OS161

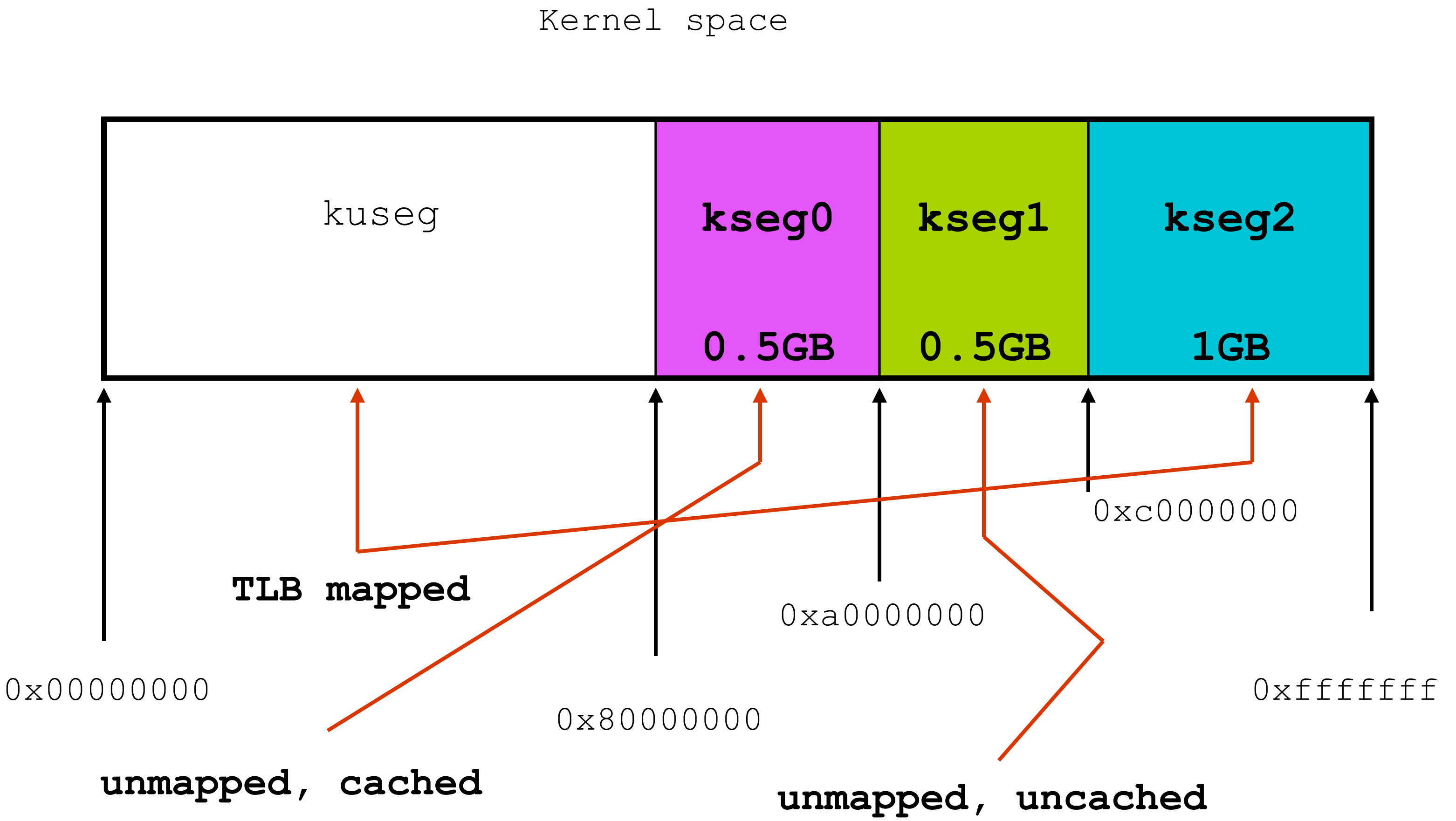
Address Space & Memory Management

Dumbvm and kmalloc

- Contiguous allocation
- Allocating by page multiples (4096 byte frame)
  - getppages (dumbvm.c): calls ram_stealmem (in mutual exclusion)
  - ram_stealmem (ram.c): allocates contiguous RAM starting at firstpaddr, that is increased
- Allocator is common to
  - 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

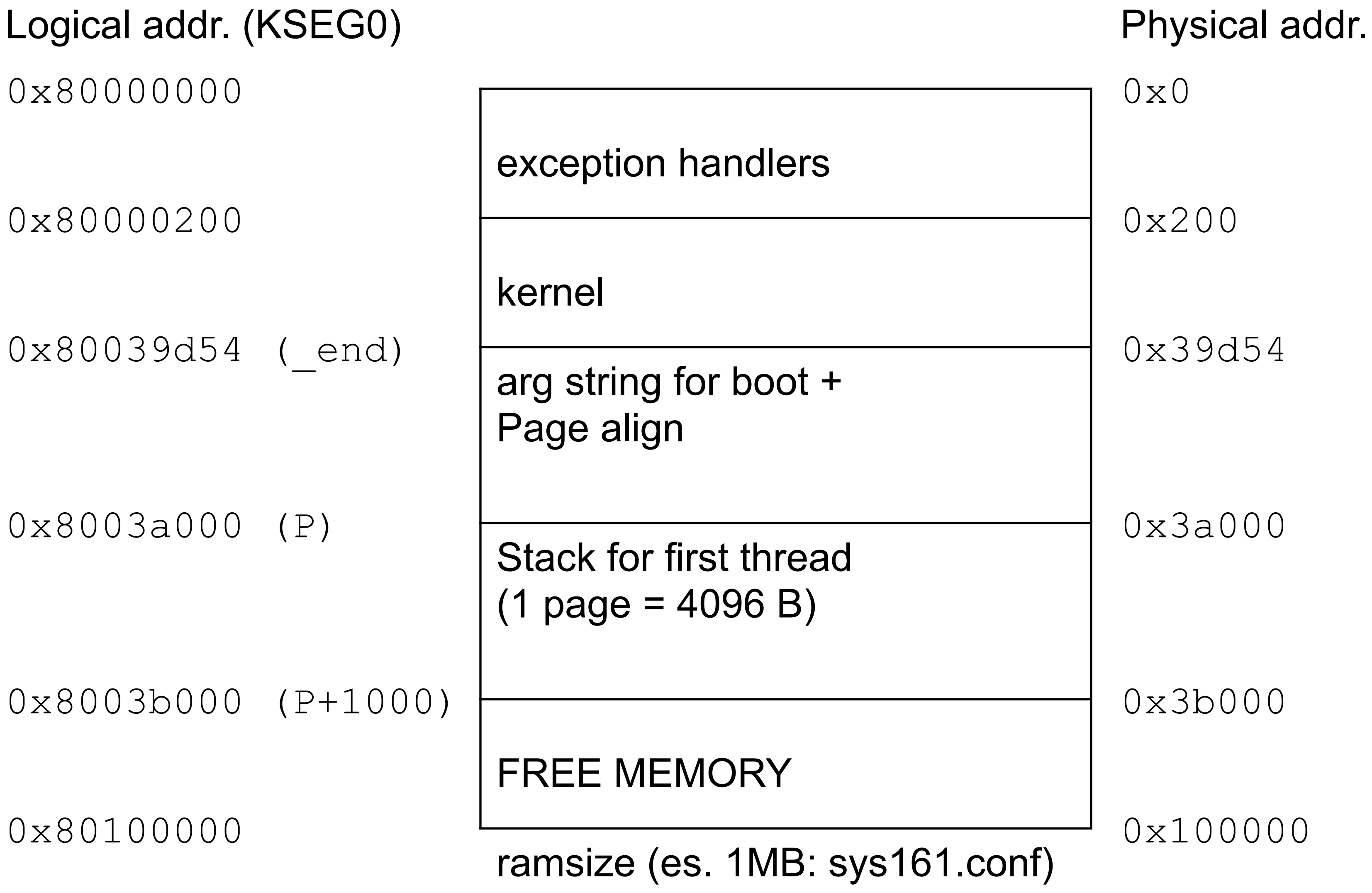
mips is a 32 bits microprocessor means 4gigs, 2 for kernel and 2 for? Difference is the first bit, the most significant bit in the 32 bits. So kuseg is the user segment which is user memory. There are microprocessors 1000 for users and 1000 for processes.

