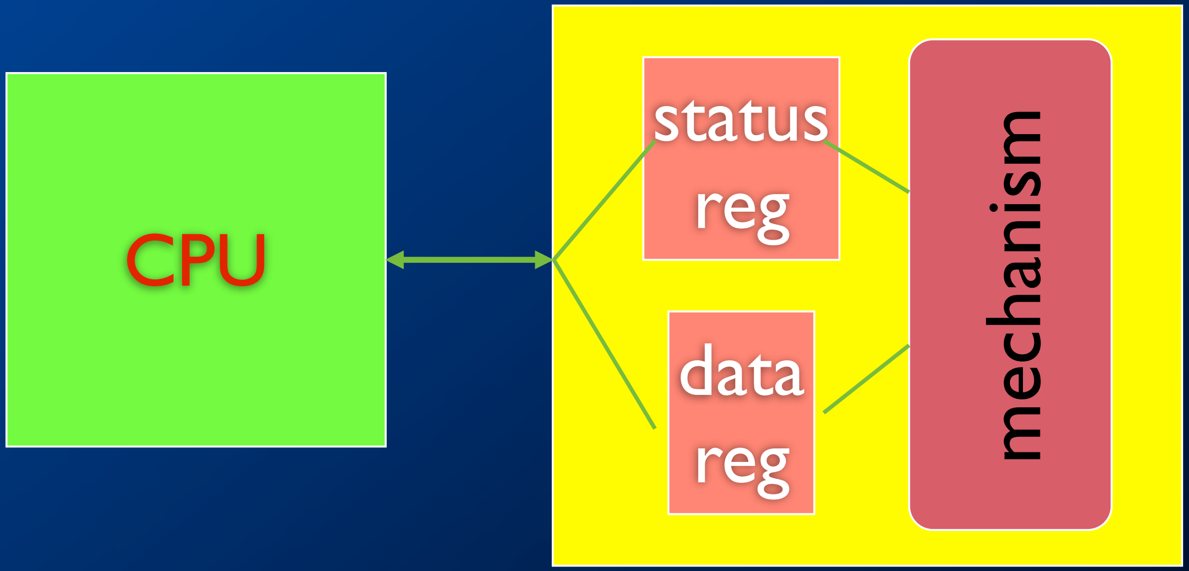
Bus-based computering systems

Agenda

- Input and output
- Busses
- Memory Architectures

I/O devices

- Usually includes some non-digital component.
- Typical digital interface to CPU:

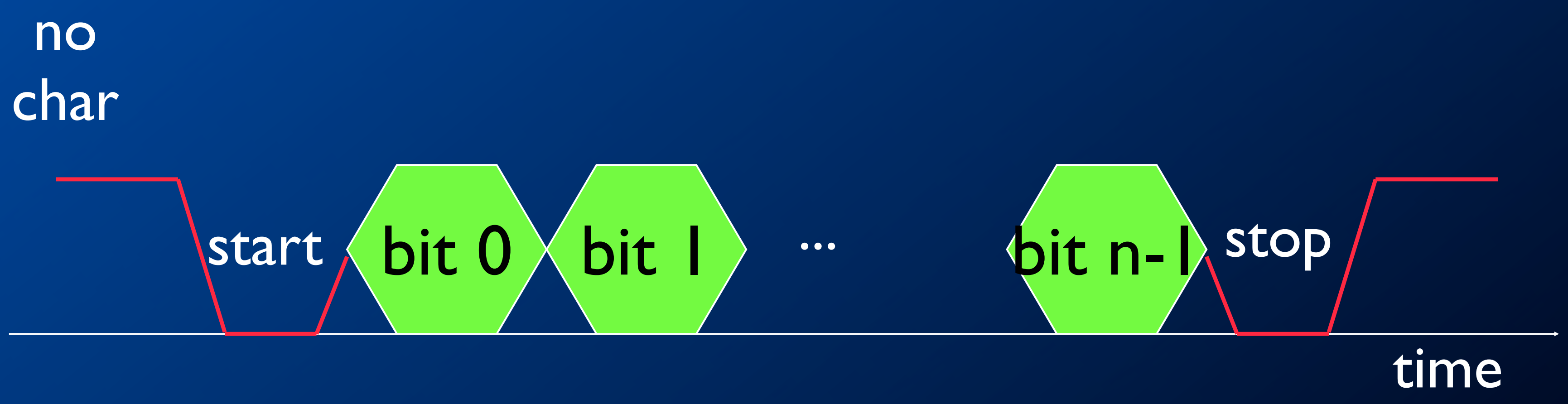


Application: 8251 UART

- Universal asynchronous receiver transmitter (UART) : provides serial communication.
- 8251 functions are integrated into standard PC interface chip.
- Allows many communication parameters to be programmed.

Serial communication

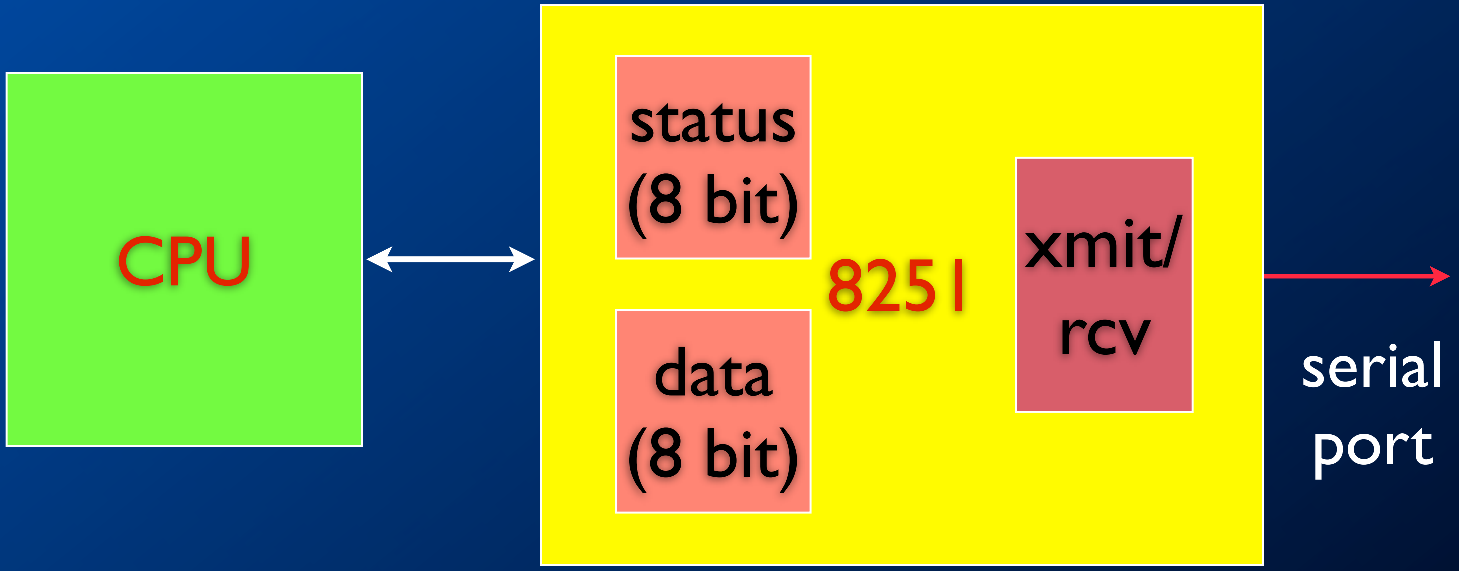
- Characters are transmitted separately:



Serial communication parameters

- Baud (bit) rate.
- Number of bits per character.
- Parity/no parity.
  - Even/odd parity.
- Length of stop bit (1, 1.5, 2 bits).

8251 CPU interface



Programming I/O

- Two types of instructions can support I/O:
  - special-purpose I/O instructions;
  - memory-mapped load/store instructions.
- Intel x86 provides in, out instructions. Most other CPUs use memory-mapped I/O.
- I/O instructions do not preclude memory-mapped I/O.

ARM memory-mapped I/O

- Define location for device:

```
DEV1 EQU 0x1000
```

- Read/write code:

```
LDR r1,#DEV1 ; set up device adrs
LDR r0,[r1] ; read DEV1
LDR r0,#8 ; set up value to write
STR r0,[r1] ; write value to device
```

Peek and poke

- Traditional HLL interfaces:

```
int peek(char *location) {
    return *location; }

void poke(char *location, char newval) {
    (*location) = newval; }
```

Busy/wait output

- Simplest way to program device.
  - Use instructions to test when device is ready.

```
current_char = mystring;
while (*current_char != '\0') {
    poke(OUT_CHAR,*current_char);
    while (peek(OUT_STATUS) != 0);
    current_char++;
}
```

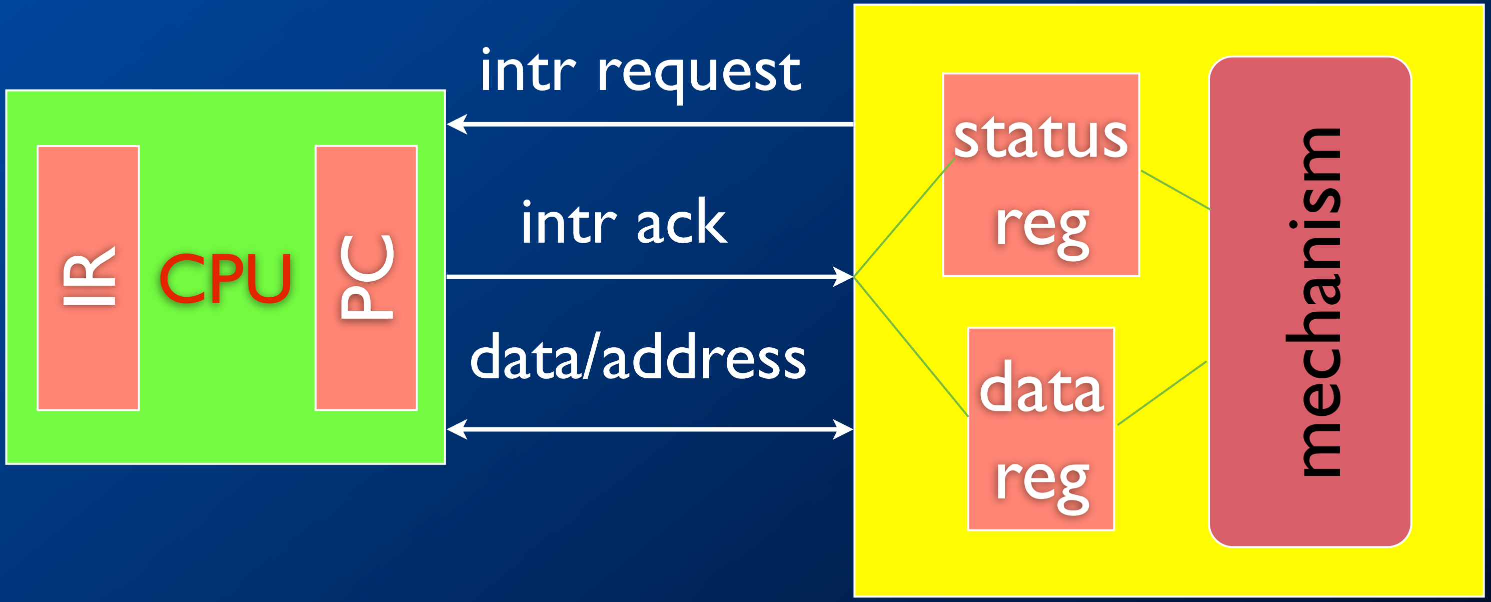
Simultaneous busy/wait input and output

```
while (TRUE) {
    /* read */
    while (peek(IN_STATUS) == 0);
    achar = (char)peek(IN_DATA);
    /* write */
    poke(OUT_DATA,achar);
    poke(OUT_STATUS,1);
    while (peek(OUT_STATUS) != 0);
}
```

Interrupt I/O

- Busy/wait is very inefficient.
  - CPU can’t do other work while testing device.
  - Hard to do simultaneous I/O.
- Interrupts allow a device to change the flow of control in the CPU.
  - Causes subroutine call to handle device.

Interrupt interface



Interrupt behavior

- Based on subroutine call mechanism.
- Interrupt forces next instruction to be a subroutine call to a predetermined location.
    - Return address is saved to resume executing foreground program.

Interrupt physical interface

- CPU and device are connected by CPU bus.
- CPU and device handshake:
  - device asserts interrupt request;
  - CPU asserts interrupt acknowledge when it can handle the interrupt.

