Caches

Eine Einführung in Caches und ihre Verdrängungsalgorithmen

Inhaltsübersicht

- Was ist ein Cache überhaupt?
- Virtueller Cache
- Wozu braucht man Cache-Algorithmen?
- Was sind Cache misses und Cache hits?
- Der optimale Cache-Algorithmus

Inhaltsübersicht

- Statische Algorithmen
- Algorithmen im Gebrauch (workloads)
- Konklusion aus den workloads
- Dynamische Algorithmen

Was ist ein Cache überhaupt?

- Magnetbänder
- Mehr als 6TB pro Band
- ~4€ pro TByte an Speicher
- Bis zu 30 Jahre lagerbar (ohne Strom)
- Datenrate bis zu 280 MB/s

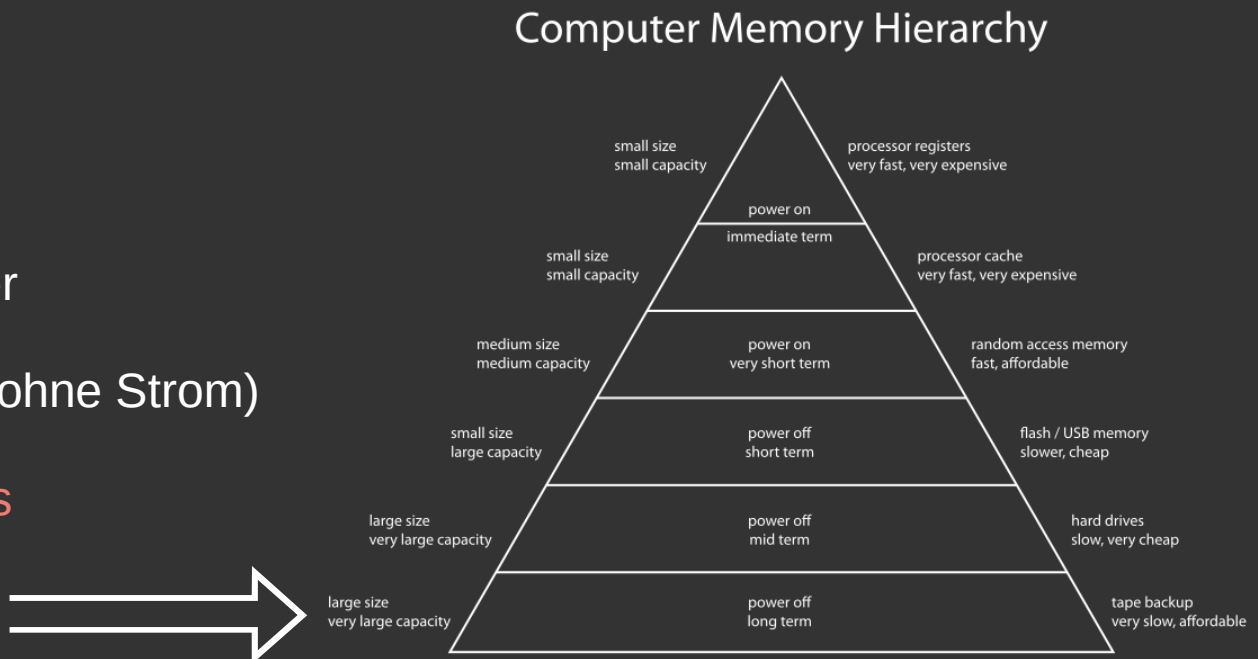


Fig. 1. https://de.wikipedia.org/wiki/Speicherhierarchie

Was ist ein Cache überhaupt?

- Festplatten HDD
- Mehr als 2TB pro Festplatte
- ~30€ pro TB an Speicher
- Bis zu 1 Jahr lagerbar (ohne Strom)
- Datenrate bis zu 200 MB/s
- Erlauben wahlweisen Zugriff im Vergleich zu Magnetbändern

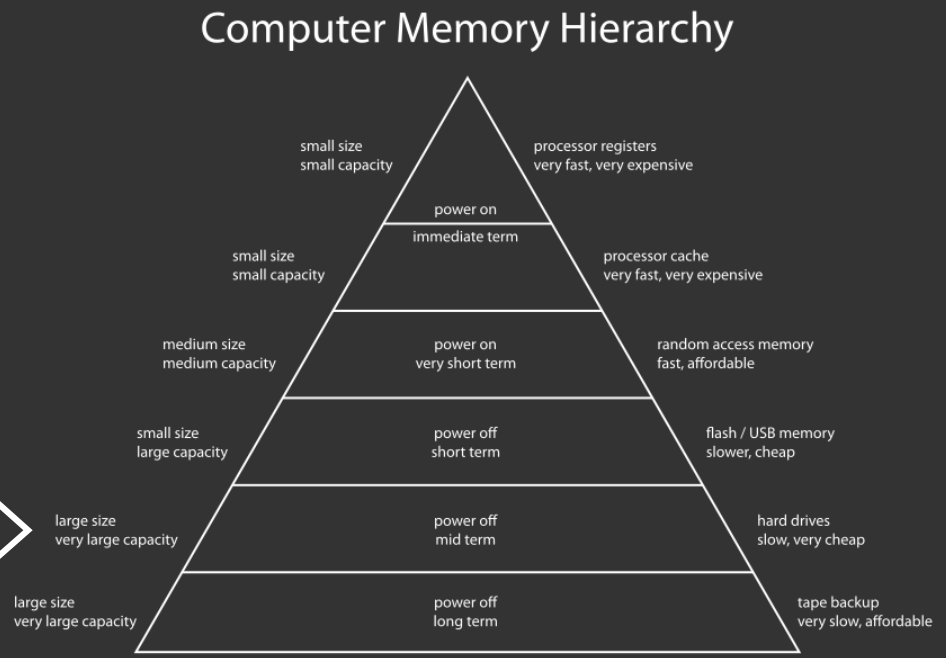


Fig. 1. https://de.wikipedia.org/wiki/Speicherhierarchie

Was ist ein Cache überhaupt?

