Implementation of Direct Segments on a RISC-V Processor

Nikhita Kunati, Michael M. Swift
University of Wisconsin-Madison

Key Points

Past analysis shows

- TLB misses can spend 5%-50% of execution cycles on TLB misses.
- Rich features of Paged VM is not needed by most applications

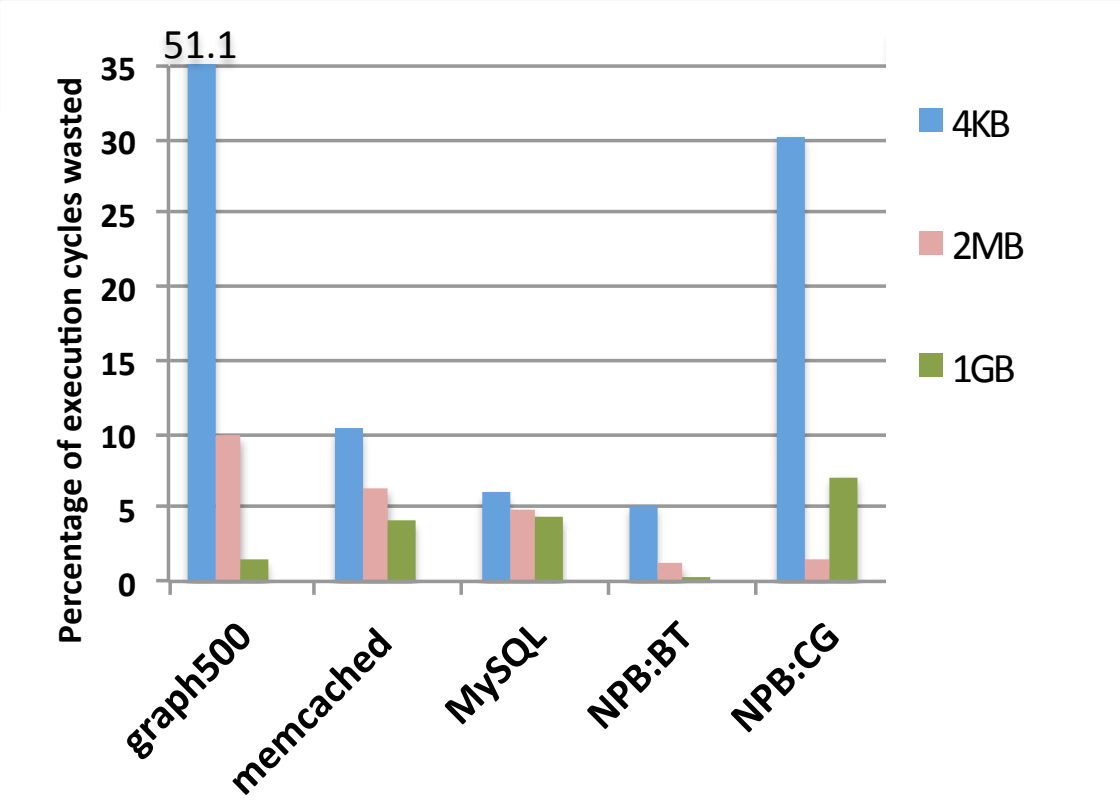
Direct Segments on a RISC-V Rocket Core

- Paged VM as usual where needed and Segmentation where possible
- Perform Direct Segment Lookup on a TLB Miss.

Software Support : RISC-V Linux Kernel

- Contiguous memory allocator to reserve and use a contiguous region of Physical memory
- Allocate Primary Regions (contiguous range of virtual addresses).

