

Assignment 3 Intro

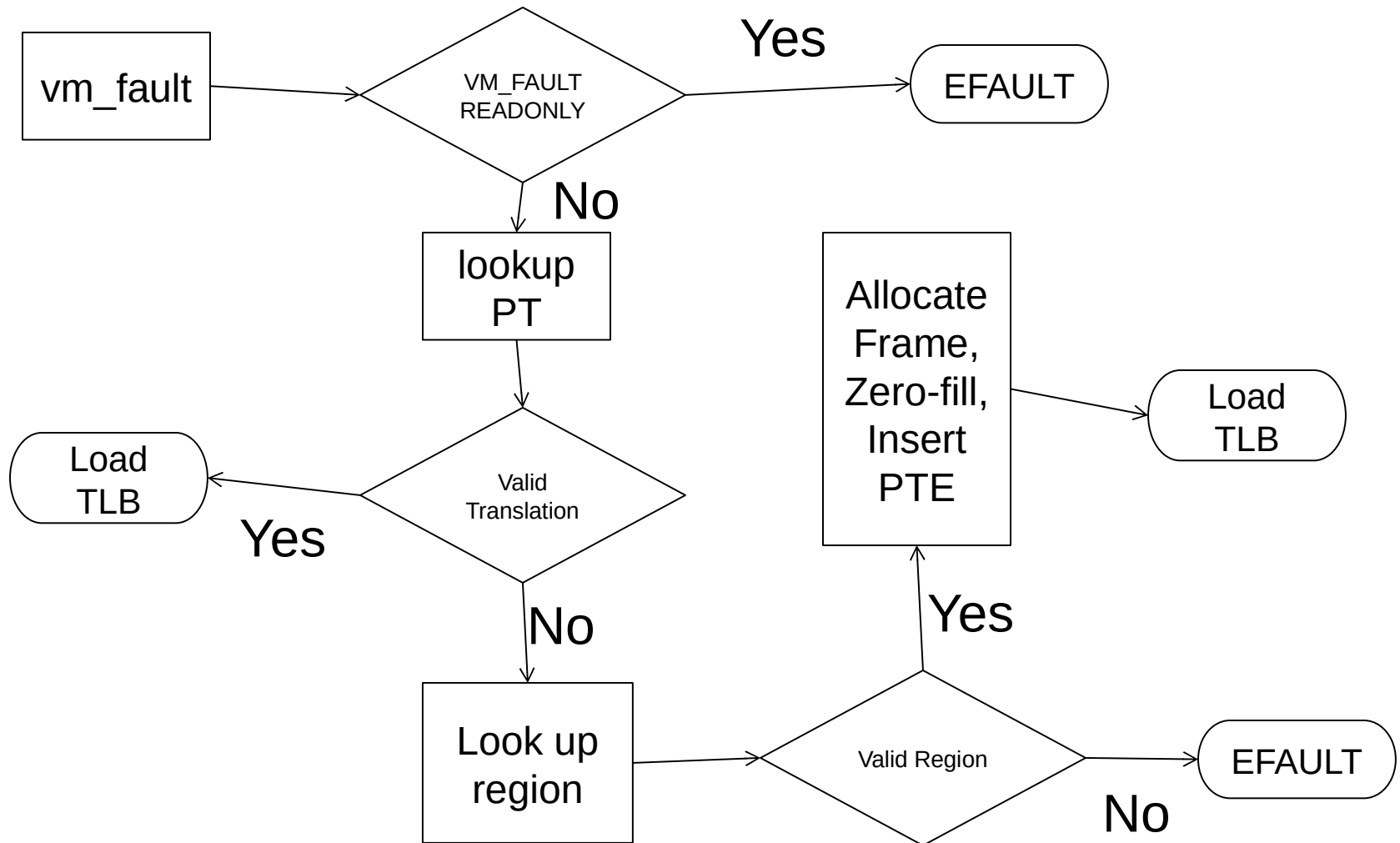


Assignment 3

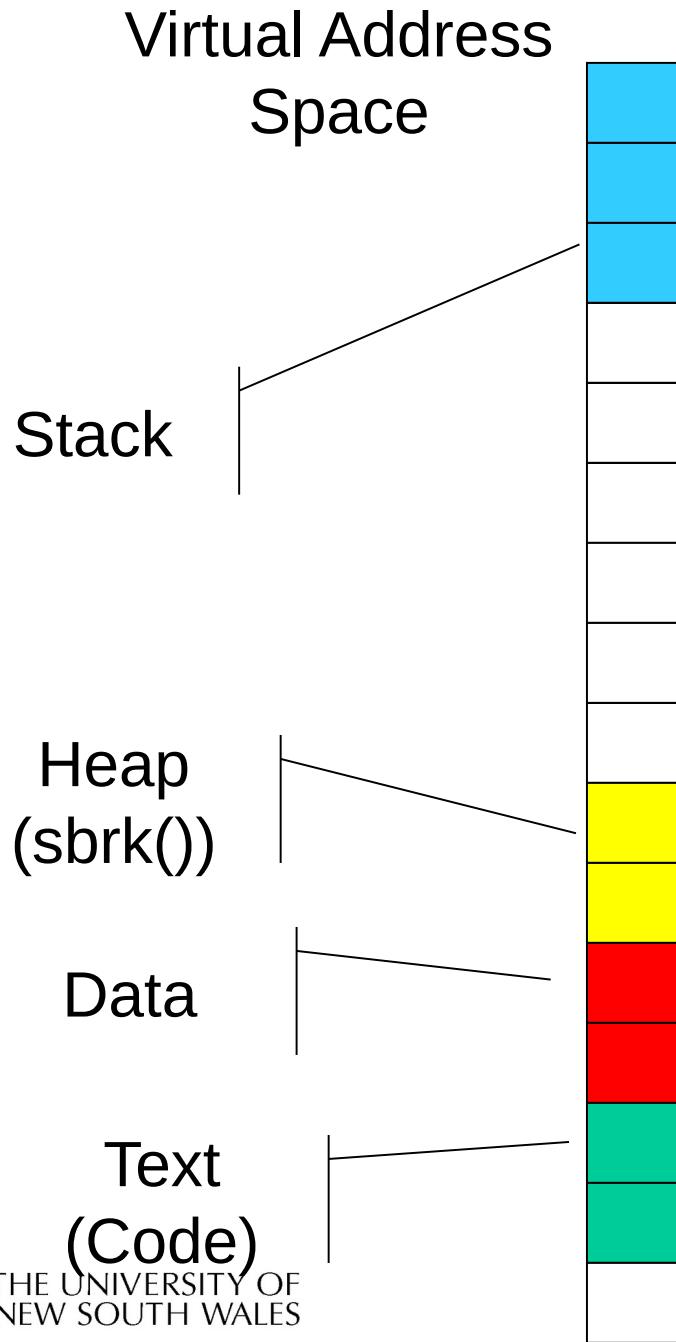
- Page table and ‘region’ support
 - Virtual memory for applications
 - Only the memory resident part, paging to disk is an advanced part.



