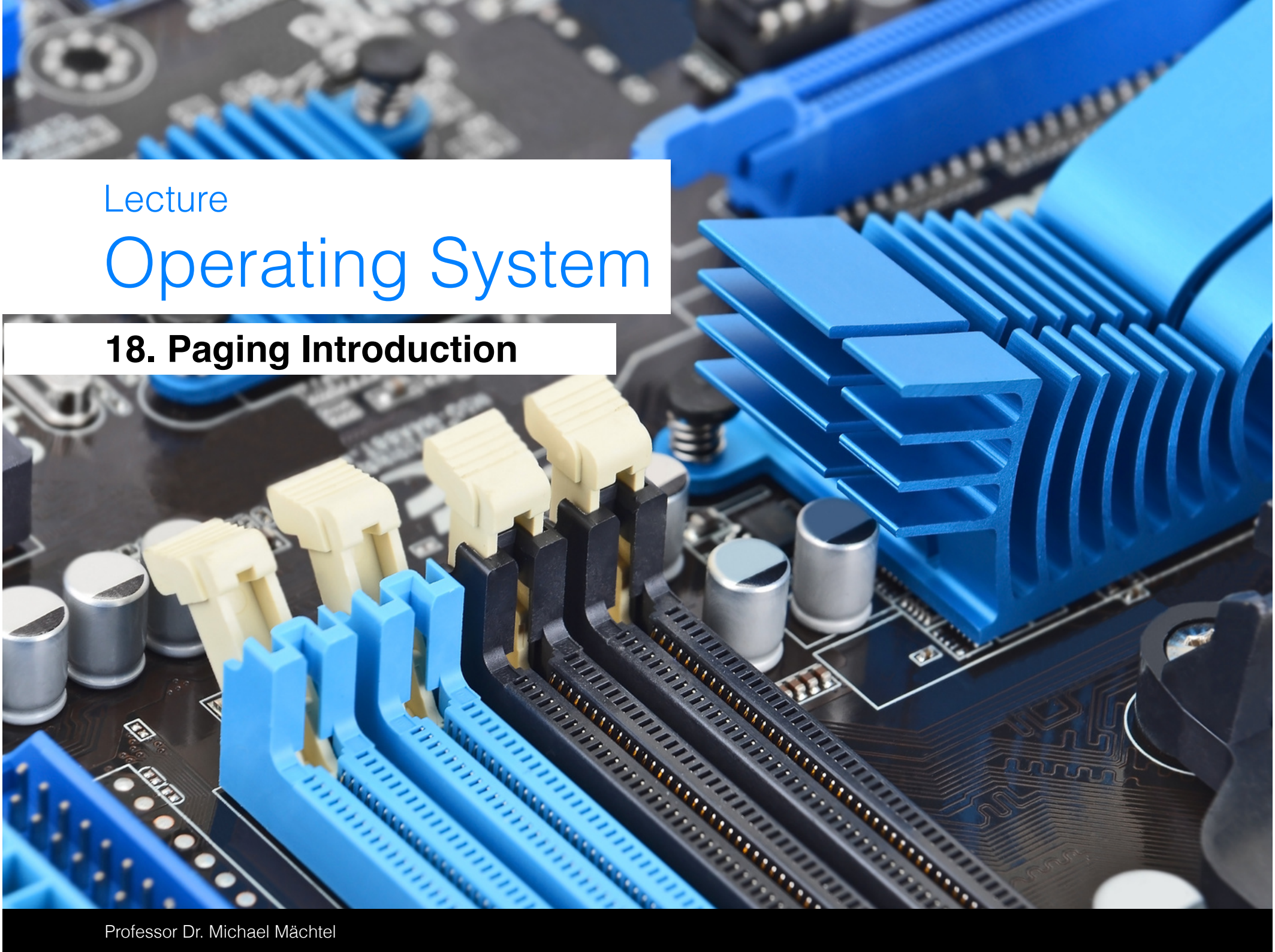


Lecture

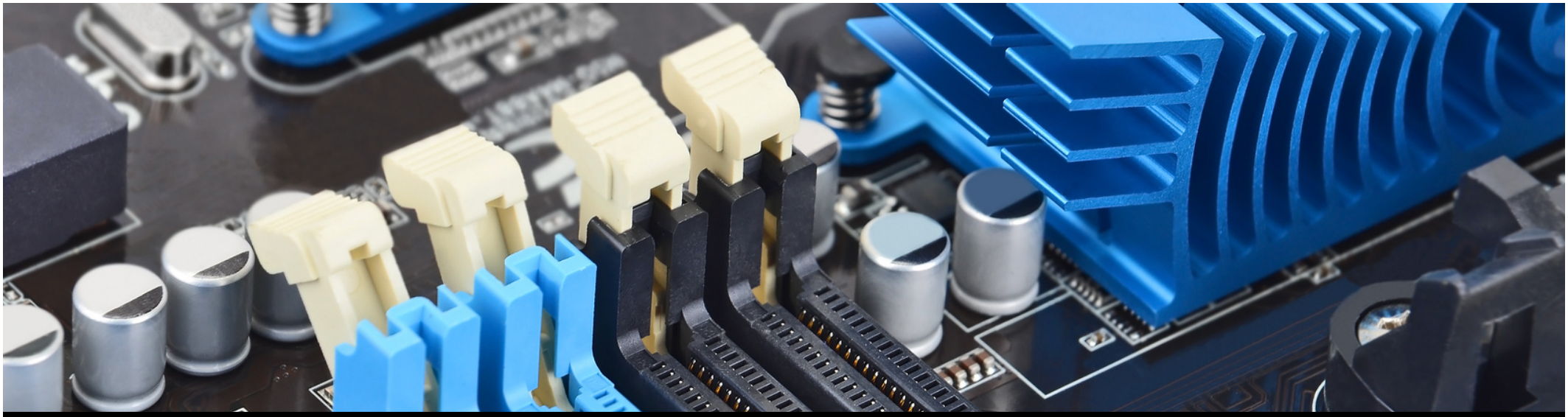
Operating System

18. Paging Introduction

Professor Dr. Michael Mächtel

18. Paging Introduction

1. **How can we virtualize memory with pages, so as to avoid the problems of segmentation?**
2. **What are the basic techniques?**
3. **How do we make those techniques work well, with minimal space and time overheads?**



