PPG19

Bessere Performance durch Cache-Optimierungen

Jannek Squar

2019-04-25

Scientific Computing
Department of Informatics
Universität Hamburg
https://wr.informatik.uni-hamburg.de

Table of Contents

Table of Contents

Speicher-Hierarchie

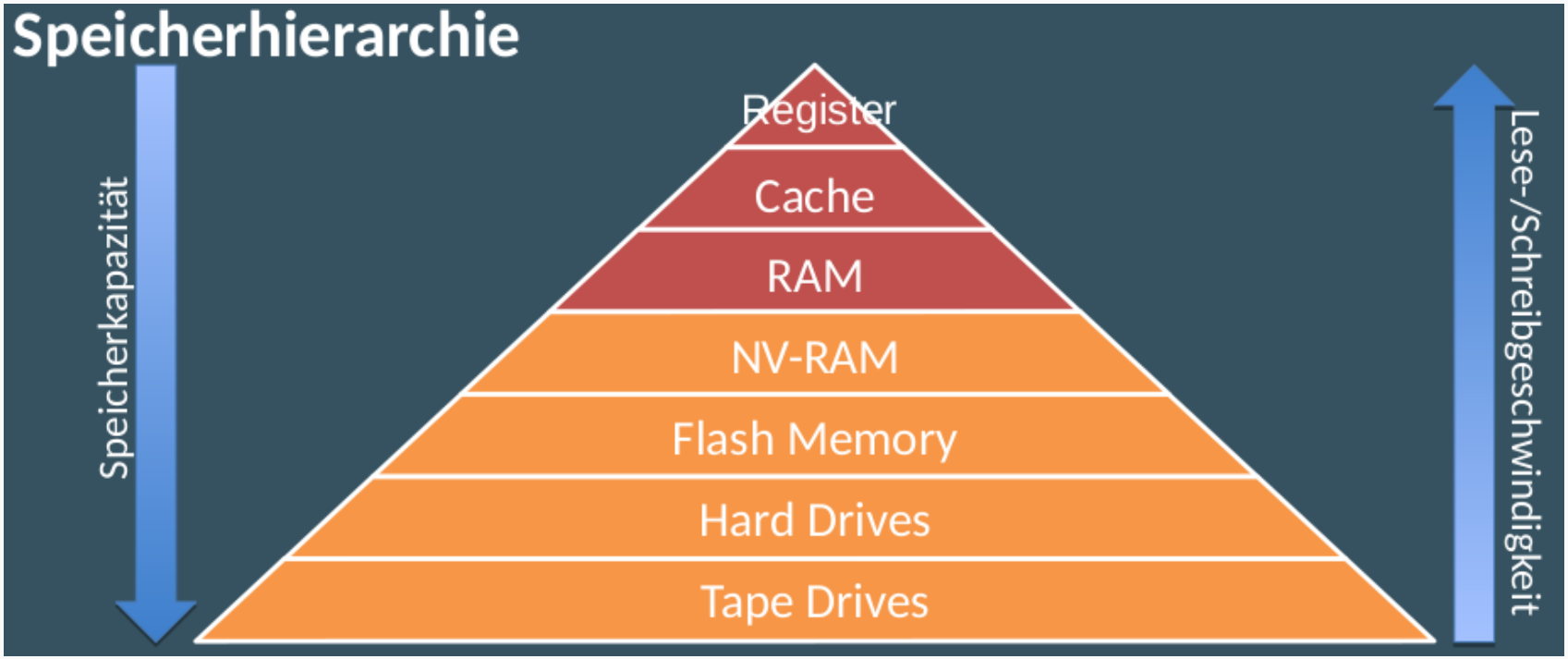

Figure 1: Speicher-Hierarchie Schema [Gü19]

Laden eines Datums

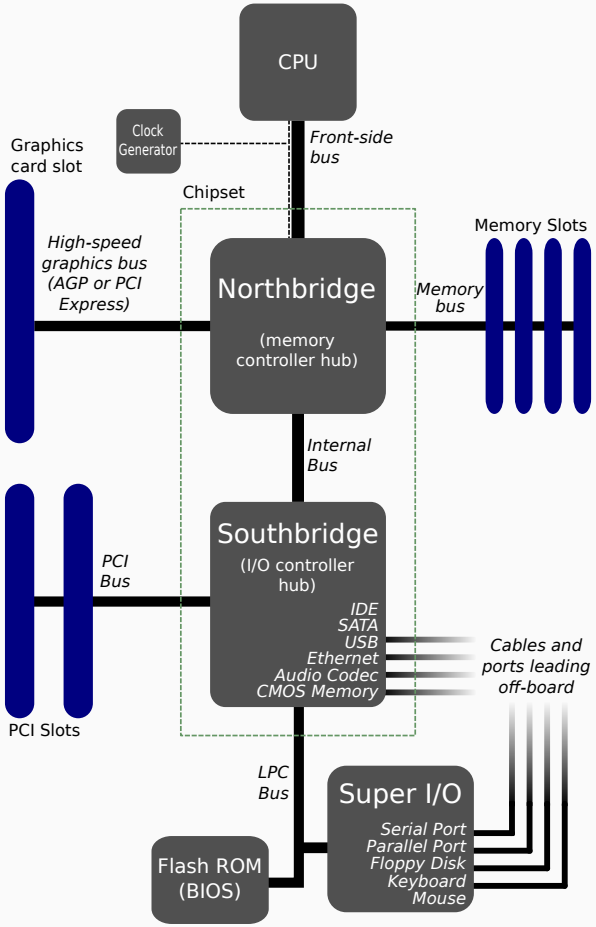

Figure 2: Motherboard Schema [Gri07]

