# OS 161:
# Address Space & Memory Management


Politecnico di Torino

Department of Control and Computer Engineering


System Programming - Sarah Azimi

CAD & Reliability Group
DAUIN– Politecnico di Torino

# Today:

- Memory Management in OS161
  - Kmalloc for kernel side
  - Dumbvm for user side
- MIPS Virtual Address Space
- Kernel Loader
- Allocating Memory

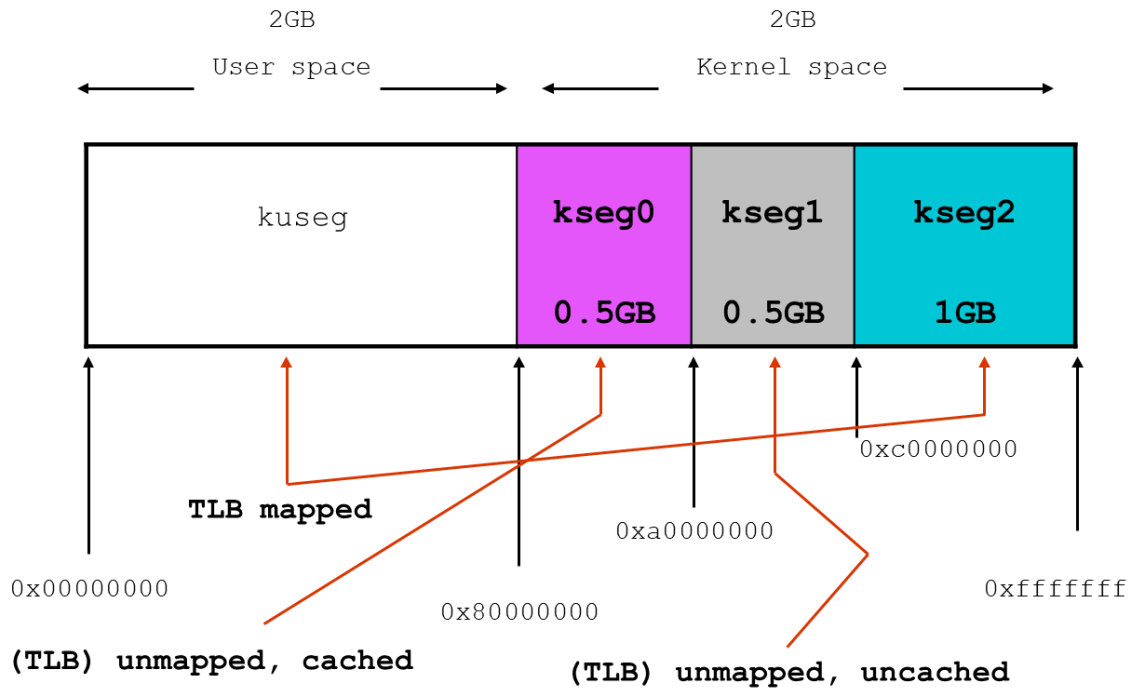
# Dumbvm and kmalloc

- Memory management of OS161 consists of two elements:
  - Kmalloc allows memory allocation of kernel side
  - Dumbvm memory manager user side
- Memory manager of OS161 use **Contiguous allocation**
- However, it also uses pagination (minimum amount of available memory is page(s))

# Dumbvm and kmalloc

- Allocation of memory is done in two levels:
  - getppages (dumbvm.c): calls ram_stealmem (in mutual exclusion) (more generic abstraction)
  - ram_stealmem (ram.c): allocates contiguous RAM starting at firstpaddr, that is increased (containing codes architecture dependent)
- Allocator is common to (for both user and kernel)
  - User memory: **as_prepare_load** calls **getppages** for 2 user segments and a stack
  - Dynamic kernel memory: **kmalloc** is based on **alloc_kpages**, that calls **getppages**

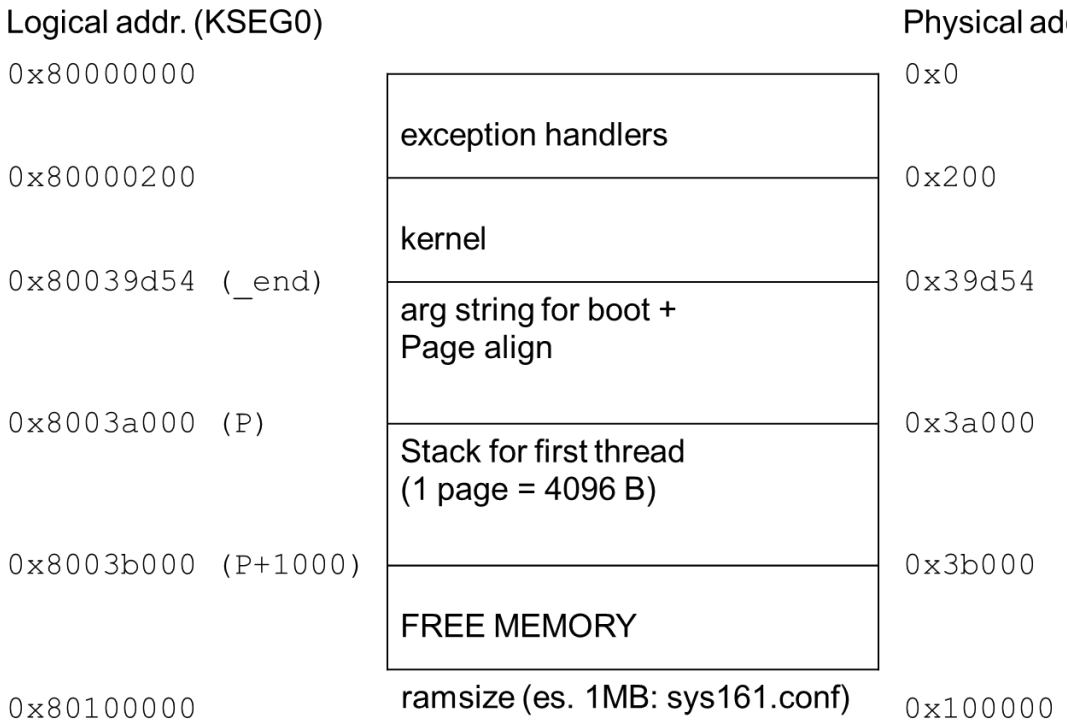
# MIPS VIRTUAL ADDRESS SPACE

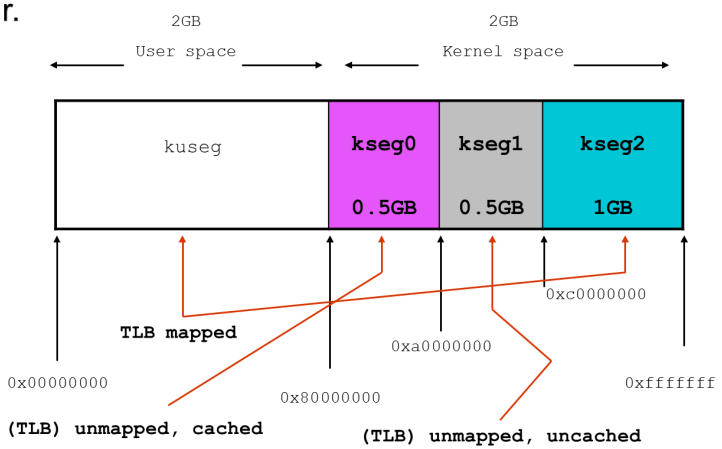



Mips maps the kernel of OS in the logical memory space of the process. So the process sees the user and also kernel in its memory space. (When boot of os161 is done, the kernel is mapped in a physical address that then later, through tlb, it is mapped in this logical memory of the process)

The kuseg is for user memory.
Kseg0 is for kernel, kseg01 I/O devices, and kseg02 is not used.

