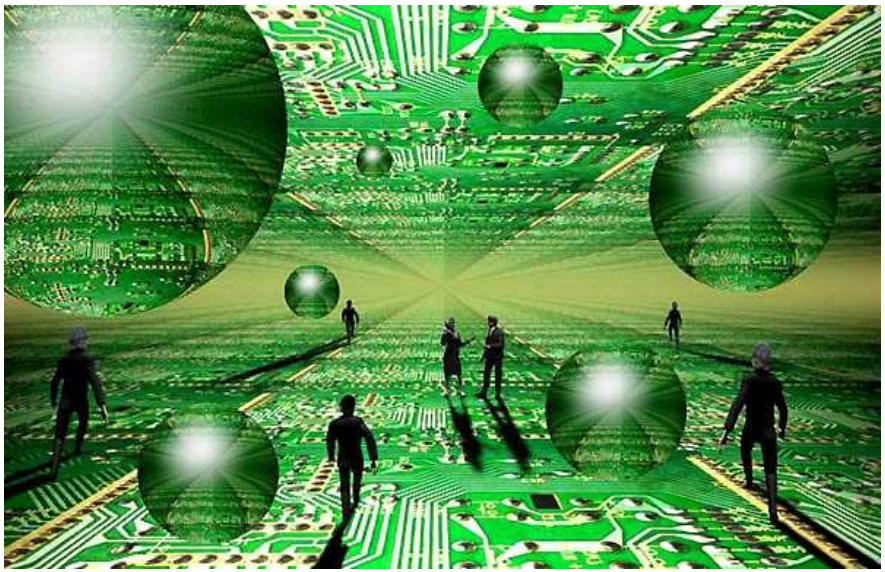
NYU
New York University

CSCI-GA.3033-016

Virtual Machines: Concepts & Applications

Lecture 9: Advanced Topics

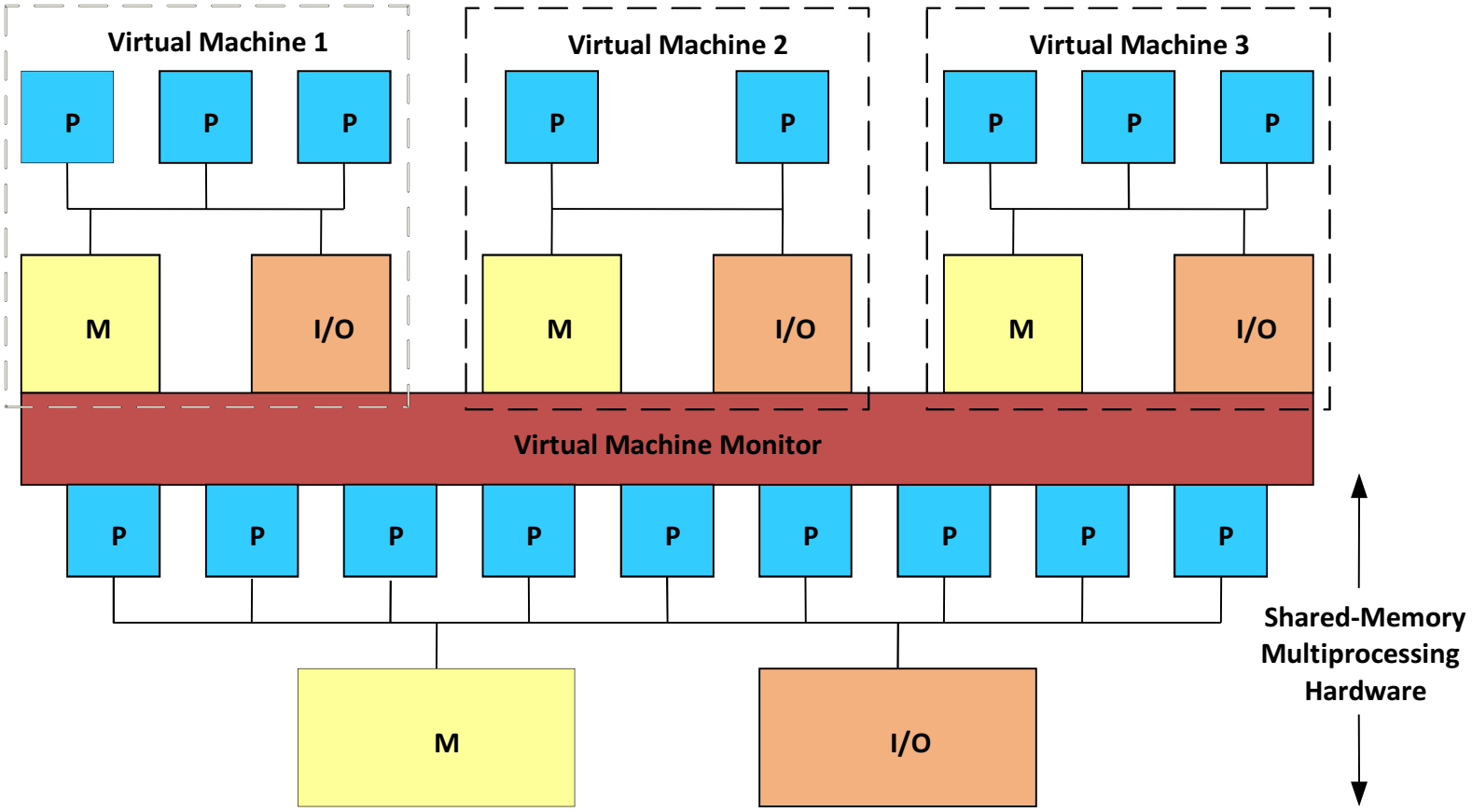
Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com



Multiprocessor Virtualization

A Common Theme: Partitioning

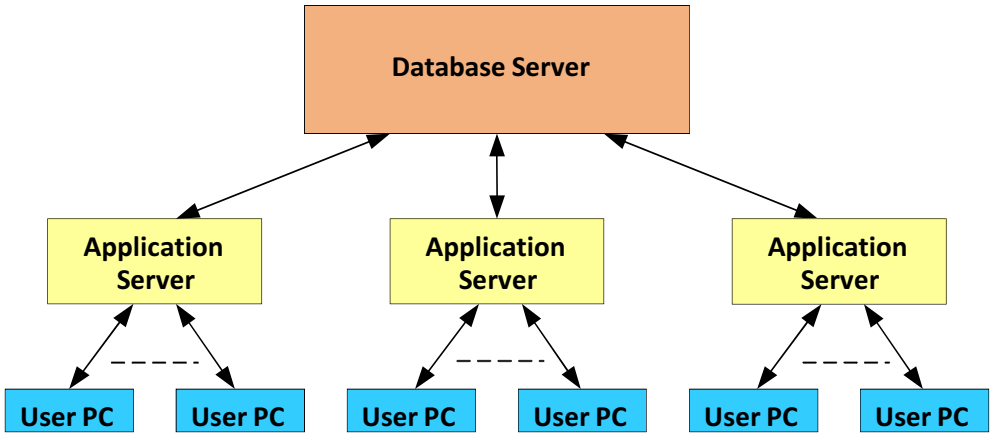
- Divide single large MP into multiple, smaller MPs



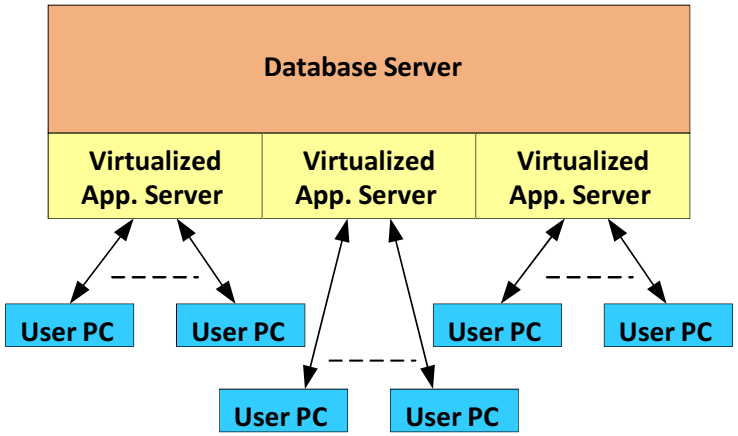
Partitioning of Multiprocessor Systems

- Partitioning in time
  - Processor in System VMs partitioned in time
  - Multiprogramming
- Partitioning in space
  - Multiprocessor system partitioned into
    - Multiple independent systems
    - Cluster of smaller multiprocessor systems
- Small SMPs on a Large SMP
  - Several virtual shared-memory systems on a single large shared-memory system

Partitioning Advantages



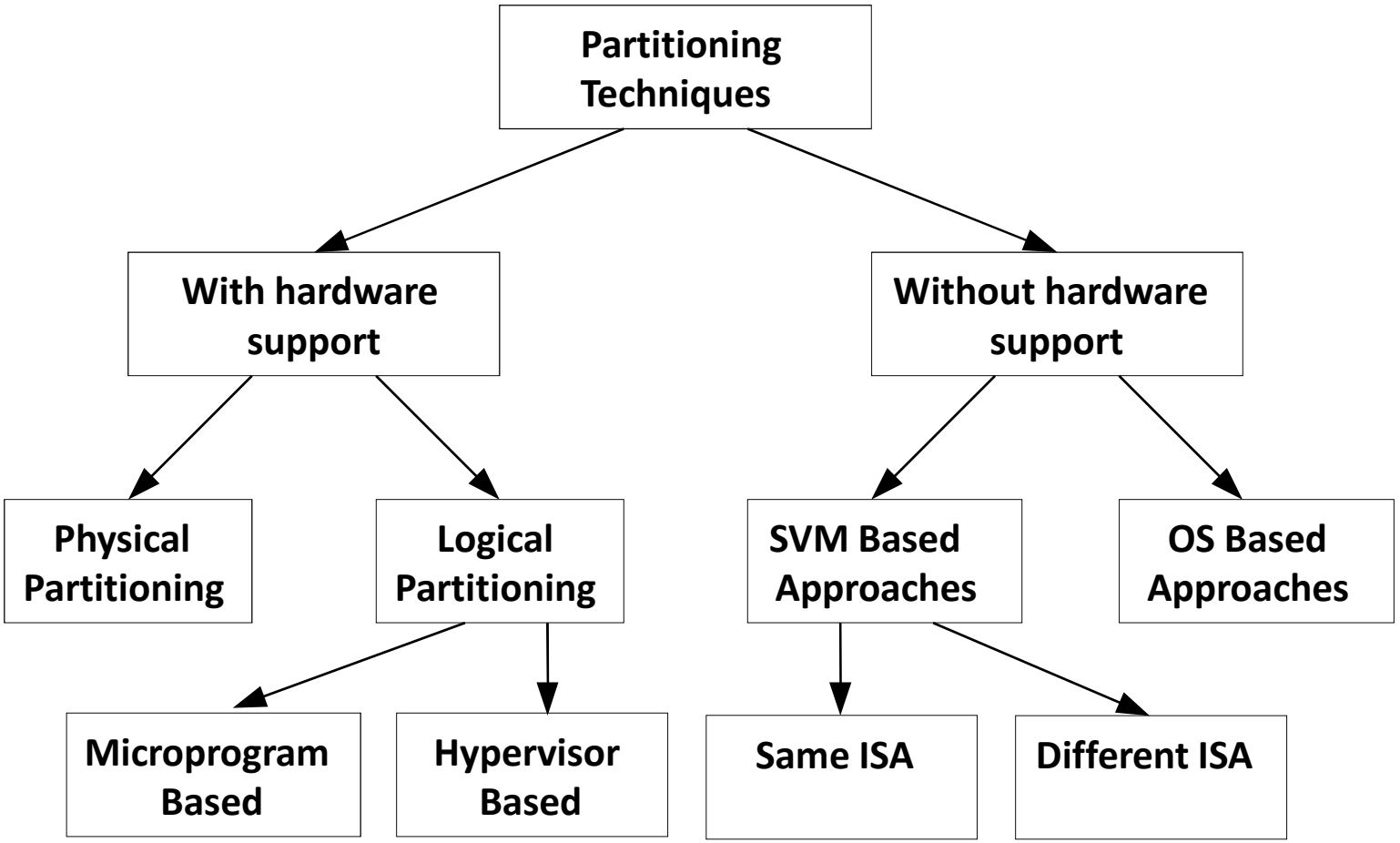
Typical 3-tier server model



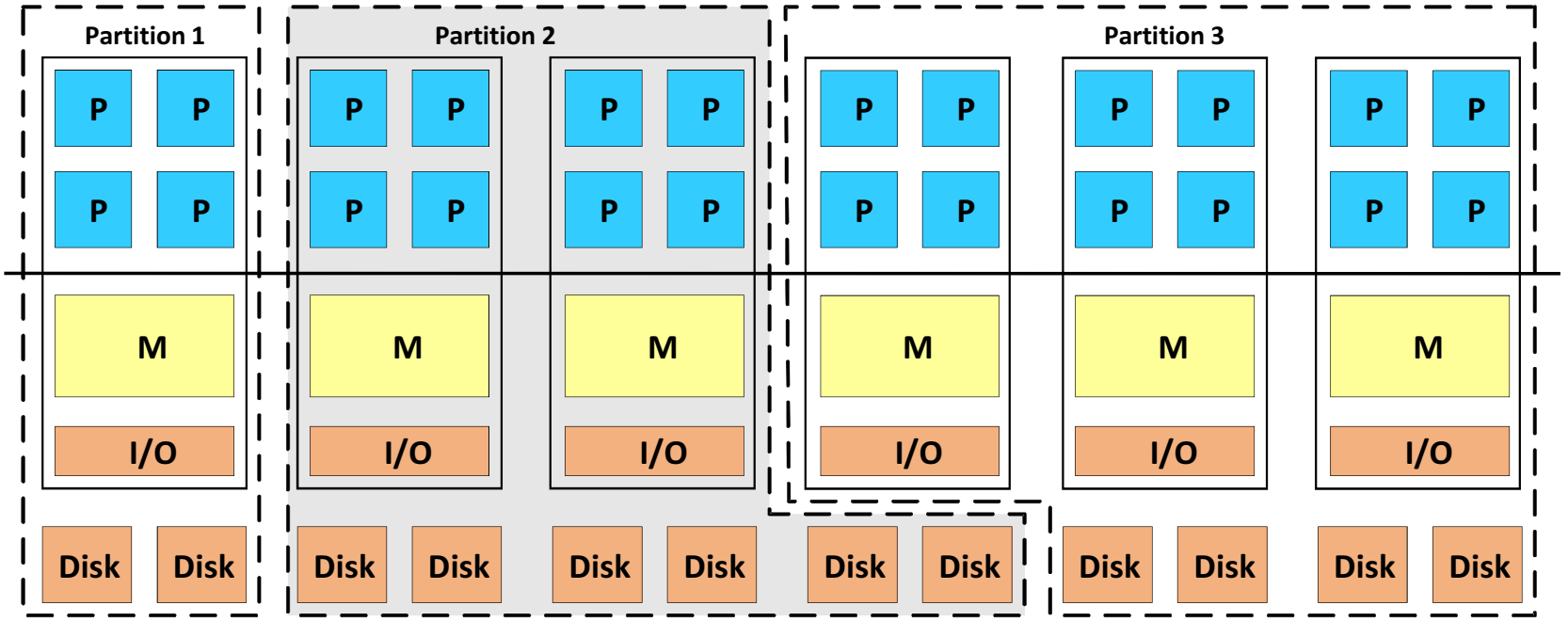
Consolidation of Application Server with Database Server

Less systems to support = less admin costs

Partitioning Techniques



Physical Partitioning



- The entire system is controlled through a main console (not shown above).
- A central control unit receives commands from console and send them to the hardware recourses.

