Karlsruhe Institute of Technology

A Light-Weight Virtual Machine Monitor for Blue Gene/P

Jan Stoess[§¶1] Udo Steinberg[‡1] Volkmar Uhlig[¶1]

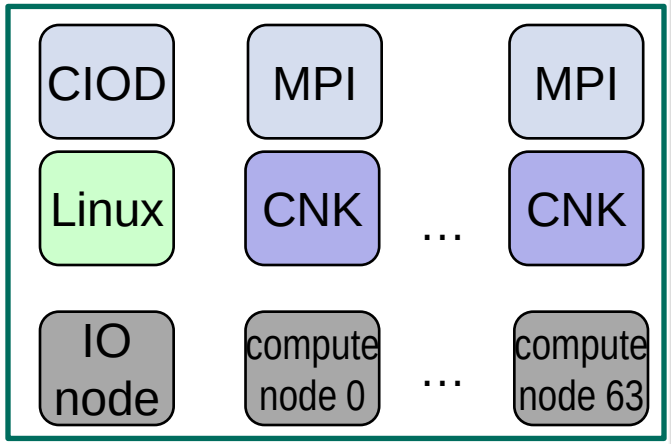
Jonathan Appavoo[†1] Amos Waterland[|1] Jens Kehne[§]

[§]Karlsruhe Institute of Technology [‡]Technische Universität Dresden

[¶]HStreaming LLC [|]Harvard School of Engineering and Applied Sciences [†]Boston University

BG/P Programming Model

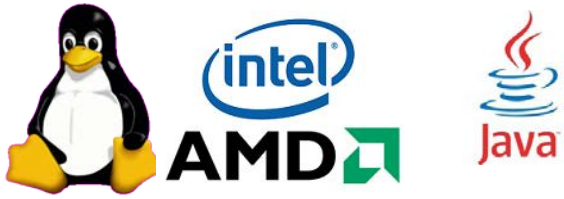
- Traditional BG/P supercomputer programming model:
  - Parallel programming run-time (MPI)
  - Compute-Node Kernel

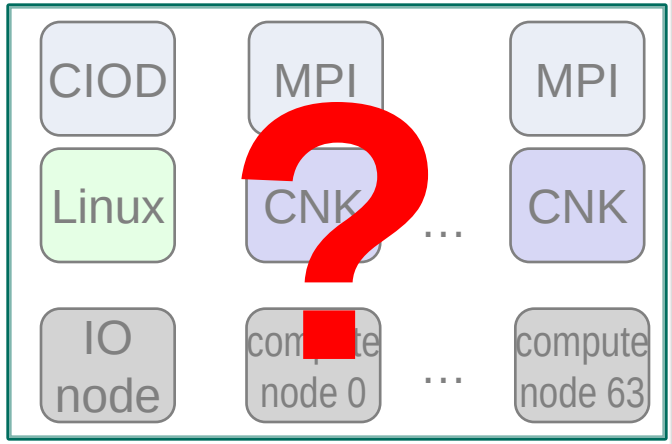


- CNK: OS for massive parallel applications
  - Light-weight kernel, “POSIX-ish”
  - Function-shipping to IO-nodes
  - Perfect choice for current HPC apps
    - MPI programming model
    - Low OS noise
    - Performance, scalability, customizability

Application Scale-Out

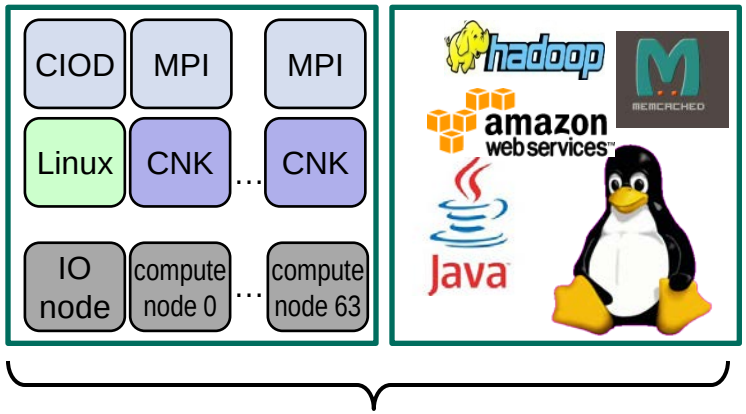
- Standard Server / Commercial workloads are scaling out
  - Big data (hadoop, stream processing, caching)
  - Clouds (ec2, vcloud)
  - Commodity OSes, runtimes, (HW)
    - Linux OS, Java, Ethernet
- CNK not truly compatible:
  - No full Linux/POSIX compatibility
  - No compatibility to standardized networking protocols

