Hardware Support for Spin Management in Overcommitted Virtual Machines

Philip Wells **Koushik Chakraborty**
Gurindar Sohi

{pwells, kchak, sohi}@cs.wisc.edu

THE UNIVERSITY of WISCONSIN MADISON

Paper overview

- **Want to run unmodified OSs in overcommitted VM**
  - But OS will spin excessively
- **Propose hardware spin detection to mitigate**
  - Allows more flexible scheduling
  - Enables other applications
- **Server consolidation case study**
  - Improve throughput & performance isolation

Talk outline

Background & motivation

- Overcommitted VM: What & why?

Problem with overcommitted VMs

Hardware spin detection

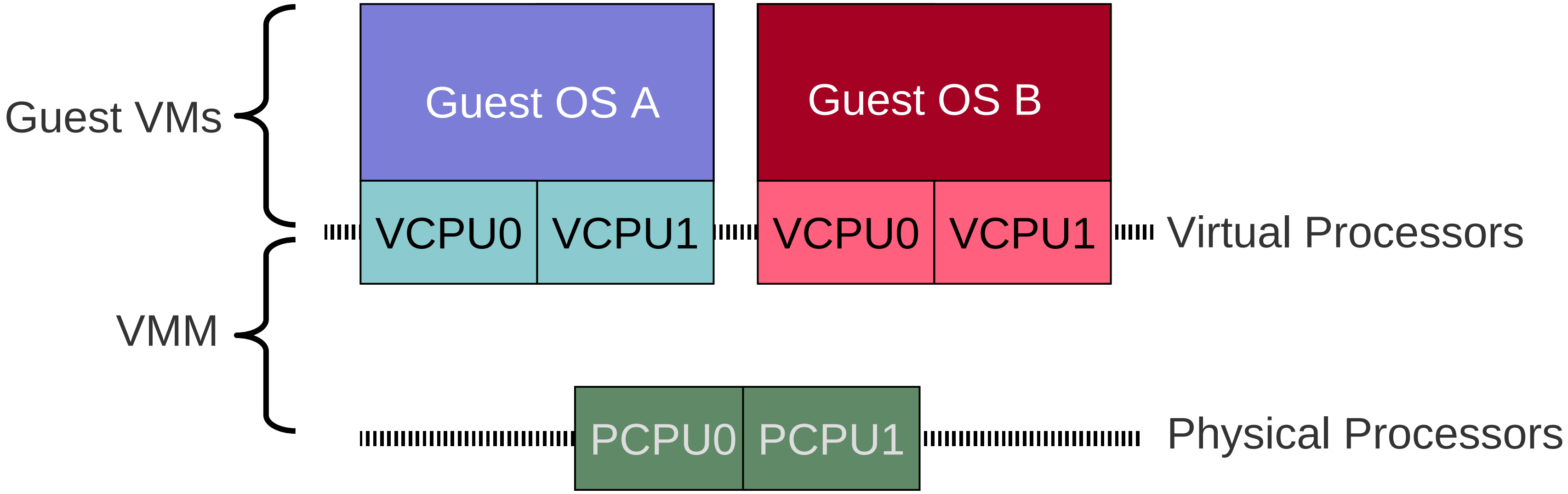
Case study

Summary

Background: VMMs

- **Virtual Machine Monitors (VMMs)**
  - Translate interface *exposed* by hardware into interface *expected* by OS
- **Focus on *pure virtualization***
  - No modifications to OS (e.g. VMWare)
- **Focus on *processor virtualization***
  - Mapping virtual processors to physical processors

