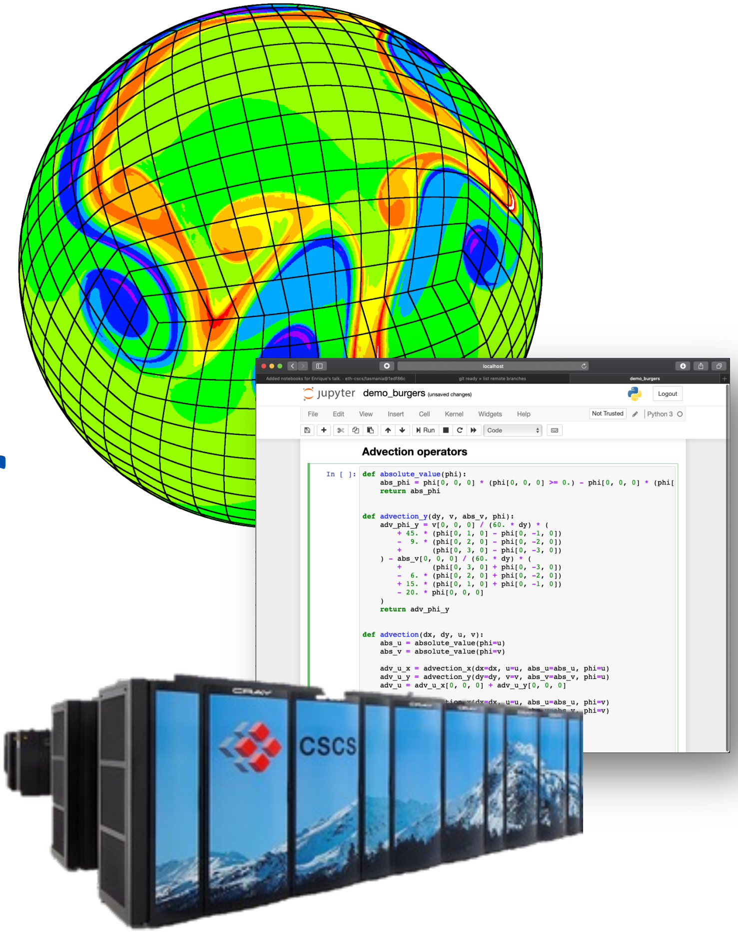
High Performance Computing for Weather and Climate (HPC4WC)



Content: Caches and Data Locality
Lecturers: Oliver Fuhrer
Block course 701-1270-00L
Summer 2022

Why HPC?

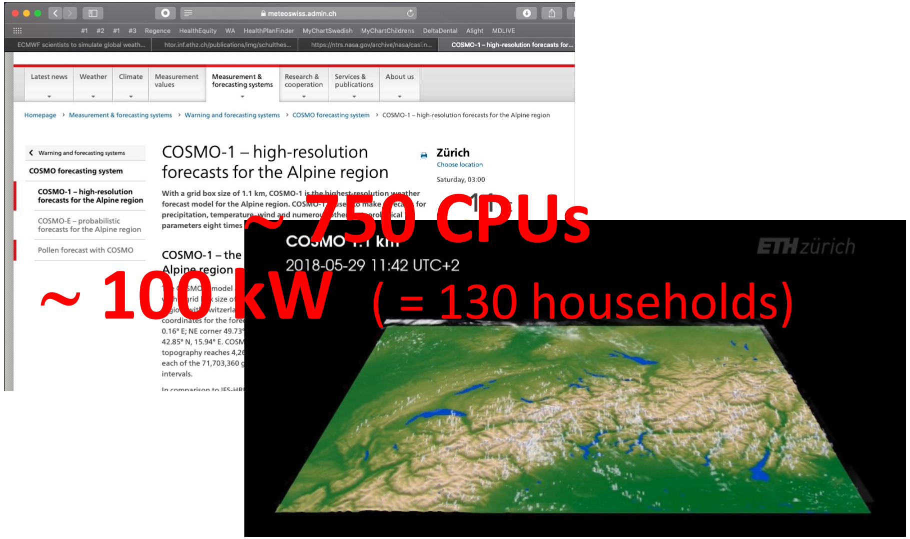
Swiss national weather forecast

1152 x 774 x 80 gridpoints

$\Delta x = 1.1$ km

$\Delta t = 10$ s

80x faster than real time



~ 750 CPUs

~ 100 kW (= 130 households)

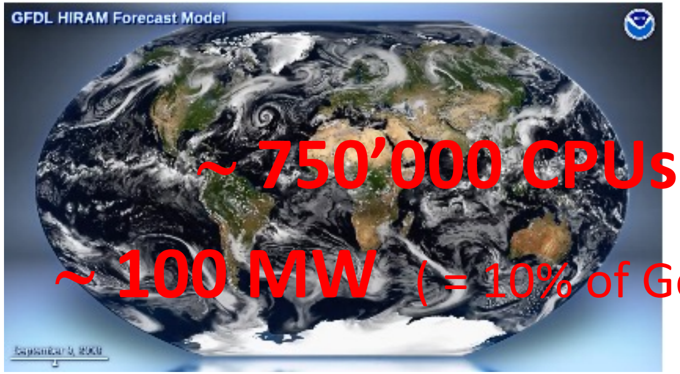
Global climate model at 3 km

57×10^6 x 80 gridpoints

$\Delta x = 3.0$ km

$\Delta t = 9$ s

1000x faster than real time



~ 750’000 CPUs

~ 100 MW (= 10% of Gösgen)

