Virtual Machines

Olivier Gruber, Ph.D.

Full-time Professor

Université Joseph Fourier

Laboratoire d'Informatique de Grenoble

Senior Resarcher @ INRIA

Olivier.Gruber@inria.fr

Acknowledgments

- Reference Book

  Virtual Machines
  Versatile Platforms for systems and processes

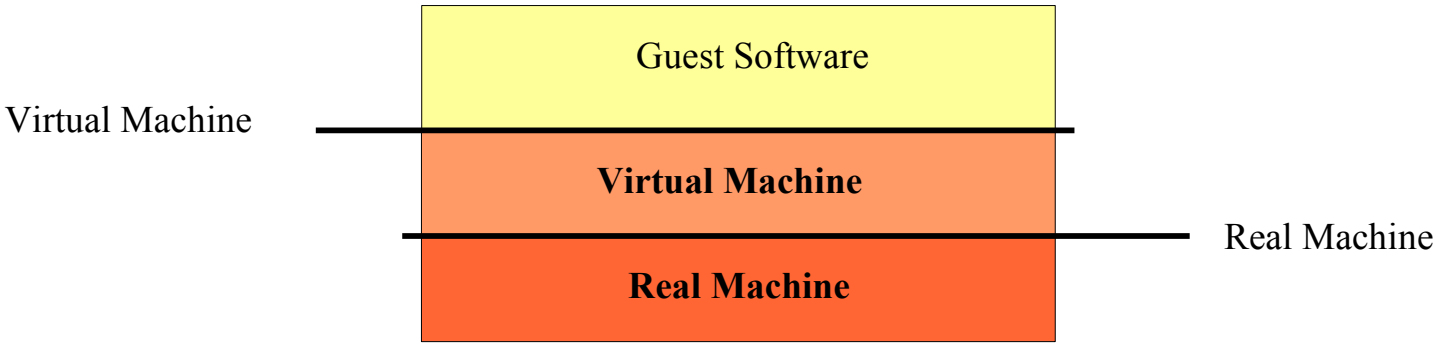
  James E. **Smith**, Ravi **Nair**

  Morgan Kaufmann

- Research Articles
  - Cited on various slides

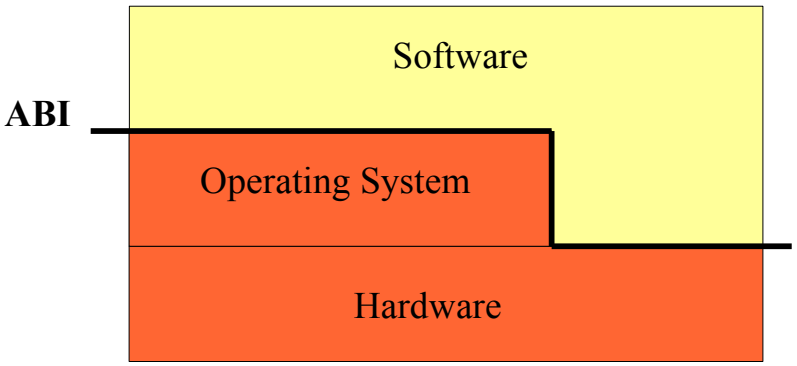
Virtual Machine Basics

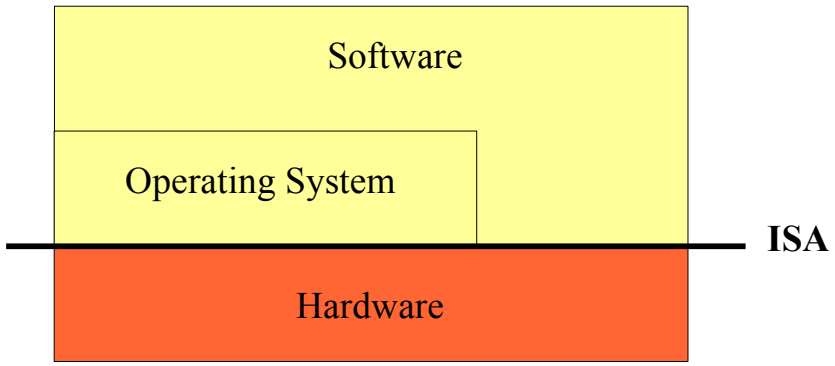
- ***Virtual Machines*** versus ***Real Machines***
  - *A virtual machine defines a machine (interface)*
  - *A virtual machine is a machine (implementation)*



Operating System Basics

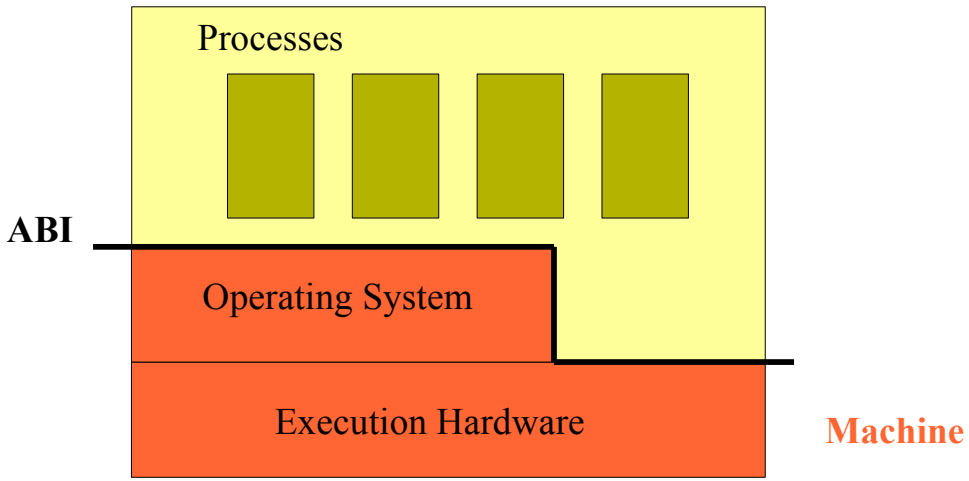
- *Instruction Set Architecture* (ISA)
  - Defines the instruction set
  - Defines other concepts such as memory, traps, interrupts, etc.
- Application Binary Interface (ABI)
  - Defines core concepts above the ISA
  - Example: Linux kernel system calls
    - Related to processes, threads, files, and devices





Operating System Basics

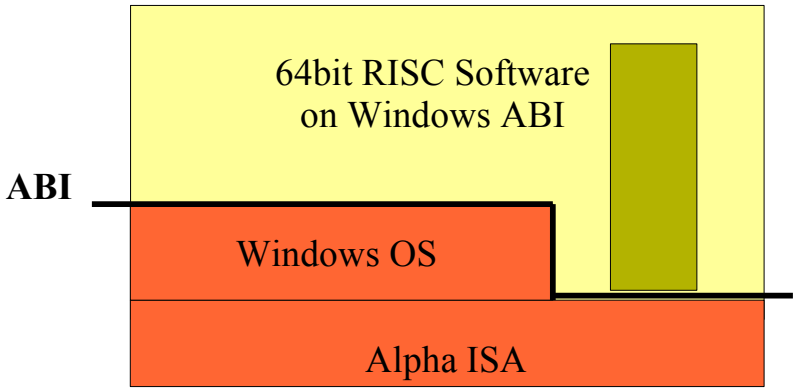
- Operating system design
  - Main goal is sharing hardware resources across processes
  - It provides each process with the illusion it runs alone on a real machine
- Operating systems are virtual machines
  - Subset of the processor instruction set (user-mode)
  - Concepts of the ABI

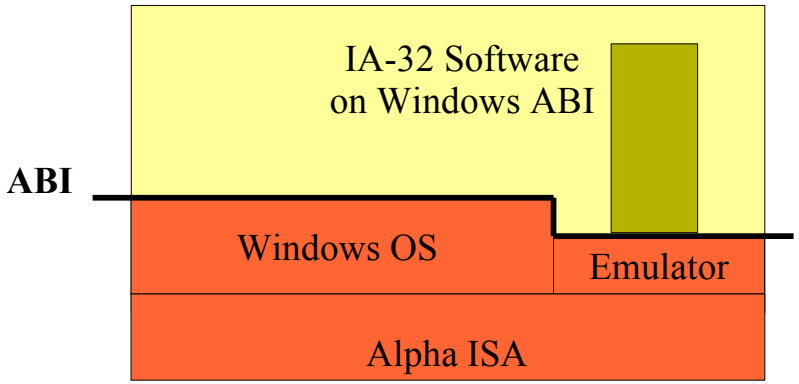


Olivier.Gruber@inria.fr

Dec Alpha Example

- Digital Alpha machine
  - Early provider of a 64bit RISC processor
    - Challenge: no existing software...
  - Support program binaries compiled to a different ISA / same ABI
    - Same ABI: ported the operating system
    - Different ISA: emulate one instruction set on a different instruction set

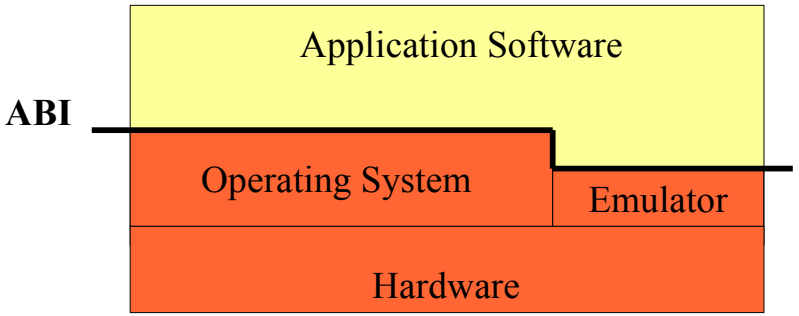




Hookway and Herdeg. *Digital FX!32: Combining Emulation and Binary Translation*.
Digital Technical Journal, January 1997, pp 3-17
Zheng and Thompson. *PA-RISC to IA-64: Transparent Execution, No Recompilation.*
IEEE Computer, March 2000, pp. 47-53

