CS347 Lecture #9

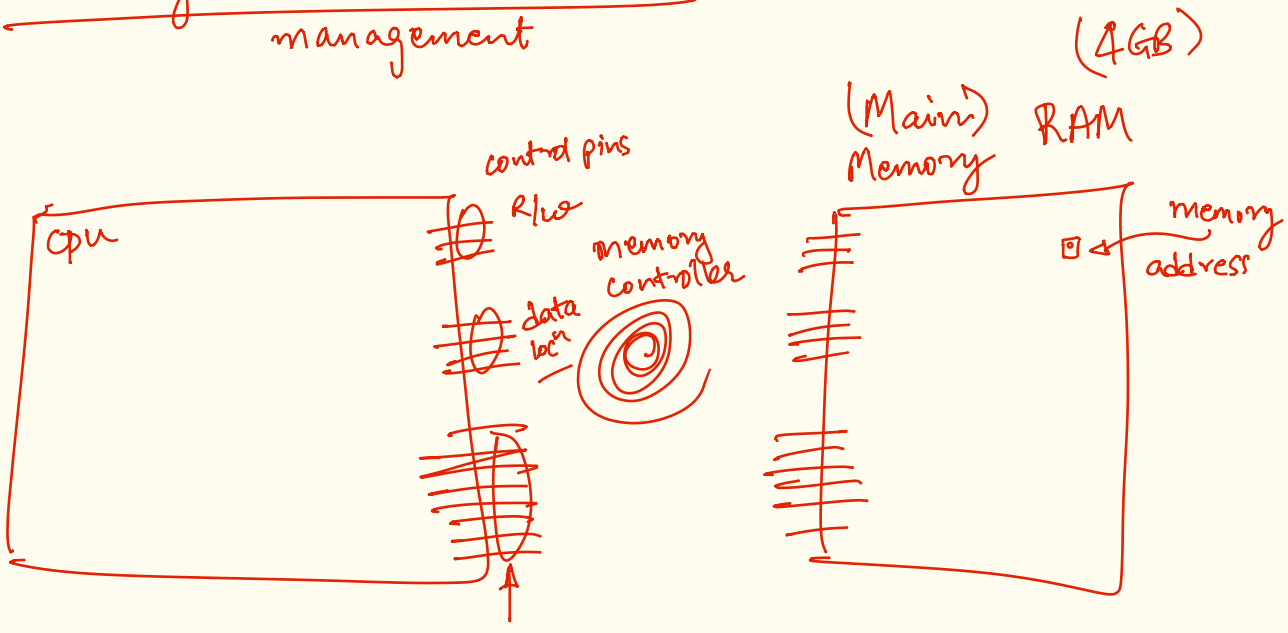
the process abstraction

multiplexing
sharing

- decoupled logic program development from execution
- cpu virtualization
  - scheduling.
    - context save & restore.

⊛ the memory virtualization management

ISA view



K-bit wide — each bit is 0 or 1
2^K addresses

K = 16 ~ 2^{16}

K = 32 ~ 2^{32} ~ 4GB

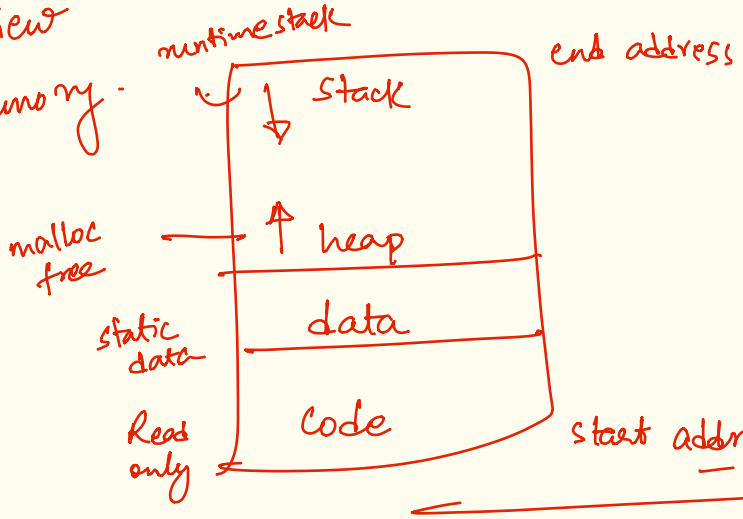
K = 64 ~ 2^{64} ~

} max. addressable range of the CPU

Q) Why do processes need memory?

