ARM Processor

COMP375

“What is the world’s fastest computer and how fast is it ? Currently, it’s an HP notebook. It’s used on the Space Shuttle to compute orbital position and has been clocked at 17,500 mph.”

Robert Hyatt

ARM Processor

- Most desktop and laptops computers use the Intel architecture
- Most cell phones and tablets use ARM processors
- The ARM processor is a RISC machine

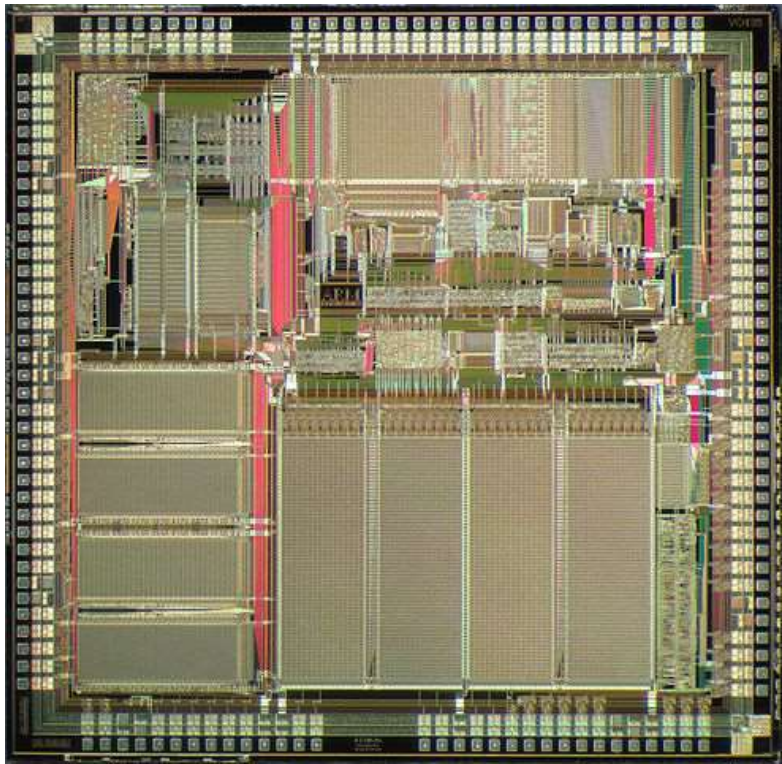


Photo By Pauli Rautakorpi

Architecture Design Company

- Originally the **Acorn RISC Machine** it was later named the **Advanced RISC Machine**
- British company ARM Holdings developed the architecture and licenses it to other companies
- ARM Holdings develops the design which is built by chip producers
- ARM's main CPU competitors include Intel (Atom), Imagination Technologies (MIPS) and AMD

ARM Success

- With over 50 billion ARM processors produced as of 2014, ARM is the most widely used instruction set architecture in terms of quantity produced
- In 2010, producers of chips based on ARM architectures reported shipments of 6.1 billion ARM-based processors, representing 95% of smartphones, 35% of digital televisions and set-top boxes and 10% of mobile computers

Click One

A. ARM & Intel design but do not manufacture their chips
B. ARM & Intel both designs and manufactures the chips
C. ARM & Intel design, but only Intel manufactures their chips
D. ARM & Intel design, but only ARM manufactures their chips
E. None of the above

History

By Jussi Mononen - Flickr: Sophie Wilson Presenting

- First developed in 1983 by Sophie Wilson
- First ARM based computer build in 1987
- Originally a 32 bit architecture
- Upgraded to 64 bit in 2011

