barkhausen institut

Microkernel Construction

Capabilities

Nils Asmussen

06/04/2026

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

Motivation

1. How do you find/access resources?

2. How do you restrict access to resources?

Global Names

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

Global Names

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

Pros & Cons

🙂 Simple

Global Names

- 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

Access Control Lists

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

Access Control Lists

- 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

Access Control Lists

- 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

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 acts on behalf of client, but opens files with **its own** permissions

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 acts on behalf of client, but opens files with **its own** permissions

Solution

Client opens files and passes file descriptors (capabilities) to service

- Introduction
- **Capabilities in General**
  - **Overview**
  - Operations
  - Implementation Considerations and Variants
- Capabilities in NOVA

Capabilities

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

Capabilities

- 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

Capabilities

- 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

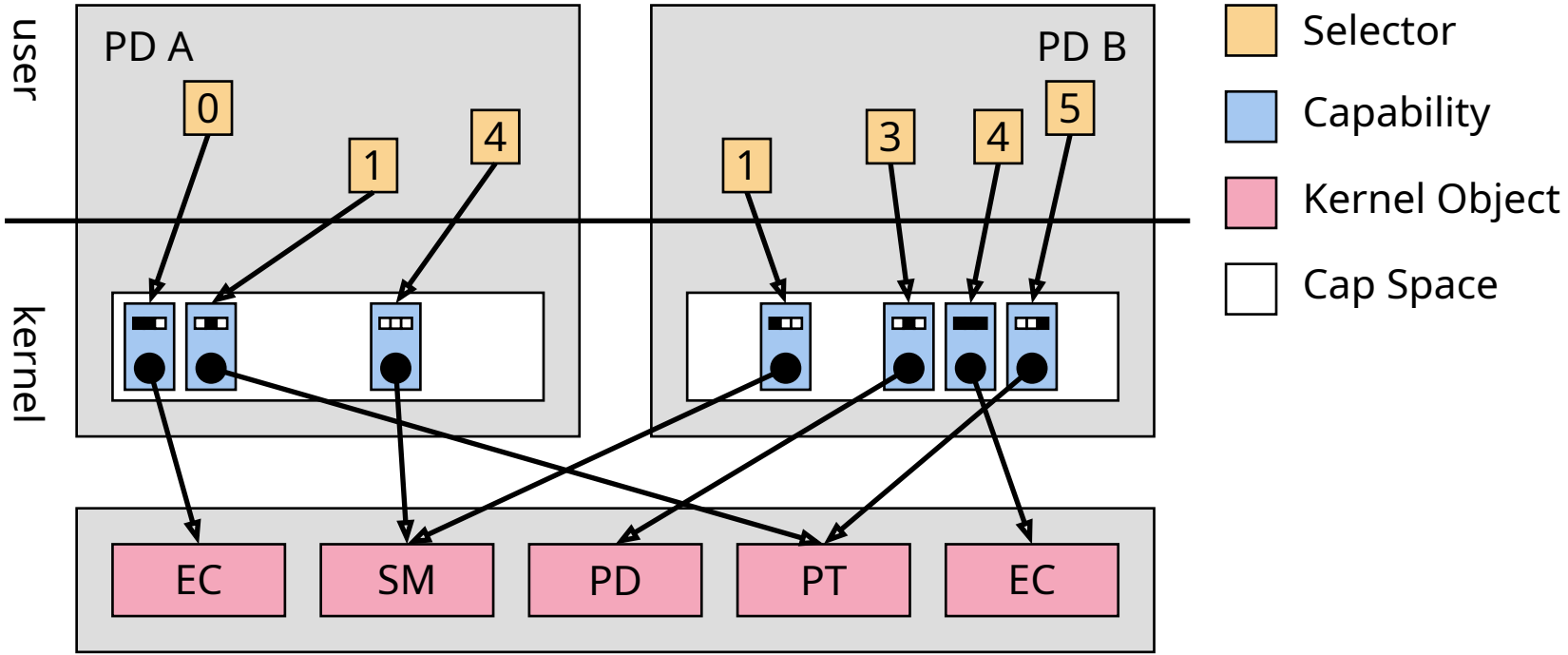
🙂 Separation of subsystems, composable, independent

🙂 POLA is easy to achieve

😕 Need to give permissions explicitly

😕 Exchanging, especially revoking, capabilities is difficult

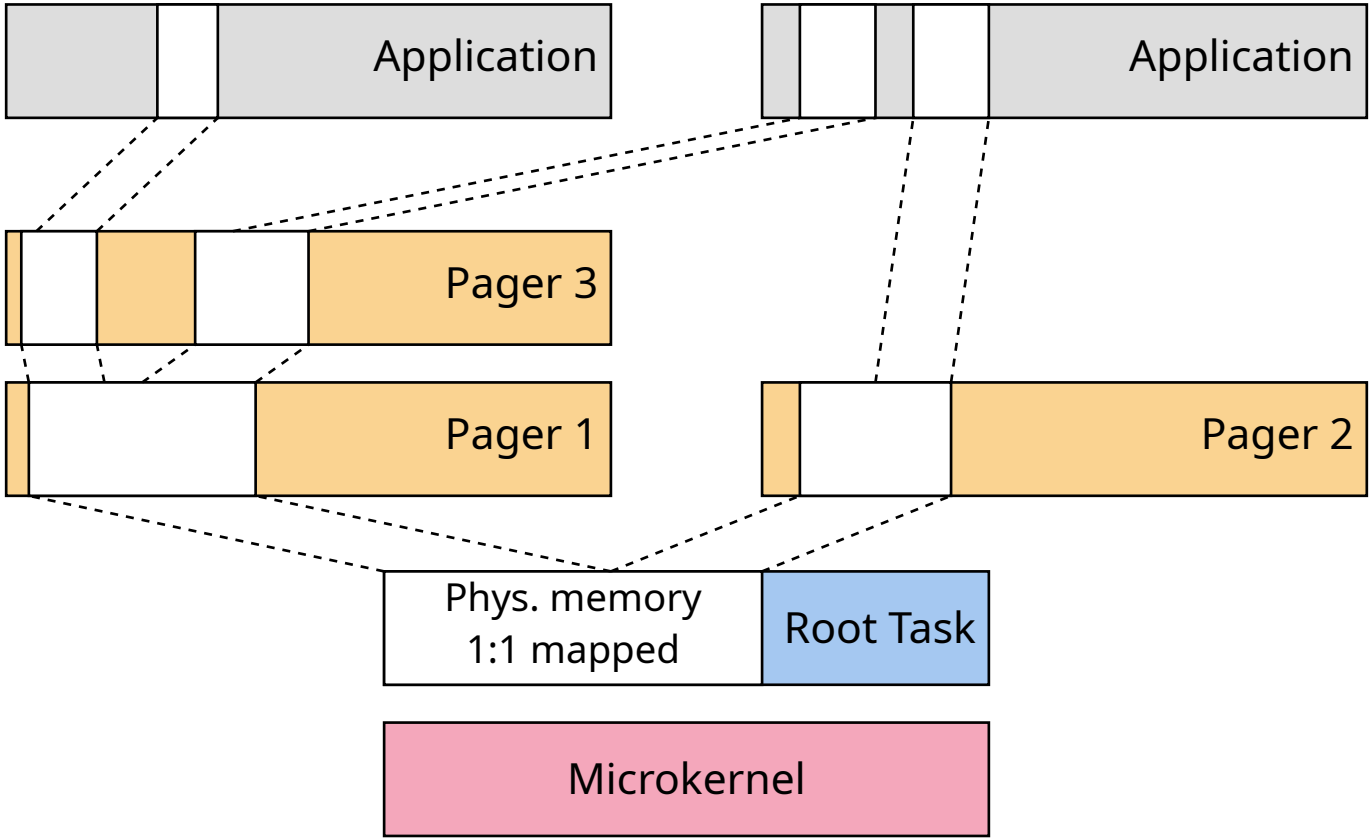
Capabilities Overview



Operations

- 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**
  - Overview
  - Operations
  - **Implementation Considerations and Variants**
- Capabilities in NOVA

Revocability

Fully Selective (examples: NOVA, M³ , memory caps in L4Re)

- Every capability can be revoked individually
- Child capabilities will be revoked recursively

Revocability

Fully Selective (examples: NOVA, M³ , memory caps in L4Re)

- Every capability can be revoked individually
- Child capabilities will be revoked recursively

None Selective (example: object caps in L4Re)

- Revoking any cap for a specific object revokes all caps for that object
- Can be combined with a per-cap delete-permission

Revocability

Fully Selective (examples: NOVA, M³ , memory caps in L4Re)

- Every capability can be revoked individually
- Child capabilities will be revoked recursively

None Selective (example: object caps in L4Re)

- Revoking any cap for a specific object revokes all caps for that object
- Can be combined with a per-cap delete-permission

Semi Selective (example: distributed capability system in FractOS)

- Like the previous, but revocable subtrees can be created
- Creates a new object that can be revoked separately

Requirements

- Finding capabilities by selector
  - Organize them in a tree with selector as key
  - Common shortcut: large table for object capabilities

Requirements

- Finding capabilities by selector
  - Organize them in a tree with selector as key
  - Common shortcut: large table for object capabilities
- Finding child capabilities (delegation, derivation)
  - Tree required for fully selective revocation
  - Without any further measures: unbounded recursion
  - Easier approach (not fully selective): list

Requirements

- Finding capabilities by selector
  - Organize them in a tree with selector as key
  - Common shortcut: large table for object capabilities
- Finding child capabilities (delegation, derivation)
  - Tree required for fully selective revocation
  - Without any further measures: unbounded recursion
  - Easier approach (not fully selective): list
- Delegation is performance critical
  - Challenging in distributed capability systems
  - FractOS does not track delegations, but invalidates objects at owner

