An Exploration of the seL4 Kernel from Genode's Perspective

Norman Feske
<norman.feske@genode-labs.com>

Outline

1. Background (Genode)
2. The seL4 project
3. Capabilities and kernel objects
4. Virtual memory
5. What's next?

Outline

1. Background (Genode)
2. The seL4 project
3. Capabilities and kernel objects
4. Virtual memory
5. What's next?

Motivations behind Genode

- Principle of least privilege

Motivations behind Genode

- Principle of least privilege
- Mixed criticality

Motivations behind Genode

- Principle of least privilege
- Mixed criticality
- Dependability

Motivations behind Genode

- Principle of least privilege
- Mixed criticality
- Dependability
- Scalability

Motivations behind Genode

- Principle of least privilege
- Mixed criticality
- Dependability
- Scalability
- Flexibility

Key technologies

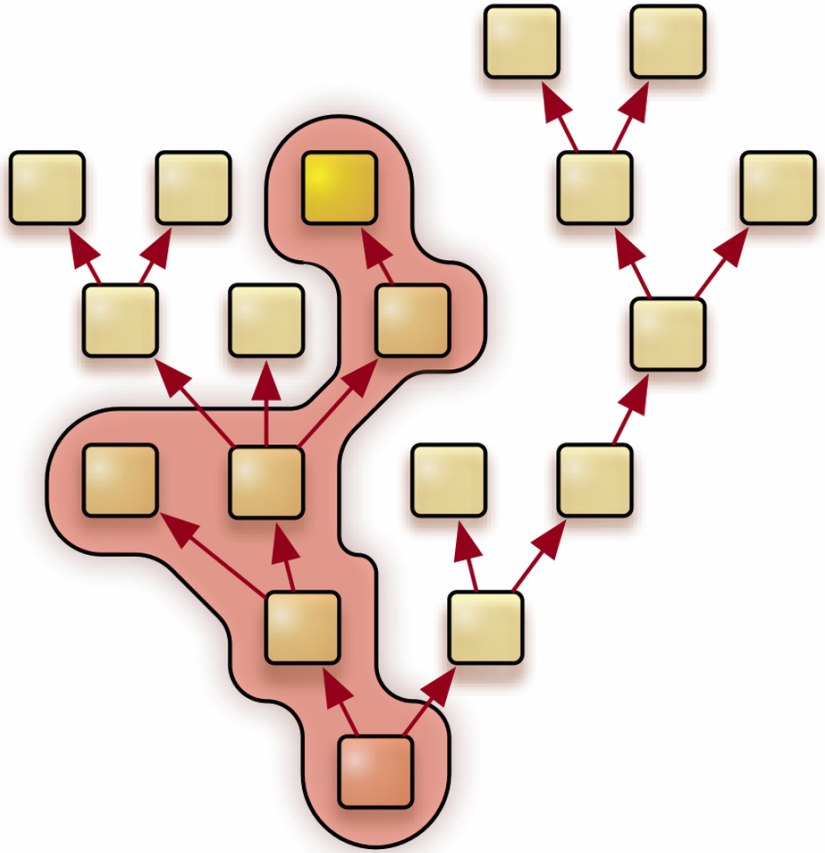
- Microkernels
- Componentization, kernelization
- Capability-based security
- Virtualization

Key technologies

- Microkernels
- Componentization, kernelization
- Capability-based security
- Virtualization

...but how to compose those?

