Paging

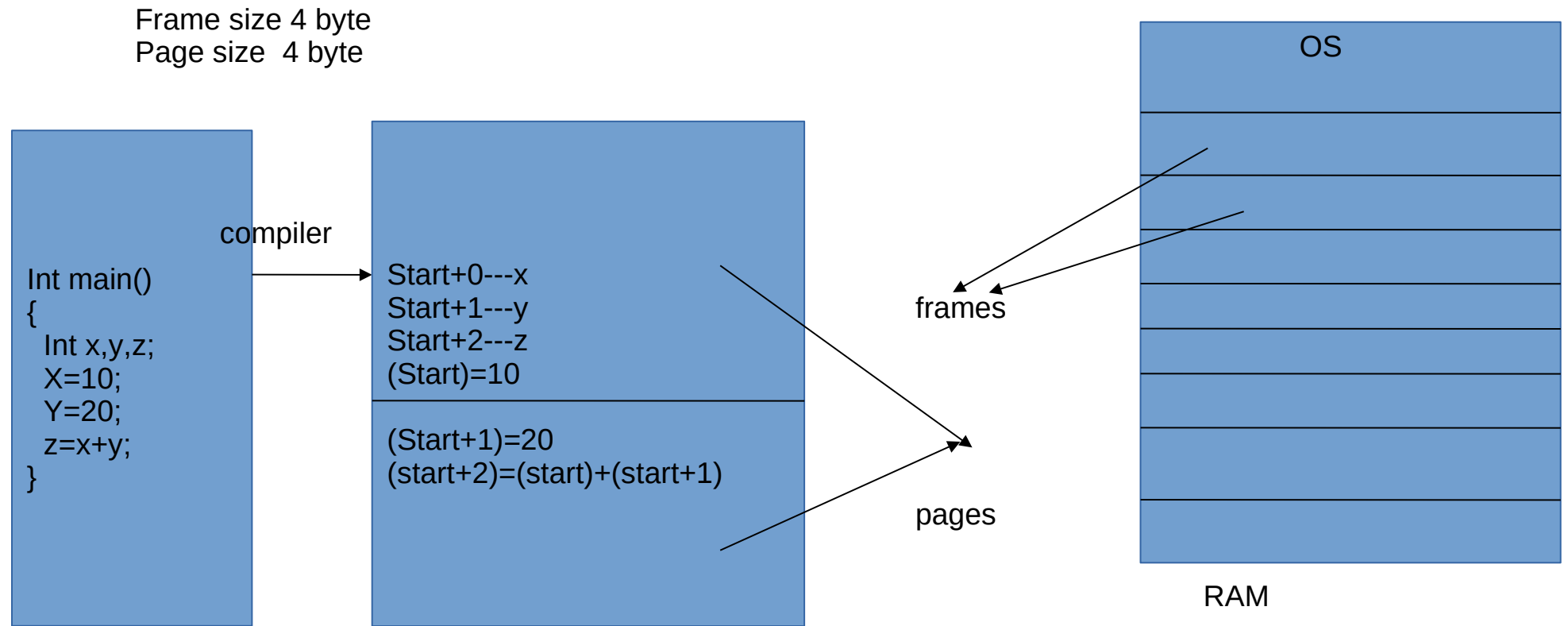
An address generated by the CPU is commonly refereed to as a logical address, whereas an address seen by the memory unit(the one loaded in to the MAR) is commonly referred to as a physical address.

The set of all logical addresses generated by a program is referred to as a logical address space.

Whereas the set of all physical addresses corresponding to these logical addresses is referred to as a physical address space.

