xv6©-rev10

(Copyright Frans Kaashoek, Robert Morris, and Russ Cox.)

First Process Creation, I

Carmi Merimovich

Tel-Aviv Academic College

November 10, 2017

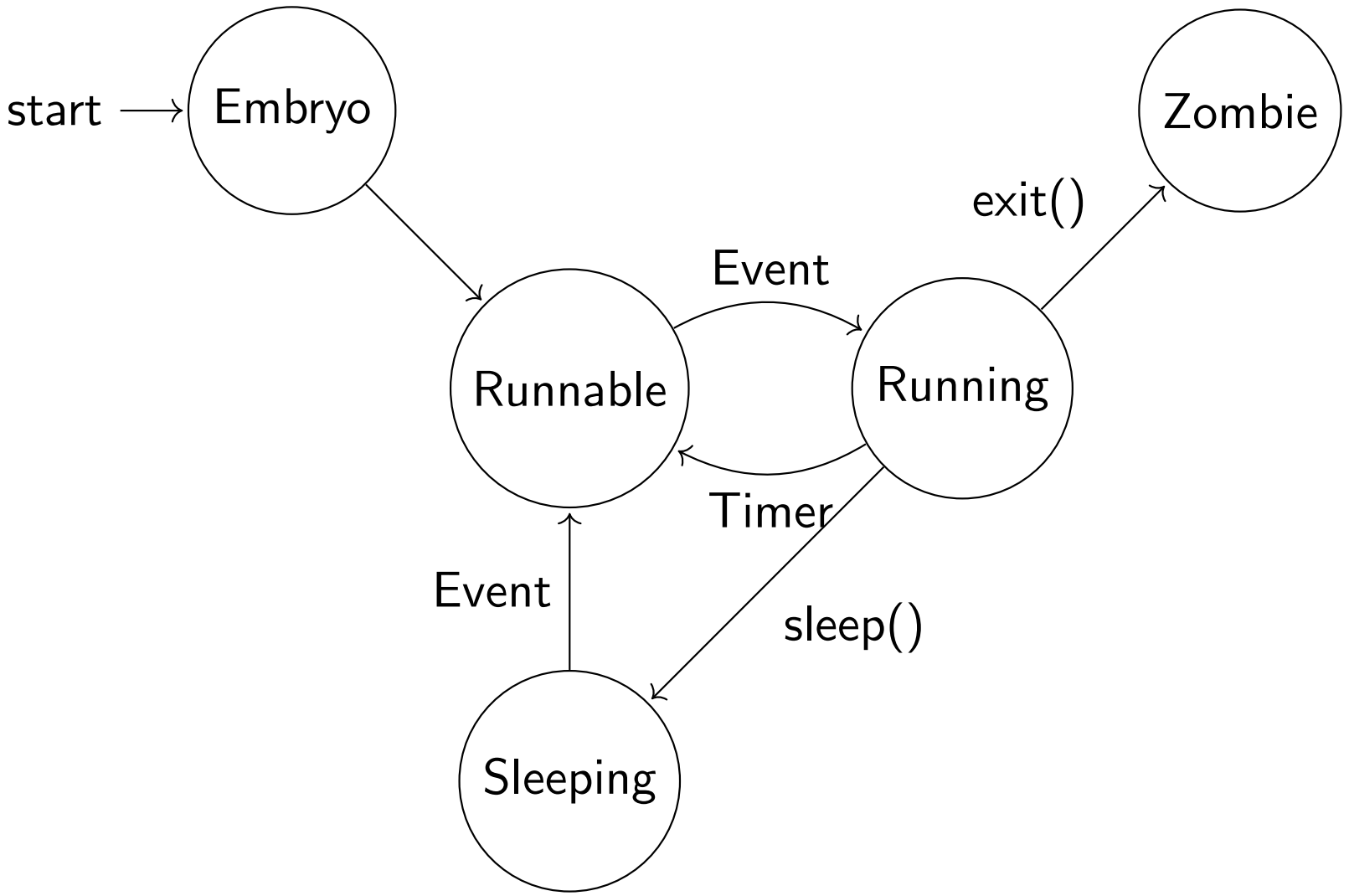
Context

```
kinit1(end, P2V(4*1024*1024)); // phys page alloca
kvmalloc();        // kernel page table
   ⋮
seginit();         // set up segments
   ⋮
pinit();           // process table
   ⋮
   ⋮
kinit2(P2V(4*1024*1024), P2V(PHYSTOP)); // must co
userinit();        // first user process
mpmain();
```

struct proc

```
2337 struct proc {
     uint sz; // Size of process memory (bytes)
     pde_t* pgdir; // Page table
     char *kstack; // Bottom of kernel stack for this pr
     enum procstate state; // Process state
     volatile int pid; // Process ID
     struct proc *parent; // Parent process
     struct trapframe *tf; // Trap frame for current sys
     struct context *context; // swtch() here to run pro
     void *chan; // If non zero, sleeping on chan
     int killed; // If non zero, have been killed
     struct file *ofile[NOFILE]; // Open files
     struct inode *cwd; // Current directory
     char name[16]; // Process name (debugging)
    };
```

state transition



All transition are due to INTERRUPTS.

proc structure storage

```
2334 enum procstate { UNUSED, EMBRYO, SLEEPING,
                      RUNNABLE, RUNNING, ZOMBIE }

2409 struct {
      struct spinlock lock;
      struct proc proc[NPROC];
     } ptable;
```

First process simplifying assumptions

- Only 4KB user space area.
- The code/data content is stored at some static place in the kernel.
- User code begins running at address 0.
- Top of user stack shall be at 4096.

Initial user mode state for first process

eax	0
ebx	0
ecx	0
edx	0
ebp	0
esi	0
edi	0
esp	4096
eip	0

don't care

cs	27
ds	35
ss	35
es	0
fs	0
gs	0

0

```
pushl $argv
pushl $init
pushl $0
movl $SYS_exec,%eax
int $64
exit:
movl $SYS_exit, %eax
int $T_SYSCALL
jmp exit

init:
.string "/init\0"

.p2align 2
argv:
.long init
.long 0
```

4095

xv6 steps taken to construct the state above

1. Allocating memory for user code and setting its content.
2. Adding mapping rules.
3. Store on the kernel stack the expected user mode registers values.

