Exception Modes

- FIQ: fast interrupt request
- IRQ: interrupt request
- SVC: SWI and reset
  - protected mode for OS
- abort: prefetch abort and data abort
  - (virtual) memory protection
- undefined: undefined instruction
  - Software emulation of HW coprocessors

IRQ and FIQ exceptions

- IRQ and FIQ exceptions only occur when a specific interrupt mask is cleared in cpsr.
- The ARM processor automaticaaly completes the instruction in the execution stage before handling the interrupt.
  1. Change the processor mode
  2. Save cpsr to spsr
  3. Save pc to lr
  4. Interrupt(s) are disabled
  5. Branch to a specific entry in the vector table.

Vector Table

- A table of addresses that the ARM core branches to when an exception is raised.

```
RSET    0x0000    ; 0xffff0000 if in the higher address
UNDEF   0x0004
SWI     0x0008
PABT    0x000c    ; prefetch abort
DABT    0x0010    ;data abort
-       0x0014    ; reserved
IRQ     0x0018
FIQ     0x001c    ; 0xffff001c if in the higher address
```

Enabling and disabling IRQ and FIQ exceptions

```
enabling_irq
        MRS     r1, cpsr
        BIC     r1, r1, #0x80
        MSR     cpsr_c, r1

enabling_fiq
        MRS     r1, cpsr
        BIC     r1, r1, #0x40
        MSR     cpsr_c, r1
```

```
disabling_irq
        MRS     r1, cpsr
        ORR     r1, r1, #0x80
        MSR     cpsr_c, r1

disabling_fiq
        MRS     r1, cpsr
        ORR     r1, r1, #0x40
        MSR     cpsr_c, r1
```

BIC: logical bit clear (AND NOT), ORR: logical bitwise or

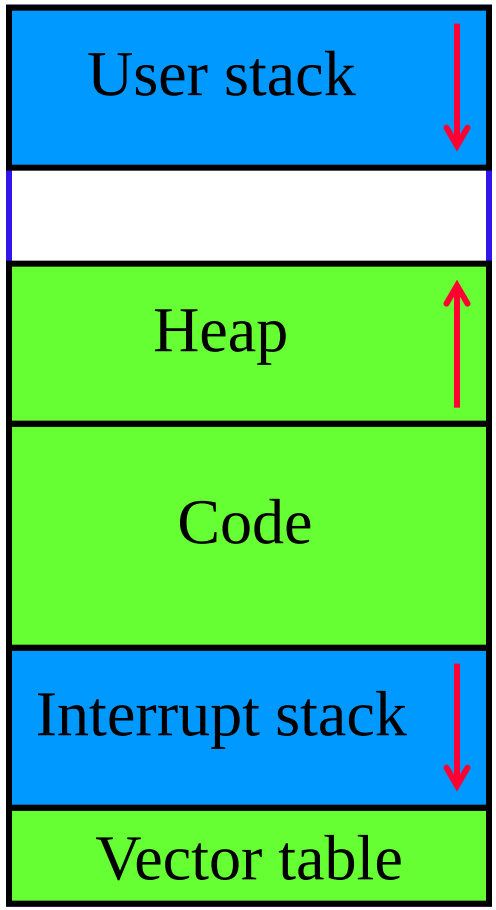
Basic interrupt stack

- Exception handlers make intensive use of stacks
- A good stack design tries to avoid stack overflow
  - Use memory protection
  - Call a stack check function
- The stack for each processor mode has to be set up in the initialization code

