CS5460/6460: Operating Systems

Lecture 8: System init

Anton Burtsev
January, 2014

How old is the shepherd?

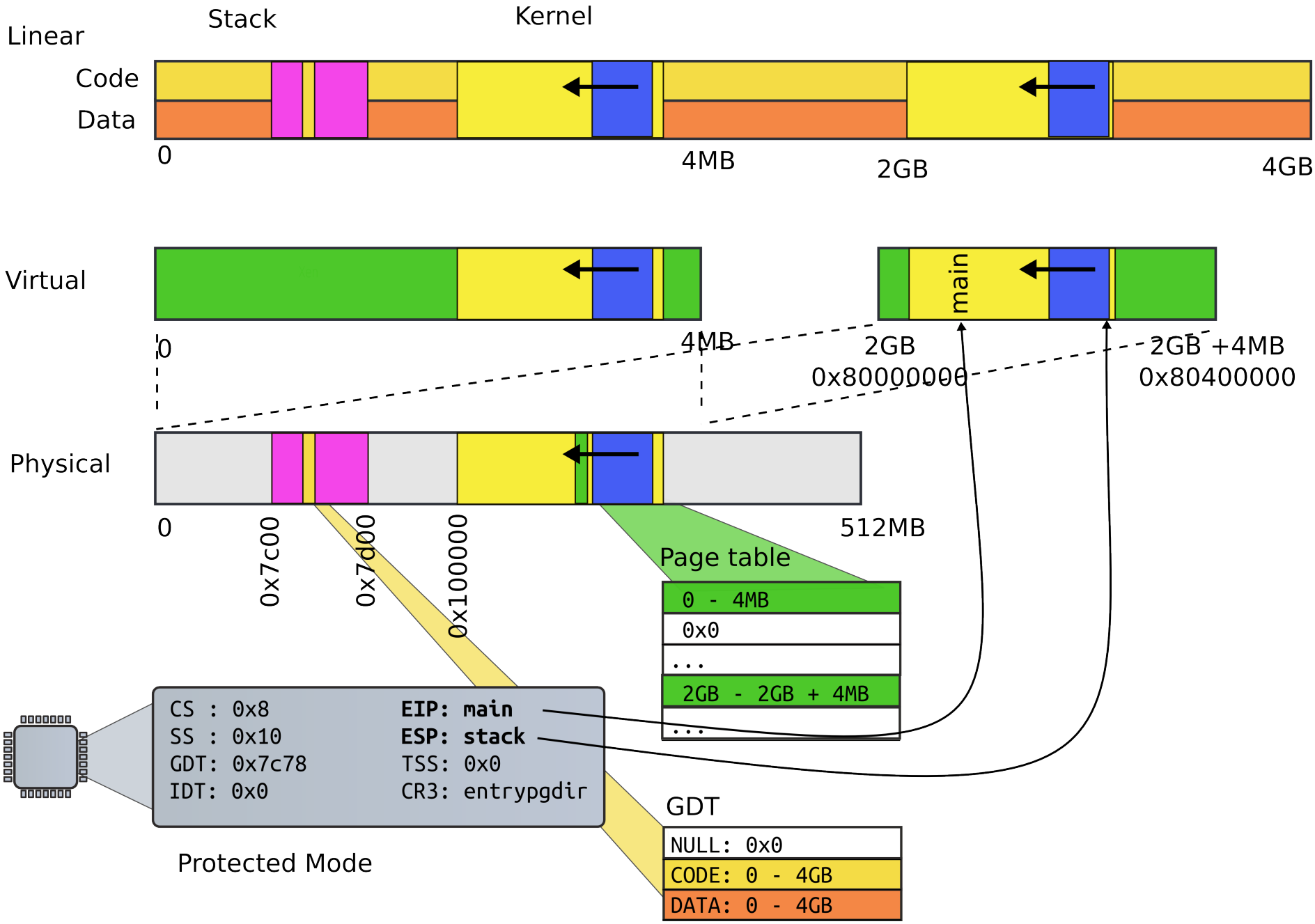
There are 125 sheep and 5 dogs in a flock. How old is the shepherd?

Recap from last time

- Setup segments (data and code)
- Switched to protected mode
  - Loaded GDT (segmentation is on)
- Setup stack (to call C functions)
- Loaded kernel from disk
- Setup first page table
  - 2 entries [0 : 4MB] and [2GB : (2GB + 4MB)]
- Setup high-address stack
- Jumped to main()

```
1157 # Set up the stack pointer.
1158 movl $(stack + KSTACKSIZE), %esp
1159
1160 # Jump to main(), and switch to executing at
1161 # high addresses. The indirect call is needed
because
1162 # the assembler produces a PC-relative instruction
1163 # for a direct jump.
1164 mov $main, %eax
1165 jmp *%eax
1166
1167 .comm stack, KSTACKSIZE
```

Jumped to main()