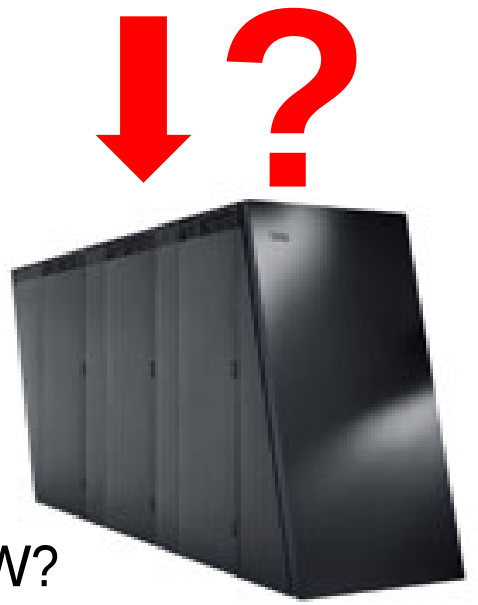




HPC readiness vs. Compatibility

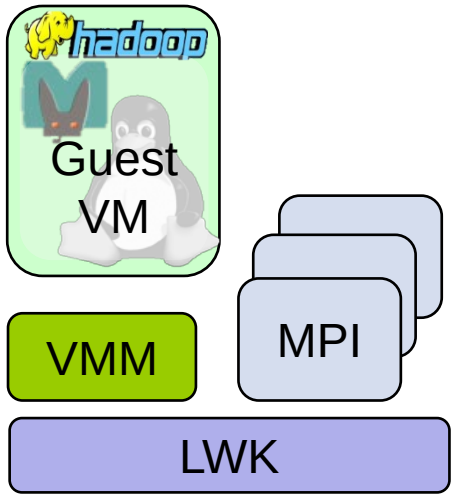
- Commodity OSes not designed for Supercomputers
  - OS footprint and complexity?
  - Network protocol overhead
  - Problem for standard scale-out software as well
- BG could run such workloads – in principle
  - “cores, memory, interconnect”
  - Reference HW for future data centers
- Can we have
  - ... the **HPC strength** of CNK **and**
  - ... the **compatibility** of commodity OS / NW?





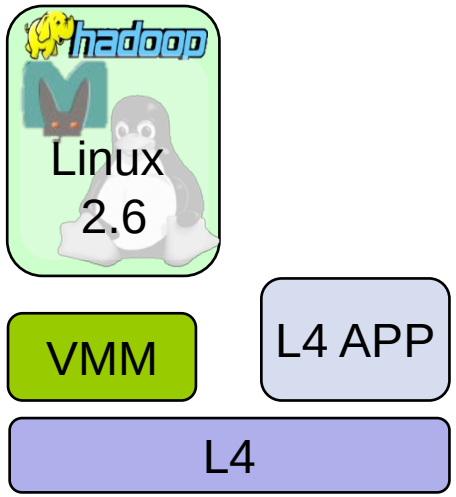
A Light-Weight VMM for Supercomputers

- Idea: use a light-weight kernel and a VMM
  - VMM gives HW-compatibility
    - Can run Linux in a VM
    - Can run Linux applications
    - Can communicate via (virtualized) Ethernet
  - Light-weight kernel preserves short path to HW
    - Run HPC apps “natively”
    - Direct access to HPC interconnects
    - Kernel small and customizable
    - Low kernel footprint
- Development path for converging platforms and workloads



