Isolation und Schutz in Betriebssystemen

4. Paging bei x86-64

Michael Schöttner

Stand bisher

Heinrich Heine
Universität Düsseldorf

- Durch die Segmentierung können wir zwischen User-Mode und Kernel-Mode Threads unterscheiden
- Dadurch können User-Mode Threads keine privilegierte Befehle ausführen
- Portzugriffe haben wir ebenfalls gesperrt (über rflags und das TSS)
- Nun folgt Paging, wodurch der Speicherzugriff beschränkt werden kann
  - Damit können Prozesse im User-Mode isoliert werden
  - Sowie der Kerneladressbereich vor User-Mode Threads geschützt werden

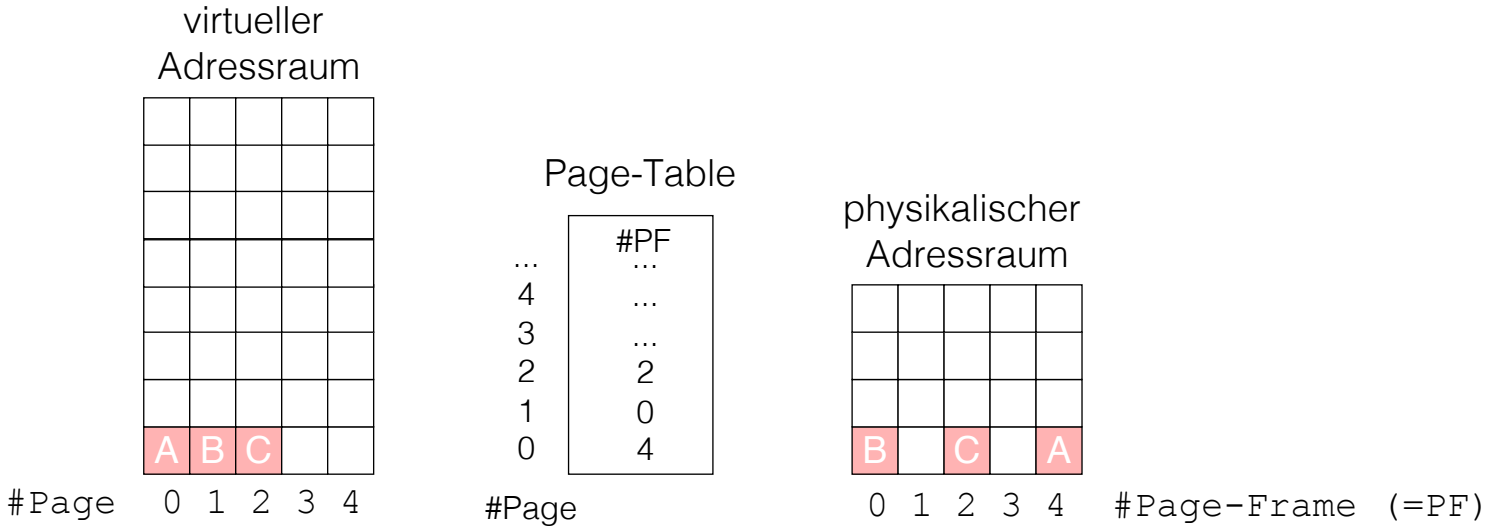
Allgemeines zu Paging

Heinrich Heine
Universität Düsseldorf

- Virtueller Adressraum wird in gleich große Seiten (engl. pages) unterteilt
- Physikalischer Adressraum wird in gleich große Kacheln (engl. page frames) zerlegt
- I.d.R. verwendet man 4 KB Pages und damit 4 KB Page-Frames
  - Ein Page-Frame speichert eine Page, daher gleiche Größe
  - Es gibt noch Sonderfälle von größeren Pages (vielfaches von 4 KB, z.B. 1 GB)
- Der Schutz ist somit sehr feingranular definierbar, jeweils für 4 KB
  - Nur lesen
  - Code ausführen
  - Zugriff im User-Mode erlaubt/verboten

