

Work Partitioning

Aufteilung einer Berechnung für die parallele Verarbeitung

Sebastian Speiser

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

Gegeben eine
Matrix

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

The matrix is filled with black squares at the following coordinates: (3, 3), (3, 4), (3, 5), (4, 3), (4, 4), (4, 5), (5, 3), (5, 4), (5, 5). This forms a 3x3 square centered at (3, 4).

Gegeben eine
Matrix, z.B. ein
Bild

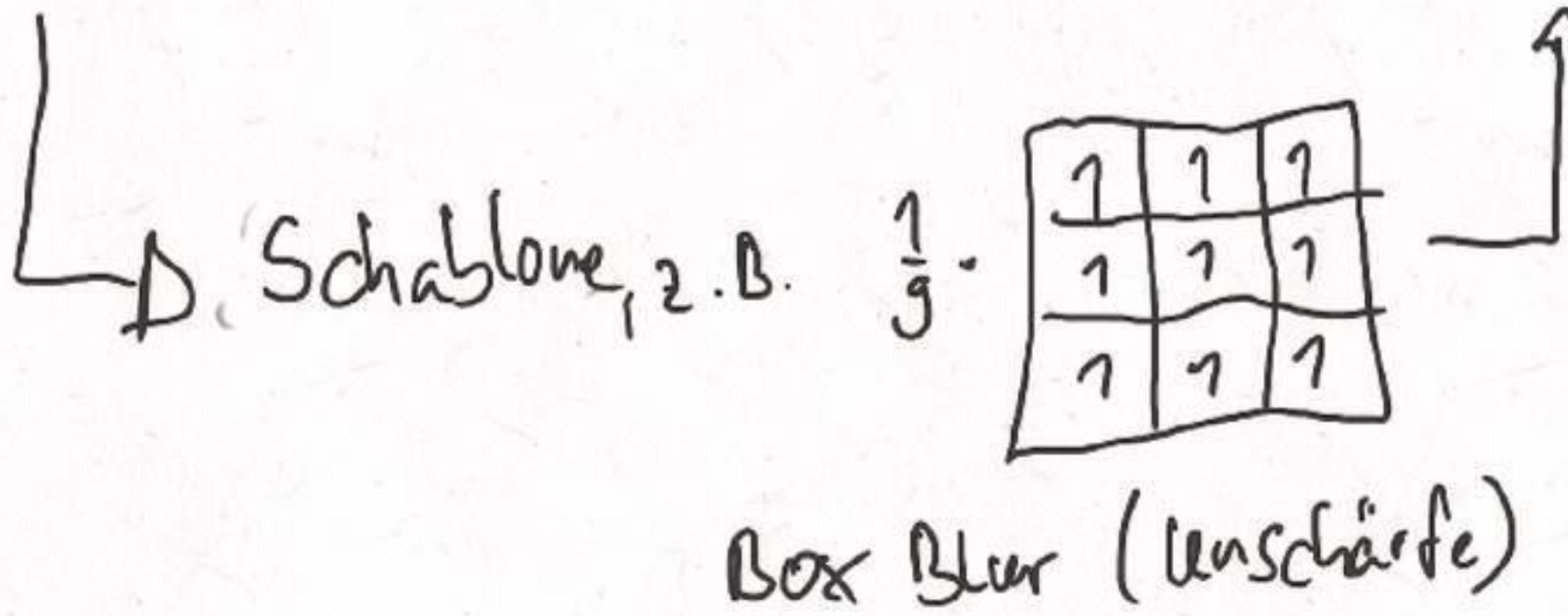
	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4								
5								
6								
7								

The matrix is filled with black squares representing a 3x3 pattern. The pattern is centered at index (4,4) and spans from index (3,3) to (6,6). The indices are labeled from 0 to 7 along both the horizontal and vertical axes.

Gegeben eine
Matrix, z.B. ein
Bild
(von einem Hand!-)

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

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



Gegeben eine Matrix, z.B. ein Bild (von einem Hand! -)

Es wird eine Schablone angewandt, d.h. jeder Wert (Pixel) im Ziel ist eine gewichtete Summe der umgebenden Pixel

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

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

Jeder Zelwert wird aus der Ausgangsmatrix berechnet

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

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

Jeder Zielwert wird aus der Ausgangsmatrix berechnet

Jeder Zielwert kann unabhängig berechnet werden

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

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

Jeder Zielwert wird aus der Ausgangsmatrix berechnet

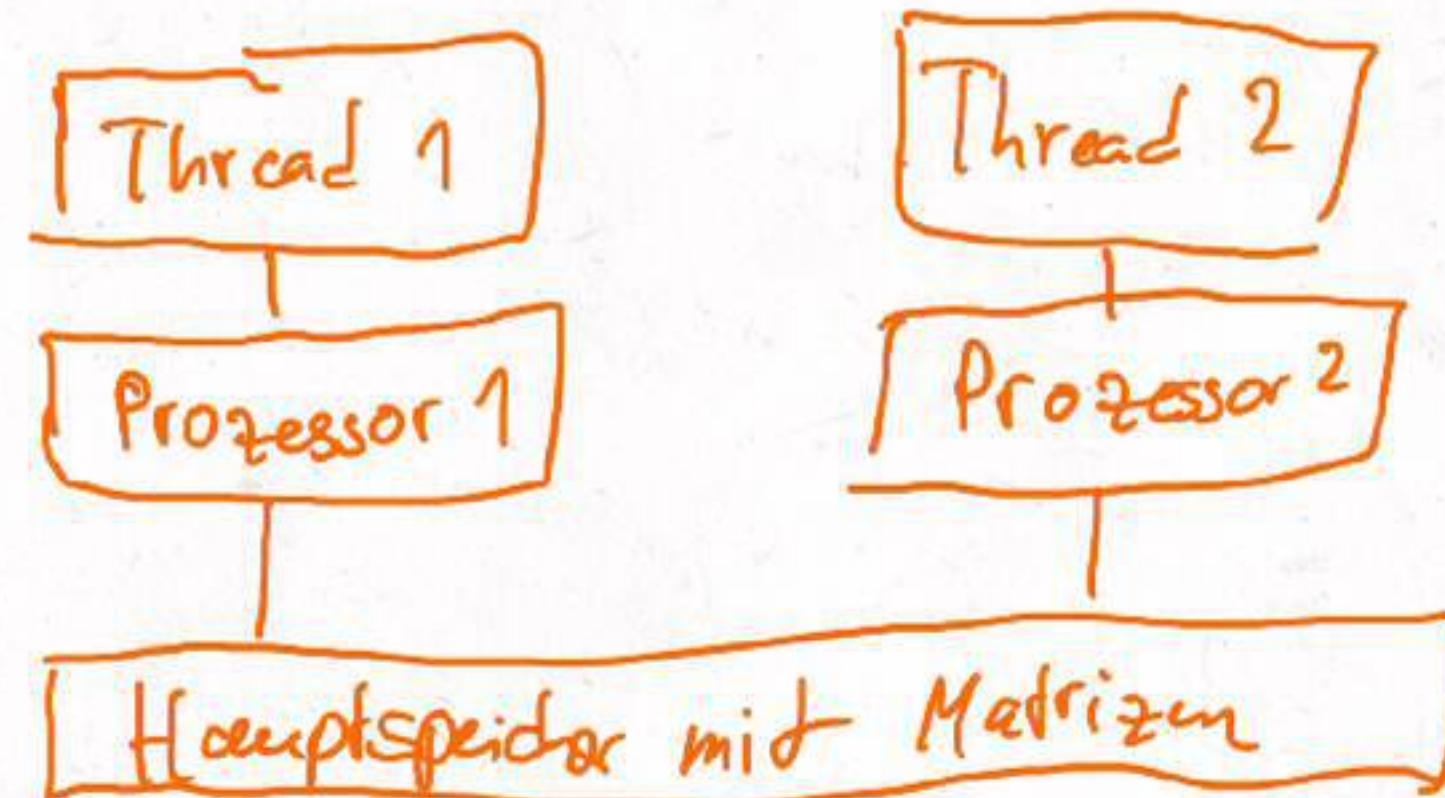
Jeder Zielwert kann unabhängig berechnet werden

Lächerlich einfach zu parallelisieren

Wir weisen die Pixel einfach
irgendwie mehreren Threads zu

Wir weisen die Pixel enthal
irgendwie mehreren Threads zu

Die Threads laufen dann parallel
auf Prozessoren :)

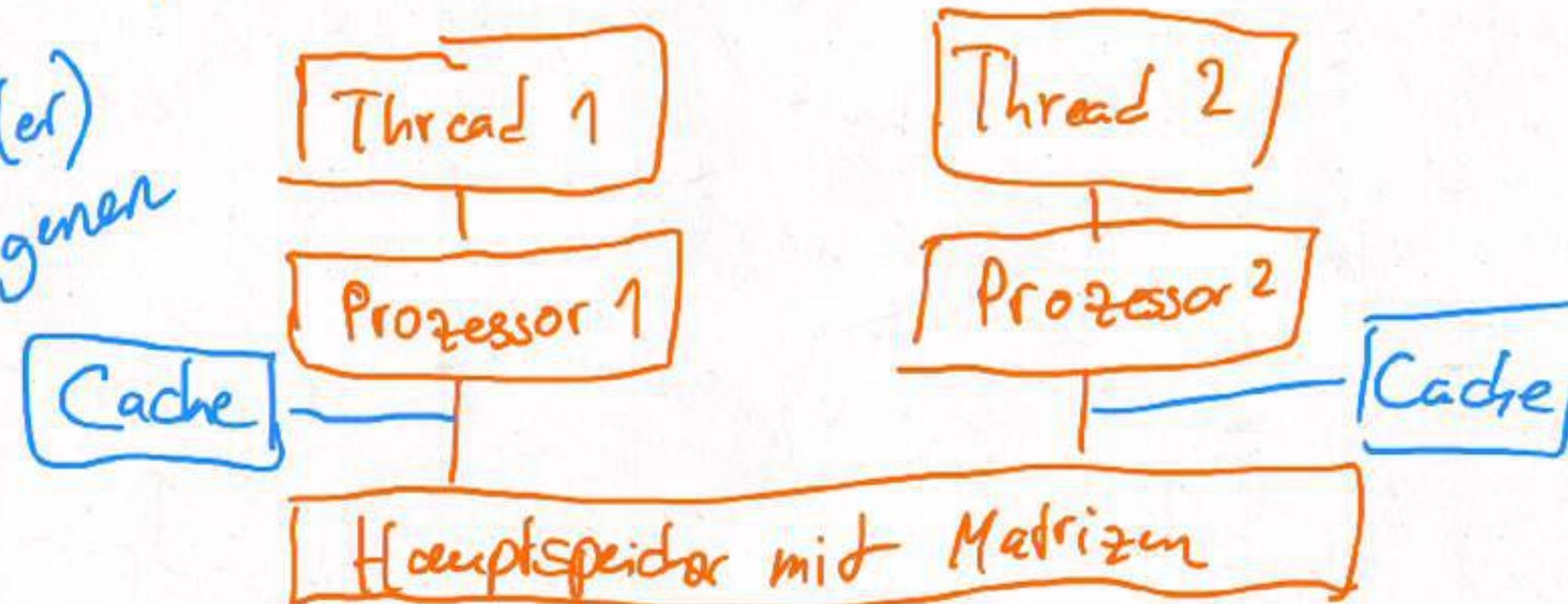


Wir weisen die Pixel einfach
irgendwie mehreren Threads zu

Die Threads laufen dann parallel
auf Prozessoren :)

Aber:

