Reasoning about Paging Data Structure Walks on x86-64 Machines

Shilpi Goel
shigoel@cs.utexas.edu

ACL2 Seminar
3rd March, 2015

Goals of this talk

- Explain some x86 memory management terminology
- Present a part of my effort to enable reasoning about system-level x86 programs
- Feedback

Goals of this talk

- Explain some x86 memory management terminology
- Present a part of my effort to enable reasoning about system-level x86 programs
- Feedback

Disclaimer: I'm presenting ongoing work, and there are more than a few rough edges here.

Outline

- **Background**
- Description of paging
- Proving a memory RoW theorem in the context of paging
- Challenges
- Future Work and Conclusion

Background

- **Physical (main) memory** is the memory that the processor addresses on its bus.
- System programs offer a simpler memory interface (**linear memory**) to application programs.

Background

- **Physical (main) memory** is the memory that the processor addresses on its bus.
- System programs offer a simpler memory interface (**linear memory**) to application programs.
- Application program verification can be done at the level of linear address space.

Background

- **Physical (main) memory** is the memory that the processor addresses on its bus.
- System programs offer a simpler memory interface (**linear memory**) to application programs.
- Application program verification can be done at the level of linear address space.
- Verification of system programs must necessarily be done at the level of physical address space.

Background (contd.)

- On x86-64 machines, memory management via **paging is always enabled**, and 64-bit code cannot directly access physical memory.
- Reasoning at the level of physical memory requires **reasoning about the address translations** performed by the paging mechanism.

Background (contd.)

- On x86-64 machines, memory management via **paging is always enabled**, and 64-bit code cannot directly access physical memory.
- Reasoning at the level of physical memory requires **reasoning about the address translations** performed by the paging mechanism.
- Every linear memory address needs to be translated to a physical address by "walking" **paging data structures**.

Background (contd.)

- On x86-64 machines, memory management via **paging is always enabled**, and 64-bit code cannot directly access physical memory.
- Reasoning at the level of physical memory requires **reasoning about the address translations** performed by the paging mechanism.
- Every linear memory address needs to be translated to a physical address by "walking" **paging data structures**.
- This greatly **complicates proofs** of theorems like **memory read-over-write** that are otherwise simple in the context of linear address space.

Reasoning about Updates to Data Structures

Rockwell Challenge to ACL2 users (2002): "Dynamic Datastructures in ACL2"

- Reasoning about complex and **pointer-rich data structures** embedded in a linear address space
- Called for efficient solutions for proving **non-interference** properties of data structures
  - Does the proof scale quadratically with the number of entries in the data structure? Can we do better?

Some Solutions to the Rockwell Challenge

1. Memory Taggings (J Moore)
2. Address Enumeration (David Greve)
    - Multisets/bags library (Eric Smith et al.)
3. Separating data structure traversals from modifications (Hanbing Liu)

This work is similar to (2) and (3).

Outline

- Background
- **Description of paging**
- Proving a memory RoW theorem in the context of paging
- Challenges
- Future Work and Conclusion

Paging

- Linear address space is divided into pages; an OS tracks these pages via **hierarchical data structures**.

Paging

- Linear address space is divided into pages; an OS tracks these pages via **hierarchical data structures**.

- For every linear memory access, these data structures are “walked” to obtain the translation to the corresponding physical address.

Paging

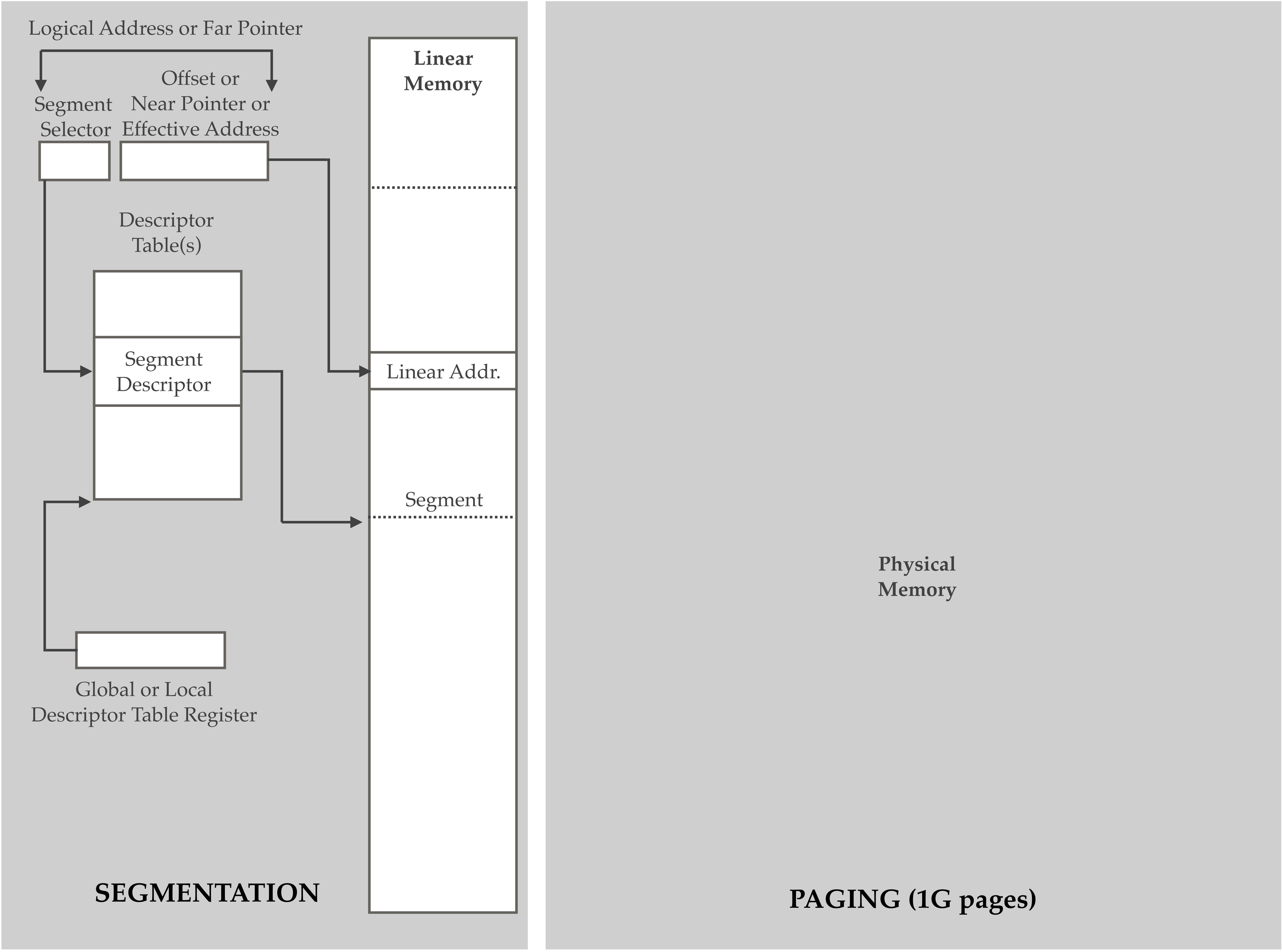
- Linear address space is divided into pages; an OS tracks these pages via **hierarchical data structures**.

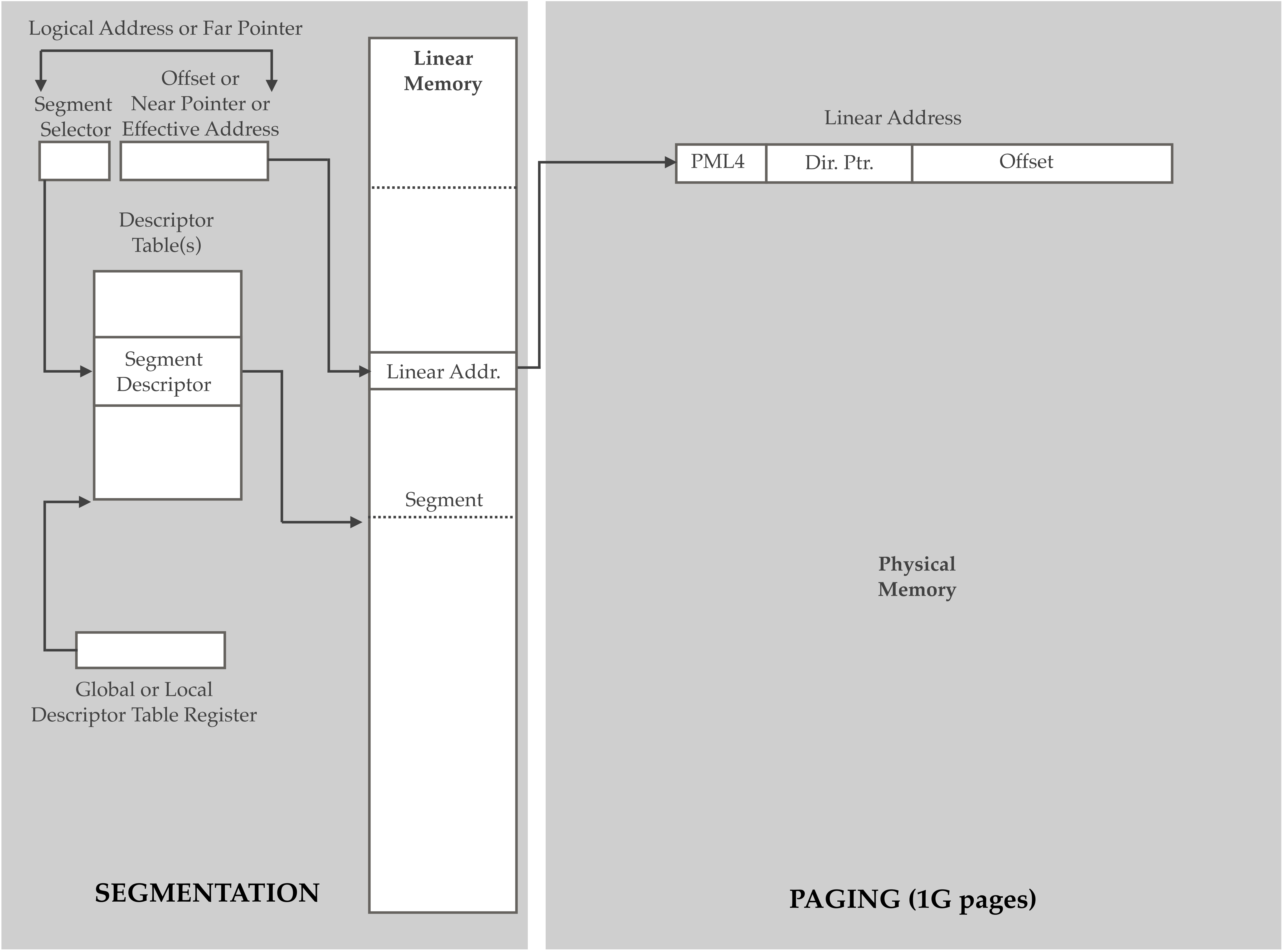
- For every linear memory access, these data structures are “walked” to obtain the translation to the corresponding physical address.