Genode architecture

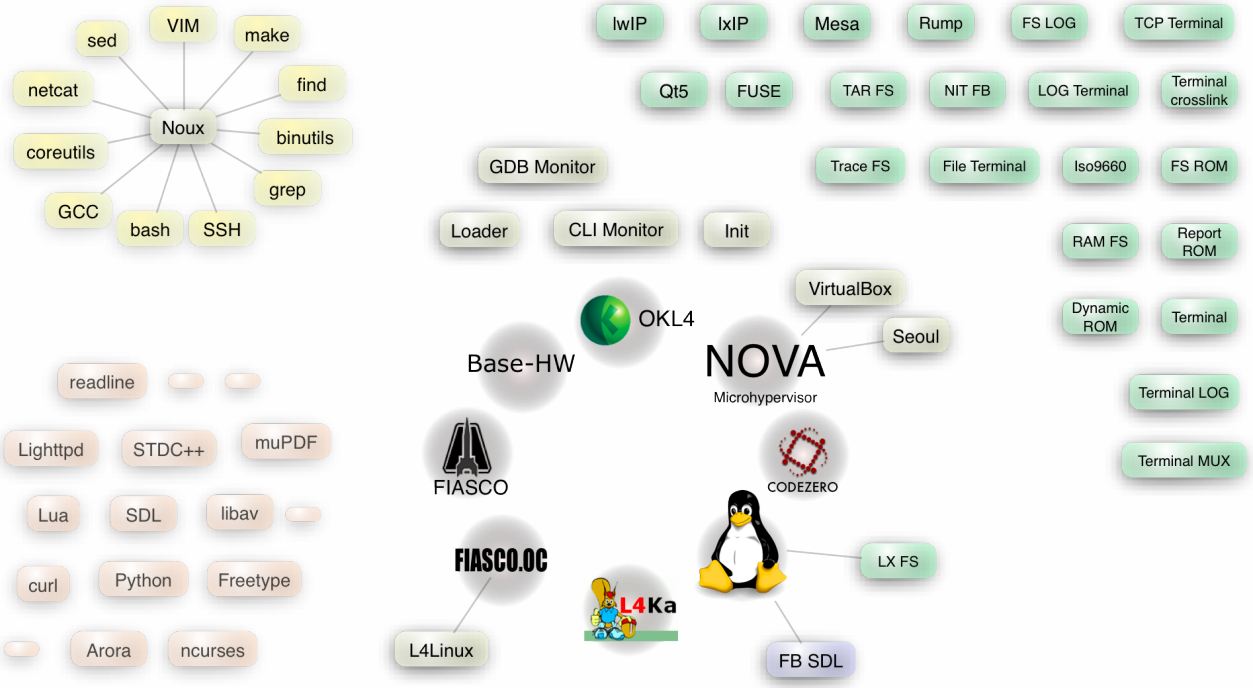


→ Application-specific TCB

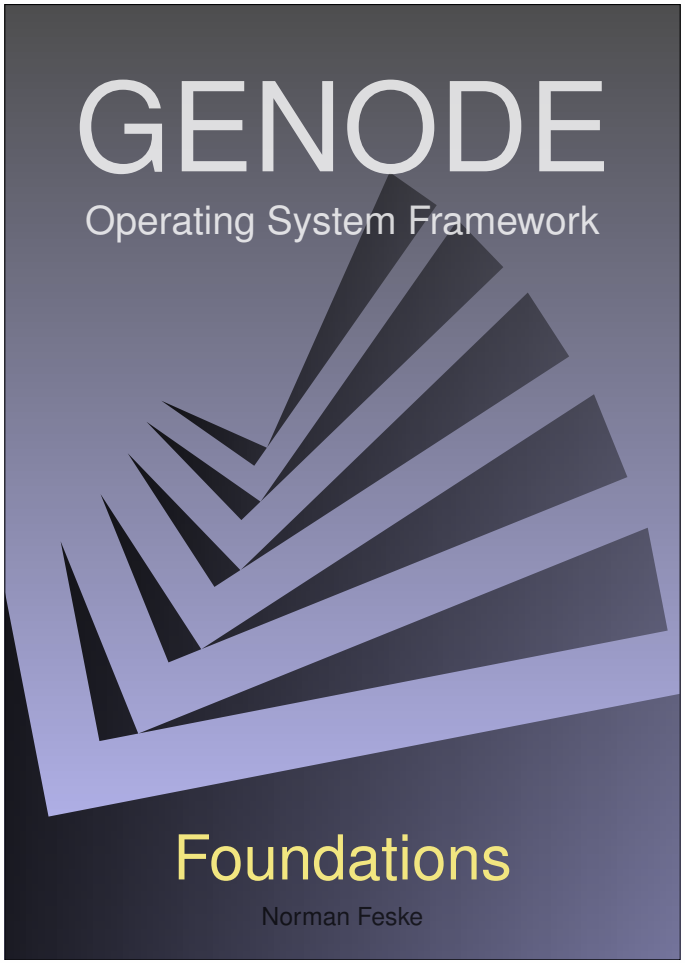
Combined with virtualization

Genode operating-system framework

Genode operating-system framework



The Book “Genode Foundations”



http://genode.org/documentation/genode-foundations-15-05.pdf

Outline

The seL4 kernel

- Developed by NICTA (DATA61) / UNSW in Sydney
- The world's first formally verified OS kernel

The seL4 kernel

- Developed by NICTA (DATA61) / UNSW in Sydney
- The world's first formally verified OS kernel
- Capability-based security
- Resilient against kernel-resource exhaustion
- Supports ARM and x86

The seL4 kernel

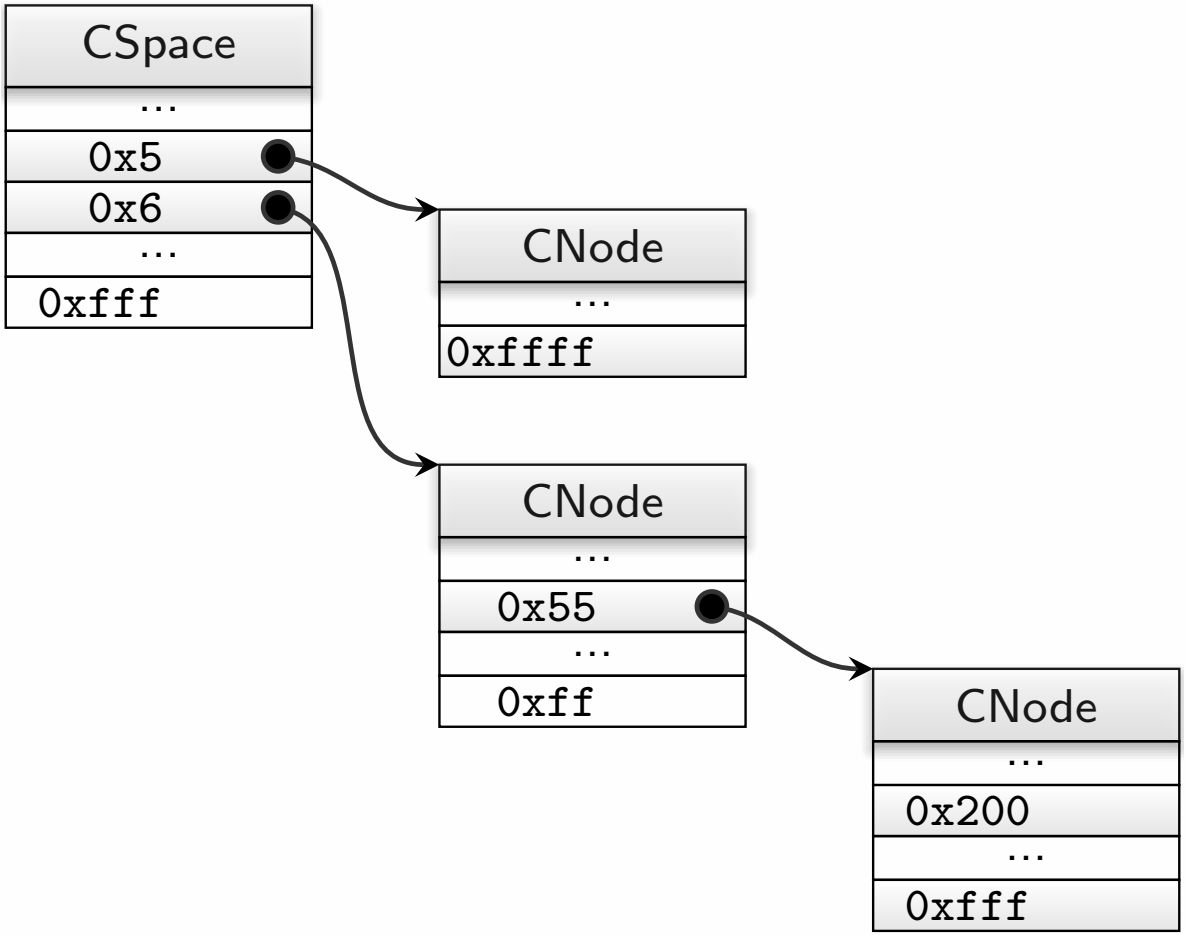
- Developed by NICTA (DATA61) / UNSW in Sydney
- The world's first formally verified OS kernel
- Capability-based security
- Resilient against kernel-resource exhaustion
- Supports ARM and x86
- GPLv2 since August 2014
- Active and dedicated community

Outline

seL4 kernel-object inventory

seL4 kernel object	Analogy
`UntypedObject`	Range of physical memory
`TCBObject`	Thread
`EndpointObject`	Destination of IPC calls
`AsyncEndpointObject`	Recipient of signals
`CapTableObject` (“CNode”)	Array of capabilities
`IA32_4K`	4 KiB page frame
`IA32_4M`	4 MiB page frame
`IA32_PageTableObject`	Page table
`IA32_PageDirectoryObject`	Protection domain

seL4 capabilities (“selectors”)