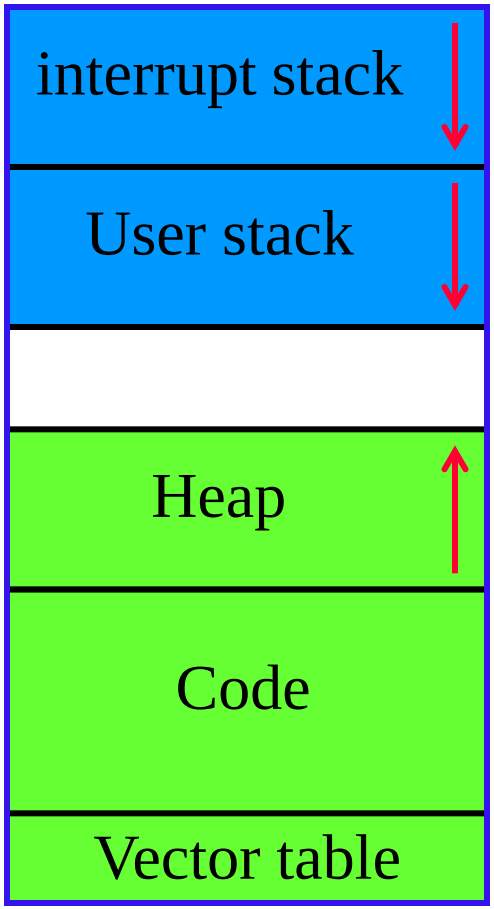
Typical memory layouts

A

traditional

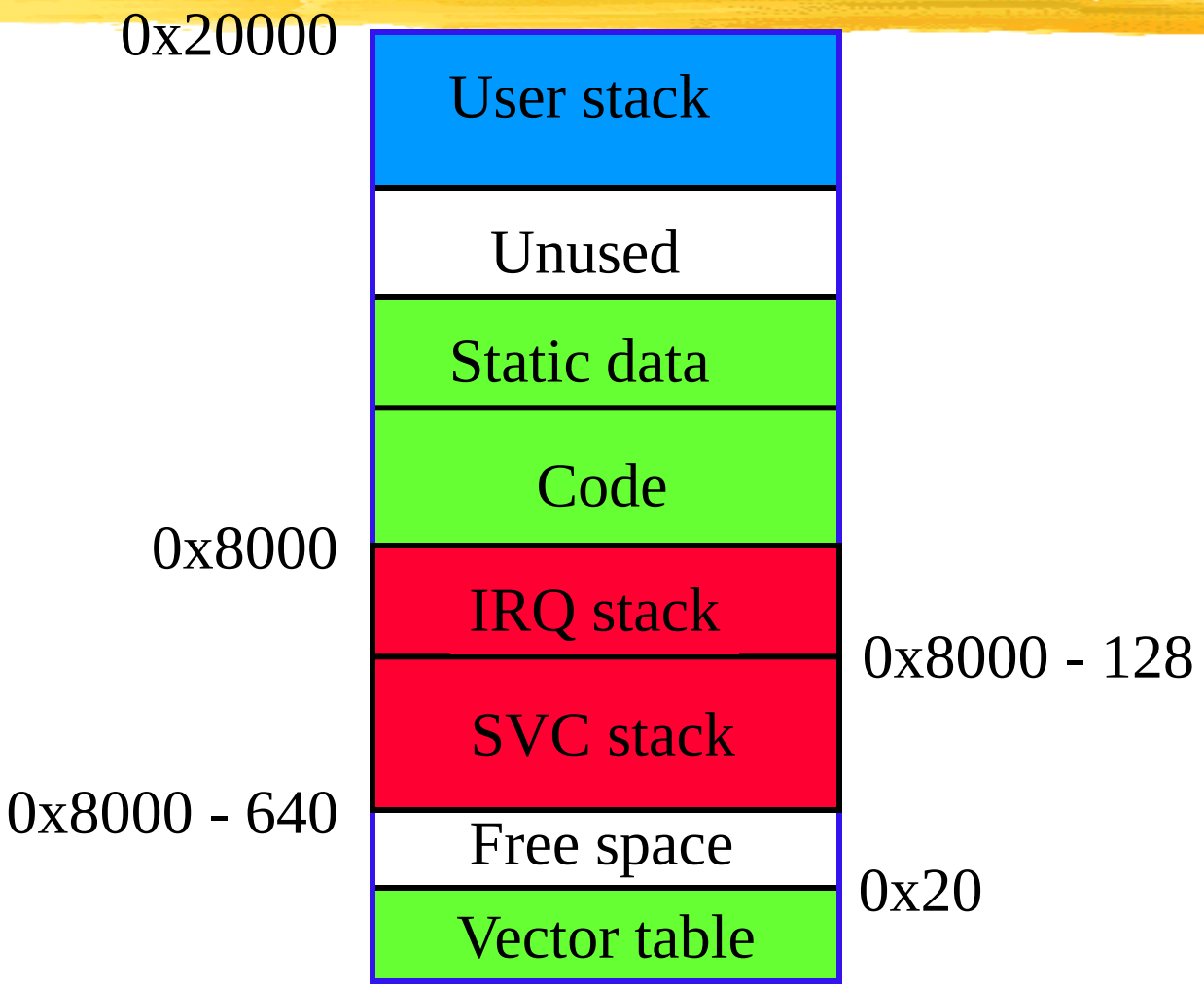


B



Layout B does not corrupt the vector table when a stack overflow occurs

Example: Basic interrupt stack



Stack design

- Depends on
  - OS requirement for stack design
  - Target HW provides a physical limit to the size and positioning of the stack memory.
- ARM-based system : stack grow downward with top of the stack at a high memory address
- Stack overflow must be avoided

SVC Stack Setup

```
    USER_Stack          EQU     0x200000
    IRQ_Stack           EQU     0x8000
    SVC_Stack           EQU     IRQ_Stack – 128
    USER32md            EQU     0x10            ;User mode
    IRQ32md             EQU     0x12            ;IRQ mode
    SVC32md             EQU     0x13            ;SVC mode
    Sys32md             EQU     0x1f            ;System mode
    NoInt               EQU     0xc0            ;disable interrupts
; supervisor mode stack set up : core starts in supervisor mode
        LDR             r13, SVC_NewStack       ; r13_svc
        . . .
SVC_NewStack
        DCD             SVC_Stack
```

IRQ Stack Setup

```
; IRQ mode stack set up : change to IRQ mode
        MOV         r2, #NoInt | IRQ32md
        MSR         cpsr_c, r2
        LDR         r13, IRQ_NewStack      ; r13_irq
        . . .
IRQ_NewStack        DCD             IRQ_Stack

; USER mode stack set up :  last in the initialization code
        MOV         r2, Sys32md
        MSR         cpsr_c, r2
        LDR         r13, USR_NewStack      ; r13_usr
        . . .
USR_NewStack        DCD             USR_Stack
```

IRQ Handler

```
; assume that the IRQ stack has been set up by the initialization code
Interrupt_handler
        SUB     r14, r14, #4                ; adjust lr
        STMFD sp!, {r0-r3, r12, r14}        :save context
        LDR     r0, =IRQStatus              ;interrupt status address
        LDR     r0, [r0]                    ;get interrupt status
        TST     r0, #0x0080                 ;if counter timer
        BNE     timer_isr                   ;           then branch to ISR
        TST     r0,#0x0001                  ;else if button press
        BNE     button_isr                  ;           then call button ISR
        LDMFD sp!,{r0-r3,r12,pc}^           ;return
```

Often confused

- *Exception:* an internal CPU event such as
  - floating point overflow
  - MMU fault (e.g., page fault)
  - trap (SWI)
- *Interrupt:* an external I/O event such as
  - I/O device request
  - reset
- In the ARM architecture manuals, the two terms are mixed together

SWI exception

- Only a branch to the SWI handler is at x08.

```
SWI_handler
        STMFD       sp!, {r0-r12,r14}       ; save context
        LDR         r10, [r14,#-4]          ; load SWI instruction
                                            ; lr points to next instruction
        BIC         r10, r10, #0xff000000   ;mask off the MSB 8 bits
        MOV         r1,r13                  ;copy SVC stack to r1
        MRS         r2, spsr                ;copy spsr to r2
        STMFD       sp!, {r2}               ; save r2  on stack
        BL          swi_jumptable           ;branch to swi_jumptable
```

SWI exception

- Only a branch to the SWI handler is at x08.

```
        LDMFD       sp!, {r2}               ;restore r2 (spsr)
        MSR         spsr_cxsf, r2           ;copy r2 back to cpsr
                    ; control, extension, status, flags bytes
        LDMFD       sp!, {r0-r12, pc}       ;restore context and return
swi_jumptable
        MOV         r0,r10                  ;mov SWI number to r0
                                            ; SWI number in r1
        B           eventSWIhanlder         ;branch to SWI handler
```

Exception

- Exception: internally detected error.
- Exceptions are synchronous with instructions but unpredictable.
- Build exception mechanism on top of interrupt mechanism.
- Exceptions are usually prioritized and vectorized.

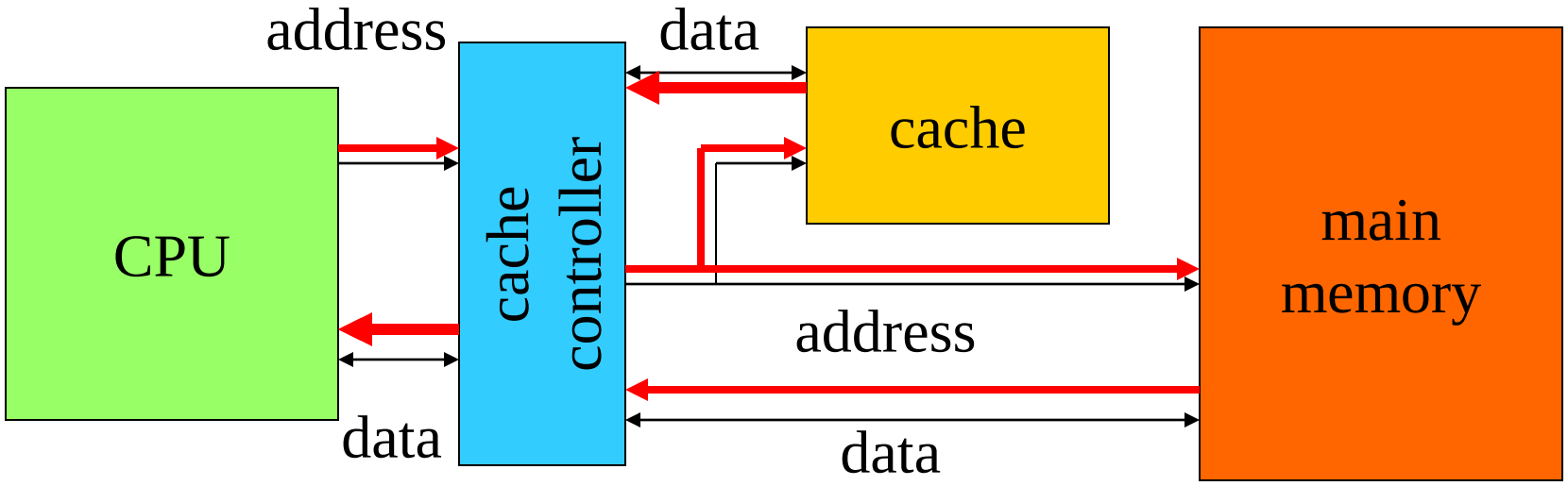
Trap

- Trap (software interrupt): an exception generated by an instruction.
  - Call supervisor mode.
- ARM uses SWI instruction for traps.

