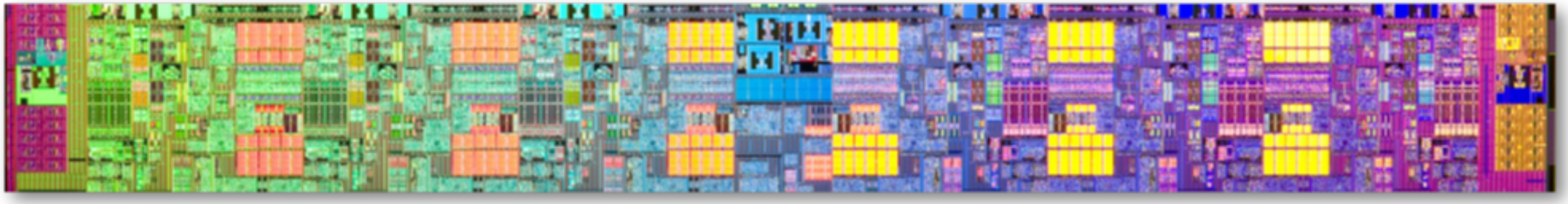
Scalable address spaces using RCU balanced trees

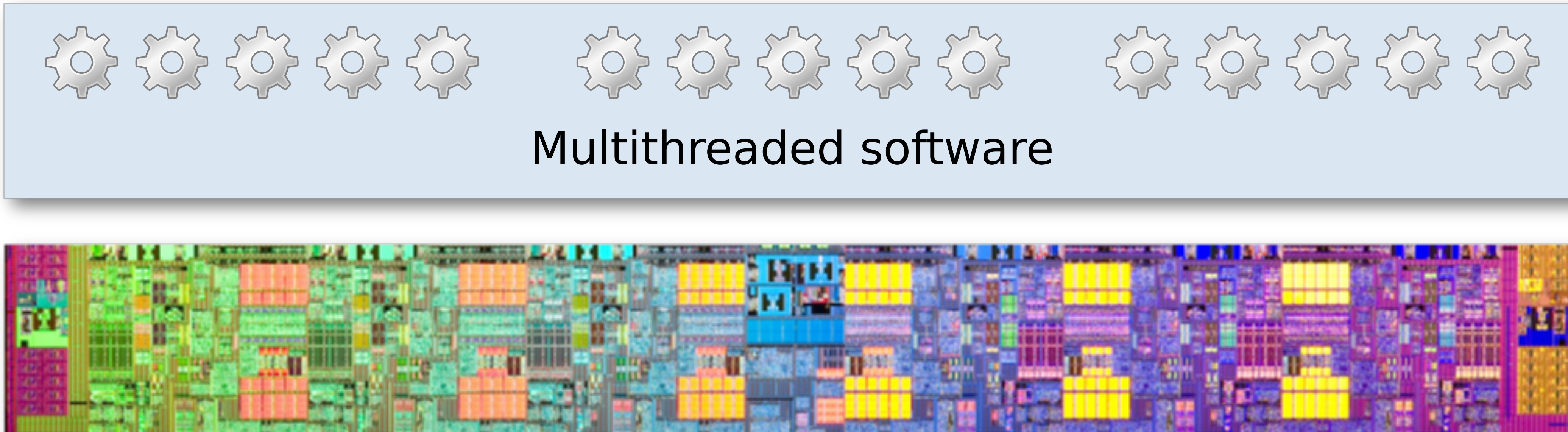
Austin T. Clements M. Frans Kaashoek Nickolai Zeldovich

MIT CSAIL

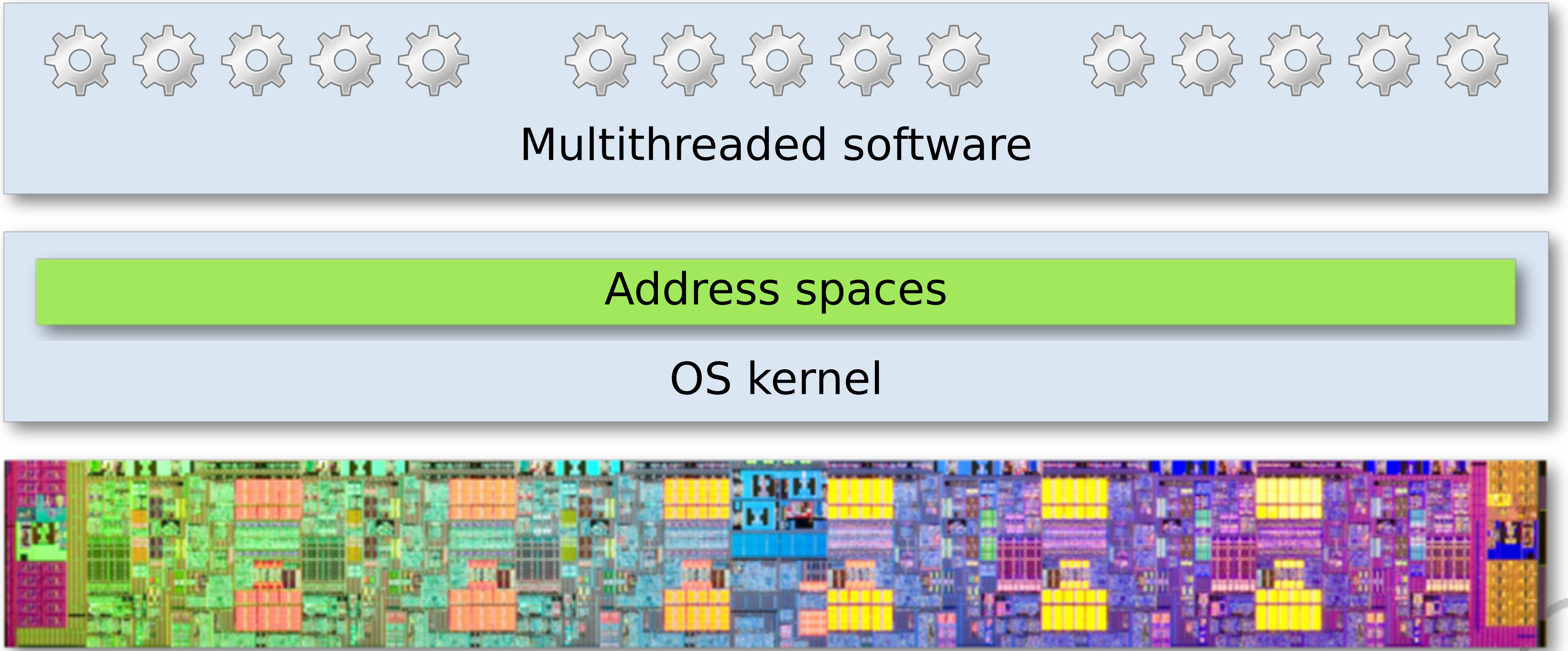
Motivation



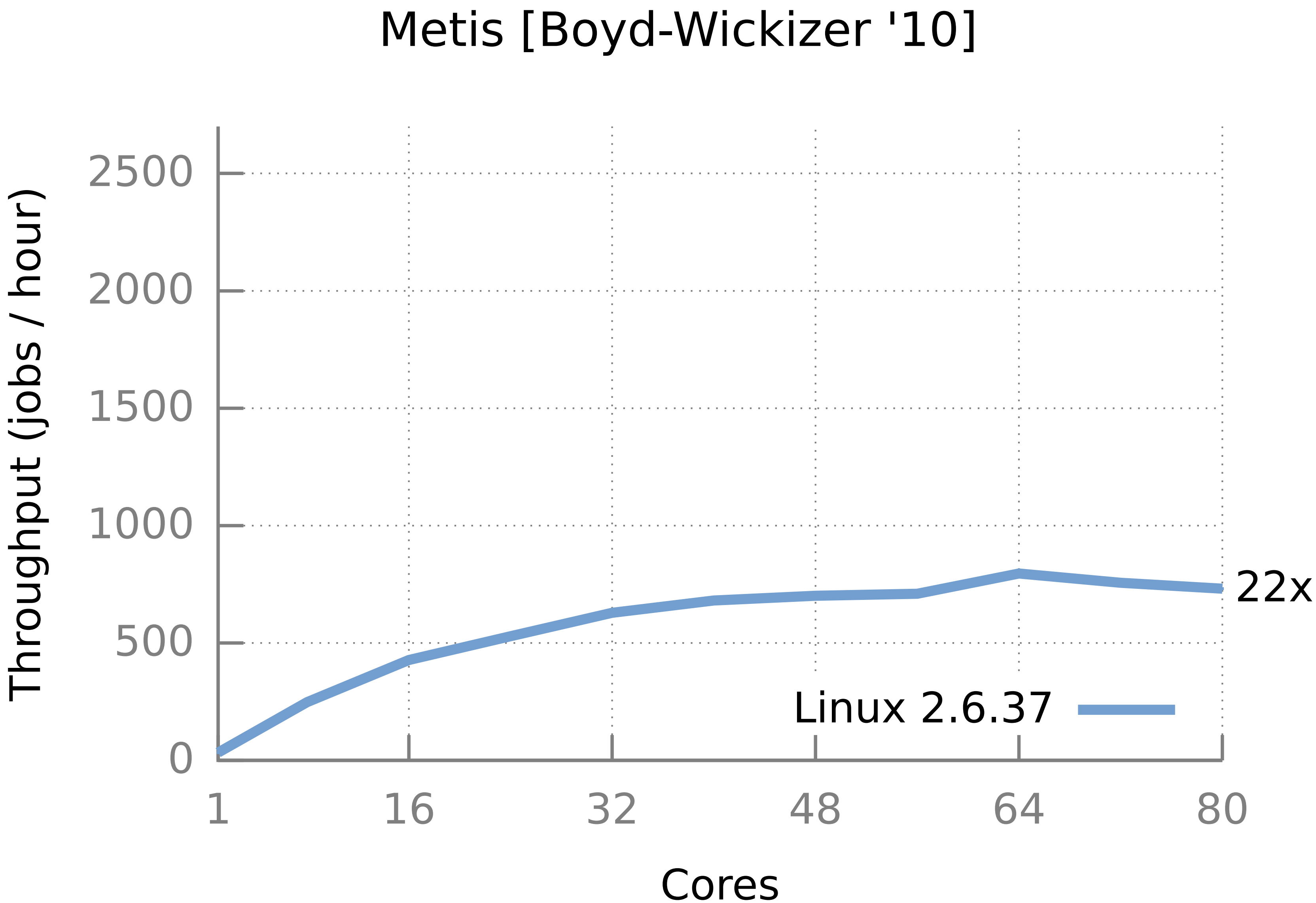
Motivation



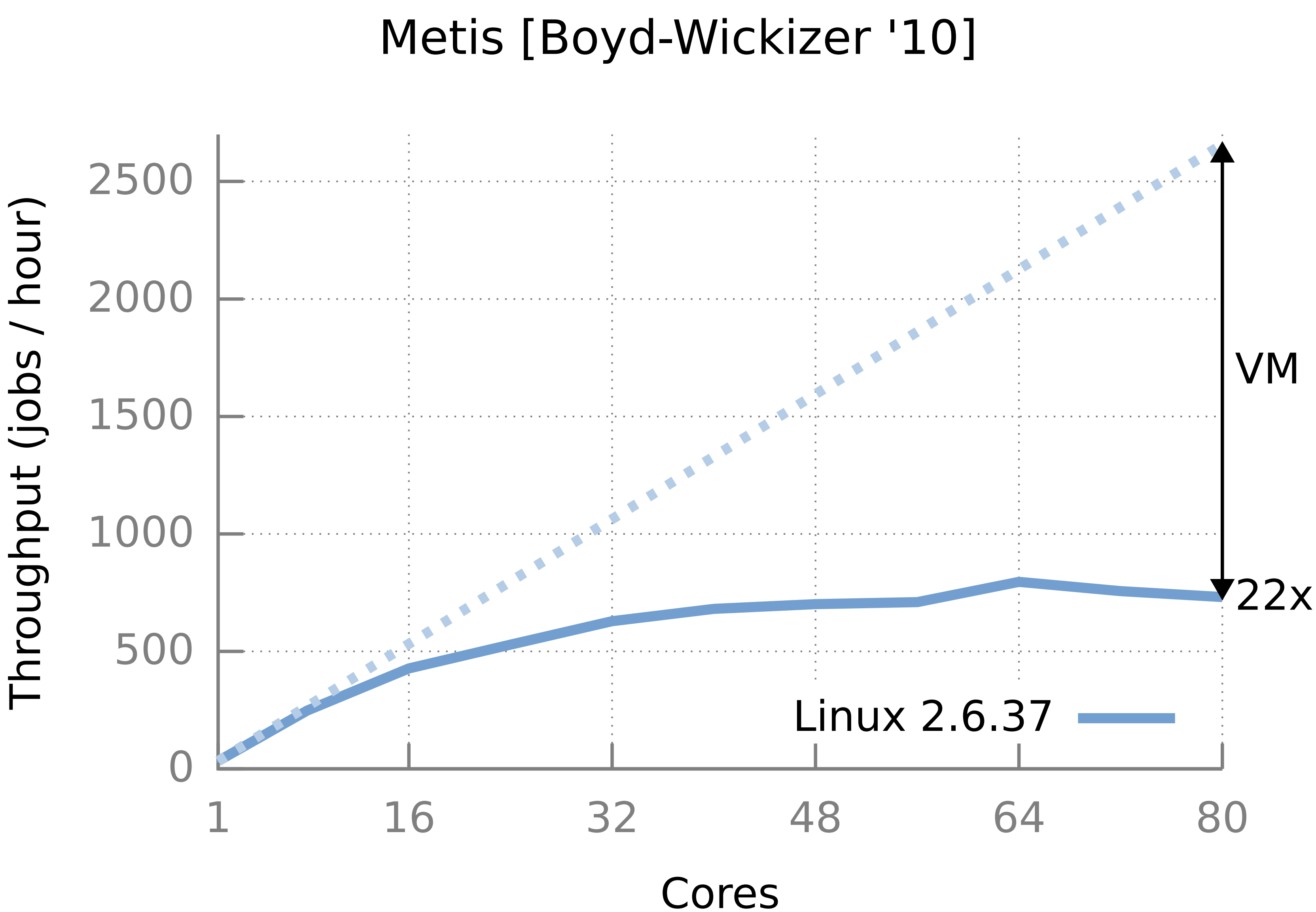
Motivation



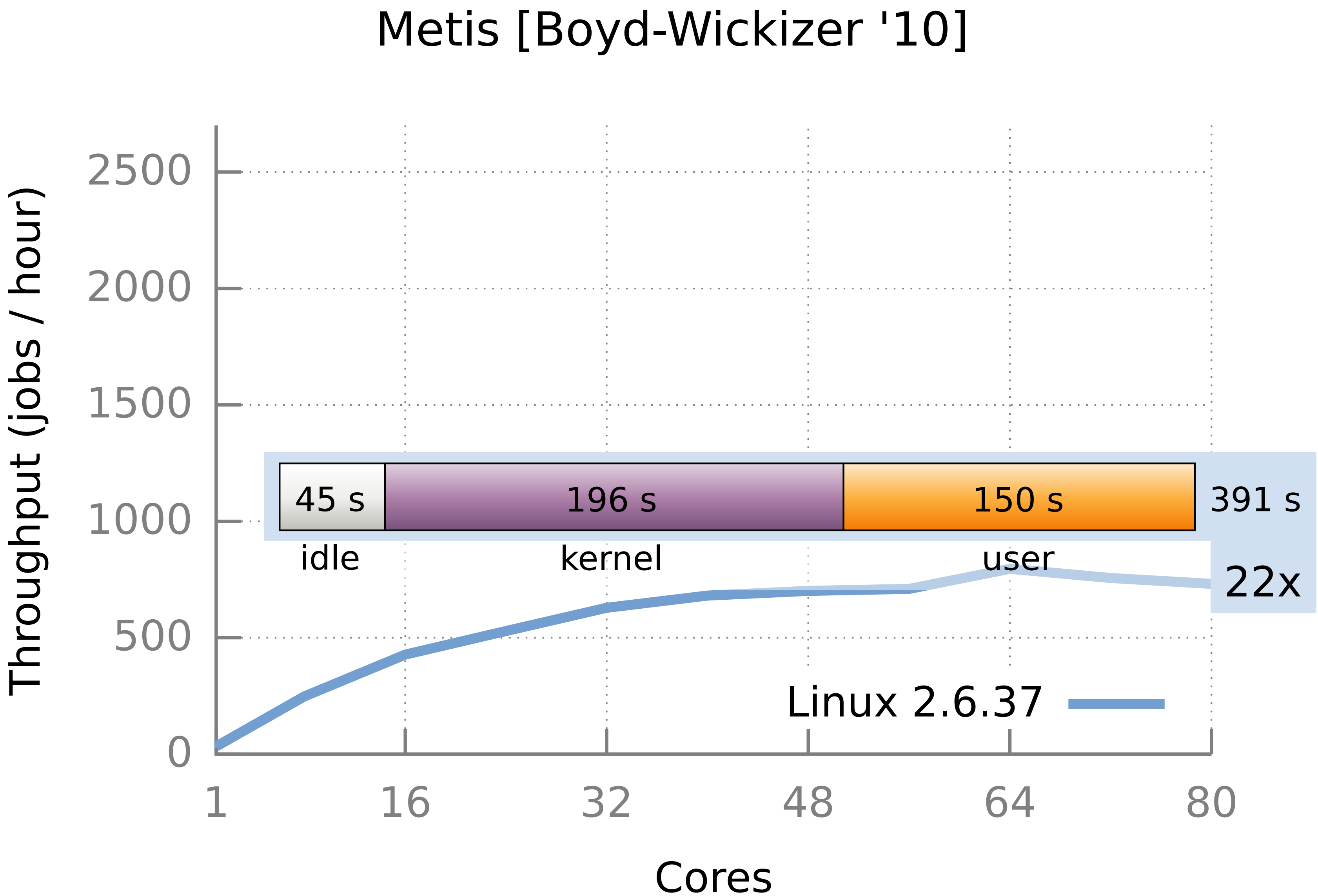
Application scalability



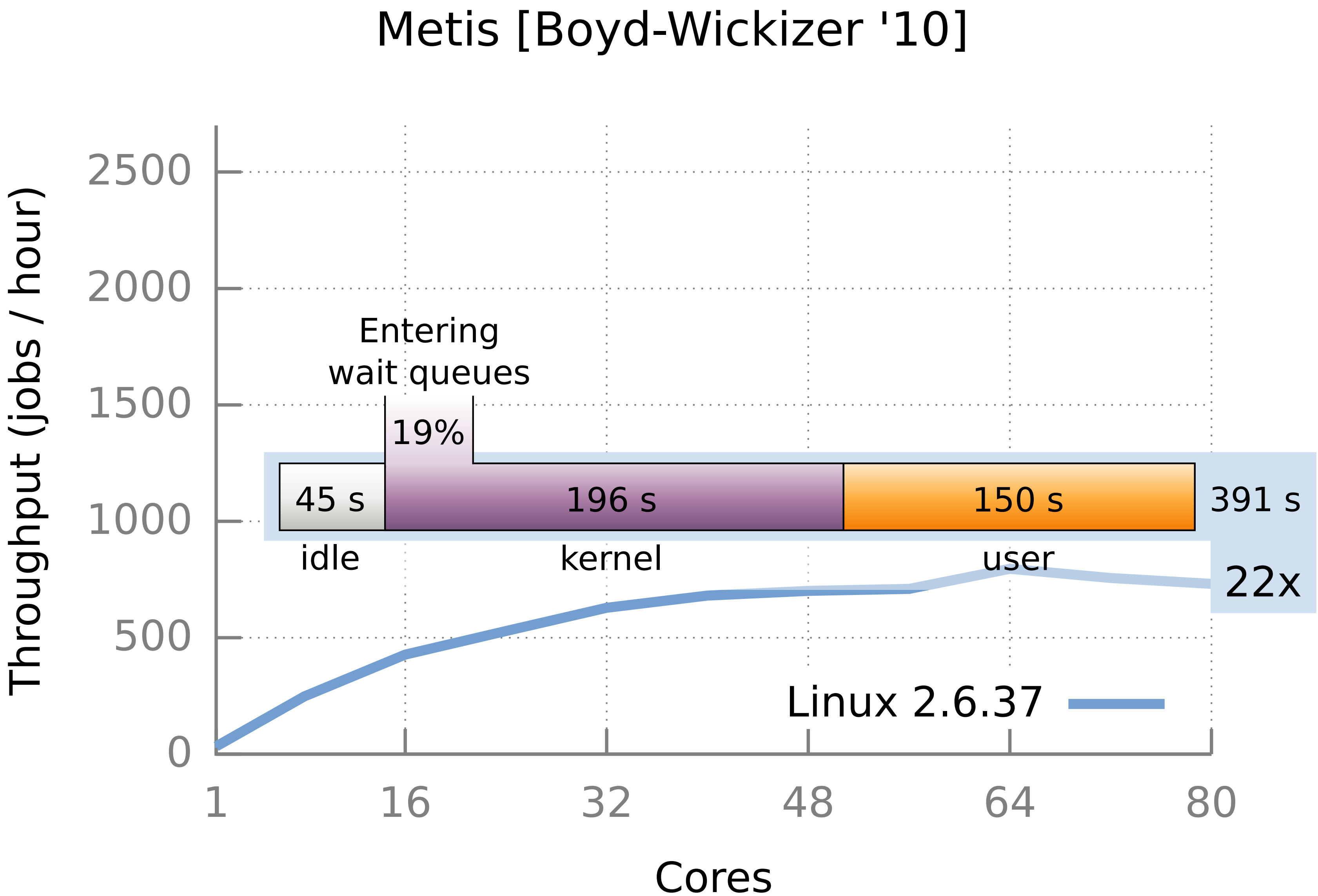
Application scalability



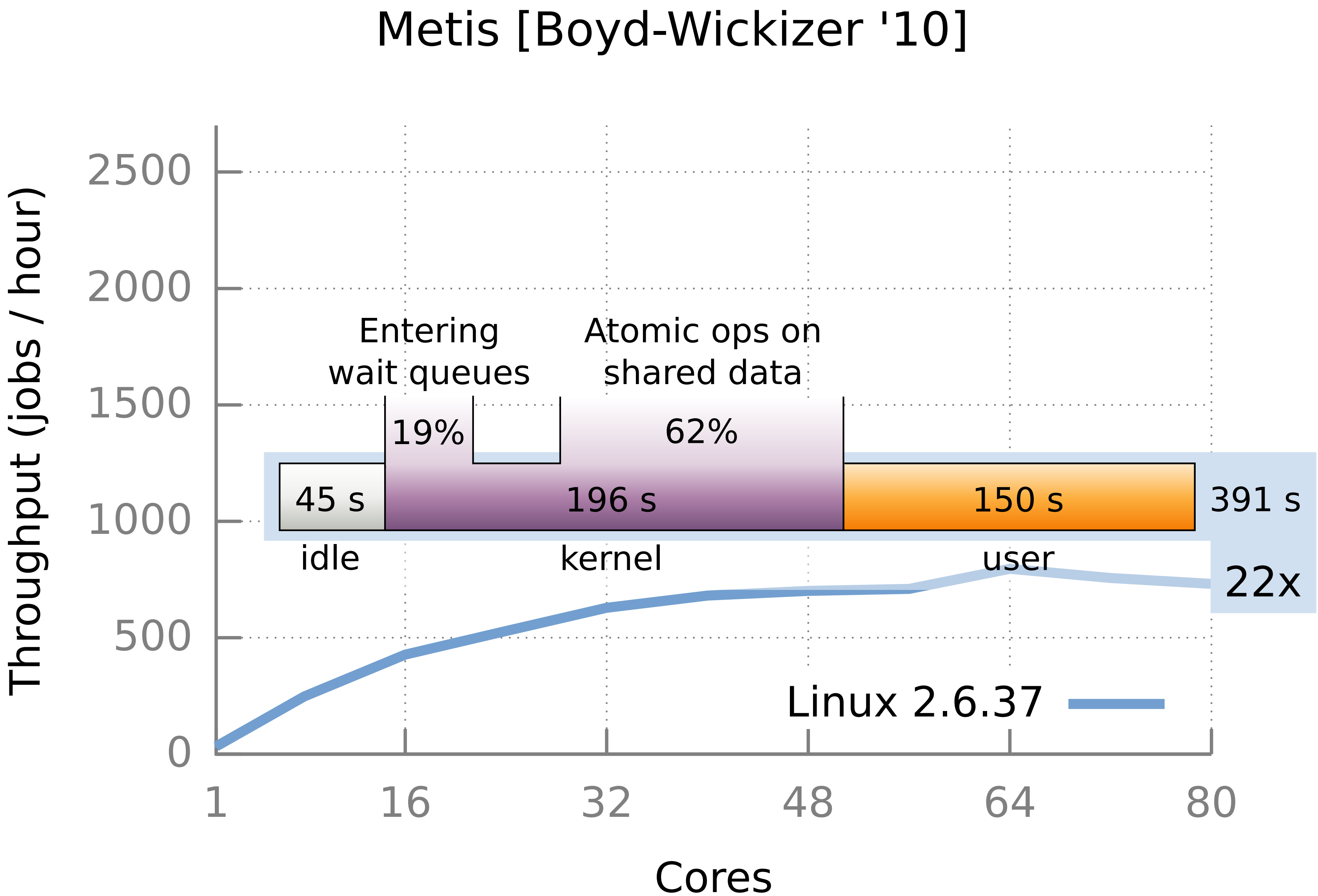
Application scalability



Application scalability



Application scalability



Goal

Enable VM-intensive multithreaded software to scale.