Hardware Implementation

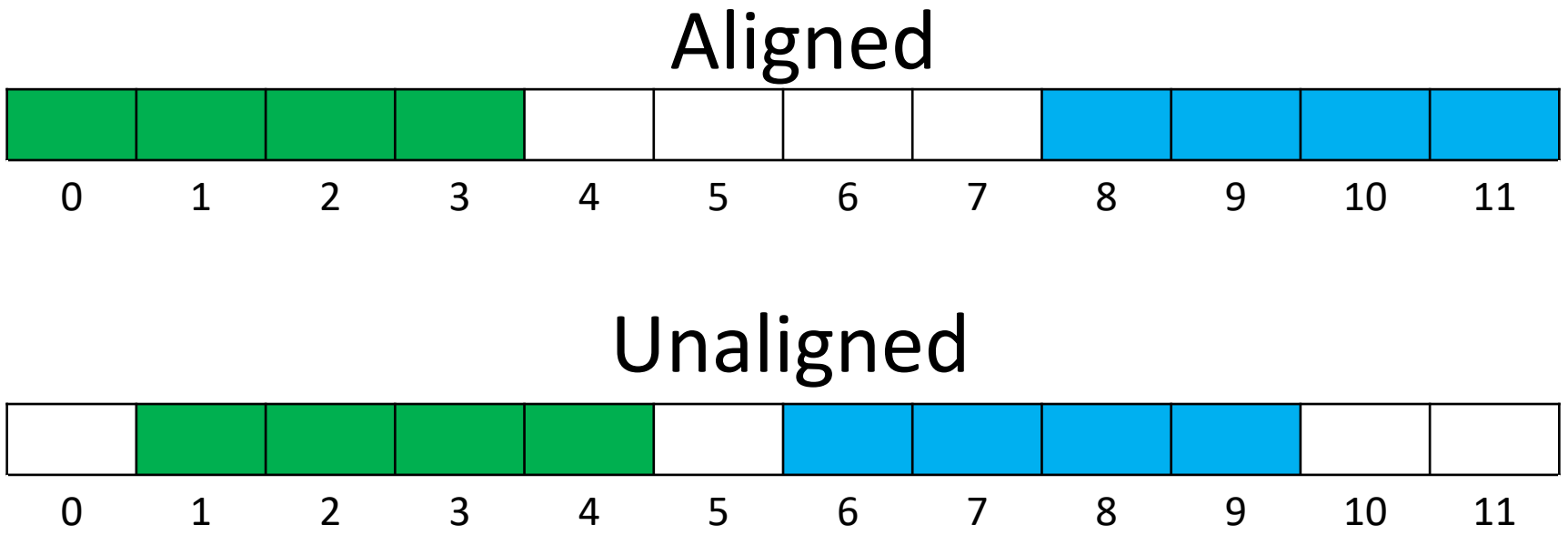
- The first ARM processors had about 30,000 transistors
- The Motorola 68000 (designed about six years earlier) had around 40,000 transistors
- Much of this simplicity came from the lack of microcode (which represents about one-quarter to one-third of the 68000 transistors) and from not including any cache (common in that era)
- This simplicity enabled low power consumption, yet better performance than the Intel 80286

Architectural Features

- Load/store architecture
- 16 uniform 32-bit registers, including the program counter, stack pointer and the link register
- Fixed instruction width of 32 bits
- Mostly single clock-cycle execution
- No support for unaligned memory accesses

Alignment

- Many computer architectures require instructors and/or data to aligned
- A 4 byte data value is aligned if its address is evenly divisible by 4



Registers

- The ARM processor has 16 registers of 32 bits
- R13 is the Stack Pointer
- R14 is the Link Register
- R15 is the Program Counter
- When an FIQ interrupt occurs, the CPU uses a special set of R8 to R12 registers. Therefore these registers do not have to be saved upon an interrupt
- System mode has its own R13 and R14

A jump instruction can be implement as

A. Load R15, address
B. Store R15, address
C. Load R14, address
D. Load R15, R14

FIQ Interrupts

- Fast Interrupt Requests (FIQs) are a specialized type of Interrupt Request
- A context save is not required for servicing an FIQ since it has its own set of registers. This reduces the overhead of context switching
- Only one source of FIQs at a time, so the interrupt handler does not have to determine the source

CPU Modes

- User mode: The only non-privileged mode
- FIQ mode: Entered by an FIQ interrupt
- IRQ mode: Entered by an IRQ interrupt
- Supervisor (svc) mode: Entered when an SVC is executed
- Abort mode: Entered whenever a prefetch abort or data abort exception occurs
- Undefined mode: Entered when an undefined instruction exception occurs
- System mode: Entered by setting the status bits