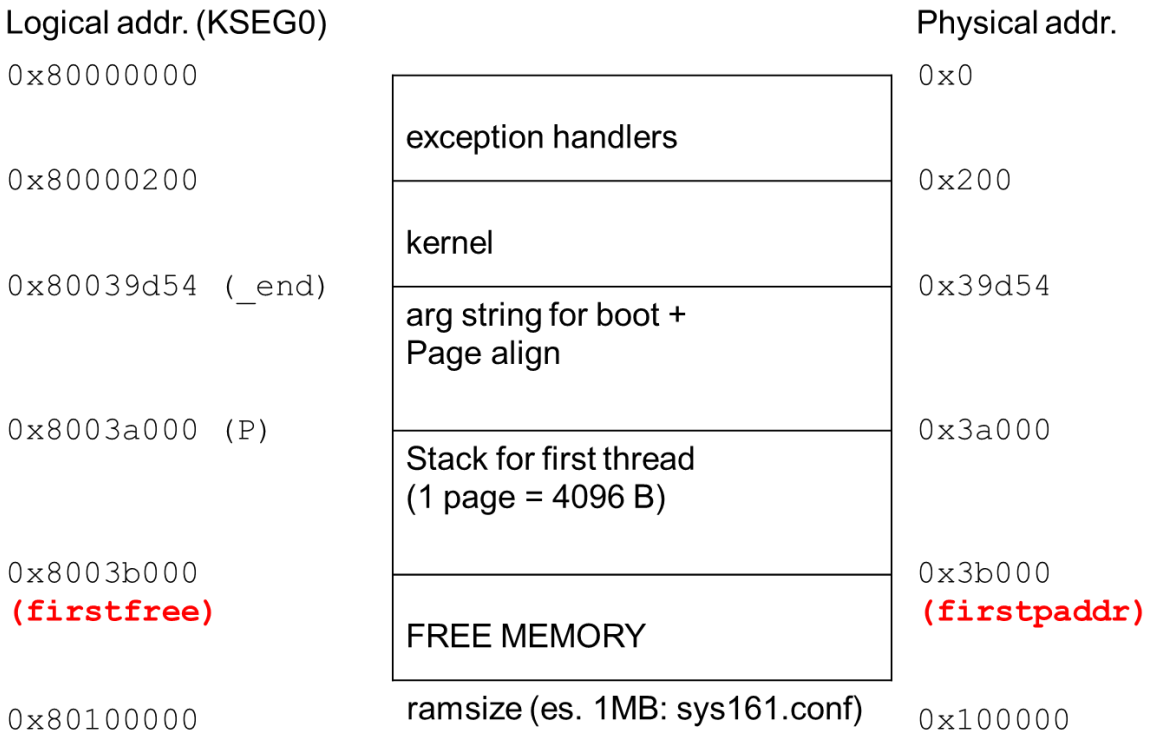
# Kernel loader (sys161: start.S)



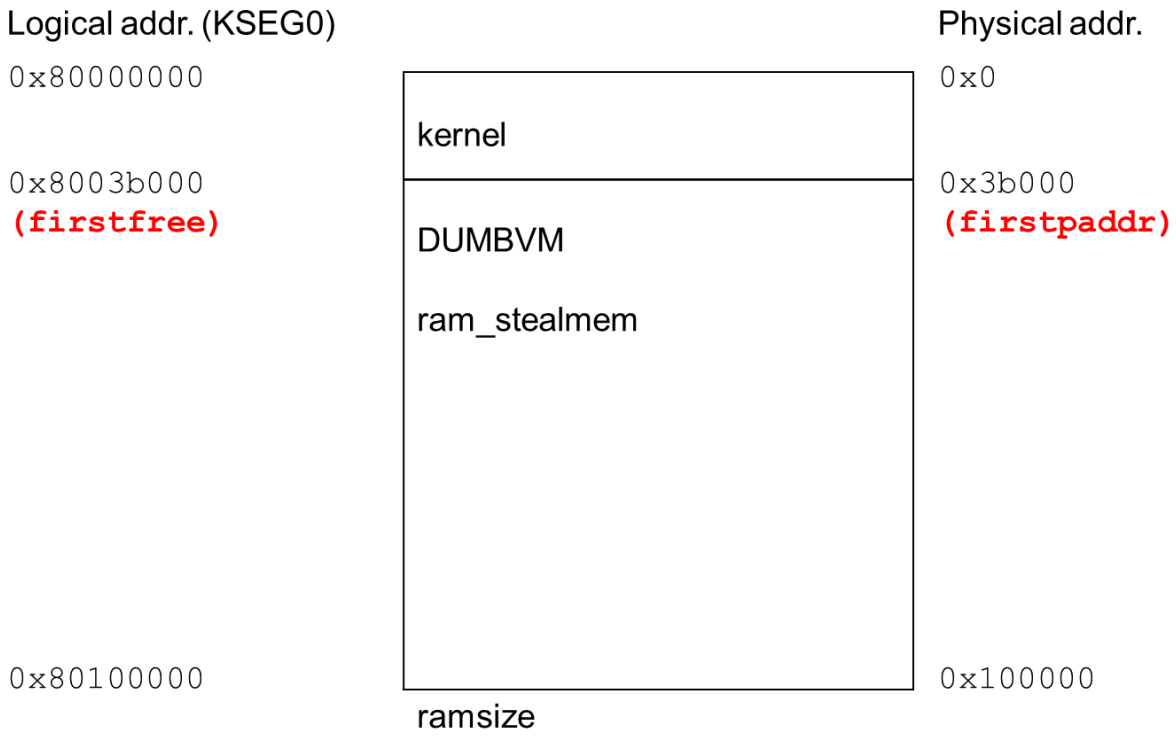

What happens during the boot time of Os161?
Using part of the memory for loading the kernel of OS161, starting from address 0. Starting from exception handlers, kernel, the memory space for the argument of command line of kernel (it is empty right now), stack for the first process that will be created, and free physical memory).

# Kernel loader (sys161: start.S)




At the end of the boot phase, OS saves the first available physical address (Free Memory)

# Dumbvm




Complete layout of physical memory:
Starting from the first free address (first physical address), arriving to the dimension of the memory.
The first free address depends on the size of the kernel while the last depends on the ram size.

# ram_bootstrap

```
void
ram_bootstrap(void) {
  /* Get size of RAM. */
  size_t ramsize = mainbus_ramsize();
  if (ramsize > 512*1024*1024) {
    ramsize = 512*1024*1024;
  }
  lastpaddr = ramsize;
  /* Get first free virtual address from where
     start.S saved it. Convert to physical address. */
  firstpaddr = firstfree - MIPS_KSEG0;
}
```

During the boot, the **ram_bootstrap** is called which is initializing different parameters such as **ramsize** using function **mainbus_ramsize**, saving **lastpaddr** as the ram size, and calculating the first free address with respect to KSEG0.

