Effiziente Programme

Robert Bachmann
Benjamin Biesinger
Gernot Fritz
Patrick Leitgeb
Arthur Oberhauser
Sebastian Wurzer

WS 2009/2010

Ziele / Ergebnisse

Ziel

Optimierung des „`uid2name`“ Programms

Ergebnis

Original: 850,4 M.Zyklen

Optimiert: 001,4 M.Zyklen

Vorweg …

- In Kürze… 7 Optimierungen
- Folien und Quelltexte auf LVA-Seite (Gr. 112)

1. Optimierung -- fseek

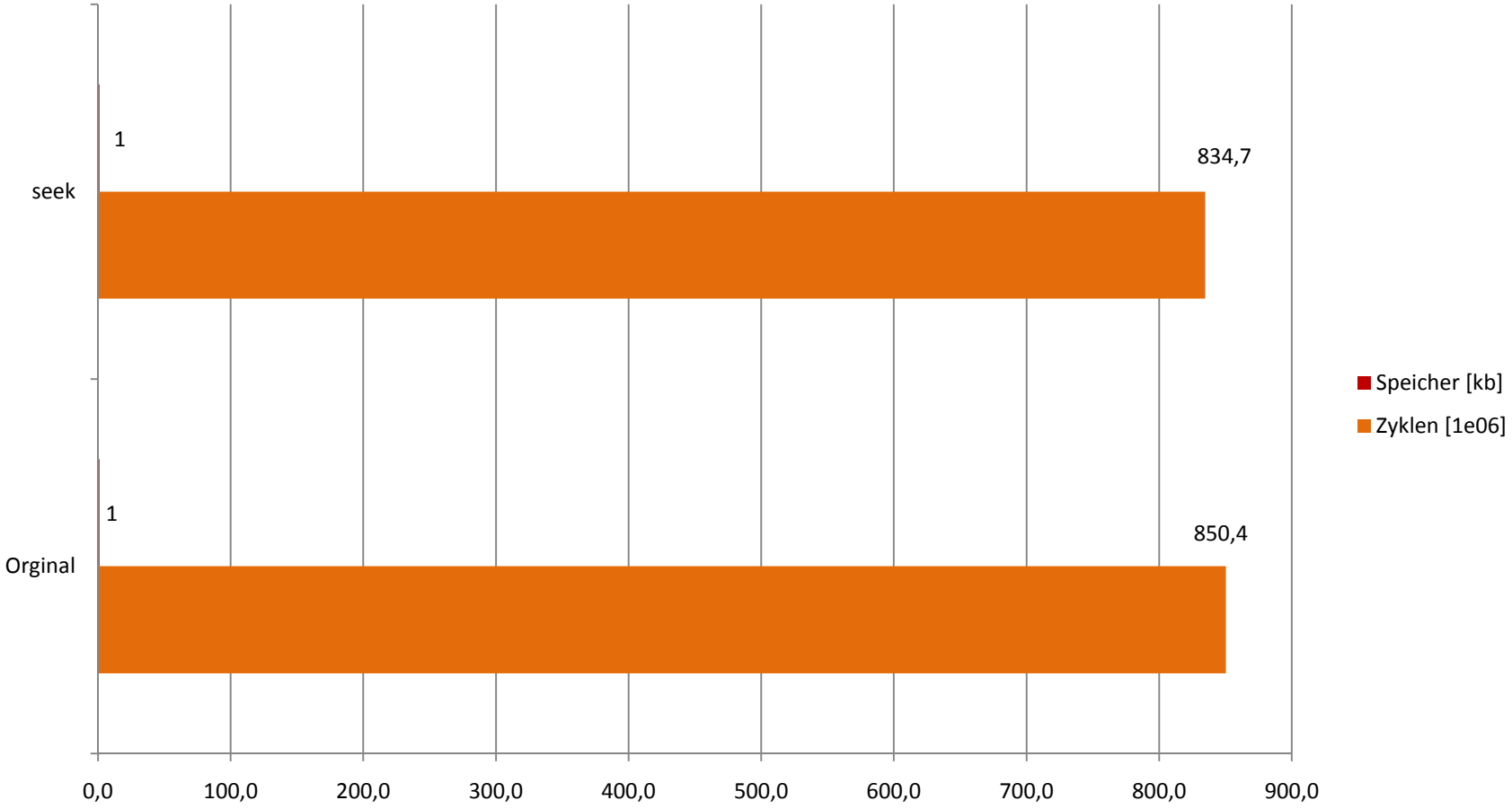
Original

1 `fopen` von „passwd“ pro `uid2name` Aufruf

Änderung

`fopen` in `main`, `fseek` in `uid2name`

1. Optimierung -- fseek



2. Optimierung -- strchr

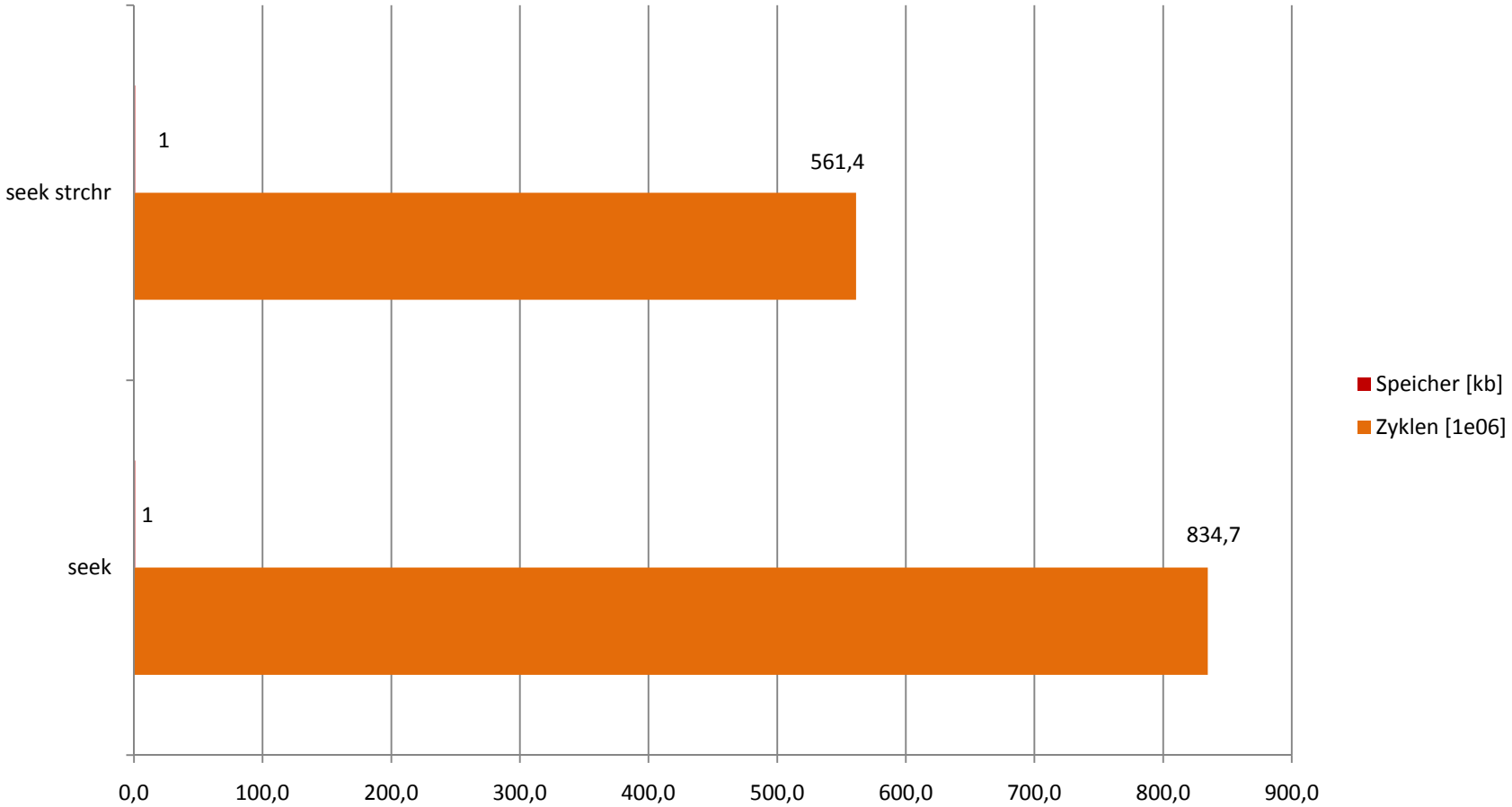
Original

