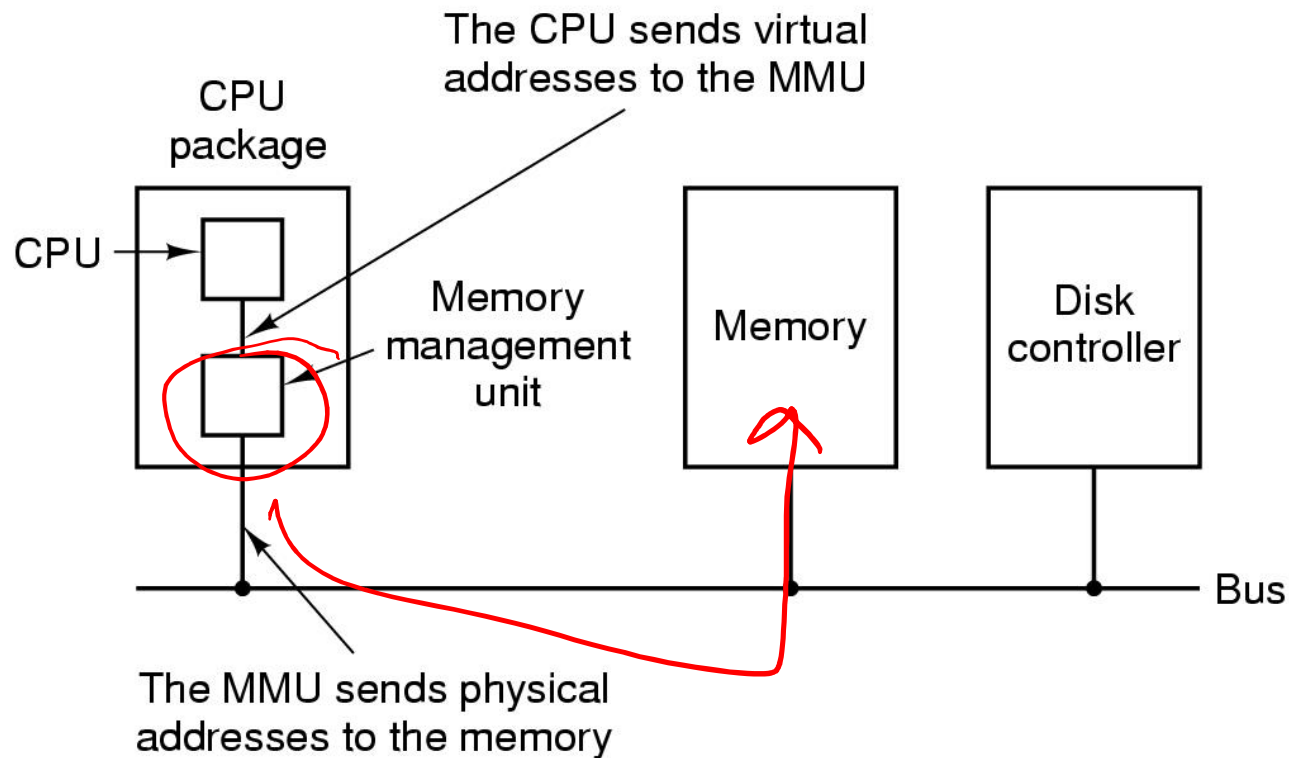# Virtual Memory

THE UNIVERSITY OF
NEW SOUTH WALES

Learning Outcomes

- An understanding of page-based virtual memory in depth.
  - Including the R3000's support for virtual memory.

Memory Management Unit (or TLB)
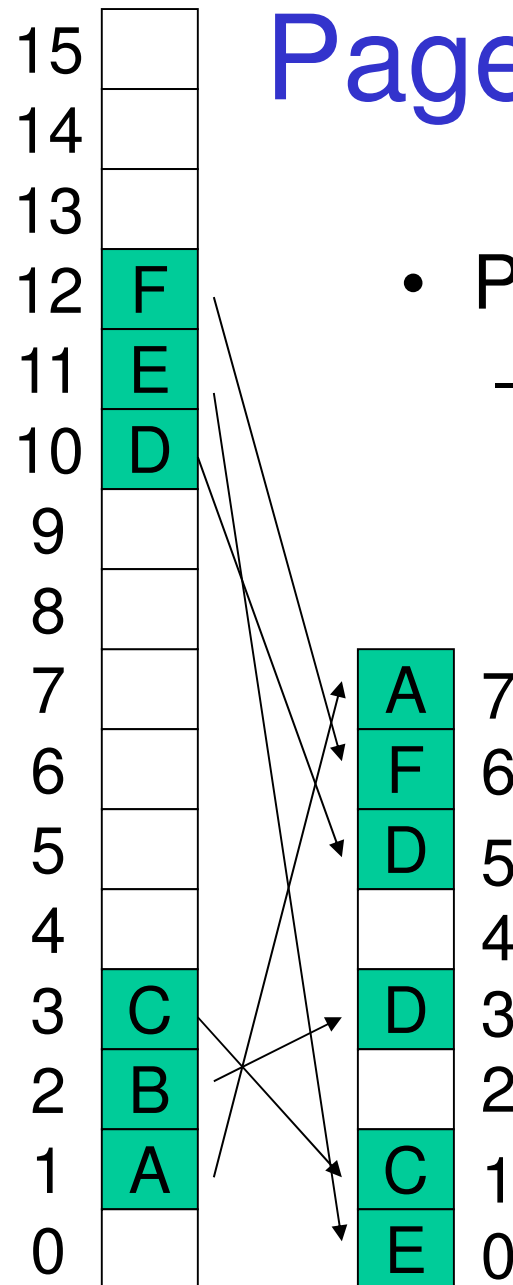

The position and function of the MMU

Page-based VM

Virtual Address Space

- **Virtual Memory**
  - Divided into equal-sized *pages*
  - A *mapping* is a translation between
    - A page and a frame
    - A page and null
  - Mappings defined at runtime
    - They can change
  - Address space can have holes
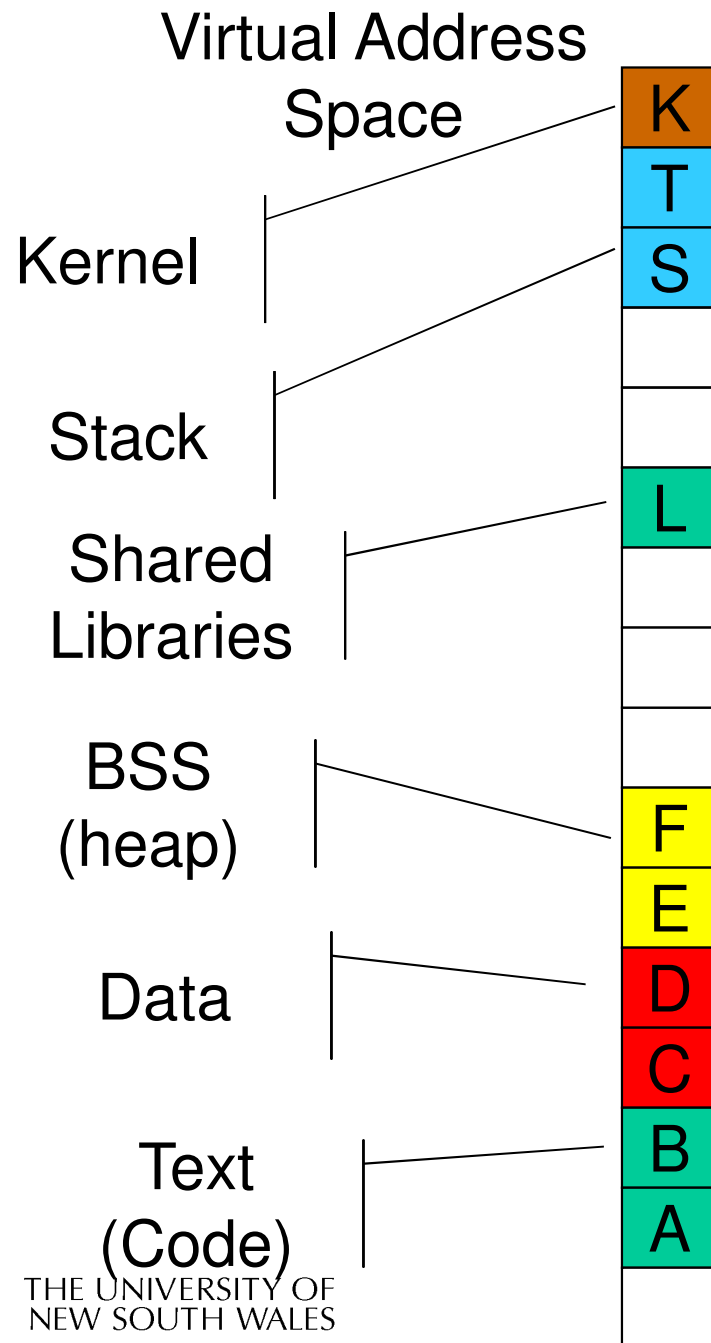 - Process does not have to be contiguous in physical memory

- **Physical Memory**
  - Divided into equal-sized *frames*

Physical Address Space

4

Typical Address Space Layout

Virtual Address Space

Kernel

Stack

Shared Libraries

BSS (heap)

Data

Text (Code)

K
T
S
L
F
E
D
C
B
A

- Stack region is at top, and can grow down
- Heap has free space to grow up
- Text is typically read-only
- Kernel is in a reserved, protected, shared region
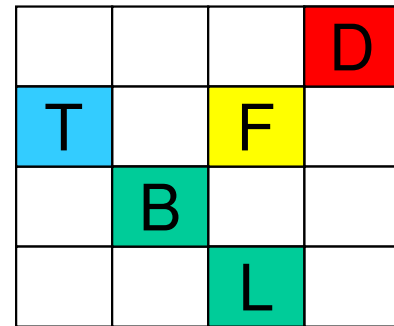- 0-th page typically not used, why?