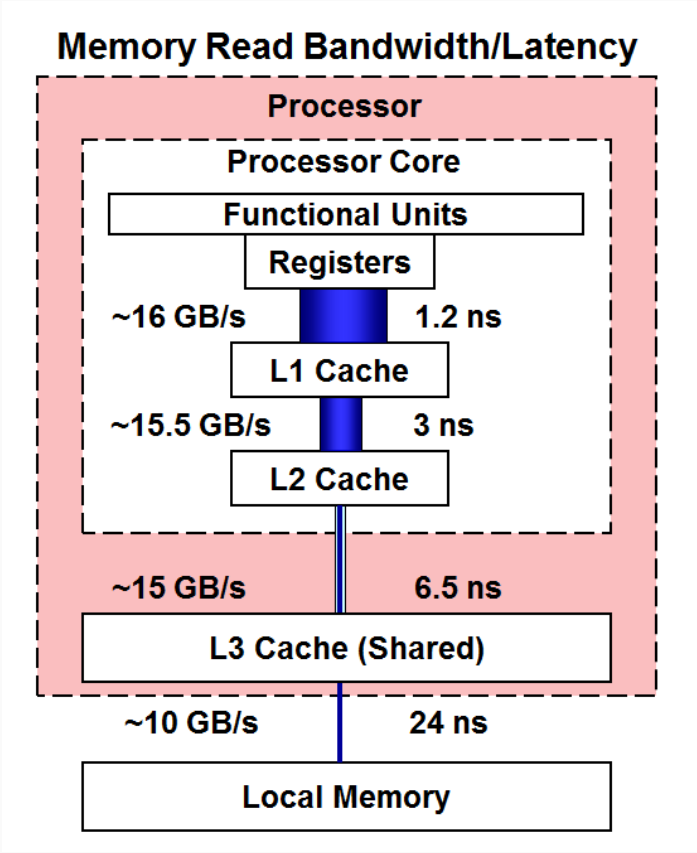

Figure 3: Sandy Bridge (Latenz, Bandbreite) [htt, SCJ13]

