An Introduction to Virtual Memory

Russell Lewis

A Virtual Computer

Process A

```
load  r1, mem(0)
inc   r1
store r1, mem(1)
```

r1: 0

Memory

0	17
1	0
2	0
3	0
4	0
5	0
6	0
7	0

A Virtual Computer

Process A

load r1, mem(0)
inc r1
store r1, mem(1)

r1 17

Memory

0	17
1	0
2	0
3	0
4	0
5	0
6	0
7	0

A Virtual Computer

Process A

```
load  r1, mem(0)
inc   r1
store r1, mem(1)
```

r1: **18**

Memory

0	17
1	0
2	0
3	0
4	0
5	0
6	0
7	0

A Virtual Computer

Process A

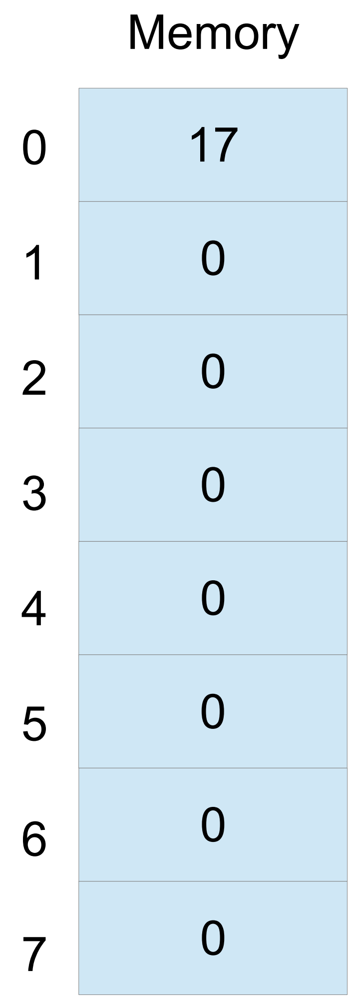
```
load  r1, mem(0)
inc   r1
store r1, mem(1)
```

r1: 18

Memory

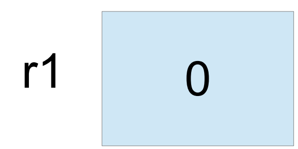
0	17
1	**18**
2	0
3	0
4	0
5	0
6	0
7	0

A Virtual Computer



Process B

```
load  r1, mem(0)
mul   r1, r1,r1
store r1, mem(1)
```



A Virtual Computer

Memory

0	17
1	0
2	0
3	0
4	0
5	0
6	0
7	0

Process B

load r1, mem(0)
mul r1, r1,r1
store r1, mem(1)

r1	**17**

A Virtual Computer

Memory

0	17
1	0
2	0
3	0
4	0
5	0
6	0
7	0

Process B
load r1, mem(0)
mul r1, r1,r1
store r1, mem(1)

r1 **289**

A Virtual Computer

Memory

0	17
1	**289**
2	0
3	0
4	0
5	0
6	0
7	0

Process B

```
load  r1, mem(0)
mul   r1, r1,r1
store r1, mem(1)
```

r1	289

A Virtual Computer

Process A
```
load  r1, mem(0)
inc   r1
store r1, mem(1)
```

r1: 0

Memory

0	17
1	0
2	0
3	0
4	0
5	0
6	0
7	0

Process B
```
load  r1, mem(0)
mul   r1, r1,r1
store r1, mem(1)
```

r1: 0

A Virtual Computer

Process A

```
load  r1, mem(0)
inc   r1
store r1, mem(1)
```

r1: **18**

Memory

0	17
1	????
2	0
3	0
4	0
5	0
6	0
7	0

Process B

```
load  r1, mem(0)
mul   r1, r1,r1
store r1, mem(1)
```

r1: 289

Isolation

- How to keep processes from corrupting each other's memory?

Page Tables

Memory

0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	17

Process A

```
load  r1, mem(0)
inc   r1
store r1, mem(1)
```

r1: 0

Page Table

virt	phys
0	7
1	4

Process B

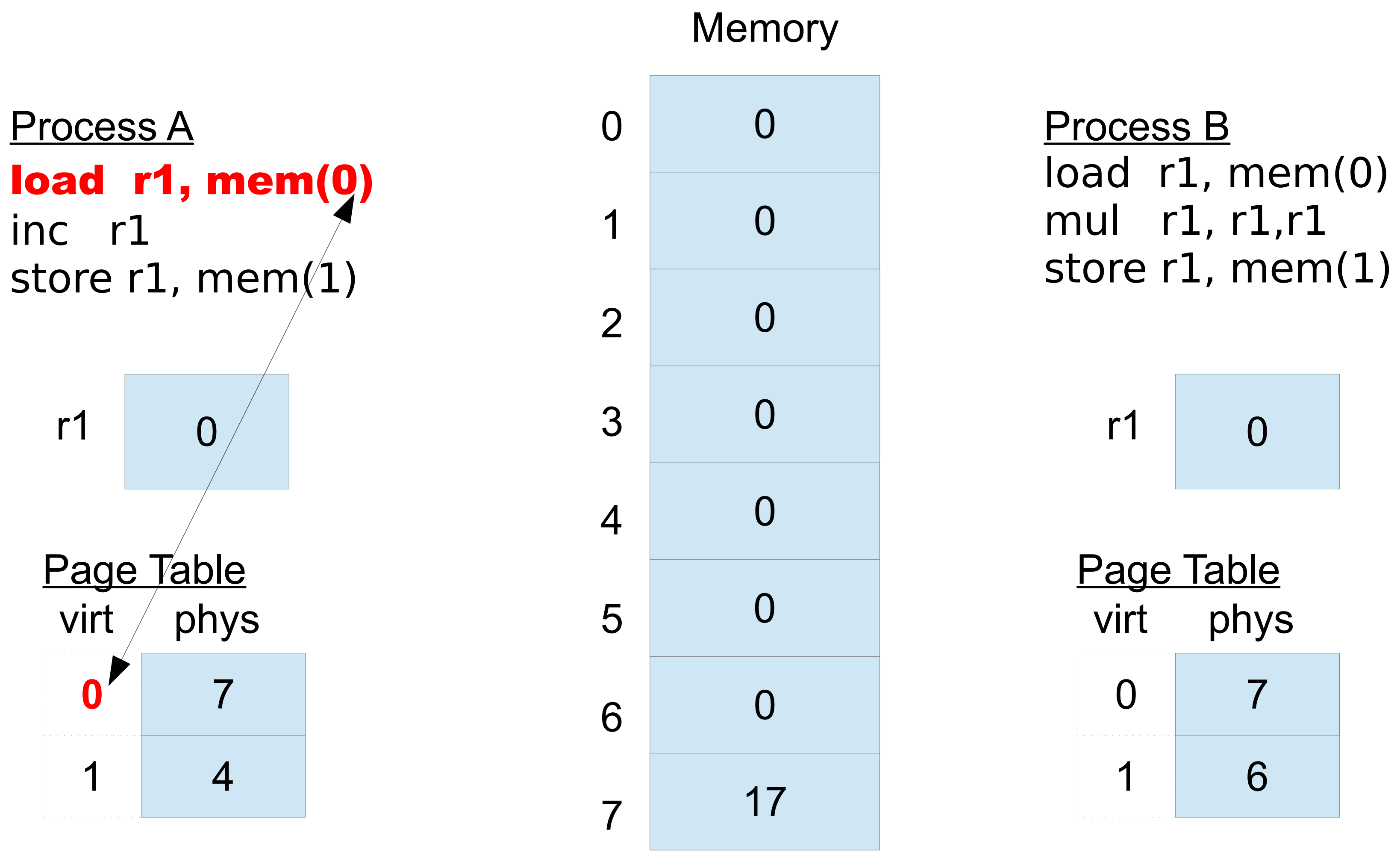
```
load  r1, mem(0)
mul   r1, r1,r1
store r1, mem(1)
```