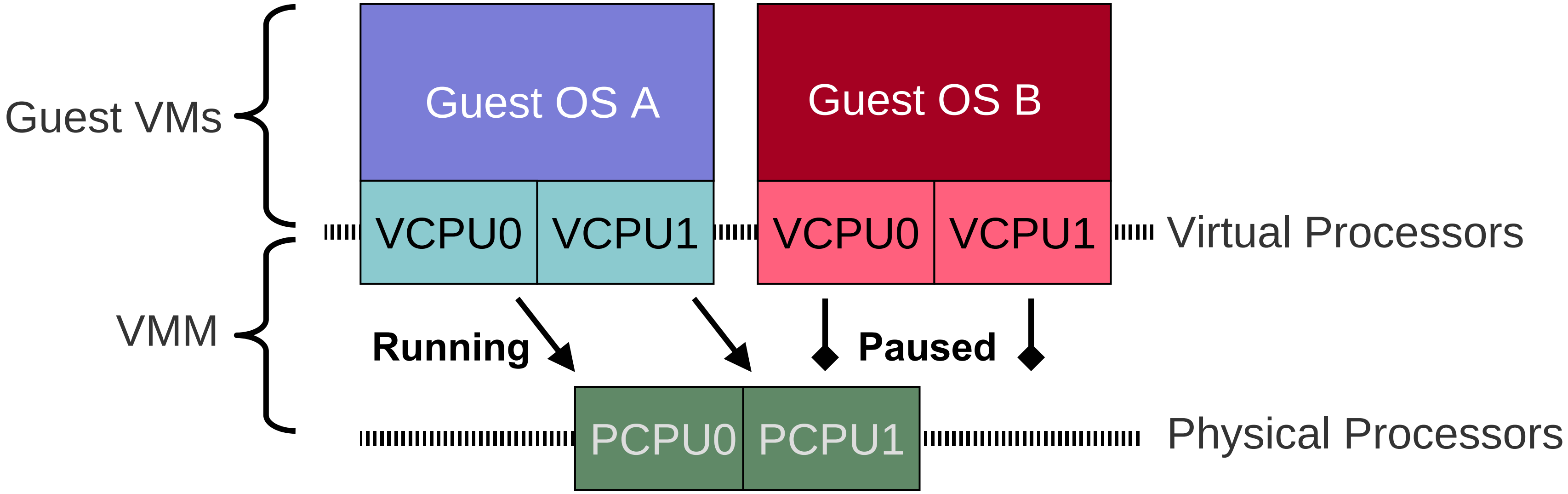
Background: Processor virtualization



VMM exposes more VCPUs than PCPUs

- **Machine is *overcommitted***
- **How to map VCPUs to PCPUs?**

Background: Processor virtualization

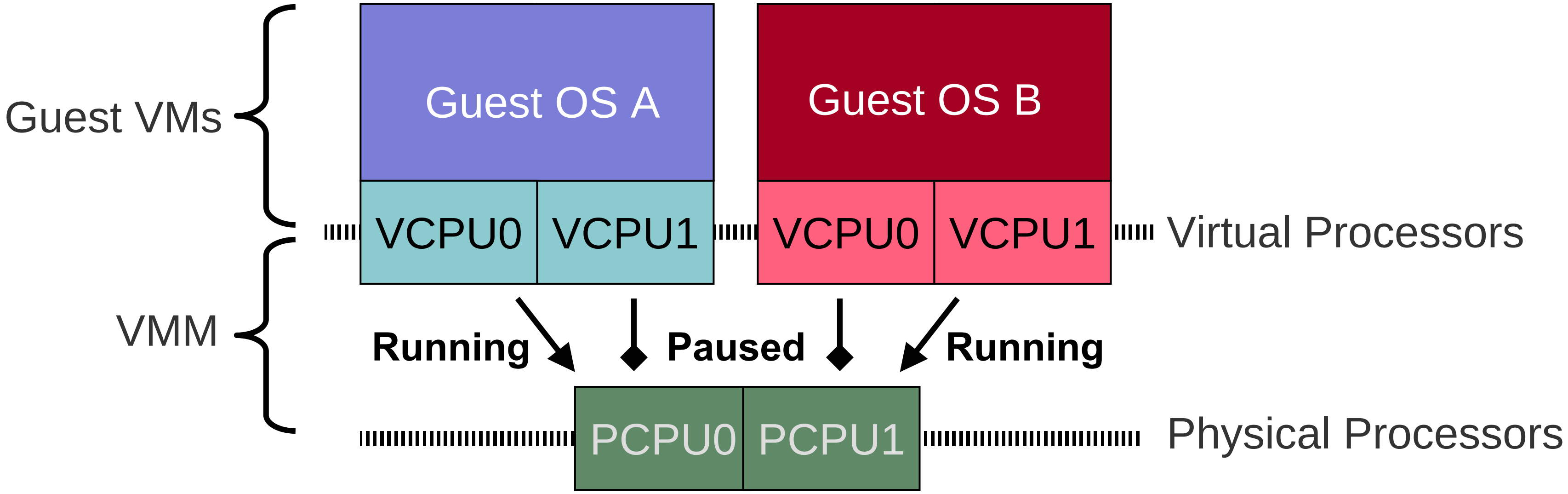


Gang scheduling (or co-scheduling)

- All VCPUs are running, or none are
- Ensures environment similar to non-virtualized
- Used, e.g., by VMWare, Cellular Disco

➢ **But not flexible**

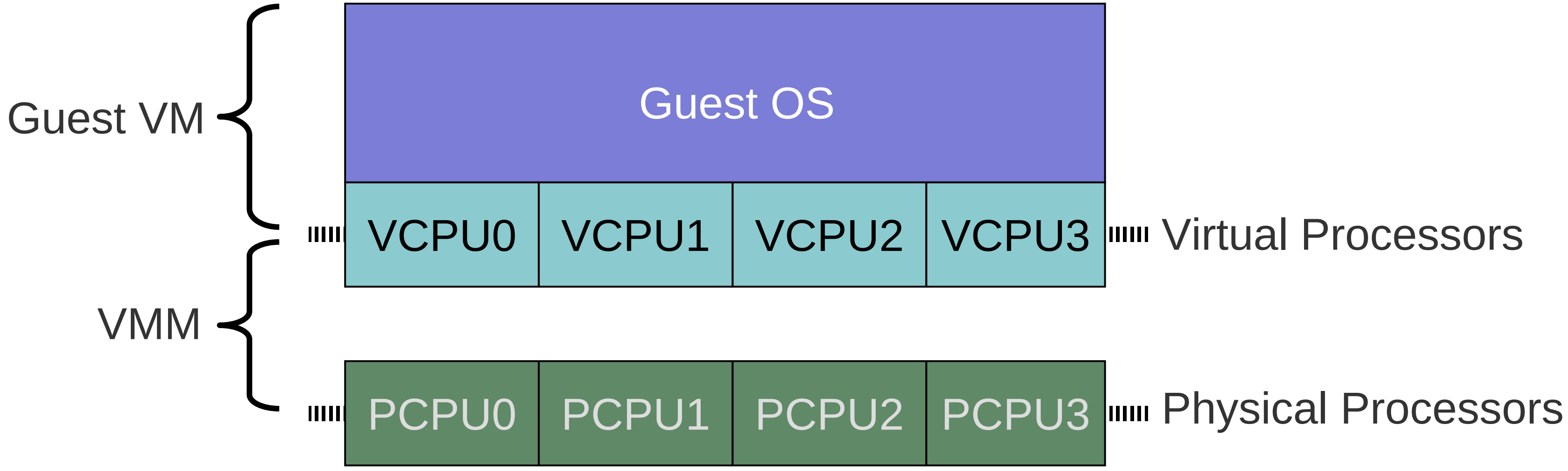
Background: Processor virtualization



Desire to restrict available PCPUs

- Individual guest VM is ***overcommitted***
- **Server consolidation case study later…**

Background: Processor virtualization



Desire to restrict available PCPUs

- **Thermal management – without OS support**

  [e.g. *Heat & Run,* Powell, ASPLOS '04]

- **Dynamic specialization**

  [e.g. *Computation Spreading,* Chakraborty, ASPLOS '06]

➢ **Gang scheduling infeasible**

Talk outline

- Background & motivation
- Problem with overcommitted VMs
  - Spin overhead: How much and why?
- Hardware spin detection
- Case study
- Summary

So, what’s the problem? Spin!