MIPS VIRTUAL ADDRESS SPACE



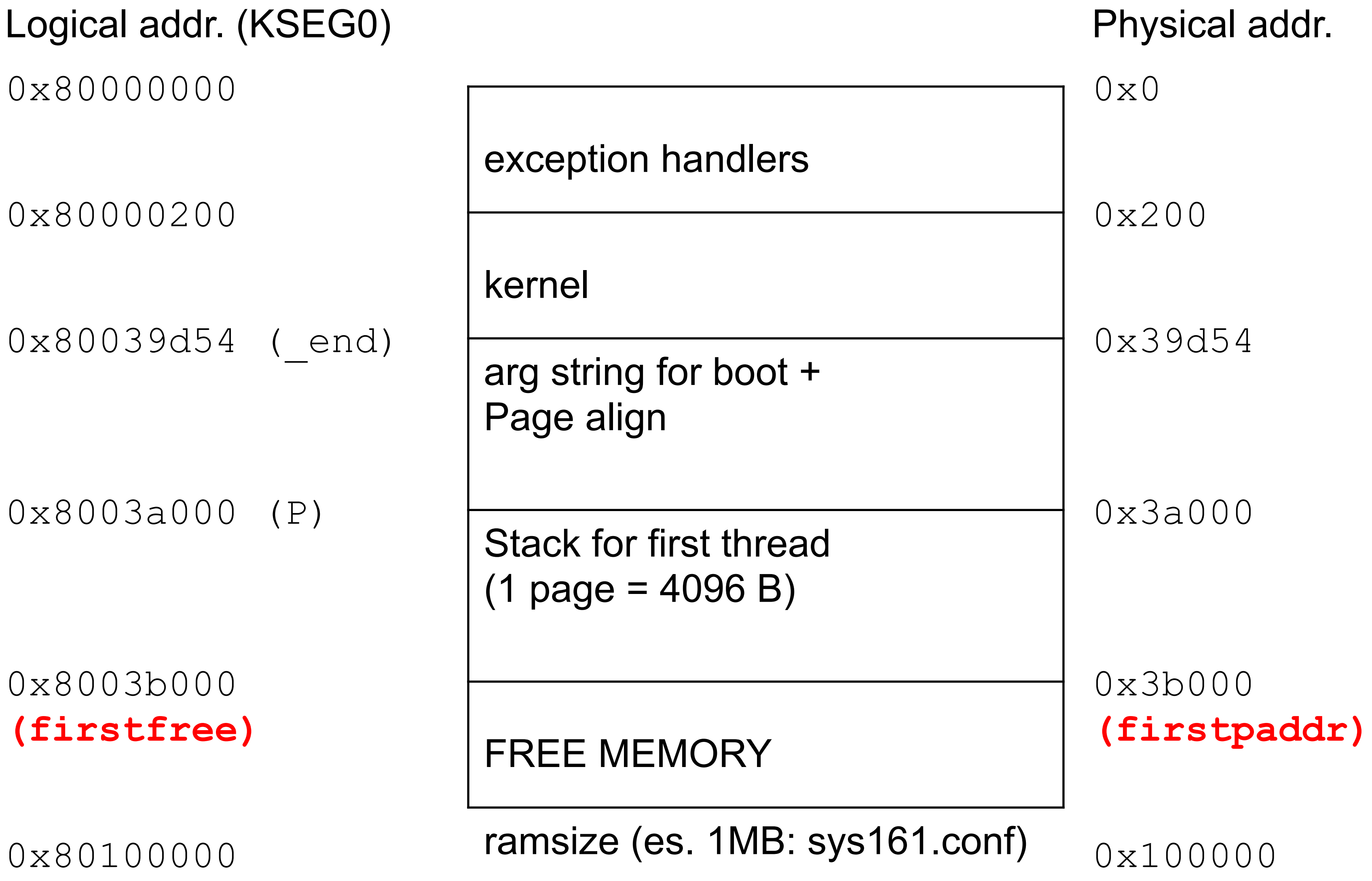
In OS/161, user programs live in kuseg, kernel code and data structures live in kseg0, devices are accessed through kseg1, and kseg2 is not used.

Kernel loader (sys161: start.S)



Logical addr. (KSEG0)		Physical addr.
0x80000000		0x0
	exception handlers	
0x80000200		0x200
	kernel	
0x80039d54 (_end)		0x39d54
	arg string for boot + Page align	
0x8003a000 (P)		0x3a000
	Stack for first thread (1 page = 4096 B)	
0x8003b000 (P+1000)		0x3b000
	FREE MEMORY	
0x80100000	ramsize (es. 1MB: sys161.conf)	0x100000

Kernel loader (sys161: start.S)

