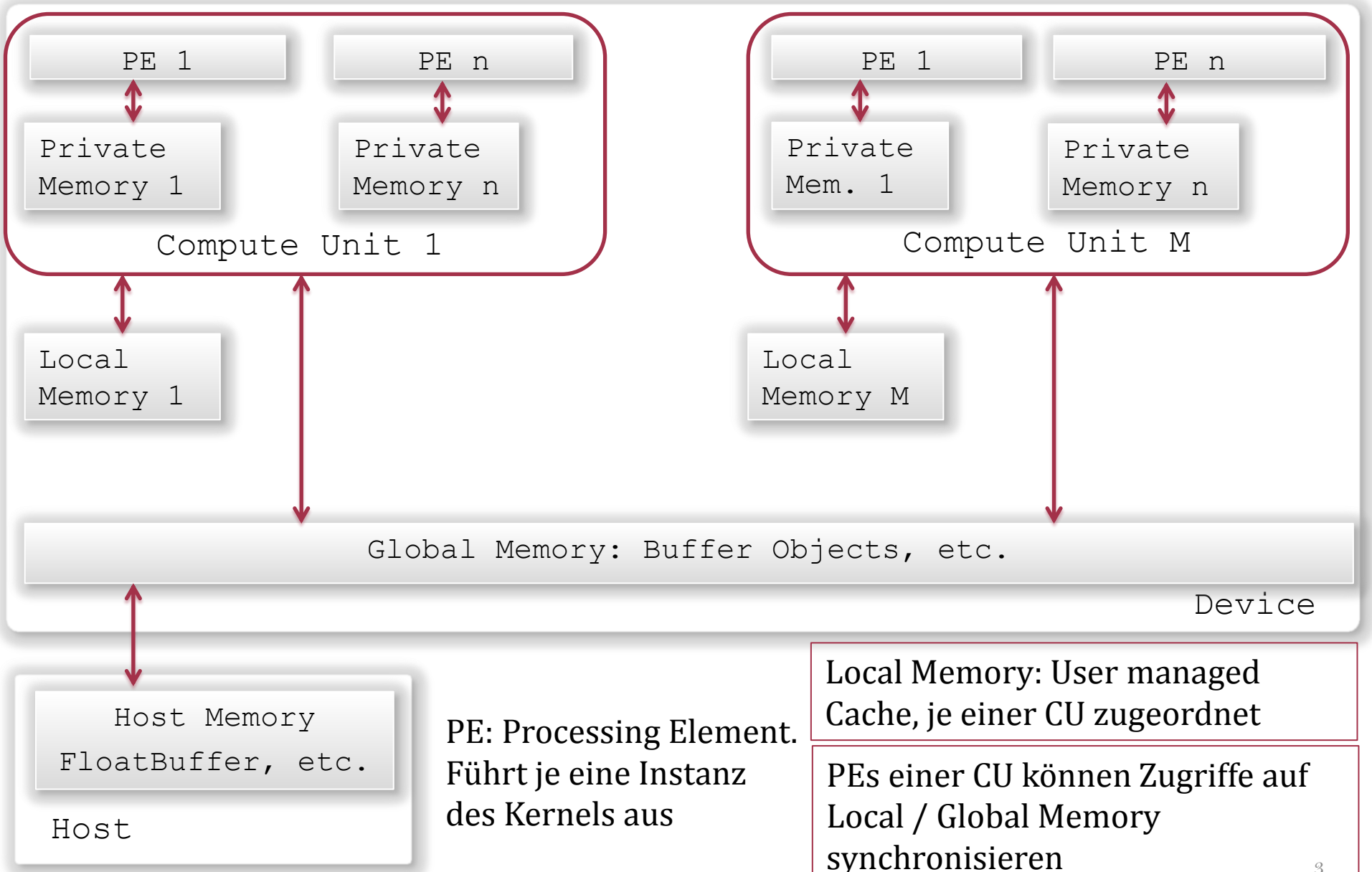
Computergrafik

Universität Osnabrück, Henning Wenke, 2012-06-26

Noch Kapitel XV:

Parallele Algorithmen mit OpenCL

OpenCL Speichermodell



PE: Processing Element. Führt je eine Instanz des Kernels aus

Local Memory: User managed Cache, je einer CU zugeordnet

PEs einer CU können Zugriffe auf Local / Global Memory synchronisieren

15.6

OpenCL C Fragen

Was passiert hier? (I)

```
kernel void weirdKernel_1(
  global float4* b,
  local  float4* c) {

  c[get_local_id(0)] = b[get_global_id(0)];

}
```

- Nichts, da keine globalen Daten geschrieben werden
- Lokaler Speicher ist temporär und überlebt die Kernellaufzeit nicht

Was passiert hier? (II)

```
kernel void weirdKernel_2(
  global float* a,
  global float* b) {

  int index = get_local_id(0);

  float tmp = a[index];
  a[index]  = b[index];
  b[index]  = tmp;
}
```

- Synchronisationsproblem, bei mehr als einer Work Group
- Lokaler Index nur per Workgroup eindeutig
- Welche Werte überhaupt angefasst?
  - Die ersten Local Size Werte
- Wie oft?
  - Group Count mal, potentiell gleichzeitig
- Richtig wäre tauschen der Werte von a und b mit:
  - `int index = get_global_id(0);`

Was passiert hier? (III)

```
kernel void weirdKernel_3(
  global float* a,
  global float* b) {

  // Nur diese Zeile ist neu:
  int index = get_group_id(0) * get_local_size(0) + get_local_id(0);

  float tmp = a[index];
  a[index]  = b[index];
  b[index]  = tmp;
}
```

- Tauschen der Werte von a und b
- Index ist eindeutig und identisch mit `get_global_id(0);`

Was passiert hier? (IV)

```
kernel void weirdKernel_4(
  global float* a,
  global float* b) {

  int index = get_global_id(0);

  if(index % 2 == 0)
    b[index + 1] = a[index];
  else
    b[index - 1] = a[index];
}
```

- Schreibe Werte aus a paarweise vertauscht in b

Was passiert hier? (V)

```
// Wie eben, aber nur eine globale Variable
kernel void weirdKernel_5(
  global float* a) {
  int index = get_global_id(0);

  float tmp = a[index];

  if(index % 2 == 0)
    a[index + 1] = tmp;
  else
    a[index - 1] = tmp;
}
```

- Potentieller Konflikt: Zu lesender Wert möglicherweise bereits durch Nachbarn überschrieben