Multiprocessor OSs make assumption:

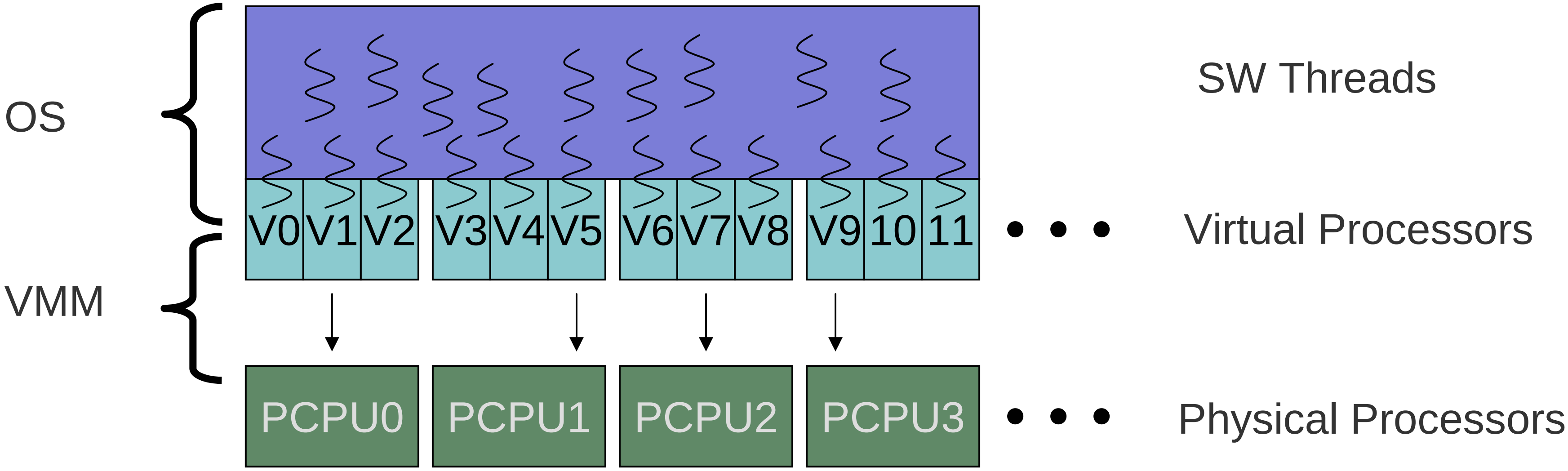
➢ **All VCPUs are always executing**

- **Clearly not possible in overcommitted environment**
- **Causes severe performance problems with synchronization**

