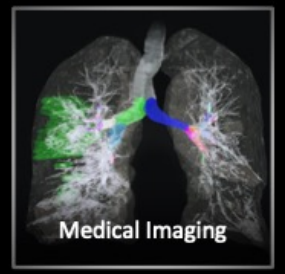
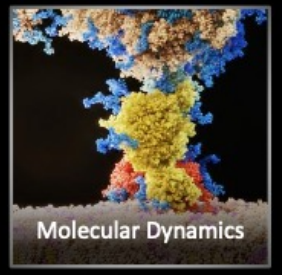
INTRODUCTION TO GPU COMPUTING

N-WAYS GPU BOOTCAMP





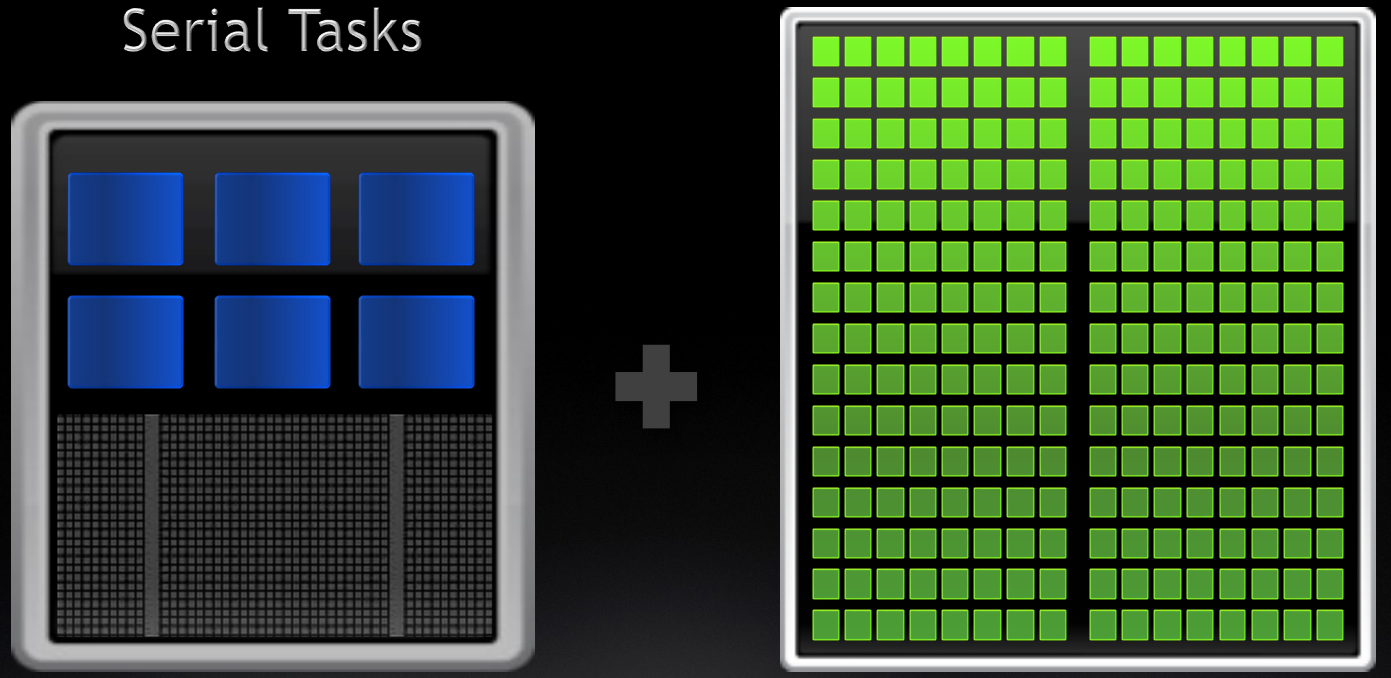
Universe of GPU Computing

GPU PROGRAMMING
(FOUNDATIONS)

