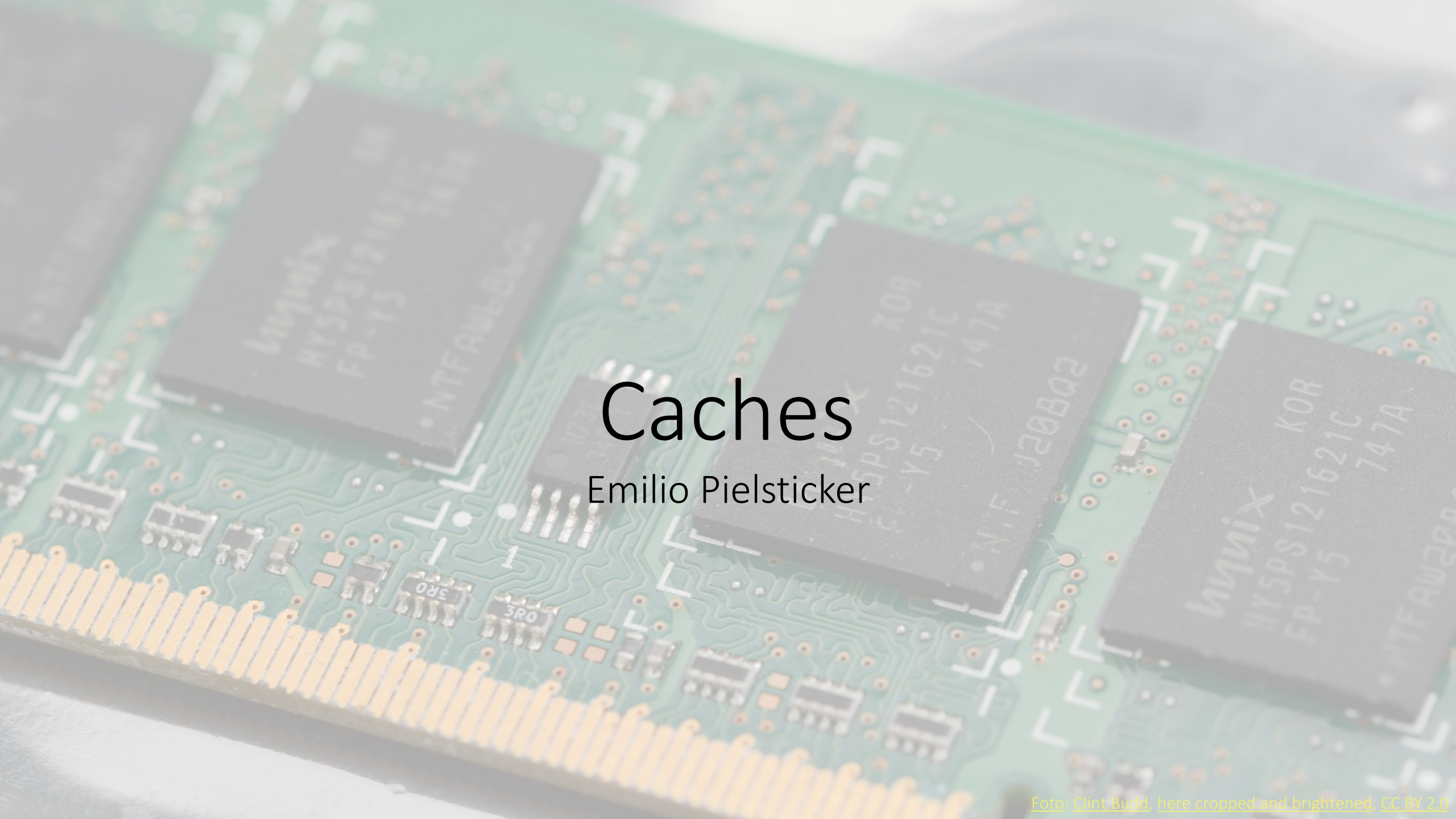


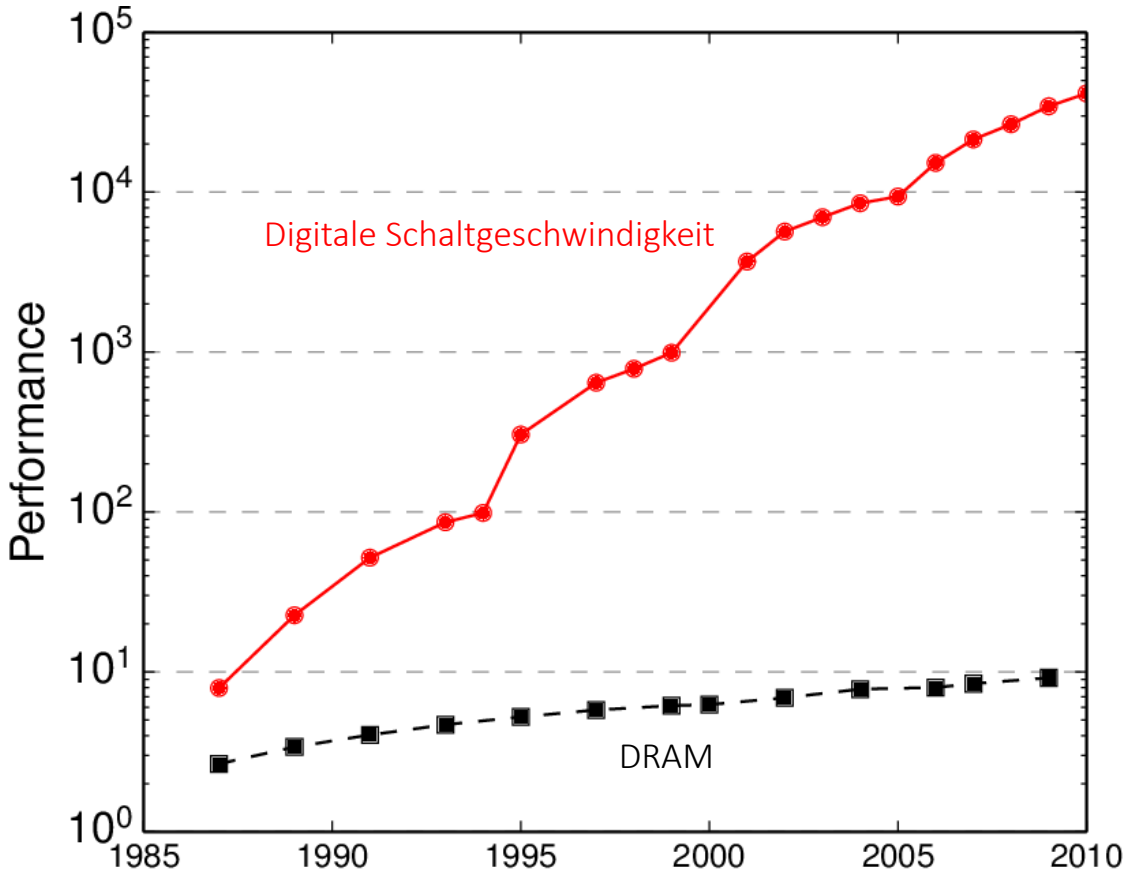
Caches

Emilio Pielsticker

Von-Neumann-Memory-Wall

Das **Memory-Wall Problem** beschreibt die im Laufe der Zeit immer größer werdende Lücke zwischen den Lese/Schreib-Geschwindigkeiten von DRAM-Speichern und der Geschwindigkeit sonstiger Mikroelektronik

