

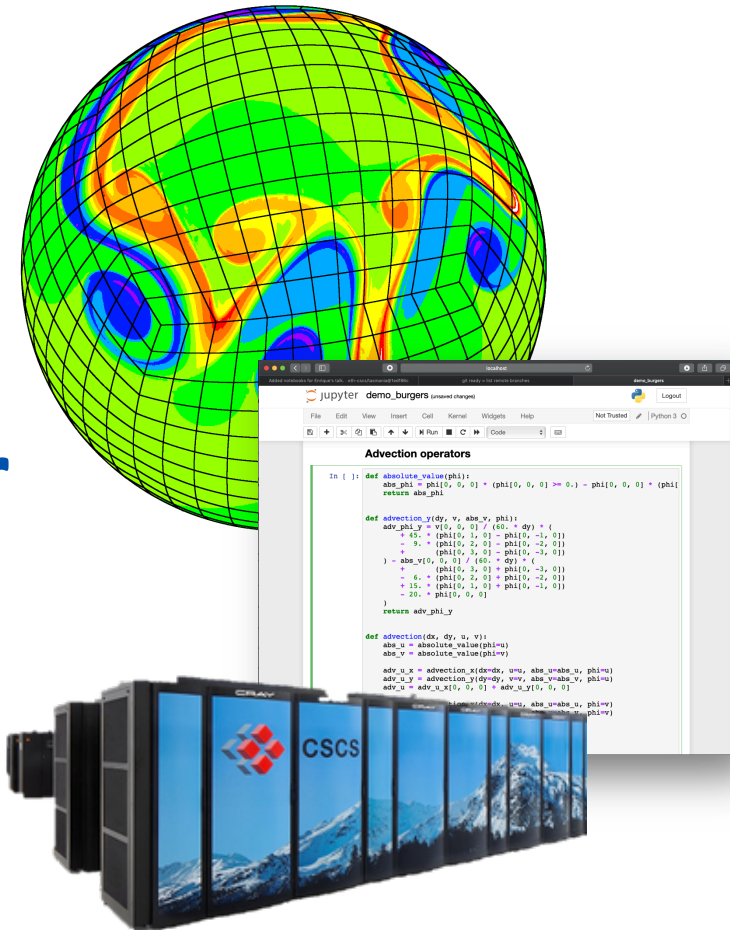
High Performance Computing for Weather and Climate (HPC4WC)

