OS Memory

Sept 10^{th}, 2018

Some changes to Class MemMgr

Add the memory array which will represent the memory we are going to manage

We use numpy.

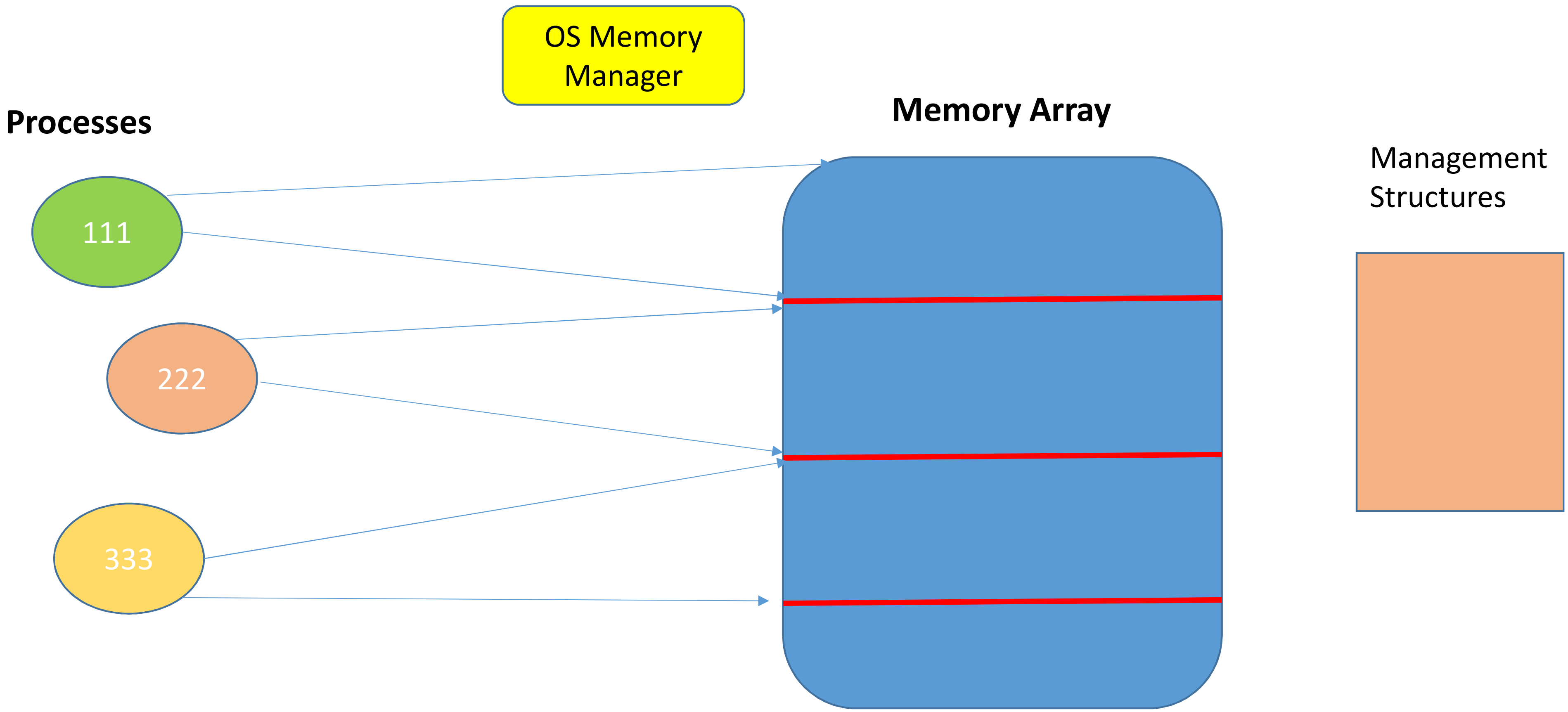
mem = np.zeros(dshape = (row_size, column_size), dtype='int8')