Stencil Program

$$\frac{d}{dt}\mathbf{v} = -2\mathbf{\Omega} \times \mathbf{v} - \frac{1}{\rho}\nabla_3 p + \mathbf{g} + \mathbf{F}$$

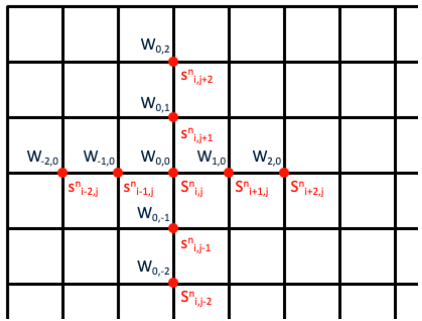
$$C_v \frac{d}{dt}(\rho q) + p\frac{d}{dt}\left(\frac{1}{\rho}\right) = J$$

$$\frac{\partial}{\partial t}(\rho) = -\nabla_3 \cdot (\rho \mathbf{v})$$

$$\frac{\partial}{\partial t} = -\nabla_3 \cdot (\rho \mathbf{v} q) + \rho\,(E - C)$$

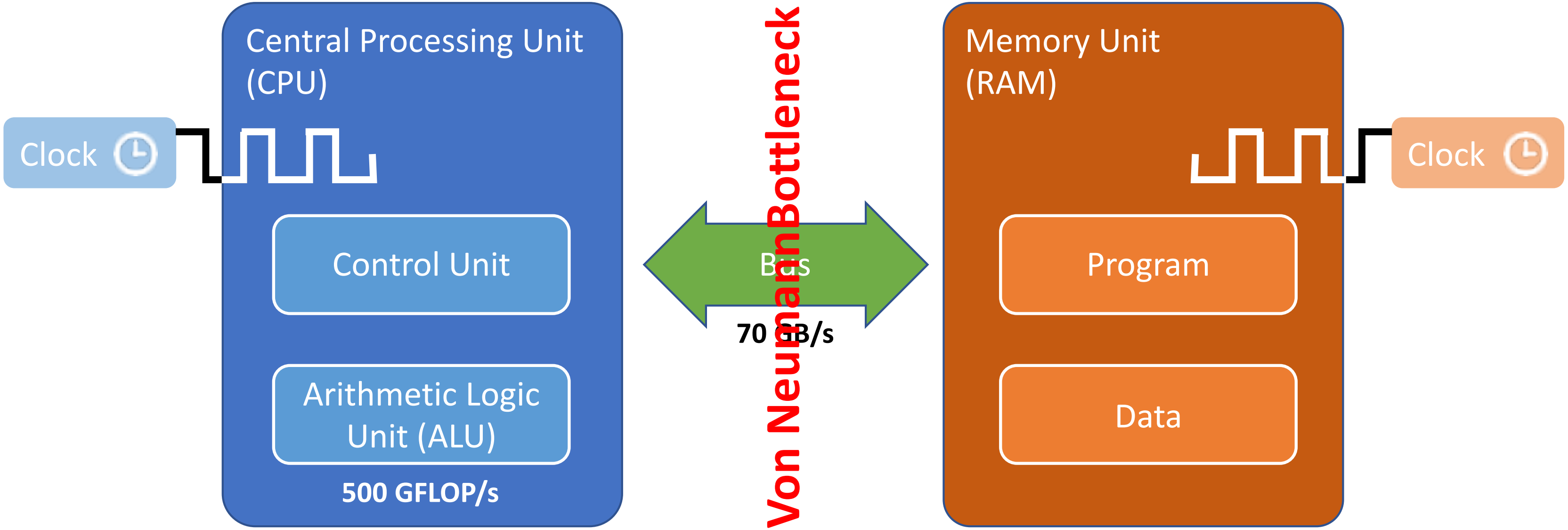
$$p = \rho R T$$

$$s^{n+1}_{i,j} = \sum_{i_{\mathrm{rel}},j_{\mathrm{rel}}} w_{i_{\mathrm{rel}},j_{\mathrm{rel}}}\; s^{n}_{i+i_{\mathrm{rel}},j+j_{\mathrm{rel}}}$$