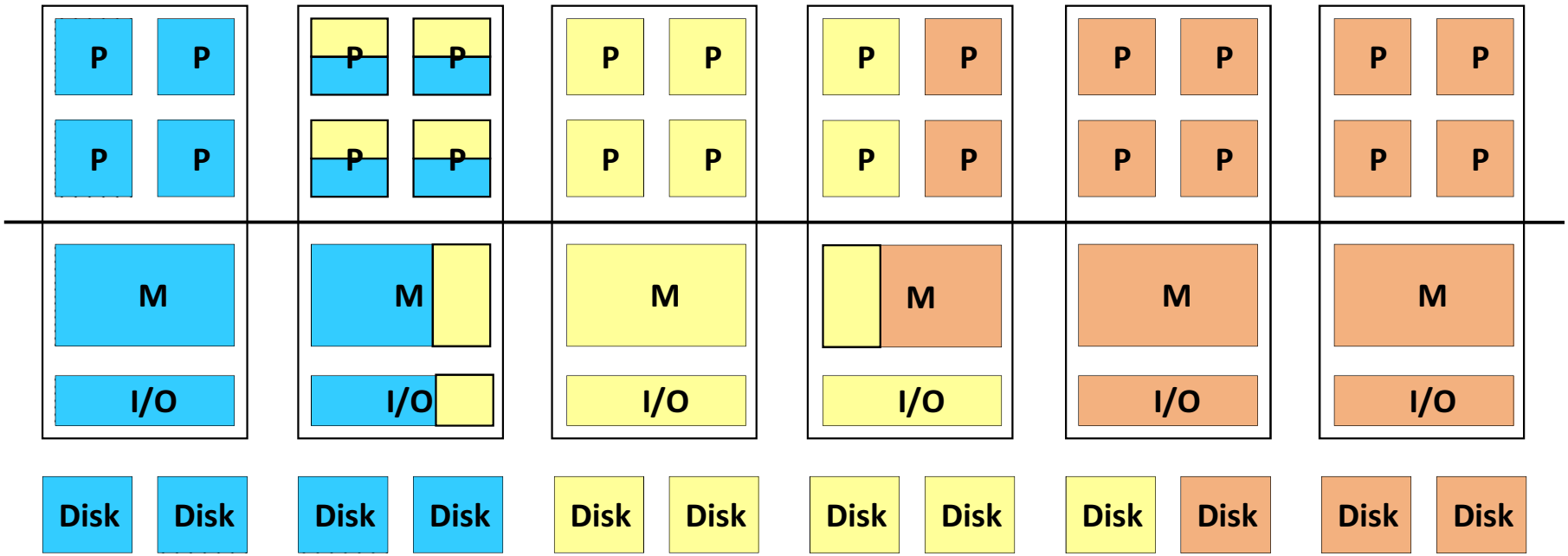
Features of Physical Partitioning

- Robustness to failures
  - Control unit reboots only the failing OS
- Good security isolation
  - One partition cannot attack another
  - Each partition has its own system administration
- Ability to meet system-level objectives
  - Behavior is similar to hardware dedicated to guest OS
- System utilization not optimal
  - Utilization determined by guest OS and applications on each partition

Logical Partitioning with Firmware

- Avoid software overhead of conventional VMM via combination of firmware (microcode) and special hardware
- Allocation of resources happens before OSes are booted
- Hardware resources are physically shared, but logically partitioned

Logical Partitioning with Firmware



Logical Partitioning with Hypervisors

- Software-only approach
  - Fewer microcoded processors today
- Add special operating mode not visible to ISA
  - *Hypervisor* software running in this mode can be hidden from conventional OS
    - Similar to co-designed VMs
    - Tends to be lightweight
  - Primary use is to provide partitioning

Hypervisors Vs System VM

Hypervisors

- Runs on highest privileged mode
- Needs hardware support
- Guest OS in privileged mode

System VMs

- Runs on highest privileged mode
- Can work on unmodified hardware
- Guest OS in user mode

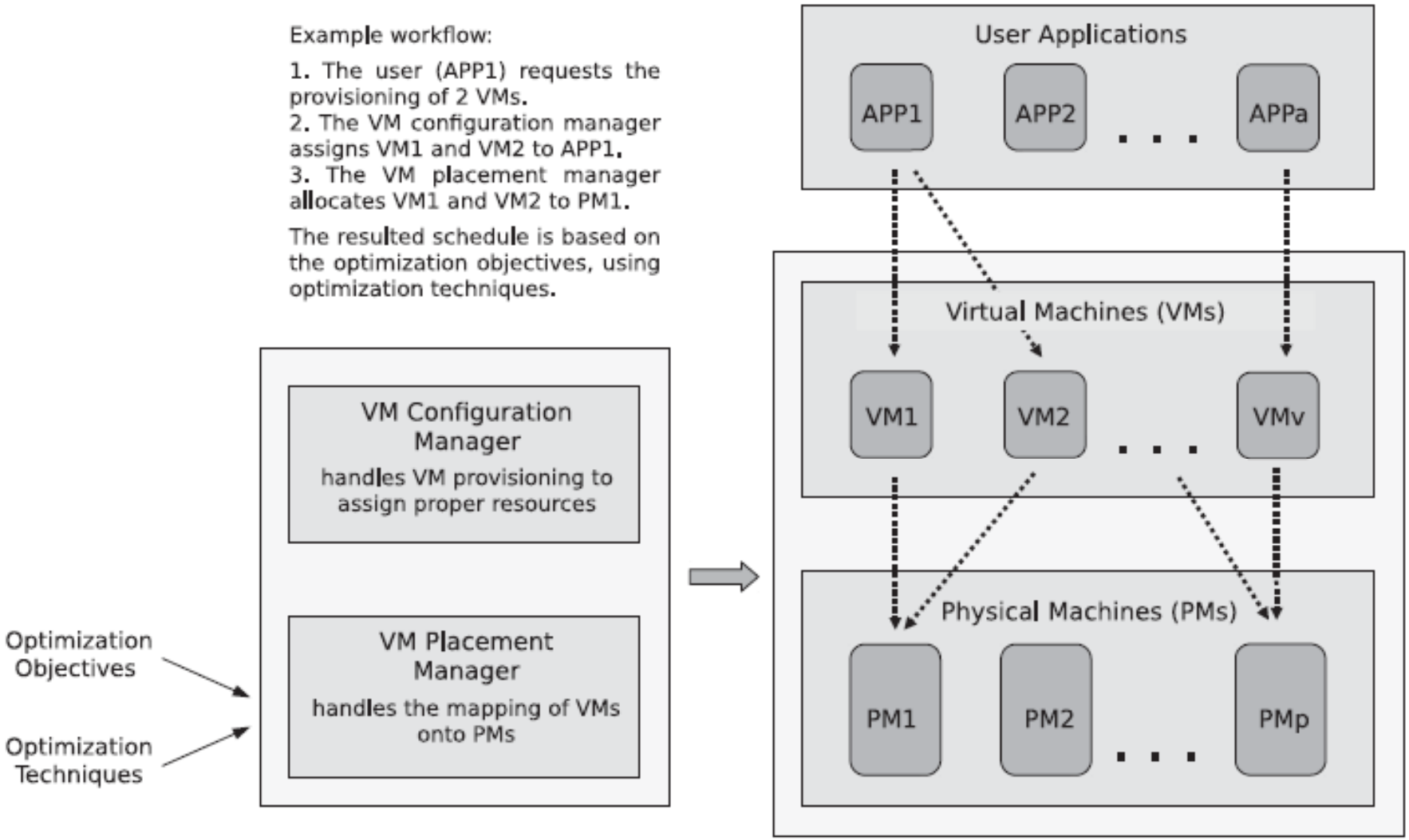
Expanding the Role of the Hypervisor

- Monitor power usage
  - Collect power information from subsystems
  - Notify OS to cut back on resource usage
- Manage fault tolerance functions
  - Manage retry
  - Isolate and reassign resources in response to failures

Dynamic Partitioning

- Allow for changes in partition configuration
  - A partition may complete its task
  - A partition may find it needs changing
  - A new partition is needed
- Hypervisor maintains pool of resources
  - Recovers resources when partition is shut down
- Dynamically change resources
  - I/O not difficult
  - Adding/deleting processors – need OS to be able to dynamically change its configuration too
  - Changing memory – difficult with contiguous address space allocation
    - Need to have smaller granularity allocation of memory

Cloud Computing
&
Resource Allocation



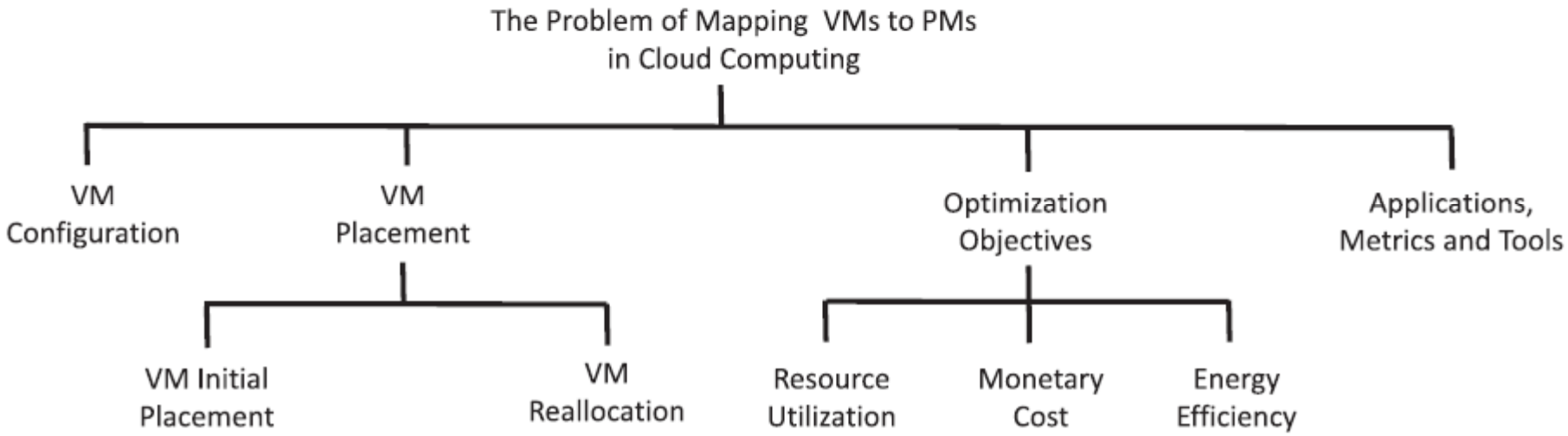
Source: Mapping Virtual Machines onto Physical Systems in Cloud Computing: A Survey
ACM Computing Surveys, vol 49, No 3, October 2016

The Question

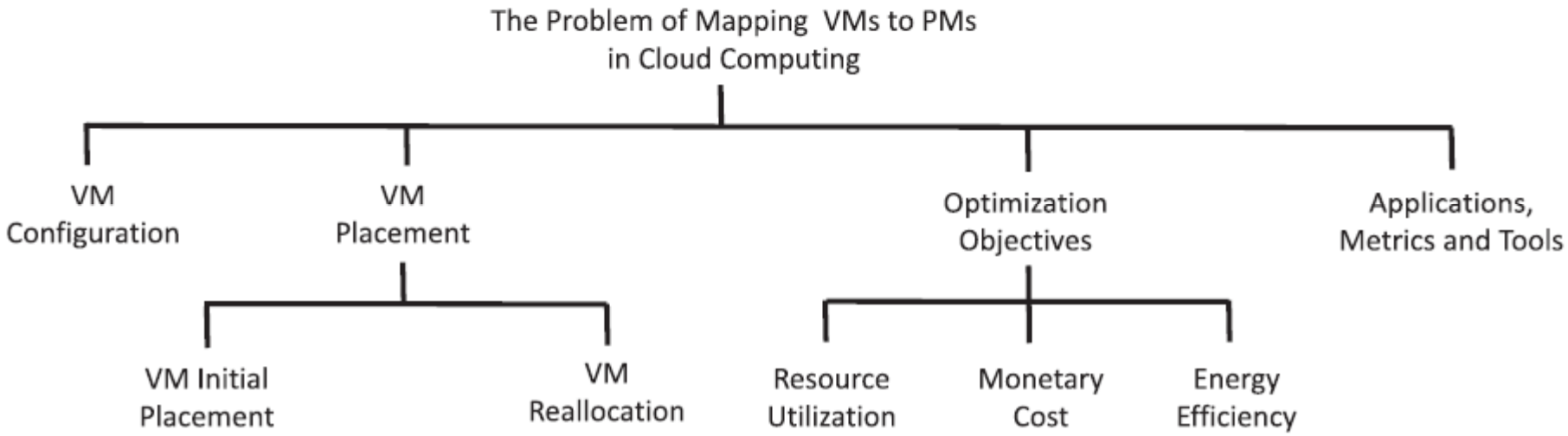
How to map virtual machines to physical resources such as:

- Provide required performance and quality of service to the VM — For the user
- Reduce operational cost — For the provider
- Increase income — For the provider

?

