# Scalable Address Spaces using RCU Balanced Trees

## RCU : Read-Copy-Update

RCU is a synchronization mechanism that is optimized for read-mostly situations.

The basic idea behind RCU is to split updates into "removal" and "reclamation" phases.

Removal - Delete references to data items.

Reclamation - Freeing the removed data item.

Works because a pointer update is atomic.

a. Remove pointers to a data structure, so that subsequent readers cannot gain a reference to it.

b. Wait for all previous readers to complete their RCU read-side critical sections.

c. At this point, there cannot be any readers who hold references to the data structure, so it now may safely be reclaimed.

## RCU Core API

a. rcu_read_lock()
b. rcu_read_unlock()
c. synchronize_rcu() / call_rcu()
d. rcu_assign_pointer()
e. rcu_dereference()

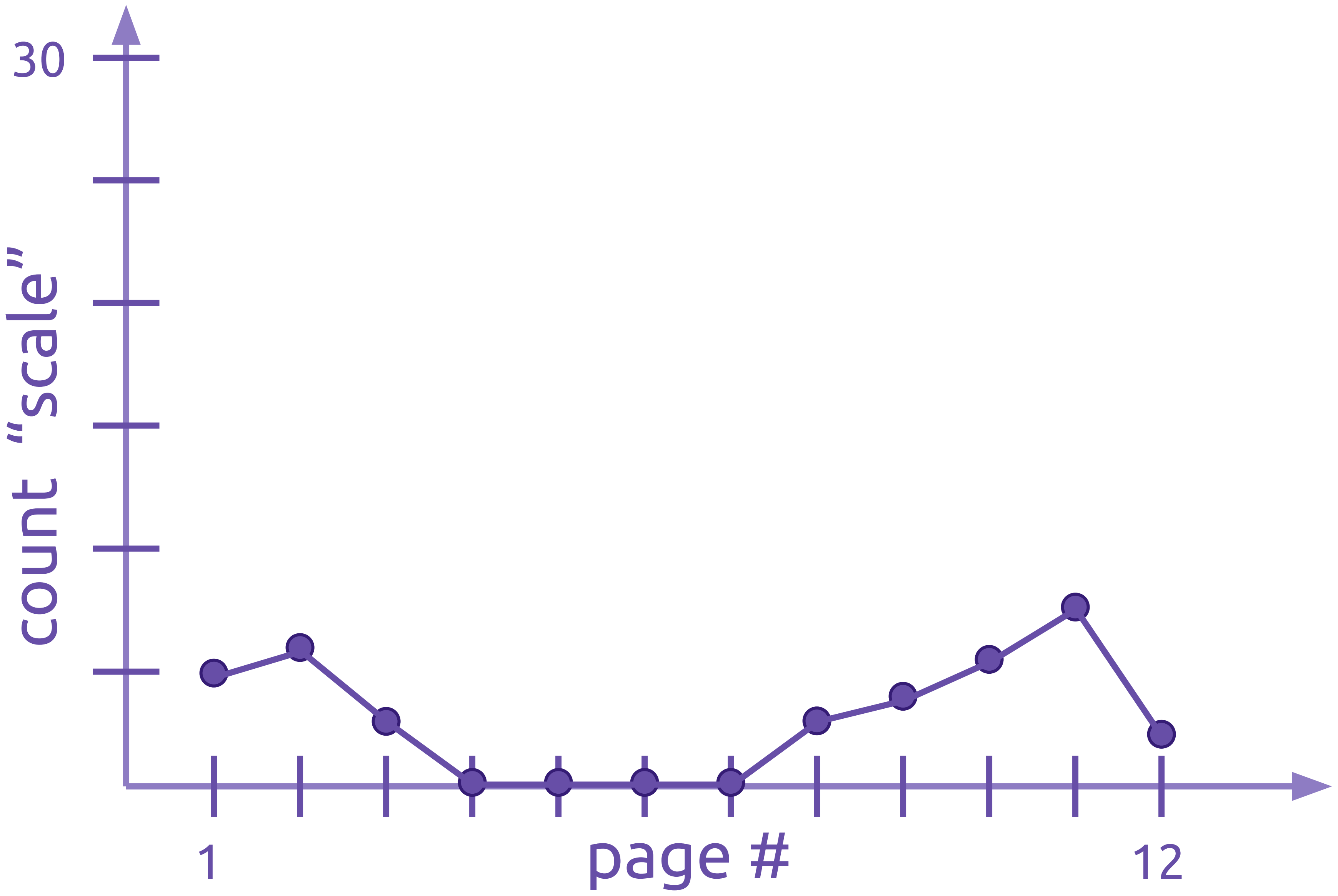
# RCU Example

```
struct foo {
        int a;
        char b;
        long c;
};
struct foo *gbl_foo;
DEFINE_SPINLOCK(foo_mutex);

int foo_get_a(void)
{
        int retval;

        rcu_read_lock();
        retval = rcu_dereference(gbl_foo)->a;
        rcu_read_unlock();
        return retval;
}
```

```
void foo_update_a(int new_a)
{
    struct foo *new_fp;
    struct foo *old_fp;

  new_fp = kmalloc(sizeof(*new_fp), … );
  spin_lock(&foo_mutex);
  old_fp = gbl_foo;
  *new_fp = *old_fp;
  new_fp->a = new_a;
  rcu_assign_pointer(gbl_foo, new_fp);
  spin_unlock(&foo_mutex);
  synchronize_rcu();
  kfree(old_fp);
}
```

## Discussion

# Brief meta-analysis



## Isn’t this just COW?

How does it compare to MVCC?

Are there cases in which one is better than the others?

What are the trade-offs?

## Optimizing for writes

RCU makes the read case fast

What about use cases where we do a lot of writes?

# User space RCU

## Use cases?

`memcached`

## Tradeoffs?

## Exokernel designs

How would exokernel solve VM scalability?

Can we do it with lower complexity than with RCU?

# Generalizing

What are typical application workloads?

How common is driving VM this hard?

## Other use-cases

What other parts of the Linux Kernel can take advantage of this?

# Functional vs Scalable

General principle?

- Functional DS -> C implementation

## Commutativity

Can we understand the VM interface using the scalable commutativity rule?