TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Systemarchitektur, Professur für Betriebssysteme

OPERATING-SYSTEM CONSTRUCTION

The Programming Model of the x86-64 Architecture

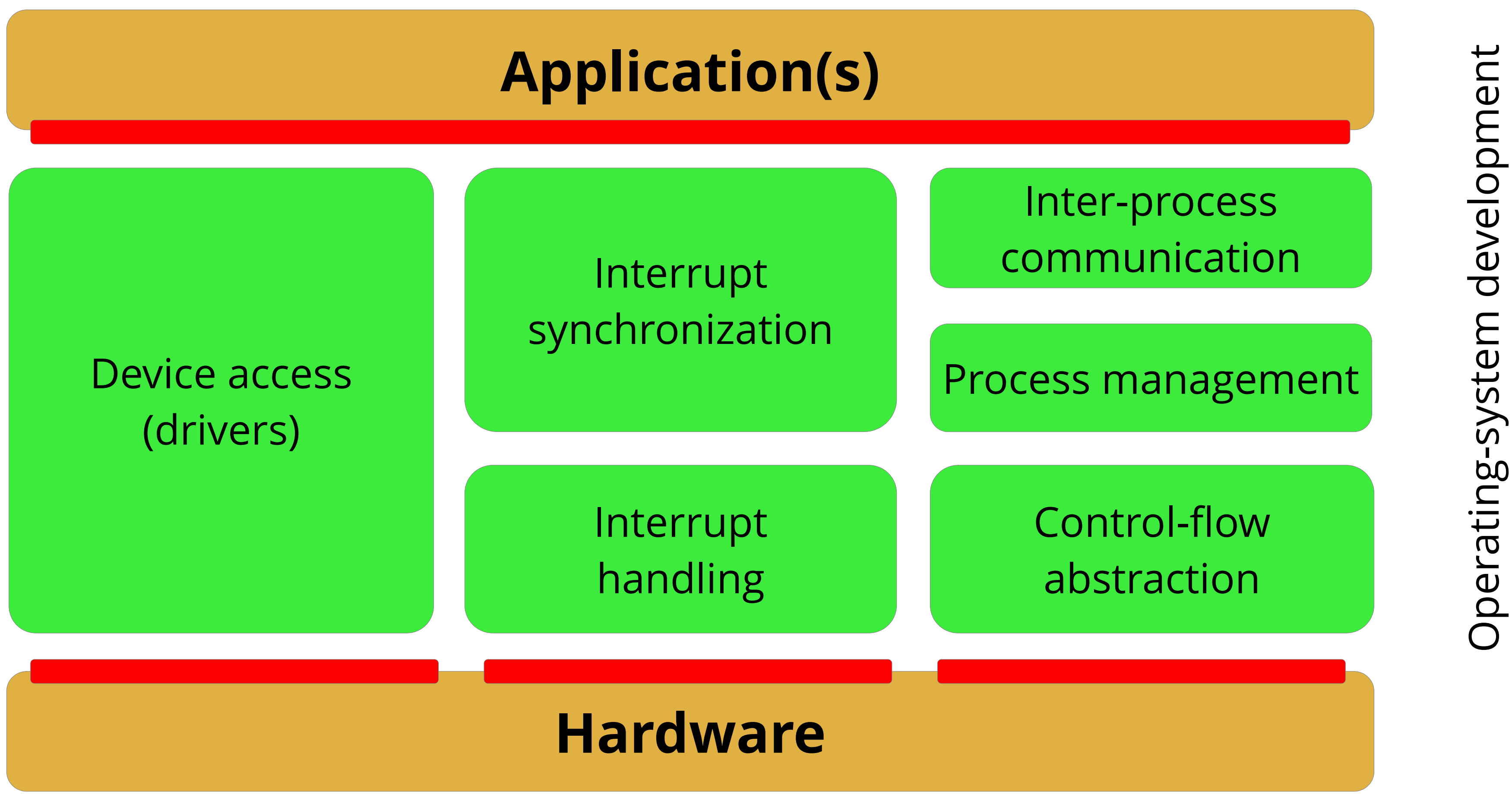
Material based on slides by Olaf Spinczyk, Universität Osnabrück

https://tud.de/inf/os/studium/vorlesungen/betriebssystembau

HORST SCHIRMEIER

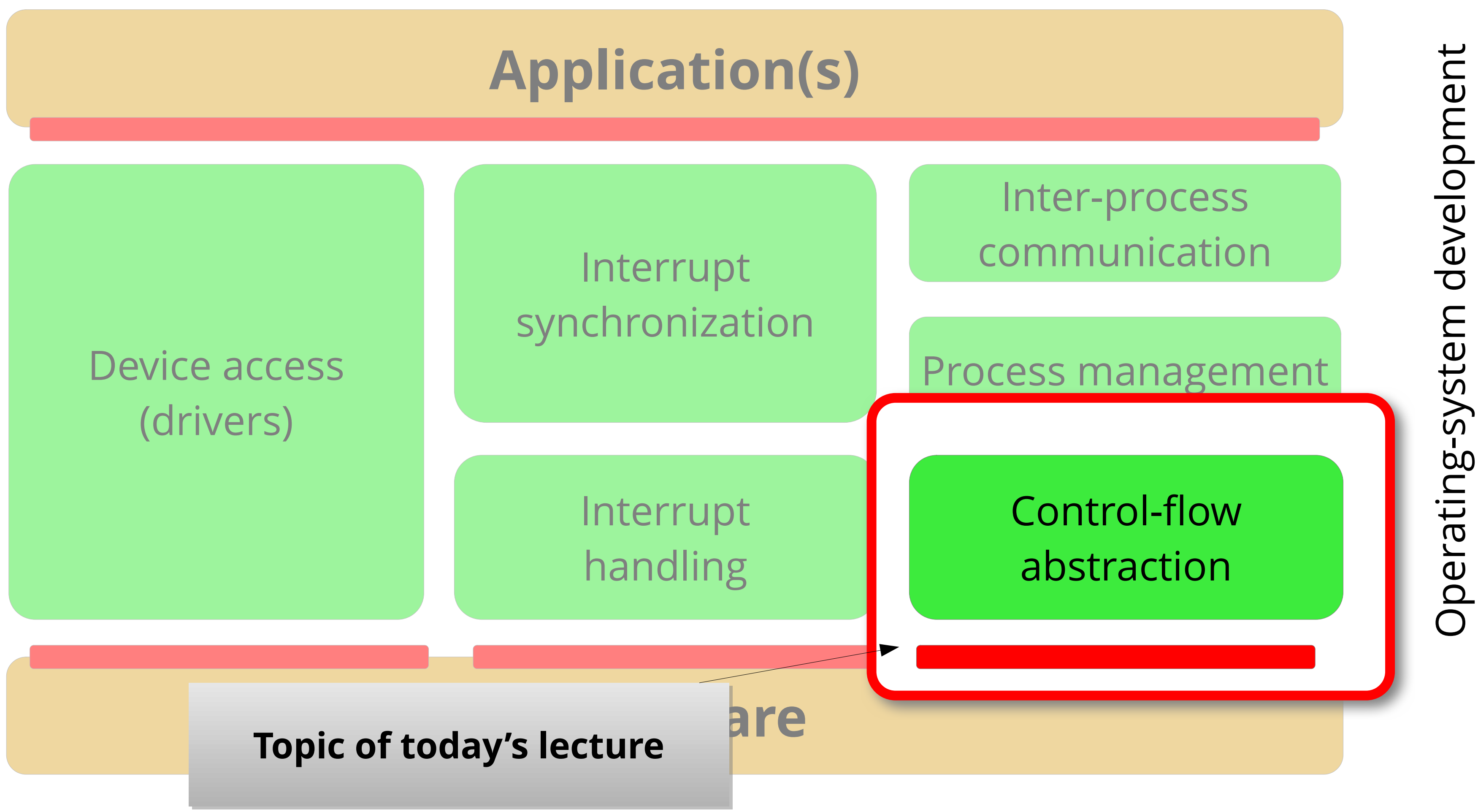
Overview: Lectures

Structure of the “OO-StuBS” operating system:



Overview: Lectures

Structure of the “OO-StuBS” operating system:



Agenda

- History
- Basic Programming Model
- Memory Management and Addressing
- Protection
- “Tasks”
- Summary

