

Advanced Operating Systems

20MCAT172

Module IV

Syllabus

Multiprocessor Operating Systems:- Basic Multiprocessor System Architectures – Interconnection Networks – Structures – Design Issues – Threads – Process – Synchronization – Processor Scheduling – Memory Management - Virtualization – Types of Hypervisors – Paravirtualization – Memory Virtualization – I/O Virtualization.

(Mukesh Singhal and Niranjan G. Shivaratri, “***Advanced Concepts in Operating Systems*** – Distributed, Database, and Multiprocessor Operating Systems”, Tata McGraw-Hill, 2001.)

Multiprocessor Operating Systems

- A **multiprocessor system** is defined as "a system with more than one processor".
- A **multiprocessor system** consists of several processors that share a common physical memory.
- All the processors operate under the control of a single operating system.
- Users of a **multiprocessor system** see a single powerful computer system.
- Multiplicity of the processors in a multiprocessor system and the way processors act together to perform a computation are transparent to the users.

Motivations For Multiprocessor Systems

- The main motivations for a **multiprocessor system** are to achieve enhanced performance and fault tolerance.
- Enhanced Performance: Multiprocessor systems increase system performance in two ways –
 - Concurrent execution of several tasks by different processors increases the system throughput. (The number of tasks completing per time unit without speeding up the execution of individual tasks)
 - A multiprocessor system can speed up the execution of a single task in the following way:
 - If parallelism exists in a task, it can be divided into many subtasks and these subtasks can be executed in parallel on different processors.
- Fault Tolerance: A multiprocessor system exhibits graceful performance degradation to processor failures because of the availability of multiple processors.

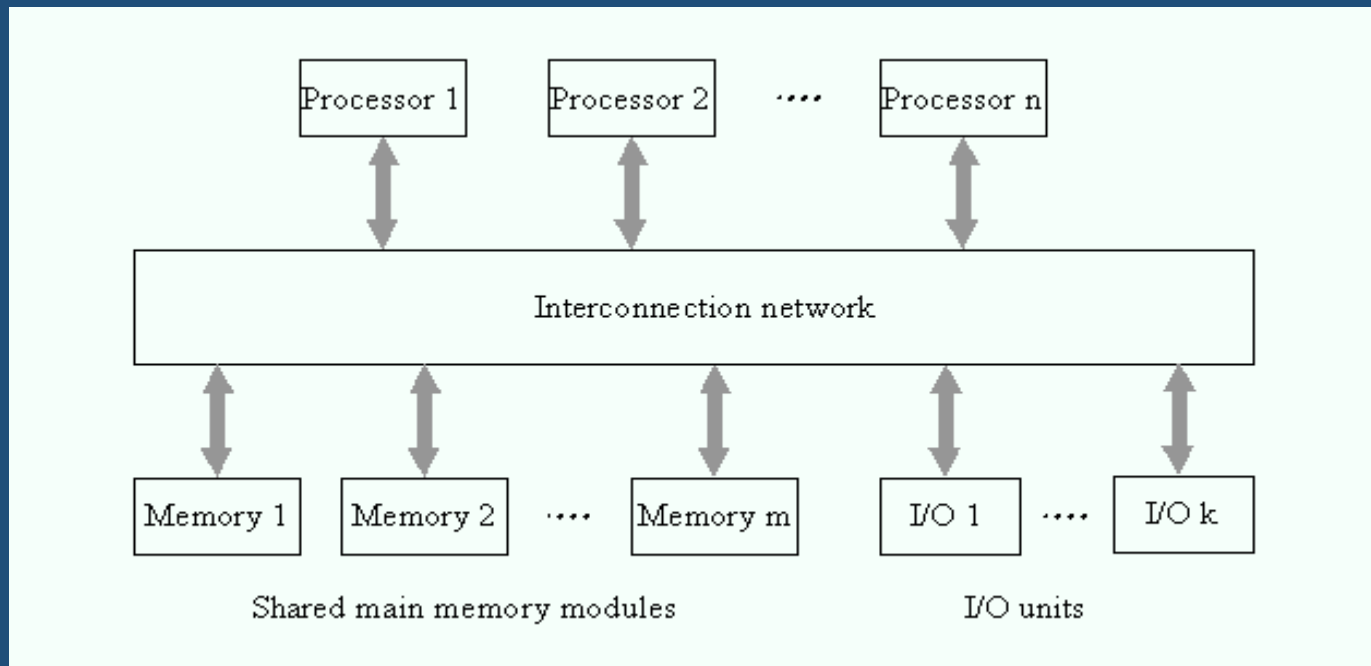
Basic Multiprocessor System Architectures

Basic Multiprocessor System Architectures

- Based on whether a memory location can be directly accessed by a processor or not, there are two types of multiprocessor systems:
 - **Tightly Coupled Multiprocessor System.**
 - **Loosely Coupled Multiprocessor System.**
- Based on the vicinity and accessibility of the main memory to the processors, there are three types of multiprocessor systems:
 - **UMA (Uniform Memory Access)**
 - **NUMA (Nonuniform Memory Access)**
 - **NORMA (No Remote Memory Access)**

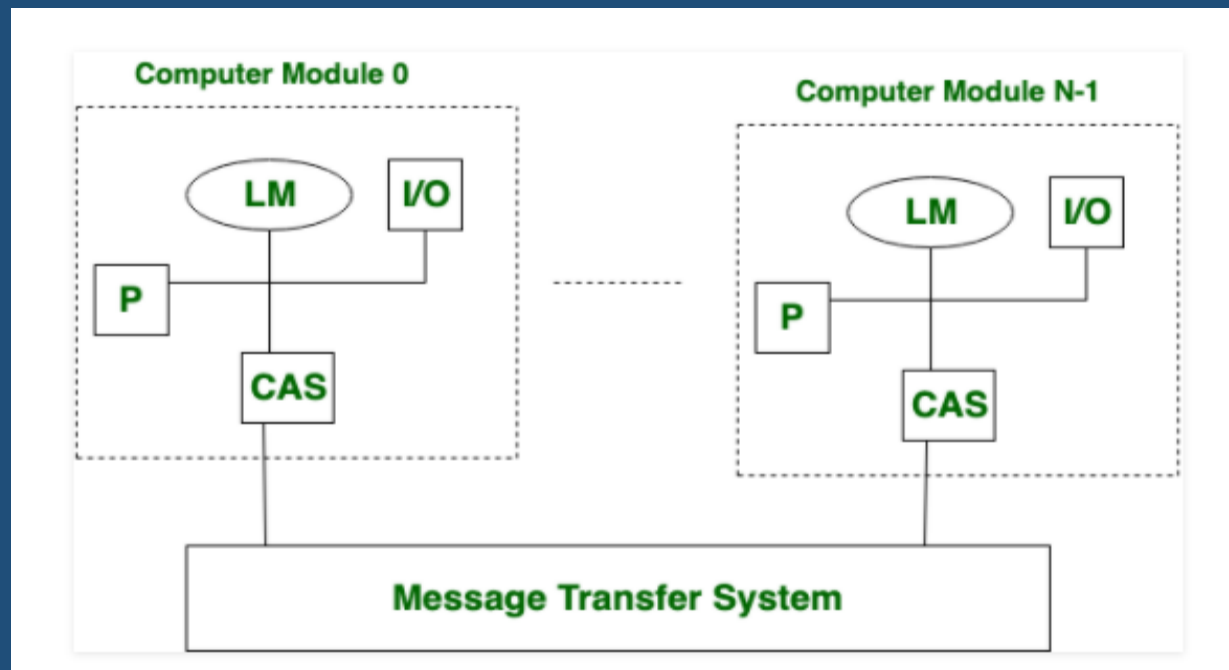
Tightly Coupled Multiprocessor System.

- In Tightly Coupled Multiprocessor System, all processors share the same memory address space and all processors can directly access a global main memory.
- Tightly Coupled Systems can use the main memory for interprocessor communication and synchronization.
- Memory Contention occurs.



Loosely Coupled Multiprocessor System.

- There is distributed memory instead of shared memory.
- In loosely coupled multiprocessor system, data rate is low rather than tightly coupled multiprocessor system.
- In loosely coupled multiprocessor system, modules are connected through MTS (Message transfer system) network..



- **UMA (Uniform Memory Access)**

- Main Memory is located at a central location

- **NUMA (Nonuniform Memory Access)**

- Main Memory is physically partitioned and partitions are attached to the processors
- Processor can directly access the memory.

- **NORMA (No Remote Memory Access)**

- Main Memory is physically partitioned and partitions are attached to the processors.
- Processor cannot directly access the memory.

Interconnection Networks For Multiprocessor systems

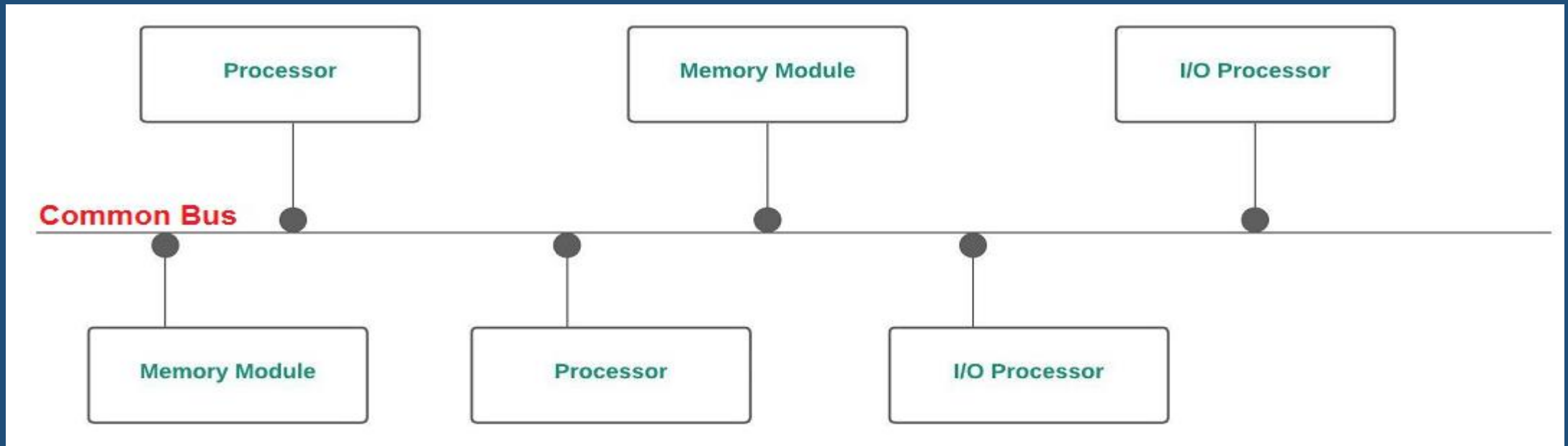
Interconnection Networks

- The interconnection network in multiprocessor systems provides data transfer facility between processors and memory modules for memory access.
- The design of the interconnection network is the most crucial hardware issue in the design of multiprocessor systems.
- Generally, circuit switching is used to establish a connection between processors and memory modules.
- During a data transfer, a dedicated path exists between the processor and the memory module.
- Types of interconnection networks include:
 - **Bus**
 - **Cross-bar Switch**
 - **Multistage Interconnection Network**

Interconnection Networks

- **Bus**

- Processors are connected to memory module via a bus.
- Simplest multiprocessor architecture. Less Expensive.
- Can only support a limited number of processors because of limited bandwidth.
- This problem can be avoided by using multiple buses.