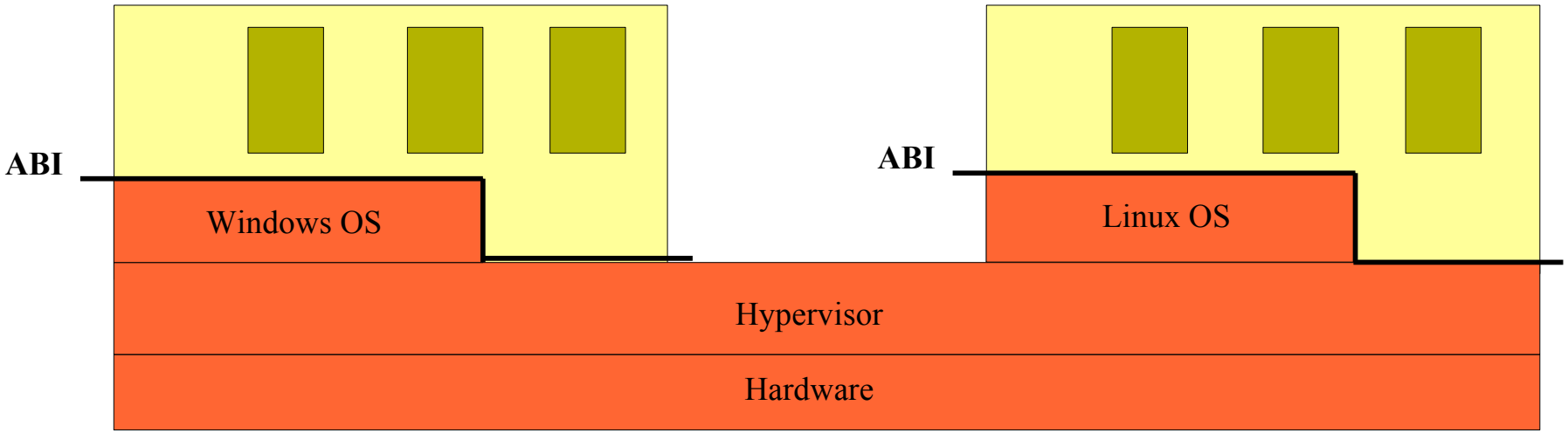
Virtual Machine Basics

- Emulator Designs
  - ***Interpretation:***
    - *Interpretation* of individual guest instructions (fetch, decode and emulate)
    - Easy but slower
  - ***Binary translation***
    - *Binary translation* of blocks of guest instructions to native instructions
    - More complex but fast (close to native performance)
  - Classical trade-off
    - Slow interpretation versus high overhead of binary translation



Virtual Machine Basics

- System Vms (hypervisors)
  - Such as Vmware ESX or Xen
  - Goal: multiplex out-of-the-box operating systems
  - Often virtualize a similar hardware (but not always)

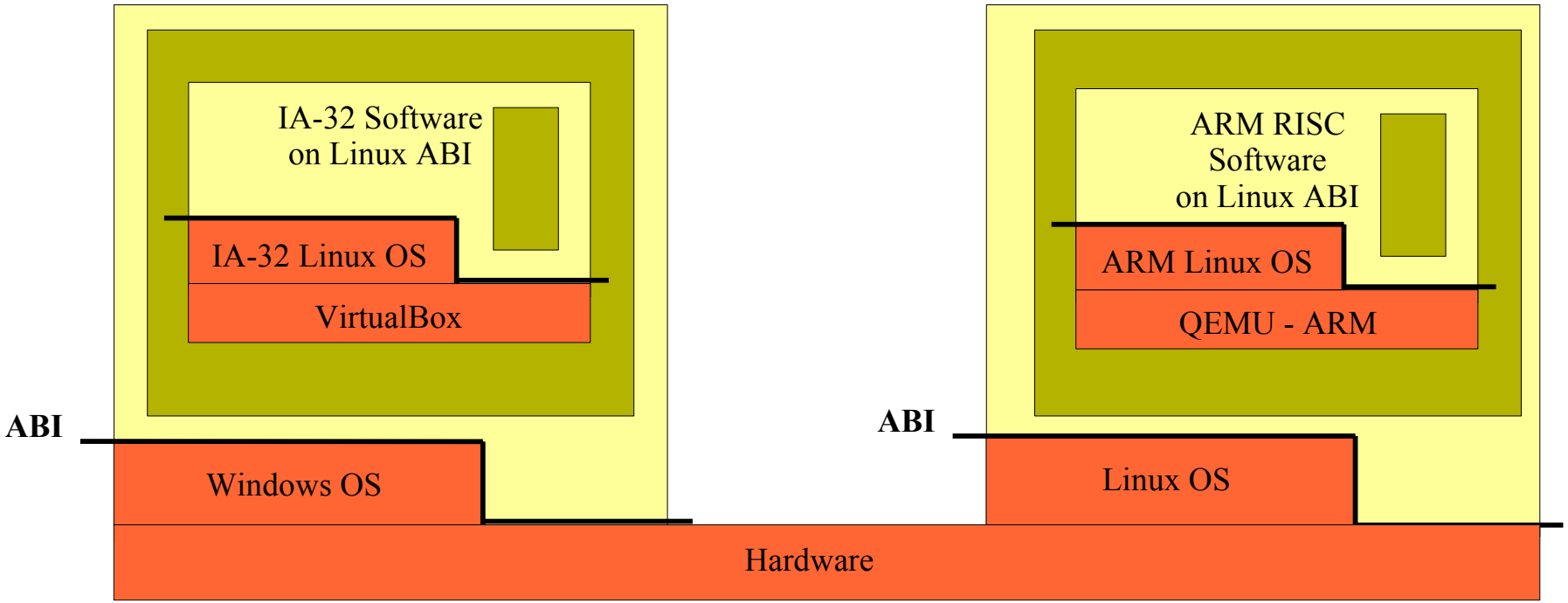


Virtual Machine Basics

- System Vms (hypervisors)
  - Typical use in the Cloud
    - Provides an ubiquitous hardware
    - Provides remote management
  - Virtualize a similar hardware
    - Because performance is critical
    - Same instruction set
    - Similar devices, maybe less memory or less cores
  - Enables hardware sharing to reduce the costs
    - Energy-saving strategy
    - Use a few real machine as necessary
  - The VM has a long life
    - Until the underlying real machine is rebooted

Virtual Machine Basics

- In-Process Virtualization (Process VMs)
  - Virtualizing a real machine within a process
  - Runs one out-of-the-box operating systems
  - Same hardware or not

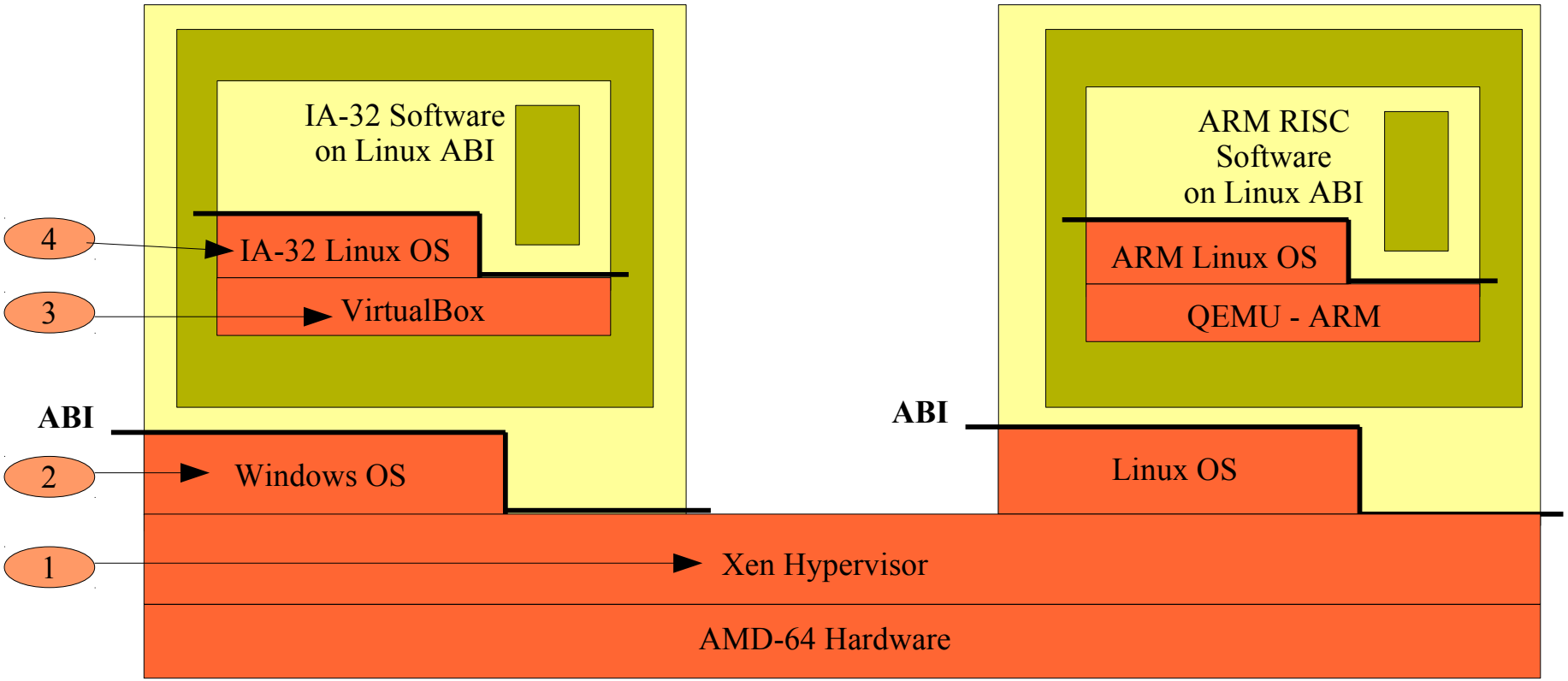


Virtual Machine Basics

- In-Process System Vms
  - Typical uses
    - Kernel development
    - Application availability
    - Help desks
  - Virtualize a similar hardware or not
    - Performance is often not as critical as it is for hypervisors
    - It is the availability of the platform that matters most
    - Similar devices, maybe less memory or less cores
  - The VM has a shorter life
    - Until the process is killed
    - More like an application than a real machine

