Memory Management:-

Major challenges in VMM?
* How to handle scalable memory
* holes in memory - how to manage with non contiguous memory
* Expand physical memory (+swap area)
* Protection b'n processes (Addresss space)

CPU issues logical/virtual addresses (as generated by compiler/linker)
MMU maps virtual address into physical address (as applicable in real mem)

MMU is h/w component

Paging:-

Pages -- atomic units of process image (logica/virtual)
        -- page size is fixed h/w design, e.g. in x86 page size is 4kb

Physical Frames -- equal sized partitions in physical memory
      page size == frame size

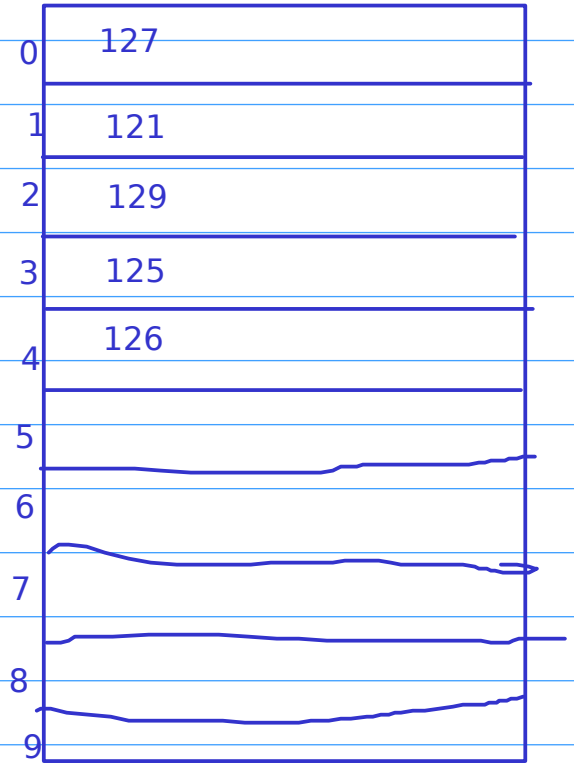
Contiguous vs Non Contiguous Memory

Pages will be contiguous, but not frames (not necessary)

Mapping of pages to frames - every process maintains a page table

40 KB -- Logical Memory

Linear addr	Page no.	Offset	Physical
1250	0	1250	127*4096+1250
4200	1	104	121*4096+104
8500	2	308	129*4096+308



Page Table Entry (PTE)

0-4095 : page0
4096-8191 : page 1
8192-12287 : page2
12288-16383 : page3

In x86(32 bit arch) ==> address width is 32 bit
Max memory : 2 pow 32 ==> 4GB

12 bits for offset within page ==> least significant
20 bits of page number ==> most significant

physical address = frame number * page size + offset
frame number --> lookup in page table

complexity of page table lookup ==> O(1),
random access, page number as index

TLB Cache:-
TLB : Translation Lookaside Buffer

Few PTEs are in TLB cache, all PTEs are in original table in memory

Cache Hit -- refer PTE in cache and compute physical address
Cache Miss -- locate PTE from page table in memory and brought into cache

Locality Principles -- cache hit ratio is better than cache miss ratio
Temporal Locality -- repeated access to same code & data e.g. loops
Spatial Locality -- most of code & data in adjacent, e.g. arrays, code

Lookup complexity in TLB? Not O(1), few entries are loaded random (non linear)

PTEs are loaded in TLB -- on demand

Physical Memory -- 1GB
Attach some disk area (swap) -- 2GB -- Backing Store

At any time, few pages are loaded in physical memory (frames)
and all pages are stored in swap area (even if in phy mem or not)

PTE have additional field ==> valid bit
==> whether is page is in physical mem or not

valid bit - 1 : page in physical memory PAGE HIT
- 0 : not in physical memory, but in swap area PAGE FAULT

Page no not found in page table , i.e index out of range
==> illegal memory access

Page Table

			valid
0	:	127	1
1	:	xxx	0
2	:	129	1
3	:	125	1
4	:	xxx	0
5	:	126	1

Page Fault Exception arises if valid bit is zero (by CPU)

Page Fault Exception Handler:-
* Locate desired page in swap area
* Identify a free frame in physical memory. If free frame not available,
select a victim frame, and write to swap area, i.e. remove from physical
(make valit bit 0 for PTE of victim page) -- PAGE OUT
* bring desired page from swap areas to identified free frame in physical mem
update frame number in PTE,make valid bit:1 PAGE IN

* Re-execute instruction caused the page fault

Page Hit -- page is loaded in physical memory
Page Fault -- not in physical memory, but in swap area

If 20%-30% pages are in main memory
Hit Ratio vs Fault Ratio?
Still HIT ratio is better, due to Locality principle

Lazy Allocation / Demand Paging:-
PAGE IN only on FAULT
Start with zero pages in physical memory, and load pages
only when page fault occurs

virtual pages

Major Faults -- disk access involved
(loading code, loading data etc from executable on disk)
Minor Faults -- no disk access, e.g. dynamic memory, stack

Bringing from swap area or allocating blank pages --> minor faults
Reading content from disk and load in VM --> major faults

Page Hit vs Page Fault
Page Fault Exception, Exception handler
Lazy allocation / Demand paging

Dirty Bit
Page Permissions
Copy on Write
Thrashing
Hierarchial Paging

Pre Fetching + locking for critical operations
locked pages are exempted from page replacement (victim page selection)
system calls in Linux:- mlock, mlockall, munlock, munlockall