`strtok` zum zerlegen der Zeilen

Änderung

`strchr` zum zerlegen der Zeilen

2. Optimierung -- strchr



3. Optimierung – strcmp für uid

Original

n-mal: `i == strtoul(uid_str)`

Änderung

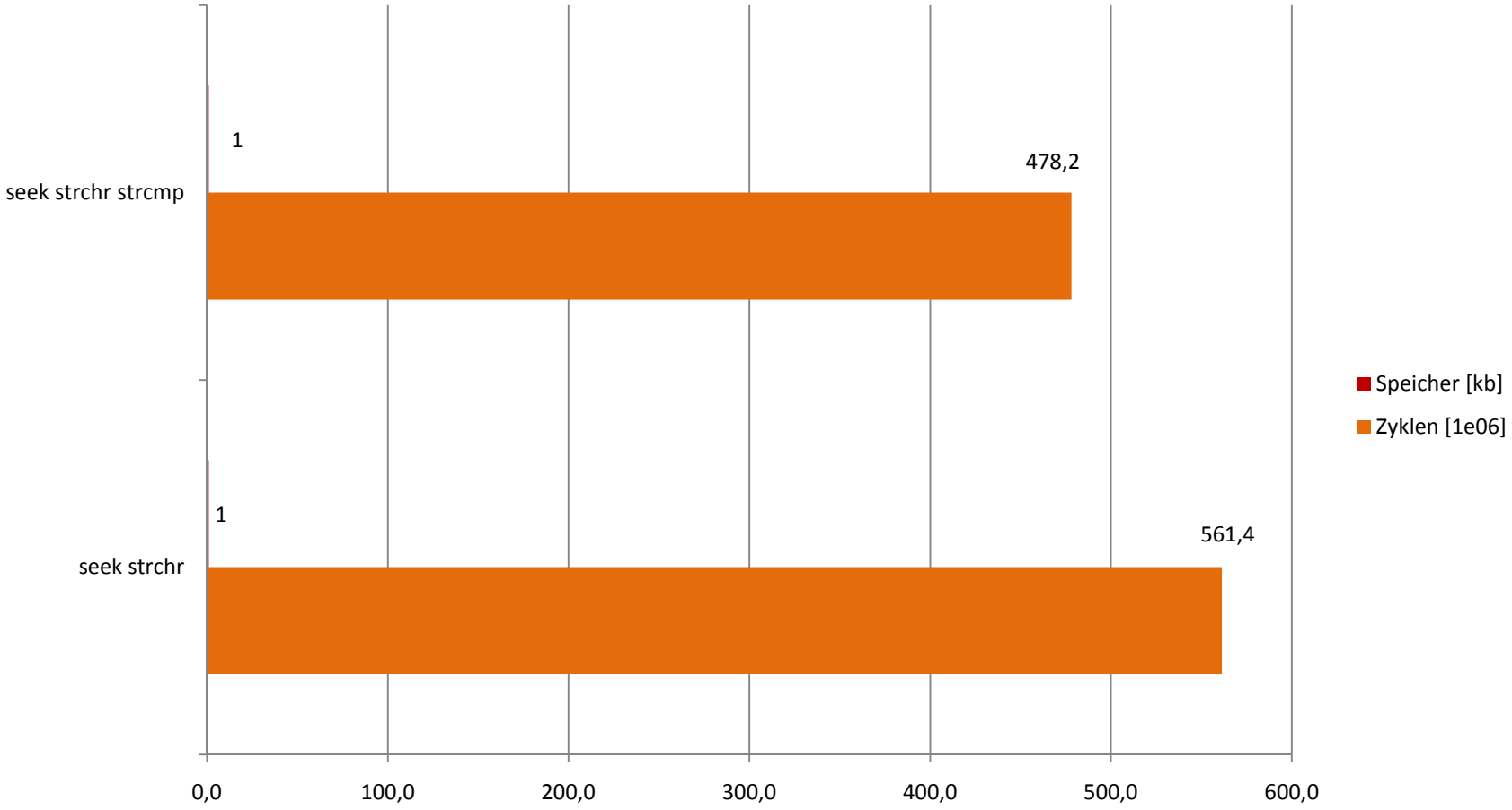
1-mal: `sprintf(s, …)`

n-mal: `strcmp(s, uid_str) == 0`

Nachteil

Führende Nullen, Leer-, Vorzeichen.