THE UNIVERSITY OF NEW SOUTH WALES

5

Virtual Address Space

Programmer's perspective: logically present
System's perspective: Not mapped, data on disk

- A process may be only partially resident
  - Allows OS to store individual pages on disk
  - Saves memory for infrequently used data & code
- What happens if we access non-resident memory?

Disk

Physical Address Space

Proc 1 Address Space

Proc 2 Address Space

Currently running

Physical Address Space

Memory Access

Disk

THE UNIVERSITY OF NEW SOUTH WALES
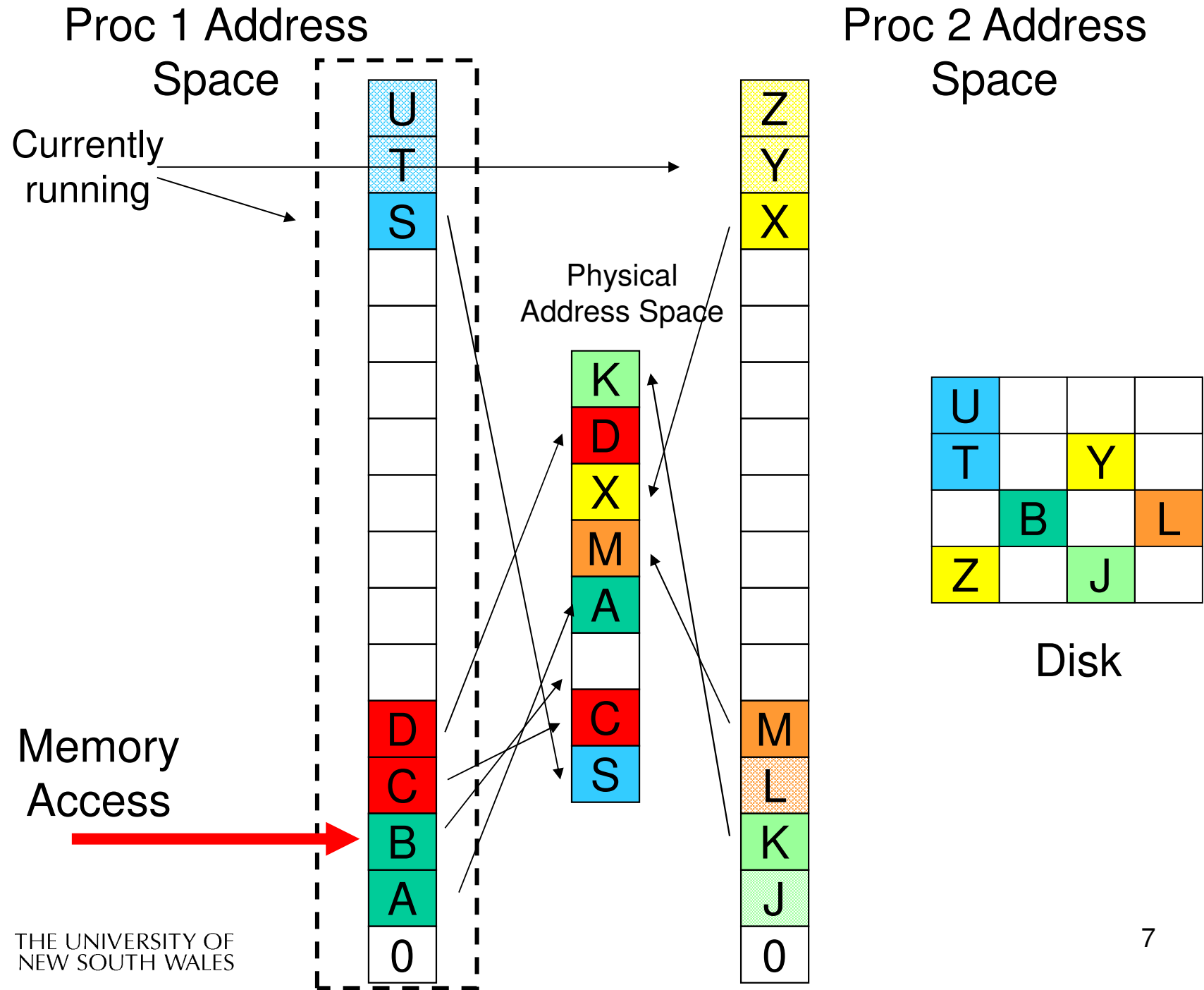
7

Page Faults

- Referencing an invalid page triggers a page fault
    - An exception handled by the OS

- Broadly, two standard page fault types
    - Illegal Address (protection error)
        - Signal or kill the process
    - Page not resident
        - Get an empty frame
        - Load page from disk
        - Update page (translation) table (enter frame #, set valid bit, etc.)
        - Restart the faulting instruction

THE UNIVERSITY OF
NEW SOUTH WALES

Virtual Address Space

- Page table for resident part of address space

Page Table

Physical Address Space

Shared Pages

- Private code and data
  - Each process has own copy of code and data
  - Code and data can appear anywhere in the address space

- Shared code
  - Single copy of code shared between all processes executing it
  - Code must not be self modifying
  - Code must appear at same address in all processes

Proc 1 Address Space

Proc 2 Address Space

0	

Two (or more) processes running the same program and sharing the text section

1	
7	
2	

Page Table

U
T
S

D
C
B
A

Physical Address Space

B

X
N

A
C
S

Z
Y
X

N
M
B
A

5	

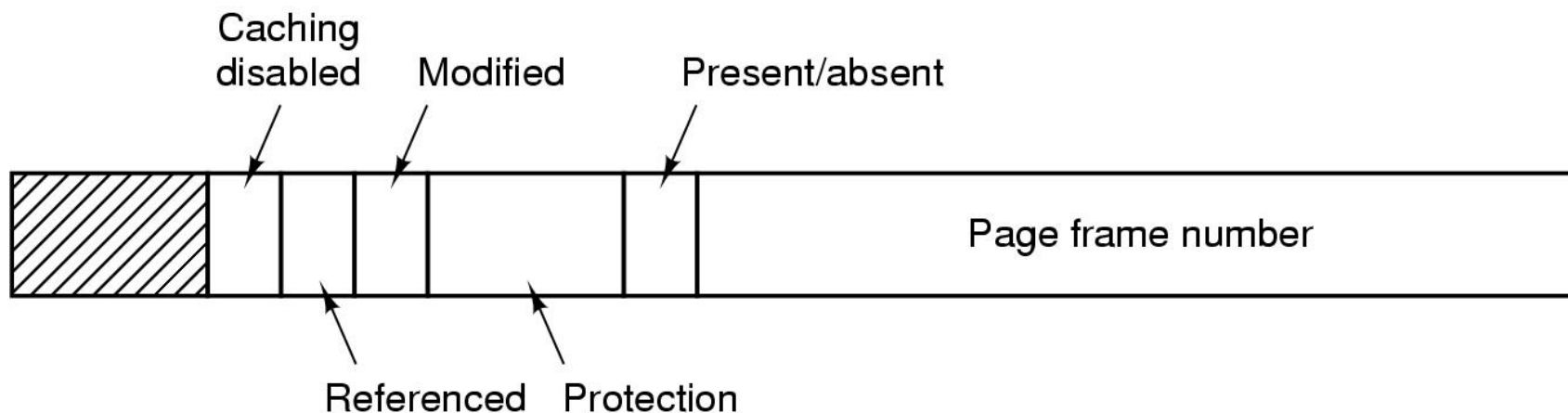|	
4	
7	
2	

Page Table

11

Page Table Structure

- Page table is (logically) an array of frame numbers
  - Index by page number

- Each page-table entry (PTE) also has other bits

Caching disabled   Modified   Present/absent

Page frame number

Referenced   Protection

5
4
7
2

Page Table

THE UNIVERSITY OF
NEW SOUTH WALES

PTE Attributes (bits)

