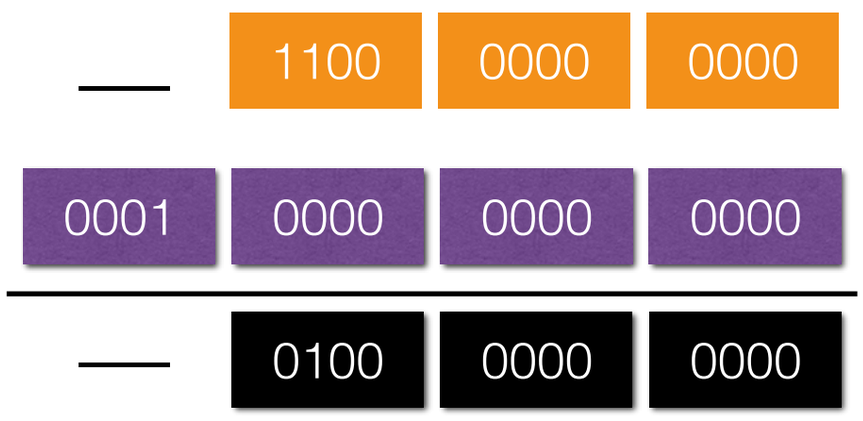
PA = Base[Segment] + Offset - Offset (for first address)
= 28K + (-01 0000 0000)b
= 28K - 1 K = 27 K



Segment Register

Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

15K VA -> 27K PA



Segment Registers with Negative Growth

Segment	Base	Size	Grows Positive?
Code	32K	2K	1
Heap	34K	2K	1
Stack	28K	2K	0

Sharing Support

Segment	Base	Size	Grows Positive?	Protection
Code	32K	2K	1	Read-Execute
Heap	34K	2K	1	Read-Write
Stack	28K	2K	0	Read-Write

Sharing Support

Why would you share code across processes?

Segment	Base	Size	Grows Positive?	Protection
Code	32K	2K	1	Read-Execute
Heap	34K	2K	1	Read-Write
Stack	28K	2K	0	Read-Write

Sharing Support

Why would you share code across processes?

Segment	Base	Size	Grows Positive?	Protection
Code	32K	2K	1	Read-Execute
Heap	34K	2K	1	Read-Write
Stack	28K	2K	0	Read-Write

Sharing Support

Why would you share code across processes?

- Exact copy of process?

Segment	Base	Size	Grows Positive?	Protection
Code	32K	2K	1	Read-Execute
Heap	34K	2K	1	Read-Write
Stack	28K	2K	0	Read-Write

Sharing Support

Why would you share code across processes?

- Exact copy of process?
- Same physical address space for 2 virtual code segments?

Segment	Base	Size	Grows Positive?	Protection
Code	32K	2K	1	Read-Execute
Heap	34K	2K	1	Read-Write
Stack	28K	2K	0	Read-Write

OS Support for Segmentation

OS Support for Segmentation

- Context switch?

OS Support for Segmentation

- Context switch?
  - Store and restore segment registers

OS Support for Segmentation

- Context switch?
  - Store and restore segment registers
- Manage free space

OS Support for Segmentation

- Context switch?
  - Store and restore segment registers
- Manage free space
  - Allocate and de-allocate upon creation/ termination

OS Support for Segmentation

- Context switch?
  - Store and restore segment registers
- Manage free space
  - Allocate and de-allocate upon creation/ termination
- Compaction/Defragmentation

Segmentation Pros

Segmentation Pros

1. Still fairly simple:

Segmentation Pros

1. Still fairly simple:
    1. Protection (Segment Exists): N comparisons for N segments.

Segmentation Pros

1. Still fairly simple:
    1. Protection (Segment Exists): N comparisons for N segments.
    2. Translation: one addition. (Once segment located.)

Segmentation Pros

1. Still fairly simple:
    1. Protection (Segment Exists): N comparisons for N segments.
    2. Translation: one addition. (Once segment located.)
2. Can organize and protect regions of memory appropriately.

Segmentation Pros

1. Still fairly simple:
    1. Protection (Segment Exists): N comparisons for N segments.
    2. Translation: one addition. (Once segment located.)
2. Can organize and protect regions of memory appropriately.
3. Better fit for address spaces leading to less internal fragmentation

Segmentation Cons

Segmentation Cons

1. Still requires entire segment be contiguous in memory!

Segmentation Cons

1. Still requires entire segment be contiguous in memory!
2. Potential for external fragmentation due to segment contiguity.

Thought Experiment

Thought Experiment

1. Large contiguous memory causes problems

Thought Experiment

1. Large contiguous memory causes problems
    1. What happens if we map every byte of VA to a byte of PA?

Thought Experiment

1. Large contiguous memory causes problems
    1. What happens if we map every byte of VA to a byte of PA?
        1. Reduces fragmentation?

Thought Experiment

1. Large contiguous memory causes problems
    1. What happens if we map every byte of VA to a byte of PA?
        1. Reduces fragmentation?
            1. External?

Thought Experiment

1. Large contiguous memory causes problems
    1. What happens if we map every byte of VA to a byte of PA?
        1. Reduces fragmentation?
            1. External?
            2. Internal?

Thought Experiment

1. Large contiguous memory causes problems
    1. What happens if we map every byte of VA to a byte of PA?
        1. Reduces fragmentation?
            1. External?
            2. Internal?
    2. How much space needed per-process to store mapping?

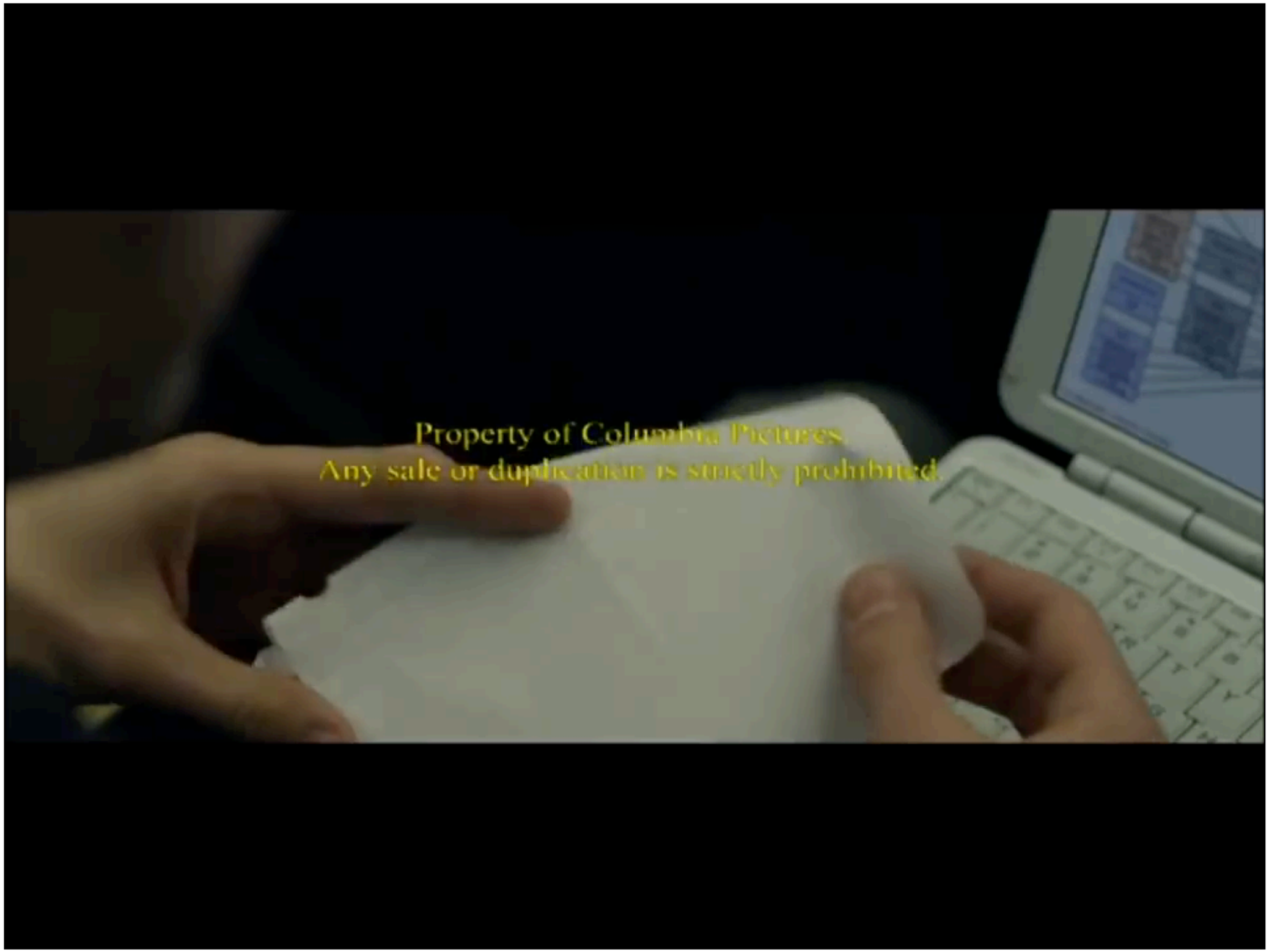
Thought Experiment

1. Large contiguous memory causes problems
    1. What happens if we map every byte of VA to a byte of PA?
        1. Reduces fragmentation?
            1. External?
            2. Internal?
        2. How much space needed per-process to store mapping?
2. Middle ground?

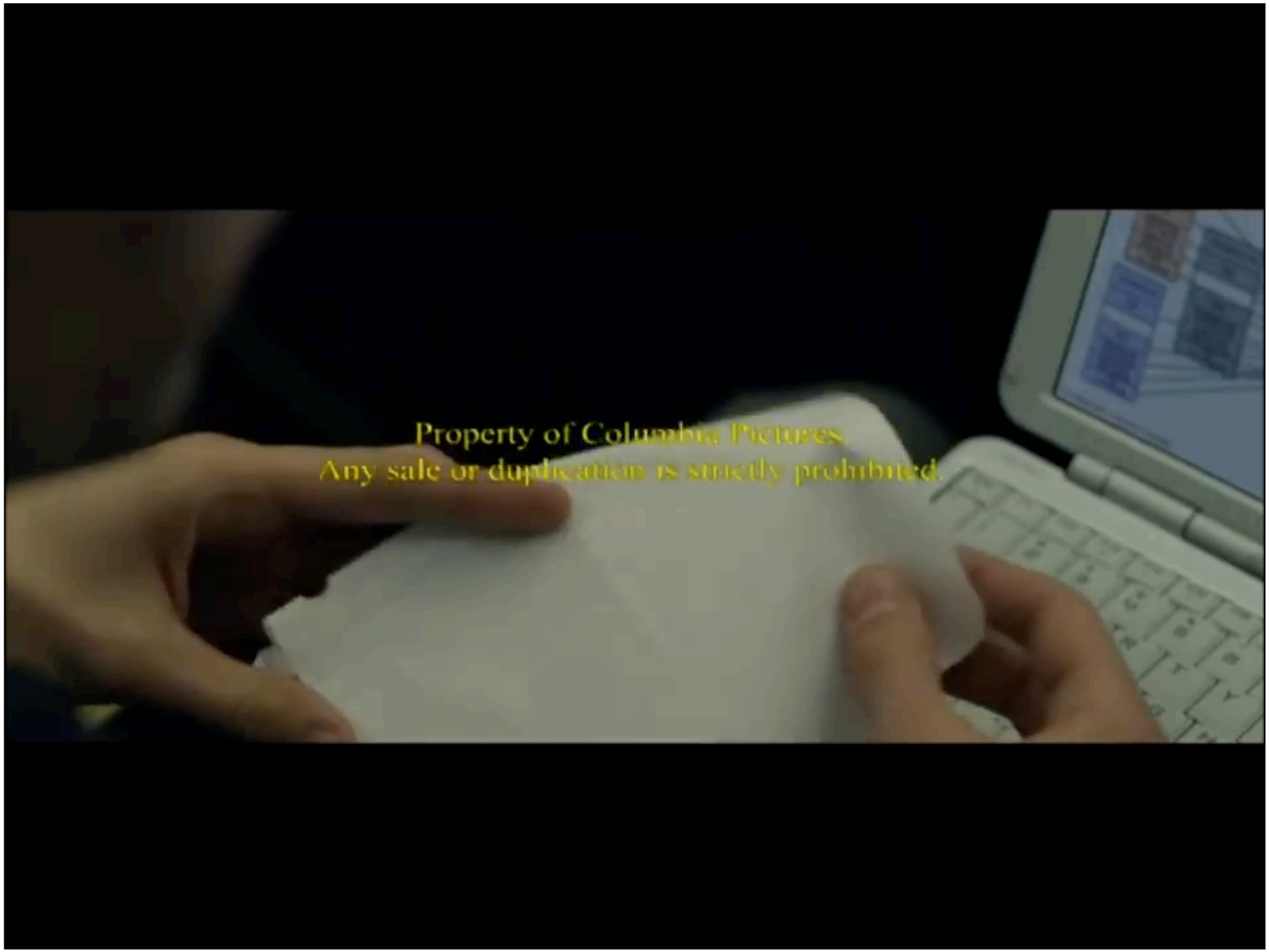
Thought Experiment

1. Large contiguous memory causes problems
    1. What happens if we map every byte of VA to a byte of PA?
        1. Reduces fragmentation?
            1. External?
            2. Internal?
        2. How much space needed per-process to store mapping?
2. Middle ground?

Paging!


Property of Columbia Pictures
Any sale or duplication is strictly prohibited.

Paging!


Property of Columbia Pictures
Any sale or duplication is strictly prohibited.