Co-processor

- Co-processor: added function unit that is called by instruction.
  - Floating-point units are often structured as co-processors.
- ARM allows up to 16 designer-selected co-processors.
  - Floating-point co-processor uses units 1, 2.

Caches and CPUs



Cache operation

- Many main memory locations are mapped onto one cache entry.
- May have caches for:
  - instructions;
  - data;
  - data + instructions (unified).
- Memory access time is no longer deterministic.

Terms

- Cache hit: required location is in cache.
- Cache miss: required location is not in cache.
- Working set: set of locations used by program in a time interval.

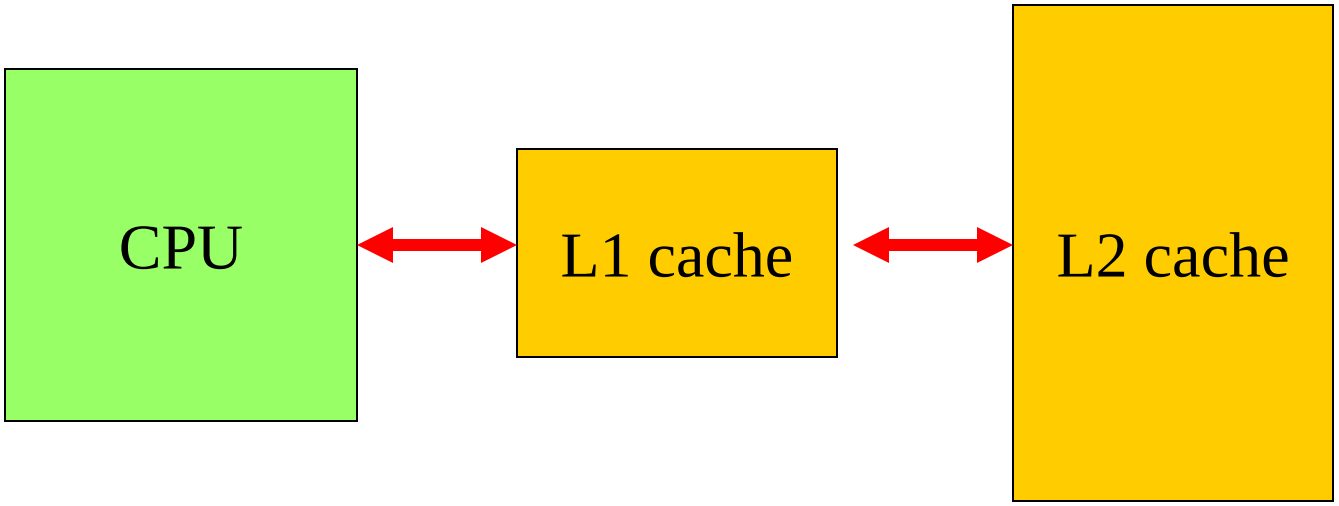
Types of misses

- Compulsory (cold) miss : location has never been accessed.
- Capacity miss: working set is too large.
- Conflict miss: multiple locations in working set map to same cache entry.

Memory system performance

- h = cache hit rate.
- t_{cache} = cache access time, t_{main} = main memory access time.
- Average memory access time:
  - $t_{av} = ht_{cache} + (1-h)t_{main}$

Multiple levels of cache



Multi-level cache access time

- h_1 = cache hit rate.
- h_2 = hit rate on L2 and miss on L1.
- Average memory access time:
  - $t_{av} = h_1 t_{L1} + (1-h_1)h_2 t_{L2} + (1- h_2)(1-h_1)t_{main}$
    $= h_1 t_{L1} + h_{2*} t_{L2} + (1- h_1-h_{2*})t_{main}$

Replacement policies

- Replacement policy: strategy for choosing which cache entry to throw out to make room for a new memory location.
- Two popular strategies:
  - Random.
  - Least-recently used (LRU).