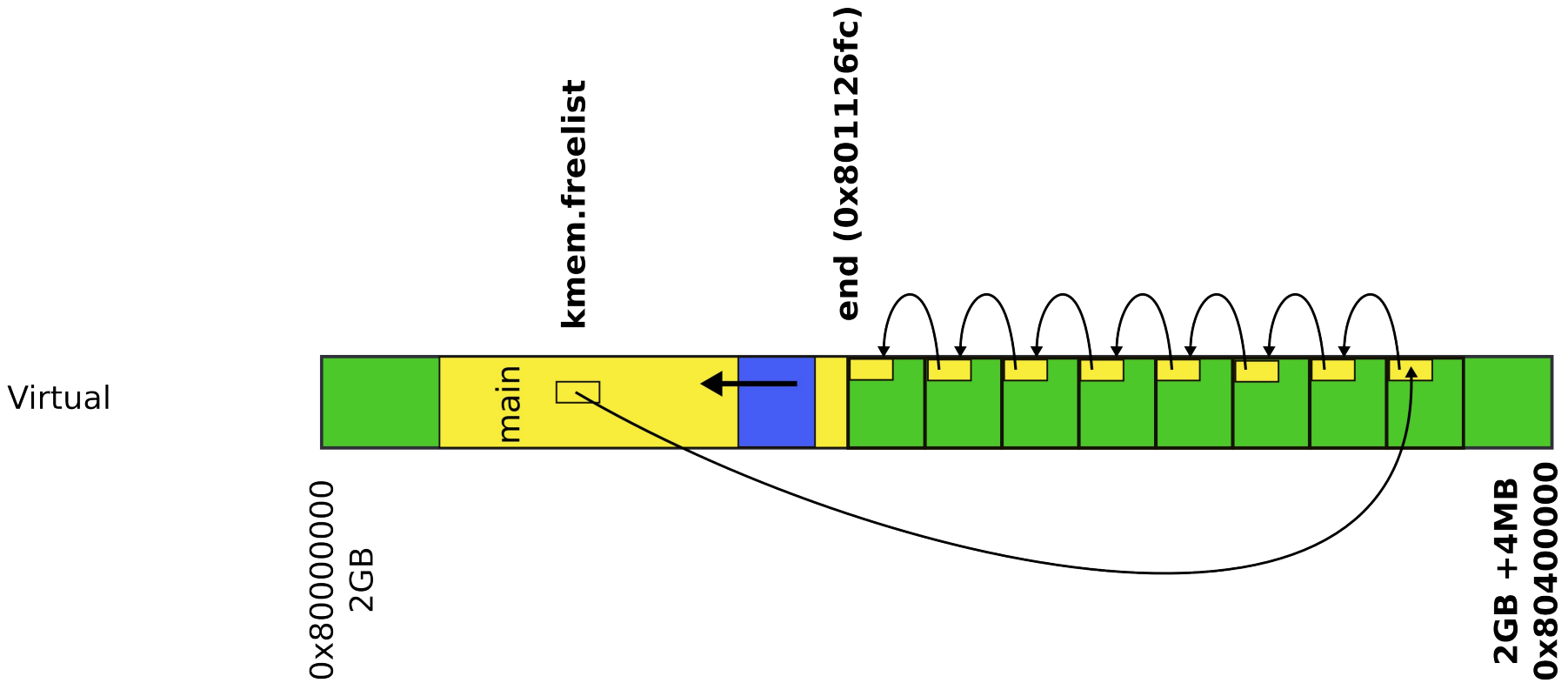


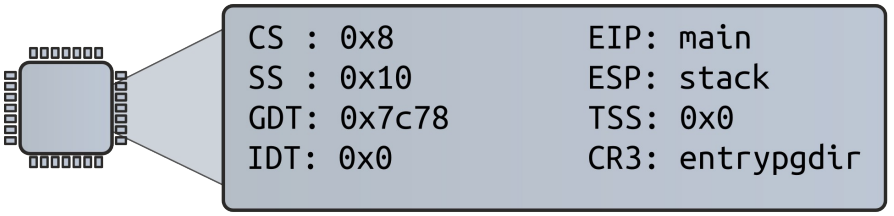
```
1316 int
1317 main(void)
1318 {
1319     kinit1(end, P2V(4*1024*1024)); // phys page allocator
1320     kvmalloc(); // kernel page table
1321     mpinit(); // detect other processors
1322     lapicinit(); // interrupt controller
1323     seginit(); // segment descriptors
1324     cprintf("\ncpu%d: starting xv6\n\n", cpunum());
1325     picinit(); // another interrupt controller
1326     ioapicinit(); // another interrupt controller
1327     consoleinit(); // console hardware
1328     uartinit(); // serial port
...
1340 }
```

Physical page allocator

- Goal:
  - List of free physical pages
  - To allocate page tables, stacks, data structures, etc.
  - Remember current page table is only 1! page
- Where to get memory to keep the list itself?
  - 1 level, only 4MB entries
    - You don't even have space to keep the second level page tables