Was passiert hier? (VI)

```
// Wichtig: Sei Local Size % 2 Null
kernel void weirdKernel_6(
  global float* a) {
  int index = get_global_id(0);
  float tmp = a[index];

  // Nur diese Zeile neu
  barrier(CLK_GLOBAL_MEM_FENCE);

  if(index % 2 == 0)
    a[index + 1] = tmp;
  else
    a[index - 1] = tmp;
}
```

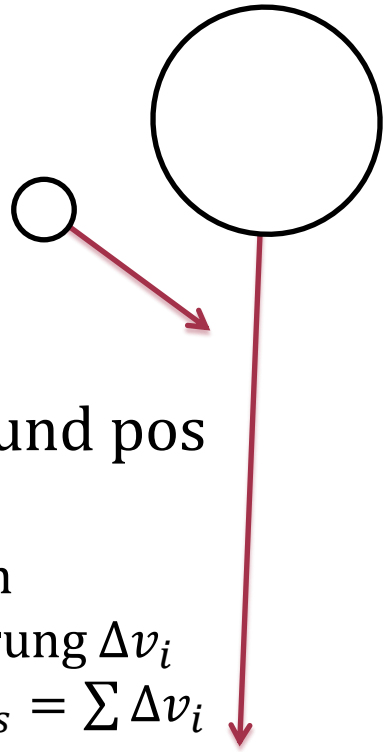
- Da Local Size geradzahlig, sind alle Vertauschungen innerhalb je einer Work Group
- Deshalb Synchronisierung möglich
- Also werden die Elemente von a korrekt paarweise vertauscht

15.7

Unser Partikelsystem

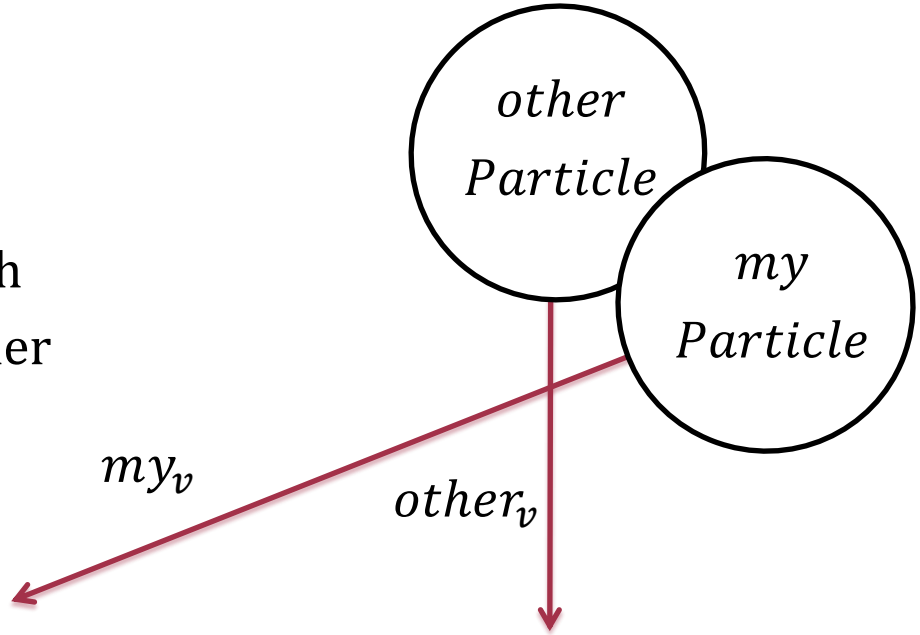
Ansatz

- Orientiert sich an “Fast N-Body Simulation with CUDA“ aus GPU Gems III
- Gegeben: N Partikel, jeweils mit:
  - Radius (heute: bei allen gleich)
  - Masse (heute: bei allen gleich)
  - Aktuelle Geschwindigkeit v
  - Aktuelle Position pos
- Jedes Partikel bekommt zufällige Startwerte für v und pos
- In jedem Zeitschritt für alle Partikel:
  - Überprüfe mit jedem anderen Partikel i auf eine Kollision
  - Bestimme ggf. dadurch bewirkte Geschwindigkeitsänderung Δv_i
  - Berechne resultierende Geschwindigkeitsänderung $\Delta v_{res} = \sum \Delta v_i$
  - Bestimme neue Geschwindigkeit $v_{new} = v_{old} + \Delta v_{res}$
  - Bewege Partikel mit neuer Geschwindigkeit einen Zeitschritt: $pos_{new} = pos_{old} + dt \cdot v_{new}$
- Brute Force Ansatz: $O(n^2)$



Kollision zweier Partikel

- Berechne Auswirkung auf ein Partikel “myParticle“ durch je ein anderes “otherParticle“
- Führe Berechnungen unabhängig durch
- Auswirkung 0, falls Distanz > Summe der Radien
- Methode “collide“ (mehr auf nächsten Folien)

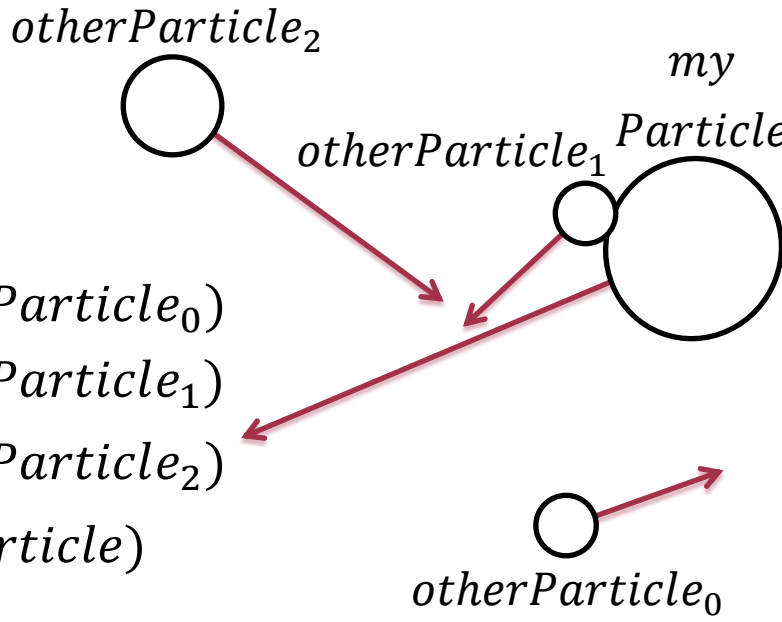



```
float4 collide( float4 myPos, float4 otherPos,
                float4 myV,   float4 otherV,
                [weiteres: Radien, Massen,…]
               ){…};
```