Concurrency and Interuptibility

Concurrency

- Many threads could delegate/revoke capabilities simultaneously
- Kernel needs to ensure correctness
- Typical guaruantee: if revoke completes, no one has access anymore

Concurrency and Interuptibility

Concurrency

- Many threads could delegate/revoke capabilities simultaneously
- Kernel needs to ensure correctness
- Typical guaruantee: if revoke completes, no one has access anymore

Interruptibility

- What if a large capability tree is revoked?
- System should remain usable
- Ideally, revoke should be interruptible

Locking Strategies

NOVA

- locks only small code snippets
- global lock for list of delegations
- per-space lock for new nodes

Locking Strategies

NOVA

- locks only small code snippets
- global lock for list of delegations
- per-space lock for new nodes

L4Re

- locks entire revoke operation
- but lock is per object/frame

Capabilities in Other Systems

Files

- File descriptor is equivalent of selector
- File table is similar to capability space
- FDs can be passed between processes, but is seldomly used
- Revocation not possible
- Only used for a subset of kernel resources

POSIX Capabilities

- not associated with any object
- define whether a process has certain permissions
  (example: open port under 1024)
- main purpose: restrict power of root

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

Capability Spaces

Each protection domain (Pd) has

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

Capability Spaces

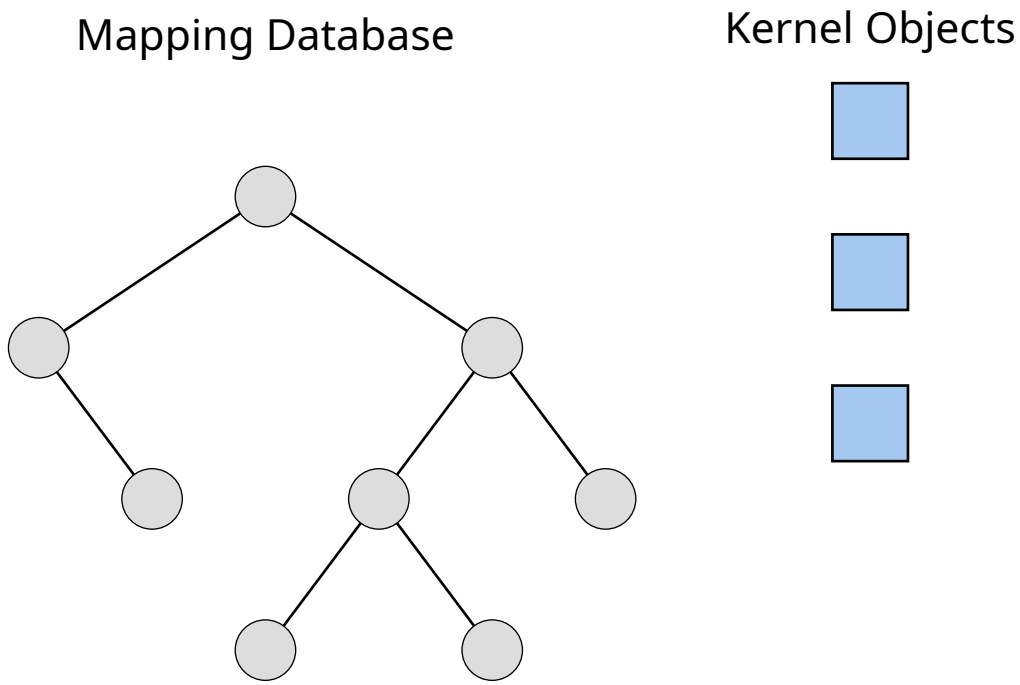
Each protection domain (Pd) has

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

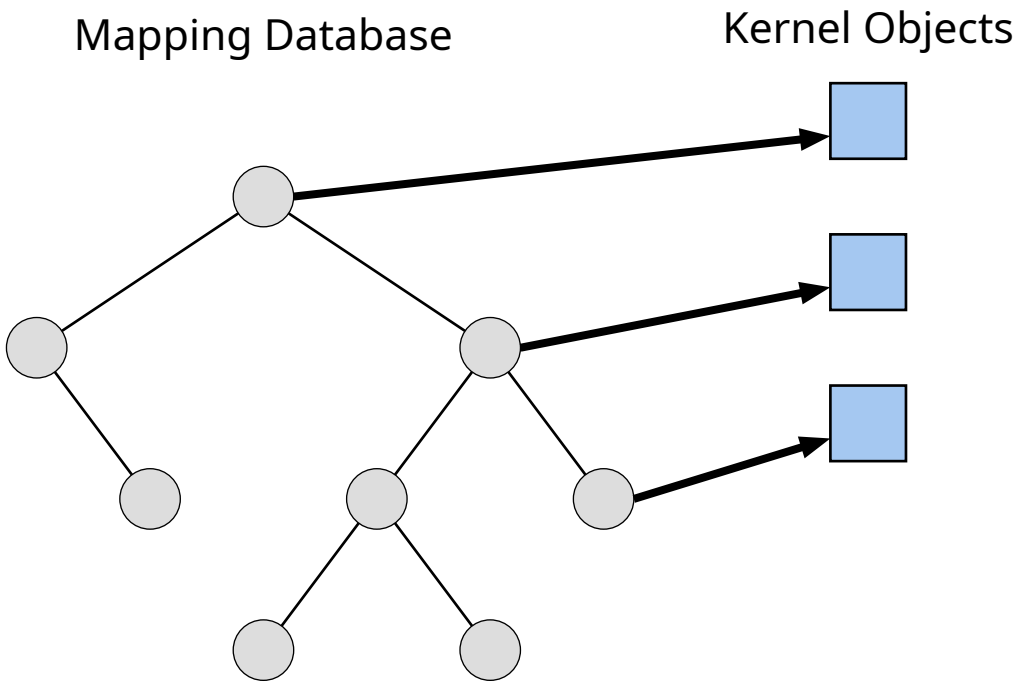
Similarities and differences

- 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

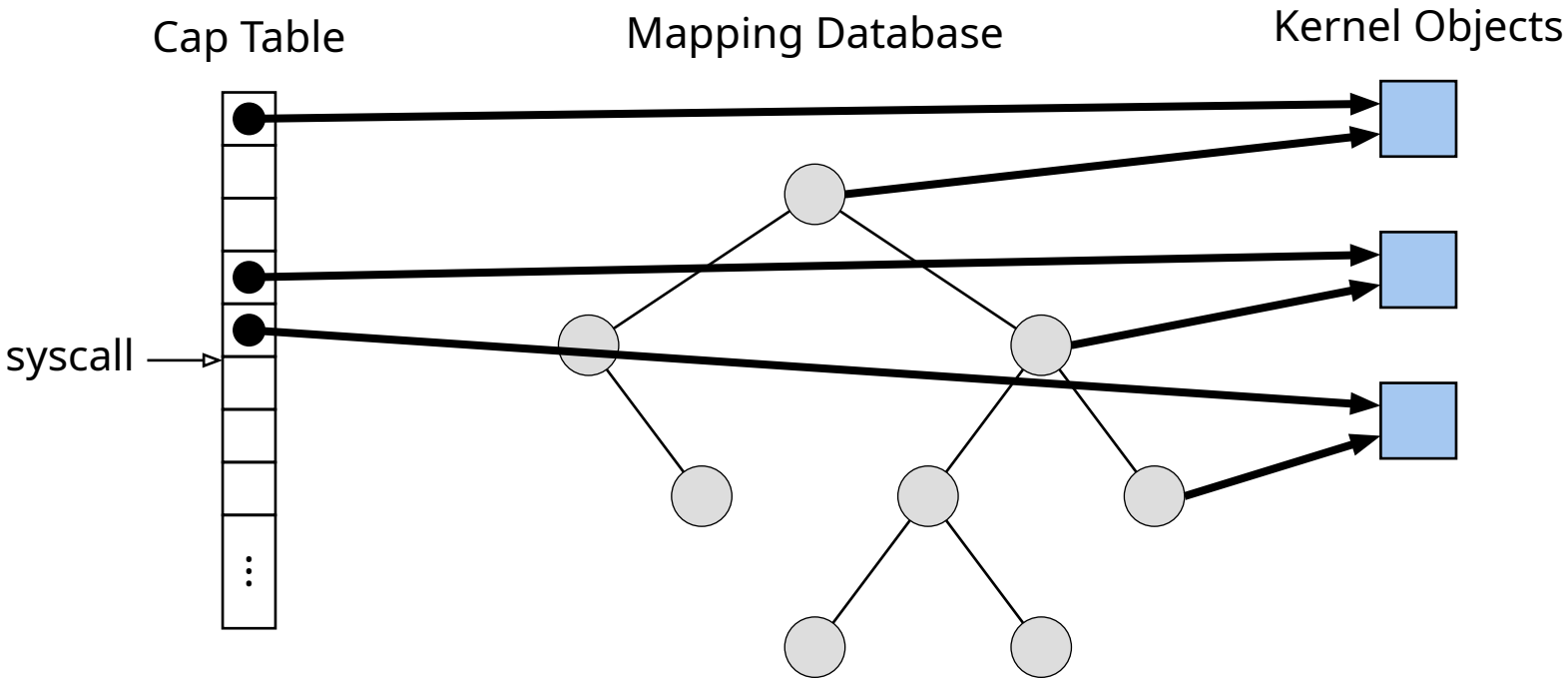
Object Capability Space



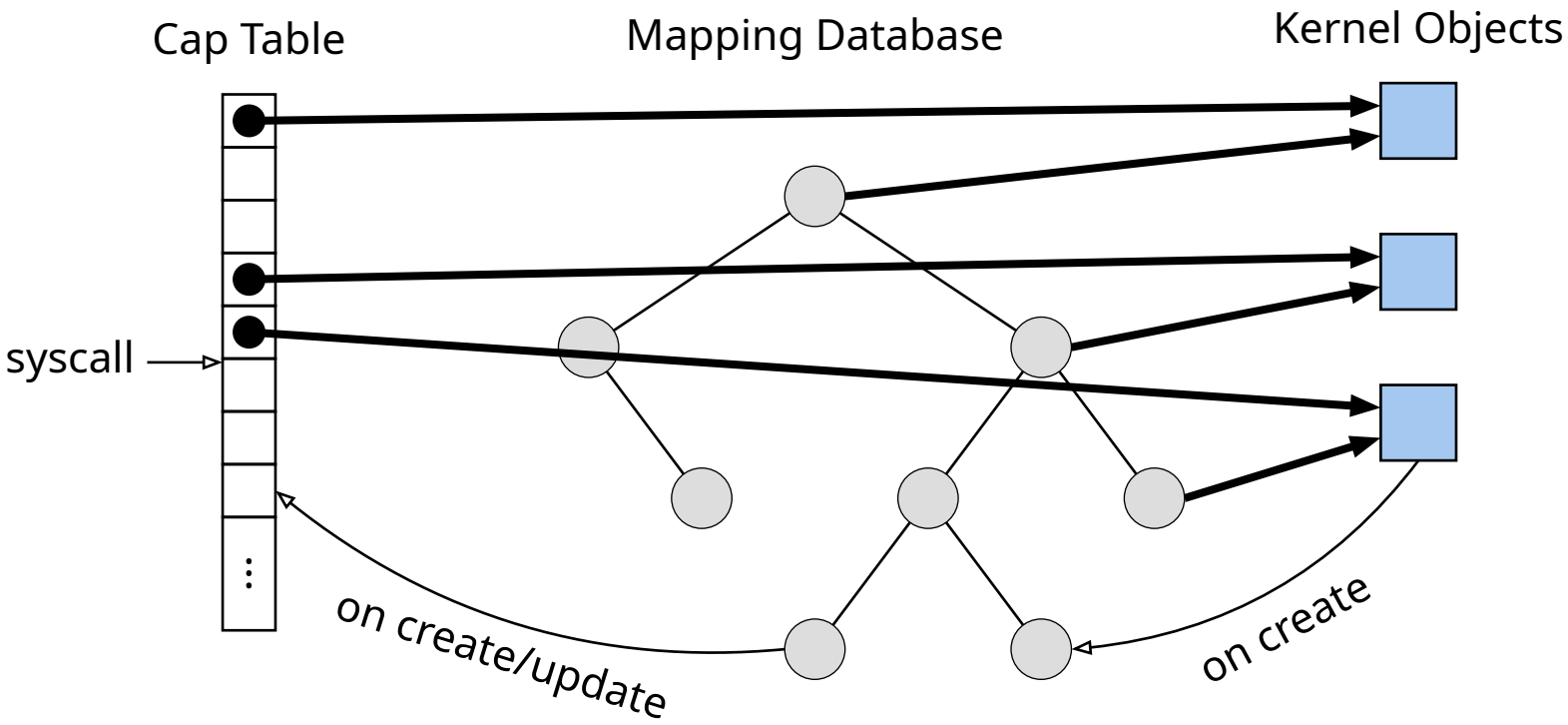
Object Capability Space



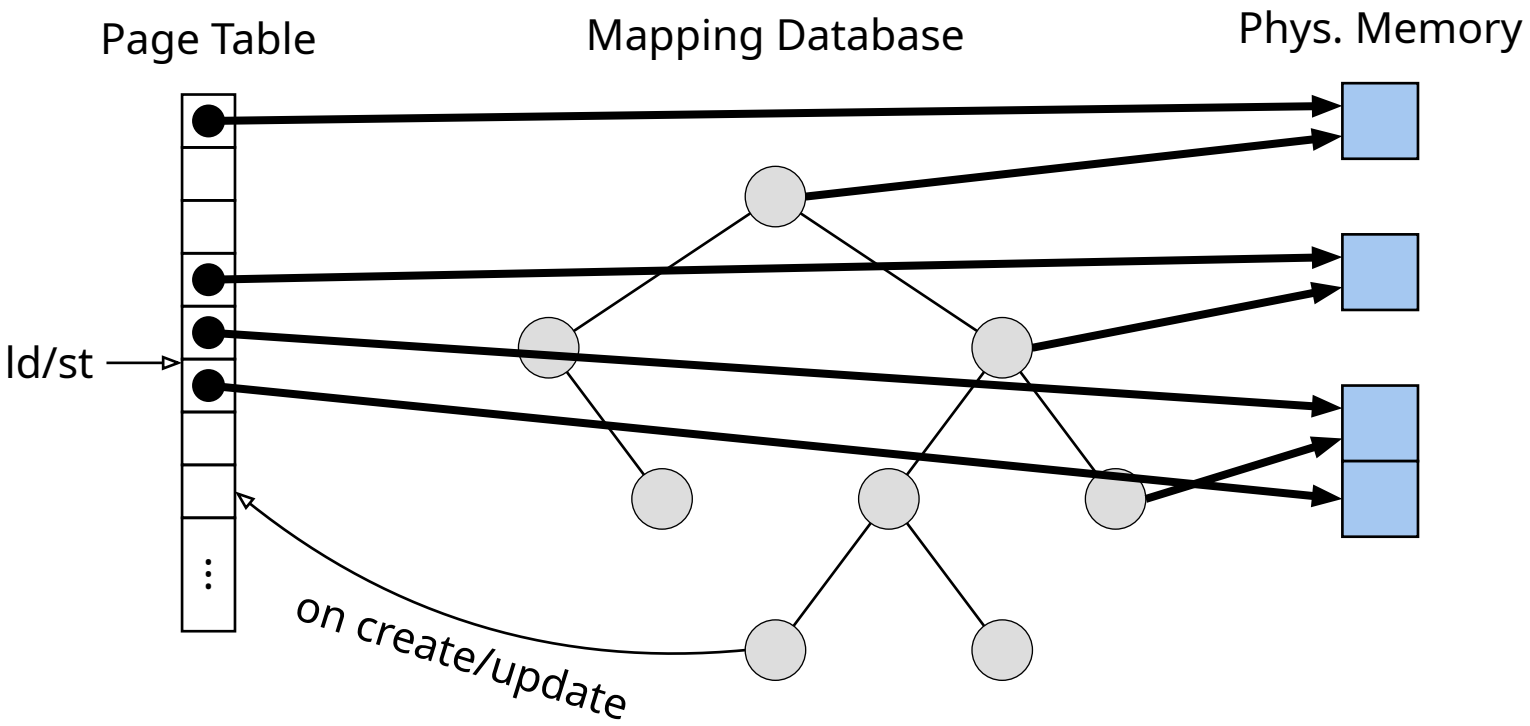
Object Capability Space



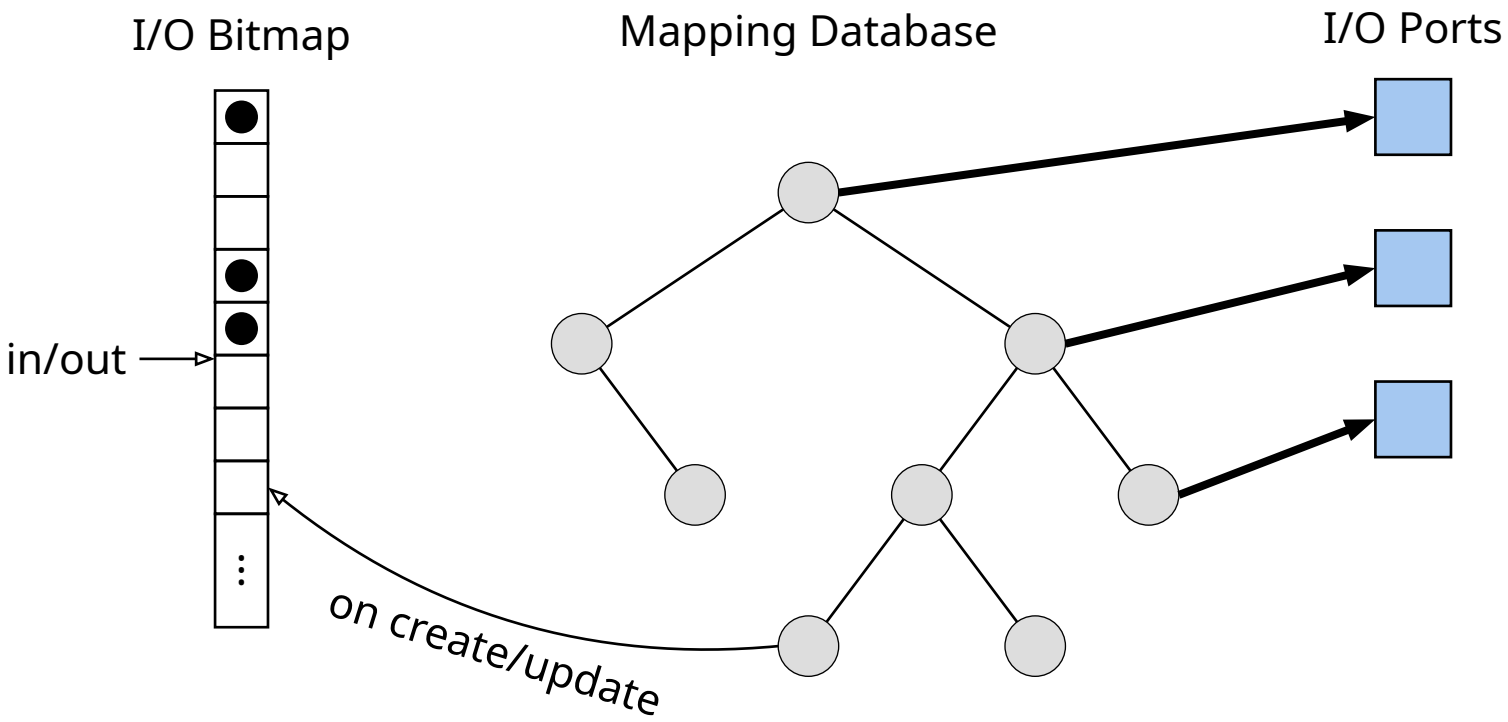