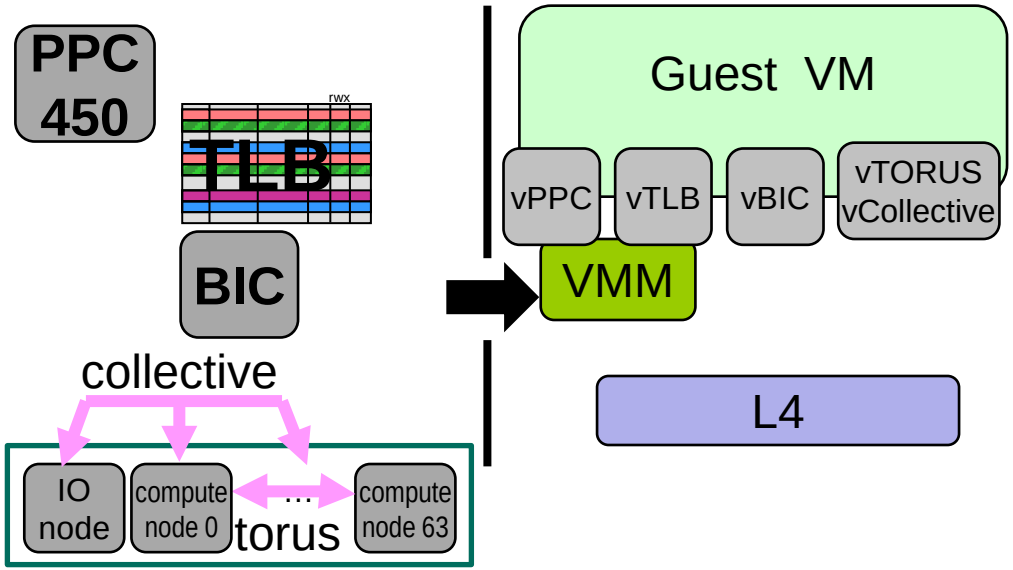
Prototype

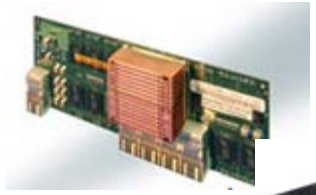
- L4 based prototype
  - Small, privileged micro-kernel
  - User-level VMM component
- Current focus: VMM layer *(this talk)*
  - Virtual BG cores, memory, interconnects
  - Support for Standard OSes
- Future work: Native HPC app support
  - L4 has native API
  - Leverage ex. research on
    - L4 OS frameworks
    - Native HPC app layers [kitten/palacios]



BG Overview and VMM agenda

- Compute Nodes
  - 4 PowerPC 450 cores
  - 2 GB physical memory
  - MMU/TLB
  - Interrupt controller
  - Torus and Collective
  - other HW
    (mailboxes, JTAG) not considered

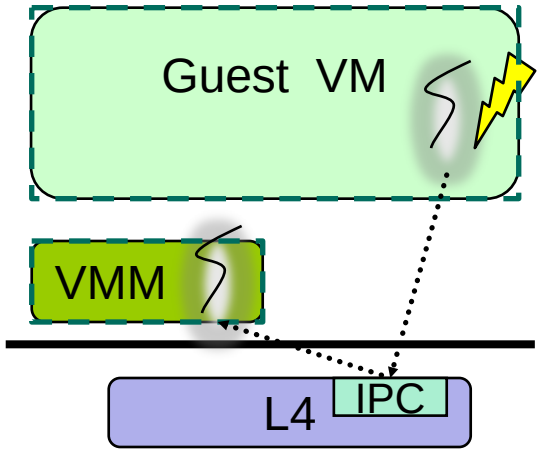




- IO nodes:
  - Not virtualized
  - Run special Linux for booting

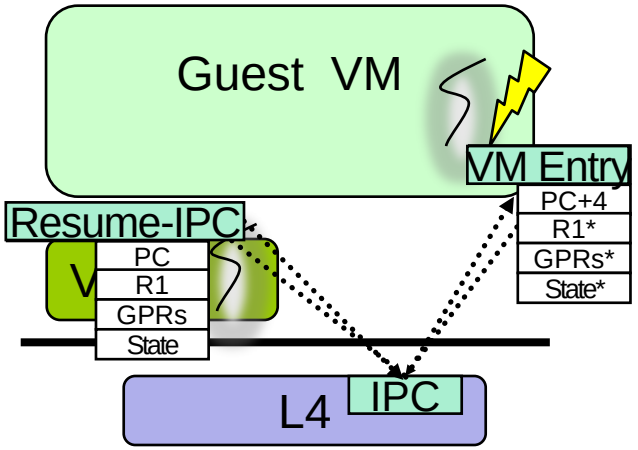
L4 and VMM architecture

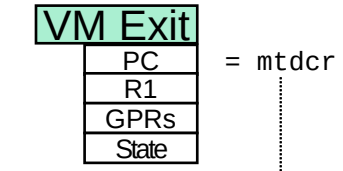
- L4 offers generic OS abstractions
  - Threads
  - Address spaces
  - Synchronous IPC
  - IPC-based exception / IRQ handling
- VMM is just a user-level program
  - Receives “VM exit” message from VM
  - Emulates it and replies with an update message
- L4 virtualization enhancements
  - Empty address spaces
  - Extended VM/thread state handling
  - Internal VM TLB handling



Virtual PowerPC processor

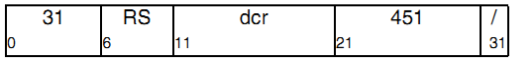
- VM runs at user mode, privileged PPC instructions trap
- L4 propagates traps to user-level VMM
  - kernel-synthesized IPC
  - VM/thread state included
- VMM receives trap IPC
  - Decodes message (faulting PC)
  - Emulates instruction (e.g. device IO)
  - Sends back a reply IPC
- Upon reception
  - Kernel installs new state
  - Resumes guest