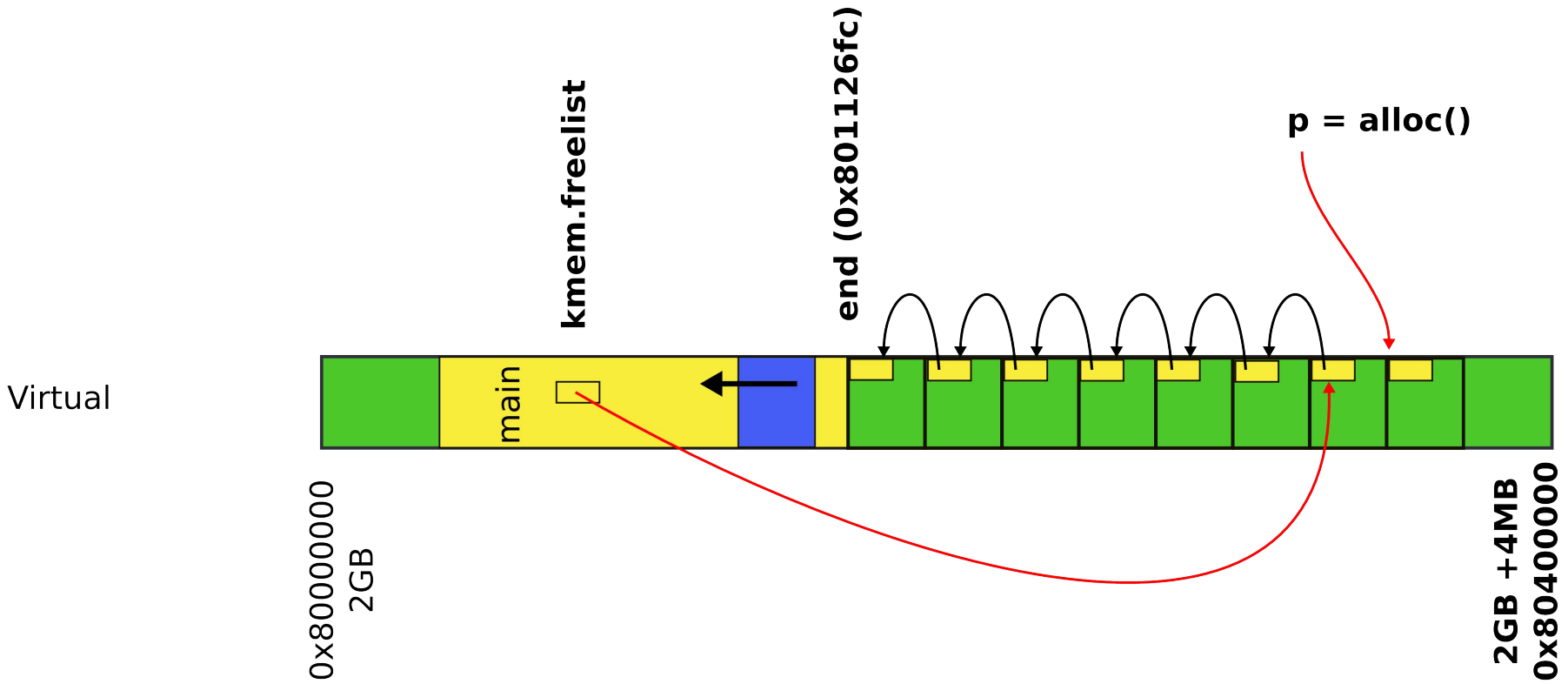
Physical page allocator

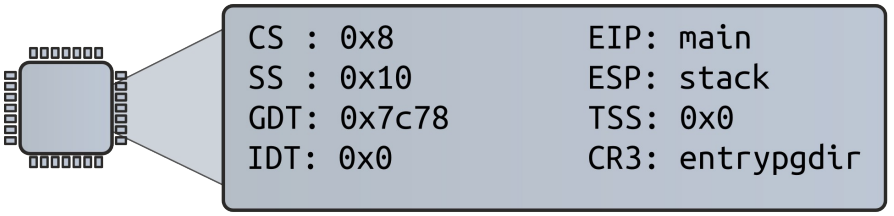




Protected Mode

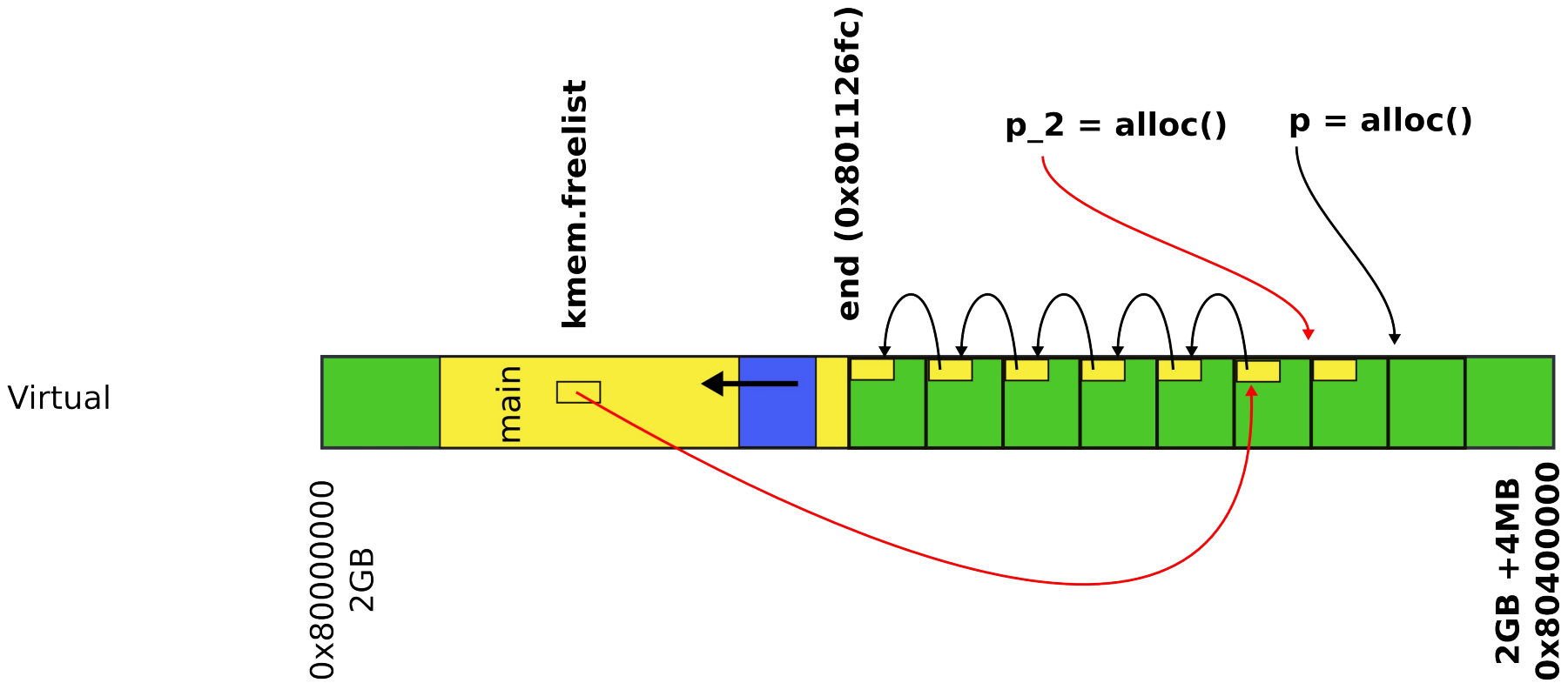
Physical page allocator

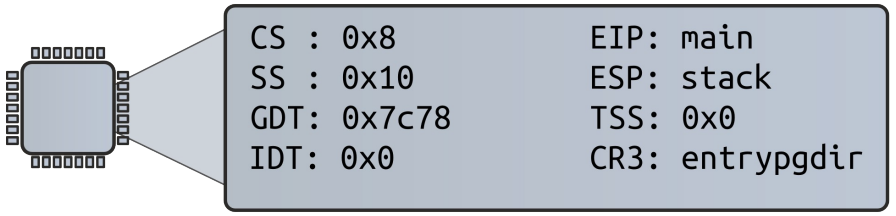




Protected Mode

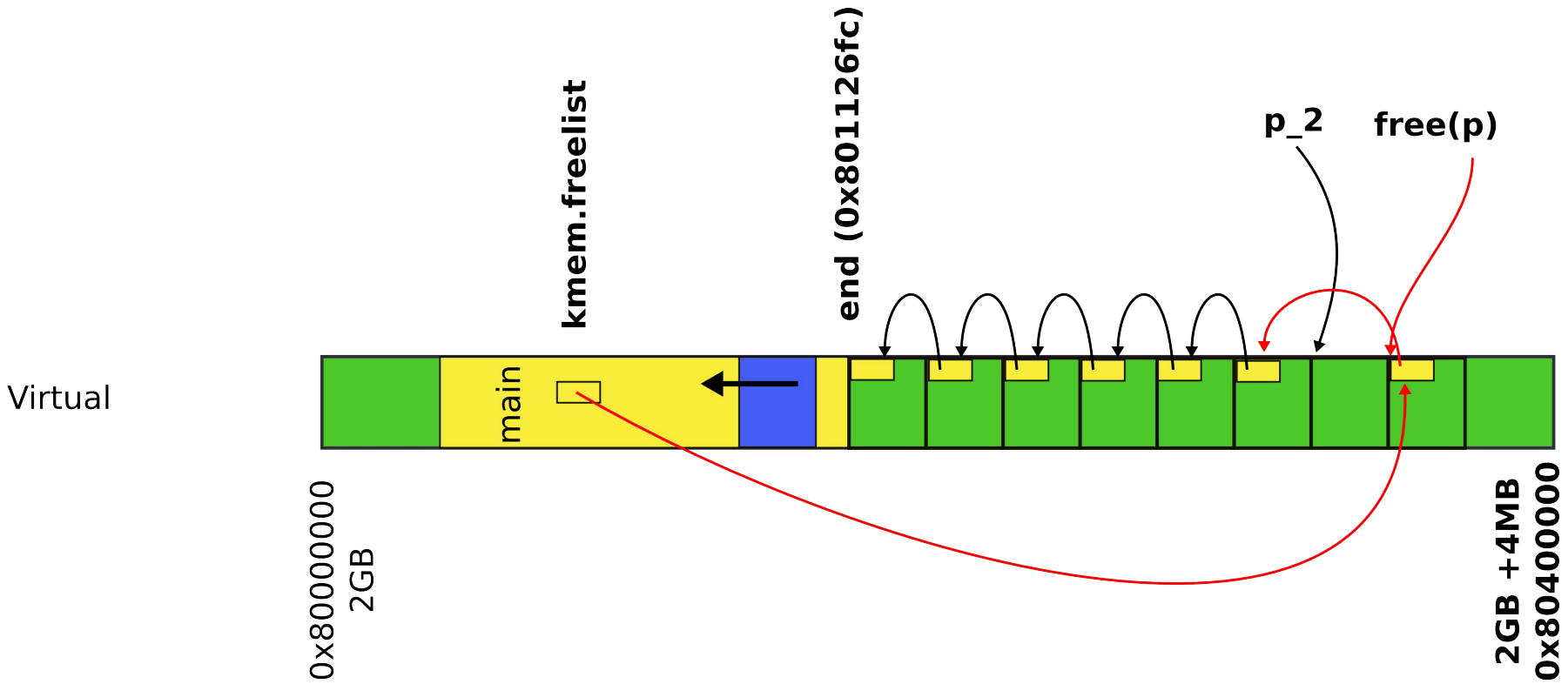
Physical page allocator

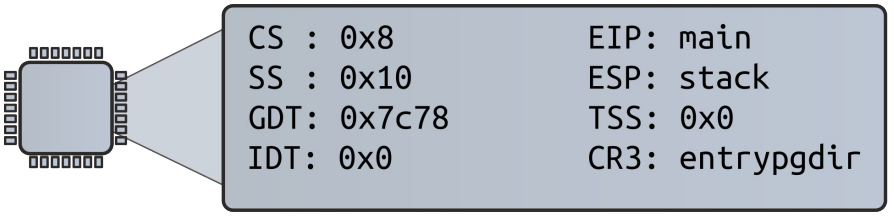




Protected Mode

Physical page allocator





Protected Mode

kalloc() - kernel allocator

```
3087 char*
3088 kalloc(void)
3089 {
3080   struct run *r;
...
