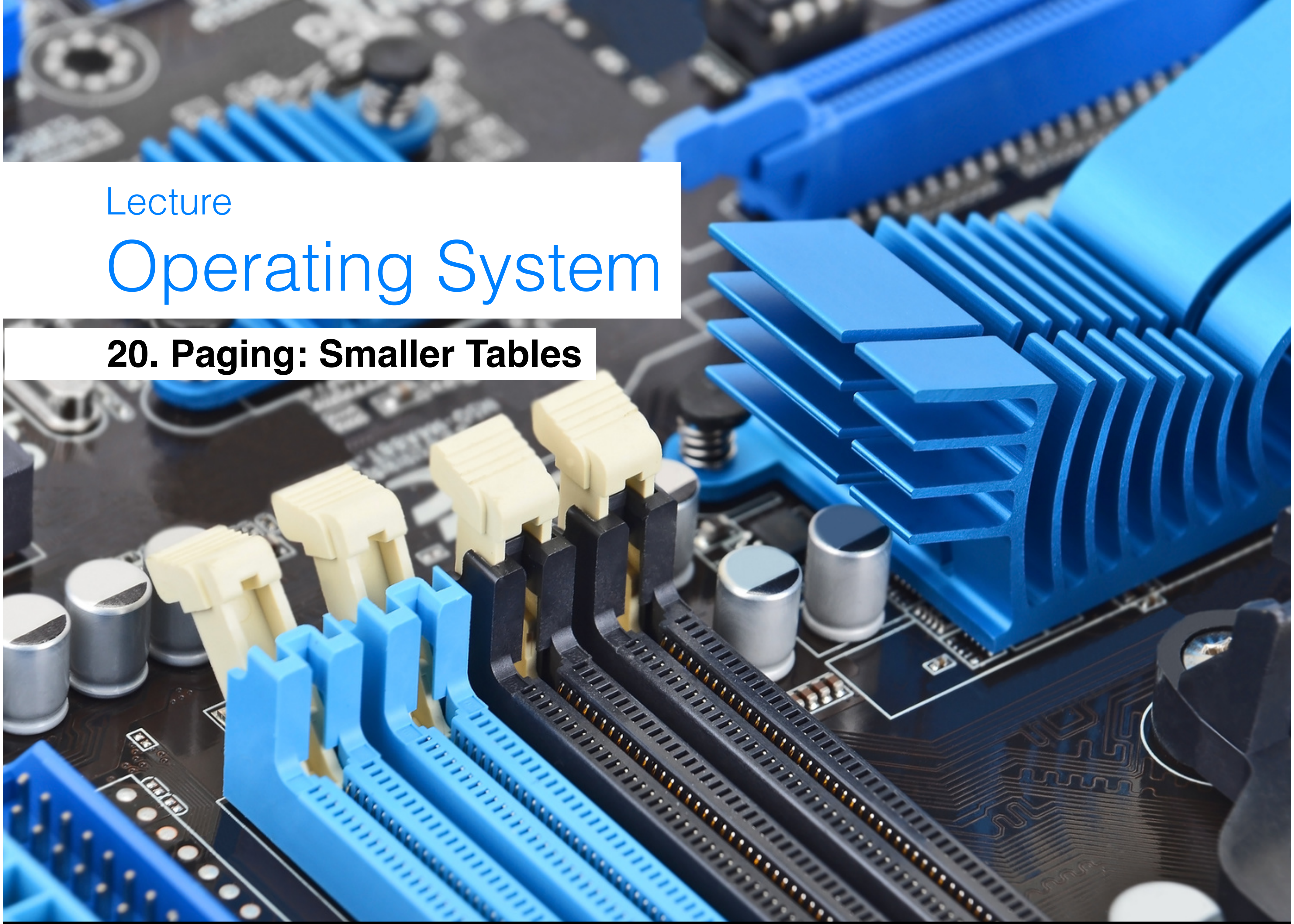


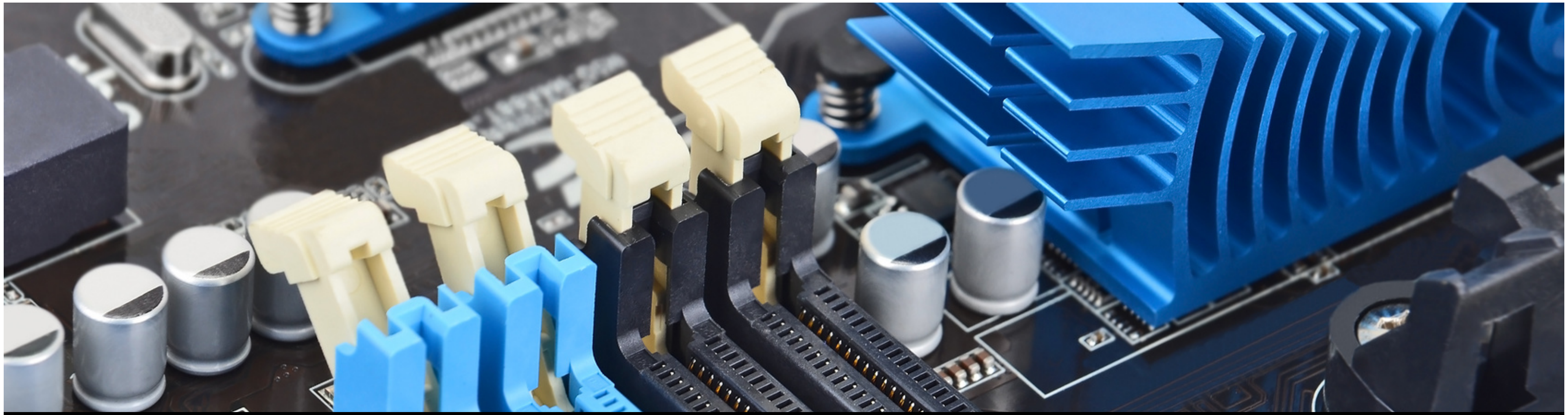
Lecture

# Operating System

## 20. Paging: Smaller Tables

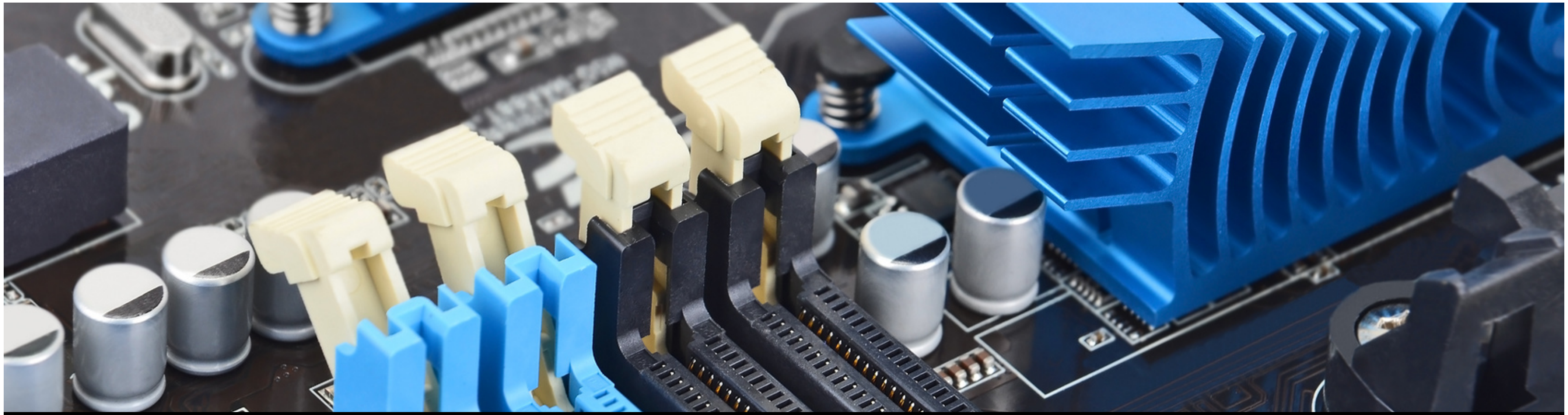
Professor Dr. Michael Mächtel

# 20. Paging: Smaller Tables



# 20. Paging: Smaller Tables

1. **Problem with a Linear Page Table**
2. **Hybrid Approach: Segmented Pagetables**
3. **Multi-level Pagetables**
4. **Inverted Page Tables**



## Questions to solve

- Review: What are problems with paging?
- Review: How large can page tables be?
- How can large page tables be avoided with different techniques?
- What happens on a TLB miss?

## Disadvantages of Paging

- Additional memory reference to look up in page table
  - Very inefficient
  - Page table must be stored in memory
  - MMU stores only base address of page table
  - Avoid extra memory reference for lookup with TLBs (previous slides)
- Storage for page tables may be substantial
  - Simple page table: Requires PTE for all pages in address space
    - Entry needed even if page not allocated