- Besides address translation, paging data structures determine the **access rights** for each translation.

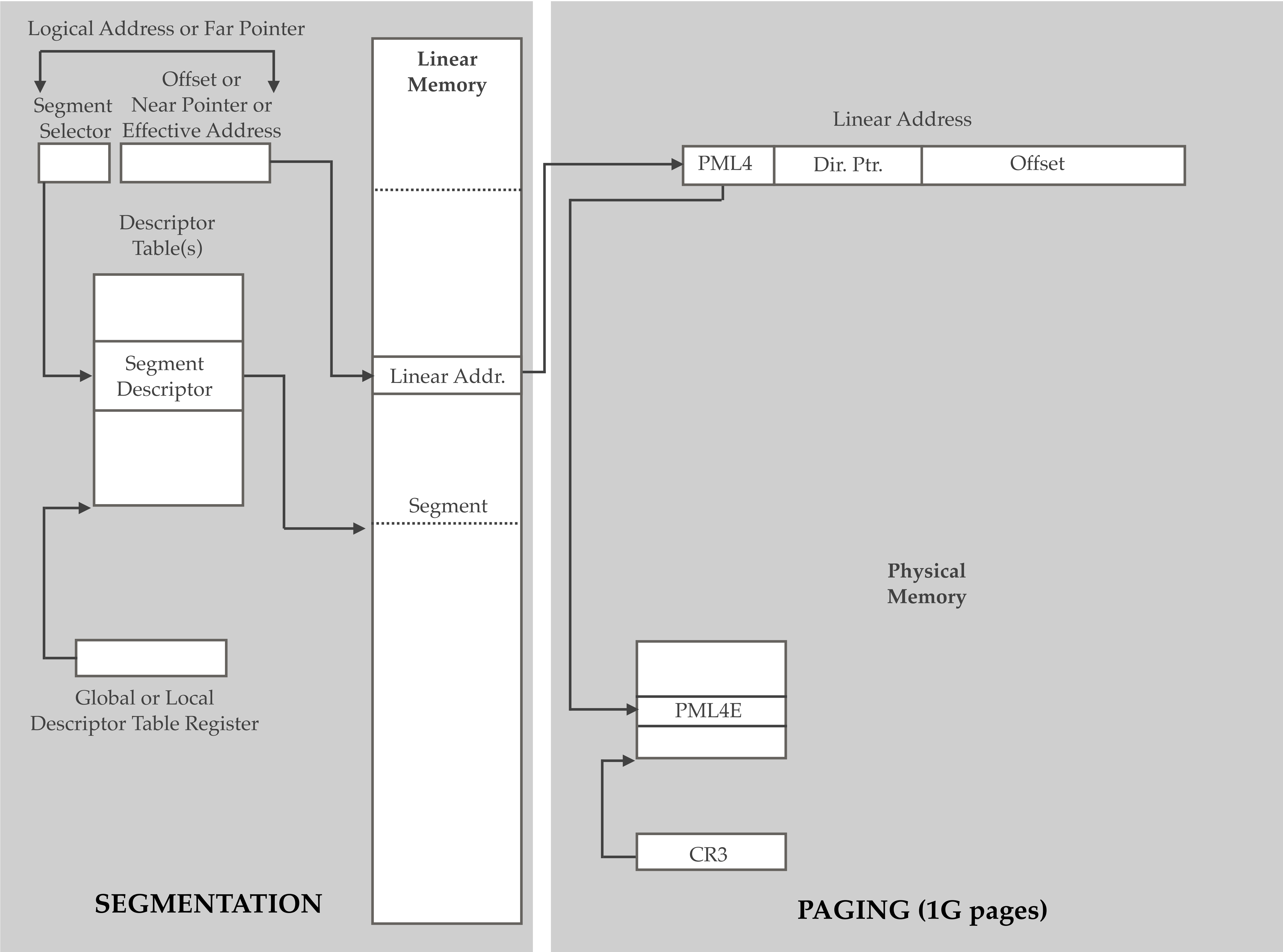
Paging

- Linear address space is divided into pages; an OS tracks these pages via **hierarchical data structures**.
- For every linear memory access, these data structures are "walked" to obtain the translation to the corresponding physical address.
- Besides address translation, paging data structures determine the **access rights** for each translation.
- A **page-fault exception** is generated:
  - if the required page is located in secondary storage.
  - the access rights do not permit the access.

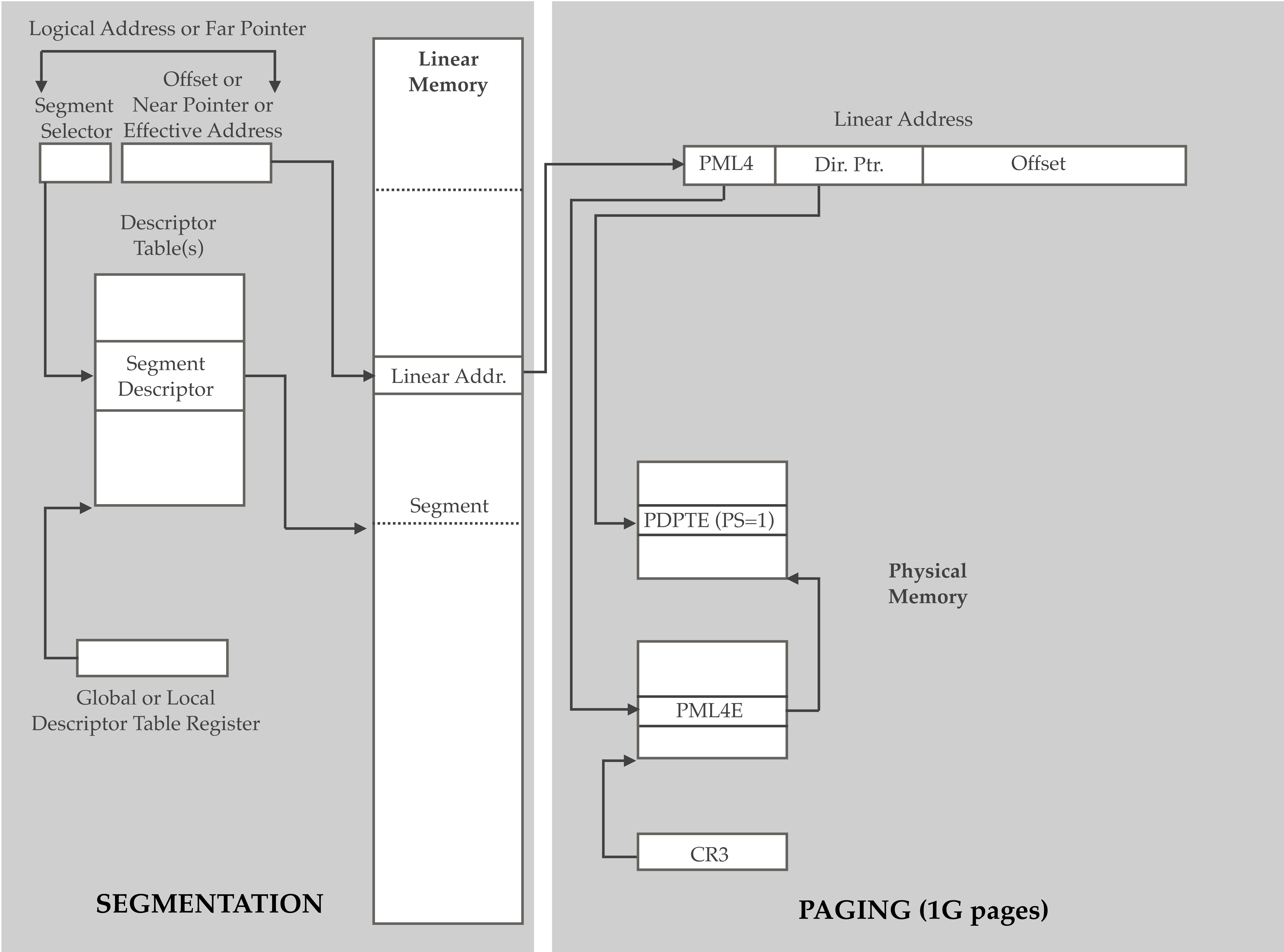

Logical Address or Far Pointer
Segment Selector
Offset or Near Pointer or Effective Address
Descriptor Table(s)
Segment Descriptor
Global or Local Descriptor Table Register
Linear Memory
Linear Addr.
Segment
SEGMENTATION
Physical Memory
PAGING (1G pages)