Example: character I/O handlers

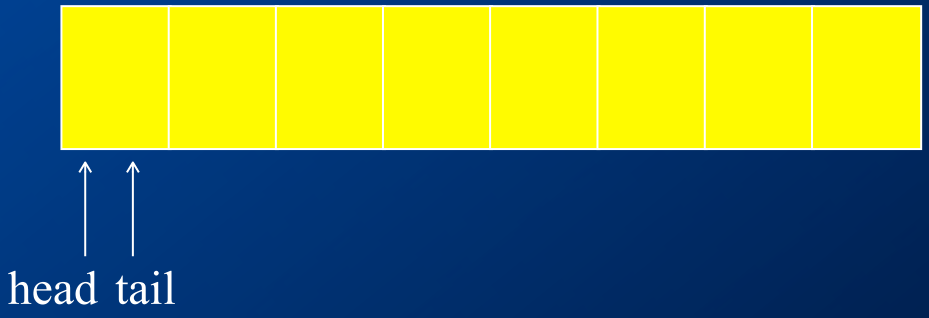
```
void input_handler() {
    achar = peek(IN_DATA);
    gotchar = TRUE;
    poke(IN_STATUS,0);
}
void output_handler() {
}
```

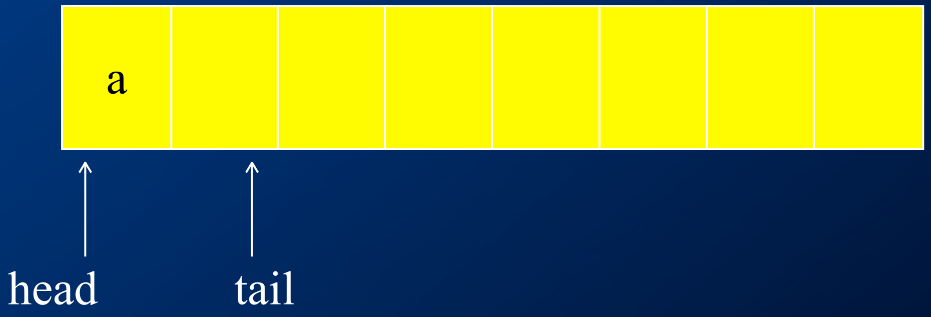
Example: interrupt-driven main program

```
main() {
    while (TRUE) {
        if (gotchar) {
            poke(OUT_DATA,achar);
            poke(OUT_STATUS,1);
            gotchar = FALSE;
        }
    }
    other processing....
}
```

Example: interrupt I/O with buffers

- Queue for characters:



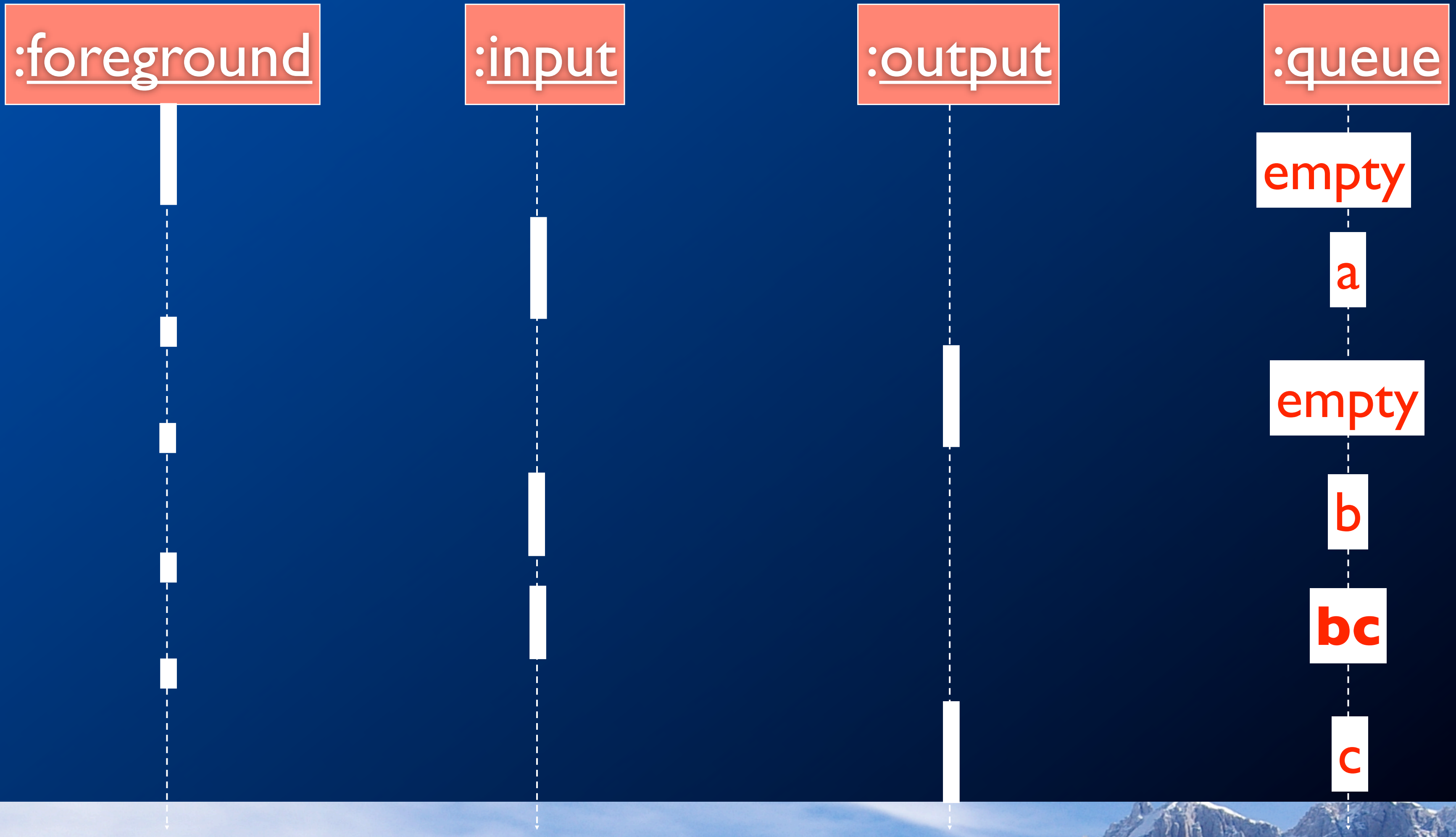


leave one empty slot to allow full buffer to be detected

Buffer-based input handler

```
void input_handler() {
    char achar;
    if (full_buffer()) error = 1;
    else {
        achar = peek(IN_DATA);
        add_char(achar); }
    poke(IN_STATUS,0);
    if (nchars == 1)
    {
        poke(OUT_DATA,remove_char();
         poke(OUT_STATUS,1); }
}
```

I/O sequence diagram



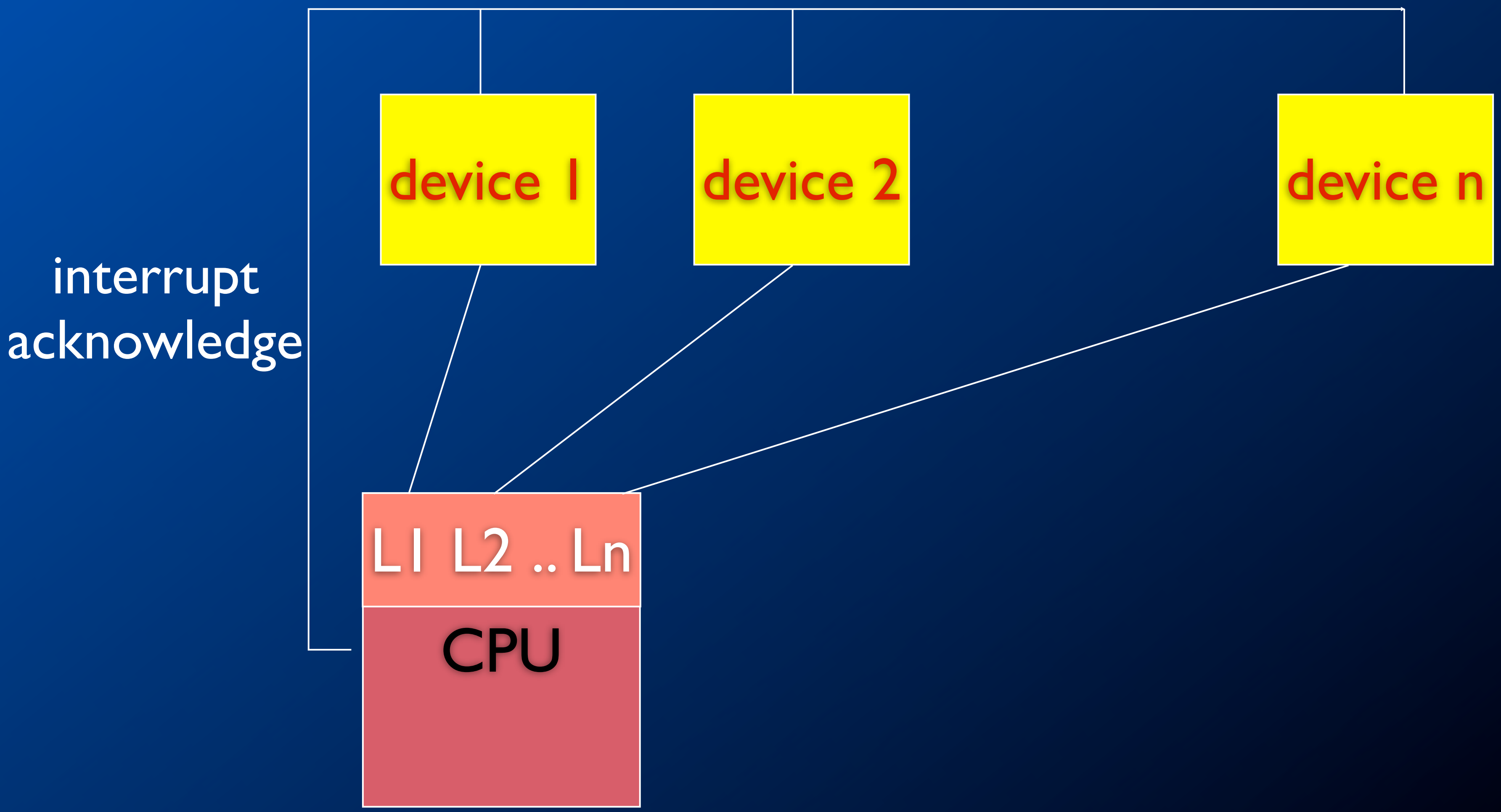
Debugging interrupt code

- What if you forget to change registers?
  - Foreground program can exhibit mysterious bugs.
  - Bugs will be hard to repeat---depend on interrupt timing.

Priorities and vectors

- Two mechanisms allow us to make interrupts more specific:
  - Priorities determine what interrupt gets CPU first.
  - Vectors determine what code is called for each type of interrupt.
- Mechanisms are orthogonal: most CPUs provide both.

