

Pr. Olivier Gruber

Full-time Professor

Laboratoire d'Informatique de Grenoble

Université de Grenoble-Alpes

Olivier.Gruber@imag.fr

- Reference Book

*Virtual Machines*  
*Versatile Platforms for systems and processes*

James E. **Smith**, Ravi **Nair**

Morgan Kaufmann

- Research Articles

- Cited on various slides

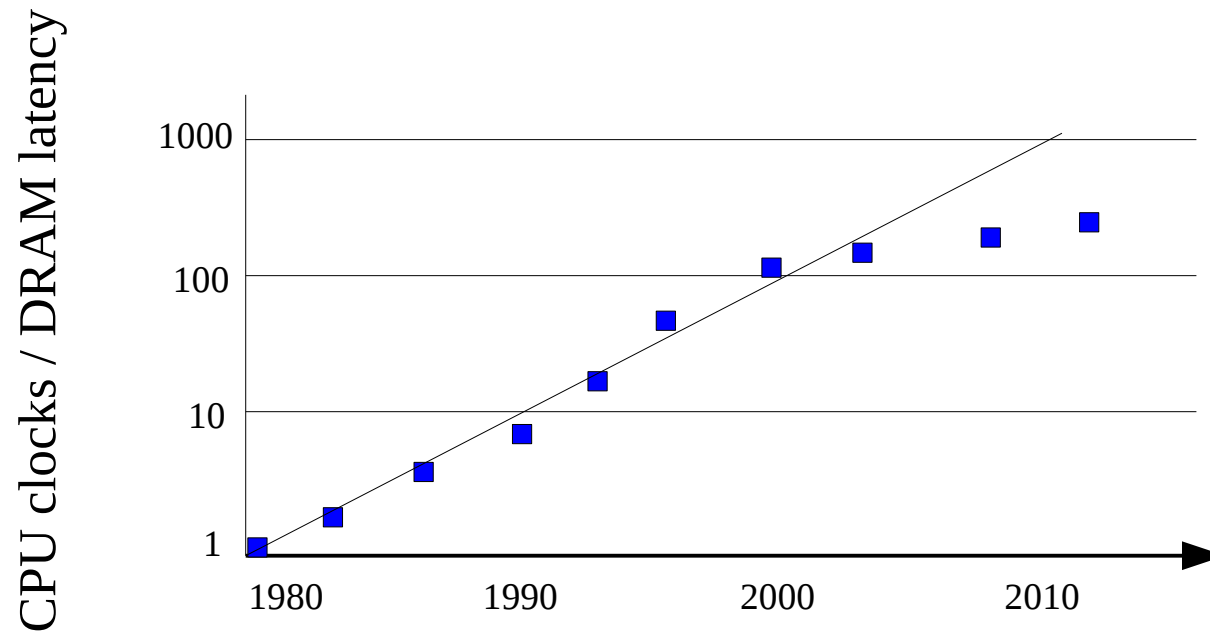
## Extra Slides On Real Machines

- Multiple facets
  - Transistor improvements
  - Architectural improvements
- Challenges
  - Memory latency
  - Power efficiency
  - Software obesity
  - Exploiting parallelism

# Memory Wall

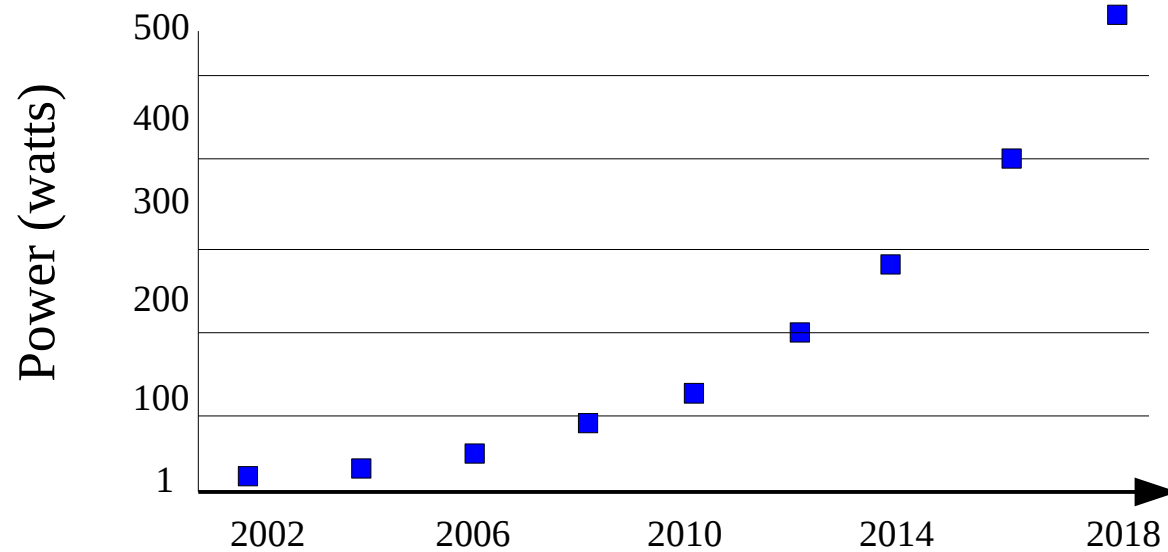
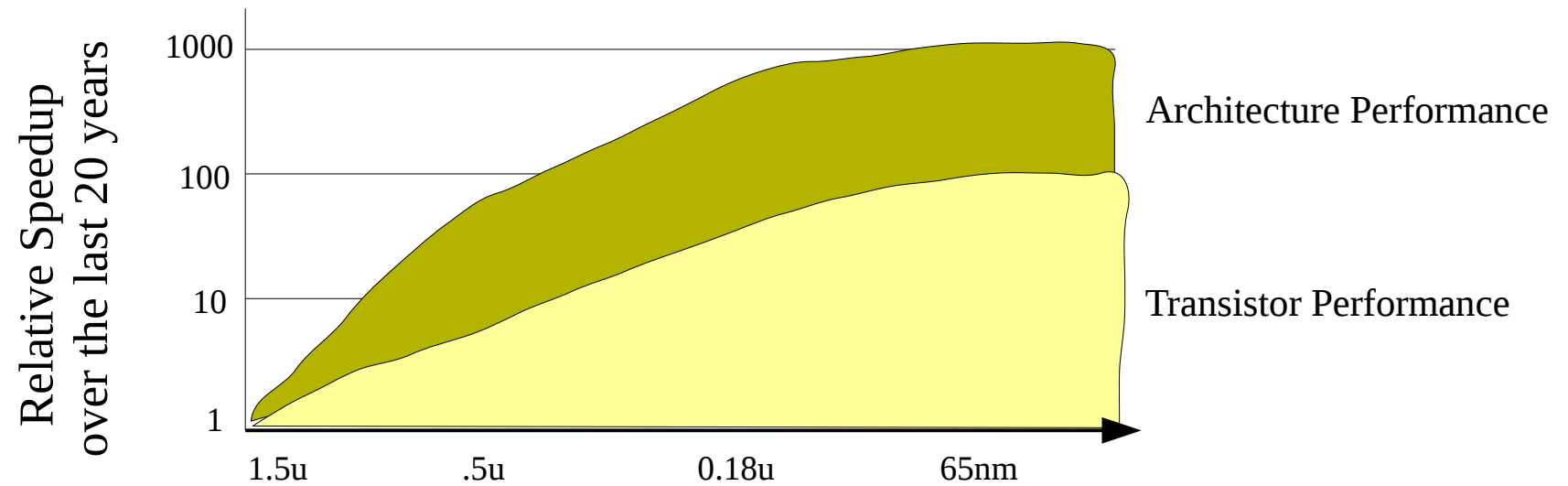
5

- The challenge is memory latency...
  - Well over 100 of cycles
  - Flattening due to the flattening of CPU clock frequency



# Discussing the Limits

6



- Memory Hierarchy
  - L1 and L2 caches
  - TLBs
- Internal core parallelism
  - Pipelining and Superscalar
  - Out-of-order execution (dependency driven execution)
- Speculative execution
  - Branch prediction and speculative execution
  - Complete speculative execution
- Hardware JIT compilation
  - Optimize, reorder, and parallelize
  - But maintain the outside contract (apparent sequential execution)

- Pros

- Poorly written software execute fast
- Peak performance is fantastic
- Some of the gains can only be done in hardware at runtime

- Cons

- Energy consumption and therefore heat production
- Large surface on dies (limits registers, L1 cache size, etc.)
- Not always effective (some programs are not faster)
- Traps and interrupts are more expensive
- Impacts the programming model
  - Requires fences with multiple cores
  - Cache flushes when writing device drivers