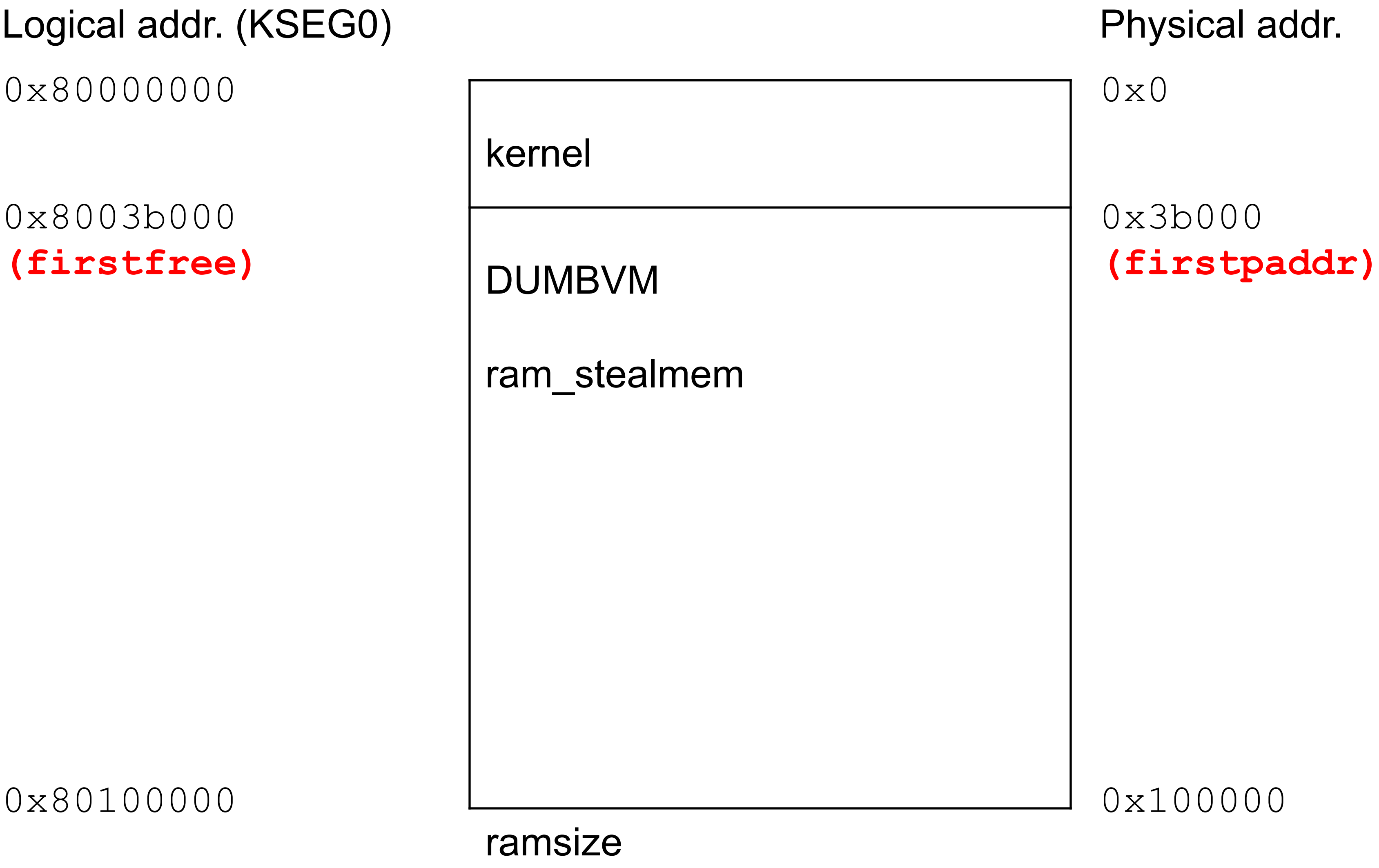


Logical addr. (KSEG0)		Physical addr.
`0x80000000`		`0x0`
	exception handlers	
`0x80000200`		`0x200`
	kernel	
`0x80039d54 (_end)`		`0x39d54`
	arg string for boot + Page align	
`0x8003a000 (P)`		`0x3a000`
	Stack for first thread (1 page = 4096 B)	

`0x8003b000`
`(firstfree)`
		`0x3b000`
		`(firstpaddr)`

	FREE MEMORY	
`0x80100000`	ramsize (es. 1MB: sys161.conf)	`0x100000`

Dumbvm



Logical addr. (KSEG0)

Physical addr.

0x80000000

0x0

kernel

0x8003b000 **(firstfree)**

0x3b000 **(firstpaddr)**

DUMBVM

ram_stealmem

0x80100000

0x100000

ramsize

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;
}
```

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;
}
```

getppages (kern/arch/mips/vm/dumbvm.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 (kern/arch/mips/vm/dumbvm.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;
}
```

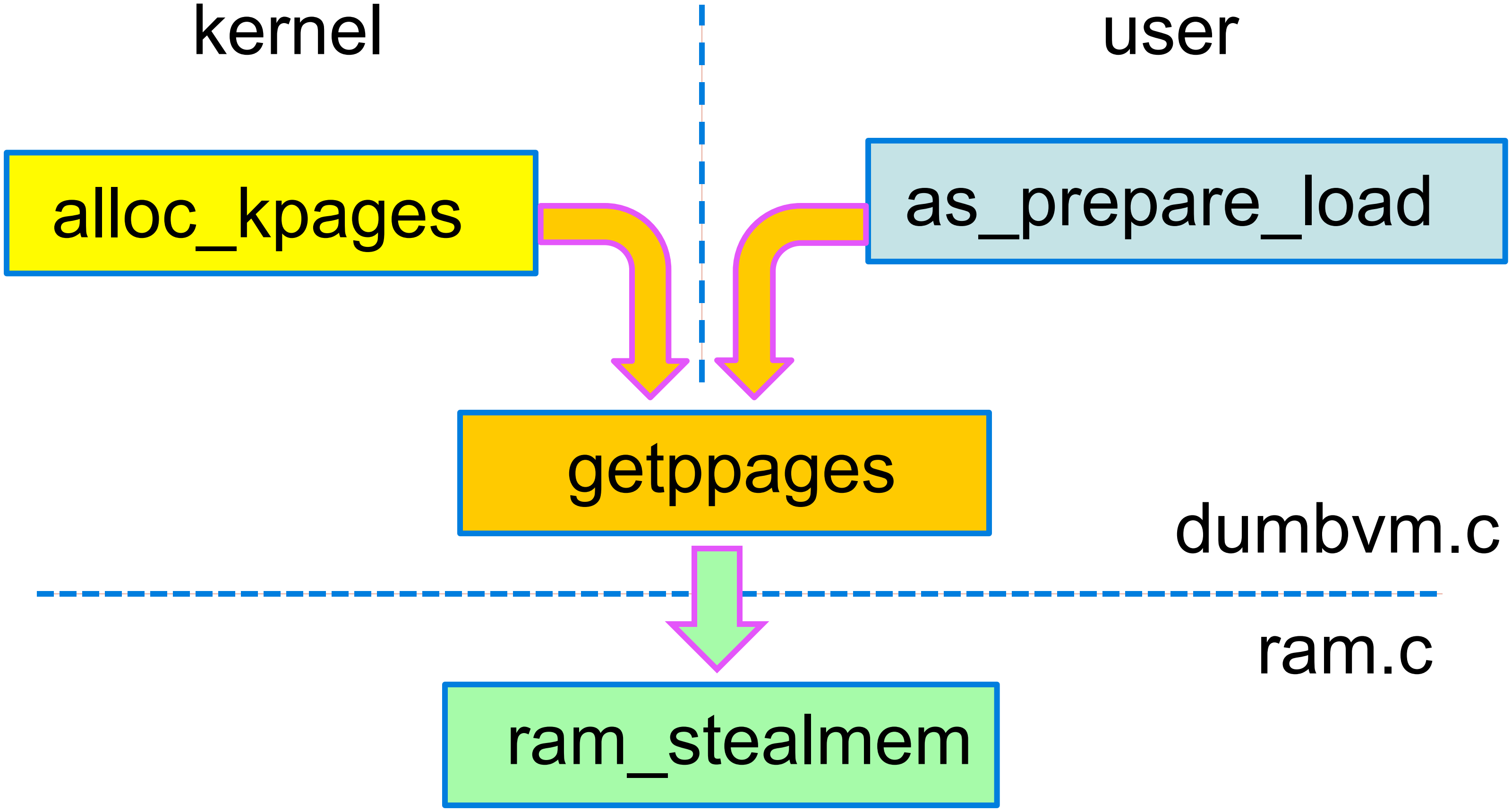
Internal (dumbvm) function

getppages (kern/arch/mips/vm/dumbvm.c)