Goal: Make “vm_fault” work



Theoretical Typical Address Space Layout



- Stack region is at top, and can grow down
- Heap has free space to grow up
- Text is typically read-only
- Implicit in this diagram
 - Multiple regions (ranges of virtual memory) to keep track of
 - Translation between each virtual page and physical frame currently accessible

Proc 1 Address Space

0
1
2

Page Table

U
T
S
D
C
A

Physical Address Space

K
X
N
A
C
S

Proc 2 Address Space

Z
Y
X
N
M
K

Page Table

5
4
7



Real R3000 Address Space Layout

- kuseg:
 - 2 gigabytes
 - TLB translated (mapping loaded from page table)
 - Cacheable (depending on 'N' bit)
 - user-mode and kernel mode accessible
 - Page size is 4K

0xFFFFFFFF

kseg2

0xC0000000

kseg1

0xA0000000

kseg0

0x80000000

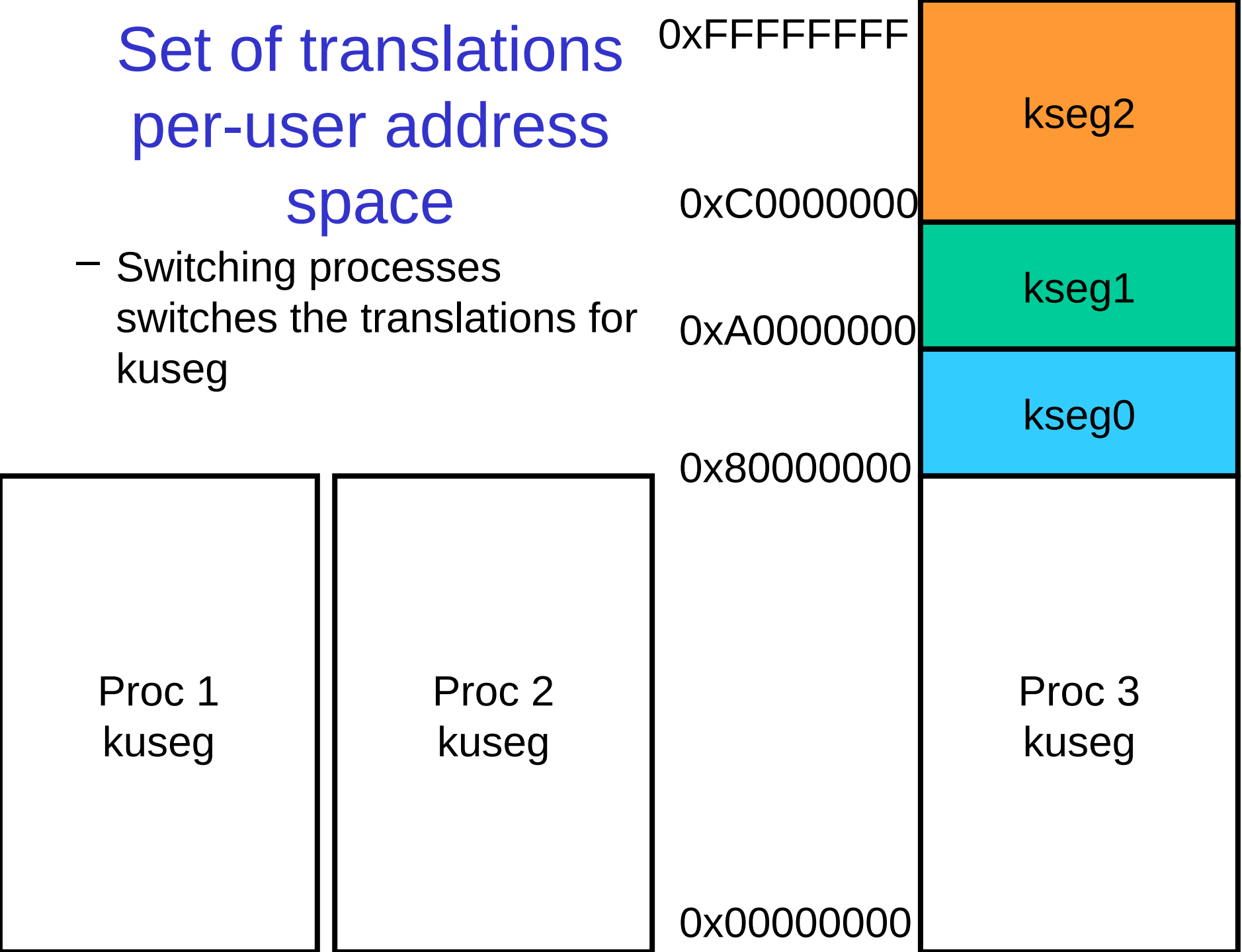
kuseg

0x00000000



Set of translations per-user address space

- Switching processes
switches the translations for
kuseg



The TLB

Each TLB entry contains

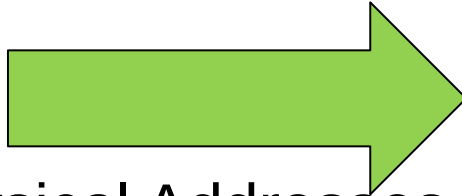
- EntryHi to match page# and ASID
- EntryLo which contains frame# and permissions

TLB (64 entries)

EntryHi	EntryLo
EntryHi	EntryLo
EntryHi	EntryLo
EntryHi	EntryLo
EntryHi	EntryLo
EntryHi	EntryLo
EntryHi	EntryLo
EntryHi	EntryLo