Entwicklung der Geschwindigkeit von DRAMs



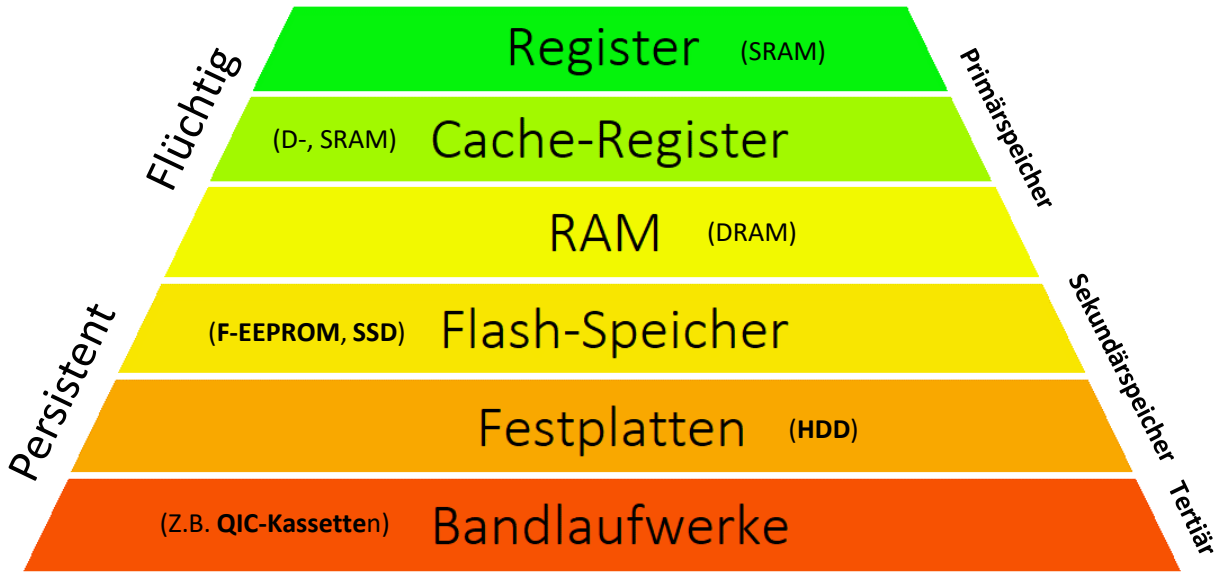
Das Memory-Wall Problem im Detail

- ⊕ Schaltgeschwindigkeiten wuchsen weitestgehend exponentiell, wie es auch das Moorsche Gesetz vorhersagen würde
- ⊖ Die Geschwindigkeitszunahmen von DRAM-Speicherbausteinen entwickelte sich deutlich langsamer mit einer zusätzlichen Stagnation

Flaschenhals

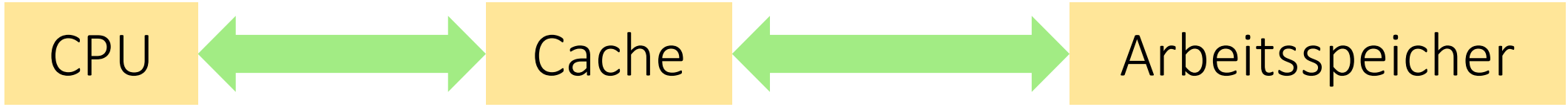
- Von einem **Flaschenhals** (engl. **Bottleneck**) spricht man in Kompositionen von Systemen genau dann, wenn Systeme mit hohen Durchsatz mit Systemen mit deutlich niedrigeren Durchsatz kombiniert werden
  → Durchsatz des Gesamtsystems sinkt!
- Das Memory-Wall Problem führt zu einem solchen Flaschenhals, dessen negativer Effekt durch die Speicherhierarchie und ein bedarfs-gesteuertes **Umlagern von Seiten** zwischen den einzelnen **Hierarchieebenen** stark reduziert werden kann