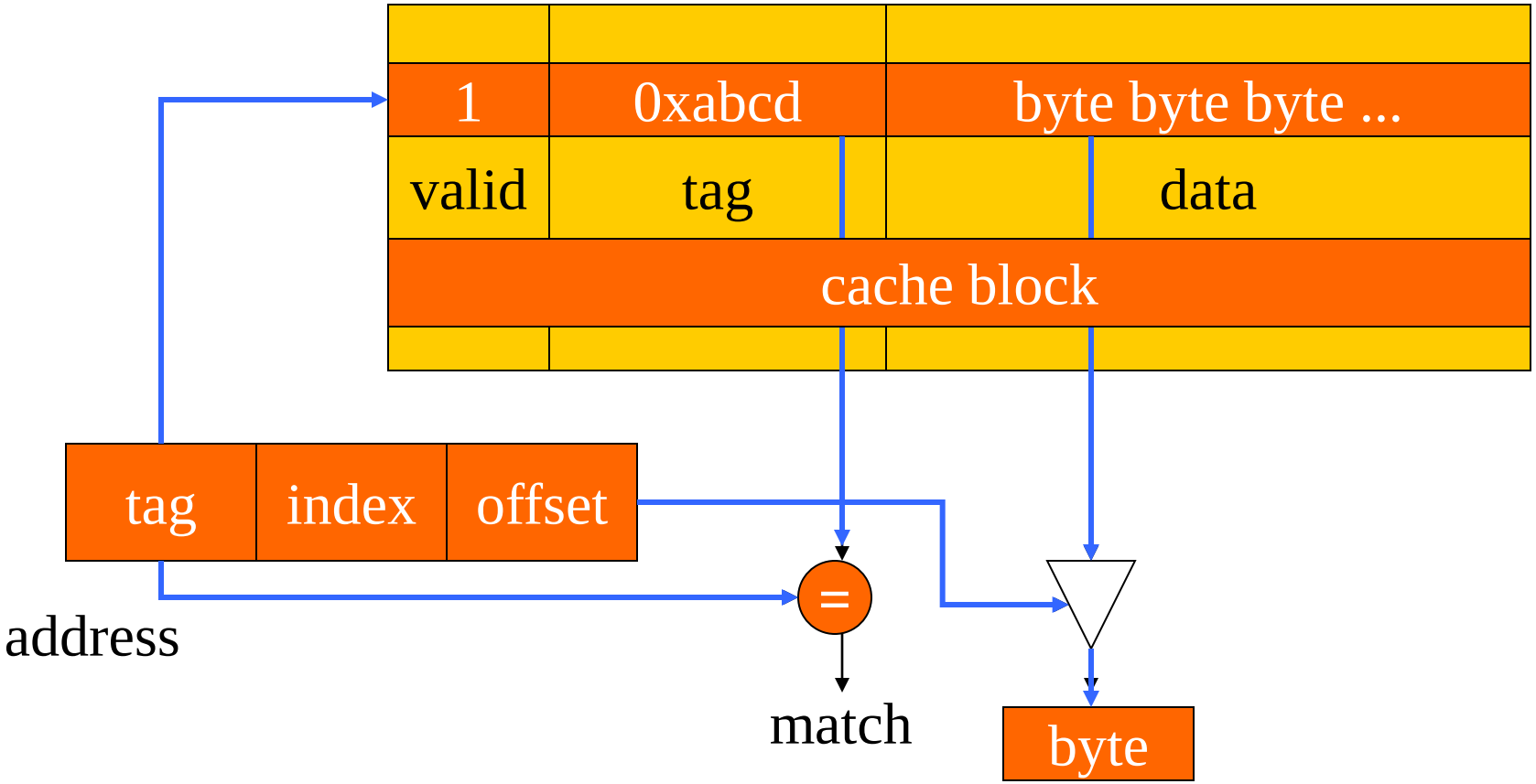
Cache organizations

- Fully-associative: any memory location can be stored anywhere in the cache (almost never implemented).
- Direct-mapped: each memory location maps onto exactly one cache entry.
- N-way set-associative: each memory location can go into one of n sets.

Cache performance benefits

- Keep frequently-accessed locations in fast cache.
- Cache retrieves more than one word at a time.
  - Sequential accesses are faster after first access.
  - Spatial locality

Direct-mapped cache



hit=match and valid

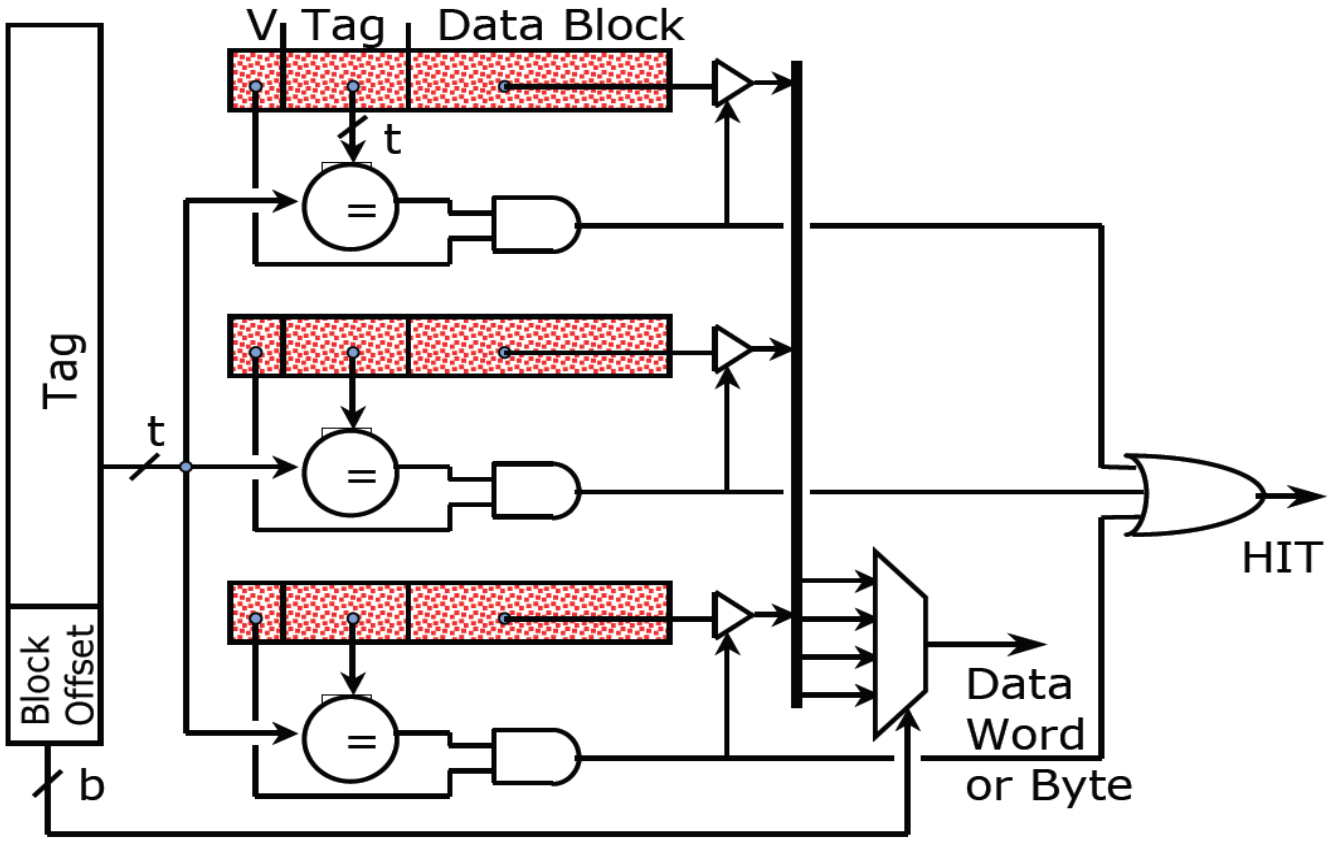
Write operations

- Write-through: immediately copy write to main memory.
- Write-back: write to main memory only when location is removed from cache.

Direct-mapped cache locations

- Many locations map onto the same cache block.
- Conflict misses are easy to generate:
  - Array a[] uses locations 0, 4, 8, …
  - Array b[] uses locations 1024, 1028, 1032, …
  - Operation a[i] + b[i] generates conflict misses.