```
static paddr_t
getppages(unsigned long npage
  paddr_t addr;

  spinlock_acquire(&stealmem_lock);

  addr = ram_stealmem(npages);

  spinlock_release(&stealmem_lock);

  return addr;
}
```

Mutual exclusion for ram_stealmem

dumbvm.c (alloc)



kernel

user

alloc_kpages

as_prepare_load

getppages

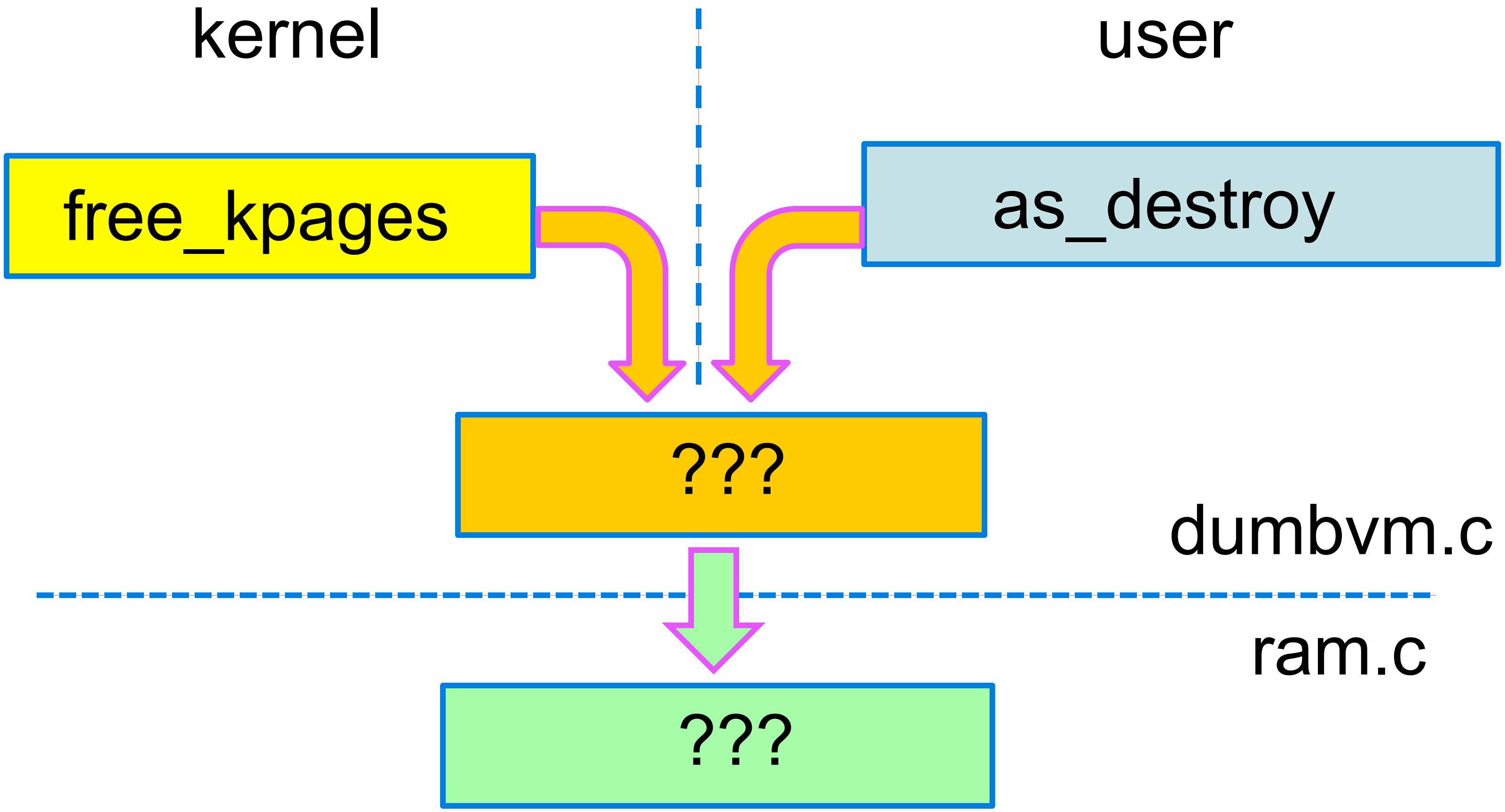
dumbvm.c

ram.c

ram_stealmem

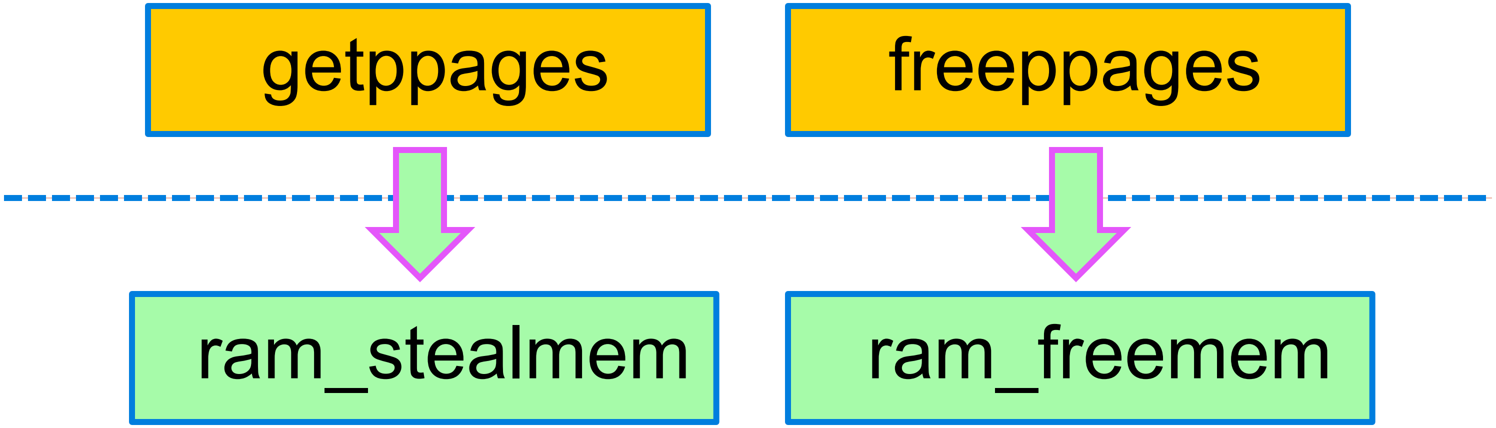
dumbvm.c (free)

not implemented -> TO DO!



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***