Address spaces

Address spaces

mmap

munmap

Address spaces

mmap

page fault

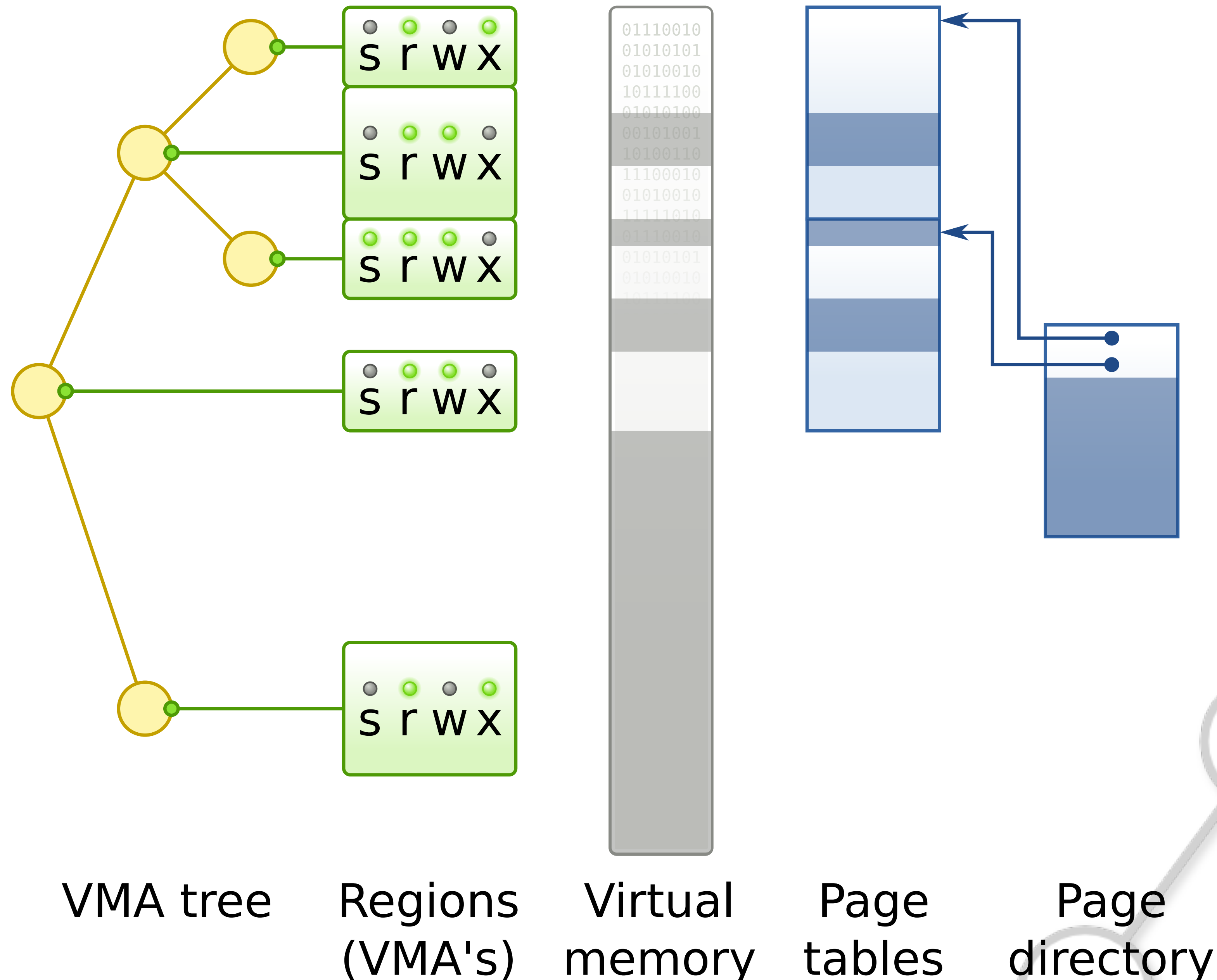
munmap

Address spaces

mmap

page fault

munmap

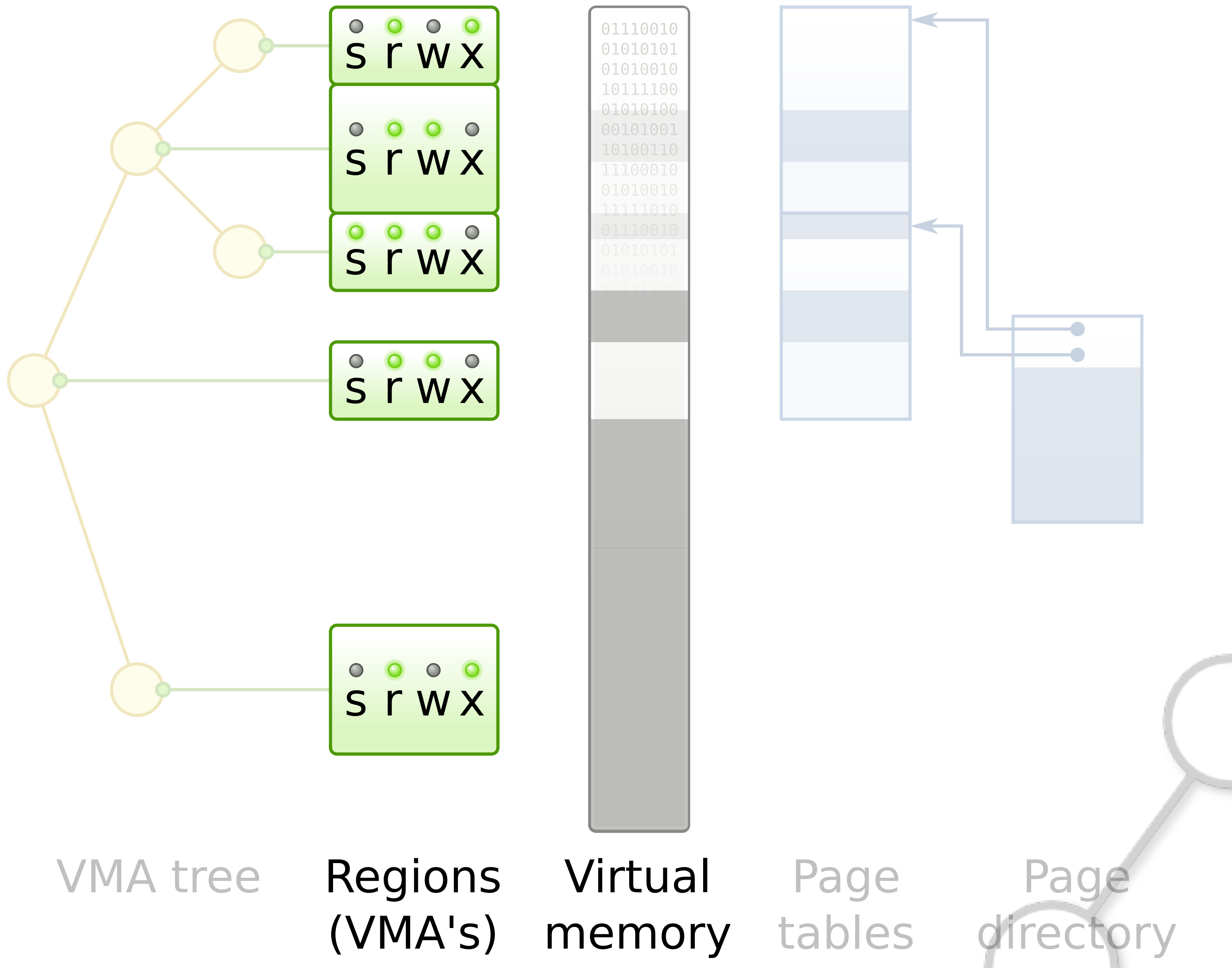


Address spaces

mmap

page fault

munmap

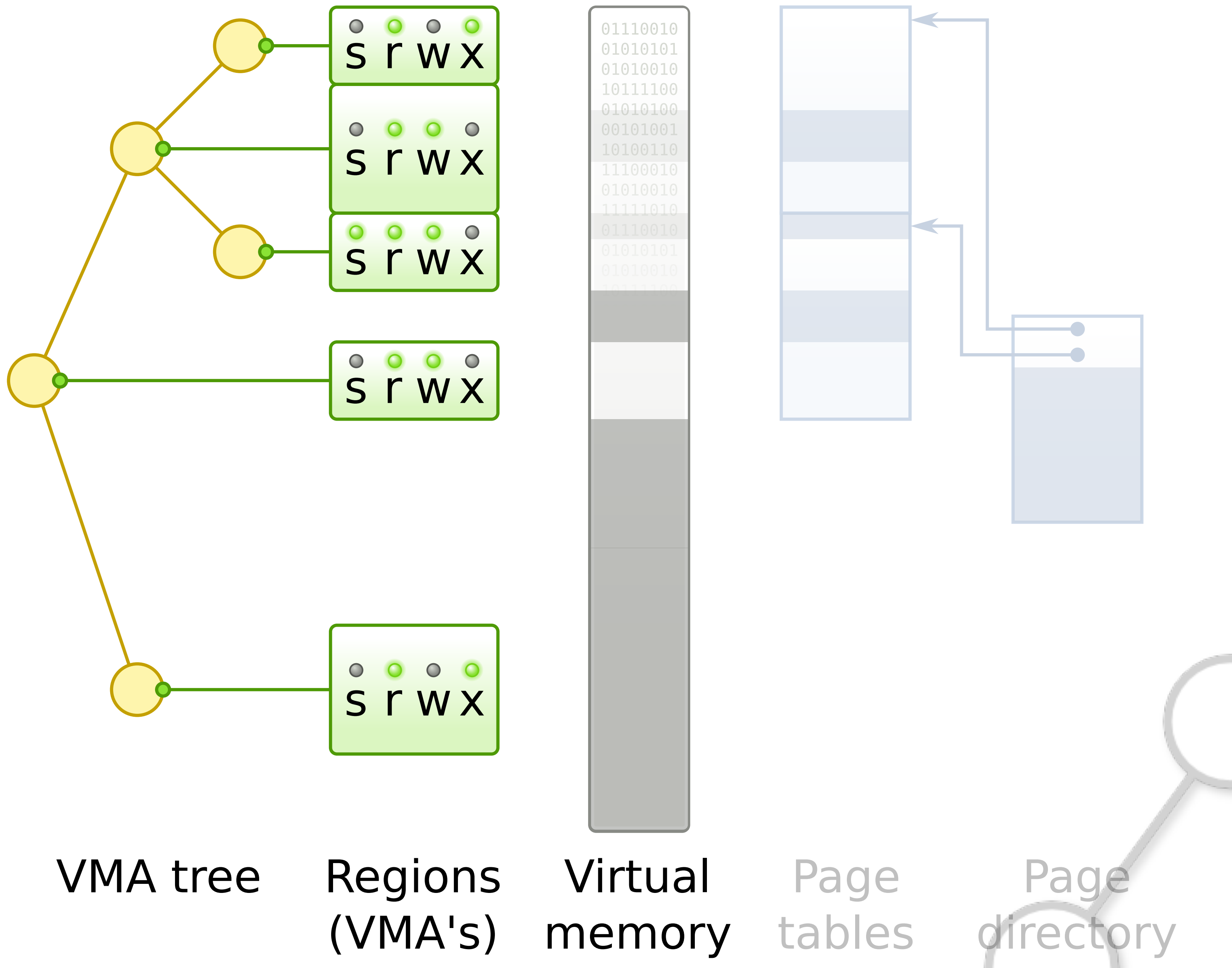


Address spaces

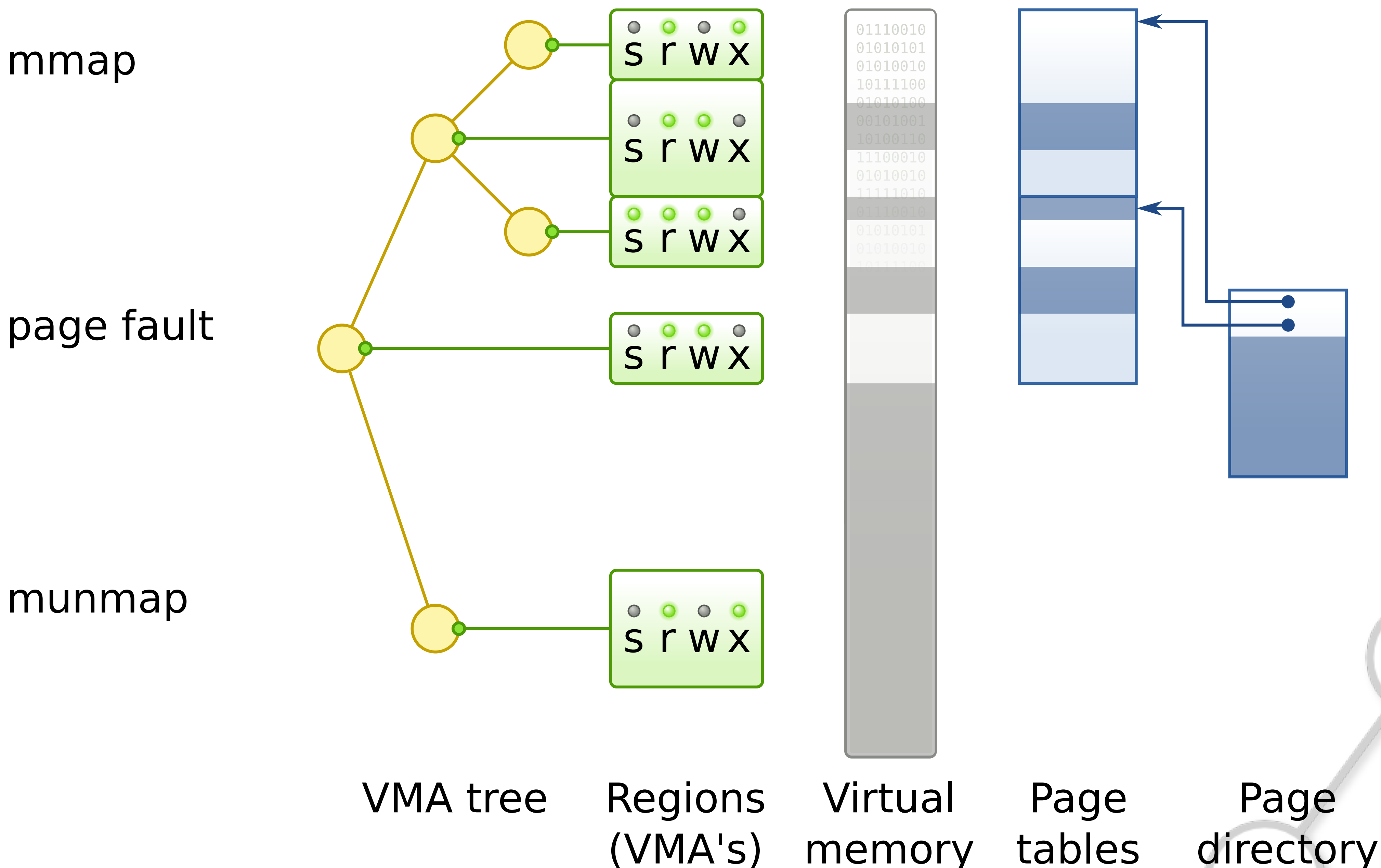
mmap

page fault

munmap



Address spaces

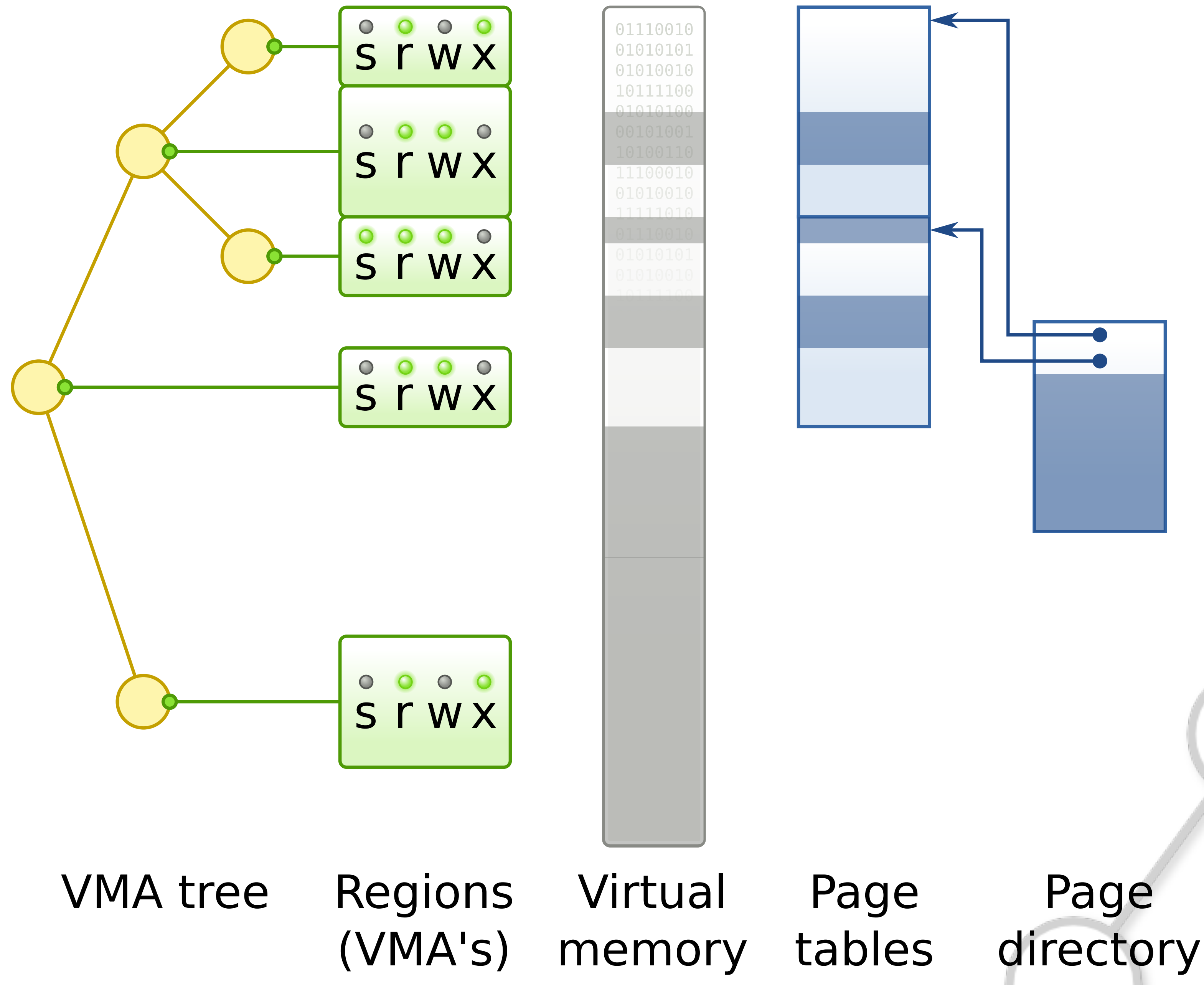


Address spaces

mmap
Creates VMAs

page fault

munmap

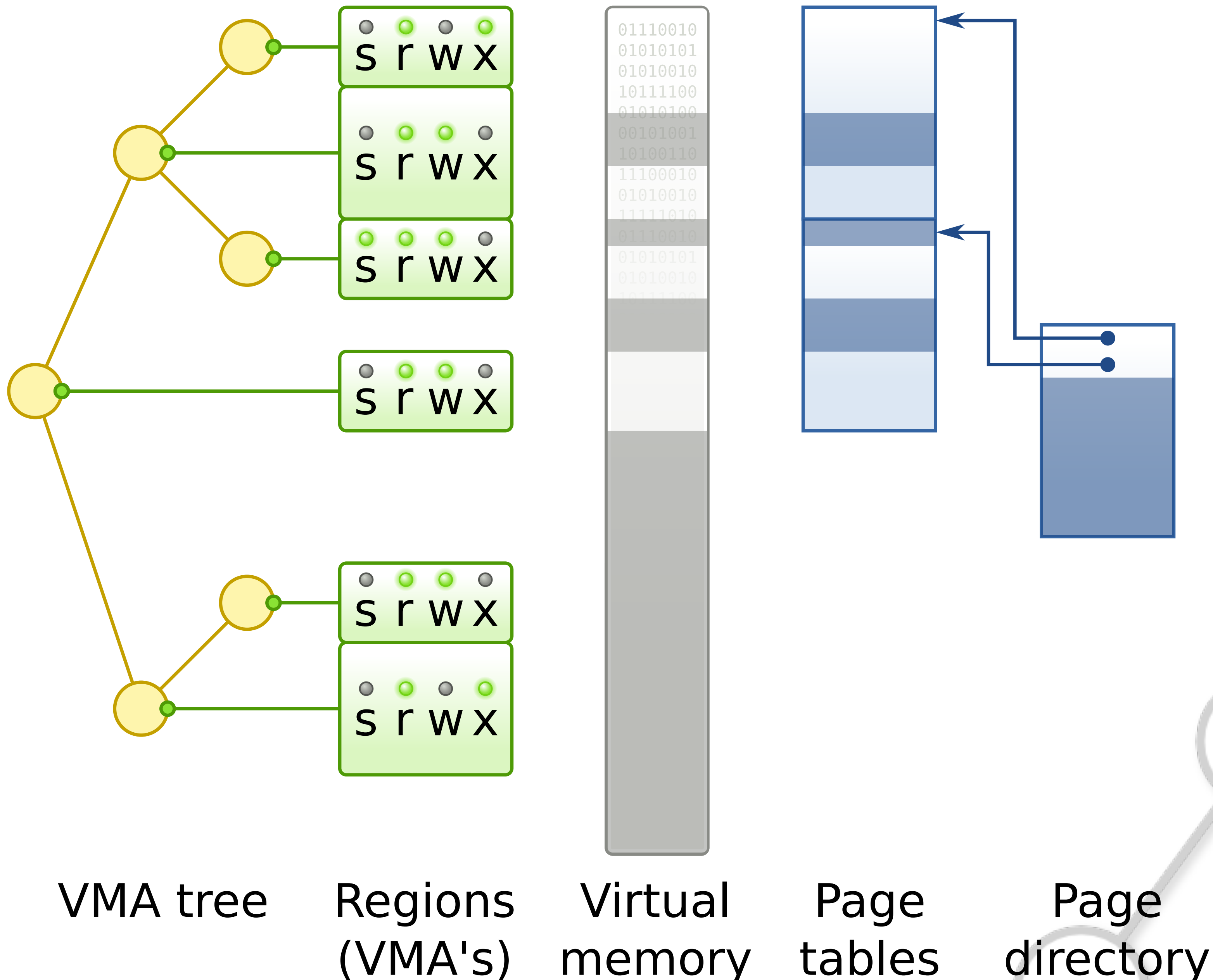


Address spaces

mmap
Creates VMAs

page fault

munmap

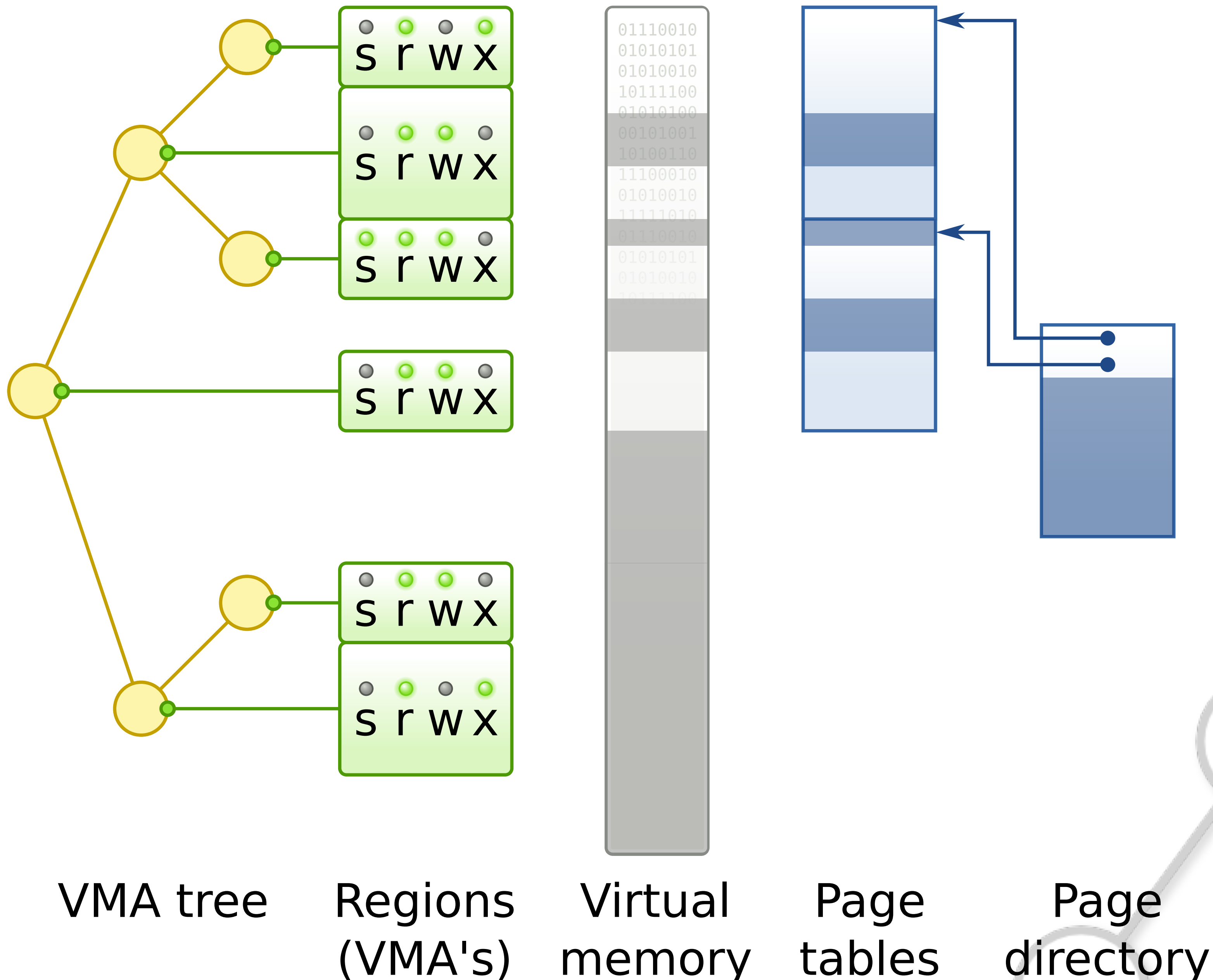


Address spaces

mmap
Creates VMAs

page fault
Create and fill page table on demand

munmap

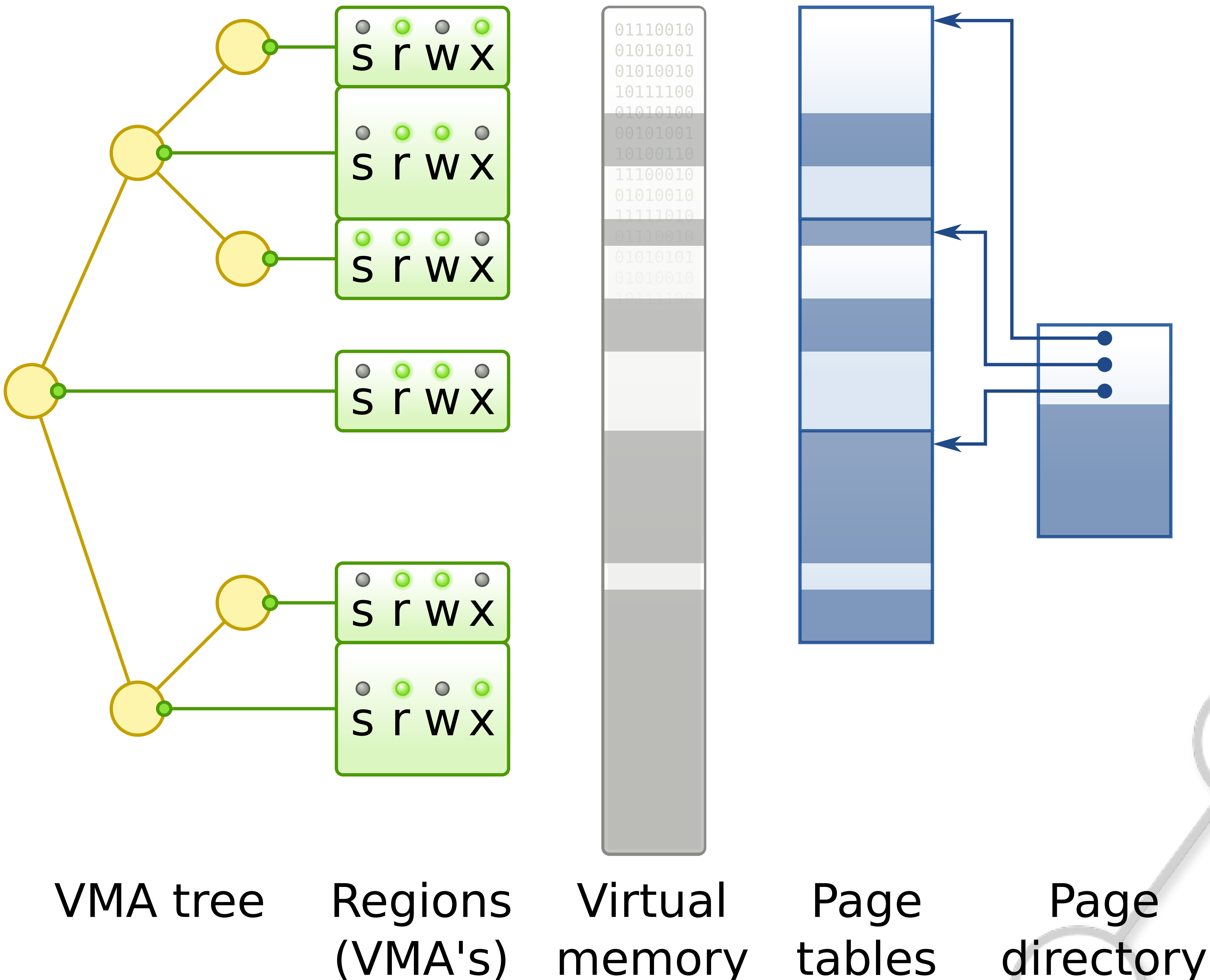


Address spaces

mmap
Creates VMAs

page fault
Create and fill page table on demand

munmap

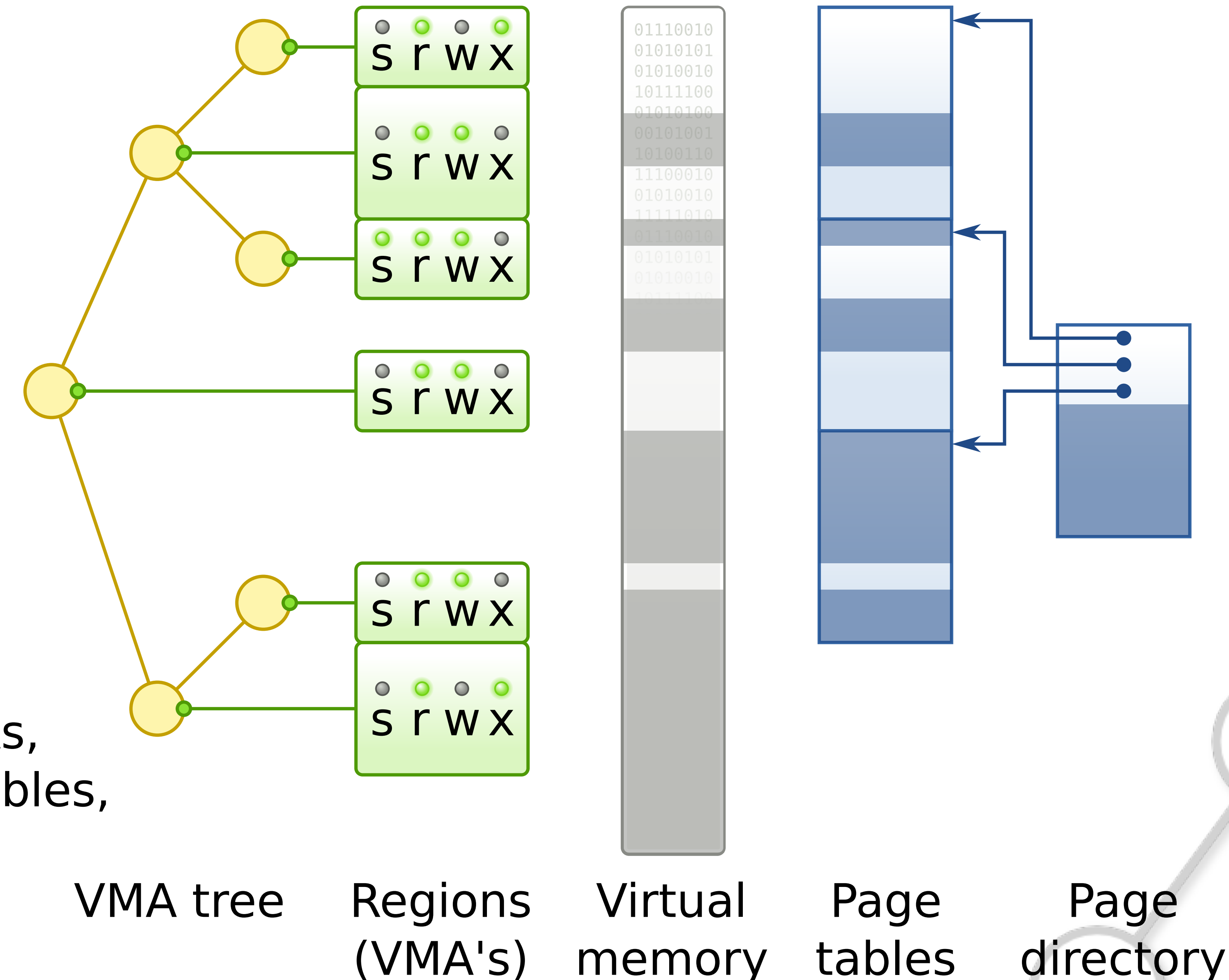


Address spaces

mmap
Creates VMAs

page fault
Create and fill page table on demand

munmap
Removes VMAs, clears page tables, frees pages

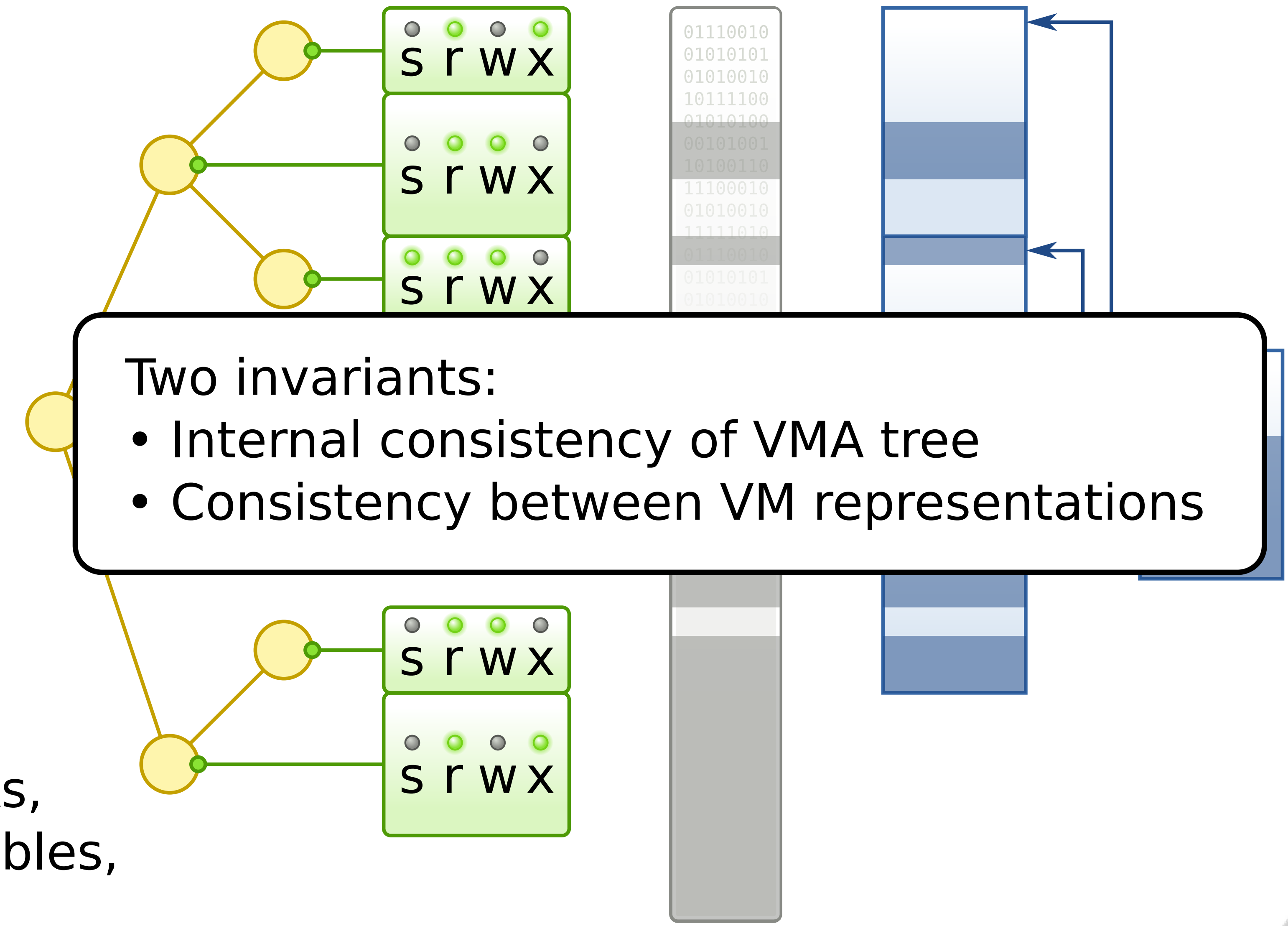


Address spaces

mmap
Creates VMAs

page fault
Create and fill page table on demand

munmap
Removes VMAs, clears page tables, frees pages



Two invariants:

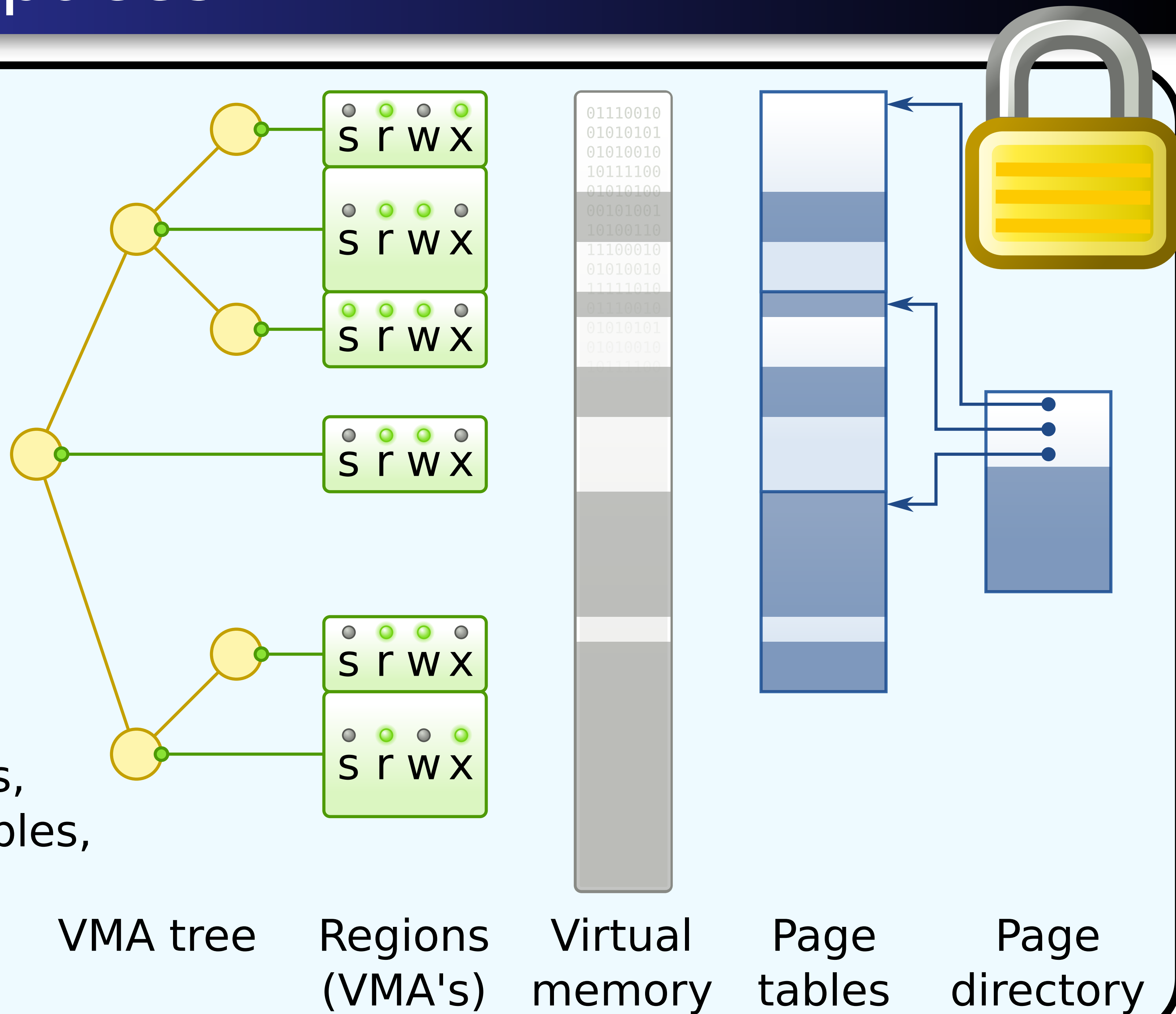
- Internal consistency of VMA tree
- Consistency between VM representations

Address spaces

mmap
Creates VMAs

page fault
Create and fill page table on demand

munmap
Removes VMAs, clears page tables, frees pages

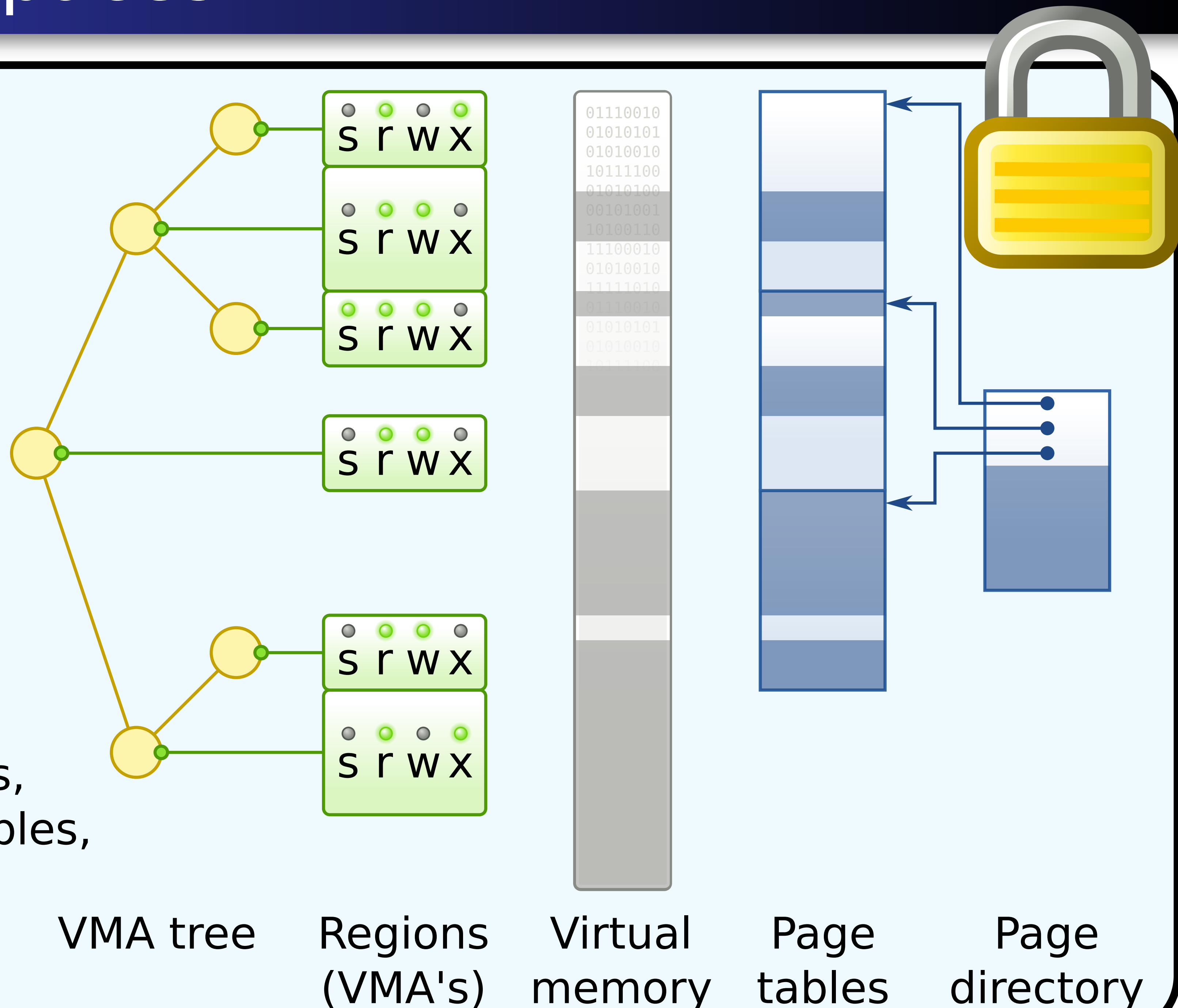


Address spaces

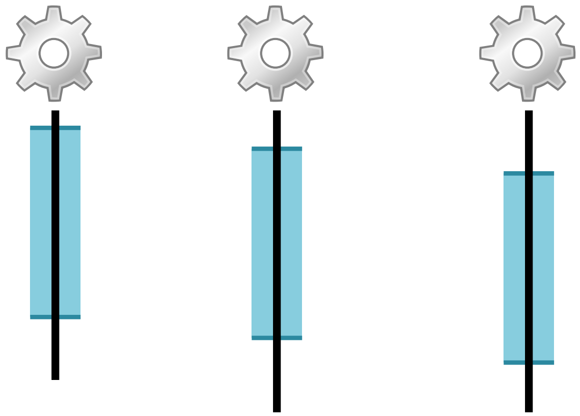
mmap W
Creates VMAs

page fault R
Create and fill page table on demand

munmap W
Removes VMAs, clears page tables, frees pages

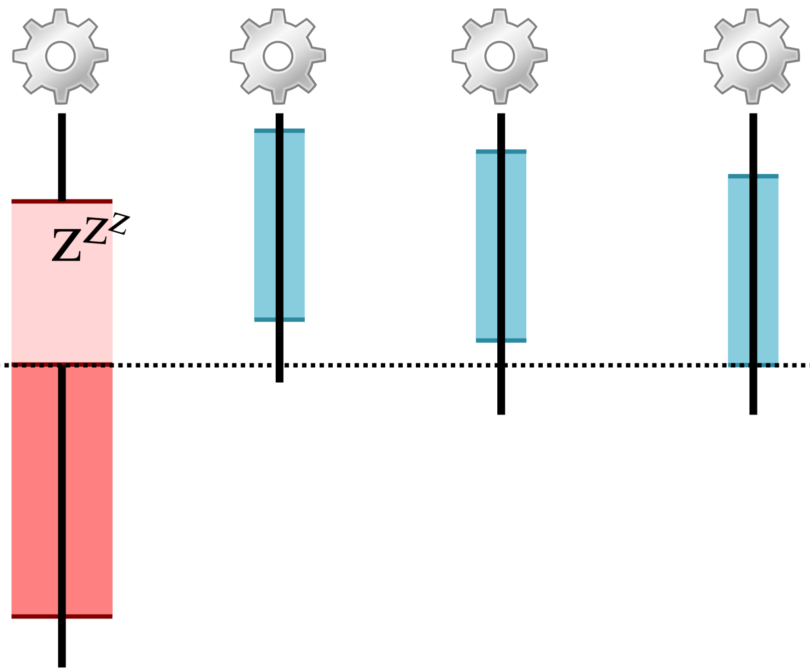


What's so bad about RW locks?



