Part 7

Virtual Memory:
from relocation registers
to
paged & segmented virtual memory

VM

- Finished discussing CPUs
- hardware to implement Virtual Memory (VM)
  - an important part of most cpus
  - various versions, including

Terms Used . . .

- Relocation register
- base register
- segment registers
- page registers
- Dynamic Address Translation (DAT) box
- Memory Management Unit (MMU)

Problem: relocate this program!

Assembler

```
Geo:  CLA  X
      STA  Y
      JMP  Geo
X:    BSS  1
Y:    BSS  1
```

Half-assembled

```
0000  CLA  0003
0001  STA  0004
0002  JMP  0000
0003  (value of X)
0004  (value of Y)
```

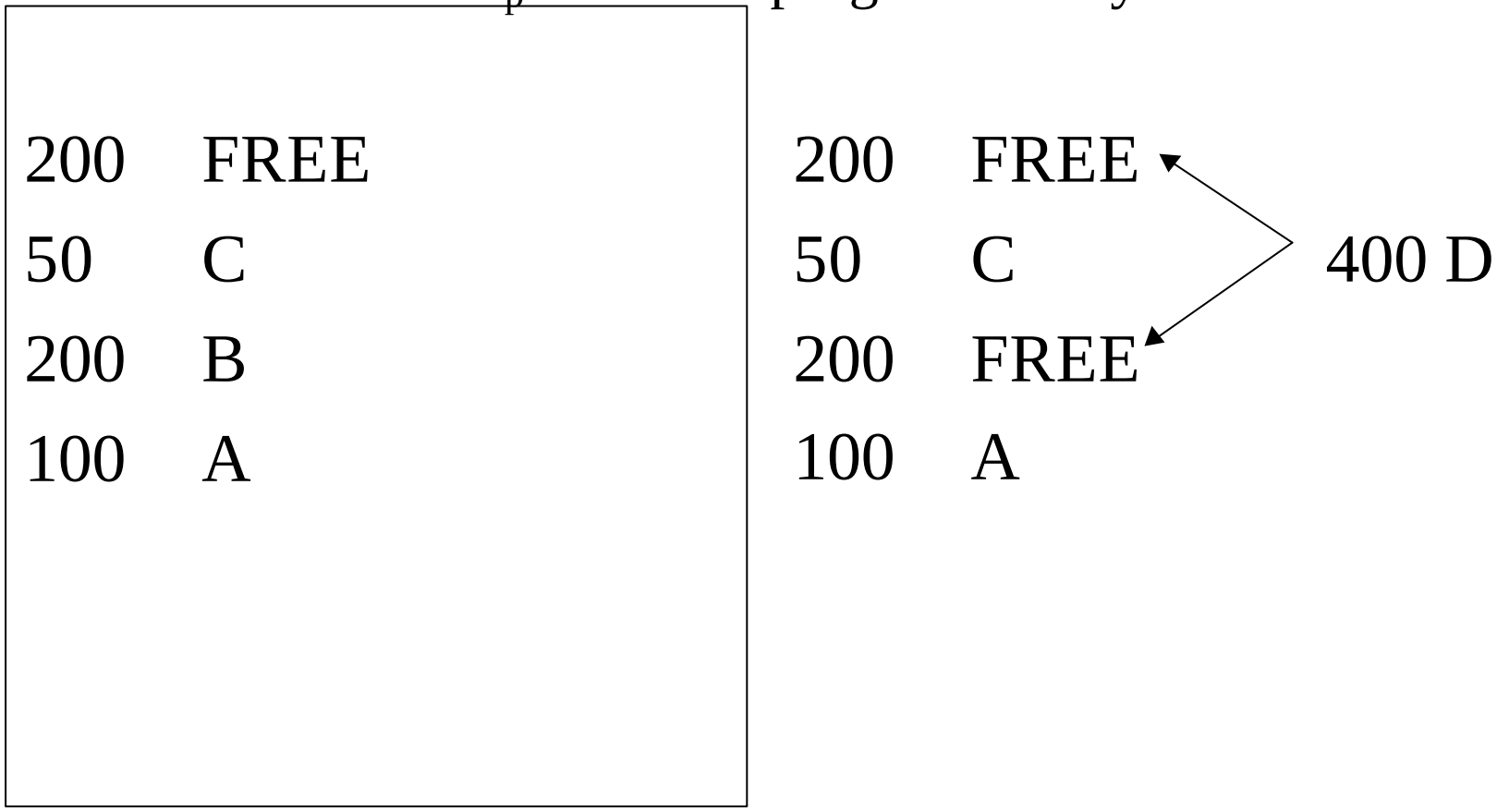
Move to 1000 ff:
fixup needed:

1000	CLA	0003	0003 + 1000 = 1003
1001	STA	0004	0004 + 1000 = 1004
1002	JMP	0000	0000 + 1000 = 1000
1003	(value of X)		
1004	(value of Y)		

Program relocation: who cares?

The Problem:

Use of M_p in a multiprogrammed system:



Solutions:

- O/S 360 solution
  - : wait until C dies, then load D (awk!)

- rational solution:
  - move or relocate C
    to create
  - 400K of contiguous free memory

NB

- Same problem arises

  when swapping between disc and primary memory

How to relocate?

1] Position Independent code

- all addresses relative to Program Counter value
- rare (PDP-11)

2] Re-execute the linking loader (boo!)

3] Simple, fast hardware solution (good)

all of these cause a constant

(*relocation bias* or *offset*)

to be added to each memory address just before it is referenced.

How?

- Provide a single *relocation register* RELOC
  - invisible to application programmers
  - accessible to operating system programmers

- Let

  Y <-- y + c(IR) + c(RELOC)

  done in hardware

How?

