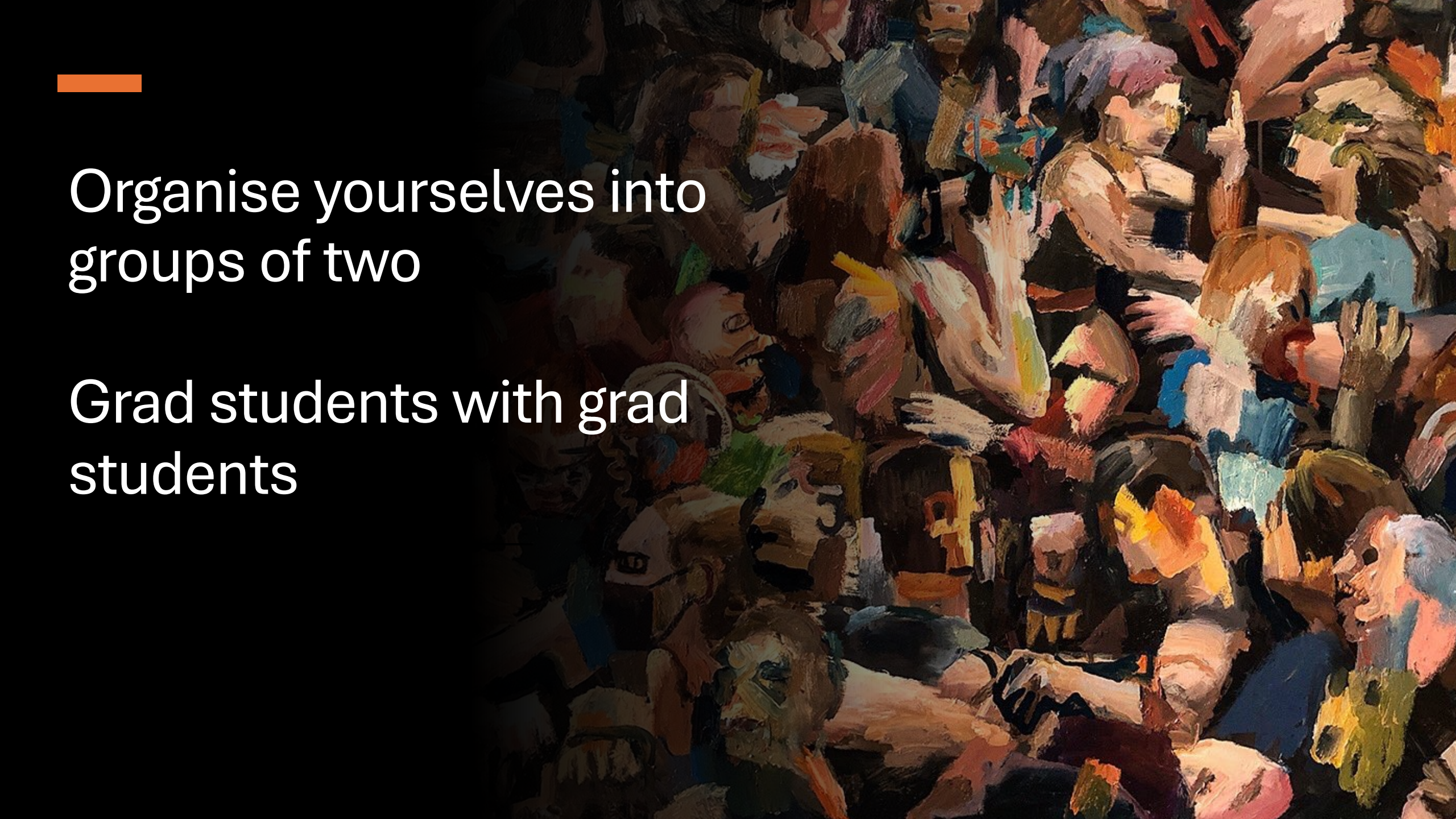


Lecture 21: HPCG and Singularity

Singularity = Apptainer



Organise yourselves into
groups of two

Grad students with grad
students



Organise yourselves into groups