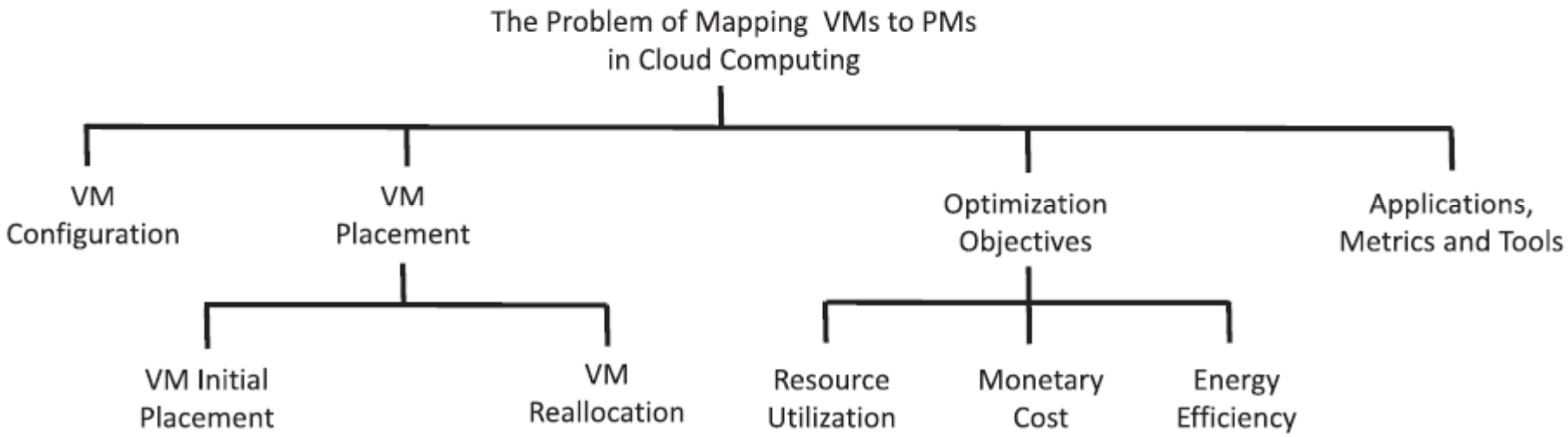


Source: Mapping Virtual Machines onto Physical Systems in Cloud Computing: A Survey ACM Computing Surveys, vol 49, No 3, October 2016



Avoid:

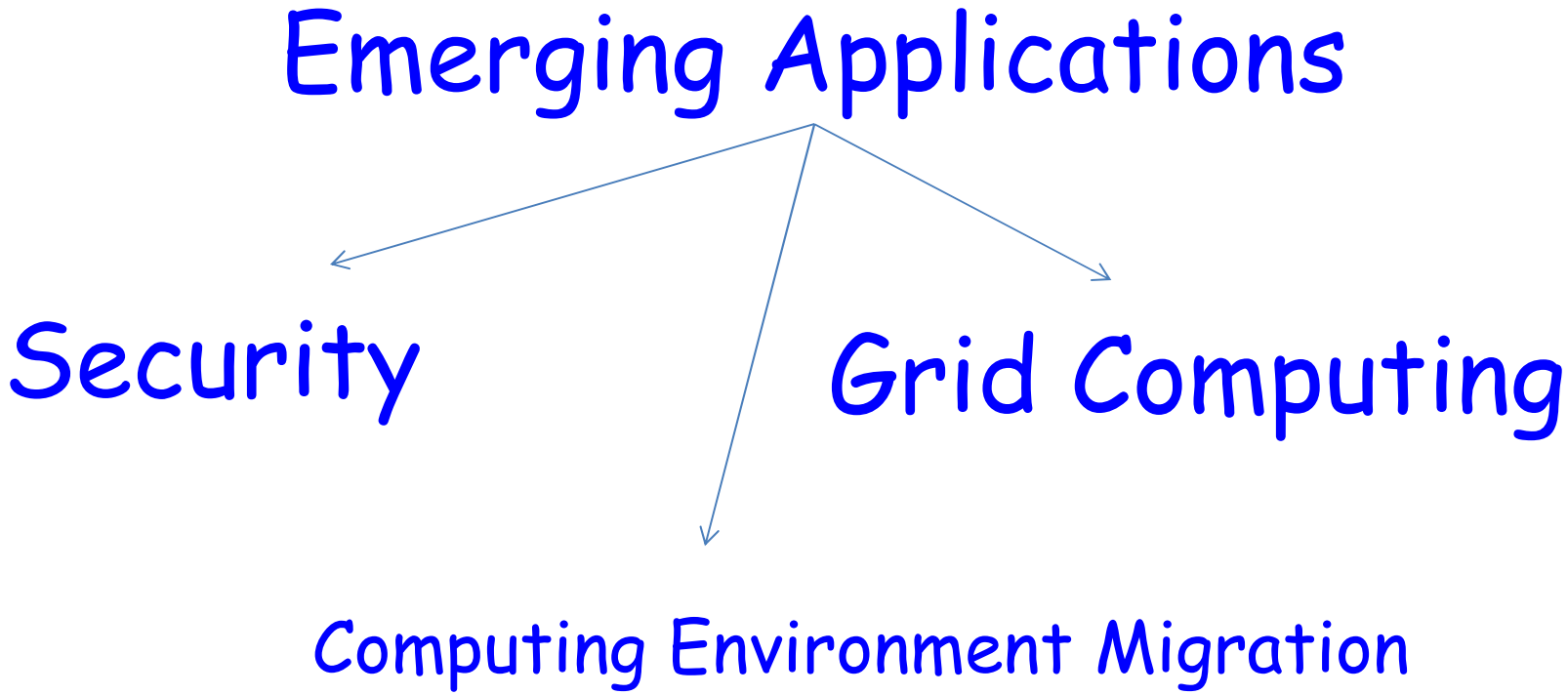
- under-provisioning
- over-provisioning

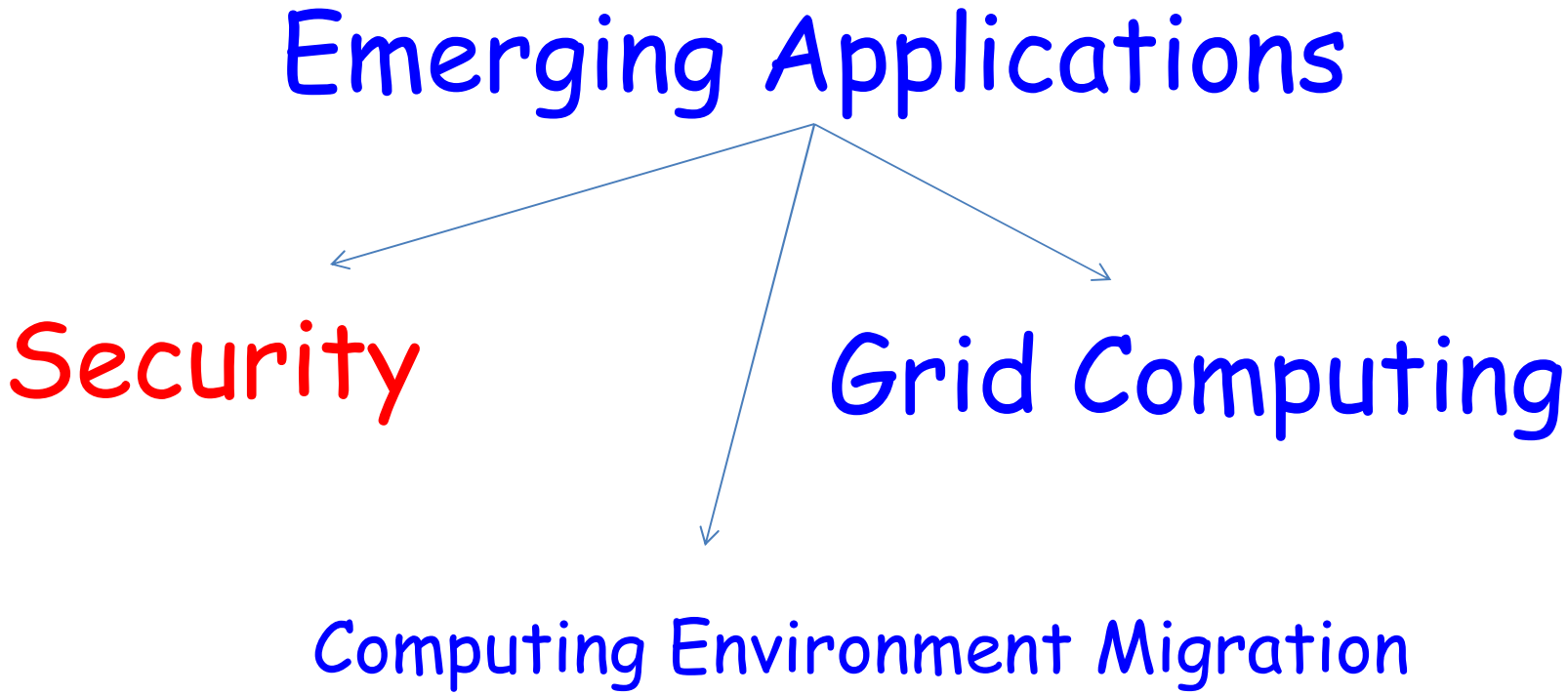


VM Migration

Triggered:

- Event-driven
- Periodic
- Hybrid
- Threshold-based (performance related)

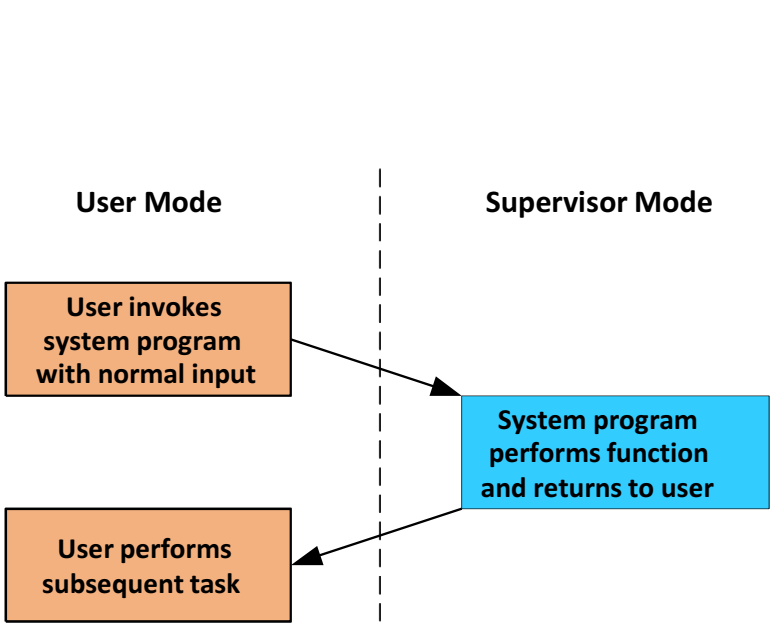

Emerging Applications
Security
Grid Computing
Computing Environment Migration


Emerging Applications
Security
Grid Computing
Computing Environment Migration

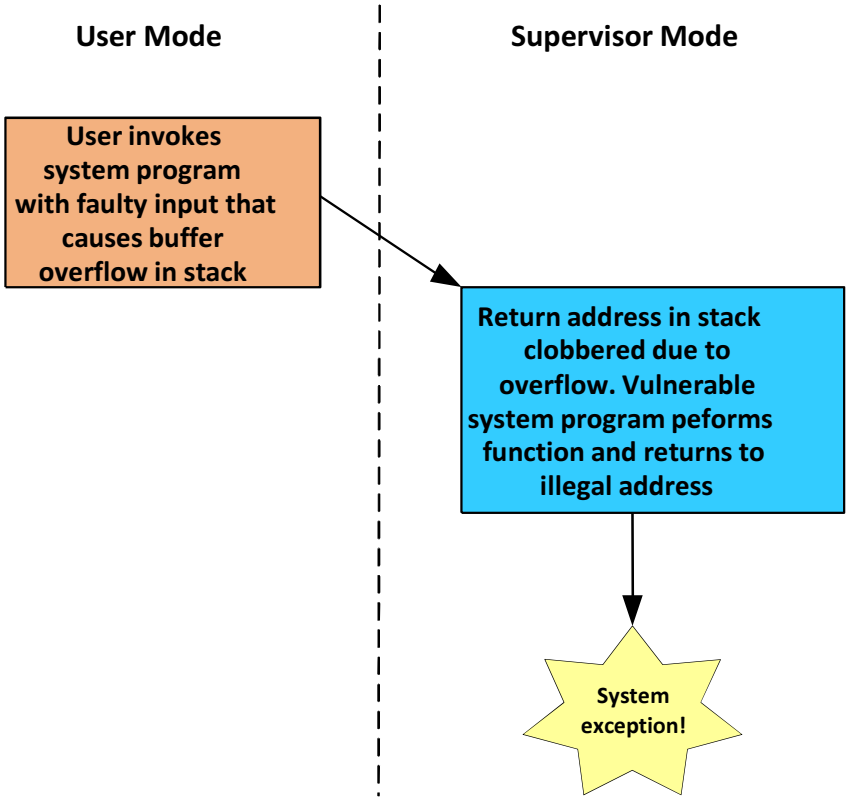
Security

- The VM "Killer App"
  - Allows controlled isolation/inspection
- Many Security Concerns
  - Worms, viruses, Trojan horses, etc.
- Typical attack – get access to privileged part of system
  - Often, very little effort
    - Compromised passwords
    - "Easy" passwords
    - Mechanically repeated efforts
  - Exploit weakness in system software
    - Unchecked accesses to system arrays
    - Can get control in privileged state by causing overflows