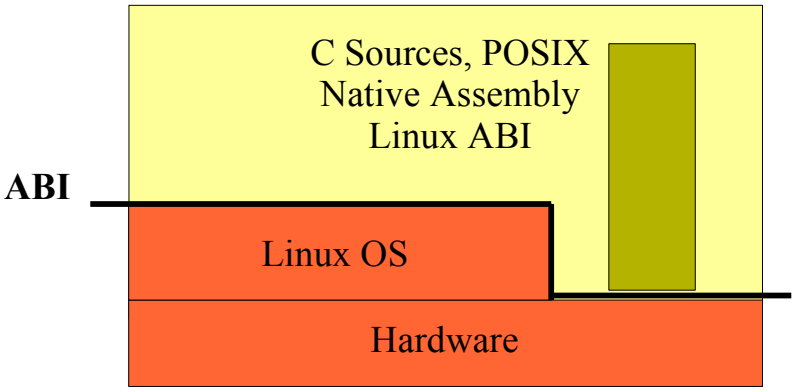
Virtual Machine Basics

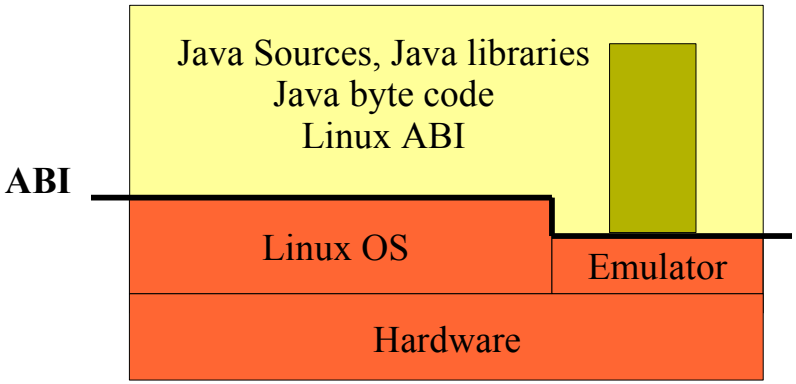
- Virtualization at different layers...
  - Hypervisors, operating systems, and in-process system VMs



Virtual Machine Basics

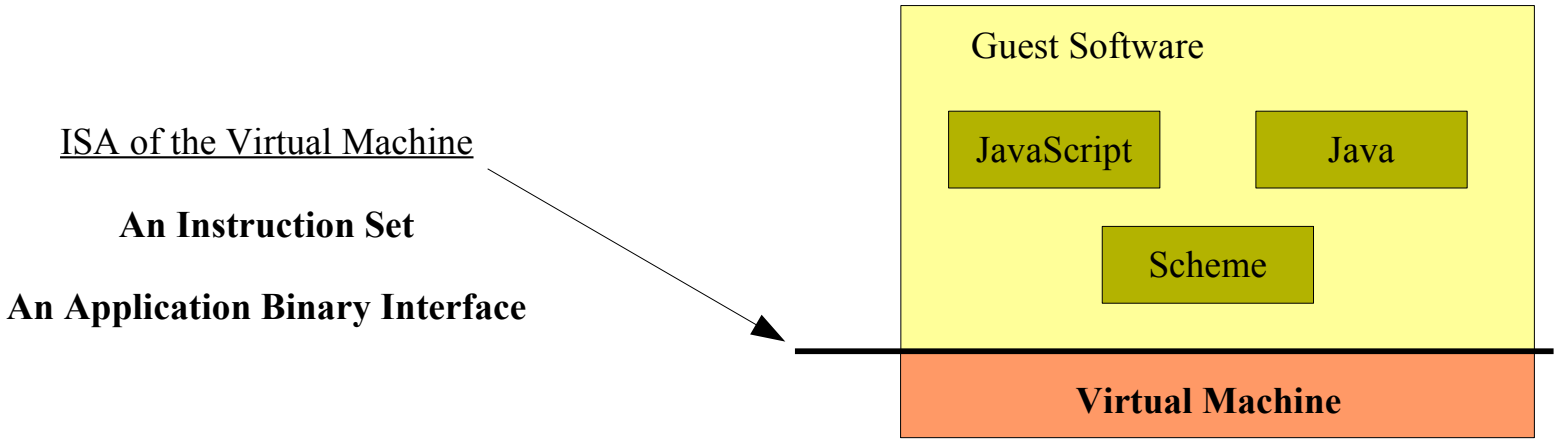
- Process-level Virtual Machines
  - High-level language virtual machines..
  - Examples:
    - Oracle Java or Microsoft C#
    - Eclipse Rich Client Platform (Java and Eclipse libraries)
    - Google Android (Java and Android's libraries)
    - Web applications (Flash or HTML5)





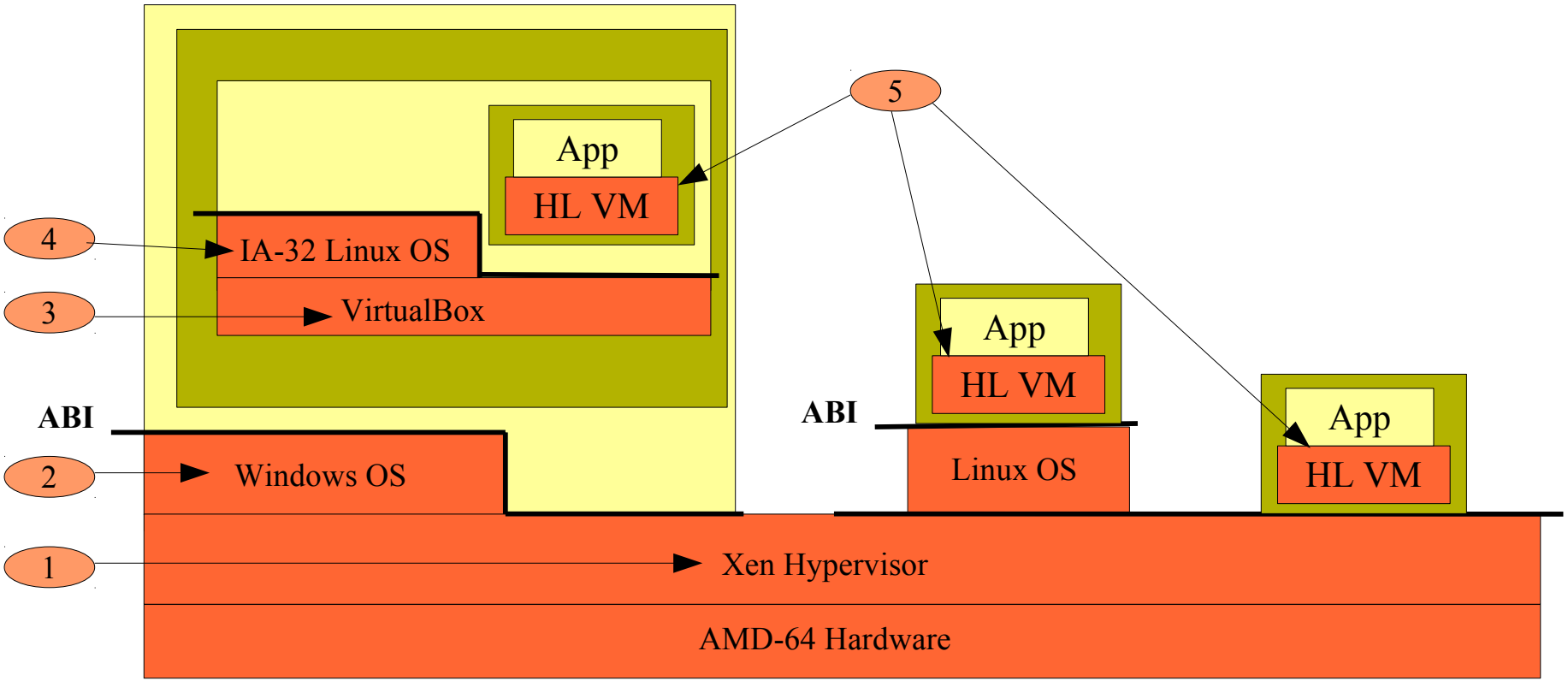
Virtual Machine Basics

- Towards language independence
  - Microsoft Common Language Infrastructure
    - Common Language Runtime (CLR) and Common Type System (CTS)
  - The Java Virtual Machine is going in the same direction
    - Already a target for many languages (JavaScript, Scheme, Perl, Python, etc.)

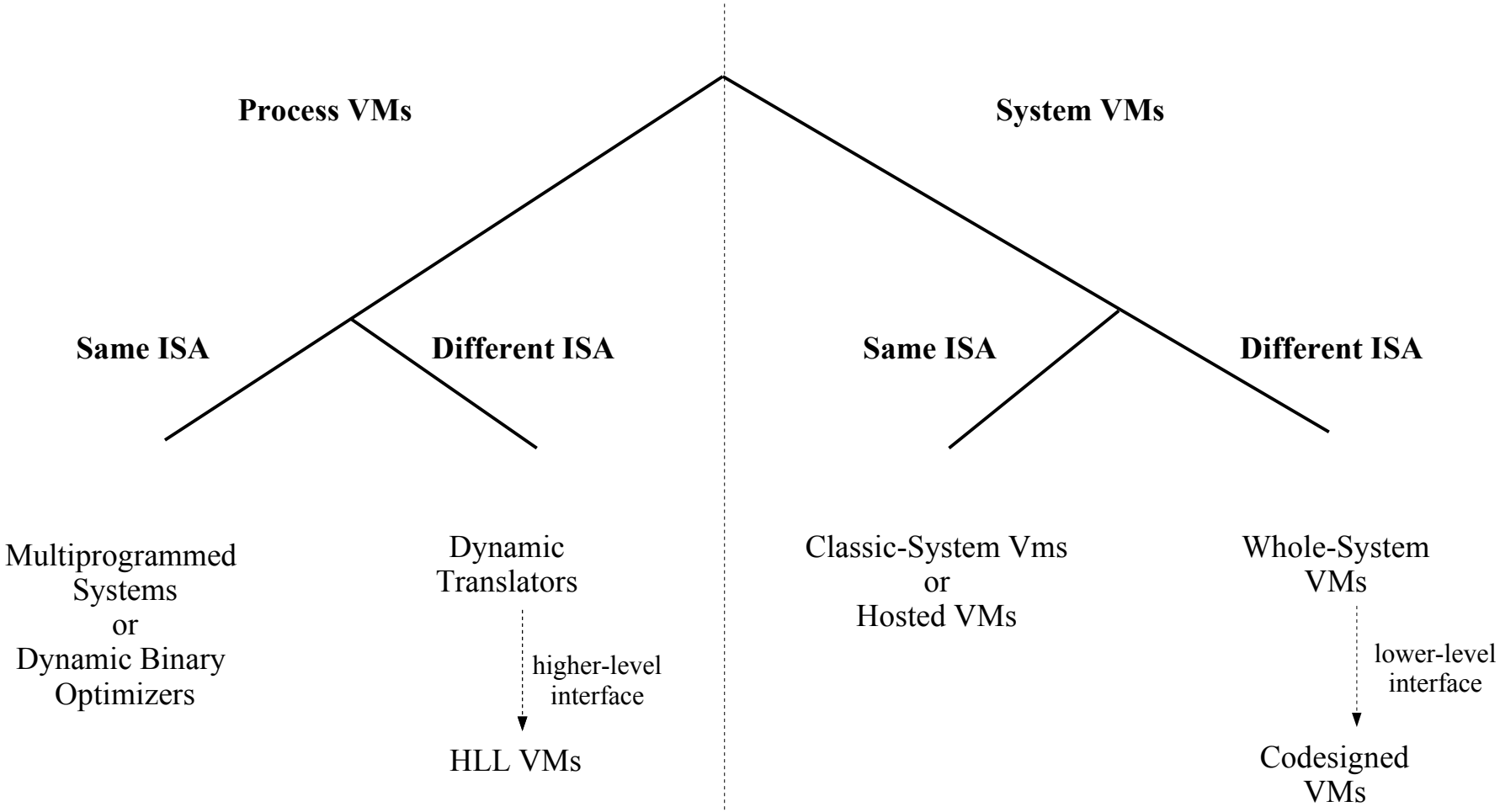


Virtual Machine Basics

- Virtualization at different layers...
  - Hypervisors, operating systems, and in-process system VMs
  - Adding high-level language VMs



