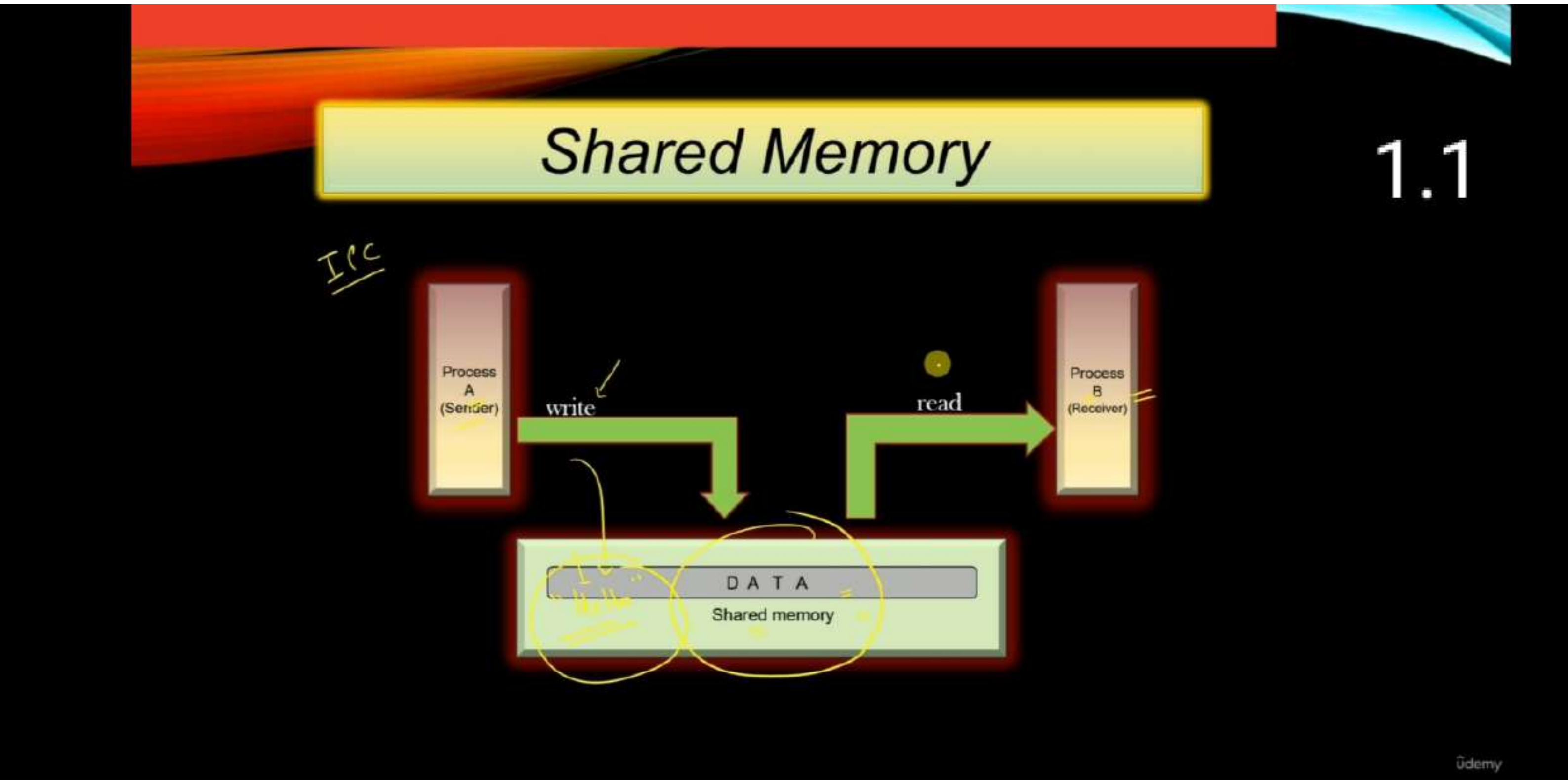

Shared Memory
1.1
IPC
Process
A
(Sender)
write
read
Process
B
(Receiver)
D A T A
Shared memory

1080P CC 1.0X
1.2

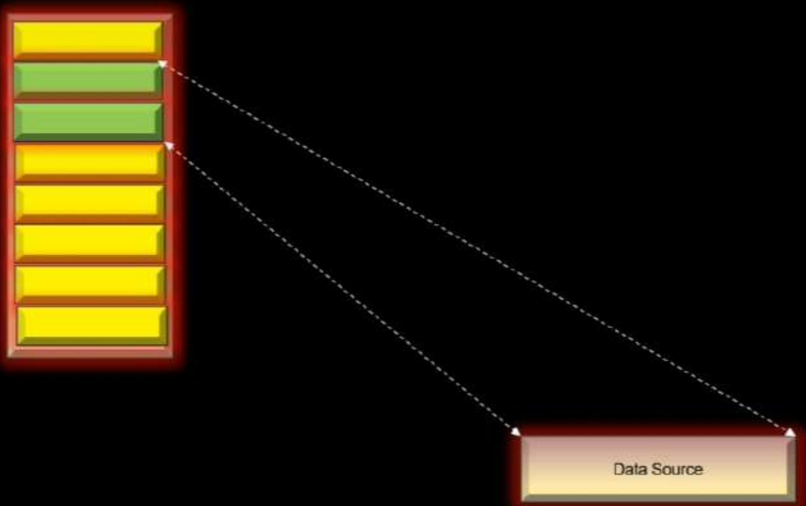
Agenda

Memory Mapping

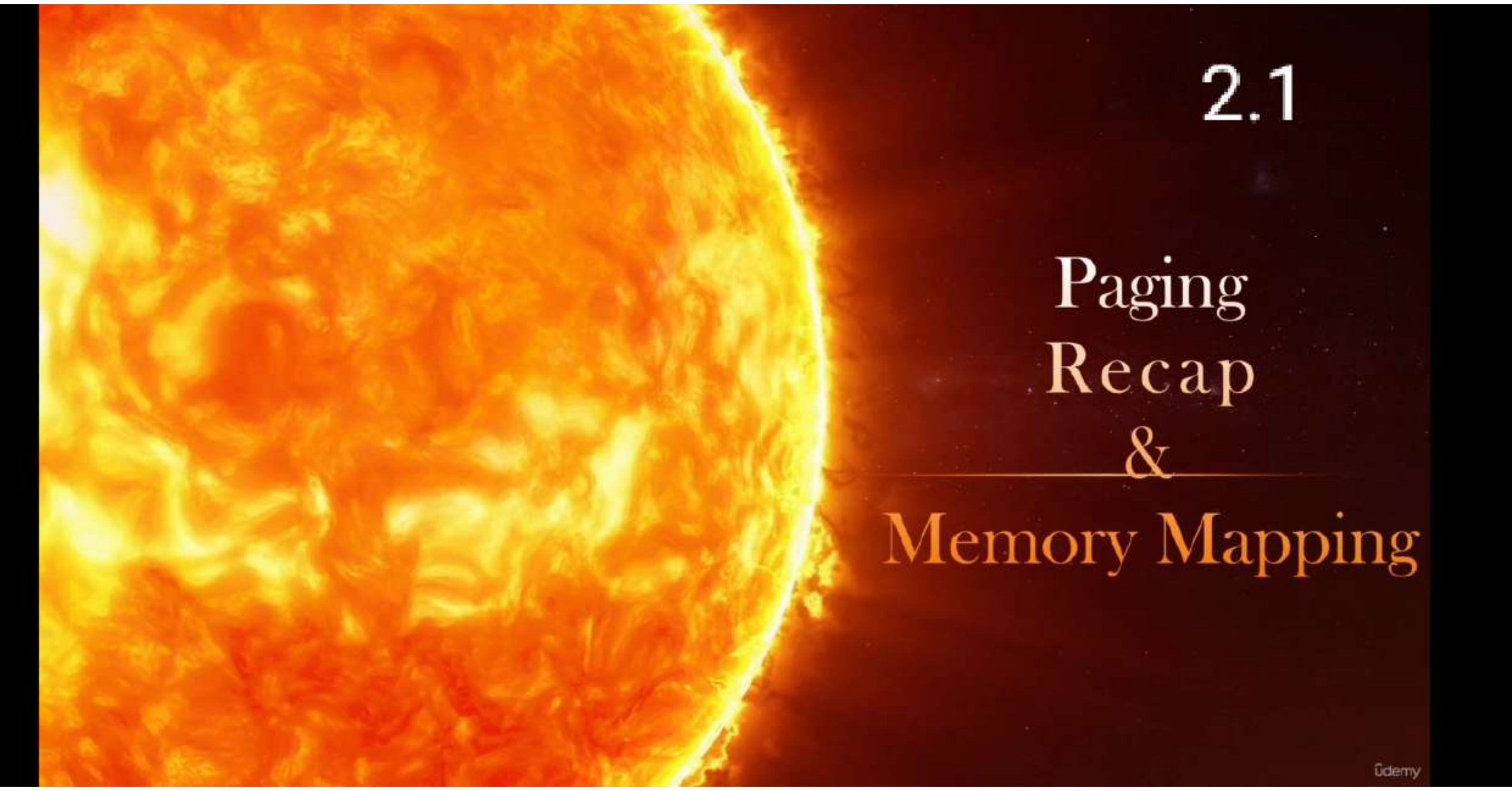
1. What is Memory Mapping ?
2. Goals ?
3. APIs to implement memory mappings
4. Shared Memory using Memory Mapping

