

# Translation Lookaside Buffer

Technische Universität München

Grundlagenpraktikum Rechnerarchitektur

19. August 2024

Gruppe 151

Lukas Wolf

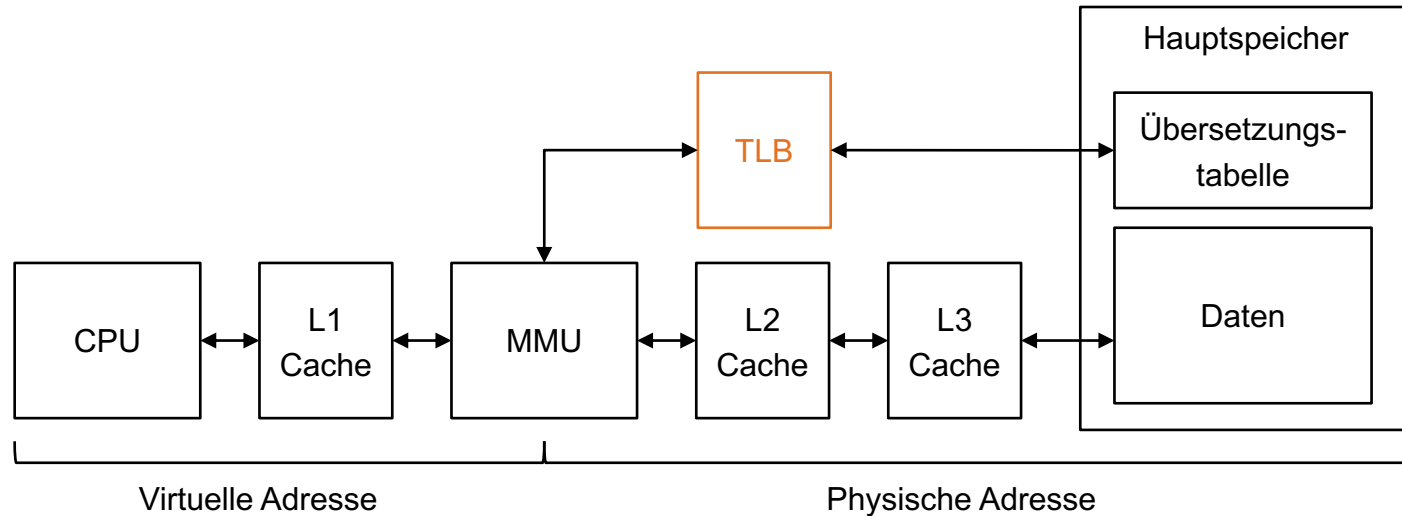
Elena Reinbold Fraire

Jonah Zabel



# Was ist ein TLB

Levelled Cache Architektur in einem Prozessor



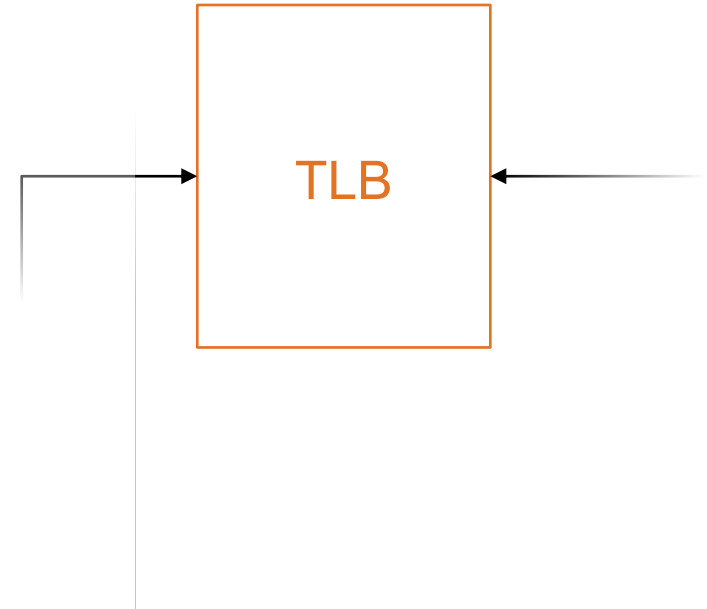
# Was ist ein TLB

Simulation mittels SystemC

Rahmenprogramm in C-Programmiersprache

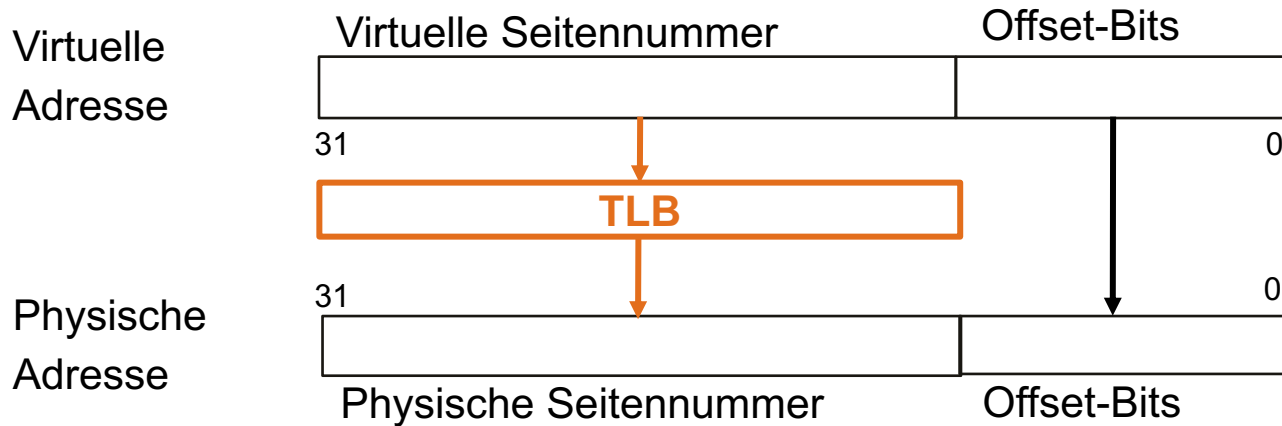
Ziele der Simulation:

- Auswirkungen auf Hauptspeicherzugriffe?
- Welche Bauart liefert optimale Ergebnisse?
- Performanzverbesserung?



# Unser Lösungsansatz

## Modul 1: Berechnung der Adresse



# Unser Lösungsansatz

## Modul 1: Berechnung der Adresse

Virtuelle Seitennummer	Physische Seitennummer
0x0	0x0
0x56	0x78
0x783	0x956
0x3	0x7

# Unser Lösungsansatz

## Modul 1: Berechnung der Adresse

### Wichtige Abstraktionen:

- Speicherung der im TLB enthaltenen virtuellen Seitennummern in einem `std::vector`
- Abfangen von Cold Misses durch ein `std::set`
- Page Table durch eine einfache Berechnung ersetzen
- Ergebnis direkt berechnen und danach die benötigten Clock-Zyklen durchlaufen lassen

# Unser Lösungsansatz

## Modul 2: Speicherung der Daten

### Wichtige Abstraktionen:

- Simulation des Speichers durch eine `std::map`
- Auch hier: Ergebnis direkt berechnen und danach die benötigten Clock-Zyklen durchlaufen lassen

# Unser Lösungsansatz

## Modul 3: Requests abarbeiten

- Synchronisation der beiden Untermodule
- Die Requests in dem übergebenen Array nacheinander abarbeiten → durch for-Schleife abstrahiert
- Anzahl der benötigten Gatter berechnen
- Abstraktion: fast alles → gehört nicht zur Logik, sondern zur Messung der Effizienz