3094   r = kmem.freelist;
3095   if(r)
3096     kmem.freelist = r->next;
…
3099   return (char*)r;
3099 }
```

```
3065 kfree(char *v)
3066 {
3067   struct run *r;
...
3077   r = (struct run*)v;
3078   r->next = kmem.freelist;
3079   kmem.freelist = r;
...
2832 }
```

Back to kinit1()

```
3030 kinit1(void *vstart, void *vend)
3031 {
...
3034   freerange(vstart, vend);
3035 }


3051 freerange(void *vstart, void *vend)
3052 {
3053   char *p;
3054   p = (char*)PGROUNDUP((uint)vstart);
3055   for(; p + PGSIZE <= (char*)vend; p += PGSIZE)
3056     kfree(p);
3057 }
```

Wait! Where do we start?

```
1316 int
1317 main(void)
1318 {
1319     kinit1(end, P2V(4*1024*1024)); // phys page allocator
1320     kvmalloc(); // kernel page table
1321     mpinit(); // detect other processors
```

- What is this **`end`**?

```
1311 extern char end[];
```

Wait! Where do we start?

```
1316 int
1317 main(void)
1318 {
1319     kinit1(end, P2V(4*1024*1024)); // phys page allocator
1320     kvmalloc(); // kernel page table
1321     mpinit(); // detect other processors
```