Paging - Divide em up

Paging - Divide em up

64 bytes address space

Paging - Divide em up

64 bytes address space

Paging - Divide em up

64 bytes address space

- 4 pages

Paging - Divide em up

64 bytes address space

- 4 pages
- 16 bytes/page

Paging - Divide em up

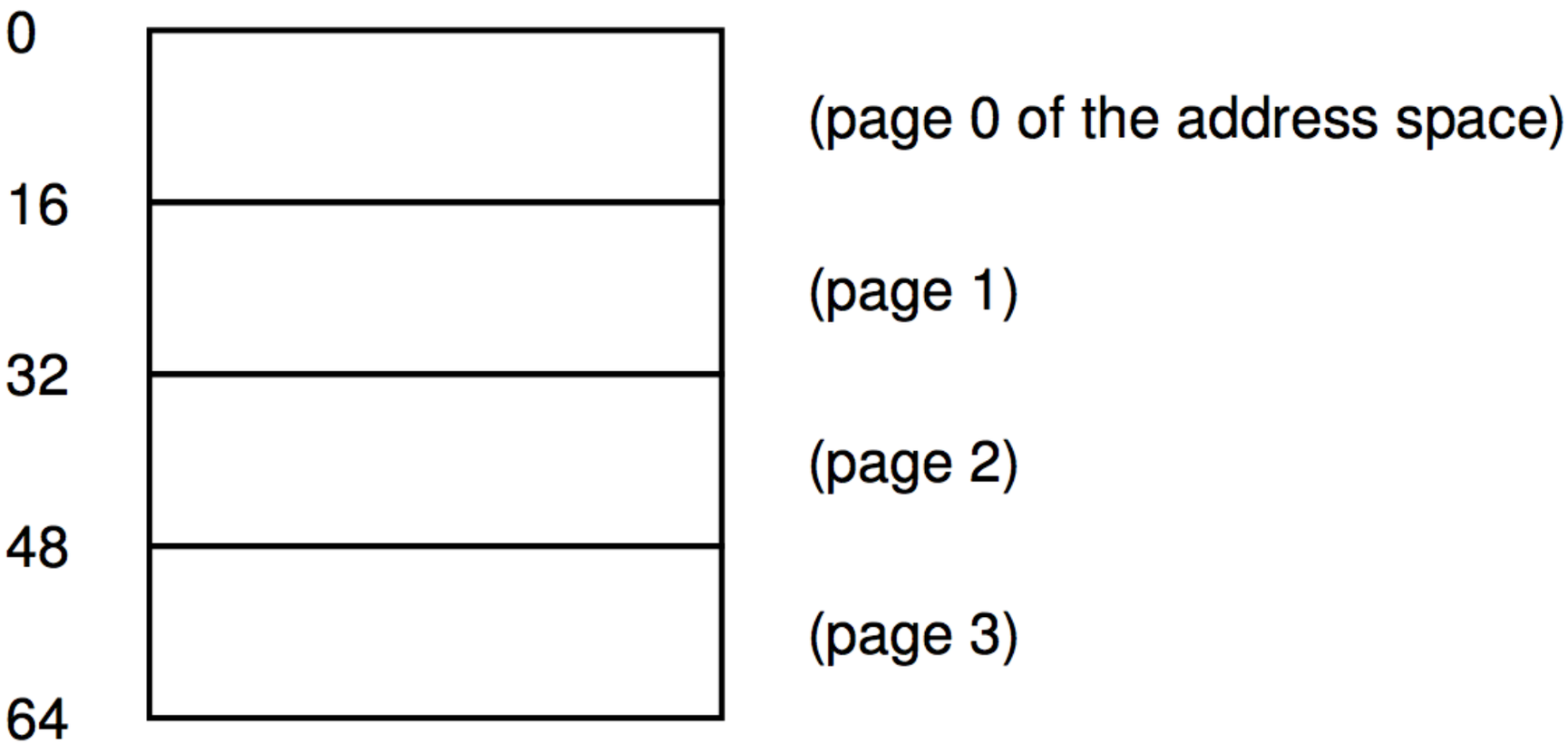
64 bytes address space

- 4 pages
- 16 bytes/page

Paging - Divide em up

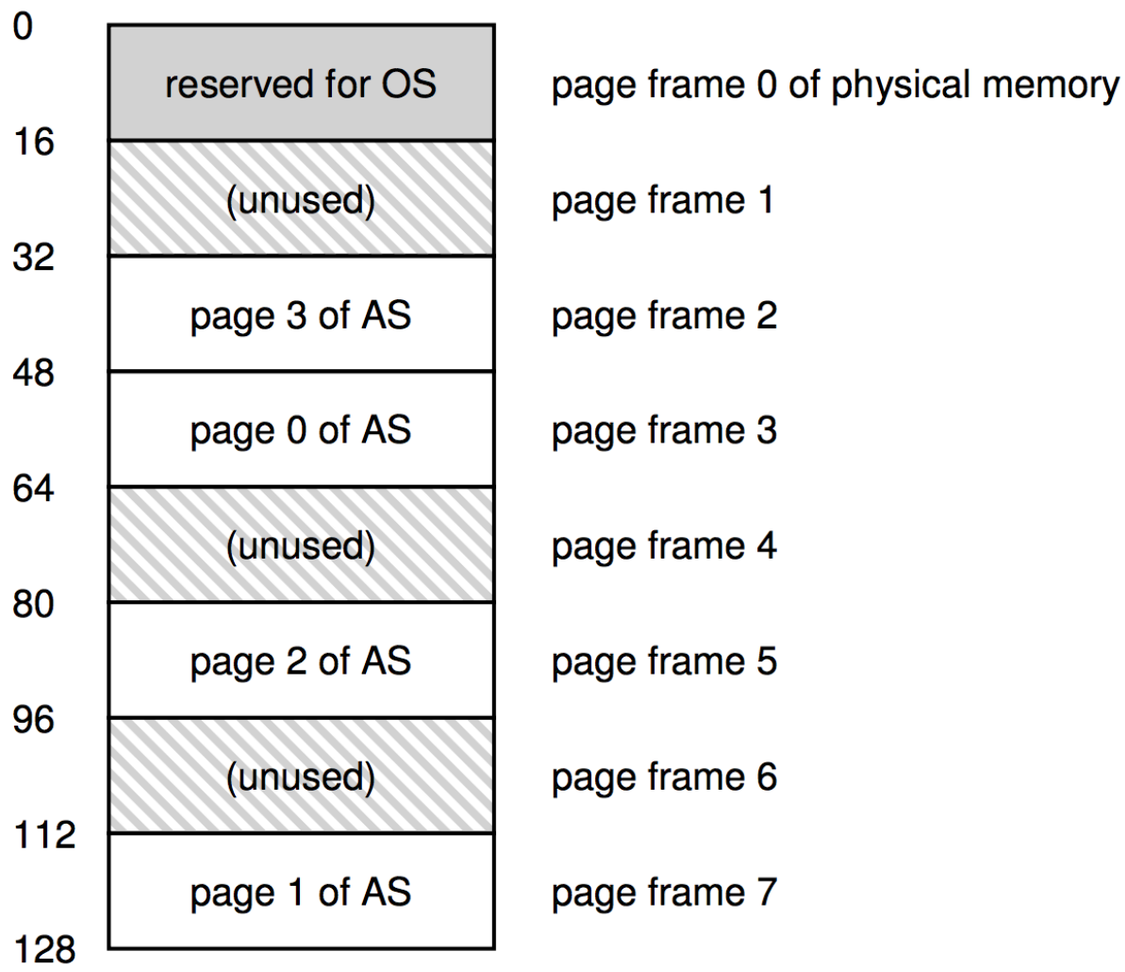
64 bytes address space

- 4 pages
- 16 bytes/page

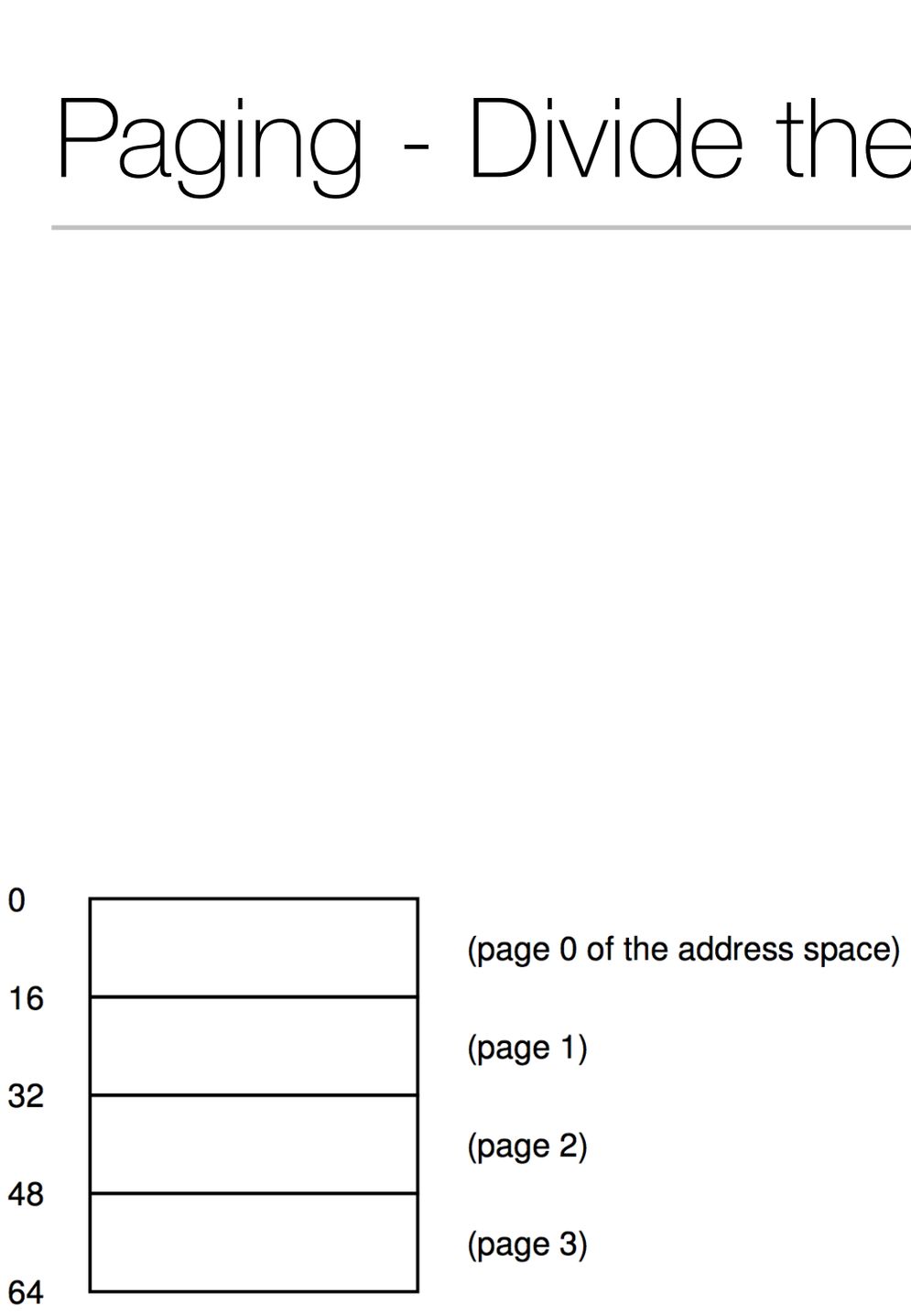


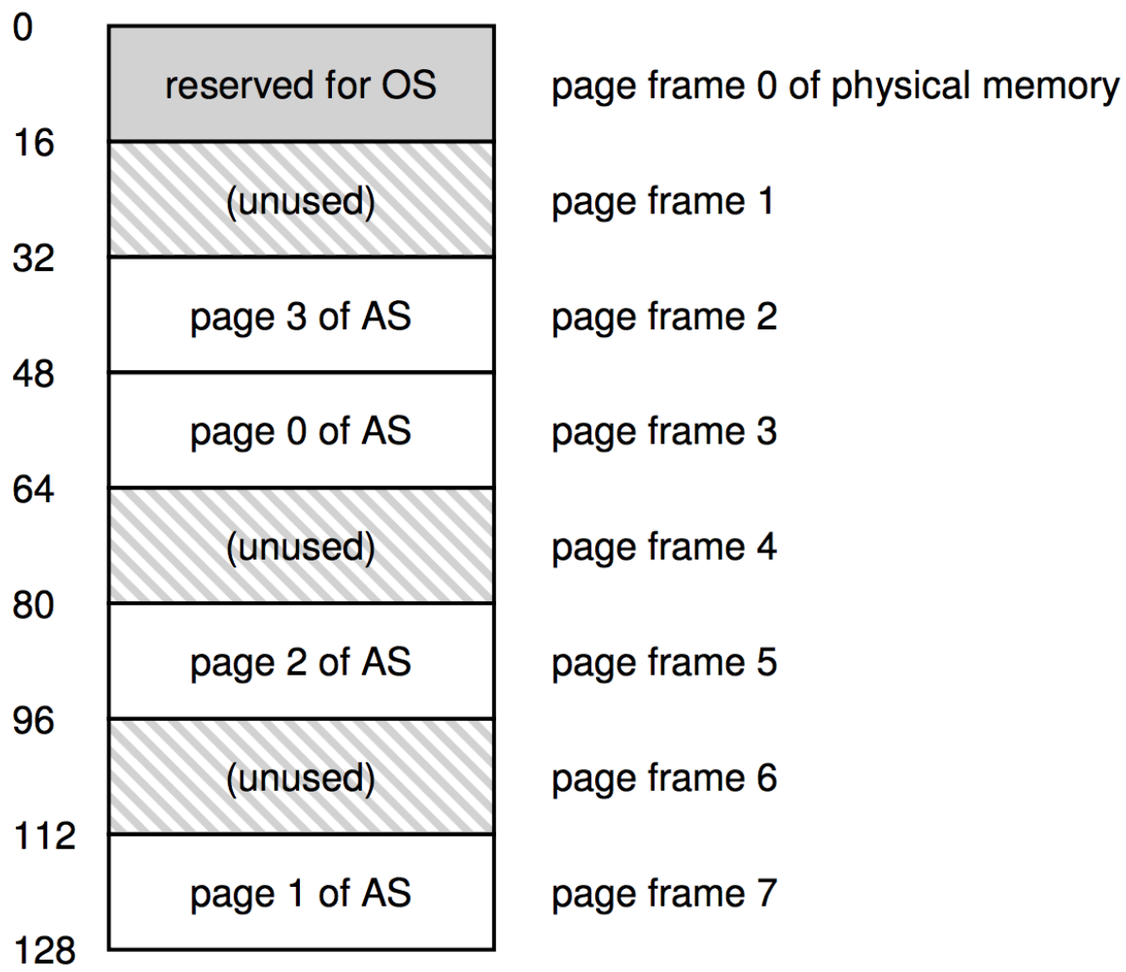
Paging - Divide the Physical Memory too!

Paging - Divide the Physical Memory too!

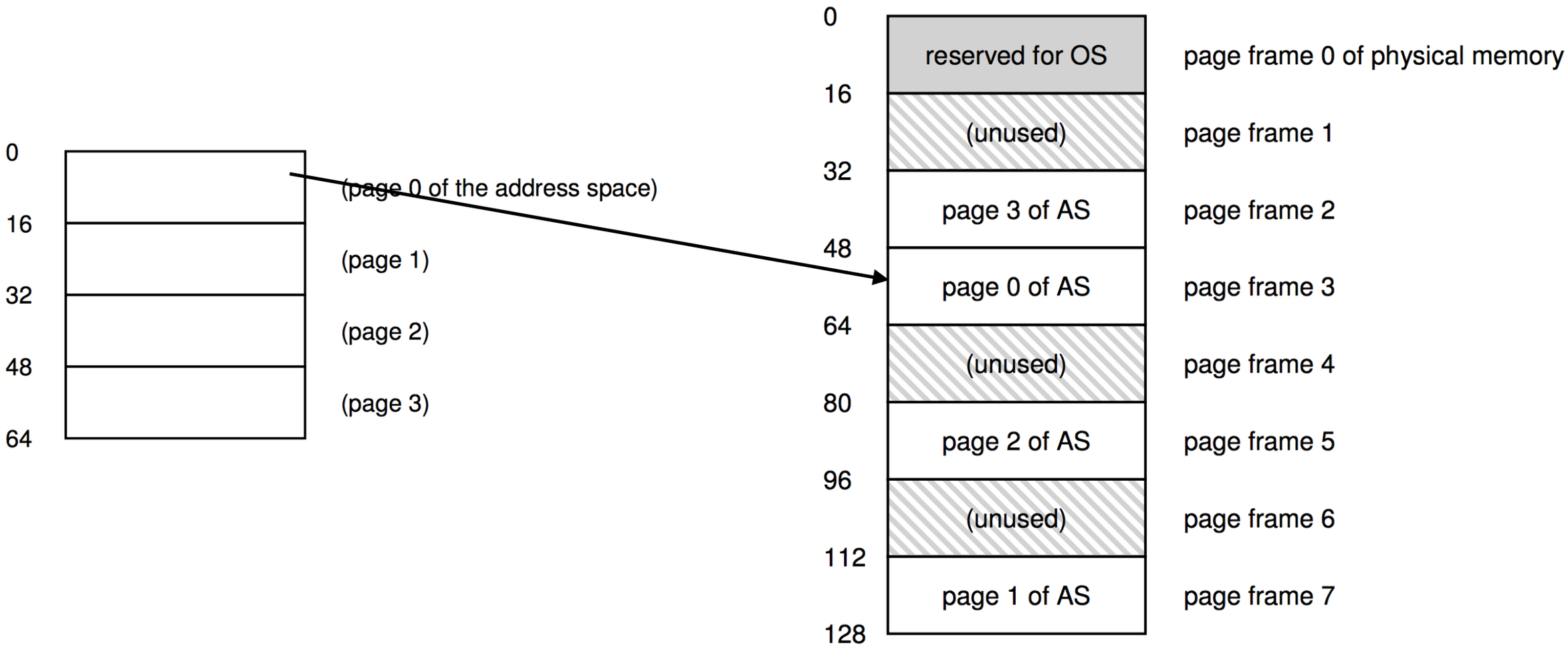


Paging - Divide the Physical Memory too!