# **ram_stealmem** (kern/arch/mips/vm/ram.c)

```
paddr_t ram_stealmem(unsigned long npages) {
  paddr_t paddr;
  size_t size = npages * PAGE_SIZE;
  if(firstpaddr  + size > lastpaddr) {
    return 0;
  }
  paddr = firstpaddr;
  firstpaddr += size;
  return paddr;
}
```

In **ram.c**, the most important function is **ram_stealmem** that allows to ask for memory.

**Npages** is the number of pages to ask the OS.

Calculating the size as the multiply of number of pages and page size.

# **ram_stealmem** (kern/arch/mips/vm/ram.c)

```
paddr_t ram_stealmem(unsigned long npages) {
  paddr_t paddr;
  size_t size = npages * PAGE_SIZE;
  if(firstpaddr  + size > lastpaddr) {
    return 0;
  }
  paddr = firstpaddr;
  firstpaddr += size;
  return paddr;
}
```

Checking whether the first free address + the requested size is more than the last physical address. If yes, retuning 0 which shows failing in memory allocation

# **ram_stealmem** (kern/arch/mips/vm/ram.c)

```
paddr_t ram_stealmem(unsigned long npages) {
  paddr_t paddr;
  size_t size = npages * PAGE_SIZE;
  if(firstpaddr  + size > lastpaddr) {
    return 0;
  }
  paddr = firstpaddr;
  firstpaddr += size;
  return paddr;
}
```

Checking whether the first free address + the requested size is more than the last physical address. If not, successful memory allocation, returning the address of the first available physical free memory.

Ram.c is regarding the architecture dependent part. For example, for changing from continues allocation to another one, this module should be changed.

# **Getppages** (kern/arch/mips/vm/ram.c)

```
static paddr_t
getppages(unsigned long npages) {
  paddr_t addr;
  spinlock_acquire(&stealmem_lock);
  addr = ram_stealmem(npages);
  spinlock_release(&stealmem_lock);
  return addr;
}
```

Getppages is for asking numbers of physical pages, asking for page number and returning pointer to the zone of the memory.

# **Getppages** (kern/arch/mips/vm/ram.c)

```
static paddr_t
getppages(unsigned long npages) {
  paddr_t addr;
  spinlock_acquire(&stealmem_lock);
  addr = ram_stealmem(npages);
  spinlock_release(&stealmem_lock);
  return addr;
}
```

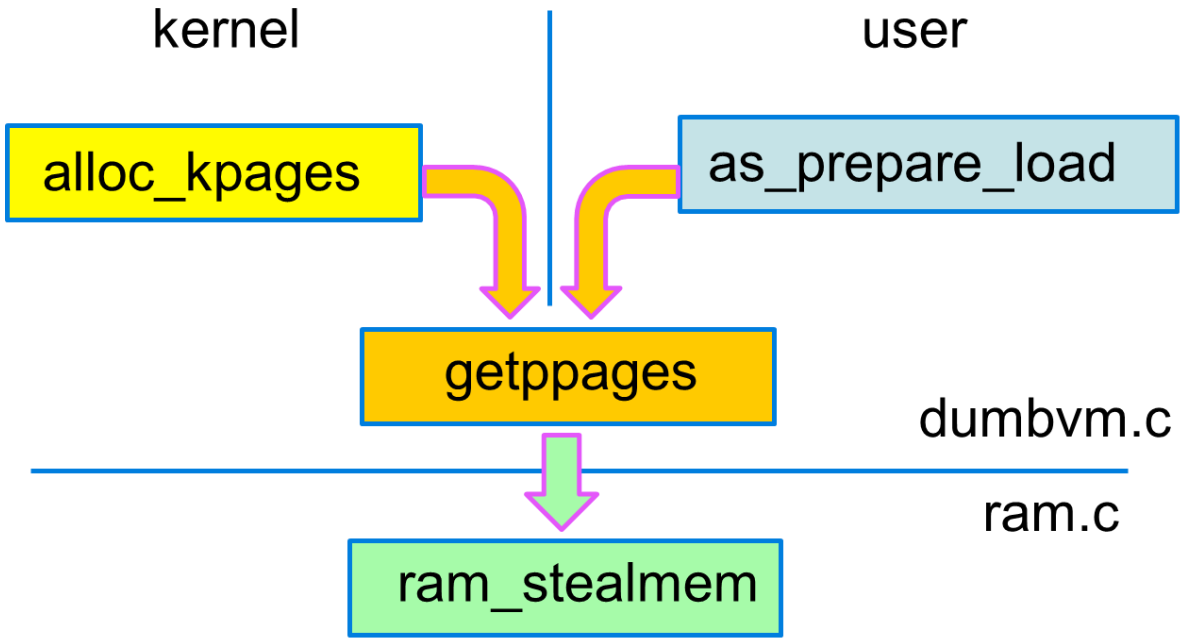
Spinlock allows the mutual exclusion in case of multiple process, to avoid corrupting of ram_stealmem function.

# **Getppages** (kern/arch/mips/vm/ram.c)

```
static paddr_t
getppages(unsigned long npages) {
  paddr_t addr;

  spinlock_acquire(&stealmem_lock);

  addr = ram_stealmem(npages);

  spinlock_release(&stealmem_lock);


  return addr;
}
```

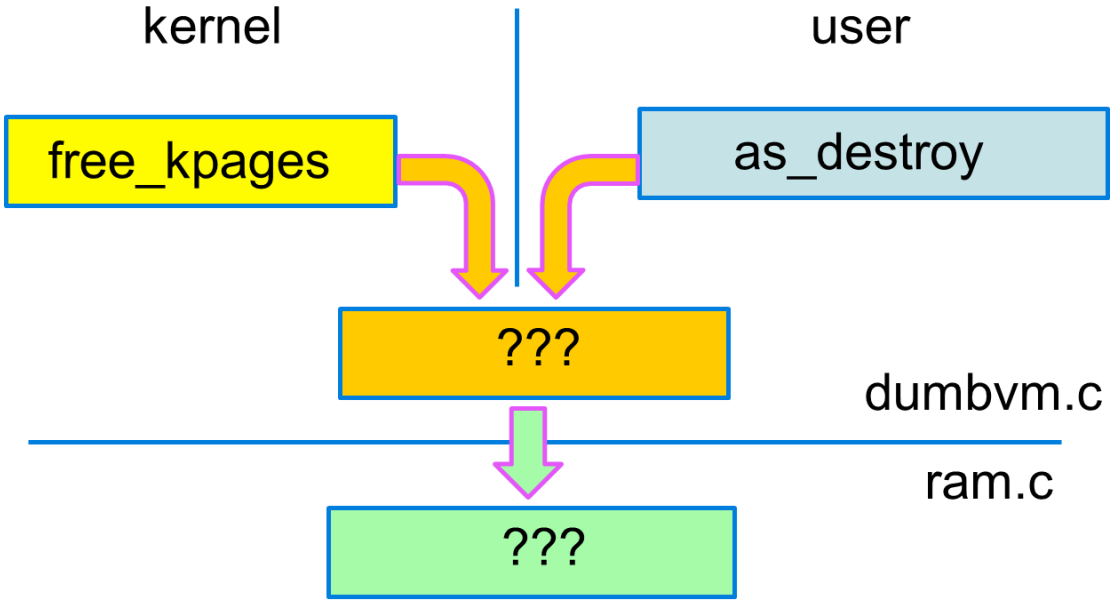
Internal (dumbvm) function

# Dumbvm.c (alloc)




Memory manager architecture of OS161:
**ram_stealmem** for memory management at low level (architecture dependent)
**getppages** for memory management in dumbvm.c
Allocation function for user side (**as_prepare_load**) and kernel side (**alloc_kpages**)

# Dumbvm.c – Not implemented >> to do




The function for allocation of memory is implemented BUT the functions for releasing memory is not implemented.