Geschwindigkeitsänderung des Partikels “myParticle“ bewirkt durch “otherPartikel“

Geschwindigkeitsänderung eines Partikels

- Gesamtgeschwindigkeitsänderung eines Partikels ergibt sich durch Vergleich mit allen anderen Partikeln



$$\Delta v_{res} = 0 \quad collide(myParticle, otherParticle_0)$$
$$+ \Delta v_1 \quad collide(myParticle, otherParticle_1)$$
$$+ 0 \quad collide(myParticle, otherParticle_2)$$
$$+ 0 \quad collide(myParticle, myParticle)$$

```
float4 collide( float4 myPos, float4 otherPos,
                float4 myV,   float4 otherV,
                [weiteres: Radien, Massen,…]
              ) {…};
```

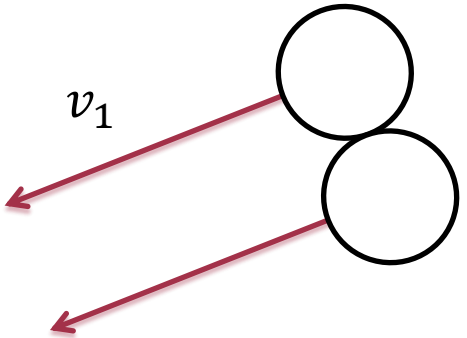
Geschwindigkeitsänderung des Partikels “myParticle“ bewirkt durch “otherPartikel“

Gesammt Δv des Partikels “myParticle“: $\sum_{particles\ i} collide(myParticle, otherParticle_i)$

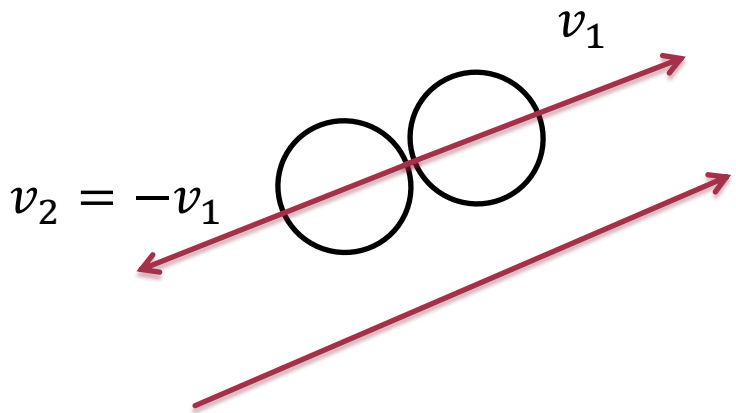
Relative Geschwindigkeit

- Parallel bewegte Partikel:
  - „Kollision“ bewirkt nichts
- Entgegengesetzt bewegte Partikel:
  - Maximale Wirkung
- Entscheidend: Relative Geschwindigkeit:
  - $v_{relative} = -v_2 + v_1$

$v_{relative} = -v_2 + v_1 = 0$



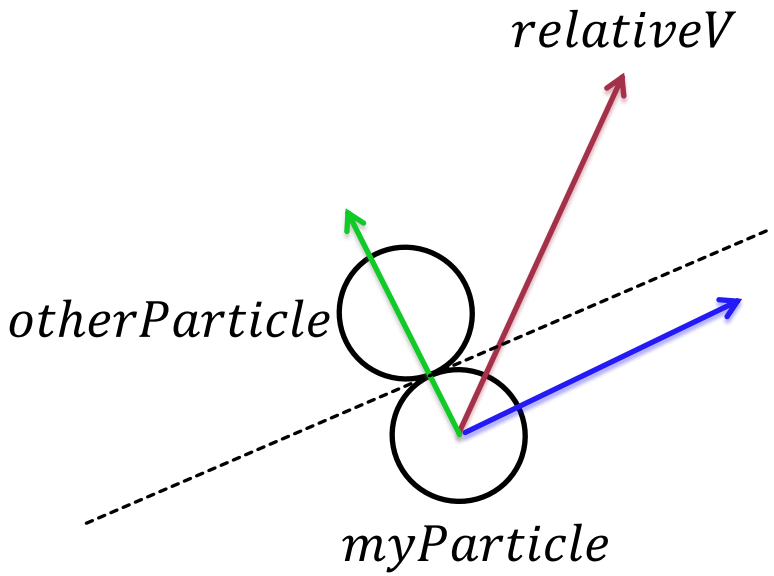
$v_2 = v_1$



$v_{relative} = -v_2 + v_1 = 2v_1$

∥ & ⊥ Anteil der relativen Geschw.

- Vorstellung:
  - otherPartikel bewegt sich mit relativer Geschwindigkeit
  - myPartikel steht still
- Zerlege relative Geschwindigkeit in parallele und senkrechte Komponente zur Tangentialebene
- Verwende beide unterschiedlich zur Kollisionsberechnung
- Beispiel:
  - Ignoriere tangentiale Komponente
  - Stoße in senkrechter Richtung elastisch oder inelastisch
- Spielt da ein wenig mit herum!
- Weitere Tipps: Übung



$\longrightarrow$ $relativeV_{direct}$: $\perp$ *Komponente der relativen Gechwindigkeit*

$\longrightarrow$ $relativeV_{tan}$: $\parallel$ *Komponte der relativen Gechwindigkeit*