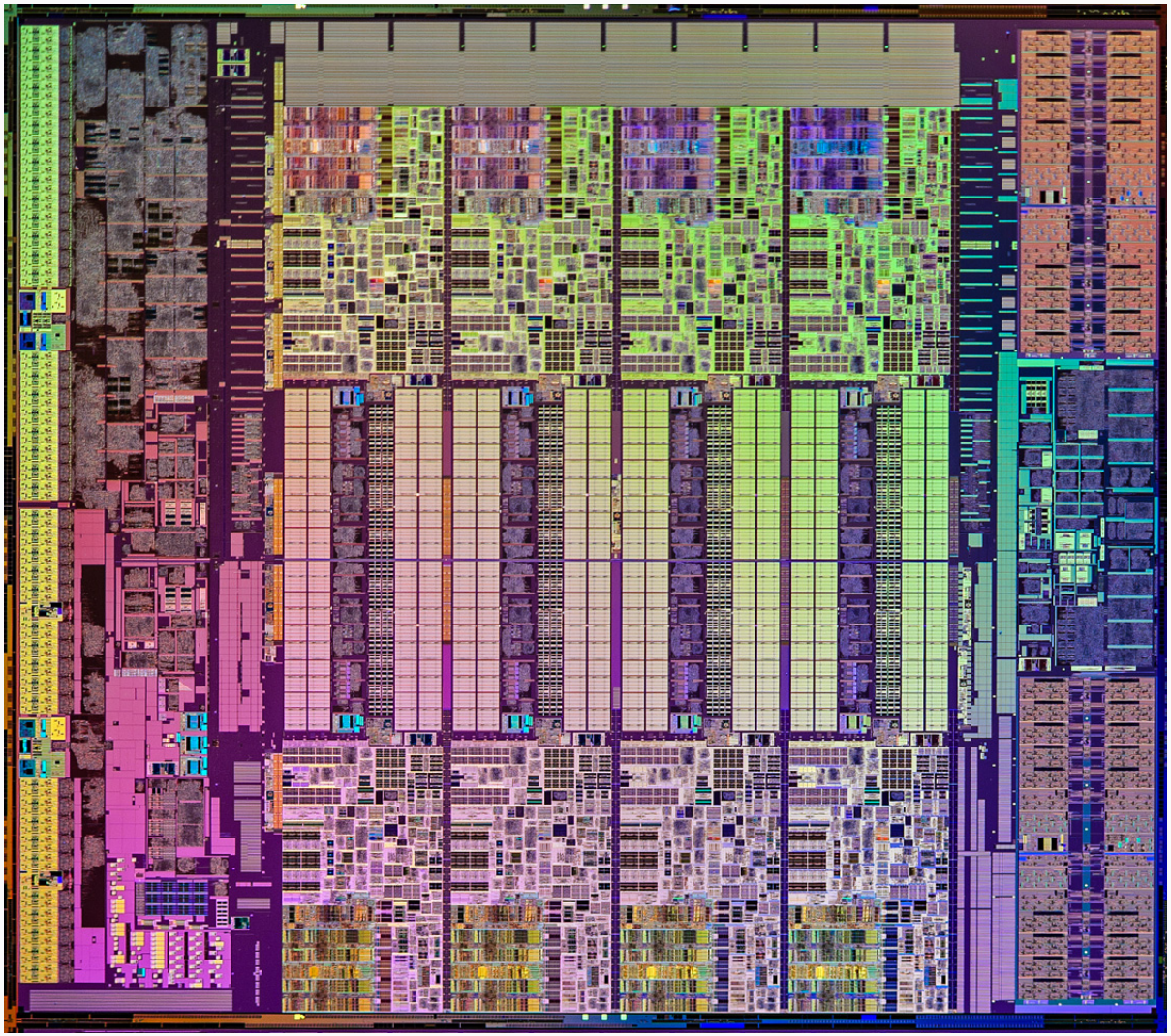

Figure 4: Haswell Die [Hru14]

Table of Contents

- Laden von Daten zeitintensiv
  - Cache liegt auf CPU Die
    $\rightarrow$ niedrige Latenz, hohe BW
  - Cache-Speicher teuer
    $\rightarrow$ Speicher-Hierarchie
- **Lokalitätseigenschaften**
  - Zeitliche Lokalität:
    $\rightarrow$ Verdrängung "alter" Daten
  - Räumliche Lokalität:
    $\rightarrow$ Read-ahead auf Cache-Line
    ($\sim$ 64 Byte)

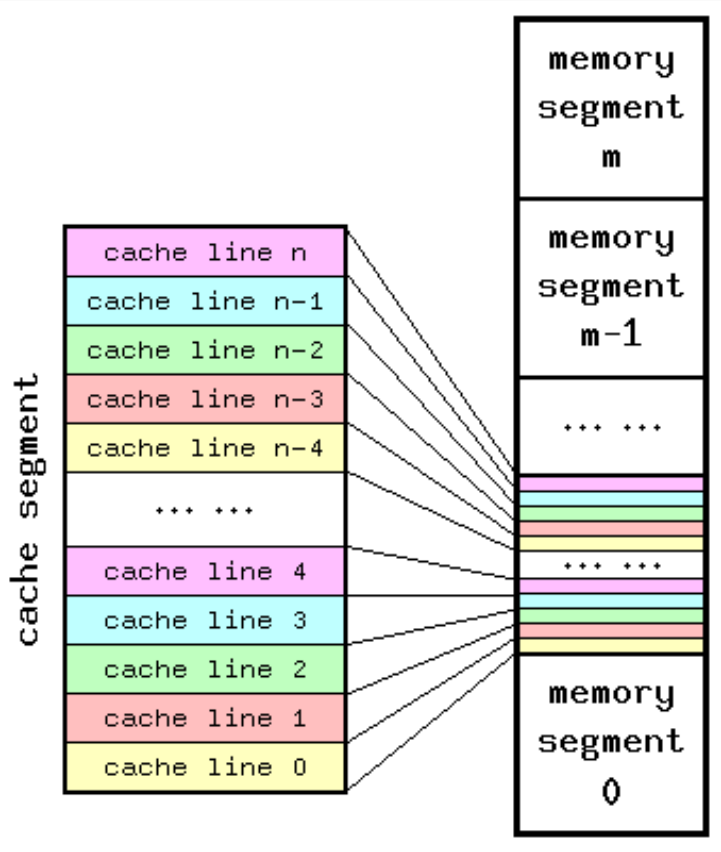

Figure 5: Cache-Lines [cac]

Bewertung der Cache-Performance mittels Hit-Ratio:

$$Hit\ Ratio = \frac{Hit}{Hit + Miss}$$

Durchschnittliche Latenz:

$$t_{avg} = p \times t_c + (1 - p) \times t_m$$

p	Wahrscheinlichkeit für Cache-Hit ($\approx$ Hit-Ratio)
$1 - p$	Wahrscheinlichkeit für Cache-Miss, $1 - p > 0$
t_c	Latenz Cache
t_m	Latenz Memory
	mit $t_c \ll t_m$