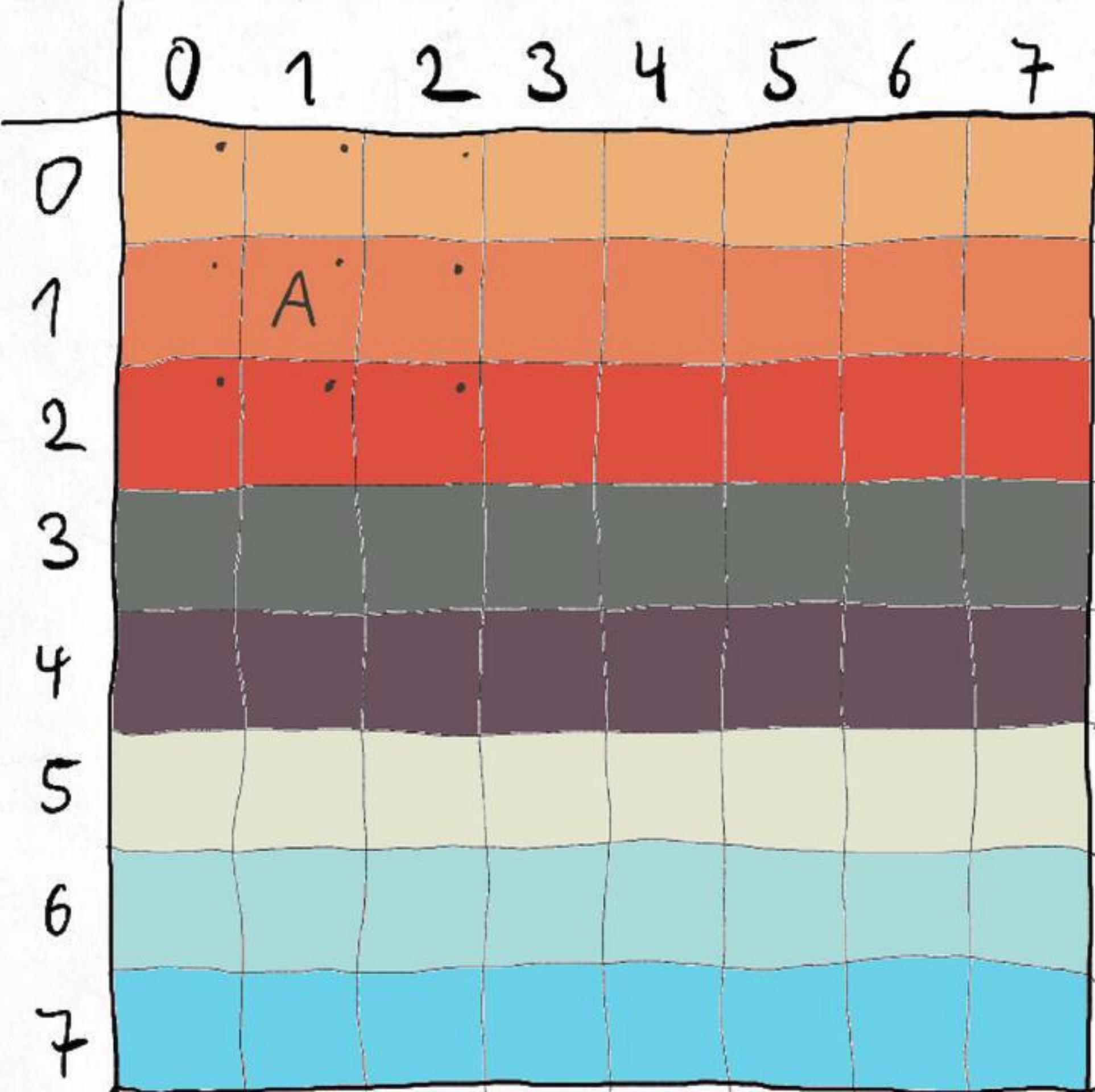
- Hauptspeicher ist langsam
- Cache sind schnell(er)
- Cache sind eigener
- Jeder Prozessor hat eigenen Cache



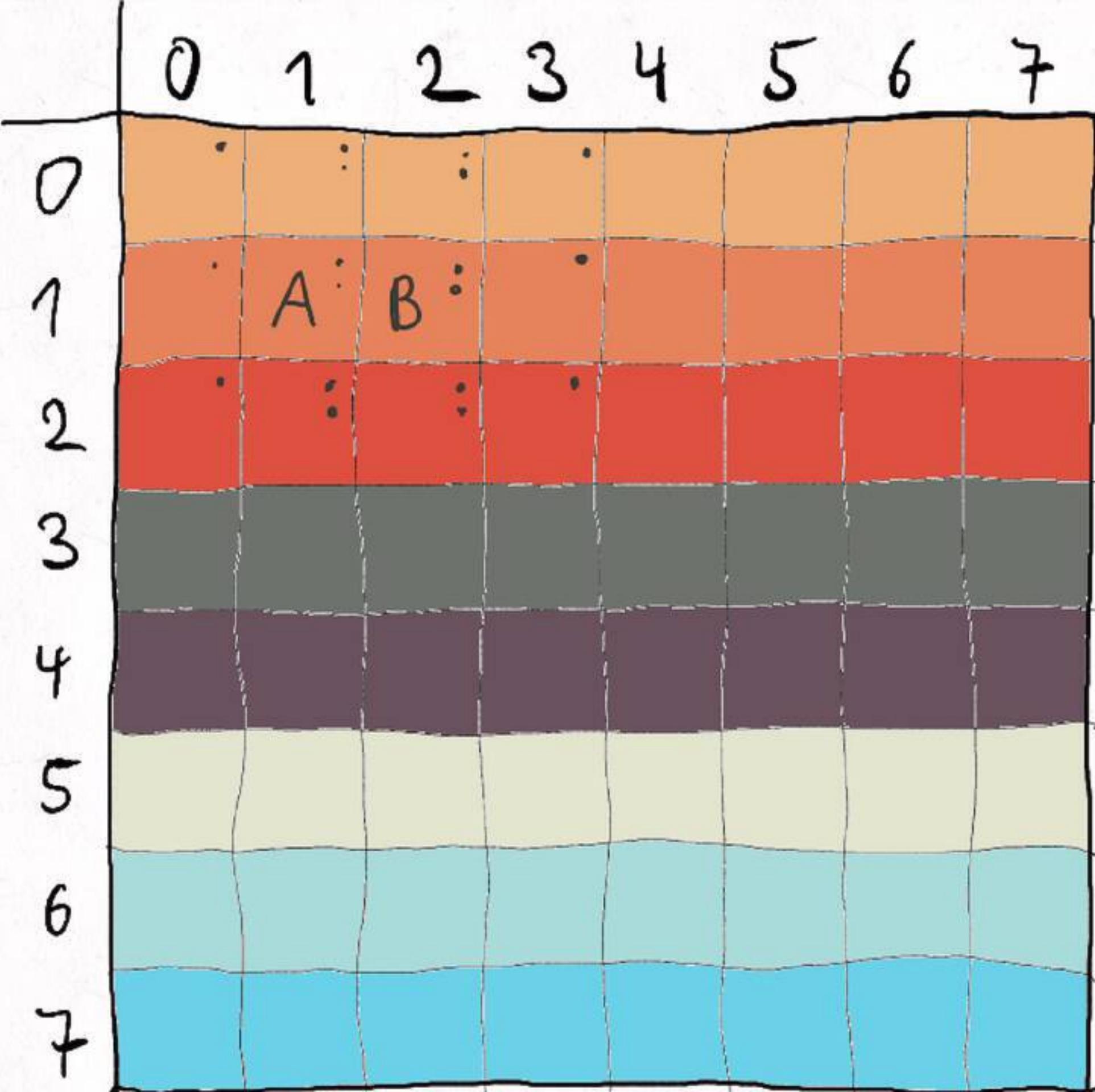
Wie teilen wir die Matrix
auf Threads auf, wenn wir
die Cache-Nutzung optimieren wollen
damit der Hauptspeicher nicht die
CPU ausbraust?

	0	1	2	3	4	5	6	7
0	orange							
1	orange							
2	red							
3	grey							
4	dark purple							
5	light green							
6	teal							
7	blue							

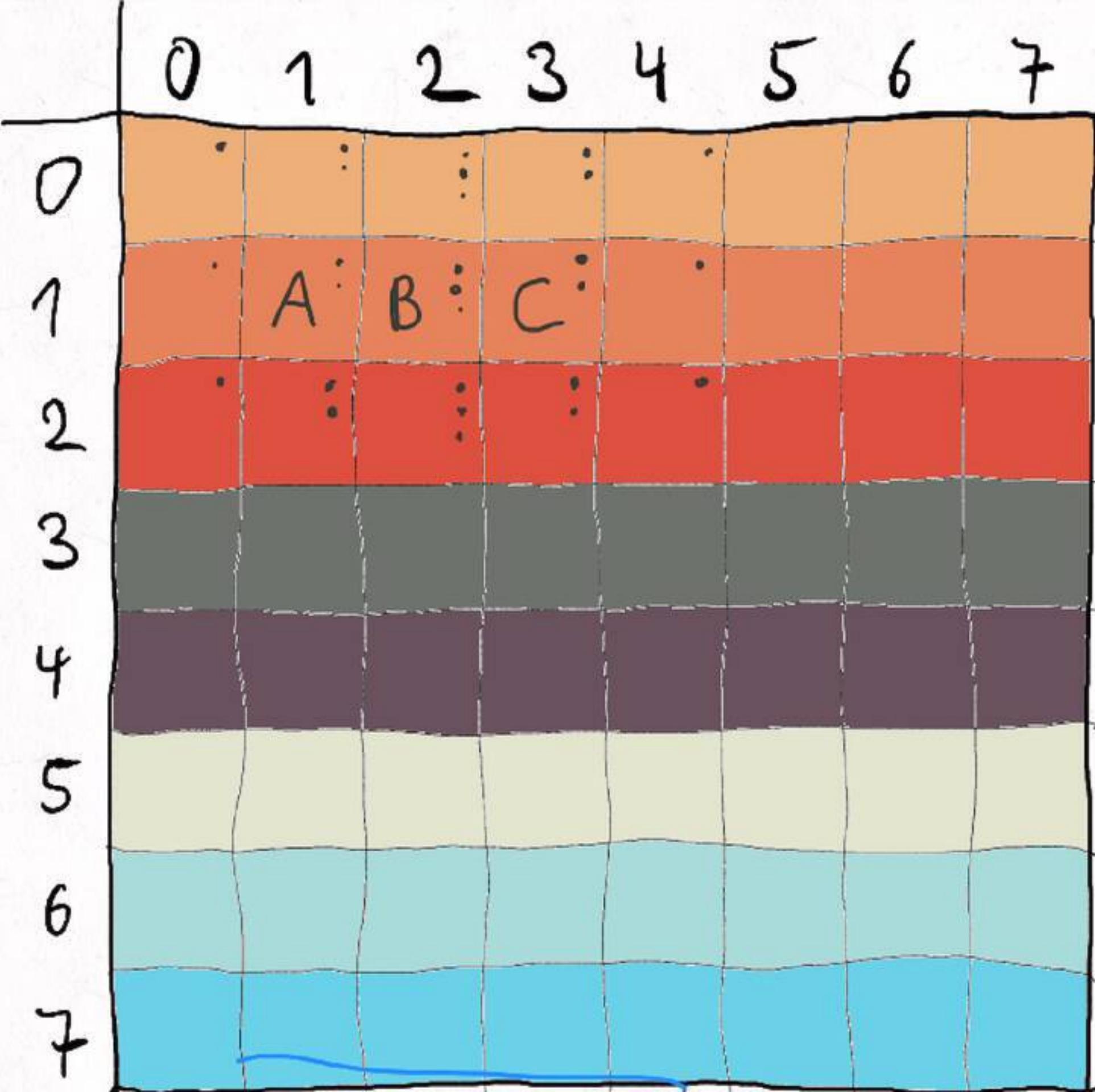
Erster Ansatz:
 Zeilenweise
 Aufteilung
 Jede Farbe stellt
 einen anderen
 Thread dar
 (hier: 8 Zeilen mit
 je 8 Spalten und
 8 Threads / CLBs
 ... wie praktisch)