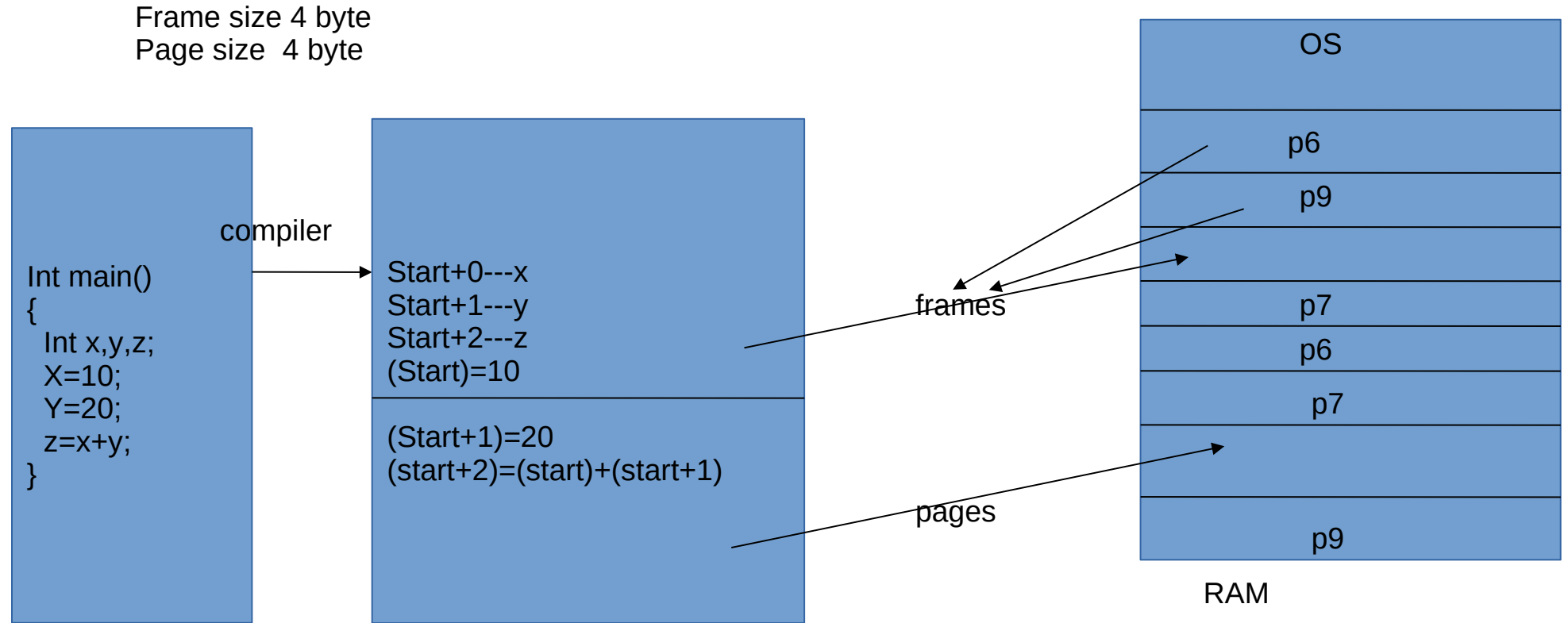
Physical memory is broken into fixed size blocks called **frames** .

Logical memory is also broken into blocks of the same size called **pages.**



When a process is to be executed, its pages (which are in the backing storage) are loaded into any available memory frames.

