CS550 "Advanced Operating Systems"

Instructor: **Professor Xian-He Sun**

- Email: sun@iit.edu
- Office: SB235C
- Class time: Monday, Wed., 3:15pm-4:30pm, **HH MEZZANINE**
- Office hour: Monday, Wednesday, 4:45-5:45pm
- http://www.cs.iit.edu/~sun/cs550.html

- TA: Mr. Hua Xu, Email: hxu40@hawk.iit.edu
- Office Hour: 11am - 12pm, Tuesday
- meet.google.com/kfp-pysg-cat
- Office Hour: 12pm - 1pm, Tuesday & Thursday

  meet.google.com/bnn-eqao-htg

- Blackboard*:*
  - http://blackboard.iit.edu
- Substitute lecturer:
  - Anthony Kougkas, assistant research professor
  - akougkas@hawk.iit.edu

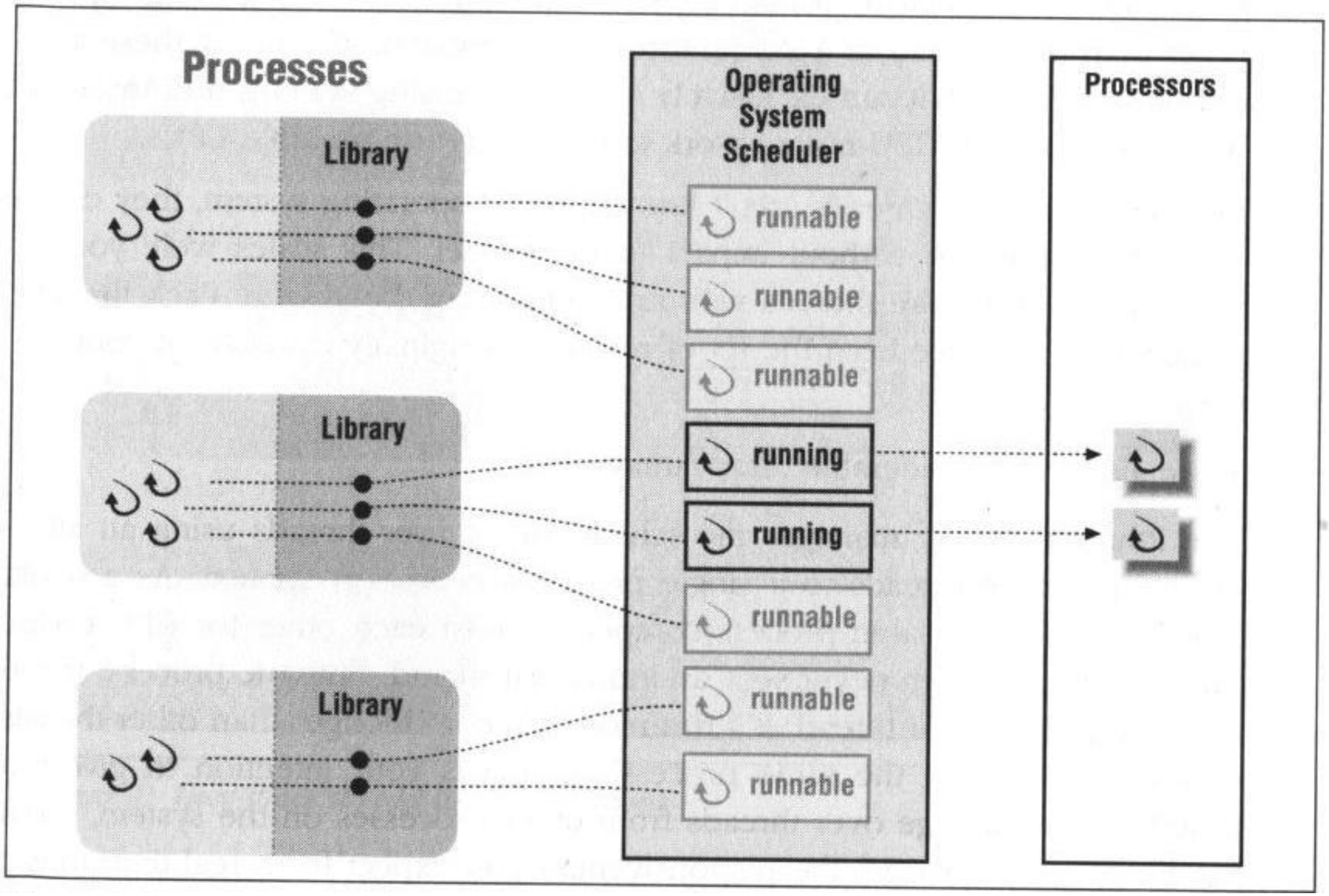
Chapter 3: Processes

- Overview of processes and threads
  - Processes
    - Process scheduling
  - Thread
    - Why use thread
    - Thread types
  - Virtualization
  - Client
  - Server
  - Code/Process Migration

Kernel-level threads

- A **kernel thread** is a thread that the OS knows about
- Does switching between kernel threads of the same process require context switch or not?
- The kernel must manage and schedule threads (as well as processes), but it can use the same process scheduling algorithms

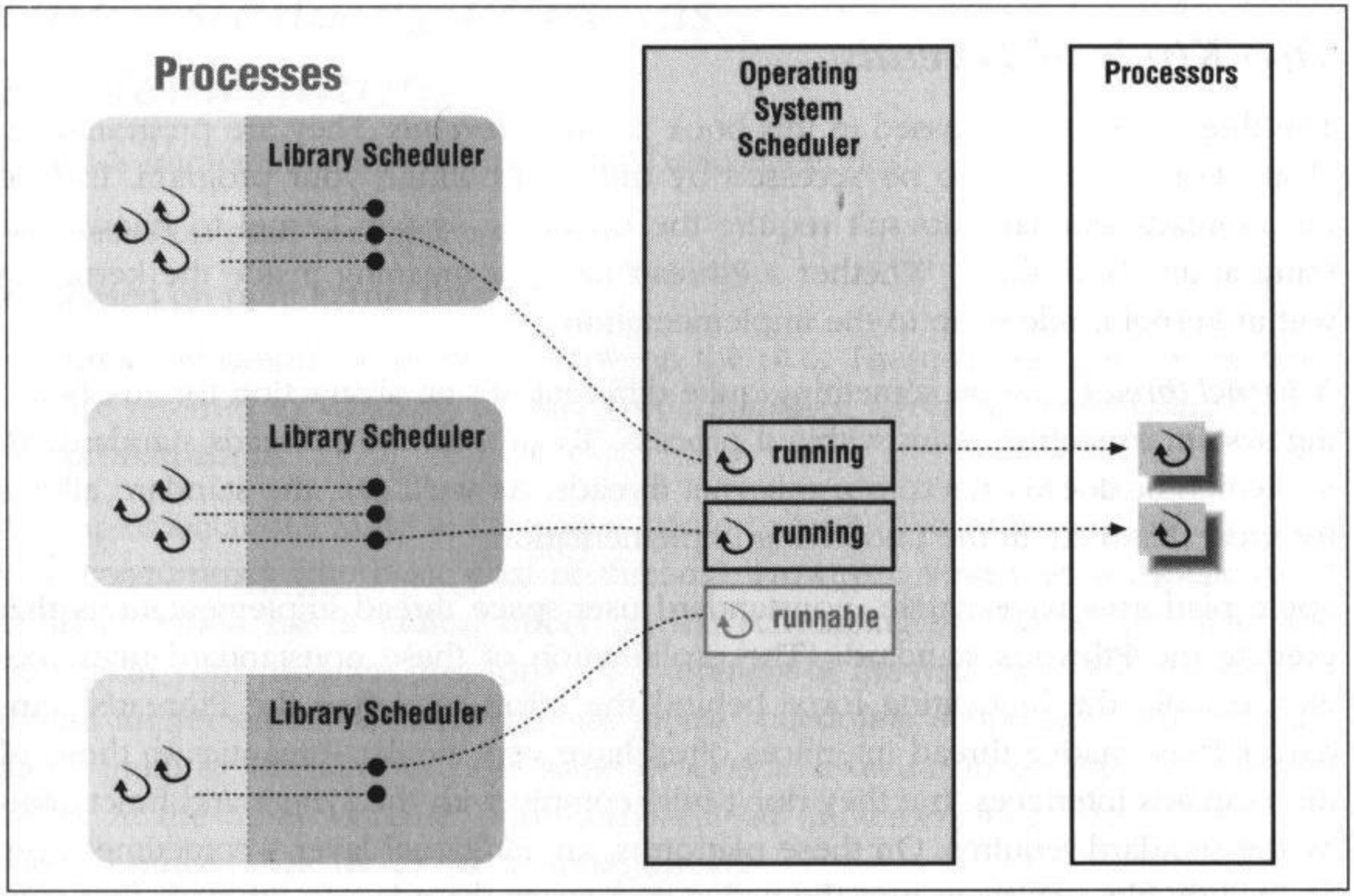
Kernel-level threads



User-level Threads

- A **user-level thread** is a thread that the OS does not know about
- The OS only knows about the process containing the threads
- The OS only schedules the process, not the threads within the process
- The programmer uses thread library to manage threads
  - Create and delete
  - Synchronize
  - Schedule

User-level threads



User-level Threads

- Advantages?