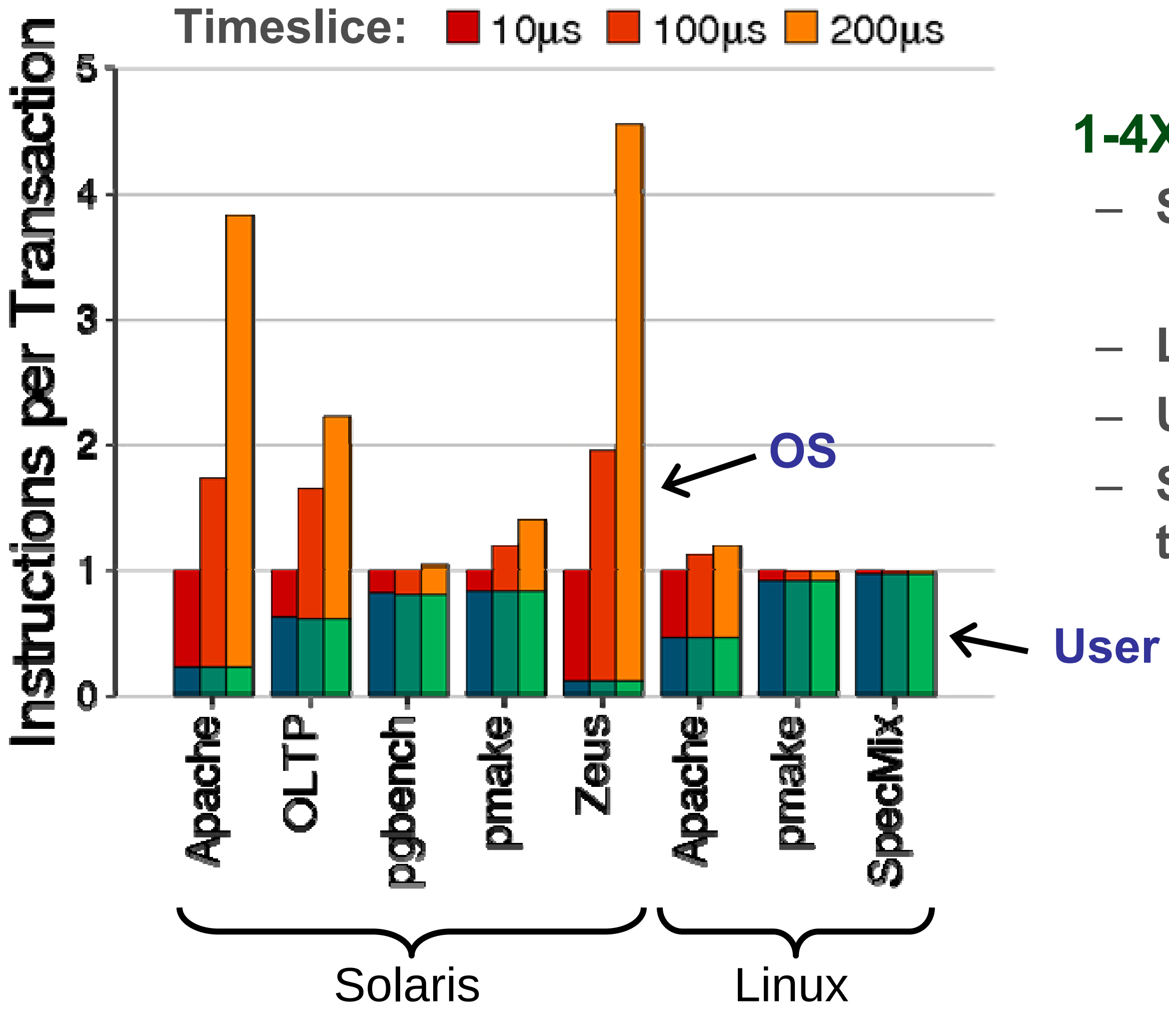
OS spin: Methodology

Highly overcommitted for illustration

- 24 VCPUs, 8 PCPUs
- Simics full-system simulation running Solaris & Linux (SPARC)
- VCPU is given a *timeslice* on PCPU
- Share physical processor, TLB arrays, caches
- Assume HW mechanism for switching VCPU state



OS spin: Overhead



1-4X more instructions

- Same amount of work
  - ➢ Spinning!
- Longer slices worse
- User instr stable
- Solaris spins more than Linux

OS spin: Where does it come from?

OS mutex locks

- Requester often spins on held lock
- Especially if OS thinks lock holder is currently executing

Frequent cross-calls (software interrupts)

- TLB shootdowns & scheduling
- Initiator blocks until recipient processes interrupt
- *Much* more frequent in Solaris than Linux

Other workloads have user spin and idle loop

➢ Propose **hardware spin detection** to mitigate

Talk outline

- Background & motivation
- OS spin overhead
- Hardware spin detection
  - Spin Detection Buffer (SDB)
- Case study
- Summary

Hardware spin detection

Observation:

> ***A program that’s not performing useful work makes few changes to program state***

Use hardware to detect changes to state

- **Requiring *no* changes misses cases of spinning**
  - Or even temporally silent changes…
- **Allowing too many changes causes *false positives***
  - Performance (not correctness) issues if too frequent

No software modifications

Hardware spin detection cont…

Proposed heuristic takes the middle ground

- Observe < 8 *unique* stores in 1k commits ➔ spin
- Uniqueness defined by address & data
- Works very well for OS

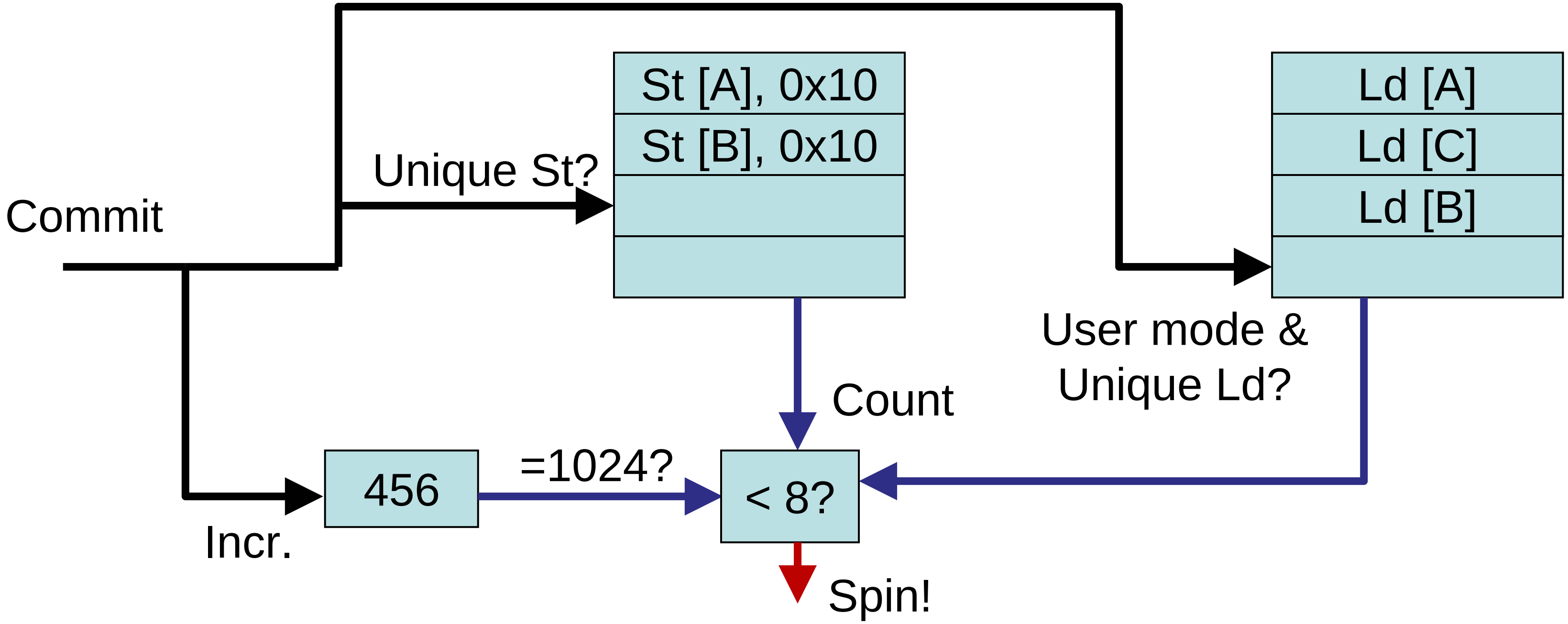
But, user programs search

- Register allocated index ➔ no stores
- Also check for < 8 unique loads in user code
- Avoids false positives from user code