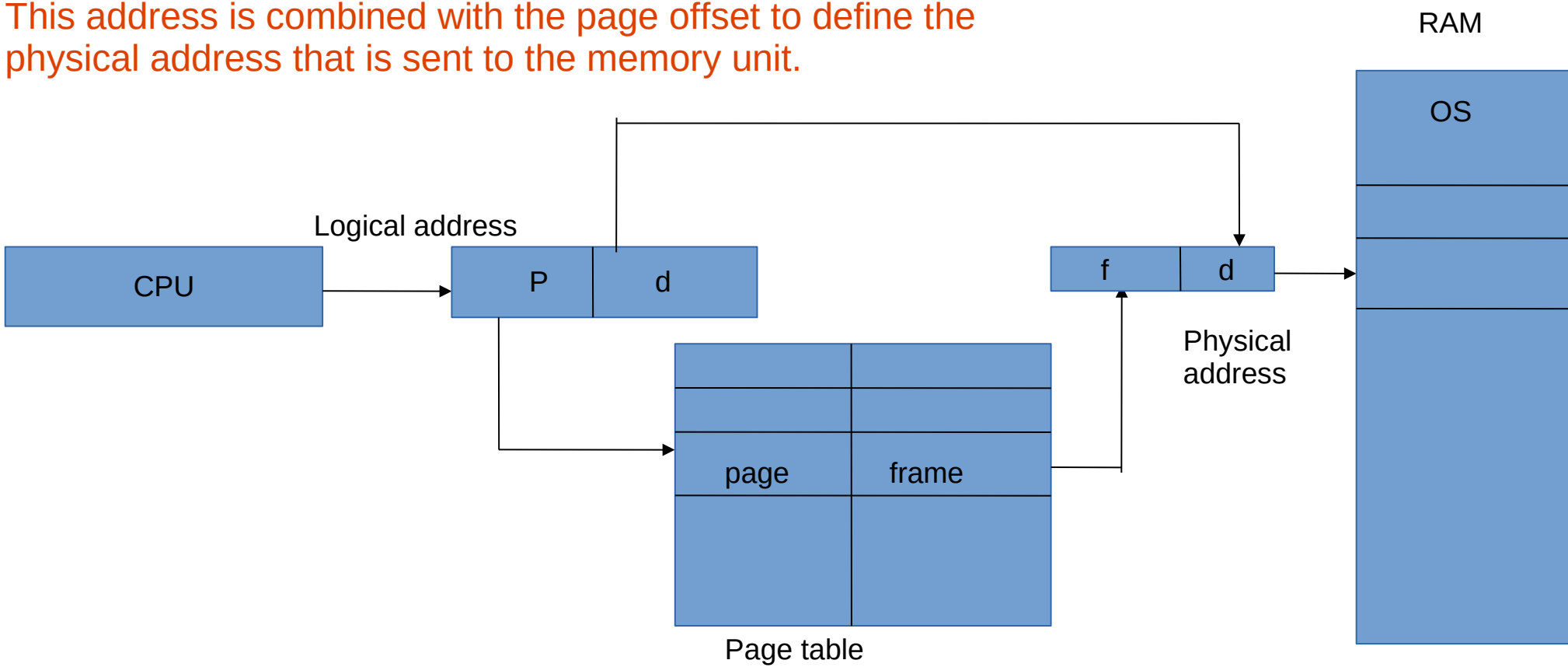
Therefore the pages of a process may not be contiguous.



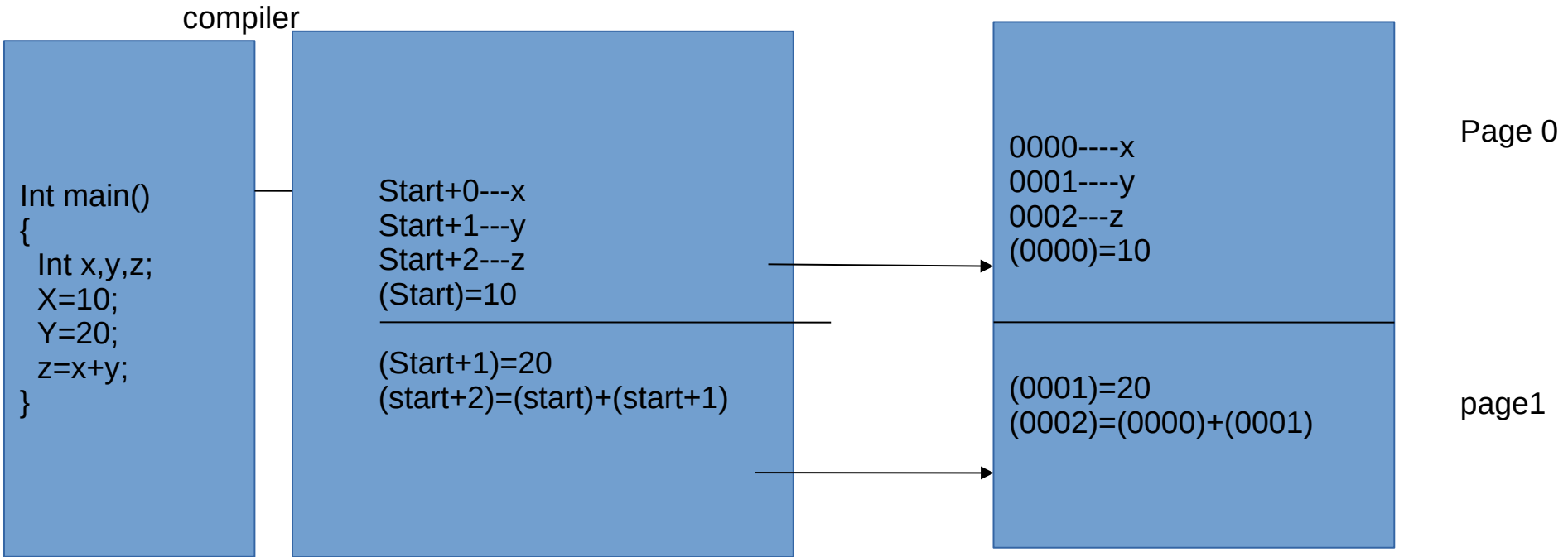
Every address generated by the CPU is divided into 2 parts, a page number(p) and a page offset(d).