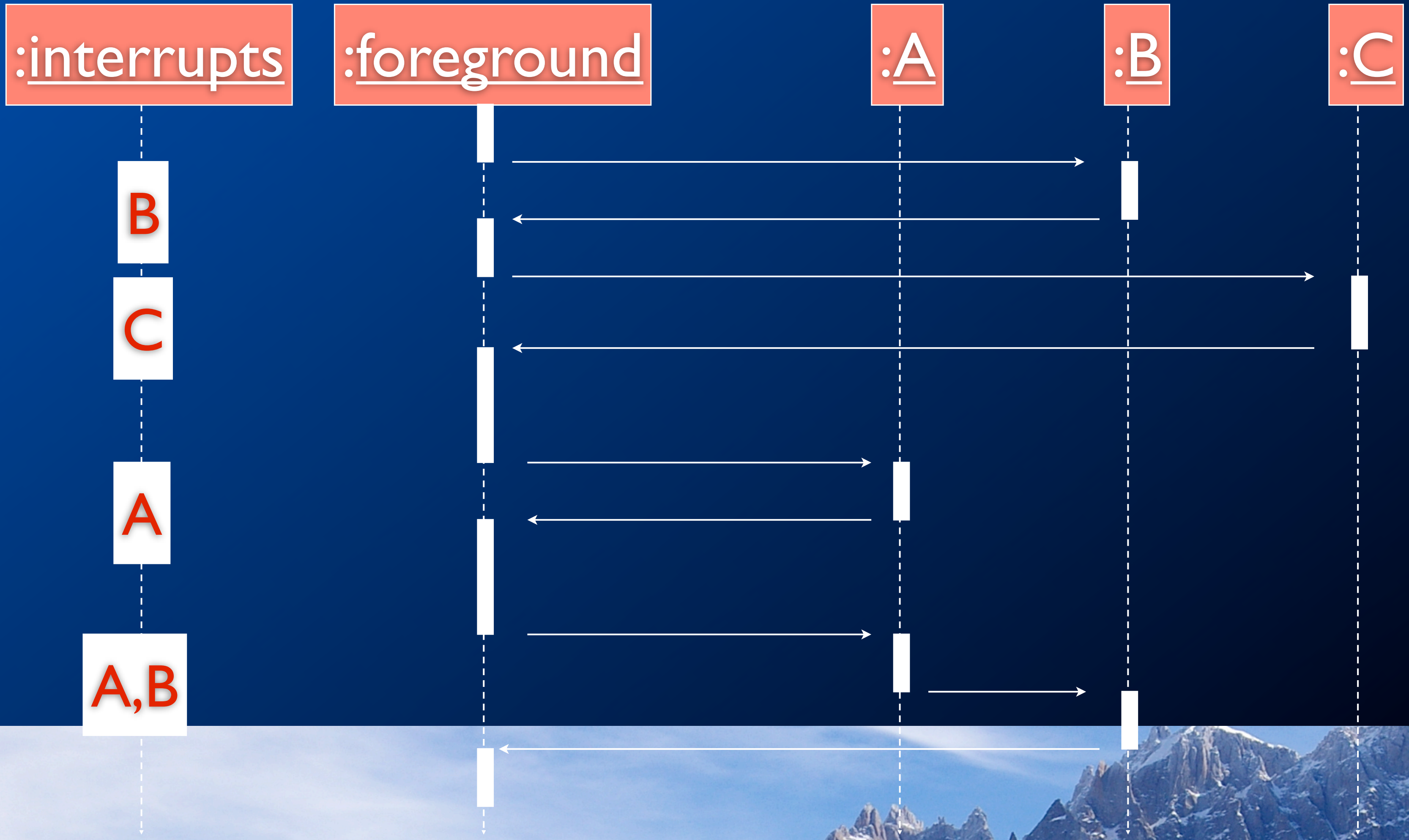
Prioritized interrupts



Interrupt prioritization

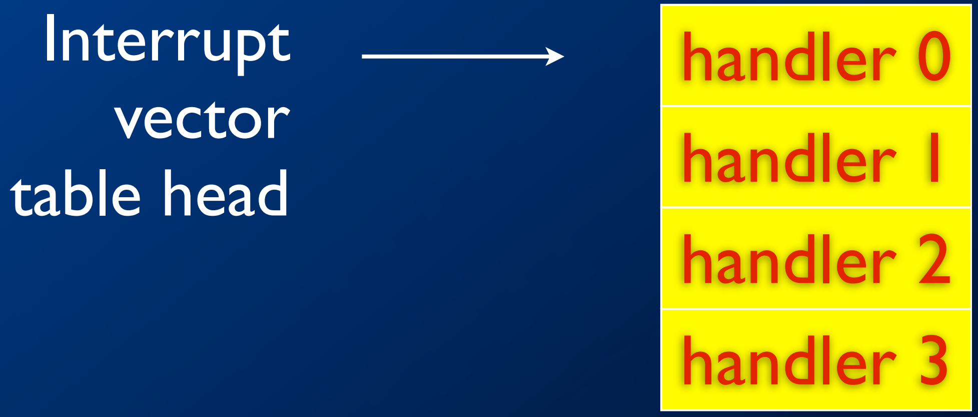
- Masking: interrupt with priority lower than current priority is not recognized until pending interrupt is complete.
- Non-maskable interrupt (NMI): highest-priority, never masked.
  - Often used for power-down.

Example: Prioritized I/O

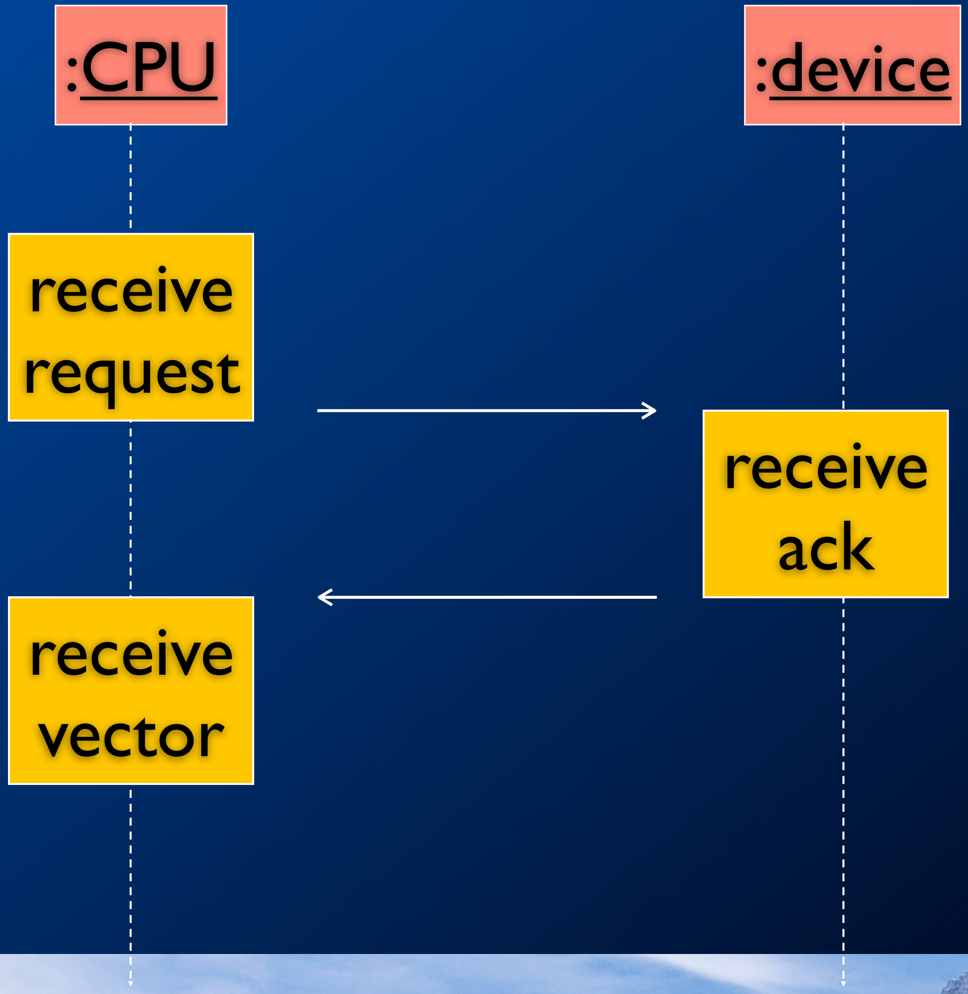


Interrupt vectors

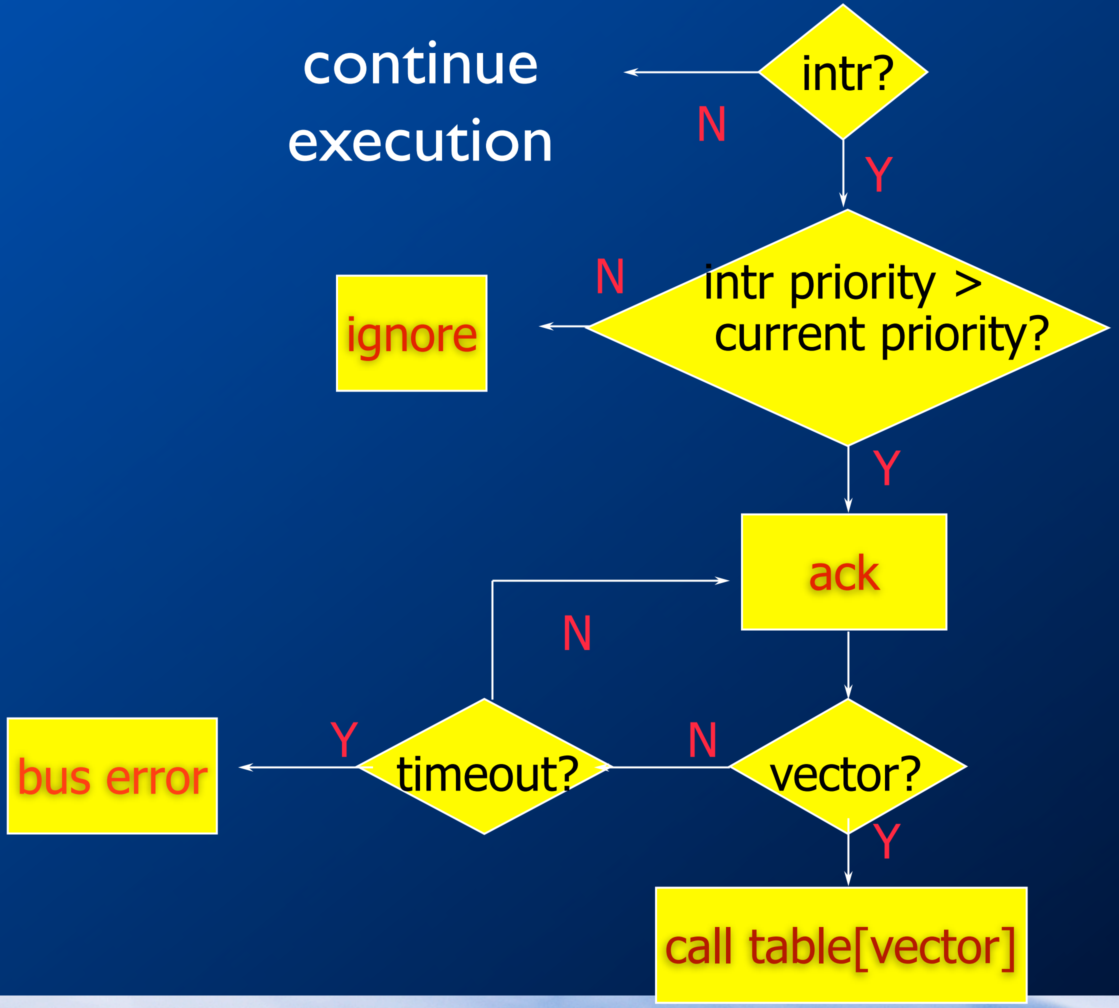
- Allow different devices to be handled by different code.
- Interrupt vector table:



Interrupt vector acquisition



Generic interrupt mechanism



Assume priority selection is handled before this point.

Interrupt sequence

- CPU acknowledges request.
- Device sends vector.
- CPU calls handler.
- Software processes request.
- CPU restores state to foreground program.

Sources of interrupt overhead

- Handler execution time.
- Interrupt mechanism overhead.
- Register save/restore.
- Pipeline-related penalties.
- Cache-related penalties.

ARM interrupts

- ARM7 supports two types of interrupts:
  - Fast interrupt requests (FIQs).
  - Interrupt requests (IRQs).
- Interrupt table starts at location 0.

ARM interrupt procedure

- CPU actions:
  - Save PC. Copy CPSR to SPSR.
  - Force bits in CPSR to record interrupt.
  - Force PC to vector.
- Handler responsibilities:
  - Restore proper PC.
  - Restore CPSR from SPSR.
  - Clear interrupt disable flags.

ARM interrupt latency

- Worst-case latency to respond to interrupt is 27 cycles:
  - Two cycles to synchronize external request.
  - Up to 20 cycles to complete current instruction.
  - Three cycles for data abort.
  - Two cycles to enter interrupt handling state.

Supervisor mode

- May want to provide protective barriers between programs.
  - Avoid memory corruption.
- Need supervisor mode to manage the various programs.

ARM supervisor mode

- Use SWI instruction to enter supervisor mode, similar to subroutine:
  - SWI CODE_1
- Sets PC to 0x08.
- Argument to SWI is passed to supervisor mode code.
- Saves CPSR in SPSR.

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.

Agenda

- Input and output
- Busses
- Memory Architectures

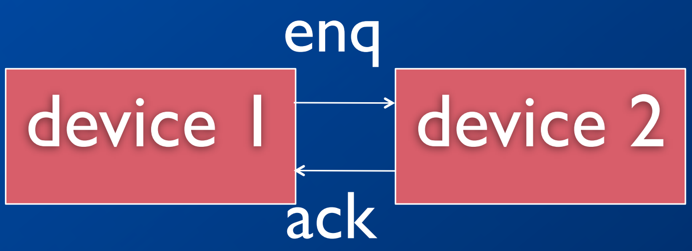
The CPU bus

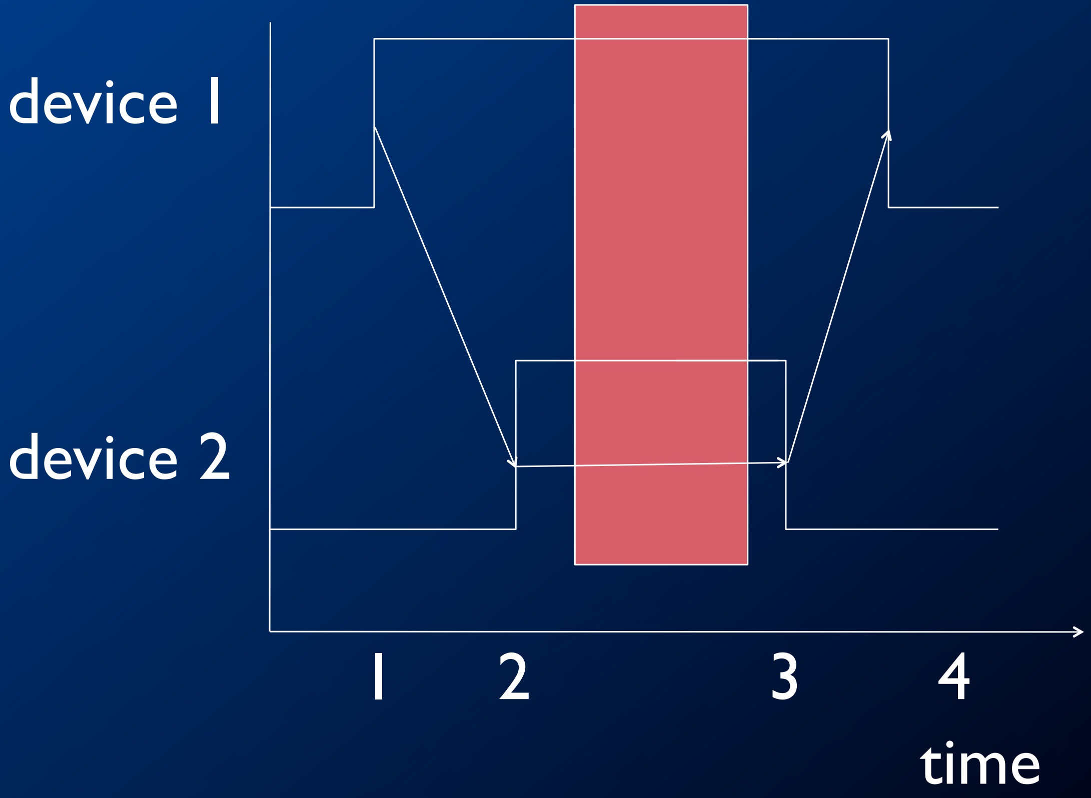
- Bus allows CPU, memory, devices to communicate.
  - Shared communication medium.
- A bus is:
  - A set of wires.
  - A communications protocol.