Virtuelle vs. physikalische Adressen

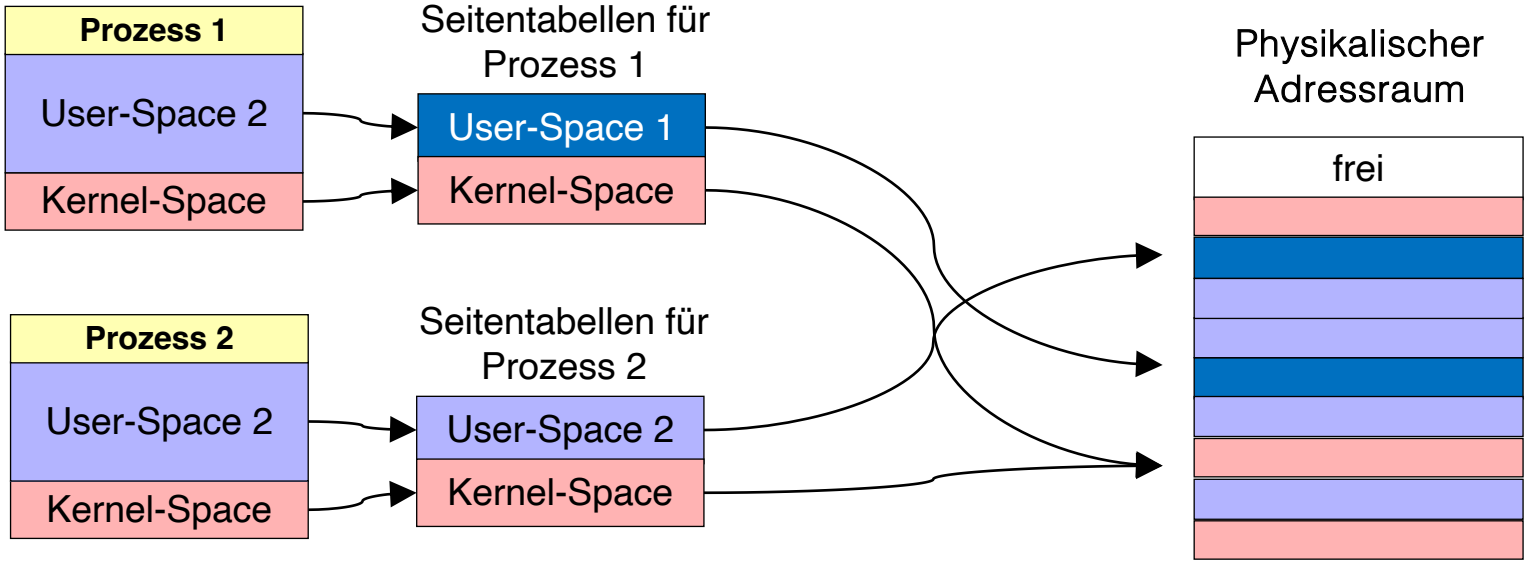
- Wir unterscheiden virtuelle/logische Adressen (je nach Prozessor bis 57 Bit) von physikalischen Adressen (abhängig vom installierten DRAM)
  - Ein Speicherblock im virtuellen Adressraum kann auf beliebige Page-Frames abgebildet werden



Isolation von Prozessen

- Jeder Prozess hat seine eigenen Seitentabellen die seinen Adressraum beschreiben
- Wir blenden den Kernel in jeden Adressraum ein