Speicherhierarchie



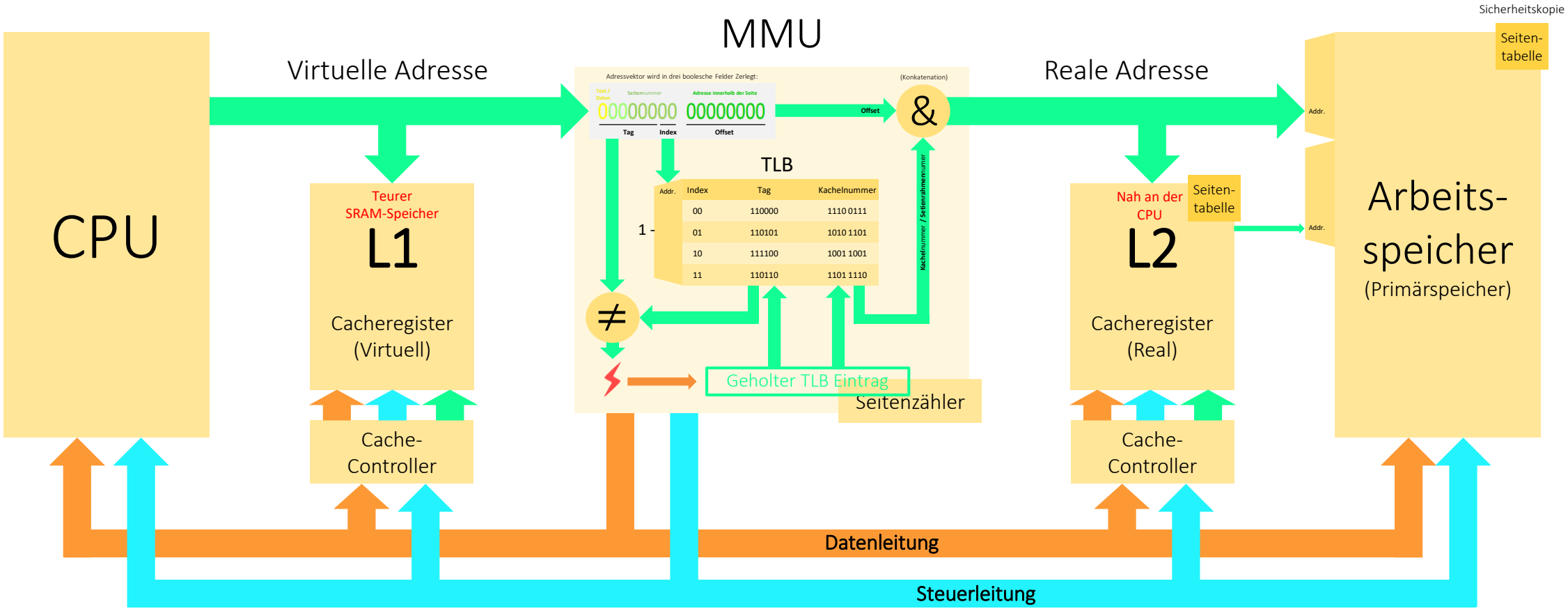
Caches

- Der **Cache** (franz. cacher, dt. versteckt) ist ein zusätzlicher Speicher mit hoher Zugriffsgeschwindigkeit und liegt zwischen RAM und Aktivitätsträger:



- Bei jedem Seitenaufruf wird überprüft, ob diese bereits im Cache liegt
  Falls ja, spricht man von einem **Cache Hit** und falls nein, wird dieser Fall als **Cache Miss** bzw. **Seitenfehler** bezeichnet.
- Inhalte häufig verwendeter Seiten werden in den **Cache** kopiert um bei dem nächsten Aufruf höhere Geschwindigkeiten erzielen zu können

Reale und virtuelle Caches

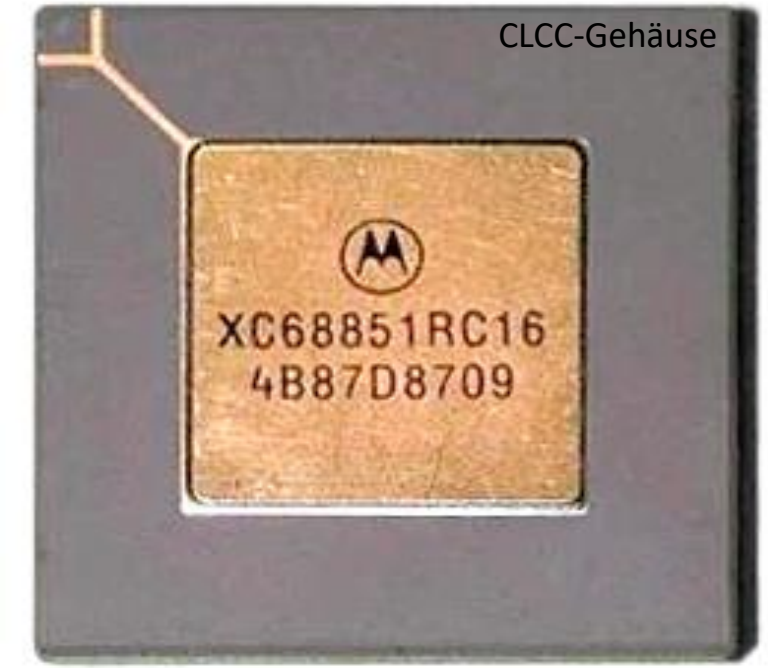


Cache und MMU in historischen Systemen

Prozessor mit L2 Cache
Intel Pentium II

L2 Caches sind auf die ICs in SMD-Bauform links und rechts verteilt

MMU (Assoziativ)
Motorola XC 68851



Cache-Zeile

- Die Wortbreite des Caches (also die Cache-Zeile c_l) muss nicht unbedingt der Wortbreite der restlichen Systemkomponenten entsprechen: Byte wird mittels Decoder aus Cache-Zeile gewählt
- Die kleinstmögliche Speichereinheit innerhalb eines Caches die physikalisch 'angesprochen' werden kann wird **Cache-Zeile** (engl. **Cache line**) genannt und besitzt die Länge c_l (in Byte)

