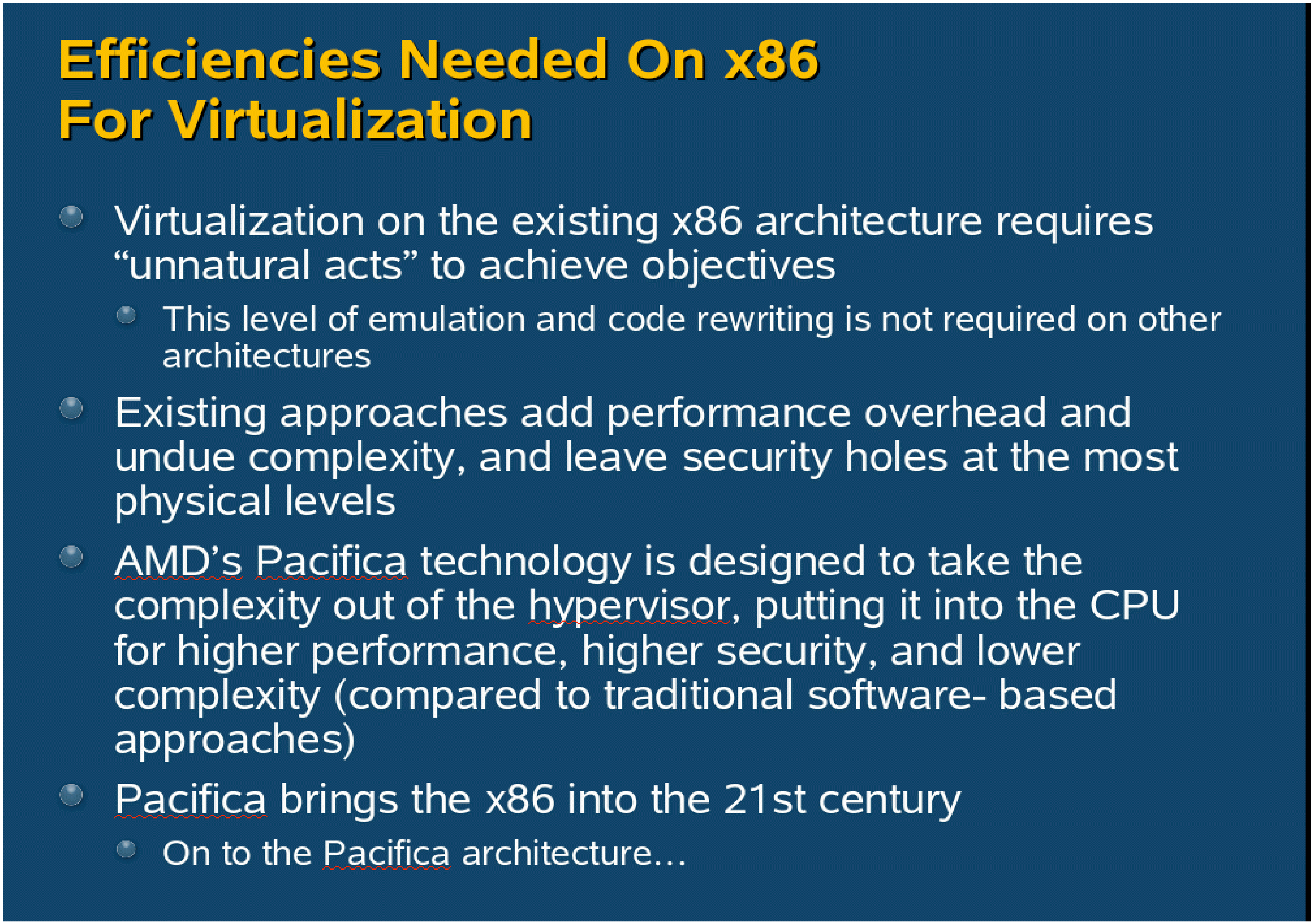
A Comparison of Software and Hardware Techniques for x86 Virtualization (Paper: ASPLOS 2006)

Keith Adams
Ole Agesen

“Unnatural Acts” (WinHEC 2005)



Efficiencies Needed On x86 For Virtualization

- Virtualization on the existing x86 architecture requires “unnatural acts” to achieve objectives
  - This level of emulation and code rewriting is not required on other architectures
- Existing approaches add performance overhead and undue complexity, and leave security holes at the most physical levels
- AMD’s Pacifica technology is designed to take the complexity out of the hypervisor, putting it into the CPU for higher performance, higher security, and lower complexity (compared to traditional software- based approaches)
- Pacifica brings the x86 into the 21st century
  - On to the Pacifica architecture…

