

Linux Kernel Walkthroughs

Booting



oclug.on.ca



TheCodeFactory.ca



Future topics

- System calls
- VFS
- Scheduling
- Memory management

<http://oclug.on.ca/KernelWalk>

Booting

Outline

- PC hardware overview
- software componets
- boot overview
- follow the code

PC hardware basics

PC hardware basics

- CPU
- memory
- disk controller
- HDD

PC hardware basics

- CPU
- memory
- disk controller
- HDD

A yellow rounded rectangle with a red border containing the text "CPU".

CPU

A yellow rounded rectangle with a red border containing the text "RAM".

RAM

A yellow rounded rectangle with a red border containing the text "ATA".

ATA

A yellow cylinder with a red outline representing a hard disk drive, with the text "HDD" on its top surface.

HDD

PC hardware basics

- CPU
- memory
- disk controller
- HDD
- video

CPU

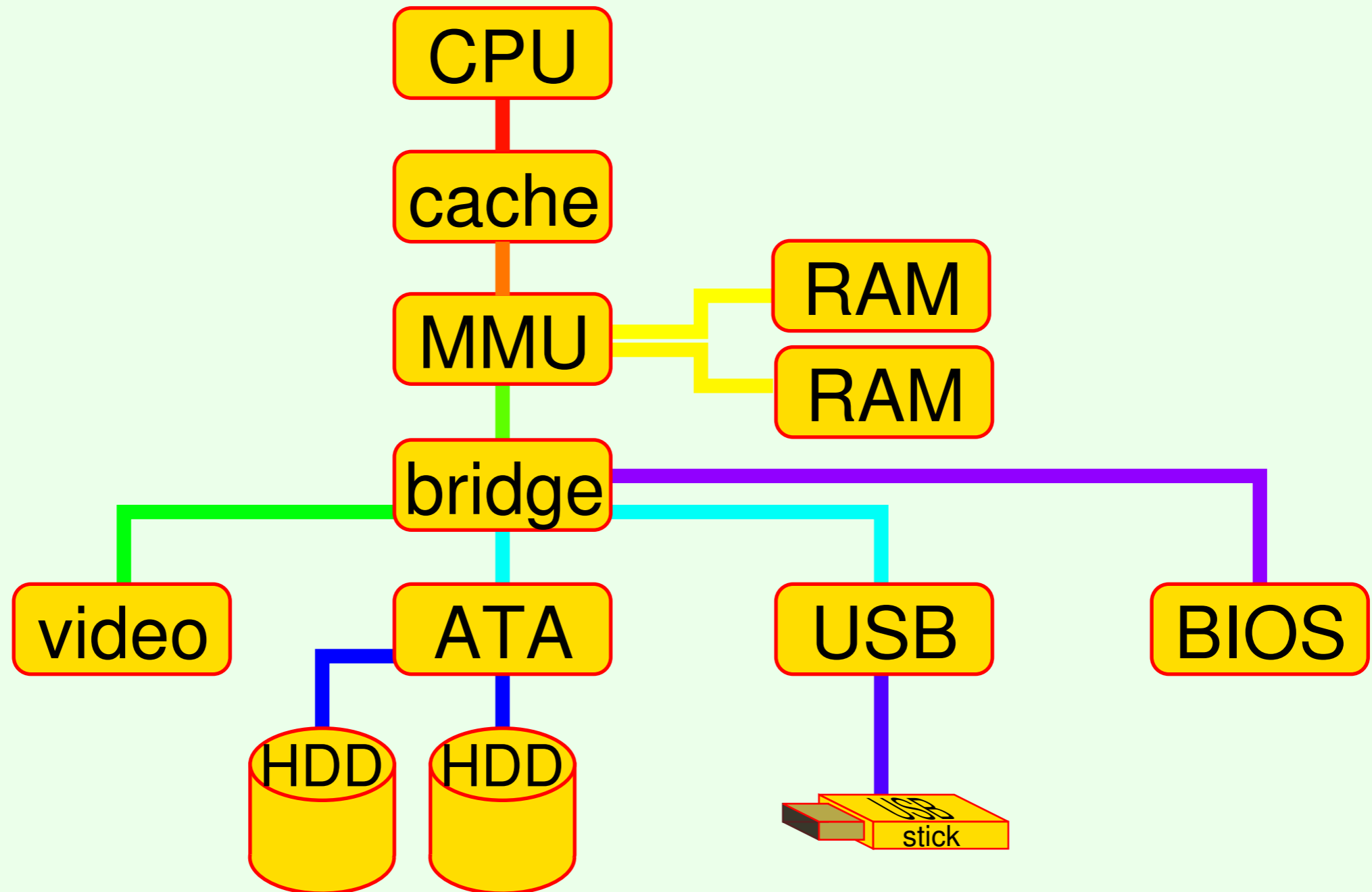
RAM

video

ATA

HDD

PC hardware basics



PC hardware basics

CPU

- lots of variants
 - Intel Core2 Duo
 - AMD Opteron
 - VIA Eden
- backward compatible to 1970s
 - emulates 8086
 - 16bit
 - 1M memory

PC hardware basics

CPU

"real mode"

- all BIOS services
- 1M addressable

"protected mode"