Dirty bit : 0 means no changes to which is page loaded in phy memory
(clean, w.r.t. page in swap area, no changes)
no need to write into swap area
: 1 means some changes, sync changes with page in swap area

Significance of dirty during PAGE OUT (replacement)
---
page permissions -- read, write, executable

copy on write:-

fork:- duplicate pages ideally
but child may not modify many pages, child may
detach/discard from these pages in case of execl/execlp

solution:-
conditional sharing of pages b'n parent & child
Till no write operation sharing applies, duplicate page
just before write operations
Or if execl occurs, pages belongs to parent only with
suitable rwx perms (no conditional sharing required)

Thrashing:-
spending more time in CPU due to page faults, not accounted
on any processes

reasonable HIT ratio -- without thrashing -- 2%%-30% of Free Frames

---

which software used absolute addressing only? (relative addressing not
required/not applicable)

Localiyi Principle:- Spatial vs Temporal locality

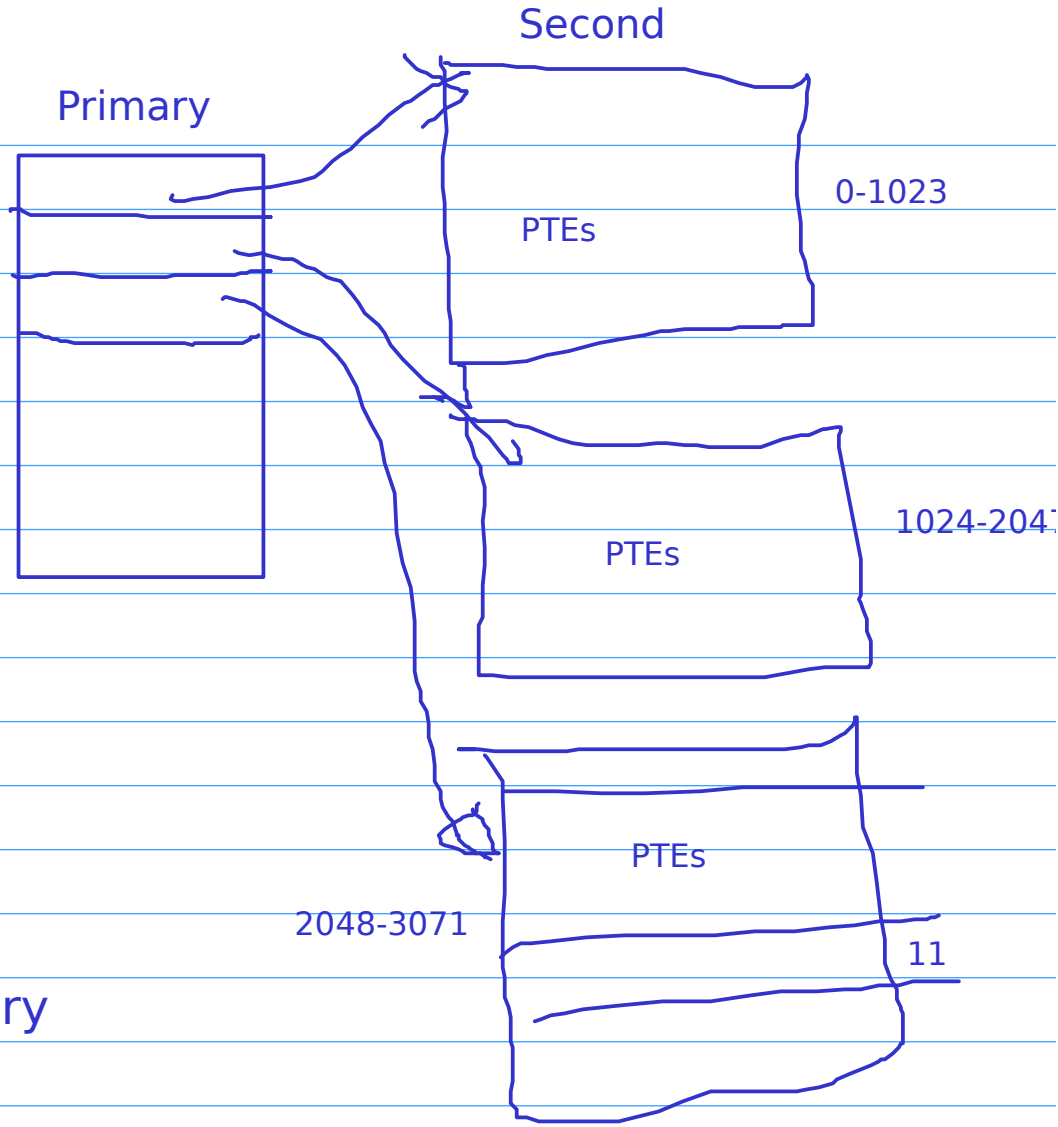
Zones in Actial Memory:-

ZONE_DMA
ZONE_NORMAL
ZONE_HIGH

single table	hierarchial
2200	2, 104

32 bit address filed:-
10 bit ==> primary index
10 bit ==> seondary index
12 bit ==> offset within page

mmap : attach blank memory
: load file contents into memory
: private vs shared mapping



mmap mappings ==>

- Anoymous private mapping (no file attached)
- Anonmou shared mapping
- Named private mapping (file linking)
- Named shared mapping (file linking)

Shared Meory -- SysV / POSIX

Preferred - POSIX