Move To Device Control Register
XFX-form

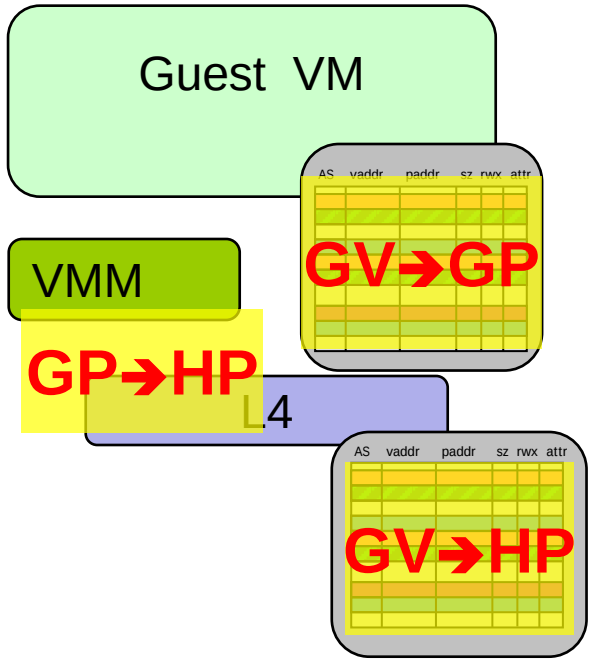
mtdcr DCRN,RS
[Category: Embedded.Device Control]



$DCRN \leftarrow dcr_{0:4} \,||\, dcr_{5:9}$
$DCR(DCRN) \leftarrow (RS)$

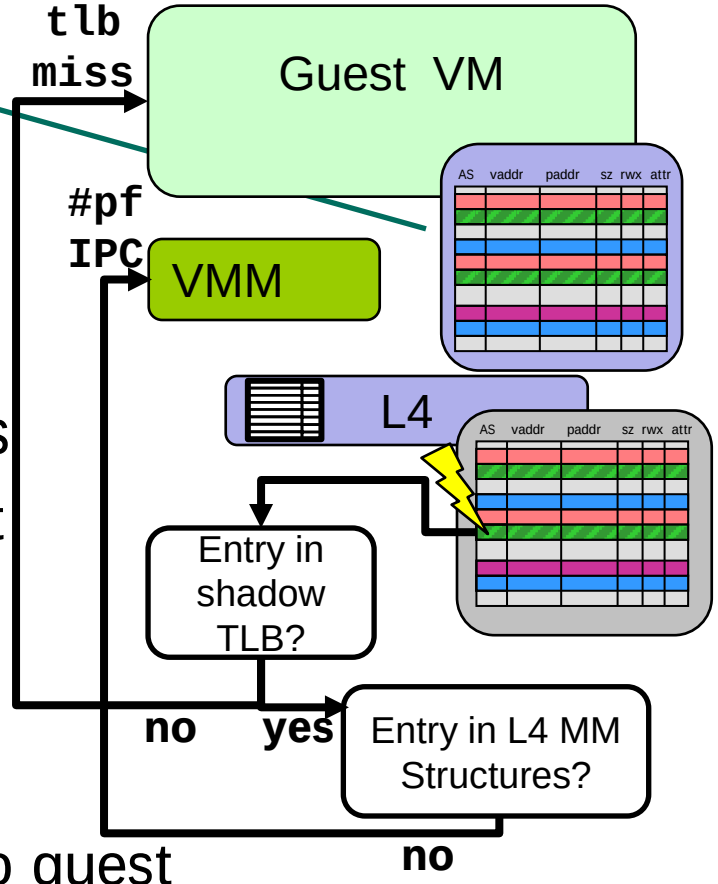
Virtual MMU/TLB

- PowerPC 450
  - 64-entry TLB
  - No HW-walked Page Tables
- Need to virtualize MMU translation
  - Two levels
  - Guest virtual to guest physical (guest managed)
  - Guest physical to host physical (L4/VMM managed)
  - Compressed into HW TLB