- Write all code to begin at “virtual” cell 000
  - (as though they were going to run in cells

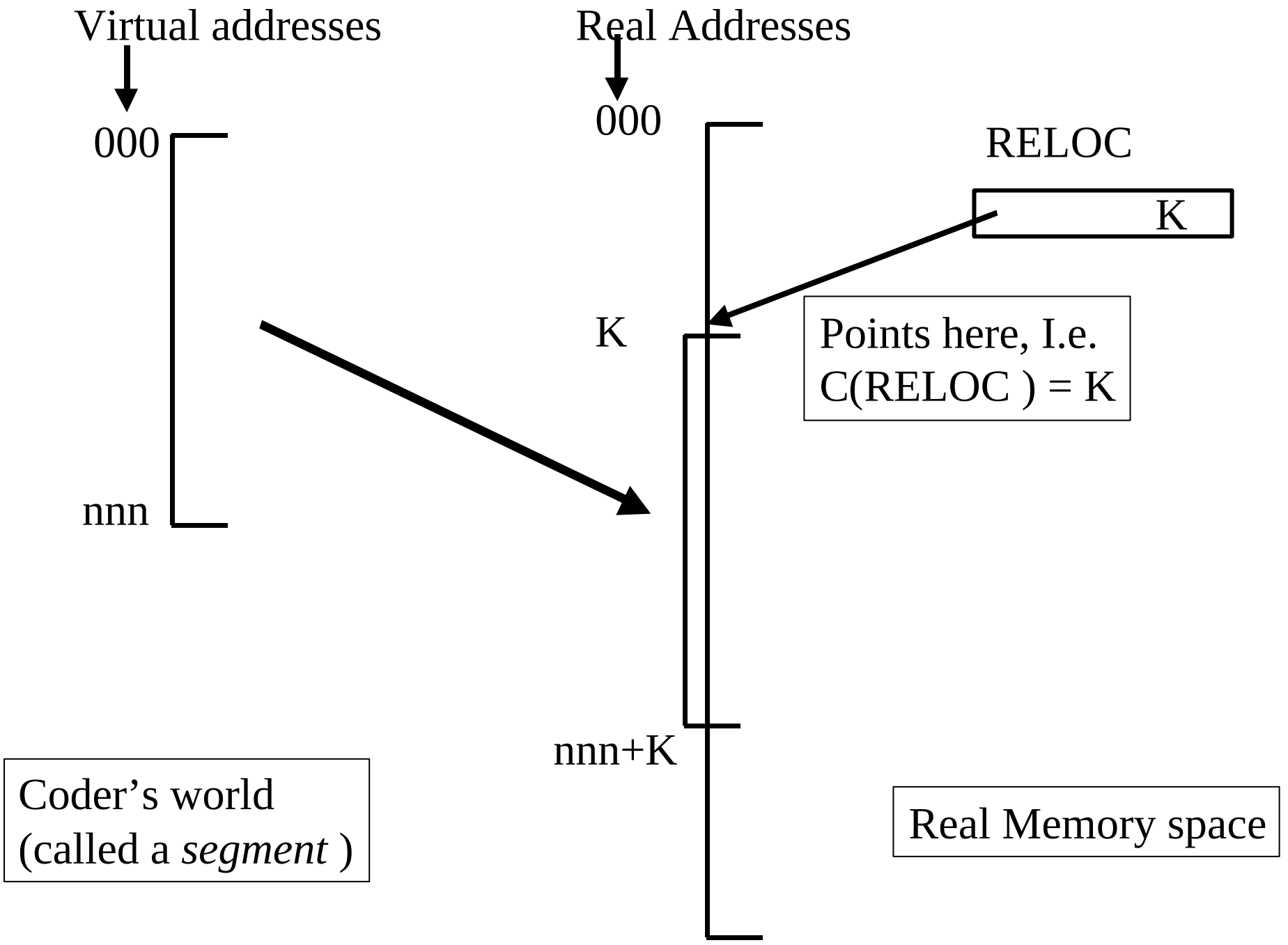
    000

    001

    002

    003

    . . .)
- Load the real starting address in RELOC
- So . . .


Virtual addresses
Real Addresses
000
000
RELOC
K
K
Points here, I.e.
C(RELOC) = K
nnn
nnn+K
Coder's world
(called a *segment*)
Real Memory space

Refinement

- Add a *bound* register where

c(BOUND) = nnn //segment length

Do in hardware:

```
IF [ y + c(IR) > nnn]
THEN INTERRUPT;
```

--Provides some *protection*

Assessment:

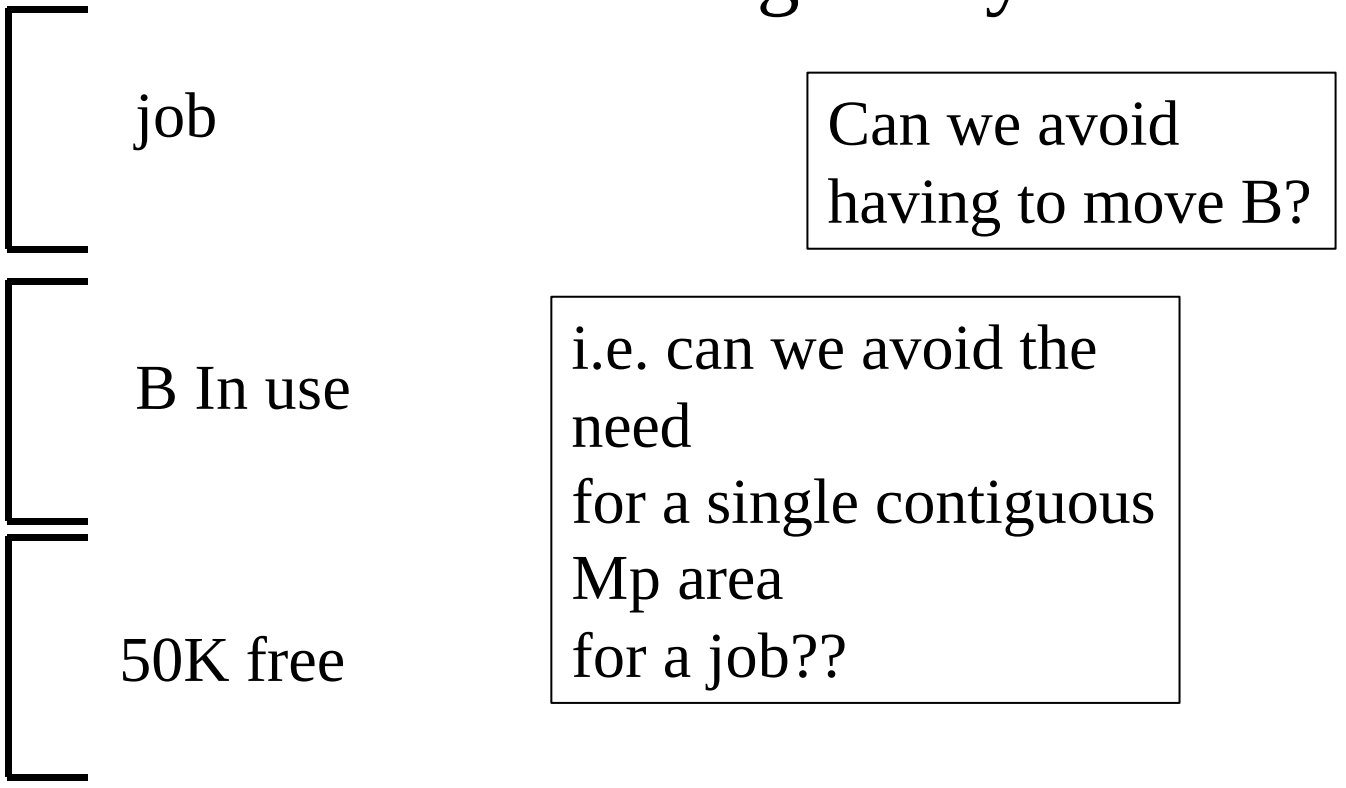
- What have we achieved so far?
  - Relocation register
    - (sometimes called a *base register* , do not confuse with S/360 base register)