- Problematic with dynamic stack and heap within address space (today)

## QUIZ: How big are page Tables?

- PTE's are 2 bytes, and 32 possible virtual page numbers

  32 * 2 bytes = 64 bytes

- PTE's are 2 bytes, virtual addrs are 24 bits, pages are 16 bytes

  2 bytes * 2^(24 – lg 16) = **2^21 bytes** (2 MB)

- PTE's are 4 bytes, virtual addrs are 32 bits, and pages are 4 KB

  4 bytes * 2^(32 – lg 4K) = **2^22 bytes** (4 MB)

- PTE's are 4 bytes, virtual addrs are 64 bits, and pages are 4 KB

  4 bytes * 2^(64 – lg 4K) = **2^54** bytes (18 PetaBytes)

## Paging: Linear Tables

- We usually have one page table for every process in the system.
  - Assume that 32-bit address space with 4KB pages and 4-byte page-table entry.

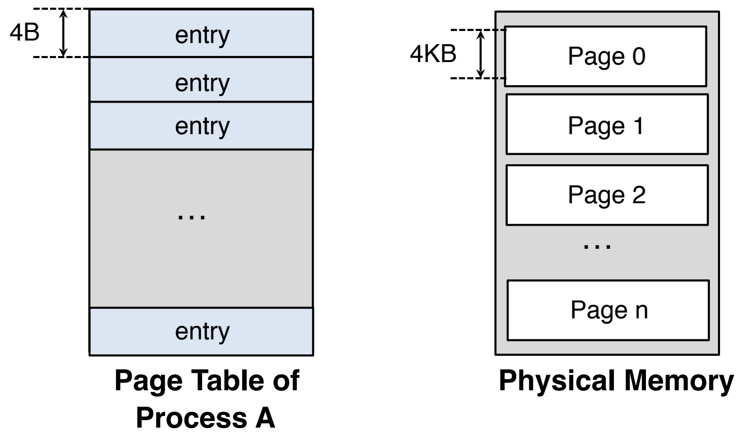


table size: $\frac{2^{32}}{2^{12}} * 4\ Byte = 4\ MByte$

**Page table are too big and thus consume too much memory.**

# Paging: Smaller Tables

- Page table are too big and thus consume too much memory.
  - Assume that 32-bit address space with 16KB pages and 4-byte page-table entry.