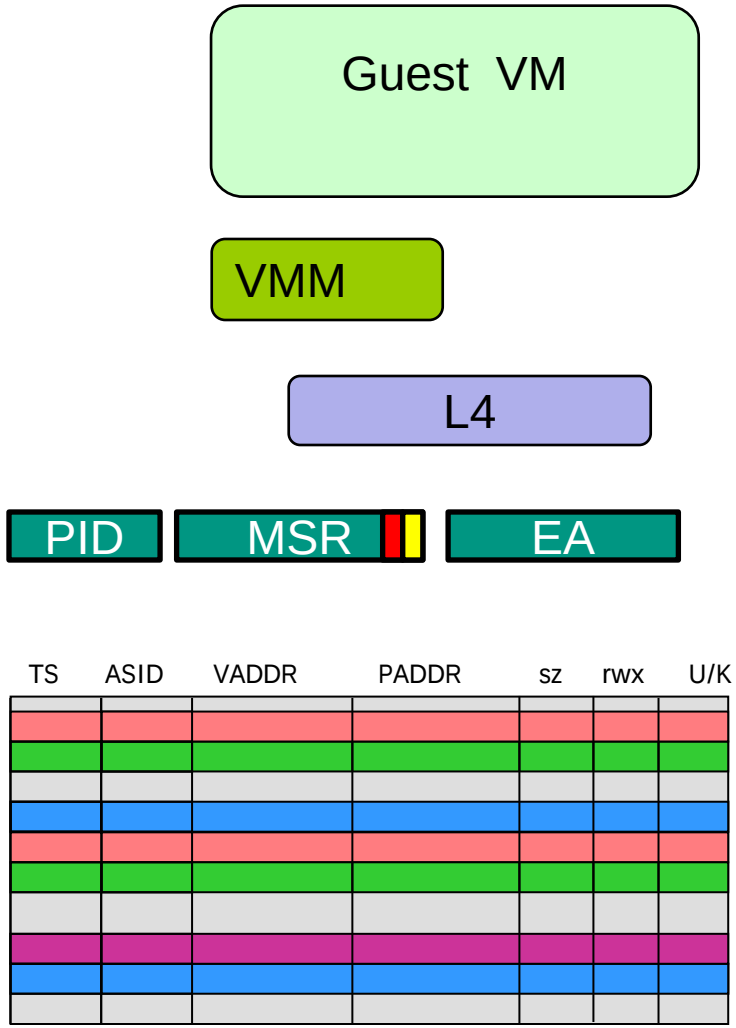
Virtual MMU/TLB

- L4 keeps a per-VM "shadow TLB"
  - Intercepts guest TLB access (`tlbwe, tlbre, …`)
  - Fills shadow TLB accordingly
  - Stores **GV➔GP** mappings
- L4 keeps per-AS memory mappings
  - Standard L4 memory management
  - Stores **GP➔HP** mappings
  - User-directed, VMM carries out
- TLB miss handling:
  - If guest virtual TLB miss, deliver to guest
  - If guest physical TLB miss, deliver to VMM