An undefined instruction

A. has an unused opcode value
B. does not use the address field
C. performs fuzzy logic
D. is for later implementation

ARM models

- The original ARM series runs up to ARM11
- In 2004, the Cortex series was created
  - Microcontroller
  - Real-time
  - Application(32-bit)
  - Application (64-bit)

Saving Bits

- The ARM2 featured a 32-bit data bus, 26-bit address space and 27 32-bit registers
- Eight bits from the program counter register were used for other purposes
  - the top six bits served as status flags
  - bottom two bits (available because the program counter was always word-aligned) were used for setting modes

Extended Addressing

- In ARM6, addressing was extended to 32 bits
- Program machine language still had to lie within the first 64 MB of memory in 26-bit compatibility mode
- Other execution modes allowed full 32 bit addressing

Conditional execution

- Like the Intel Pentium processors, ARM arithmetic instructions and the compare instruction set the condition codes
- In the Intel Pentium, conditional jump instructions are based on the condition code
- In the ARM, any instruction can be based on the condition codes

Example Program

- Consider the C or Java code

```
do {
        if (dog > cat)
                dog -= cat;
        else if (dog < cat)
                cat -= dog;
} while (dog != cat);
```

Intel Assembler

- Assume eax is dog and ebx is cat

```
        cmp eax, ebx
loop:   jle   small      ; jump if dog ≤ cat
        sub   eax, ebx   ; dog = dog - cat
        jmp   equal
small   je    equal      ; jump if dog = cat
        sub   ebx, eax   ; cat = cat - dog
equal   cmp eax, ebx
        jne   loop       ; loop if cat != dog
```

ARM Assembler

- Assume R1 is dog and R2 is cat

```
loop:   CMP    R1, R2       ; set condition "NE" if (dog != cat),
                            ;                  "GT" if (dog > cat),
                            ;                  or "LT" if (dog < cat)
        SUBGT  R1, R1, R2   ; if "GT" (Greater Than), dog = i-cat;
        SUBLT  R2, R2, R1   ; if "LT" (Less Than), cat = cat-dog;
        BNE    loop         ; if "NE" (Not Equal), then loop
```

Intel Pentium vs. ARM

- To implement the same code, there were 8 Intel assembler instructions and 4 ARM instructions
- The Intel assembler has 4 jumps while the ARM has 1
- The conditional execution does not interfere with the pipeline while Intel jumps do

Pipelining

- Early implementations had a three-stage pipeline
  - fetch, decode and execute
- Later models have longer pipelines
  - The Cortex-A8 has thirteen stages

Java Bytecodes

- Java is generally compiled to bytecodes which are stored in a .class file
- Bytecodes are like machine language for an imaginary stack oriented computer
- Bytecodes are usually interpreted by the Java Virtual Machine or compiled to native machine language by a Just In Time (JIT) compiler

Java on ARM

- **Jazelle** models of ARM can provide hardware execution of Java bytecodes
- Jazelle provides an additional execution state for execution bytecodes instead of ARM instructions
- Between 134 and 149 bytecodes (out of 203 bytecodes specified in the JVM specification) are translated and executed directly in the hardware
- ARM has a **Branch to Java** instruction to start Java bytecode execution

Combined Hardware and Software Interpretation

- With Jazelle, an ARM processor can execute the most commonly use bytecode instructions
- Bytecode instructions not executed directly by the hardware are routed to software in standard ARM instructions
- ARM claims that approximately 95% of bytecodes in typical program usage are directly processed in the hardware