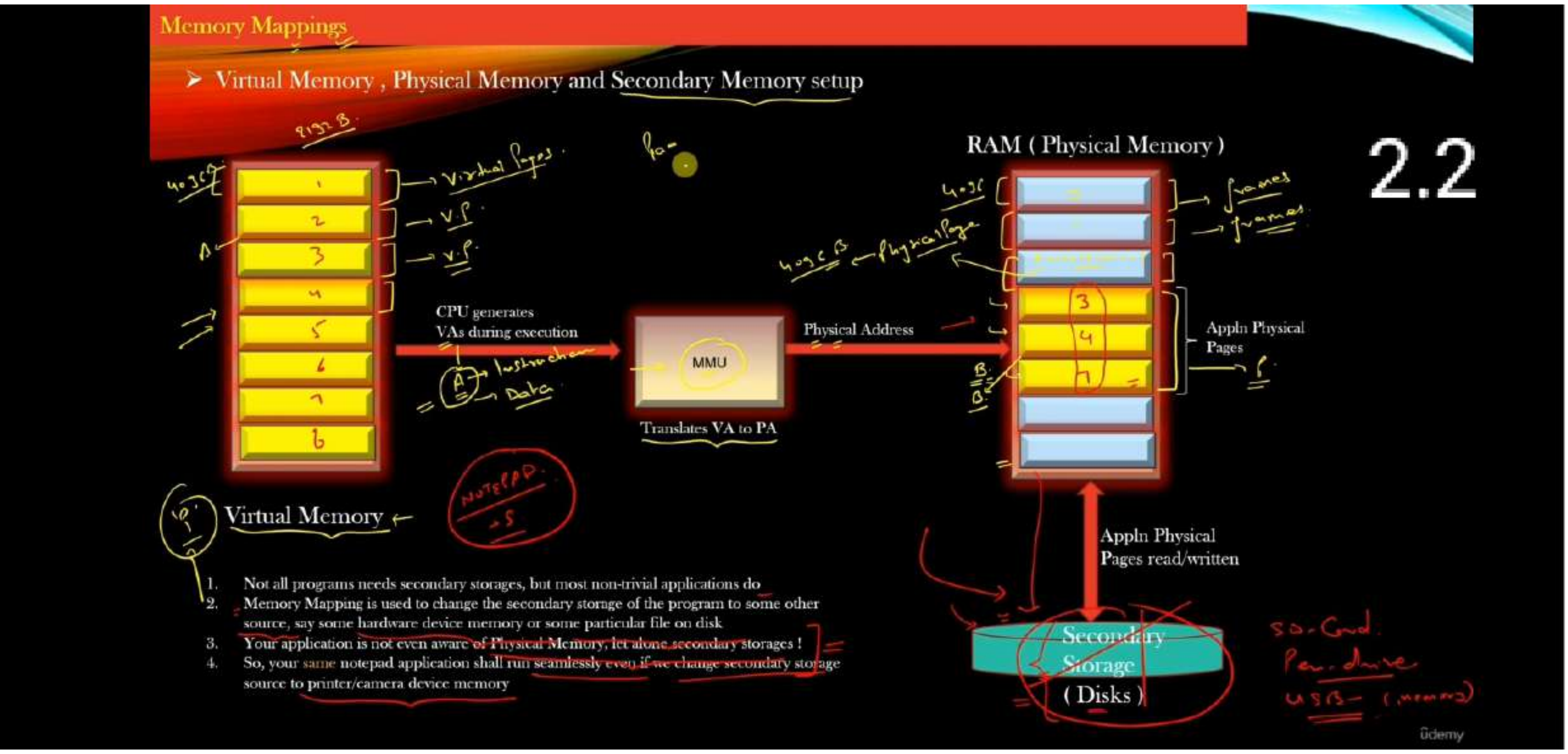
Pre-requisite : Paging and Virtual Memory Basics