Kacheln können beliebig zugeordnet werden



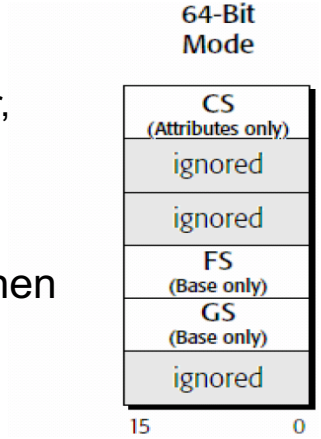
Isolation von Prozessen

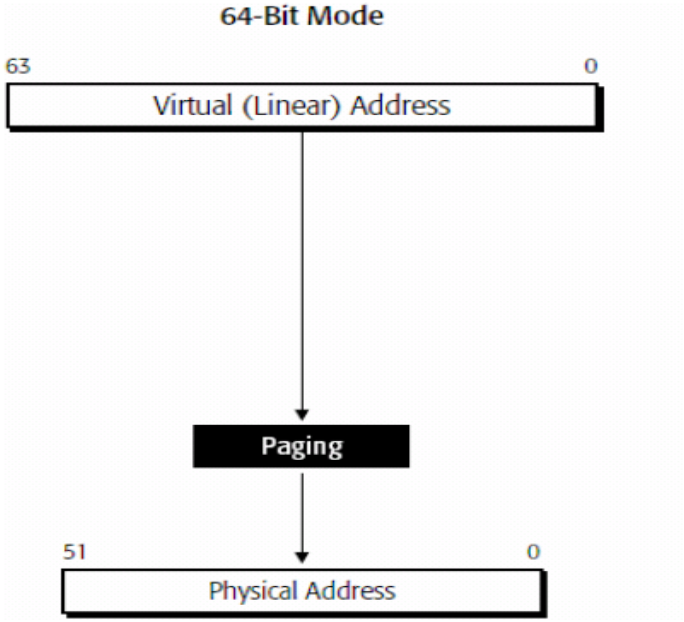
Heinrich Heine
Universität Düsseldorf

- Ein Thread läuft in einem Prozessadressraum und kann nur auf virtuellen Adressen von Pages zugreifen, sofern der Seitentabelleneintrag dies erlaubt.
- Hierfür gibt es Bits in jedem Seitentabelleneintrag (siehe später)
- Falls eine Seite als nicht präsent markiert ist, so ist diese entweder ausgelagert oder nicht genutzt. Eine Zugriff darauf löst einen Page-Fault aus.
- Die Seitentabellen selbst müssen vor dem Zugriff geschützt werden, genauso wie der Kernel-Code. Auch hierfür gibt es ein entsprechendes Bit, siehe später

Long Mode: Segmentierung (Wdlg.)

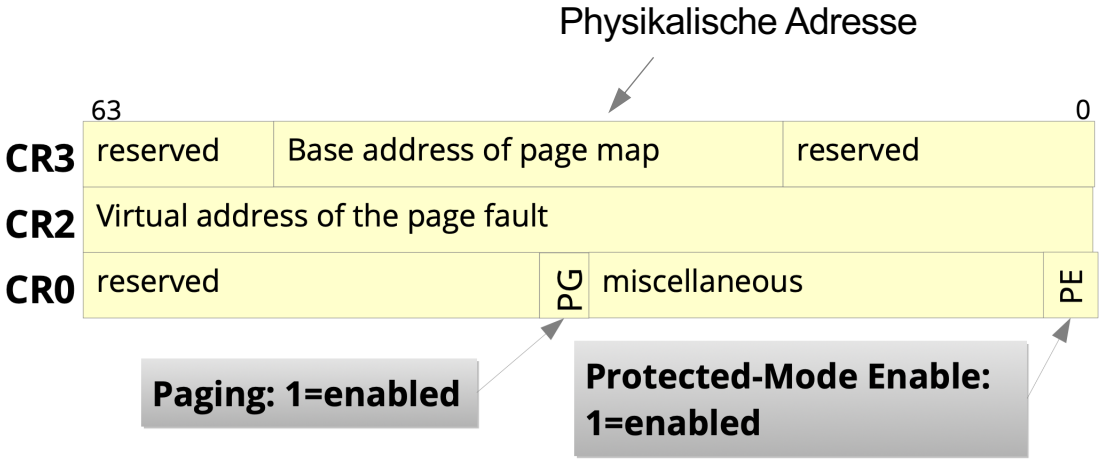
- 16-Bit Selektoren
- Startadressen werden bei fast allen Segmenten ignoriert, außer FS und GS
  - fs: für threadlokalen Speicher, verwendet von Thread-Bibliotheken
  - gs: Linux verwendet dieses Register, um bei Fast Systemcalls hierüber den Stack zu bekommen
- Keine Limit-Prüfung; im Wesentlichen ist nur noch die Ring-Nummer in CS relevant = CPL