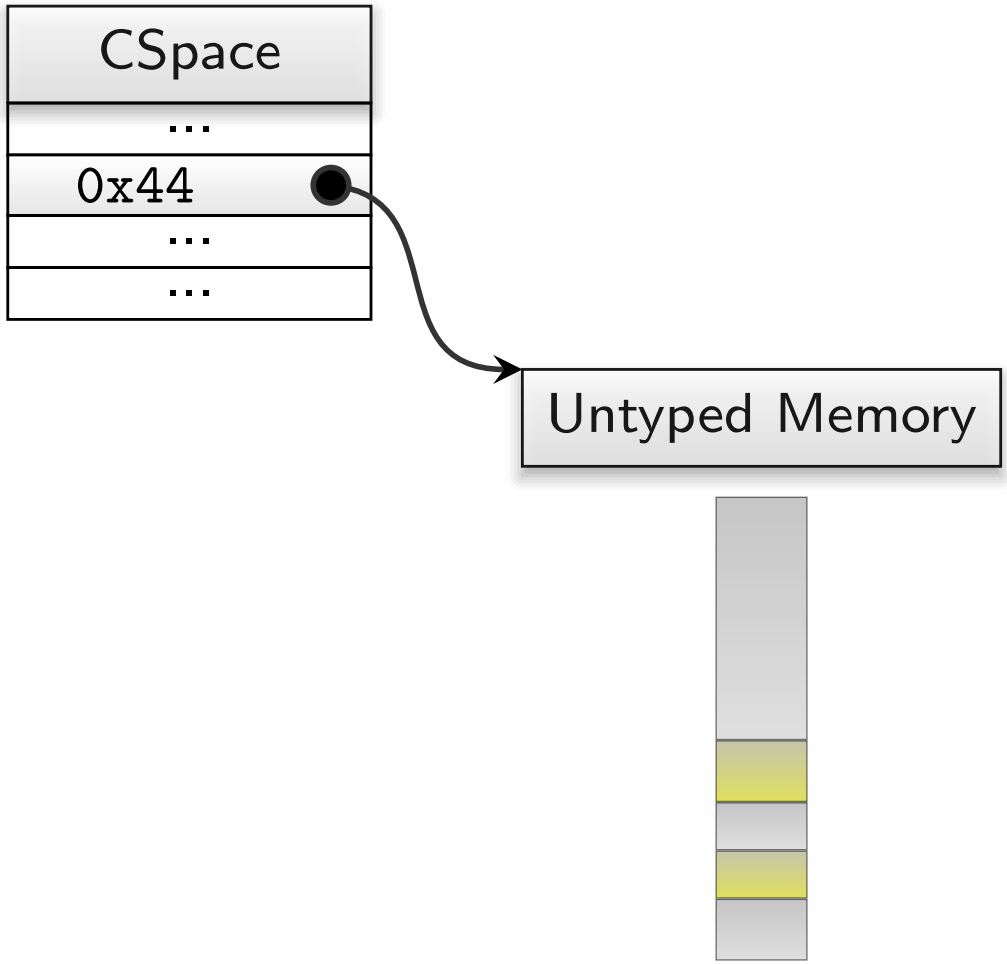


Startup

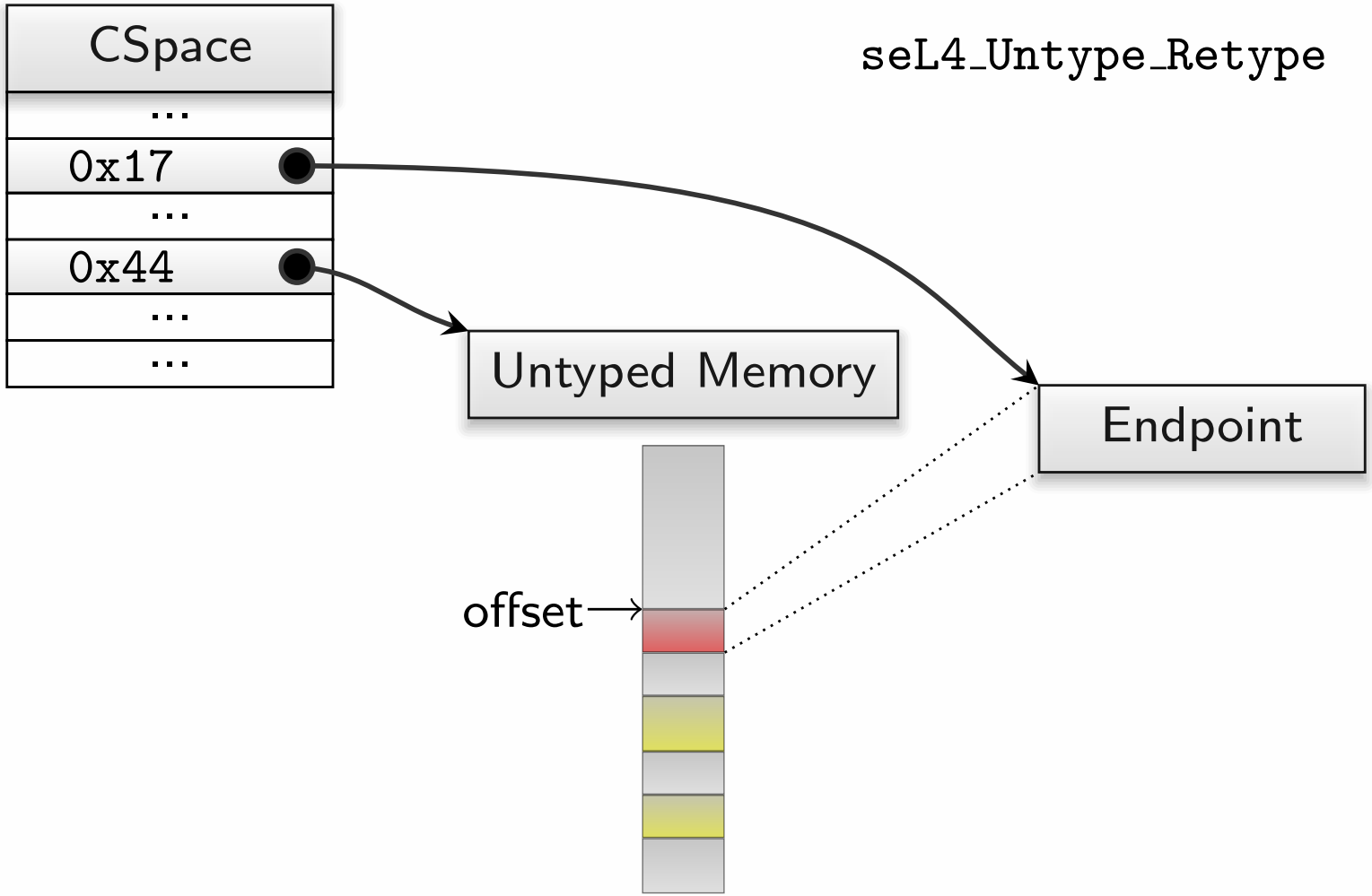
Once upon a time, there was untyped memory...

```
--- boot info ---
 initThreadCNodeSizeBits: 12
 untyped:                 [38,4d)
                          [38] [00100000,00107fff]
                          [39] [00108000,00109fff]
                          [3a] [001a0000,001bffff]
                          [3b] [001c0000,001fffff]
                          [3c] [00200000,003fffff]
                          [3d] [00400000,007fffff]
                          [3e] [00800000,00ffffff]
                          [3f] [01000000,01ffffff]
                          [40] [02000000,02ffffff]
                          [41] [03000000,037fffff]
                          [42] [03800000,03bfffff]
                          [43] [03c00000,03dfffff]
                          [44] [03e00000,03efffff]
                          [45] [03f00000,03f7ffff]
                          [46] [03f80000,03fbffff]
                          [47] [03fc0000,03fdffff]
                          [48] [03fe0000,03feffff]
                          [49] [03ff0000,03ff7fff]
                          [4a] [03ff8000,03ffbfff]
                          [4b] [03ffc000,03ffdfff]
                          [4c] [00189000,001897ff]
```