Fully Associative Cache



Example caches

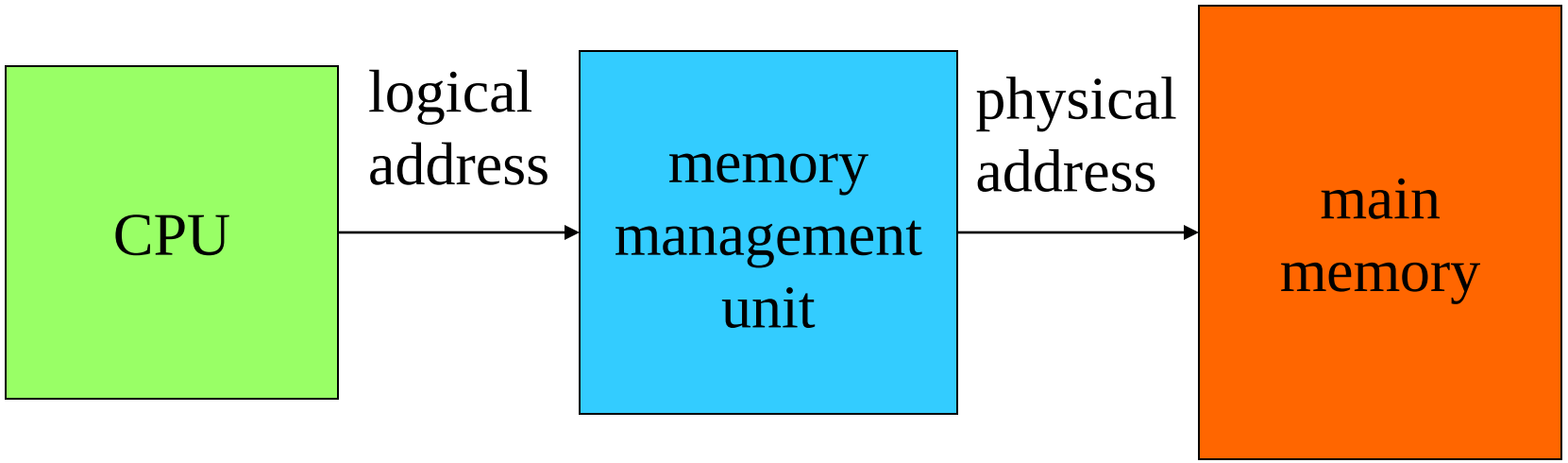
- StrongARM:
  - 16 Kbyte, 32-way, 32-byte block instruction cache.
  - 16 Kbyte, 32-way, 32-byte block data cache (write-back).
- C55x:
  - Various models have 16KB, 24KB cache.
  - Can be used as scratch pad memory.

Scratch pad memories

- Alternative to cache:
  - Software determines what is stored in scratch pad.
- Provides predictable behavior at the cost of software control.
- C55x cache can be configured as scratch pad.

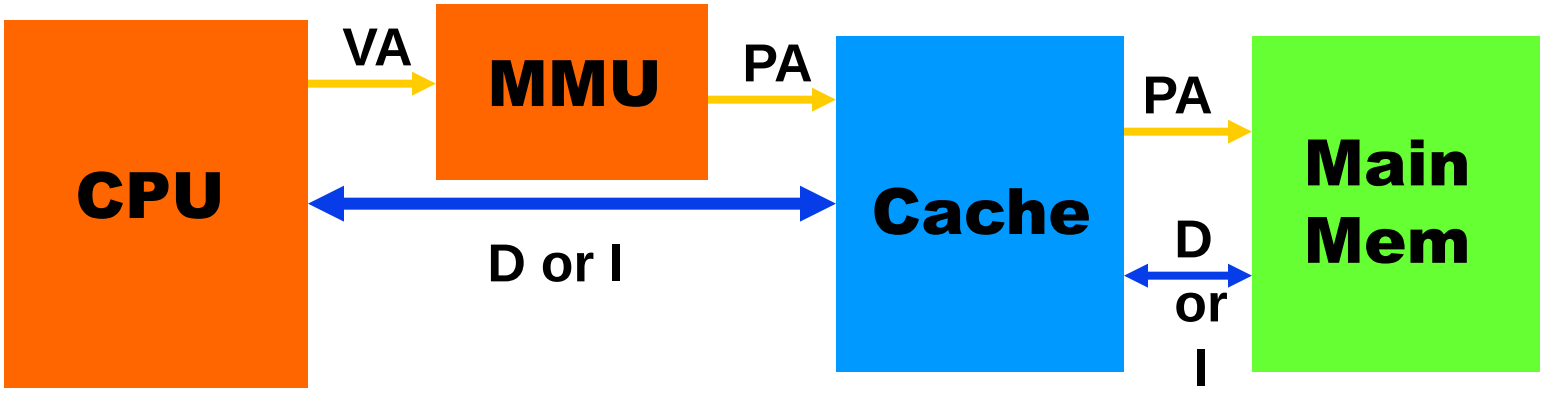
Memory management units

- Memory management unit (MMU) translates addresses:



MMU

- Responsible for
  VIRTUAL → PHYSICAL
  address mapping
- Sits between CPU and cache



Access time comparison

Media	Read	Write	Erase
DRAM	**60ns (2B) 2.56us (512B)**	**60ns (2B) 2.56us (512B)**	**N/A**
NOR flash	**150ns (2B) 14.4us (512B)**	**211us (2B) 3.53ms (512B)**	**1.2s (128KB)**
NAND flash	**10.2us (2B) 35.9us (512B)**	**201us (2B) 226us (512B)**	**2ms (16KB, 128K)**
Disk	**12.5ms (512B) (Average seek)**	**14.5ms (512B) (Average seek)**	**N/A**