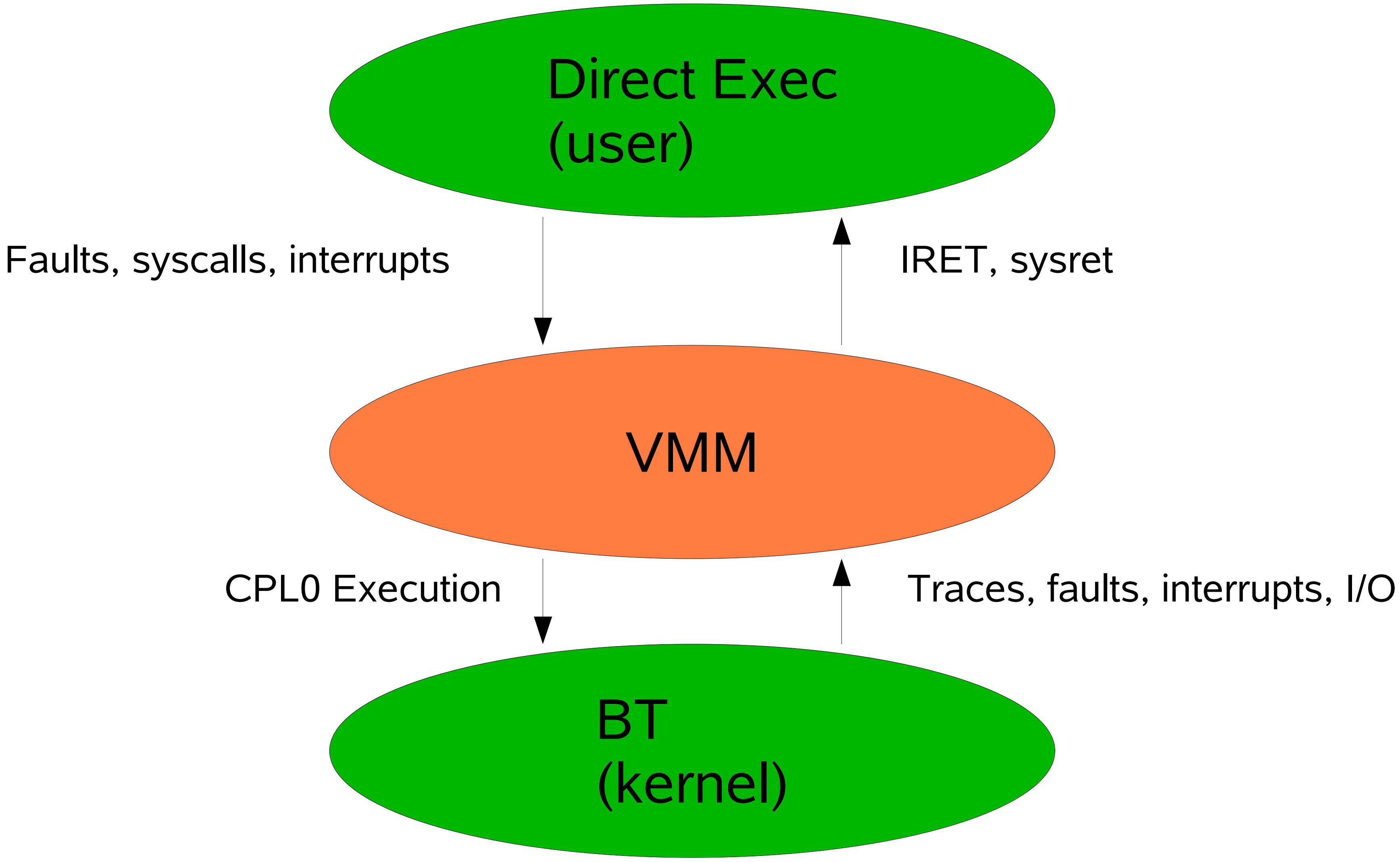
Classical virtualization (IBM 360)

- Trap-and-emulate
  - Run guest operating system *deprivileged*
  - All privileged instructions *trap into VMM*
  - VMM emulates instruction against virtual state
    - E.g.: disable virtual interrupts, not physical interrupts
  - Resume *direct execution* from next VM instruction
- This is just one **implementation technique**
- Popek and Goldberg permit others

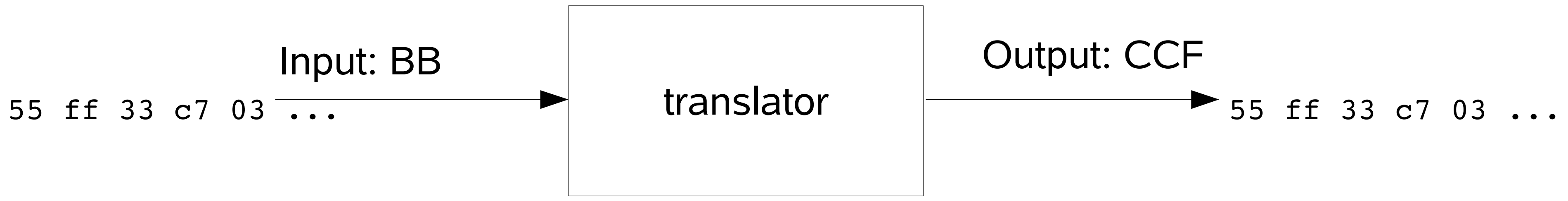
Classical VM performance

- Native speed except for traps
  - Overhead = trap frequency * avg trap cost
- Trap sources
  - Privileged instructions
  - Traces (page tables): *most frequent trap cause*
  - Memory mapped devices (a form of trace)

Combining BT and direct execution



BT mechanics



- Each translator invocation
  - Consume one basic block
  - Produce one compiled code fragment
- Store CCF in translation cache
  - Future reuse
  - Amortize translation costs
  - Not “patching in place”

IDENT and other translations

- Most code translated IDENT
- Runs at speed (Popek, Goldberg)

```
80304a69 push %ebp
80403a6a push (%ebx)
80403a6c mov  (%ebx), ffffffff
80403a72 mov  %edx, %esp
80403a74 mov  %esp, 81c(%ebx)
80403a7a push %edx
80403a7b mov  %ebp, %eax
80403a7d call 80460ba4
```

BB

$\longrightarrow$

```
25555b0 push %ebp
25555b1 push (%ebx)
25555b3 mov  (%ebx), ffffffff
25555b9 mov  %edx, %esp
25555bb mov  %esp, 81c(%ebx)
25555c1 push %edx
25555c2 mov  %ebp, %eax
25555c4 push 80403a82
25555c9 int  3a
25555cb data: 80460ba4
```

CCF

25555c4: push return address
25555c9: invoke translator on callee

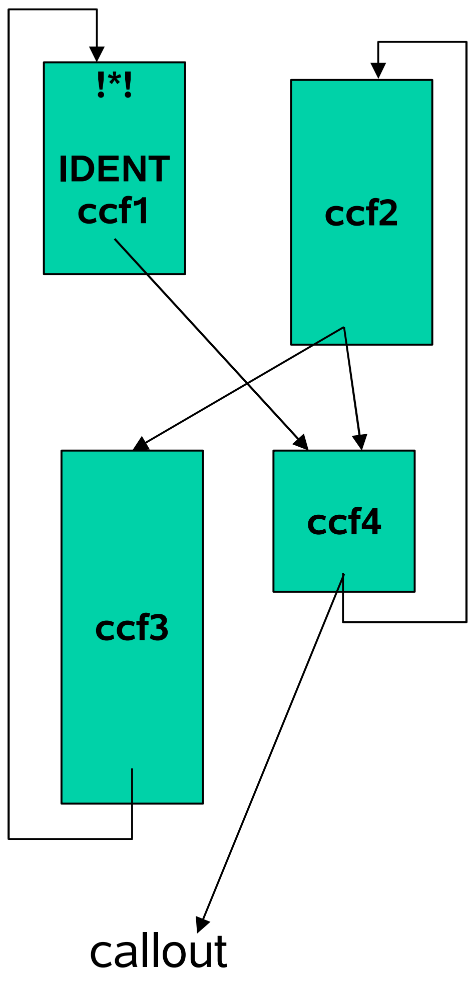
Primary and shadow structures

- **Shadow** page tables active on physical CPU
- Contain composite of two mappings:
  - Guest “virtual” to “physical” mapping from **primary** page tables
  - Guest physical to machine memory mapping from VMM pmap
- Result: physical TLB supports guest memory access

