

# Microkernel Construction

## Capabilities

Nils Asmussen

05/07/2020

- Introduction
  - Global Names
  - ACL
- Capabilities in General
- Capabilities in NOVA

- How do you find/access resources?
- How do you restrict access to resources?

- One global namespace for (one type of) resources
- Example: semaphores, processes, devices, ... on UNIX

## Pros & Cons

- + Simple
- Name clashes: people need to agree on names.
- What if a malicious process registers a name first?
- All resources are visible: just try to access them

- Attach a list of permissions (subjects) to each object
- Permission depends on who you are, not what you have

## Pros & Cons

- + No need to give permissions explicitly
- + Makes it easy to restrict access to specific objects
- Makes it hard to restrict specific subjects
- POLA is more difficult to achieve
- Requires (global) names
- Confused deputy problem

# Confused Deputy Problem

- Compiler service: `compile <source> <object>`
- Service stores billing information in file “bill”
- Client executes: `compile foo bill`
- Service has access to bill file, client does not
- Problem: service is acting on behalf of the client, but opens files with *its own* permissions
- One solution: the client opens files and passes file descriptors (capabilities) to service

- Introduction
- Capabilities in General
  - Overview
  - Operations
- Capabilities in NOVA

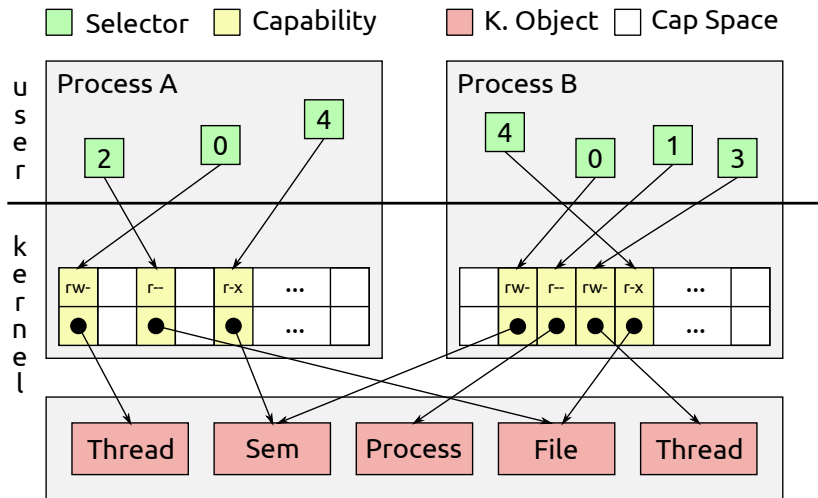
- Give each subject a local namespace
- Operations to exchange objects between namespaces
- Permission depends on what you have

## Pros & Cons

- + Makes it easy to restrict specific subjects
- + Separation of subsystems, composable, independent
- + POLA is easy to achieve
- Need to give permissions explicitly
- Exchanging, especially revoking, capabilities is difficult