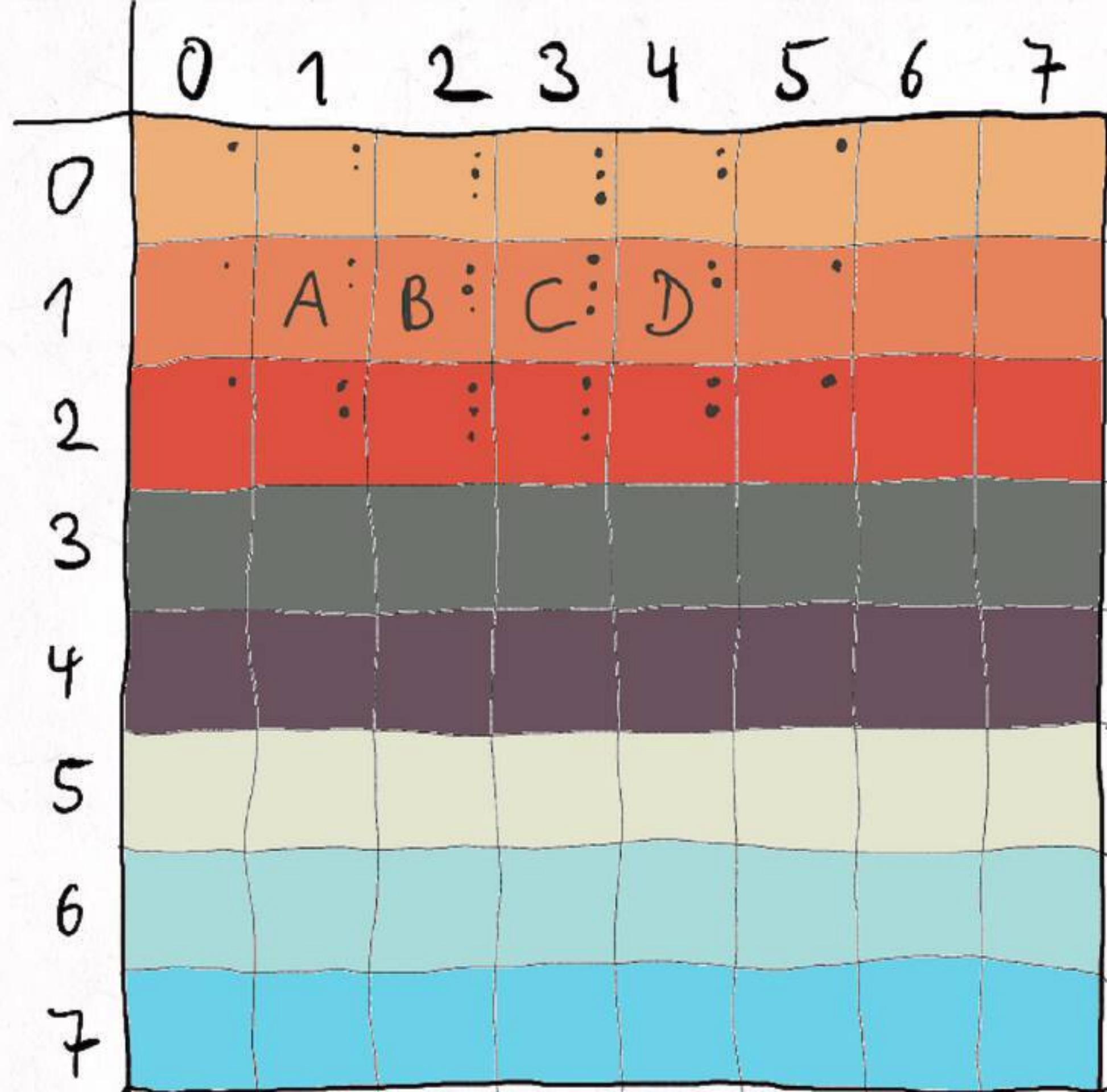
Thread 1
New Cached
A 9 0



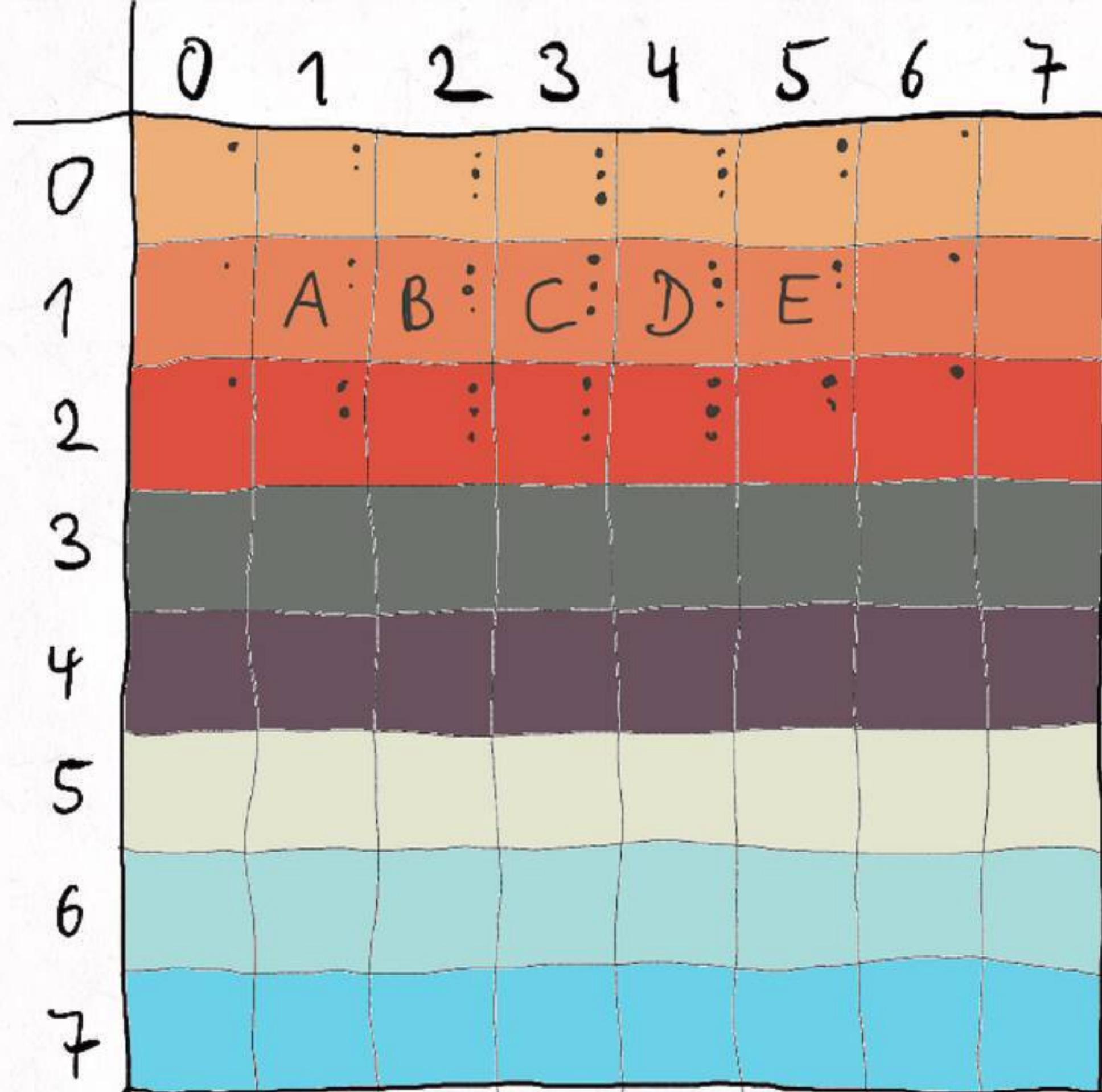
Thread 1	New	Cached
A	9	Ø
B	3	6



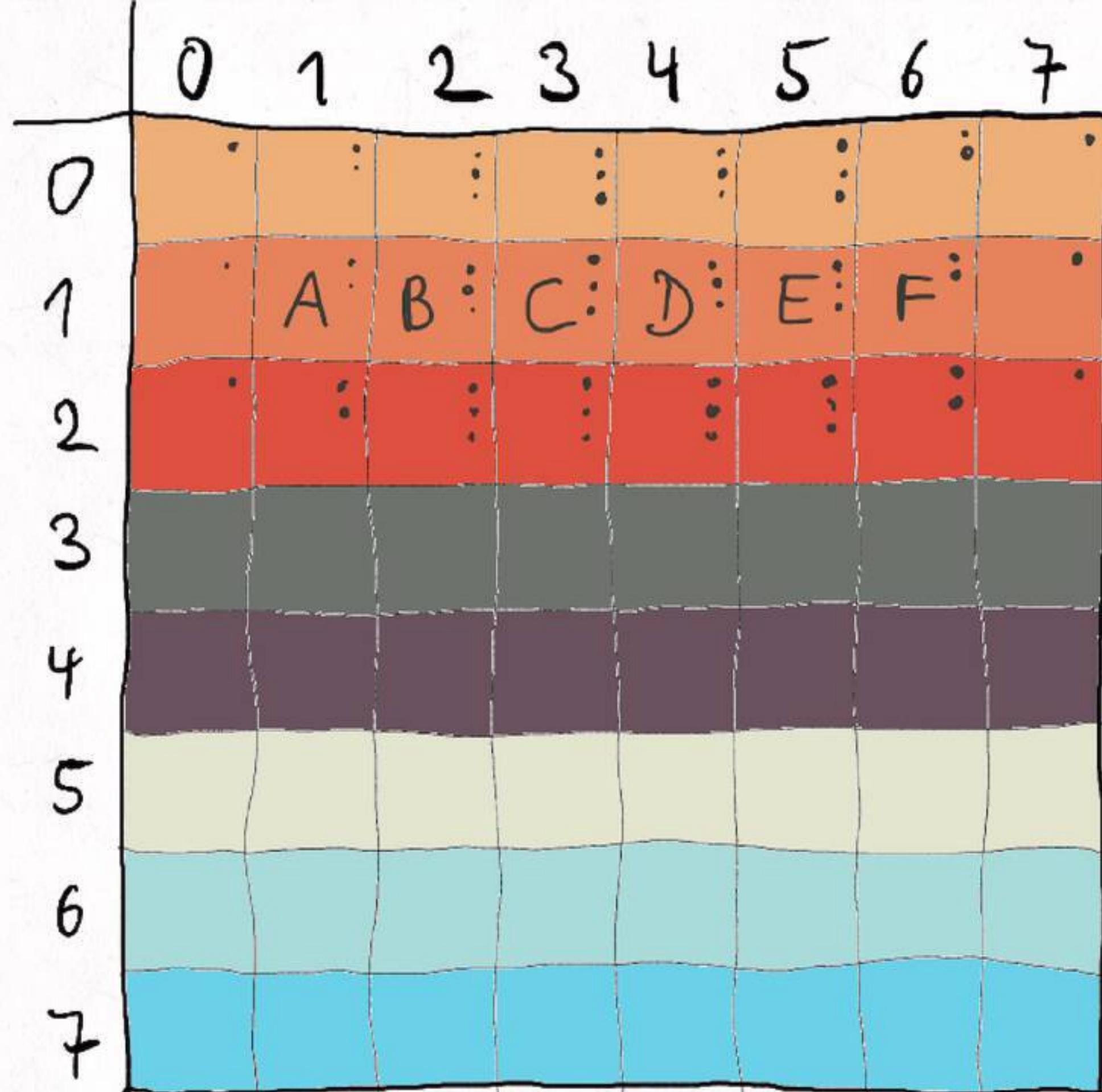
Thread 1	New	Cached
A	9	Ø
B	3	6
C	3	6



Thread 1		
	New	Cached
A	9	Ø
B	3	6
C	3	6
D	3	6



Thread 1		
	New	Cached
A	9	Ø
B	3	6
C	3	6
D	3	6
E	3	6



Thread 1		
	New	Cached
A	9	Ø
B	3	6
C	3	6
D	3	6
E	3	6
F	3	6

	0	1	2	3	4	5	6	7
0	:	:	:	:	:	:	:	:
1	:	A	B	C	D	E	F	G
2	:	:	:	:	:	:	:	:
3								
4								
5								
6								
7								

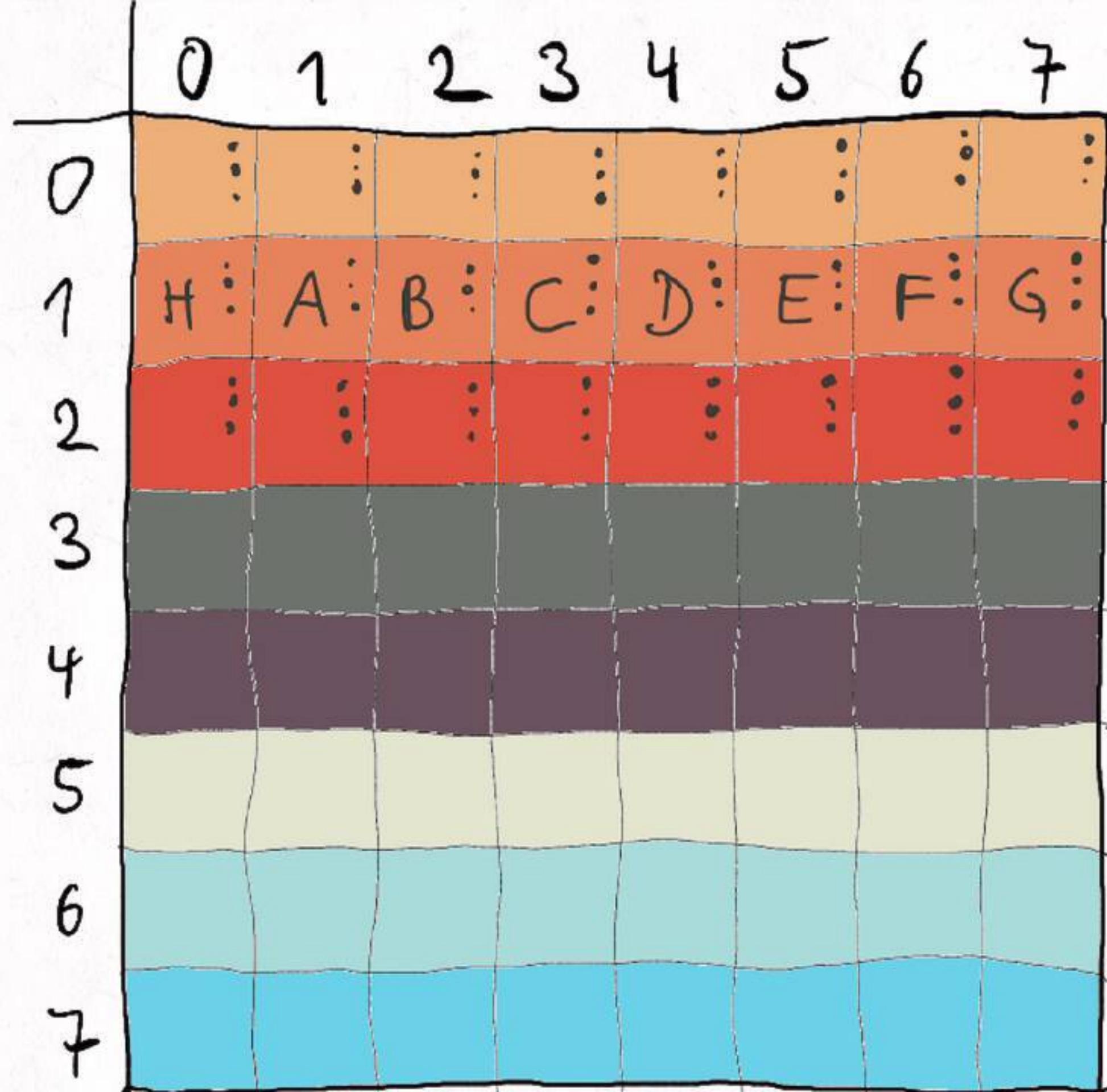
Thread 1	New	Cached
A	9	Ø
B	3	6
C	3	6
D	3	6
E	3	6
F	3	6
G	0	9

	0	1	2	3	4	5	6	7
0	:	:	:	:	:	:	:	:
1	H	A	B	C	D	E	F	G
2	:	:	:	:	:	:	:	:
3								
4								
5								
6								
7								

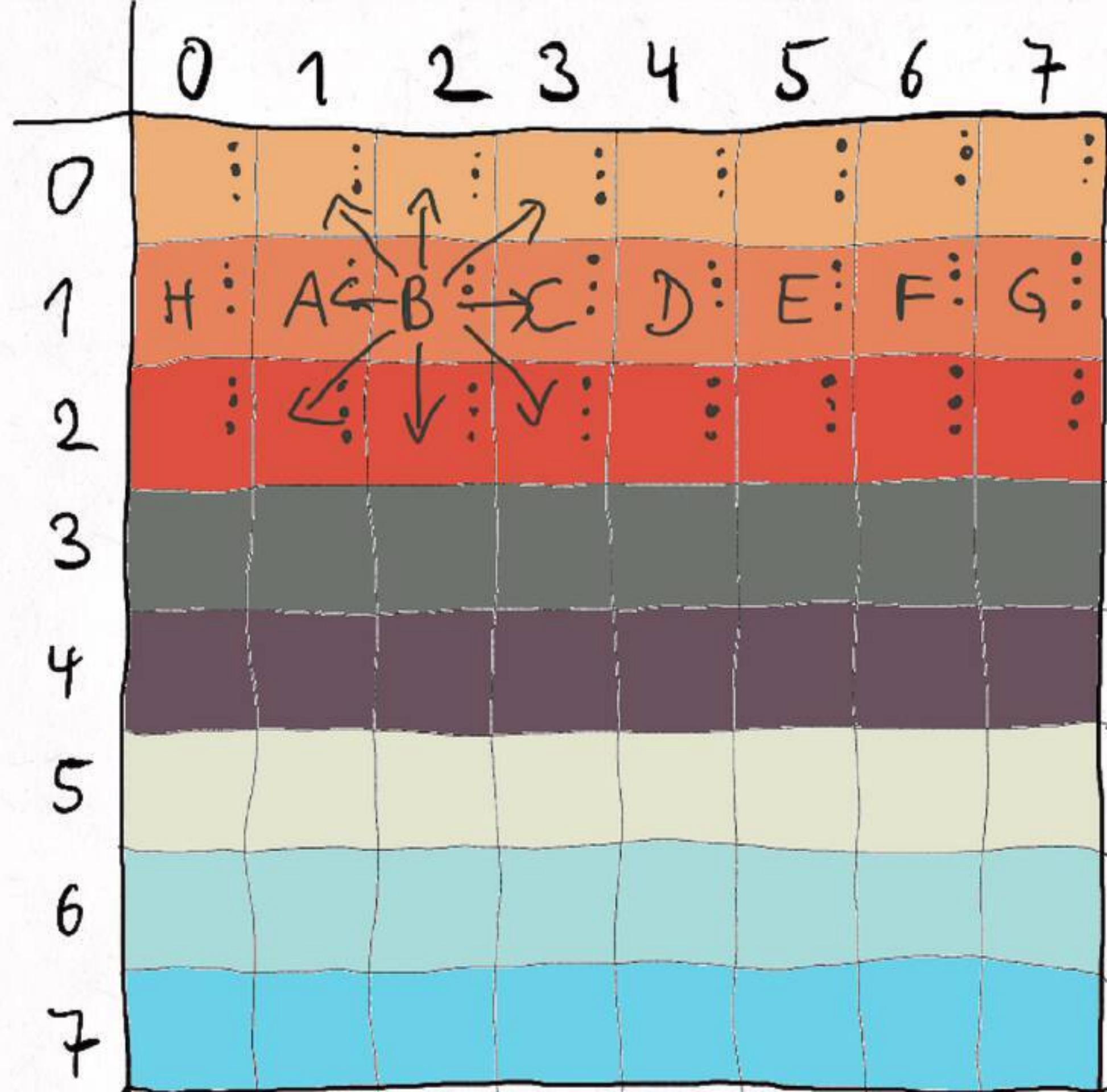
Thread 1	New	Cached
A	9	Ø
B	3	6
C	3	6
D	3	6
E	3	6
F	3	6
G	0	9
H	0	9

	0	1	2	3	4	5	6	7
0	:	:	:	:	:	:	:	:
1	H	A	B	C	D	E	F	G
2	:	:	:	:	:	:	:	:
3								
4								
5								
6								
7								

Thread 1	New	Cached
A	9	Ø
B	3	6
C	3	6
D	3	6
E	3	6
F	3	6
G	0	9
H	0	9
		$\frac{8 \times 9}{24 + 48} = 72$



Thread 1
24 Werte aus Speicher
in Cache geladen



Thread 1
24 Werte aus Speicher
in Cache geladen

Jeder Wert wird
auch noch in 2
weiteren Threads
genutzt

Jeder Thread (1 pro Kern)
hat eigenen Cache
Muss also erneut geladen
werden

8 Threads X 24 Reads = 192

	0	1	2	3	4	5	6	7
0	orange							
1		orange						
2			red					
3				grey				
4					dark purple			
5						light green		