**Use class methods as we are only using two common data structures.
Sometimes not the best way, but it is the simplest for getting to know memory management**

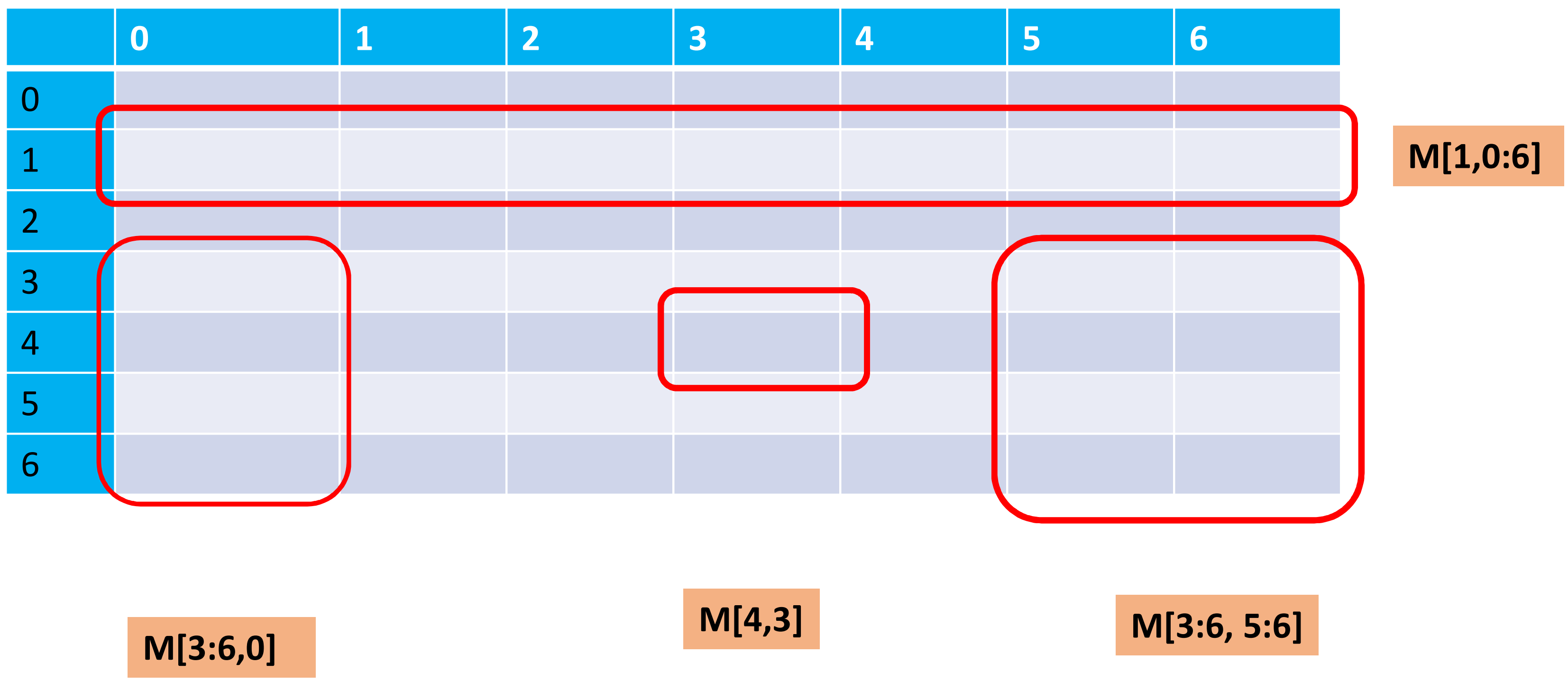
OS Memory

Sept 12, 2018

So far, simulating an OS Memory Manager



Array addressing scheme



Class Exercise: class-sept-12-a

Bring up Notebook
Download class912a.ipynb from moodle

Declare a 6x6 array with all zeros.