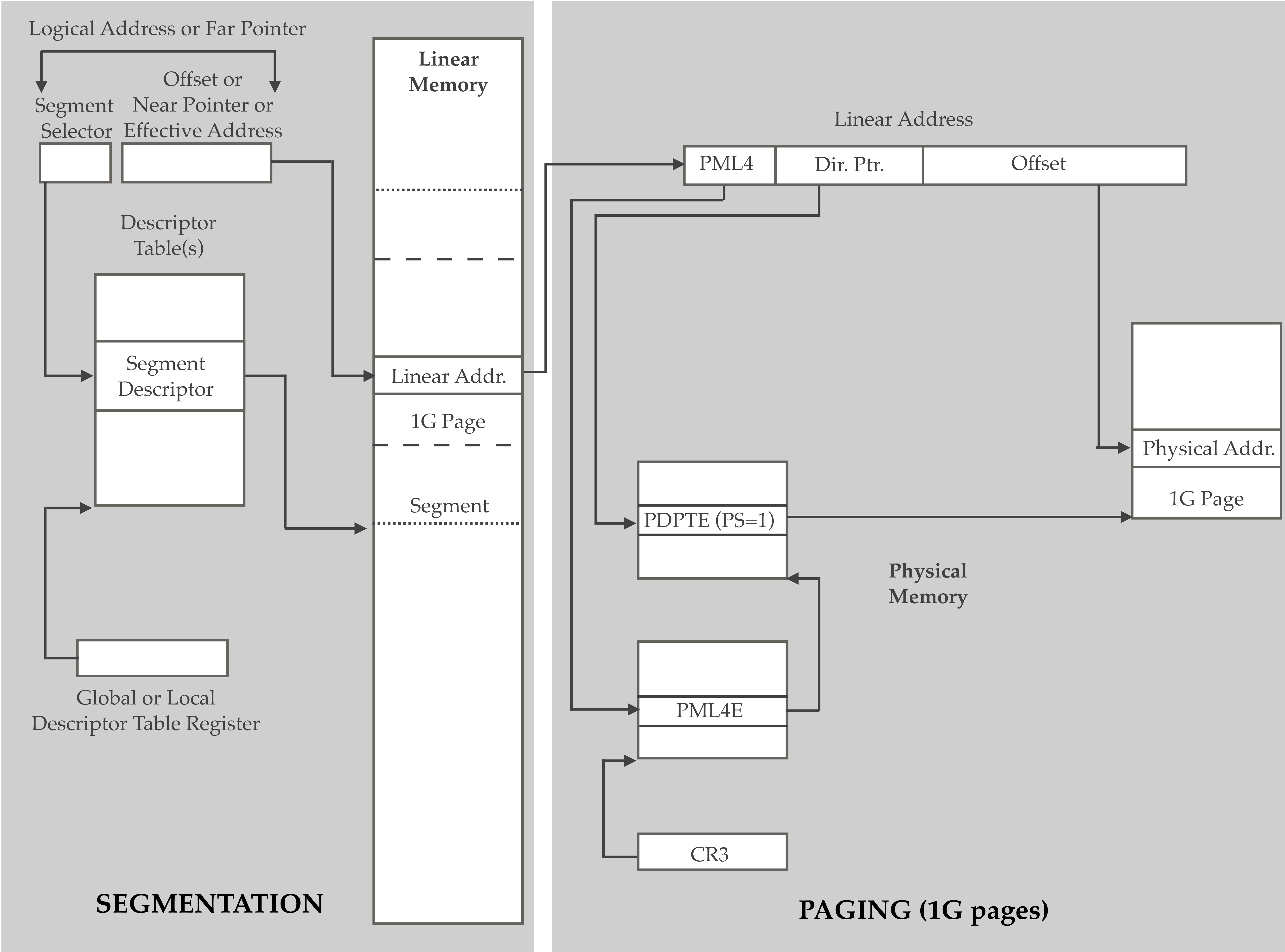


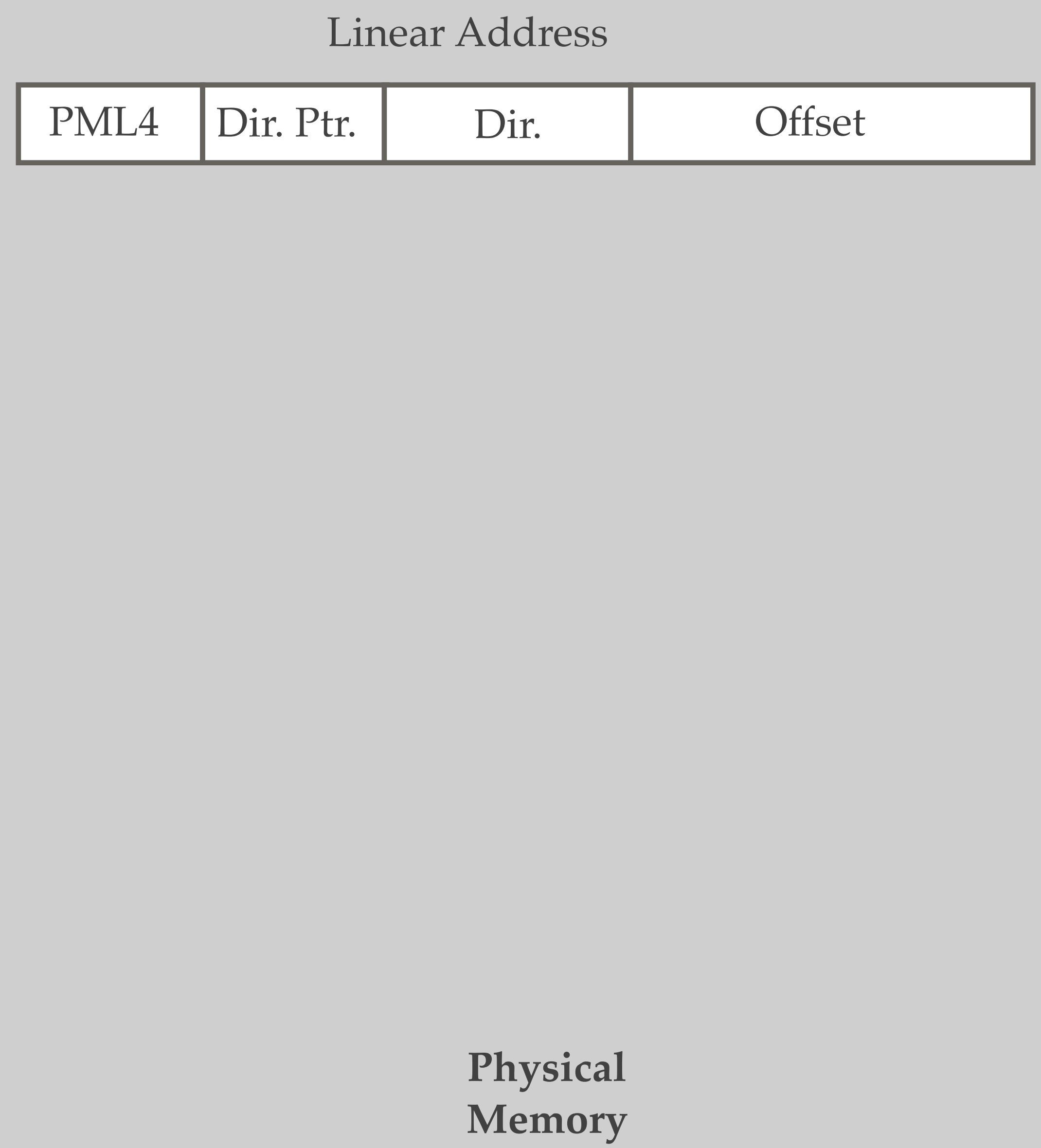
Logical Address or Far Pointer
Segment Selector
Offset or Near Pointer or Effective Address
Descriptor Table(s)
Segment Descriptor
Global or Local Descriptor Table Register
Linear Memory
Linear Addr.
Segment
SEGMENTATION
Linear Address
PML4
Dir. Ptr.
Offset
Physical Memory
PAGING (1G pages)



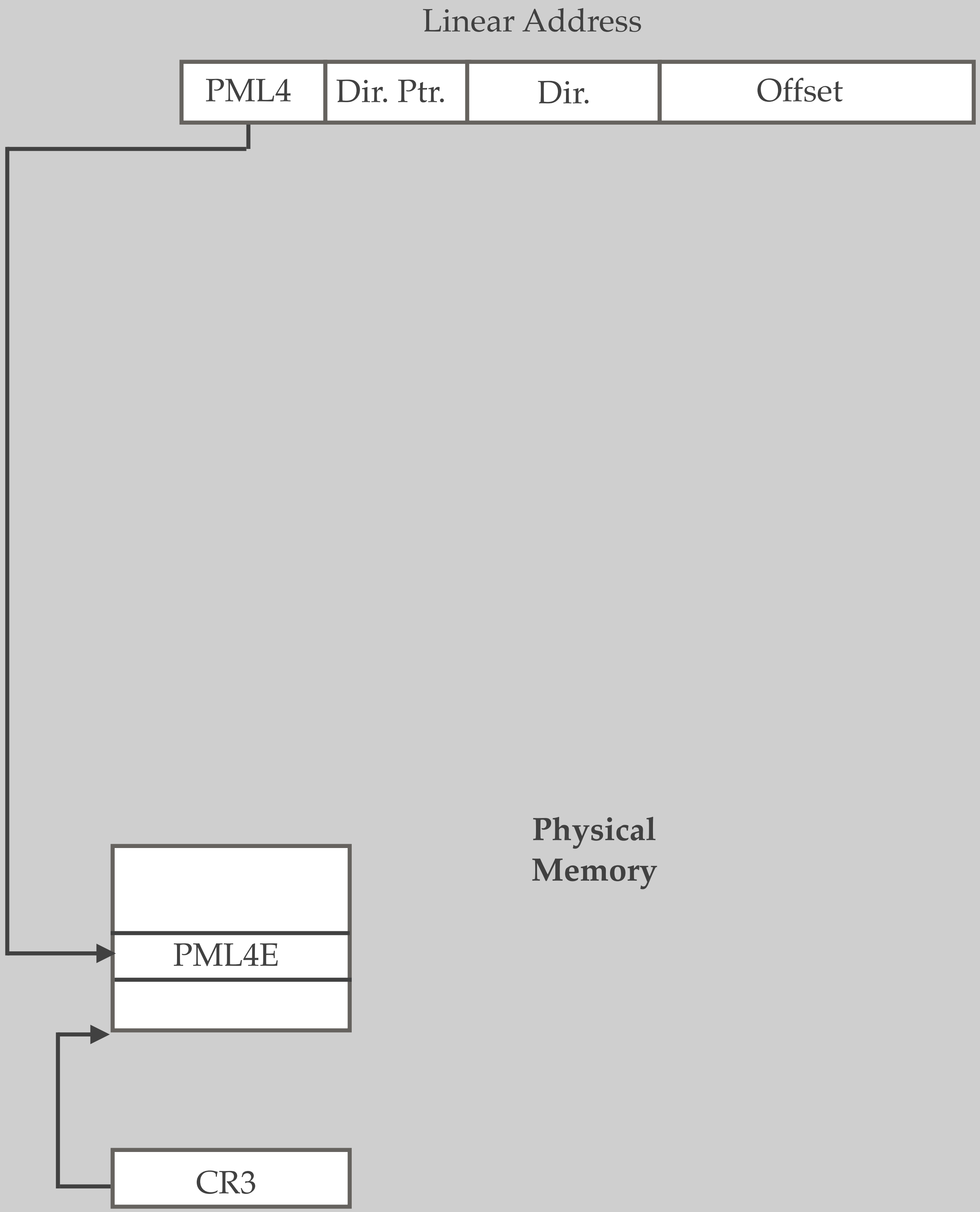
Logical Address or Far Pointer
Segment Selector
Offset or Near Pointer or Effective Address
Descriptor Table(s)
Segment Descriptor
Global or Local Descriptor Table Register
Linear Memory
Linear Addr.
Segment
SEGMENTATION
Linear Address
PML4
Dir. Ptr.
Offset
Physical Memory
PML4E
CR3
PAGING (1G pages)