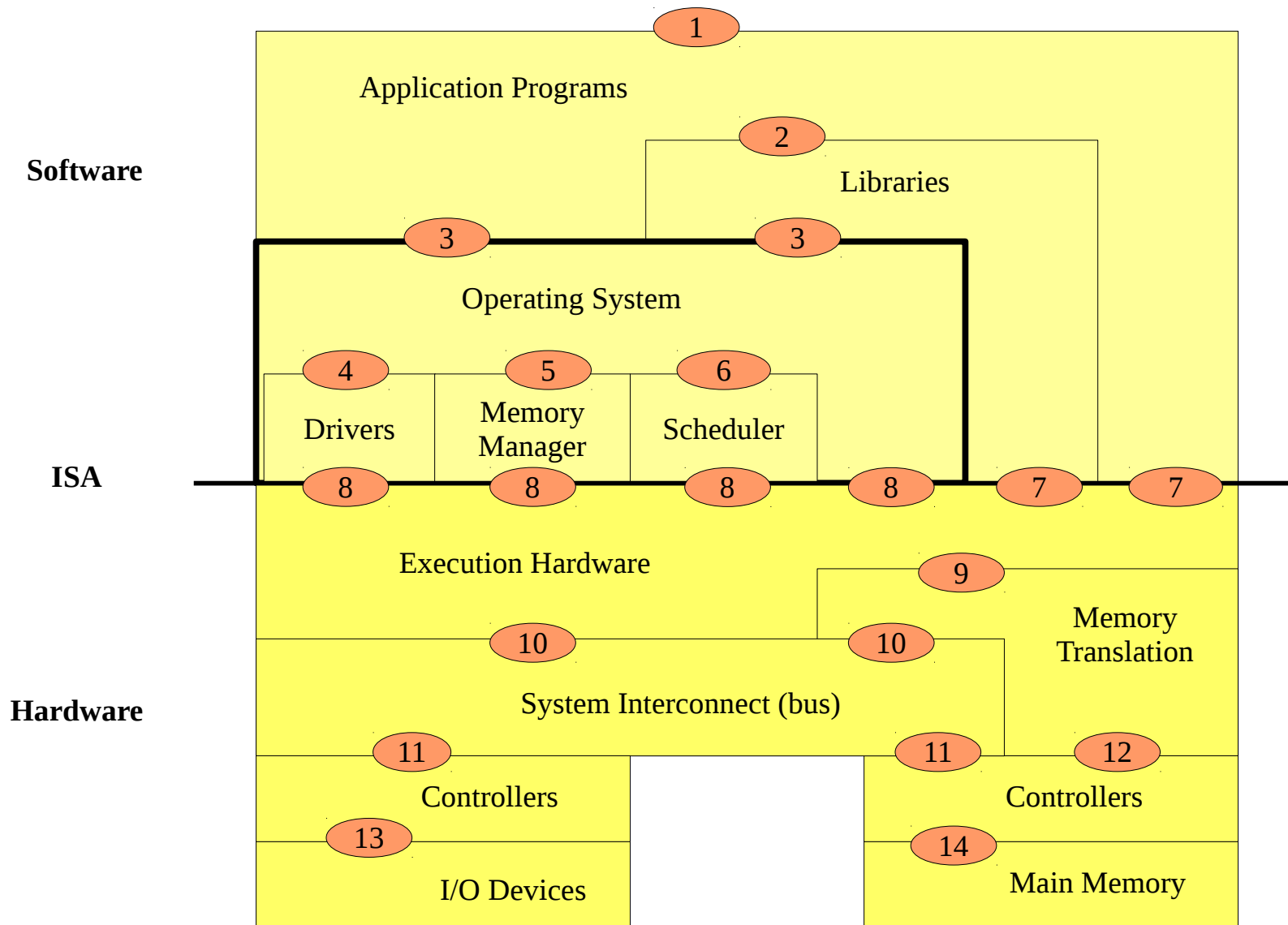
- Multiple cores
  - Performance through parallelism
- Success stories
  - GPUs for graphics
  - High-Performance Computing (HPC)
- Challenges
  - Who master parallel programming?
  - Not all applications have parallelism
  - Load-balancing versus execution locality
  - Race update conditions on shared data
  - Scalability of locking mechanisms
  - Energy consumption

- Computer System Basics

- Outline of the major components of computer systems
  - Their interfaces
  - The resources managed through those interfaces
- We will look at
  - Primary hardware components
    - Processor, memory and I/O
    - Instruction Set Architecture (ISA)
  - Organization of a traditional operating system
    - Emphasis on managing system resources
    - Such as the processor, memory, or I/O devices
  - Discussing microkernels
    - Architecture, design, acceptance and performance

# Computer System Architecture

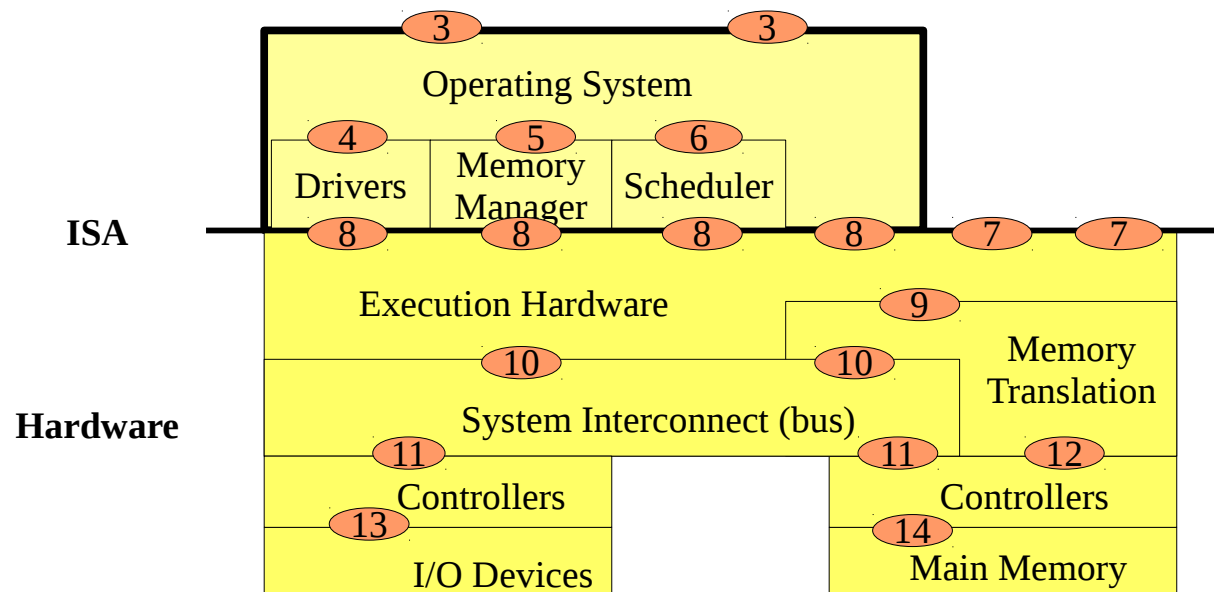
11



# Instruction Set Architecture

12

- Instruction Set Architecture has two parts
  - User-level ISA (7)
    - Aspects that are visible to non-privileged code
  - System-level ISA (8)
    - Aspects that are visible to privileged code
    - Of course, the system-level ISA includes the user-level ISA



