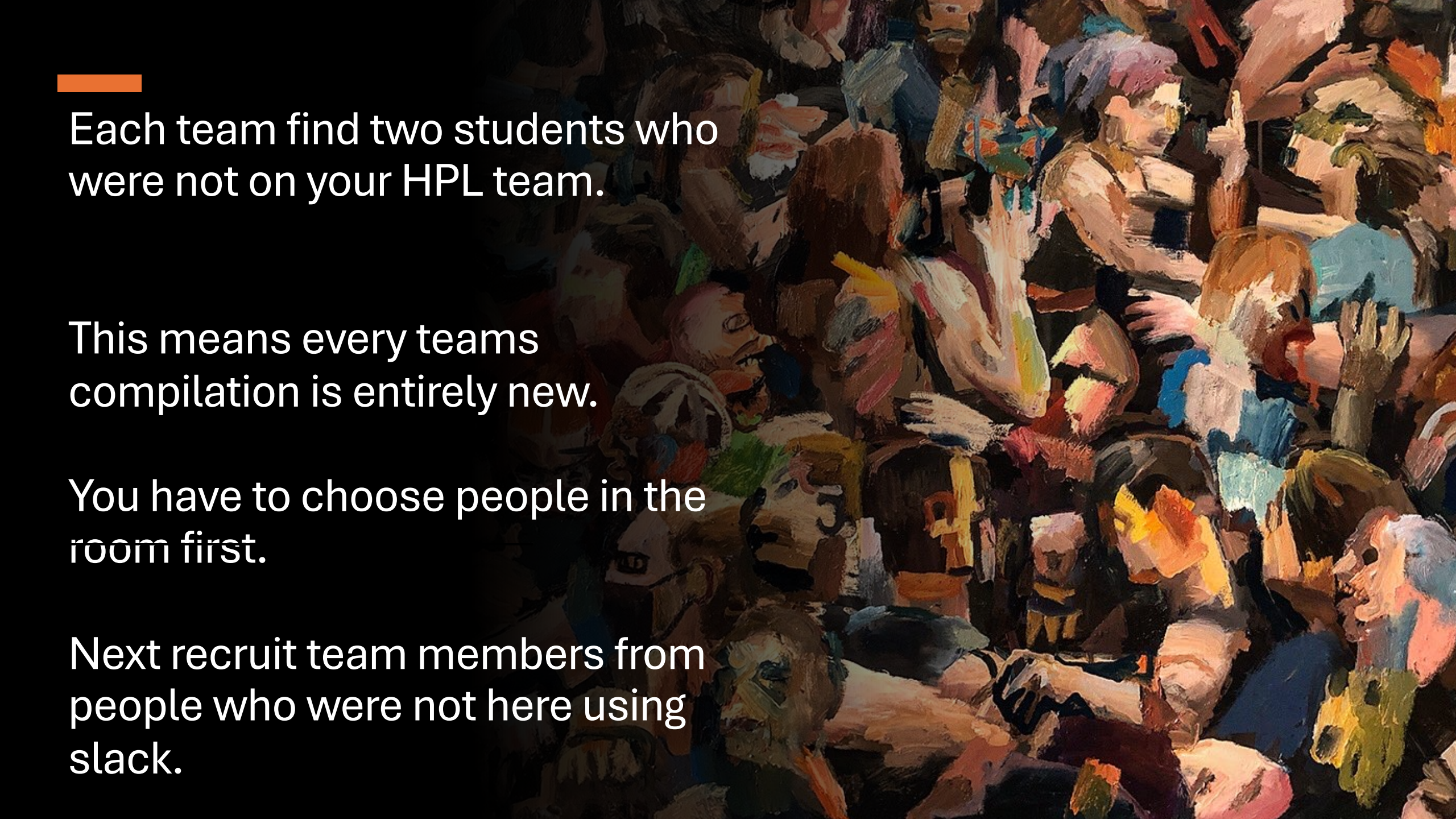
Lecture 21: HPCG and Singularity

Singularity = Apptainer