- Present/Absent bit
  - Also called *valid bit,* it indicates a valid mapping for the page
- Modified bit
  - Also called *dirty bit,* it indicates the page may have been modified in memory
- Reference bit
  - Indicates the page has been accessed
- Protection bits
  - Read permission, Write permission, Execute permission
  - Or combinations of the above
- Caching bit
  - Use to indicate processor should bypass the cache when accessing memory
    - Example: to access device registers or memory

Address Translation

- Every (virtual) memory address issued by the CPU must be translated to physical memory
  - Every *load* and every *store* instruction
  - Every instruction fetch
- Need Translation Hardware
- In paging system, translation involves replace page number with a frame number

Virtual Memory Summary

virtual and physical mem chopped up in pages/frames

- **programs use virtual addresses**

- **virtual to physical mapping by MMU**

  -first check if page present (present/absent bit)

  -if yes: address in page table form MSBs in physical address

  -if no: bring in the page from disk ➔ page fault

THE UNIVERSITY OF
NEW SOUTH WALES
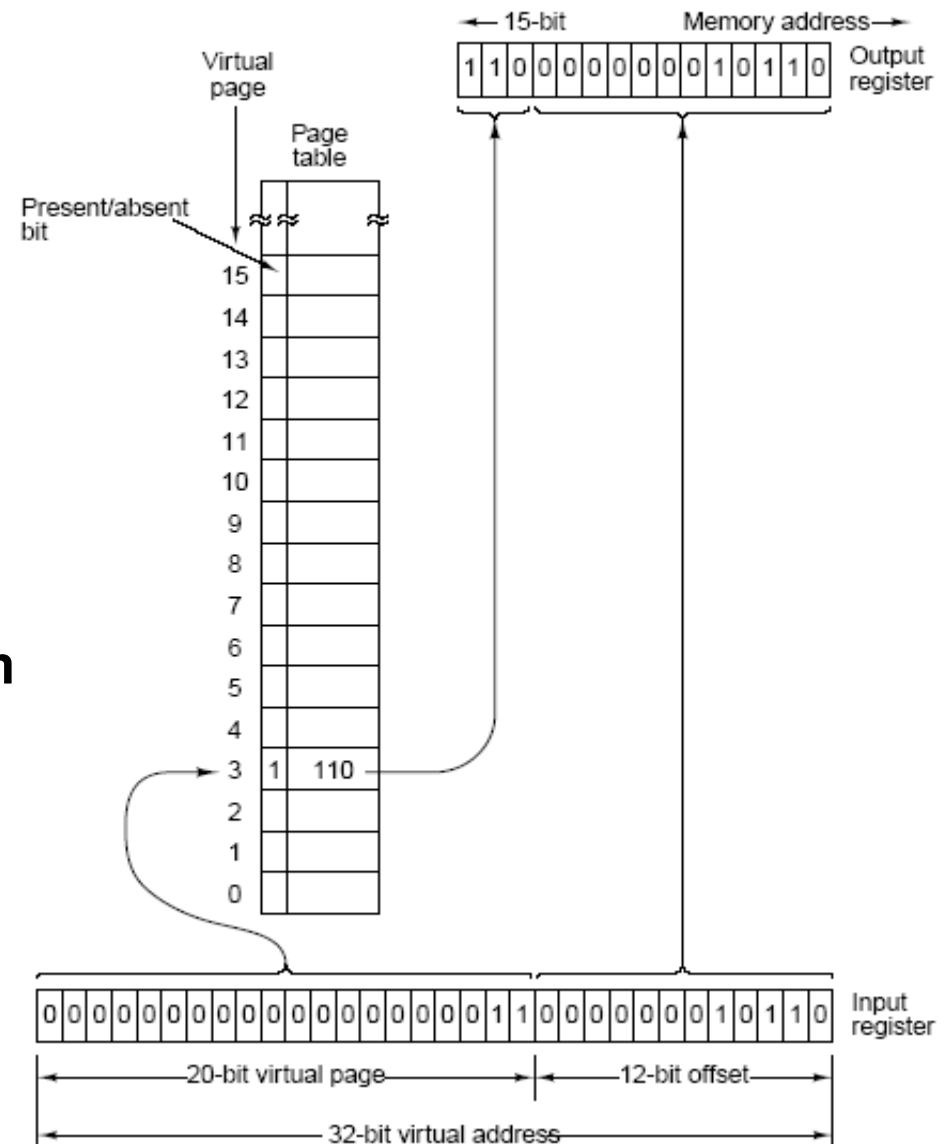
Page Tables

- ## Assume we have

    - 32-bit virtual address (4 Gbyte address space)

    - 4 KByte page size

    - How many page table entries do we need for one process?

Page Tables

- ## Assume we have
  - 64-bit virtual address (humungous address space)
  - 4 KByte page size
  - How many page table entries do we need for one process?
- ## Problem:
  - Page table is very large
  - Access has to be fast, lookup for every memory reference
  - Where do we store the page table?
    - Registers?
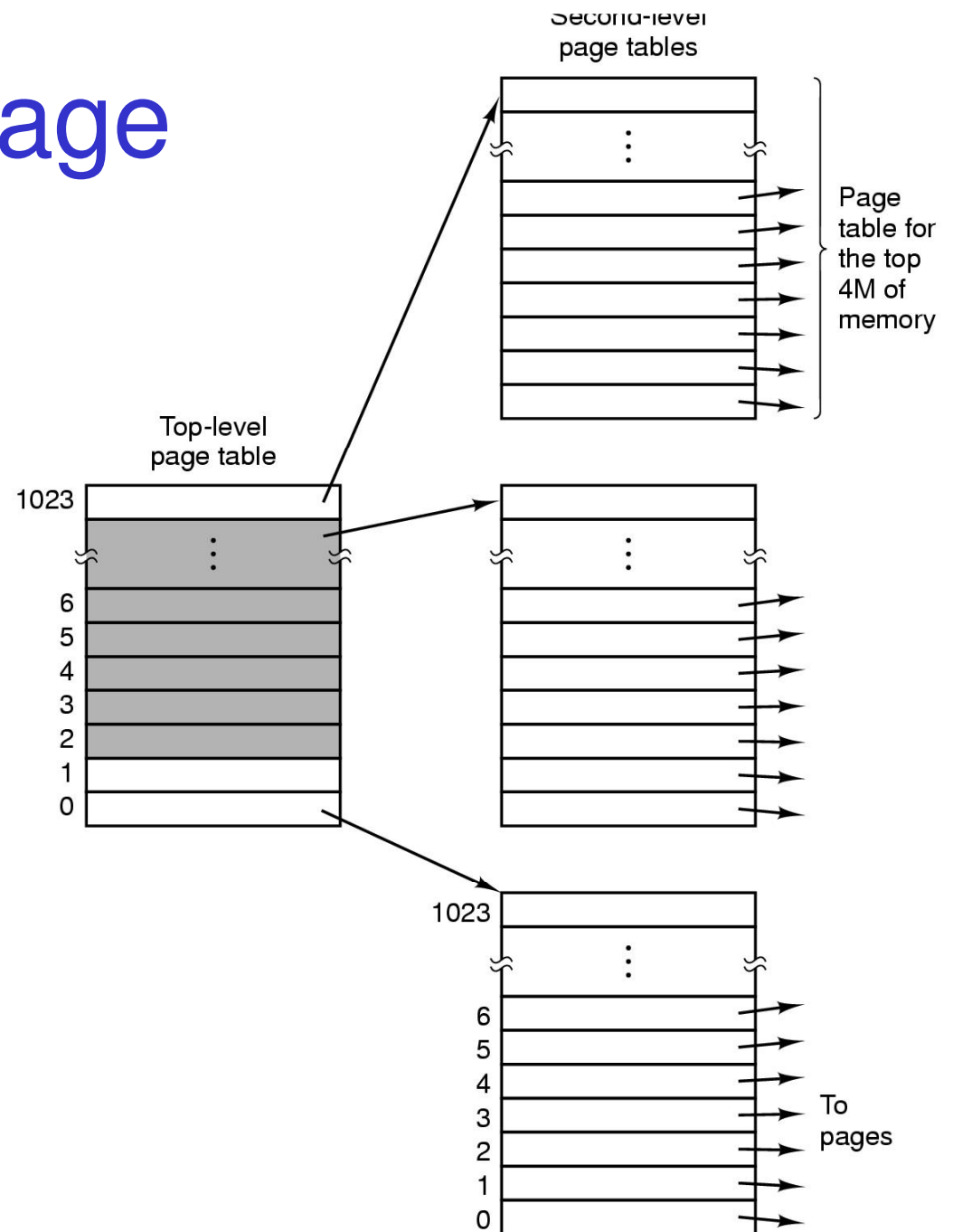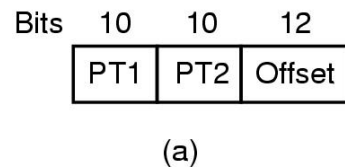   - Main memory?

