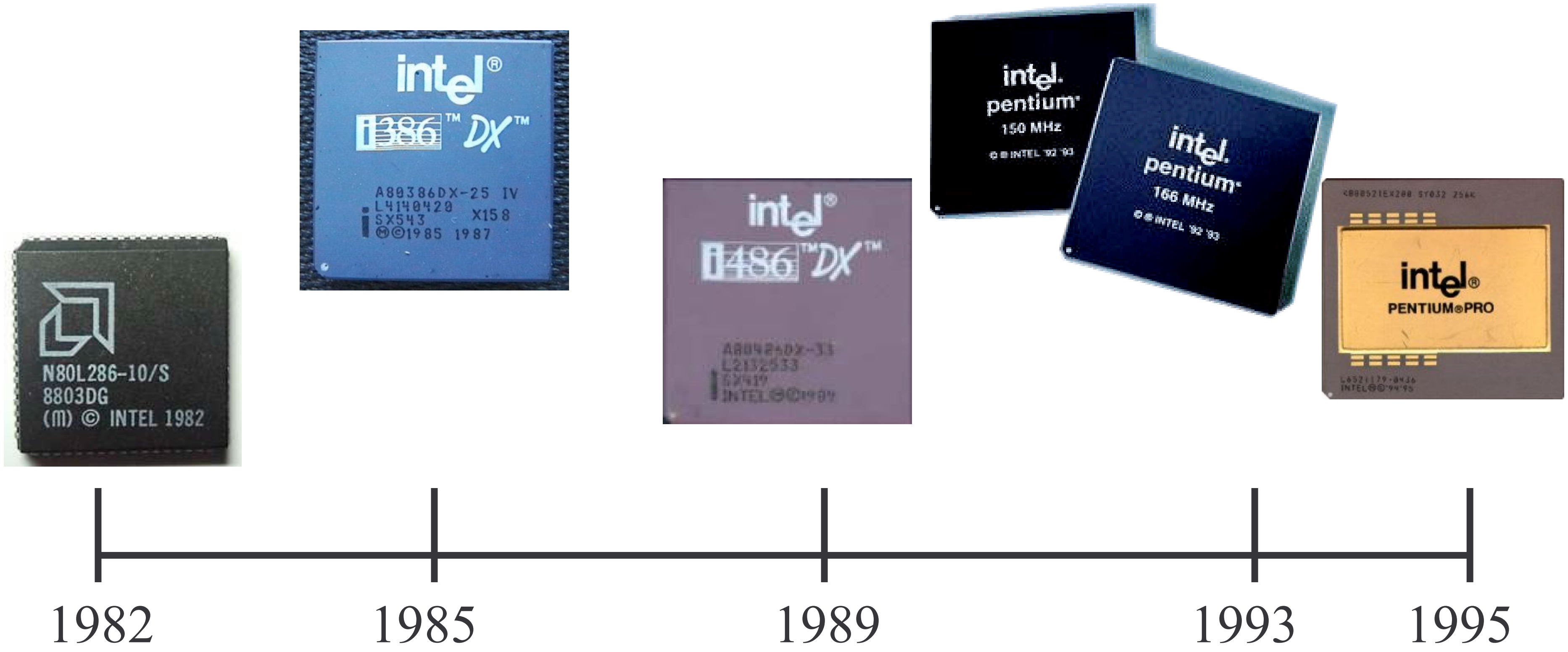
Checking/Correcting Hardware

Jan Möbius

Gliederung

- Pentium Bug
  - → Prüfer/Korrektor für Multiplikation
  - → Prüfer/Korrektor für Division
- Prüfen von Speicher
  - → Offline Prüfer
  - → Online Prüfer

Prozessor Geschichte



Prozessor Geschichte

1982 1985 1989 1993 1995

November 1994

Pentium Bug

Eigenschaften des Fehlers :

- tritt bei der Division auf
- weniger als eine von acht Milliarden Eingaben liefert falsches Ergebnis
- erste Veröffentlichung des Fehlers durch Thomas Nicely Nov. 1994 bei Untersuchungen der Eigenschaften von Primzahlen
- Fehler tritt hauptsächlich in bestimmten Zahlenbereichen auf
- PLA falsch programmiert (zu kurze FOR Schleife)