- Flashspeicher SSD
- Mehr als 500GB pro Platte
- ~160€ pro TB an Speicher
- Bis zu 1 Jahr lagerbar (ohne Strom)
- Datenrate bis zu 600 MB/s

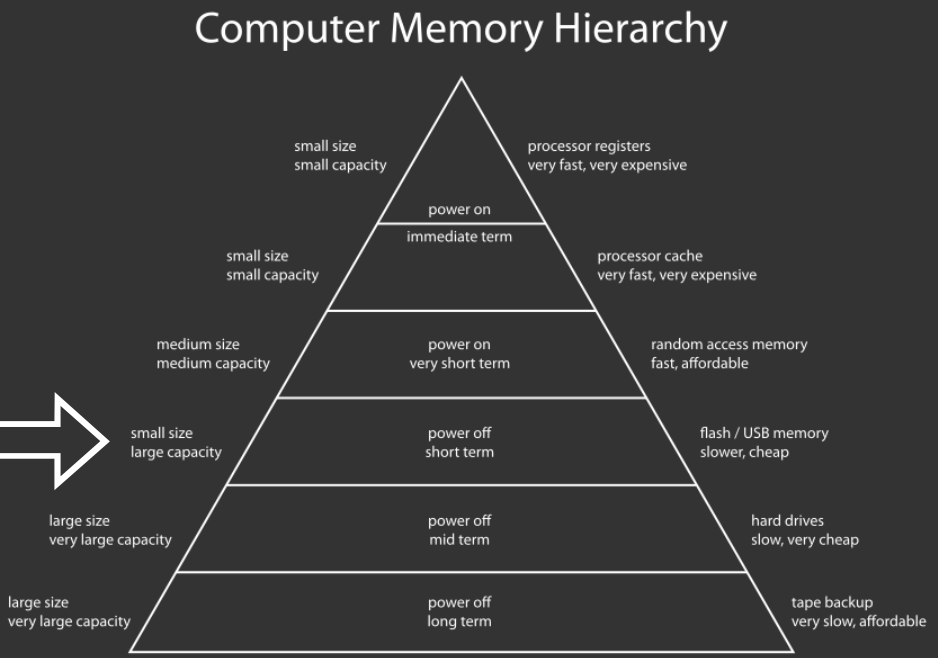


Fig. 1. https://de.wikipedia.org/wiki/Speicherhierarchie

Was ist ein Cache überhaupt?

- RAM Speicher
- Mehr als 8GB pro Riegel
- ~8500€ pro TB an Speicher
- Nicht ohne Strom lagerbar
- Datenrate bis zu 3200 MB/s (x2)

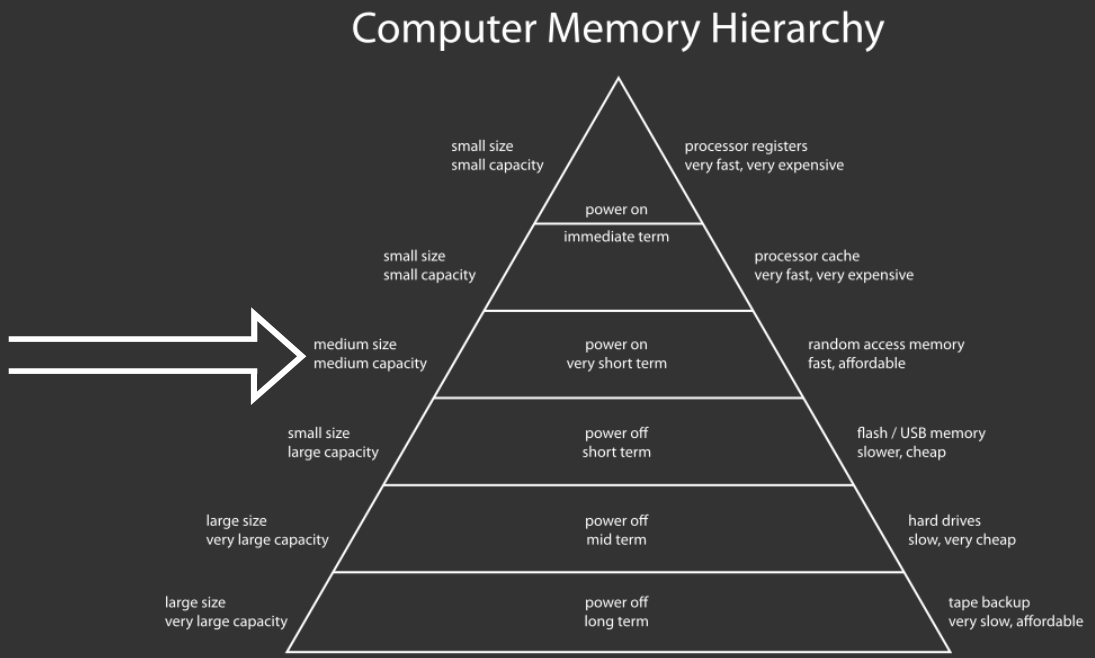


Fig. 1. https://de.wikipedia.org/wiki/Speicherhierarchie

Was ist ein Cache überhaupt?