Codes : www.github.com/sachinites/IPC/NewSHM

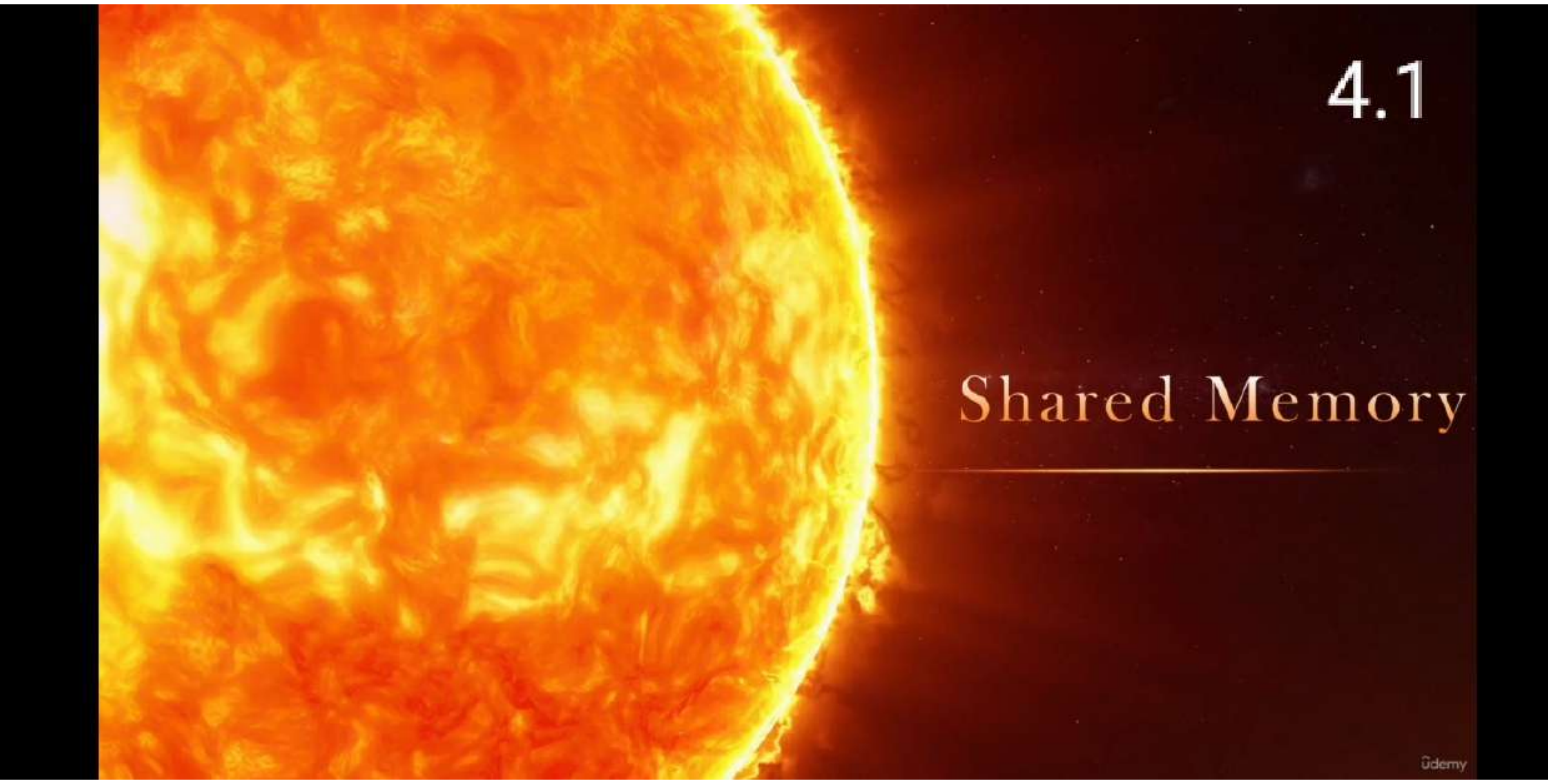


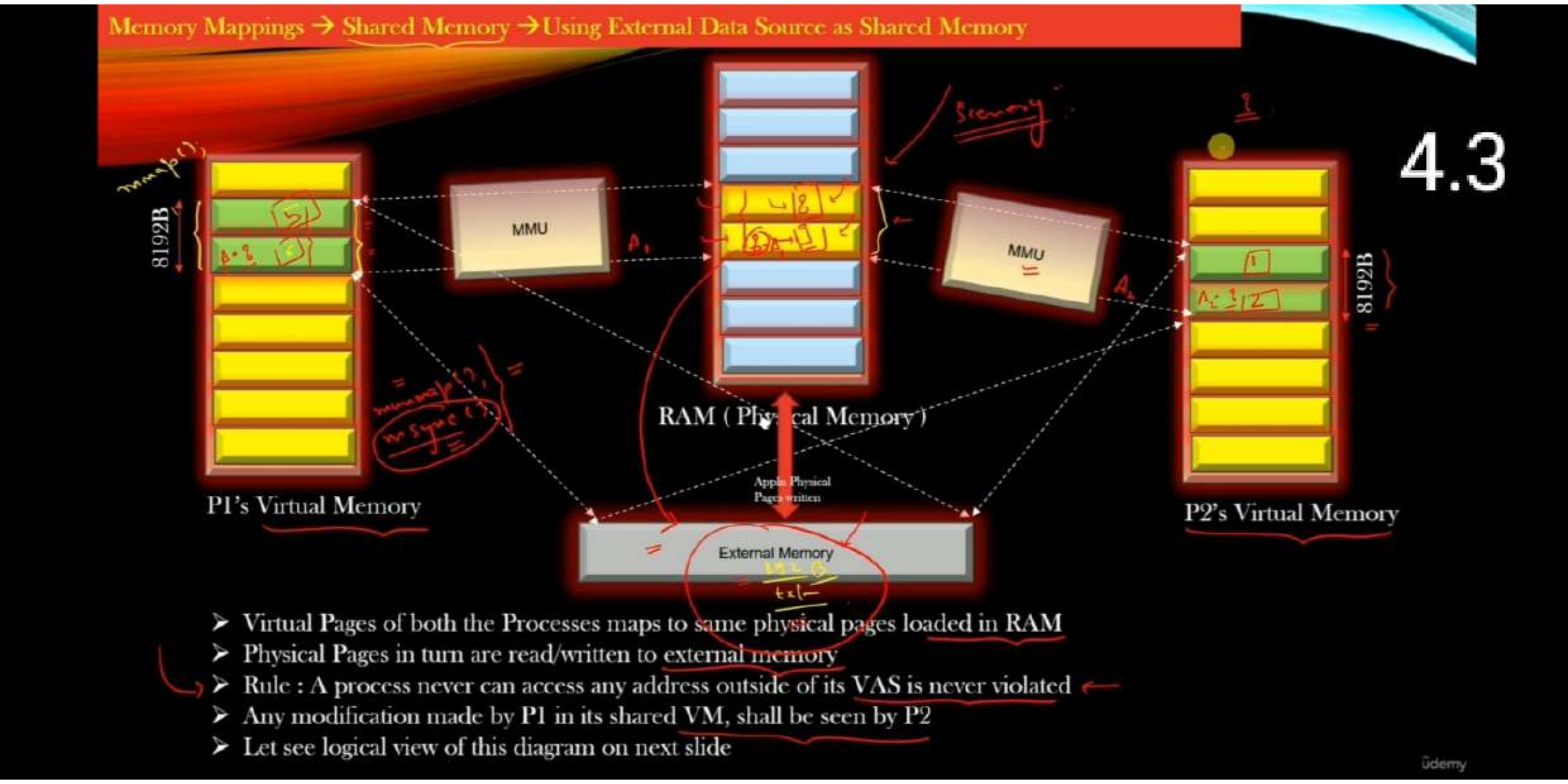
ūdemy


2.1
Paging
Recap
&
Memory Mapping
ûdemy

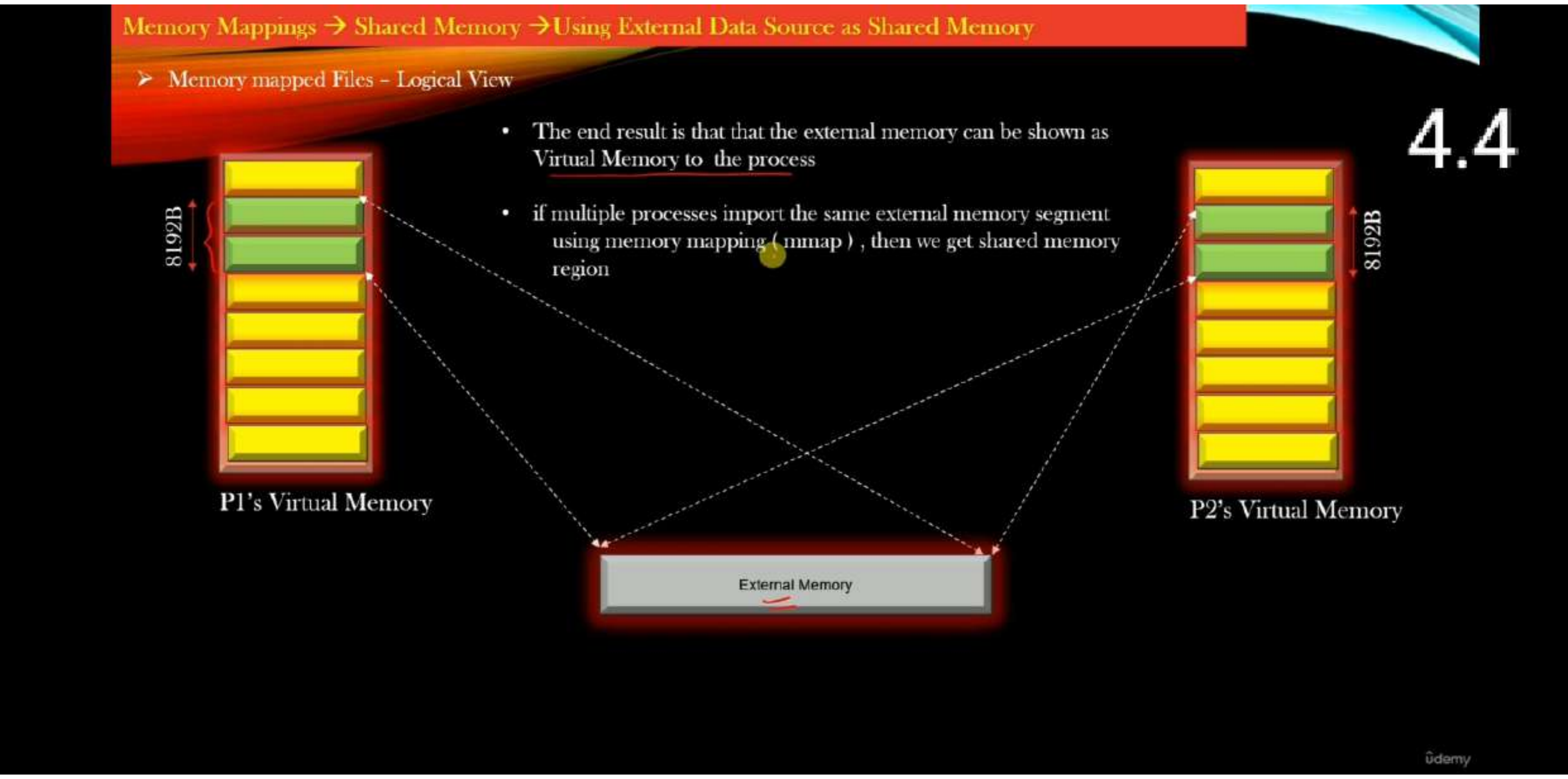


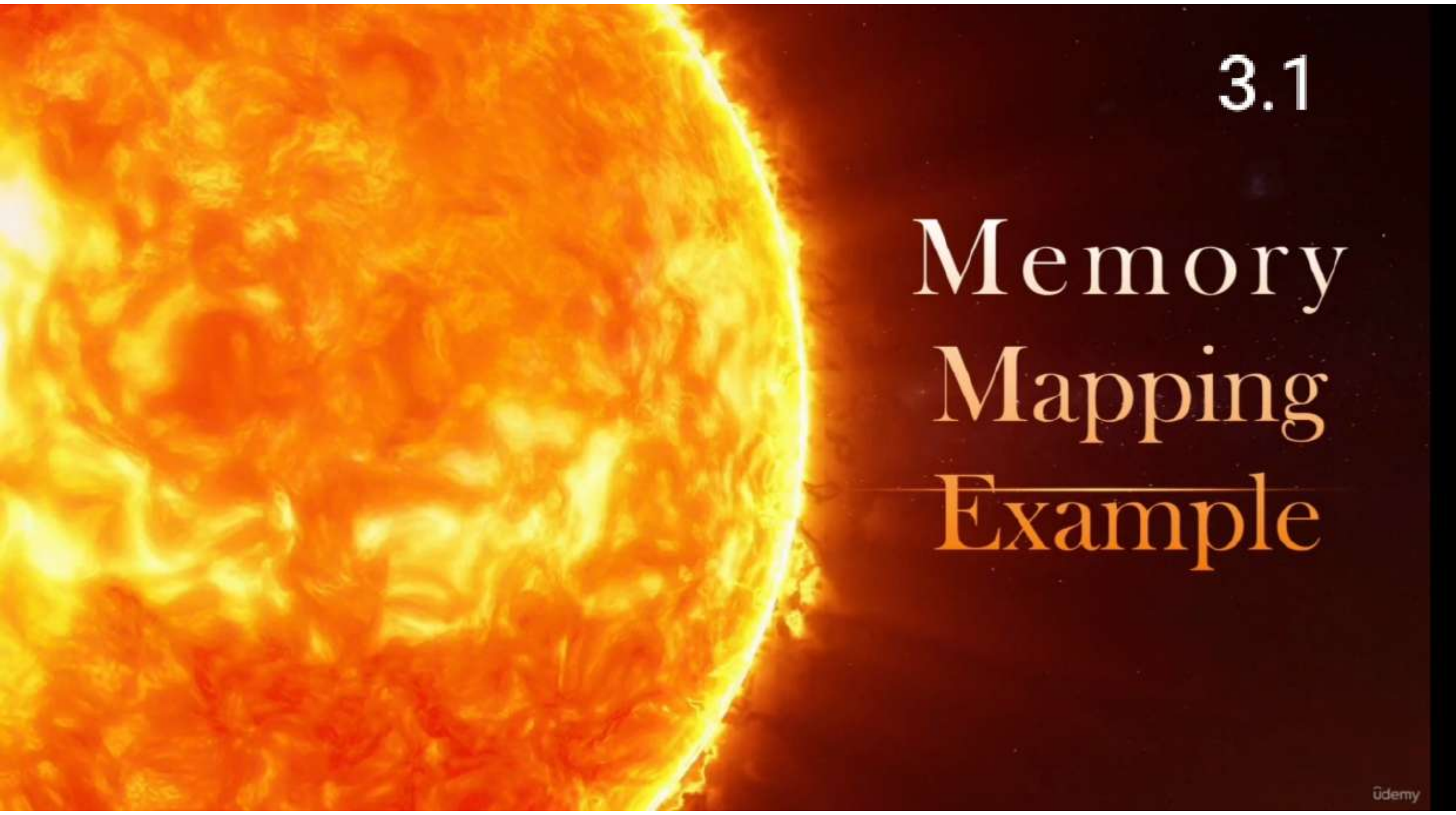
Memory Mappings
Virtual Memory , Physical Memory and Secondary Memory setup
2.2
RAM (Physical Memory)
CPU generates
VAs during execution
MMU
Translates VA to PA
Physical Address
Appln Physical
Pages
Appln Physical
Pages read/written
Secondary
Storage
(Disks)
Virtual Memory
1. Not all programs needs secondary storages, but most non-trivial applications do
2. Memory Mapping is used to change the secondary storage of the program to some other source, say some hardware device memory or some particular file on disk
3. Your application is not even aware of Physical Memory, let alone secondary storages !
4. So, your same notepad application shall run seamlessly even if we change secondary storage source to printer/camera device memory