r1: 0

Page Table

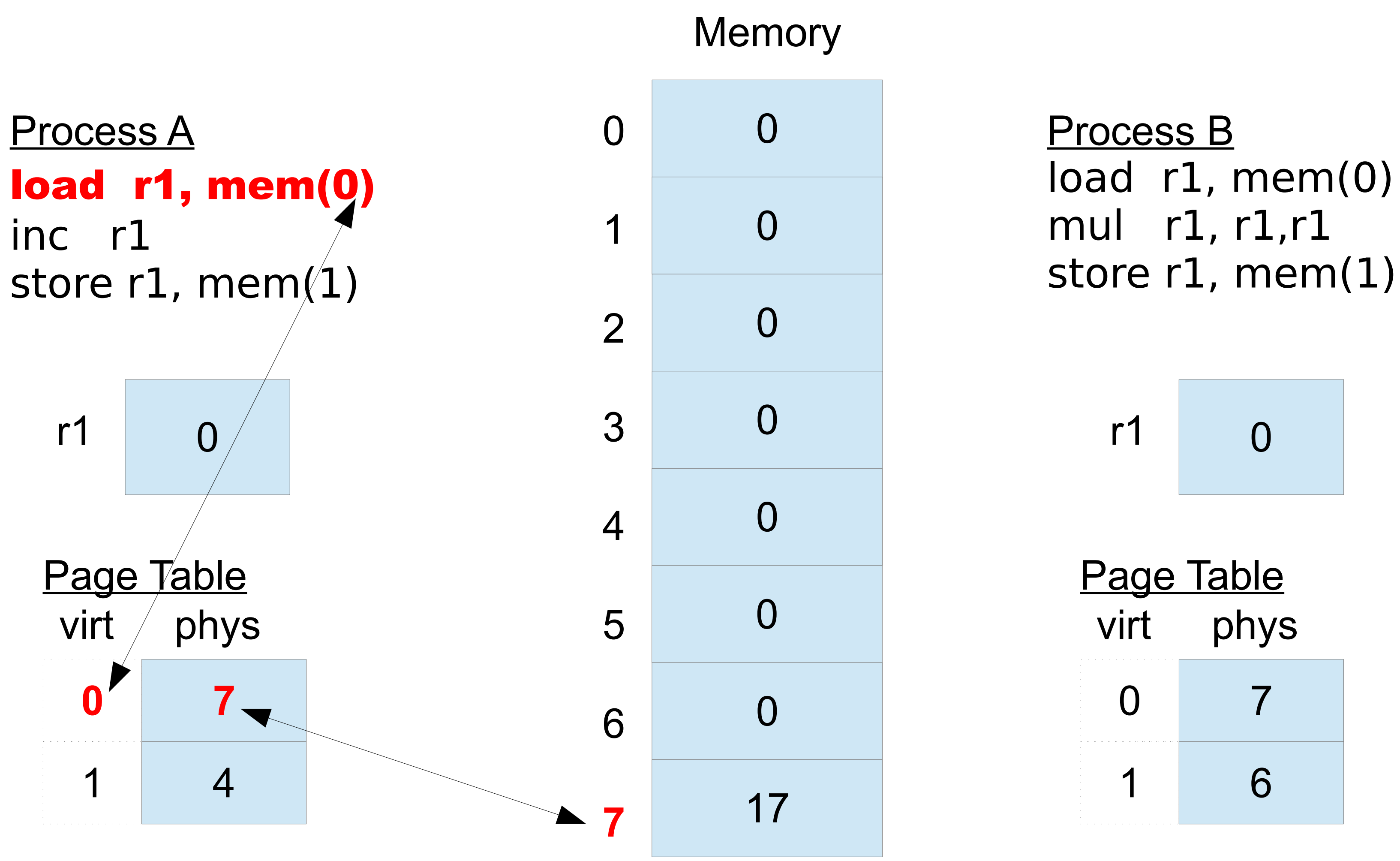
virt	phys
0	7
1	6

Page Tables

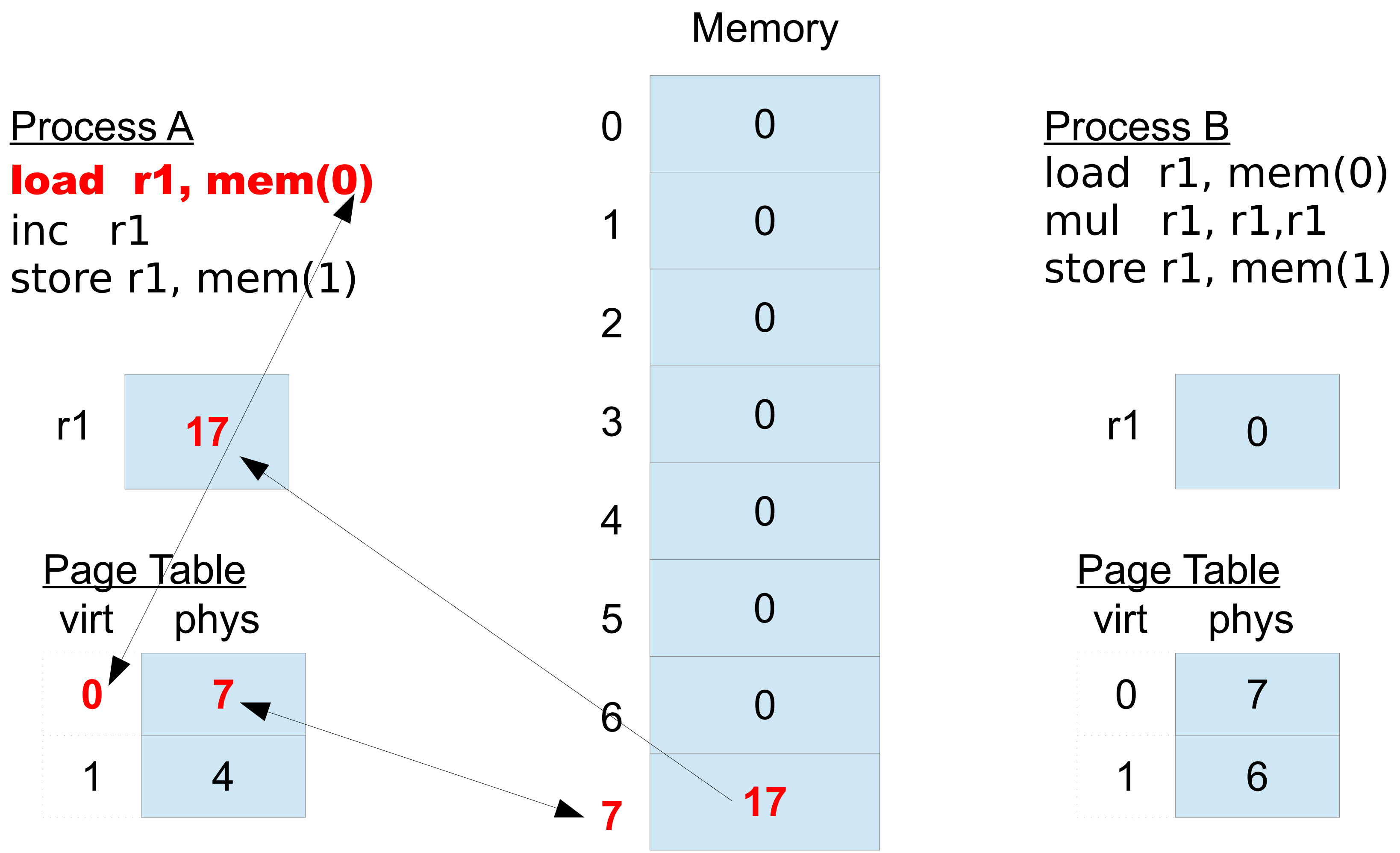


Memory
0 0
1 0
2 0
3 0
4 0
5 0
6 0
7 17
Process A
load r1, mem(0)
inc r1
store r1, mem(1)
r1 0
Page Table
virt phys
0 7
1 4
Process B
load r1, mem(0)
mul r1, r1,r1
store r1, mem(1)
r1 0
Page Table
virt phys
0 7
1 6

Page Tables


Memory
Process A
load r1, mem(0)
inc r1
store r1, mem(1)
r1
0
Page Table
virt
phys
0
7
1
4
0
1
2
3
4
5
6
7
0
0
0
0
0
0
0
17
Process B
load r1, mem(0)
mul r1, r1,r1
store r1, mem(1)
r1
0
Page Table
virt
phys
0
7
1
6

Page Tables



Memory
Process A
load r1, mem(0)
inc r1
store r1, mem(1)
r1
17
Page Table
virt phys
0 7
1 4
0 1 2 3 4 5 6 7
0 0 0 0 0 0 0 17
Process B
load r1, mem(0)
mul r1, r1,r1
store r1, mem(1)
r1
0
Page Table
virt phys
0 7
1 6

Page Tables

Memory

0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	**17**

Process A

```
load  r1, mem(0)
inc   r1
store r1, mem(1)
```

r1: 17

Page Table

virt	phys
0	7
1	4

Process B

```
load  r1, mem(0)
mul   r1, r1,r1
store r1, mem(1)
```

r1: **17**

Page Table

virt	phys
0	**7**
1	6

Page Tables

Memory

0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	17

Process A

load r1, mem(0)
inc r1
store r1, mem(1)

r1 **18**

Page Table

virt	phys
0	7
1	4

Process B

load r1, mem(0)
mul r1, r1,r1
store r1, mem(1)

r1 **289**

Page Table

virt	phys
0	7
1	6

Page Tables

Process A

```
load  r1, mem(0)
inc   r1
store r1, mem(1)
```

r1: 18

Page Table

virt	phys
0	7
1	4

Memory

0	0
1	0
2	0
3	0
4	18
5	0
6	0
7	17

Process B

```
load  r1, mem(0)
mul   r1, r1,r1
store r1, mem(1)
```

r1: 289

Page Table

virt	phys
0	7
1	6

Page Tables

<u>Process A</u>

```
load  r1, mem(0)
inc   r1
store r1, mem(1)
```

r1: 18

<u>Page Table</u>

virt	phys
0	7
1	4

Memory

0	0
1	0
2	0
3	0
4	18
5	0
6	**289**
7	17

<u>Process B</u>

load r1, mem(0)
mul   r1, r1,r1
store r1, mem(1)

r1: 289

<u>Page Table</u>

virt	phys
0	7
1	**6**

Virtual Address Space

- What memory can each process see?

Virtual Address Space

Virtual Memory

Process A
load r1, mem(0)
inc r1
store r1, mem(1)

17	**0**	**17**
18	**1**	**289**

Process B
load r1, mem(0)
mul r1, r1,r1
store r1, mem(1)

r1 18

r1 289

The Flat Address Model

- Every process has its own isolated memory
- Every address is usable
  - No holes
  - No limits
- App/OS decide what addresses are in use
  - Things can go **anywhere!**

Implementation

- Must be fast!
- Must be flexible!

Implementation

- Must be fast!
- Must be flexible!