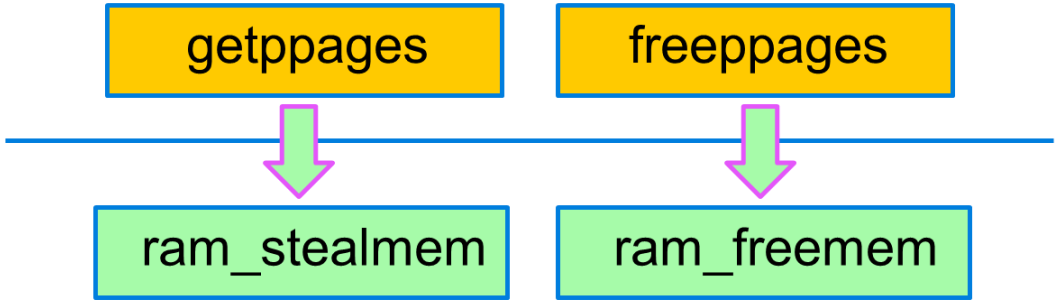
**free_kpages** and **as_destroy** (equivalent of alloc_kpages and as_prepare_load) is defined but EMPTY.

Your Job to solve this issue.

# De-alloc (free) in ram.c

*(solution A1)*

- `freeppages` just an interface to `ram_freemem`
- Data structure (free-list or bitmap) and ***memory management in ram.c***

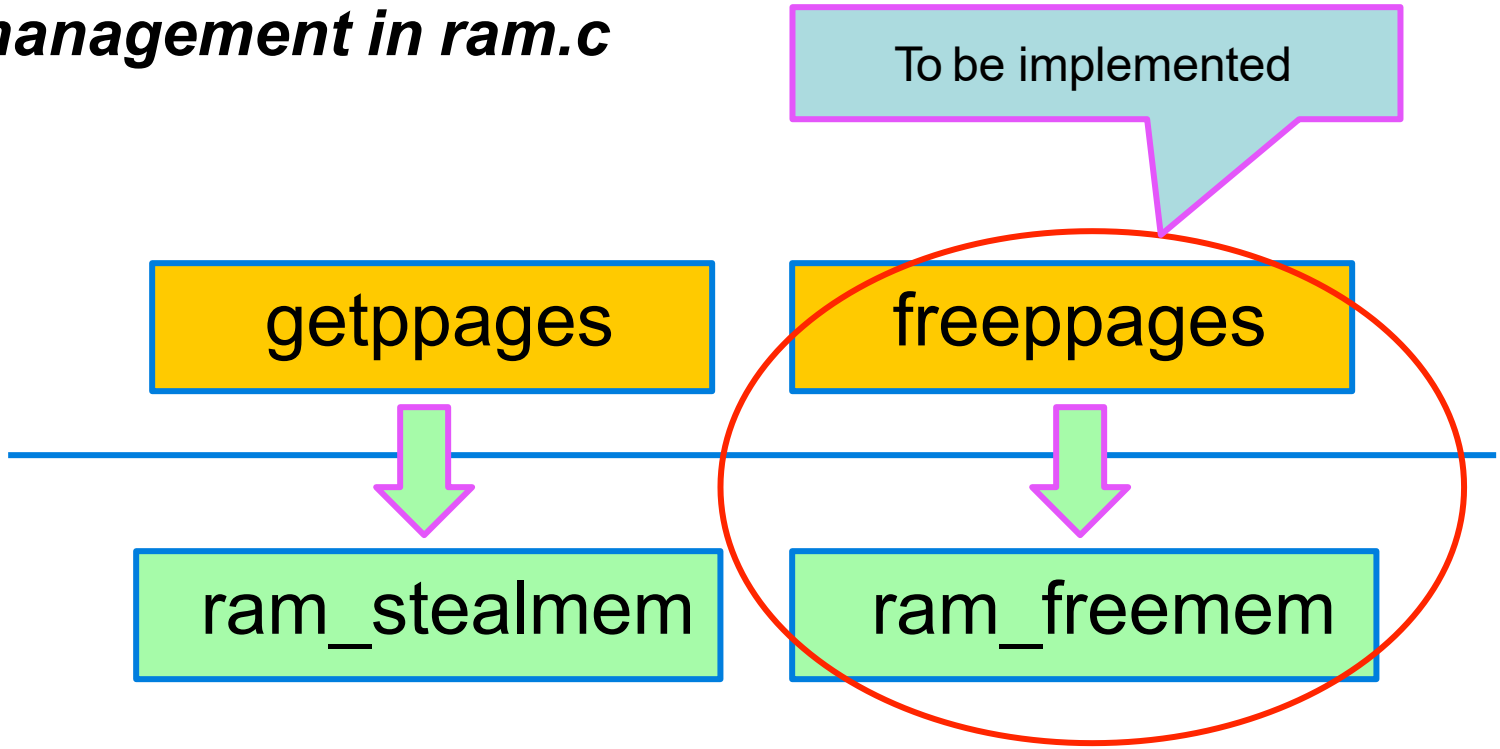



Defining the equivalent system of **ram_stealmem** and **getppages**,
Defining a **freeppages** that we be called **by free_kpages** and **as_destroy** (high level) that will call **ram_freemem** for freeing memory.

# De-alloc (free) in ram.c

*(solution A1)*

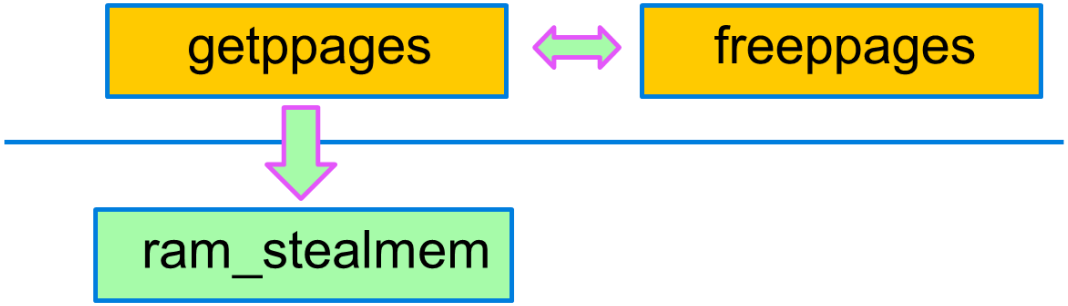
- `freeppages` just an interface to `ram_freemem`
- Data structure (free-list or bitmap) and ***memory management in ram.c***