Logical Address or Far Pointer
Segment Selector
Offset or Near Pointer or Effective Address
Linear Memory
Descriptor Table(s)
Segment Descriptor
Linear Addr.
Segment
Global or Local Descriptor Table Register
SEGMENTATION
Linear Address
PML4
Dir. Ptr.
Offset
PDPTE (PS=1)
Physical Memory
PML4E
CR3
PAGING (1G pages)

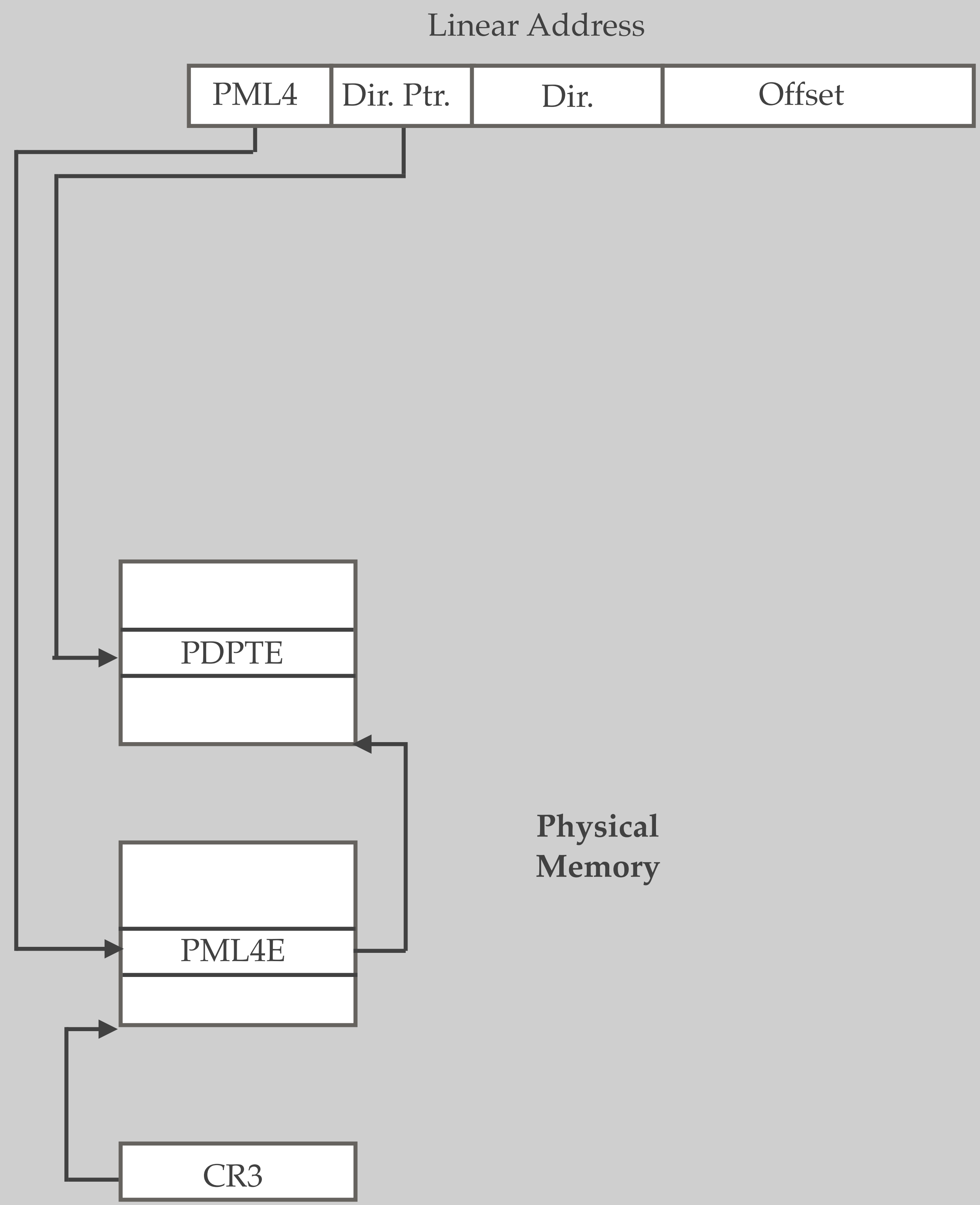

Logical Address or Far Pointer
Segment Selector
Offset or Near Pointer or Effective Address
Descriptor Table(s)
Segment Descriptor
Global or Local Descriptor Table Register
Linear Memory
Linear Addr.
1G Page
Segment
SEGMENTATION
Linear Address
PML4
Dir. Ptr.
Offset
PDPTE (PS=1)
PML4E
CR3
Physical Addr.
1G Page
Physical Memory
PAGING (1G pages)



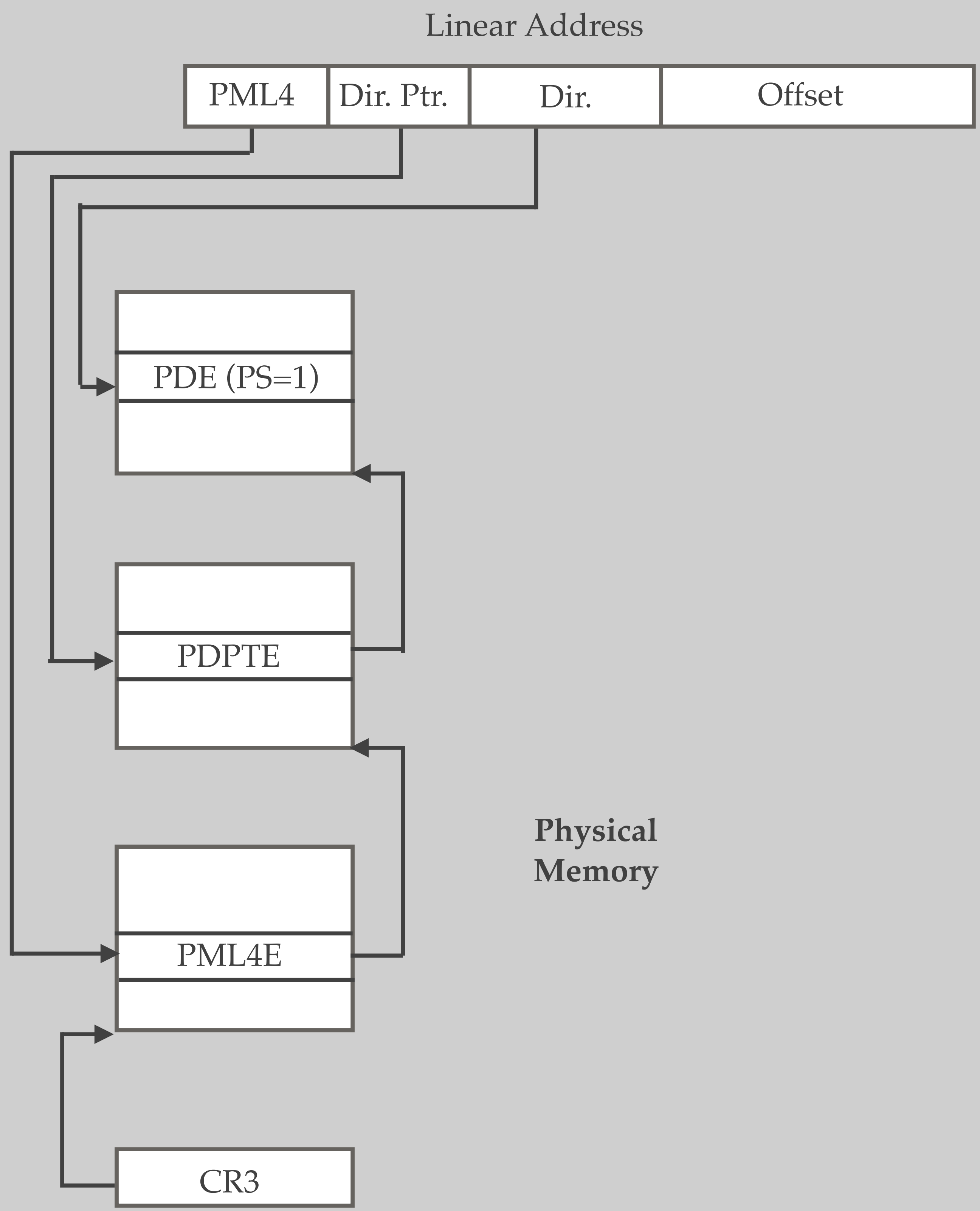
PAGING (2M pages)



