Drawbacks of a linear page table design

- Occupies much space (Page table entries (PTEs) for entire address space need to be present in the page table even if most of the address space region is unused)
- Need to be contiguous (allocating large contiguous memory)

Why not increase page size?

- This again brings back the problem of internal fragmentation

Goals

- Page table size should be proportional to used memory within address space
- Flexibility (Remove the need for contiguity)

Segmentation + Paging

- The idea of having logical segment per memory region (segment each for code, heap and stack) could be used on top of paging
- Introduce different page tables for code, heap and stack region and the start of the page table is contained in the segment registers
- Drawbacks
  - Unused space within a segment (e.g. heap) still requires page table entries
  - Every page table still need to be contiguous

Paging over paging (multi-level paging)

- Similar to the idea of applying segmentation over paging, apply the idea of paging over page tables
- Split the contiguous page table into pages and every such page can be residing in memory anywhere (they need not be contiguous)
- Two questions arise
  - If page table is broken and is placed in different pages across memory, how are they tracked?
  - Address translation in hardware (MMU) worked since the page table was contiguous but the contiguity is removed now
- New directory page is introduced that tracks the pages belonging to the page table
- New address translation
  - First index the directory page to obtain the sub page table
  - From the page table obtain the PTE
  - Finally physical address by combining PFN (from PTE) with offset

Page directory

- Size of the page directory should not exceed the page size
- If this happens then a meta-directory page indexing pages belonging to the directories has to be introduced
- Else contiguous pages for page directory need to be allocated (We try to avoid this)
Questions

- When is linear page table better than multi-level page table in terms of space occupied by the page table?
- With multi-level page table, more than one memory reference is needed to access the page table (including page directories). With N-level page table, how many memory accesses are needed? (Assume two cases of TLB hit and a TLB miss)
- We all know that 32-bit processors could address up to $2^{32}$ bytes. Does that mean those processors could use only 4 GB of RAM? (Read about PAE mode).
- Super pages are pages with larger sizes (Processor design support more than one page size - 4K, 1M) - How does it make OS design harder? (Issue of contiguity)
- How does super pages have an impact on TLB? (Think in terms of hit rate and miss rate)

How large must PTE’s (page table entry) be?

(a) Physical address space contains 1024 pages. 7 bits needed for extras (protection, valid etc.)
(b) Physical address are 16-bits, pages are 32 bytes, 8 bits needed for extras
(c) Physical address space is 4 GB, pages are 4 KB, 6 bits needed for extras

How large is the virtual address space, assuming 4-KB pages and 4-byte entries with N levels?
(Page Table can now no longer require more than 1 contiguous page for bookkeeping)
(a) N = 1
(b) N = 2
(c) N = 3