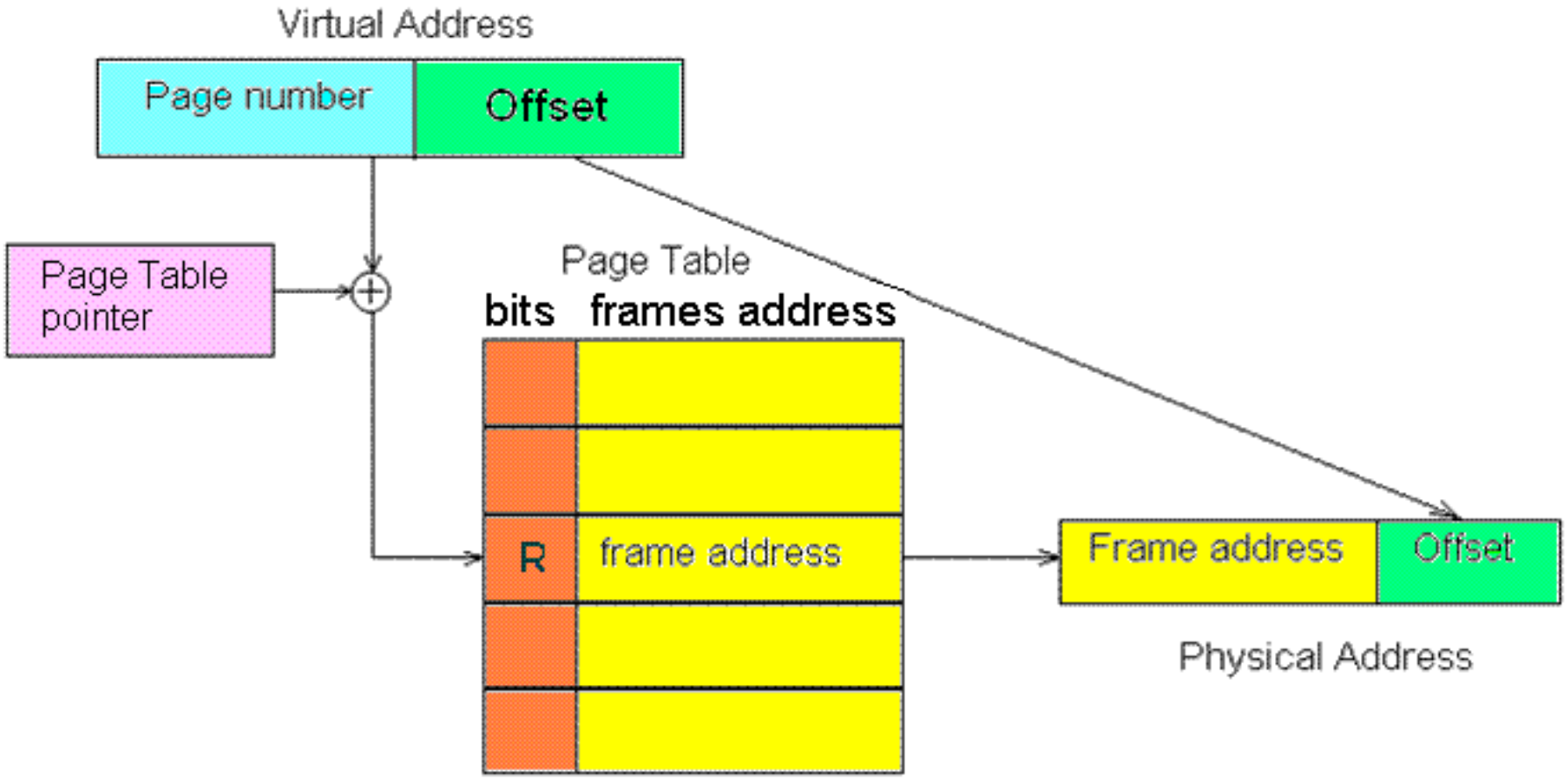
Translation on Instruction Fetch

- Jazelle uses a low-level binary translation, implemented as an extra stage between the fetch and decode stages in the processor instruction pipeline
- Recognized bytecodes are converted into a string of one or more native ARM instructions

Virtual Memory

- The ARM processor supports virtual memory
- Many large programs can be run in a small amount of RAM by only keeping the needed portions of programs in RAM
- When a program accesses a memory location not in RAM (indicated by the residency bit for the page being clear), the hardware creates a Page Fault Interrupt
- The OS finds a page, reads that portion of the program into the page, updates the page table and has the program re-execute the instruction