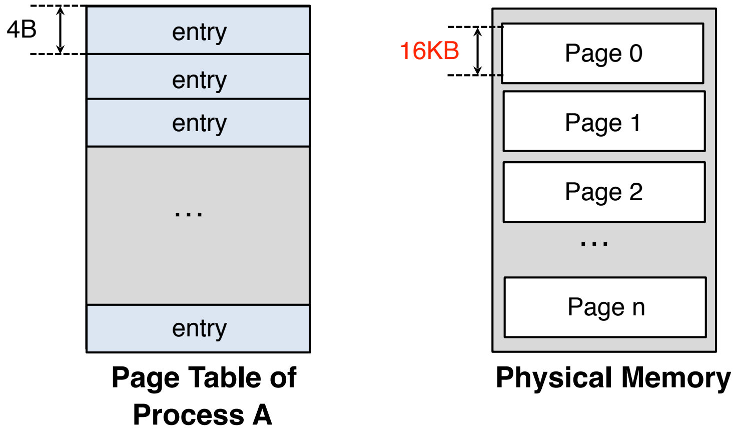
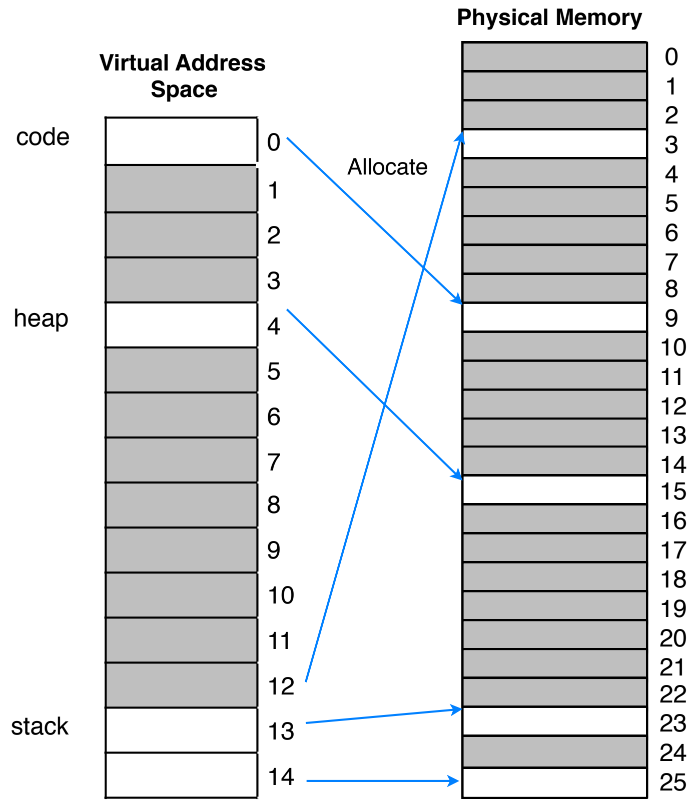


table size: $\frac{2^{32}}{2^{14}} * 4\, Byte = 1\, MByte$

**Big pages lead to internal fragmentation.**

## Problem

- Single page table for the entries address space of process.



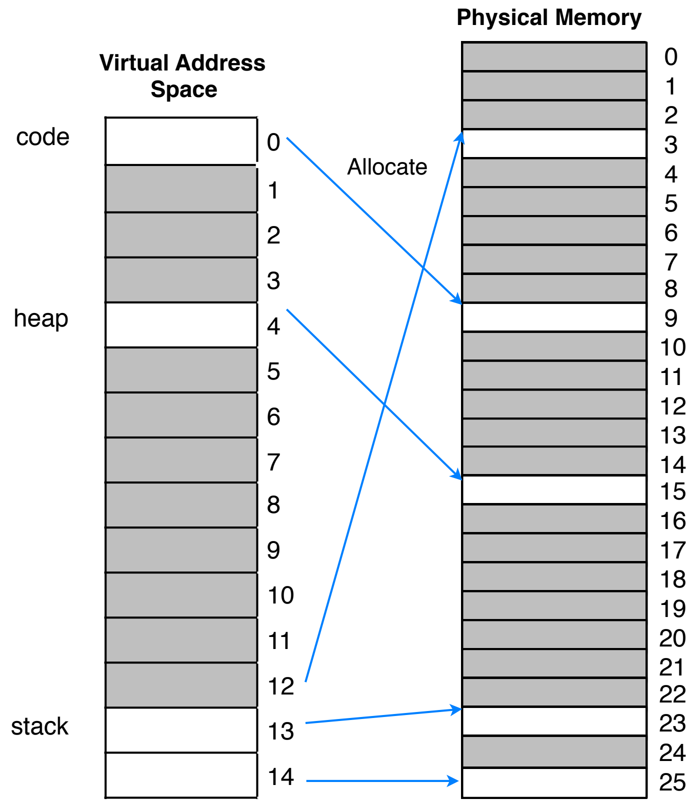
**A 16KB Address Space with 1KB Pages**

| PFN | valid | prot | present | dirty |
|---|---|---|---|---|
| 10 | 1 | r-x | 1 | 0 |
| - | 0 | - | - | - |
| - | 0 | - | - | - |
| - | 0 | - | - | - |
| 15 | 1 | rw- | 1 | 1 |
| … | … | … | … | … |
| - | 0 | - | - | - |
| 3 | 1 | rw- | 1 | 1 |
| 23 | 1 | rw- | 1 | 1 |

**A Page Table For 16KB Address Space**

## Problem

- Most of the page table is unused, full of invalid entries.



A 16KB Address Space with 1KB Pages

| PFN | valid | prot | present | dirty |
|---|---|---|---|---|
| 10 | 1 | r-x | 1 | 0 |
| - | 0 | - | - | - |
| - | 0 | - | - | - |
| - | 0 | - | - | - |
| 15 | 1 | rw- | 1 | 1 |
| … | … | … | … | … |
| - | 0 | - | - | - |
| 3 | 1 | rw- | 1 | 1 |
| 23 | 1 | rw- | 1 | 1 |