Object Capability Space



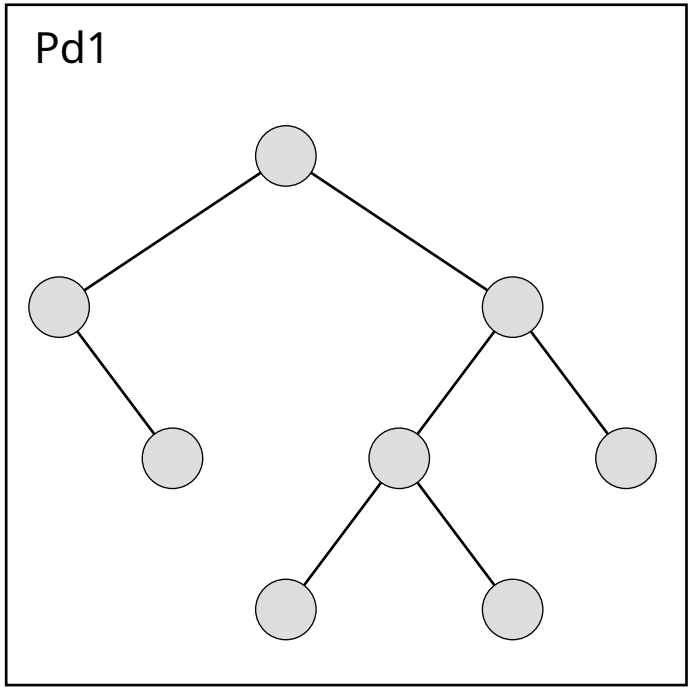
Mapping Capability Space

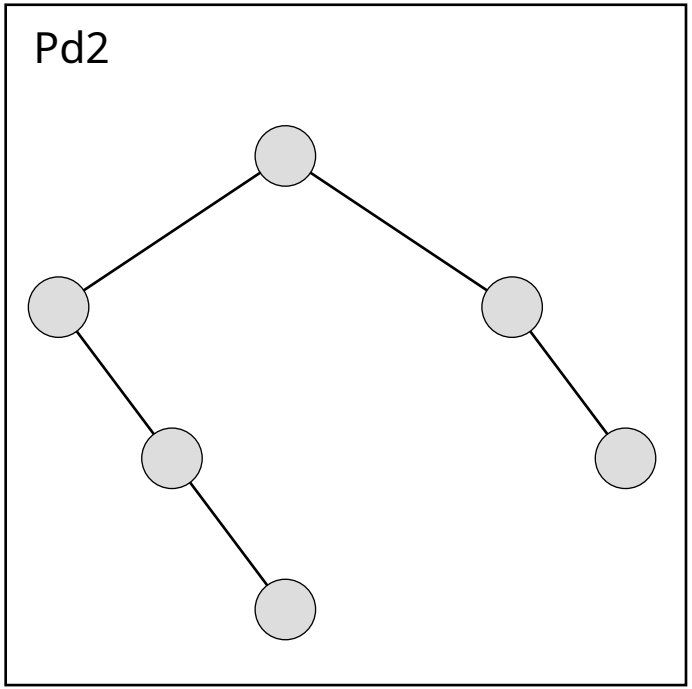


I/O Capability Space

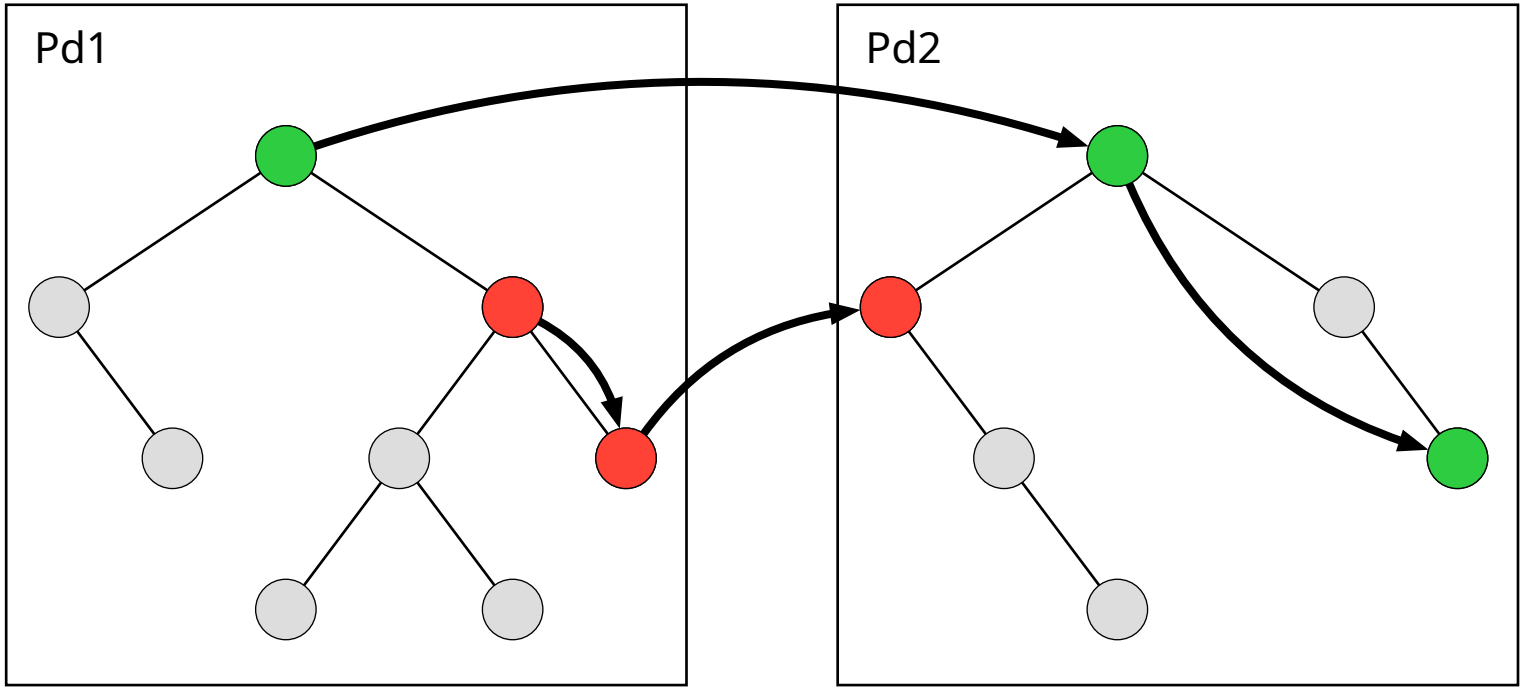


Mapping Database

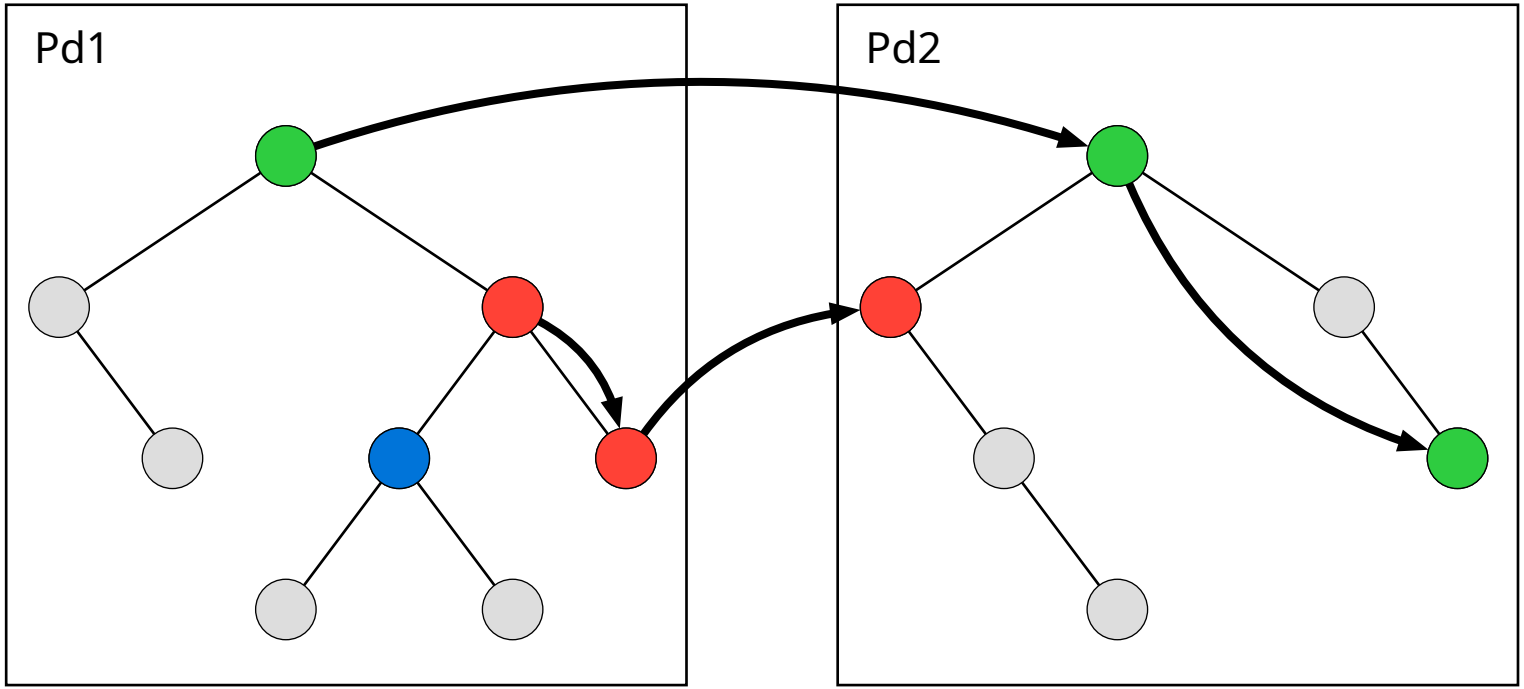




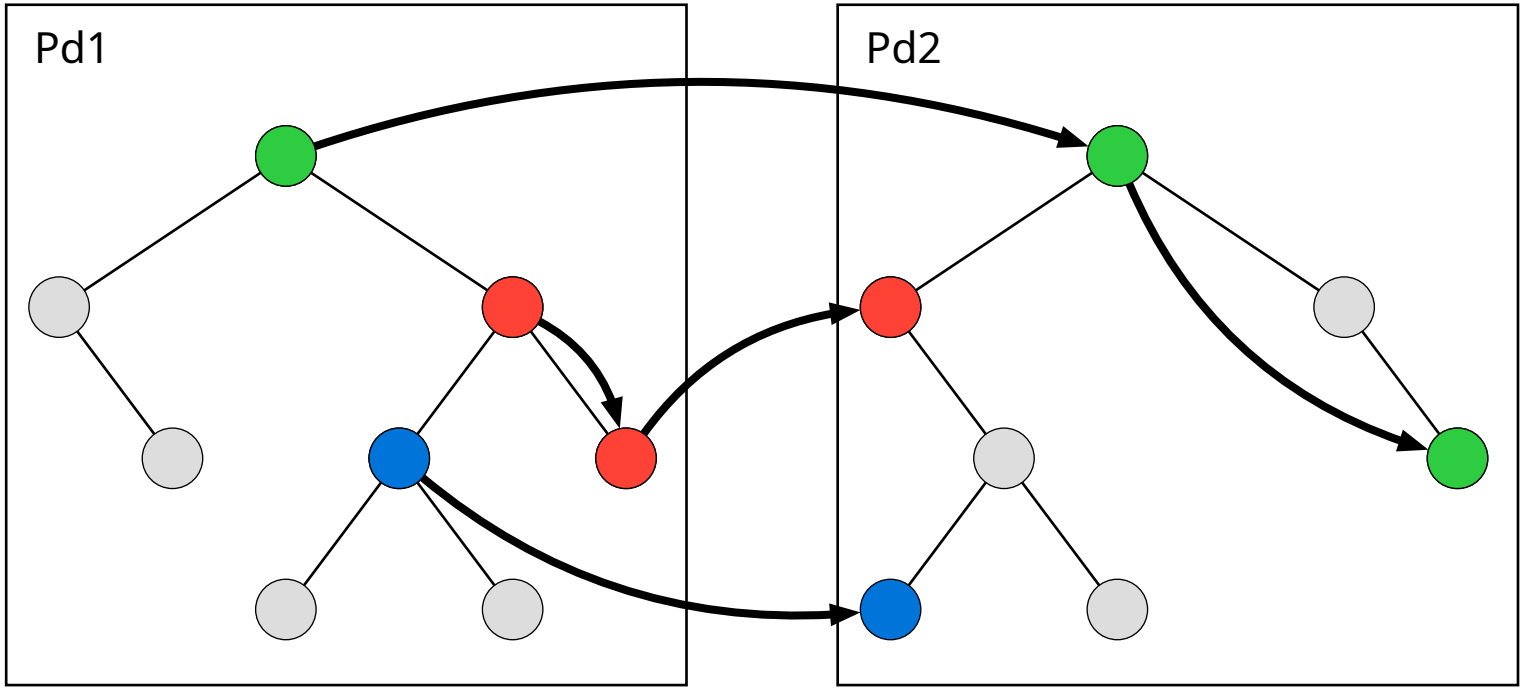
Mapping Database



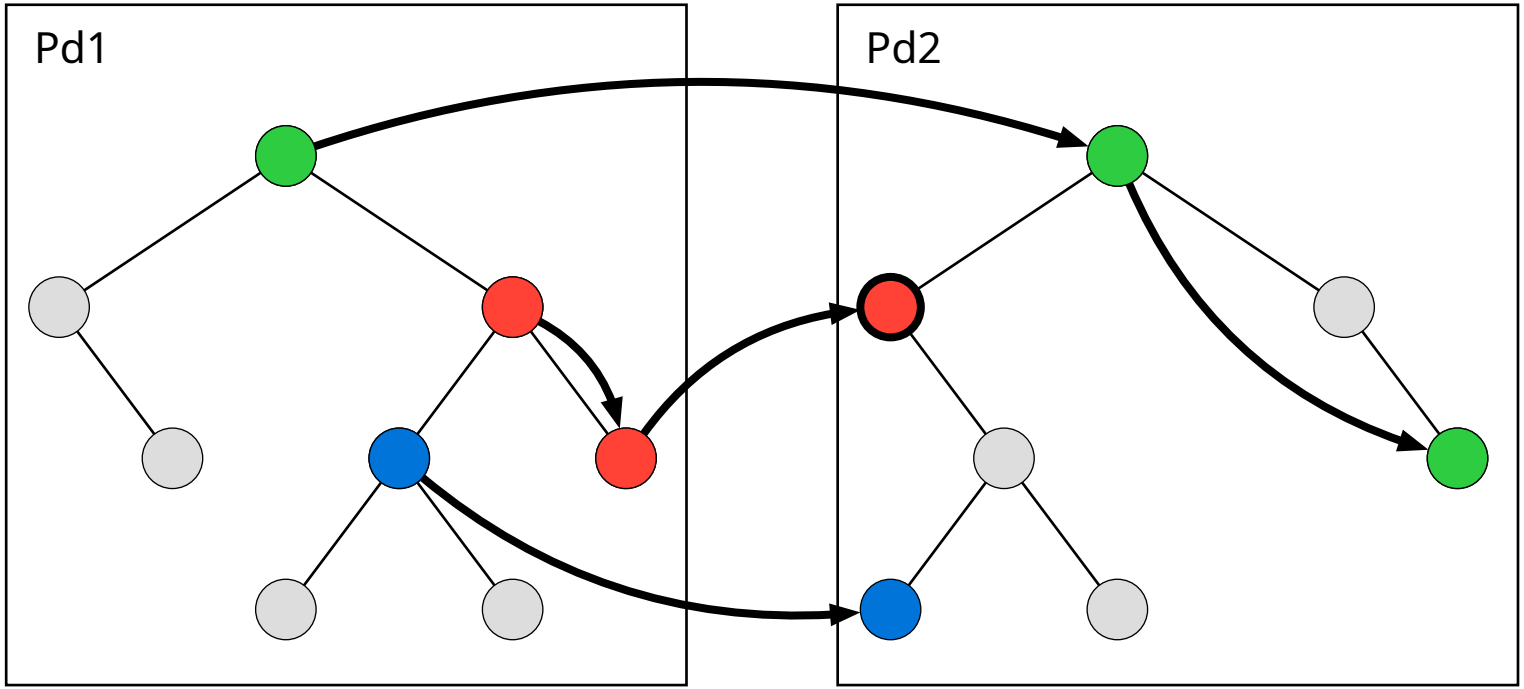
Mapping Database – Delegate



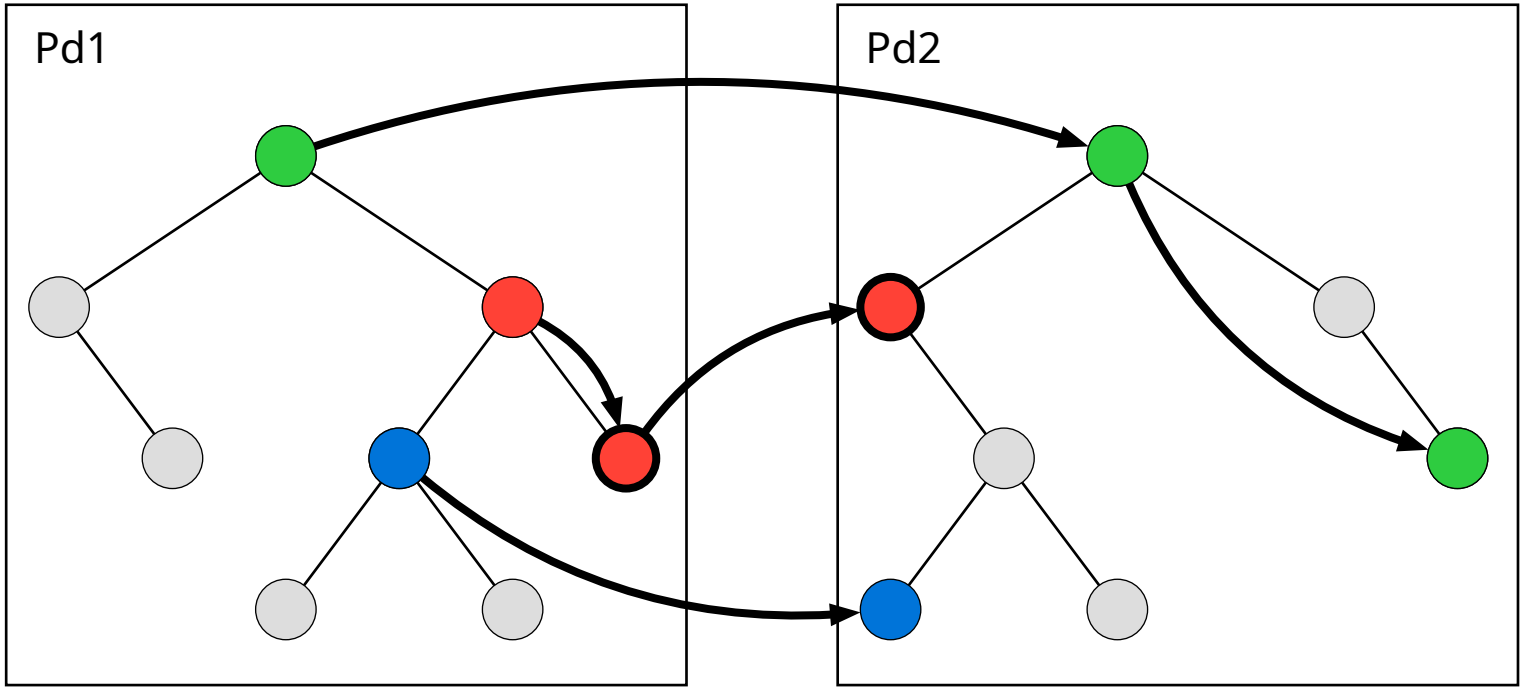
Mapping Database – Delegate



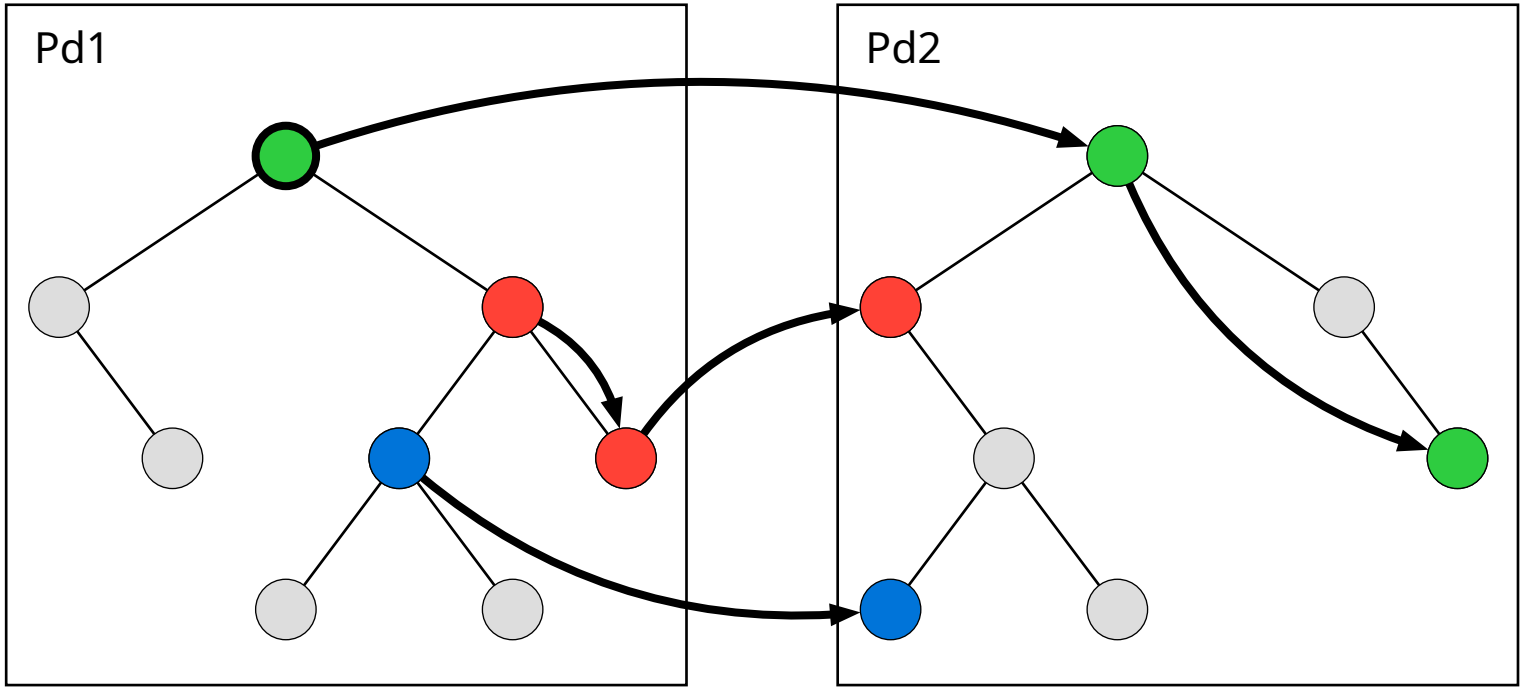
