



**TECHNISCHE
UNIVERSITÄT
DRESDEN**

Faculty of Computer Science Institute of Systems Architecture, Operating Systems Group

EXERCISE: L4 BOOTCAMP

CARSTEN WEINHOLD

- first contact with a microkernel OS
- talk about system booting
- getting to know QEMU
- compile Fiasco
- compile minimal system environment
- the usual „Hello World“
- look at source and config, play with it

- developing your own kernel usually requires a dedicated machine
- we will use a virtual machine
- QEMU is open-source, provides a virtual machine by binary translation
- it emulates a complete x86 PC
- ... many other system architectures, too
- our QEMU will boot from an ISO image

BOOTING

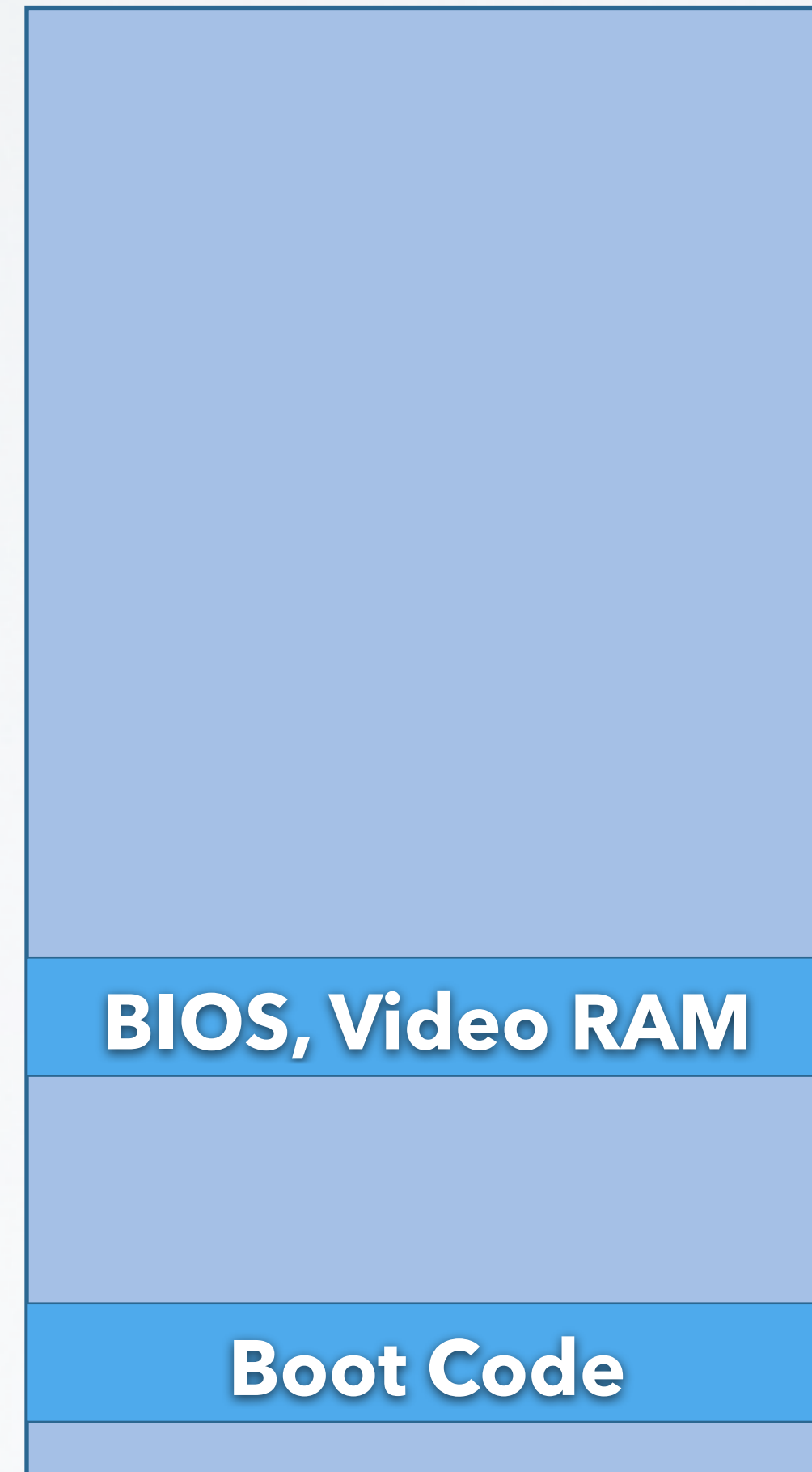
- Basic Input Output System
- fixed entry point after „power on“ and „reset“
- initializes the CPU in 16-bit real-mode
- detects, checks, and initializes platform hardware (RAM, PCI, ATA, ...)
- finds the boot device

A dark teal rectangular box with the word "BIOS" in white, bold, sans-serif capital letters.

- Extensible Firmware Interface
 - plug-ins for new hardware
- no legacy PC-AT boot (no A20 gate)
- built-in boot manager
 - more than four partitions, no 2TB limit
 - boot from peripherals (USB)

- first sector on boot disk
- 512 bytes
- contains first boot loader stage and partition table
- BIOS loads code into RAM and executes it
- problem: How to find and boot an OS in 512 bytes?