    allows us to move a job or process or executable around in Mp **as a single unit**

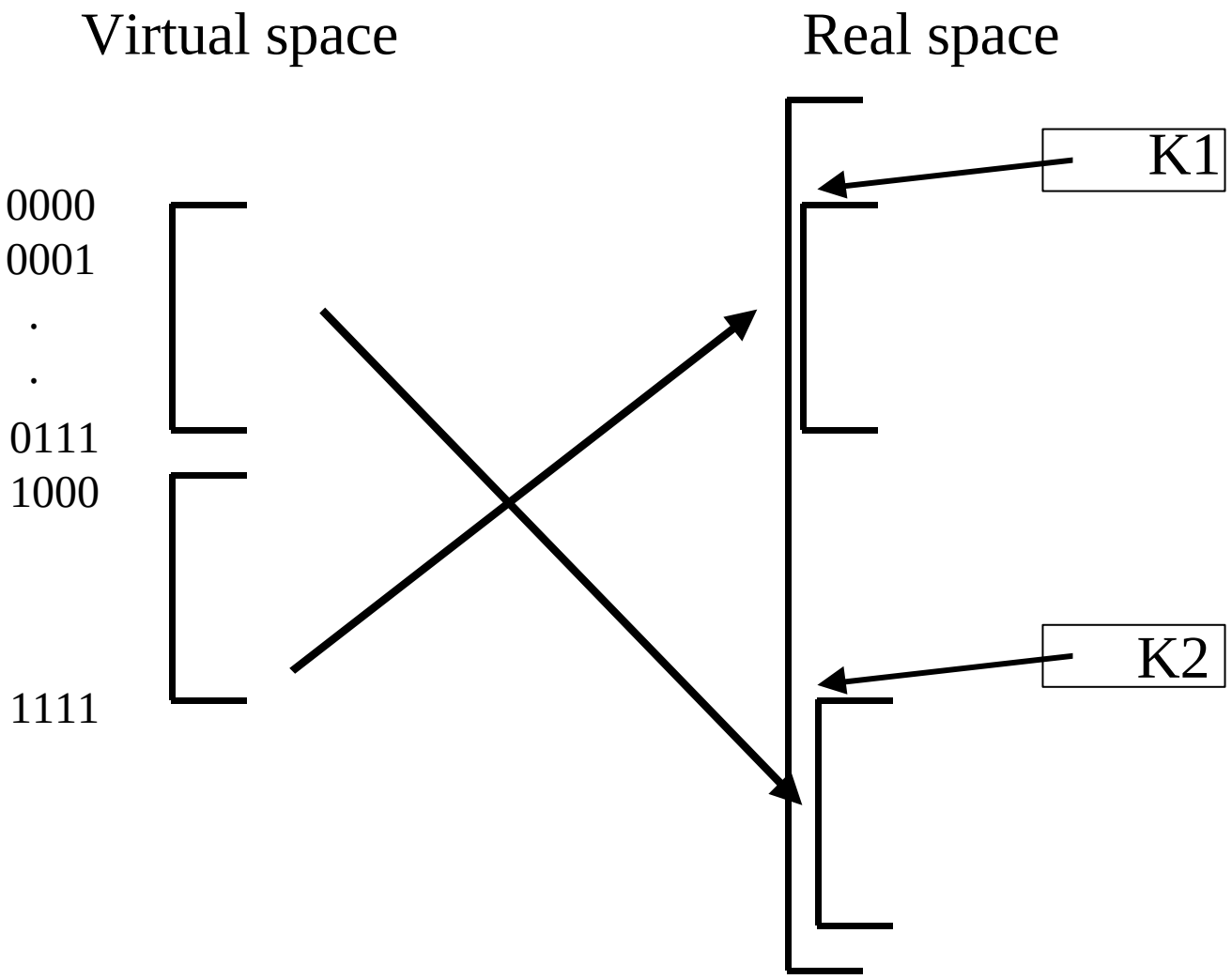
- Example: Honeywell 6050

Next step:

- Suppose our job needs more Mp
- Mp is available but not contiguously:



Supply two relocation registers
Use bit 0 of the effective address to select which register to use:



What have we done?

- A process can now be moved around in real Mp (or swapped in and out)

  by ***segment***
- Mp need only be contiguous within each segment
- relocation registers RELOC are called