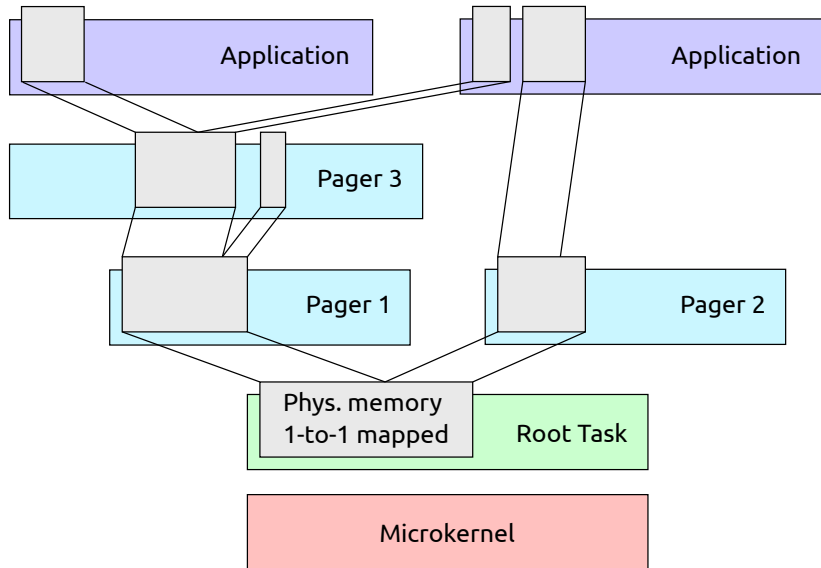


# Overview



- Map/delegate:
  - Copy capability from one Cap Space to the other
- Grant:
  - Move capability from one Cap Space to the other
- Revoke:
  - Remove capability, recursively
- Lookup:
  - Search capability by selector and return its permissions
- Translate:
  - Translate selector from one Cap Space to the other

# Hierarchical Organization



- Introduction
- Capabilities in General
- Capabilities in NOVA
  - Capability Spaces
  - Mapping Database
  - Delegate, Translate and Revoke
  - Data Types
  - Receive Windows

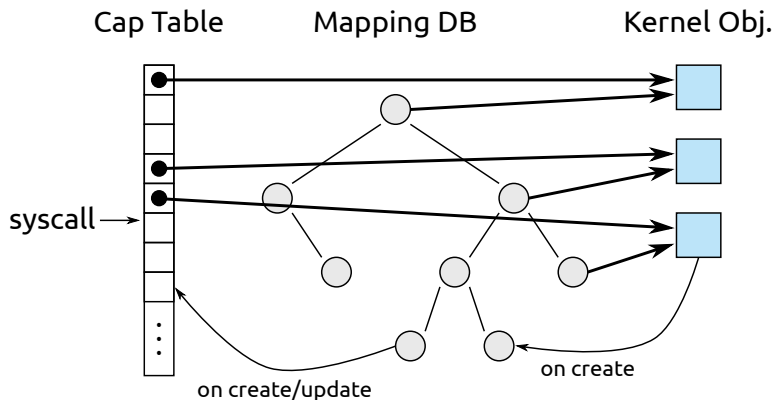
## Each protection domain (Pd) has

- `Space_obj`: object capabilities
- `Space_mem`: memory capabilities (pages)
- `Space_pio`: I/O port capabilities

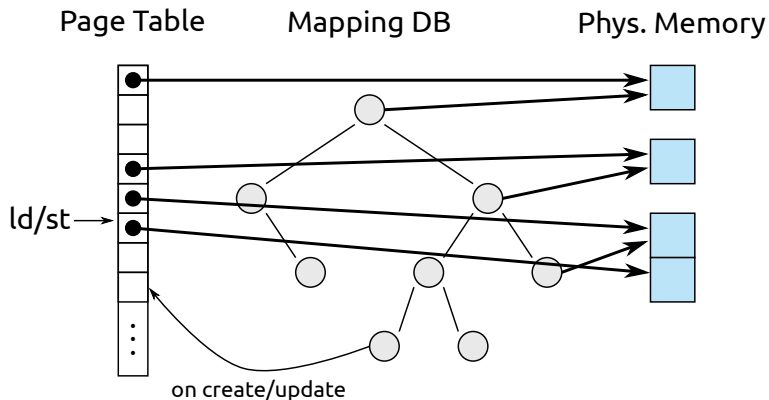
## Similarities and differences

- Shared: capability delegation, revocation, ...
- Differences:
  - Object caps are created and used via system calls
  - Port and memory caps are referring to existing resources
  - Passed to root task, distributed in the system via delegation
  - Memory capabilities lead to page table entries
  - Port capabilities lead to bits set in the I/O bitmap