- **Application Binary Interface (ABI)**

- User-level ISA

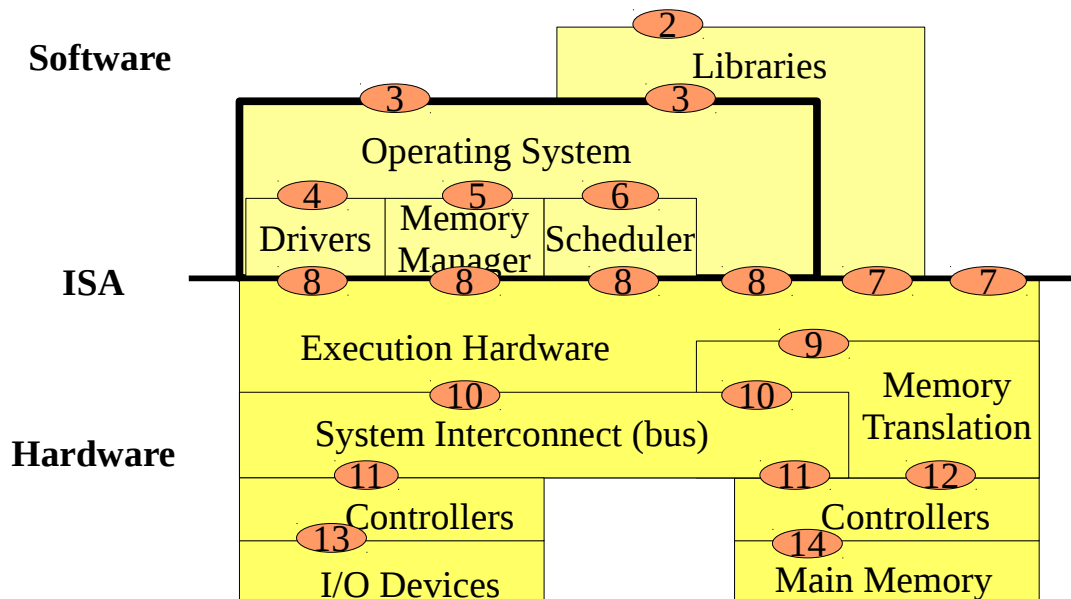
7

- Aspects that are visible to non-privileged code

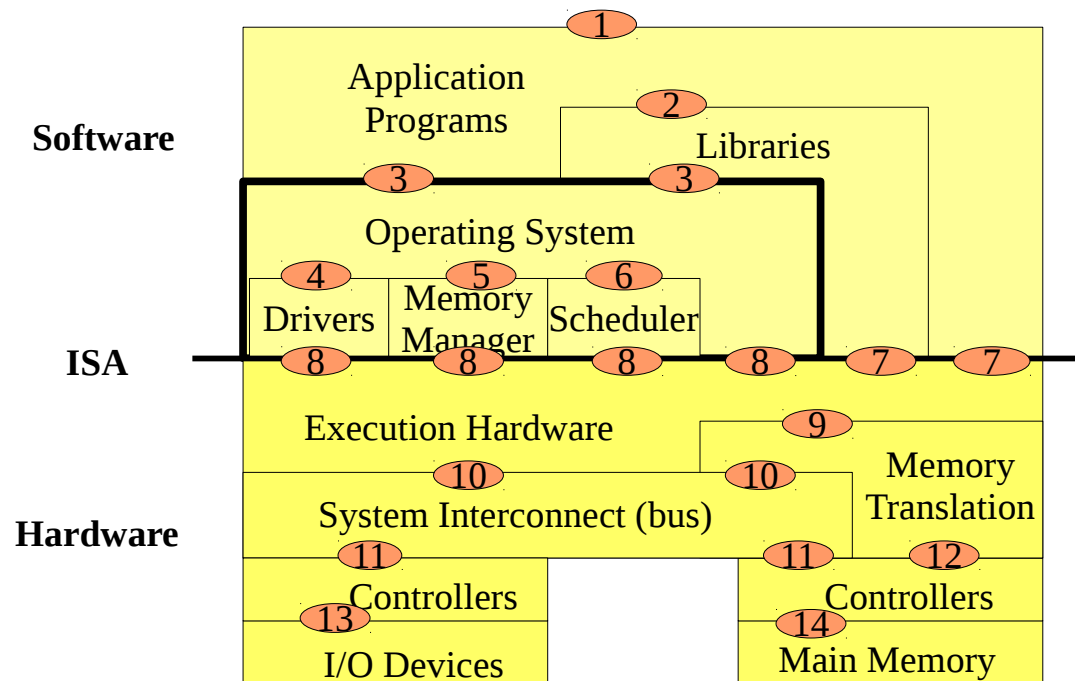
- System-call interface

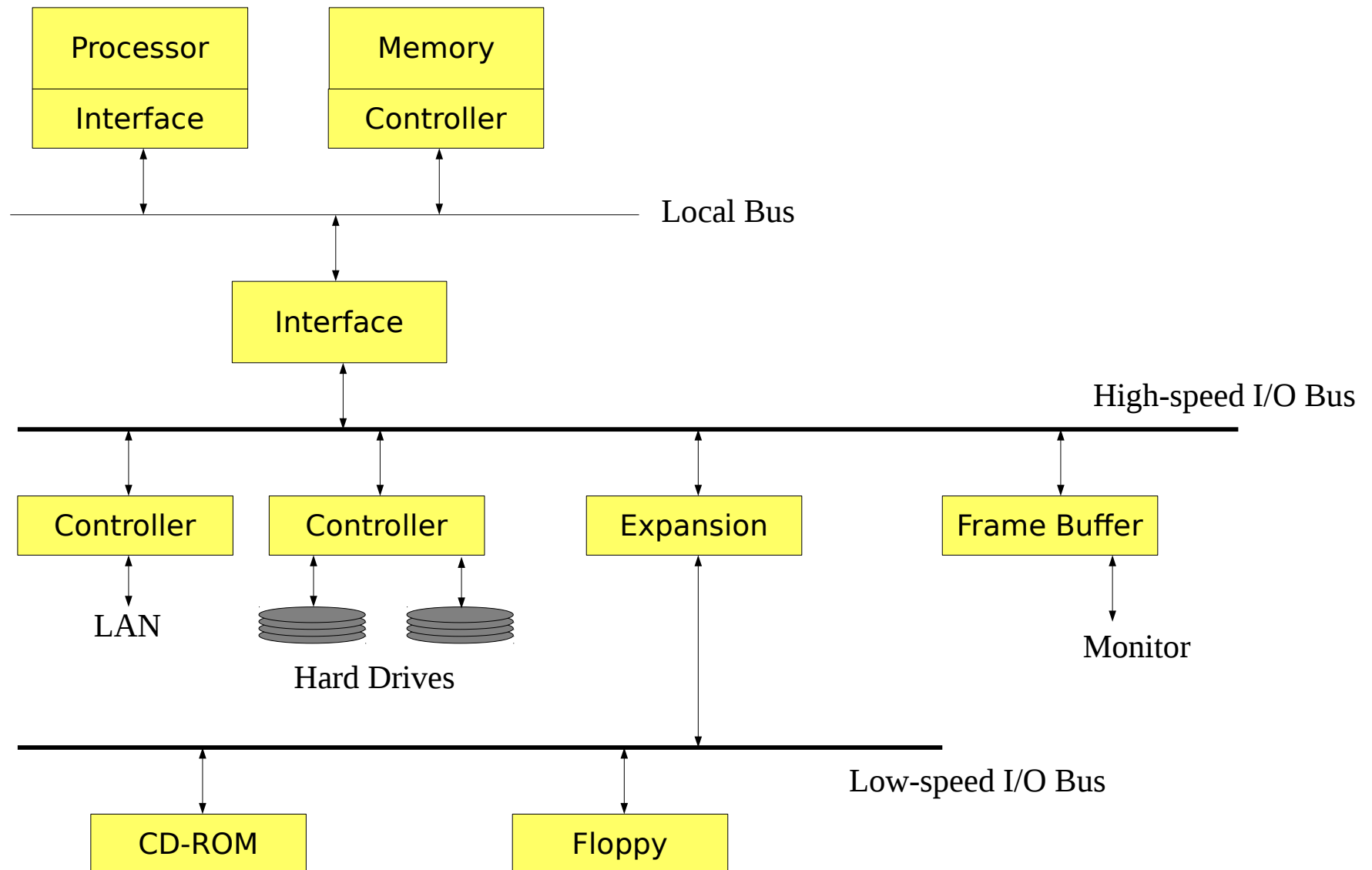
3

- Provide indirect access to shared resources
    - System calls use a trap mechanism to privileged code in the OS
    - Each operating system specifies how parameters are passed

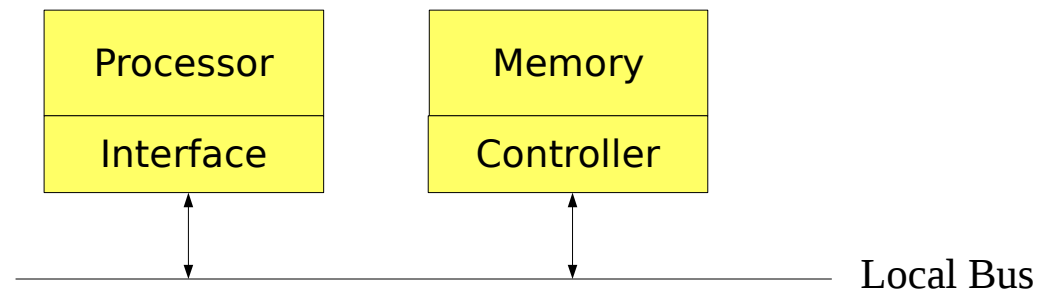


- Application Programming Interface (API) 2
  - Usually defined with respect to a *High-Level Language* (HLL)
  - A key element is the definition of standard libraries
  - Such libraries are defined at source-code level





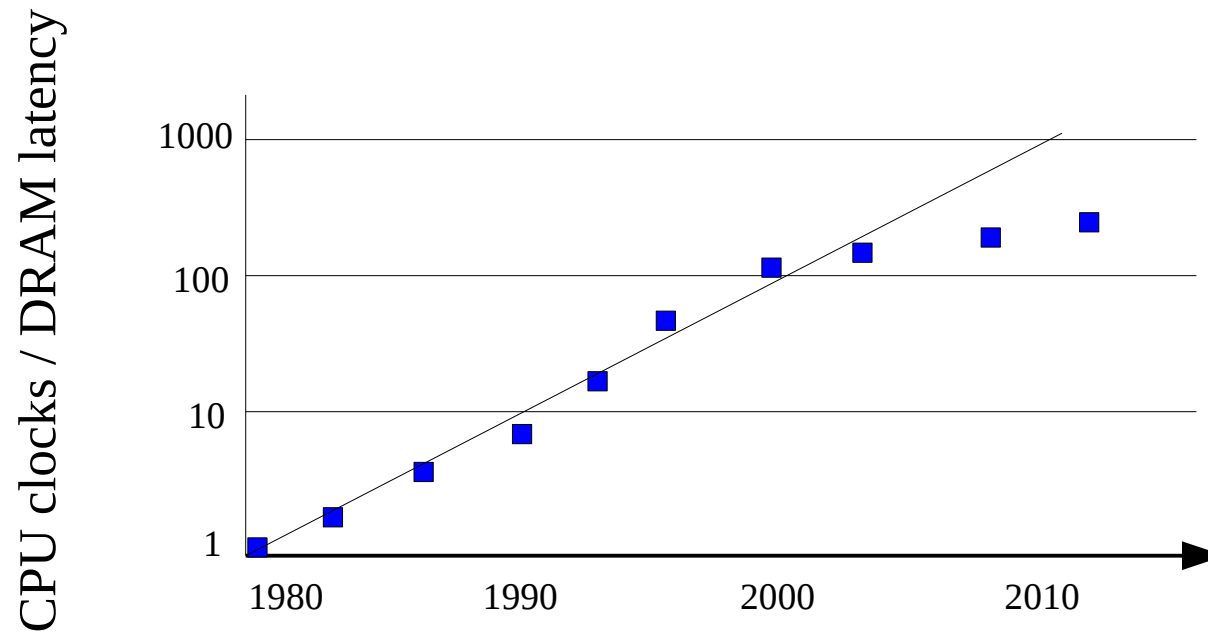
- Basic processing
  - Fetch, decode, and issue instructions
  - You could write a simple interpreter...
- CISC<sup>(1)</sup> or RISC<sup>(2)</sup>
  - Complex instructions versus simpler instructions
  - RISC = "*Relegate Interesting Stuff to Compilers*"
  - Sometimes CISC outside, but RISC inside...