- Implemented by CPU, configured by OS

    (page tables)

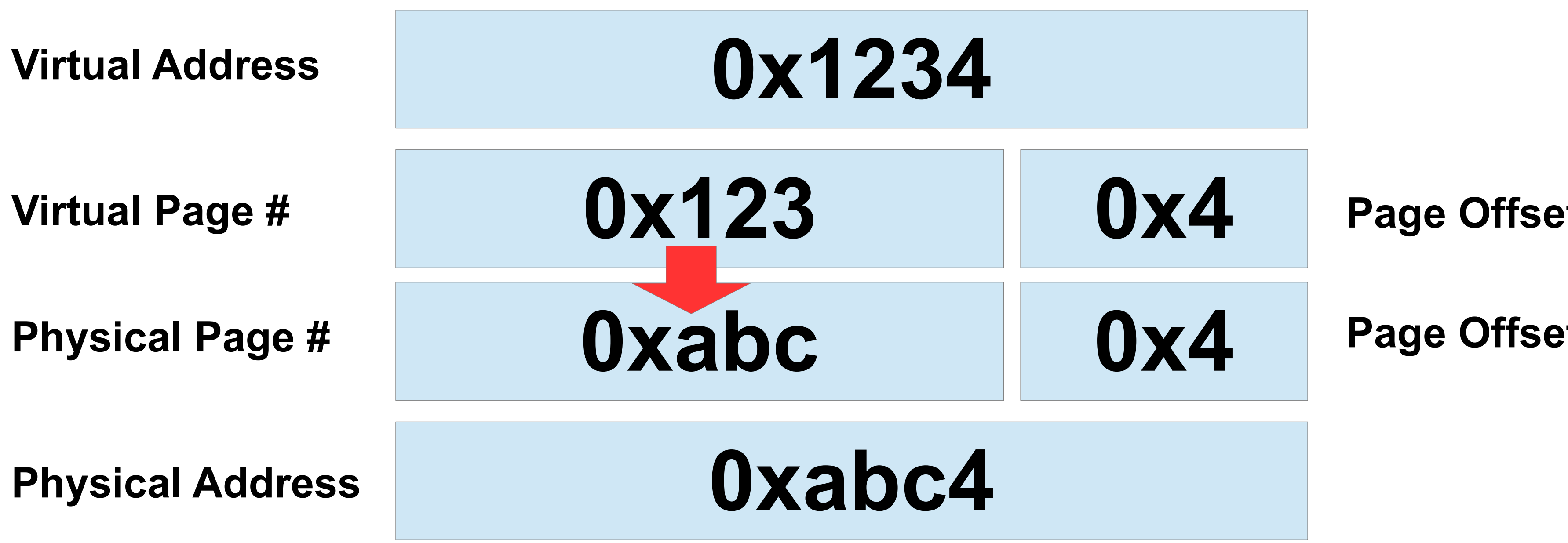
- CPU asks OS what to do when it's confused

    (page fault)

The Mechanics of Translation

<u>Process A</u>

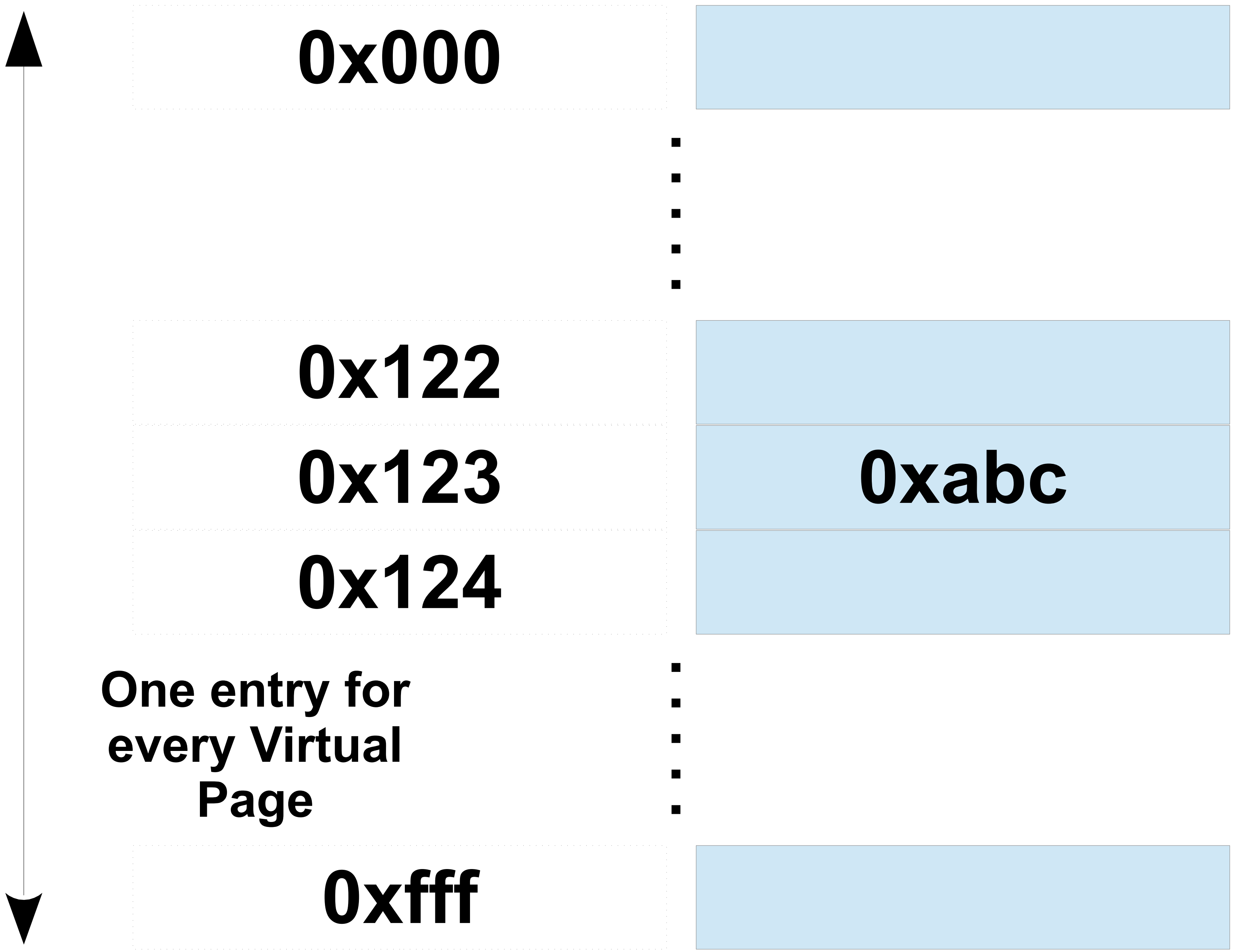
```
load  r1, mem(0x1234)
inc   r1
store r1, mem(0x1235)
```