Bus protocols

- Bus protocol determines how devices communicate.
- Devices on the bus go through sequences of states.
    - Protocols are specified by state machines, one state machine per actor in the protocol.
- May contain asynchronous logic behavior.

Four-cycle handshake



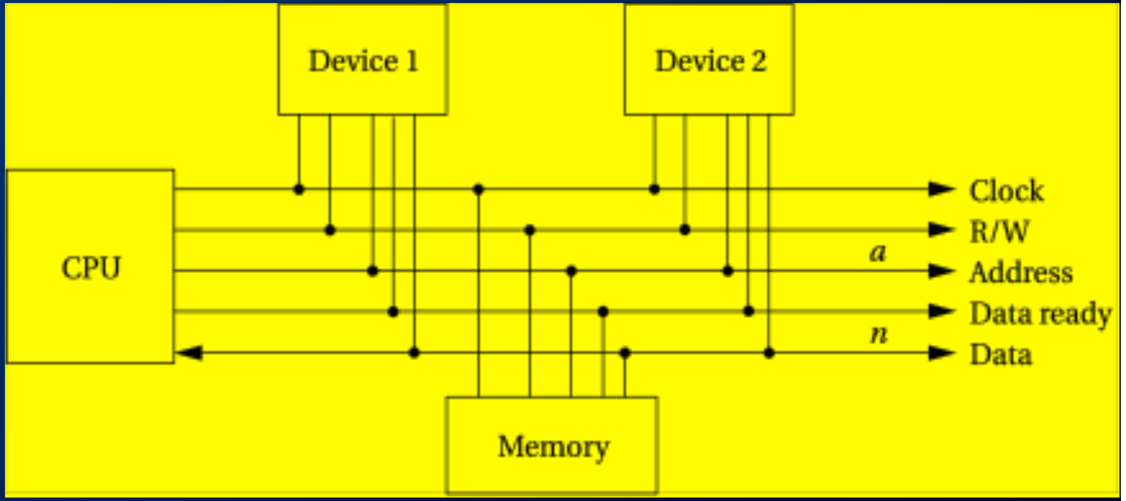


Four-cycle handshake, cont’d.

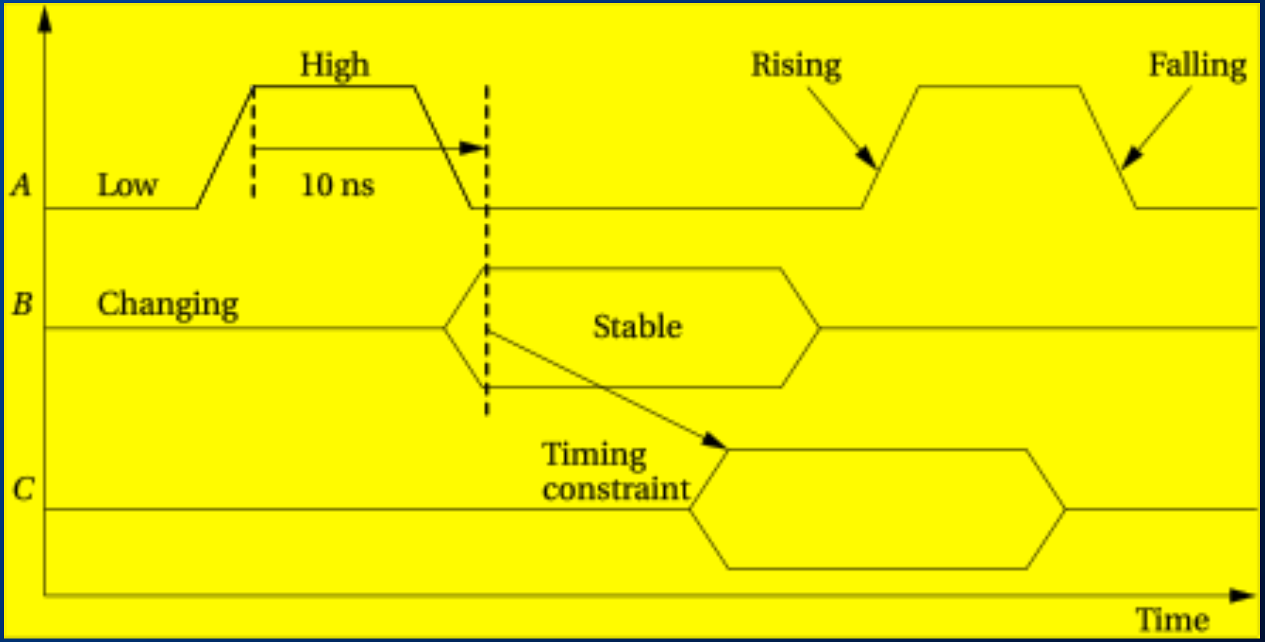
1. Device 1 raises enq.
2. Device 2 responds with ack.
3. Device 2 lowers ack once it has finished.
4. Device 1 lowers enq.

Microprocessor busses

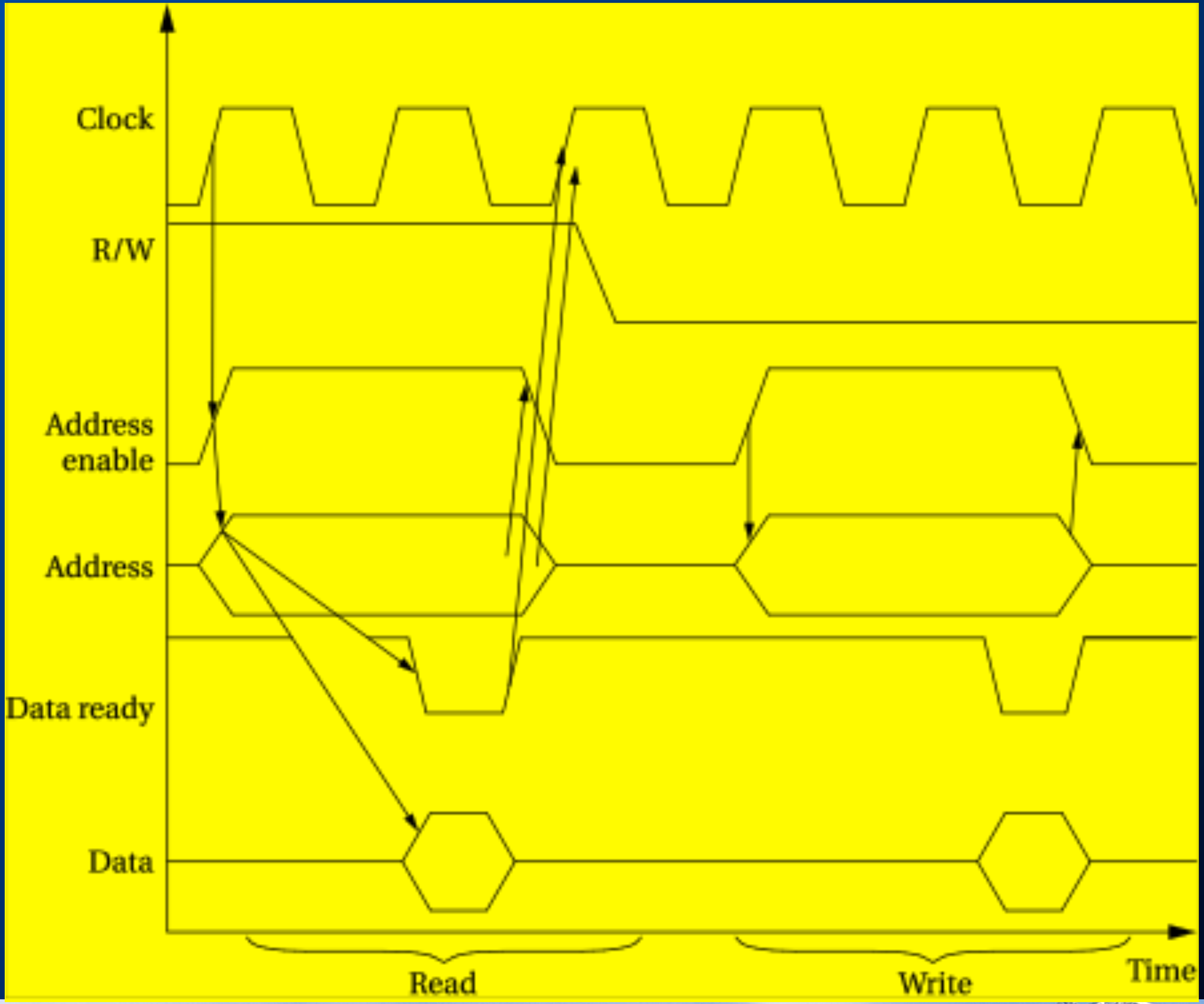
- Clock provides synchronization.
- R/W is true when reading (R/W' is false when reading).
- Address is a-bit bundle of address lines.
- Data is n-bit bundle of data lines.
- Data ready signals when n-bit data is ready.



