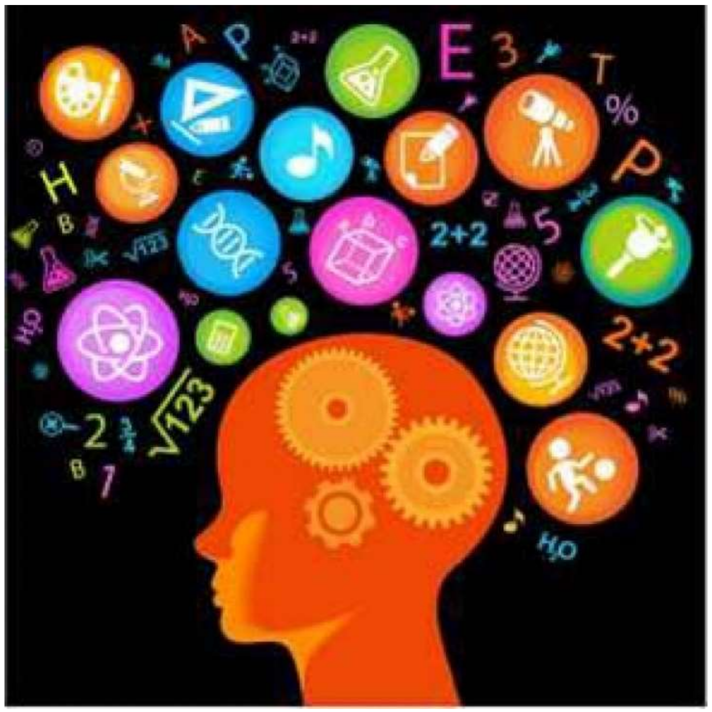
Memory Management
Part -2



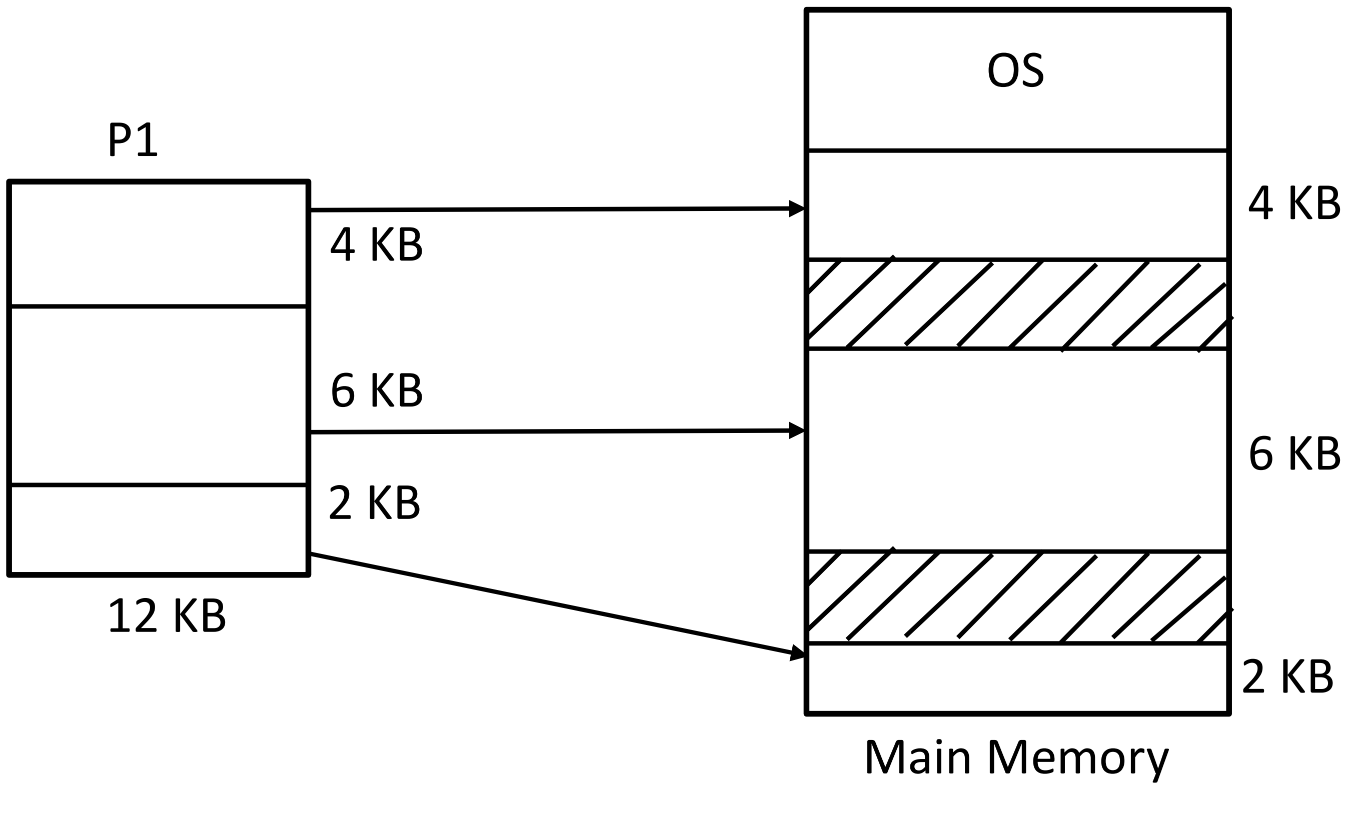
Prachi Pandey
C-DAC Bangalore
prachip@cdac.in

Topics

- Non contiguous Memory Allocation
  - Segmentation
    - About segmentation
    - HW required
    - Segment table
  - Paging
    - Paging and page table
    - TLB
    - Dirty bit
    - Shared pages and reentrant code
  - Shared pages and reentrant code

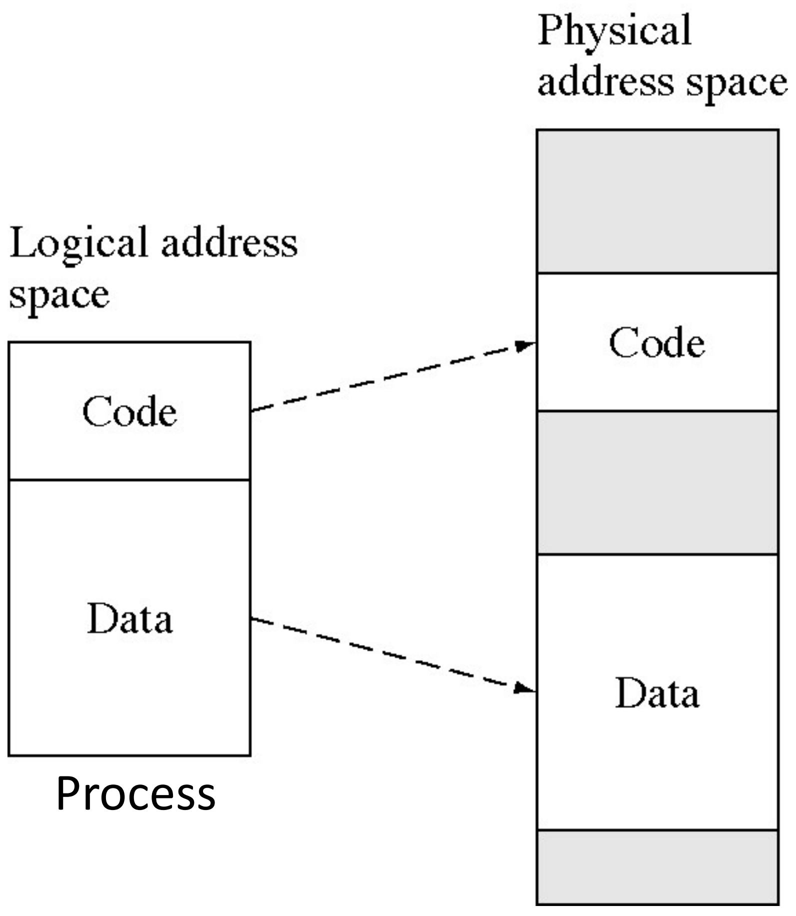
Non contiguous memory allocation

- Without memory mapping, programs require physical continuous memory
- Large blocks mean large fragments and wasted memory