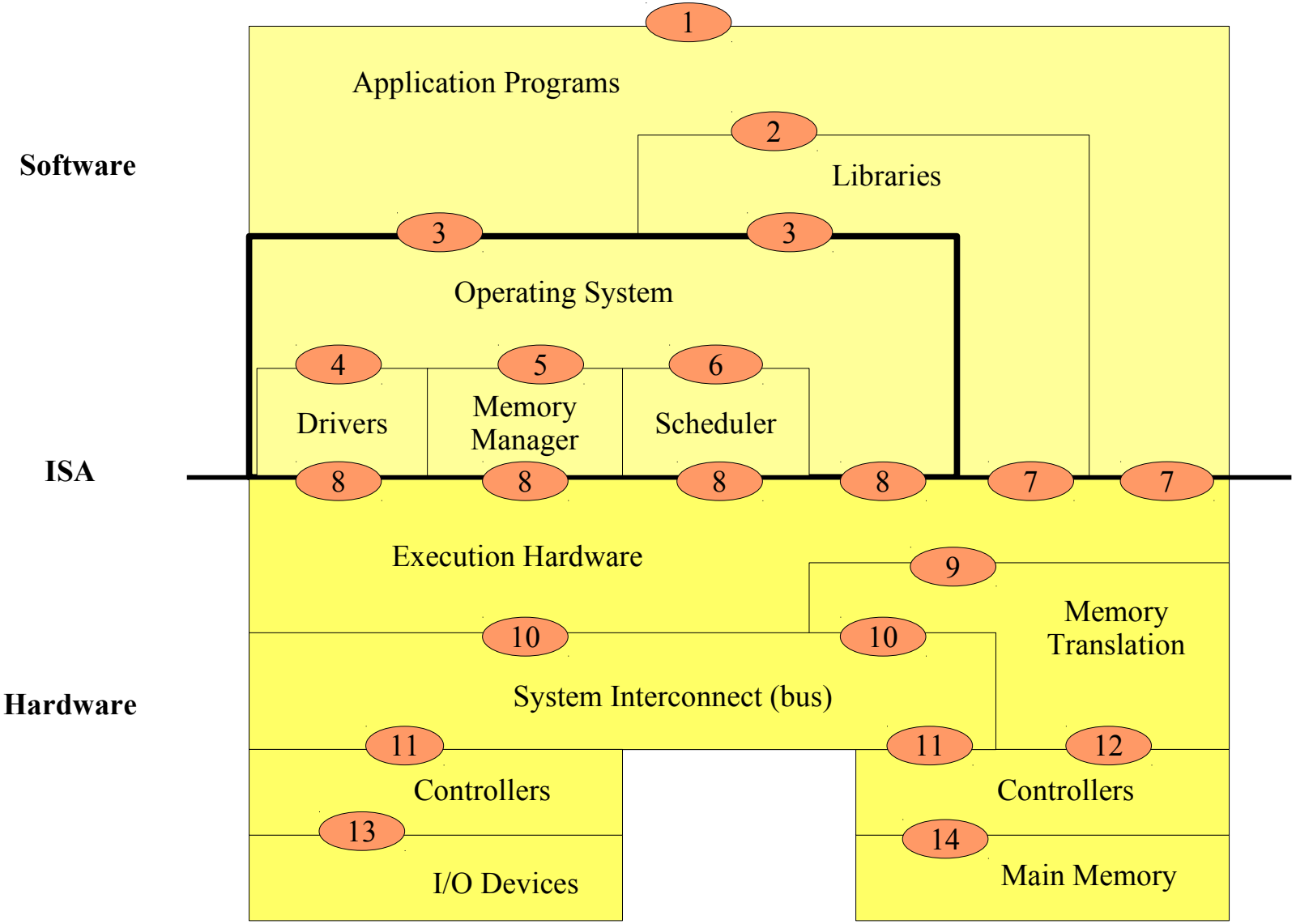
Virtual Machine Taxonomy



Real Machines

- 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

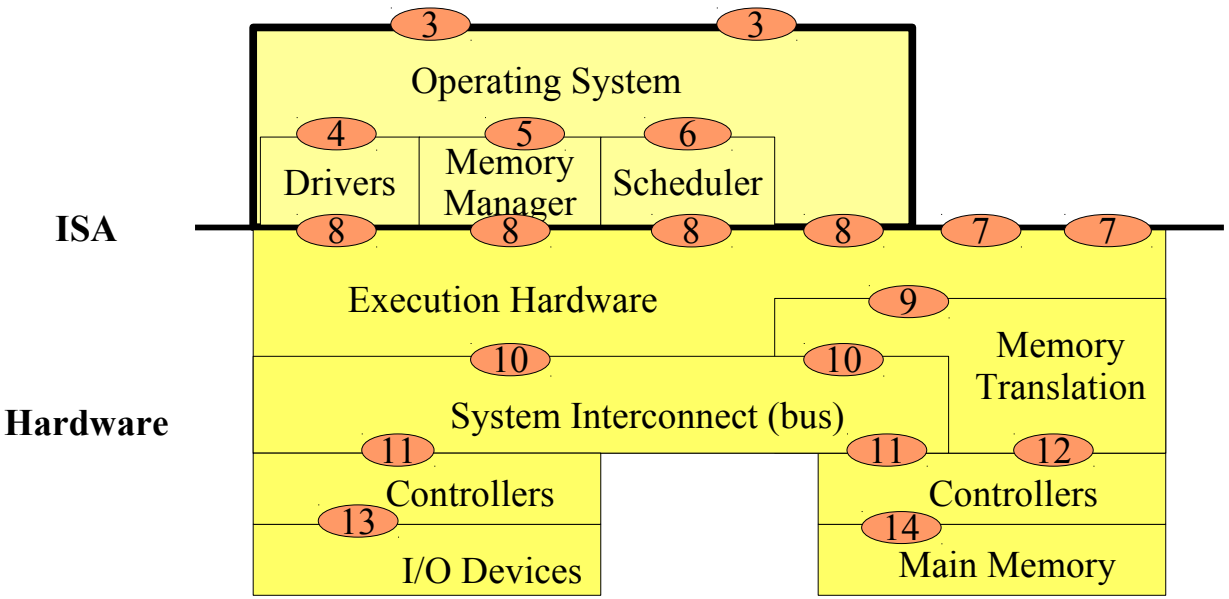
Computer System Architecture



Olivier.Gruber@inria.fr

Instruction Set Architecture

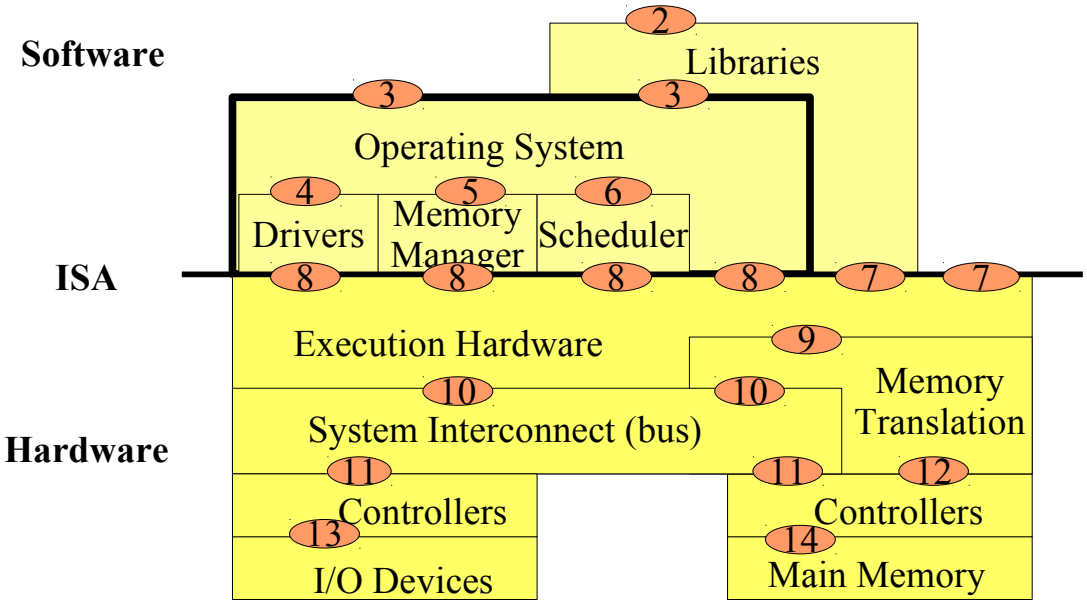
- Instruction Set Architecture has two parts
  - User-level ISA 7
    - Aspects that are visible to non-priviledged code
  - System-level ISA 8
    - Aspects that are visible to priviledged code
    - Of course, the system-level ISA includes the user-level ISA