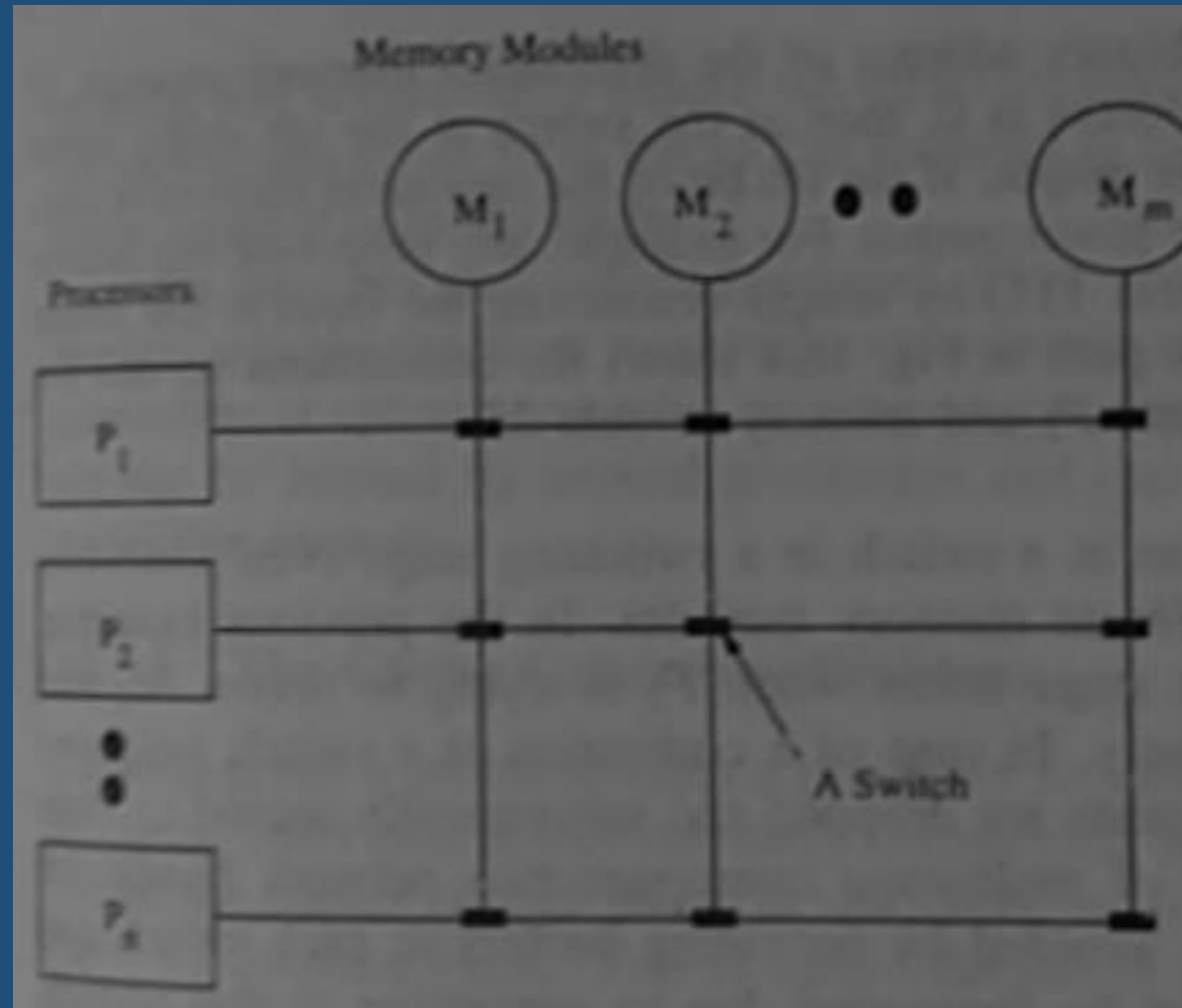
Interconnection Networks

- **Cross-bar Switch**

- It is a **matrix or grid structure** that has a switch at every cross-point.
- A cross-bar is capable of providing an **exclusive connection** between any processor-memory pair.
- All n processors can concurrently access memory modules provided that each processor is accessing a different memory module.
- A cross bar switch does not face contention at the interconnection network level.
- A **contention** can occur only at the memory module level.
- Cross-bar based multiprocessor systems are relatively **expensive** and have **limited scalability** because of the quadratic growth of the number of switches with the system size. ($n \times n$ if there are n processors and n memory modules).
- Crossbar needs N^2 switches for fully connected network between processors and memory.

Interconnection Networks

- Cross-bar Switch



Interconnection Networks

- **Multistage Interconnection Network**

- It is a compromise between a bus and a cross-bar switch.
- A multistage interconnection network permits simultaneous connections between several processor-memory pairs and is more cost-effective than a cross-bar.
- A typical multistage interconnection network consists of several stages of switches.
- Each stage consists of an equal number of cross-bar switches of the same size.
- The outputs of the switches in a stage is connected to the inputs of the switches in the next stage.
- These connection are made in such a way that any input to the network can be connected to any output of the network.

Interconnection Networks

- **Multistage Interconnection Network**

- Depending upon how output-input connections between adjacent stages are made, there are numerous types of interconnection network.
- An $N \times N$ multistage interconnection network can connect N processors to N memory modules.
- In multistage switch based system all inputs are connected to all outputs in such a way that no two-processor attempt to access the same memory at the same time.
- But the problem of contention, at a switch, arises when some memory modules are contested by some fixed processor.
- In this situation only one request is allowed to access and rest of the requests are dropped.
- The processor whose requests were dropped can retry the request or if buffers are attached with each switch the rejected request is forwarded by buffer automatically for transmission.
- This Multistage interconnection networks also called store-and-forward networks.

Interconnection Networks

- Multistage Interconnection Network

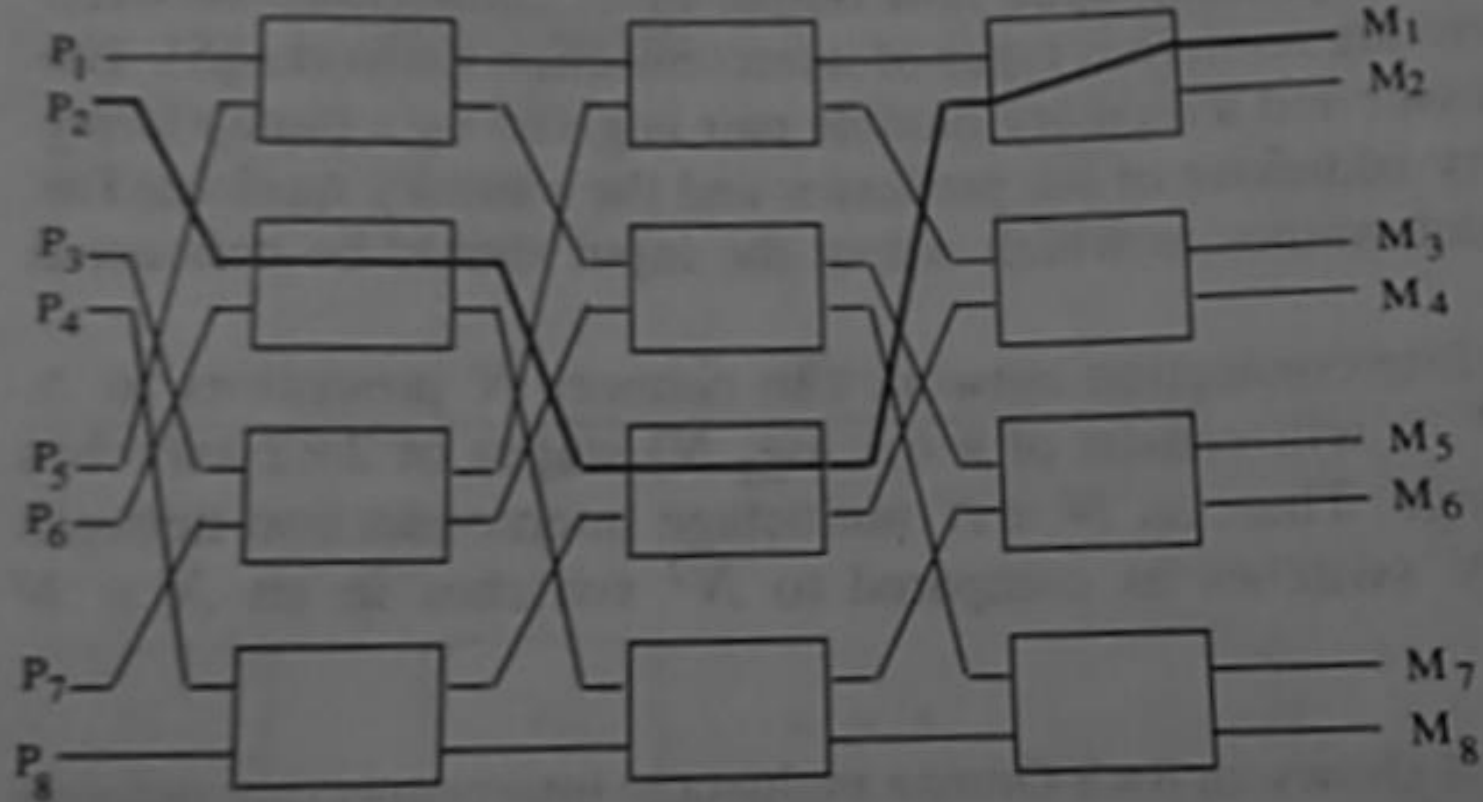


FIGURE 16.4

An 8×8 Omega multistage interconnection network.

Structures of Basic Multiprocessor Operating System

Structures of Basic Multiprocessor Operating System

- The multiprocessor operating systems are complex in comparison to multiprograms on an uniprocessor operating system because multiprocessor executes tasks concurrently.
- Therefore, it must be able to support the concurrent execution of multiple tasks to increase processors performance.
- Depending upon the control structure and its organisation the three basic types of multiprocessor operating system are:
 - 1) Separate supervisor
 - 2) Master-slave
 - 3) Symmetric Supervision

Structures of Basic Multiprocessor Operating System

1) Separate supervisor

- In separate supervisor system each process behaves independently.
- Each system has its own operating system which manages local input/output devices, file system and memory well as keeps its own copy of kernel, supervisor and data structures, whereas some common data structures also exist for communication between processors.
- The access protection is maintained, between processor, by using some synchronization mechanism like semaphores.
- Such architecture will face the following problems:
 - 1) Little coupling among processors.
 - 2) Parallel execution of single task.
 - 3) During process failure it degrades.
 - 4) Inefficient configuration as the problem of replication arises between supervisor/kernel/data structure code and each processor.