3. Optimierung – strcmp für uid



4. Optimierung
Lineare Suche in dyn. Array

Original

n-mal: File durchsuchen

Änderung

1-mal: File parsen

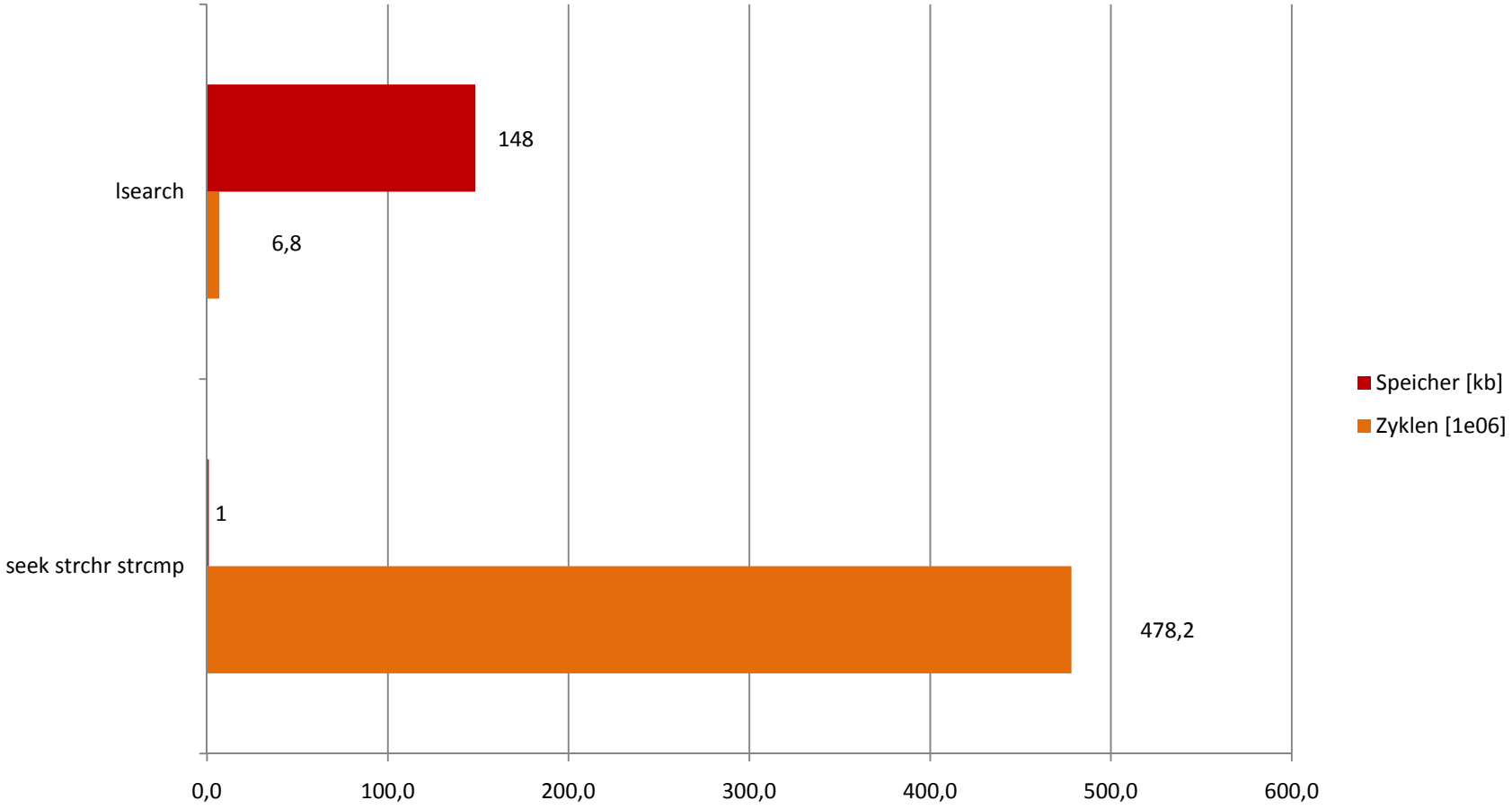
`struct{int;char[32]}[p]`

n-mal: Array linear durchsuchen

Nachteil

Mehr Speicherplatz notwendig