of two

Grad students with grad
students

This is your project 2 group

Condo Partition

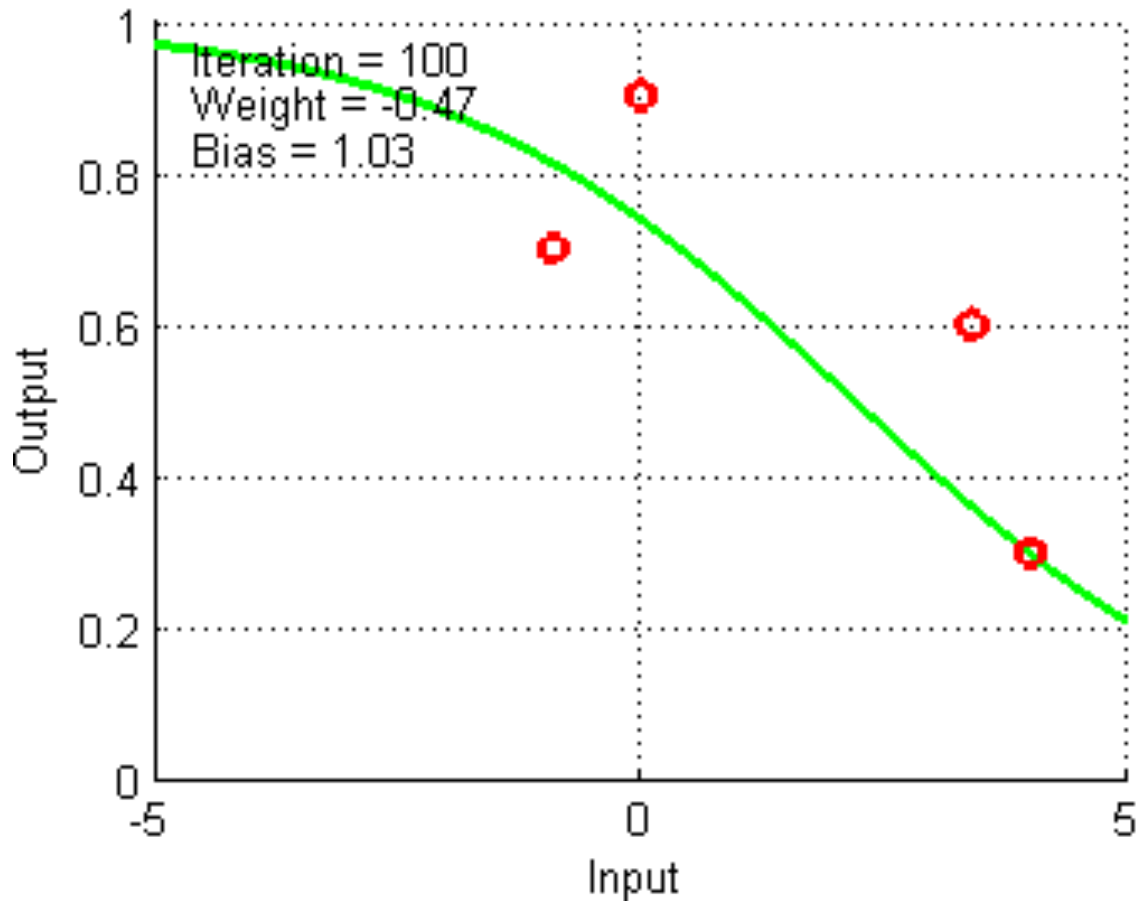
- `sinfo --partition condo`
- `squeue --partition condo`