# Berechnung der Gatteranzahl

$$(2 * (32 - \log(\text{blocksize})) + 1) * 4 * \text{tlb\_size} + 182$$

Anzahl Bits pro Zeile

Anzahl Gatter für die Speicherung von einem Bit

Anzahl Zeilen insgesamt

Anzahl Gatter für einen Subtrahierer

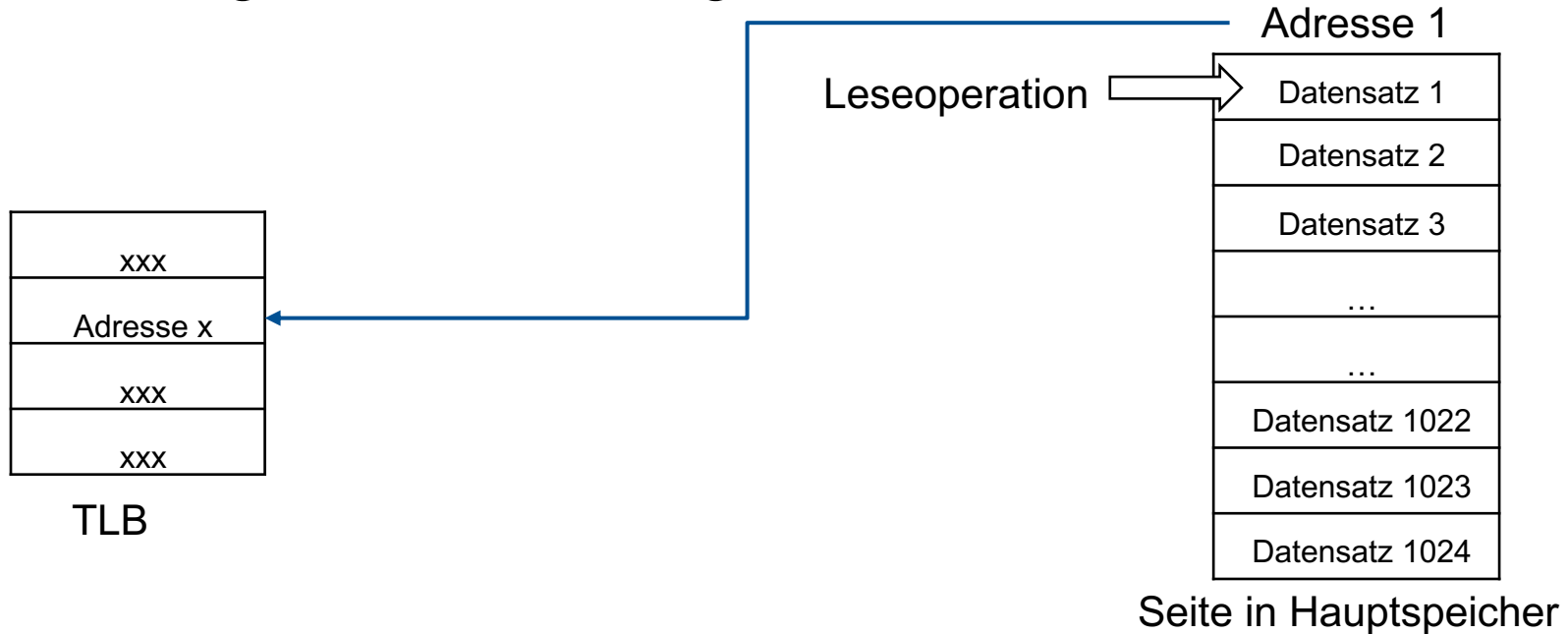
# Übliche Größen eines TLB und des Hauptspeichers

- Größe: 16 – 512 Einträge
- Größe eines geladenen Blockes: 1 Seite (4096 Bytes)
- Hit-Rate: 99 % - 99,9 %
- Assoziativität: Voll- / Mengenassoziativ
- Zugriffszeit: ~ 1 Zyklus
  
- Zugriffszeit Hauptspeicher: 100 Zyklen

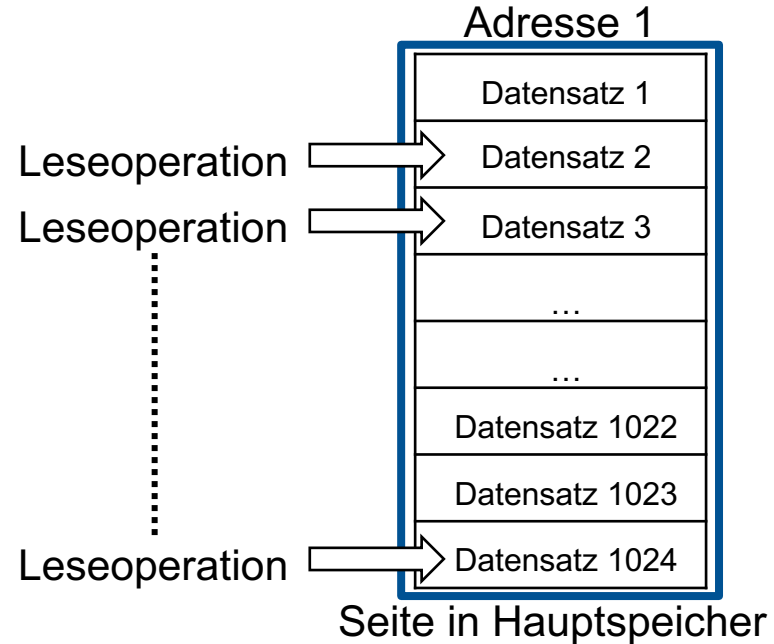
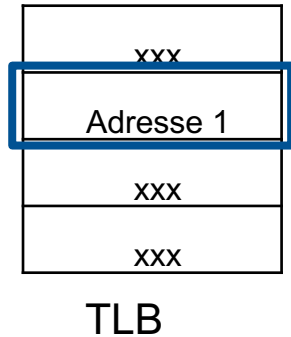
# Speicherzugriffsverhalten über eine verkettete Liste