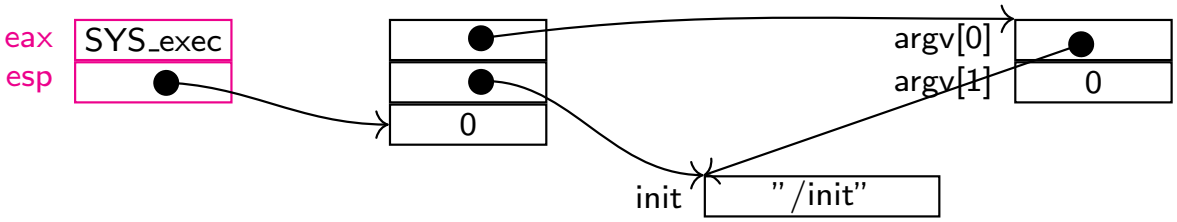
Putting initial user code into kernel data vector

initcode.S: User mode code of 1st process

```
8408 #include "syscall.h"
#include "traps.h"
  .globl start
start:
  pushl $argv
  pushl $init
  pushl $0 // caller pc
  movl  $SYS_exec,%eax
  int   $T_SYSCALL

exit:
  movl $SYS_exit, %eax
  int $T_SYSCALL
  jmp exit
```

```
init:
  .string "/init\0"

  .p2align 2
argv:
  .long init
  .long 0
```

State on the first **int $64**



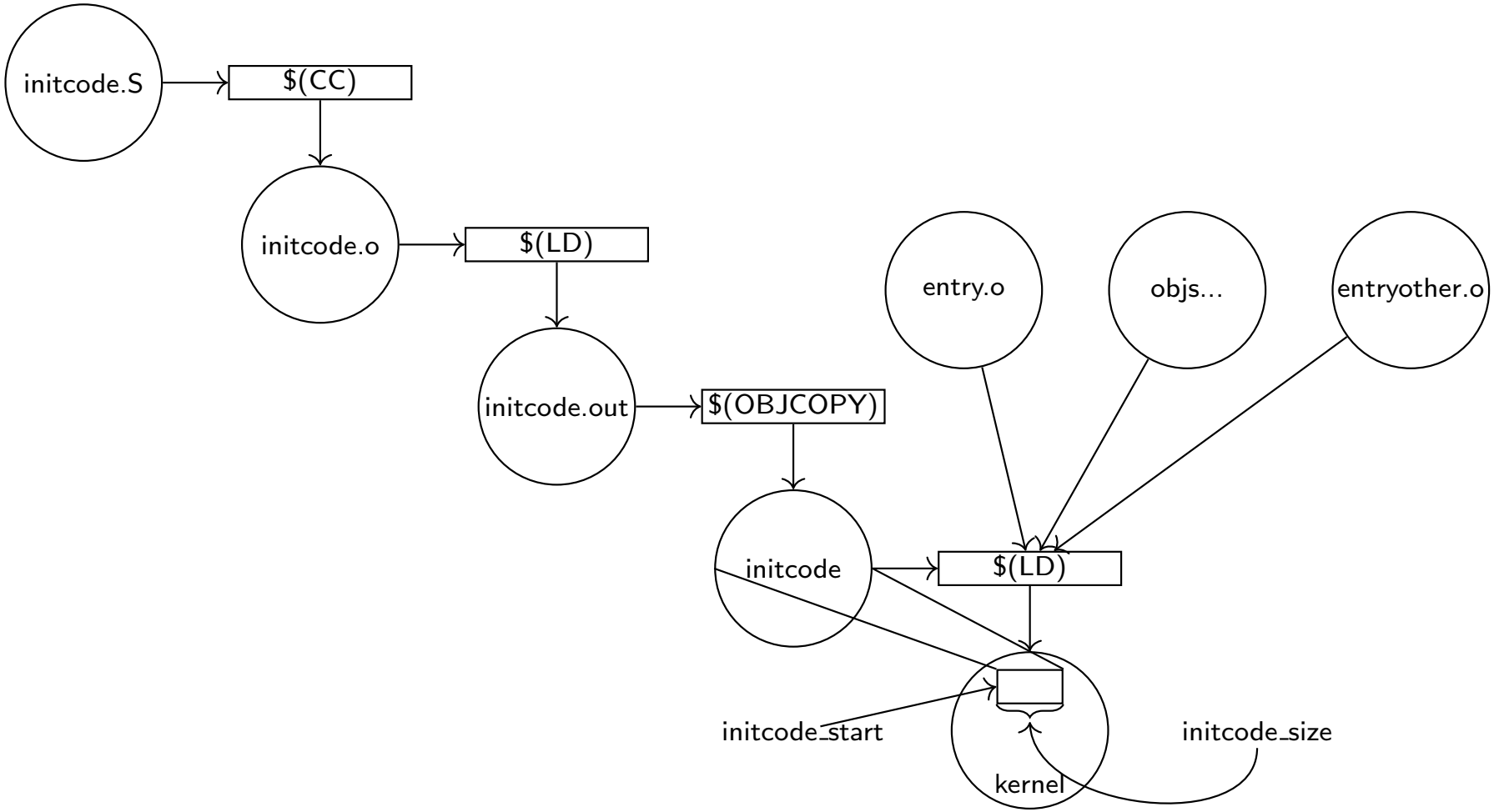
C code equivalent

```
char *argv[] = {"/init", 0};
main() {
 exec(argv[0], argv);
 for (;;)
  exit{};
}
```

initcode.S: Compiling, Linking, Embedding

```
initcode: initcode.S
 $(CC) $(CFLAGS) -nostdinc -I. -c initcode.S
 $(LD) $(LDFLAGS) -N -e start -Ttext 0 \
                        -o initcode.out initcode.o
 $(OBJCOPY) -S -O binary initcode.out initcode

kernel: $(OBJS) entry.o entryother initcode kernel.ld
 $(LD) $(LDFLAGS) -T kernel.ld -o kernel \
       entry.o $(OBJS) -b binary initcode entryother
```

