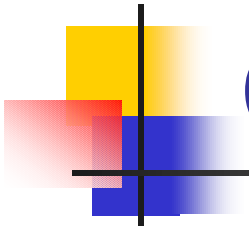
Apropos of Machine Virtualization

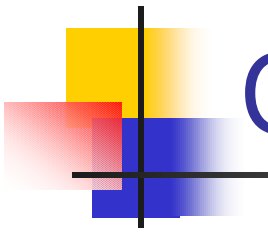
Andrew Whitaker

Steve Gribble



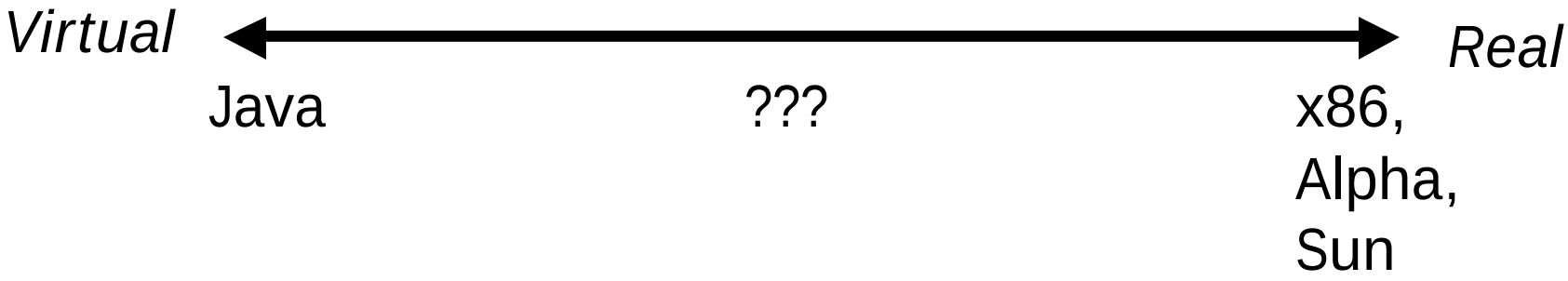
Outline

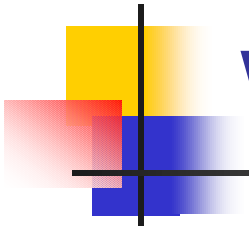
- What virtual architecture?
- Resource management
- Protection
- Where we go from here?



Question

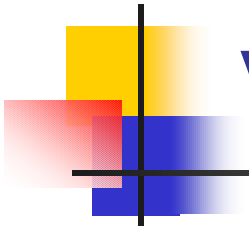
What architecture should we virtualize?





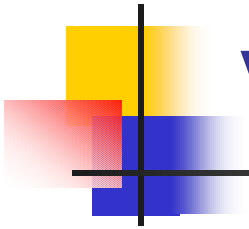
What's in an architecture?

- CPU
  - Instruction set
  - Register file
- Memory subsystem
  - Translation hardware: segmentation, paging
  - Privilege levels: user/kernel, protection rings
- I/O
  - Console, disk, network, and other devices
  - Interrupts and exceptions
- Other stuff: timers, processor modes, BIOS, multiple processors



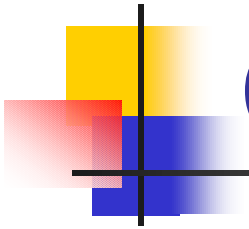
Virtual vs. Real machines

- Normal operating system environment:
  - running in supervisor mode
  - full access to machine state and I/O devices
- Virtualized guest operating systems:
  - running in user mode
  - no direct access to machine state
- Tasks of the virtual machine monitor:
  - reconciling the virtual and physical architecture
  - preventing virtual machines from interfering with each other or the monitor



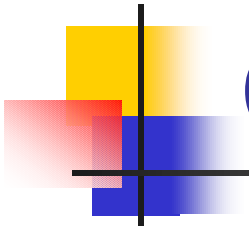
Virtualizable Instruction Sets

- Question: which instruction sets are amenable to implementing VM monitors?
- Definition of virtualizability due to Goldberg (1974):
  - For efficiency, most instructions execute natively
  - “Privileged” instructions must be trapped and emulated
    - accessing processor state: status registers, TLB
    - I/O instructions



Case study: x86

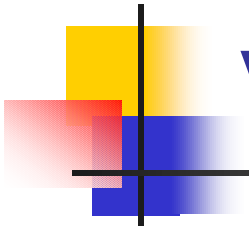
- 17 privileged x86 instructions do not trap in user mode
  - e.g., accessing the interrupt-enabled flag
- Therefore, x86 is NOT virtualizable!
- VMWare uses binary rewriting to insert manual traps
  - also have to worry about self-modifying and self-referencing code



Other Virtualization Issues

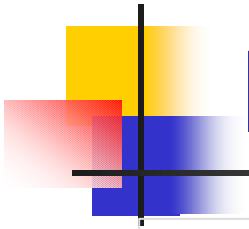
- I/O initialization
- Timers
- Disabling interrupts
- Processor “cruft”: Segmentation hardware, BIOS, processor modes

Faithfully emulating a processor is hard to do!