Address Translation



Android Memory

- The Android OS runs on ARM computers
- Android does not support virtual memory
- All of an application program is copied to RAM before execution starts
- Android limits program size to a much greater extent than Windows or Linux

The ARM processor

A. does not support virtual memory
B. has the hardware to support virtual memory
C. does not support Android
D. All the above
E. None of the above

Android Memory Categories

- Clean or Dirty
  - Clean memory is mapped into the address space and has not been changed
  - Similar in concept to traditional virtual memory
- Shared or Private
  - shared: used by many processes
  - private: used by only one process

Android Memory Use

- Application Dex files are clean private
- Library Dex files are clean shared
- Application heap and local data are dirty private
- Dirty shared memory includes
  - library “live” Dex structures
  - shared copy-on-write heap (mostly not written)

Linking to Libraries

- Android supports shared libraries. All applications use the same copy of the library in memory
- Windows has the same concept called Dynamic Link Libraries (DLL)
- The pages of a shared library are included in a user's page table

Copy on Write

- Android shares both the machine language instructions and the data portion of libraries
- If a page of the shared library is changed by an app, a copy of the shared page is made for the app to use
- Shared pages are always clean

Copy on Write Implementation

- The data pages of a shared memory page are marked as read-only in the page table
- When an app tries to store into the page, an interrupt occurs
- The OS finds available memory, copies the shared page to the new space and updates the app's page table to point to the new page