Diagram of compiling, linking, embedding

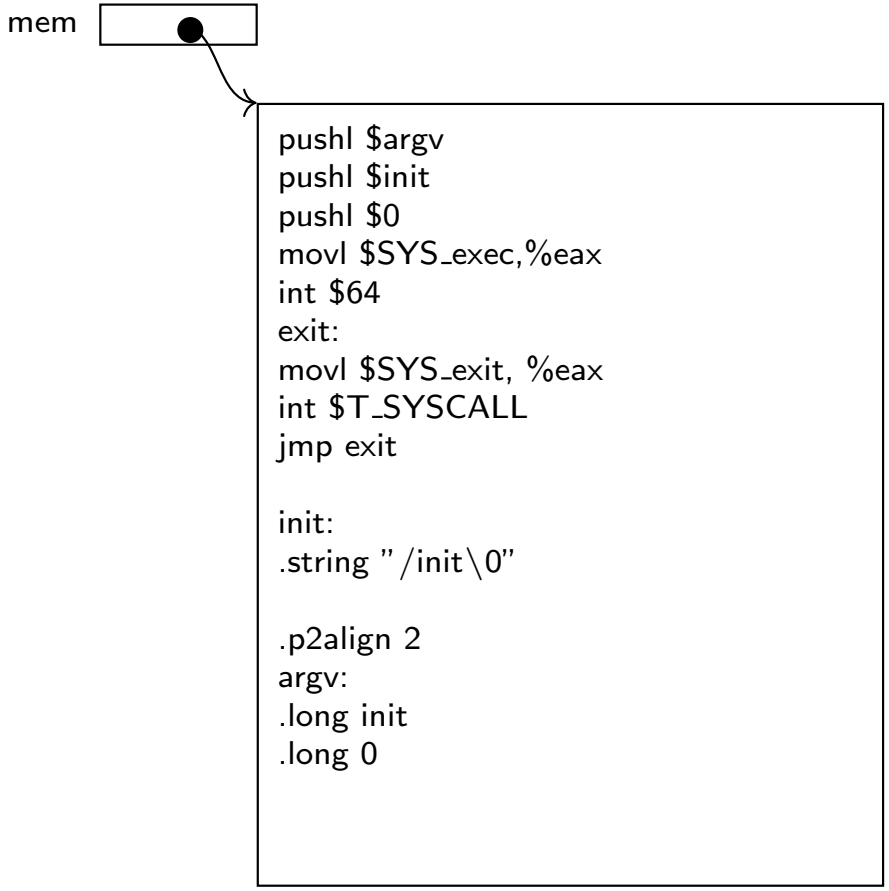


Building the memory area for the user mode.

Building user memory

```
extern char initcode_start[], initcode_size[];
mem = kalloc();
memset(mem, 0, PGSIZE);
memmove(mem, initcode_start, initcode_size);
```