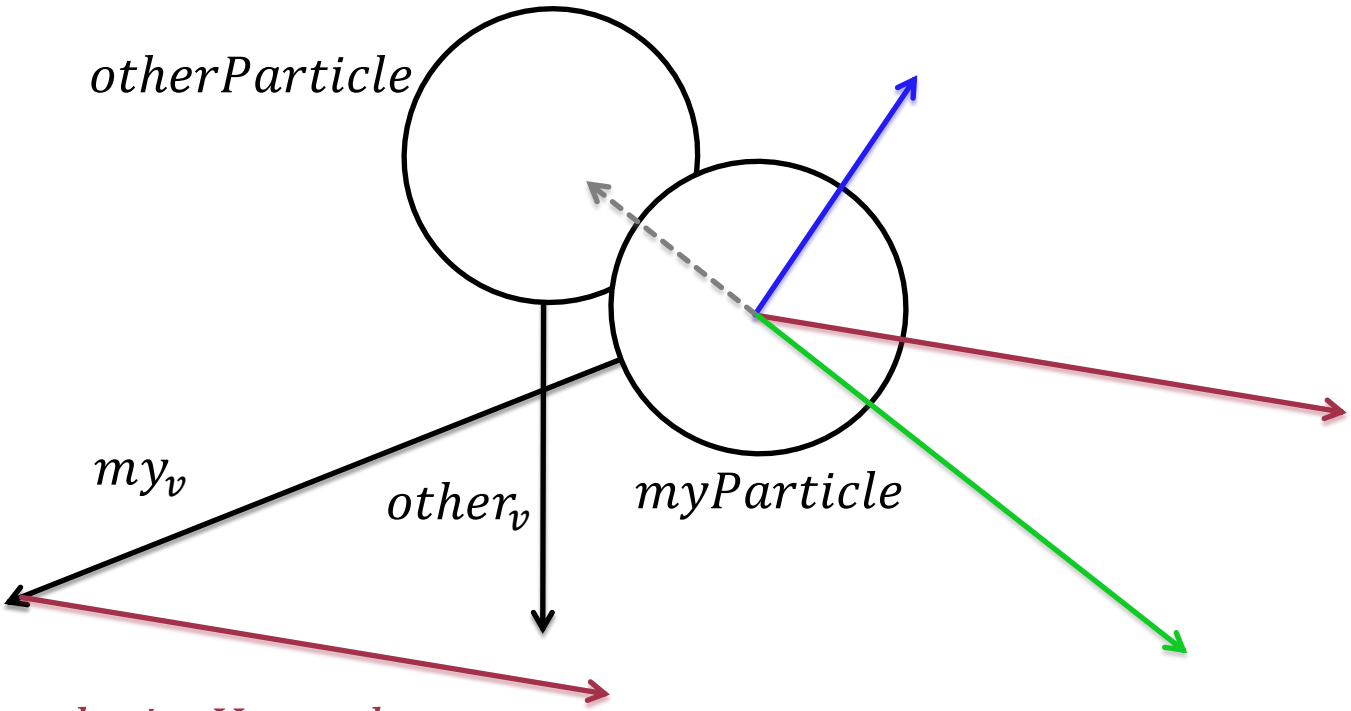
Ber. des ∥ und ⊥ Anteils von relativeV

$my_{pos}other_{pos} = normalize(other_{pos} - my_{pos})$

Betrag von relativeV in senkrechter Richtung

$d = dot(relative_v, my_{pos}other_{pos})$

$Hier: d \approx -2{,}5$



$d = 0$

$d = length(relative_v)$

$d = -length(relative_v)$

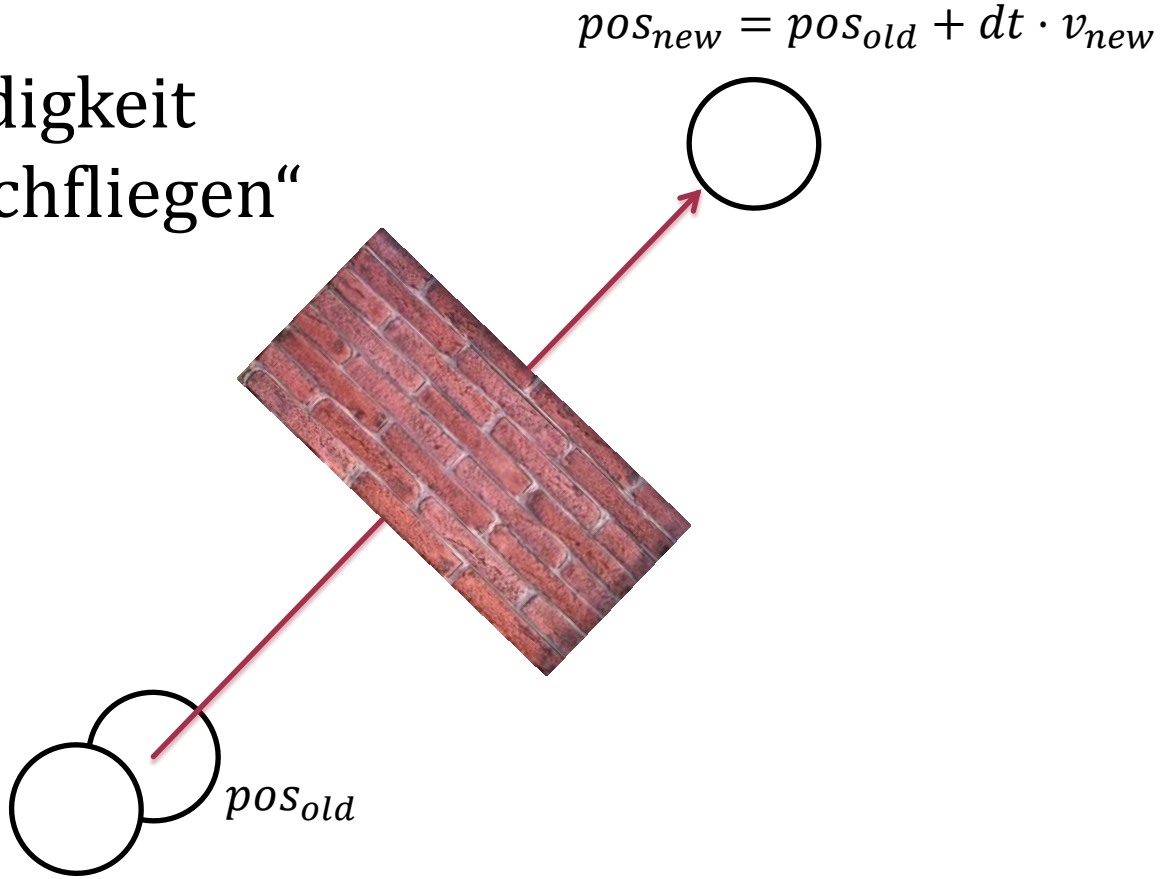
$relativeV = other_v - my_v$

$relativeV_{direct} = d \cdot my_{pos}other_{pos}$

$relativeV_{tan} = realtive_v - relativeV_{direct}$

Ein Effekt

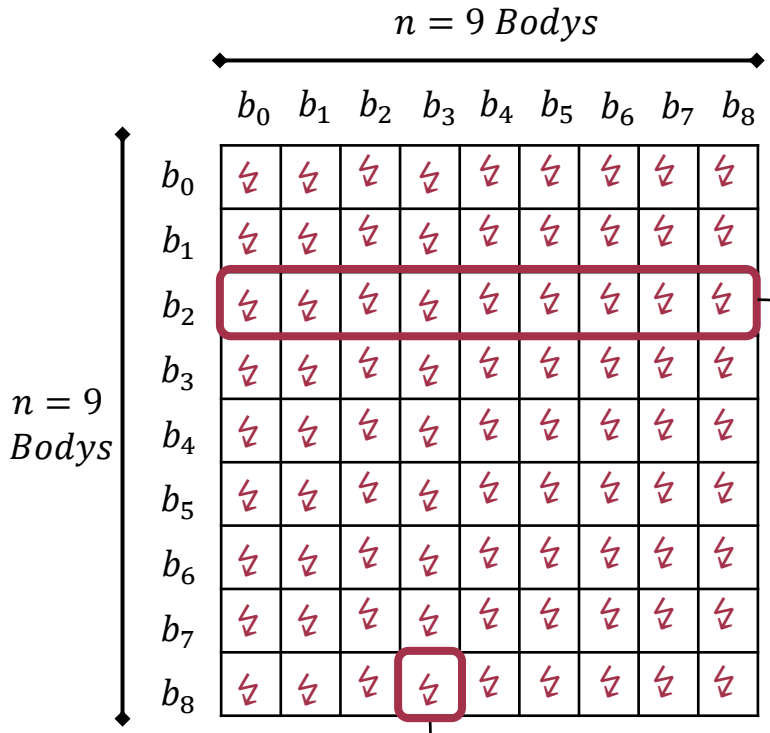
- Partikel werden zu diskreten Zeitschritten aktualisiert
- Hohe Geschwindigkeit ermöglicht “durchfliegen“ anderer Partikel