Kernel-object creation



Kernel-object creation (2)



Managing untyped memory

- Book keeping
  - Tracking of free physical memory
  - *seL4:* physical address range $\leftrightarrow$ untyped memory selector

Managing untyped memory

- Book keeping
  - Tracking of free physical memory
  - *seL4:* physical address range $\leftrightarrow$ untyped memory selector
- Untyped memory regions are naturally aligned

Managing untyped memory

- Book keeping
  - Tracking of free physical memory
  - *seL4:* physical address range $\leftrightarrow$ untyped memory selector
- Untyped memory regions are naturally aligned
- There are adjacent untyped memory regions

Managing untyped memory

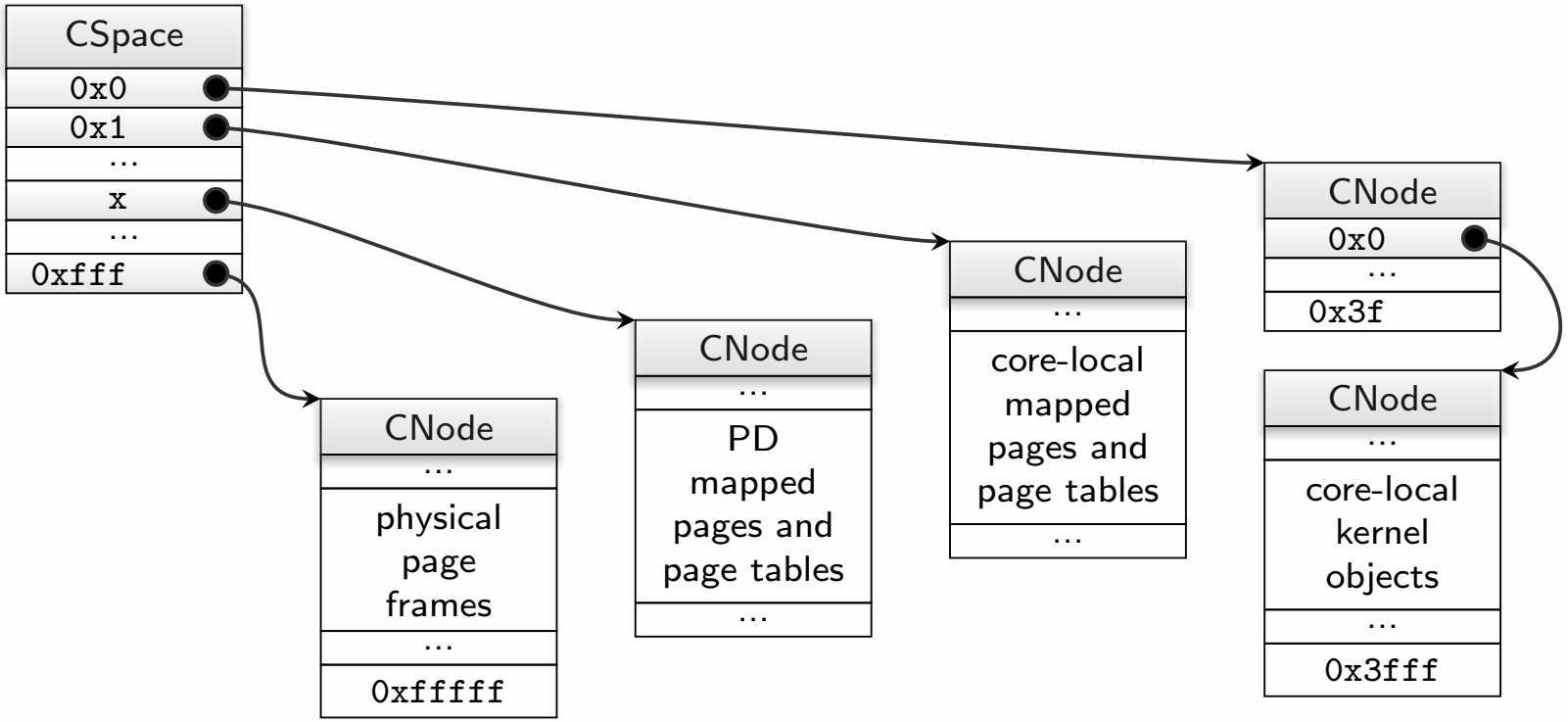
- Book keeping
  - Tracking of free physical memory
  - *seL4:* physical address range $\leftrightarrow$ untyped memory selector
- Untyped memory regions are naturally aligned
- There are adjacent untyped memory regions
- Kernel objects cannot span multiple untyped memory regions

Managing untyped memory

- Book keeping
  - Tracking of free physical memory
  - *seL4:* physical address range $\leftrightarrow$ untyped memory selector
- Untyped memory regions are naturally aligned
- There are adjacent untyped memory regions
- Kernel objects cannot span multiple untyped memory regions