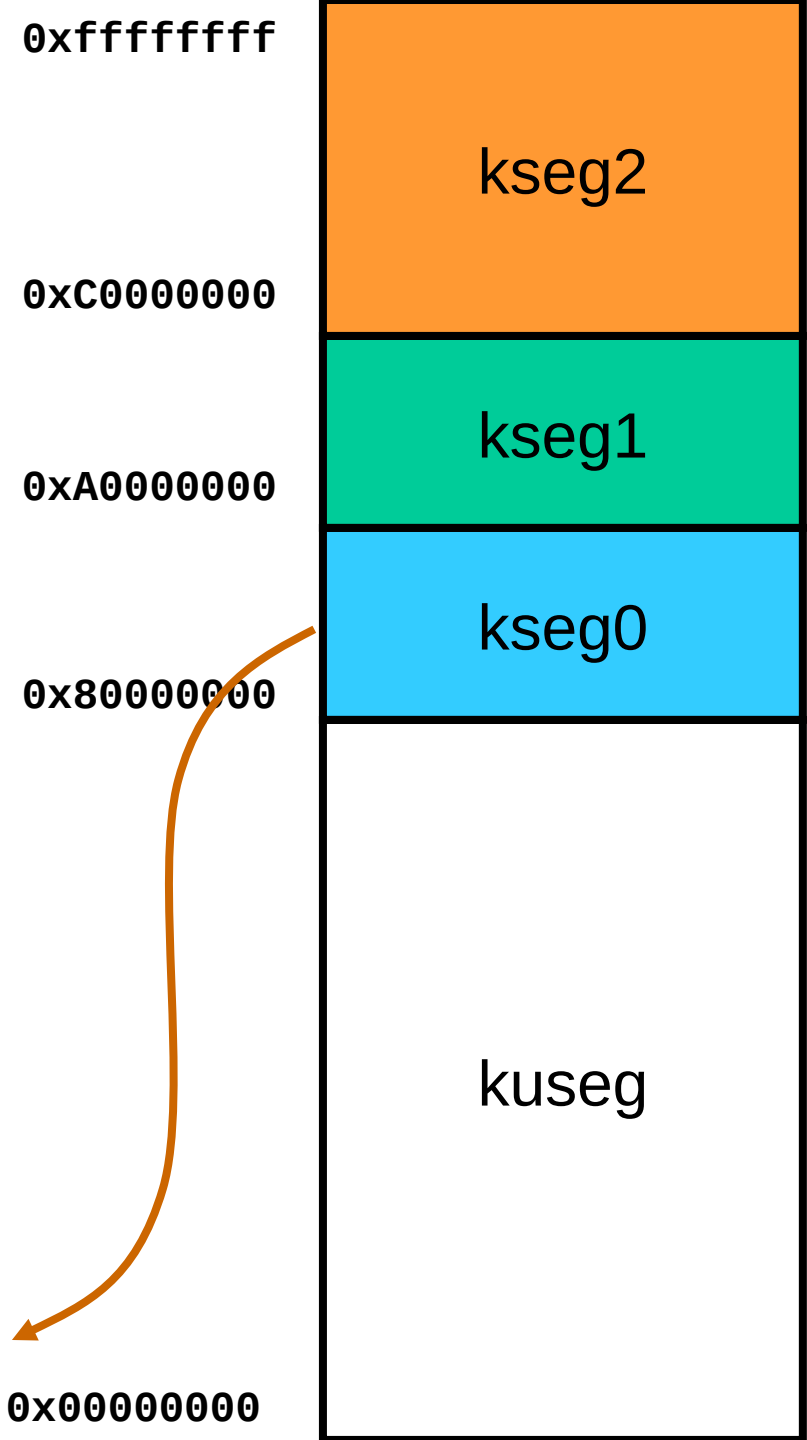
kuseg Virtual
Addresses



Physical Addresses

Kernel Address Space Layout

- kseg0:
 - 512 megabytes
 - Fixed translation window to physical memory
 - $0x80000000 - 0x9ffffff$ virtual = $0x00000000 - 0x1ffffff$ physical
 - TLB not used
 - Cacheable
 - Only kernel-mode accessible
 - Usually where the kernel code is placed



OS/161 Kernel

- Placed in Kseg0
 - lower part of physical memory
 - 16 meg of physical RAM
 - 31 busctl ramsize=16777216, in sys161.conf