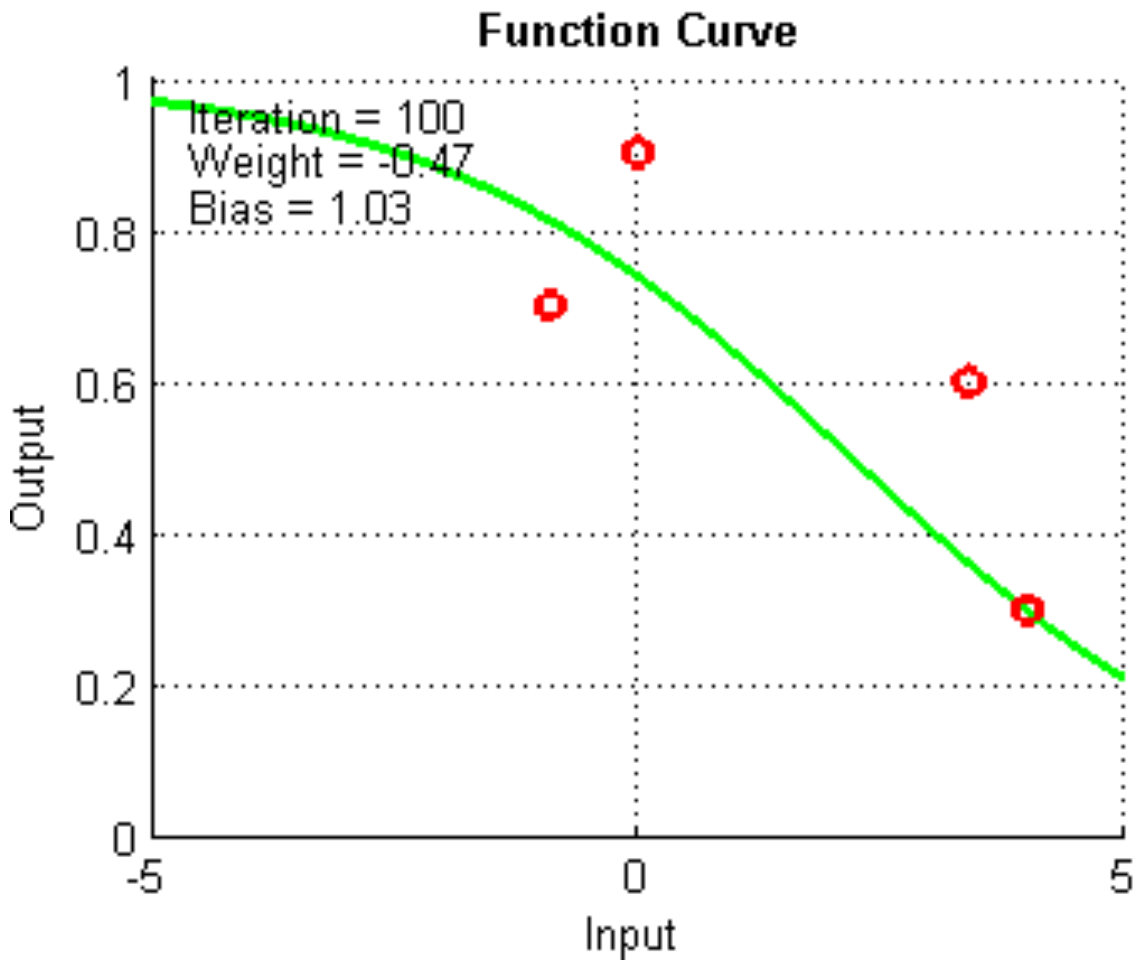
Each team find two students who were not on your HPL team.