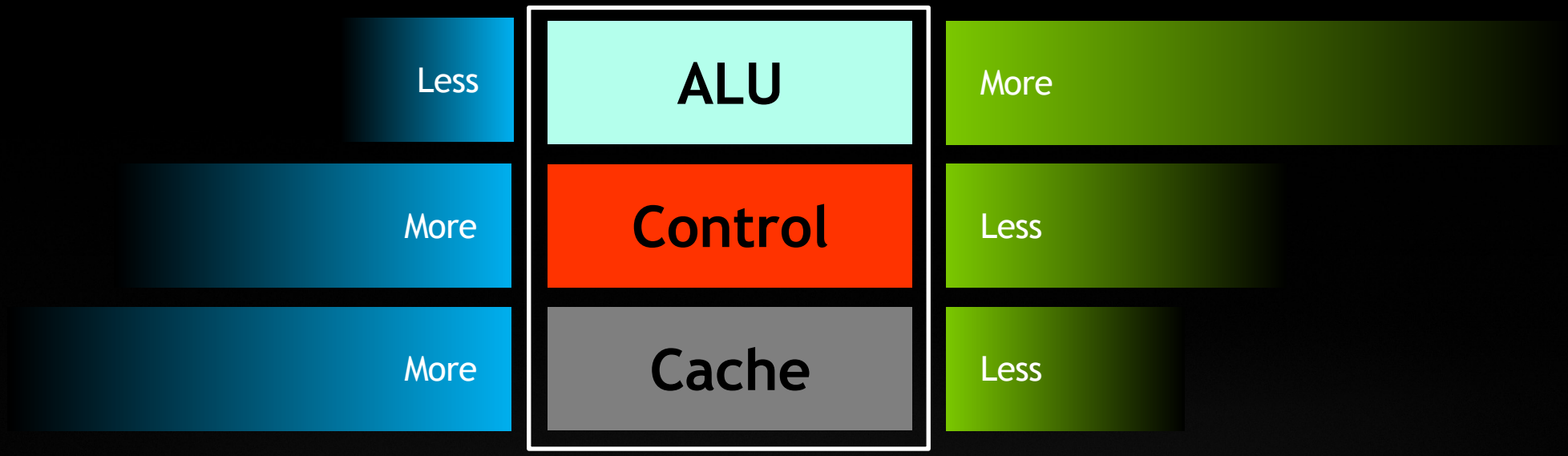
FUNDAMENTALLY DIFFERENT

CPU
Optimized for
Serial Tasks

GPU Accelerator
Optimized for
Parallel Tasks

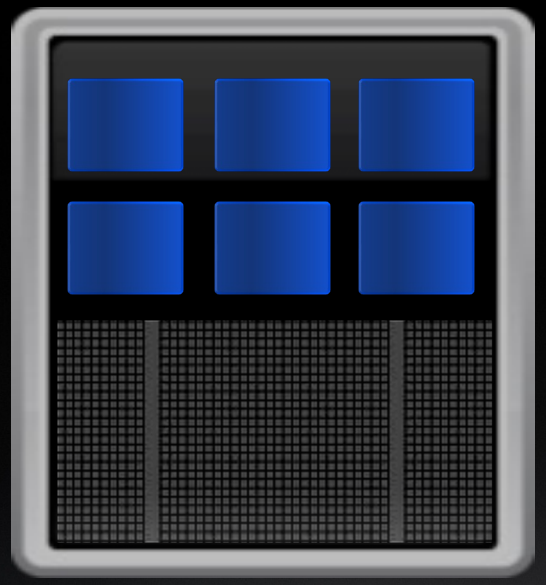


SILICON BUDGET



CPU IS A LATENCY REDUCING ARCHITECTURE

CPU
Optimized for
Serial Tasks



CPU Strengths

- Very large main memory
- Very fast clock speeds
- Latency optimized via large caches
- Small number of threads can run very quickly

CPU Weaknesses

- Relatively low memory bandwidth
- Cache misses very costly
- Low performance/watt

GPU IS ALL ABOUT HIDING LATENCY

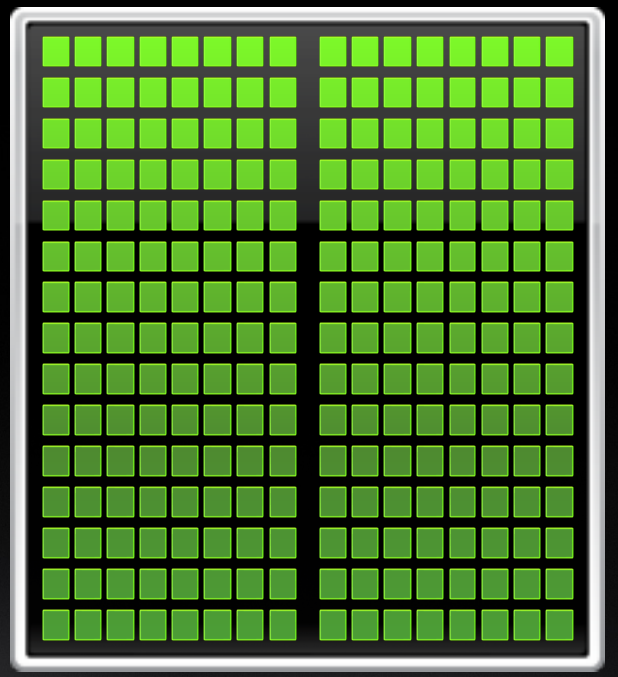
GPU Strengths

- High bandwidth main memory
- Significantly more compute resources
- Latency tolerant via parallelism
- High throughput
- High performance/watt