Traces

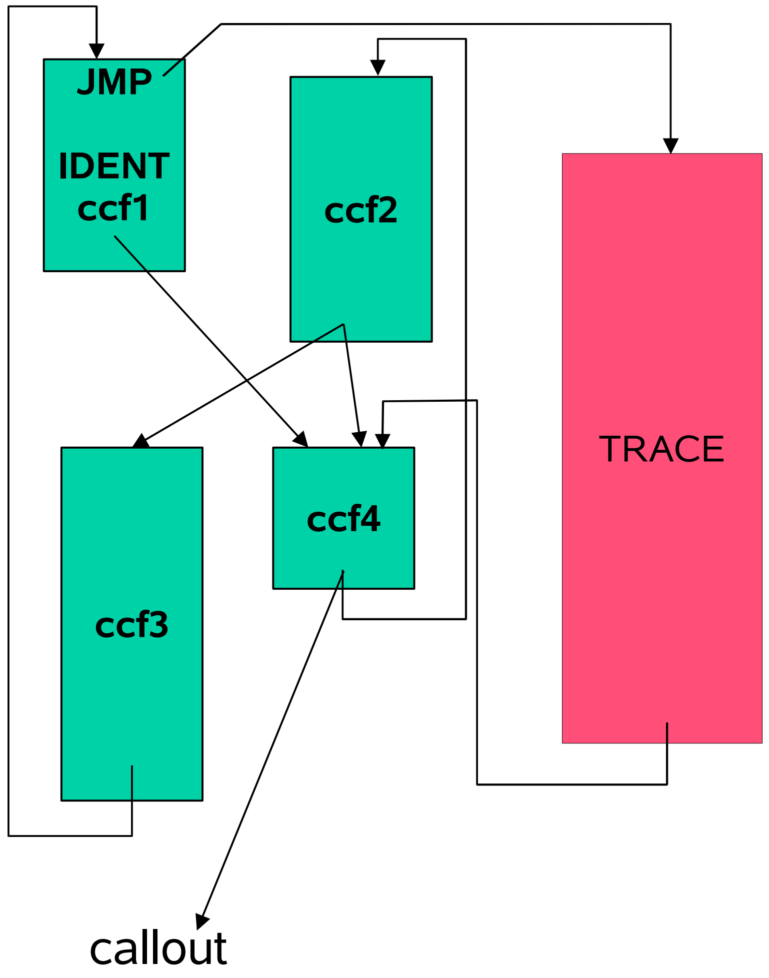
- Shadow page tables are derived from primaries
  - Must keep in sync
- Coherency protocol
  - Trap guest writes by write-protecting primary: **trace**
  - Propagate change from primary to shadow
    - Compute new shadow page table entry
    - Or just invalidate
  - x86 permits deferred coherency
    - Like hardware TLB
    - INVLPG: synchronize one page's mapping
    - CR3: synchronize entire address space

Adaptive BT



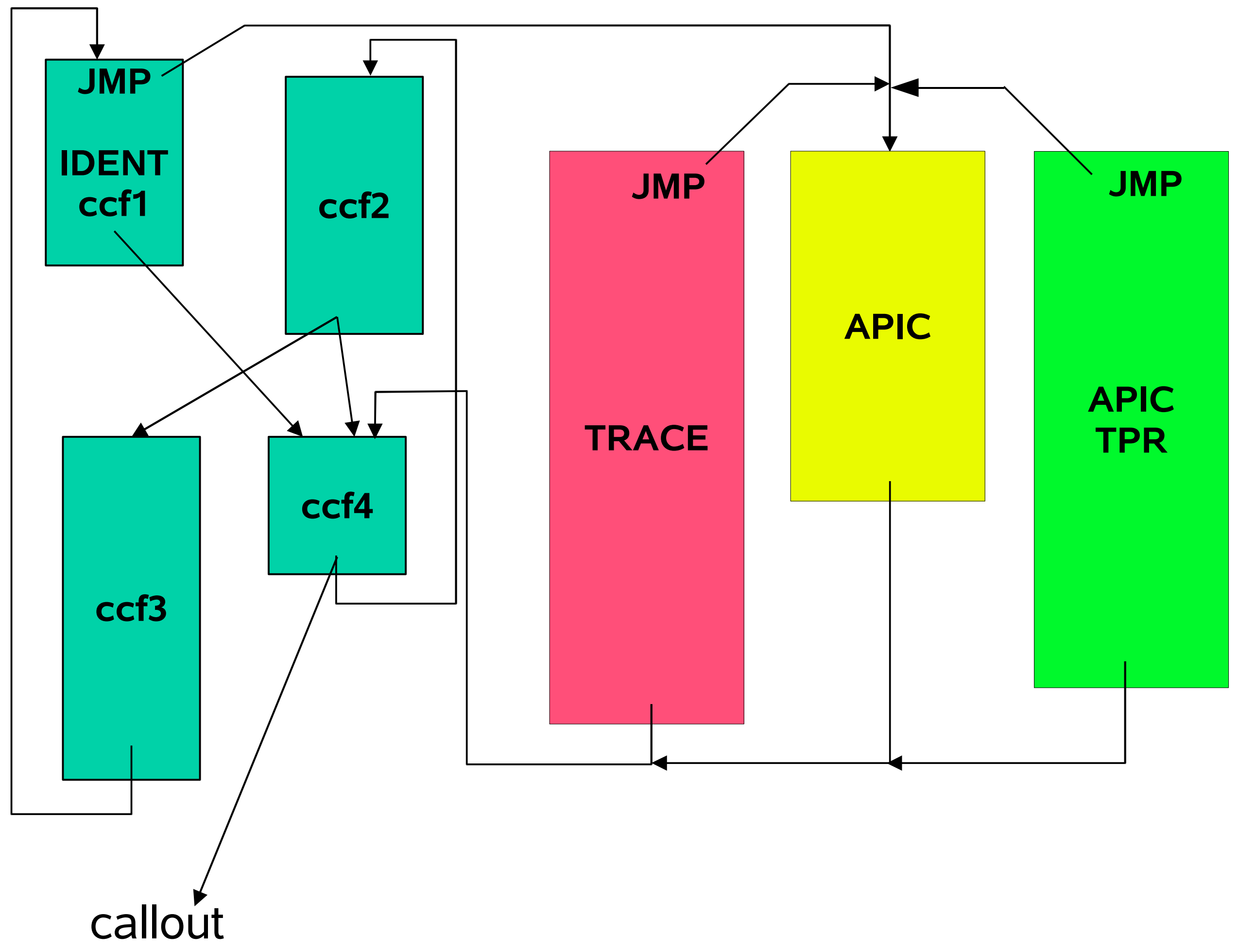
- Most translation run “at speed”
- Exception: writing traced memory
  - #PF (trace fault !*!)
  - Decode instruction
  - Interpret
  - Fire trace callbacks
  - Resume execution
- Can take 1000s of cycles