0xffffffff

0xC0000000

0xA0000000

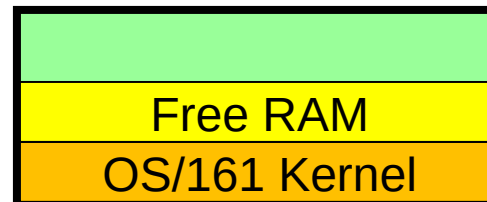
0x80000000

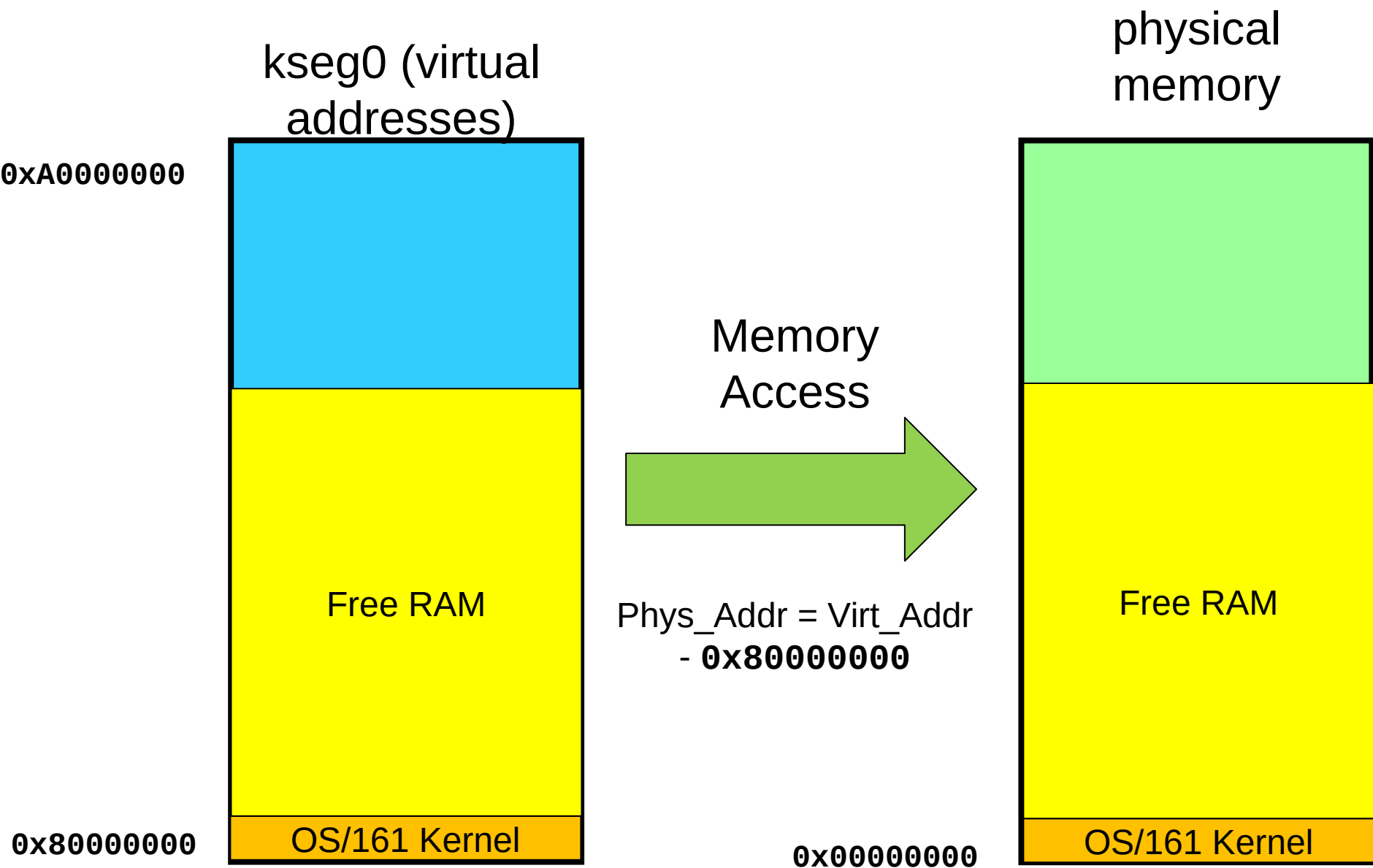
Free RAM

OS/161 Kernel

kuseg

0x00000000



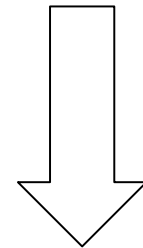


kmalloc()

```
/* Allocate/free some kernel-space virtual pages */
vaddr_t
alloc_kpages(unsigned npages)
{
    paddr_t paddr;
    if (npages > 1 ) {
        paddr = alloc_multiple_frames(npages);
    }
    else {
        paddr = alloc_one_frame(npages);
    }

    if (paddr == 0) {
        return 0;
    }
    return PADDR_TO_KVADDR(paddr);
}
```

void *
kmalloc()

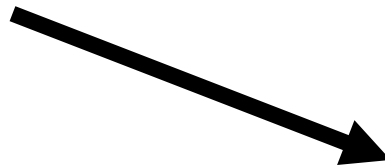


vaddr_t
alloc_kpage(
)

```
paddr = alloc_multiple_frames(npages);  
paddr = alloc_one_frame(npages);
```

See kern/arch/mips/vm/unsw.c

Frame Table
(bitmap)



0xA0000000

kseg0 (virtual
addresses)

Free RAM

0x80000000

OS/161 Kernel

physical
memory

Free
RAM

OS/161

0x00000000



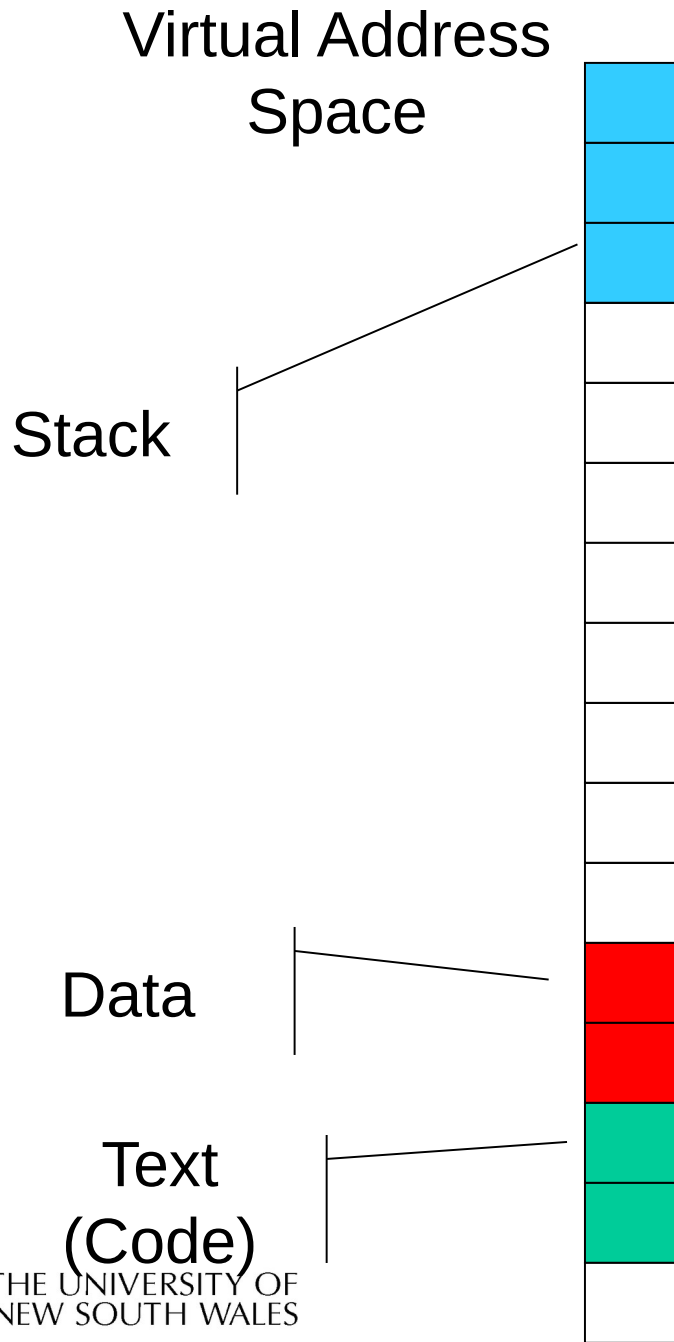
alloc_kpage()/free_kpage()