Concept of Paging

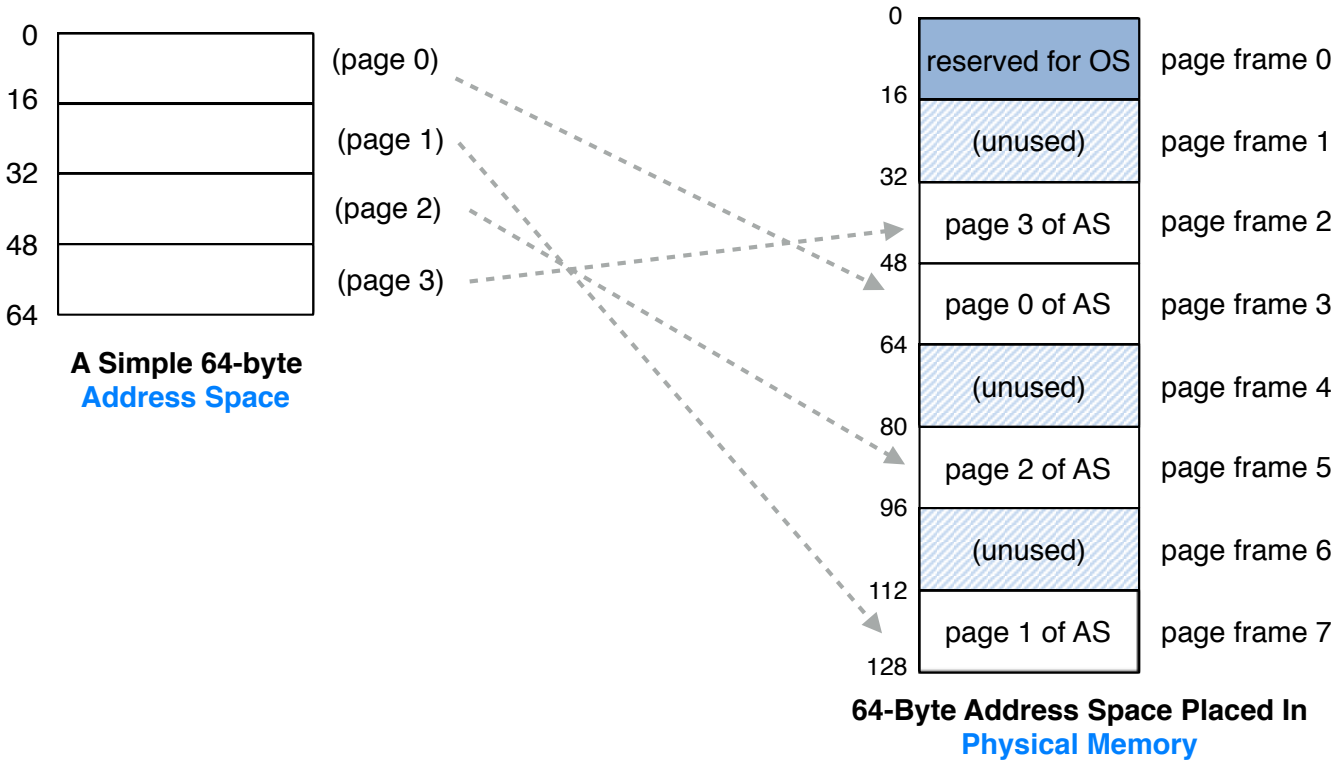
- Paging **splits up** address space into **fixed-sized** unit called a **page**.
  - Segmentation: variable size of logical segments (code, stack, heap, etc.)
- With paging, **physical memory** is also **split** into some number of pages called a **page frame**.
- **Page table** per process is needed to **translate** the virtual address to physical address.