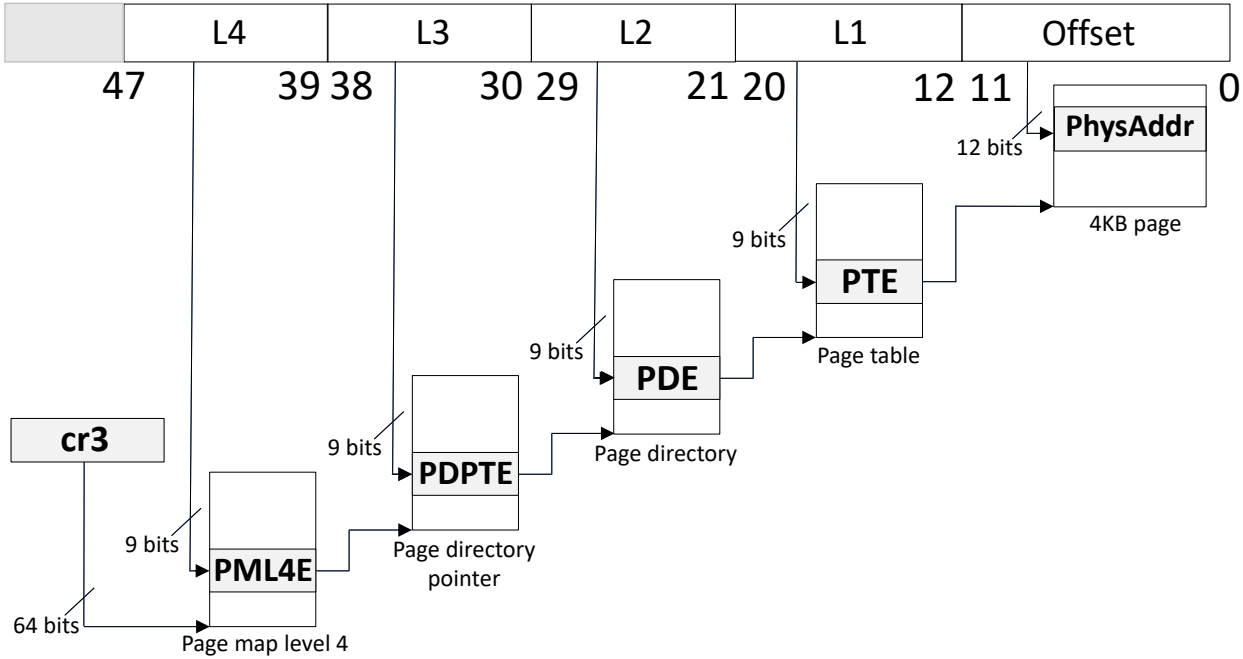
Long Mode: Paging

- Paging ist für den Long Mode zwingend erforderlich
- Die wichtigsten Verwaltungsinformationen für das Paging stehen in den CRx Steuerregistern (CR = Control Register):



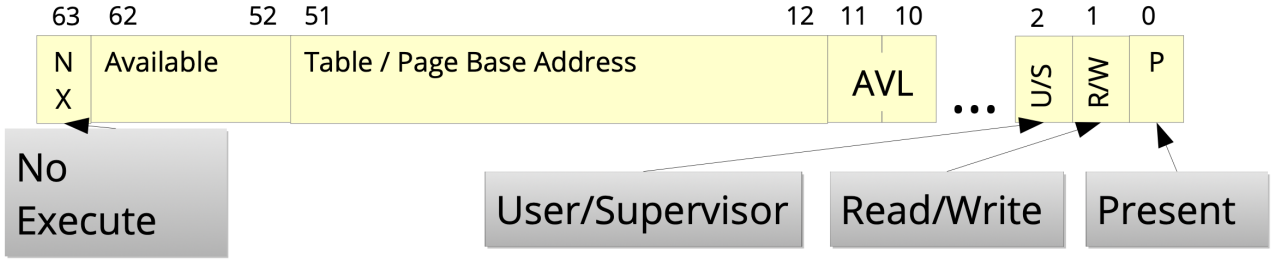
Long Mode: Page Tables

- Vierstufiges Paging → (auch 5stufig)
- Index: 9 Bit
  - 4KB pro Tabelle
  - 8 Bytes pro Eintrag
  - 512 Einträge pro Tab.



Page Table Entry

- Einstellungen sind auf allen Ebenen der Page-Table-Hierarchie möglich
- Schutzbits:
  - R/W = Read/Write → Schutz vor Schreibzugriffen
  - NX = NoeXecute → keinen Code in Daten ausführen
  - U/S = User/Supervisor → Zugriff für alle oder nur im Ring 0