- few BIOS services
- 4G memory
- virtual memory

PC hardware basics

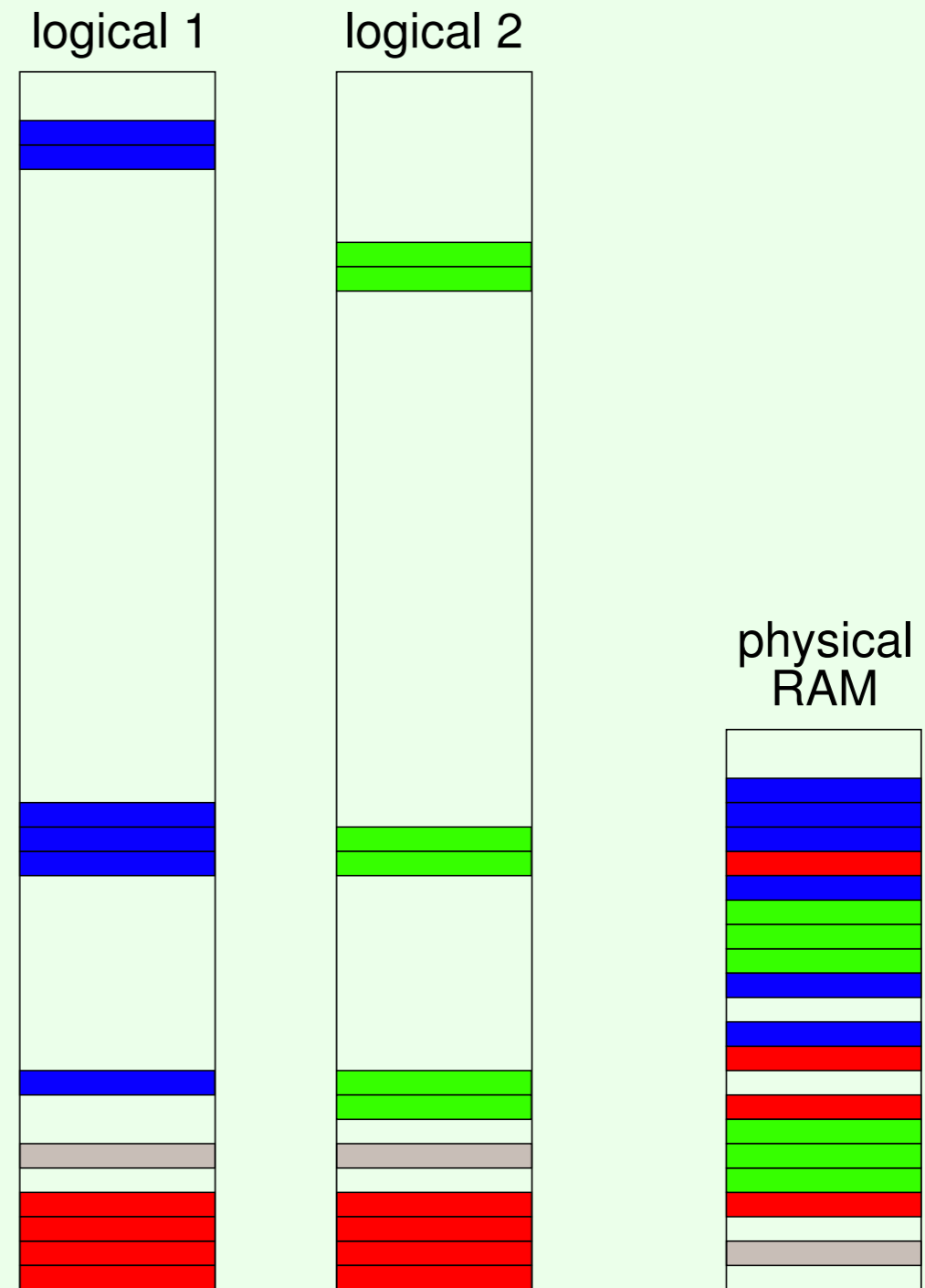
MMU

- memory management unit
- "north bridge" or in CPU
- paging
- virtual memory

PC hardware basics

MMU

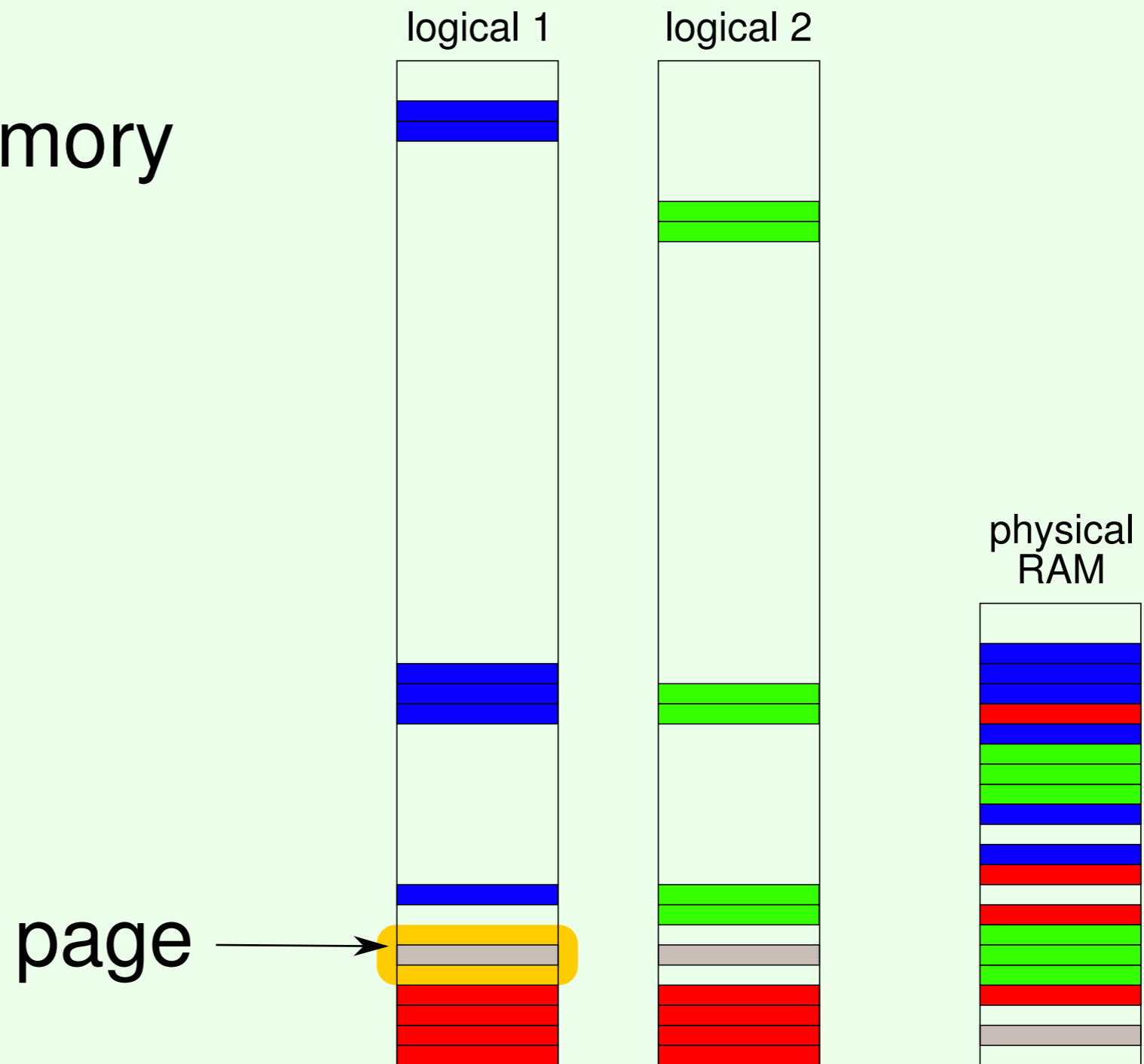
virtual memory



PC hardware basics

MMU

virtual memory

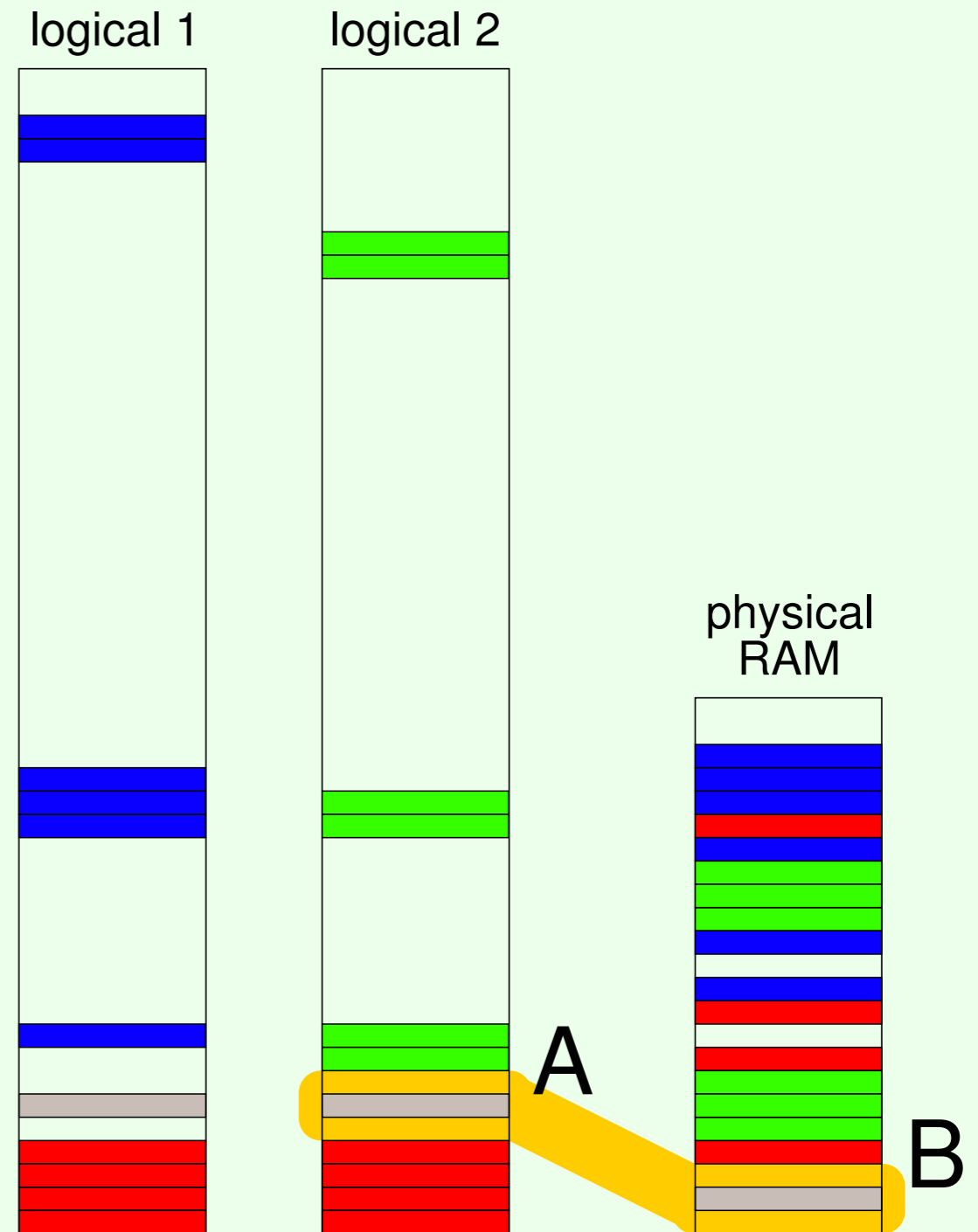


PC hardware basics

MMU

virtual memory

- page mappings



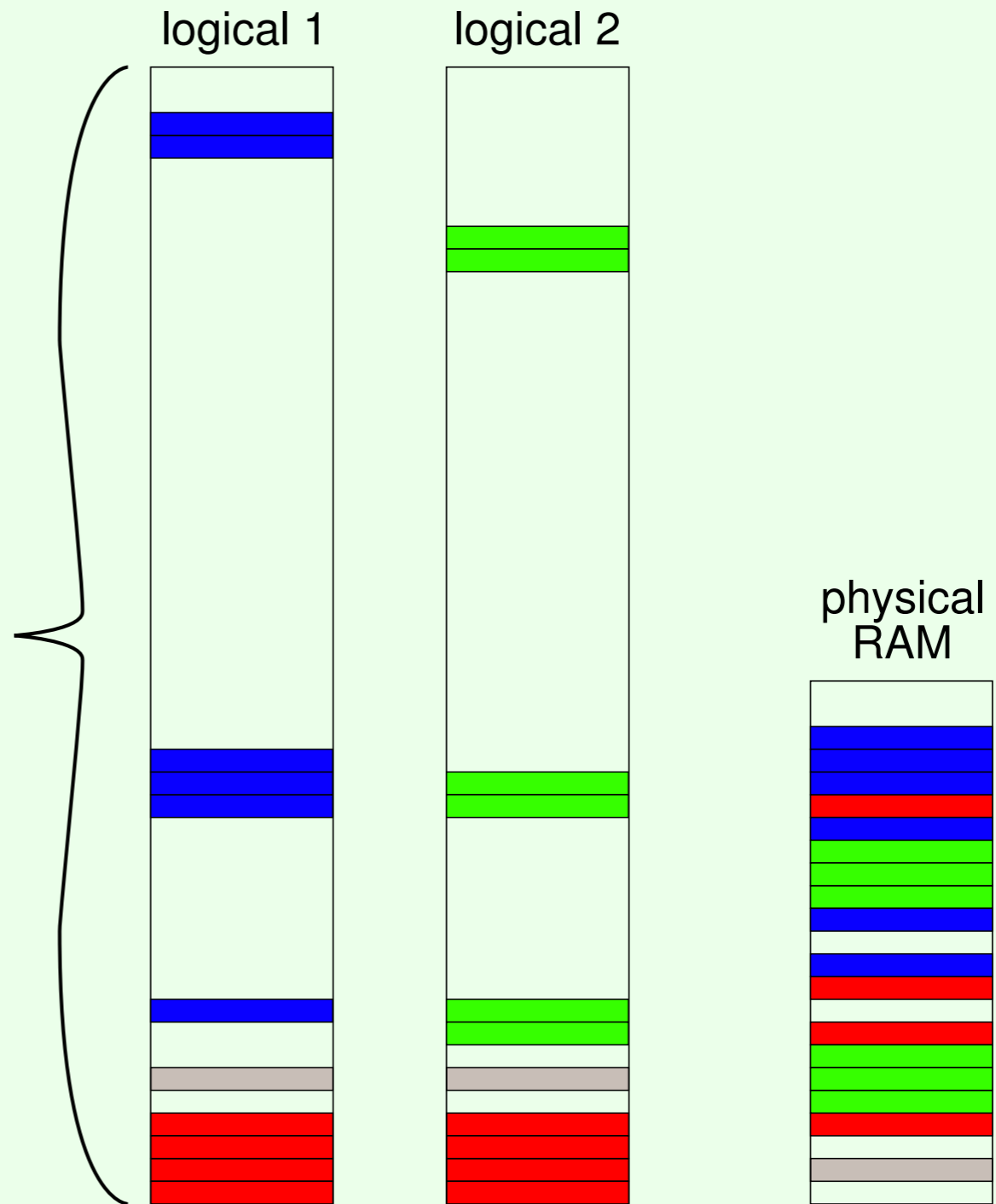
PC hardware basics

MMU

virtual memory

- addressing

4GB*

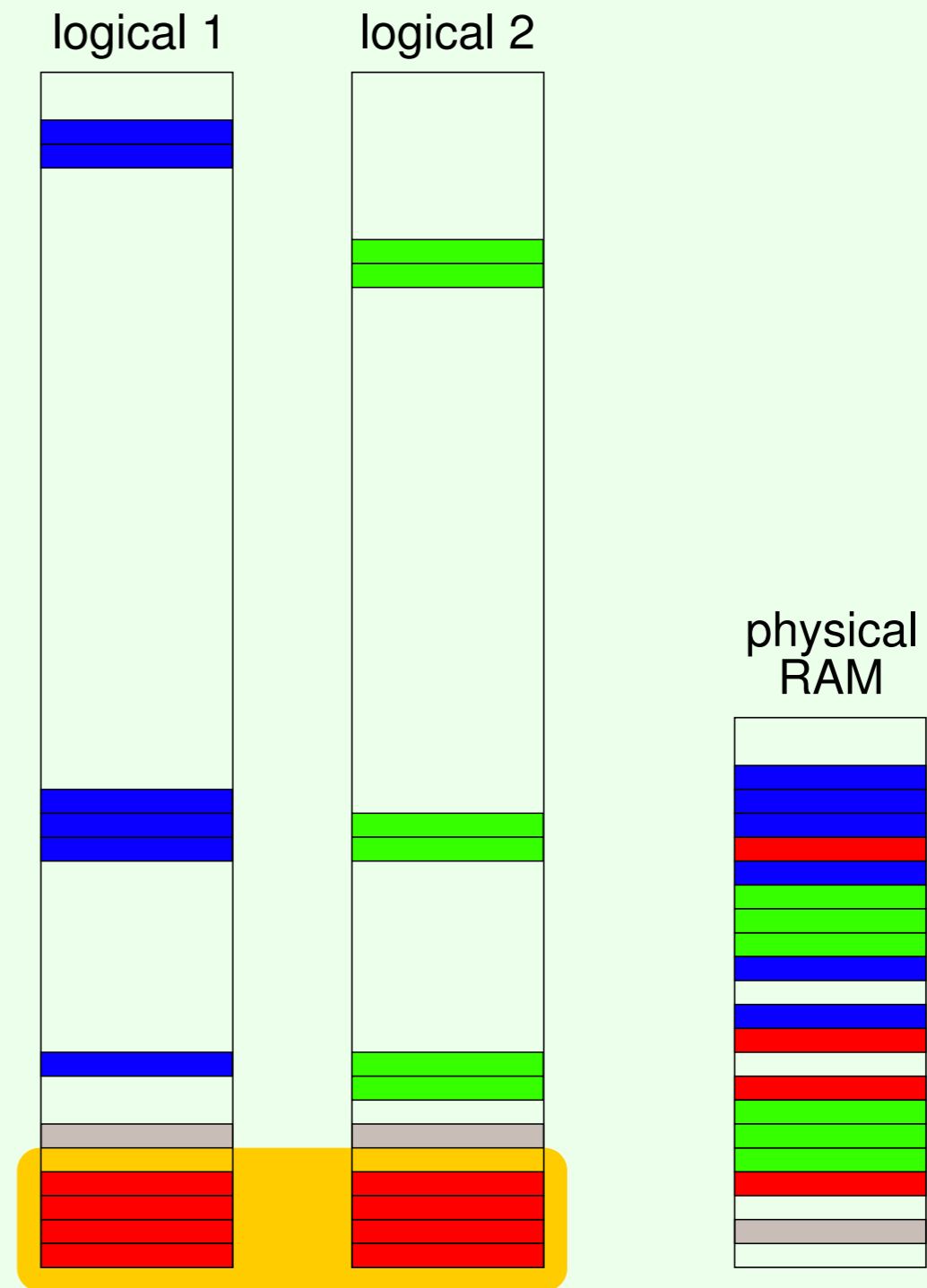


PC hardware basics

MMU

virtual memory

- shared pages



PC hardware basics

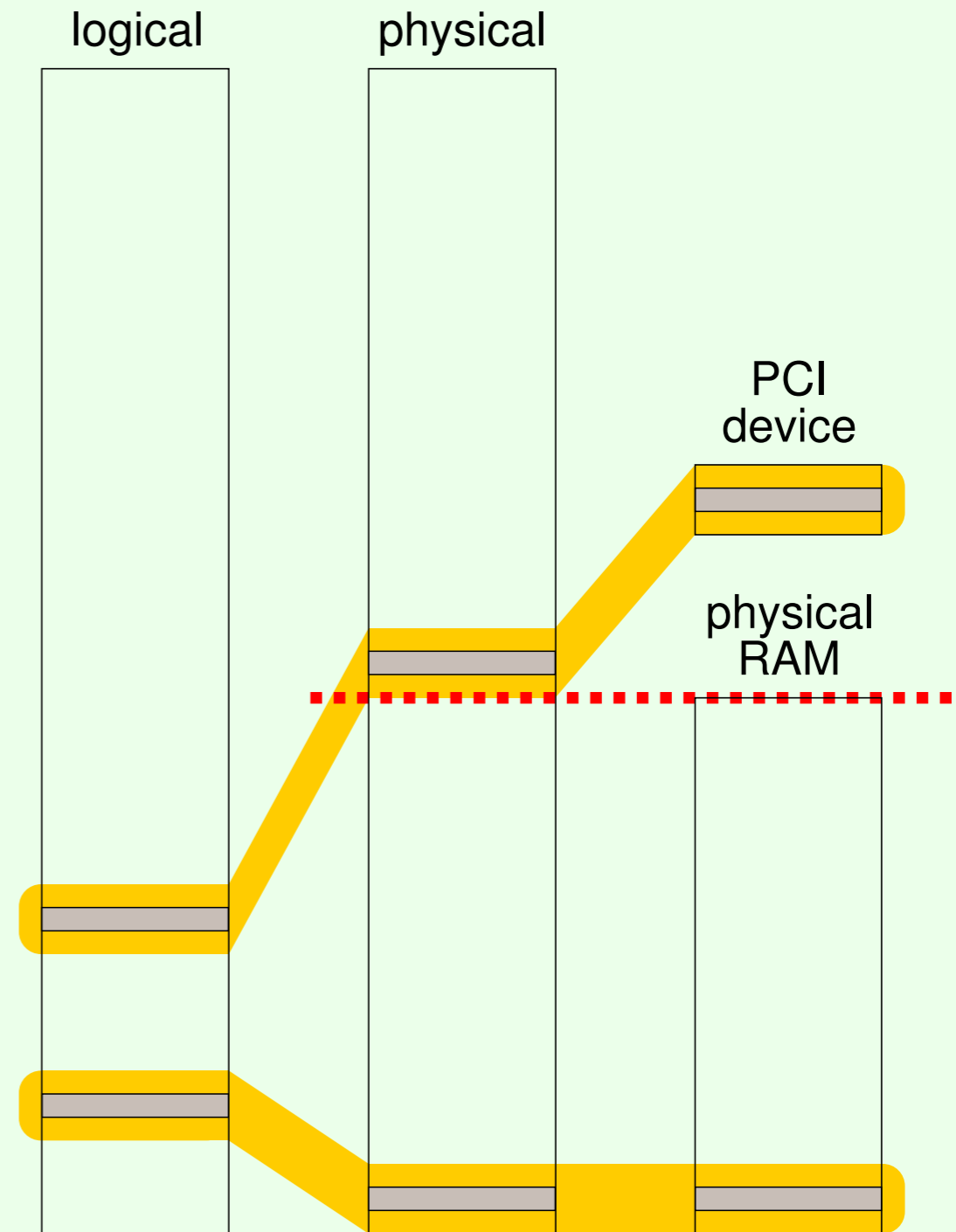
bridge

- "south bridge"
- local bus
- PCI
- peripheral controllers (ATA, USB, etc)
- memory addressable h/w devices
- routes interrupts

PC hardware basics

bridge

memory mapped IO



... for more on MM topics tune in next time.

Software

Software

- firmware
- BIOS
- MBR
- boot loader
- kernel

Software Firmware

Software

Firmware

- CPU
- microcode
- initialize processor(s)

Software BIOS

Software BIOS

- real mode
- 16bit registers
- 1M addressable memory
- limited HW access
- slow

Software

BIOS

Initialization

- CPU init
- cache init
- interconnect

- HW enumeration
- reset / setup

- POST

Software

BIOS

Services

- memory layout
- disk
- PCI
- APM & ACPI
- graphics

Software

BIOS

Disk services

AH = operation

DL = drive, first HDD is 0x80

...

INT 0x13

Software

Master Boot Record

Software

Master Boot Record

- 440 bytes of code
- first disk block
- bootloader unique
- loads rest of the bootloader

Software

Master Boot Record

- aka "1st stage bootloader"

Software Bootloader

Software Bootloader

- BIOS calls
- loads kernel and initrd into RAM
- slow

Software

Bootloader

- BIOS services only see 1M of memory
- typical kernel is ~2M
- typical initrd is ~5M

Software Bootloader

- BIOS services only see 1M of memory
- typical kernel is ~2M
- typical initrd is ~5M

... that's a problem!

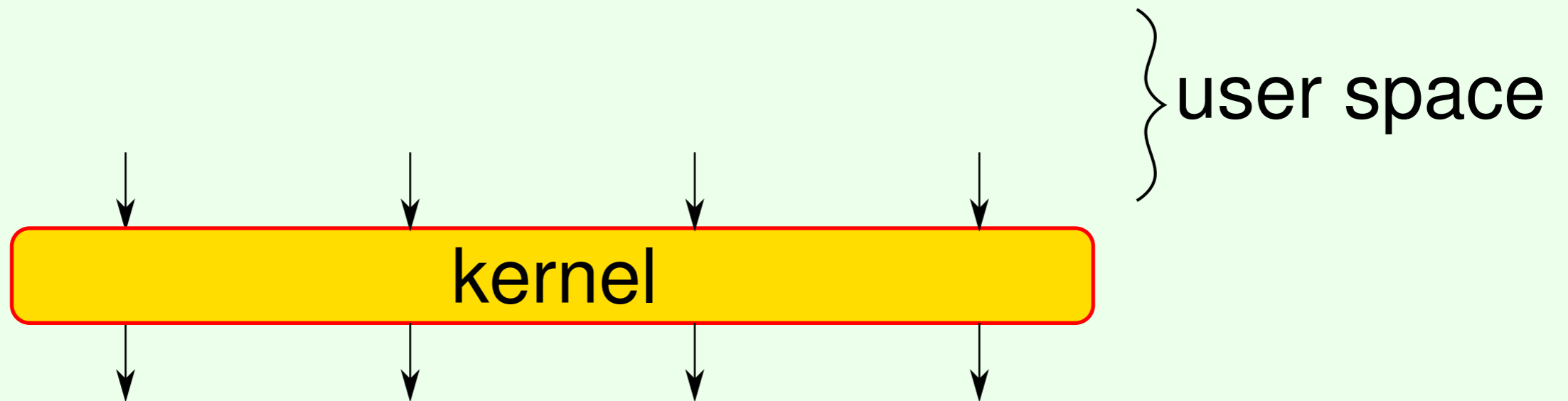
Software Bootloader

"unreal" mode

- switch to protected mode
- enable flat 32bit memory
- return to real mode

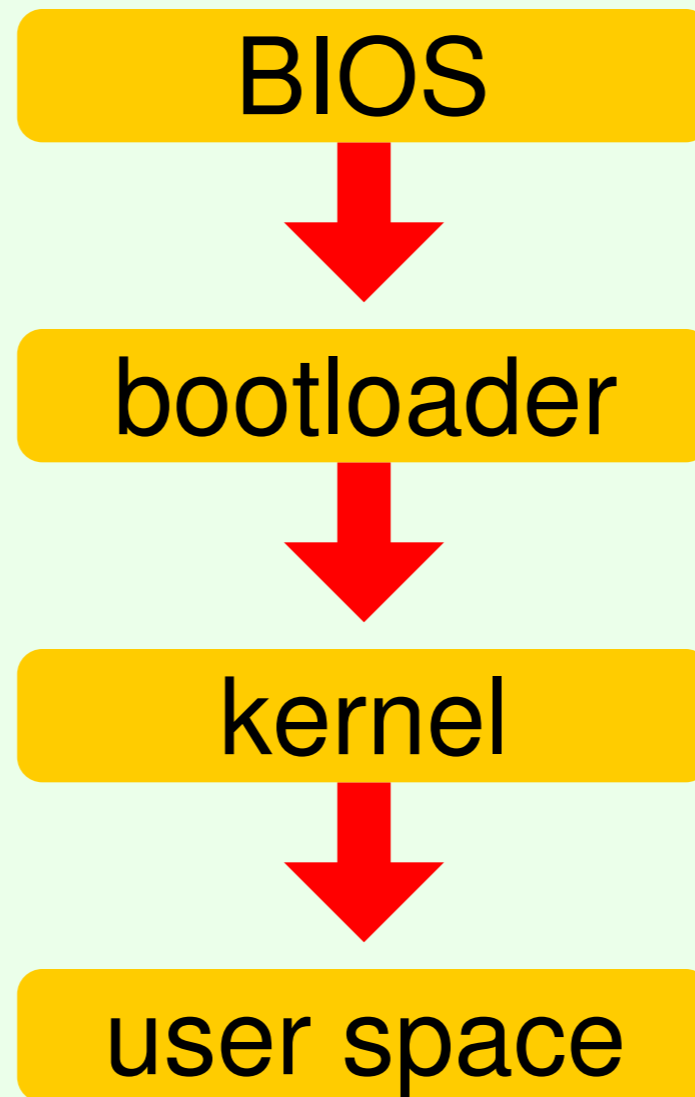
Software Kernel

Software Kernel



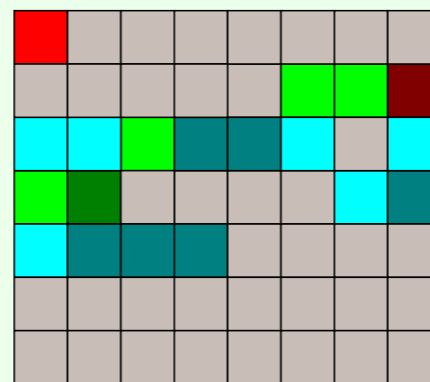
Booting

Booting



Booting On disk...