Pentium Bug Beispiel 1

Beispiel (Thomas Nicely):

$$p = 824633702441$$

$$q = 1 - \left(\frac{1}{p}\right) \cdot p = 0$$

Korrekte Ergebnisse bei Gleitkommarechnung : $q \approx 1{,}11e^{-16}$

Pentium : $q \approx 3{,}72e^{-9}$

Fehlerhäufung im Bereich (nach Niceley):
$824633702418 \leq p \leq 824633702449$

Pentium Bug Beispiel 2

Beispiel in Matlab (Tim Cole) :

$$x = 4195835$$
$$y = 3145727$$
$$z = x - \left(\frac{x}{y}\right) \cdot y = 0$$

Korrekte Ergebnisse bei Gleitkommarechnung : $z < 9{,}3e^{-10}$

Pentium : $z = 255$

Pentium Bug Lösung

Software Patch :

Fehler tritt nur in einigen Zahlenbereichen auf (empirisch bestimmt)

$\Rightarrow$ Verlagern der Division aus den fehlerhaften Zahlenbereichen

$$\frac{N}{D} = \frac{N \cdot \frac{15}{16}}{D \cdot \frac{15}{16}}$$

Letztendlich aber Austausch der defekten Prozessoren durch Intel(Dezember 94).

Zahlendarstellung

Gleitkomma-Darstellung :

$$z = \pm \underbrace{m}_{\text{Mantisse}} \times \underbrace{b}_{\text{Basis}}{}^{\overbrace{\pm d}^{\text{Exponent}}}$$

$$x \cdot y = \left(m_x \cdot m_y\right) \cdot 2^{d_x + d_y}$$

$$x : y = \left(m_x : m_y\right) \cdot 2^{d_x - d_y}$$

Einfacher Prüfer für Multiplikation

$$x \cdot y = \left(m_x \cdot m_y\right) \cdot 2^{d_x + d_y}$$

Annahmen zur Vereinfachung :

- Addition korrekt
- Ignorieren des Dezimalpunktes
- exakte Multiplikationen von Ganzen Zahlen
- Fehler höchstens bei einer von einer Milliarde Multiplikationen

$\Longrightarrow$ Multiplikation zweier ganzer Zahlen A und B

$$AB = C$$

Einfacher Prüfer für Multiplikation

$$AB = C$$

$$((A \bmod r)(B \bmod r) \bmod r) = (AB \bmod r) = C \bmod r$$

r zufällige kleine Zahl

Für falsche Multiplikationen ist die Wahrscheinlichkeit sehr hoch, das die Residuen sich unterscheiden

Verbesserungen :

- Wiederholungen mit verschiedenen r
- Erstellen einer Tabelle für r
- r Primzahl

Korrektor für Multiplikation

A, B n-Bit Zahlen

R_1, R_2 zufällige n-Bit Zahlen

$$AB = \left[2\left(\frac{A+R_1}{2}\right) - R_1\right]\left[2\left(\frac{B+R_2}{2}\right) - R_2\right] =$$

$$= 4\left(\frac{A+R_1}{2}\right)\left(\frac{B+R_2}{2}\right) - 2R_1\left(\frac{B+R_2}{2}\right) - 2R_2\left(\frac{A+R_1}{2}\right) + R_1R_2 = C$$

Aufwand : Im wesentlichen vier Multiplikationen

Korrektor für Multiplikation

$$4\left(\frac{A+R_1}{2}\right)\left(\frac{B+R_2}{2}\right)-2R_1\left(\frac{B+R_2}{2}\right)-2R_2\left(\frac{A+R_1}{2}\right)+R_1R_2=C$$

$$\left.\begin{array}{l} R_1, R_2 \\ \left(\frac{A+R_1}{2}\right), \left(\frac{B+R_2}{2}\right) \end{array}\right\}$$

zufällig über die Hälfte aller möglichen (n+1)-Bit Zahlen verteilt

Paare sind stochastisch unabhängig

Korrektor für Multiplikation

$$\underbrace{4\left(\frac{A+R_1}{2}\right)\left(\frac{B+R_2}{2}\right)}_{\frac{1}{4}}\underbrace{-2R_1\left(\frac{B+R_2}{2}\right)}_{\frac{1}{4}}\underbrace{-2R_2\left(\frac{A+R_1}{2}\right)}_{\frac{1}{4}}+\underbrace{R_1R_2}_{\frac{1}{4}}=C$$