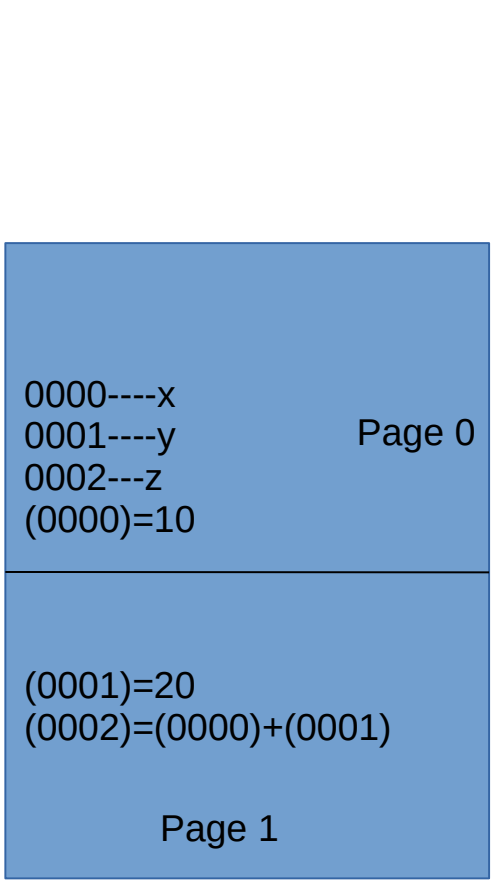
The page number p is used as index to a page table which keeps the starting address of each page in the physical memory(frame).