Building user memory



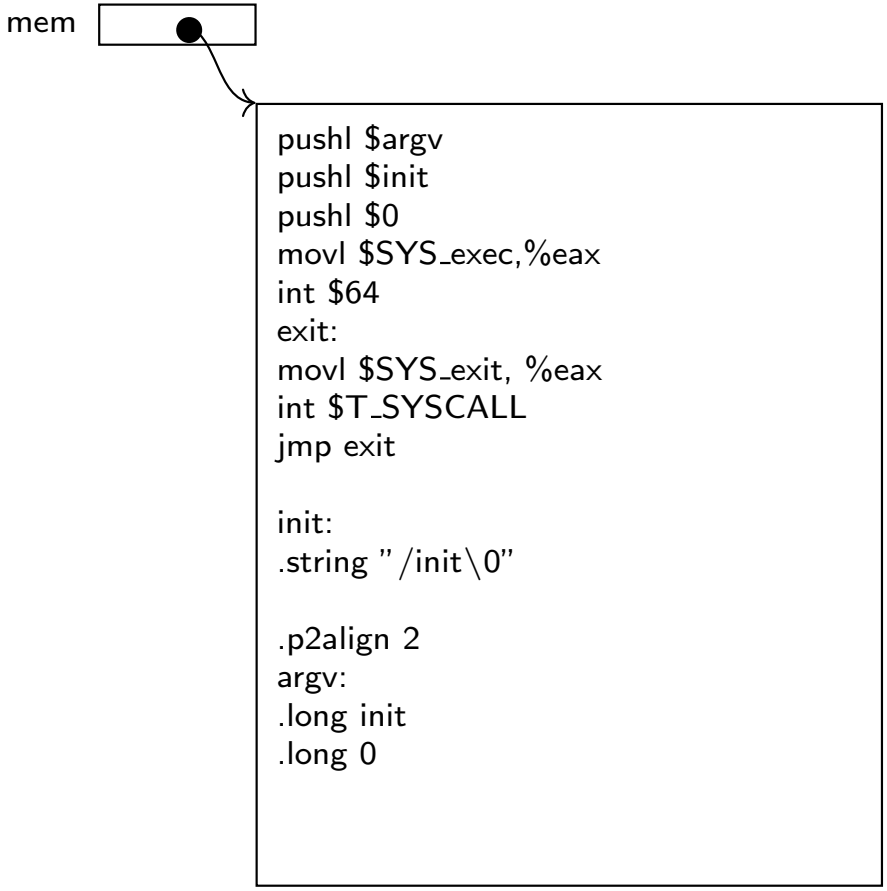
Building user memory

mem

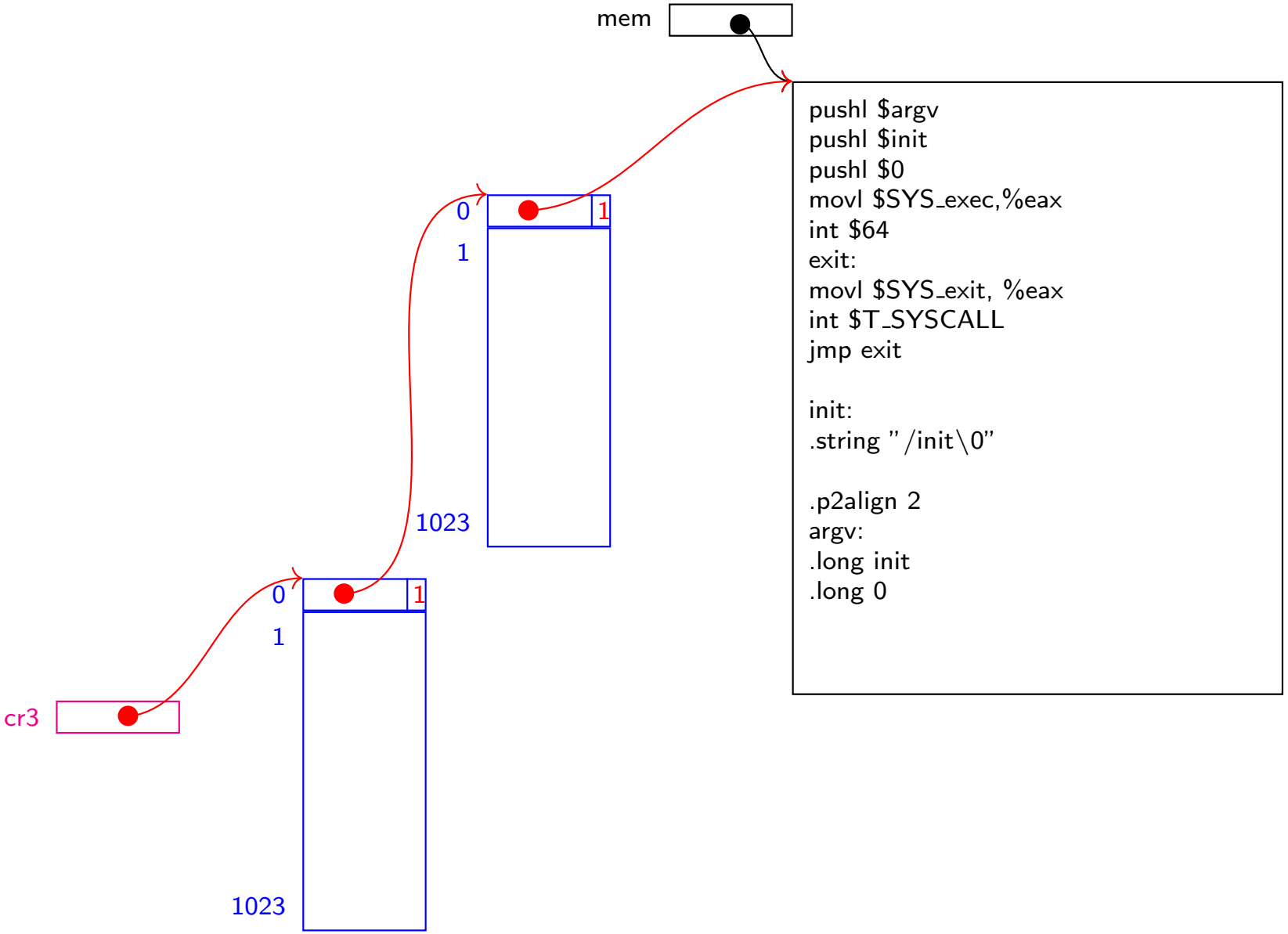
```
pushl $argv
pushl $init
pushl $0
movl $SYS_exec,%eax
int $64
exit:
movl $SYS_exit, %eax
int $T_SYSCALL
jmp exit

init:
.string "/init\0"

.p2align 2
argv:
.long init
.long 0
```

Do we have user mode addresses here?