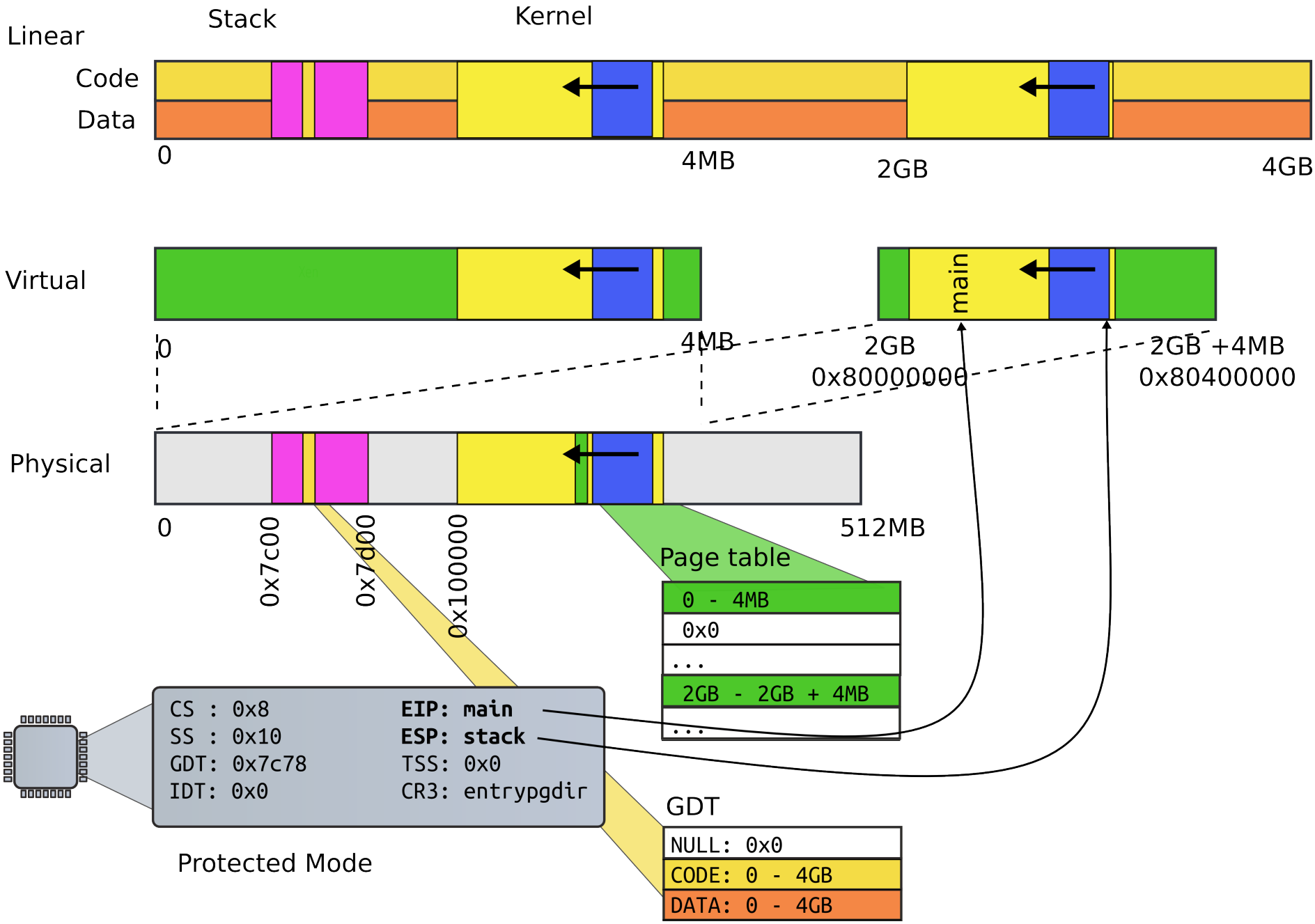
- What is this **`end`**?

```
1311 extern char end[]; // first address after
                       kernel loaded from ELF file
```

```
1157 # Set up the stack pointer.
1158 movl $(stack + KSTACKSIZE), %esp
1159
1160 # Jump to main(), and switch to executing at
1161 # high addresses. The indirect call is needed
because
1162 # the assembler produces a PC-relative instruction
1163 # for a direct jump.
1164 mov $main, %eax
1165 jmp *%eax
1166
1167 .comm stack, KSTACKSIZE
```

How come `$main` makes sense?

Why is it there...0x80000000 + something?



Makefile

```
bootblock: bootasm.S bootmain.c
        $(CC) $(CFLAGS) -fno-pic -O -nostdinc -I. -c bootmain.c
        $(CC) $(CFLAGS) -fno-pic -nostdinc -I. -c bootasm.S
        $(LD) $(LDFLAGS) -N -e start -Ttext 0x7C00 -o bootblock.o bootasm.o bootmain.o
        $(OBJDUMP) -S bootblock.o > bootblock.asm
        $(OBJCOPY) -S -O binary -j .text bootblock.o bootblock


kernel: $(OBJS) entry.o entryother initcode kernel.ld
        $(LD) $(LDFLAGS) -T kernel.ld -o kernel entry.o $(OBJS) -b binary initcode
                                                                    entryother
        $(OBJDUMP) -S kernel > kernel.asm
        $(OBJDUMP) -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$$/d' > kernel.sym
```

kernel.ld: Linker script

```
OUTPUT_FORMAT("elf32-i386", "elf32-i386", "elf32-i386")
OUTPUT_ARCH(i386)
ENTRY(_start)

SECTIONS
{
        /* Link the kernel at this address: "." means the current address */
        /* Must be equal to KERNLINK */
        . = 0x80100000;
….
        .bss : {
                *(.bss)
        }
        PROVIDE(end = .);
}
```

Back to main(): Kernel page table

```
1316 int
1317 main(void)
1318 {
1319     kinit1(end, P2V(4*1024*1024)); // phys page allocator
1320     kvmalloc(); // kernel page table
1321     mpinit(); // detect other processors
1322     lapicinit(); // interrupt controller
1323     seginit(); // segment descriptors
1324     cprintf("\ncpu%d: starting xv6\n\n", cpunum());
1325     picinit(); // another interrupt controller
1326     ioapicinit(); // another interrupt controller
1327     consoleinit(); // console hardware
1328     uartinit(); // serial port
...
1340 }
```

kvmalloc()

```
1857 kvmalloc(void)
1858 {
1859   kpgdir = setupkvm();
1860   switchkvm();
1861 }
```