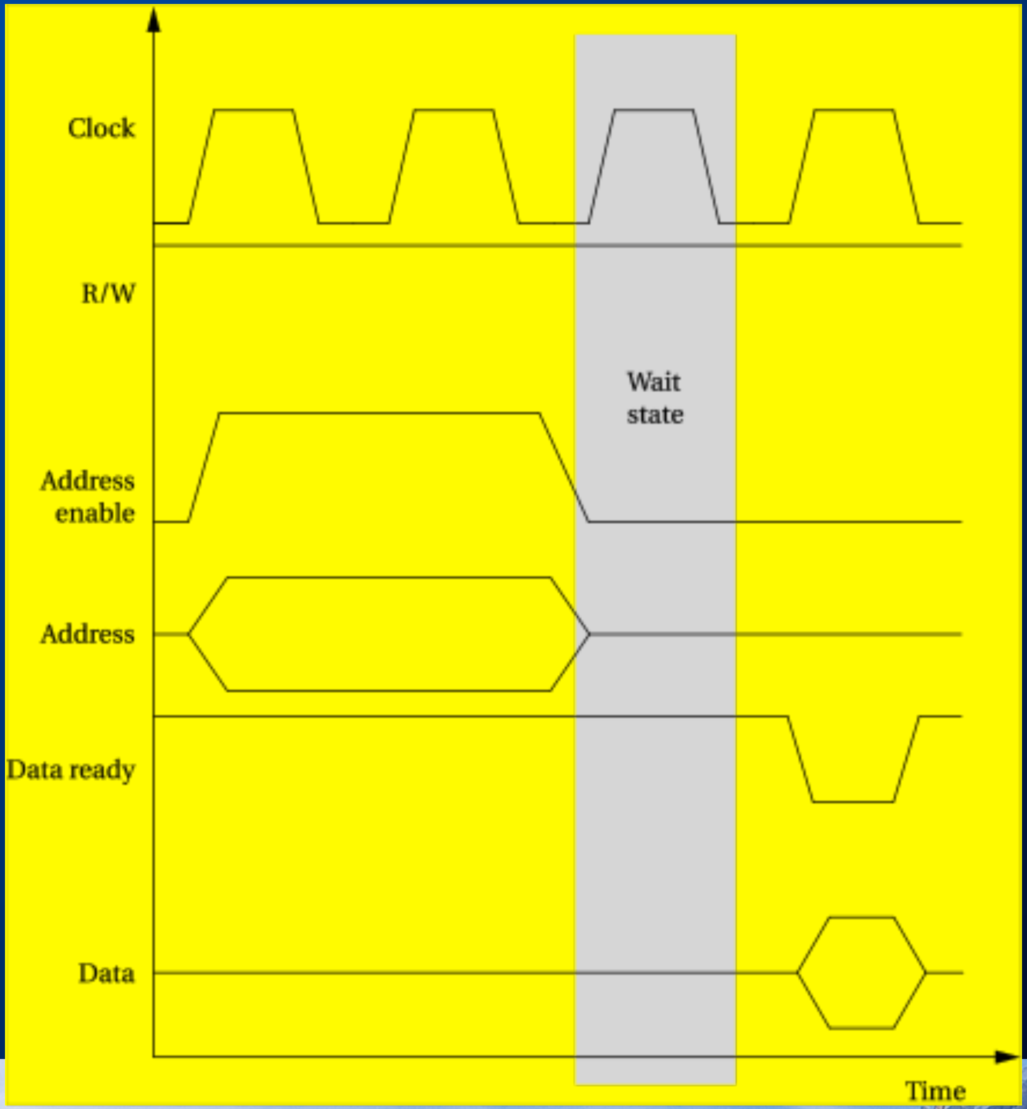
Timing diagrams



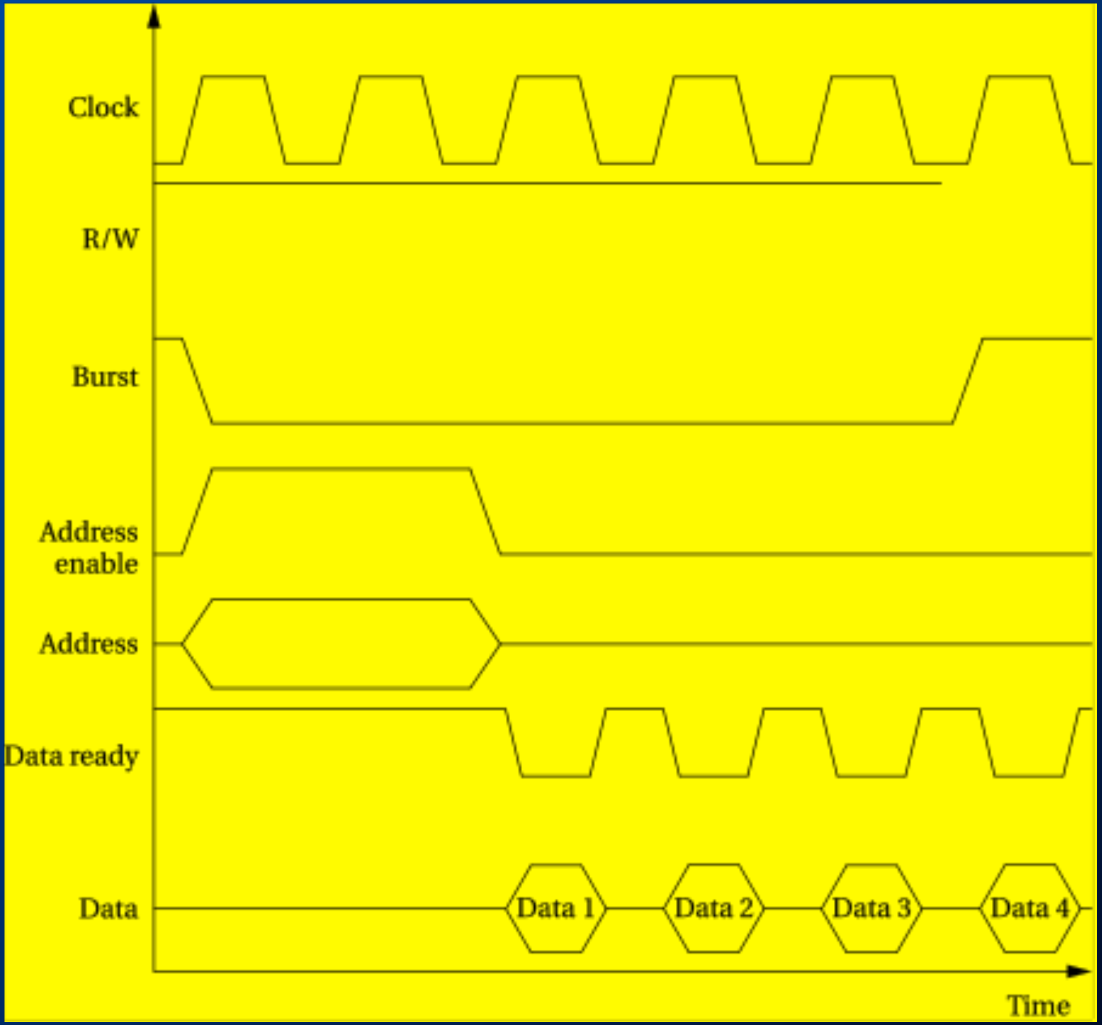
Bus read


Clock
R/W
Address enable
Address
Data ready
Data
Read
Write
Time

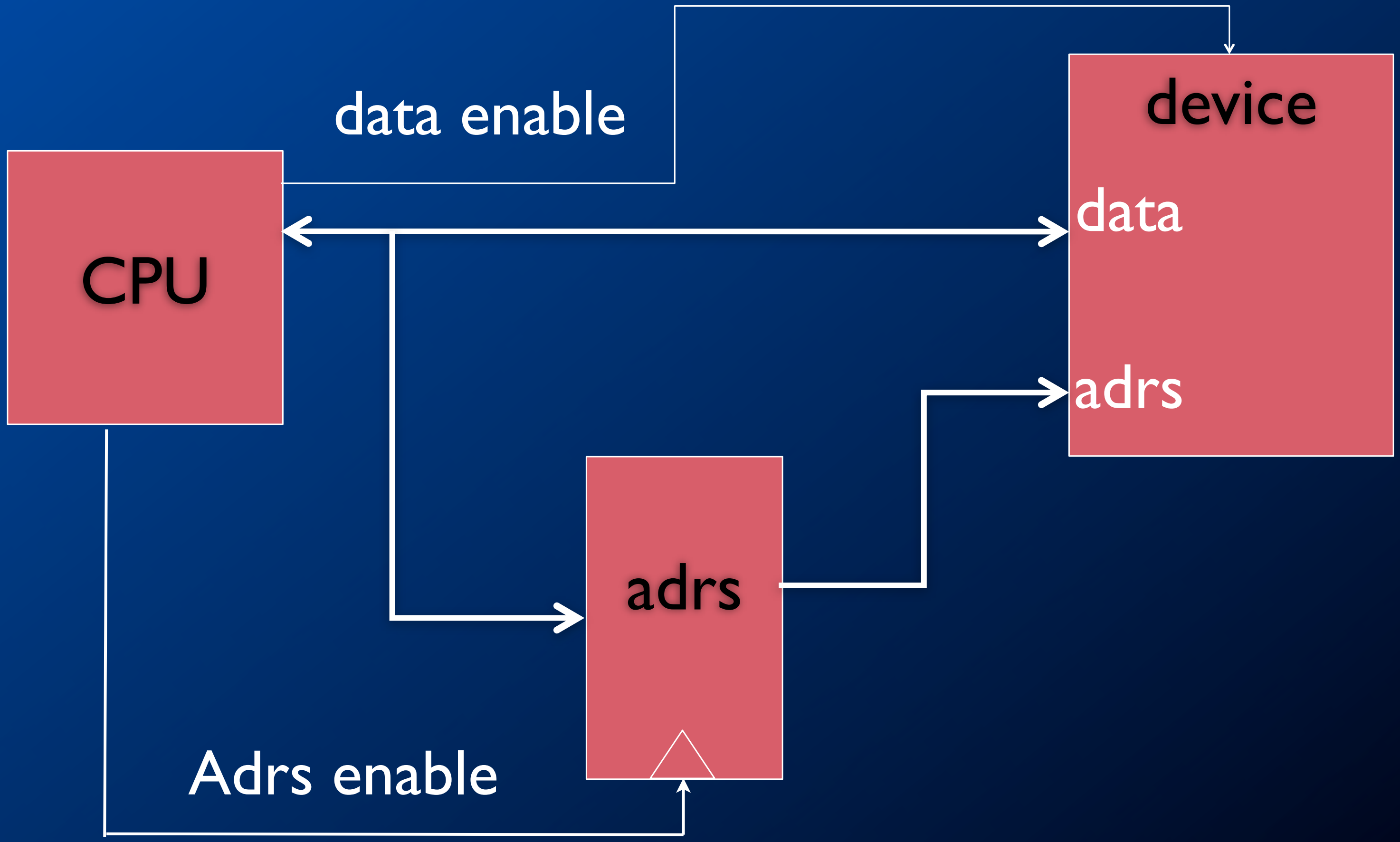
Bus wait state



Bus burst read

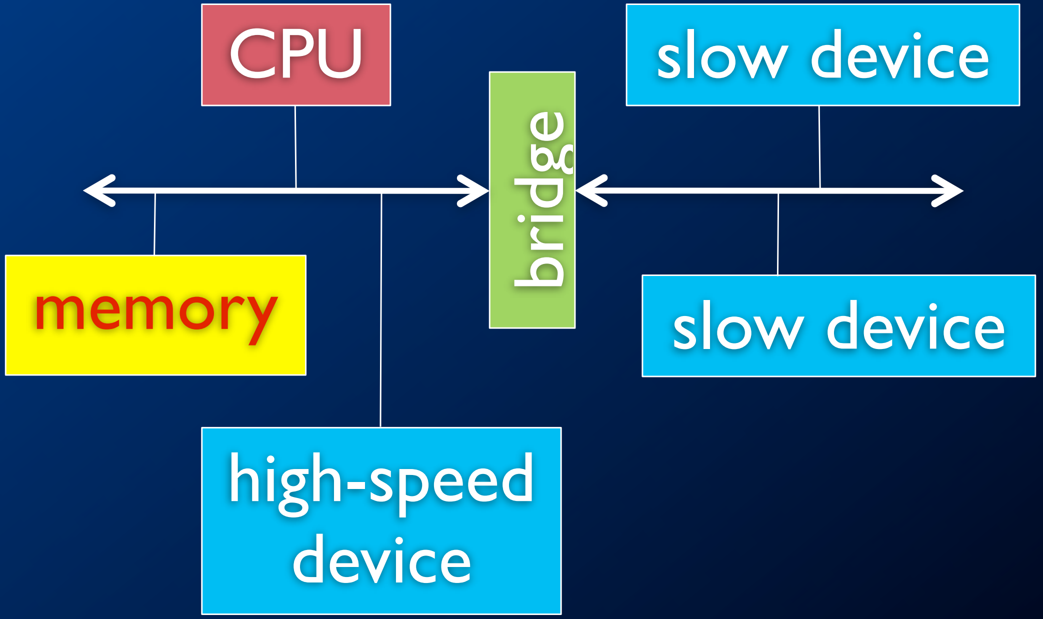


Bus multiplexing

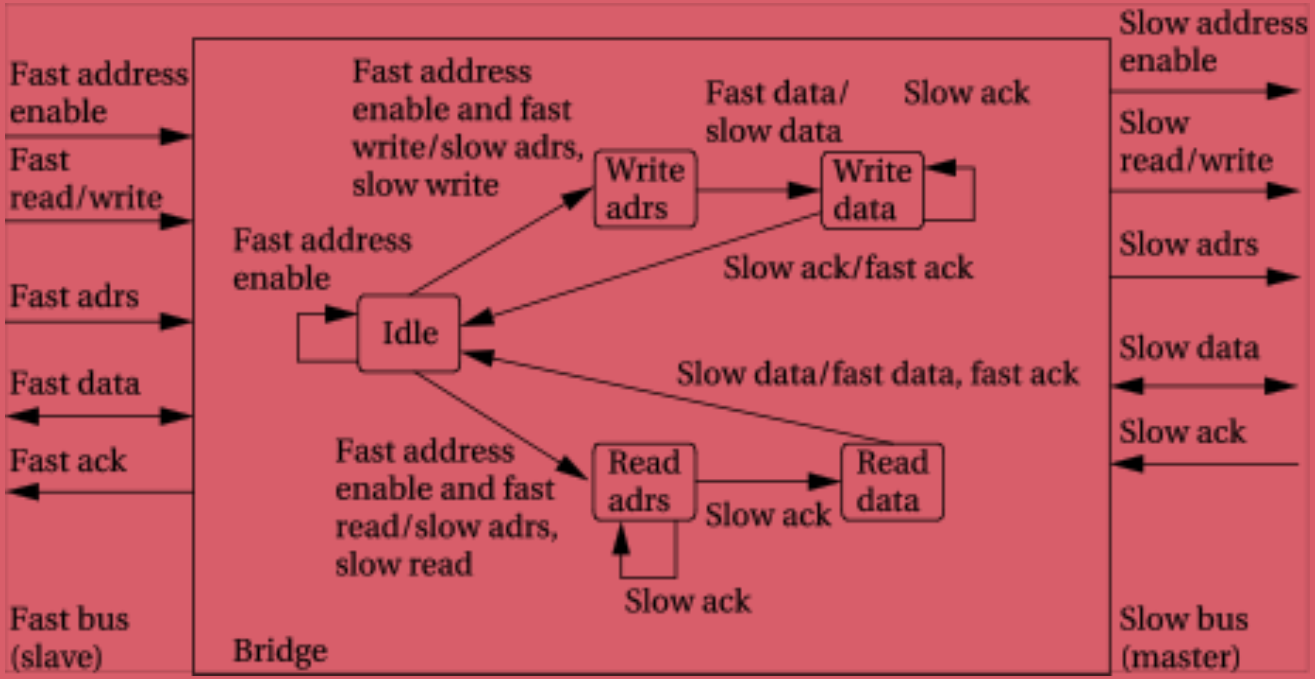


System bus configurations

- Multiple busses allow parallelism:
  - Slow devices on one bus.
  - Fast devices on separate bus.
- A bridge connects two busses.