Non-contiguous Memory allocation

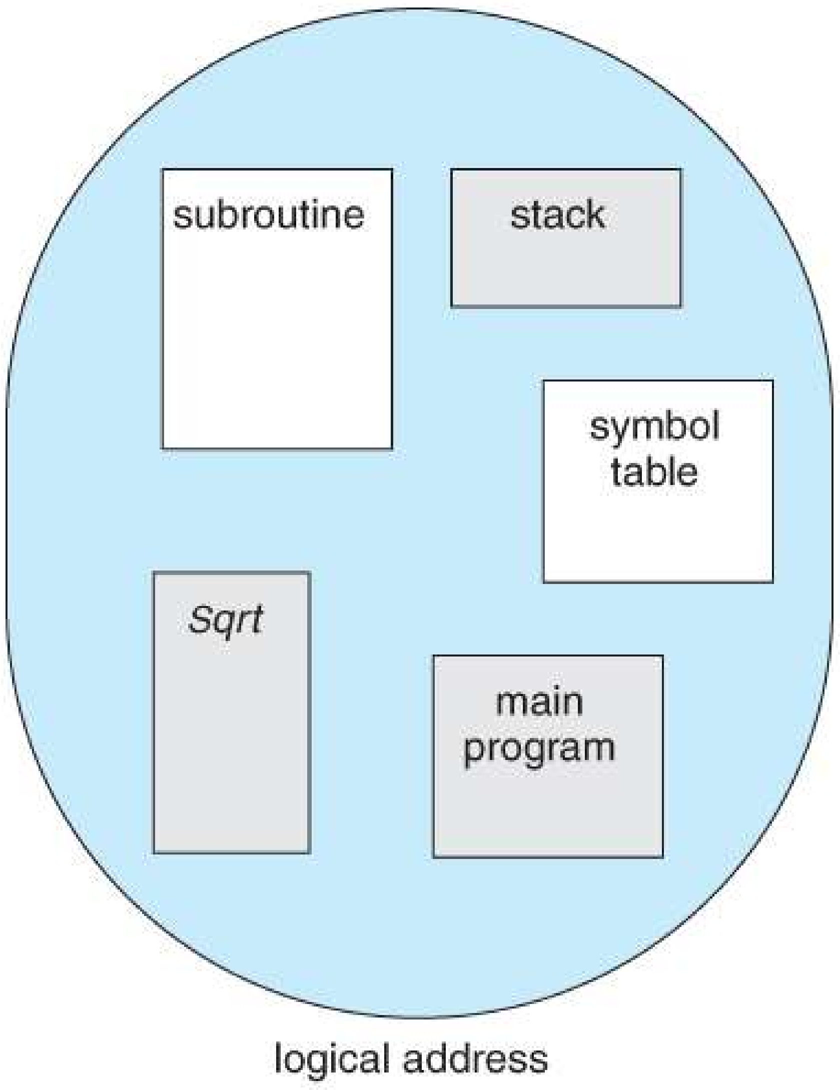
- We need hardware memory mapping to address this problem
  - Segments
  - Pages



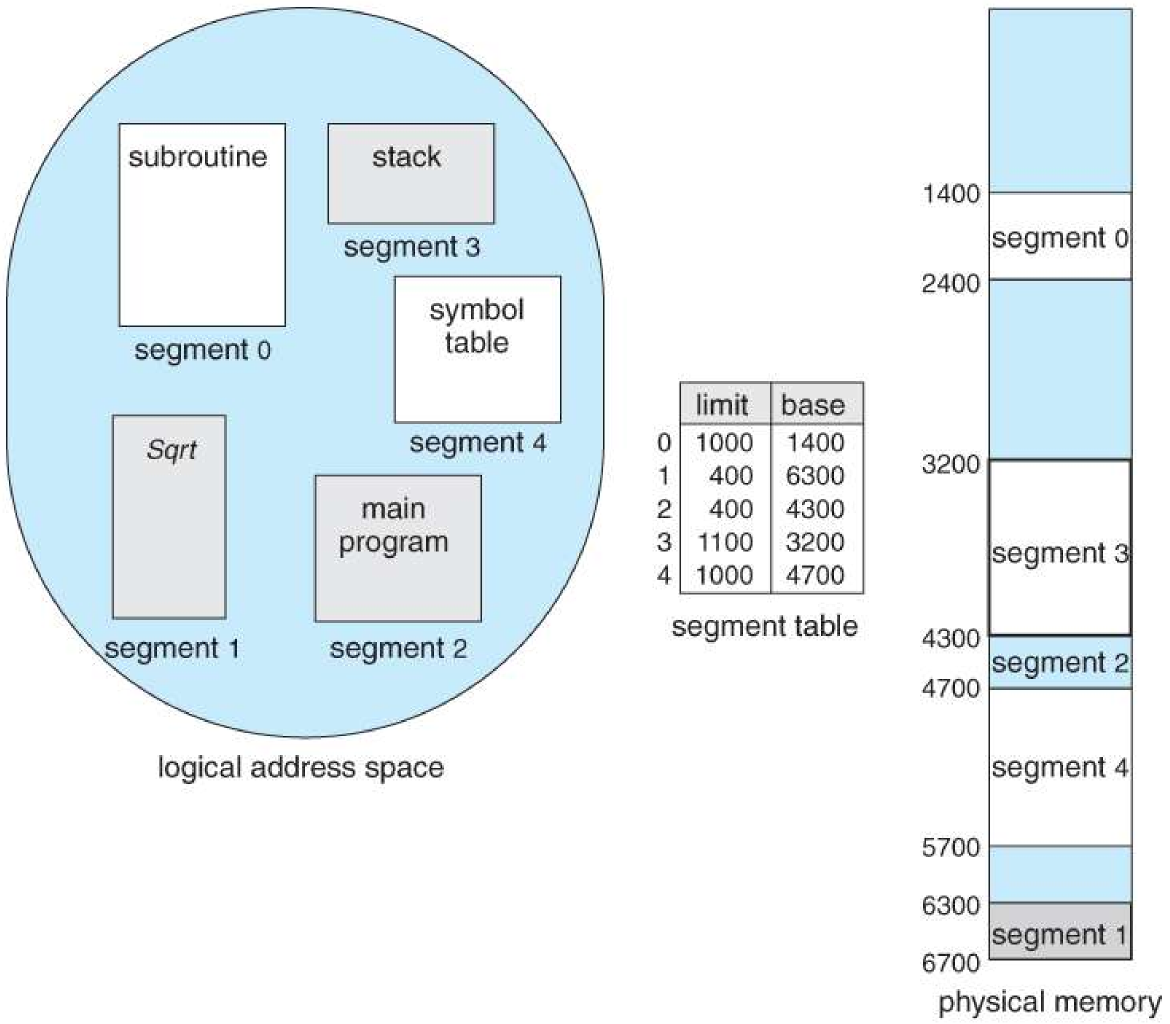
Separate code and data spaces

Segmentation

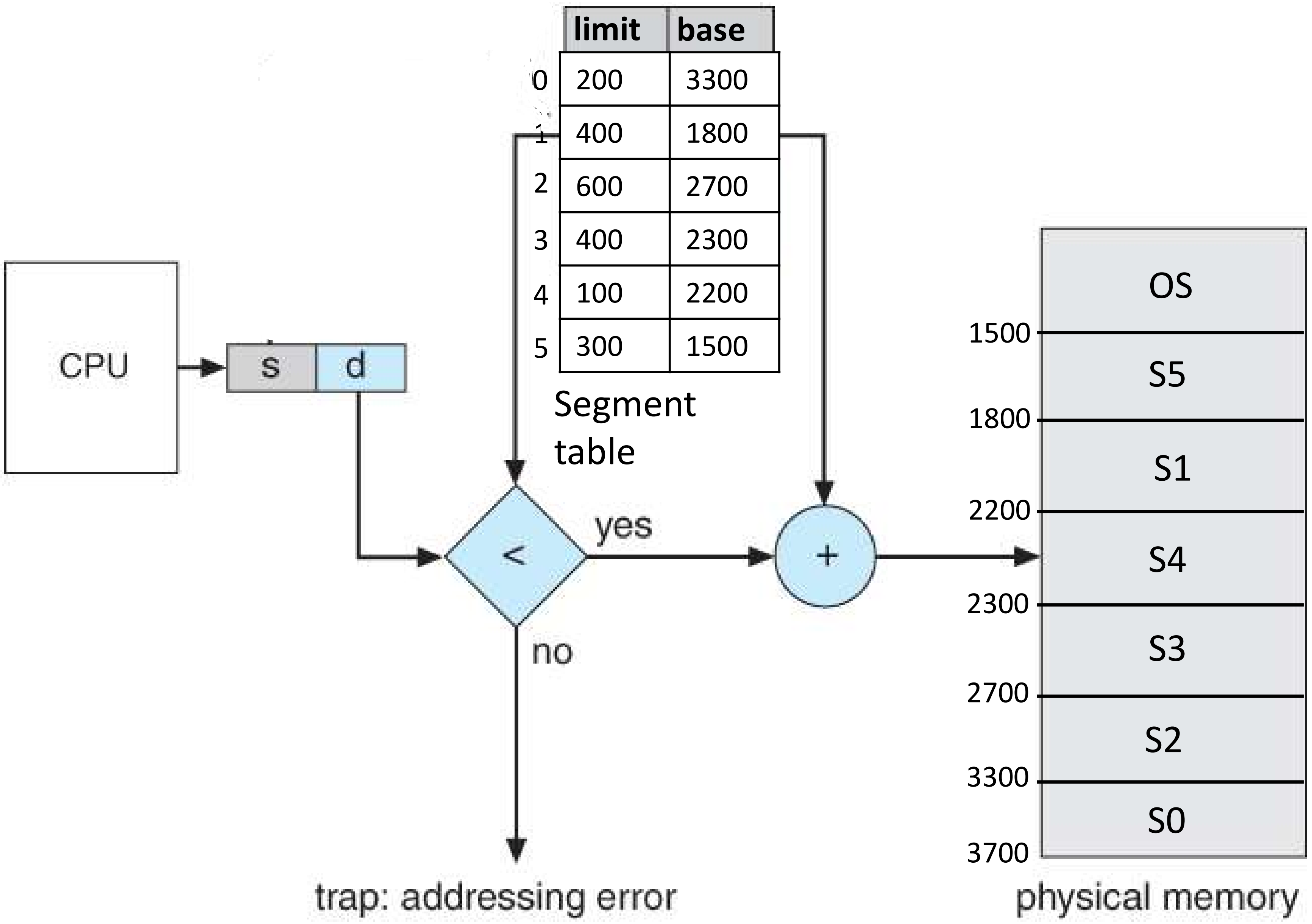
- Most **users** (programmers) do not think of their programs as existing in one continuous linear address space.
- Rather they tend to think of their memory in multiple ***segments***, each dedicated to a **particular use**, such as code, data, the stack, the heap, etc.
- Memory ***segmentation*** supports this view by providing addresses with a segment number (mapped to a segment base address) and an offset from the beginning of that segment.