  segment registers if there are >1 of them

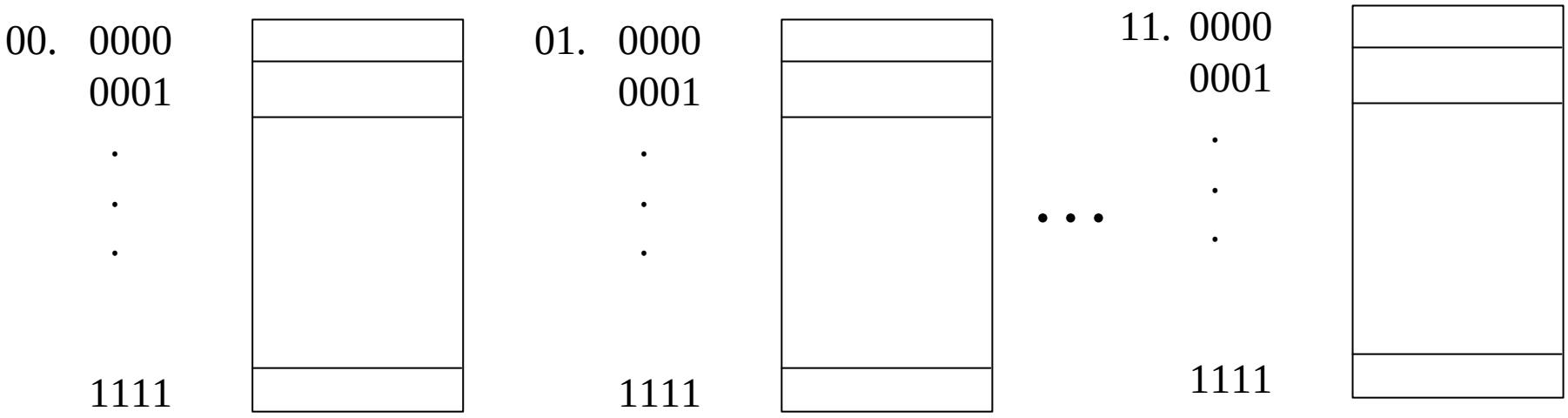
Generalization to *n* segment registers:

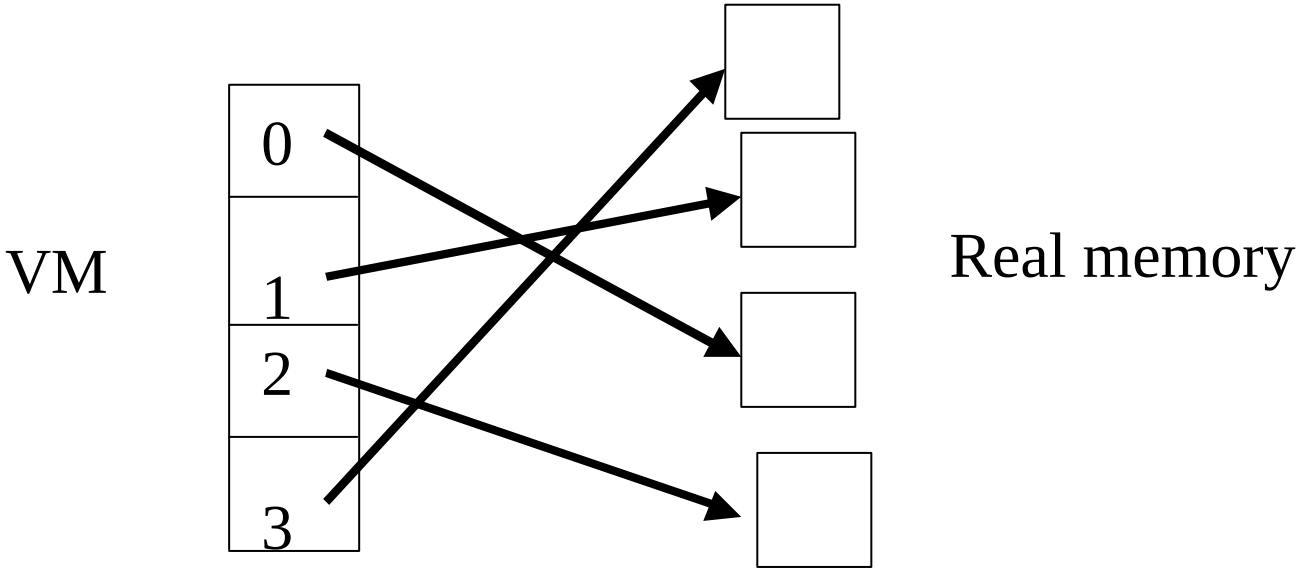
Structure of coder's virtual addresses:

Segment # cell within segment

Example

- For
  - 6 bit virtual address (64 addresses)
  - 2 bit segment number (4 segments)
  - 4 bits of cell-within-segment address (16 cells/segment)
- we have:


00. 0000
0001
1111
01. 0000
0001
1111
11. 0000
0001
1111
Segment 00
Segment 01
Segment 11


VM
0
1
2
3
Real memory

NB

- Real memory segments are variable length, but < 17 cells long

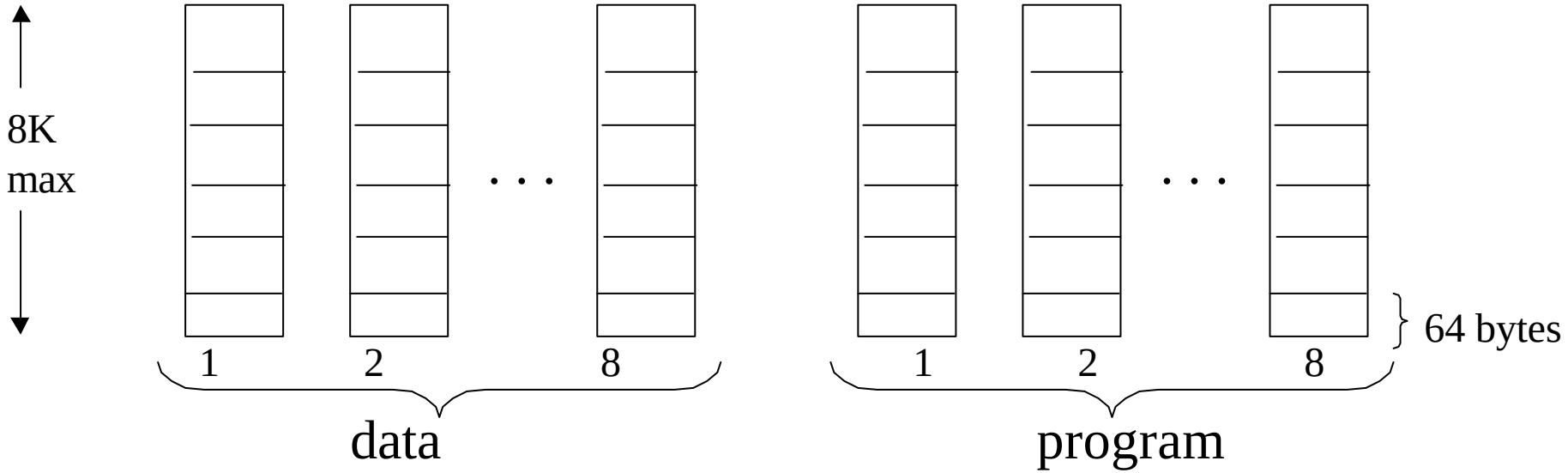
- the real space may be
  - smaller,
  - the same size, or
  - bigger than the virtual space