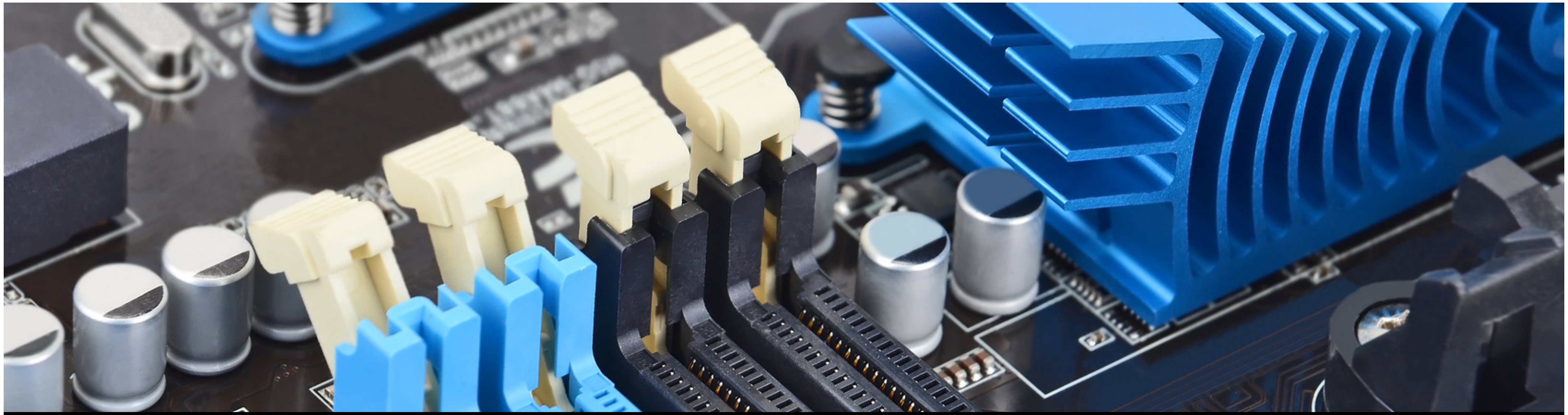
A Page Table For 16KB Address Space

## Avoid simple linear Page Table

- Use more complex page tables, instead of just big array
  - Any data structure is possible with software-managed TLB
  - Hardware looks for vpn in TLB on every memory access
  - If TLB does not contain vpn, TLB miss
    - Trap into OS and let OS find vpn->ppn translation
    - OS notifies TLB of vpn->ppn for future accesses

# 20. Paging: Smaller Tables

1. Problem with a Linear Page Table
2. **Hybrid Approach: Segmented Pagetables**
3. Multi-level Pagetables
4. Inverted Page Tables

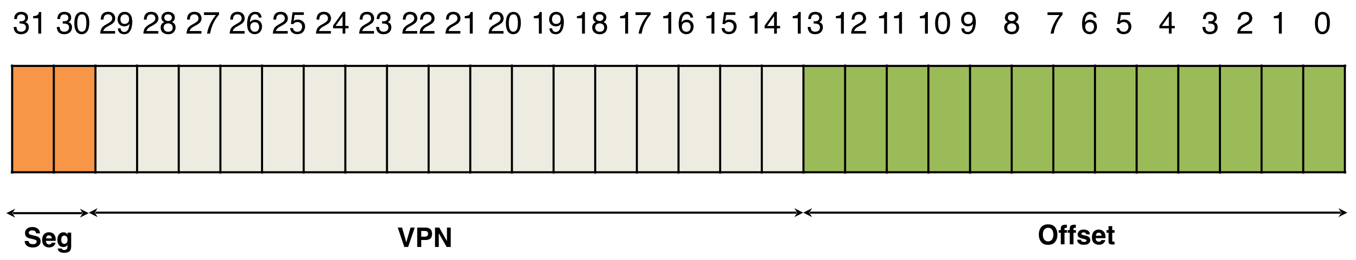


## Hybrid Approach: Paging and Segments

- In order to reduce the memory overhead of page tables:
  - Divide address space into segments (code, heap, stack)
  - Segments can be variable length
  - Divide each segment into fixed-sized pages
- Implementation
  - Each segment has a page table
    - Using base not to point to the segment itself but rather to hold the physical address of the page table of that segment.
    - The bounds register is used to indicate the end of the page table.

## Simple Example of Hybrid Approach

- Each process has three page tables associated with it.
    - When process is running, the base register for each of these segments contains the physical address of a linear page table for that segment.



**32-bit Virtual address space with 4KB pages**

| Seg value | Content |
|---|---|
| 00 | unused segment |
| 01 | code |
| 10 | heap |
| 11 | stack |

## TLB miss on Hybrid Approach

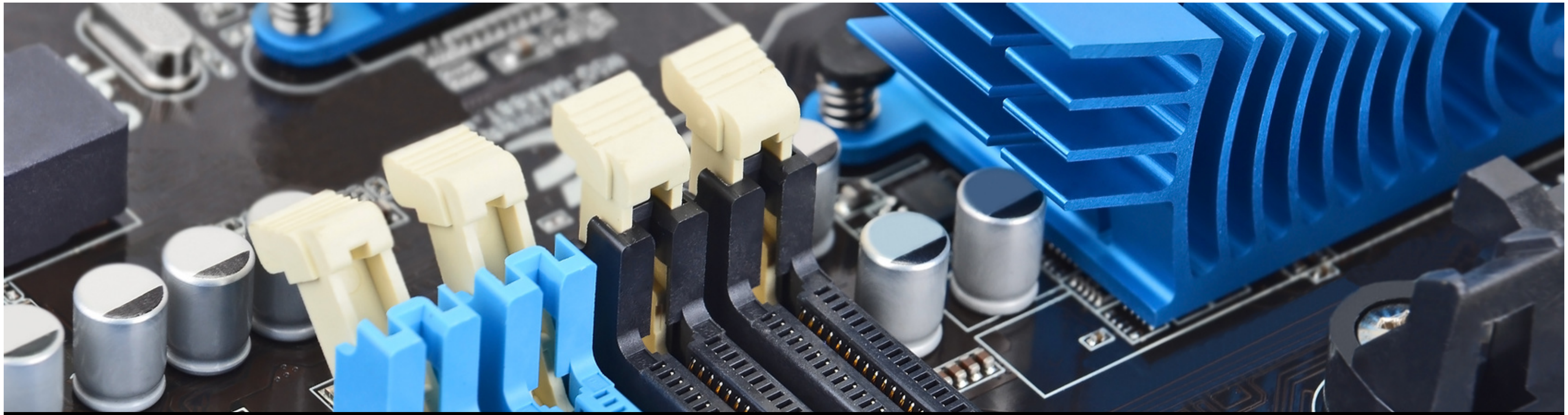
- The hardware get to **physical address** from **page table**.
  - The hardware uses the segment bits(SN) to determine which base and bounds pair to use.
  - The hardware then takes the physical address therein and combines it with the VPN as follows to form the address of the page table entry(PTE) .

```
SN = (VirtualAddress & SEG_MASK) >> SN_SHIFT
VPN = (VirtualAddress & VPN_MASK) >> VPN_SHIFT
AddressOfPTE = Base[SN] + (VPN * sizeof(PTE))
```

## Problem of Hybrid Approach

- Hybrid Approach is not without problems.
  - If we have a large but sparsely-used heap, we can still end up with a lot of page table waste.
  - Causing external fragmentation to arise again.
  - Must allocate each page table contiguously
  - Each page table is:
    - Assume 2 bits for segment, 18 bits for page number, 12 bits for offset
    - Number of entries * size of each entry
    - Number of pages * 4 bytes
    - 2^18 * 4 bytes = 2^20 bytes = 1 MB!!!