6							light blue	
7								blue

- Geschrieben wird in separates Ziel - Array
- Austausch Ziel- und Quell - Array
- Erneute Iteration

	0	1	2	3	4	5	6	7
0								
1	x	x	x					
2	✓	A ✓	✓					
3	x	x	x					
4								
5								
6								
7								

- Geschrieben wird in separates Ziel - Array
 - Austausch Ziel- und Quell - Array
 - Erneute Iteration
- Im eigenen Thread berechnete Werte sind im Cache - Rest muss gelesen werden

	0	1	2	3	4	5	6	7
0								
1	x	x	x					
2	✓	A ✓	✓					
3	x	x	x					
4								
5								
6								
7								

Im eigenen Thread berechnete Werte sind im Cache - Rest muss gelesen werden

New Cache

A	6	3
---	---	---

	0	1	2	3	4	5	6	7
0								
1	x	x	x	x				
2	✓	A ✓	B ✓	✓				
3	x	x	x	x				
4								
5								
6								
7								

Im eigenen Thread berechnete Werte sind im Cache - Rest muss gelesen werden

	New	Cache
A	6	3
B	2	7

	0	1	2	3	4	5	6	7
0								
1	x	x	x	x	x			
2	v	A v	B v	C v	v			
3	x	x	x	x	x			
4								
5								
6								
7								

Im eigenen Thread berechnete Werte sind im Cache - Rest muss gelesen werden

	New	Cache
A	6	3
B	2	7
C	2	7

	0	1	2	3	4	5	6	7
0								
1	x	x	x	x	x	x		
2	v	A v	B v	C v	D v	v		
3	x	x	x	x	x	x		
4								
5								
6								
7								

Im eigenen Thread berechnete Werte sind im Cache - Rest muss gelesen werden

	New	Cache
A	6	3
B	2	7
C	2	7
D	2	7

	0	1	2	3	4	5	6	7
0								
1	x	x	x	x	x	x	x	
2	v	A v	B v	C v	D v	E v	v	
3	x	x	x	x	x	x	x	
4								
5								
6								
7								