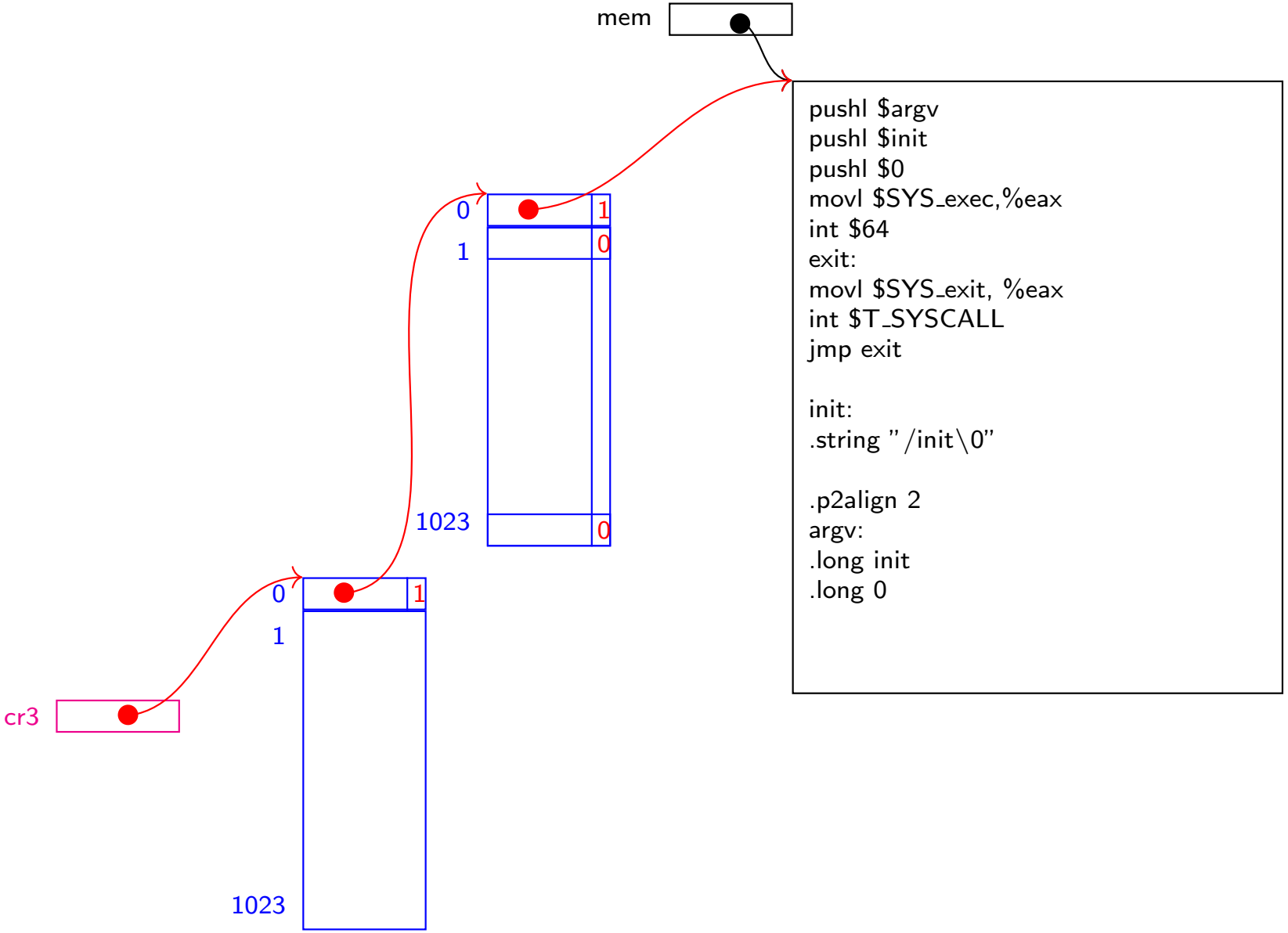
Mapping the first 4KB of user space



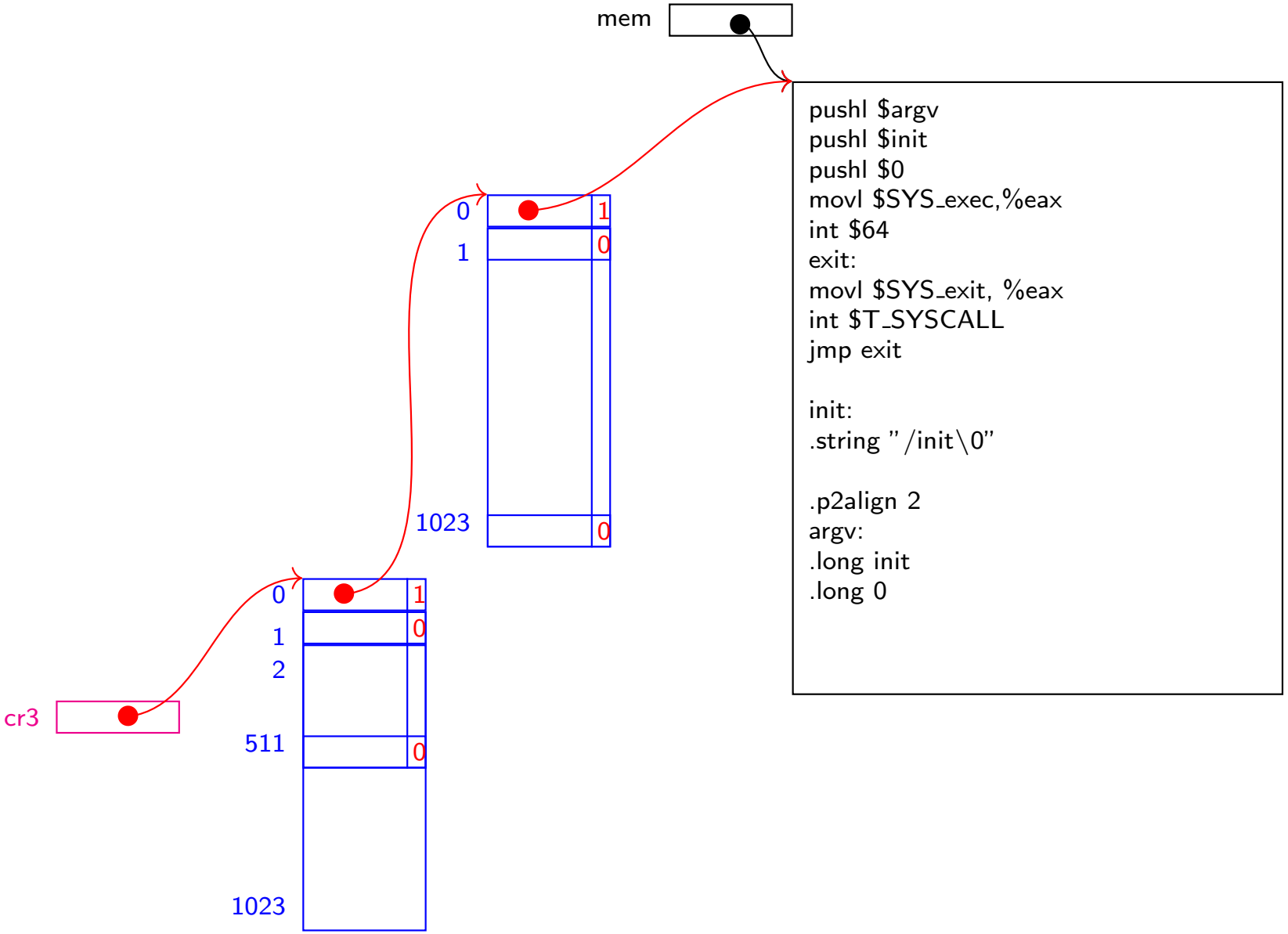
Mapping the first 4KB of user space