A dark teal rectangular box with a white border and a subtle gradient shadow, containing the text "BIOS" in white.



PHYSICAL MEMORY



- popular boot loader
- used by most (all?) Linux distributions
- uses a two-stage-approach
 - first stage fits in one sector
 - has hard-wired sectors of second stage files
 - second stage can read common file systems

Boot Loader

BIOS

- second stage loads a menu.lst config file to present a boot menu
- from there, you can load your kernel
- supports loading multiple modules
- files can also be retrieved from network

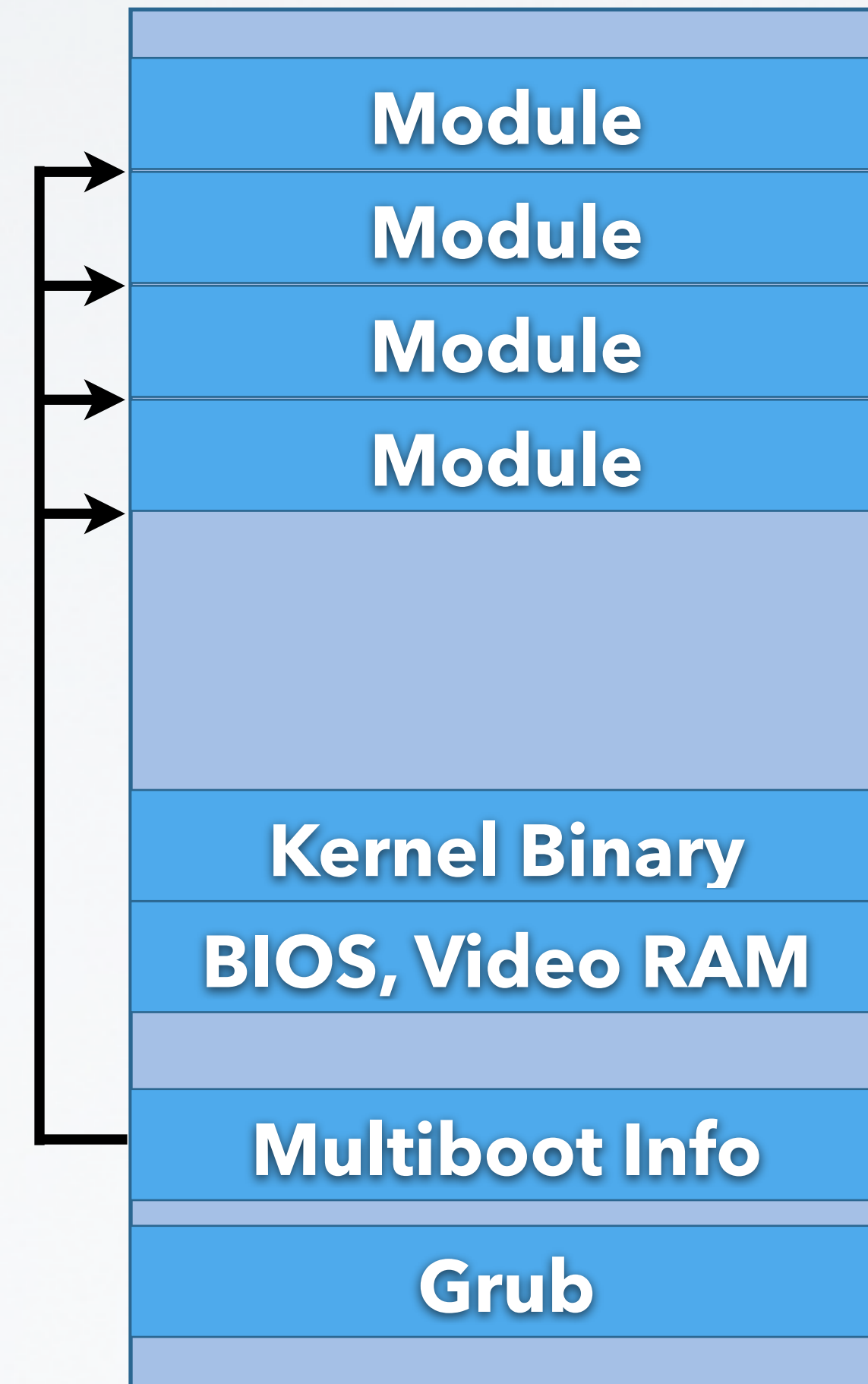
Boot Loader

BIOS

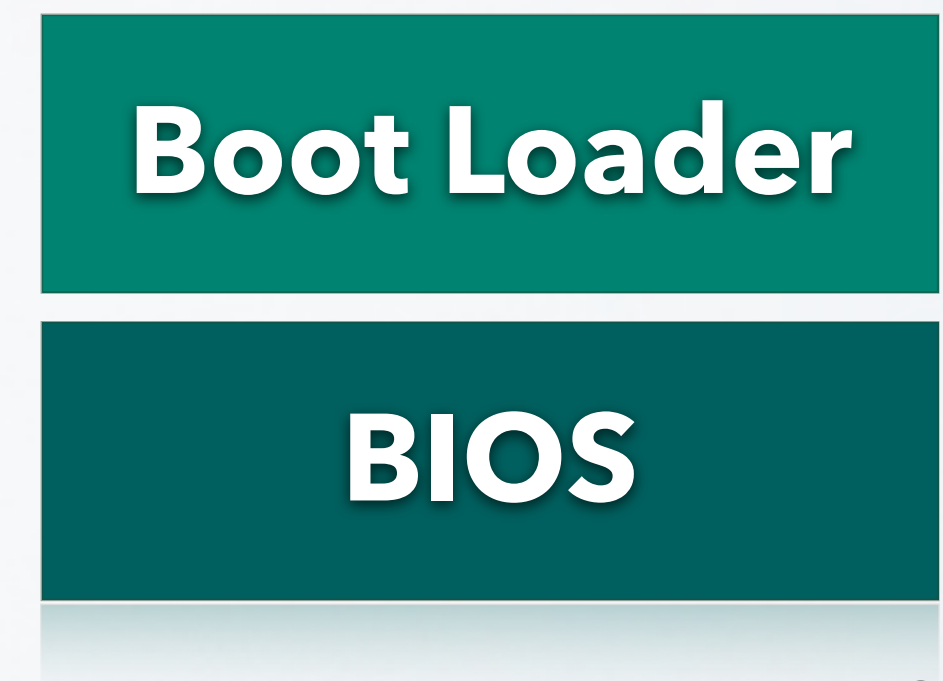