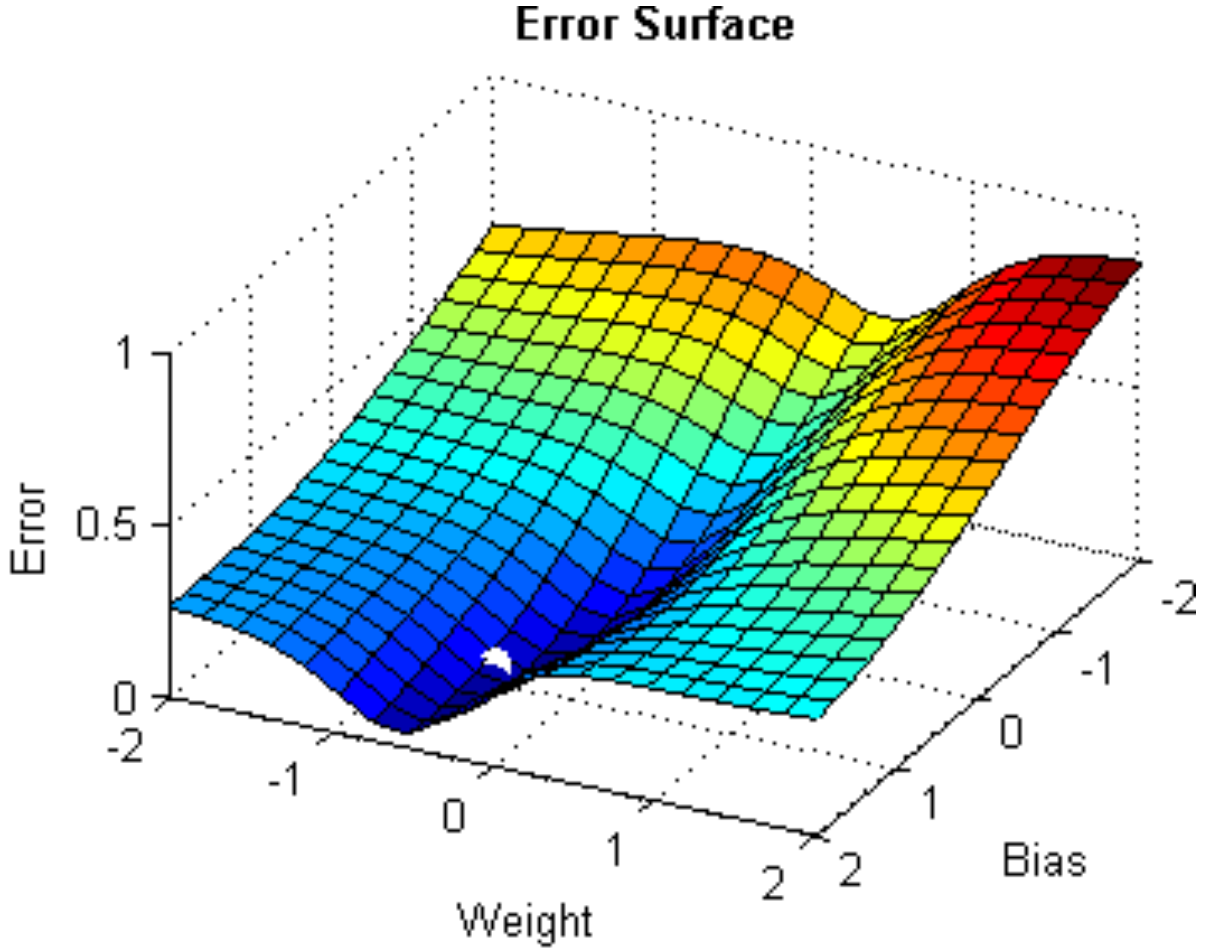
This means every teams compilation is entirely new.

You have to choose people in the room first.

Next recruit team members from people who were not here using slack.

