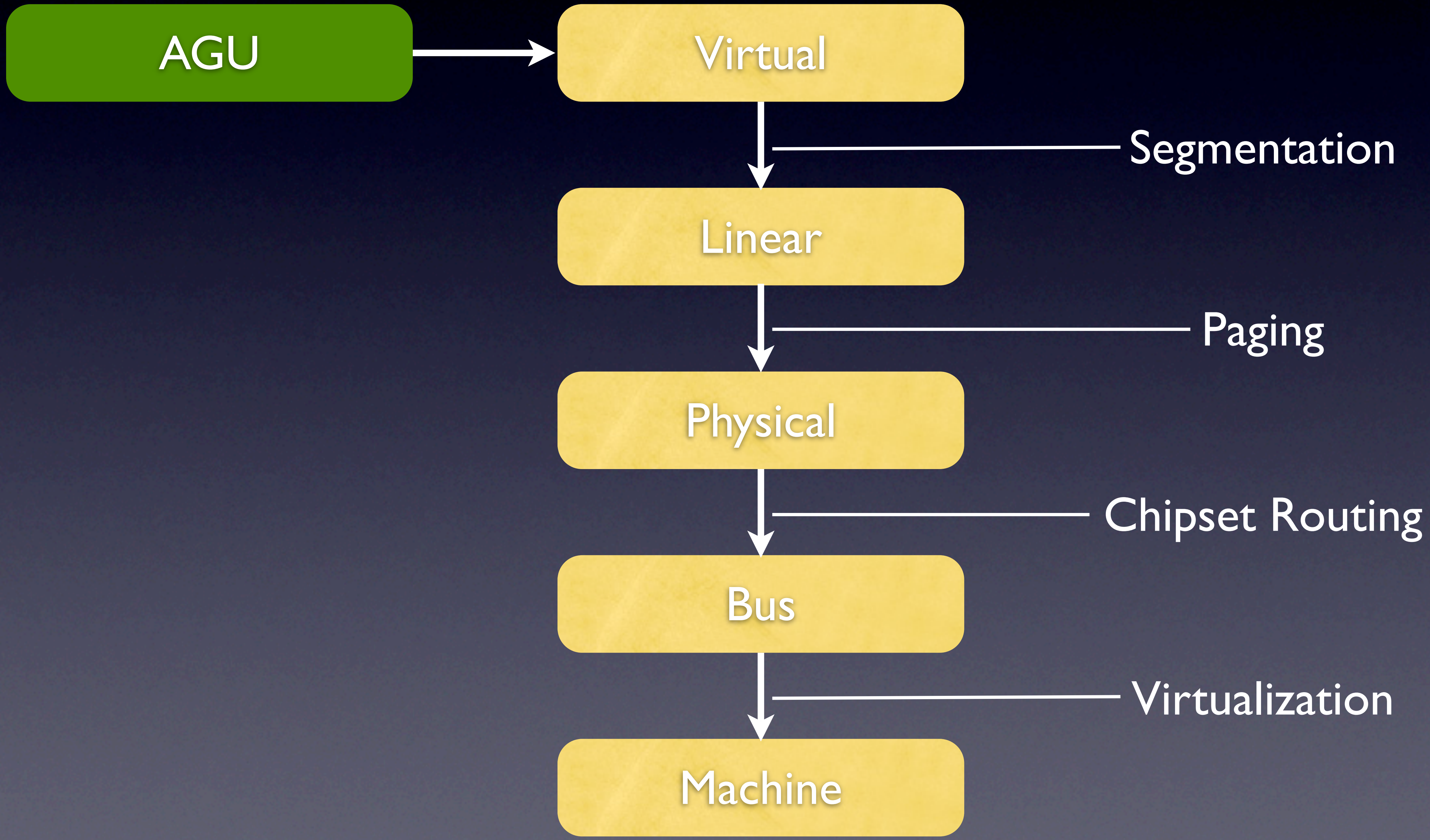
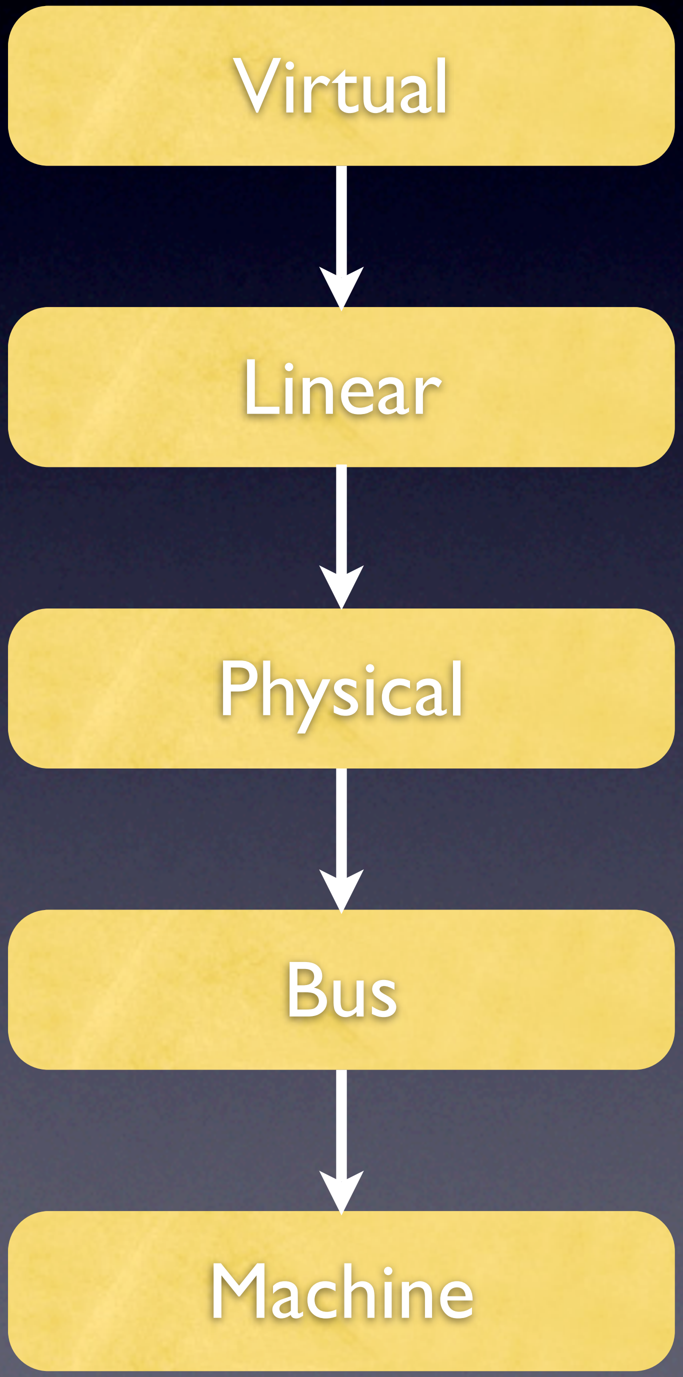
Layers of Memory

