Capabilities and Memory Management

This week: Physical Memory Management

- You will get:
  - Barrelfish CPU driver
  - Simple init program
  - Libraries
- Your task: implement memory allocation
  - Accounting for physical memory
  - Mapping virtual memory

ACCESS CONTROL MODELS

Access control matrix

- Representation/definition of permissible operations in a system

	Objects				
Subjects	$user_1$	$user_2$	$user_3$	$file_1$	$file_2$
$user_1$		Send msg		RW	R
$user_2$	Send msg				RW
$user_3$	Set passwd	Set passwd	Set passwd		R

- **Subjects**: users, processes, groups, etc.
- **Objects**: other users/processes, files, memory objects, etc.
- **Privileges/rights**: depends on object
  - for file: read, write, execute, etc.

Access control matrix properties

- Dynamic data structure, frequent changes
- Very sparse with many repeated entries
- Impractical to store explicitly

Most common discretionary mechanisms:

- **Access control list**: stores a column (who can access this)
- **Capabilities**: store a row (what this can access)

Issues for discretionary access control

- **Propagation**:
  - Can a subject grant access to another?
- **Restriction**:
  - Can a subject propagate a subset of its own rights?
- **Revocation**:
  - Can access, once granted, be revoked?
- **Amplification**:
  - Can an unprivileged subject perform restricted operations?
- **Determination of object accessibility**:
  - Which subjects have access to a particular object?
  - Is an object accessible by any subject? (garbage collection)
- **Determination of subject's protection domain**:
  - Which objects are accessible to a particular subject?

Access Control Lists

- Implemented by most commodity systems
- ACL associated with object
  - Propagation: meta right (eg. owner may chmod)
  - Restriction: meta right
  - Revocation: meta right
  - Amplification: protected invocation right (eg. setuid)
  - Accessibility: explicit in ACL
  - Protection domain: hard (if not impossible) to determine
- Usually condensed via groups / classes
- Can have negative rights
- Sometimes implicit (eg. UNIX process hierarchy)

UNIX ACLs

- Despite modern terminology, classic UNIX file privileges are a (restricted) ACL representation

```
drwxrwsr-x   8 simpeter netos          4096 Nov 29  2007 pistachio-ka/
drwxr-xr-x  21 scadrian netos          4096 Dec  5  2007 PROT_SYS/
drwxrwsr-x   2 simpeter netos          4096 Sep 11  2011 replay/
drwxrwsr-x   5 troscoe  netos          4096 May 20  2011 scc/
drwxrwsr-x   2 simpeter netos          4096 May 14  2009 simnow/
-rw-rw-r--   1 simpeter inf     2563440640 Nov 25  2009 sol-10-u8-ga-x86-dvd.iso
drwxrwsr-x   4 troscoe  netos          4096 Mar 30  2012 solarflare/
```

ACL

rights

subject (owner)

subject (group)

object

- Permissions for *other* are an implicit group of subjects

CAPABILITIES

Capabilities

- Capability list associated with subject
- Each capability confers a certain right to its holder
  - Propagation: copy/transfer capabilities between subjects
  - Restriction: requires creation of new (derived) caps
  - Revocation: requires invalidation of caps from all subjects (may be difficult)
  - Amplification: special invocation capability
  - Accessibility: requires inspection of all capability lists (hard if not impossible to determine)
  - Protection domain: explicit in capability list

(Partial) history of capability systems

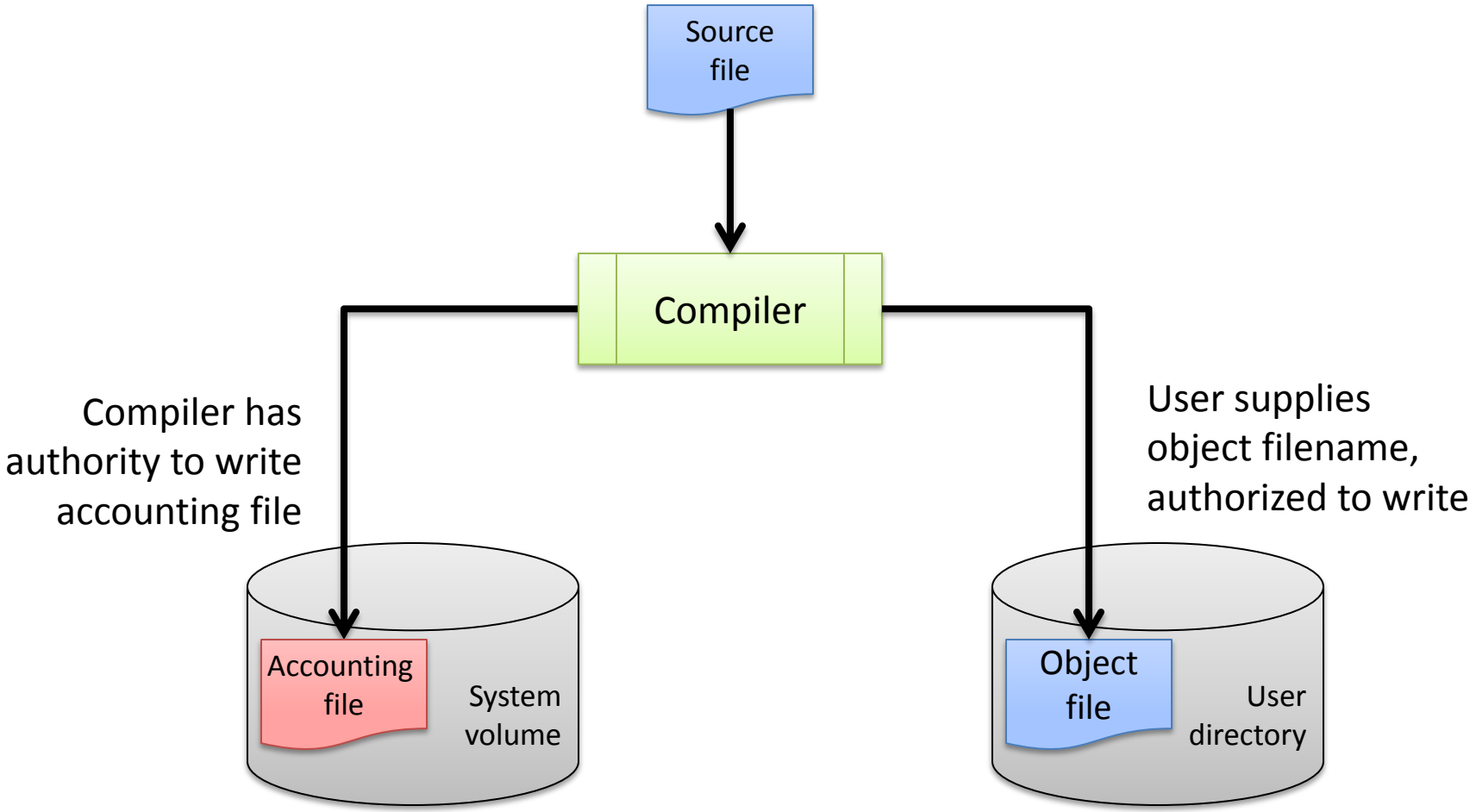
1961	Burroughs B5000
1966	Dennis and Van Horn's supervisor
1967	MIT PDP-1 Timesharing System Chicago Magic Number Machine
1968	Berkeley CAL-TSS
1972	Plessey System 250
1970-79	Cambridge CAP Computer
1974	CMU Hydra; then StarOS
1979-2007	Gnosis, KeyKOS, Eros, Coyotos…
1980-present	IBM System/38, AS/400, iSeries, System i, …
1981	Intel iAPX 432
1986	Linn Rekursiv
2005-present	seL4
2009-present	Capsicum / CHERI (BSD)

Capabilities

- Main advantage of capabilities is fine-grained access control
  - ⇒ Easy to provide access to specific subjects
  - ⇒ Easy to delegate permissions to others
- A cap presents prima facie evidence of right to access
  - Think of it as a key
  - Any representation must protect capabilities against forgery
- Consists of object identifier and a set of access rights
  - Implies *object naming*

Solves the "confused deputy" problem

The Confused Deputy



The Confused Deputy

- Regular command line:

```
cc –o ~/program program.c
```

- But what about:

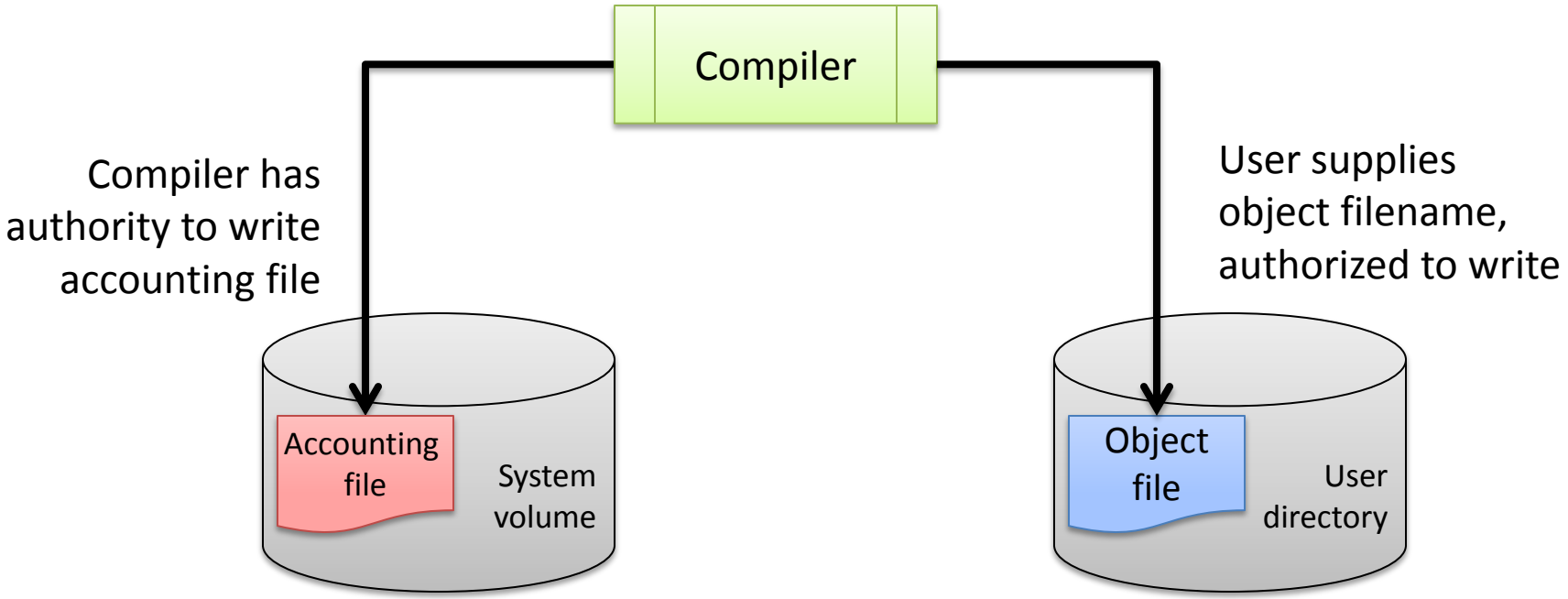
```
cc –o /var/spool/accounts program.c
```

?