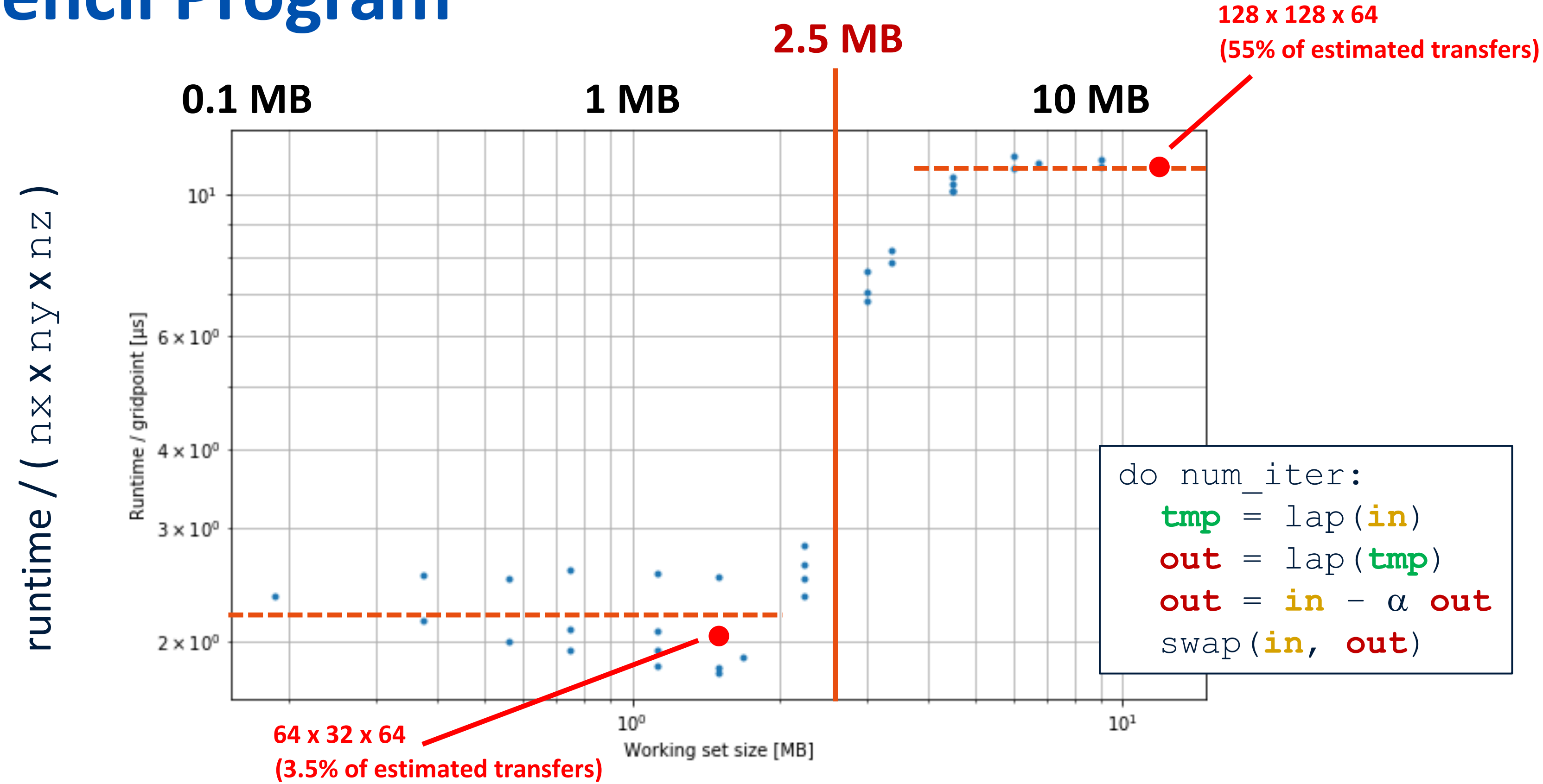

```
# weights literal constants
for j = 1, nj
  for i = 1, ni
    s_new(i, j) = &
      0.125 * s(i-1, j-1) + 0.25 * s(i, j-1) + 0.125 * s(i+1, j-1)  &
      0.25  * s(i-1, j  ) + 1.00 * s(i, j  ) + 0.25  * s(i+1, j  )  &
      0.125 * s(i-1, j+1) + 0.25 * s(i, j+1) + 0.125 * s(i+1, j+1)
```

Von Neumann Architecture



~100 floating-point operations per load/store of a data value!

Stencil Program



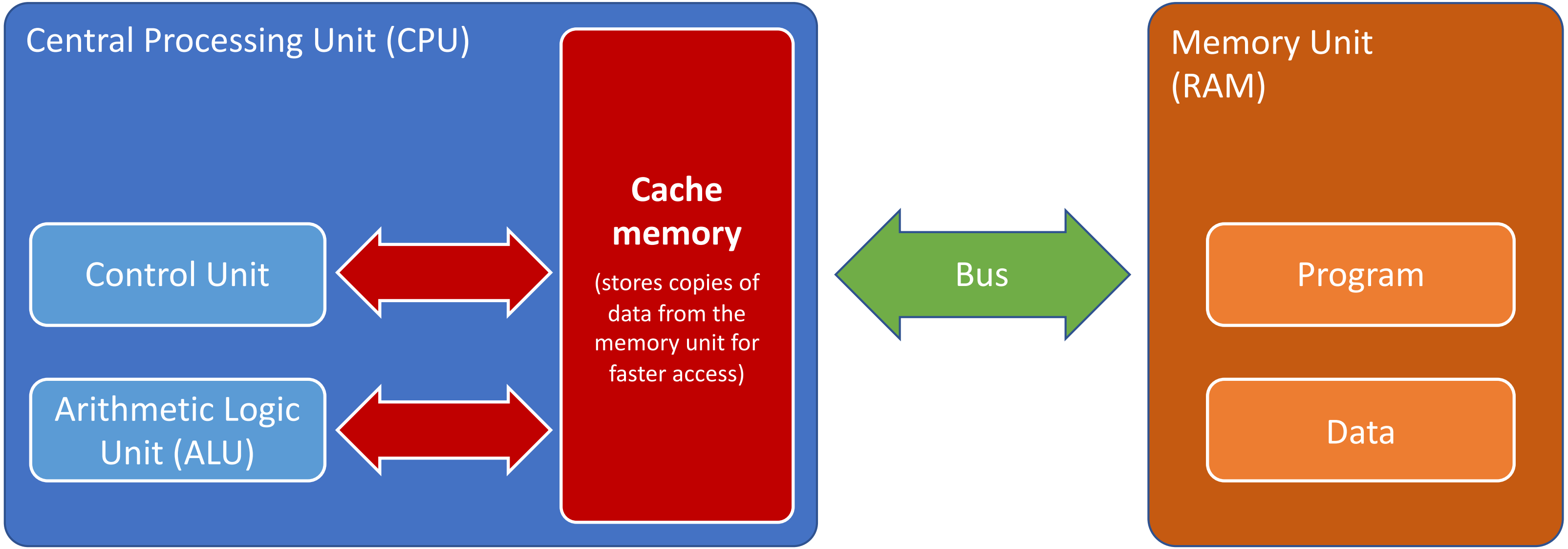

```
do num_iter:
   tmp = lap(in)
   out = lap(tmp)
   out = in - α out
   swap(in, out)
```