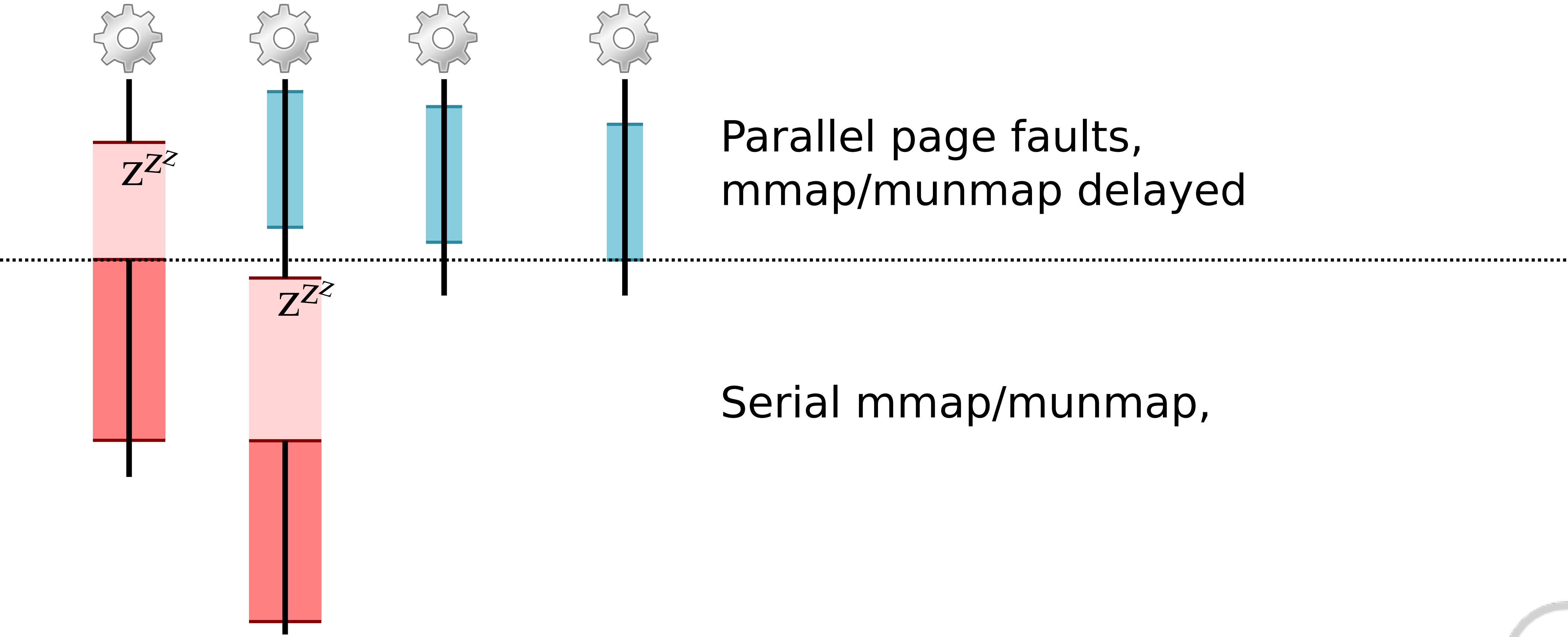
Parallel page faults,

What's so bad about RW locks?



Parallel page faults,
mmap/munmap delayed

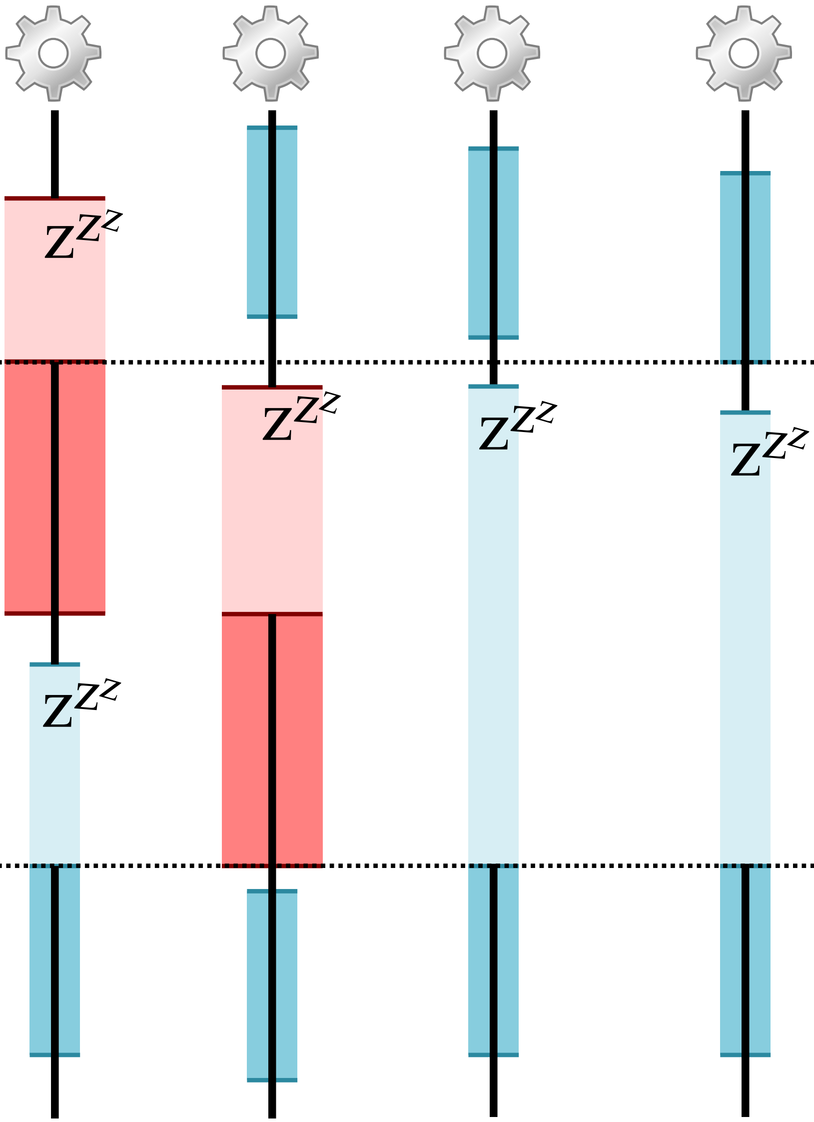
What's so bad about RW locks?



Parallel page faults,
mmap/munmap delayed

Serial mmap/munmap,

What's so bad about RW locks?

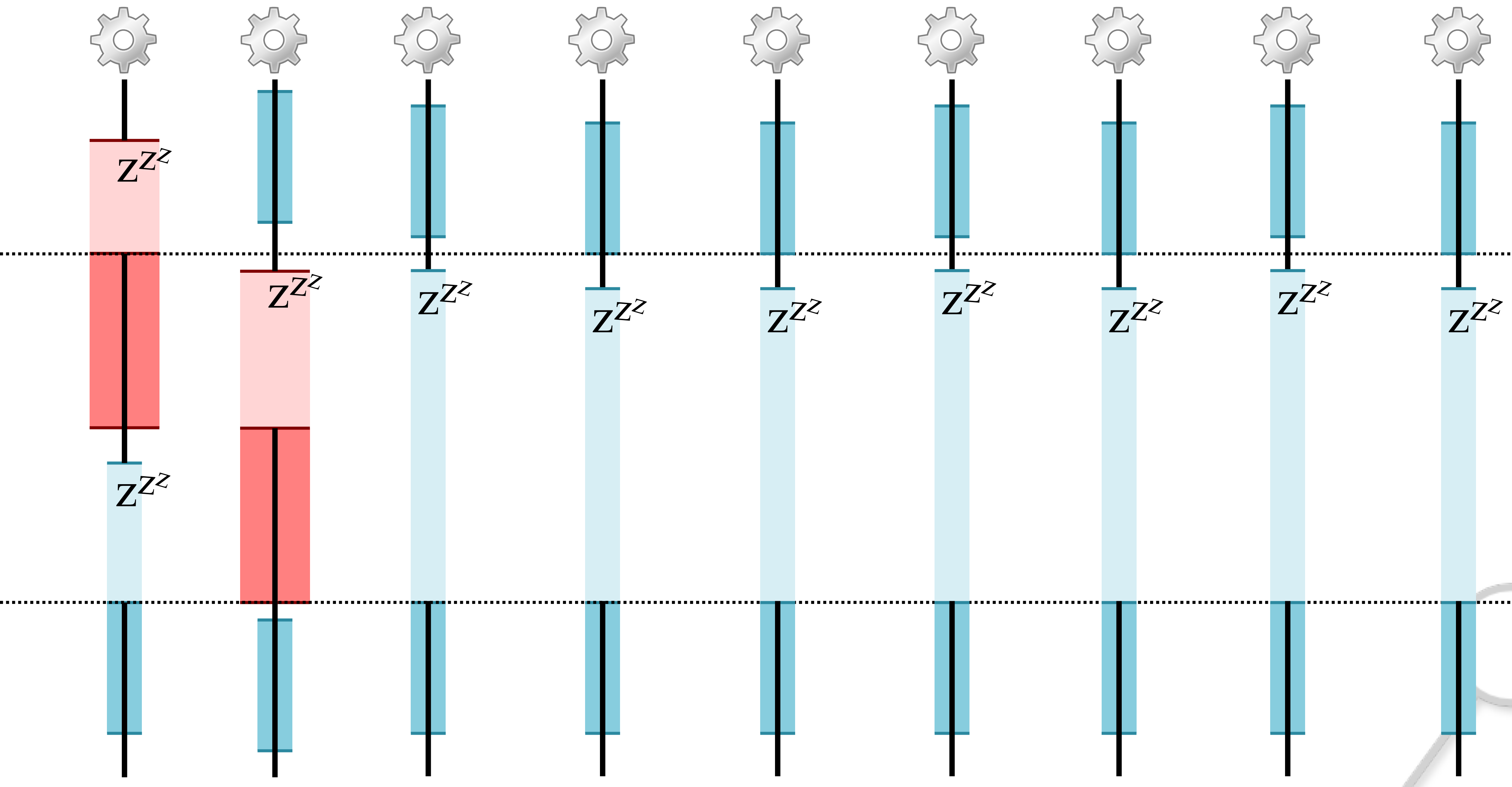


Parallel page faults, mmap/munmap delayed

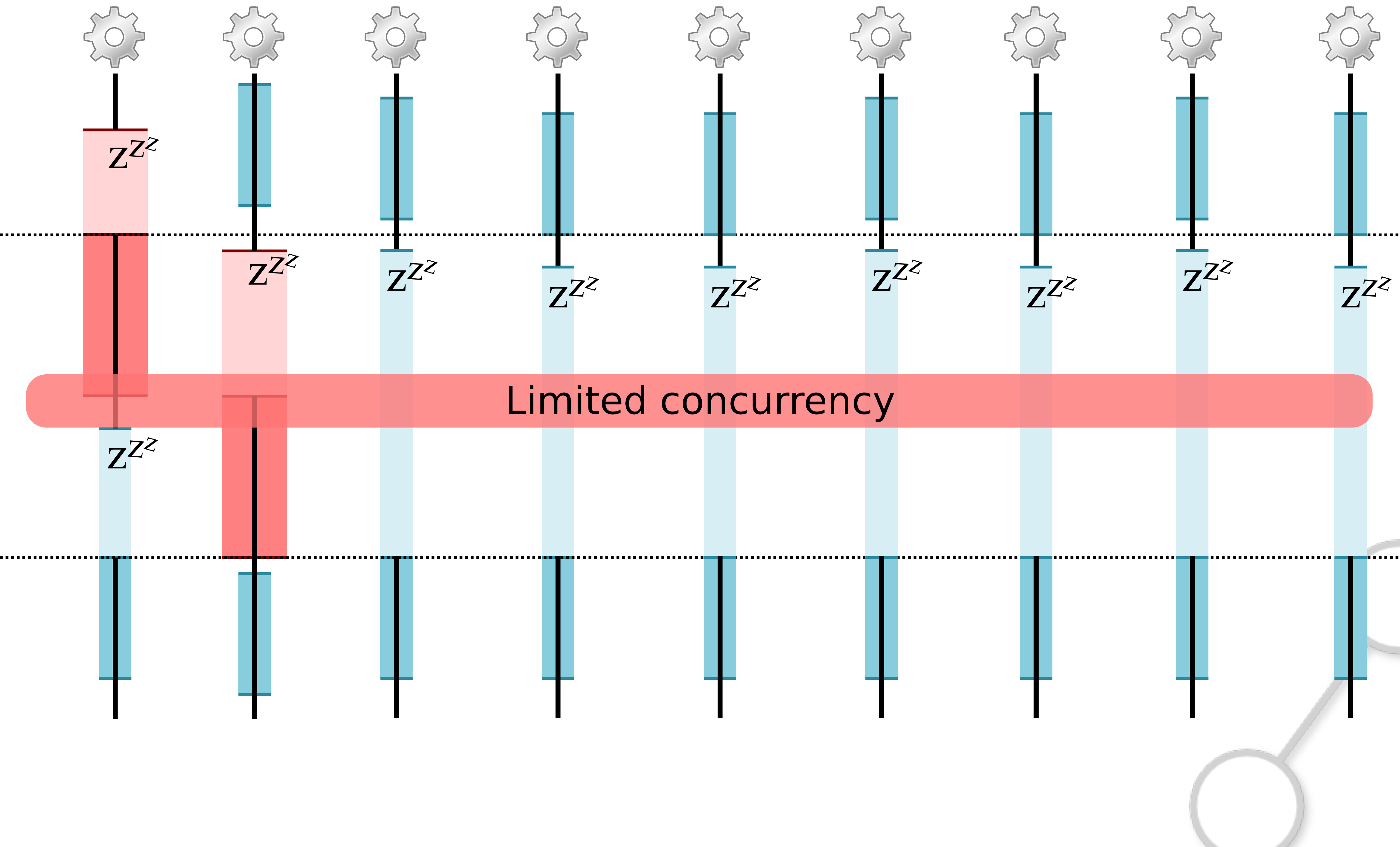
Serial mmap/munmap, page faults delayed

Parallel page faults

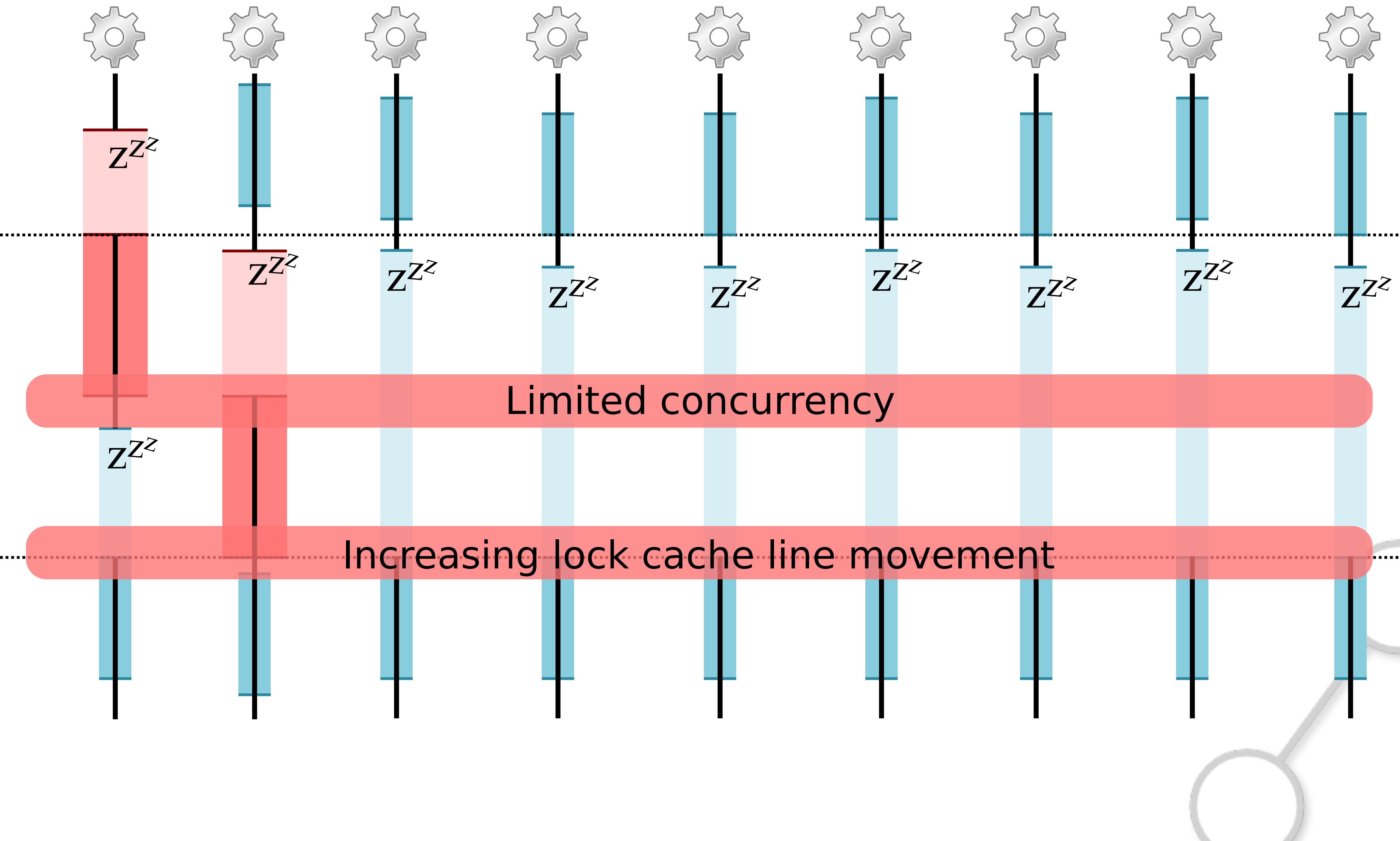
What's so bad about RW locks?



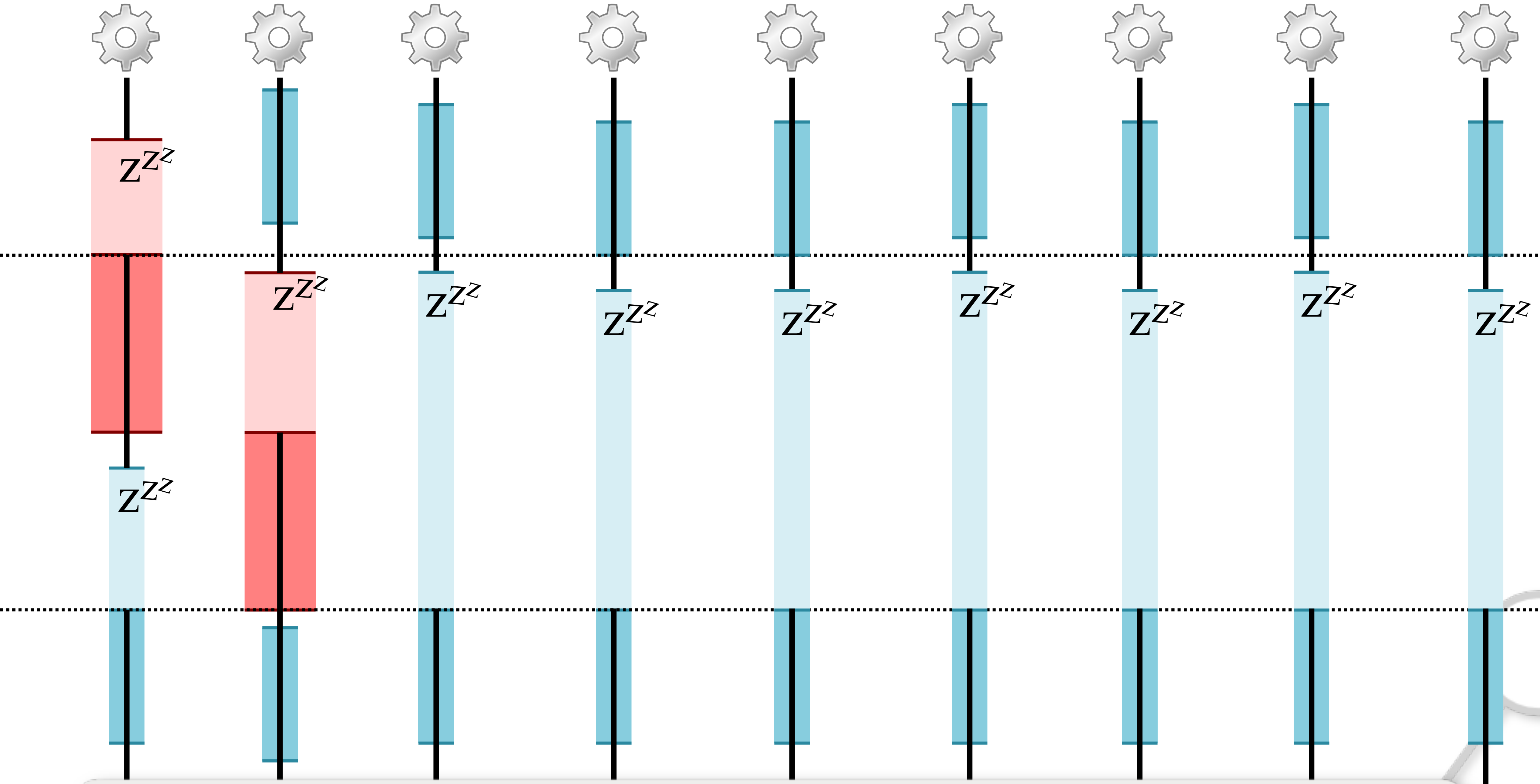
What's so bad about RW locks?



What's so bad about RW locks?

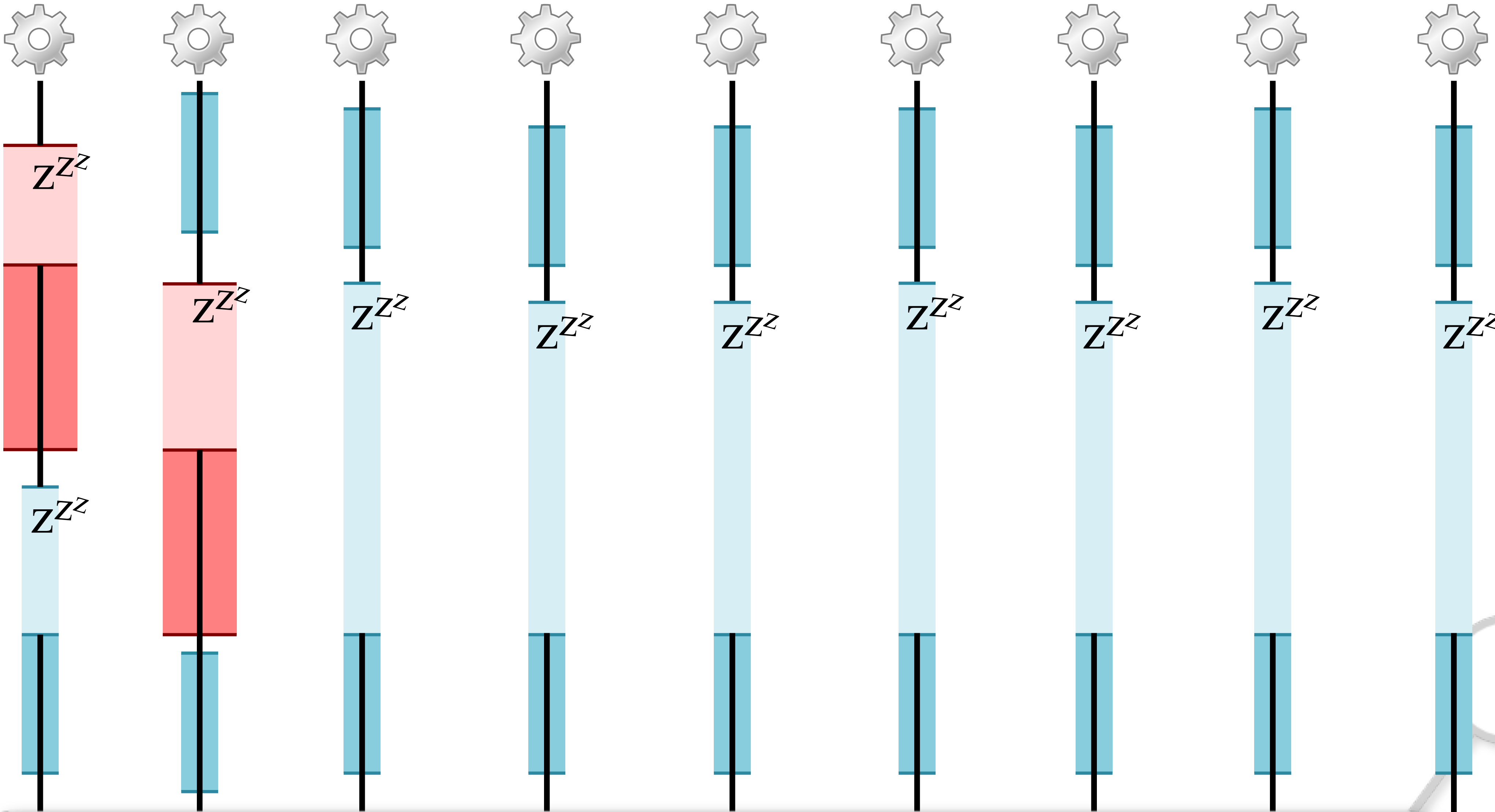


What's so bad about RW locks?



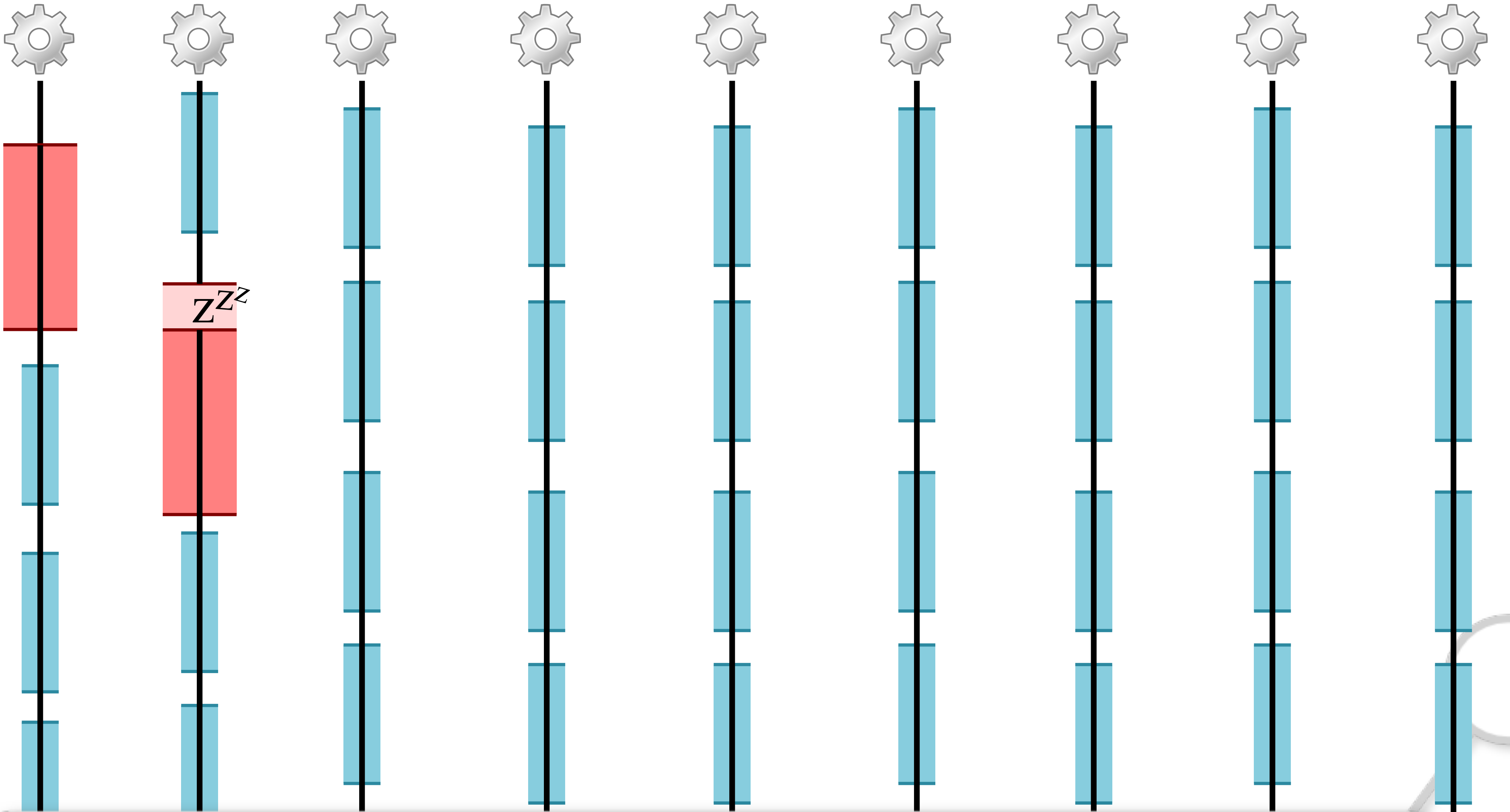
Must minimize cache line movement.
Page faults must run concurrently with mapping operations.

What's so bad about RW locks?



Must minimize cache line movement.
Page faults must run concurrently with mapping operations.

What's so bad about RW locks?



Must minimize cache line movement.
Page faults must run concurrently with mapping operations.

Contributions

An RCU-based scheme for highly scalable page fault handling

An RCU-compatible balanced binary tree (Bonsai)

Implementation and evaluation in Linux 2.6.37

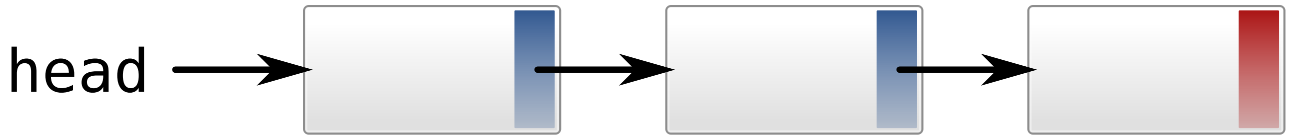
A Read-Copy-Update primer

A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free

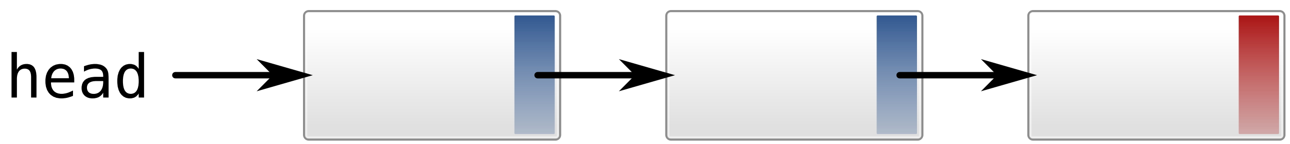
A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free



A Read-Copy-Update primer

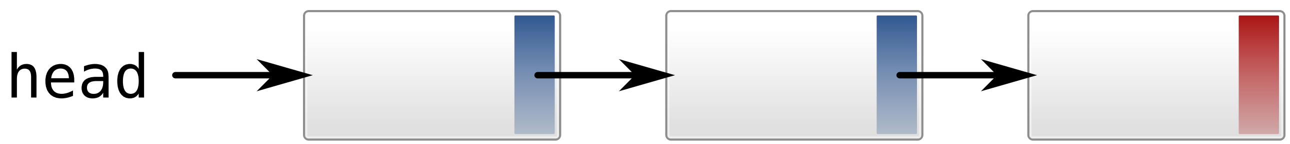
Lock-free reads + Single pointer update + Delayed free




```
length()
  p = head;
  for (i=0; p; p=p->next, i++);
  return i;
```

A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free

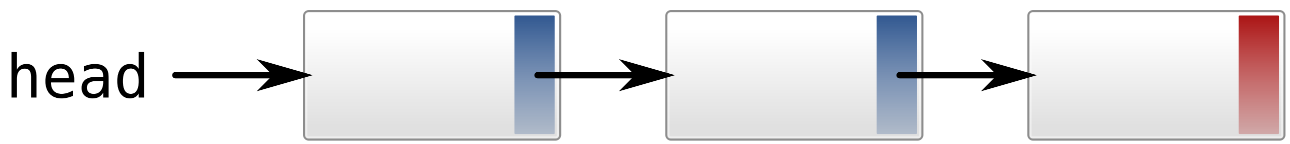



```
length()
  p = head;
  for (i=0; p; p=p->next, i++);
  return i;
```

No locks, no barriers

A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free



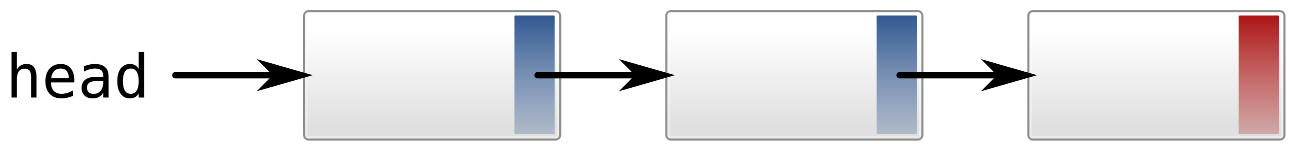

```
length()
  p = head;
  for (i=0; p; p=p->next, i++);
  return i;
```

No locks, no barriers

```
pop_n(n)
  lock();
  for (p=head; p&&n; p=p->next, n--)
    rcu_free(p);
  head = p;
  unlock();
```

A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free




```
length()
  p = head;
  for (i=0; p; p=p->next, i++);
  return i;
```

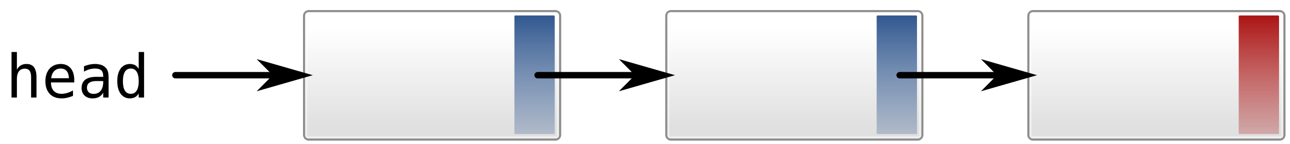
No locks, no barriers

```
pop_n(n)
  lock();
  for (p=head; p&&n; p=p->next, n--)
    rcu_free(p);
  head = p;
  unlock();
```

Updates exactly one pointer

A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free




```
length()
  p = head;
  for (i=0; p; p=p->next, i++);
  return i;
```

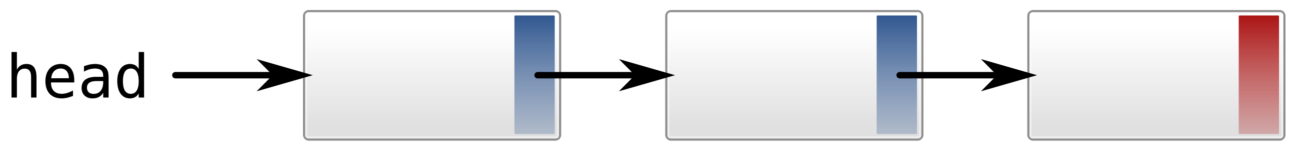
No locks, no barriers

```
pop_n(n)
  lock();
  for (p=head; p&&n; p=p->next, n--)
    rcu_free(p);
  head = p;
  unlock();
```