15.8

Implementation mit OpenCL

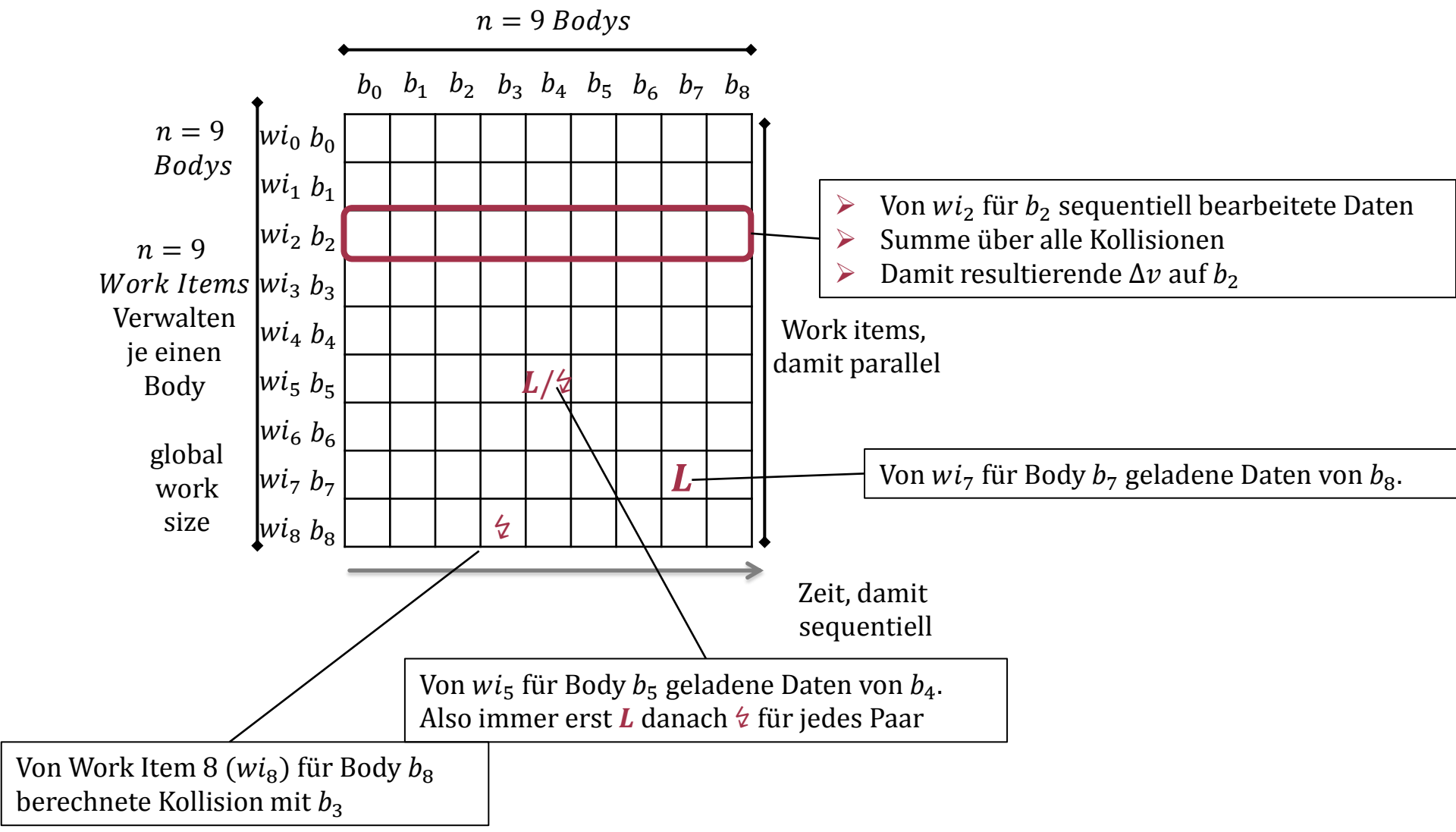
Schema für Berechnungen



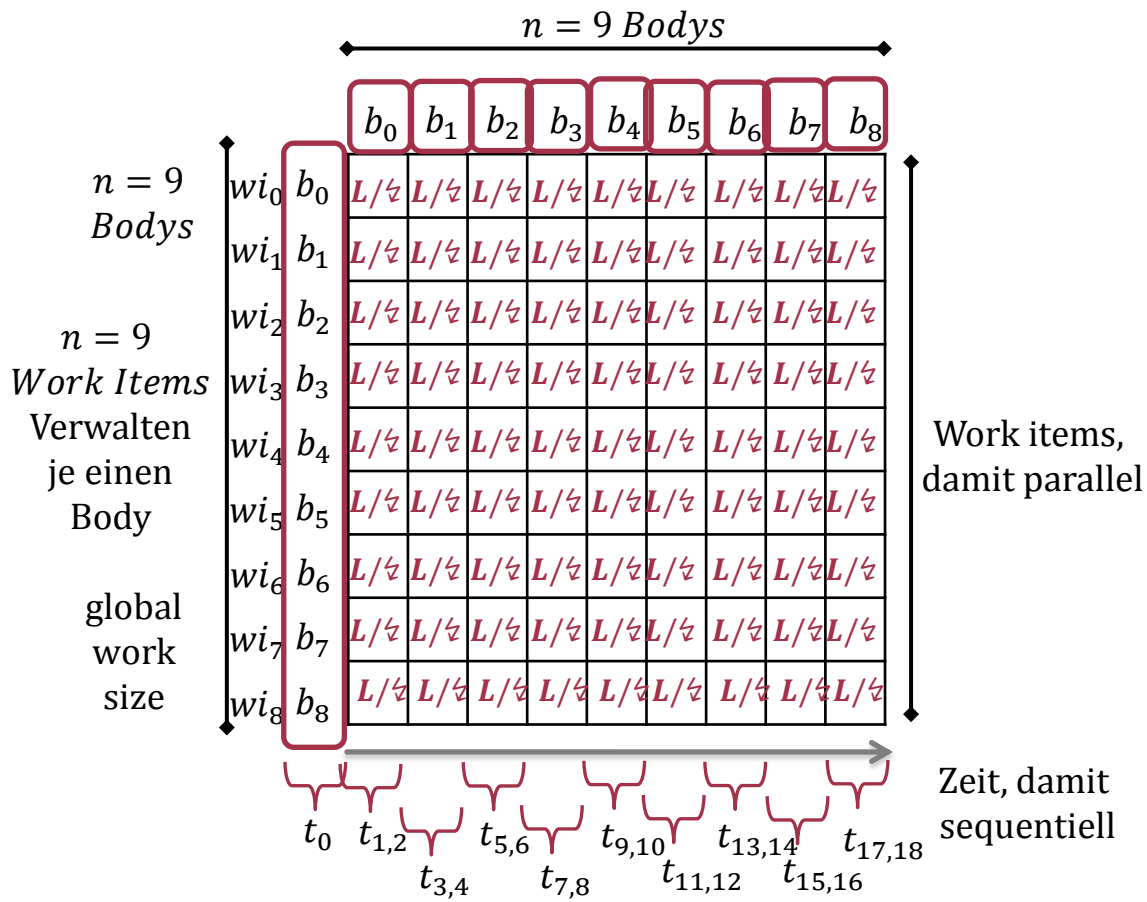
- Summe über alle Δv (Spalten) einer Zeile
- Gibt Δv_{res} auf Body dieser Zeile an
- Hier: Δv_{res} von b_2

- Kollisionen, z.B. von b_8 mit b_3 ↯
- Liefert hier Δv für b_8 bewirkt durch b_3
- Können alle unabhängig voneinander berechnet werden