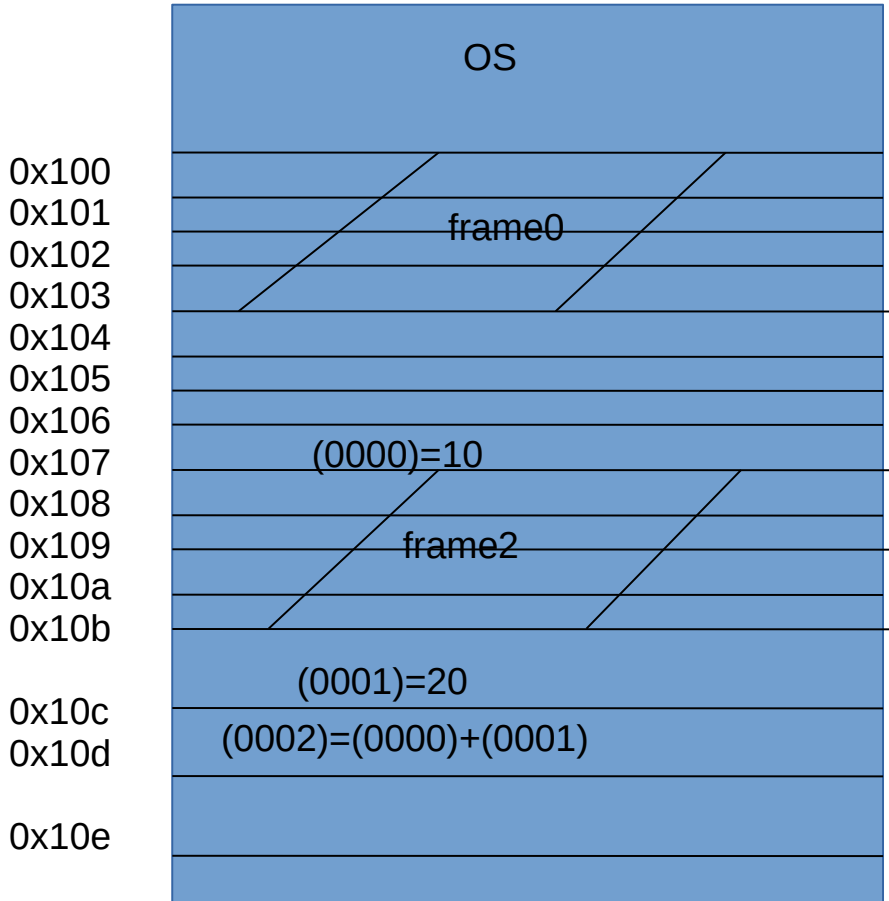
This address is combined with the page offset to define the physical address that is sent to the memory unit.