Page Tables

- Page tables are implemented as data structures in main memory
- Most processes do not use the full 4GB address space
    - e.g., 0.1 – 1 MB text, 0.1 – 10 MB data, 0.1 MB stack
- We need a compact representation that does not waste space
    - But is still very fast to search
- Three basic schemes
    - Use data structures that adapt to sparsity
    - Use data structures which only represent resident pages
    - Use VM techniques for page tables (details left to extended OS)
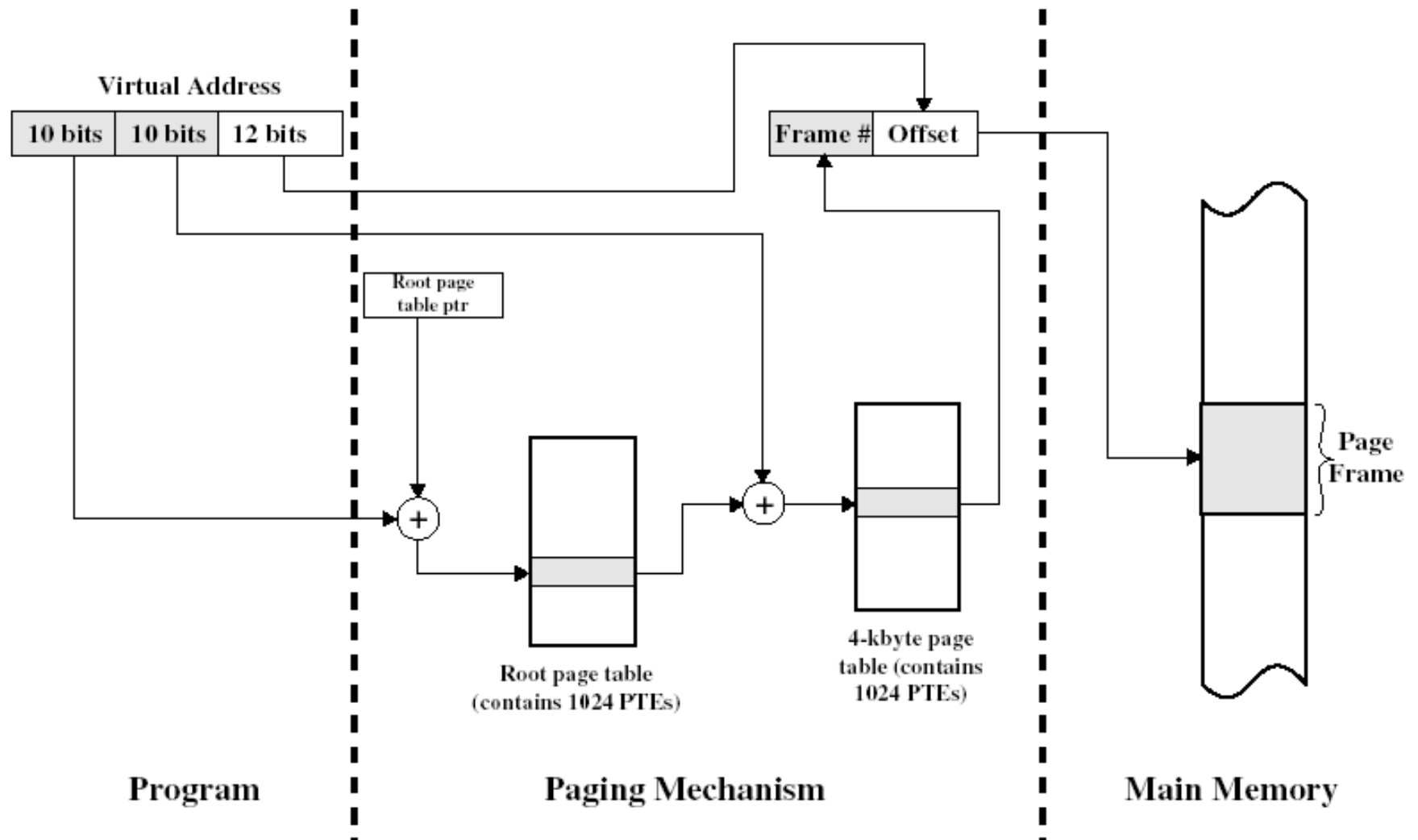
THE UNIVERSITY OF
NEW SOUTH WALES

Two-level Page Table

- 2nd –level page tables representing unmapped pages are not allocated
  - Null in the top-level page table

Bits   10   10   12

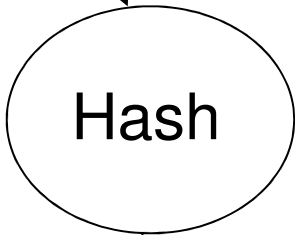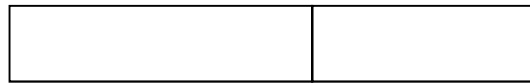| PT1 | PT2 | Offset |

(a)

Second-level
page tables

Page table for the top 4M of memory

Top-level page table

1023

6
5
4
3
2
1
0

1023

6
5
4
3
2
1
0

To pages

(b)

Two-level Translation

Example Translations

THE UNIVERSITY OF
NEW SOUTH WALES

Alternative: Inverted Page Table

PID   VPN   offset

Hash

Hash Anchor Table (HAT)
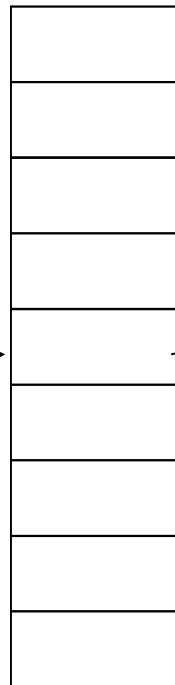
Index	PID	VPN	ctrl	next
0				
1				
2				
3				
4				
5				
6				
…				

IPT: entry for each *physical* frame

THE UNIVERSITY OF NEW SOUTH WALES

Alternative: Inverted Page Table

PID   VPN   offset
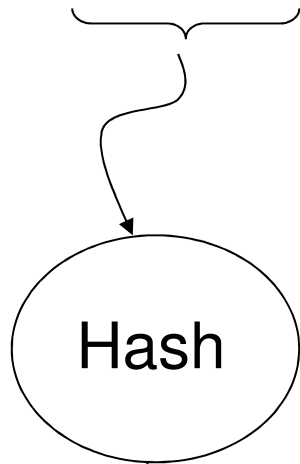
| 0 | 0x5 | 0x123 |

Hash

Hash Anchor Table
(HAT)

2

Index	PID	VPN	ctrl	next
0				
1				
2	1	0x1A		0x40C
…				
0x40C	0	0x5		0x0
0x40D				
…				
…				

ppn   offset

| 0x40C | 0x123 |

Inverted Page Table (IPT)

- "Inverted page table" is an array of page numbers sorted (indexed) by frame number (it's a frame table).

- Algorithm
  - Compute hash of page number
  - Extract index from hash table
  - Use this to index into inverted page table
  - Match the PID and page number in the IPT entry
  - If match, use the index value as frame # for translation
  - If no match, get next candidate IPT entry from chain field
  - If NULL chain entry $\Rightarrow$ page fault

THE UNIVERSITY OF
NEW SOUTH WALES

Properties of IPTs

- IPT grows with size of RAM, NOT virtual address space
- Frame table is needed anyway (for page replacement, more later)
- Need a separate data structure for non-resident pages
- Saves a vast amount of space (especially on 64-bit systems)
- Used in some IBM and HP workstations

THE UNIVERSITY OF
NEW SOUTH WALES

Given *n* processes

- how many page tables will the system have for
  - 'normal' page tables
  - inverted page tables?