Jeffrey Sheldon

Layers of Memory



Terminology



VPN, VA

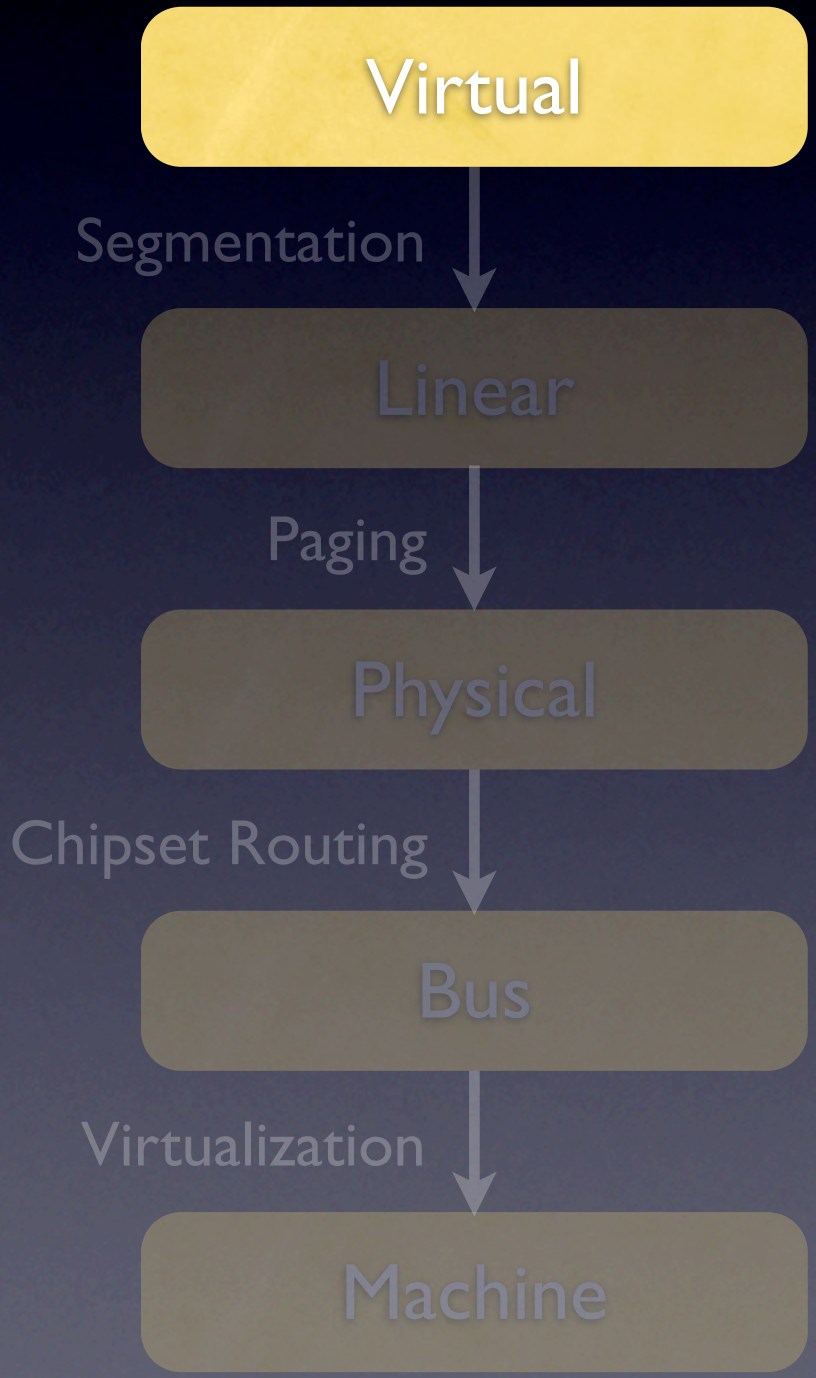
LPN, LA

PPN, PA

BPN, BA

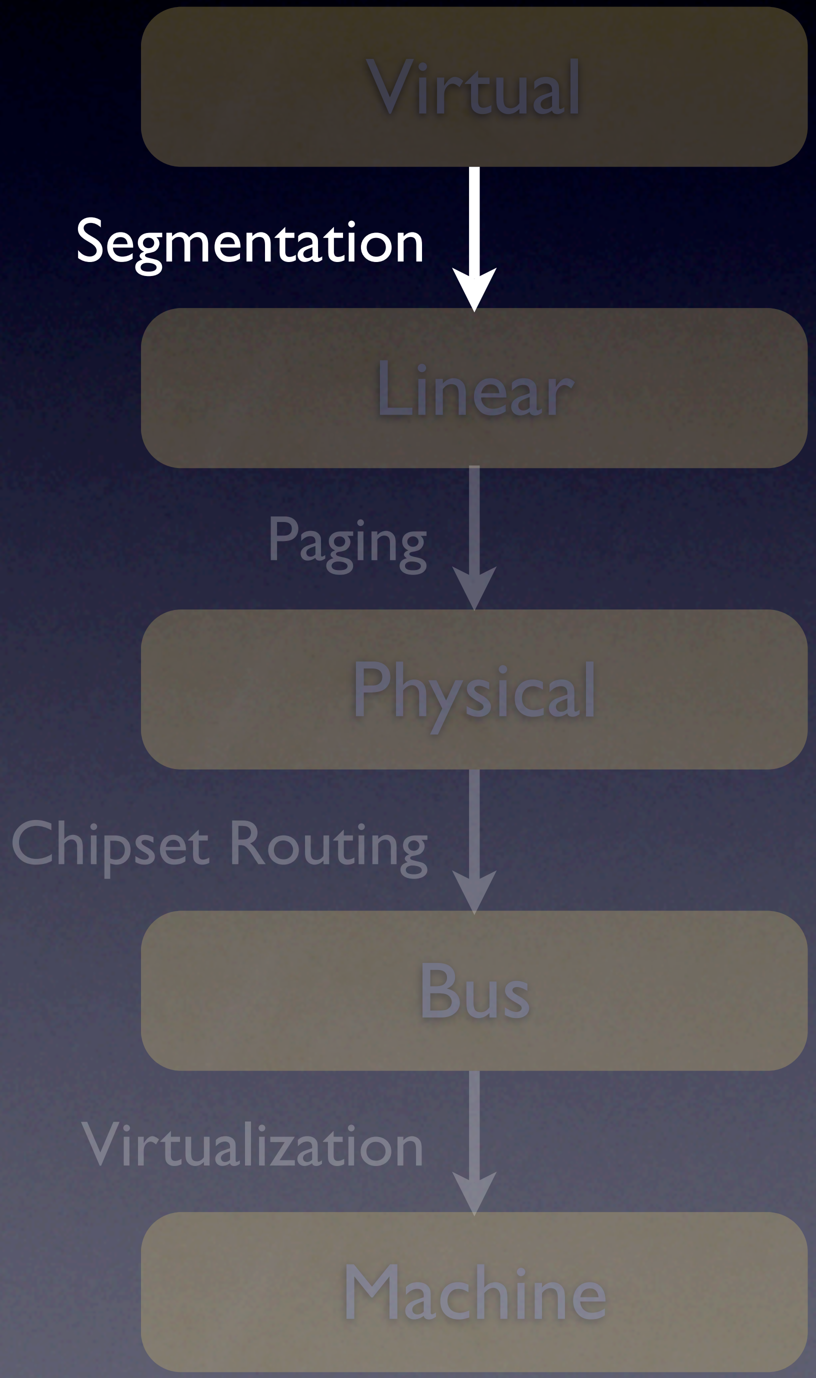
MPN, MA

Virtual Addresses



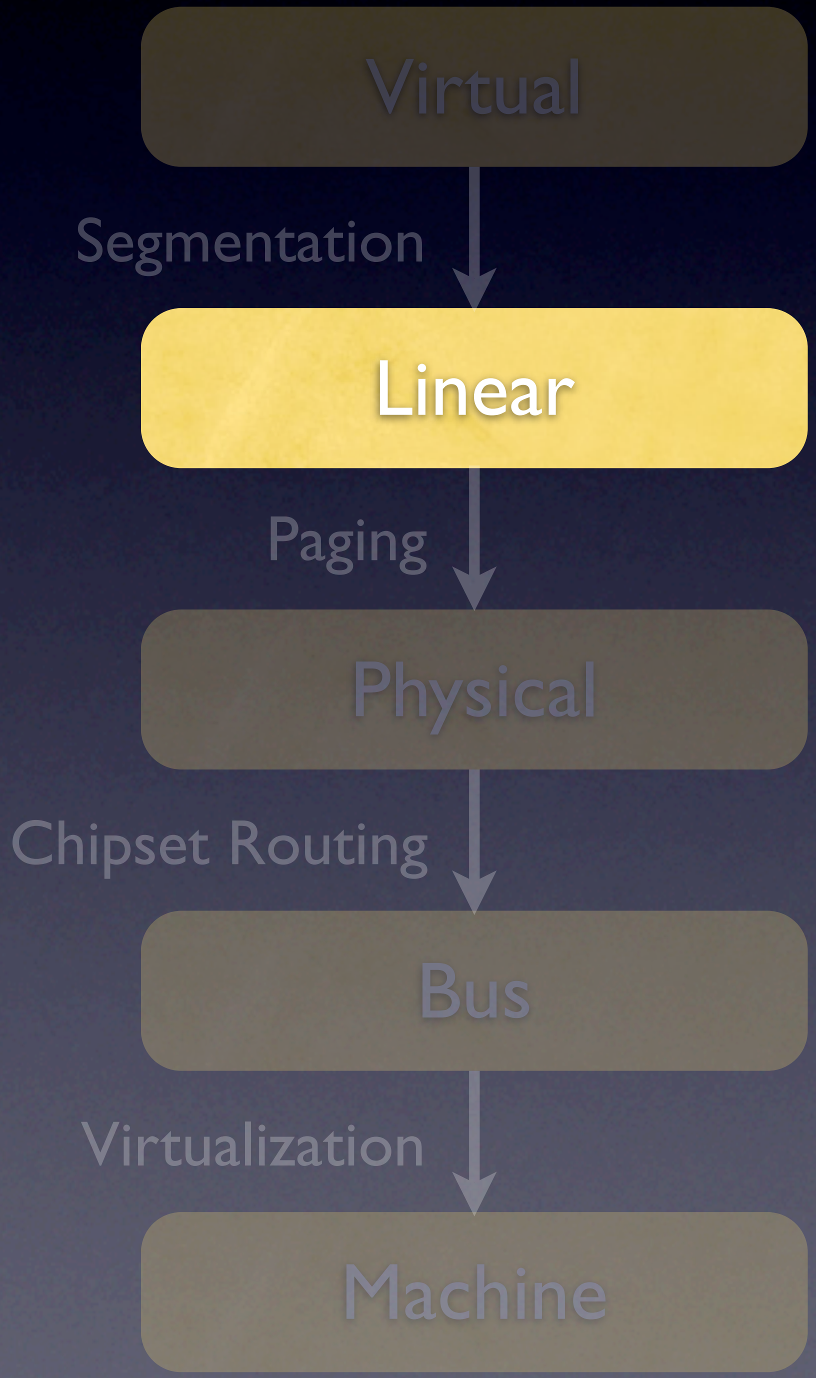
- Virtual addresses contain a segment and an address
- Six segments: cs, ds, es, ss, fs, gs
- Virtual addresses are 64-bit

Segmentation



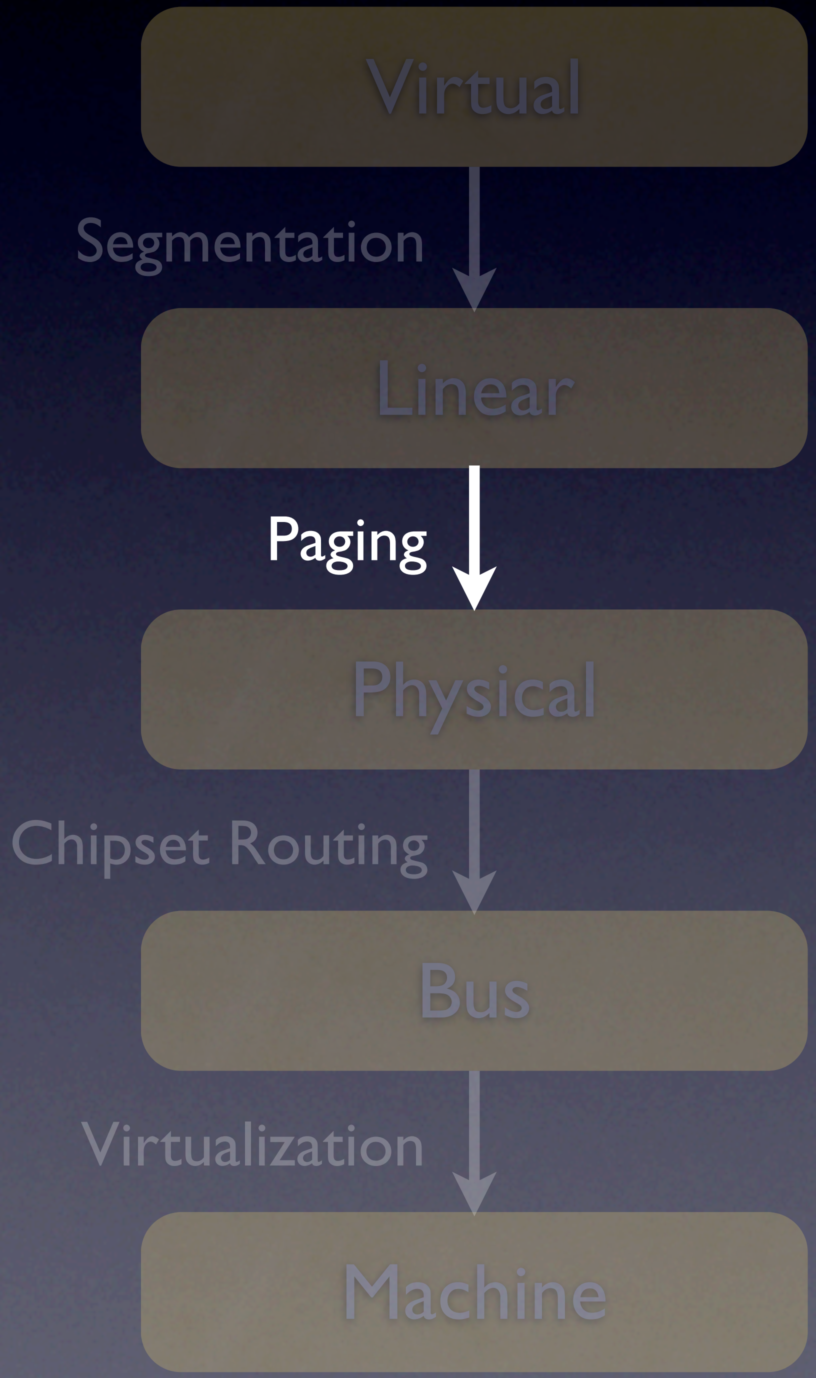
- May compare the virtual address against a limit
- Verifies the access against certain permissions
- May adds a base to the virtual address to create a linear address

Linear Addresses



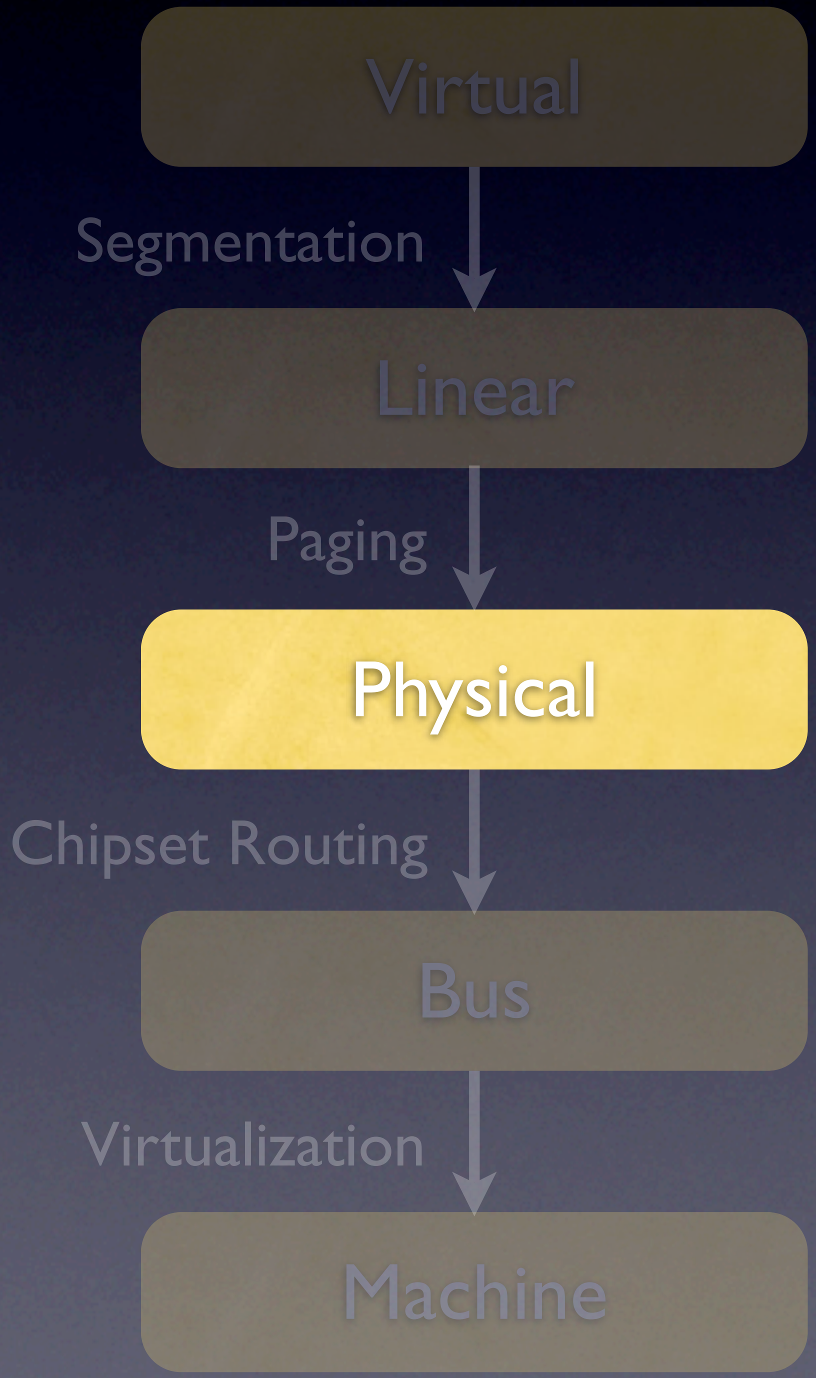
- In legacy mode linear address space is a flat 32-bit space
- In “64-bit mode” it is 48 bit address space
- Represented as a 64-bit number by sign-extending the top most bit

Paging



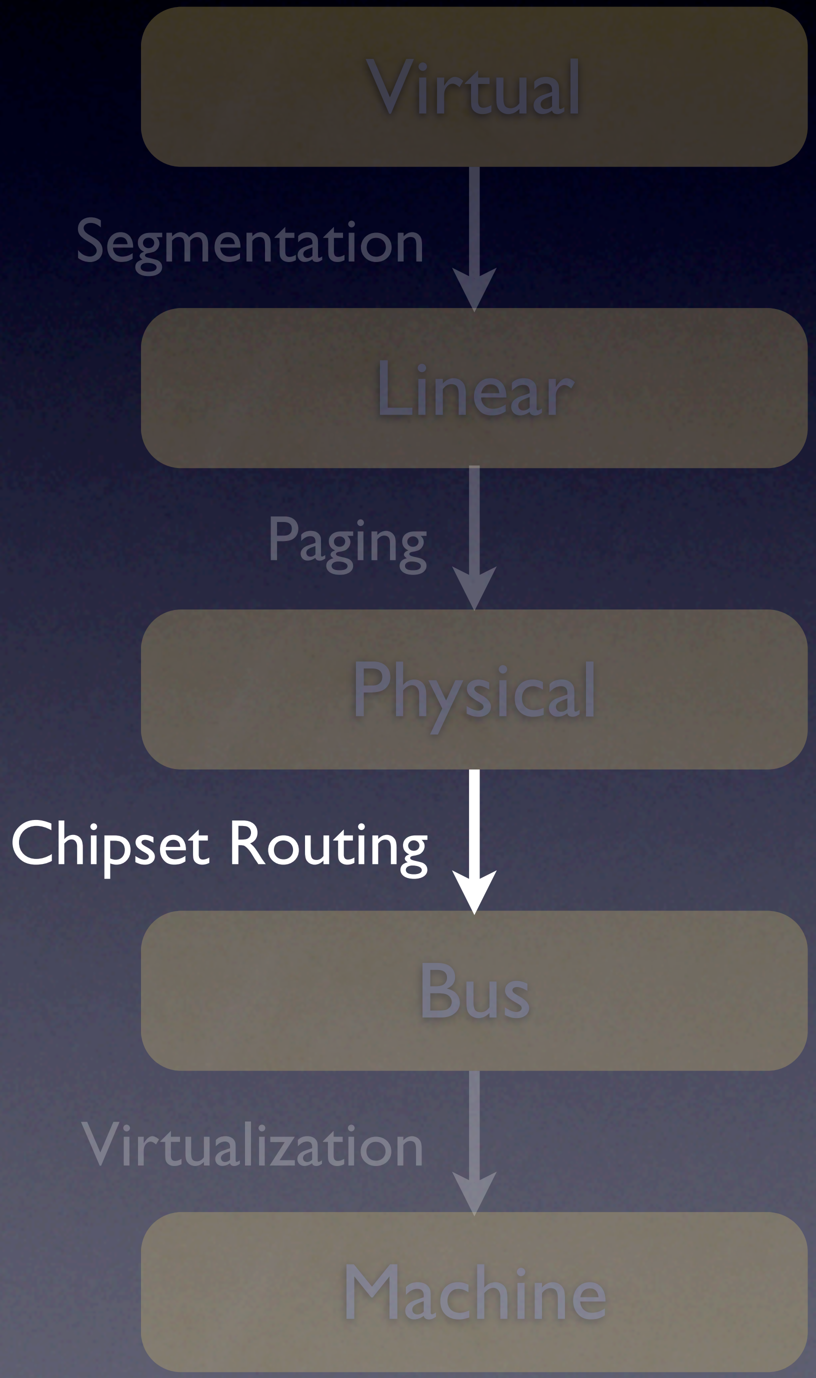
- Feed linear address into 2, 3 or 4 level page tables
- TLB is filled by hardware walking page tables