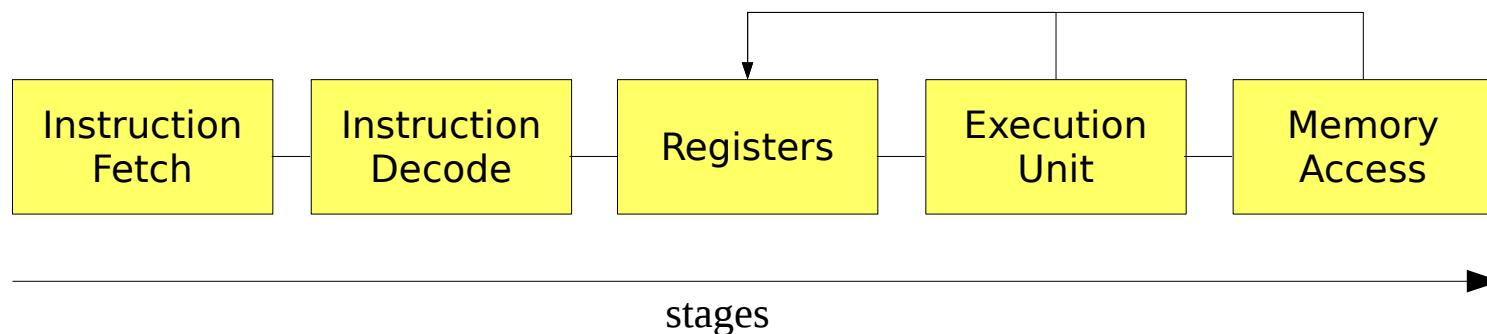
1. Complex Instruction Set Computers (CISC)
2. Reduced Instruction Set Computers (RISC)



- The challenge is the memory barrier...
  - Well over 100 of cycles
  - Flattening due to the flattening of CPU clock frequency



- Different types of processors
  - In-order pipeline
  - Superscalar
  - Very Long Instruction Word (VLIW)
- In-order pipeline
  - Multiple instructions may be in the pipeline at the same time
  - Only one instruction is in each stage at any given time
  - Stalls happen when instructions must wait for their operands



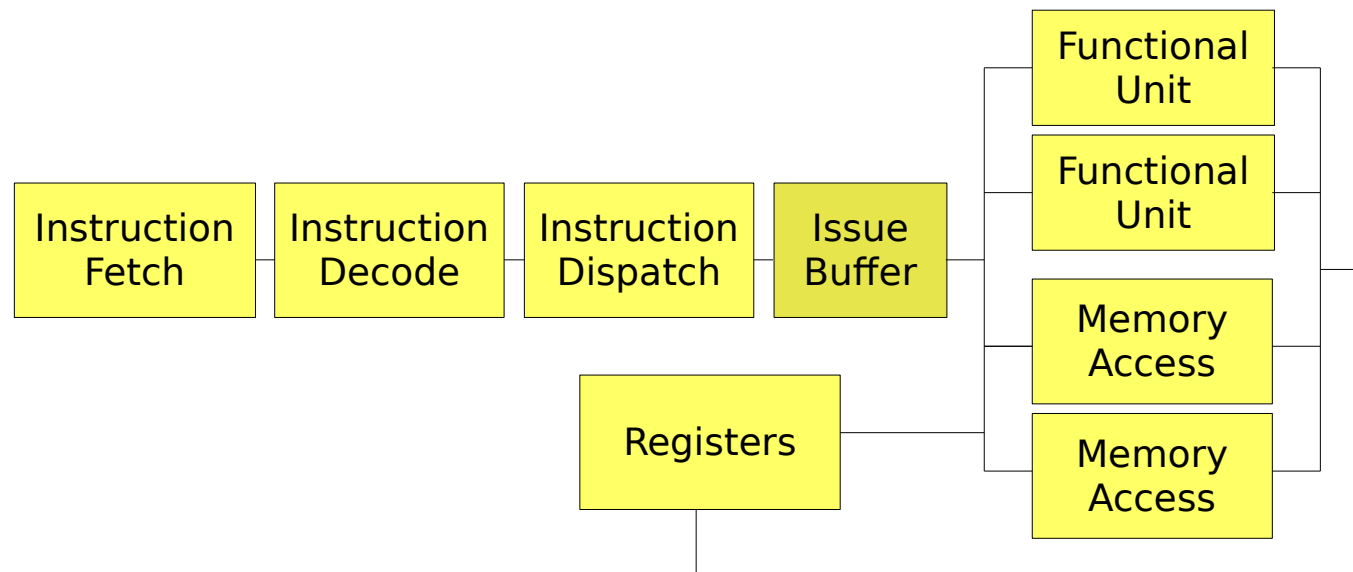
- High-performance processors

- Introduce automatic instruction-level parallelism

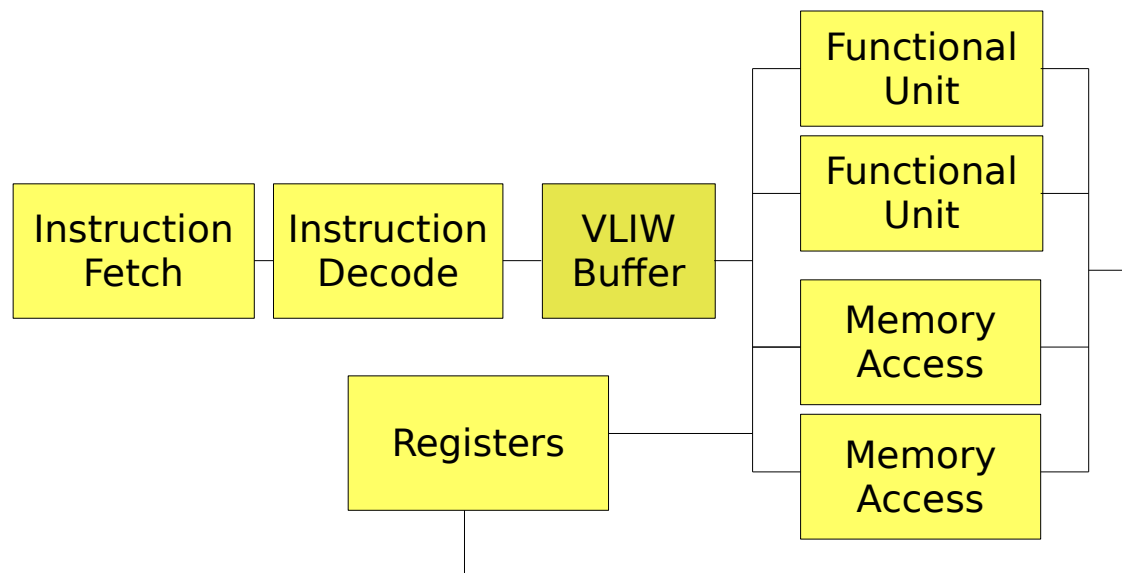
- Several instructions can be fetched and decoded in the same clock cycle
    - Decoded instructions are dispatch into instruction issue buffer
      - Begin execution when their input operands are ready
      - Without regard to the original program sequence