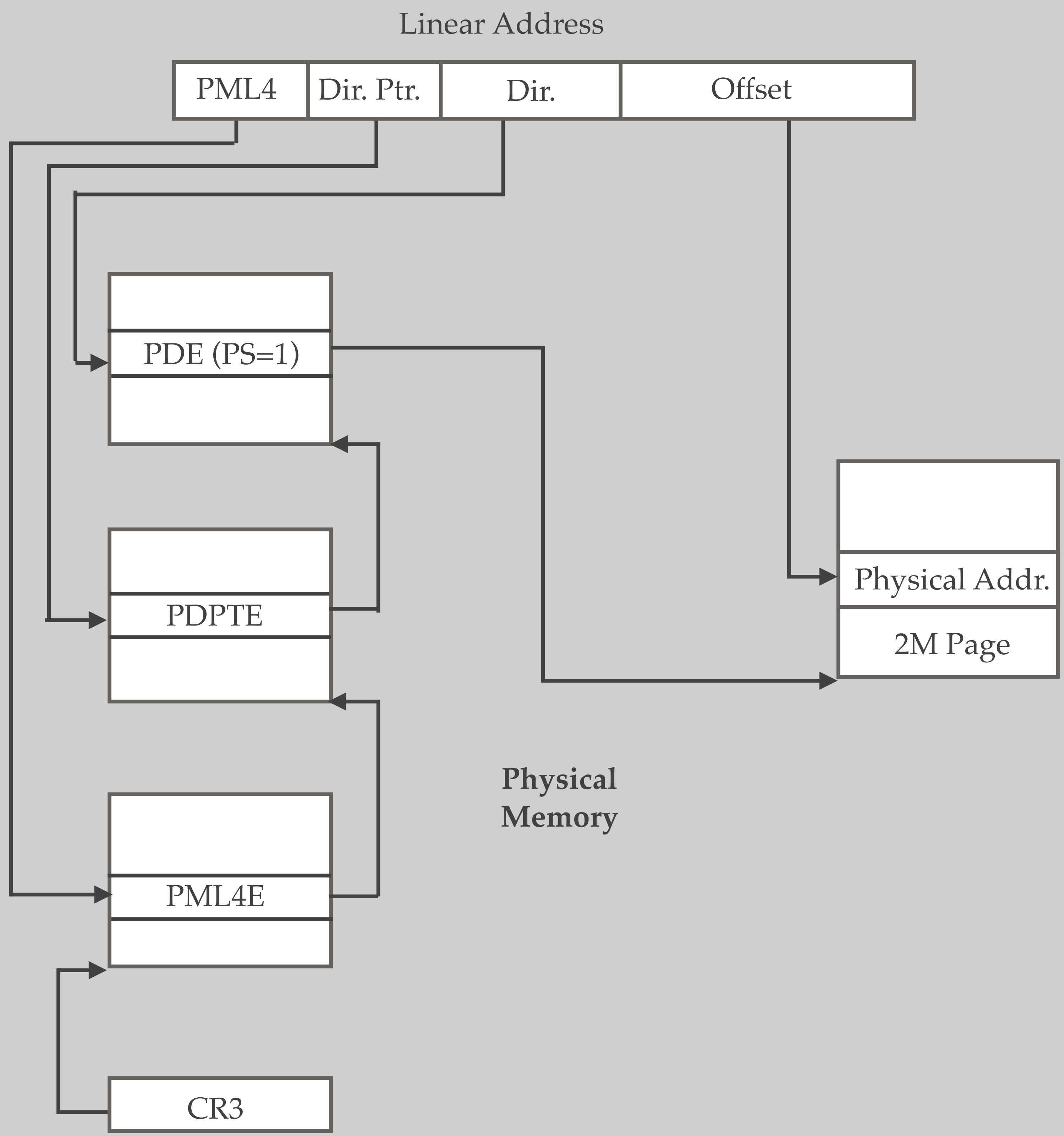
PAGING (2M pages)



PAGING (2M pages)



PAGING (2M pages)



PAGING (2M pages)

Physical
Memory

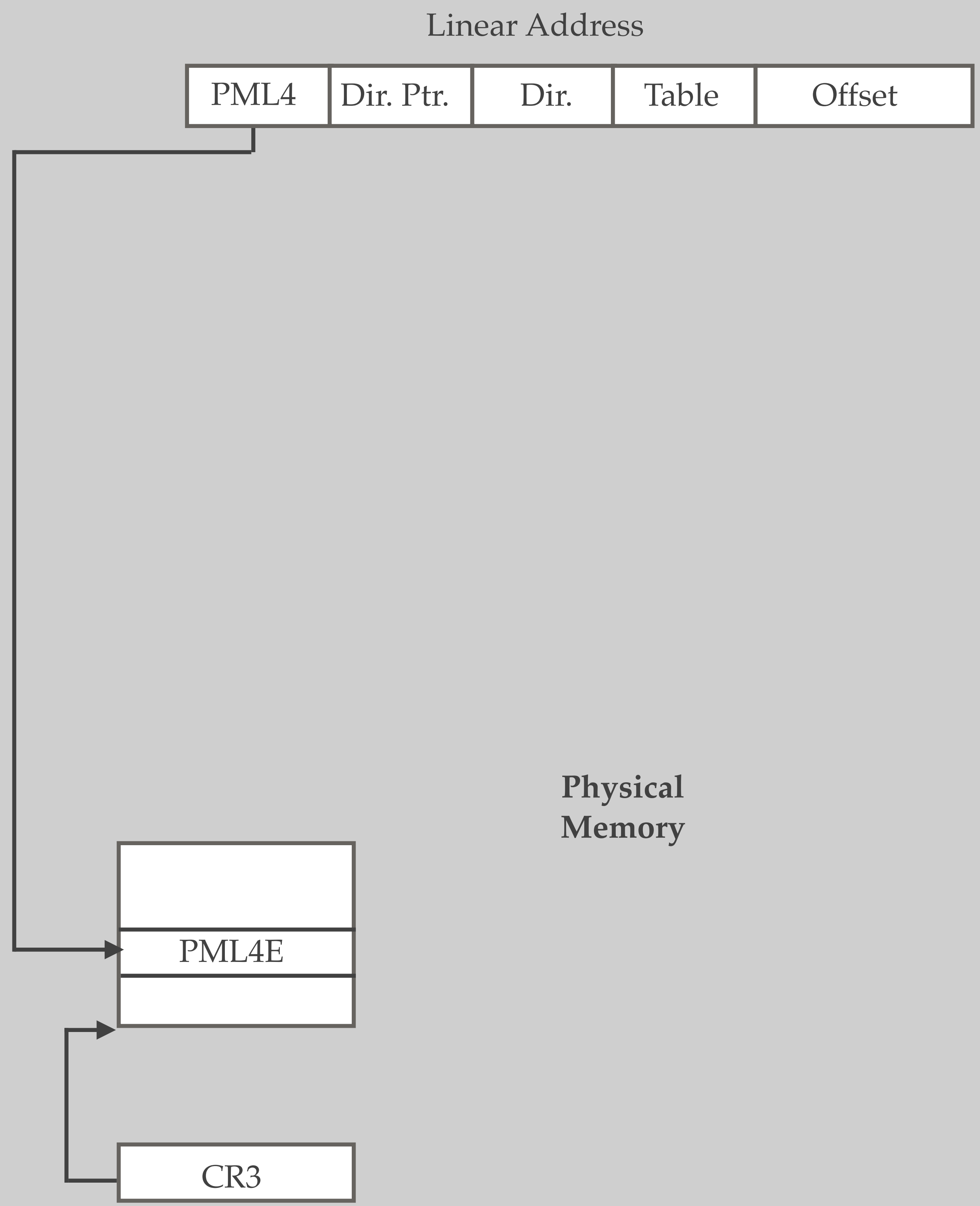
PAGING (4K pages)