Agenda

- **History**
- Basic Programming Model
- Memory Management and Addressing
- Protection
- “Tasks”
- Summary

History of Intel x86 Processors (1)

- **8086** (1978) – the PC processor's prime father
- **80286** (1982) – introduction of **Protected Mode**
  - Segment-based memory protection
- **80386** (1985) – first IA-32 processor
  - Page-based **virtual memory**
- **80486** (1989) – integrated FPU, first RISC rudiments
- **Pentium** (1993) – superscalar, 64-bit data bus
  - SMM, MMX, APIC, dual-processor capable
- **Pentium Pro** (1995) – Server, high-end
- **Pentium II** (1997) – Pentium Pro + MMX
  - RISC-like micro instructions
- **Pentium III** (1999) – SSE
- **Pentium 4** (2000) – Netburst architecture
  - SSE2, Hyperthreading, Vanderpool, Intel 64/EM64T

History of Intel x86 Processors (2)

- **Core** (2005) – Dual-core, low-power
  - Pentium III derivate, SSE3
- **Core 2** (2006) – Dual/quad core, 64 bit (amd64/x86-64)
- **Core i3/i5/i7** (2008) – SSE4, QPI, SMT, L3 cache
  - On-die L3 cache, integrated memory controller and partially also GPU
- **Atom** (2008) – even less power
- **Sandy Bridge** (2011) – AVX, AES-NI
- **Has-/Broadwell** (2013) – AVX2, TSX, FMA3
- **Sky-/Kabylake** (2015) – AVX-512, MPX, SGX, ADX
  - integrated southbridge, USB 3.1, GPU for 3D and 4k video
- **Cannon/Coffee/Whiskey/Cascade Lake** (2017) – up to 8 cores
- **Ice/Comet Lake** (2019) – up to 10 cores, 5-level paging, ~~TSX~~
- **Rocket/Tiger Lake** (2021) – DL-Boost, memory encryption, CET, ~~SGX~~
- **Alder/Raptor Lake** (2021, 2022) – subdivision into P- and E-cores (max. 8+8 / 8+16)

Agenda

- History
- **Basic Programming Model**
- Memory Management and Addressing
- Protection
- “Tasks”
- Summary

8086: Programming Model

- 16-bit architecture, little-endian (LE)
- 20-bit address bus (max. 1 MiB memory addressable)
- Few registers
  - (at least from today's perspective)
- 123 instructions
  - non-orthogonal instruction set
- Opcode lengths of 1 to 4 bytes
- Segmented memory
- Still relevant
  - Although from 1978: still supported by every x86-64 CPU
    - Real Mode, Virtual 8086 Mode

Intel attaches great importance to **backwards compatibility**.

8086: Register File

Instruction and Stack Pointer

15	0
IP	
SP	

Flags register

15	0
FLAGS	

General-purpose registers

15	0
AH	AL
BH	BL
CH	CL
DH	DL
SI	
DI	
BP	

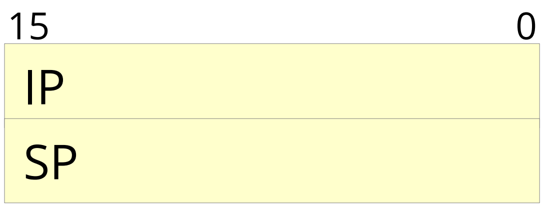
Segment registers

15	0	
CS		Code
SS		Stack
DS		Data
ES		Extra

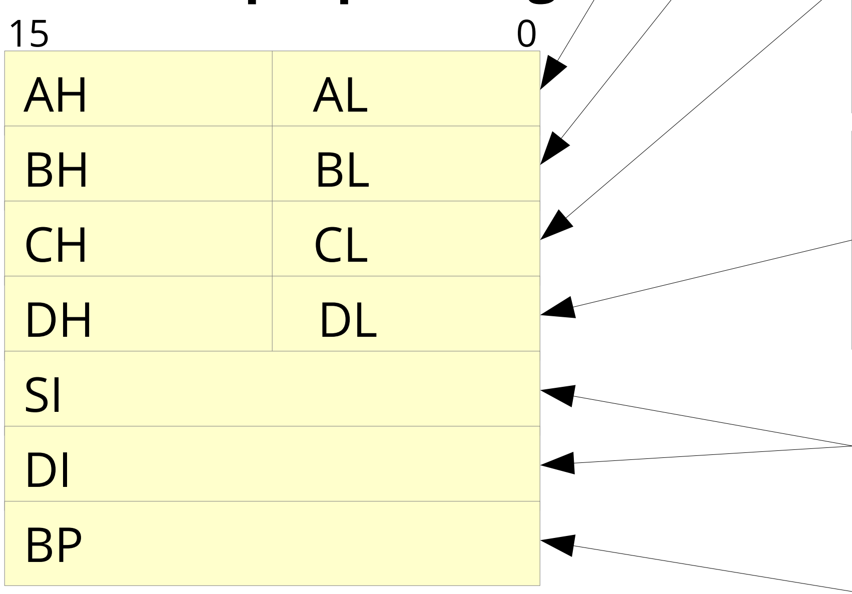
Each **“general-purpose” register** fulfills a specific purpose.

8086: Register File

Instruction and Stack Pointer



General-purpose registers



AX: Accumulator Register
- arithmetic + logical operations
- I/O
- shortest machine code

BX: Base Address Register

CX: Count Register
- for LOOP instruction
- for string operations with REP
- for bit-shift and rotate

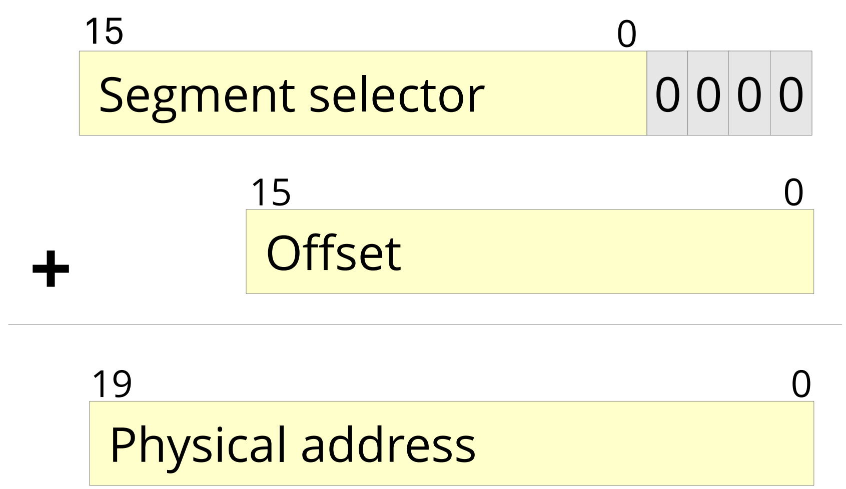
DX: Data Register
- DX:AX have 32 bits for MUL/DIV
- port number for IN and OUT

SI, DI: Index Register
- for array accesses (displacement)

BP: Base Pointer

8086: Segmented Memory

- Logical addresses on 8086 consist of
  - segment selector (usually the value of a **segment register**)
  - offset (usually from a **general-purpose register** or the **instruction**)
- Calculation of physical addresses:



16-bit competitors could usually **address only 64 kiB**.

8086: Segmented Memory

- Logical addresses on 8086 consist of
  - –
  - –
- Cal

“640K ought to be enough for anybody”

wrongly ascribed to Bill Gates, 1981

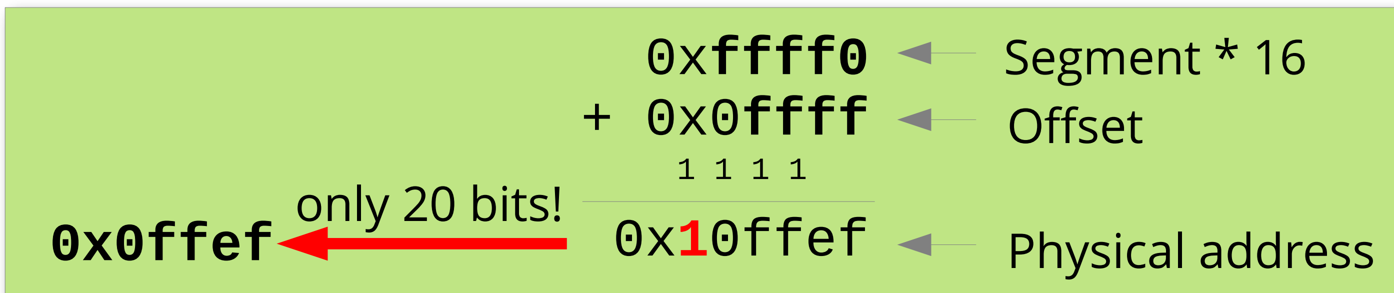
19 0

Physical address

16-bit competitors could usually **address only 64 kiB**.