Im eigenen Thread berechnete Werte sind im Cache - Rest muss gelesen werden

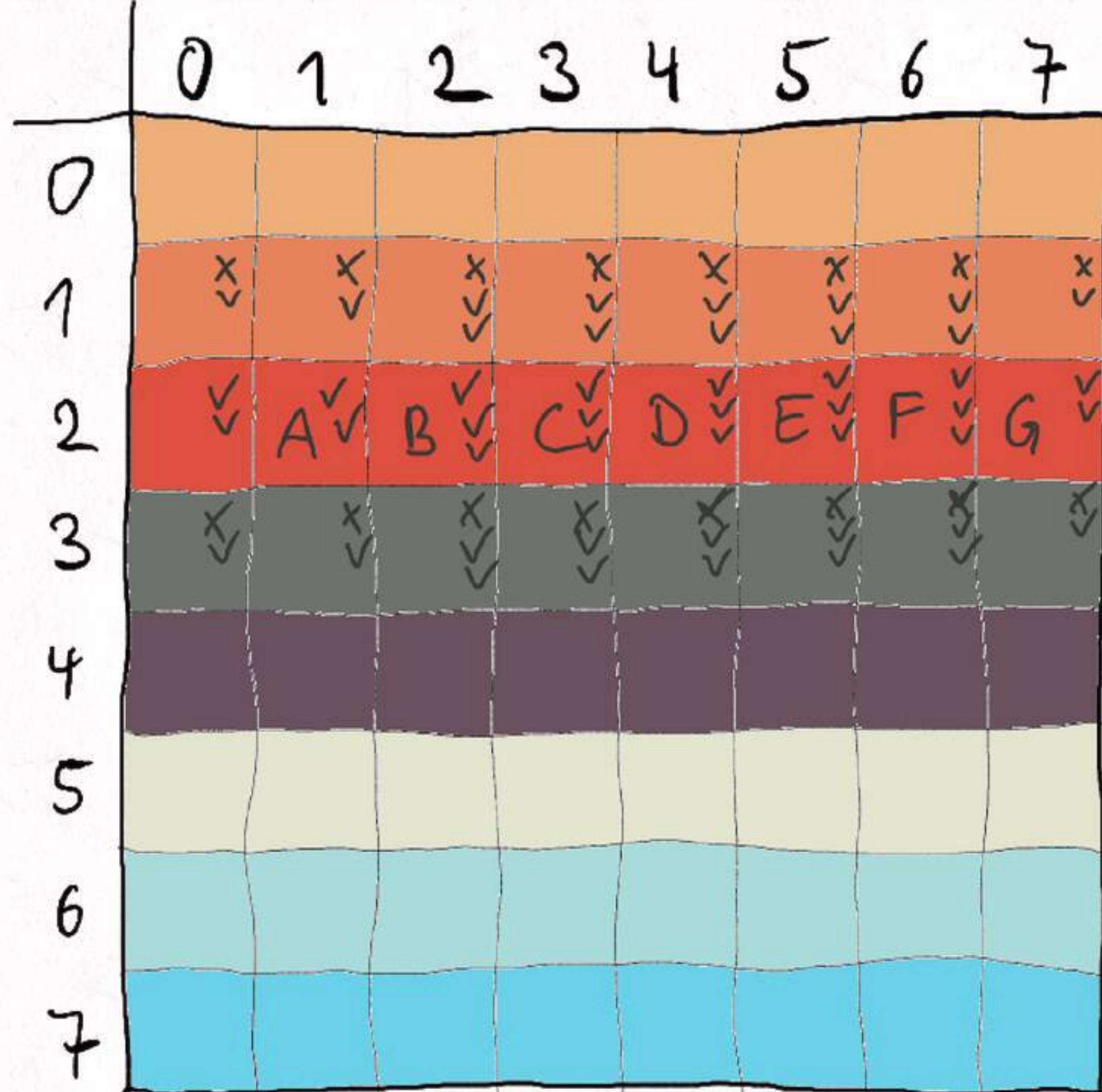
New Cache

A	6	3
B	2	7
C	2	7
D	2	7
E	2	2

	0	1	2	3	4	5	6	7
0								
1	x	x	x	x	x	x	x	x
2	v	A	B	C	D	E	F	v
3	x	x	x	x	x	x	x	x
4								
5								
6								
7								

Im eigenen Thread berechnete Werte sind im Cache - Rest muss gelesen werden

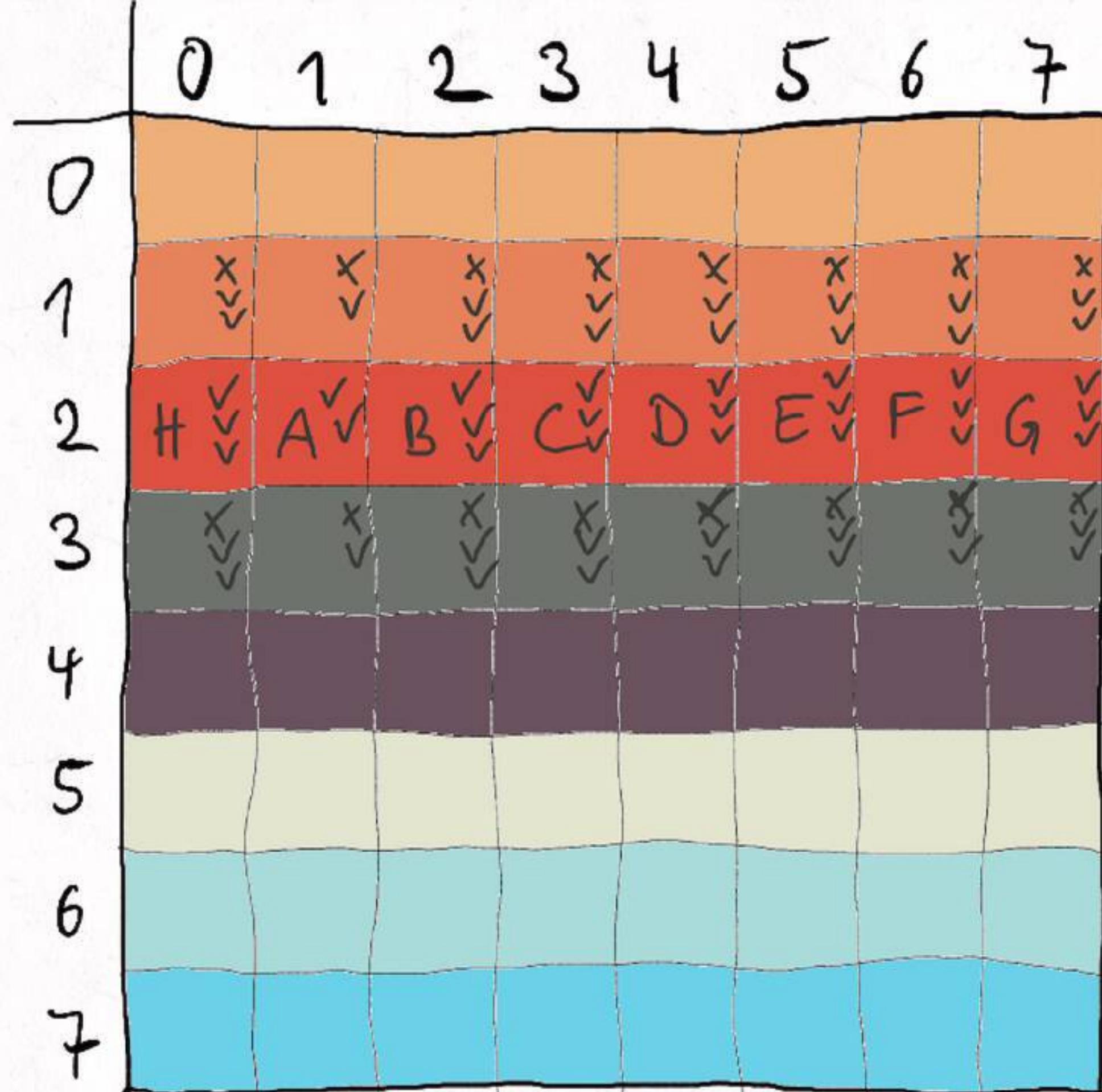
	New	Cache
A	6	3
B	2	7
C	2	7
D	2	7
E	2	7
F	2	7



Im eigenen Thread berechnete Werte sind im Cache - Rest muss gelesen werden

New Cache

A	6	3
B	2	7
C	2	7
D	2	7
E	2	7
F	2	7
G	0	9



Im eigenen Thread berechnete Werte sind im Cache - Rest muss gelesen werden

New Cache

A	6	3
B	2	7
C	2	7
D	2	7
E	2	7
F	0	7
G	0	9
H	0	9

	0	1	2	3	4	5	6	7
0								
1	x v	x v	x v	x v	x v	x v	x v	x v
2	H v v	A v	B v	C v	D v v	E v v	F v v	G v v
3	x v	x v	x v	x v	x v	x v	x v	x v
4								
5								
6								
7								

Im eigenen Thread berechnete Werte sind im Cache - Rest muss gelesen werden

New Cache

A	6	3
B	2	7
C	2	7
D	2	7
E	2	7
F	2	7
G	0	9
H	0	9

$$76 + 56 = 72 (8 \times 9)$$

Wie computer-intensive ist unser Ansatz?

$$8 \times 16 = 128 \text{ Reads pro Durchgang}$$

Wie computer-intensive ist unser Ansatz?

$$8 \times 16 = 128 \text{ Reads pro Durchgang}$$

$$+ 8 \times 8 = 64 \text{ Writes pro Durchgang}$$

jeder Wert wird von einem anderen Thread
ausgelesen - muss also in den Speicher

Wie computer-intensive ist unser Ansatz?

$8 \times 16 = 128$ Reads pro Durchgang

+ $8 \times 8 = 64$ Writes pro Durchgang

jeder Wert wird von einem anderen Thread
ausgelesen - muss also in den Speicher

$\overline{192} \times 4$ Bytes (Single Prec. Float) = 768 Bytes

Floating Point Ops: 64 Felder $\times (1$ Multiplikation + 8 Additionen) = 576

Wie computer-intensive ist unser Ansatz?

$8 \times 16 = 128$ Reads pro Durchgang

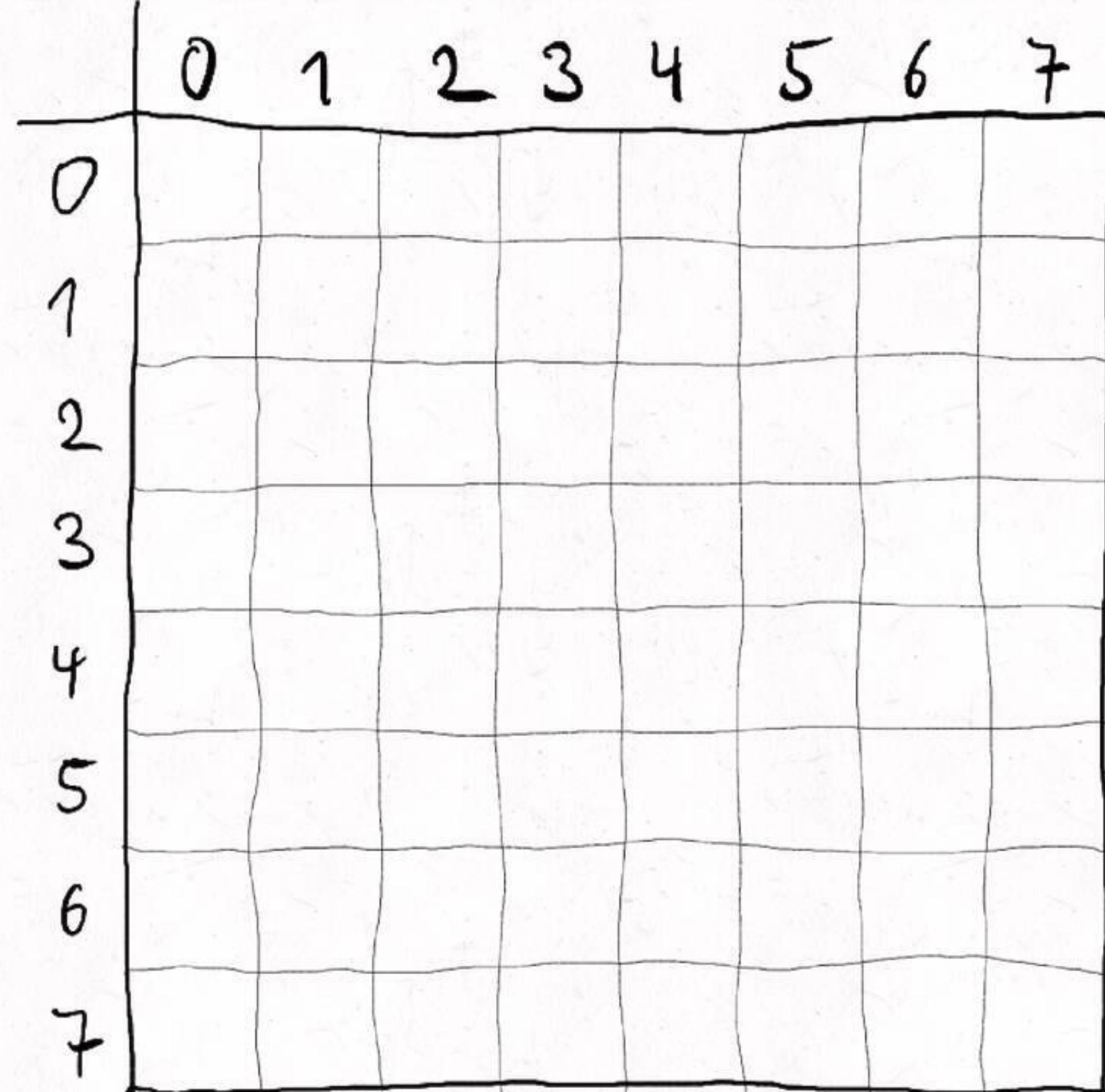
+ $8 \times 8 = 64$ Writes pro Durchgang

