RadixVM

Scalable address spaces for multithreaded applications

Austin T. Clements,
M. Frans Kaashoek,
Nickolai Zeldovich

Presented by Simon Pratt

February 12, 2016

Abstract

RadixVM solves 2 problems:

Abstract

RadixVM solves 2 problems:

1. Multithreaded address space scaling

Abstract

RadixVM solves 2 problems:

1. Multithreaded address space scaling
    1. (And reference counting)

Abstract

RadixVM solves 2 problems:

1. Multithreaded address space scaling
   1. (And reference counting)
2. Remote TLB shootdown

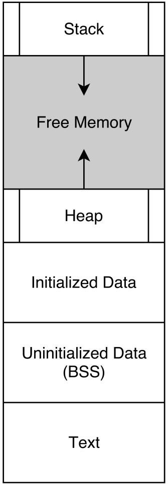
Abstract

RadixVM solves 2 problems:

1. Multithreaded address space scaling
    1. (And reference counting)
2. Remote TLB shootdown

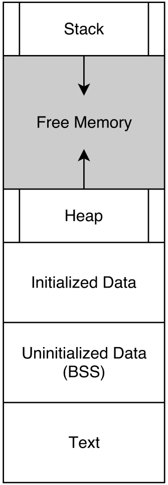
But first, some background...

Background: `malloc` and `mmap`



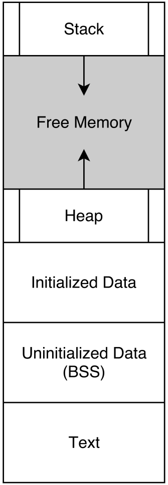
- `malloc` and `free`
  - User-level library function
  - Allocates/frees space in virtual memory
  - Often implemented using `mmap` and `munmap`

Background: `malloc` and `mmap`



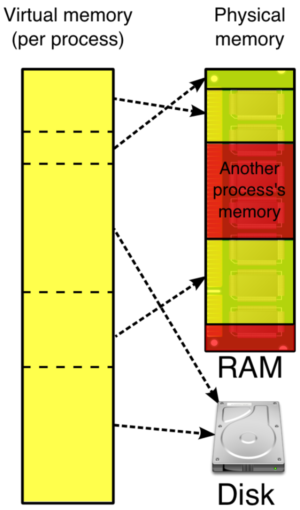
- `malloc` and `free`
  - User-level library function
  - Allocates/frees space in virtual memory
  - Often implemented using `mmap` and `munmap`
- `mmap` and `munmap`
  - System calls
  - Actually allocates/frees space in virtual memory

Background: `malloc` and `mmap`



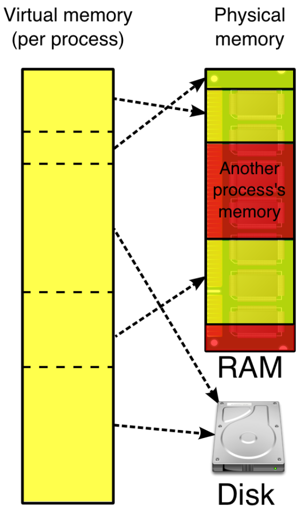
- `malloc` and `free`
  - User-level library function
  - Allocates/frees space in virtual memory
  - Often implemented using `mmap` and `munmap`
- `mmap` and `munmap`
  - System calls
  - Actually allocates/frees space in virtual memory
- So what is virtual memory?

Background: Virtual Memory



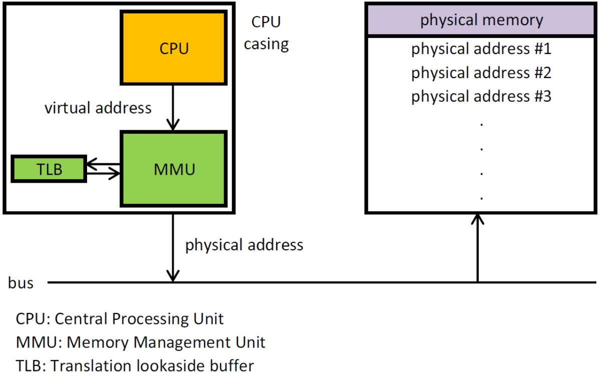
- (on x86) Instructions operate on virtual addresses

Background: Virtual Memory



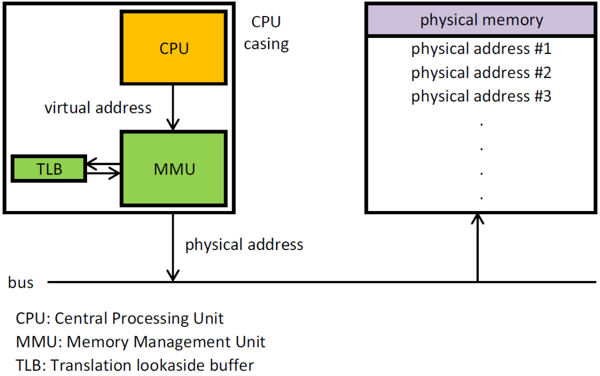
- (on x86) Instructions operate on virtual addresses
- Data may be stored:
  - In physical memory
  - On disk

Background: Virtual Memory



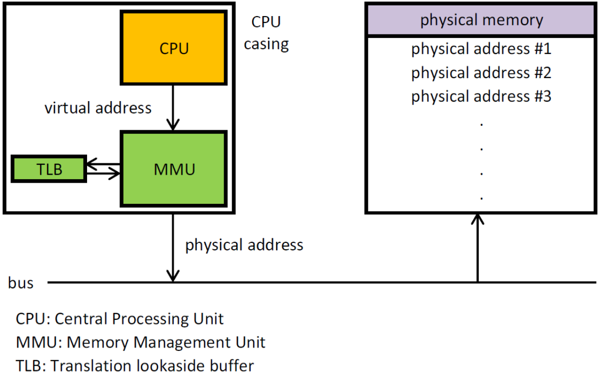
- Hardware, memory management unit (MMU)
  - Performs the translation
  - Keeps a cache (TLB) of virtual → physical mappings
  - No entry in TLB → page fault

Background: Virtual Memory



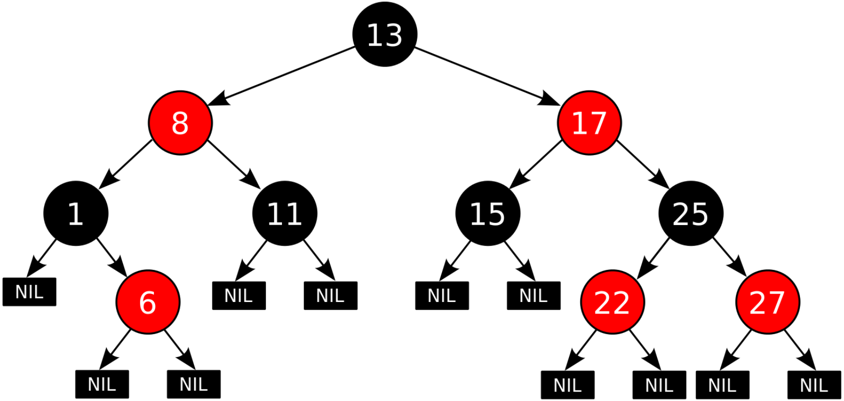
- Hardware, memory management unit (MMU)
  - Performs the translation
  - Keeps a cache (TLB) of virtual $\rightarrow$ physical mappings
  - No entry in TLB $\rightarrow$ page fault
- Operating system
  - Maintains the mapping (per process)
  - Handles page faults