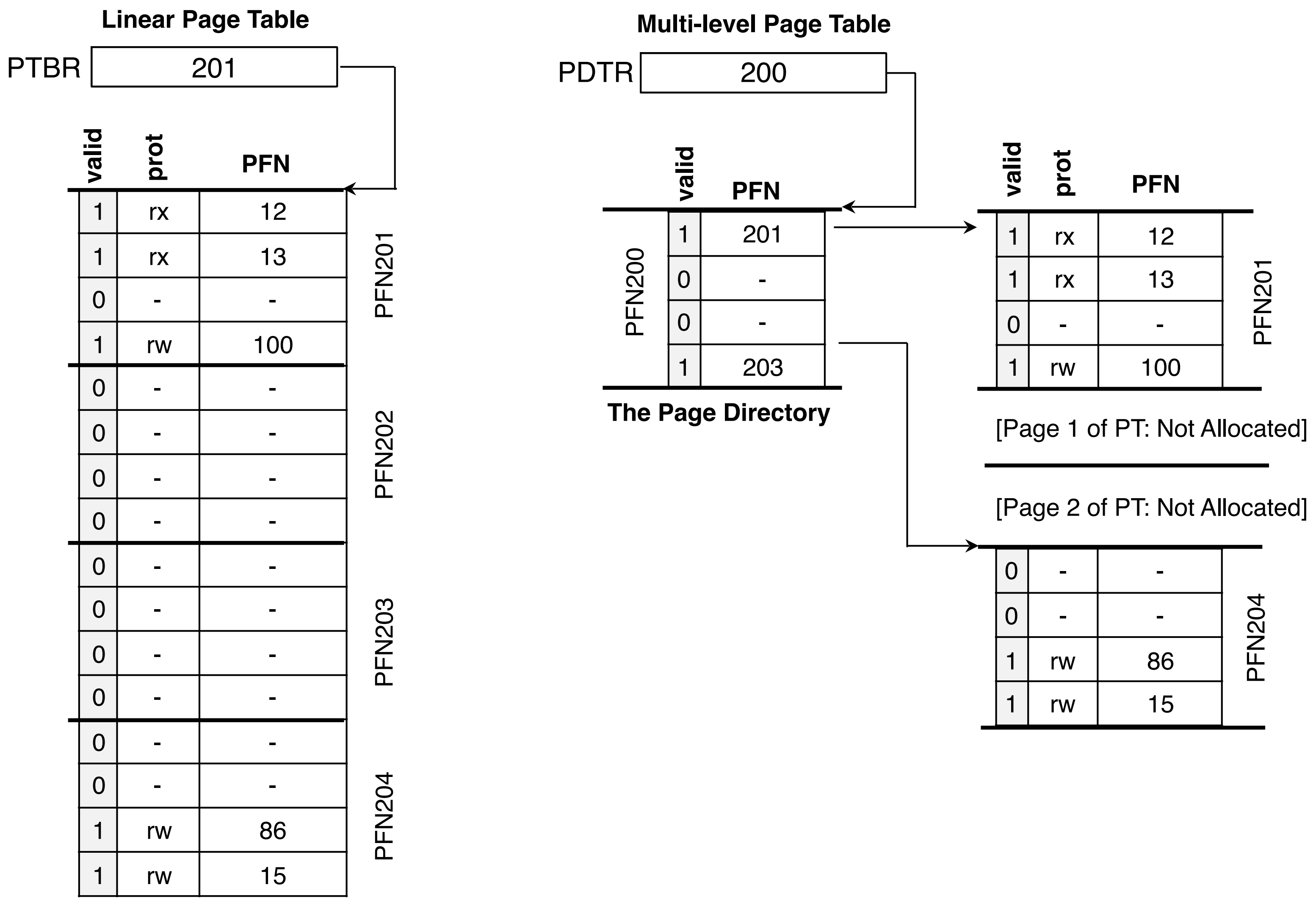
# 20. Paging: Smaller Tables



## Multi-level Page Tables

- Goal: Allow each page tables to be allocated non-contiguously
- Idea: Page the page tables
- => Turn the linear page table into something like a tree.
  - Chop up the page table into page-sized units.
  - If an entire page of page-table entries is invalid, don't allocate that page of the page table at all.
  - To track whether a page of the page table is valid, use a new structure, called page directory.

# Multi-level Page Tables: Page directory



**Linear Page Table** (PTBR 201)

| valid | prot | PFN | |
|---|---|---|---|
| 1 | rx | 12 | PFN201 |
| 1 | rx | 13 | PFN201 |
| 0 | - | - | PFN201 |
| 1 | rw | 100 | PFN201 |
| 0 | - | - | PFN202 |
| 0 | - | - | PFN202 |
| 0 | - | - | PFN202 |
| 0 | - | - | PFN202 |
| 0 | - | - | PFN203 |
| 0 | - | - | PFN203 |
| 0 | - | - | PFN203 |
| 0 | - | - | PFN203 |
| 0 | - | - | PFN204 |
| 0 | - | - | PFN204 |
| 1 | rw | 86 | PFN204 |
| 1 | rw | 15 | PFN204 |