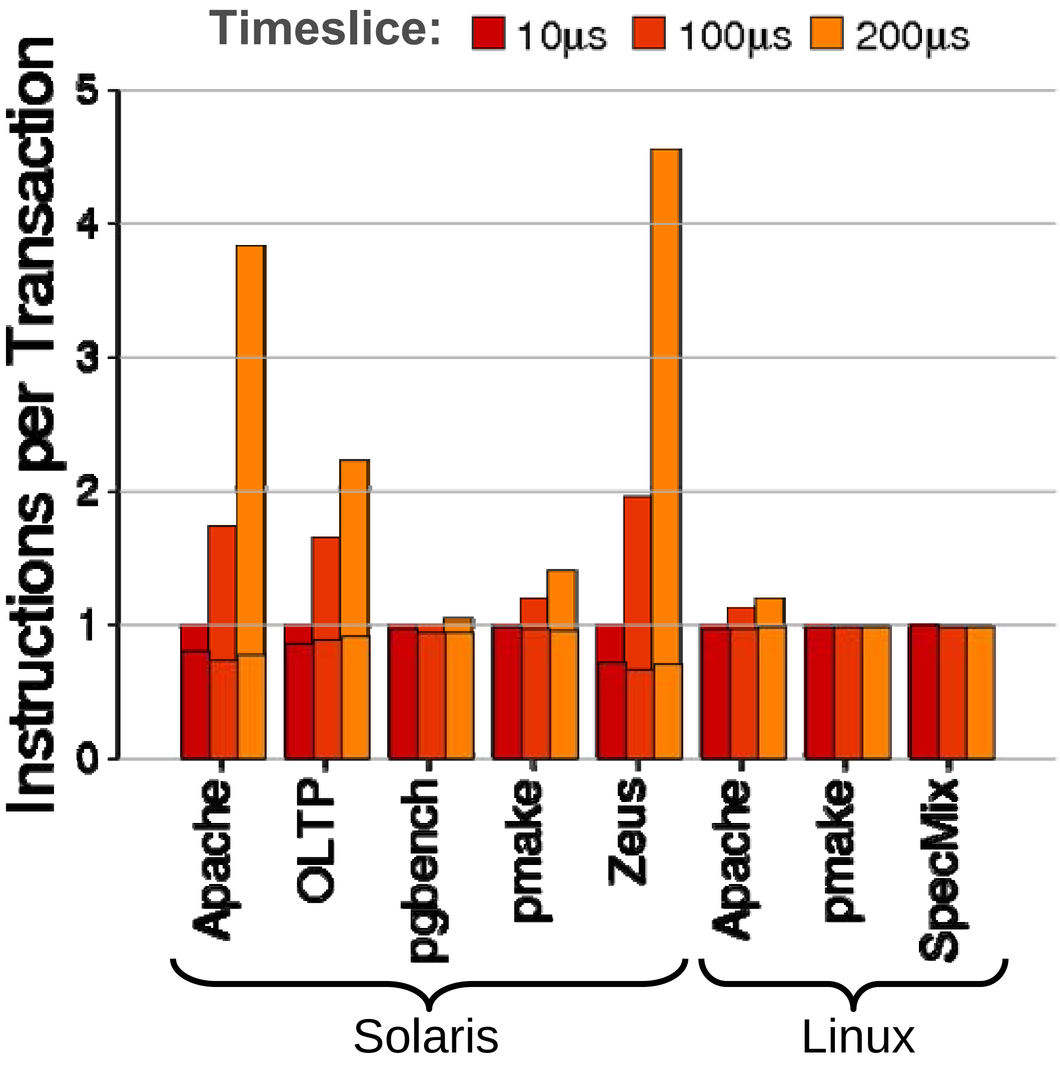
Spin Detection Buffer (SDB)

Implement heuristic with two 8-entry CAMs

- *After* ST (+LD in user) commits: Search CAM; insert if unique
- Check for less than 8 entries every 1k instr.
- *Off* critical path, has low B/W requirements
- Low activity



Spin Detection Buffer: Effectiveness



No spin detection

Using SDB

- No undetected spins (that we are aware)
- Very few false positives

Spin Detection Buffer: Related work

"Safe-points,"

- [Uhlig, et al., VM '04]

Spin Detection Hardware (not cited in paper)

- Li, Lebeck & Sorin [IEEE Trans. Par. Dist. Sys., June '06]

But, overcommitting individual VM was *not* goal of other work

	Hardware	False Pos	User Spin	OS Mutex	Cross calls	Idle Loop
Safe Pts	None	None	✗	✓	✗	✗
Li, et al.	A Lot	None	✓	✓ / ✗	✗	✓ / ✗
SDB	Some	Few	✓	✓	✓	✓