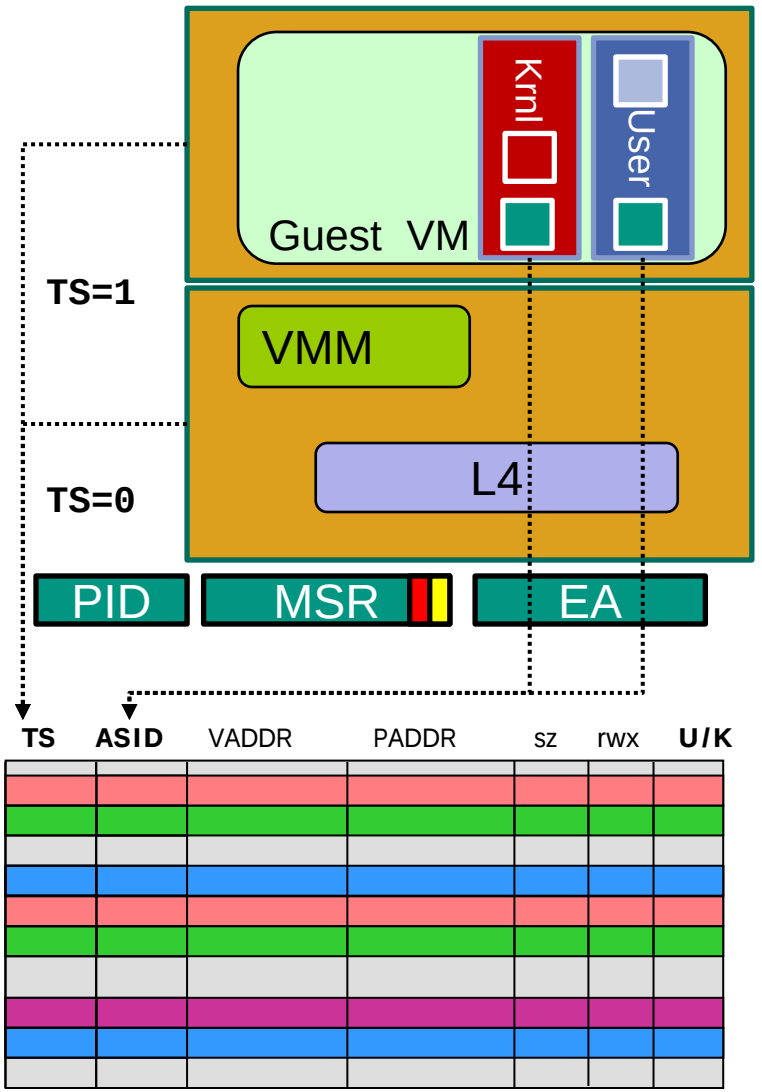
Virtual MMU/TLB Protection

- PowerPC TLB protection features
  - User/Kernel bits in TLBs
  - Address Spaces IDs (256)
  - Standard Linux behavior:
    - U/K bits for kernel separation
    - ASIDs for process separation
- Requirements:
  - Must virtualize protection
  - Guest code runs at user-level
  - Must support shared mappings
  - Compressed, as for translation
  - Minimize #TLB flushes



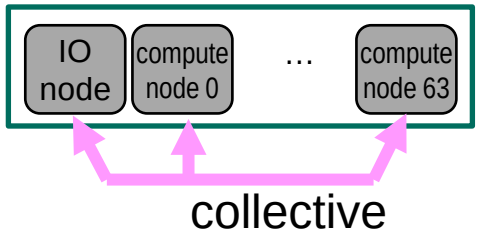
Virtual MMU/TLB Protection

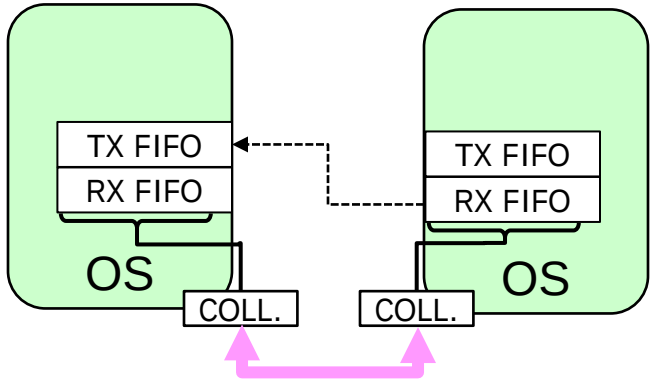
- Virtualized
  - L4/VMM **(TS=0)**
    - Use U/K bits and ASIDs
  - VM **(TS=1)**
    - All user-level (no U/K)
    - ASID=1: Guest Kernel
    - ASID=2: Guest User
    - ASID=0: Shared Mappings
- Analysis
  - No TLB flush on guest syscall
  - No TLB flush on VM exit
  - only on guest process or world switches



Virtual Collective Interconnect

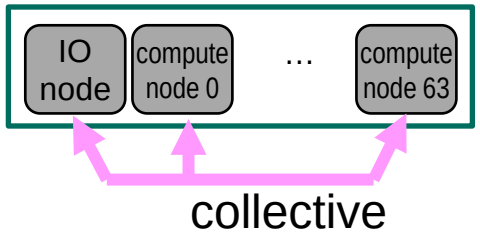
- Collective:
  - Tree Network, 7.8Gbit/s, <6µs latency
  - Packet-based, two virtual channels
    - Packet header
    - 16 * 128-bit FPU words payload
    - RX/TX FIFOs