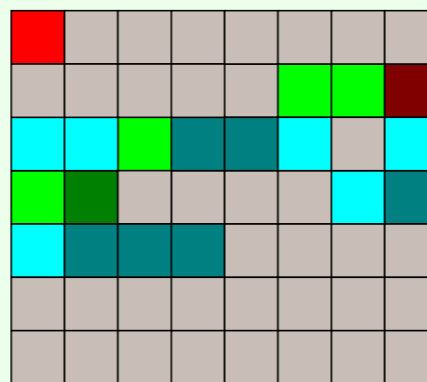
HDD




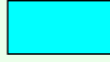



Booting On disk...

1 block ■ 512 bytes

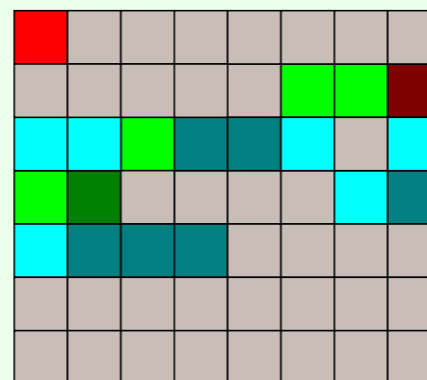
HDD



Booting On disk...

stage1   kernel
stage2   initrd
config 

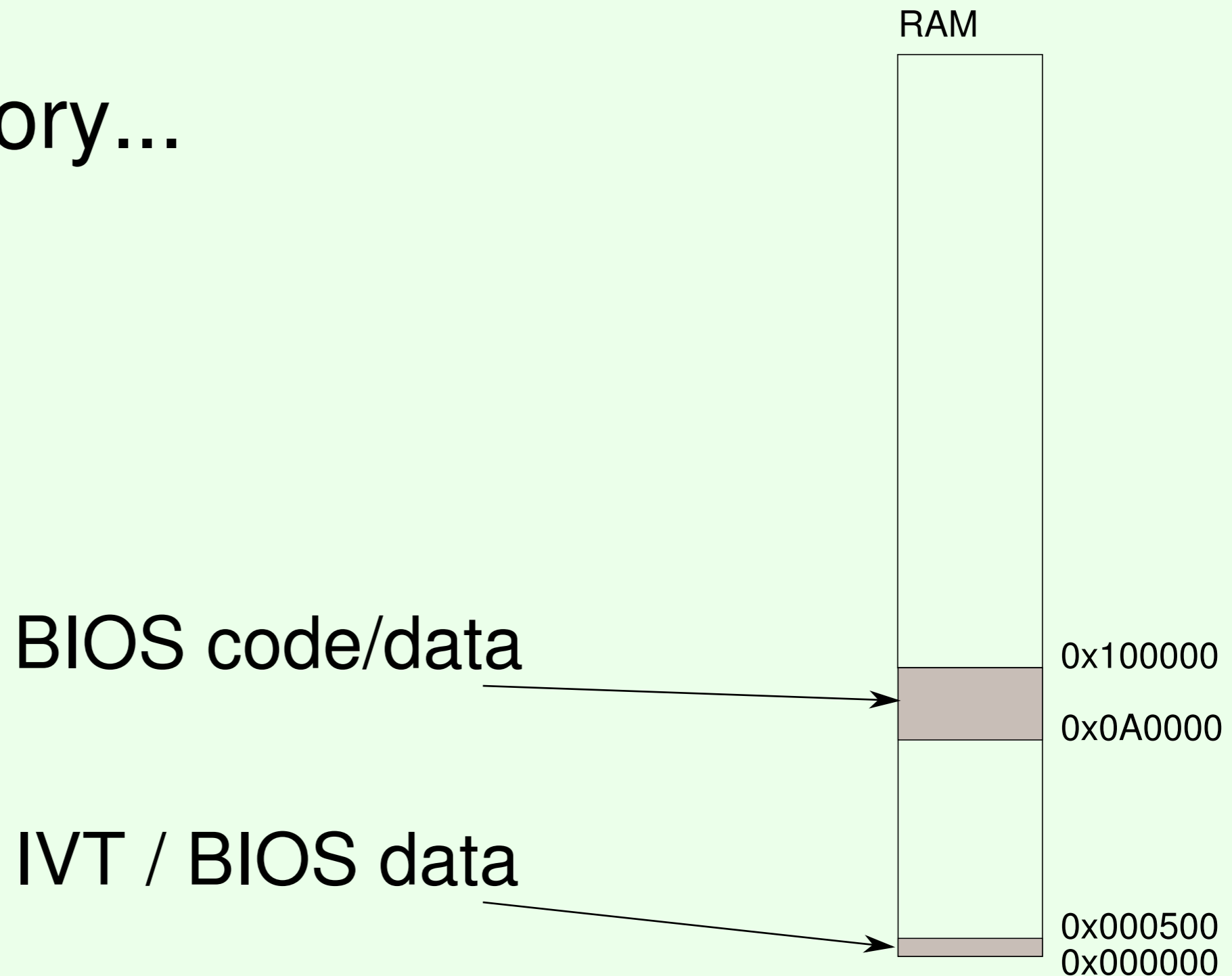
HDD



Booting BIOS

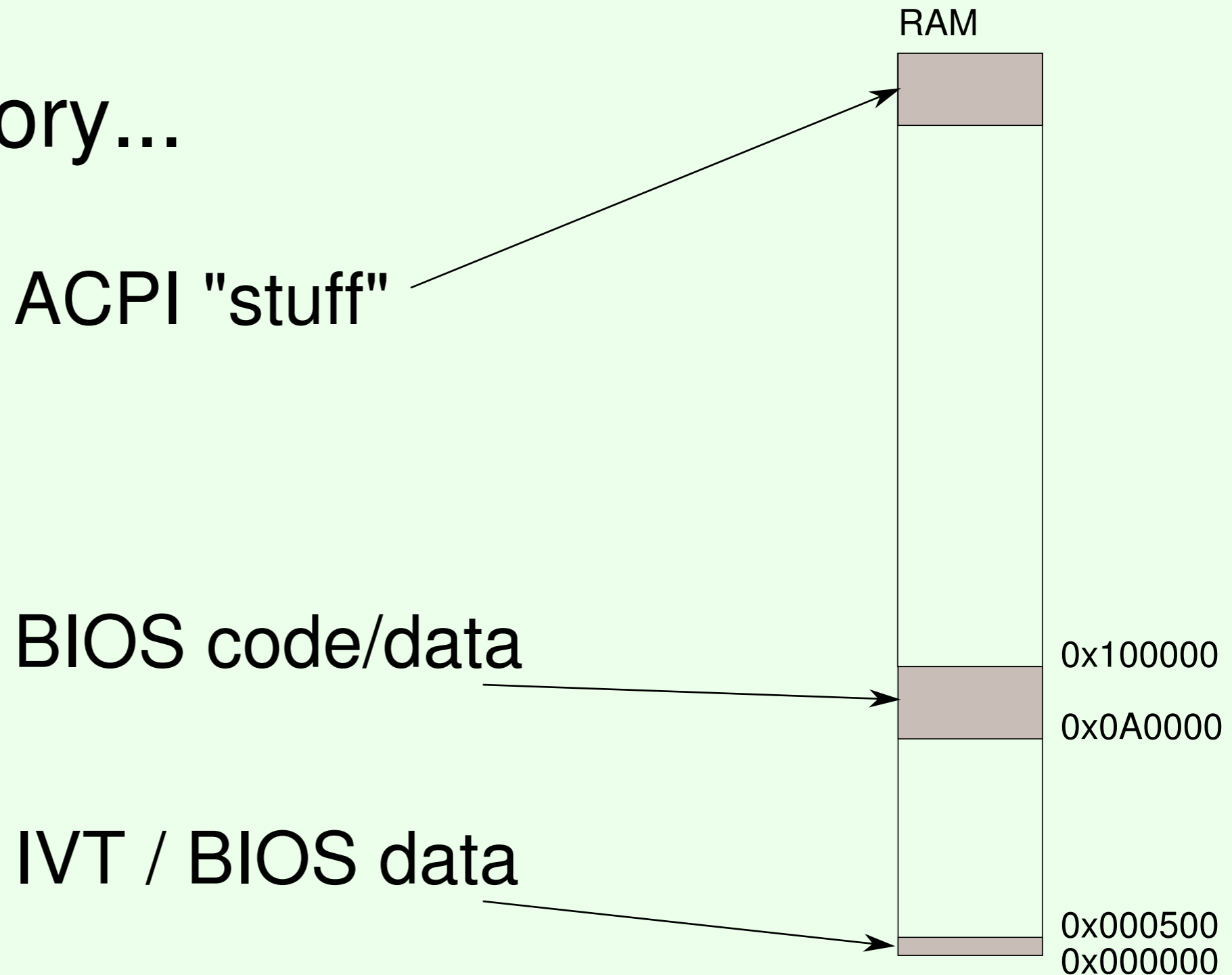
Booting BIOS

In memory...



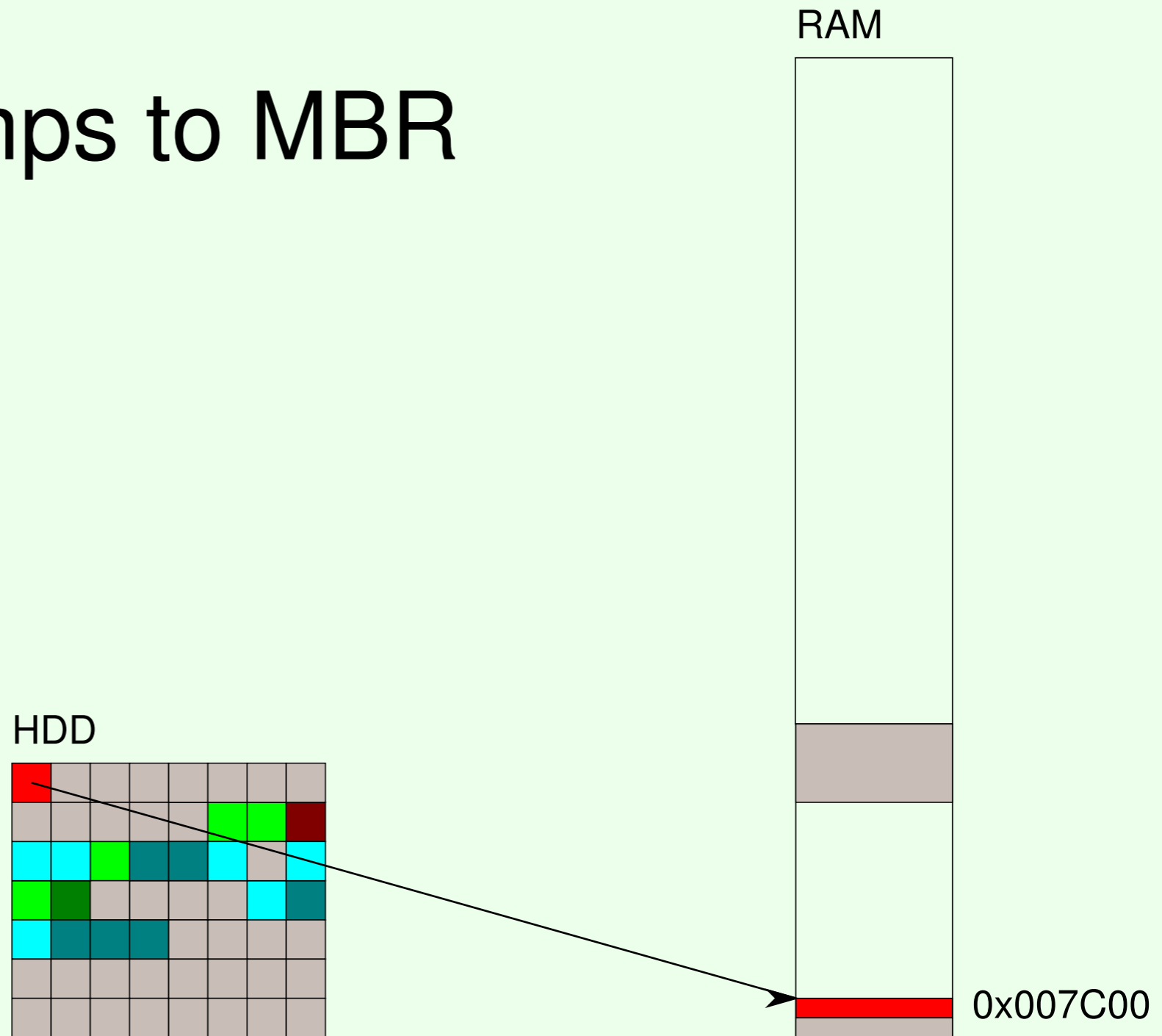
Booting BIOS

In memory...



Booting BIOS

Loads & jumps to MBR



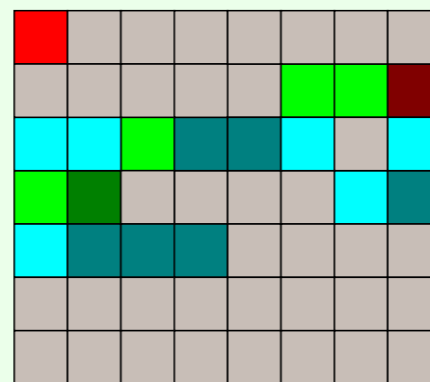
Booting

Bootloader

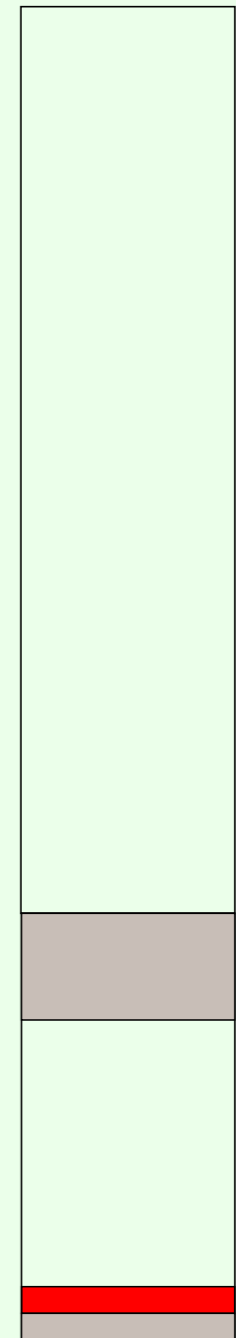
Booting Bootloader

MBR loads "step2 block"

HDD



RAM

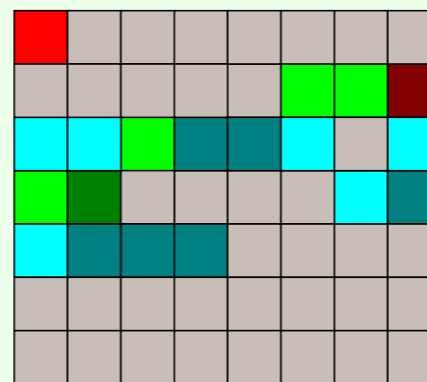


Booting Bootloader

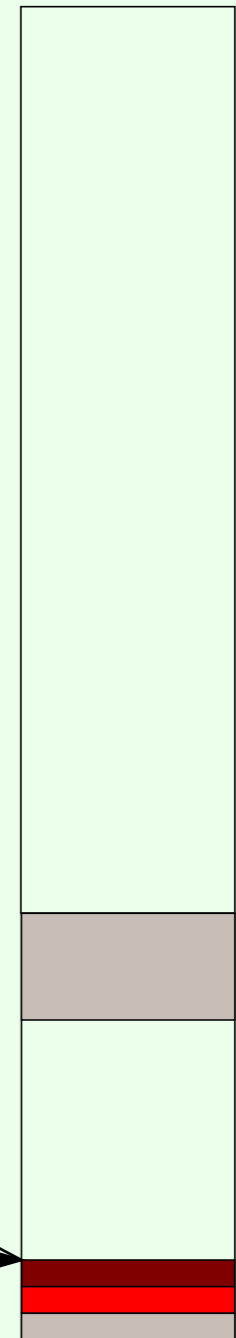
MBR loads "step2 block"

```
step2:                # "disk address packet"  
    .word 16, 1  
    .word 0x7e00, 0    # destination  
    .qword 0x...      # LBA disk location  
  
mov    $0x42, %ah     # extended read  
mov    $0x80, %dl     # drive c:  
lea    step2, %si     # descriptor in DS:SI  
int    $0x13
```