- The low-level functions that kmalloc()/kfree() use to allocate/free memory in its memory pool.
- Results are page aligned.
- Addresses are in the address range of kseg0
 - Need to convert to physical address to use as frame.
 - KVADDR_TO_PADDR(vaddr)

Summary

- Application virtual memory in kuseg
 - Translated by TLB
 - TLB content determined by
 - vm_fault()
 - Page Table
 - Valid Regions
- Kernel memory in kseg0
 - Translated by fixed offset
 - Allocators already provided





KUseg layout

- Stack region is at top, and can grow down
- Other regions determined by ELF file
 - see `load_elf()`
 - number can vary
 - permissions specified also
 - `os161-objdump -p testbin/huge`


```
thresher% os161-objdump -h ../bin/true
```

```
../bin/true:      file format elf32-tradbigmips
```

Sections:

Idx	Name	Size	VMA	LMA	File off	Algn
0	.reginfo	00000018	00400094	00400094	00000094	2**2
	CONTENTS, ALLOC, LOAD, READONLY, DATA, LINK_ONCE_SAME_SIZE					
1	.text	000001d0	004000b0	004000b0	000000b0	2**4
	CONTENTS, ALLOC, LOAD, READONLY, CODE					
2	.data	00000000	10000000	10000000	00001000	2**4
	CONTENTS, ALLOC, LOAD, DATA					
3	.sbss	00000008	10000000	10000000	00001000	2**2
	ALLOC					
4	.bss	00000000	10000010	10000010	00001008	2**4
	ALLOC					
5	.comment	00000036	00000000	00000000	00001008	2**0
	CONTENTS, READONLY					
6	.pdr	000004a0	00000000	00000000	00001040	2**2
	CONTENTS, READONLY					
7	.mdebug.abi32	00000000	00000000	00000000	000014e0	2**0
	CONTENTS, READONLY					



```
thresher% os161-objdump -p ../bin/true
```

```
../bin/true:      file format elf32-tradbigmips
```

Program Header:

```
0x70000000 off      0x00000094 vaddr 0x00400094 paddr 0x00400094 align 2**2
      filesz 0x00000018 memsz 0x00000018 flags r--
      LOAD off      0x00000000 vaddr 0x00400000 paddr 0x00400000 align 2**12
      filesz 0x00000280 memsz 0x00000280 flags r-x
      LOAD off      0x00001000 vaddr 0x10000000 paddr 0x10000000 align 2**12
      filesz 0x00000000 memsz 0x00000010 flags rw-
private flags = 1001: [abi=032] [mips1] [not 32bitmode]
```

Zero fill fresh
pages prior to
mapping



Walk through load elf

- Load the ELF header from executable file
- Check it's an ELF file
- For each “Program Header”
 - call `as_define_region()`
- For each “Program Header”
 - load the segment from the file if required

Process Layout

- Process layout in KUseg
 - regions specified by calls to
 - as_define_stack()
 - as_define_region()
 - usually implemented as a linked list of region specifications
 - as_prepare_load()
 - make READONLY regions READWRITE for loading purposes
 - as_complete_load()
 - enforce READONLY again

0xffffffff

0xc0000000

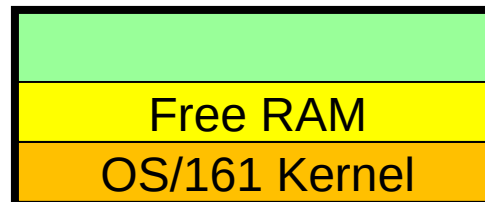
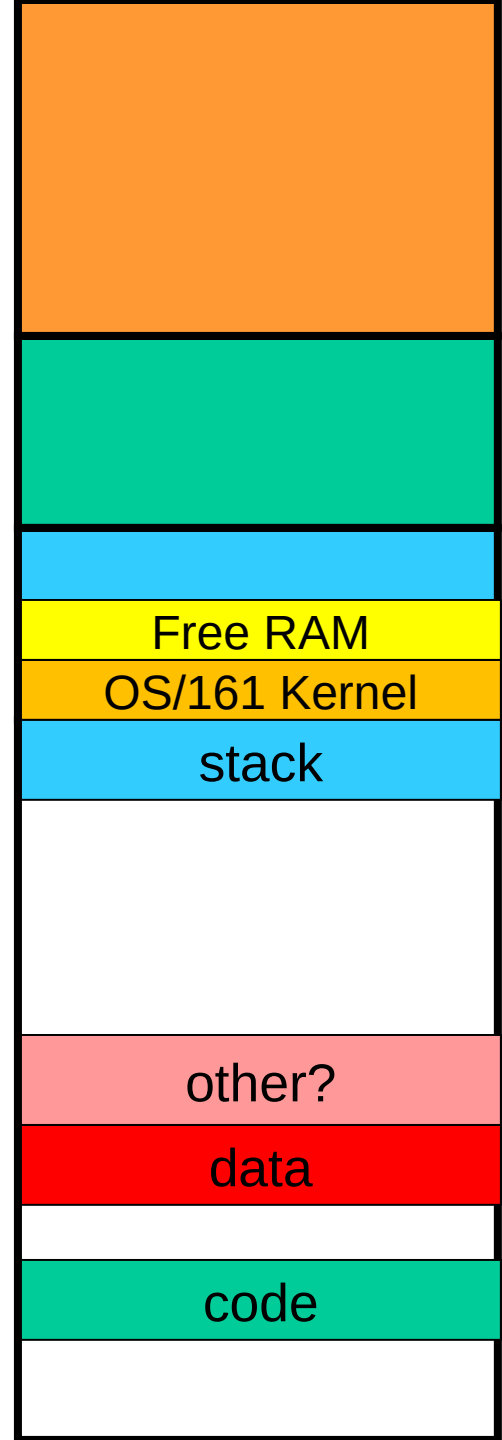
0xa0000000