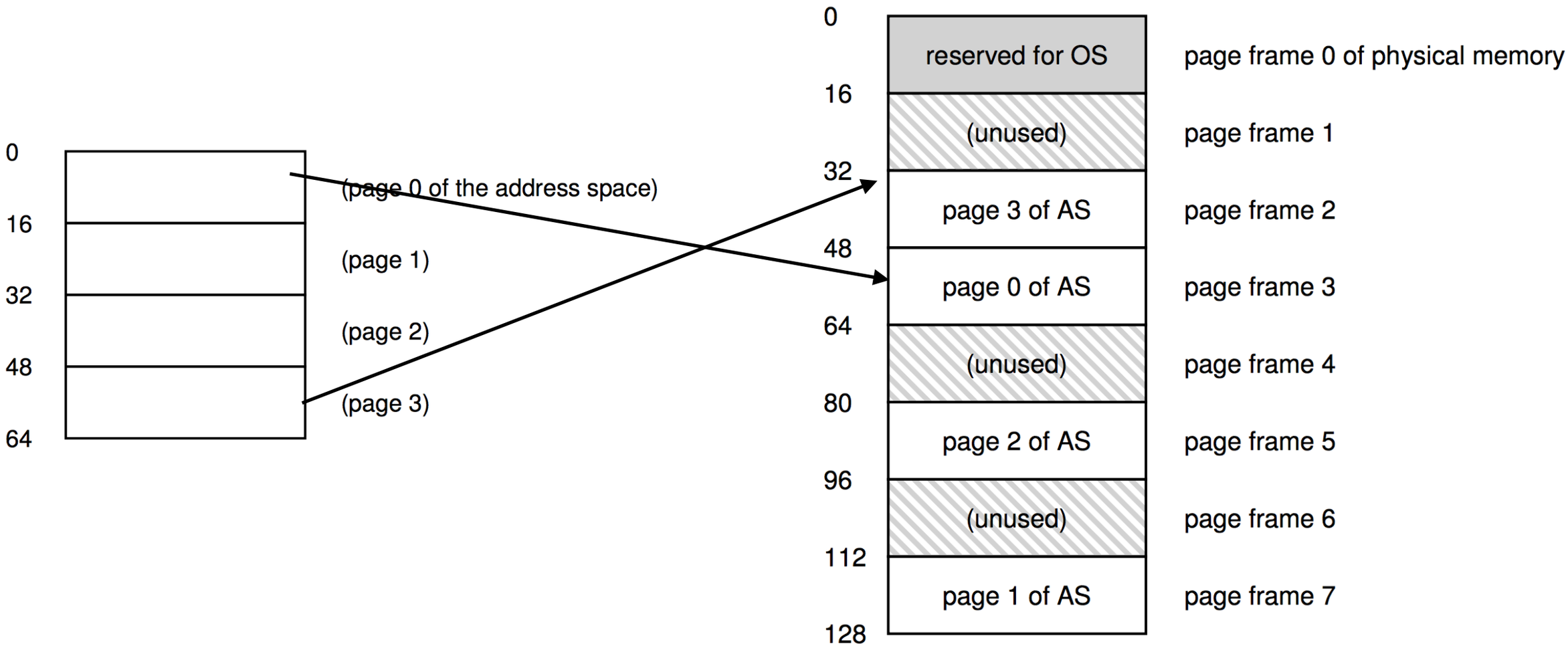




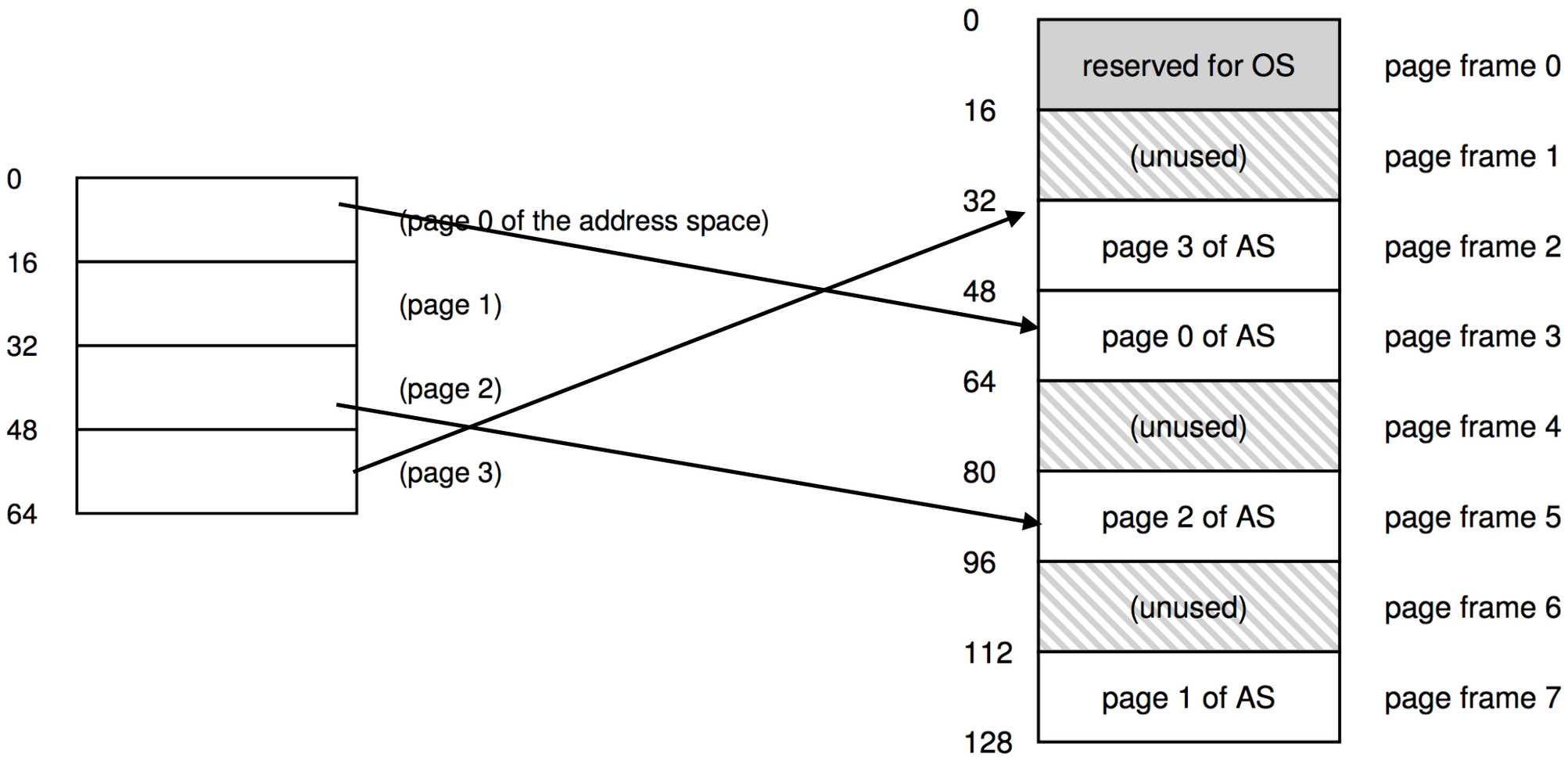
Paging - Divide the Physical Memory too!



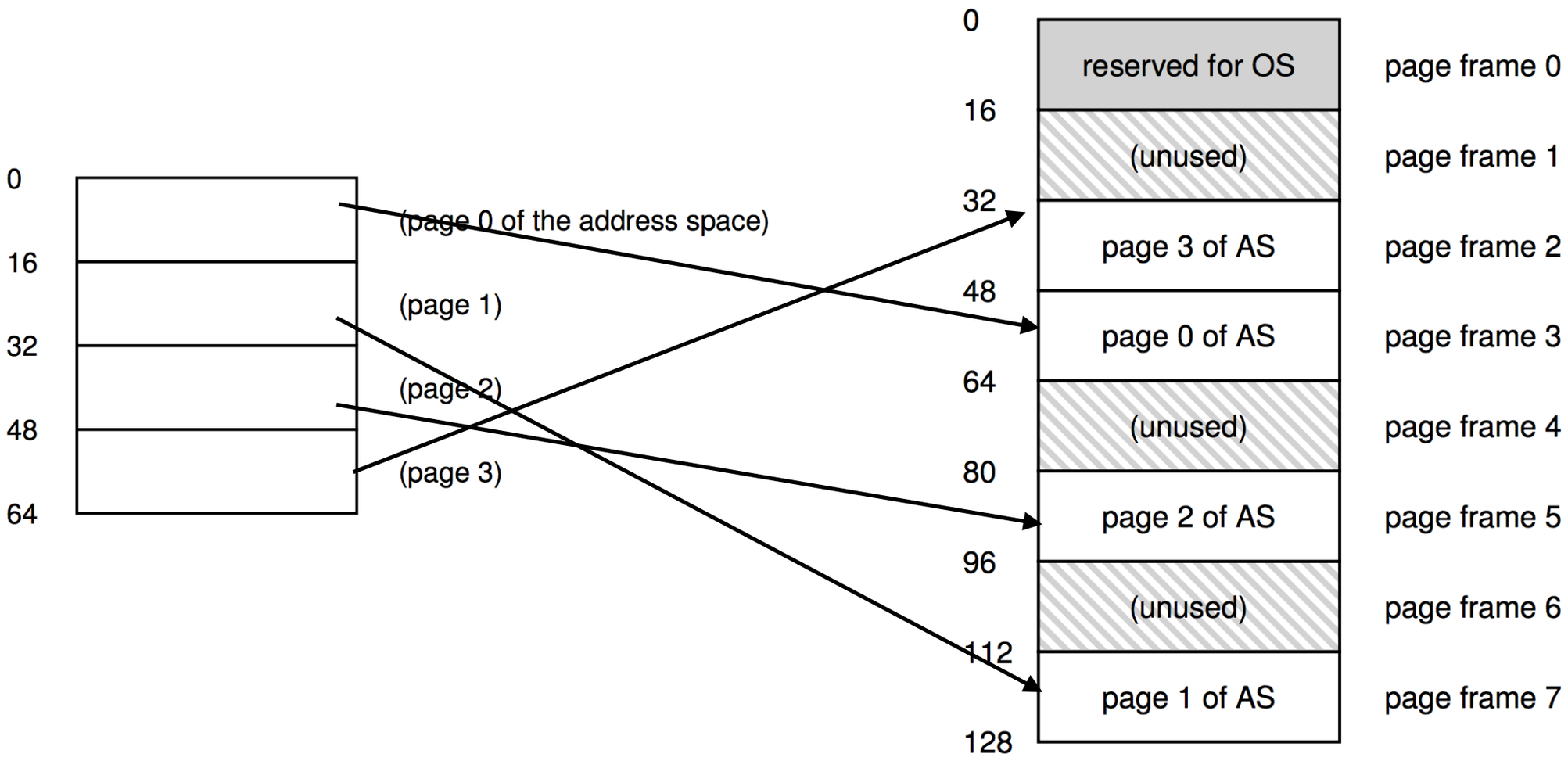
Paging - Divide the Physical Memory too!



Paging - Divide the Physical Memory too!



Paging - Divide the Physical Memory too!



Page Table

Page Table

1. Data structure for storing the mapping between Virtual Page and Physical Page/Frame

Page Table

1. Data structure for storing the mapping between Virtual Page and Physical Page/Frame
2. VP0 -> PF3 …

Page Table

1. Data structure for storing the mapping between Virtual Page and Physical Page/Frame
2. VP0 -> PF3 …
3. Per-process …

Page Table

1. Data structure for storing the mapping between Virtual Page and Physical Page/Frame
2. VP0 -> PF3 …
3. Per-process …

Page Table

64 bytes require 6 bits to represent

Page Table

000000

64 bytes require 6 bits to represent

Page Table

000000

64 bytes require 6 bits to represent

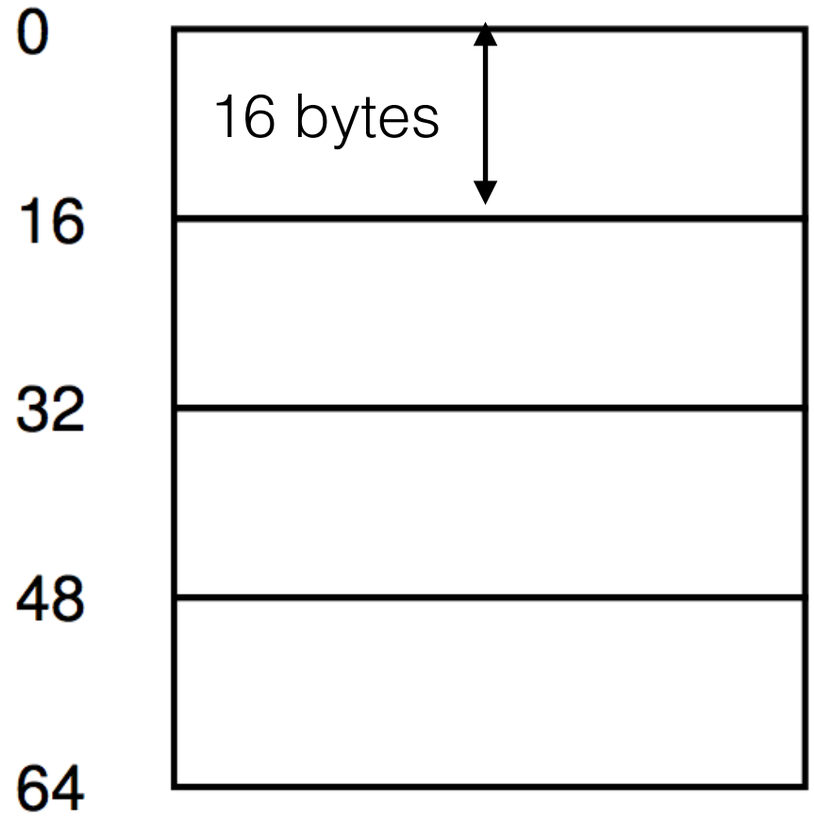
111111

Page Table

000000

64 bytes require 6 bits to represent

111111



Page Table

000000

64 bytes require 6 bits to represent

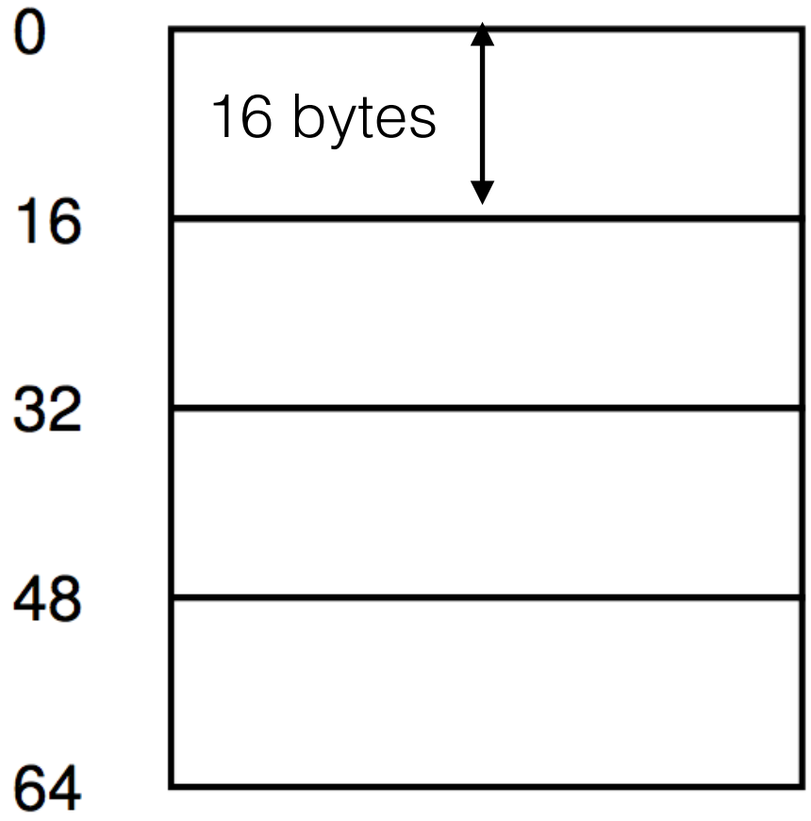
111111

(page 0 of the address space)