Buffer Overflow

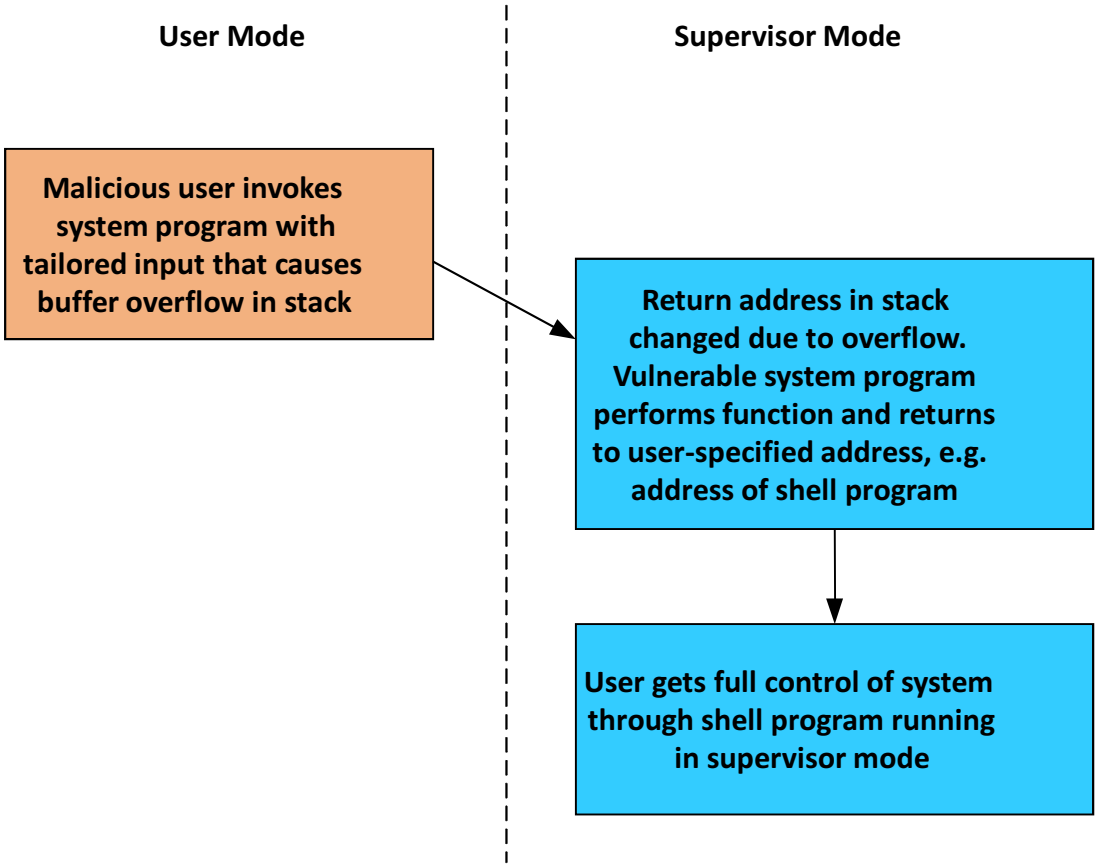


Normal Input



Faulty Input

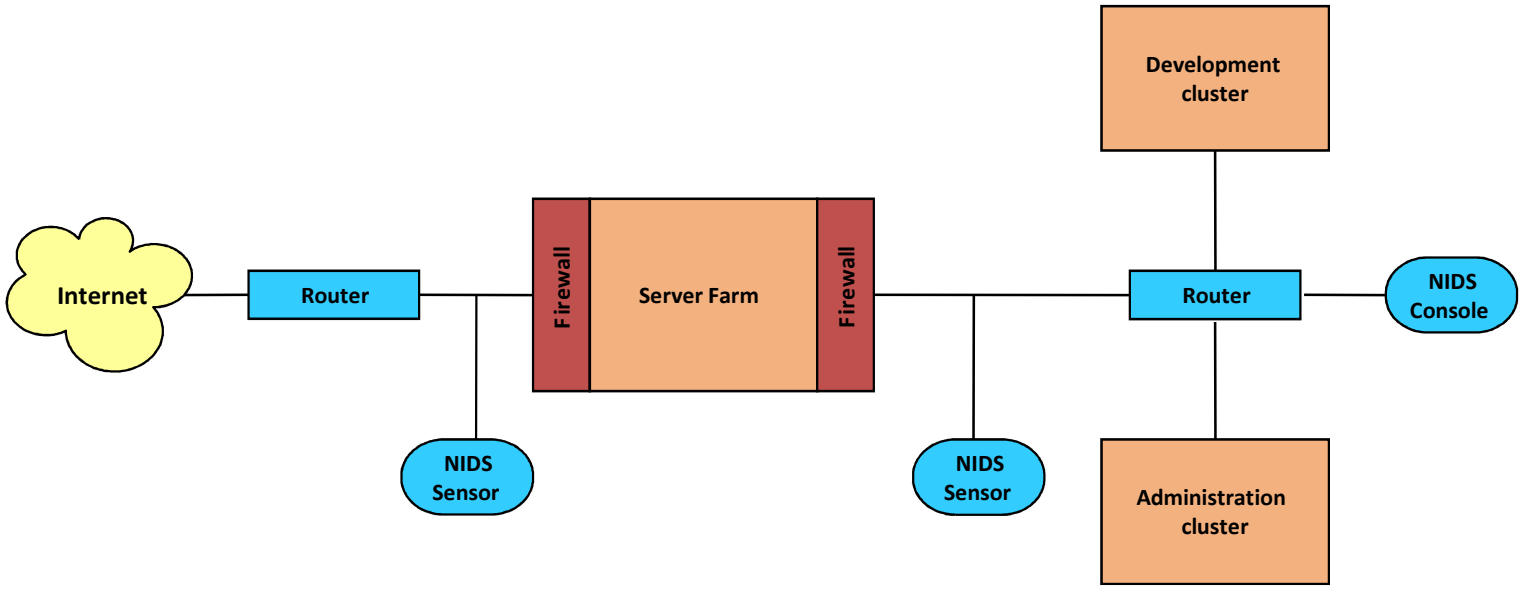
Malicious Input – Intrusion



Intrusion Detection Systems

- Complete Isolation is not an option
  - Increasing dependence on communication over networks
- Language-level checking
  - Java, MSIL – range- and type-checking
  - Restricted to processes in HLL VM
  - Legacy applications and legacy style not protected
- Need for Intrusion Detection Systems (IDS)
  - Depend on knowledge of potential attacks
  - Network-based Intrusion Detection Systems (NIDS)
  - Host-based Intrusion Detection Systems (HIDS)

Network Intrusion Detection Systems



- Decoupled from system being protected
- Monitor data packets moving through network
- Take action on detecting suspicious activity
  - Block traffic or reroute traffic to quarantine location

Host Intrusion Detection Systems

- Directly examine activity on host
  - Knowledge of host operating system
  - Look for repeated attempts
    - To crack password
    - To access unauthorized files, etc.
- HIDS has significantly better viewpoint compared to NIDS
- *But* HIDS can be disabled by attack
  - Or can provide misleading information

Monitoring and Recovering from Attacks

- Importance of understanding attacks
  - To recover from an attack
  - To prevent future attacks
- Logging
  - Save information about critical activity on system
    - Know the events that caused the failure
  - Save checkpoint of state of system
    - Reconstruct the attack from a known good state

What is the role of VM in all that?

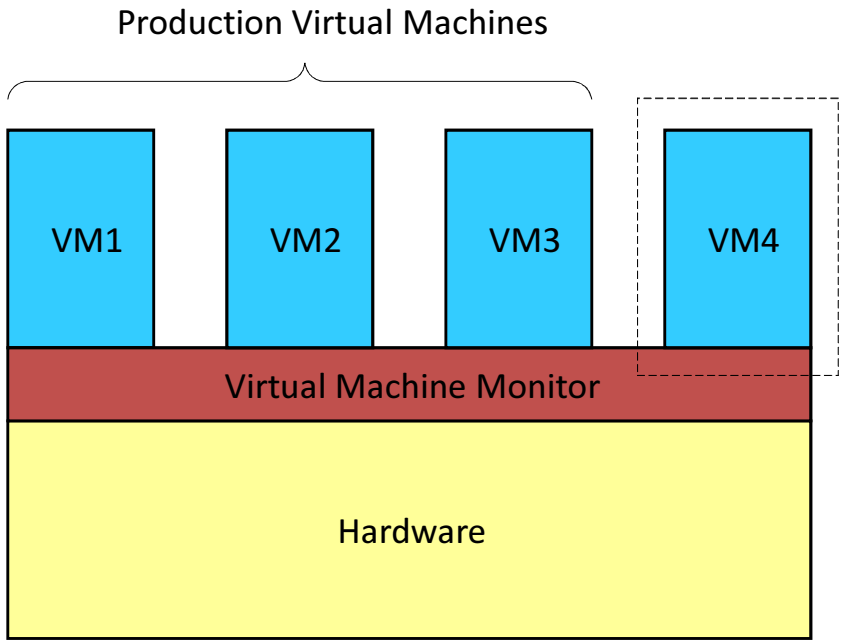
Two Conflicting Requirements

Security features are integrated closely to the rest of the system

Techniques used to enforce security and monitor activity must be separate from the system being attached.

1. Virtual Machines as a Sandbox

- Fault containment important feature of VMs
- VM Isolation helps in close examination of attack
  - Clone system that has been attacked for later analysis
- Use VM as a "honey-pot"
  - Permit attacks that can be monitored



2. Virtual Machine for Monitoring

- Separates IDS (Intrusion Detection Systems) from VMM
- IDS configures the VMM to monitor activity at more than the usual points
  - Signature of suspicious activity may be specified
- After initialization, IDS enters the picture only in analyzing data from suspicious activity
- Feedback – suggest new monitoring based on analysis
  - E.g. monitor system call activity after repeated login attempts
- May need knowledge of OS to analyze data, e.g. crash dumps

3. Secure and Complete Logging

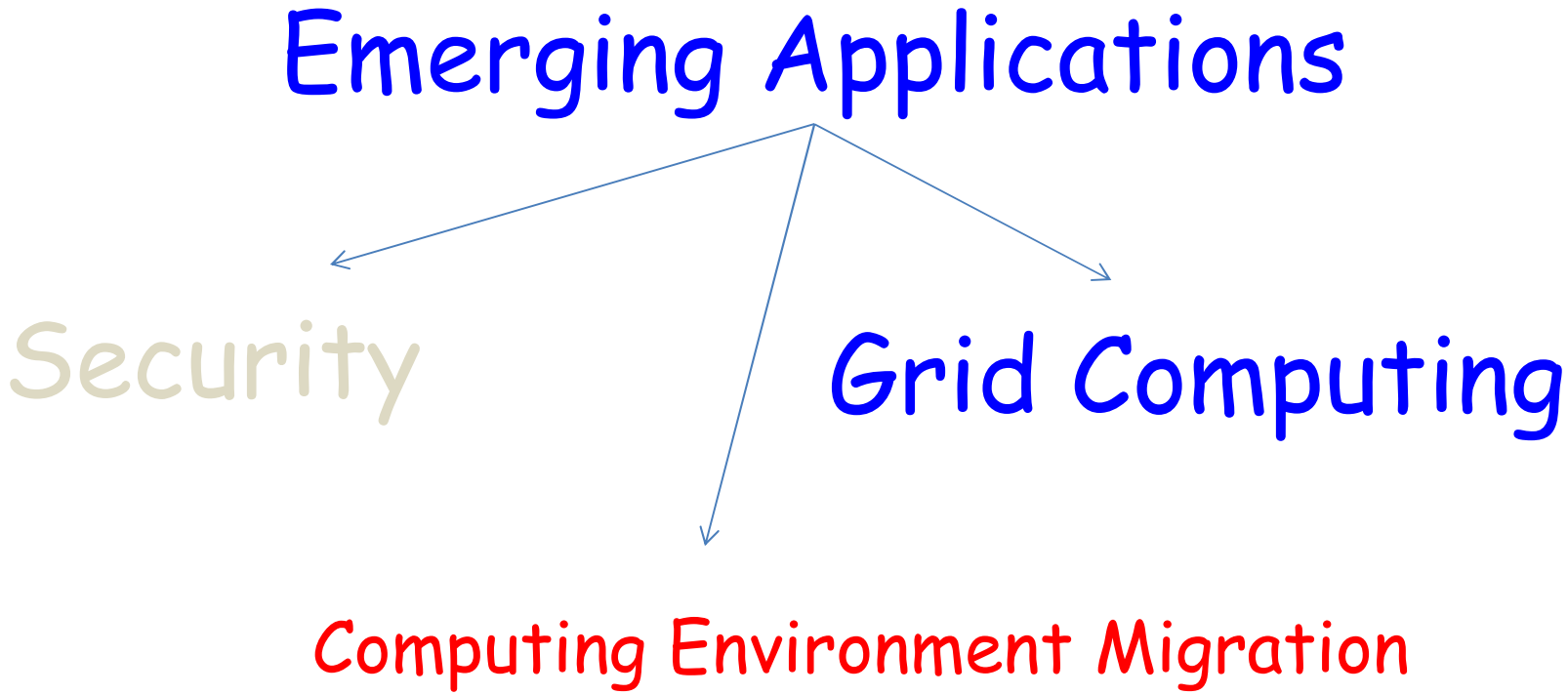
- Logging of activity
  - Enables analysis of attack
    - Techniques to anticipate attack
    - Know when attack has occurred
- Could log access to critical components
  - Login attempts, network activity, registry activity, etc.
  - Intruders could mask information or change logs
  - May only provide evidence of intrusion, not reason

Problems

- **ReVirt** (U. Michigan)
  - Uses VM to separate logging from system activity
  - Collects all information needed to replay the system

4. Dynamic Binary Rewriting

- Process VM
- Guest and Host same ISA
- Program shepherding
  - Control execution of program
    - Prevent program from being attacked
    - Prevent program from being launching point for attacks