Maximal eine Eingabe aus einer Milliarde Eingaben führt zu Fehlern

$\Longrightarrow$ Pro Paar führen vier aus einer Milliarde Eingaben zu einem Fehler

$\Longrightarrow$ C ist falsch mit einer Wahrscheinlichkeit von 16 zu einer Milliarde

Korrektor für Multiplikation

Bemerkungen :

- Kontrollieren des Ergebnisses mit Prüfer
- Bei Fehler erneut Korrigieren (mit anderen Zufallszahlen)

Prüfer für Division

Division läßt sich auf Multiplikation zurückführen

$$\frac{N}{D} = Q + R$$

$$\Leftrightarrow N = Q \cdot D + R$$

$$\Leftrightarrow N - R = Q \cdot D$$

$\Longrightarrow$ Multiplikation prüfen

Korrektor für Division

Annahmen wie zuvor, allerdings wird $^1\!/\!_x$ bei einer von einer Milliarde möglichen Eingaben falsch berechnet

$$\frac{N}{D} = N \cdot R \cdot \left(\frac{1}{R \cdot D} \right)$$

R zufällige n-Bit Zahl

Die Wahrscheinlichkeit eines Fehlers ist 2 zu einer Milliarde

Erkennung eines Fehlers durch Prüfer und bei Bedarf wiederholte Korrektur

Korrektor für Division

Problem :

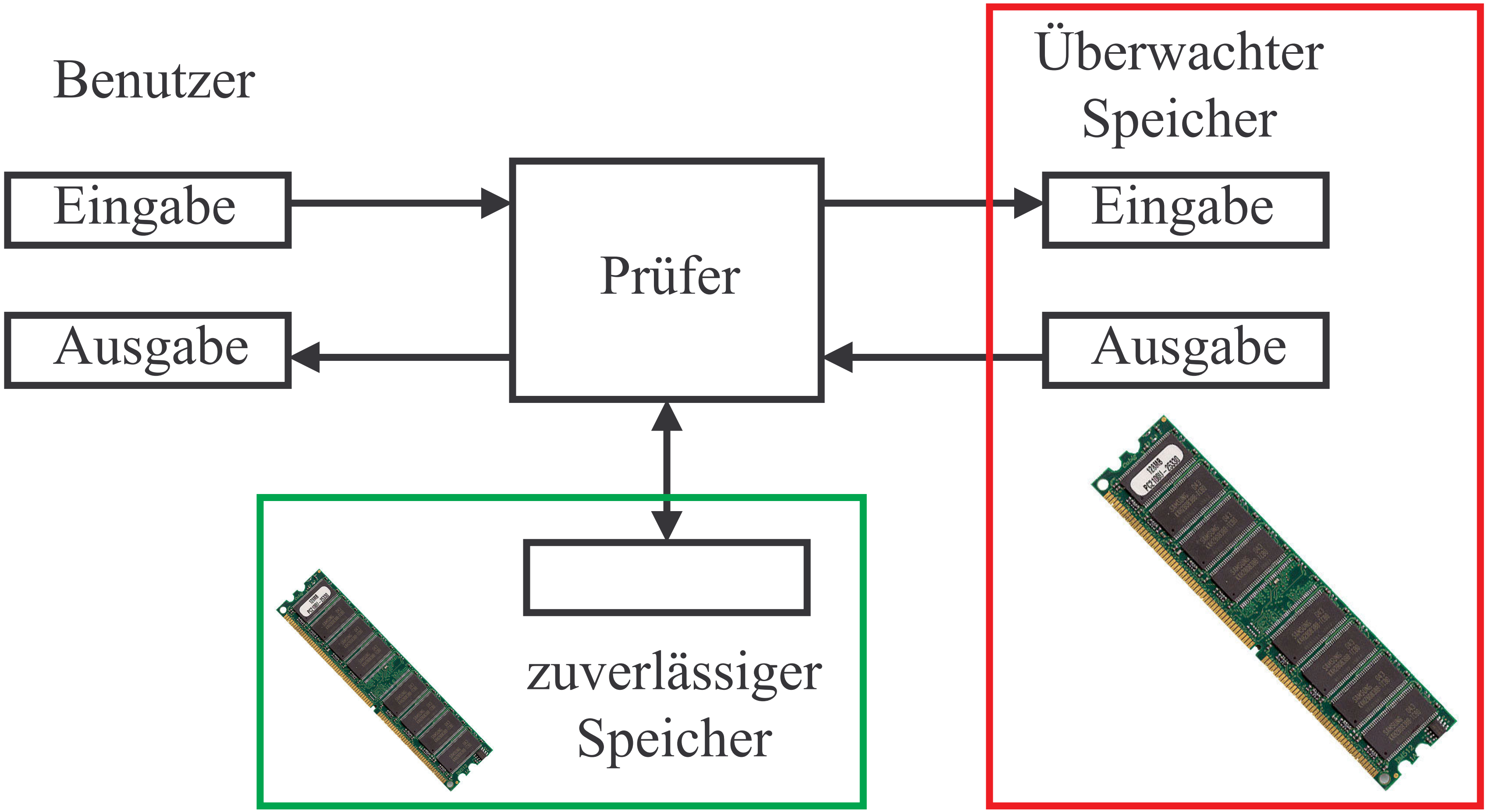
Arithmetischer Fehler durch 3 Multiplikationen und Division. Um eine auf n-bits genaue Ausgabe zu erreichen müßten im Korrektor ungefähr n+2 Bit verwendet werden.