Segment table



	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700



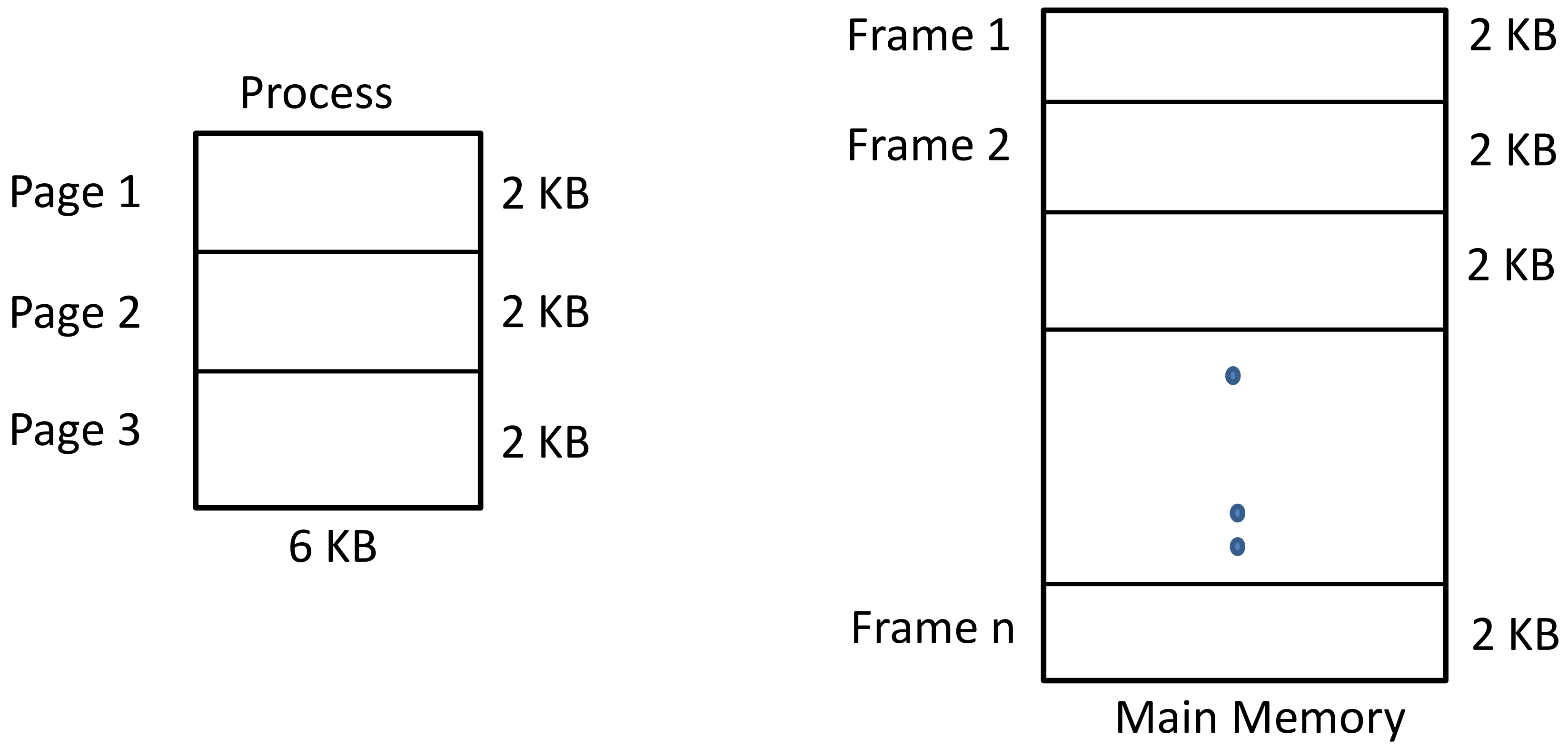
	limit	base
0	200	3300
1	400	1800
2	600	2700
3	400	2300
4	100	2200
5	300	1500

Segment table

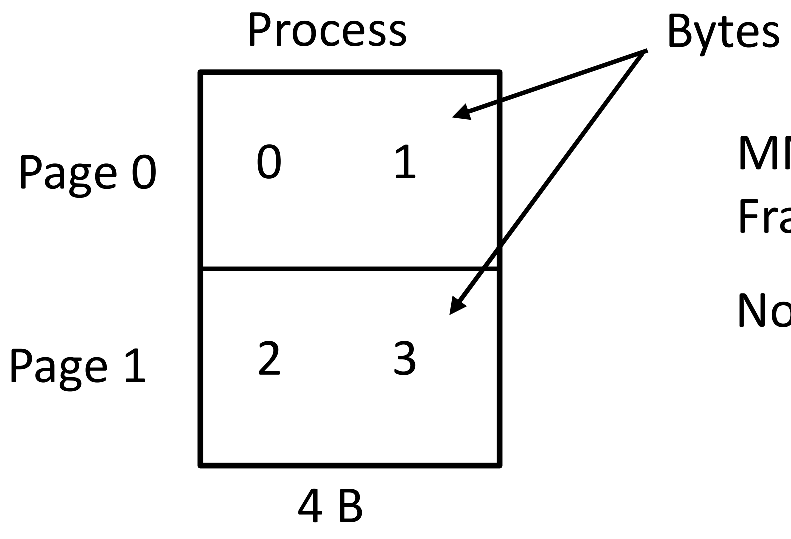
Segments to pages

- Large segments do not help the fragmentation problem
  - so we need small segments
- Small segments are usually full
  - so we don’t need a length/limit register
  - just make them all the same length
- Identical length segments are called *pages*
- We use page tables instead of segment tables
  - base register but no limit register

Paging

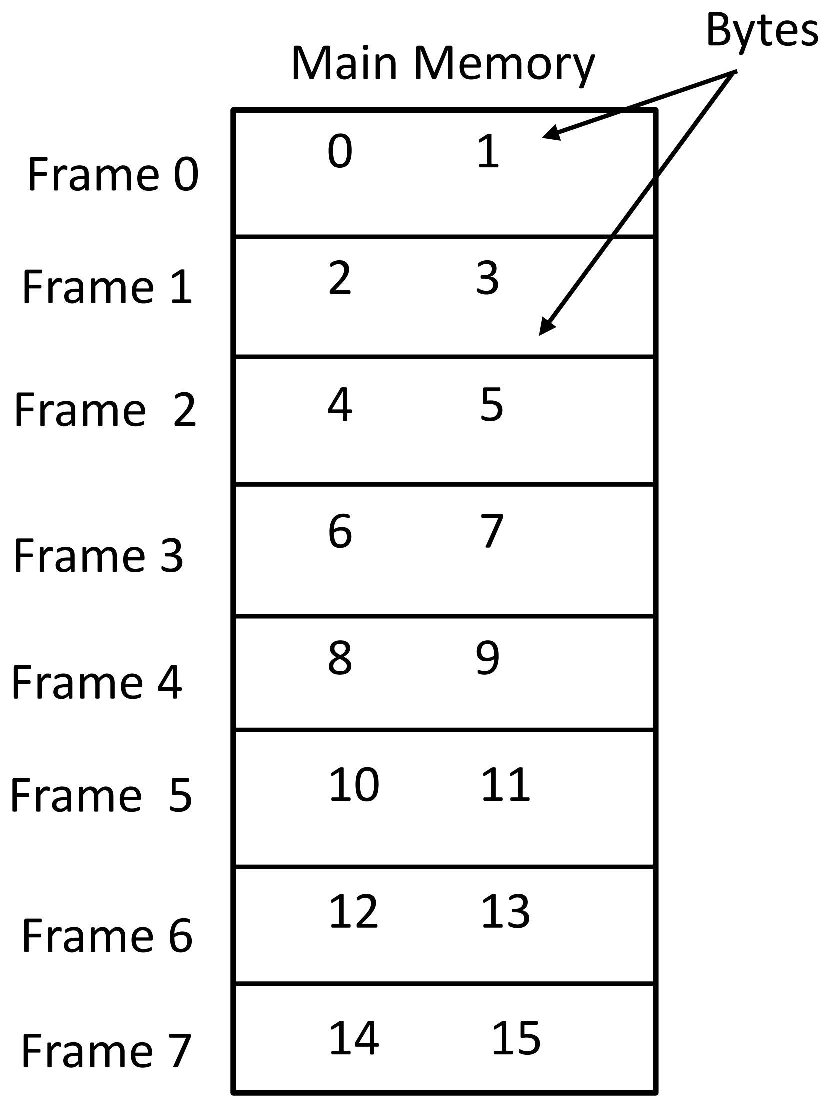

Process
Page 1
Page 2
Page 3
2 KB
2 KB
2 KB
6 KB
Frame 1
Frame 2
Frame n
2 KB
2 KB
2 KB
2 KB
Main Memory