Implementation



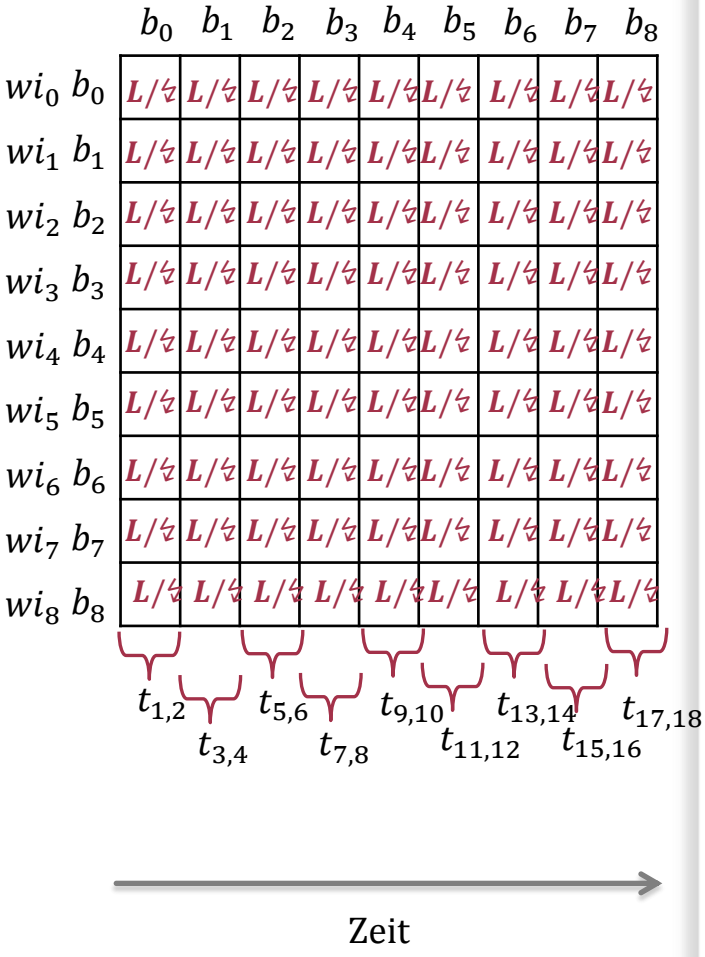
Naive Implementation I



Aktuell geladene Daten

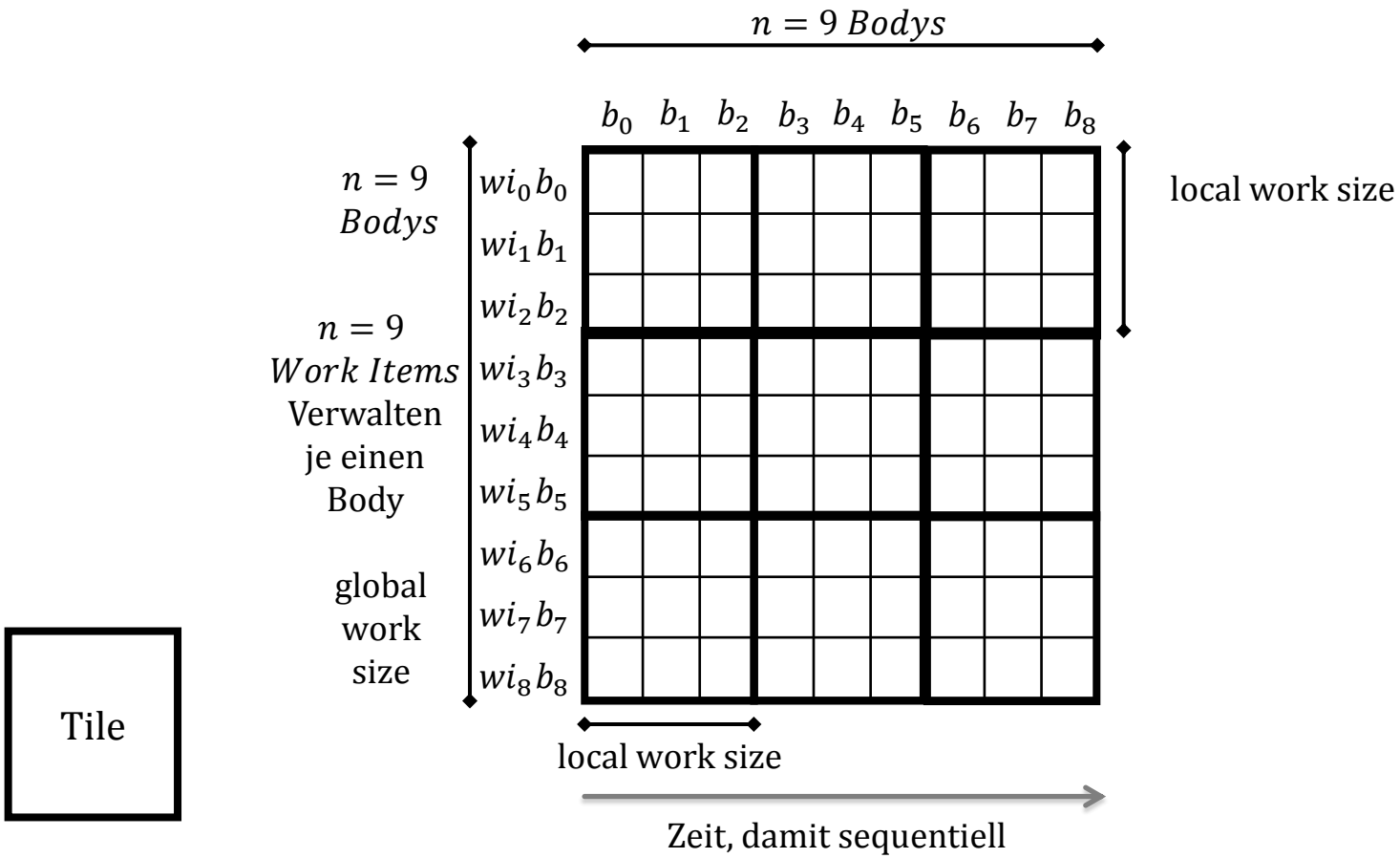
L/↯ Lade erst anderen Body zum Vergleich, dann werte Methode `collide` aus

Naive Implementation II



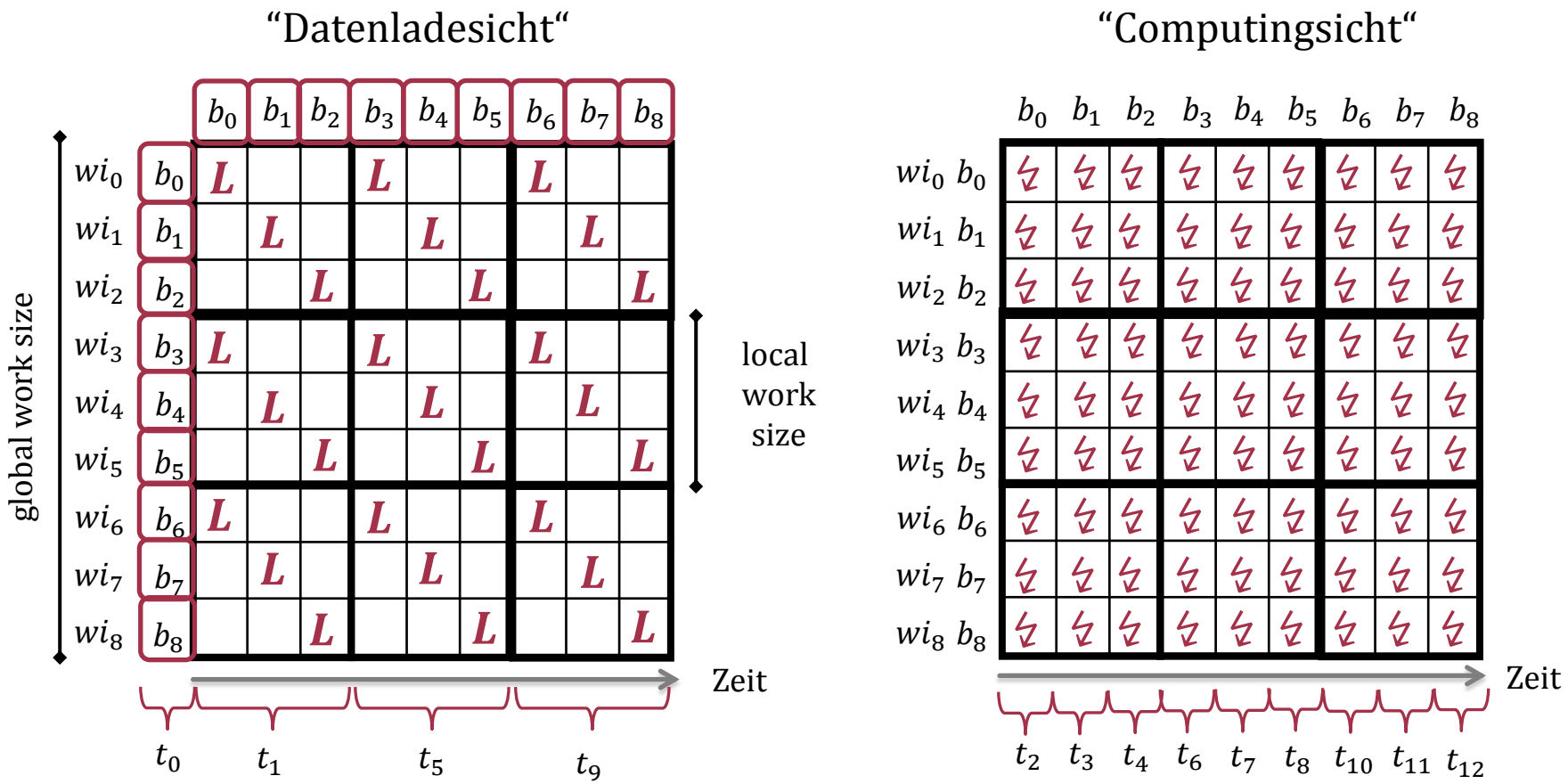

```
kernel void asteroid_sim_naive(
 global float4* posOld,
 global float4* posNew,
 global float4* vOld,
 global float4* vNew,
 float dt
) {
 uint   myId  = get_global_id(0);
 float4 myPos = posOld[myId];
 float4 myV   = vOld  [myId];
 float4 dVRes = (float4)(0,0,0,0);
 uint bodyCnt = get_global_size(0);
 for(uint i = 0; i < bodyCnt; i++) {
   float4 otherPos = posOld[i];
   float4 otherV   = vOld[i];
    dVRes += collide(myPos, myV, otherPos, otherV);
 }
 <Orbitalkorrektur>
 vNew[myId] = myV + dVRes;
 posNew[myId] = myPos + vNew[myId] * dt;
}
```