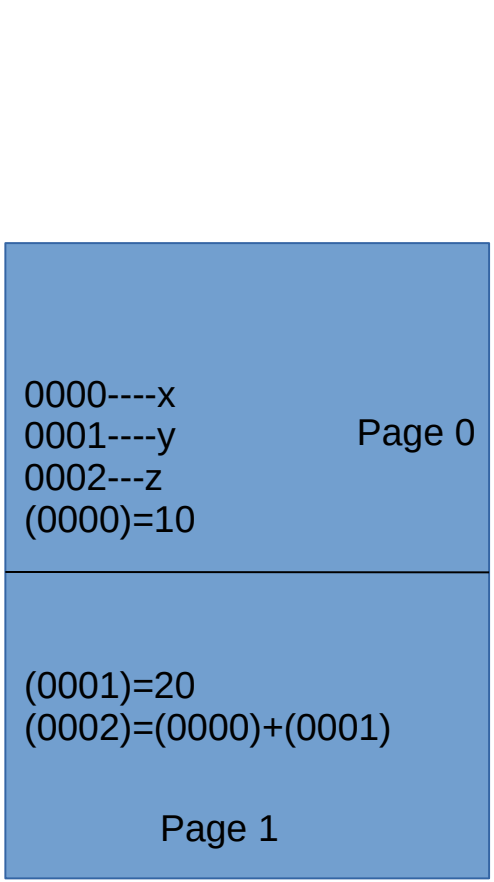


Frame size 4 byte
Page size 4 byte

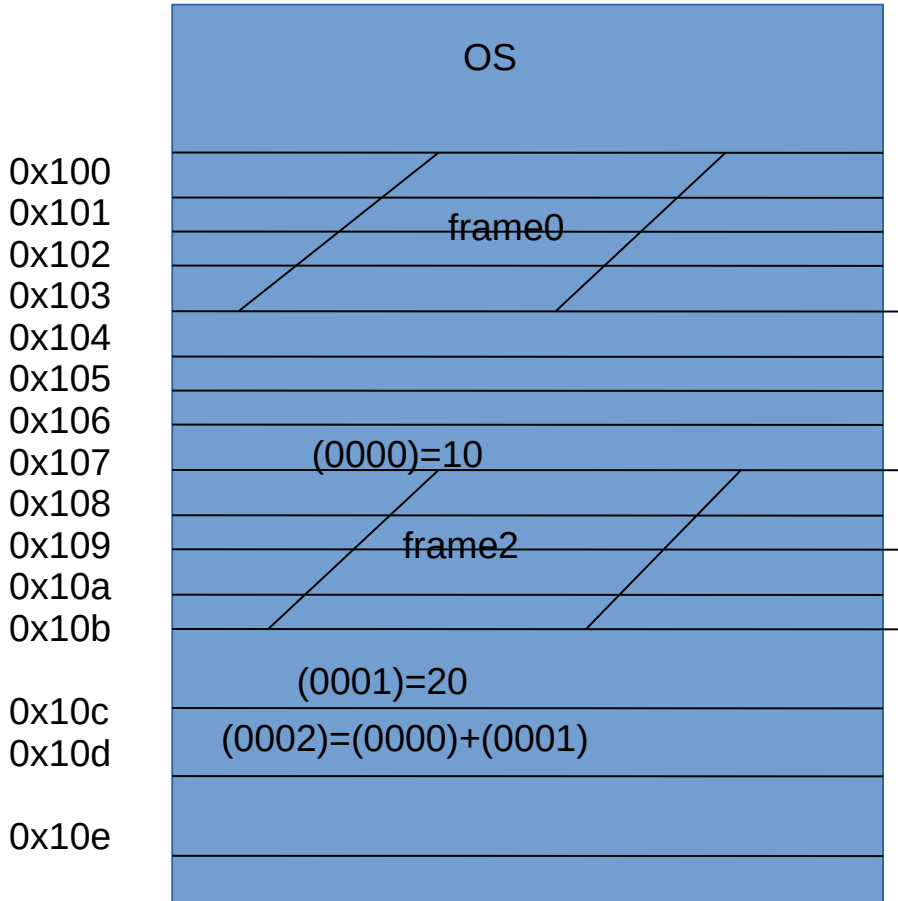


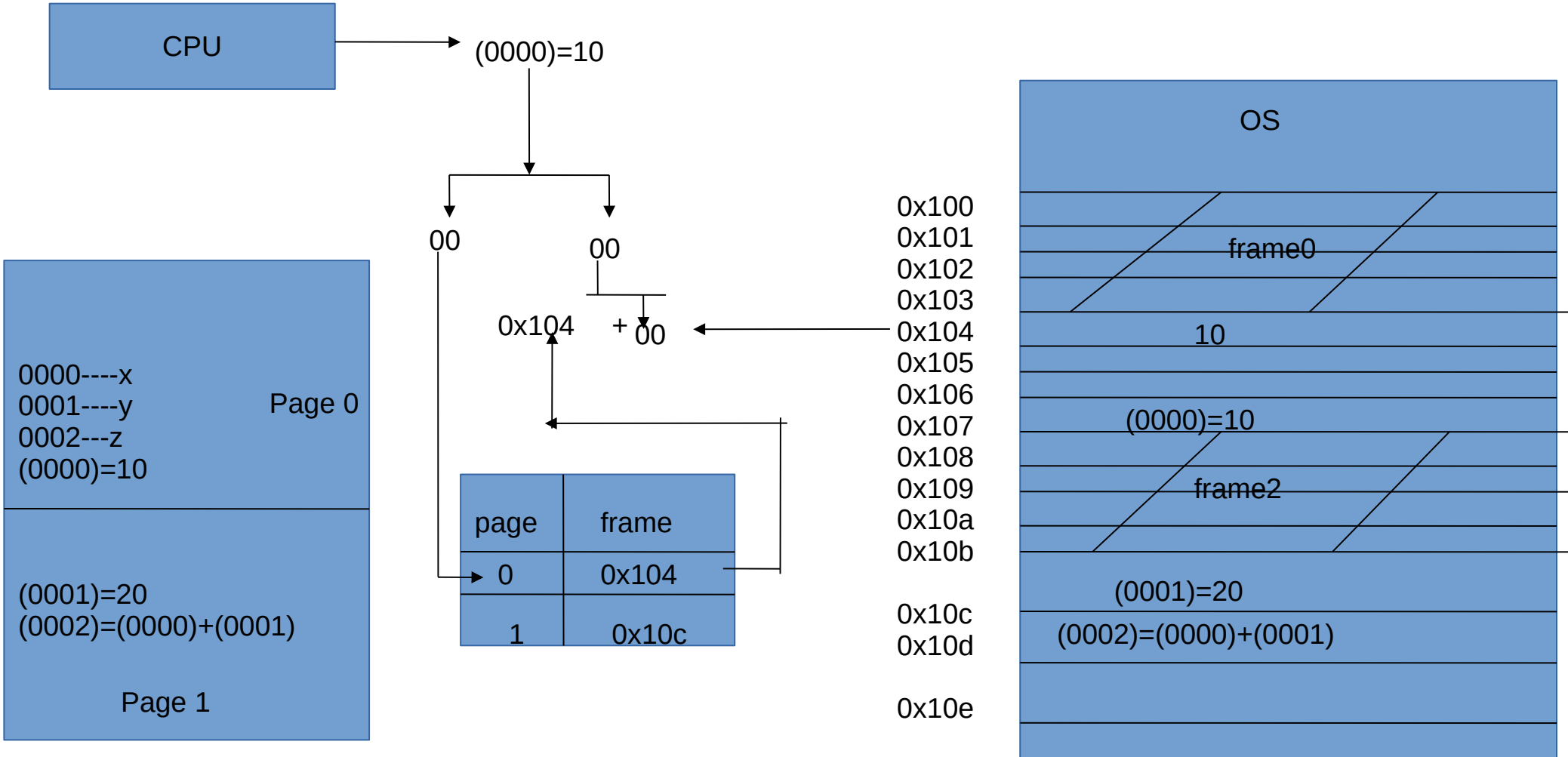

0000----x
0001----y
0002---z
(0000)=10
Page 0
(0001)=20
(0002)=(0000)+(0001)
Page 1


OS
0x100
0x101
0x102
0x103
0x104
0x105
0x106
0x107
0x108
0x109
0x10a
0x10b
0x10c
0x10d
0x10e
frame0
(0000)=10
frame2
(0001)=20
(0002)=(0000)+(0001)



page	frame
0	0x104
1	0x10c



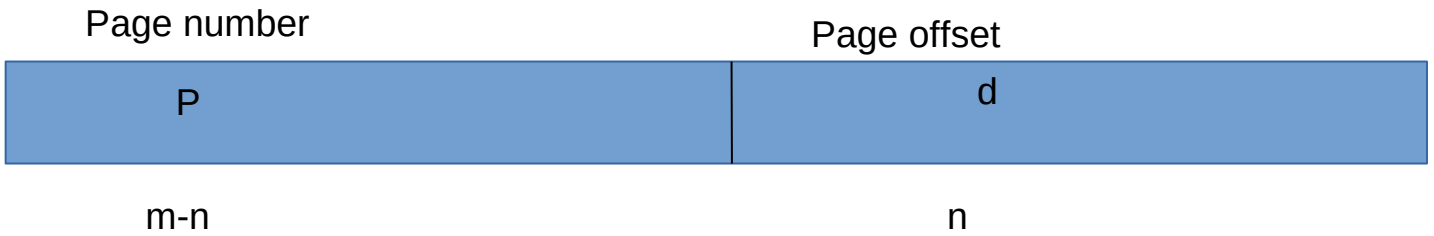

CPU
(0000)=10
00
00
0x104
+ 00
0000----x
0001----y
0002---z
(0000)=10
Page 0
(0001)=20
(0002)=(0000)+(0001)
Page 1
page
frame
0
0x104
1
0x10c
0x100
0x101
0x102
0x103
0x104
0x105
0x106
0x107
0x108
0x109
0x10a
0x10b
0x10c
0x10d
0x10e
OS
frame0
10
(0000)=10
frame2
(0001)=20
(0002)=(0000)+(0001)

How logical address is divided into page number and offset?

The page size which is defined by the hardware is typically a power of 2 varying between 512 bytes and 8192 bytes.