- Price
  - HDD<<NAND<DRAM<NOR

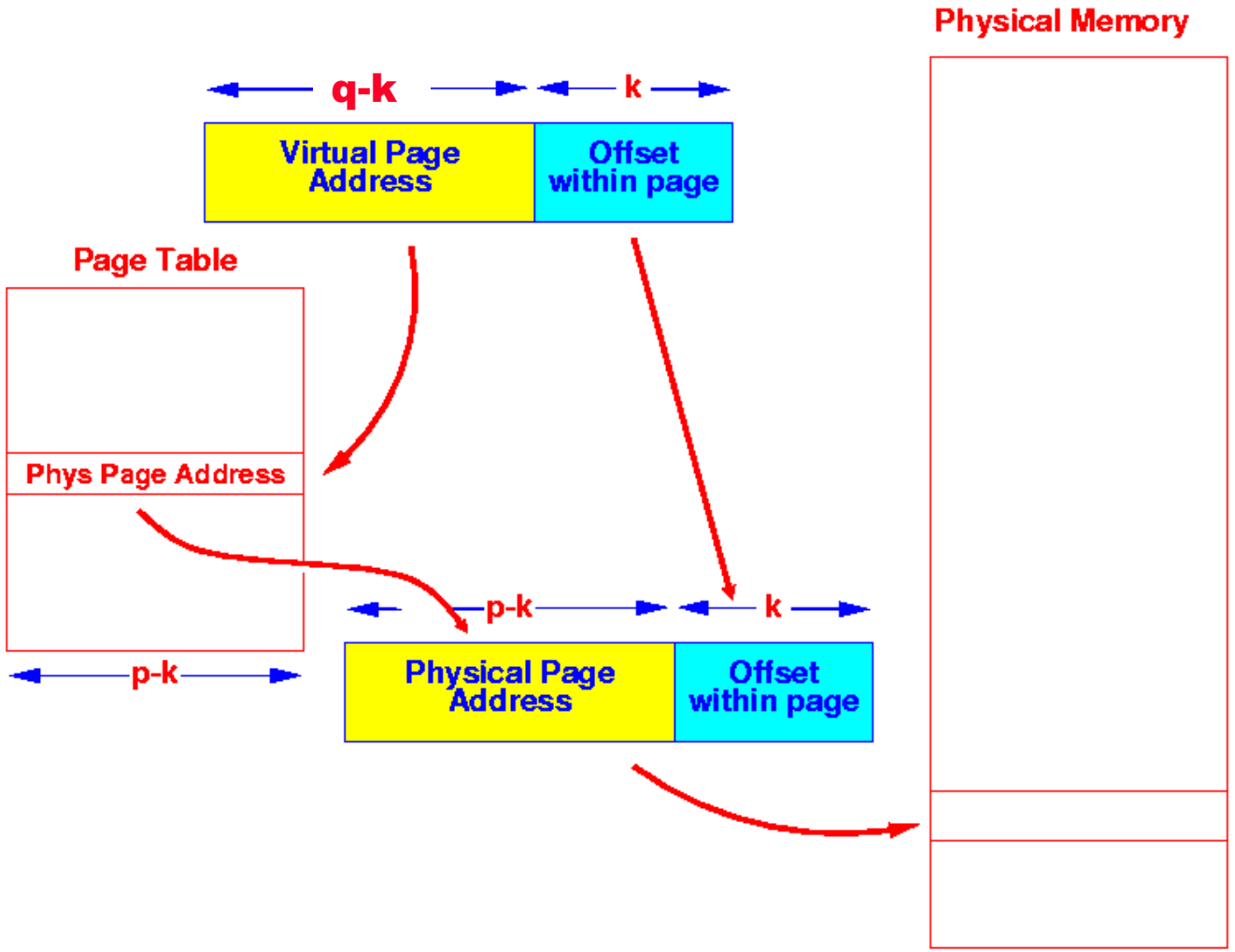
MMU - operation

- Operating System allocates pages of physical memory to users
- OS constructs page tables - *one for each user*
- Page address from memory address selects a page table entry
- Page table entry contains physical page address

Memory management tasks

- Allows programs to move in physical memory during execution.
- Allows virtual memory:
  - memory images kept in secondary storage;
  - images returned to main memory on demand during execution.
- Page fault: request for location not resident in memory.

MMU – address translation



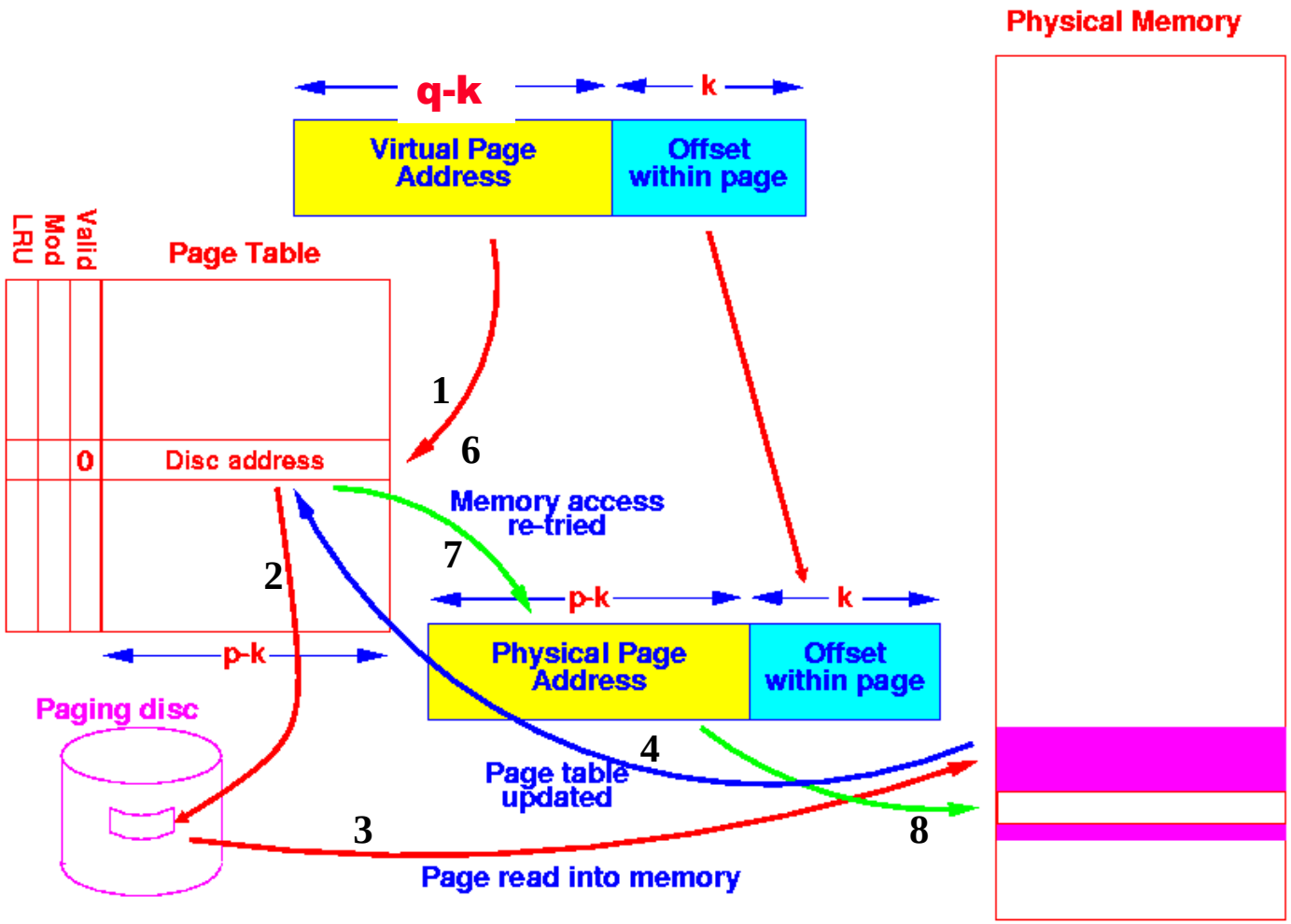
MMU - Virtual memory space

- Page Table Entries can also point to disc blocks
  - If Valid bit is set, page in memory (address is physical page address); cleared, page "swapped out" (address is disc block address)
  - MMU hardware generates page fault when swapped out page is requested
- Allows virtual memory space to be larger than physical memory
  - Only "working set" is in physical memory
  - Remainder on paging disc

MMU - Page Faults

- Page Fault Handler
  - Part of OS kernel
  - Finds usable physical page
    - LRU algorithm
  - Writes it back to disc if modified
  - Reads requested page from paging disc
  - Adjusts page table entries
  - Memory access re-tried

Page Fault


Physical Memory
q-k
k
Virtual Page Address
Offset within page
LRU
Mod
Valid
Page Table
1
6
0
Disc address
Memory access re-tried
7
2
p-k
k
Physical Page Address
Offset within page
p-k
Paging disc
4
Page table updated
3
8
Page read into memory

MMU - Page Faults

- Page Fault Handler
  - Part of OS kernel
  - Finds usable physical page
    - LRU algorithm
  - Writes it back to disc if modified
  - Reads requested page from paging disc
  - Adjusts page table entries
  - Memory access re-tried
- Can be an expensive process!
  - Usual to allow page tables to be swapped out too!
  - ⇨ Page fault can be generated on the page tables!

