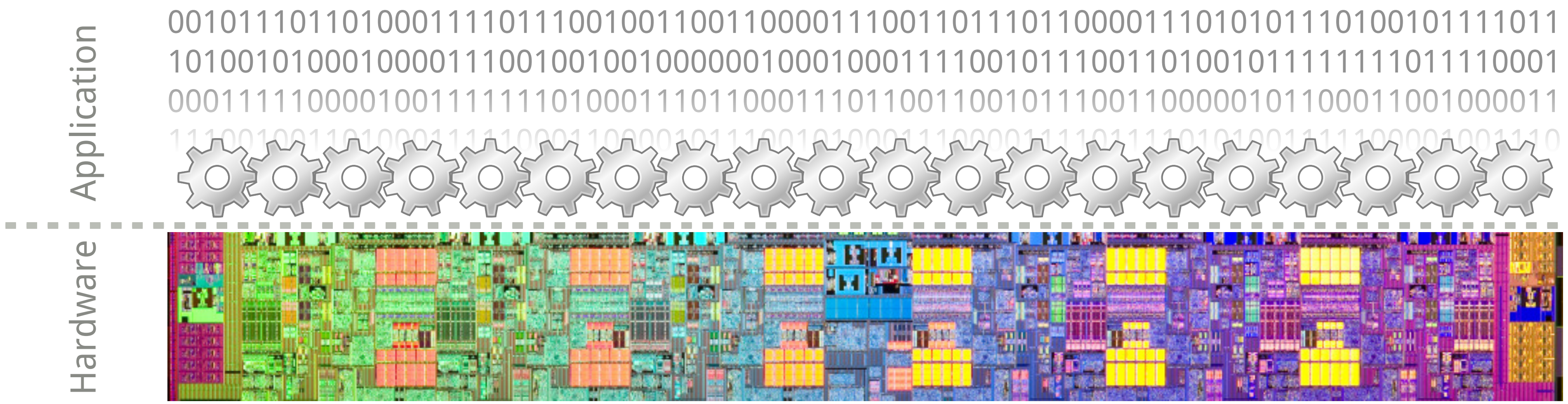
RadixVM: Scalable address spaces for multithreaded applications

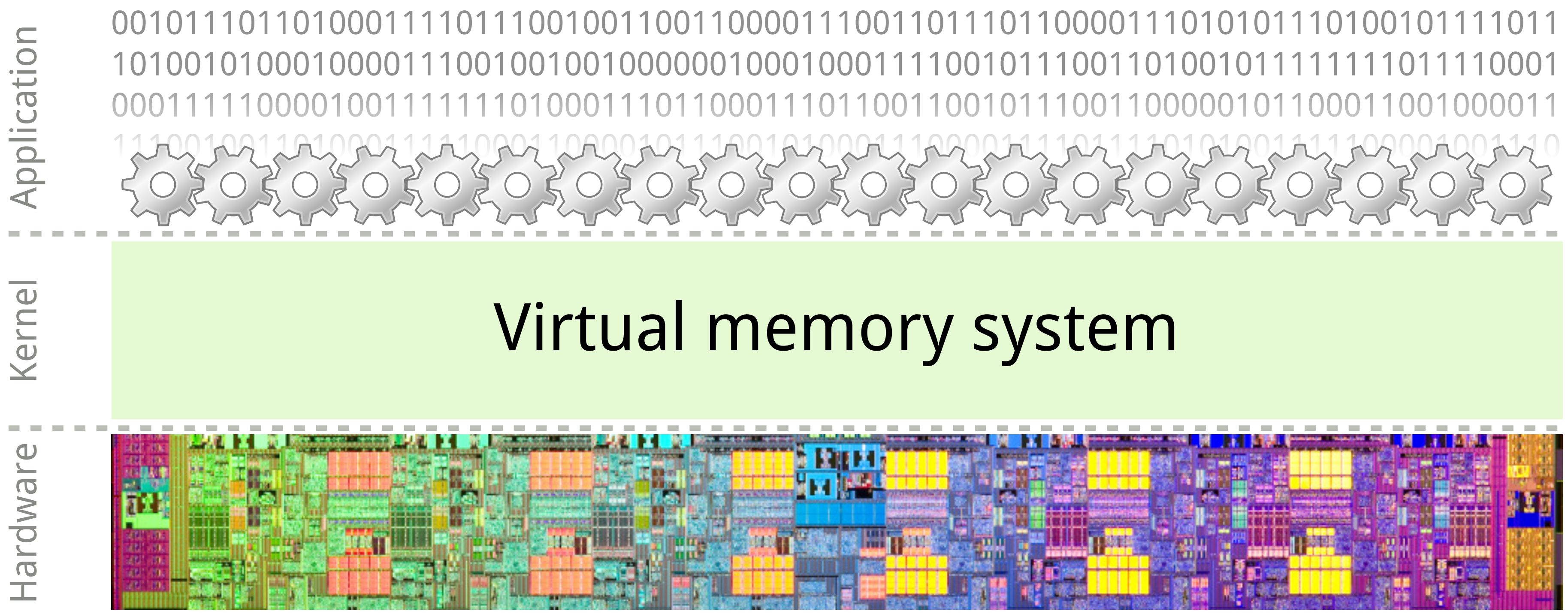
Austin T. Clements
M. Frans Kaashoek
Nickolai Zeldovich

MIT CSAIL

Parallel applications use VM intensively

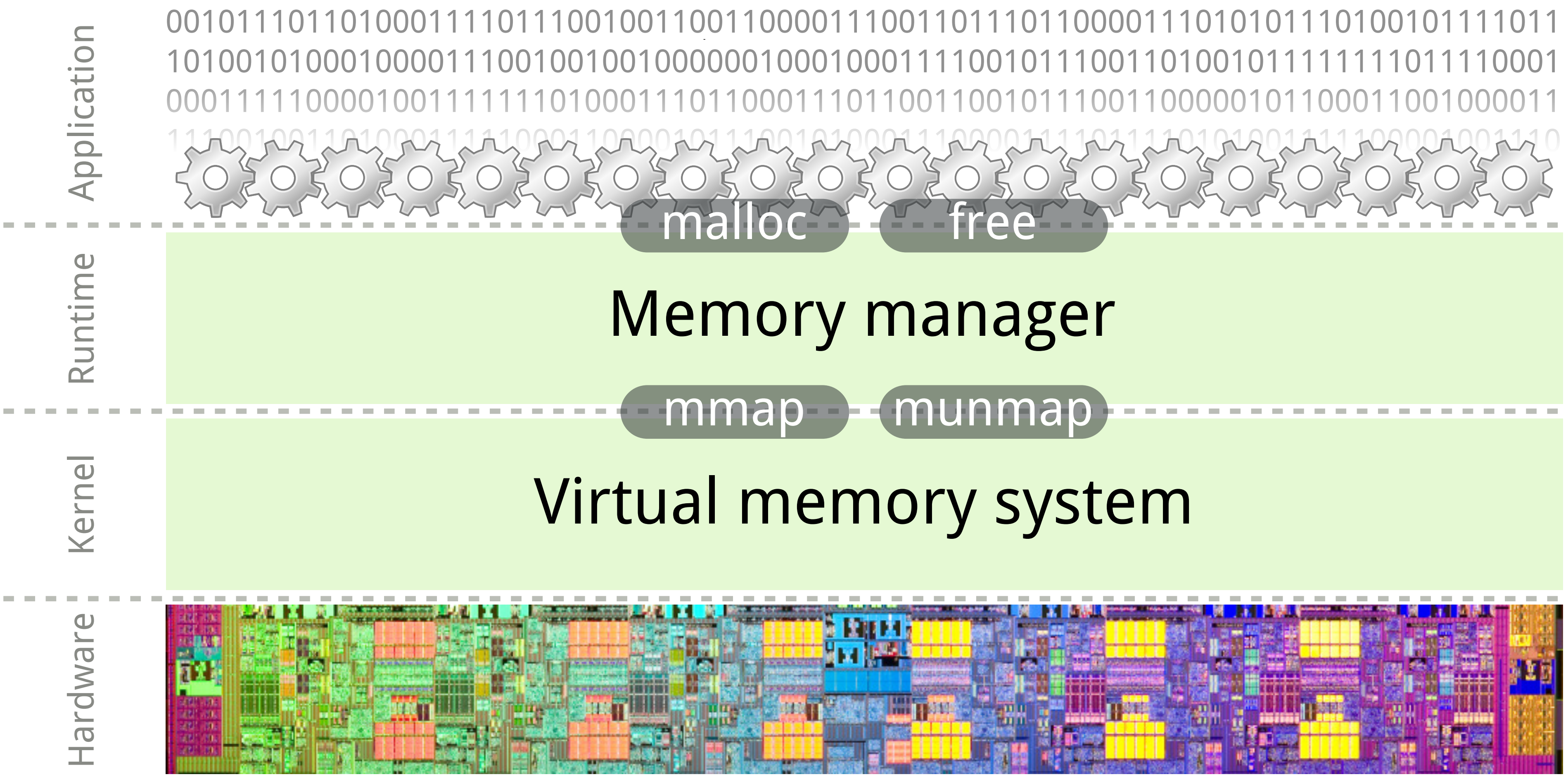


Parallel applications use VM intensively



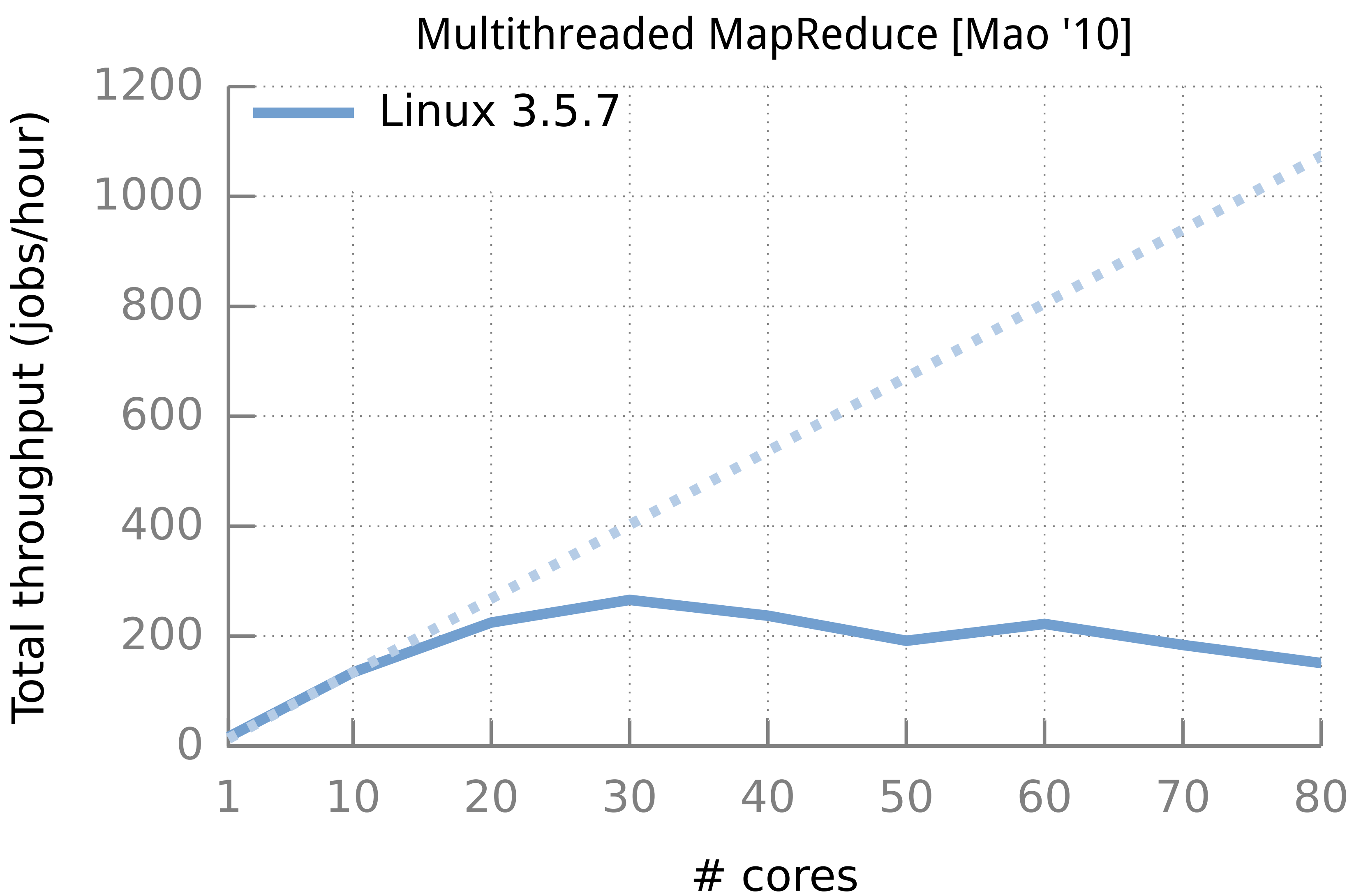
Every popular operating system serializes basic VM operations like mmap and munmap.

Parallel applications use VM intensively

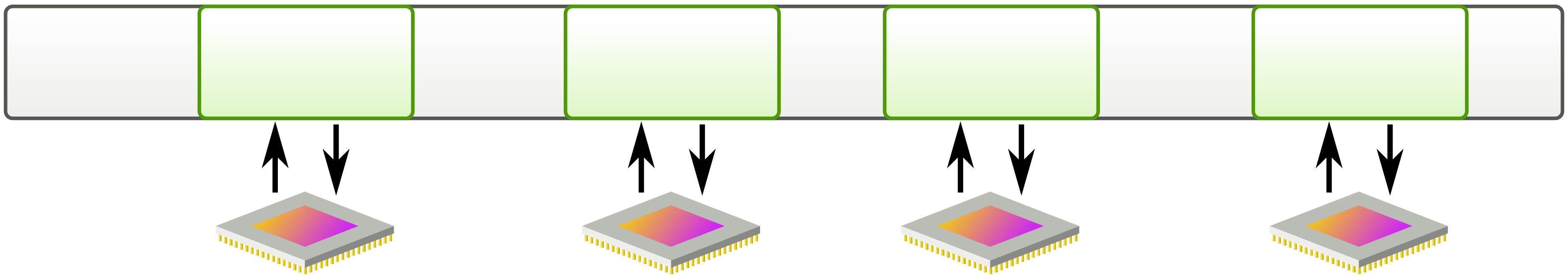


Every popular operating system serializes basic VM operations like mmap and munmap.

Application performance suffers

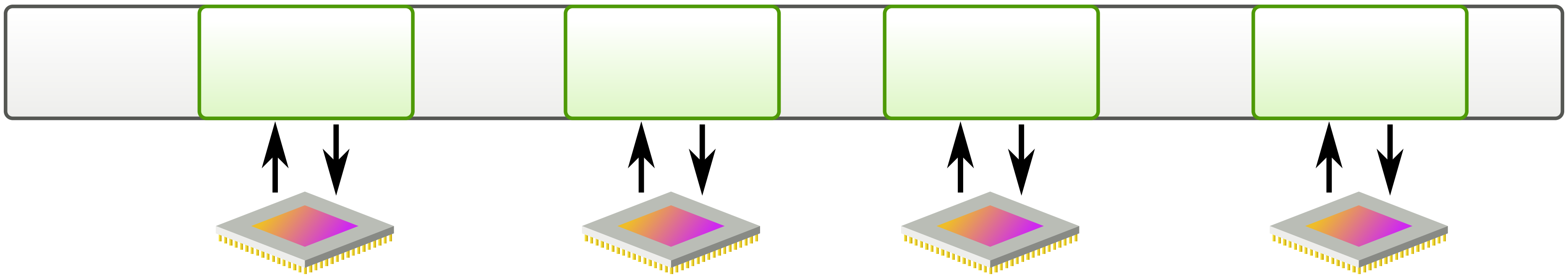


Inside parallel applications



Independent VM operations on non-overlapping regions.

Inside parallel applications



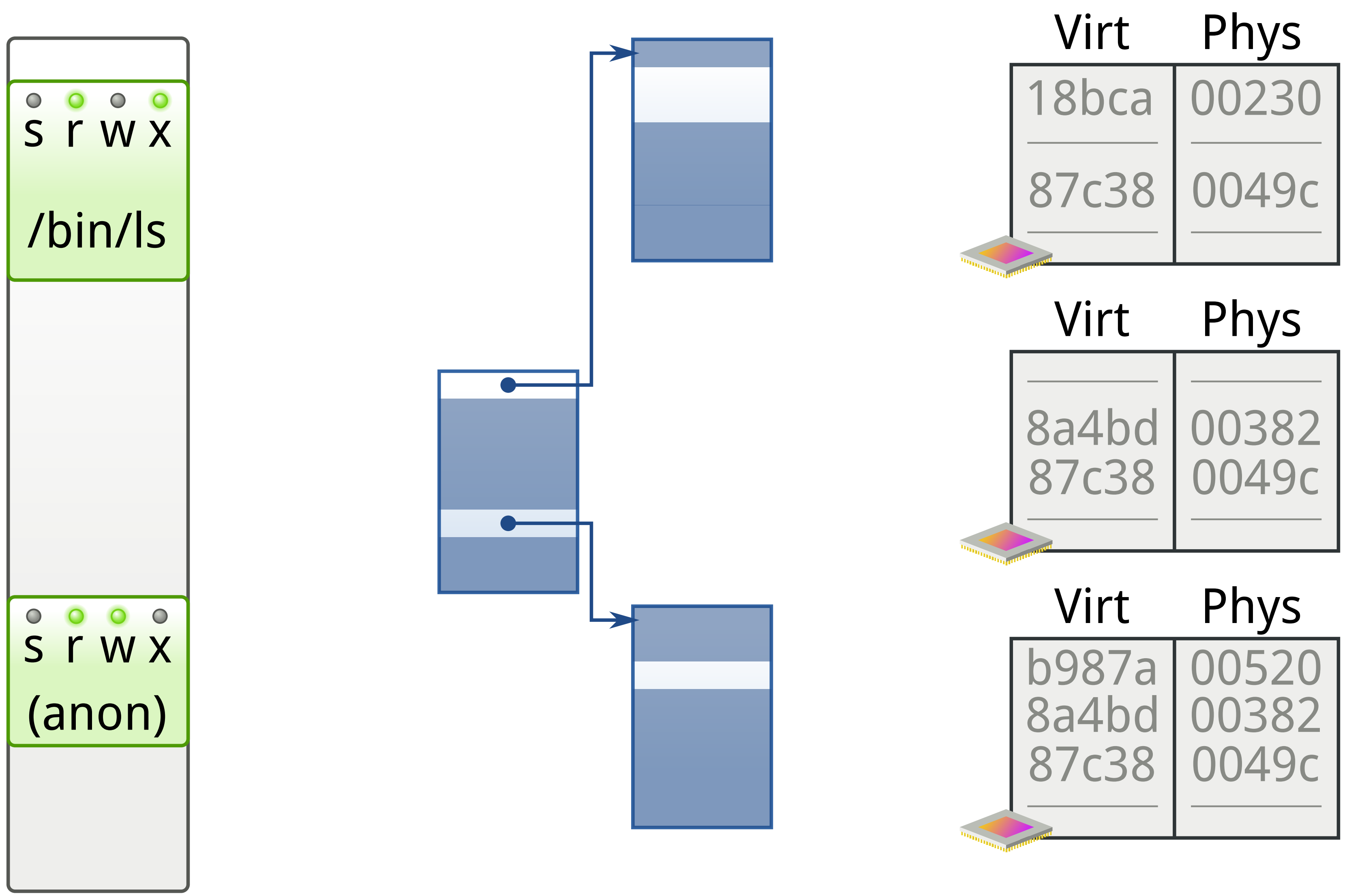
Independent VM operations on non-overlapping regions.

Common pattern for parallel applications.

Goal

Perfectly scalable mmap, munmap, and page fault operations on non-overlapping address space regions.