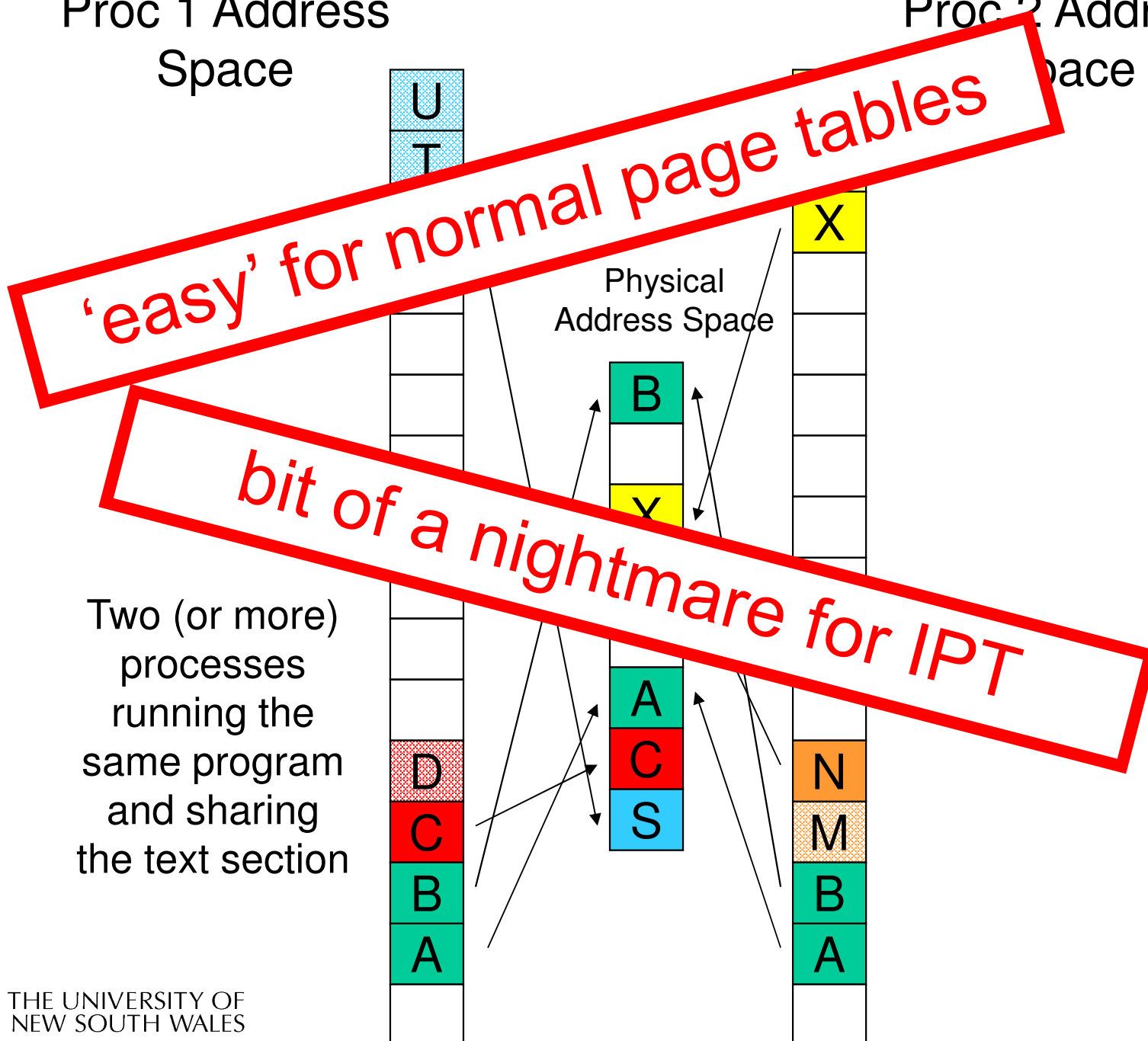# Another look at sharing...

Proc 1 Address Space

Proc 2 Address Space

Physical Address Space

U

T

X

B

X

A

C

S

Two (or more) processes running the same program and sharing the text section

D

C

B

A

N

M

B

A

'easy' for normal page tables
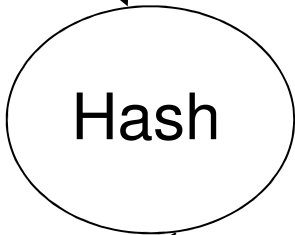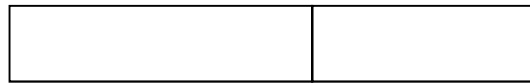
bit of a nightmare for IPT

Improving the IPT: Hashed Page Table

- Retain fast lookup of IPT
  - A single memory reference in best case

- Retain page table sized based on physical memory size (not virtual)
  - Enable efficient frame sharing
  - Support more than one mapping for same frame

THE UNIVERSITY OF
NEW SOUTH WALES

Hashed Page Table

PID   VPN   offset

Hash

Best-case lookup: one memory reference

HPT: Frame number stored in table

THE UNIVERSITY OF
NEW SOUTH WALES

Hashed Page Table

PID   VPN   offset

| 0 | 0x5 | 0x123 |

Hash
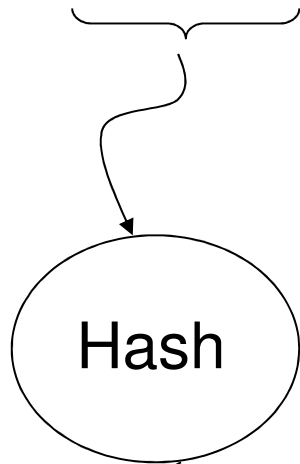
	PID	VPN	PFN	ctrl	next
1					
2					
3	0	0x5	0x42		0x0
4					
5					
6	1	0x1A	0x13		0x3
…					

ppn   offset

| 0x42 | 0x123 |

Sharing Example

Sizing the Hashed Page Table

- HPT sized based on physical memory size

- With sharing

  - Each frame can have more than one PTE

  - More sharing increases number of slots used

    - Increases collision likelihood

- However, we can tune HPT size based on:

    - Physical memory size

    - Expected sharing

    - Hash collision avoidance.

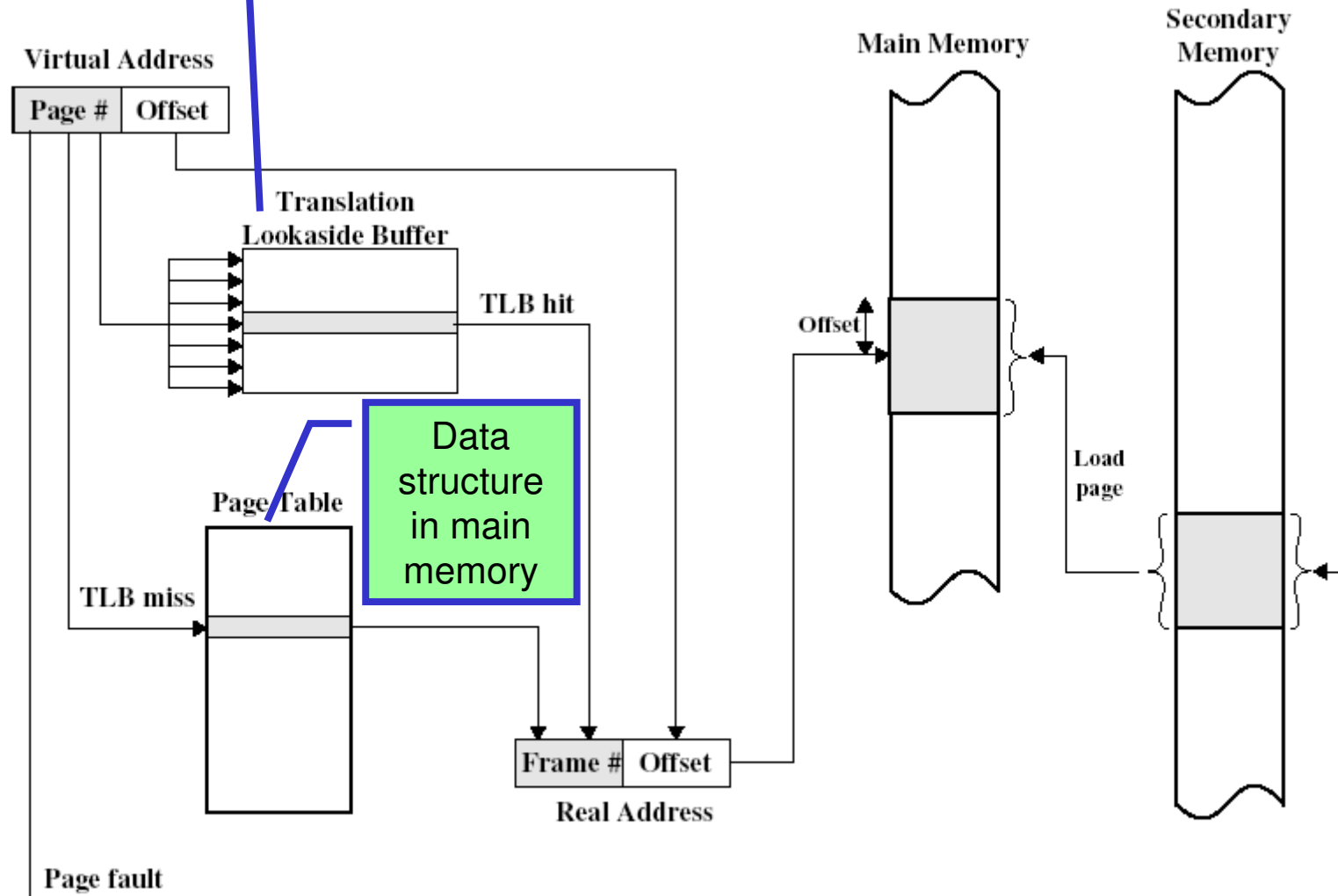 - HPT a power of 2 multiple of number of physical memory frame

VM Implementation Issue

- Performance?
  - Each virtual memory reference can cause two physical memory accesses
    - One to fetch the page table entry
    - One to fetch/store the data
    - $\Rightarrow$Intolerable performance impact!!