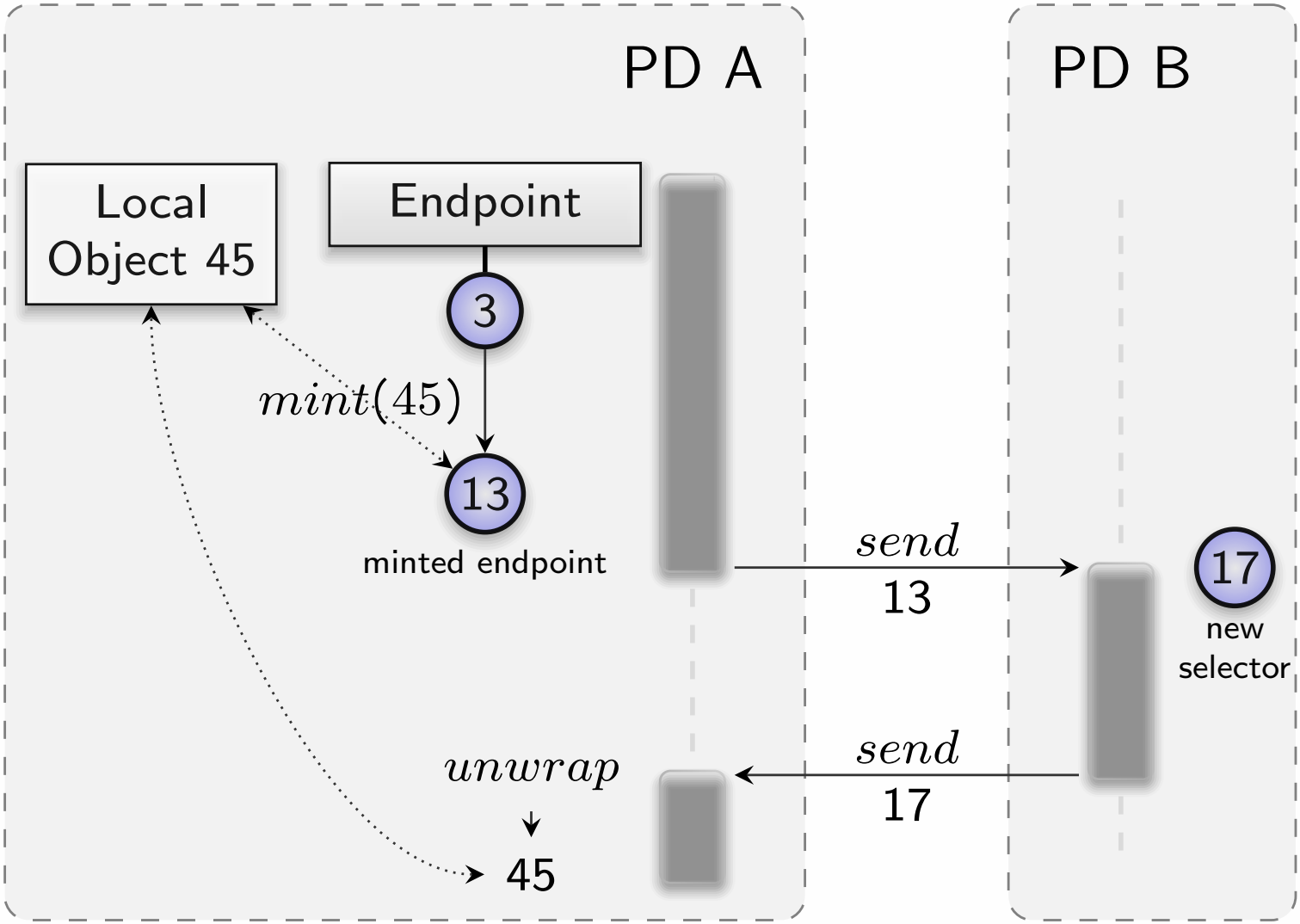
$\rightarrow$ *Trick: natural alignment of all allocations*