- data ~ static / dynamic
  - malloc
  - free
- code ~ read only instructions
- stack
  - dynamic based on calls during execution

process-view of memory.



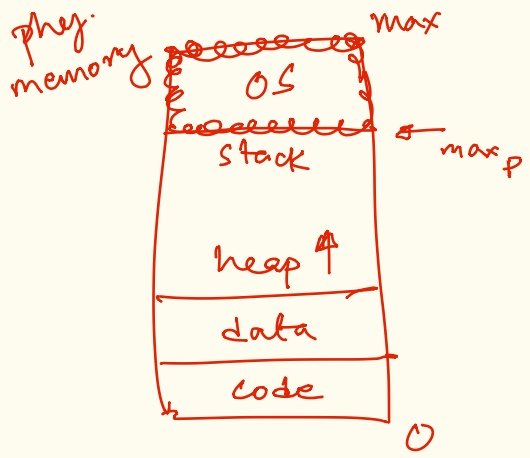
Qs) of memory mgmt.?

- which memory?
- how much?
- when to allocate?
- how to allocate?

properties / requirements?

- ~ isolation
- ~ zero-starting address
- ~ full addressability
- ~ transparent.

design 0 : similar to context save & restore idea.



extend context to
CPU state
+
phy. memory state

+ve: 1:1 process view : phy. memory

-ves:

temporal sharing

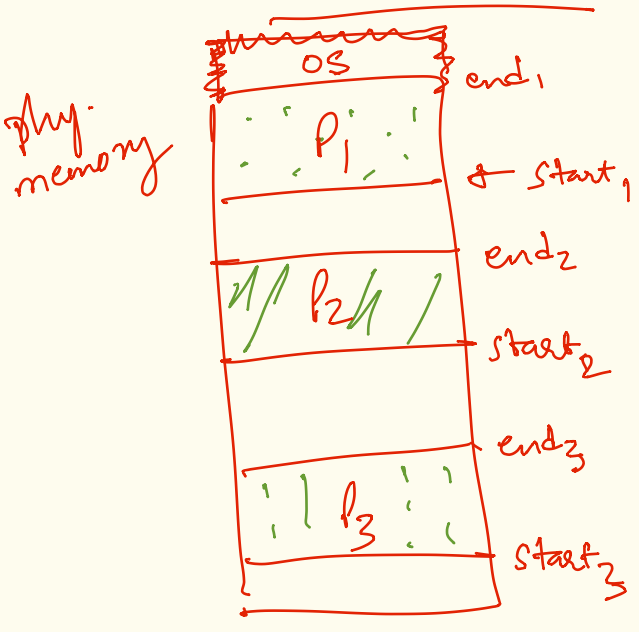
time slicing

* context save-restore volume is HUGE.
* context will need the disk for save & restore
  (much much slower than memory.)

---

* design 1: move towards spatial multiplexing sharing
* processes do not/may not need/use the full address range.

~ fragmentation → extent of unused allocated/reserved memory.



+ves

- efficiency improves
  - no memory save & restore
  - simultaneous allocation to multiple processes

-ves:

- violate full-addressability requirement
- static boundaries for allocation
  - fragmentation
  - out of memory

(*) design 2:

the address space abstraction

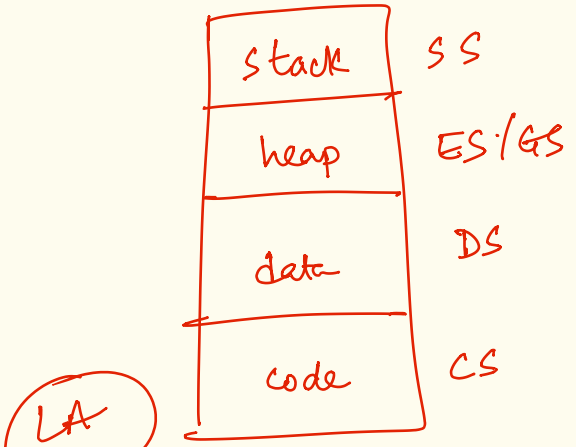
↳ decouple addressing from allocation / access

per process logical/virtual address

process view → its own address space (virtual address range)

2a segmentation

process logical addresses & divided into segments



- each process has a fixed sized collection of segments.
- process issues an address from its (logical) address space for access.