**Multi-level Page Table** (PDTR 200)

The Page Directory (PFN200)

| valid | PFN |
|---|---|
| 1 | 201 |
| 0 | - |
| 0 | - |
| 1 | 203 |

PFN201

| valid | prot | PFN |
|---|---|---|
| 1 | rx | 12 |
| 1 | rx | 13 |
| 0 | - | - |
| 1 | rw | 100 |

[Page 1 of PT: Not Allocated]

[Page 2 of PT: Not Allocated]

PFN204

| valid | prot | PFN |
|---|---|---|
| 0 | - | - |
| 0 | - | - |
| 1 | rw | 86 |
| 1 | rw | 15 |

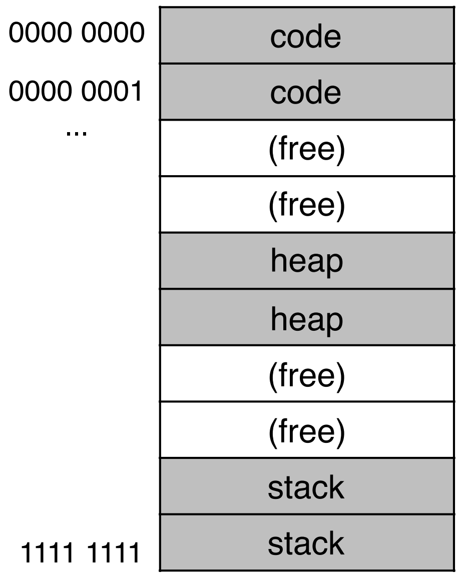
## Page directory entries

- The page directory contains one entry per page of the page table.
  - It consists of a number of page directory entries(PDE).
- PDE has a **valid bit** and page frame number(**PFN**).
- Multi-level Page Table: Level of indirection
  - A multi-level structure can adjust level of indirection through use of the page directory.
  - Indirection place page-table pages wherever we would like in physical memory.

## Advantage & Disadvantage

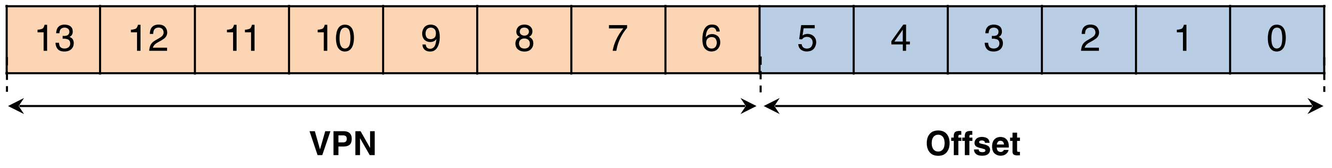
- Advantage
  - Only allocates page-table space in proportion to the amount of address space you are using.
  - The OS can grab the next free page when it needs to allocate or grow a page table.
- Disadvantage
  - Multi-level table is a small example of a time-space trade-off.
  - Complexity.

## Multi-Level Example



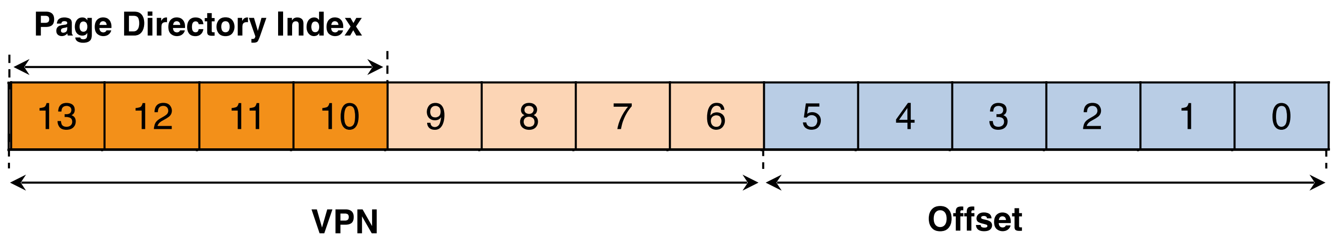
| Flag | Detail |
|---|---|
| Address space | 16 KB |
| Page size | 64 byte |
| Virtual address | 14 bit |
| VPN | 8 bit |
| Offset | 6 bit |
| Page table entry | $2^8$ (256) |

**A 16-KB Address Space With 64-byte Pages**



## Multi-Level Example

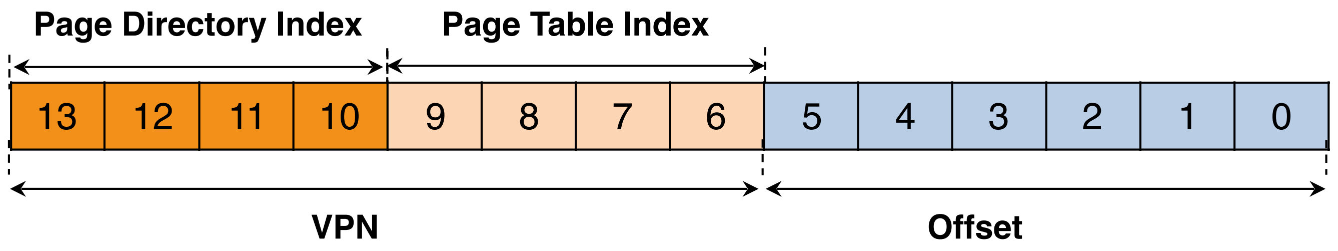
- The page directory needs one entry per page of the page table
  - it has 16 entries.
- The page-directory entry is invalid
  - Raise an exception (The access is invalid)