Structures of Basic Multiprocessor Operating System

2) Master-slave

- In master-slave, out of many processors one processor behaves as a master whereas others behave as slaves.
- The master processor is dedicated to executing the operating system. It works as scheduler and controller over slave processors. It schedules the work and also controls the activity of the slaves.
- Therefore, usually data structures are stored in its private memory.
- Slave processors are often identified and work only as a schedulable pool of resources, in other words, the slave processors execute application programmes.

Structures of Basic Multiprocessor Operating System

2) Master-slave

- This arrangement allows the parallel execution of a single task by allocating several subtasks to multiple processors concurrently.
- Since the operating system is executed by only master processors this system is relatively simple to develop and efficient to use.
- Limited scalability is the main limitation of this system, because the master processor become a bottleneck and will consequently fail to fully utilise slave processors.

Structures of Basic Multiprocessor Operating System

2) Symmetric

- In symmetric organisation all processors configuration are identical.
- All processors are autonomous and are treated equally.
- To make all the processors functionally identical, all the resources are pooled and are available to them.
- This operating system is also symmetric as any processor may execute it. In other words there is one copy of kernel that can be executed by all processors concurrently.
- To that end, the whole process is needed to be controlled for proper interlocks for accessing scarce data structure and pooled resources.

Structures of Basic Multiprocessor Operating System

2) Symmetric

- The simplest way to achieve this is to treat the entire operating system as a critical section and allow only one processor to execute the operating system at one time.
- This method is called 'floating master' method because in spite of the presence of many processors only one operating system exists.
- The processor that executes the operating system has a special role and acts as a master.
- As the operating system is not bound to any specific processor, therefore, it floats from one processor to another.
- Parallel execution of different applications is achieved by maintaining a queue of ready processors in shared memory.
- Processor allocation is then reduced to assigning the first ready process to first available processor until either all processors are busy or the queue is emptied.
- Therefore, each idled processor fetches the next work item from the queue.

Design Issues

Operating System Design Issues

- A multiprocessor operating system encompasses all the functional capabilities of the operating system of a multiprogrammed uniprocessor system.
- The design of a multiprocessor system is complicated due to following reasons:
 - Able to support Concurrent task Execution
 - Able to exploit the power of multiprocessors
 - It should fail gracefully
 - It should work correctly despite physical concurrency in the execution of processes.

Operating System Design Issues

- The Design of multiprocessor operating system involves the following major issues:
 - Threads
 - Process Synchronization
 - Processor Scheduling
 - Memory Management
 - Reliability and Fault Tolerance

Operating System Design Issues

- **Threads**

- Threads have been widely utilized in recent systems to run applications concurrently on many processors.
- **Thread** is a single sequence stream within a process.
- **Threads** have same properties as of the process so they are called as light weight processes.
- **Threads** are executed one after another but gives the illusion as if they are executing in parallel.
- The effectiveness of parallel computing depends greatly on the performance of the primitives that are used to express and control parallelism.

Operating System Design Issues

- **Process Synchronization**

- A synchronization mechanism must be carefully designed so that it is efficient.
- A more elaborate mechanism that is based on shared variables is needed.
- In a multiprocessor operating system, disabling interrupts is not sufficiently to synchronize concurrent access to shared data.

- **Processor Scheduling**

- To ensure the efficient use of its hardware, a multiprocessor operating system must be able to utilize the processors effectively in executing the tasks.

Operating System Design Issues

- **Memory Management**

- The design of virtual memory is complicated because the main memory is shared by many processors.
- The OS must maintain a separate map table for each processor for address translation.
- When several processors share a page or segment, the OS must enforce the consistency of their entries in respective map table.
- Efficient page replacement is a complex issue.

Operating System Design Issues

- **Reliability and Fault Tolerance**

- The performance of a multiprocessor system must be able to degrade gracefully in the event of failures.
- So a multiprocessor OS must provide reconfiguration schemes to restructure the system in the face of failures to ensure graceful degradation.

Threads

Threads

- **Thread** is an execution unit that consists of its own program counter, a stack of activation records, and a control block.
- The control block contains the state information necessary for thread management such as putting a thread into a ready list and synchronizing with other threads.
- Threads are also known as **Lightweight processes**.
- Threads are a popular way to improve the performance of an application through **parallelism**.
- The CPU switches rapidly back and forth among the threads giving the illusion that the threads are running in parallel.

Threads

- As each thread has its own independent resource for process execution; thus Multiple processes can be executed parallelly by increasing the number of threads.
- It is important to note here that each thread belongs to exactly one process and outside a process no threads exist.
- Each thread basically represents the flow of control separately.
- In the implementation of network servers and web servers threads have been successfully used.
- Threads provide a suitable foundation for the parallel execution of applications on shared-memory multiprocessors.

Advantages of Thread

- Some advantages of thread are given below:
 - Responsiveness
 - Communication
 - Resource sharing, hence allowing better utilization of resources.
 - Economy. Creating and managing threads becomes easier.
 - Scalability. One thread runs on one CPU. In Multithreaded processes, threads can be distributed over a series of processors to scale.
 - Context Switching is smooth. Context switching refers to the procedure followed by the CPU to change from one task to another.
 - Enhanced Throughput of the system. Let us take an example for this: suppose a process is divided into multiple threads, and the function of each thread is considered as one job, then the number of jobs completed per unit of time increases which then leads to an increase in the throughput of the system.

Thread

- Thread is a single sequence stream within a process. Threads have same properties as of the process so they are called as light weight processes. Threads are executed one after another but gives the illusion as if they are executing in parallel. Each thread has different states. Each thread has
 - A program counter
 - A register set
 - A stack space
- Threads are not independent of each other as they share the code, data, OS resources etc.

Similarity between Threads and Processes –

- Only one thread or process is active at a time
- Within process both execute sequentiall
- Both can create children

Differences between Threads and Processes –

- Threads are not independent, processes are.
- Threads are designed to assist each other, processes may or may not do it

Types of Thread

- There are two types of threads:
 - User Level Threads
 - Kernel Level Threads
- **User threads** are above the kernel and without kernel support. These are the threads that application programmers use in their programs.
- **Kernel threads** are supported within the kernel of the OS itself.
- All modern OSs support kernel-level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.

User Level Threads

- A run-time library package provides the routines necessary for thread management operations.
- These routines are linked at runtime to applications.
- Kernel intervention is not required for the management of threads.
- They are not created using the system calls.
- Low cost

Advantages of ULT –

- Can be implemented on an OS that doesn't support multithreading.
- Simple representation since thread has only program counter, register set, stack space.
- Simple to create since no intervention of kernel.
- Thread switching is fast since no OS calls need to be made.
- They are flexible.

User Level Threads

Disadvantages of ULT –

- No or less co-ordination among the threads and Kernel.
- If one thread causes a page fault, the entire process blocks.

Kernel Level Threads

- Kernel knows and manages the threads.
- Instead of thread table in each process, the kernel itself has thread table (a master one) that keeps track of all the threads in the system.
- In addition kernel also maintains the traditional process table to keep track of the processes.
- OS kernel provides system call to create and manage threads.

Kernel Level Threads

Advantages of KLT –

- Since kernel has full knowledge about the threads in the system, scheduler may decide to give more time to processes having large number of threads.
- Good for applications that frequently block.

Disadvantages of KLT –

- Slow and inefficient.
- It requires thread control block so it is an overhead.

Process – Synchronization

Process – Synchronization

- The execution of a concurrent program on a multiprocessor system may require the processors to access shared data structures and thus may cause the processors to concurrently access a location in the shared memory.
- So a mechanism is needed to serialize shared data access to guarantee the correctness of data.
- This is the classic Mutual Exclusion Problem

Issues in Process – Synchronization

- Numerous solutions exist for uniprocessor systems but these are not suitable for a multiprocessor system.
- This is because busy-waiting by processors can cause excessive traffic on the interconnection network. Thereby degrades System performance.
- To overcome this problem, multiprocessor systems provide *instructions to automatically read and write a single memory location.*
- *Mutual exclusion can be implemented completely in hardware* provided the operation on the shared data is elementary.

Issues in Process – Synchronization

- If an access to a shared data constitutes several instructions, then *primitives such as lock and unlock (P and V) operations are needed to ensure mutual exclusion.*
- Atomic machine language instructions can be used to implement the lock operation, which automatically serialize concurrent attempts to acquire a lock.

Issues in Process – Synchronization

- Several Atomic Hardware Instructions to Implement P and V operations are:
 1. The Test-and-Set Instruction
 2. The Swap Instructions
 3. The Fetch-and-Add Instruction of the Ultracomputer
 4. SLIC Chip of the Sequent
 5. Implementation of Process Wait
 6. The Compare-and-Swap Instruction

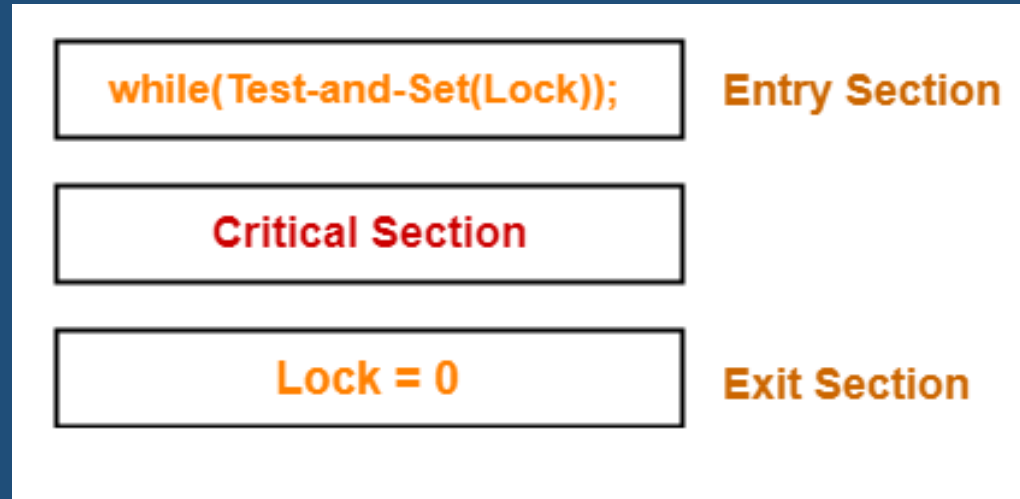
Issues in Process – Synchronization

1. The Test-and-Set Instruction

- It uses a test and set instruction to provide the synchronization among the processes executing concurrently.
- It is an instruction that returns the old value of a memory location and sets the memory location value to 1 as a single atomic operation.
- If one process is currently executing a test-and-set, no other process is allowed to begin another test-and-set until the first process test-and-set is finished.