0x80000000

0x10000000

0x04000000

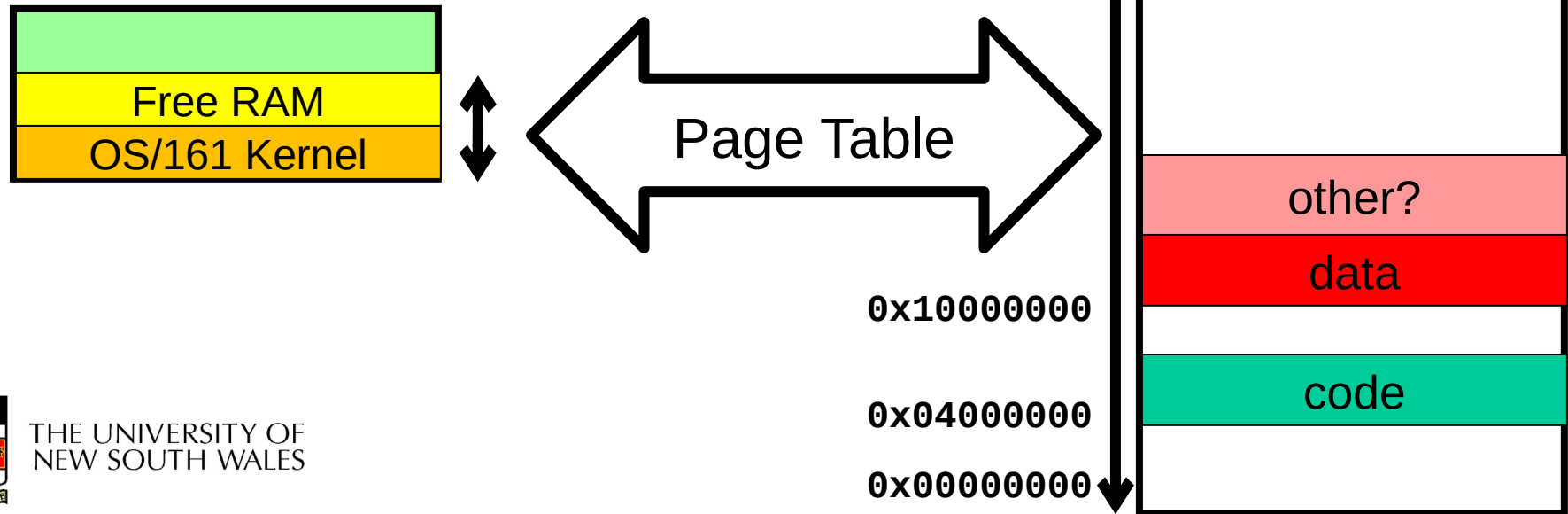
0x00000000



- `as_create()`
 - allocate a data structure used to keep track of an address space
 - i.e. regions
 - `proc_getas()` used to get access to current address space struct
 - `struct addrspace *as;`
- `as_destroy()`
 - deallocate book keeping and page tables.
 - deallocate frames used

Process Layout

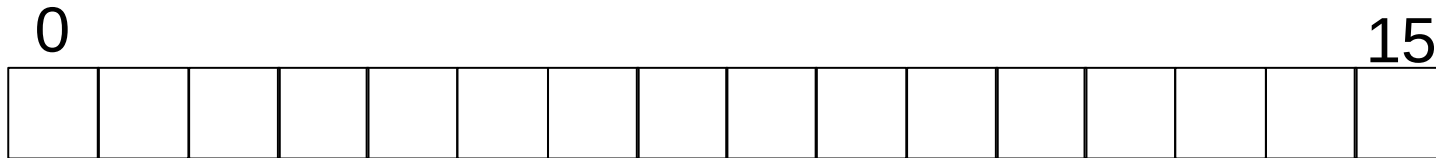
- Need to keep translation table for KUSEG



Pointer Recap

Memory

- 4-bit addresses, i.e. address range 0 – 15



```
char *c;  
int *i;
```

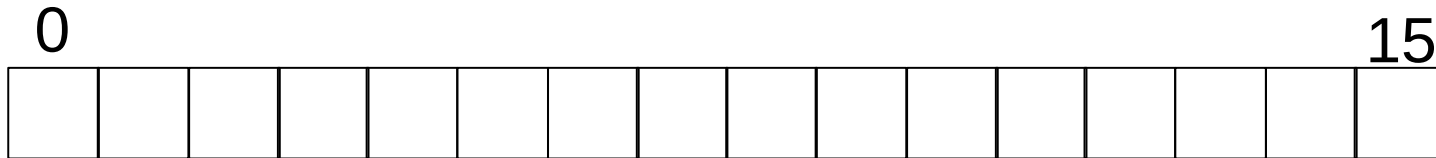
Examples

```
c = 5; *c = 'x'  
i = 4; *i = 42
```

Indexing off Pointers

Memory

- 4-bit addresses, i.e. address range 0 – 15



```
char *c;  
int *i;
```

Examples

```
c = 5; c[0] = 'h'; c[1] = 'i';  
i = 4; i[0] = 42; i[2] = 7;
```

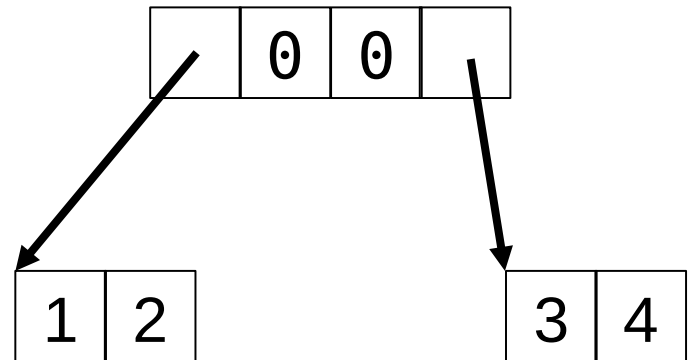

2-level table in 'C'

```
unsigned int **table;  
table=malloc(4* sizeof(unsigned int *));
```

```
table[0] = malloc(2 * sizeof(unsigned  
int));  
table[1] = NULL;  
table[2] = NULL;  
table[3] = malloc(2 * sizeof(unsigned  
int));
```

```
table[0][0] = 1;  
table[0][1] = 2;  
table[3][0] = 3;  
table[3][1] = 4;
```

```
table[1][0] = 42; /* fails dereferencing  
NULL */
```