Advantages of Paging

- **Flexibility**: Supporting the abstraction of address space effectively
  - Don’t need assumption how heap and stack grow are used.
- **Simplicity**: ease of free-space management
  - The page in address space and the page frame are the same size.
  - Easy to allocate and keep a free list

Example: A Simple Paging

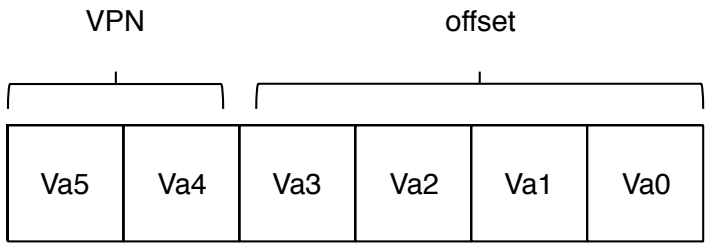
- 128-byte physical memory with 16 bytes page frames
- 64-byte address space with 16 bytes pages

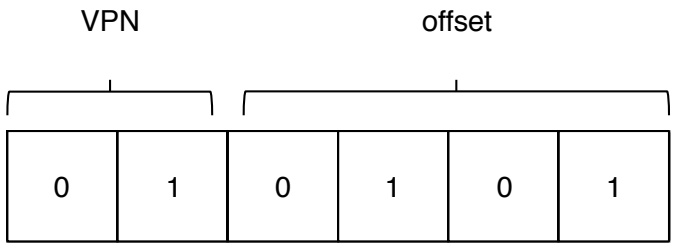


Address Translation

- Two components in the virtual address:
  - VPN: virtual page number
  - Offset: offset within the page
- Example: 64-byte address space
  - 21 in 64-byte address space