Updates exactly one pointer

A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free




```
length()
  p = head;
  for (i=0; p; p=p->next, i++);
  return i;
```

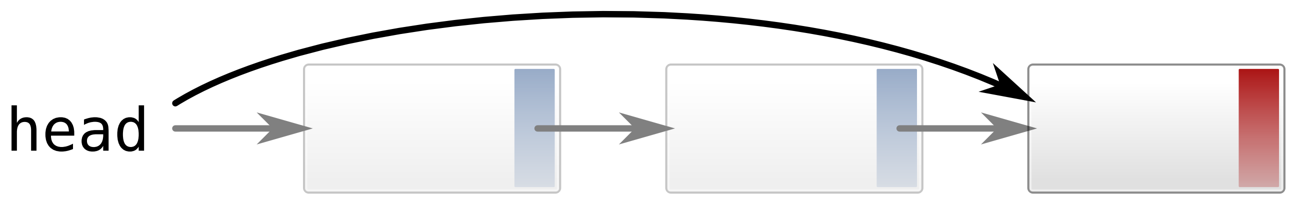
No locks, no barriers

```
pop_n(n)
  lock();
  for (p=head; p&&n; p=p->next, n--)
    rcu_free(p);
  head = p;
  unlock();
```

Updates exactly one pointer

A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free




```
length()
  p = head;
  for (i=0; p; p=p->next, i++);
  return i;
```

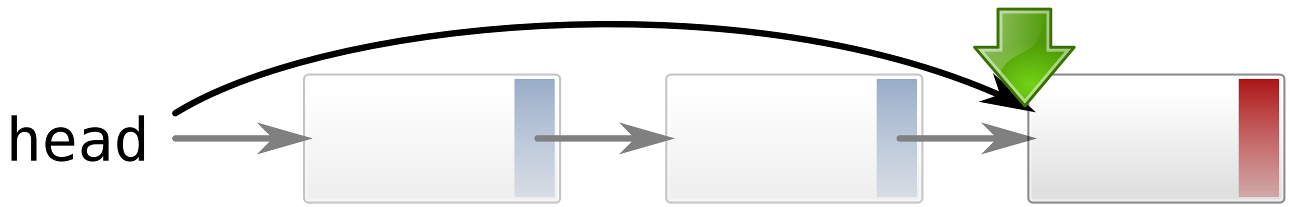
No locks, no barriers

```
pop_n(n)
  lock();
  for (p=head; p&&n; p=p->next, n--)
    rcu_free(p);
  head = p;
  unlock();
```

Updates exactly one pointer

A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free




```
length()
  p = head;
  for (i=0; p; p=p->next, i++);
  return i;
```

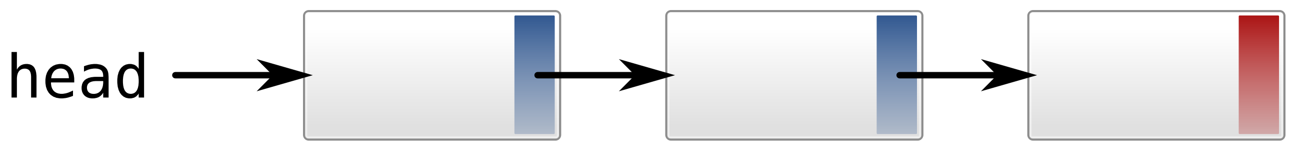
No locks, no barriers

```
pop_n(n)
  lock();
  for (p=head; p&&n; p=p->next, n--)
    rcu_free(p);
  head = p;
  unlock();
```

Updates exactly one pointer

A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free




```
length()
  p = head;
  for (i=0; p; p=p->next, i++);
  return i;
```

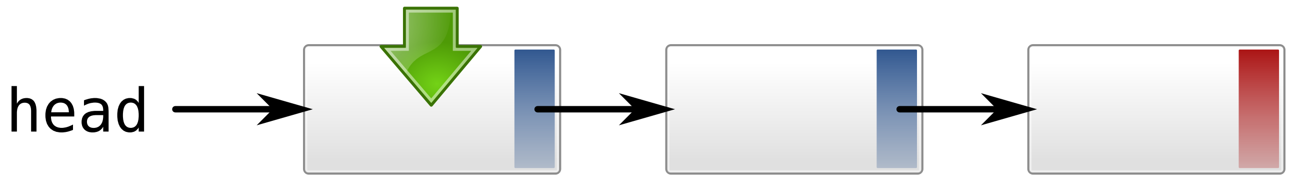
No locks, no barriers

```
pop_n(n)
  lock();
  for (p=head; p&&n; p=p->next, n--)
    rcu_free(p);
  head = p;
  unlock();
```

Updates exactly one pointer

A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free




```
length()
  p = head;
  for (i=0; p; p=p->next, i++);
  return i;
```

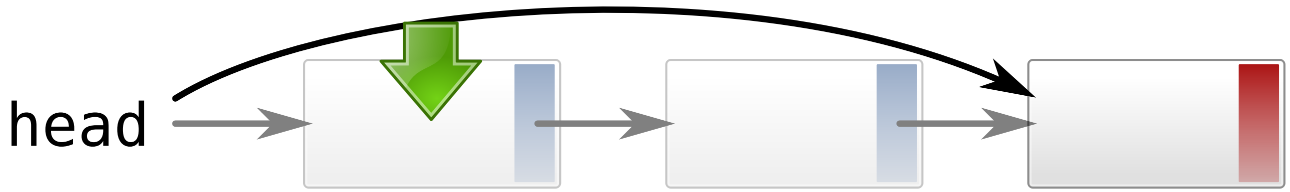
No locks, no barriers

```
pop_n(n)
  lock();
  for (p=head; p&&n; p=p->next, n--)
    rcu_free(p);
  head = p;
  unlock();
```

Updates exactly one pointer

A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free




```
length()
  p = head;
  for (i=0; p; p=p->next, i++);
  return i;
```

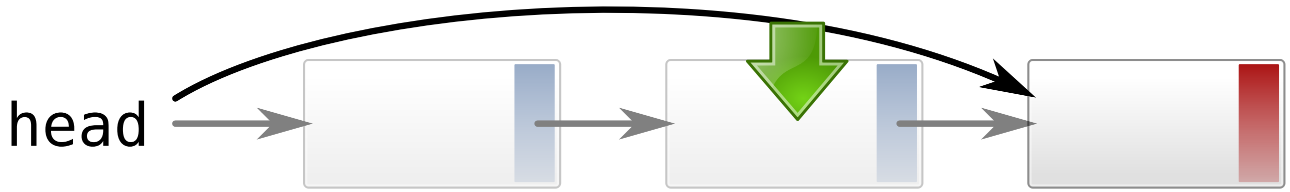
No locks, no barriers

```
pop_n(n)
  lock();
  for (p=head; p&&n; p=p->next, n--)
    rcu_free(p);
  head = p;
  unlock();
```

Updates exactly one pointer

A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free




```
length()
  p = head;
  for (i=0; p; p=p->next, i++);
  return i;
```

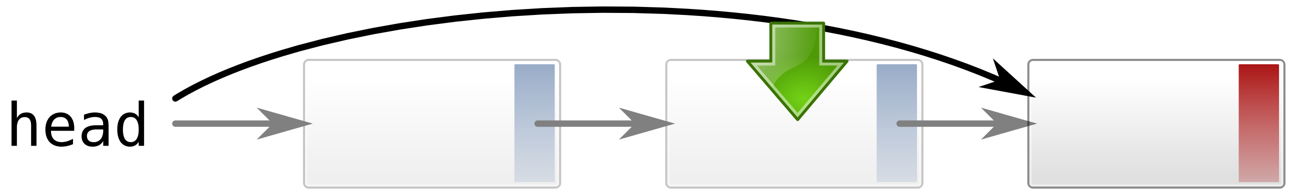
No locks, no barriers

```
pop_n(n)
  lock();
  for (p=head; p&&n; p=p->next, n--)
    rcu_free(p);
  head = p;
  unlock();
```

Updates exactly one pointer

A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free




```
length()
  p = head;
  for (i=0; p; p=p->next, i++);
  return i;
```

No locks, no barriers

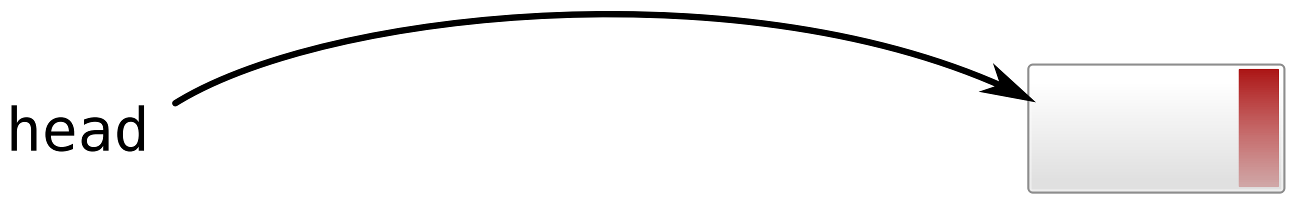
```
pop_n(n)
  lock();
  for (p=head; p&&n; p=p->next, n--)
    rcu_free(p);
  head = p;
  unlock();
```

Updates exactly one pointer

Free is delayed until all readers return
(e.g., by waiting for all CPU's to schedule)

A Read-Copy-Update primer

Lock-free reads + Single pointer update + Delayed free




```
length()
  p = head;
  for (i=0; p; p=p->next, i++);
  return i;
```

No locks, no barriers

```
pop_n(n)
  lock();
  for (p=head; p&&n; p=p->next, n--)
    rcu_free(p);
  head = p;
  unlock();
```

Updates exactly one pointer

Free is delayed until all readers return
(e.g., by waiting for all CPU's to schedule)

RCU challenges

RCU is already used pervasively in Linux

RCU challenges

RCU is already used pervasively in Linux
... except in the VM system

RCU challenges

RCU is already used pervasively in Linux
... except in the VM system

Address space operations update many pointers

The page fault handler isn't read-only

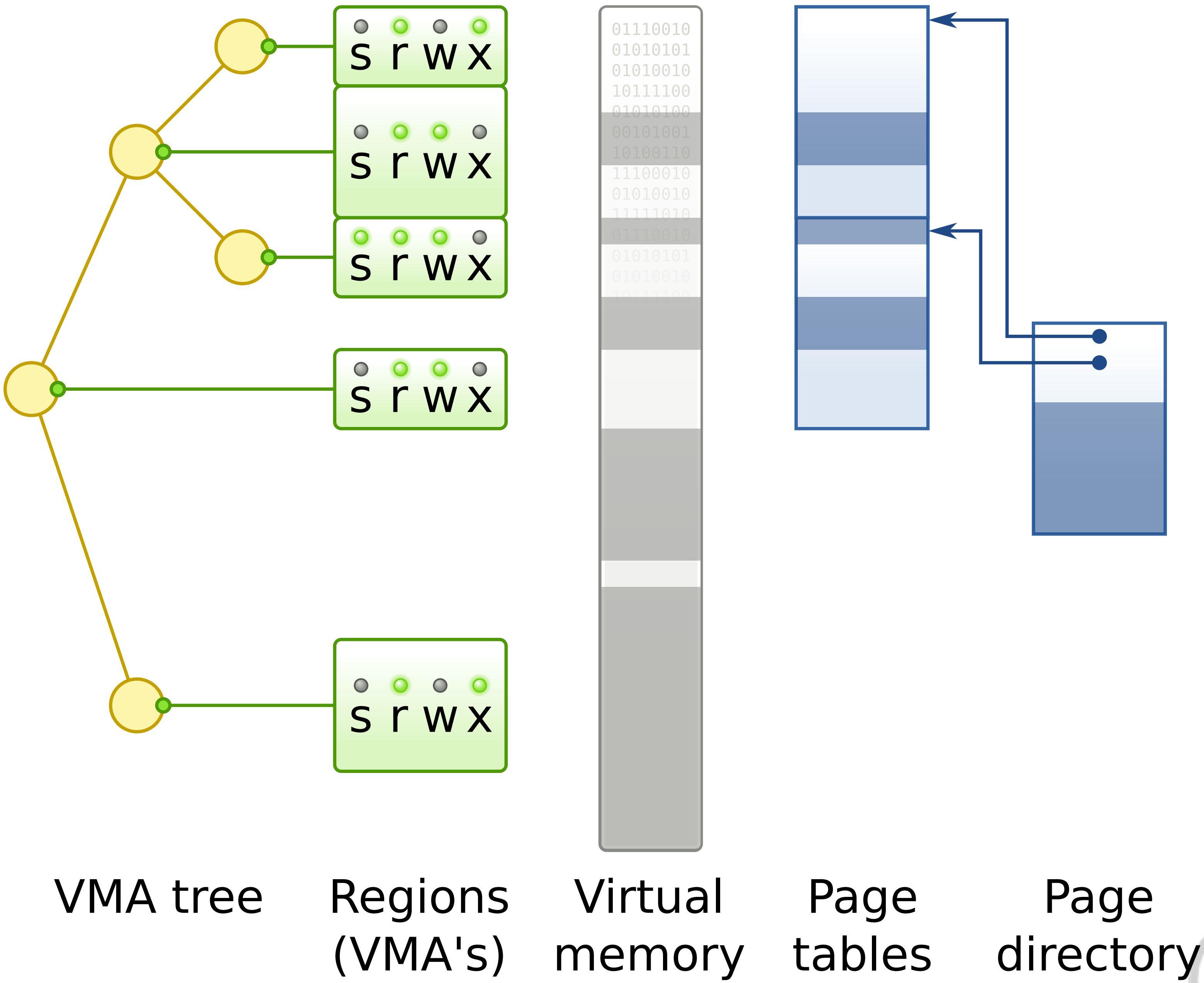
General approach

Remove address space lock from page fault fast path

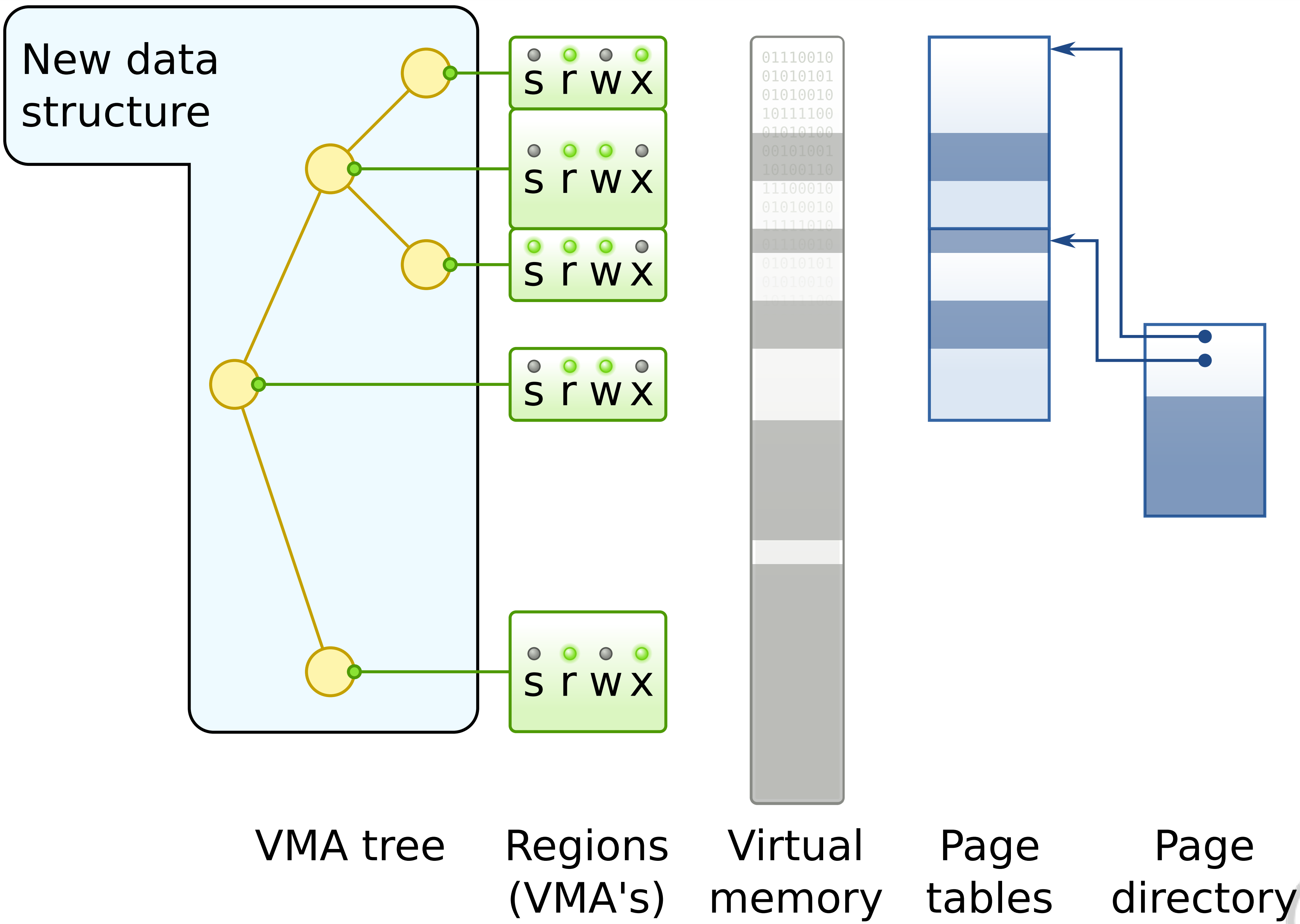
Reduce writes required to map/unmap memory

Detect remaining races and retry racing page faults

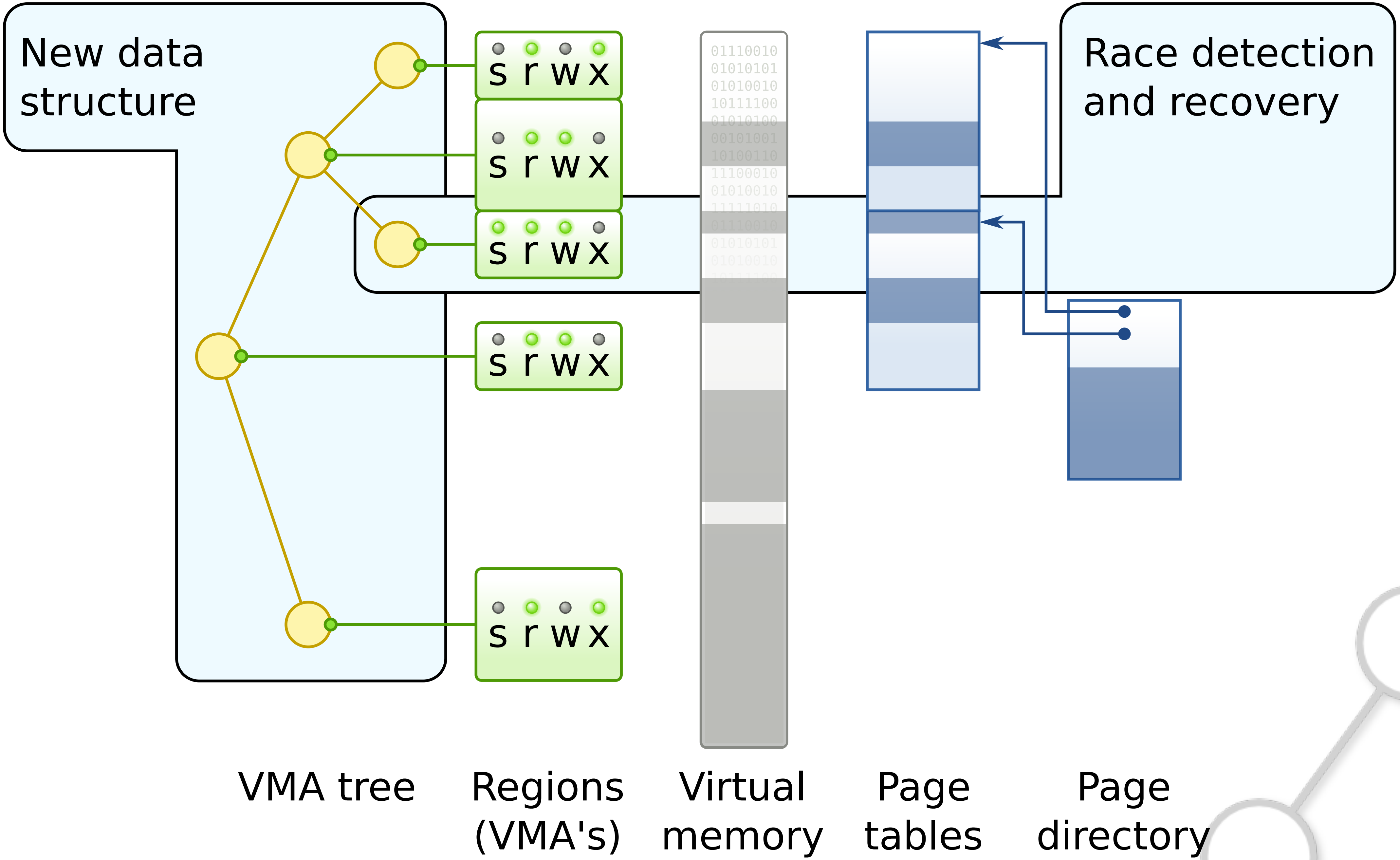
General approach



General approach



General approach

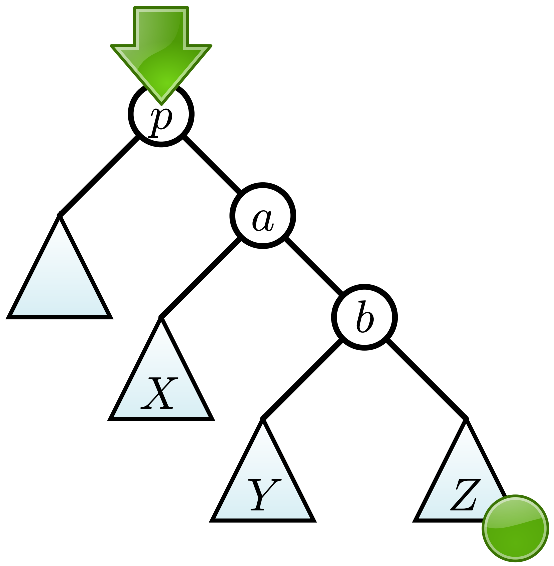


The VMA tree problem

Tree balancing is not RCU-friendly

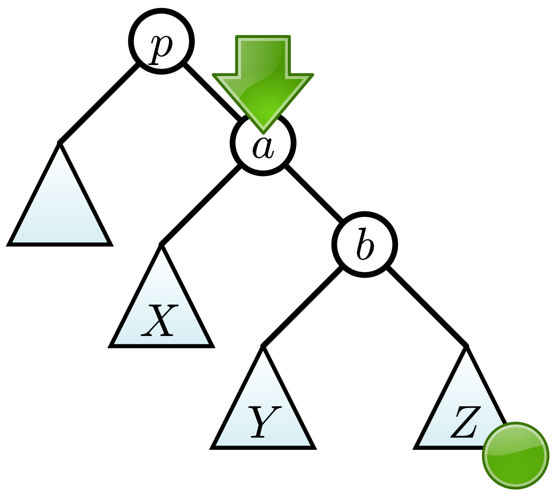
The VMA tree problem

Tree balancing is not RCU-friendly



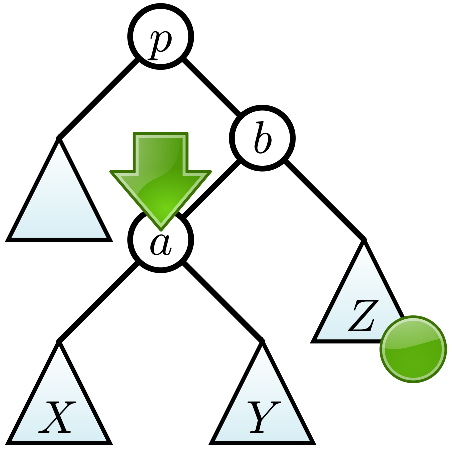
The VMA tree problem

Tree balancing is not RCU-friendly



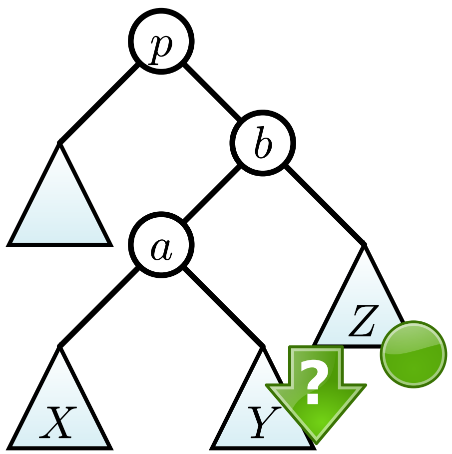
The VMA tree problem

Tree balancing is not RCU-friendly



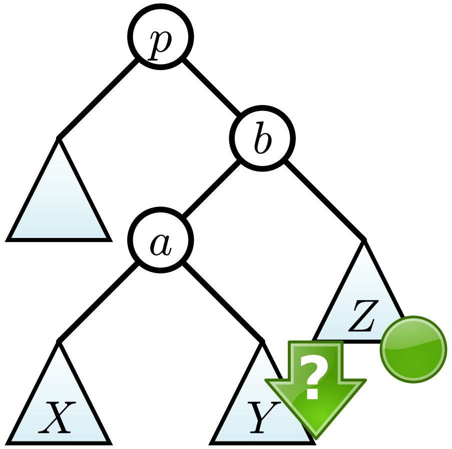
The VMA tree problem

Tree balancing is not RCU-friendly



The VMA tree problem

Tree balancing is not RCU-friendly



Balancing performs multiple pointer updates, so lock-free readers can see partial updates [Fraser '03, Fraser '07, Howard '10, etc]

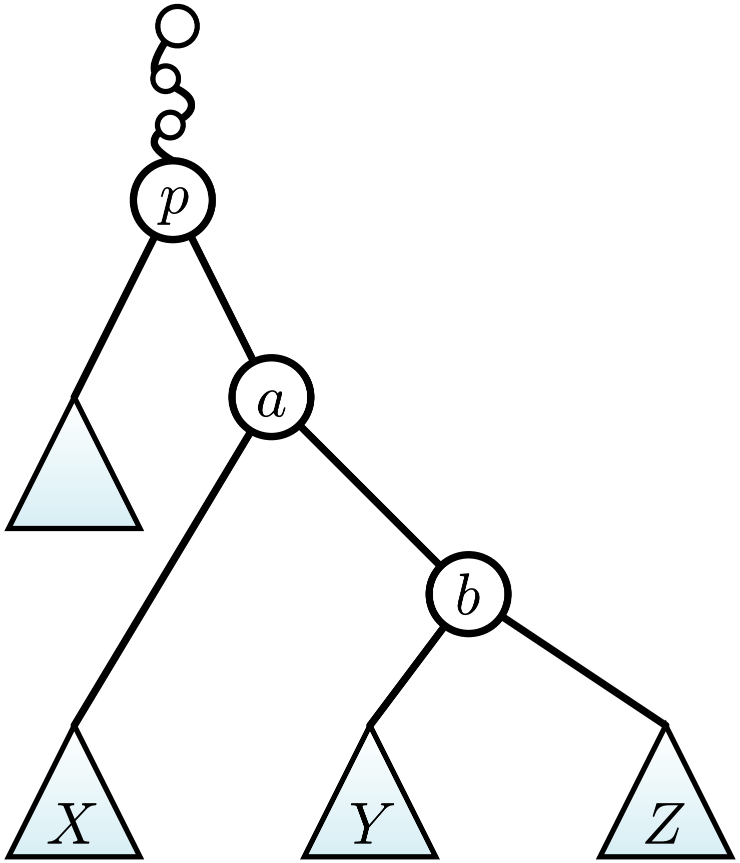
The basic Bonsai tree

The basic Bonsai tree

Functional data structures don't update any pointers!
"Modification" creates a new structure

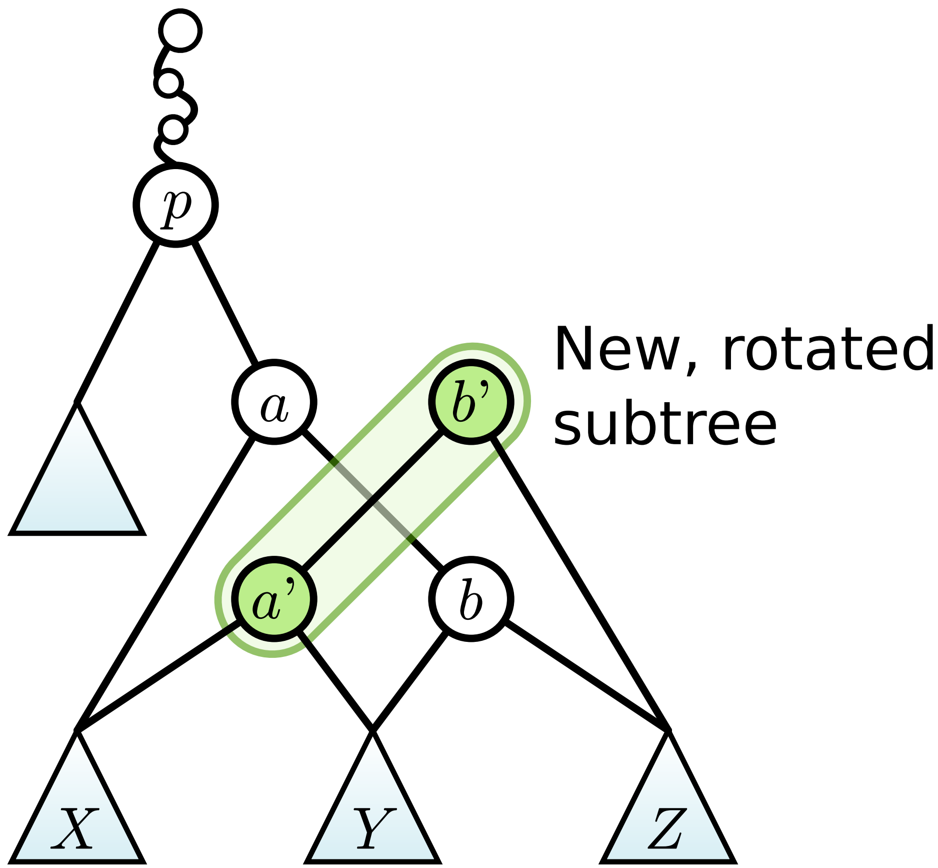
The basic Bonsai tree

Functional data structures don't update any pointers!
"Modification" creates a new structure



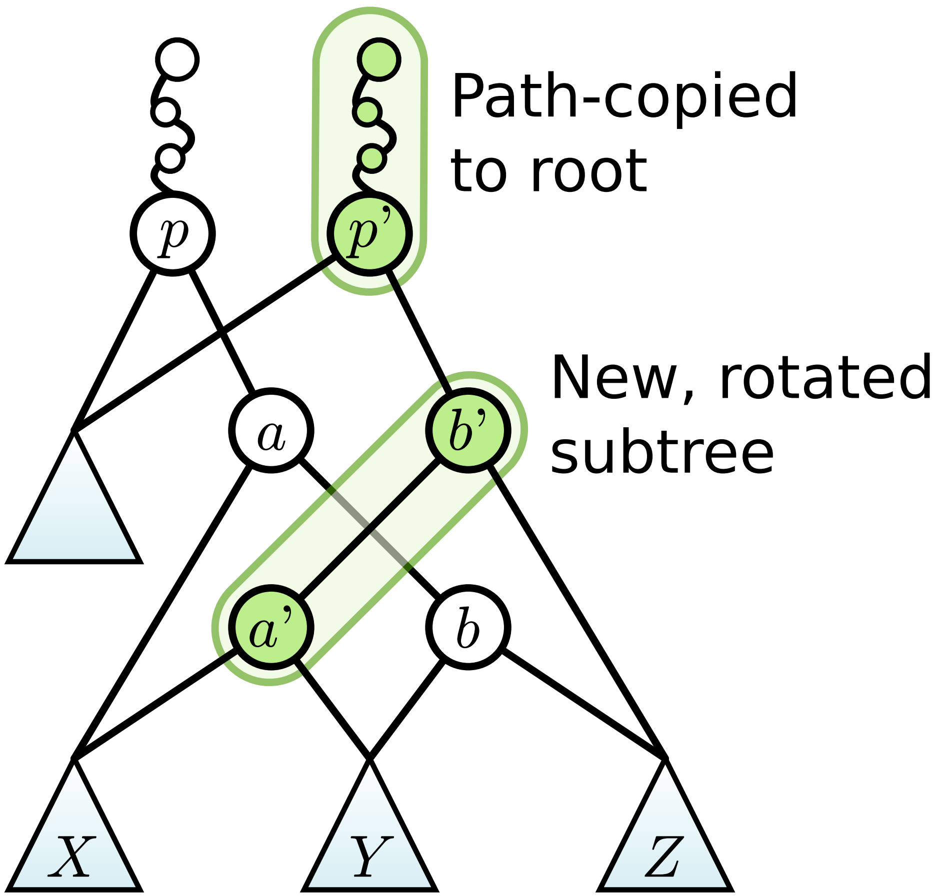
The basic Bonsai tree

Functional data structures don't update any pointers!
"Modification" creates a new structure



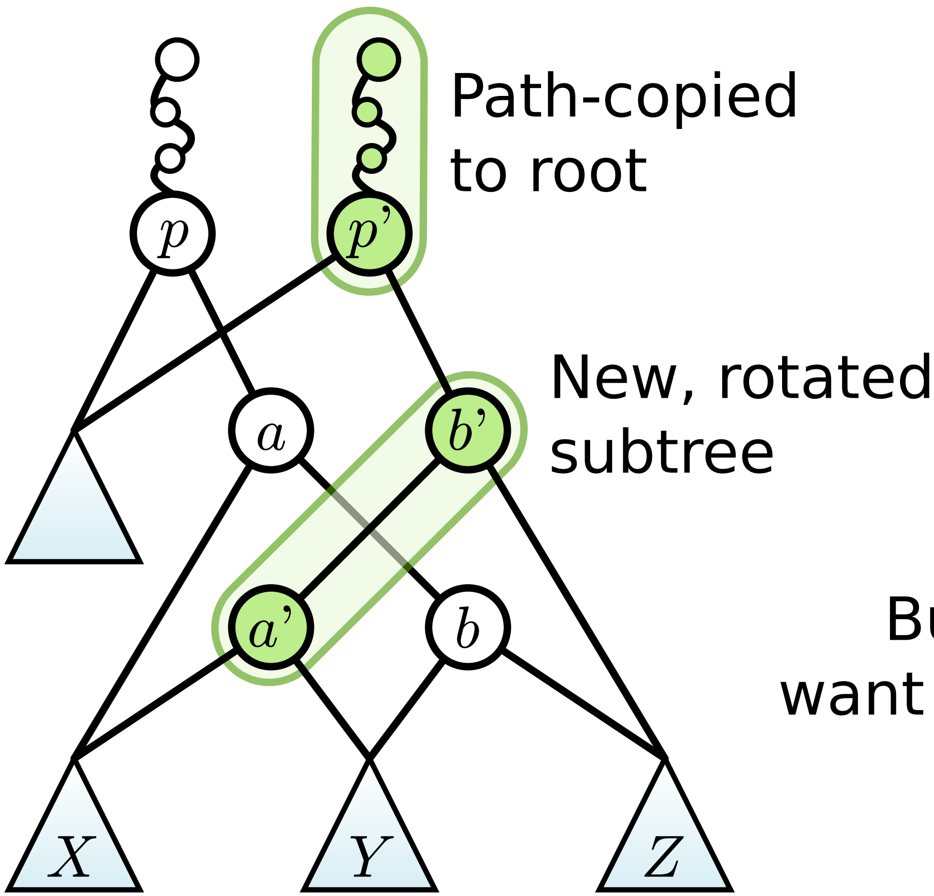
The basic Bonsai tree

Functional data structures don't update any pointers!
"Modification" creates a new structure