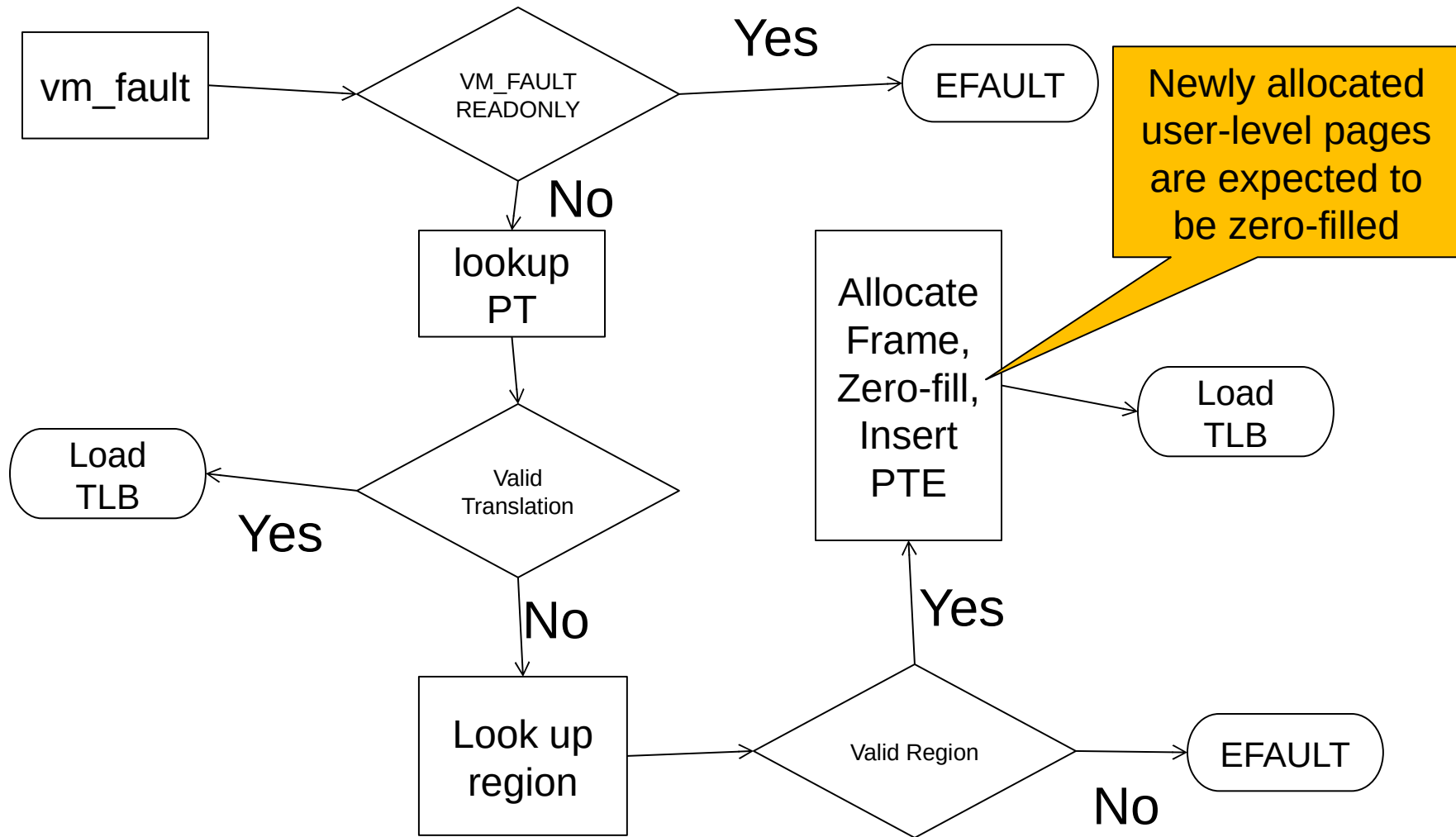
2-level page table in 'C'

```
paddr_t **pagetable;
```



- `as_copy()`
 - allocates a new (destination) address space
 - adds all the same regions as source
 - roughly, for each mapped page in source
 - allocate a frame in dest
 - copy contents from source frame to dest frame
 - add PT entry for dest
- `as_activate()`
 - flush TLB
 - (or set the hardware asid)
- `as_deactivate()`
 - flush TLB
 - (or flush an asid)

VM Fault Approximate Flow Chart



kprintf()

- Do not use it in `vm_fault()`
 - `kprintf()` blocks current process while printing
 - Switches to another process
 - Context switch flushes TLB
 - Flushes what you just inserted
 - Endless loop

trace161 can help with debugging

<http://cgi.cse.unsw.edu.au/~cs3231/06s1/os161/man/sys161/index.html>

- The following additional options control trace161's tracing and are ignored by sys161:
- *-f tracefile*
 - Set the file trace information is logged to. By default, stderr is used. Specifying *-f* sends output to stdout instead of stderr.
- *-t traceflags*
 - Tell System/161 what to trace. The following flags are available:
 - d Trace disk I/O
 - e Trace emufs I/O
 - j Trace jumps and branches
 - k Trace instructions in kernel mode
 - n Trace network I/O
 - t Trace TLB/MMU activity
 - u Trace instructions in user mode
 - x Trace exceptions
- Caution: tracing instructions generates huge amounts of output that may overwhelm smaller host systems.