- Properties

- Peak instruction throughput is higher
    - Hopefully reduces stalls



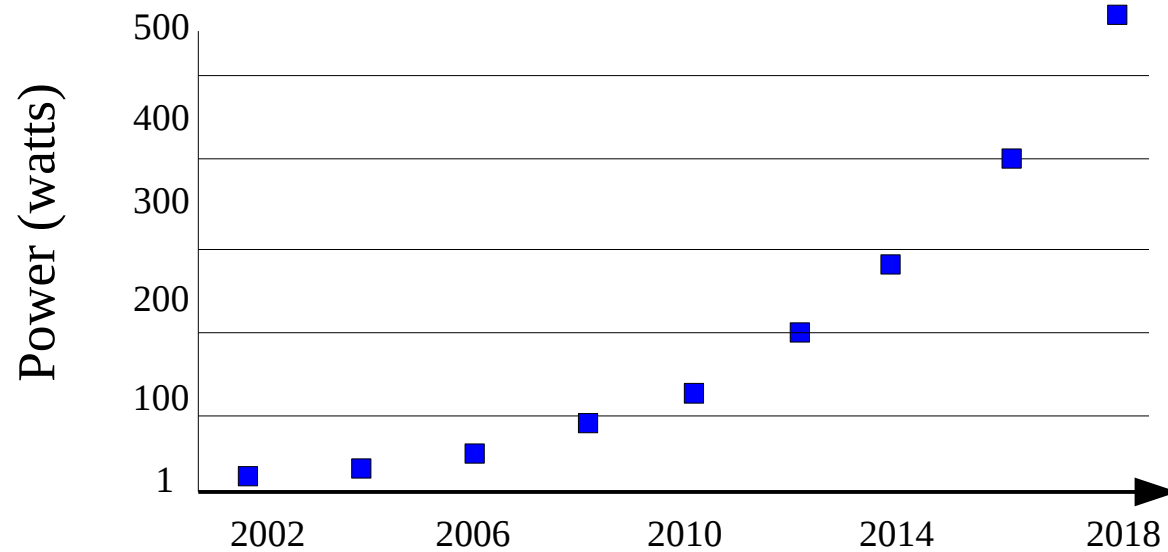
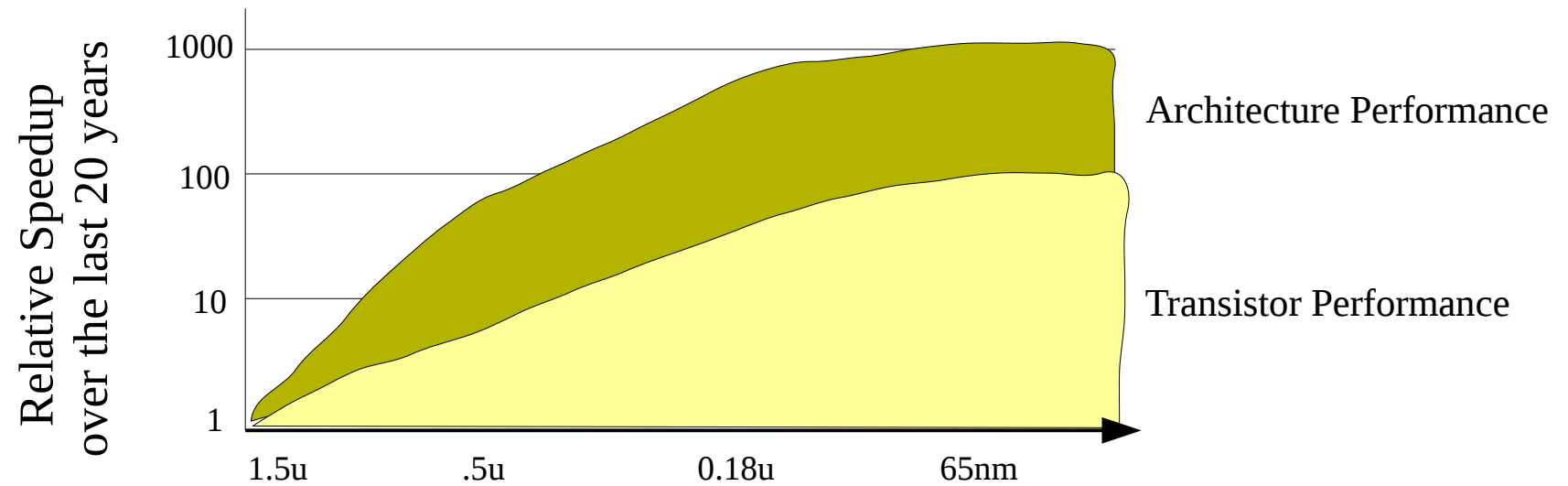
- Compilers must produce VLIW
  - Combine parallel instructions into a very long instruction word (VLIW)
  - VLIW are executed in sequence
- VLIW parallelism
  - A VLIW can be fetched and decoded in the same clock cycle
  - The instructions of the VLIW proceed in parallel
  - Begin execution when their input operands are ready



- Still the memory barrier...
  - As processors are going faster, the memory barrier is increasing
  - Can the hardware switch threads when staling?
- Operating system scheduling
  - More often well over thousands of instructions
  - Incompatible with the few-hundred-instruction-long stalls
- Faking cores
  - The OS sees multiple cores, but they are virtual cores
  - The hardware has everything it needs to context switch

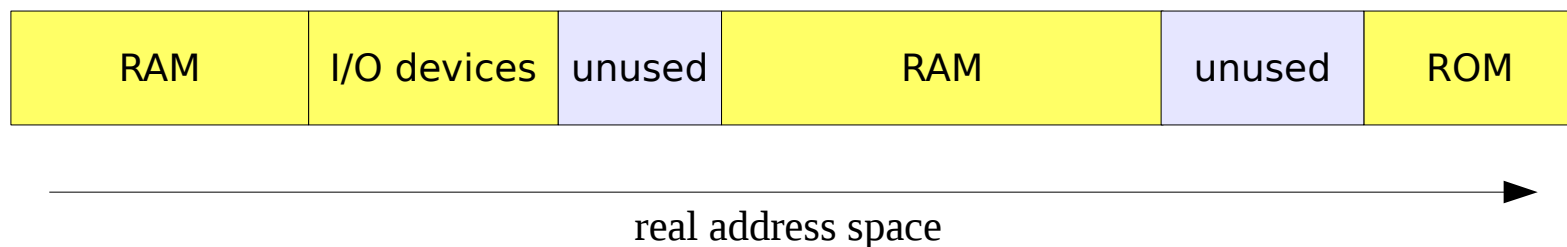
# Discussing the Limits

22



- Memory System

- A combination of main memory and cache memories
  - Cache memories are generally hidden from software (hardware managed)
  - Memory access is at least per byte, but it may be a word (16 or 32 bits)
  - Often, memory access is per *line* of 32 to 128 bytes
- Composite main memory
  - The address space may be composed of different types of memory
    - RAM, ROM, I/O memory, others
    - Each may have its own instruction sets for reading or writing
  - Usually divided in pages (like 4KB pages)
    - Pages may have different access privileges (read, write, execute)



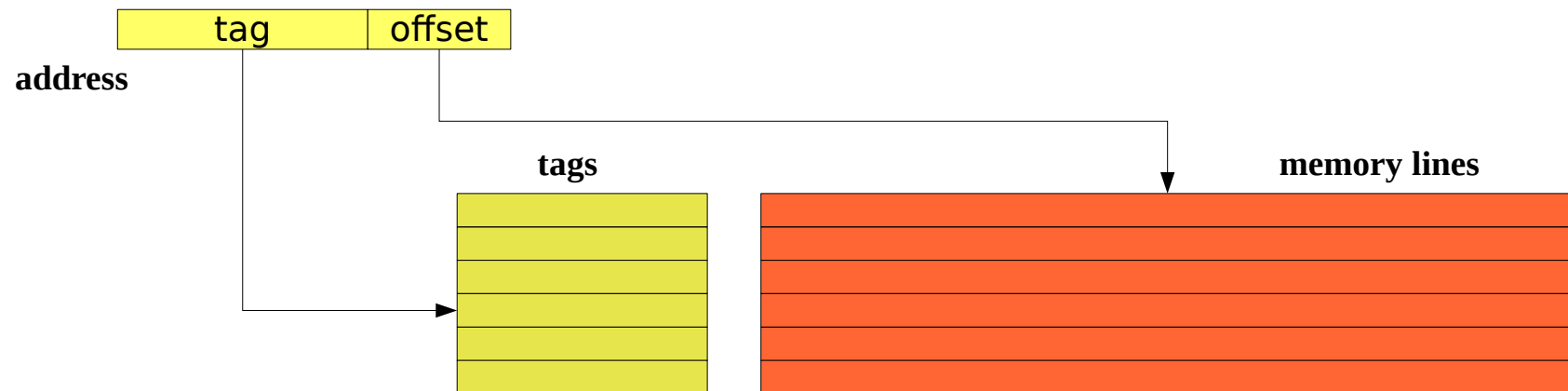
- Cache Memories

- Hiding high memory latencies
  - Many tens or hundred clock cycles (for in-memory pages)
- Works on the principle of *locality*
  - *Temporal locality* (what has been used recently is likely to be used again)
  - *Spatial locality* (what is close to what is being used is likely to be used)
- In 65nm technology
  - 10MB on-die cache (L1)
  - As much as 40% of total die area



- Cache Memories

- Caches memory lines (called *cache lines*)
  - A *cache hit* finds the addressed data in the cache
  - A *cache miss* does not and loads a memory line
  - A replacement algorithm must be in place to free cache lines



- Architecture

- Consist of a number of buses
  - That connect the processor and memory to I/O devices
  - Such buses are often standardized (PCI or AGP)
  - Devices often use a controller to connect to such buses
- A bus is a conduit for device commands and for data transfers