- Preemption

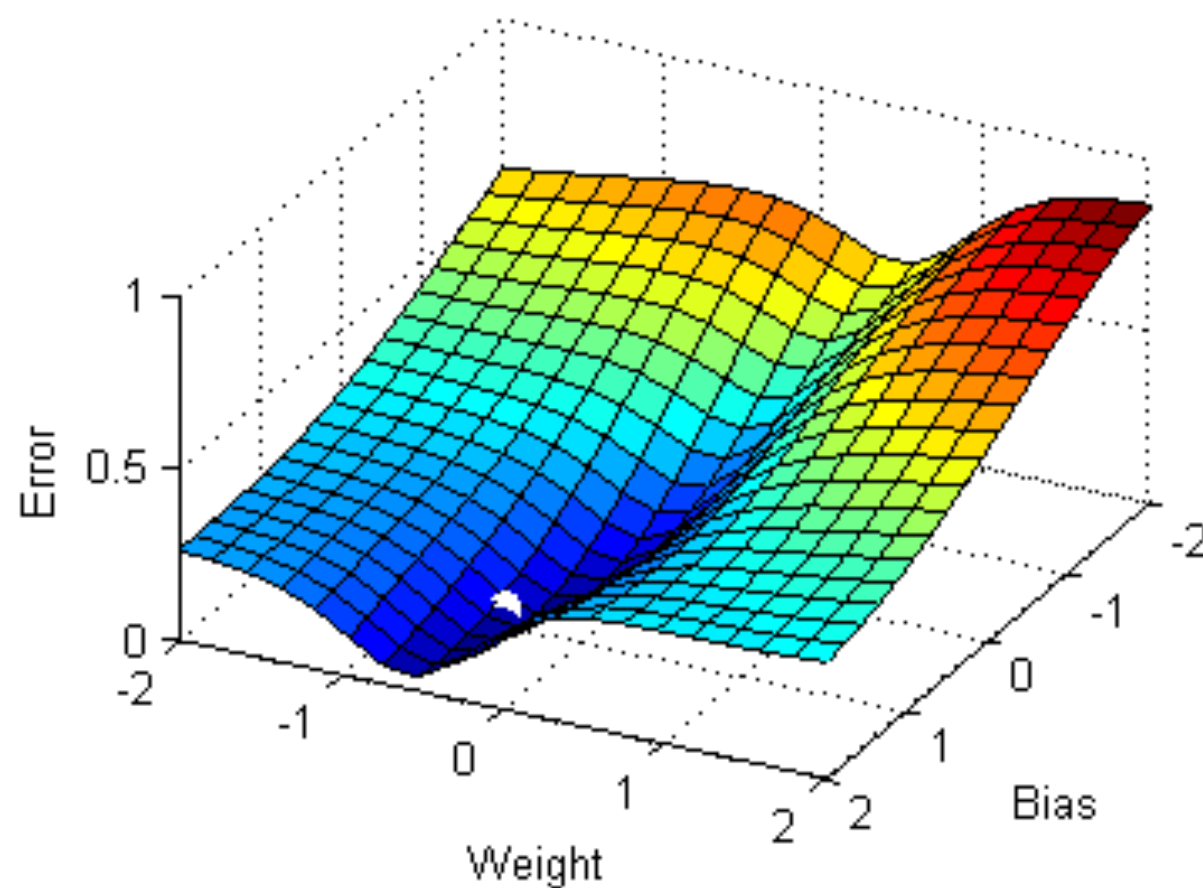
- Do not use the biocomp partition!