4. Optimierung
Lineare Suche in dyn. Array



5. Optimierung
Binäre Suche in dyn. Array

Original

n-mal: File durchsuchen

Änderung

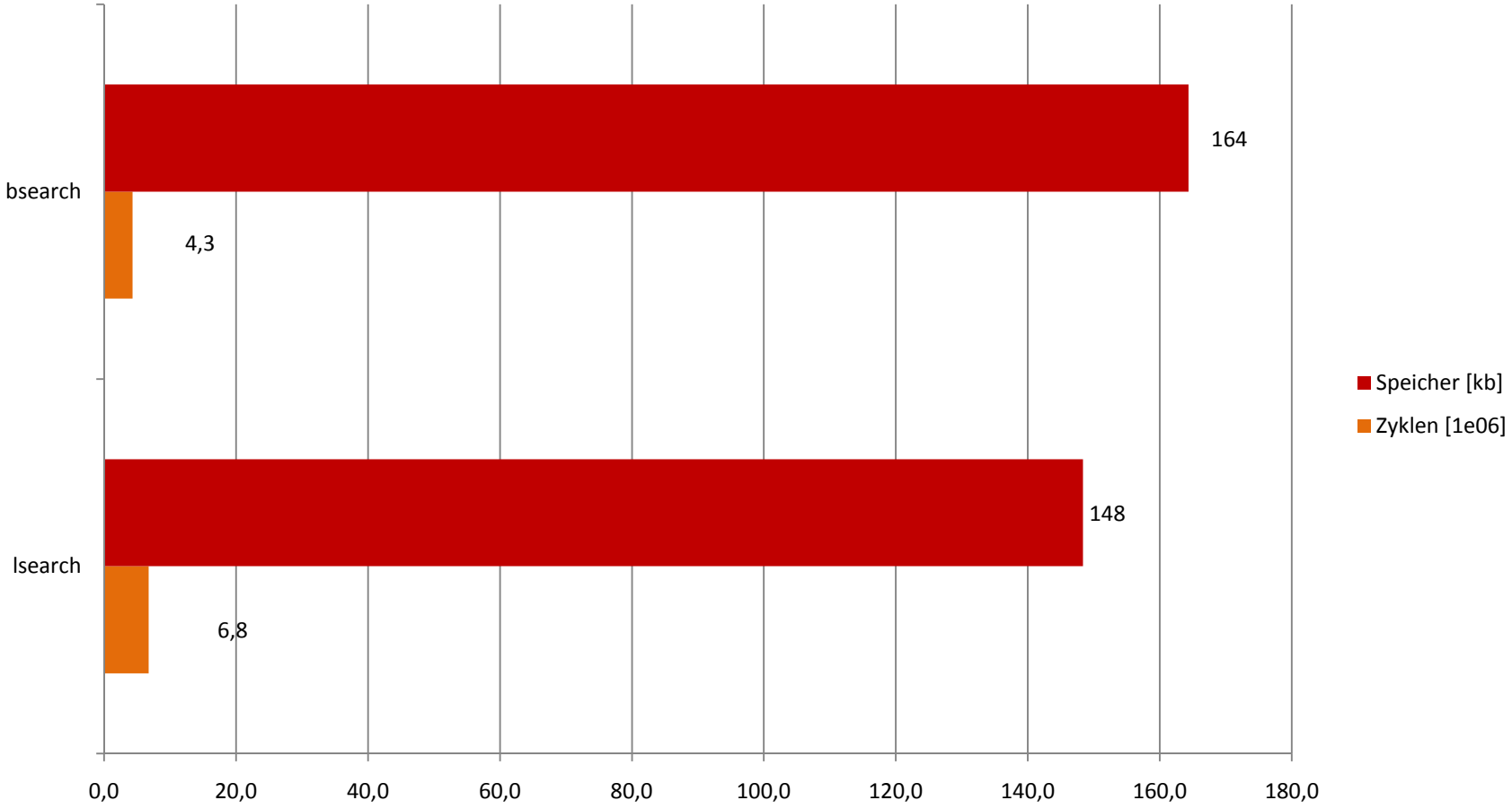
1-mal: File parsen und sortieren (`qsort`)
`struct{int;char[32]}[p]`

n-mal: Array durchsuchen (`bsearch`)