Example:
PDP-11/45 segment box

- Sits between cpu and (real) memory
- maps virtual addresses emanating from cpu into real addresses for memories
- 11/45 virtual memory architecture:

11/45 virtual memory architecture

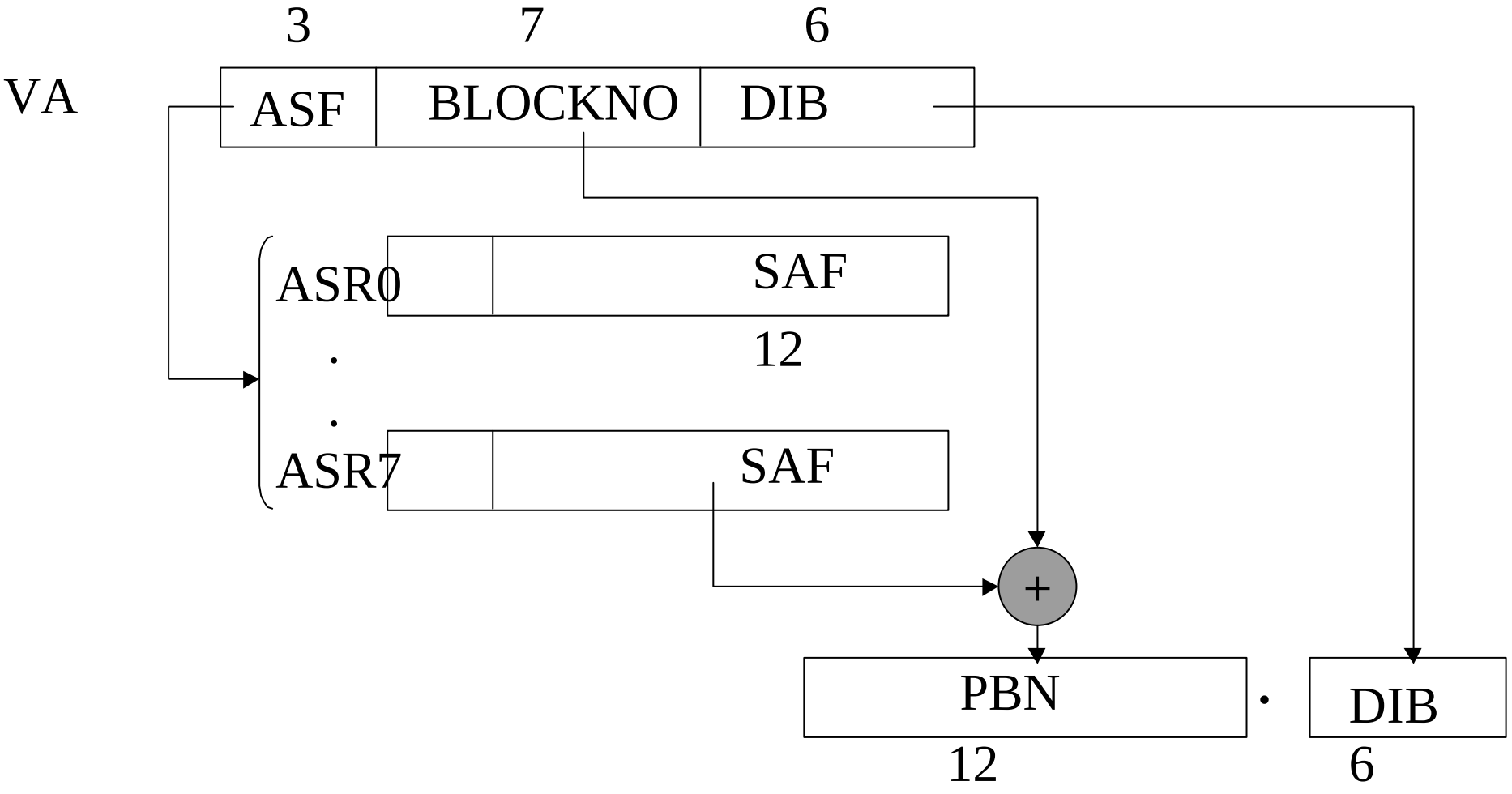
3 bit	segment number
13 bit	displacement in segment, organized as
	7 bit block number
	6 bit displacement in block



11/45 virtual memory architecture

- Segments can be relocated independently of one another
- blocks cannot
- so, swapping and relocation are by segment

11/45 virtual memory architecture
the memory management unit (MMU)



11/45 virtual memory architecture

- Physical addresses (18 bit = 256K) form a **bigger** space than virtual addresses

  (64 + 64 K)

- access control (read-write-execute permission bits) in registers associated with Active Segment Registers (ASRs)
- MMU registers writable and readable when processor is in System mode only

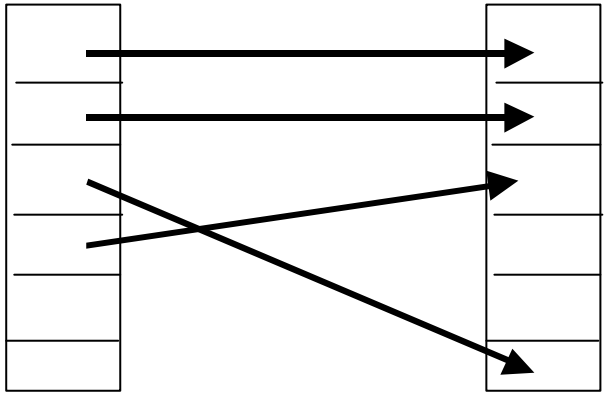
11/45 MMU

- Price (1978)
  - $5 000 US
  - 90 nsec latency per memory access

For our next trick . . .