Issues in Process – Synchronization

1. The Test-and-Set Instruction



- Lock value = 0 means the critical section is currently vacant and no process is present inside it.
- Lock value = 1 means the critical section is currently occupied and a process is present inside it.

Issues in Process – Synchronization

1. The Test-and-Set Instruction

- Here, the shared variable is lock which is initialized to false.
- TestAndSet(lock) algorithm works in this way – it always returns whatever value is sent to it and sets lock to true.
- The first process will enter the critical section at once as TestAndSet(lock) will return false and it'll break out of the while loop.
- The other processes cannot enter now as lock is set to true and so the while loop continues to be true.
- Mutual exclusion is ensured.
- Once the first process gets out of the critical section, lock is changed to false. So, now the other processes can enter one by one.
- Progress is also ensured.
- However, after the first process any process can go in.
- There is no queue maintained, so any new process that finds the lock to be false again, can enter. So bounded waiting is not ensured.

Issues in Process – Synchronization

1. The Test-and-Set Instruction

The set-and-test instruction automatically reads and modifies the contents of a memory location in one memory cycle. It is defined as follows:

```
function Test-and-Set (Var m:boolean); boolean; // m – memory location  
begin  
    Test-and-Set:=m;  
    m:=true  
end;
```

The test-and-set instruction returns the current value of variable m and sets it to true.

Issues in Process – Synchronization

1. The Test-and-Set Instruction

- This instruction can be used to implement P and V operations on a binary semaphore, S, in the following way:

P(S): while Test-and-Set(S) do nothing;

V(S): S:=false;

- Initially, S is set to false. When a P(S) operation is executed for the first time, Test-and-Set(S) returns a false value (and sets S to true) and the “while” loop of the P(S) operation terminates.
- All subsequent execution of P(S) keep looping because S is true until a V(S) operation is executed.

Issues in Process – Synchronization

2. The Swap Instruction

- The swap instruction automatically exchanges the contents of two variables (Eg: Memory Locations).
- It is defined as follows (x and y are two variables):

```
Procedure swap(var x,y: boolean);
```

```
Var temp: boolean;
```

```
begin
```

```
    temp:=x;
```

```
    x:=y;
```

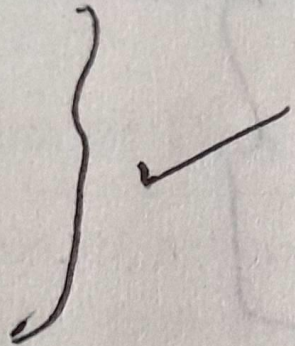
```
    y:=temp;
```

```
end;
```

Swap Instructions

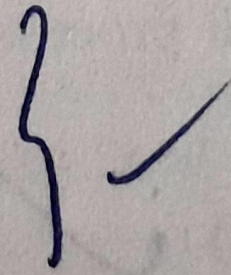
...rows (x and y are two variables):

```
procedure swap(var x, y: boolean);  
var temp: boolean;  
begin  
    temp := x;  
    x := y;  
    y := temp;  
end;
```



P and V operations can be implemented using the swap instruction in the following way (p is a variable private to the processor and S is a memory location):

```
P(S): p := true;  
repeat swap(S, p) until p = false;  
V(S): S := false;
```

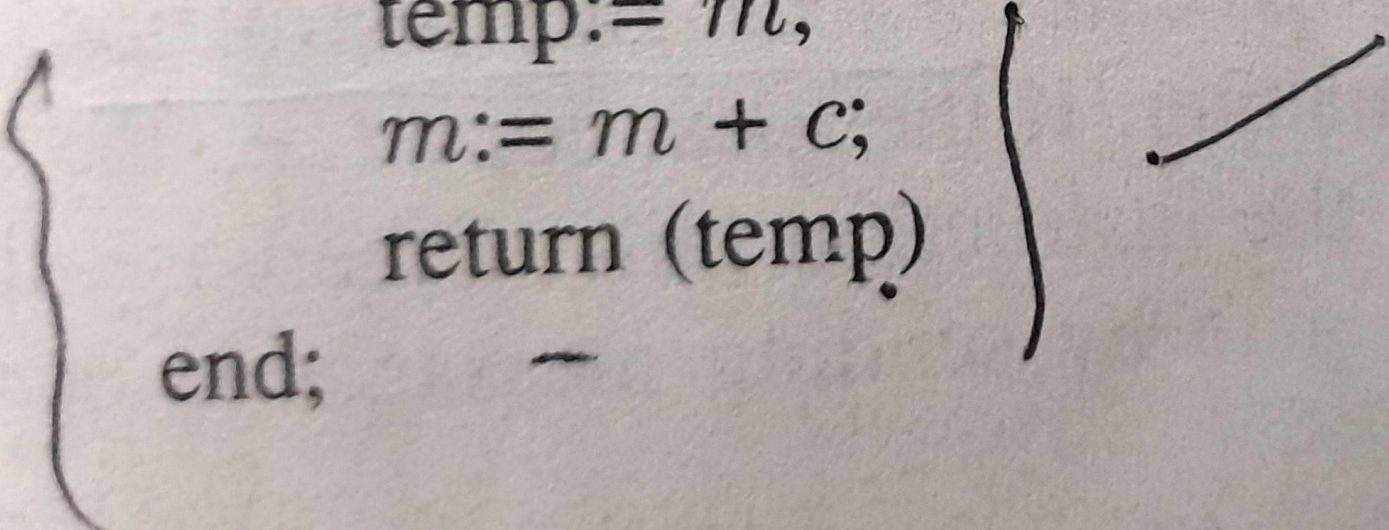


Drawbacks of Test-and-Set instructions and Swap Instruction

- The above two implementations of the P operation employ busy-waiting and therefore increases the traffic on the interconnection network.
- Another problem is that if n processors execute any of these operations on the same memory location, the main memory will perform n such operations on the location even though only one of these operations will succeed.

3. The Fetch-and-Add Instruction

```
Function Fetch-and-Add(m: integer; c: integer);  
var temp: integer;  
begin  
    temp := m;  
    m := m + c;  
    return (temp);  
end;
```



3. The Fetch-and-Add Instruction

```
P(S): while (Fetch-and-Add(S, -1) < 0) do  
begin  
    Fetch-and-Add(S, 1);  
    while (S ≤ 0) do nothing;  
end;
```

Scanned by TapScanner

```
V(S): Fetch-and-Add(S, 1)
```

Scanned by TapScanner

4. SLIC chip of the sequent

- Uses atomic multi-operation machine language instructions.
- The main component is a SLIC (system link and interrupt controller) Chip that supports many other functions in addition to low-level mutual exclusion.
- A SLIC chip contains 64 single-bit registers and supports the operation necessary for process synchronization.
- Each processor has a SLIC chip and all the SLIC chips are connected by a separate SLIC bus.
- Each bit in the SLIC chip called a gate acts as a separate lock and stores the status of corresponding lock.
- This status is get replicates over all the processors instead of keeping them at a central place.
- Thus this method substantially reduces traffic on the network that connects memory modules in the processors and reduce access time.

SLIC chip of the sequent

- To lock a gate in the SLIC chip, a processor executes a lock-gate instruction.
- If the local copy indicates the gate is closed, the instruction fails.
- Otherwise the local SLIC of the processor attempts to close the gate by sending messages to other SLIC Chips over the SLIC bus.
- The following code implements P and V operations on a semaphore:

```
P(S): while (lock-gate(S) = failed) do nothing;  
V(S): unlock-gate(S);
```

Scanned by TapScanner

5. Implementation of process wait

- **Busy waiting**
 - Continuously Execute to check the status.
 - Spin Lock
- **Sleep-lock**
 - A process is suspended when it fails to obtain a lock.
 - All inter process interrupts are disabled.
 - Reduces network traffic.
- **Queuing**
 - Global Queue
 - A waiting process is dequeued and activated by a V operation on the semaphore

6. Compare - and - Swap instruction

r1 and r2 are two registers of a processor and m is memory location

```
Compare-and-Swap(var r1, r2, m: integer);  
var temp: integer;  
begin  
    temp := m;  
    if temp = r1 then {m := r2; z := 1}  
    else {r1 := temp; z := 0}  
end;
```

Scanned by TapScanner

6. Compare - and - Swap instruction

- The compare and swap instruction can be used to synchronize concurrent access to a shared variable, say m .
- A processor first reads the value of m into a register $r1$.
- It then computes a new value, which is x plus the original value to be stored in m and stores it in register $r2$.
- The processor then performs a compare-and-swap ($r1,r2,m$) operation.
- If $z=1$ – no other process has modified location m since it was read by this processor.
- Thus mutually exclusive access to m is maintained.
- If $z=0$, then other processor has modified m .

Compare - and - Swap instruction

```
    r1 := m  
label: r2 := r1 + x  
    compare-and-swap(r1, r2, m)  
    if .z = 0 then go to label
```

Processor Scheduling

Processor scheduling

- A parallel program is a task force consisting of several tasks.
- In processor scheduling, Ready tasks are assigned to the processors so that performance is maximized.
- These tasks may belong to a single program or they may come from different programs.

Issues in processor scheduling

- Preemption inside Spinlock-controlled critical sections –This situation occurs when a task is pre-empted inside a critical section when there are other tasks spinning the lock to enter the same critical section. These tasks waste CPU cycles.
- Cache corruption - if tasks executed successively by a processor come from different applications, it is likely that on every task switch, a big chunk of data needed by the previous tasks must be purged from the cache and new data must be brought to the cache. High miss ratio whenever a processor switches to another task
- Context switching overheads – entails the execution of a large number of instructions to save and store registers, to initialize registers, to switch address space etc.

Scheduling strategies

Several Multiprocessor scheduling strategies that address the above issues are:

- Co-scheduling of the Medusa OS
- Smart scheduling
- Scheduling in the NYU Ultracomputer
- Affinity based scheduling
- Scheduling in the Mach Operating System

Scheduling strategies

Co-scheduling of the Medusa OS

- All runnable tasks of an application are scheduled on the processors simultaneously.
- Whenever a task of an application need to be pre-empted, all the task of that applications are pre-empted.
- Co-scheduling does context switching between applications rather than between task of several different applications.
- that is all the task in an application are Run for a Time slice, then all the task in another application are Run for another Time slice, and so on.
- Co-scheduling removes the problem of tasks waiting resources in lock-spinning while they wait for a pre-empted task to release the critical section.
- It does not remove the overhead due to context switching nor performance decoration due to cache corruption.

Scheduling strategies

Smart scheduling

- Smart scheduler has 2 features:
 - First it avoids the preempting a task when the task is inside is critical section.
 - Second, it avoids the rescheduling of tasks that were busy-waiting at the time of their preemption until the task that is executing the corresponding critical section releases it.
 - When a task enters a critical section, it sets a flag.
 - The scheduler does not preempt a task if its flag is set.
 - On exit from a critical section, a task resets the flag.
 - The smart scheduler eliminates the resource waste due to processor spinning.
 - Doesnot reduce the overhead due to context switching.