Talk outline

Background & motivation

OS spin overhead

Hardware spin detection

Case study

- Server consolidation

Summary

Case study: Consolidated servers

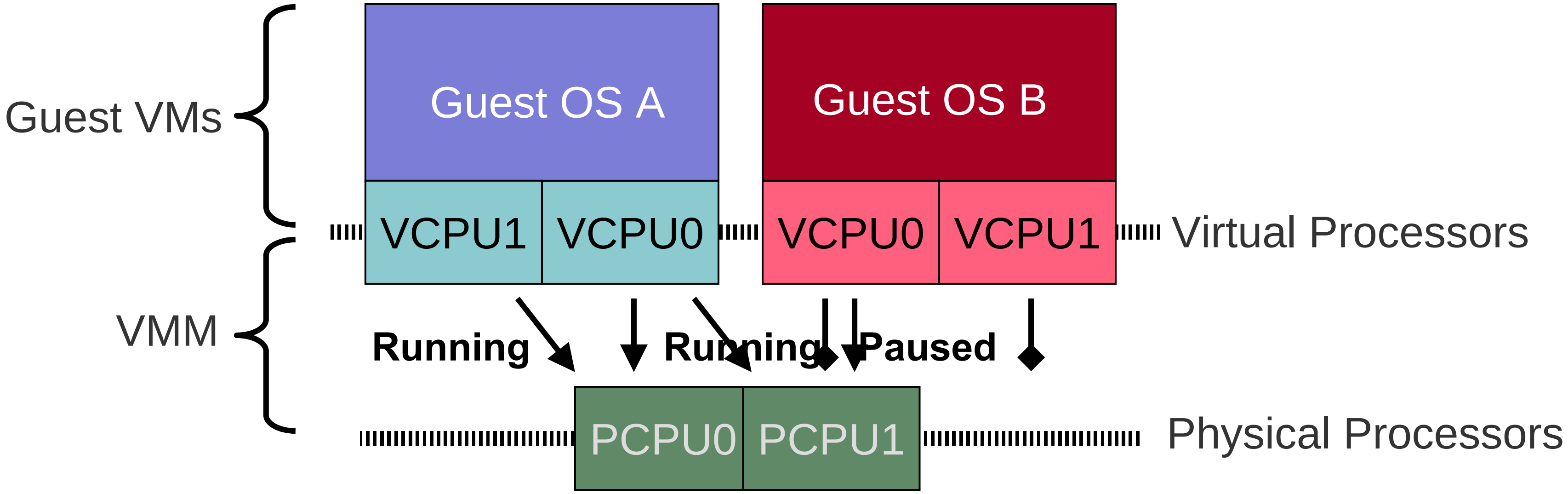
Run multiple services on one physical server

- Better utilize physical servers
- Centralize data center

Minimal changes to guest OSs and configs

- Pure virtualization (e.g., VMWare ESX Server)