Interlude: The A20 Gate

- … is a relic from the 80286 era (IBM AT)
  - On the IBM XT (8086), address calculation could overflow, at max.:



  - MS-DOS (and other systems) relies on this overflow “trick”.
  - For compatibility reasons, in the IBM AT the A20 line was masked via the “A20 gate” (a register in the **keyboard controller**).
    - A20 must explicitly be enabled to address memory > 1 MiB.
  - 80486: A20 gate integrated in the CPU
  - Intel Haswell (2013): A20 gate removed

8086: Memory Models

Addressing can be done differently by programs. This resulted in different **memory models**:

- ***Tiny:*** Code, data and stack segments are identical: 64 kiB in total
- ***Small:*** Code separated from data & stack: 64 kiB + 64 kiB
- ***Medium:*** 32- (or in essence 20-) bit **“far” pointers** for code, 16 bit **“near” pointers** for data & stack (fixed 64 kiB segment)
- ***Compact:*** 16-bit **“near” pointers** for code (fixed 64-kiB segment), 32- (20-) bit **“far” pointers** for data & stack
- ***Large:*** **“far” pointers** for everything – 1 MiB completely usable
- ***Huge:*** like “large”, but with **normalized pointers**

```
unsigned char * far videomem =
    (unsigned char * far) 0xb8000;
```

		Data-pointer size	
		near	far
Code-pointer size	near	***Small***	***Compact***
	far	***Medium***	***Large***

8086: Conclusion

- The PC processor's prime father
  - The first PC's CPU
  - Today, x86 and x86-64 processors are still compatible.
- Advantages through segment registers
  - 1 MiB of memory in spite of 16-bit architecture
  - Segments separate logical modules in memory
- Difficult program and compiler development
  - Different memory models
  - Non-orthogonal instruction set

IA-32 – Intel's 32-bit Architecture

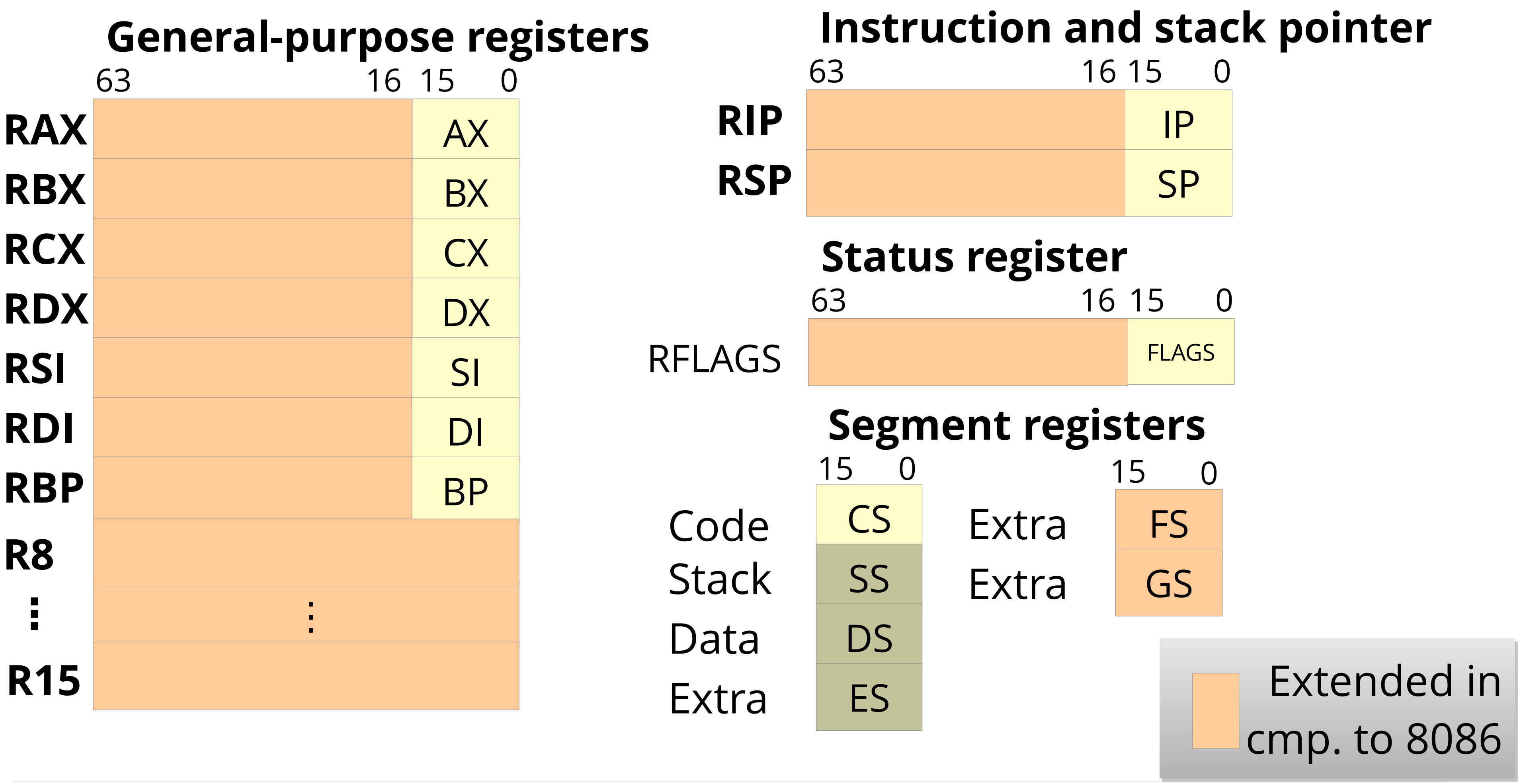
- First IA-32 CPU: **Intel 80386**
  (the term "IA-32" was coined much later, however)
- 32-bit technology: Registers, data and address bus
  - starting with Pentium Pro: 64-bit data and 36-bit address bus
- Additional registers
- Complex support for protection and **multitasking**
  - **Protected Mode**
  - originally introduced with the 80286 (16 bit)
- Compatibility
  - with older operating systems through **Real Mode**
  - with older applications through **Virtual 8086 Mode**
- Segment-based programming model
- **Page-based MMU**

amd64 / x86-64 – 64-Bit AMD/Intel Arch.

- IA-32 was limited to max. 4 GiB virtual memory
- Solution: 64-bit architecture
  - Currently 48-bit virtual address space (up to 256 TiB) (Ice Lake Xeons, 2019: 57-bit, up to 128 PiB)
- More (and 64 bits wide) registers
- Page-based programming model with memory protection
  - **No Execute** bit for individual pages
  - (Almost) no segmentation
- Still backwards compatible
  - **Long Mode** for putting the new features to use
  - 32-bit **Legacy Mode** for systems or individual applications
- Developed by AMD, adopted by Intel (as “Intel 64”)
  - Usually called “x86-64” or “x64” nowadays
  - The **completely different Intel IA-64 (“Itanium”)** did not succeed.

x86-64: Register File (Extensions)

- Extended registers prefixed with R… for compatibility



x86-64: Register File (Additions)

Memory-management registers

	15 … 0	63 … 0	31 … 0 (15 … 0)
TR	TSS sel.	TSS Base Address	TSS Limit
LDTR	LDT sel.	LDT Base Address	LDT Limit
IDTR		IDT Base Address	IDT Limit
GDTR		GDT Base Address	GDT Limit

Details follow …

Control registers

	63 … 32	31 … 0
CR8		
CR4		
CR3		
CR2		
CR0		

Debug registers

	63 … 32	31 … 0
DR0		
DR1		
DR2		
DR3		
DR6		
DR7		

Model-specific registers (MSRs)

Agenda

- History
- Basic Programming Model
- **Memory Management and Addressing**
- Protection
- “Tasks”
- Summary

x86-64: Addressing Modes

- The CPU calculates **effective addresses** (EA) along a simple formula
  - all general-purpose registers can be used equally (!)