•



MM size = 16 B
Frame size = 2 B

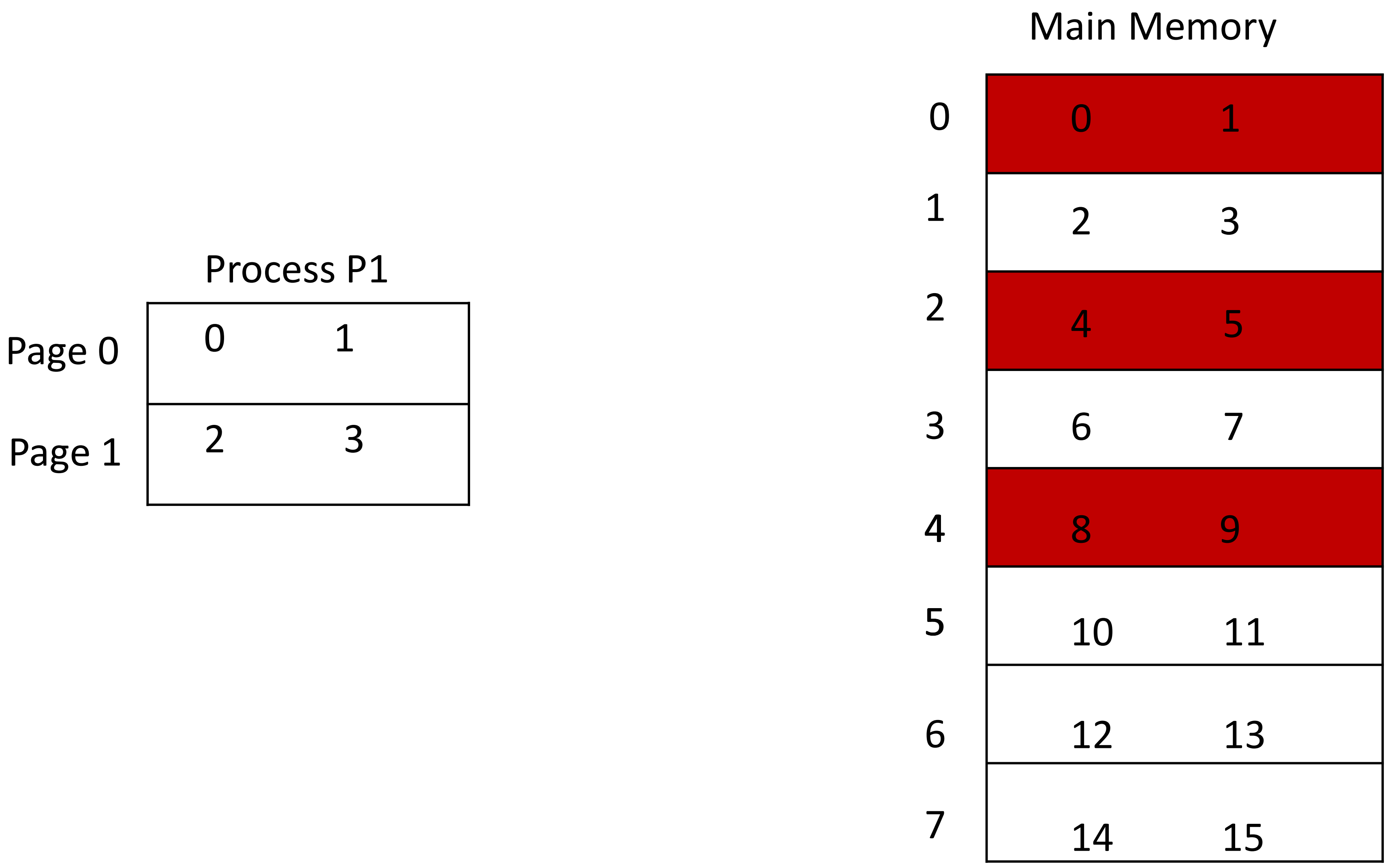
No. of frames = 8



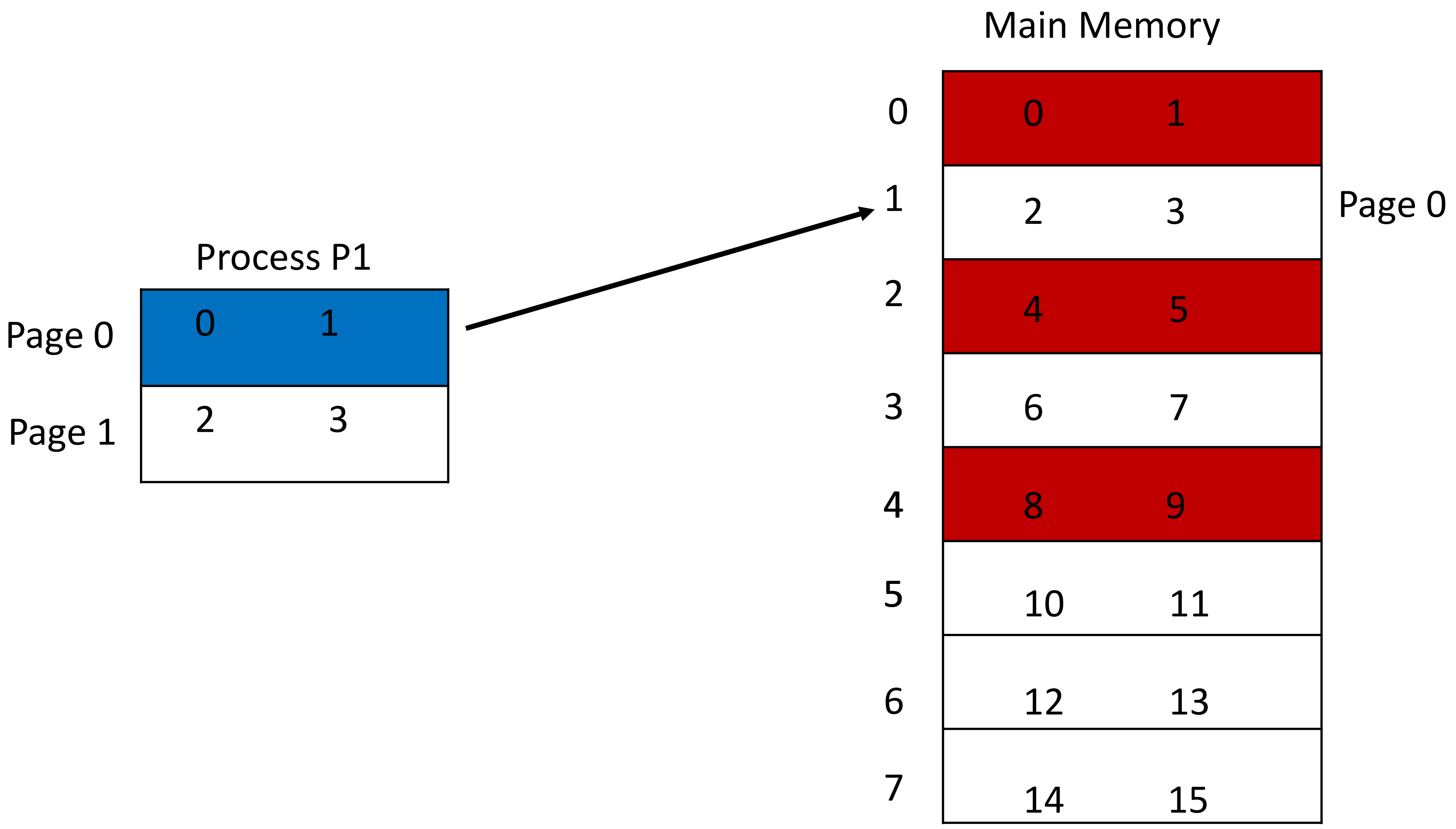
•

Process size = 4 B
Page size = 2 B

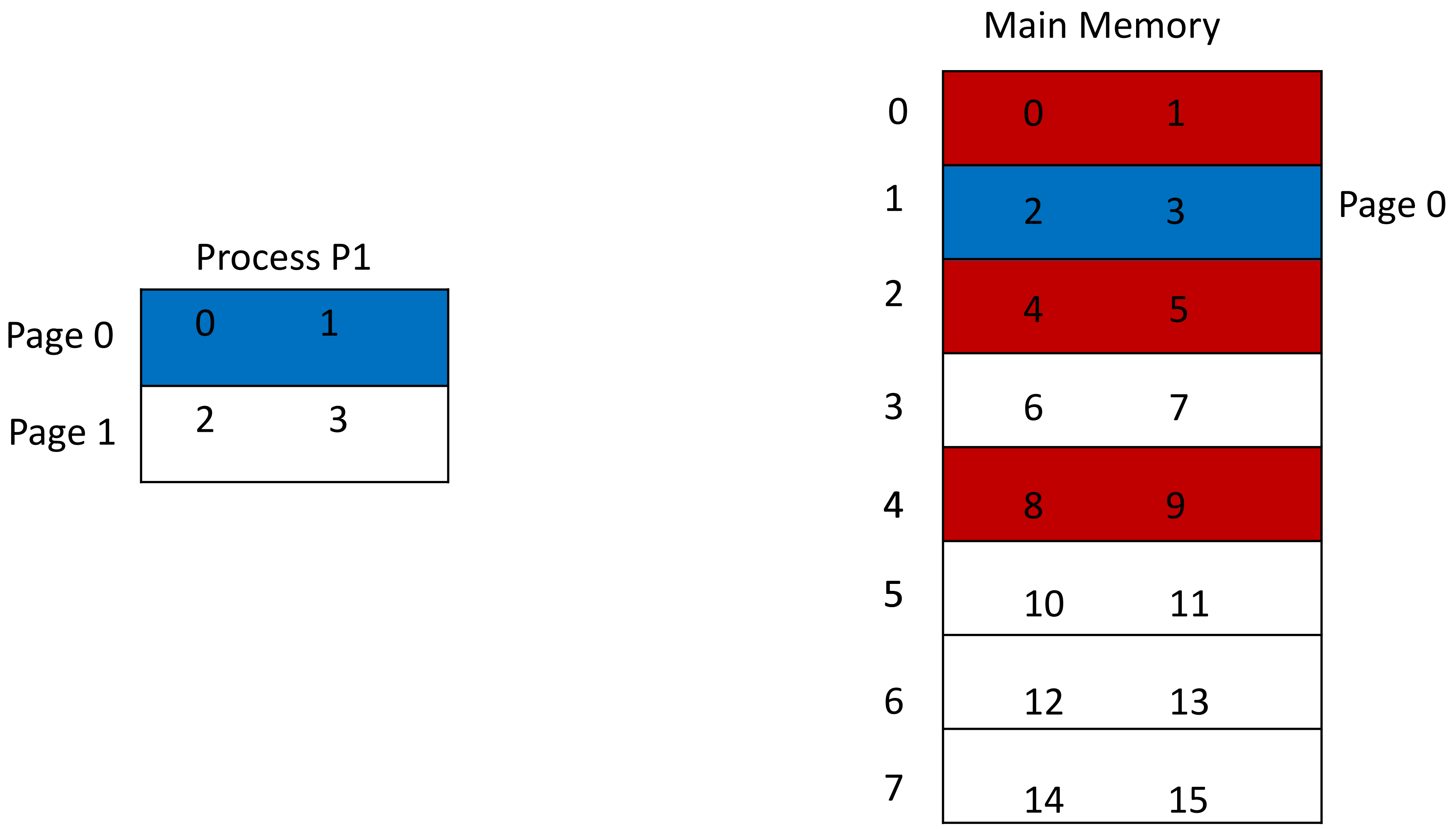
No. of pages = 2


Process P1
Page 0
Page 1
0
1
2
3
Main Memory
0
1
2
3
4
5
6
7
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