Physical Addresses



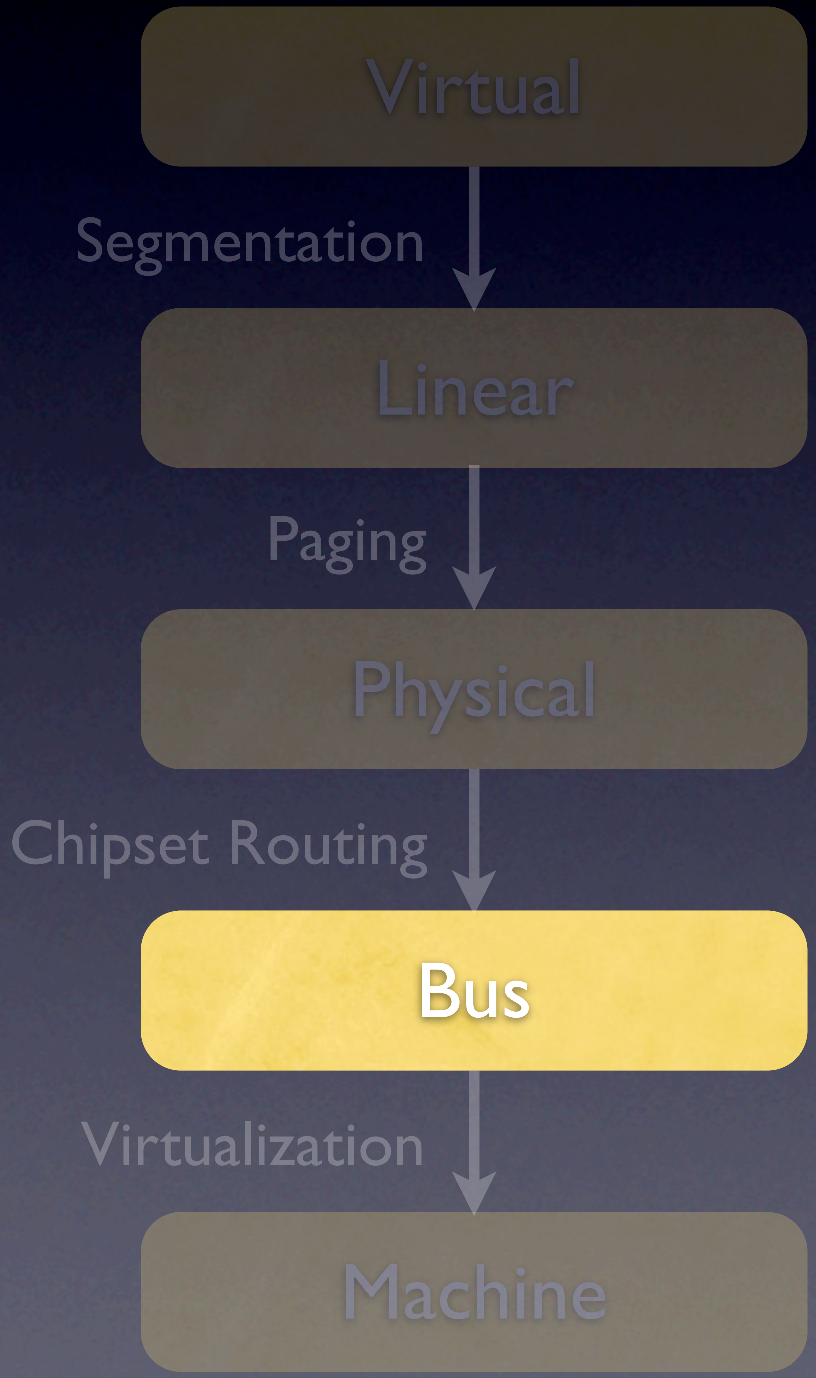
- Physical address space is 40-bits (as of HWv7)
- Previously we had a 36-bit physical address space
- Some modern CPUs are 52-bit
- Available even in legacy mode

Chipset Routing



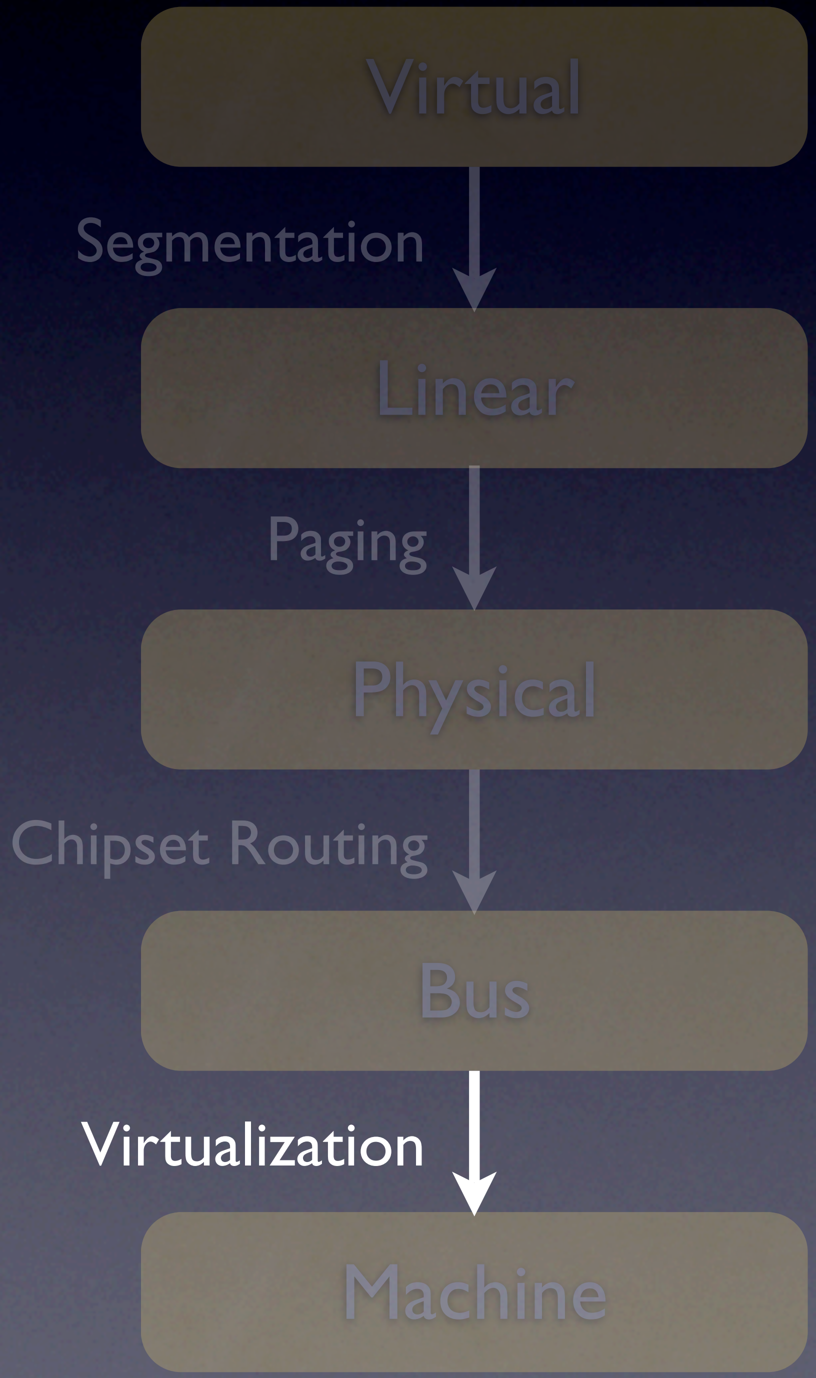
- Really a function of the CPU, memory controller, chipset, PCI-bus, etc.
- Routes a physical address coming from the CPU to the device that backs that address

Bus Addresses



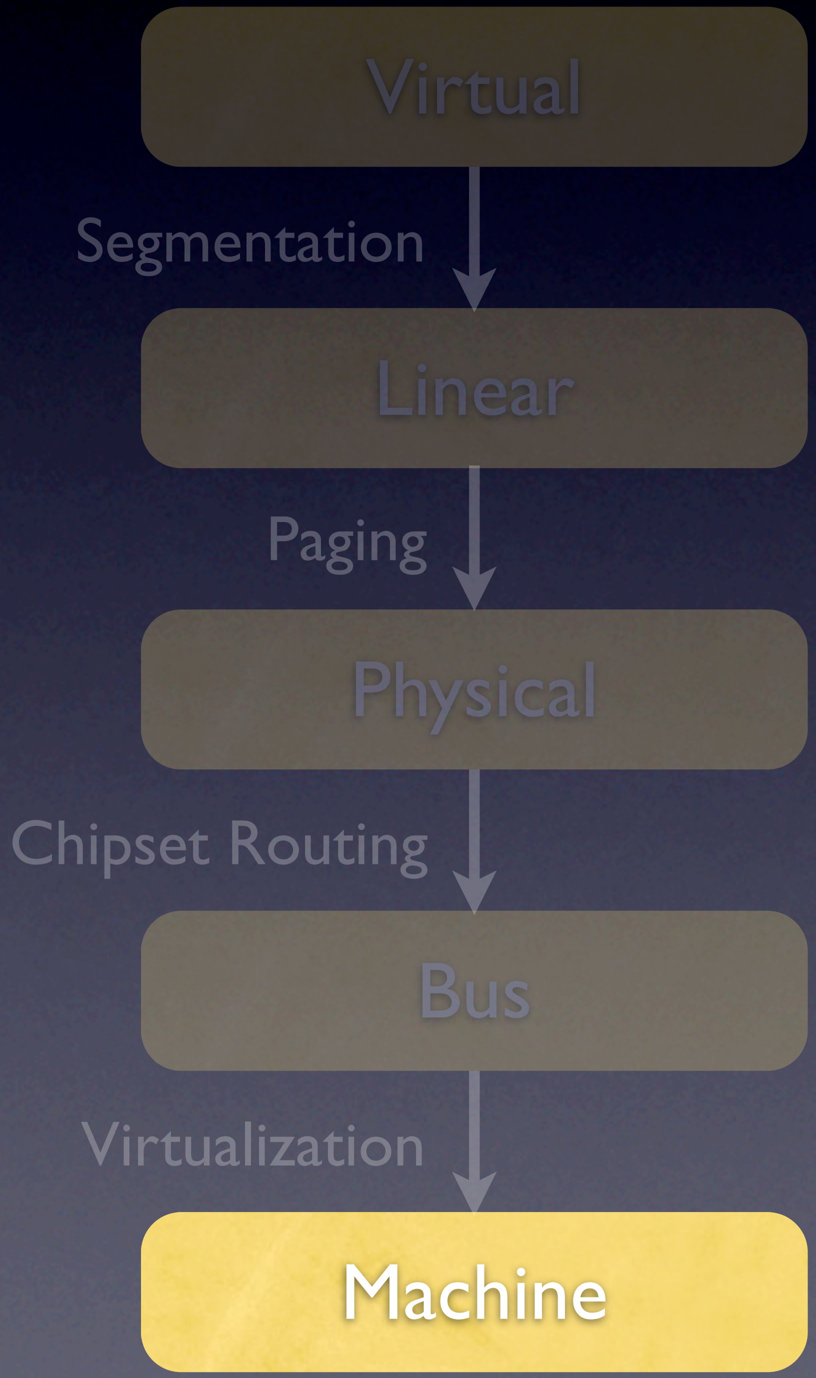
- VMware formalization of what real hardware does
- Logically its a tuple of device ID and offset in device
- BPNs are stored as uint32s