Core's CSpace organization



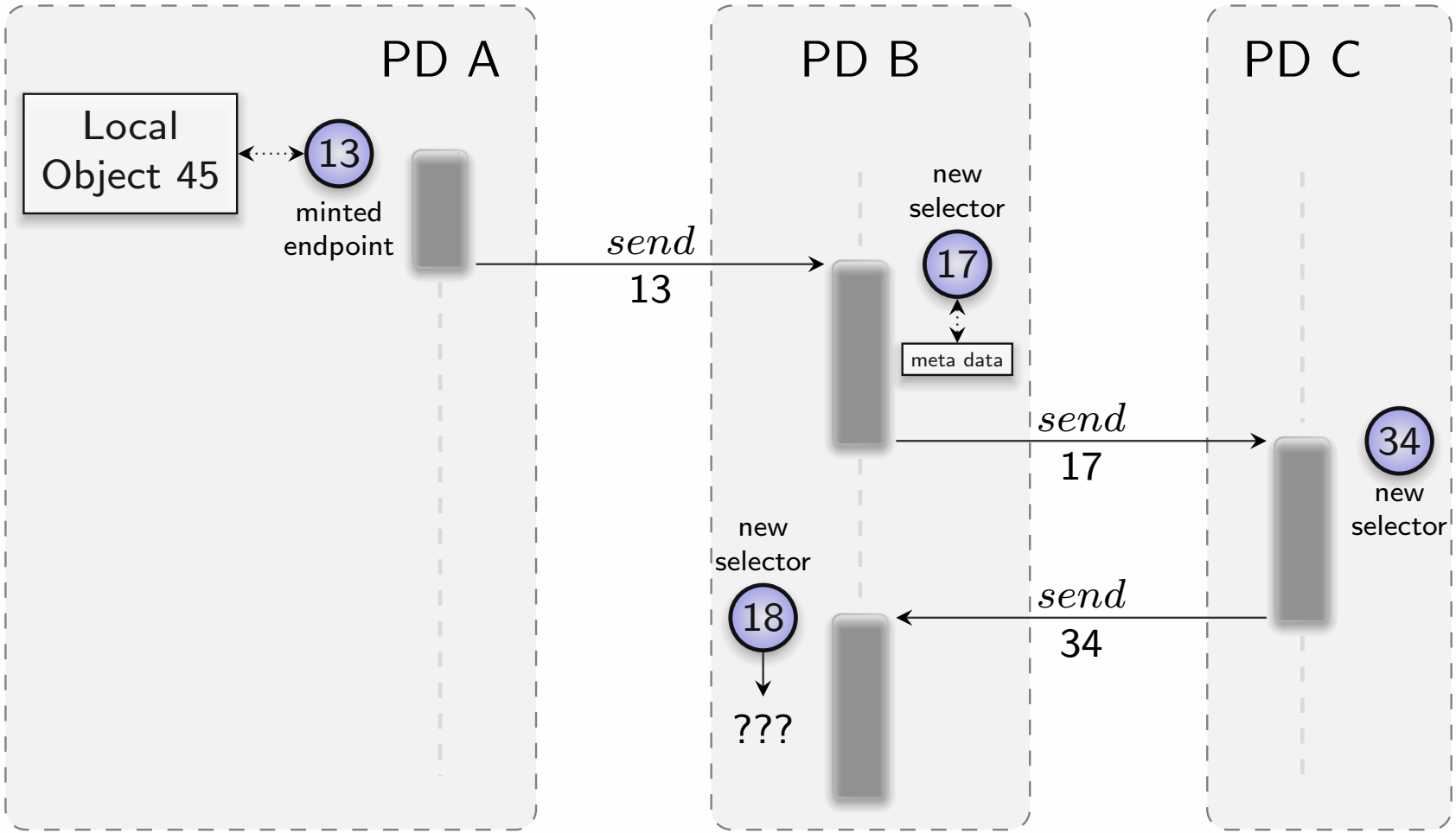
Capability delegation and invocation

Capability delegation and invocation

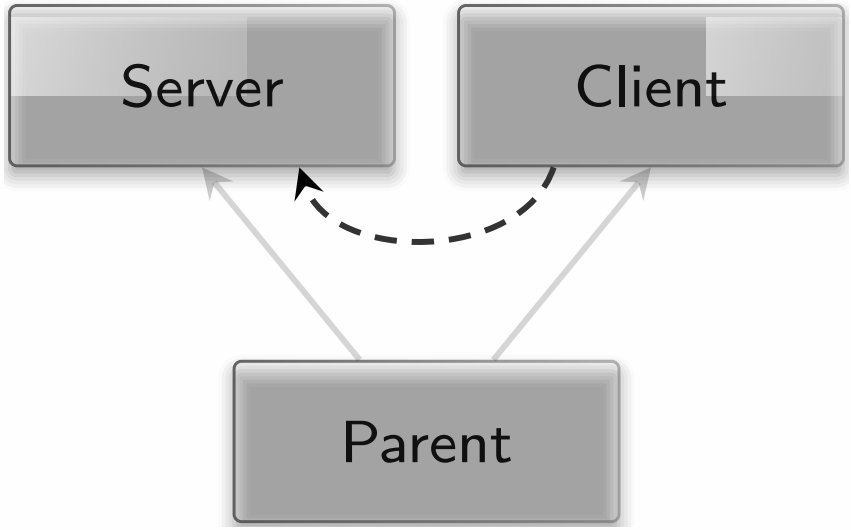


Capability re-identification problem

Capability re-identification problem

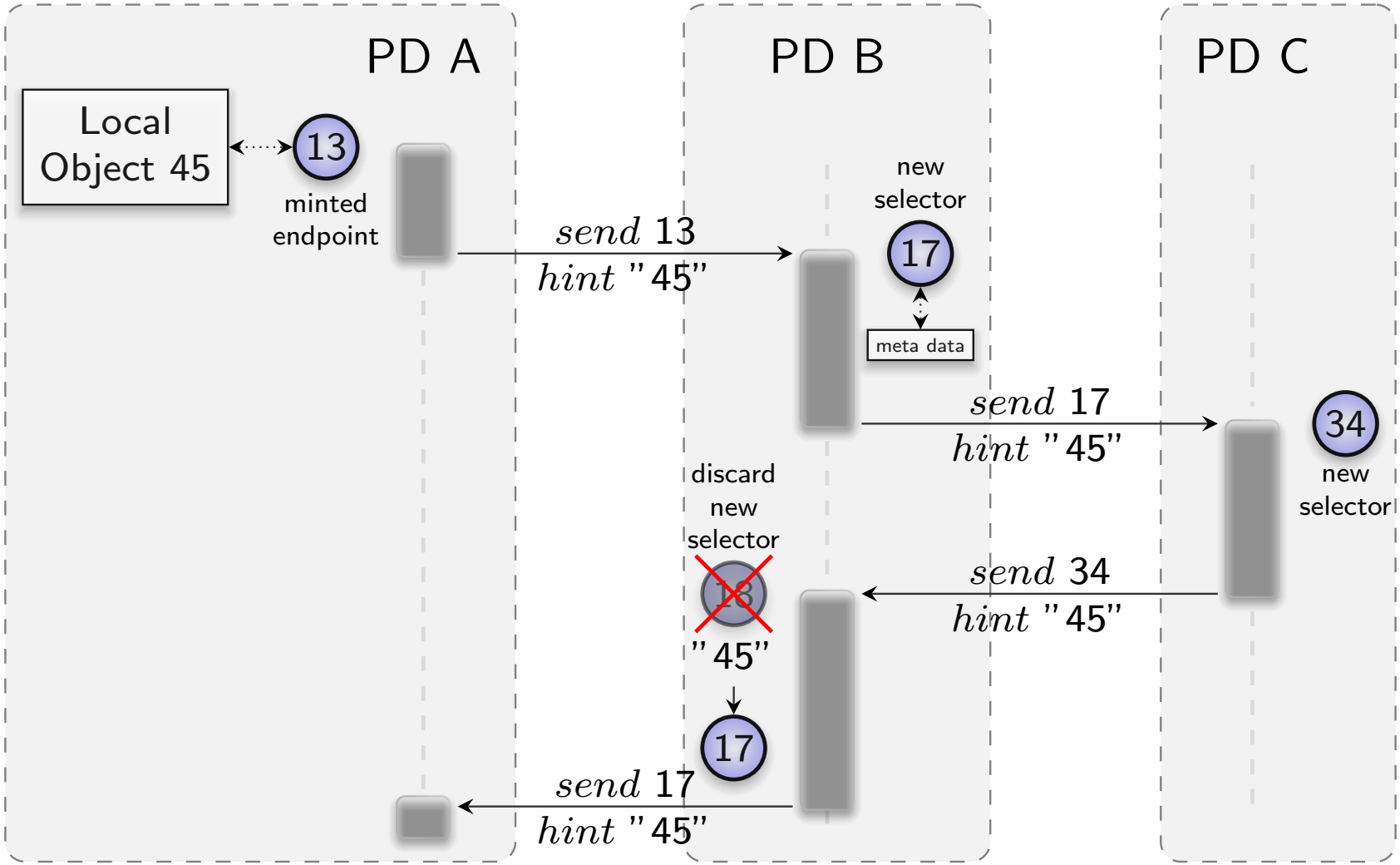


Problem in Genode



Current workaround

Current workaround



Outline

Virtual memory management

Problem

- Applications live in virtual memory
  $\rightarrow$ *The kernel maintains meta data and page tables*

Virtual memory management

Problem

- Applications live in virtual memory
  $\rightarrow$ *The kernel maintains meta data and page tables*

- Where to take the memory from?

Virtual memory management

Problem

- Applications live in virtual memory
  $\rightarrow$ *The kernel maintains meta data and page tables*

- Where to take the memory from?
  $\rightarrow$ *Traditional approach: kernel-local memory pool*

Virtual memory management

Problem

- Applications live in virtual memory
  $\rightarrow$ *The kernel maintains meta data and page tables*
- Where to take the memory from?
  $\rightarrow$ *Traditional approach: kernel-local memory pool*
- What happens when the memory get exhausted?

Virtual memory management

Problem

- Applications live in virtual memory
  $\rightarrow$ *The kernel maintains meta data and page tables*
- Where to take the memory from?
  $\rightarrow$ *Traditional approach: kernel-local memory pool*
- What happens when the memory get exhausted?
  $\rightarrow$ *Panic!*
- Who provokes kernel memory consumption?