1. **Write ’11’ diagonally**
2. **Write ‘22’ on column idx 1**

Follow the order of instructions.

3. Write row of ‘33’ to row idx 3
4. Write ‘44’ to diagonal
5. Fill the empty cells with ‘55’

Instructions from 1 to 4 overwrite existing values

Do this on your own

0	0	1	2	3	4	5
0	11					
1		11				
2			11			
3				11		
4					11	
5						11

02	0	1	2	3	4	5
0	11	22	55	55	55	44
1	55	22	55	55	44	55
2	55	22	11	44	55	55
3	33	33	44	33	33	33
4	55	44	55	55	11	55
5	44	22	55	55	55	11

find_free_space() algorithm

Index	PID
0	0
1	0
2	0
3	0
4	0
5	0
6	0

1. Loop for row until the first empty spot

```
found = false
for r in range(0, row_size):
    if m[r,0] == 0:
        found = true
```

2. When found,
check whether there are n blocks free

```
found = false
for r in range(0, row_size):
    if m[r,0] == 0:
        # check for n free blocks
        found = true
        for x in range(r, n):
            if m[x,0] == 0:
                continue
            else:
                found = false
                break
        if found:
            start_idx = r
            break
return start_idx, start_idx+n
```

Let’s change the memory array to 7x6

Simulate this with your test program

Index	PID
0	0
1	0
2	0
3	0
4	0
5	0
6	0

get_mem(222,2)

Index	PID
0	222
1	222
2	0
3	0
4	0
5	0
6	0

get_mem(333,1)

Index	PID
0	222
1	222
2	333
3	0
4	0
5	0
6	0

get_mem(444,1)

Index	PID
0	222
1	222
2	333
3	444
4	0
5	0
6	0

release_mem(333)

Index	PID
0	222
1	222
2	0
3	444
4	0
5	0
6	0

get_mem(555,2)

Index	PID
0	222
1	222
2	0
3	444
4	555
5	555
6	0

Cleaning up implementation

- Make a class MemMgmt for managing the management structures
  - Use classmethod
- Define a class MemCommon to store common and constant values
- Replace all numbers with some constant name.

```
@classmethod
def get_mem(cls, pid, nbr):
    start_index,end_index = MemMgmt.find_free_mem(pid,nbr)
    if start_index == MemCommon.Invalid:
         # no memory
          return MemCommon.NULLARRAY
          # or raise an exception
    else:
        viewa = MemMgr.memarray[start_index:end_index,
                               MemCommon.startColumn:MemCommon.endColumn]
        # check for error
        return viewa

@classmethod
def release_mem(cls, pid):
     MemMgmt.release_mem(cls)
       # do we expect errors in release
```

Code Assignment: code-9-12-a

Three files:

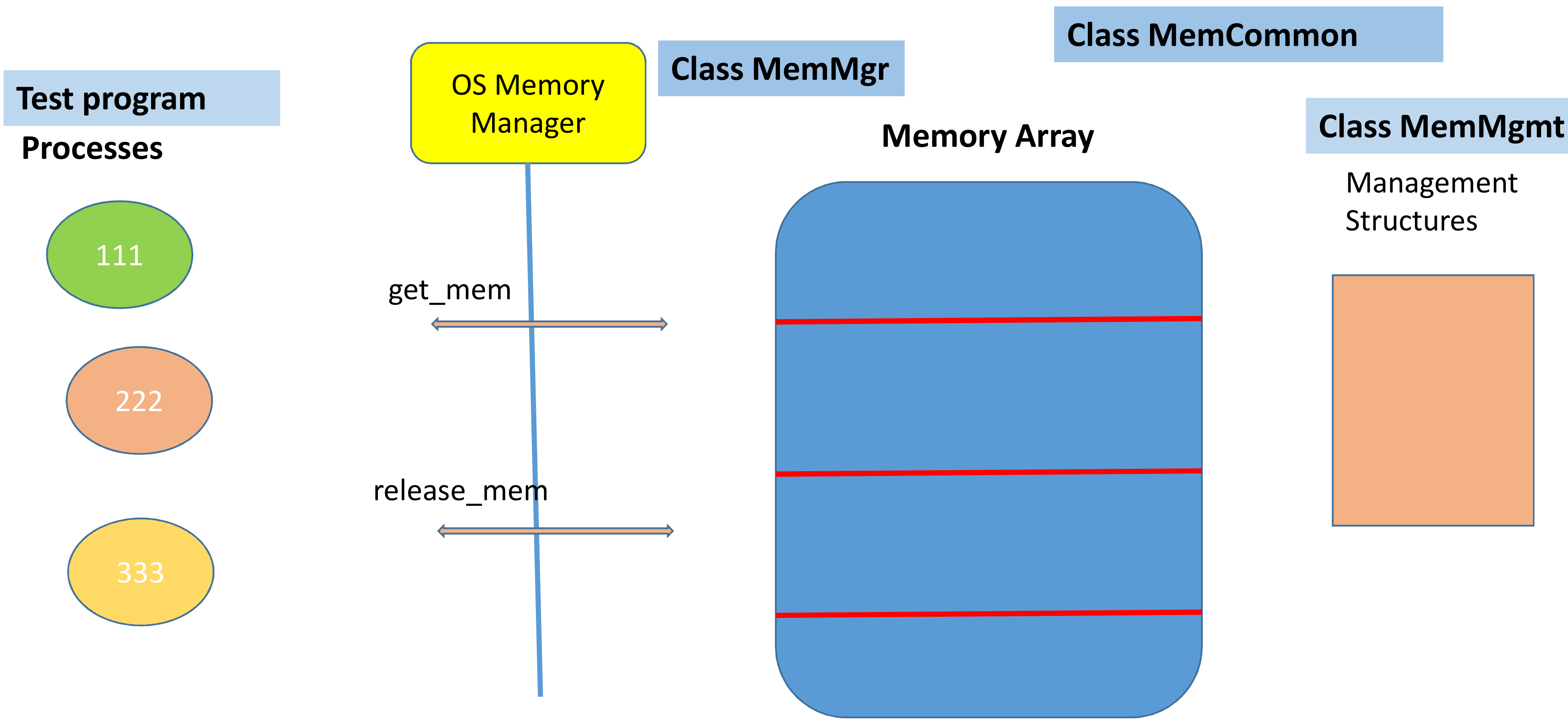
1. **memmgr.py [class MemMgr]**
2. **memcommon.py [class MemCommon]**
3. **memmgmt.py [class MemMgmt]**

DUE: Sept 17, 2018

OS Memory

Sept 14, 2018

OS Memory Manager so far



Things will go always perform normally and correctly

In reality, this is not always the case – Errors and Exceptions

BIG ASSUMPTION: You have a working version of the memmgr.py with a good testmem1.py

Testmem2.py

Index	PID
0	0
1	0
2	0
3	0
4	0
5	0
6	0

get_mem(222,3)

Index	PID
0	222
1	222
2	222
3	0
4	0
5	0
6	0

get_mem(333,2)

Index	PID
0	222
1	222
2	222
3	333
4	333
5	0
6	0

get_mem(444,1)

Index	PID
0	222
1	222
2	222
3	333
4	333
5	444
6	0

release_mem(333)

Memory is fragmented

Index	PID
0	222
1	222
2	222
3	0
4	0
5	444
6	0

get_mem(555,3)

Error here

Two ways to handle errors:

- Normal return error from a call
- Try and Except

```
aa = MemMgr.get_mem(222,3)
```

What kind of error would get_mem() return?