HDD



RAM

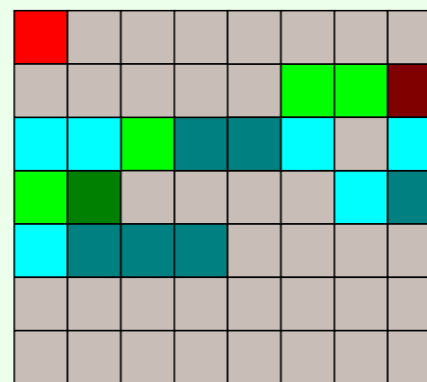


Booting Bootloader

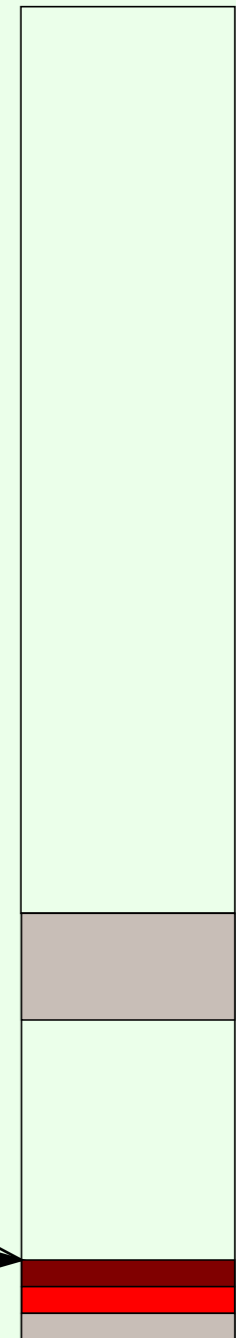
MBR loads "step2 block"

```
step2:                # "disk address packet"  
    .word 16, 1  
    .word 0x7e00, 0    # destination  
    .qword 0x...      # LBA disk location  
  
mov    $0x42, %ah     # extended read  
mov    $0x80, %dl     # drive c:  
lea    step2, %si    # descriptor in DS:SI  
int    $0x13
```