- Auswirkung der Größe des TLB auf die Auswirkungszeit
- Verkettete Liste: Verweis auf Daten

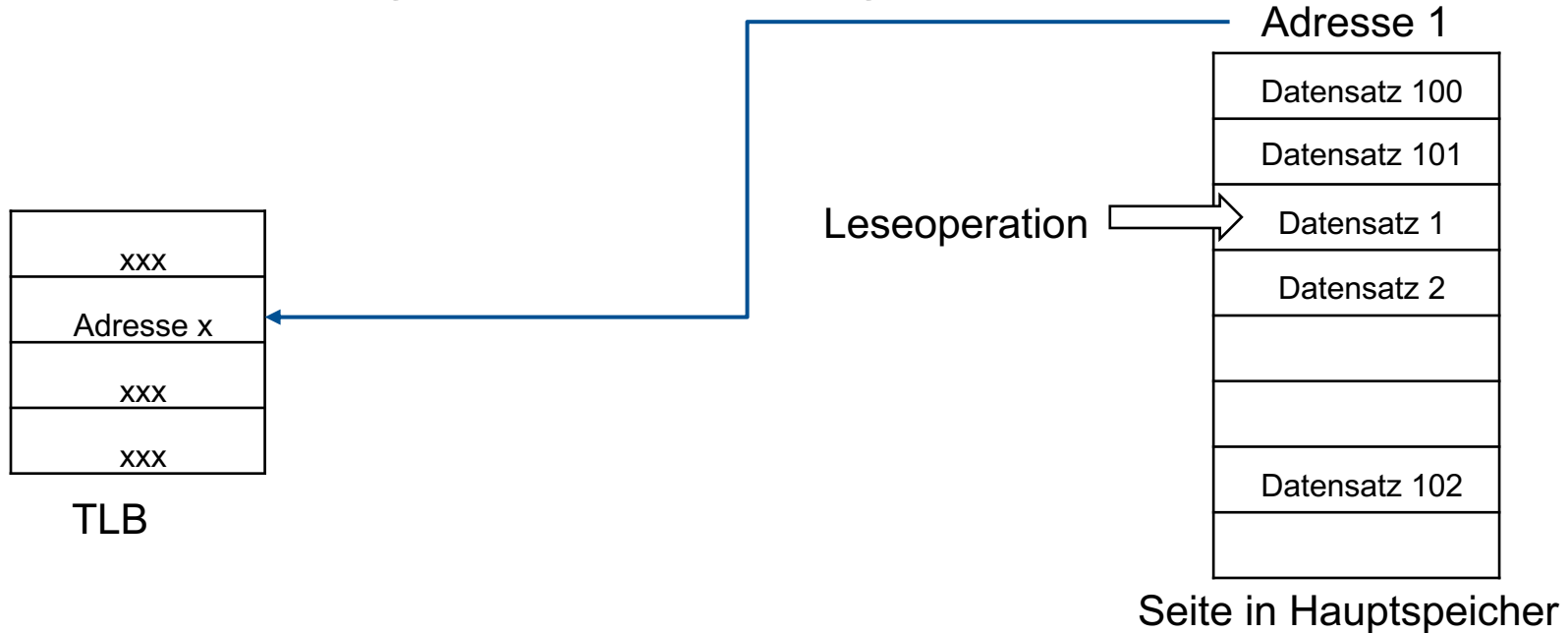
# Chronologischer Datenzugriff



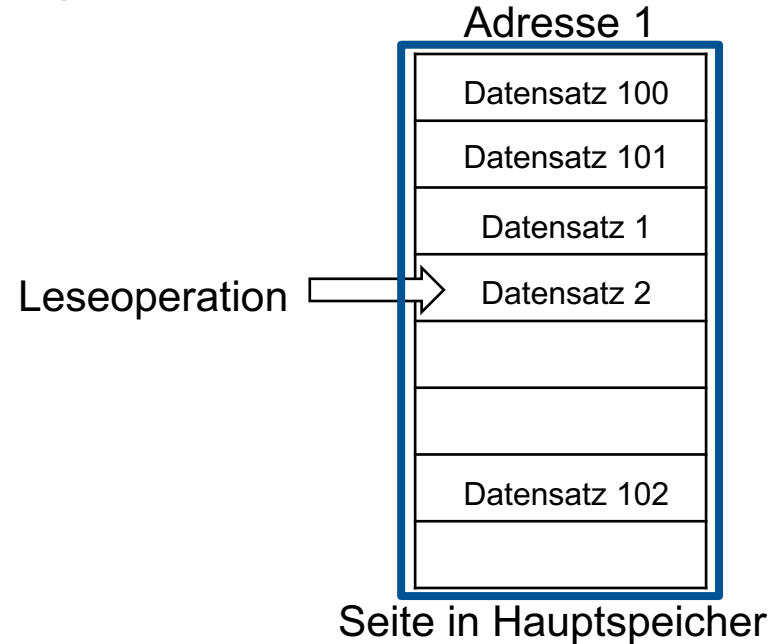
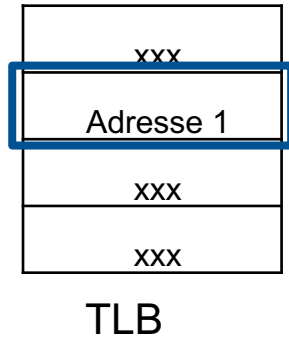
# Chronologischer Datenzugriff