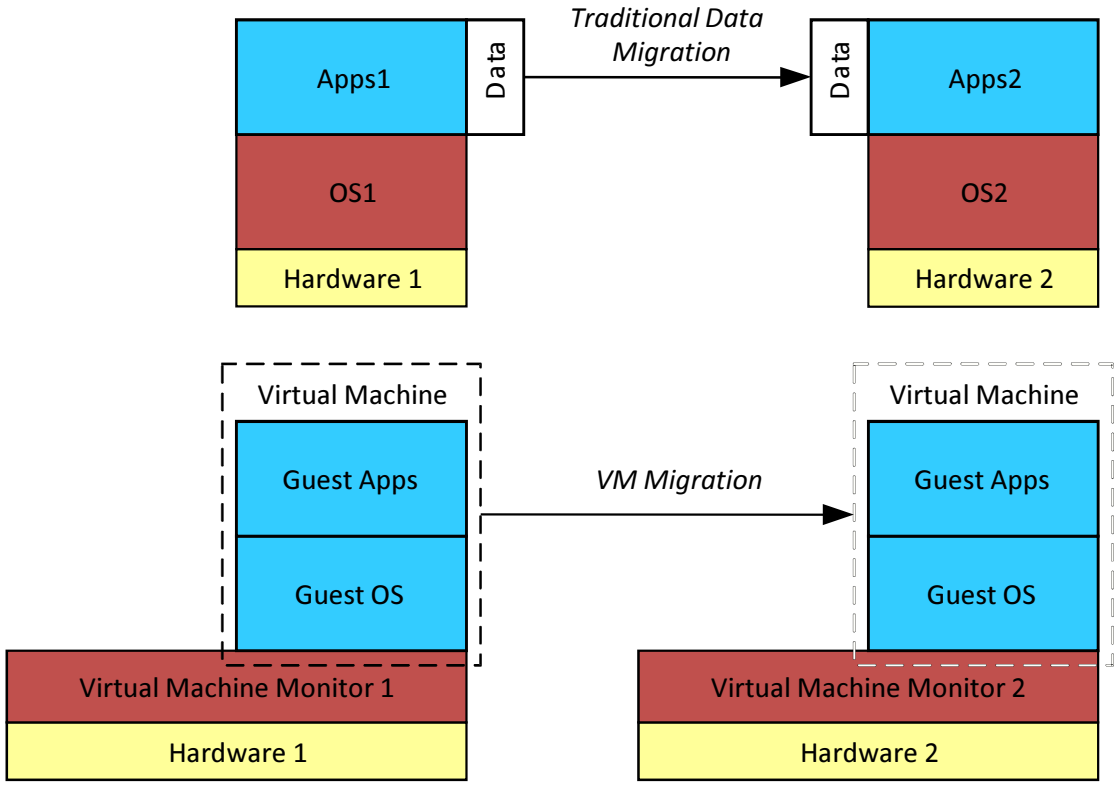

Emerging Applications
Security
Grid Computing
Computing Environment Migration

Migration of Computing Environments

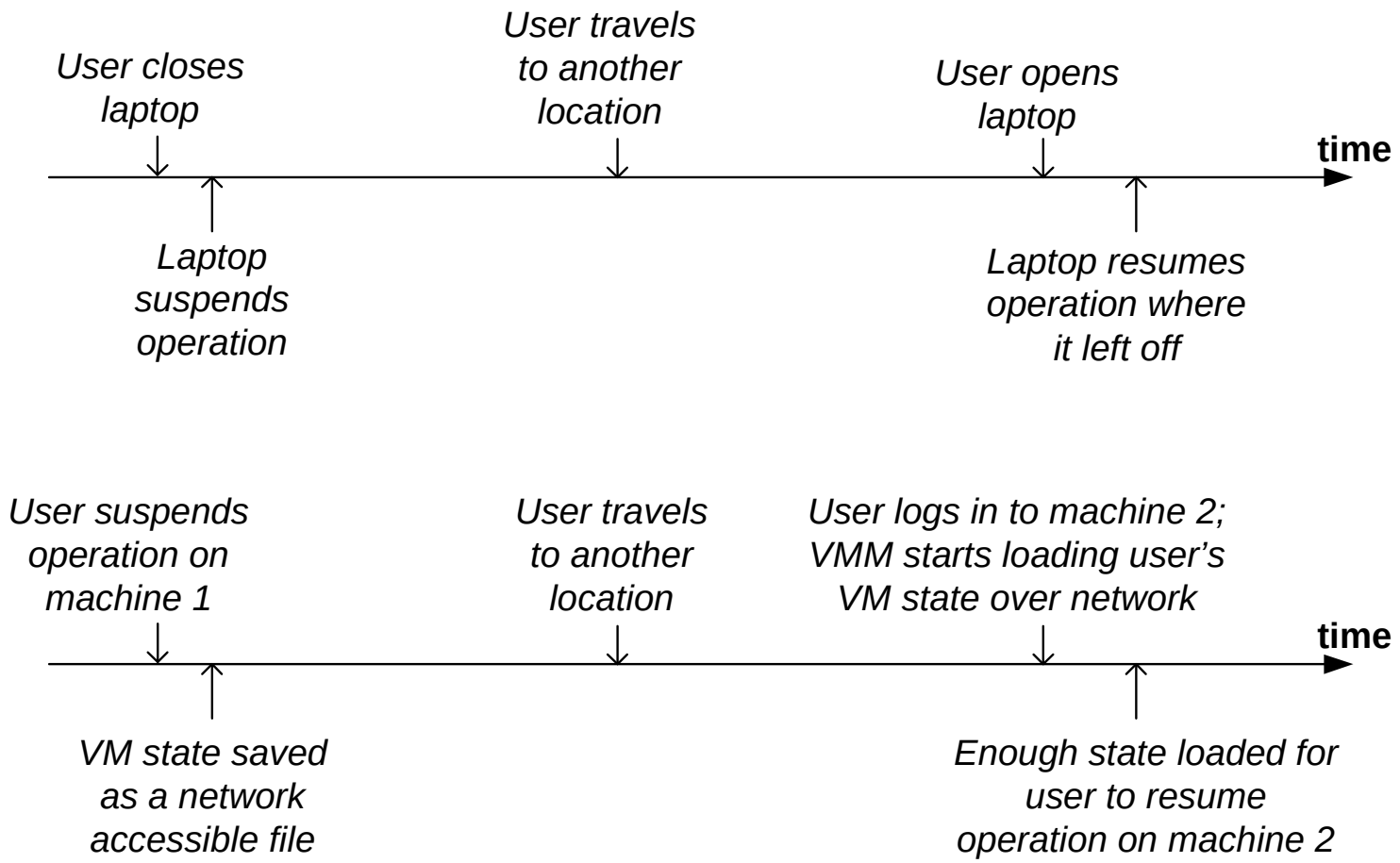
- Identical environment at any work location
  - When moving from one location to another
    - E.g. Home to work and back
  - Effect similar to carrying hardware back and forth
    - Physical security has to be taken care of
- Entire state of machine must be transported
  - State of processor resources
    - For OS as well as applications
  - Includes active code and data
- Concept of a *capsule*
  - Compressed information about entire system
    - Can be transported from one location to another

Virtual Computers

- Encapsulation simplified through use of virtual machines
- Encapsulation has the effect of checkpointing
  - Suspend operation on one platform and resume execution at exactly same point on another platform

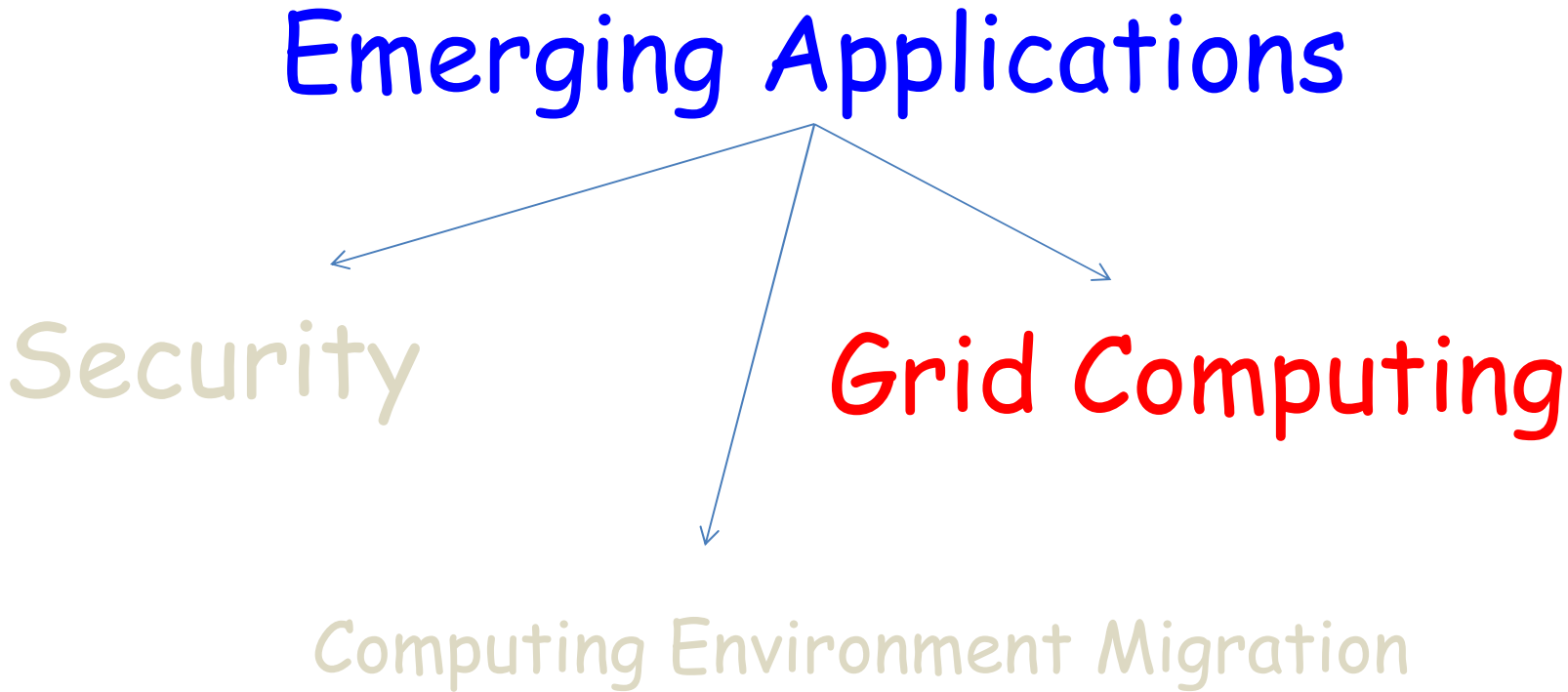


The Internet Suspend/Resume Scheme



ISR scheme (contd.)

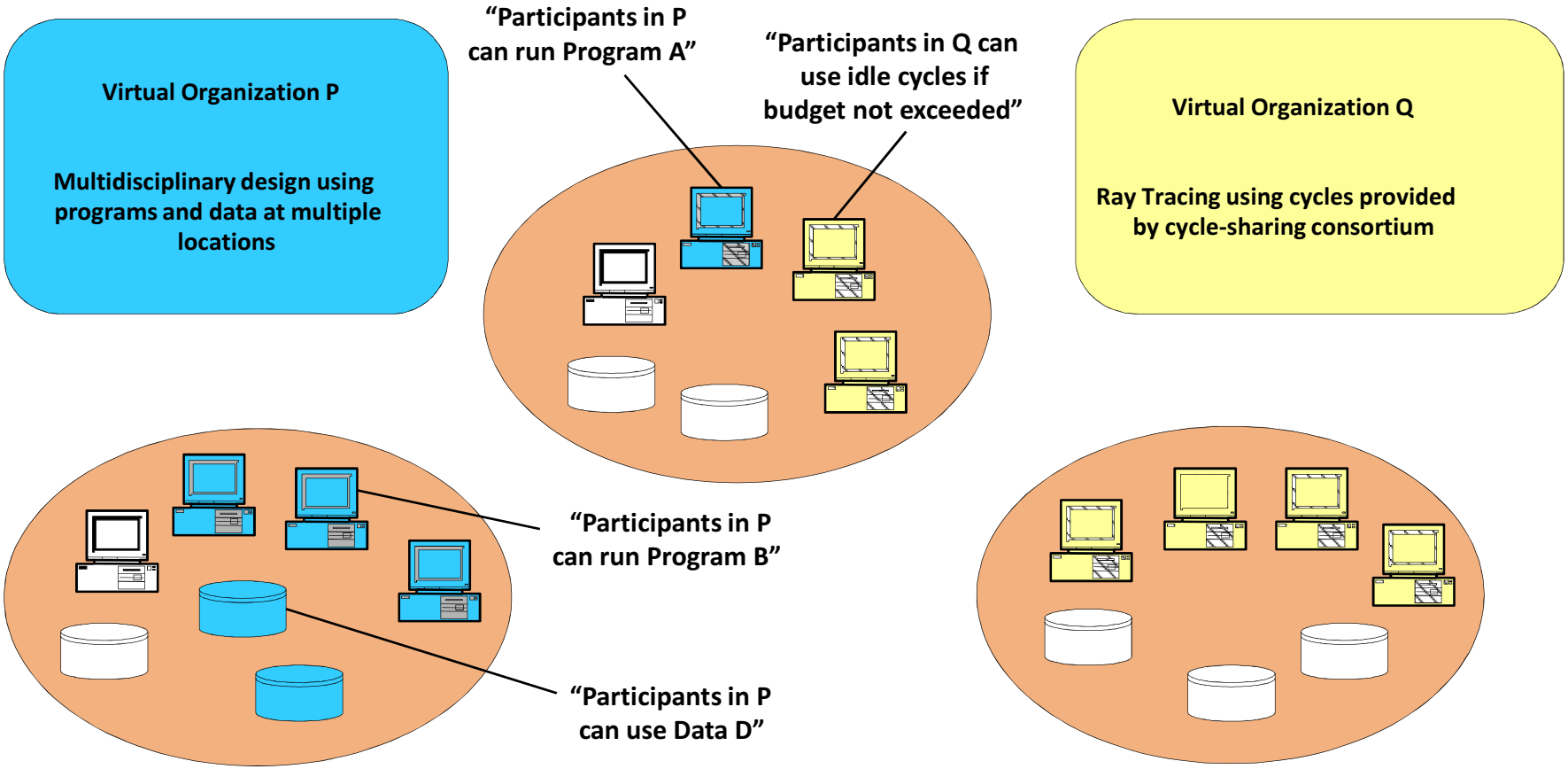
- Capsule saved in a distributed file system
  - Accessible from both source and destination locations
- "Pull" model
  - Load capsule on demand
- Load incrementally
  - Essential parts of capsule loaded initially
  - Rest loaded on demand, or in background
  - Could learn usage pattern
- Parts of system may be already available at destination
  - Reuse from other users at destination
- Secure transmission
  - Encrypt capsule information