Adaptive BT: fast trace handling



- Detect and track trace faults
- Splice in TRACE translation
  - Avoid #PF
  - No re-decoding
  - Faster resumption
- Order of magnitude faster traces

Adaptation in general



Adapt to guest behavior by moving JMPs around to select active translation

Software VMM evaluation

- Benefits
  - Adaptation
  - Fast traces
  - Fast I/O emulation
  - Flexibility
- Costs
  - Running translator
  - Path lengthening
  - System call slowdown
  - Complexity

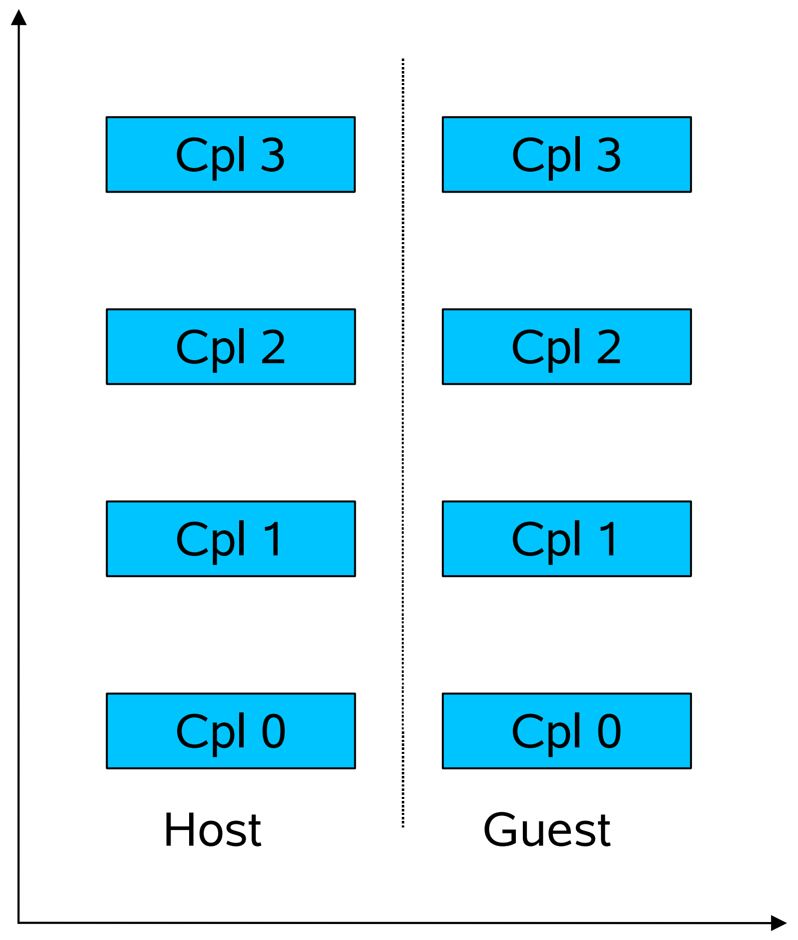
AMDel, Inc.

- Summer 2001 at AMDel, Inc.
- Architecting the **Penteron 3++**, due in 2005
- Virtualization is hot, *hot*, ***hot***
- How can the next-generation Penteron support virtualization?

100% direct execution

- VMMs nowadays are slow
- VMMs in the mainframe era weren't so slow
- VMMs nowadays use binary translation
- BT must be slowing things down. **QED**