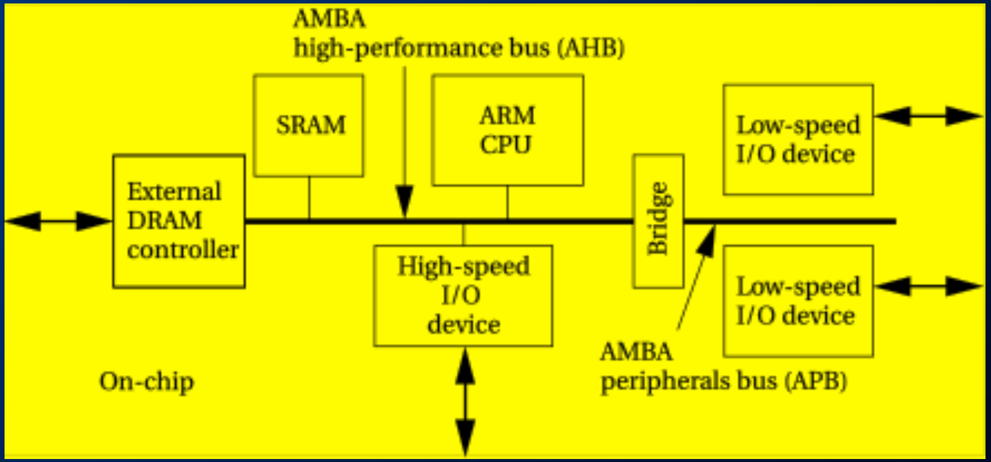


Bridge state diagram



ARM AMBA bus

- Two varieties:
  - AHB is high-performance.
  - APB is lower-speed, lower cost.
- AHB supports pipelining, burst transfers, split transactions, multiple bus masters.
- All devices are slaves on APB.

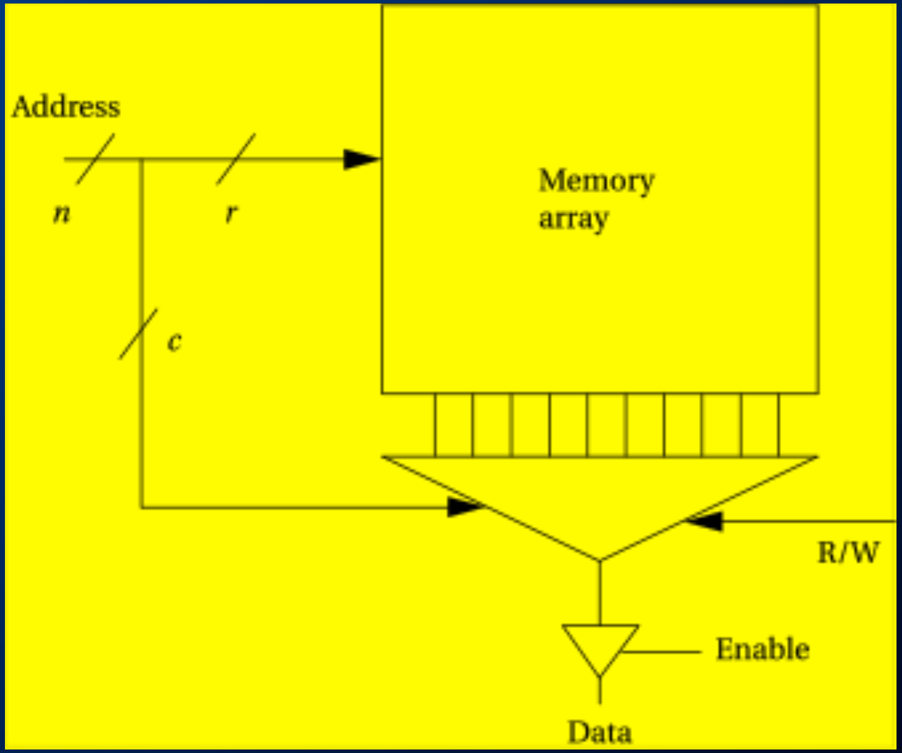


Agenda

- Input and output
- Busses
- Memory Architectures

Memory components

- Several different types of memory:
  - DRAM.
  - SRAM.
  - Flash.
- Each type of memory comes in varying:
  - Capacities.
  - Widths.



Random-access memory

- Dynamic RAM is dense, requires refresh.
    - Synchronous DRAM is dominant type.
    - SDRAM uses clock to improve performance, pipeline memory accesses.
- Static RAM is faster, less dense, consumes more power.

Read-only memory

- ROM may be programmed at factory.
- Flash is dominant form of field-programmable ROM.
    - Electrically erasable, must be block erased.
    - Random access, but write/erase is much slower than read.
    - NOR flash is more flexible.
    - NAND flash is more dense.

Flash memory

- Non-volatile memory.
  - Flash can be programmed in-circuit.
- Random access for read.
- To write:
  - Erase a block to 1.
  - Write bits to 0.

Flash writing

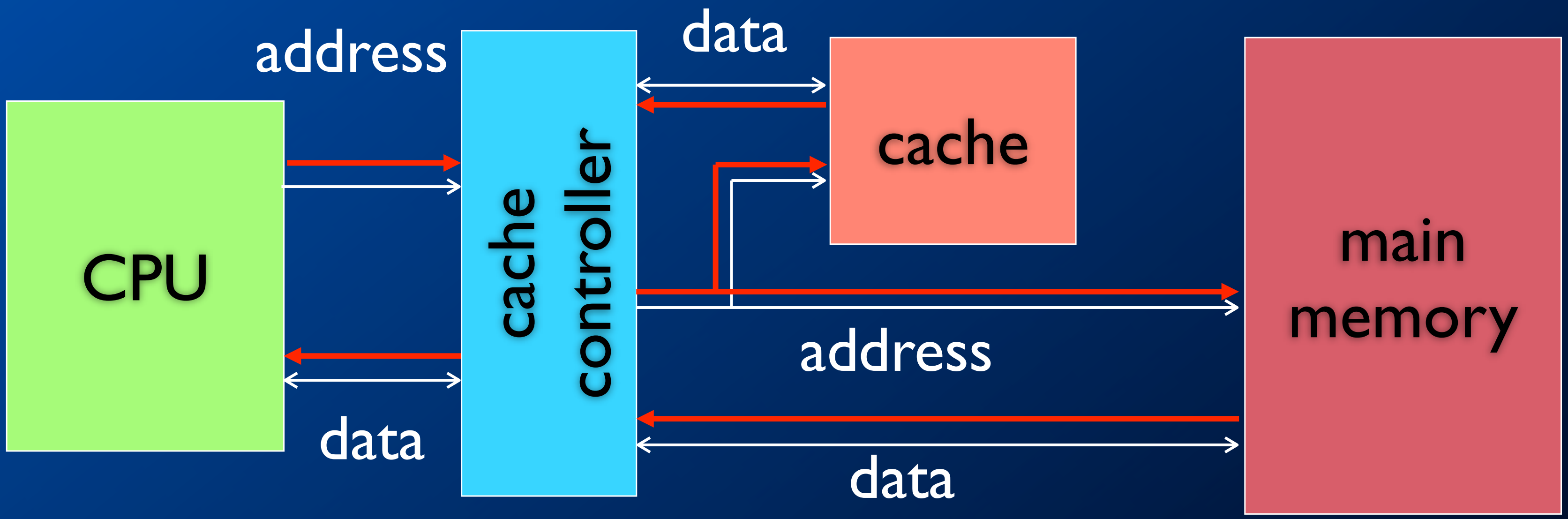
- Write is much slower than read.
  - 1.6 µs write, 70 ns read.
- Blocks are large (approx. 1 Mb).
- Writing causes wear that eventually destroys the device.
  - Modern lifetime approx. 1 million writes.

Types of flash

- NOR:
  - Word-accessible read.
  - Erase by blocks.
- NAND:
  - Read by pages (512-4K bytes).
  - Erase by blocks.
- NAND is cheaper, has faster erase, sequential access times.

	NOR	NAND
写入/擦除一个块的操作时间	1～5s	2～4ms
读性能	1200～1500KB	600～800KB
写性能	<80KB	200～400KB
接口/总线	SRAM接口/独立的地址数据总线	8位地址/数据/控制总线，I/O接口复杂
读取模式	随机读取	串行地存取数据
成本	较高	较低，单元尺寸约为NOR的一半，生产过程简单，同样大小的芯片可以做更大的容量
容量及应用场合	1～64MB，主要用于存储代码	8MB～4GB，主要用于存储数据
擦写次数(耐用性)	约10万次	约100万次
位交换(bit位反转)	少	较多，关键性数据需要错误探测/错误更正(EDC/ECC)算法
坏块处理	无，因为坏块故障率少	随机分布，无法修正

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.

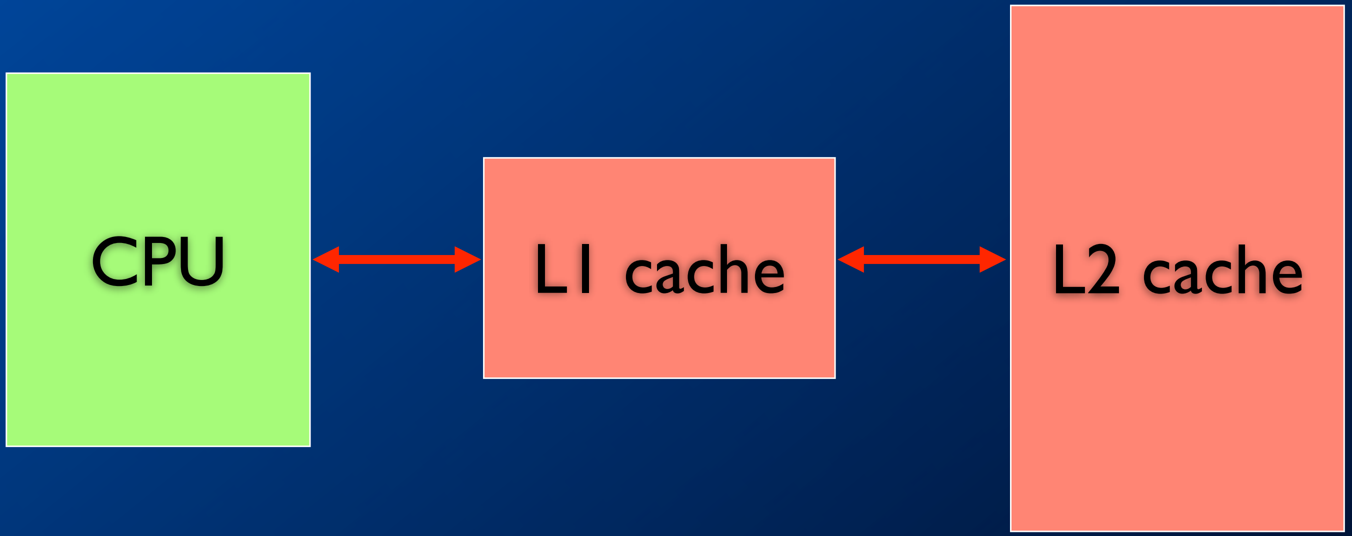
Types of misses

- Compulsory (cold): location has never been accessed.
- Capacity: working set is too large.
- Conflict: 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.
- Average memory access time:
  - $t_{av} = h_1 t_{L1} + h_2 t_{L2} + (1 - h_2 - h_1) t_{main}$

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).