Background: Virtual Memory



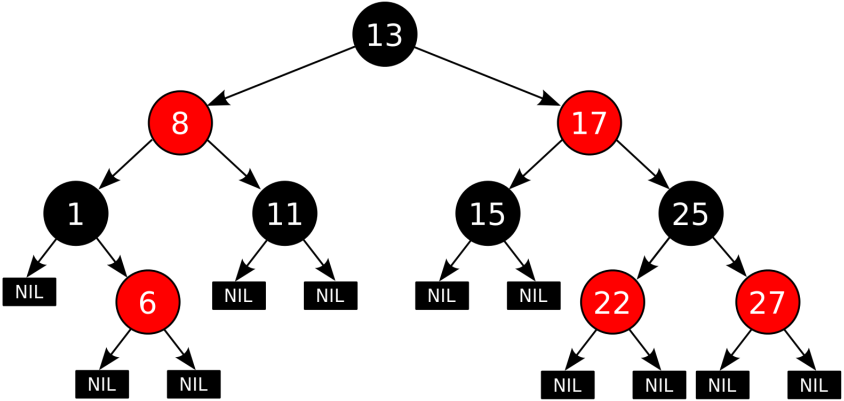
- Hardware, memory management unit (MMU)
  - Performs the translation
  - Keeps a cache (TLB) of virtual $\rightarrow$ physical mappings
  - No entry in TLB $\rightarrow$ page fault
- Operating system
  - Maintains the mapping (per process)
  - Handles page faults
- So how does the OS store this?

Background: Linux Virtual Memory



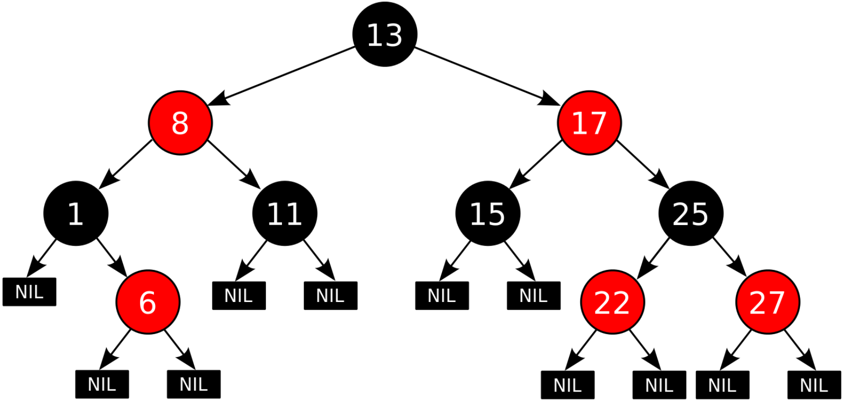
- Red-black tree
- Allows the kernel to search for memory area covering a virtual address

Background: Linux Virtual Memory



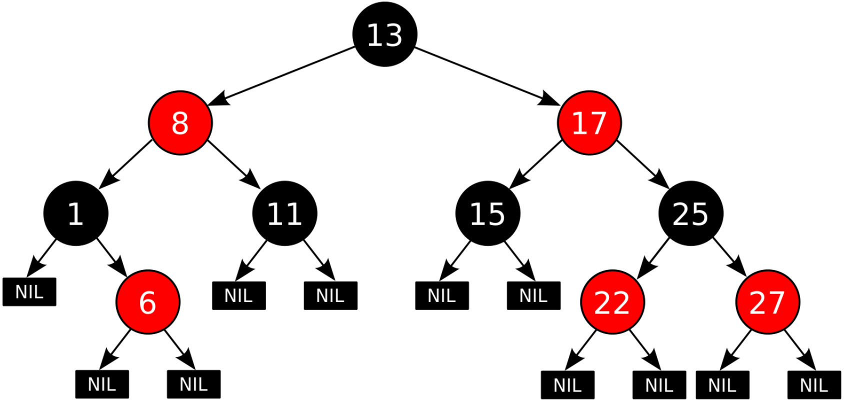
- Red-black tree
- Allows the kernel to search for memory area covering a virtual address
- Problem: A single lock per address space!

Problem 1: Multithreaded Address Space Scaling



- A single lock on this structure $\rightarrow$ `mmap` within a single process is serialized

Problem 1: Multithreaded Address Space Scaling



- A single lock on this structure → `mmap` within a single process is serialized
- Aside: prwlock paper notes that Psearchy is `mmap`-intensive

Abstract

RadixVM solves 2 problems:

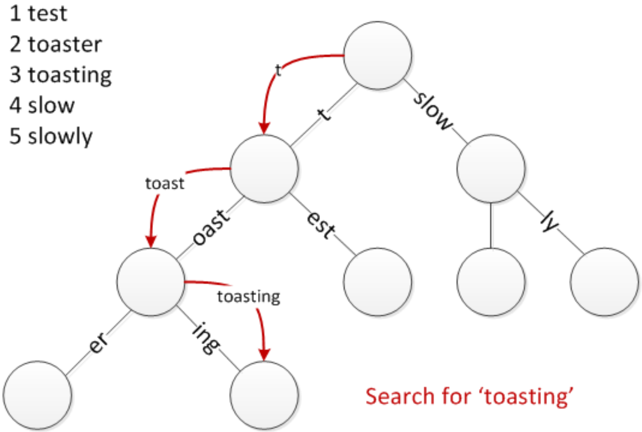
1. Multithreaded address space scaling
    1. (And reference counting)
2. Remote TLB shootdown

Abstract

RadixVM solves 2 problems:

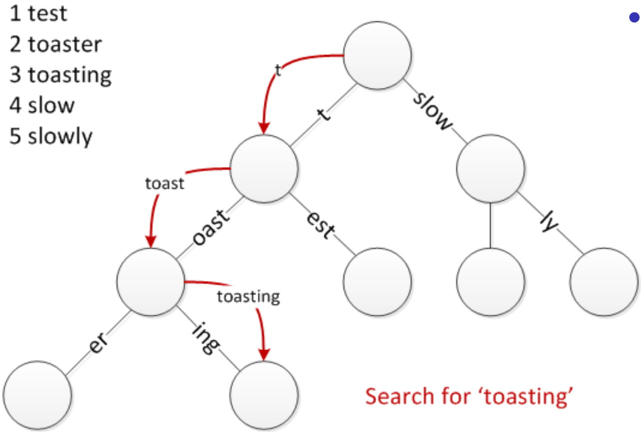
1. Multithreaded address space scaling
    1. (And reference counting)
2. Remote TLB shootdown