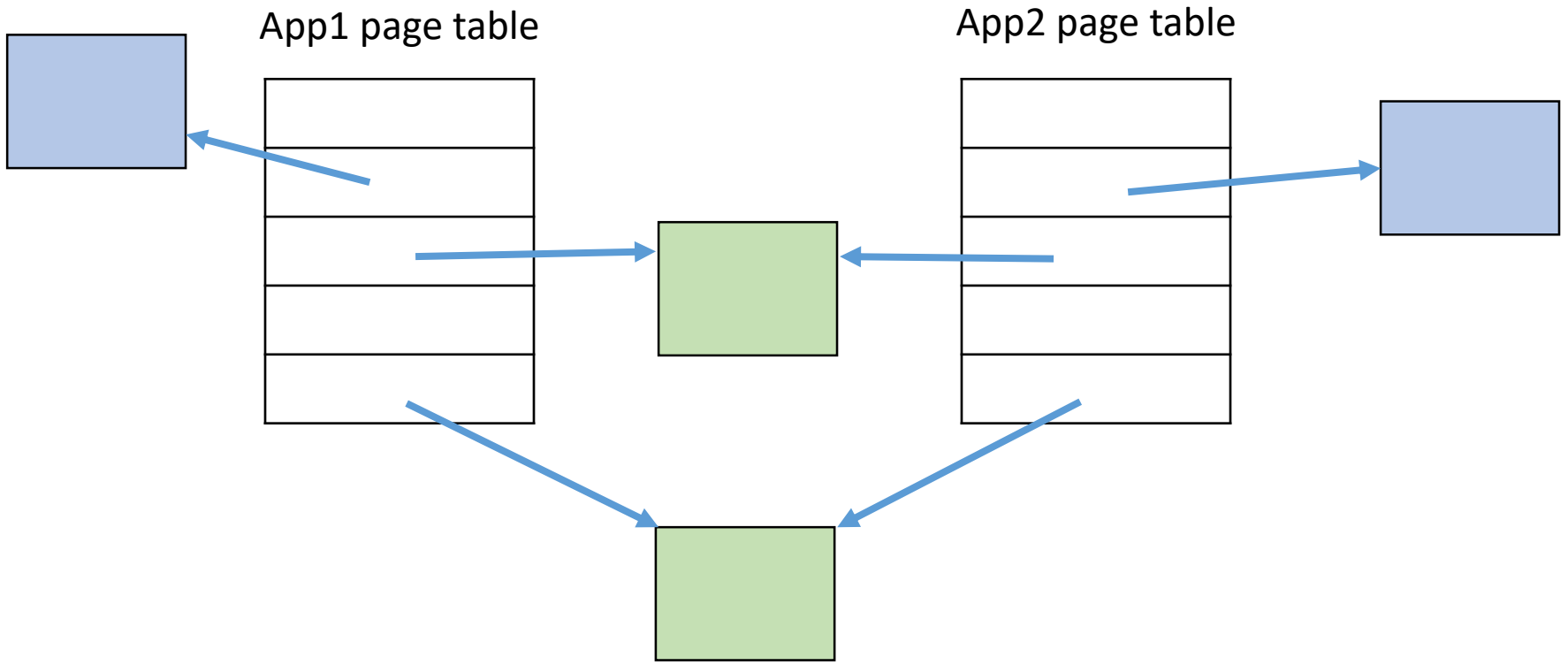
Copy on Write

- Android shares both the machine language instructions and the data portion of libraries
- If a page of the shared library is changed by an app, a copy of the shared page is made for the app to use
- Shared pages are always clean

Copy on Write Implementation

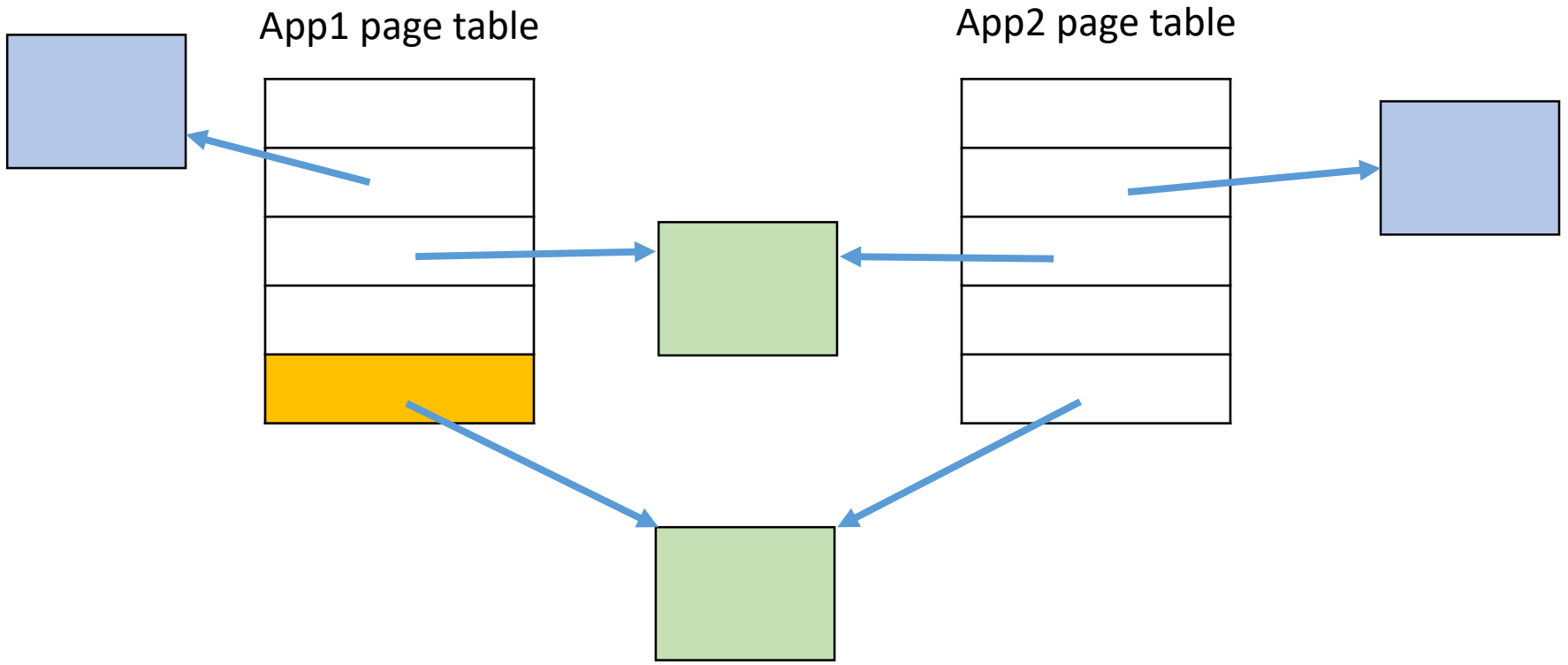
- The data pages of a shared memory page are marked as read-only in the page table
- When an app tries to store into the page, an interrupt occurs
- The OS finds available memory, copies the shared page to the new space and updates the app's page table to point to the new page