Allocate page table directory

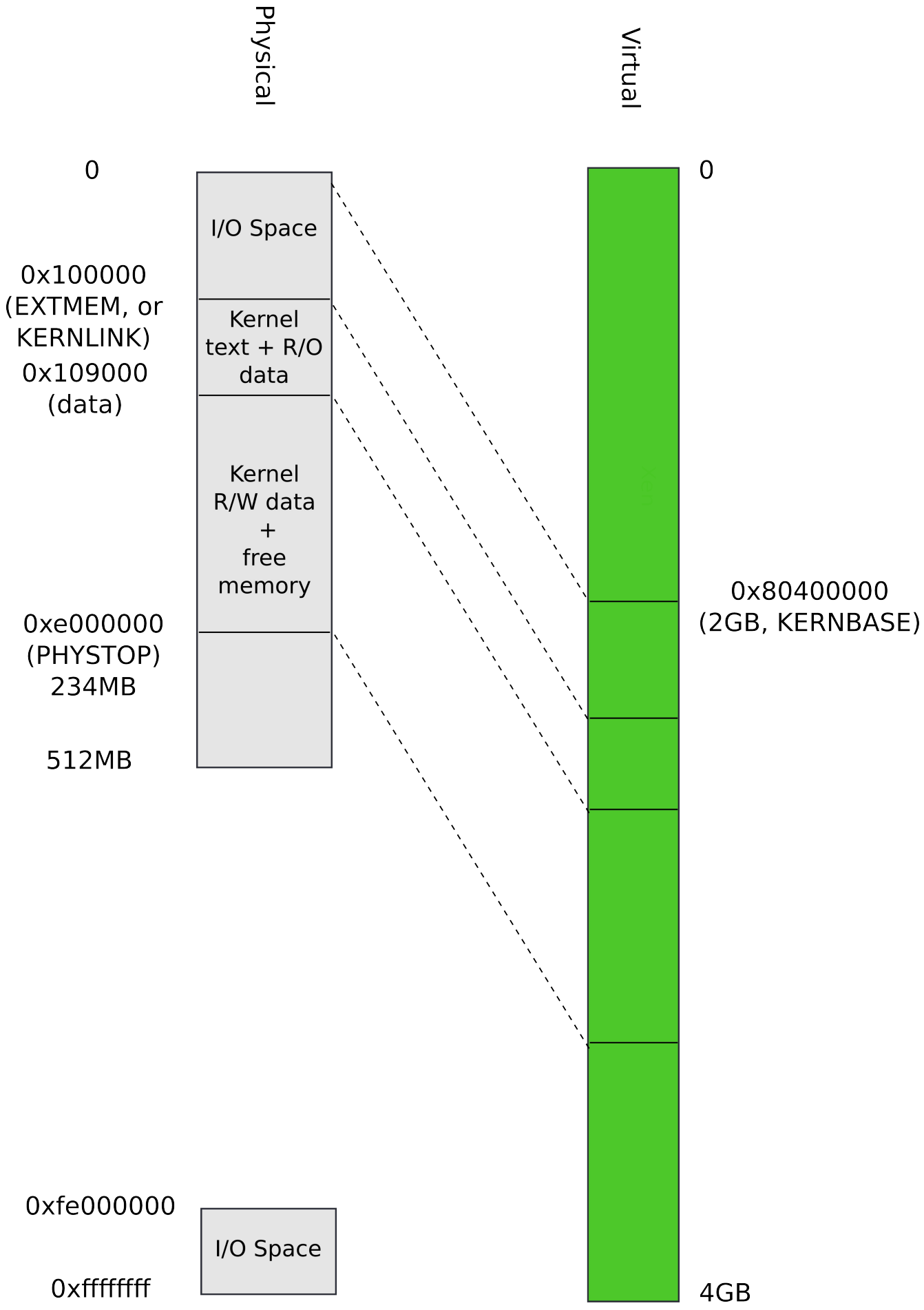
```
1836 pde_t*
1837 setupkvm(void)
1838 {
1839   pde_t *pgdir;
1840   struct kmap *k;
1841
1842   if((pgdir = (pde_t*)kalloc()) == 0)
1843     return 0;
1844   memset(pgdir, 0, PGSIZE);
...
1847   for(k = kmap; k < &kmap[NELEM(kmap)]; k++)
1848     if(mappages(pgdir, k->virt, k->phys_end - k->phys_start,
1849                 (uint)k->phys_start, k->perm) < 0)
1850       return 0;
1851   return pgdir;
1852 }
```

Remap physical pages

```
1836 pde_t*
1887 setupkvm(void)
1838 {
1839   pde_t *pgdir;
1840   struct kmap *k;
1841
1842   if((pgdir = (pde_t*)kalloc()) == 0)
1843     return 0;
1844   memset(pgdir, 0, PGSIZE);
...
1847   for(k = kmap; k < &kmap[NELEM(kmap)]; k++)
1848     if(mappages(pgdir, k->virt, k->phys_end - k->phys_start,
1849                 (uint)k->phys_start, k->perm) < 0)
1850       return 0;
1851   return pgdir;
1852 }
```

Kmap – kernel map

```
1823 static struct kmap {
1824     void *virt;
1825     uint phys_start;
1826     uint phys_end;
1827     int perm;
1828 } kmap[] = {
1829     { (void*)KERNBASE, 0, EXTMEM, PTE_W}, // I/O space
1830     { (void*)KERNLINK, V2P(KERNLINK), V2P(data), 0}, // kern
text+rodata
1831     { (void*)data, V2P(data), PHYSTOP, PTE_W}, // kern
data+memory
1832     { (void*)DEVSPACE, DEVSPACE, 0, PTE_W}, // more devices
1833 };
```