Scheduling strategies

Scheduling in the NYU Ultracomputer

- It combines the strategies of the previous two scheduling techniques.
- In this technique tasks can be formed into groups and the task in a group can be scheduled in any of the following ways:
 - A task can be scheduled or preempted in the normal manner.
 - All the task in a group are scheduled or preempted simultaneously.
 - Task in a group are never preempted.
- The scheduling techniques is flexible.

Scheduling strategies

Affinity based scheduling

- It is the first scheduling policy to address the problem of cache corruption.
- In this policy the task is scheduled on the processor where it last executed.
- This policy removes the problem of cache corruption because it is likely that a significant portion of the working set of that task is present in the cache of the processor when the task is rescheduled.
- Affinity based scheduling also decreases bus traffic due to cache reloading.
- It does not restrict load balancing among processes because it cannot be scheduled on any processor.

Scheduling strategies

Scheduling in the Mach Operating System

- In the Mach operating system, an application or a task consists of several threads.
- It is the smallest independent unit of execution and scheduling in Mach.
- In the mach operating system, all the processes of a multiprocessor are grouped in in disjointed set called processors set.
- The processes in a processor set are assigned a subset of threads for execution.
- These processes use priority scheduling to execute the Threads are find to their processor set.
- Can have priority ranging from 0 to 31, where 0 and 31 are the highest and the lowest priorities.
- Each processor set has an array of 32 ready queues- one queue to store the ready Threads of each priority.
- When a thread with priority i becomes ready, it is appointed to the i th queue.
- Every processor has a local ready queue that consists of the Threads that must be executed only by that processor.
- Two level priority
 - All the threads in a local queue have priority over all the threads in the global queue and there are also priorities inside is each of these two queus.

Memory management

- Memory management in multiprocessor operating system.
- Issues in the design of memory management in multiprocessor operating system are:

Design Issues

- **Portability**
- **Data sharing**
- **Protection**
- **Efficiency**

Memory management

- **Portability**

- Portability implies the ability of an operating system to run on several machines with different architectures.
- For widespread applicability of an Operating System, Architecture-Independence should be an important consideration in the design of a virtual memory system.

- **Data sharing**

- In multiprocessor systems, an application is typically executed as a collection of processes that run on different processors.
- These processes generally shared data for communication and synchronisation.
- A virtual memory system must provide facility for flexible data sharing to support the execution of parallel programs.

Memory management

- **Protection**

- When memory is shared among several processes, memory protection becomes an important requirement.
- The operating system must support mechanism that a virtual memory system can employ to protect memory object against unauthorised access.

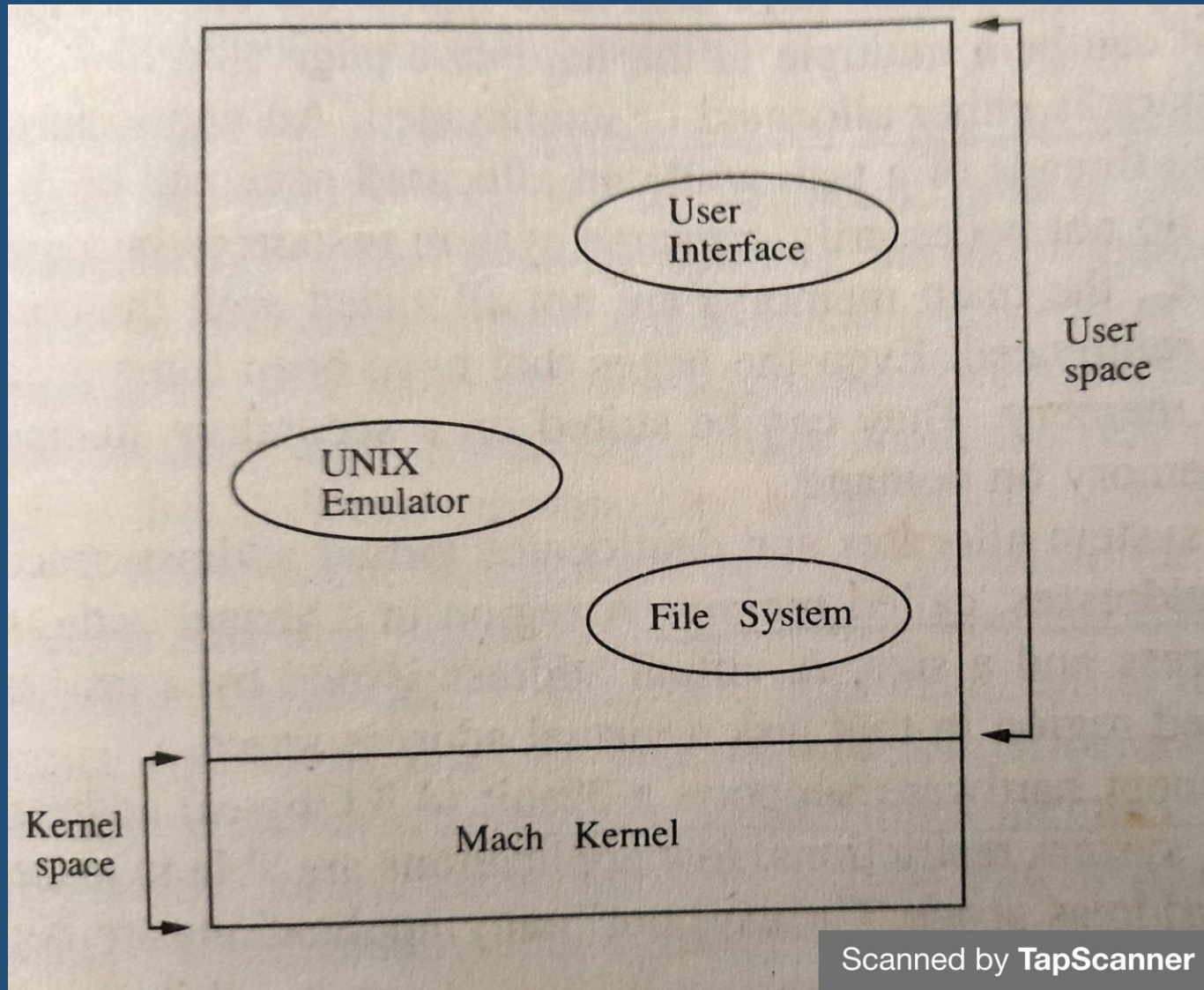
- **Efficiency**

- A virtual memory system can become a bottleneck and limit the performance of the multiprocessor operating system.
 - Ever said memory system must be efficient in performing address translations, page table lookups, page replacements, etc.
 - It should run in parallel to take advantage of multiple processors.
- The Mach operating system is designed for parallel and distributed environment.
 - It can run on multiprocessor systems and support the execution of parallel applications.

Mach Kernel

- Key component of Mach OS
- Provide primitives for,
 - Building parallel and distributed applications
 - Process management
 - Memory management
 - Interprocess communication
 - I/O services
- Other operating system services, which are useful to developers or end users, are built on top of the Mach Kernel.

Mach OS



Mach Kernel supports Five abstractions

- Threads
 - A thread is the smallest independent unit of execution in Mach.
 - A thread has a program counter and a set of registers.
- Tasks
 - A task is an execution environment that may consist of many threads.
 - A task includes a paged virtual address space and protected access to the system resources.
 - A task is the basic unit of resource allocation.
- Ports
- Messages
- Memory objects

Mach Kernel supports Five abstractions

- Ports
 - A port is a unidirectional channel associated with an object (example task, thread) that queues up messages for that object.
 - Task and Threads communicate with other task and Threads by performing send and receive operations on their ports,
 - A port is protected in the Kernel to ensure that only authorised task of threads can read or write to a port.
- Messages
 - A message is a typed collection of data used by Threads for communication.
 - Messages may be of an arbitrary size and can contain pointers and capabilities.
- Memory objects
 - A memory object is a contiguous repository of data, indexed by byte, upon which various operations such as read and write can be performed.
 - Memory objects act as a secondary storage in the Mac Operating System.

Task address space

- A Page in a task address space is either allocated / unallocated.
- Regions : Continuous chunks of virtual addresses.
 - A region in a virtual address space is specified by a base address and a size.
 - A virtual address issued by a task is valid only if it falls in an allocated region in that task's virtual address space.
- The Mach virtual memory system support several operations that are often needed in advance application such as:
 - Accessing address space of the task
 - Access address space of other tasks
 - Copy a region within an address space

Memory protection

- Virtual memory protection is enforced at the page level.
- Each allocated page has the following two protection codes associated with it.
- 1) The Current protection code
 - Which corresponds to the protection associated with a page for memory references.
- 2) Maximum protection code
 - Which limits the value of the current protection.
- A page protection consists of a combination of read, write and execute permissions.
- Mach Provides that set the current or maximum protection.
- The current protection can only include the permission specified in the maximum protection.
- The maximum protection can only be lowered. That is permission specified in the maximum protection can be deleted but new permissions cannot be added.

Machine independence

- To support portability across a wide range of architectures, a machine - independent virtual memory system is the goal of Mach virtual memory system.
- Splitting the implementation into two parts:
 - Machine independent part –
 - This split is based on the assumption that there exists a paged memory management unit (MMU) with minimal functionality.
 - responsible for maintaining high level machine independent data structures.
 - In case of page fault, entire mapping information can be constructed from the machine-independent data structures.
 - Machine dependent part –
 - The pmap module(physical address space) is the only machine-depended part in the Mach Virtual memory system.
 - responsible for management of the physical address space.
- The Mach Virtual memory system provides two types of independence to higher layers:
 - OS independence
 - Paging-store independence

Memory sharing

- The ability to share memory among several task is very important for the efficient execution of parallel applications.
- These applications can use shared memory for efficient process synchronisation and interprocess communication.
- Different task can share a page by installing that page in the virtual address space.
- Only one copy of the page is present in main memory
- The Mach virtual memory system allows the sharing of memory via the inheritance mechanism.
- In Mach, a new address space is created when a task is created.

Memory sharing

- The inheritance attribute of a page can take the values:
 - None, Copy and share.
- If a page is in none inheritance mode, the child task does not inherit that page.
- If a page is in the copy mode, the child receives a copy of the page and subsequent modifications to that page only affect the task making the modifications.
- If a page is in the share mode, the same copy of the page is shared between the parent and the child task.

Efficiency considerations

- The Mach virtual memory system uses the following techniques to increase efficiency:
 - Parallel implementation -
 - All algorithms are designed to run in parallel and all data structures are designed to allow a highly parallel access.
 - Simplicity – Use simple algorithms and data structures.
 - Lazy evaluation –
 - The evaluation of a function is postponed as long as possible in the hope that the evaluation will never be needed.
 - to increase time and space efficiency
 - Copy-on-write operation

Implementation: Data structures and Algorithms

The Mach virtual memory system uses four basic data structures:

1. Memory objects:
 - It is repository of data that can be mapped into address space of a task.
2. Pmap structures:
 - Is a Hardware defined physical map that translates a virtual address to a physical address.
3. Resident page tables –Information about physical pages is maintained in a page table also called a resident page table whose entries are indexed by physical page number. A page entry in the page table may be linked into the following list: memory object list, memory allocation queues, object/offset hash bucket
4. Address maps – maps contiguous chunks of virtual addresses.