n = 3 fields x (nx x ny x nz) x 4 bytes

Learning goals

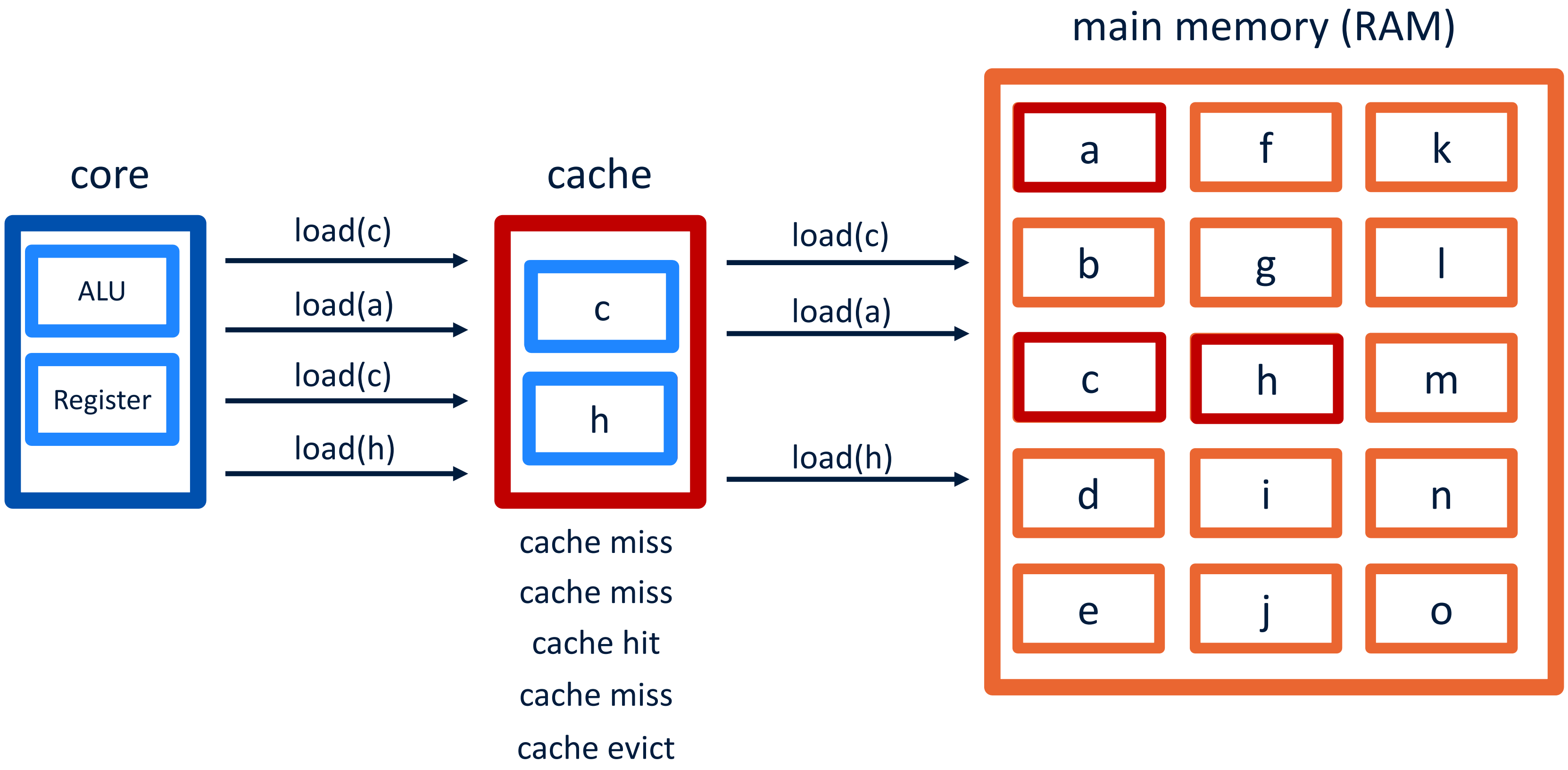
- Understand how data arrays are stored in computer memory
- Understand the implications of the cache hierarchy in a modern multi-core CPU
- Able to do basic data-locality optimizations (fusion, inlining) to improve performance

Cache Memory



Cache is computer memory with short access time used for the storage of frequently or recently used data

Cache Mechanics



Cache Performance Metrics

Miss Rate

- fraction of memory references not found in cache (misses / accesses)
- miss rate = 1 – hit rate
- typical numbers (in percentages)
  - 3-10% for L1
  - can be quite small (< 1%) for L2, depending on size

Hit Time

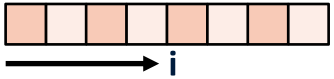
- Time to deliver a line in the cache to the processor
  - includes time to determine whether the line is in the cache

Miss Penalty

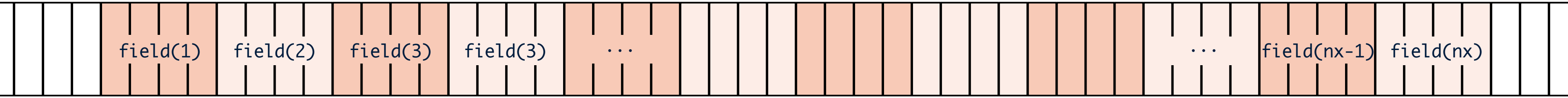
- Additional time required because of a miss

How is data stored in memory?