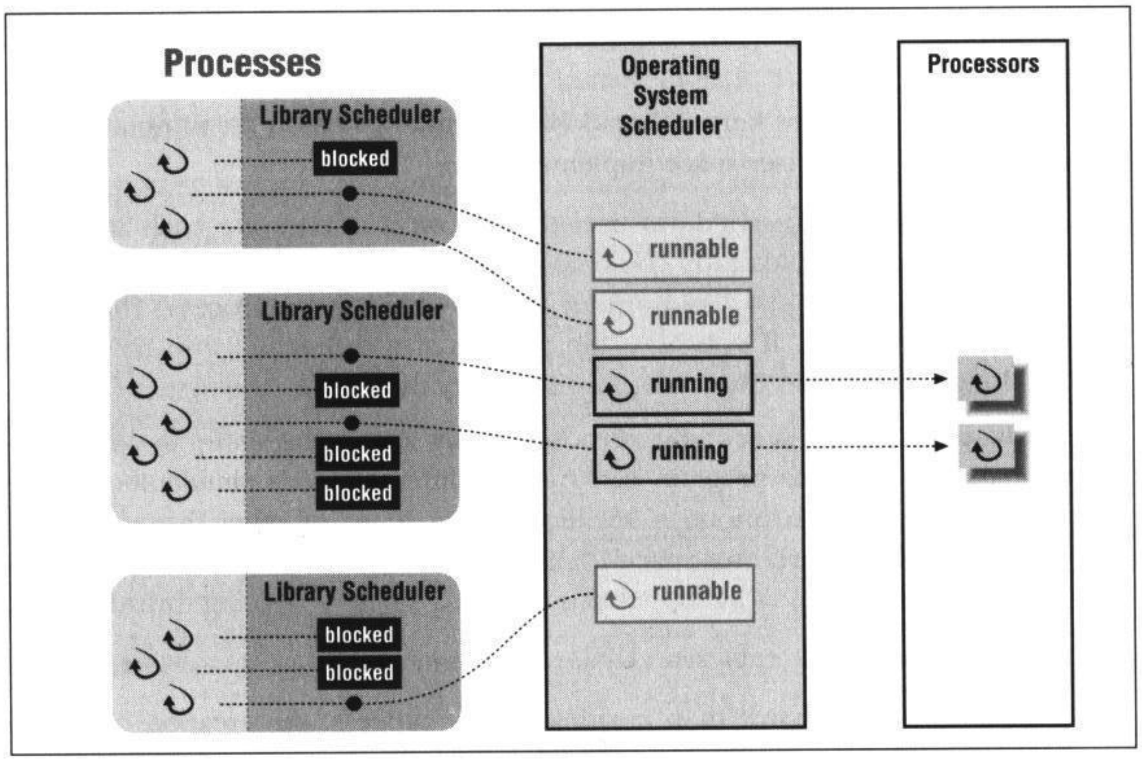
- Disadvantages?

Hybrid Implementation: Light-Weight Processes (LWP)

- Several LWPs per heave-weight process
- User-level threads package
- Multithreaded applications – create multiple threads, assign threads to LWPs (one-one, many-one, many-many)
- Two-level scheduling:
  - Each LWP, when scheduled, searches for a runable thread
    - Shared thread table: no kernel support needed
  - When a LWP thread block on system call, switch to kernel mode and OS context switches to another LWP

\

LWP Example



Processes
Library Scheduler
blocked
Library Scheduler
blocked
blocked
blocked
Library Scheduler
blocked
blocked
Operating
System
Scheduler
runnable
runnable
running
running
runnable
Processors

Light-weight Processes

- Advantages:
  - ?

- Disadvantages:
  - ?

Thread Management

- Creation and deletion of threads
  - Static versus dynamic
- Critical sections
  - Peer threads
  - Synchronization primitives: blocking, spin-lock (busy-wait)
  - Condition variables
- Global thread variables
- Kernel versus user-level threads

Threads: Summary

- Thread: a single execution stream within a process
- Switching between user-level threads is faster than between kernel threads since a context switch is not required
- User-level threads may result in the kernel making poor scheduling decisions, resulting in slower process execution than if kernel threads were used
- Many scheduling algorithms exist, which should be based on characteristics of processes and OS

Thread Packages

- Posix Threads (pthreads)
  - Widely used threads package
  - Conforms to the Posix standard
  - Sample calls: pthread_create,…
  - Typical used in C/C++ applications
  - Can be implemented as user-level or kernel-level or via LWPs
- Java Threads
  - Native thread support built into the language
  - Threads are scheduled by the JVM

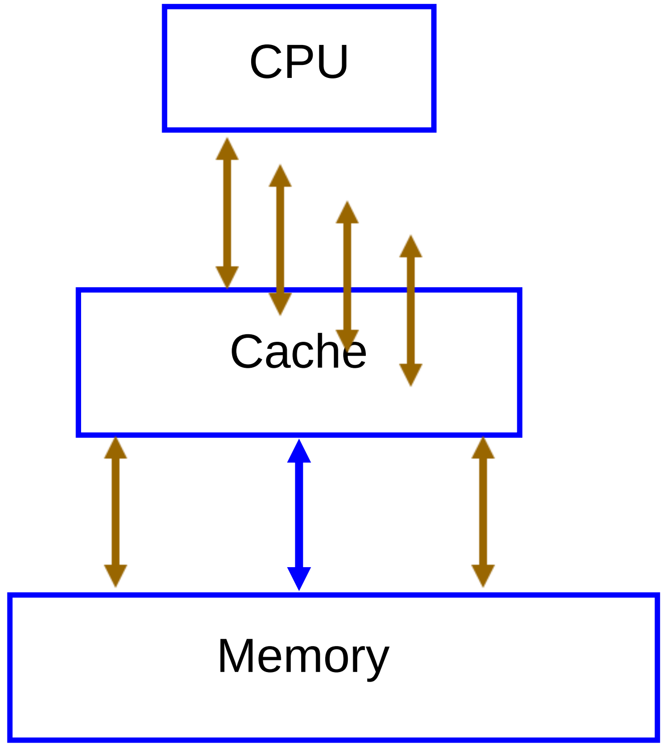
Discussion

- Threats are lightweight Processes
- Could be anything even better (lighter) than threat?
- Yes, Out-of-Order Execution
- What is Out-of-Order execution?
- How it is related to recent Intel Meltdown and Spectre

Solution: Memory Hierarchy & **Concurrency**

Multi-core
Multi-threading
Multi-issue

Out-of-order Execution
Speculative Execution
Runahead Execution



Multi-banked Cache
Multi-level Cache

Pipelined Cache
Non-blocking Cache
Data Prefetching
Write buffer

Multi-channel
Multi-rank
Multi-bank

Pipeline
Non-blocking
Prefetching
Write buffer

Input-Output (I/O)