(page 1)

(page 2)

(page 3)



Page Table

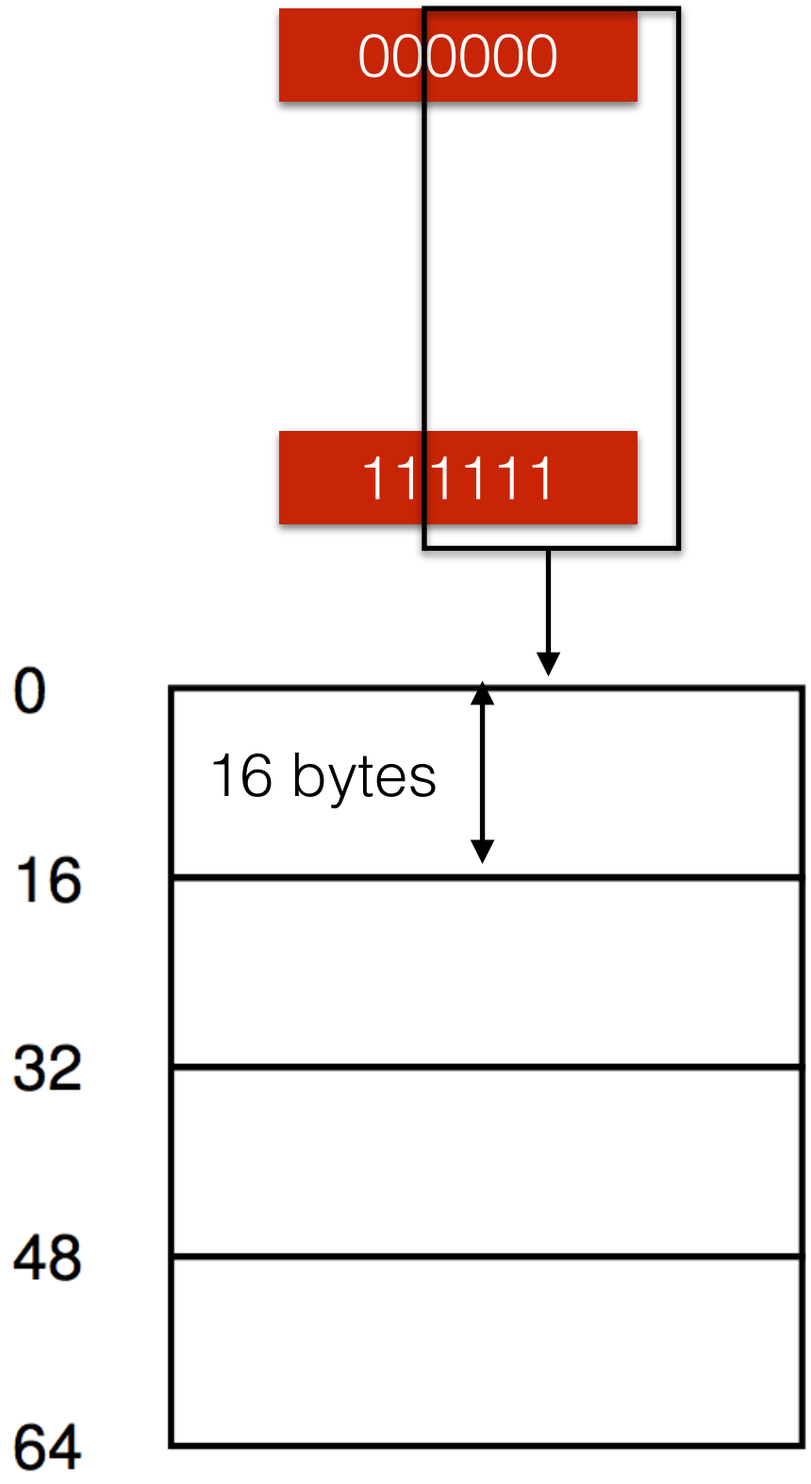
64 bytes require 6 bits to represent

(page 0 of the address space)

(page 1)

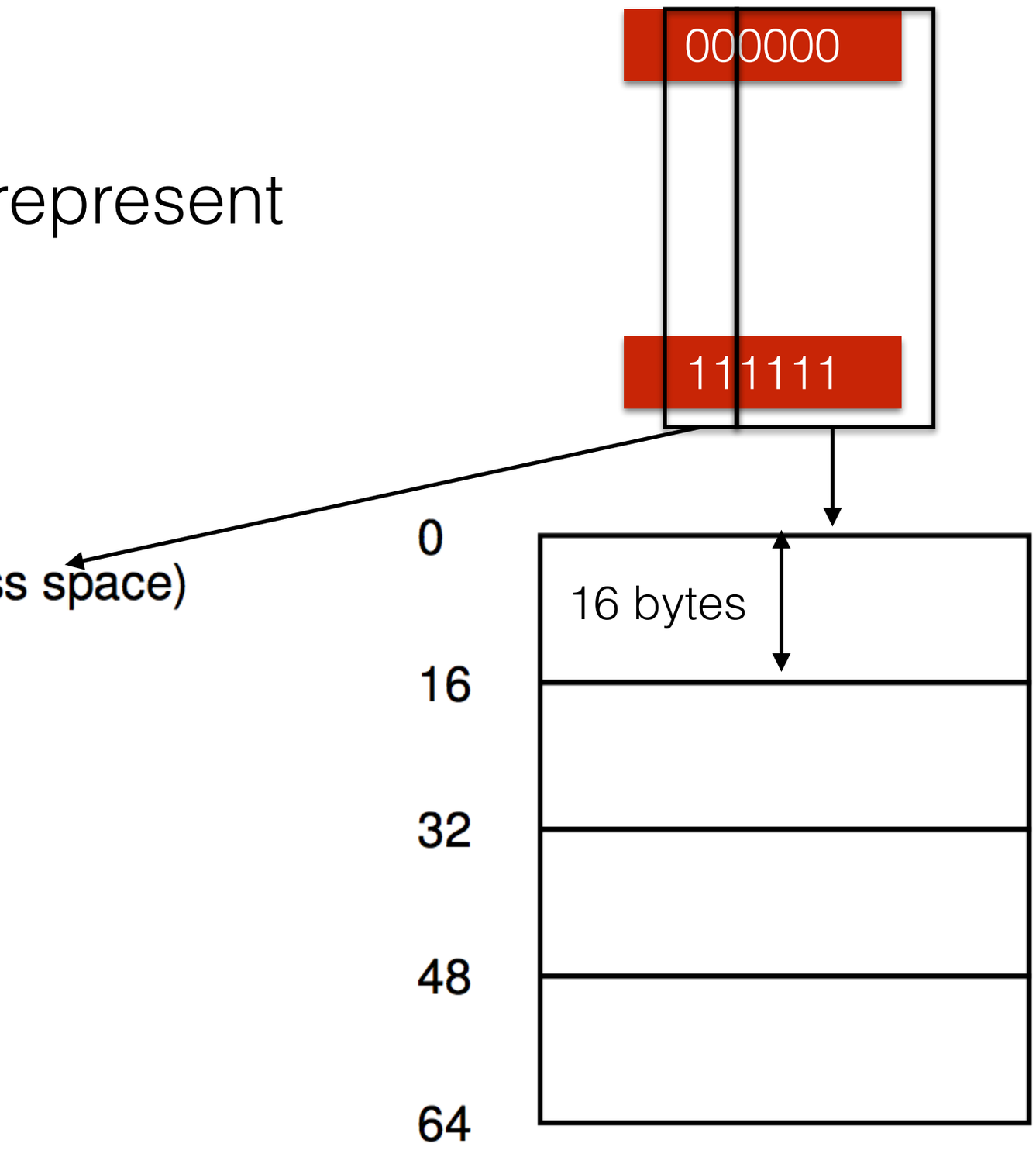
(page 2)

(page 3)