Virtualization



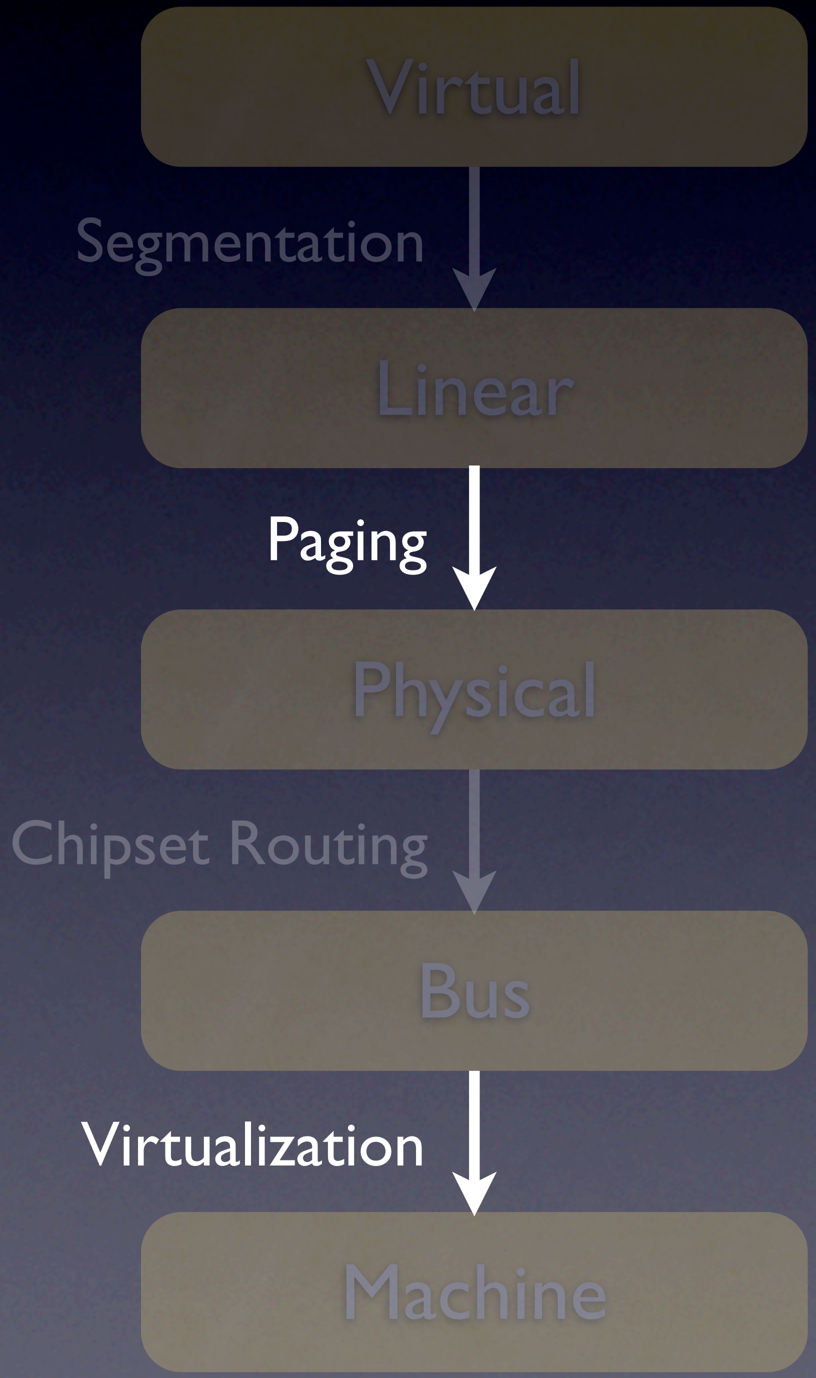
- Layer of translation with no analogue on a physical computer
- Allows us to overcommit, swap, page share, balloon, compress, share, do whatever

Machine Address



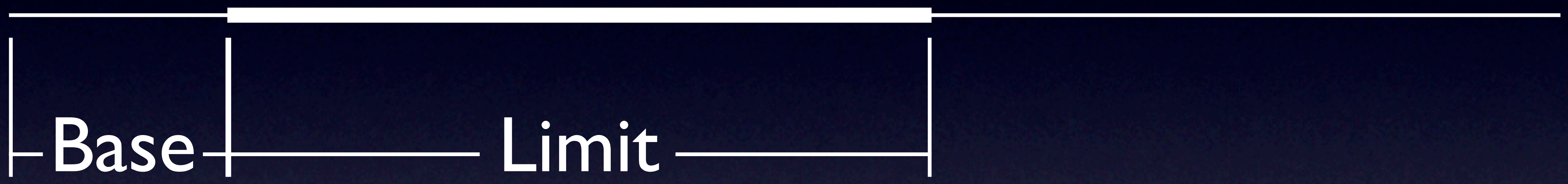
- Currently ~44 bits
- Corresponds to a physical address of the host

Layers of Memory



- For more about converting from linear to physical, see Ravi’s talk on the MMU
- For more about converting from bus addresses to machine addresses, see Alex’s talk on Memory Management

Segmentation



- Each segment has a base, limit, and some properties

Segmentation

- Six segment registers: cs, ds, es, ss, fs, gs
- Each register is 16-bits and can be accessed (read or written) with special move instructions
- Each segment register also contains the base, limit, and properties for that segment, but this state is “hidden”

CPU State

cs	sel_{cs}	$base_{cs}$	$limit_{cs}$	$prop_{cs}$
ds	sel_{ds}	$base_{ds}$	$limit_{ds}$	$prop_{ds}$
es	sel_{es}	$base_{es}$	$limit_{es}$	$prop_{es}$
ss	sel_{ss}	$base_{ss}$	$limit_{ss}$	$prop_{ss}$
fs	sel_{fs}	$base_{fs}$	$limit_{fs}$	$prop_{fs}$
gs	sel_{gs}	$base_{gs}$	$limit_{gs}$	$prop_{gs}$

CPU State

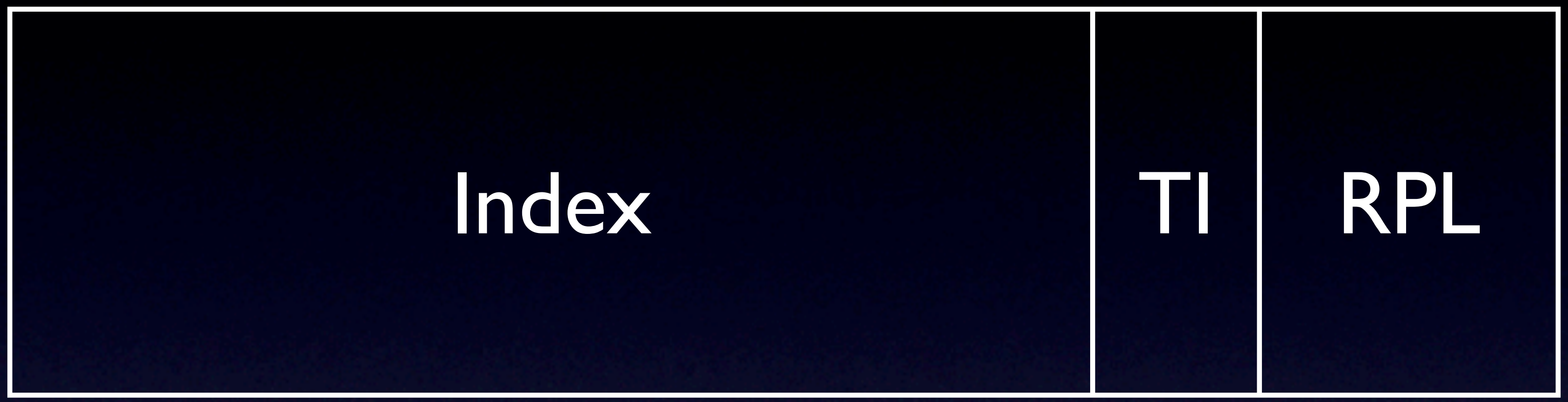
Hidden State

cs	sel_{cs}	$base_{cs}$	$limit_{cs}$	$prop_{cs}$
ds	sel_{ds}	$base_{ds}$	$limit_{ds}$	$prop_{ds}$
es	sel_{es}	$base_{es}$	$limit_{es}$	$prop_{es}$
ss	sel_{ss}	$base_{ss}$	$limit_{ss}$	$prop_{ss}$
fs	sel_{fs}	$base_{fs}$	$limit_{fs}$	$prop_{fs}$
gs	sel_{gs}	$base_{gs}$	$limit_{gs}$	$prop_{gs}$

Selectors

cs	sel_{cs}	base_{cs}	limit_{cs}	prop_{cs}
ds	sel_{ds}	base_{ds}	limit_{ds}	prop_{ds}
es	sel_{es}	base_{es}	limit_{es}	prop_{es}
ss	sel_{ss}	base_{ss}	limit_{ss}	prop_{ss}
fs	sel_{fs}	base_{fs}	limit_{fs}	prop_{fs}
gs	sel_{gs}	base_{gs}	limit_{gs}	prop_{gs}

Selectors

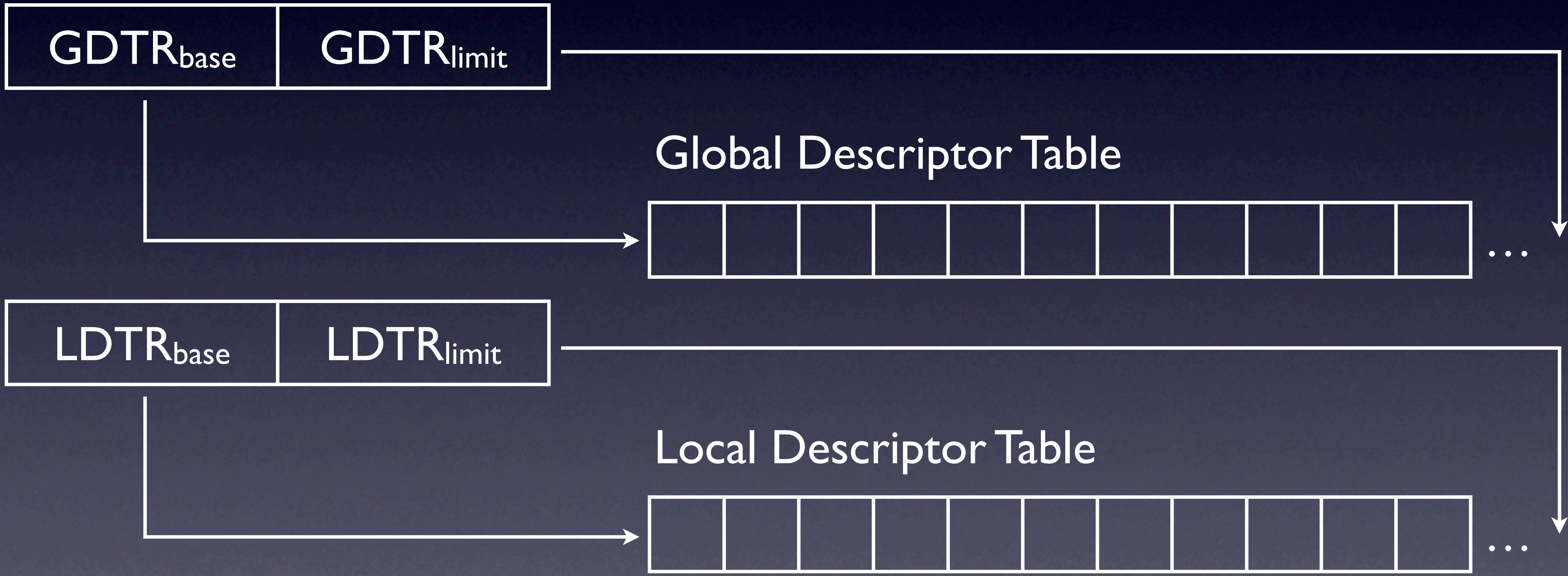


- RPL is two bits represented "requested privilege level"
- Selector used to index into descriptor tables
- TI bit selects table while index selects offset into table

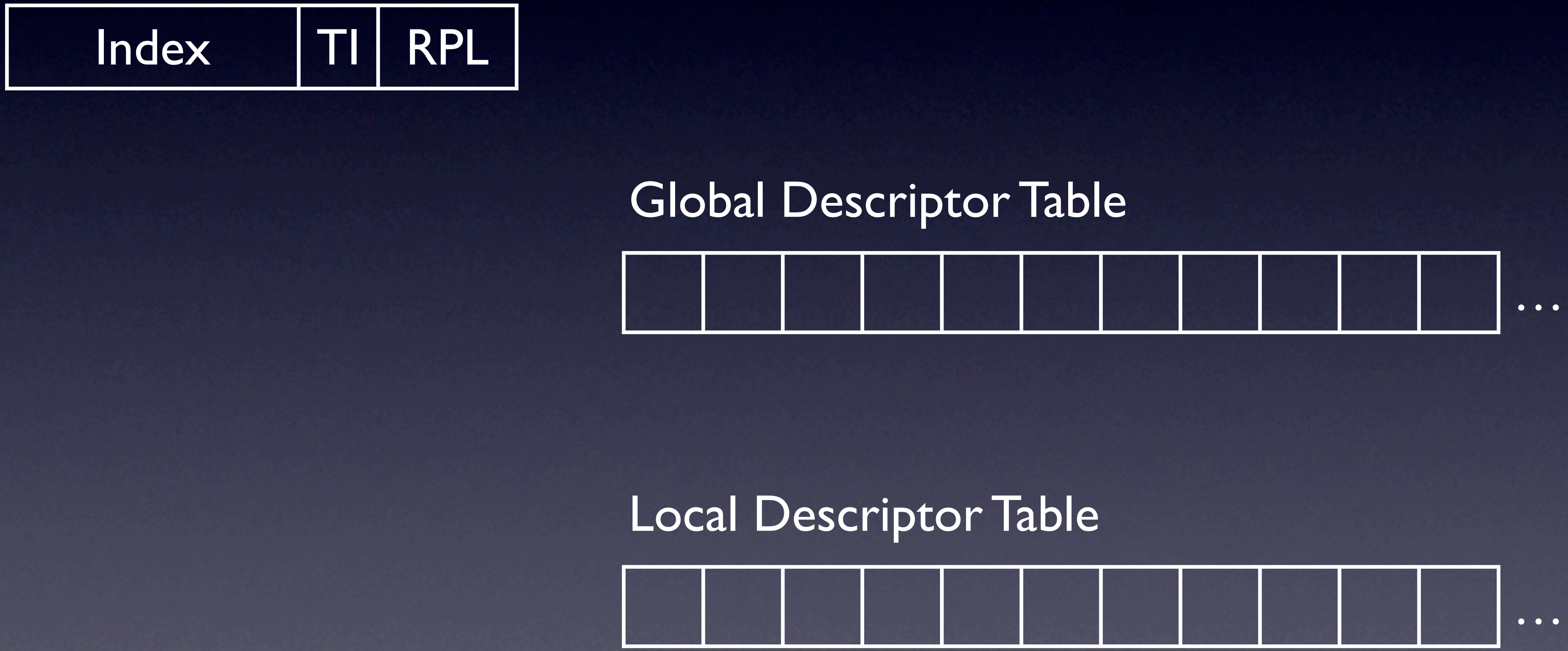
Descriptor Tables

- Two descriptor tables: global and local
- Contains an array of 8 byte descriptors
- Each table contains a maximum of 8192 descriptors, but also has an adjustable limit