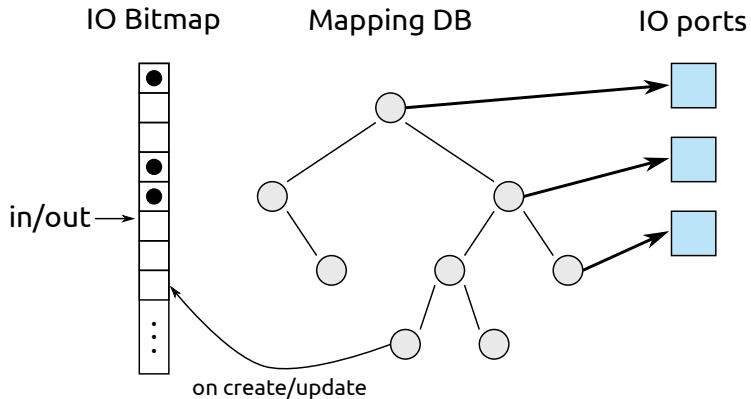
# Object Capability Space



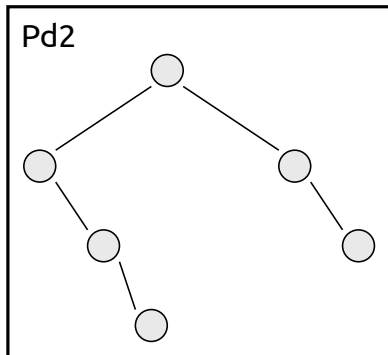
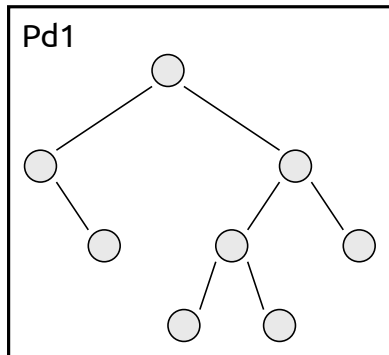
# Memory Capability Space