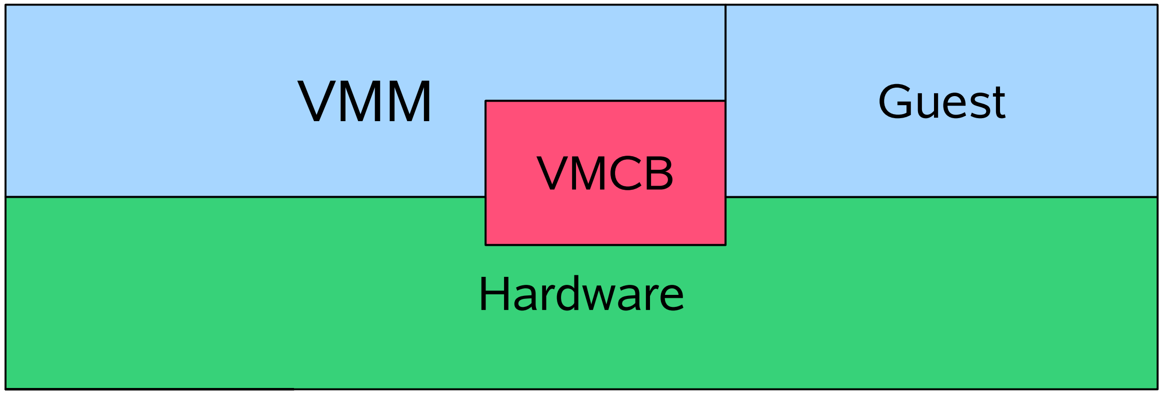
VT/SVM Architecture



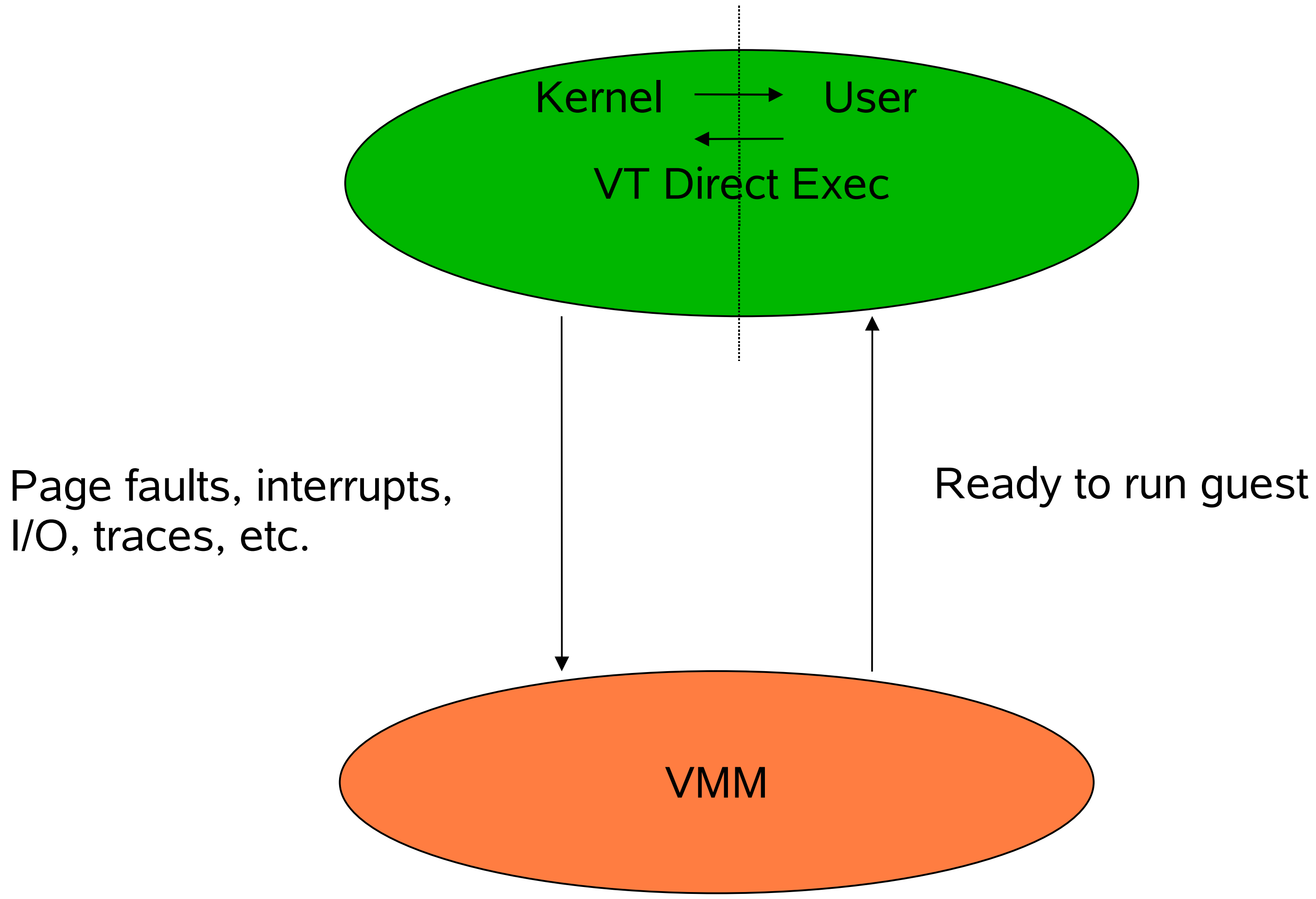
- Y-axis: old school x86 privilege (CPL)
- X-axis: virtualization privilege
- Unmodified OS'es run in host mode, guest mode
- In guest mode, sensitive operations “trap out” to host mode

VMCB

- *Virtual Machine Control Block*
- VMM controlled, hardware-walked
- Buffers simple exits
- Communicates guest state to HW, VMM
- In VT, read/written via special instructions



Guest and host mode execution



What SVM/VT are not

- “CPL -1”
  - No special mode for VMM
  - VMM no more privileged than a regular OS
- Pure trap-and-emulate
  - Many privileged ops buffered by VMCB
  - Special accommodations for some ops
    - E.g., RDTSC (cycle counter) can have offset applied

Hardware VMM evaluation

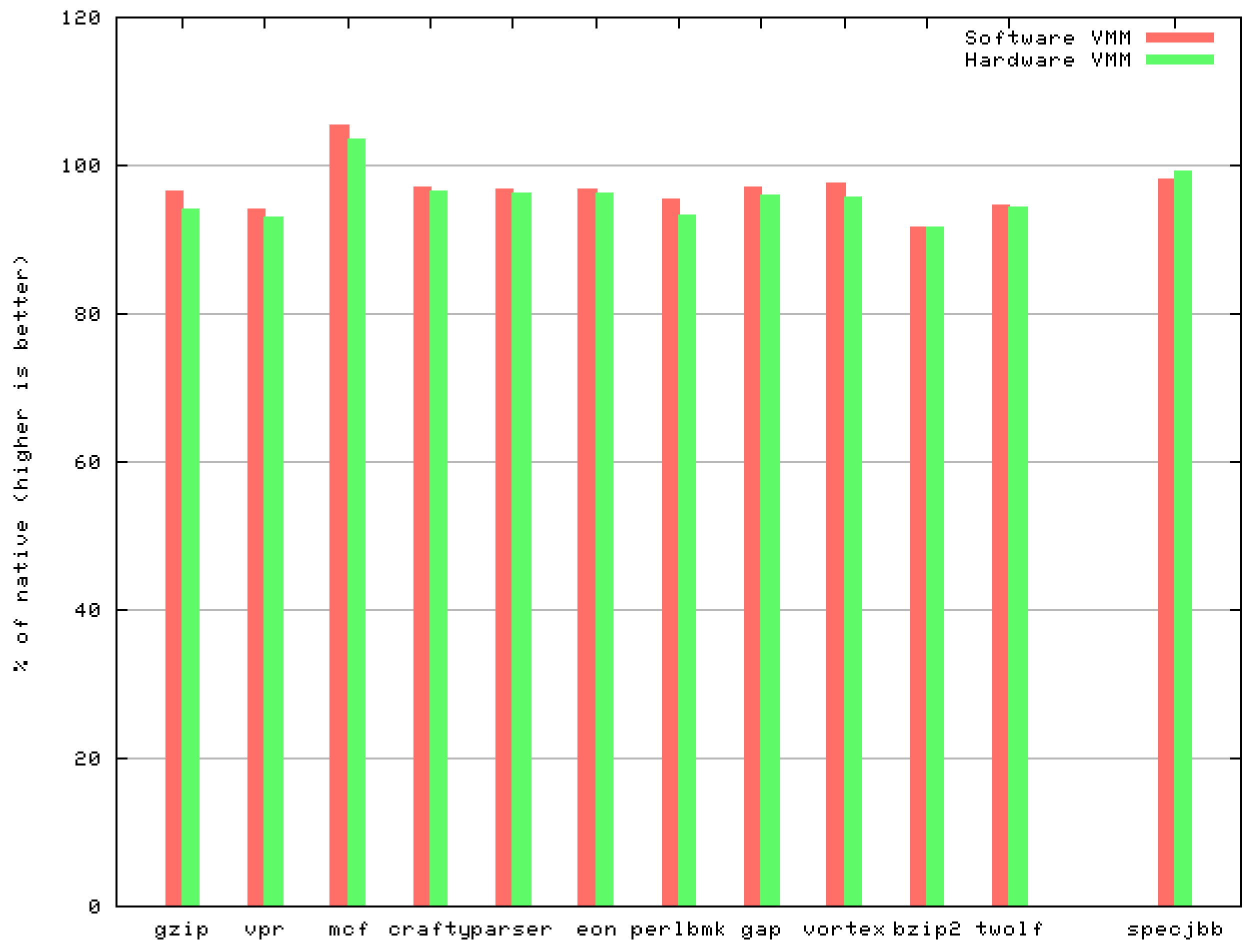
- Benefits
  - Simplicity (no BT)
  - Fast system calls
  - No translator overheads