Gradient Descent

Function Curve

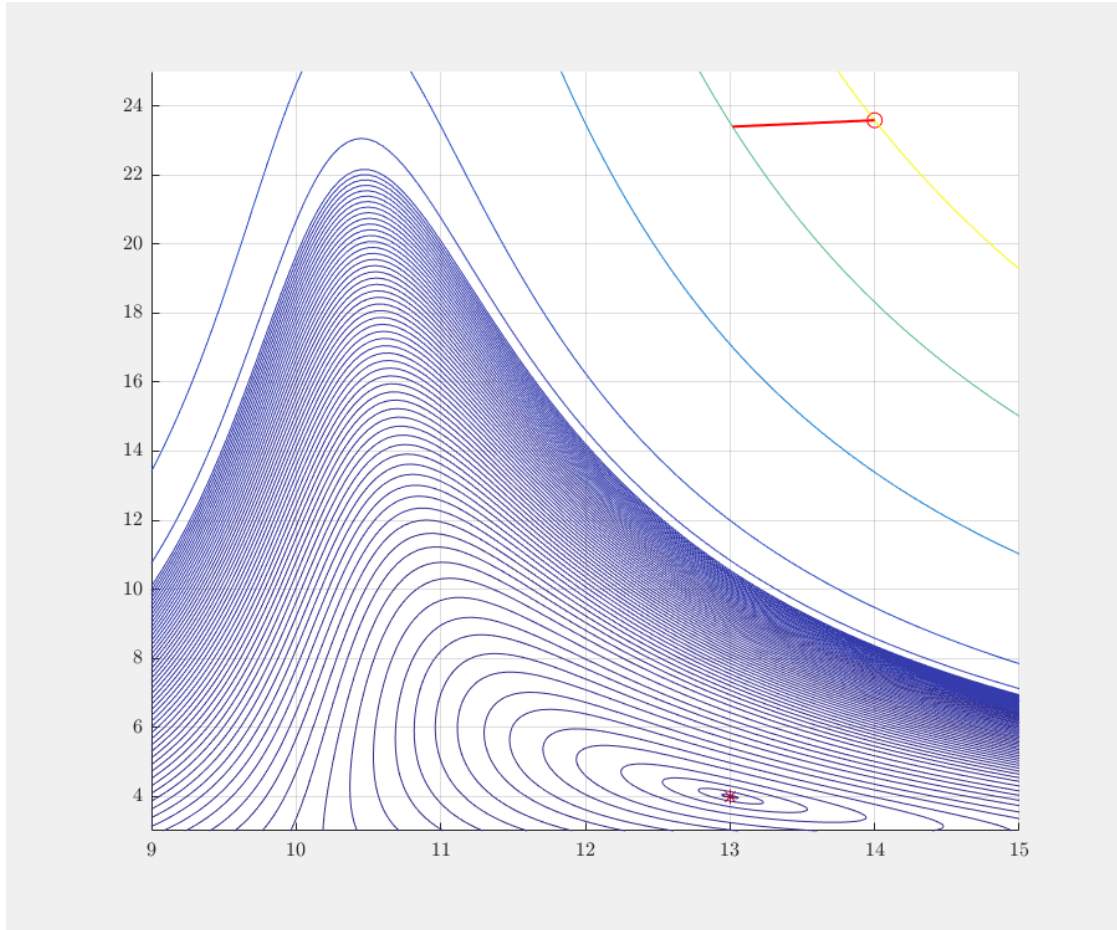


Error Surface

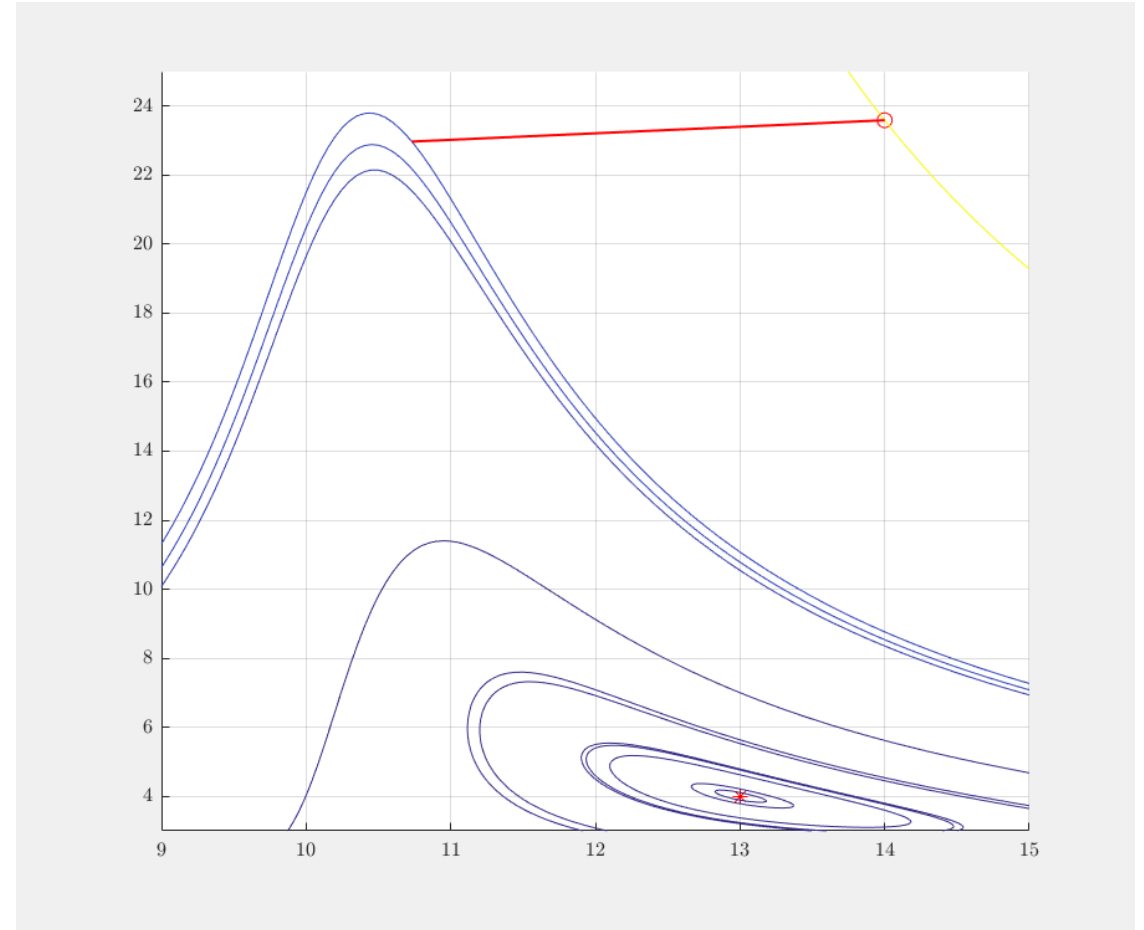


Gradient descent is used for many optimisation problems. Most of Machine Learning is gradient descent (minimizing the difference between predictions and observation). Most of computational chemistry is gradient descent (minimizing the energy of a molecule). Computational fluid dynamics uses gradient descent.

Conjugate Gradient Descent



Steepest Descent



Conjugate Gradient Descent

The difference is that with conjugate gradient descent your next step has to be orthogonal (90 degree turn in 2D) to the last step. Often results in fewer steps to find the optima.

HPCG

- The High Performance Conjugate Gradients (HPCG) Benchmark project is an effort to create a new metric for ranking HPC systems. HPCG is intended as a complement to the High Performance LINPACK (HPL) benchmark, currently used to rank the TOP500 computing systems.
- Gradient Conjugates can be used to approximate the solution to linear systems.

Here is an implementation in Julia*

""" conjugate_gradient!(A, b, x) Return the solution to $A * x = b$ using the conjugate gradient method. """

```
function conjugate_gradient!(
    A::AbstractMatrix, b::AbstractVector, x::AbstractVector; tol=eps(eltype(b))
)
    # Initialize residual vector