Auswirkung auf Laufzeit

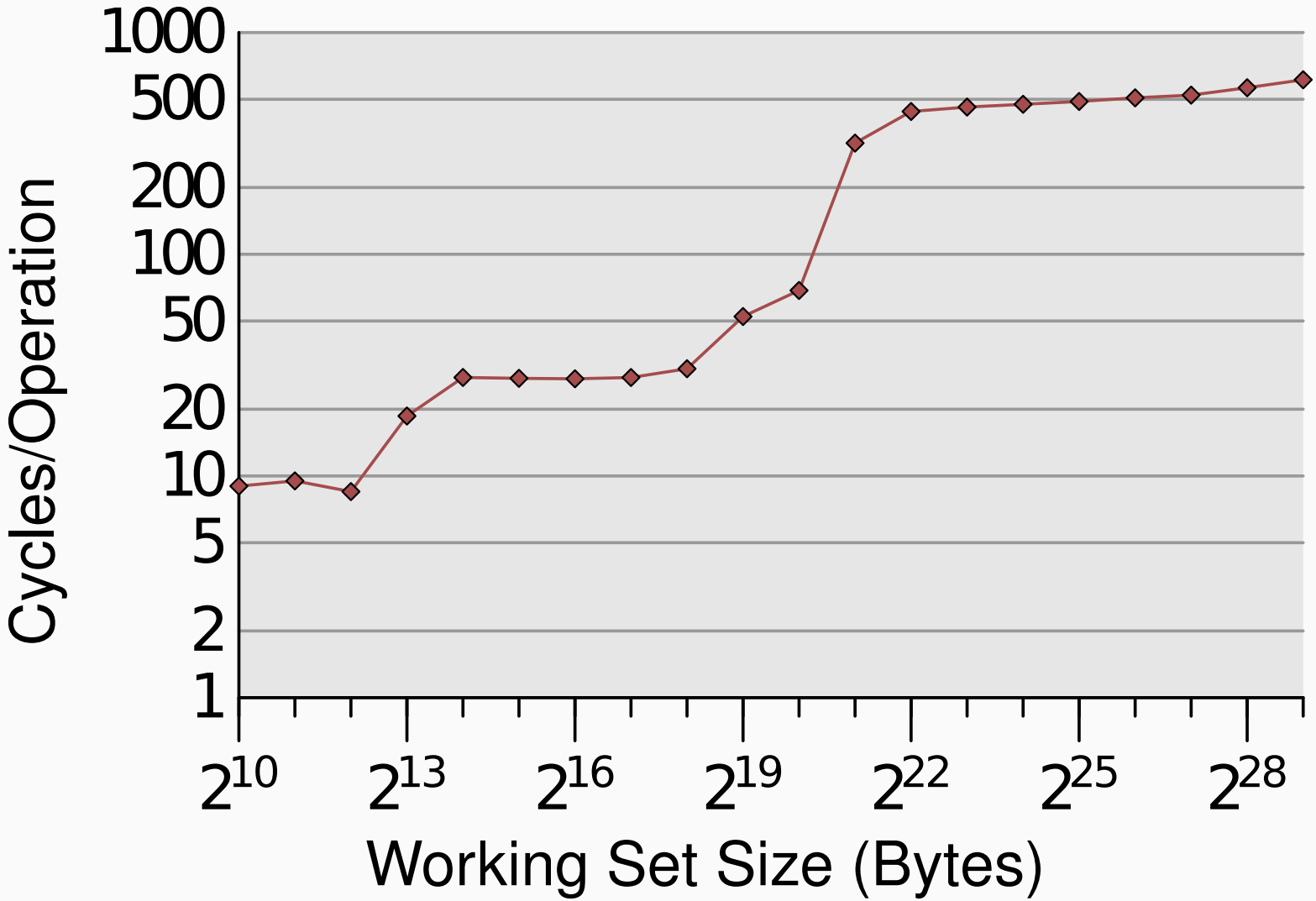

Figure 6: Latenz: L1, L2, main memory [Dre07]

Optimierungsansatz

- **Ausnutzen der Datenlokalität**
  $\rightarrow$ weniger Datenbewegung
- Kein direkter Einfluss auf Cache-Funktionalität
  $\rightarrow$ indirekte Optimierungen
- Evaluierung einer Optimierung:
  - Cache-Kapazität: `lscpu`
  - Cache-Miss-Zähler: `perf stats -e cache-misses ./APPLICATION`
  - Zeitmessung: `time ./APPLICATION`

Table of Contents

Speicher-Layout

Ablage eines n-Dim Arrays im Speicher:

- Zeilenweise bzw. row-major
  - Letzter Index ist der *schnellste* Index
  - Z.B.: C/C++
- Spaltenweise bzw. column-major
  - Erster Index ist der *schnellste* Index
  - Z.B.: FORTRAN
- Weder noch
  - Z.B.: Java, Python

2D Speicher-Layout (I)