EA := Base-Reg. + (Index-Reg. * Scale) + Displacement

1/2/4/8

1/2/4 bytes

EA

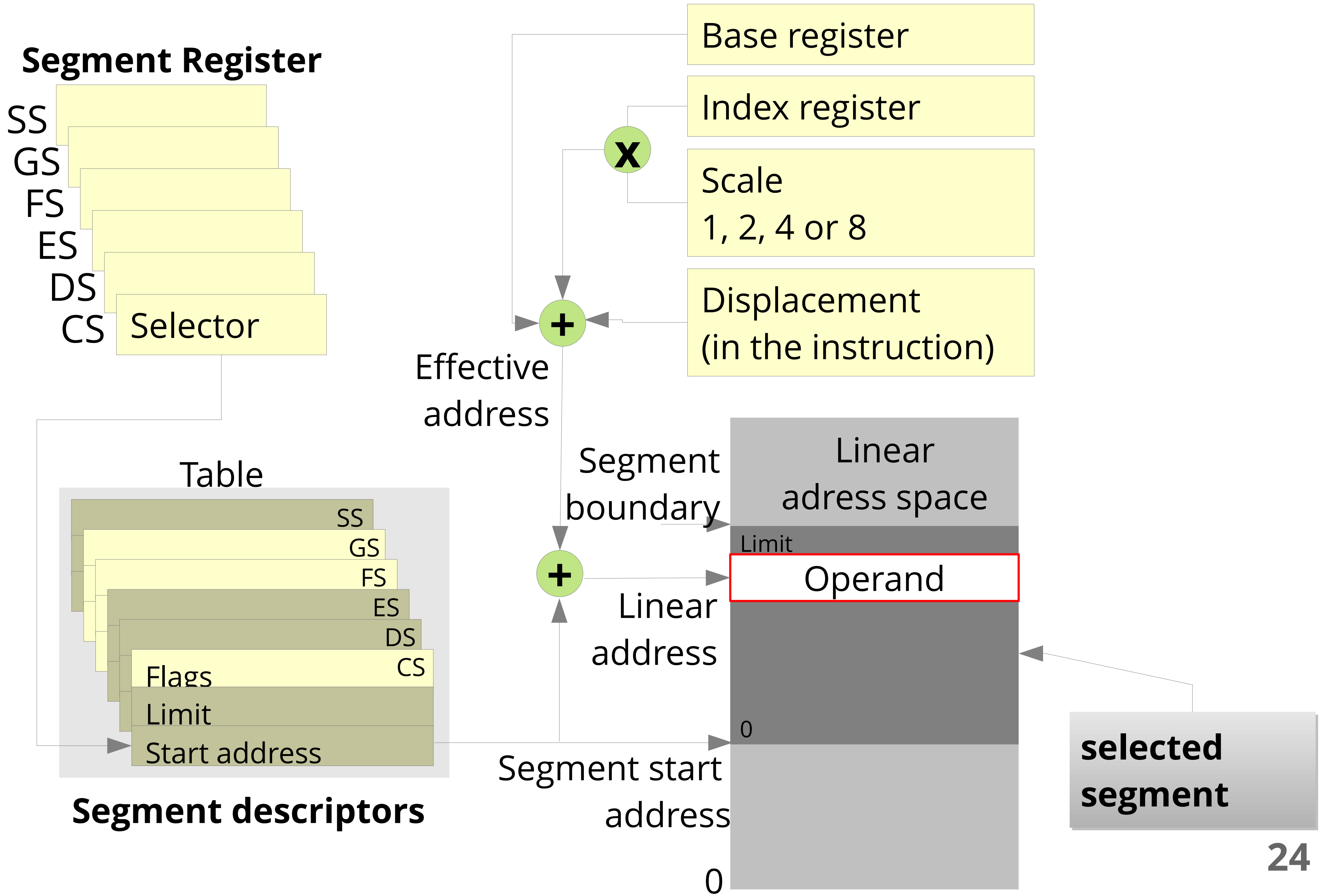
- Example: `MOV RAX, array[RSI * 4]`
  - Read from array with 4-byte elements, using RSI as index
- New with x86-64: IP-relative addressing

EA := RIP + Displacement

IA-32: *Protected Mode* – Segments

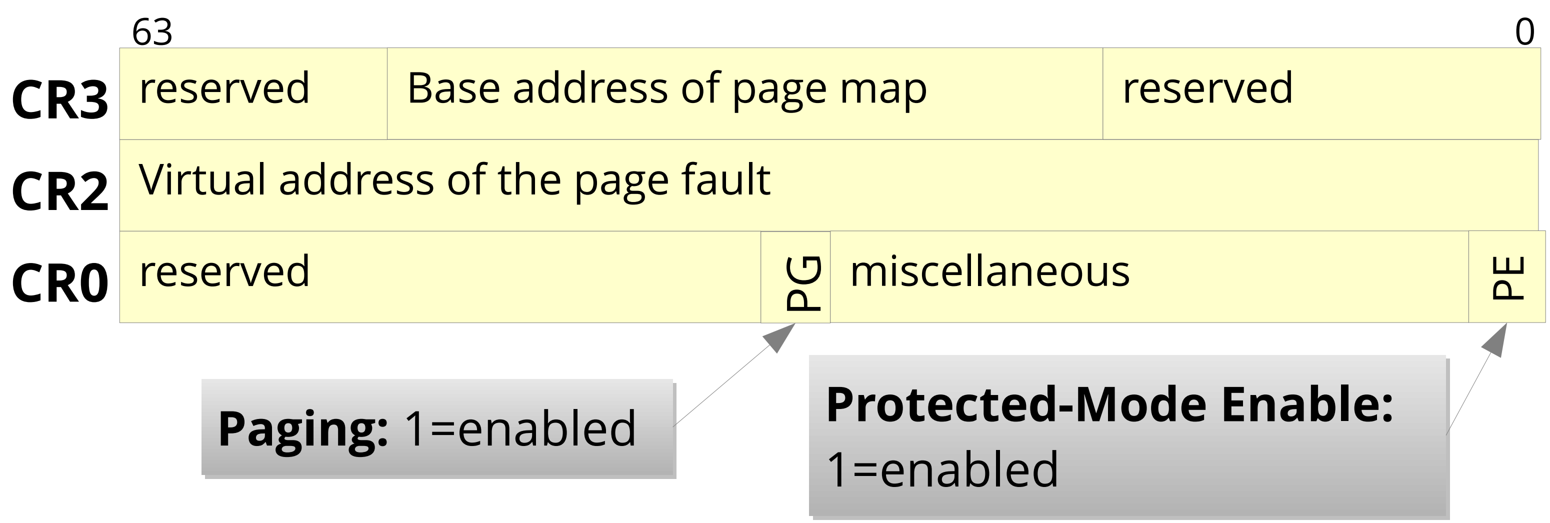
- A (running) program consists of multiple memory segments
  - Traditionally at least CODE, DATA and STACK
  - Segment selectors (indirectly) describe address and size
- "Linear address" is segment start address + EA
  - Corresponds to physical address if **paging unit** is disabled
  - Segments may overlap, e.g. start addresses == 0
  - In practice, such a "flat address space" is often used.
- No segment-based protection in Long Mode
  - Only FS/GS start address and CS attributes are respected