Structure of a VM system

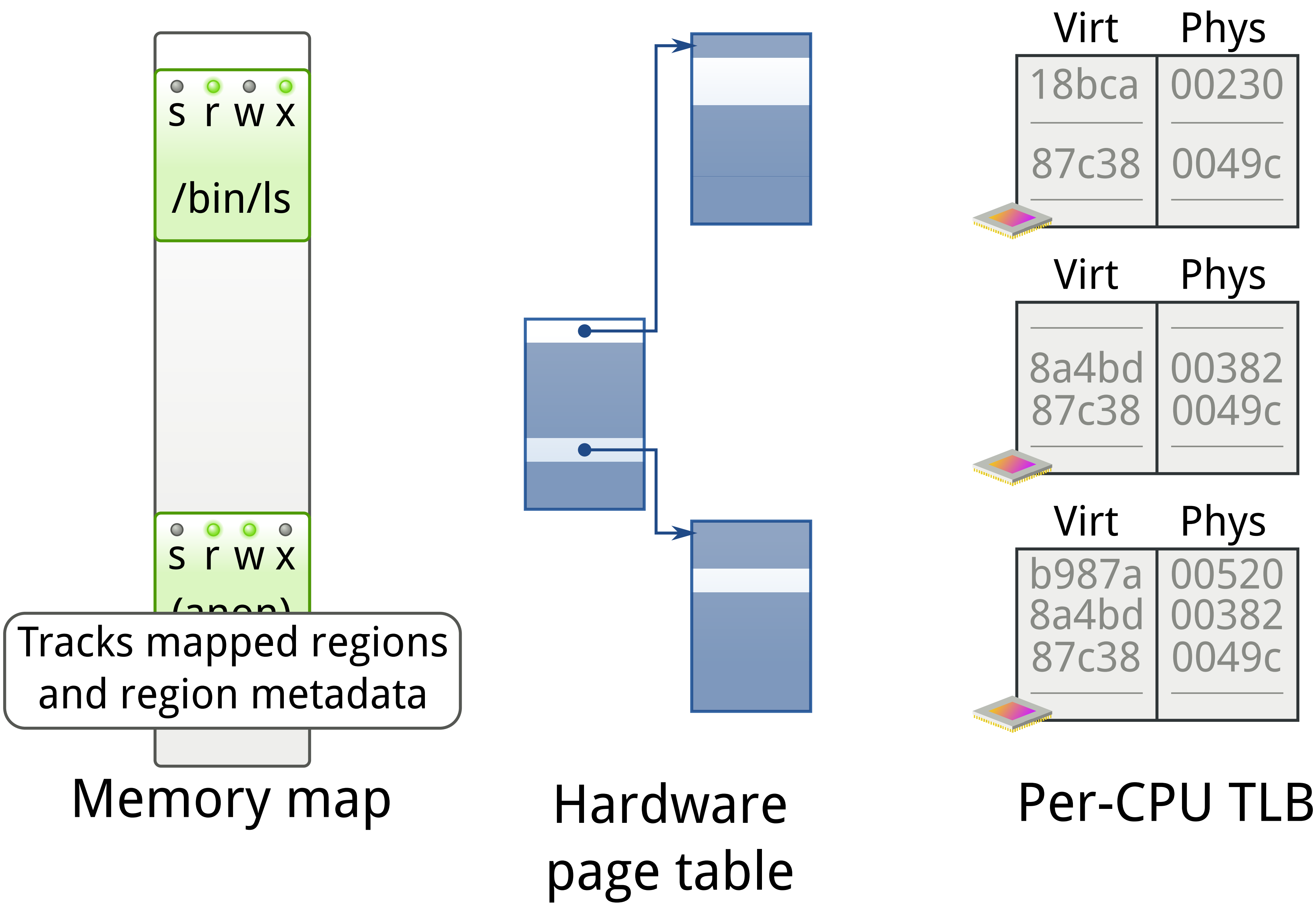


Memory map

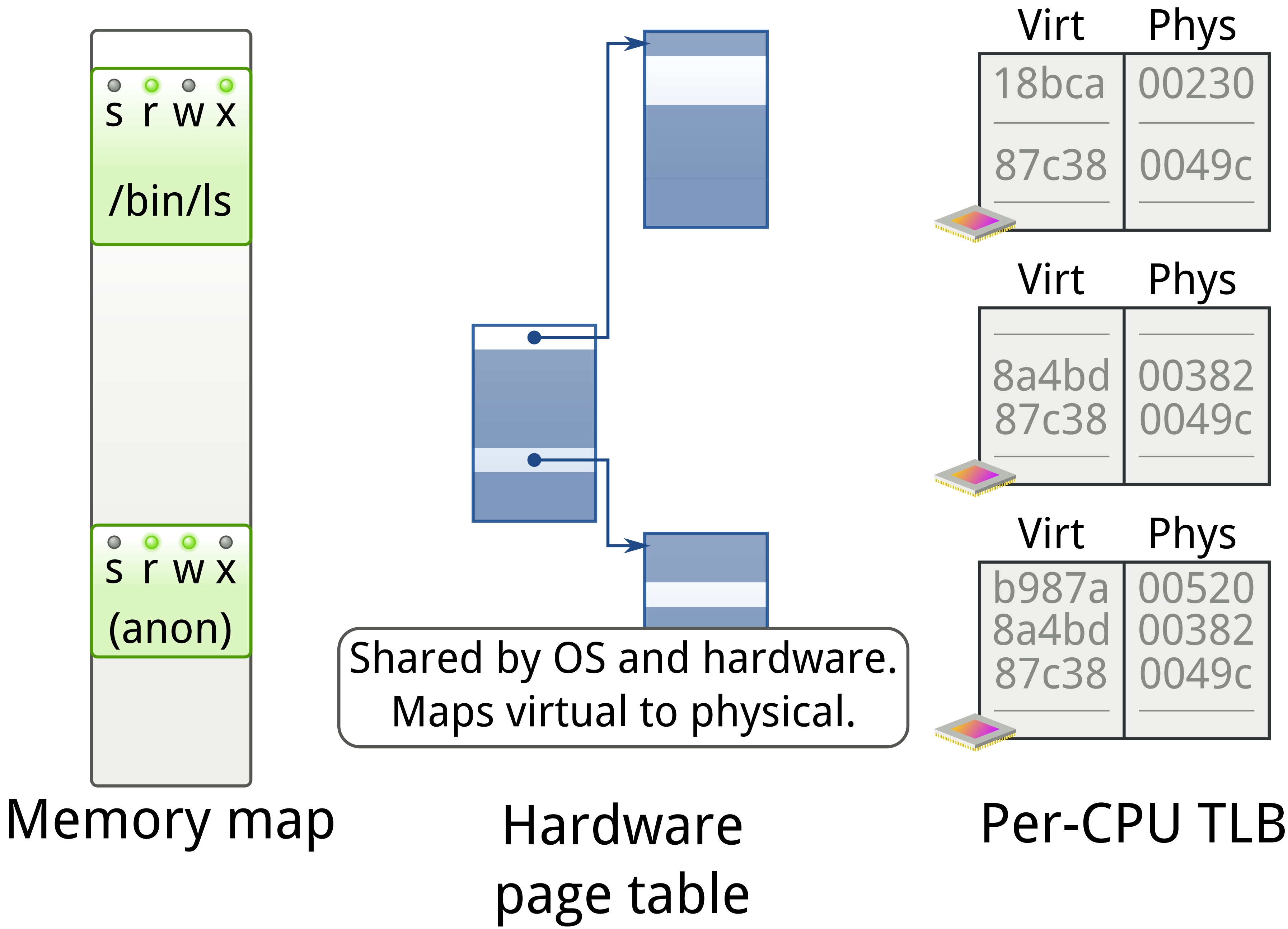
Hardware page table

Per-CPU TLB

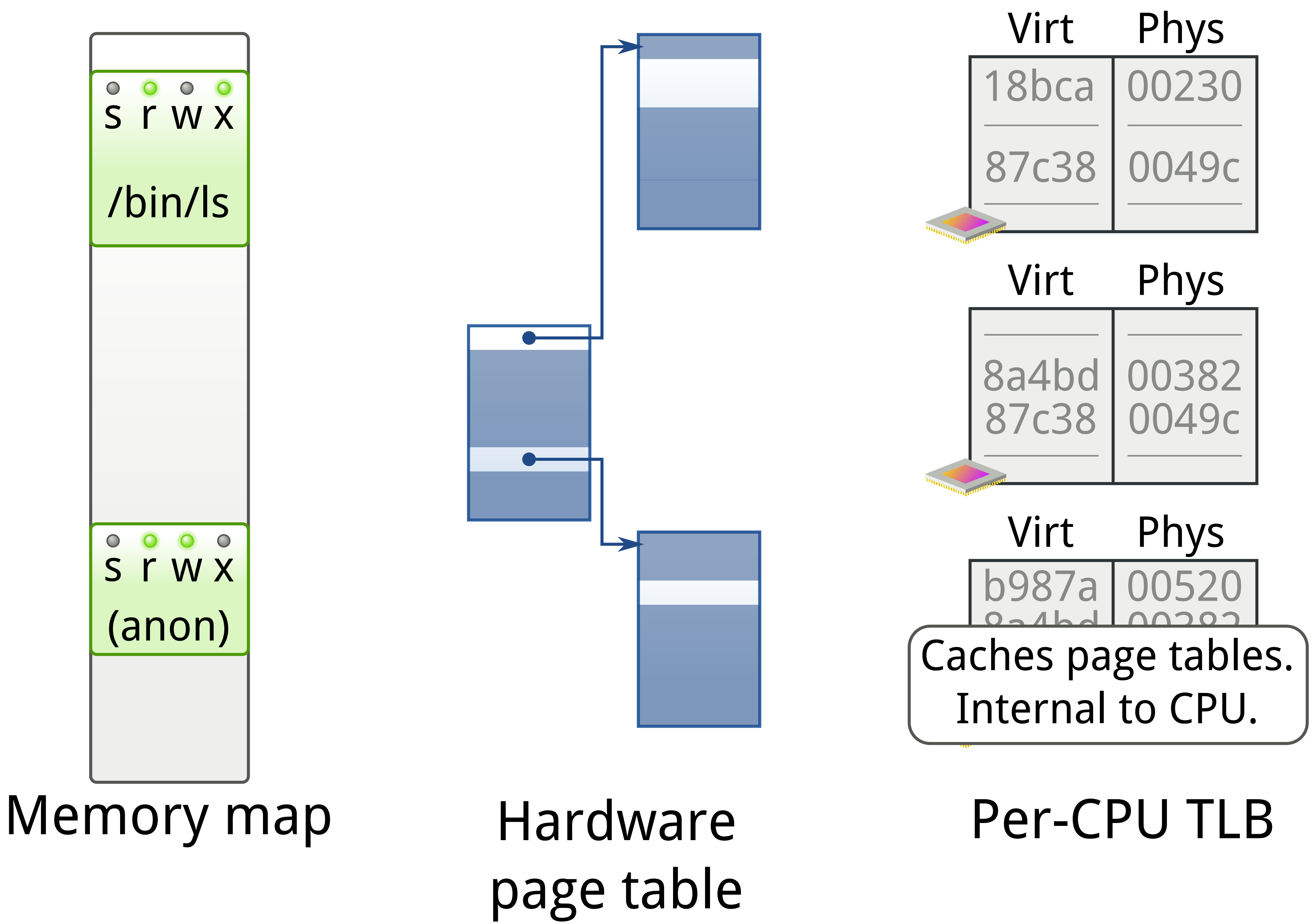
Structure of a VM system



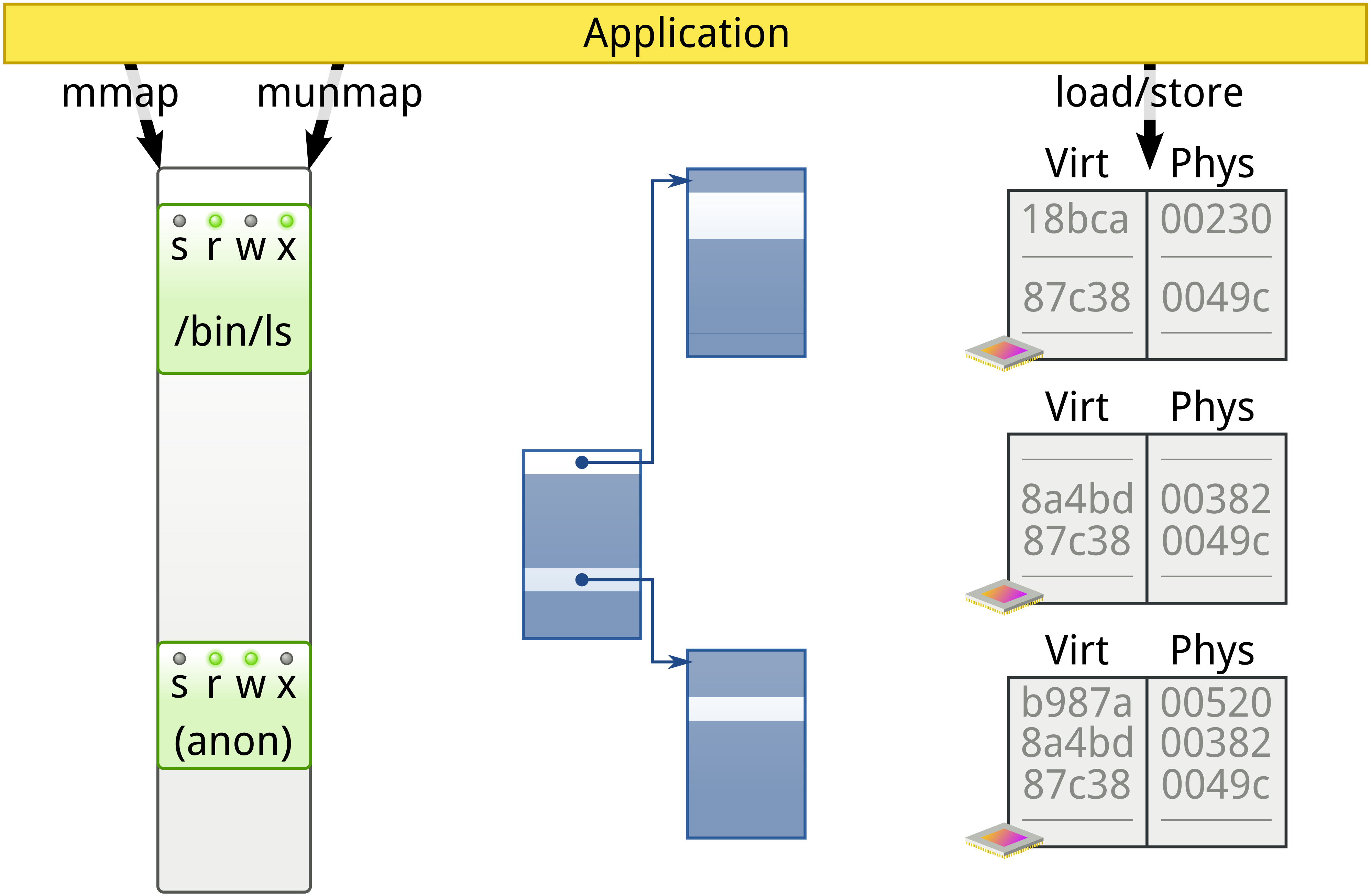
Structure of a VM system



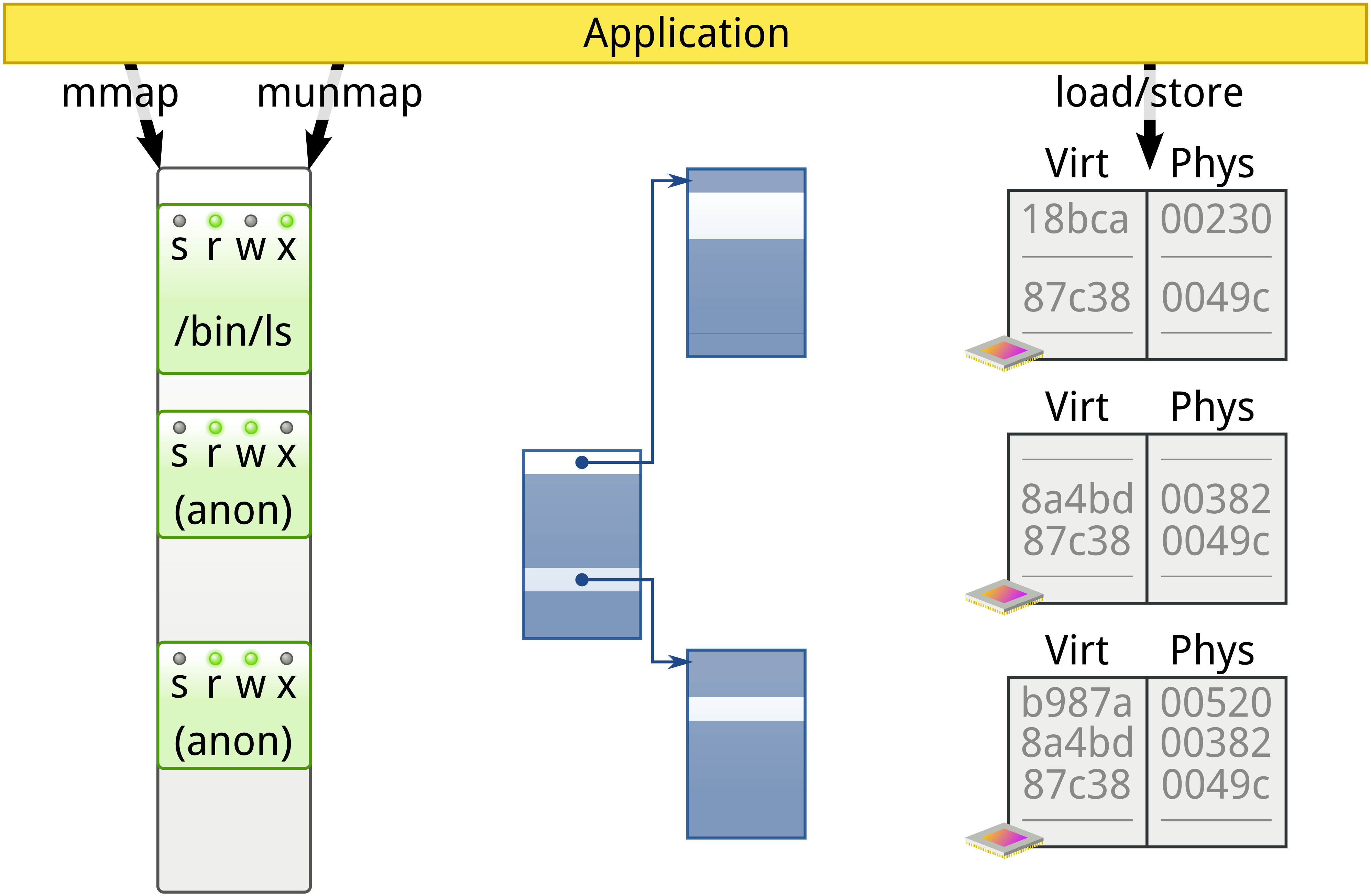
Structure of a VM system



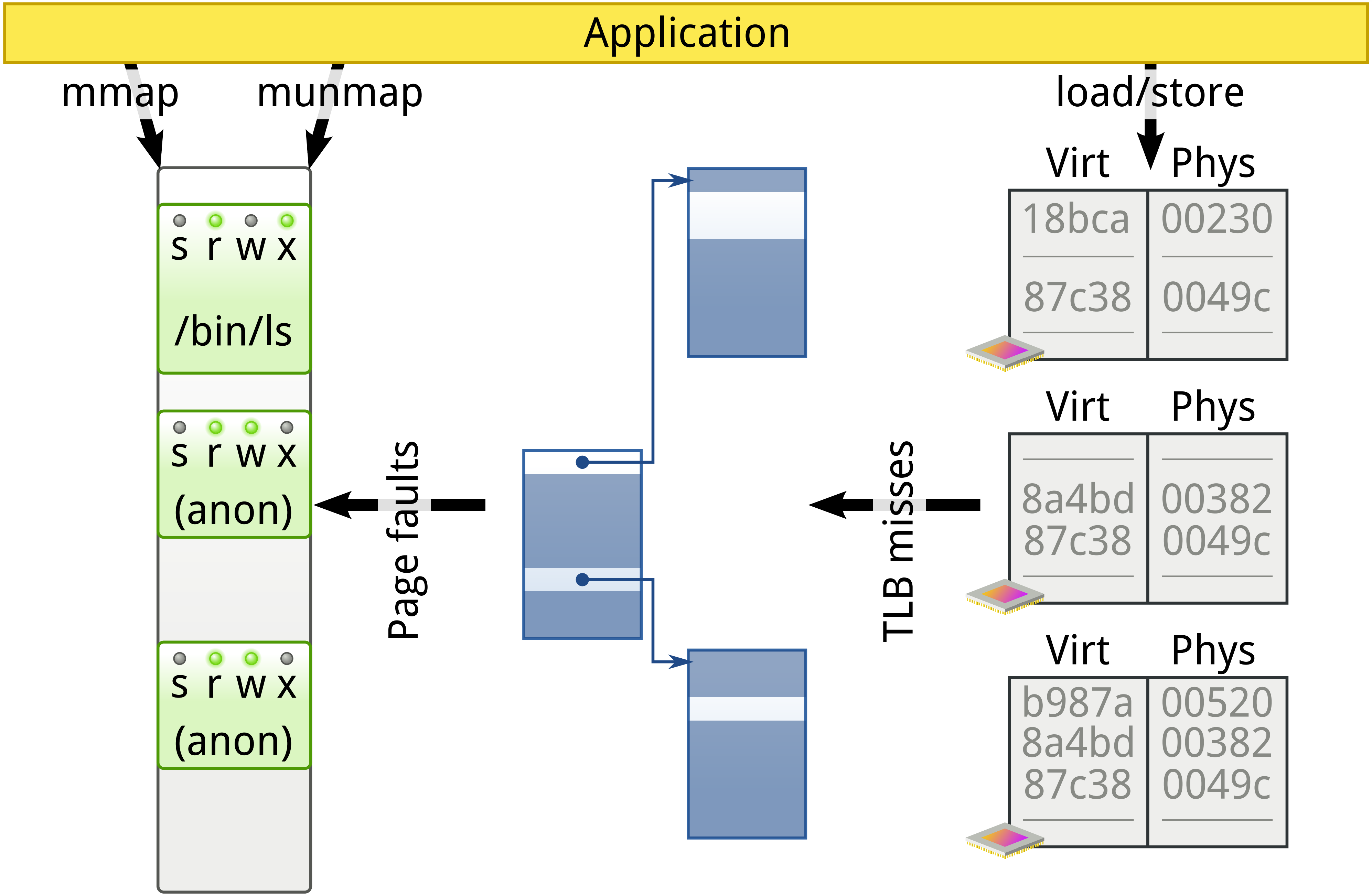
Structure of a VM system



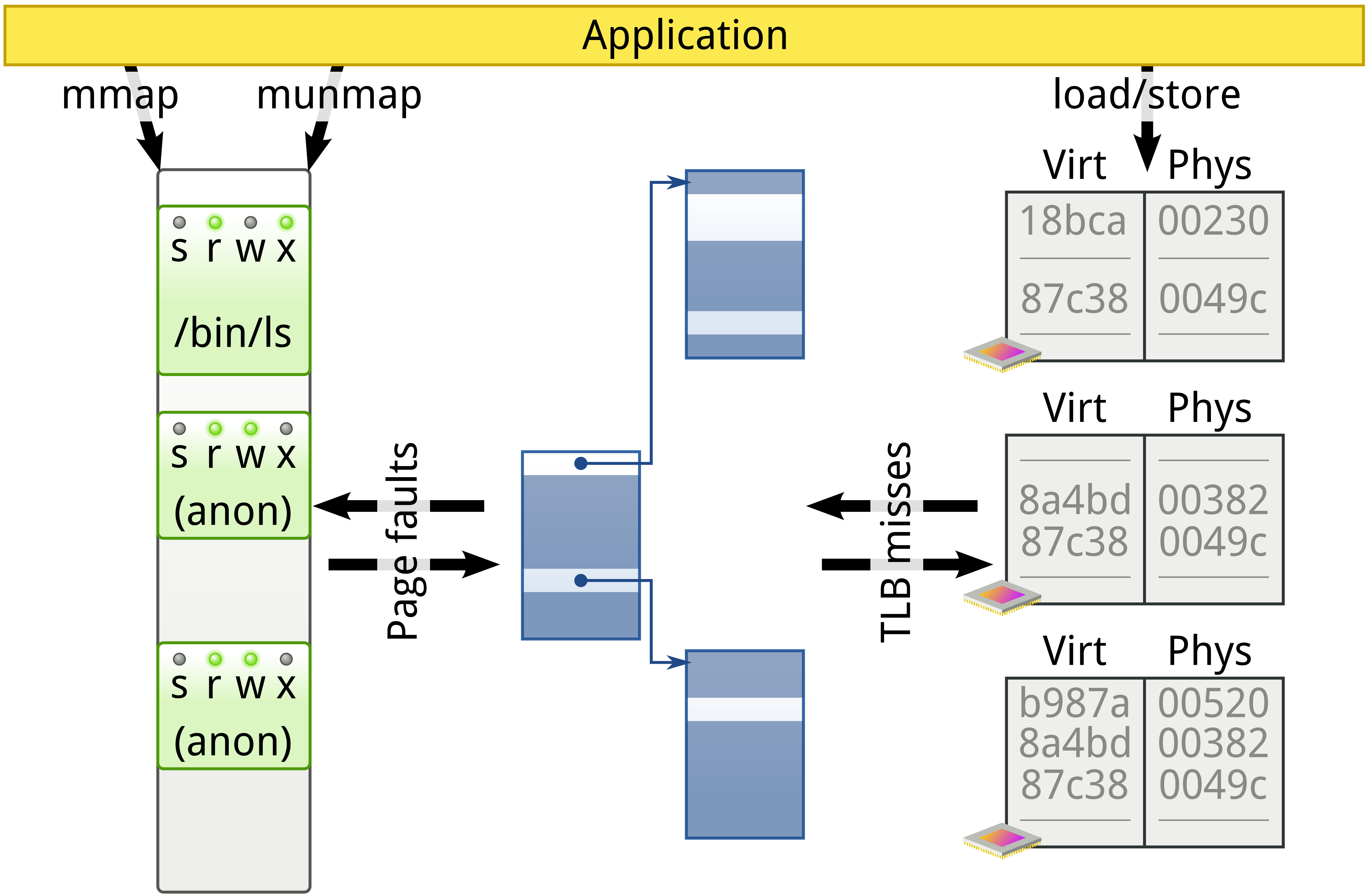
Structure of a VM system



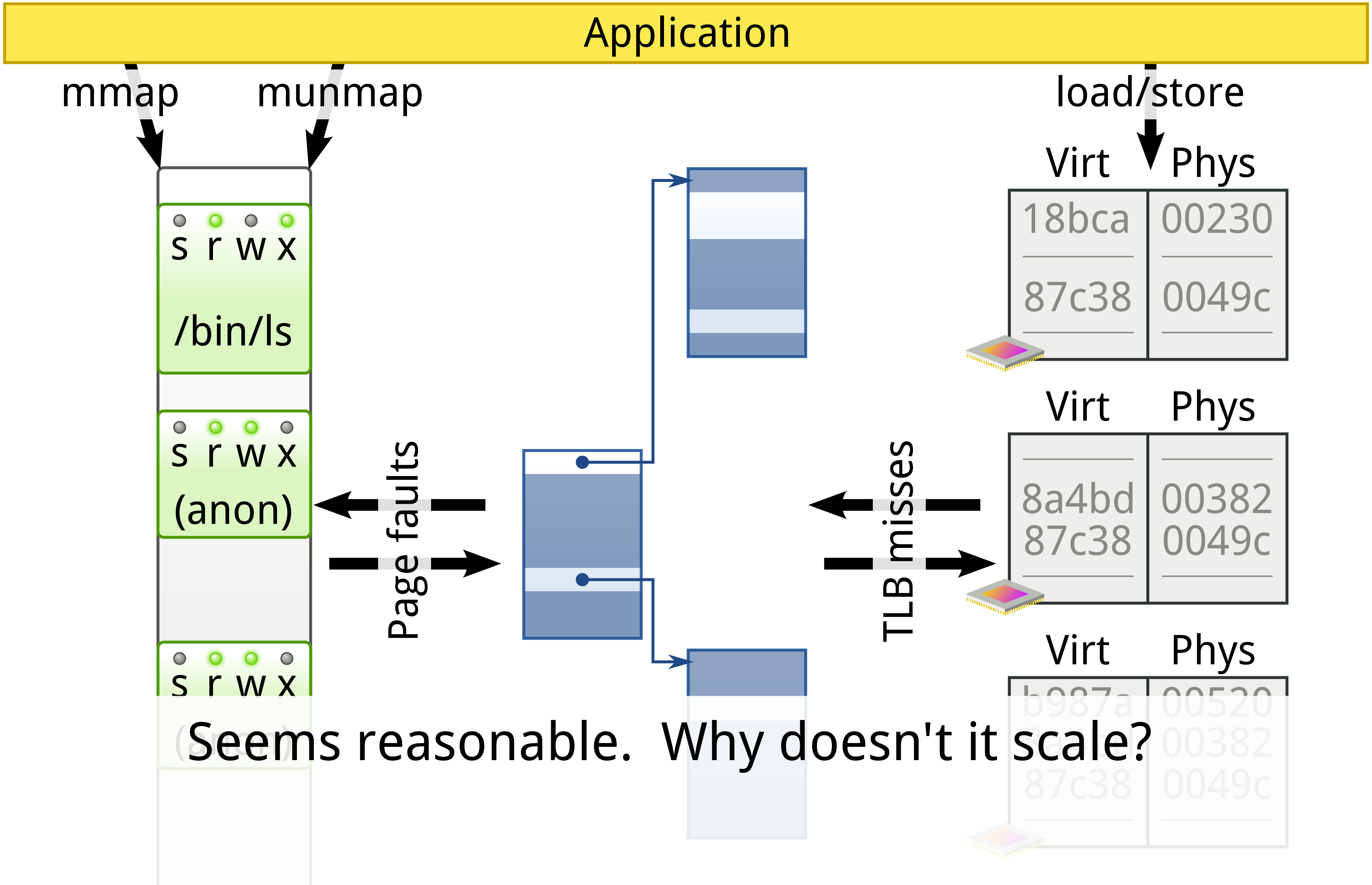
Structure of a VM system



Structure of a VM system

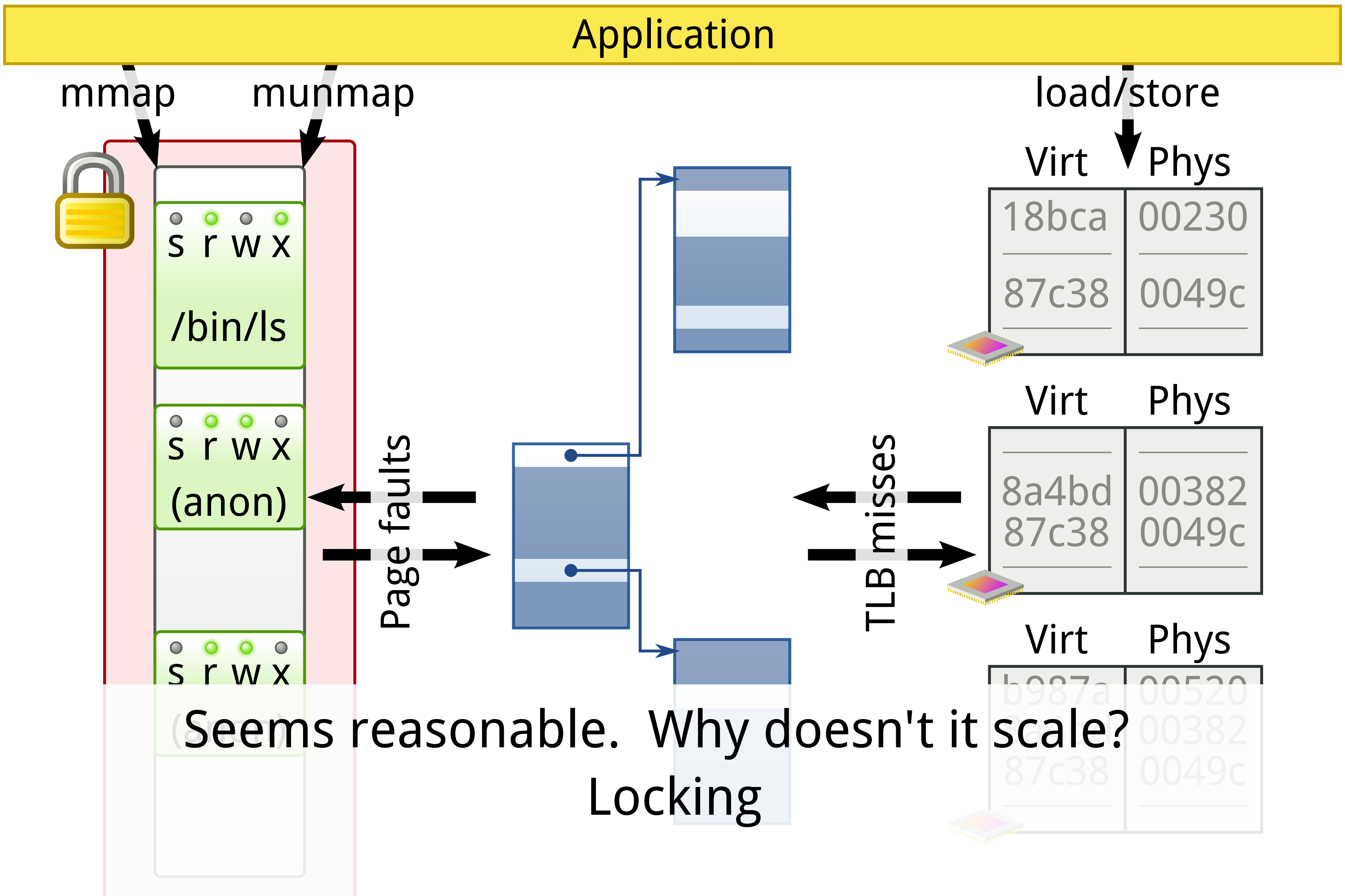


Structure of a VM system



Seems reasonable. Why doesn't it scale?

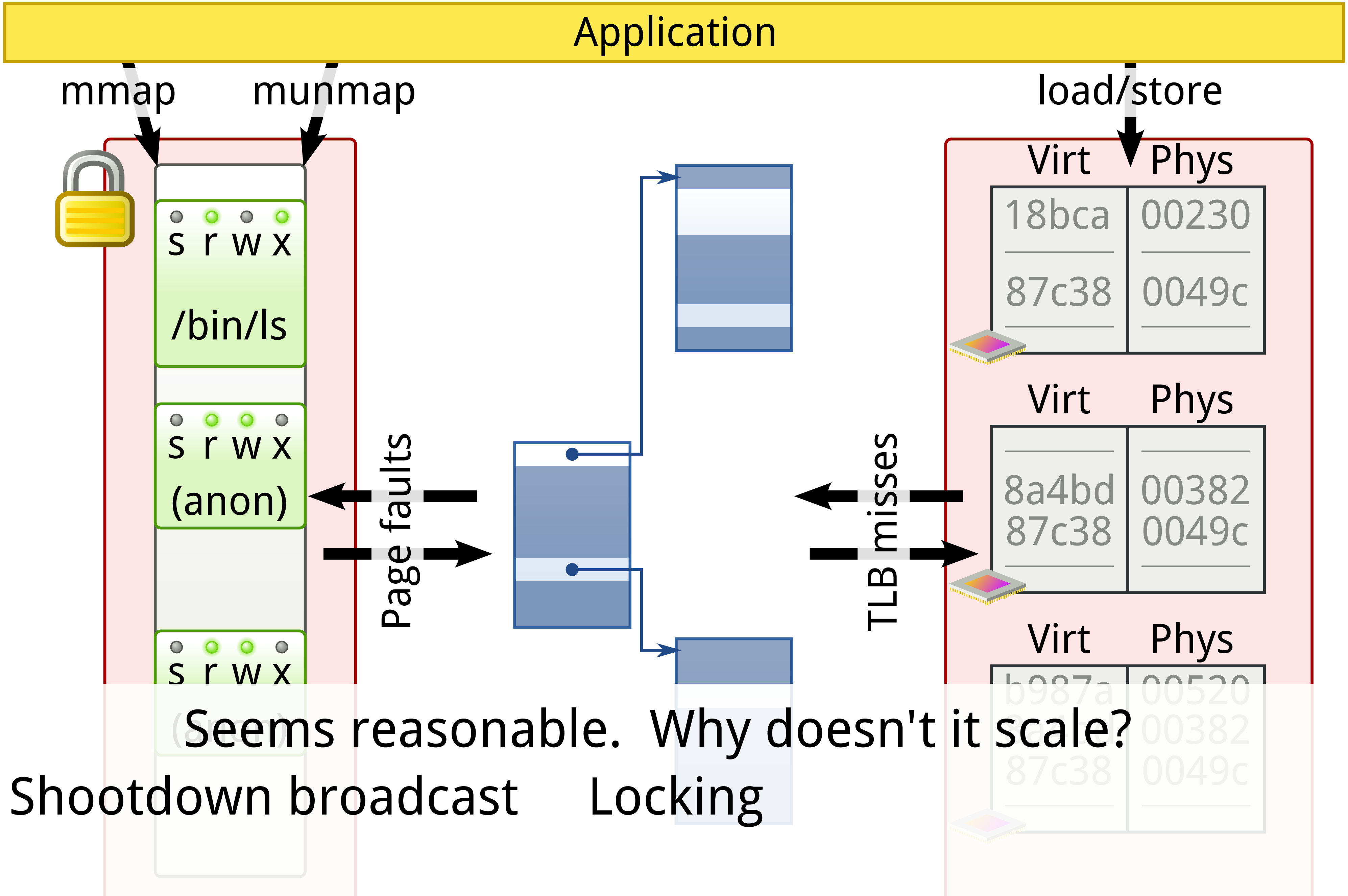
Structure of a VM system



Seems reasonable. Why doesn't it scale?

Locking

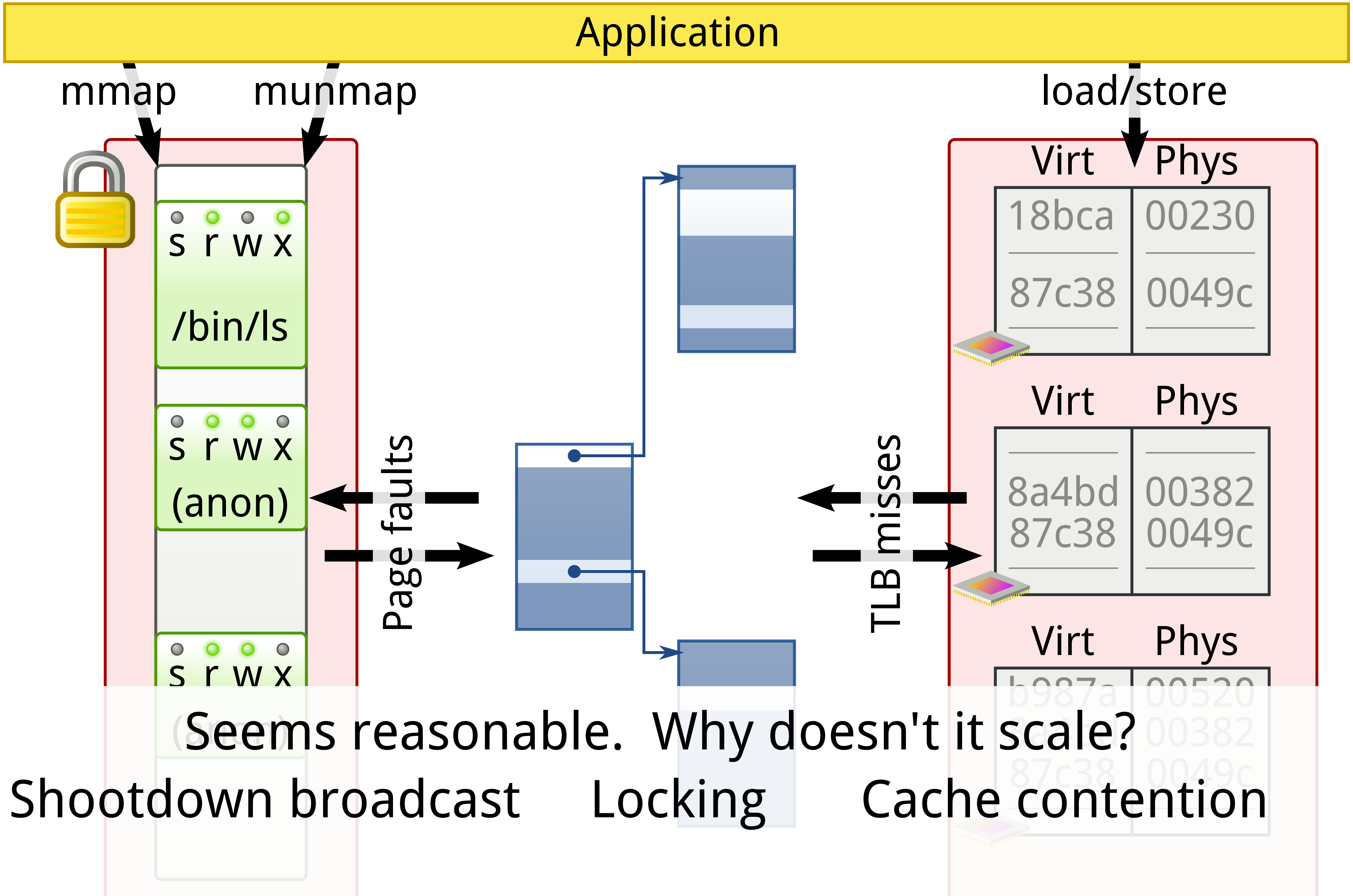
Structure of a VM system



Seems reasonable. Why doesn't it scale?

Shootdown broadcast   Locking

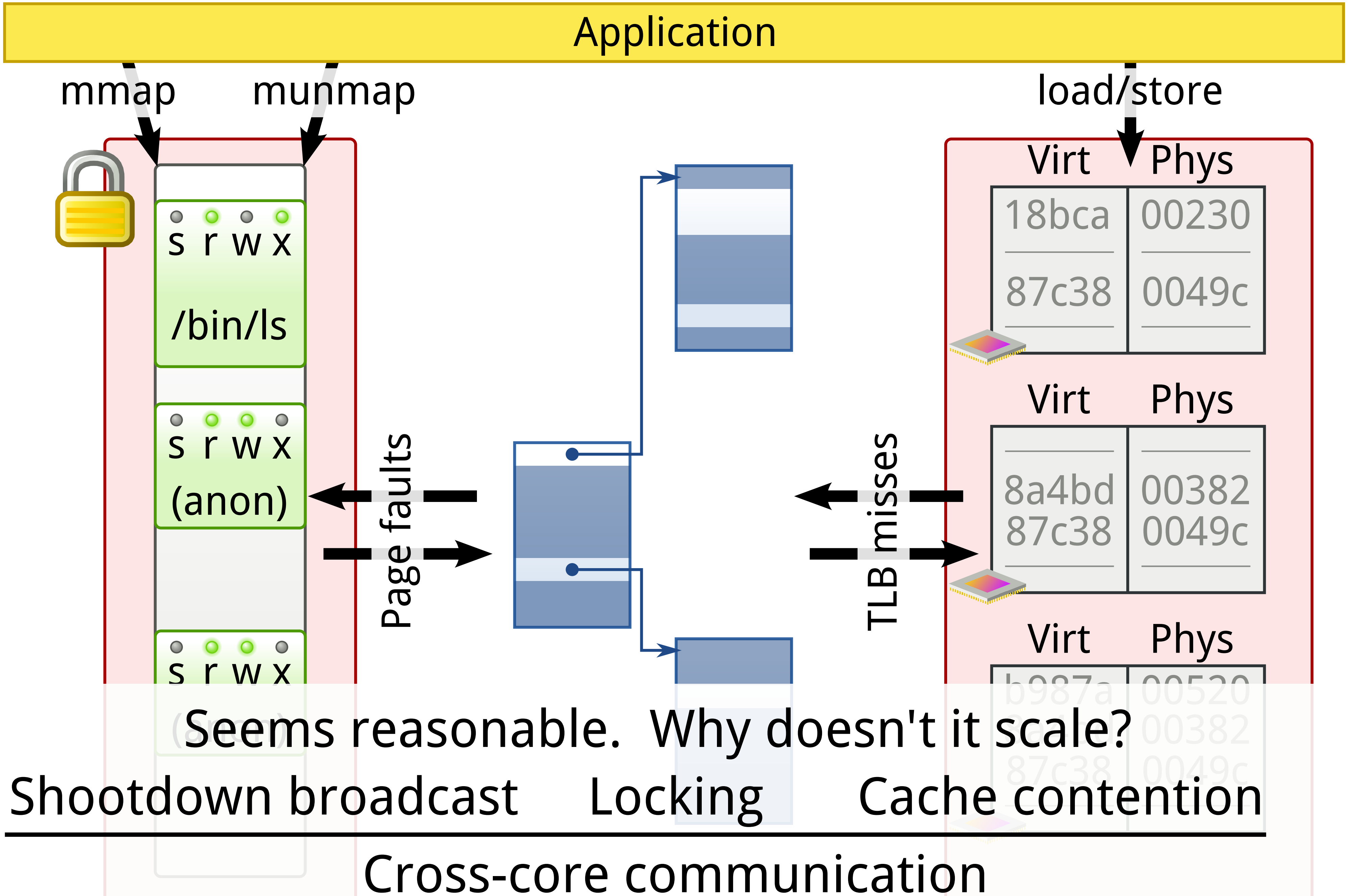
Structure of a VM system



Seems reasonable. Why doesn't it scale?

Shootdown broadcast Locking Cache contention

Structure of a VM system



Seems reasonable. Why doesn't it scale?

Shootdown broadcast Locking Cache contention

Cross-core communication

This talk: RadixVM

To achieve perfectly scalable non-overlapping operations, we eliminate communication between such operations.

Concurrent memory map representation

Method of targeting TLB shootdowns

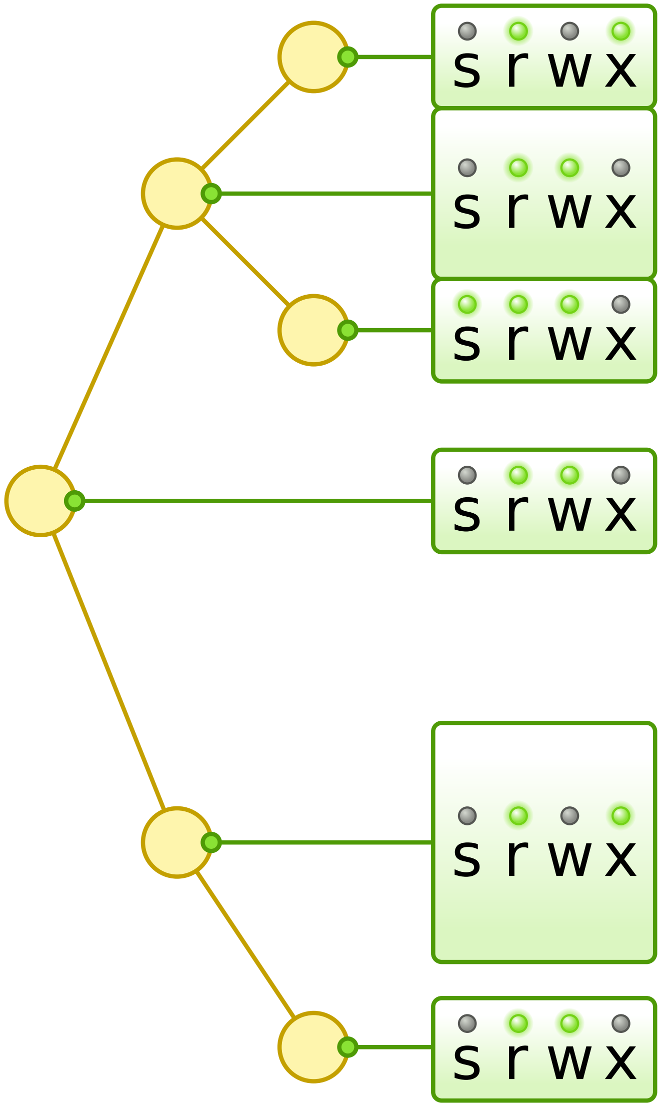
Scalable, space-efficient reference counting

Metadata management

Need to store OS-level memory mapping metadata

Metadata management

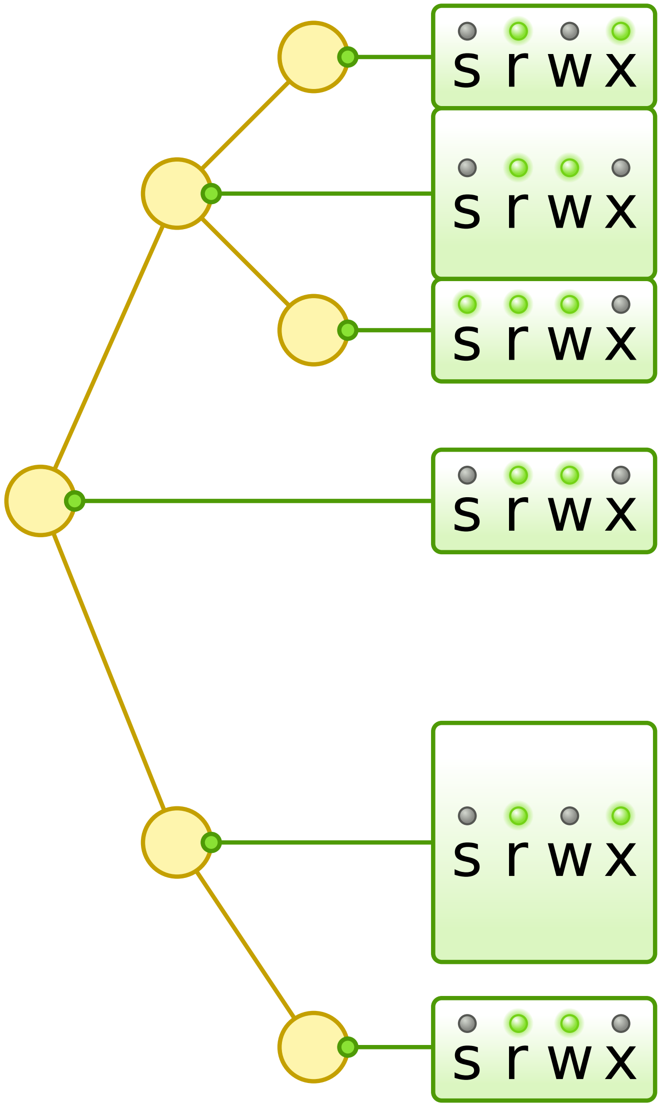
Need to store OS-level memory mapping metadata



Popular operating systems use a balanced tree of region objects.