- blocks are fixed length, must be contiguous
- Segments are **variable** length in units of 1 block or 64 bytes
- so swapping will fragment Mp (areas of free Mp of different, hence wrong sizes)
- IF we can have noncontiguous, fixed size blocks, the problem disappears
- i.e.

noncontiguous, fixed size blocks



segment

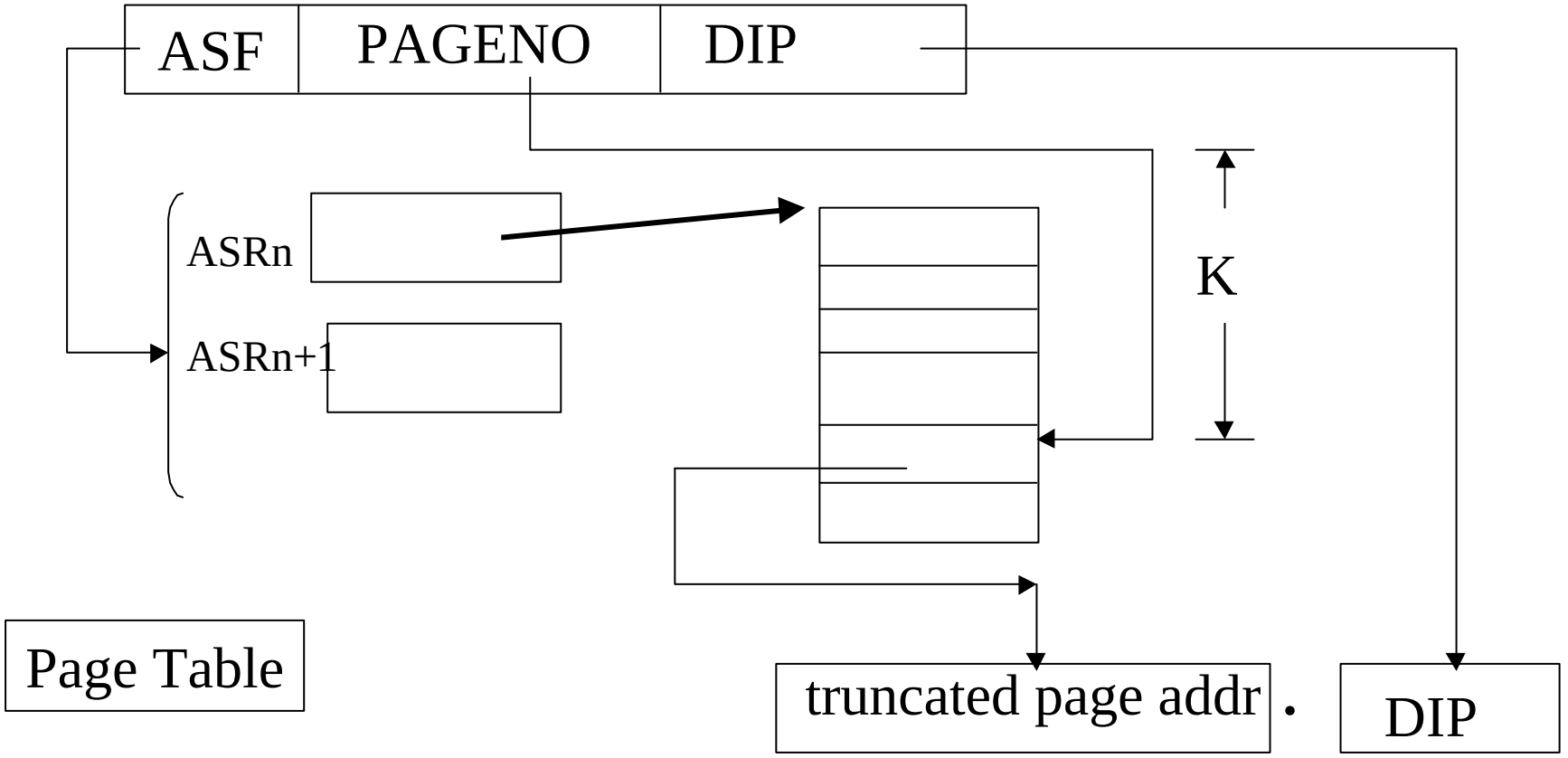
Blocks of physical memory

DEFINITION: such blocks are called *pages*

Paged virtual memory: How to implement?

- Segment Register points not to a segment but to a *page table*
- page table translates or maps page numbers (1,2,3, …) to real Mp addresses of pages IF the page is memory-resident
- IF the page is not memory-resident , raise an interrupt, and enter a procedure to get it
- diagram:

Architecture of MMU for segmented and paged VM



NB

- One page TABLE per segment
- pages cannot start anywhere:
  - start address must be 0 modulo pagesize
- address translation time now includes memory cycles
- the pagetable might not be memory-resident (AWK!)

Sample systems

- 1 segment, no pages: Honeywell 6000
- 16 segments, no pages: DEC PDP-11/45
- paged, no segments: ATLAS,
  MIPS chip

- paged & segmented: IBM 360/67
  GE 645 (multics)

GE 645 / multics overview

- Mother of modern multiprocess Oss
  - originated many of the concepts
- needed an infinite virtual memory,

  based on a very finite physical Mp (512K words)
- *segment* as fundamental primitive: processes, files, I/O devices . . .

GE 645 / multics overview

- Segments can be large, so paging needed too (pages of 1K and 64 words)
- process structure:
  - large set of large segments
  - some data
  - some code
  - some *files*