Olivier.Gruber@inria.fr

Application Interface

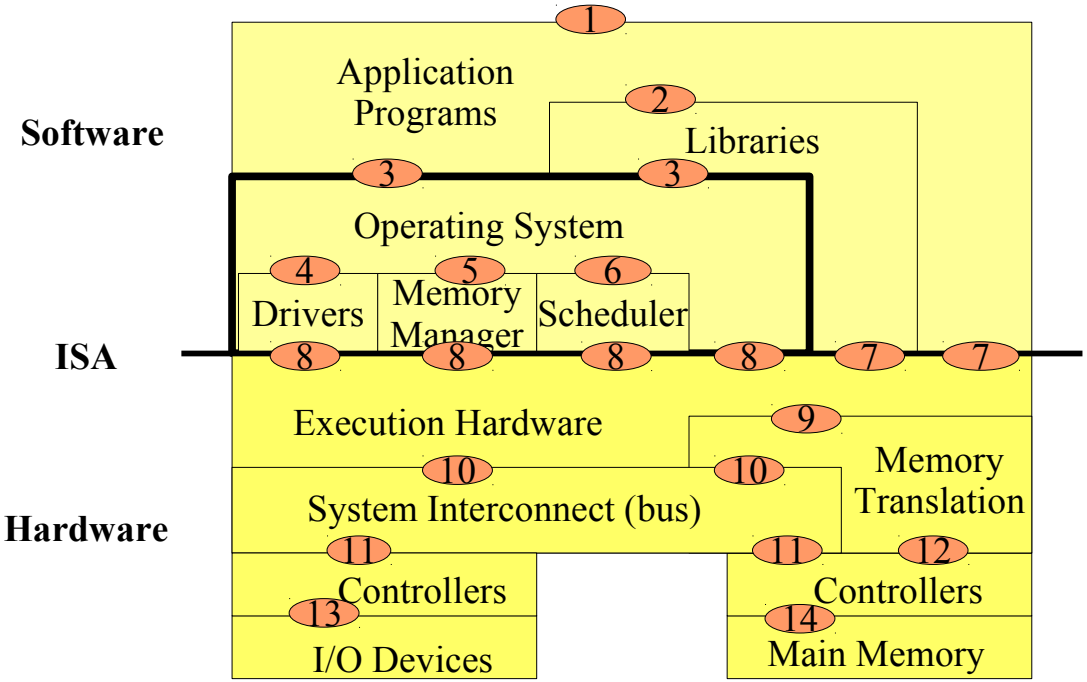
- Application Binary Interface (**ABI**)
  - User-level ISA 7
    - Aspects that are visible to non-priviledged code
  - System-call interface 3
    - Provide indirect access to shared resources
    - System calls use a trap mechanism to priviledged code in the OS
    - Each operating system specifies how parameters are passed



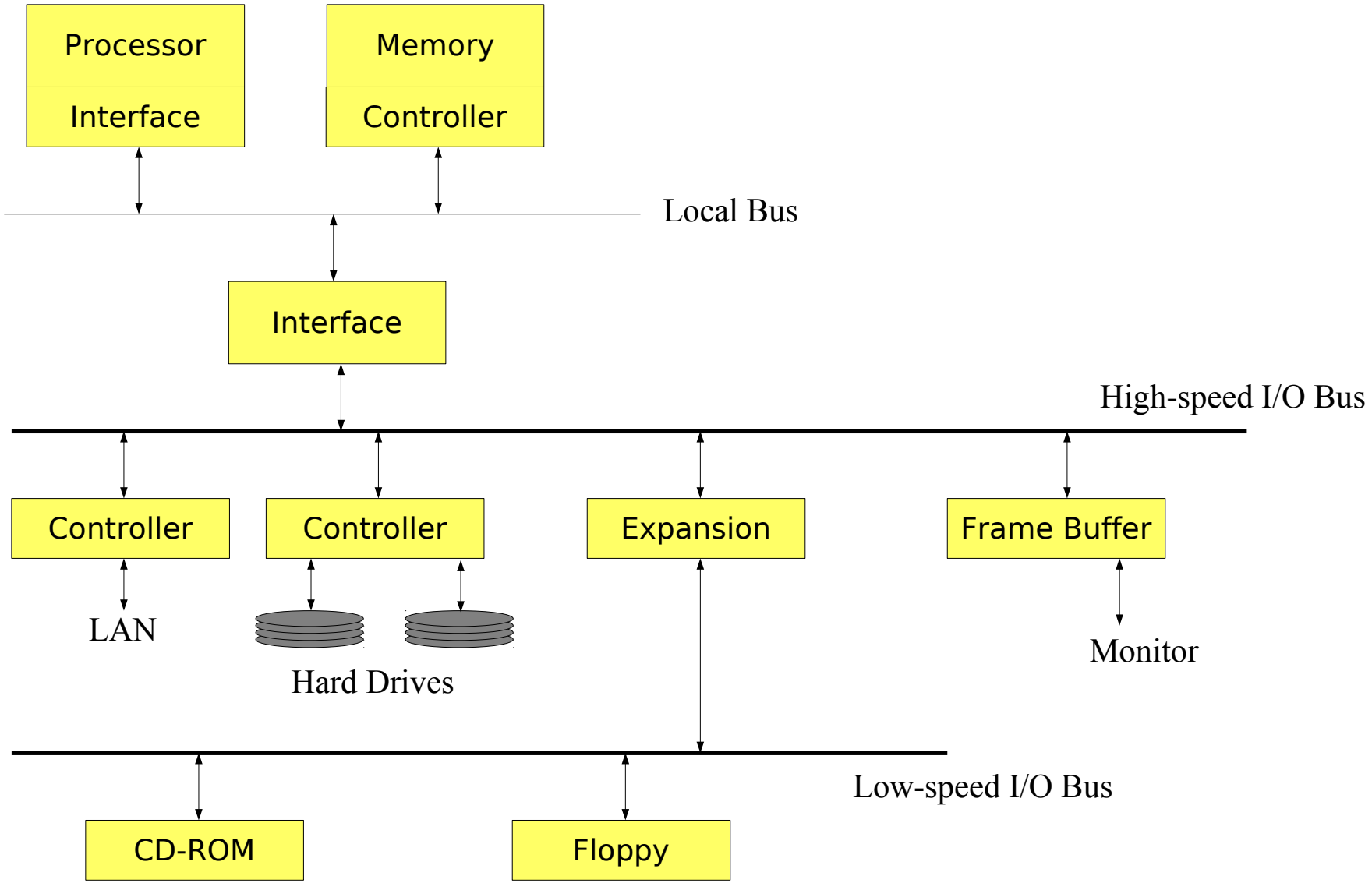
Olivier.Gruber@inria.fr

Application Interface

- 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



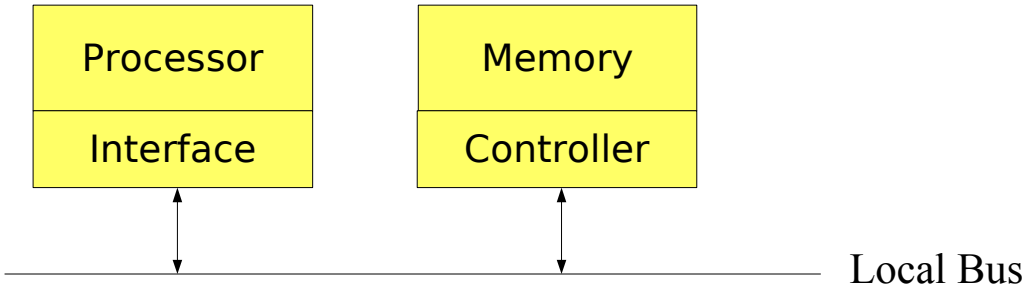
Hardware Architecture



Olivier.Gruber@inria.fr

Processor

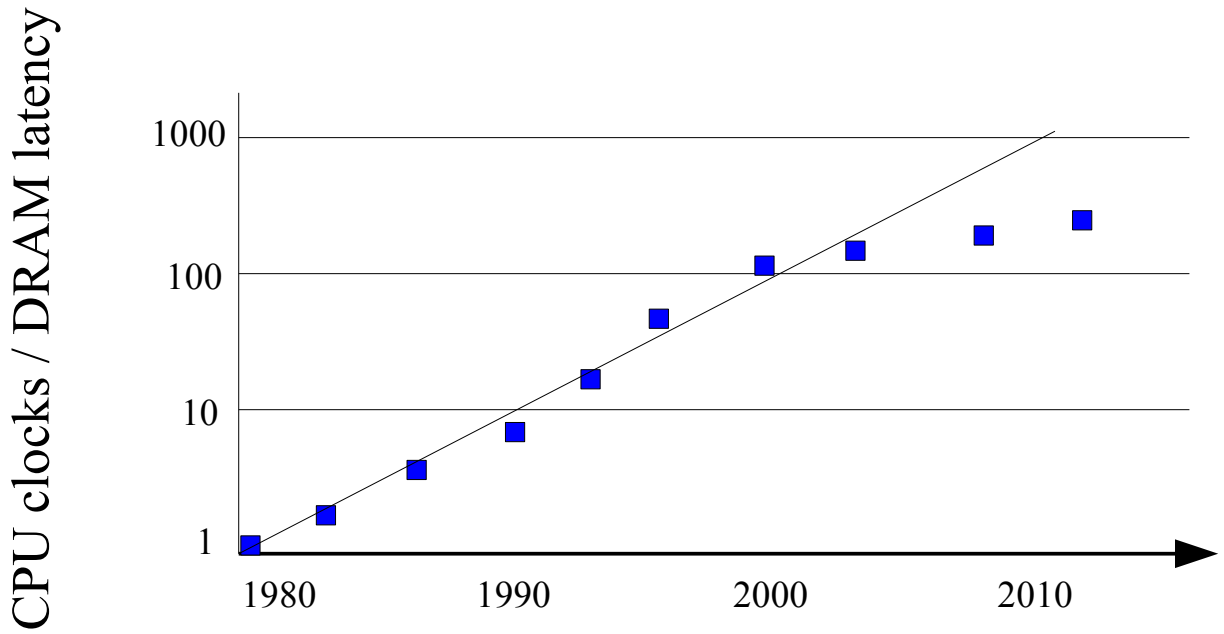
- Basic processing
  - Fetch, decode, and issue instructions
  - You could write a simple interpreter...
- CISC(1) or RISC(2)
  - Complex instructions versus simpler instructions
  - RISC = ''*Relegate Interesting Stuff to Compilers*''
  - Sometimes CISC outside, but RISC inside...