- switches CPU to 32-bit protected mode
- loads and interprets the „kernel“ binary
- loads additional modules into memory
- sets up multiboot info structure
- starts the kernel

Boot Loader

BIOS



PHYSICAL MEMORY



- our modules are ELF files: executable and linkable format
- contain multiple sections
 - code, data, BSS
- bootstrap interprets the ELF modules
- copies sections to final location in physical memory

Bootstrap

Boot Loader

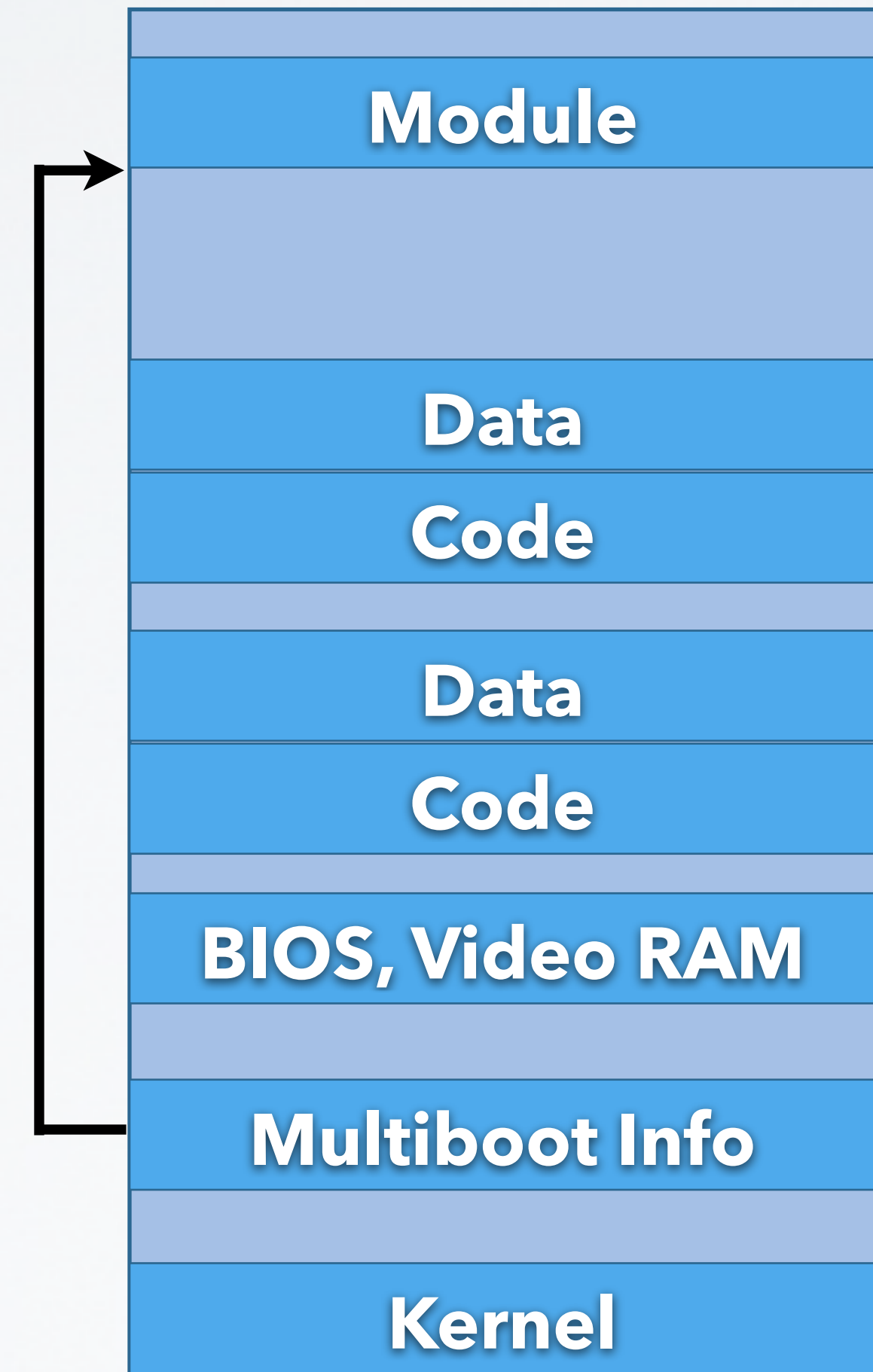
BIOS

- actual L4 kernel is the first of the modules
- must know about the other modules
- bootstrap sets up a kernel info page
 - contains entry point + stack pointer of sigma0 and moe
- passes control to the kernel