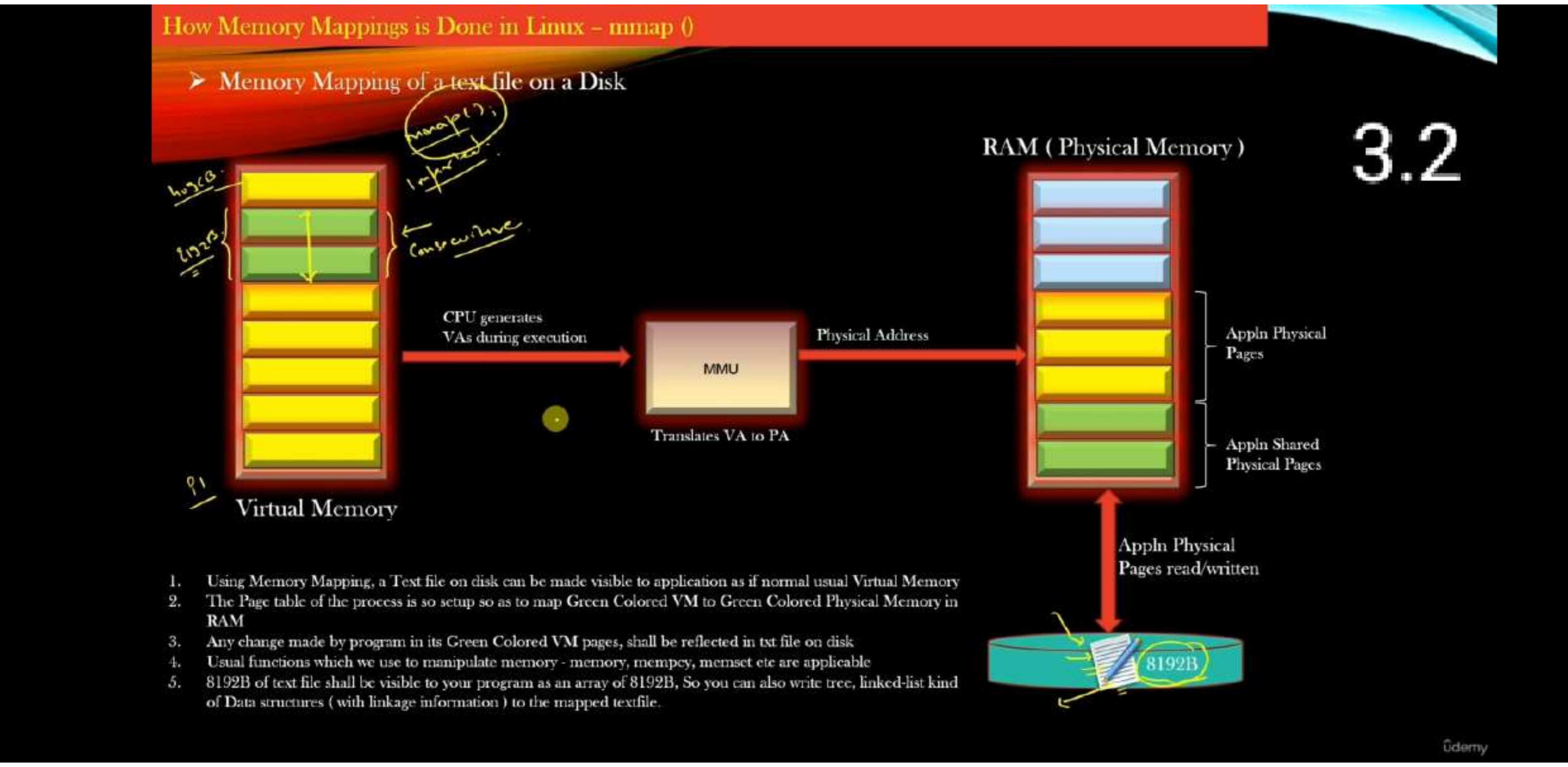

4.1
Shared Memory
ūdemy

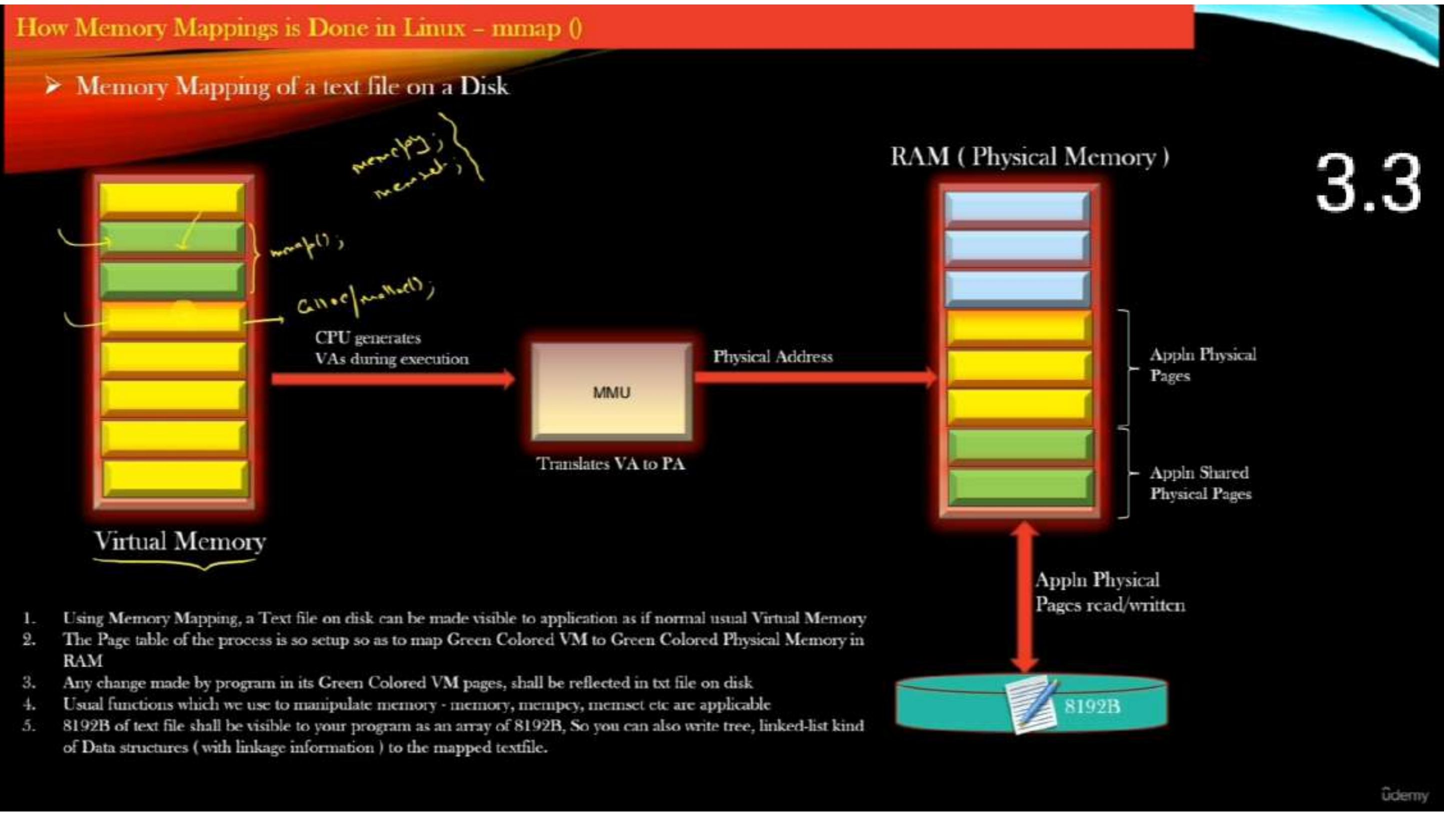


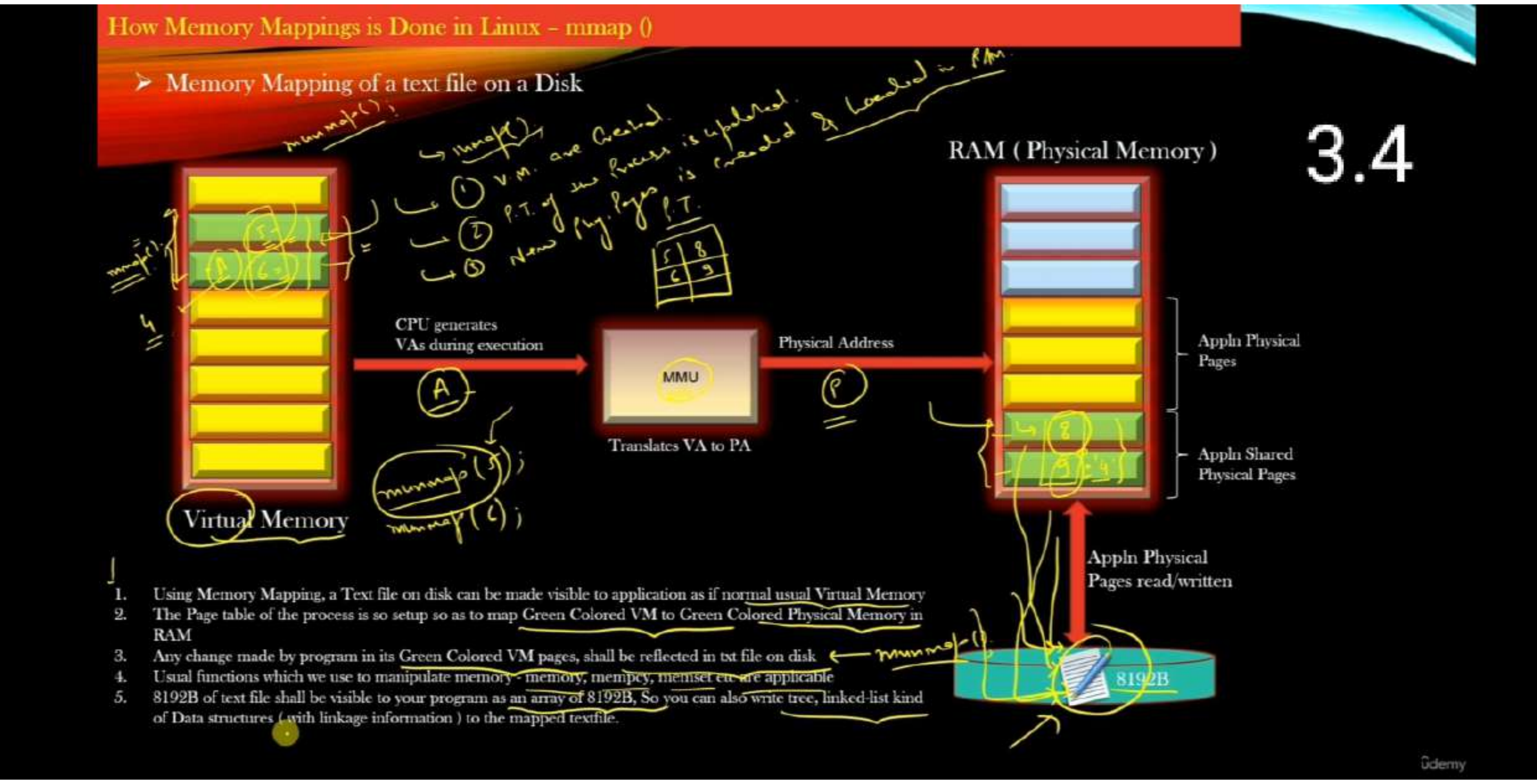
Memory Mappings → Shared Memory → Using External Data Source as Shared Memory
4.3
mmap()
8192B
MMU
MMU
RAM (Physical Memory)
Apply Physical
Pages written
External Memory
P1's Virtual Memory
P2's Virtual Memory
8192B
➢ Virtual Pages of both the Processes maps to same physical pages loaded in RAM
➢ Physical Pages in turn are read/written to external memory
➢ Rule : A process never can access any address outside of its VAS is never violated
➢ Any modification made by P1 in its shared VM, shall be seen by P2
➢ Let see logical view of this diagram on next slide
udemy


Memory Mappings → Shared Memory →Using External Data Source as Shared Memory
➢ Memory mapped Files – Logical View
4.4
• The end result is that that the external memory can be shown as Virtual Memory to the process
• if multiple processes import the same external memory segment using memory mapping (mmap) , then we get shared memory region
8192B
8192B
P1's Virtual Memory
P2's Virtual Memory
External Memory
ûdemy