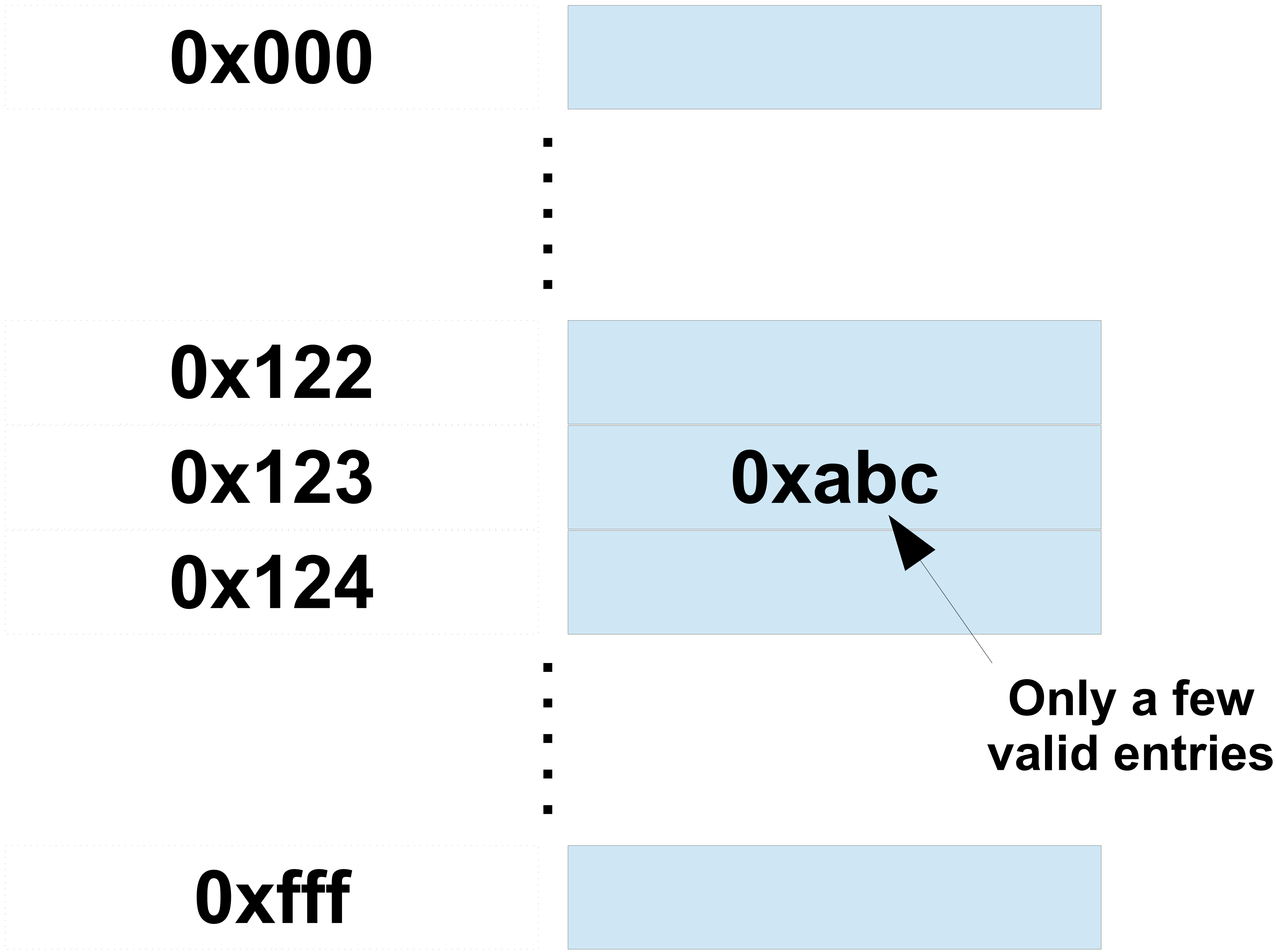
The Page Table

0x000	
⋮	
0x122	
0x123	0xabc
0x124	
⋮	
0xfff	

The Page Table



The Page Table

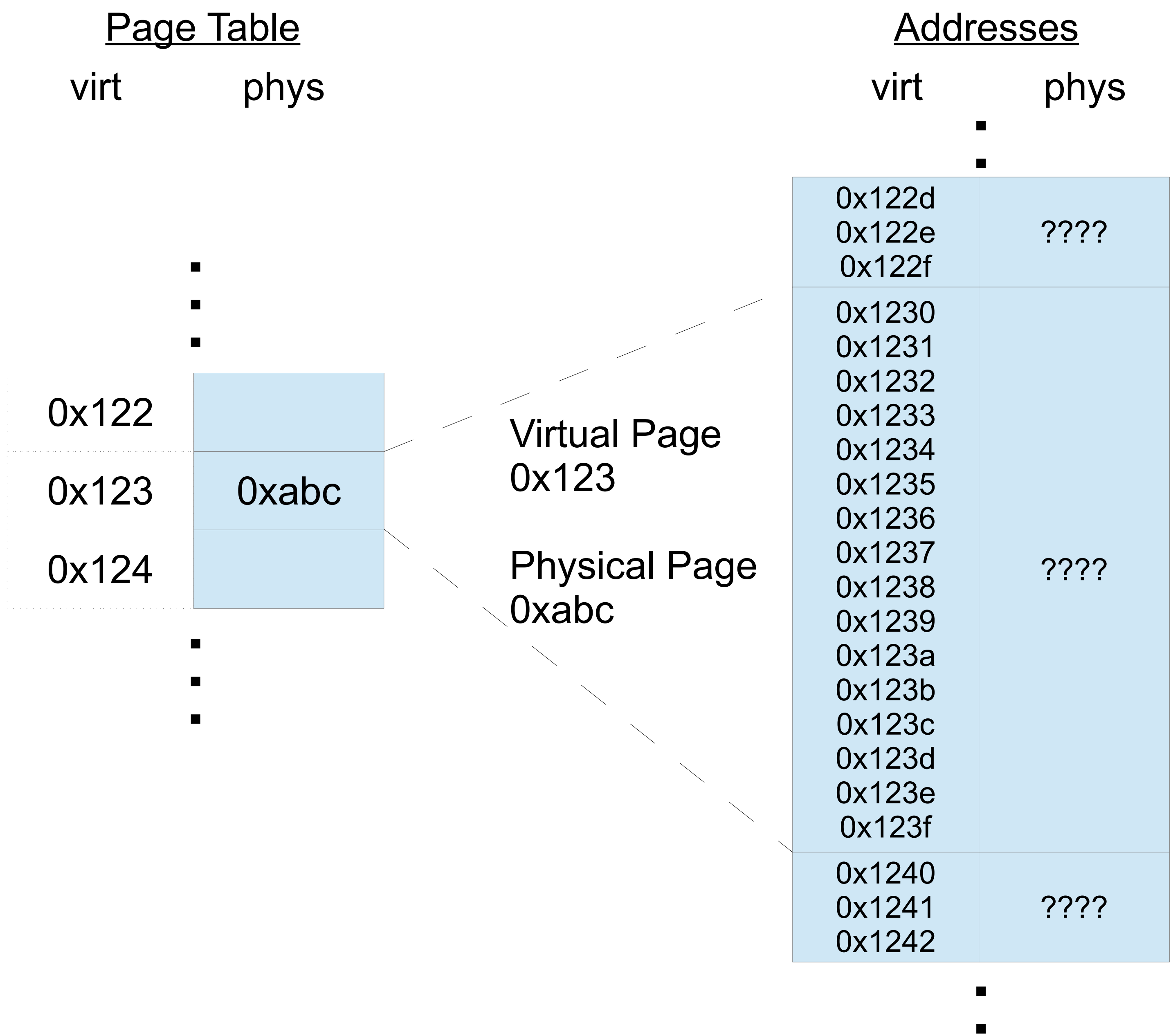


The Mechanics of Translation

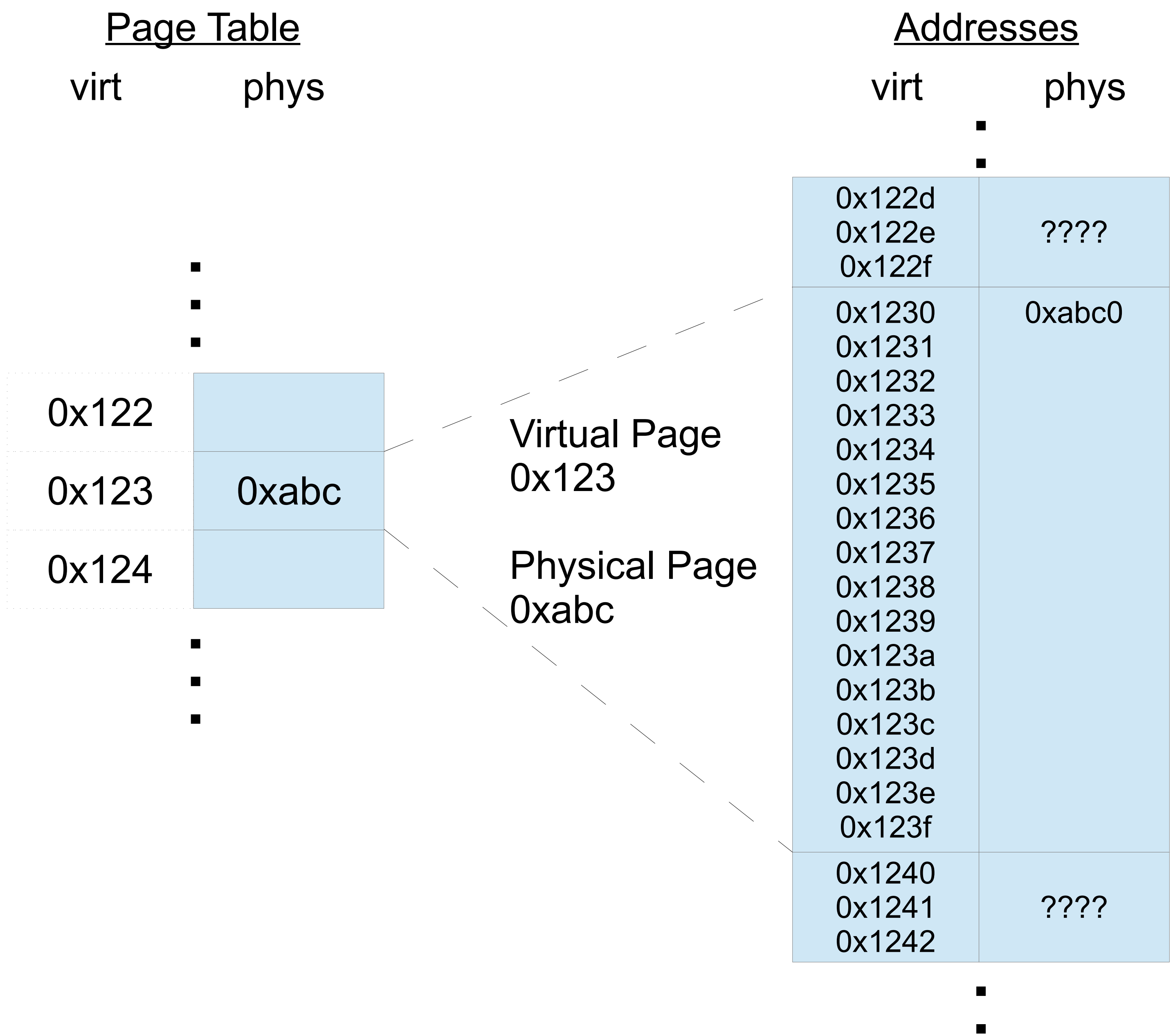
Page Table

virt	phys
	⋮
0x122	
0x123	0xabc
0x124	
	⋮

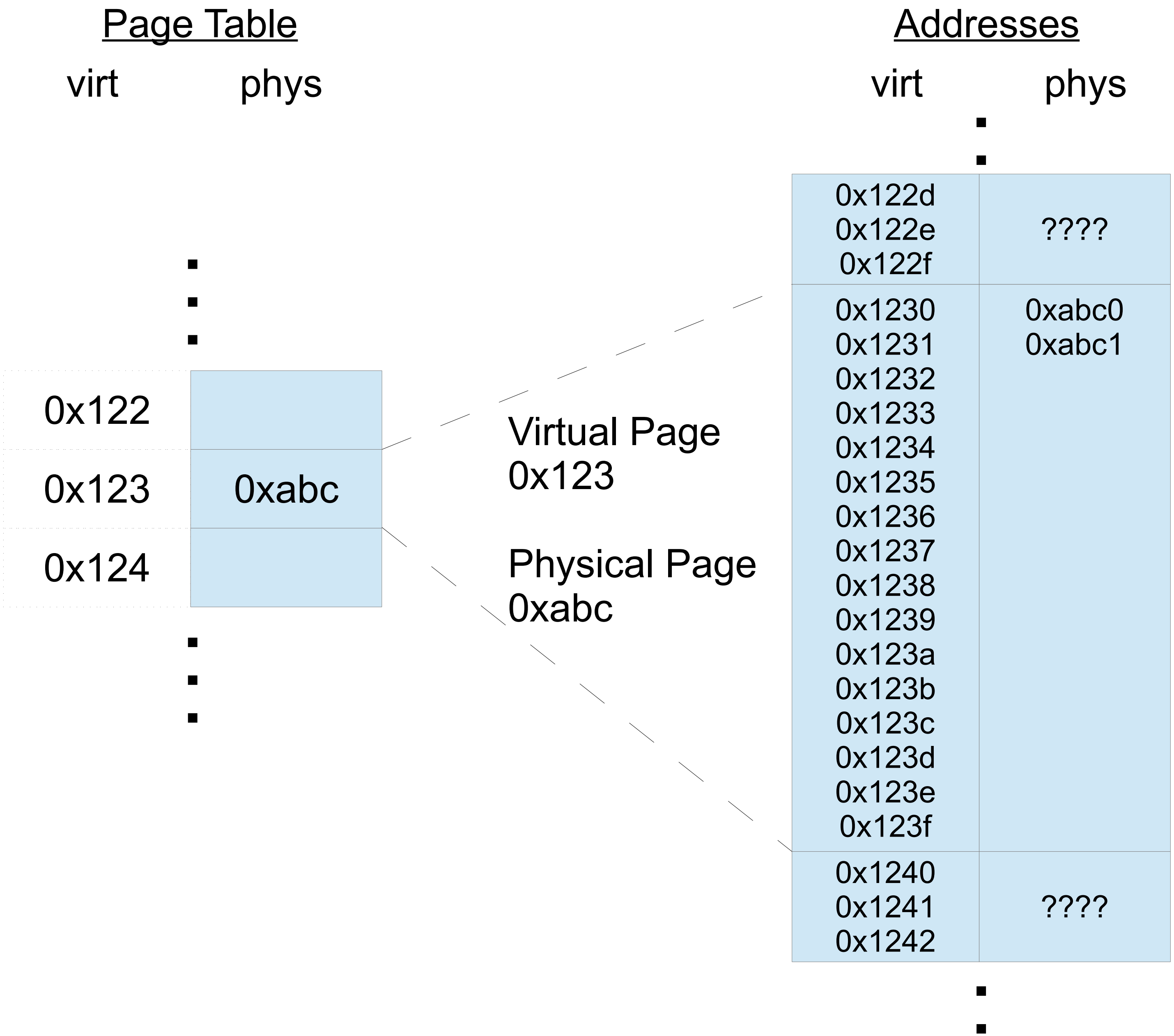
The Mechanics of Translation



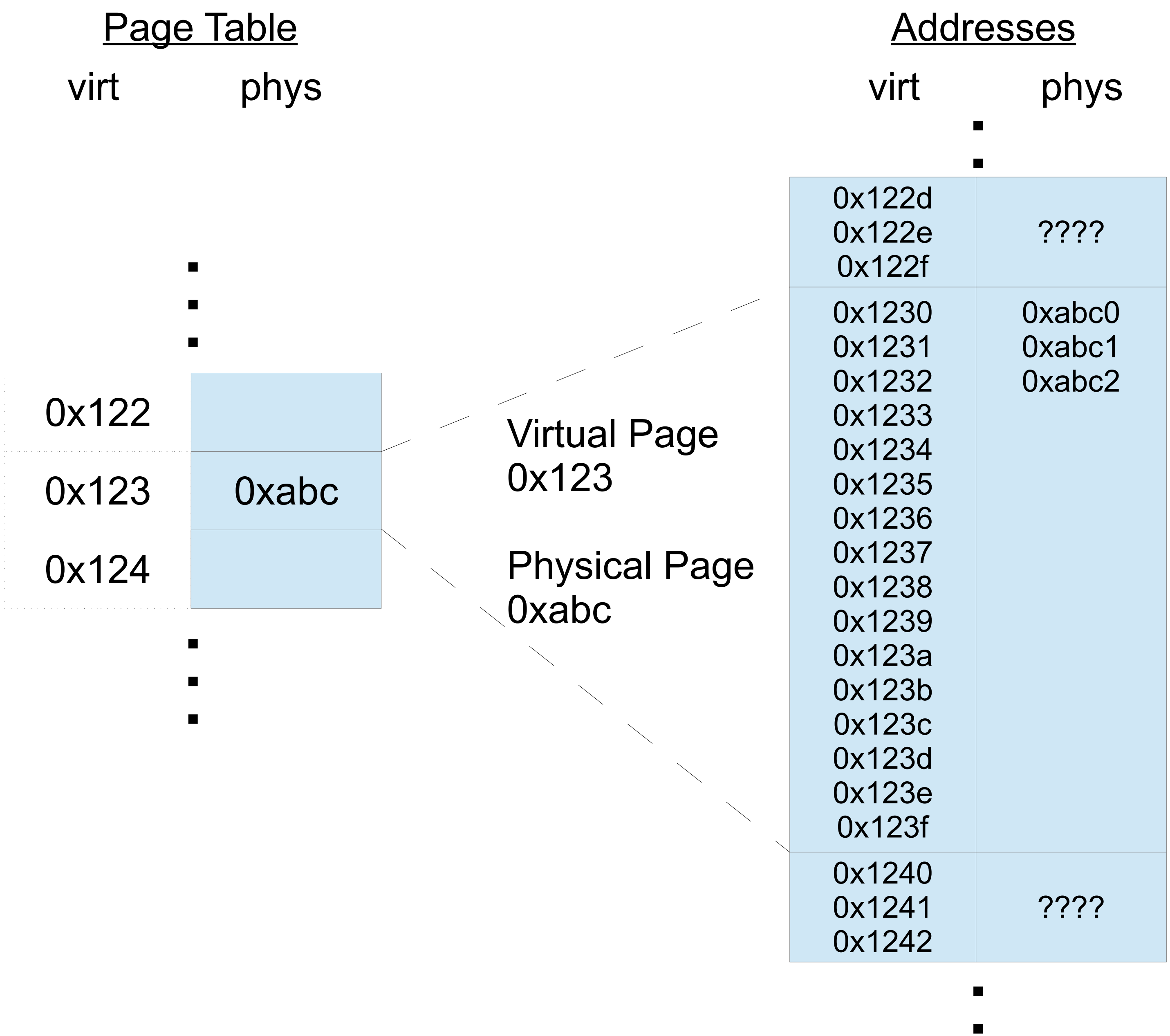
The Mechanics of Translation



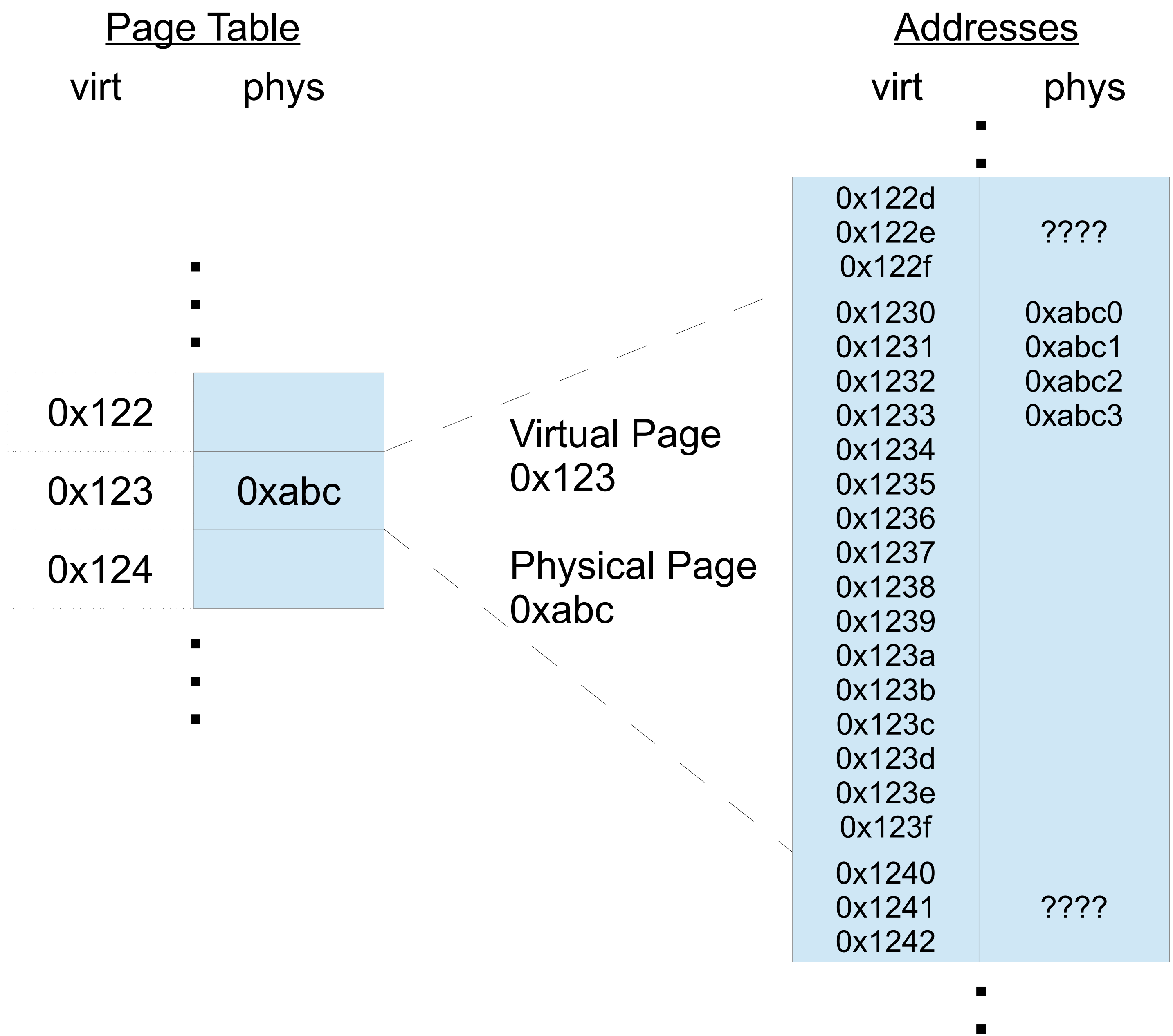
The Mechanics of Translation



The Mechanics of Translation



The Mechanics of Translation



The Mechanics of Translation

Page Table

virt	phys
0x122	
0x123	0xabc
0x124	

Virtual Page
0x123

Physical Page
0xabc

Addresses

virt	phys
0x122d	
0x122e	
0x122f	
0x1230	0xabc0
0x1231	0xabc1
0x1232	0xabc2
0x1233	0xabc3
0x1234	0xabc4
0x1235	0xabc5
0x1236	0xabc6
0x1237	0xabc7
0x1238	0xabc8
0x1239	0xabc9
0x123a	0xabca
0x123b	0xabcb
0x123c	0xabcc
0x123d	0xabcd
0x123e	0xabce
0x123f	0xabcf
0x1240	
0x1241	
0x1242	