Physical
Virtual
0
I/O Space
0x100000
(EXTMEM, or
KERNLINK)
Kernel
text + R/O
data
0x109000
(data)
Kernel
R/W data
+
free
memory
0xe000000
(PHYSTOP)
234MB
512MB
0
0x80400000
(2GB, KERNBASE)
0xfe000000
I/O Space
0xffffffff
4GB

Remap physical pages

```
1836 pde_t*
1887 setupkvm(void)
1838 {
1839   pde_t *pgdir;
1840   struct kmap *k;
1841
1842   if((pgdir = (pde_t*)kalloc()) == 0)
1843     return 0;
1844   memset(pgdir, 0, PGSIZE);
...
1847   for(k = kmap; k < &kmap[NELEM(kmap)]; k++)
1848     if(mappages(pgdir, k->virt, k->phys_end - k->phys_start,
1849                 (uint)k->phys_start, k->perm) < 0)
1850       return 0;
1851   return pgdir;
1852 }
```

```
1779 mappages(pde_t *pgdir, void *va, uint size, uint pa, int perm)
1780 {
1781   char *a, *last;
1782   pte_t *pte;
1783
1784   a = (char*)PGROUNDDOWN((uint)va);
1785   last = (char*)PGROUNDDOWN(((uint)va) + size - 1);
1786   for(;;){
1787     if((pte = walkpgdir(pgdir, a, 1)) == 0)
1788       return -1;
1789     if(*pte & PTE_P)
1790       panic("remap");
1791     *pte = pa | perm | PTE_P;
1792     if(a == last)
1793       break;
1794     a += PGSIZE;
1795     pa += PGSIZE;
1796   }
1797   return 0;
1798 }
```

Create page table entries

PDX()

```
0805 // +--------10------+-------10-------+---------12----------+
0806 // | Page Directory |    Page Table   | Offset within Page  |
0807 // |    Index       |     Index       |                     |
0808 // +----------------+-----------------+---------------------+
0809 // \--- PDX(va) --/  \--- PTX(va) --/
0810
0811 // page directory index
0812 #define PDX(va) (((uint)(va) >> PDXSHIFT) & 0x3FF)
...
0827 #define PDXSHIFT 22 // offset of PDX in a linear address
```

P2V and V2P

```
0206 // Key addresses for address space layout (see kmap in vm.c for
layout)

0207 #define KERNBASE 0x80000000 // First kernel virtual address

0208 #define KERNLINK (KERNBASE+EXTMEM) // Address where kernel is linked

0209

0210 #define V2P(a) (((uint) (a)) - KERNBASE)

0211 #define P2V(a) (((void *) (a)) + KERNBASE)
```

Walk page table

```
1754 walkpgdir(pde_t *pgdir, const void *va, int alloc)
1755 {
1756     pde_t *pde;
1757     pte_t *pgtab;
1758
1759     pde = &pgdir[PDX(va)];
1760     if(*pde & PTE_P){
1761         pgtab = (pte_t*)P2V(PTE_ADDR(*pde));
1762     } else {
1763         if(!alloc || (pgtab = (pte_t*)kalloc()) == 0)
1764             return 0;
1765         // Make sure all those PTE_P bits are zero.
1766         memset(pgtab, 0, PGSIZE);
...
1770         *pde = V2P(pgtab) | PTE_P | PTE_W | PTE_U;
1771     }
1772     return &pgtab[PTX(va)];
1773 }
```

Walk page table

```
1754 walkpgdir(pde_t *pgdir, const void *va, int alloc)
1755 {
1756     pde_t *pde;
1757     pte_t *pgtab;
1758
1759     pde = &pgdir[PDX(va)];
1760     if(*pde & PTE_P){
1761         pgtab = (pte_t*)P2V(PTE_ADDR(*pde));
1762     } else {
1763         if(!alloc || (pgtab = (pte_t*)kalloc()) == 0)
1764             return 0;
1765         // Make sure all those PTE_P bits are zero.
1766         memset(pgtab, 0, PGSIZE);
...
1770         *pde = V2P(pgtab) | PTE_P | PTE_W | PTE_U;
1771     }
1772     return &pgtab[PTX(va)];
1773 }
```

```
1779 mappages(pde_t *pgdir, void *va, uint size, uint pa, int perm)
1780 {
1781   char *a, *last;
1782   pte_t *pte;
1783
1784   a = (char*)PGROUNDDOWN((uint)va);
1785   last = (char*)PGROUNDDOWN(((uint)va) + size - 1);
1786   for(;;){
1787     if((pte = walkpgdir(pgdir, a, 1)) == 0)
1788       return -1;
1789     if(*pte & PTE_P)
1790       panic("remap");
1791     *pte = pa | perm | PTE_P;
1792     if(a == last)
1793       break;
1794     a += PGSIZE;
1795     pa += PGSIZE;
1796   }
1797   return 0;
1798 }
```

Create page table entries

kvmalloc()

```
1757 kvmalloc(void)
1758 {
1759   kpgdir = setupkvm();
1760   switchkvm();
1761 }
```

Switch to the new page table

```
1765 void
1766 switchkvm(void)
1767 {
1768   lcr3(v2p(kpgdir));
1769 }
```

Thank you!