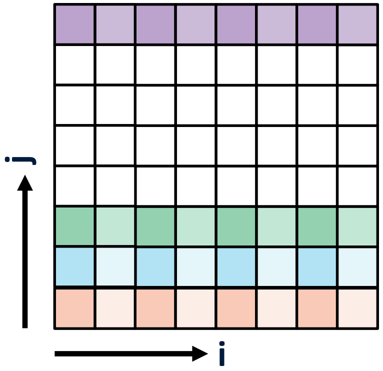
`real (kind=4) :: field(nx)`



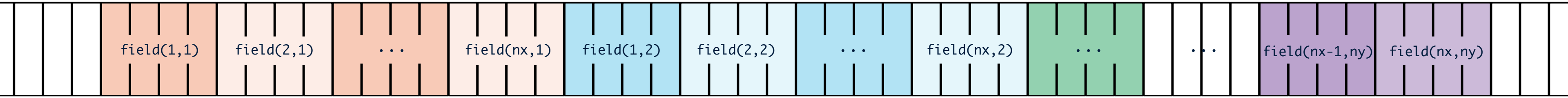
Stride in i-direction is 4 bytes



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



Stride in i-direction is 4 bytes
Stride in j-direction is 4 x `nx` bytes



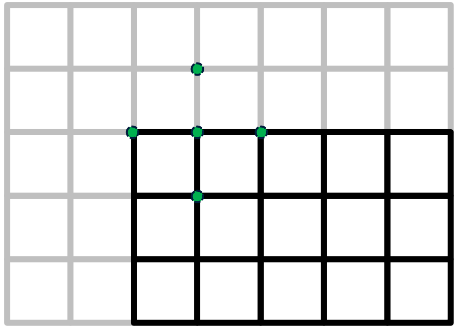
Why Caches Work

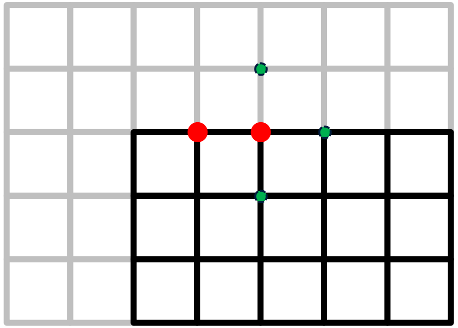
Locality: Programs tend to use data and instructions with addresses near or equal to those they have used recently

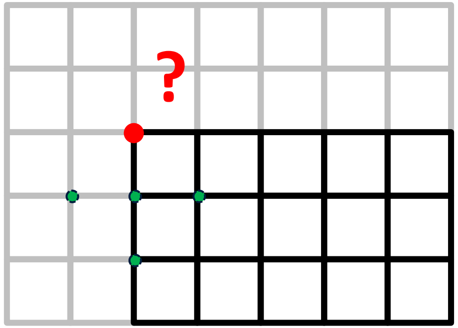
e.g. stencil computations

```
# weights literal constants
for j = 1, nj
  for i = 1, ni
    s_new(i, j) = &
      0.125 * s(i-1, j-1) + 0.25 * s(i, j-1) + 0.125 * s(i+1, j-1)  &
      0.25  * s(i-1, j  ) + 1.00 * s(i, j  ) + 0.25  * s(i+1, j  )  &
      0.125 * s(i-1, j+1) + 0.25 * s(i, j+1) + 0.125 * s(i+1, j+1)
```