3.1
Memory
Mapping
Example
ûdemy


How Memory Mappings is Done in Linux – mmap ()
➤ Memory Mapping of a text file on a Disk
3.2
mmap();
important
consecutive
Virtual Memory
CPU generates
VAs during execution
MMU
Translates VA to PA
Physical Address
RAM (Physical Memory)
Appln Physical
Pages
Appln Shared
Physical Pages
Appln Physical
Pages read/written
8192B
1. Using Memory Mapping, a Text file on disk can be made visible to application as if normal usual Virtual Memory
2. The Page table of the process is so setup so as to map Green Colored VM to Green Colored Physical Memory in RAM
3. Any change made by program in its Green Colored VM pages, shall be reflected in txt file on disk
4. Usual functions which we use to manipulate memory - memory, mempcy, memset etc are applicable
5. 8192B of text file shall be visible to your program as an array of 8192B, So you can also write tree, linked-list kind of Data structures (with linkage information) to the mapped textfile.
Udemy

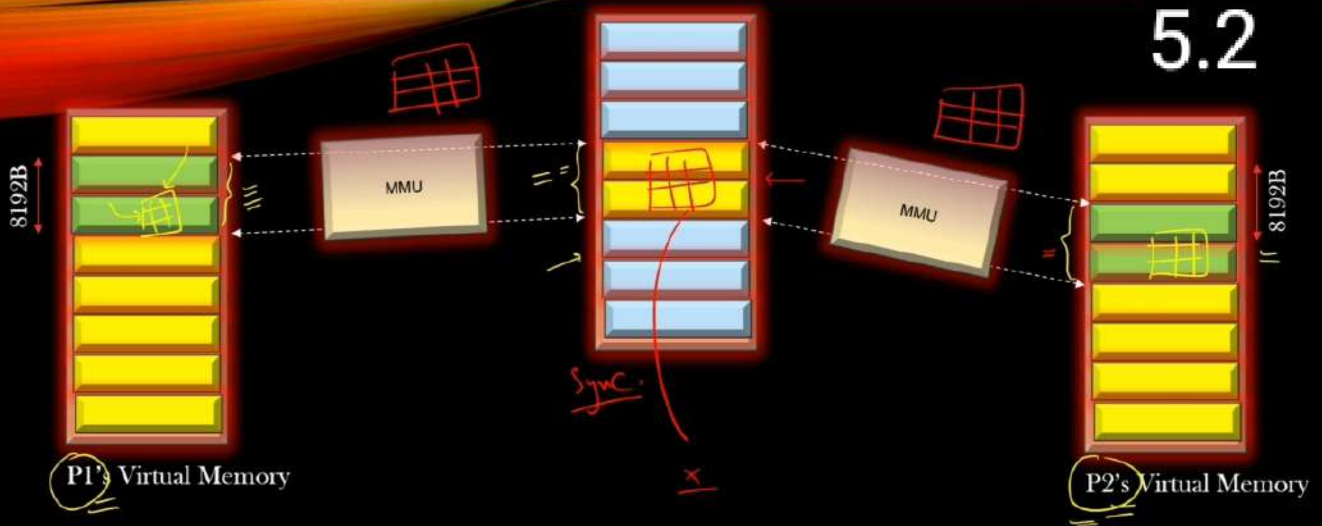

How Memory Mappings is Done in Linux – mmap ()
➢ Memory Mapping of a text file on a Disk
memcpy;
memset;
mmap();
calloc/malloc();
CPU generates
VAs during execution
MMU
Translates VA to PA
Physical Address
RAM (Physical Memory)
Appln Physical Pages
Appln Shared Physical Pages
Appln Physical Pages read/written
8192B
Virtual Memory
3.3
1. Using Memory Mapping, a Text file on disk can be made visible to application as if normal usual Virtual Memory
2. The Page table of the process is so setup so as to map Green Colored VM to Green Colored Physical Memory in RAM
3. Any change made by program in its Green Colored VM pages, shall be reflected in txt file on disk
4. Usual functions which we use to manipulate memory - memory, mempcy, memset etc are applicable
5. 8192B of text file shall be visible to your program as an array of 8192B, So you can also write tree, linked-list kind of Data structures (with linkage information) to the mapped textfile.
ûdemy


How Memory Mappings is Done in Linux – mmap ()
➤ Memory Mapping of a text file on a Disk
3.4
RAM (Physical Memory)
CPU generates
VAs during execution
MMU
Translates VA to PA
Physical Address
Appln Physical
Pages
Appln Shared
Physical Pages
Appln Physical
Pages read/written
8192B
Virtual Memory
1. Using Memory Mapping, a Text file on disk can be made visible to application as if normal usual Virtual Memory
2. The Page table of the process is so setup so as to map Green Colored VM to Green Colored Physical Memory in RAM
3. Any change made by program in its Green Colored VM pages, shall be reflected in txt file on disk
4. Usual functions which we use to manipulate memory - memory, mempcy, memset etc are applicable
5. 8192B of text file shall be visible to your program as an array of 8192B, So you can also write tree, linked-list kind of Data structures (with linkage information) to the mapped textfile.
Udemy


5.1
Using RAM
as
Shared Memory
udemy

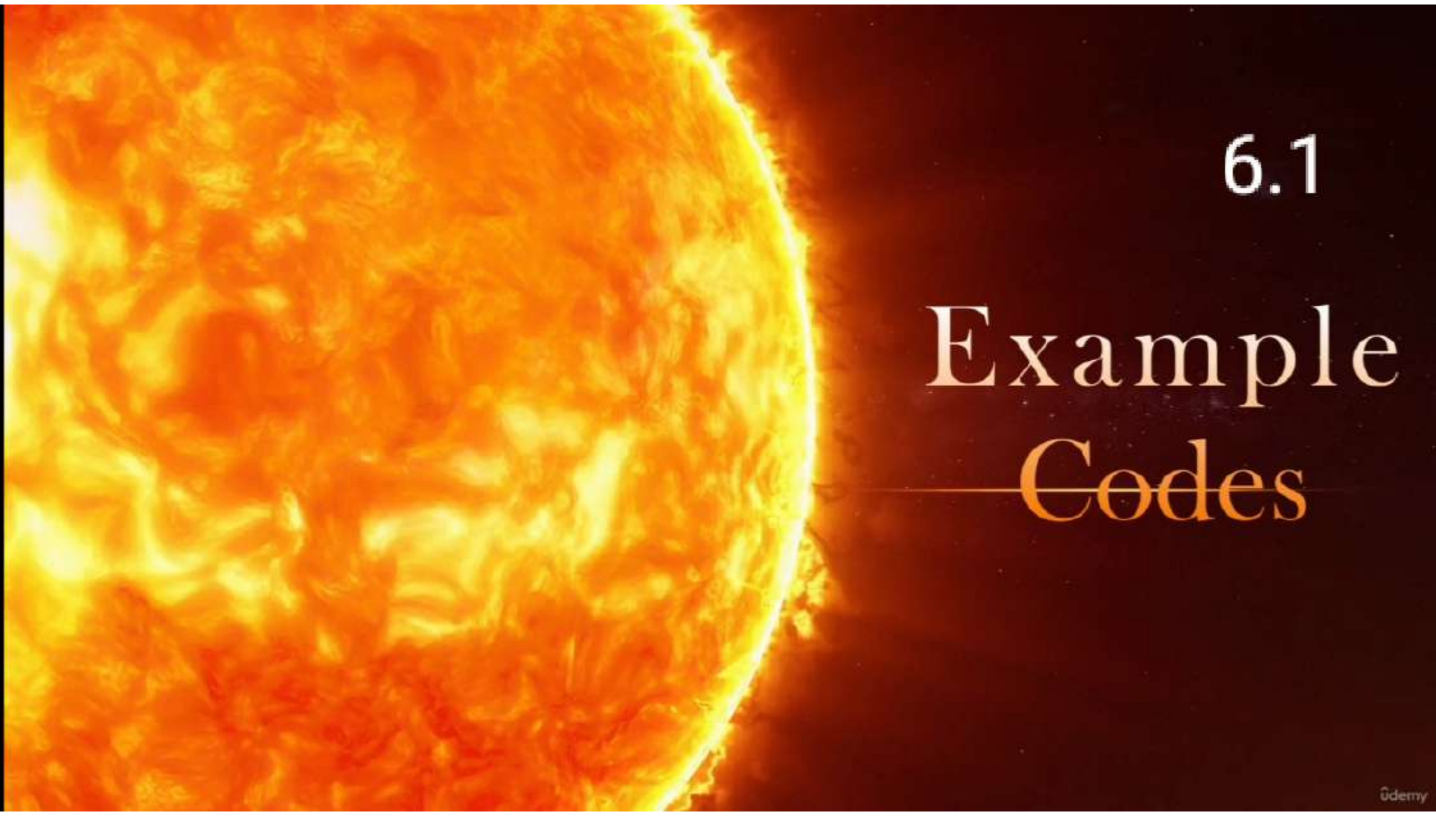
Memory Mappings → Shared Memory →Using RAM itself as Data Source

5.2



- Virtual Pages of both the Processes maps to same physical pages loaded in RAM
- Used Widely for IPC
- ~~Physical Pages in turn are read/written to external memory~~

Ûdemy


6.1
Example
Codes
ûdemy

6.2

- Example :
  - Using mmap() as substitute for malloc()
  - Using Text file as a data source
  - Using RAM as a Data Source
  - Memory Mapped being shared by Multiple Processes

Steps :

1. Initialize the Shared Memory segment shm_open()
2. Define the size of the SHM segment – ftruncate( )
3. Map the Shared Memory segment to Data Source – mmap( )
4. Use the shared Memory (read/write)
5. Destroy the mapping between process and Shared Memory segment – munmap( )
6. Destroy shared memory segment – shm_unlink( )