- Prozessor Cache
- Intel i5-8600k hat 9MB Cache
- Preis… hoch
- Nicht ohne Strom lagerbar
- Wenige Nanosekunden Zugriffszeit

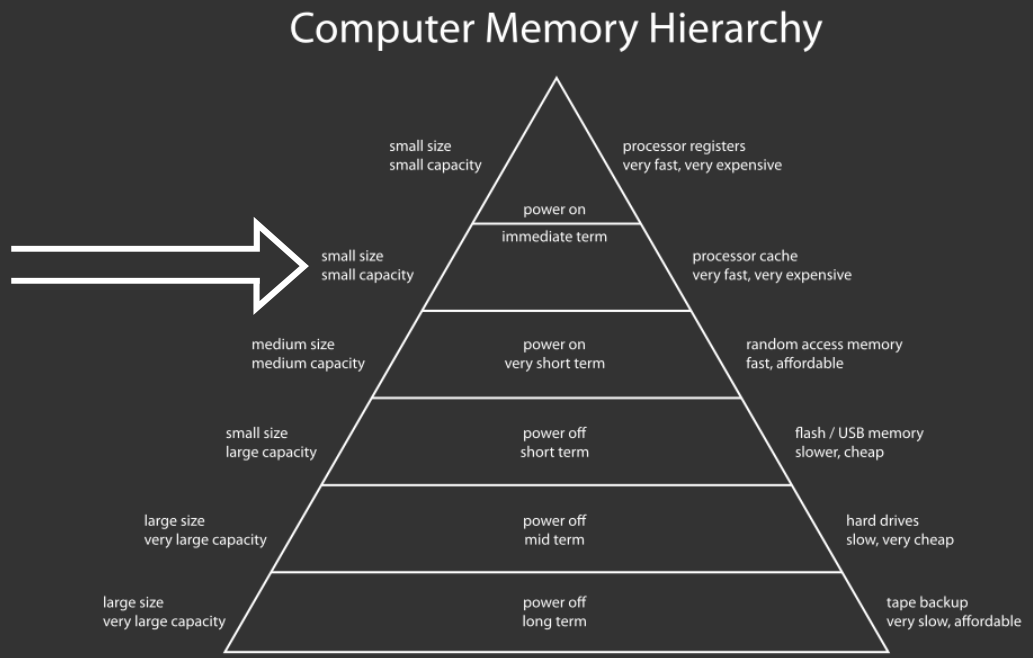


Fig. 1. https://de.wikipedia.org/wiki/Speicherhierarchie

Was ist ein Cache überhaupt?

- Register
- Nicht ohne Strom lagerbar
- Zugriffszeit = Taktfrequenz
  3200MHz =
  3.200.000.000 Zugriffe pro Sekunde
- ~3 Zugriffe pro Nanosekunde

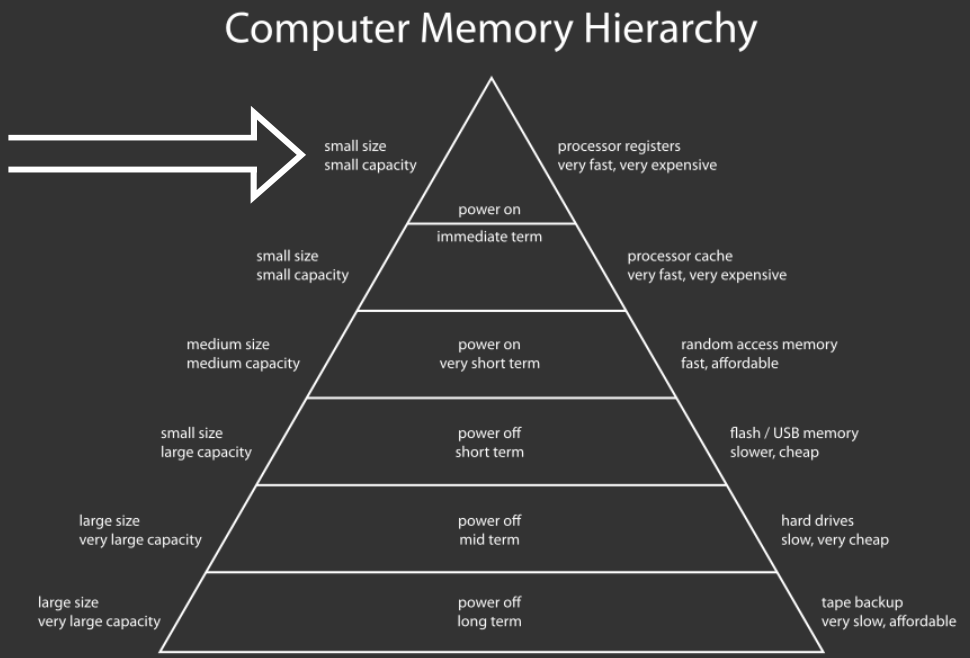


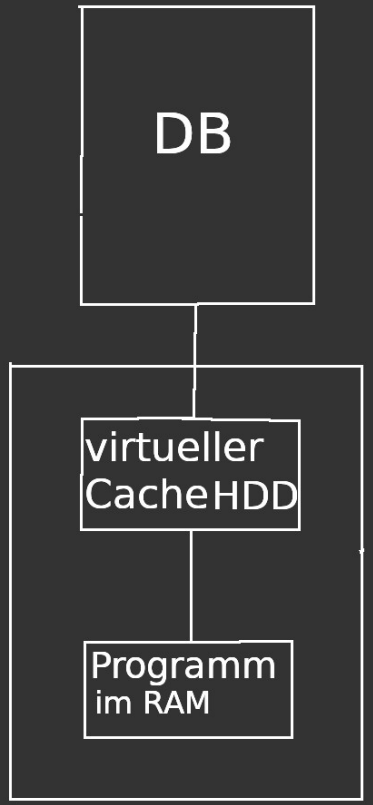
Fig. 1. https://de.wikipedia.org/wiki/Speicherhierarchie

Was ist ein Cache überhaupt?