wagner% trace161 -tt kernel

sys161: System/161 release 2.0.8, compiled Feb 19 2017 14:31:56

sys161: Tracing enabled: tlb

trace: 00 tlb: 81000/000 -> 00000 ----: [0]
trace: 00 tlb: 81001/000 -> 00000 ----: [1]
trace: 00 tlb: 81002/000 -> 00000 ----: [2]
trace: 00 tlb: 81003/000 -> 00000 ----: [3]
trace: 00 tlb: 81004/000 -> 00000 ----: [4]
trace: 00 tlb: 81005/000 -> 00000 ----: [5]
trace: 00 tlb: 81006/000 -> 00000 ----: [6]
trace: 00 tlb: 81007/000 -> 00000 ----: [7]
trace: 00 tlb: 81008/000 -> 00000 ----: [8]
trace: 00 tlb: 81009/000 -> 00000 ----: [9]
trace: 00 tlb: 8100a/000 -> 00000 ----: [10]
trace: 00 tlb: 8100b/000 -> 00000 ----: [11]
trace: 00 tlb: 8100c/000 -> 00000 ----: [12]
trace: 00 tlb: 8100d/000 -> 00000 ----: [13]
trace: 00 tlb: 8100e/000 -> 00000 ----: [14]
trace: 00 tlb: 8100f/000 -> 00000 ----: [15]
trace: 00 tlb: 81010/000 -> 00000 ----: [16]
trace: 00 tlb: 81011/000 -> 00000 ----: [17]
trace: 00 tlb: 81012/000 -> 00000 ----: [18]
trace: 00 tlb: 81013/000 -> 00000 ----: [19]
trace: 00 tlb: 81014/000 -> 00000 ----: [20]



.....

trace: 00 tlbp: 8103f/000 -> 00000 ----: [63]
trace: 00 tlbp: 81040/000 -> NOT FOUND
trace: 00 tlbiwi: [0] 81000/000 -> 00000 ---- ==> 81040/000 -> 00000 ----
trace: 00 tlbp: 81041/000 -> NOT FOUND
trace: 00 tlbiwi: [1] 81001/000 -> 00000 ---- ==> 81041/000 -> 00000 ----
trace: 00 tlbp: 81042/000 -> NOT FOUND
trace: 00 tlbiwi: [2] 81002/000 -> 00000 ---- ==> 81042/000 -> 00000 ----
trace: 00 tlbp: 81043/000 -> NOT FOUND
trace: 00 tlbiwi: [3] 81003/000 -> 00000 ---- ==> 81043/000 -> 00000 ----
trace: 00 tlbp: 81044/000 -> NOT FOUND
trace: 00 tlbiwi: [4] 81004/000 -> 00000 ---- ==> 81044/000 -> 00000 ----
trace: 00 tlbp: 81045/000 -> NOT FOUND
trace: 00 tlbiwi: [5] 81005/000 -> 00000 ---- ==> 81045/000 -> 00000 ----
trace: 00 tlbp: 81046/000 -> NOT FOUND
trace: 00 tlbiwi: [6] 81006/000 -> 00000 ---- ==> 81046/000 -> 00000 ----
trace: 00 tlbp: 81047/000 -> NOT FOUND
trace: 00 tlbiwi: [7] 81007/000 -> 00000 ---- ==> 81047/000 -> 00000 ----
trace: 00 tlbp: 81048/000 -> NOT FOUND
trace: 00 tlbiwi: [8] 81008/000 -> 00000 ---- ==> 81048/000 -> 00000 ----



.....

```
trace: 00 tlbwi: [60] 8103c/000 -> 00000 ---- ==> 8107c/000 -> 00000 ----  
trace: 00 tlbp:    8107d/000 -> NOT FOUND  
trace: 00 tlbwi: [61] 8103d/000 -> 00000 ---- ==> 8107d/000 -> 00000 ----  
trace: 00 tlbp:    8107e/000 -> NOT FOUND  
trace: 00 tlbwi: [62] 8103e/000 -> 00000 ---- ==> 8107e/000 -> 00000 ----  
trace: 00 tlbp:    8107f/000 -> NOT FOUND  
trace: 00 tlbwi: [63] 8103f/000 -> 00000 ---- ==> 8107f/000 -> 00000 ----
```

OS/161 base system version 2.0.3

(with locks/CVs, system calls solutions)

Copyright (c) 2000, 2001-2005, 2008-2011, 2013, 2014

President and Fellows of Harvard College. All rights reserved.

Put-your-group-name-here's system version 0 (ASST3 #29)

16208k physical memory available

Device probe...

lamebus0 (system main bus)

emu0 at lamebus0



End of trace from bin/true

trace: 00 tlblookup: 00400/000 -> no match
trace: 00 tlbwr: [58] 8003a/000 -> 00000 ---- ==> 00400/000 -> 00034 -V--
trace: 00 tlblookup: 00400/000 -> 00034 -V--: [58] - OK
trace: 00 tlblookup: 00400/000 -> 00034 -V--: [58] - OK
trace: 00 tlblookup: 00410/000 -> no match
trace: 00 tlbwr: [34] 80022/000 -> 00000 ---- ==> 00410/000 -> 00036 -VD-
trace: 00 tlblookup: 00400/000 -> 00034 -V--: [58] - OK
trace: 00 tlblookup: 00400/000 -> 00034 -V--: [58] - OK
trace: 00 tlblookup: 00410/000 -> 00036 -VD-: [34] - OK
trace: 00 tlblookup: 00410/000 -> 00036 -VD-: [34] - OK
trace: 00 tlblookup: 00400/000 -> 00034 -V--: [58] - OK
trace: 00 tlblookup: 7ffff/000 -> 00035 -VD-: [25] - OK
trace: 00 tlblookup: 00400/000 -> 00034 -V--: [58] - OK
trace: 00 tlblookup: 00400/000 -> 00034 -V--: [58] - OK
trace: 00 tlblookup: 00400/000 -> 00034 -V--: [58] - OK



TLB refill

- Use `tlb_random()`
- Cost of book keeping to do something smarter costs more than potential benefit

TLB_random()

Disable interrupts when writing to the TLB in vm_fault!

```
spl = splhigh();  
tlb_random(entry_hi, entry_lo);  
splx(spl);
```

```
tlb_random:  
    mtc0 a0, c0_entryhi /* store the passed  
                        entry into the */  
    mtc0 a1, c0_entrylo /* tlb entry registers  
*/  
    ssnop                /* wait for pipeline  
                        hazard */  
    ssnop  
    tlbwr                /* do it */  
    j ra  
    nop  
    .end tlb_random
```



Summary

- Assignment 3: Implement VM
- Track memory “regions”
- Manage page table
- Manage filling TLB from page table
 - “vm_fault”