Algorithms

- The Page Replacement Algorithm:
 - A page replacement algorithm decides which page in the physical memory to replace in the event of a page fault.
 - The replacement algorithm in Mach is a modified-FIFO algorithm that keeps all the physical memory pages in one of the following three FIFO queues:
 1. The free list:
 - This contains pages that are free to use. These pages are not currently allocated to any task and can be allocated to any tough
 2. The active list:
 - This contains all pages that are actively in use by task. when a page is allocated it is removed from the free list and placed at the end of the active list
 3. The inactive list:
 - This contains pages that are not in use in any address space, but very recently in use.
 - Special kernel thread called a **Pageout Daemon** performs page replacement and management of these list.

Algorithms

- The Page fault Handler:
 - Page fault handler is invoked when a page is referenced for which there is either an invalid mapping or a protection violation.
 - The page fault handler has the following responsibilities:
 - 1. Validity and protection
 - 2. Page lookup
 - 3. Hardware validation
- Locking protocols:
 - All algorithms and data structures used in virtual memory implementation are designed to run in a multiprocessor environment and other fully parallel.
 - The synchronisation of accesses to share data structures is achieved by the following locks:
 - 1. Map locks
 - 2. Object locks
 - 3. Hash table bucket locks
 - 4. Busy page locks

Sharing of Memory Objects

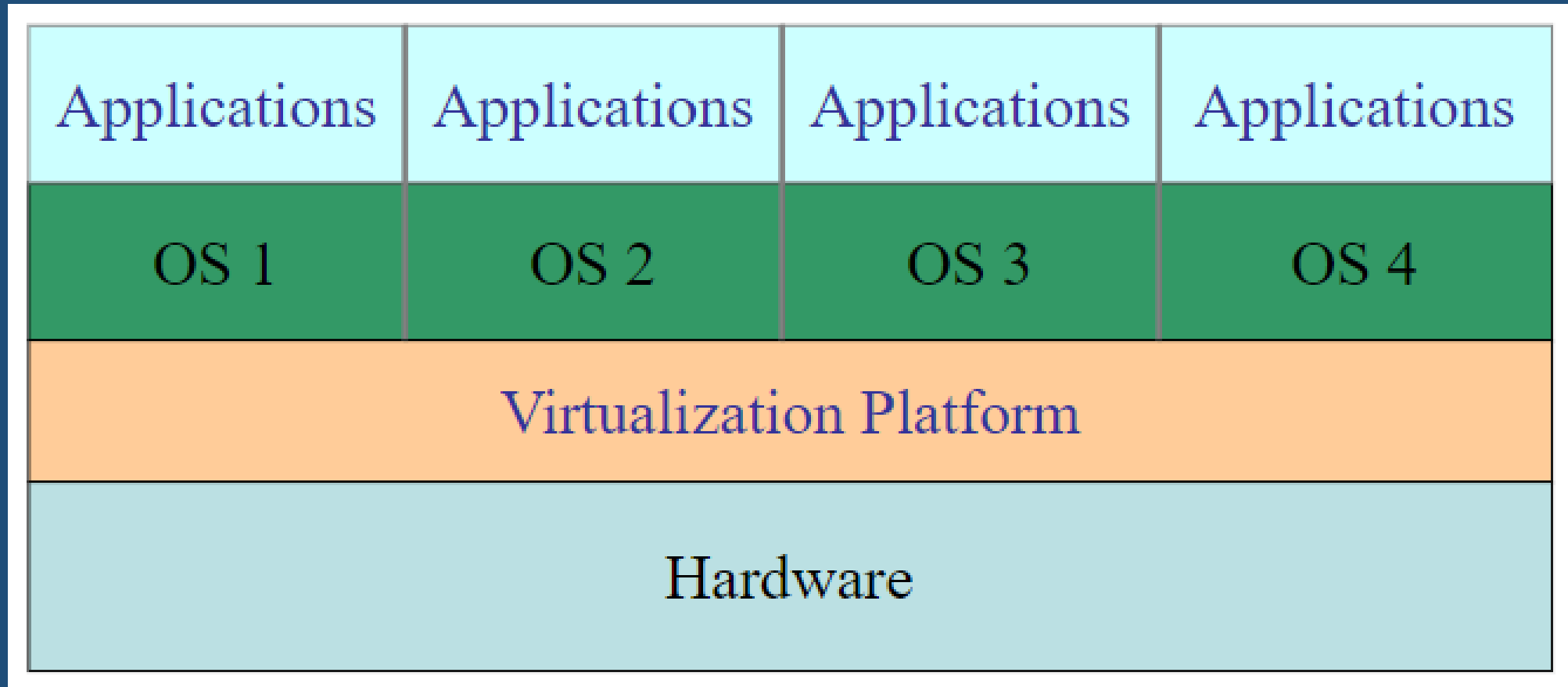
- Supports copy-on-write operations, which allowed the sharing of the same copy of a memory object by several task as long as all the task only read the memory object.
- When a task performed a copy on write operation on a memory object, its address map starts pointing to a new copy of the memory object; That is it shares the same copy with the original owner of the memory object.
- If one of the task writes data in a copied memory object using the copy on write operation, a new page for that data is allocated, which is accessible only to the writing task. This new page contains the modifications by the writing task.
- The system maintains special objects called Shadow objects to hold pages of a memory object that have been modified.
- A shadow object correctly remembers all the modified pages of a memory object copied/ shared using the copy on write operations.
- A shadow object can be shadowed on subsequent copy on write operations, thus creating a chain of Shadows.

Virtualization

Virtualization

- Virtualization in operating system changes a normal operating system so that it can run different types of applications that may be handled on a single computer system by many users.
- The operating system may appear different to each user and each of them may believe they are interacting with the only operating system.
- Virtualization is a technology that helps us to install different Operating Systems on a hardware.
- They are completely separated and independent from each other.
- Virtualization is often – The creation of many virtual resources from one physical resource.

Virtualization



Advantages of Virtualization

- Some of the advantages of virtualization are –
 - Virtualization allows a **finite number of hardware resources** to be easily distributed to multiple processes that require them. (Strong Isolation)
 - **Fewer physical machines** saves money on hardware and electricity and takes up less office space.
 - **Checkpoint and migrating** virtual machines (eg: load balancing across multiple servers) is much easier than migrating processes running on a normal operating system.
 - Can able to **run legacy applications** o operating systems no longer supported or which do not work on current hardware.
 - **Software Development** (To check a programmer to make sure that his software works on different versions from a single computer).
 - **Improved security** can be obtained by using virtualisation. This happens because each process inhabits its own instance of the operating system and works independently.
 - Operating system virtualization is very useful for establishing a **virtual hosting environment**.
 - There is only a **little overhead** involved for operating system virtualization and so it is very beneficial.

Disadvantages of Virtualization

- Some of the disadvantages of virtualization are –
 - **Specialized experts** are required to implement and manage a virtualized system. This results in need for virtualization experts and increased costs.
 - There are many **upfront costs** involved in virtualization. These include the cost for virtualization software as well as the additional hardware required. The costs also depends on the existing system network.

Despite some problems, virtualization is quite useful. It has numerous advantages and its disadvantages are merely simple challenges that can be overcome with the help of experts in operating system virtualization.

Hypervisors

- A **hypervisor**, also known as a **virtual machine monitor or VMM**, is software that creates and runs virtual machines (VMs).
- A **hypervisor** allows one host computer to support multiple guest VMs by virtually sharing its resources, such as memory and processing.
- Hypervisors are two types –
 - Type 1 Hypervisors (Native of Bare Metal Hypervisor)
 - Type 2 Hypervisors (Hosted Hypervisor)

Type 1 Hypervisors

Type 1 Hypervisor Runs on “bare metal”

- Virtual machines run in user mode.
- VM runs the guest OS (which thinks it is running in kernel mode) – Virtual kernel Mode
- If guest OS calls sensitive instructions, hypervisor will trap and execute the instructions.
- If application on guest OS calls sensitive instructions (system calls), hypervisor traps to guest OS.

Type 1 Hypervisors

Type 1 Hypervisors

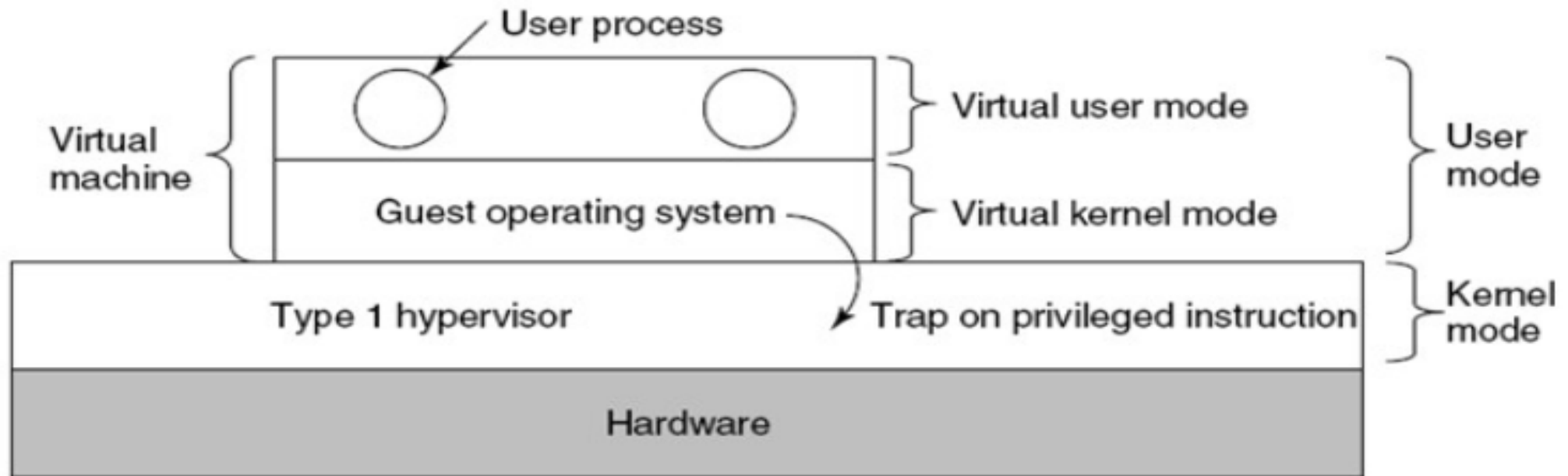
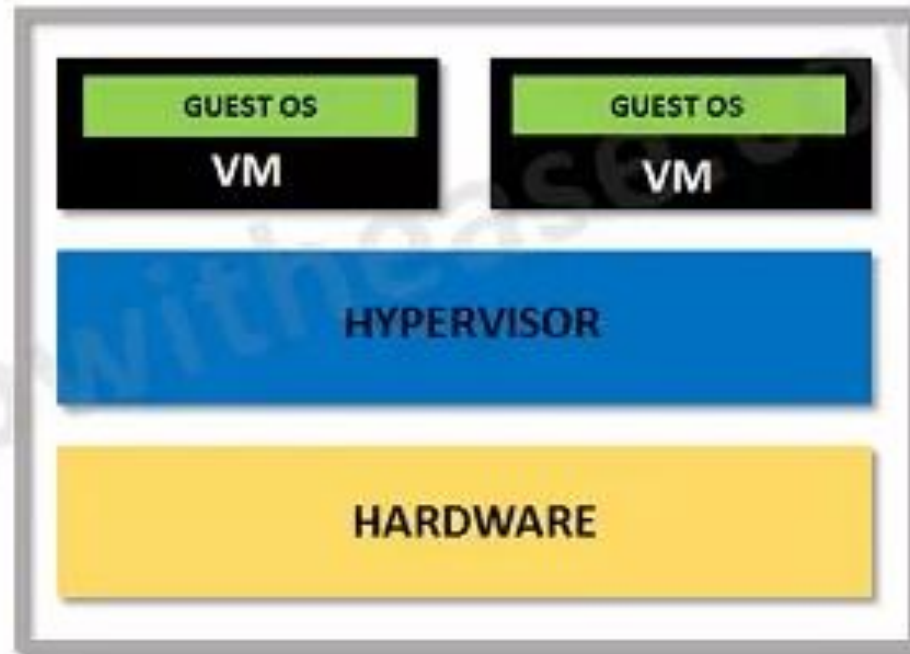