Lösung :

Prüfer und Korrektoren in die Prozessoren integrieren

(Ausnutzung der im Prozessor zusätzlich verfügbaren Stellen)

Prüfen von Speicher



Prüfen von Speicher

Offline Prüfer

- Fehler werden erst nach allen Operationen bemerkt
- gut für kurze Berechnungen

Online Prüfer

- Fehler werden direkt bemerkt
- auch für lange Berechnungen geeignet

invasive Prüfer

- zusätzliche Daten werden im Speicher abgelegt

nichtinvasive Prüfer

- nur die Daten des Benutzers werden gespeichert

ε-biased Hash Funktionen

Problem : Prüfer soll weniger Speicher benötigen als dieser überwacht

Lösung : Hash-Funktionen

x, y	n-bit Zahlen
h	Hash Funktion beschrieben durch $O(\log n + k)$ bits

$$h : x \mapsto h(x)$$

Benötigte Speicherbits : n k

ε-biased Hash Funktionen

x, y n-bit Zahlen

h Hash Funktion beschrieben durch $O(\log n + k)$ bits

$l = O(k)$ Unterscheidungsstrings r_i der Länge n

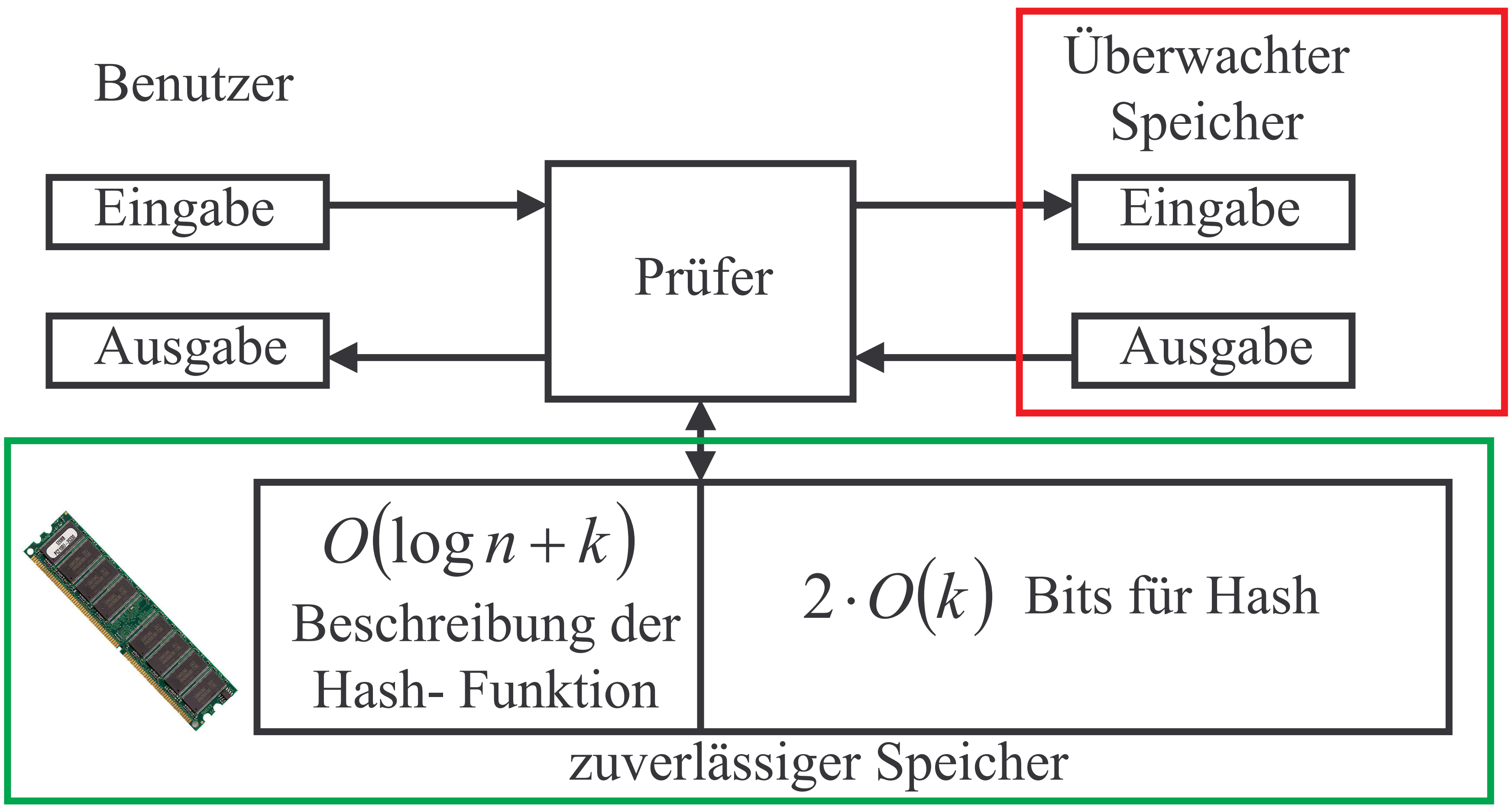
$$h(x) = \left(\langle x, r_1\rangle \bmod 2, \langle x, r_2\rangle \bmod 2, \ldots, \langle x, r_l\rangle \bmod 2\right)$$