- Prozessor Cache ist nicht der einzige Cache in einem Rechner
- Speichermedien verfügen über zusätzlichen schnelleren Speicher (Cache) für die Speicher „über“ ihnen
- Beispielsweise sind viele Festplatten mit zusätzlichem Speicher ausgestattet, der in etwa so schnell ist wie RAM-Speicher
- Es gibt auch virtuellen Cache

Virtueller Cache

- Virtueller Cache ist normaler Speicher auf der Festplatte
- Wird z.B. als Buffer für Datenbanken oder Internetseiten genutzt
- Speichert Informationen für Webseiten oder Datenbankeinträge, damit diese nicht extra angefordert werden müssen
- Daher die „Weisheit“: Leer deinen Cache, dann funktioniert es!
  - Veraltete Daten bleiben im Cache, bis sie gelöscht werden



Wozu braucht man Cache-Algorithmen?

- Cache Speicher ist sehr schnell, aber auch sehr teuer
- Bei großen Programmen unmöglich, genug Speicher zu bekommen

- 15GB Festplattenspeicher
- 2GB RAM empfohlen

Fig. 2. https://basic-tutorials.de/counter-strike-global-offensive-server-unter-linux-installieren/

Wozu braucht man Cache-Algorithmen?

- Speicher muss oft geleert und befüllt werden
- Cache-Algorithmen sollen diesen Prozess optimieren
  - Wichtige Daten bleiben im Cache
  - Unwichtige Daten werden verworfen
  - (Im Idealfall!)
- Idealfall ist die große Ausnahme und nicht die Regel
- Cache misses so gut es geht reduzieren

Cache misses und hits

- Benötigte Daten sind bereits im Cache ⟹ Cache hit
- Benötige Daten sich nicht im Cache ⟹ Cache miss

- Angenommen Zugriffszeit auf den Cache der Festplatte ist 100ns und Zugriffszeit auf die Festplatte ist 10ms

Cache misses und hits

- Cache 100ns
- Festplatte 10ms

AMAT
Average Memory Access Time

- 90% hit = 100ns + 0.1*10ms = 1.0001ms
- 95% hit = 100ns + 0.05*10ms = 0.5001ms
  - Doppelt so schnell
- 99.9% hit = 100ns + 0.001*10ms = 0.0101ms
  - Etwa 100x schneller als 90% hit

Der optimale Cache-Algorithmus

- Cache ist anfangs leer $\Longrightarrow$ misses
- Tausche die Page aus, die erst nach allen anderen wieder gebraucht wird

Blick in die Zukunft nicht möglich!

- Eignet sich jedoch gut als Richtwert
- Hits(6)/Miss+Hit(11) = 54.5%

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
1	Miss		0, 1
2	Miss		0, 1, 2
0	Hit		0, 1, 2
1	Hit		0, 1, 2
3	Miss	2	0, 1, 3
0	Hit		0, 1, 3
3	Hit		0, 1, 3
1	Hit		0, 1, 3
2	Miss	3	0, 1, 2
1	Hit		0, 1, 2

Fig. 3. Beyond Physical Memory: Policies (Seite 3 Fig. 22.1)

FIFO

- Cache ist anfangs leer $\Longrightarrow$ misses
- Tausche die Page aus, die am längsten im Cache ist

  \+ Sehr leicht zu implementieren

  - Kein intelligenter Algorithmus
  - Hit Rate hängt stark vom Programm ab

36,36% Hit << 54,5% Hit

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		First-in→ 0
1	Miss		First-in→ 0, 1
2	Miss		First-in→ 0, 1, 2
0	Hit		First-in→ 0, 1, 2
1	Hit		First-in→ 0, 1, 2
3	Miss	0	First-in→ 1, 2, 3
0	Miss	1	First-in→ 2, 3, 0
3	Hit		First-in→ 2, 3, 0
1	Miss	2	First-in→ 3, 0, 1
2	Miss	3	First-in→ 0, 1, 2
1	Hit		First-in→ 0, 1, 2

Fig. 4. Beyond Physical Memory: Policies (Seite 5 Fig. 22.2)

Random

- Cache ist anfangs leer $\Longrightarrow$ misses
- Tausche eine zufällige Page aus

  \+ Ebenfalls sehr leicht zu implementieren

  - Wie FIFO auch kein intelligenter Algorithmus
  - Hit Rate ist komplett willkürlich

45,45% Hit < 54,5% Hit

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
1	Miss		0, 1
2	Miss		0, 1, 2
0	Hit		0, 1, 2
1	Hit		0, 1, 2
3	Miss	0	1, 2, 3
0	Miss	1	2, 3, 0
3	Hit		2, 3, 0
1	Miss	3	2, 0, 1
2	Hit		2, 0, 1
1	Hit		2, 0, 1

Fig. 5. Beyond Physical Memory: Policies (Seite 6 Fig. 22.3)

LRU – der perfekte Algorithmus?