Metadata management

Need to store OS-level memory mapping metadata

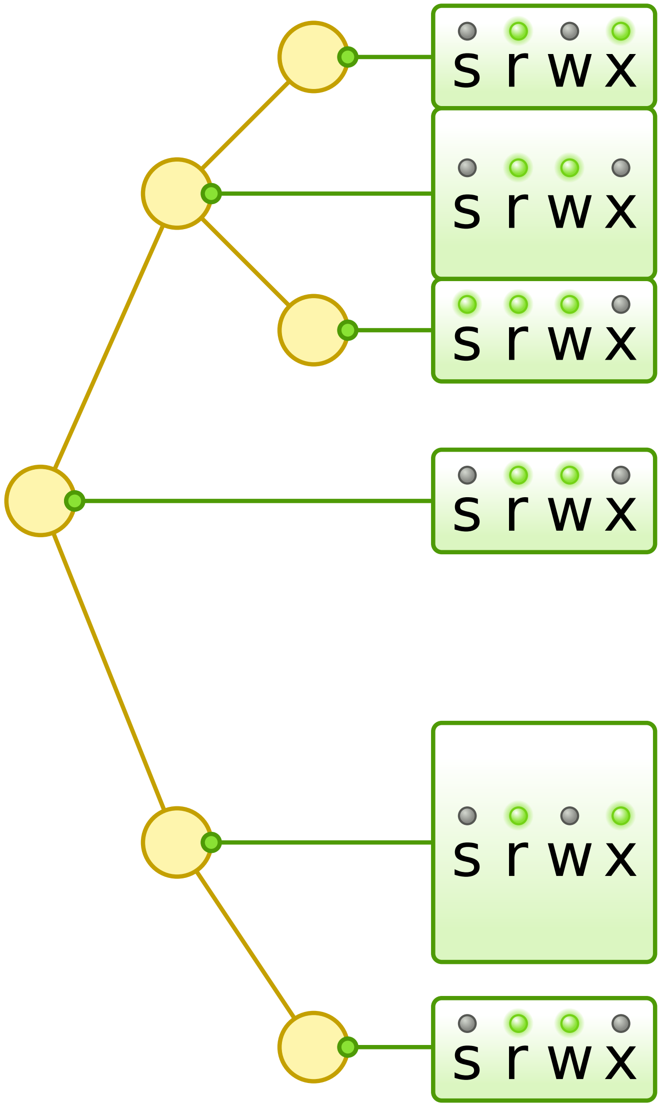


Popular operating systems use a balanced tree of region objects.

→ Memory-efficient

Metadata management

Need to store OS-level memory mapping metadata



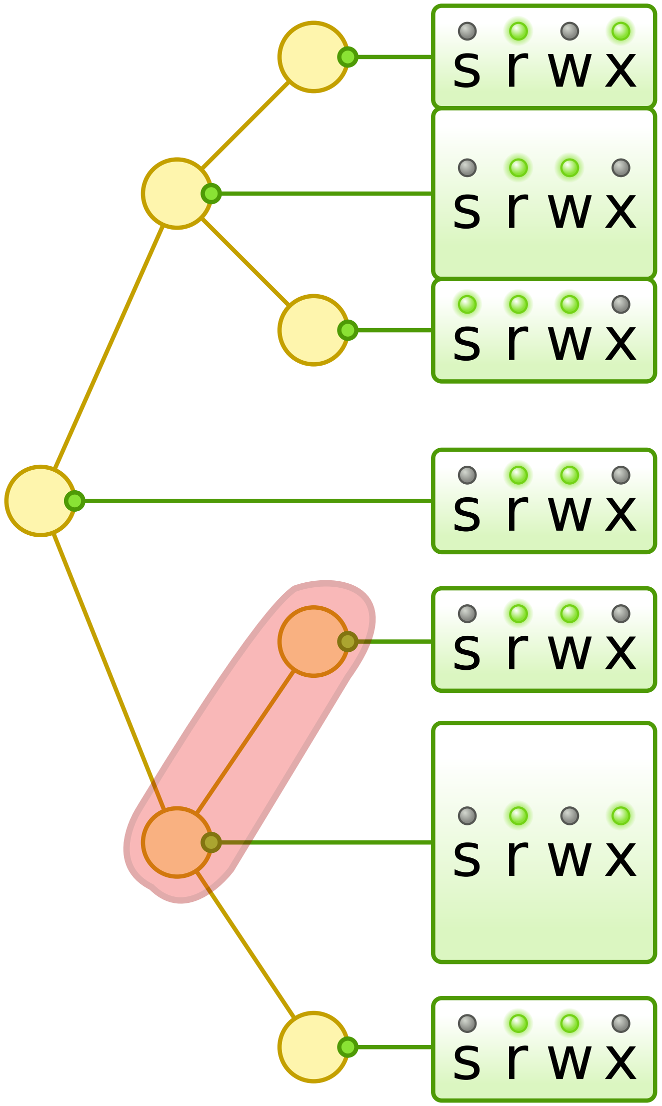
Popular operating systems use a balanced tree of region objects.

Unnecessary communication

Memory-efficient

Metadata management

Need to store OS-level memory mapping metadata



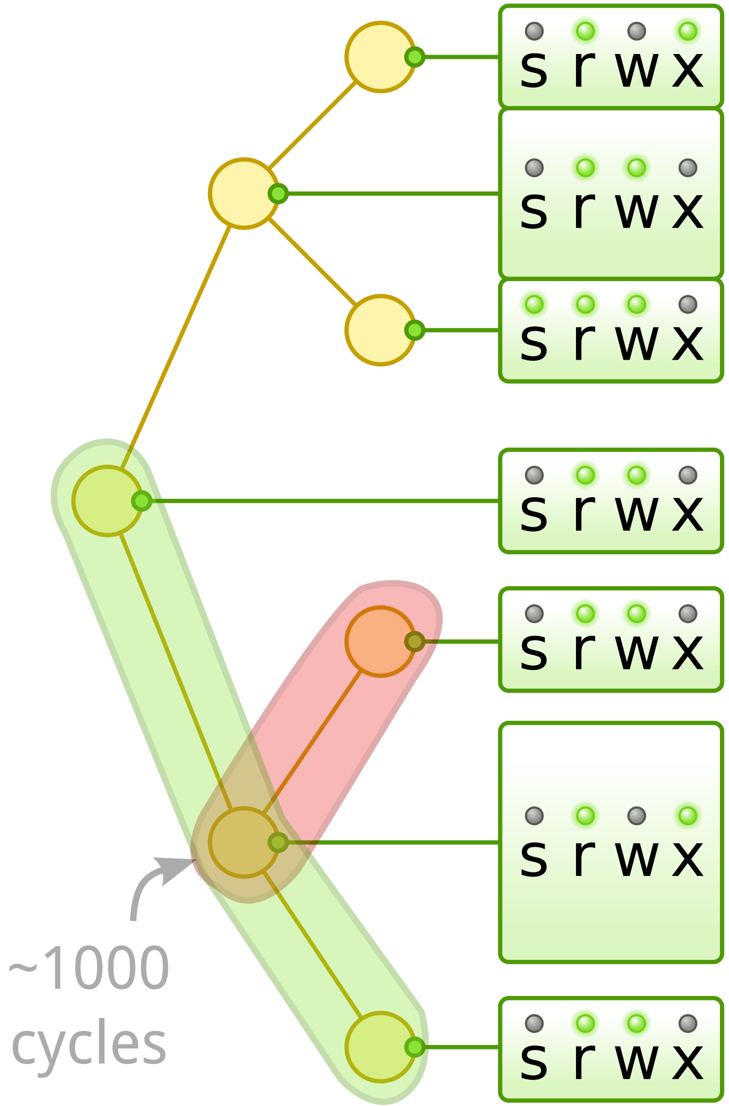
Popular operating systems use a balanced tree of region objects.

Unnecessary communication

Memory-efficient

Metadata management

Need to store OS-level memory mapping metadata



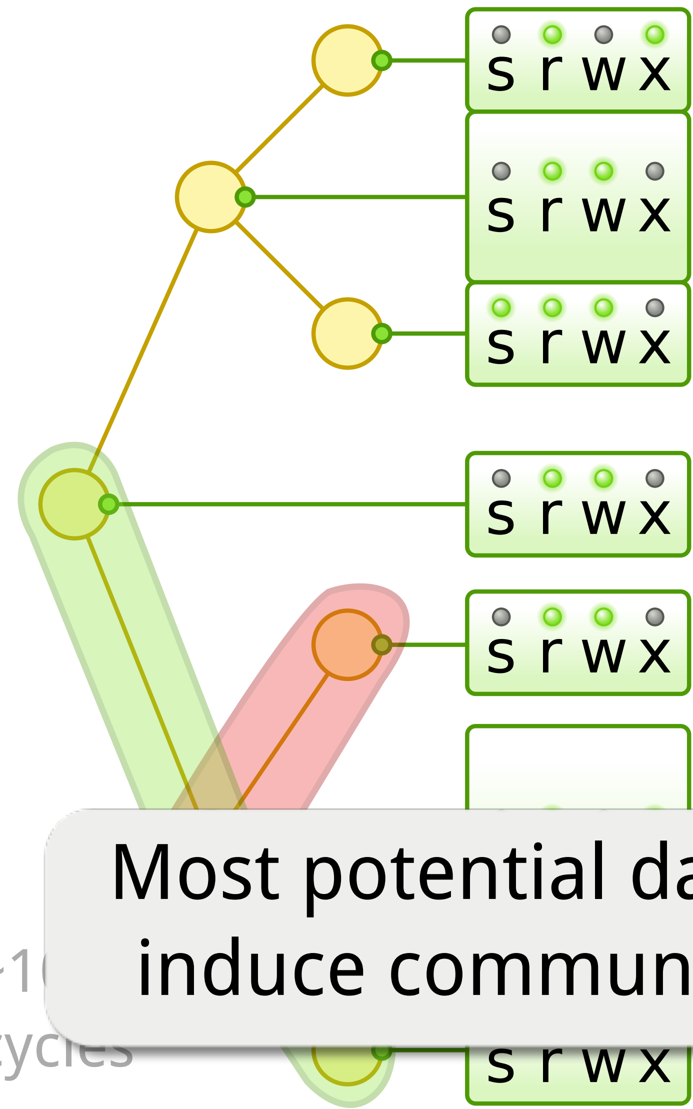
Popular operating systems use a balanced tree of region objects.

Unnecessary communication

Memory-efficient

Metadata management

Need to store OS-level memory mapping metadata



Popular operating systems use a balanced tree of region objects.

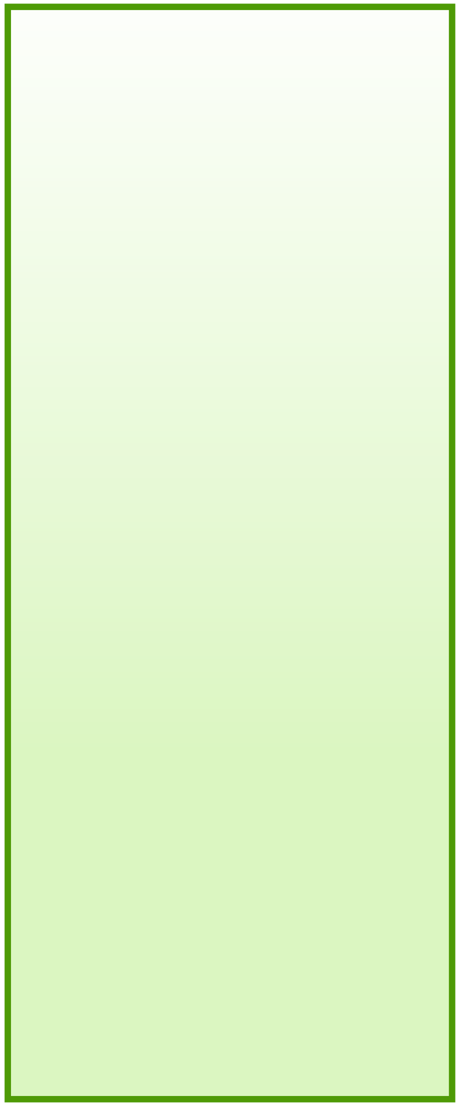
Unnecessary communication

Memory-efficient

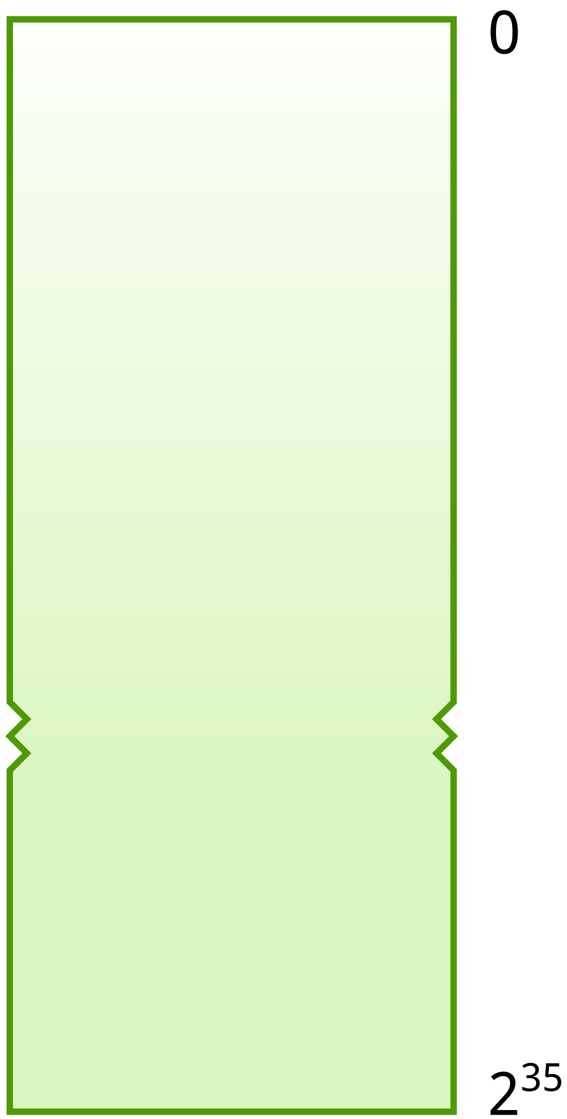
Most potential data structures (skip lists, B-trees, etc.) induce communication between disjoint operations.

Array-based memory map

Array-based memory map



Array-based memory map

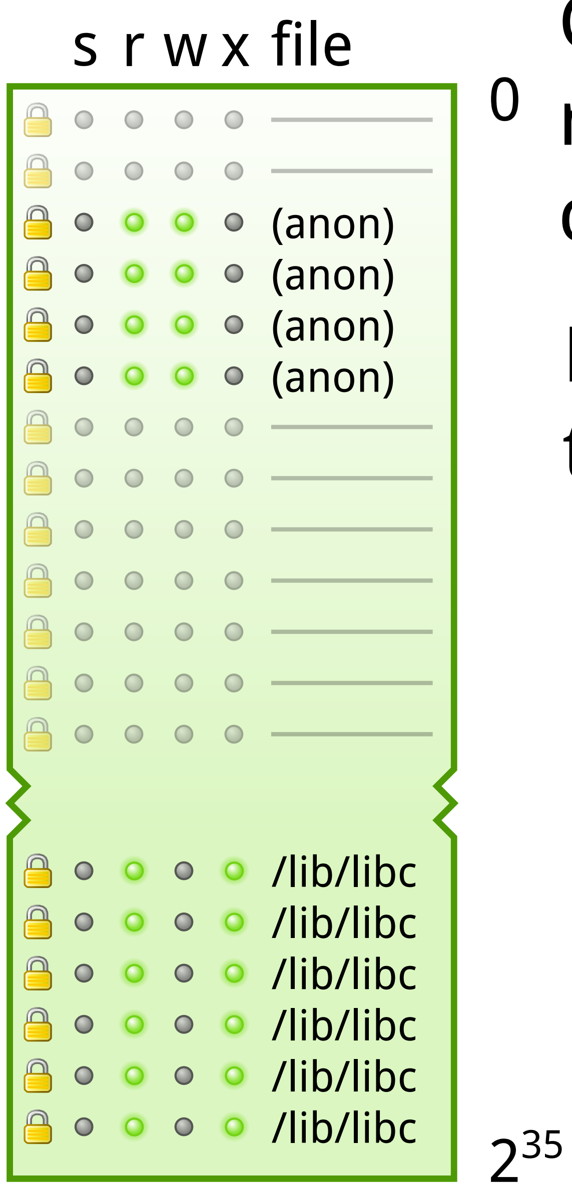


Array-based memory map

Array-based memory map

Good: Operations on non-overlapping regions are concurrent and induce no communication.

Array-based memory map



Good: Operations on non-overlapping regions are concurrent and induce no communication.

Bad: Space use is obscene, time is proportional to region size

Array-based memory map

Good: Operations on non-overlapping regions are concurrent and induce no communication.

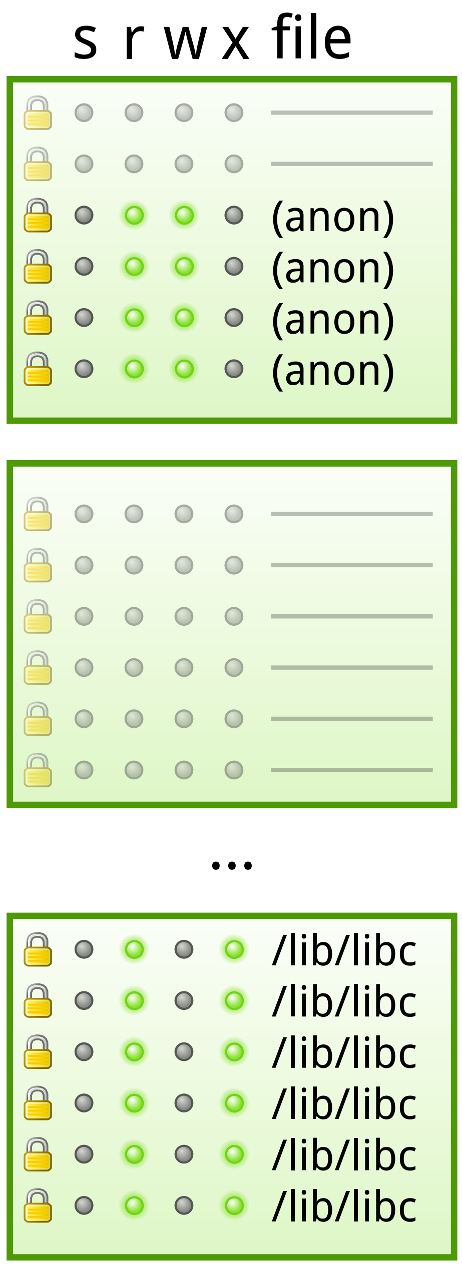
Bad: Space use is obscene,
time is proportional to region size

How can we achieve good concurrency while keeping space and time under control?

Radix tree

Solution: Range-oriented radix tree

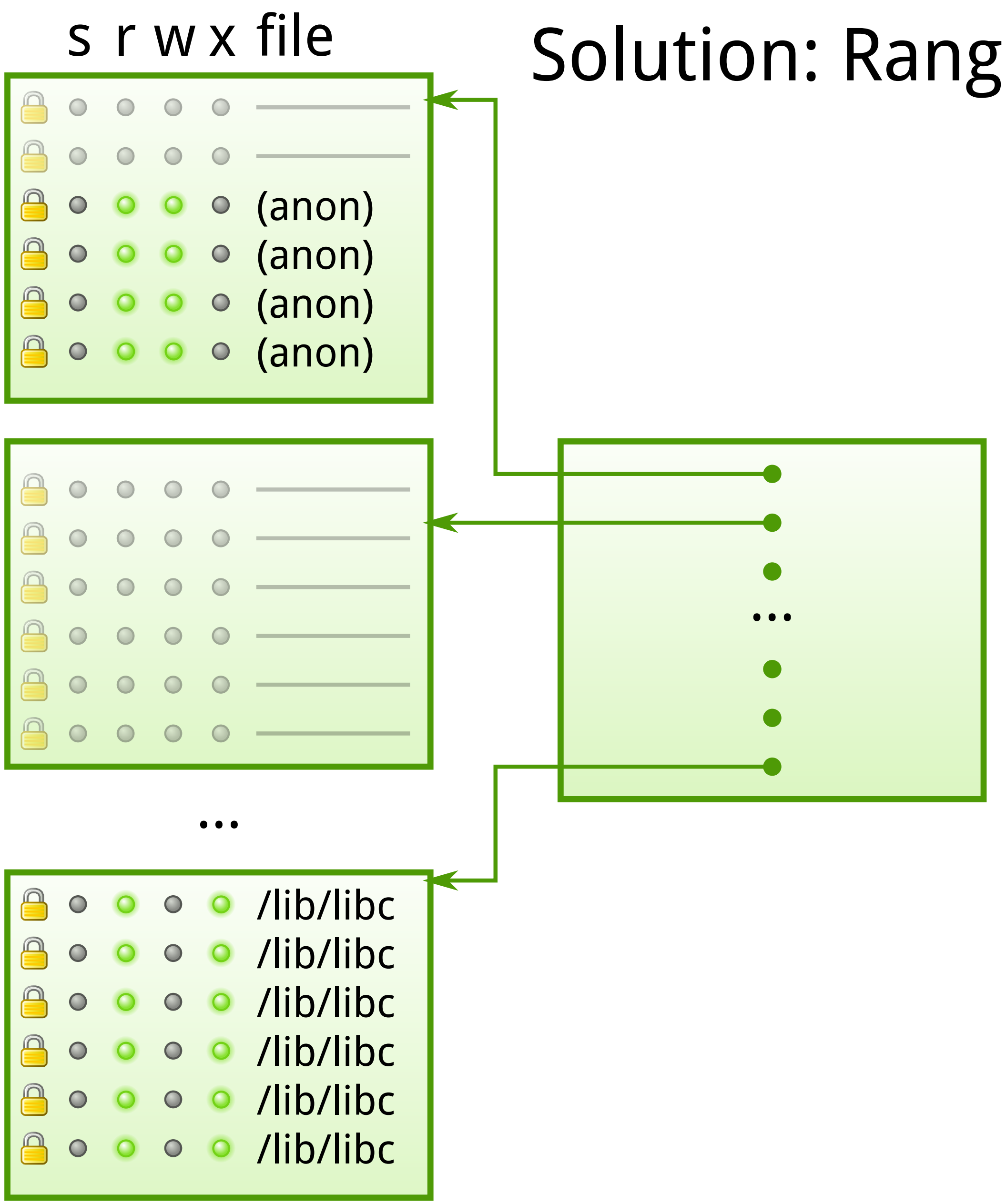
Radix tree



Solution: Range-oriented radix tree

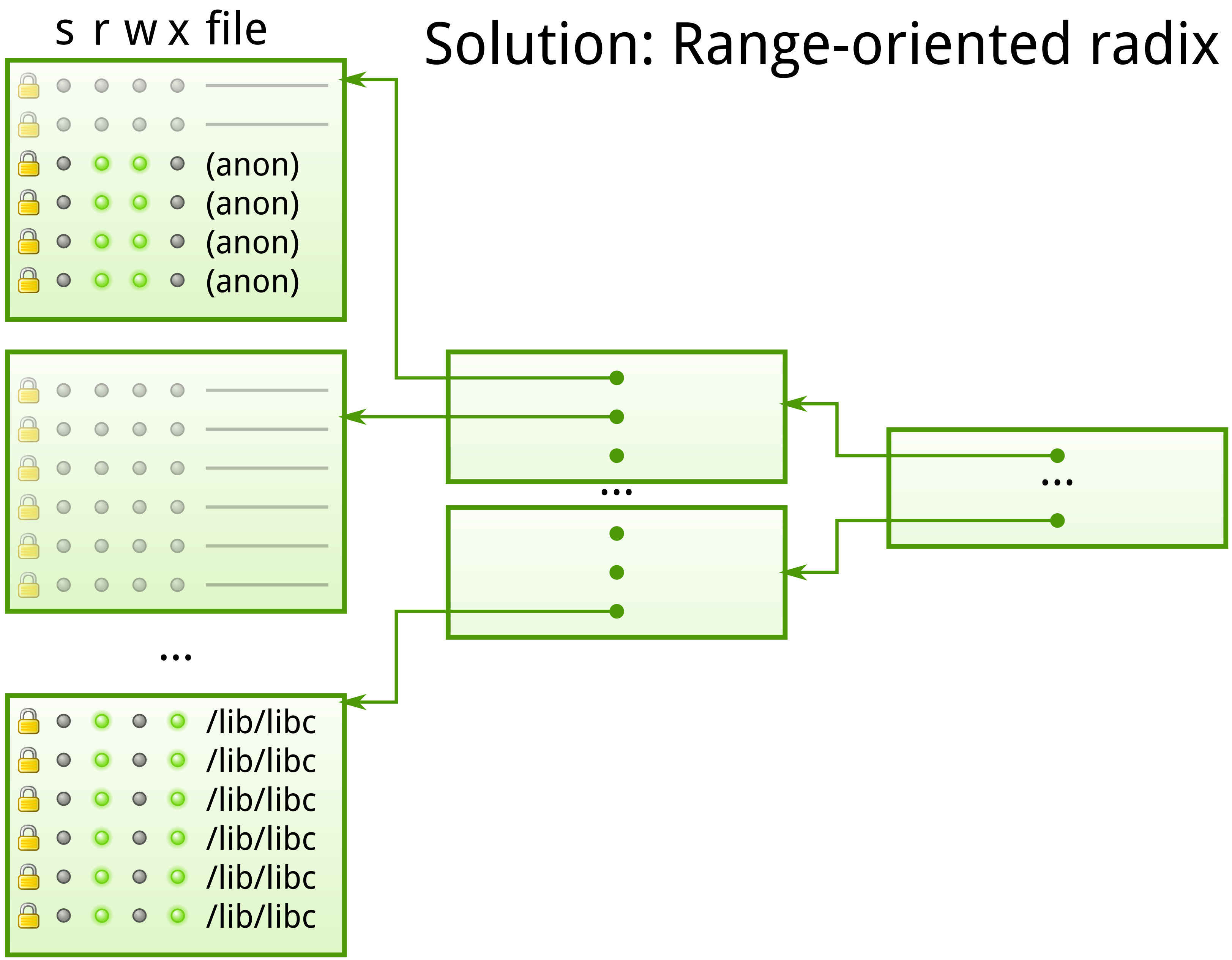
Radix tree

Solution: Range-oriented radix tree

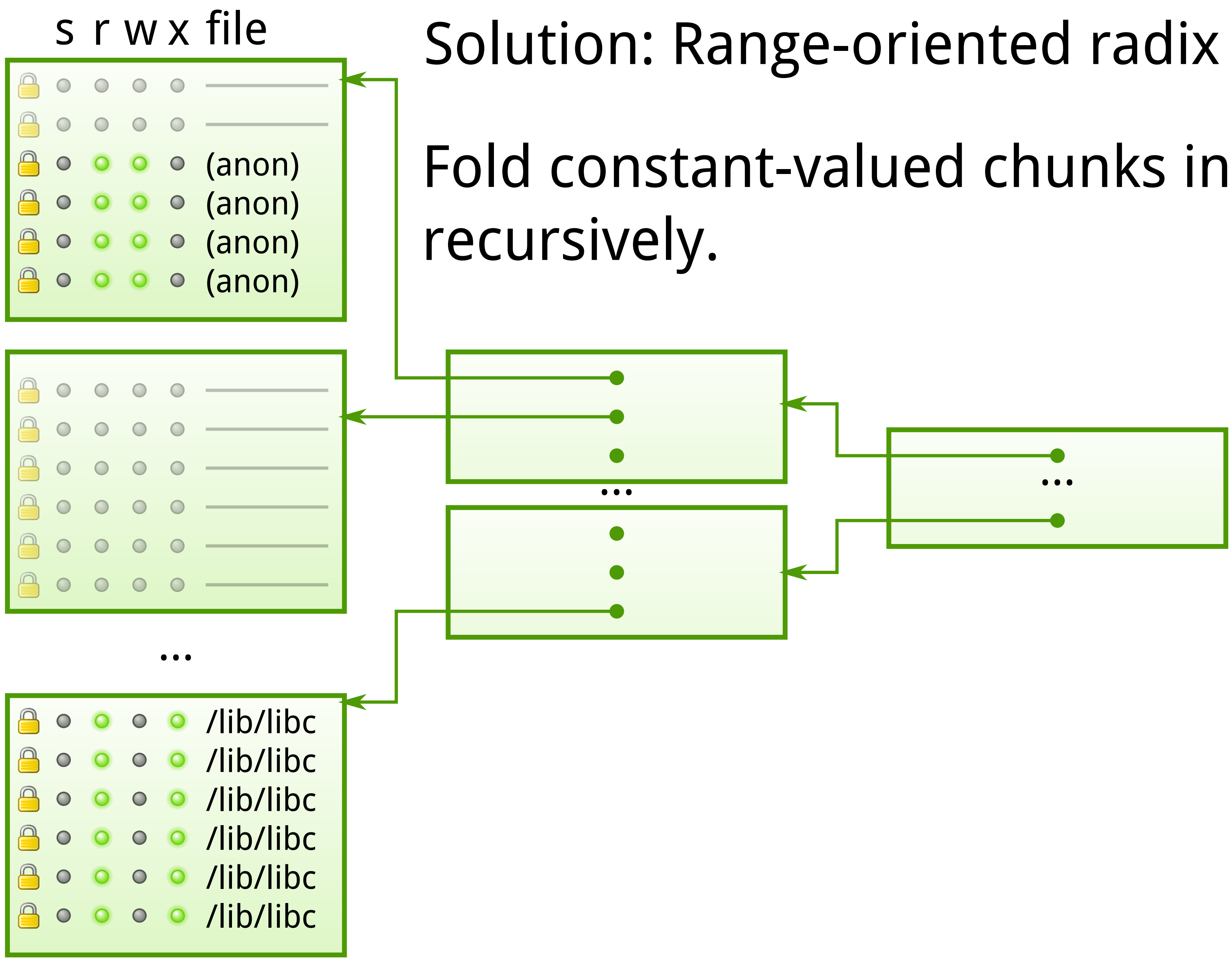


Radix tree

Solution: Range-oriented radix tree



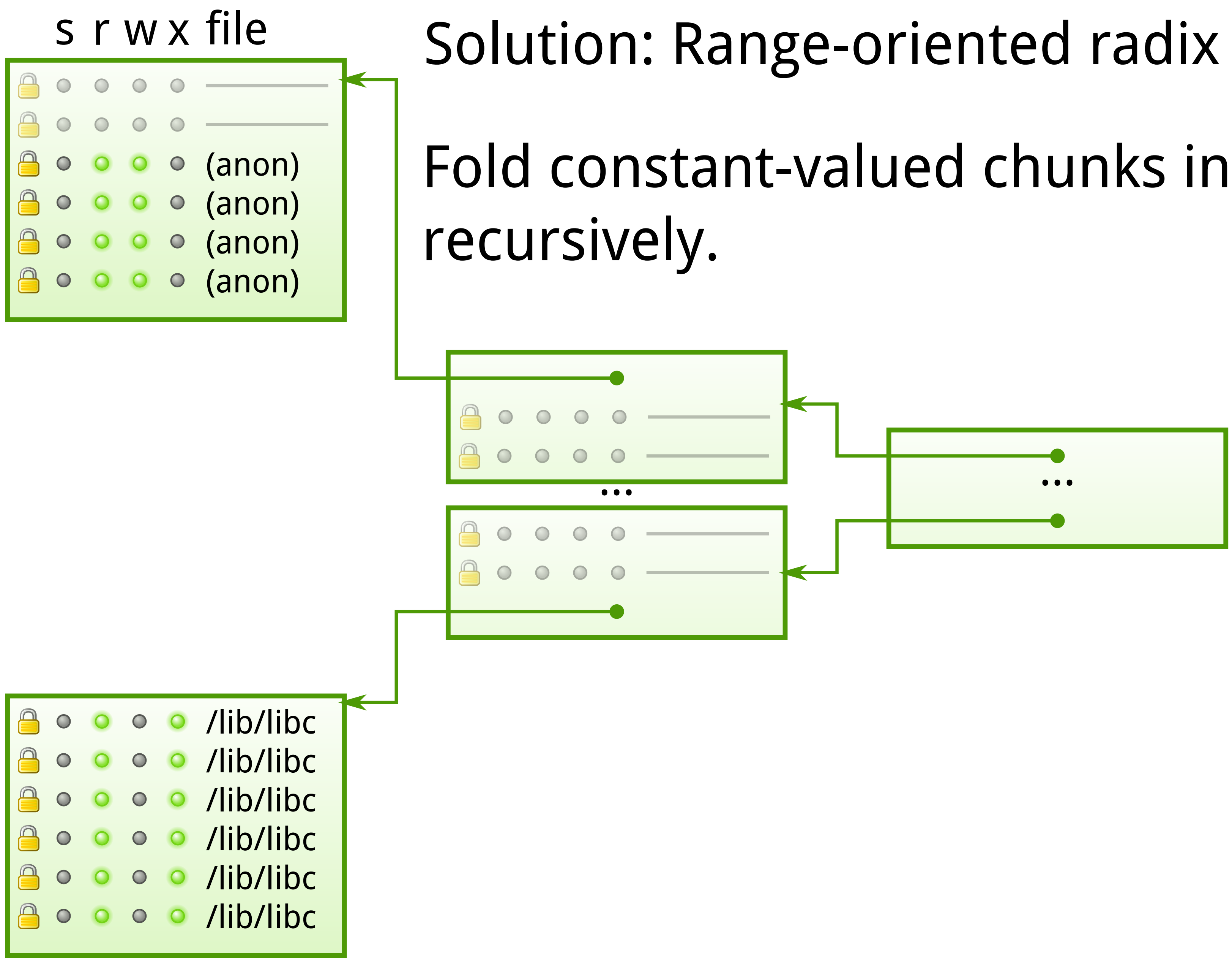
Radix tree



Solution: Range-oriented radix tree

Fold constant-valued chunks into parent, recursively.

Radix tree



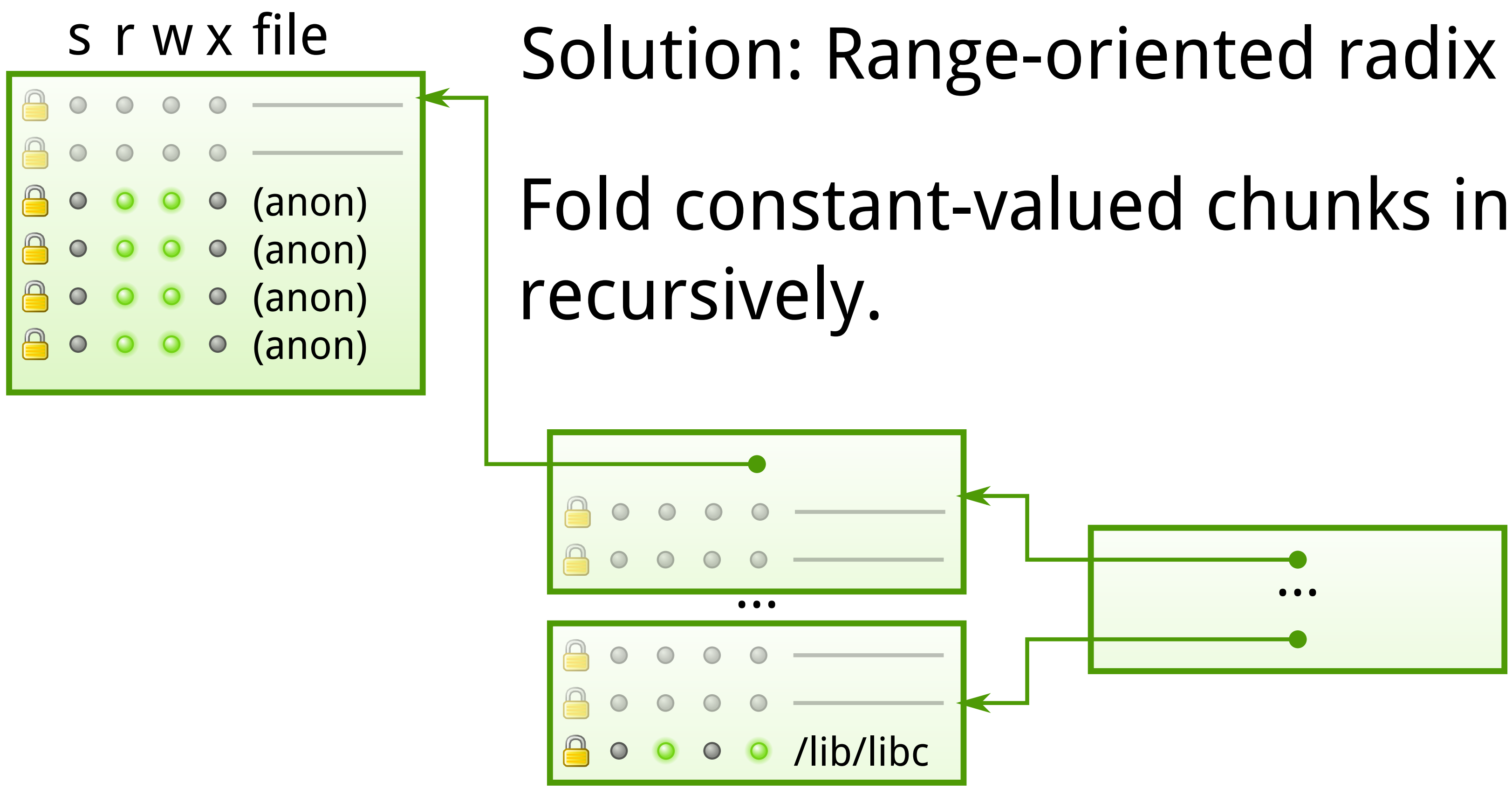
Solution: Range-oriented radix tree

Fold constant-valued chunks into parent, recursively.