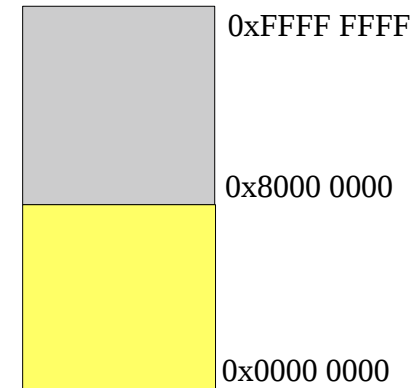
- Different Designs

- Programmed I/O
  - Processor issues a request and polls for its completion
- Interrupt-driven I/O
  - Processor issues a request and is interrupted when completed
  - Processor controls any data transfer from the controller to memory
- DMA I/O
  - Processor issues a request and is interrupted when completed
    - Controllers have *Direct Memory Access* (DMA)
  - Could use special processors called *I/O Processors* (IOPs)

- Instruction Set Architecture
  - *Storage resources*, e.g. memory and registers
  - *An instruction set*
- Register Architecture
  - General-purpose registers
    - Used to hold any operands to instructions
  - Typed registers
    - Such as floating-point registers
  - Special-purpose registers,
    - Program Counter (PC), status registers or stack registers

- **Memory Architecture**

- Defines through an address space
  - Usually 32bit addresses
  - Could be 64bit on newer processors
  - Usually divided between user and kernel
- Flat or segmented address space
  - Flat address space
    - Addresses in load/store instructions represent virtual addresses
    - The MIPS 32-bit ISA has a flat address space, from 0x00 to 0x7FFF FFFF
  - Segmented
    - Addresses in load/store instructions are relative to segments
    - The Intel IA-32 and PowerPC have both a segmented address space



- Memory Architecture

- The Intel IA-32 memory

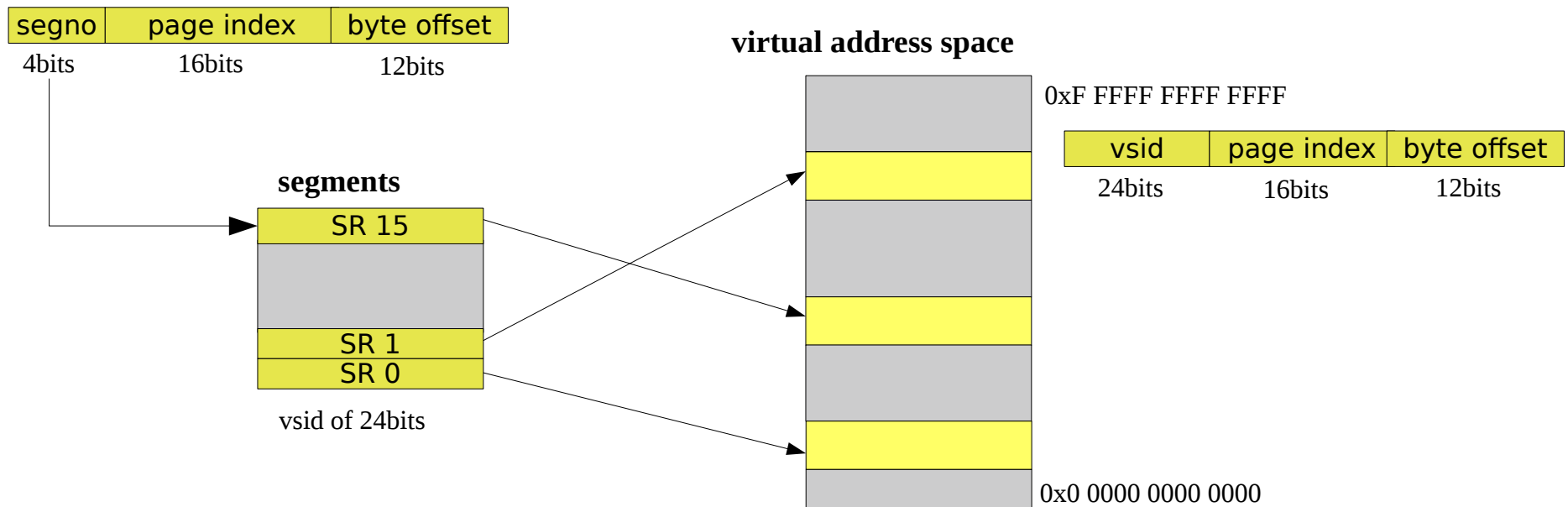
- Virtual addresses are 32bit addresses
    - Supports up to 64K segments, each segment is up to 4 GB
    - Provide only 6 segment registers
      - Hence, at any point in time, only 6 segments are accessible
      - Each load/store instruction specifies a segment and 32bit offsets
        - The offset can be an immediate value or the addition of an immediate value and the content of a general-purpose register

- Could be used as a flat address space

- By setting all segment registers to the same base address
    - Done by both Unix and Windows

- Memory Architecture

- The PowerPC three types of addresses
  - Effective addresses (32-bit address space)
    - Divided into 16 segments of 256MB (28 bit)
    - Top 4 bits index the segment register (SR0 to SR15)
    - Notice that pointer arithmetics may change the segment index
  - Virtual addresses (52-bit address space)
    - But real addresses are 32-bit addresses



- User-level Instruction Set

- A mean of transforming data held in registers and memory
- Instructions are grouped according to what they manipulate

Memory Instructions	Integer Instructions	Floating-point Instructions	Branch Instructions
load byte load word store byte load double load float ...	add compare logical exclusive OR rotate left with carry to-byte or to-long ...	add float add double convert to integer compare double compare float ...	relative branch absolute branch branch if-negative jump to subroutine return ...

- Memory Instructions
  - Load from memory to a register, store a register to memory
  - User-level addresses called *virtual*, *logical* or *effective addresses*
- Integer Instructions
  - Such as arithmetic, logical and shift operations
  - In CISC<sup>(1)</sup> ISAs
    - Addressing mode may be a mix of registers and offsets
    - Arithmetic instructions may involve registers and memory locations
  - In RISC<sup>(2)</sup> ISAs
    - A simpler instruction format
    - Addressing mode may still be a mix of registers and offsets
    - Arithmetic instructions are only on registers

1.Complex Instruction Set Computers (CISC)

2.Reduced Instruction Set Computers (RISC)



- Floating-point Instructions

- Usually refers to floating-point registers
  - The Intel IA-32 uses a stack for floating-point registers
  - Other architectures may use directly accessible typed registers

- Branch Instructions

- Branch instructions change the flow of control
  - Accomplished by changing the *Program-Counter* register
  - Changes where the next instruction is fetched
  - Greatly impacts the pipeline effectiveness
- Different branch instructions
  - Branch instructions may be conditional or indirect (using a register)
  - Branch-and-link, a jump to a subroutine that also saves a return address

- Resource Management

- User-level ISA

- Mostly about getting user tasks done

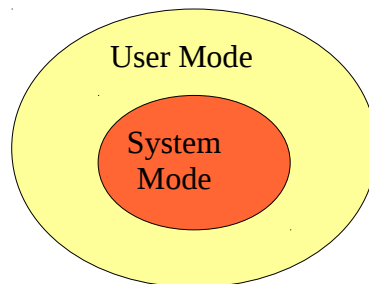
- System-level ISA

- Mostly about management of system resources

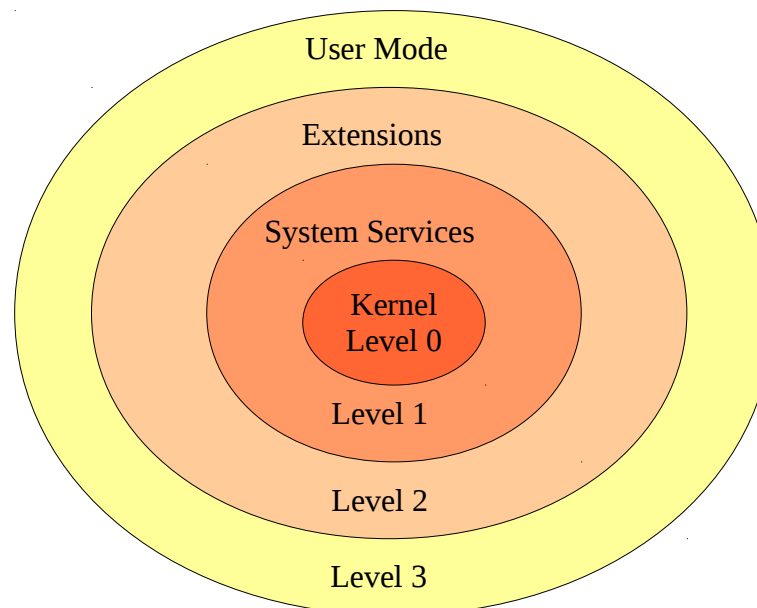
- Process, memory and I/O management

- Require privileges

- *User mode versus system mode (also called kernel or privileged)*



**Most OS relies  
on two levels only**



**Intel IA-32 supports  
up to four levels**

- System-level Registers
  - Most ISAs include special registers
    - To assist with hardware resource management
  - System clock register
    - Records the number of clock ticks elapsed since last reset
  - Trap and interrupt registers
    - Records information about the occurrence of traps and interrupts
    - *Mask register* inhibits or allows traps and interrupts
  - Translation Table Pointers
    - Support virtual address spaces
    - Maps memory pages or segments to real memory