Parallel File System

Disks

Summary

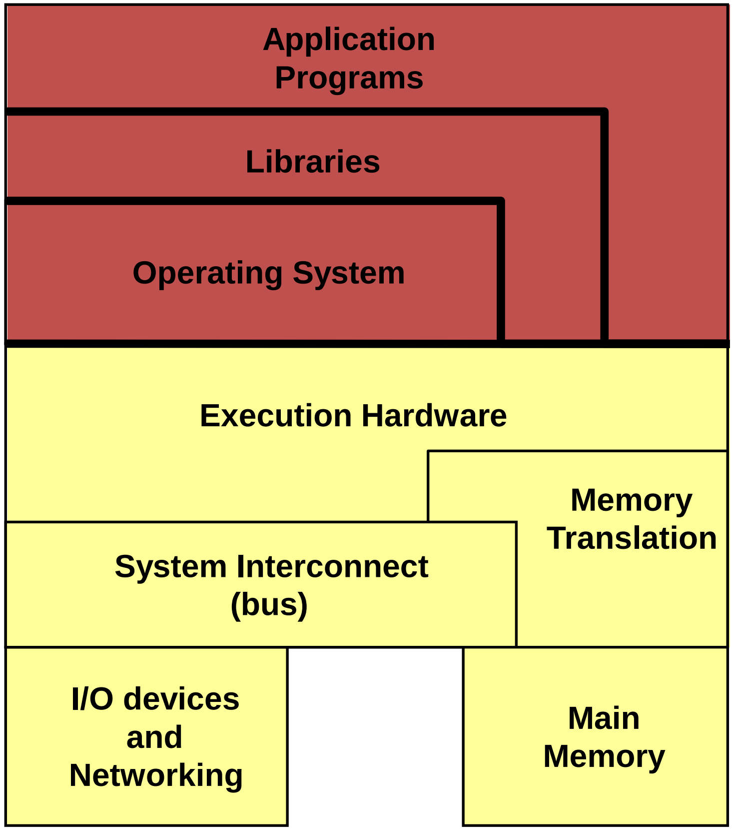
- Overview of processes and threads (today)
  - Processes
  - Process scheduling
  - Thread
  - Why use thread
  - Thread types
- Readings:
  - Chpt 3

Chapter 3: Processes

- Overview of processes and threads
- Virtualization
- Clients
- Servers
- Code/Process migration

The “Machine”

- Different perspectives on what the *Machine* is:
- OS developer



Instruction Set Architecture

- ISA
- Major division between hardware and software

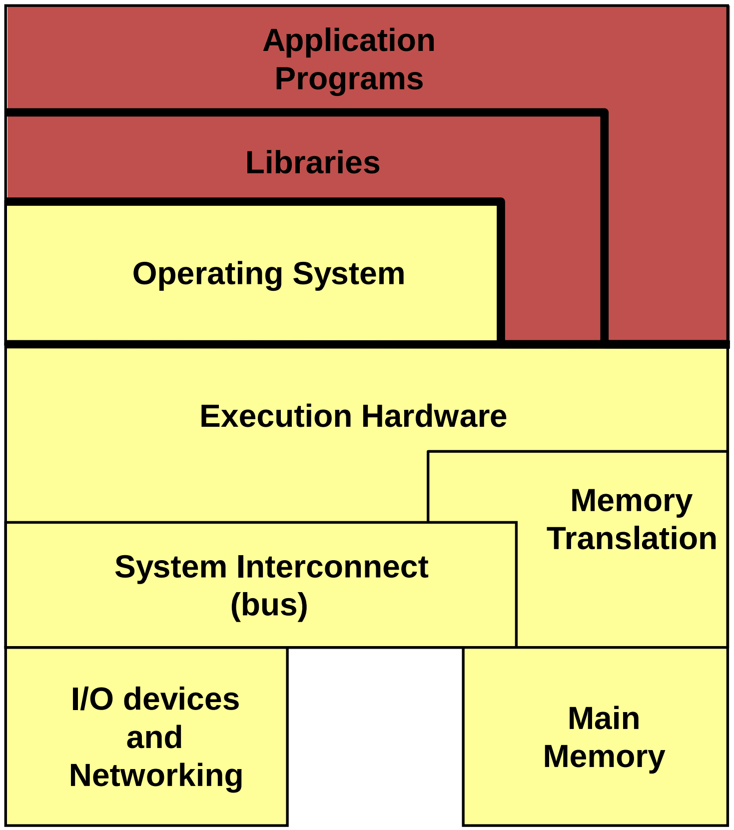
The “Machine”

- Different perspectives on what the *Machine* is:
- Compiler developer

Application Binary Interface

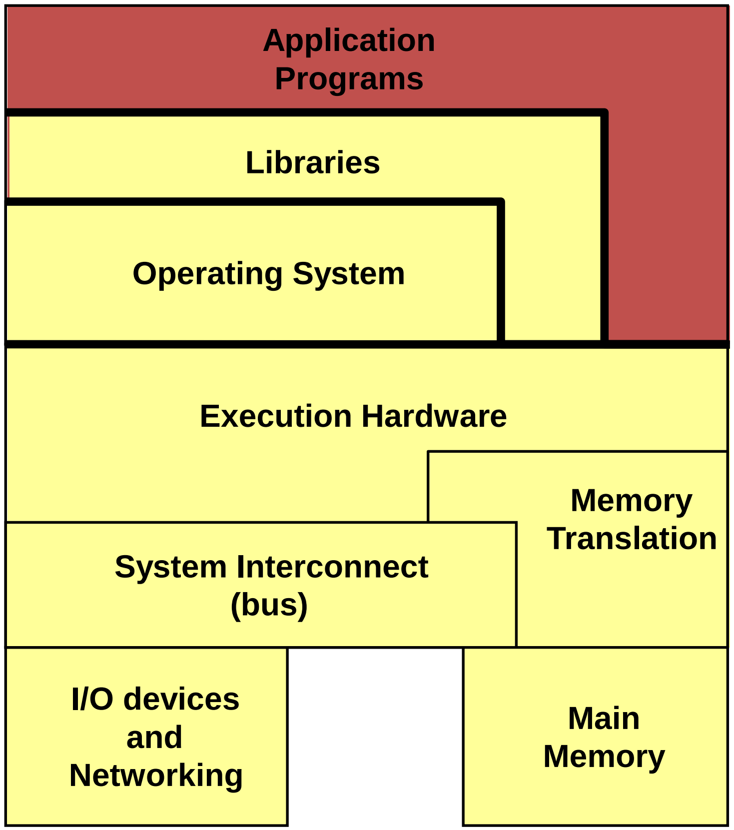
– ABI

– User ISA + OS calls



The "Machine"