Emerging Applications
Security
Grid Computing
Computing Environment Migration

Grids

- Existence of large amounts of computing resources
  - Most desktops are used lightly
- Energy consumption is also becoming a problem
- Computing as a utility
  - Similar to power grid – pool resources together for use by a much larger number of users
  - Allow customers to use as much computation as desired and pay according to use
- High-end scientific computation
  - Spearheading collaborative techniques
  - Grid as a virtual organization serving a community

Grids: Virtual Organizations



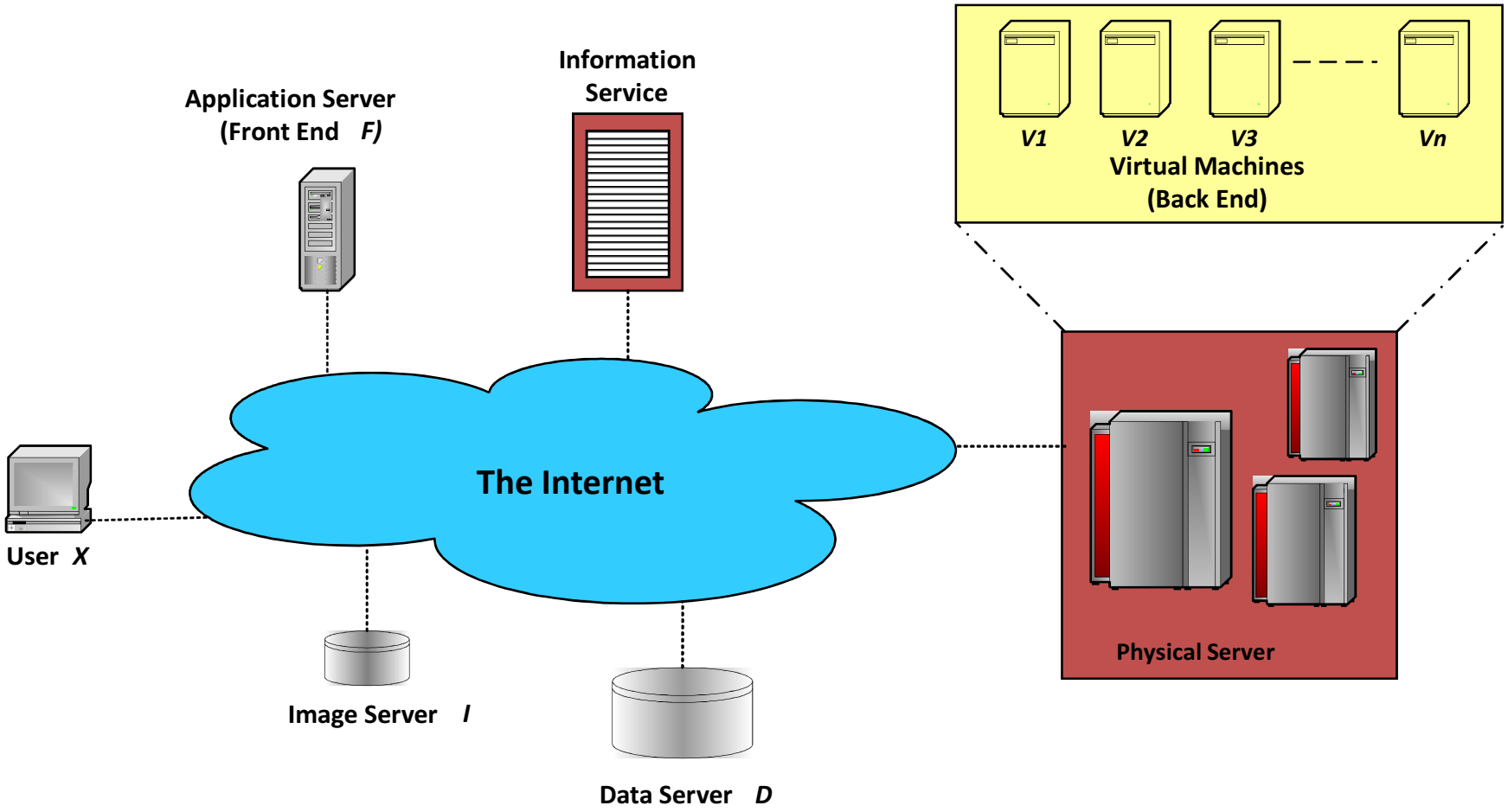
Comparison with Conventional VMs

- Efficient utilization of resources
  - Similar in motivation to original system VMs
- Sharing of resources
  - Grid concerned with sharing of content also
    - Not just sharing of resources
- Distributed control
  - Grid has global scope
    - Users negotiate with each other to share and use resources
- Heterogeneous nodes
  - Nodes in a grid may be different types of machines
- Adaptation of applications
  - Applications may need to be adapted for the grid
- Portability of applications
  - Conceptually similar to goals of HLL VMs

Role of System VMs in a Grid

- Grid has to manage and schedule resources
  - Like an operating system
- However, grid has to deal with heterogeneity
  - Accounting, for example, is dependent on accounting policies of each grid participant
- System VM-based approach
  - Treat a VM as the unit of transactions on a grid
    - Not tasks, or programs

System-VM Based Grid



Advantages of SVM based Approach

- User isolation
  - Protect user from host and other users
  - Protect host from users
- Platform independence
  - User specifies type of machine, not actual machine
- Task management and accounting
  - Simplifies allocation and accounting
    - Allocate based on compute requirements
    - Charge based on performance of VM
- Portability
  - Allows applications to be written for execution on the widest range of platforms
  - Eases encapsulation and migration of jobs between nodes on grid; e.g. Java VMs can be migrated

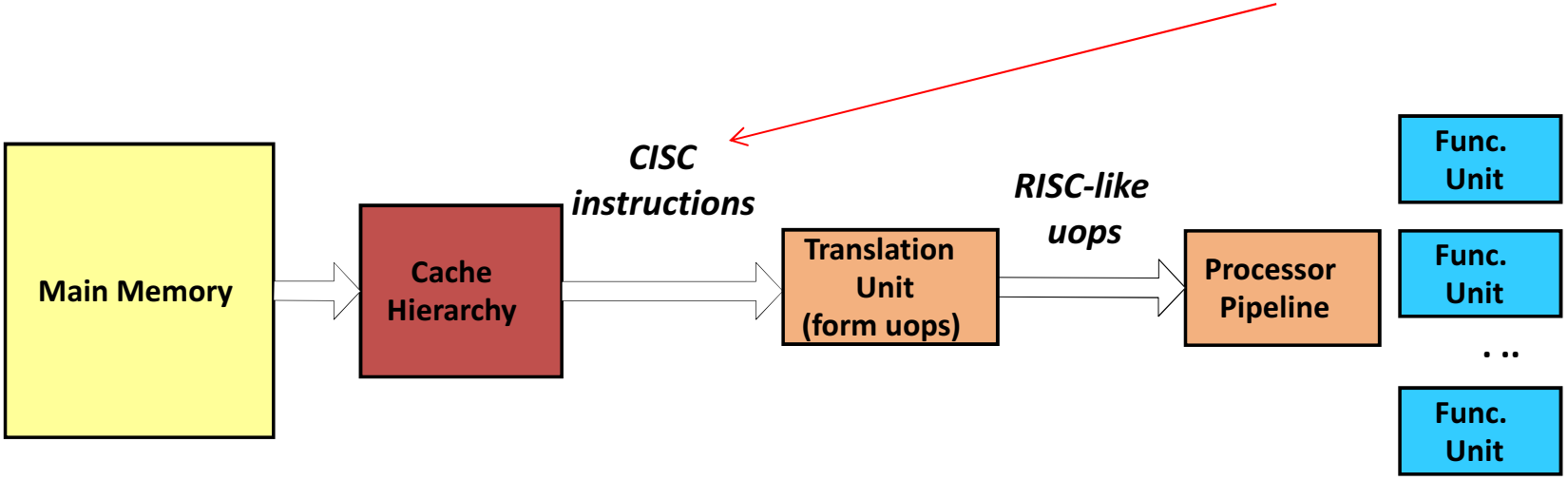
Co-Designed VMs

The ISA

- The main interface between hardware and software
- Hardware and software evolved over the years.
- ISA cannot evolve as such because of backward compatibility
- Consequently, ISAs currently in use reflect a perspective and a division of labor between hardware and software that is decades old.

CISC-to-RISC

underlying processor hardware has grown to be quite different from the image presented by the commonly used ISAs



Conventional superscalar implementation

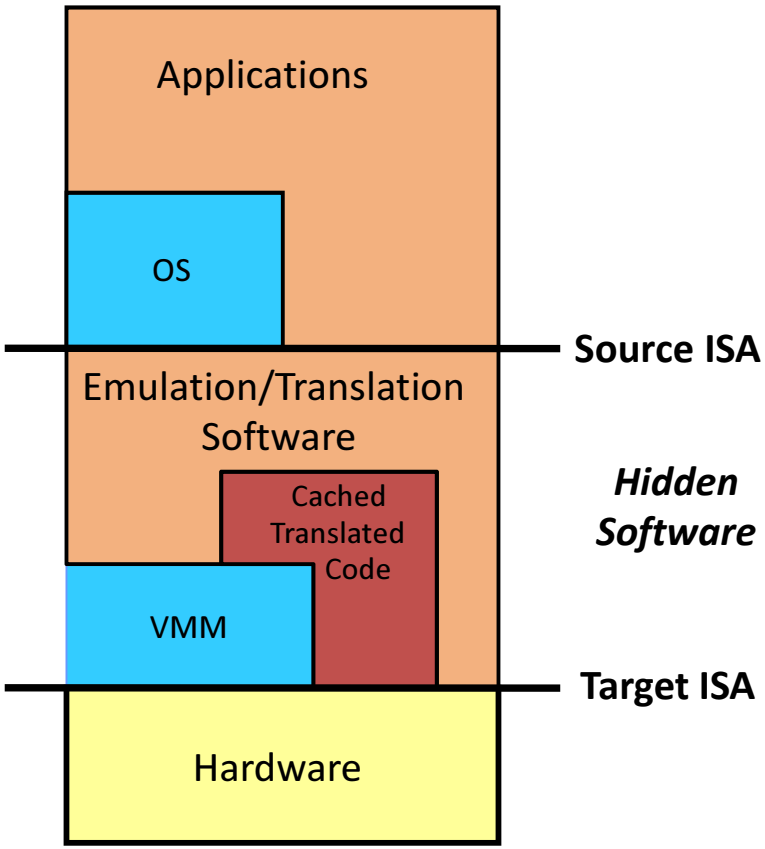
How can VM Help?

- VM technologies permit new ISAs by enabling a different approach to general-purpose processor design.
- Host architecture (the target) is designed concurrently with the VM software that runs on it.
- The interface between hardware and conventional software is shifted upward.
- A special kind of system VM?

System VM but …

- Are not intended to virtualize hardware resources other than the processor.
- Are not intended to support multiple VM environments.
- Goals include:
  - performance
  - power efficiency
  - design simplicity.

Co-Designed VMs



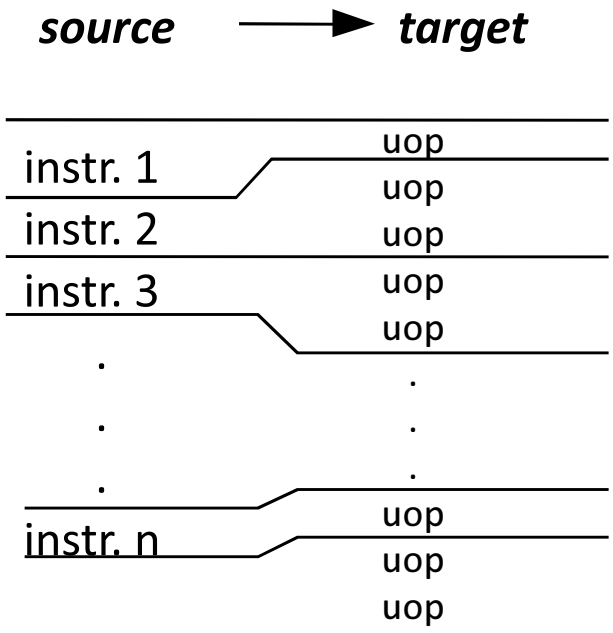
Similar to process VM in:

- emulation done in stages
- code cache for dynamic translation

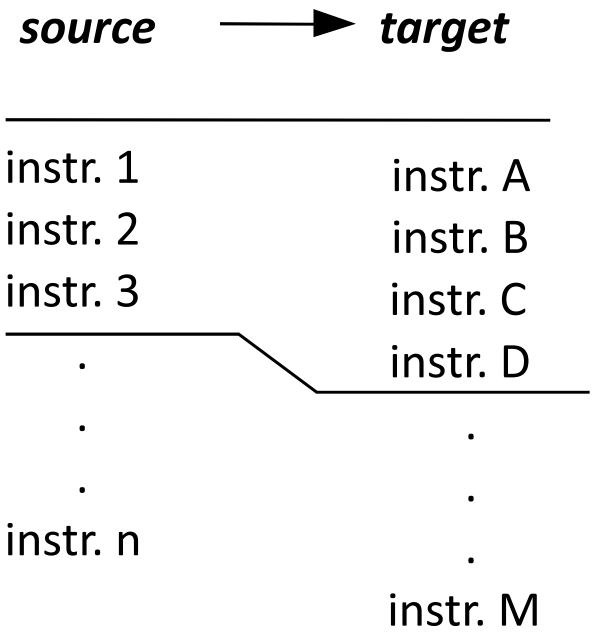
Different from process VM in:

- Intrinsic compatibility at ISA level
- The main goals are performance, power efficiency, design simplicity, or combination of that.