The basic Bonsai tree

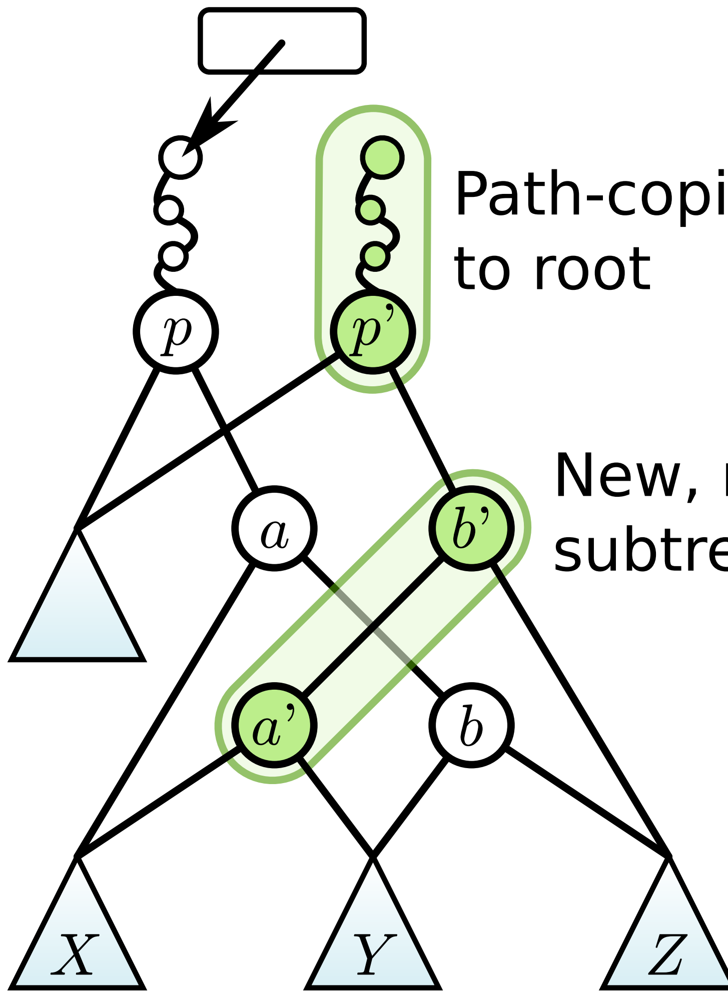
Functional data structures don't update any pointers!
"Modification" creates a new structure



But we don't
want a *new* tree

The basic Bonsai tree

Functional data structures don't update any pointers!
"Modification" creates a new structure



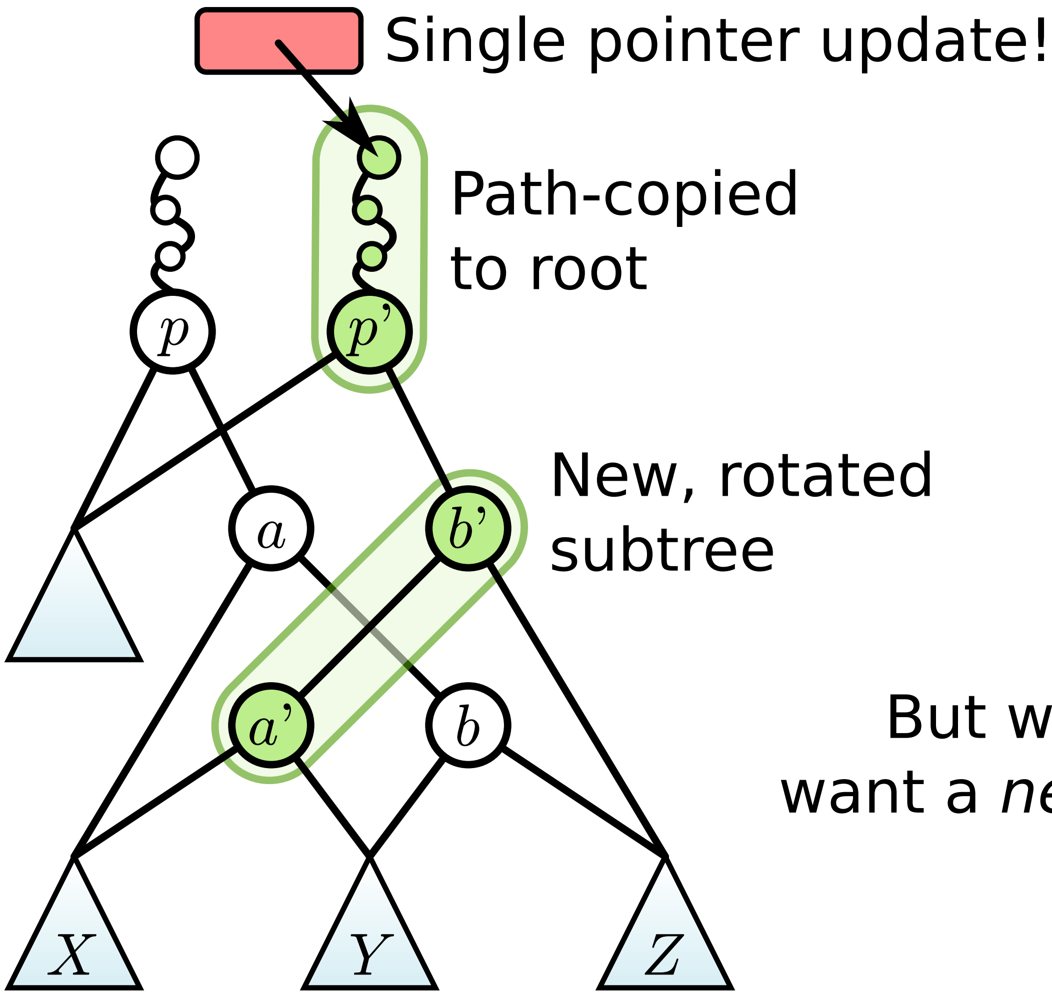
Path-copied to root

New, rotated subtree

But we don't want a *new* tree

The basic Bonsai tree

Functional data structures don't update any pointers!
"Modification" creates a new structure

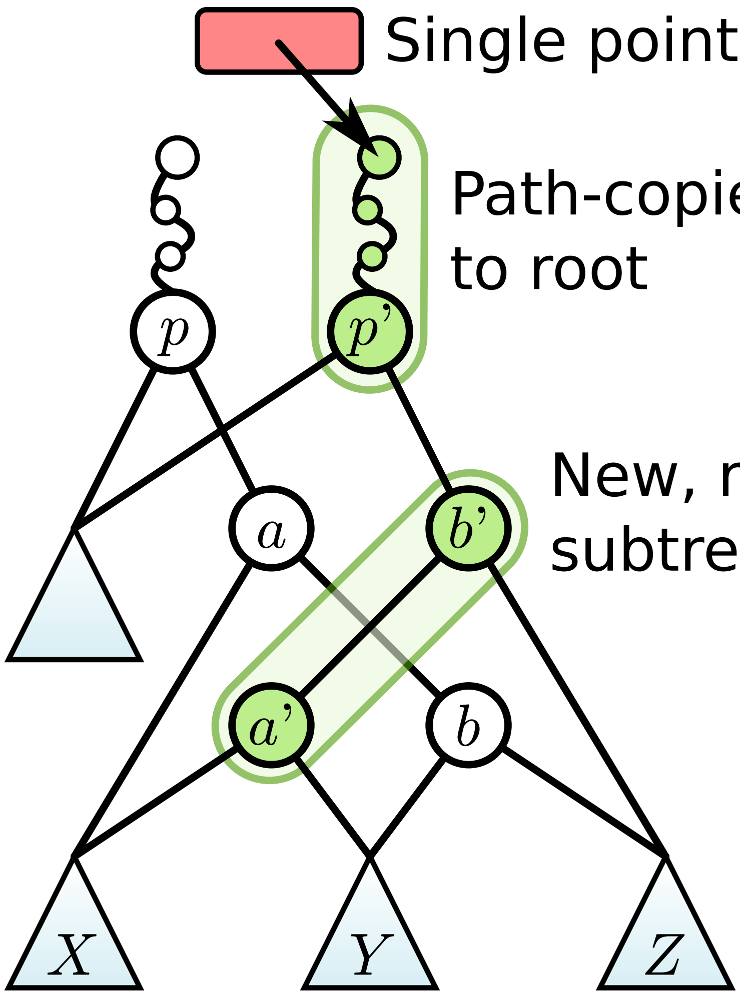


But we don't want a *new* tree

The basic Bonsai tree

Functional data structures don't update any pointers!
"Modification" creates a new structure

Single pointer update!



Path-copied to root

New, rotated subtree

We don't need the old tree

But we don't want a *new* tree

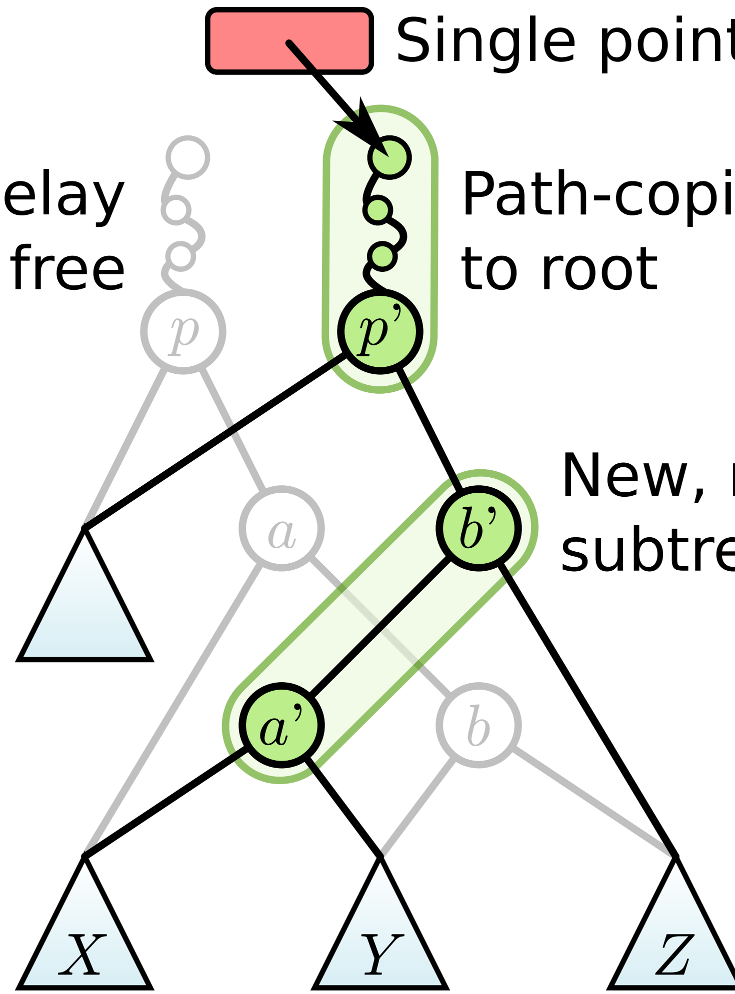
The basic Bonsai tree

Functional data structures don't update any pointers!
"Modification" creates a new structure

Single pointer update!

RCU delay free

Path-copied to root



New, rotated subtree

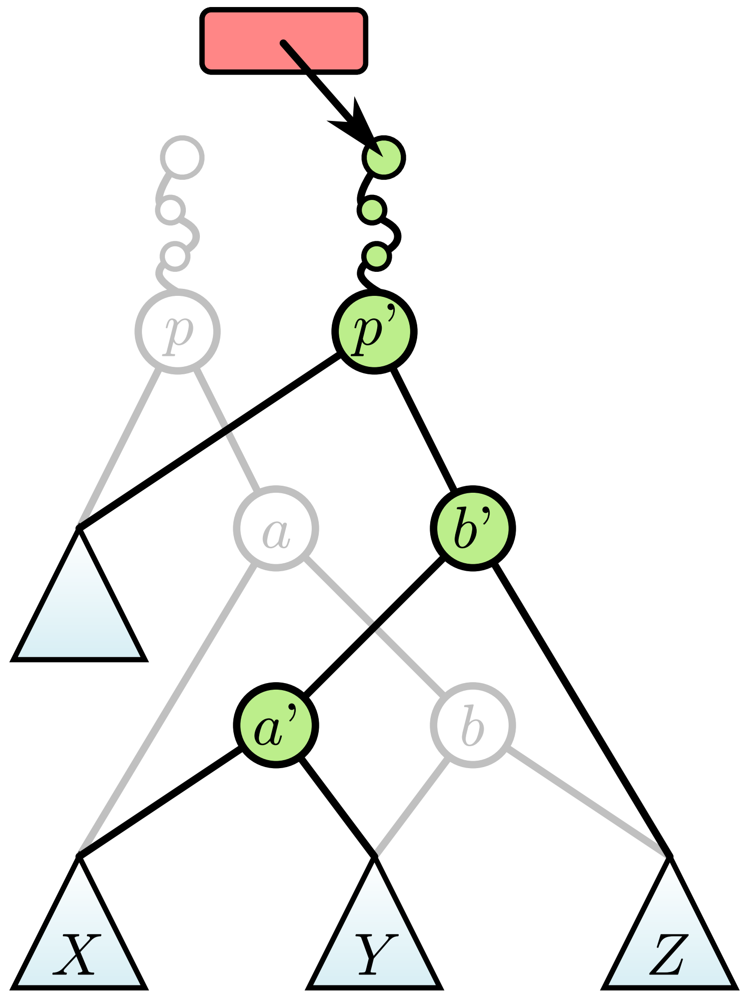
We don't need the old tree

But we don't want a *new* tree

The better Bonsai tree

We can do better

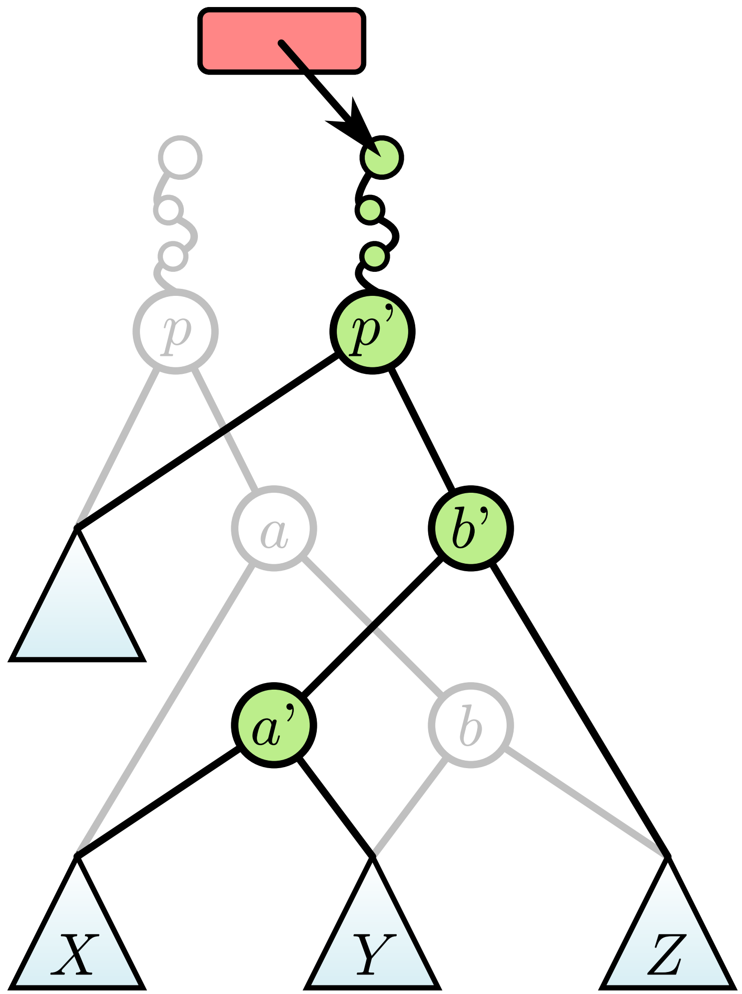
O n O n



The better Bonsai tree

We can do better

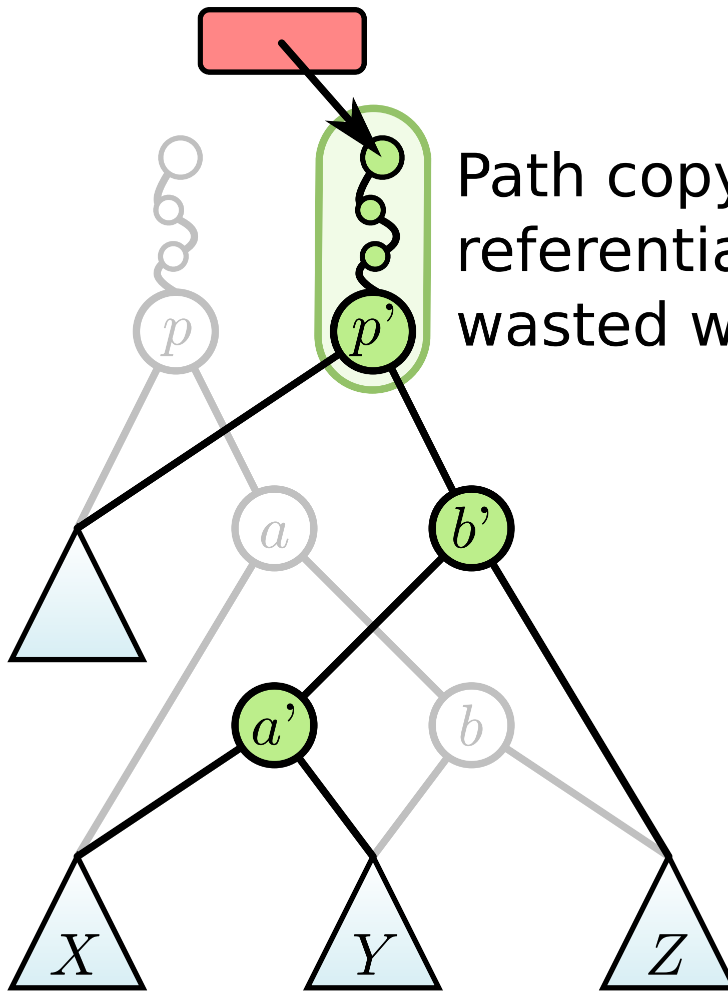
Insertion allocates $O(\log n)$ nodes, frees $O(\log n)$ nodes



The better Bonsai tree

We can do better

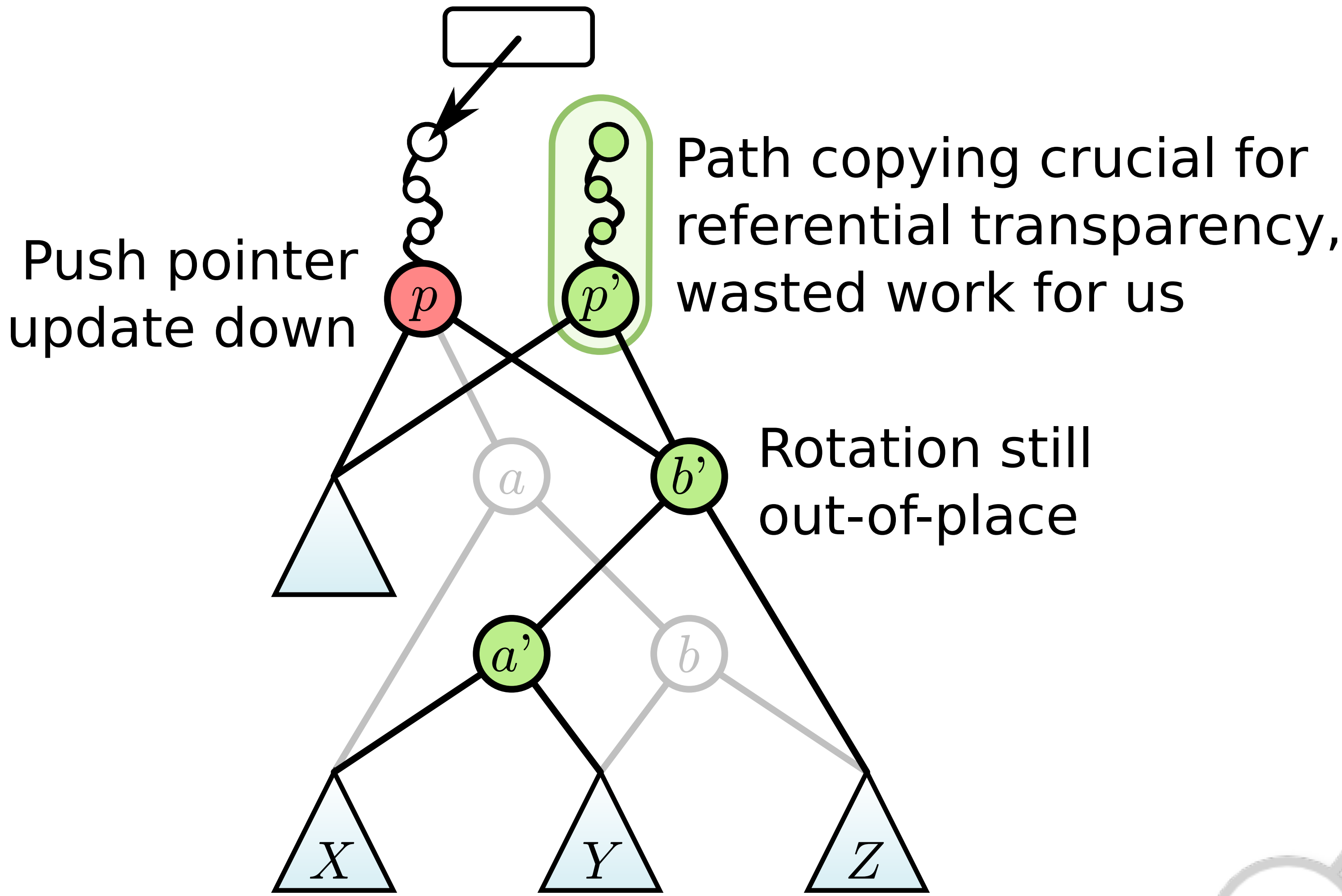
Insertion allocates $O(\log n)$ nodes, frees $O(\log n)$ nodes



Path copying crucial for referential transparency, wasted work for us

The better Bonsai tree

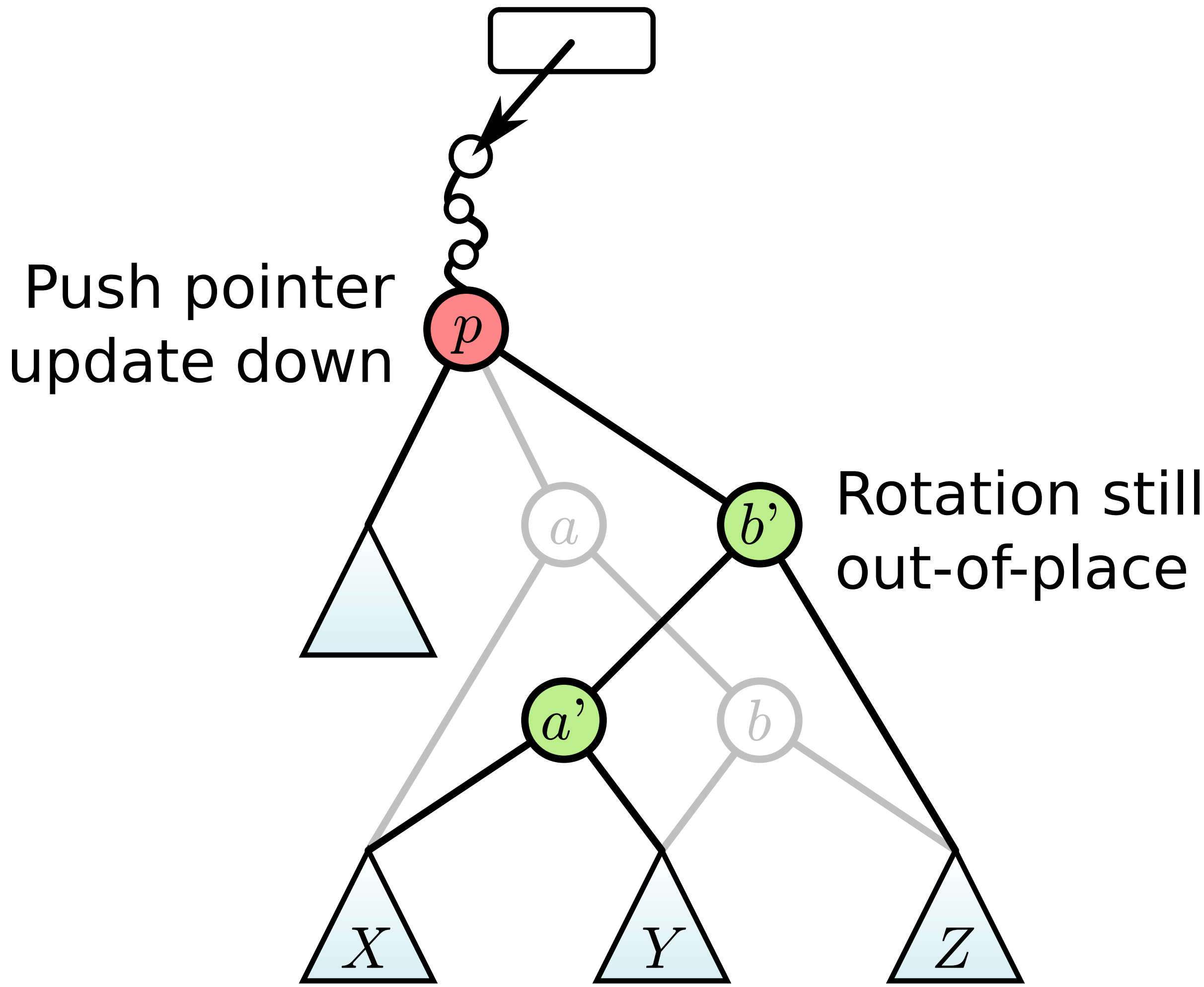
We can do better
Insertion allocates $O(\log n)$ nodes, frees $O(\log n)$ nodes



The better Bonsai tree

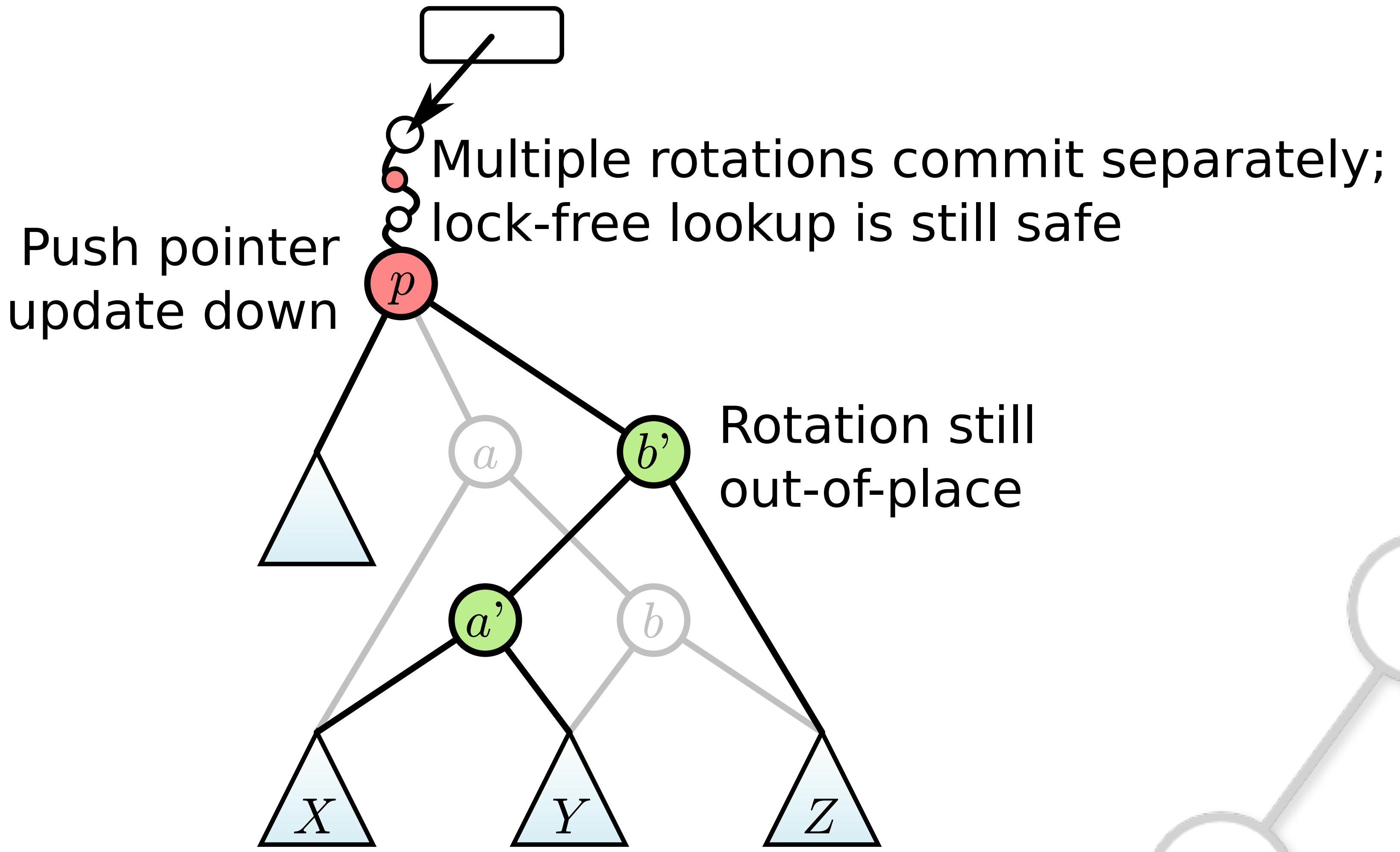
We can do better

Insertion allocates $O(\log n)$ nodes, frees $O(\log n)$ nodes



The better Bonsai tree

We can do better
Insertion allocates $O(\log n)$ nodes, frees $O(\log n)$ nodes



The better Bonsai tree

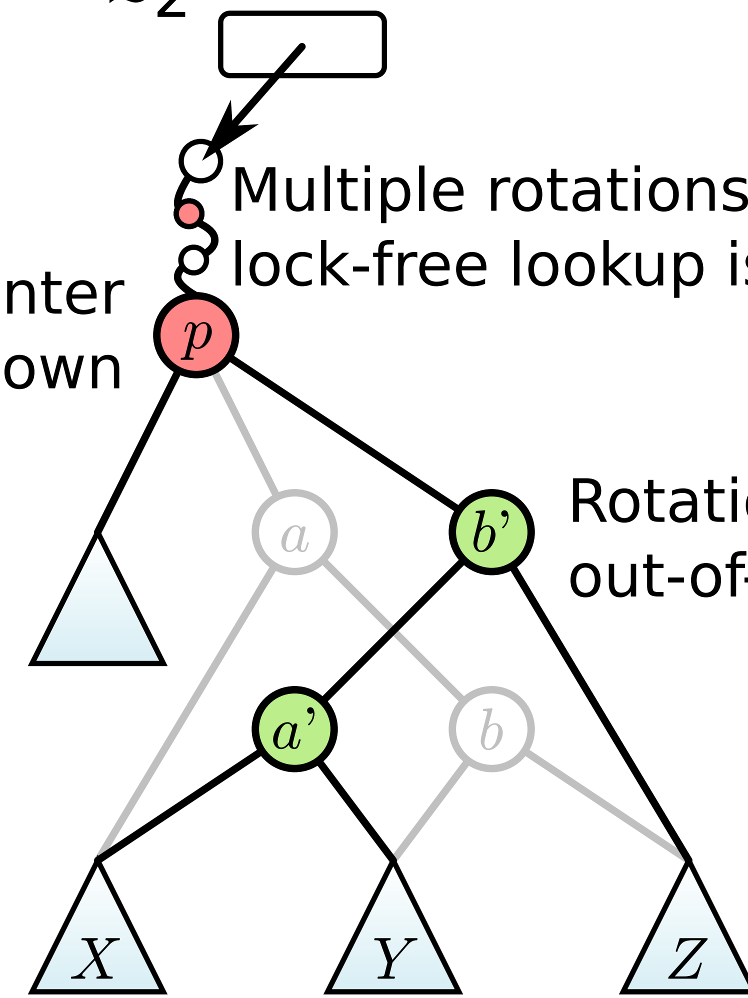
We can do better

Insertion allocates ~~$O(\log n)$~~ ~2 nodes, frees ~~$O(\log n)$~~ ~1 nodes

Multiple rotations commit separately; lock-free lookup is still safe

Push pointer update down

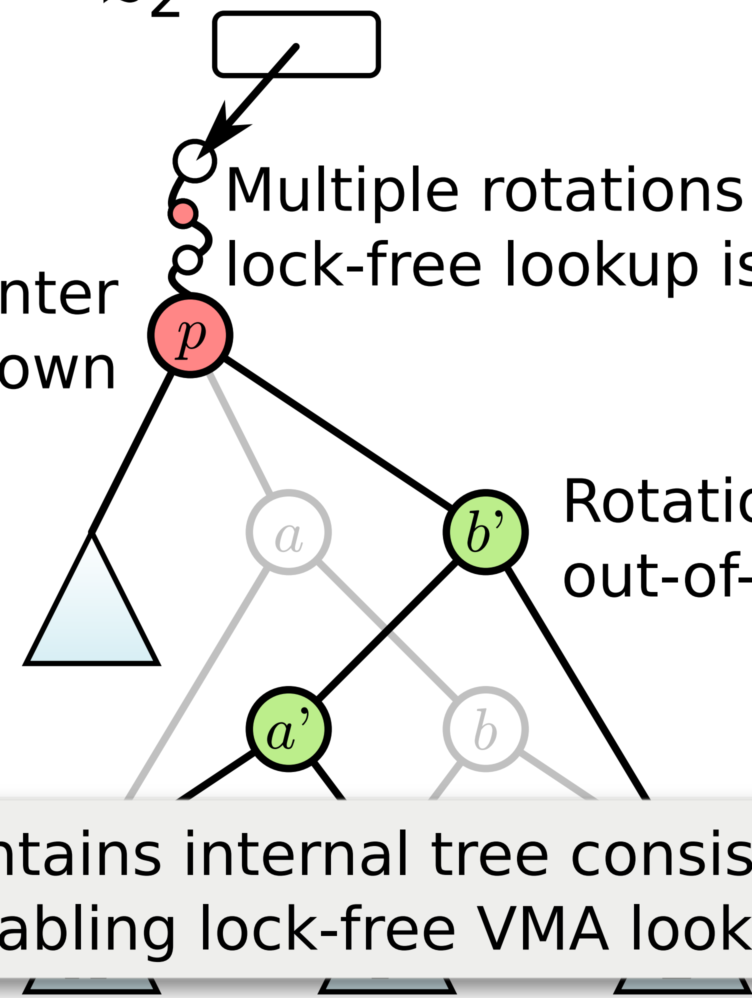
Rotation still out-of-place



The better Bonsai tree

We can do better

Insertion allocates ~~*O*(log *n*)~~ ~2 nodes, frees ~~*O*(log *n*)~~ ~1 nodes



Multiple rotations commit separately;
lock-free lookup is still safe

Push pointer
update down

Rotation still
out-of-place

Bonsai maintains internal tree consistency
while enabling lock-free VMA lookup

Page table race

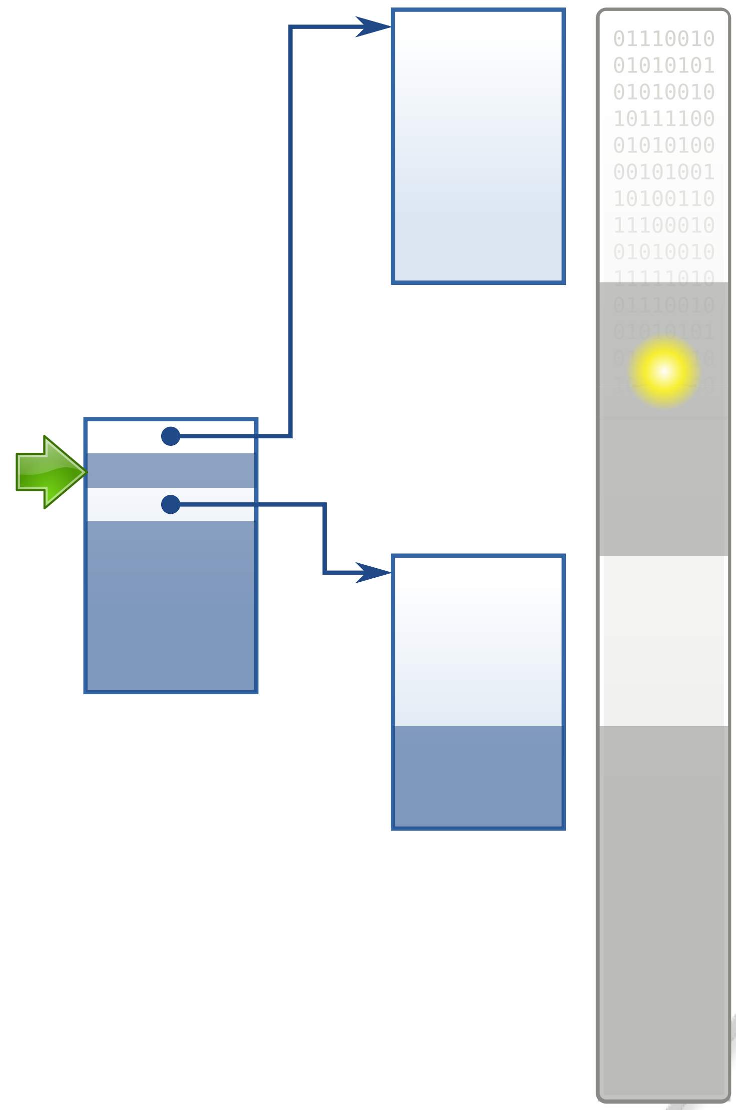
page_fault(va)