MMU - practicalities

- Page size
  - 8 kbyte pages $\Rightarrow k = 13$
  - $q = 32,\ q - k = 19$
  - So page table size
    - $2^{19} \approx 0.5 \text{ x } 10^6$ entries
    - Each entry 4 bytes

      $\Rightarrow 0.5 \text{ x } 10^6 \times 4 \approx 2$ Mbytes!
- Page tables can take a lot of memory!
  - Larger page sizes reduce page table size *but* can waste space (fragmentation)

MMU - practicalities

- Page tables are stored in main memory
  - They're too large to be in smaller memories!
  - MMU needs to read page table for address translation
  - ∴ Address translation can require additional memory accesses!

MMU - Protection

- Page table entries
  - Extra bits are added to specify access rights
    - Set by OS (software)

    but

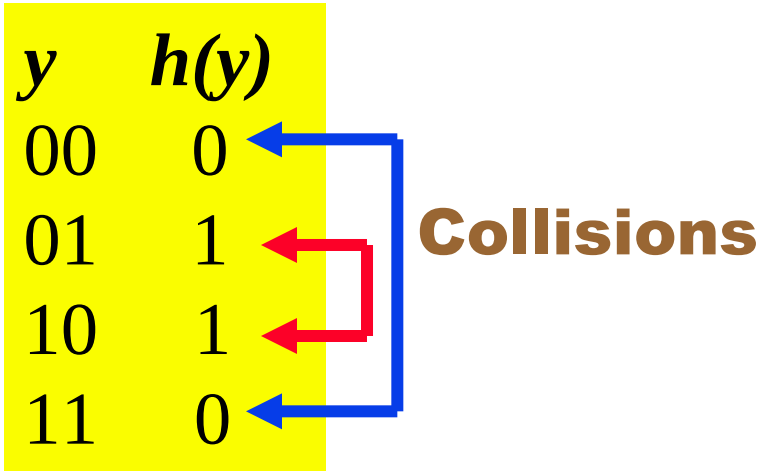
    - Checked by MMU hardware!
  - Access control bits
    - Read
    - Write
    - Read/Write
    - Execute only

MMU - Alternative Page Table Styles

- Inverted Page tables
  - One page table entry (PTE) / page of physical memory
  - MMU has to search for correct VA entry
  - ∴ PowerPC hashes VA → PTE address
    - PTE address = h(VA)
    - h – hash function
      - Hashing ⇒ collisions
- Hash functions in hardware
  - "hash" of n bits to produce m bits (Usually $m < n$)
  - Fewer bits reduces information content

MMU - Alternative Page Table Styles

- Hash functions in hardware
  - "Fewer bits reduces information content
    - There are only 2^m distinct values now!
    - Some n-bit patterns will reduce to the same m-bit patterns
  - Trivial example
    - 2-bits $\rightarrow$ 1-bit with xor
    - $h(x_1\ x_0) = x_1 \text{ xor } x_0$



MMU - Alternative Page Table Styles

- Inverted Page tables
  - One page table entry (PTE) / page of physical memory
  - MMU has to search for correct VA entry
  - ∴ PowerPC hashes VA → PTE address
    - PTE address = h(VA)
    - h – hash function
      - Hashing ⇒ collisions
      - PTEs are linked together
        - PTE contains tags (like cache) and link bits
      - MMU searches linked list to find correct entry
  - Smaller Page Tables / Longer searches

Address Translation - Speeding it up

- Two+ memory accesses for each datum?
  - Page table 1 - 3 (single - 3 level tables)
  - Actual data 1
  - *system can be slowed down*
- Translation Look-Aside Buffer
  - Acronym: TLB or TLAB
  - Small cache of recently-used page table entries
  - Usually fully-associative
  - Can be quite small!

Address Translation - Speeding it up

- TLB sizes
  - MIPS R10000 1996 64 entries
  - Pentium 4 (Prescott) 2006 64 entries
- One page table entry / page of data
- *Locality of reference*
  - Programs spend a lot of time in same memory region

⇨ TLB hit rates tend to be very high

- 98%

⇨ Compensate for cost of a miss

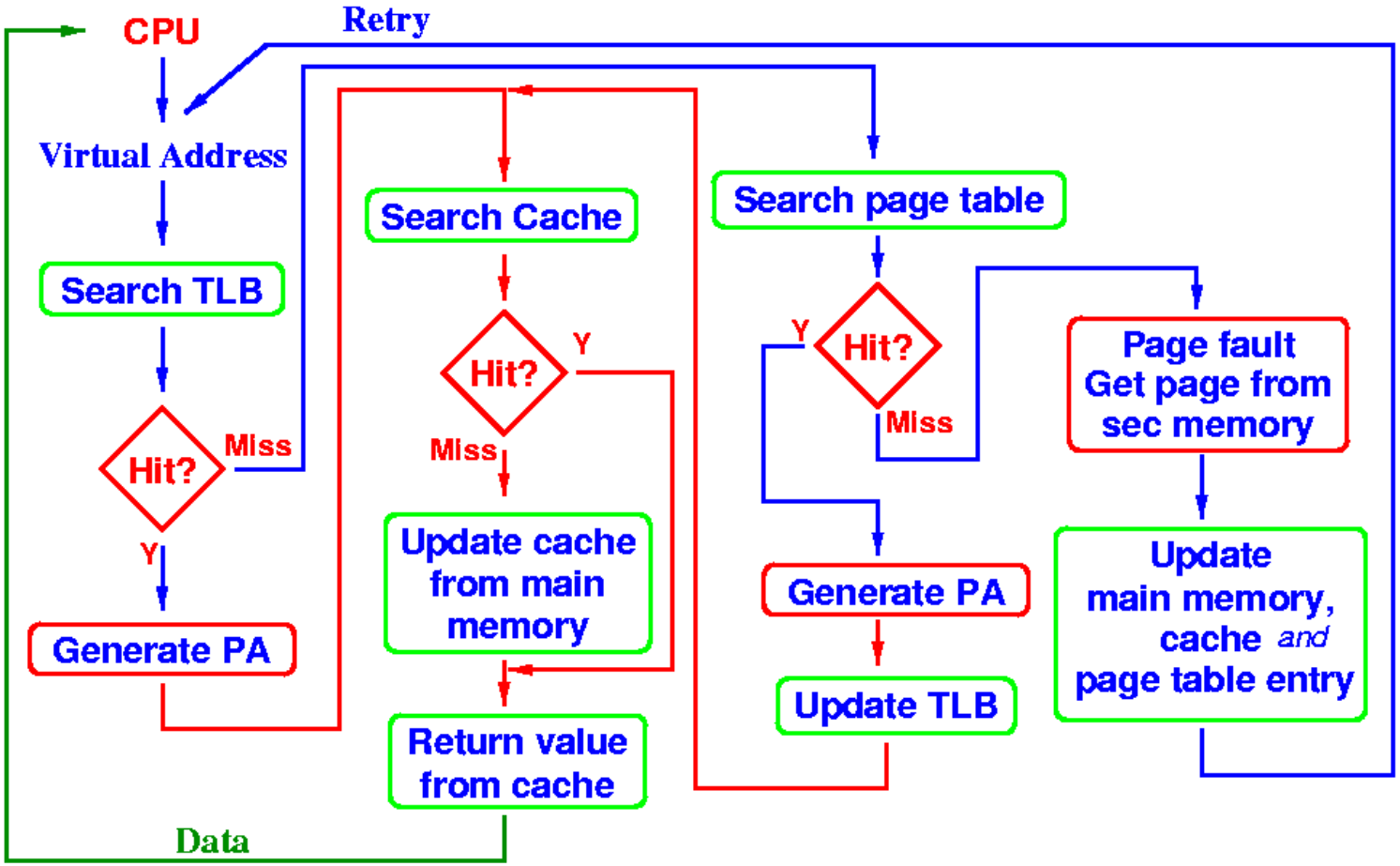
(many memory accesses –
but for only 2% of references to memory!)