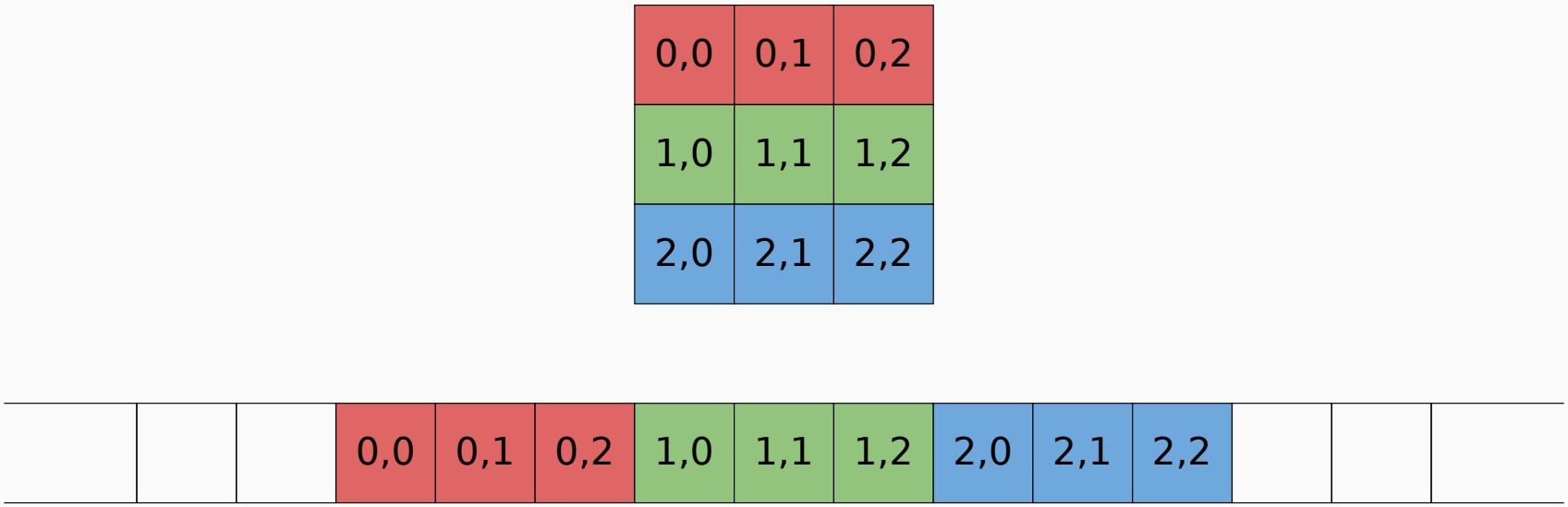

Figure 7: 2D row-major [Ben15]

2D Speicher-Layout (II)

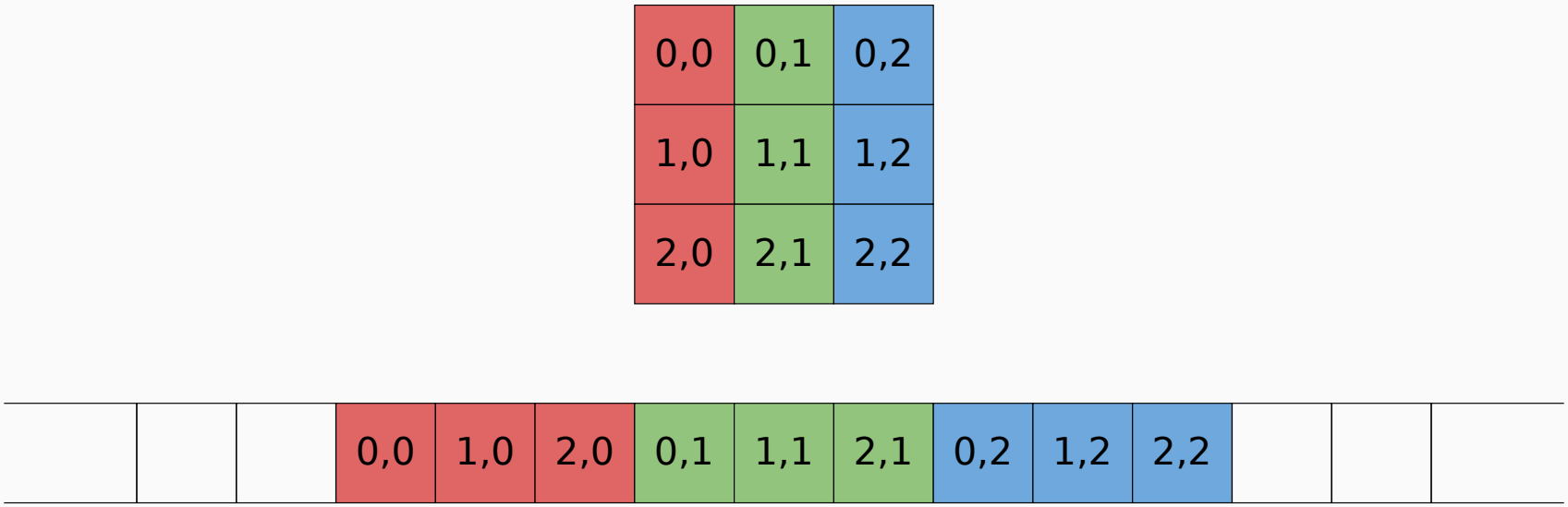

Figure 8: 2D column-major [Ben15]