Let's say it returns a zero size array
as get_mem always return an array

```
aa = MemMgr.get(222,3)
If aa.size == 0:
     # what to do?
      exit() ?? Or call some error display function
```

```
Try
     aa = MemMgr.get_mem(222,3)

except:
      #what to do?
      exit()?
      # or something
```

Let's use Exception error

Error from Chrome when you have too many tabs open

Not enough memory to open this page

Try closing other tabs or programs to free up memory.

Learn more

Send feedback

Code Assignment: code-9-14-a

Use try except

- Make changes to your code to handle errors
  - Raise exception
  - Make changes to your test code to generate the error
  - Files:
    1. Memmgr.py
    2. Memcommon.py
    3. Memmgmt.py
    4. Testmem2.py – show the error expected
    5. Output2.txt

Memory Policy: For now keep it simple

1. **Not enough memory to fulfill request: get_mem to raise exception**
2. **If get_mem is called a 2nd time by Process, exception**
3. **If release_mem – if pid does not have any memory assigned, no error**

Changes to the test program testmem2.py to handle errors

Reasoning about Error handling and policy issues

- If a process calls get_mem() twice, would that be an error?

Can we interpret get_mem() the 2nd time as asking for more memory?
- **If there is a free block “below” can we just give it?**
- **What changes are required?**

We assume that The process knows Best.

Is it better to have a new method more_mem()?
- **What kind of changes are required for this method for the process and management structures?**

How to incorporate this feature in your implementation of MM?

Index	PID
0	0
1	0
2	0
3	0
4	0
5	0
6	0

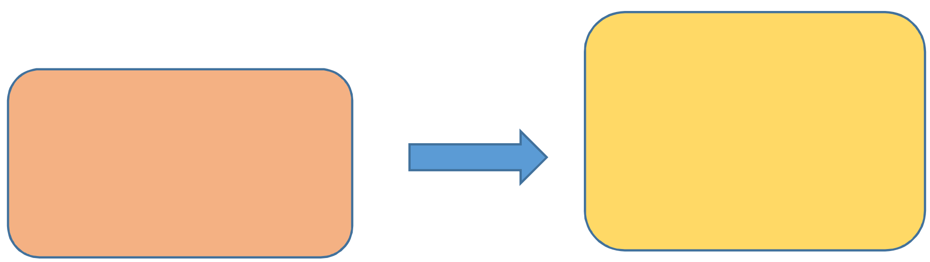
get_mem(222,3)

Index	PID
0	222
1	222
2	222
3	0
4	0
5	0
6	0

get_mem(222,1)

Index	PID
0	222
1	222
2	222
3	222
4	0
5	0
6	0

Adding to the management structures
Is easy enough, but how do you
tell the process to move to the "new" array



What are the logistics issues?
- implementation issues
Assume for simplicity:
- that something is doing the moving
or copying of content from one space to another

What about this scenario?

Index	PID
0	222
1	222
2	333
3	0
4	0
5	0
6	0

get_mem(222,3)

Process is still running.

Changing addresses for a running process is really difficult.

What if the process is not running or active?

Can we suspend an active process?

YES, we can

BUT be careful. If the process is an active process and is Interactive, then it would not be nice?

Release_mem()

- If release_mem() is called but there is no memory to release, is that an error?

Code Assignment: code-sept-14-b

Submit two files:
1. **Code914a.py**
2. **Output1.txt**

Swap file – store the contents of memory

- File: code914a
- Folder: main
- Instructions:
  1. Create an array 6x6 initialized to zero
  2. Get a view-a of 2x6
  3. Change first row to ‘22’ and 2nd to ’44’
  4. Store this view-a in a binary file.
  5. Get another view-b of 3x6
  6. Restore the binary file into this new view-b
  7. Change 3rd row to ’33’
  8. Print contents of view-b into output1.txt
  9. Print contents of big array into output1.txt

ASSUMPTION: Someone is storing the registers

Use this later for moving memory around to make space

Final view of memory after step 7

0	1	2	3	4	5
1	22	22	22	22	22
2	44	44	44	44	44
3	22	22	22	22	22
4	44	44	44	44	44
5	33	33	33	33	33