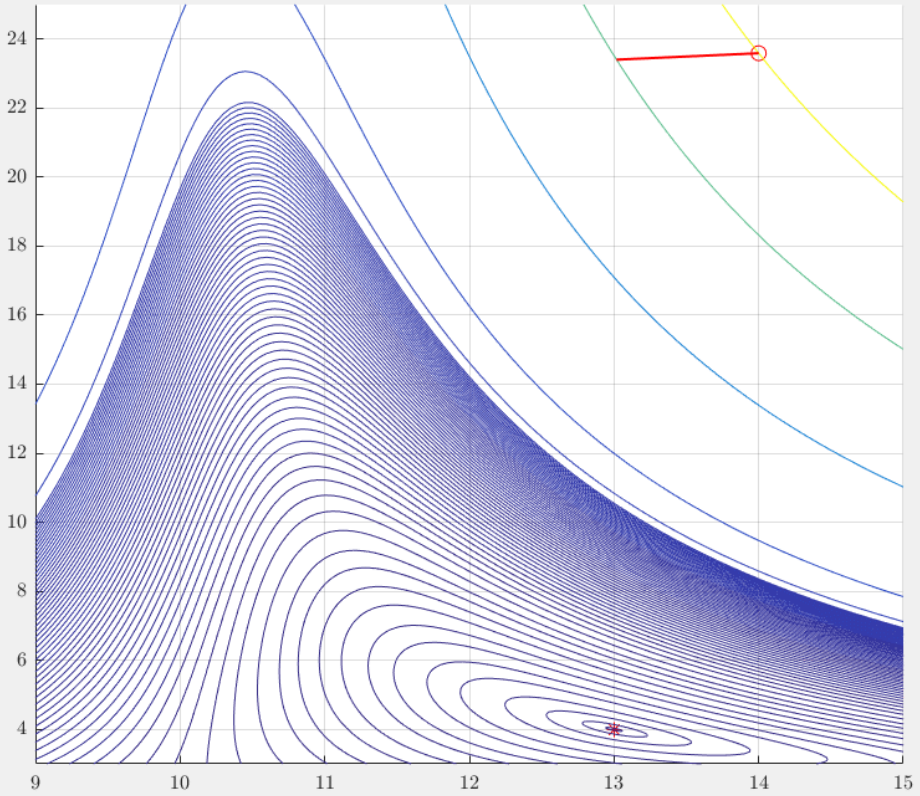
Gradient Descent



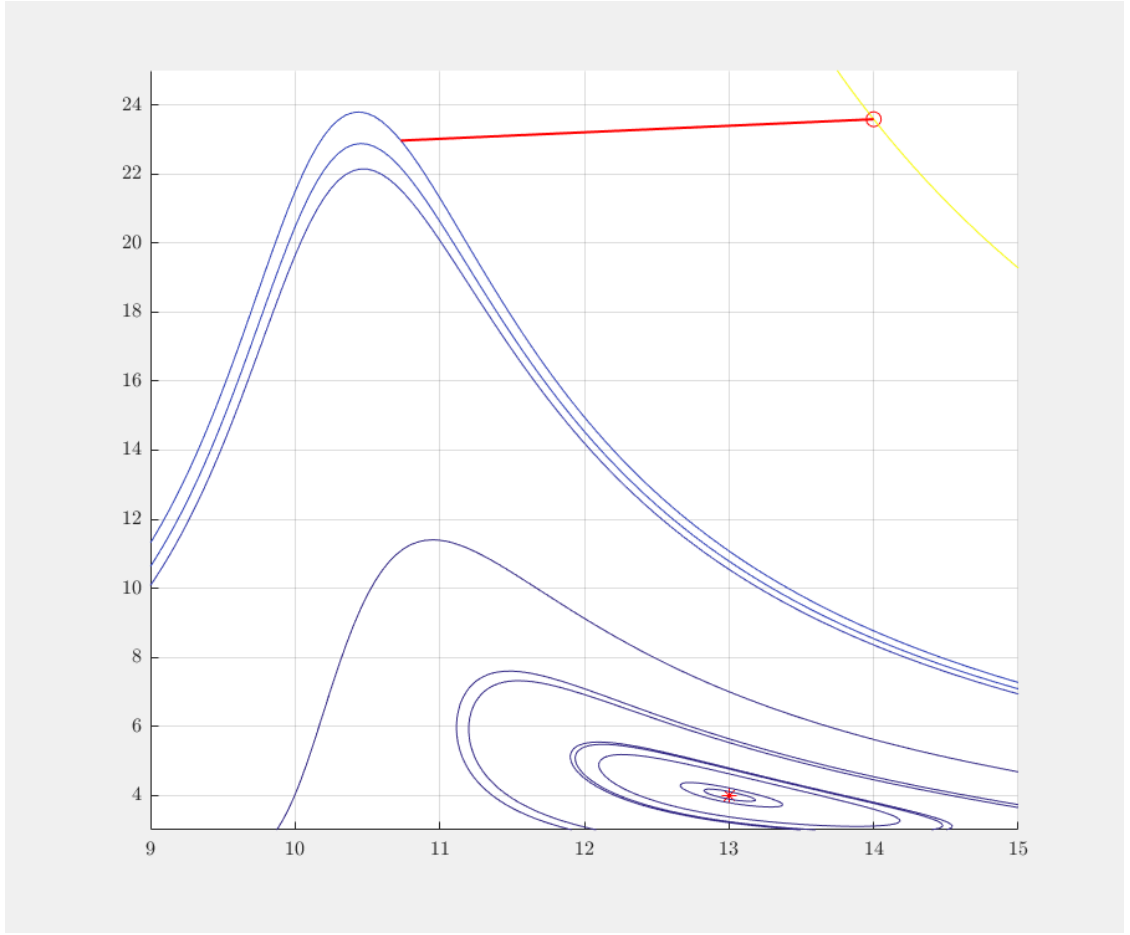


Gradient descent is used for many optimisation problems. Most of Machine Learning is gradient descent (minimizing the difference between predictions and observation). Most of computational chemistry is gradient descent (minimizing the energy of a molecule). Computational fluid dynamics uses gradient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