jeder Wert wird von einem anderen Thread
ausgelesen - muss also in den Speicher

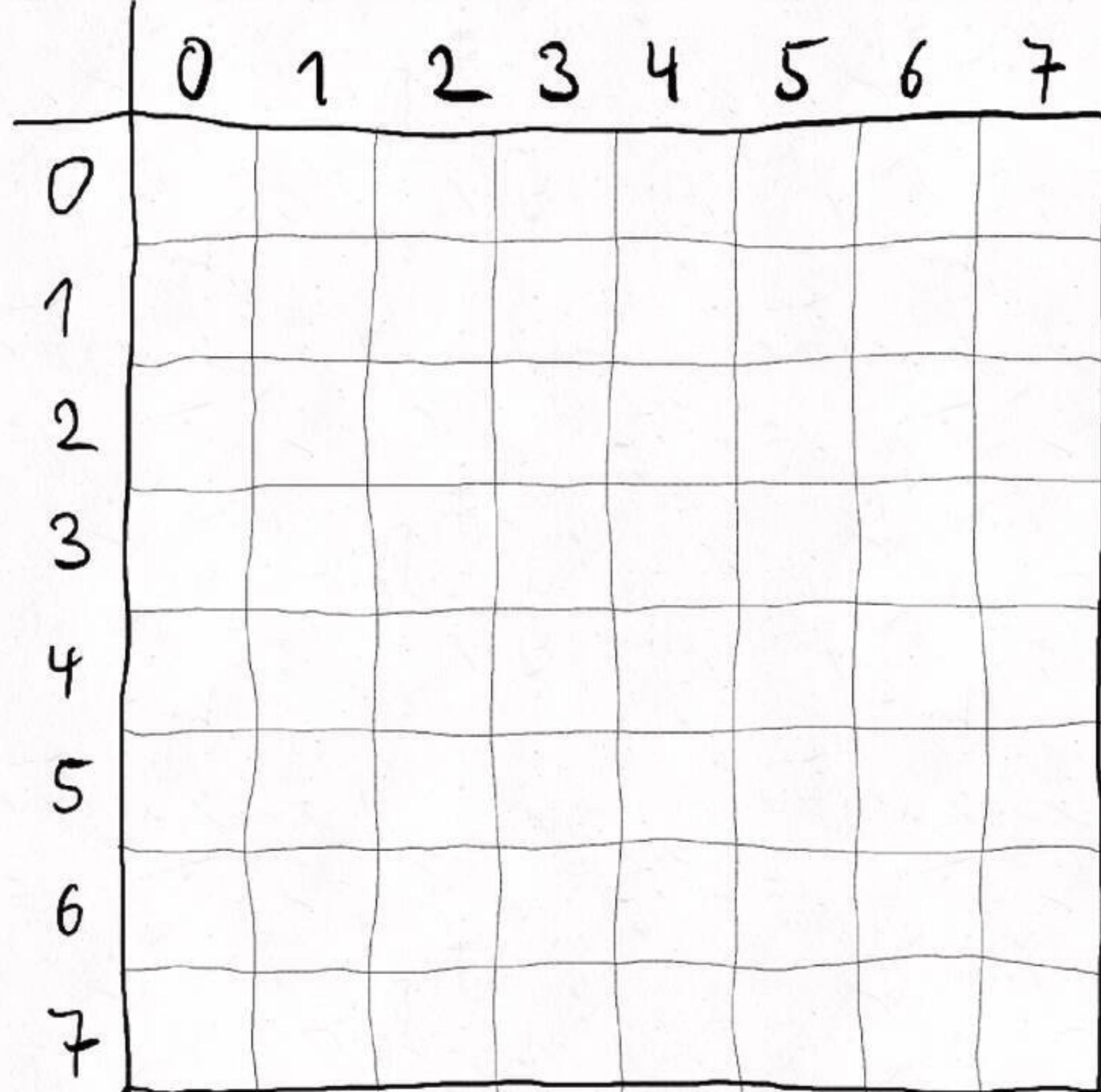
$\overline{192} \times 4$ Bytes (Single Prec. Float) = 768 Bytes

Floating Point Ops: 64 Felder $\times (1 \text{ Multiplikation} + 8 \text{ Additionen}) = 576$

Intensität: $576 / 768 = 0,75$

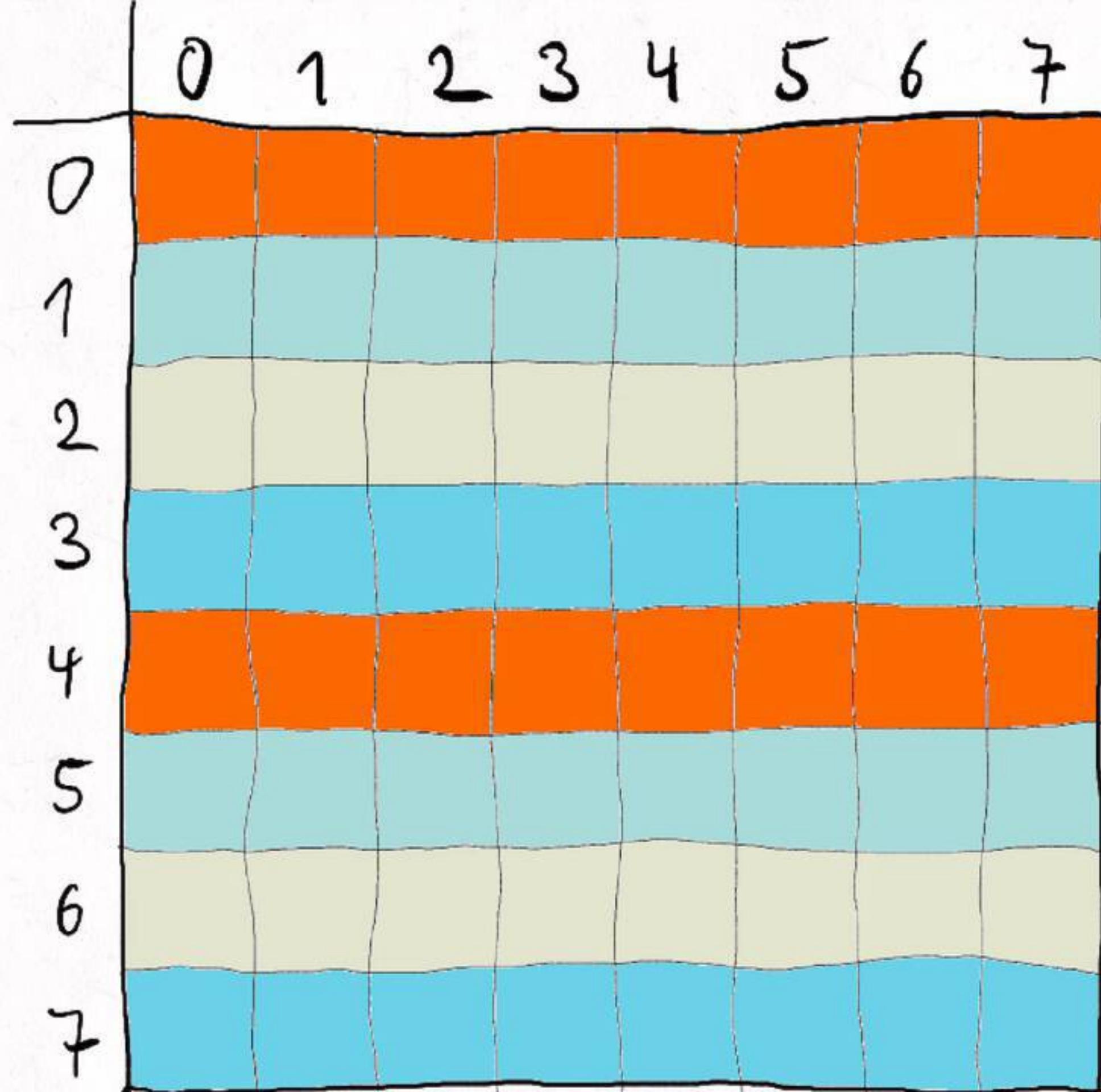


Ändert sich die Intensität, wenn wir nur 4 Threads haben?



Ändert sich die Intensität, wenn wir nur 4 Threads haben?

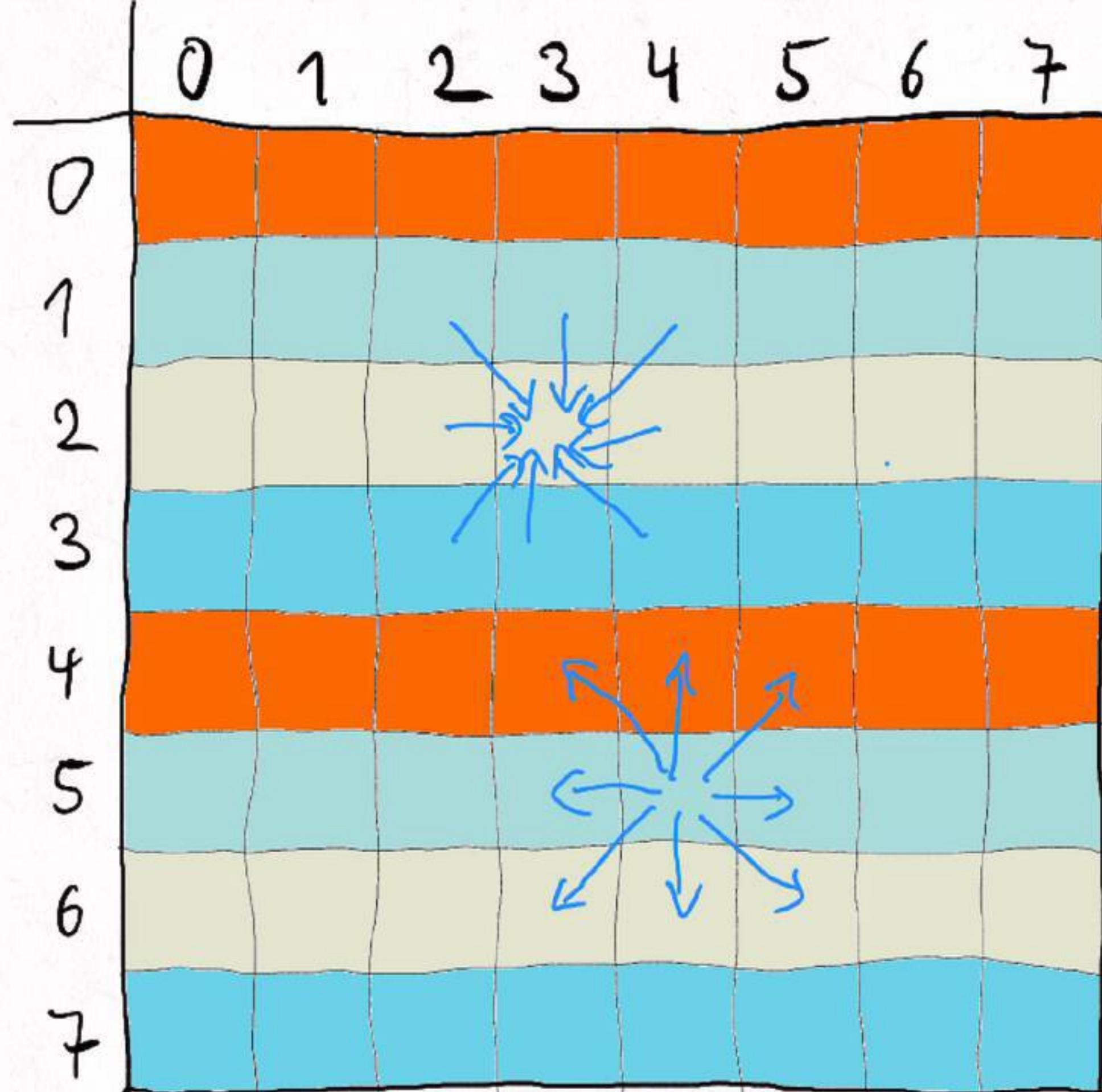
Kommt auf die Zuteilung an!