Consolidated servers cont…



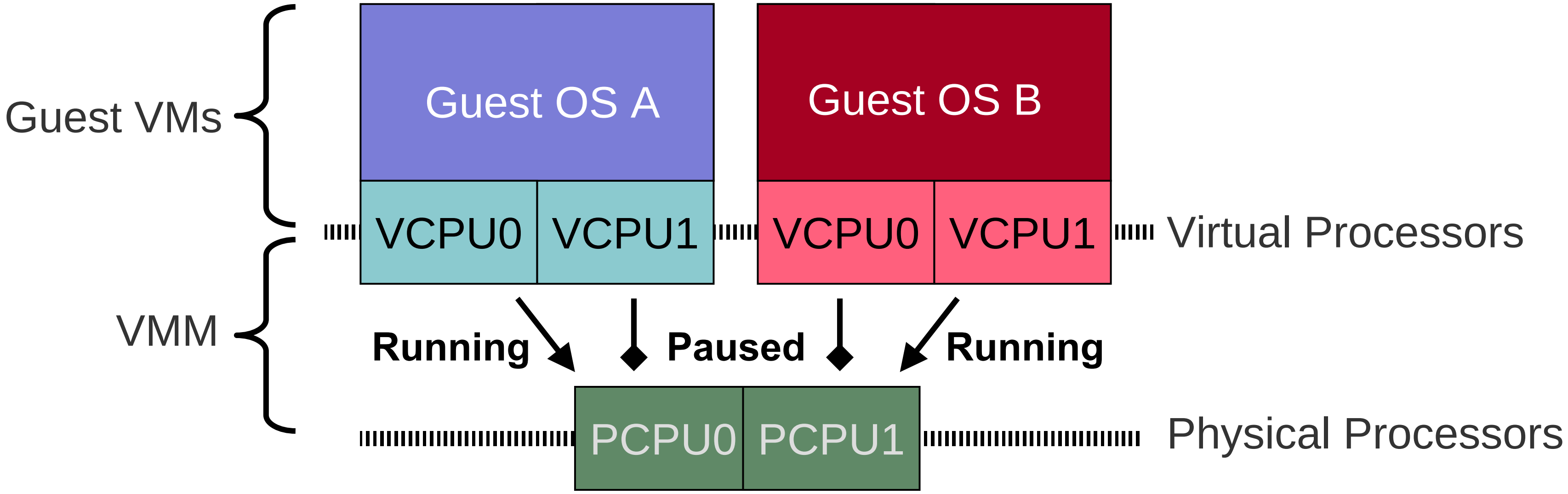
Partition PCPUs among guest OSs

- Good cache locality, performance isolation & response latency

Or, overcommit VMM and gang schedule

- Allows guest VMs to handle bursts in demand

Consolidated servers cont…



Use SDB for more flexibility for a variety of scenarios

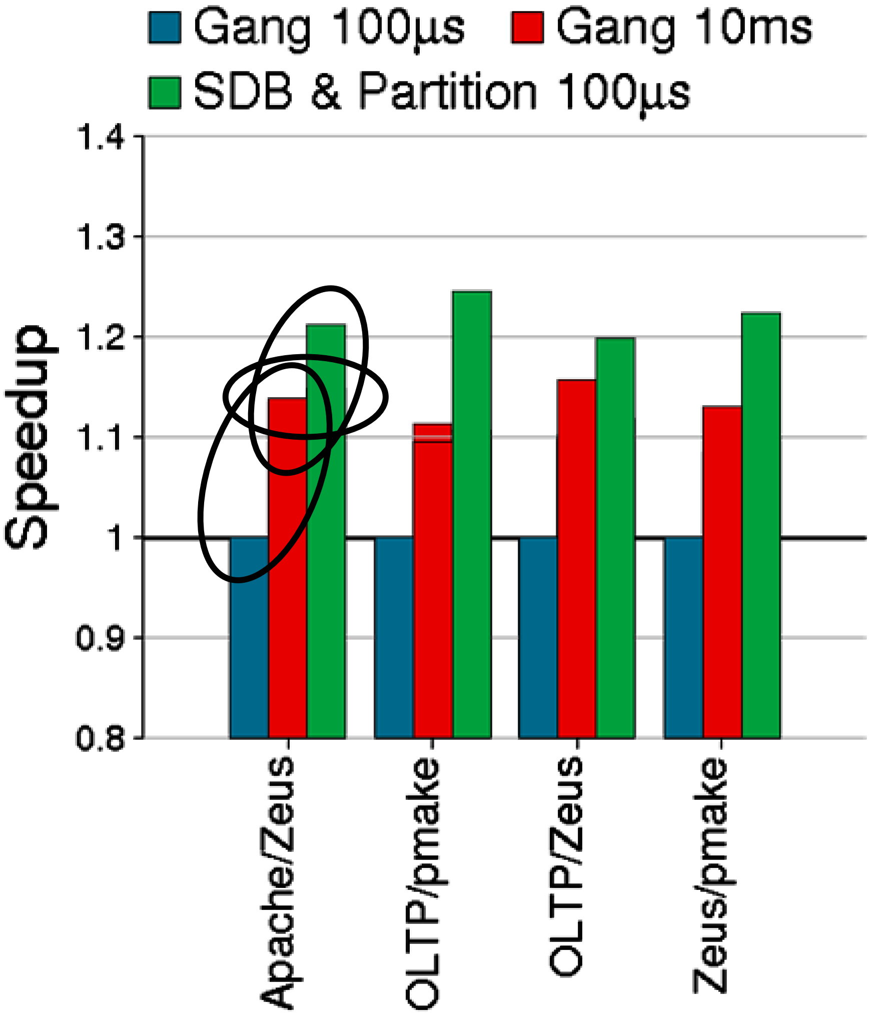
– **E.g. partition PCPUs among VMs**

Consolidated servers: Methodology

Two workloads consolidated into one checkpoint

- 16 VCPUs, 8 PCPUs
- Do not share any physical memory
- Share physical processor, TLB arrays, caches
- Idealized software VMM
  - No overhead from virtualizing memory, I/O

Consolidated servers: Locality



16 VCPUs 100% utilized

- 10ms has better locality than 100µs
- SDB allows throughput of 10ms & response latency of 100us

16 VCPUs ~50% utilized

- Expected case
- SDB avoids wasting resources on idle VCPUs

Summary

- **Many reasons to overcommit a guest VM**
  - But **unmodified OS** will spin
- **Spin Detection Buffer (SDB)**
  - **Hardware technique** to detect useless 'work'
  - Performs much better than other proposals
- **Consolidated server case study**
  - SDB allows **more flexibility** than gang scheduling
  - Can optimize for cache locality, performance isolation, etc.

Backup slides

THE UNIVERSITY
of
WISCONSIN
MADISON

Workloads

Multithreaded (Solaris & Linux)

- Apache (Solaris) – 15k trans
- Apache (Linux) – 30k trans
- pmake (Solaris & Linux) – 1.5B user instr
- Zeus (Solaris) – 7500 trans
- OLTP (Solaris) – 750 trans
- pgbench (Solaris) – 3000 trans
- Spec2000 Mix (Linux) – 24 at once, 500M cycles

Methodology

Simics full system simulation

- Commercial workloads on Solaris 9 & Linux 2.6.10
- Multiple trials, avg w/ 95% C.I. on runtime graphs