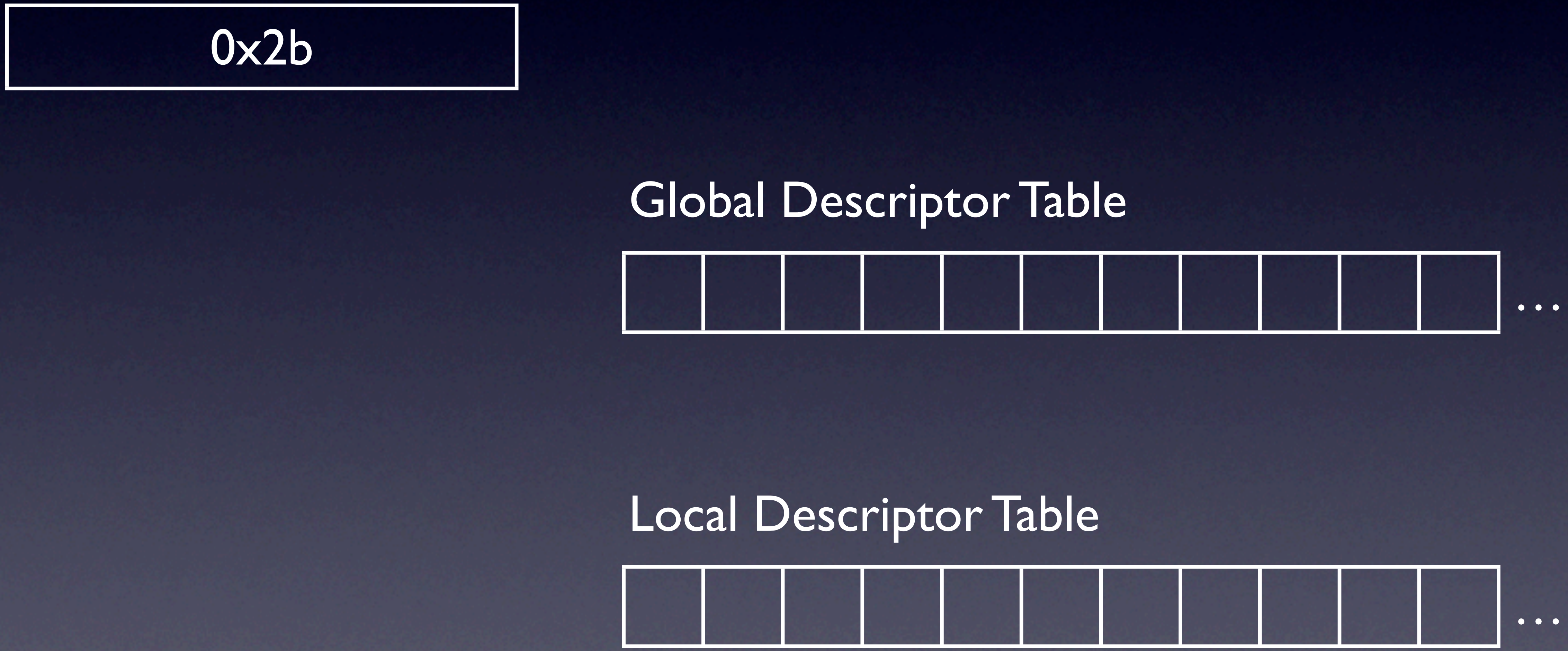
Descriptor Tables



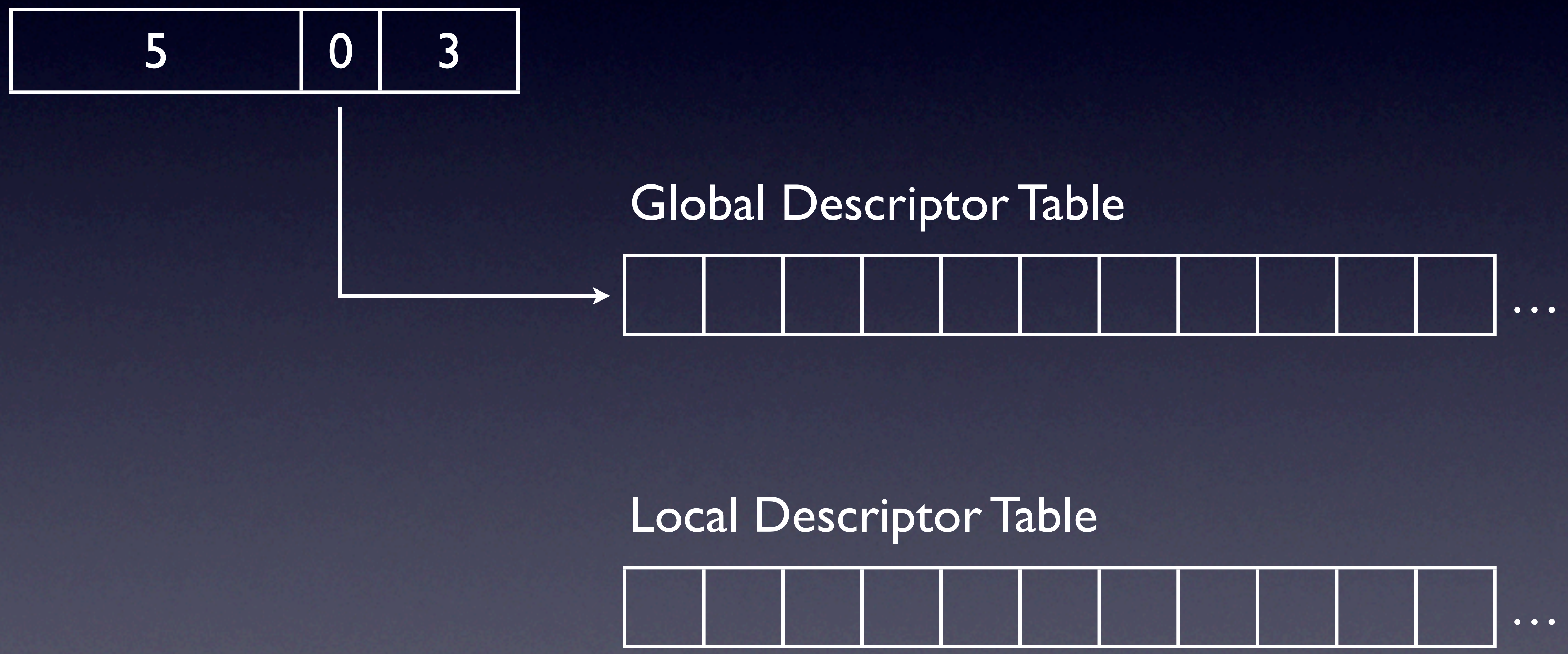
Selectors


Index
TI
RPL
Global Descriptor Table
...
Local Descriptor Table
...

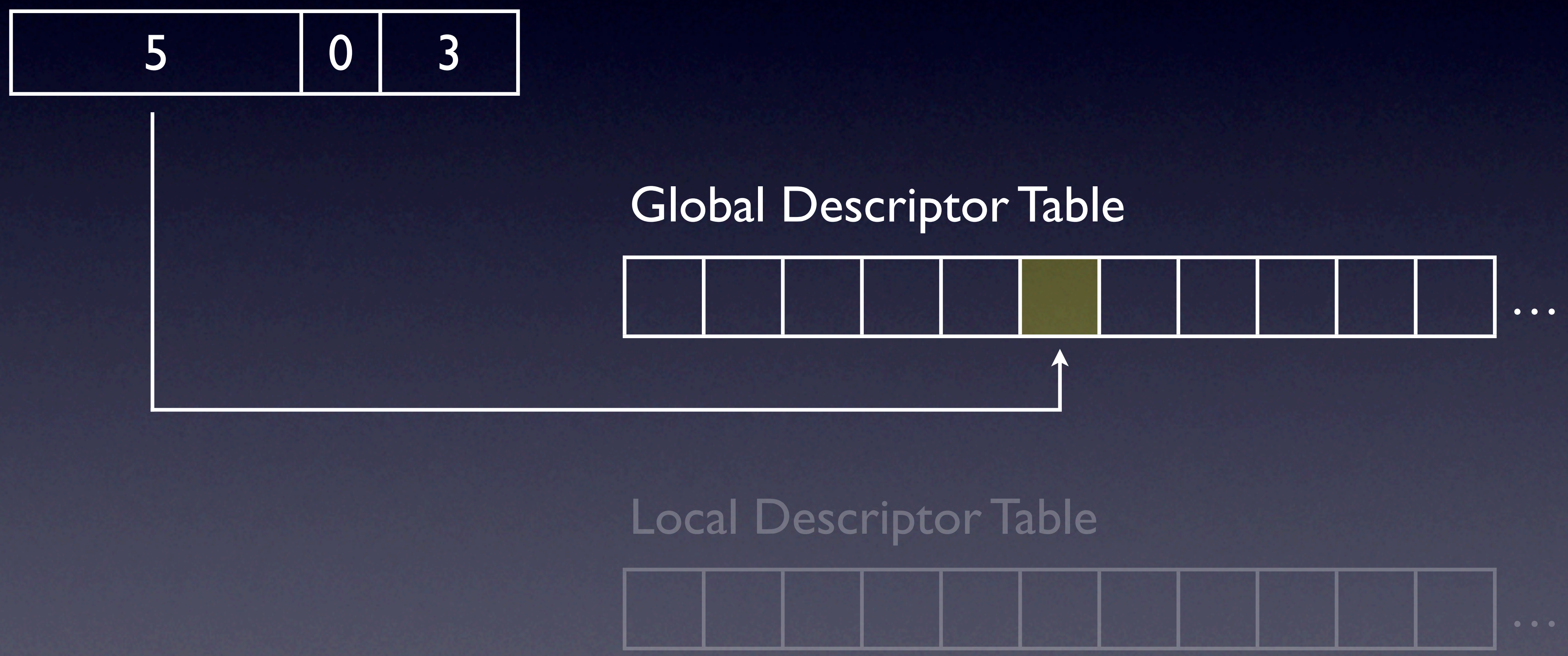
Selectors



Selectors



Selectors



Descriptor

31 24	23	22		20	19 16	15	14 13	12	11	10	9	8	7 0	
Base Address 31–24	G	D / B		A V L	Limit 19–16	P	DPL	S	C / D	C	R	A	Base Address 23–16	+4
Base Address 15–0						Limit 15–0								+0

Loading a segment

- Move a selector from a register to a segment register
- CPU implicitly using selector to index into a descriptor table and loads a descriptor from memory and stores into the segment’s hidden state

Modifying a Descriptor

- What if descriptor in memory changes?
- Hidden segment state remains the same until segment is reloaded

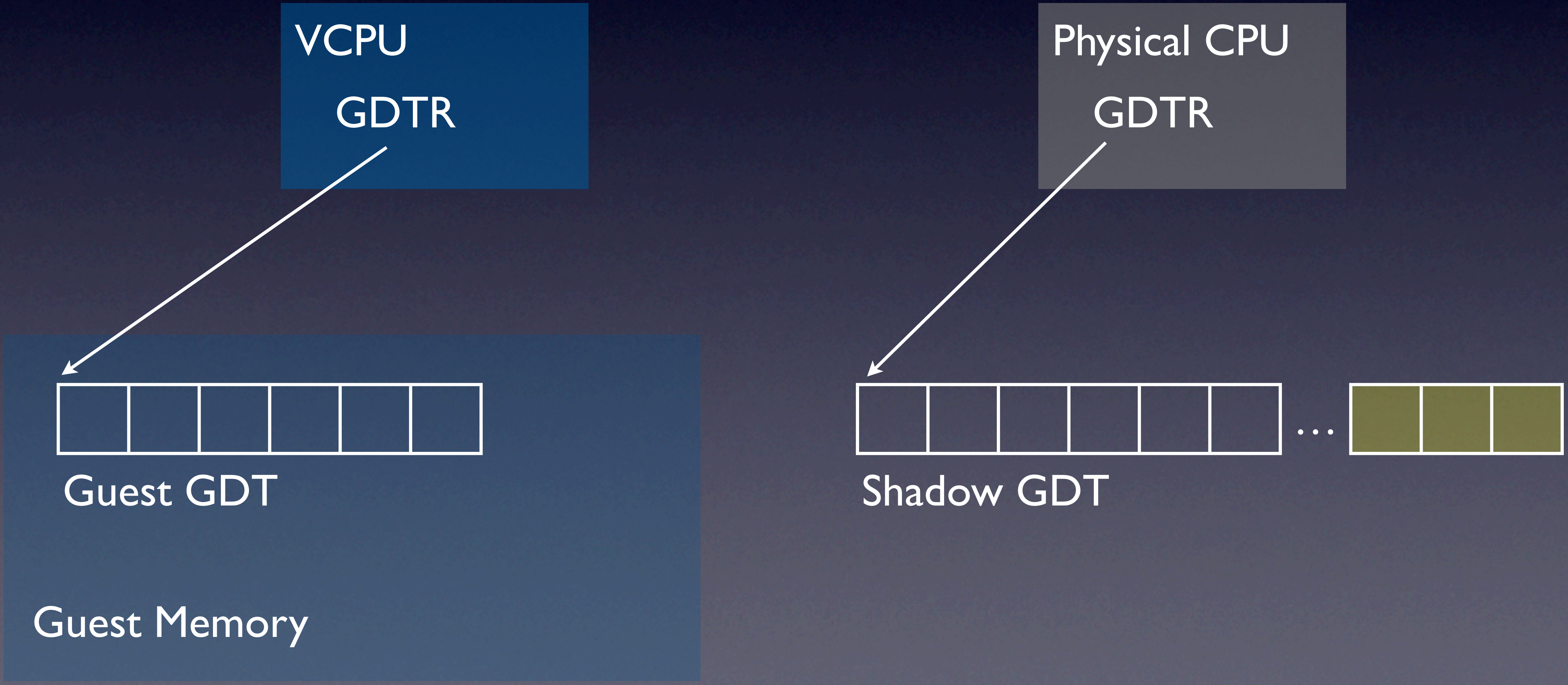
Virtualizing Segmentation

- When using HV?
    - Nothing much to do — hardware maintains separate monitor and guest state