- Cache ist anfangs leer $\Longrightarrow$ misses
- Tausche die Page aus, die am längsten nicht benutzt wurde

  \+ Zieht vergangenes Verhalten in Betracht

  \- Implementation nicht so simpel wie FIFO und Random

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		LRU→ 0
1	Miss		LRU→ 0, 1
2	Miss		LRU→ 0, 1, 2
0	Hit		LRU→ 1, 2, 0
1	Hit		LRU→ 2, 0, 1
3	Miss	2	LRU→ 0, 1, 3
0	Hit		LRU→ 1, 3, 0
3	Hit		LRU→ 1, 0, 3
1	Hit		LRU→ 0, 3, 1
2	Miss	0	LRU→ 3, 1, 2
1	Hit		LRU→ 3, 2, 1

Fig. 6. Beyond Physical Memory: Policies (Seite 7 Fig. 22.5)

54,5% Hit = 54,5% Hit

Workloads [No-Locality]

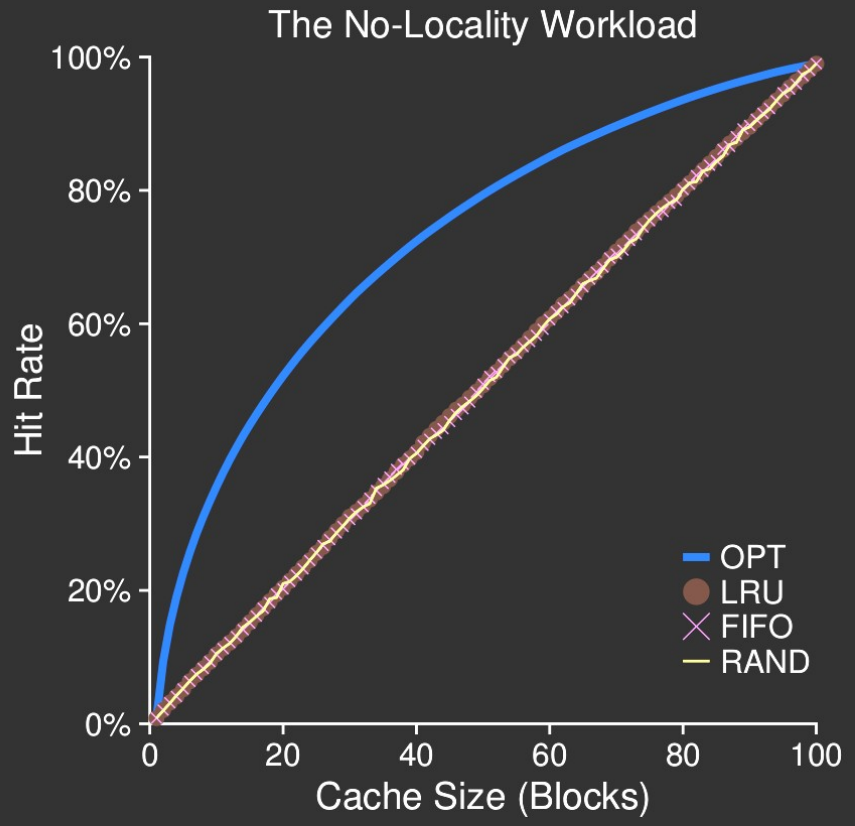


Fig. 7. Beyond Physical Memory: Policies (Seite 9 Fig. 22.6)

100 verschiedene Pages
10.000x wurden diese Pages abgefragt

No-Locality (neue Pages sind zufällig)

- Wenig überraschend verhalten sich alle Algorithmen gleich, da die Pages willkürlich bestimmt werden
- LRU braucht eine Historie, die auf die Zukunft schließen lässt, um zu wirken

Workloads [80-20]

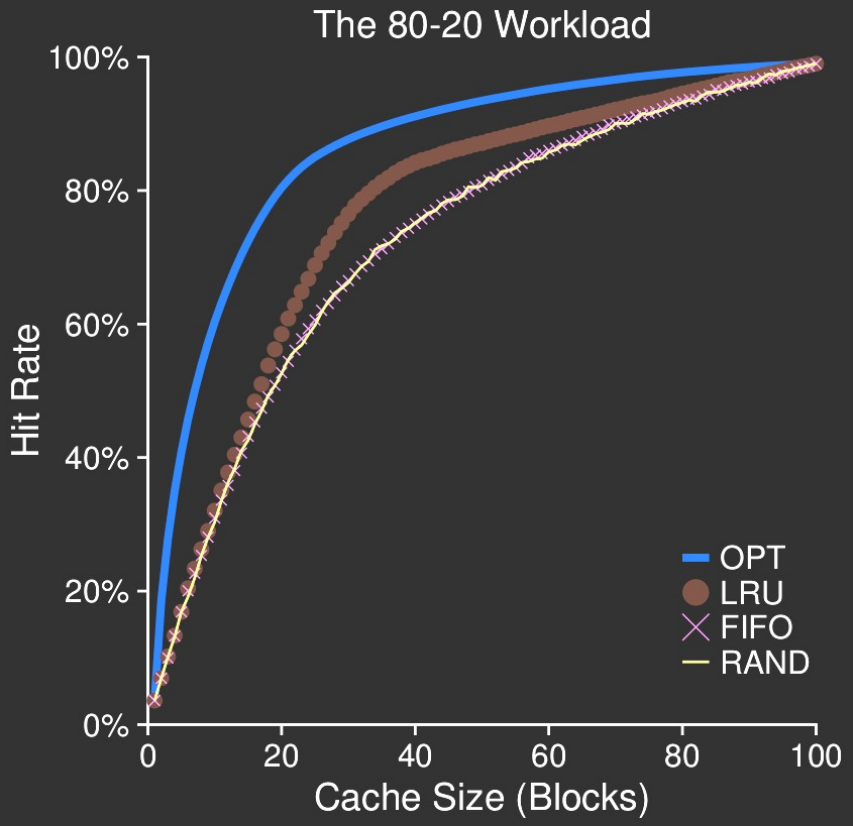


Fig. 8. Beyond Physical Memory: Policies (Seite 10 Fig. 22.7)

100 verschiedene Pages
10.000x wurden diese Pages abgefragt

80-20 (ein Programm nutzt zu 80% die gleichen 20% der Pages)

- LRU hat einen deutlichen Vorteil, da es ein Muster in der Benutzung erkennt
- FIFO und RANDOM schlagen sich trotzdem relativ gut