- Oh dear…

The Confused Deputy

- Compiler has two masters:
  - User
  - System accounting
- Acquires blanket authority from both



Solution using capabilities

Capability: File

Source file

+

Output file

Compiler

Accounting file

Compiler has authority to write accounting file

User supplies object filename, authorized to write

Accounting file

System volume

Object file

User directory

IMPLEMENTATION OPTIONS FOR CAPABILITIES

How are capabilities implemented and protected?

Options:

1. Tagged: protected by hardware
2. Sparse: protected by sparsity
   - (probabilistically secure, like encryption
3. Partitioned: protected by software

Tagged capabilities

AKA hardware capabilities

- Extra tag bit with every memory word (or group thereof)
  - Tag identifies capabilities
  - Capabilities may be used and copied like “normal” pointers
  - Hardware checks permissions when dereferencing capability
  - Modifications turn the tag off (reverting caps to plain data)
  - Only the kernel can turn a tag bit on

  ✓ Propagation easy
  - Restriction requires kernel to create new (weaker) capability
  ✗ Revocation virtually impossible (requires memory scan)
  ✗ Accessibility virtually impossible to determine

Tagged capabilities outside RAM

- Disk has no tags
- AS/400 simulates them by restricting physical I/O to the low-level OS
  - Extra bit stored for every word on disk
  - Page-out $\Rightarrow$ page scanned and tags collected
  - Page-in $\Rightarrow$ tags are reconstructed
- Significant processing overhead on all disk I/O

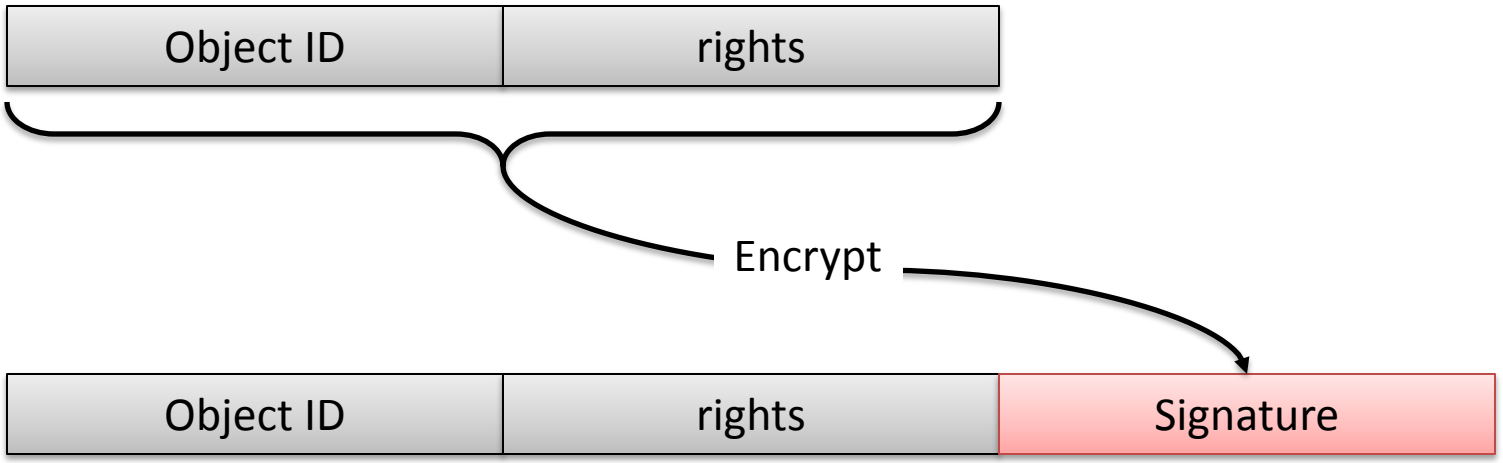
Tagged capabilities: summary

- Secure through hardware protection
- Convenient for applications (appear as normal pointers)
- Checked by hardware $\Rightarrow$ fast validation
- Capability hardware is complex (hence slow)
- Separate mechanisms required for I/O and distribution

Sparse capabilities

- Basic idea similar to encryption:
- Add bit string to capability
  - Makes valid capabilities a tiny subset of the capability space
  - Secure by infeasibility of exhaustive search of cap space

Encrypted sparse capabilities



Signature: bit string is encrypted object info

- Cap consists of object ID, rights, and signature
- Signature = object ID and rights encrypted with a private key
- Validated by checking signature

Password capabilities

Capability:

Object ID	Password

Global object table:

OID	Password	Rights

- Password: bit string is random data
  - Cap consists of object ID, and password
  - Rights determined by looking up password in a global object table

Sparse capabilities: summary

- Sparse caps are regular user-level objects
  - Can be passed like other data
  - Similar to tagged caps, but without hardware support
- Validated at invocation time (either explicitly or implicitly)

Issues:

- Full mediation requires extra support
  - See Mungi
- High amplification of leaked data
  - Problem with covert channels

Partitioned capabilities

- System maintains capabilities for each process,
  - eg. as a capability list (clist)
- User code uses only handles (indirect references) to caps
- System validates access when performing any privileged operation (eg. mapping a page)
  - Validation is explicit at syscall time
  - Propagation: system call to copy a cap between clients
  - Restriction: invoke kernel to create new cap
  - Revocation: invoke kernel to remove cap from clist
  - Accessibility: requires scanning all clists
  - Protection domain: explicitly represented in clist
- Used in Hydra, Mach, KeyKOS, EROS, seL4, Barrelfish, many others

Memory partitioning

- CNode capabilities:
  - Storage for capability representations
  - Divided into “slots”
  - Unreadable from user space!
- Frame capabilities
  - May be “mapped” into user virtual address space
  - Multiple of page size

There must **never** be an area of memory referred to by both a CNode cap and a Frame cap!

Partitioned capabilities: summary

- Secure through kernel protection
- Real caps live only in kernel space
- Validation at mapping/invocation time
- Apps use “normal” pointers
  - ✓ Fast validation possible (for memory objects, validation is cached by MMU)
- Propagation requires kernel intervention
  - ✓ Reference counting and revocation possible with kernel support
  - ✓ No special hardware requirements

CAPABILITIES IN BARRELFISH

Barrelfish design decision

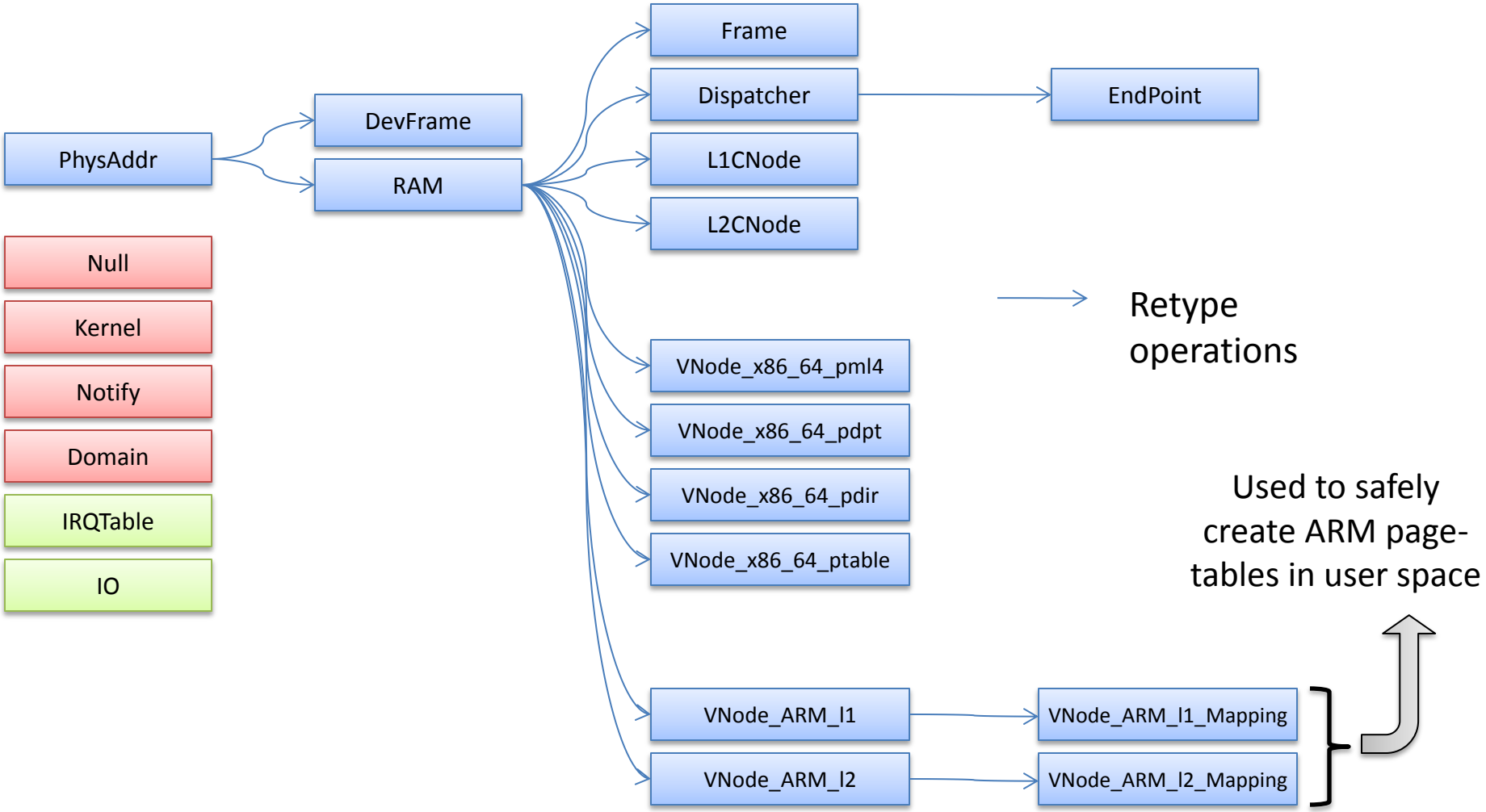
Provide one general mechanism for tracking all OS resources