- Costs
  - Exits: 1000+ cycles
    - Traces
    - I/O
  - Stateless model
  - No adaptation
  - No SW flexibility

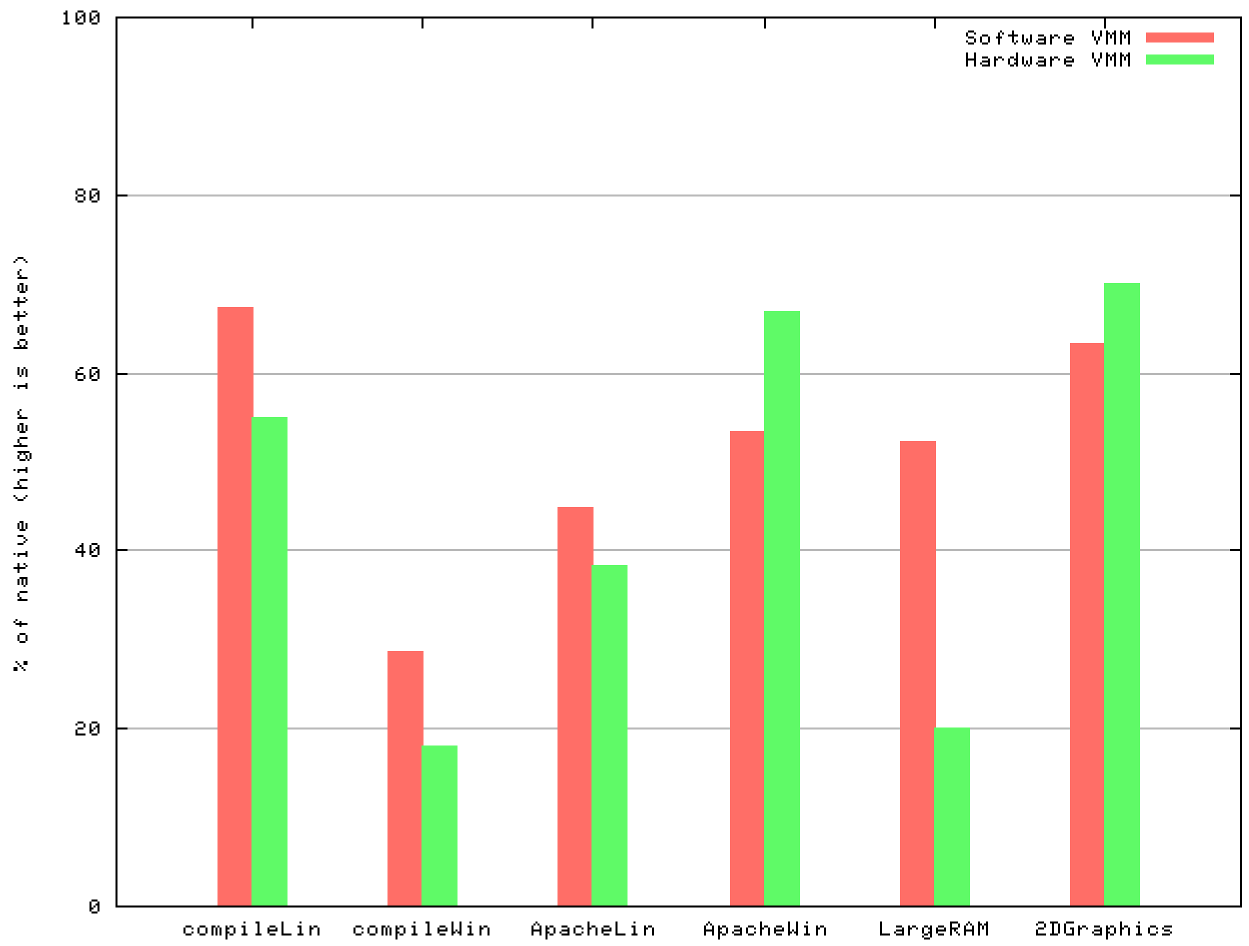
Match lineup

- System: Intel Pentium 4 672, 3.8 GHz, VT
- Software VMM: VMware Player 1.0.1
- Hardware VMM: VMware Player 1.0.1 (same!)