*wikipedia

High Performance Gradient Conjugate

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

Singularity Containers

Apptainer

On hopper

```
module load singularity
singularity pull hpc-benchmarks:24.03.sif
docker://nvcr.io/nvidia/hpc-benchmarks:24.03
```

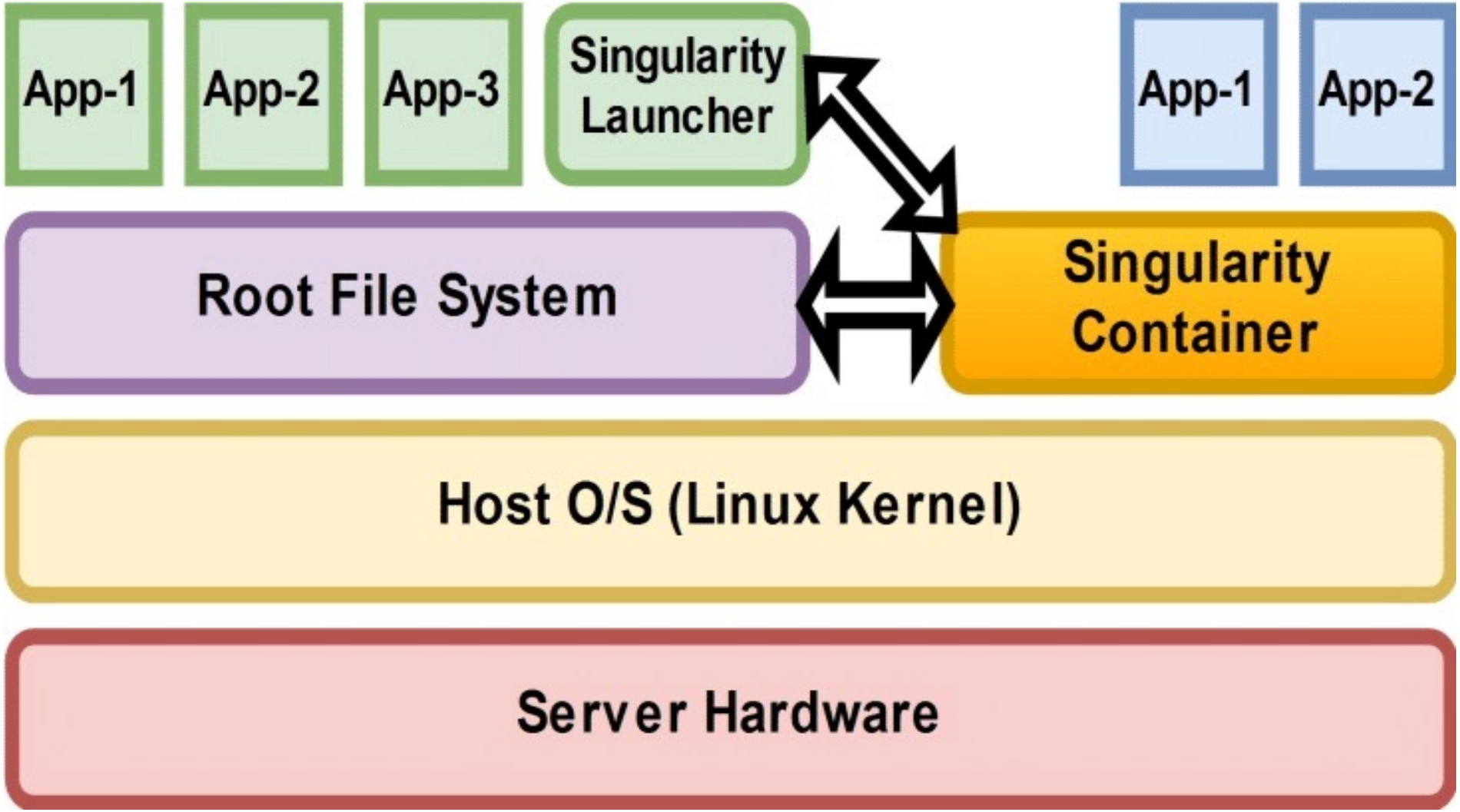
We use a singularity image containing HPCG built by Nvidia.
Already setup to use GPUs.