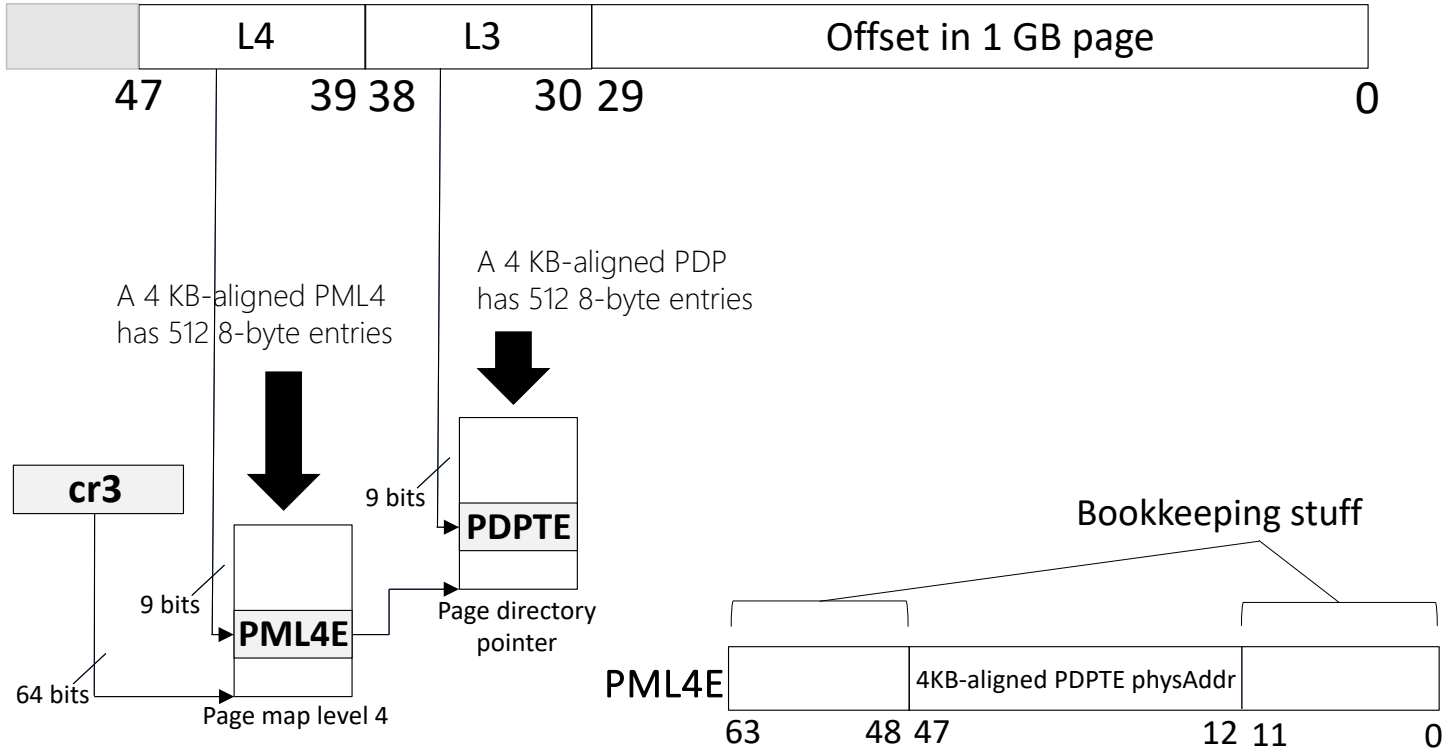


  - Bem.: Available = AVL = für das Betriebssystem frei verwendbare Bits

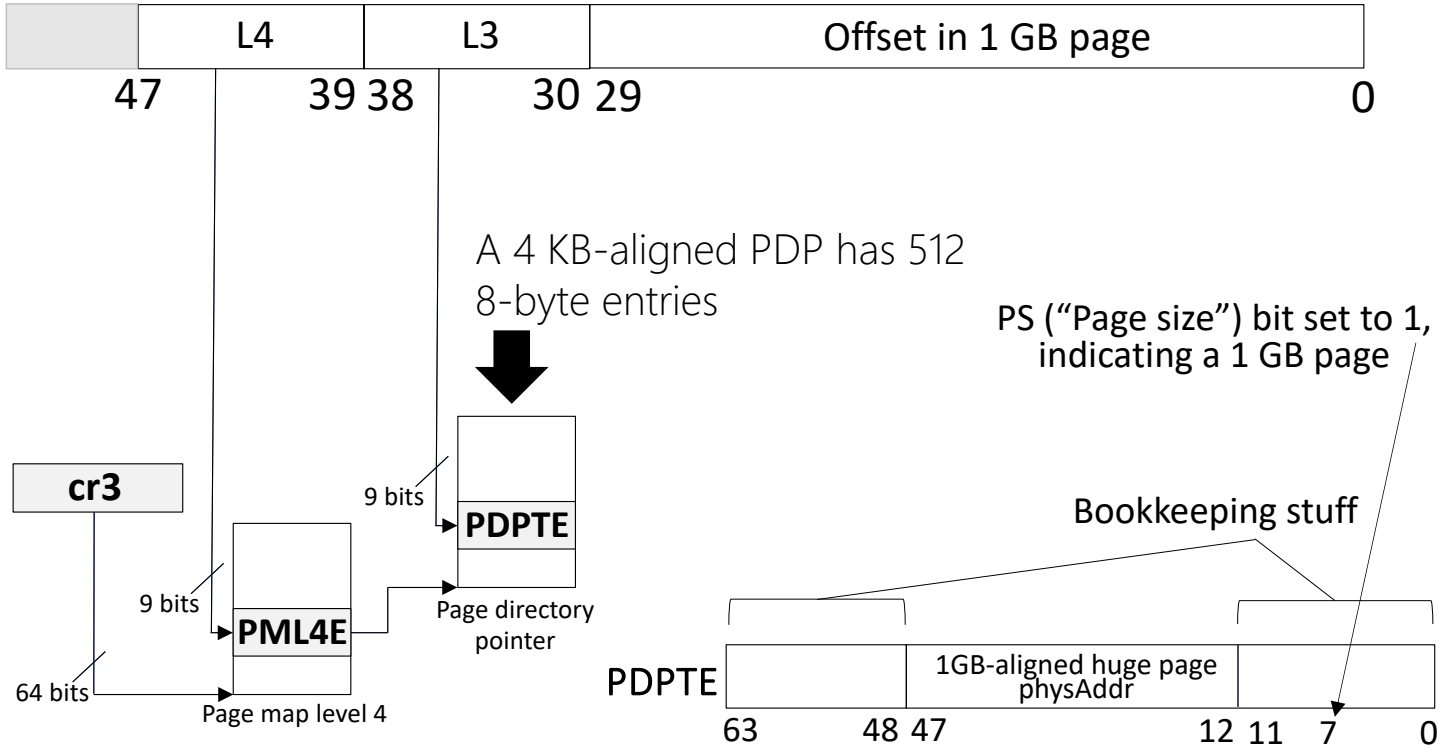
Bemerkungen

- Warum gibt es so viele Stufen?
  - I.d.R. wird der virtuelle Adressraum nur spärlich und verstreut genutzt
  - Allenfalls großer Heap bei Big-Data-Systemen
  - Aber Stacks und Code benötigen nicht Gigabytes
- Es werden nur notwendige Tabellen angelegt
  - Ein Eintrag in PML4E beschreibt 512 GB
  - Falls dieser Adressbereich komplett ungenutzt ist, so wird PML4E.present bit = 0 gesetzt
  - Und entsprechend werden nicht alle zugehörigen Tabellen PDP/PD/PT Tabellen in diesem Unterbaum angelegt