Radix tree

Solution: Range-oriented radix tree

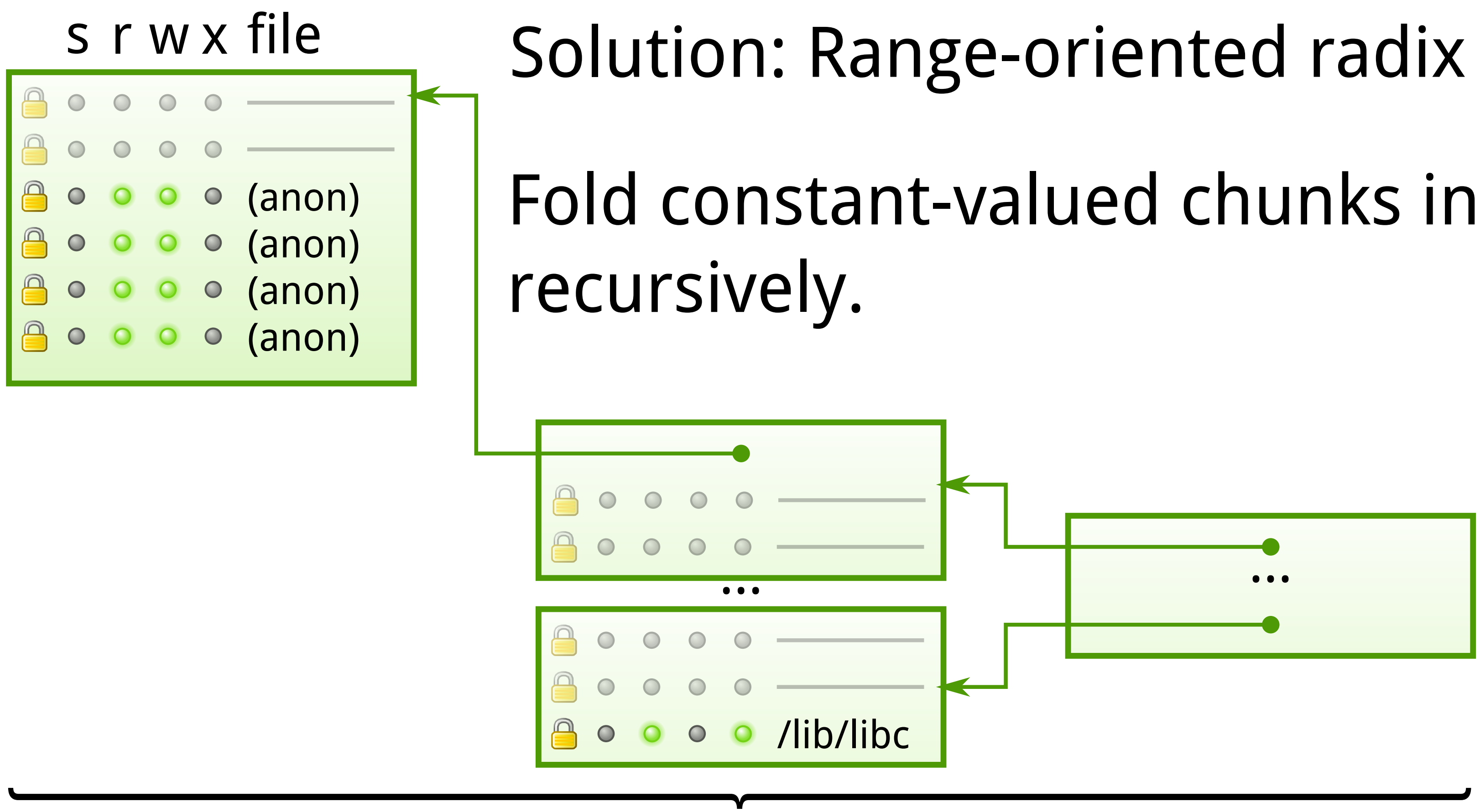
Fold constant-valued chunks into parent, recursively.



Radix tree

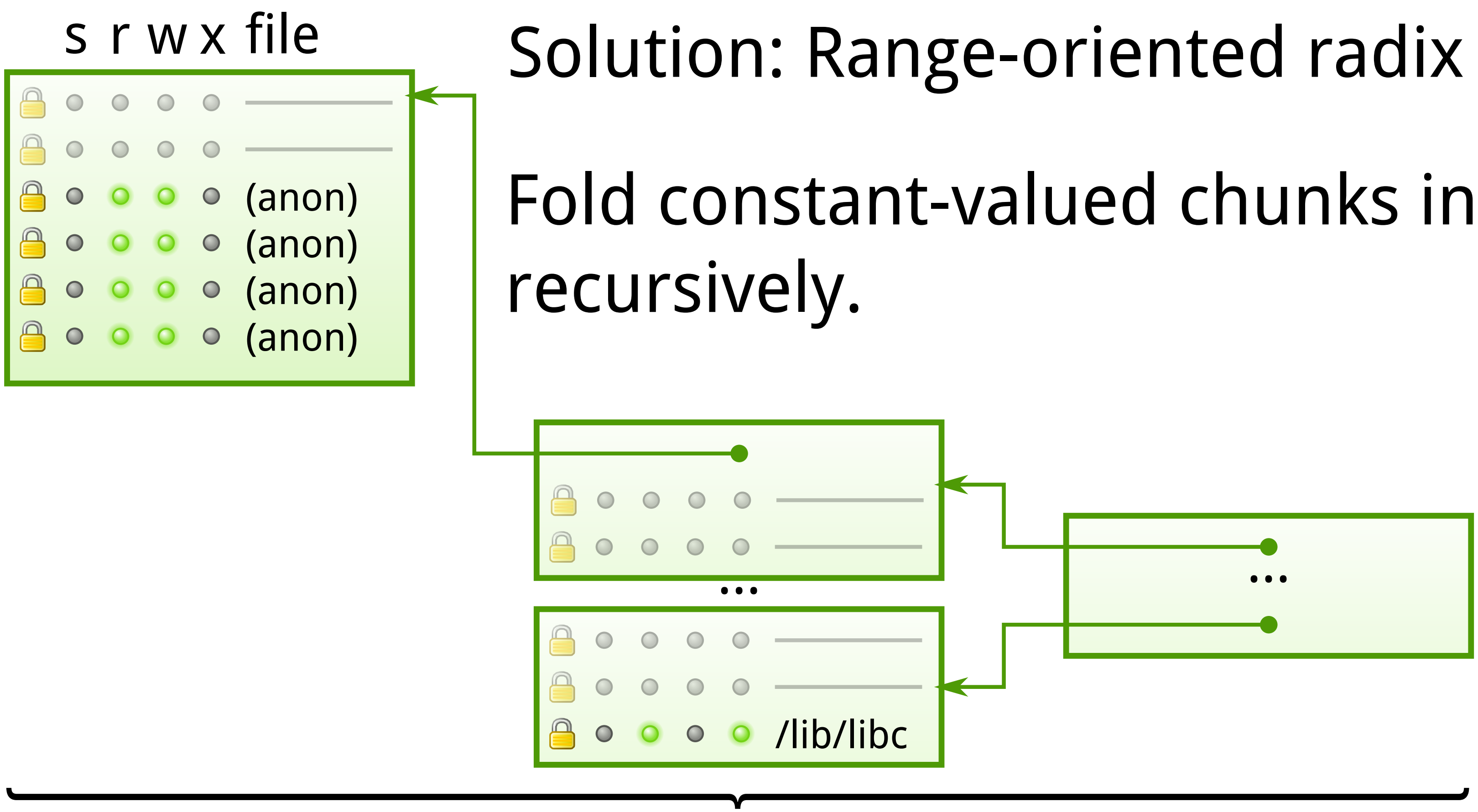
Solution: Range-oriented radix tree

Fold constant-valued chunks into parent, recursively.



2-3x the size of the balanced region tree

Radix tree



Solution: Range-oriented radix tree

Fold constant-valued chunks into parent, recursively.

2-3x the size of the balanced region tree

We can achieve array-like concurrency
with time and space similar to the balanced tree.

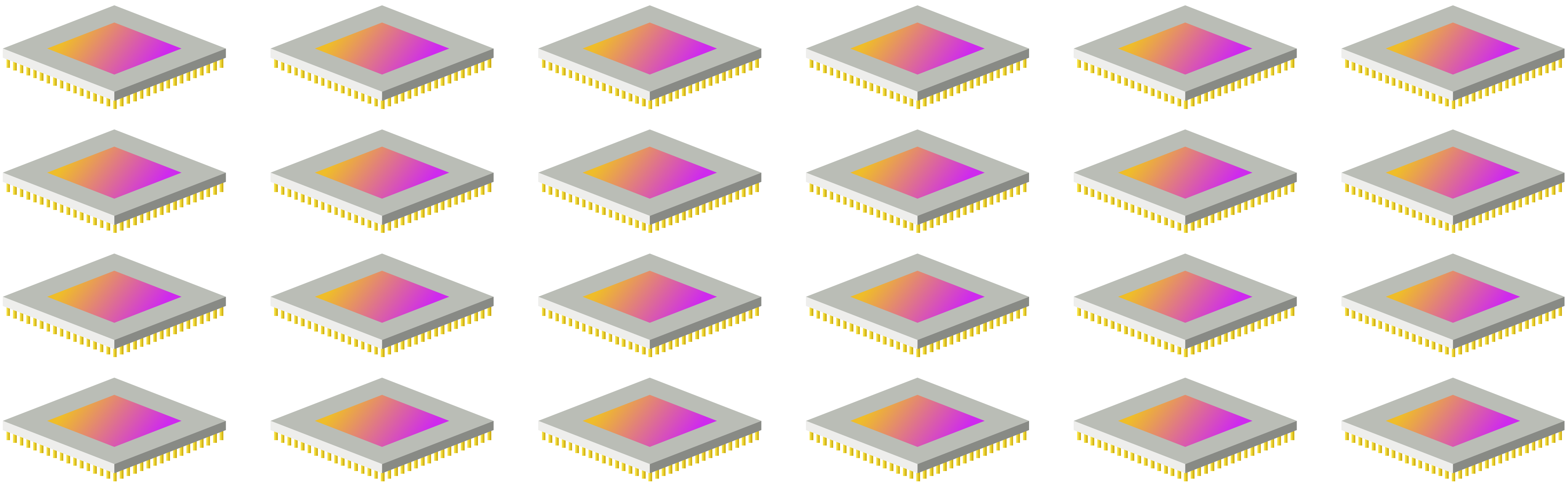
TLB shootdown

munmap must notify cores of changes to cached mappings

TLB shootdown

munmap must notify cores of changes to cached mappings

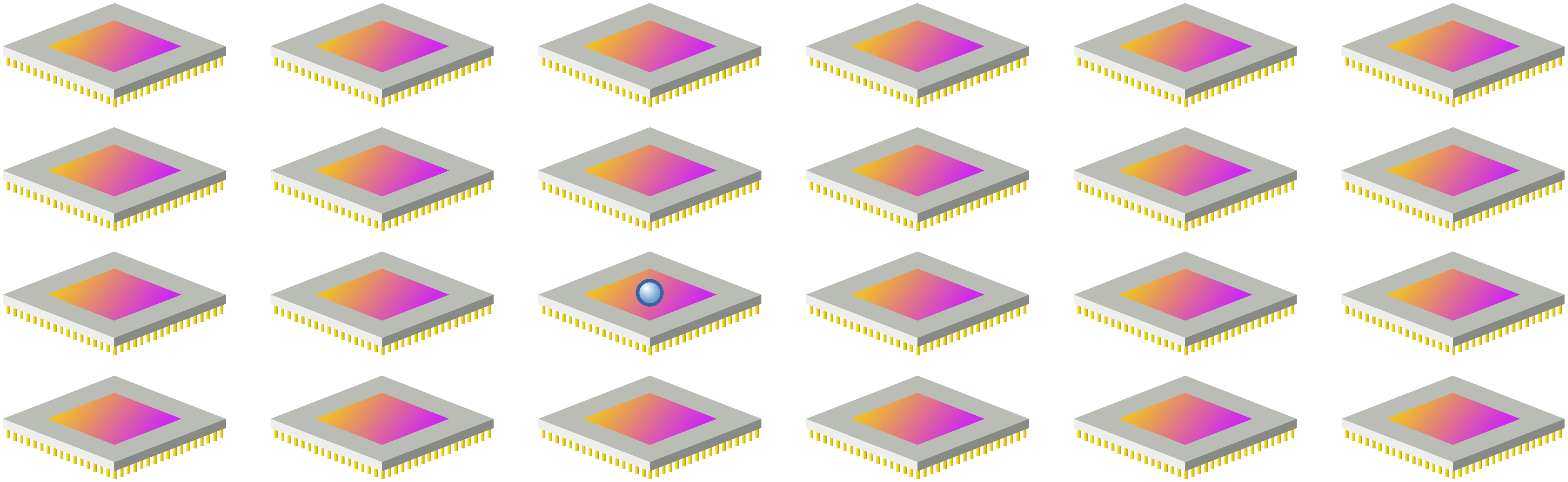
Which cores have a mapping cached? Who knows?!



TLB shootdown

munmap must notify cores of changes to cached mappings

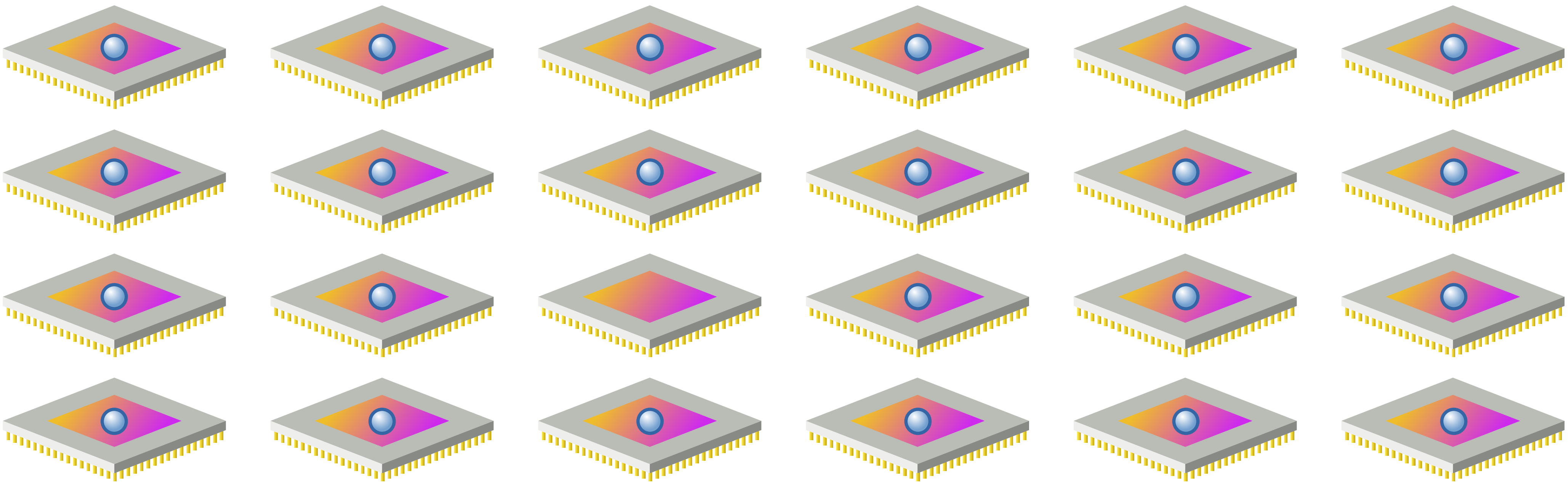
Which cores have a mapping cached? Who knows?!



TLB shootdown

munmap must notify cores of changes to cached mappings

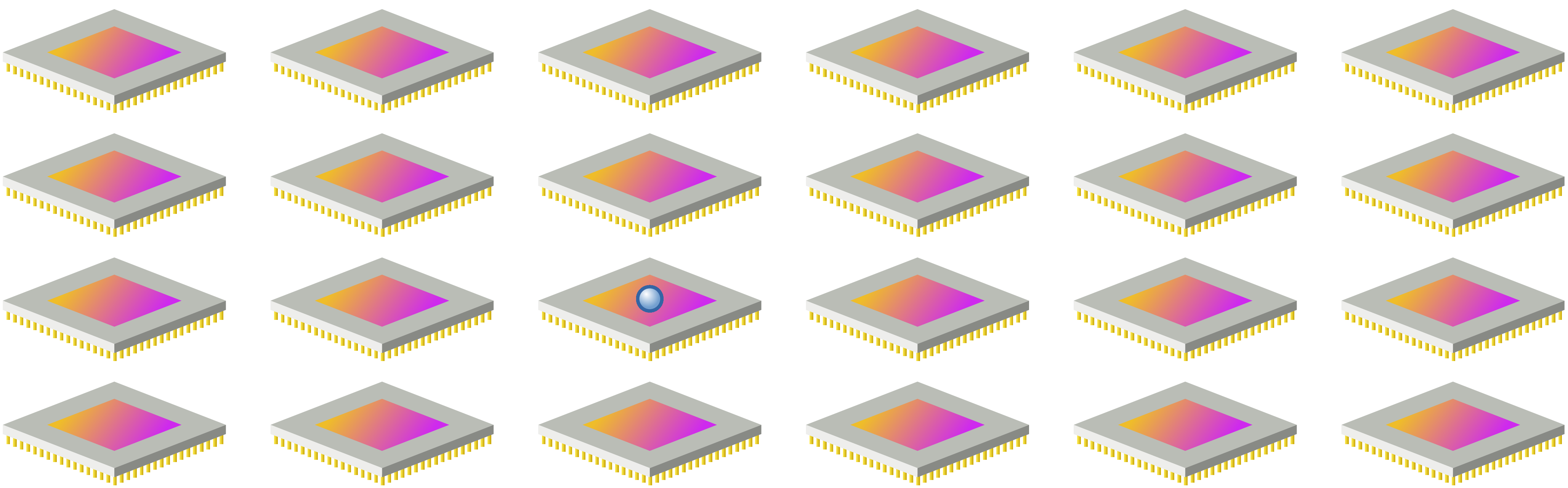
Which cores have a mapping cached? Who knows?!



TLB shootdown

munmap must notify cores of changes to cached mappings

Which cores have a mapping cached? Who knows?!

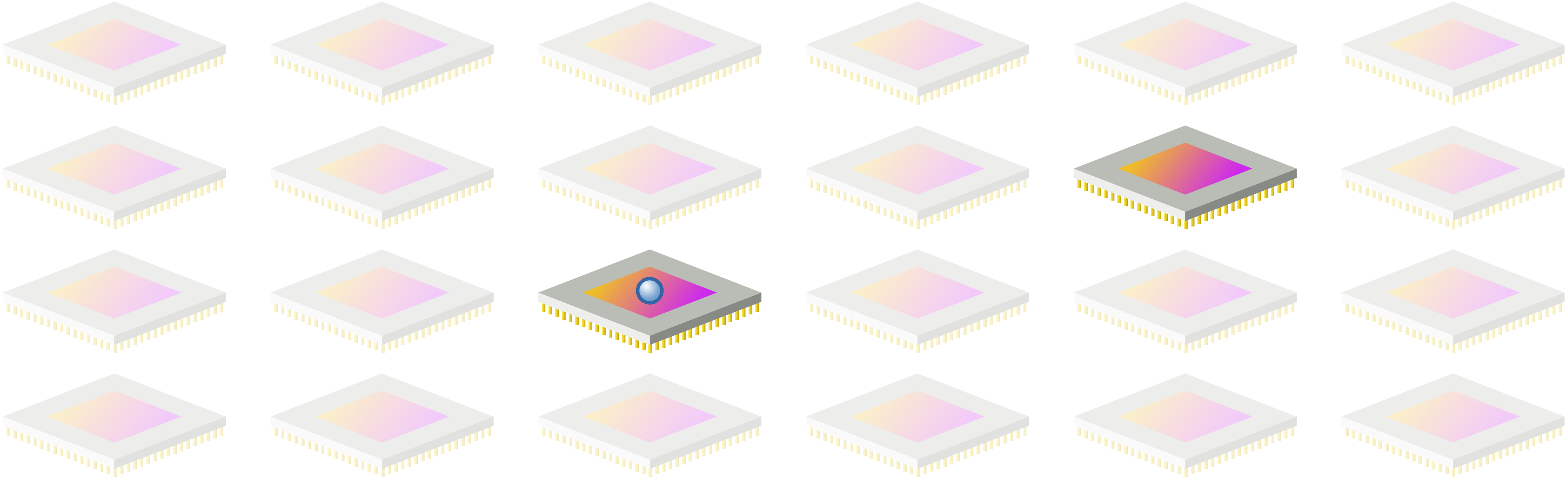


TLB shootdown

munmap must notify cores of changes to cached mappings

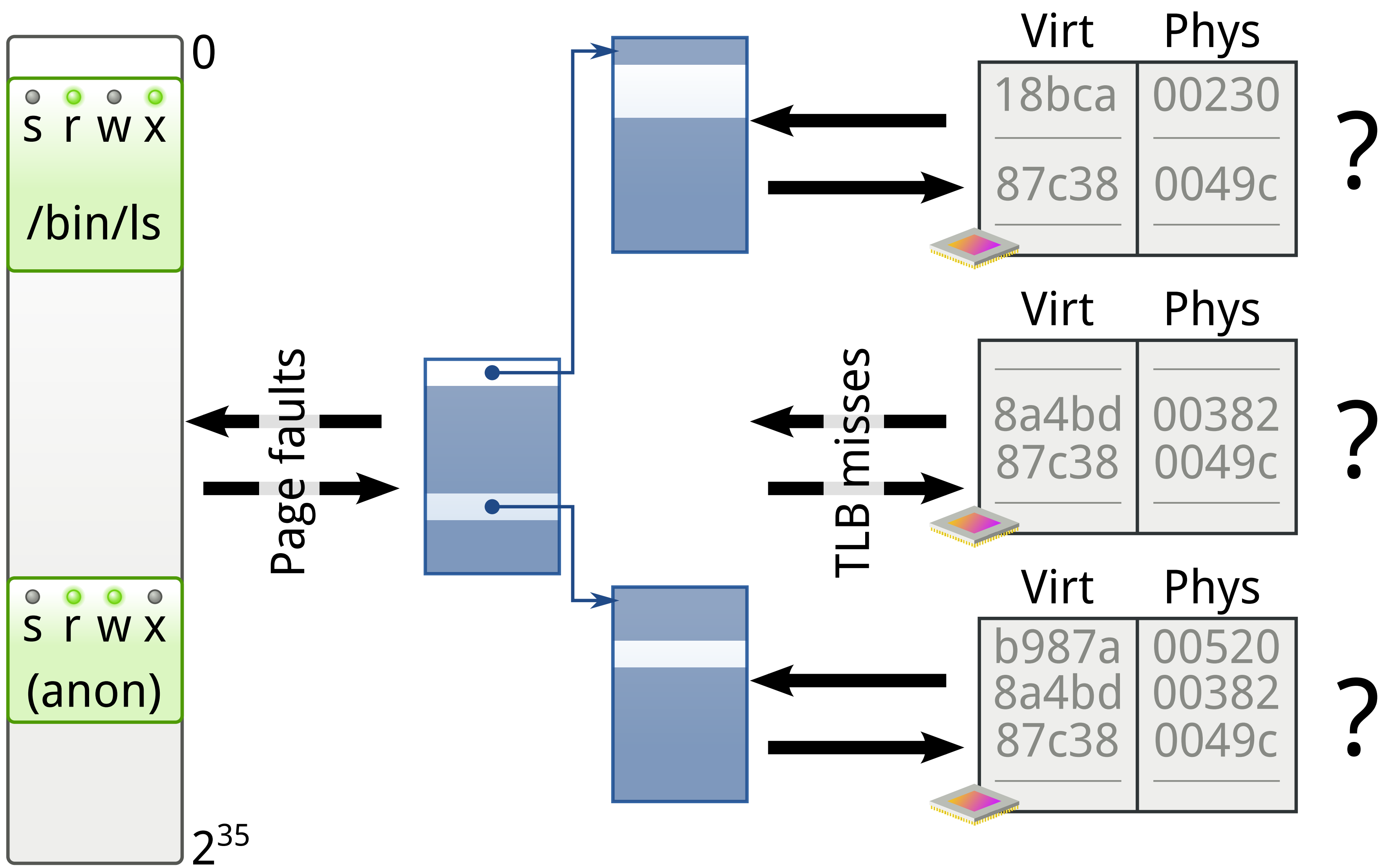
Which cores have a mapping cached? Who knows?!

In the common case, there is little or no sharing.



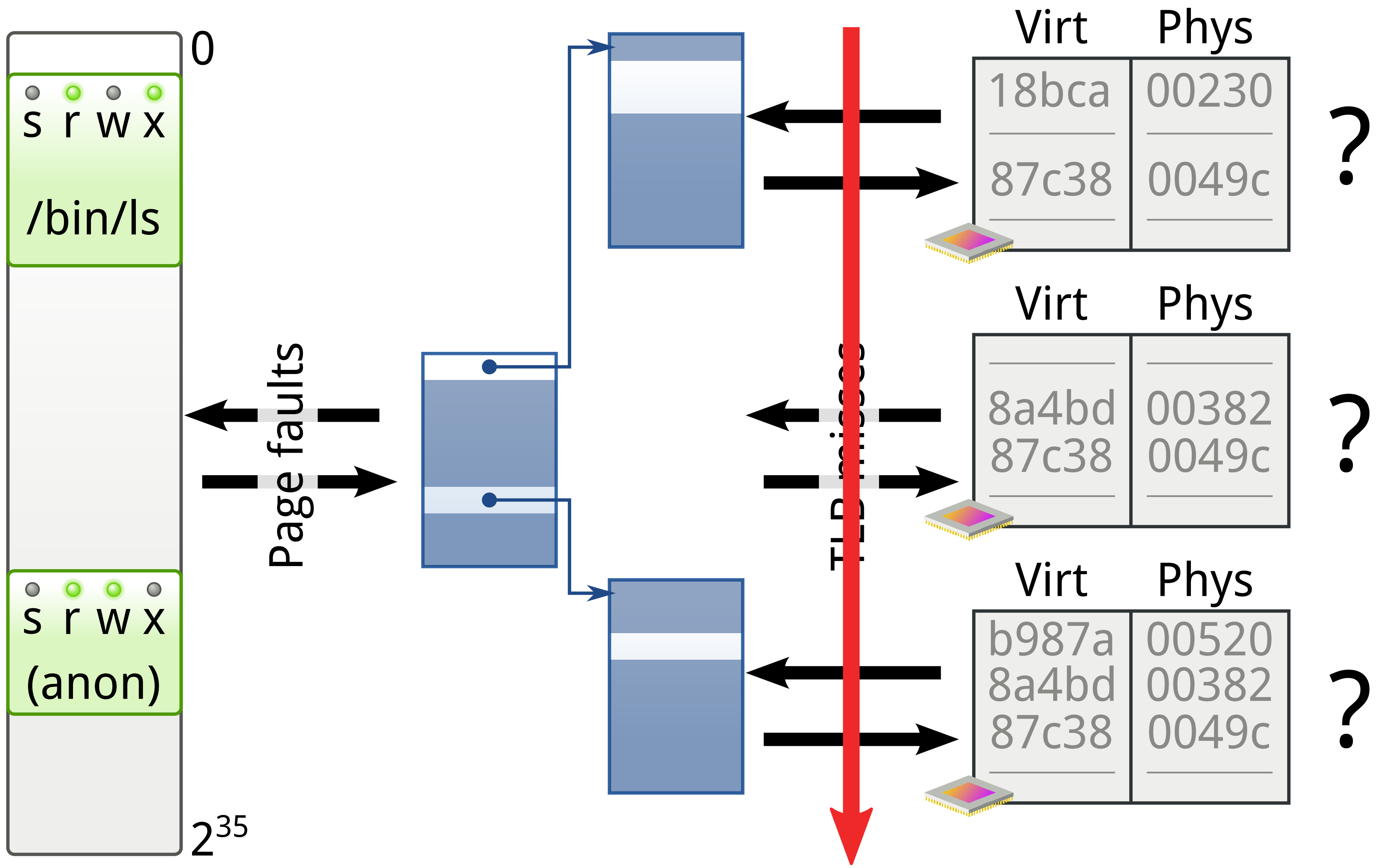
TLB tracking

A software-managed TLB would make this easy.



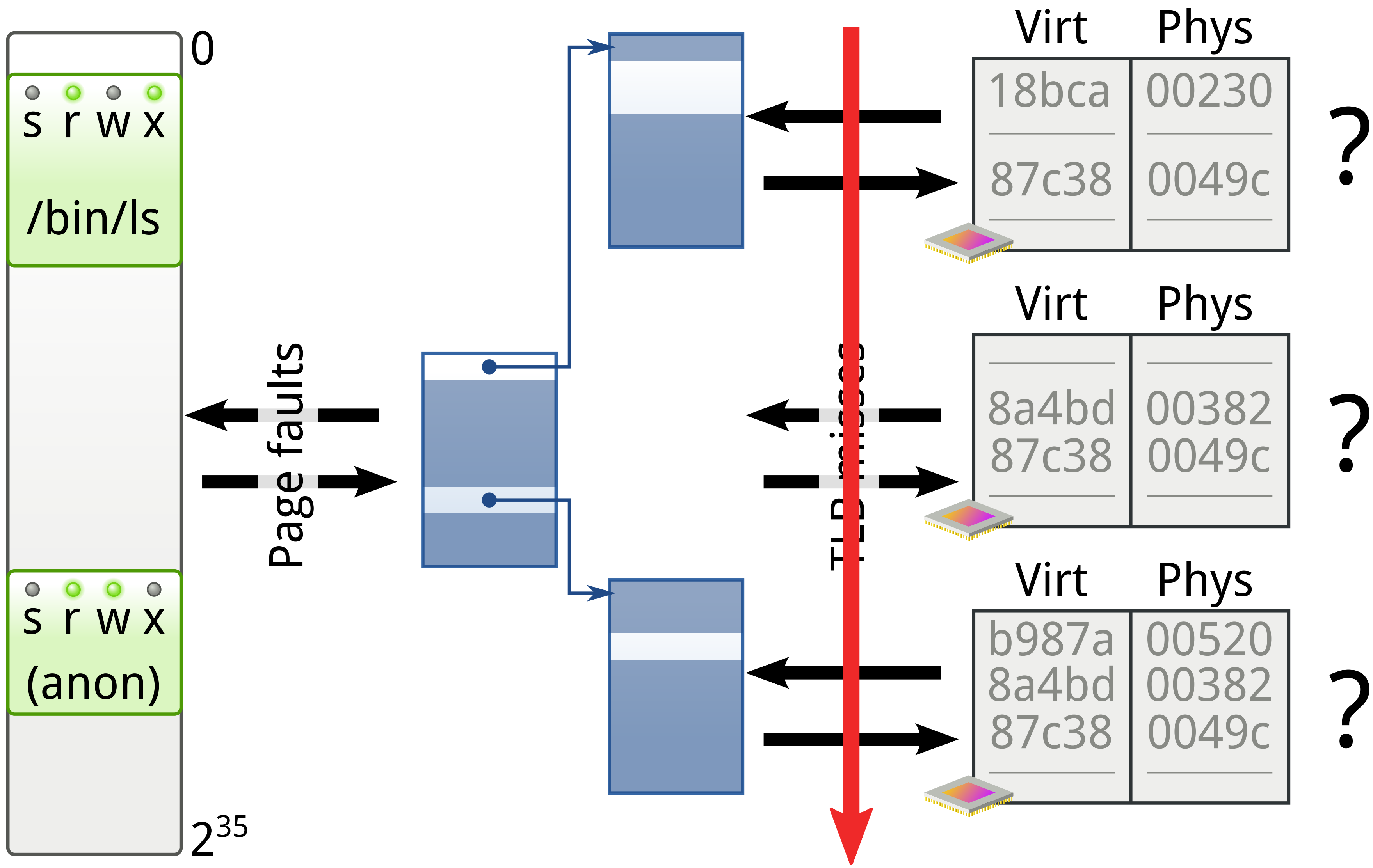
TLB tracking

A software-managed TLB would make this easy.