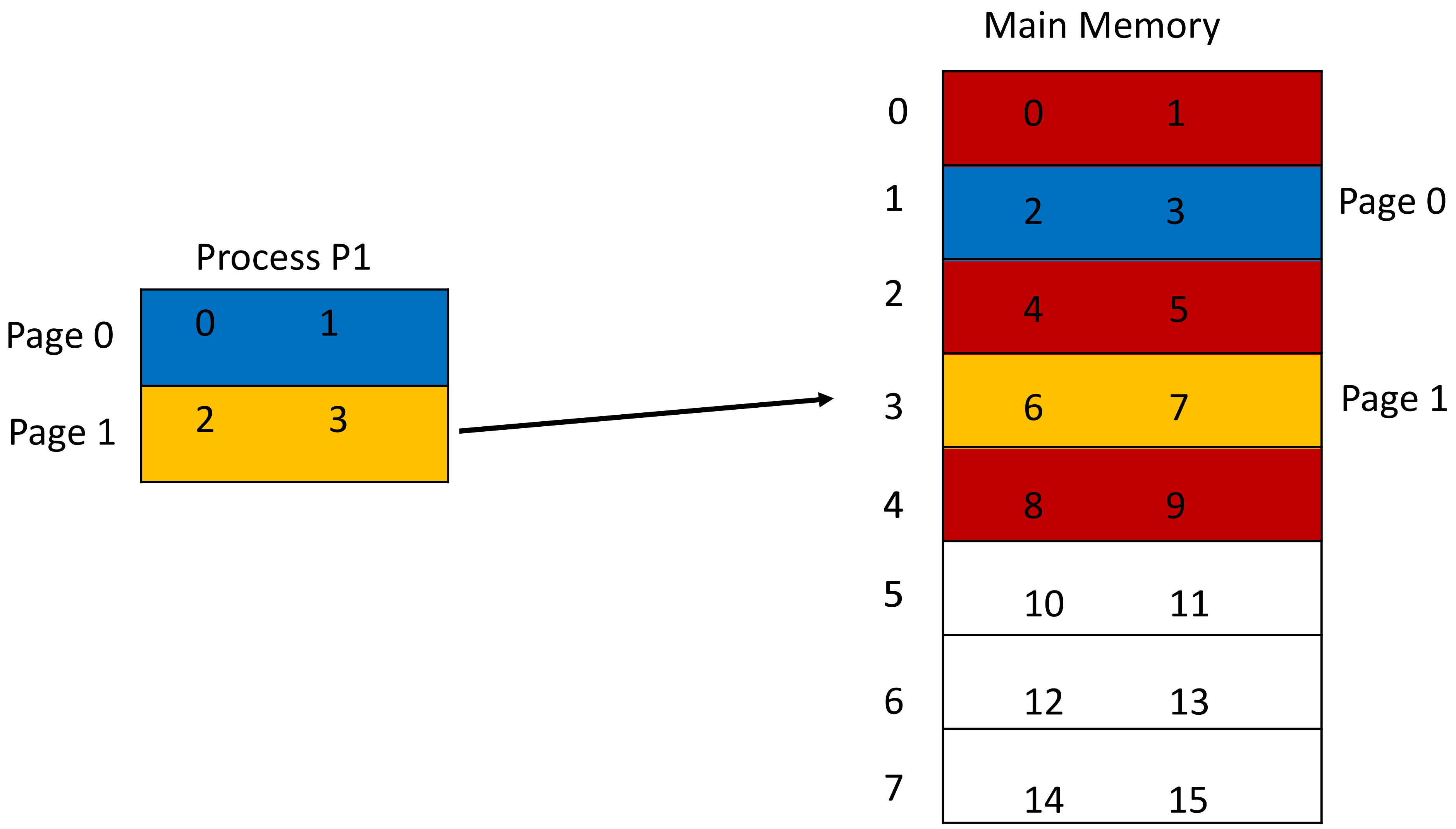
•


Main Memory
Process P1
Page 0
Page 1
0 1
2 3
Page 0
0
1
2
3
4
5
6
7
0 1
2 3
4 5
6 7
8 9
10 11
12 13
14 15