Page Tables and Program Memory



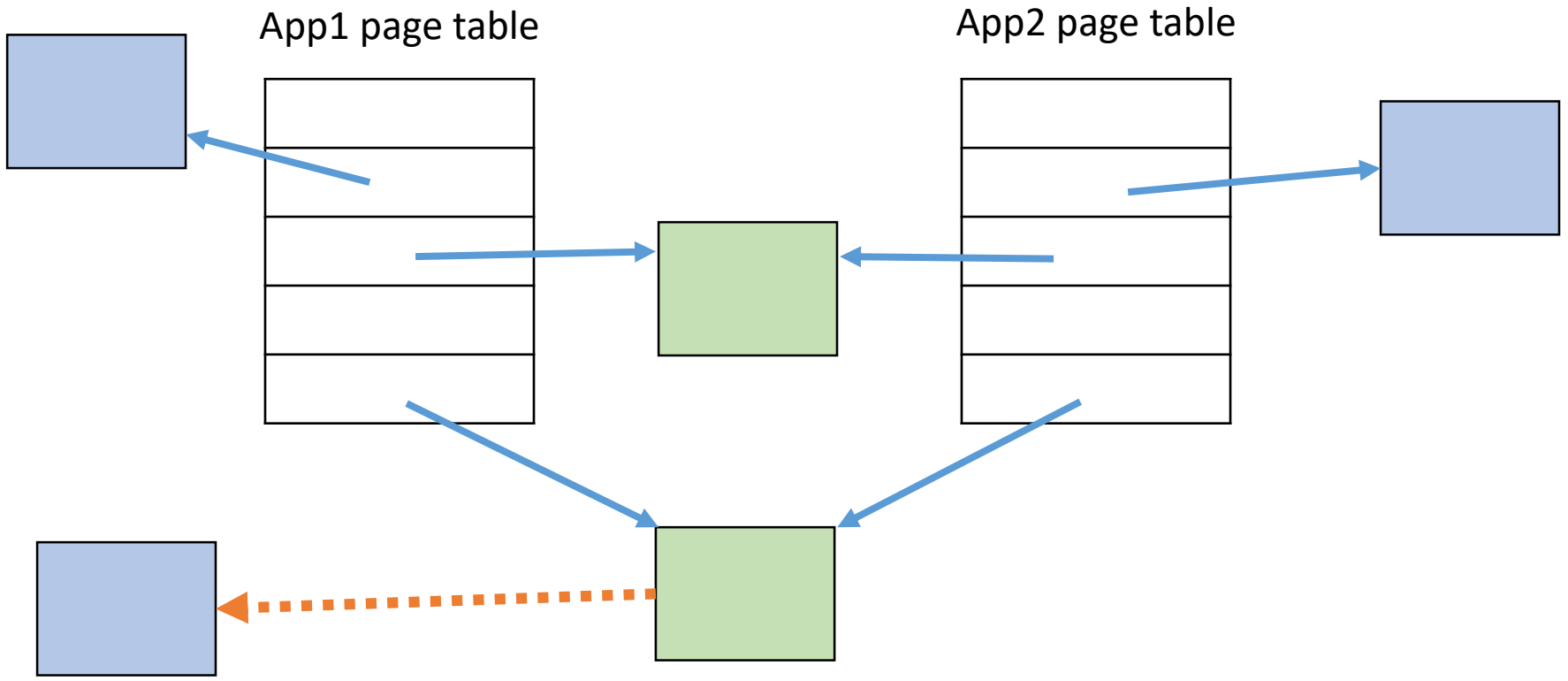
All shared pages are marked as read-only in the page tables