- Solution:
  - High-speed cache for page table entries (PTEs)
    - Called a *translation look-aside buffer* (TLB)
    - Contains recently used page table entries
    - Associative, high-speed memory, similar to cache memory
    - May be under OS control (unlike memory cache)

TLB operation

On-CPU hardware device!!!

Virtual Address

| Page # | Offset |

Translation Lookaside Buffer

TLB hit

Page Table

Data structure in main memory

TLB miss

Main Memory

Offset

Secondary Memory

Load page

| Frame # | Offset |

Real Address

Page fault

Translation Lookaside Buffer

- Given a virtual address, processor examines the TLB

- If matching PTE found (*TLB hit*), the address is translated

- Otherwise (*TLB miss*), the page number is used to index the process's page table
  - If PT contains a valid entry, reload TLB and restart
  - Otherwise, (page fault) check if page is on disk
    - If on disk, swap it in
    - Otherwise, allocate a new page or raise an exception

THE UNIVERSITY OF
NEW SOUTH WALES

TLB properties

- **Page table is (logically) an array of frame numbers**

- **TLB holds a (recently used) subset of PT entries**
  - Each TLB entry must be identified (tagged) with the page # it translates
  - Access is by associative lookup:
    - All TLB entries' tags are concurrently compared to the page #
    - TLB is associative (or content-addressable) memory

page #	frame #	V	W
. . .	. . .	.	.
. . .	. . .	.	.

THE UNIVERSITY O
NEW SOUTH WALE

TLB properties

- TLB may or may not be under direct OS control
  - Hardware-loaded TLB
    - On miss, hardware performs PT lookup and reloads TLB
    - Example: x86, ARM
  - Software-loaded TLB
    - On miss, hardware generates a TLB miss exception, and exception handler reloads TLB
    - Example: MIPS, Itanium (optionally)

- TLB size: typically 64-128 entries

- Can have separate TLBs for instruction fetch and data access

- TLBs can also be used with inverted page tables (and others)

THE UNIVERSITY OF
NEW SOUTH WALES

TLB and context switching

- TLB is a shared piece of hardware
- Normal page tables are per-process (address space)
- TLB entries are *process-specific*
  - On context switch need to *flush* the TLB (invalidate all entries)
    - high context-switching overhead (Intel x86)
  - **or** tag entries with *address-space ID* (ASID)
    - called a *tagged TLB*
    - used (in some form) on all modern architectures
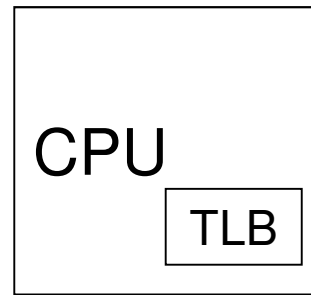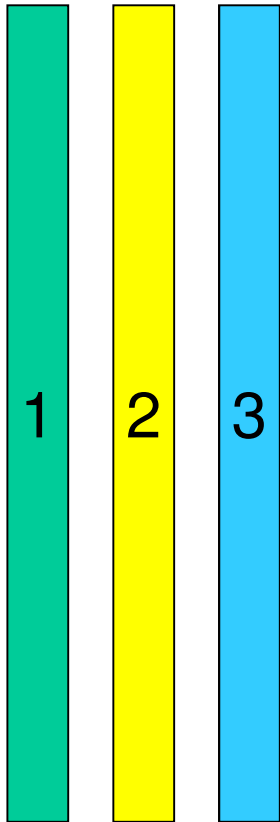   - TLB entry: ASID, page #, frame #, valid and write-protect bits

THE UNIVERSITY OF
NEW SOUTH WALES

TLB effect

- ## Without TLB
  - Average number of physical memory references per virtual reference

    $= 2$

- ## With TLB (assume 99% hit ratio)
  - Average number of physical memory references per virtual reference
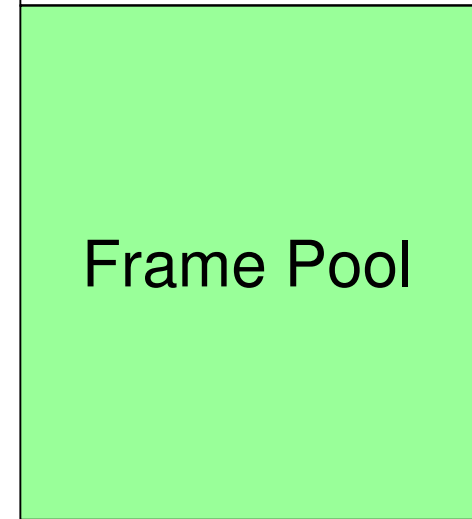
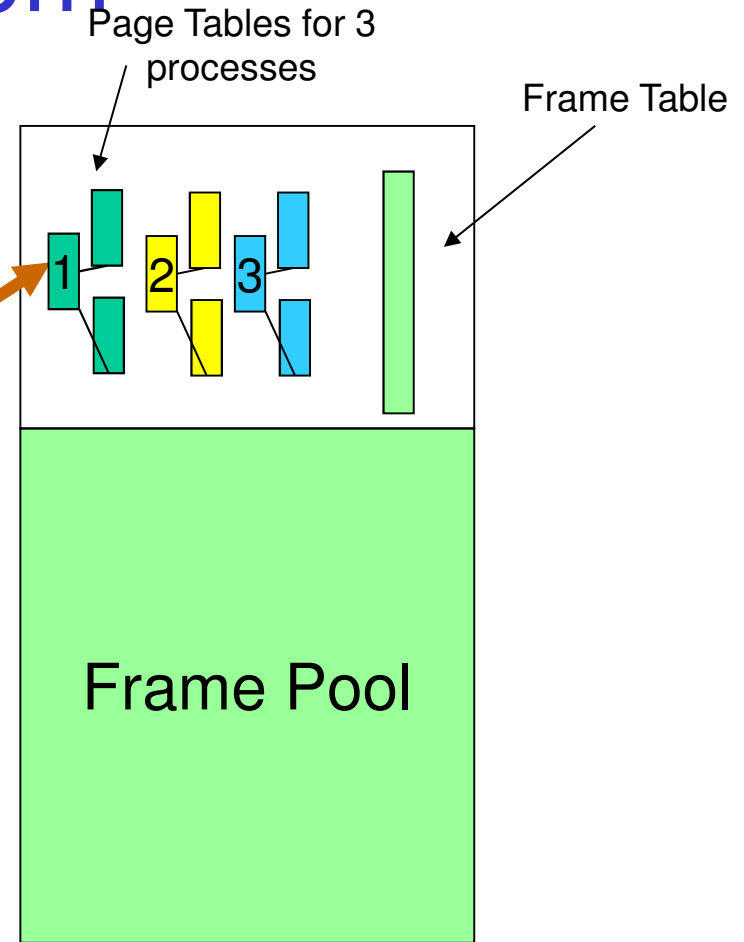 $= .99 * 1 + 0.01 * 2$

    $= 1.01$

Recap - Simplified Components of VM System
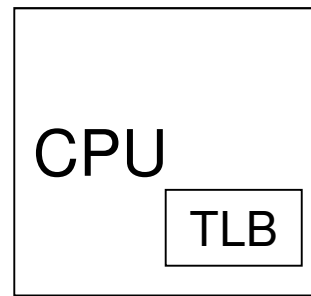
Virtual Address Spaces
(3 processes)

Page Tables for 3 processes

Frame Table

CPU

TLB

TLB Refill Mechanism

Frame Pool

Physical Memory

THE UNIVERSITY OF
NEW SOUTH WALES

41

Recap - Simplified Components of VM System

Virtual Address Spaces
(3 processes)