Page Table

64 bytes require 6 bits to represent



Example

21 xorl %eax %eax

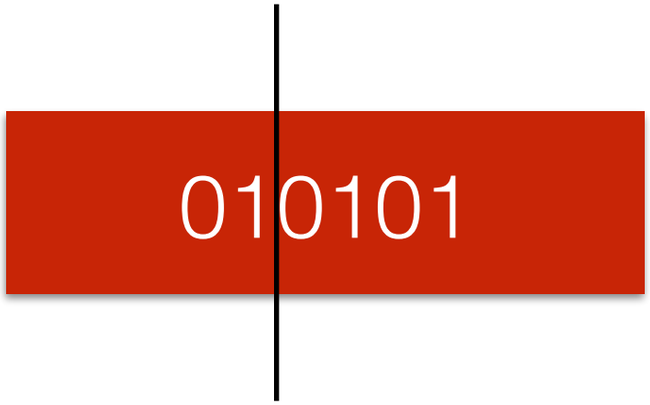
Example

21 xorl %eax %eax

010101

Example

21 xorl %eax %eax

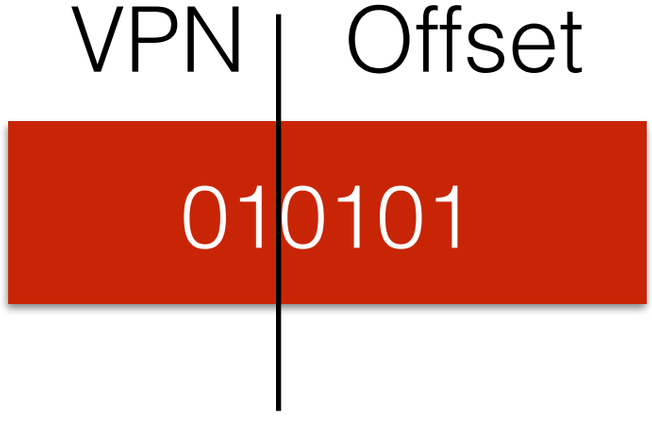


Example

21 xorl %eax %eax

Example

21 xorl %eax %eax

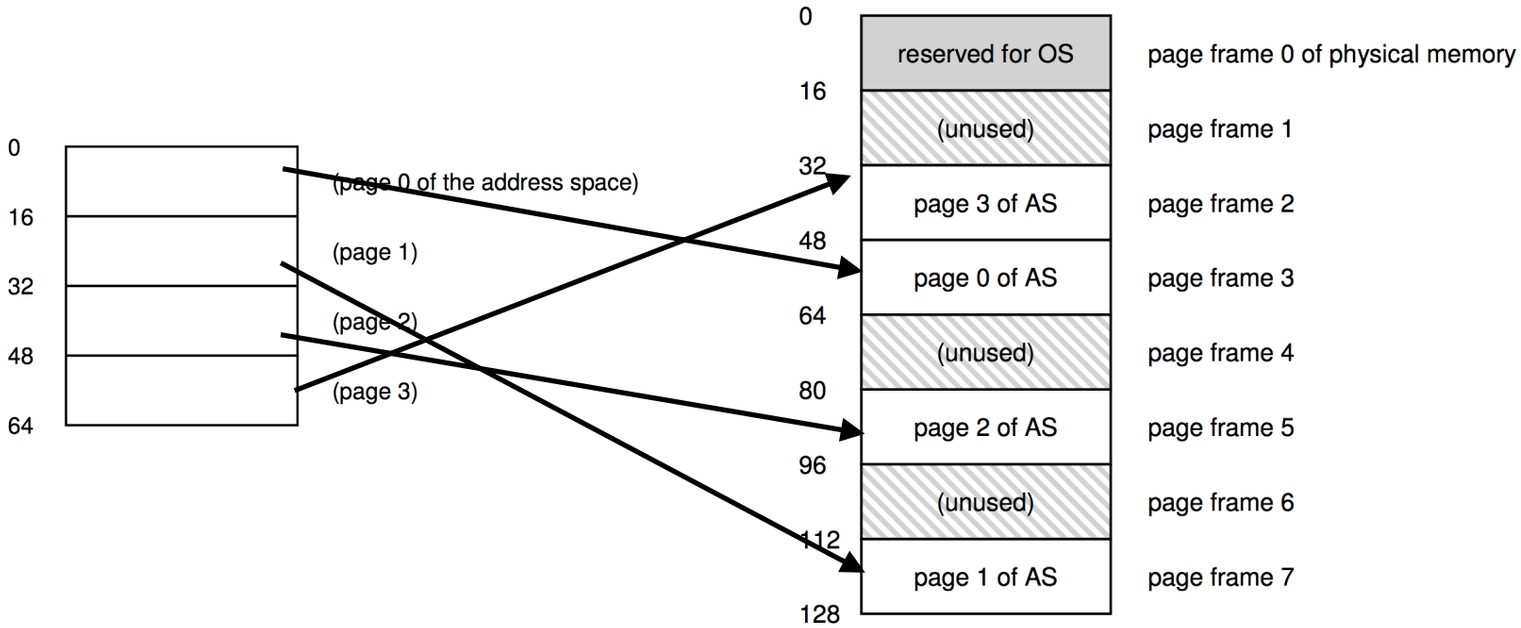


Example

21 xorl %eax %eax

VPN | Offset

010101

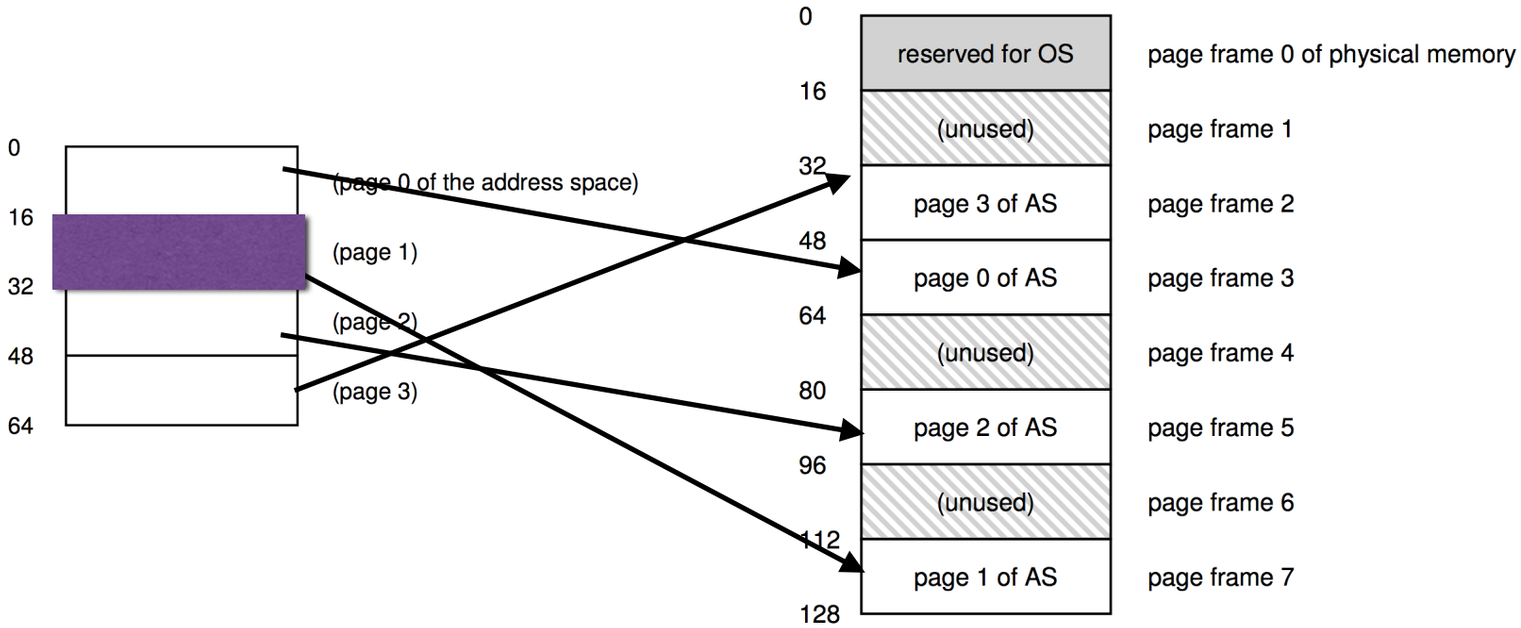


Example

21 xorl %eax %eax

VPN | Offset

010101

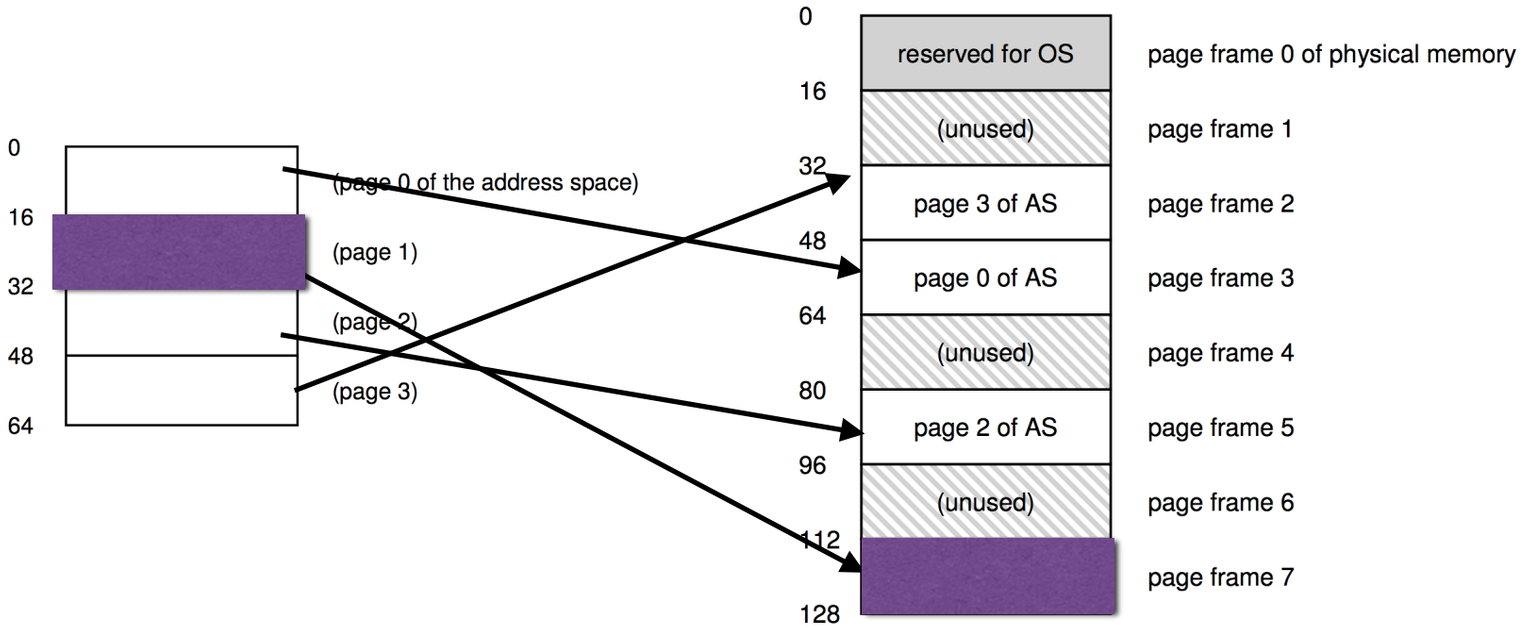


Example

21 xorl %eax %eax

VPN | Offset

010101



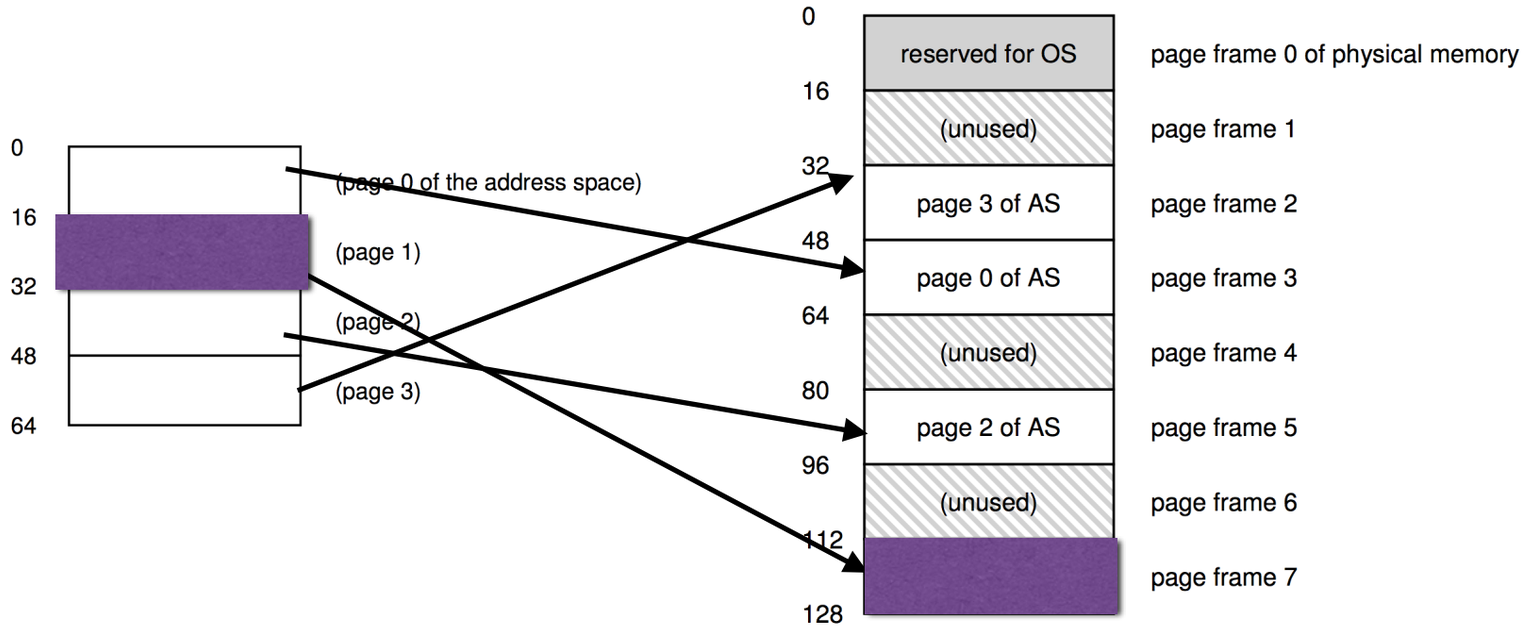
Example

21 xorl %eax %eax

VPN | Offset

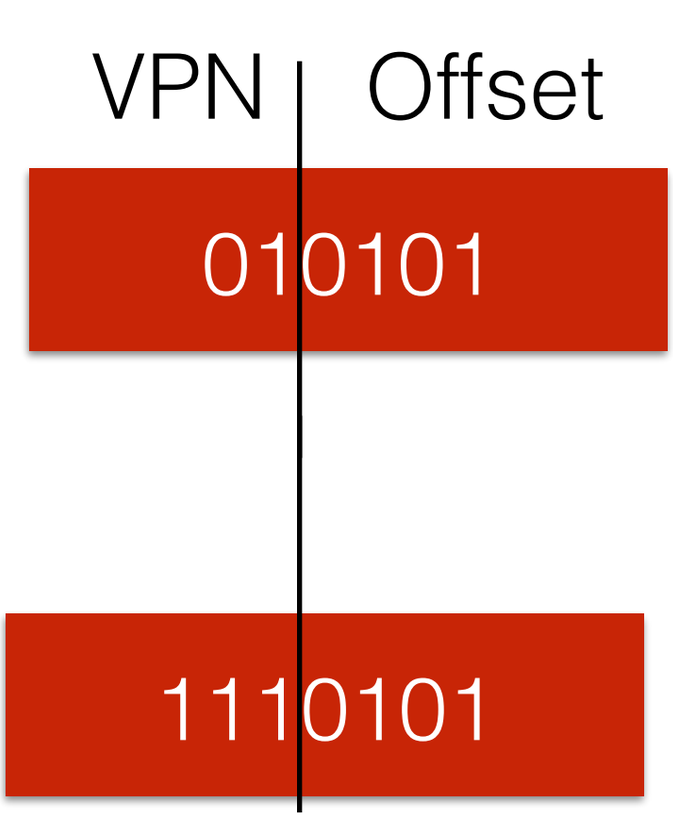
010101

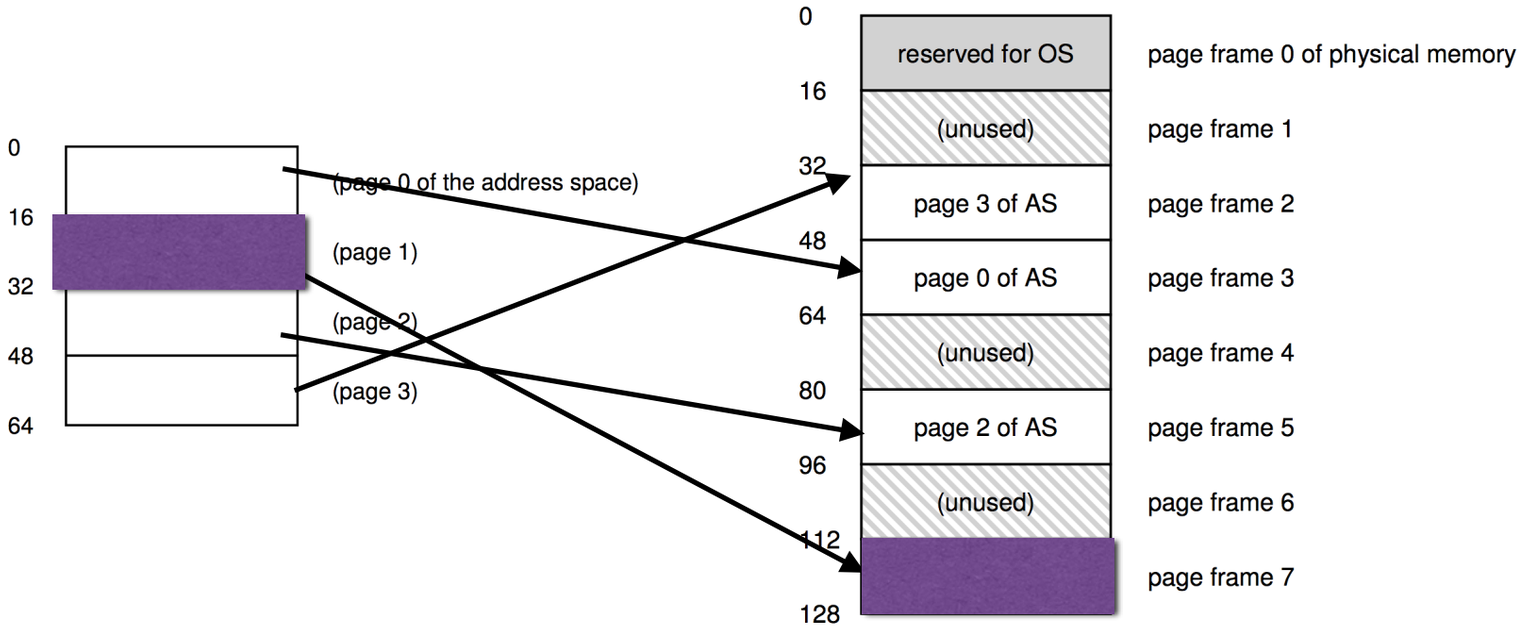
1110101



Example

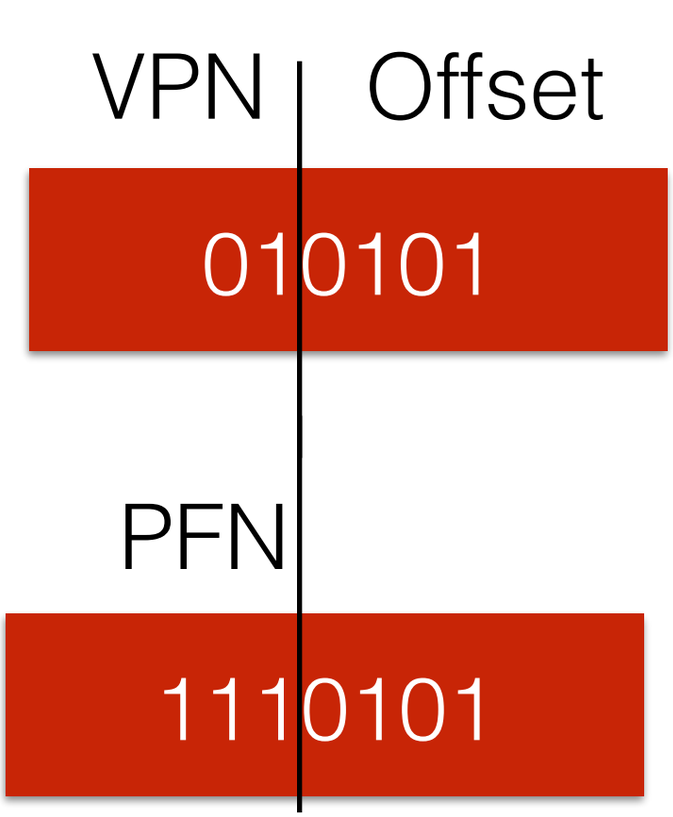
21 xorl %eax %eax

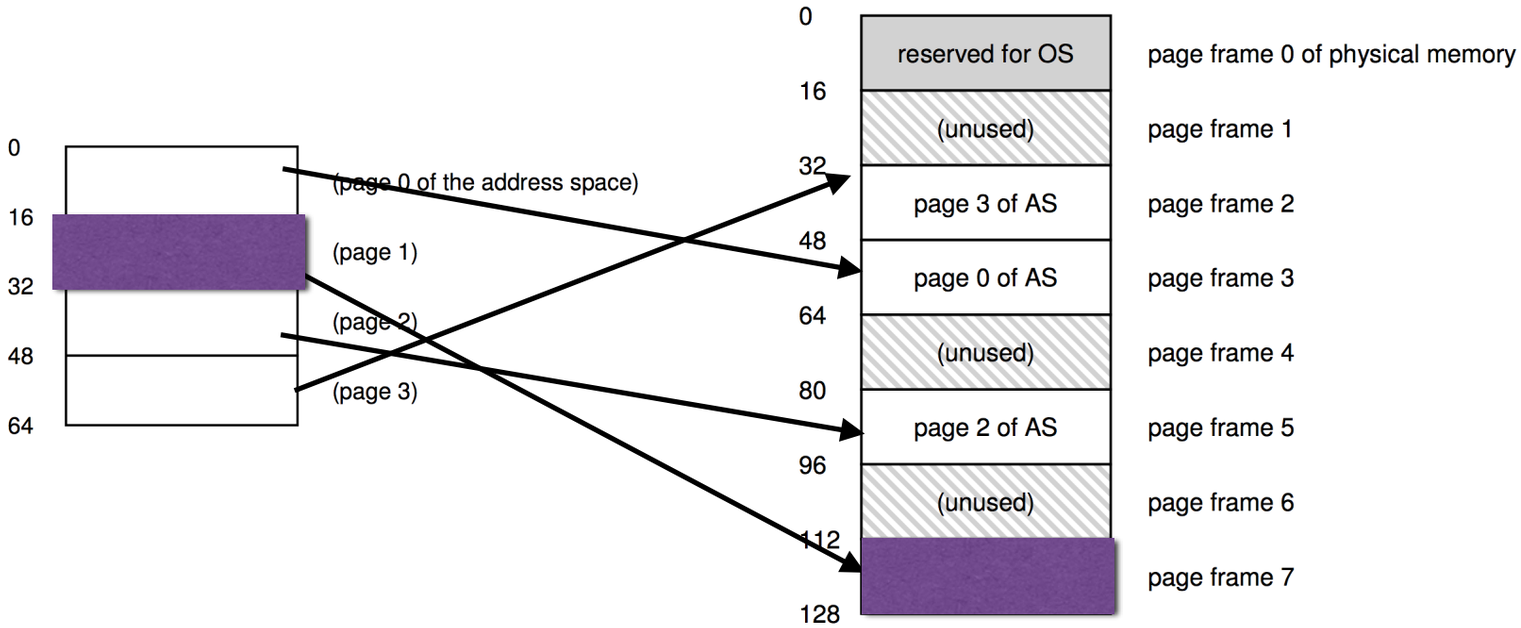




Example

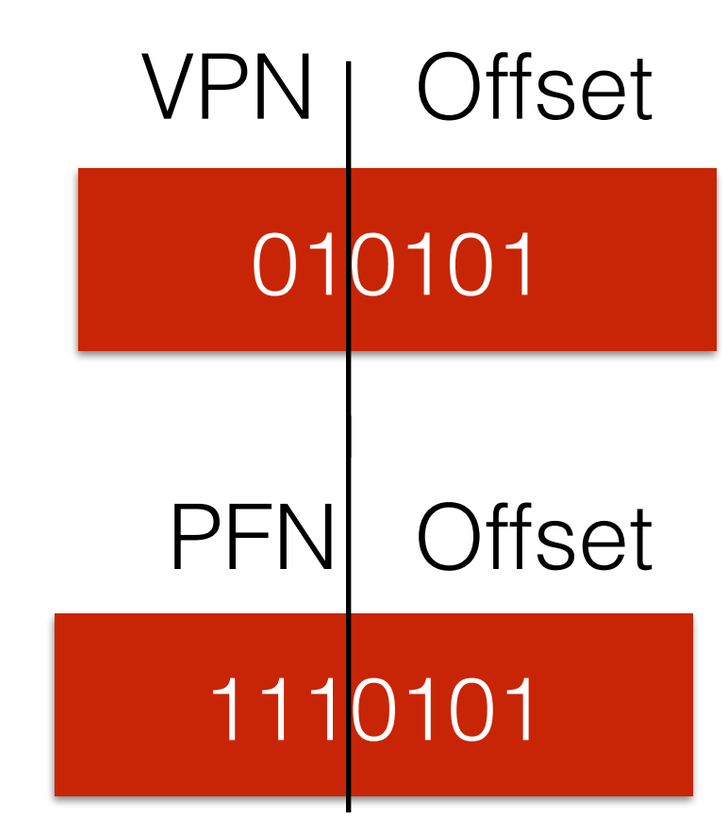
21 xorl %eax %eax

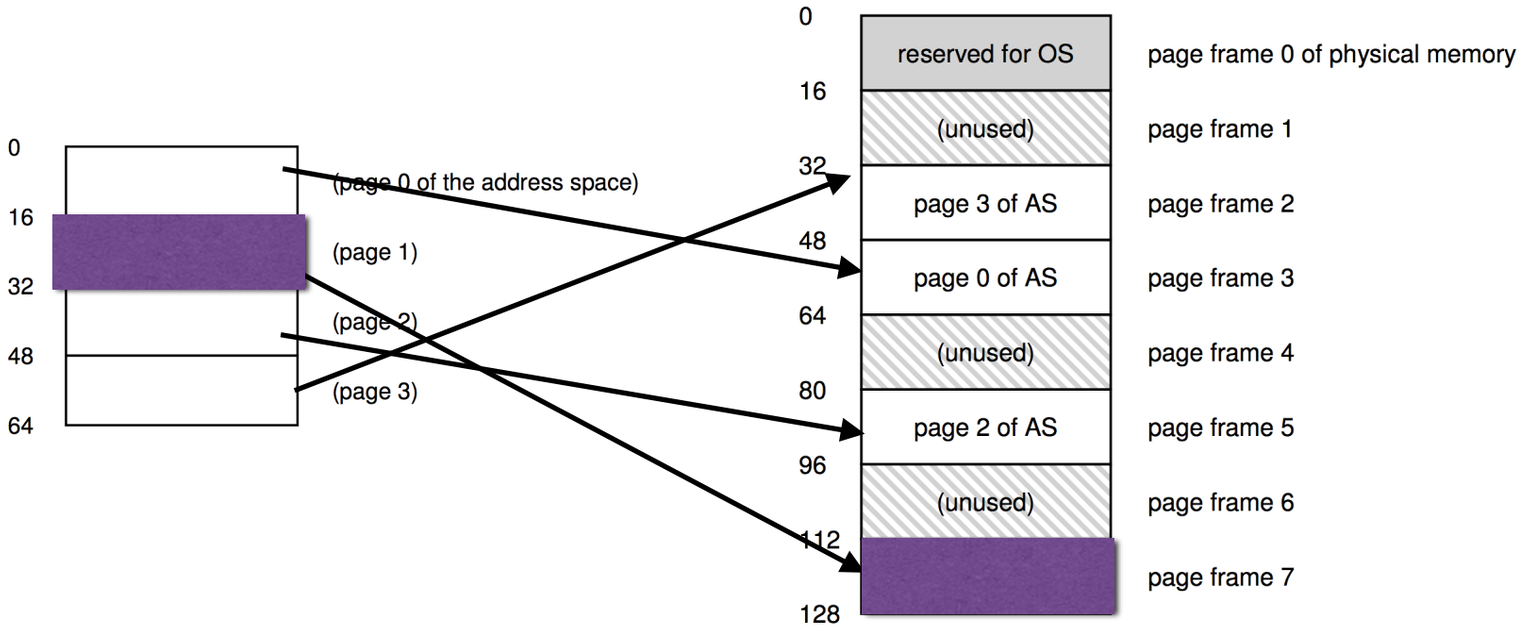




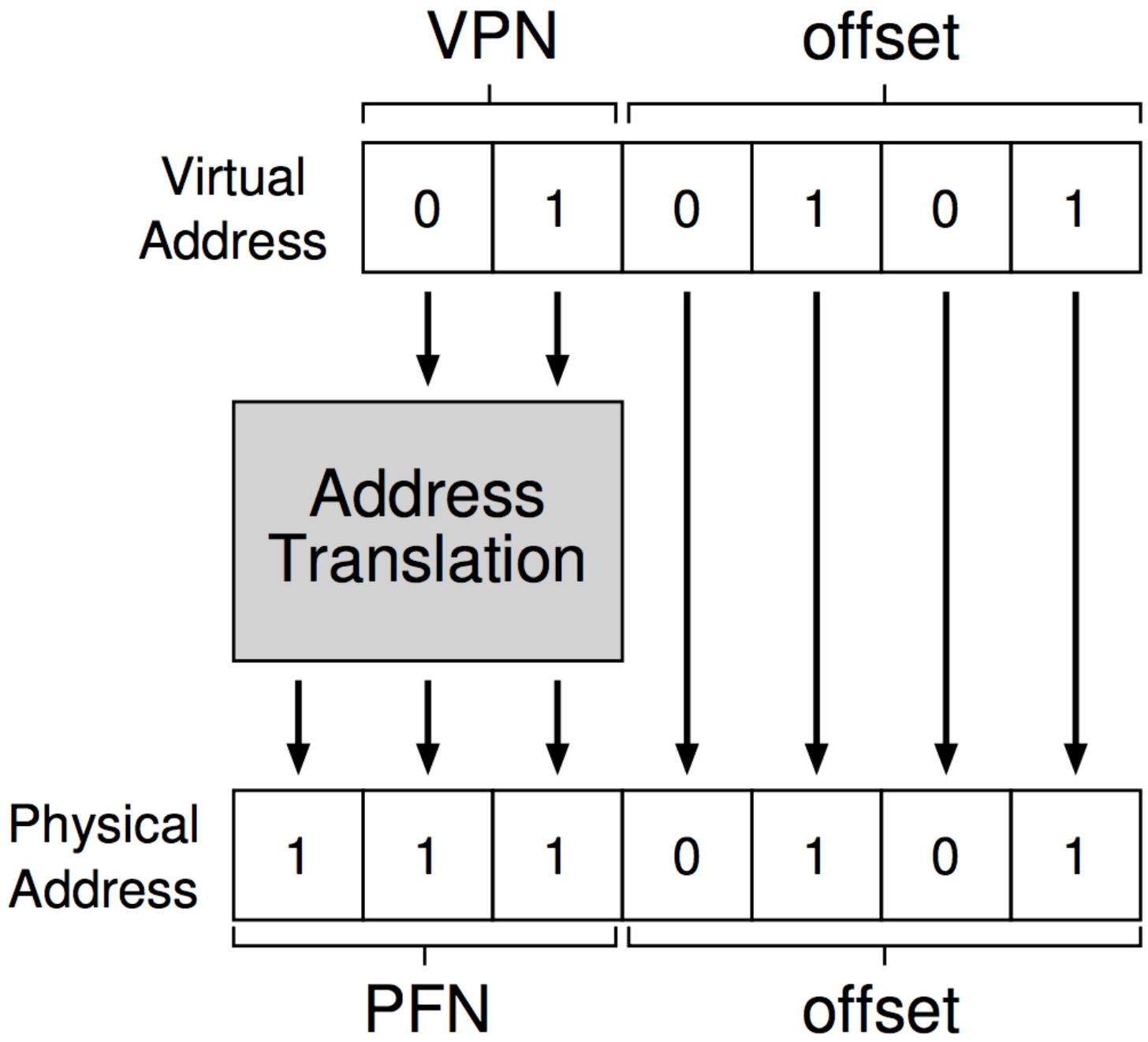
Example

21 xorl %eax %eax





Address Translation Summary



Page Table Storage

Page Table Storage

- Let’s consider 32 bit address space

Page Table Storage

- Let's consider 32 bit address space
- 32 bit address space with 4 KB pages

Page Table Storage

- Let's consider 32 bit address space
- 32 bit address space with 4 KB pages
- 4 KB pages -> ____ bits?

Page Table Storage

- Let's consider 32 bit address space
- 32 bit address space with 4 KB pages
- 4 KB pages -> ____ bits?
  - 12 bits Offset

Page Table Storage

- Let's consider 32 bit address space
- 32 bit address space with 4 KB pages
- 4 KB pages -> ____ bits?
  - 12 bits Offset
- Remaining bits = 32 - 12 = 20

Page Table Storage

- Let's consider 32 bit address space
- 32 bit address space with 4 KB pages
- 4 KB pages -> ____ bits?
  - 12 bits Offset
- Remaining bits = 32 - 12 = 20
  - 20 bit VPN

Page Table Storage

- Let's consider 32 bit address space
- 32 bit address space with 4 KB pages
- 4 KB pages -> ____ bits?
  - 12 bits Offset
- Remaining bits = 32 - 12 = 20
  - 20 bit VPN
  - # pages = 2^20

Page Table Storage

- Let's consider 32 bit address space
- 32 bit address space with 4 KB pages
- 4 KB pages -> ____ bits?
  - 12 bits Offset
- Remaining bits = 32 - 12 = 20
  - 20 bit VPN
  - # pages = 2^20
  - # translations required = ____

Page Table Storage

- Let's consider 32 bit address space
- 32 bit address space with 4 KB pages
- 4 KB pages -> ____ bits?
  - 12 bits Offset
- Remaining bits = 32 - 12 = 20
  - 20 bit VPN
  - # pages = 2^20
  - # translations required = ____
    - 2^20

Page Table Storage

- Let's consider 32 bit address space
- 32 bit address space with 4 KB pages
- 4 KB pages -> ____ bits?
  - 12 bits Offset
- Remaining bits = 32 - 12 = 20
  - 20 bit VPN
  - # pages = 2^20
  - # translations required = ____
    - 2^20
- 4 bytes per translation -> 4 * 2^20 MB = 4 MB/ process

Page Table Storage

Page Table Storage

- Let’s consider 32 bit address space

Page Table Storage

- Let’s consider 32 bit address space
- 32 bit address space with 8 KB pages (previously 4 KB)

Page Table Storage

- Let’s consider 32 bit address space
- 32 bit address space with 8 KB pages (previously 4 KB)
- 8 KB pages -> ____ bits?

Page Table Storage

- Let's consider 32 bit address space
- 32 bit address space with 8 KB pages (previously 4 KB)
- 8 KB pages -> ____ bits?
  - 13 bits Offset

Page Table Storage

- Let's consider 32 bit address space