Virtualizing Segmentation

- When using BT? Two goals:
  - Emulate the guest's use of segmentation hardware
  - Protect the monitor when in BT mode (segment truncation)

Shadow Descriptor Tables



Shadow Contents

- Does not truncate limits
- Set the A bit eagerly (replay)
- DPL 0 → DPL 1
- Hide L bit when guest in legacy mode
- Call gate / task gate / TSS

Virtualizing Segmentation

- Translate segment loads into software implementation
- Maintain an explicit copy of hidden state in the VCPU
- Track when the hidden state matches the “in memory” version — called clean

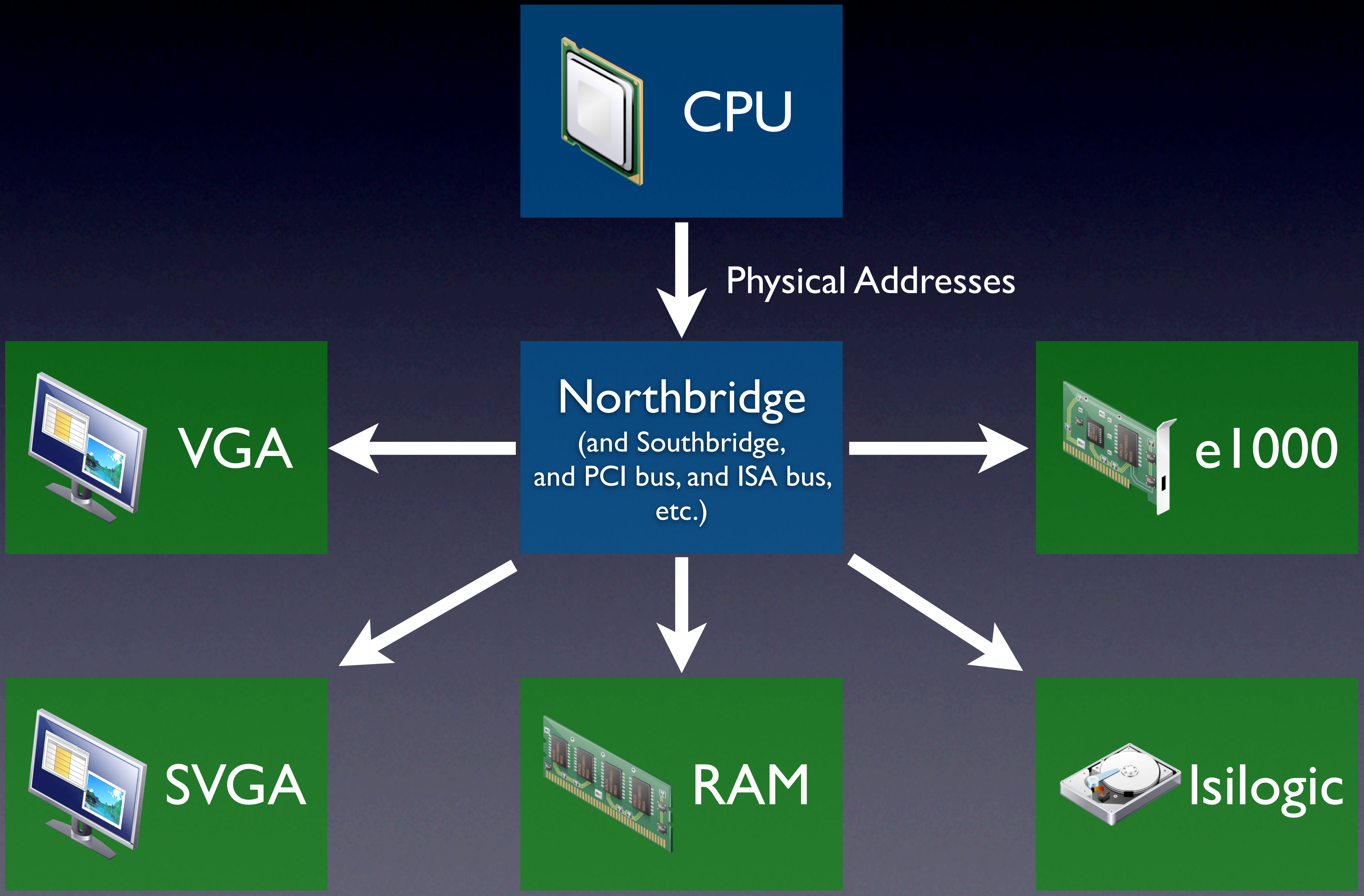
Virtualizing Segmentation

- When all segments are clean, direct exec is allowed
- When in direct exec, guest loads segments and hardware is redirected to shadow
- Update hidden state in VCPU when exiting direct exec

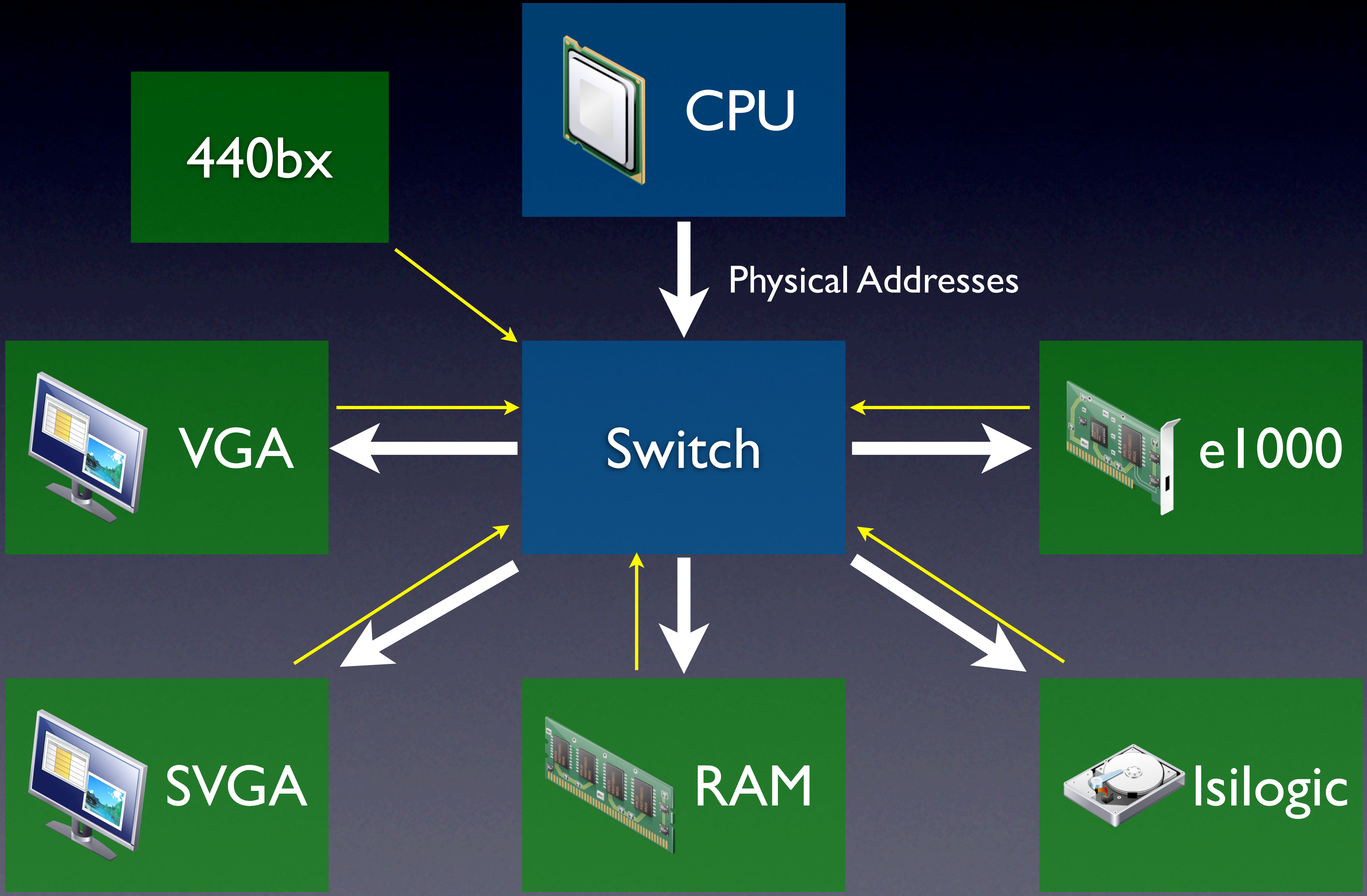
Segment Truncation

- Not used while in direct exec
- Load special truncated monitor segments while in BT
- Perform all guest segmentation (base/limit checks) in software