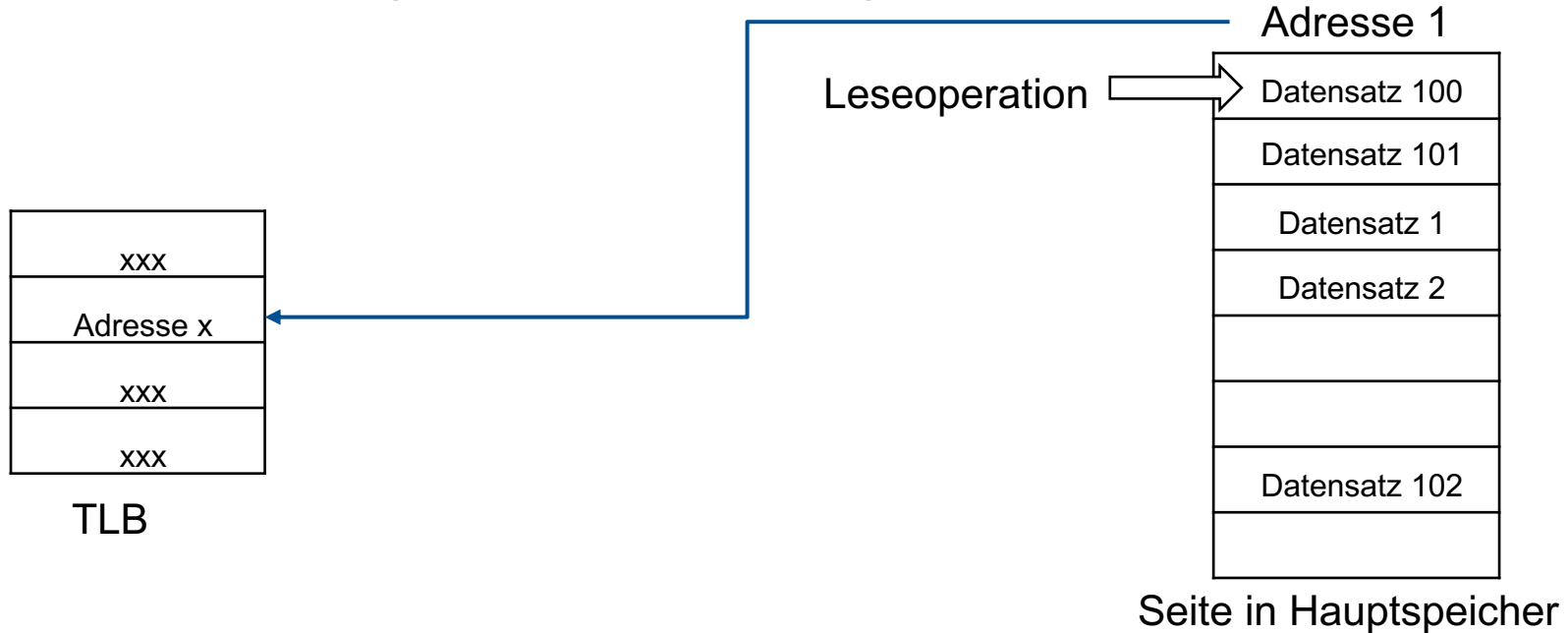
# Nicht chronologischer Datenzugriff



# Nicht chronologischer Datenzugriff








# Nicht chronologischer Datenzugriff



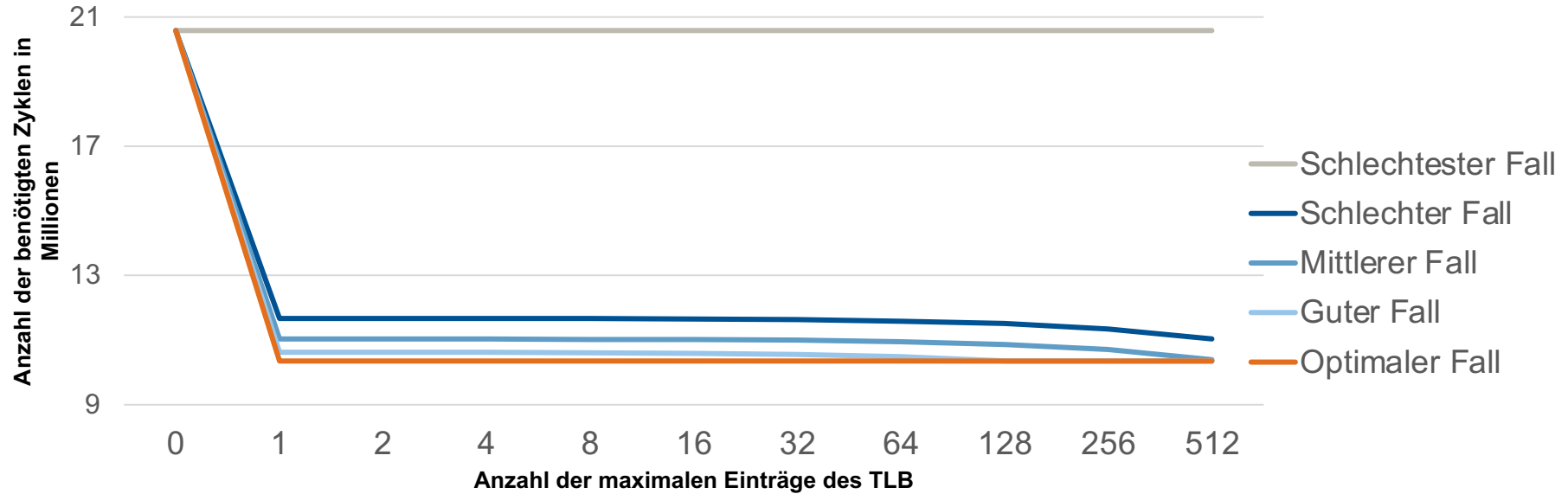


# Messung des TLB's

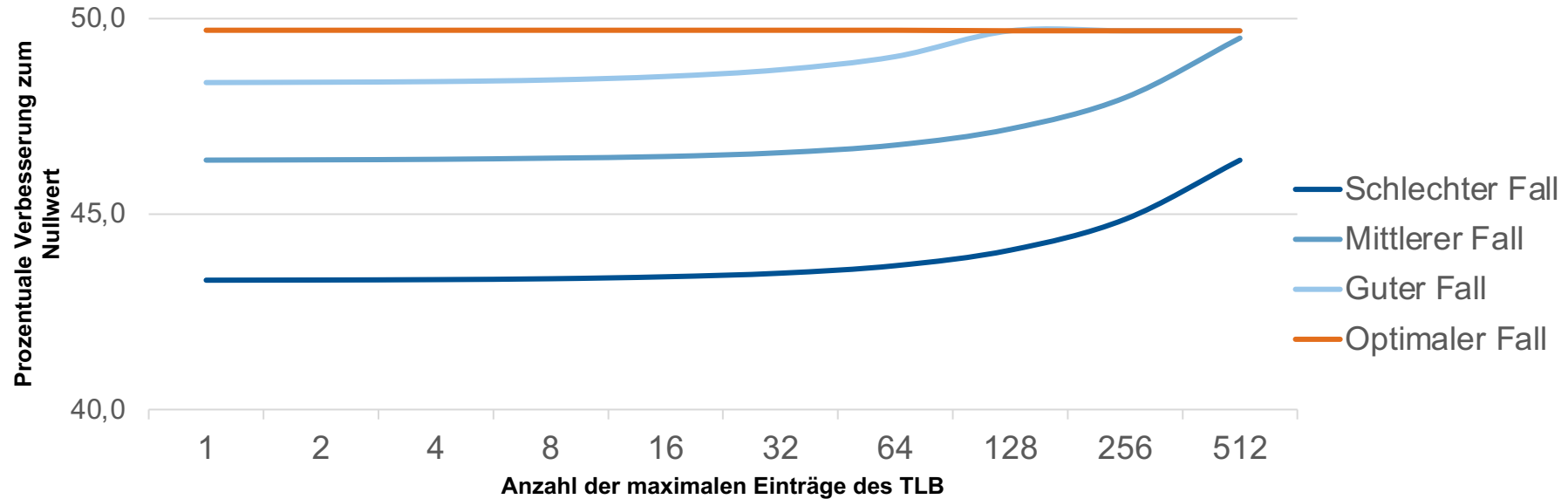
Jeweils 102400, 32-Bit Integer Werte

	Anzahl der Seiten	Ø-Anzahl von Einträgen pro Seite
Optimale Verteilung 	100	1024, chronologisch
Gute Verteilung 	128	800, nicht chronologisch
Mittlere Verteilung 	512	200, nicht chronologisch
Schlechte Verteilung 	1024	100, nicht chronologisch
Schlechteste Verteilung 	102400	1