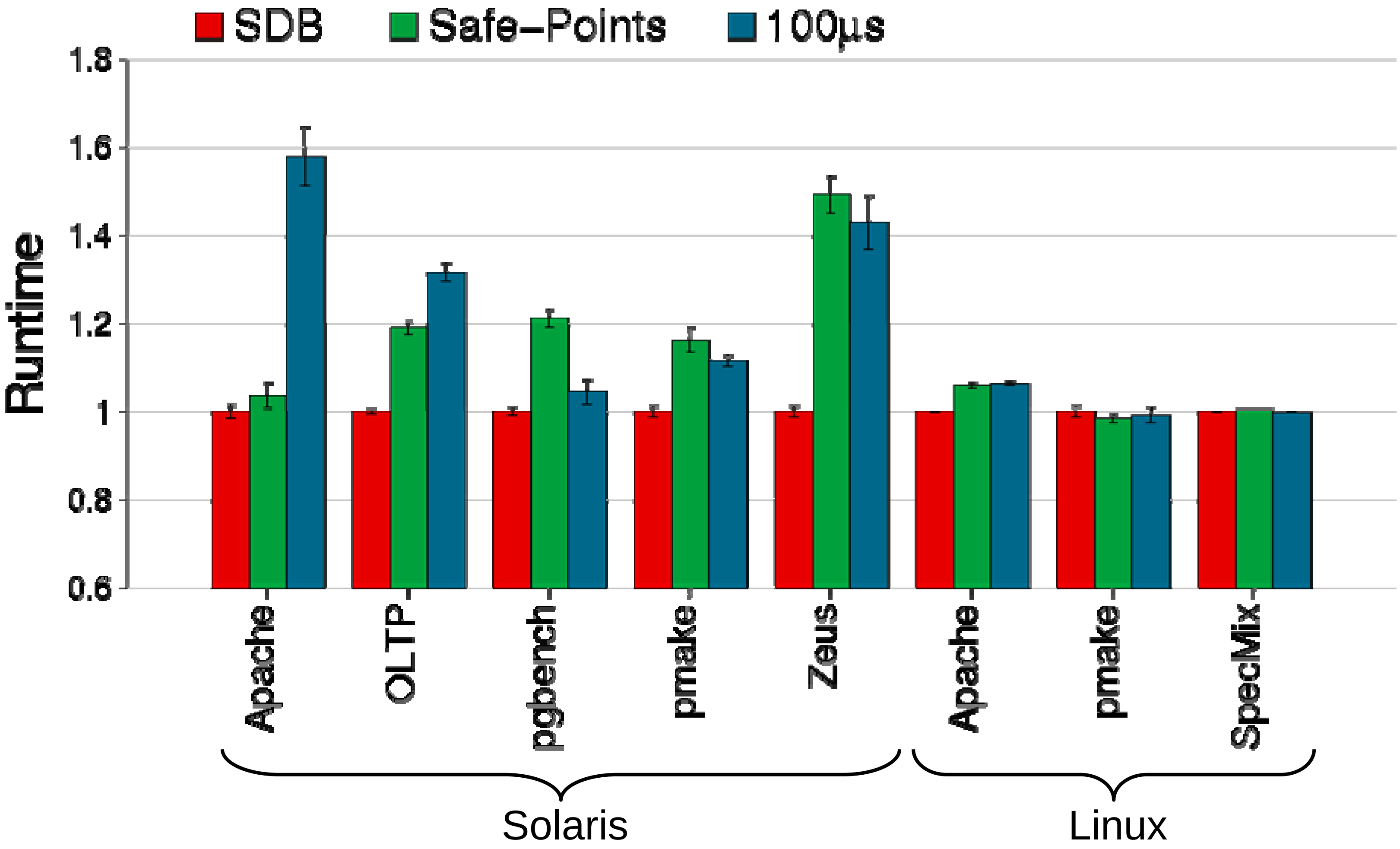
Each Core

- In-order, idealized, 1 GHz
- Private L2s: 4-way, 15 cycle, 512k

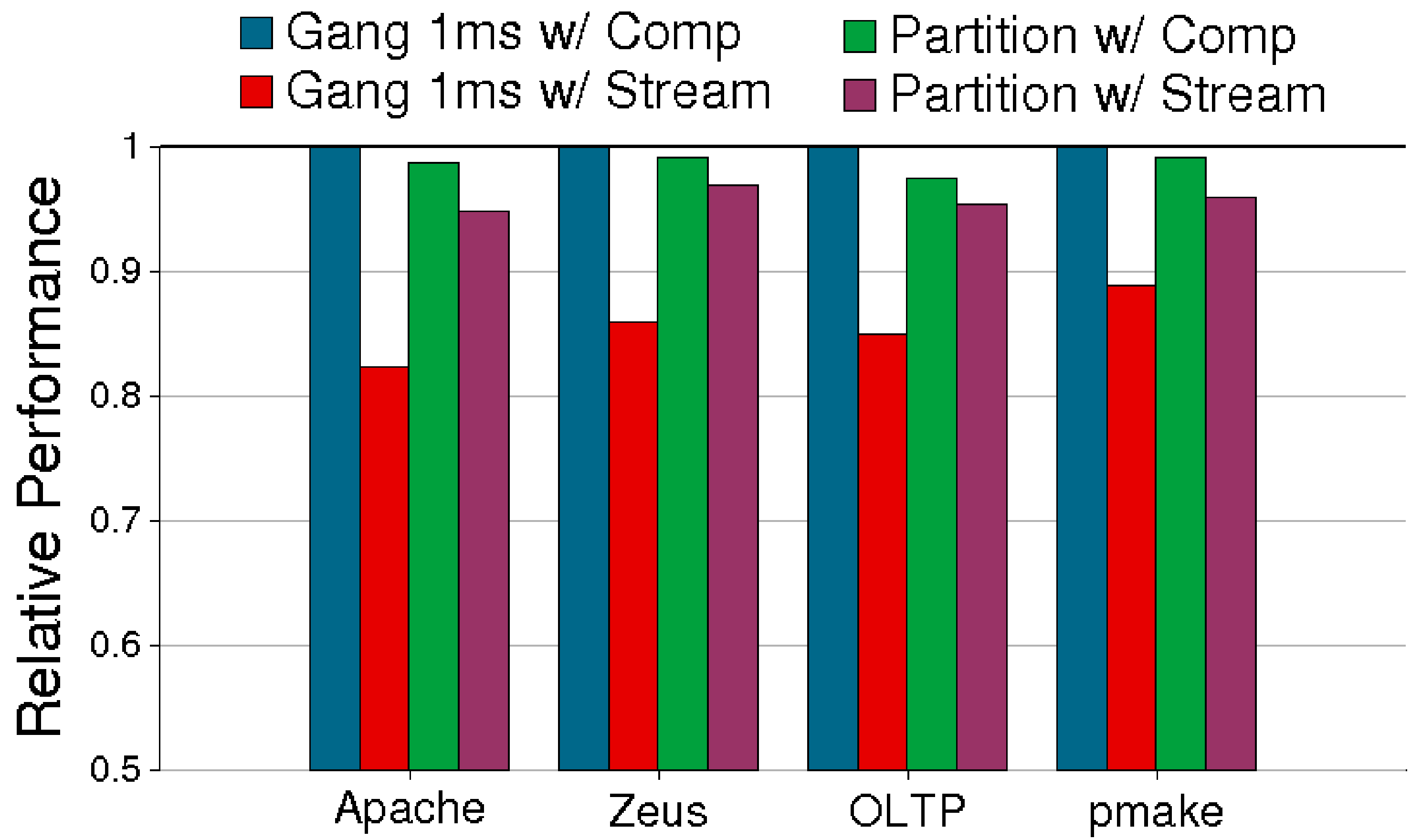
Shared

- Exclusive L3: 8MB, 16-way, 55 cyc. load-to-use, 8 banks

Safe points comparison



Consolidated servers: Isolation


Gang 1ms w/ Comp
Gang 1ms w/ Stream
Partition w/ Comp
Partition w/ Stream
Relative Performance
Apache
Zeus
OLTP
pmake