getppages

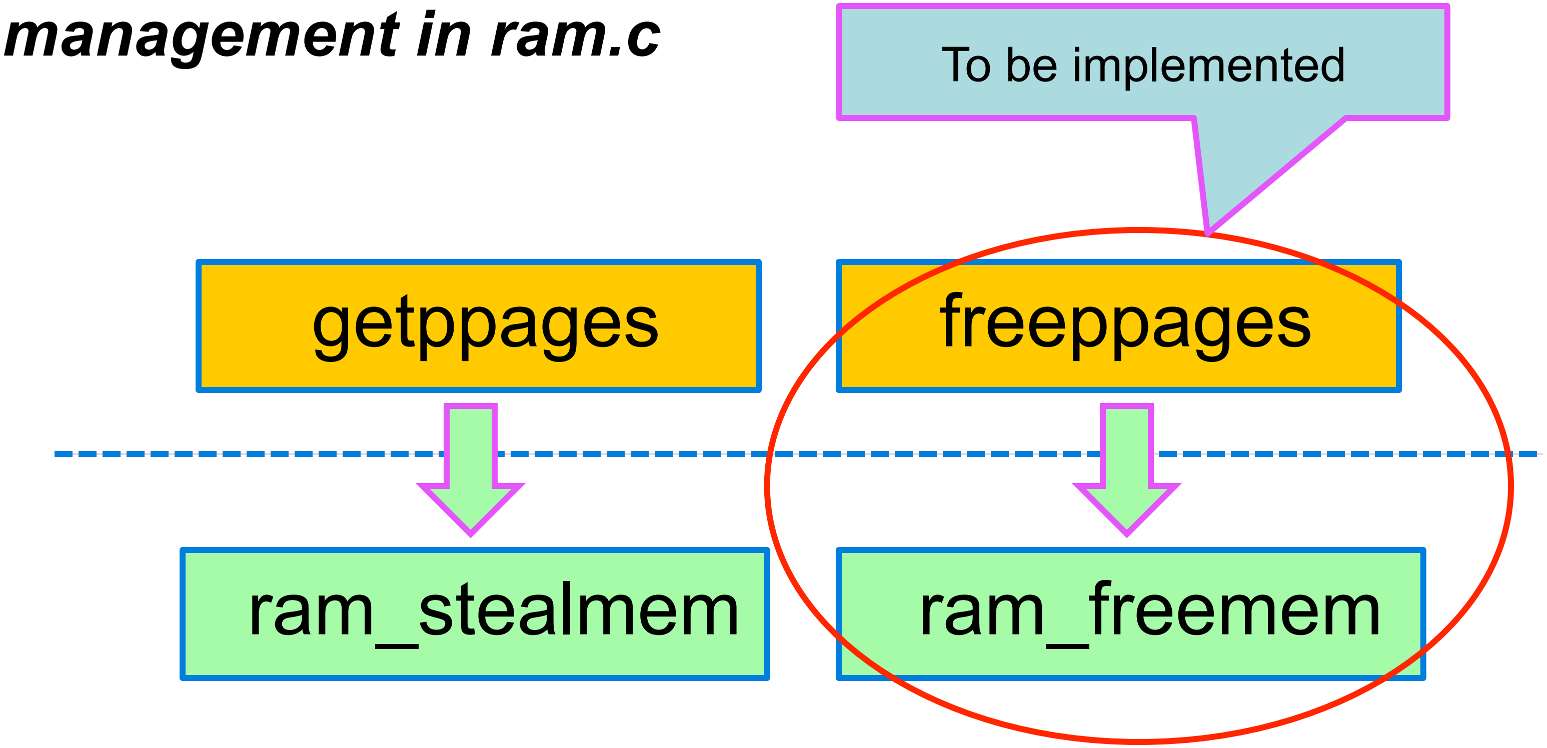
freeppages

ram_stealmem

ram_freemem

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***



To be implemented

getppages

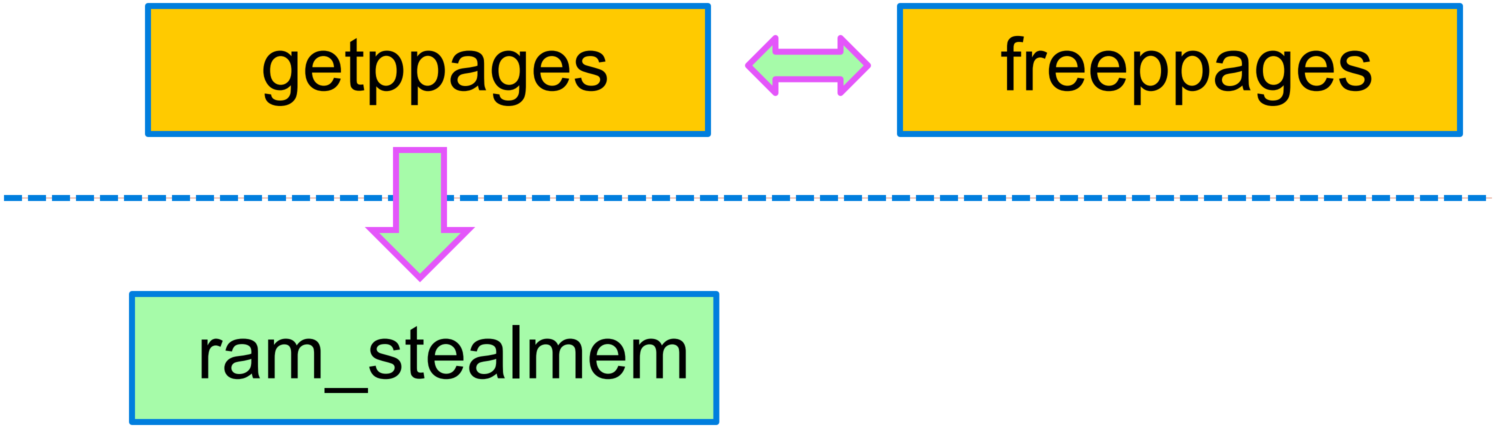
freeppages

ram_stealmem

ram_freemem

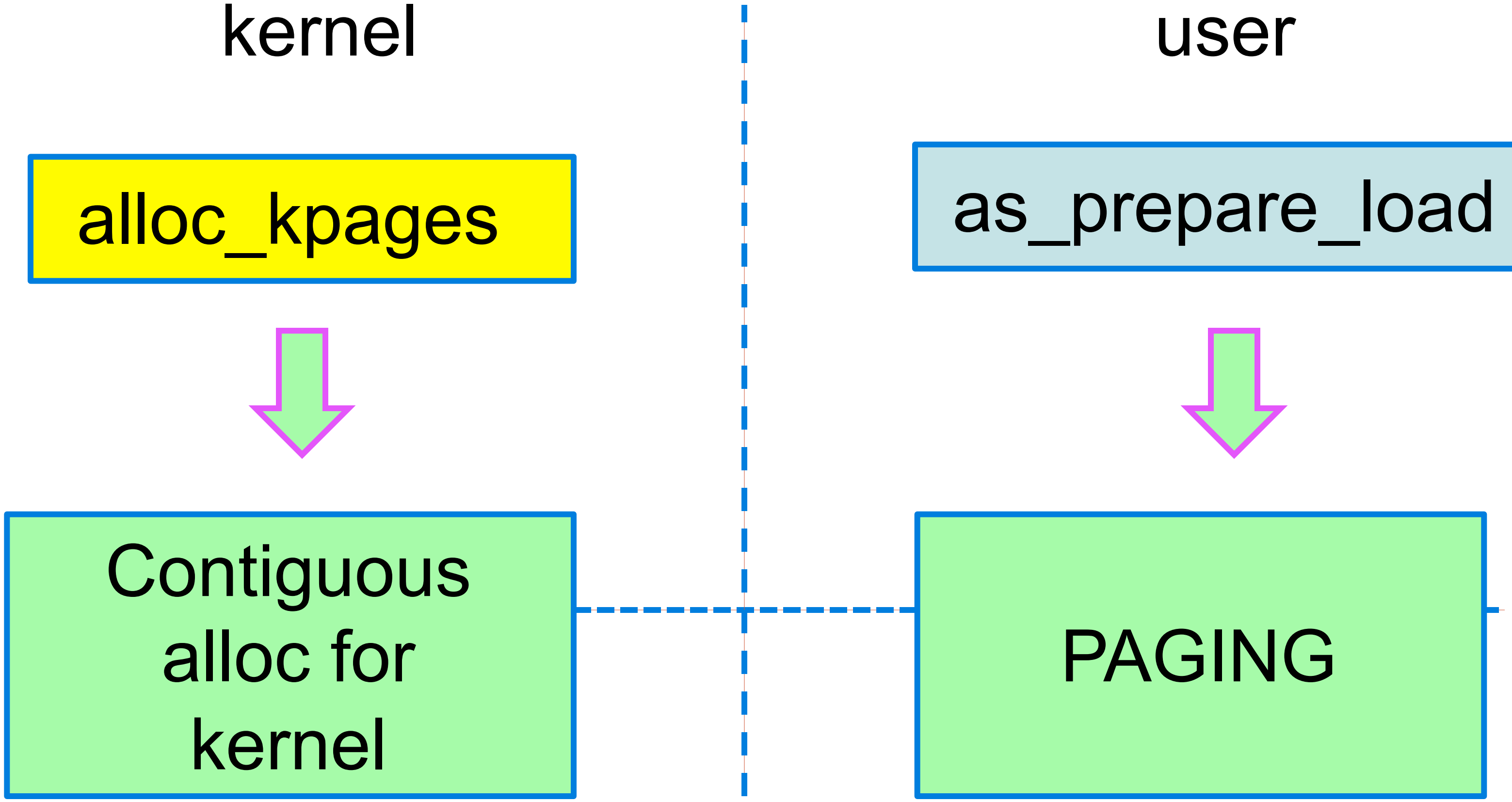
De-alloc (free) in ram.c (solution A2)

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



Paging in user space

(solution B)



kernel

alloc_kpages

Contiguous alloc for kernel