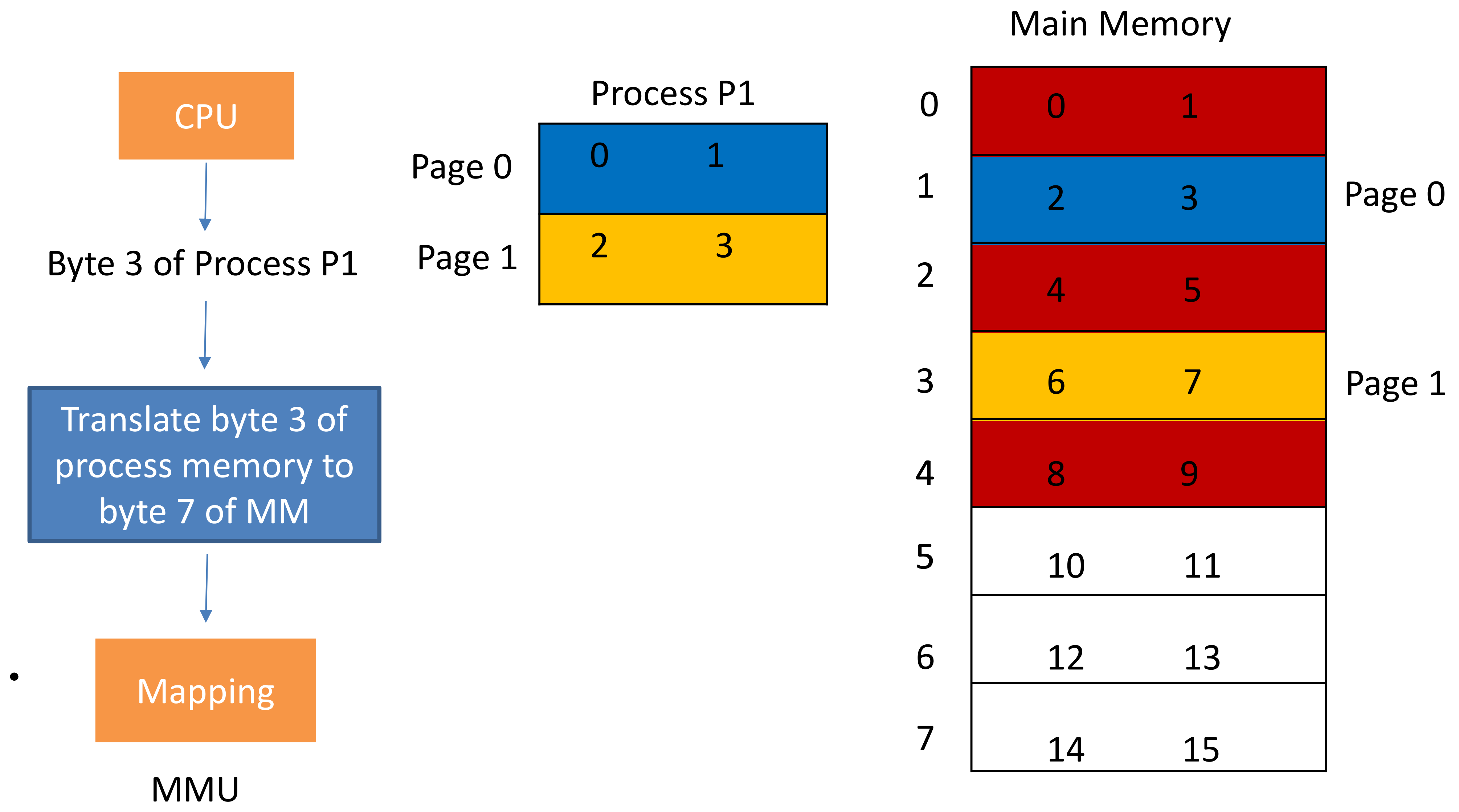
•

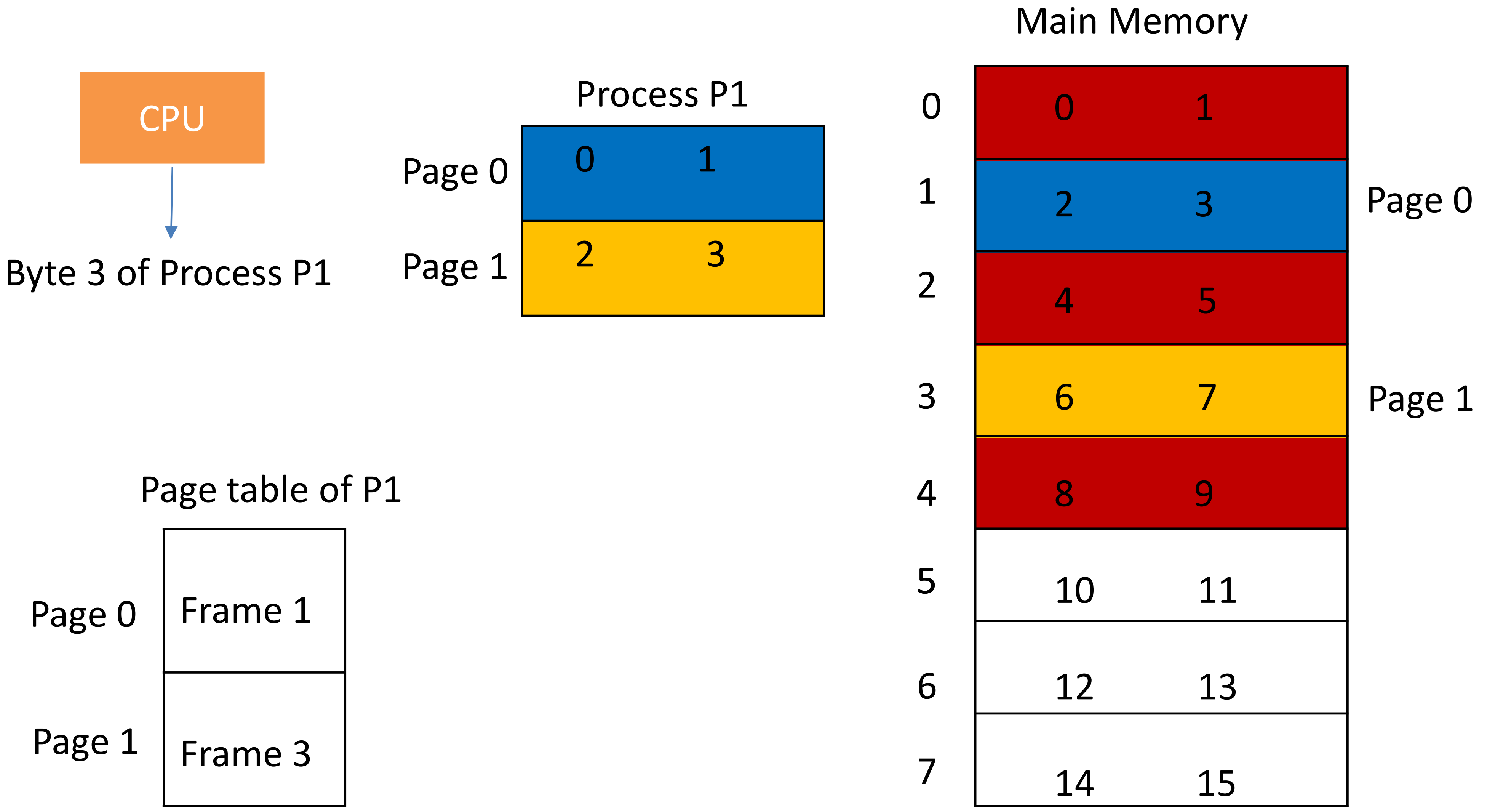

Process P1
Page 0
Page 1
0
1
2
3
Main Memory
0
1
2
3
4
5
6
7
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Page 0

•


Process P1
Page 0
Page 1
0
1
2
3
Main Memory
0
1
2
3
4
5
6
7
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Page 0
Page 1

•

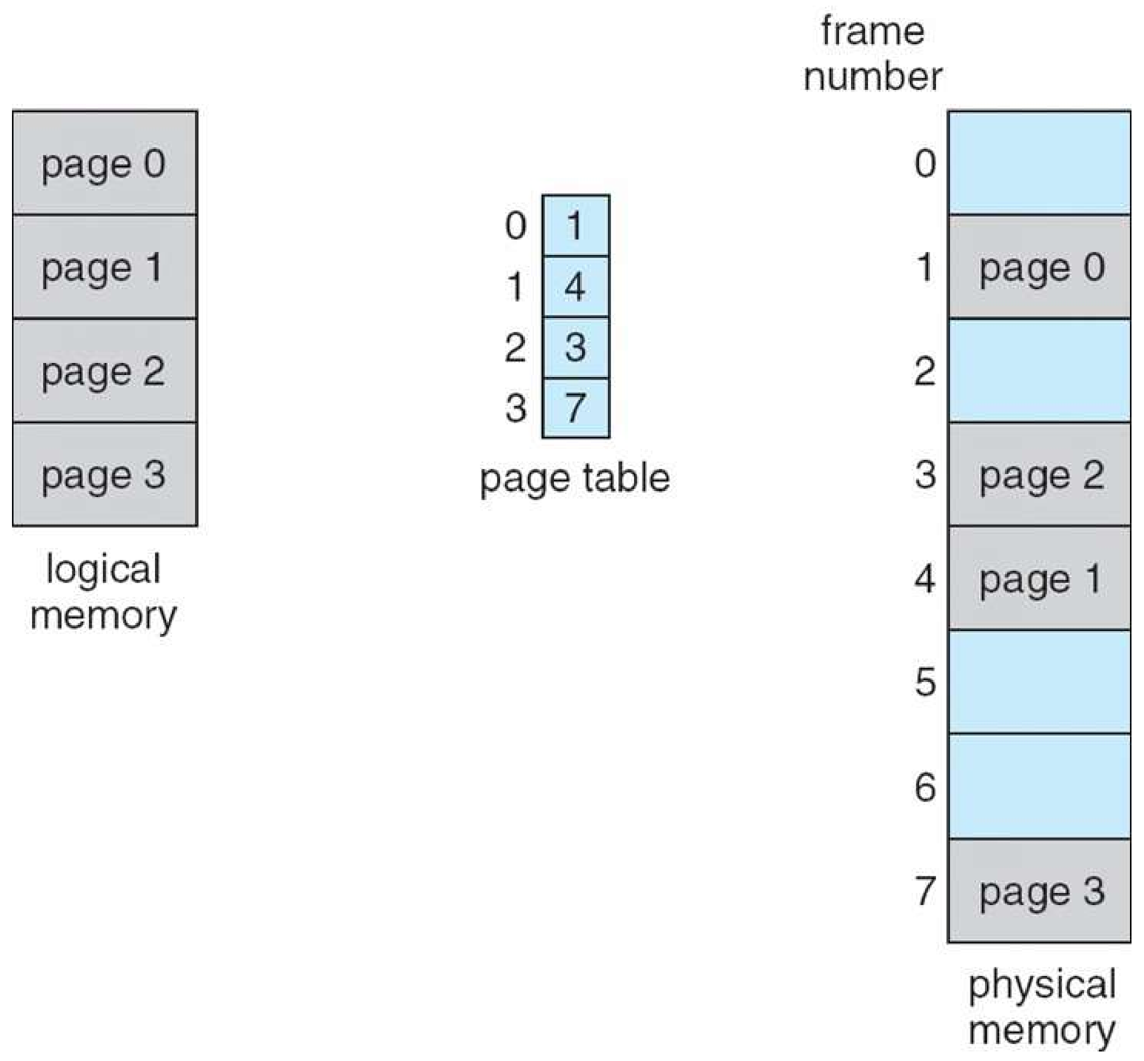

CPU
Byte 3 of Process P1
Translate byte 3 of process memory to byte 7 of MM
Mapping
MMU
Process P1
Page 0
Page 1
0
1
2
3
Main Memory
0
1
2
3
4
5
6
7
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Page 0
Page 1



CPU
Byte 3 of Process P1
Process P1
Page 0
Page 1
0
1
2
3
Main Memory
0 1 2 3 4 5 6 7
0 1
2 3
4 5
6 7
8 9
10 11
12 13
14 15
Page 0
Page 1
Page table of P1
Page 0
Frame 1
Page 1
Frame 3