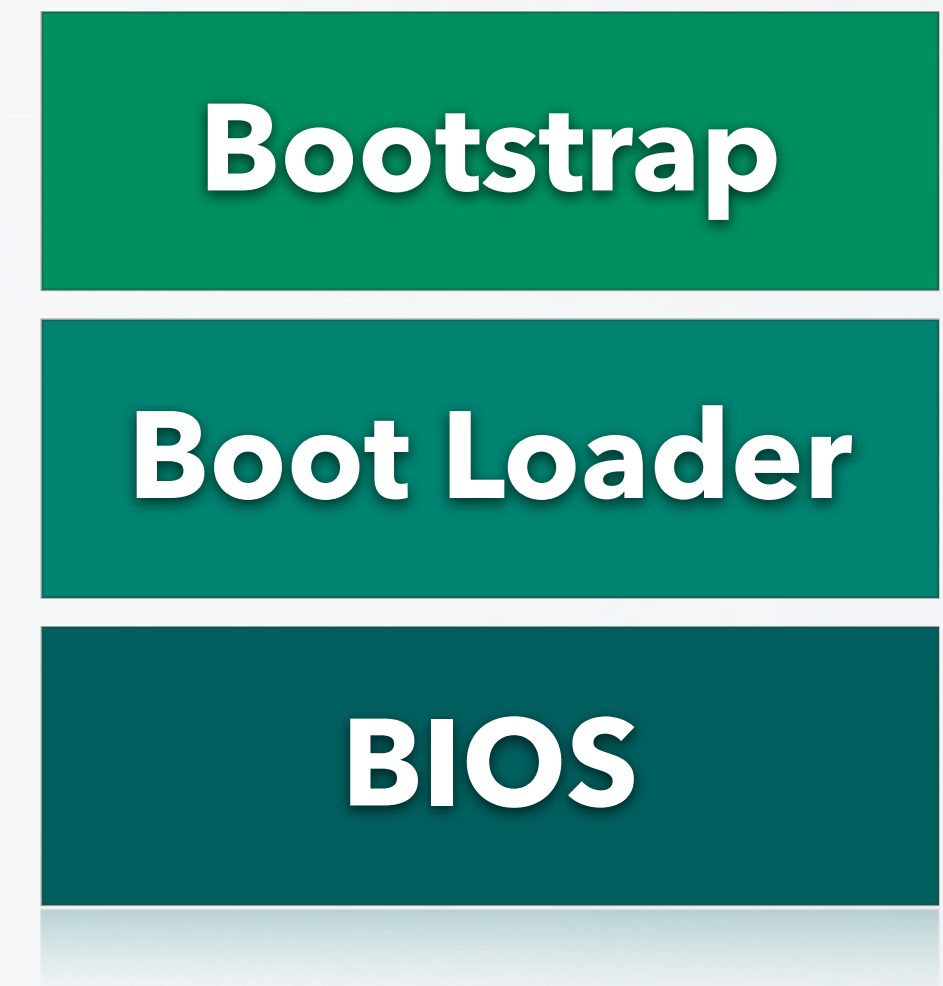
Bootstrap

Boot Loader

BIOS



PHYSICAL MEMORY



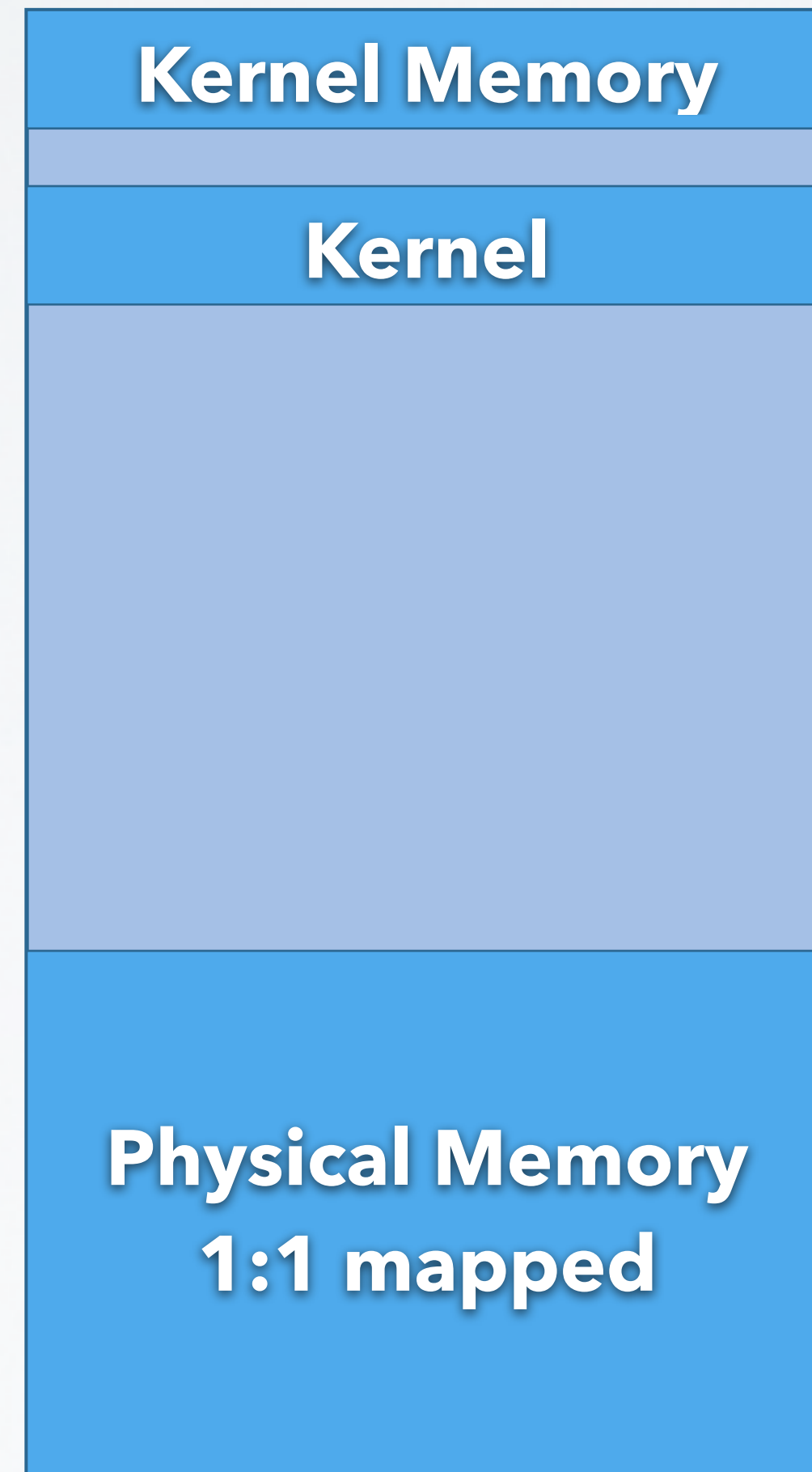
- initial kernel code
- basic CPU setup
 - detecting CPU features
 - setup various CPU-tables
- sets up basic page table
- enables virtual memory mode
- runs the actual kernel code