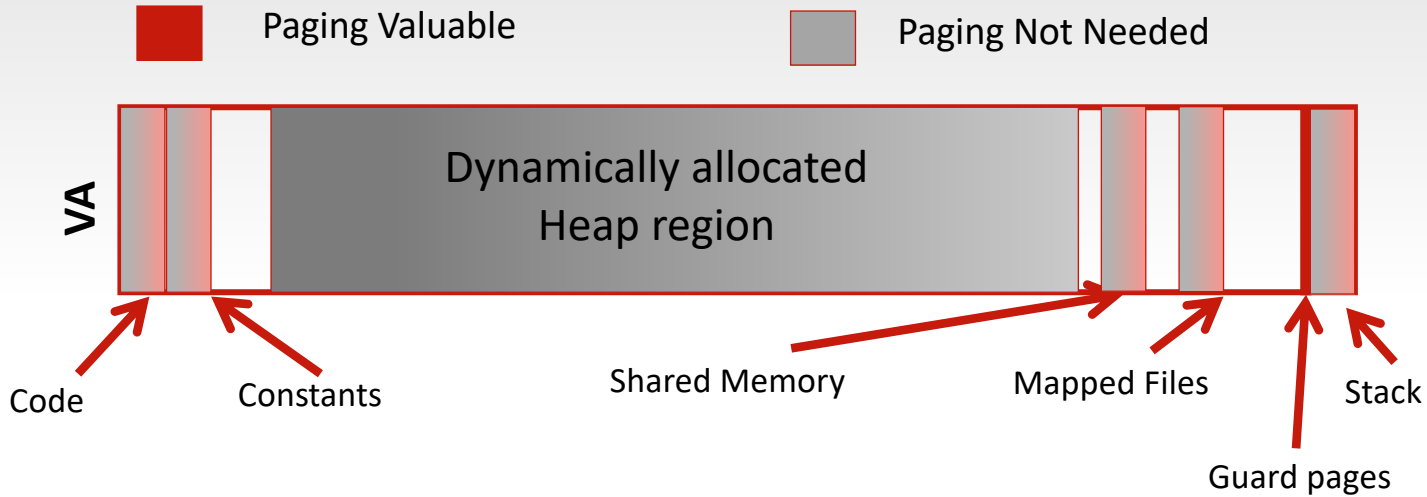
How Bad Is It ?


51.1
Percentage of execution cycles wasted
35
30
25
20
15
10
5
0
4KB
2MB
1GB
graph500
memcached
MySQL
NPB:BT
NPB:CG

Paged VM: Why is it needed ?

- Shared memory regions for Inter-Process-Communication
- Code regions protected by per-page R/W/E
- Copy on-write uses per-page R/W for lazy implementation of fork.
- Guard pages at the end of thread stacks.

Paged VM: Why is it needed ?