HDD



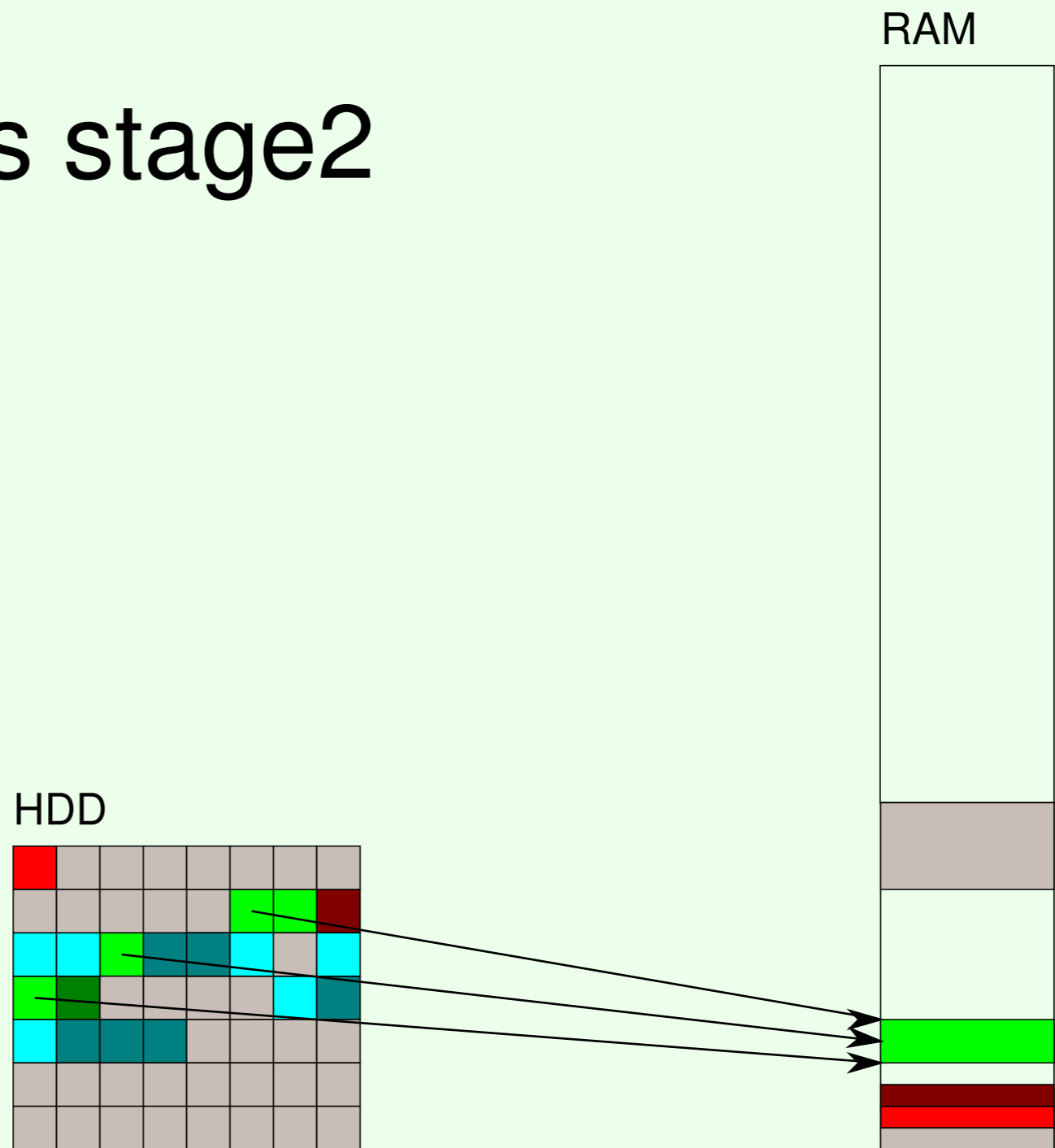
RAM



Booting

Bootloader

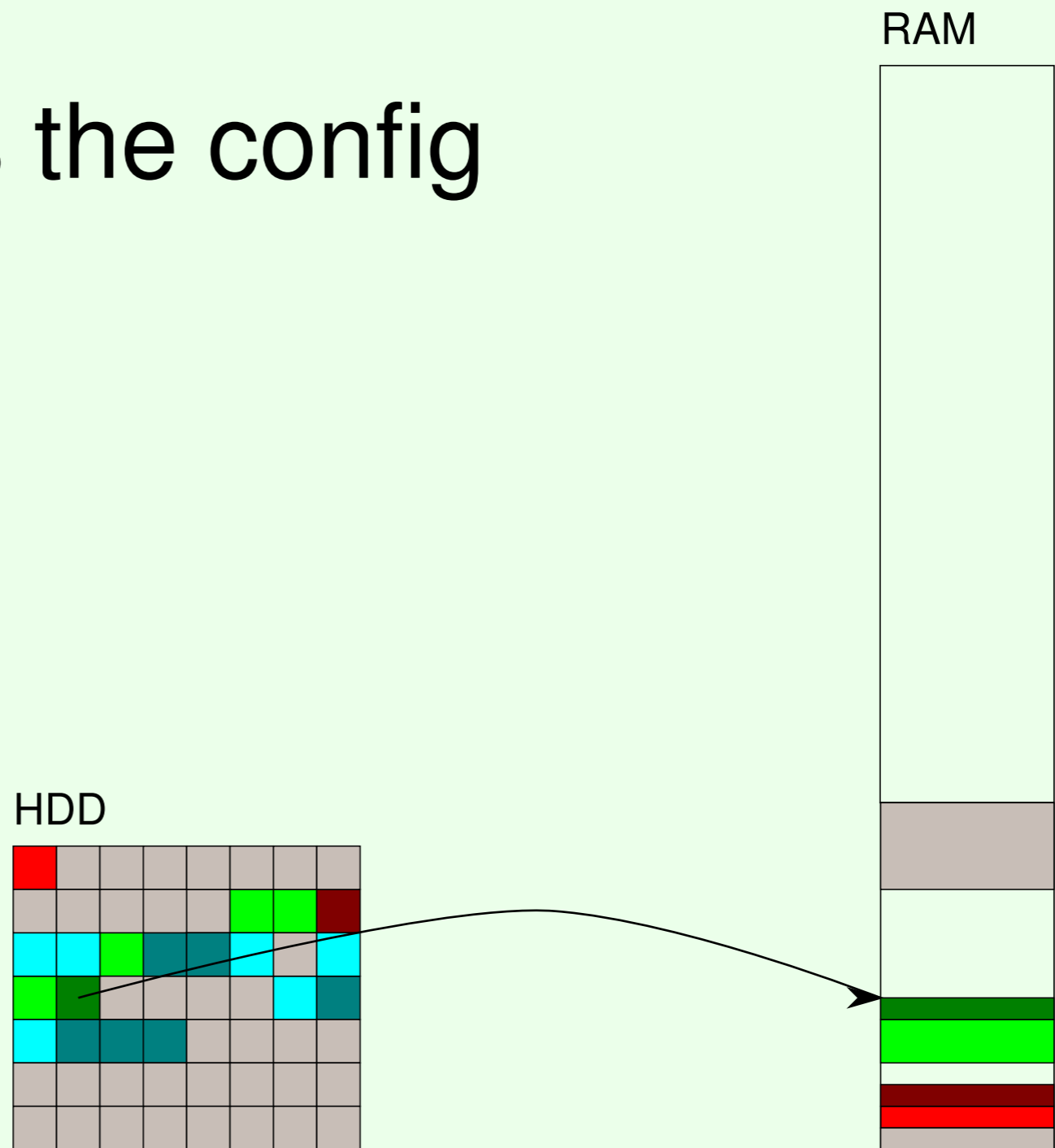
stage1 loads stage2



Booting

Bootloader

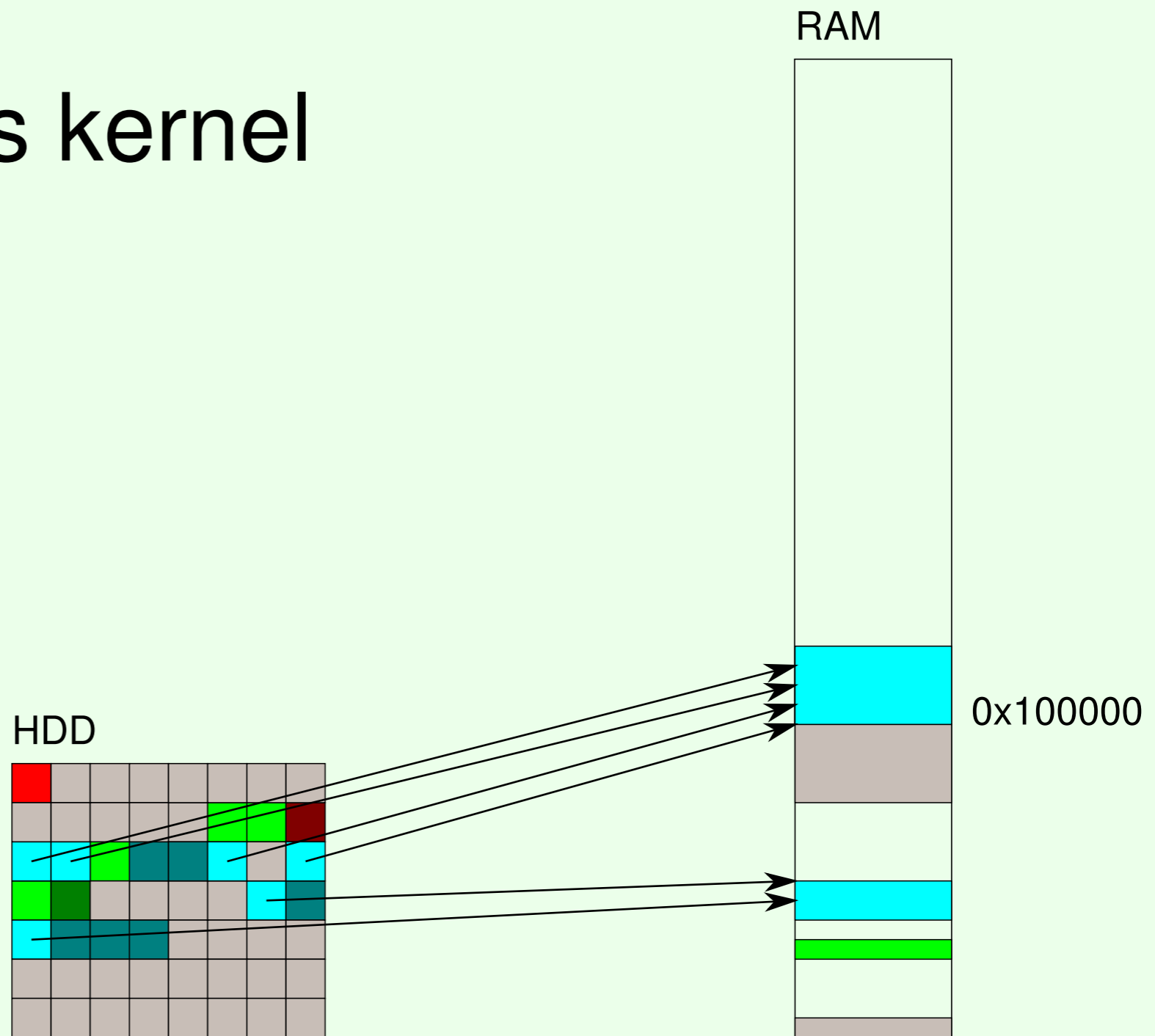
stage2 finds the config



Booting

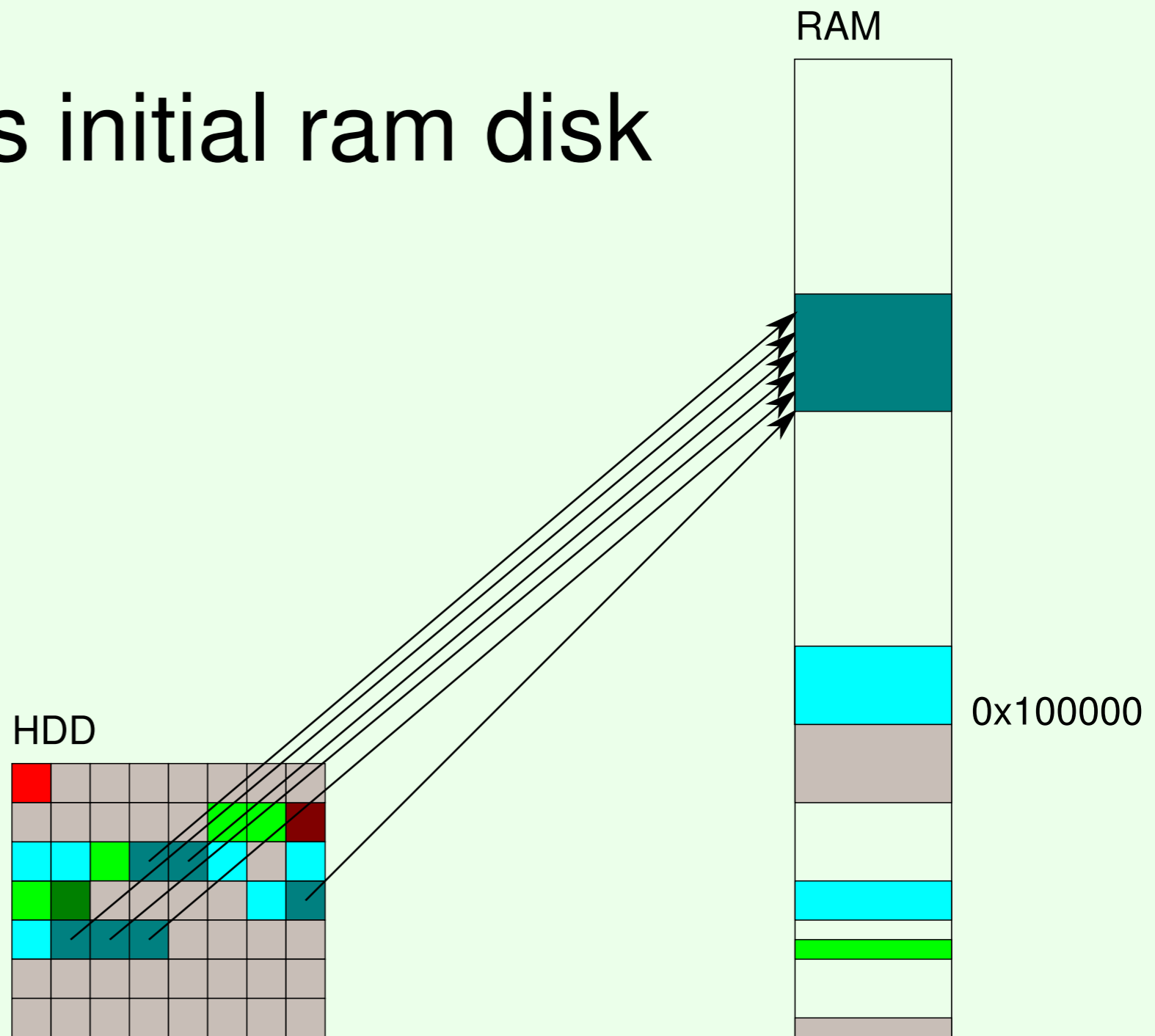
Bootloader

stage2 loads kernel



Booting Bootloader

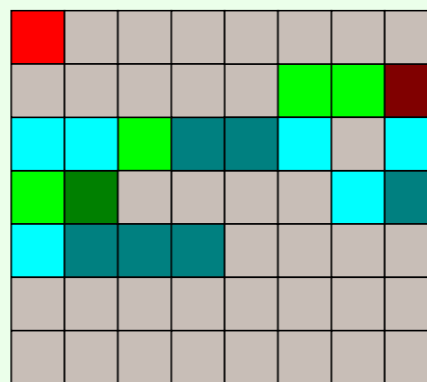
stage2 loads initial ram disk



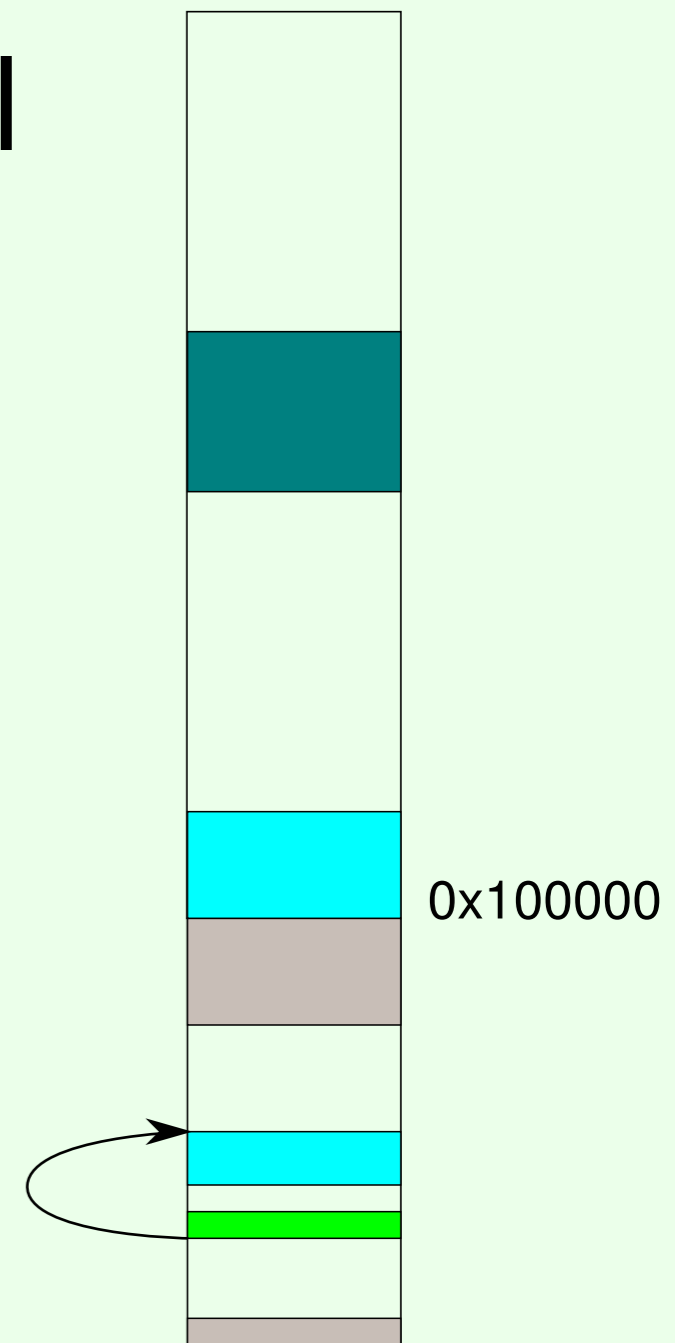
Booting Bootloader

stage2 jumps into the kernel

HDD



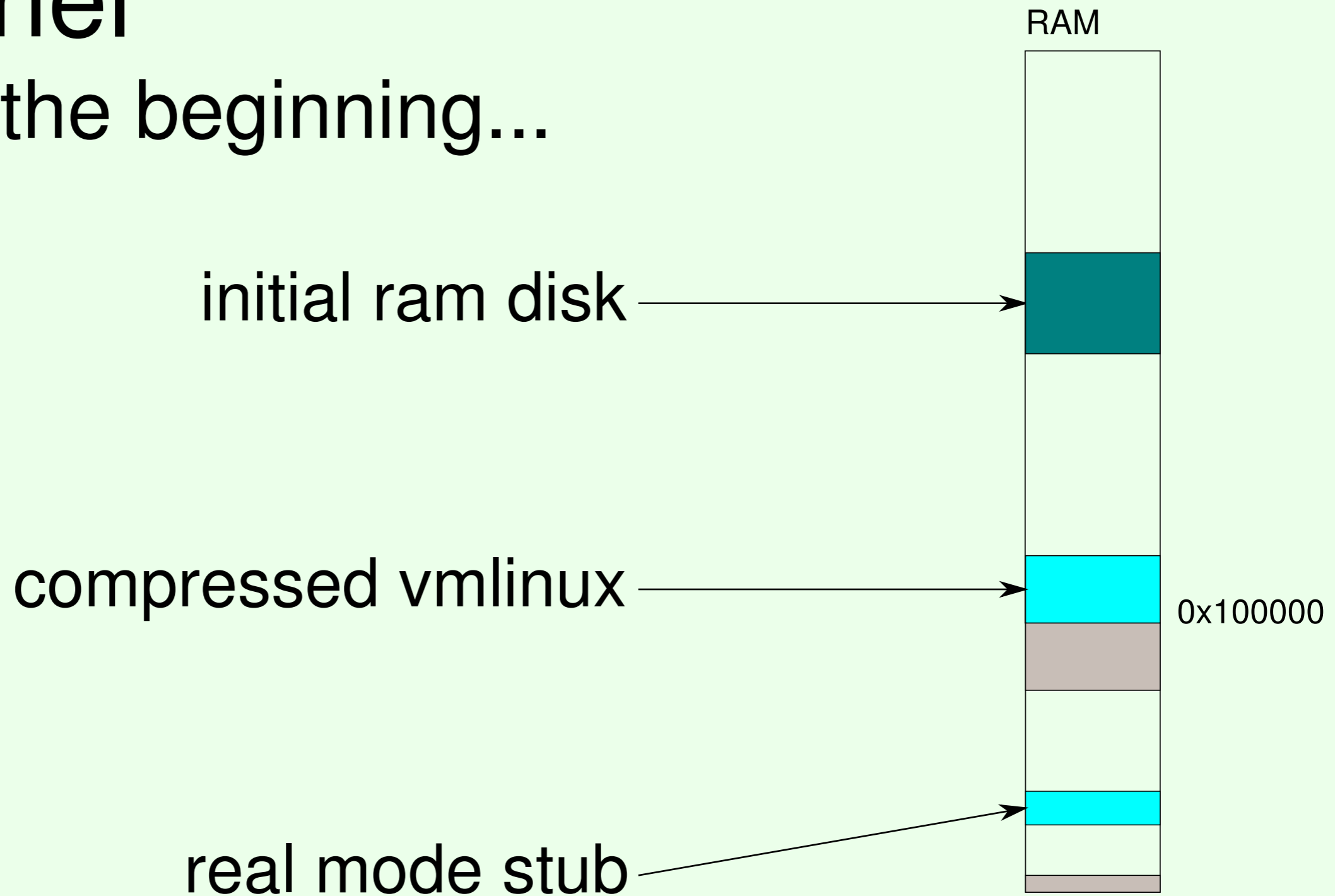
RAM



Booting Kernel

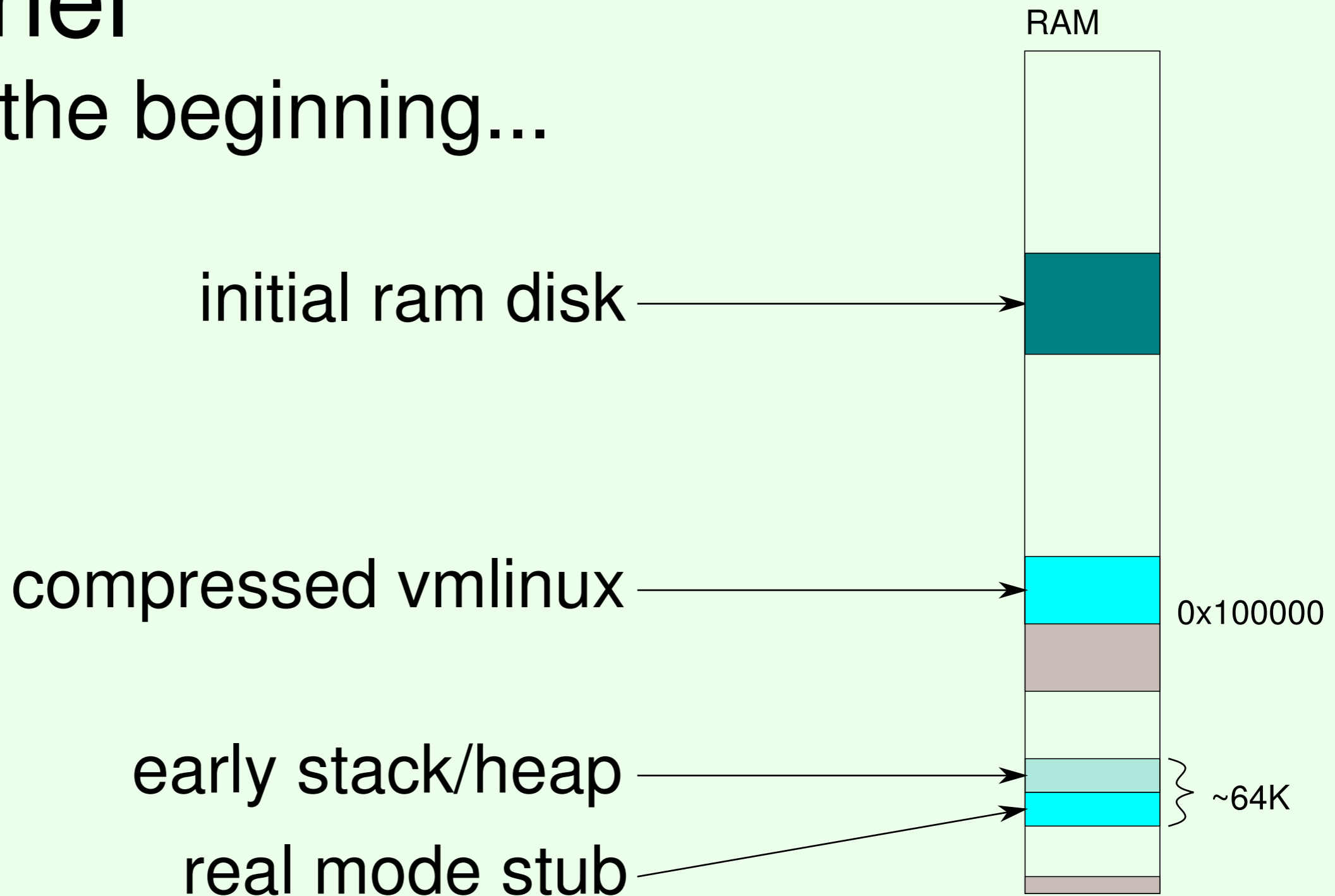
Booting Kernel

In the beginning...



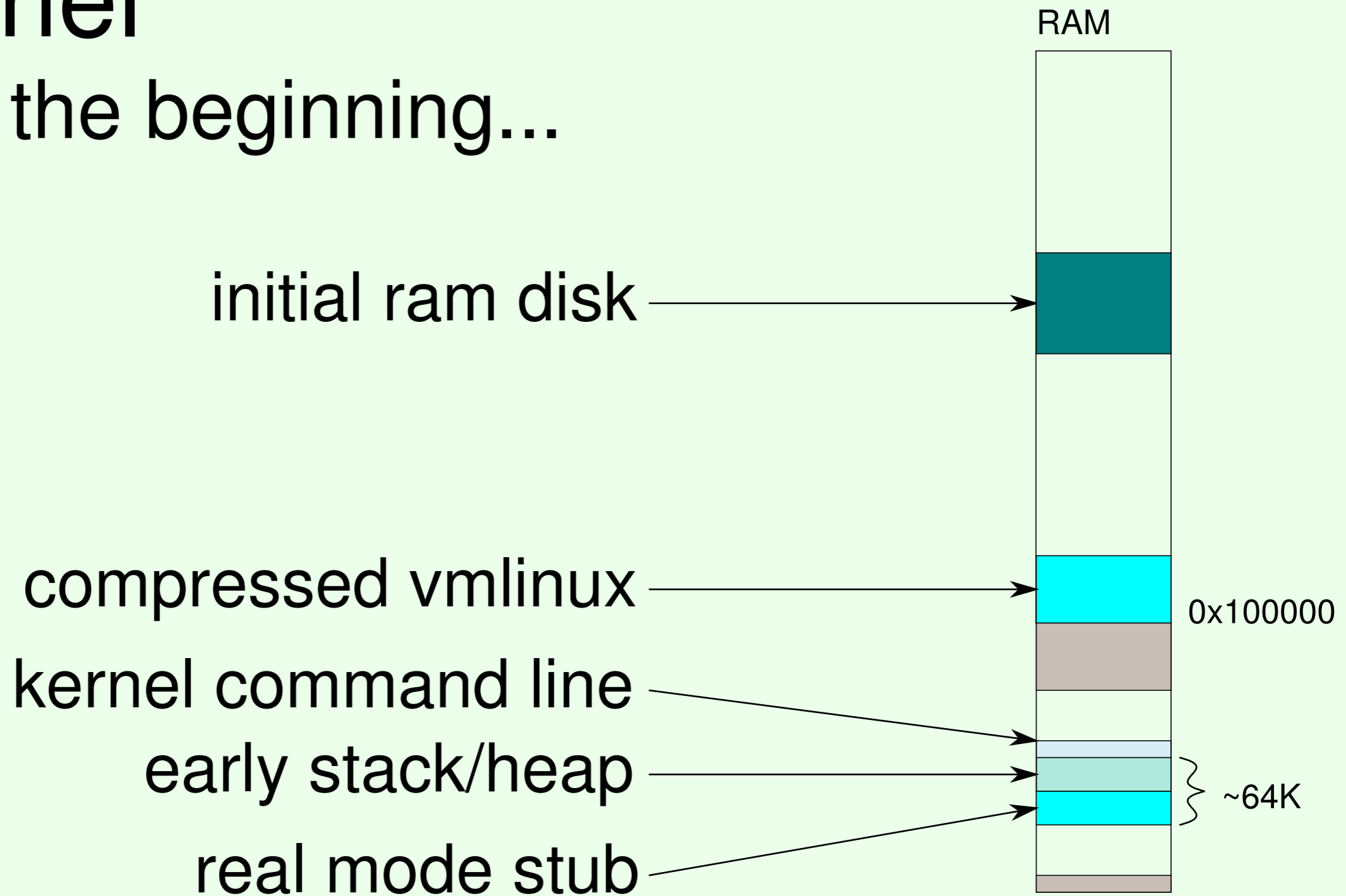
Booting Kernel

In the beginning...



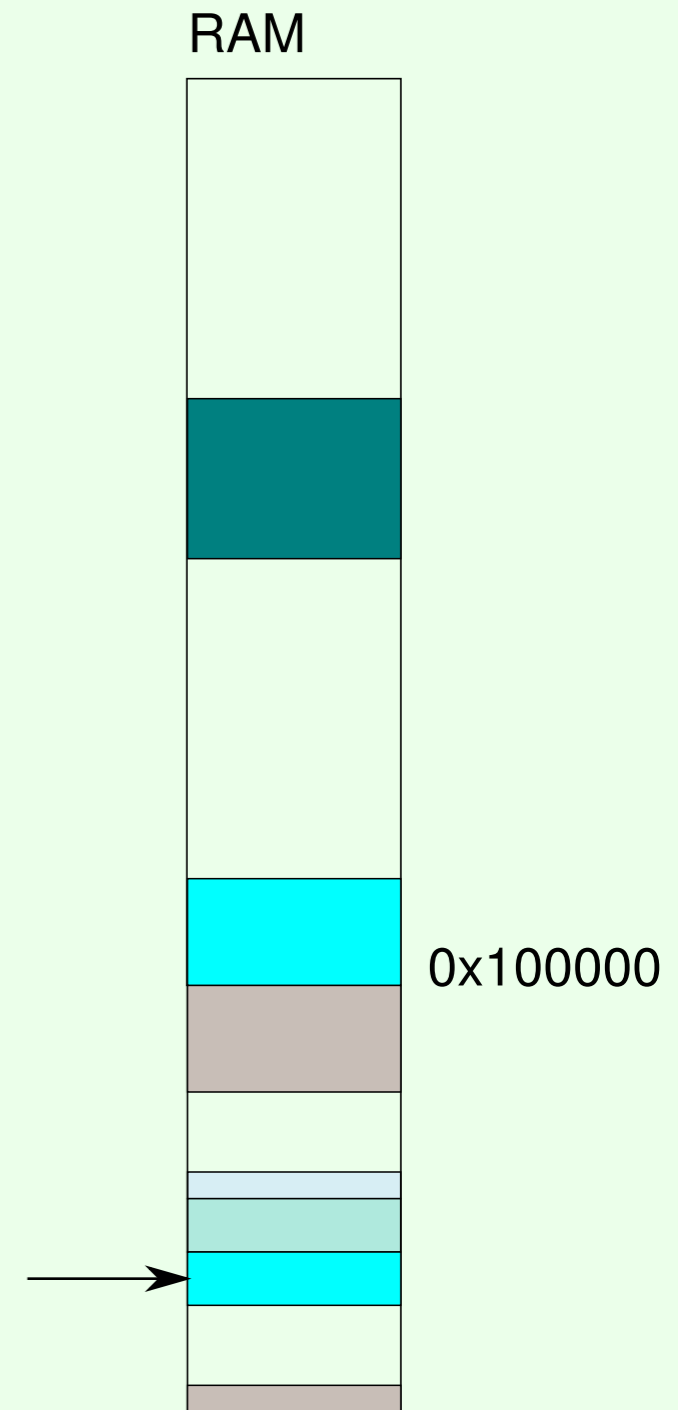
Booting Kernel

In the beginning...