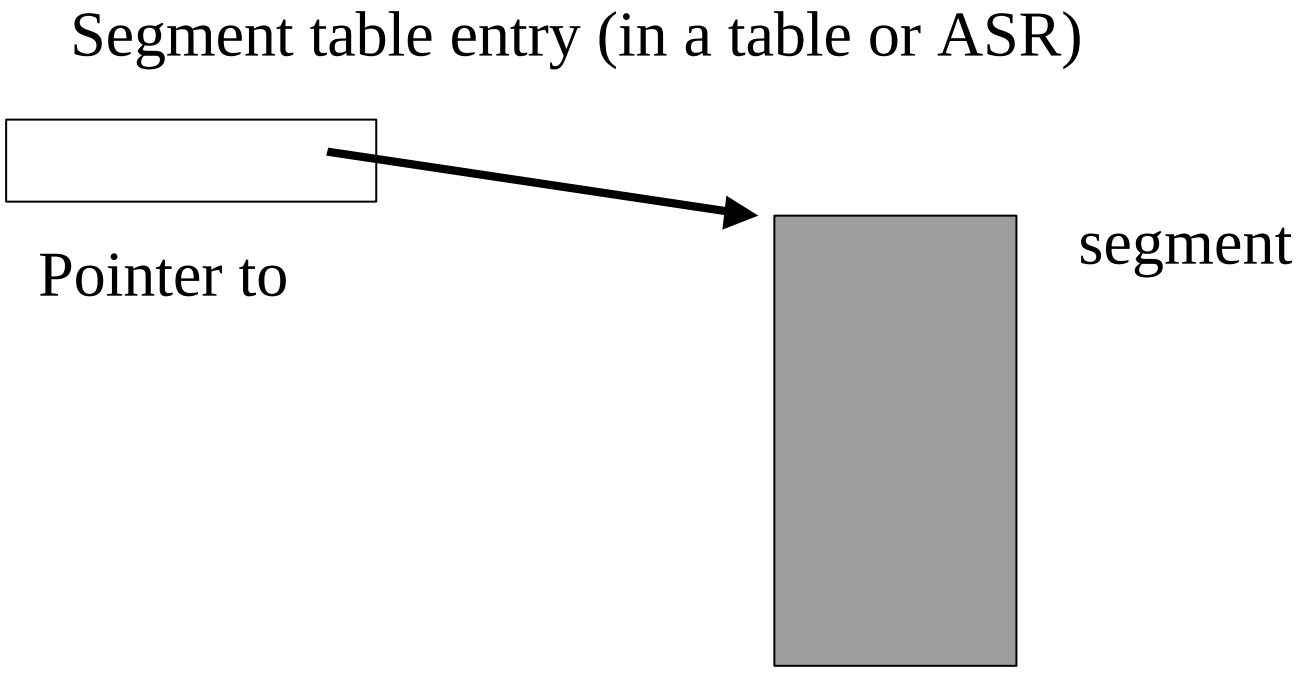
Process structure

- A single *descriptor base register* (dbr)
- holds a *process descriptor* which points to a
- *segment descriptor table*, whose entries point to
- *page tables*, whose entries point to pages
- context switch: reload the dbr
- multiple users can share a segment or page
- address translation time: unbounded

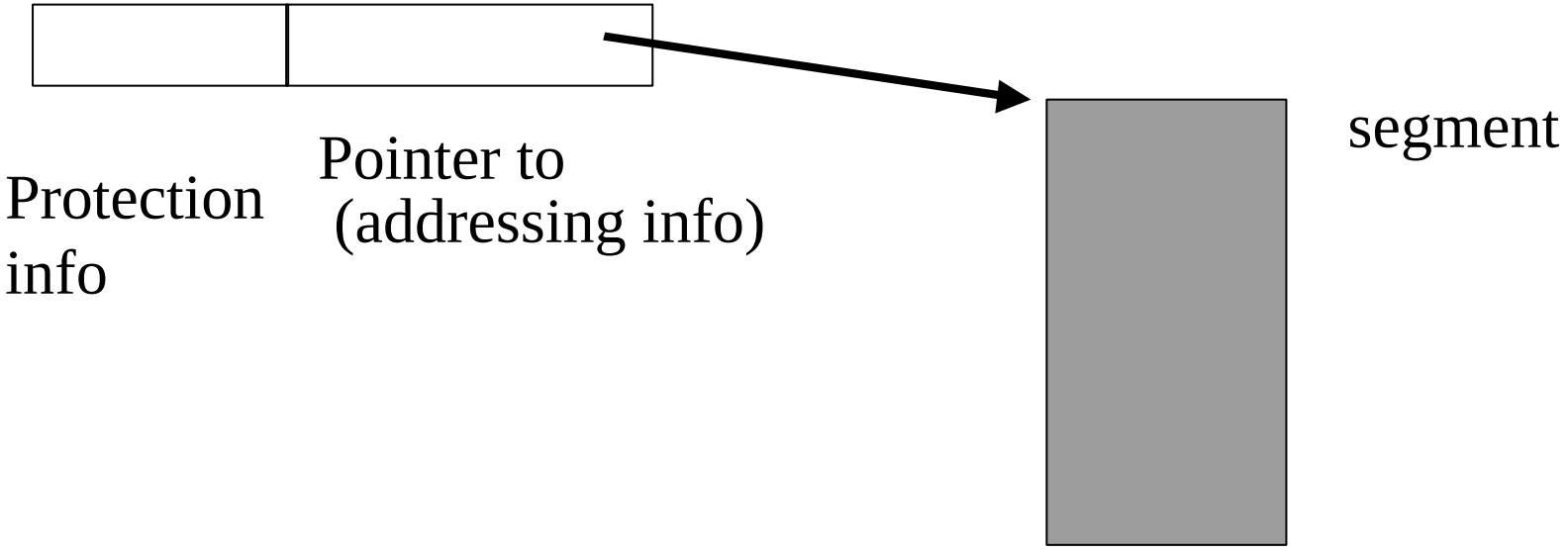
Protection or access control

- Definition: ensuring that a process has access to *every segment we wish it to, in the modes we wish (R-W-E)*
- -- and *no others*!
- Invaluable to
  - limit damage from bugs
  - enforce security
- How?? The Segment Table Entry

Adding protection information to STEs:



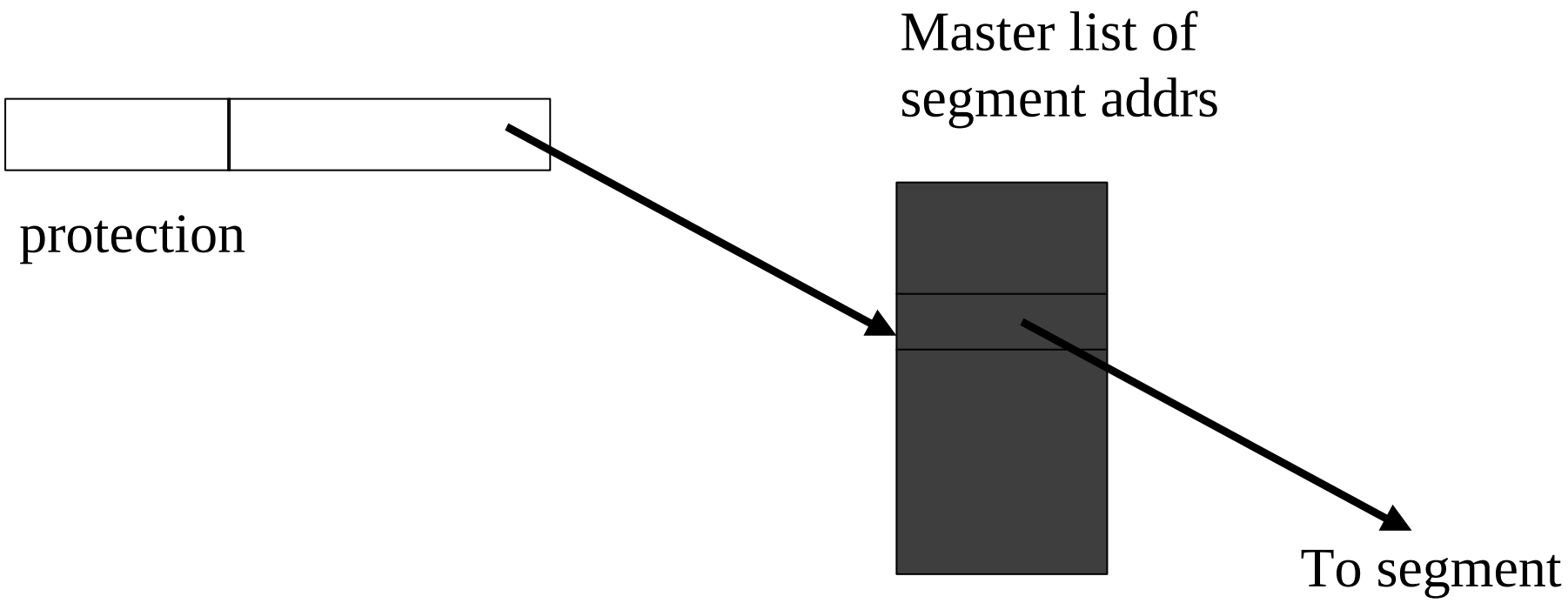
Adding protection information to STEs:



- Bad modularity -- STE now has two different *kinds* of information…

- so separate them! . . .

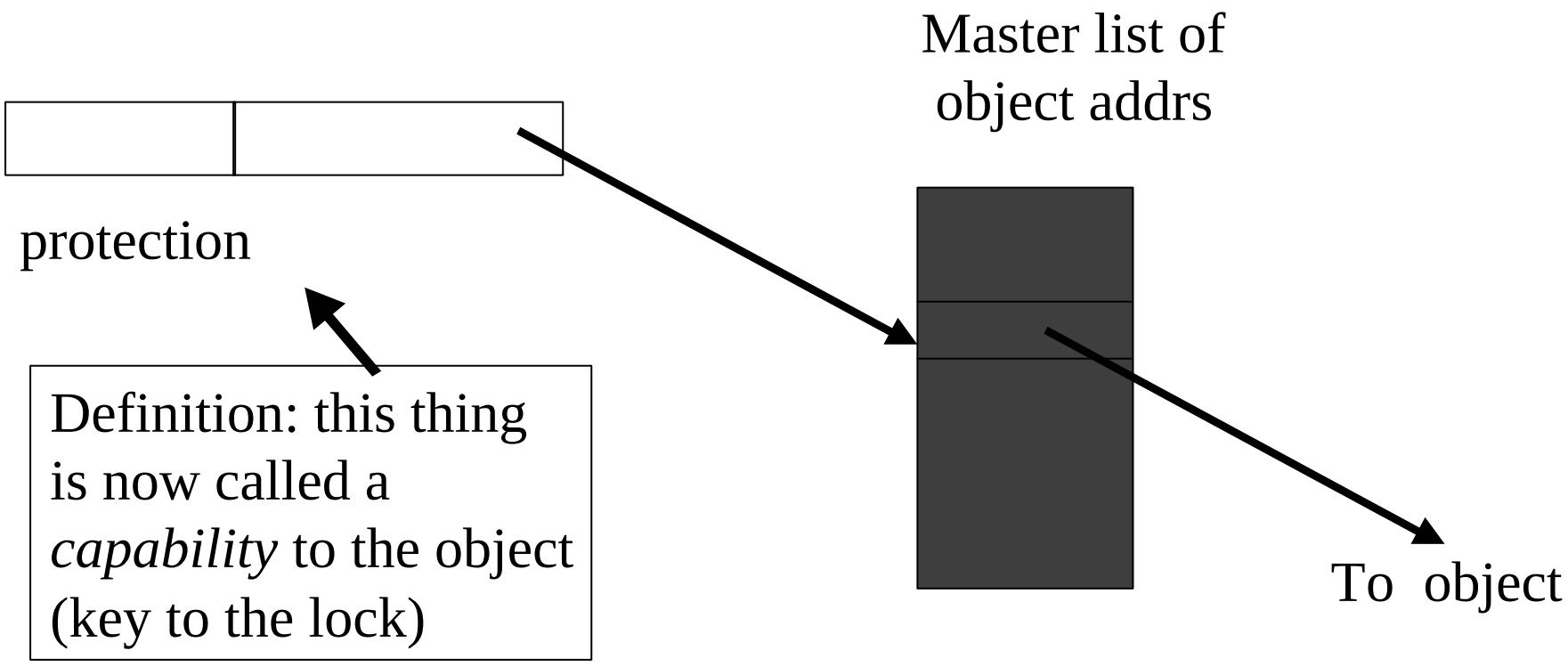
Address & protection info separated:



Generalize:

- Why just segments?
- Allow pointer to point to ***anything:***
  - subroutine
  - process
  - I/O device . . .

capability


Master list of
object addrs
protection
Definition: this thing
is now called a
capability to the object
(key to the lock)
To object

More ideas

- Allow a user program to load and store capabilities:
  - Load Cap
  - Store Cap instructions
- Preventing forgery??

- Provide special *capability registers* where active caps must be held
- LC, SC only work for special memory areas
- another process can give you a cap by moving it (what rules??)
- caps are encrypted to prevent forgeries
- see: The CAP-1 Capability Machine, R. Needham et al, circa 1980

PH Break
Virtual Memory & the MIPS chip

- You learn:

- Section 7.4 (Virtual Memory)