IA-32 / x86-64: Segments

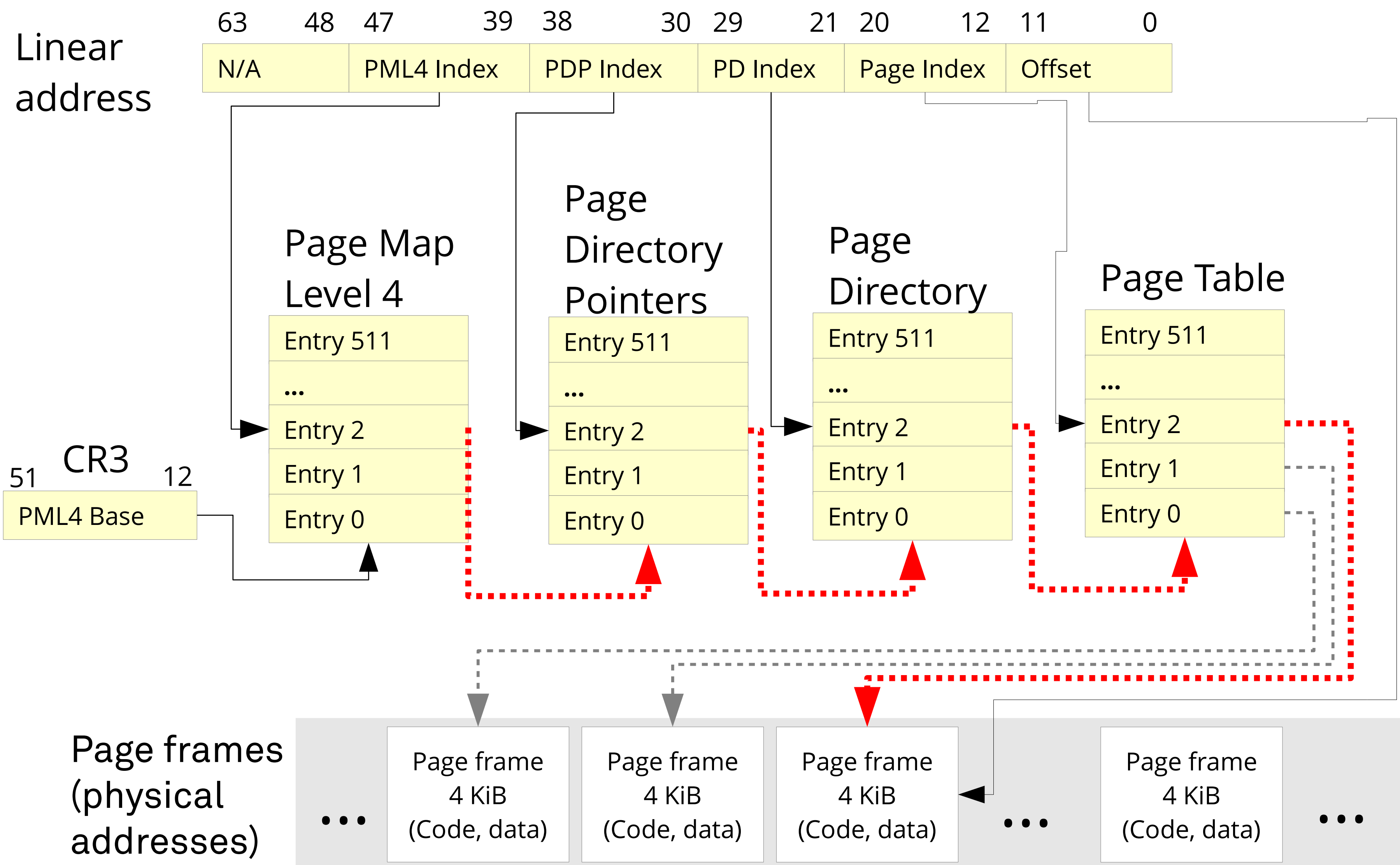


x86-64: Page-based MMU

- **Demand paging** (for virtual memory)
- 80386: ***Paging Unit*** (PU) can be enabled optionally
  - x86-64: PU obligatory
- Configured via control registers CR0/2/3:



x86-64: Page Table



x86-64: TLB

- Problem: Indirection via *PML4*, *PDP*, *Page Directory* and *Page Table* slows down memory accesses
- Solution: the ***Translation Lookaside Buffer*** (TLB):
  - an associative cache
    - Tag: PML4/PDP/PD/PT
    - Data: Page-frame address
    - Size and associativity depend on CPU
  - For regular applications, the TLB achieves a hit rate of ~98%.
  - Writing CR3 invalidates the TLB
    - Not anymore since Intel Westmere (2010) and AMD Zen 3 (2020): TLB tags include 12-bit process-context ID (PCID)

Agenda

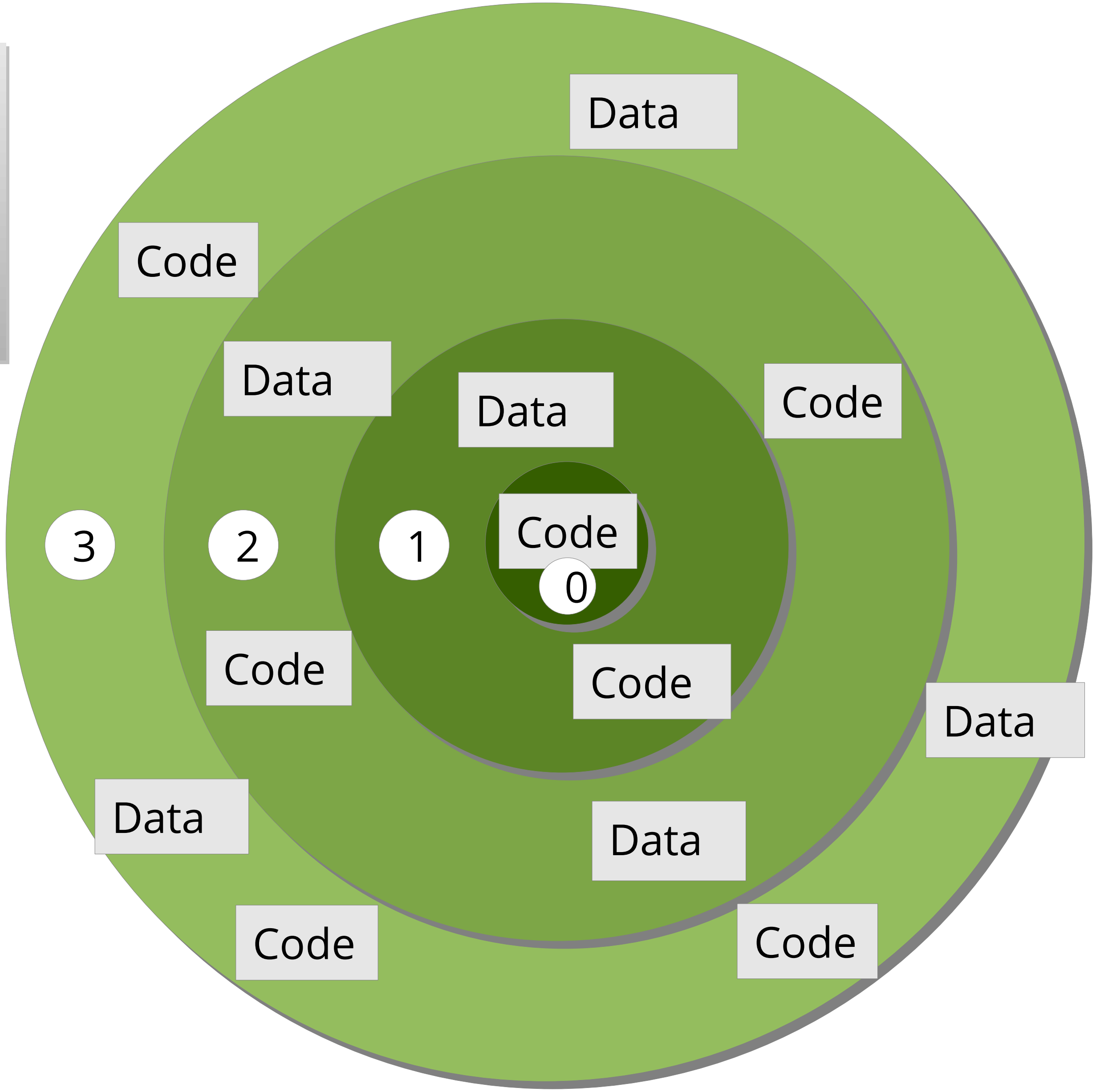
- History
- Basic Programming Model
- Memory Management and Addressing
- **Protection**
- “Tasks”
- Summary