Booting Kernel

real-mode

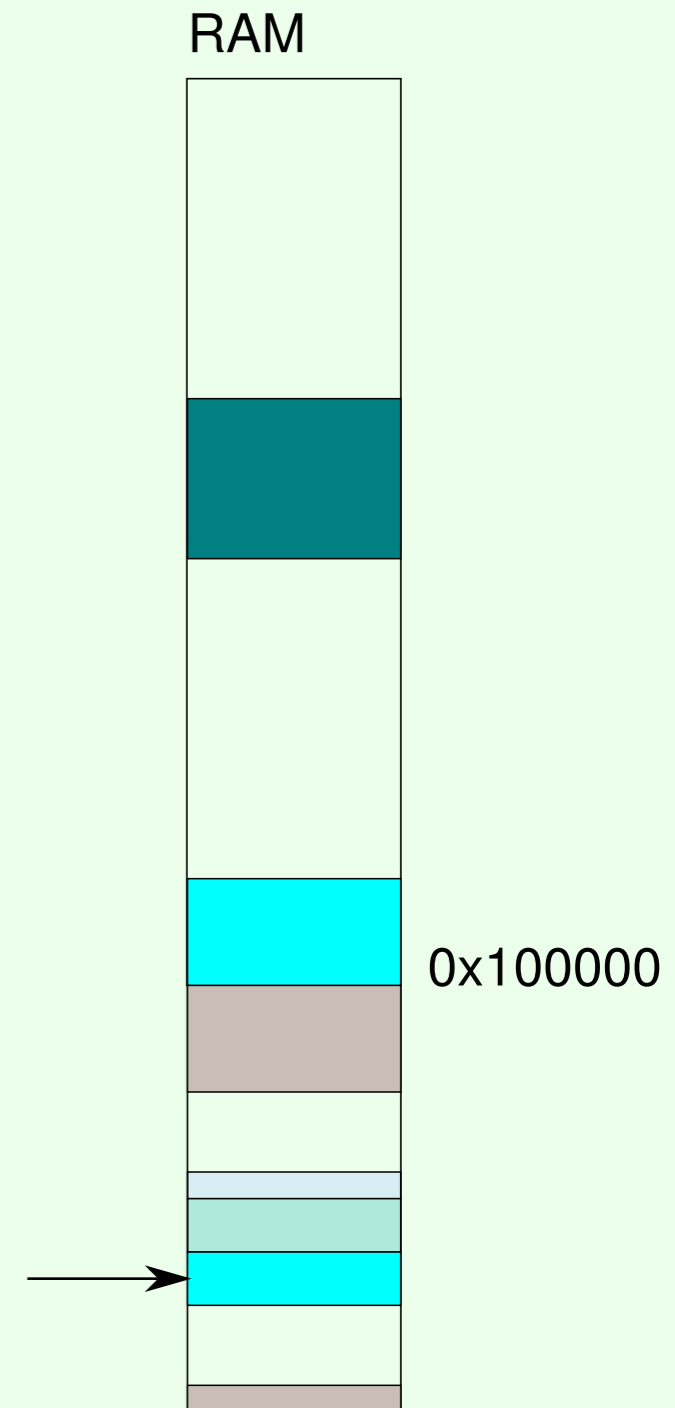


Booting Kernel

real-mode

`arch/x86/boot/header.S`

- 8086 asm
- enters at "start_of_setup"
- resets disk controllers
- sets up stack
- calls main()



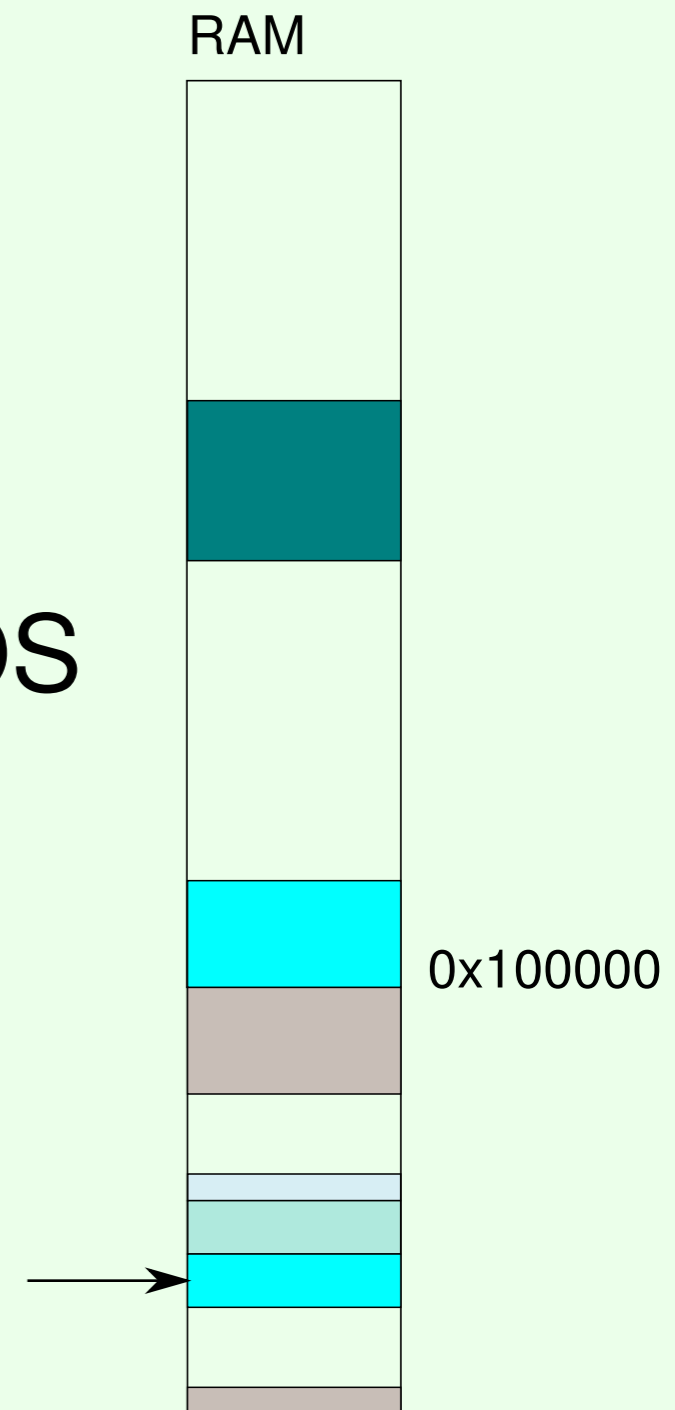
Booting Kernel

real-mode

`arch/x86/boot/main.c`

- parse some config options
- probe some HW through BIOS
- memory
- APM
- set video

- call `go_to_protected_mode()`



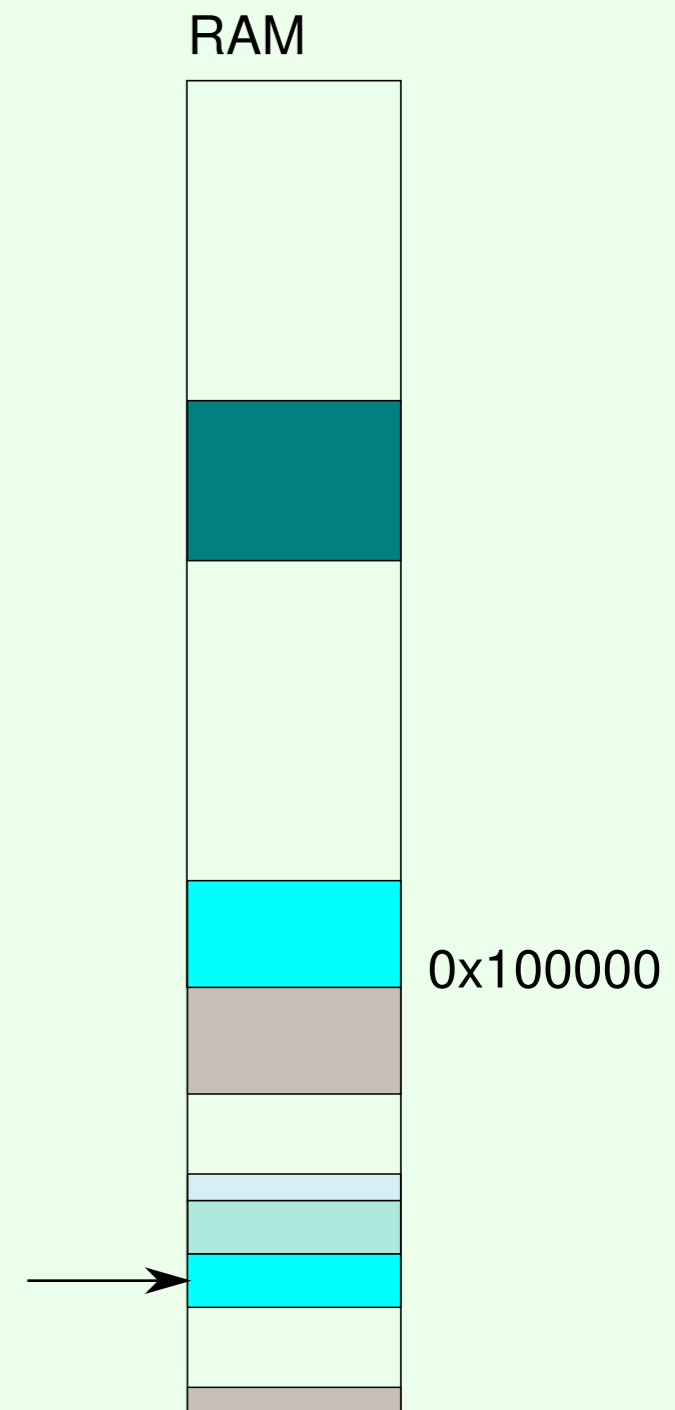
Booting Kernel

real-mode

`arch/x86/boot/pm.c`

- setup for switch
- IDT (null table)
- GDT (flat memory layout)

- switch to protected mode

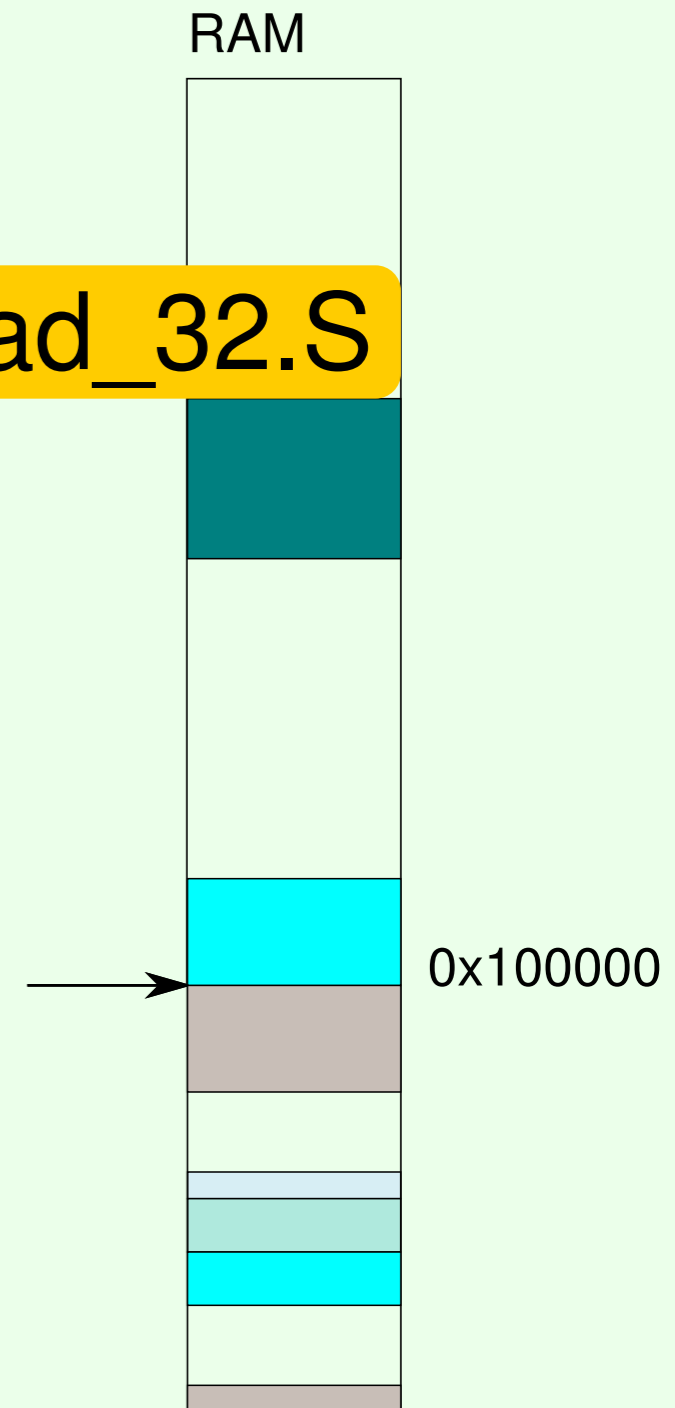


Booting Kernel

decompressing

`arch/x86/boot/compressed/head_32.S`

- 386 asm
- enters at "startup_32"
- sets up segments & stack

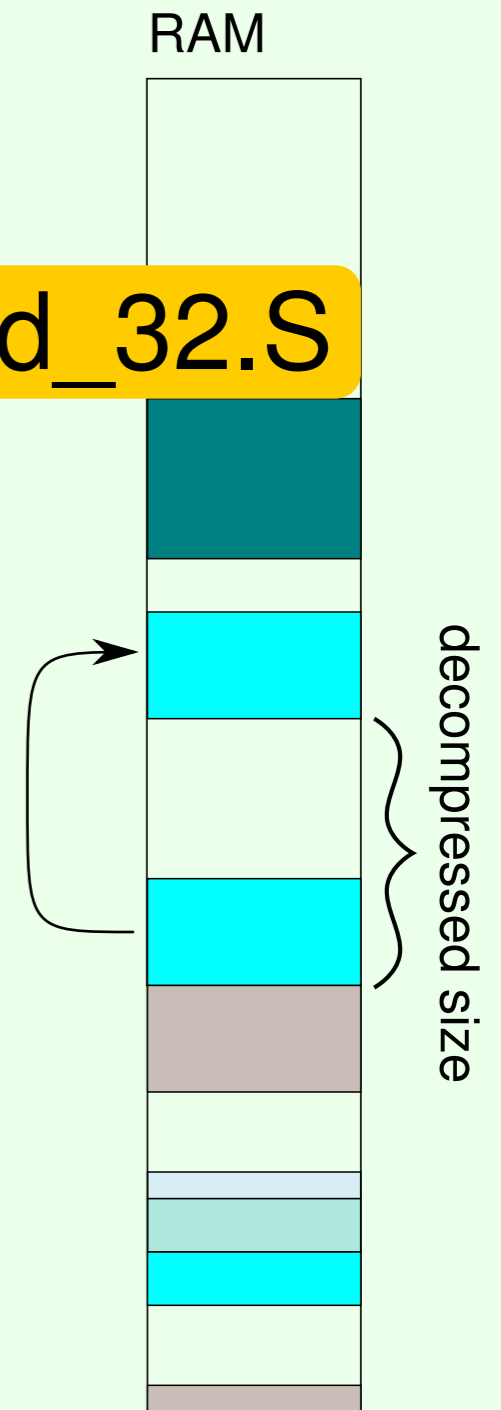


Booting Kernel

decompressing

`arch/x86/boot/compressed/head_32.S`

- 386 asm
- enters at "startup_32"
- sets up segments & stack
- moves things around