The Mechanics of Translation

<u>Page Table</u>

virt	phys
0x122	
0x123	0xabc
0x124	0xdef

Virtual Page
0x123

Physical Page
0xabc

<u>Addresses</u>

virt	phys
0x122d	
0x122e	
0x122f	
0x1230	0xabc0
0x1231	0xabc1
0x1232	0xabc2
0x1233	0xabc3
0x1234	0xabc4
0x1235	0xabc5
0x1236	0xabc6
0x1237	0xabc7
0x1238	0xabc8
0x1239	0xabc9
0x123a	0xabca
0x123b	0xabcb
0x123c	0xabcc
0x123d	0xabcd
0x123e	0xabce
0x123f	0xabcf
0x1240	0xdef0
0x1241	0xdef1
0x1242	0xdef2

The Mechanics of Translation

<u>Page Table</u>

virt	phys
0x122	
0x123	0xabc
0x124	0xdef

<u>Addresses</u>

	virt	phys
	0x122d	
	0x122e	
	0x122f	
	0x1230	0xabc0
	0x1231	0xabc1
	0x1232	0xabc2
	0x1233	0xabc3
	0x1234	0xabc4
	0x1235	0xabc5
	0x1236	0xabc6
	0x1237	0xabc7
	0x1238	0xabc8
	0x1239	0xabc9
	0x123a	0xabca
	0x123b	0xabcb
	0x123c	0xabcc
	0x123d	0xabcd
H	0x123e	0xabce
e	0x123f	0xabcf
l	0x1240	0xdef0
l	0x1241	0xdef1
o	0x1242	0xdef2

What is a Page?

- Fixed-size, aligned section of virtual memory
- All bytes in the same virtual page **MUST** be in the same physical page