Hardware Vs Software Translation



context-free translation to uops



context-sensitive translation to target ISA.

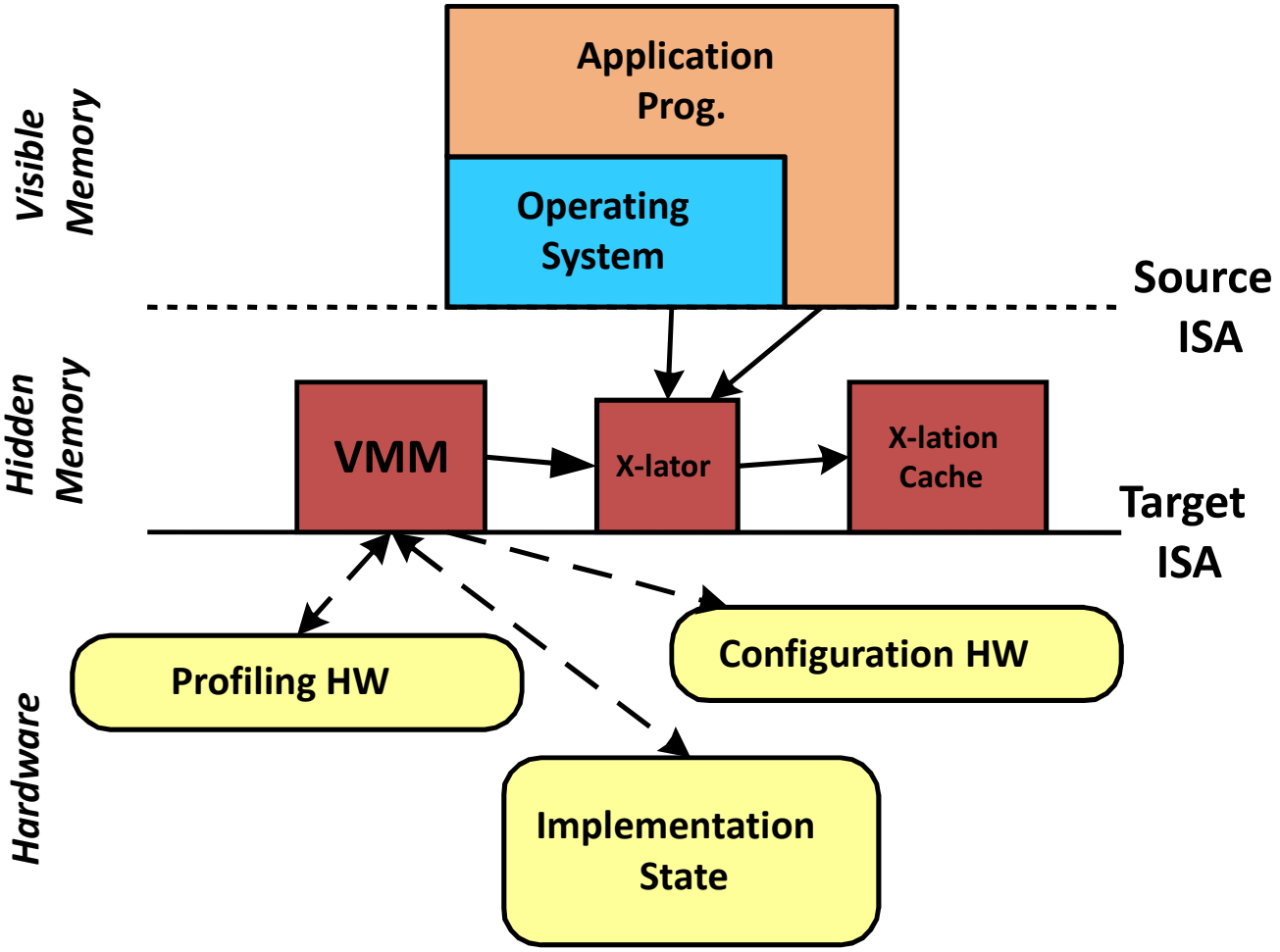
Hardware Vs Software Translation

- Hardware solution leads to higher cost of design verification
- Higher power consumption due to the added complexity
- Not flexible

What Do We Gain from Co-design VM?

- Can implement new high performance ISA
- Enhancing existing ISA
  - Add functional instructions (and still benefit existing binaries)
  - Delete instructions (and still run existing binaries)
- Managing adaptive hardware
  - Add special "sensor/actuator" instructions
    - Read dynamic performance data from profile registers
    - Write configuration control registers

Co-Designed VMs: Big Picture



Register Mapping

- Situation here is easy
- target ISA is designed specifically for the source ISA
- host register file(s):
  - can be made large enough to accommodate the guest's requirements
  - extra scratch registers left over to enhance performance and/or simplify the translation process

Code Caching … Indirect Jumps

- Doesn't software prediction solve the problem?

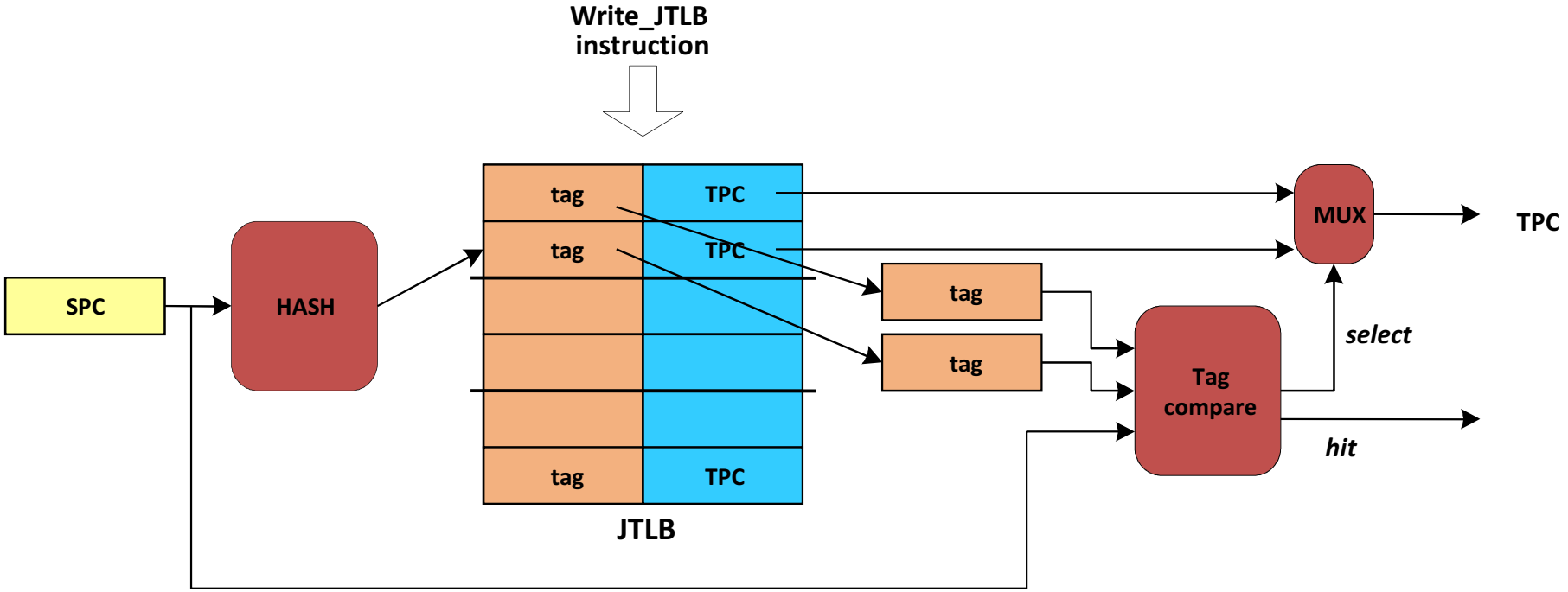
```
if ((Rx) == #addr_1) goto #target_1
else if ((Rx) == #addr_2) goto #target_2
else map_lookup (Rx)          ; do it the slow way
```

  - If it misses then more overhead is added beside map table access.
  - Some jumps are very hard to predict (e.g. return from procedures).
- The indirect-jump problem is probably the greatest source of performance loss in a software-only code cache system.

How do codesigned VMs solve this?

Support for Code Caching

- Add Jump TLB
  - A hardware cache of dispatch table entries
  - Similar to software-managed TLB in virtual memory
  - Entries are written into the JTLB by the VMM



Example of a two-way set associative JTLB

Incorporating JTLB in codesigned ISA

- Method 1:
  - JTLB_Lookup instruction that accesses the JTLB
  - SPC address held in a register
  - reads out a TPC and places it in a second register.
  - A third register (or condition code) indicates a hit or miss in the JTLB.

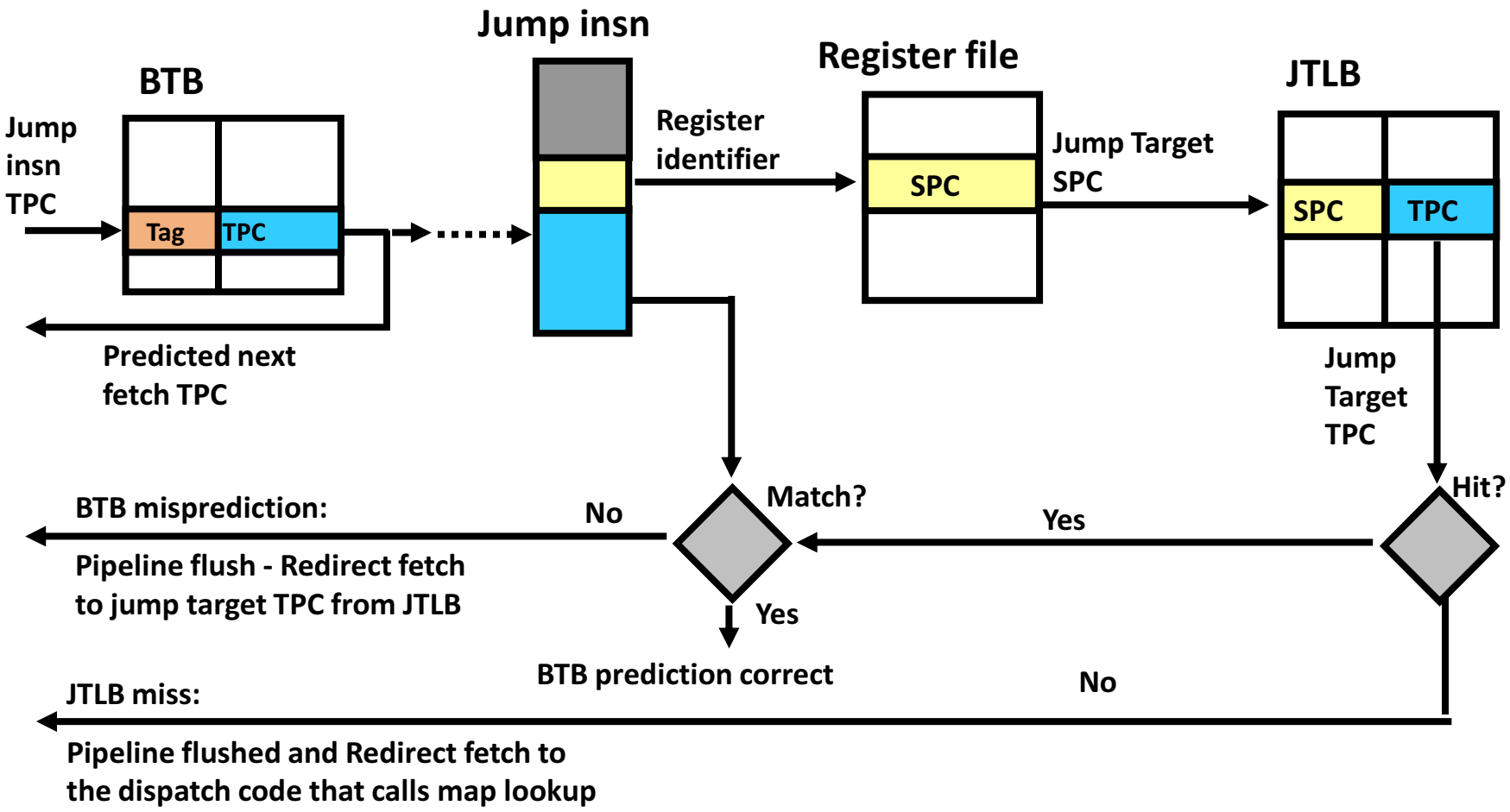
```
JTLB_Lookup Ri, Rj, Rk      ; TPC to Ri, hit/miss to Rj
Jump Ri, Rj==0              ; conditional indirect jump
Jump map_lookup             ; do it the slow way
```

Incorporating JTLB in codesigned ISA

- Method 2:
  - combine the lookup with the conditional jump
  - Lookup_Jump Rk performs the jump to the TPC if there is a hit, otherwise it falls through.
- when a jump is encountered, instruction fetching must stall until the JTLB is accessed and the jump is executed → bottleneck!

How about predicting the TPC value immediately after a Lookup_Jump instruction is fetched?