- 32 bit address space with 8 KB pages (previously 4 KB)
- 8 KB pages -> ____ bits?
  - 13 bits Offset
- Remaining bits = 32 - 13 = 19

Page Table Storage

- Let's consider 32 bit address space
- 32 bit address space with 8 KB pages (previously 4 KB)
- 8 KB pages -> ____ bits?
  - 13 bits Offset
- Remaining bits = 32 - 13 = 19
  - 19 bit VPN

Page Table Storage

- Let's consider 32 bit address space
- 32 bit address space with 8 KB pages (previously 4 KB)
- 8 KB pages -> ____ bits?
  - 13 bits Offset
- Remaining bits = 32 - 13 = 19
  - 19 bit VPN
  - # pages = 2^19

Page Table Storage

- Let's consider 32 bit address space
- 32 bit address space with 8 KB pages (previously 4 KB)
- 8 KB pages -> ____ bits?
  - 13 bits Offset
- Remaining bits = 32 - 13 = 19
  - 19 bit VPN
  - # pages = 2^19
  - # translations required = ____

Page Table Storage

- Let's consider 32 bit address space
- 32 bit address space with 8 KB pages (previously 4 KB)
- 8 KB pages -> ____ bits?
  - 13 bits Offset
- Remaining bits = 32 - 13 = 19
  - 19 bit VPN
  - # pages = 2^19
  - # translations required = ____
    - 2^19

Page Table Storage

- Let's consider 32 bit address space
- 32 bit address space with 8 KB pages (previously 4 KB)
- 8 KB pages -> ____ bits?
  - 13 bits Offset
- Remaining bits = 32 - 13 = 19
  - 19 bit VPN
  - # pages = 2^19
  - # translations required = ____
    - 2^19
- 4 bytes per translation -> 4 * 2^19 MB = 2 MB/process

Page Table Storage

Page Table Storage

- Let’s consider 32 bit address space

Page Table Storage

- Let’s consider 32 bit address space
- 32 bit address space with 512 KB pages (previously 4 KB)

Page Table Storage

- Let’s consider 32 bit address space
- 32 bit address space with 512 KB pages (previously 4 KB)
- 512 KB pages -> ____ bits?

Page Table Storage

- Let’s consider 32 bit address space
- 32 bit address space with 512 KB pages (previously 4 KB)
- 512 KB pages -> ____ bits?
  - 19 bits Offset

Page Table Storage

- Let’s consider 32 bit address space
- 32 bit address space with 512 KB pages (previously 4 KB)
- 512 KB pages -> ____ bits?
  - 19 bits Offset
- Remaining bits = 32 - 19 = 13

Page Table Storage

- Let's consider 32 bit address space
- 32 bit address space with 512 KB pages (previously 4 KB)
- 512 KB pages -> ____ bits?
  - 19 bits Offset
- Remaining bits = 32 - 19 = 13
  - 13 bit VPN

Page Table Storage

- Let's consider 32 bit address space
- 32 bit address space with 512 KB pages (previously 4 KB)
- 512 KB pages -> ____ bits?
  - 19 bits Offset
- Remaining bits = 32 - 19 = 13
  - 13 bit VPN
  - # pages = 2^13

Page Table Storage

- Let's consider 32 bit address space
- 32 bit address space with 512 KB pages (previously 4 KB)
- 512 KB pages -> ____ bits?
  - 19 bits Offset
- Remaining bits = 32 - 19 = 13
  - 13 bit VPN
  - # pages = 2^13
  - # translations required = ____

Page Table Storage

- Let's consider 32 bit address space
- 32 bit address space with 512 KB pages (previously 4 KB)
- 512 KB pages -> ____ bits?
  - 19 bits Offset
- Remaining bits = 32 - 19 = 13
  - 13 bit VPN
  - # pages = 2^13
  - # translations required = ____
    - 2^13

Page Table Storage

- Let's consider 32 bit address space
- 32 bit address space with 512 KB pages (previously 4 KB)
- 512 KB pages -> ____ bits?
  - 19 bits Offset
- Remaining bits = 32 - 19 = 13
  - 13 bit VPN
  - # pages = 2^13
  - # translations required = ____
    - 2^13
- 4 bytes per translation -> 4 * 2^13

Page Size Tradeoffs?