Udemy


7.1
mmap()
udemy

mmap_as_malloc_Demo1.c

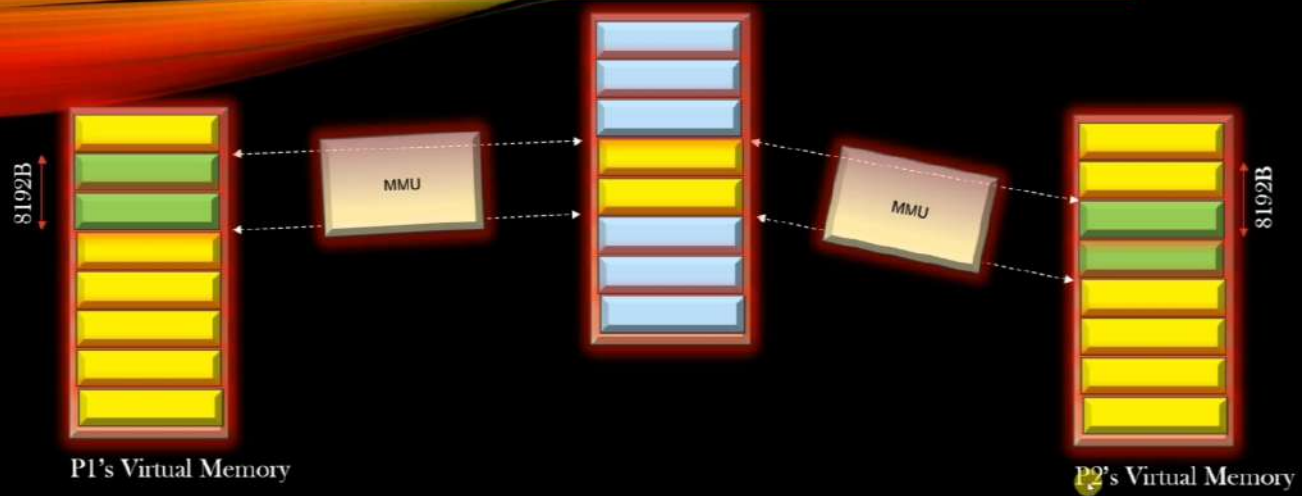
IPC > NewSHM > mmap_as_malloc_Demo1.c

```c
is no external data source or memory mapping involved. This example simply
shows how mmap() can be used as substitution for malloc()/free().
*/

#if 0
void *
mmap(void *addr, size_t length, int prot, int flags,
            int fd, off_t offset);

int munmap(void *addr, size_t length);
#endif

int
main(int argc, char **argv) {

int N=5;
int *ptr = mmap ( NULL,     /*  Le the OS choose the starting virtual address of the memory, just
                                like you do not have a control as to what address malloc() will return */
                N*sizeof(int),     /* Size of memory in bytes being requested for allocation */
                PROT_READ | /* Memory is Readable */
                PROT_WRITE, /* Memory is Writable */
                MAP_PRIVATE | /* This memory is not sharable with any other process, use MAP_SHARED instead */
                MAP_ANONYMOUS,   /* This memory is not mapped to extern data source, but only RAM by default */
                0,    /* FD 0, since no external data source is specified */
                0 ); /* offset value as zero, since no extern data source is specified */

if(ptr == MAP_FAILED){
   printf("Mapping Failed\n");
   return 1;
}

for(int i=0; i<N; i++)
   ptr[i] = i*10;

for(int i=0; i<N; i++)
   printf("[%d] ",ptr[i]);
```

7.2

Memory Mappings → Shared Memory →Using RAM itself as Data Source

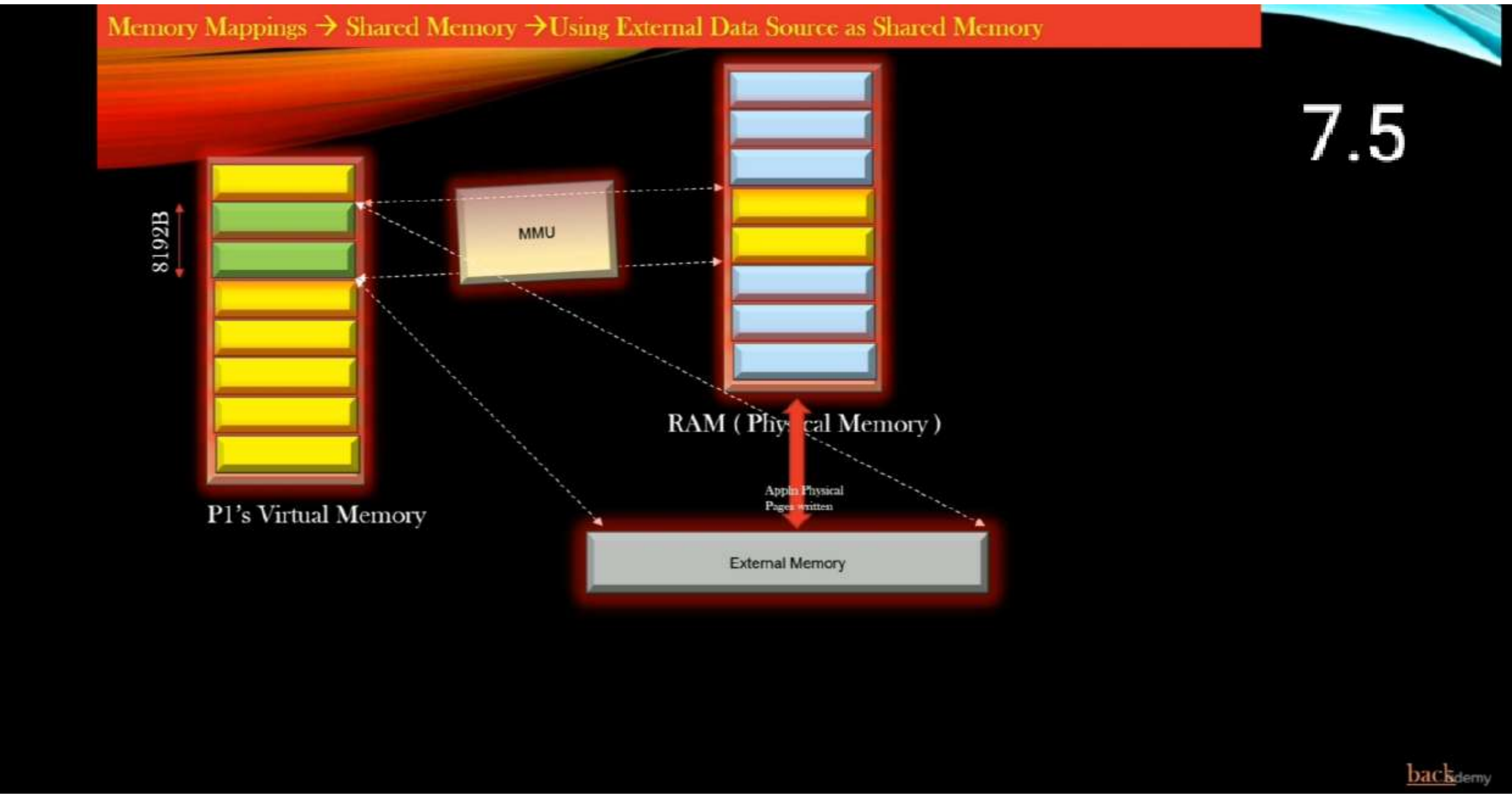
7.3



- Virtual Pages of both the Processes maps to same physical pages loaded in RAM
- Used Widely for IPC
- ~~Physical Pages in turn are read/written to external memory~~