Linear Address

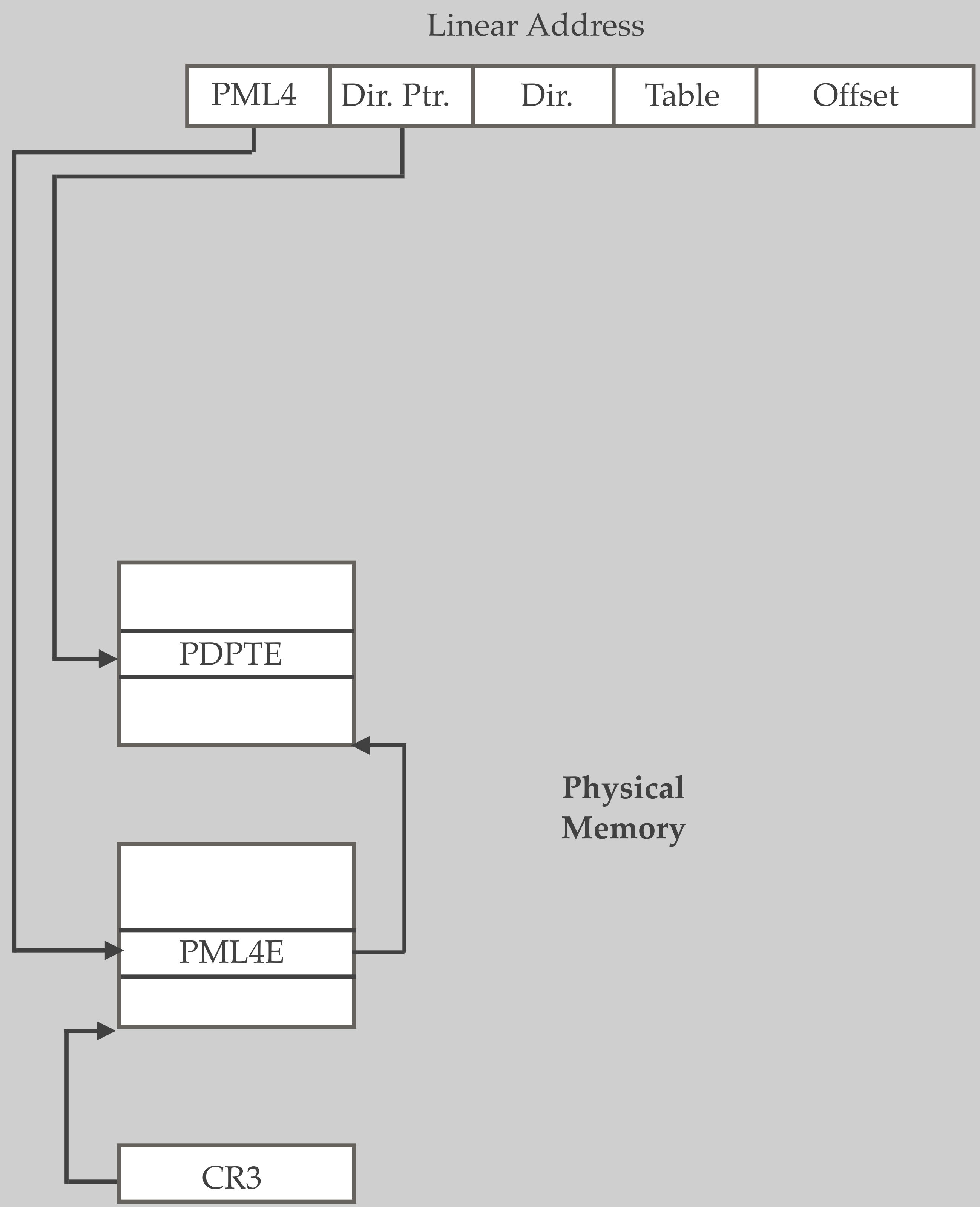
PML4	Dir. Ptr.	Dir.	Table	Offset

Physical Memory

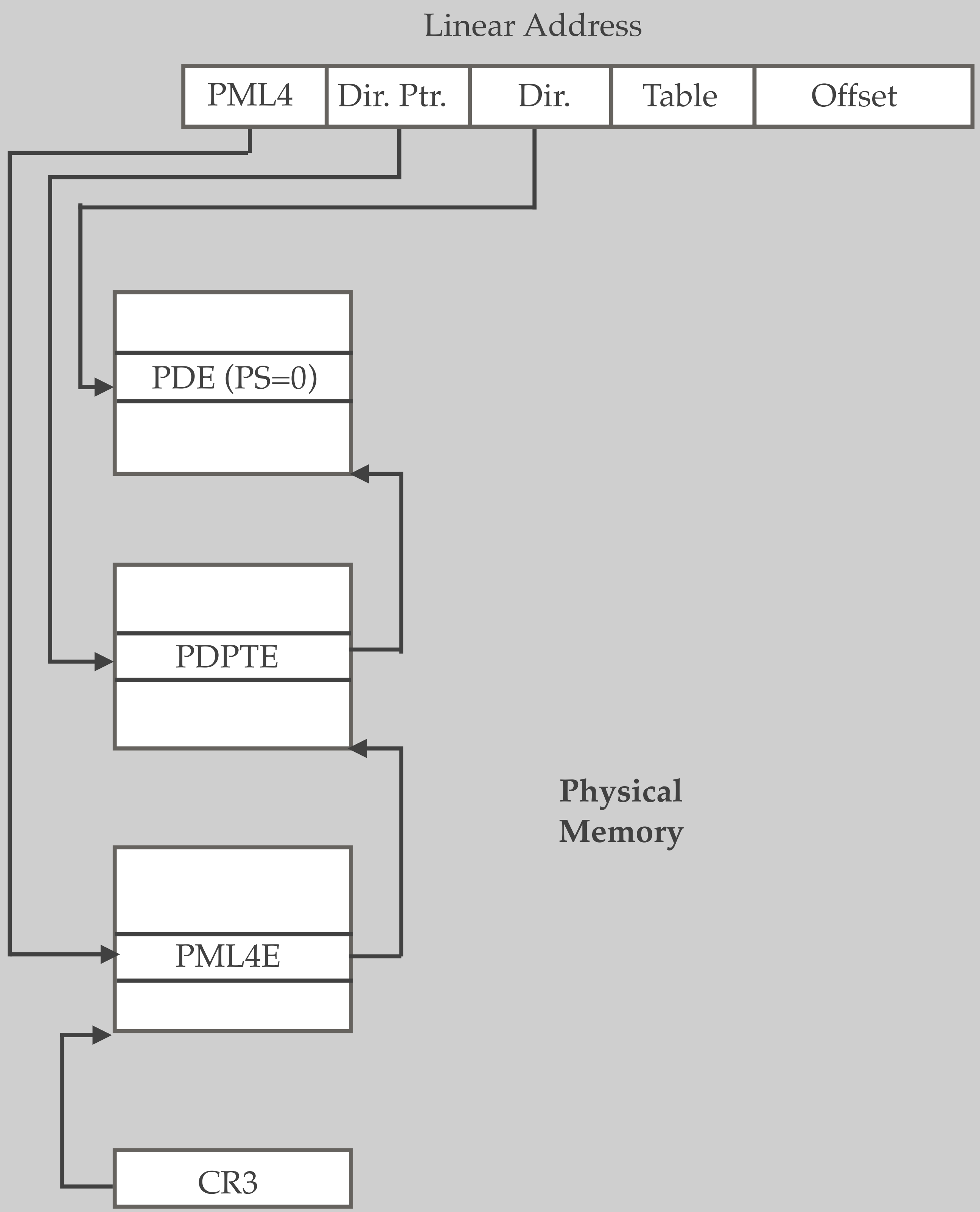
PAGING (4K pages)



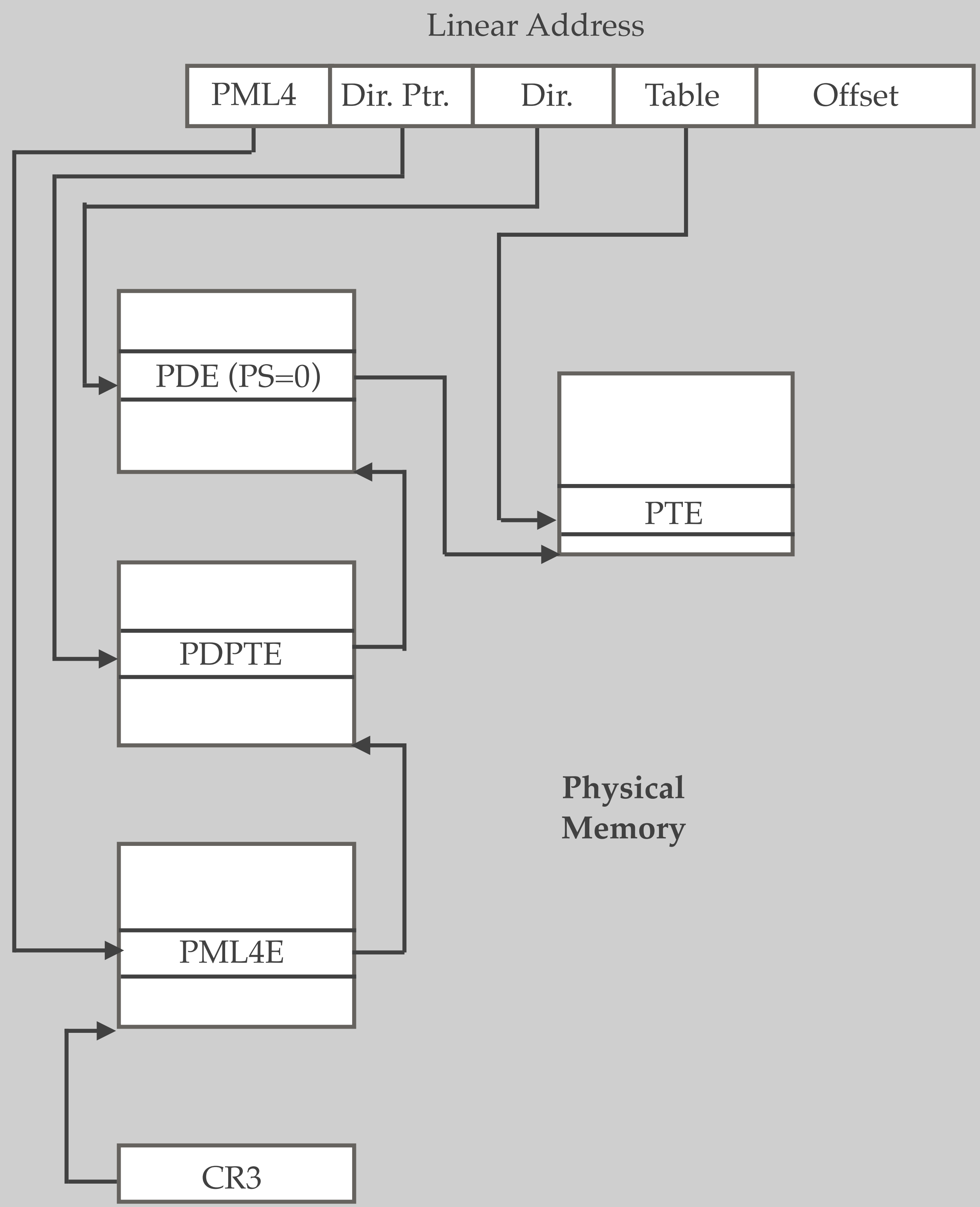
PAGING (4K pages)



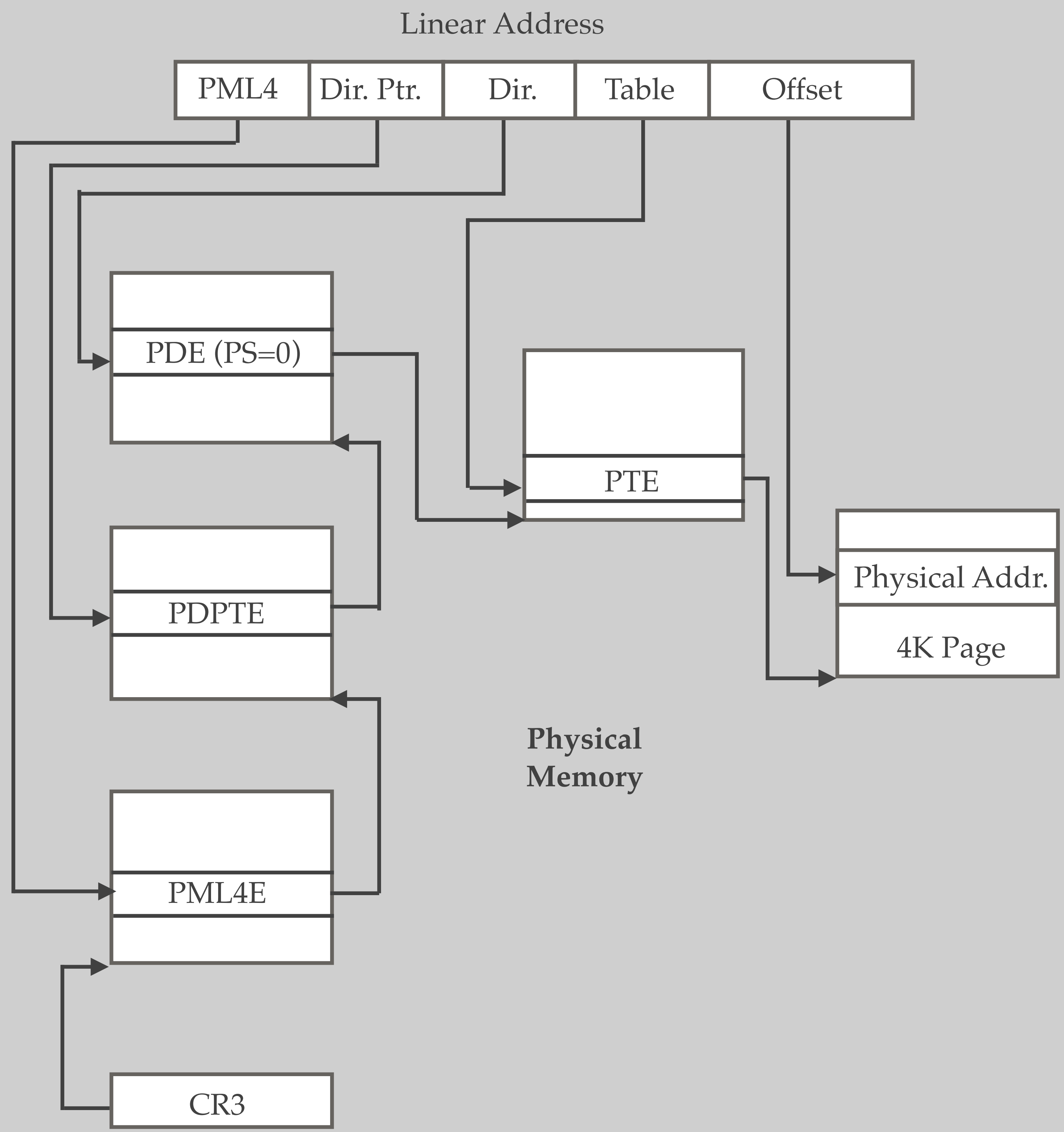
PAGING (4K pages)