# De-alloc (free) in ram.c

*(solution A2)*

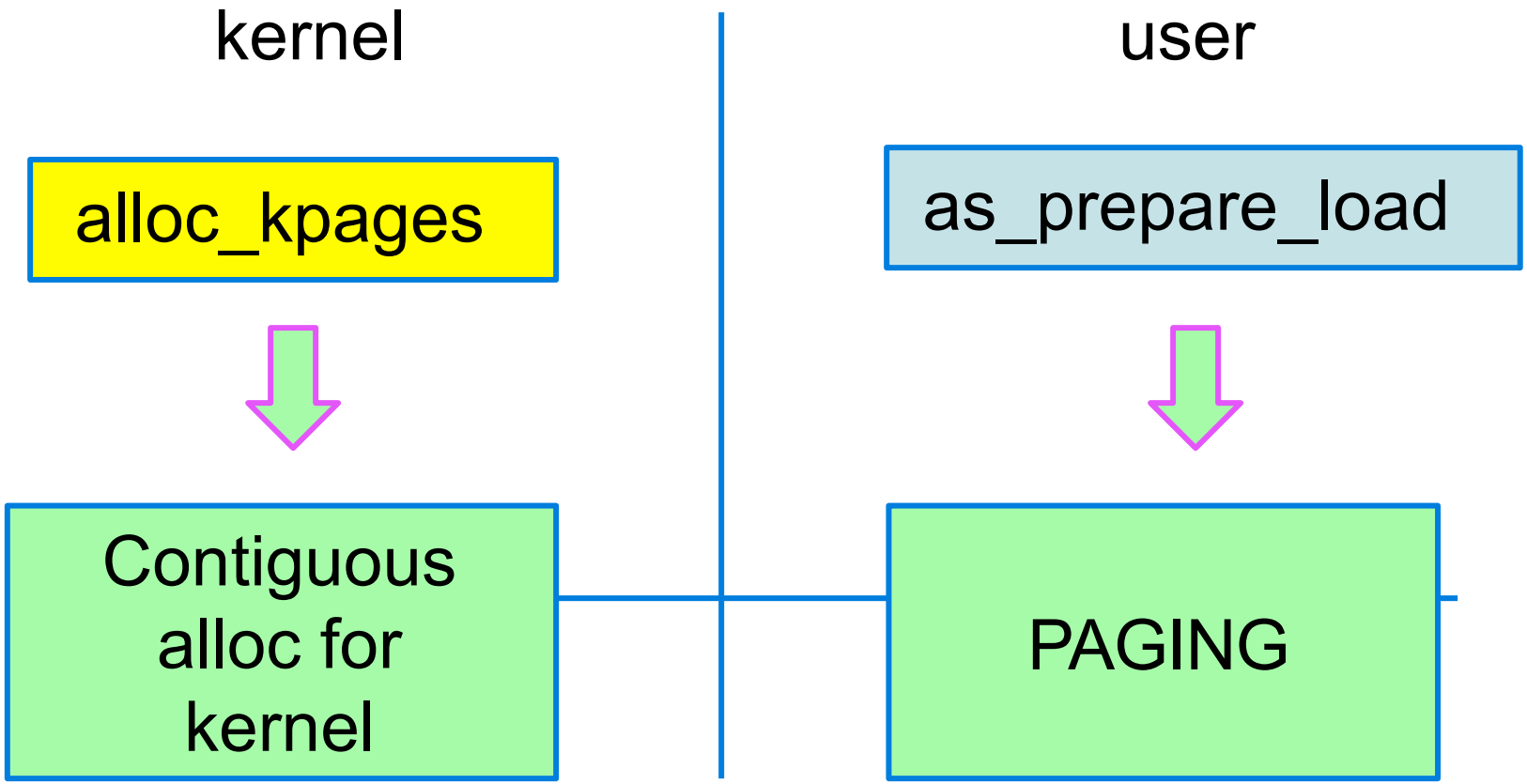
- Memory not returned to RAM
- Data structure (free-list o bitmap) and ***memory management in dumbvm.c***
- `Freeppages` coordinates with `getppages`




Managing the liberation of memory at low level but just at the dumbvm.c level.
Eveytime that we are asking for memory allocation, before asking ram_stealmem, looking if the memory page that has been already allocated, is free now.
Vantages: working with one system call

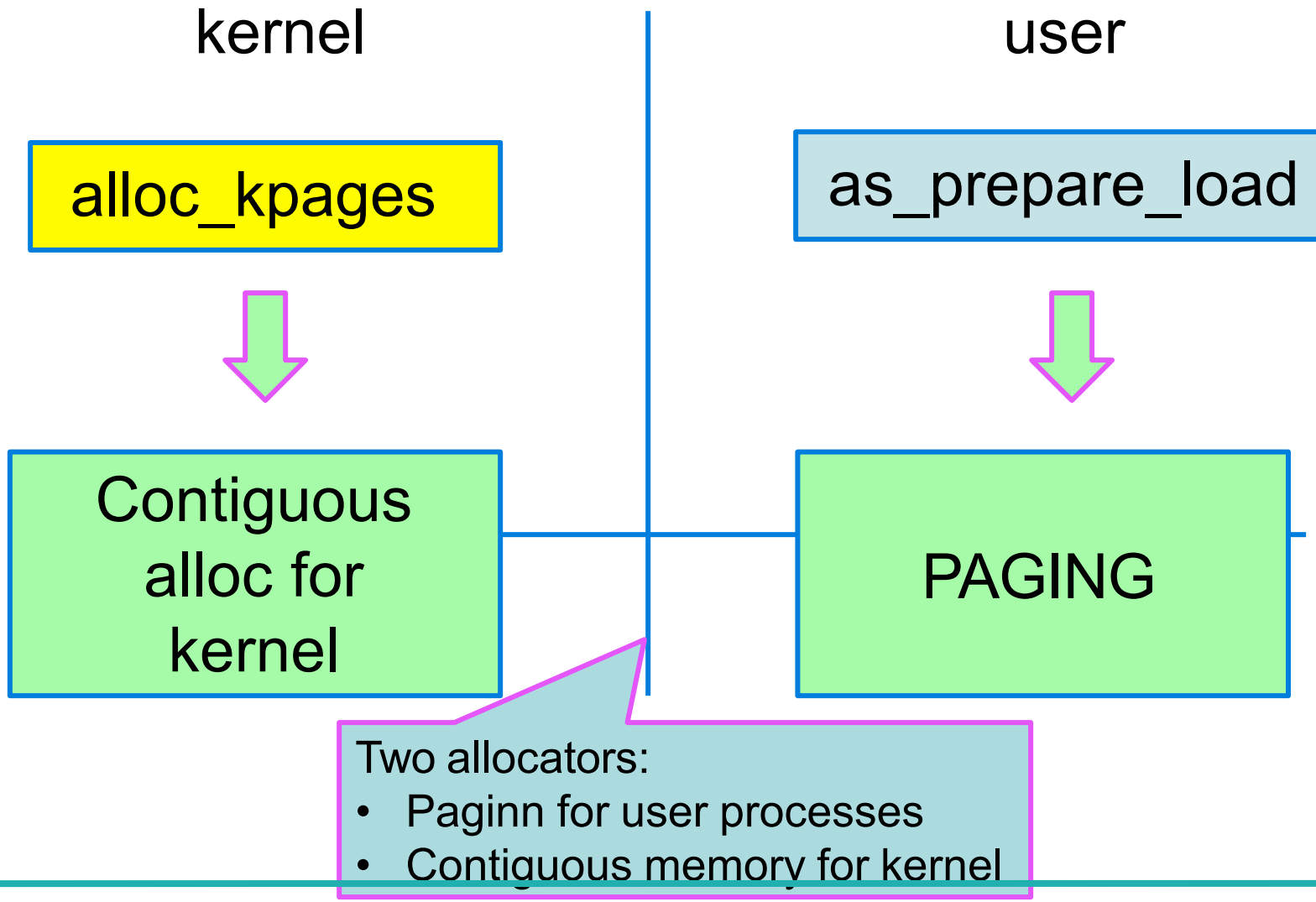
# Paging in user space

*(solution B)*



# Paging in user space

*(solution B)*



# Proposed Solution

## *de-alloc in dumbvm (sol. A2)*

- Contiguous allocation(by pages) common to kernel and user
- Allocator in dumbvm: keep track (using a bitmap) of previously freed pages. In order to alloc
  - When calling **getppages**, first search among (previously) freed pages **(an interval of contiguous free pages)**
  - **If not found, call ram_stealmem**
- Bitmap implemented as an array of char (for simiplicity)
  - `freeRamFrames[i] = 1/0` (free/alloated): free=***FREED!*** (by ***freeppages***)
- In order to free we need to know
  - Pointer (or index) to first page in interval
  - Size, i.e. number of (contiguous) pages to free
- We need a table to store sizes (number of pages in allocated intervals) for each alloc performad
  - `void free_kpages(vaddr_t addr)`: table needed as only pointer passed
  - `void as_destroy(struct addrspace *as)`: table not needed ad size is stored in address space
- `allocSize[i] = /* numbeer of pages allocated starting at i-th */`