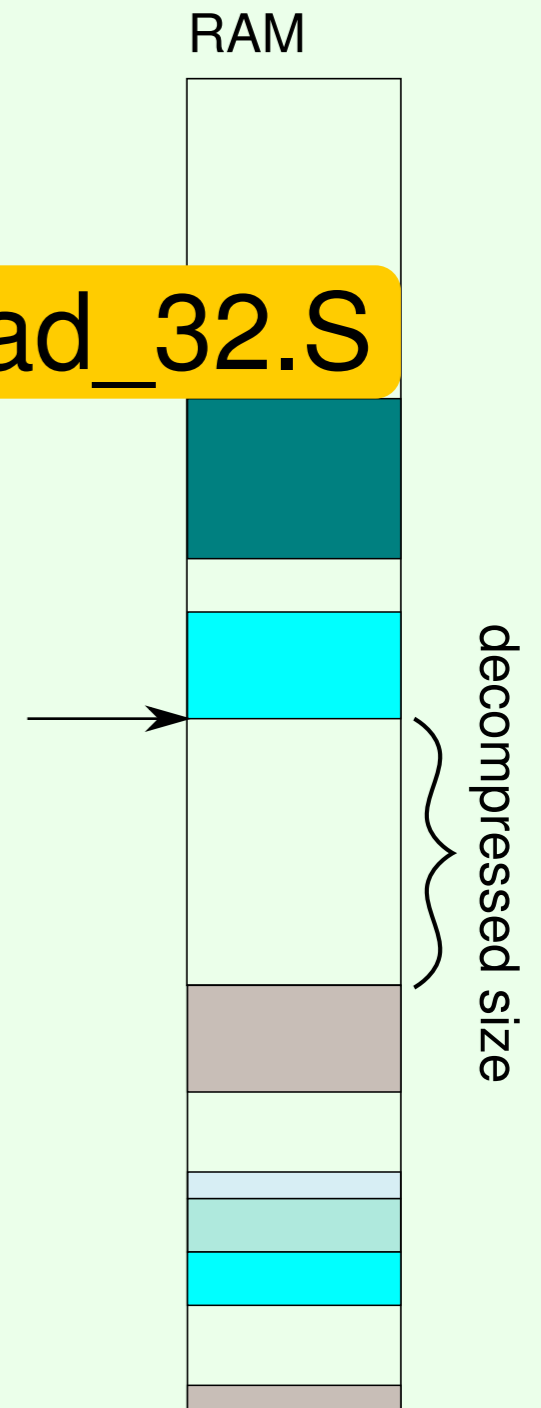


Booting Kernel

decompressing

`arch/x86/boot/compressed/head_32.S`

- 386 asm
- enters at "startup_32"
- sets up segments & stack
- moves things around

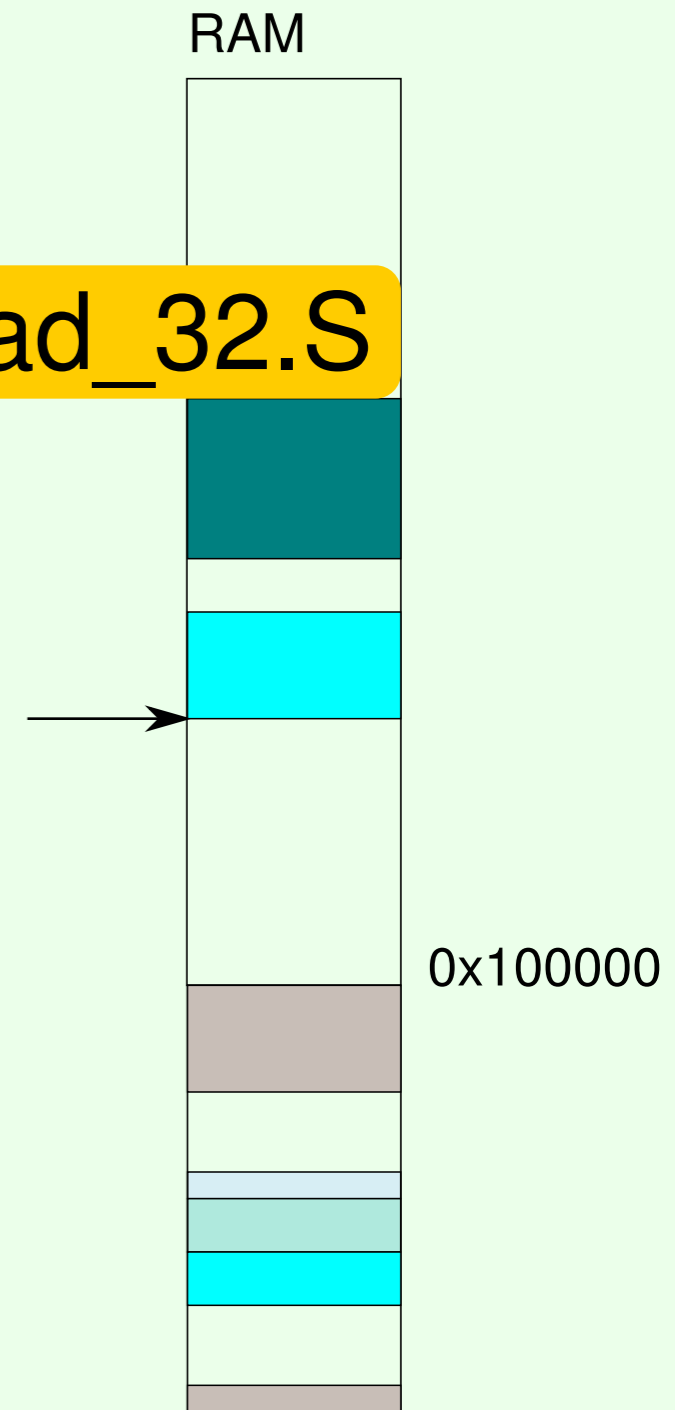


Booting Kernel

decompressing

`arch/x86/boot/compressed/head_32.S`

- 386 asm
- enters at "startup_32"
- sets up segments & stack
- moves things around
- call `decompress_kernel()`