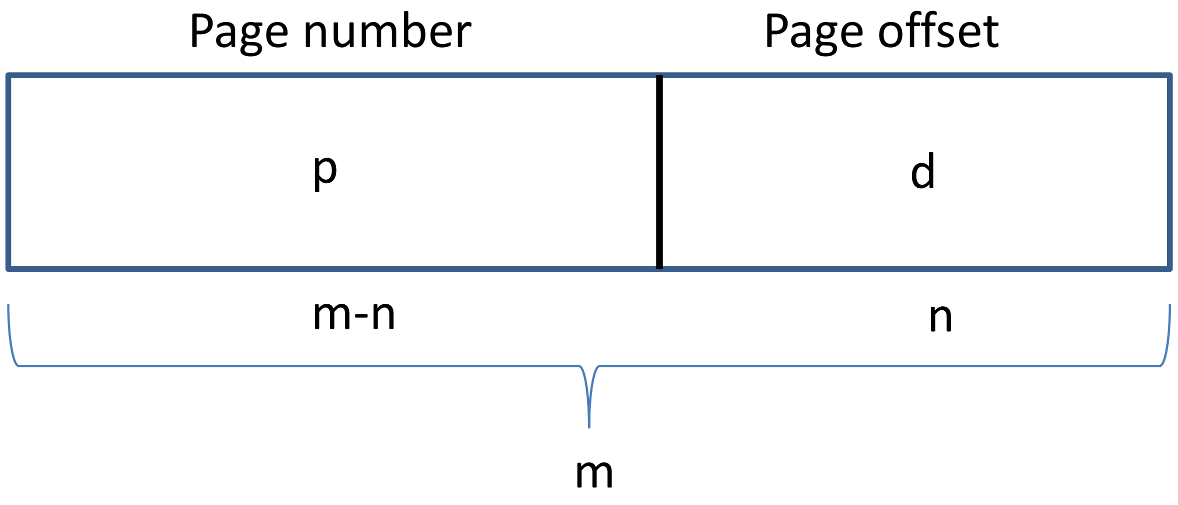
-

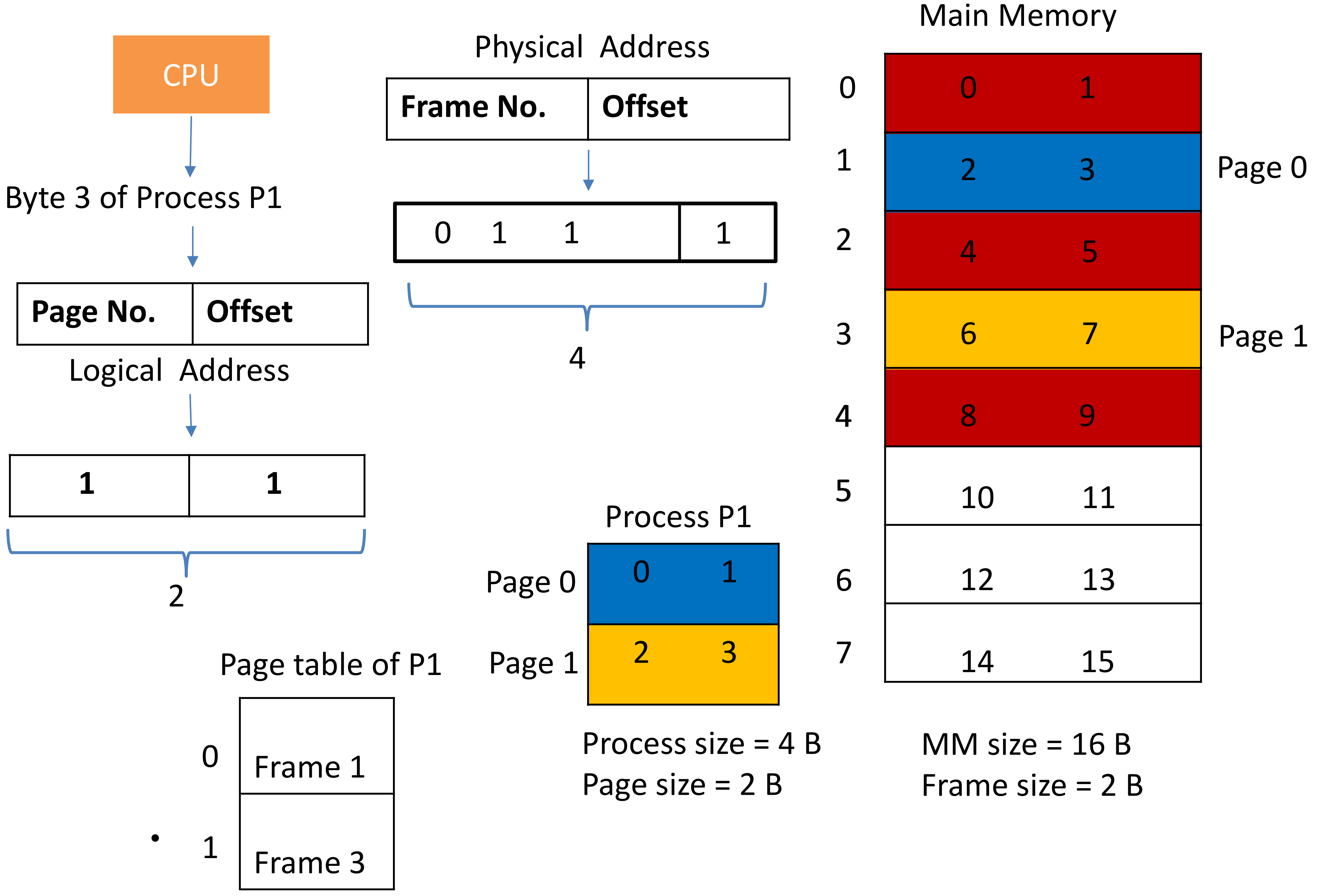
Paging model of Logical and Physical Memory



Logical address

- Address generated by CPU is divided into:
  - P**age number (*p) – used as an index into a page table which*** contains base address of each page in physical memory
  - P**age offset (*d) – combined with base address to define the*** physical memory address that is sent to the memory unit
  - For given logical address space 2^*m and page size 2^n*

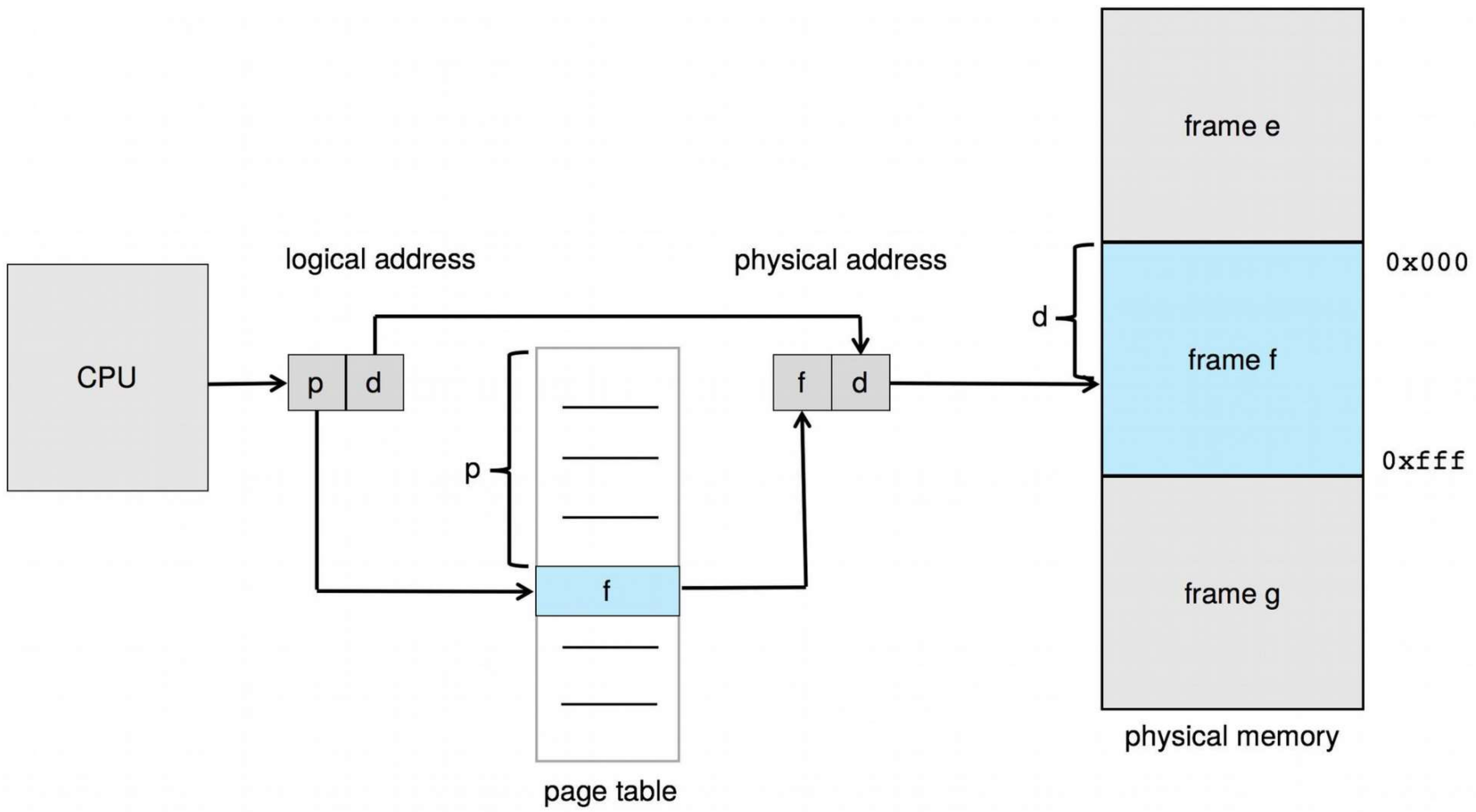




Process size = 4 B
Page size = 2 B

MM size = 16 B
Frame size = 2 B

Paging Hardware



Paging summary

- Physical address space of a process can be noncontiguous
- Process is allocated physical memory whenever it is available
- Paging is a memory management scheme that allows processes physical memory to be discontinuous, and which eliminates problems with fragmentation by allocating memory in ***equal sized blocks*** known as ***pages***.
- Divide physical memory into fixed-sized blocks called **frames**
- Divide logical memory into blocks of same size called **pages**
- To run a program of size ***N pages, need to find N free frames and*** load program
- Set up a **page table to translate logical to physical addresses**
- Any page (from any process) can be placed into any available frame.
- Can still have internal fragmentation

Page table entry

Frame No.	Valid (1) / Invalid (0)	Protection (rwx)	Reference	Caching	Dirty Bit / Modified bit
Mandatory field	Optional fields				

Page Table Implementation

- Page table is kept in main memory
  - **Page-table base register (PTBR) points to the page** table
  - **Page-table length register (PTLR) indicates size of the** page table