3D Speicher-Layout

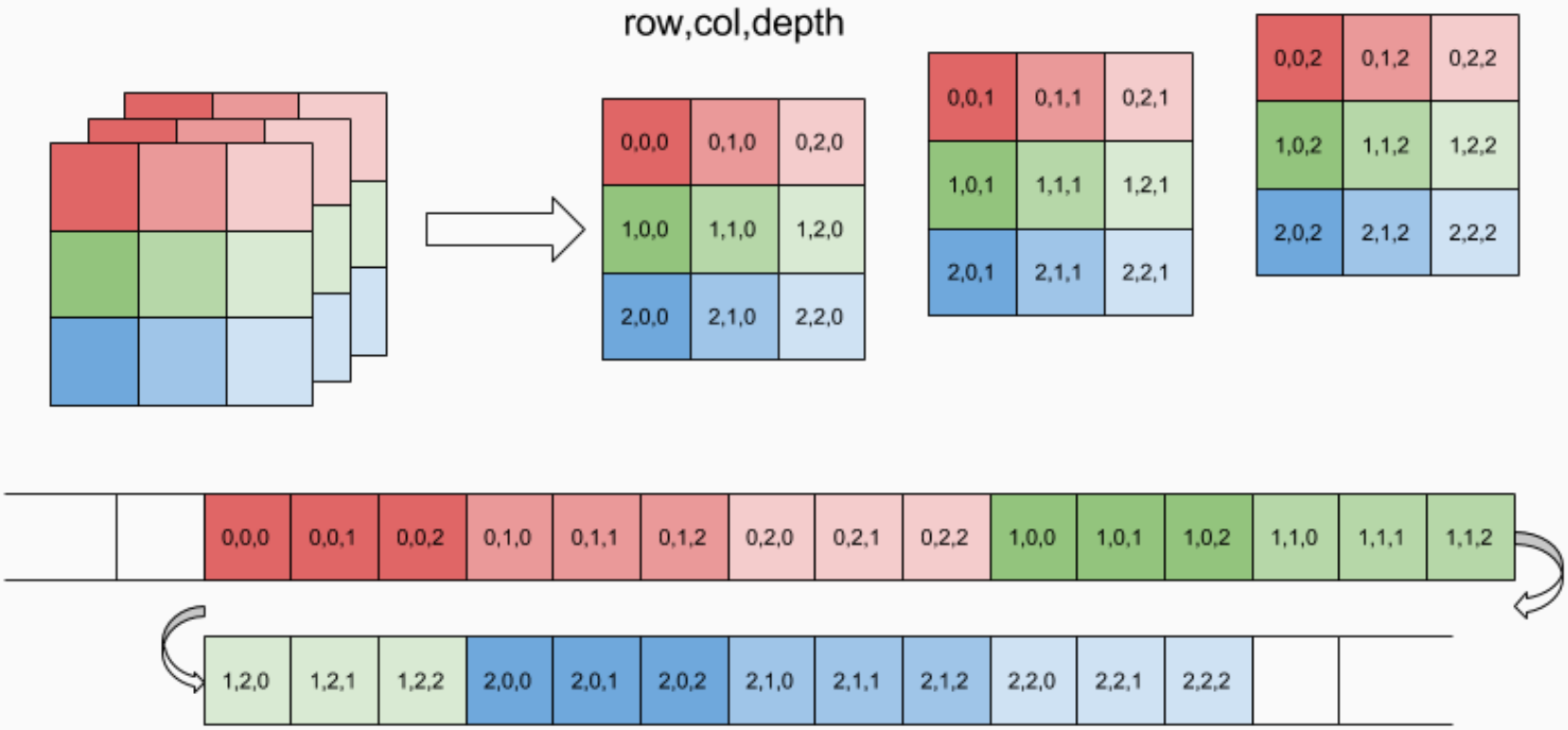

Figure 9: 3D row-major [Ben15]

Speicher-Layout - Demo

Demo

- Möglichst kleine Datentypen
  $\rightarrow$ Mehr Daten pro Cache-Line
- Padding:
  - "Leere" Speicherblöcke (Padding) für Alignment
    $\rightarrow$ Weniger Lesevorgänge auf Segmenten
  - Erhöhter Speicherplatzbedarf
  - Neuordnung der Daten reduziert Anzahl an Paddings
    $\rightarrow$ Mehr Daten pro Cache-Line