Pure computation (SPEC)



A tougher set of benchmarks



Microbenchmark: forkwait

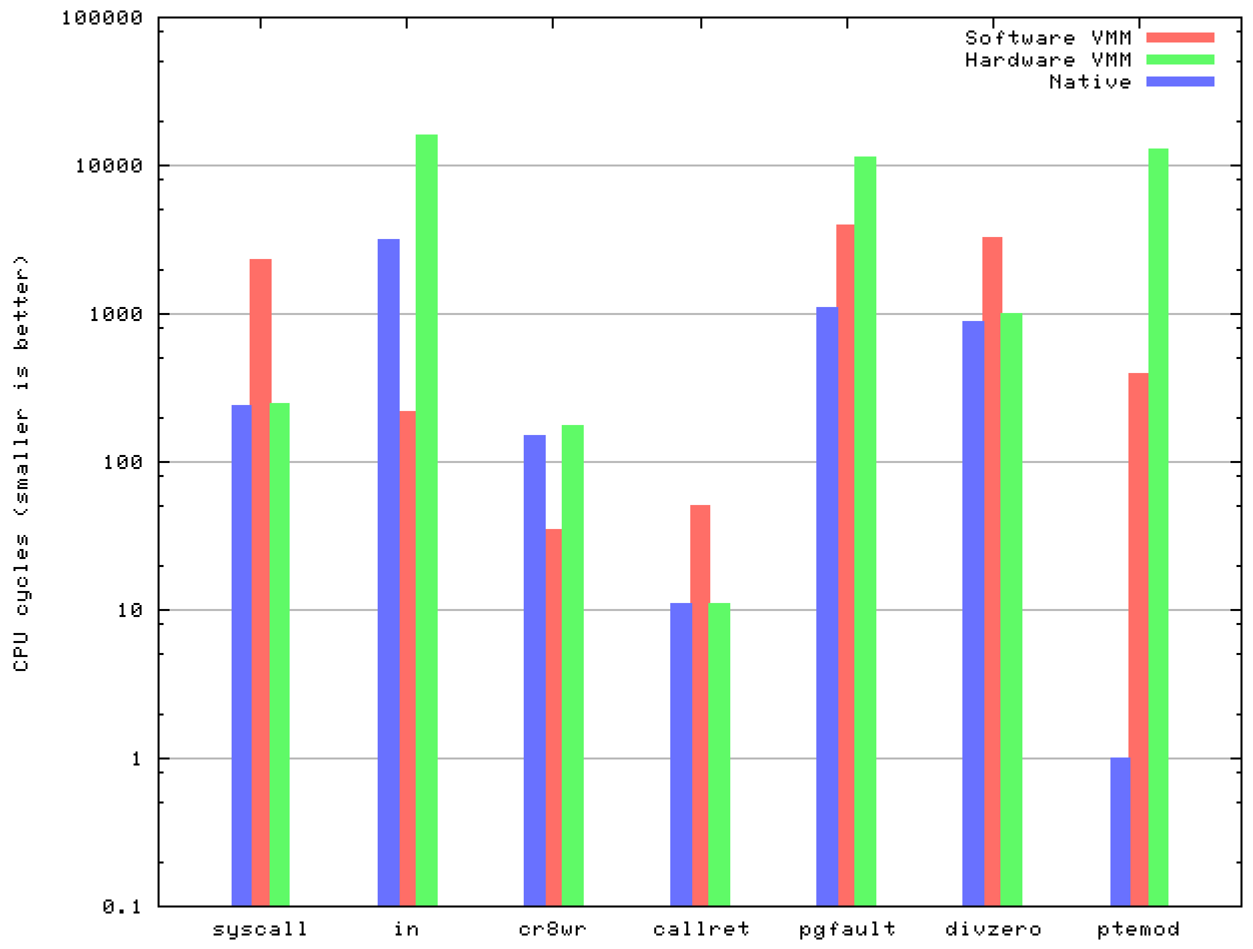
```
int main(int argc, char *argv[]) {
  for (int i = 0; i < 40000; i++) {
    int pid = fork();
    if (pid < 0)  return -1;
    if (pid == 0) return 0;
    waitpid(pid);
  }
  return 0;
}
```