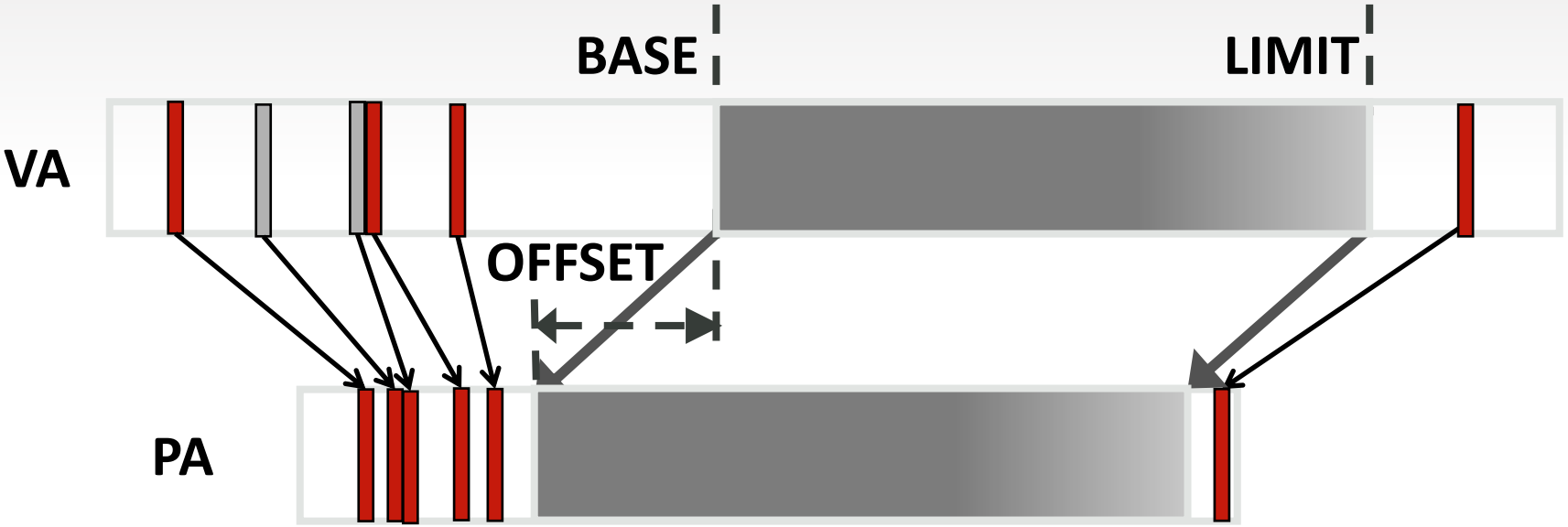


Paged VM not needed for MOST memory

Direct Segments

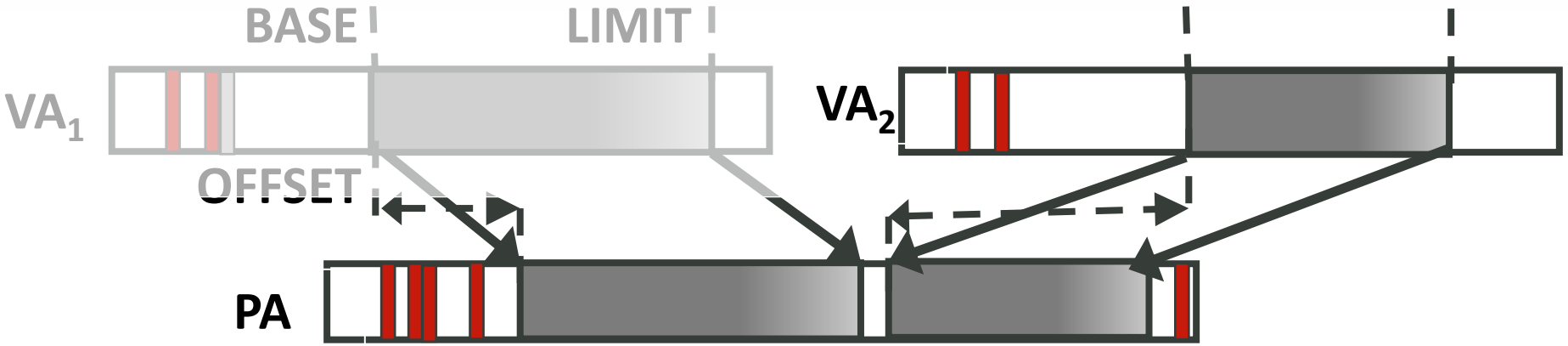
1 Conventional Paging

2 Direct Segment



Direct Segment Registers

BASE = Start VA of Direct Segment
LIMIT = End VA of Direct Segment
OFFSET = BASE – Start PA of Direct Segment

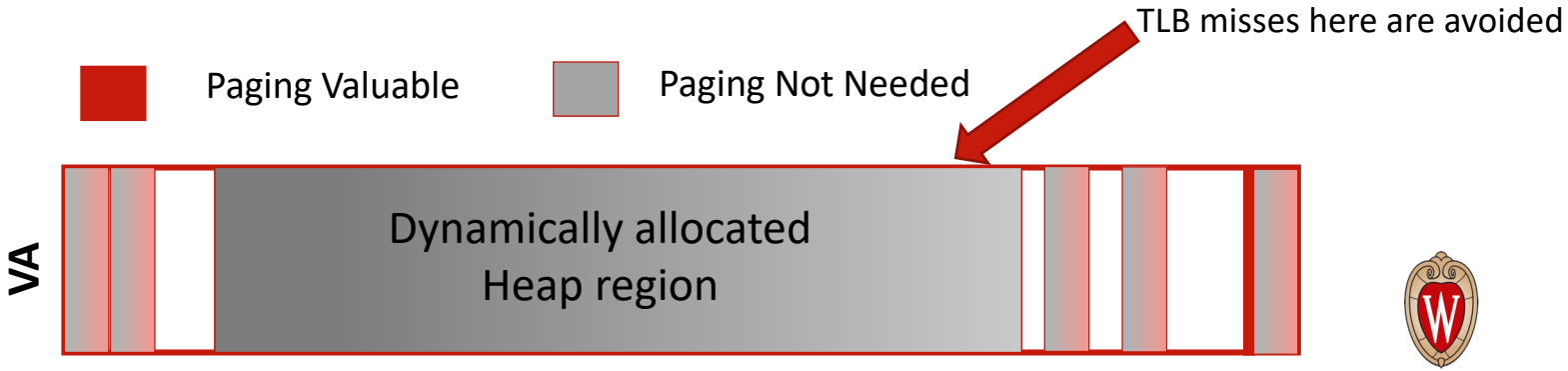


Prior Evaluation: BadgerTrap

- Tool to instrument x86-64 TLB misses.
- Trap all TLB misses by duping the system into believing that the PTE residing in memory is invalid.
- Insert translations into TLB, mark invalid in page table
- Once evicted from the TLB subsequent accesses causes a trap.