- Always power of 2 in size
  - Power of 2 = simple hardware = fast

How Big is a Page?

- 4K is common (Intel, PowerPC)
- 8K is rare (SPARC)

- “Huge Pages” sometimes also supported
  - Larger pages = smaller page table
  - Can be mixed with ordinary 4K pages

The Page Table

0x000	
⋮	⋮
0x122	
0x123	0xabc
0x124	
⋮	⋮
0xfff	

A Page Table Entry

1	**0xabc**	**rw**
Valid Bit	**Physical Page #**	**Permissions**

Page Fault

- Program accesses invalid virtual memory
  - Page table entry not valid, or permissions don't allow operation
- CPU interrupts program, forces into OS
- OS decides what to do

Page Fault: Demand Paging

Process A

```
load  r1, mem(0)
inc   r1
store r1, mem(1)
```

r1	
r1	0

Page Table

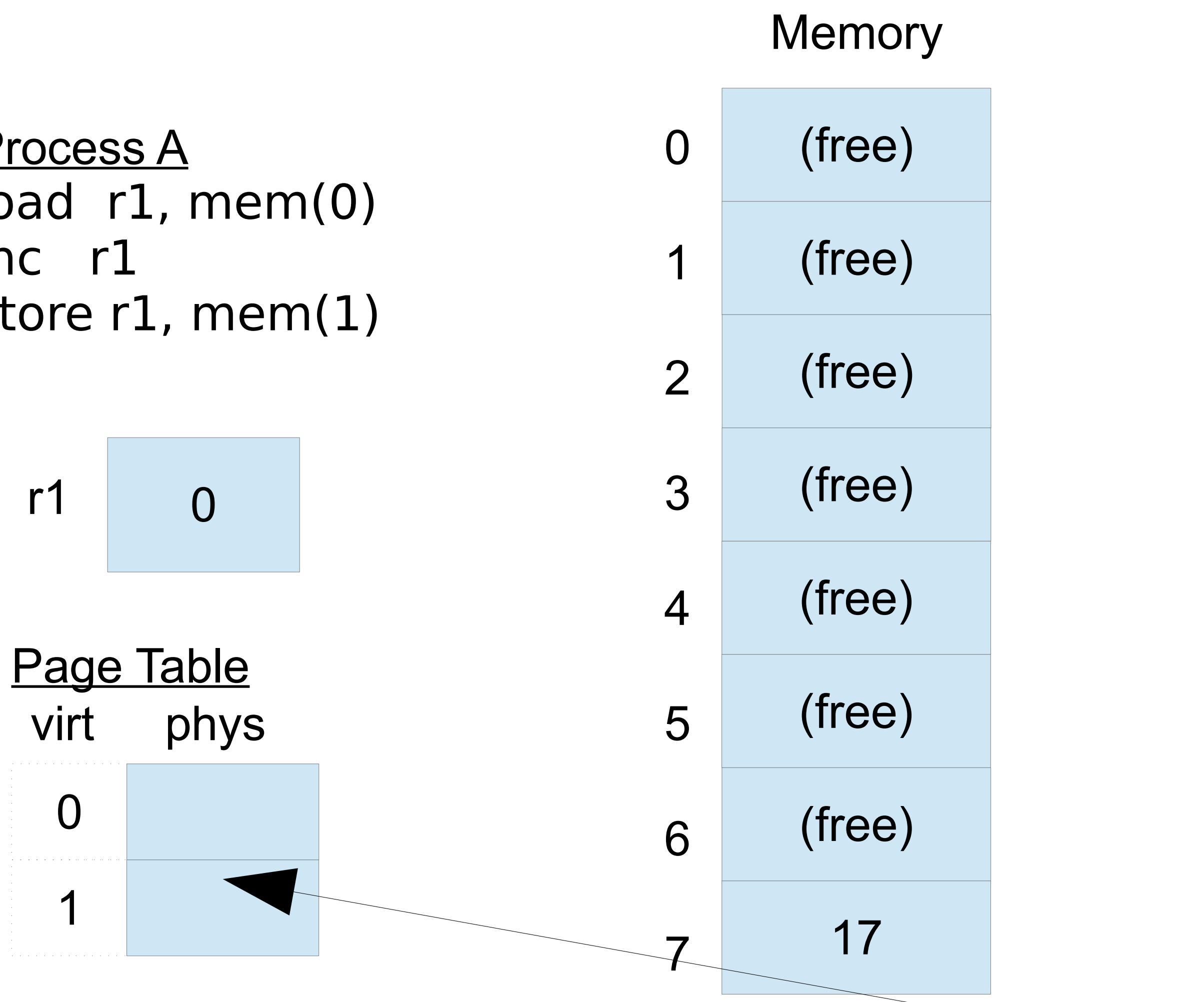
virt	phys
0	
1	

Memory

0	(free)
1	(free)
2	(free)
3	(free)
4	(free)
5	(free)
6	(free)
7	17

Page Fault: Demand Paging

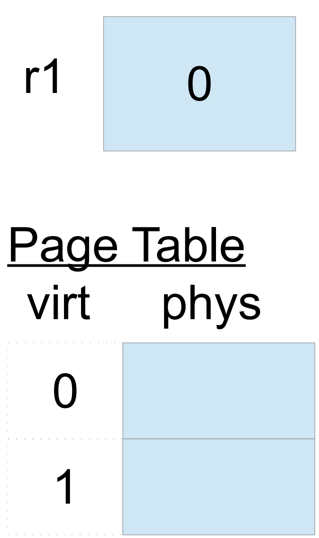
<u>Process A</u>
load r1, mem(0)
inc   r1
store r1, mem(1)