- ▷ Find VMA for va
    - ▷ Allocate page table as needed
- ▷ Allocate page and set PTE

Page table race

page_fault(va)

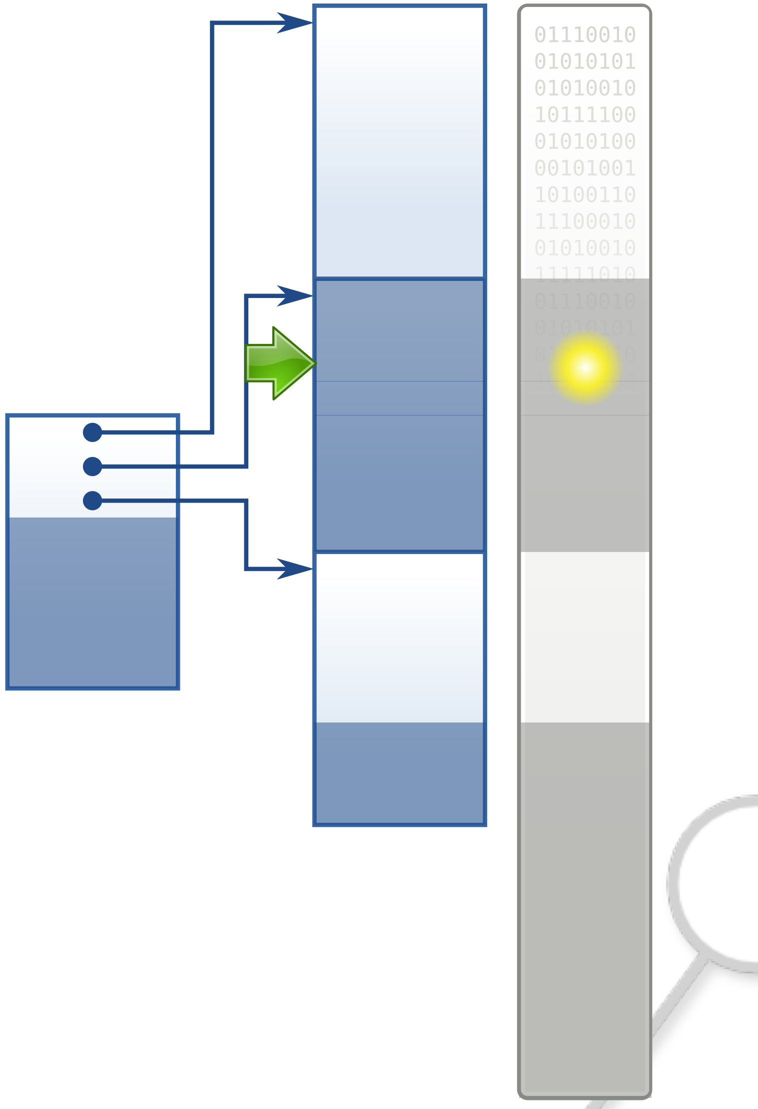
- ▹ Find VMA for va
- ▹ Walk page table tree
    - ▹ Allocate page table as needed
- ▹ Allocate page and set PTE



Page table race

page_fault(va)

- Find VMA for va
- Walk page table tree
  - Allocate page table as needed
- Allocate page and set PTE

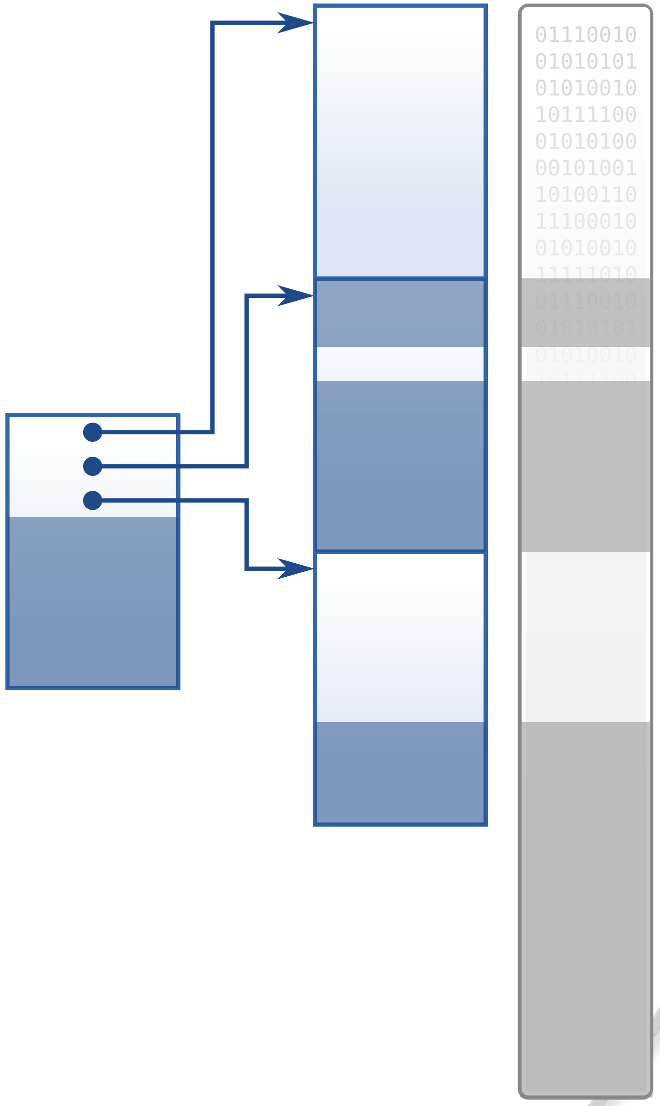


Page table race

page_fault(va)

- ▹ Find VMA for va
- ▹ Walk page table tree
  - ▹ Allocate page table as needed

- ▹ Allocate page and set PTE

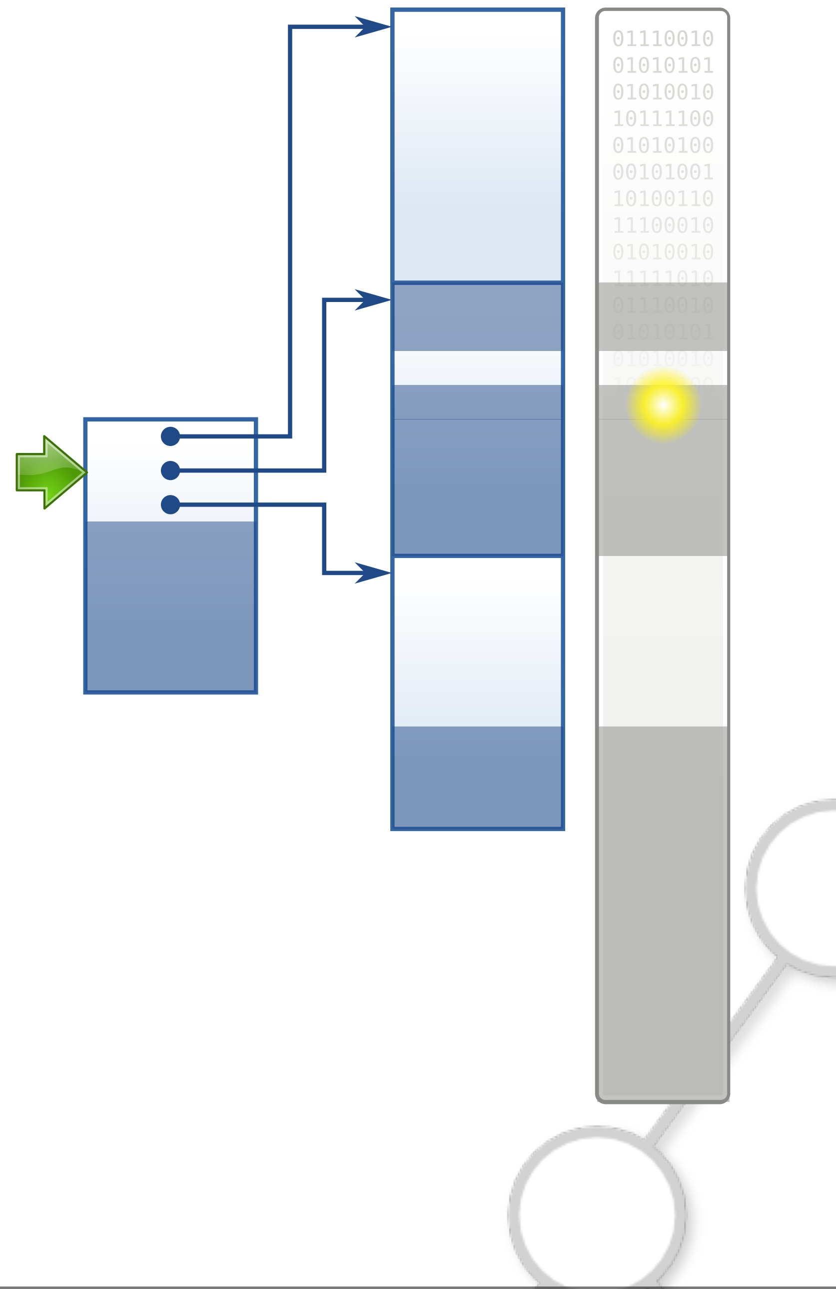


Page table race

page_fault(va)

- Find VMA for va
- Walk page table tree
    - Allocate page table as needed

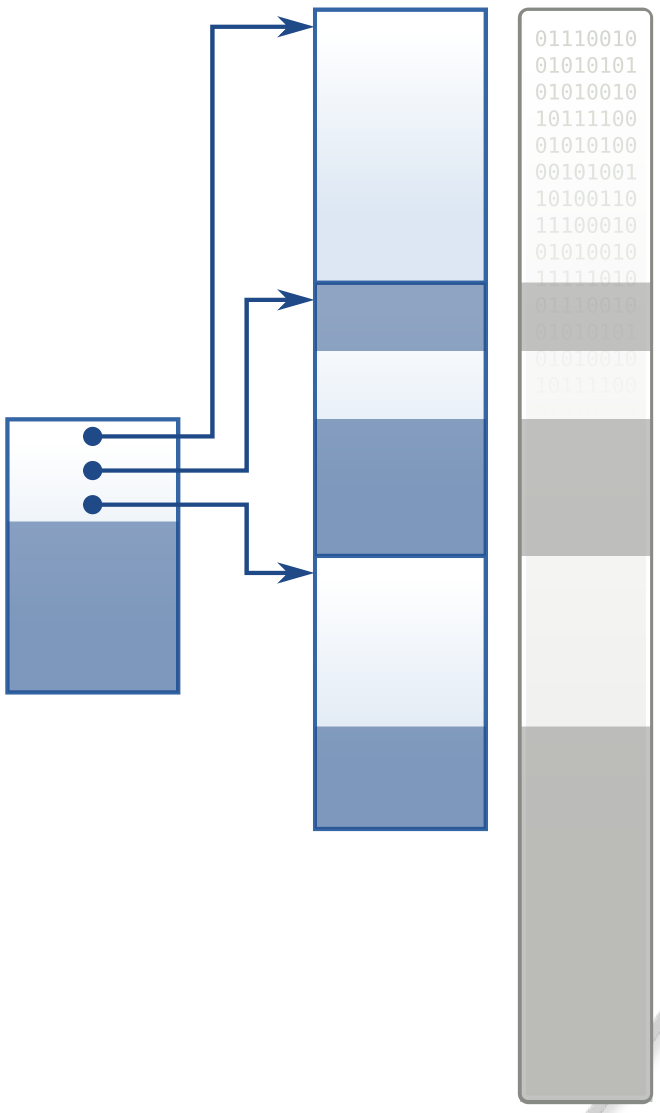
- Allocate page and set PTE



Page table race

page_fault(va)

- ▹ Find VMA for va
- ▹ Walk page table tree
  - ▹ Allocate page table as needed
- ▹ Allocate page and set PTE

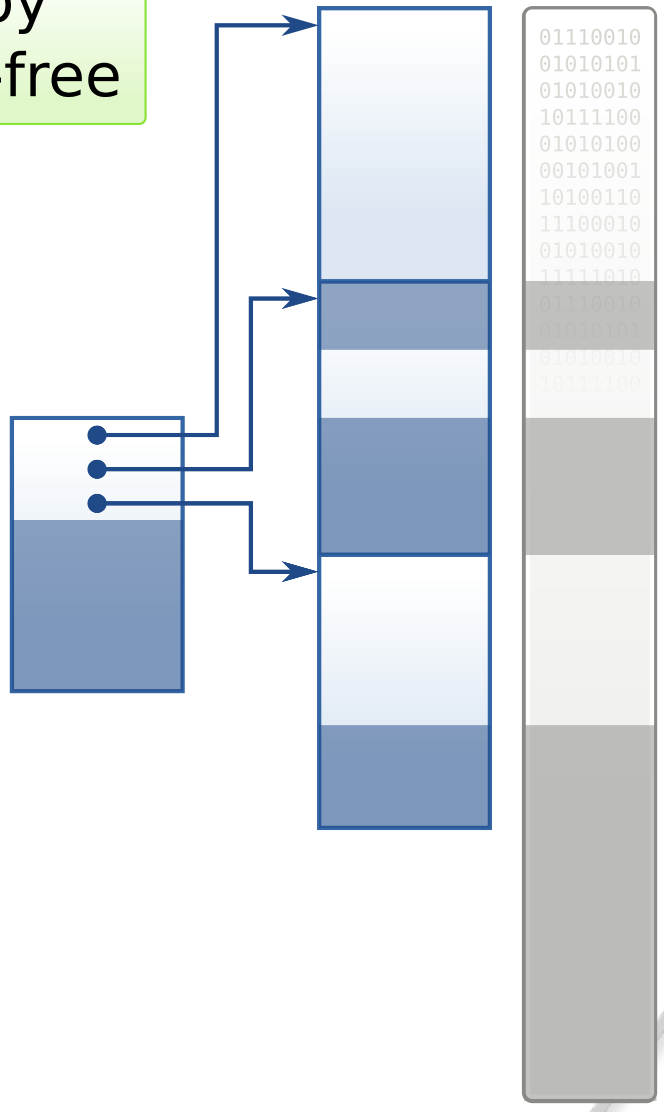


Page table race

page_fault(va)

- ▹ Find VMA for va
- ▹ Walk page table tree
  - ▹ Allocate page table as needed
- ▹ Allocate page and set PTE

Previously protected by the AS lock; now lock-free



Page table race

page_fault(va)

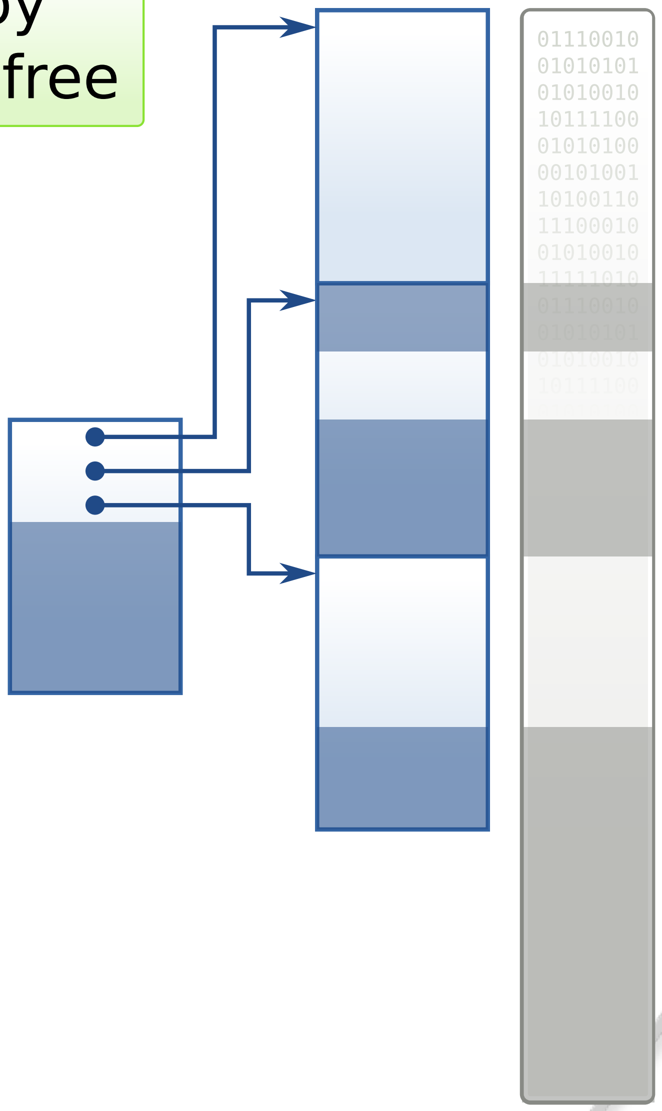
- ▹ Find VMA for va
- ▹ Walk page table tree
  - ▹ Allocate page table as needed

Previously protected by the AS lock; now lock-free

- ▹ Allocate page and set PTE

munmap(start, end)

- ▹ Removes/adjusts VMA's
- ▹ Clear page table entries and free pages
- ▹ Free unused page tables



Page table race

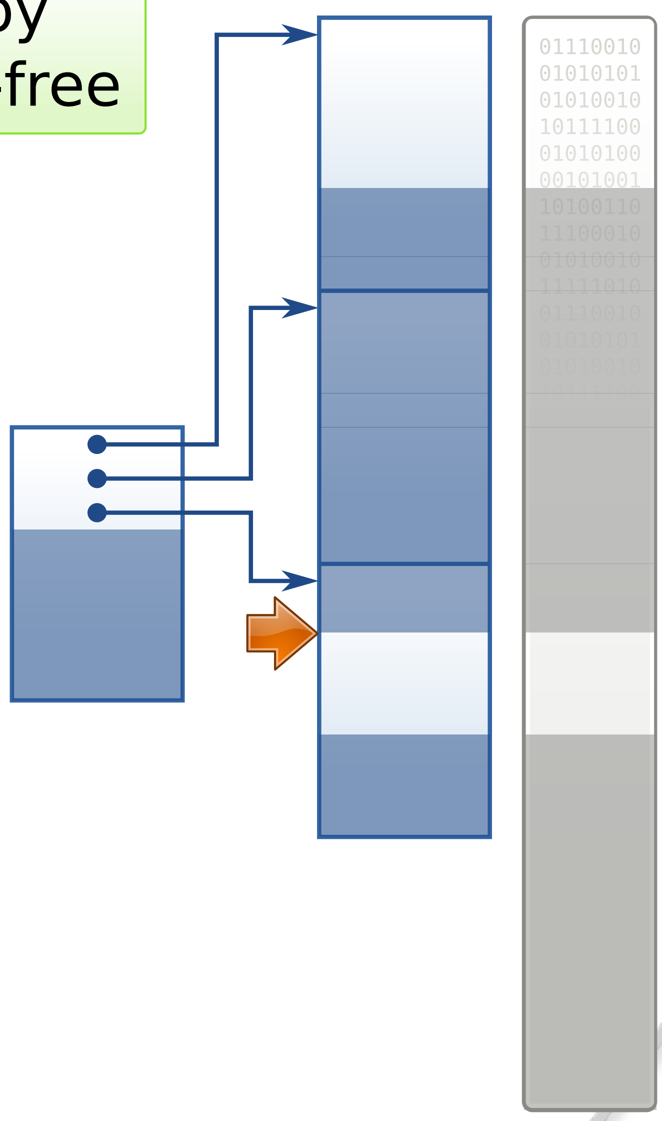
page_fault(va)

- ▹ Find VMA for va
- ▹ Walk page table tree
  - ▹ Allocate page table as needed
- ▹ Allocate page and set PTE

Previously protected by the AS lock; now lock-free

munmap(start, end)

- ▹ Removes/adjusts VMA's
- ▹ Clear page table entries and free pages
- ▹ Free unused page tables



Page table race

page_fault(va)

▹ Find VMA for va

▹ Walk page table tree

  ▹ Allocate page table as needed

Previously protected by the AS lock; now lock-free

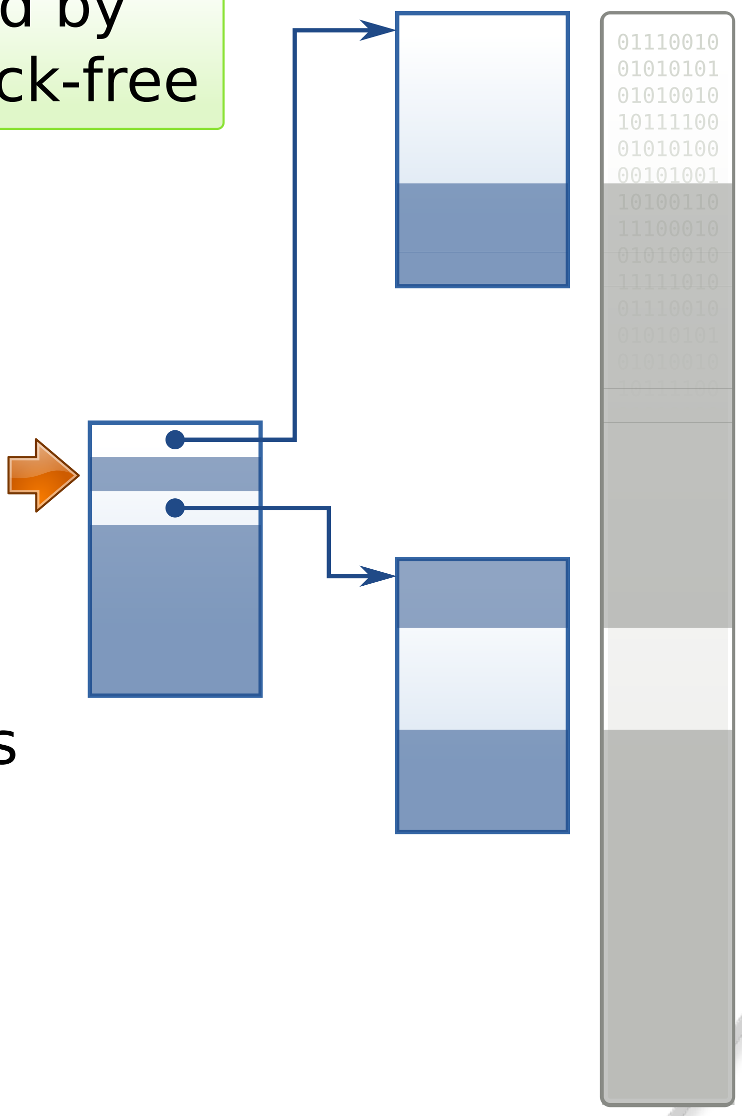
▹ Allocate page and set PTE

munmap(start, end)

▹ Removes/adjusts VMA's

▹ Clear page table entries and free pages

▹ Free unused page tables

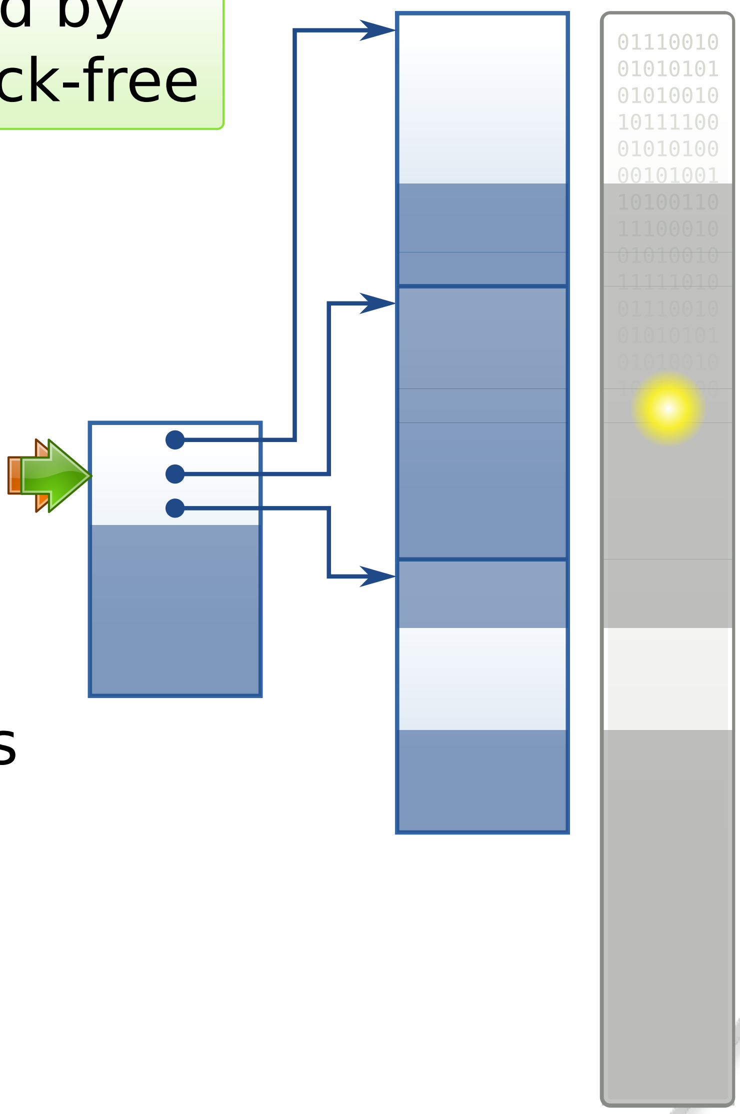


Page table race

Previously protected by the AS lock; now lock-free

page_fault(va)

- ▹ Find VMA for va
- ▹ Walk page table tree
  - ▹ Allocate page table as needed
- ▹ Allocate page and set PTE



munmap(start, end)

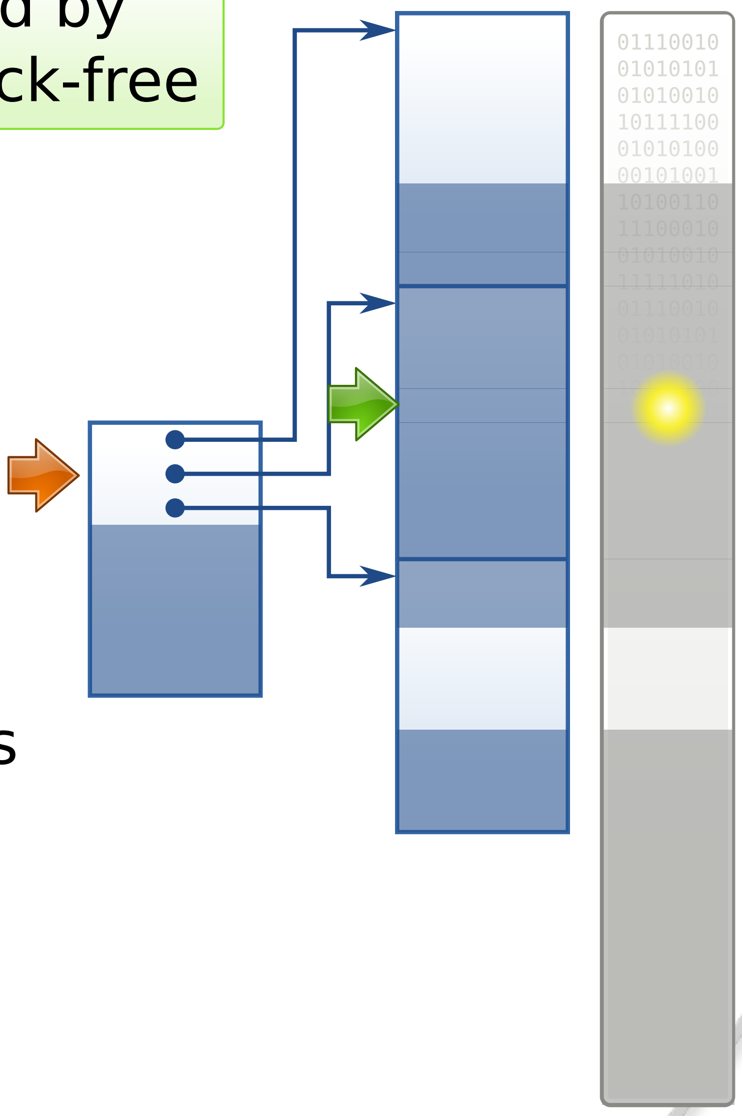
- ▹ Removes/adjusts VMA's
- ▹ Clear page table entries and free pages
- ▹ Free unused page tables

Page table race

Previously protected by the AS lock; now lock-free

page_fault(va)
- ▹ Find VMA for va
- ▹ Walk page table tree
  - ▹ Allocate page table as needed
- ▹ Allocate page and set PTE

munmap(start, end)
- ▹ Removes/adjusts VMA's
- ▹ Clear page table entries and free pages
- ▹ Free unused page tables



Page table race

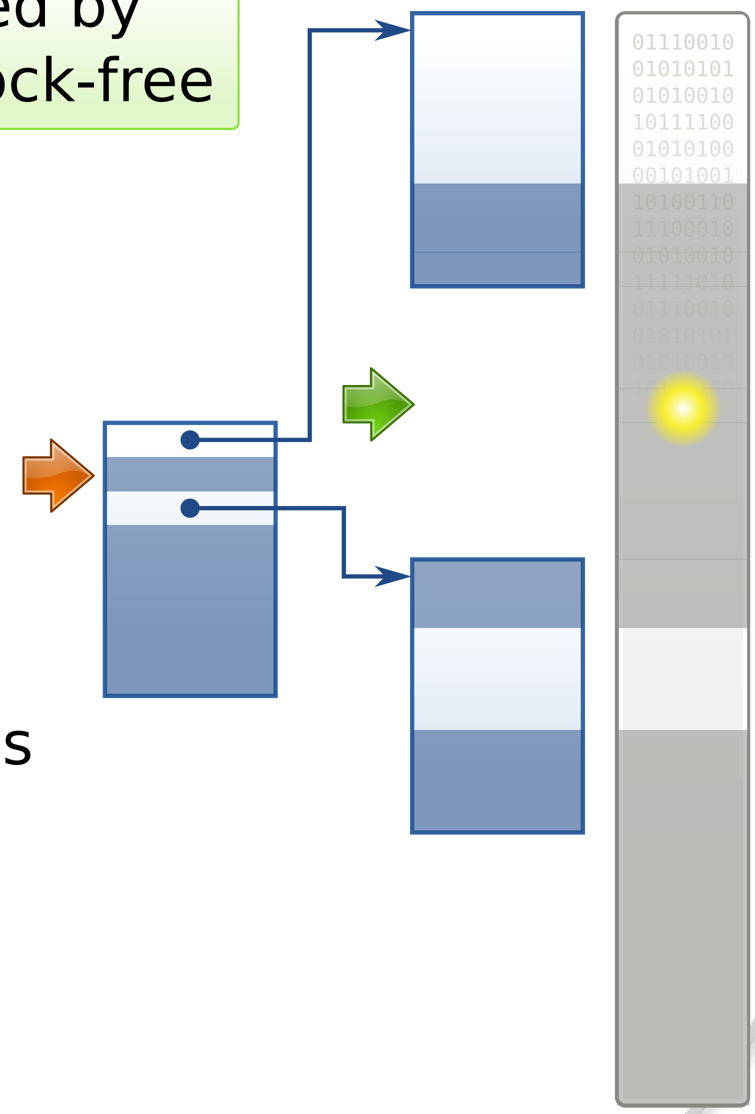
Previously protected by the AS lock; now lock-free

page_fault(va)

- ▹ Find VMA for va
- ▹ Walk page table tree
  - ▹ Allocate page table as needed
- ▹ Allocate page and set PTE

munmap(start, end)

- ▹ Removes/adjusts VMA's
- ▹ Clear page table entries and free pages
- ▹ Free unused page tables



Page table race

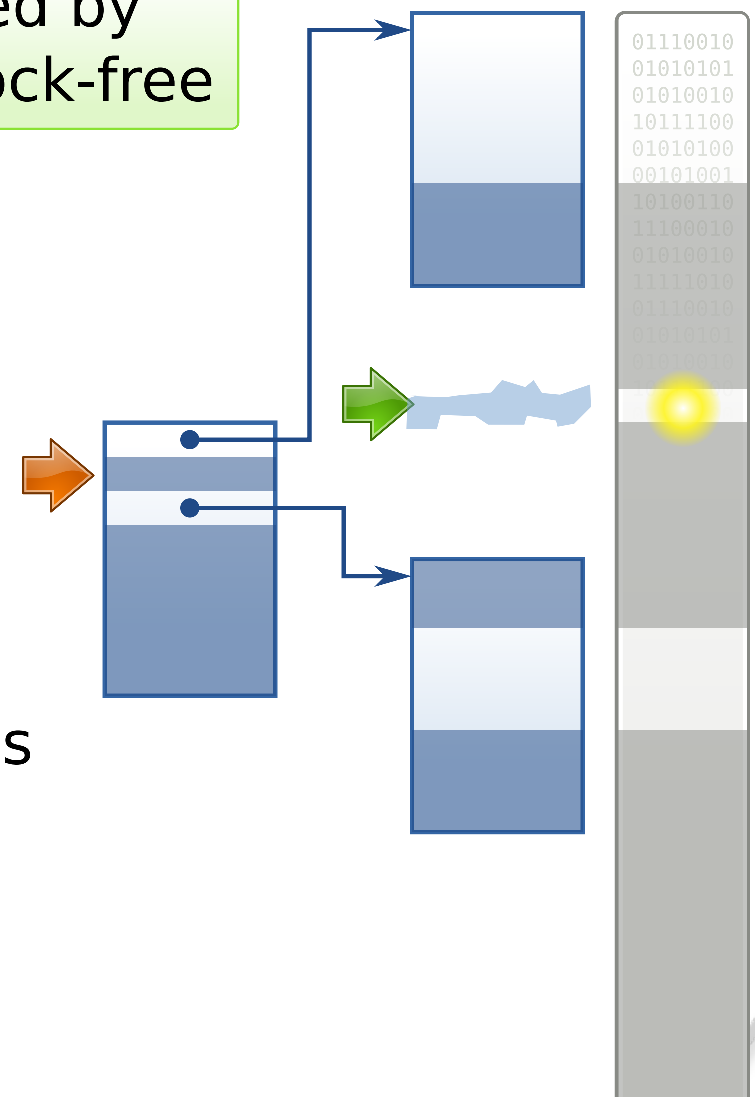
Previously protected by the AS lock; now lock-free

page_fault(va)

- ▹ Find VMA for va
- ▹ Walk page table tree
  - ▹ Allocate page table as needed
- ▹ Allocate page and set PTE

munmap(start, end)

- ▹ Removes/adjusts VMA's
- ▹ Clear page table entries and free pages
- ▹ Free unused page tables



Page table race

page_fault(va)