Previous Evaluation of Direct Segments

- In the handler -
  - Record whether the address falls in the primary region mapped using direct segment
  - Reload the PTE into the TLB
  - Again mark the PTE to invalid in memory



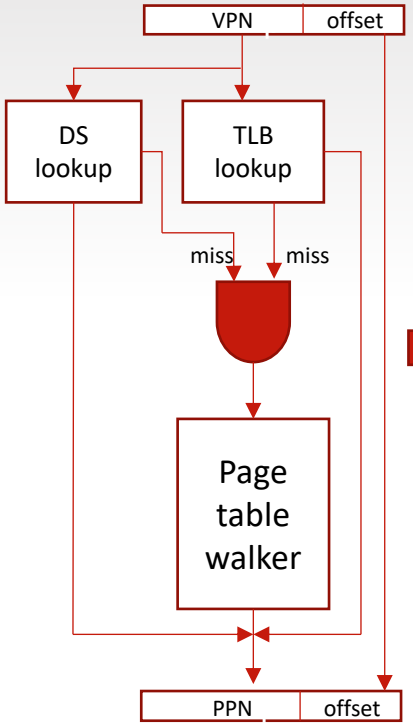
Shortcomings of the previous evaluation

- Emulation code checks the Direct Segment on a L2 TLB miss.
- Cannot accurately determine the cycles saved.
- Does not include the effects on pipeline timing from adding comparisons to the Base and Limit registers

Outline

- Design choices for Direct Segment Hardware.
- Hardware support in Rocket
- OS support
- Lessons learned
  RISC-V Ecosystem successes and challenges.