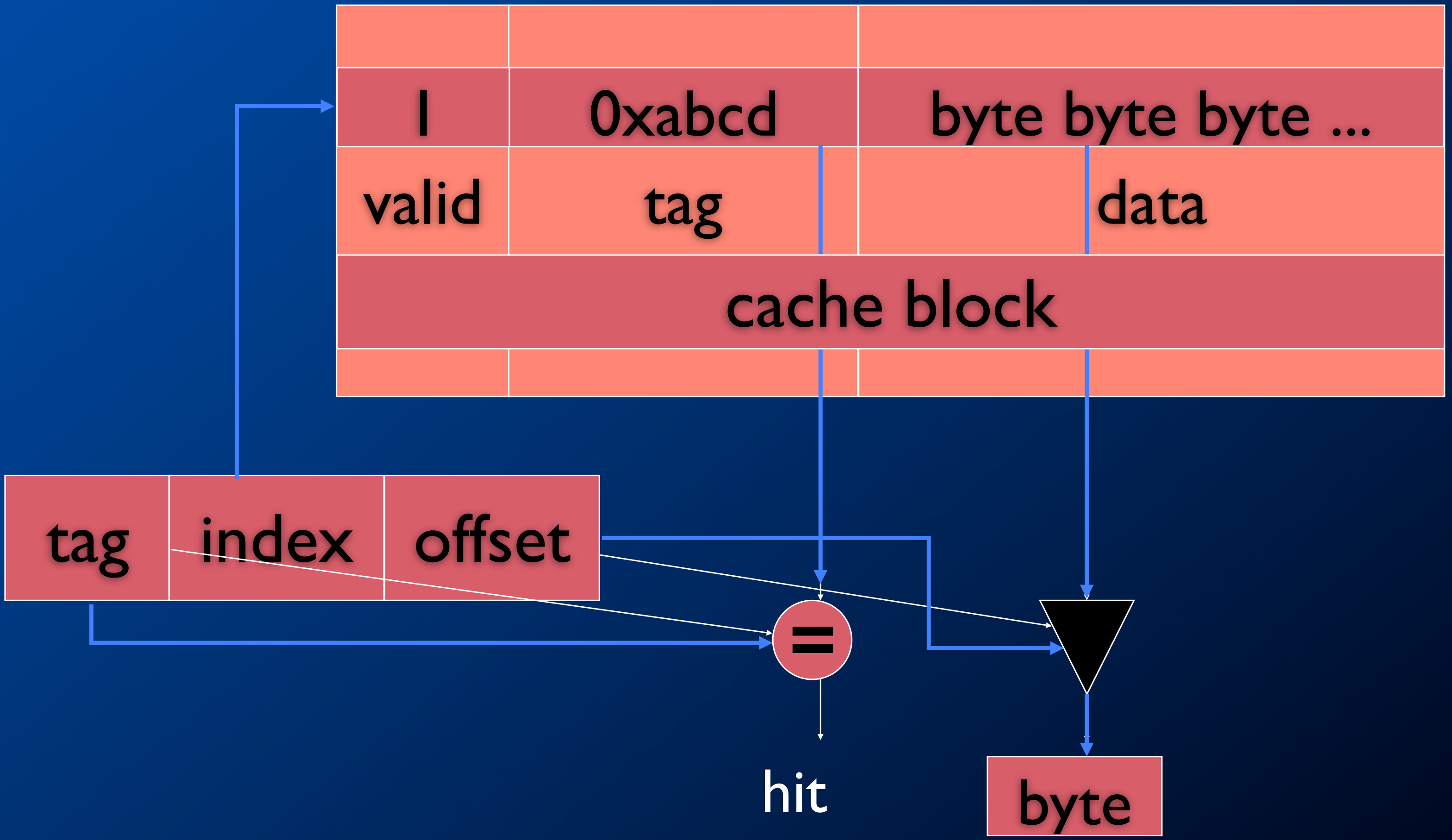
Cache organizations

- 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.

Direct-mapped cache



Write operations

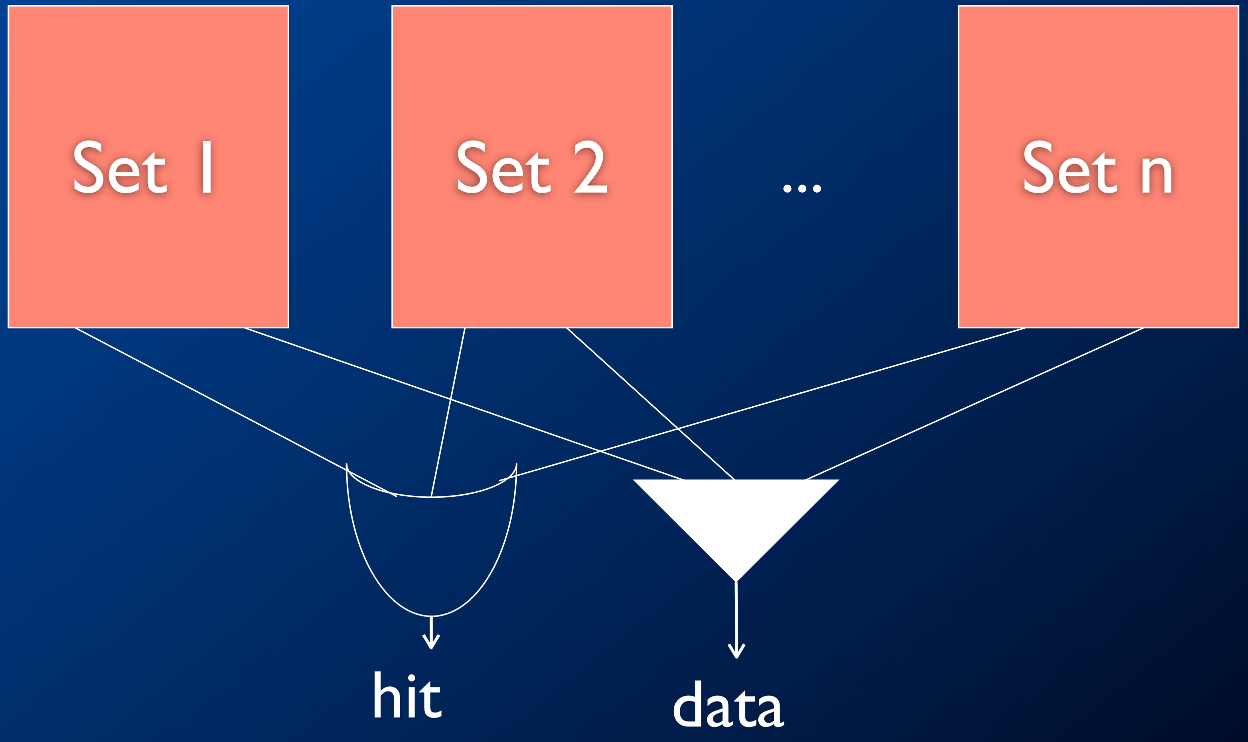
- Write-through: immediately copy write to main memory.
- Write-back: write to main memory only when location is removed from cache.

Direct-mapped cache locations

- Many locations map onto the same cache block.
- Conflict misses are easy to generate:
  - Array a[] uses locations 0, 1, 2, …
  - Array b[] uses locations 1024, 1025, 1026, …
  - Operation a[i] + b[i] generates conflict misses.

Set-associative cache

- A set of direct-mapped caches:



Example: direct-mapped vs. set-associative

address	data
000	0101
001	1111
010	0000
011	0110
100	1000
101	0001
110	1010
111	0100

Direct-mapped cache behavior

- After 001 access:

block	tag	data
00	-	-
01	0	1111
10	-	-
11	-	-

- After 010 access:

block	tag	data
00	-	-
01	0	1111
10	0	0000
11	-	-

Direct-mapped cache behavior, cont'd.

- After 011 access:

block	tag	data
00	-	-
01	0	1111
10	0	0000
11	0	0110

- After 100 access:

block	tag	data
00	1	1000
01	0	1111
10	0	0000
11	0	0110

Direct-mapped cache behavior, cont'd.

- After 101 access:

block	tag	data
00	1	1000
01	1	0001
10	0	0000
11	0	0110

- After 111 access:

block	tag	data
00	1	1000
01	1	0001
10	0	0000
11	1	0100

2-way set-associtive cache behavior

- Final state of cache (twice as big as direct-mapped):

Block	Way 0 tag	Way 0 data	Way 1 tag	Way 1 data
00	1	1000	-	-
01	0	1111	1	0001
10	0	0000	-	-
11	0	0110	1	0100

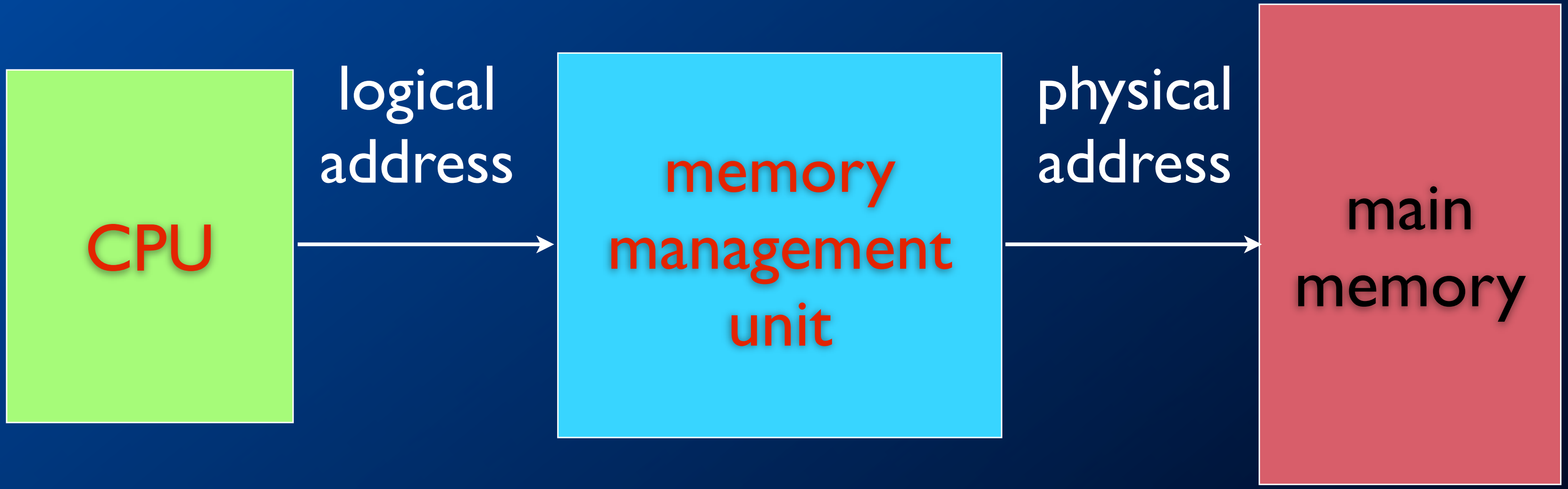
2-way set-associative cache behavior

- Final state of cache (same size as direct-mapped):

Block	Way 0 tag	Way 0 data	Way 1 tag	Way 1 data
0	01	0000	10	1000
1	10	0001	11	0100

Memory management units

- Memory management unit (MMU) translates addresses:



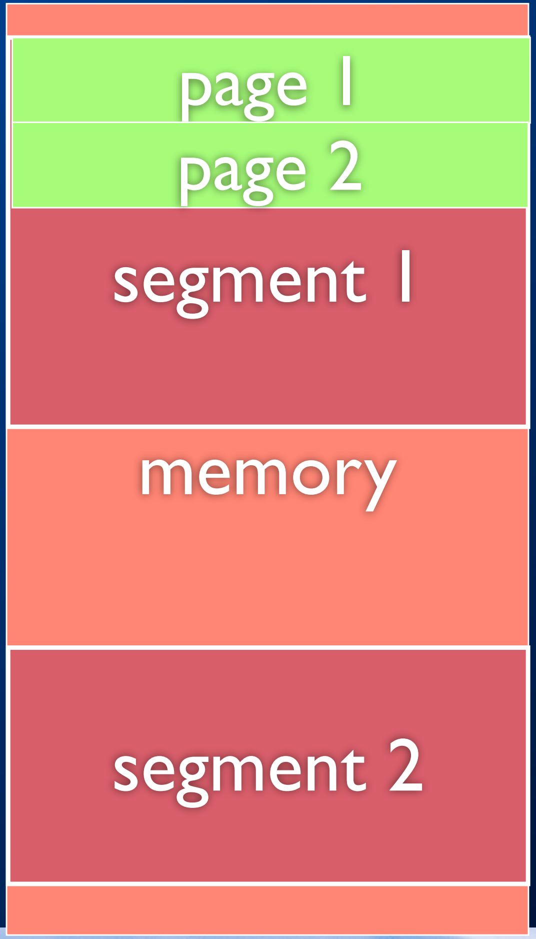
Memory management tasks

- 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.

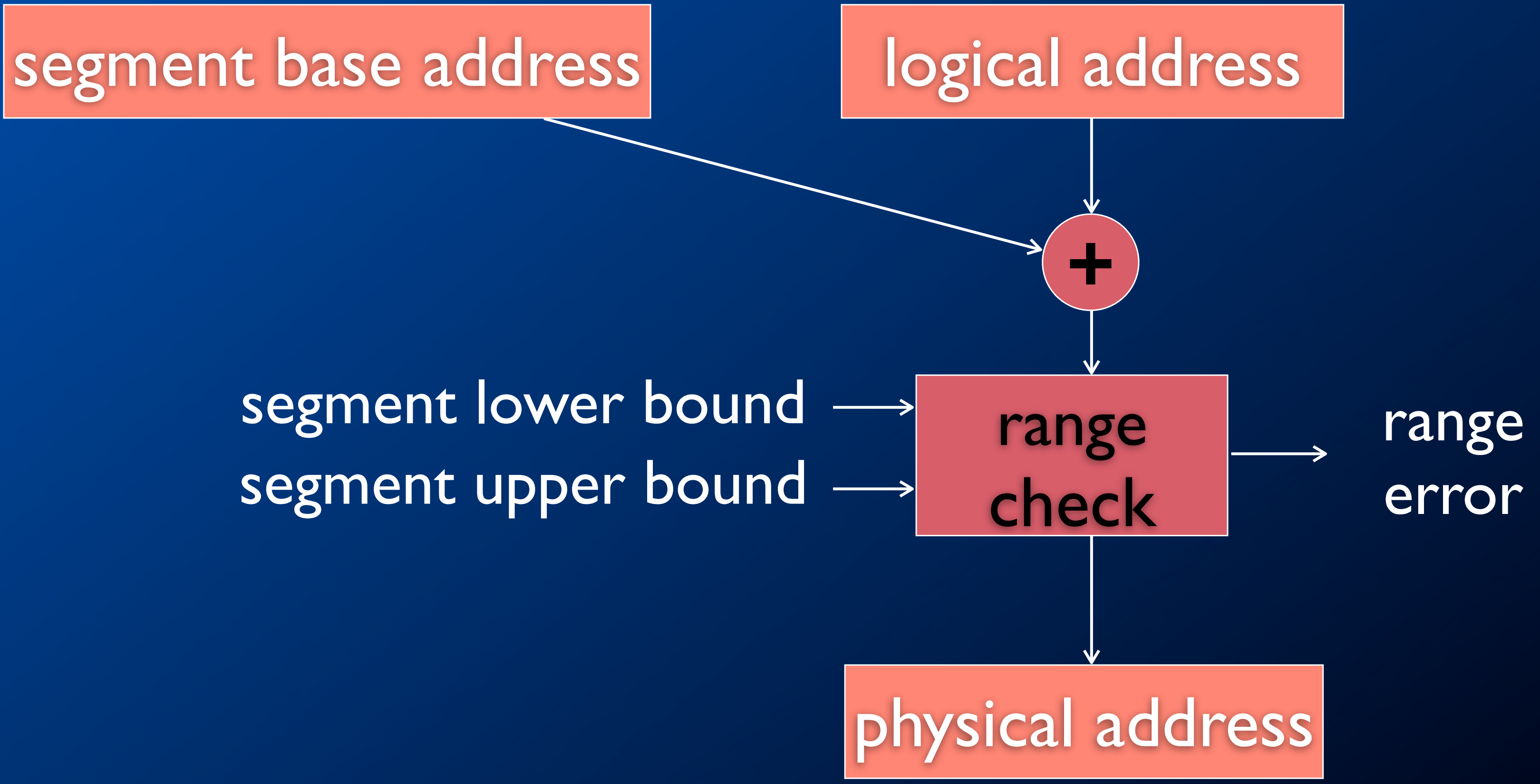
Address translation

- 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).