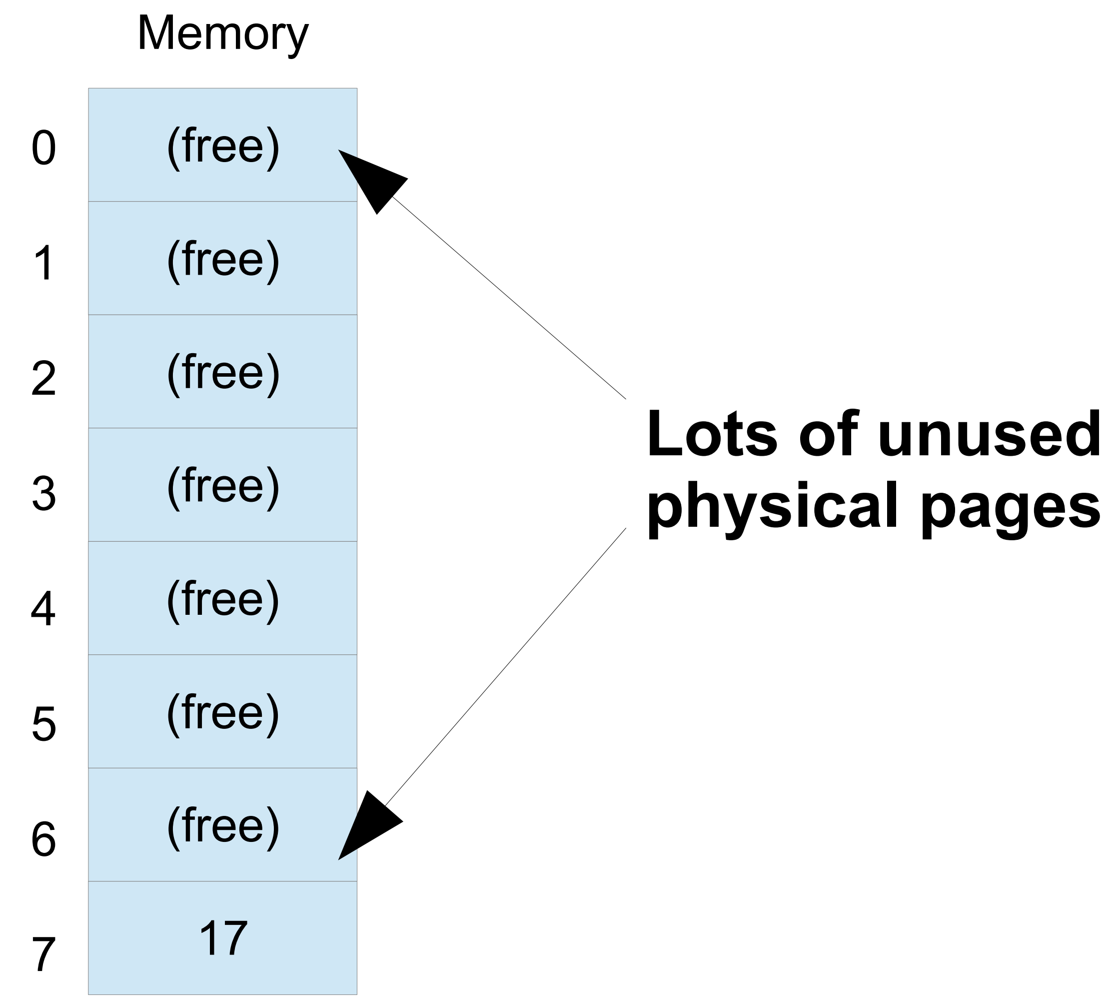


No valid page table entries (yet)

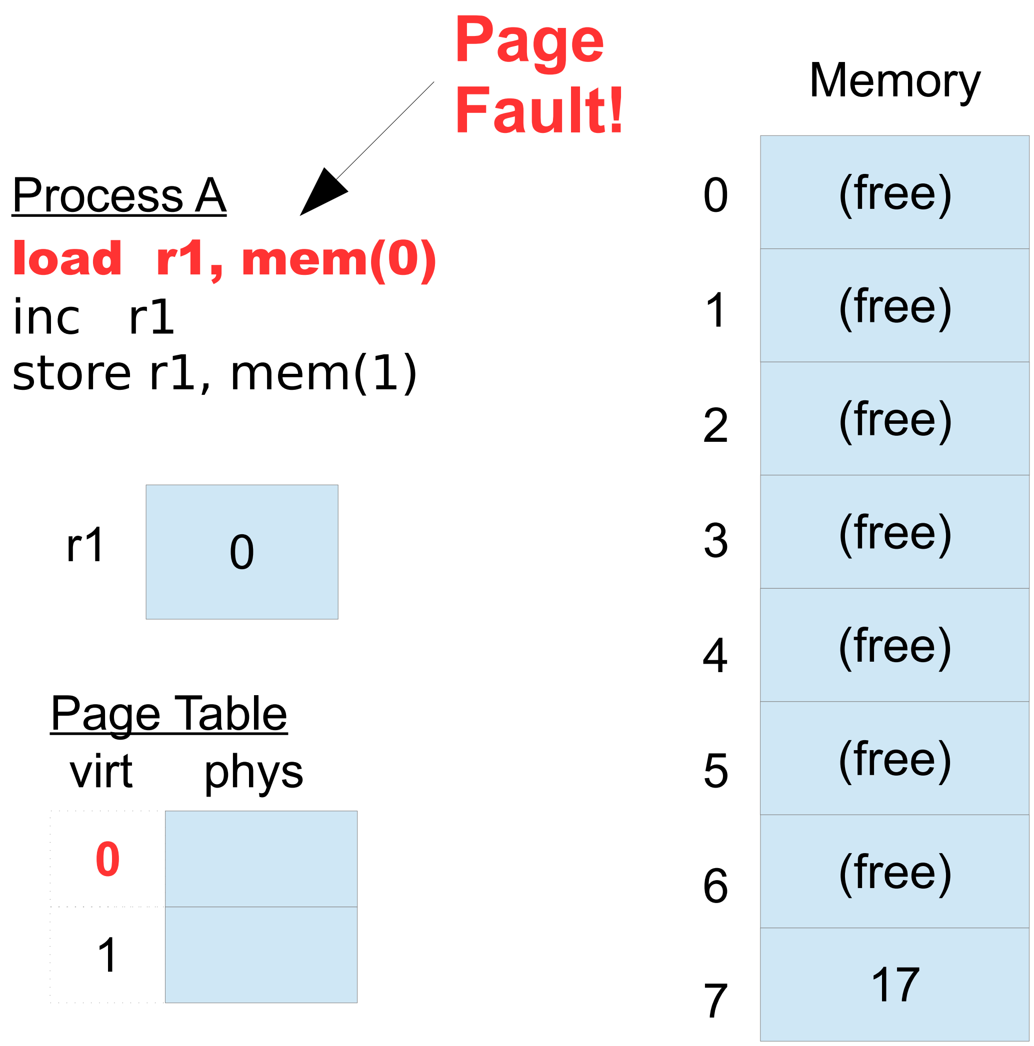
Page Fault: Demand Paging

Process A
```
load  r1, mem(0)
inc   r1
store r1, mem(1)
```





Page Fault: Demand Paging



Page Fault: Demand Paging

Process A

```
load  r1, mem(0)
inc   r1
store r1, mem(1)
```

r1: 0

Page Table

virt	phys
0	7
1	

Add Entry

Memory

0	(free)
1	(free)
2	(free)
3	(free)
4	(free)
5	(free)
6	(free)
7	17

Page Fault: Demand Paging

Retry

Process A

load r1, mem(0)
inc r1
store r1, mem(1)

r1	0

Page Table

virt	phys
0	**7**
1	

Memory

0	(free)
1	(free)
2	(free)
3	(free)
4	(free)
5	(free)
6	(free)
7	17

Page Fault: Demand Paging

Process A

load r1, mem(0)
inc r1
store r1, mem(1)

r1: **17** — **Success!**

Page Table

virt	phys
0	**7**
1	

Memory

0	(free)
1	(free)
2	(free)
3	(free)
4	(free)
5	(free)
6	(free)
7	17

Page Fault: Demand Paging

Process A
load r1, mem(0)
inc r1
store r1, mem(1)

Page Fault!

r1: 18

Page Table

virt	phys
0	7
1	

Memory

0	(free)
1	(free)
2	(free)
3	(free)
4	(free)
5	(free)
6	(free)
7	17

Page Fault: Demand Paging

Process A
```
load  r1, mem(0)
inc   r1
store r1, mem(1)
```

r1: 18

Page Table

virt	phys
0	7
1	4

Memory

0	(free)
1	(free)
2	(free)
3	(free)
4	0
5	(free)
6	(free)
7	17

Allocate memory

Page Fault: Demand Paging

Retry

Process A

```
load  r1, mem(0)
inc   r1
store r1, mem(1)
```

r1: 18

Page Table

virt	phys
0	7
1	4

Memory

0	(free)
1	(free)
2	(free)
3	(free)
4	0
5	(free)
6	(free)
7	17

Page Fault: Demand Paging

Process A

```
load  r1, mem(0)
inc   r1
store r1, mem(1)
```

r1: 18

Page Table

virt	phys
0	7
1	4

Memory

0	(free)
1	(free)
2	(free)
3	(free)
4	18
5	(free)
6	(free)
7	17

Success!

Demand Paging: Summary

- Valid virtual address, but page table entry is not filled in
- Page fault on first access to the page
- OS verifies that address is valid, then fills in page table entry
  - May need to allocate & init a page
- Retry operation

Virtual Address Space

Virtual Memory

<u>Process A</u>

```
load  r1, mem(0)
inc   r1
store r1, mem(1)
```

17	0	17
18	1	289

<u>Process B</u>

```
load  r1, mem(0)
mul   r1, r1,r1
store r1, mem(1)
```

r1 18

r1 289

Page Fault: Copy-On-Write (COW)

Memory

0	(free)
1	(free)
2	(free)
3	(free)
4	18
5	(free)
6	289
7	17

Process A
...code...