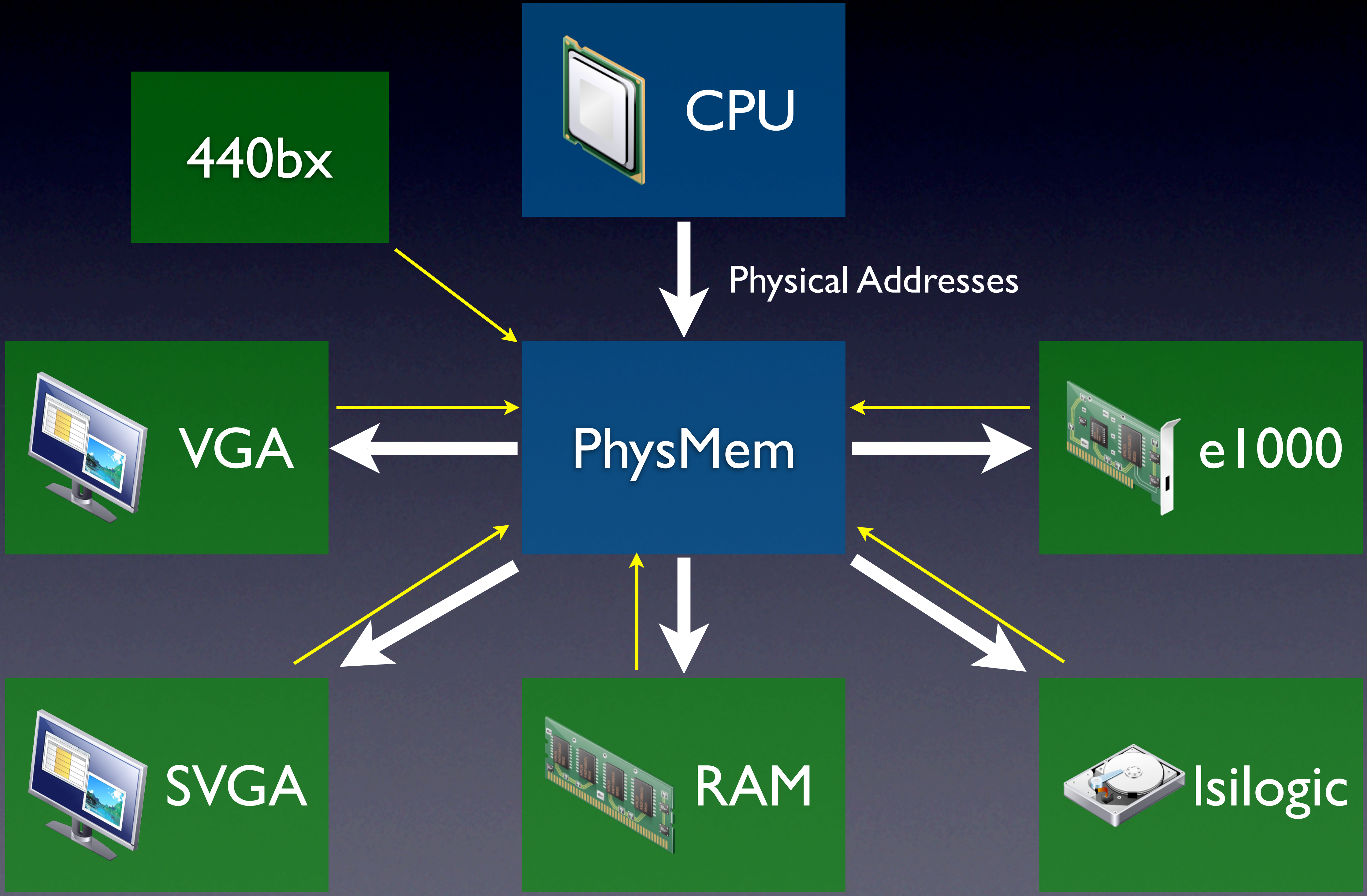
Oversimplified Computer



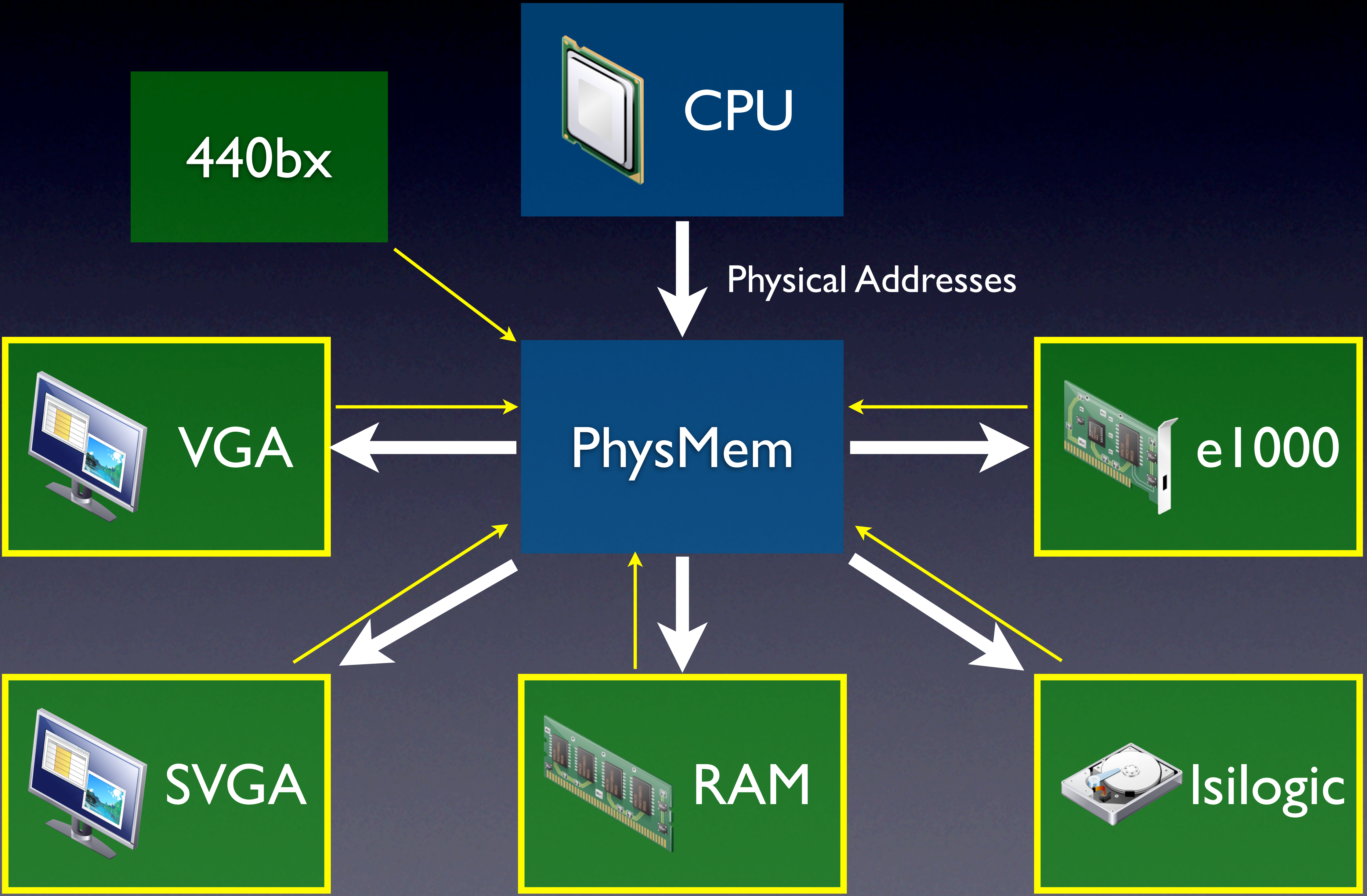
Split Policy and Mechanism



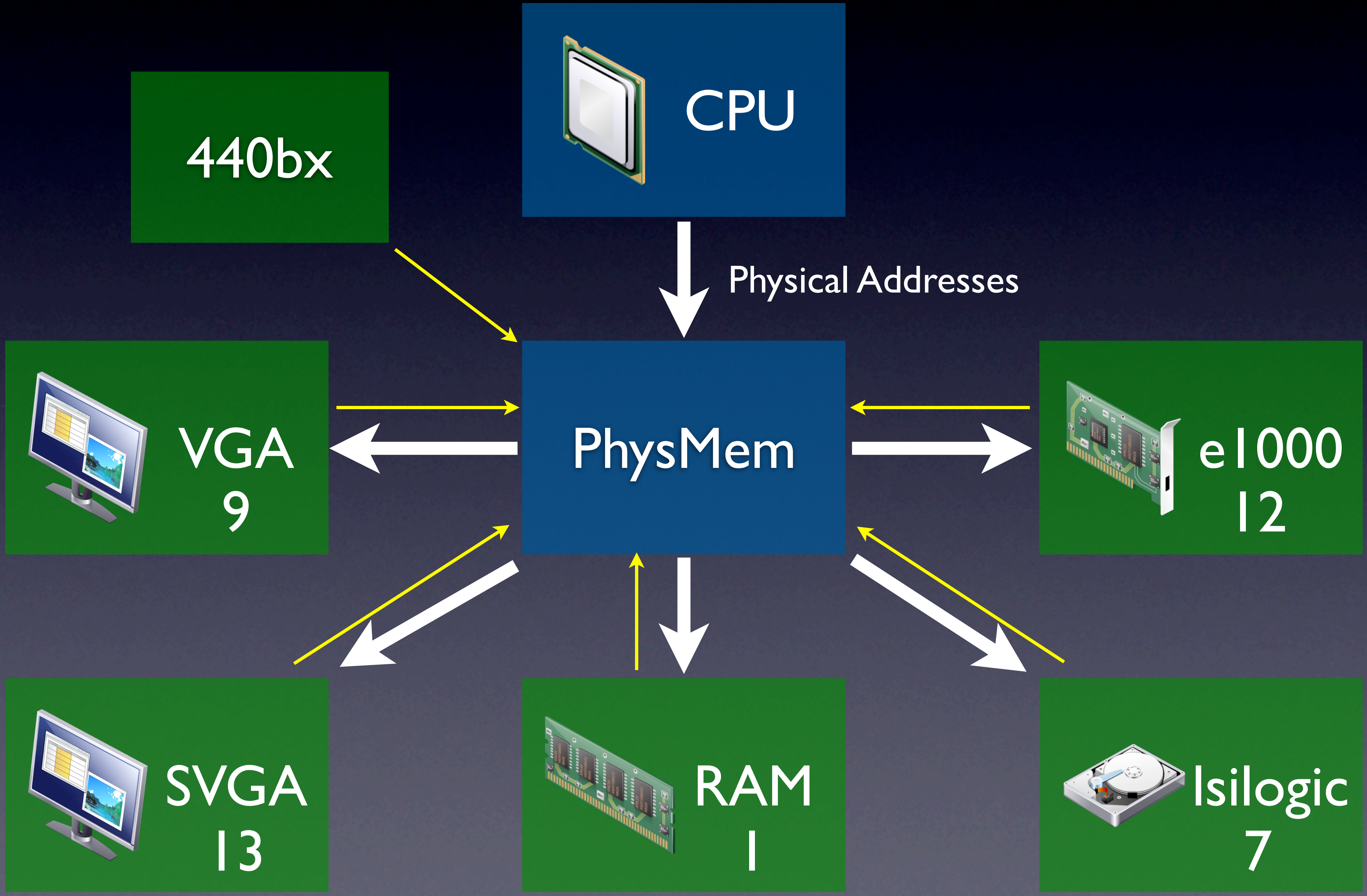
PhysMem


440bx
CPU
Physical Addresses
VGA
PhysMem
e1000
SVGA
RAM
lsilogic

Regions


440bx
CPU
Physical Addresses
PhysMem
VGA
e1000
SVGA
RAM
lsilogic

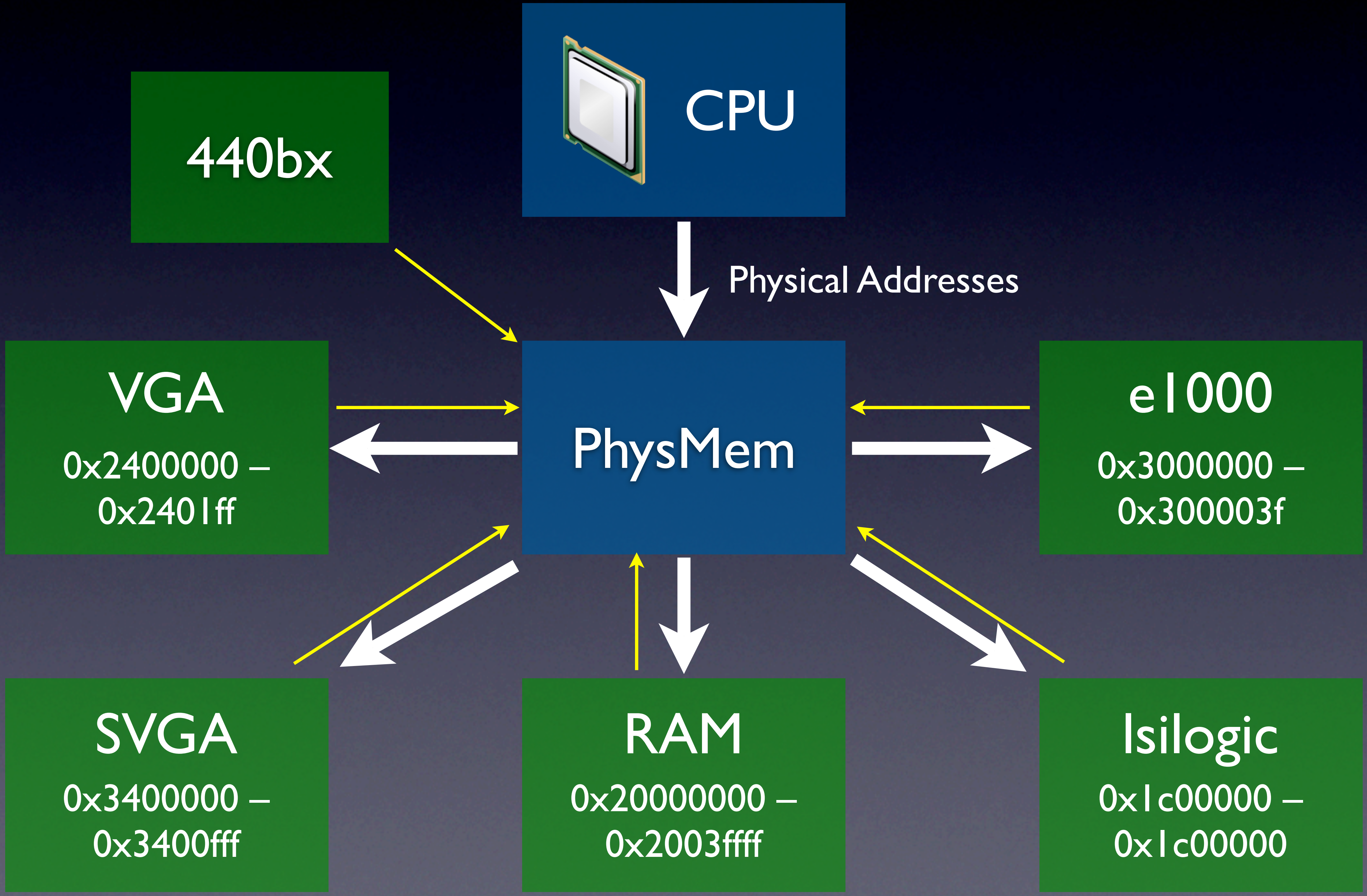
Assign IDs to each Region



Regions

- Each region has a size
- Contains contiguous sequence of BPNs
- VGA is 32 pages, so VGA memory is represented by BPNs 0x2400000 – 0x240200

BPNs


440bx
CPU
Physical Addresses
VGA
0x2400000 –
0x2401ff
PhysMem
e1000
0x3000000 –
0x300003f
SVGA
0x3400000 –
0x3400fff
RAM
0x20000000 –
0x2003ffff
lsilogic
0x1c00000 –
0x1c00000

Regions

- Each region specifies where the memory comes from
  - Memory can be allocated by physmem/ busmem
  - Memory can provided by the device
  - Memory can be “MMIO” with callbacks either in VMM or VMX

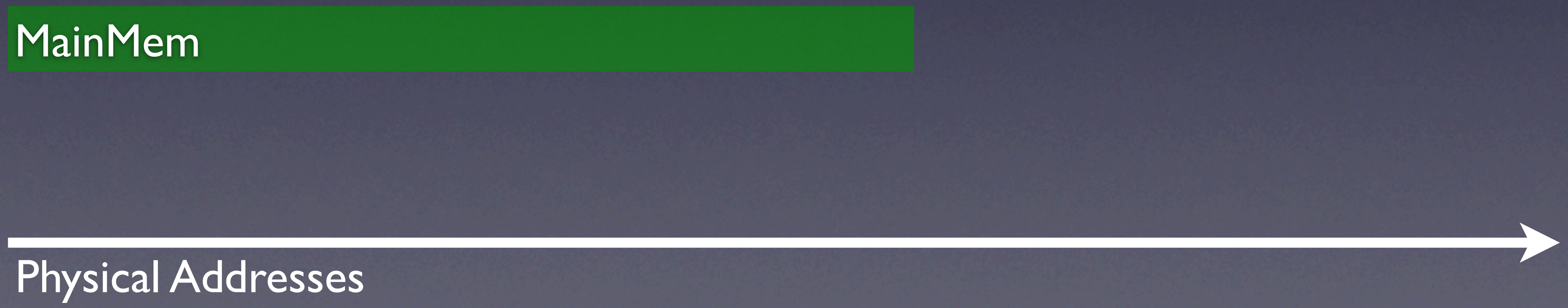
Mappings

- Regions do not have a position in the guest physical address space
- Instead we have PhysMem mappings