Content: Caches and Data Locality

Lecturers: Oliver Fuhrer, Tobias Wicky

Block course 701-1270-00L

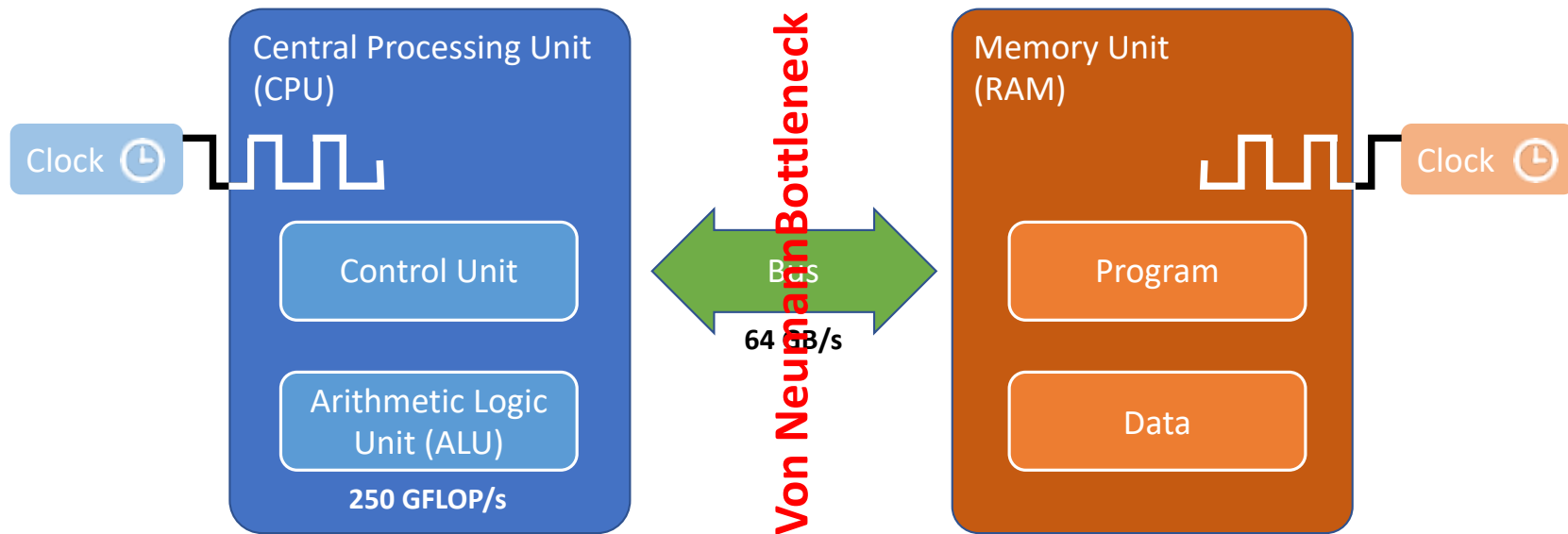
Summer 2020



Learning goals

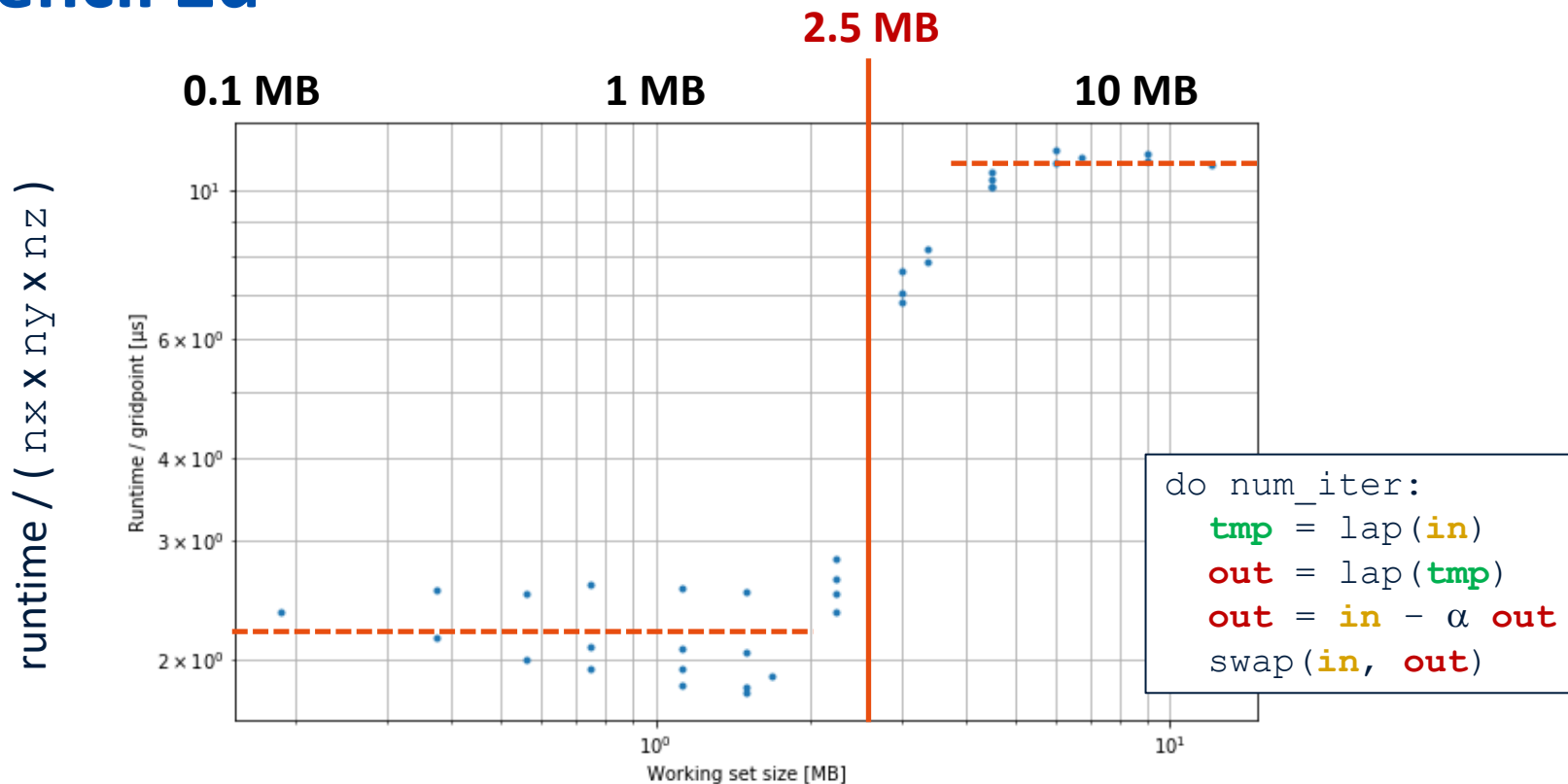
- Understand how data arrays are stored in computer memory
- Understand the implications of the cache hierarchy in a modern multi-core CPU
- Able to do basic data-locality optimizations (fusion, inlining) to improve performance