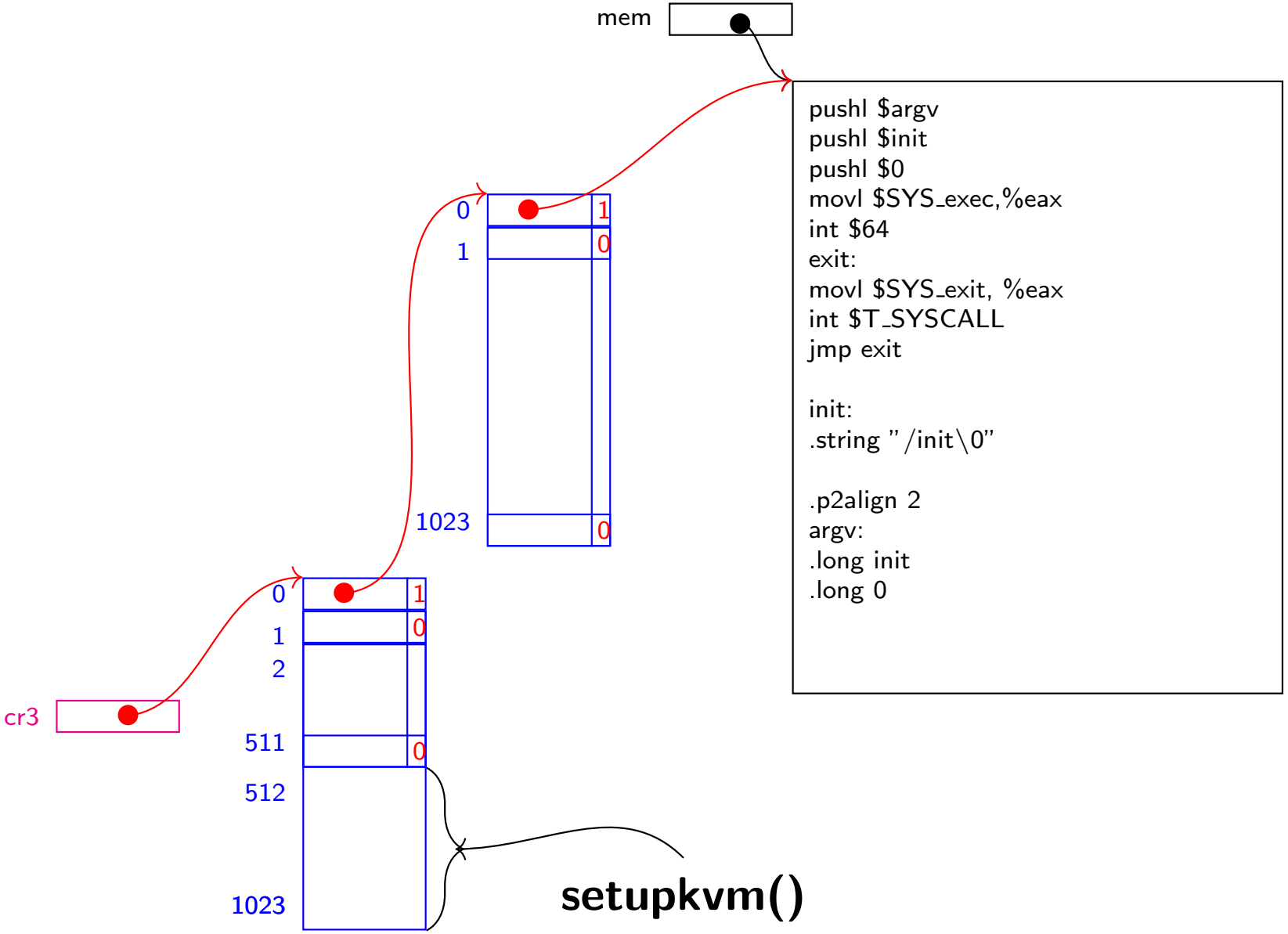
Mapping the first 4KB of user space



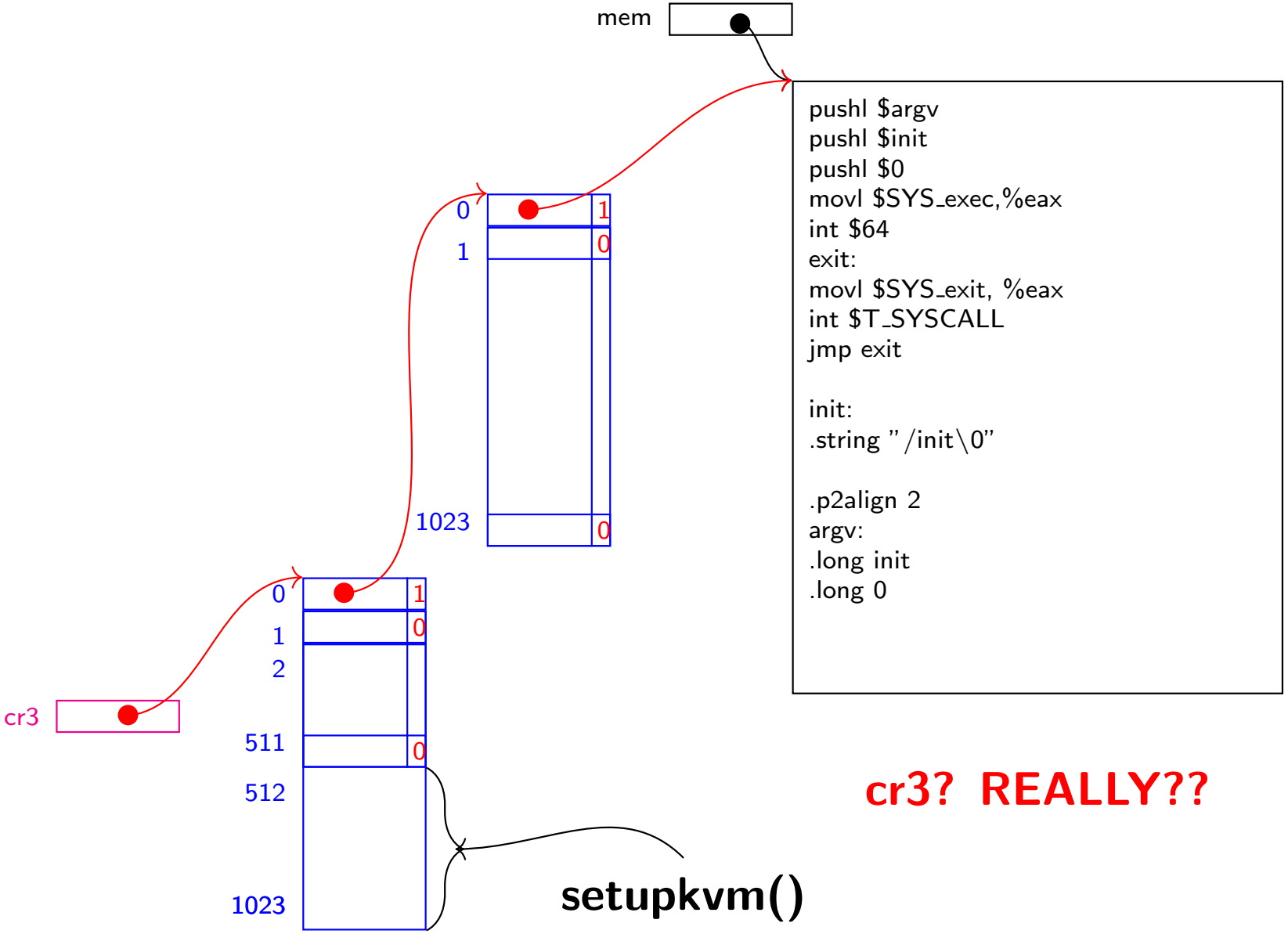
Mapping the first 4KB of user space