Background: Radix Tree



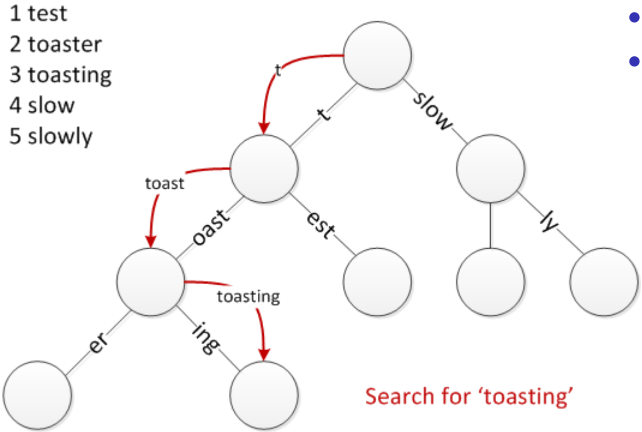
- A.K.A. prefix tree

Background: Radix Tree



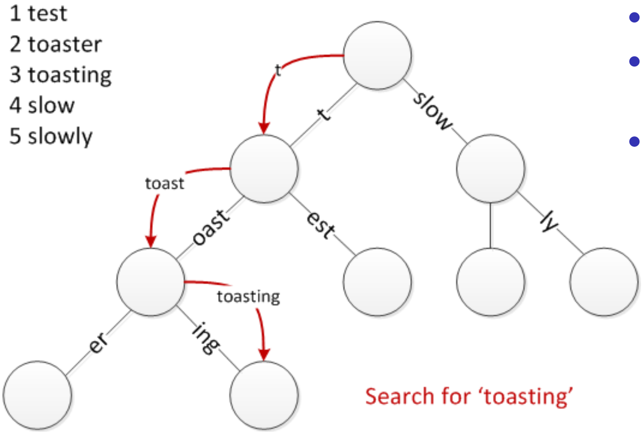
- A.K.A. prefix tree
- Edges labeled

Background: Radix Tree



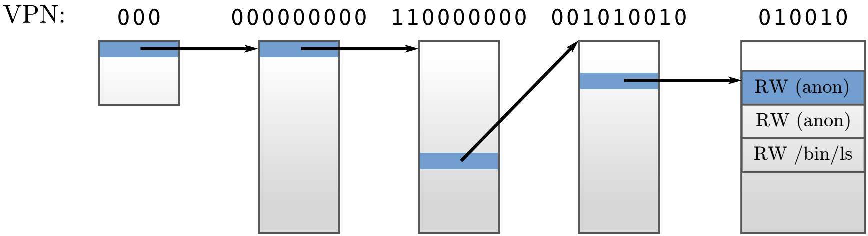
- A.K.A. prefix tree
- Edges labeled
- Concatenation of edge labels along root$\rightarrow$node path gives a string

Background: Radix Tree



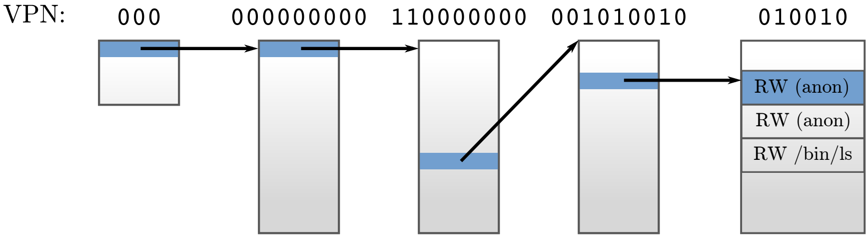
- A.K.A. prefix tree
- Edges labeled
- Concatenation of edge labels along root→node path gives a string
- In OSes, usually strings of bits

Design: RadixVM Data Structure



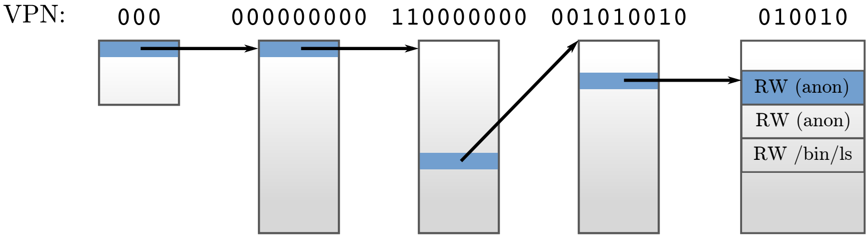
- Pretty much a page table

Design: RadixVM Data Structure



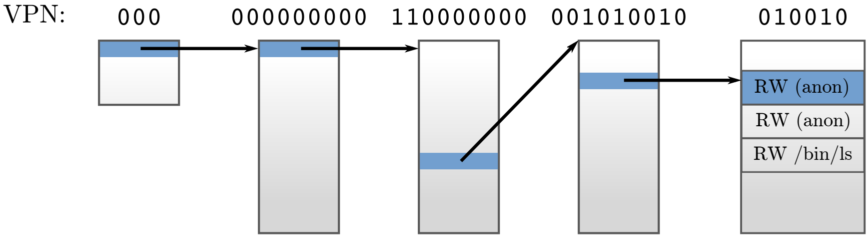
- Pretty much a page table
- Each level indexed by up to 9 bits

Design: RadixVM Data Structure



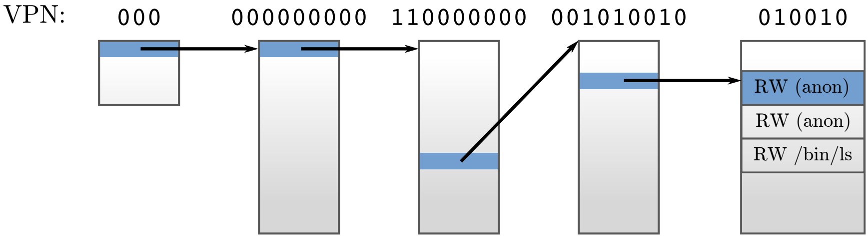
- Pretty much a page table
- Each level indexed by up to 9 bits
- Fixed-height $\rightarrow$ no balancing needed