Kernel Loader

Bootstrap

Boot Loader

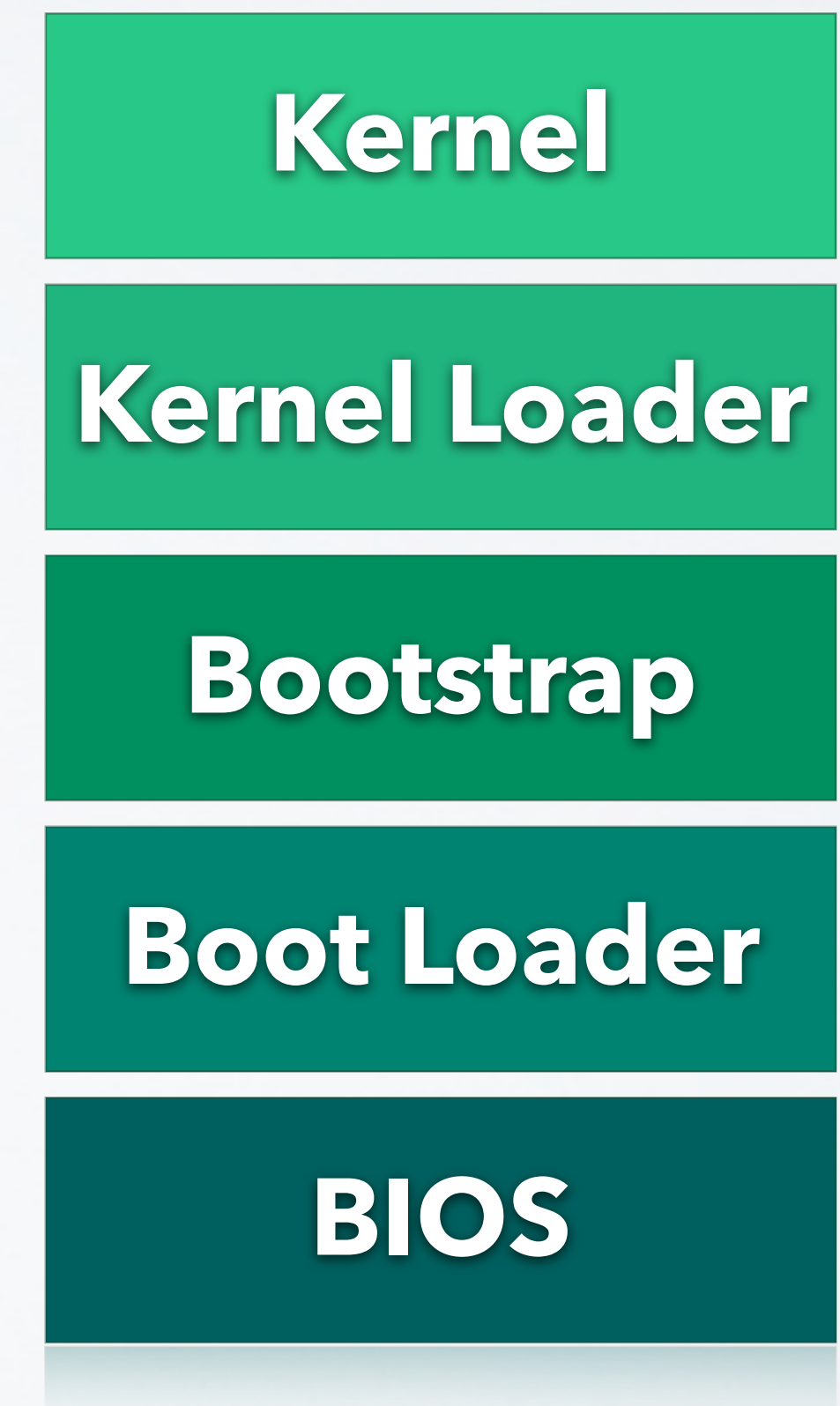
BIOS



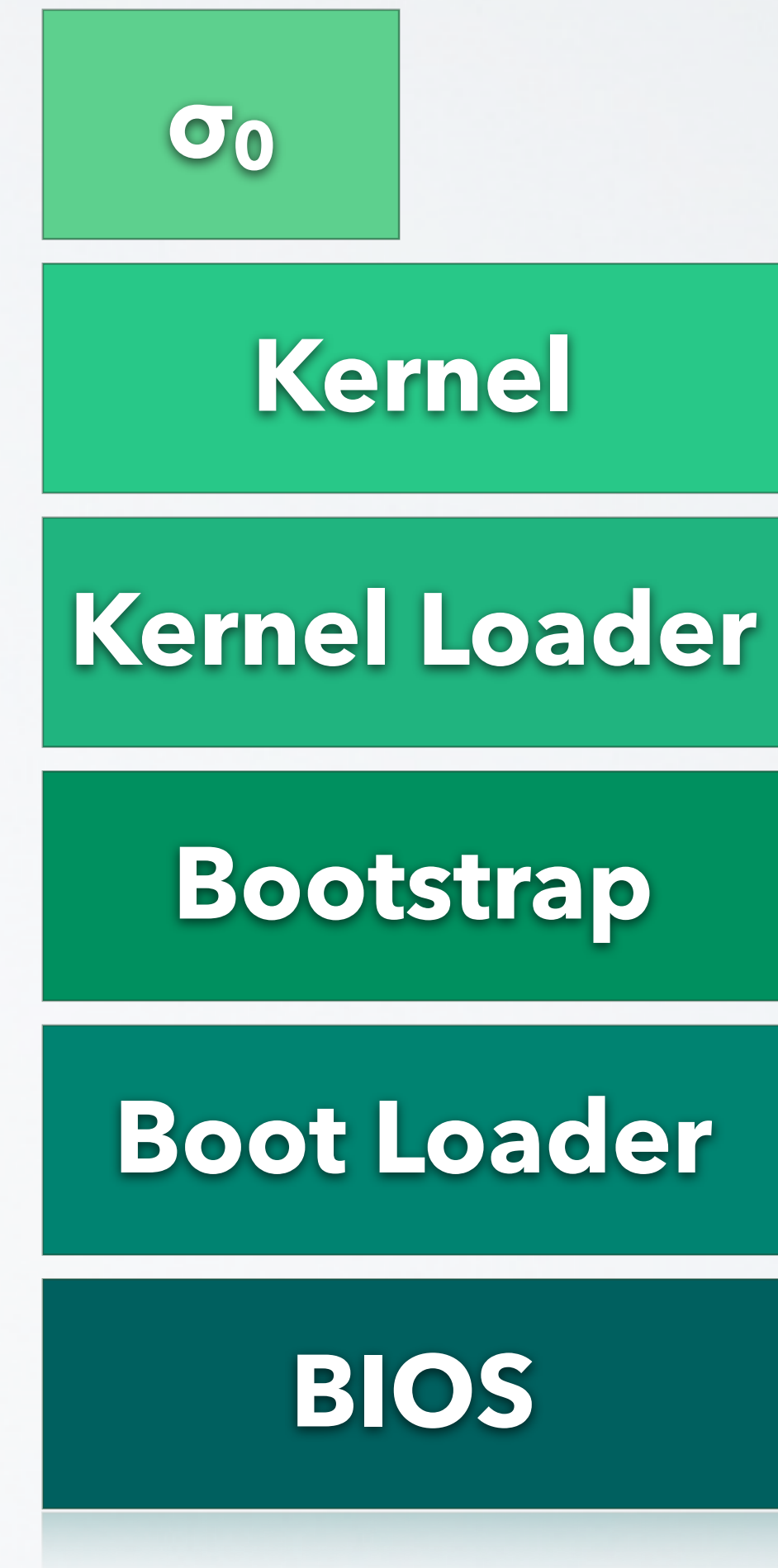
VIRTUAL MEMORY



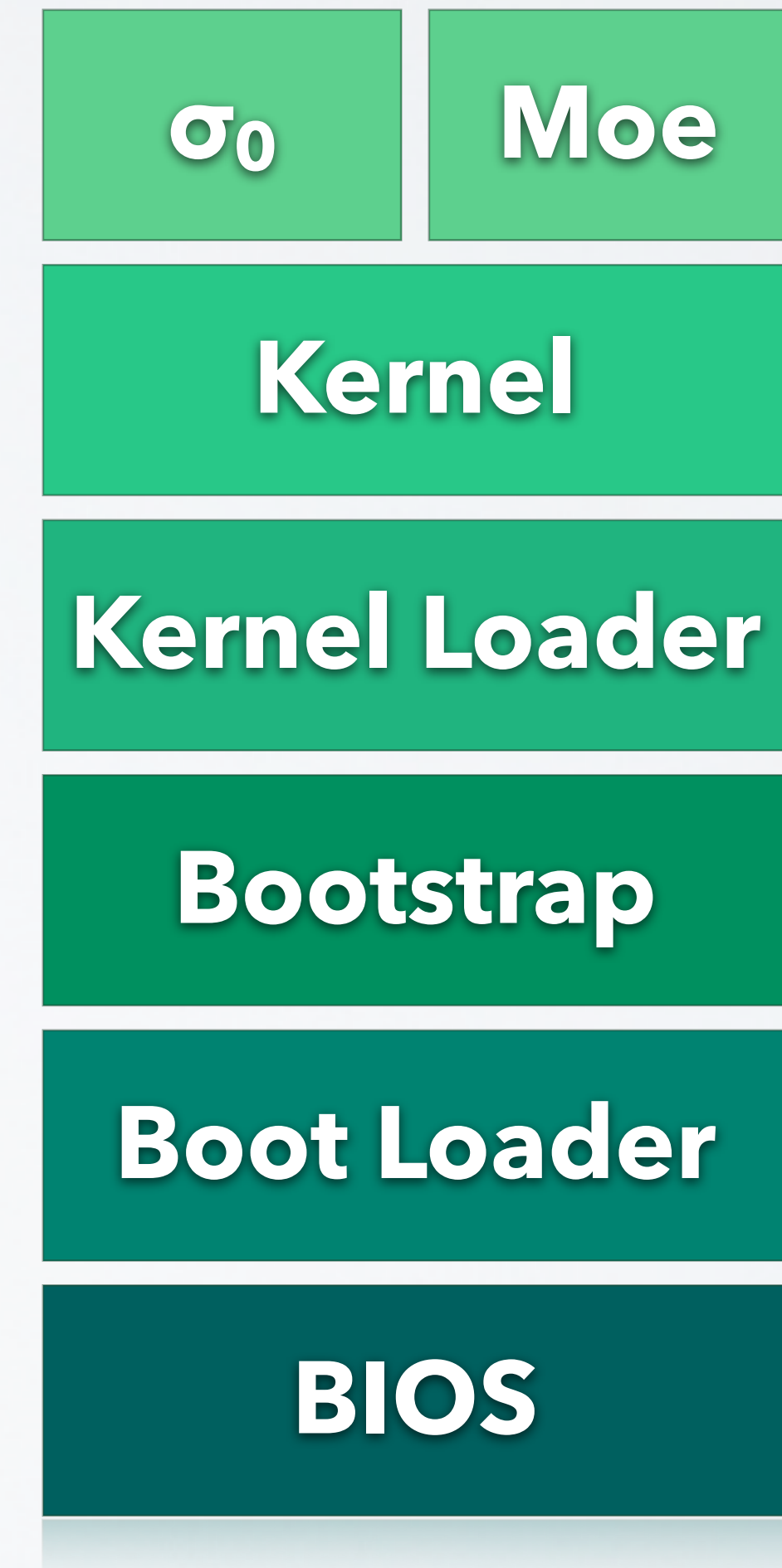
- sets up kernel structures
- sets up scheduling timer
- starts first pager
- starts first task
- starts scheduling
- scheduler hands control to userland for the first time