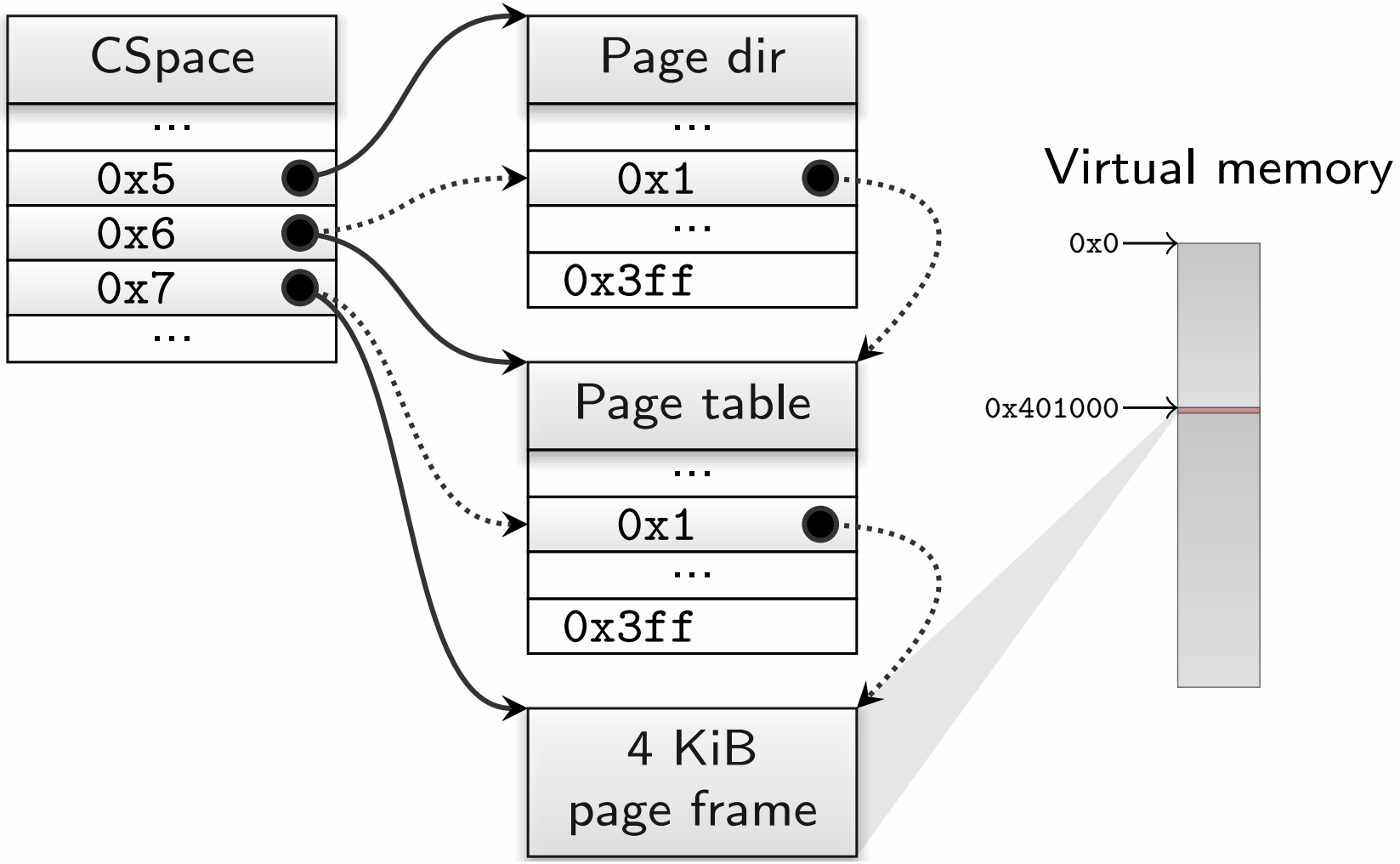
Virtual memory management

Problem

- Applications live in virtual memory
  → *The kernel maintains meta data and page tables*
- Where to take the memory from?
  → *Traditional approach: kernel-local memory pool*
- What happens when the memory get exhausted?
  → *Panic!*
- Who provokes kernel memory consumption?
  → *Untrusted application code!*

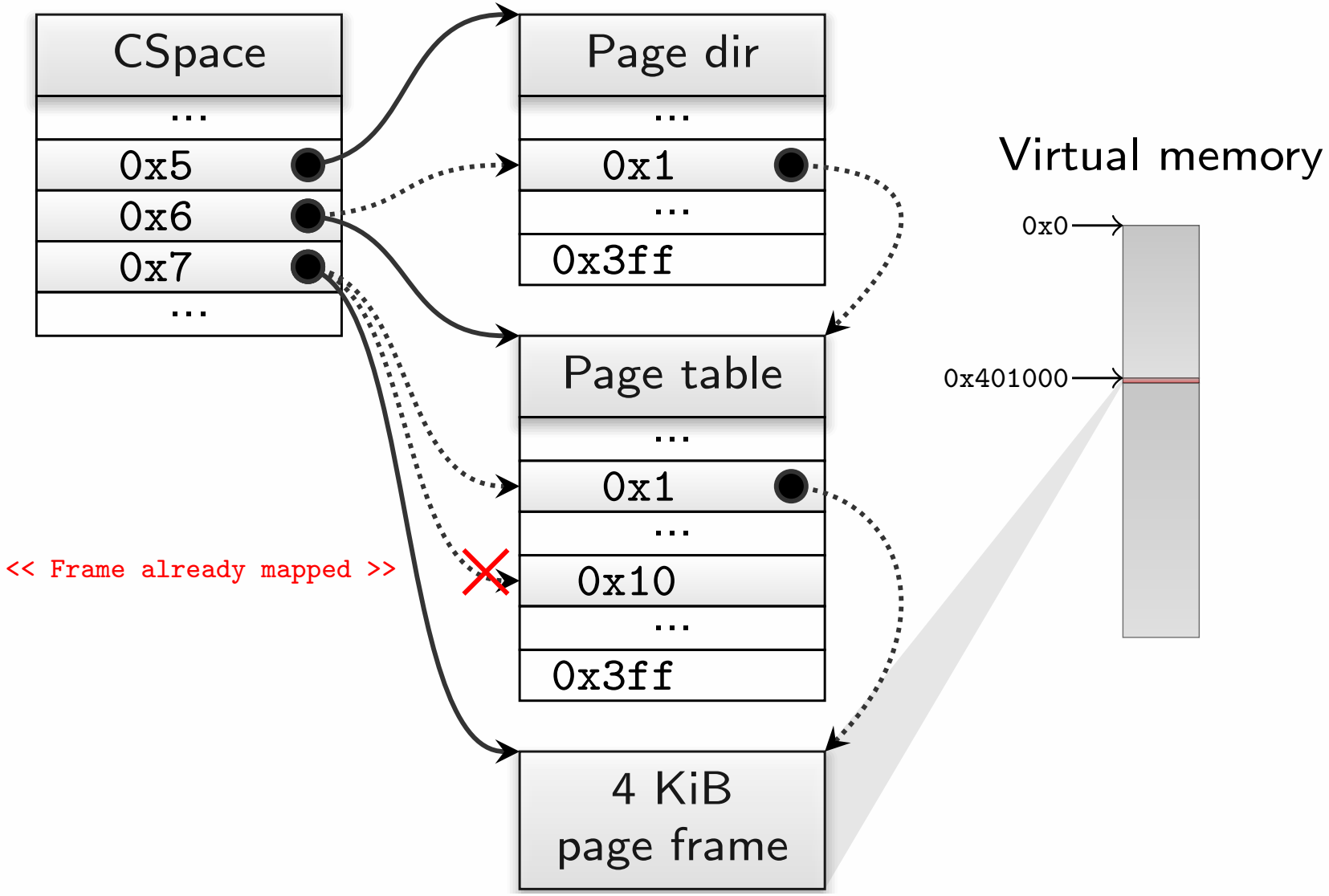
The seL4 way of virtual memory management