Design: RadixVM Data Structure



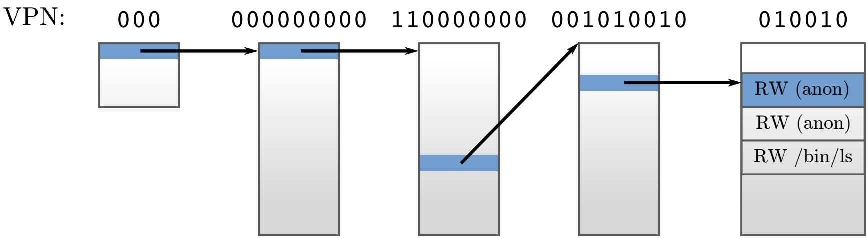
- Pretty much a page table
- Each level indexed by up to 9 bits
- Fixed-height $\rightarrow$ no balancing needed
- Lazy expansion/collapsing

Design: RadixVM Data Structure



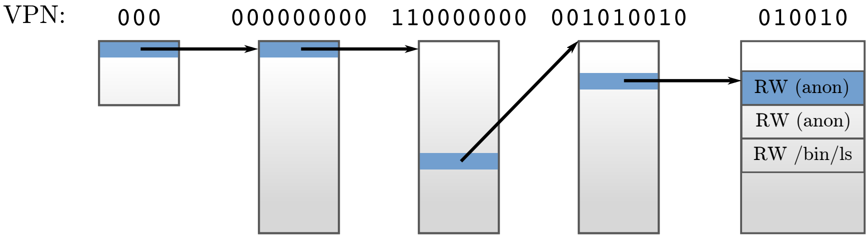
- Minimizes cache-line contention

Design: RadixVM Data Structure



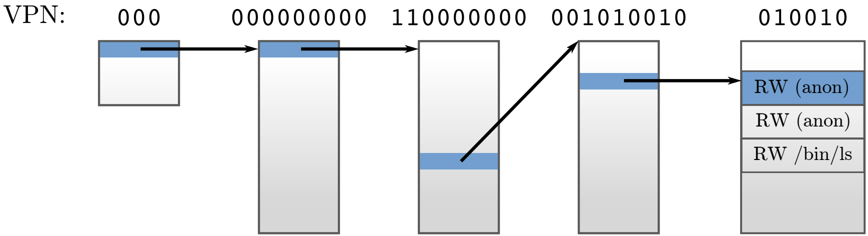
- Minimizes cache-line contention
- Supports precise range locking

Problem 1.1: When to Free



- How do we know when we can free the underlying pages?

Problem 1.1: When to Free



- How do we know when we can free the underlying pages?
- Reference counting!

Abstract

RadixVM solves 2 problems:

1. Multithreaded address space scaling
    1. (And reference counting)
2. Remote TLB shootdown

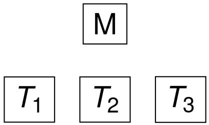
Abstract

RadixVM solves 2 problems:

1. Multithreaded address space scaling
    1. (And reference counting)
2. Remote TLB shootdown

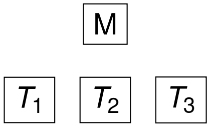
Background: Reference Counting

- 3 operations: inc, dec, zero?

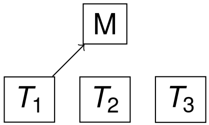


Background: Reference Counting

- 3 operations: inc, dec, zero?
  1. References: 0

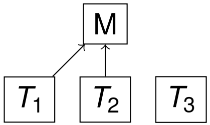


Background: Reference Counting



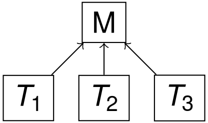
- 3 operations: inc, dec, zero?
  1. References: 0
  2. inc $\rightarrow$ References: 1

Background: Reference Counting



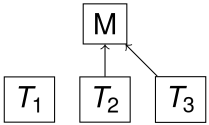
- 3 operations: inc, dec, zero?
  1. References: 0
  2. inc $\rightarrow$ References: 1
  3. inc $\rightarrow$ References: 2

Background: Reference Counting



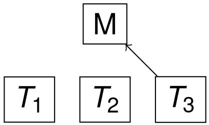
- 3 operations: inc, dec, zero?
  1. References: 0
  2. inc $\rightarrow$ References: 1
  3. inc $\rightarrow$ References: 2
  4. inc $\rightarrow$ References: 3

Background: Reference Counting



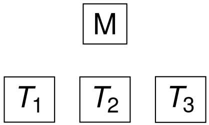
- 3 operations: inc, dec, zero?
  1. References: 0
  2. inc $\rightarrow$ References: 1
  3. inc $\rightarrow$ References: 2
  4. inc $\rightarrow$ References: 3
  5. dec $\rightarrow$ References: 2

Background: Reference Counting



- 3 operations: inc, dec, zero?
  1. References: 0
  2. inc $\rightarrow$ References: 1
  3. inc $\rightarrow$ References: 2
  4. inc $\rightarrow$ References: 3
  5. dec $\rightarrow$ References: 2
  6. dec $\rightarrow$ References: 1

Background: Reference Counting



- 3 operations: inc, dec, zero?
  1. References: 0
  2. inc $\rightarrow$ References: 1
  3. inc $\rightarrow$ References: 2
  4. inc $\rightarrow$ References: 3
  5. dec $\rightarrow$ References: 2
  6. dec $\rightarrow$ References: 1
  7. dec $\rightarrow$ References: 0

Background: Reference Counting

T_1 T_2 T_3

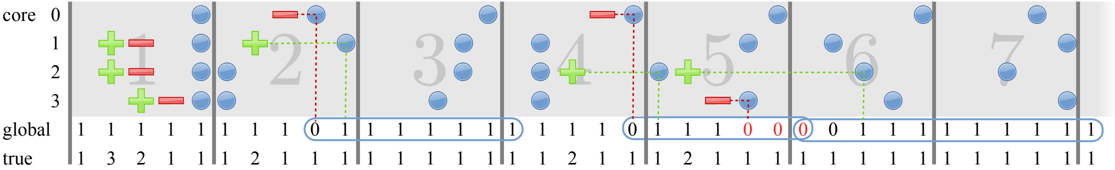
- 3 operations: inc, dec, zero?
  1. References: 0
  2. inc $\rightarrow$ References: 1
  3. inc $\rightarrow$ References: 2
  4. inc $\rightarrow$ References: 3
  5. dec $\rightarrow$ References: 2
  6. dec $\rightarrow$ References: 1
  7. dec $\rightarrow$ References: 0
  8. zero? $\rightarrow$ Free M!