Ändert sich die Intensität, wenn wir nur 4 Threads haben?

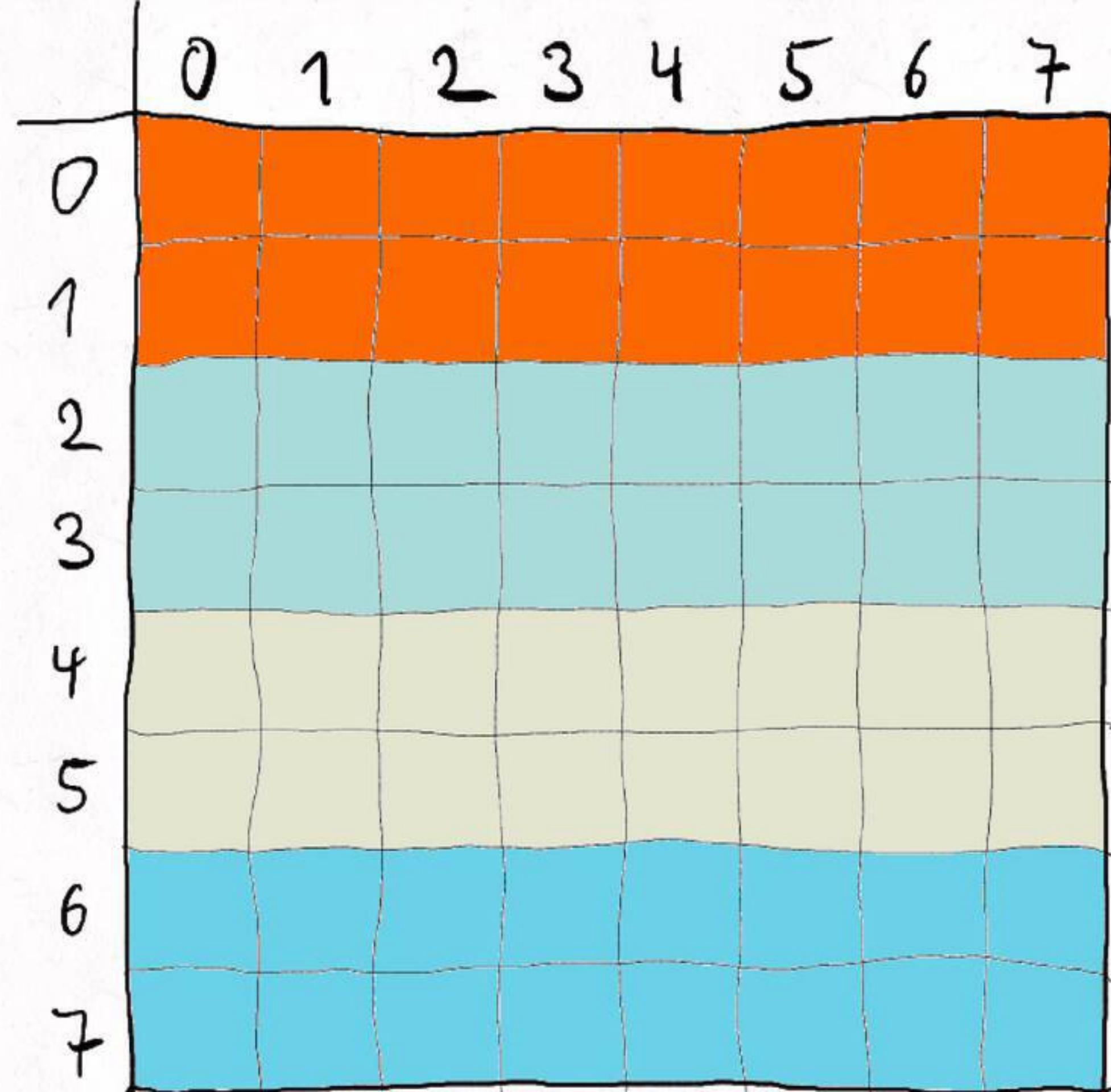
Kommt auf die Zuteilung an!

Möglichkeit 1: wie bisher: 8 Elemente zeilenweise pro Thread
Bei Zeile 4 wieder der erste Thread

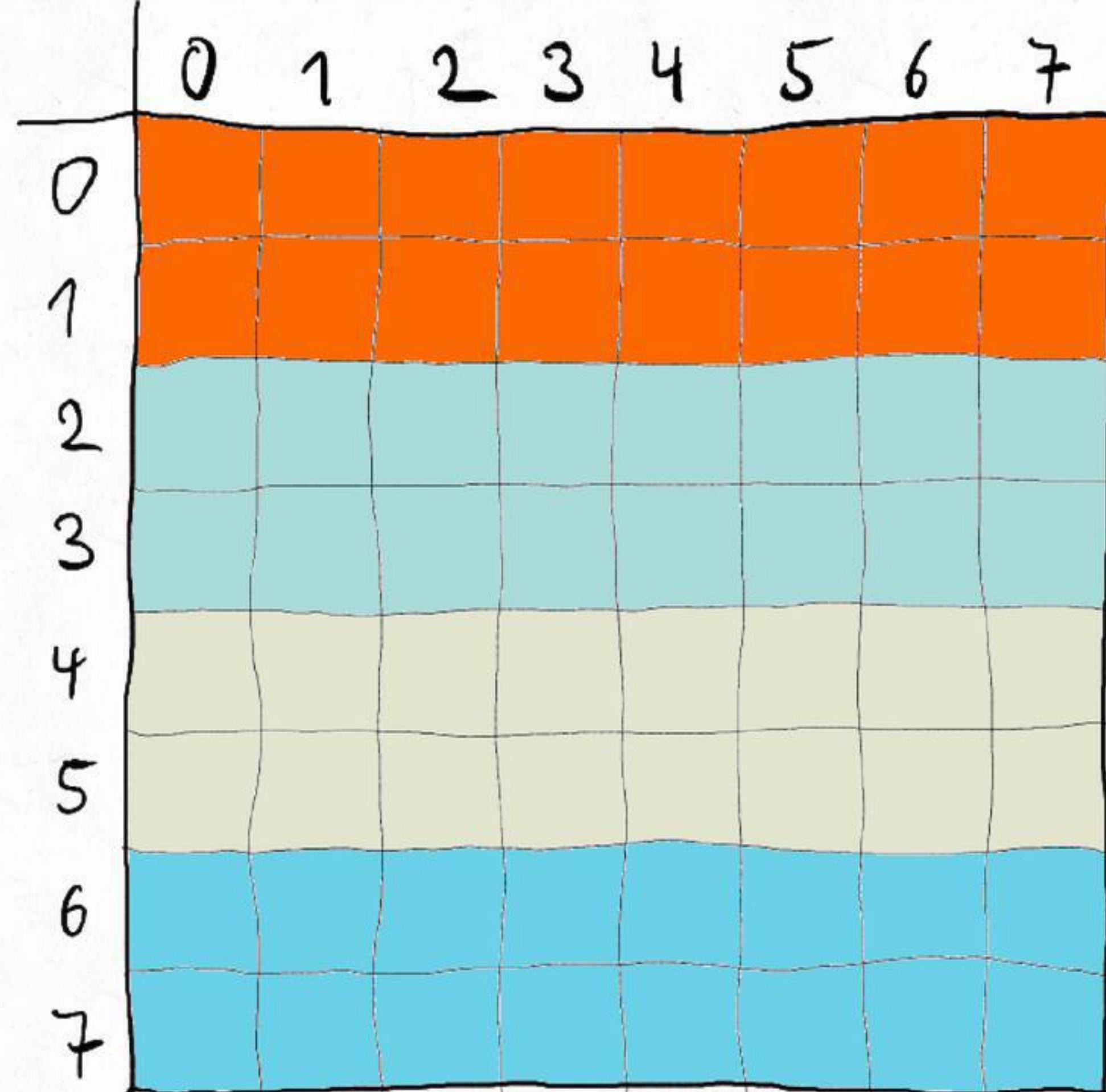


Möglichkeit 1: wie bisher: 8 Elemente zeilenweise pro Thread
Bei Zeile 4 wieder der erste Thread

Keine Veränderung
Jedes Element hängt von Daten aus 2 anderen Threads ab und jedes Element wird von anderen Threads benötigt



Möglichkeit 2:
Erste $N(64)/n(4) = 16$
Elemente an 1. Thread
usw.