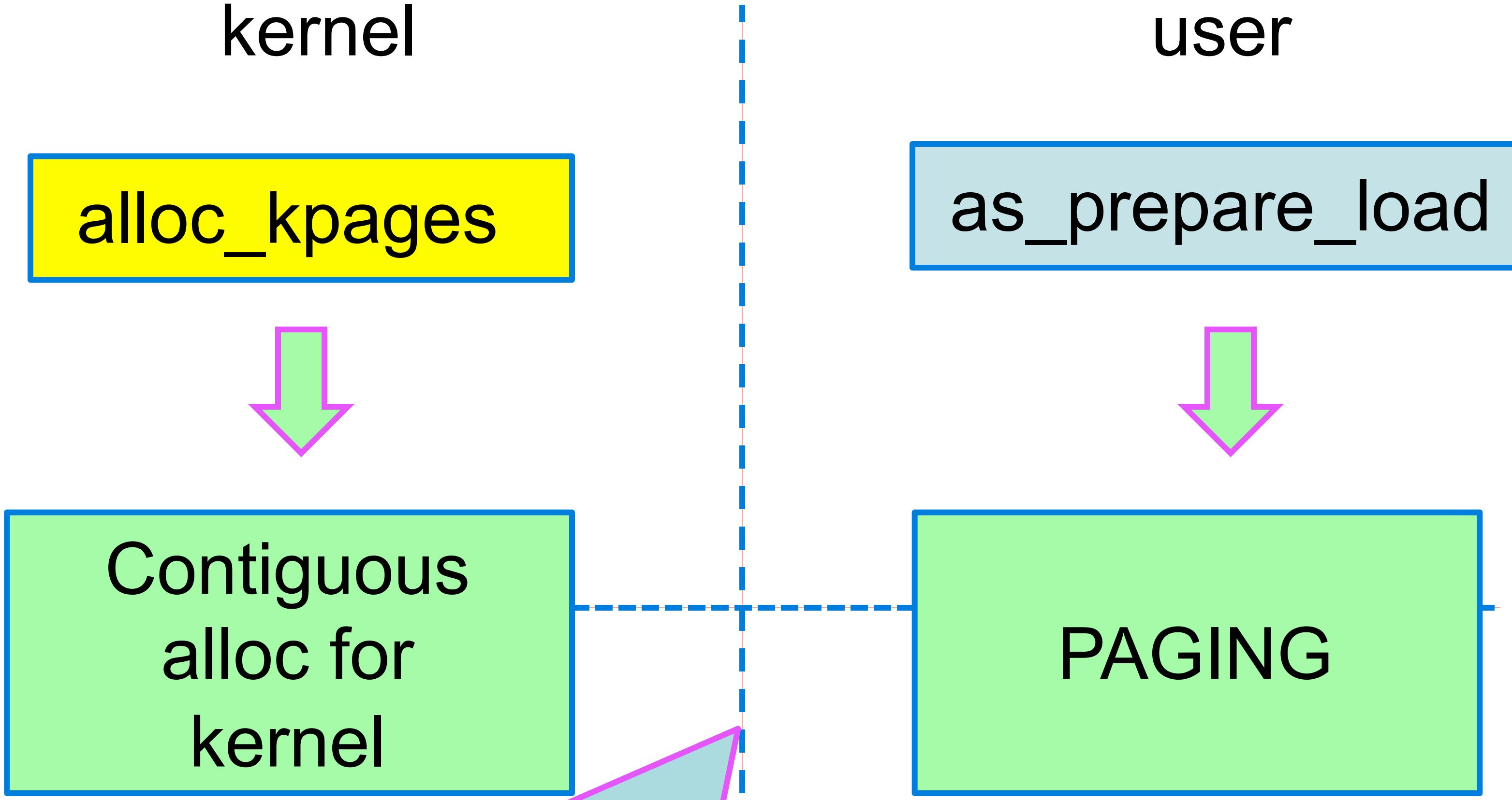
user

as_prepare_load

PAGING

Paging in user space

(solution B)



kernel

user

alloc_kpages

as_prepare_load

Contiguous alloc for kernel

PAGING

Two allocators:

- Paginn for user processes
- Contiguous memory for kernel

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
  - 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)

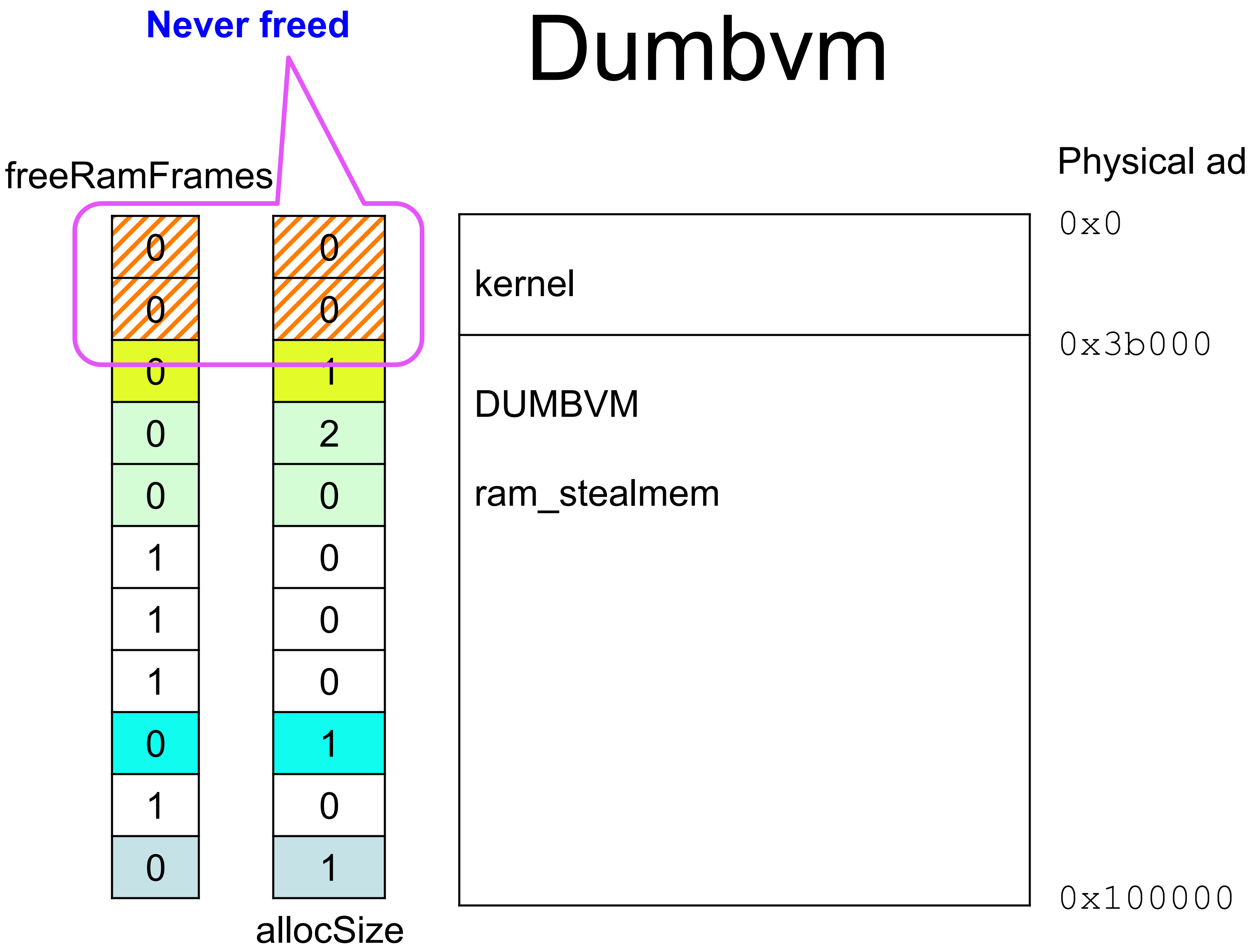
```
static struct spinlock freemem_lock = SPINLOCK_INITIALIZER;