Protection on IA-32

- Protection concept is a central property of *Protected Mode*
- **Goal**: Isolate buggy or untrusted code
  - Protect from system crashes
  - Protect from unauthorized data accesses
  - Prevent unauthorized operations, e.g. I/O port accesses
- **Preconditions**: Code and data …
  - are categorized depending on their trustworthiness
  - have an owner (cf. “multitasking”)

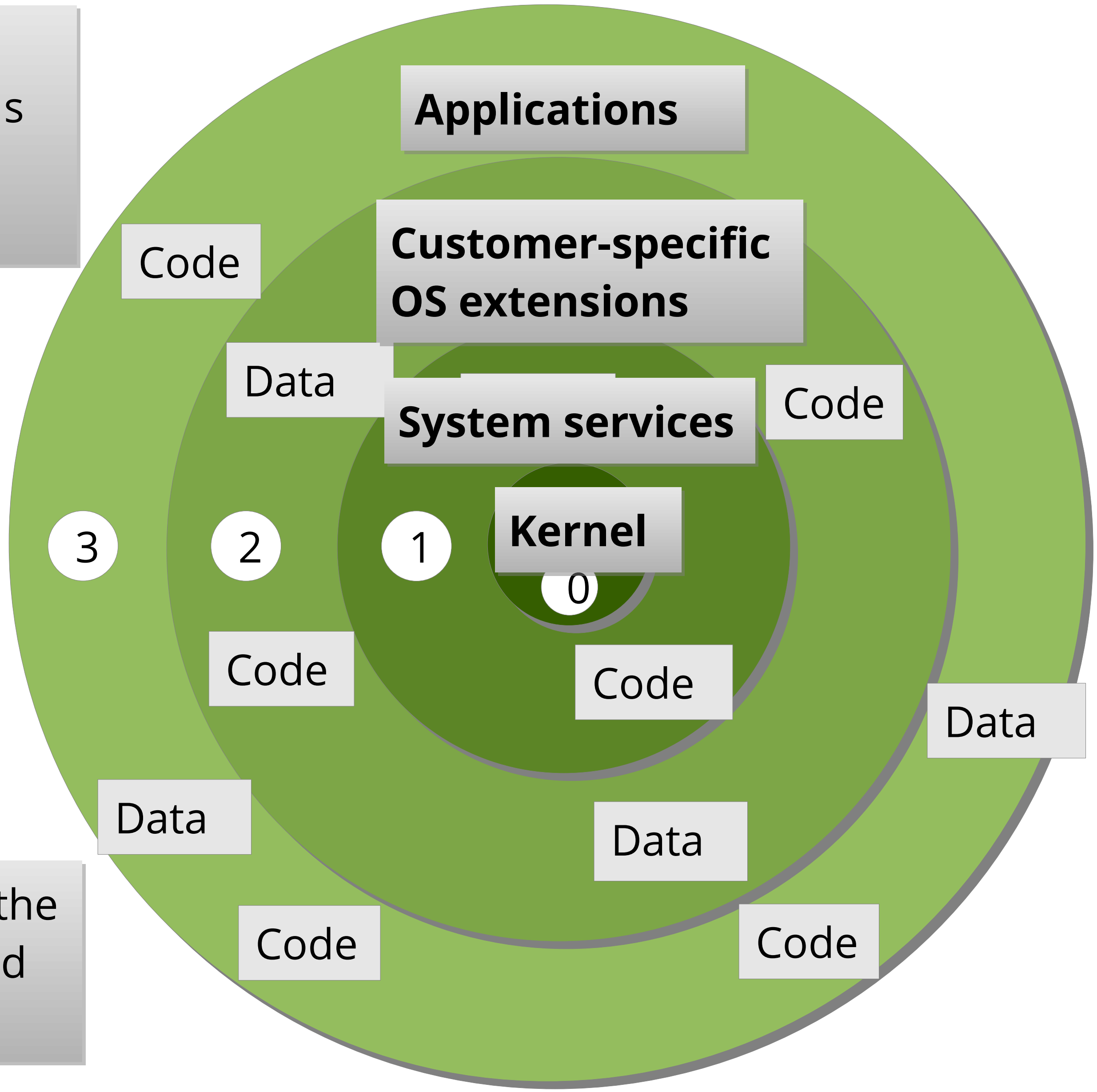
Protection Rings and Gates

A 2-bit entry in the segment descriptor assigns each segment to a **privilege level**.



Protection Rings and Gates

Privilege level 3 is the lowest of the four levels and meant for **applications**.

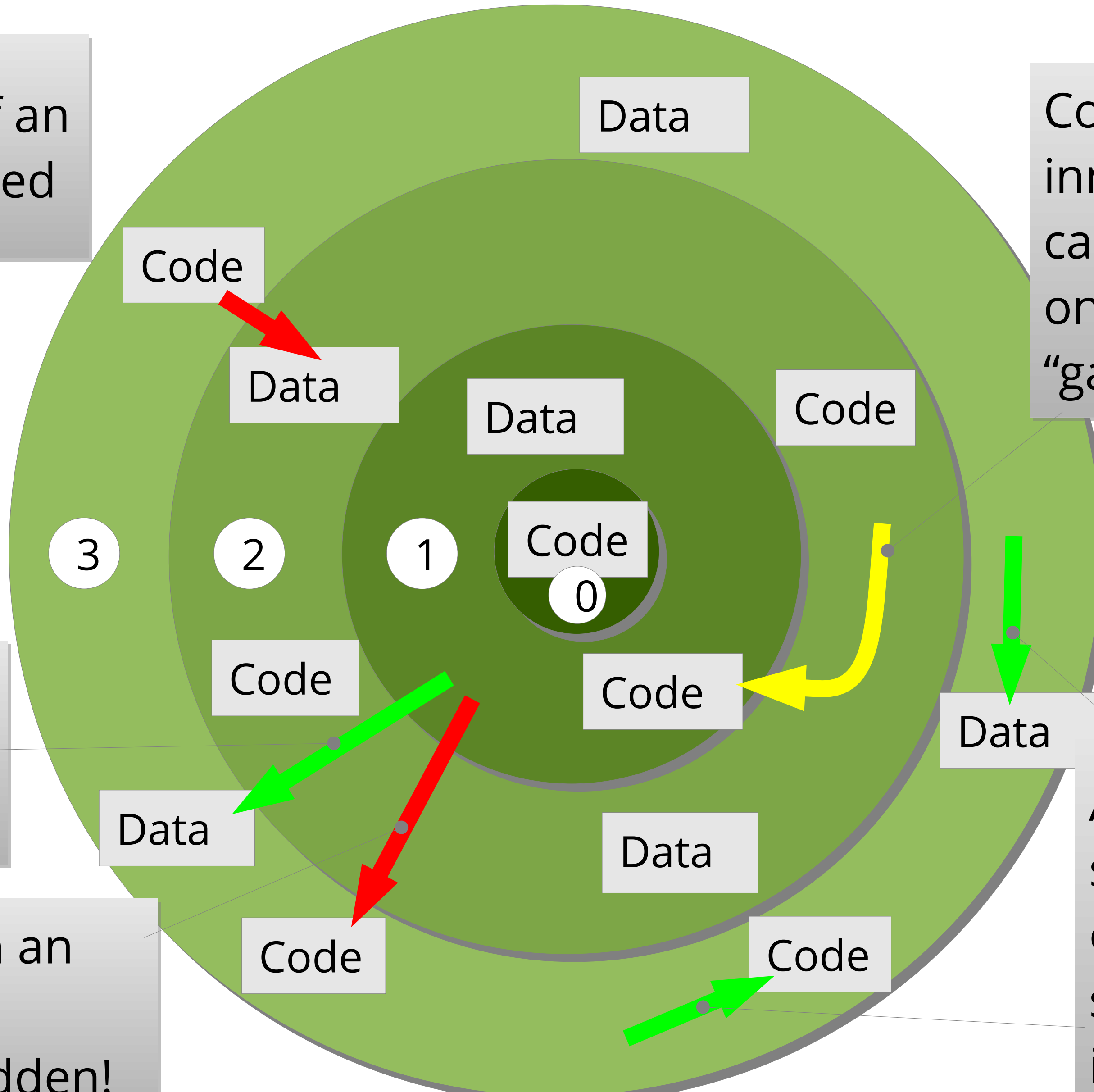


Privilege level 0 is the highest and reserved for the **OS kernel**.