- Different perspectives on what the *Machine* is:
- Application programmer



Application Program Interface

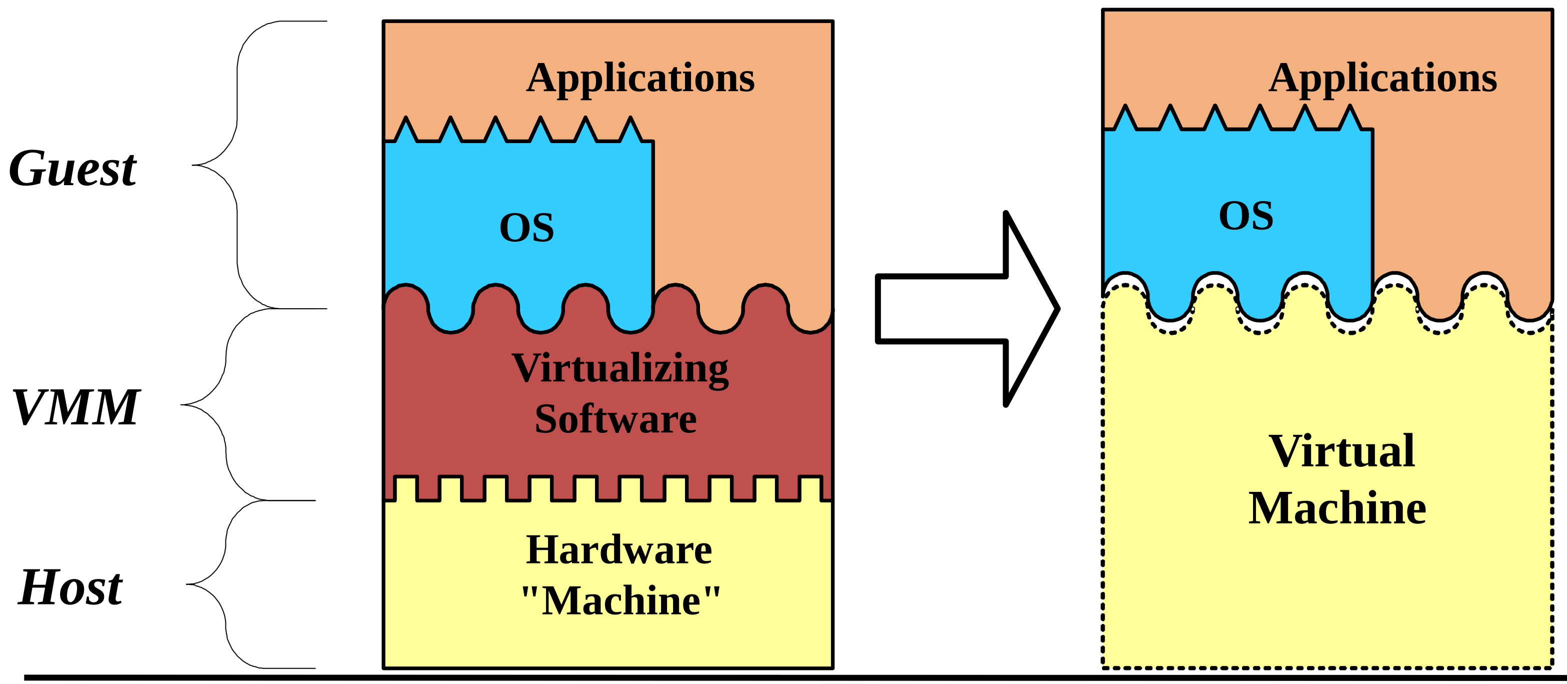
- API
- User ISA + library calls

Virtual Machines

add *Virtualizing Software* to a *Host* platform

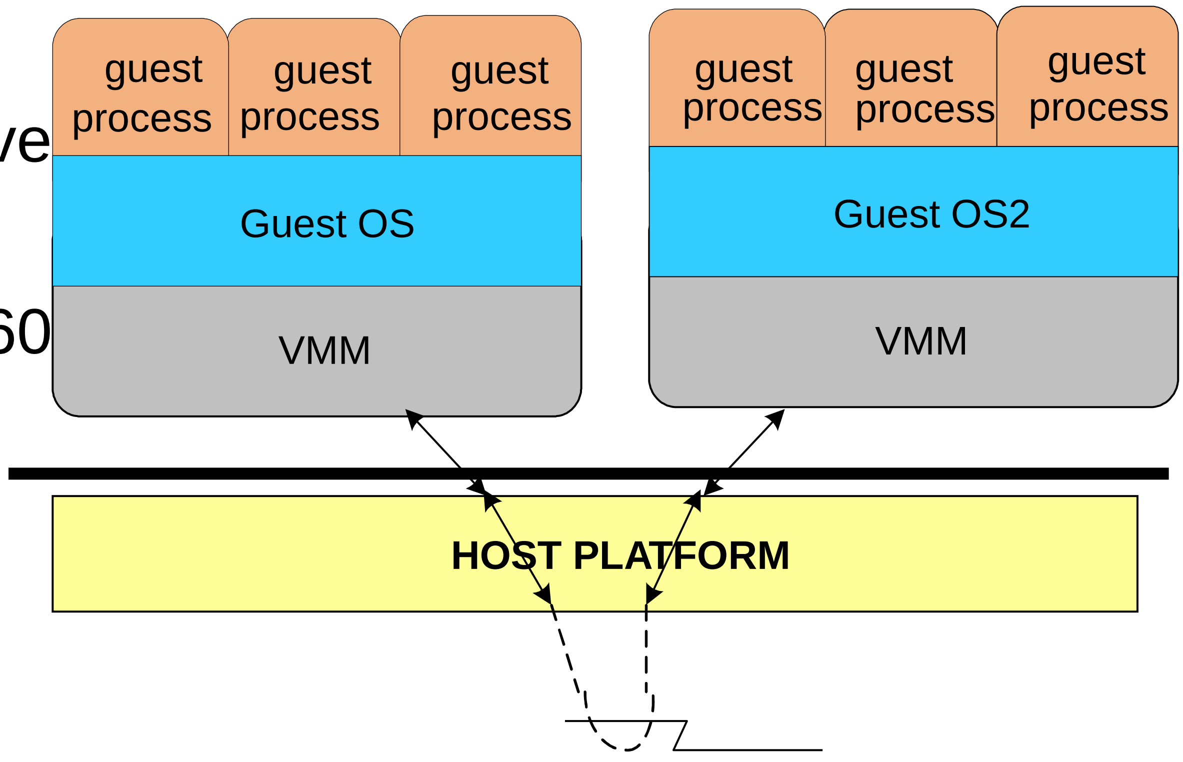
and support *Guest* process or system on a *Virtual Machine* (VM)

Example: System Virtual Machine



System Virtual Machines

- Provide a system environment
- Constructed at ISA level
- Persistent
- Examples: IBM VM/360, VMware, Transmeta Crusoe



Applications

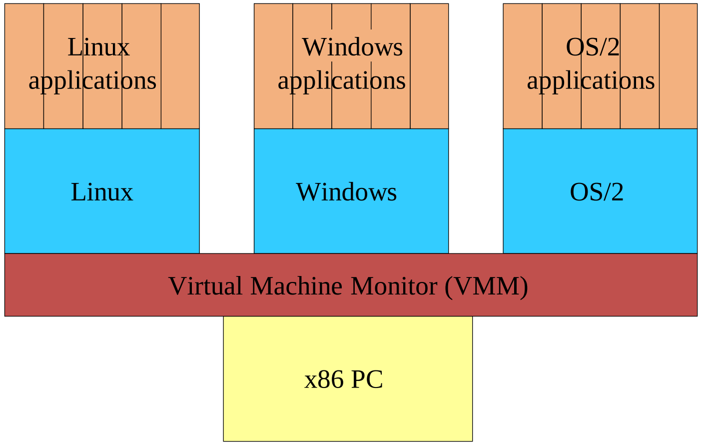
- Simultaneous support for multiple OSes/Apps
  - Easy way to implement timesharing
- Simultaneous support for different OSes/Apps
  - E.g. Windows and Unix
- Error containment (security)
  - If a VM crashes, the other VMs can continue to work
    Assumes VMM is correct (smaller/simpler)
- Operating System debugging
  - Can proceed while system is being used for normal work

Applications, contd.

- Retrofitting new features
  - Have VMM transform new device into a virtual device
- Support for multiple networked machines on one physical machine
  - Allows debug of network software
- Enables complex debugging and performance monitoring tools
  - By putting them in the VMM (not the guest OS)
- Education

System VMs

- Virtual Machine Monitor (VMM) manages real hardware resources
- All Guest systems must be given logical hardware resources
- All resources are *virtualized*
  - By partitioning real resources
  - By sharing real resources
- Guest state must be managed
  - By using indirection
  - By copying



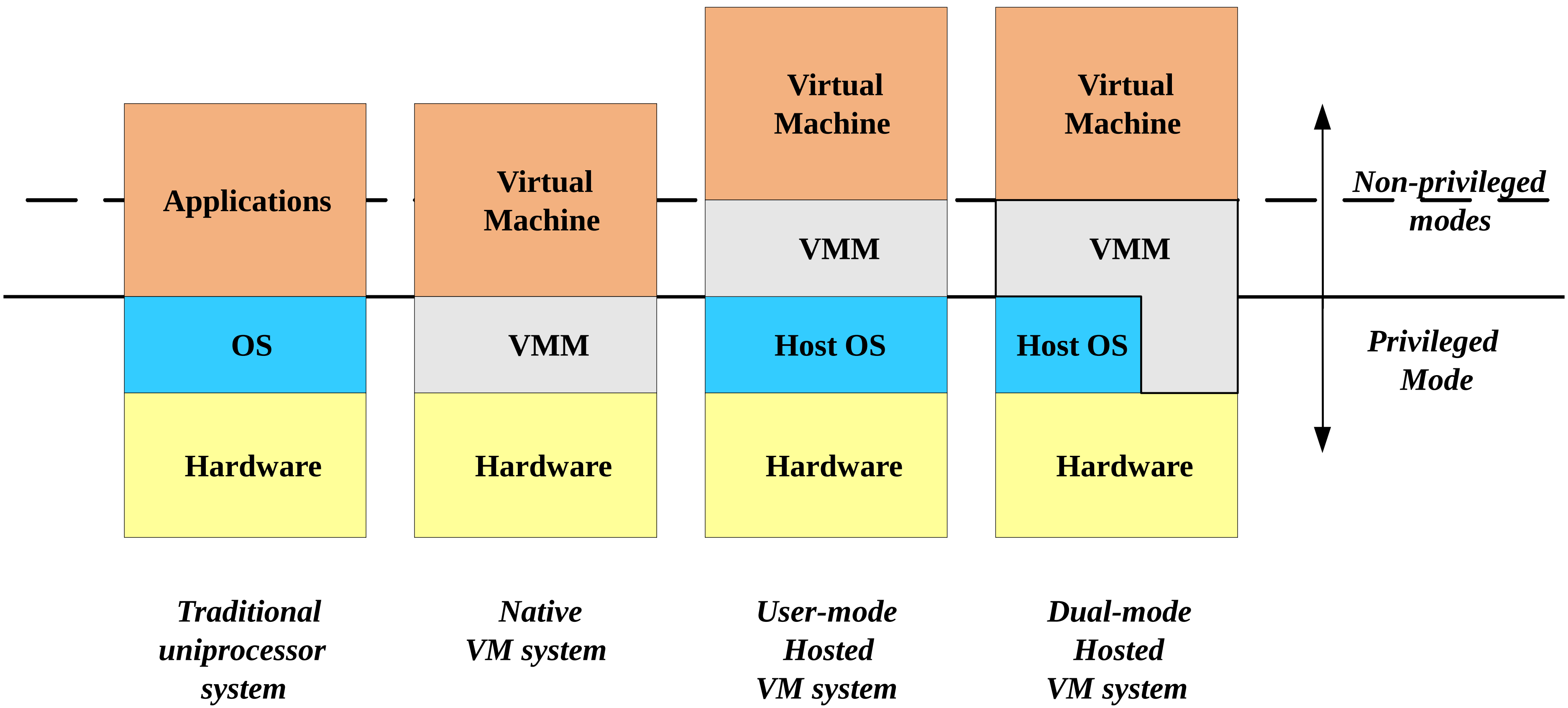
System VMs: Processor Mgmt/Protection

- VMM runs in *system* mode
  - VMM manages/protects processor through conventional mechanisms
- Guest OSes run in *user* mode