Page Size Tradeoffs?

- Small size

Page Size Tradeoffs?

- Small size
  - More # of translations

Page Size Tradeoffs?

- Small size
  - More # of translations
    - More memory overhead/process

Page Size Tradeoffs?

- Small size
  - More # of translations
    - More memory overhead/process
    - Less chances of fragmentation

Page Size Tradeoffs?

- Small size
  - More # of translations
    - More memory overhead/process
    - Less chances of fragmentation
- Large size

Page Size Tradeoffs?

- Small size
  - More # of translations
    - More memory overhead/process
    - Less chances of fragmentation
- Large size
  - Less # of translations

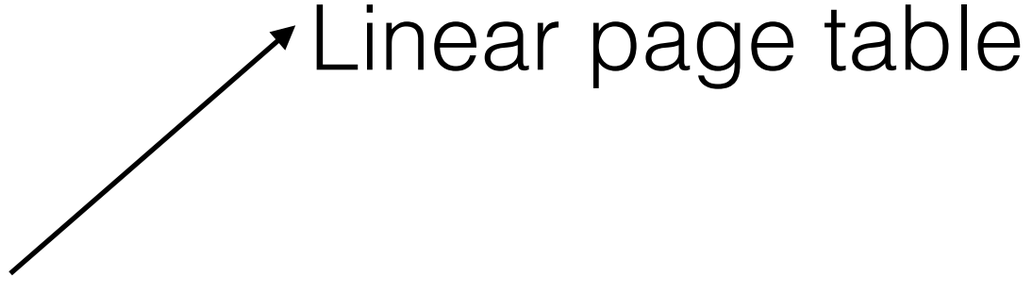
Page Size Tradeoffs?

- Small size
  - More # of translations
    - More memory overhead/process
    - Less chances of fragmentation
- Large size
  - Less # of translations
    - Less memory overhead/process

Page Size Tradeoffs?

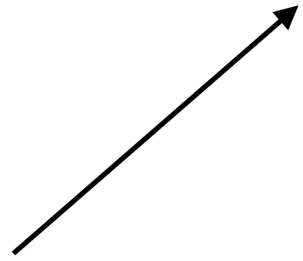
- Small size
  - More # of translations
    - More memory overhead/process
    - Less chances of fragmentation
- Large size
  - Less # of translations
    - Less memory overhead/process
    - More chances of fragmentation