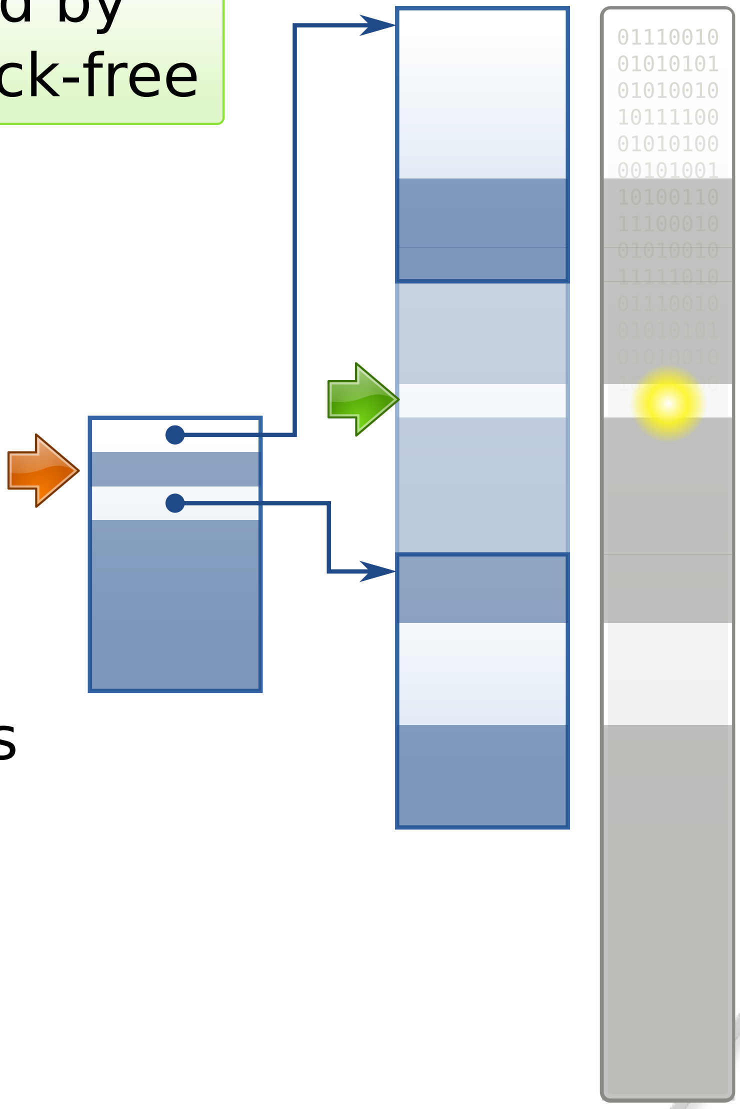
- Find VMA for va
- Walk page table tree
  - Allocate page table as needed
- Allocate page and set PTE

Previously protected by the AS lock; now lock-free

munmap(start, end)

- Removes/adjusts VMA's
- Clear page table entries and free pages
- RCU-Free unused page tables

Makes lock-free walk safe



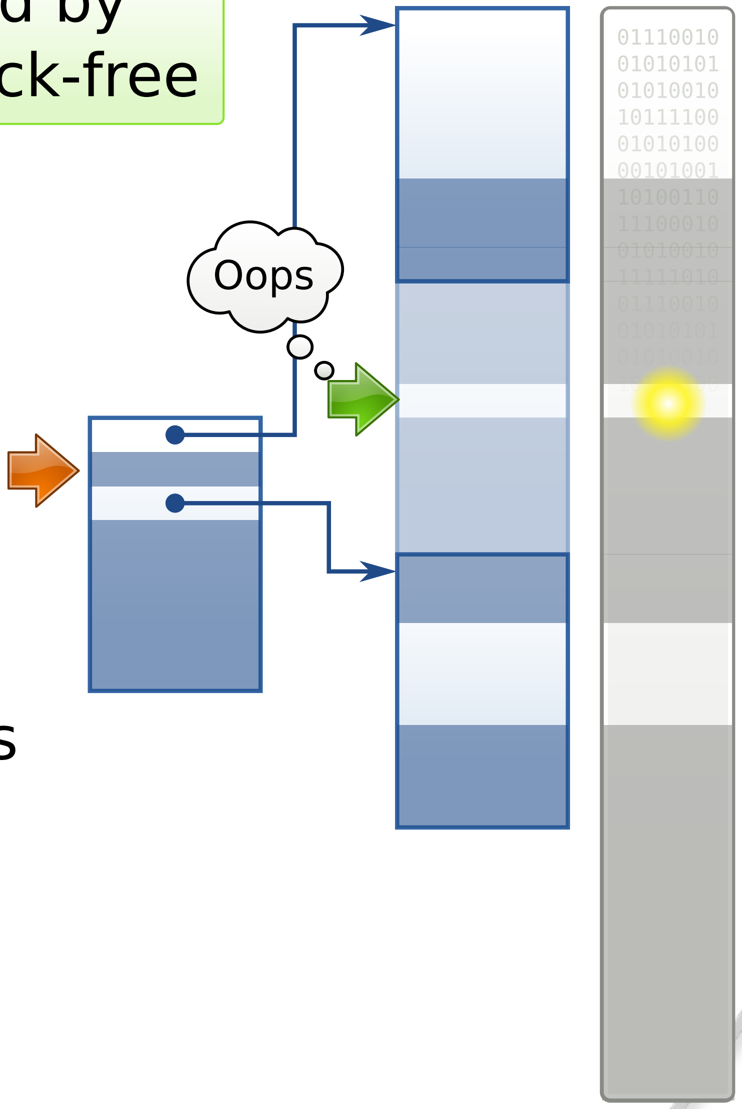
Page table race

page_fault(va)
- ▹ Find VMA for va
- ▹ Walk page table tree
  - ▹ Allocate page table as needed
- ▹ Allocate page and set PTE

Previously protected by the AS lock; now lock-free

munmap(start, end)
- ▹ Removes/adjusts VMA's
- ▹ Clear page table entries and free pages
- ▹ RCU-Free unused page tables

Makes lock-free walk safe

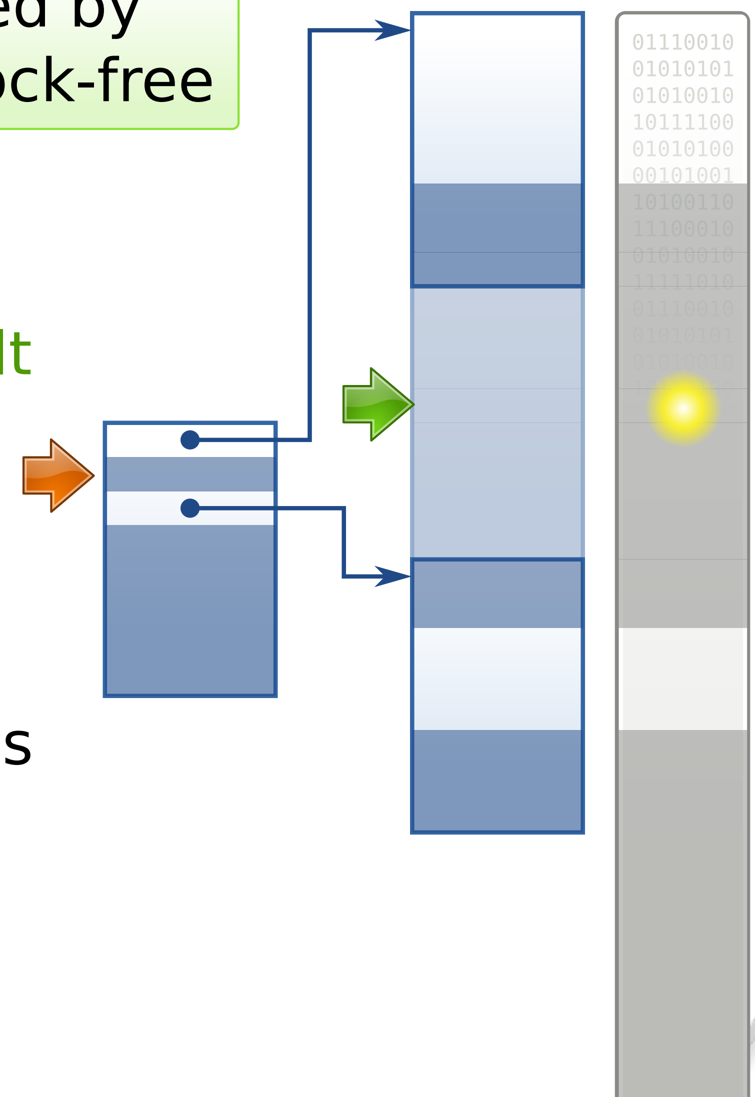


Page table race

Previously protected by the AS lock; now lock-free

page_fault(va)

- ▹ Find VMA for va
- ▹ Walk page table tree
  - ▹ Allocate page table as needed
- ▹ If VMA unlinked or va invalid, retry fault
- ▹ Allocate page and set PTE



munmap(start, end)

- ▹ Removes/adjusts VMA's
- ▹ Clear page table entries and free pages
- ▹ RCU-Free unused page tables

Makes lock-free walk safe

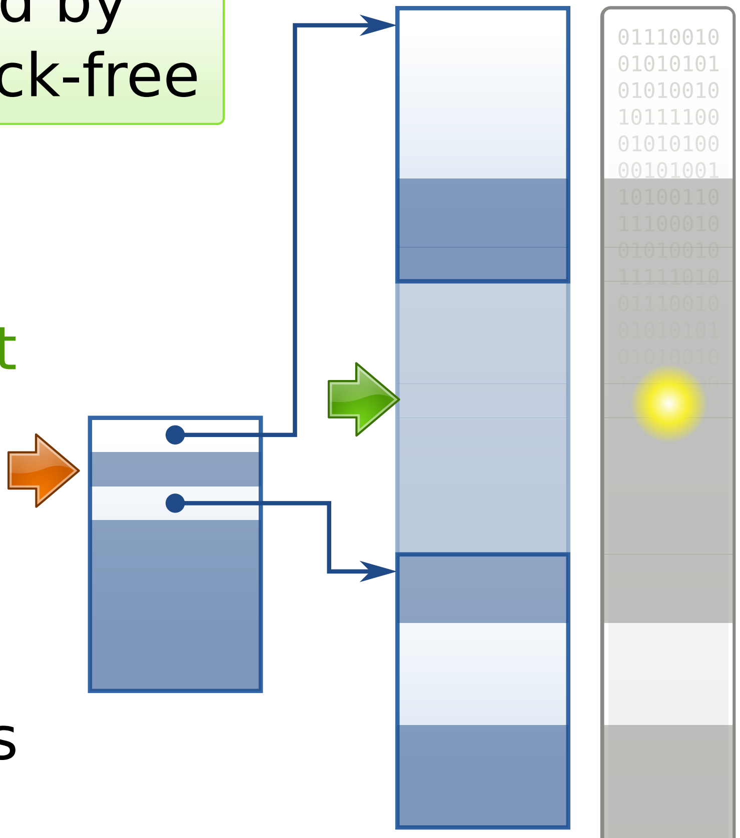
Page table race

page_fault(va)
- ▹ Find VMA for va
- ▹ Walk page table tree
  - ▹ Allocate page table as needed
- ▹ If VMA unlinked or va invalid, retry fault
- ▹ Allocate page and set PTE

Previously protected by the AS lock; now lock-free

munmap(start, end)
- ▹ Removes/adjusts VMA's
- ▹ Clear page table entries and free pages
- ▹ RCU-Free unused page tables

Makes lock-free walk safe



RCU + race detection + retries is a potent combination.

Evaluation

Evaluation

How much does our design help?

Evaluation

How much does our design help?

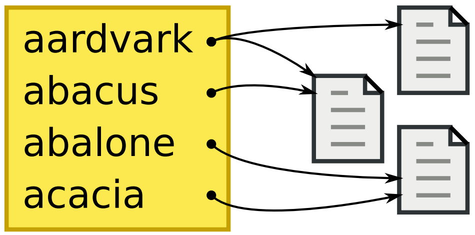
Is this just a stop-gap solution?

Approach

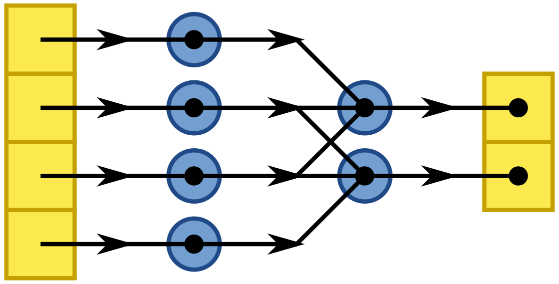
Implemented in
Linux 2.6.37

Approach

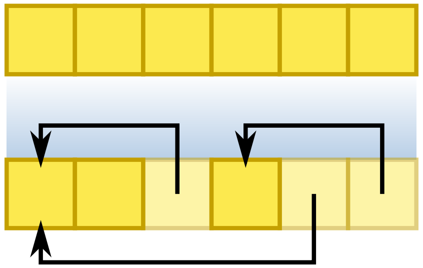
Implemented in
Linux 2.6.37



Psearchy



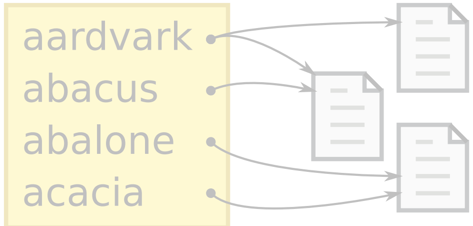
Metis



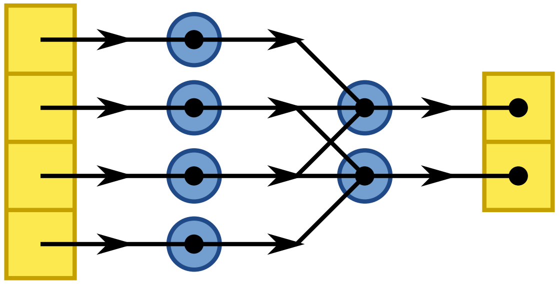
Dedup

Approach

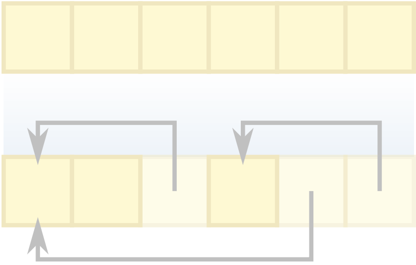
Implemented in
Linux 2.6.37



Psearchy

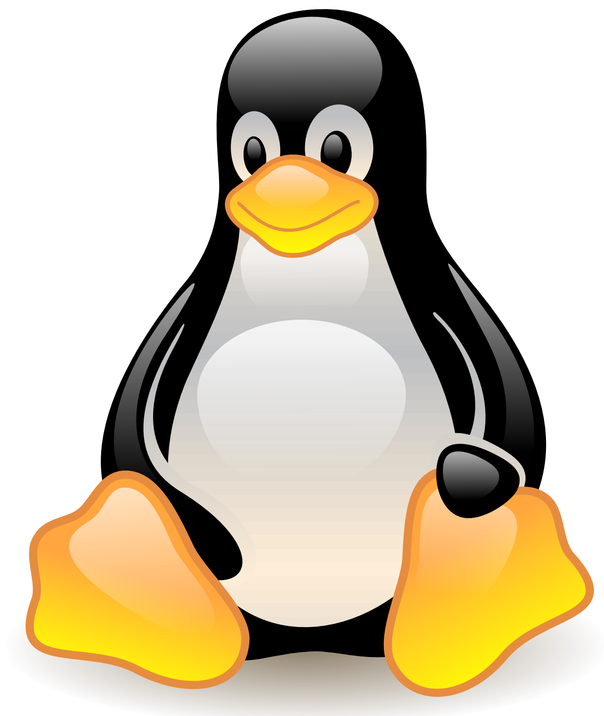


Metis

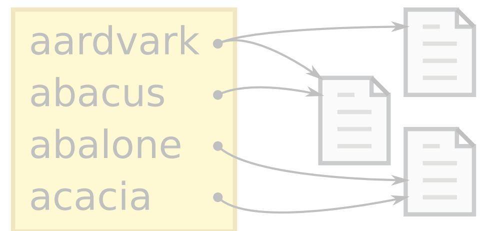


Dedup

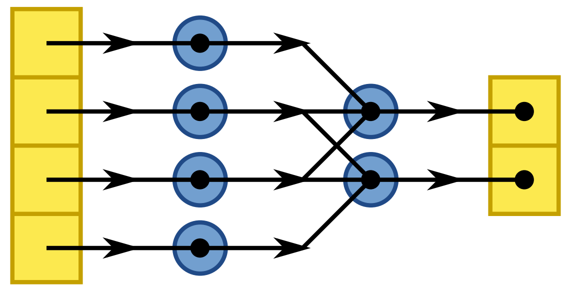
Approach



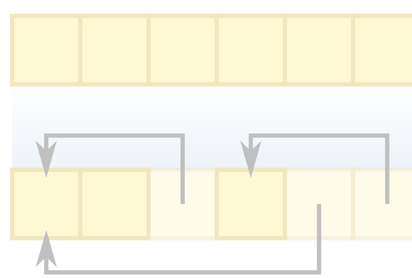
Implemented in Linux 2.6.37



Psearchy



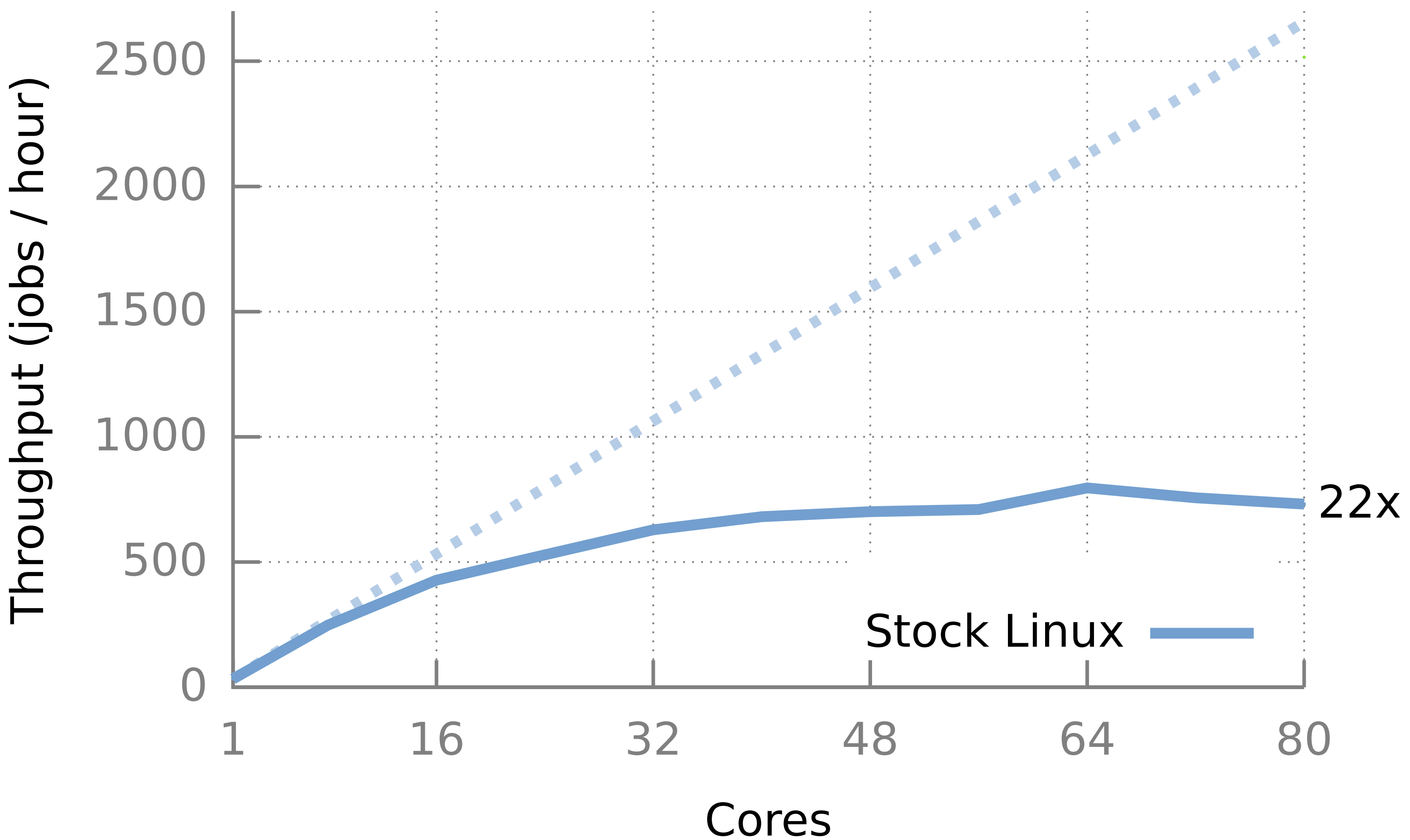
Metis



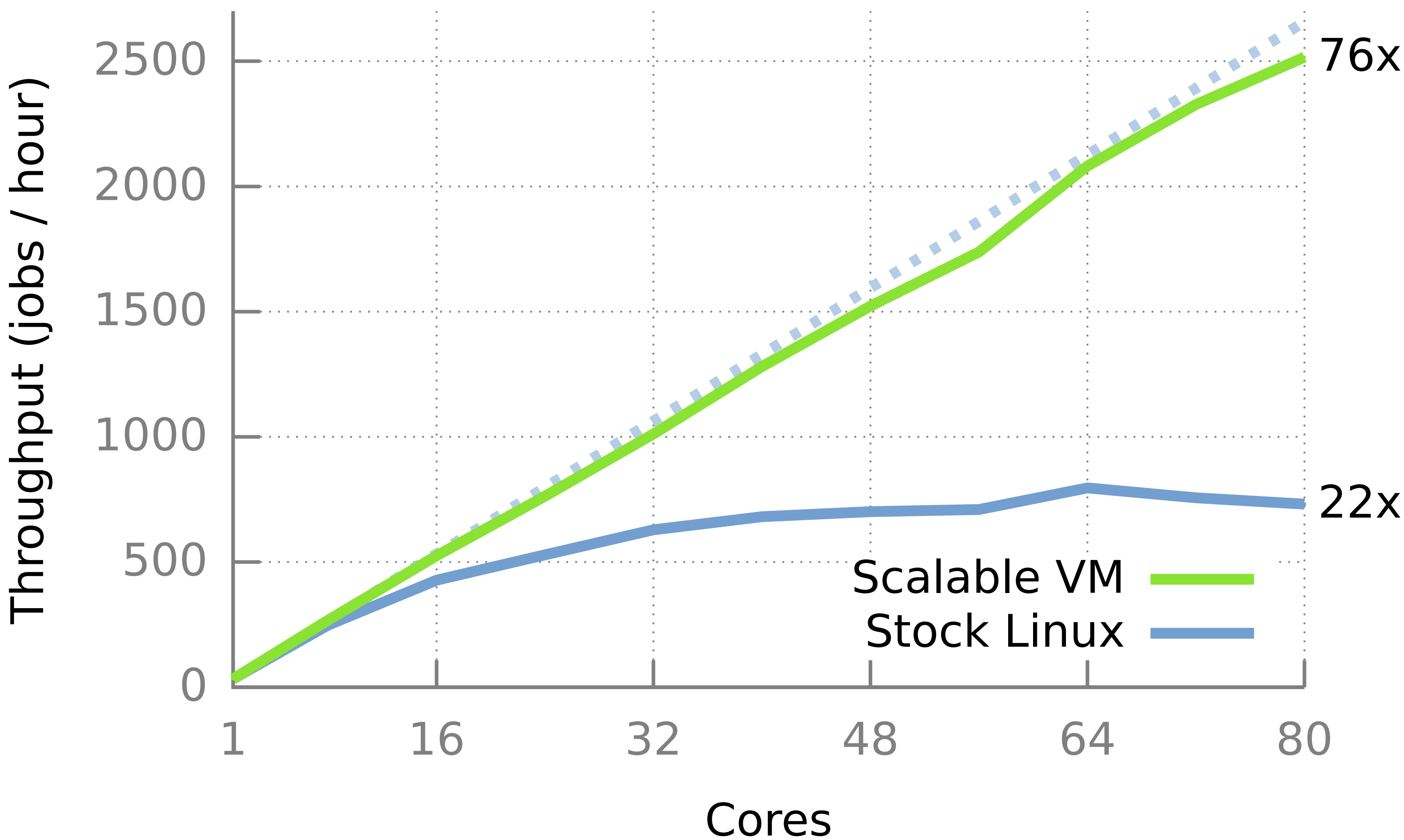
Dedup

80 core Intel Xeon

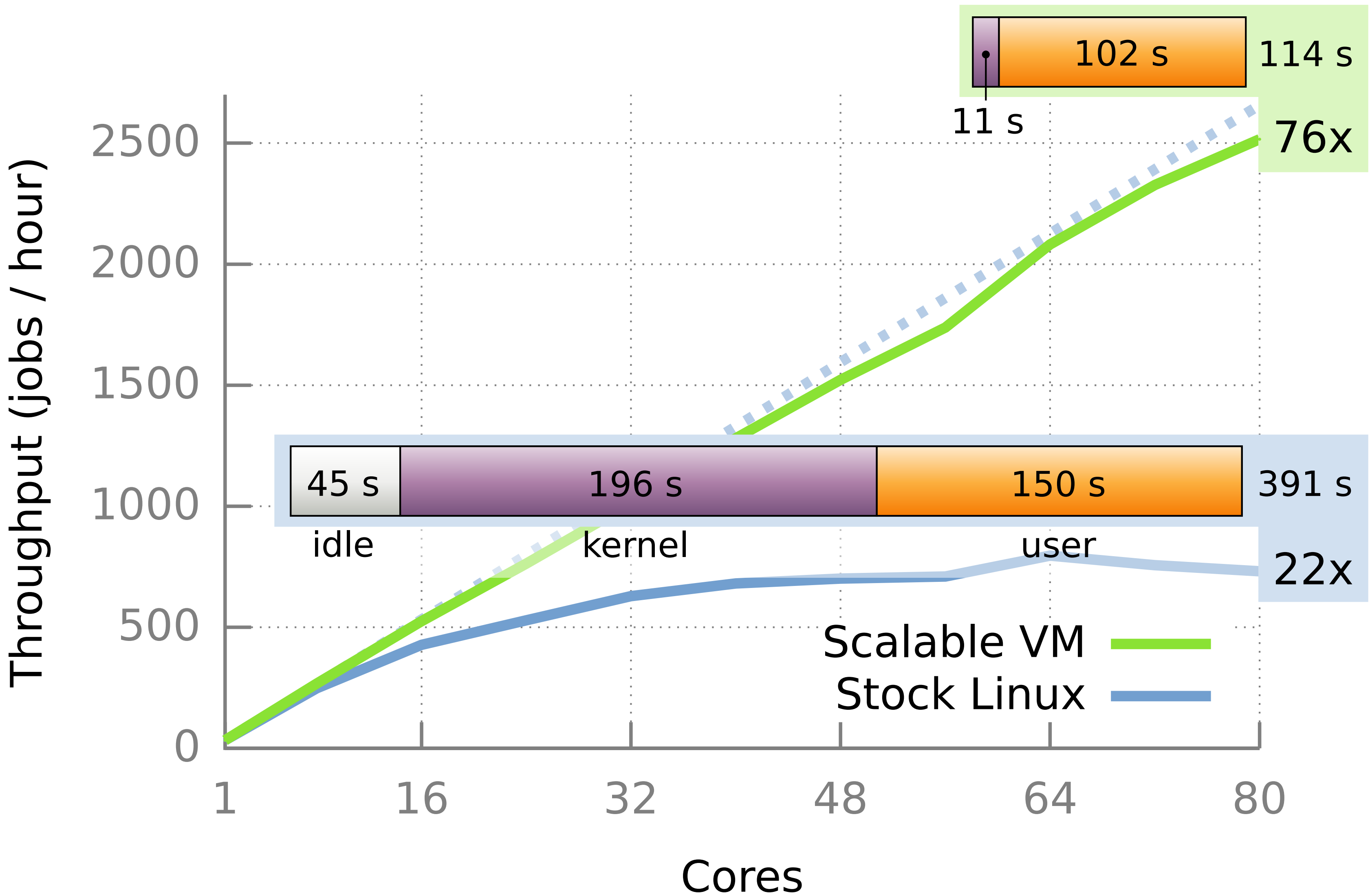
Application scalability (Metis)



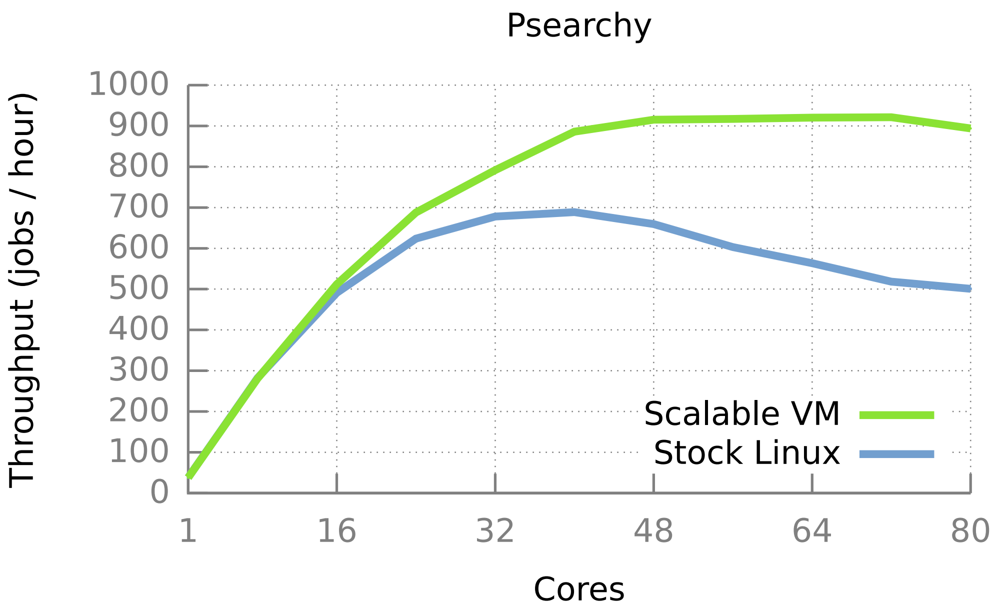
Application scalability (Metis)

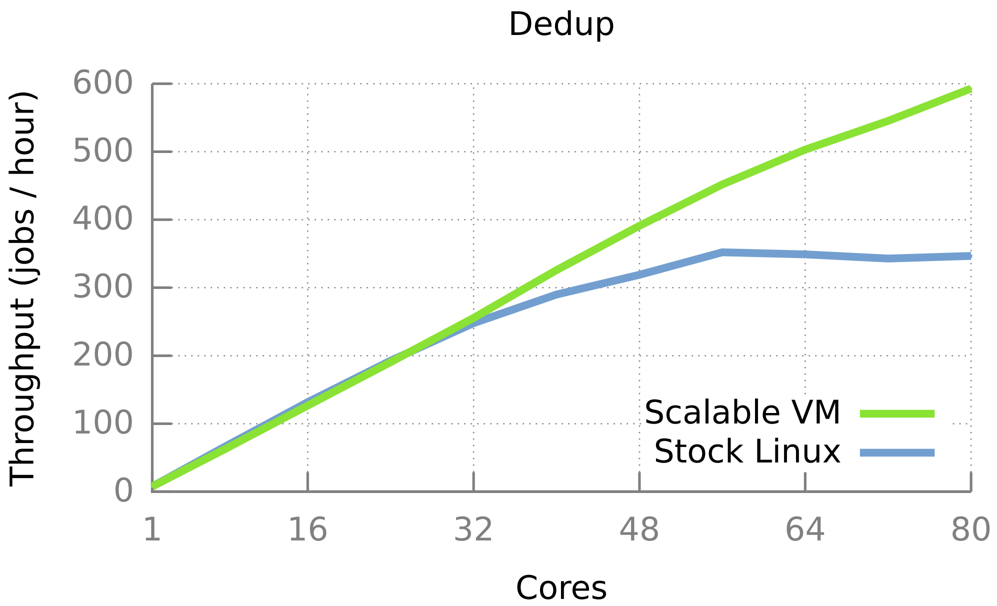


Application scalability (Metis)

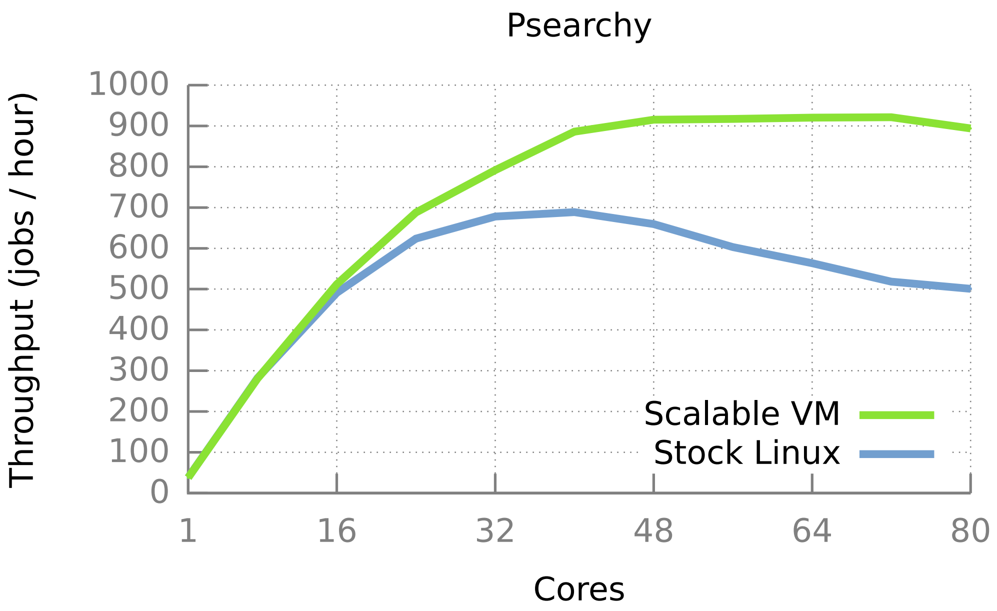


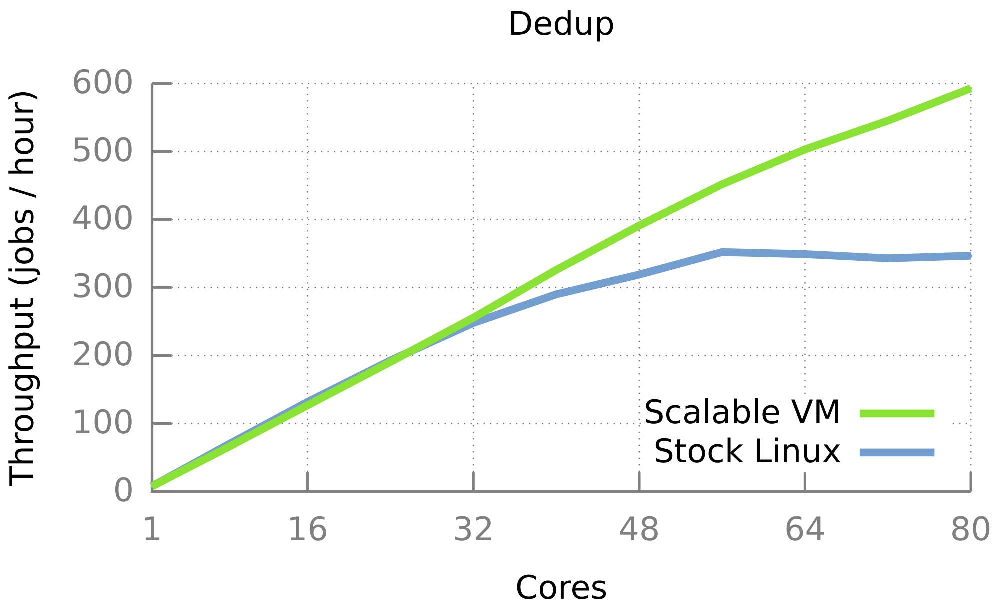
Application scalability (Psearchy, dedup)





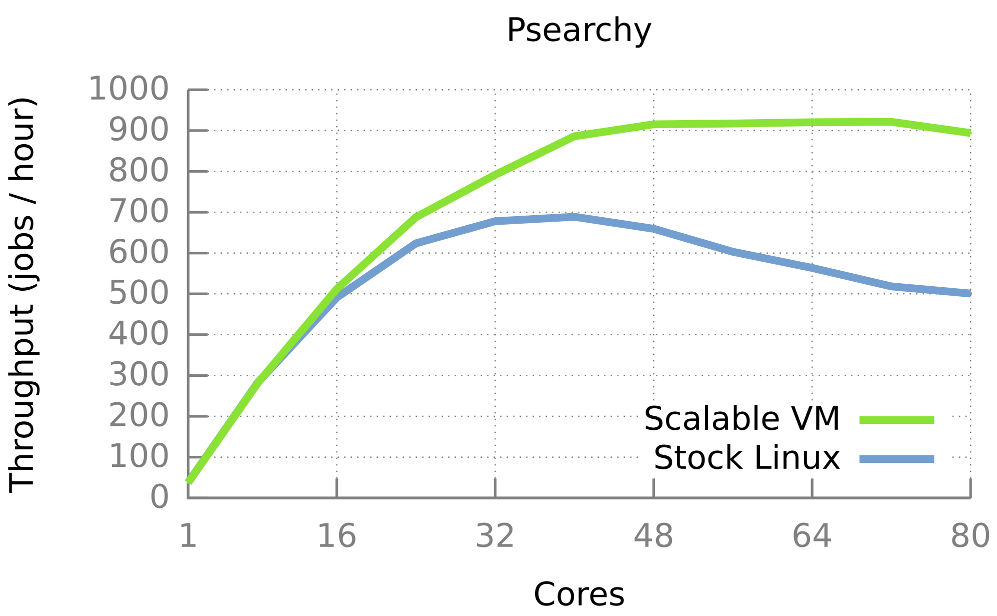
Application scalability (Psearchy, dedup)

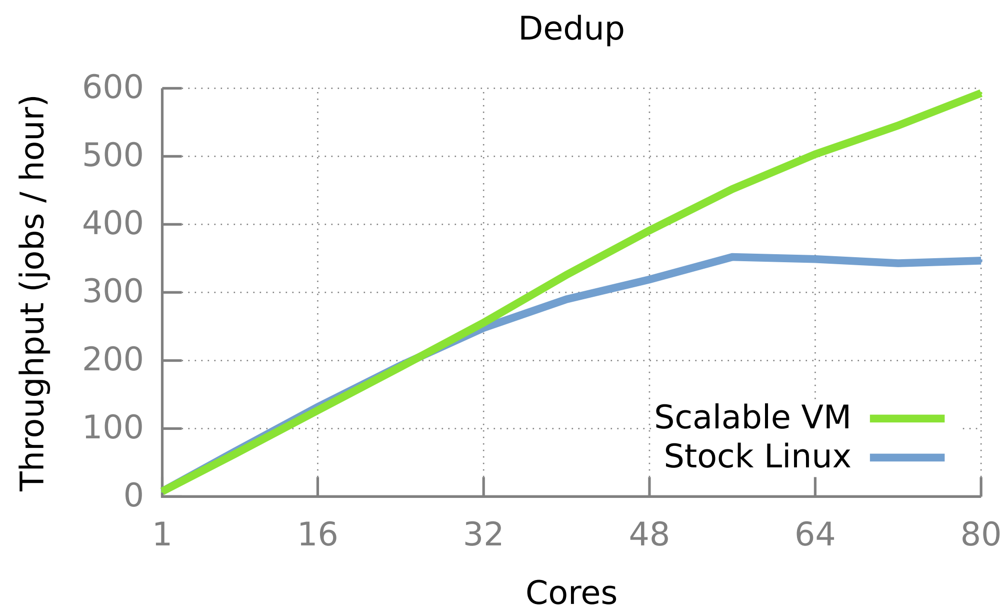




Address space scalability matters for real applications.
Our design eliminates kernel bottlenecks.