  $\Rightarrow$ Guest OSes do not have direct control over hardware resources

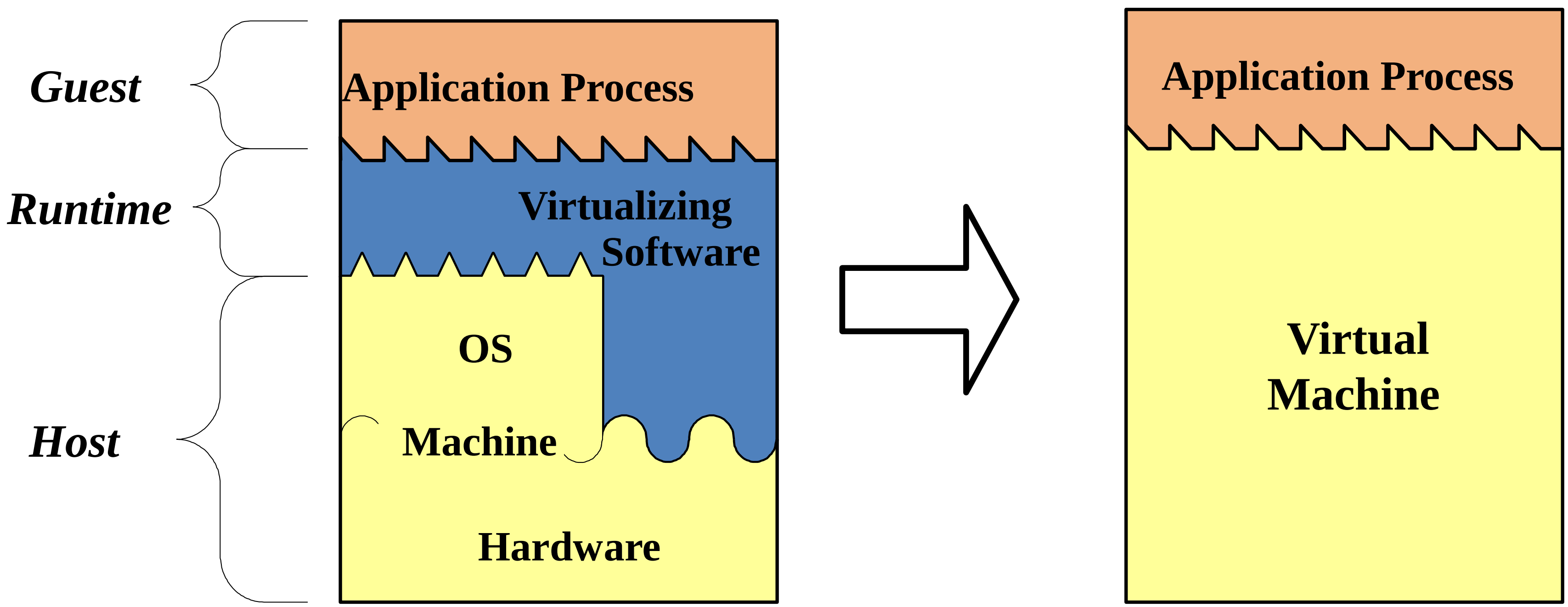
  All attempts to interact w/ hardware resources are intercepted by VMM
- VMM manages shadow copies of Guest System state (incl. control registers)
- VMM schedules and runs Guest Systems

Native and Hosted VMs



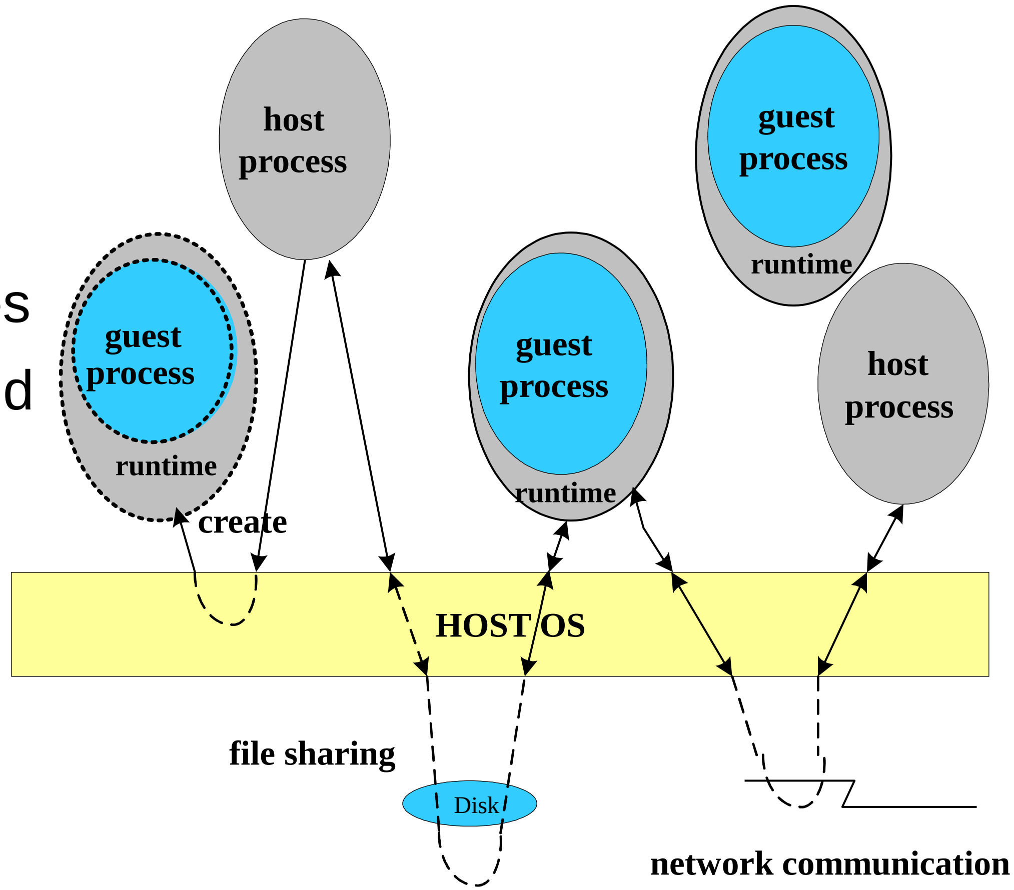
Process VMs

- Execute application binaries with an ISA *different* from hardware platform
- Couple at ABI level via *Runtime System*
- Examples: IA-32 EL, FX!32