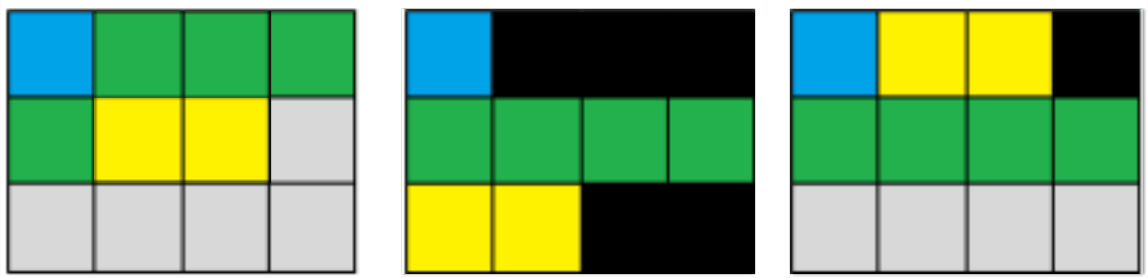

Figure 10: Datenanordnung mit/ohne Padding [Wis17]

```
1  struct s1
2  {
3      char a;
4      short a1;
5      char b1;
6      float b;
7      int c;
8      char e;
9      double f;
10 };
11 // Size of Struct s1 = 32
```

Listing 1: Ungeordneter Struct, 11 Padding-Bytes [Int13]

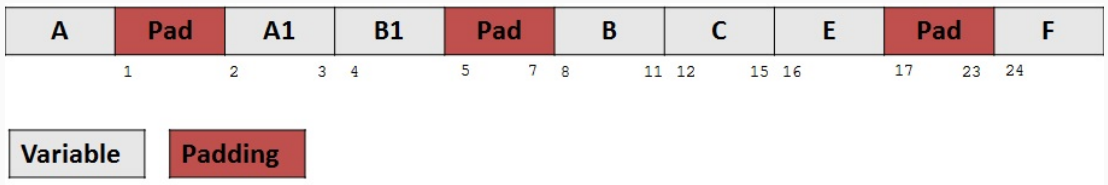

Figure 11: Ungeordneter Struct, 11 Padding-Bytes [Int13]

```
1  struct s2
2  {
3      double f;
4      float b;
5      int c;
6      short a1;
7      char a,b1,e;
8  };
9  // Size of Struct s2 = 24
```

Listing 2: Geordneter Struct, 3 Padding-Bytes [Int13]

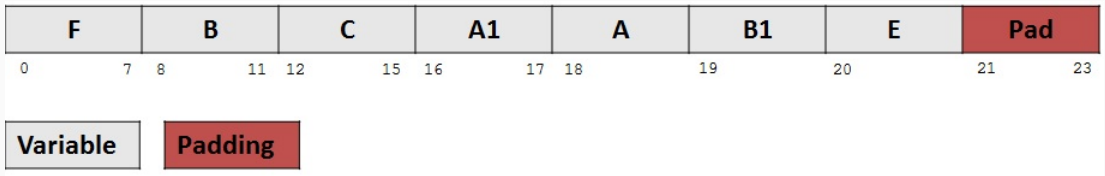

Figure 12: Geordneter Struct, 3 Padding-Bytes [Int13]

SoA vs. AoS

Ausnutzen von Lokalitätseigenschaften durch Anpassung:

Datenstruktur $\leftrightarrow$ Zugriffsmuster

```
1 struct {
2   uint r, g, b, w;
3 } MyAoS[N];
```

Listing 3: Array of structs [M.16]

```
1 struct {
2   uint r[N];
3   uint g[N];
4   uint b[N];
5   uint w[N];
6 } MySoA;
```

Listing 4: Struct of arrays [M.16]

Loop Blocking (I)