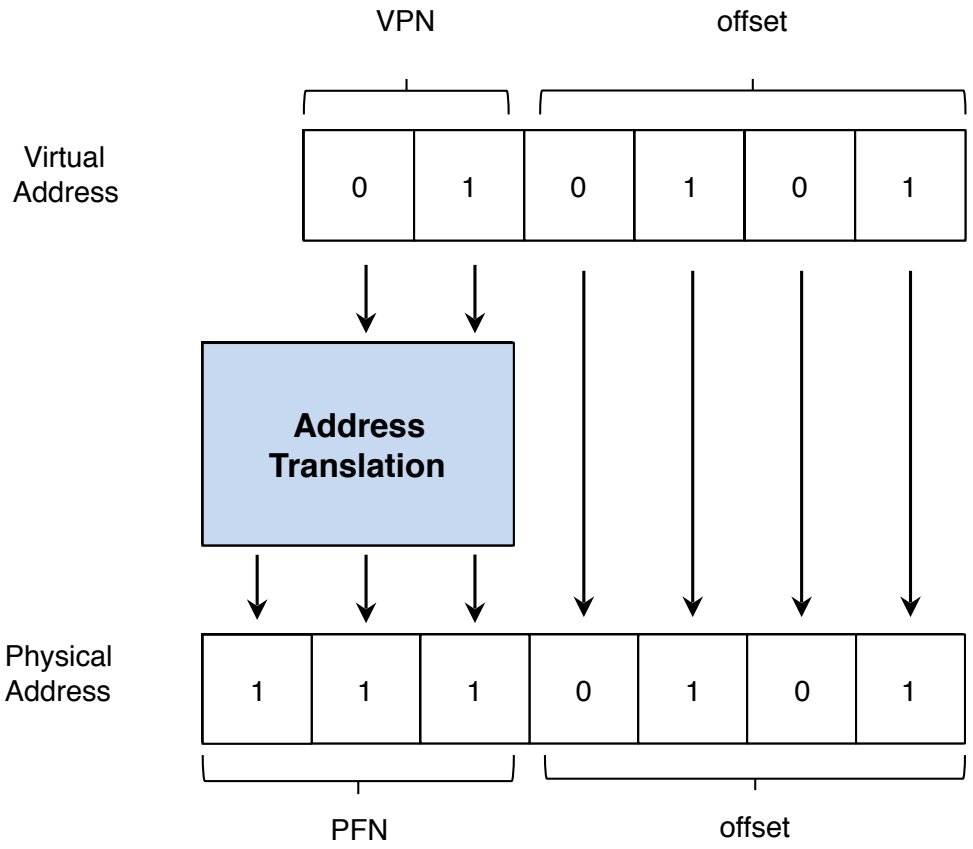
    `-010101`





Example: virtual address 21 in 64-byte address space

Address Translation



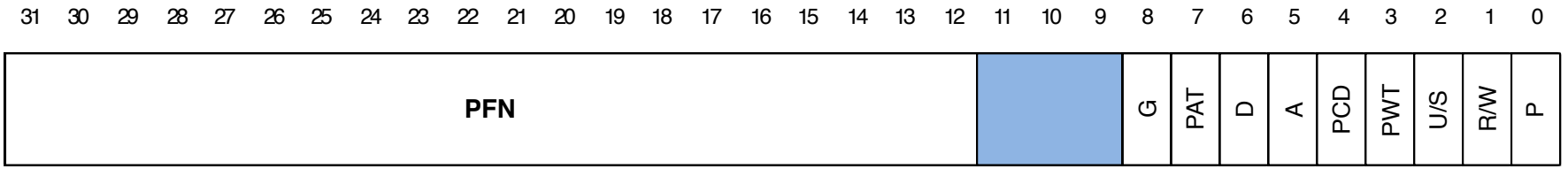
Page Tables

- The page table is just a data structure that is used to map the virtual address to physical address.
  - Simplest form: a linear page table, an array
- The OS indexes the array by VPN, and looks up the page-table entry.
- Page tables can get awfully **large**
  - 32-bit address space with 4-KB pages, 20 bits for VPN
    - *4MB = 2^{20} entries * 4 Bytes per page table entry*
- Page tables for **each** process are stored in memory.

Common Flags of Page Table Entry

- **Valid Bit**: Indicating whether the particular translation is valid.
- **Protection Bit**: Indicating whether the page could be read from, written to, or executed from
- **Present Bit**: Indicating whether this page is in physical memory or on disk(swapped out)
- **Dirty Bit**: Indicating whether the page has been modified since it was brought into memory
- **Reference Bit (Accessed Bit)**: Indicating that a page has been accessed

Page Table Entry



An x86 Page Table Entry(PTE)

P:	present
R/W:	read/write bit
U/S:	supervisor
A:	accessed bit
D:	dirty bit
PFN:	the page frame number

Paging: Too Slow

- To find a location of the desired PTE, the **starting location** of the page table is needed.
- For every memory reference, paging requires the OS to perform one **extra memory reference**.

```
1       // Extract the VPN from the virtual address
2       VPN = (VirtualAddress & VPN_MASK) >> SHIFT
3
4       // Form the address of the page-table entry (PTE)
5       PTEAddr = PTBR + (VPN * sizeof(PTE))
6
7       // Fetch the PTE
8       PTE = AccessMemory(PTEAddr)
9
10      // Check if process can access the page
11      if (PTE.Valid == False)
12              RaiseException(SEGMENTATION_FAULT)
13      else if (CanAccess(PTE.ProtectBits) == False)
14              RaiseException(PROTECTION_FAULT)
15      else
16              // Access is OK: form physical address and fetch it
17              offset = VirtualAddress & OFFSET_MASK
18              PhysAddr = (PTE.PFN << PFN_SHIFT) | offset
19              Register = AccessMemory(PhysAddr)
```