Mapping Database – Translate



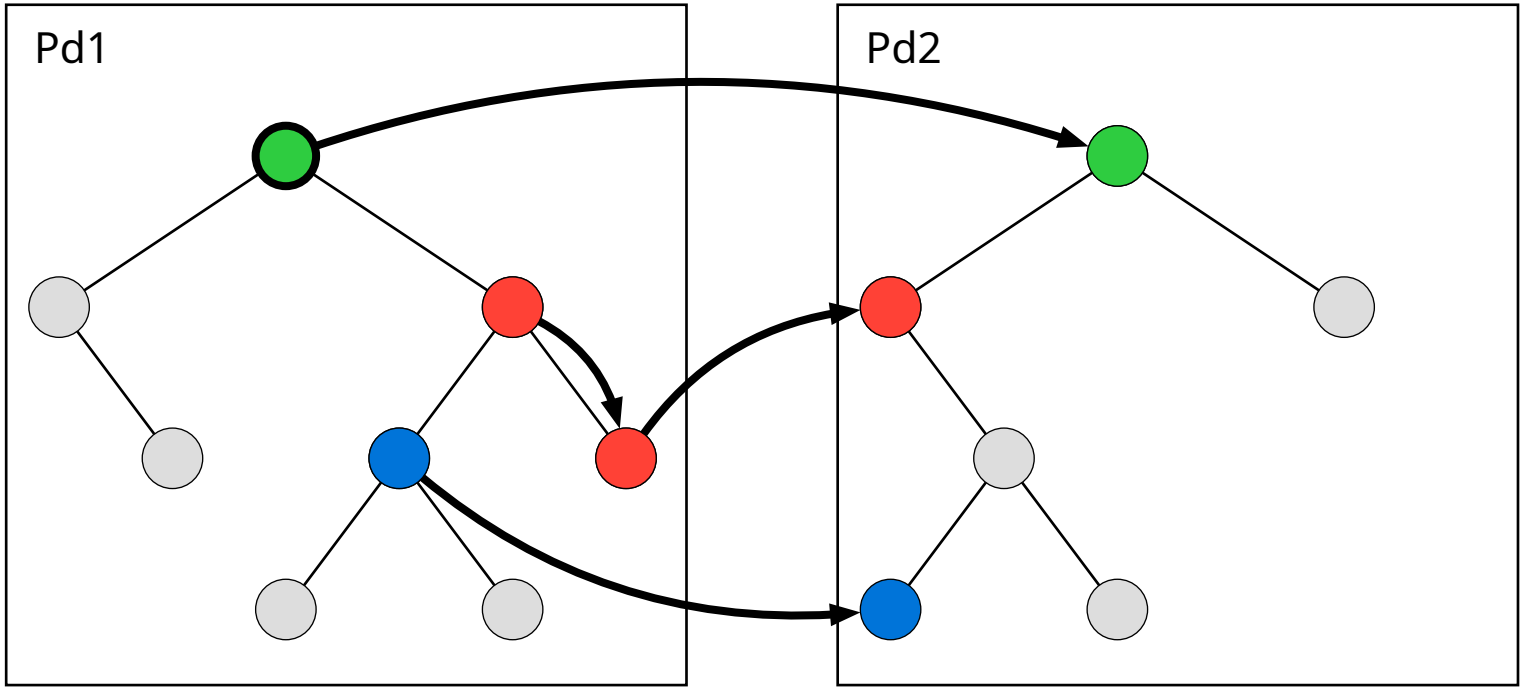
Mapping Database – Translate



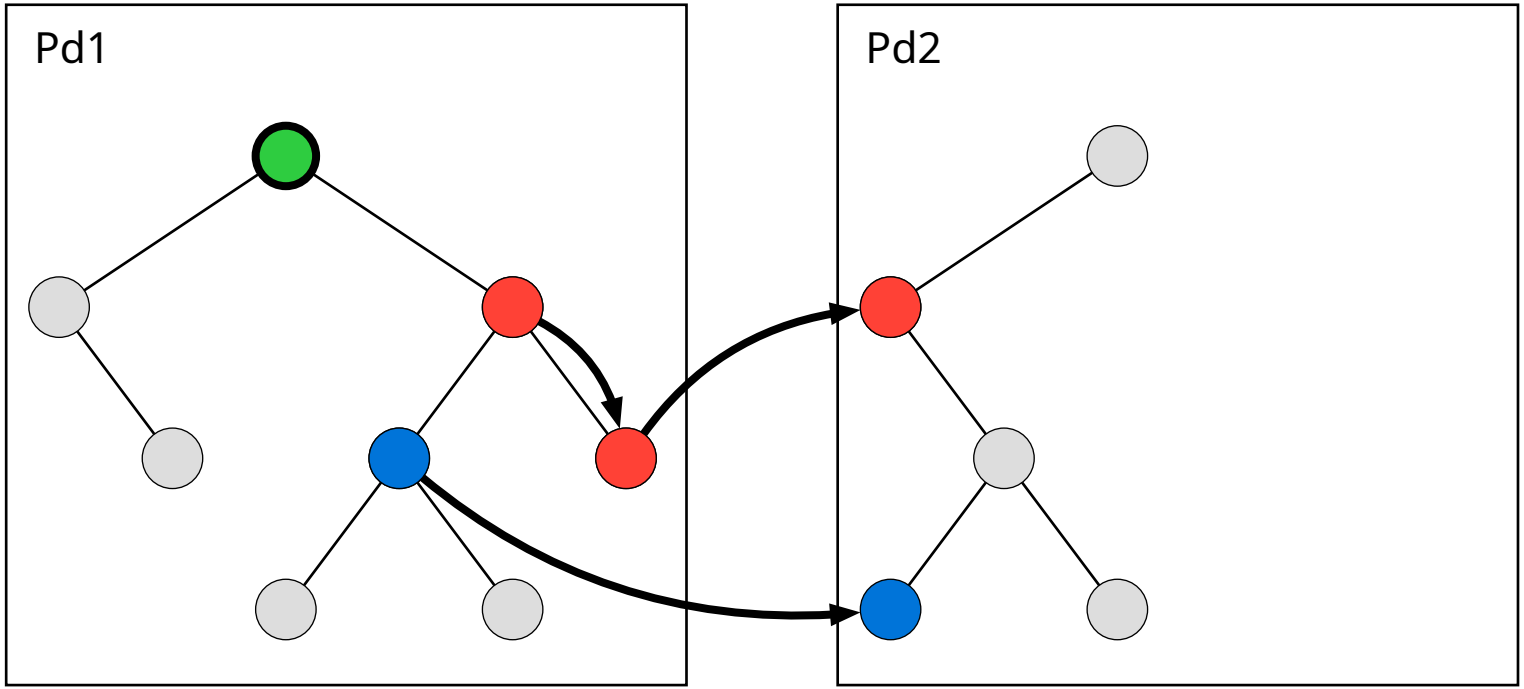
Mapping Database – Revoke



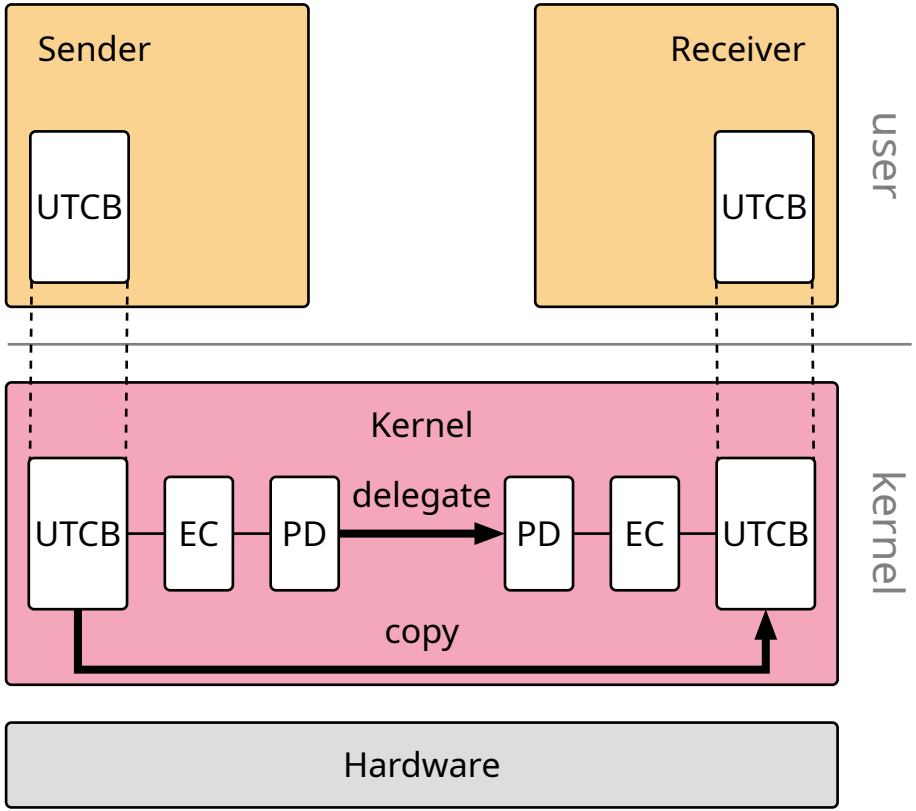
Mapping Database – Revoke



Mapping Database – Revoke

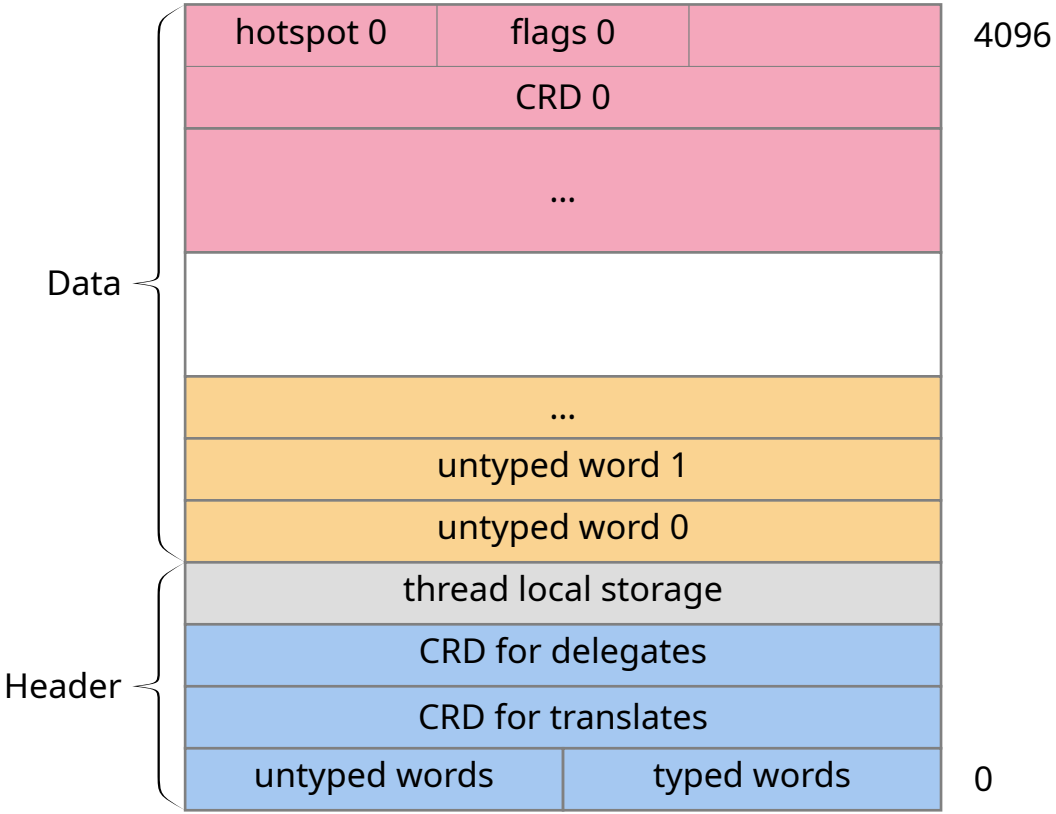


IPC With Delegate

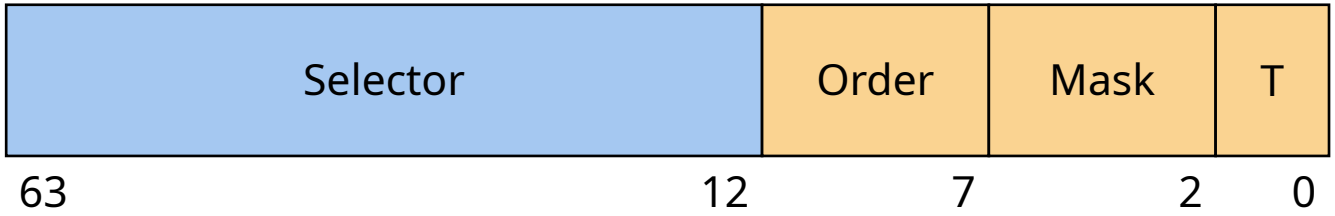


- EC has exactly one UTCB
- EC belongs to exactly one PD
- Kernel copies between UTCBs
- Kernel delegates from sender PD to receiver PD