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

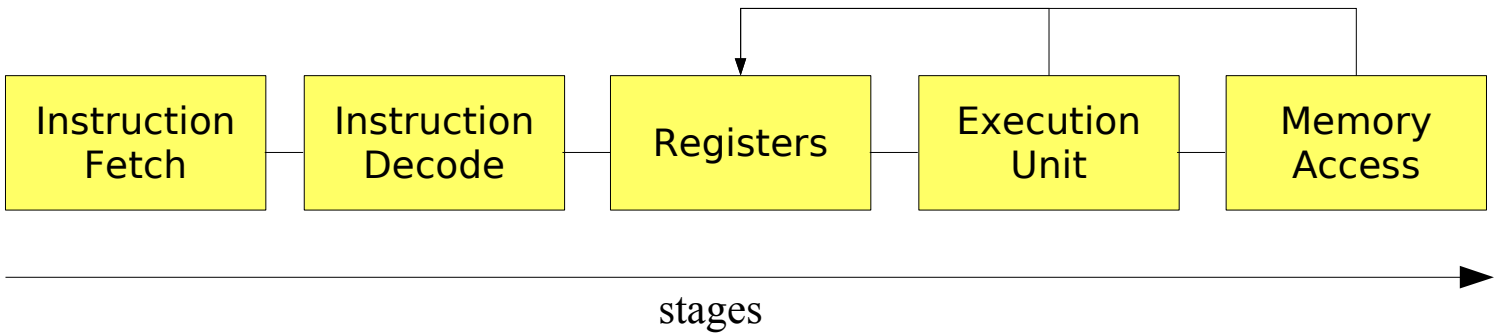
Processor

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



Processors

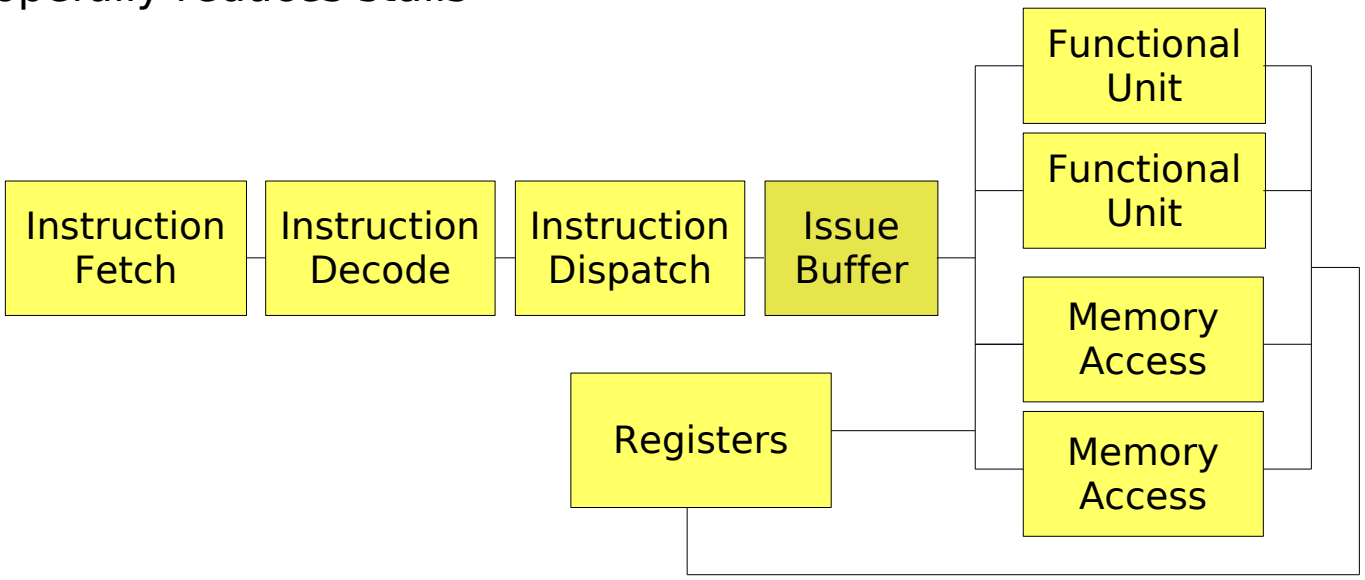
- Different types of processors
  - In-order pipeline
  - Superscalar
  - Very Long Instruction Word (VLIW)
- In-order pipeline
  - Multiple instructions may be in the pipeline at the same time
  - Only one instruction is in each stage at any given time
  - Stalls happen when instructions must wait for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



Olivier.Gruber@inria.fr

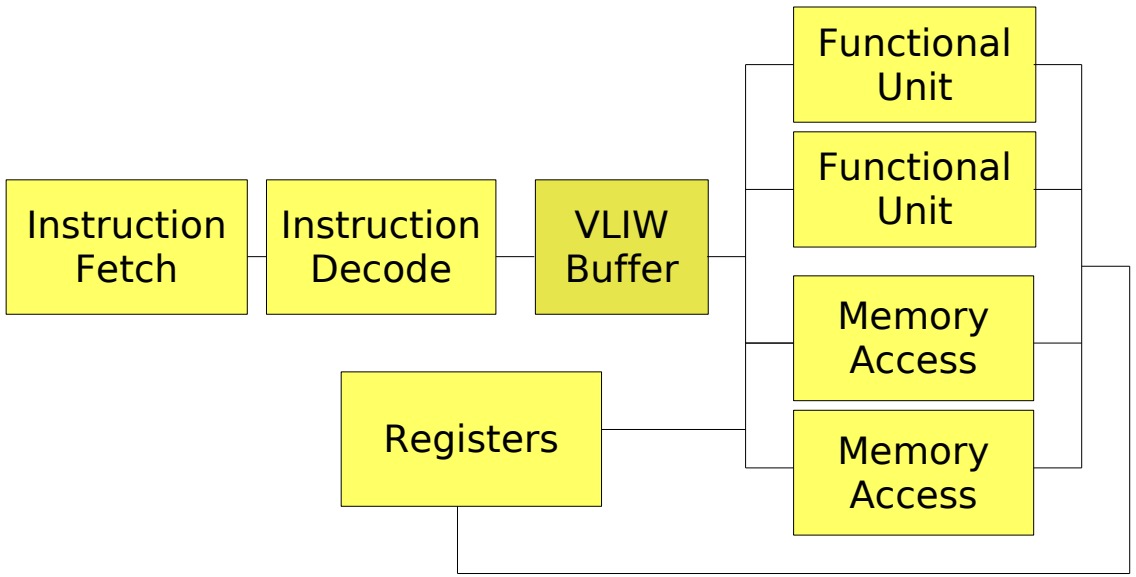
Superscalar Processors

- High-performance processors
  - Introduce automatic instruction-level parallelism
    - Several instruction 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



VLIW Processors

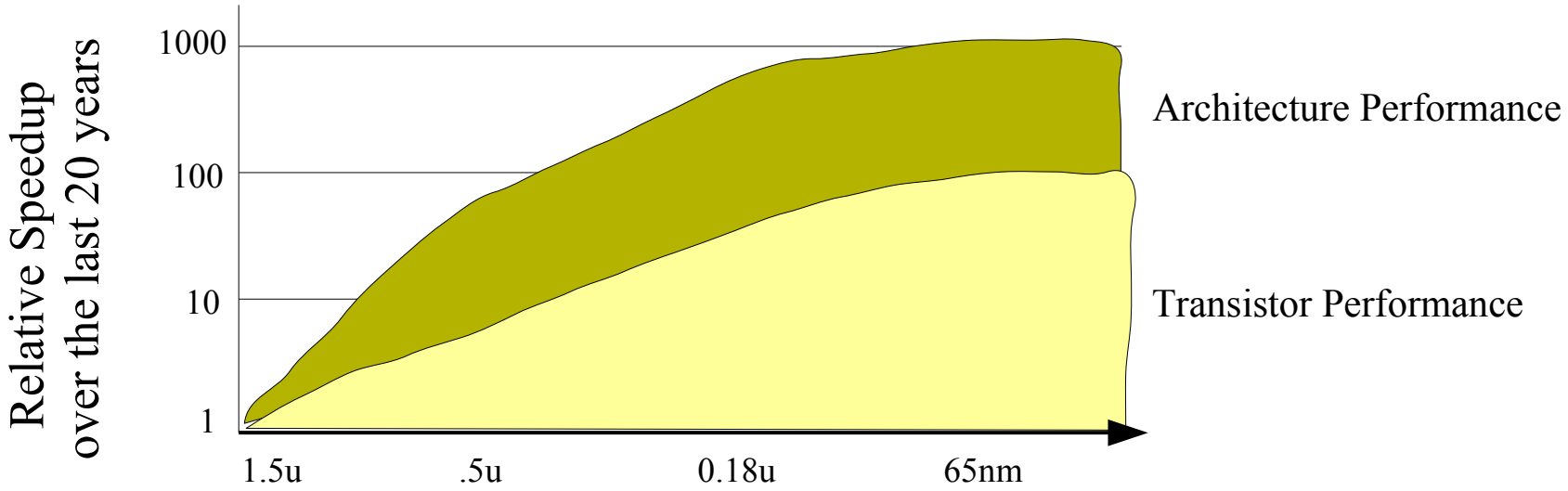
- Compilers must produce VLIW
  - Combine parallel intructions into a very long instruction word (VLIW)
  - VLIW are executed in sequence
- VLIW parallelism
  - A VLIW can be fetched and decoded in the same clock cycle
  - The instructions of the VLIW proceed in parallel
  - Begin execution when their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

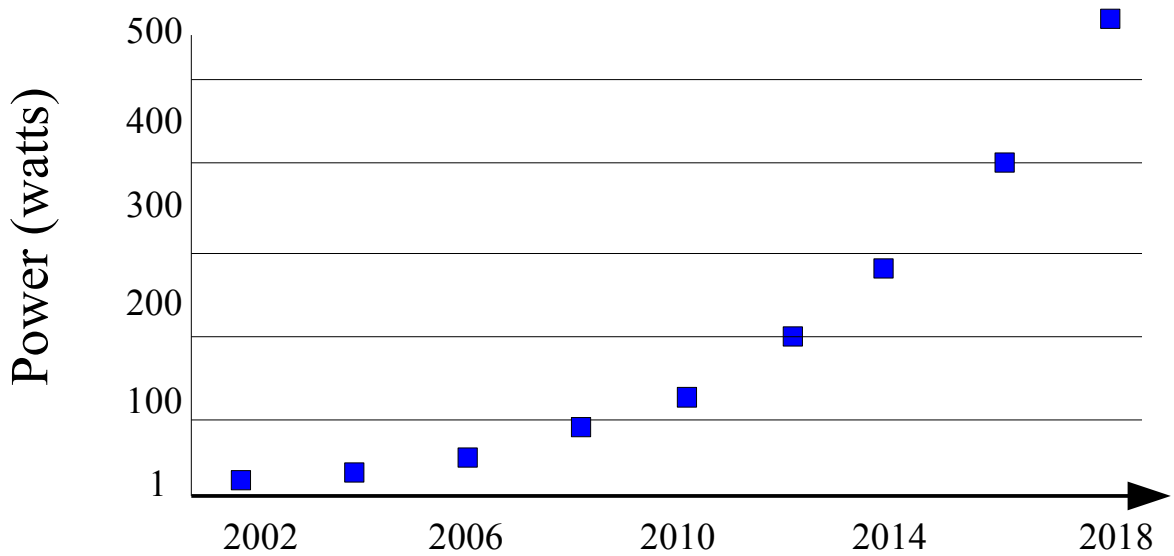


Hyper-threaded Processors

- 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 all what it needs to context switch...