A Memory Trace

Example: A Simple Memory Access

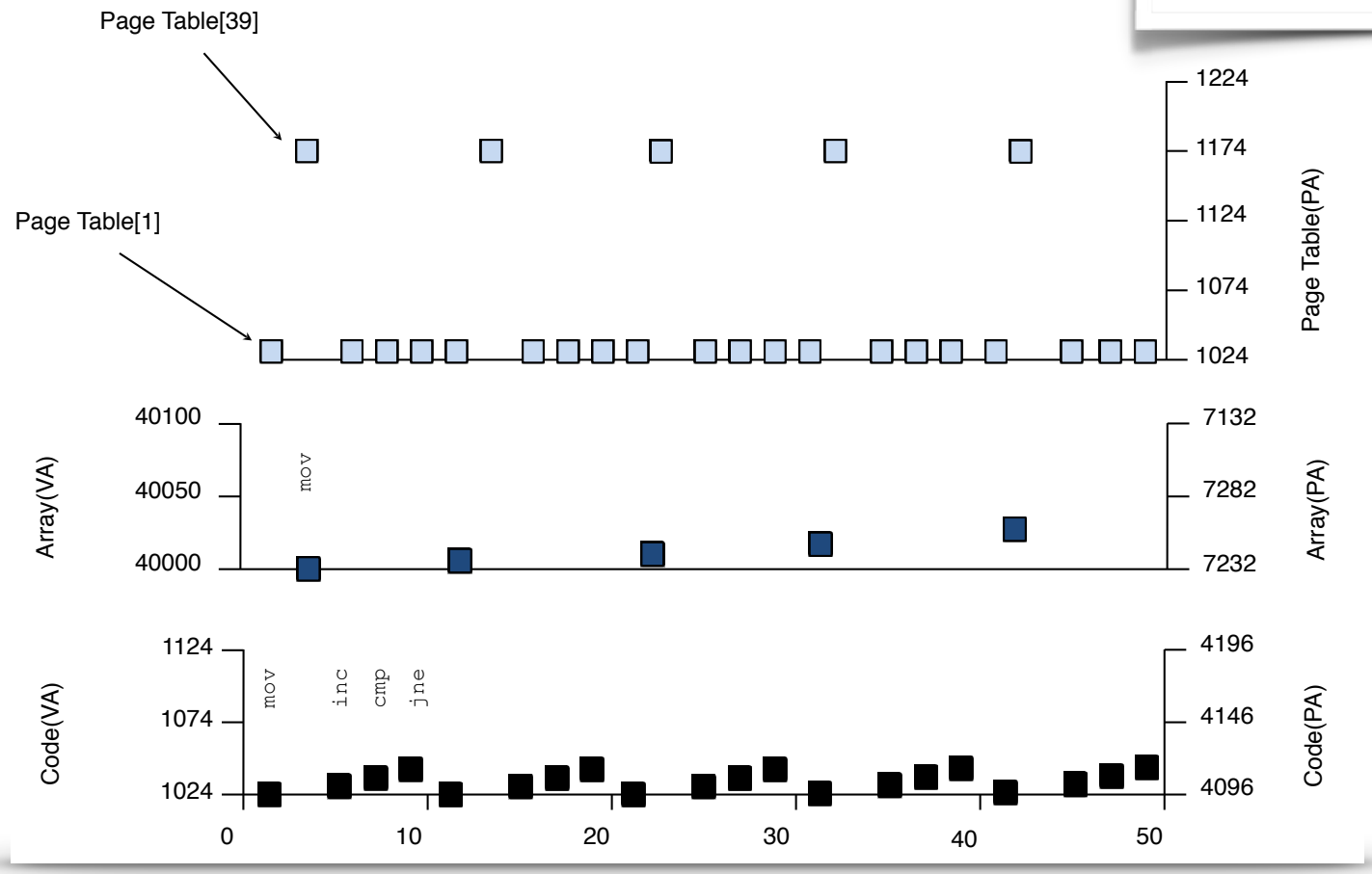
```
int array[1000];
...
for (i = 0; i < 1000; i++)
        array[i] = 0;
```

```
prompt> gcc -o array array.c -Wall -o
prompt>./array
```

```
0x1024 movl $0x0,(%edi,%eax,4)
0x1028 incl %eax
0x102c cmpl $0x03e8,%eax
0x1030 jne 0x1024
```