Example: Laplacian

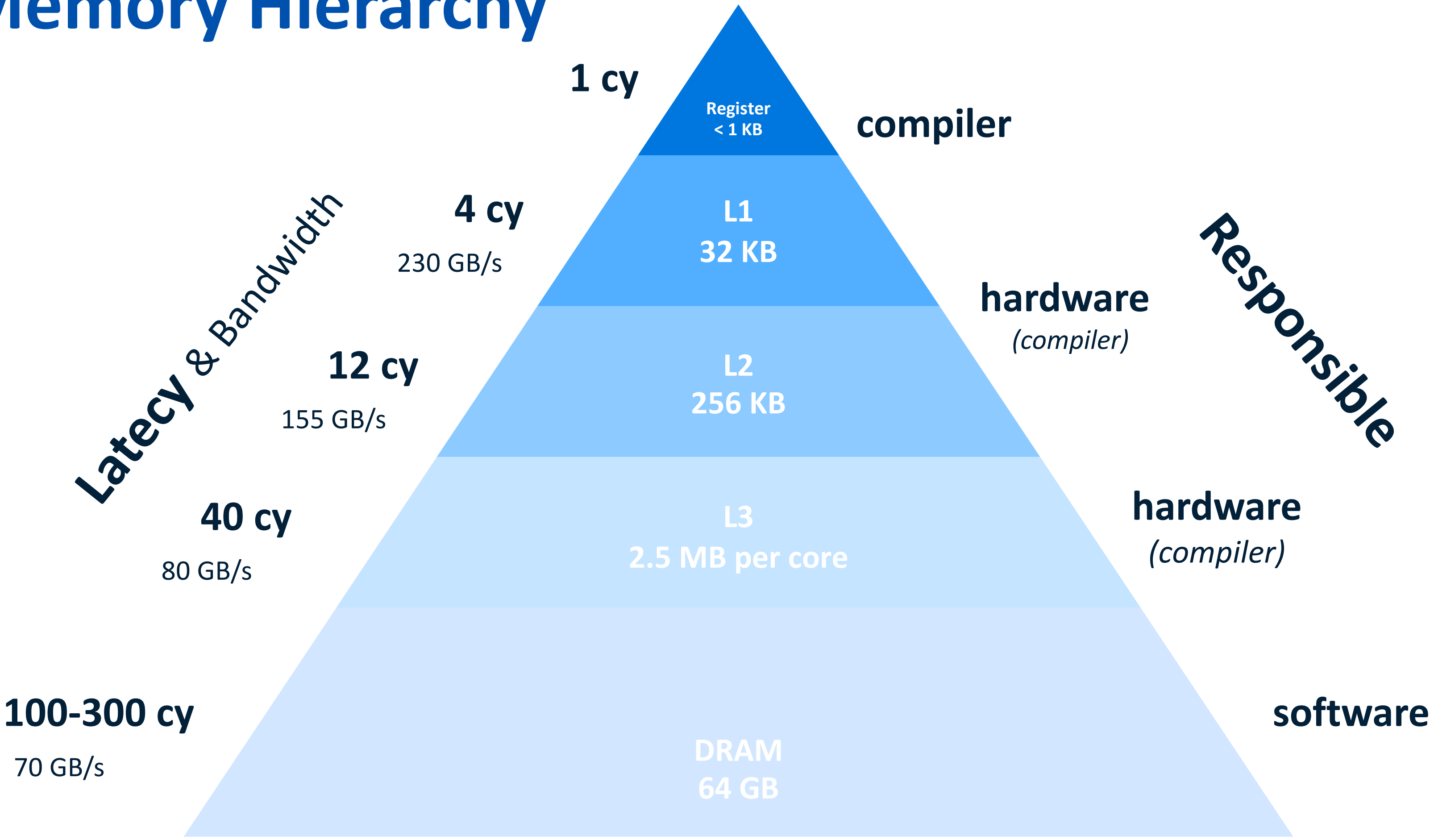




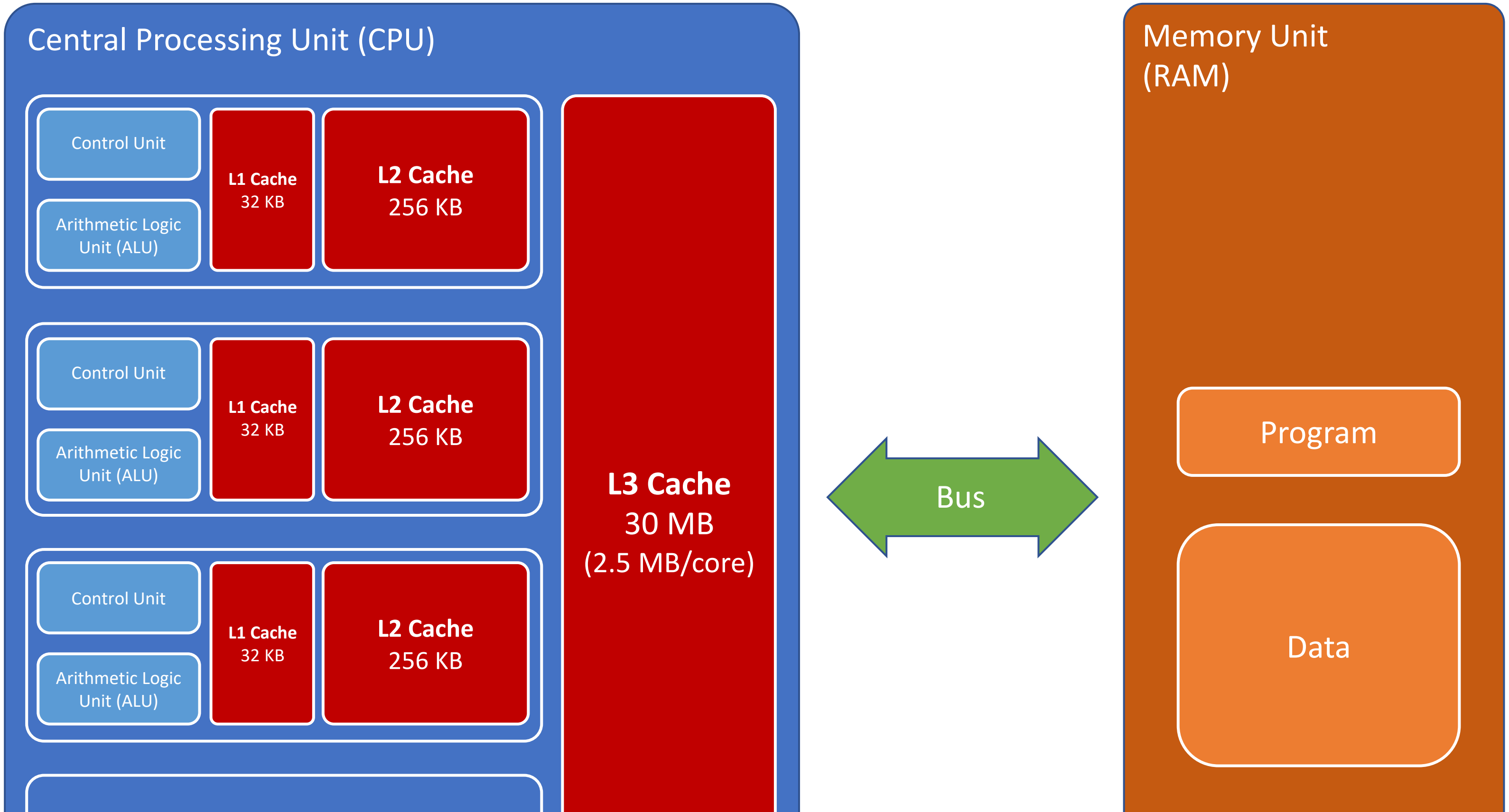


Loops (iteration order) are