PAGING (4K pages)



PAGING (4K pages)



PAGING (4K pages)

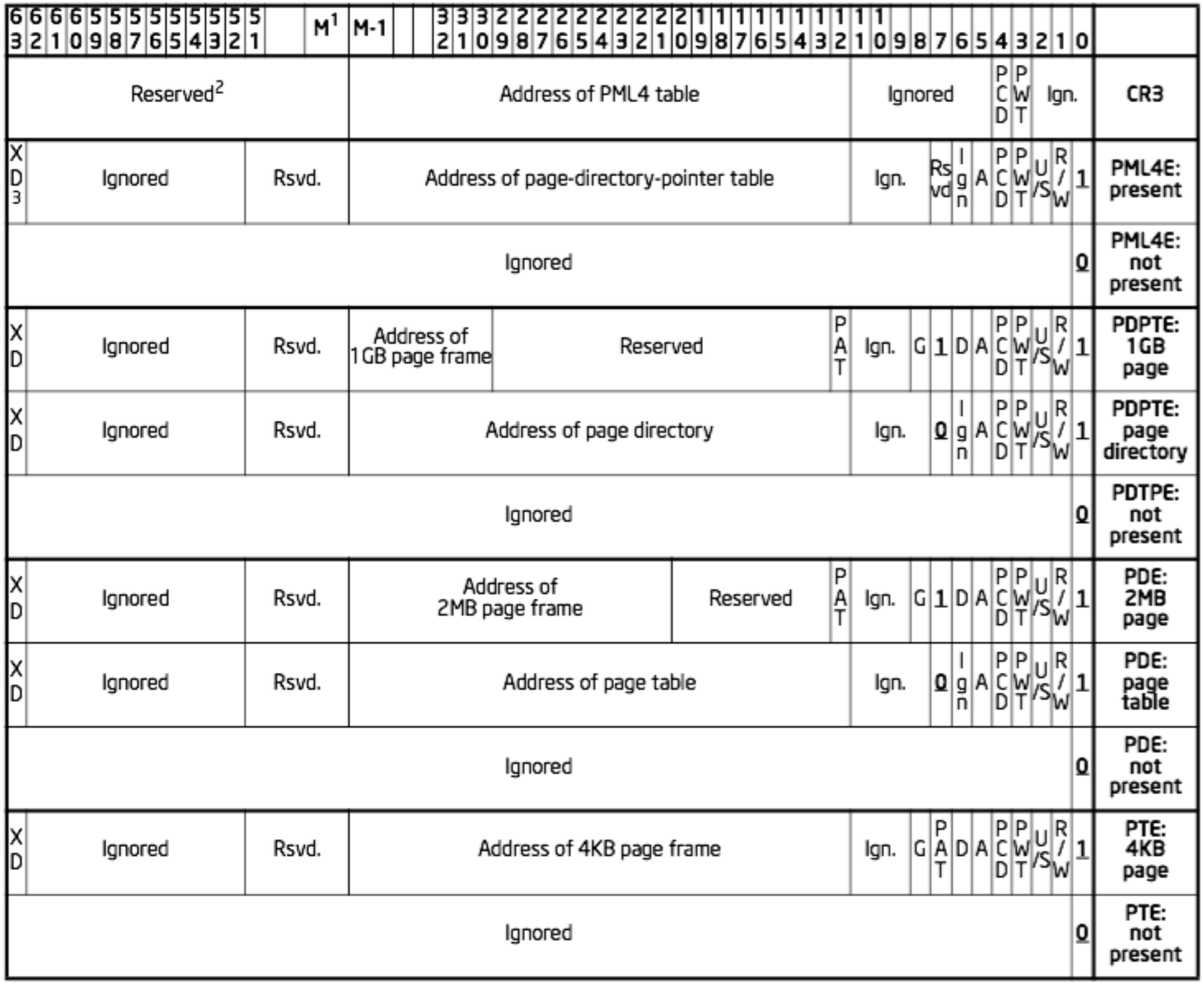


Figure 4-11. Formats of CR3 and Paging-Structure Entries with IA-32e Paging

Source: Intel Manuals, Vol. 3

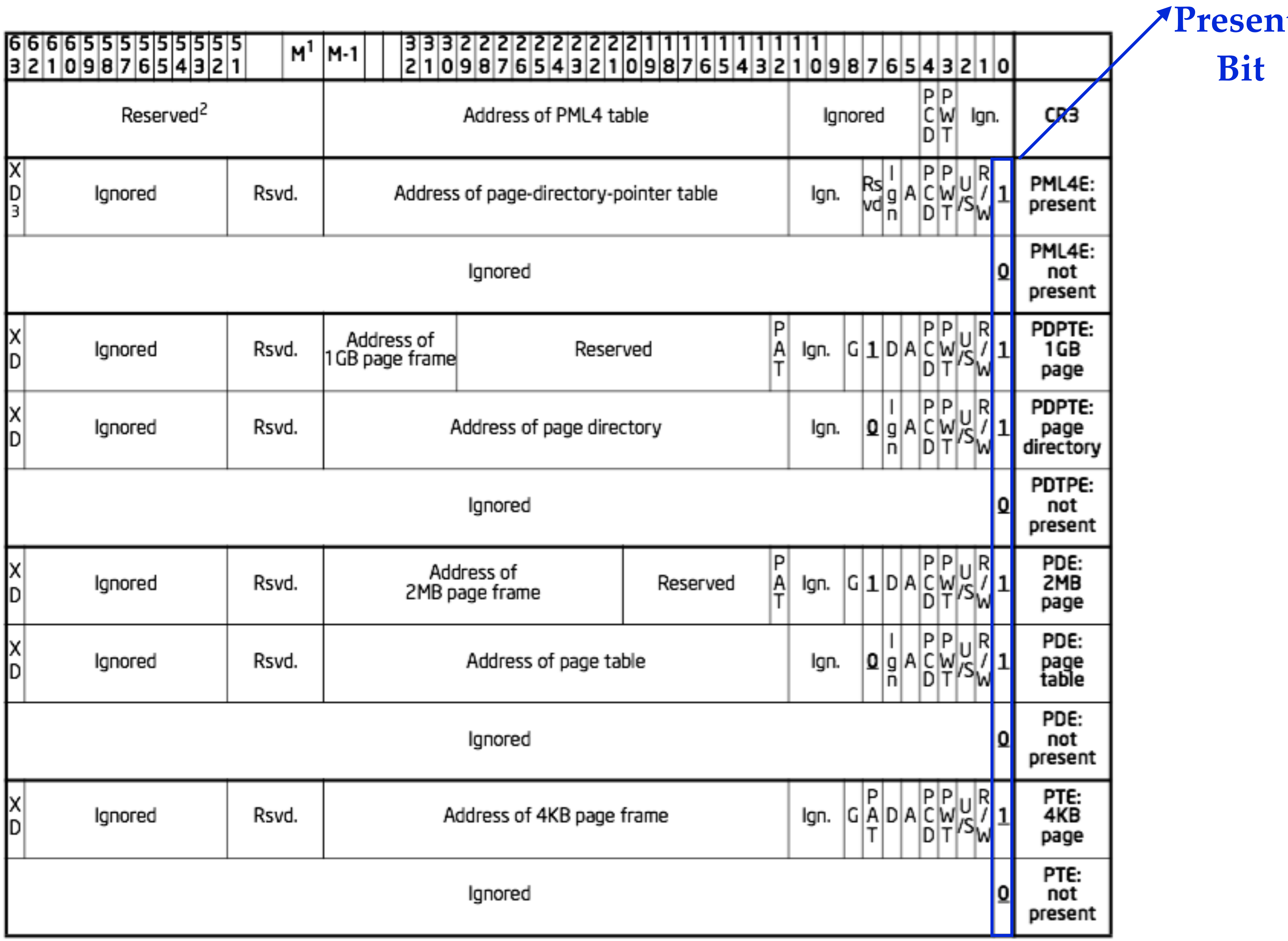


Figure 4-11. Formats of CR3 and Paging-Structure Entries with IA-32e Paging

Source: Intel Manuals, Vol. 3

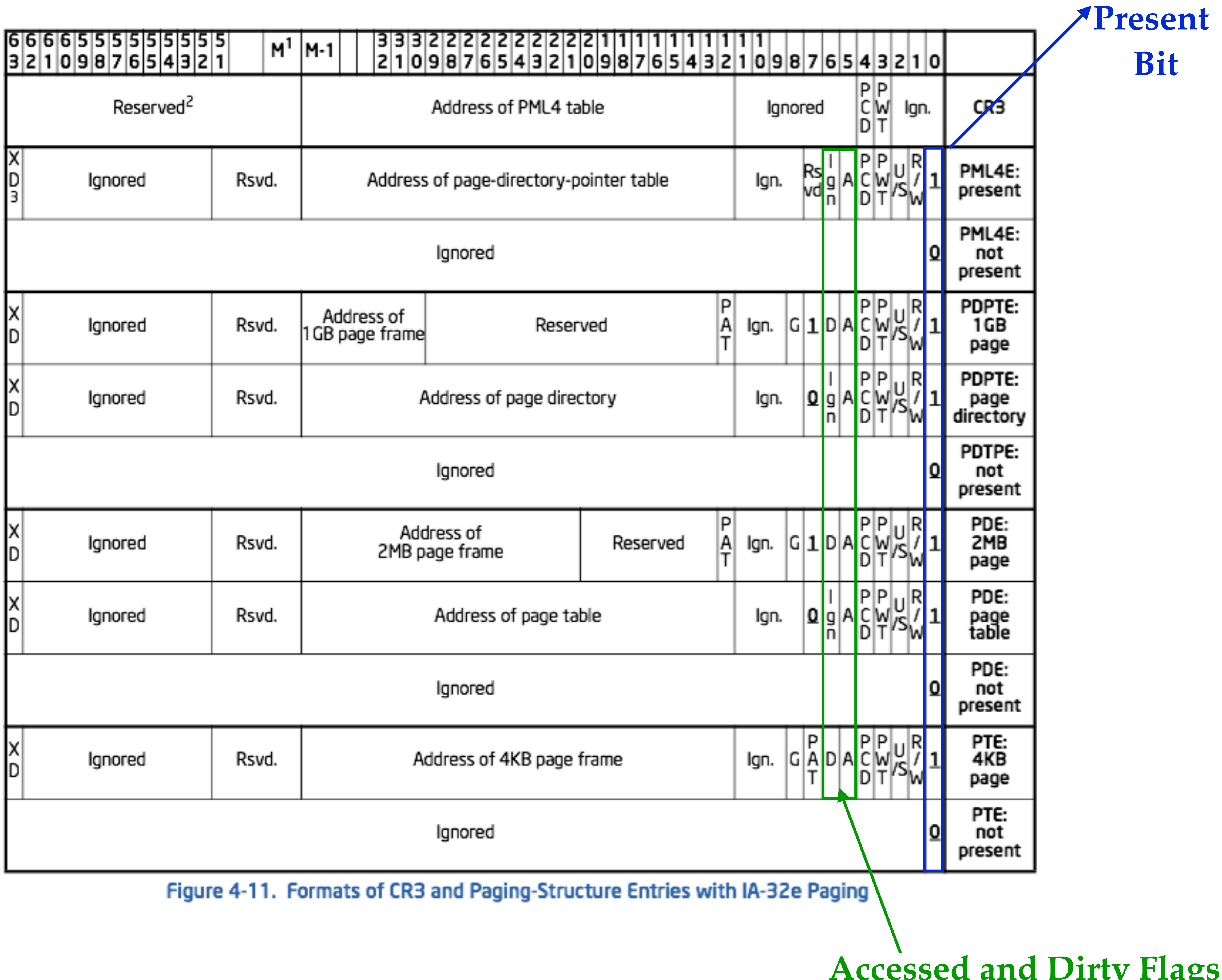


Figure 4-11. Formats of CR3 and Paging-Structure Entries with IA-32e Paging

Source: Intel Manuals, Vol. 3

Some Terms

- Walks
- Page fault
- Valid walk*
- Valid entry*
- Translation-governing addresses*

Outline

- Background
- Description of paging
- **Proving a memory RoW theorem in the context of paging**
- Challenges
- Future Work and Conclusion

Physical Memory Accessor and Updater

```
(memi  phy-addr x86)     => val
(!memi phy-addr val x86) => x86’
```

```
(defthm memi-!memi
  (equal (memi i1 (!memi i2 v x86))
   (if (equal i1 i2) v (memi i1 x86))))
```

Linear Memory Accessor and Updater

```
(rm08 lin-addr r-x x86) => (mv flg val x86’)
(wm08 lin-addr val x86) => (mv flg x86’)
```

Linear Memory Accessor and Updater

```
(rm08 lin-addr r-x x86) => (mv flg val x86')
(wm08 lin-addr val x86) => (mv flg x86')
```

```
(defun rm08 (lin-addr r-x x86)

  (if (programmer-level-mode x86)

      (rlm08 lin-addr x86)

    (b* ((cs (segi *cs* x86))
         (cpl (seg-sel-slice :ss-rpl cs))
         ((mv flag phy-addr x86)
          (la-to-pa lin-addr r-x cpl x86))
         ((when flag)
          (mv (list 'rm08 flag) 0 x86))
         (byte (memi phy-addr x86)))
        (mv nil byte x86))))
```

Linear Memory Accessor and Updater

```
(rm08 lin-addr r-x x86) => (mv flg val x86')
(wm08 lin-addr val x86) => (mv flg x86')
```

```
(defun rm08 (lin-addr r-x x86)

  (if (programmer-level-mode x86)

      (rlm08 lin-addr x86)

    (b* ((cs (segi *cs* x86))
         (cpl (seg-sel-slice :ss-rpl cs))
         ((mv flag phy-addr x86)
          (la-to-pa lin-addr r-x cpl x86))
         ((when flag)
          (mv (list 'rm08 flag) 0 x86))
         (byte (memi phy-addr x86)))
        (mv nil byte x86))))
```

```
(defun wm08 (lin-addr val x86)

  (if (programmer-level-mode x86)

      (wlm08 lin-addr val x86)

    (b* ((cs (segi *cs* x86))
         (cpl (seg-sel-slice :ss-rpl cs))
         ((mv flag phy-addr x86)
          (la-to-pa lin-addr :w cpl x86))
         ((when flag)
          (mv (list 'wm08 flag) x86))
         (byte (n08 val))
         (x86 (!memi phy-addr byte x86)))
        (mv nil x86))))
```

“Walkers”

- First few versions of the walkers were written by Robert Krug. Each of these walkers return `(mv flg phy-addr x86)`.
  - `la-to-pa`
    - `la-to-pa-pml4-table`
      - `la-to-pa-page-dir-ptr-table (1G pages)`
        - `la-to-pa-page-directory (2M pages)`
          - `la-to-pa-page-table (4K pages)`
- For each structure, we define recognizers for valid entries.

Linear Memory RoW Theorem

```
(rm08 lin-addr r-x x86) => (mv flg val x86’)
(wm08 lin-addr val x86) => (mv flg x86’)
```

Let `addr1` and `addr2` be two linear addresses mapped to two ***distinct*** physical addresses.

Linear Memory RoW Theorem

```
(rm08 lin-addr r-x x86) => (mv flg val x86’)
(wm08 lin-addr val x86) => (mv flg x86’)
```

Let `addr1` and `addr2` be two linear addresses mapped to two ***distinct*** physical addresses.

```
(implies <hyps>
         (equal (mv-nth 1 (rm08 addr1 r-x (mv-nth 1 (wm08 addr2 val x86))))
                (mv-nth 1 (rm08 addr1 r-x x86))))
```

Linear Memory RoW Theorem

```
(rm08 lin-addr r-x x86) => (mv flg val x86')
(wm08 lin-addr val x86) => (mv flg x86')
```

Let `addr1` and `addr2` be two linear addresses mapped to two ***distinct*** physical addresses.

```
(implies <hyps>
         (equal (mv-nth 1 (rm08 addr1 r-x (mv-nth 1 (wm08 addr2 val x86))))
                (mv-nth 1 (rm08 addr1 r-x x86))))
```

Most of talk is about what needs to be done to prove the above theorem in the context of physical memory!

Approach

Address Enumeration: enumerate all the translation-governing addresses to state disjointness properties about them

Approach

Address Enumeration: enumerate all the translation-governing addresses to state disjointness properties about them

Hypotheses of the memory RoW theorem:

Approach

Address Enumeration: enumerate all the translation-governing addresses to state disjointness properties about them

Hypotheses of the memory RoW theorem:

1. The entries at the translation-governing addresses of `addr1` and `addr2` are valid.
2. The physical addresses corresponding to `addr1` and `addr2` are distinct.