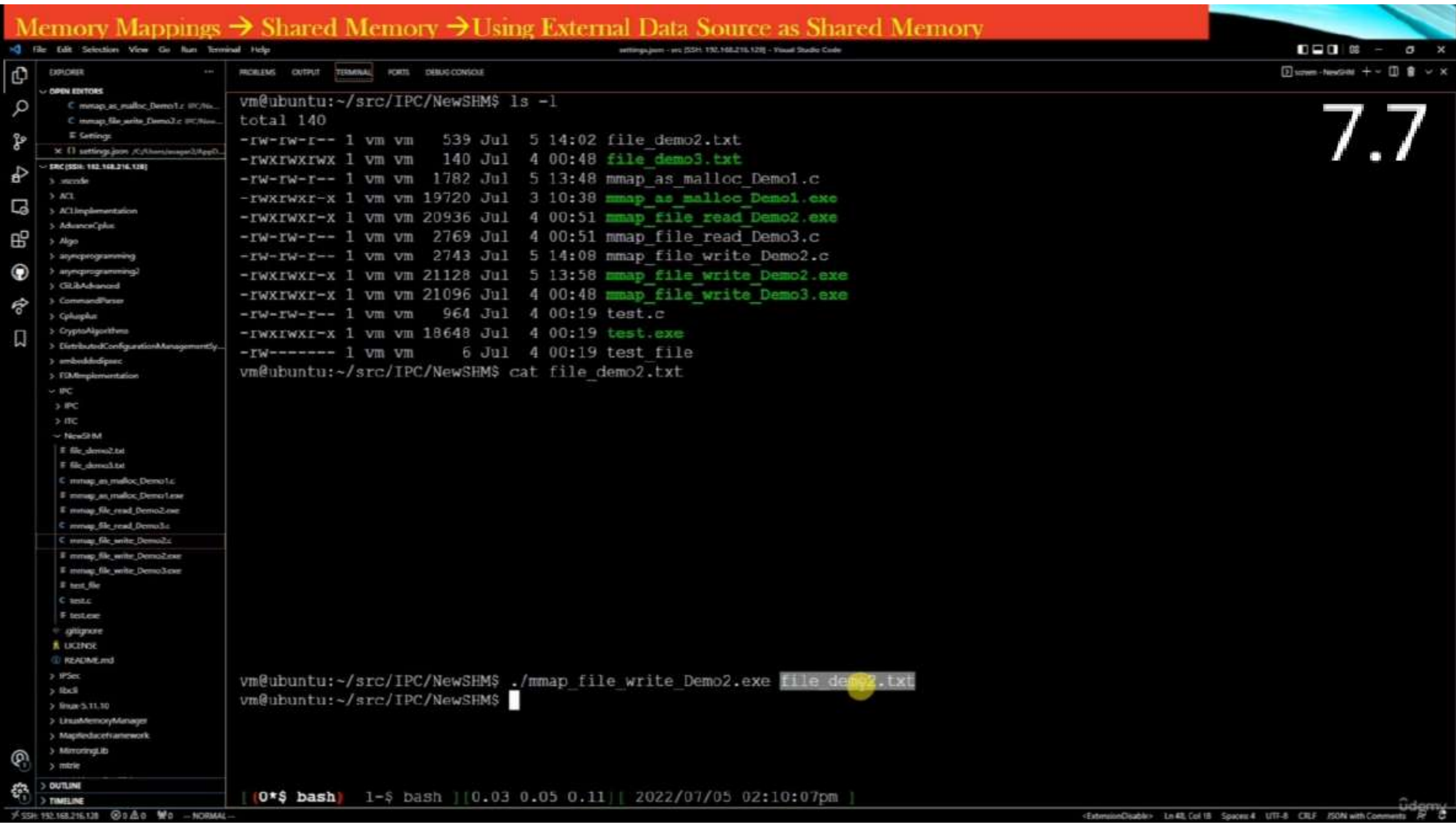
ûdemy

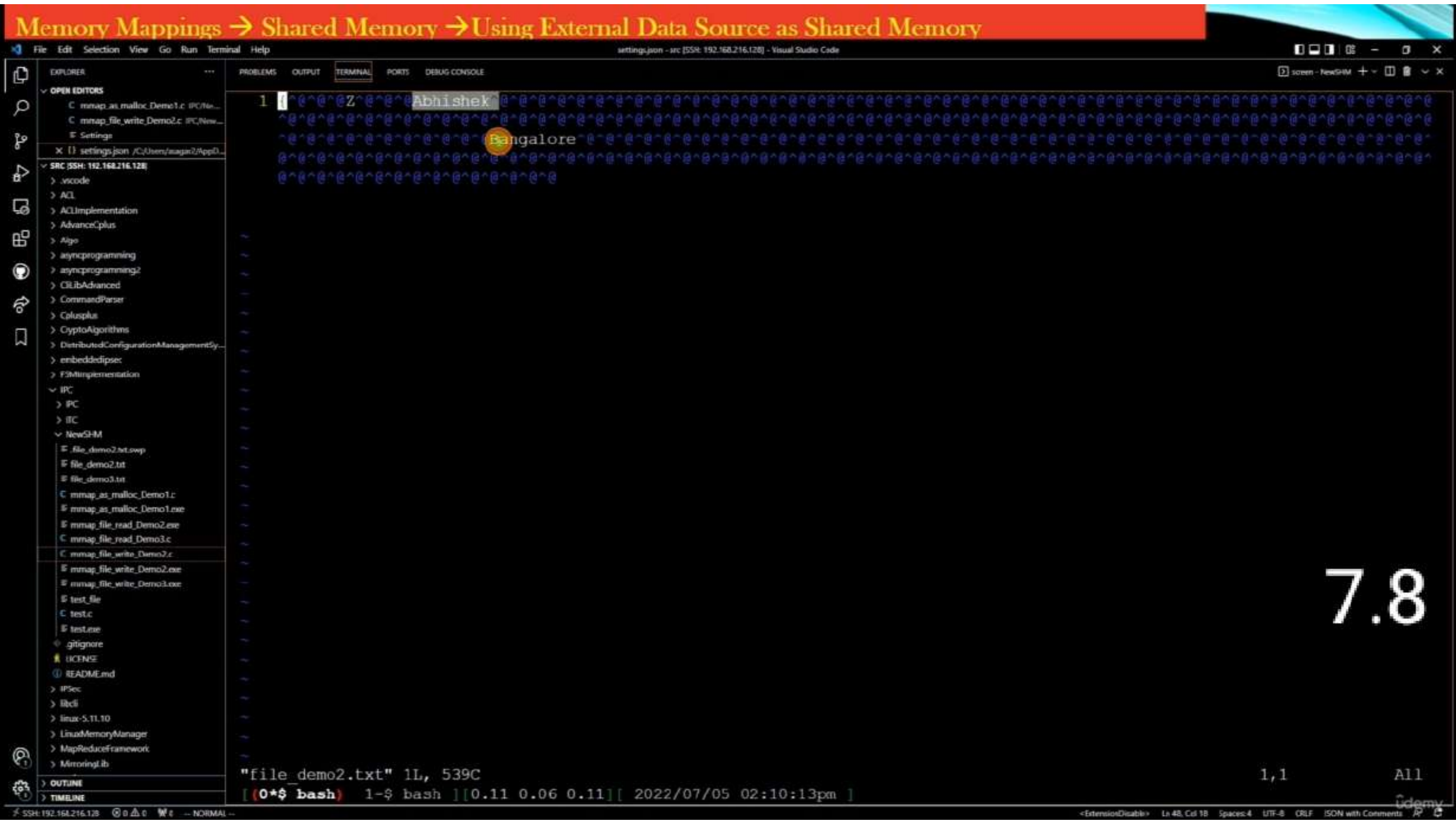

7.4
Mapping
~~External M~~emory
ūdemy

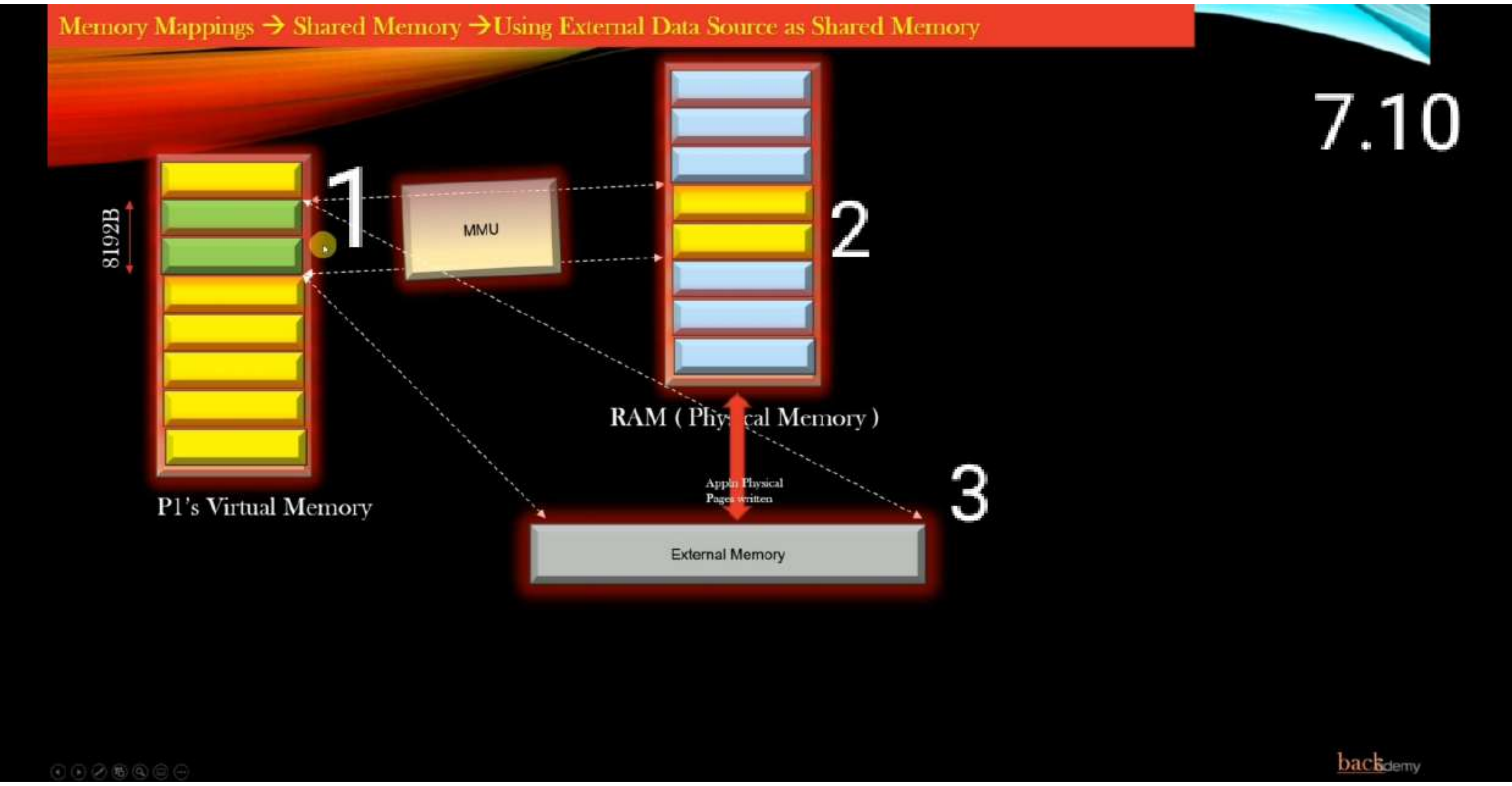


Memory Mappings → Shared Memory →Using External Data Source as Shared Memory
7.5
8192B
MMU
RAM (Phy cal Memory)
P1's Virtual Memory
Appln Physical
Pages written
External Memory


Memory Mappings → Shared Memory → Using External Data Source as Shared Memory
mmap_as_malloc_Demo1.c
mmap_file_write_Demo2.c
#include <errno.h>
#include <unistd.h>
#include <memory.h>
/* This program shows how mmap() can be used to memory map the text
file present on disk into process's Virtual address space */
static void breakpoint () {}
typedef struct student_ {
int roll_no;
int marks;
char name[128];
char city[128];
} student_t;
int
main(int argc, char *argv[]){
if(argc < 2){
printf("File path not mentioned\n");
exit(0);
}
const char *filepath = argv[1];
/* OPen the file in Read-Write OMode */
int fd = open(filepath, O_RDWR);
if(fd < 0){
printf("\n\"%s \" could not open\n",
filepath);
exit(1);
}
/* Extract the size of the file */
7.6


Memory Mappings → Shared Memory → Using External Data Source as Shared Memory
7.7
vm@ubuntu:~/src/IPC/NewSHM$ ls -l
total 140
-rw-rw-r-- 1 vm vm 539 Jul 5 14:02 file_demo2.txt
-rwxrwxrwx 1 vm vm 140 Jul 4 00:48 file_demo3.txt
-rw-rw-r-- 1 vm vm 1782 Jul 5 13:48 mmap_as_malloc_Demo1.c
-rwxrwxr-x 1 vm vm 19720 Jul 3 10:38 mmap_as_malloc_Demo1.exe
-rwxrwxr-x 1 vm vm 20936 Jul 4 00:51 mmap_file_read_Demo2.exe
-rw-rw-r-- 1 vm vm 2769 Jul 4 00:51 mmap_file_read_Demo3.c
-rw-rw-r-- 1 vm vm 2743 Jul 5 14:08 mmap_file_write_Demo2.c
-rwxrwxr-x 1 vm vm 21128 Jul 5 13:58 mmap_file_write_Demo2.exe
-rwxrwxr-x 1 vm vm 21096 Jul 4 00:48 mmap_file_write_Demo3.exe
-rw-rw-r-- 1 vm vm 964 Jul 4 00:19 test.c
-rwxrwxr-x 1 vm vm 18648 Jul 4 00:19 test.exe
-rw------- 1 vm vm 6 Jul 4 00:19 test_file
vm@ubuntu:~/src/IPC/NewSHM$ cat file_demo2.txt
vm@ubuntu:~/src/IPC/NewSHM$ ./mmap_file_write_Demo2.exe file_demo2.txt
vm@ubuntu:~/src/IPC/NewSHM$
[(0*$ bash) 1-$ bash][0.03 0.05 0.11][2022/07/05 02:10:07pm]


Memory Mappings → Shared Memory → Using External Data Source as Shared Memory
Abhishek
Bangalore
"file_demo2.txt" 1L, 539C
1,1
All
(0*$ bash) 1-$ bash][0.11 0.06 0.11][2022/07/05 02:10:13pm]
7.8



Memory Mappings → Shared Memory →Using External Data Source as Shared Memory
7.10
8192B
1
MMU
2
P1's Virtual Memory
3
External Memory
backademy