Sample test program to cause an error

Index	PID
0	0
1	0
2	0
3	0
4	0
5	0
6	0

get_mem(222,3)

Index	PID
0	222
1	222
2	222
3	0
4	0
5	0
6	0

get_mem(333,2)

Index	PID
0	222
1	222
2	222
3	333
4	333
5	0
6	0

get_mem(444,1)

Index	PID
0	222
1	222
2	222
3	333
4	333
5	444
6	0

release_mem(333)

Index	PID
0	222
1	222
2	222
3	0
4	0
5	444
6	0

get_mem(555,3)

This should generate an error

Observe behavior of request and memory availability

- Assumption: contiguous memory
- Requests ask for 3 blocks, 2 blocks and 1 block
- SUGGESTION: Why don’t we make partitions where partition is a certain number of block memory
  - So, one partition consists of 3 blocks, another partition of 2 blocks, etc
  - Would this be better??

Scheme-2b (Fixed Partition)

RAM

We call each memory area a Partition.
Each partition has a fixed size.
We need to keep track of memory used.

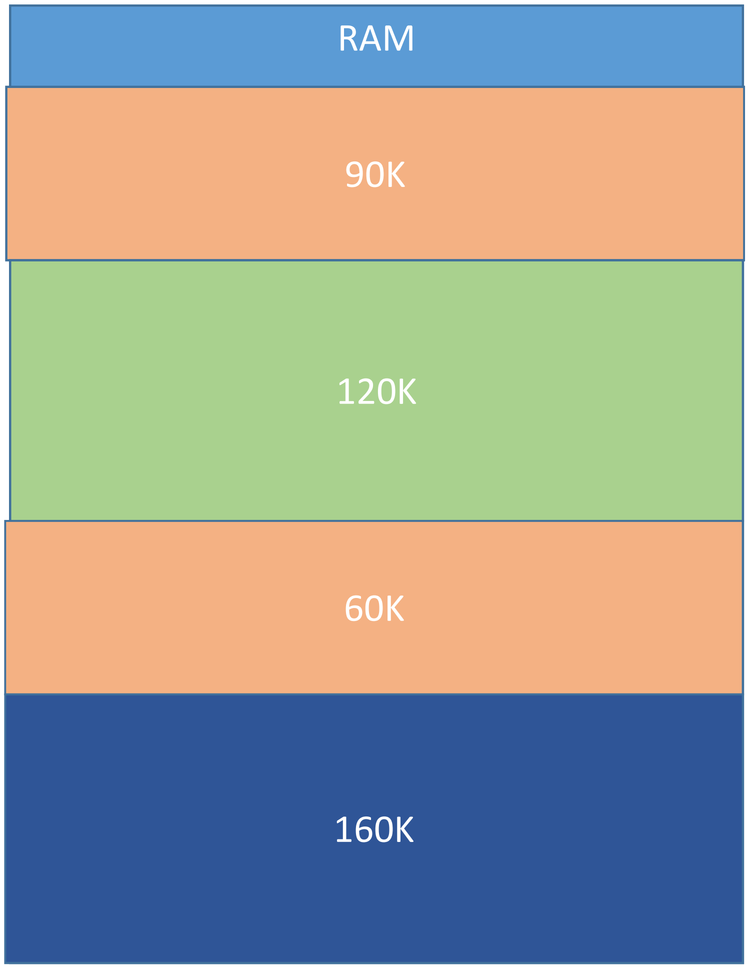
PARTITION MEMORY TABLE

Size	Start Address	Name of Job	Status
60K	100K	JOB 1	BUSY
200K	180K	JOB 2	BUSY
100K	280K		FREE
50K	380K	JOB 3	BUSY

Size of partition is static. Meaning the size cannot be changed.

JOB 3

Only way to change the size is to reboot the computer.



The sizes of partitions are fixed at system start.