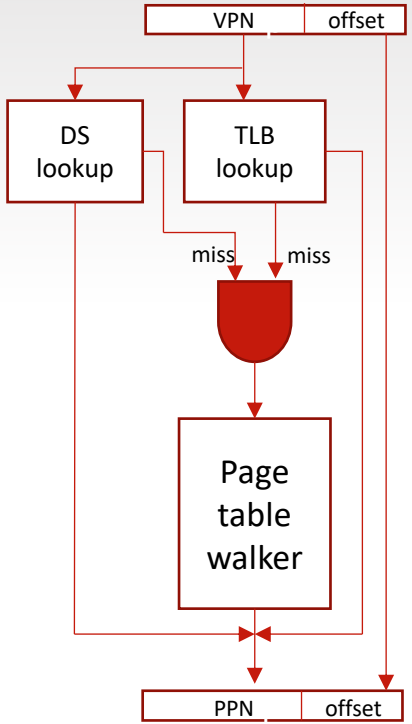
Design Choices



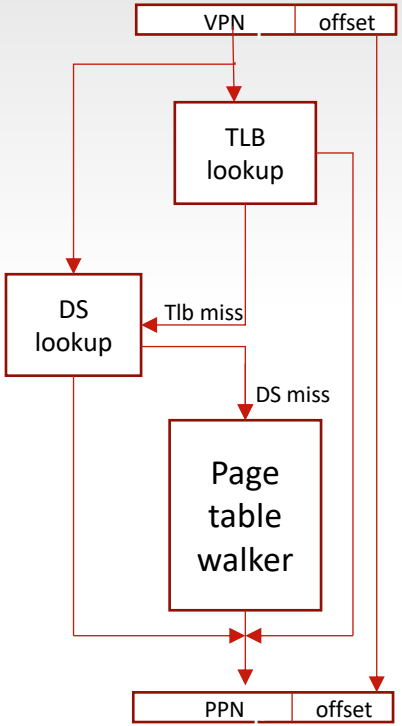
Original Design

Original Direct Segment paper proposes this

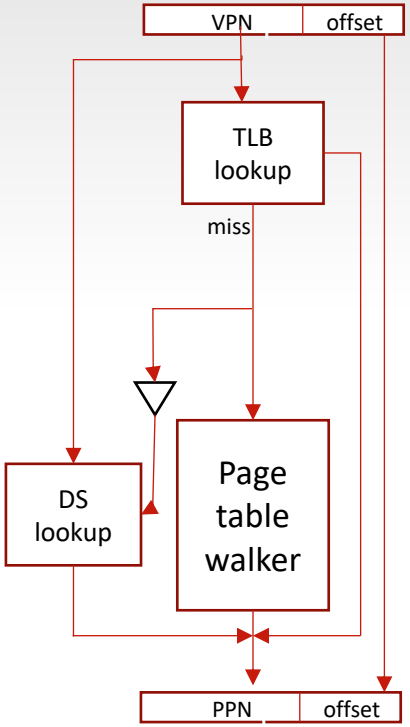
Design Choices



1. Original Design

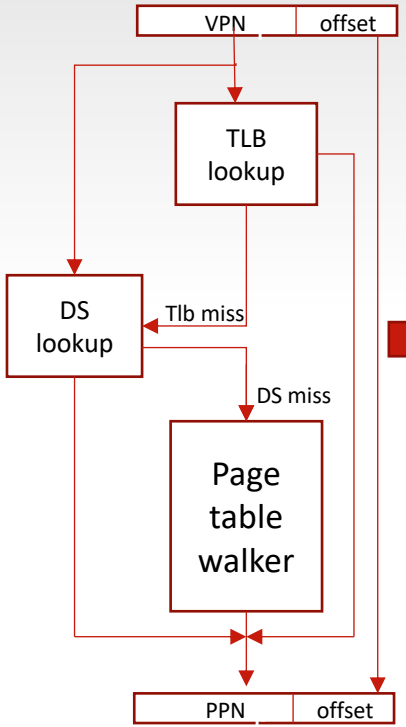


2. Before pagewalk