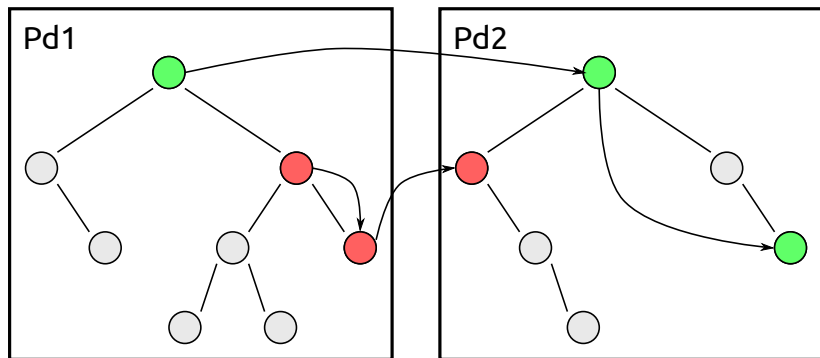


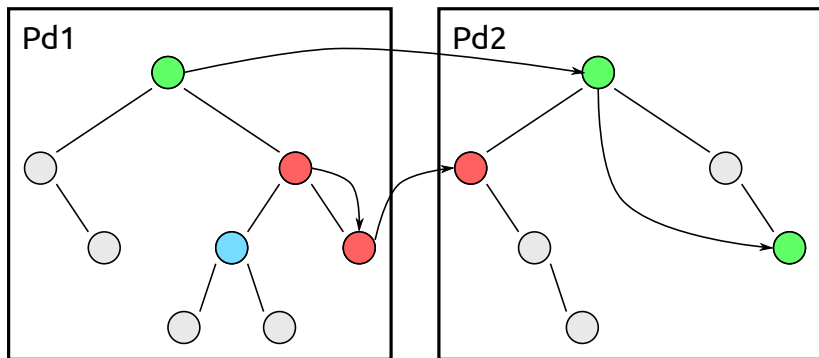
# I/O Capability Space

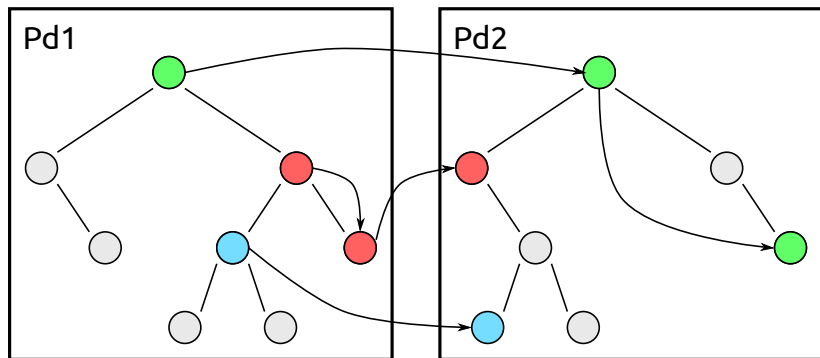




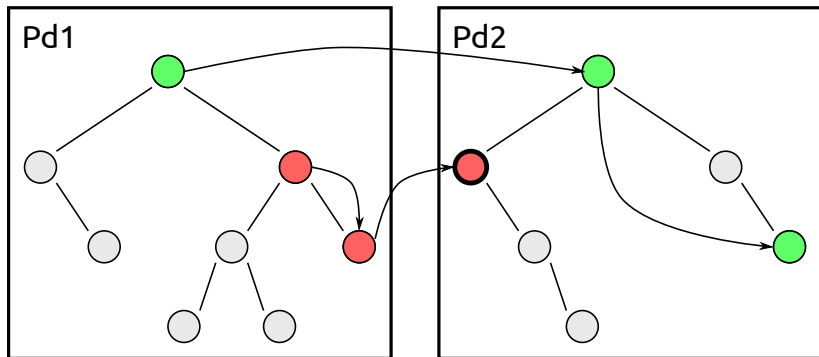




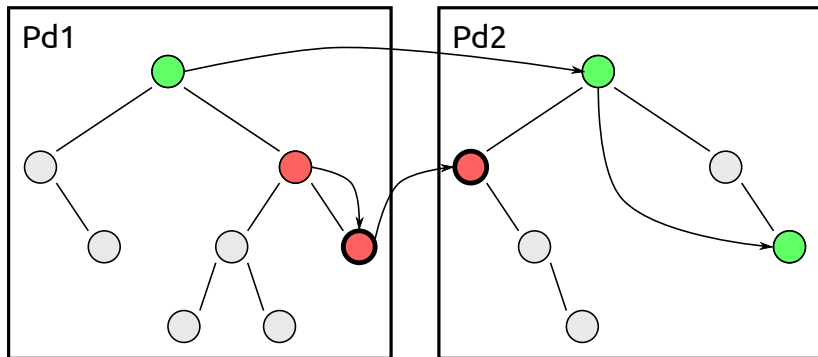




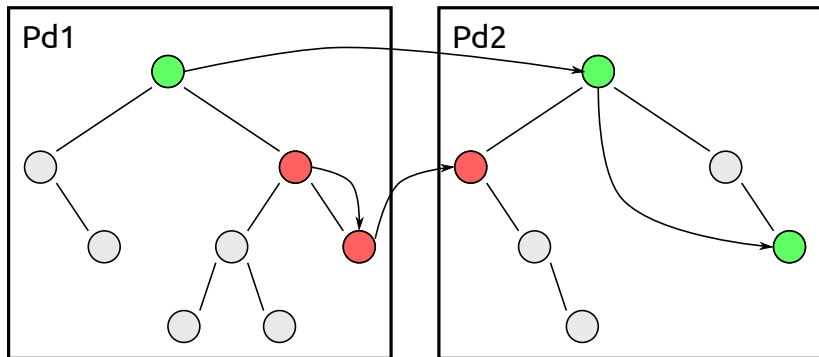
# Mapping Database – Translate



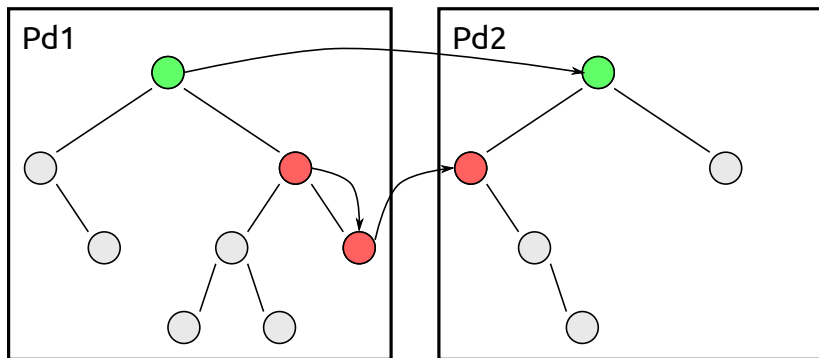
# Mapping Database – Translate



# Mapping Database – Revoke

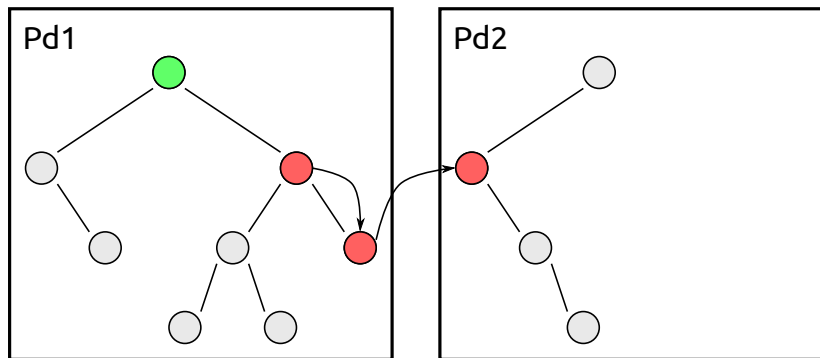