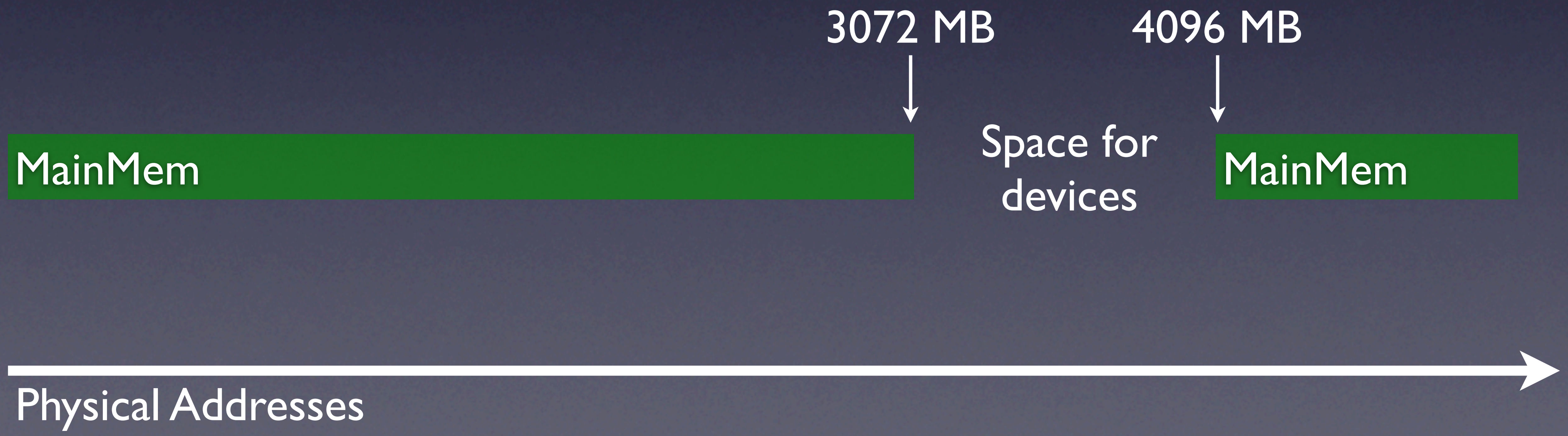
Mappings

- A PhysMem mapping ties a portion of a region to a sequence of PPNs
- Multiple mappings may point at the same region, or even the same BPNs in that region
- Mappings can be per-VCPU
- Mappings have relative priorities

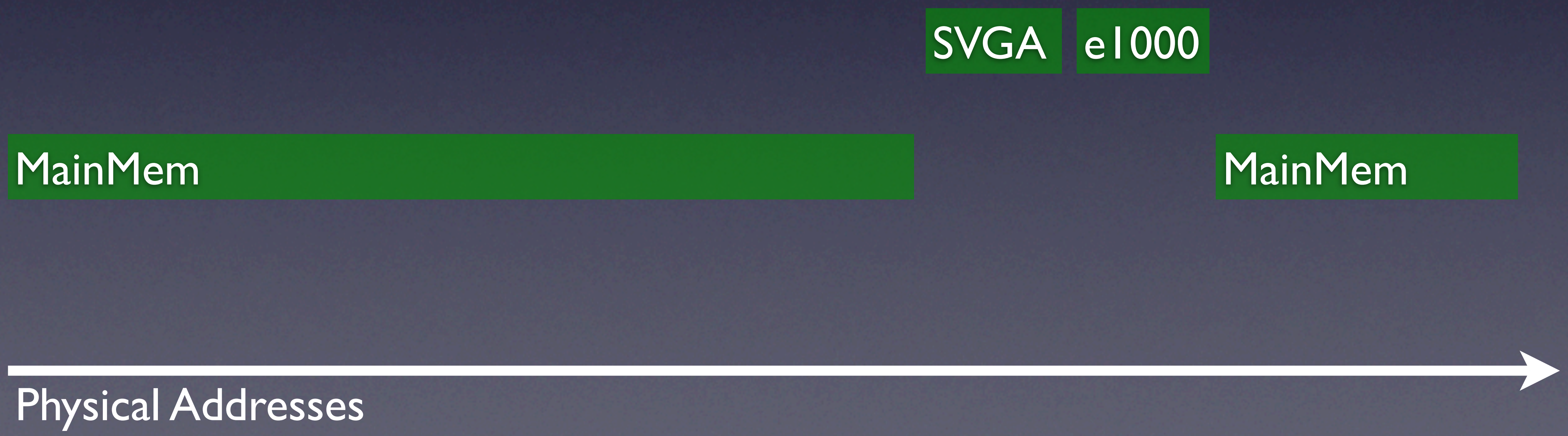
Mappings



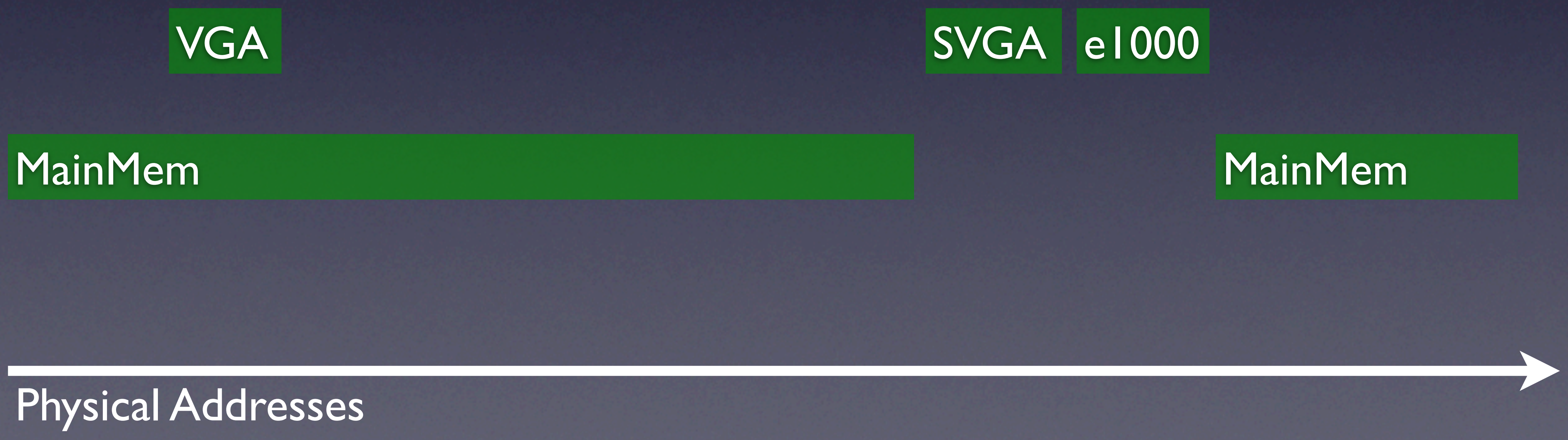
Mappings


3072 MB
4096 MB
MainMem
Space for devices
MainMem
Physical Addresses

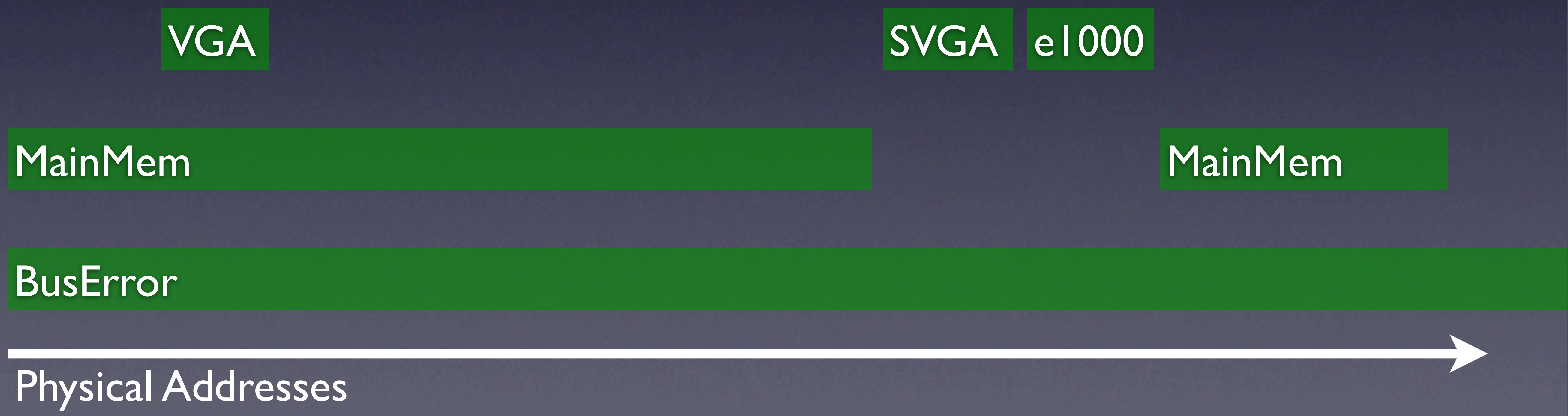
Mappings



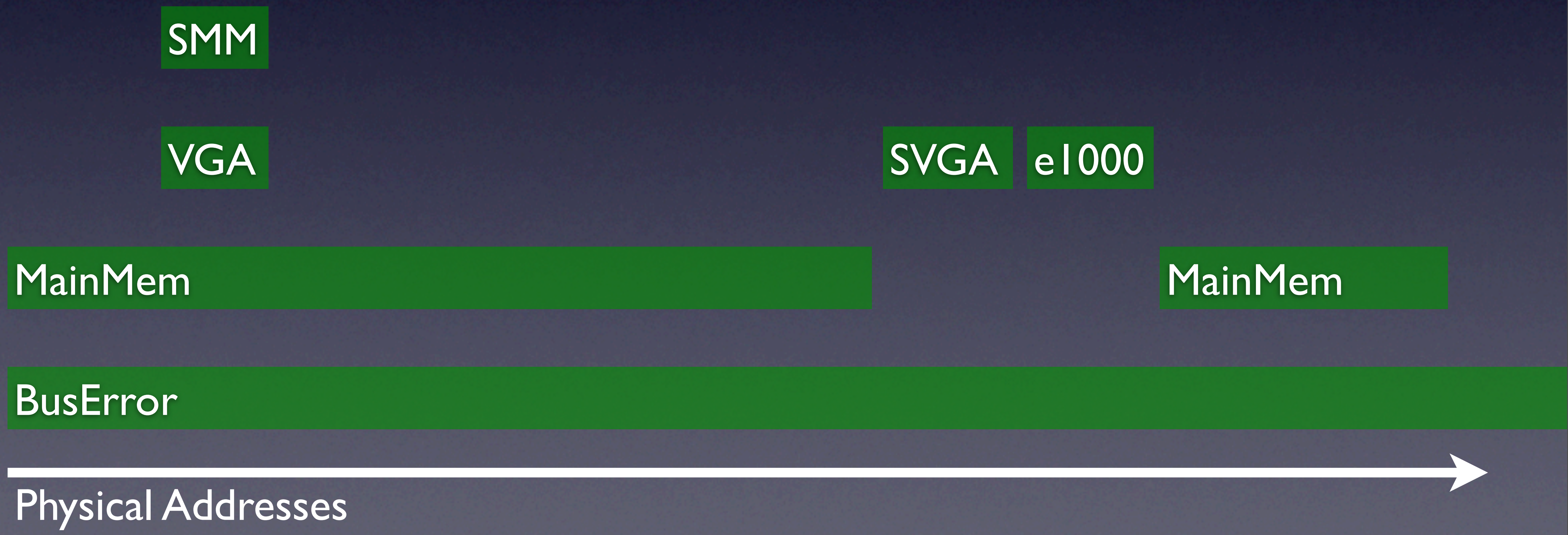
Mappings



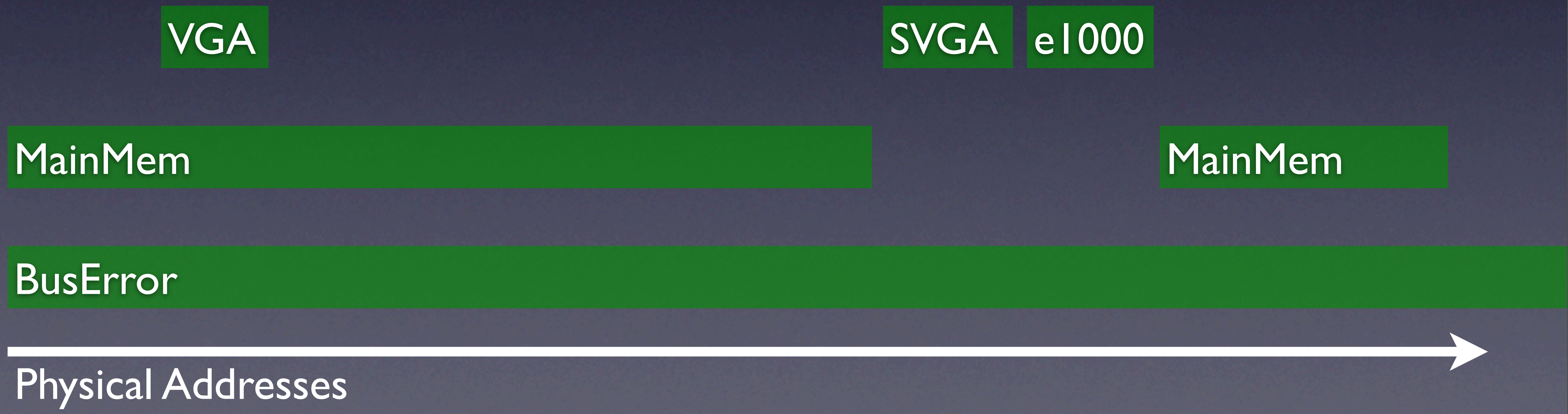
Mappings


VGA
SVGA
e1000
MainMem
MainMem
BusError
Physical Addresses

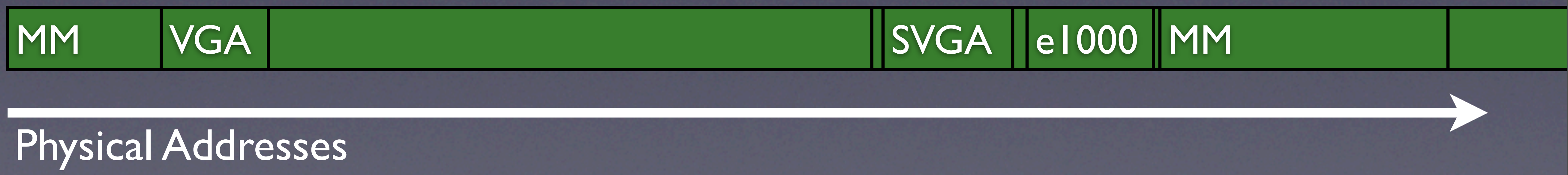
Mappings


SMM
VGA
SVGA
e1000
MainMem
MainMem
BusError
Physical Addresses

Mappings


VGA
SVGA
e1000
MainMem
MainMem
BusError
Physical Addresses

Lookups


MM
VGA
SVGA
e1000
MM
Physical Addresses