3. Parallel to pagewalk

Design Choices

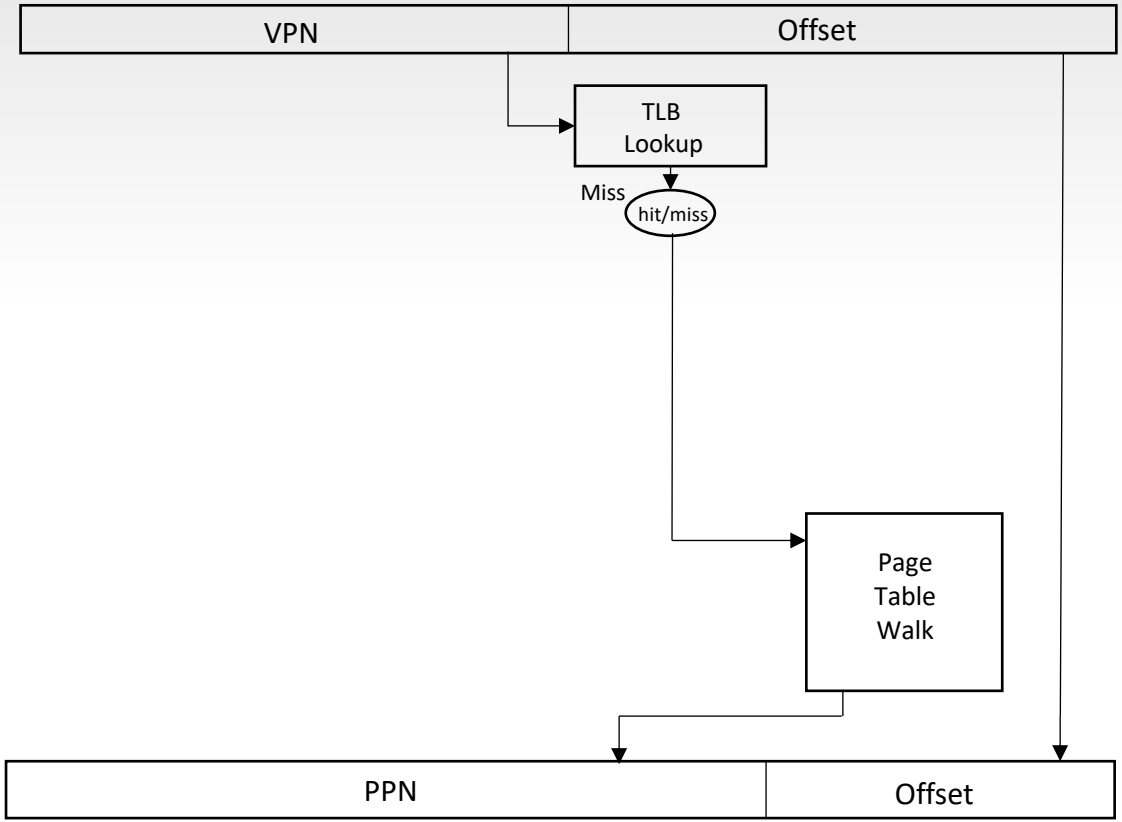


Our Implementation

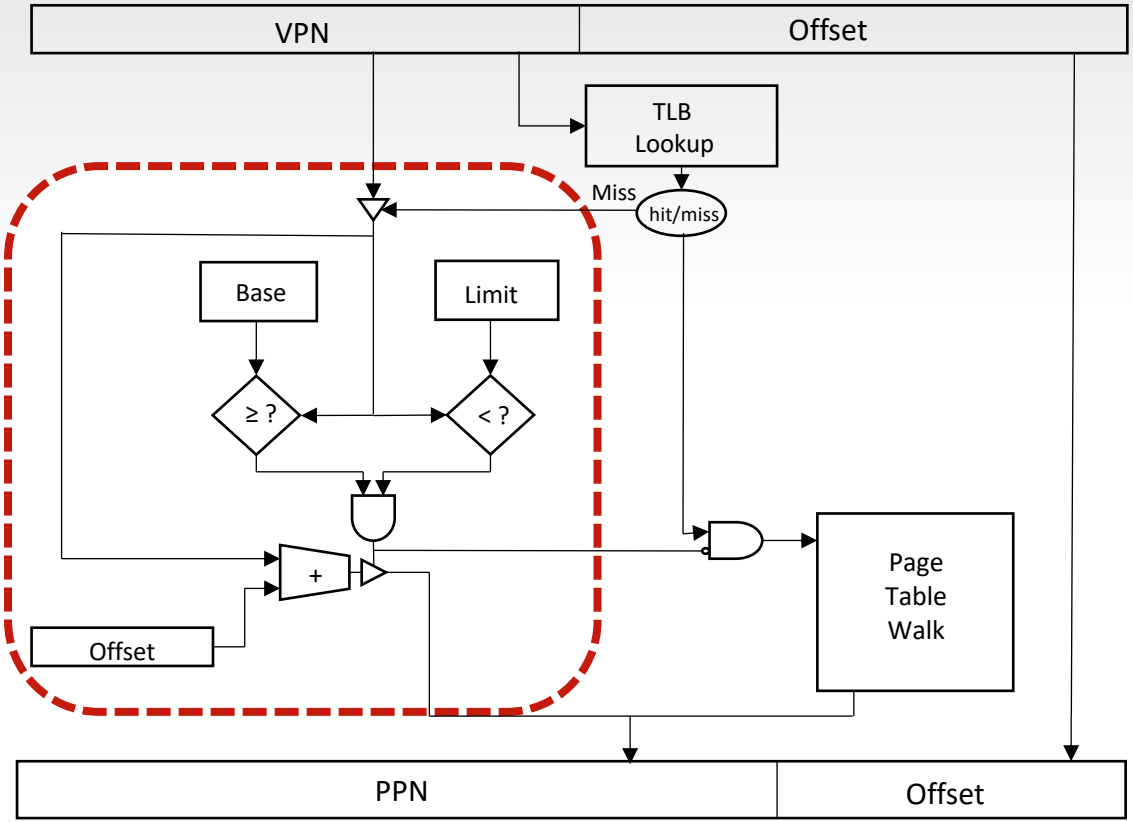
Outline

- Design choices for Direct Segment Hardware.
- Hardware support in Rocket
- OS support
- Lessons learned
  RISC-V Ecosystem successes and challenges.