**14-bits Virtual address**

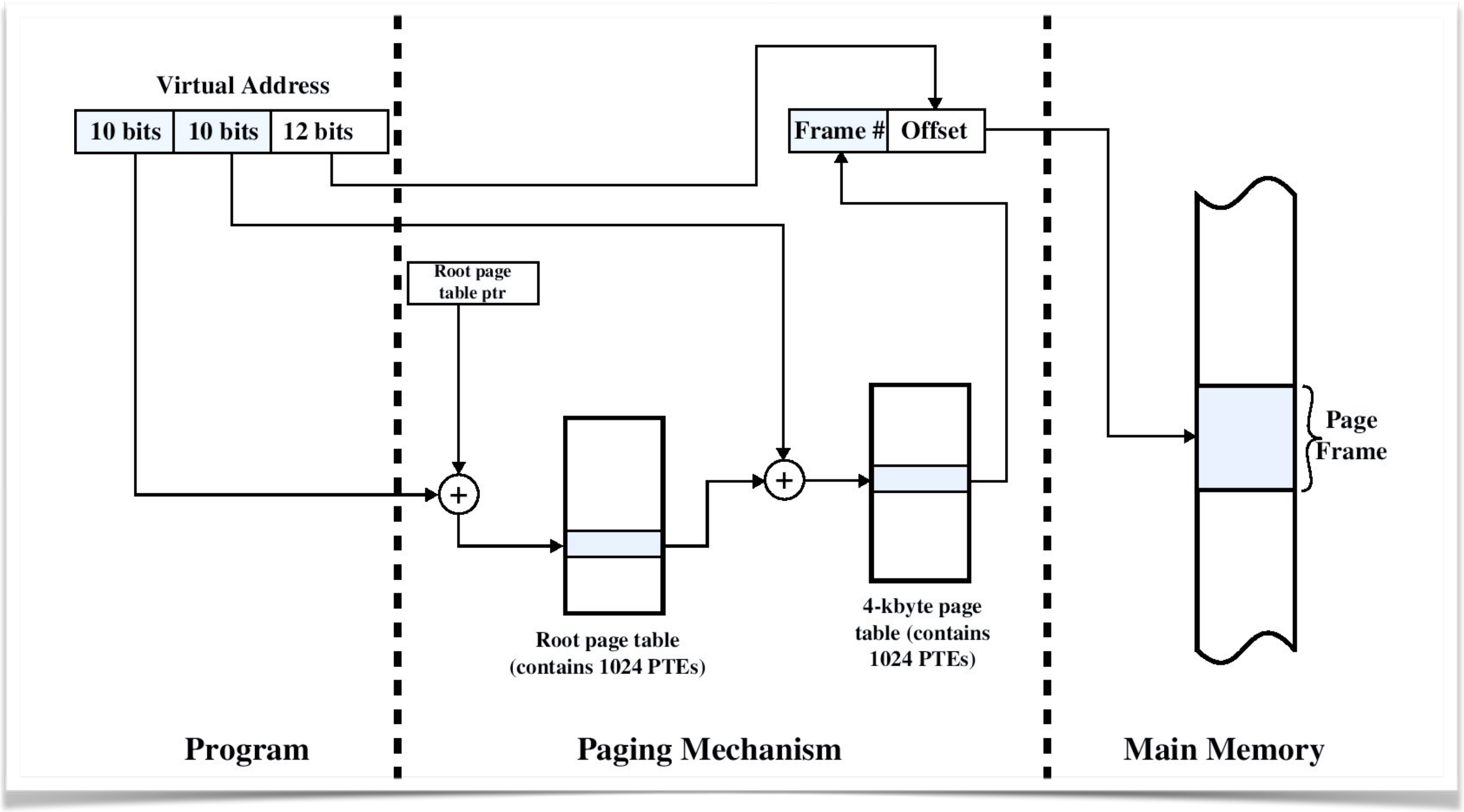
## Multi-Level Example

- The PDE is valid, we have more work to do.
  - To fetch the page table entry(PTE) from the page of the page table pointed to by this page-directory entry.
- This page-table index can then be used to index into the page table itself.



14-bits Virtual address

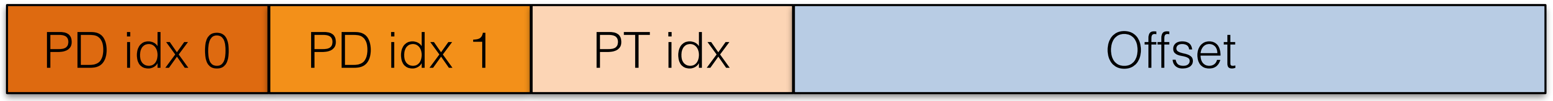
# 2-level translation



Quelle: Stallings

## Problem with 2 levels?

- Problem: page directories may not fit in a page
- Solution:
  - Split page directories into pieces
  - Use another page dir to refer to the page dir pieces



  - Exercise: How large is virtual address space with
    - 4 KB pages, 4 byte PTEs, each page table fits in page
    - given 1,2,3 levels

# Multi-level Page Table Control Flow

```
VPN = (VirtualAddress & VPN_MASK) >> SHIFT
(Success,TlbEntry) = TLB_Lookup(VPN)
if(Success == True)    //TLB Hit
  if(CanAccess(TlbEntry.ProtectBits) == True)
       Offset = VirtualAddress & OFFSET_MASK
       PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
       Register = AccessMemory(PhysAddr)
  else RaiseException(PROTECTION_FAULT);
else // PLD Miss
      // perform the full multi-level lookup
     PDIndex = (VPN & PD_MASK) >> PD_SHIFT
     PDEAddr = PDBR + (PDIndex * sizeof(PDE))
     PDE = AccessMemory(PDEAddr)
     if(PDE.Valid == False)
       RaiseException(SEGMENTATION_FAULT)
     else // PDE is Valid: now fetch PTE from PT
```

extract the virtual page number(VPN)

check if the TLB holds the transalation for this VPN

extract the page frame number from the relevant TLB entry, and form the desired physical address and access memory