Kenngrößen des Caches

Byte wird mittels Decoder aus Cache-Zeile gewählt:

- Die dazu benötigte Offset-Adresse besitzt die Bitbreite $\log_2 c_l$
- n_{cl} ist die Anzahl der maximal lagerbaren Cache-Zeilen
- $n_{\mathrm{cache}} = n_{\mathrm{cl}} \cdot c_l$ ist also die Speicherkapazität des Caches (in Byte)
- Meist gilt $n_{\mathrm{L1cache}} \ll n_{\mathrm{L2cache}} \ll \cdots \ll n_{RAM}$

Verwaltung von Caches

Caches werden ähnlich verwaltet wie TLBs und besitzen daher ähnliche Kenngrößen für die die folgenden Zusammenhänge gelten:

- Der Cache wird in **Arbeitsmengen** (engl. **Working sets** oder nur **Sets**) eingeteilt, von der jede einen Indexfeld besitzt: $\|\mathrm{Index}\| = \log_2 n_{\mathrm{sets}}$
- n_{sets} sei hierbei die Anzahl der Arbeitsmengen

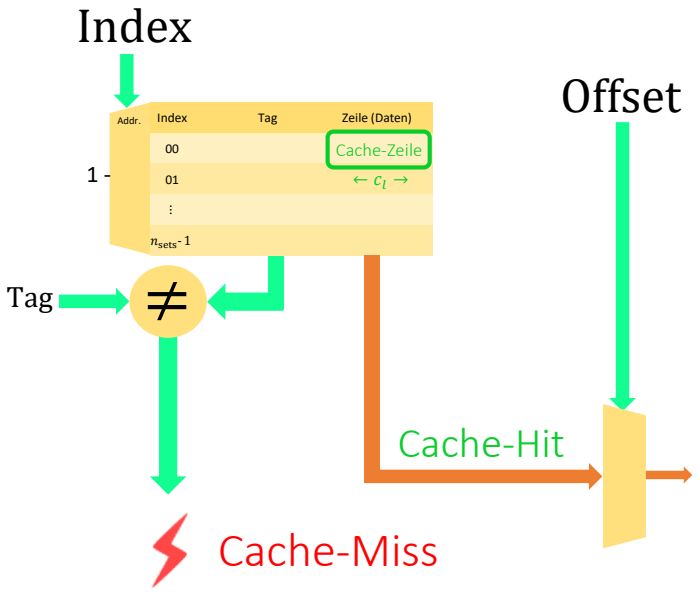
Verwaltung von Caches (Fortsetzung)

Caches werden häufig ähnlich verwaltet wie TLBs und besitzen daher ähnliche Kenngrößen für die folgenden Zusammenhänge gelten:

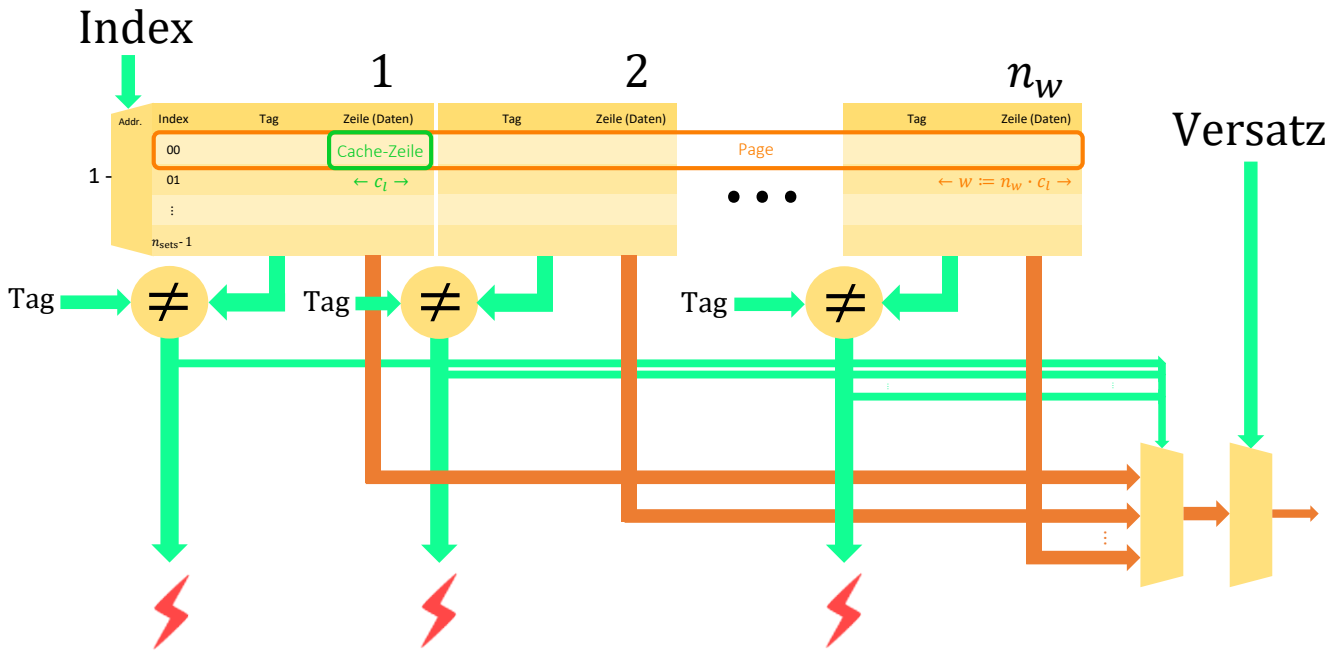
- Da bei größer angelegten Seiten das ständige vollständige Umladen von Seiten zwischen Hauptspeicher und Cache sehr kostenintensiv wäre, werden in Caches die Cache-Zeilen einzeln ein-/ausgelagert
- Tags eingehender Adressen werden mit den Tags der Arbeitsmengen verglichen und falls nicht vorhanden muss die angeforderte Zeile eingelagert werden

Arbeitsmenge voll: ERSETZUNGSPROBLEM

Verwaltung von Caches (Skizze)



Verwaltung von n_w-Way Caches (Skizze)



Kenngrößen des Caches (Berechnungsbeispiel)

Beispiel: Man stelle sich einen assoziativen 4-Weg Cache ($n_w = 4$) mit $n_{\text{sets}} = 256$ Arbeitsmengen und eine Adressbreite von $\log_2 n = 24$ vor:

- Die Größe einer Cache-Zeile beträgt hier $c_l = 4\ \text{B}$.
- Wie breit müssen Index, Tag und Offset sein?
- $\|\text{Index}\| = \log_2 n_{\text{sets}} = \log_2 256 = 8$
- $\|\text{Tag}\| = \log_2 n - \|\text{Index}\| - \log_2 c_l = 24 - 8 - 2 = 14$
- $\|\text{Offset}\| = \log_2 4 = 2$

Kontrollrechnung: $8 + 14 + 2 = 24$ ✓

Demand Paging und Seitenflattern

- Seitenflattern **verlangsamt** den ausgeführten Prozess
- **Speichermangel** im Primärspeicher
  ⇒ Seitenflattern
  ⇒ Prozessverlangsamung

Ursache und Auswirkung von Seitenflattern

- **Demand Paging:** Seiten bzw. Cache-Zeilen werden bei voll ausgelasteten Arbeitsspeicher auf deutlich langsamere, aber dafür deutlich größere (da günstigere) Speichermedien ausgelagert
- Ein ständiges Umladen von Seiten zwischen den unterschiedlichen Speichermedien, ist als **Seitenflattern** bzw. **Trashing** bekannt