Möglichkeit 2:
Erste $N(64)/n(4) = 16$
Elemente an 1. Thread
usw.
Wir betrachten ab dem
2. Durchgang, d.h. die
Elemente, die dem Thread
zugeordnet sind, sind
schon im Cache

	0	1	2	3	4	5	6	7
0	x	x	x					
1								
2	✓	✓	✓	A				
3	✓	✓	✓					
4								
5								
6								
7								

Möglichkeit 2:
Erste $N(64)/n(4) = 16$
Elemente an 1. Thread
usw.
Wir betrachten ab dem
2. Durchgang, d.h. die
Elemente, die dem Thread
zugeordnet sind, sind
schon im Cache:
A: 3

	0	1	2	3	4	5	6	7
0	x	x	x	x				
1								
2	✓	✓	A ✓	B ✓	✓			
3	✓	✓	✓		✓			
4								
5								
6								
7								

Möglichkeit 2:
Erste $N(64)/n(4) = 16$
Elemente an 1. Thread
usw.
Wir betrachten ab dem
2. Durchgang, d.h. die
Elemente, die dem Thread
zugeordnet sind, sind
schon im Cache:
A: 3 ; B: 1

	0	1	2	3	4	5	6	7
0	x	x	x	x	x			
1								
2	✓	✓	✓	✓	✓	✓		
3	A ✓	B ✓	C ✓					
4								
5								
6								
7								

Möglichkeit 2:
Erste $N(64)/n(4) = 16$
Elemente an 1. Thread
usw.
Wir betrachten ab dem
2. Durchgang, d.h. die
Elemente, die dem Thread
zugeordnet sind, sind
schon im Cache:
A: 3 ; B: 1 ; C: 1

	0	1	2	3	4	5	6	7
0	x	x	x	x	x	x		
1								
2	✓	✓	✓	✓	✓	✓	✓	
3	A ✓	B ✓	C ✓	D ✓				
4								
5								
6								
7								

Möglichkeit 2:
Erste $N(64)/n(4) = 16$
Elemente an 1. Thread
usw.
Wir betrachten ab dem
2. Durchgang, d.h. die
Elemente, die dem Thread
zugeordnet sind, sind
schon im Cache:
A: 3 ; B: 1 ; C: 1 ; D: 1

	0	1	2	3	4	5	6	7
0	x	x	x	x	x	x	x	
1								
2	✓	✓	✓	✓	✓	✓	✓	
3	A ✓	B ✓	C ✓	D ✓	E ✓			
4								
5								
6								
7								

Möglichkeit 2:
Erste $N(64)/n(4) = 16$
Elemente an 1. Thread
usw.
Wir betrachten ab dem
2. Durchgang, d.h. die
Elemente, die dem Thread
zugeordnet sind, sind
schon im Cache:
A: 3 ; B: 1 ; C: 1 ; D: 1
E: 1

	0	1	2	3	4	5	6	7
0	x	x	x	x	x	x	x	x
1								
2	✓	✓	✓	✓	✓	✓	✓	✓
3	A ✓	B ✓	C ✓	D ✓	E ✓	F ✓		✓
4								
5								
6								
7								

Möglichkeit 2:
Erste $N(64)/n(4) = 16$
Elemente an 1. Thread
usw.
Wir betrachten ab dem
2. Durchgang, d.h. die
Elemente, die dem Thread
zugeordnet sind, sind
schon im Cache:
A: 3 ; B: 1 ; C: 1 ; D: 1
E: 1 ; F: 1

	0	1	2	3	4	5	6	7
0	x	x	x	x	x	x	x	x
1								
2	✓	A ✓	B ✓	C ✓	D ✓	E ✓	F ✓	G ✓
3	✓	✓	✓	✓	✓	✓	✓	✓
4								
5								
6								
7								

Möglichkeit 2:
Erste $N(64)/n(4) = 16$
Elemente an 1. Thread
usw.
Wir betrachten ab dem
2. Durchgang, d.h. die
Elemente, die dem Thread
zugeordnet sind, sind
schon im Cache:
A: 3 ; B: 1 ; C: 1 ; D: 1
E: 1 ; F: 1 ; G: 0

	0	1	2	3	4	5	6	7
0	x	x	x	x	x	x	x	x
1	v	v	v	v	v	v	v	v
2	H v v	A v v	B v v	C v v	D v v	E v v	F v v	G v v
3	v v							
4								
5								
6								
7								

Möglichkeit 2:
Erste $N(64)/n(4) = 16$
Elemente an 1. Thread
usw.
Wir betrachten ab dem
2. Durchgang, d.h. die
Elemente, die dem Thread
zugeordnet sind, sind
schon im Cache:
A: 3 ; B: 1 ; C: 1 ; D: 1
E: 1 ; F: 1 ; G: 0 ; H: 0

	0	1	2	3	4	5	6	7
0	x	x	x	x	x	x	x	x
1	v	v	v	v	v	v	v	v
2	H v v	A v v	B v v	C v v	D v v	E v v	F v v	G v v
3	v v v	I v v	v v v	v v v	v v v	v v v	v v v	v v v
4	x	x	x					
5								
6								
7								

Möglichkeit 2:
Erste $N(64)/n(4) = 16$
Elemente an 1. Thread
usw.
Wir betrachten ab dem
2. Durchgang, d.h. die
Elemente, die dem Thread
zugeordnet sind, sind
schon im Cache:
A: 3 ; B: 1 ; C: 1 ; D: 1
E: 1 ; F: 1 ; G: 0 ; H: 0
I: 3

	0	1	2	3	4	5	6	7
0	x	x	x	x	x	x	x	x
1	v	v	v	v	v	v	v	v
2	H v v	A w v	B w v	C v v	D v v	E v v	F v v	G v v
3	w v v	I v v	J v v	v v v	v v v	v v v	v v v	v v v
4	x	x	x	x				
5								
6								
7								

Möglichkeit 2:
Erste $N(64)/n(4) = 16$
Elemente an 1. Thread
usw.
Wir betrachten ab dem
2. Durchgang, d.h. die
Elemente, die dem Thread
zugeordnet sind, sind
schon im Cache:
A: 3 ; B: 1 ; C: 1 ; D: 1
E: 1 ; F: 1 ; G: 0 ; H: 0
I: 3 ; J: 1

	0	1	2	3	4	5	6	7
0	x	x	x	x	x	x	x	x
1	v	v	v	v	v	v	v	v
2	H v v	A w v	B n v	C v v	D v v	E v v	F v v	G v v
3	P n v	I v v	J v v	K v v	L v v	M v v	N v v	O v v
4	x	x	x	x				
5								
6								
7								

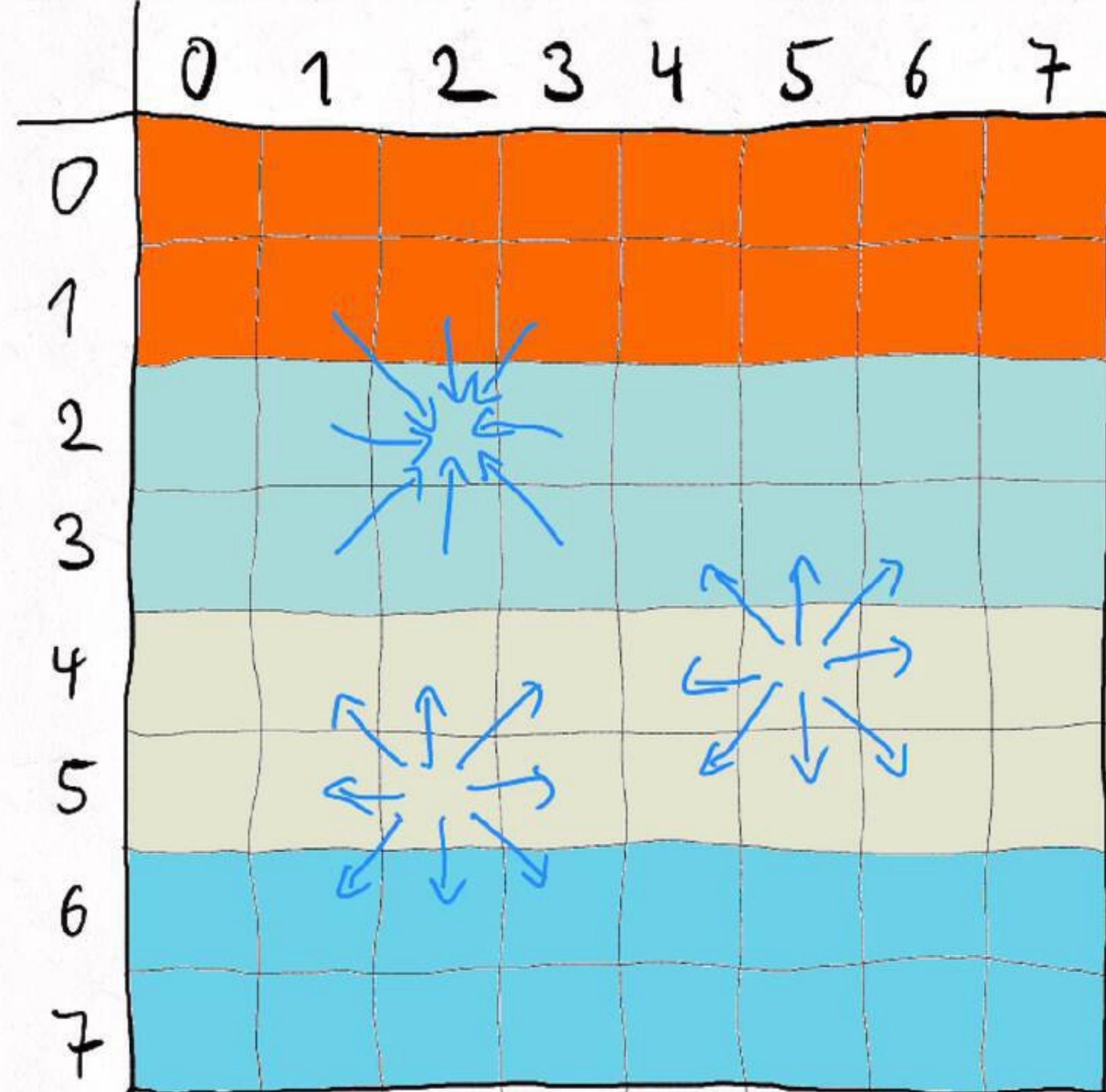
Möglichkeit 2:
Erste $N(64)/n(4) = 16$
Elemente an 1. Thread
usw.
Wir betrachten ab dem
2. Durchgang, d.h. die
Elemente, die dem Thread
zugeordnet sind, sind
schon im Cache:
A: 3 ; B: 1 ; C: 1 ; D: 1
E: 1 ; F: 1 ; G: 0 ; H: 0
I: 3 ; J: 1 ; K: 1 ; L: 1
M: 1 ; N: 1 ; O: 0 ; P: 0
 $\Sigma: 16$ Reads

	0	1	2	3	4	5	6	7
0	x	x	x	x	x	x	x	x
1	v	v	v	v	v	v	v	v
2	H v v	A v w	B v w	C v v	D v v	E v v	F v v	G v v
3	P v w	I v v	J v v	K v v	L v v	M v v	N v v	O v v
4	x	x	x	x				
5								
6								
7								

Möglichkeit 2:
Erste $N(64)/n(4) = 16$
Elemente auf 1. Thread
usw.
Wir betrachten ab dem
2. Durchgang
Wieder 16 Reads
pro Durchgang
und Thread?

	0	1	2	3	4	5	6	7
0	x	x	x	x	x	x	x	x
1	v	v	v	v	v	v	v	v
2	H v v	A v w	B v w	C v v	D v v	E v v	F v v	G v v
3	P v w	I v v	J v v	K v v	L v v	M v v	N v v	O v v
4	x	x	x	x				
5								
6								
7								

Möglichkeit 2:
Erste $N(64)/n(4) = 16$
Elemente an 1. Thread
usw.
Wir betrachten ab dem
2. Durchgang
Wieder 16 Reads
pro Durchgang
und Thread?
Ja, aber nur noch
4 Threads ...
... also $4 \times 16 = 64$ statt 128!



Möglichkeit 2:
Erste $N(64)/n(4) = 16$
Elemente an 1 Thread
usw.

wir betrachten ab dem
2. Durchgang,

- Jedes Element braucht
nur noch Daten aus 1
anderen Thread j
- Jedes Element wird
immer noch von 1 anderen
Thread gebraucht ...
... immer noch 64 Writes

Wie computer-intensive ist Möglichkeit 2?

$4 \times 16 = 64$ Reads pro Durchgang

+ $8 \times 8 = 64$ Writes pro Durchgang

jeder Wert wird von einem anderen Thread
ausgelesen - muss also in den Speicher

$\overline{128} \times 4$ Bytes (Single Prec. Float) = 512 Bytes

Floating Point Ops: 64 Felder $\times (1$ Multiplikation + 8 Additionen) = 576

Intensität: $576 / 512 = 1,125$ (+50% zu Möglichkeit 1)

	0	1	2	3	4	5	6	7
0								

Gibt es eine
Aufteilung
auf 4 Threads
mit noch höherer
Indensität als
Möglichkeit 2?

	0	1	2	3	4	5	6	7
0								

Gibt es eine
Aufteilung
auf 4 Threads
mit noch höherer
Indensität als
Möglichkeit 2?

Ja!

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

Gibt es eine
Aufteilung
auf 4 Threads
mit noch höherer
Indensität als
Möglichkeit 2?

Ja!

Ideen?