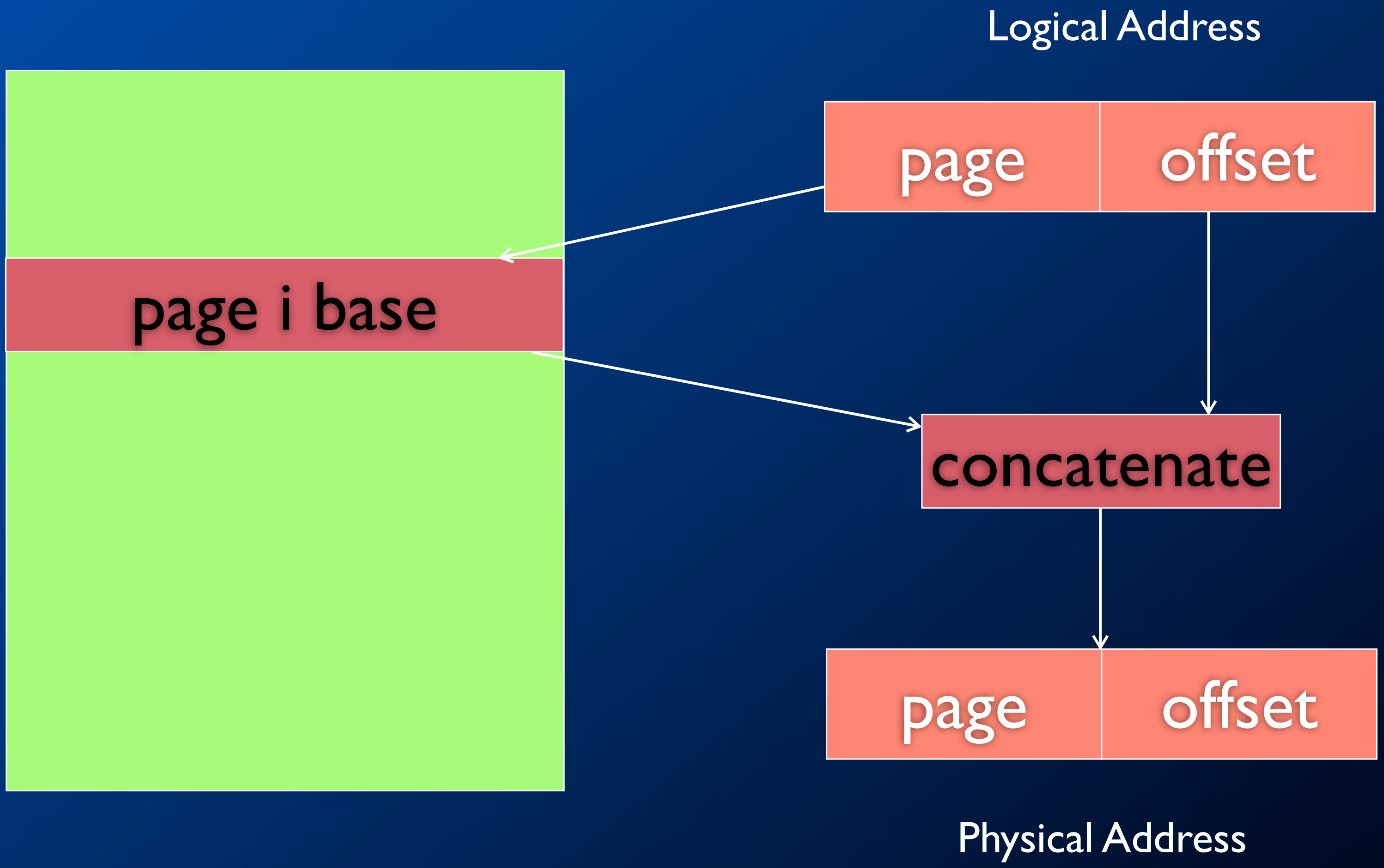
Segments and pages



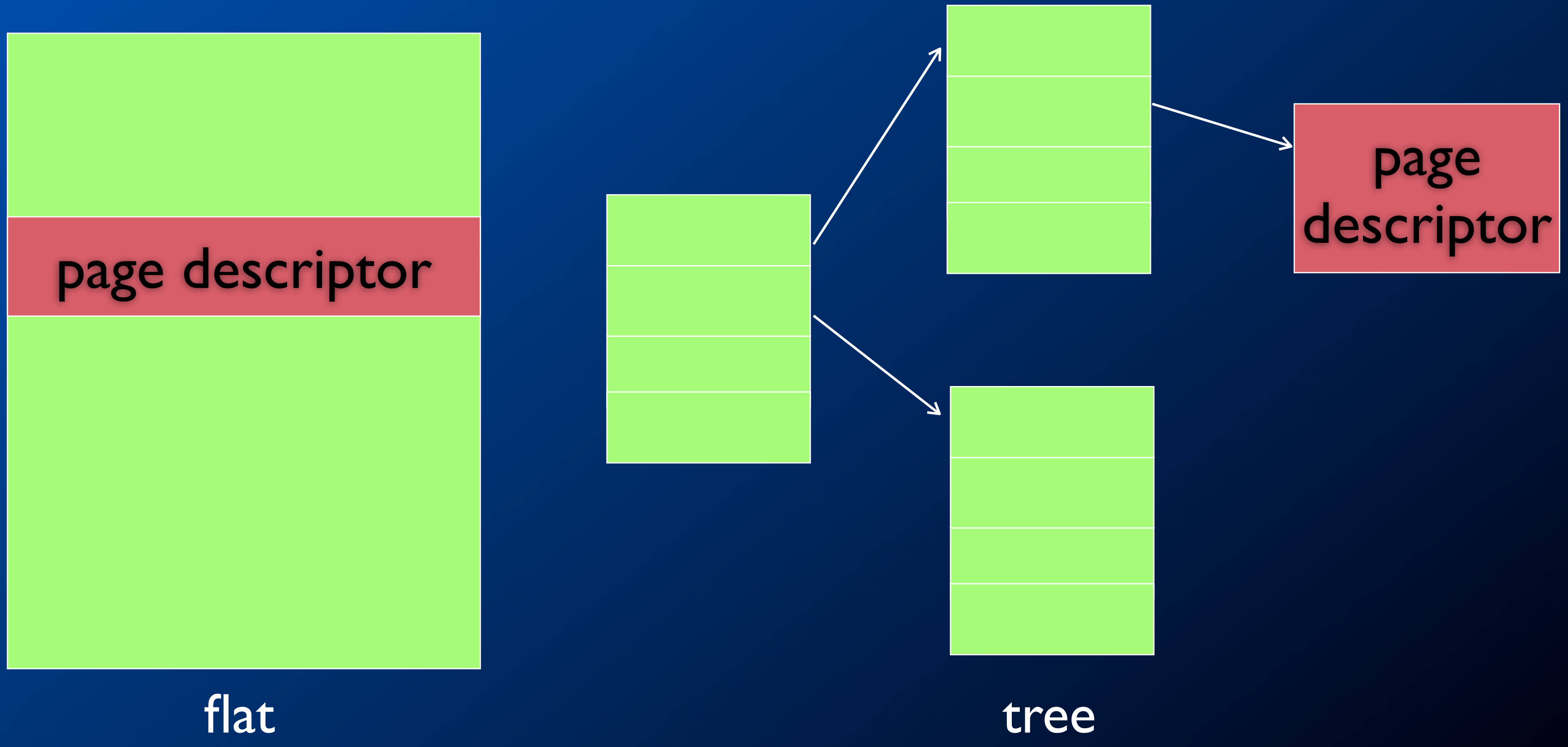
Segment address translation



Page address translation



Page table organizations



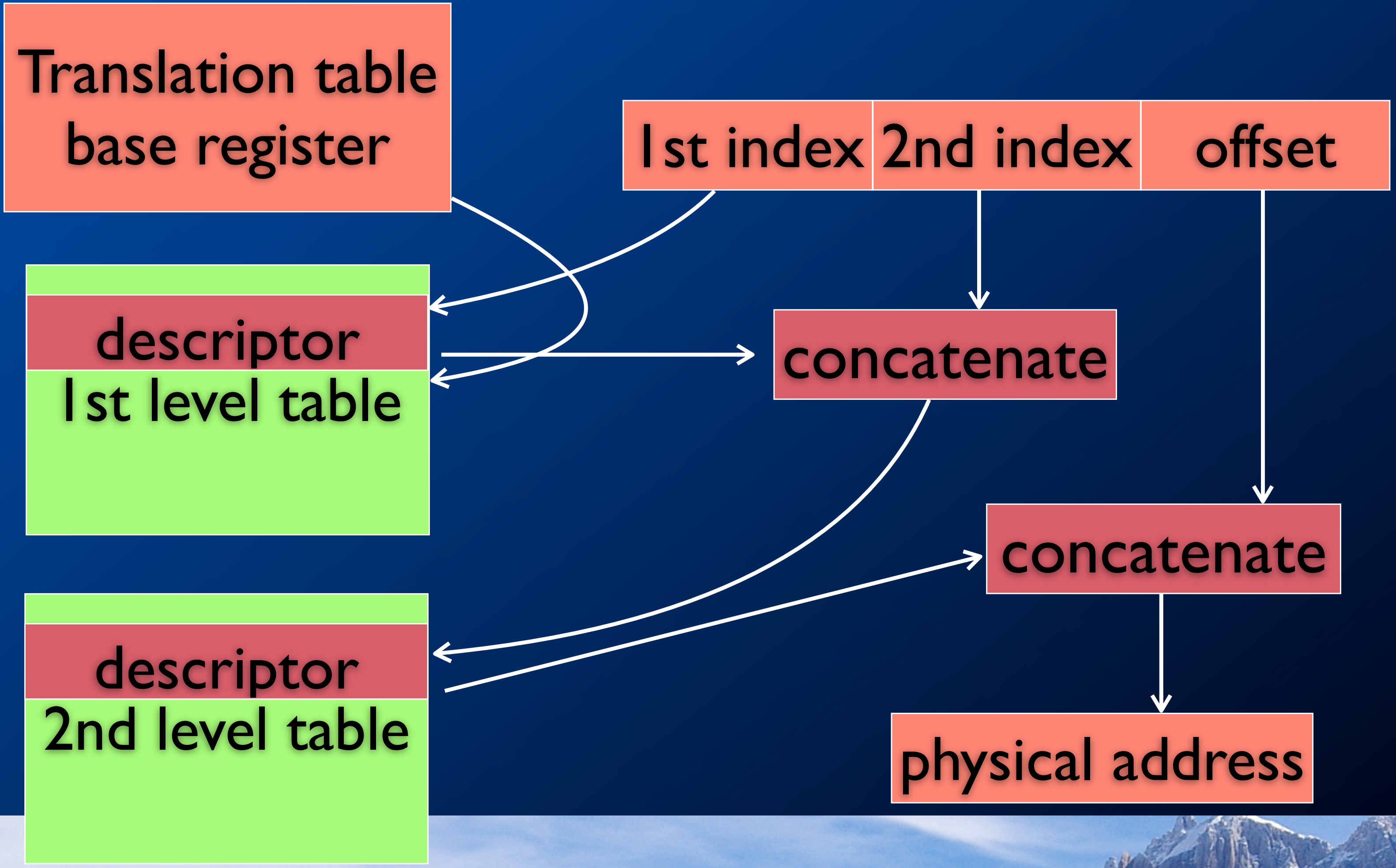
Caching address translations

- Large translation tables require main memory access.
- TLB: cache for address translation.
    - Typically small.

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



Thank you!