Figure 8-26. When the operating system in a virtual machine executes a kernel-only instruction, it traps to the hypervisor if virtualization technology is present.

Type 1 Hypervisors

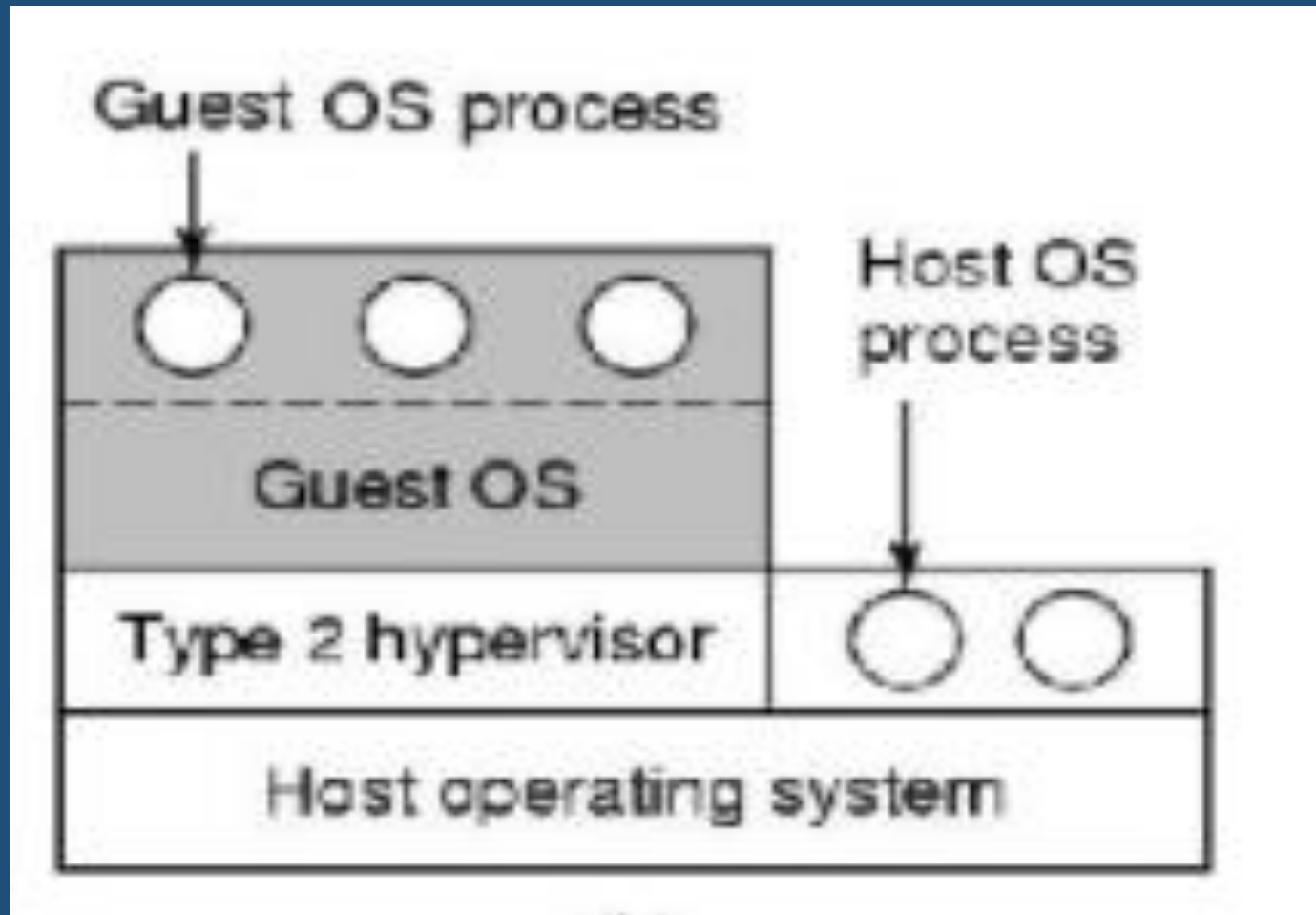
**TYPE 1 HYPERVISOR
(BARE-METAL ARCHITECTURE)**



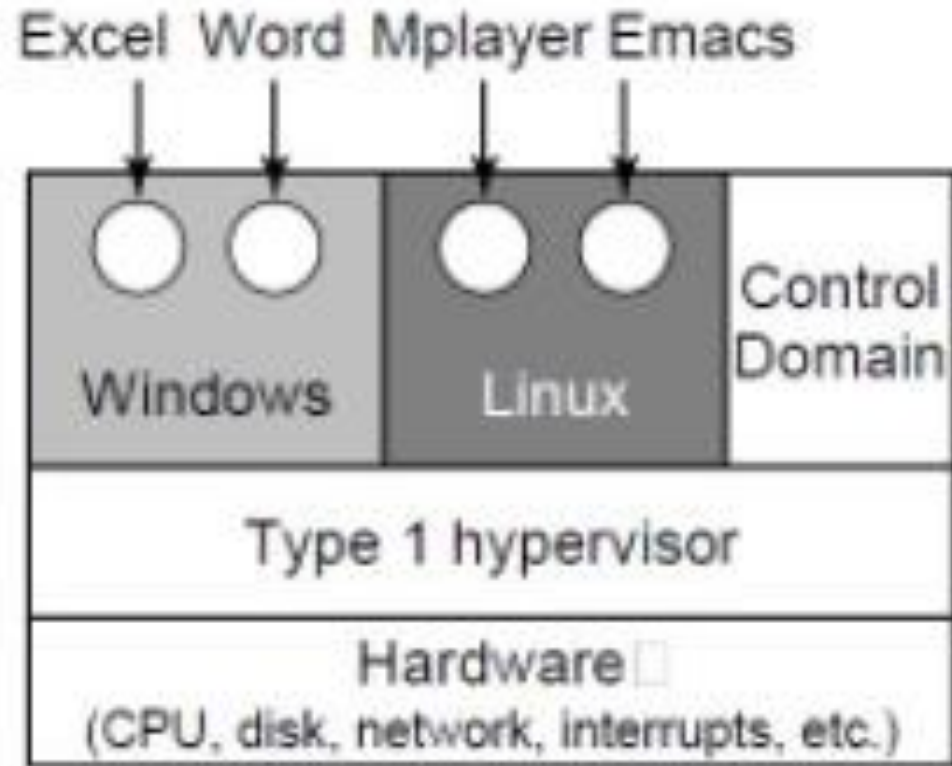
Type 2 Hypervisors

- Runs from within a OS.
- Supports guest OSs above it.
 - Boot from CD to load new OS
 - Read in code, looking for basic blocks
 - Then inspect basic block to find sensitive instructions. If found, replace with VM call (process called binary translation). Then, cache block and execute.
 - Eventually all basic blocks will be modified and cached, and will run at near native speed.

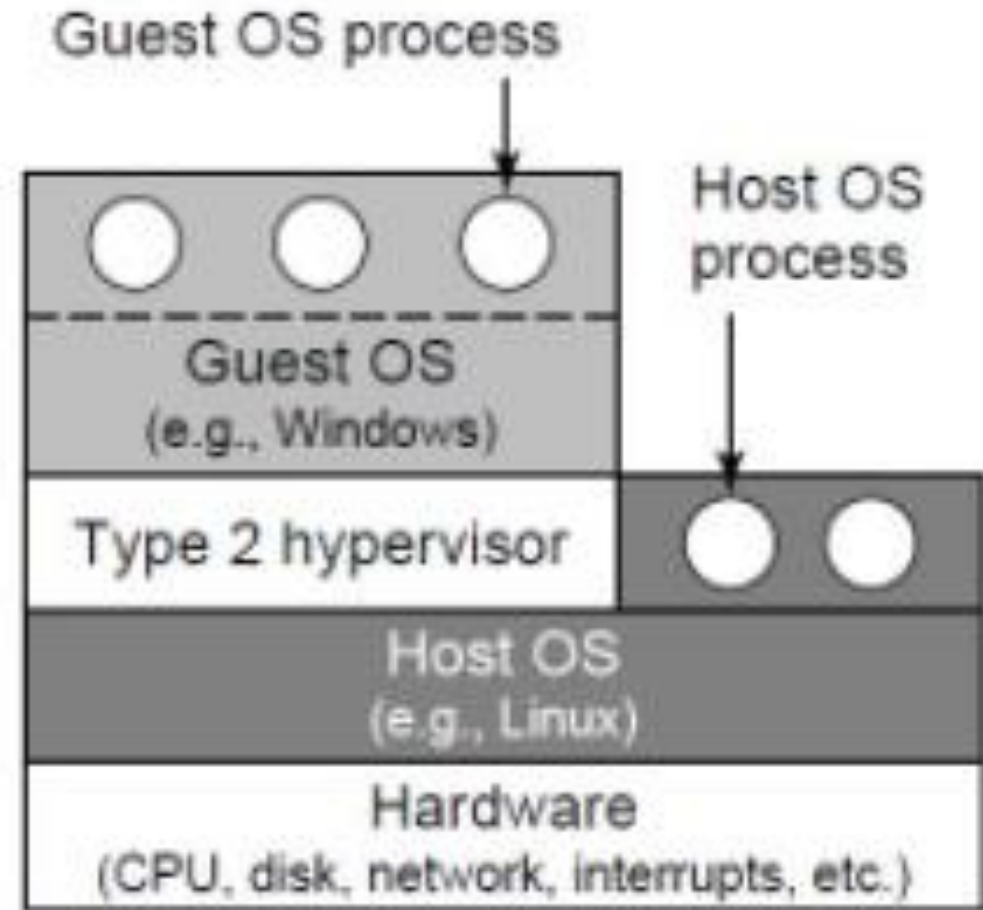
Type 2 Hypervisors



Type 1 hypervisor & Type 2 Hypervisor



(a)