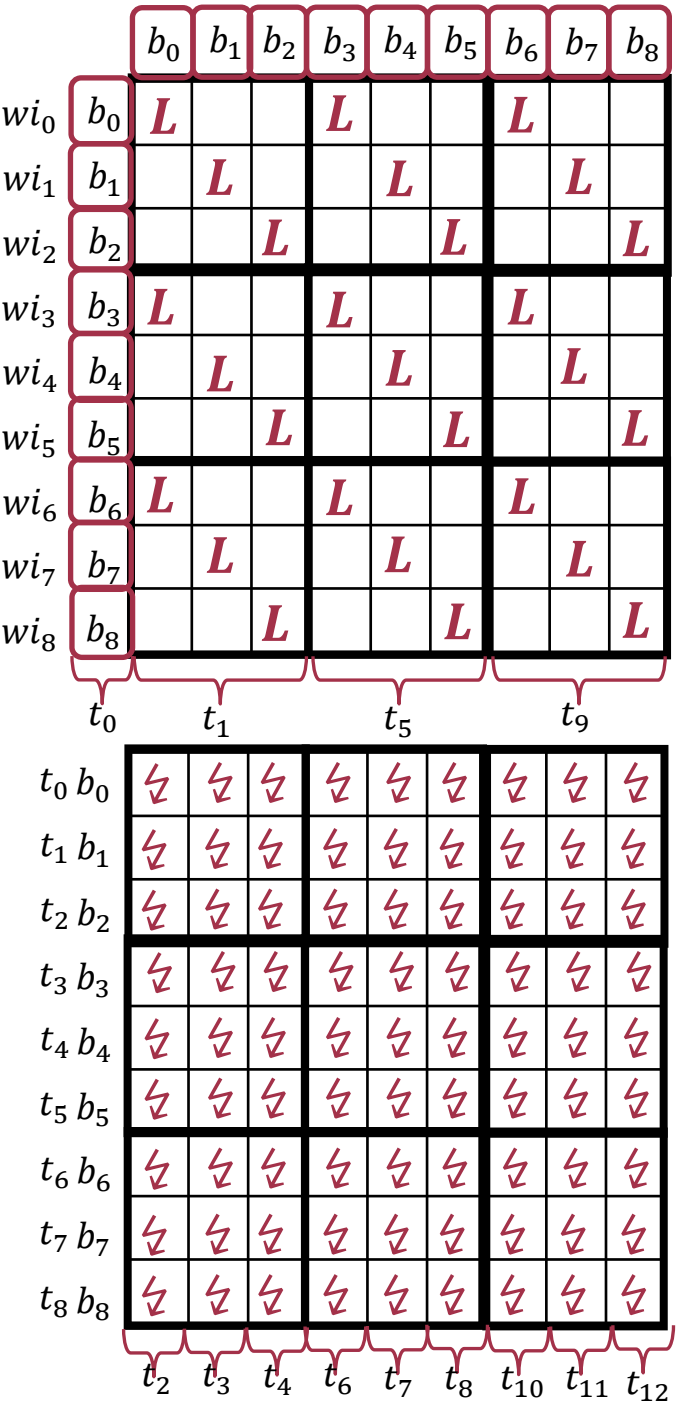
Ansatz: Cache Daten lokal I



- Von Tile überlappte Berechnungen: Führe in einer Work Group aus
  - Anzahl Berechnungen je WG: $LocalWorkSize^2$
- Lade benötigte Daten nur einmal pro Work Group
  - Menge geladener Daten je WG: $LocalWorkSize$

Ansatz: Cache Daten lokal I






```
kernel void asteroid_sim(
global float4* posOld, global float4* posNew,
global float4* vOld, global float4* vNew,
local  float4* otherPositionsInTile,
float dt) {
  int    myId  = get_global_id(0);
  float4 myPos = posOld[myId];
  float4 myV   = vOld  [myId];
  float4 dVRes   = 0;
  int localid   = get_local_id(0);
  int tileSize  = get_local_size(0);
  int tileCnt   = get_num_groups(0);
  for(int tile = 0; tile < tileCnt; ++tile)  {
    otherPositionsInTile[localid]
      = posOld[tileSize * tile + localid];
    barrier(CLK_LOCAL_MEM_FENCE);
    for(int j = 0; j < tileSize; ++j) {
      float4 otherPos = otherPositionsInTile[j];
      float4 otherV = vOld[tileSize * tile + j];
      dVRes += collide(myPos, myV, otherPos , otherV);
    }
    barrier(CLK_LOCAL_MEM_FENCE);
  }
 <Orbitalkorrektur>
 vNew[myId] = myV + dVRes;
 posNew[myId] = myPos + vNew[myId] * dt;
}
```

Grober Ablauf

- 4 Buffer: pos1, v1, pos2, v2
- In pos1 und v1 befinden sich Startwerte
- Rufe Kernel für alle Bodys auf mit:
  - Pos1 angebunden posOld
  - V1 angebunden an vOld
  - Pos2 angebunden an posNew
  - v2 angebunden an vNew
- Rendere pos2, v2
- Rufe Kernel für alle Bodys auf mit:
  - Pos2 angebunden posOld
  - V2 angebunden an vOld
  - Pos1 angebunden an posNew
  - v1 angebunden an vNew
- Rendere pos1, v1
- …