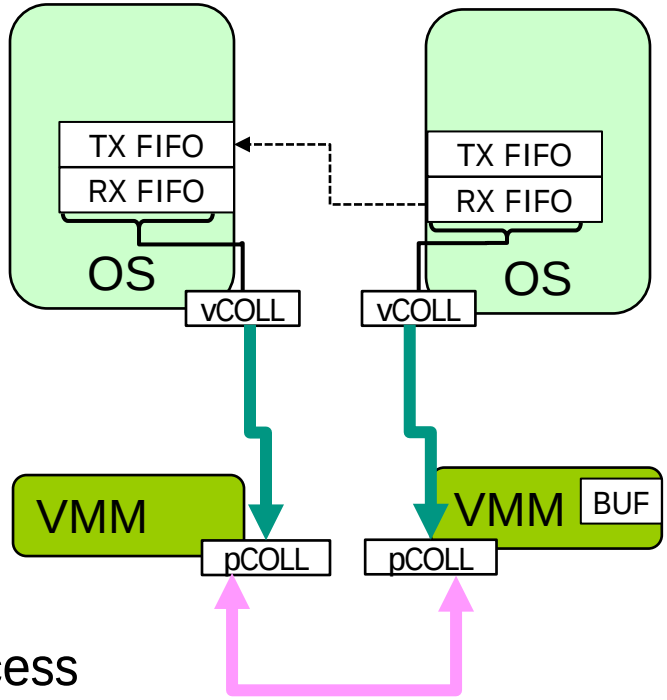




Virtual Collective Interconnect

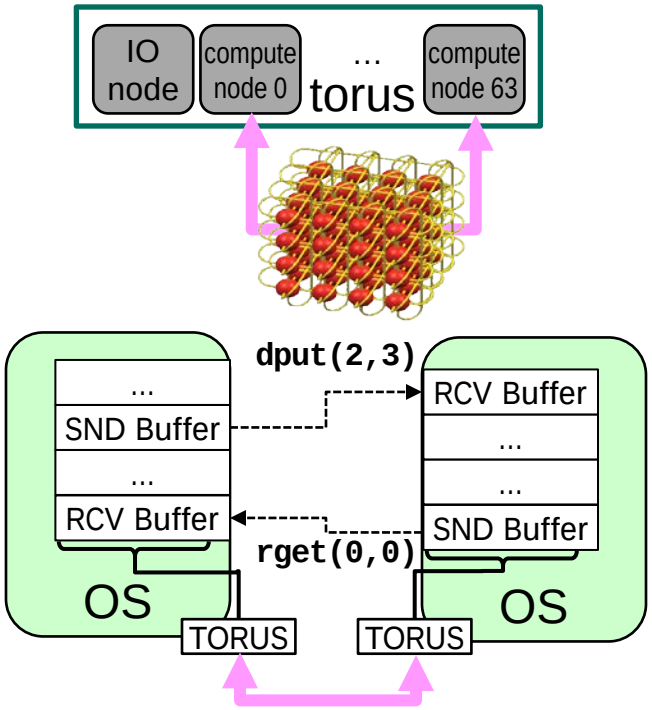
- Collective:
  - Tree Network, 7.8 Gbit/s, < 6 µs latency
  - Packet-based, two virtual channels
    - Packet header
    - 16 * 128-bit FPU words payload
    - RX/TX FIFOs
  - Virtualized collective:
    - TX:
      - Trap guest channel accesses
      - Issue on physical collective link
    - RX:
      - Copy GPR/FPU into private buffer
      - Notify guest, then trap vCOLL access





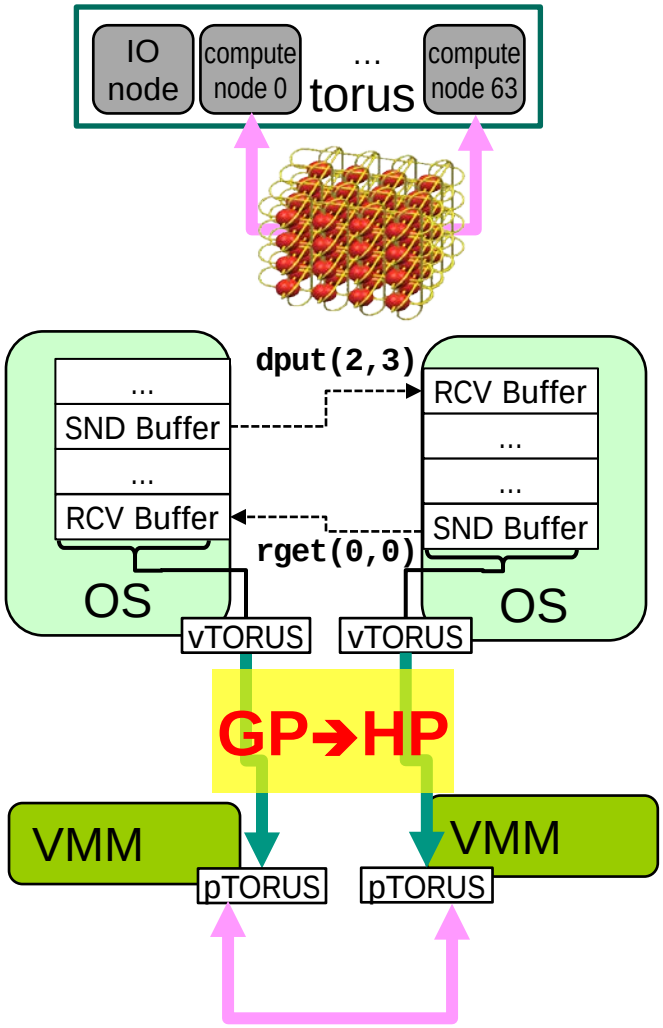
Virtual Torus Interconnect

- Torus:
  - 3D network, 40.8 Gbit/s, 5 µs latency
  - Packet-based, 4 RX/TX groups
  - (Buffer-based) and rDMA
  - rDMA:
    - direct access by (user) software
    - Memory descriptors
    - put/get interface
      (direct-put, remote-get)