TLB tracking

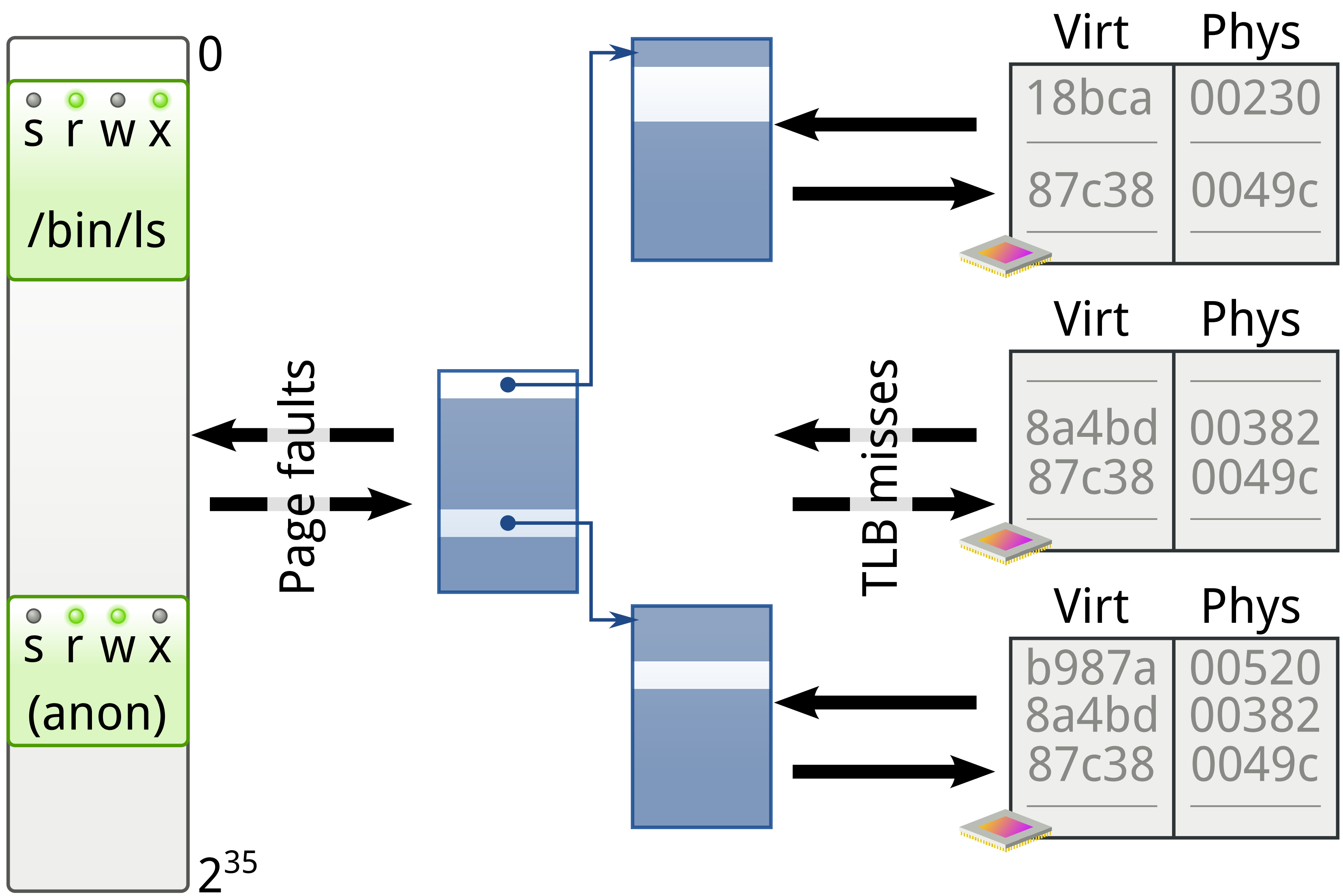
A software-managed TLB would make this easy.



~~Trap and track~~

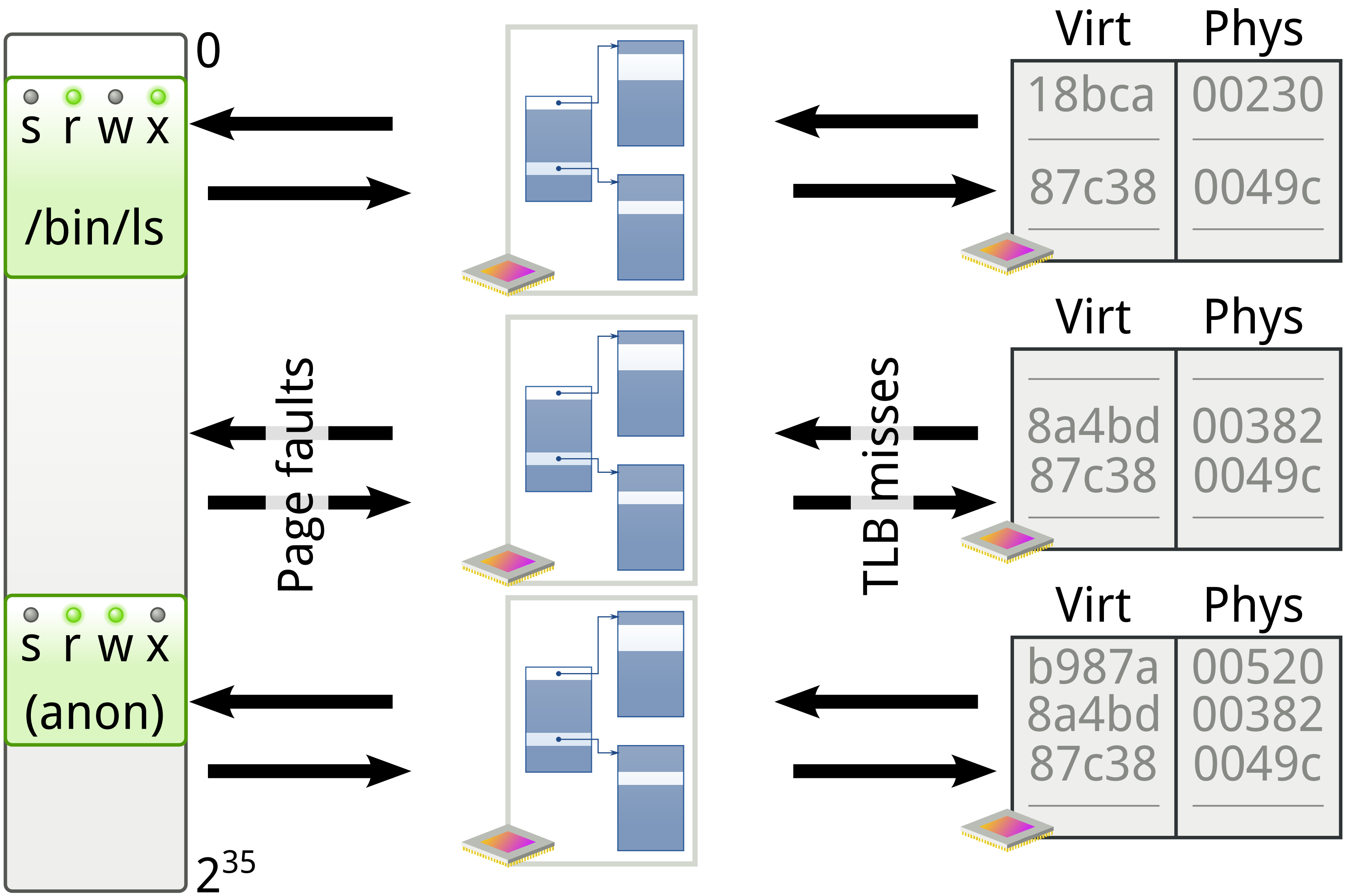
Soft TLBs, the hard way

Solution: Per-core page tables for precise TLB tracking



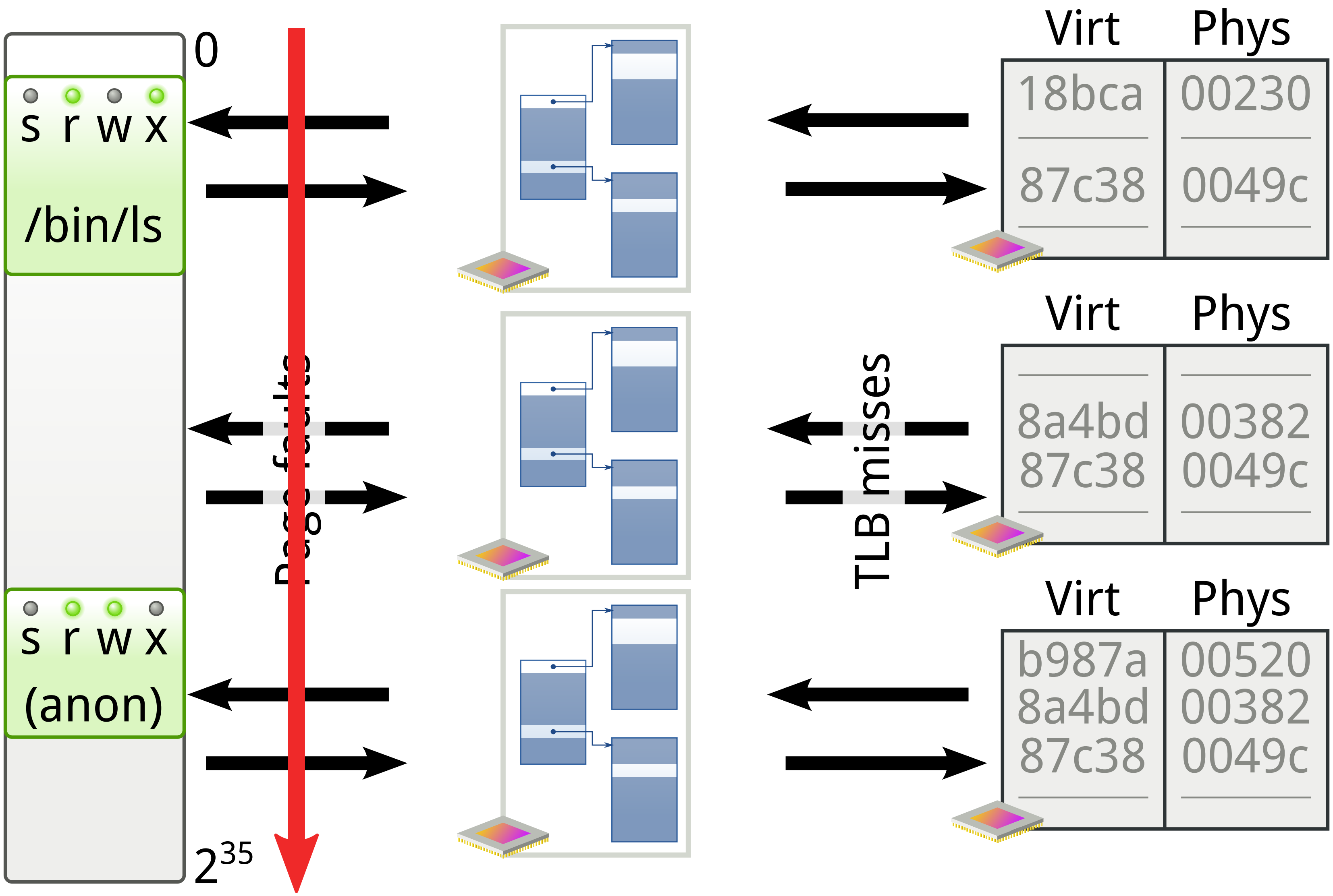
Soft TLBs, the hard way

Solution: Per-core page tables for precise TLB tracking



Soft TLBs, the hard way

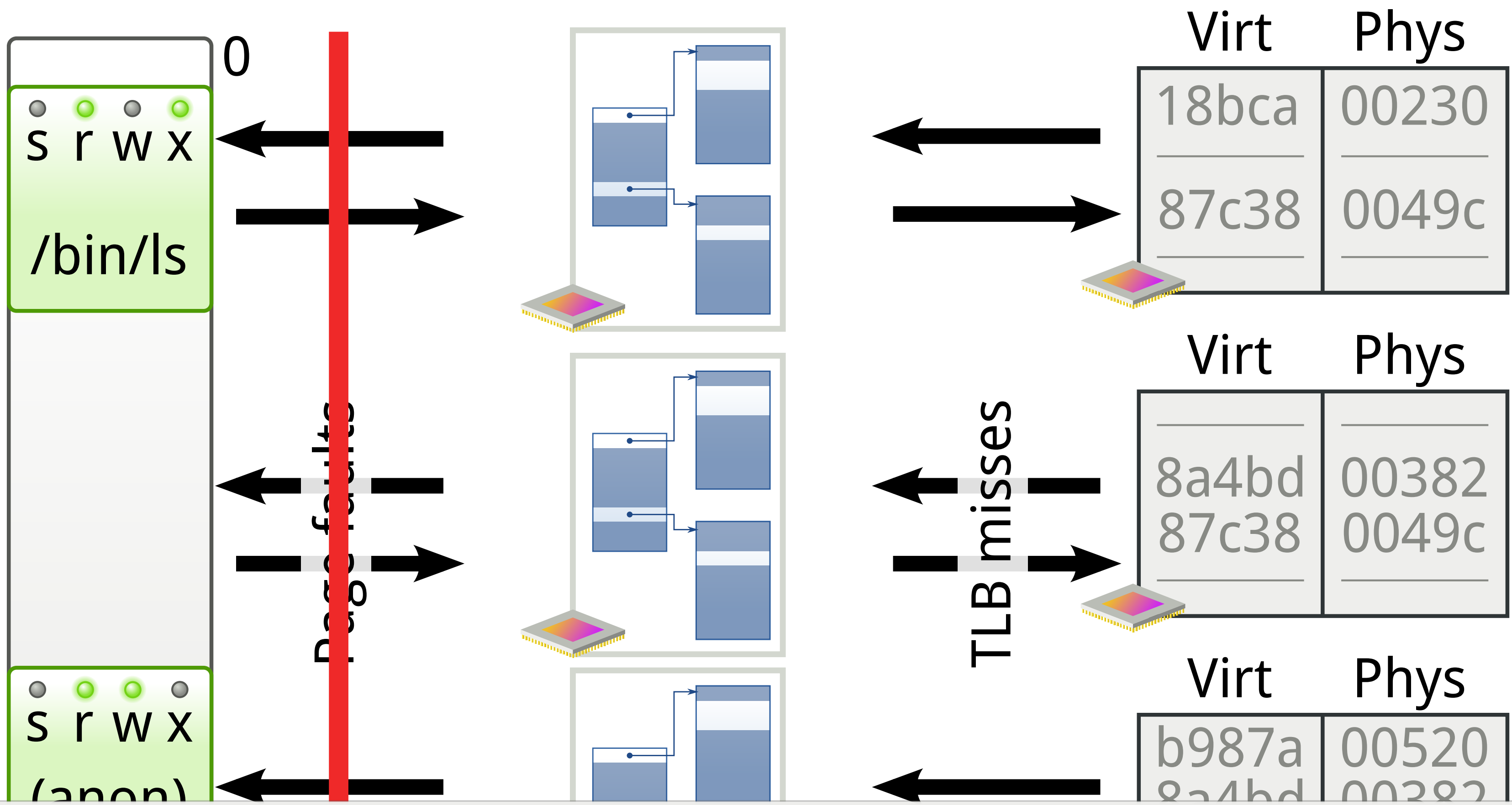
Solution: Per-core page tables for precise TLB tracking



Trap and track

Soft TLBs, the hard way

Solution: Per-core page tables for precise TLB tracking



TLB tracking allows us to target TLB shootdowns, eliminating unnecessary shootdown communication.

Trap and track

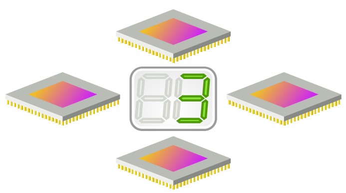
Reference counting

Reference counting for physical pages and radix nodes

Reference counting

Reference counting for physical pages and radix nodes

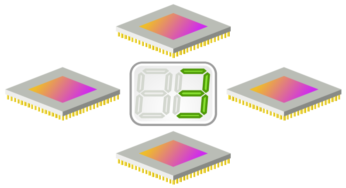
Shared counters

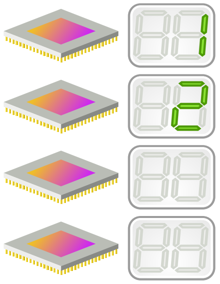


	Shared counters
Scalable inc/dec	N

Reference counting

Reference counting for physical pages and radix nodes

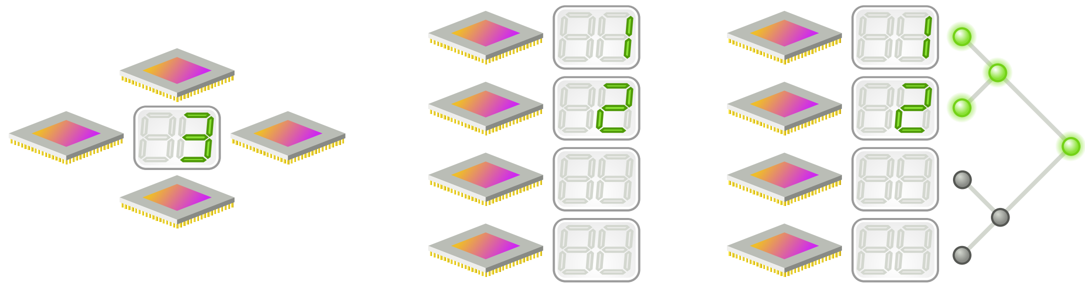




	Shared counters	Distributed counters
Scalable inc/dec	N	Y
Zero-detection cost	O(1)	O(objs*cpus)
Space	O(1)	O(cpus)

Reference counting

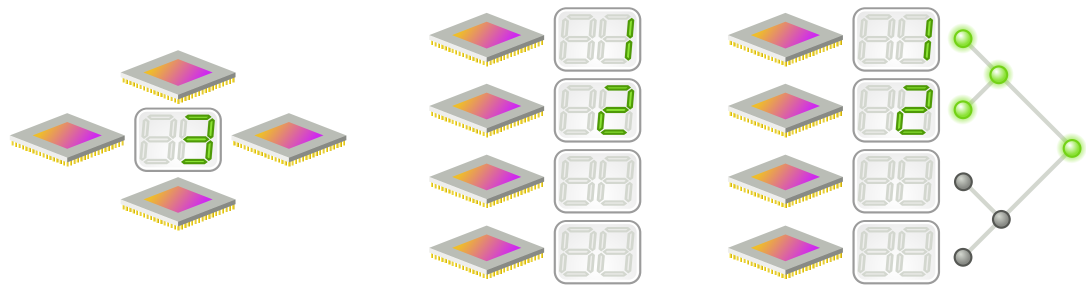
Reference counting for physical pages and radix nodes



	Shared counters	Distributed counters	SNZIs [Ellen '07]
Scalable inc/dec	N	Y	Mostly
Zero-detection cost	O(1)	O(objs*cpus)	O(1)
Space	O(1)	O(cpus)	O(cpus)

Reference counting

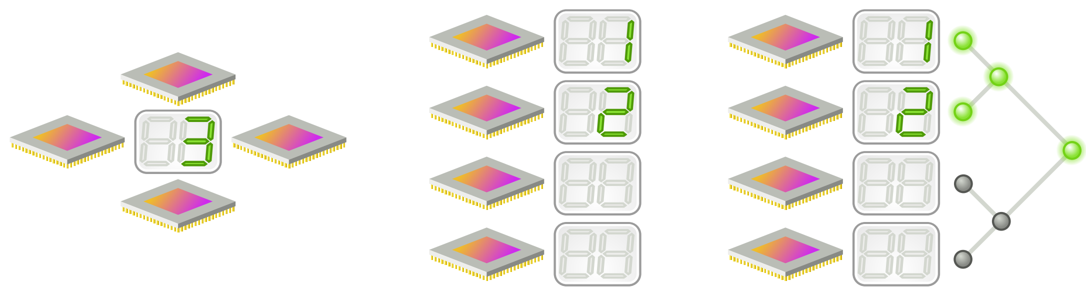
Reference counting for physical pages and radix nodes



	Shared counters	Distributed counters	SNZIs [Ellen '07]	Refcache
Scalable inc/dec	N	Y	Mostly	Y
Zero-detection cost	O(1)	O(objs*cpus)	O(1)	O(1)
Space	O(1)	O(cpus)	O(cpus)	O(1)

Reference counting

Reference counting for physical pages and radix nodes



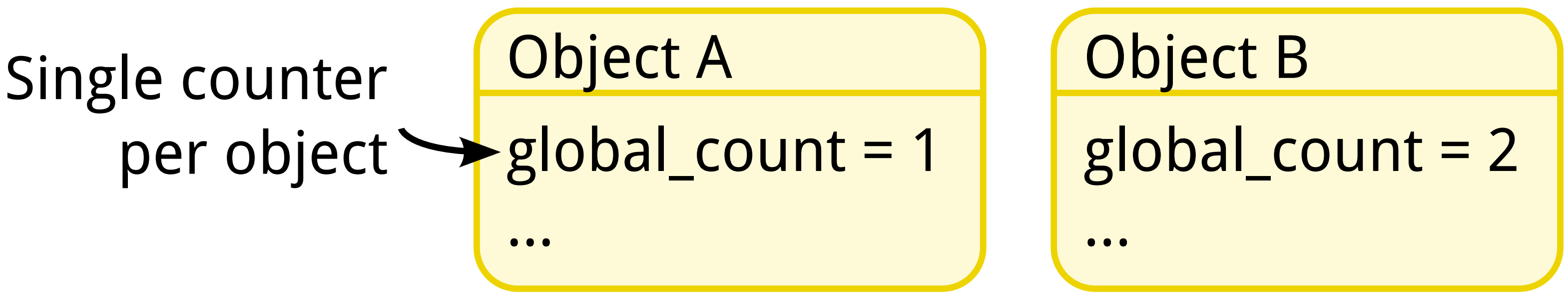
	Shared counters	Distributed counters	SNZIs [Ellen '07]	Refcache
Scalable inc/dec	N	Y	Mostly	Y
Zero-detection cost	O(1)	O(objs*cpus)	O(1)	O(1)
Space	O(1)	O(cpus)	O(cpus)	O(1)
Immediate zero detection	Y	N	Y	N

Refcache

Approach: Shared counters with per-core delta caches

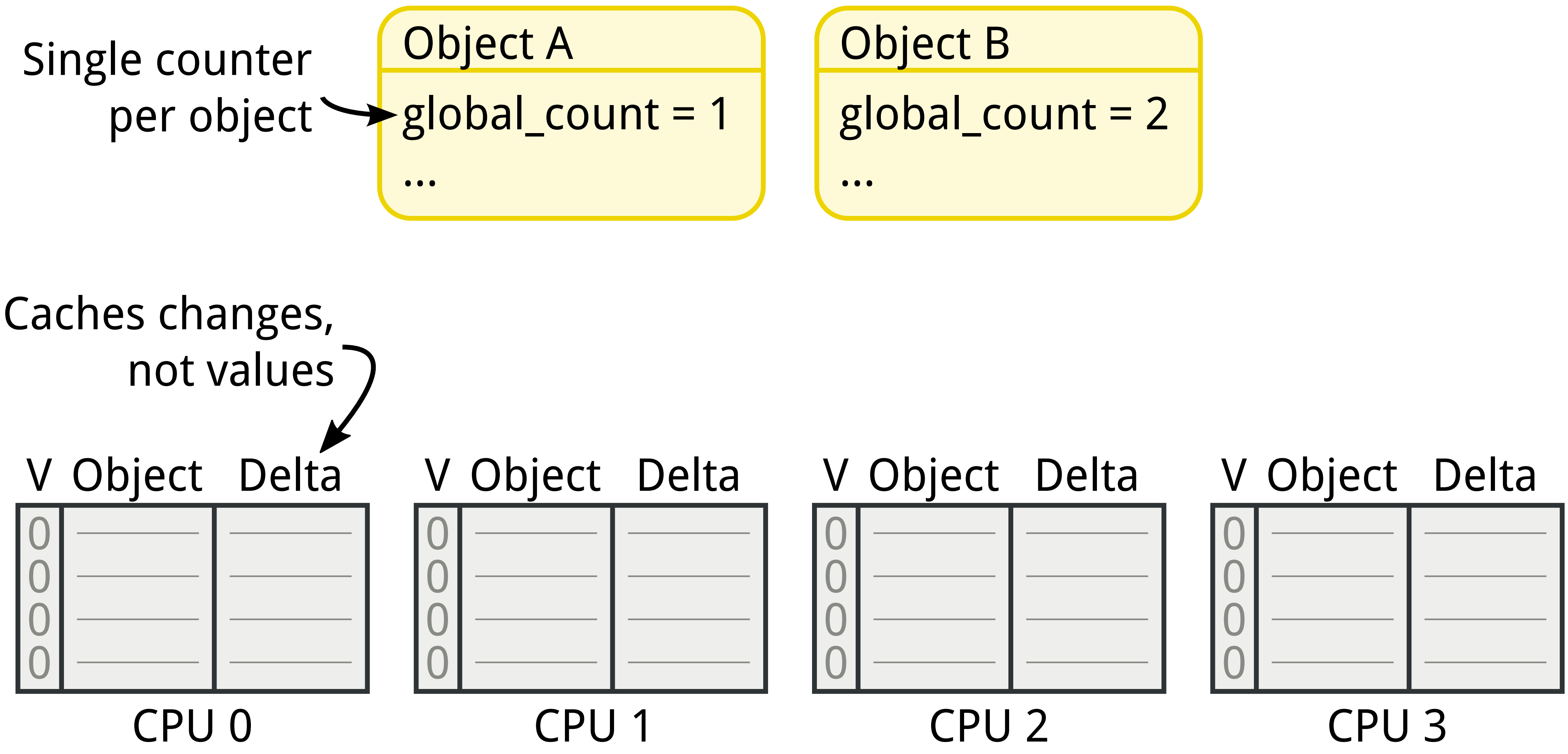
Refcache

Approach: Shared counters with per-core delta caches



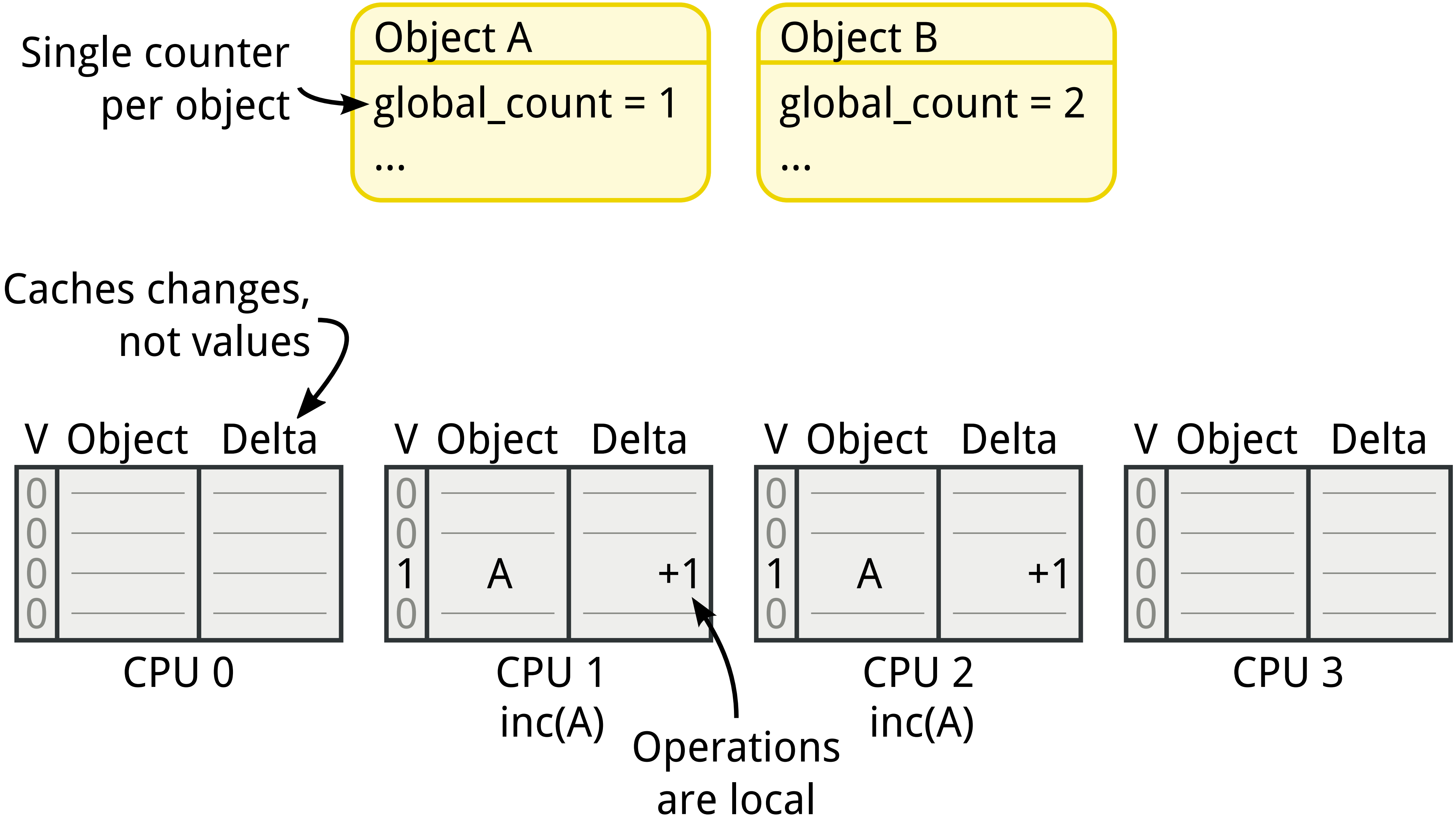
Refcache

Approach: Shared counters with per-core delta caches



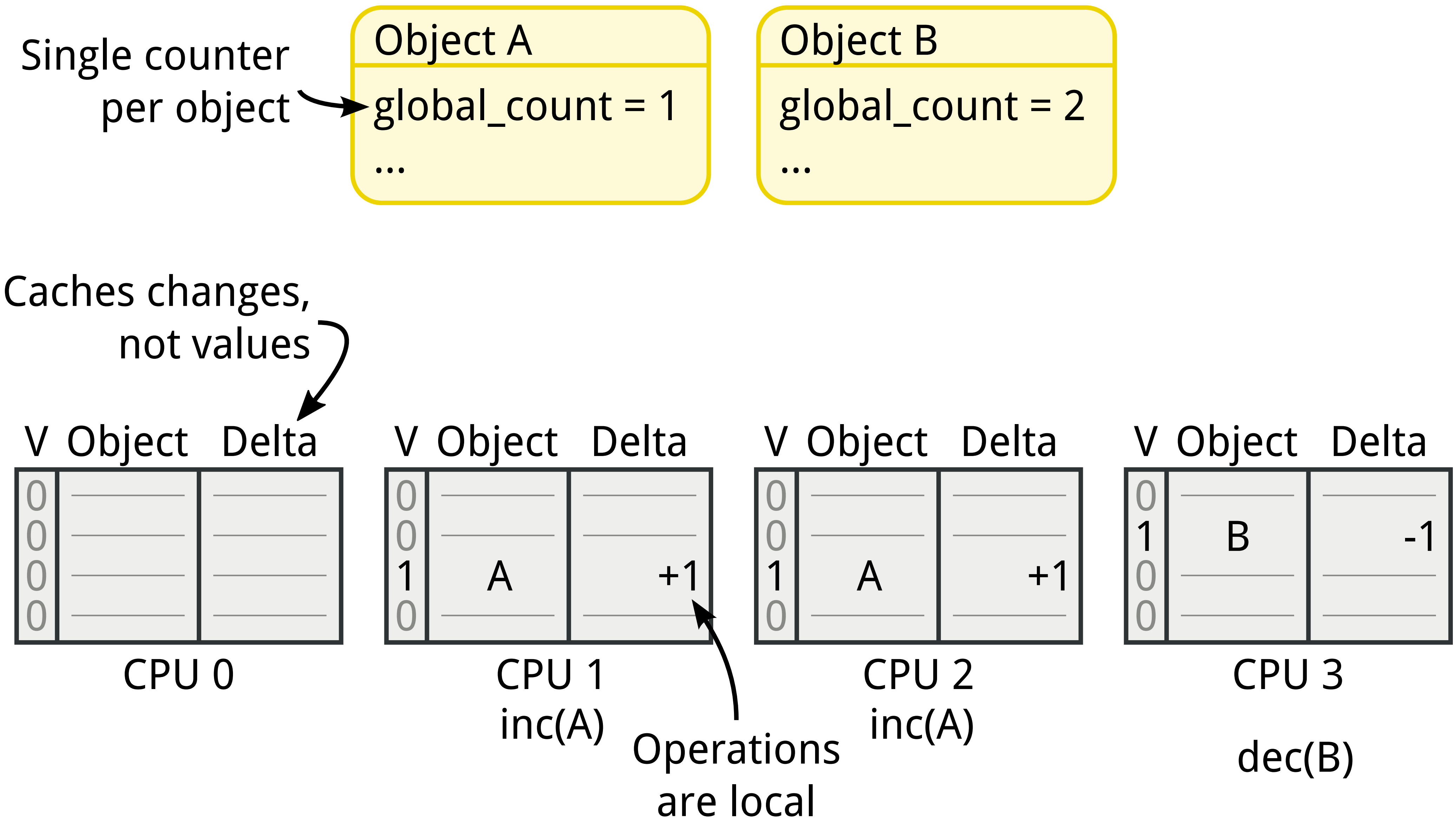
Refcache

Approach: Shared counters with per-core delta caches



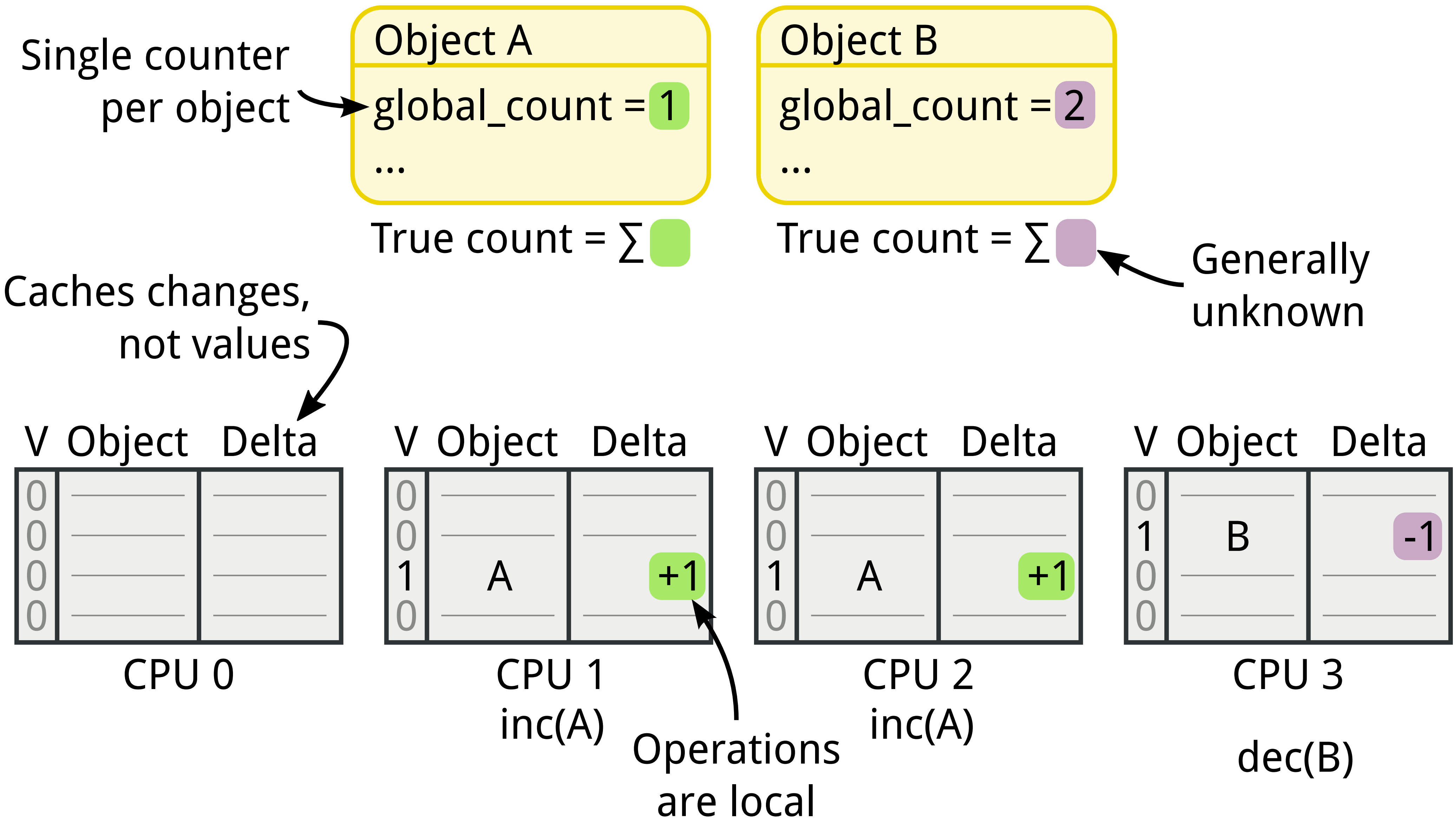
Refcache

Approach: Shared counters with per-core delta caches



Refcache

Approach: Shared counters with per-core delta caches



Refcache

When is the true count zero?