Jump TLB


Jump insn
BTB
Register file
JTLB
Jump
insn
TPC
Tag
TPC
Register
identifier
SPC
Jump Target
SPC
SPC
TPC
Predicted next
fetch TPC
Jump
Target
TPC
Hit?
Match?
BTB misprediction:
No
Yes
Pipeline flush - Redirect fetch
to jump target TPC from JTLB
Yes
BTB prediction correct
No
JTLB miss:
Pipeline flushed and Redirect fetch to
the dispatch code that calls map lookup

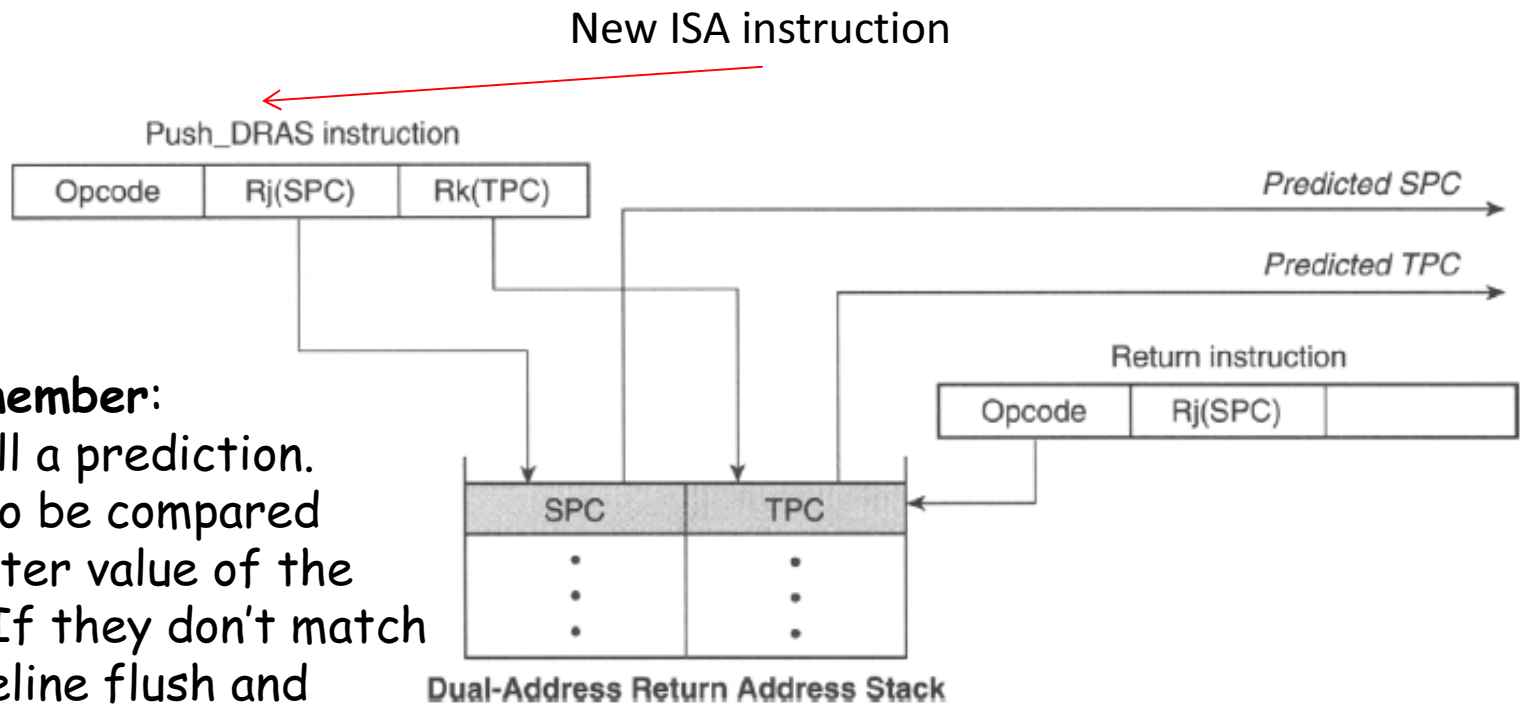
Does It Work?

- As long as BTB predictions are correct a high percentage of the time.
- It is usually so, except for procedure return jumps.
- Most modern processors employ a hardware return address stack (RAS) mechanism that can predict a return instruction's target address very accurately → It basically mimics the software procedure stack by pushing the fall-through PC onto a hardware prediction stack.
- Can we apply this in codesigned VM? Not right away, because:
  - the saved return destination address would be an SPC and we need TPC
  - If the procedure jump is at the end of a translated superblock then the address of the instruction following the jump is not the correct return address (the correct address is at the beginning of a different superblock).

What can we do?

Dual-address RAS

- Solution: save both SPC and TPC
- Sometimes we may need to put invalid TPC if it is not known at the time
- Later when the return target superblock is constructed, the invalid address is replaced with a valid TPC.



Remember:
This is still a prediction. It needs to be compared to the register value of the indirect jump. If they don't match then a pipeline flush and redirection to the map table.

Precise Exceptions

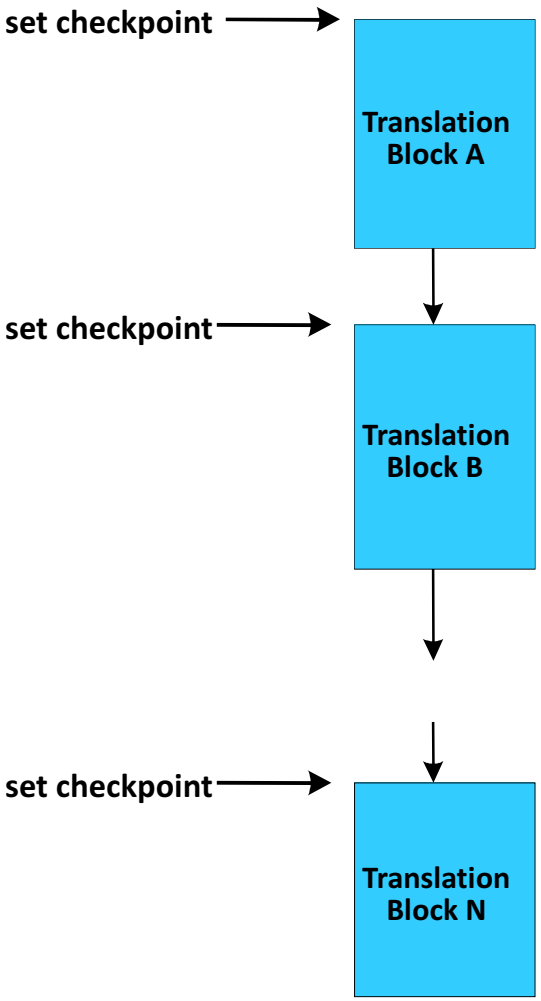
- Traps must be precise wrt original binary
  - All conventional software is unaware of underlying VM
  - Code may undergo heavy duty re-organization
    - E.g. CISC $\rightarrow$ VLIW
- Checkpoint and rollback
  - Have VMM periodically checkpoint state
  - Consistent with a point in original binary
  - On fault, rollback and interpret original binary

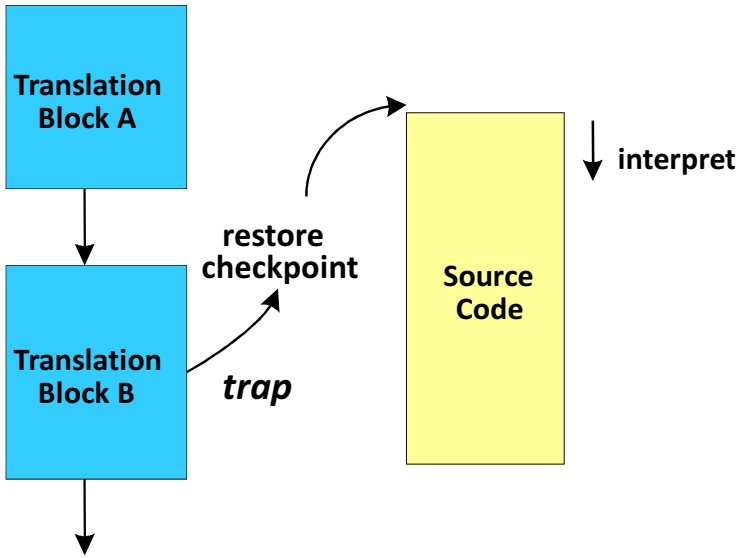
Do We Need Hardware Support?

- If code motion is involved:
  - register live ranges are extended to ensure that checkpoint values can be restored when there is a trap. Then interpretation beginning at a checkpoint prior to a trapping instruction,
  - In codesigned VMs this approach is facilitated because the host ISA can be designed to have enough registers so that live ranges can be extended without excessive register pressure.
- However, this software-based approach does limit the types of code motion that can be performed.
  - For example, many instructions cannot be moved below store instructions.
- By adding hardware support for checkpoints, this restriction on code motion can be removed

Checkpoint and Rollback

Checkpointing can be done in hardware





Page Faults

- Were not an issue in process VMs (why?)
- A codesigned VM is a system VM, not a process VM → The guest OS must observe exactly the same page faults as it would if it were running on a native platform.
- We have to deal with two types of page faults:
  - From conventional memory:
    - For data (no problem)
    - When interpreting instructions
  - From concealed memory
    - When executing translated instructions (more difficult!)

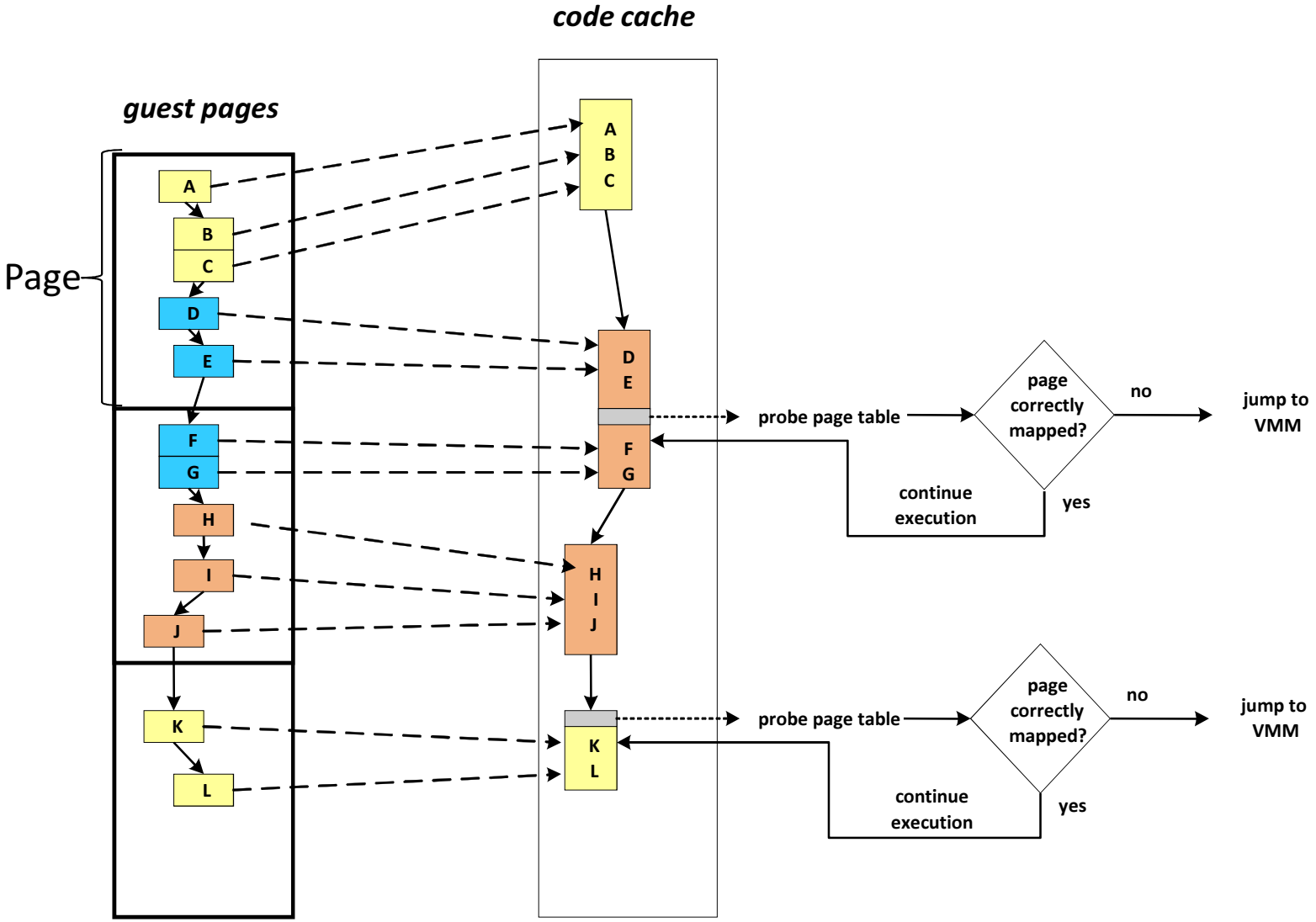
Active Page Fault Detection

- Monitor Guest Page Table updates
  - Make page table read only
  - *Also monitor* page table pointer (just in case page table changes)
- VMM needs to keep track of which pages correspond to which translated blocks.
- When a page holding translated code is removed, flush corresponding code cache entries
  - Requires side tables that map pages to code cache entries
  - As well as corresponding DRAS and JTLB entries
- After flushing, code must be re-interpreted
  - And page fault will be detected at that time

Lazy Page Fault Detection

- when a source code page is replaced by the guest OS, the code cache is not immediately flushed of corresponding translations. It waits until there is an attempt to actually use the translated code.
- Every time the translated code crosses a source page boundary, the page table is probed to see if the mapping is the same as on the original translation
- Probe page table when control crosses page boundary
  - Use special **`Verify_Translation`** instruction (inserted by translator)
    - Arguments: source virtual address
    - real address at time code was translated
  - This instruction probes page table
  - Jump to VMM if there has been a change (or page fault)

Page Crossings



Input/Output

- VMM itself uses no I/O
- Run guest I/O drivers as-is
  - Let I/O drivers directly control I/O signals
- Problems w/ Memory-Mapped I/O
  - Use access-protect in TLB to detect accesses to volatile pages
  - De-optimize code that accesses volatile pages
  - Enhance ISA w/ load/store opcodes that over-ride access-protect

So

- Co-designed VMs enable hardware change that was prohibited by backward compatibility
- Main goals: power efficiency, performance, and/or design simplicity.

Conclusions

- Multiprocessors are available in different setups: multicore, multiprocessor, SMP, distributed or shared memory, ….
- Partitioning resources is key to virtualizing them.
- As computer systems become more sophisticated, the applications for VM emerge.