Application scalability (Psearchy, dedup)





Address space scalability matters for real applications.
Our design eliminates kernel bottlenecks.

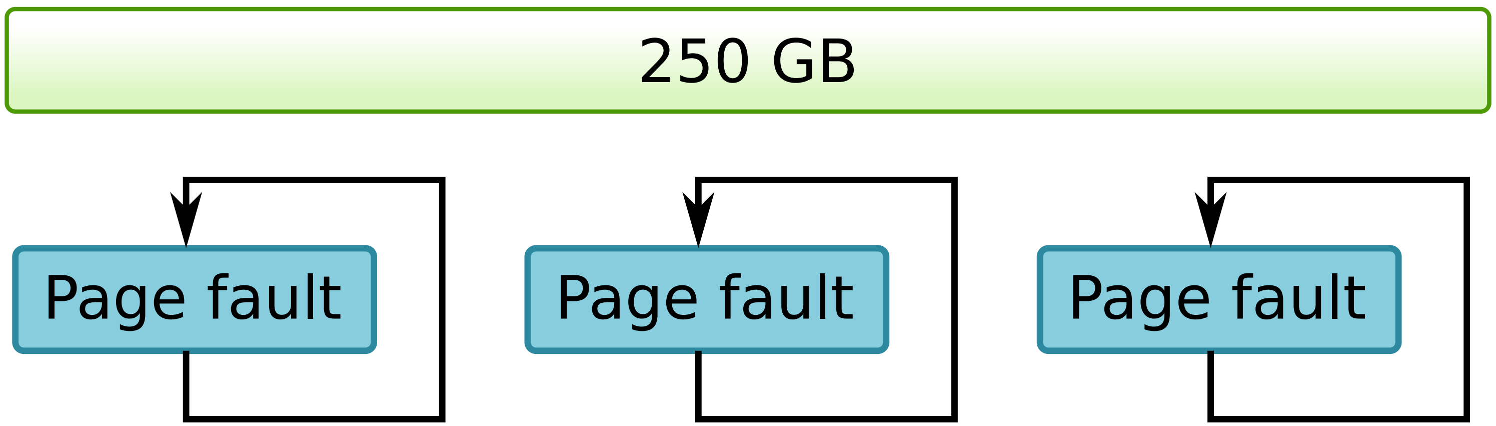
Metis, Psearchy and Dedup are >80% user space
How does the VM system perform in isolation?

Microbenchmark design

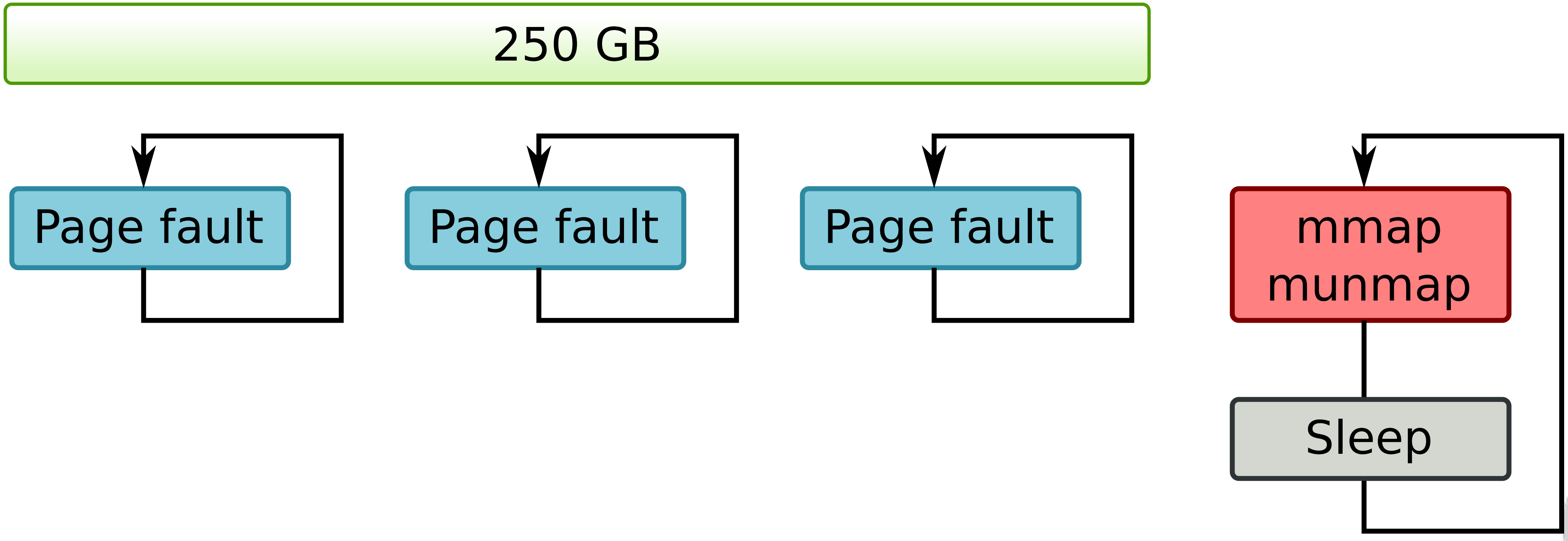
Microbenchmark design

250 GB

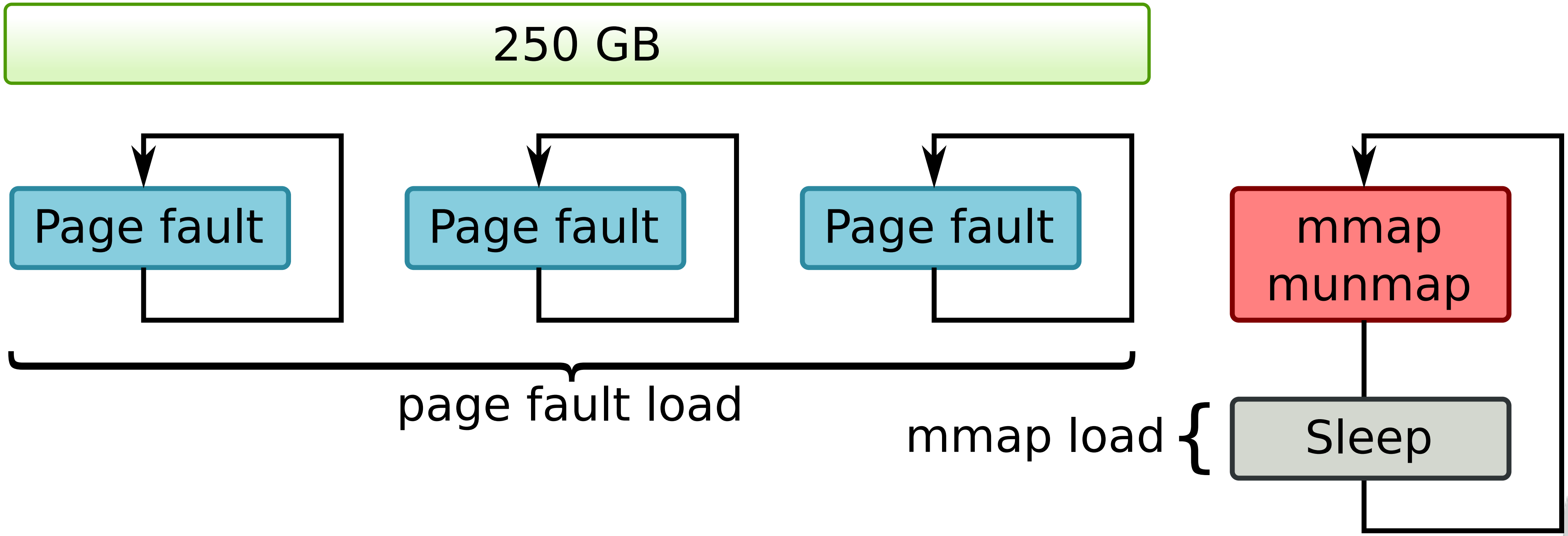
Microbenchmark design



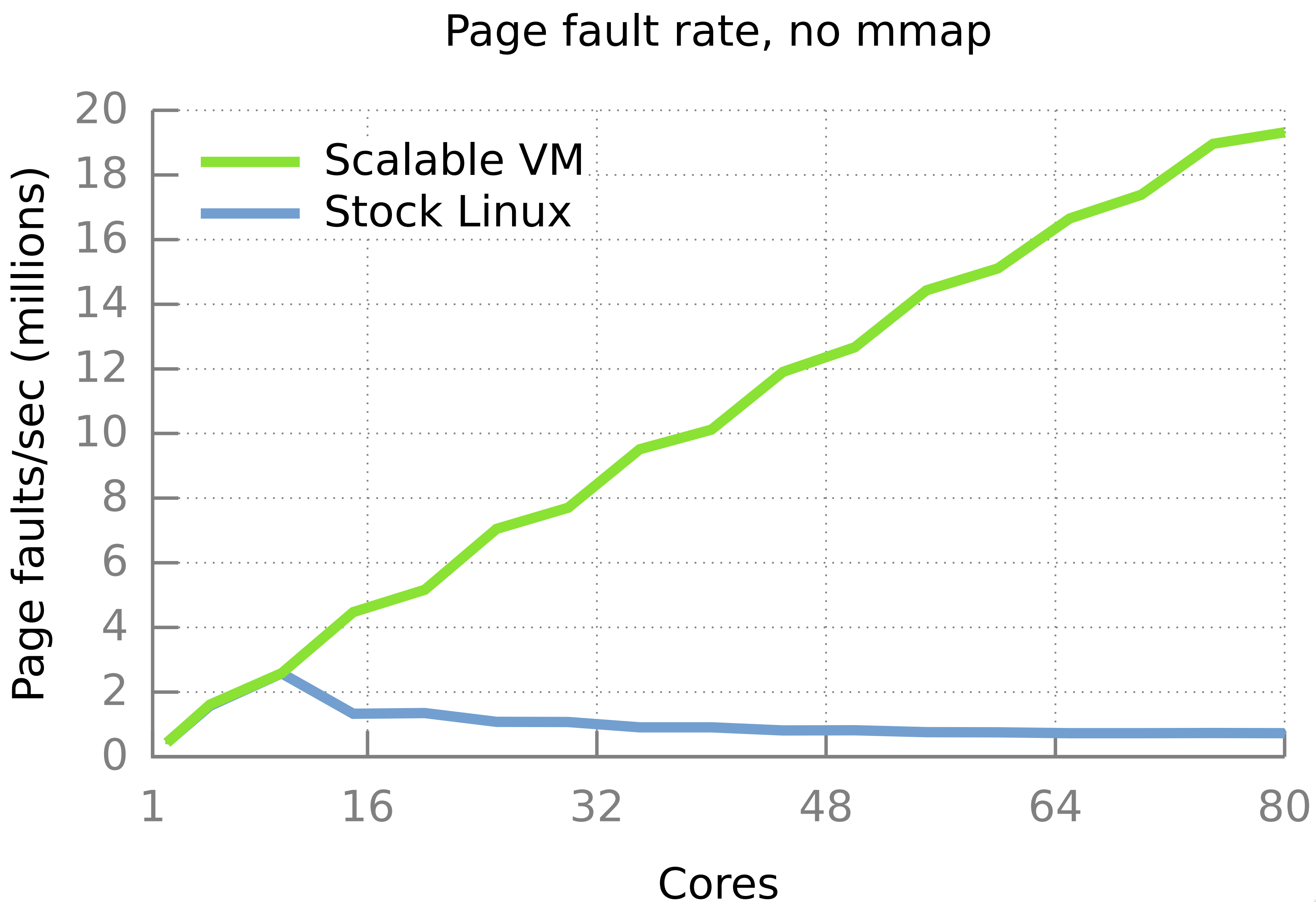
Microbenchmark design



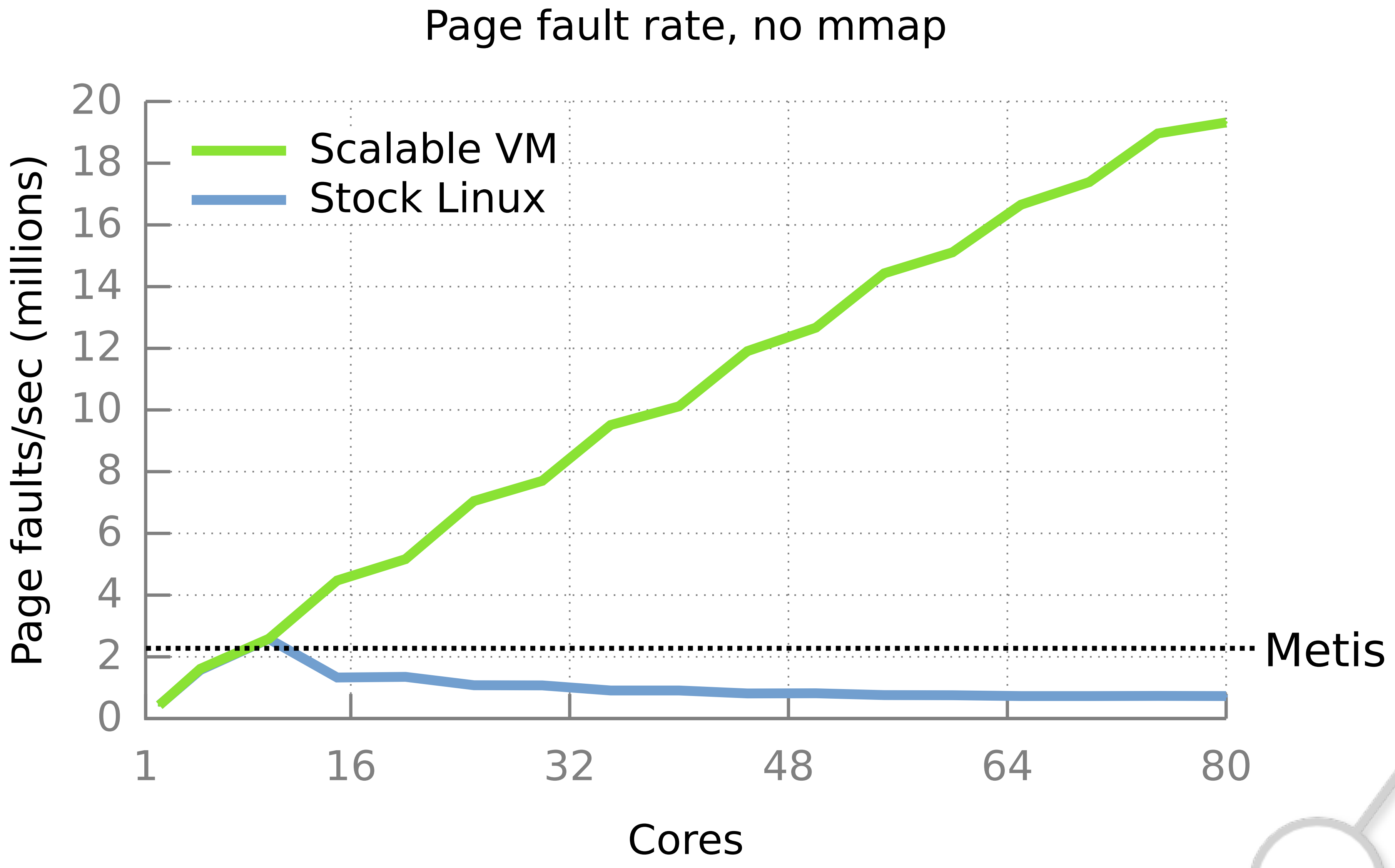
Microbenchmark design



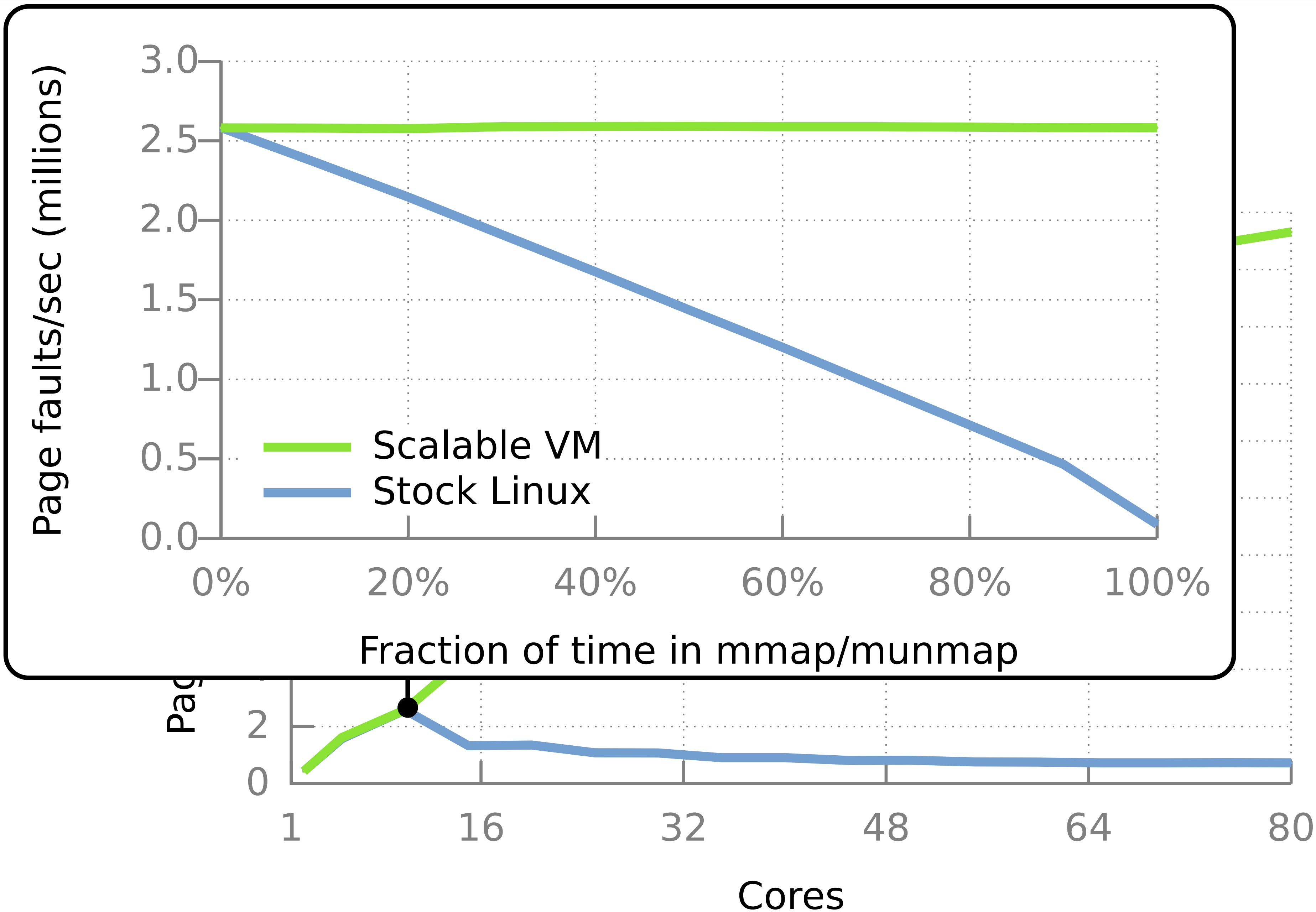
Page faults have constant cost



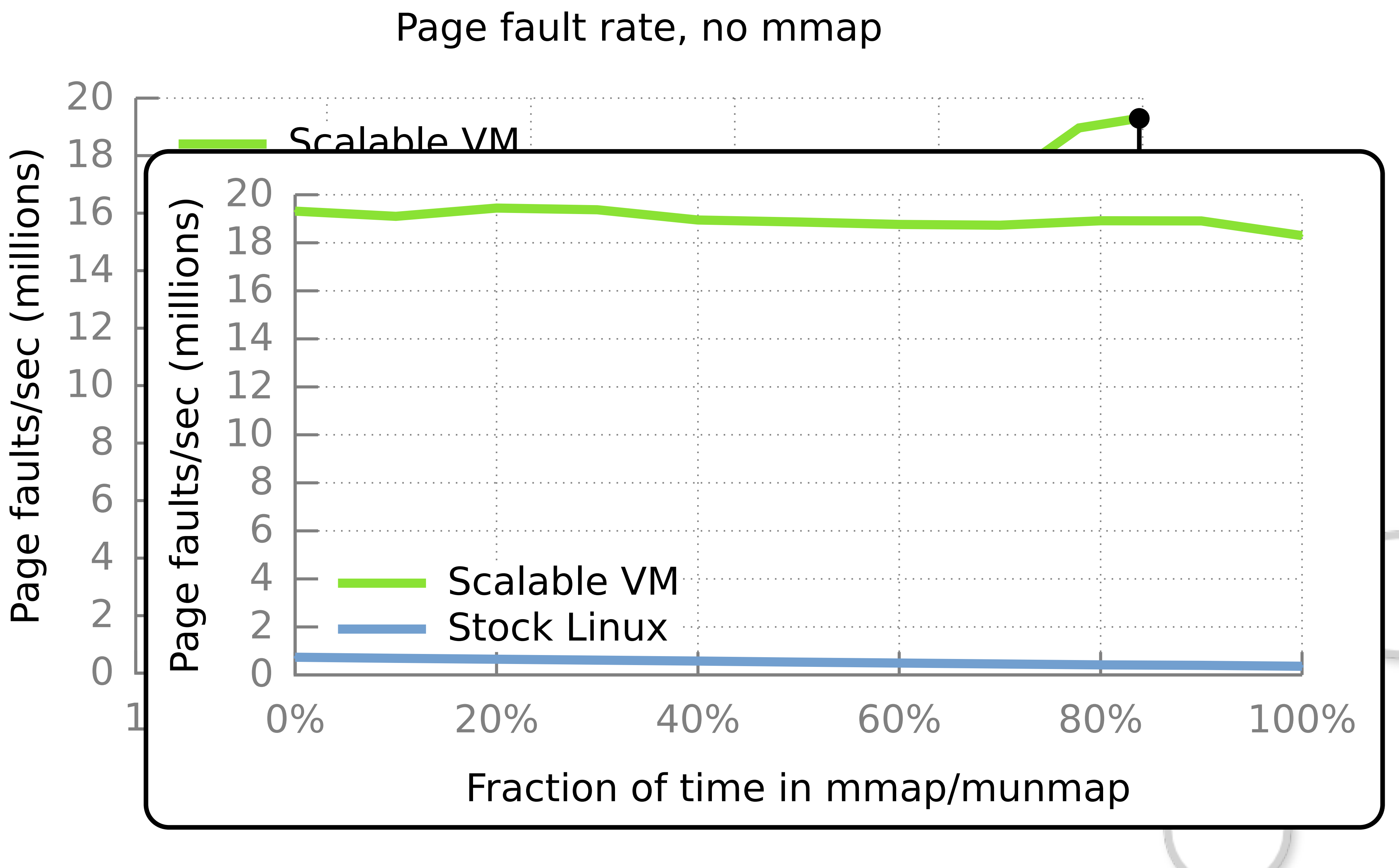
Page faults have constant cost



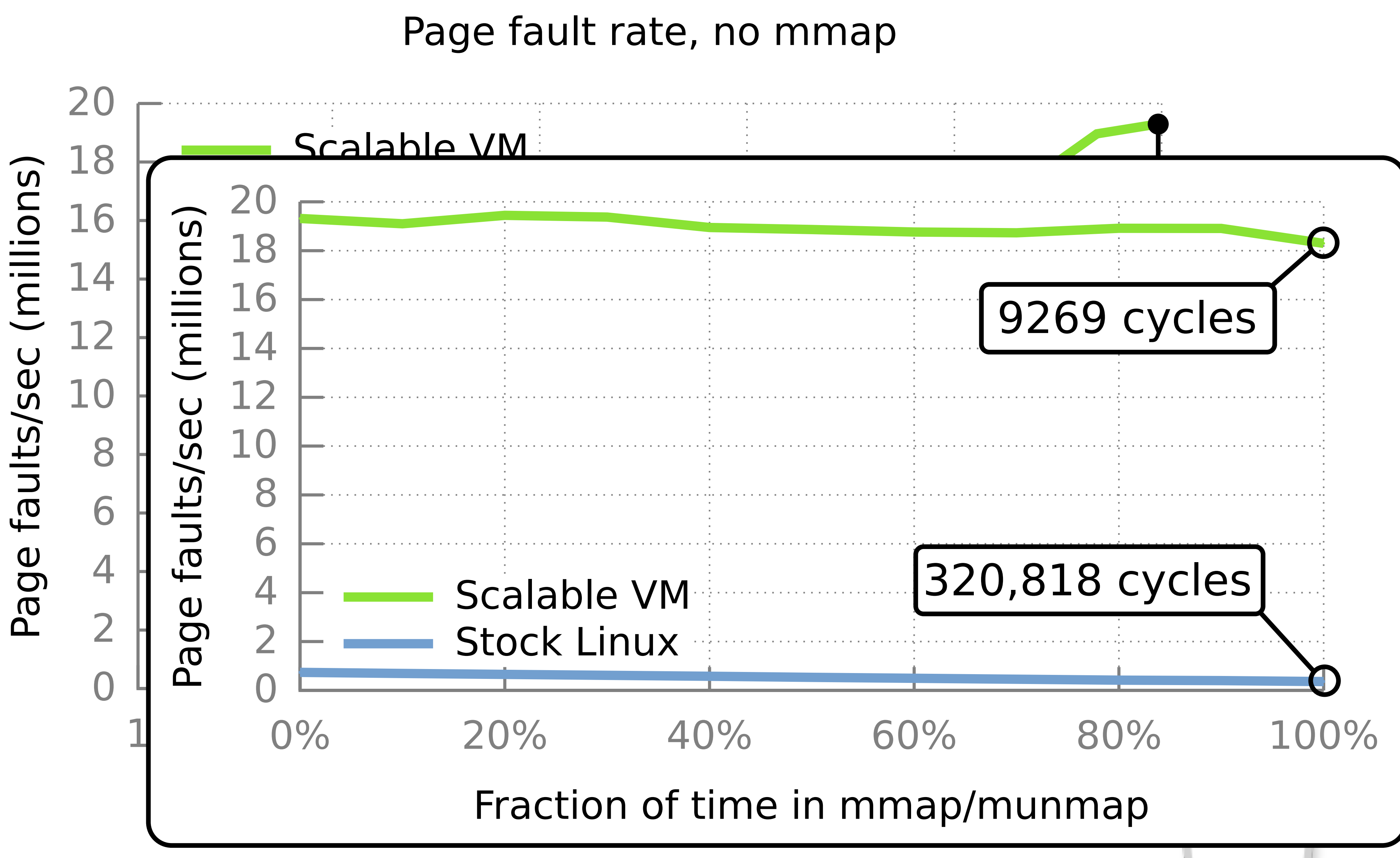
Page faults have constant cost



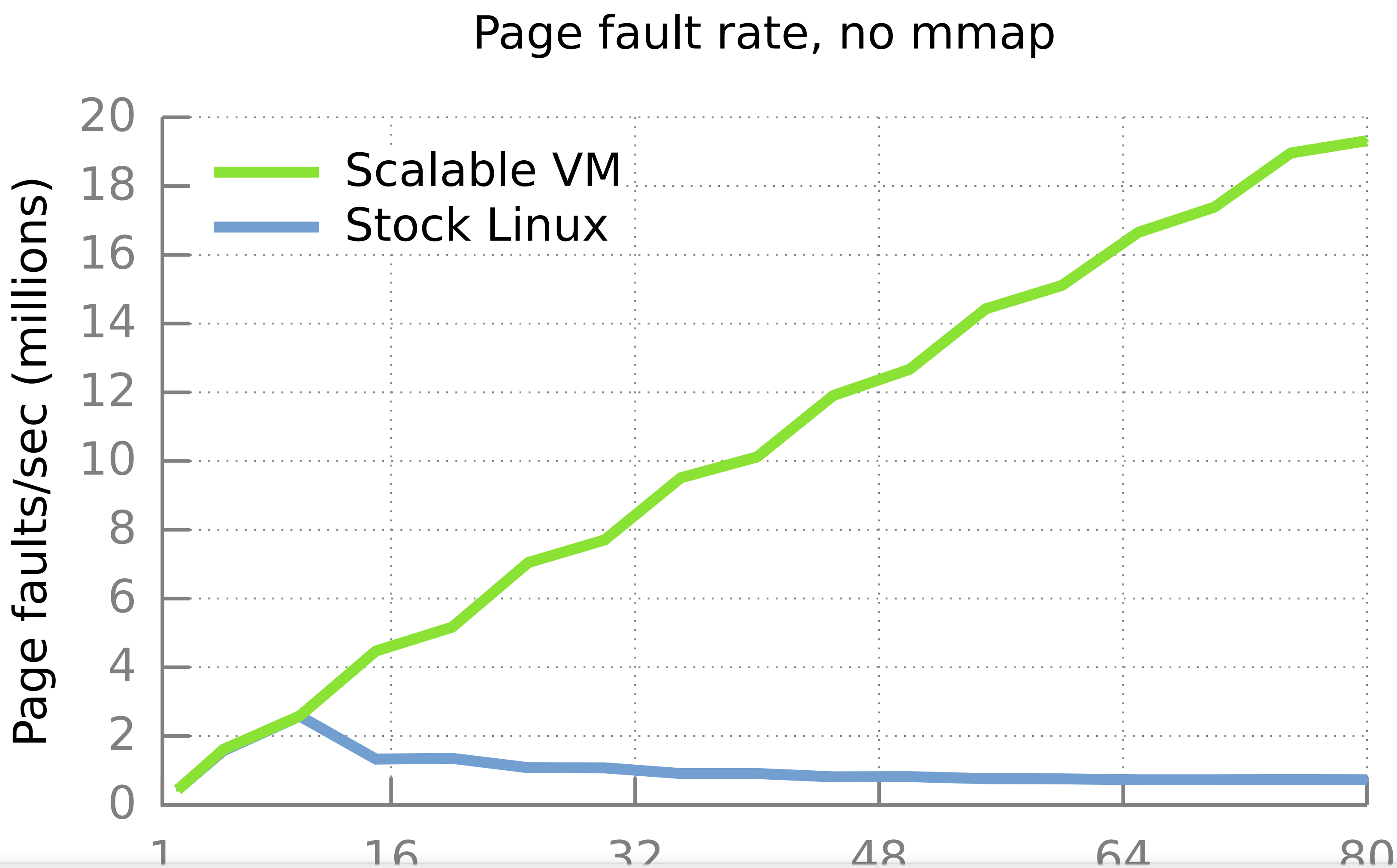
Page faults have constant cost



Page faults have constant cost



Page faults have constant cost



Page faults can scale at constant cost to very high rates and regardless of memory mapping load.

Related work

RCU [McKenney '98, '01, '04, '06]

Data structures
- RCU [Fraser '03, '07, Howard '10]
- Concurrent [Lehman '81, Pugh '90]
- Functional [Nievergelt '72, Adams '92]

Multicore benchmarks [Boyd-Wickizer '10, Bienia '11]

Conclusion

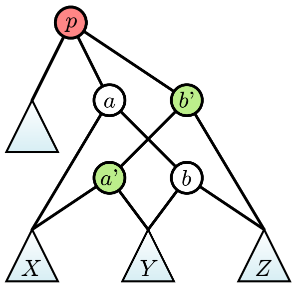
22x → 76x

Address space scalability matters

Conclusion

22x → 76x

Address space scalability matters

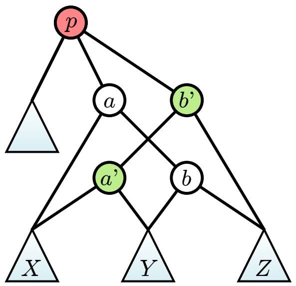


With new data structures like Bonsai,
page faults can shed coarse-gained locks

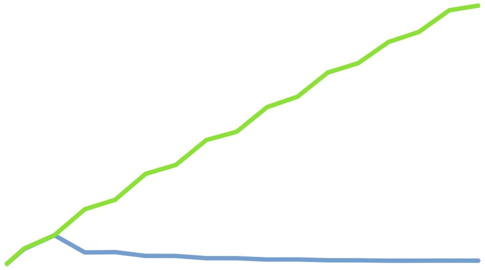
Conclusion

22x → 76x

Address space scalability matters



With new data structures like Bonsai,
page faults can shed coarse-gained locks

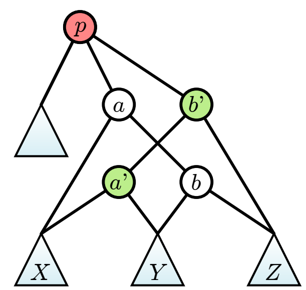


Page faults can scale at constant cost

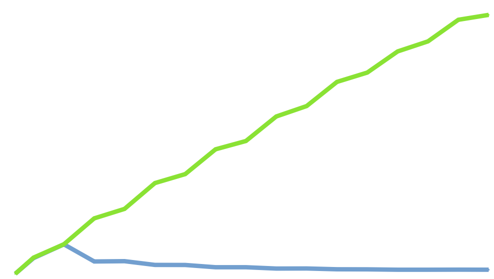
Conclusion

22x → 76x

Address space scalability matters



With new data structures like Bonsai,
page faults can shed coarse-gained locks



Page faults can scale at constant cost

Thank you