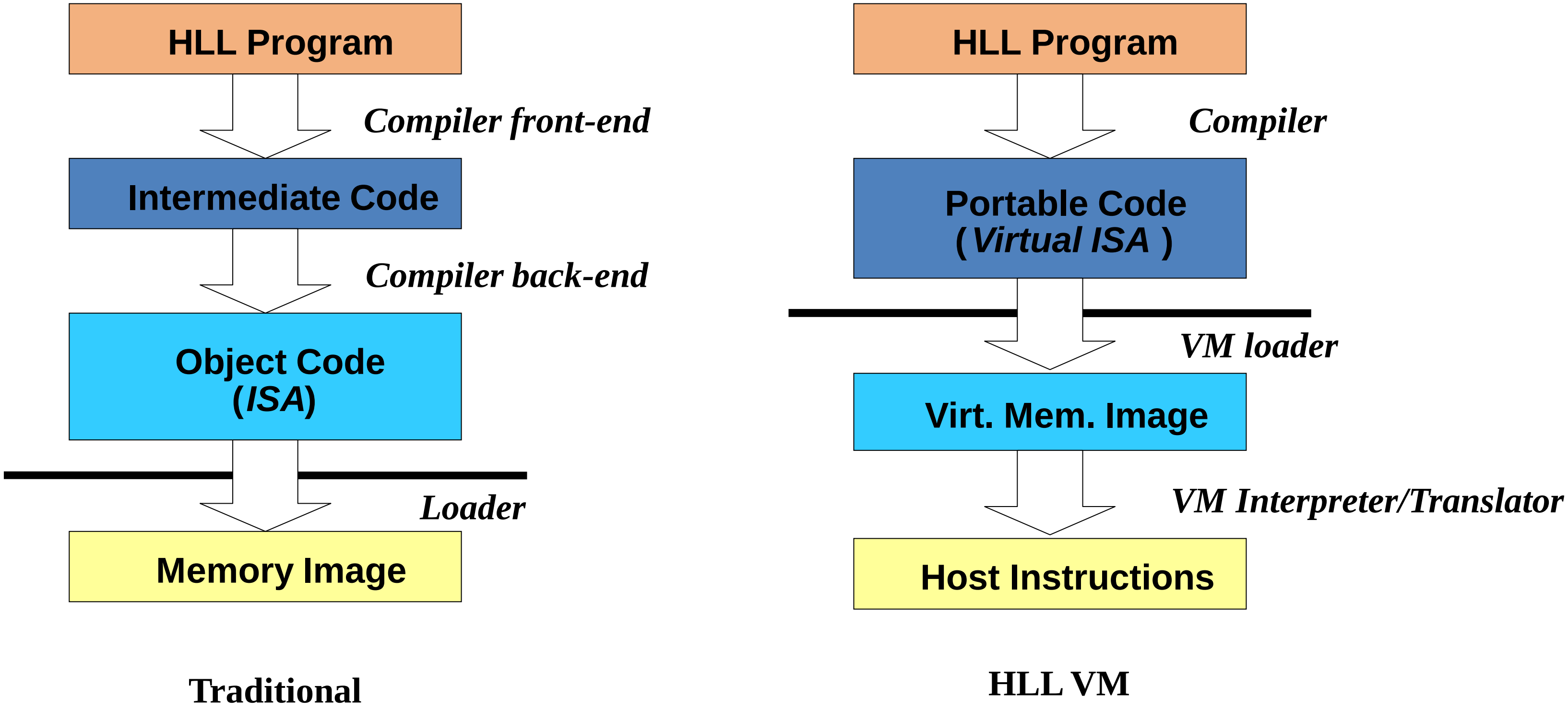
Process Virtual Machines

- Constructed at ABI level
- *Runtime* manages guest process
- Not persistent
- Guest processes may intermingle with host processes
- As a practical matter, guest and host OSes are often the same
- Dynamic optimizers are a special case
- Examples: IA-32 EL, FX!32, Dynamo



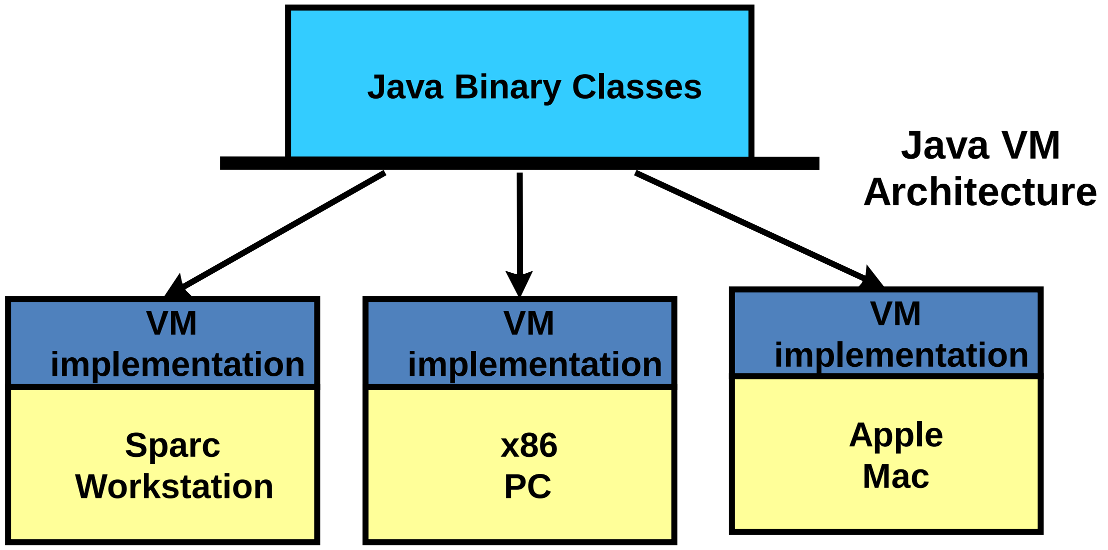
High Level Language Virtual Machines

- Raise the level of abstraction
  - User higher level virtual ISA
  - OS abstracted as standard libraries
- Process VM (or API VM)



HLL VMs

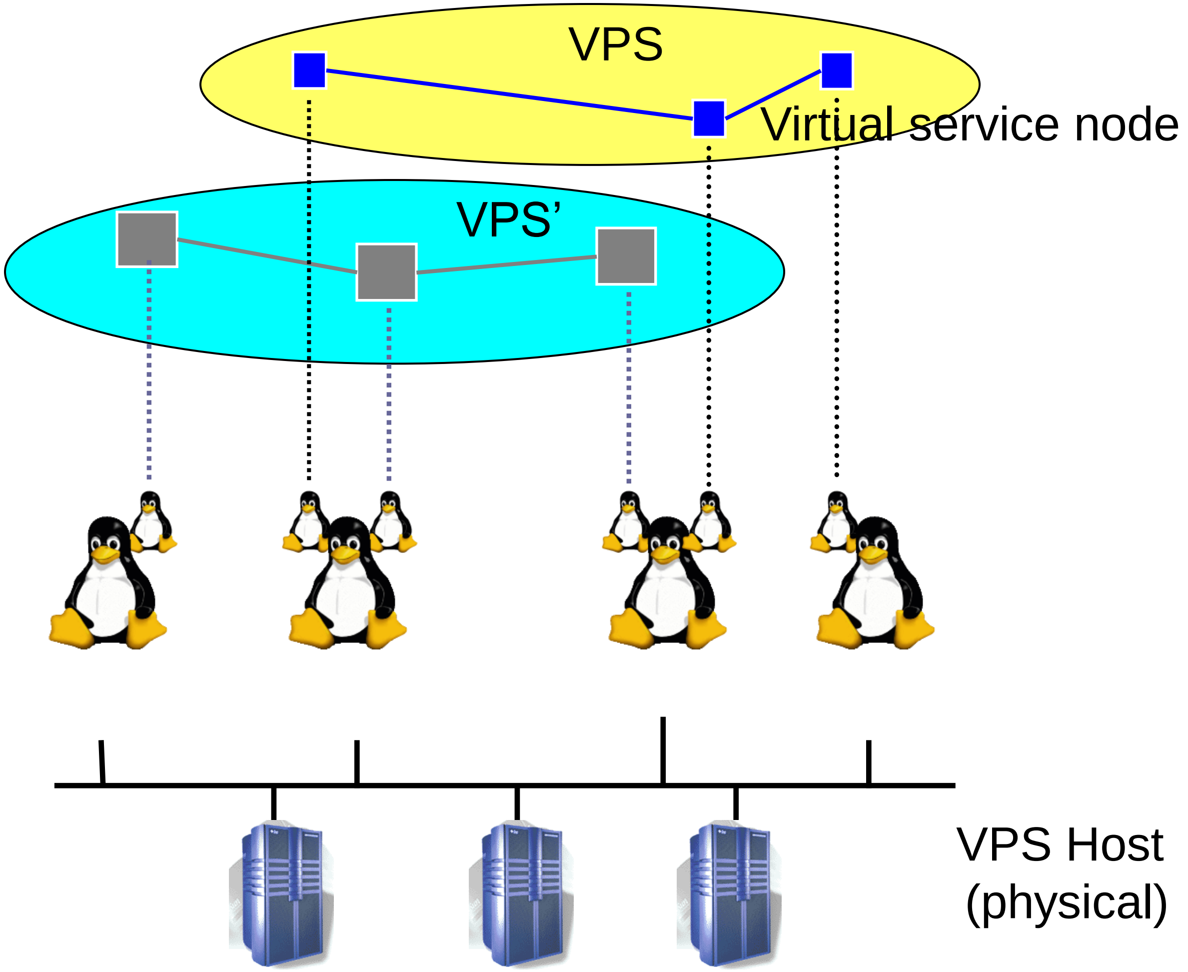
- Java and CLI are recent examples
- Binary class files are distributed
- “ISA” is part of binary class format
- OS interaction via APIs (part of VM platform)