Mapping the first 4KB of user space

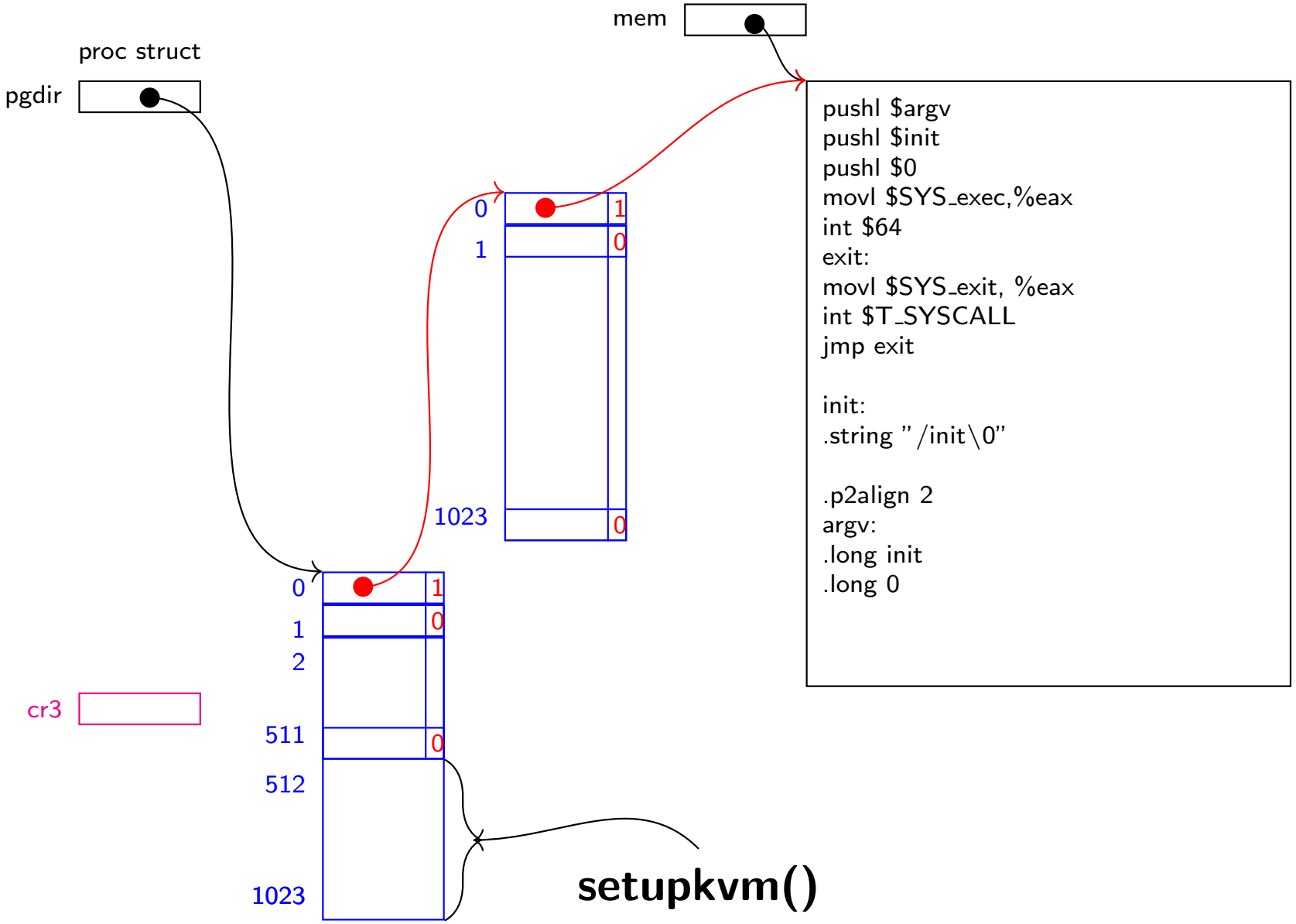


Mapping the first 4KB of user space

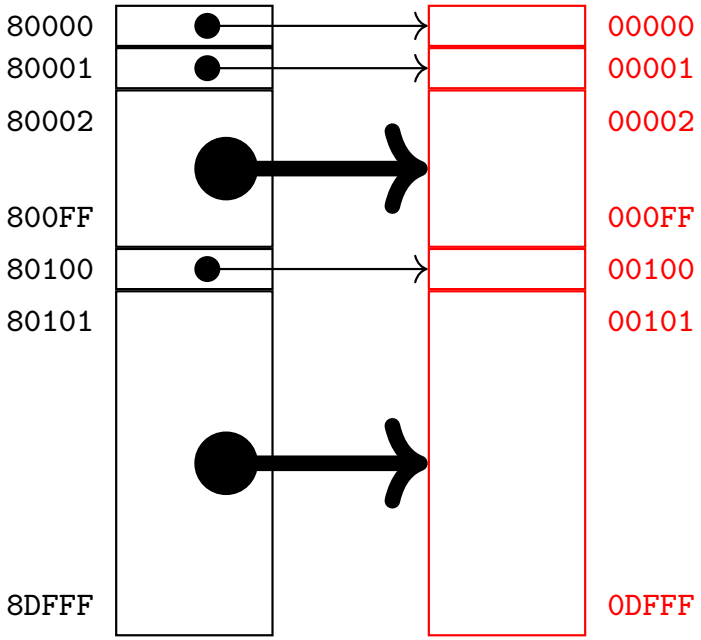


cr3? REALLY??