Class exercise: on scheme-2b

- How to implement Scheme-2b using the code assignment
- Spend 5 mins thinking how to implement this
  - What needs to be changed?

Issues with scheme-2

- Memory sizes are fixed, and only way to change them is to either reboot the system, or clear memory and start again
- Some jobs may not be able to find the right memory partition because the memory partition sizes are fixed.
- To change the size we have to reboot.

- We need a scheme where memory sizes can be dynamic upon on request or demand.

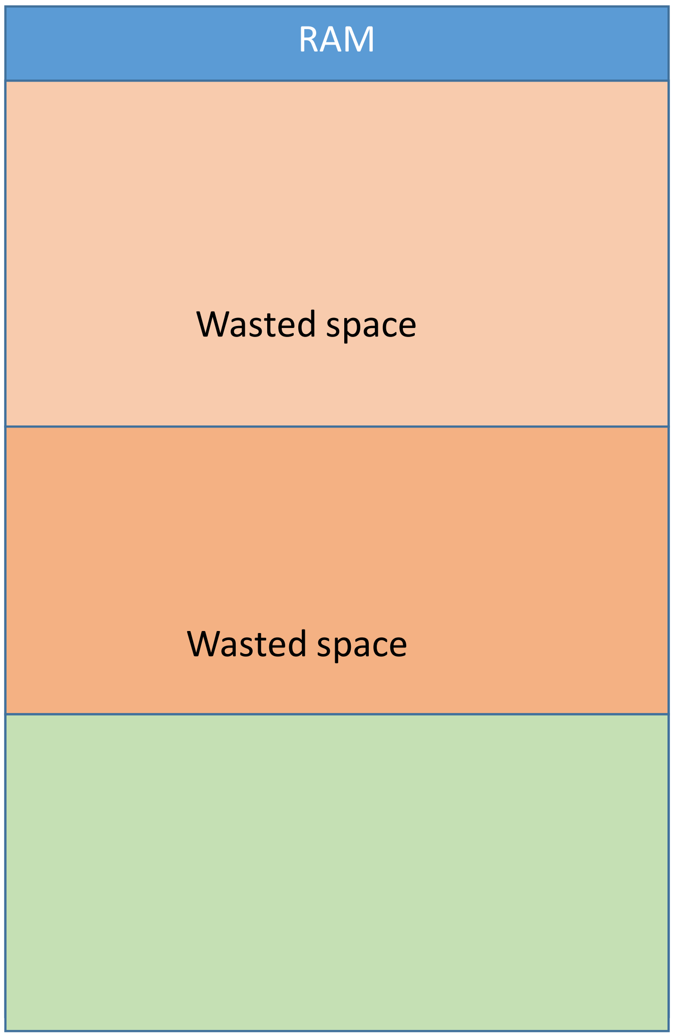
Scheme-3: Dynamic Partition

Your code is based on this model.
Use the idea of a block of memory.
Process **requests** number of blocks.

- Don’t predetermined the size of the memory partition
- Only allocate the memory on request
  - When a job requests 100K of memory
  - Allocate 100K of contiguous memory to job

- This solves the problem with fixed partition on having to reboot to adjust the size of memory partition
- It does prevent wasted memory

Two ways to fit job into memory for fixed and dynamic partition



JOB 4

JOB 3

First fit all

Lots of wasted space

First fit Allocation

Find the smallest partition to fit

Slower performance in finding memory
Better efficiency in use Of memory

When a job is finished, it is removed from memory

This is called DeAllocation

For fixed partition, the job is just removed and the size of partition remains the same.

For dynamic partition, the job is removed and the manager will try to join two adjacent memory partitions into one big partition.

Issues with these 3 schemes

- Each scheme requires that the whole program be stored in memory in a single contiguous block within a partition.

- Can we divide the program into multiple smaller chunks?
  - YES, we can.
  - This brings us to the topic of Virtual Memory