App1 writes to a shared page



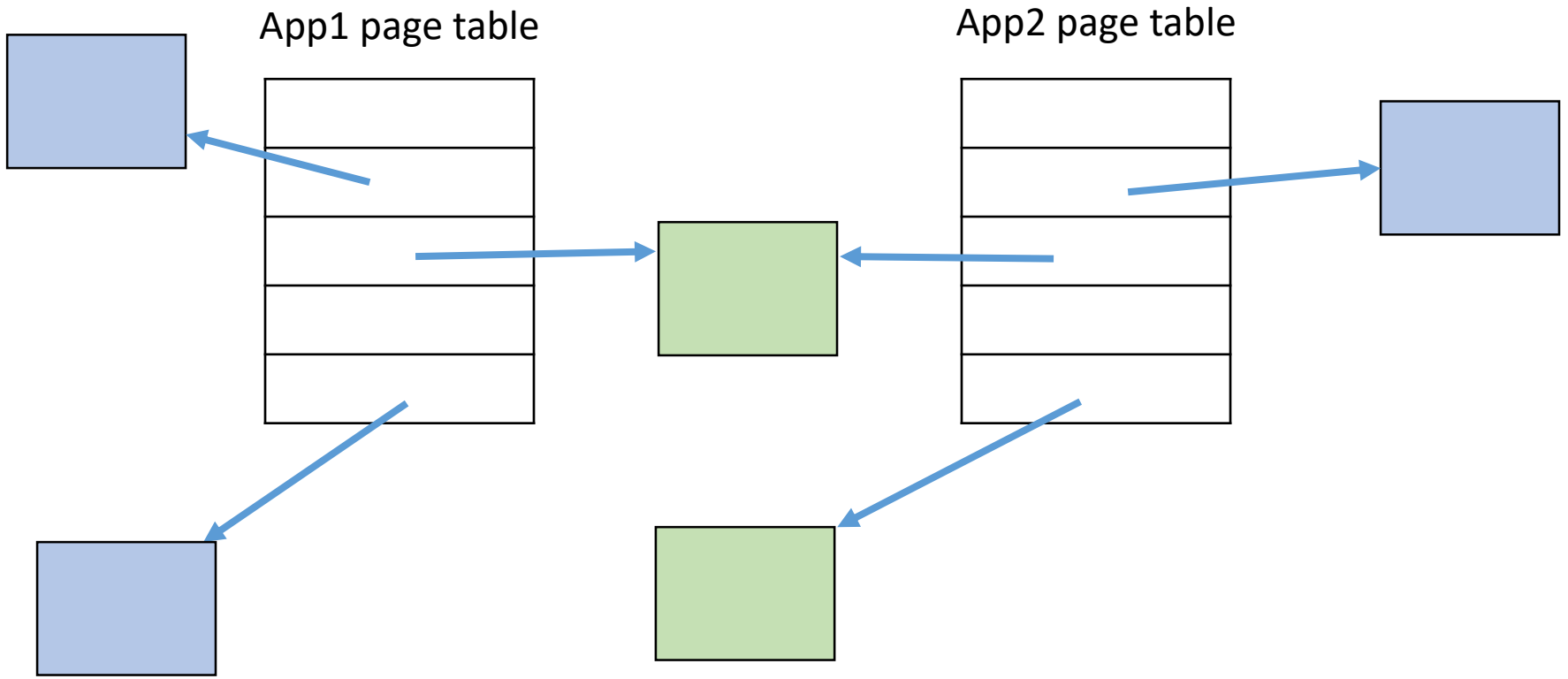
A read-only violation interrupt occurs

A copy is made of the shared page



A free page is acquired and the shared page is copied to the new page

The unshared page is now used



The page table for that address now points to the new copy