VMware: an x86 System Virtual Machine

- Applying Conventional VMs to PCs – Problems:
  - Installing the VMM on bare hardware, then booting Guests onto VMM.
  - Need to support many device types, many more drivers
- VMware solves both problems
- Uses Host OS/Guest OS model
  - “Hosted VM”
  - Uses Host OS for some VMM functions
    - Including I/O

Virtual Private Service Environment



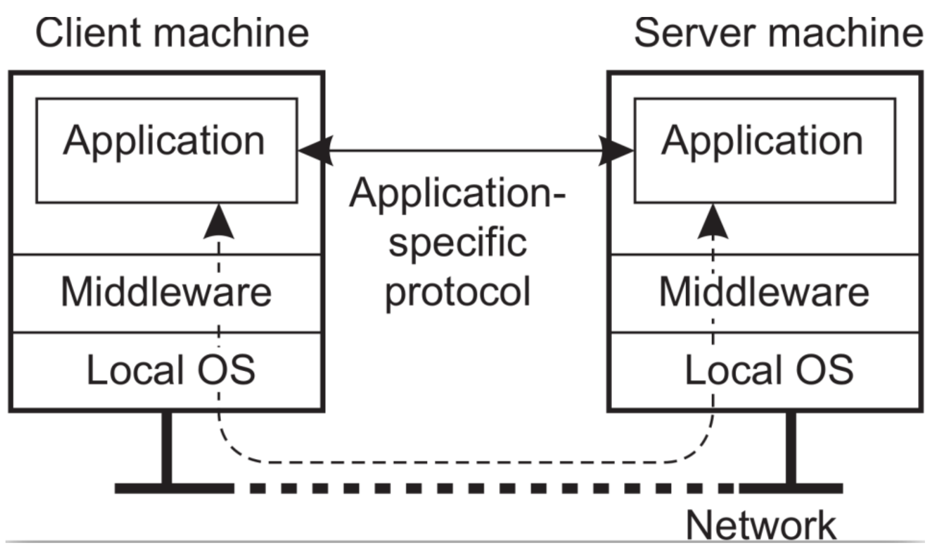
Client & Server

Outline

- Networked user interfaces
- Thin-client network computing
- Client-side software for distribution transparency
- Concurrent versus iterative servers
- End-points
- Interrupting a server
- Stateless vs stateful servers
- Server clusters

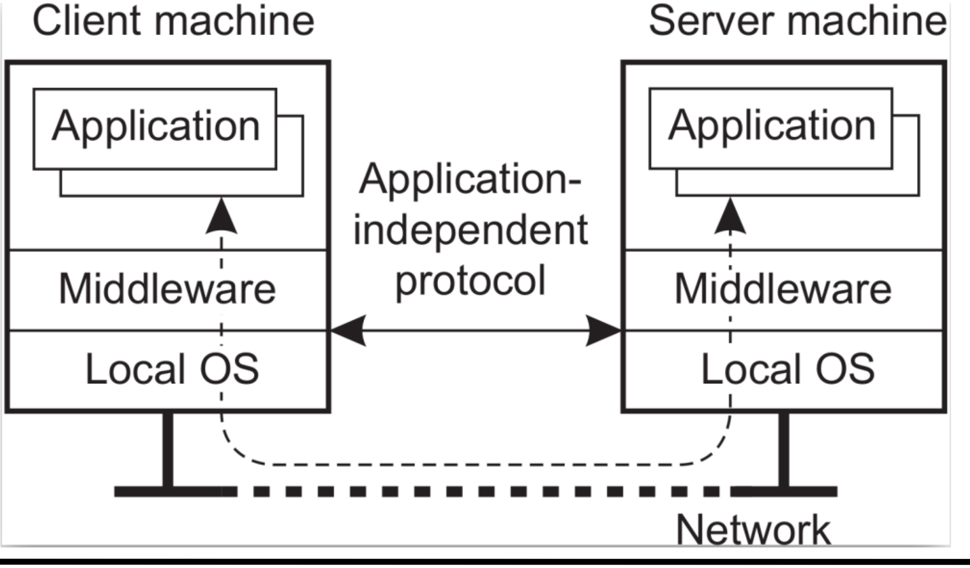
Networked user interfaces

- Provide the means for users to interact with remote servers.
- Application-level protocol
  - Separate interface for each remote service
  - e.g. Calendar sync with cloud



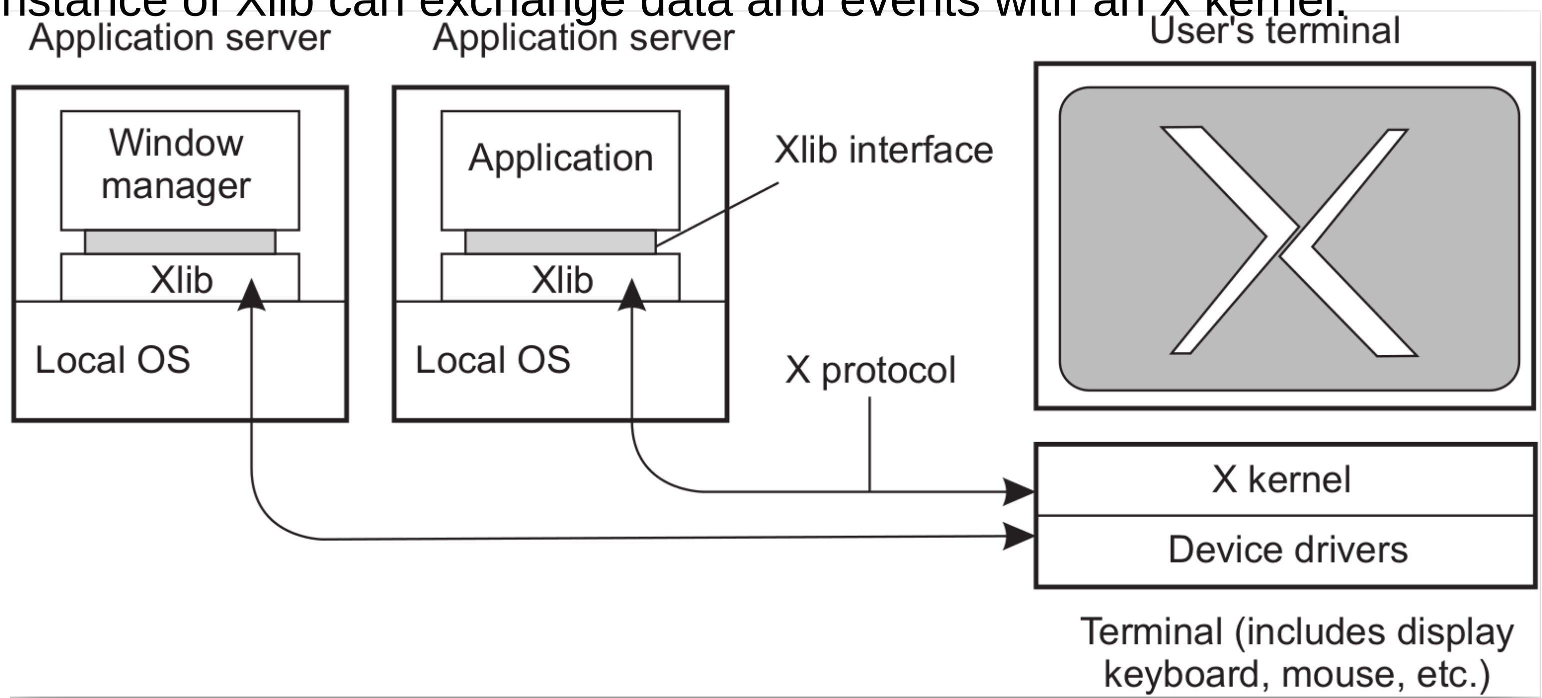
Networked user interfaces

- Direct access to remote services by offering a User Interface.
- Client machine is used only as a terminal
  - No need for local storage
  - Application-neutral solution



Example: The X window system

- Is used to control bit-mapped terminals
- X kernel contains all the terminal-specific device drivers
- This interface is made available to applications as a library called Xlib
- X kernel and the X applications need not necessarily reside on the same machine
- X protocol is an application-level communication protocol by which an instance of Xlib can exchange data and events with an X kernel.