Von Neumann Architecture



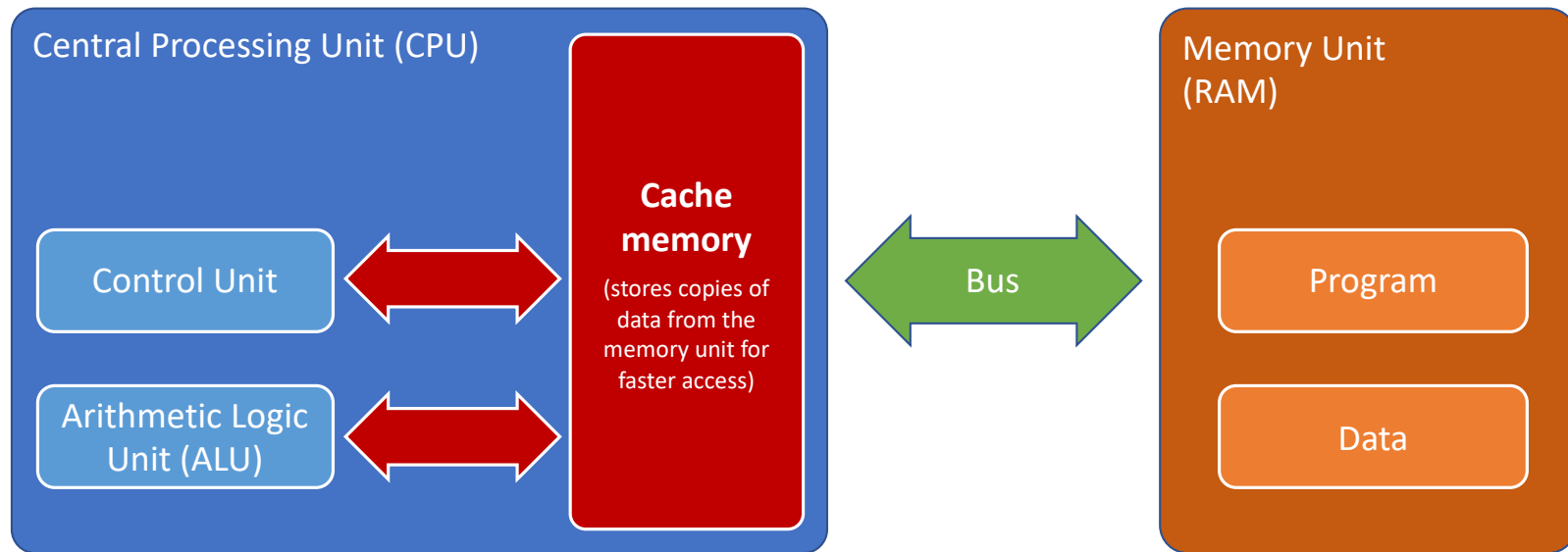
32 floating-point operations per load/store of a data value!