- Disadvantages:
  - Mechanism must be fully general (more complex than point solutions)
  - Potential performance hit for some operations (price of generality)
- Advantages:
  - Solve this once, everything works
  - Provides insight into the general problem for multicore hardware
    $\Rightarrow$ good from research perspective

Capabilities in Barrelfish

- **All** memory described using capabilities
  - Along with some other system resources
- Capabilities are typed ⇒ type of memory
  - Memory can be retyped by its owner
  - Subject to certain rules
- Memory regions can be split
  - Result: 2 new capabilities, of the same type
  - One for each half

Most of the Barrelfish capability system

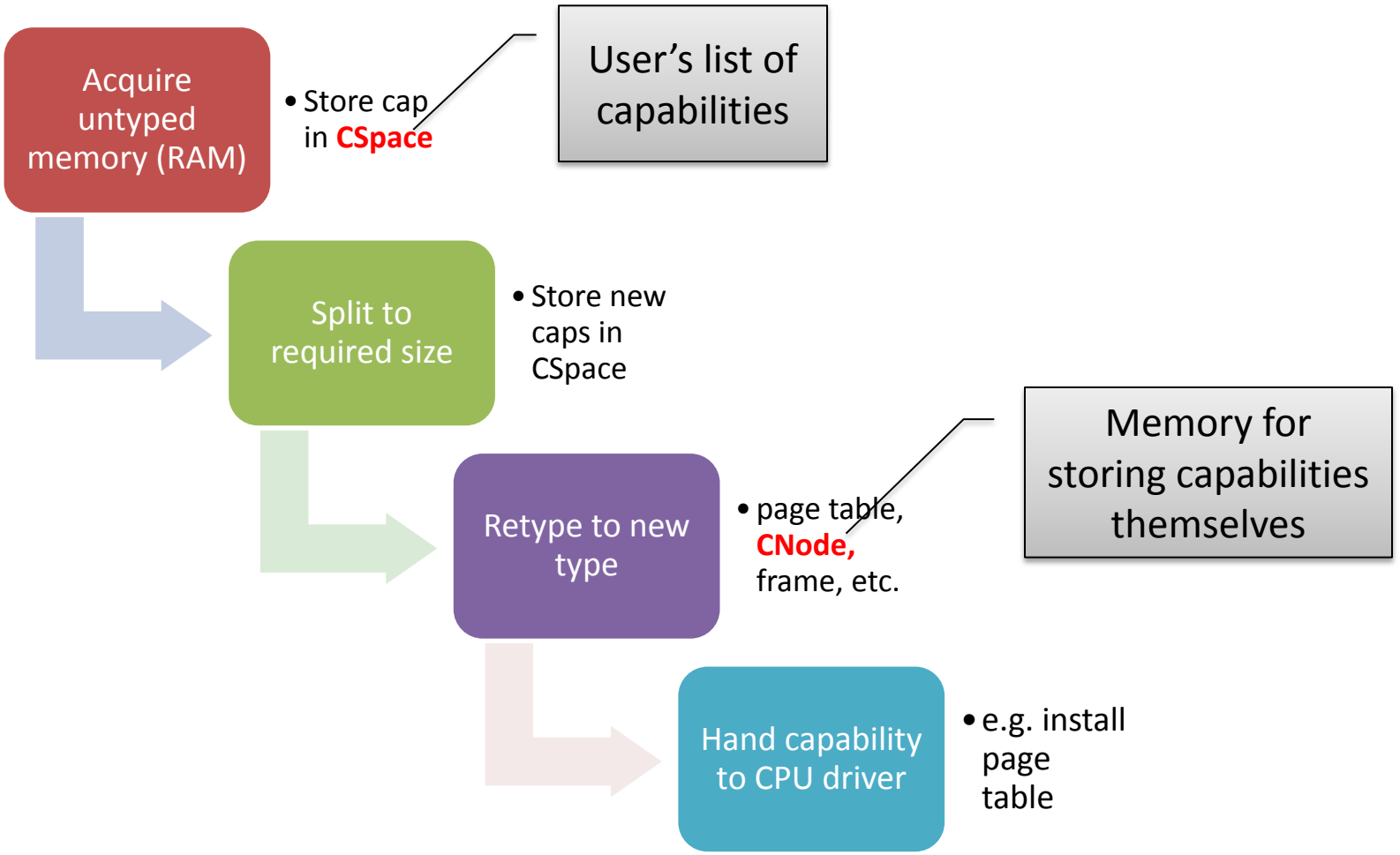


Allocation and management of physical memory

- Problem:
  - Most kernels dynamically allocate memory.
  - What do you do when it runs out?
- seL4 model, extended in Barrelfish:
  - All memory initially untyped
  - Users control allocation via capabilities

$\Rightarrow$ No dynamic allocation performed in kernel!

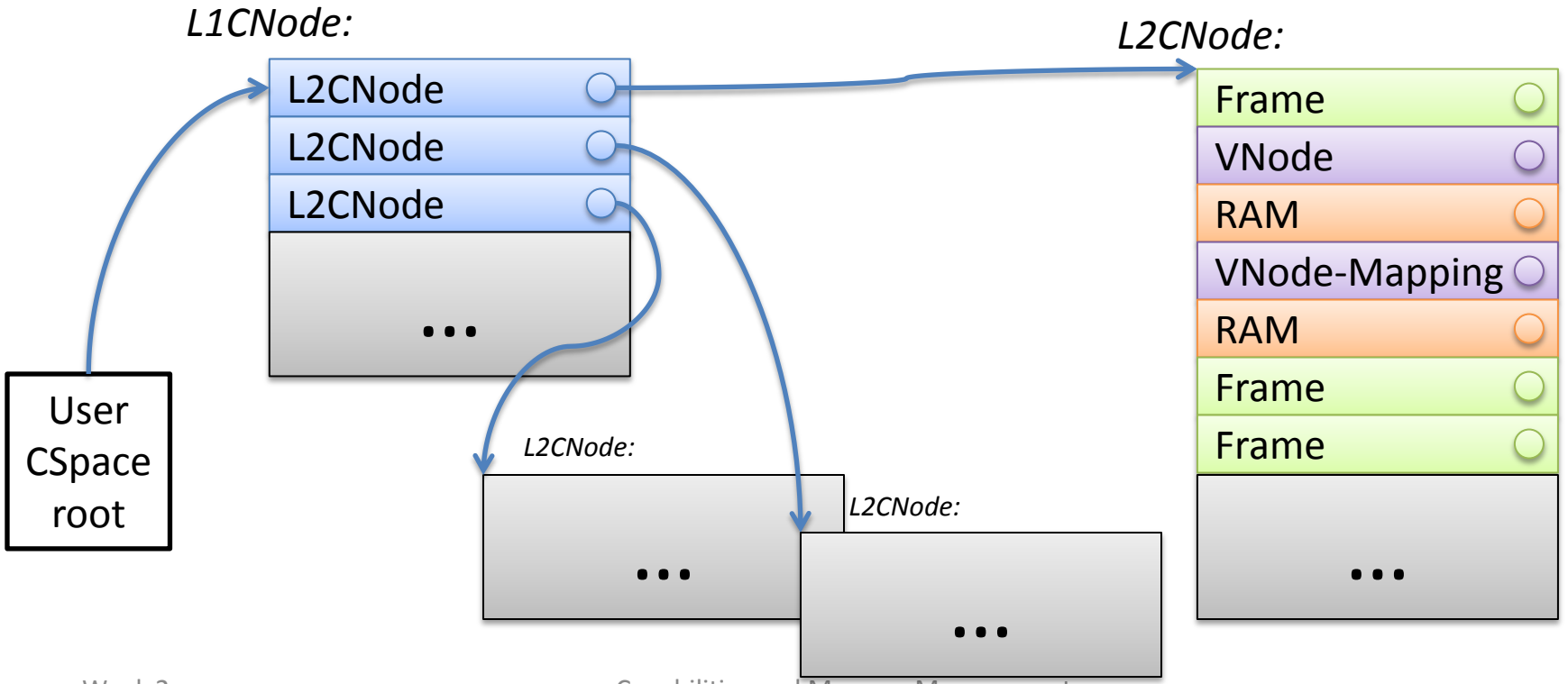
How it works


Acquire untyped memory (RAM)
• Store cap in CSpace
User's list of capabilities
Split to required size
• Store new caps in CSpace
Retype to new type
• page table, CNode, frame, etc.
Memory for storing capabilities themselves
Hand capability to CPU driver
• e.g. install page table

CSpace: a two-level table

Data is in physical memory – not mapped!