Previous Address Translation in Rocket Core



Changed Address Translation in Rocket Core



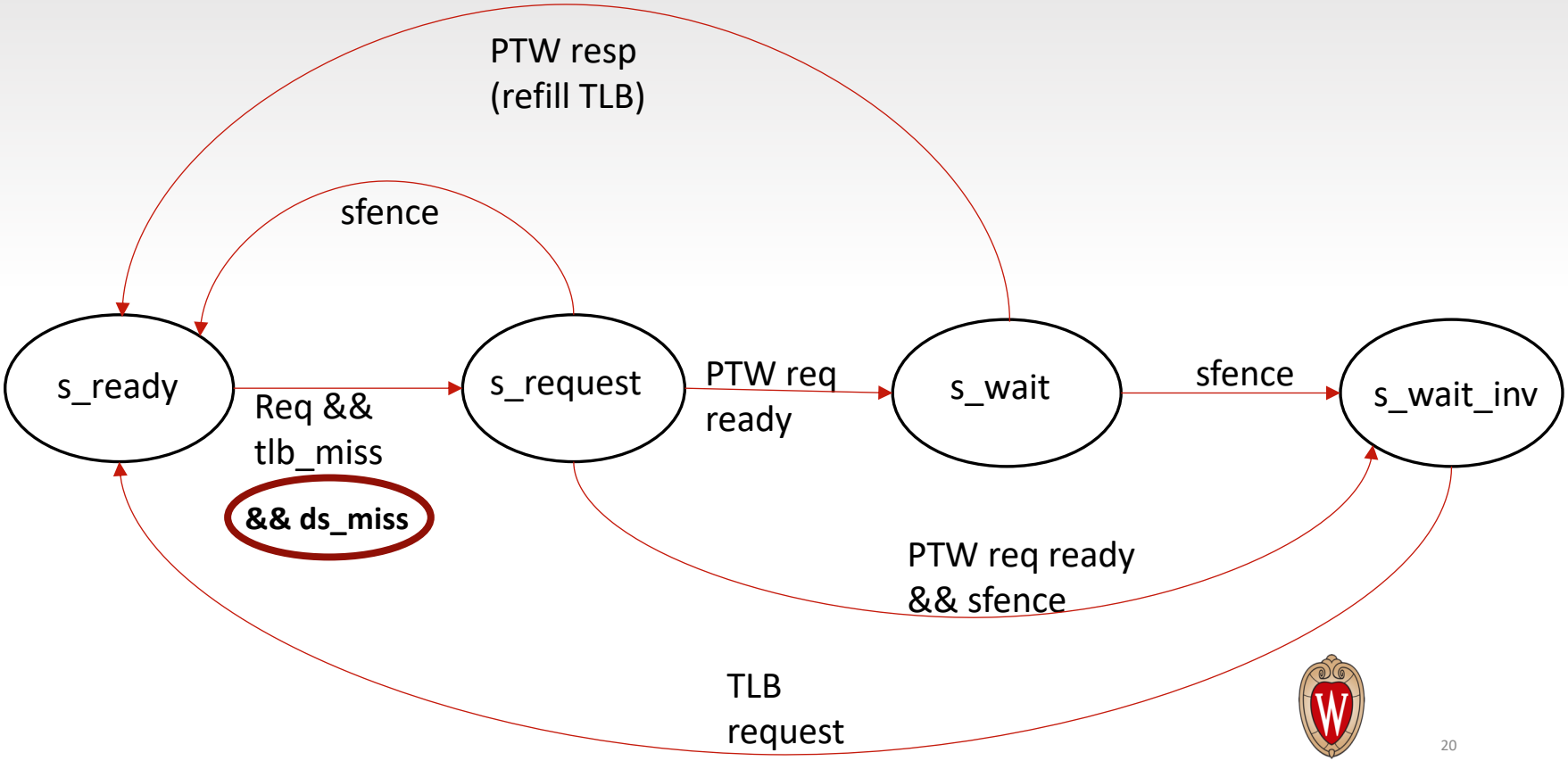
Hardware Support in Rocket Core

- Added CSR registers - Supervisor Direct Segment Base (SDSB), Supervisor Direct Segment Limit (SDSL), and Supervisor Direct Segment Offset (SDSO) to store the base, limit and offset.
- The least significant bit of SDSL is the enable bit, to enable/disable Direct Segments on a per-process basis.
- Direct Segment lookup performed on a TLB miss. This was chosen because of the ease of integrating the Direct Segment lookup into the existing TLB unit in Rocket.

Changes made to the TLB unit in Rocket

- If TLB miss and DS enabled then check if Virtual Address lies in between base and limit.
- We also check the protection bits in the Limit register.
- If Direct segment lookup successful compute Physical address by adding offset to Virtual Address.
- If Direct segment lookup unsuccessful set the ds_miss signal

Changes made to the TLB unit in Rocket



Outline

- Design choices for Direct Segment Hardware.
- Hardware support in Rocket
- OS support
- Lessons learned
  RISC-V Ecosystem successes and challenges.