Beispiel: 1GB Pages



Beispiel: 1GB Pages



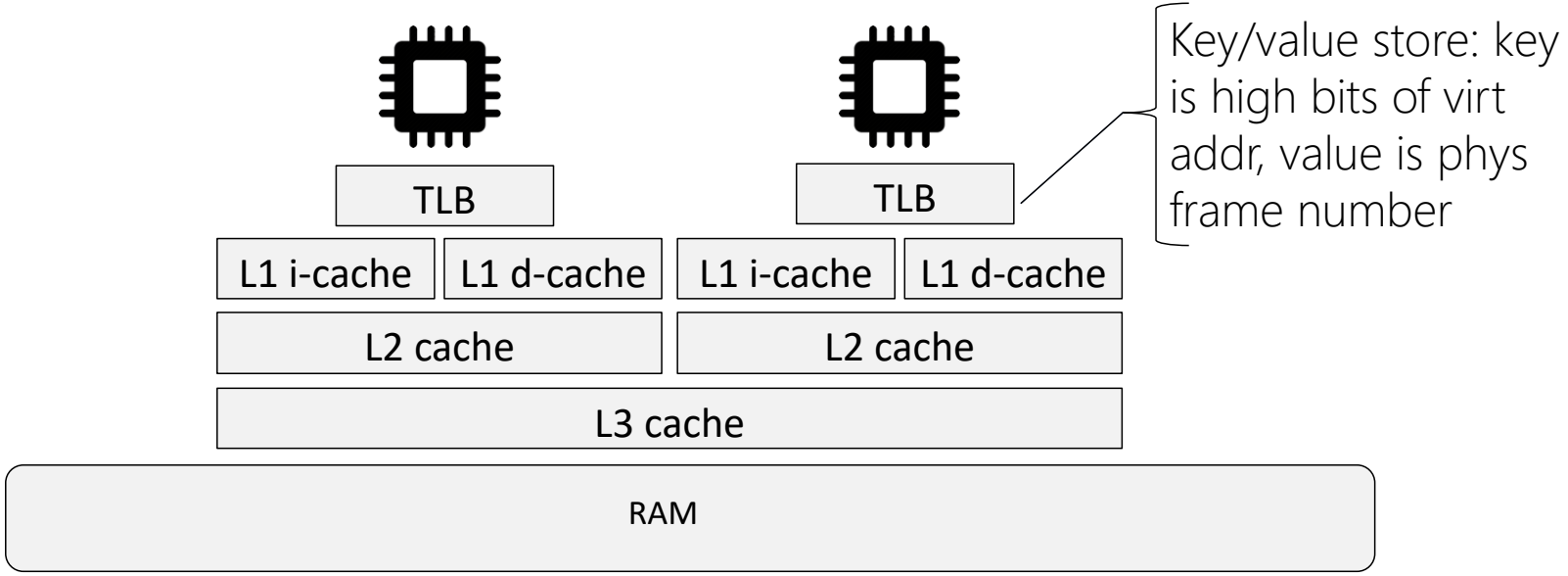
Long Mode: TLB

Heinrich Heine
Universität Düsseldorf

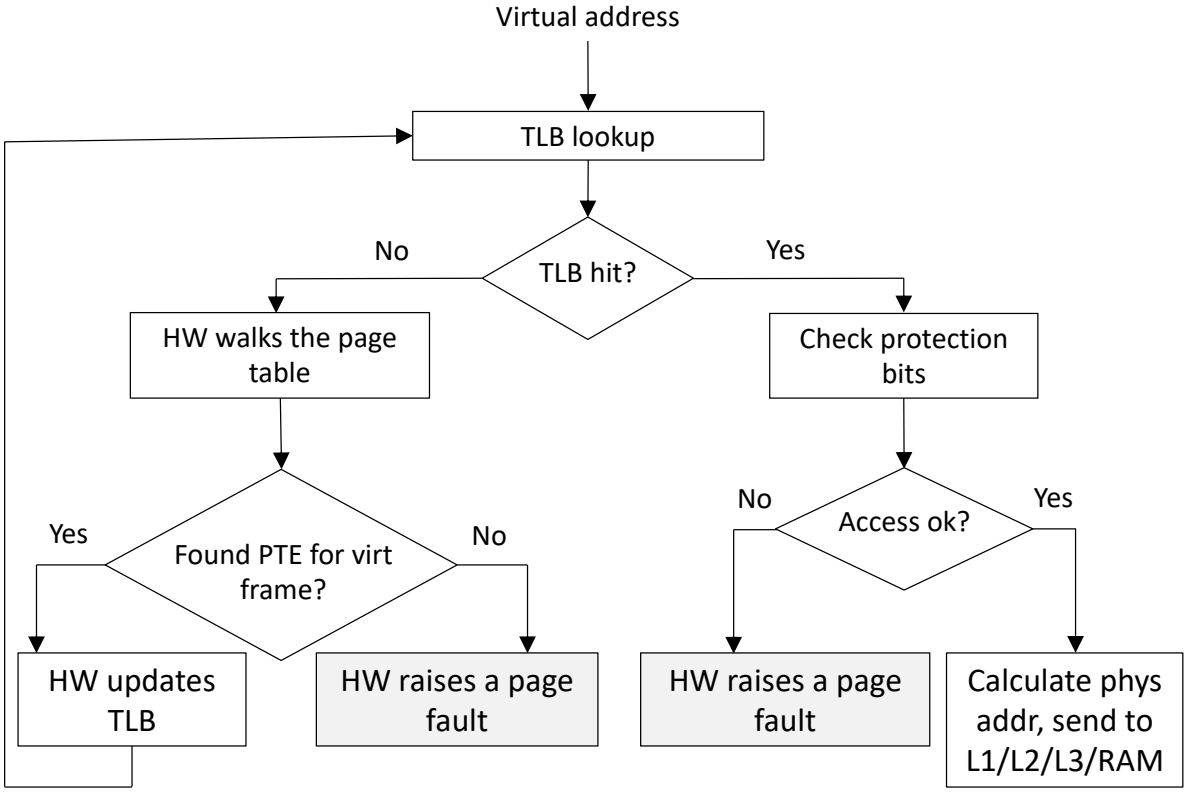
- Problem: Paging erfordert eine Adressübersetzung durch die Memory Management Unit (MMU) – Funktionseinheit pro Core
  - Durch die vier oder sogar fünf Stufen ist die Adressübersetzung langsam
  - Der Long Mode erzwingt jedoch Paging