Stencil 2d



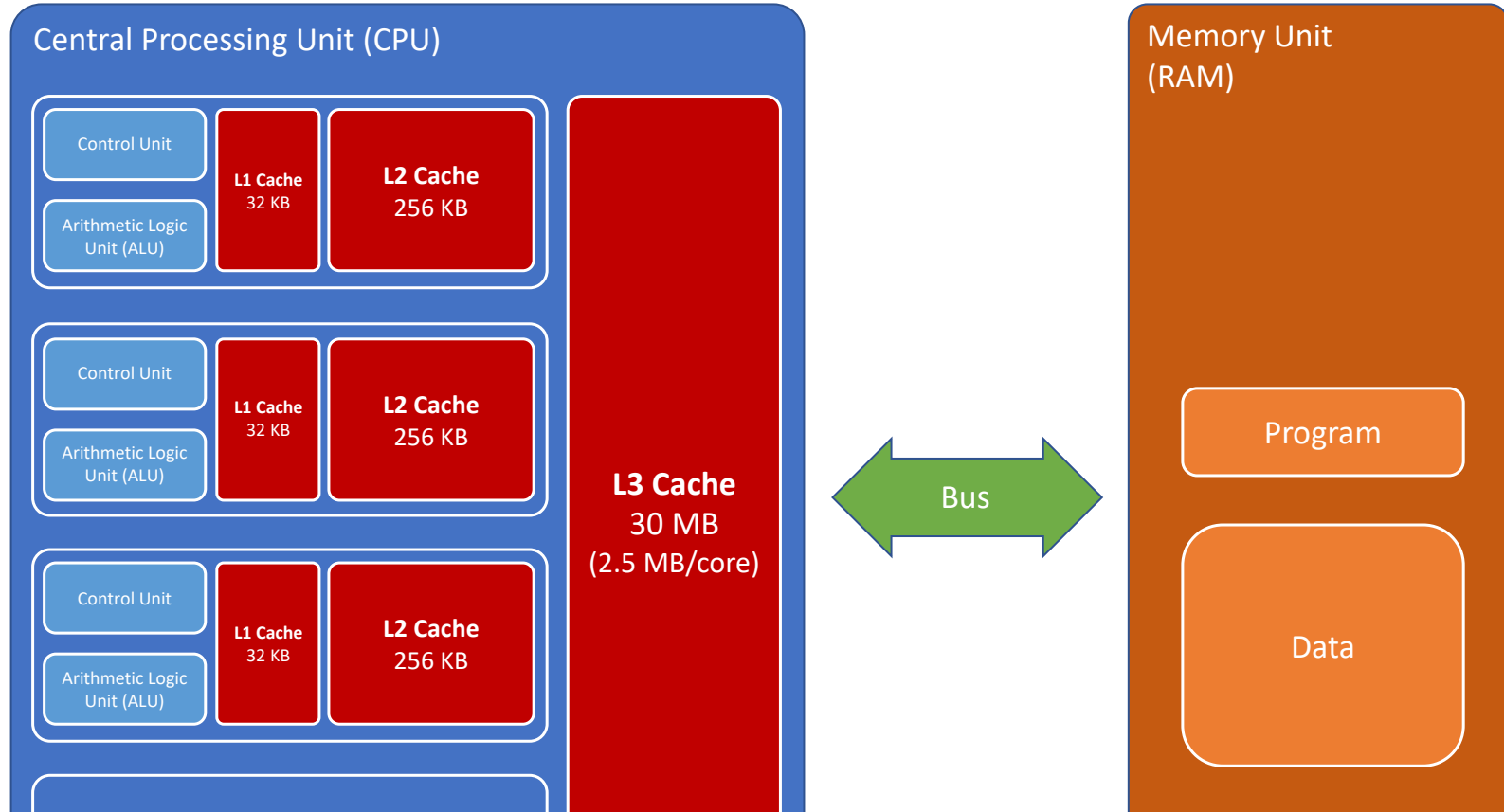
$n = 3 \text{ fields} \times (n_x \times n_y \times n_z) \times 4 \text{ bytes}$