Nachteil

- Sortieren notwendig
- Problem mit doppelten UIDs -- später

5. Optimierung
Binäre Suche in dyn. Array



6. Optimierung -- Binäre Suche in dyn. Array mit Cache File

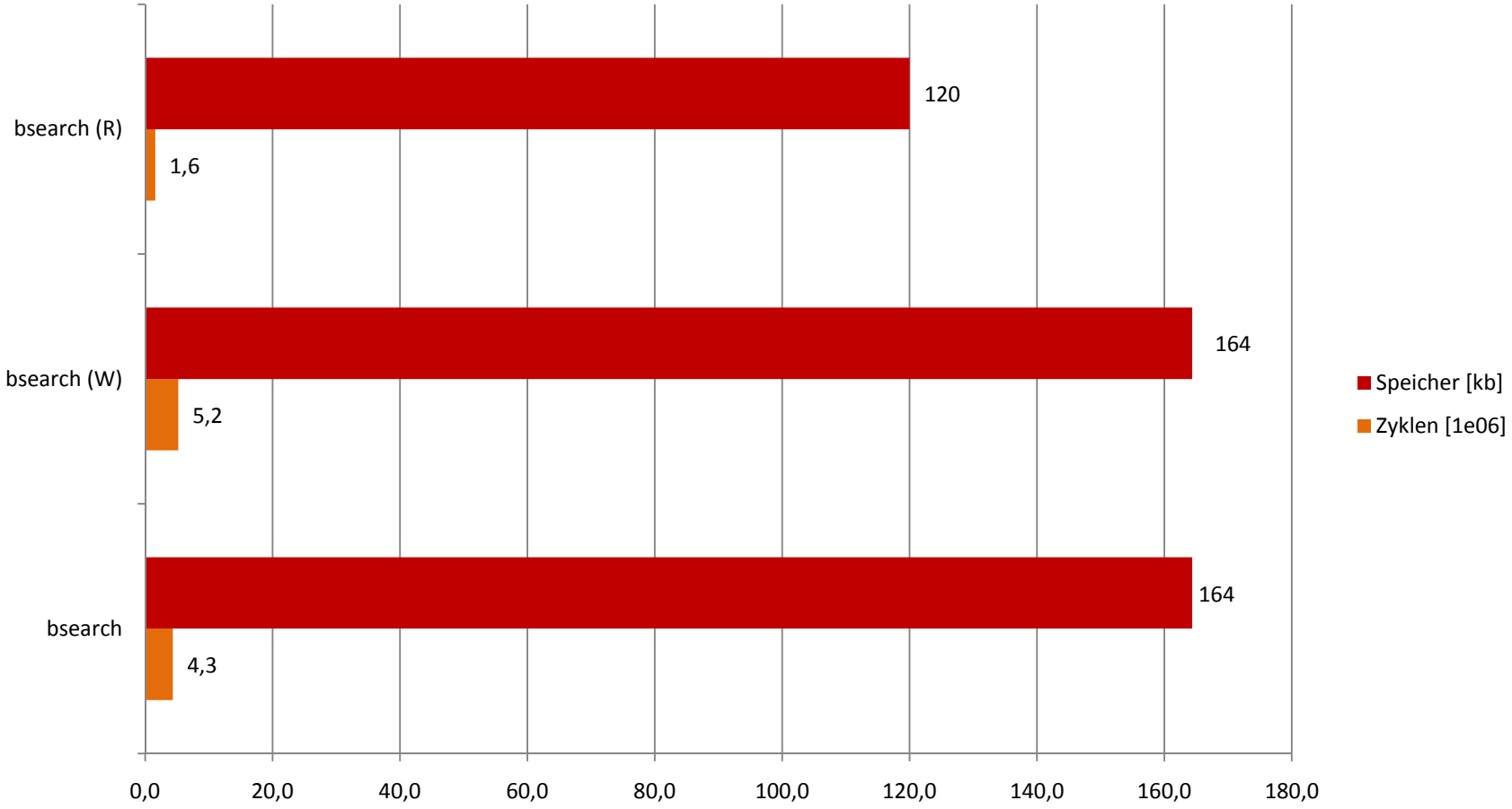
Original

1-mal: passwd File parsen und sortieren

Änderung

1-mal: File parsen und sortieren, Ergebnis in Cache File ablegen. (Cache File erneuern falls Zeitstempel von passwd neuer als der des Cache Files ist.)