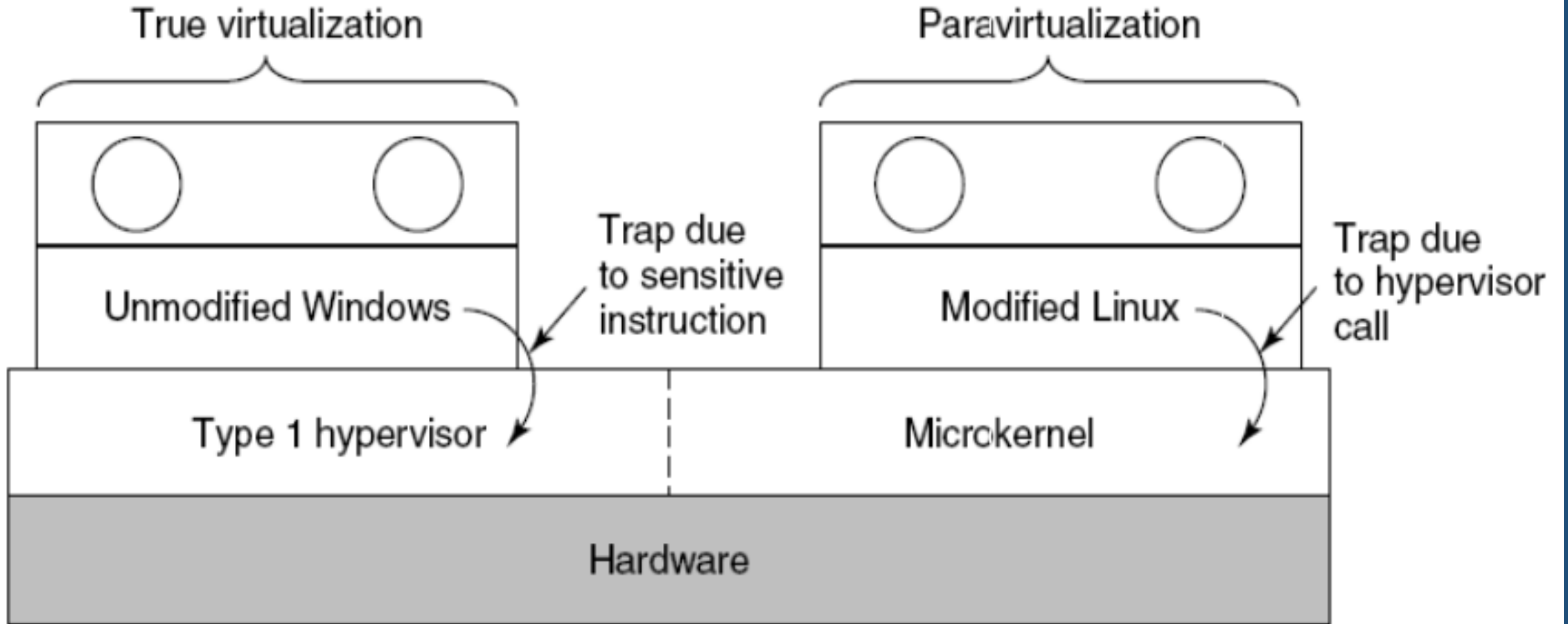


(b)

Paravirtualization

- Modify Guest OS so that all calls to sensitive instructions are changed to hypervisor calls.
- The guest OS is acting like a user program making system calls to the OS.
- Much easier (and more efficient) to modify source code than to emulate hardware instructions (as in binary translation).
- In effect, turns the hypervisor into a microkernel.

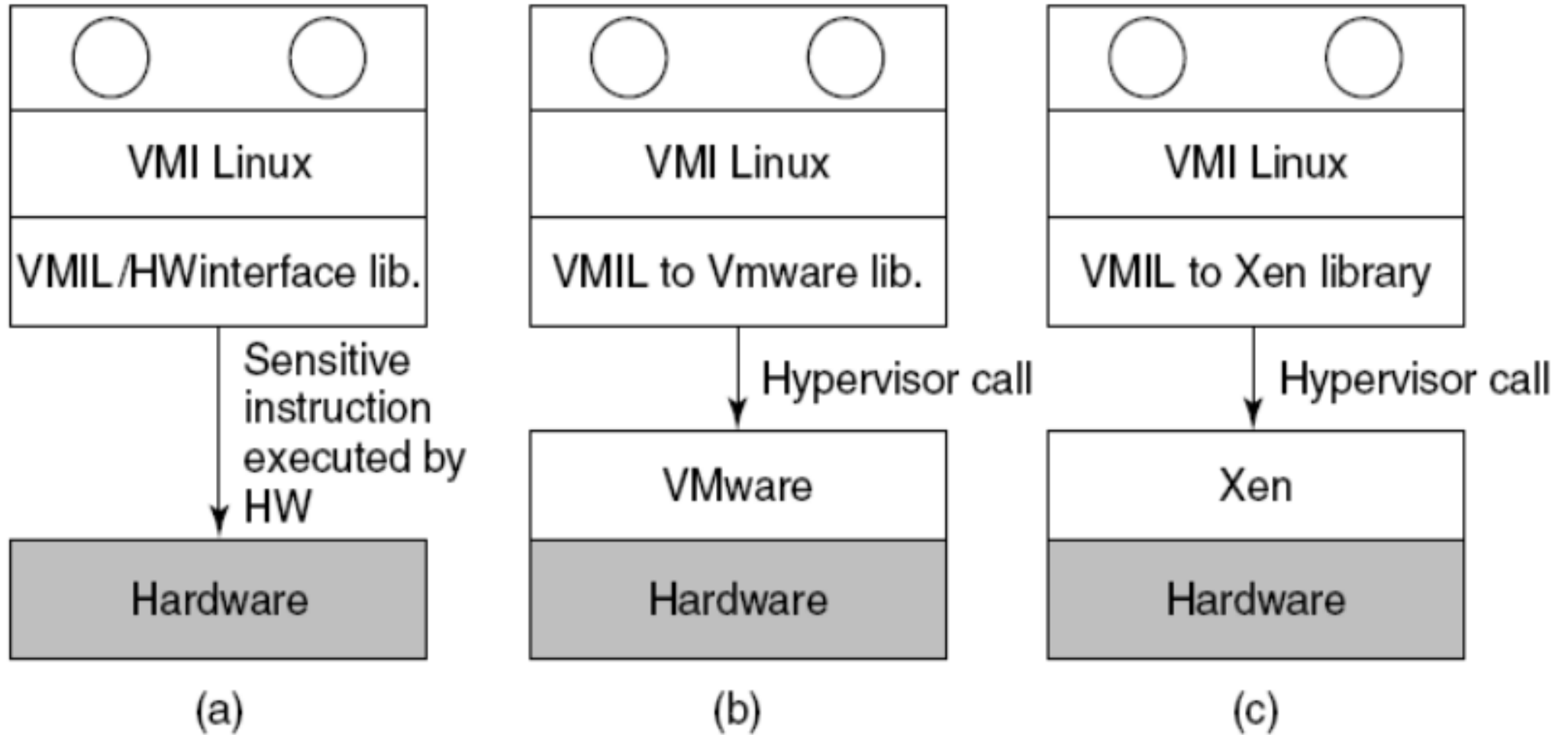
Paravirtualization



Problems with Paravirtualization

- Paravirtualized systems won't run on native hardware.
- There are many different paravirtualization systems that use different commands, etc. – VMware, Xen, etc.
- Proposed solution: – Modify the OS kernel so that it calls a special set of procedures to execute sensitive instructions (Virtual Machine Interface).
 - Bare metal – link to library that implement code.
 - On VM – link to VM specific library.

Problems with Paravirtualization



Memory Virtualization

Memory Virtualization

- Modern operating systems support virtual memory, which is basically a mapping of pages in the virtual address space onto pages of physical memory. This mapping is defined by (multilevel) page tables.
- Typically the mapping is set in motion by having the operating system set a control register in the CPU that points to the top-level page table.
- Suppose, for example, a virtual machine is running, and the guest operating system in it decides to map its virtual pages 7, 4, and 3 onto physical pages 10, 11, and 12, respectively.
- It builds page tables containing this mapping and loads a hardware register to point to the top-level page table.
- This instruction is sensitive. with dynamic translation it will cause a call to a hypervisor procedure; on a paravirtualized operating system, it will generate a hypercall.

Memory Virtualization

- Hypervisor needs to create a **shadow page table** that maps the virtual pages used by the virtual machine onto the actual pages the hypervisor gave it.
- A possible solution is for the hypervisor to keep track of which page in the guest's virtual memory contains the top-level page table.
- It can get this information the first time the guest attempts to load the hardware register that points to it because this instruction is sensitive and traps.
- The hypervisor can create a shadow page table at this point and also map the top-level page table and the page tables it points to as read only.
- A subsequent attempts by the guest operating system to modify any of them will cause a page fault and thus give control to the hypervisor, which can analyze the instruction stream.
- Allow hypervisor to manage page mapping, and use shadow page tables for the VMs

Memory Virtualization

- VMM creates and manages page tables that map virtual pages directly to machine pages
- These tables are loaded into the MMU on a context switch
- VMM page tables are the shadow page tables
- VMM needs to keep its Virtual-Physical
- Map tables consistent with changes made by OS to its Virtual-Physical tables
- VMM maps OS page tables as read only
- When OS writes to page tables, trap to VMM
- VMM applies write to shadow table and OS table

I/O Virtualization

I/O Virtualization

- The guest operating system will typically start out probing the hardware to find out what kinds of I/O devices are attached.
- These probes will trap to the hypervisor.
- One approach is for it to report back that the disks, printers, and so on are the ones that the hardware actually has.
- The guest will then load device drivers for these devices and try to use them.
- When the device drivers try to do actual I/O, they will read and write the device's hardware device registers.
- These instructions are sensitive and will trap to the hypervisor, which could then copy the needed values to and from the hardware registers, as needed.

Summary

- Virtualization provides a way to consolidate OS installations onto fewer hardware platforms
- 3 basic approaches
 - type 1 hypervisor
 - type 2 hypervisor
 - Paravirtualization
- Must also account for virtual access to shared resources (memory, I/O)

Thank You