Page Table

virt	phys
0	7 (ro)
1	4

Process B
```
set   r1, 19
store r1, mem(0)
```

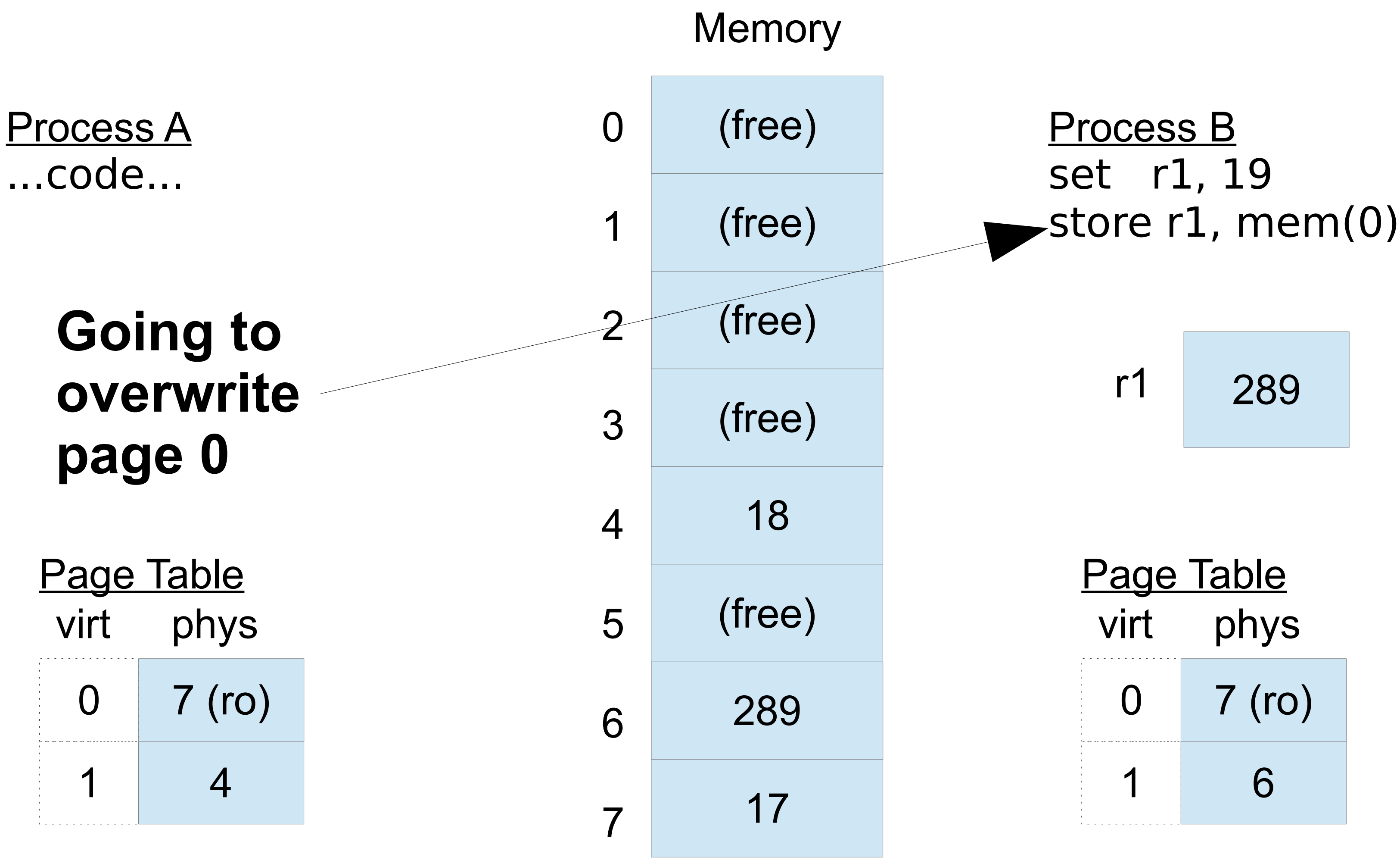
r1 | 289

Page Table

virt	phys
0	7 (ro)
1	6

Shared physical page.
Both marked read-only.

Page Fault: Copy-On-Write (COW)



Page Fault: Copy-On-Write (COW)

<u>Process A</u>
...code...

Memory

0	(free)
1	(free)
2	(free)
3	(free)
4	18
5	(free)
6	289
7	17

<u>Process B</u>
```
set   r1, 19
store r1, mem(0)
```

Page Fault!

r1: 19

<u>Page Table</u> (Process A)

virt	phys
0	7 (ro)
1	4

<u>Page Table</u> (Process B)

virt	phys
0	**7 (ro)**
1	6

Page Fault: Copy-On-Write (COW)

Process A
...code...

Allocate page, copy data

Page Table

virt	phys
0	7 (ro)
1	4

Memory

0	(free)
1	(free)
2	(free)
3	(free)
4	18
5	**17**
6	289
7	17

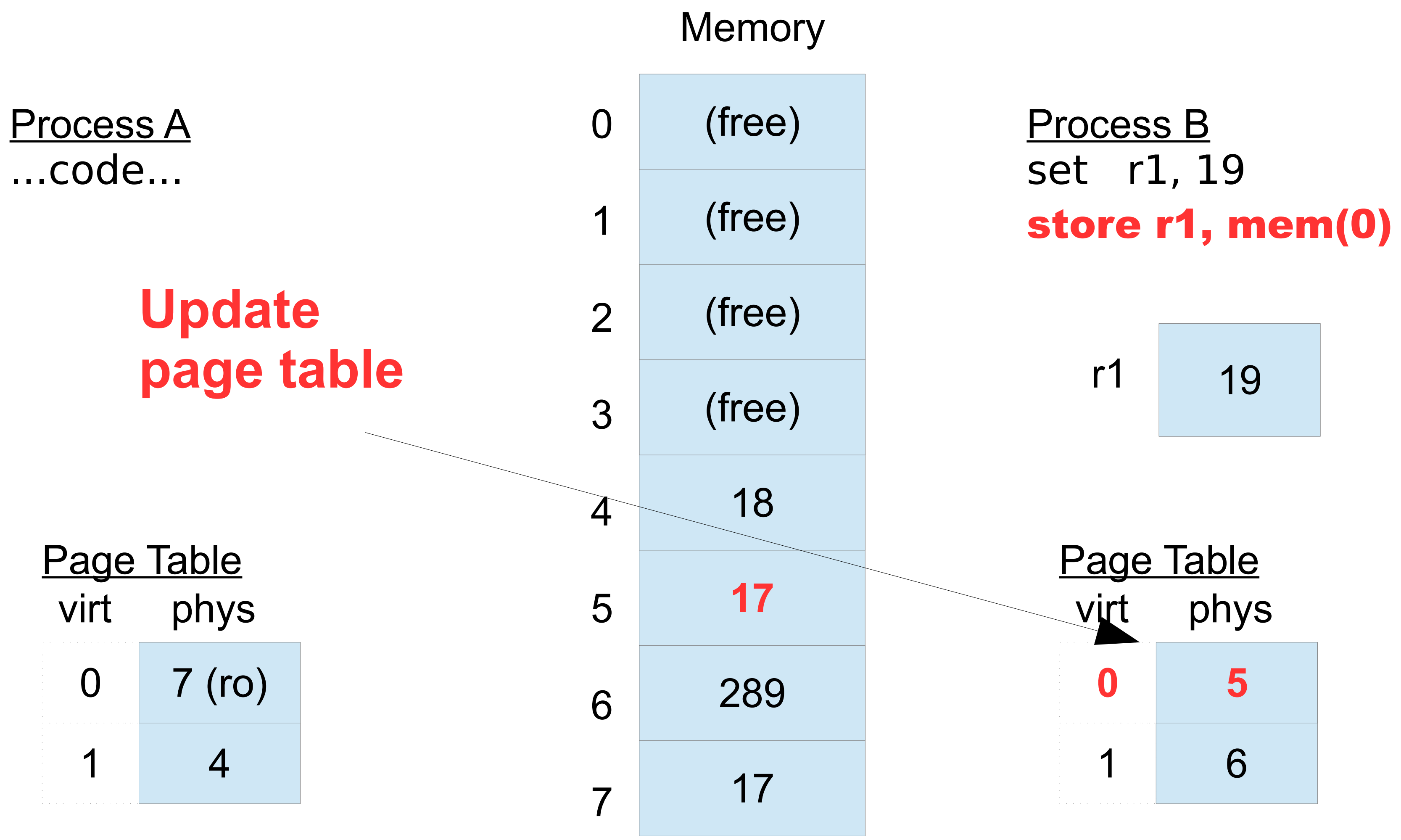
Process B
set r1, 19
store r1, mem(0)

r1	19

Page Table

virt	phys
0	7 (ro)
1	6

Page Fault: Copy-On-Write (COW)



Page Fault: Copy-On-Write (COW)

Memory

0	(free)
1	(free)
2	(free)
3	(free)
4	18
5	17
6	289
7	17

Process A
...code...

Page Table

virt	phys
0	7 (ro)
1	4

Process B
```
set   r1, 19
store r1, mem(0)
```

Retry

r1: 19

Page Table

virt	phys
0	5
1	6

Page Fault: Copy-On-Write (COW)

COW: Summary

- Single physical page
- Lots of page table entries point to it
  - All read-only
- Page fault on first write
  - Allocate new page
  - Copy data
  - Update page table
- Retry operation

Other things to think about:

- How to implement fork() ?
- How to implement a swap file?
- How to implement memory-mapped files?
- How to implement shared memory?
- How to implement pages filled with zeroes?

Other things to think about:

- Policy questions:
  - How much physical memory should a process be allowed to consume?
  - How much virtual memory should a process be allowed to allocate?

Other things to think about:

- How do we reduce the size of the page table?
- How do we cache page table entries?
  - And when does the cache need to be flushed?
- When, if ever, can two processes share the same page table?
  - And how do you handle cache flushing if they do?