Unterteilung großer Matrizen in Chunks, die komplett in den Cache passen

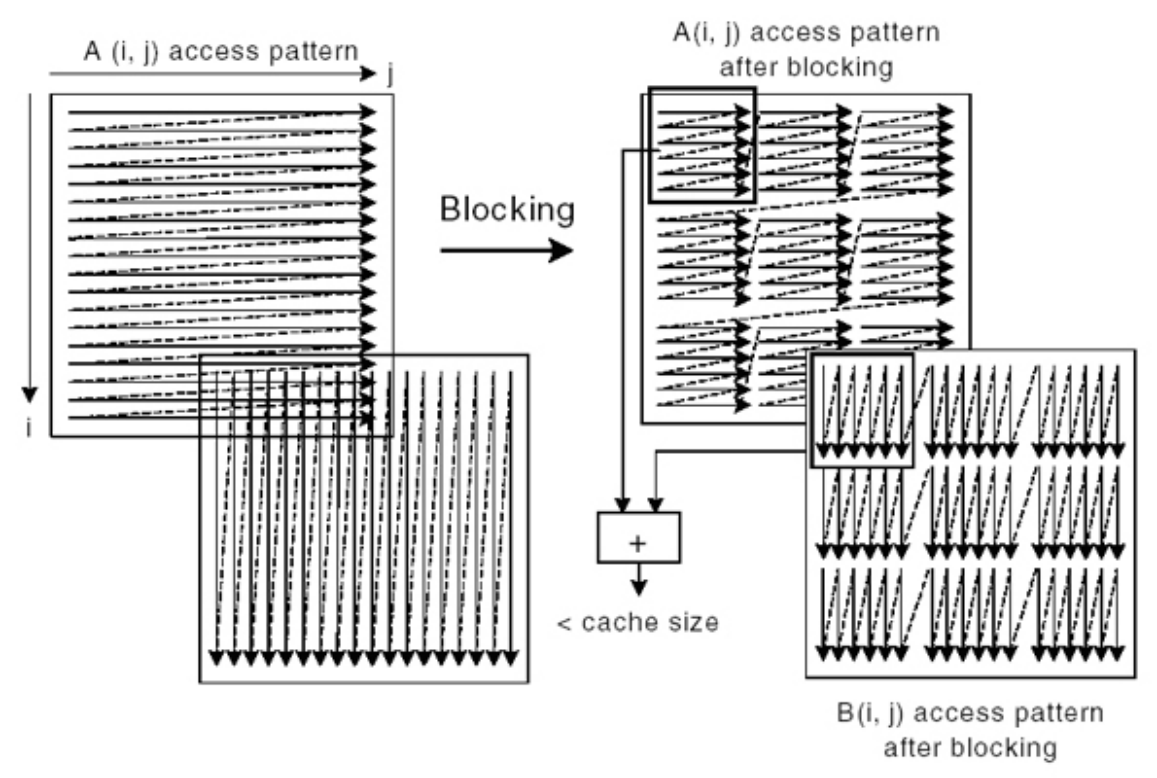

Figure 13: Unterteilung der Matrix in Chunks [Int08]

Loop Blocking (II)

```
1  float A[MAX, MAX], B[MAX, MAX];
2  for (i=0; i< MAX; i++)
3    for (j=0; j< MAX; j++)
4      A[i,j] = A[i,j] + B[j, i];
```

Listing 5: Original [Int08]

```
1  float A[MAX, MAX], B[MAX, MAX];
2  for (i=0; i< MAX; i+=block_size)
3    for (j=0; j< MAX; j+=block_size)
4      for (ii=i; ii<i+block_size; ii++)
5        for (jj=j; jj<j+block_size; jj++)
6          A[ii,jj] = A[ii,jj] + B[jj, ii];
```

Listing 6: Verbessert [Int08]

Ausblick: False sharing

- Schreiben auf Cache-Line *kann* Neuladen erzwingen
- Nebenläufiges Lesen $\rightarrow$ unnötiger Cache-Miss
- Finden und beheben mittels Tools und Compiler-Instruktionen

```
1  struct foo {
2      int x;
3      int y;
4  };
5  static struct foo f;
6
7  /* sum_a und inc_b nebenlaeufig */
8  int sum_a(void)
9  {
10     int s = 0;
11     int i;
12     for (i = 0; i < 1000000; ++i)
13         s += f.x;
14     return s;
15 }
16 void inc_b(void)
17 {
18     int i;
19     for (i = 0; i < 1000000; ++i)
20         ++f.y;
21 }
```

Listing 7: Beispiel

Table of Contents

The End ...

Fragen?

References

[Ben15] Eli Bendersky, *Memory layout of multi-dimensional arrays*, 2015.

[cac] *Functional principles of cache memory*, http://alasir.com/articles/cache_principles/cache_line_tag_index.html.

[Dre07] Ulrich Drepper, *What every programmer should know about memory*, Red Hat, Inc **11** (2007), 2007.

[Gri07] Gribeco, *Motherboard diagram*, https://commons.wikimedia.org/wiki/File:Motherboard_diagram.svg, 2007.

[Gü19] Alexander Günther, *Speichergeräte*, https://wr.informatik.uni-hamburg.de/_media/teaching/wintersemester_2018_2019/sds-1819-guenther-speichergeraete-praesentatio pdf, 2019.

[Hru14] Joel Hruska, *haswell die*, http://www.extremetech.com/wp-content/uploads/2014/08/haswell-e-die-shot-high-res.jpg, 2014.

[htt] https://cvw.cac.cornell.edu, *Memory access times*, https://cvw.cac.cornell.edu/codeopt/memtimes.

[Int08] Intel, *How to use loop blocking to optimize memory use on 32-bit intel architecture*, https://software.intel.com/en-us/articles/how-to-use-loop-blocking-to-optimize-memory-u 2008.

[Int13] _____, *Coding for performance: Data alignment and structures*, https://software.intel.com/en-us/articles/coding-for-performance-data-alignment-and-stru 2013.

[M.16] David M., *Putting your data and code in order: Data and layout - part 2*, https://software.intel.com/en-us/articles/putting-your-data-and-code-in-order-data-and- 2016.

[SCJ13] Subhash Saini, Johnny Chang, and Haoqiang Jin, *Performance evaluation of the intel sandy bridge based nasa pleiades using scientific and engineering applications*, International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems, Springer, 2013, pp. 25–51.

[Wis17] Tim Wischof, *Memory management and optimizations*, techreport, Universität Hamburg, 2017.