Referred to with *capref*: like a virtual address in CSpace



Capabilities in Barrelfish

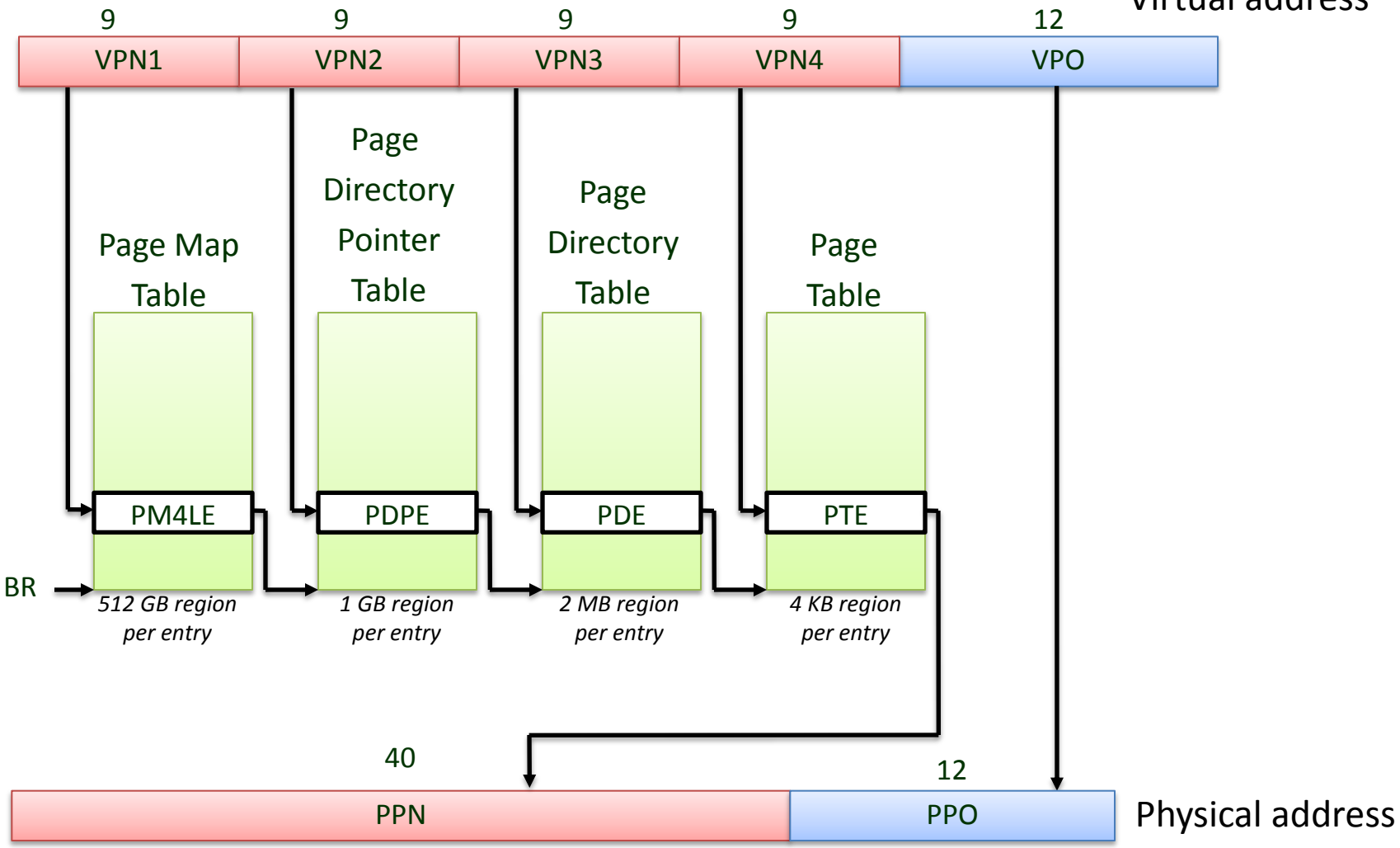
- Each type supports a set of operations:
  - Think of them a bit like object references
  - Most “system calls” actually capability operations
- Restrictions on operations enforce invariants
  - E.g. It should be impossible to construct a bad page table from user space

CREATING PAGE TABLES IN USER SPACE

Example: x86-64 paging

(you'll be doing the same with ARM)

Virtual address



Physical address

Capability operations

Capability	Type	Size
$Capref_a$	`VNode_x86_64_ptable`	4kB
$Capref_b$	`Frame`	4kB

$Capref_a \rightarrow$`install(`$Capref_b$, *slot* , *flags*`)`

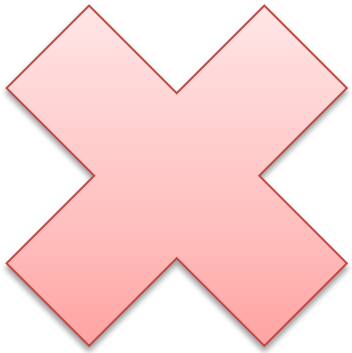
Map a 4kB page frame in the page table page

Which entry (0-511)

Capability operations

Capability	Type	Size
$Capref_a$	`VNode_x86_64_ptable`	4kB
$Capref_b$	**`VNode_x86_64_pdpt`**	4kB

$Capref_a \rightarrow \texttt{install}(Capref_b, slot, flags)$



Capability operations

Capability	Type	Size
$Capref_a$	`VNode_x86_64_ptable`	4kB
$Capref_b$	`Frame`	**16kB**

$Capref_a \rightarrow \texttt{install}(Capref_b, slot, flags)$

May work, depending on existing mappings and alignment

Capability operations

Capability	Type	Size
$Capref_a$	**`VNode_x86_64_pdir`**	4kB
$Capref_b$	`Frame`	**2MB**

$Capref_a \rightarrow \texttt{install}(Capref_b, slot, flags)$

Create a large page mapping

Capability operations

Capability	Type	Size
$Capref_a$	**`VNode_x86_64_pdpt`**	4kB
$Capref_b$	`Frame`	**1GB**

$Capref_a \rightarrow \texttt{install}(Capref_b, slot, flags)$

Create a huge page mapping

Capability operations

Capability	Type	Size
$Capref_a$	**`VNode_x86_64_pml4`**	4kB

$Domain \rightarrow \texttt{switch}(Capref_a)$

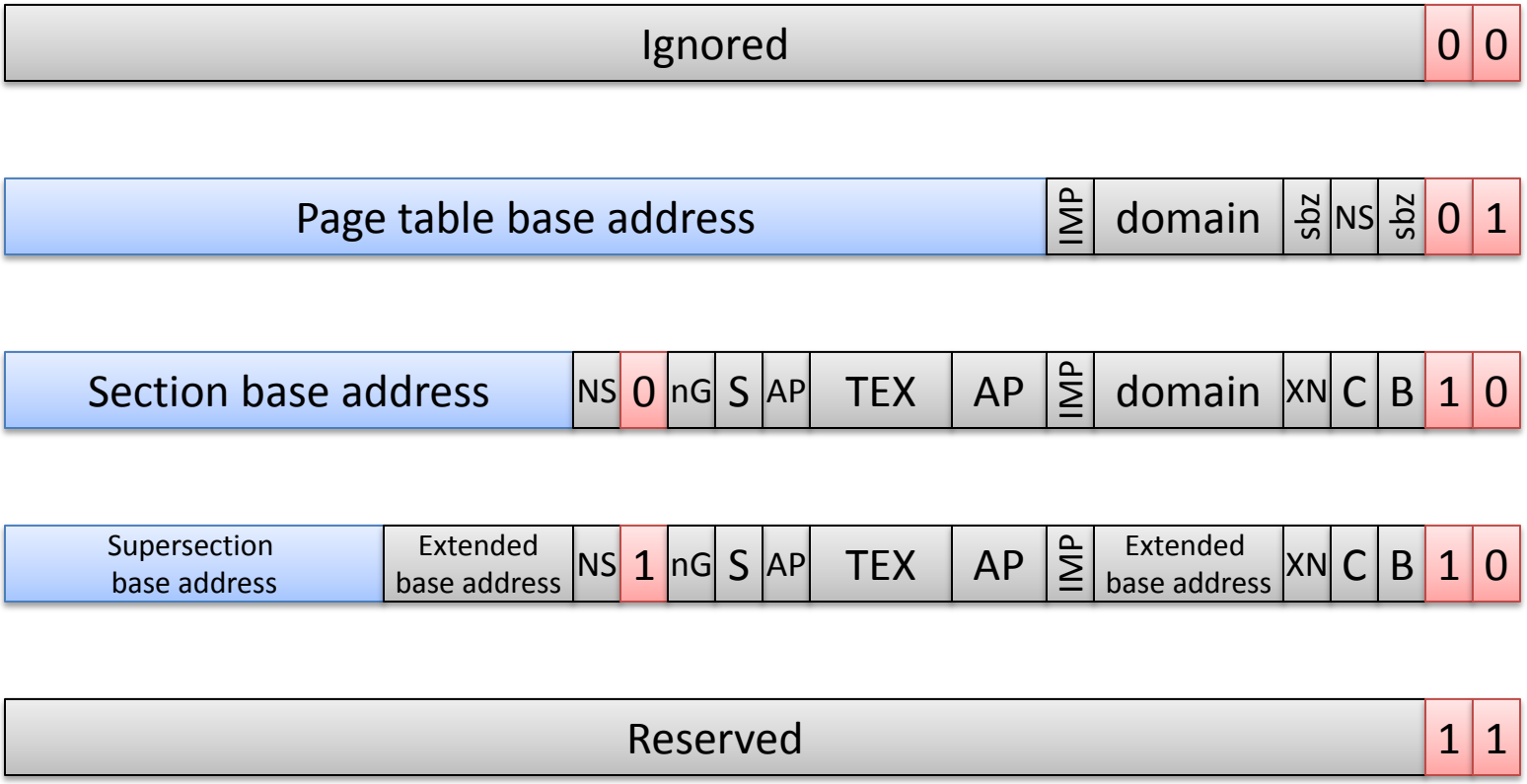
Install new page table as our virtual address space