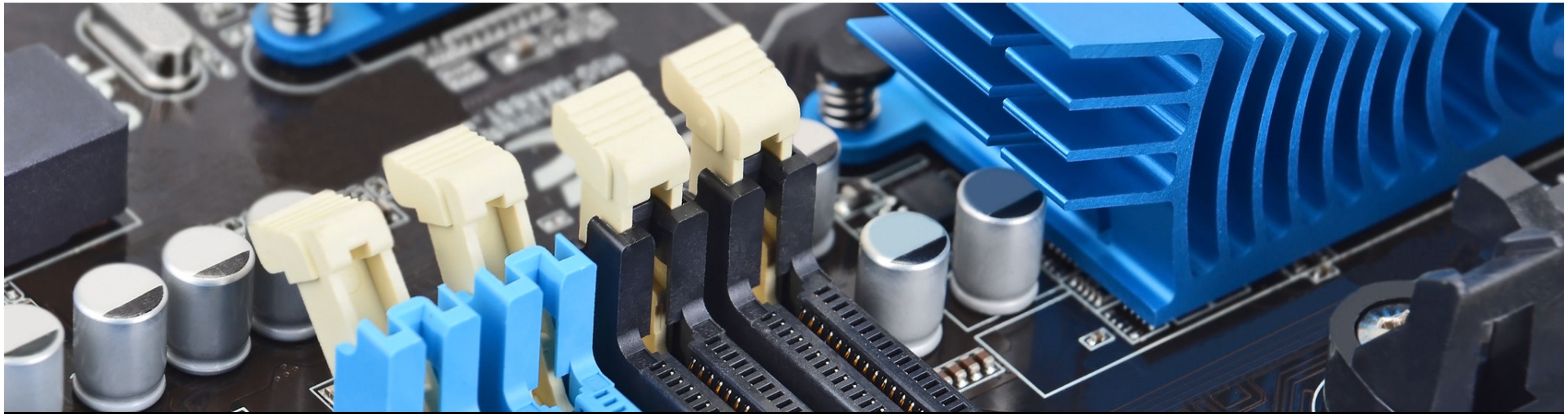
extract the Page Directory Index (PDIndex)

get Page Directory Entry(PDE)

Check PDE valid flag. If valid flag is true, fetch Page Table entry from Page Table

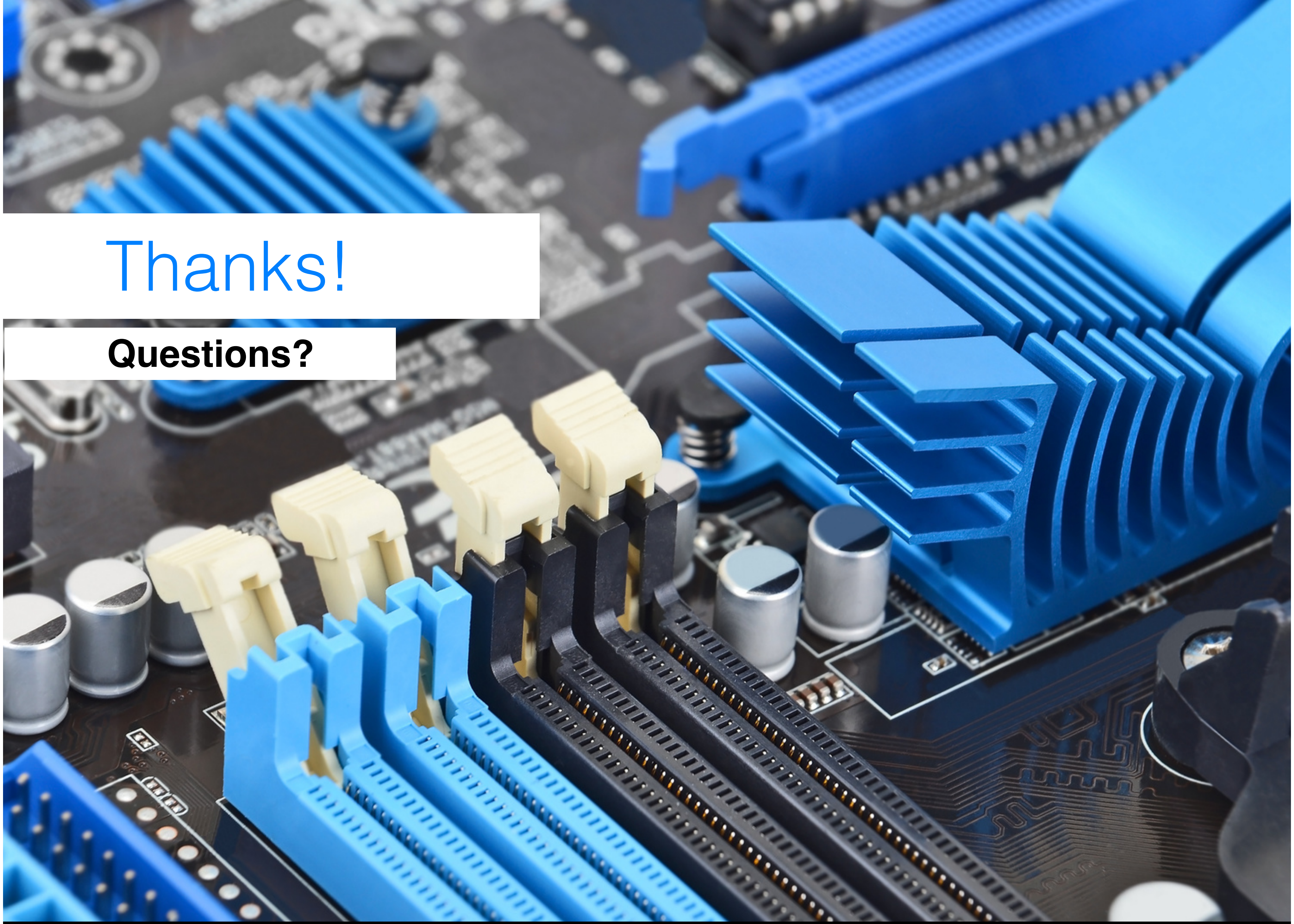
# 20. Paging: Smaller Tables

1. Problem with a Linear Page Table
2. Hybrid Approach: Segmented Pagetables
3. Multi-level Pagetables
4. **Inverted Page Tables**



## Inverted Page Tables

- More extreme space saving
- Keeping a single page table that has an entry for each physical page of the system.
- The entry tells us which process is using this page, and which virtual page of that process maps to this physical page.
- Finding the correct entry is searching through this data structure
  - Linear scan expensive
  - Hash table is build
- https://www.youtube.com/watch?v=Hgo89fGH1X0



# Thanks!

## Questions?