- Lösung: Translation Lookaside Buffer (TLB) – für jeden Core vorhanden:
  - puffert bereits übersetzte Adressen
  - vollassoziativer Cache → O(1)-Lookup
    - Tag: bestehend aus Page-Table-Indizes
    - Daten: Page Frame Adresse

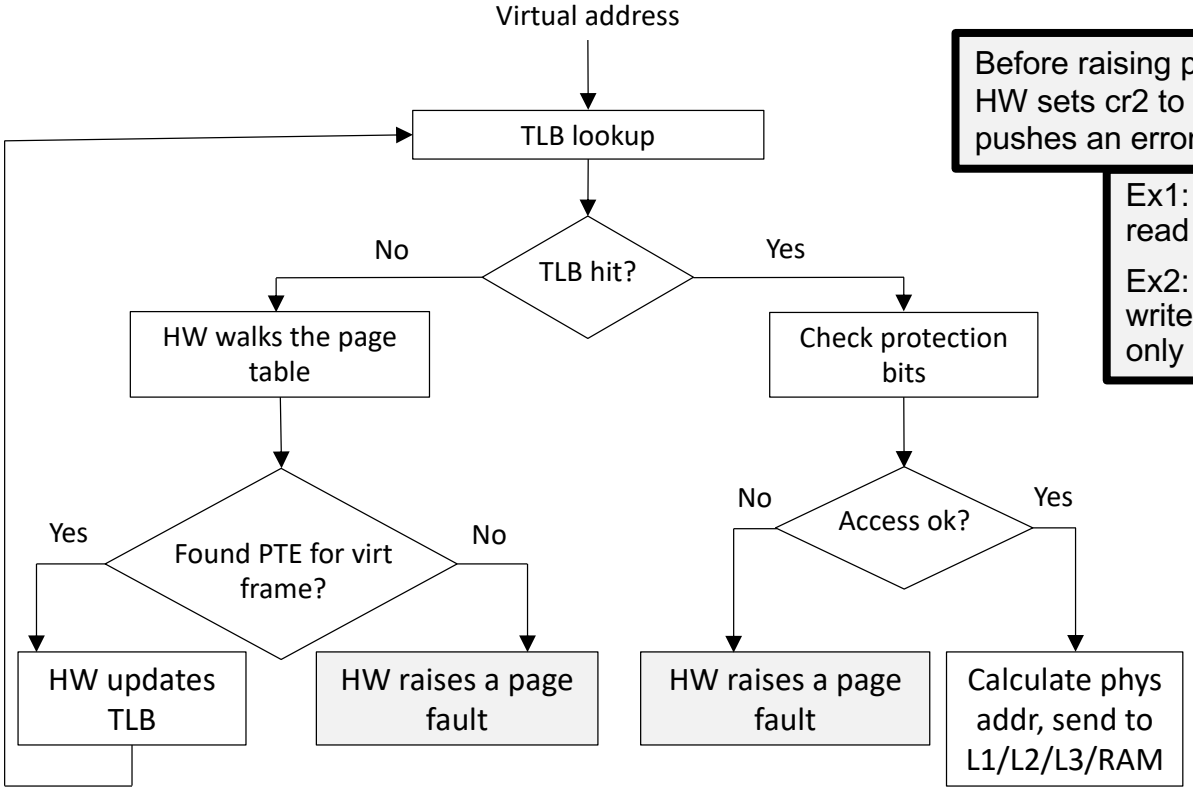
Long Mode: TLB



Long Mode: TLB



Long Mode: TLB



Before raising page fault exception, HW sets cr2 to faulting address, and pushes an error code onto stack

Ex1: User thread tried to read a non-present page

Ex2: User thread tried to write a present but read-only page

Long Mode: TLB

Heinrich Heine
Universität Düsseldorf

- TLB hat nicht viele Einträge ~32 – 100 Einträge, je nach Prozessor
- Bei normalen Anwendungen erreicht der TLB eine Trefferrate jedoch von bis zu 98%
- Warum? → Lokalitätsprinzip
  - Arrays, Schleifen, etc.

- Bemerkungen
  - Schreiben in das CR3 Register (Umschalten des Adressraums) invalidiert den TLB komplett
  - Falls Schutzbits oder Access-Bits in einem Page-Table-Eintrag geändert werden, muss auch der TLB respektive der betroffene TLB-Eintrag gelöscht werden!
    - Bei Multicore notfalls auf allen Cores!

Fortsetzung folgt

Heinrich Heine
Universität Düsseldorf

- Kernel isolation after Meltdown
- Process Context Identifiers
- Memory protection keys