ARMV7-A MEMORY MANAGEMENT HARDWARE

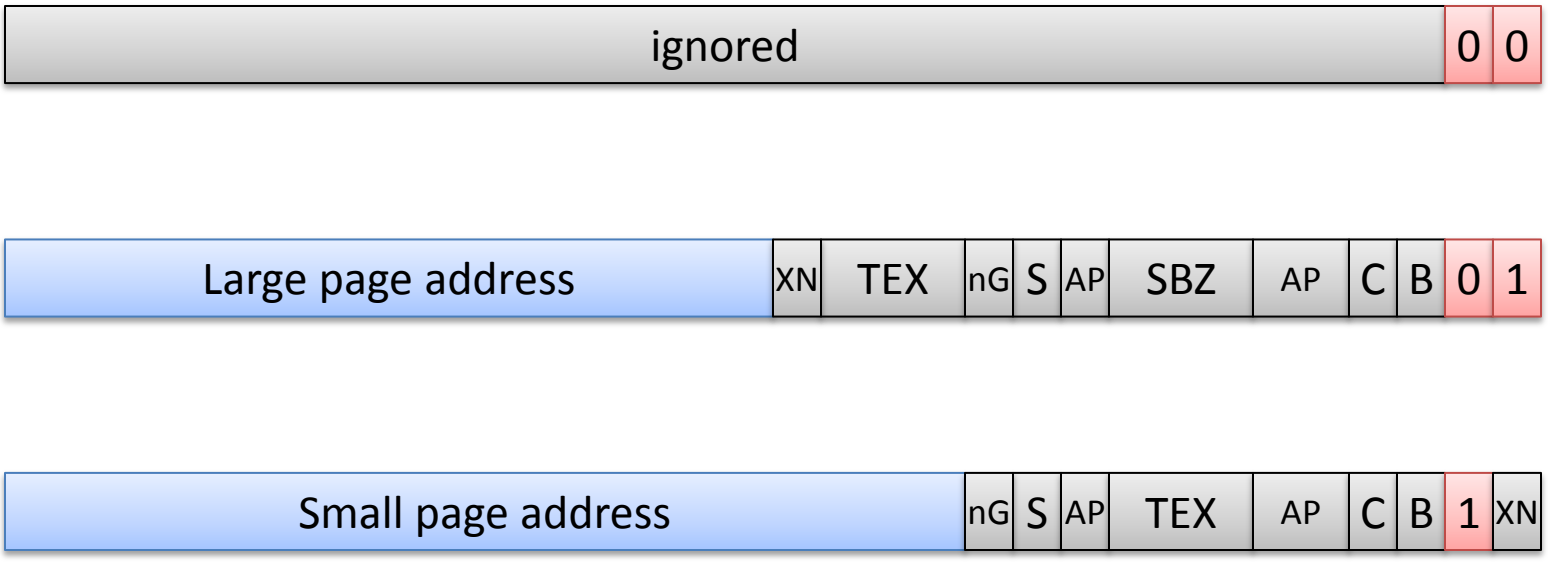
ARMv7-A page tables

- 32b, two-level tables:
  - L1 (16Kib in size) maps sections and points to L2
  - L2 maps pages
- Large pages:
  - L1: Sections (1MiB), Supersections (16MiB)
  - L2: Large pages (64kiB), Small pages (4kiB)

ARMv7-A L1 Format

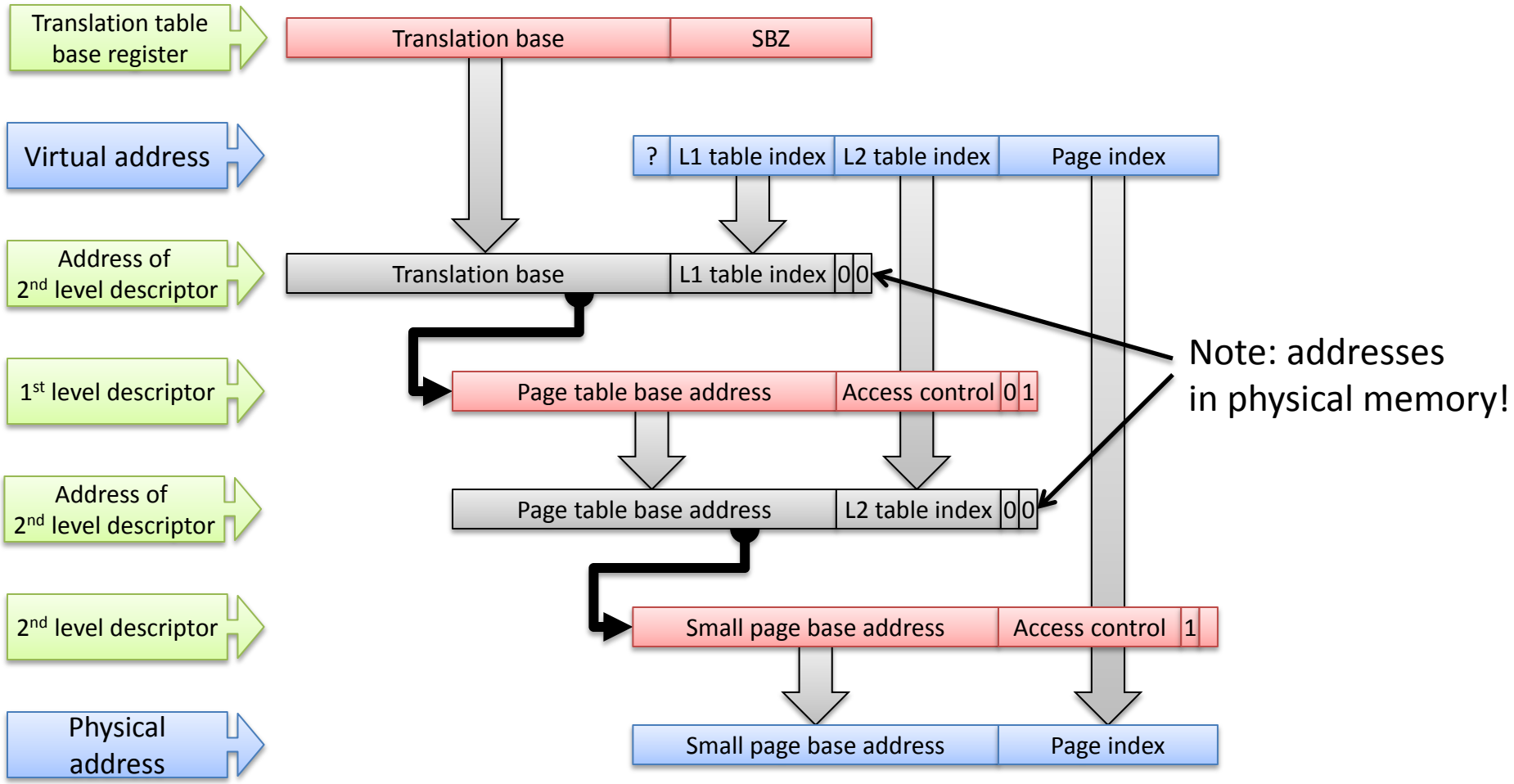


ARMv7-A L2 Format

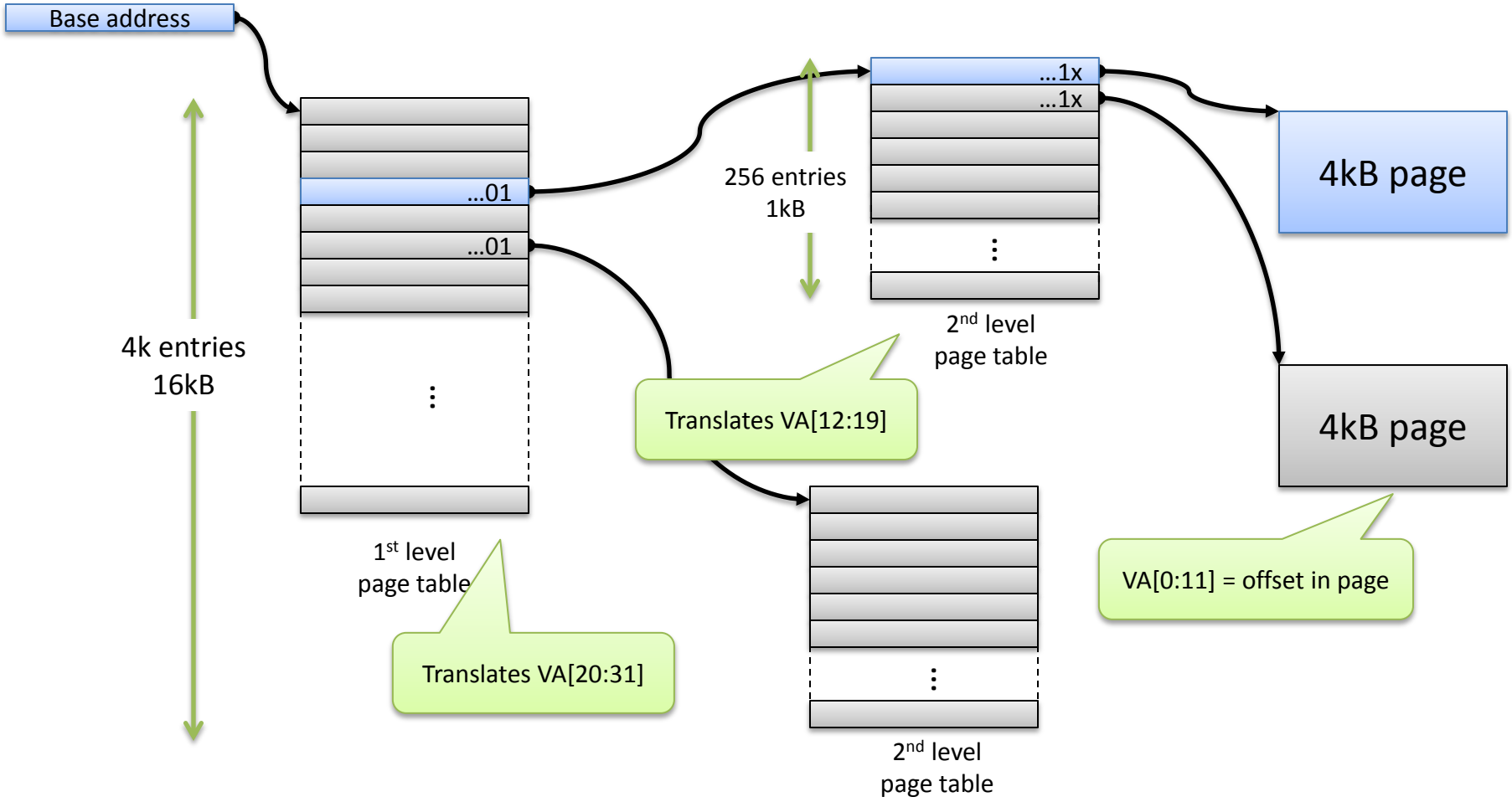


Note: no hardware-managed dirty or accessed bits!