# Global Variables and Test Function

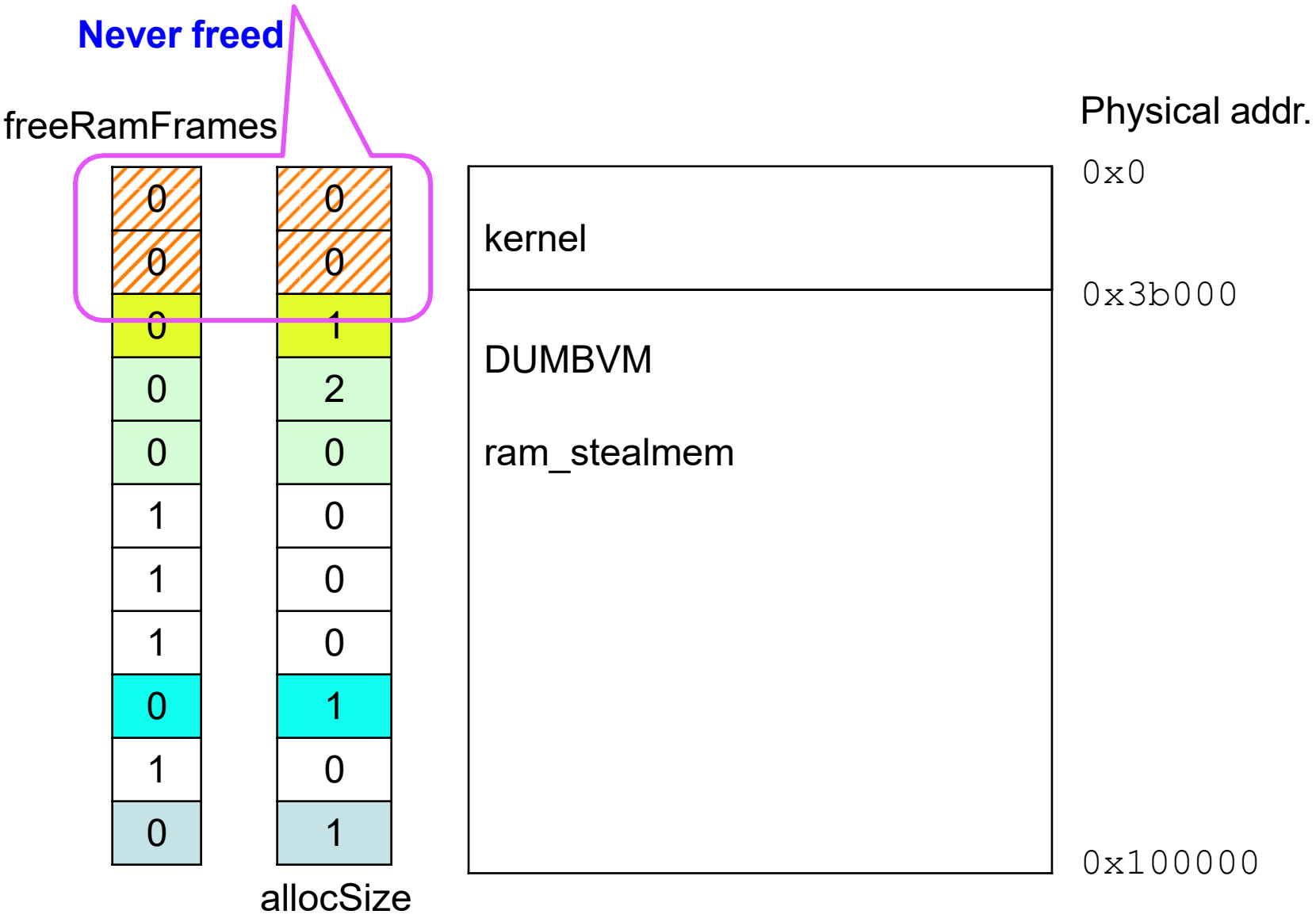
```
static struct spinlock freemem_lock = SPINLOCK_INITIALIZER;

static unsigned char *freeRamFrames = NULL;
static unsigned long *allocSize = NULL;
static int nRamFrames= 0;

static int allocTableActive = 0;

static int isTableActive () {
  int active;
  spinlock_acquire(&freemem_lock);
  active = allocTableActive;
  spinlock_release(&freemem_lock);
  return active;
}
```