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.

Greg Greg Kurtzer, CEO of CIQ, inventor of Rocky Linux, Warewulf, and Singularity. SCC 24

Don Becker and Thomas Sterling, creators of the first Beowulf Cluster in 1994 at NASA Goddard, Beowulf Bash 2025

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> cd /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@easley:~HPCG/$
     module module load gcc/14.2.0-cuda-jeua apptainer
mfricke@easley:~HPCG/$ srun –mem 32GB --mpi=pmi2 --
partition l40s --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
```

```
mfricke@easley:~/HPCG $
srun --mem 32GB --mpi=pmi2 --partition l40s --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

You have been allocated one or more GPUs.
Job 128286 running on easley055

================================================================
================ NVIDIA HPC Benchmarks ================
================================================================
NVIDIA Release 24.03
Copyright (c) 2023, NVIDIA CORPORATION & AFFILIATES. All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION & AFFILIATES.  All rights reserved.

This container image and its contents are governed by the NVIDIA Deep Learning Container License.
By pulling and using the container, you accept the terms and conditions of this license:
https://developer.nvidia.com/ngc/nvidia-deep-learning-container-license

NOTE: Mellanox network driver detected, but NVIDIA peer memory driver not
      detected.  Multi-node communication performance may be reduced.

HPCG-NVIDIA 24.3.0  -- NVIDIA accelerated HPCG benchmark -- NVIDIA
Build v0.2.1

Start of application (GPU-Only) ...
Initial Residual = 5660.69
Iteration = 1   Scaled Residual = 0.185784
Iteration = 2   Scaled Residual = 0.101895
Iteration = 3   Scaled Residual = 0.0702417
Iteration = 4   Scaled Residual = 0.0535779
Iteration = 5   Scaled Residual = 0.0432483
Iteration = 6   Scaled Residual = 0.0362099
Iteration = 7   Scaled Residual = 0.0311113

…
```

```
mfricke@easley:~ $ squeue --me
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
            128286       l40s singular  mfricke  R       3:38      1 easley055
mfricke@easley:~ $ ssh easley055
mfricke@easley:~ $ nvtop
```

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 *per process (a rank 3 tensor)*

So total problem size is N processes x 64 x 32 x 128 and is reported in the output file

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

*xhpcg reads the hpcg.dat in the current directory

Parameters to play with

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