- Processor Management
  - Requires minimal support
    - A *system-return instruction*
      - Jumps to user code
      - Switches to user-level mode
    - Interval timer
      - Getting back the control after some elapsed time
      - Uses an interrupt to switch back to system mode
  - Traps and interrupts
    - Need specific support (mix of hardware and software)

- Processor Management

- Traps and interrupts

- A trap occurs as a side effect of the execution of an instruction
      - Corresponds to *exception conditions* such as arithmetic overflows, page faults or violations of memory-access privileges...
      - The ISA specifies traps on a per instruction basis
    - Interrupts are caused by the occurrence of external events
      - Interrupts are not related to the execution of specific instructions
      - Examples are I/O interrupts or timer interrupts
    - Traps and interrupts may be masked

- Trap-like Instructions

- Some instructions are designed to act as explicit or conditional traps
      - The most important example is the system-call instruction
      - Details about system calls are part of the *Application Binary Interface*

- Trap Handling

- Processor goes into a *precise state* with respect to the trapping instruction
  - All instructions prior to the trapping instruction are completed and make all their specified register and memory modifications
  - Depending on the ISA, the instruction causing the exception either completes (e.g. an overflow exception) or does not cause any change of state (e.g. page faults)
  - None of the instructions following the trapping instructions modify the registers or memory in any way (this is important when having instruction-level parallelism, either pipeline or superscalar)
- The program counter is saved
  - In an ISA-specific location (either register or memory).
  - Some or all of the registers may be saved by the hardware implementation
  - On RISC processors, this is left to the trap- or interrupt-handling software
- The processor is placed in system mode

- Trap Handling

- Control is transferred to a memory location that is specified in the ISA
  - This code may complete the save of the *precise state* of the processor
    - E.g. saves registers if the hardware didn't do it
  - This code may transfer execution to a user-level handler
    - Like in the case of arithmetic overflow
- Upon trap-handling completion
  - Restore the saved *precise state*
  - Jumps back to the location that trapped
    - For most traps, the trapped instruction is re-executed
    - Otherwise, the trapped instruction just completes and the execution proceeds with the next instruction in sequence

- Interrupt Handling

- Interrupts are treated in a manner similar to traps
  - The *precise state* of the processor must be produced
- Some liberty
  - Because it is caused by an external event, there is some liberty in deciding when to treat an interrupt, making the saving of a precise state simpler
- Interrupts may be disabled
  - Some interrupts are not maskable such as *power-failure* or *high-temperature* interrupts



- Memory Management

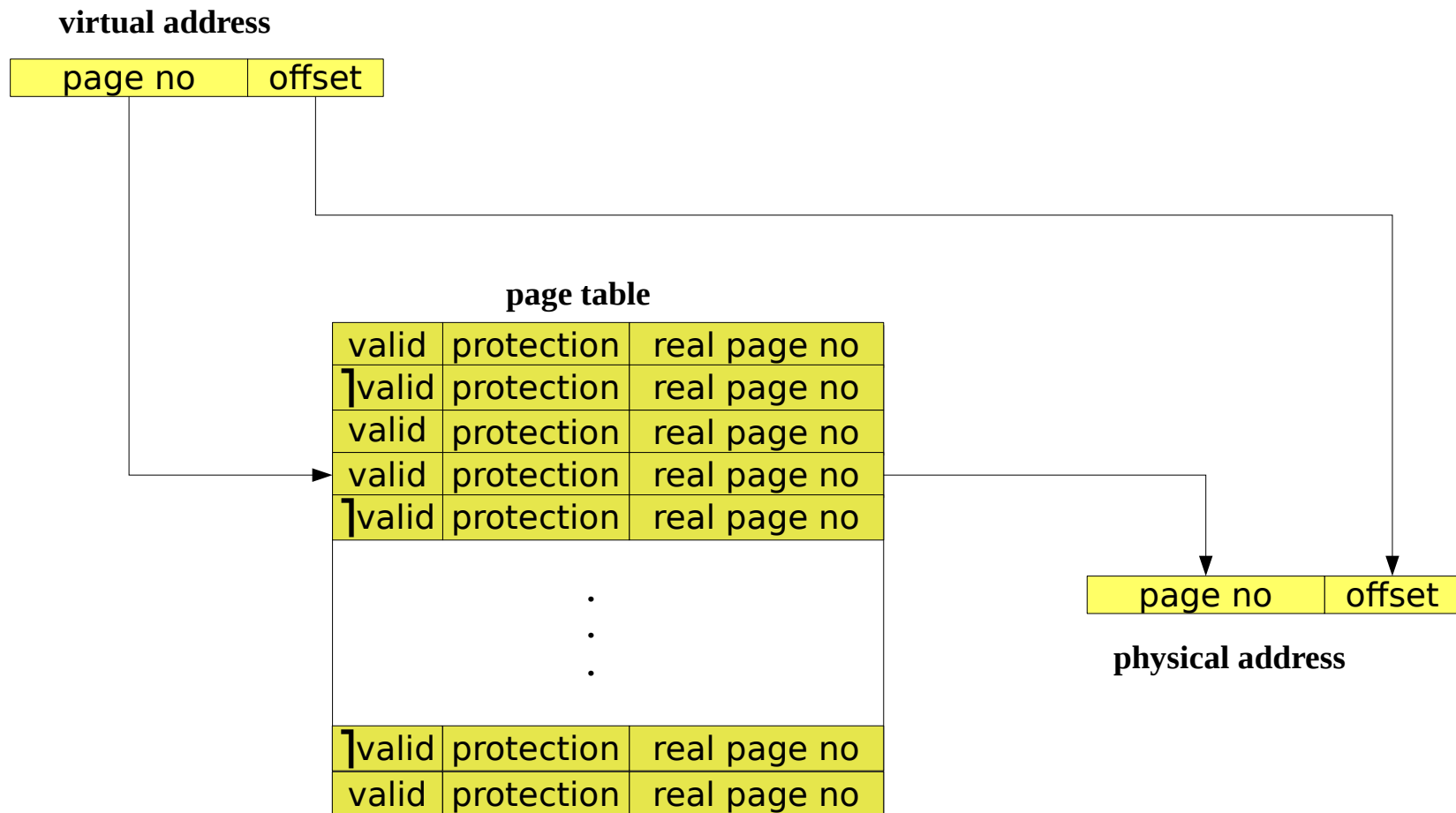
- Goals

- Provide virtual memory larger than physical memory
    - Share physical memory amongst processes
    - Isolate processes
    - Provide fine-grain access protection (read/write/execute)

- Main concepts

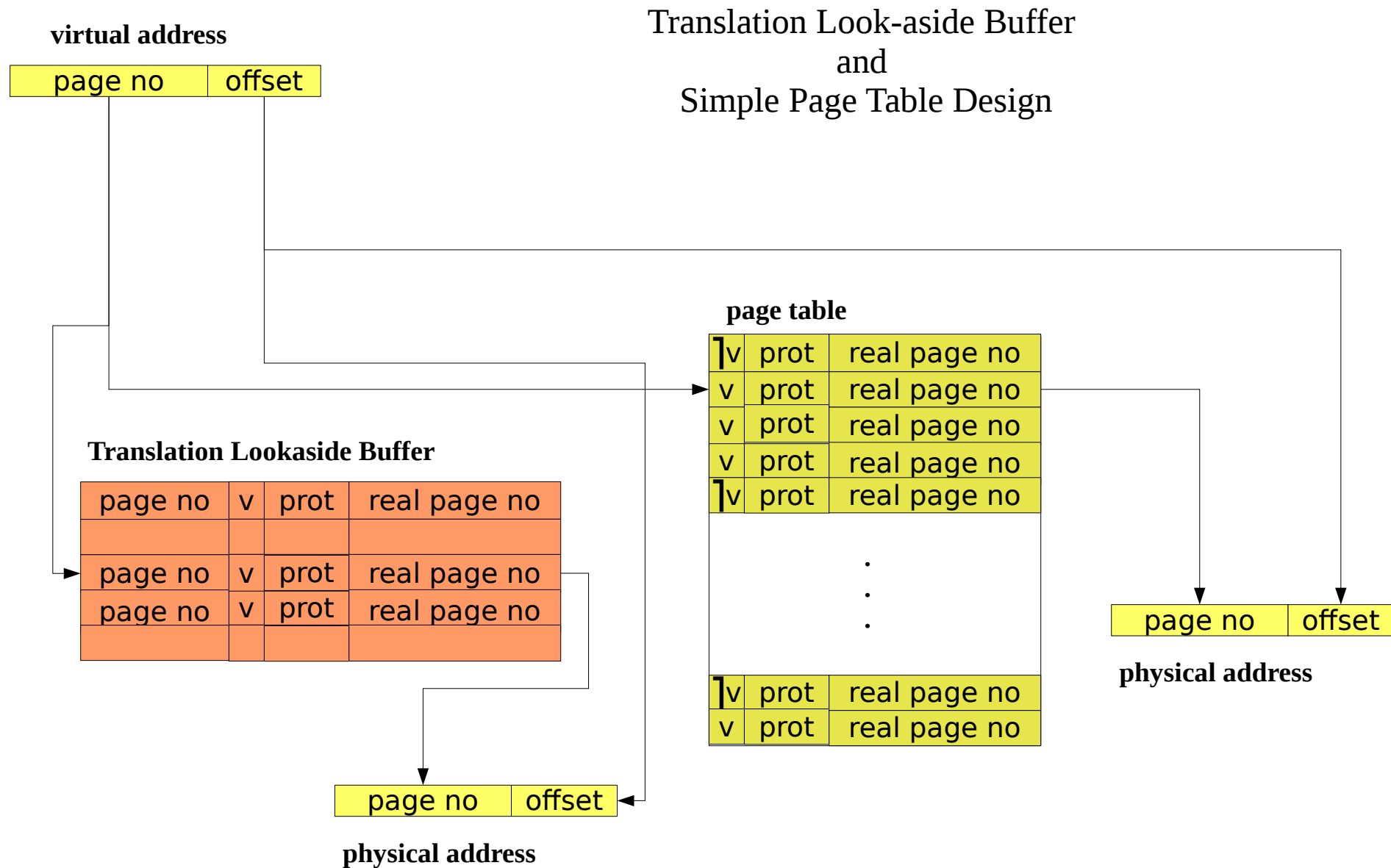
- Page Tables
      - Supports virtual-to-physical memory mapping
    - Translation Look-aside Buffer
      - Small associative cache to speed-up address translation