Background: Reference Counting

T_1 T_2 T_3

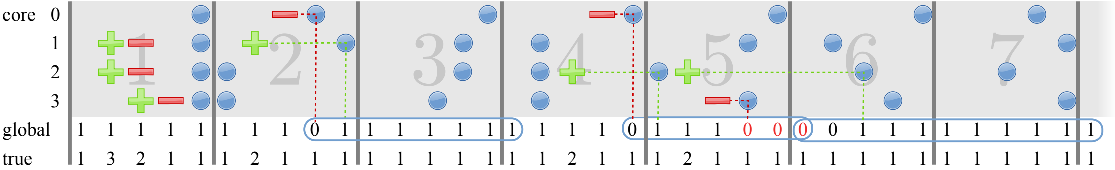
- 3 operations: inc, dec, zero?
  1. References: 0
  2. inc $\rightarrow$ References: 1
  3. inc $\rightarrow$ References: 2
  4. inc $\rightarrow$ References: 3
  5. dec $\rightarrow$ References: 2
  6. dec $\rightarrow$ References: 1
  7. dec $\rightarrow$ References: 0
  8. zero? $\rightarrow$ Free M!
- But: single counter $\rightarrow$ contention

Design: Refcache



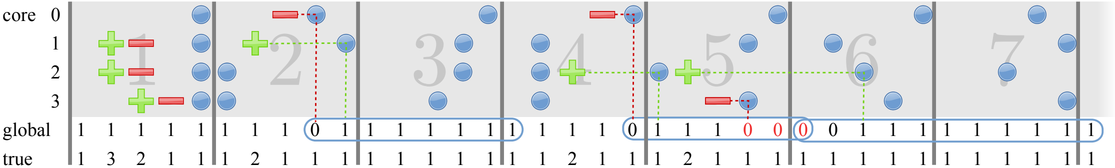
- Per-core lazy counting

Design: Refcache



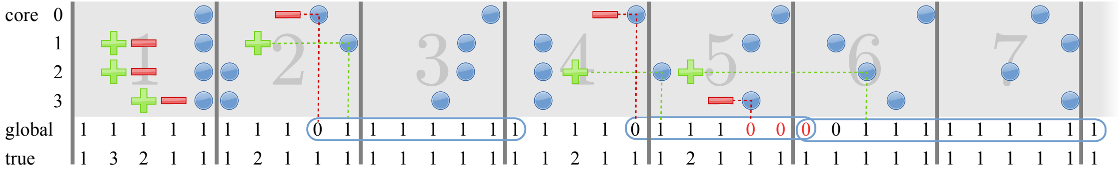
- Per-core lazy counting
  - Each core stores a delta

Design: Refcache



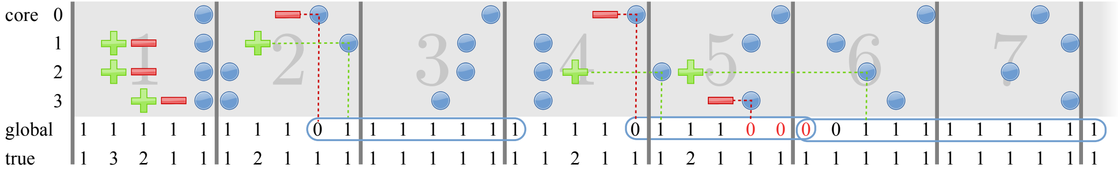
- Per-core lazy counting
  - Each core stores a delta
  - Delta updated lazily (blue circles)

Design: Refcache



- Per-core lazy counting
  - Each core stores a delta
  - Delta updated lazily (blue circles)
- Divides time into epochs

Design: Refcache



- Per-core lazy counting
  - Each core stores a delta
  - Delta updated lazily (blue circles)
- Divides time into epochs
- Ref. count zero for an entire epoch → free

Abstract

RadixVM solves 2 problems:

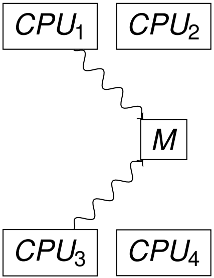
1. Multithreaded address space scaling
    1. (And reference counting)
2. Remote TLB shootdown

Abstract

RadixVM solves 2 problems:

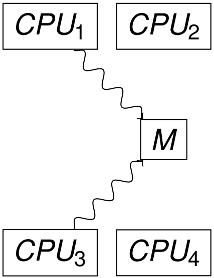
1. Multithreaded address space scaling
    1. (And reference counting)
2. Remote TLB shootdown

Problem 2: Remote TLB Shootdowns



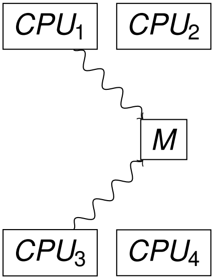
- Processes on CPU_1 and CPU_3 share memory area M

Problem 2: Remote TLB Shootdowns



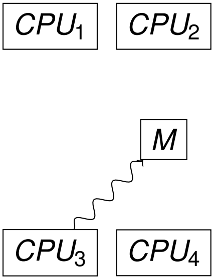
- Processes on CPU_1 and CPU_3 share memory area M
- A process on CPU_1 unmaps M

Problem 2: Remote TLB Shootdowns



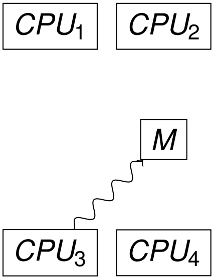
- Processes on CPU_1 and CPU_3 share memory area M
- A process on CPU_1 unmaps M
  - Flush local TLB entry

Problem 2: Remote TLB Shootdowns



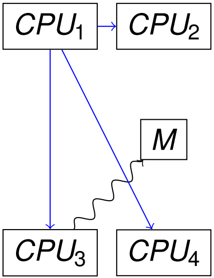
- Processes on CPU_1 and CPU_3 share memory area M
- A process on CPU_1 unmaps M
  - Flush local TLB entry

Problem 2: Remote TLB Shootdowns



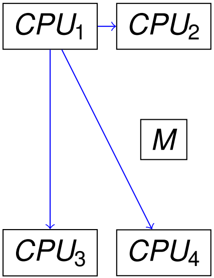
- Processes on CPU_1 and CPU_3 share memory area M
- A process on CPU_1 unmaps M
  - Flush local TLB entry
  - Flush remote TLB entries (of anyone sharing)

Problem 2: Remote TLB Shootdowns



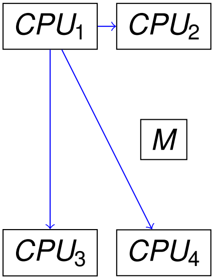
- Processes on CPU_1 and CPU_3 share memory area M
- A process on CPU_1 unmaps M
  - Flush local TLB entry
  - Flush remote TLB entries (of anyone sharing)
- Linux: sends a message to all CPU

Problem 2: Remote TLB Shootdowns



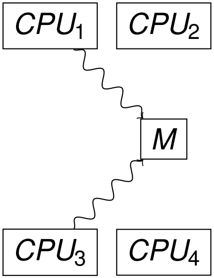
- Processes on CPU_1 and CPU_3 share memory area M
- A process on CPU_1 unmaps M
  - Flush local TLB entry
  - Flush remote TLB entries (of anyone sharing)
- Linux: sends a message to all CPU

Problem 2: Remote TLB Shootdowns



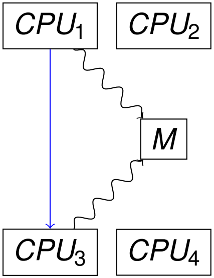
- Processes on CPU_1 and CPU_3 share memory area M
- A process on CPU_1 unmaps M
  - Flush local TLB entry
  - Flush remote TLB entries (of anyone sharing)
- Linux: sends a message to all CPU
- This is expensive!