# Mapping Database – Revoke

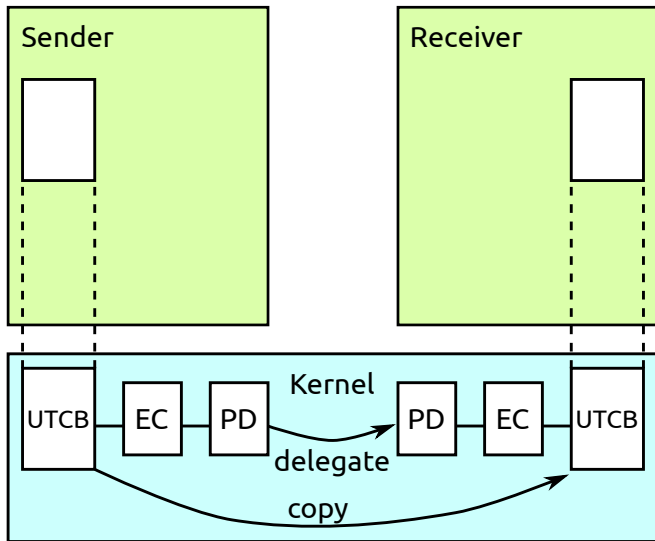




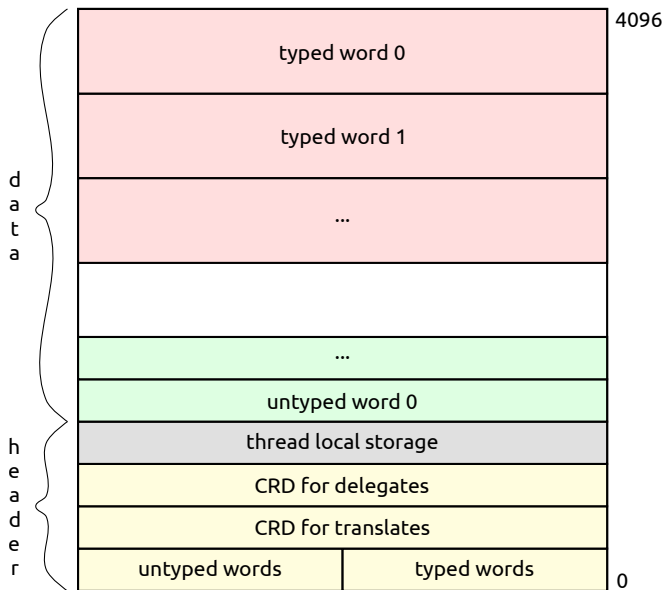
# Mapping Database – Revoke



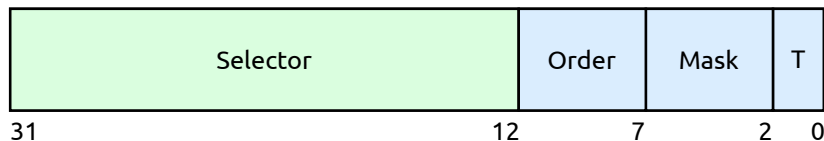
# IPC with Delegate



# UTCB Layout



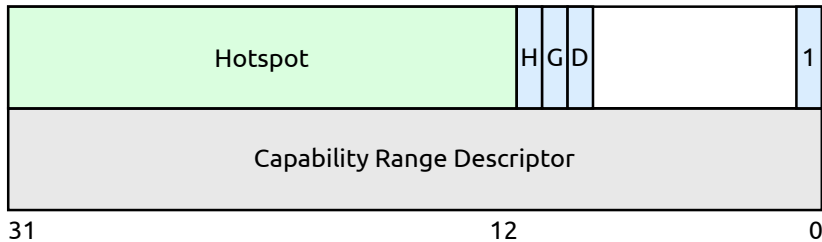
# Capability Range Descriptor



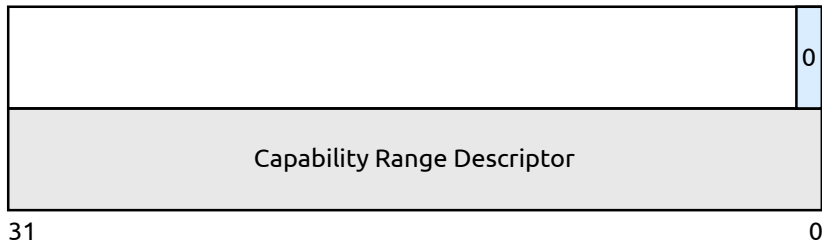
- Order specifies the number of capabilities ( $2^{order}$ )
- Selector specifies the first capability
- Selector has to be size aligned, i.e., a multiple of  $2^{order}$
- **wrong**: order=2, selector=6, okay: order=2, selector=8
- Mask allows to reduce permissions
- T specifies capability space (objects, memory, I/O)

# Typed Words

Delegate:



Translate:



- Receiver sets up receive window (writes CRD into UTCB)
- Receiver waits for IPC
- Sender puts typed item into UTCB
- Sender calls portal
- Kernel delegates typed item
- Kernel puts typed item into UTCB, telling receiver about caps
- Kernel switches to receiver
- But: what if receive window and sent caps don't match?