Workloads [Looping sequential]

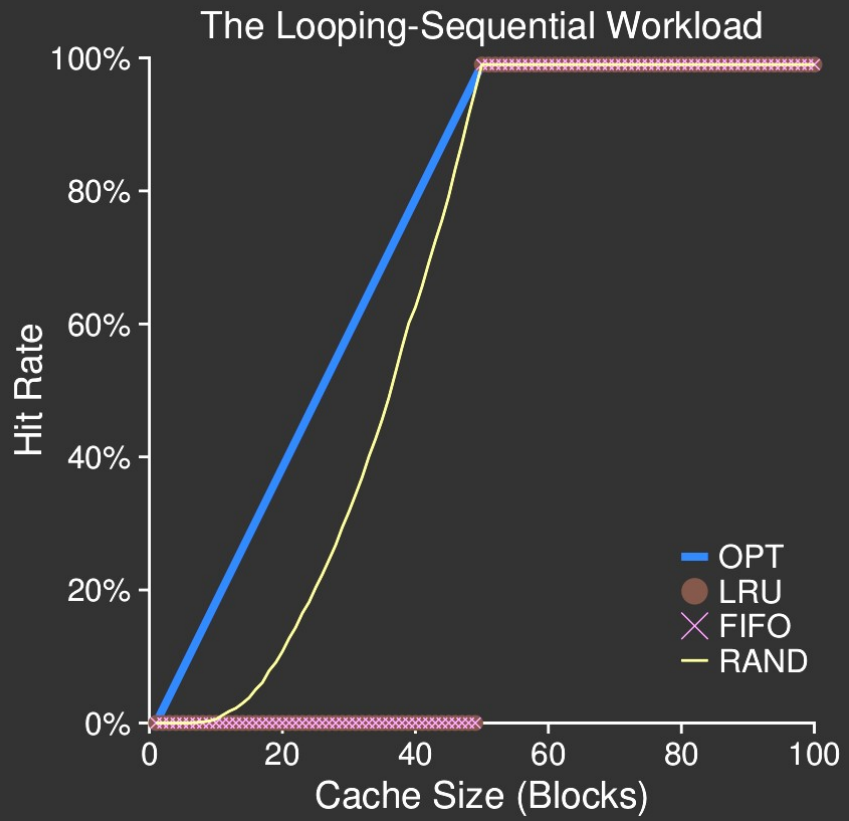


Fig. 9. Beyond Physical Memory: Policies (Seite 11 Fig. 22.8)

50 verschiedene Pages
10.000x wurden diese Pages abgefragt

Looping sequential
(ein Programm nutzt 50 Pages der Reihe nach und fängt von vorne an)

- LRU und FIFO sind völlig unbrauchbar bis der Cache 50 Pages fassen kann
- RANDOM kann diesen workload relativ gut bearbeiten, jedoch bleibt es noch immer zufällig

Fazit zu den Workloads

→ Selbst potentiell gute Algorithmen wie LRU haben ihre Nachteile

→ Ein statischer Algorithmus kann nur eine Nische ausfüllen, jedoch niemals alle

→ Kann man statische Algorithmen verbessern?

Dynamische Algorithmen

- Dynamische Algorithmen sind oftmals Verbindungen aus statischen Algorithmen
  - ARC (Adaptive Replacement Cache) ist ein Balanceakt zwischen LRU(Least Recently Used) und MRU(Most Recently Used)

- Dynamische Algorithmen versuchen ihre Nische zu erweitern, um möglichst viele Einsatzorte zu haben

- Sie sind wesentlich komplexer zu implementieren als statische Algorithmen

SEQ Replacement Algorithmus

- SEQ (sequentiell) Replacement ist ein 1997 von Gideon Glass und Pei Cao entwickelter Algorithmus
- SEQ Replacement besteht hauptsächlich aus dem LRU Algorithmus
- Sollte eine lange Reihe von misses mehrfach vorkommen, stellt sich der Algorithmus die nächsten Male auf MRU (Most-Recently-Used) um