Für ungleiche x, y für die Wahrscheinlichkeit p für das Auftreten von $h(x) = h(y)$

$$p \le \left(\frac{1}{2}\right)^k$$

Jedes Bit von r_i kann in $\log n$ Operationen bestimmt werden

ε-biased Hash Funktionen



RAM
(Offline Prüfer)

Schreib-Zugriffe (W)	:	(Wert(v) , Adresse(a) , Zeit(t))
Lese-Zugriffe (R)	:	analog

Kodierung der Zugriffe durch 1 in W bzw. R an Position $v + an + tn^2$

$\Rightarrow$ R und W haben polynomielle Größe

$\Rightarrow$ $h(R)$ und $h(W)$ im sicheren Speicher ablegen

RAM
(Offline Prüfer)

Schreibzugriff (v,a,t) :

- v´ und t´ aus Adresse a auslesen
- überprüfen ob t´<t
- h(R) mit (v´,a,t´) aktualisieren
- v und t in Adresse a schreiben
- h(W) mit (v,a,t) aktualisieren

RAM
(Offline Prüfer)

Lesezugriff auf Adresse a zur Zeit t :

- v´ und t´ aus Adresse a auslesen
- überprüfen ob t´<t
- h(R) mit (v´,a,t´) aktualisieren
- v´ und t in Adresse a schreiben
- h(W) mit (v´,a,t) aktualisieren

RAM (Offline Prüfer)

Aktualisierung von h(W) bzw. h(R) :

Invertieren von Bit i von h(W) falls Bit $v + an + tn^2$ in r_i gesetzt ist.

