Segmentation was variable-sized pieces, causing fragmentation and challenging allocation.

What it looks like:

Converting from Virtual to Physical Address

Step 1: Determine Virtual Page number and offset
First, do it by the diagram
   - If we had virtual address 21, where is it?
   - If we had virtual address 33, where is it?

How do we do this mathematically?

Step 2: Find the physical address
First:
Second:

```
\text{r}_{\text{phys}} = \text{pt}[\text{vpg}] \\
glue\text{them\ together}
```
What’s in a Page Table?

For now we assume a simple data structure:

For each Page Table Entry need:

- Valid Bit:
- 3 Protection Bits:
- Present Bit: Whether or not page is in physical memory (we’ll get to later)
- Reference Bit: Track if a page has been accessed (we’ll need later)
- Dirty Bit:

Issue 1:

Issue 2: PT can be Big

Tracing Memory

Let’s say we have some very simple code:

```c
int array[1000];
for(int i=0; i<1000; i++)
    array[i] = 0;
```

Use “gcc –S” (or a tool to disassemble executable) to see assembly instructions to initialize array:

```asm
0x1024  movl $0x0, (%edi,%eax,4)
0x1028  incl %eax
0x102c  $0x03e8, %eax
0x1030  jne 0x1024
```
Memory trace if the following are true:

- Virtual address space is 64KB
- Page size is 1KB
- Linear array-based page table exists at physical address 1KB (1024)
- Virtual page of code is on VPN 1, on physical frame 4
- Array exists at virtual address 40000 up until 44000, VPNs 39-42
- Mappings: VPN 39 -> 7, VPN 40 -> 8, VPN 41 -> 9, VPN 42 -> 10