Cache Memory

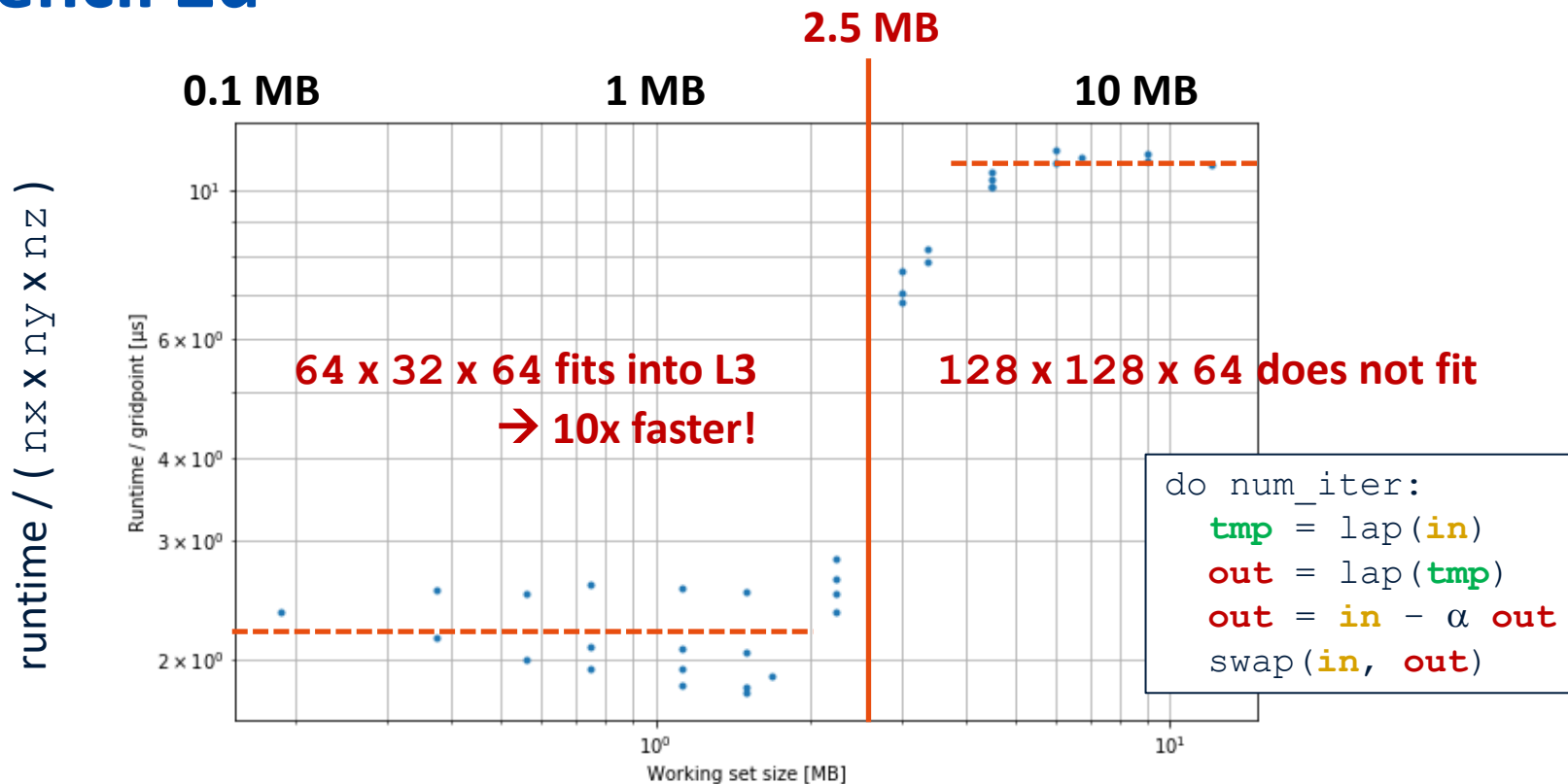


**If the working set fits into cache of limited size,
reads/writes to the memory unit can be avoided (cached)**

Cache Sizes (L1, L2, L3)



Stencil 2d



$$n = 3 \text{ fields} \times (n_x \times n_y \times n_z) \times 4 \text{ bytes}$$

- Caches hold frequently requested data and are used to reduce memory access times.
- Modern CPUs have a hierarchy of caches (L1, L2, L3) of increasing size and access time.
- Data-locality optimizations aim to improve cache use (on all levels of the hierarchy) in order to improve performance.