3. The translation-governing addresses of `addr1` and `addr2` are pairwise disjoint.
4. The physical address corresponding to `addr1` is not equal to any of the translation-governing addresses of `addr1` and `addr2`.

Attempting to prove the RoW Theorem…

<Demo>

Outline

- Background
- Description of paging
- Proving a memory RoW theorem in the context of paging
- **Challenges**
- Future Work and Conclusion

Challenges in Reasoning about Walks

Challenges in Reasoning about Walks

- There are many similar theorems about each of the hierarchical data structures, and **controlling the theory** is critical to manage proofs.

Challenges in Reasoning about Walks

- There are many similar theorems about each of the hierarchical data structures, and **controlling the theory** is critical to manage proofs.
- **Reasoning about equality of many bit fields** of two different entries is easier if a single function to capture this notion is defined.

Challenges in Reasoning about Walks

- There are many similar theorems about each of the hierarchical data structures, and **controlling the theory** is critical to manage proofs.
- **Reasoning about equality of many bit fields** of two different entries is easier if a single function to capture this notion is defined.
- Accessed and dirty flags are updated on the fly and it is often required to **separate** these updates from the traversals.

Challenges in Reasoning about Walks

- There are many similar theorems about each of the hierarchical data structures, and **controlling the theory** is critical to manage proofs.
- **Reasoning about equality of many bit fields** of two different entries is easier if a single function to capture this notion is defined.
- Accessed and dirty flags are updated on the fly and it is often required to **separate** these updates from the traversals.
- It is hard to keep track of theorems, because of their size and number.
  - Define **a small arithmetic theory**.
  - Important: **Find patterns**, stick to them! Name and order rules properly!

Challenges in Reasoning about Walks

Handy for dealing with large books and proofs:

1. ACL2(p)
2. Book misc/find-lemmas
3. define, defrule
4. `:brr` and Jared Davis's `why` macro for monitoring rewrite rules

Challenges in Reasoning about Walks

Handy for dealing with large books and proofs:

1. ACL2(p)
2. Book misc/find-lemmas
3. define, defrule
4. `:brr` and Jared Davis's `why` macro for monitoring rewrite rules

```
(defmacro why (rule)
  `(ACL2::er-progn
    (ACL2::brr t)
    (ACL2::monitor '(:rewrite ,rule) ''(:eval :go t))))

(defmacro why! (rule)
  `(ACL2::er-progn
    (ACL2::brr t)
    (ACL2::monitor '(:rewrite ,rule) ''(:unify-subst :hyps :eval :go t))))
```

Outline

- Background
- Description of paging
- Proving a memory RoW theorem in the context of paging
- Challenges
- **Future Work and Conclusion**

Future Work

- **Generalize paging walk theorems.**
  - Currently, I require translation-governing addresses of `rm08` and `wm08` in the RoW lemma to be pairwise disjoint, but only the corresponding physical addresses need to be unequal (?).

Future Work

- **Generalize paging walk theorems.**
    - Currently, I require translation-governing addresses of `rm08` and `wm08` in the RoW lemma to be pairwise disjoint, but only the corresponding physical addresses need to be unequal (?).
- **Verify a system-level program** that performs some aspect of paging data structure management to figure out what theorems about paging walks are missing.

Conclusion

- In some ways, reasoning about paging data structure walks was easier than the Rockwell challenge problem.
  - Paging data structures have **well-defined boundaries** and fixed sizes, unlike data structures embedded in linear address space where data structure “shape” has to be reconstructed.
  - Rockwell challenge asked for general solutions, but I opted for a solution **specific to the paging data structures** for the sake of efficiency.

Conclusion

- In some ways, reasoning about paging data structure walks was easier than the Rockwell challenge problem.
    - Paging data structures have **well-defined boundaries** and fixed sizes, unlike data structures embedded in linear address space where data structure "shape" has to be reconstructed.
    - Rockwell challenge asked for general solutions, but I opted for a solution **specific to the paging data structures** for the sake of efficiency.
- The hard part hasn't come yet.
    - Challenge: Paging should be **transparent** to the verification of properties of data structures in linear memory, unless an erroneous condition occurs during address translations.