UTCB Layout

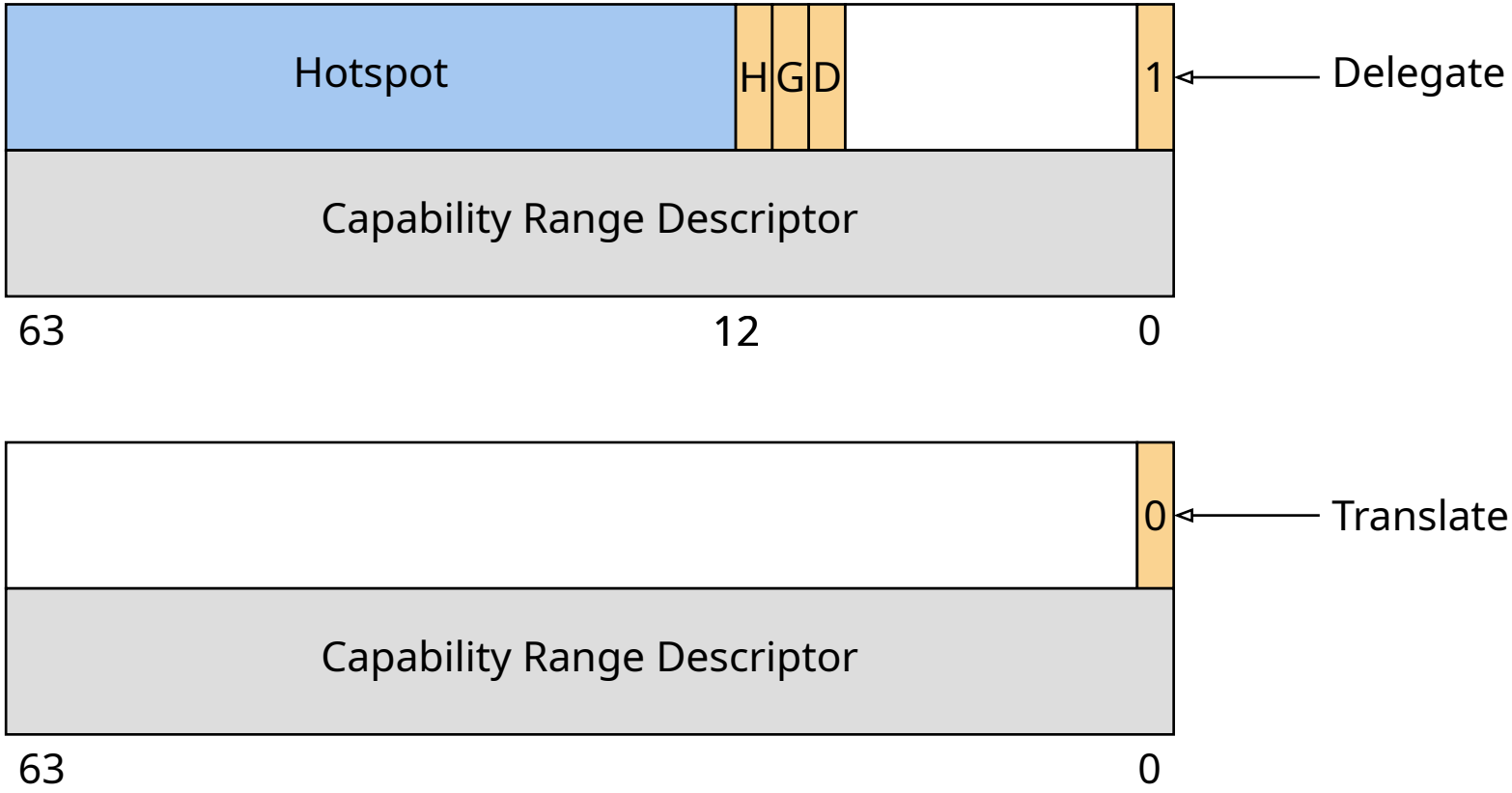

hotspot 0
flags 0
4096
CRD 0
...
Data
...
untyped word 1
untyped word 0
thread local storage
CRD for delegates
Header
CRD for translates
untyped words
typed words
0

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)

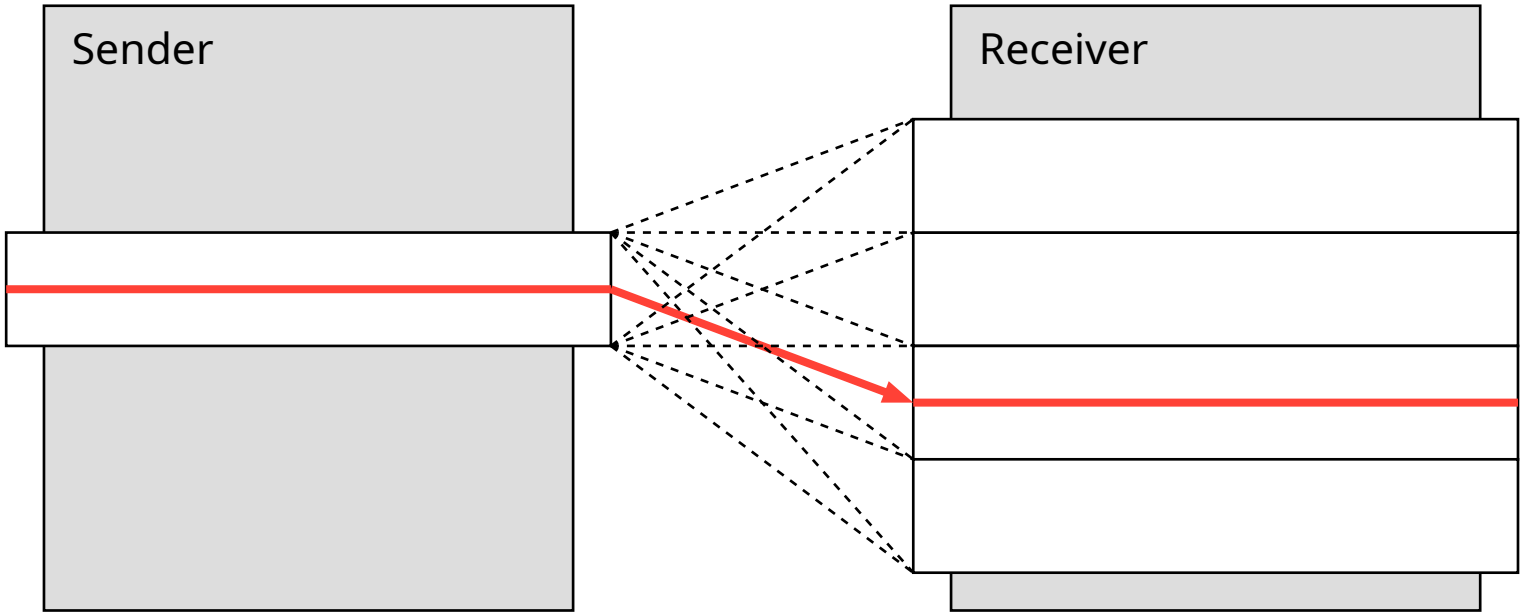
Typed Words


Hotspot
H
G
D
1
Delegate
Capability Range Descriptor
63
12
0
0
Translate
Capability Range Descriptor
63
0

Capability Delegation: Order of Events

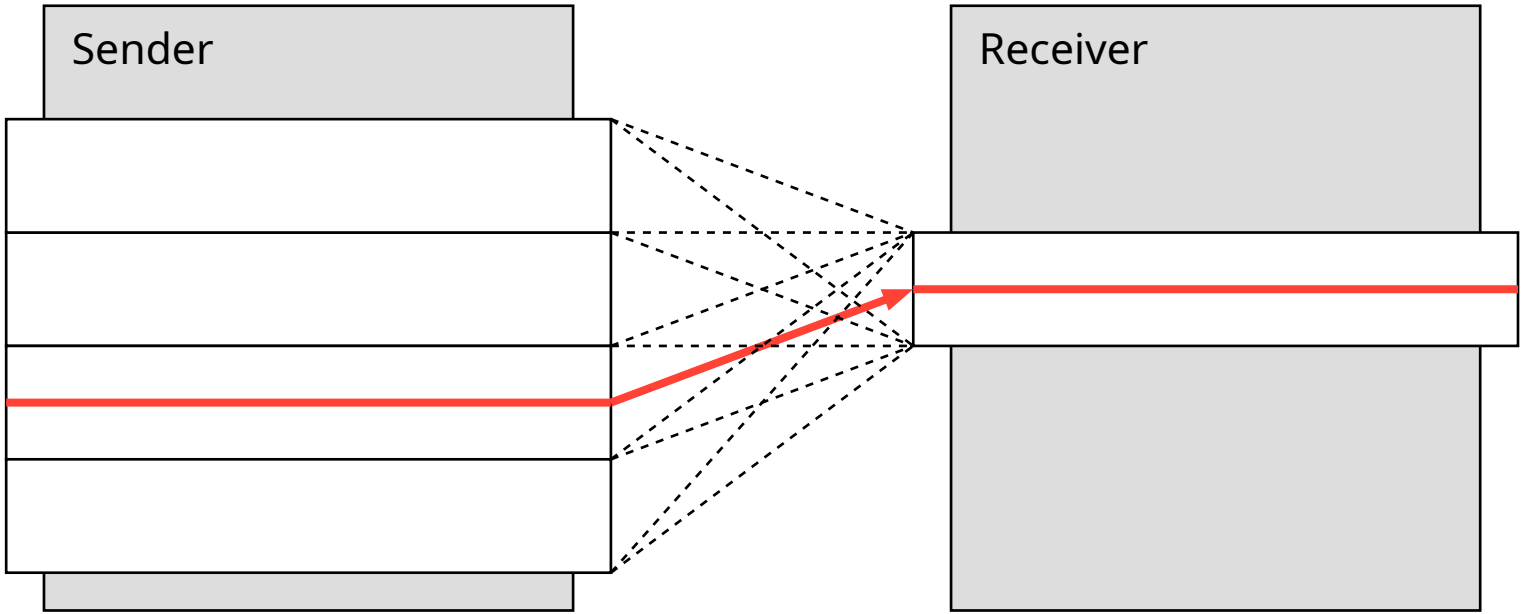
- Receiver sets up receive window (writes CRD into UTCB)
- Receivers waits for IPC
- Sender puts typed item into UTCB
- Sender calls portal
- Kernel delegates typed item
- Kernel puts typed item into UTCB, telling receiver about caps
- Kernel switches to receiver
- But: what if receive window and sent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

Matching Send and Receive Window (1)



Send window is smaller than receive window

Matching Send and Receive Window (2)



Send window is larger than receive window