Lab Exercises

01-roofline-model.ipynb

- Learn about performance metrics and how to compute theoretical peak values.
- Learn about arithmetic intensity and performance limiters.

02-stencil-program.ipynb

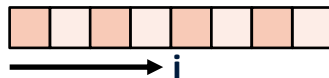
- Determine arithmetic intensity of a stencil program.
- Apply a performance profiling tool to gain insight into performance.
- Show limitations of the von Neumann model for understanding performance.

03-caches-data-locality.ipynb

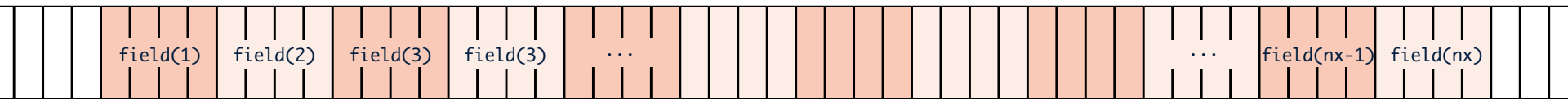
- Learn about caches.
- Apply fusion in the stencil2d program and measure performance improvement.
- Apply inlining in the stencil2d program and measure performance improvement.

How is data stored in memory?

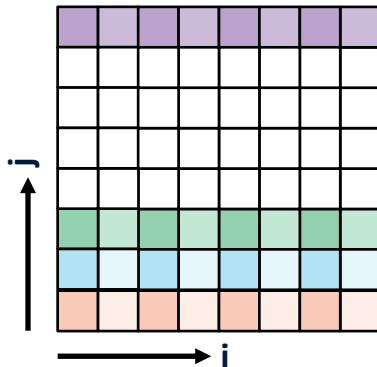
`real (kind=4) :: field(nx)`



Stride in i-direction is 4 bytes

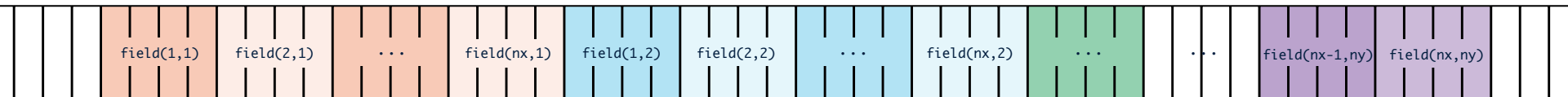


`real (kind=4) :: field(nx, ny)`



Stride in i-direction is 4 bytes

Stride in j-direction is 4 x nx bytes



What is in (which) cache?

Reminder: L1 = 32 KB, L2 = 256 KB, L3 = 2.5 MB/core

```
real (kind=4) :: in(nx + 2*nh, ny + 2*nh, nz)
real (kind=4) :: tmp(nx + 2*nh, ny + 2*nh, nz)
```

```
do k = 1, nz
  do j = 1 + nh, ny + nh
    do i = 1 + nh, nx + nh
      tmp(i,j,k) = -4.0 * in(i,j,k) &
        + in(i-1,j,k) + in(i+1,j,k) &
        + in(i,j-1,k) + in(i,j+1,k)
```

Stride in x-direction is 4 bytes

- Values $\text{in}(i,j,k)$ and $\text{in}(i-1,j,k)$ will always be in **L1 cache**

Stride in y-direction is approx. $4 \times nx$ bytes

- If $nx < 2048$ we can retain 4 x-lines in **L1 cache**
- Then values at j and $j-1$ will be in cache from access to $\text{in}(i,j+1,k)$
- Only read $\text{in}(i,j+1,k)$ and write $\text{tmp}(i,j,k)$ from main memory!

Stride in z-direction is approx. $4 \times nx \times ny$ bytes

- For $nx = ny = 128$, the stride is 64 KB (too large for L1)

A full cube is approx. $4 \times nx \times ny \times nz$ bytes (4 MB)

- For $nx = ny = 128$ and $nz = 64$ this is 4 MB
- If we start iterating again, **tmp** and **in** will be read from main memory!

$\text{in}(i,j,1)$