7.9

1 . mmap() used for import external text file in virtual memory
2 . unmap() used to destroy virtual memory like ram space
3 . msync() used to copy the data from physical memory (ram) to text file


8.1
DESIGN CONSTRAINT

Design Constraints for using Shared Memory as IPC

8.2

- **Recommendation when it is good to use Shared Memory IPC :**
  - Shared Memory approach of IPC is used in a scenario where :
    - Exactly one processes is responsible to update the shared memory (publisher process)
    - Rest of the processes only read the shared memory (Subscriber processes)
    - The frequency of updating the shared memory by publisher process should not be very high
      - For example, publisher process update the shared memory when user configure something on the software

1
2

- If multiple processes attempts to update the shared memory at the same time, then it leads to write-write conflict :

➢ We need to handle this situation using Mutual Exclusion based Synchronization

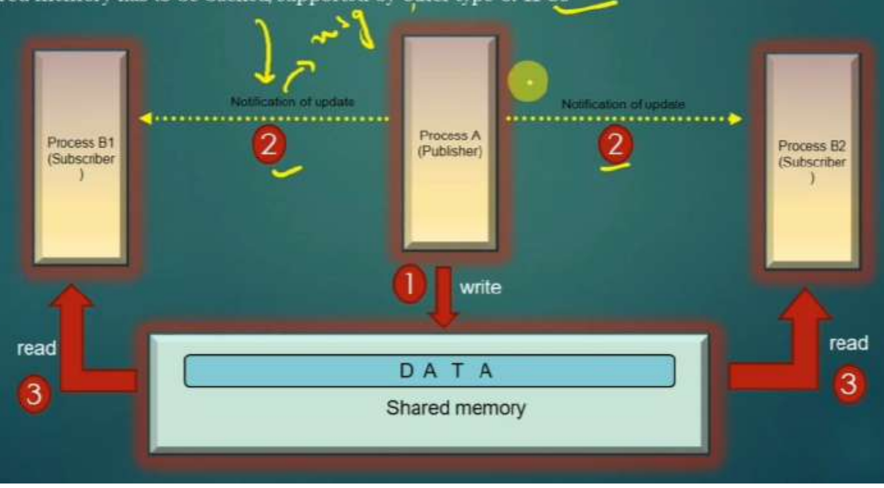
➢ Synchronization comes at the cost of performance
Because we put the threads to sleep (in addition to their natural CPU preemption) in order to prevent concurrent access to critical section

Udemy

Design Constraints for using Shared Memory as IPC

8.3

- When publisher process update the shared memory :
    - The subscribers would not know about this update
    - Therefore, After updating the shared memory, publisher needs to send a notification to all ~~publishers~~ subscribers which states that "shared memory has been updated"
    - After receiving this notification, Subscribers can read the updated shared memory and update their internal data structures, if any
    - The notification is just a small message, and can be sent out using other IPC mechanisms, such as Unix domain sockets Or Msg Queues
    - Thus IPC using shared memory has to be backed/supported by other type of IPCs



9.1

DATA SYNCHRONIZATION

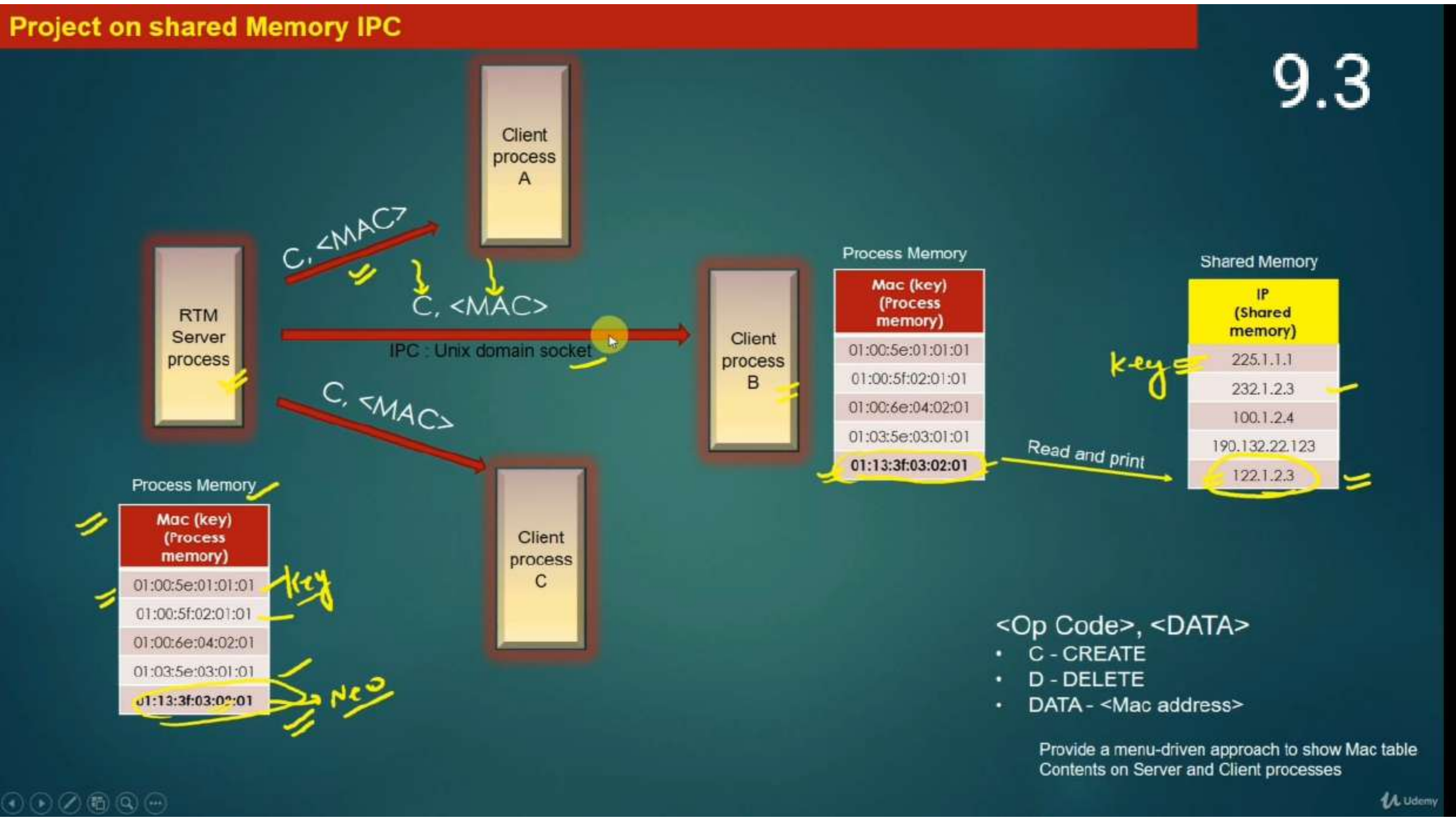
SHARED MEMORY PROJECT

Project on shared Memory IPC

9.2

- We shall extend the project that you have done on Unix domain sockets
- In Unix domain socket project, the routing table manager server process synchronized routing table to all connected clients using Unix Domain sockets IPC
- In this project, the routing table manager server process also maintains another table called ARP table and it synchronizes it to all subscribed clients using Shared Memory IPC
- Let us discuss the project step by step. The functionality of Previous project will stay as it is, no changes

Udemy

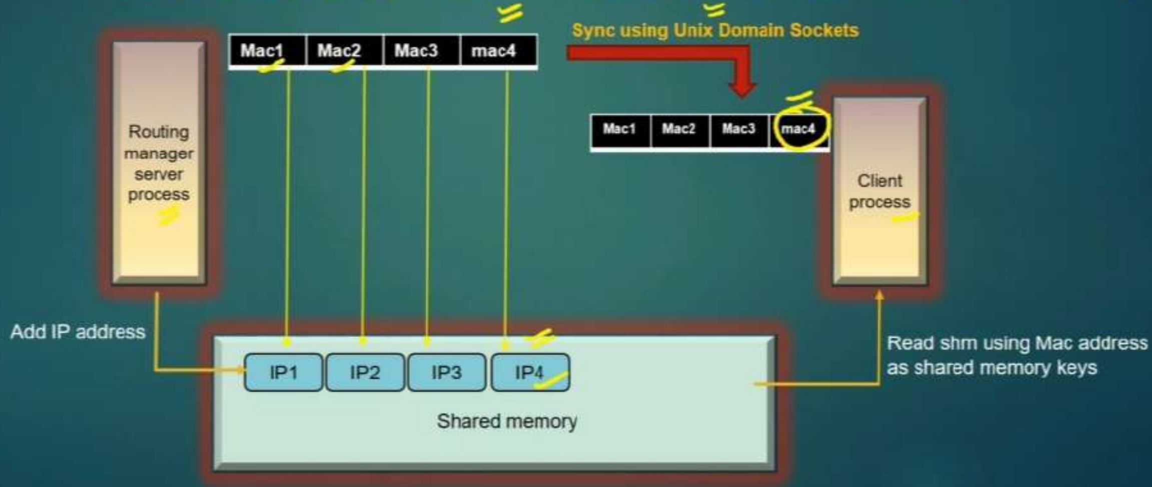


Project on shared Memory IPC
9.3
Client process A
RTM Server process
C, <MAC>
C, <MAC>
IPC : Unix domain socket
C, <MAC>
Client process B
Client process C
Process Memory
Mac (key) (Process memory)
01:00:5e:01:01:01
01:00:5f:02:01:01
01:00:6e:04:02:01
01:03:5e:03:01:01
01:13:3f:03:02:01
Read and print
Shared Memory
IP (Shared memory)
225.1.1.1
232.1.2.3
100.1.2.4
190.132.22.123
122.1.2.3
key
Process Memory
Mac (key) (Process memory)
01:00:5e:01:01:01
01:00:5f:02:01:01
01:00:6e:04:02:01
01:03:5e:03:01:01
01:13:3f:03:02:01
key
New
<Op Code>, <DATA>
C - CREATE
D - DELETE
DATA - <Mac address>
Provide a menu-driven approach to show Mac table Contents on Server and Client processes

Project on shared Memory IPC

9.4

- All the processes – Server and clients stored only the shared memory key (the Mac address) in its internal data structure
- Server process adds the Data – the ip address in the shared memory corresponding to the mac address (key)
- Server process then syncs only the mac address (shm key) to rest of the connected clients using Unix domain sockets
- Client having received this new key (the mac address), stores the mac address in their private list. Using this mac address as key they can also access the corresponding ip address which was added by Server process in shared memory
- When the fresh clients get connected to server process, besides routing table, server also syncs the entire local mac address list to new client using unix domain socket



Udemy