Refcache

When is the true count zero?

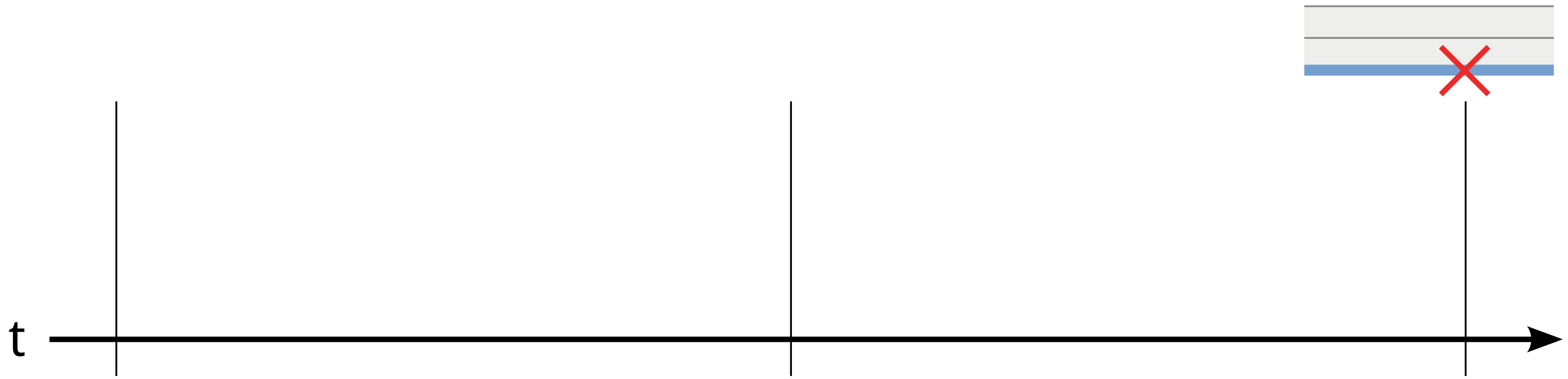
Assumption: When the true count is zero, it will stay zero.

Refcache

When is the true count zero?

Assumption: When the true count is zero, it will stay zero.

Divide time in to epochs. Each epoch, all CPUs flush their delta caches. If an object's global count stays zero for a whole epoch, then its true count is zero.

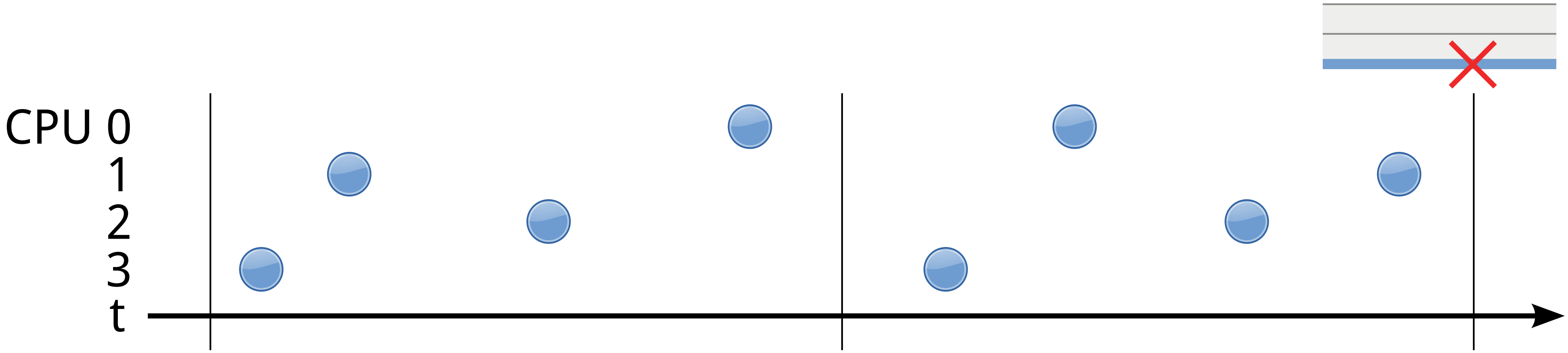


Refcache

When is the true count zero?

Assumption: When the true count is zero, it will stay zero.

Divide time in to epochs. Each epoch, all CPUs flush their delta caches. If an object's global count stays zero for a whole epoch, then its true count is zero.

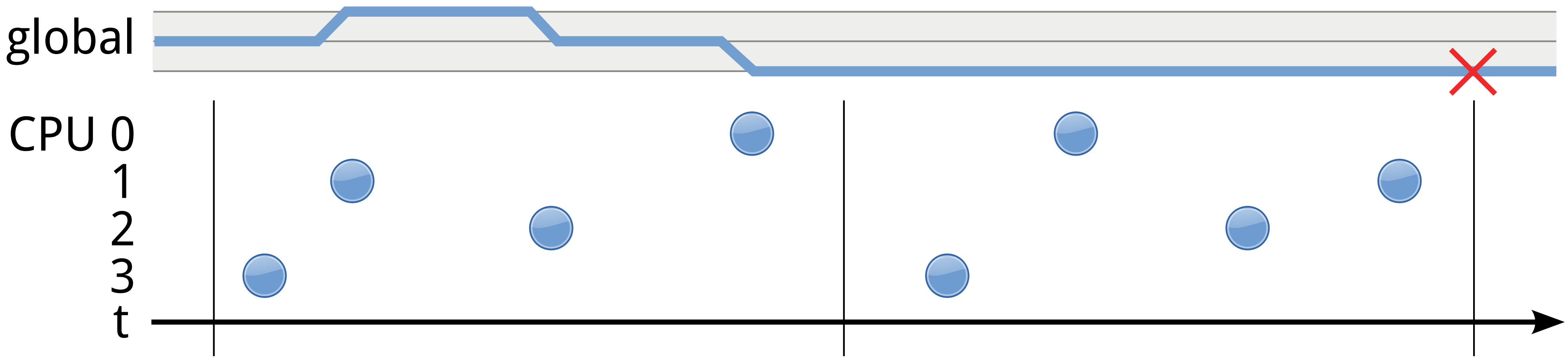


Refcache

When is the true count zero?

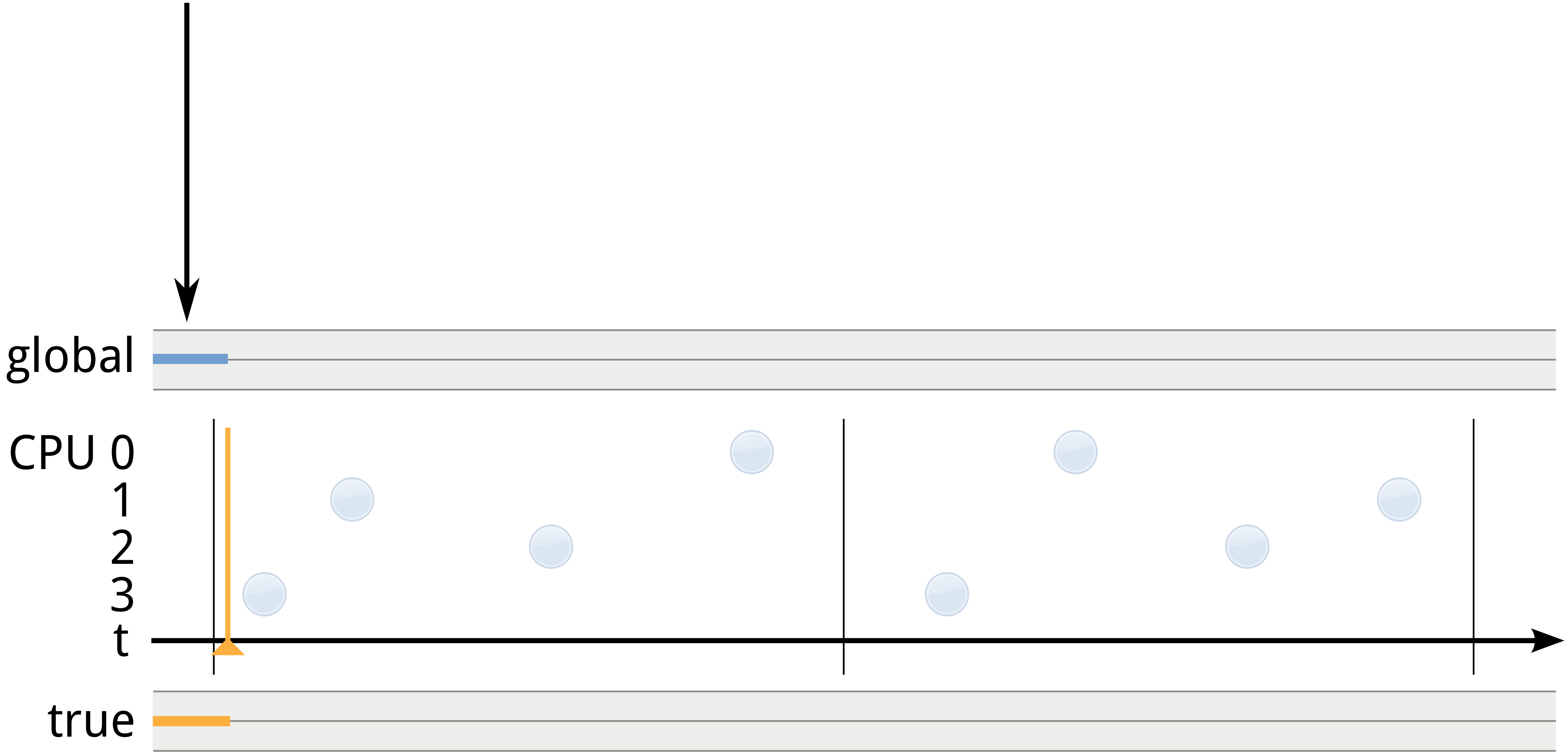
Assumption: When the true count is zero, it will stay zero.

Divide time in to epochs. Each epoch, all CPUs flush their delta caches. If an object's global count stays zero for a whole epoch, then its true count is zero.



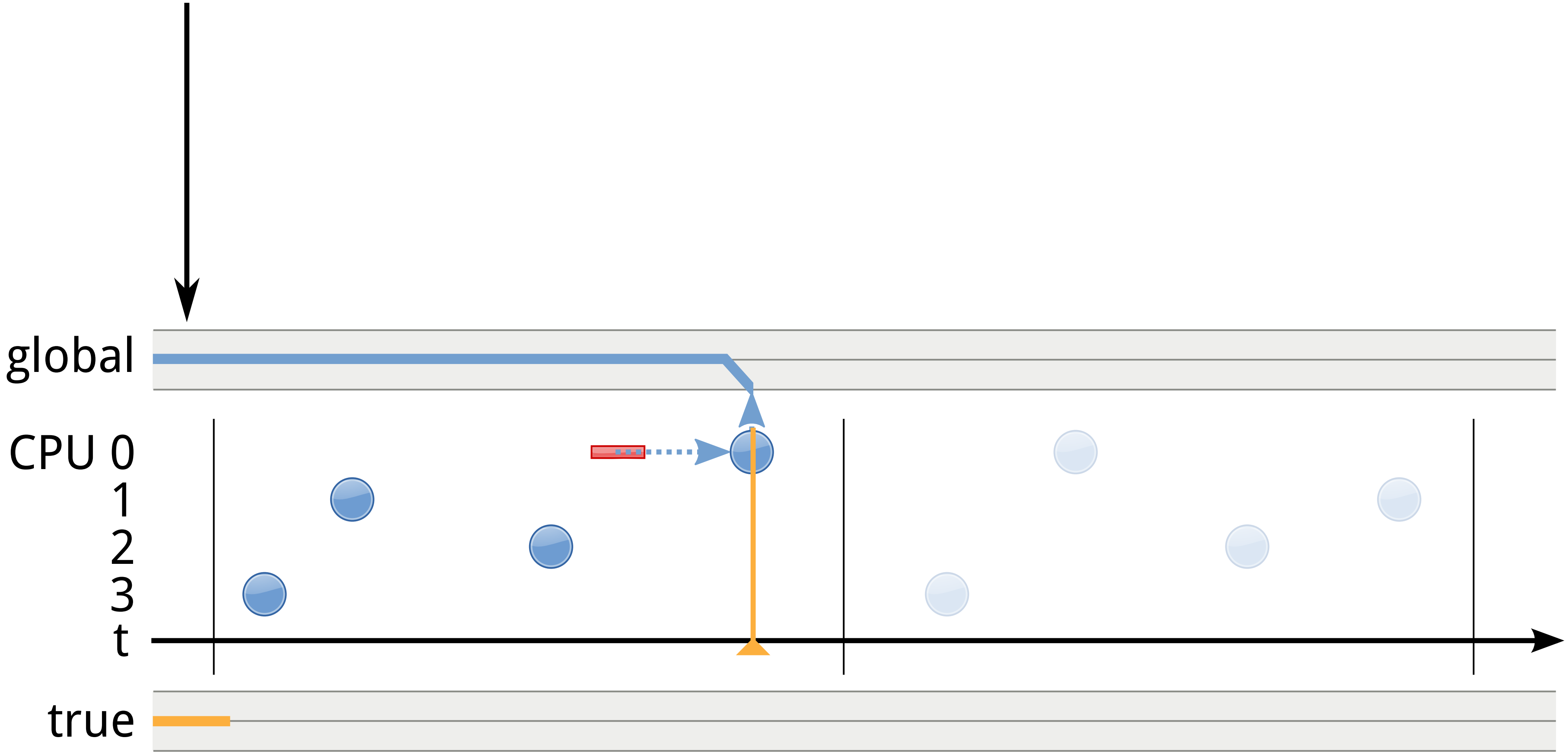
Refcache example

Initially: Global count is 1, no cached deltas (so true count is 1)



Refcache example

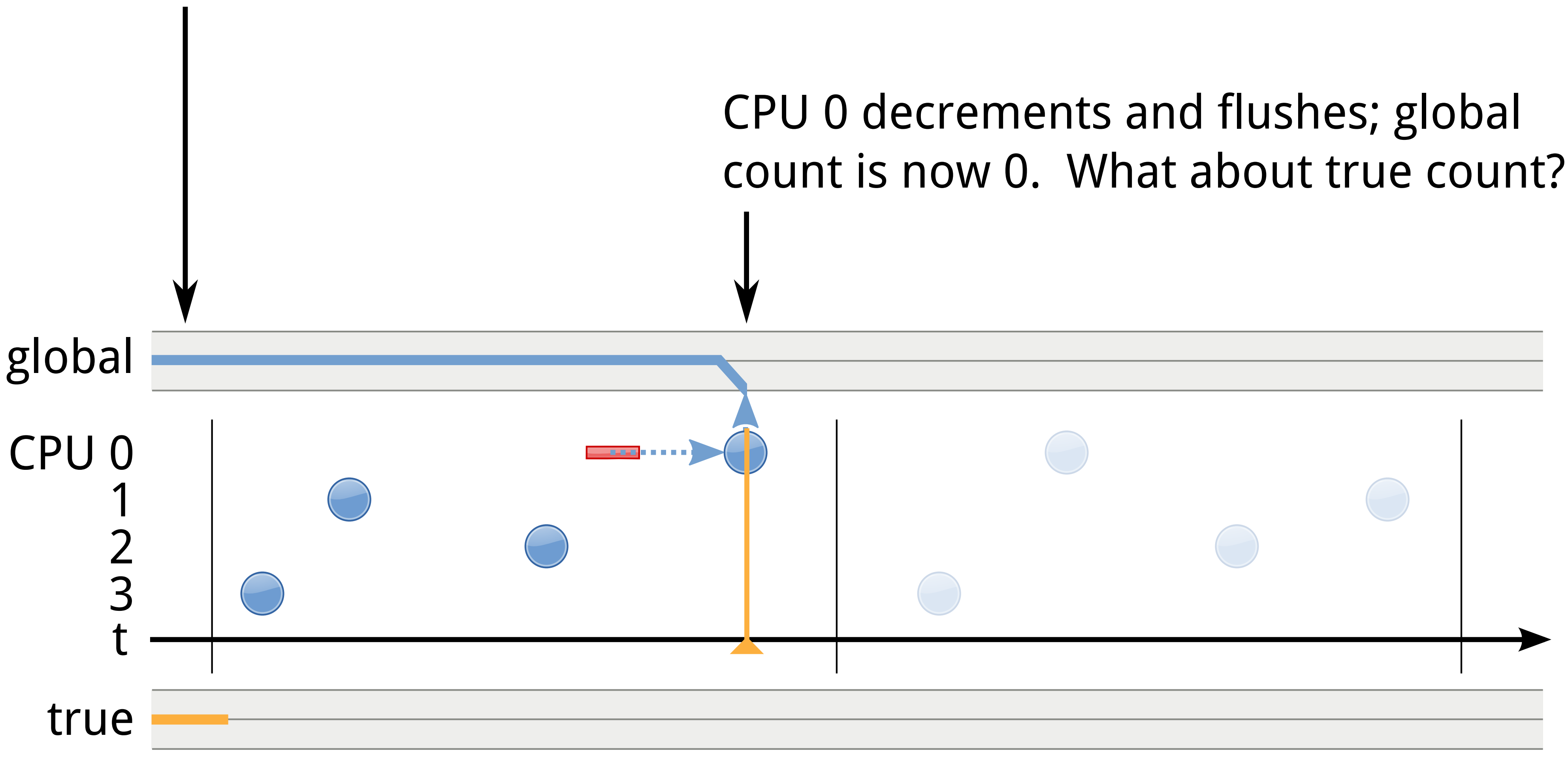
Initially: Global count is 1, no cached deltas (so true count is 1)



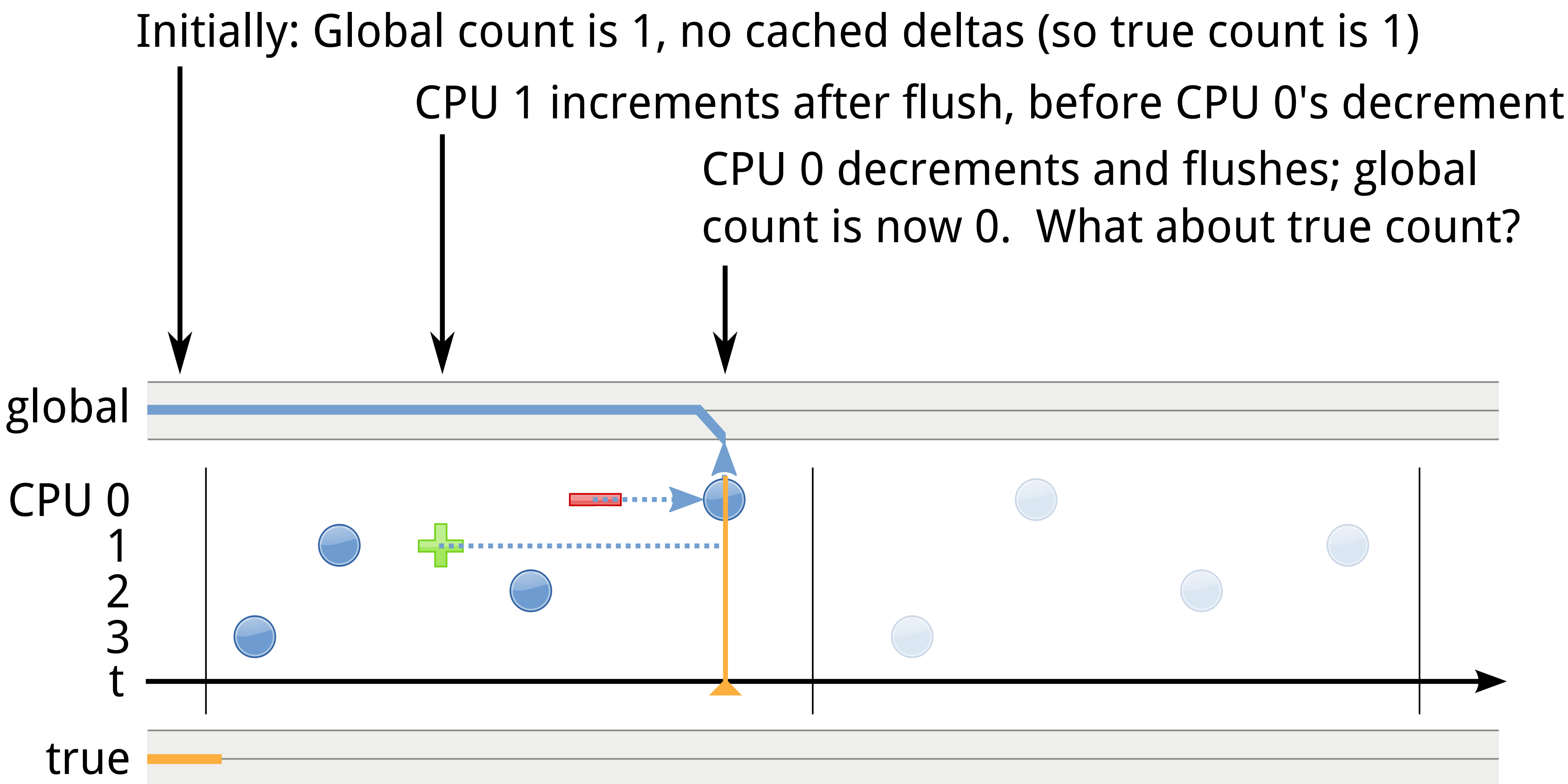
Refcache example

Initially: Global count is 1, no cached deltas (so true count is 1)

CPU 0 decrements and flushes; global count is now 0. What about true count?



Refcache example

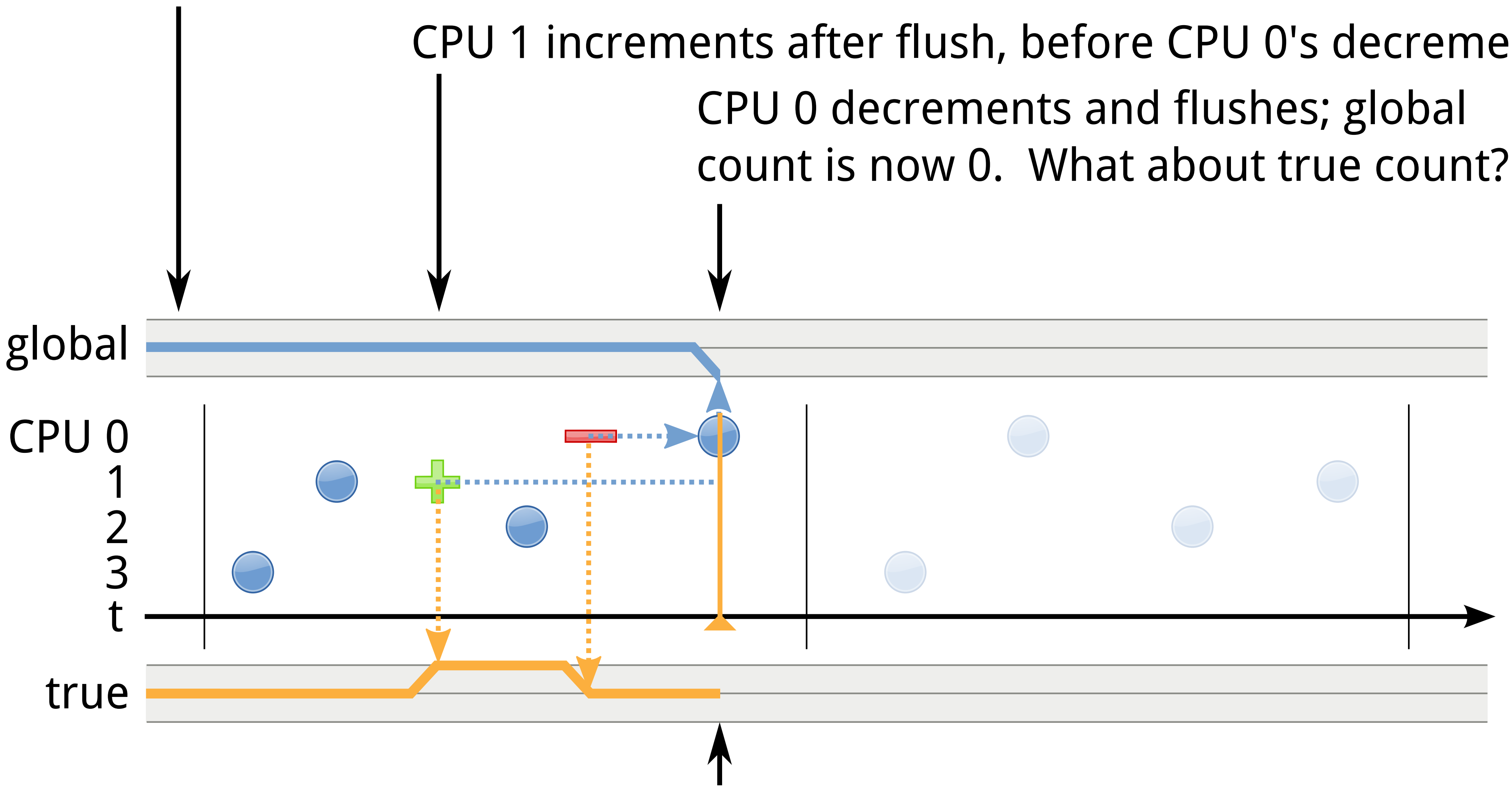


Refcache example

Initially: Global count is 1, no cached deltas (so true count is 1)

CPU 1 increments after flush, before CPU 0's decrement

CPU 0 decrements and flushes; global count is now 0. What about true count?



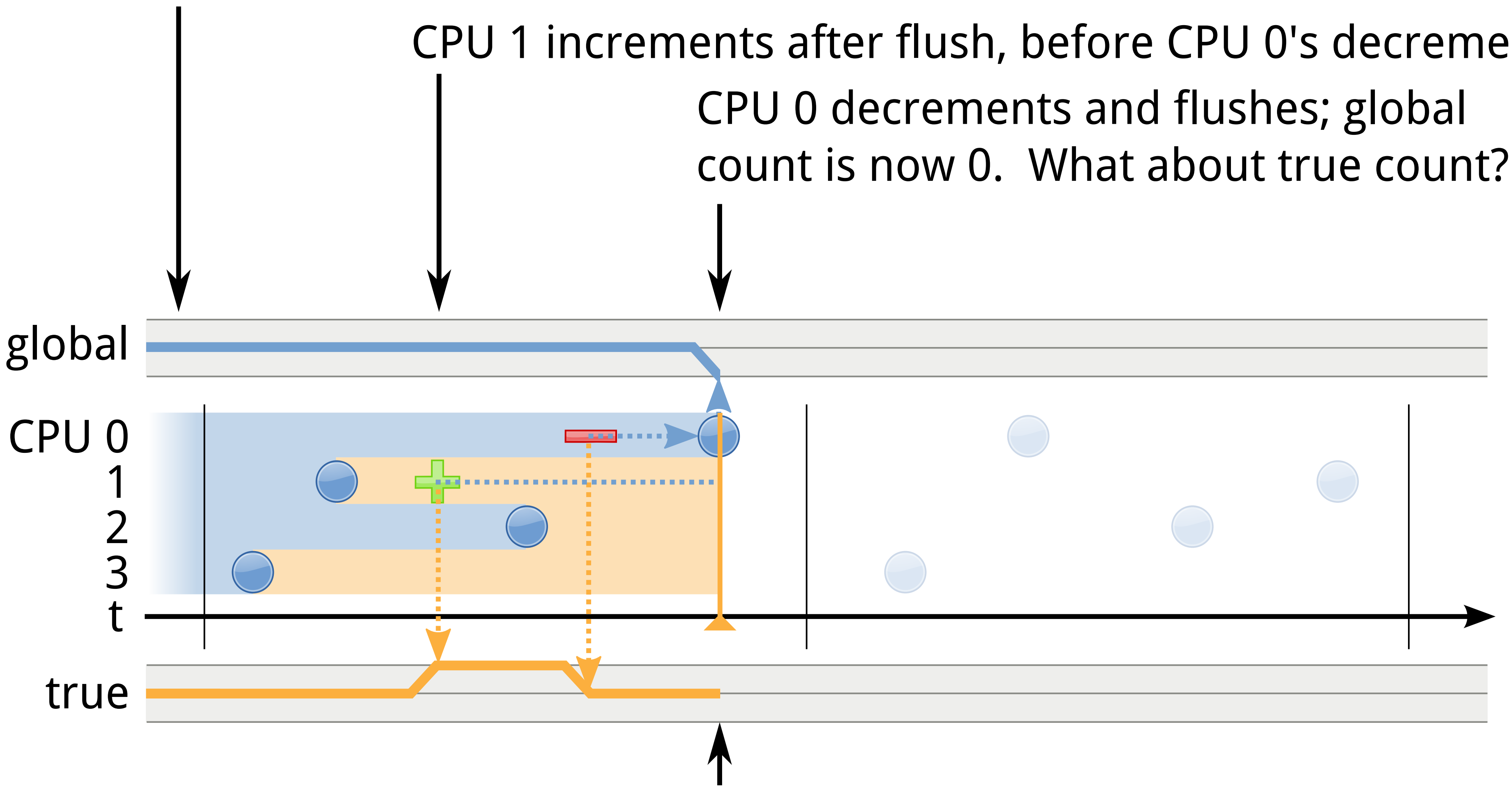
The true count is the sum of everything up to right now.

Refcache example

Initially: Global count is 1, no cached deltas (so true count is 1)

CPU 1 increments after flush, before CPU 0's decrement

CPU 0 decrements and flushes; global count is now 0. What about true count?



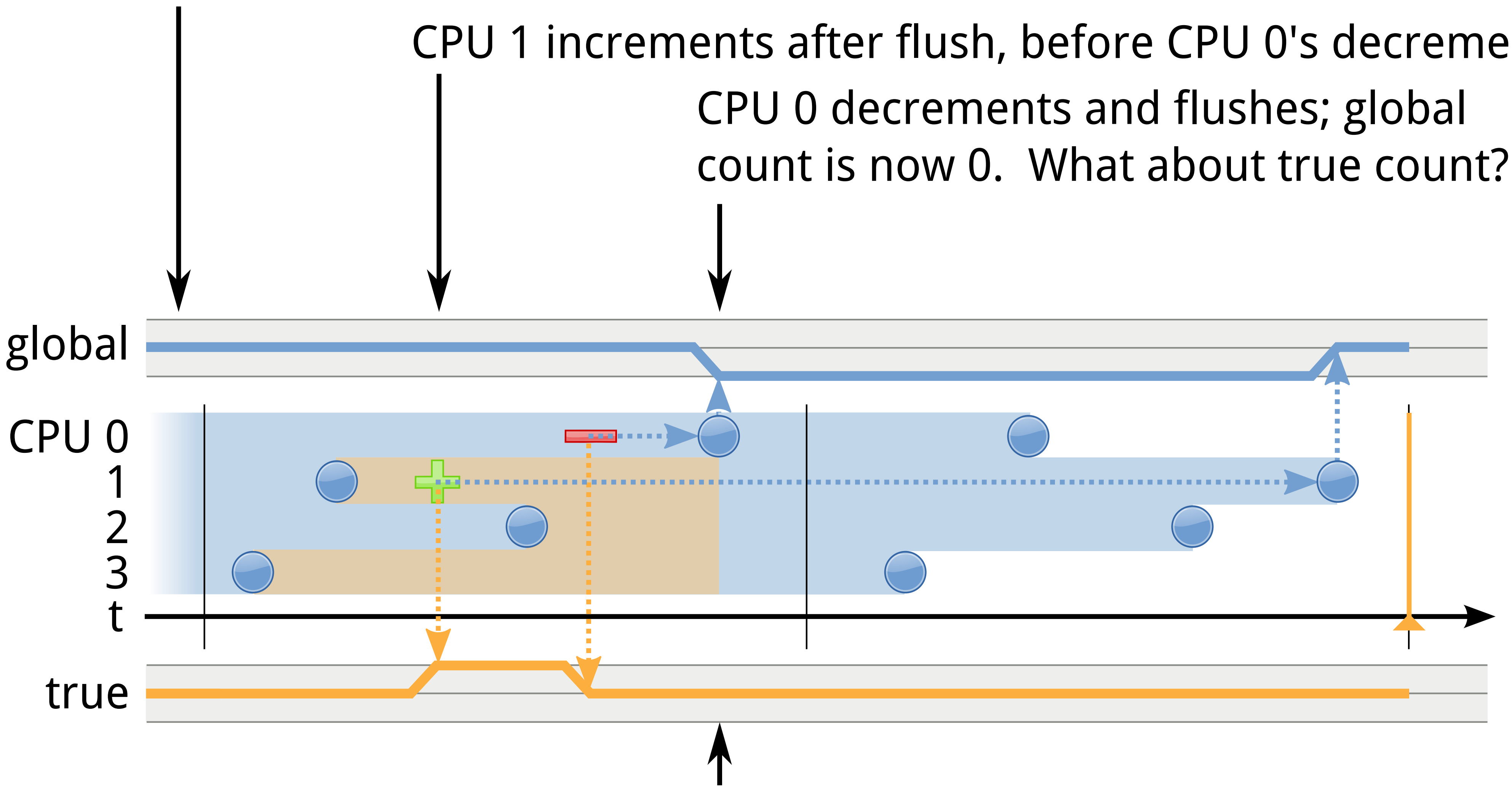
The true count is the sum of everything up to right now.
But the global count only reflects the blue region.
Operations in the orange region are still cached.

Refcache example

Initially: Global count is 1, no cached deltas (so true count is 1)

CPU 1 increments after flush, before CPU 0's decrement

CPU 0 decrements and flushes; global count is now 0. What about true count?



The true count is the sum of everything up to right now.
But the global count only reflects the blue region.
Operations in the orange region are still cached.

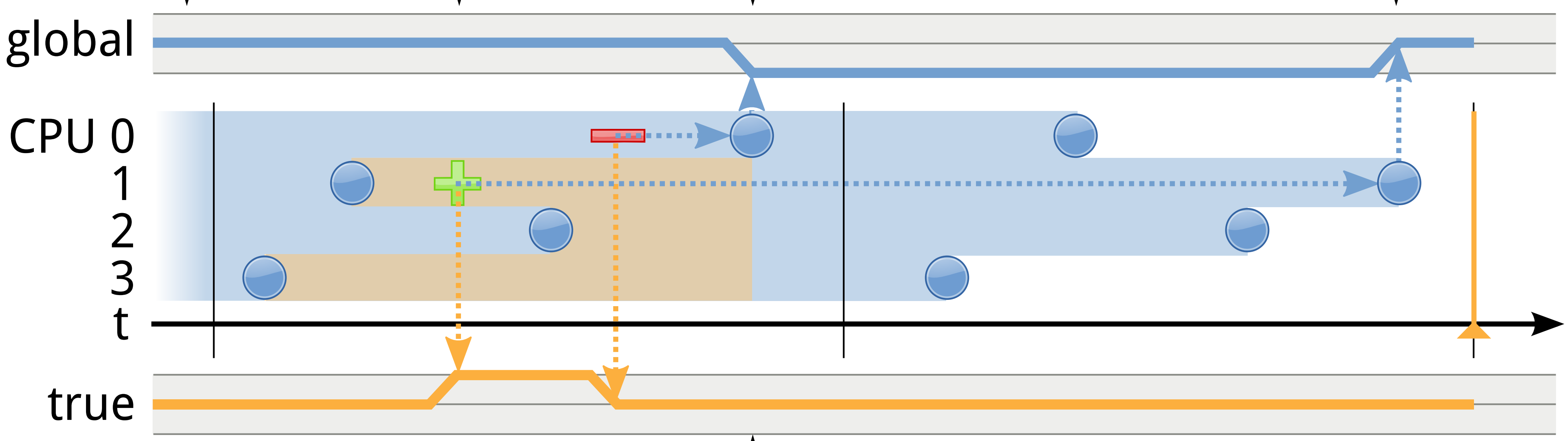
Refcache example

Initially: Global count is 1, no cached deltas (so true count is 1)

CPU 1 increments after flush, before CPU 0's decrement

CPU 0 decrements and flushes; global count is now 0. What about true count?