Native:	6.0 seconds
Software VMM:	36.9 seconds
Hardware VMM:	106.4 seconds

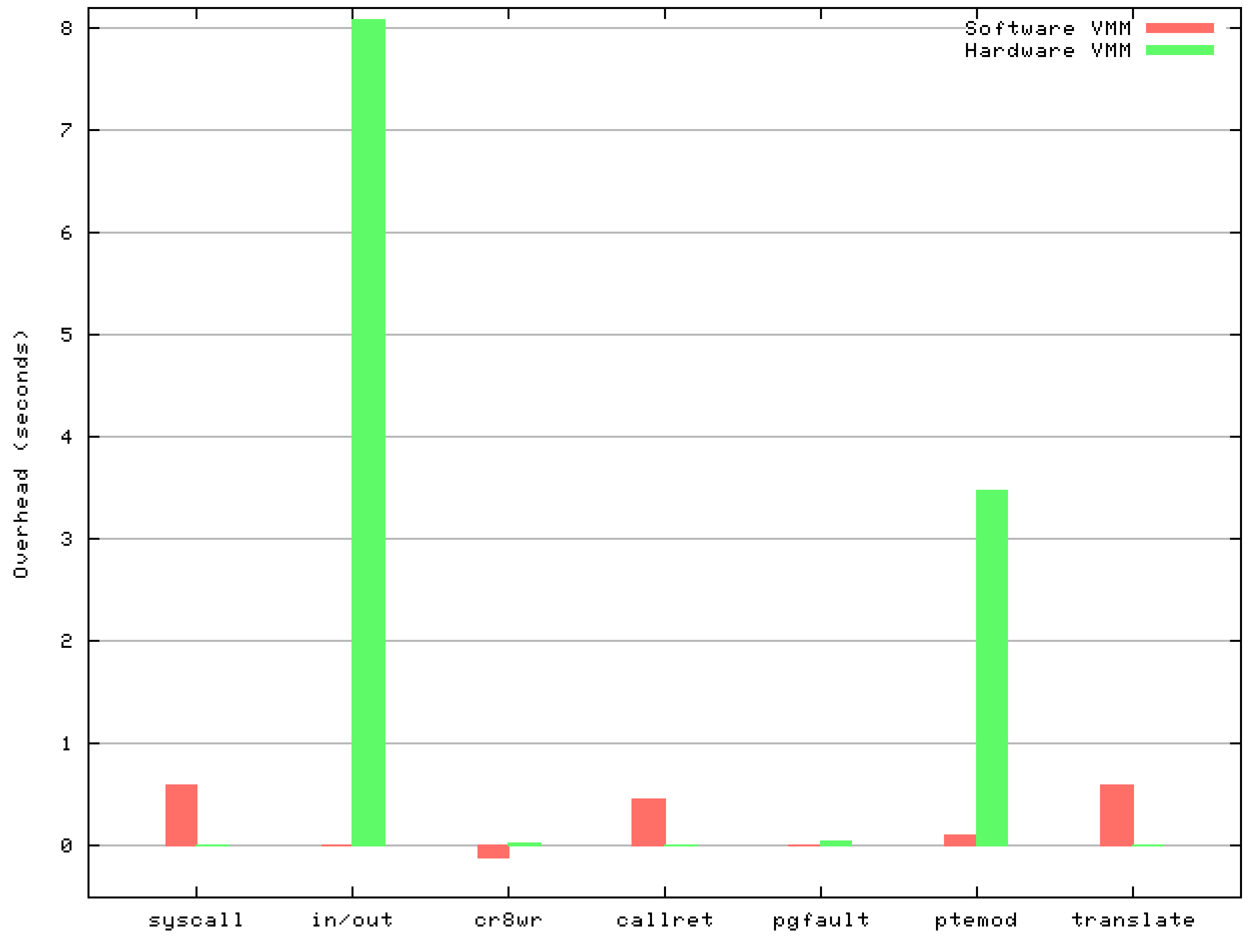
Nano vs. micro benchmarks

- Forkwait: pathologically rich mix of
  - Context switches
  - Page table updates
  - Page faults
- But still a mixture of operations
- Nano-benchmark idea
  - measure each event in isolation
  - often “single-instruction” events

Nanobenchmarks



Decomposing a macro-benchmark
XP64 boot/halt



Outline

- Virtualization primer
- Software VMM
- Hardware VMM
- Performance comparison
- **Lessons learned**
- Conclusions

Lessons learned

- Current HW support not a uniform win
  - Sometimes a bit better (e.g., system calls)
  - Sometimes quite a bit worse (MMU virtualization)
- Ideal: combine strengths of SW/HW
  - Unfortunately: HW support is all or nothing
  - Does not complement existing SW techniques

Virtualization is everywhere

- Server consolidation
- Disaster recovery
- Security
- Resource management
- Power efficiency
- Availability (fail-over to other VM)
- Software delivery (“virtual appliances”)
- Test and development
- Mobility . . .

The essence of the MMU problem

- Three cost components
  - Trace costs
  - Context switch costs
  - Hidden page faults (demand-validation of shadows)
- These trade against each other:
  - Fewer traces implies more hidden #PFs (lazy) or more expensive context switches (eager)
- Hardware VMM increases costs of all three components!

Improving virtual MMU performance

- Hardware VMM still uses software MMU
  - Drives MMU state machine with VT exits
  - Our software MMU was designed for BT costs
- Software option:
  - MMU redesign to hit different point in 3-way tradeoff
  - Less use of traces
- Hardware option:
  - Support MMU virtualization
  - Nested page tables

Conclusions

- Described software VMM and hardware VMM
- Performance comparison
  - From macro to micro to nano
- Current hardware VMM not a win over software
- Key problem to solve:
  - Not: how to execute virtual instruction stream
  - But: how to virtualize MMU efficiently
  - Combining HW support with SW flexibility?

Win2000 boot/halt translation stats

```
        -------input-------           output
    #   units   size  instr  cycles    size cyc/ins ins/unit
-------------------------------------------------------------
    0   38690   336k   120k    252M    924k    2097      3.11
    1   48839   500k   169k    318M   1164k    1871      3.48
    2    108k  1187k   392k    754M   2589k    1920      3.61
    3   29362   264k  89749    287M    951k    3197      3.06
    4   96876  1000k   337k    708M   2418k    2100      3.48
    5   58553   577k   193k    403M   1572k    2078      3.31
    6   19430   148k  50951    148M    633k    2904      2.62
    7   13081  87811  30455    124M    494k    4071      2.33
-------------------------------------------------------------
Total    413k  4101k  1384k   2994M  10748k    2161      3.35
-------------------------------------------------------------
```

Translator buzzwords

- Binary: input is x86 “hex” not source
- Dynamic: interleave translation and execution
- On demand: translate only what we are about to execute (lazy)
- System level: make no assumptions about guest code
- Subsetting: full x86 to safe subset
- Adaptive: adjust translations in response to guest runtime behavior

Outline

- Virtualization primer
- Software VMM
- Hardware VMM
- **Performance comparison**
- Lessons learned
- Conclusions

Dealing with x86

- Not classically virtualizable (popf)
- Want instruction-level control over guest
- Non-solutions:
  - Code patching
    - Problem: guest can inspect its own code
  - Prescanning pages for nonvirtualizable instrs
    - Problem: guest can jump into middle of instr
    - Problem: what if we find badness?
  - Interpretation
    - Problem: inefficient – x86 decoding slow

Binary translation of guest code

- No need for traps
- Satisfies Popek and Goldberg
  - Fidelity: instruction-level semantic precision
  - Isolation: translate from full x86 to safe subset
  - Performance: most instructions need no change
- It *is* a VMM
- Mature technology
  - Smalltalk, JVMs, Shade, Pin, Embra, Dynamo, etc.

Outline

- Virtualization primer
- **Software VMM**
- Hardware VMM
- Performance comparison
- Lessons learned
- Conclusions

Market size => hardware support

- First round shipping now:
  - Intel: VT-x (Vanderpool)
  - AMD: SVM (Pacifica)
- Transition from sw to hw VMM can start

- But *should* it?
  - We compare and contrast sw/hw techniques
  - Metrics: **performance**, flexibility, complexity

Popek and Goldberg (1974)

- A Virtual Machine Monitor (VMM) provides:
  - Fidelity
  - Performance
  - Safety (isolation)

x86 virtualization has grown up

- From the desktop (1999)
  - “Run windows on your linux computer”
- To enterprise data centers (2006)
  - Bare-metal hypervisor
  - Virtual SMP support
  - Large memories
  - 64 bit support

- All in software, on standard x86 hardware!

Outline

- **Virtualization primer**
- Software VMM
- Hardware VMM
- Performance comparison
- Lessons learned
- Conclusions

Outline

- Virtualization primer
- Software VMM
- **Hardware VMM**
- Performance comparison
- Lessons learned
- Conclusions