# Global Variables and Test Function

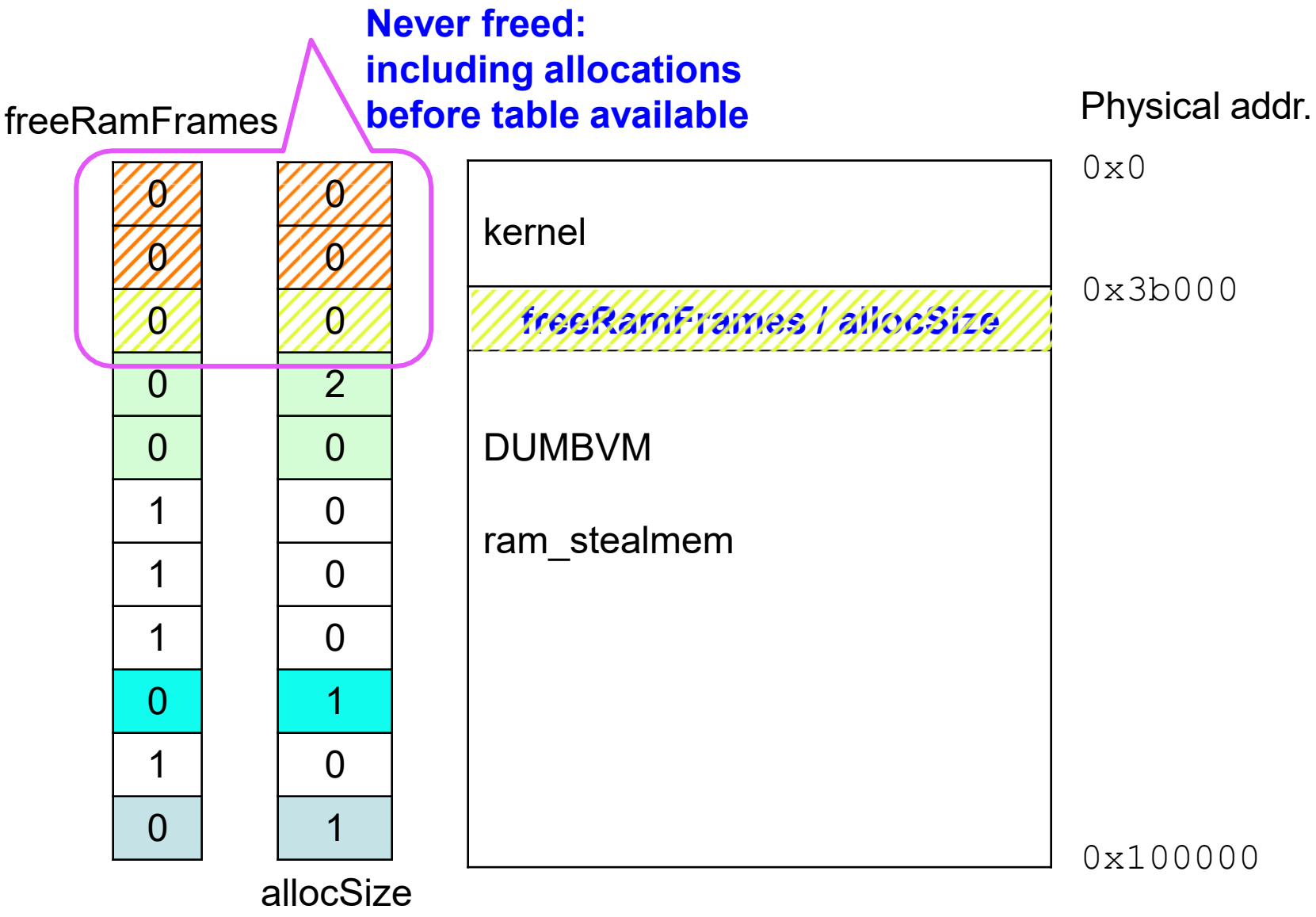
```
static struct spinlock freemem_lock = SPINLOCK_INITIALIZER;

static unsigned char *freeRamFrames = NULL;
static unsigned long *allocSize = NULL;
static int nRamFrames= 0;

static int allocTable      ve = 0;

static int
  int active;
  spinlock_
  active = allocTableActive;
  spinlock_release(&freemem_lock);
  return active;
}
```

Dynamic arrays as RAM size known at Boot (depends on da sys161.conf)
Alternative: over-dimensioned static arrays!

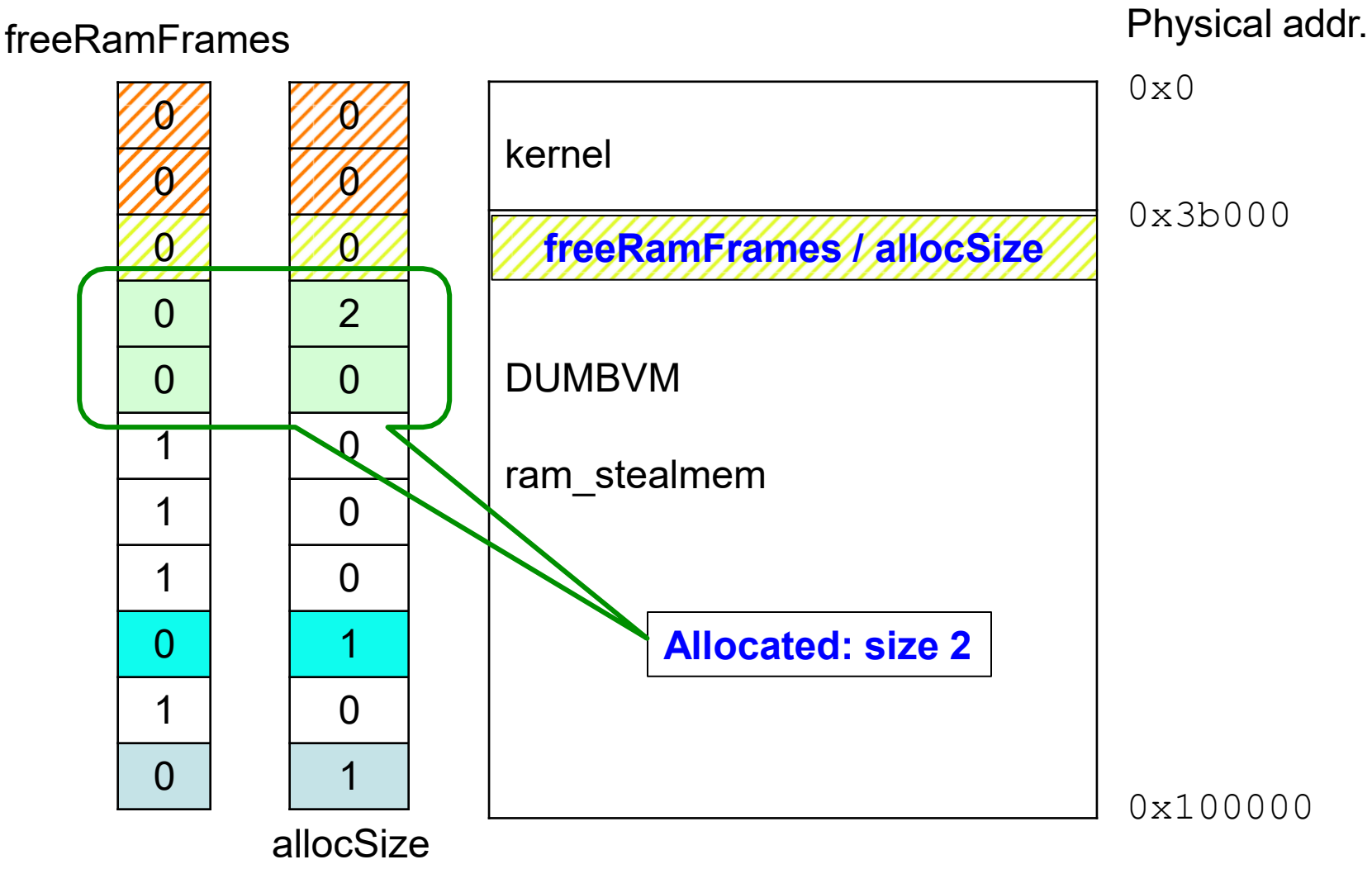
# Dumbvm


Never freed
freeRamFrames
0
0
0
0
0
1
1
1
1
0
1
0
0
0
2
0
0
0
0
1
0
1
allocSize
kernel
DUMBVM
ram_stealmem
Physical addr.
0x0
0x3b000
0x100000