$\Rightarrow$ Aktualisierung benötigt $O(k)$ Operationen

Überprüfung der Lese-/Schreib-Sequenz :

- Speicher initialisieren und h(W) aktualisieren
- Sequenz durchführen
- Alle Speicherzellen lesen und h(R) aktualisieren
- h(R) und h(W) vergleichen

RAM
(Offline Prüfer)

Beispiel :

$v + an + tn^2 \qquad n = 2$

$v + a \cdot 2 + t \cdot 4$

a \ t	0	1
0	0_0	
1	0_0	

	0	1	2	3	4	5	6	7
0	0 1	0 0	0 1	0 0	0 0	0 0	0 0	0 0
1								
2								

RAM
(Offline Prüfer)

Beispiel : $v + a \cdot 2 + t \cdot 4$

alten Wert lesen und h(R) aktualisieren

a \ t	0	1
0	0_0	0_0
1	0_0	0_0

	0	1	2	3	4	5	6	7
0	0 1	0 0	0 1	0 0	0 0	0 0	0 0	0 0
1	0 1	0 0	1 1	0 0	0 0	0 0	0 0	0 0
2								

RAM
(Offline Prüfer)

Beispiel : $v + a \cdot 2 + t \cdot 4$

alten Wert lesen und h(R) aktualisieren

neuen Wert schreiben und h(W) aktualisieren

a \ t	0	1
0	0_0	0_0
1	0_0	1_1

	0	1	2	3	4	5	6	7
0	0 1	0 0	0 1	0 0	0 0	0 0	0 0	0 0
1	0 1	0 0	1 1	0 0	0 0	0 0	0 0	0 1
2								

RAM
(Offline Prüfer)

Beispiel : $v + a \cdot 2 + t \cdot 4$

alten Wert lesen und h(R) aktualisieren

neuen Wert schreiben und h(W) aktualisieren

alle Zellen lesen und h(R) aktualisieren

a \ t	0	1
0	0_0	0_0
1	0_0	1_1

	0	1	2	3	4	5	6	7
0	0 1	0 0	0 1	0 0	0 0	0 0	0 0	0 0
1	0 1	0 0	1 1	0 0	0 0	0 0	0 0	0 1
2	1 1	0 0	1 1	0 0	0 0	0 0	0 0	1 1

RAM
(Offline Prüfer)

Beispiel : $v + a \cdot 2 + t \cdot 4$

alten Wert lesen und h(R) aktualisieren

neuen Wert schreiben und h(W) aktualisieren

alle Zellen lesen und h(R) aktualisieren

a \ t	0	1
0	0_0	0_0
1	0_0	0_1

	0	1	2	3	4	5	6	7
0	0 1	0 0	0 1	0 0	0 0	0 0	0 0	0 0
1	0 1	0 0	1 1	0 0	0 0	0 0	0 0	0 1
2	1 1	0 0	1 1	0 0	0 0	0 0	1 0	0 1

Stack (Offline Prüfer)

Modifikation des RAM Prüfers

- Adresse ist die Ebene im Stack
- Bei Pop wird nur h(R) aktualisiert (Adresse ist hinterher leer)
- Beim Push wird nur h(W) aktualisiert (Adresse vorher leer)

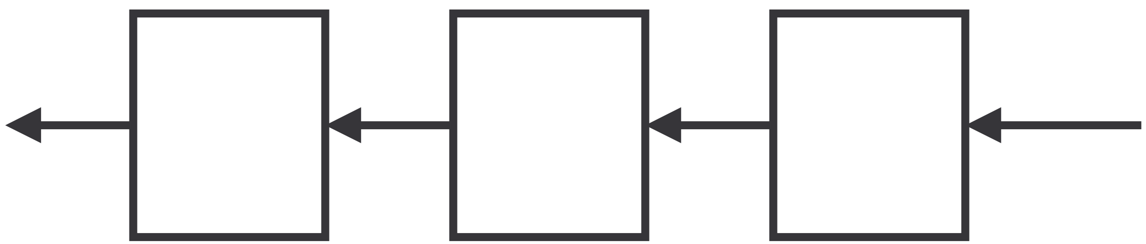
Verbesserung durch Ausnutzung des eingeschränkten Zugriffs
$\Rightarrow$ weniger invasiv, weniger Speicherverbrauch

Queue
(Offline Prüfer)

Modifikation des RAM Prüfers

- Adresse ist die Anzahl der vorausgegangenen Lese- bzw. Schreiboperationen (Im Prüfer mitgezählt)
- da die Adresse eindeutig ist, kann der Timestamp entfallen
- auch hier nur einzelnes Update von R bzw. W

Zum Überprüfen der Sequenz muss die Queue geleert werden und h(R) entsprechend aktualisiert werden.