Global count now reflects cached ops



The true count is the sum of everything up to right now.
But the global count only reflects the blue region.
Operations in the orange region are still cached.

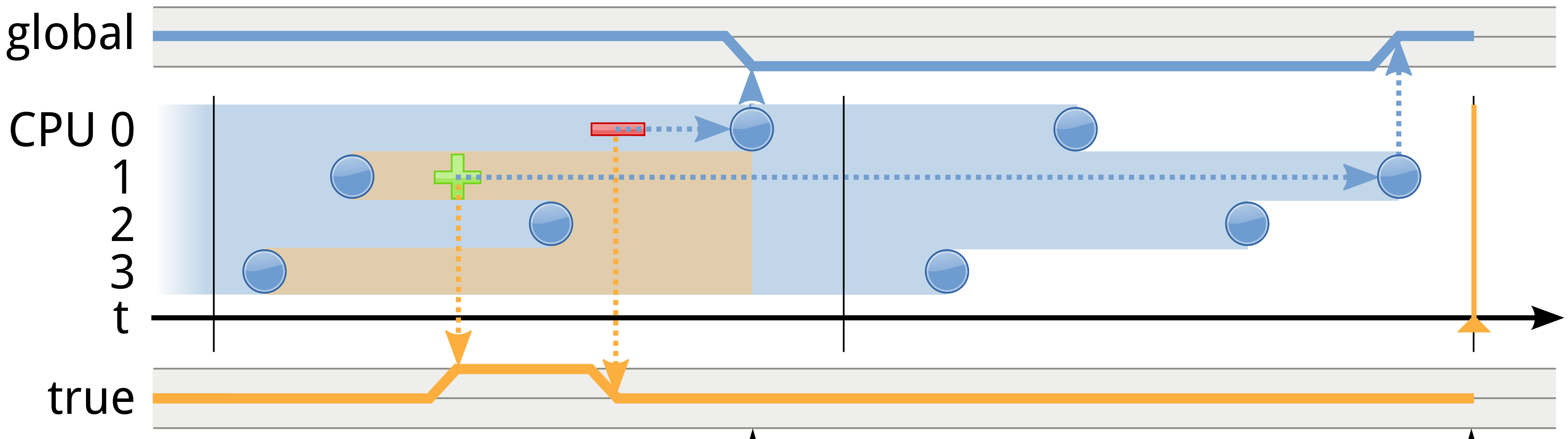
Refcache example

Initially: Global count is 1, no cached deltas (so true count is 1)

CPU 1 increments after flush, before CPU 0's decrement

CPU 0 decrements and flushes; global count is now 0. What about true count?

Global count now reflects cached ops



The true count is the sum of everything up to right now.
But the global count only reflects the blue region.
Operations in the orange region are still cached.

Abort delete

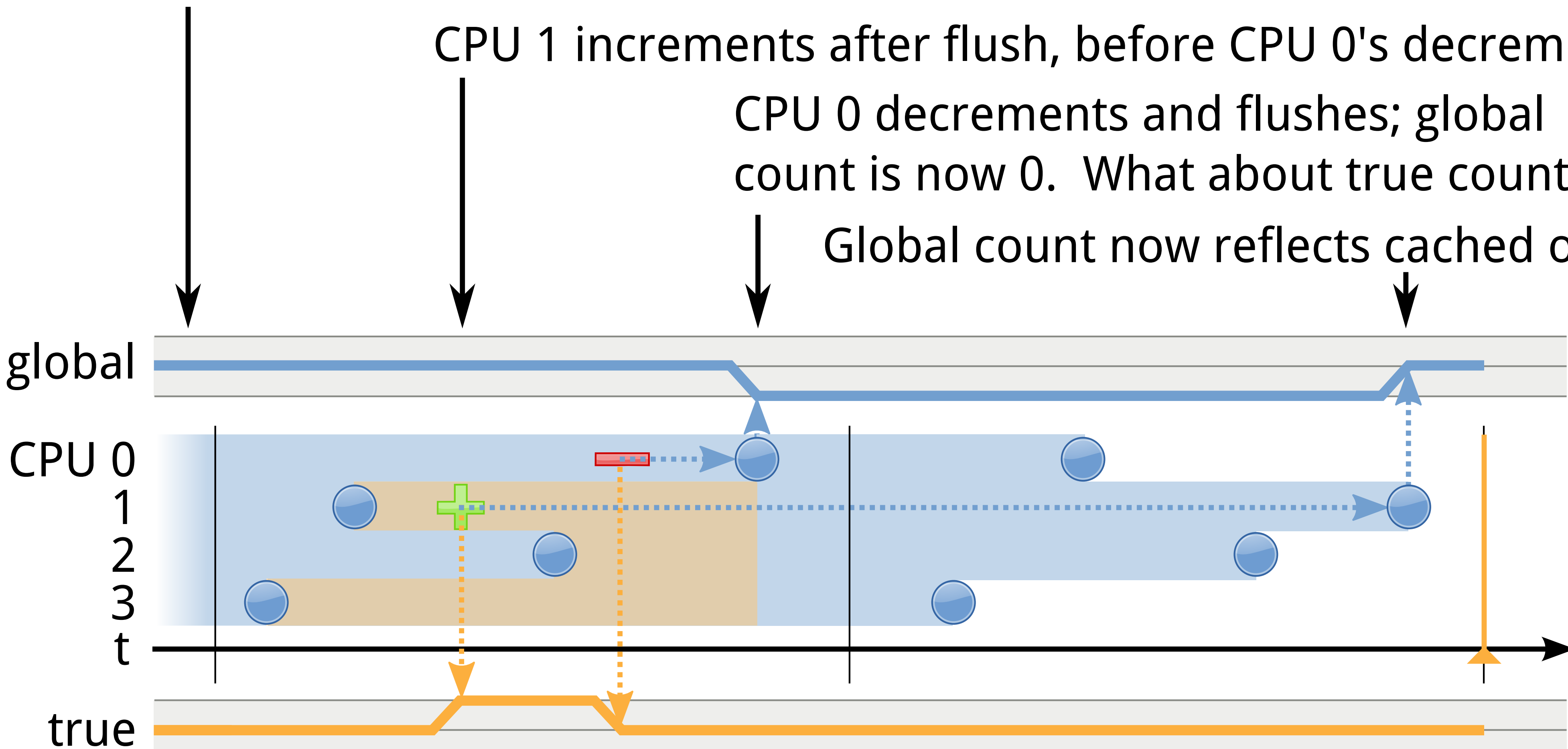
Refcache example

Initially: Global count is 1, no cached deltas (so true count is 1)

CPU 1 increments after flush, before CPU 0's decrement

CPU 0 decrements and flushes; global count is now 0. What about true count?

Global count now reflects cached ops

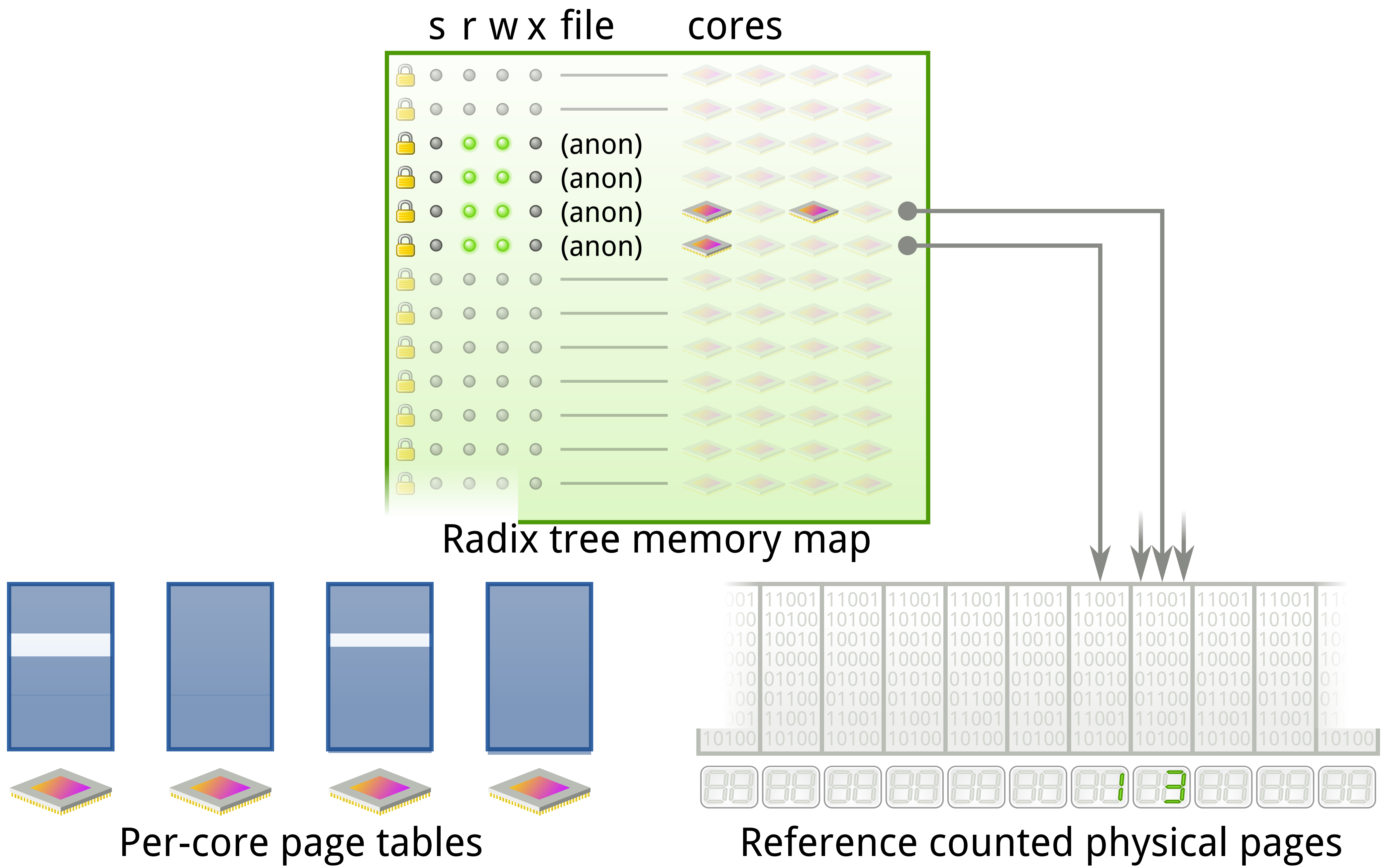


Refcache enables time- and space-efficient scalable reference counting with minimal latency.

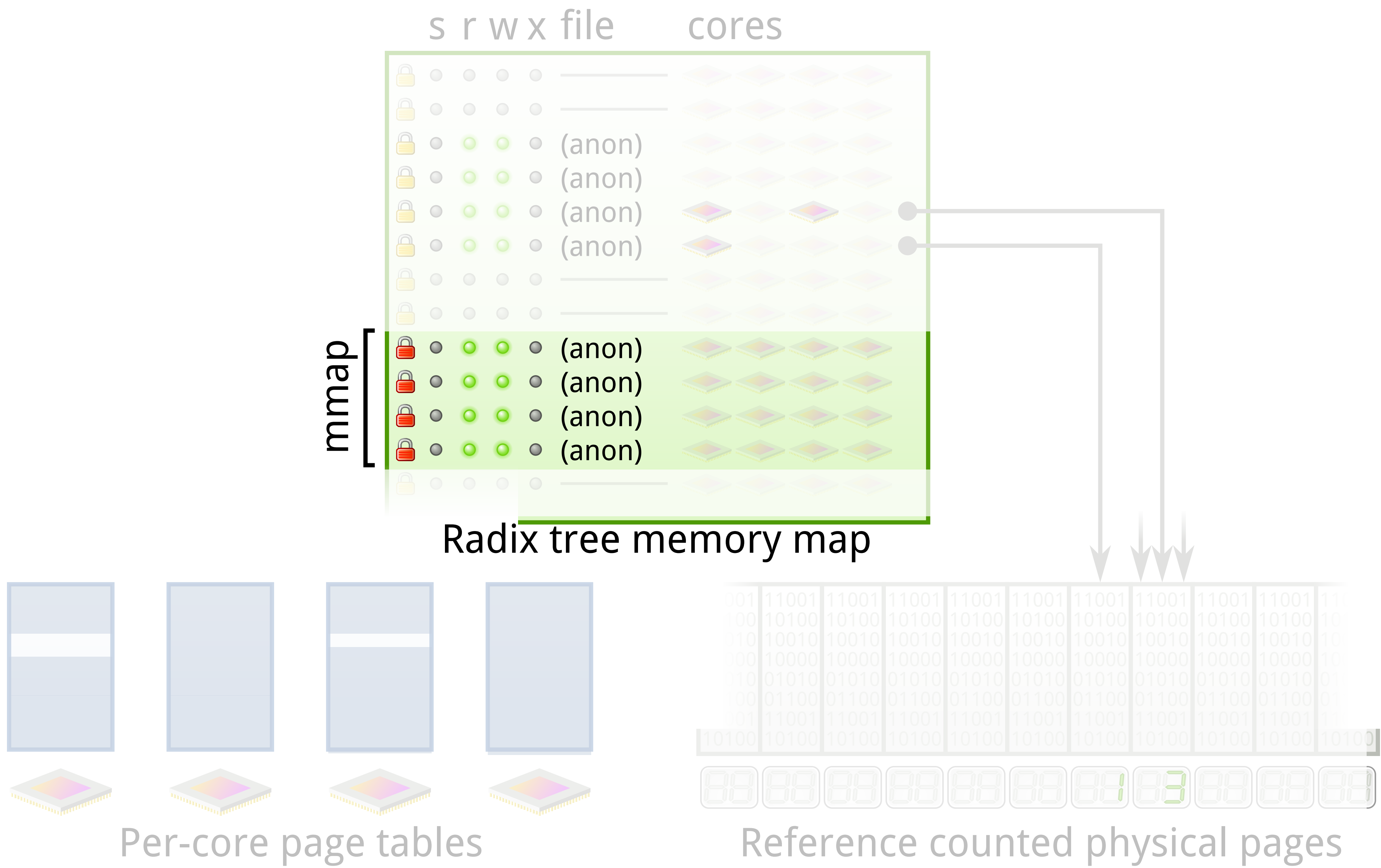
Operations in the orange region are still cached.

Abort delete

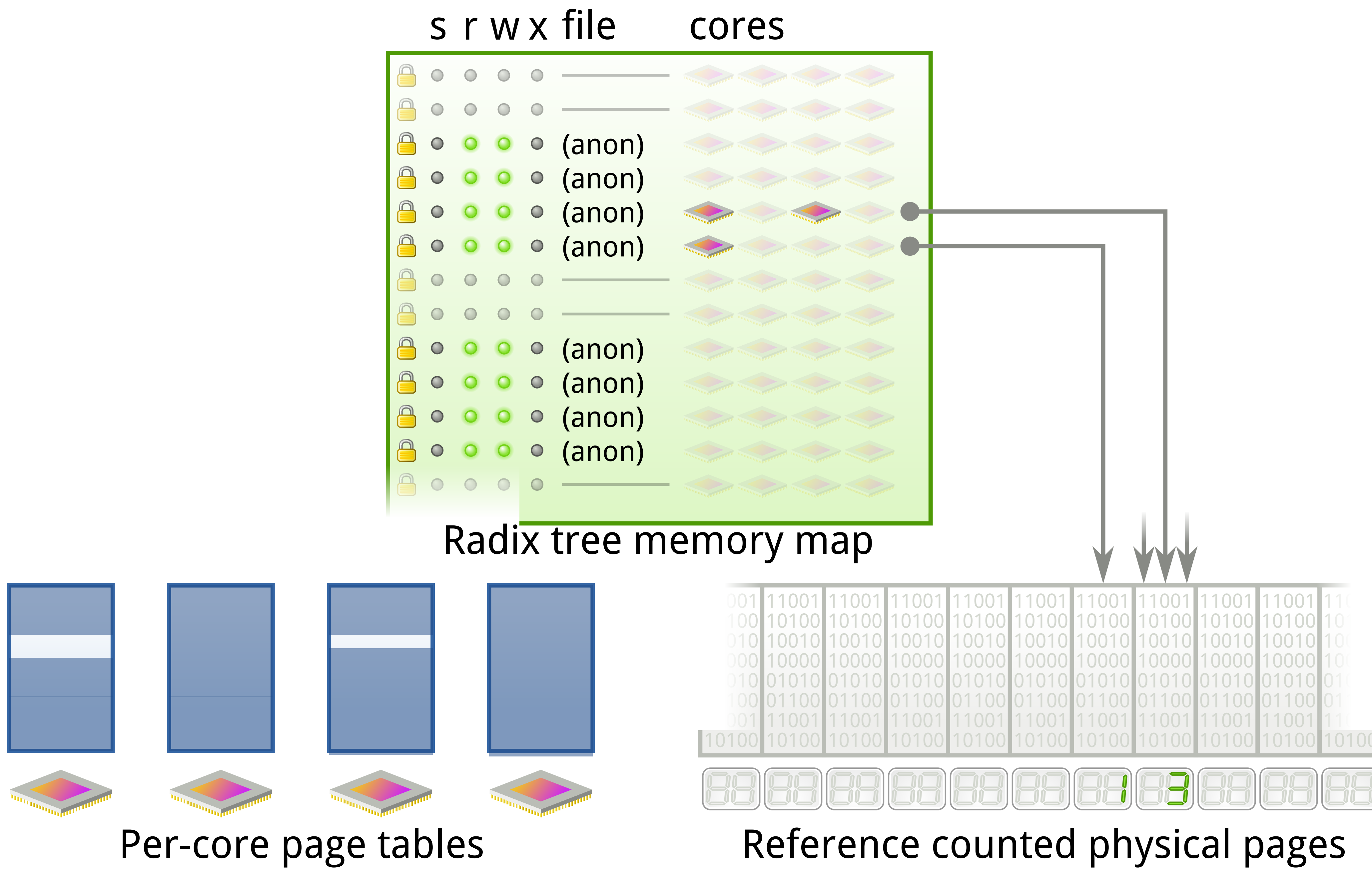
Bringing it all together



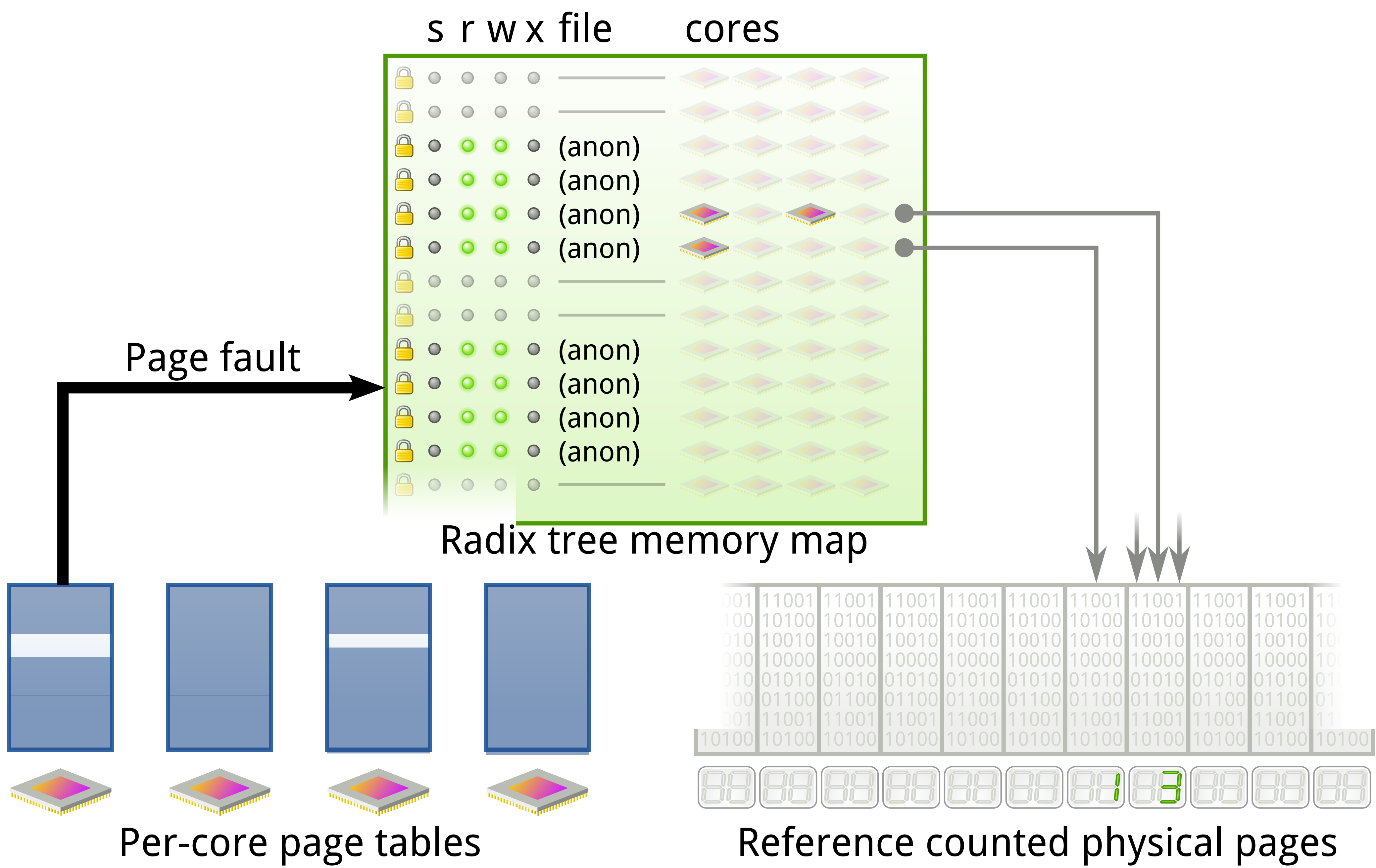
Bringing it all together



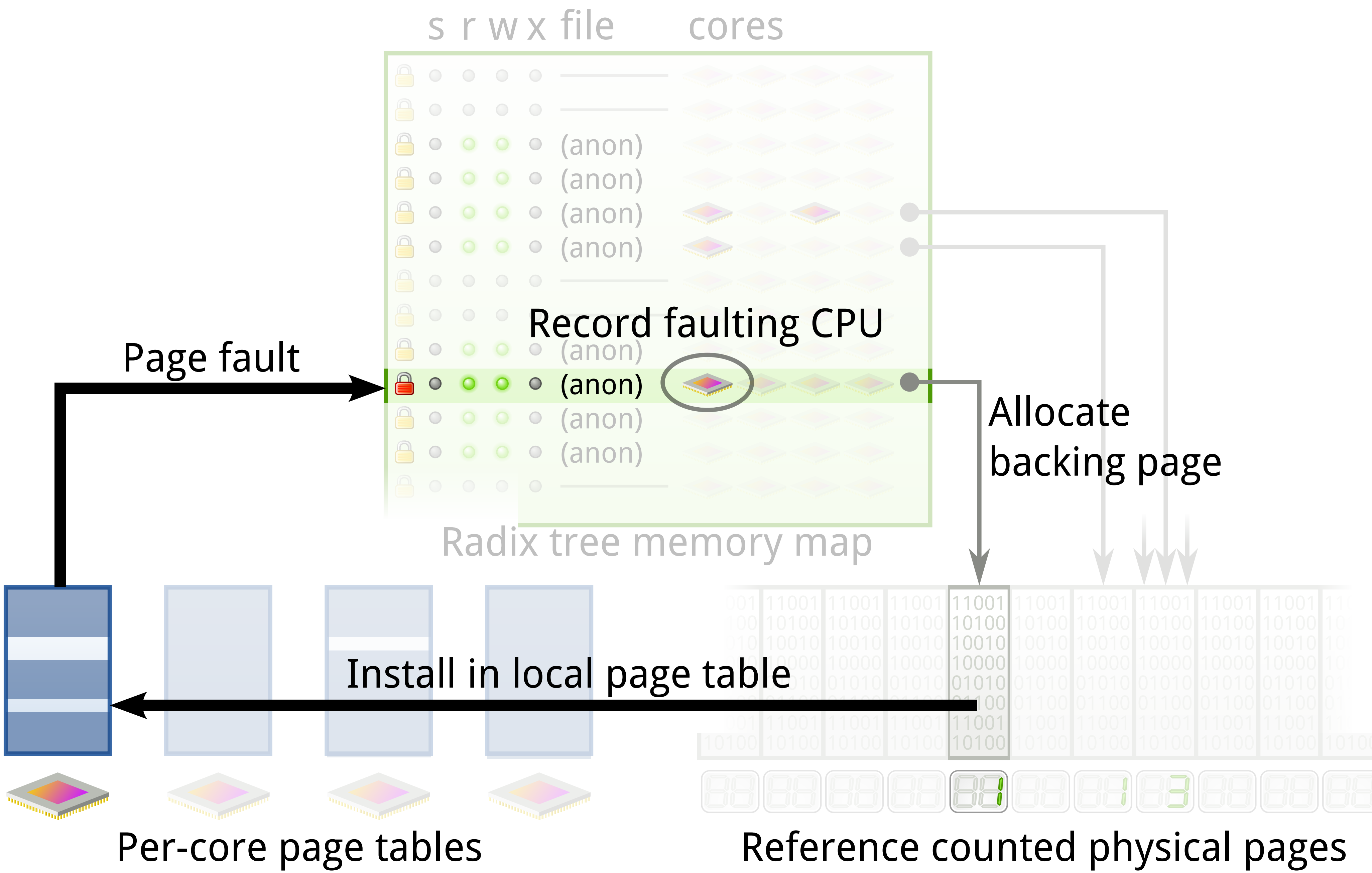
Bringing it all together



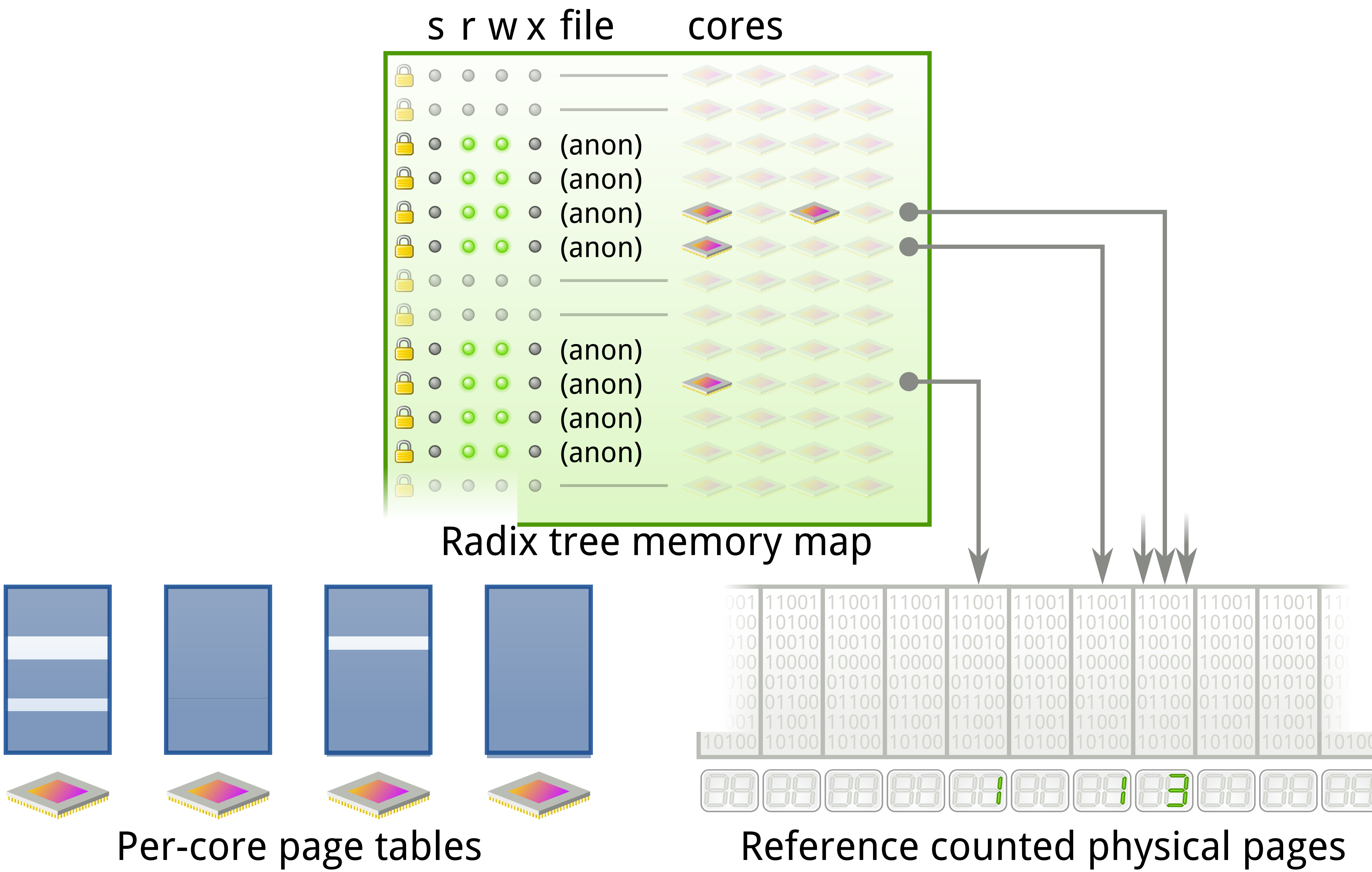
Bringing it all together



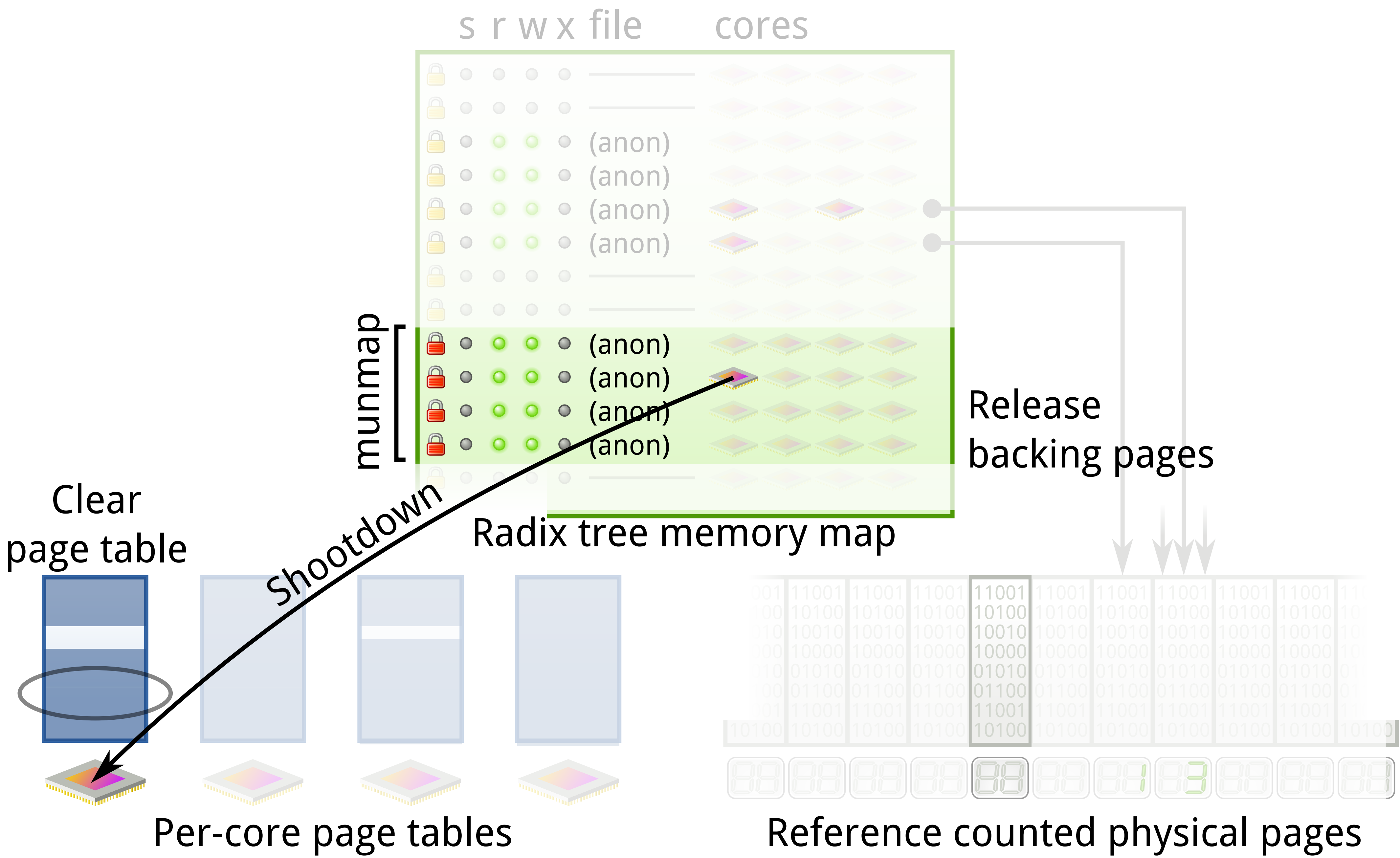
Bringing it all together



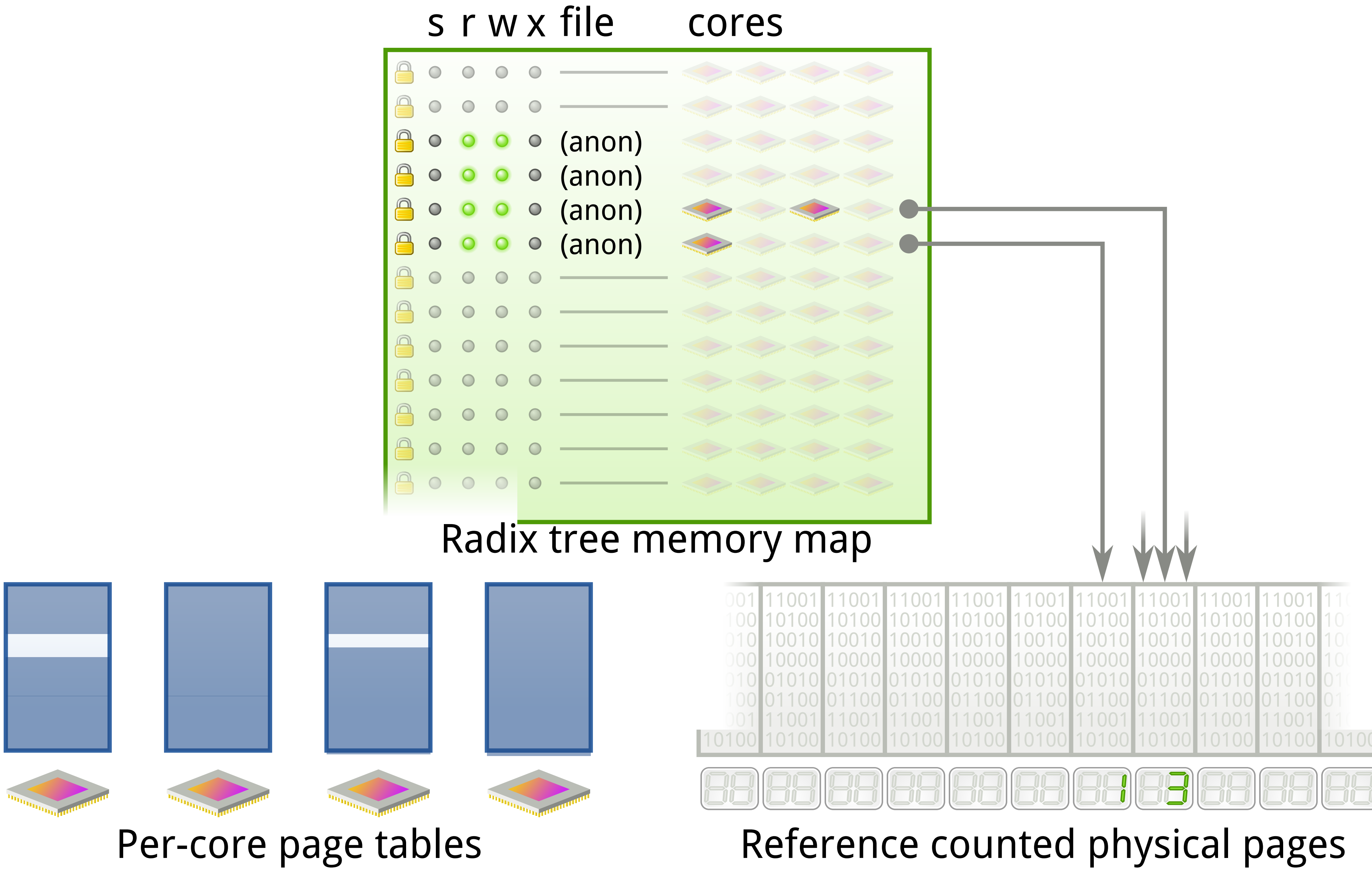
Bringing it all together



Bringing it all together



Bringing it all together



Implementation

We built RadixVM in a custom research kernel.