Inverted Page Table

TLB Refill Mechanism

CPU

TLB

1   2   3

Frame Pool

Physical Memory

THE UNIVERSITY OF
NEW SOUTH WALES

42
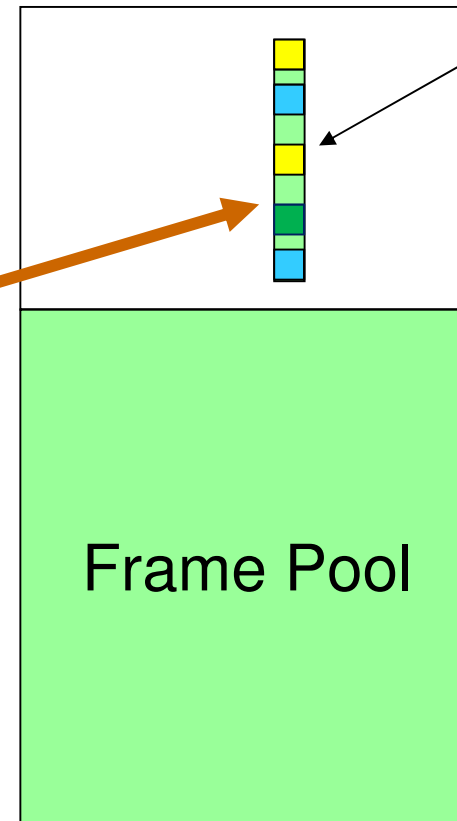
Recap - Simplified Components of VM System

Virtual Address Spaces (3 processes)

1 2 3

CPU

TLB
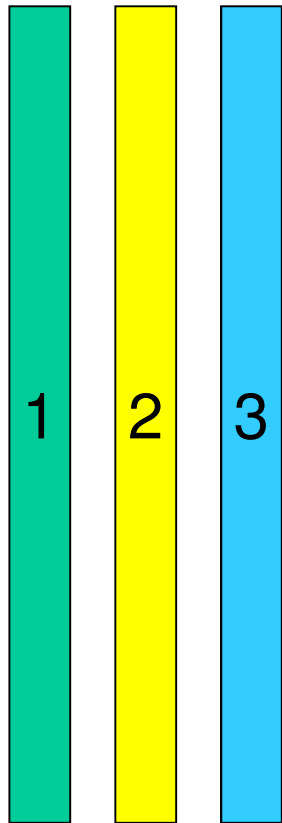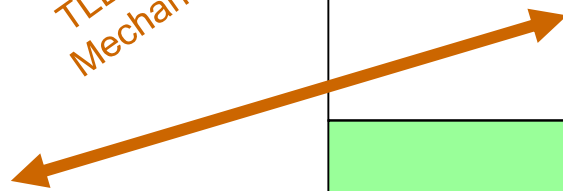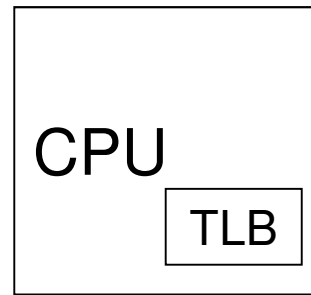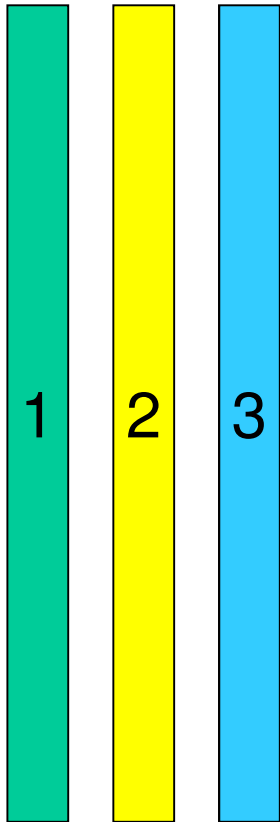
TLB Refill Mechanism

Hashed Page Table

Frame Table

Frame Pool

Physical Memory

MIPS R3000 TLB

```
31                              12  11                    6  5        0
┌─────────────────────────────────┬──────────────────────┬──────────┐
│ VPN                             │ ASID                 │ 0        │
└─────────────────────────────────┴──────────────────────┴──────────┘
```
EntryHi Register (TLB key fields)

```
31                         12  11   10    9    8    7          0
┌───────────────────────────┬────┬────┬────┬────┬─────────────┐
│ PFN                       │ N  │ D  │ V  │ G  │ 0           │
└───────────────────────────┴────┴────┴────┴────┴─────────────┘
```
EntryLo Register (TLB data fields)

- N = Not cacheable
- D = Dirty = Write protect
- G = Global (ignore ASID in lookup)

- V = valid bit
- 64 TLB entries
- Accessed via software through Cooprocessor 0 registers
  - EntryHi and EntryLo

THE UNIVERSITY OF
NEW SOUTH WALES

44
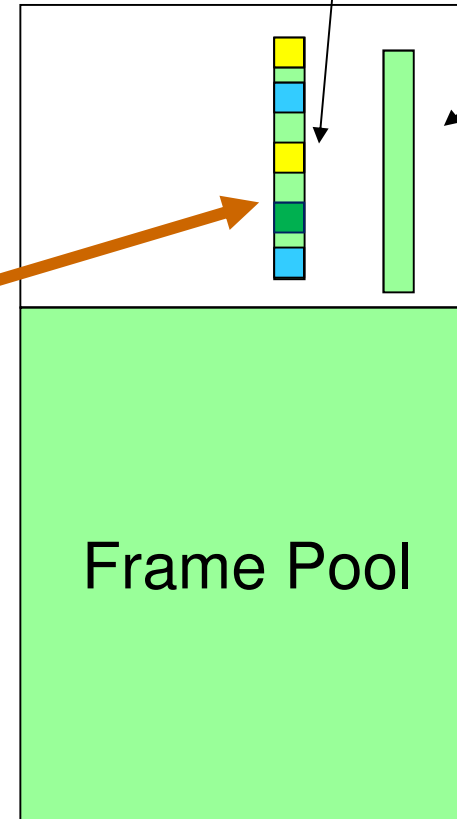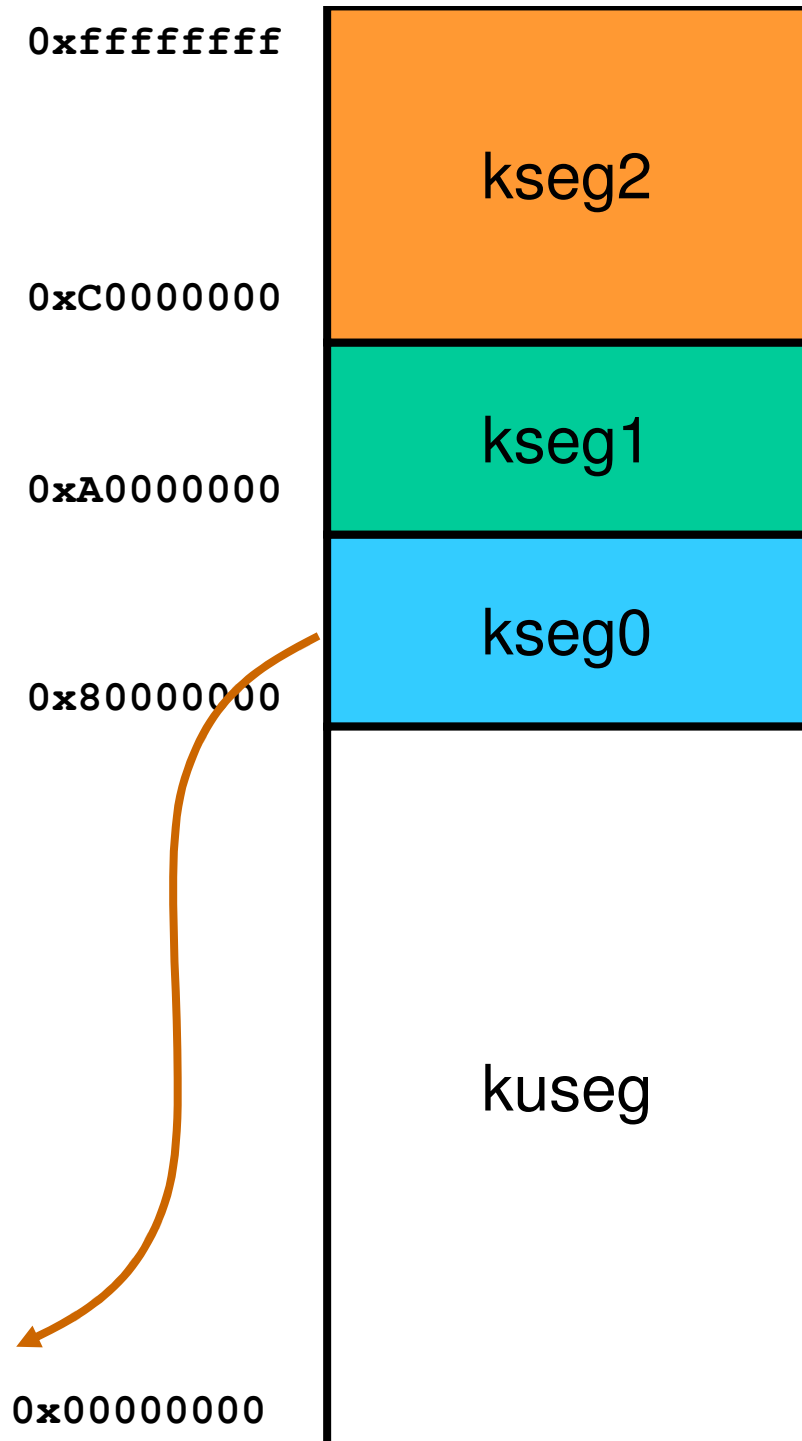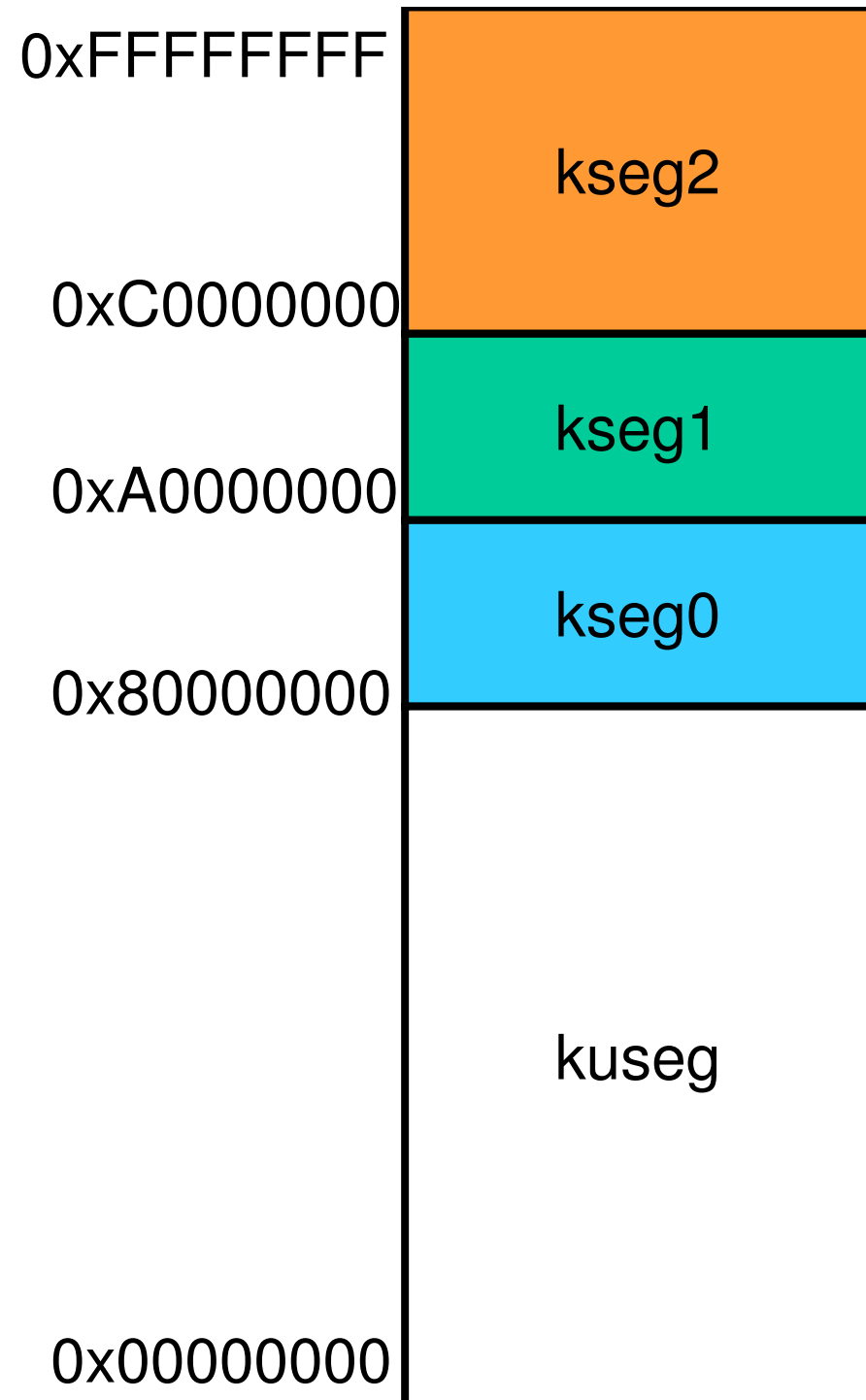
R3000 Address Space Layout