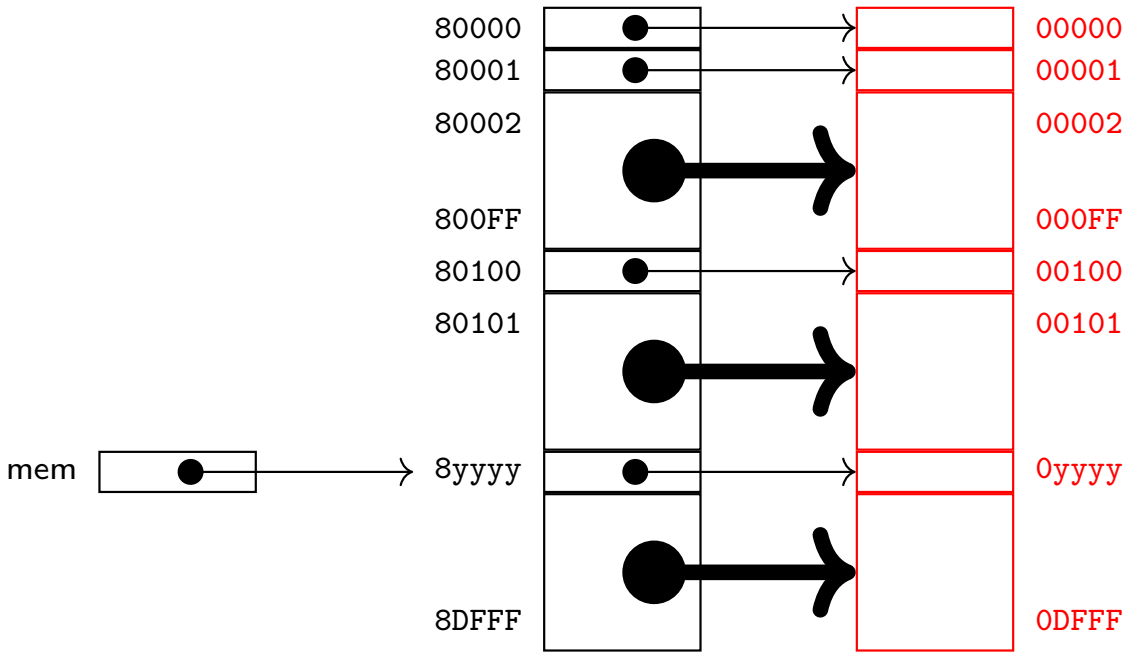
Mapping the first 4KB of user space



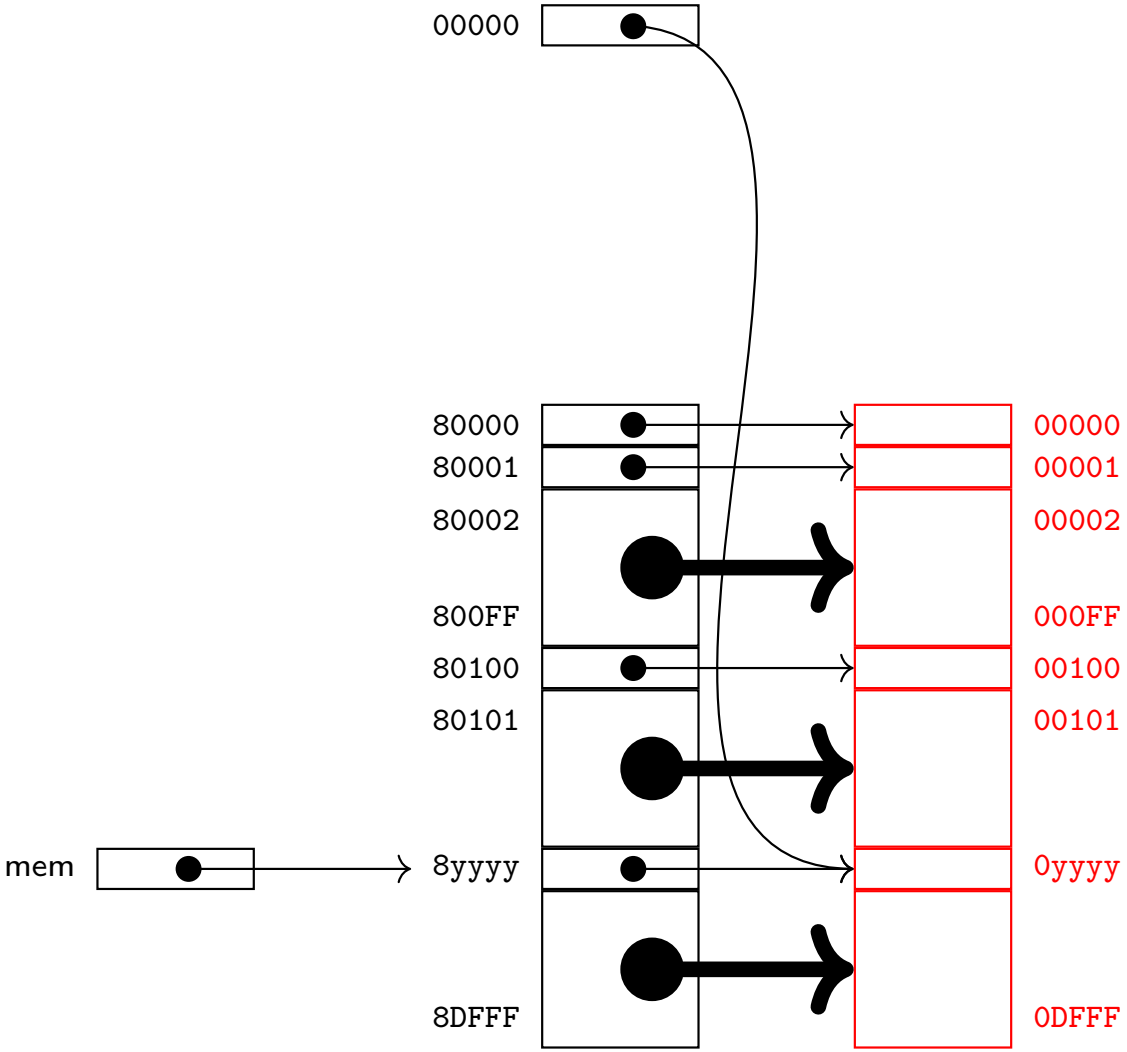
User mode pages are doubly mapped



User mode pages are doubly mapped



User mode pages are doubly mapped



Mapping first user page!

```
int *pgdir = setupkvm();

int *pte = walkpgdir(pgdir, 0, 1);
*pte = v2p(mem) | (PTE_P|PTE_U|PTE_W);
```