Page Table Storage



Page Table Storage

Not really stored on MMU

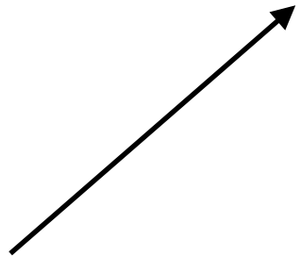


Linear page table

Page Table Storage

Not really stored on MMU

- In memory

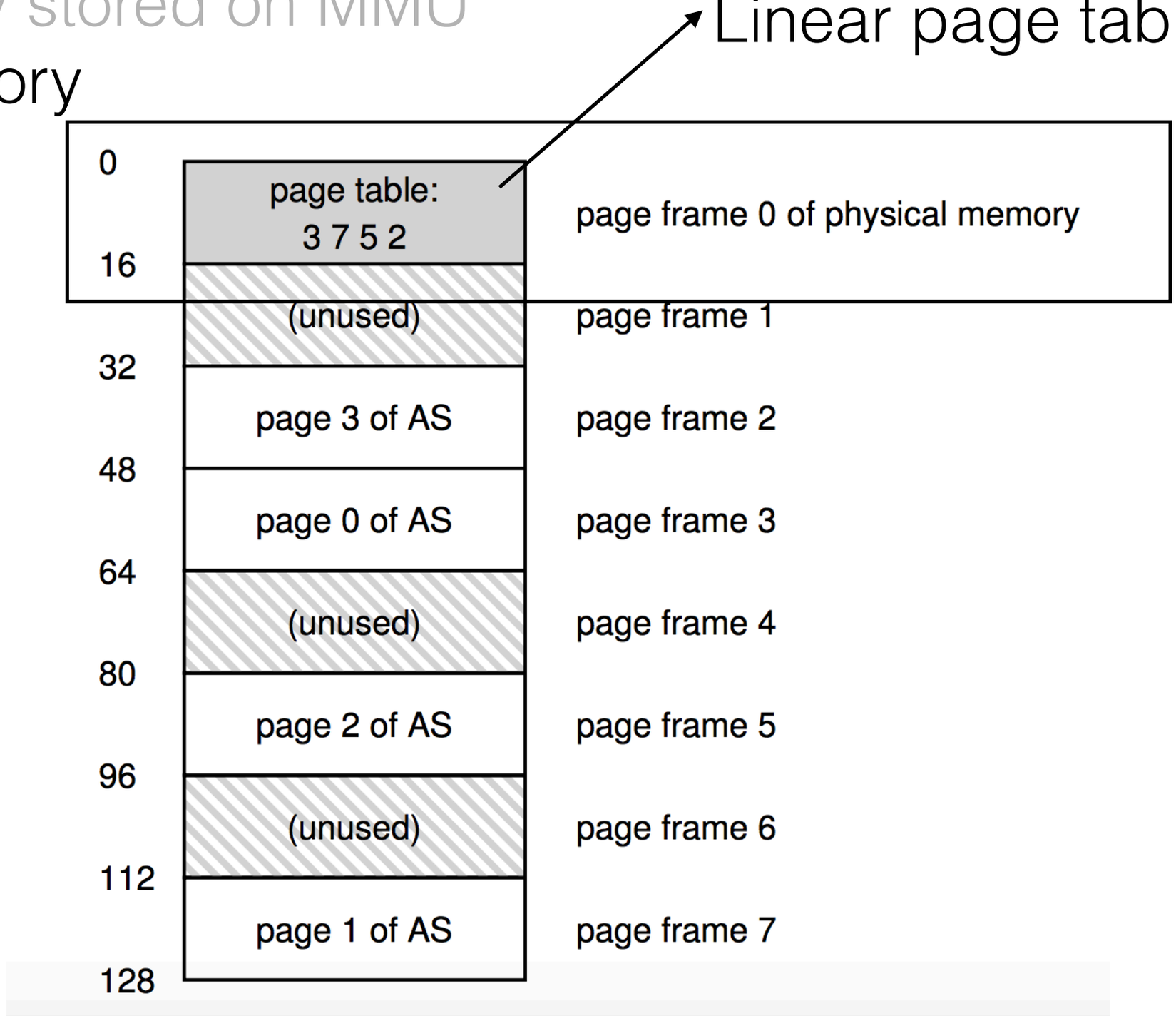


Linear page table

Page Table Storage

Not really stored on MMU

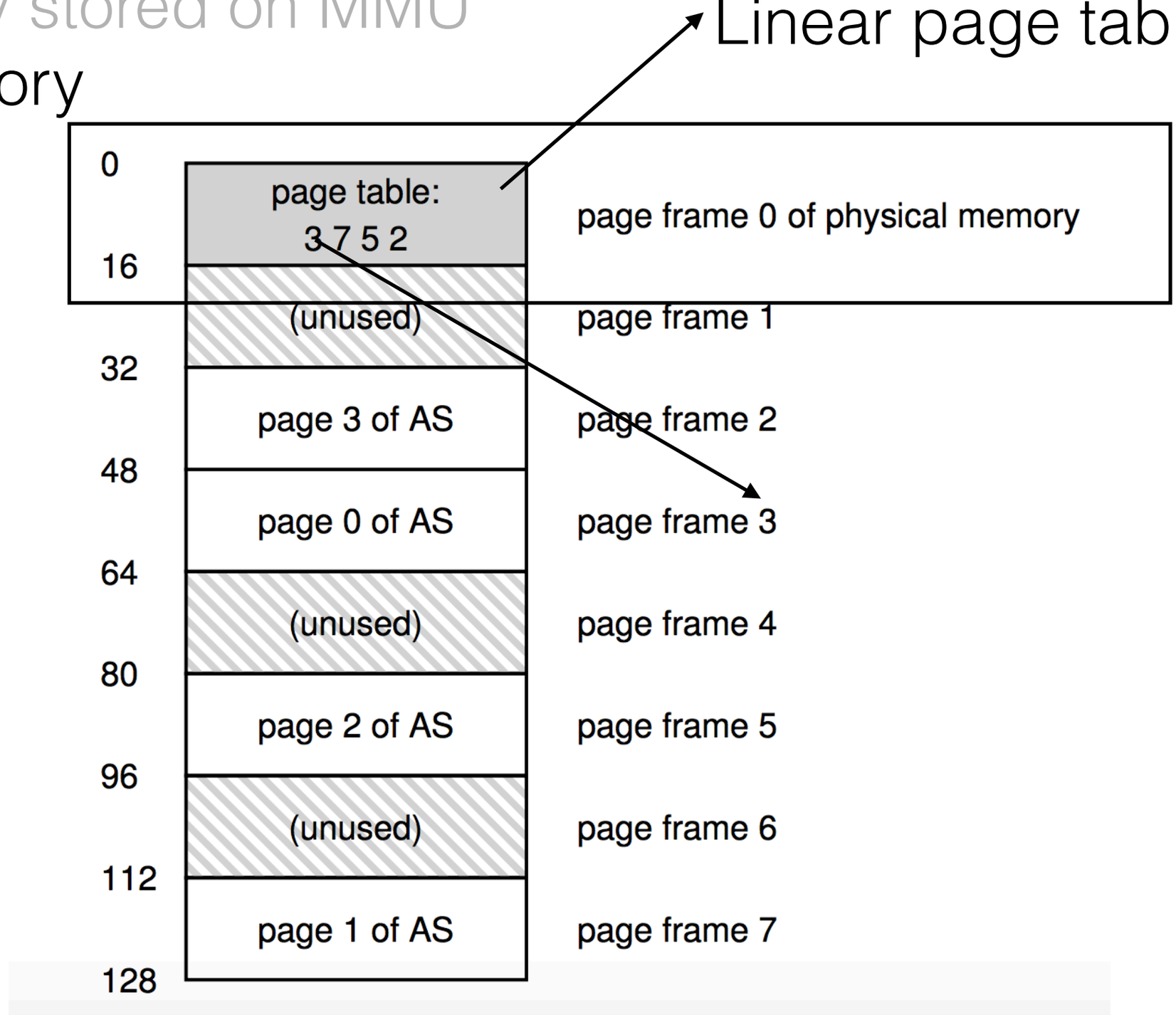
- In memory



Page Table Storage

Not really stored on MMU

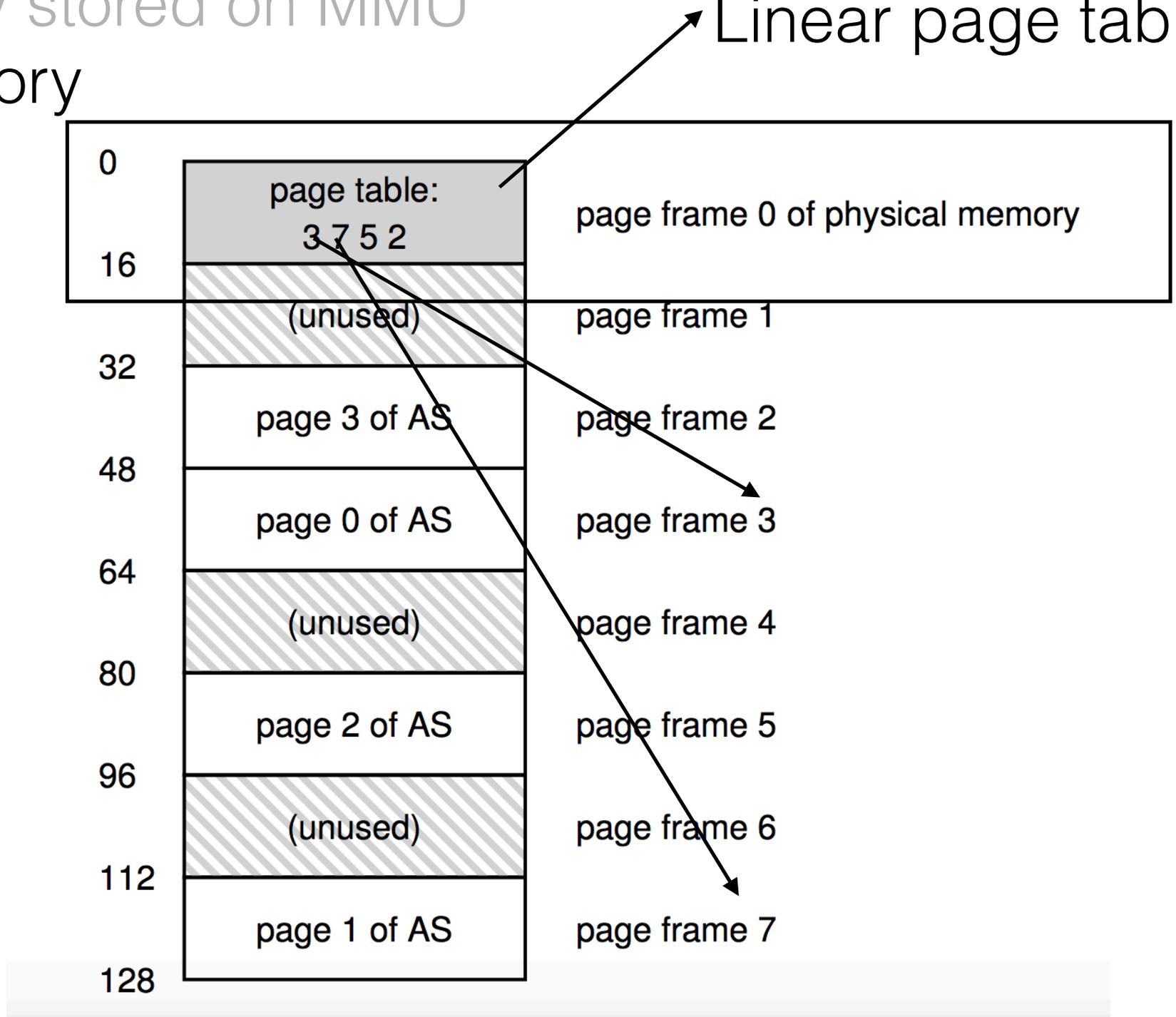
- In memory



Page Table Storage

Not really stored on MMU

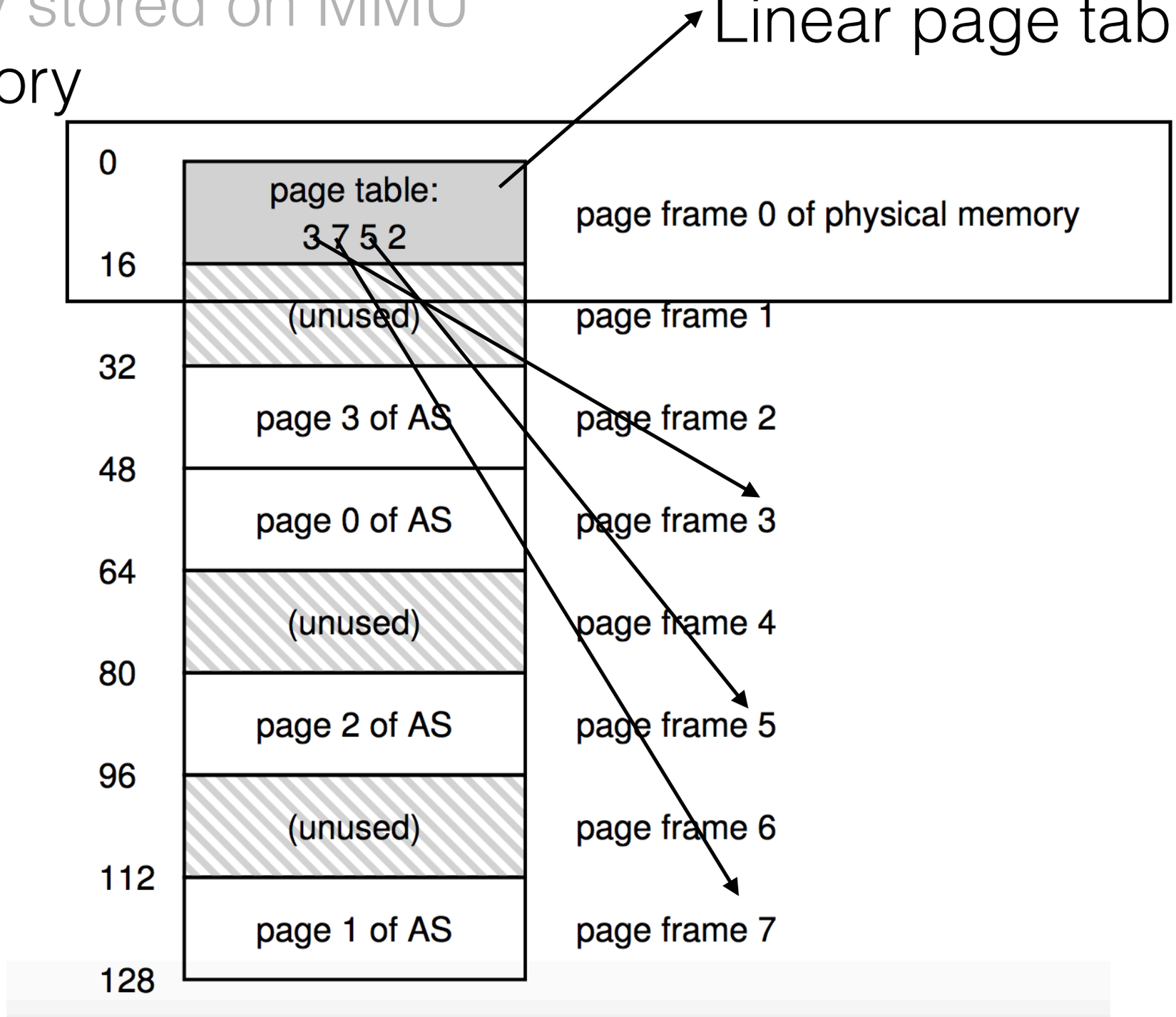
- In memory



Page Table Storage

Not really stored on MMU

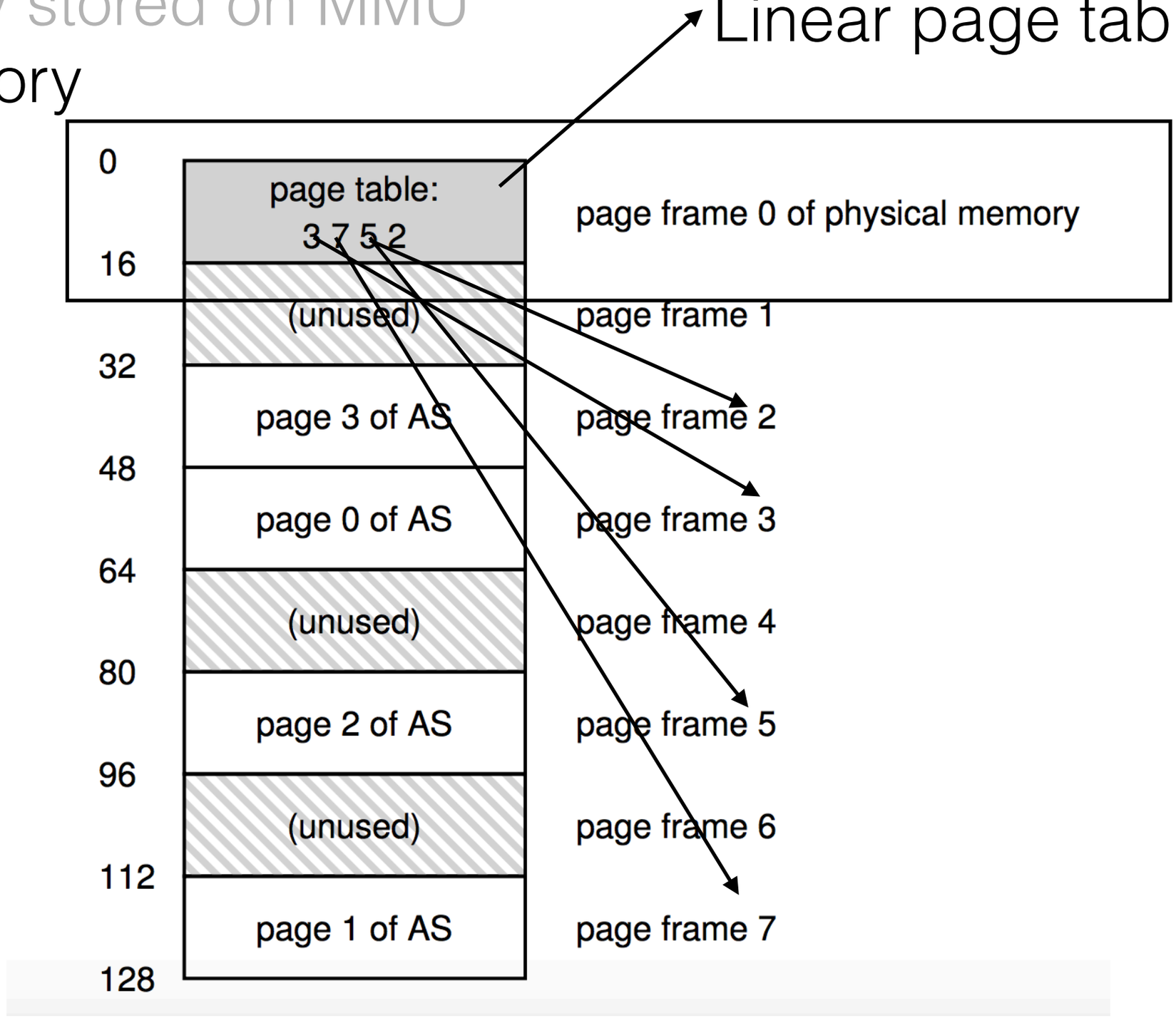
- In memory



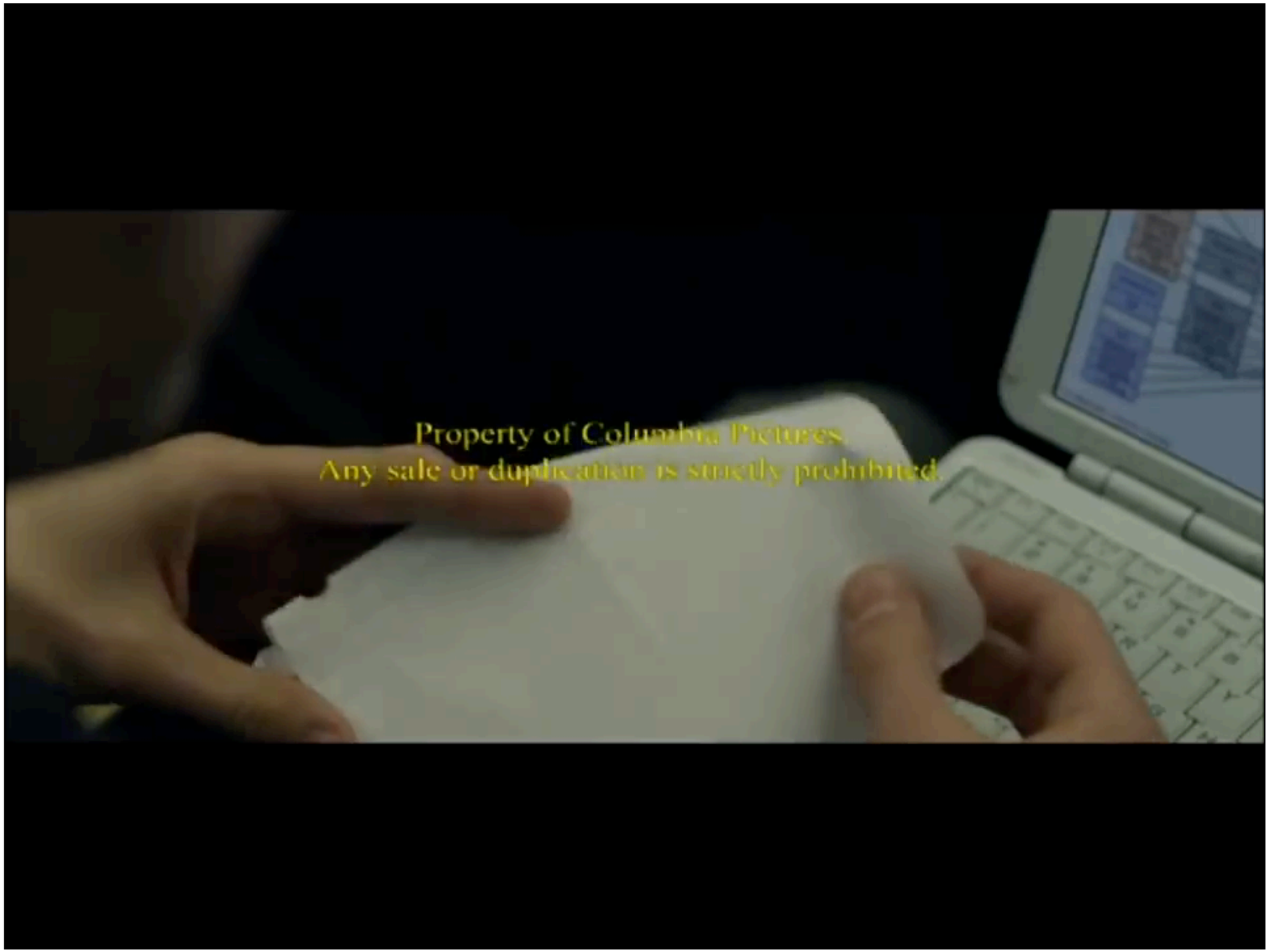
Page Table Storage

Not really stored on MMU

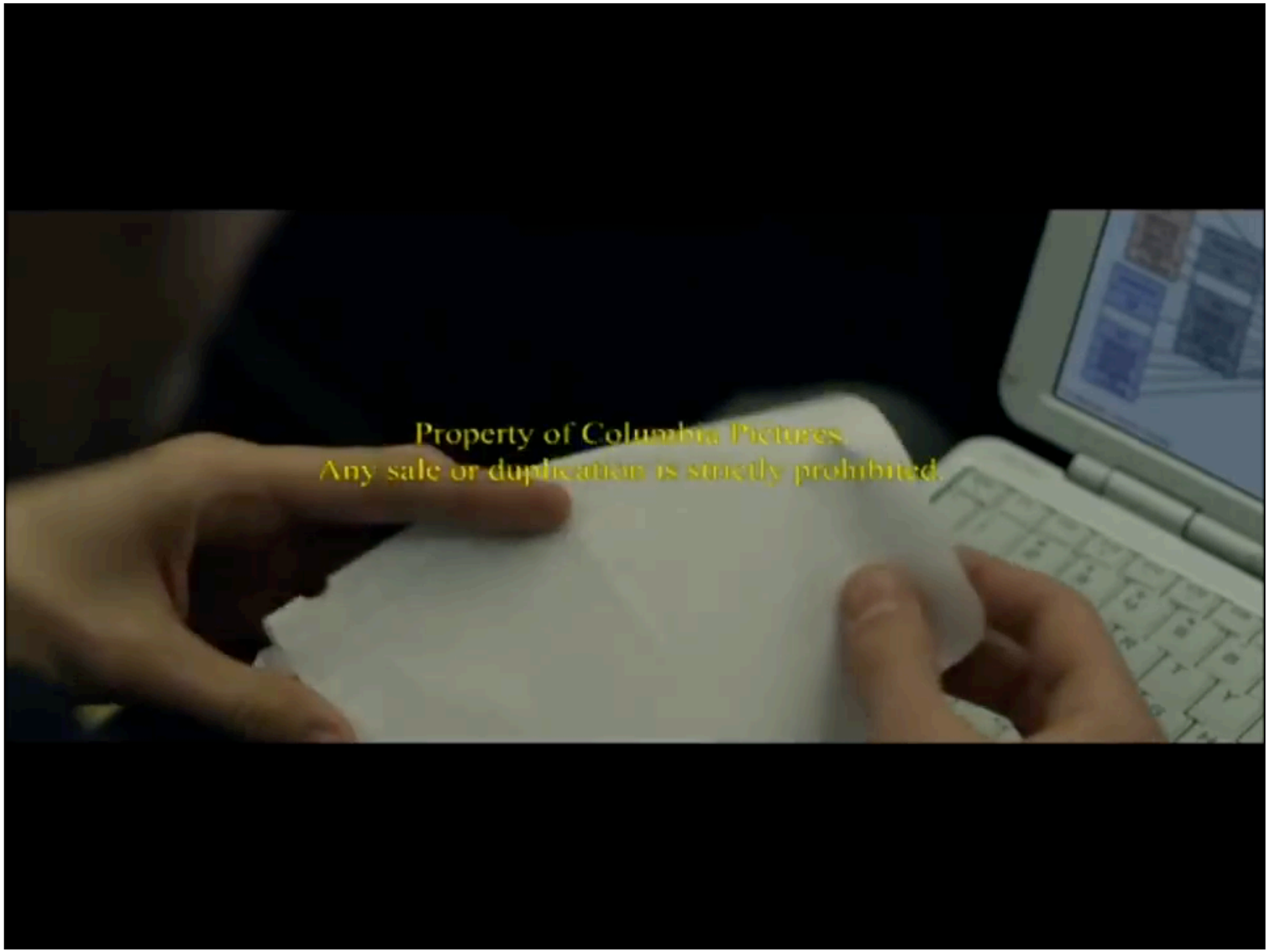
- In memory



What's in the Page Table?



What's in the Page Table?



What else is in the Page Table?

What else is in the Page Table?

What else is in the Page Table?

- Protection bit : Read/Write/Execute?

What else is in the Page Table?

- Protection bit : Read/Write/Execute?
- Present bit: On Memory or HDD/SSD?

What else is in the Page Table?

- Protection bit : Read/Write/Execute?
- Present bit: On Memory or HDD/SSD?
- Reference bit: Is the page popular/being referenced?

What else is in the Page Table?

- Protection bit : Read/Write/Execute?
- Present bit: On Memory or HDD/SSD?
- Reference bit: Is the page popular/being referenced?
  - Else?

What else is in the Page Table?

- Protection bit : Read/Write/Execute?
- Present bit: On Memory or HDD/SSD?
- Reference bit: Is the page popular/being referenced?
  - Else?
- Valid bit: Is translation valid?

What else is in the Page Table?

- Protection bit : Read/Write/Execute?
- Present bit: On Memory or HDD/SSD?
- Reference bit: Is the page popular/being referenced?
  - Else?
- Valid bit: Is translation valid?
- Dirty bit: Modified since brought to memory?

What's coming

What's coming

- Cache them up!

What's coming

- Cache them up!
- OS memory management

What's coming

- Cache them up!
- OS memory management
- Optimised data structure for storing page tables

What's coming

- Cache them up!
- OS memory management
- Optimised data structure for storing page tables
  - Page table for a page table

What's coming

- Cache them up!
- OS memory management
- Optimised data structure for storing page tables
  - Page table for a page table

What's coming

- Cache them up!
- OS memory management
- Optimised data structure for storing page tables
  - Page table for a page table