key to determining whether caches are effective.

Memory Hierarchy

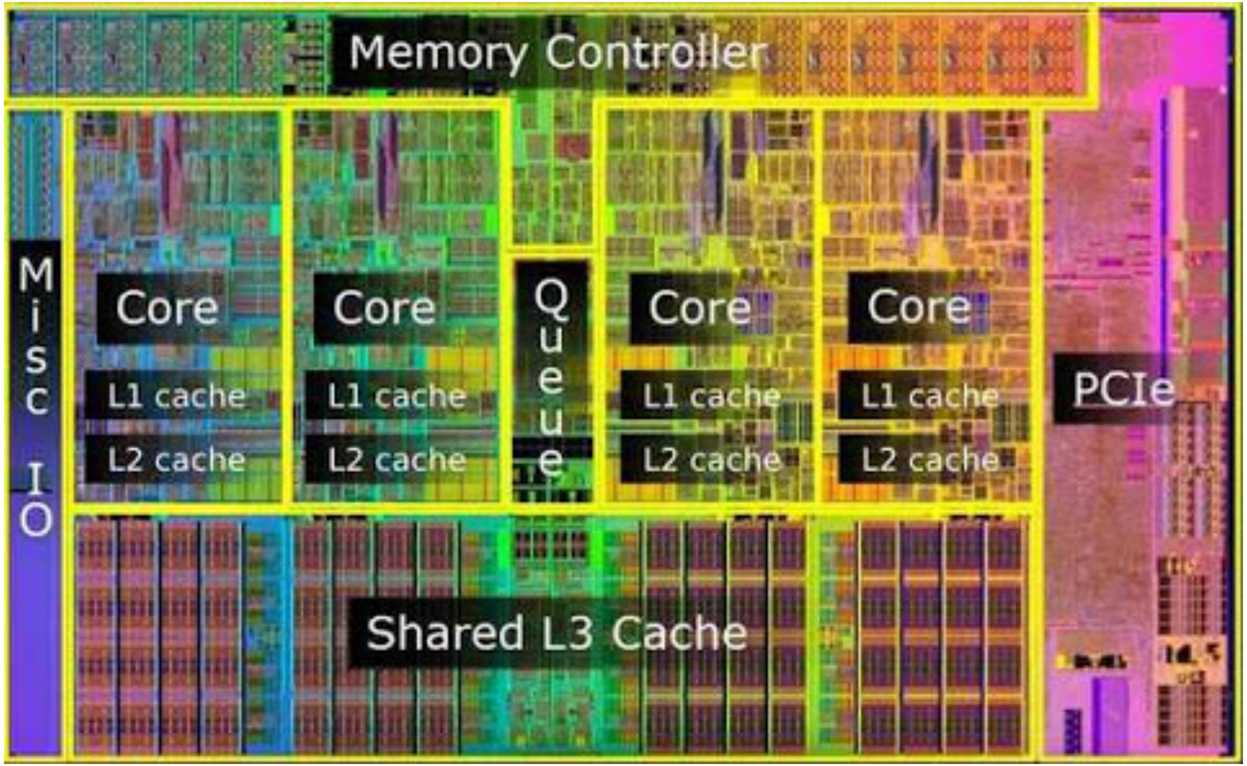


Cache Sizes (L1, L2, L3)