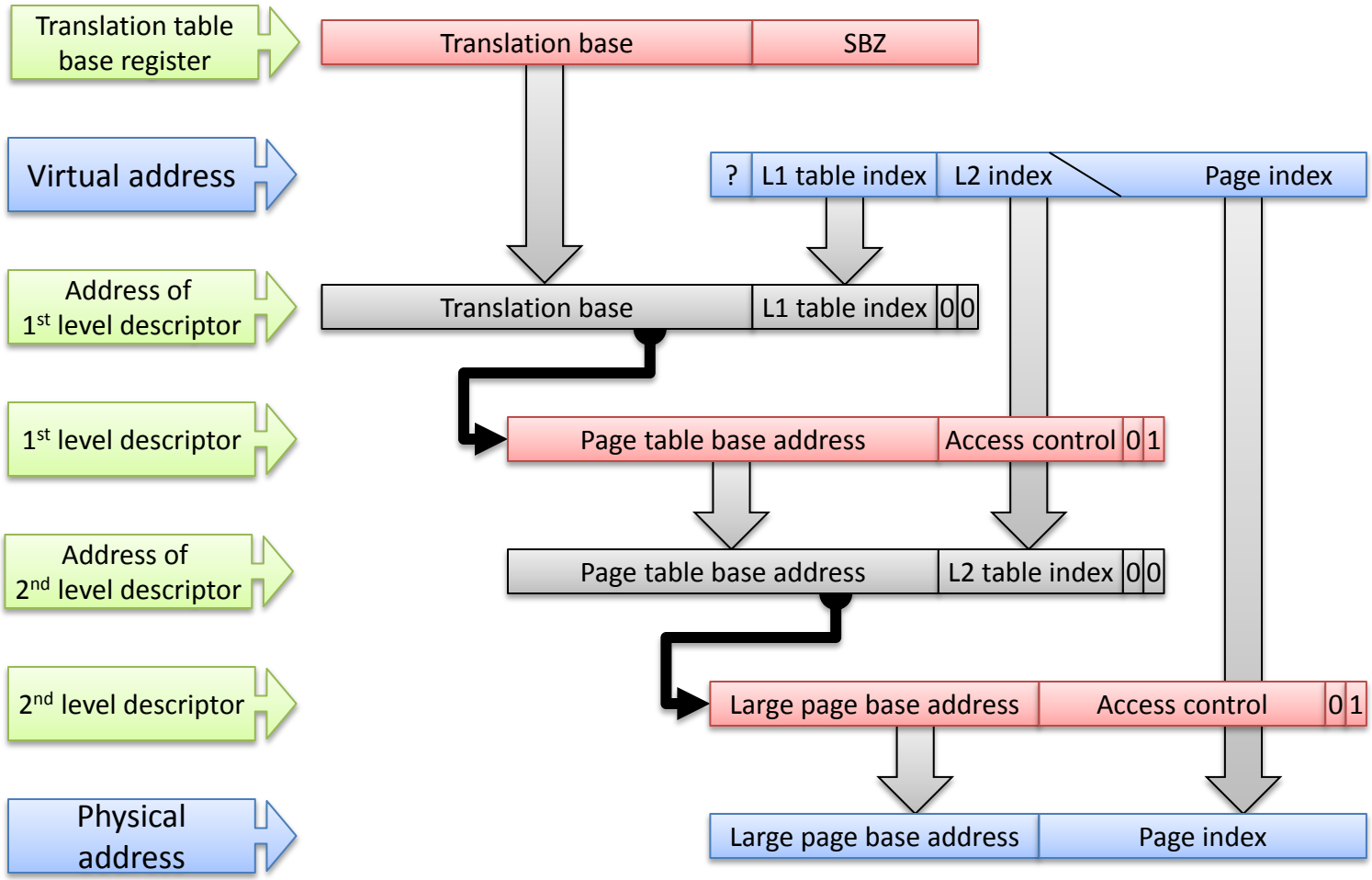
Small page translation



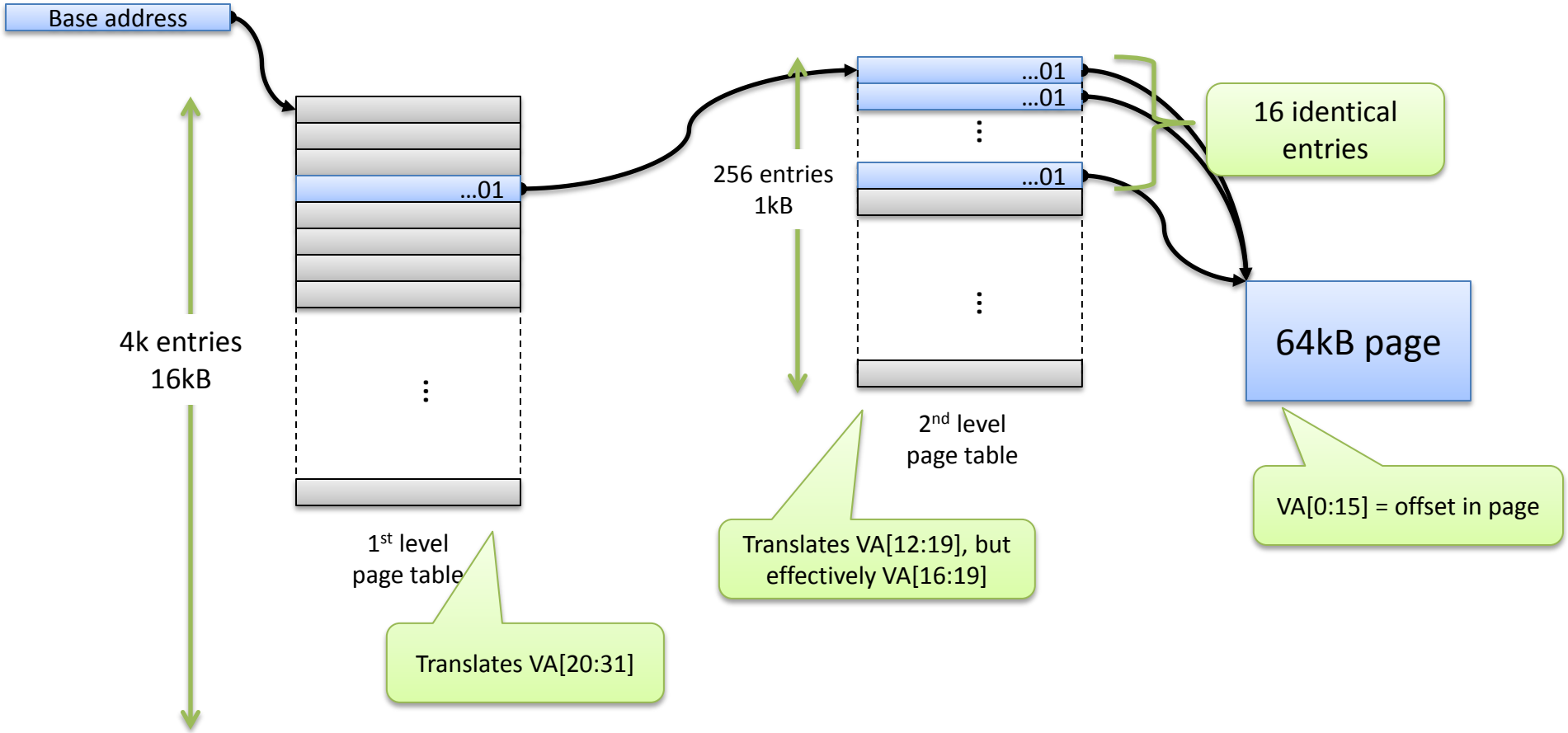
Page table for small pages



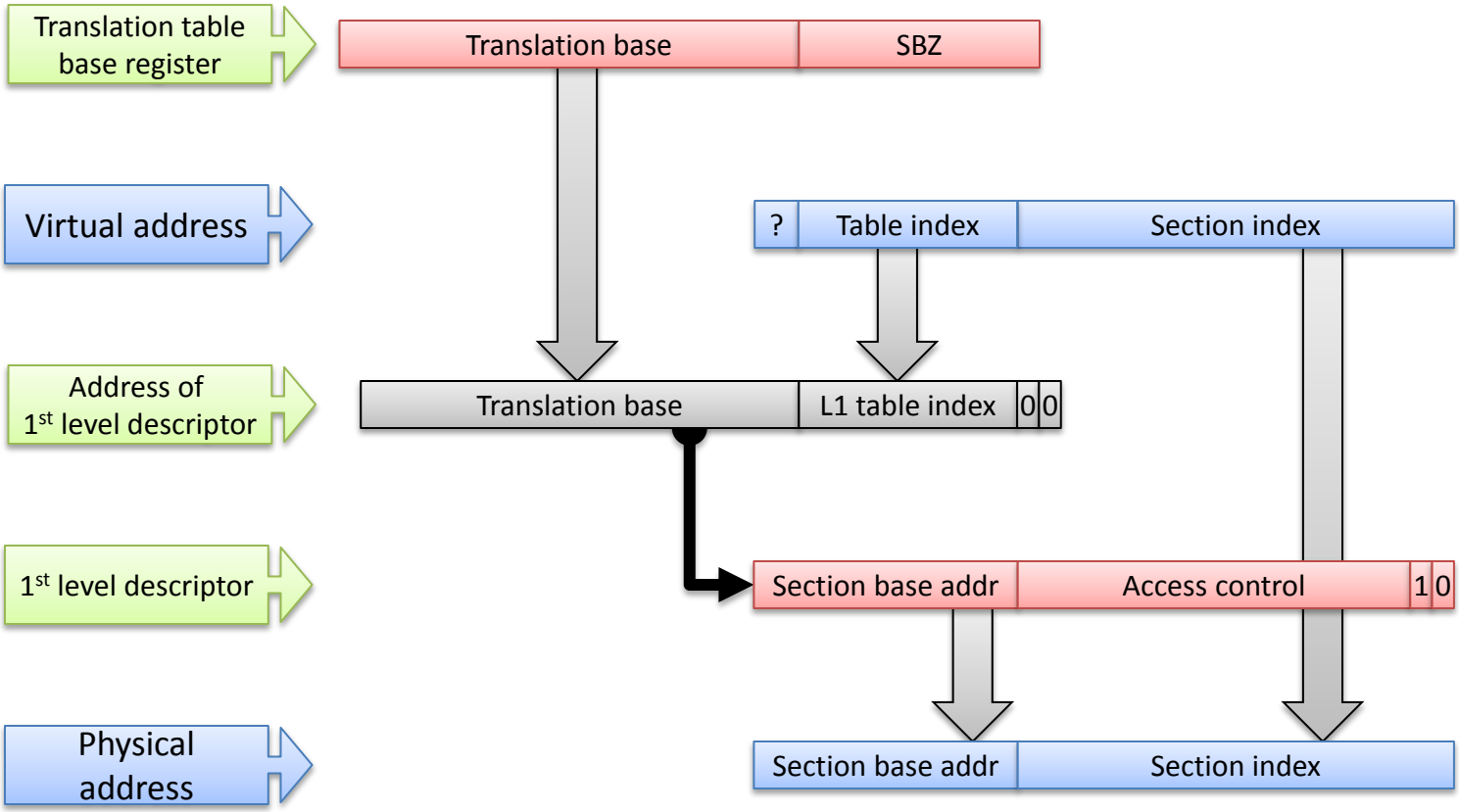
Large page translation



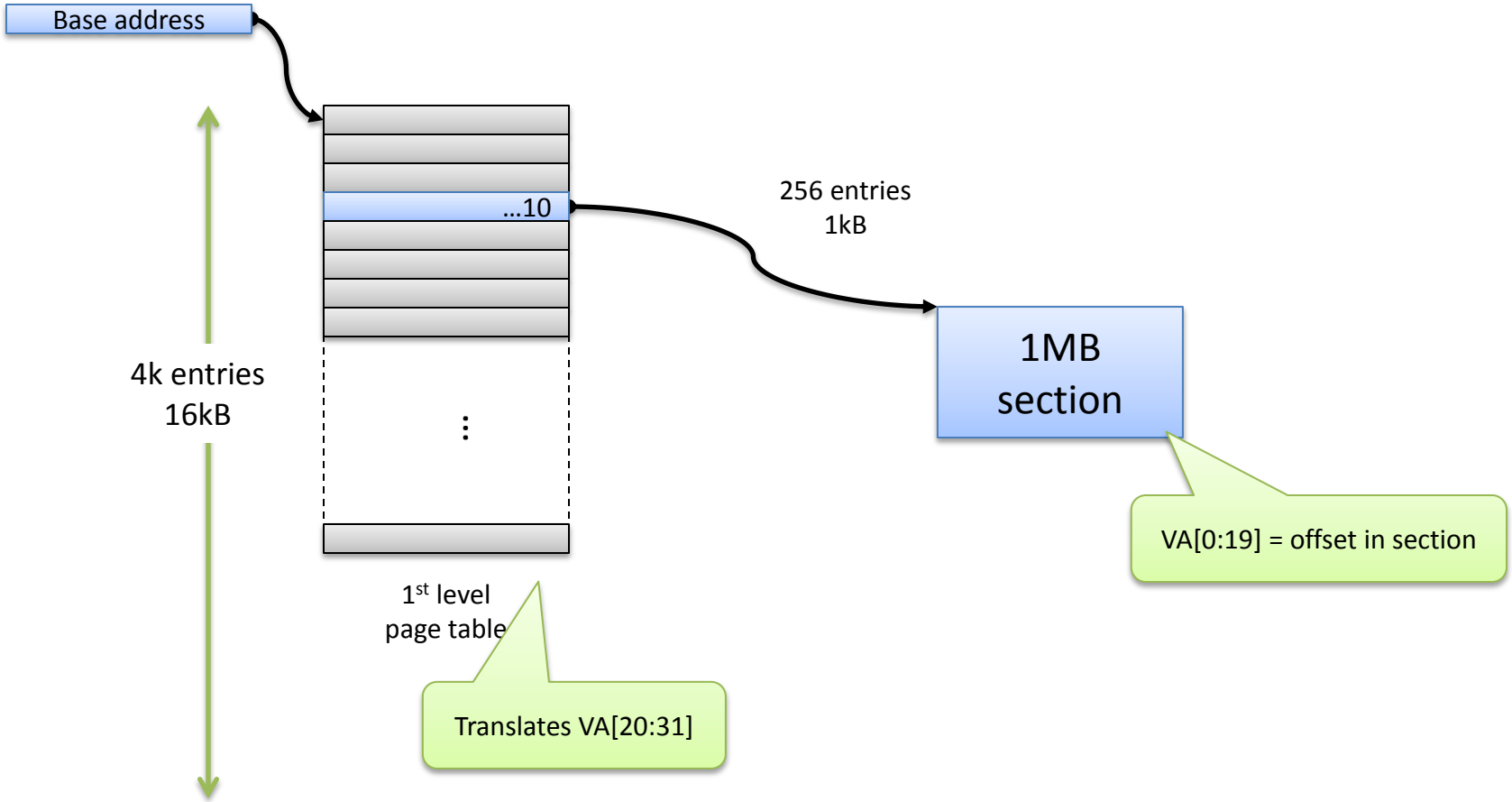
Page table for large pages



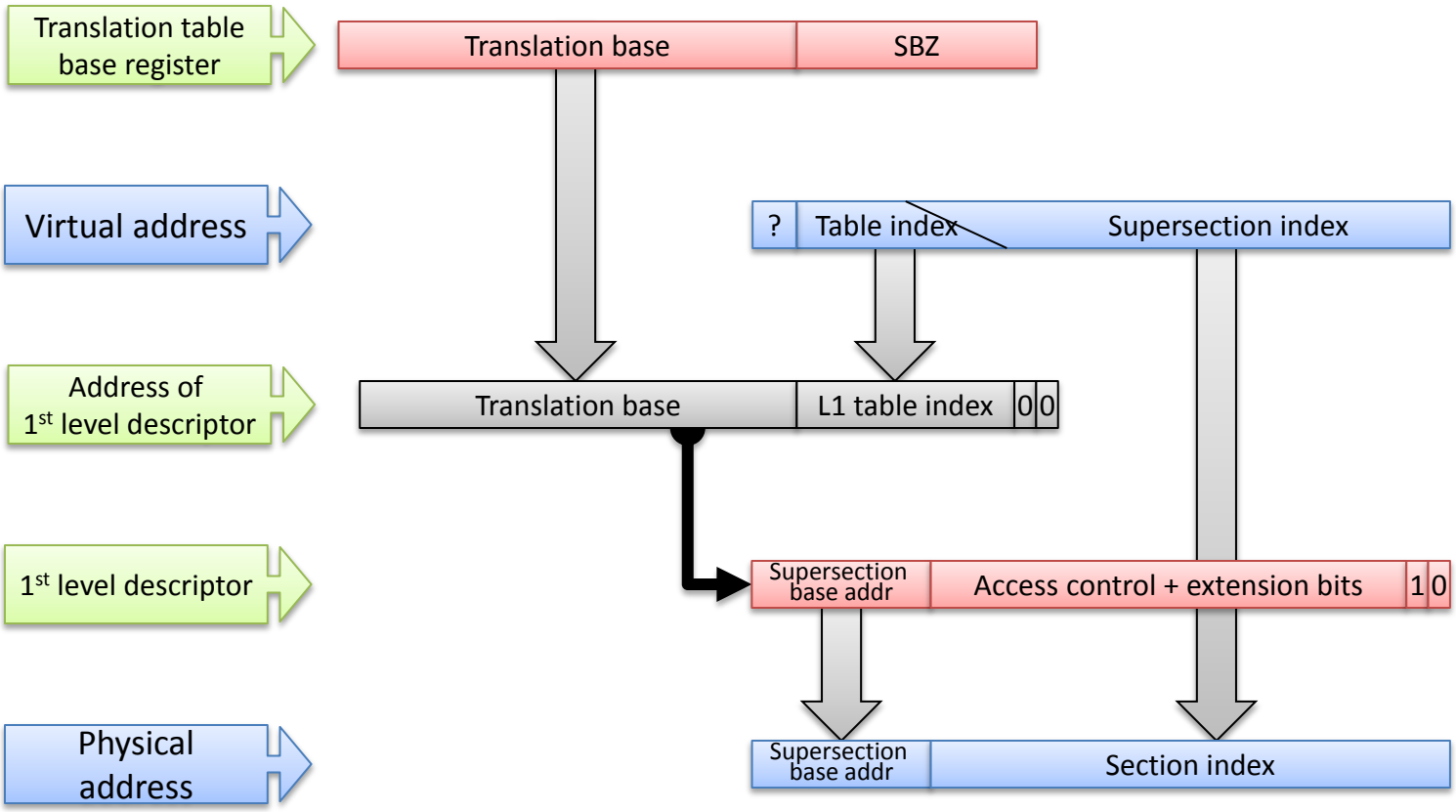
Section translation



Page table for sections



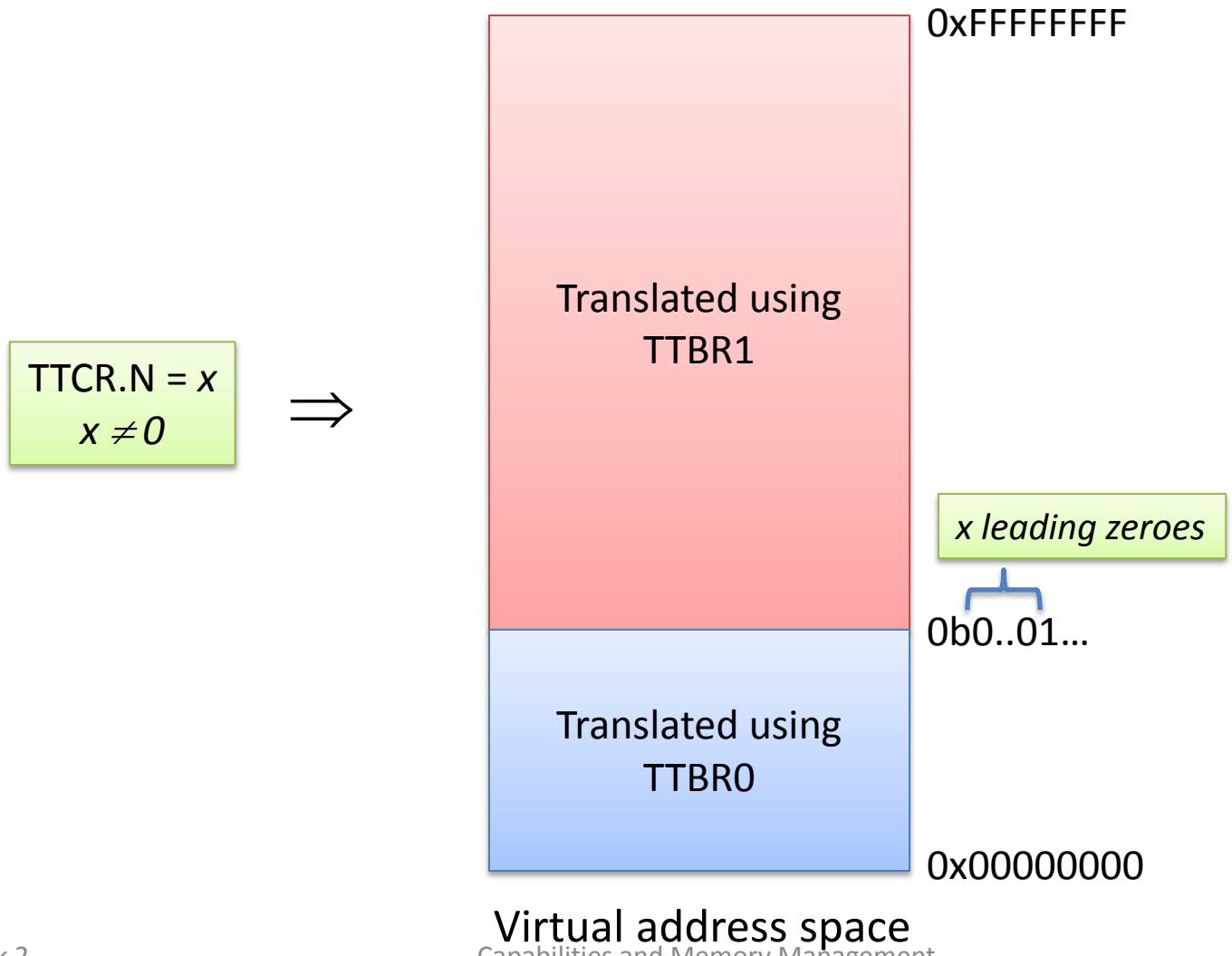
Supersection translation



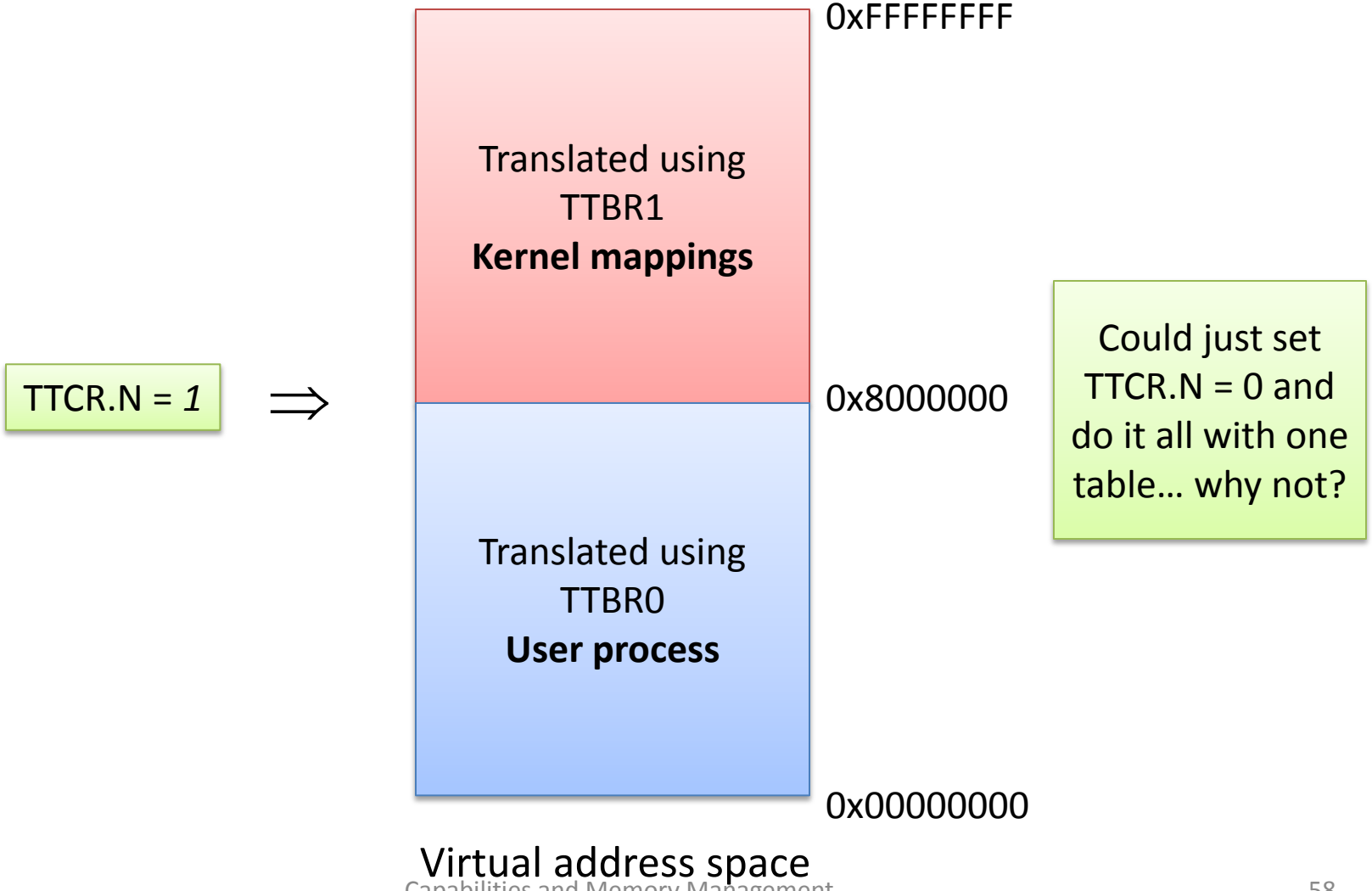
Where is the page table?

- Two page table base registers:
  - TTBR0: low addresses (MSBs == 0)
  - TTBR1: everything else
- Which one do we use?
  - TTBRC.N indicates how big the TTBR0 table is
  - N = x $\Rightarrow$ TTBR0 is used if top x bits of VA are zero