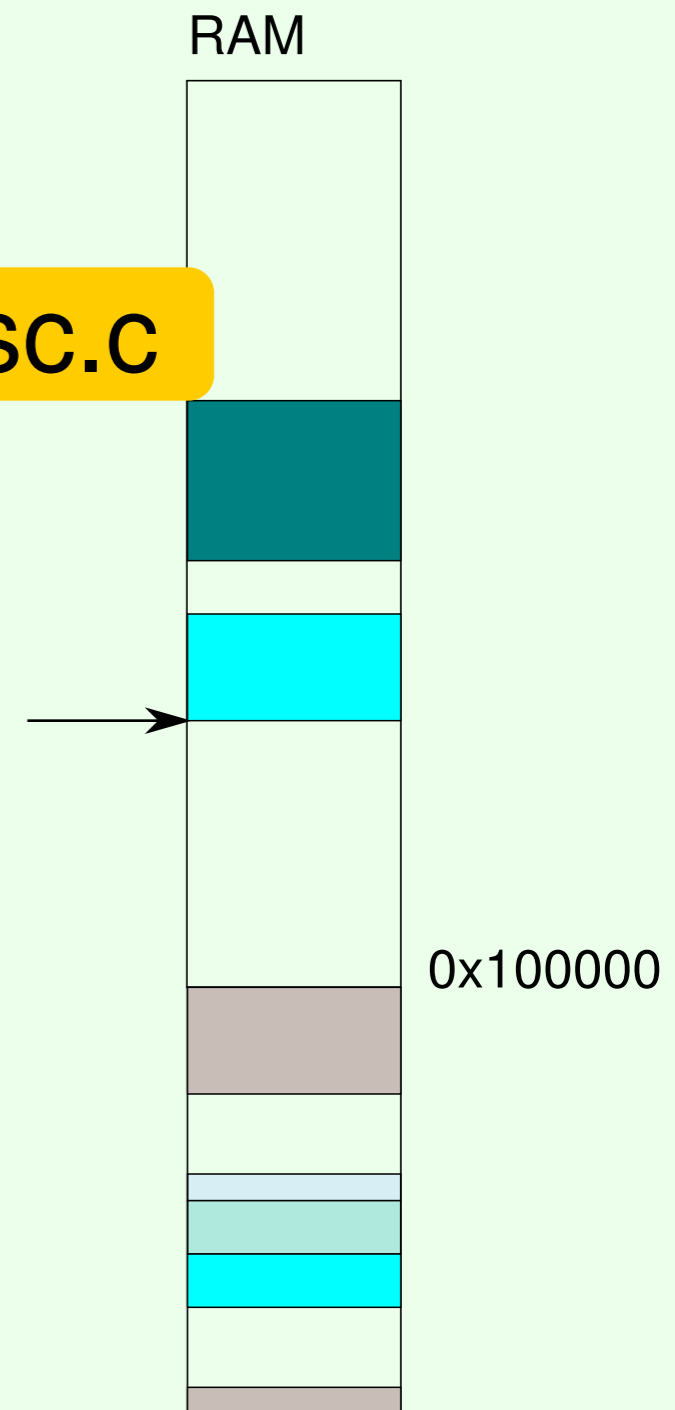


Booting Kernel

decompressing

`arch/x86/boot/compressed/misc.c`

- mostly C
- uses gunzip

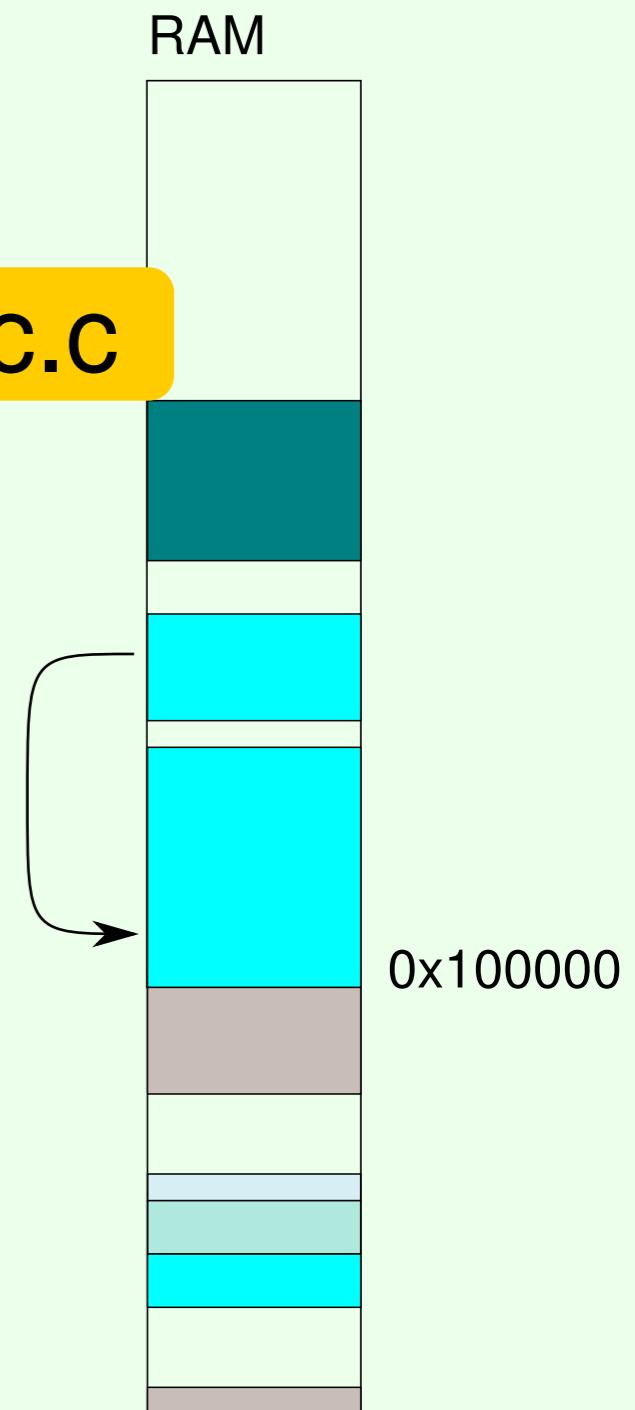


Booting Kernel

decompressing

`arch/x86/boot/compressed/misc.c`

- mostly C
- uses gunzip

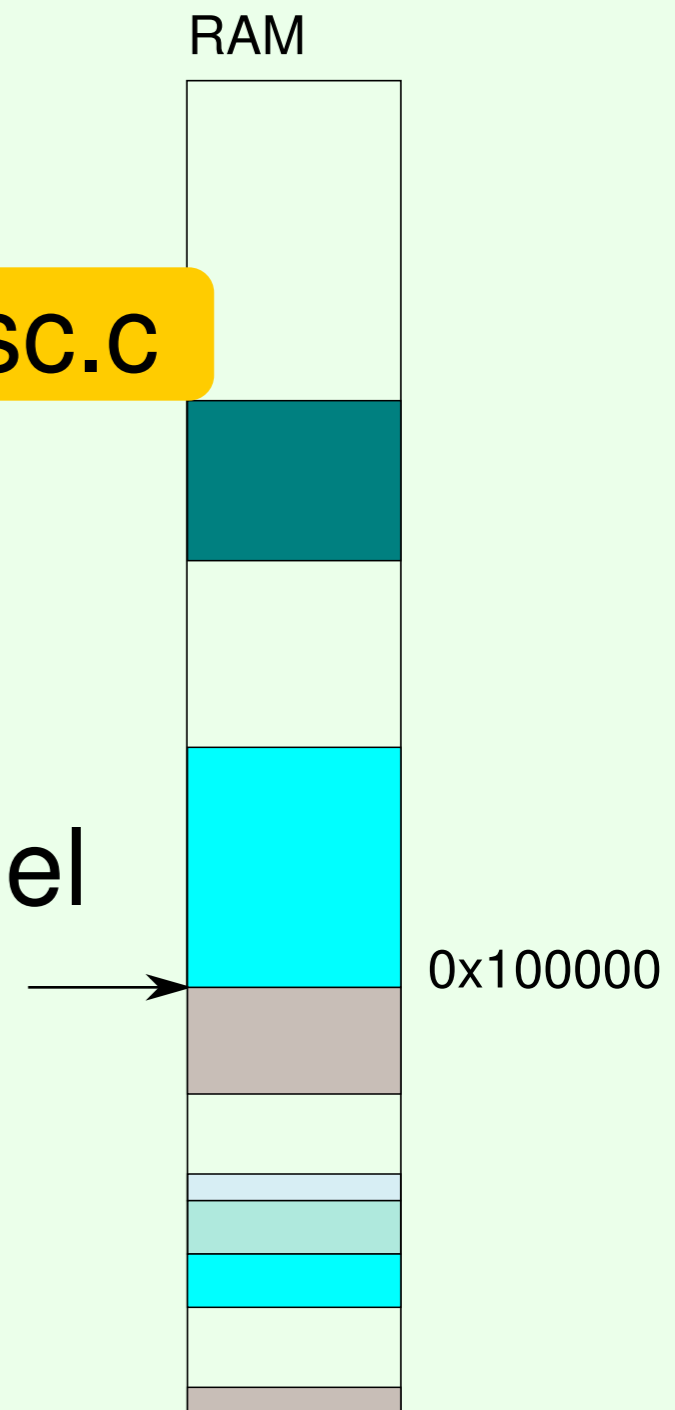


Booting Kernel

decompressing

`arch/x86/boot/compressed/misc.c`

- mostly C
- uses gunzip
- jumps to decompressed kernel

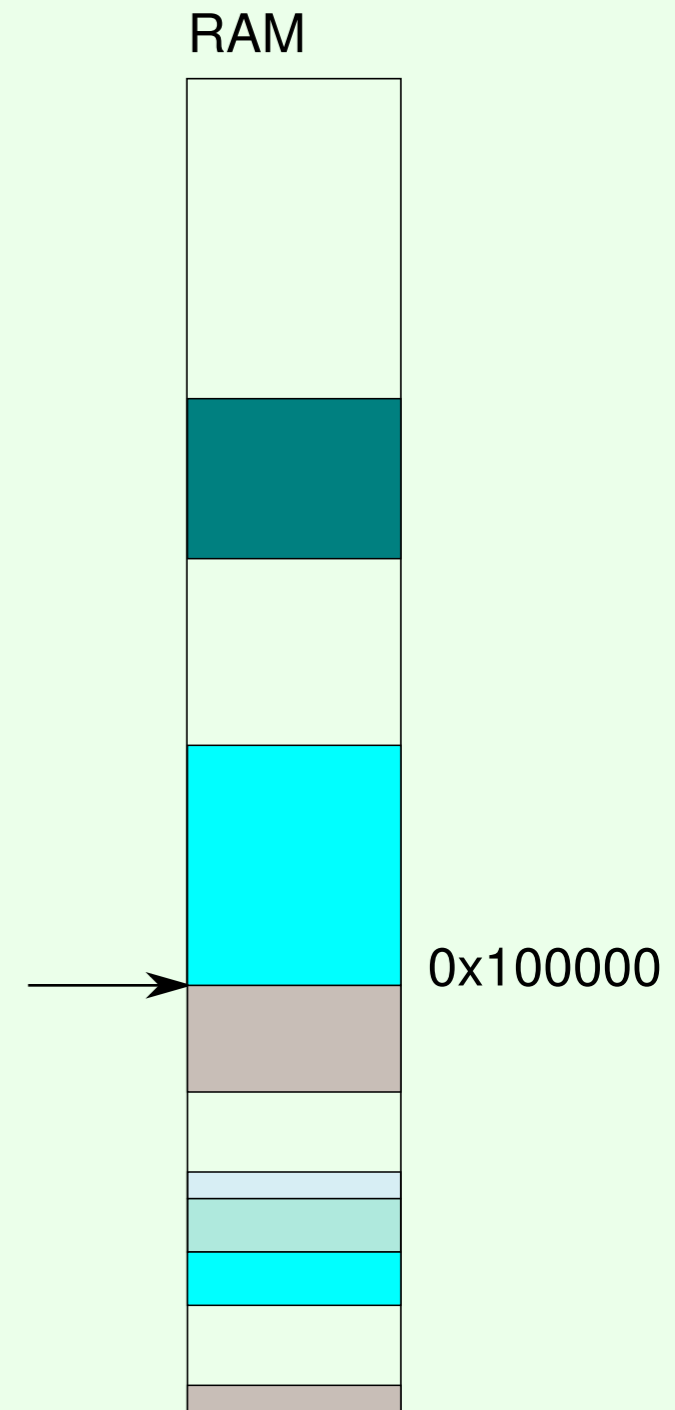


Booting Kernel

start real kernel

`arch/x86/kernel/head_32.S`

- 386 asm
- enters at "startup_32"
- setup segments & stack

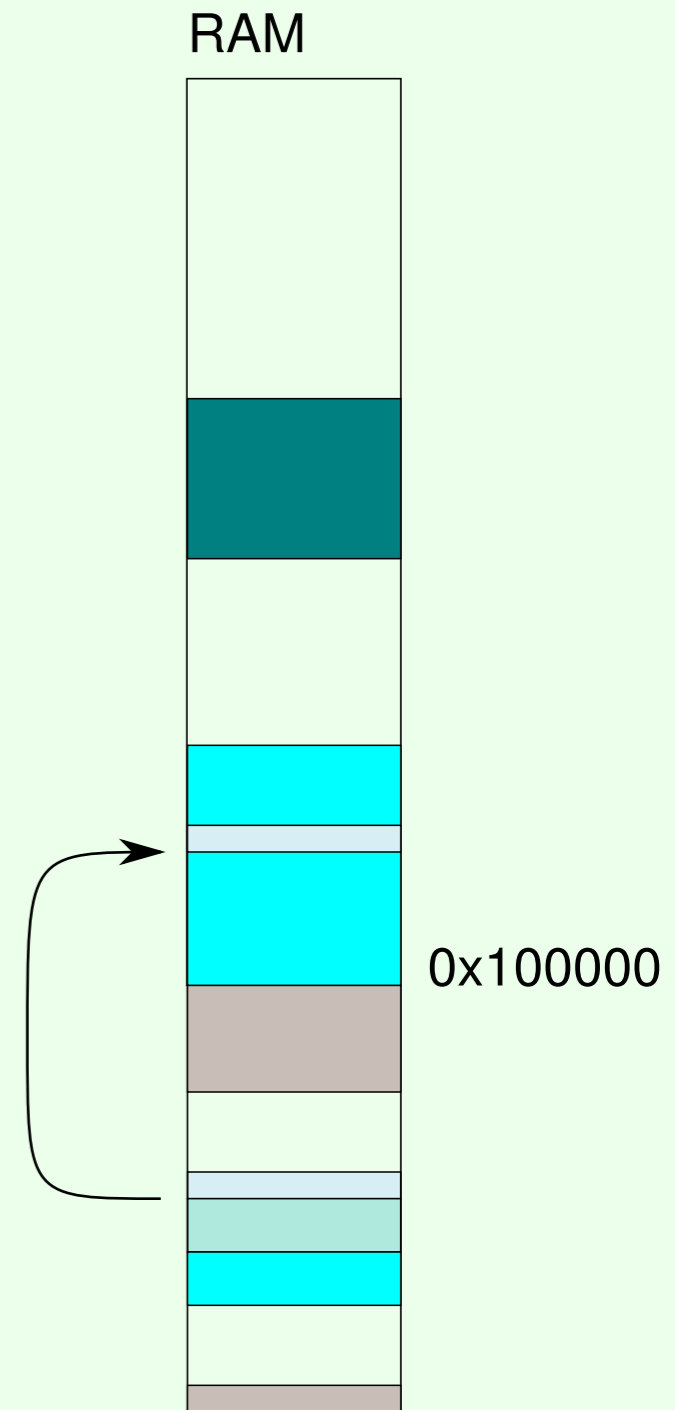


Booting Kernel

start real kernel

`arch/x86/kernel/head_32.S`

- 386 asm
- enters at "startup_32"
- setup segments & stack
- copy cmdline

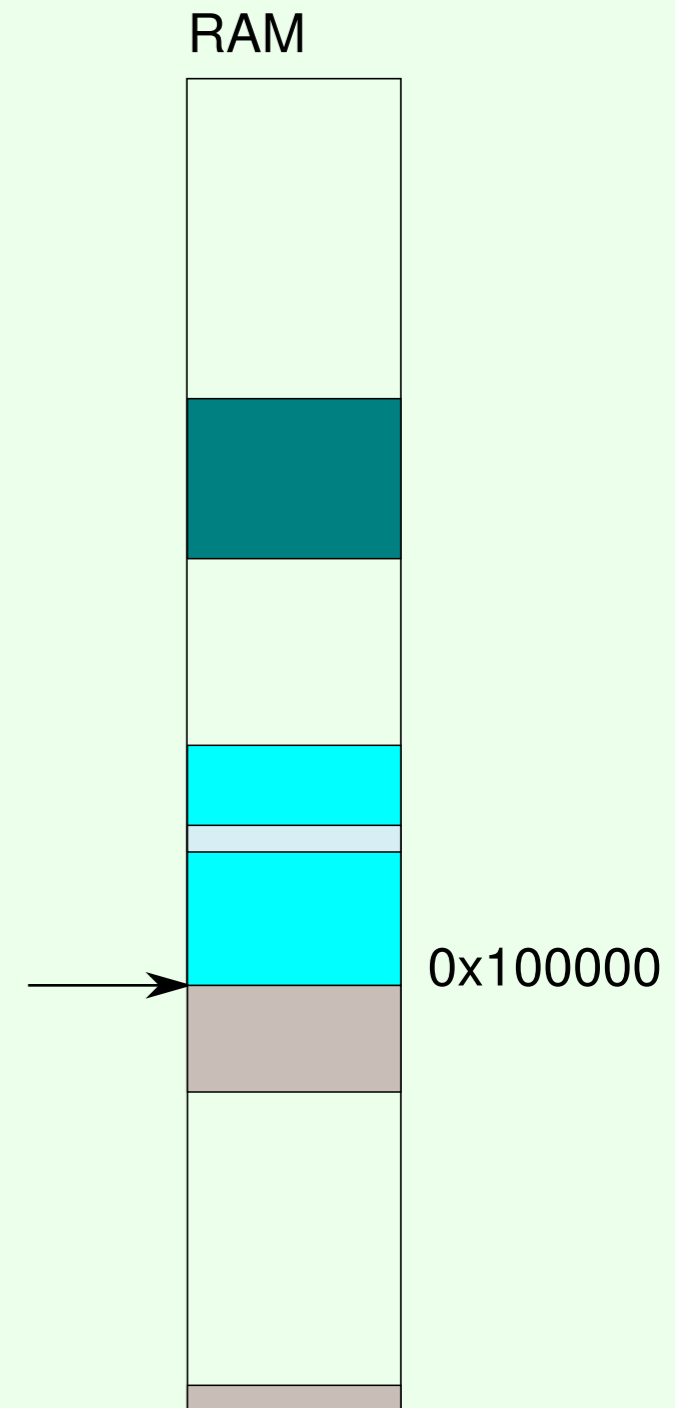


Booting Kernel

start real kernel

`arch/x86/kernel/head_32.S`

- 386 asm
- enters at "startup_32"
- setup segments & stack
- copy cmdline
- setup virtual memory
- setup_idt
- call start_kernel()



Booting Kernel

initialize subsystems

`init/main.c`

`start_kernel()`

- init all subsystems
- call `rest_init()`

Booting Kernel

kick off scheduler

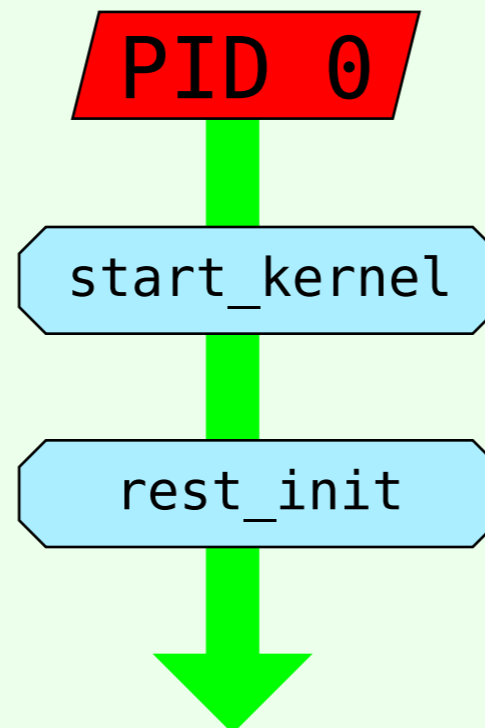
`init/main.c`

`rest_init()`

- start PID 1 task
- PID 0 calls `cpu_idle()`

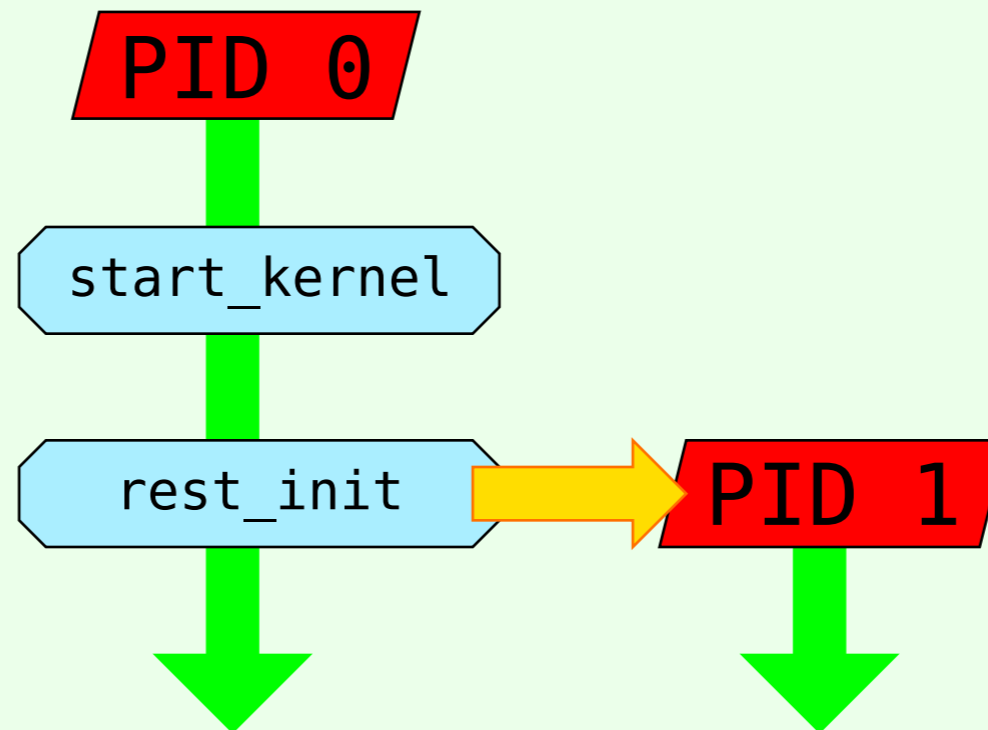
Booting Kernel

kick off scheduler



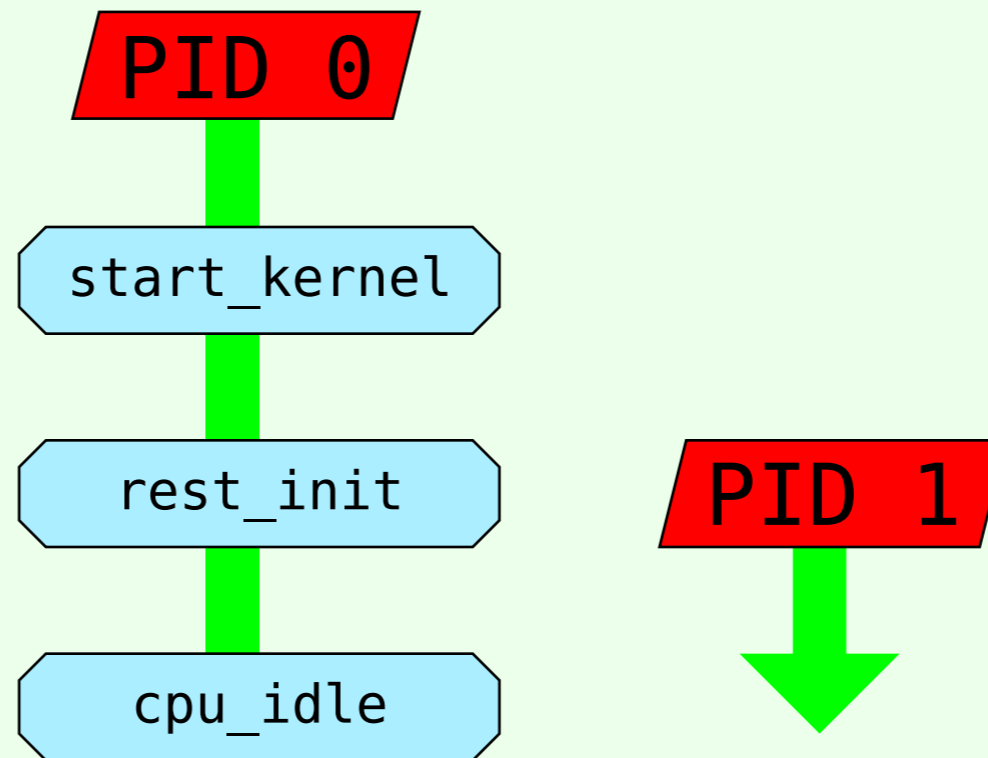
Booting Kernel

kick off scheduler



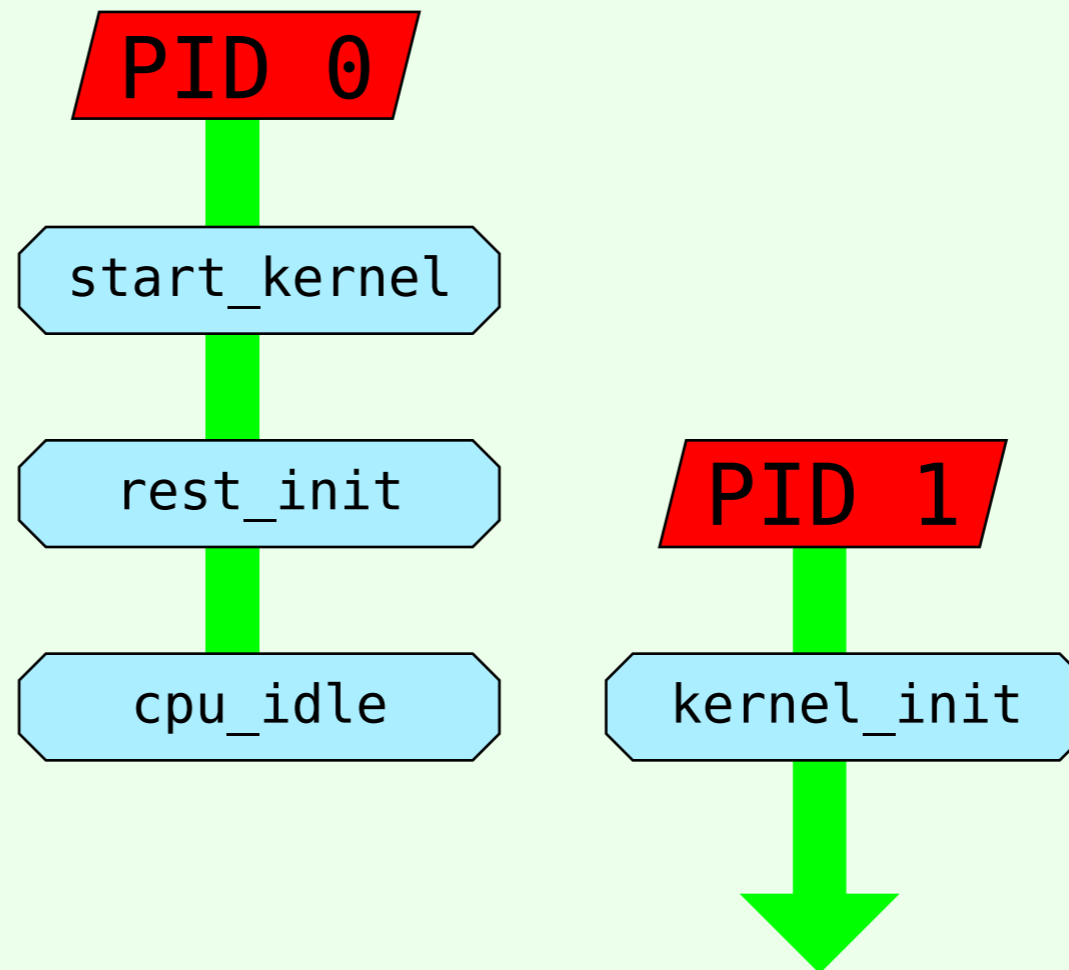
Booting Kernel

kick off scheduler



Booting Kernel

kick off scheduler



Booting Kernel

finish off init stuff

`init/main.c`

`kernel_init()`

- start SMP
- init more subsystems
- call `init_post()`

Booting Kernel

finish off init stuff

`init/main.c`

`init_post()`

- finds "init" executable
- run via `kernel_execve()`
- else panic

Any questions?

Bart Trojanowski

<http://www.jukie.net/~bart/>

August 21, 2008