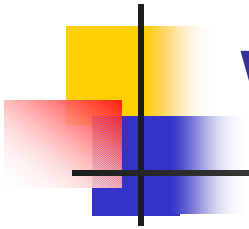
Virtualization revisited

- Observation: Many parts of the physical architecture are rarely used:
  - segmentation, BIOS, I/O devices
- New idea: Para-virtualization — virtual architecture ≠ physical architecture
  - Much easier implementation; improved performance
  - Can no longer use unmodified OS's
  - Can no longer run virtual OS's on raw hardware



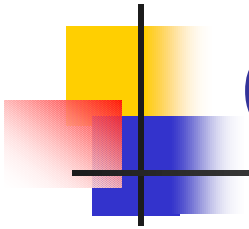
Para-x86

- Instruction set: redefine the semantics of the 17 un-virtualizable instructions
- Memory subsystem:
  - retain paged virtual memory with user/kernel boundary
  - eliminate segmentation and protection rings
- Generic console, disk, network devices
  - no probing of I/O devices
  - acts as a layer of indirection between OS and devices
- Throw out processor modes, BIOS, multiple processors, ...



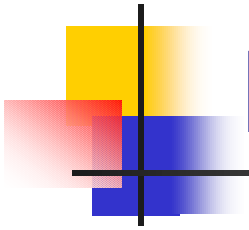
What should we add?

- Idle instruction
- Chunk writes for network packets and disk blocks
- Virtual and physical timers
- Fast interrupt disable/enable



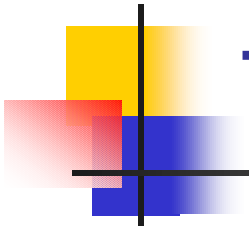
Outline

- What virtual architecture?
- Resource management
- Protection
- Where we go from here



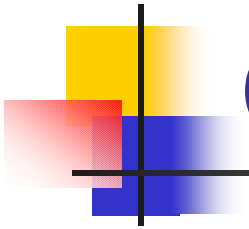
Resource Management

- Allocation goals:
  - *Isolation* between virtual machines
  - *Sharing* of idle resources
- Other factors: revocation cost, minimum resource requirements
- Two categories of resources:
  - time-multiplexed: CPU, network, disk BW
  - space-multiplexed: physical memory



Time-multiplexed resources

- Network bandwidth: use fair-queuing inside the virtual Ethernet segment
  - much simpler than the normal FQ problem
- CPU: Use a proportional share scheduler like lottery/stride scheduling
- Disk bandwidth is tricky because of variable disk access times
  - use a log disk for write-intensive applications (e.g. network monitoring applications)

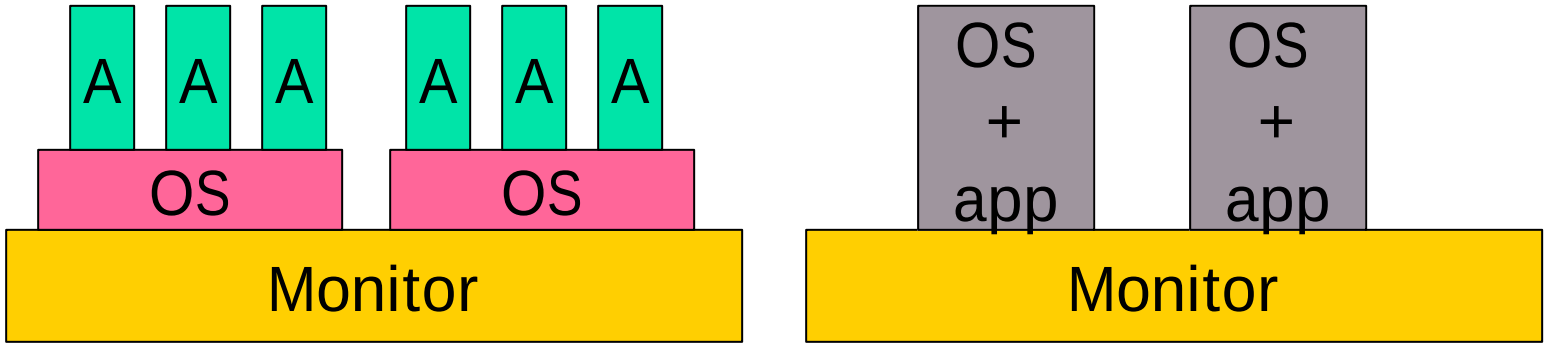


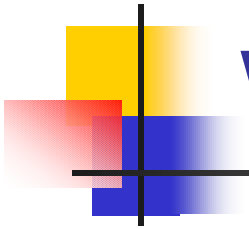
Outline

- What virtual architecture?
- Resource management
- Protection
- Where we go from here

Open question: memory protection

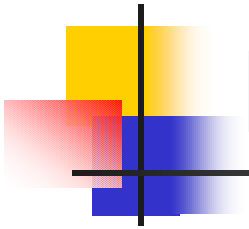
- Two possibilities:
  - Conventional page-level protection between guest OS and applications
  - No protection between guest OS and applications





Where we go from here

- Current status: lot’s of ideas, but no code :(
- Use x86 as an execution platform
- Use an operating system toolkit for building the monitor
- Add functionality in stages
  - Memory management
  - Virtualize the I/O devices
  - Fancy scheduling algorithms
  - Testing and optimizations
- Cool project name?



Memory allocation

- Allocating physical memory is hard:
  - OS's do not explicitly state their memory requirements
  - System must infer the working set
  - Possibility of thrashing
- Solution: static allocation
  - 1 GB memory / 8 MB per machine = 128 virtual machines in memory