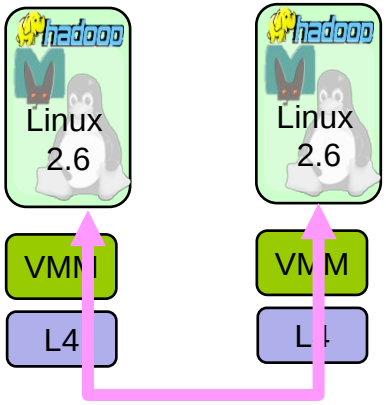
Virtual Torus Interconnect

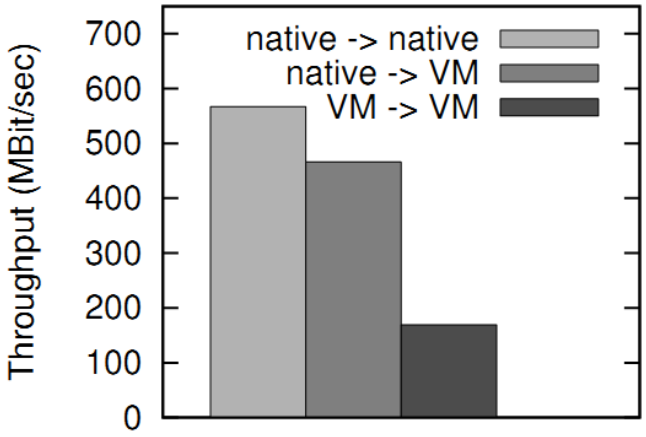
- Torus:
  - 3D network, 40.8 Gbit/s, 5 µs latency
  - Packet-based, 4 RX/TX groups
  - (Buffer-based) and rDMA
  - rDMA:
    - direct access by (user) software
    - Memory descriptors
    - put/get interface
      (direct-put, remote-get)
  - Virtualized torus model:
    - Trap guest descriptor accesses
    - Translate guest to host physical
    - Then issue on HW torus



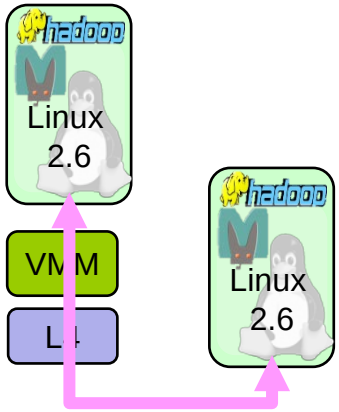
Status & Initial Evaluation

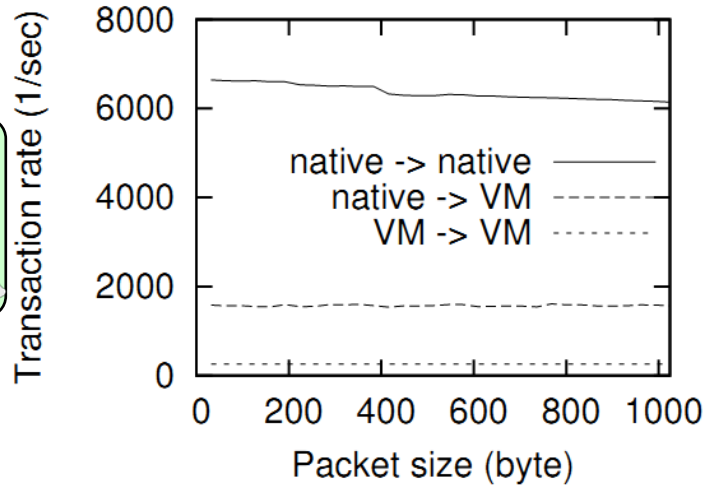
- Functionally complete:
  - Virtual PPC core, MMU
  - Virtual torus, tree
  - UP Linux 2.6 guests
  - Virtualized Ethernet (within guest)





- Initial benchmarks
  - Ethernet performance (mapped onto torus)
  - Collective much worse still testing/setup problems





Conclusion

- Standard server / commercial workloads are scaling out
  - Current BG/P programming model makes transition hard
  - Perfect choice for traditional HPC apps
  - Lacks compatibility to standard OSes (Linux) or network protocols (Ethernet)
- Idea: Use a light-weight kernel and a VMM
  - VMM for HW-compatibility, LWK for low footprint
  - Development path for converging platforms and workloads
  - L4-based VMM prototype functionally complete
    performance from acceptable (torus) to under-optimized (collective)
- Things to explore:
  - Native application support
    (MPICH2/L4 and Memcache/L4 for BG/P in preparation)
  - Performance, Isolation