- ## kseg0:
  - 512 megabytes
  - Fixed translation window to physical memory
    - 0x80000000 - 0x9fffffff virtual = 0x00000000 - 0x1fffffff physical
    - TLB not used
  - Cacheable
  - Only kernel-mode accessible
  - Usually where the kernel code is placed

0xffffffff

kseg2

0xC0000000

kseg1

0xA0000000

kseg0

0x80000000

kuseg

Physical Memory

0x00000000

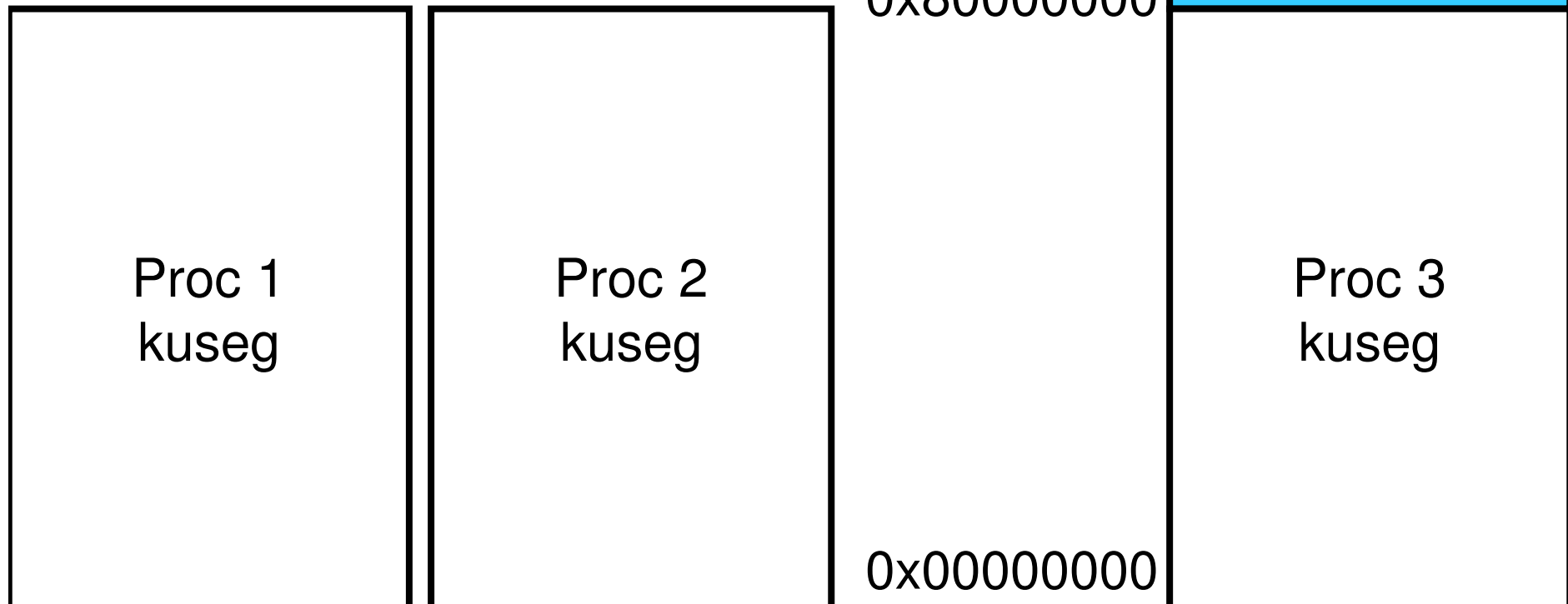THE UNIVERSITY OF
NEW SOUTH WALES

R3000 Address Space Layout

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

0xFFFFFFFF	kseg2
0xC0000000	kseg1
0xA0000000	kseg0
0x80000000	kuseg
0x00000000	

R3000 Address Space Layout

– Switching processes switches the translation (page table) for kuseg

0xFFFFFFFF

kseg2

0xC0000000

kseg1

0xA0000000

kseg0

0x80000000

0x00000000

Proc 1 kuseg

Proc 2 kuseg

Proc 3 kuseg

R3000 Address Space Layout

- kseg1:
  - 512 megabytes
  - Fixed translation window to physical memory
    - 0xa0000000 - 0xbfffffff virtual = 0x00000000 - 0x1fffffff physical
    - TLB not used
  - **NOT** cacheable
  - Only kernel-mode accessible
  - Where devices are accessed (and boot ROM)

0xffffffff

kseg2

0xC0000000

kseg1

0xA0000000

kseg0

0x80000000

kuseg

Physical Memory

0x00000000

THE UNIVERSITY OF NEW SOUTH WALES