    residual = b - A * x
    # Initialize search direction vector
    search_direction = residual
    # Compute initial squared residual norm
    norm(x) = sqrt(sum(x.^2)) old_resid_norm = norm(residual)
    # Iterate until convergence
    while old_resid_norm > tol
        A_ = A
        search_direction = A * search_direction
        step_size = old_resid_norm^2 / (search_direction' * A_ * search_direction)
        # Update solution
        @. x = x + step_size * search_direction
        # Update residual
        @. residual = residual - step_size * A_ * search_direction
        new_resid_norm = norm(residual)
        # Update search direction vector
        @. search_direction = residual + (new_resid_norm / old_resid_norm)^2 * search_direction
        # Update squared residual norm for next iteration
        old_resid_norm = new_resid_norm
    end
    return x
end
```

end

High Performance Gradient Conjugate

```
mfricke@hopper:~ $ mkdir HPCG  
mfricke@hopper:~ $ cd HPCG/  
mfricke@hopper:~/HPCG $
```

Singularity Containers

```
module load singularity
```

```
singularity pull hpc-benchmarks:24.03.sif
```

```
docker://nvcr.io/nvidia/hpc-benchmarks:24.03
```

We use a singularity image containing HPCG built by Nvidia.
Already setup to use GPUs.

Containers make deployment simple since much of the Linux environment is the same as the developers.