OS Support – RISC-V Linux kernel

Create contiguous physical and virtual memory region

- Reserve physical memory at startup – Contiguous Memory allocator. dma_contiguous_reserve(phys_addr_t limit); Default is 16MB
- Create Primary region(contiguous range of virtual address) on encountering a primary process
- Allocate the reserved CMA region *dma_alloc_from_contiguous(struct device *dev, int count, unsigned int align);

OS Support – RISC-V Linux kernel

Setup Direct Segment registers

- BASE = Start VA of Direct Segment
- LIMIT = End VA of Direct Segment
- OFFSET = BASE – Start PA of Direct Segment
- Save and restore register values as part of process metadata on context-switch

Design Methodology

Spike RISC-V ISA Simulator

- Prototype of Direct Segments modified the walk() function.
- Tested with custom RISC-V assembly tests that set up primary regions.

RISC-V ISA Qemu

- Implement Direct Segments by modifying the get_physical_address() function.
- Chose Qemu because of the ease of testing RISC-V Linux Kernel changes.

Design Methodology

Direct segment logic and RISC-V linux kernel changes were tested on Spike and Qemu first because of the challenges faced with Verilator.

Challenges with Verilator

- Very slow booting the linux kernel takes ~ 1 day.
- Lack of useful debug prints in Verilator.

Lessons Learned

RISC-V Ecosystem Successes

- Well defined instruction-set
- Ease of configuring Rocket
- Plenty of Simulators
- RISC-V assembly test suite.

RISC-V Ecosystem Challenges

- The rapid pace of development within the RISC-V ecosystem
- Documentation across RISC-V projects either insufficient or missing.

RISC-V Linux Kernel Challenges

- Only basic support in RISCV Linux
- kernel was constantly under development

RISC-V Ecosystem Successes

- Well defined instruction-set with ease of adding new registers and instructions.
- Ease of configuring Rocket(Soc Generator).

RISC-V Ecosystem Successes

- Plenty of Simulators –

  Spike, RISC-V Qemu, Verilator.

- Comprehensive RISC-V assembly test suite.

RISC-V Ecosystem Challenges

The rapid pace of development within the RISC-V ecosystem posed a challenge to successfully implement and build Direct Segment hardware.

RISC-V Ecosystem Challenges

- Lack of comments explaining the flow in a particular unit and across multiple units in Rocket.
- Documentation across RISC-V projects either insufficient or missing.

```
io.resp.ma.ld := (ma_ld_array & hits).orR
io.resp.ma.st := (ma_st_array & hits).orR
io.resp.ma.inst := false // this is up to the pipeline to figure out
io.resp.cacheable := (c_array & hits).orR
io.resp.prefetchable := (prefetchable_array & hits).orR && edge.manager.manage
io.resp.miss := do_refill || tlb_miss || multipleHits
io.resp.paddr := Cat(ppn, io.req.bits.vaddr(pgIdxBits-1, 0))

io.ptw.req.valid := state === s_request
io.ptw.req.bits <> io.ptw.status
io.ptw.req.bits.addr := r_refill_tag

if (usingVM) {
  val sfence = io.req.valid && io.req.bits.sfence.valid
  when (io.req.fire() && tlb_miss) {
    state := s_request
    r_refill_tag := lookup_tag
    r_refill_waddr := repl_waddr
    r_req := io.req.bits
  }
  when (state === s_request) {
    when (sfence) { state := s_ready }
    when (io.ptw.req.ready) { state := Mux(sfence, s_wait_invalidate, s_wait)
  }
```

RISC-V Linux Kernel Challenges

- Basic support of RISC-V added to Linux kernel 4.15 sufficient to boot and not much else.
- Memory Hotplug support not present hence we had to find an alternative(CMA).

RISC-V Linux Kernel Challenges

- RISC-V Linux kernel was constantly under development.
- Obtaining the right version of the kernel which would boot on Qemu and Verilator was difficult.

Conclusion

- RISC-V is an excellent platform for virtual memory research.
- Well defined ISA, Open source implementation aid research
- Call for more engagement for virtual memory research.

Future Work

- Getting the Kernel changes working on Verilator.
- Measure the timing impact of adding the extra logic.
- Impact of alternative design choices.

Thank You
&
Questions?