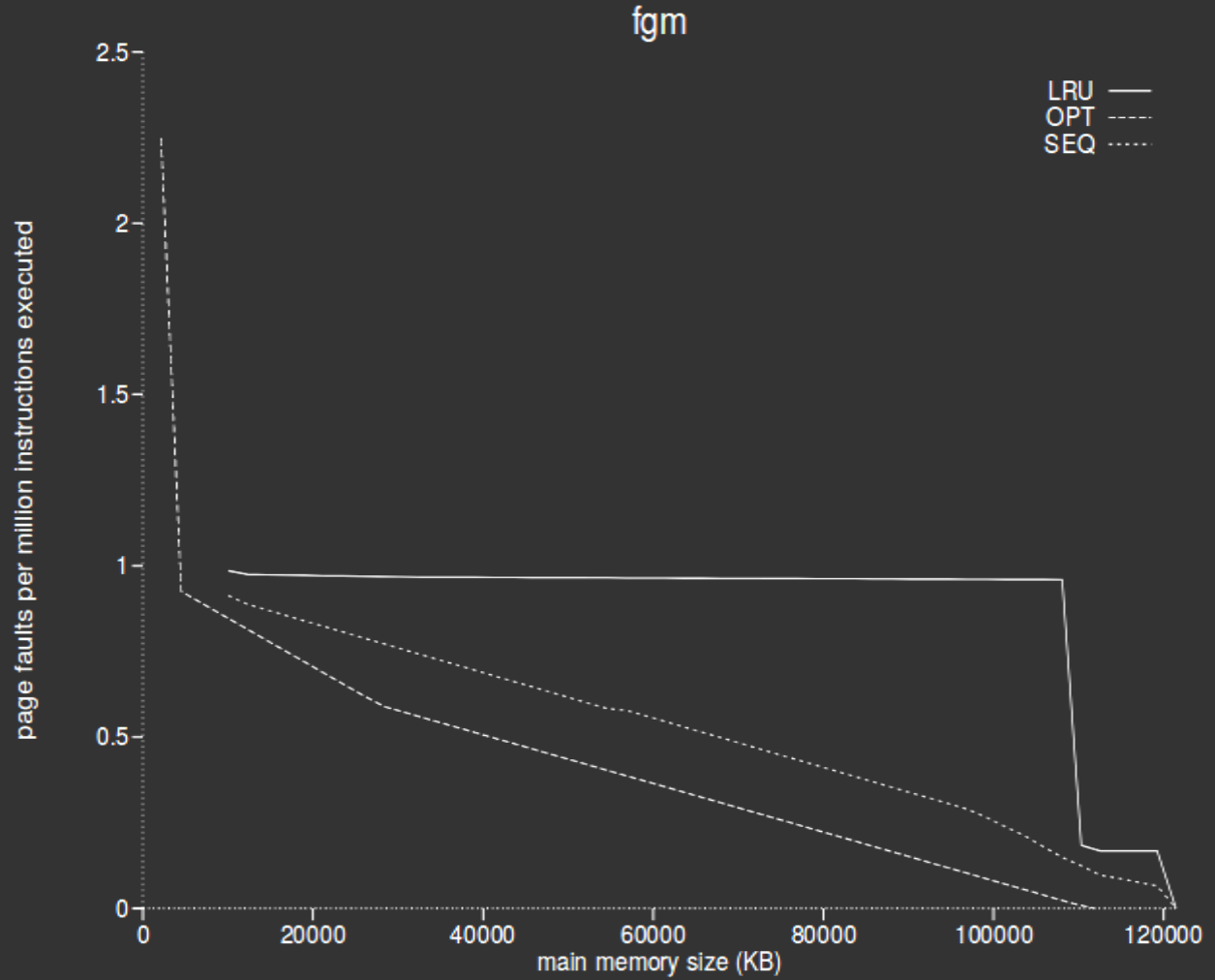


Fig. 10. ftp://ftp.cs.wisc.edu/wwt/sigmetrics97-seq.pdf (Seite 5)

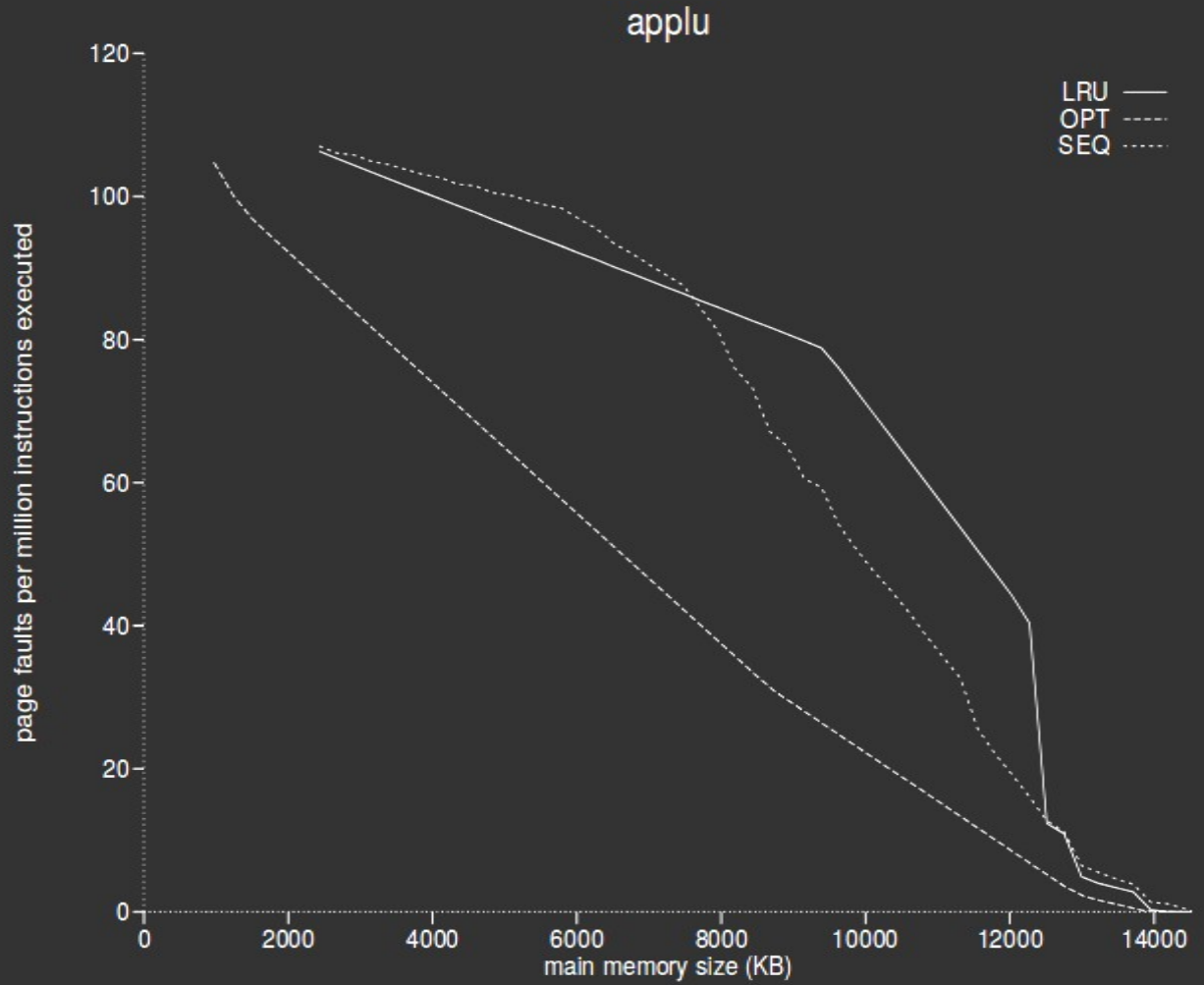


Fig. 11. ftp://ftp.cs.wisc.edu/wwt/sigmetrics97-seq.pdf (Seite 5)