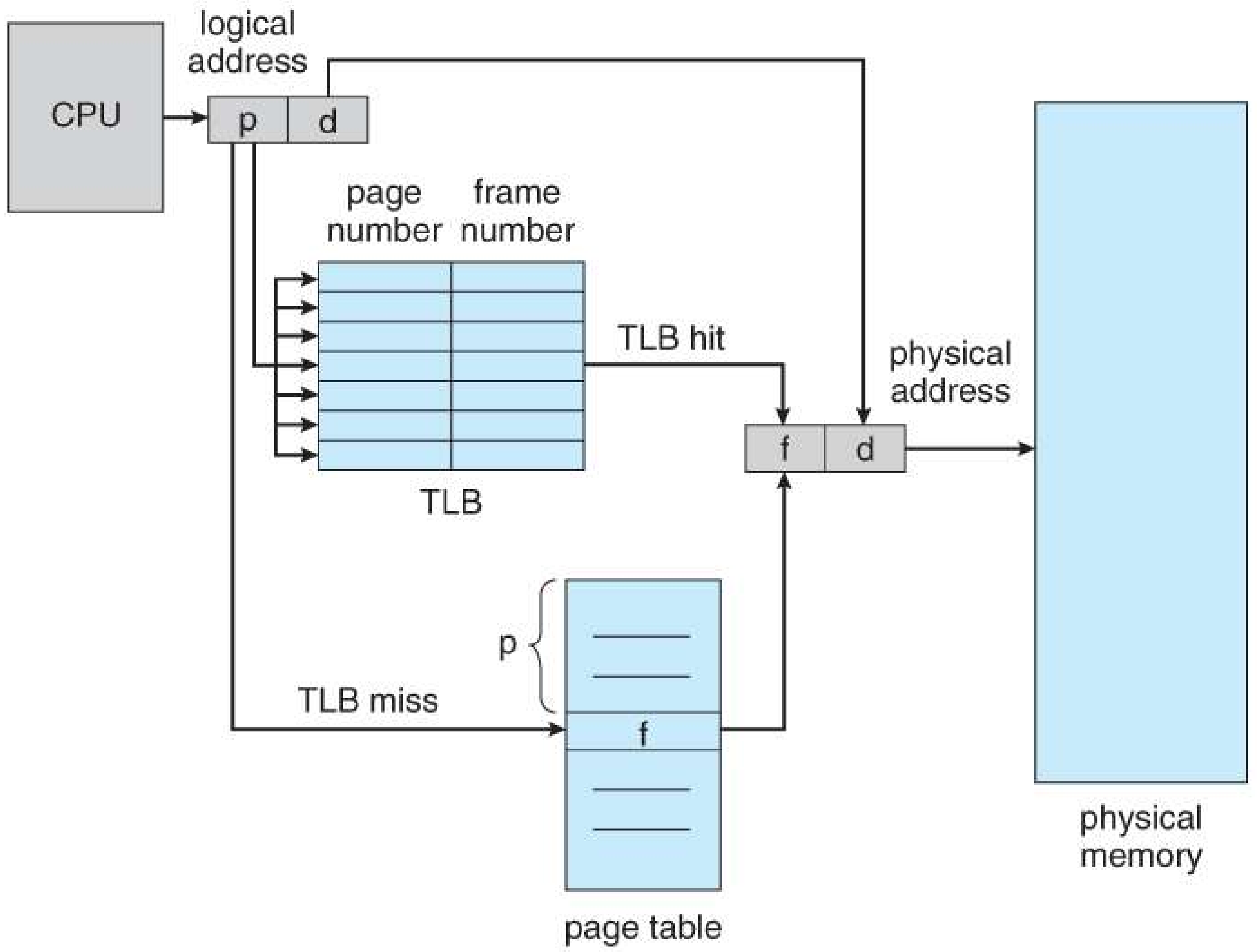
- In this scheme every data/instruction access requires two memory accesses
  - One for the page table and one for the data / instruction

- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **translation look-aside buffers (TLBs) (also called associative memory).**

TLB

- We saw that with paging, every memory access requires ***two*** memory accesses - One to fetch the block number from page table and then another one to access the desired memory location.
- The solution - a very special high-speed memory device called the ***translation look-aside buffer, TLB.***
- The TLB is on the cache is very expensive and therefore very small. (Not large enough to hold the entire page table.)
- Addresses are first checked against the TLB, and if the info is not there (a ***TLB miss***), then the frame is looked up from main memory and the TLB is updated.
- The percentage of time that the desired information is found in the TLB is termed the ***hit ratio***.

TLB



CPU
logical address
p
d
page number
frame number
TLB
TLB hit
physical address
f
d
p
TLB miss
f
page table
physical memory

Effective access time

- Hit ratio – percentage of times that a page number is found in the TLB
  - An 80% hit ratio means that we find the desired page number in the TLB 80% of the time.
  - Effective Access Time = Hit (TLB access time + Main memory access time) + Miss (TLB + Page table access time + Main memory access time)

** This is assuming that there is no page fault, because in case there is a page fault then the page fault service time will be considered i.e. the page has to be brought to the main memory from the secondary memory.

Question

- TLB access time = 10 ns
- Main memory access time = 50ns
- What is the effective memory access time (in ns) if the TLB hit ratio is 90% and there is no page fault.

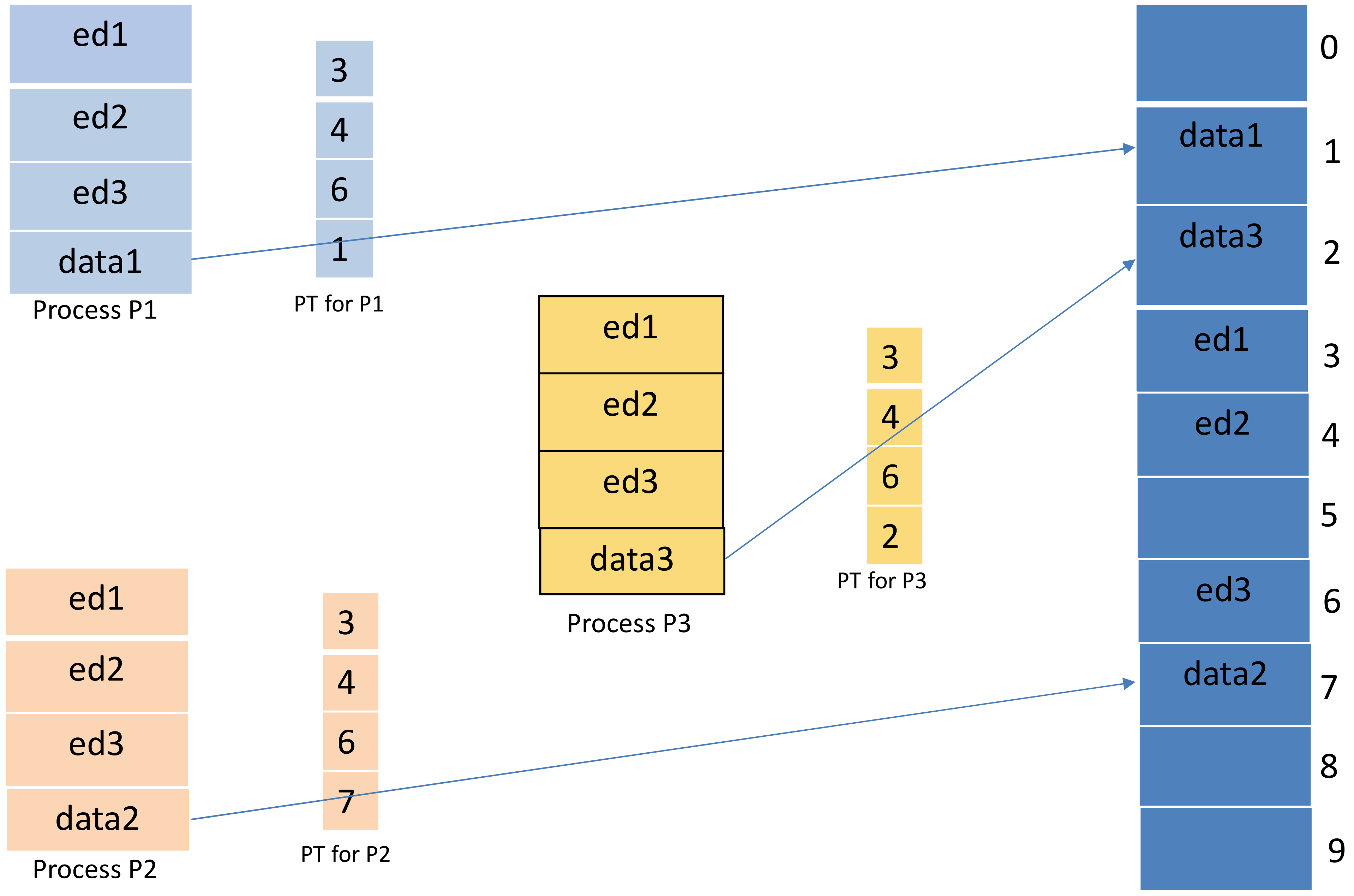
Paging- advantages and disadvantages

- **Advantages**:
  - Avoids external fragmentation
  - Provide more memory that can be used for more jobs
  - Higher degree of multiprogramming results in increased processor and memory utilization
  - Compaction overhead in relocatable partition schemes is eliminated

- **Disadvantages**:
  - Page address mapping hardware usually increases cost of computer and also slows down processor
  - Memory is used to store PMT; processor time (overhead) must be expended to maintain and update these tables
  - Though external fragmentation is eliminated, Internal Fragmentation / Page Breakage does occur.

Shared Pages & Reentrant code

- One of the advantage of Paging
- Pages with common code shared between multiple processes
- Data is unique to each process in its own registers and data storage
- Two or more processes can execute same code at the same time
- Reentrant code is non-self modifying code
- Only reentrant code can be shared



Only one copy of editor(ed*) needs to be kept in memory

Questions ??