Thin-client network computing

- Synchronous client behavior may adversely affect performance when operating over a wide-area network with long latencies.
- Solutions:
  - Reimplement X protrocol: bandwidth reduction by using small messages.
  - Application completely controls the remote display, up to the pixel. Changes in the bitmap are sent over the network to the display, where they are immediately transferred to the local frame buffer.
  - Typical: Virtual network computing.

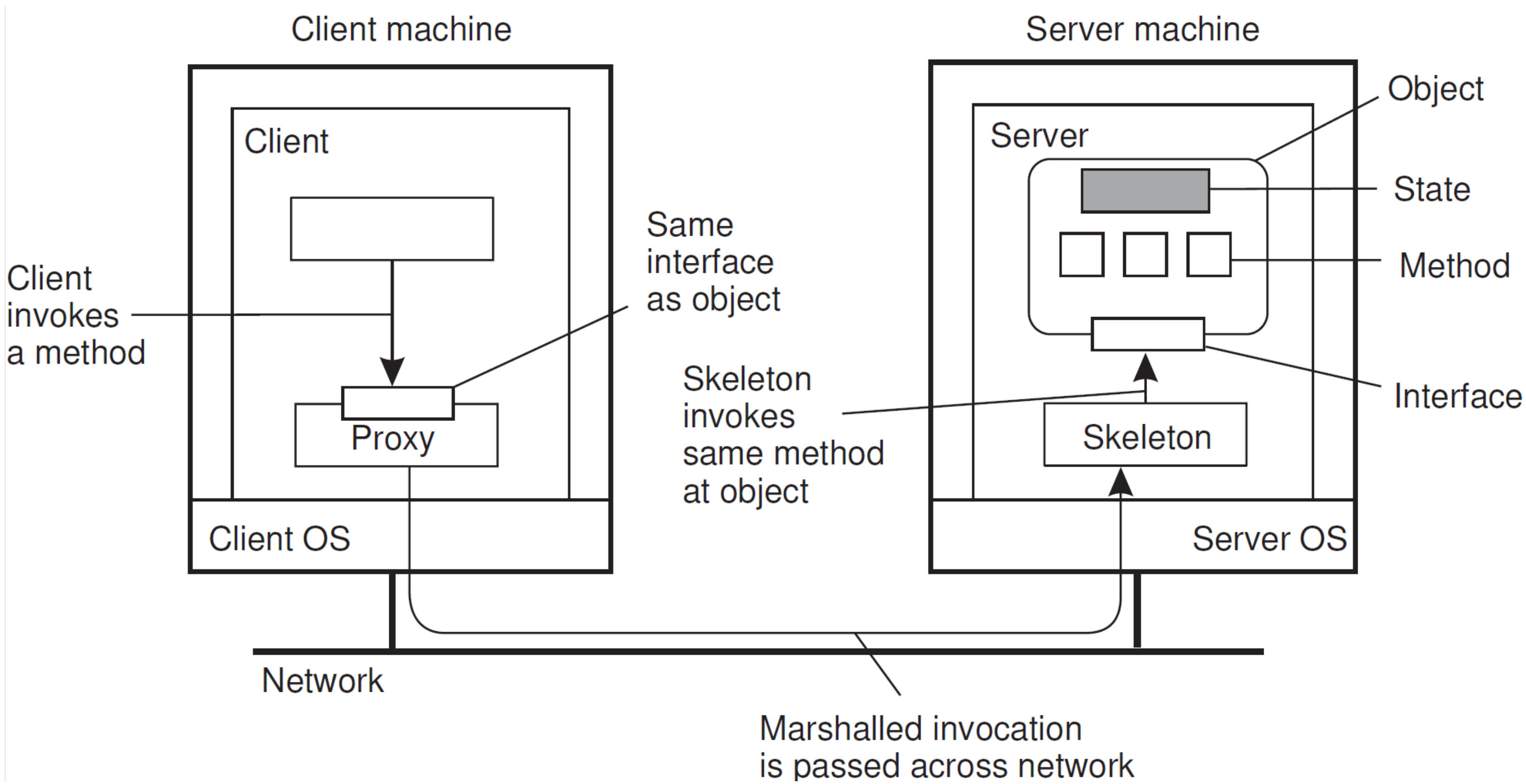
Client-side software

- Client software comprises more than just user interfaces.
- Parts of the processing and data level in a client-server application are executed on the client side:
  - automatic teller machines (ATMs), cash registers, barcode readers, TV set-top boxes, etc
  - In these cases, UI is only a small part compared to local processing and communication facilities
- Client software comprises components for achieving distribution transparency
  - Client should not be aware that it is communicating with remote processes.

Software distribution transparency

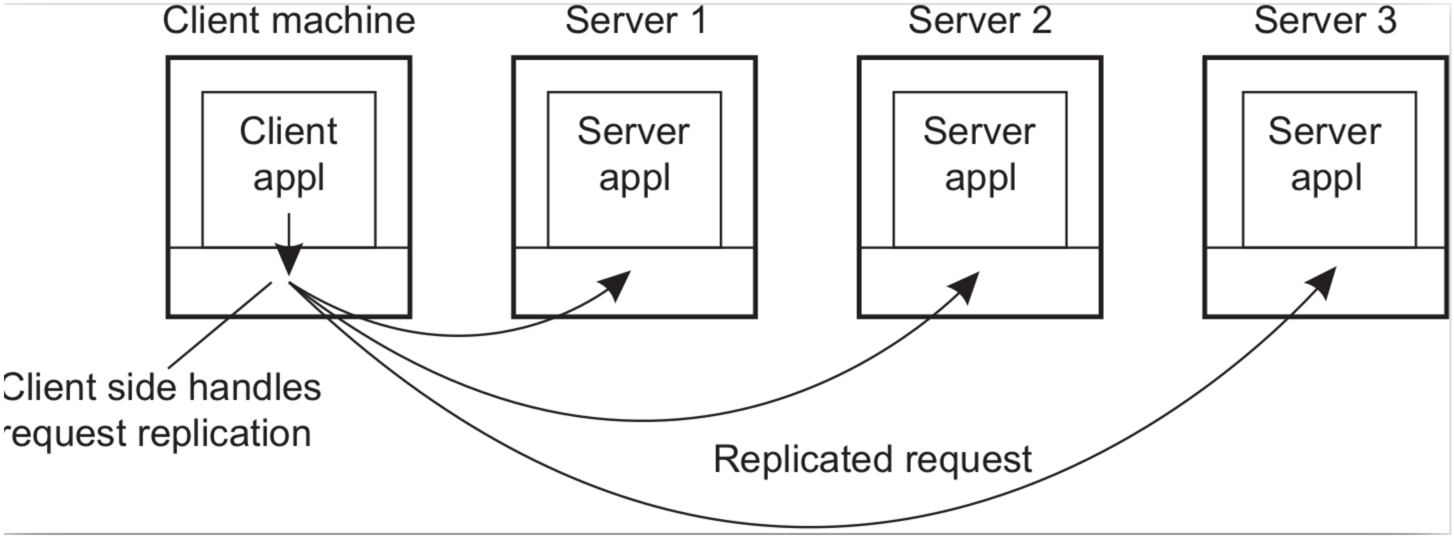
- Access transparency is generally handled through the generation of a **client stub** from an interface definition of what the server has to offer.
- Stubs:
  - hide the possible differences in machine architectures, as well as the actual communication
  - transform local calls to messages that are sent to the server, and vice versa
- Location, migration, and relocation transparency:
  - Use convenient naming system

Object-Based Style Example



Software distribution transparency

- Replication transparency?
  - Use client-side solutions
  - Forward a request to each replica
  - collect all responses and pass a single response to the client application



Software distribution transparency

- Failure transparency?
  - mask communication failures with a server through client middleware.
  - repeatedly attempt to connect to a server
  - try another server after several attempts
  - returns data it had cached during a previous session
- Concurrency transparency?
  - through special intermediate servers, notably transaction monitors,
  - requires less support from client software

Concurrent vs Iterative servers

- **Iterative server:**
  - the server itself handles the request and, if necessary, returns a response to the requesting client
- **Concurrent server:**
  - does not handle the request itself but passes it to a separate thread or another process, after which it immediately waits for the next incoming request.
  - Multithreaded: fork a new process for each new incoming request