Protection Rings and Gates

Access to data of an inner ring is denied

Code in an inner ring can be called only via “gates”



Access to data of an outer ring is allowed

Calling code in an outer ring is (usually) forbidden!

Access to segments of the same ring is allowed

Segment Descriptors

- Allow protecting code in Protected Mode
  - Any violation triggers an exception
  - On 64-bit systems, they allow executing 32-bit code

a **segment descriptor**

<table>
<tr><td colspan="4">Segment Base 31 … 24</td><td>G</td><td>D/B</td><td>L</td><td>AVL</td><td>Limit 19 … 16</td><td>+6</td></tr>
<tr><td>P</td><td>DPL</td><td>S</td><td>TYPE</td><td colspan="5">Segment Base 23 … 16</td><td>+4</td></tr>
<tr><td colspan="9">Segment Base 15 … 0</td><td>+2</td></tr>
<tr><td colspan="9">Segment Limit 15 … 0</td><td>0</td></tr>
</table>

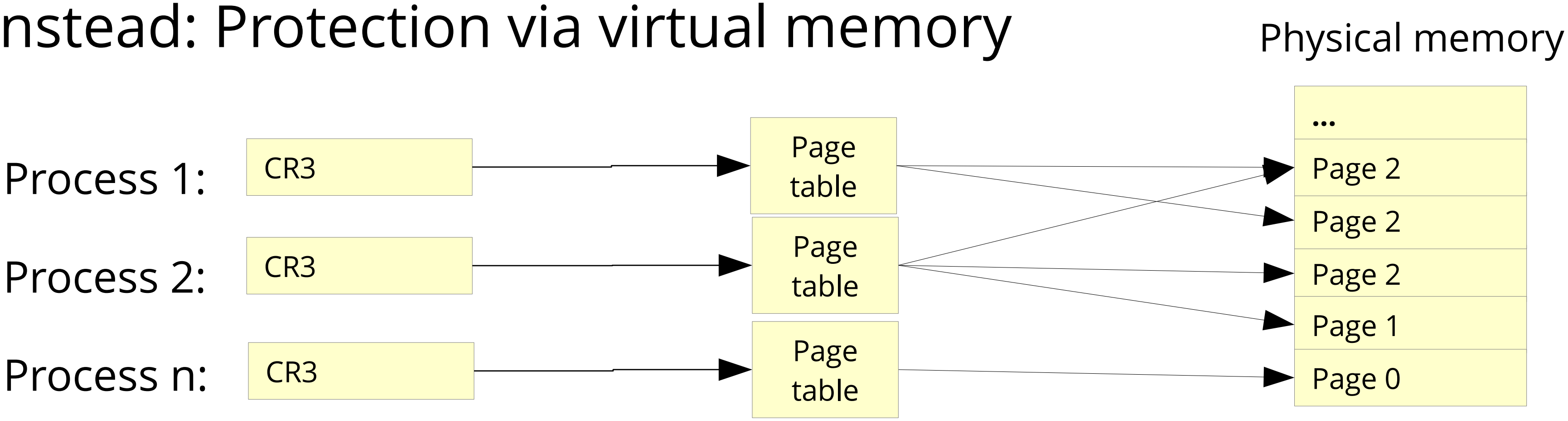
TYPE – Code:
C - Conforming
R - Readable
A - Accessed

P - Present Bit
DPL - Descriptor Privilege Level
S - System Segment

G - Granularity
D/B - 16/32 Bit Seg.
L - Long Mode aktiv

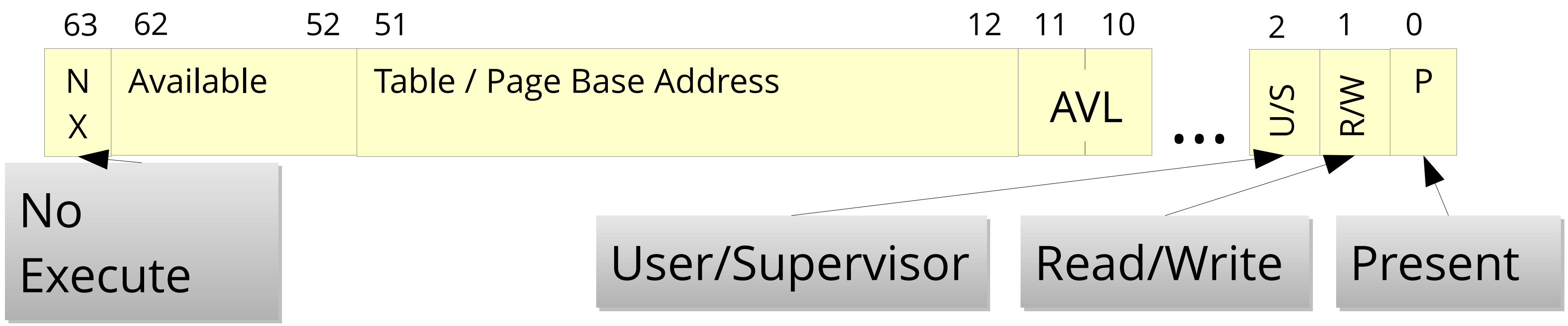
Protection on x86-64

- Segmentation was barely adopted in practice
  - Almost all IA-32 systems use a flat memory model
  - Offset of a logical address = linear address
- Instead: Protection via virtual memory



- Every process only sees its virtual address space
  - Page table is hidden from application software
  - Additional restrictions possible on page granularity

Page-Table Entries



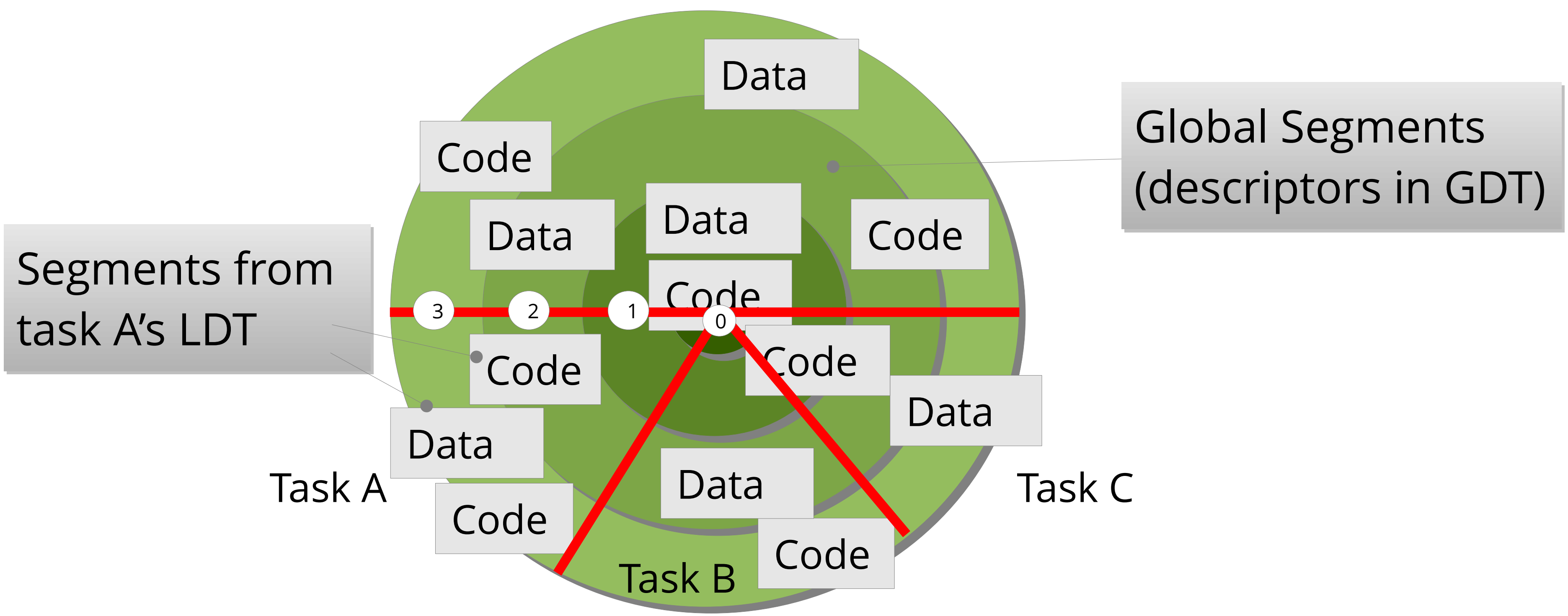
- Settings possible on all hierarchy levels
  - Data must not be executed as code (NX=1)
  - Protection from read accesses from ring 3 (U/S=0)
  - Read-only pages (R/W=0)
- Allows implementing **shared memory**
  - Or read-only access to operating-system structures
- Effectively, only ring 0 (kernel) and 1–3 (application) are separated