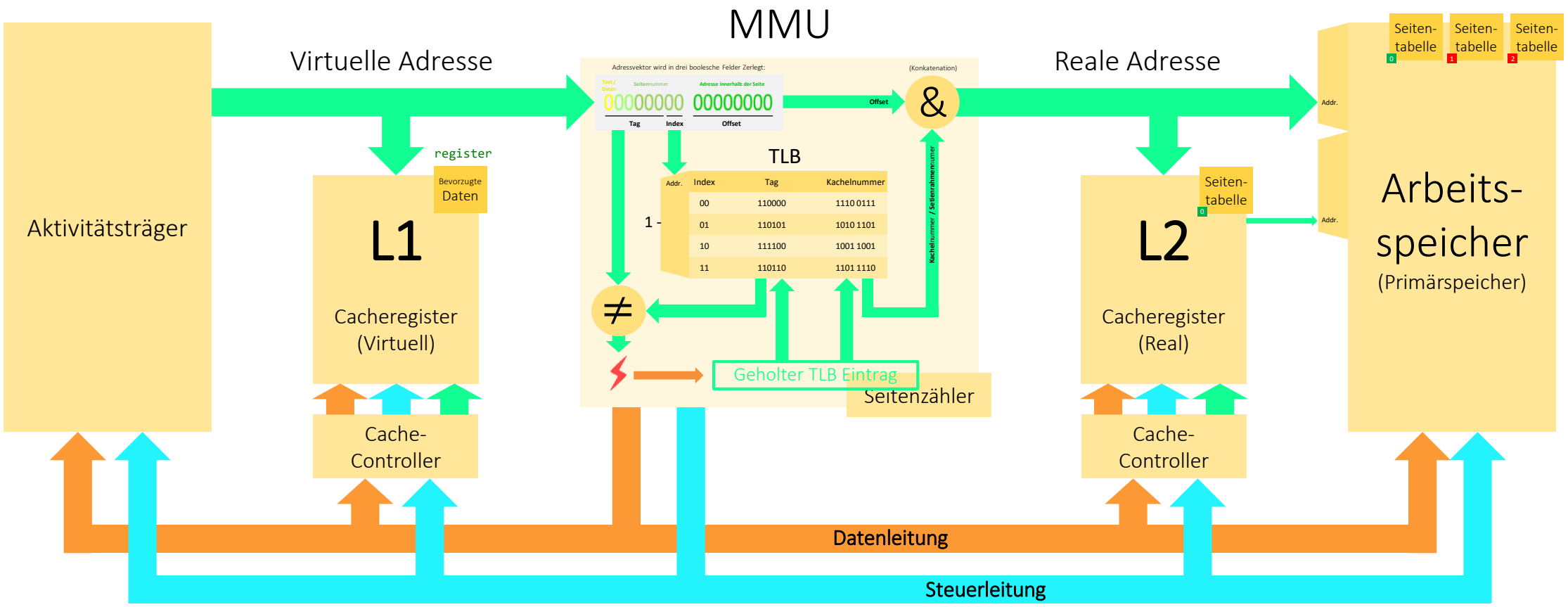
Organisation mehrerer Seitentabellen

- Der Arbeitsspeicher kann mehrere Seitentabellen beinhalten
  - **Organisation:** Eine Seitentabelle je Prozess
  - Alle Seitentabellen besitzen eine ID 0 (identisch zur PID) und ein Statusbit
- **Auswechseln von Seitentabellen:** Seitentabelle im Cache wird gesichert und mit einer anderen Seitentabelle im Arbeitsspeicher überschrieben

Kontextwechsel

- Die Speicherverwaltung kann Hardware-Schnittstellen (in Form von Hardwareregister) zur Verfügung stellen um den Betriebssystem das auswechseln von Seitentabellen (→ **Prozesswechsel**) zu ermöglichen
- Bei einem Prozesswechsel werden meistens alle Einträge aus Cache und TLB ersetzt, was als **Kontextwechsel** bezeichnet wird

Arbeitsspeicherarchitektur



Notizblockspeicher (Scratchpad-Memory)

- Bei dem **Notizblockspeicher** (engl. **Scratchpad memory**, kurz **SPM**) handelt es sich um einen unverwalteten schnellen Speicher, der virtuell in den Adressraum des Hauptspeichers abgebildet wird
- Diese Abbildung wird durch einen möglichst einfach gestalteten Adressdecoder bewerkstelligt
- Auf den Notizblockspeicher kann der Aktivitätsträger direkt zugreifen

SPM und Cache im Vergleich

Notizblockspeicher (SPM)

- ⊖ Die manuelle Verwaltung ist allerdings auch fehleranfälliger und erfordert zusätzliche Expertise seitens des Programmautors
- ⊕ Die Hardware des SPM ist einfach gehalten und kann mit sehr geringen Latenzzeiten sowohl gelesen als auch beschrieben werden

Cache

- ⊕ Das Einlagern von Zeilen passiert weitgehend automatisch ohne Zutun des Programmierers
- ⊖ Caches arbeiten mit komplexer zusätzlicher Hardware und sind langsamer als der SPM