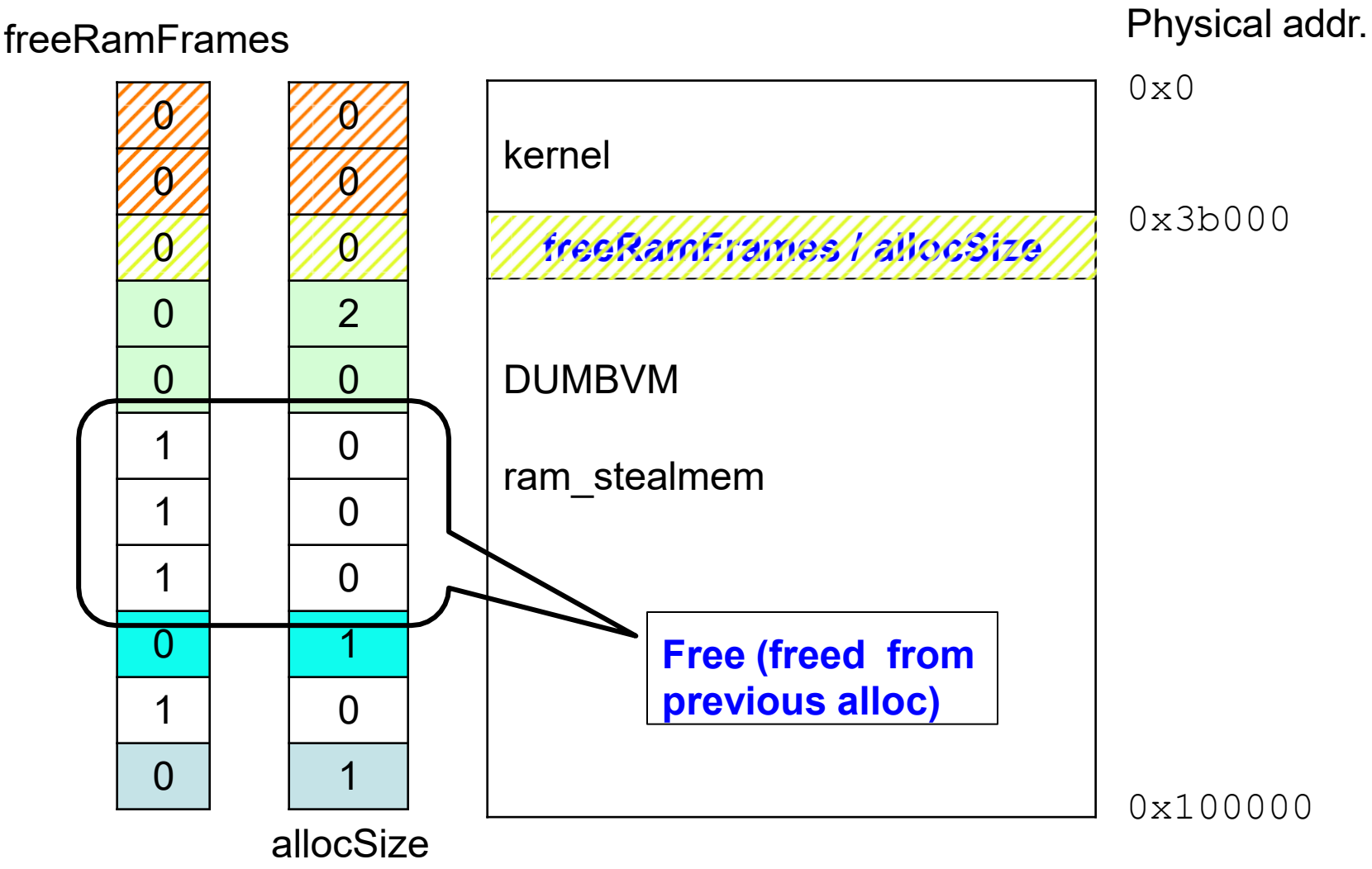
# Dumbvm


Never freed:
including allocations
before table available
freeRamFrames
0
0
0
0
0
1
1
1
0
1
0
0
0
0
2
0
0
0
0
1
0
1
allocSize
kernel
freeRamFrames / allocSize
DUMBVM
ram_stealmem
Physical addr.
0x0
0x3b000
0x100000

# Dumbvm


freeRamFrames
0
0
0
0
0
1
1
1
0
1
0
allocSize
0
0
0
2
0
0
0
0
1
0
1
kernel
freeRamFrames / allocSize
DUMBVM
ram_stealmem
Allocated: size 2
Physical addr.
0x0
0x3b000
0x100000

# Dumbvm


freeRamFrames
0
0
0
0
0
1
1
1
0
1
0
0
0
0
2
0
0
0
0
1
0
1
allocSize
kernel
freeRamFrames / allocSize
DUMBVM
ram_stealmem
Free (freed from previous alloc)
Physical addr.
0x0
0x3b000
0x100000

# Free_kpages & as_destroy

```
void free_kpages(vaddr_t addr){
   if (isTableActive()) {
     paddr_t paddr = addr - MIPS_KSEG0;
     long first = paddr/PAGE_SIZE;
     KASSERT(nRamFrames>first);
     freeppages(paddr, allocSize[first]);
   }
}
void as_destroy(struct addrspace *as){
  dumbvm_can_sleep();
  freeppages(as->as_pbase1, as->as_npages1);
  freeppages(as->as_pbase2, as->as_npages2);
  freeppages(as->as_stackpbase, DUMBVM_STACKPAGES);
  kfree(as);
}
```

# Initialization

```
void vm_bootstrap(void) {
  int i;
  nRamFrames = ((int)ram_getsize())/PAGE_SIZE;
  /* alloc freeRamFrame and allocSize */
  freeRamFrames = kmalloc(sizeof(unsigned char)*nRamFrames);
  allocSize     = kmalloc(sizeof(unsigned long)*nRamFrames);
  if (freeRamFrames==NULL || allocSize==NULL) {
    /* reset to disable this vm management */
    freeRamFrames = allocSize = NULL; return;
  }
  for (i=0; i<nRamFrames; i++) {
    freeRamFrames[i] = (unsigned char)0; allocSize[i] = 0;
  }
  spinlock_acquire(&freemem_lock);
  allocTableActive = 1;
  spinlock_release(&freemem_lock);
 }
```

# getppages

```
static paddr_t getppages(unsigned long npages) {
  paddr_t addr;

  /* try freed pages first */
  addr = getfreeppages(npages);
  if(addr== 0) {/* call stealmem   */
    spinlock_acquire(&stealmem_lock);
    addr = ram_stealmem(npages);
    spinlock_release(&stealmem_lock);
  }
  if(addr  != 0 && isTableActive()) {
    spinlock_acquire(&freemem_lock);
    allocSize[addr/PAGE_SIZE] = npages;
    spinlock_release(&freemem_lock);
  }
  return addr;
}
```

# getfreeppages

```
static paddr_t getfreeppages(unsigned long npages) {
  paddr_t addr;
  long i, first, found, np = (long)npages;

  if (!isTableActive()) return 0;
  spinlock_acquire(&freemem_lock);
  // Linear search of free interval
  for (i=0,first=found=-1; i<nRamFrames; i++) {
    if (freeRamFrames[i]) {
      if (i==0 || !freeRamFrames[i-1])
        first = i; /* set first free in an interval */
      if (i-first+1 >= np)
        found = first;
    }
  }
```

# getfreeppages

```
  if (found>=0) {
    for (i=found; i<found+np; i++) {
      freeRamFrames[i] = (unsigned char)0;
    }
    allocSize[found] = np;
    addr = (paddr_t) found*PAGE_SIZE;
  }
  else {
    addr = 0;
  }

  spinlock_release(&freemem_lock);

  return addr;
}
```

# getfreeppages

```
static int freeppages(paddr_t addr, unsigned long npages){
  long i, first, np=(long)npages;
  if (!isTableActive()) return 0;
  first = addr/PAGE_SIZE;
  KASSERT(allocSize!=NULL);
  KASSERT(nRamFrames>first);

  spinlock_acquire(&freemem_lock);
  for (i=first; i<first+np; i++) {
    freeRamFrames[i] = (unsigned char)1;
  }
  spinlock_release(&freemem_lock);

  return 1;
}
```