If the size of a logical address space is pow(2,m) and a page size is pow(2,n) addressing units then the high order m-n bits of a logical address designate the page number and low order bits designate the page offset.

There fore the logical address is as follows:

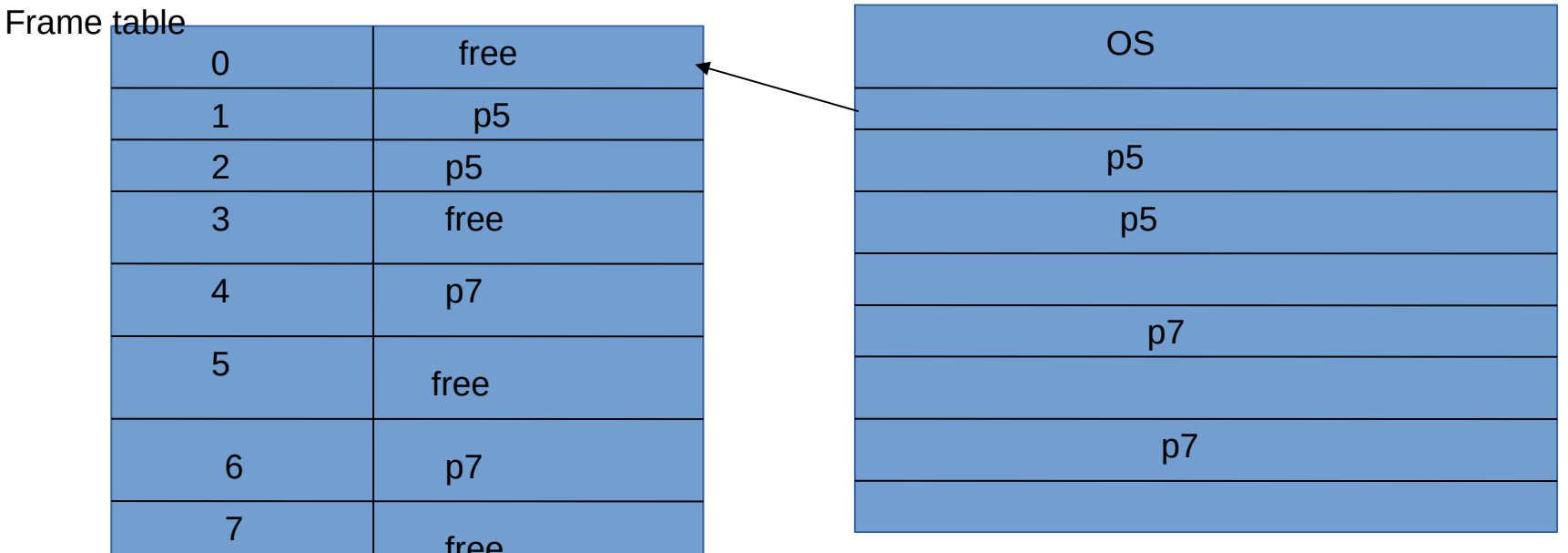


Paging eliminates external fragmentation altogether but there may be a little internal fragmentation

1. The logical address produced by the user program are translated into physical addresses.
2. This translation is controlled by the OS.
3. When a process arrives to be executed its size expressed in pages is examined.
4. Each user page need one frame.
5. If the required number of frames are available they are allocated to this process.
6. The first page of process is loaded into one of the allocated frames and the frame number is put into the page table for this process.
7. The next page is loaded into another frame and the frame number is similarly put into the page table and so on.

To keep track of the frames which are free and which are not the OS maintains a frame table, which has an entry for each of the physical frames.

The entry for a frame tells whether this frame is allocated or free and if it is allocated to which page of which process or processes.



The OS maintains a copy of the page table for each process just as it maintain a copy for the instruction counter and instruction register.

Consider page size = frame size = 4 bytes

Suppose the logical address space for a process is 20 bytes.

How many pages are there in this process?

Draw the page table for this process.....

Solution ?
Multi level paging

One way of getting around this problem is to use a two level paging scheme in which the page table itself is also paged.