Discussing the Limits

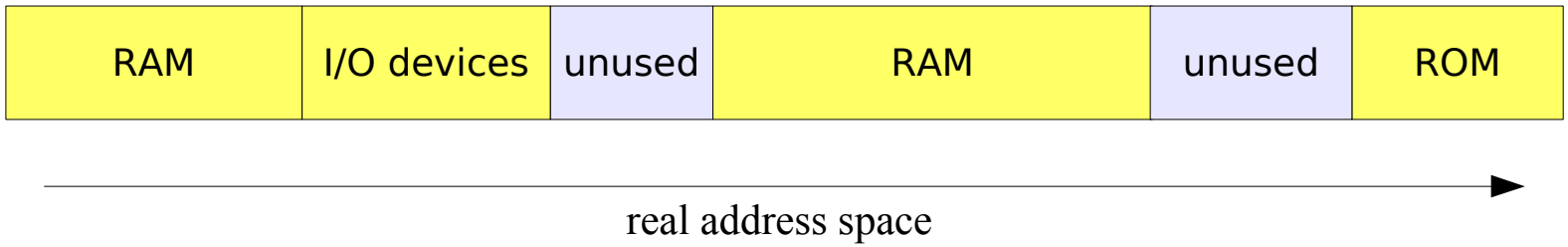




Olivier.Gruber@inria.fr

Memory

- 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)

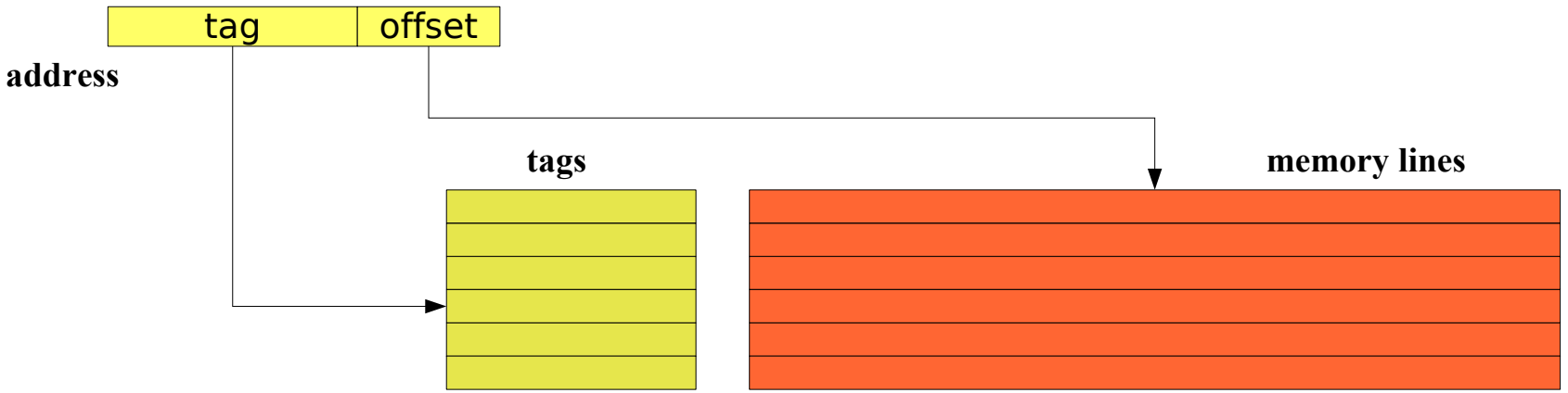


Memory

- 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

Memory

- 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



Input/Output Systems

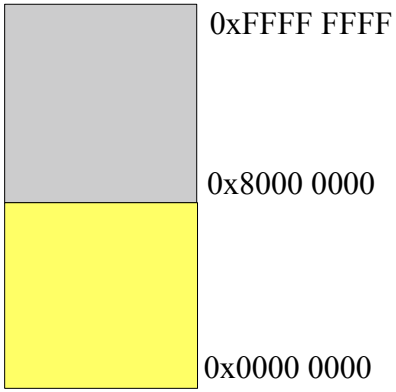
- Architecture
  - Consist of a number of buses
    - That connect the processor and memory to I/O devices
    - Such buses are often standardized (PCI or AGP)
    - Devices often use a controller to connect to such buses
  - A bus is a conduit for device commands and for data transfers
- Different Designs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

User-level ISA

- 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

User-level ISA

- 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 is 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 are both a segmented address space

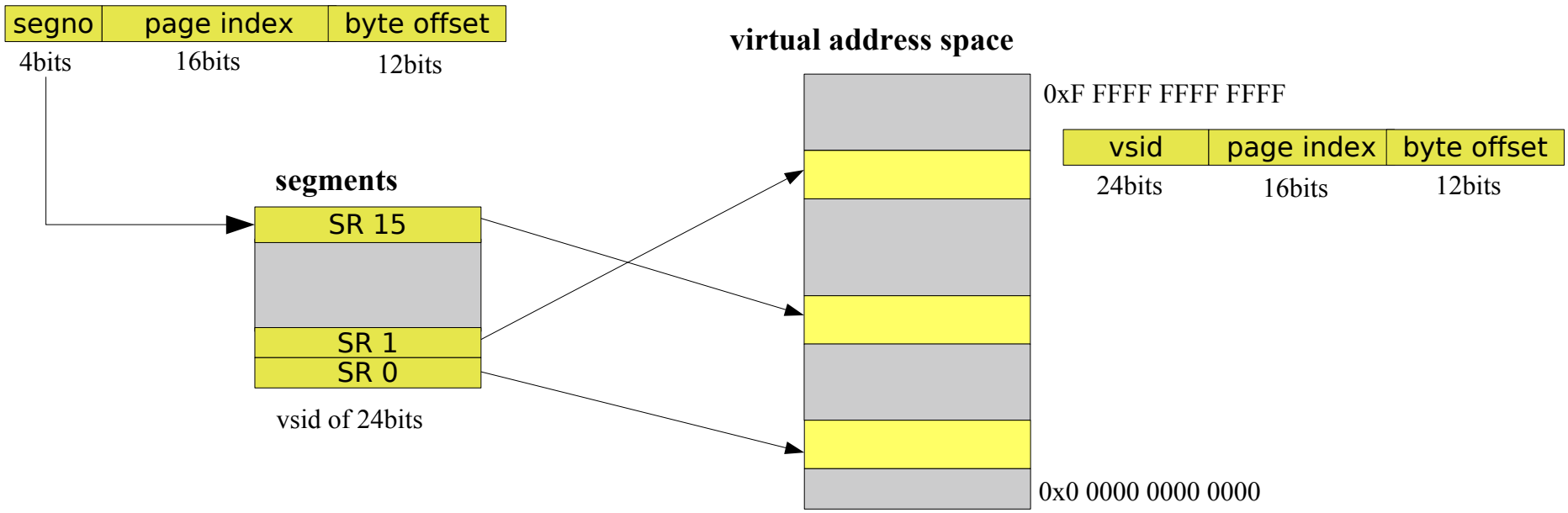


User-level ISA

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

User-level ISA

- 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 ISA

- 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 ...

User-level ISA

- 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(1) ISAs
    - Addressing mode may be a mix of registers and offsets
    - Arithmetic instructions may involve registers and memory locations
  - In RISC(2) ISAs
    - A simpler instruction format
    - Addressing mode may still be a mix of registers and offsets
    - Arithmetic instructions are only on registers

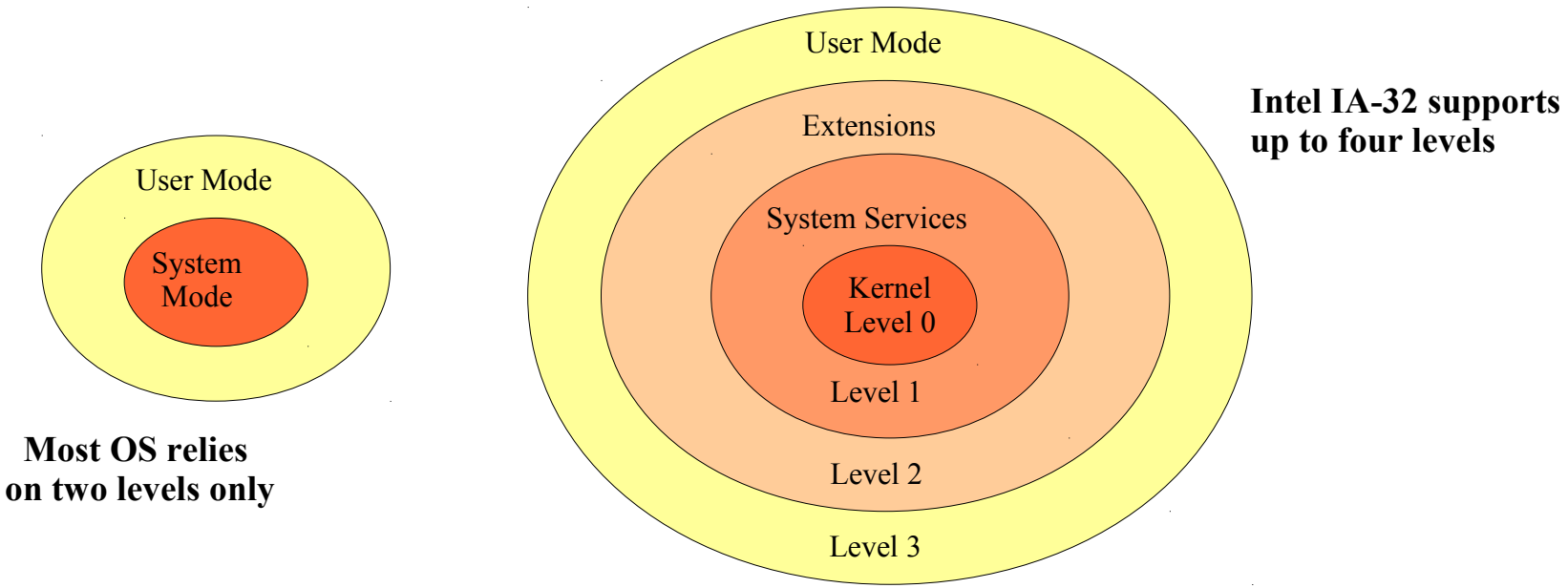
1.Complex Instruction Set Computers (CISC)
2.Reduced Instruction Set Computers (RISC)

User-level ISA

- 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

System-level ISA

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



System-level ISA

- 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 occurence 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

System-level ISA

- 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)

System-level ISA

- 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 priviledges...
      - The ISA specifies traps on a per instruction basis
    - Interrupts are caused by the occurance 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*

System-level ISA

- 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, superscalar or VLIW)
  - 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

System-level ISA

- 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-handing 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

System-level ISA

- Interrupt Handling
  - Interrupts are treated in a manner similar to traps
    - The *precise state* of the processor must be produced
    - For some interrupts, an incomplete state may be acceptable
  - 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