  - N = 0 $\Rightarrow$ TTBR0 is used for everything

Split virtual address space



Example Barrelfish virtual address space



FURTHER READING

Further reading

- Norm Hardy. 1988. The Confused Deputy: (or why capabilities might have been invented). SIGOPS Oper. Syst. Rev. 22, 4 (October 1988), 36-38.
- Henry M. Levy. 1984. Capability-Based Computer Systems. Digital Press. http://homes.cs.washington.edu/~levy/capabook/
- Norman Hardy. 1985. KeyKOS architecture. SIGOPS Oper. Syst. Rev. 19, 4 (October 1985), 8-25. http://dx.doi.org/10.1145/858336.858337
- Dhammika Elkaduwe, Philip Derrin, and Kevin Elphinstone. 2008. Kernel design for isolation and assurance of physical memory. In Proceedings of the 1st workshop on Isolation and integration in embedded systems (IIES '08), Michael Engel and Olaf Spinczyk (Eds.). ACM, New York, NY, USA, 35-40. http://dx.doi.org/10.1145/1435458.1435465
- Katelin Bailey, Luis Ceze, Steven D. Gribble, and Henry M. Levy. 2011. Operating system implications of fast, cheap, non-volatile memory. In Proceedings of the 13th USENIX conference on Hot topics in operating systems (HotOS'13). USENIX Association, Berkeley, CA, USA, 2-2. http://static.usenix.org/event/hotos11/tech/final_files/Bailey.pdf
- Wilkes, M. V.; Needham, R. M. (1979). The Cambridge CAP Computer and Its Operating System (PDF). New York: Elsevier North Holland. http://research.microsoft.com/pubs/72418/cap.pdf