Design: Targeted TLB Shootdowns



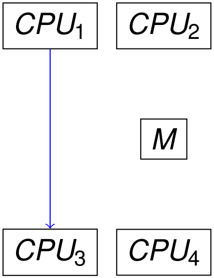
- Store metadata on which cores may have address in TLB

Design: Targeted TLB Shootdowns



- Store metadata on which cores may have address in TLB
- Only flush TLBs on cores which may share that memory

Design: Targeted TLB Shootdowns



- Store metadata on which cores may have address in TLB
- Only flush TLBs on cores which may share that memory

Implementation

- *Not* implemented on Linux

Implementation

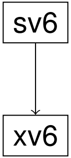
- *Not* implemented on Linux
  - Too complicated

Implementation

sv6

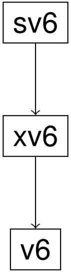
- *Not* implemented on Linux
  - Too complicated
- Implemented on sv6 (C++)

Implementation



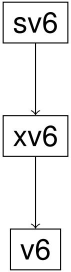
- *Not* implemented on Linux
  - Too complicated
- Implemented on sv6 (C++)
  - Based on xv6 (ANSI C)

Implementation



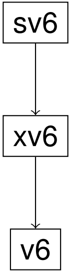
- *Not* implemented on Linux
  - Too complicated
- Implemented on sv6 (C++)
  - Based on xv6 (ANSI C)
    - Based on v6 Unix (K&R C)

Implementation



- *Not* implemented on Linux
  - Too complicated
- Implemented on sv6 (C++)
  - Based on xv6 (ANSI C)
    - Based on v6 Unix (K&R C)
  - Academic OS

Implementation



- *Not* implemented on Linux
  - Too complicated
- Implemented on sv6 (C++)
  - Based on xv6 (ANSI C)
    - Based on v6 Unix (K&R C)
  - Academic OS
  - `https://github.com/aclements/sv6`

Comparison

- Compared against:

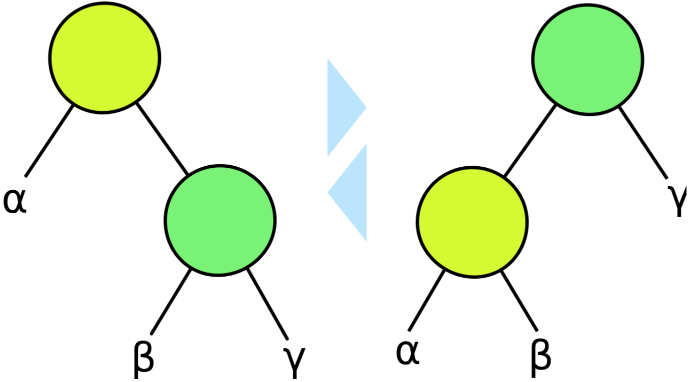
Comparison

- Compared against:
    - Default Linux

Comparison

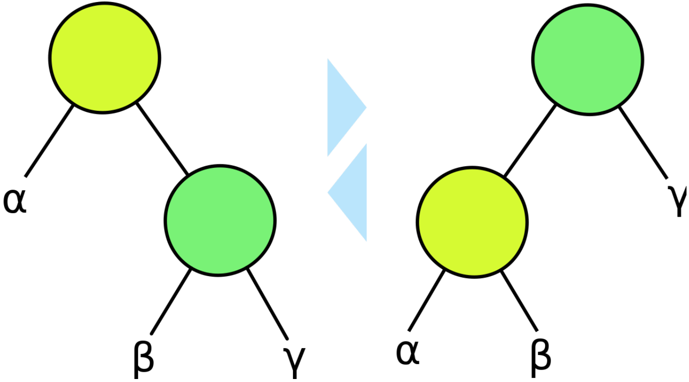
- Compared against:
    - Default Linux
    - “Bonsai”

Background: Bonsai



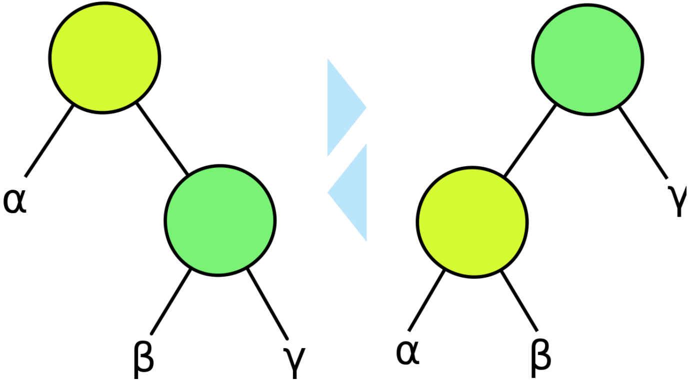
- Designed by the same authors

Background: Bonsai



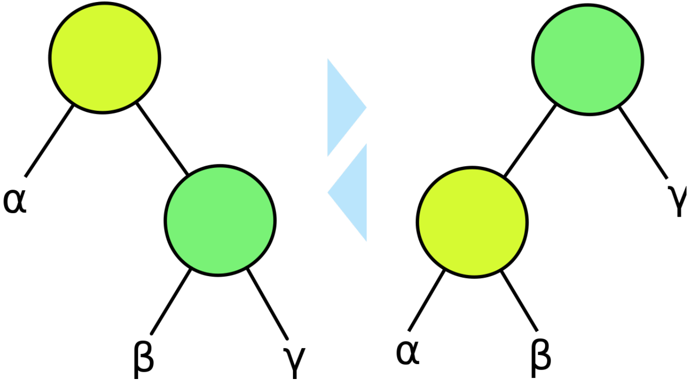
- Designed by the same authors
- Uses an RCU-based balanced binary tree

Background: Bonsai



- Designed by the same authors
- Uses an RCU-based balanced binary tree
- Maintains bounded balance rather than strict balance (this means fewer rotations)

Background: Bonsai



- Designed by the same authors
- Uses an RCU-based balanced binary tree
- Maintains bounded balance rather than strict balance (this means fewer rotations)
- Rotations construct a new subtree rather than mutate the old one