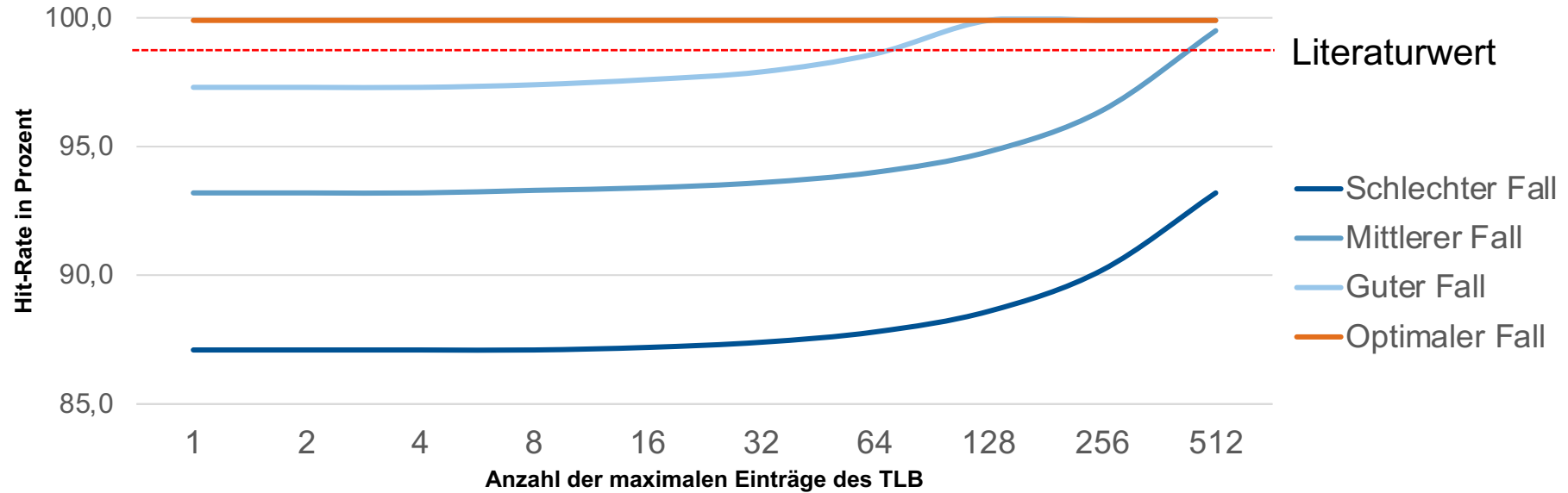
# Anzahl der benötigten Zyklen



# Anzahl der eingesparten Zyklen



# Hit-Rate

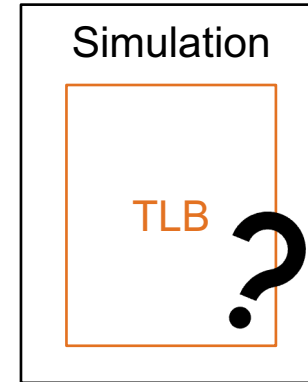


# Korrektheit

## TLB Simulation

### Diskrepanz in Hit-Rate?

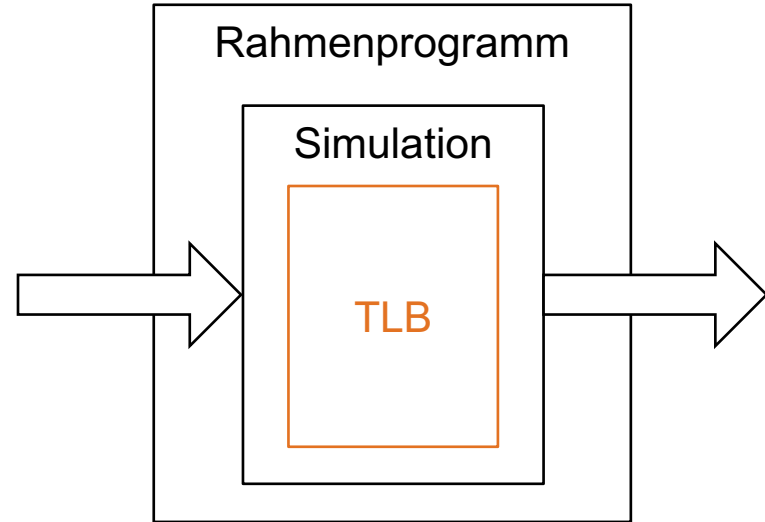
- Abhängig vom Memory Management
- Direct Mapped-Cache nicht optimal
- Keine Heuristik für Ersetzungsstrategie



# Korrektheit

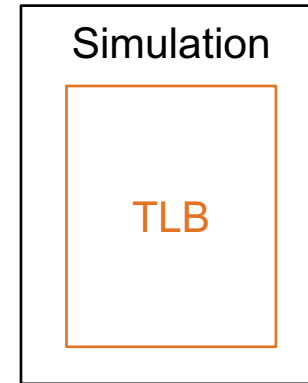
## Rahmenprogramm

- Input-Fuzzing für Startargumente
- Request-Fuzzing für Lese/Schreibzugriffe
- Verifizierung der Ausgegeben Werte
- Benutzung des Request-Generation-Tools



# Ergebnisse

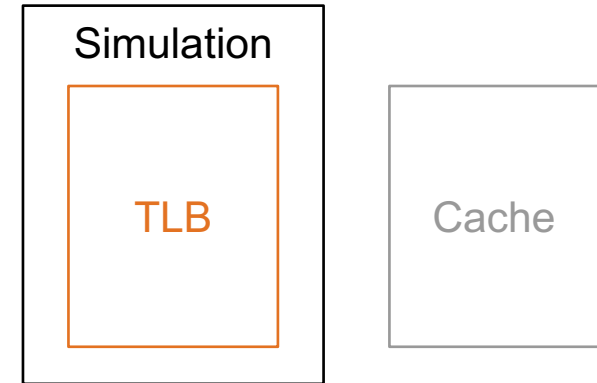
- Bauweise und vom TLB nicht sehr bedeutend
- Insbesondere Größe des TLB (bei Directly Mapped)
- Wichtig ist Verteilung der Übersetzungszugriffe
- Fokus auf gutes Memory Management
- Ebenfalls Cache Optimierte Programmierung



# Ergebnisse

- Ergänzendes Cache zum aktuellen System
- Wichtig ist Kombination mit Daten Cache
- Schon bei kleinen Größen sehr effizient
- Dementsprechend geringe Gatterkosten beim Verbau
- Geeignete Cache Assoziativität verwenden
- Verdrängungs-Heuristiken verwenden

Simulation dementsprechend erweitern,  
um realitätsnähere Ergebnisse zu erhalten





# Literatur

- M. Schulz (Vorlesung, Einführung in die Rechnerarchitektur, 2023).
- Patterson, D. A. (2009). Computer Organization and Design: The Hardware/Software Interface. (4th ed.). Morgan Kaufman.
- Arpaci-Dusseau, R. H. (2023). Operating Systems: Three Easy Pieces. Arpaci-Dusseau Books.
- R. Wille (Vorlesung, Einführung in die Rechnerarchitektur, 2023).

# Translation Lookaside Buffer

Technische Universität München

Grundlagenpraktikum Rechnerarchitektur

19. August 2024

Gruppe 151

Lukas Wolf

Elena Reinbold Fraire

Jonah Zabel