6. Optimierung -- Binäre Suche in dyn. Array mit Cache File



7. Optimierung -- Binäre Suche in dyn. Array mit „memmapped“ Cache File

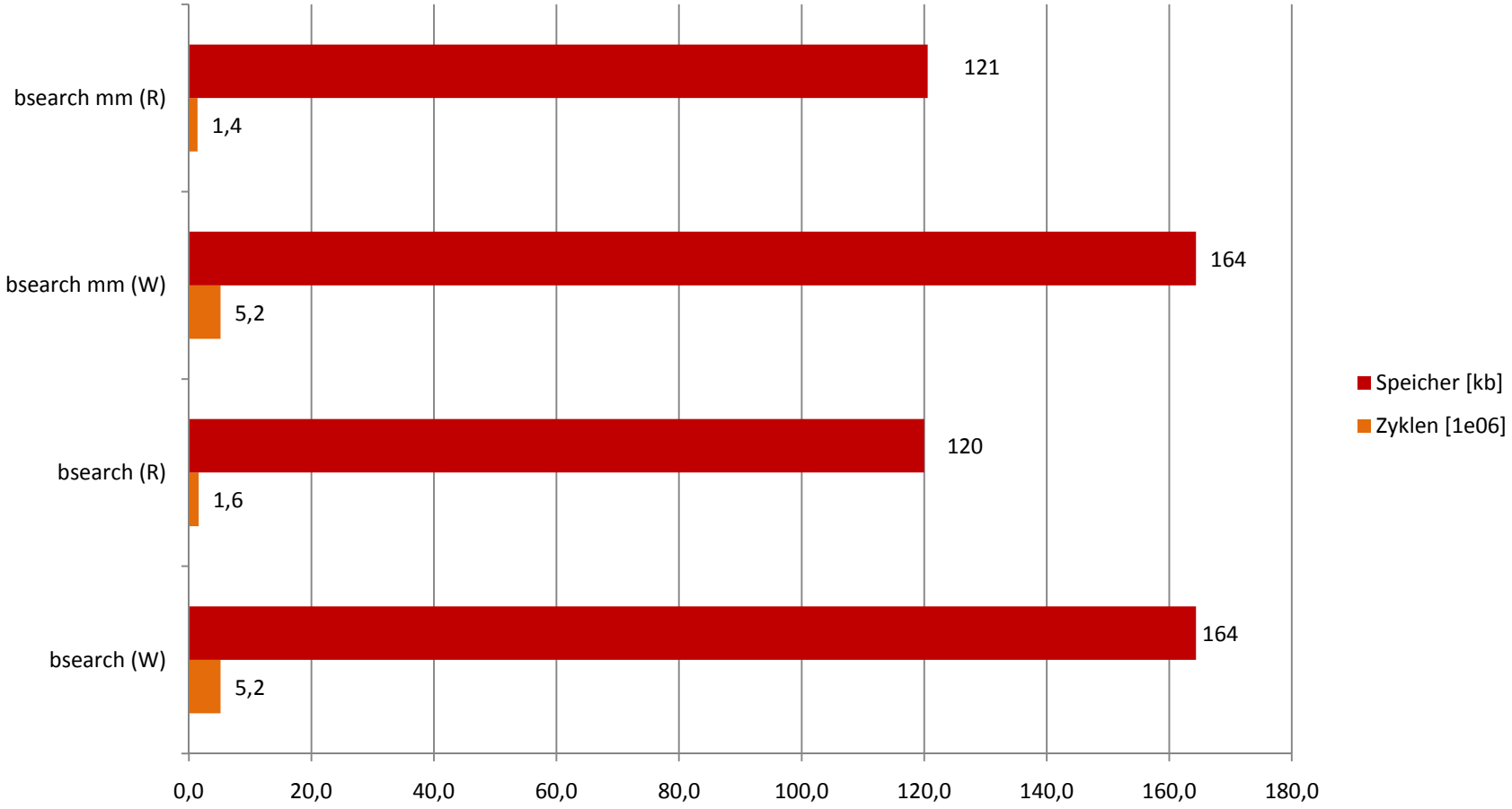
Original

1-mal: Cache File mit `fread` komplett einlesen

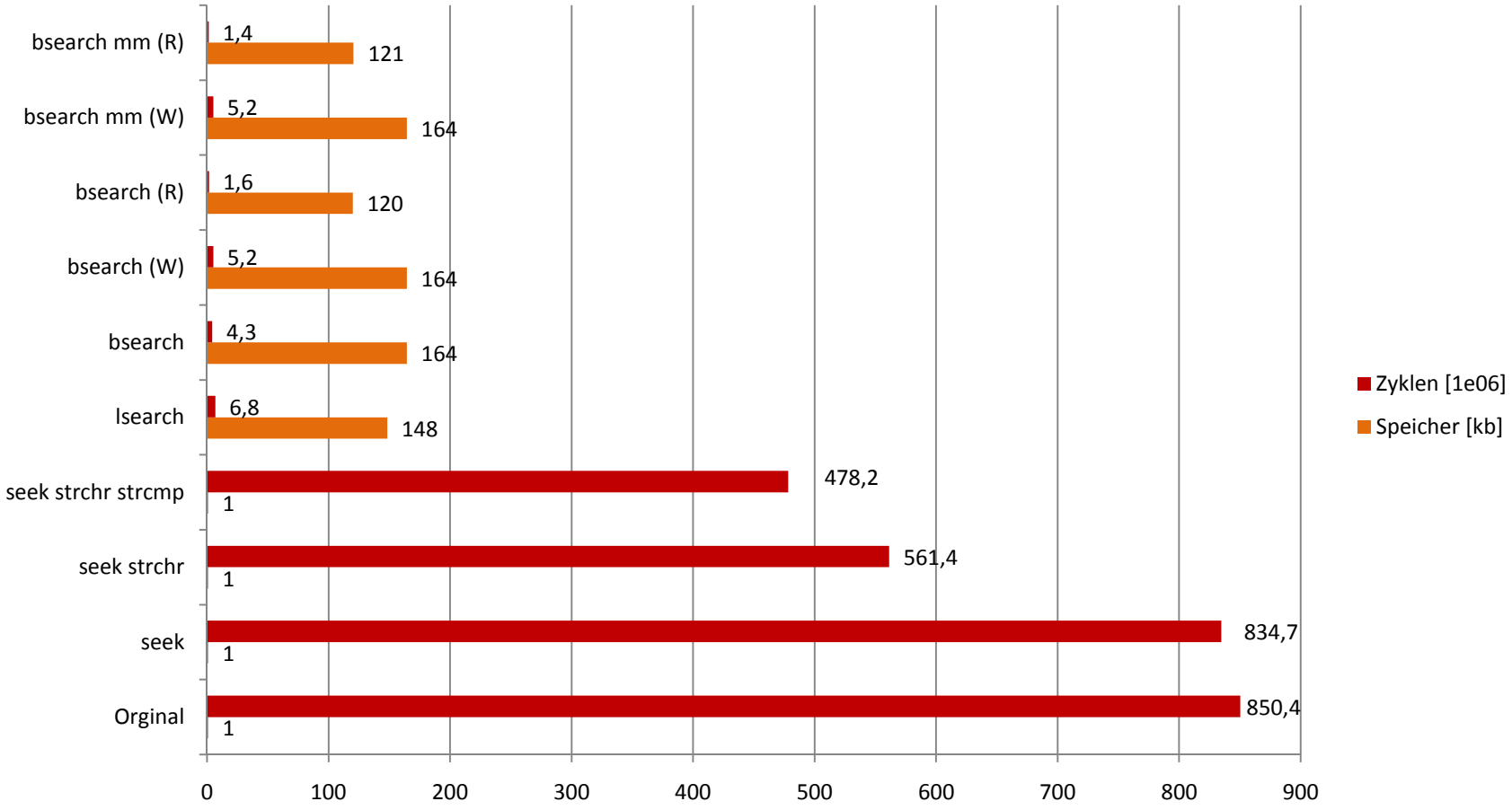
Änderung

1-mal: Cache File mit `mmap` in den Prozess-speicher mappen.

7. Optimierung -- Binäre Suche in dyn. Array mit „memmapped“ Cache File



Zusammenfassung



Doppelte UIDs

- Einige UIDs sind mehrfach vorhanden:
  0, 13060, 13065, 13098, 13099, 13106, 13117, 13258, 10207
- Originale Programm nimmt immer die erste Instanz.
- Problematisch wenn man Array sortiert.
- Aus Zeitgründen einfachen O(n) Ansatz gewählt.

Doppelte UIDs erkennen (1/2)

1) Zeilen durchnummerieren

passwd	**struct {int, char[32], uint}**
root:x:0: …	{0, "root", 0}
franz:x:12: …	{12, "franz", 1}
fritz:x:13: …	{13, "fritz", 2}
john:x:12:…	{12, "john", 3}

Doppelte UIDs erkennen (2/2)

2) Nach UID und Zeile sortieren

{0,	"root",	0}
{12,	"franz",	1}
{12,	"john",	3}
{13,	"fritz",	2}

3) Instanzen nummerieren „0“ heißt gültig.

{0,	"root",	0}
{12,	"franz",	0}
{12,	"john",	1}
{13,	"fritz",	0}

Danke für Ihre
Aufmerksamkeit