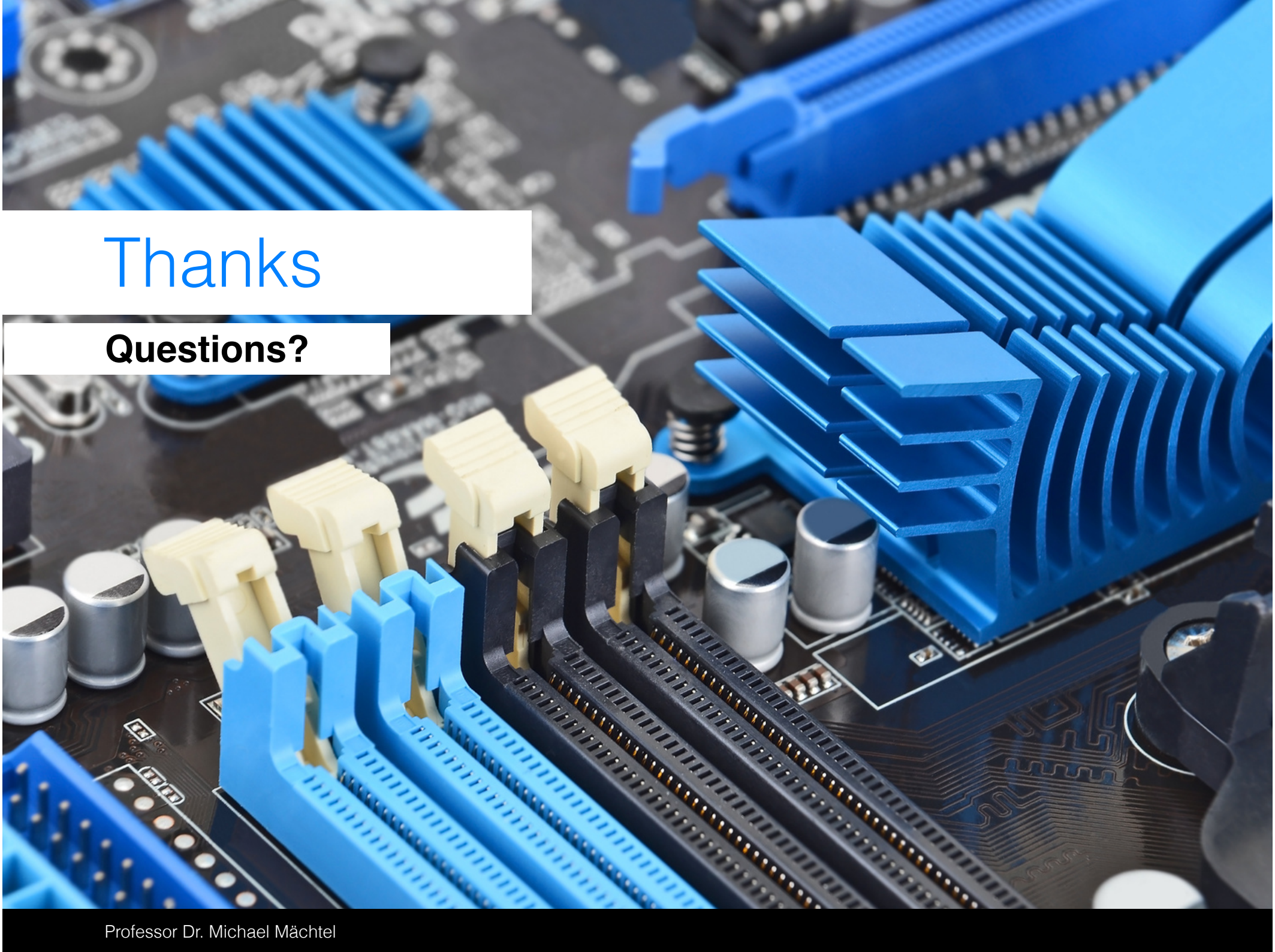
A Virtual(And Physical) Memory Trace

```
0×1024 movl $0×0,(%edi,%eax,4)
0×1028 incl %eax
0×102c cmpl $0×03e8,%eax
0×1030 jne 0×1024
```



Paging: Summary

- Advantages:
  - Does **not** lead to **external fragmentation**
    - as paging (by design) divides memory into fixed-sized units.
  - Is quite **flexible**
    - enabling the sparse use of virtual address spaces.
- Disadvantages:
  - implementing without care will lead to a **slower** machine
    - with many extra memory accesses to access the page table.
  - and to **memory waste**
    - with memory filled with page tables instead of useful application data.



Thanks

Questions?