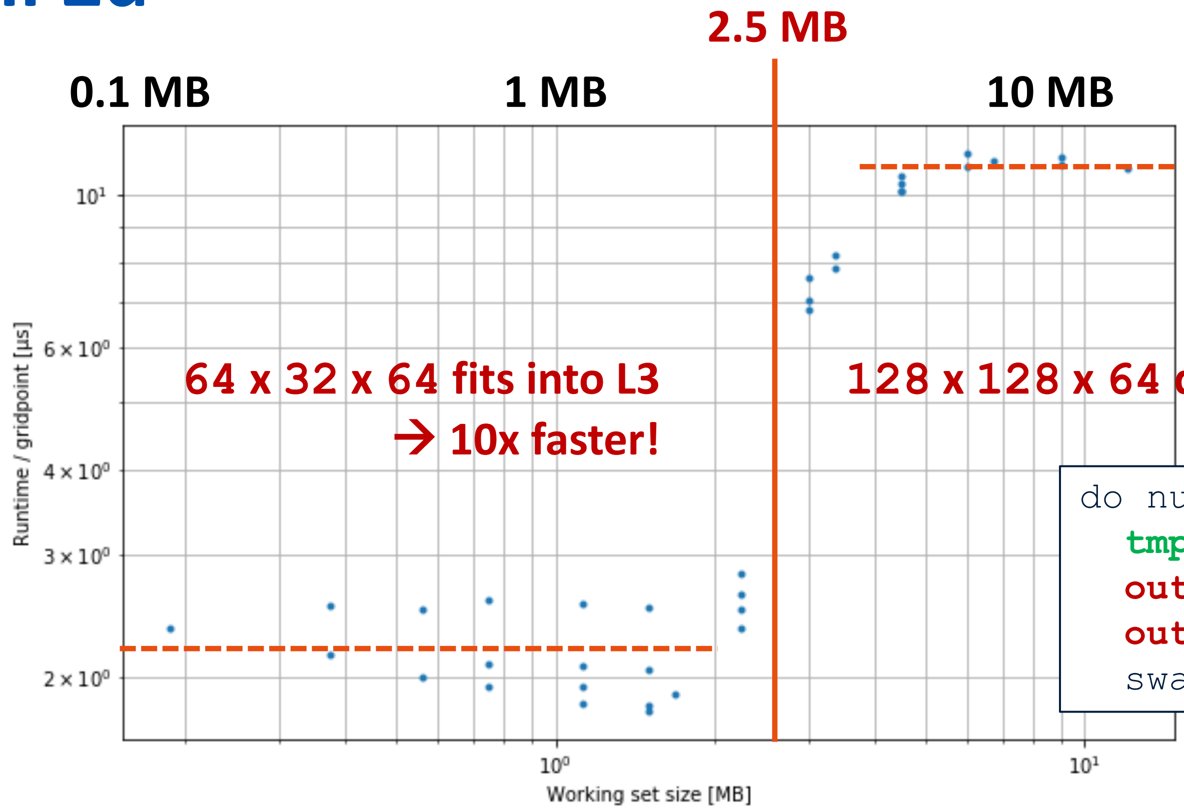


Silicon Real Estate

A large fraction of die real estate is dedicated to cache memory



Stencil 2d



runtime / (nx x ny x nz)

```
do num_iter:
   tmp = lap(in)
   out = lap(tmp)
   out = in − α out
   swap(in, out)
```

n = 3 fields x (nx x ny x nz) x 4 bytes

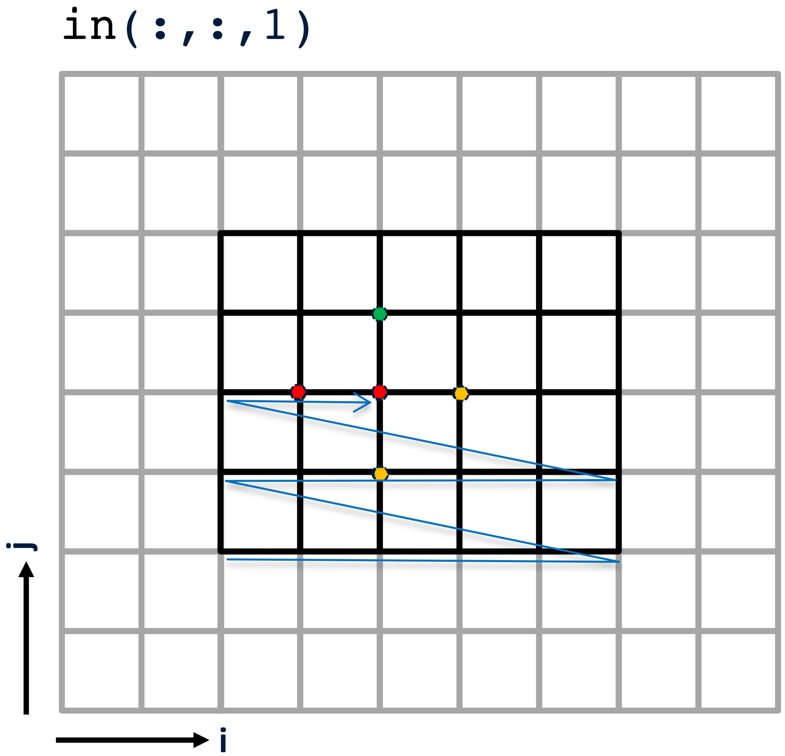
What is in (which) cache?

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)
```

(loop ordering!)



Stride in i-direction is 4 bytes

- Values `in(i,j,k)` and `in(i-1,j,k)` are probably in **L1 cache**

Stride in j-direction is approx. 4 x nx bytes

- For nx = 128, the j-stride is ~ 512 B
- If nx < 2048 we can retain approx. 4 i-lines in L1 cache (32 KB)
- For nx = 128 the loads of `in(i+1,j,k)` and `in(i,j-1,k)` will be in L1
- Only read `in(i,j+1,k)` and write `tmp(i,j,k)` from main memory!

Stride in k-direction is approx. 4 x nx x ny bytes

- For nx = ny = 128, the k-stride is ~64 KB
- **If we do multiple iterations over the ij-plane (k-blocking) we are in L2!**

A full cube is approx. 4 x nx x ny x nz bytes (4 MB)

- For nx = ny = 128 and nz = 64 this is ~4 MB
- If we do multiple iterations over ijk, `tmp` and `in` will be read from main memory!

Border effects for small domains!

Summary

- 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.