# Matching Send and Receive Window (1)

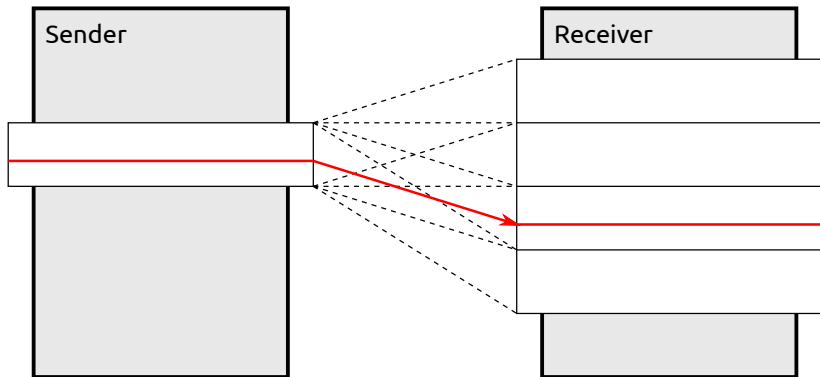


Figure: Send window is smaller than receive window

## Matching Send and Receive Window (2)

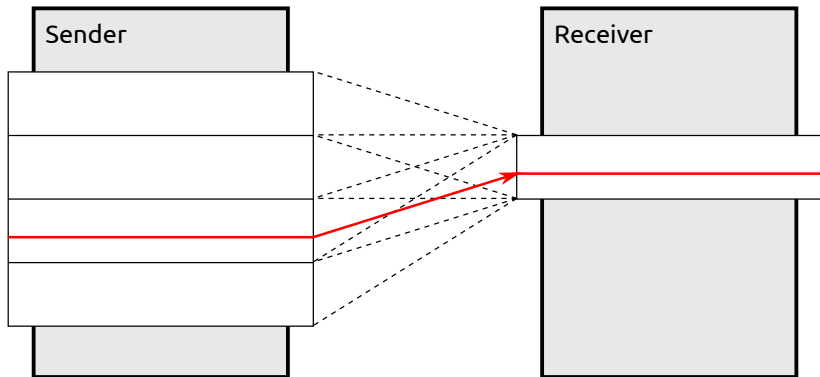


Figure: Send window is larger than receive window



## Delegation Code (1)

```
void Pd::xfer_items (Pd *src, Crd xlt, Crd del,
                    Xfer *s, Xfer *d, unsigned long ti)
{
    for (Crd crd; ti--; s--) {
        crd = *s;
        switch (s->flags() & 1) {
            case 0:
                xlt_crd (src, xlt, crd);
                break;
            case 1:
                del_crd (src, del, crd, s->flags(), s->hotspot());
                break;
        }
        if (d)
            *d-- = Xfer (crd, s->flags());
    }
}
```

## Delegation Code (2)

```
void Pd::del_crd (Pd *pd, Crd del, Crd &crd,
                 mword sub, mword hot)
{
    mword a = crd.attr() & del.attr();
    mword sb = crd.base(), so = crd.order();
    mword rb = del.base(), ro = del.order(), o = 0;

    switch (del.type()) {
        case Crd::MEM:
            o = clamp (sb, rb, so, ro, hot);
            delegate<Space_mem>(pd, sb, rb, o, a, sub);
            break;
        ...
    }

    crd = Crd (del.type(), rb, o, a);
}
```

## Delegation Code (3)

```
template <typename S>
void Pd::delegate (Pd *snd, mword snd_base, mword rcv_base,
                  mword ord, mword attr, mword sub)
{
    Mdb *mdb;
    for (mword addr = snd_base;
         (mdb = snd->S::tree_lookup (addr, true));
         addr = mdb->node_base + (1UL << mdb->node_order)) {
        Mdb *node = new Mdb (static_cast<S *>(this), ...);

        if (!S::tree_insert (node))
            ...
        if (!node->insert_node (mdb, attr))
            ...

        S::update (node);
    }
}
```

- When revoking, kernel objects should be destructed
- But what if somebody accesses them at the same time?
- We could lock them during each access
- But this is expensive
- We don't care that much when exactly they are destructed
- Can't we destruct them if nobody accesses them anymore?

- Basically: copy-on-write with lazy delete
- Don't change objects, but copy them and change the copy
- Don't delete objects immediately, but when readers are done
- In case of NOVA: no copy-on-write, but only lazy delete
- On revoke, object is removed first
- Then, the object is registered for deletion
- Timer IRQ is used to delete only if all readers are gone

# RCU Grace Period