TLB – Sequential access

- Luckily, sequential access is fine!
- Example: large (several MByte) matrix of doubles (8 bytes floating point values)
  - 8kbyte pages => 1024 doubles/page
- Sequential access, *eg* sum all values:

```
for(j=0;j<n;j++)
    sum = sum + x[j]
```

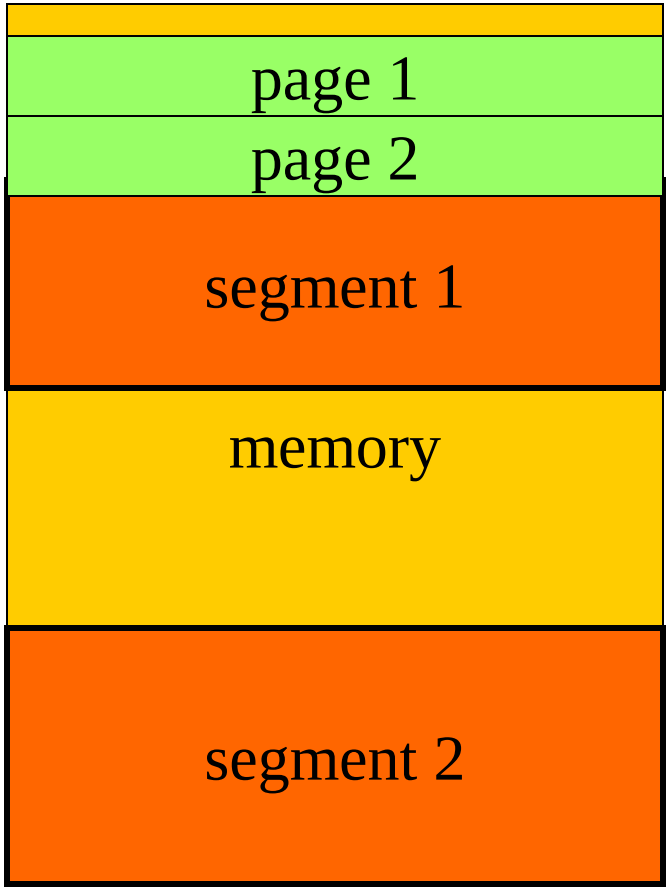
Memory Hierarchy - Operation



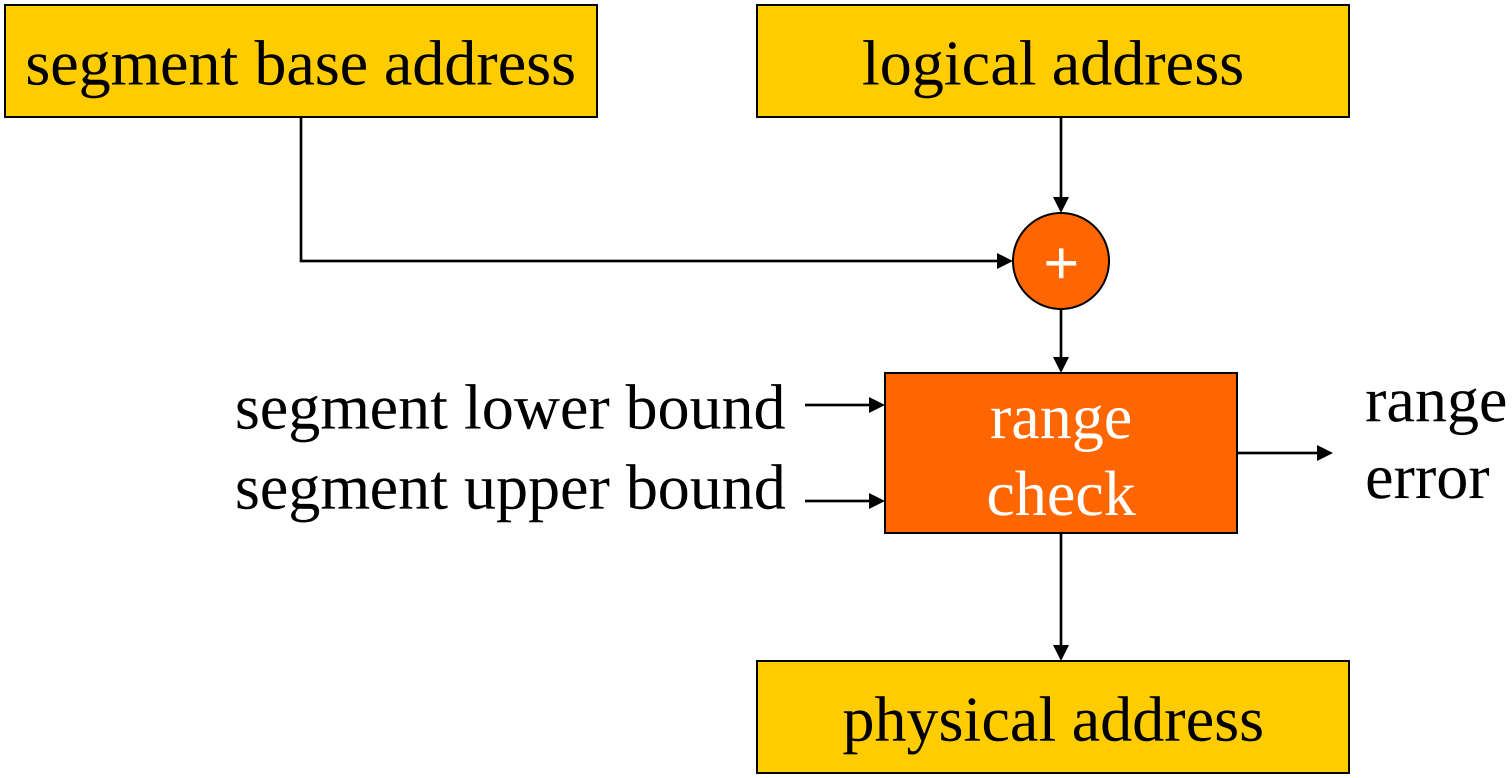
Address translation

- Requires some sort of register/table to allow arbitrary mappings of logical to physical addresses.
- Two basic schemes:
  - segmented;
  - paged.
- Segmentation and paging can be combined (x86).

Segments and pages



Segment address translation



ARM memory management

- Memory region types:
  - section: 1 Mbyte block;
  - large page: 64 kbytes;
  - small page: 4 kbytes.
- An address is marked as section-mapped or page-mapped.
- Two-level translation scheme.

ARM address translation