- Page Tables
  - Supports virtual-to-physical memory mapping
    - One such page table per process
    - Requires a replacement strategy (often Least-Recently-Used)
  - Per virtual page
    - Valid bit (the page is mapped in memory or not)
    - Protection bits (read, write and execute privileges)
    - The page address in physical memory



Simple Page Table Design

- **Memory Management**
  - Translation Look-aside Buffer
    - Small associative cache to speed-up address translation
  - In most architectures
    - A lookup is done in parallel with a cache access
    - Hence, TLBs incur no specific performance penalty
  - TLB caches page protections
    - So that accesses can be checked at the TLB level if there is a hit



- **Memory Management**
  - Mix of hardware and software, the frontier depends on the ISA
- **Architected Page Table**
  - Page table defined in the ISA
    - TLB is in hardware, mostly transparent but for a purge instruction
  - A page table miss is a trap
    - The information about the page fault is defined in the ISA
    - Page table format is defined in the ISA
- **Architected TLB**
  - TLB defined in the ISA
    - Special instructions to read or write TLB entries, a TLB miss is a trap
  - Page table is done in software
    - Without design constraints, the hardware is unaware of the page table
    - Opens the possibility for inverted page tables for large address spaces

- Memory Management

	Architected TLB	Architected Page Table
TLB entry format	Defined in ISA	Left to hardware design
TLB configuration	Defined in ISA	Left to hardware design
Page table entry format	Left to OS implementation	Defined in ISA
Page table configuration	Left to OS implementation	Defined in ISA
Miss in TLB	Causes TLB fault to OS	Hardware accesses page table
Miss in page table	Detected and handled by OS	Causes page fault
New entry in TLB	Made by OS	Made by hardware
New entry in page table	Made by OS	Made by OS

- Examples
  - Architected Hierarchical page tables
    - A page table is organized as a hierarchy
    - The hardware knows how to walk through the hierarchy
    - The software knows how to add/remove/change an entry
  - Inverted page tables
    - A page table is organized as a hash table
    - The hardware knows only how to use the TLB
    - Hashtable lookups and inserts are achieved in software

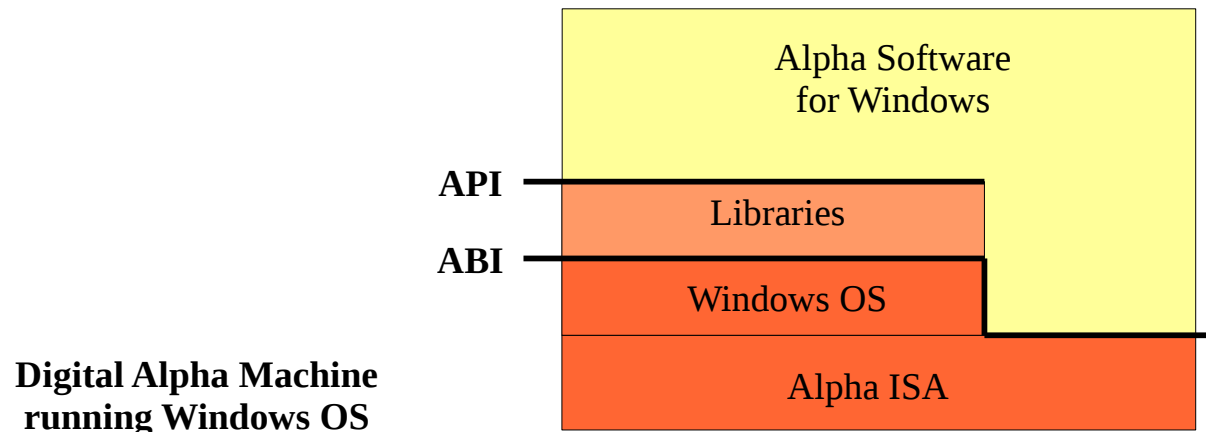


- Input/Output Management

- Some ISA have specific I/O instructions
  - The instructions look like load and store instructions
    - The address identifies the device
    - The value is either data or command
  - Examples: IBM System/360 or the Intel IA-32
- Some ISA have memory-mapped I/O
  - Use regular load and store instructions
    - Not on real memory however, within a special address range
      - The address identifies the device or a special port of a device
      - One device may be mapped at several memory location
    - The read/written value is either data or command
- Interrupts are a part of most I/O architectures
  - A way for getting the attention of the operating system
  - Indicate an external event or the completion of a request

- The foundation

- A process is the foundation of this virtualization
  - A virtual address space, with one or more threads
  - System calls, a way to request a service from the OS
  - Signals for handling traps and interrupts
- ABI versus API
  - Usually, applications do not use the binary interface directly
  - Applications use libraries offering a higher-level programming interface



- Process
  - A virtualized memory space
    - Provides two illusions
      - Owning the entire memory
      - A potentially larger amount of memory
    - Mapping to real memory through a page table
  - Process switching
    - Steps
      - Require to change the page table pointer
      - Flush the TLB, and L1,L2 caches
    - Overhead
      - Memory barrier hit
        - TLB is empty, so page table lookup will occur
        - The content of L1,L2 caches are irrelevant and must be flushed
      - Potential disk barrier hit
        - Pages in memory may not be the one needed

- Threads

- A virtualized execution flow
  - Reified through a Thread Control Block (TCB)
    - A program counter, user-level processor registers
    - A stack pointer for push and popping stack frames
  - Needs a stack
    - A contiguous memory segment for the stack
    - Using memory protection to grow the stack when necessary
- Thread switching
  - Threads are interrupted through the timer interrupt
    - The scheduler is the interrupt handler for the timer
      - The scheduler finishes the save of the thread context
      - It chooses what thread should be next to run
      - Restore the context of that thread, jump back to user-level code
      - Invalid TLB if switching between processes
  - Overhead
    - L1,L2 caches and TLB most likely irrelevant
    - Pipeline stalls (new working set and new locality)

- Signals
  - Signals virtualize ISA traps and interrupts up to user-level code
    - Timer interrupt or overflow trap
    - Memory violation traps (protection violation or non-valid address)
  - Signal handlers
    - Default handlers are provided
    - Applications may redefine them with user-level handlers
    - Through the `sigvec()` system call
  - Signals may be masked
    - Through the `sigblock()` or `sigsetmask()` system call
    - Some signals cannot be masked (SIGSTOP and SIGKILL)
  - Signal occurrences
    - Either because of real traps or interrupts
    - Could be software generated through the `kill()` system call

- System Calls

- A trap in kernel mode
- Carries different service requests
  - Either through values in ISA-specified registers
  - Or through data structures in memory
  - This is all operating system specific
- Different system calls
  - For process management
  - For memory management
  - For Input/Output operations

```
#include <syscall.h>
extern int syscall(int,...);

int file_close(int fileDescriptor) {

    return syscall(SYS_close, fileDescriptor);
}
```

- System Calls

- Process management system calls
  - Create or terminate processes
    - Examples such as Linux `fork()`, `exec()` or `exit()` system calls
  - Other system calls
    - Synchronization ones such as `wait()`, `sleep()` or `wakeup()`
    - Others such as `setpriority()` or `getrusage()`
- Memory management system calls
  - Use the `malloc/free` API, internally uses the `sbrk()` system call
  - Manipulate memory protection through `mprotect()` system call
  - Shared mapped segments through `shmget()` system call

- System Calls

- Input/Output system calls
  - Applications do not directly use I/O instructions or I/O memory
    - I/O memory is not mapped in application processes
    - I/O instructions are privileged
  - Make device-independent system calls
    - Like `open()`, `read()`, `write()`, and `close()`
- Two Device Categories
  - Character devices
    - Direct communication with application code
    - A character at a time
  - Block devices
    - Larger granularity of interactions
    - Data transfers happen through memory buffers



- **Device drivers**

- Execute privileged code
- Implements device-independent system calls in a device-dependent way
- Directly using I/O instructions or load and store instructions in I/O memory
- Responsible of both issuing commands and handling interrupts