Application: Metis

- MapReduce Library

Application: Metis

- MapReduce Library
- Single-server

Application: Metis

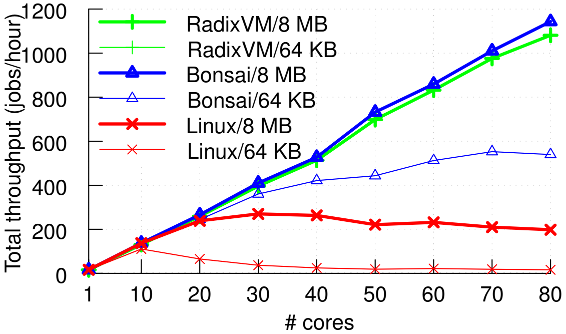
- MapReduce Library
- Single-server
- Multithreaded

Application: Metis

- MapReduce Library
- Single-server
- Multithreaded
- Stresses concurrent `mmap`s and `pagefault`s, but not concurrent `munmap`s

Application: Metis

- MapReduce Library
- Single-server
- Multithreaded
- Stresses concurrent `mmap`s and `pagefault`s, but not concurrent `munmap`s



Microbenchmark: Local

- `mmap` a private 4KB region in shared address space

Microbenchmark: Local

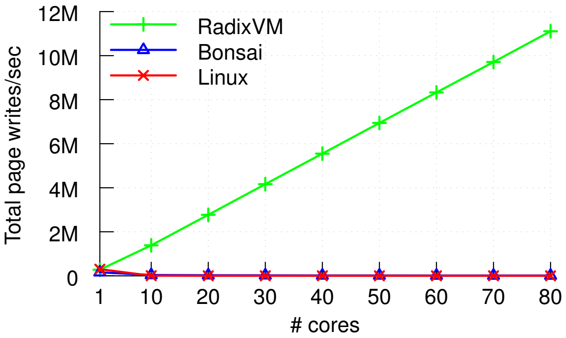
- `mmap` a private 4KB region in shared address space
- Write to every page in region

Microbenchmark: Local

- `mmap` a private 4KB region in shared address space
- Write to every page in region
- `munmap` region

Microbenchmark: Local

- `mmap` a private 4KB region in shared address space
- Write to every page in region
- `munmap` region



Microbenchmark: Pipeline

- Each thread `mmap` a region

Microbenchmark: Pipeline

- Each thread `mmap` a region
- Write to every page in region

Microbenchmark: Pipeline

- Each thread `mmap` a region
- Write to every page in region
- Pass region to next thread

Microbenchmark: Pipeline

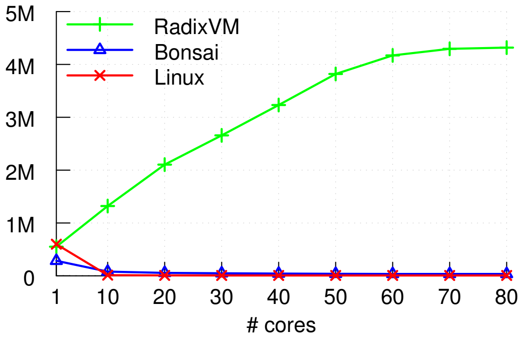
- Each thread `mmap` a region
- Write to every page in region
- Pass region to next thread
- Write to every page in passed region

Microbenchmark: Pipeline

- Each thread `mmap` a region
- Write to every page in region
- Pass region to next thread
- Write to every page in passed region
- `munmap` region

Microbenchmark: Pipeline

- Each thread `mmap` a region
- Write to every page in region
- Pass region to next thread
- Write to every page in passed region
- `munmap` region



Microbenchmark: Global

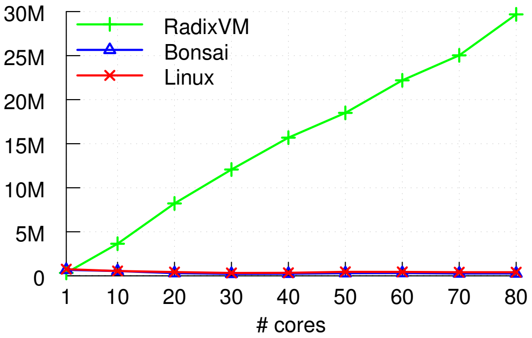
- Each thread `mmap` a 64KB region within a large region of memory

Microbenchmark: Global

- Each thread `mmap` a 64KB region within a large region of memory
- All threads access all pages in random order

Microbenchmark: Global

- Each thread `mmap` a 64KB region within a large region of memory
- All threads access all pages in random order

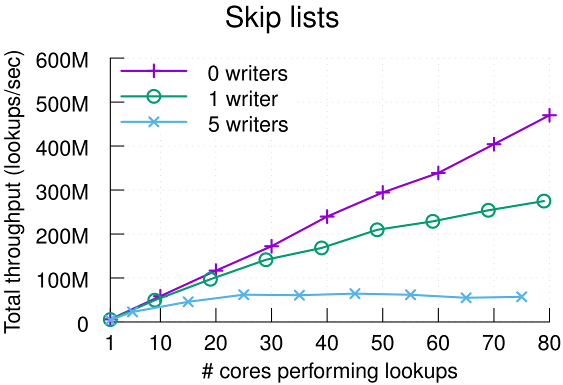


Question

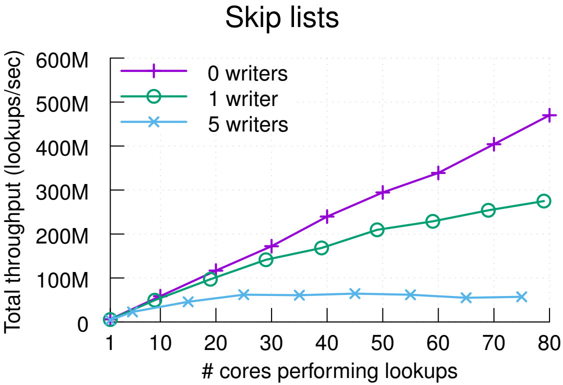
Do we really need all 3 pieces?

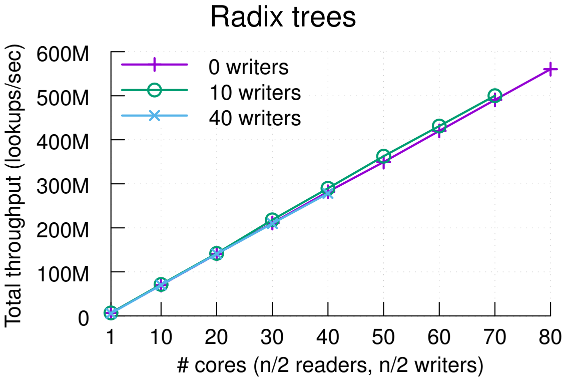
- Radix trees
- Refcache
- Targeted TLB shootdown

Question: Do we need radix trees?