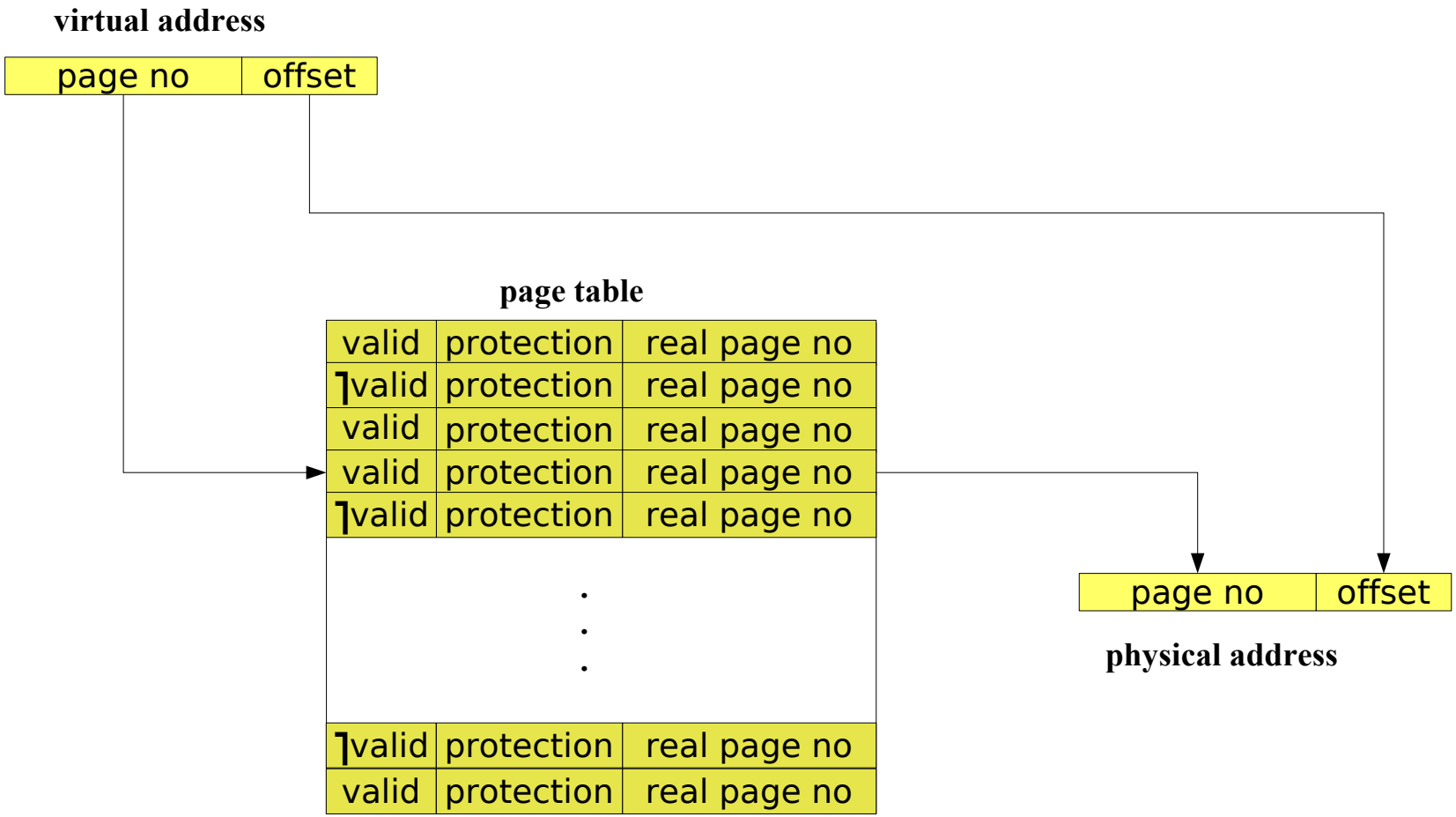
System-level ISA

- 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 Lookaside Buffer
      - Small associative cache to speed-up address translation

System-level ISA

- 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 priviledges)
    - The page address in physical memory

System-level ISA

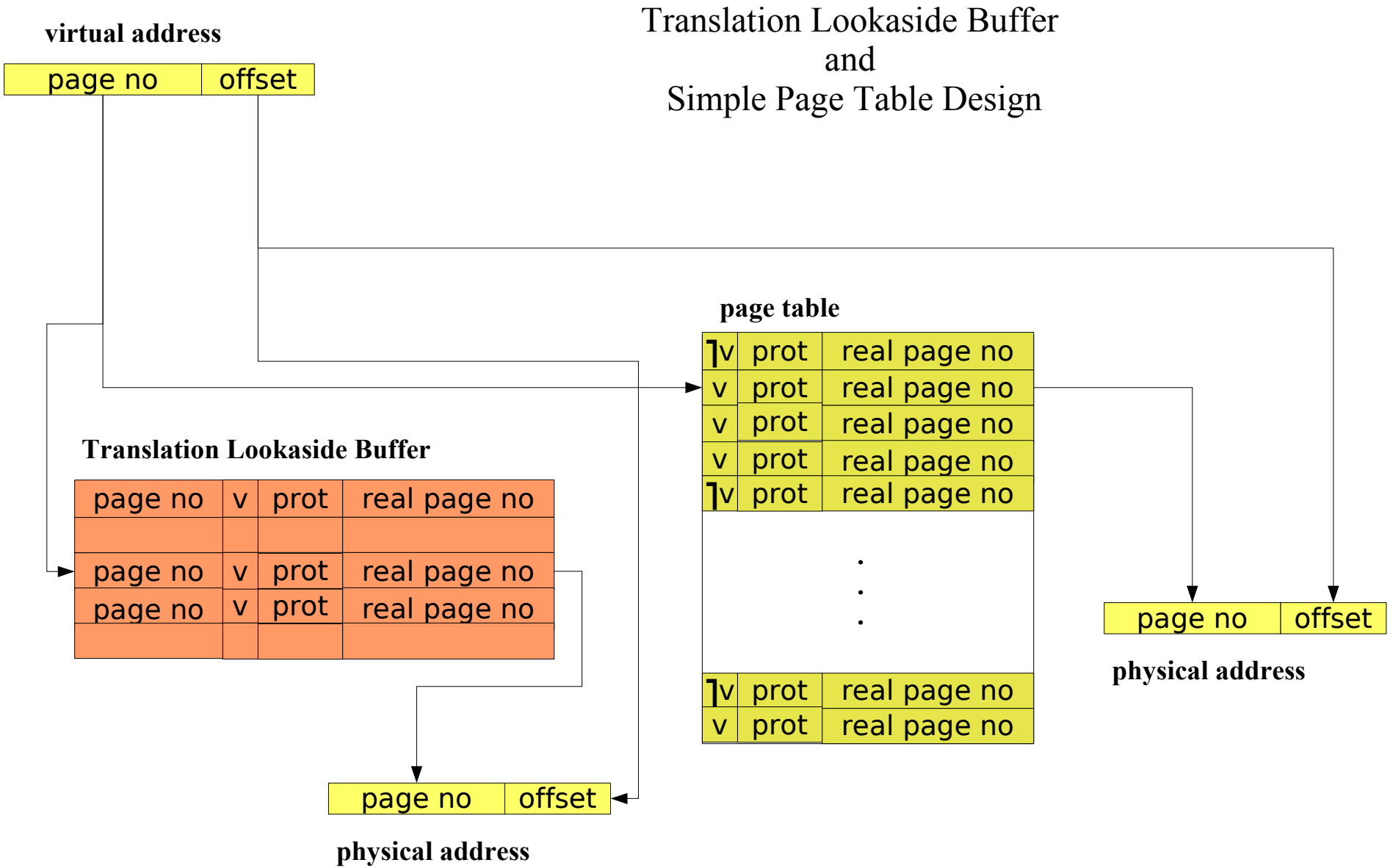


Simple Page Table Design

System-level ISA

- Memory Management
  - Translation Lookaside 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
  - Caches page protections
    - Access are checked at the TLB level if there is a hit

System-level ISA



System-level ISA

- 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

System-level ISA

- Memory Management

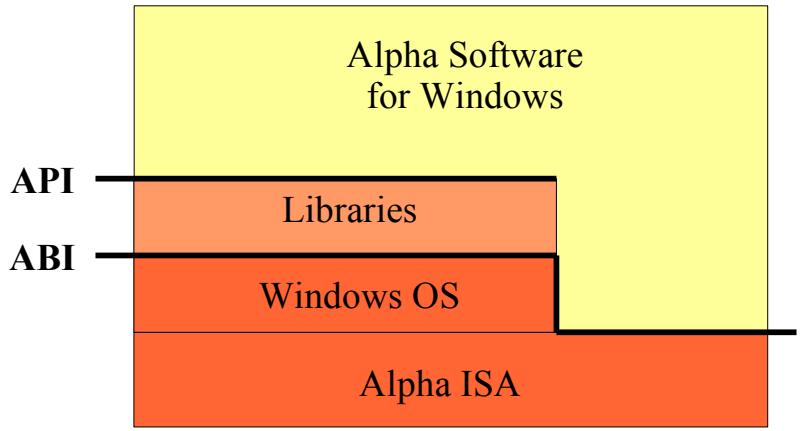
	Architected TLB	Architected Page Table
TLB entry format	Defined in ISA	Left to hardware design
TLB configuration	Defined in ISA	Left to hardware design
Page table entry format	Left to OS implementation	Defined in ISA
Page table configuration	Left to OS implementation	Defined in ISA
Miss in TLB	Causes 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

System-level ISA

- 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 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

Operating System Interface

- 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



Digital Alpha Machine running Windows OS

Operating System 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
    - Overhead
      - Memory barrier hit
        - TLB is empty, so page table lookup will occur
        - The content of L1,L2 caches are irrelevant
      - Potential disk barrier hit
        - Pages in memory may not be the one needed

Operating System Interface

- 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
      - Stalling pipeline, new working set and new locality
      - L1,L2 caches and TLB most likely irrelevant

Operating System Interface

- Signals
  - Used to expose some ISA supports traps and interrupts
    - 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 occurances
    - Either because of real traps or interrupts
    - Could be software generated through the kill() system call

Operating System Interface

- 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);
}
```

Operating System Interface

- 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

Operating System Interface

- 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 priviledged
    - Make device-independent system calls like open(), read() and close()
      - Character devices
        - Direct communication with application code
        - A character at a time
      - Block devices
        - Larger granularity of interactions
        - Data transfers happen through memory buffers

Operating System Interface

- System Calls
  - Device drivers
    - 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