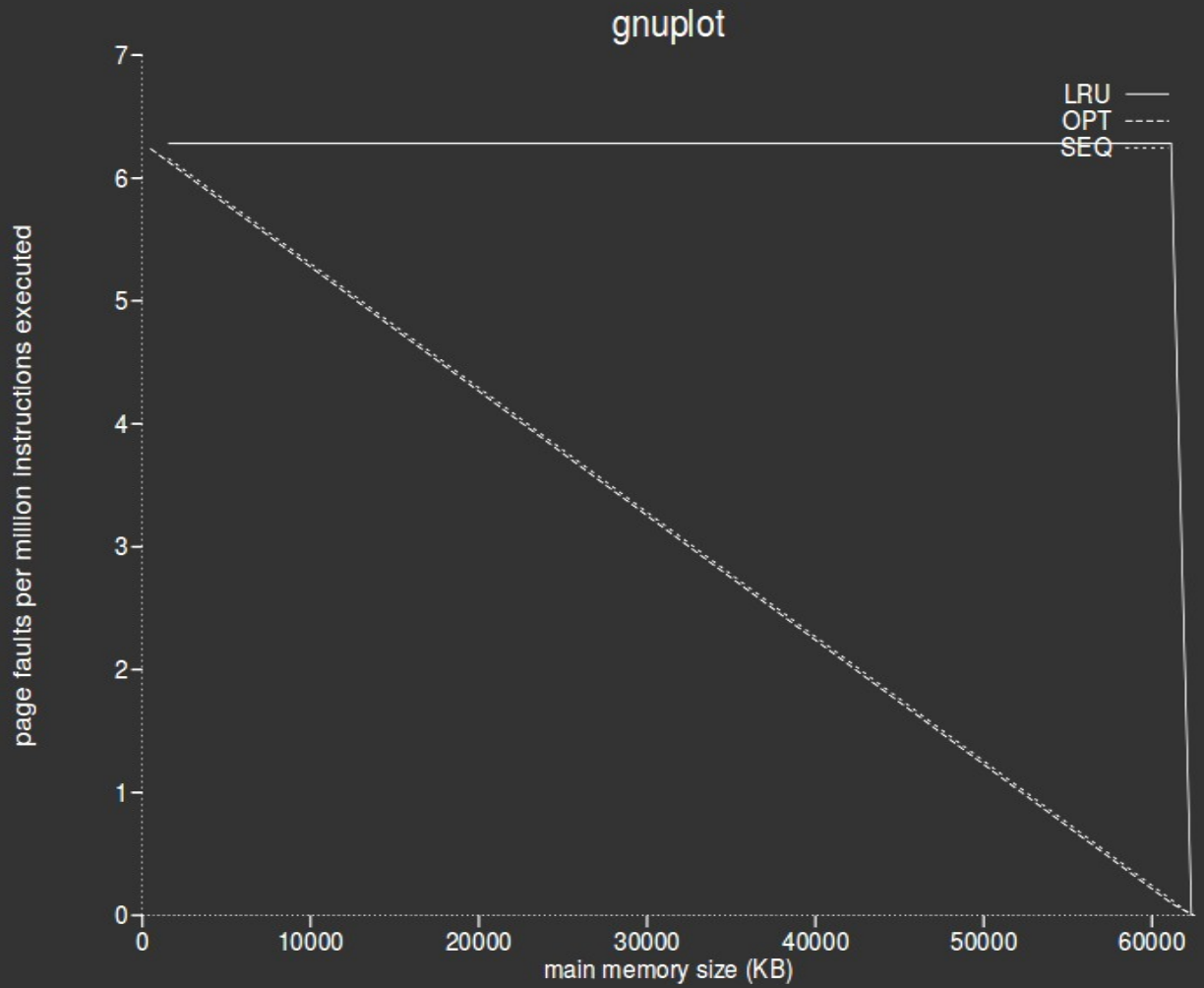


Fig. 12. ftp://ftp.cs.wisc.edu/wwt/sigmetrics97-seq.pdf (Seite 5)

Zusammenfassung

- Cache-Algorithmen sind notwendig, da schnellerer Speicher sehr viel teurer ist als langsamer Speicher und damit nur begrenzt zur Verfügung steht
- Man muss zwischen virtuellem und echten Cache unterscheiden
- Den perfekten Algorithmus gibt es nicht, jeder hat seine Nische in der er sehr gut funktioniert
- Dynamische Algorithmen sind sehr komplex, aber auf lange Sicht unumgänglich wenn es um Performance geht

Fragen?

http://www.mscperu.org/deutsch/picdeutsch/fragen.png

Quellen

- Glass, G. and Cao, P. (1997) Adaptive page replacement based on memory reference behavior
- Qi Zhu , Ying Qiao (2012) A Survey on Computer System Memory Management and Optimization Techniques
- Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau (March 2015) Operating Systems: Three Easy Pieces - Beyond Physical Memory: Policies
- Jakob Lüttgau, Michael Kuhn, Kira Duwe, Yevhen Alforov, Eugen Betke, Julian Kunkel, Thomas Ludwig (2018) Survey of Storage Systems for High-Performance Computing