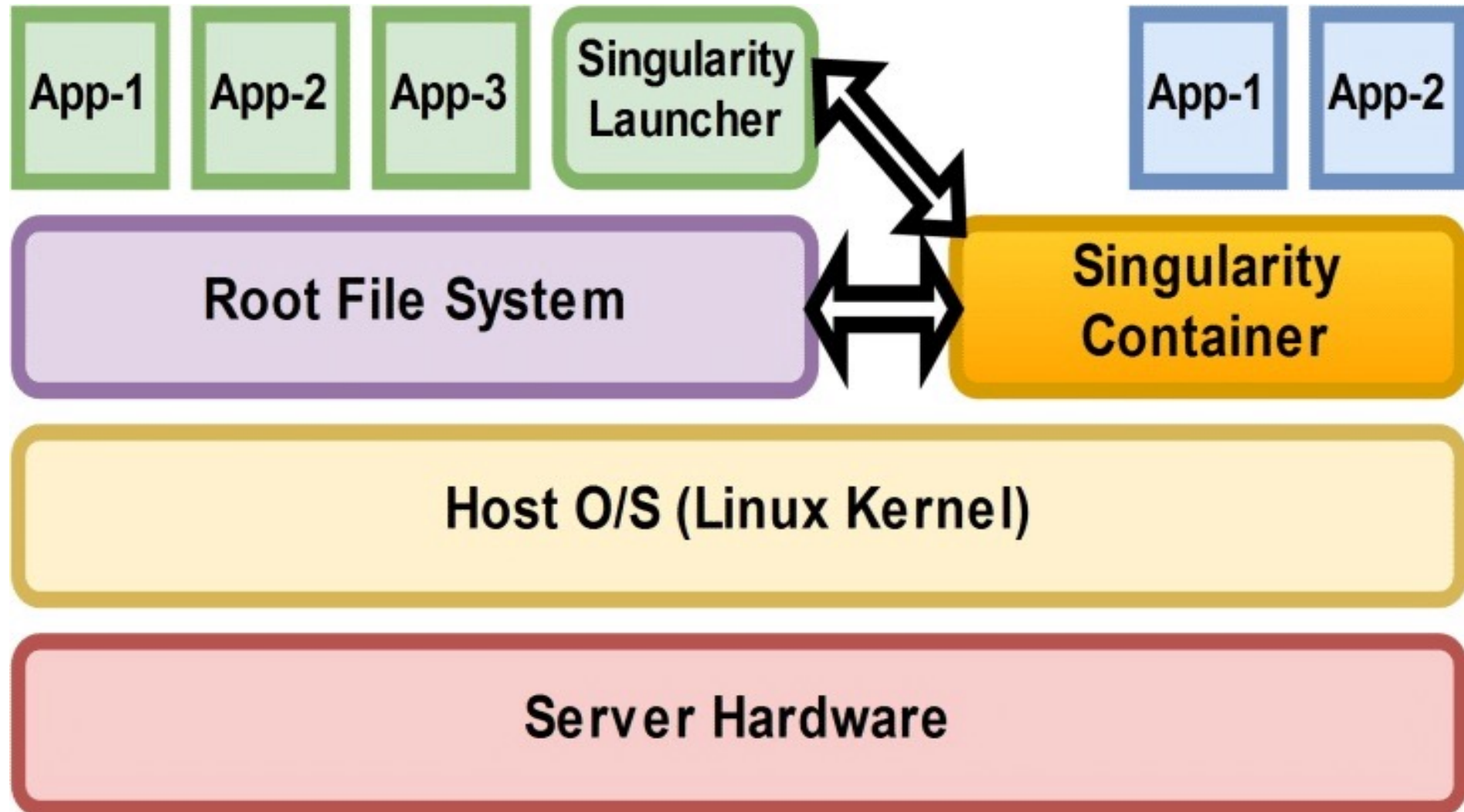
This environment makes Kernel syscalls directly, so it is about as fast as code running on the host.



Apptainer



Container Layout



Navigate the container and copy hpcg.sh and hpcg.dat from the container to the host filesystem.

```
singularity run --bind $PWD:/home_pwd hpc-benchmarks\:24.03.sif bash
```

```
Singularity> ls /workspace
```

```
Singularity> cp hpcg.sh /home_pwd/
```

```
Singularity> cp /workspace/hpcg-linux-x86_64/sample-dat/hpcg.dat  
/home_pwd/
```

```
Singularity> exit
```

```
exit
```

HPCG Input File (hpcg.dat)

HPCG benchmark input file

Sandia National Laboratories; University of
Tennessee, Knoxville

256 256 256

300

Execute HPCG in Singularity Container using Slurm

```
mfricke@hopper:~HPCG/$ srun --mpi=pmi2 --partition  
condo --nodes 1 --ntasks-per-node=1 --gpus 1  
singularity run --nv --bind ./my-dat-files hpc-  
benchmarks:/24.03.sif ./hpcg.sh --dat /my-dat-  
files/hpcg.dat
```

HPCG using Spack Installed Code

```
mfricke@hopper:~HPCG/$ module load intel/20.0.4 hpcg
```

```
mfricke@hopper:~HPCG/$ srun --mpi=pmi2 --partition debug --  
nodes=2 --ntasks=2 --mem=8G xhpcg*
```

```
mfricke@hopper:~HPCG/$ srun --mpi=pmi2 --partition debug --  
nodes=1 --ntasks=1 --mem=8G xhpcg 64 32 128 30
```

64 32 128 is the problem size

30 is the amount of time to spend solving the system of equations with HPCG in seconds

*xhpc reads the hpcg.dat in the current directory

Parameters to play with

- Problem size
- Number of tasks (MPI ranks)
- Number of threads (OpenMP)
- GPU