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 activ
  spinlock_
  active =
  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

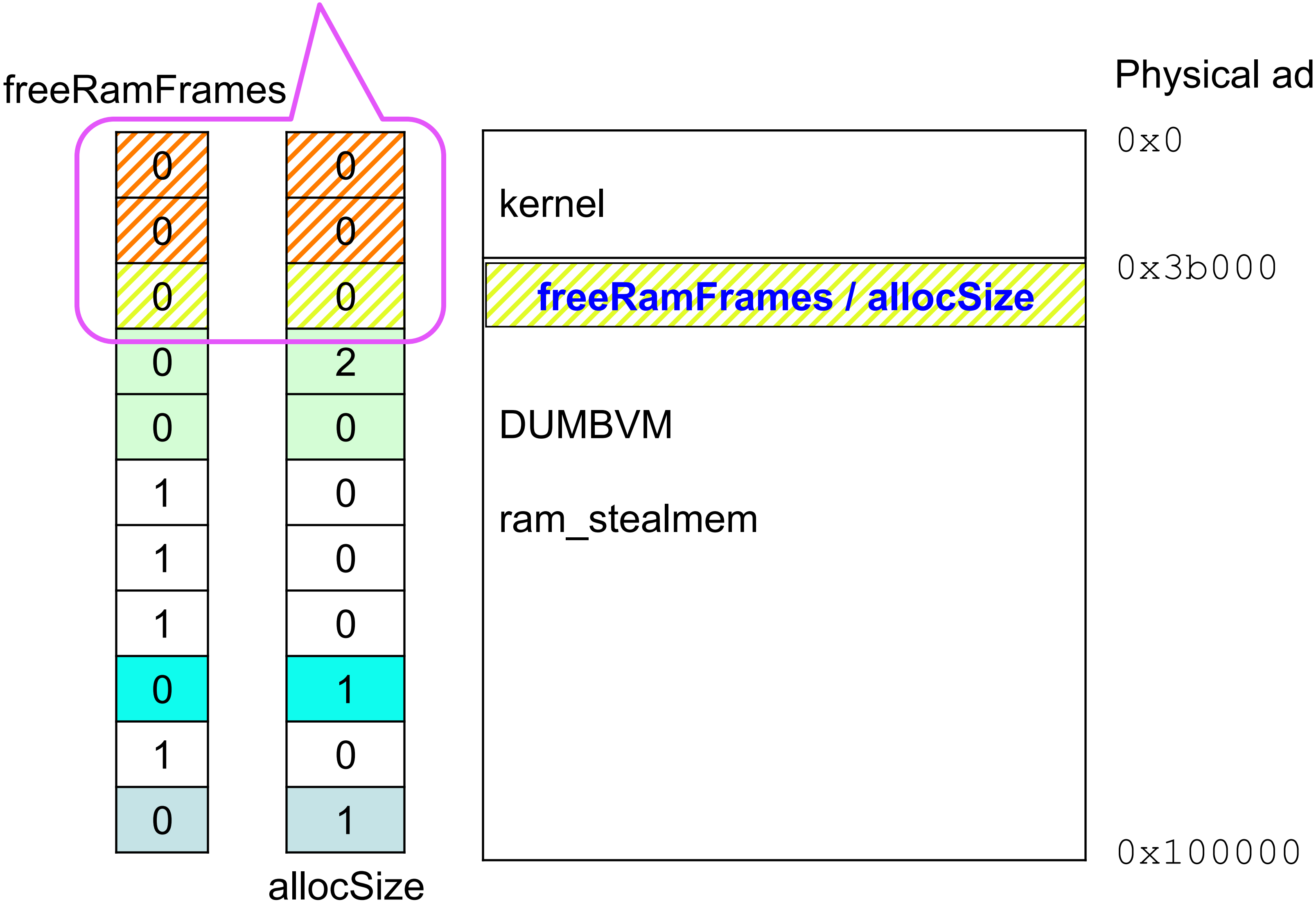


Never freed

freeRamFrames	allocSize
0	0
0	0
0	1
0	2
0	0
1	0
1	0
1	0
0	1
1	0
0	1

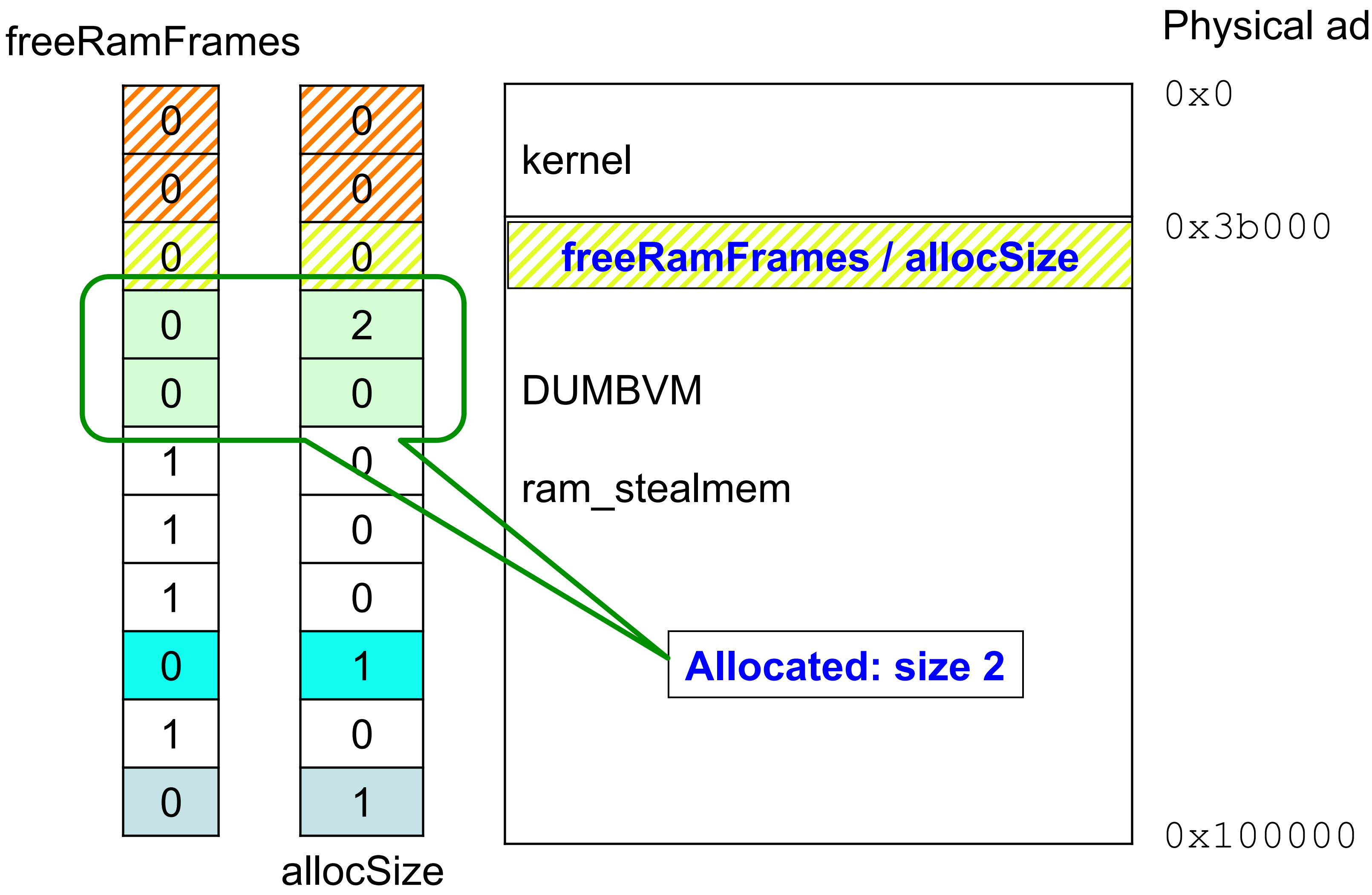
Dumbvm

Never freed:
including allocations
before table available



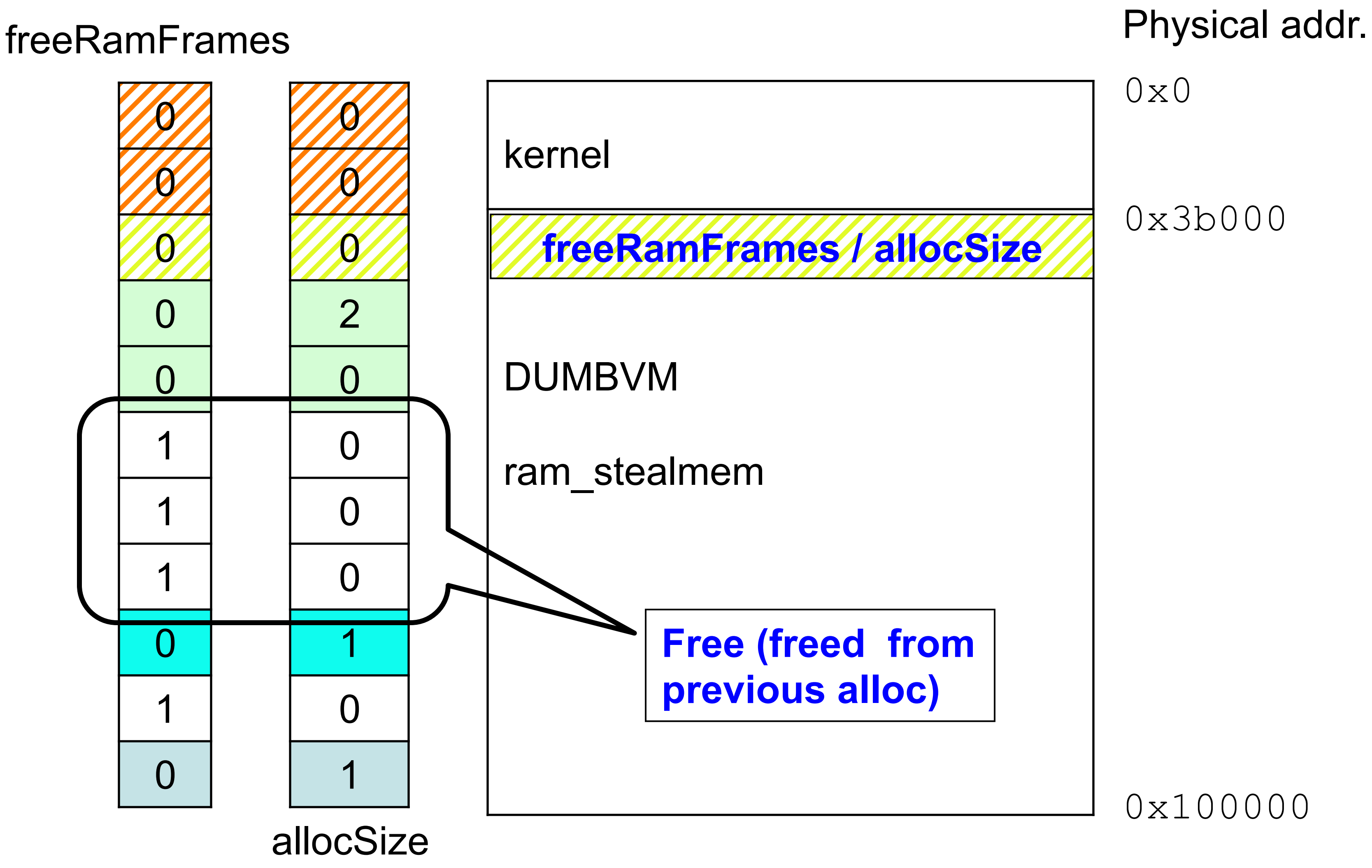
freeRamFrames	allocSize
0	0
0	0
0	0
0	2
0	0
1	0
1	0
1	0
0	1
1	0
0	1

Dumbvm



freeRamFrames	allocSize
0	0
0	0
0	0
0	2
0	0
1	0
1	0
1	0
0	1
1	0
0	1

Dumbvm



freeRamFrames	allocSize
0	0
0	0
0	0
0	2
0	0
1	0
1	0
1	0
0	1
1	0
0	1

Free (freed from previous alloc)

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;
}
```

freeppages

```
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;
}
```