RAM
(Online Prüfer)

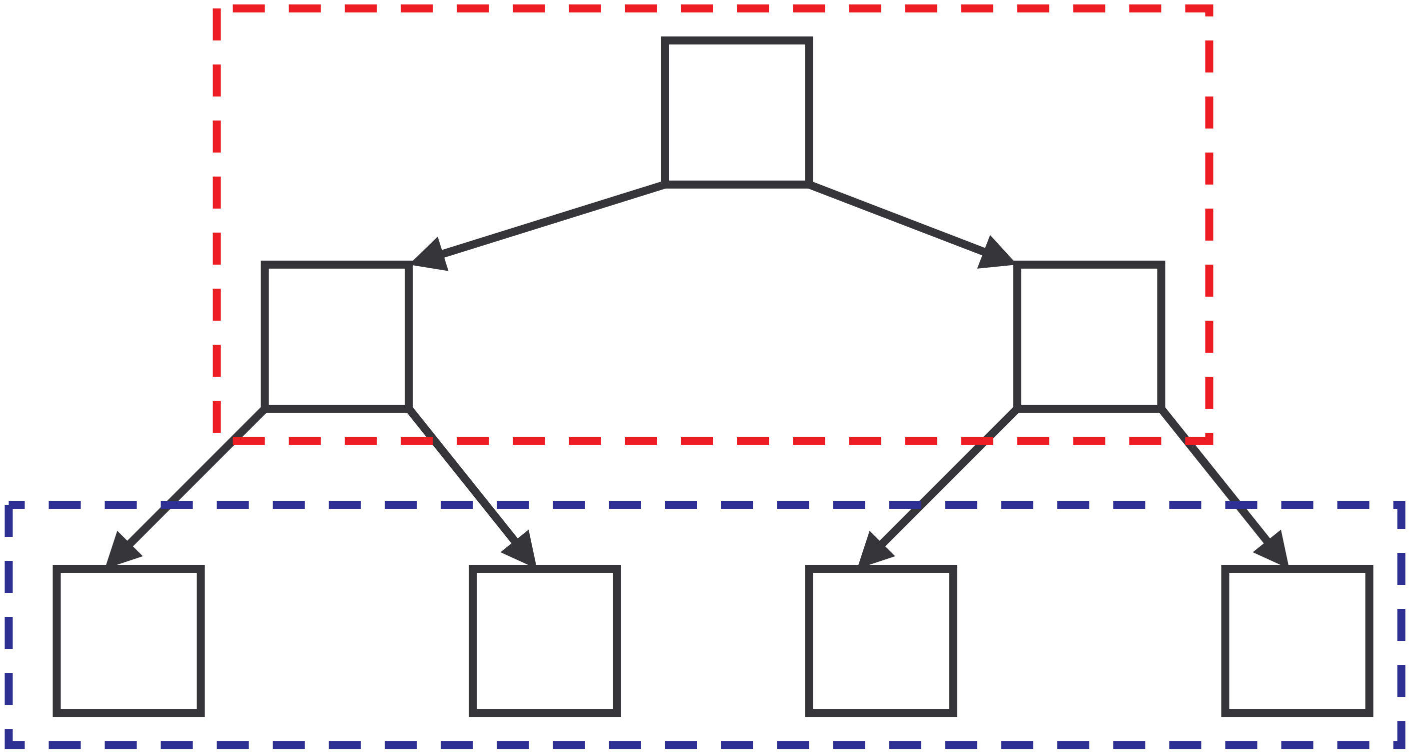
mögliche Anwendungen :

- rechenintensive Aufgaben
- sicherheitskritische Bereiche

Hash Verfahren :

- Universelle Ein-Weg Hash Funktionen
- Pseudozufällige Funktionen

RAM
(Online Prüfer)



RAM
(Online Prüfer)

Hash Verfahren :

- Universelle Ein-Weg Hash Funktionen
  (Speicher lesbar)
- Pseudozufällige Funktionen
  (Speicher geheim)

Zeitlicher Aufwand : $O(t \log n)$

n Speichergröße

t Aufwand zur Auswertung der Hash Funktion

Zusammenfassung

- Prüfer/Korrektor für Multiplikation und Division (z.B. für Anwendung in Prozessoren)
- ε-biased Hash Funktionen
- Offline Prüfer für RAM, Stack und Queue
- Online Prüfer