The seL4 way of virtual memory management

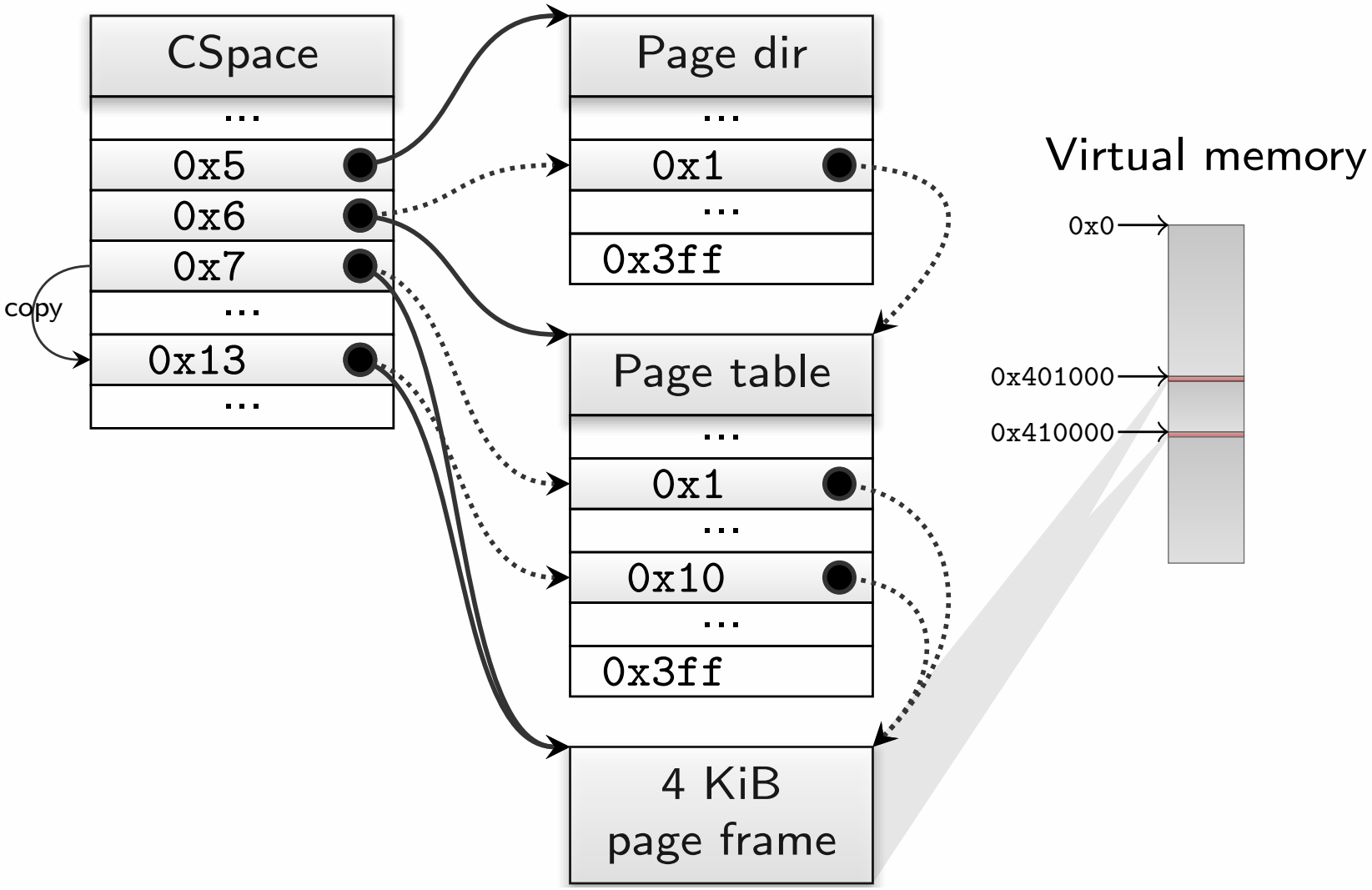


Attempt to map a page twice

Attempt to map a page twice



Mapping a page twice, the seL4 way



Implications for Genode

On Genode, each page must be considered as shared memory

Implications for Genode

On Genode, each page must be considered as shared memory

→ One kernel object (CNode entry) for each page-table entry

Implications for Genode

On Genode, each page must be considered as shared memory

→ One kernel object (CNode entry) for each page-table entry

→ How to name the selectors?

Implications for Genode

On Genode, each page must be considered as shared memory

→ One kernel object (CNode entry) for each page-table entry

→ How to name the selectors? Preallocation is infeasible.

Implications for Genode

On Genode, each page must be considered as shared memory

→ One kernel object (CNode entry) for each page-table entry

→ How to name the selectors? Preallocation is infeasible.

Solution: Virtual software-loaded TLB

- Fixed pool of page tables per PD, used in LRU fashion

Implications for Genode

On Genode, each page must be considered as shared memory

→ One kernel object (CNode entry) for each page-table entry

→ How to name the selectors? Preallocation is infeasible.

Solution: Virtual software-loaded TLB

- Fixed pool of page tables per PD, used in LRU fashion
- Leveraging Genode's resource trading mechanism:
  → Page-table pool size is a PD-specific

Outline

What's next?

- seL4 2.0
- Signal API backend, interrupts
- Memory-mapped I/O
- Real lock implementation
- Shared library support

→ *Interactive scenarios by mid 2016*

Open issues

- Asynchronous notifications

Open issues

- Asynchronous notifications
- Capability integrity protection

Open issues

- Asynchronous notifications
- Capability integrity protection
- Superpages

Thank you

Articles about Genode on seL4

http://genode.org/documentation/articles

Genode OS Framework

http://genode.org

Genode Labs GmbH

http://www.genode-labs.com

Source code at GitHub

http://github.com/genodelabs/genode