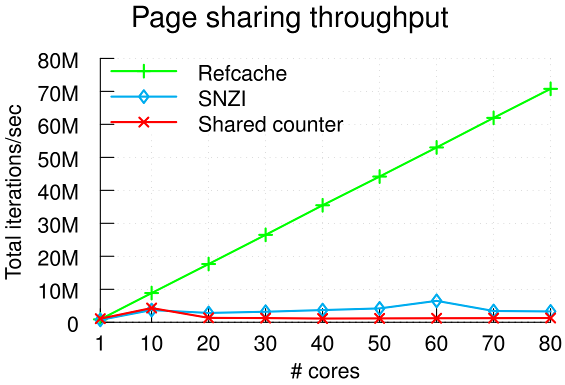


Question: Do we need radix trees?



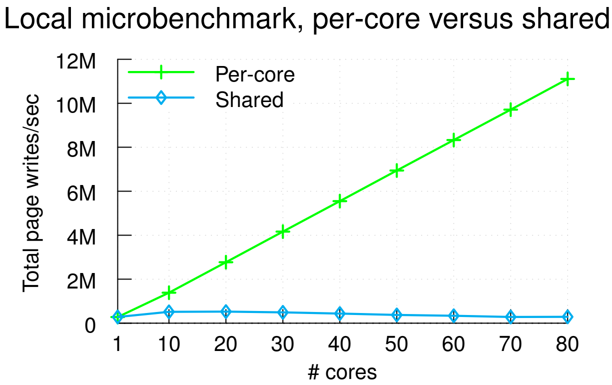


Question: Do we need Refcache?



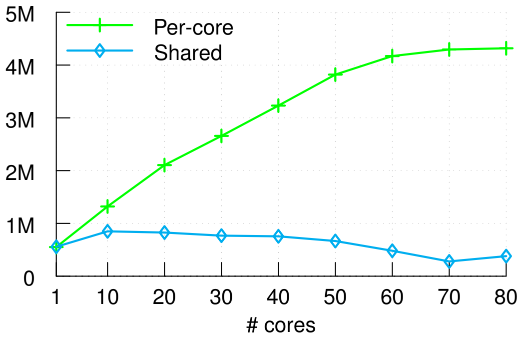
SNZI: Scalable Non-Zero Indicators

Question: Do we need targeted TLB shootdown?

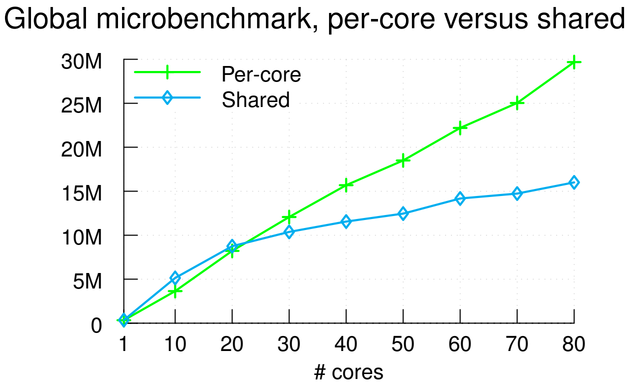


Question: Do we need targeted TLB shootdown?

Pipeline microbenchmark, per-core versus shared



Question: Do we need targeted TLB shootdown?

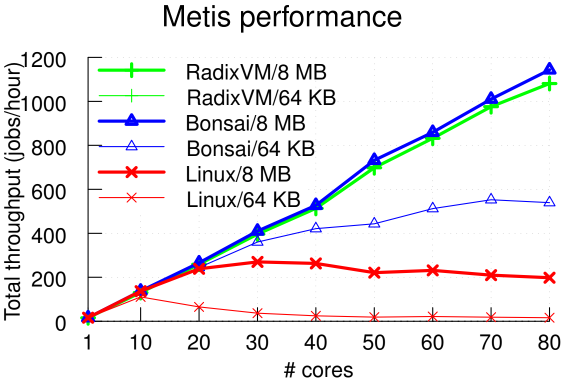


Issue: Memory Overhead

		Linux		Radix tree
	RSS	VMA tree	Page table	(rel. to Linux)
Firefox	352 MB	117 KB	1.5 MB	3.9 MB ($2.4\times$)
Chrome	152 MB	124 KB	1.1 MB	2.4 MB ($2.0\times$)
Apache	16 MB	44 KB	368 KB	616 KB ($1.5\times$)
MySQL	84 MB	18 KB	348 KB	980 KB ($2.7\times$)

- RSS
  - Resident Set Size
  - physical memory used by a process
- VMA
  - Virtual Memory Areas
  - stored in a red-black tree in Linux

Summary



- Good: Scales well on
  - Metis, real-world application
  - Microbenchmarks
- Bad: Increased memory overhead

	RSS	Linux	Radix tree (rel. to Linux)
Firefox	352 MB	1.5 MB	3.9 MB (2.4×)
Chrome	152 MB	1.1 MB	2.4 MB (2.0×)
Apache	16 MB	368 KB	616 KB (1.5×)
MySQL	84 MB	348 KB	980 KB (2.7×)

References

- Clements, Austin T., M. Frans Kaashoek, and Nickolai Zeldovich. "RadixVM: Scalable address spaces for multithreaded applications." In *Proceedings of the 8th ACM European Conference on Computer Systems*, pp. 211-224. ACM, 2013.
  - Revised version: `https://pdos.csail.mit.edu/papers/radixvm:eurosys13-2014-08-05.pdf`
- 6.828 Lecture on RadixVM: `https://www.youtube.com/watch?v=qlg7jqBtR4c`
- Clements, Austin T., M. Frans Kaashoek, and Nickolai Zeldovich. "Scalable address spaces using RCU balanced trees." *ACM SIGPLAN Notices* 47, no. 4 (2012): 199-210.
  - Available online: `https://pdos.csail.mit.edu/papers/rcuvm:asplos12.pdf`
- Linux VM info from: `http://duartes.org/gustavo/blog/post/how-the-kernel-manages-your-memory/`

Attribution

- Refcache diagram, radix data structure diagram, and RadixVM charts used with permission by Austin Clements
- Virtual memory diagram by Ehamberg (Own work) [CC BY-SA 3.0 (`http://creativecommons.org/licenses/by-sa/3.0`)], via Wikimedia Commons
- MMU diagram by Mdjango, Andrew S. Tanenbaum (Own work) [CC BY-SA 3.0 (`http://creativecommons.org/licenses/by-sa/3.0`)], via Wikimedia Commons
- Address space diagram by Majenko (Own work) [CC BY-SA 4.0 (`http://creativecommons.org/licenses/by-sa/4.0`)], via Wikimedia Commons
- Patricia trie diagram by Saffles (Microsoft Visio) [CC BY-SA 3.0 (`http://creativecommons.org/licenses/by-sa/3.0`)], via Wikimedia Commons
- Binary tree rotation diagram by Josell7 (Own work) [CC BY-SA 3.0 (`http://creativecommons.org/licenses/by-sa/3.0`)], via Wikimedia Commons

License

- These slides are distributed under the creative commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0).
- See http://creativecommons.org/licenses/by-sa/4.0/ for details.