Agenda

- History
- Basic Programming Model
- Memory Management and Addressing
- Protection
- **“Tasks”**
- Summary

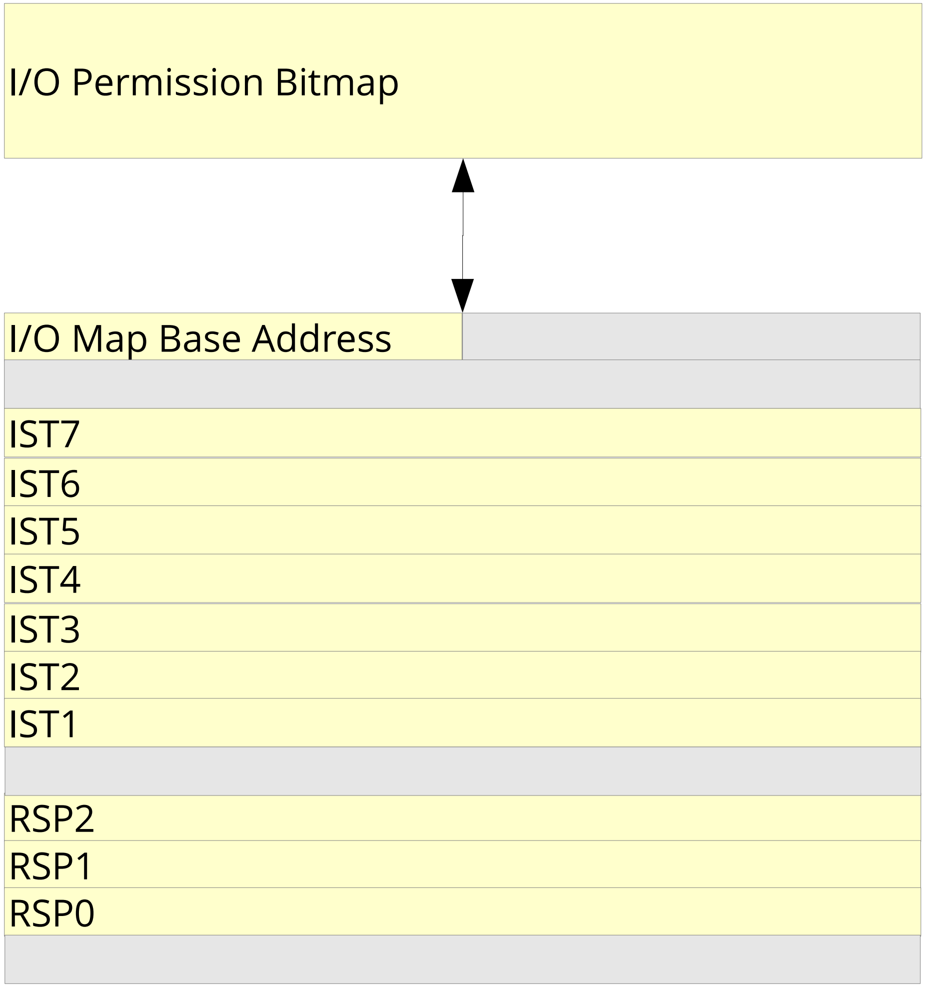
IA-32: Multitasking

- Besides protecting from unauthorized "vertical" accesses between segments on different ring levels, IA-32 also supports a **task concept** ("horizontal isolation")
- Not available in x86-64 Long Mode



Task-State Segments

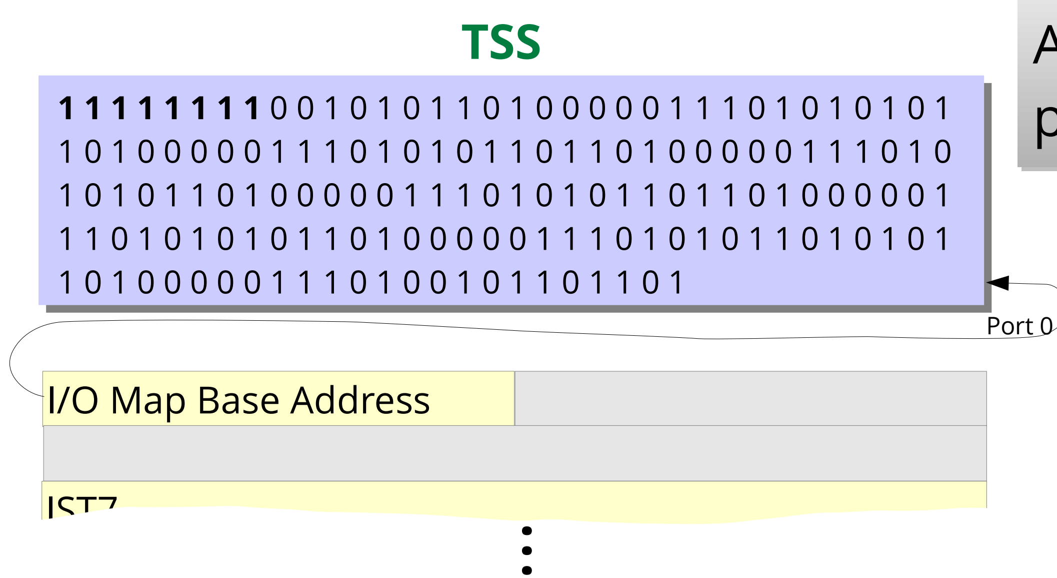
- Task Register (TR) points to the **Task State Segment** (TSS)
- on IA-32: Storage space for task state
  - Segments, register contents, ...
  - Task switches possible completely in hardware!
- ... not in x86-64 Long Mode
  - instead: TSS for I/O permissions and stack pointer



Input/Output in Long Mode

- Not every task is allowed to do I/O!
- Access to devices in memory (**memory-mapped I/O**) controllable via memory protection
- Access to **I/O ports** restricted by:
  - *I/O Privilege Level* bits in RFLAGS → I/O OK when on specified protection rings
  - Other rings: I/O Permission Bitmap in TSS controls access

Bitmap ends with the TSS segment's end, ports with higher numbers must not be accessed.



A 1 prevents port access

x86-64: What Else Is There?

- **5-level paging** (in some high-end CPUs since 2019)
  - Up to 4 PiB memory (128 PiB address space) instead of currently 64/256 TiB
- CPU Virtualization
  - **Virtual 8086 Mode**
    - 16-bit applications or OSs run as tasks in a protected environment
  - **Intel-VT**, **AMD-V**
    - Hardware support for virtual-machine solutions like VMware, VirtualBox or Xen
    - Allows running ring-0 Protected Mode code in a VM (hypervisor on **"ring –1"**)
- **System Management Mode** (SMM)
  - Hands control to the system to the firmware/BIOS (on **"ring –2"**)
  - … unbeknownst to the OS
  - System safety (high temp. shutdown), USB Legacy Support, TPM, …
- (Near?) Future: **X86-S architecture** proposal
  - Long-mode only, removal of a lot more legacy

Agenda

- History
- Basic Programming Model
- Memory Management and Addressing
- Protection
- “Tasks”
- **Summary**

Summary

- The x86-64 architecture is **highly complex**:
  - Virtual memory with 4-level (or even 5-level) page tables
  - Page-based memory protection
  - I/O port protection per task
  - Can run 16-bit code on 32-bit systems or 32-bit code on 64-bit systems (legacy modes)
- Rarely used IA-32 features were removed in x86-64
  - Segmentation (many systems use a flat address space)
  - Task switch in hardware
- Nevertheless: consequent **backwards compatibility**
  - "Legacy / Virtual 8086 Mode", "PIC", "A20 Gate"
  - May finally go away with X86-S