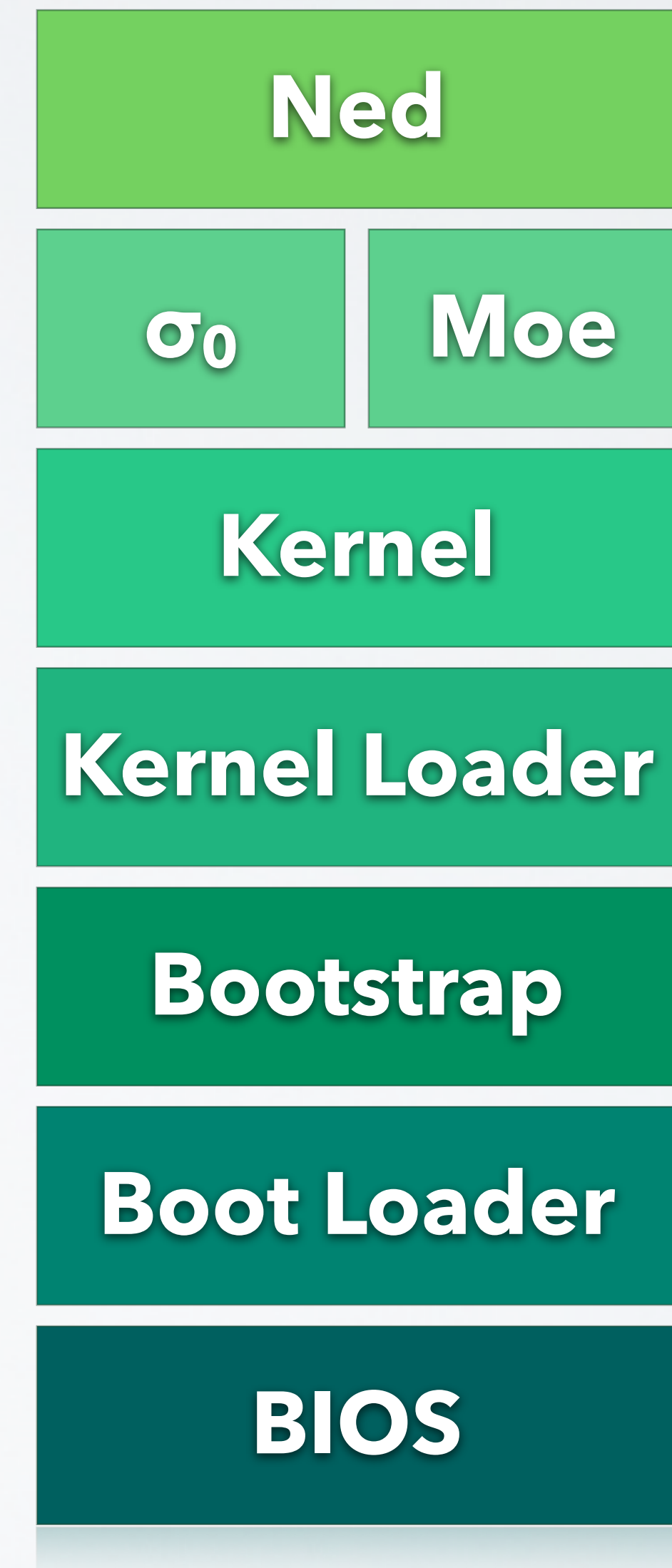
- is first pager in the system
- initially receives a 1:1 map-ping of physical memory
- ... and other platform-level resources (I/O ports)
- sigma0 is the root of the pager hierarchy
- pager for moe



- manages initial resources
 - namespace
 - memory
 - VESA framebuffer
- provides logging facility
- mini-filesystem for read-only access to boot-modules



- script-driven loader for further programs
- loads programs into memory and configures their initial capabilities
- startup-scripts written in Lua
- software can be loaded by retrieving binaries from disk or network
- ned injects common service code into every task



SETUP

- ▶ download the source tarball from
<https://os.inf.tu-dresden.de/Studium/KMB/WS2024/Exercise1.tar.bz2>
- ▶ unpack the tarball
 - ▶ it comes with a working directory
 - ▶ `cd` in there and have a look around

COMPILING THE SYSTEM

- ▶ initialize the environment with `make setup` in the top-level directory you unpacked
- ▶ run `make` within the toplevel directory

TEST-DRIVING QEMU

- ▶ create a bootable ISO image
 - ▶ the `iso` subdirectory is for the ISO's content
 - ▶ run `isocreator` from `src/14/tool/bin` on this directory
- ▶ your ISO will contain a minimal grub installation
- ▶ launch QEMU with the resulting ISO:
`qemu-system-x86_64 -m 512 -cdrom boot.iso`

BOOTING FIASCO

- ▶ copy some files to the ISO directory
 - ▶ `fiasco` from the Fiasco build directory
`obj/fiasco/amd64/`
 - ▶ `bootstrap` from
`obj/14/amd64/bin/amd64_gen/`
 - ▶ `sigma0, moe, 14re` and `ned` from
`obj/14/amd64/bin/amd64_gen/14f/`

BOOTING FIASCO

- ▶ edit `iso/boot/grub/menu.lst`:
 `title Getting Started`
 `kernel /bootstrap -serial`
 `modaddr 0x2000000`
 `module /fiasco`
 `module /sigma0`
 `module /moe`
 `module /l4re`
 `module /ned`
- ▶ rebuild the ISO and run `qemu`

PREPARING FOR HELLO

- ▶ create the file `hello.lua` in the `iso` directory with this content:

```
local L4 = require("L4");  
L4.default_loader:start({}, „rom/hello”);
```

- ▶ pass `ned` this new startup script
 - ▶ add this line to `menu.lst`:
`module /hello.lua`
 - ▶ pass `rom/hello.lua` as parameter to `moe`
 - ▶ load the future `hello` module in `menu.lst`

EXERCISE 1: HELLO WORLD

- ▶ create a directory for your hello-project
- ▶ create a Makefile with the following content:

```
PKGDIR      ?= .  
L4DIR       ?= absolute path to L4 source tree  
OBJ_BASE    = absolute path to L4 build tree  
TARGET      = hello  
SRC_C       = hello.c  
include $(L4DIR)/mk/prog.mk
```
- ▶ fill in `hello.c` and compile with `make` and run in `qemu`

EXERCISE 2: ACKERMANN

- ▶ write a program that spawns six threads
 - ▶ you can use pthreads in our system
 - ▶ add the line `REQUIRES_LIBS = libpthread` to `Makefile`
- ▶ each thread should calculate one value $a(3,0..5)$ of the Ackermann function:
 - ▶ $a(0,m) = m+1$
 - ▶ $a(n,0) = a(n-1,1)$
 - ▶ $a(n,m) = a(n-1,a(n,m-1))$