We believe RadixVM could be built in a mainstream kernel.

All benchmarks are source-compatible with Linux.

The other 99% is perspiration

Booting 80 cores (ACPI, x2APIC, IOMMU, oh my!)

NUMA-aware everything (memory allocation, per-CPU data, etc)

Performance analysis tools (NMI profiling, PEBS, load latency profiling, statistics counters)

Hardware curve balls (false sharing, bad prefetch behavior, etc)

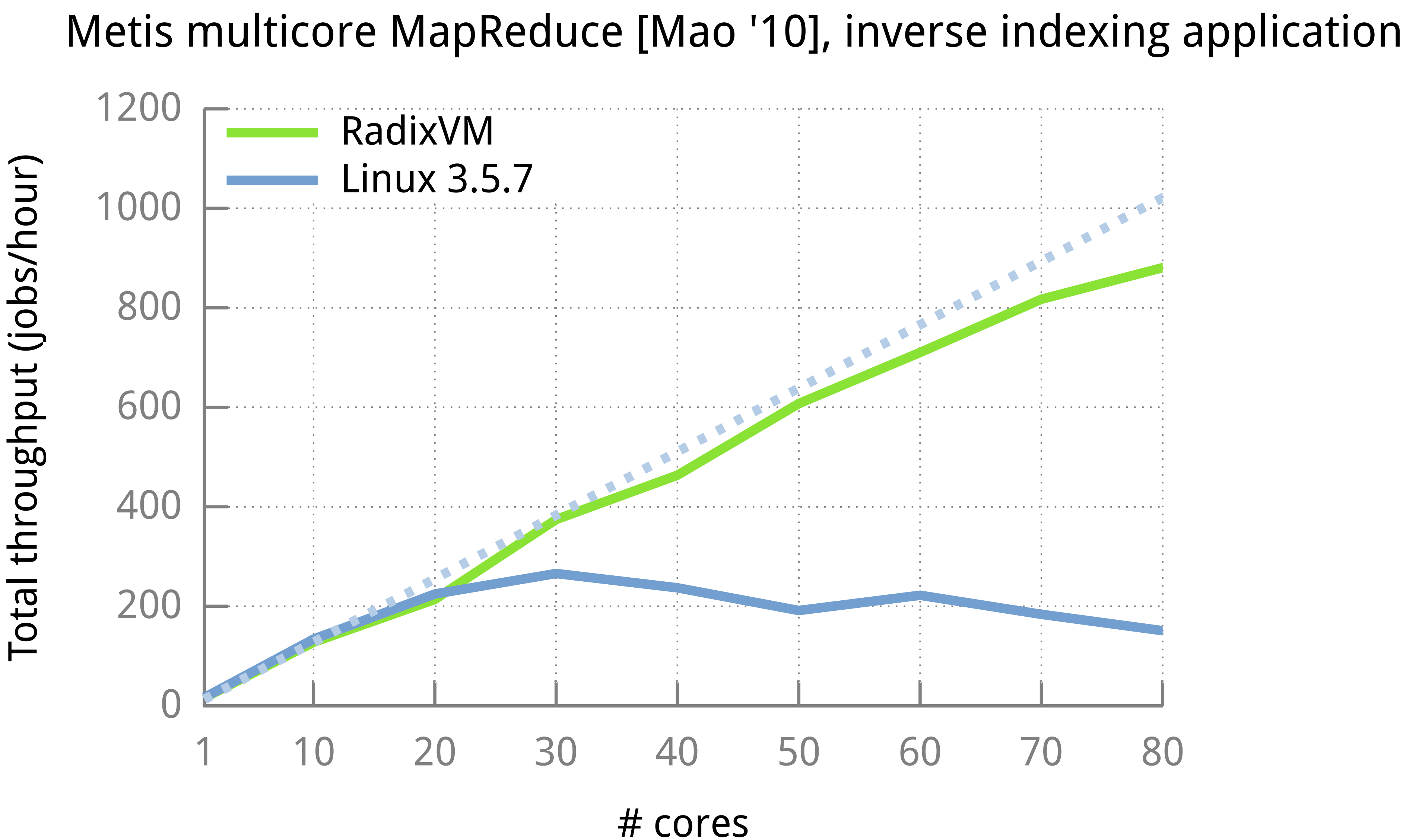
All necessary for good results; all standard engineering.

Evaluation

Does parallel mmap/munmap matter to applications?

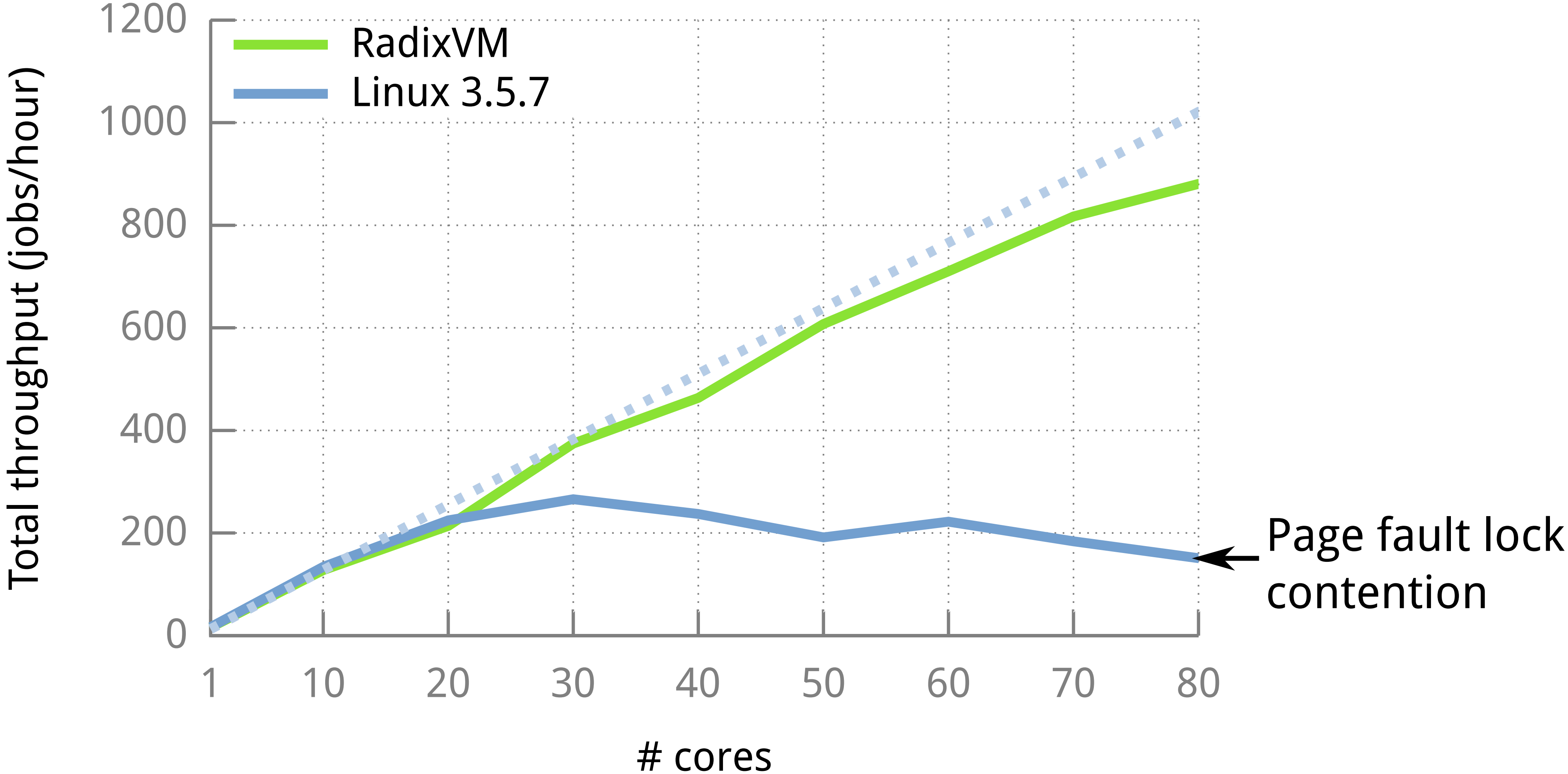
Are all of RadixVM's components necessary for scalability?

RadixVM improves application scalability

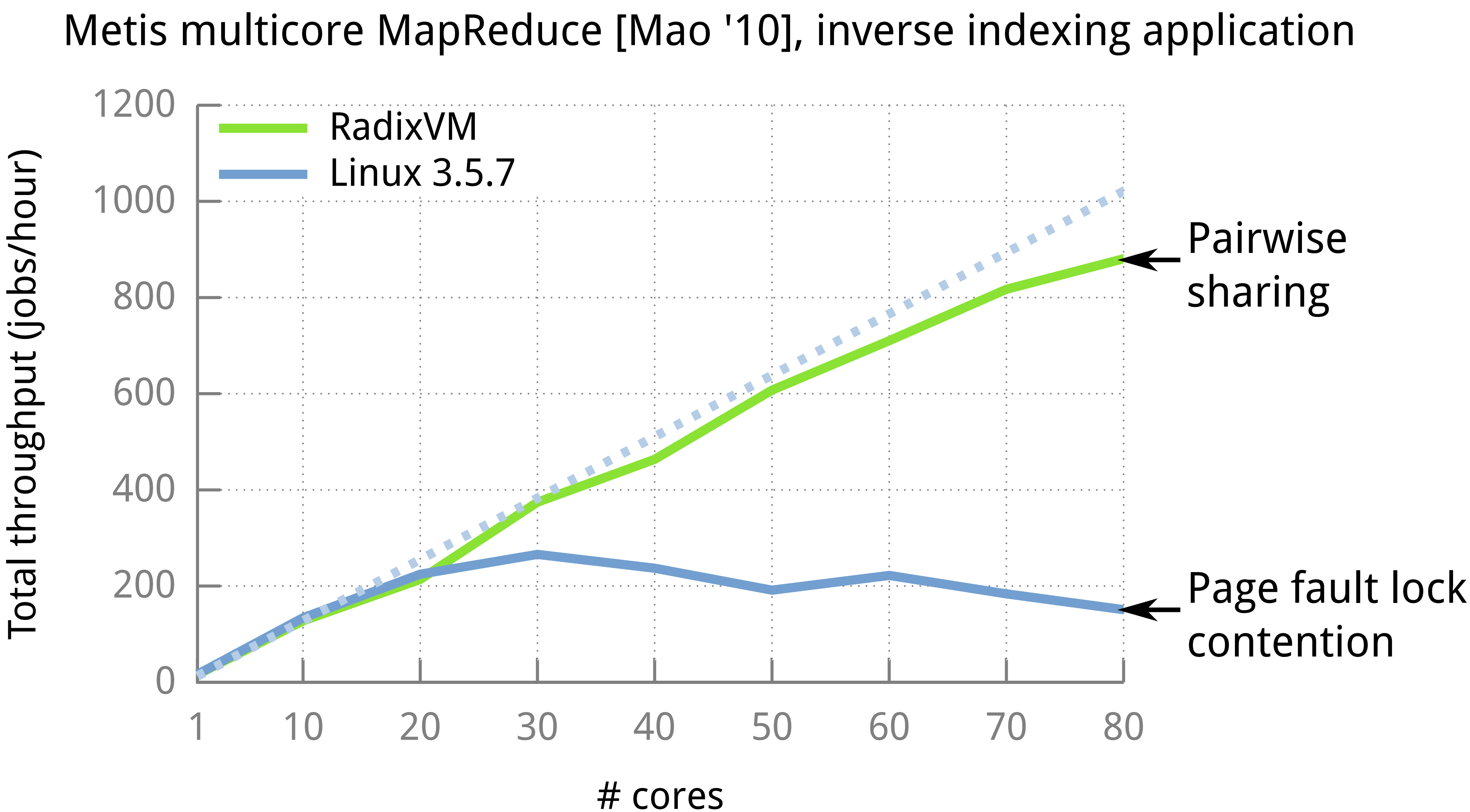


RadixVM improves application scalability

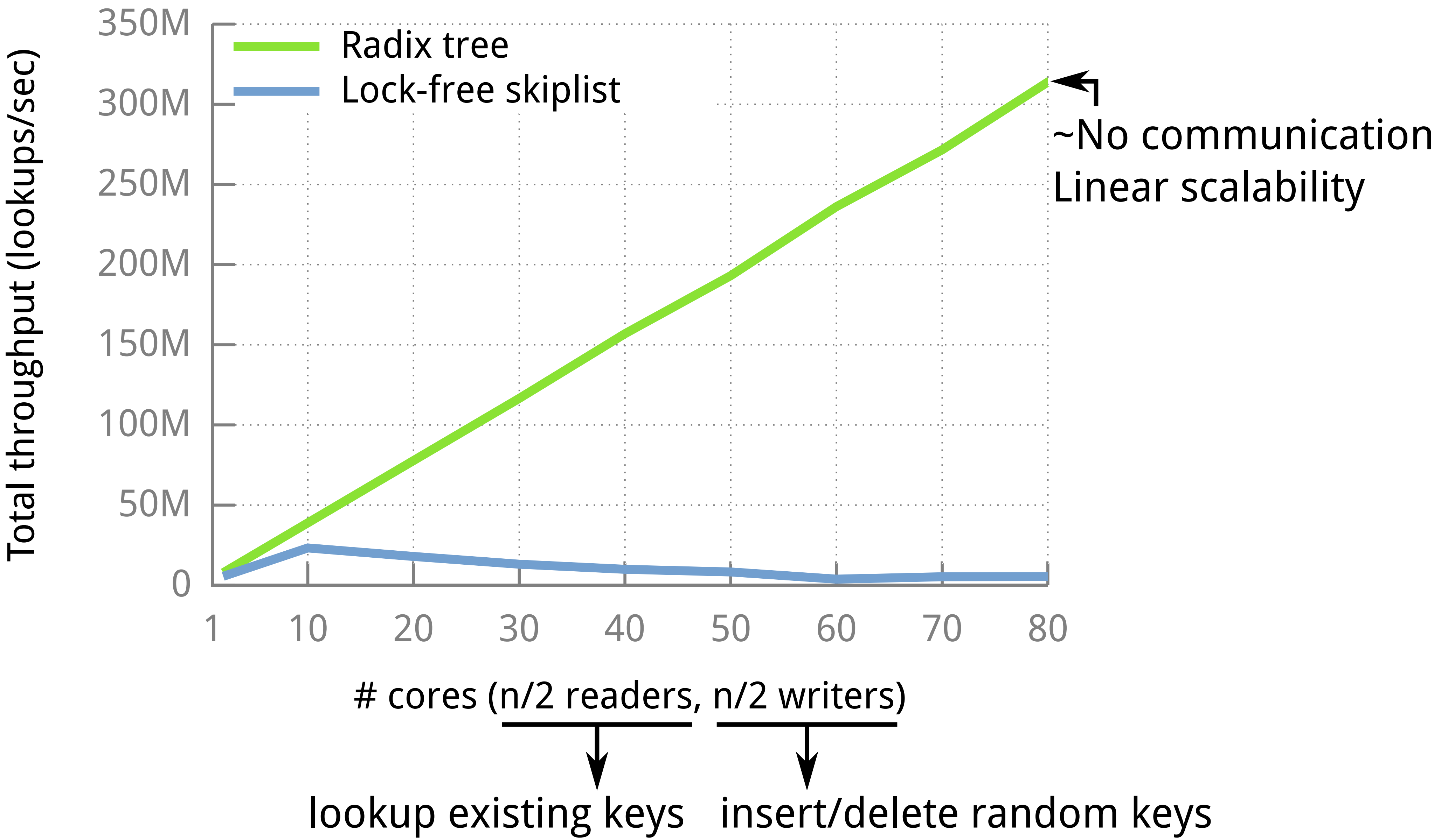
Metis multicore MapReduce [Mao '10], inverse indexing application



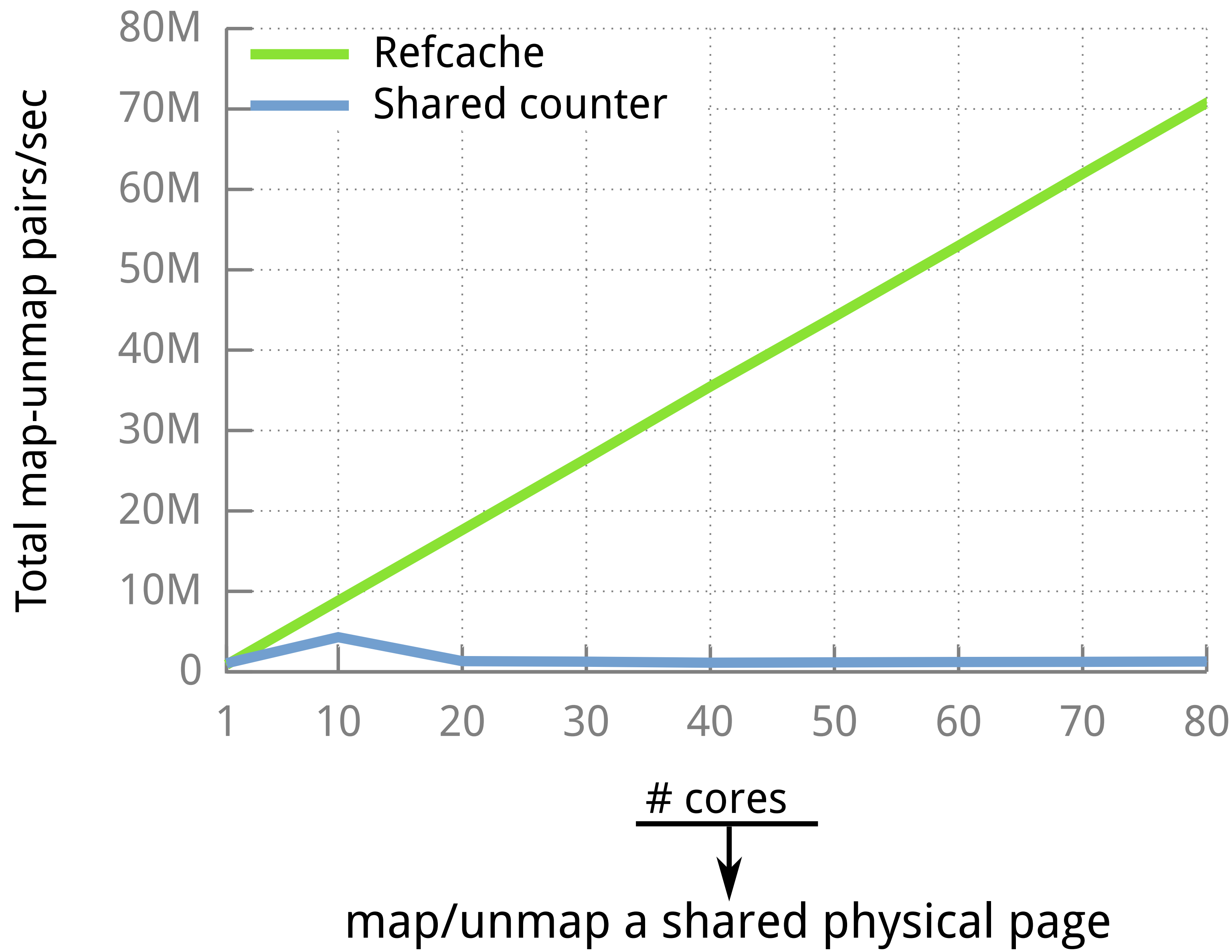
RadixVM improves application scalability



Radix trees avoid communication

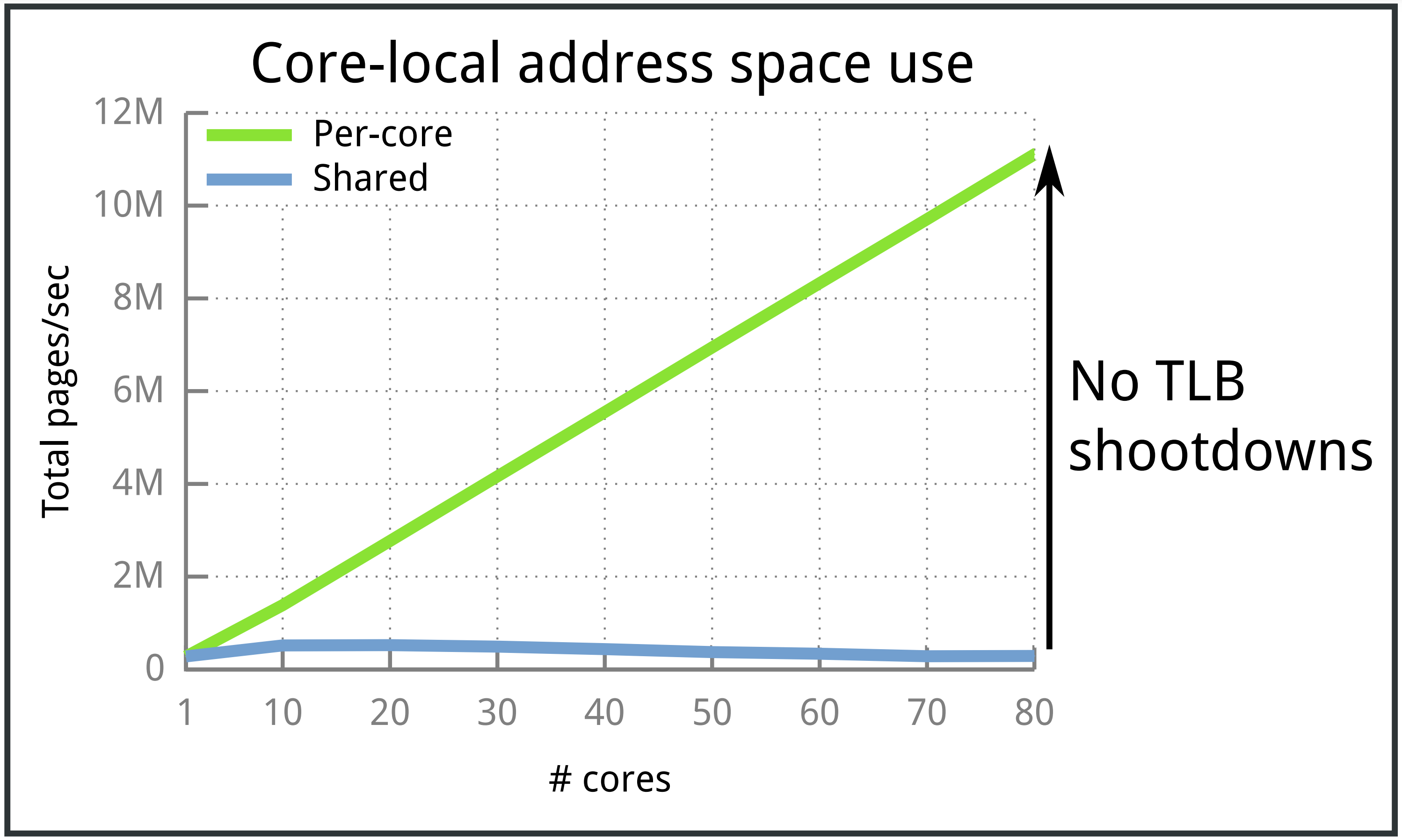


Refcache avoids cache line sharing

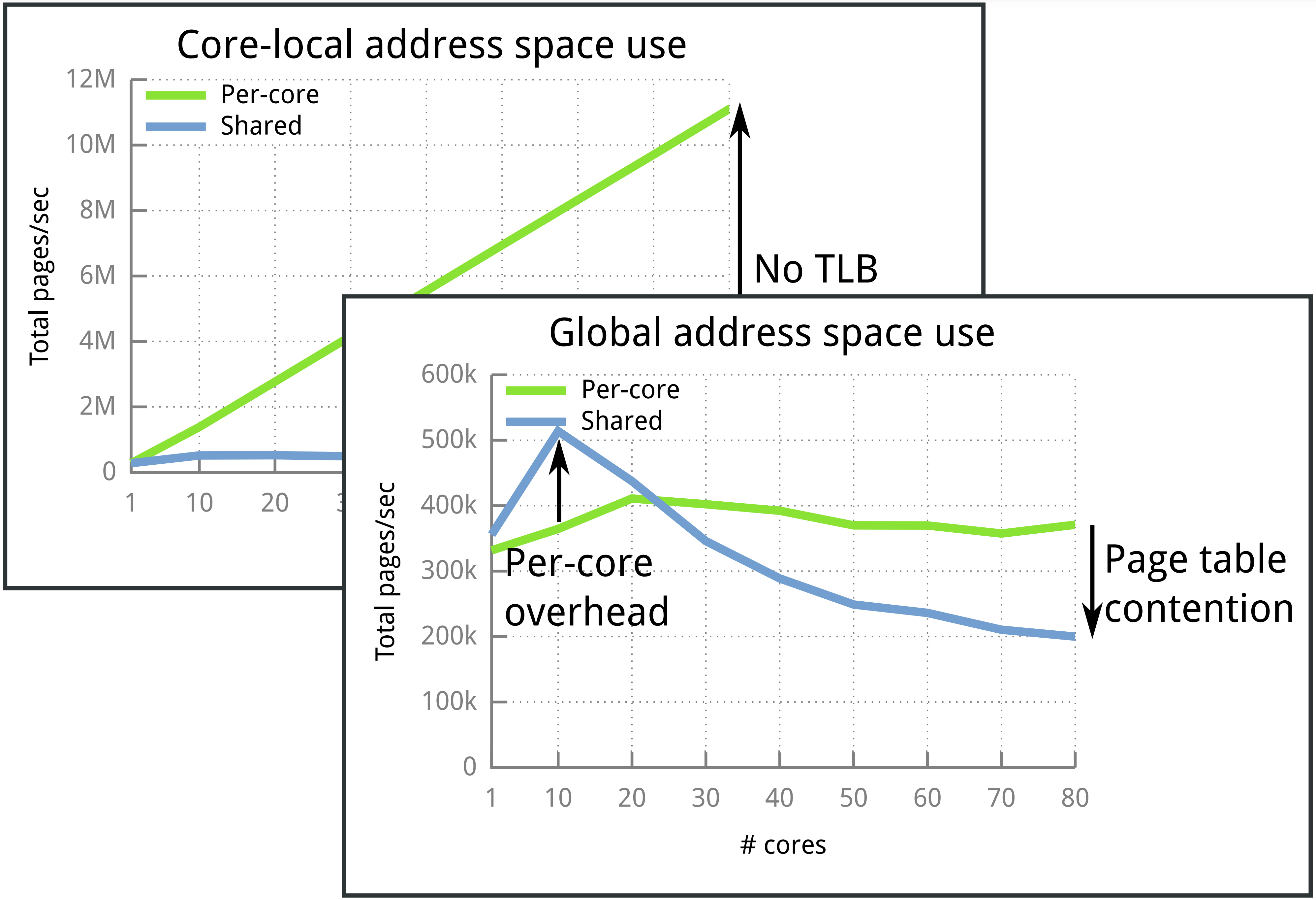


Targeted TLB shootdown improves scalability

Targeted TLB shootdown improves scalability



Targeted TLB shootdown improves scalability



Related work

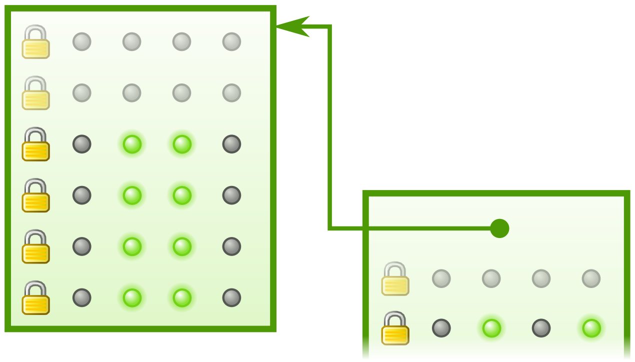
Scalable VM systems

- K42 [Krieger '06]
- Corey [Boyd-Wickizer '08]
- Bonsai [Clements '12]

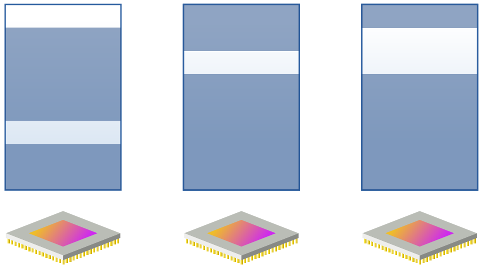
Scalable reference counters

- Modula-2+ local refs [DeTreville '90]
- Distributed counters [Appavoo '07]
- Scalable non-zero indicators [Ellen '07]
- Sloppy counters [Boyd-Wickizer '10]

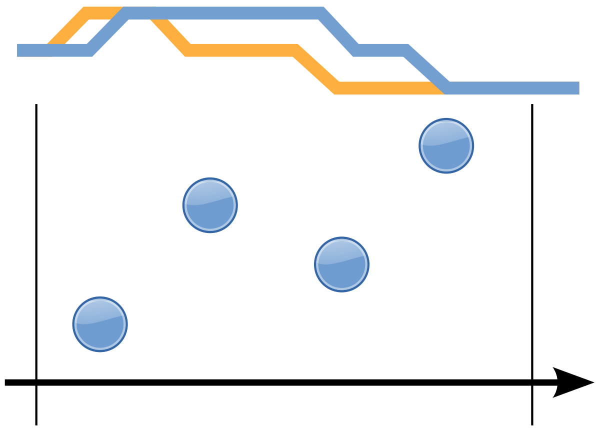
Conclusion



Radix trees

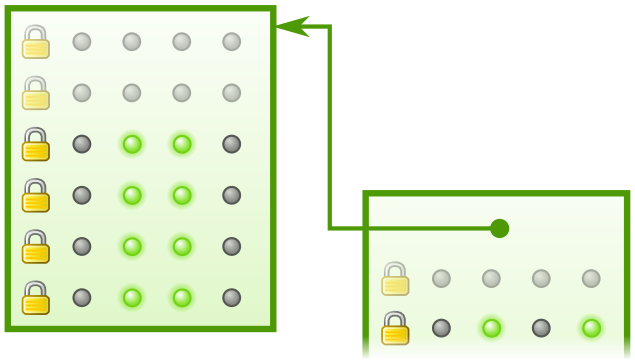


Per-core page tables

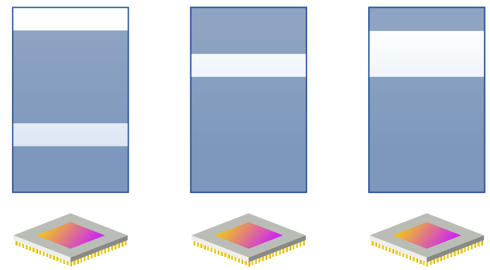


Refcache

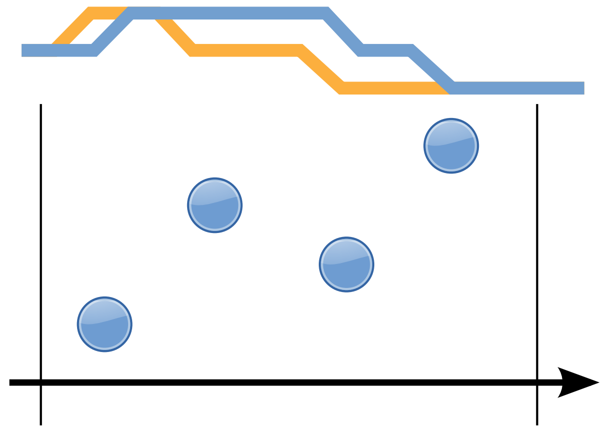
Conclusion



Radix trees



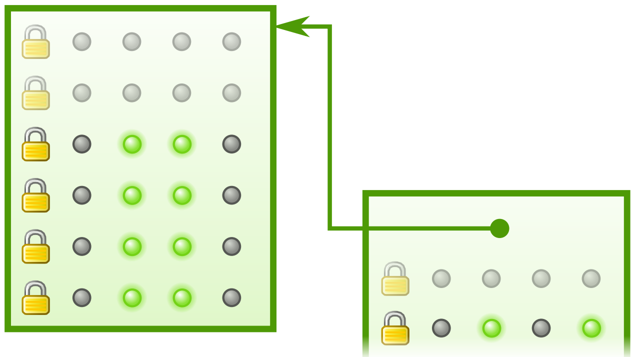
Per-core page tables



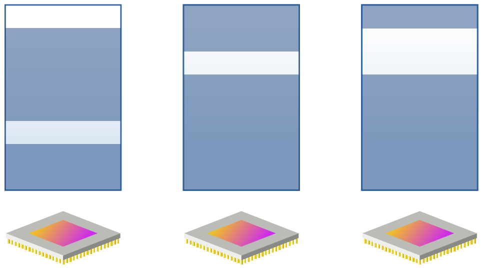
Refcache

Perfect scalability for non-overlapping VM operations

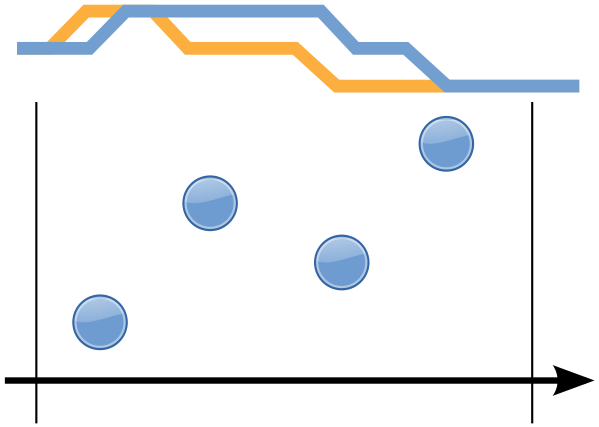
Conclusion



Radix trees



Per-core page tables



Refcache

Perfect scalability for non-overlapping VM operations

Check it out: http://pdos.csail.mit.edu/multicore