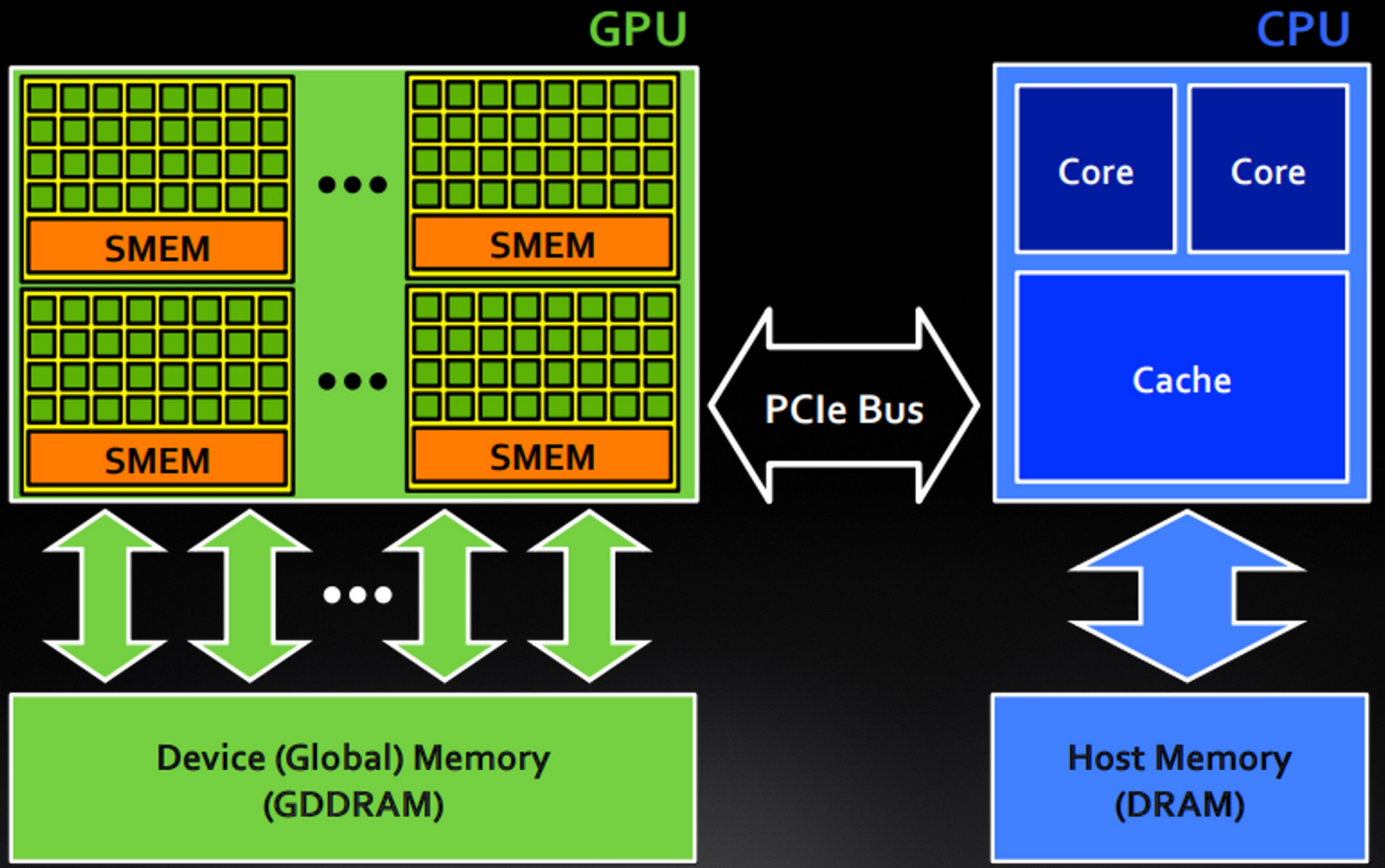
GPU Weaknesses

- Relatively low memory capacity
- Low per-thread performance

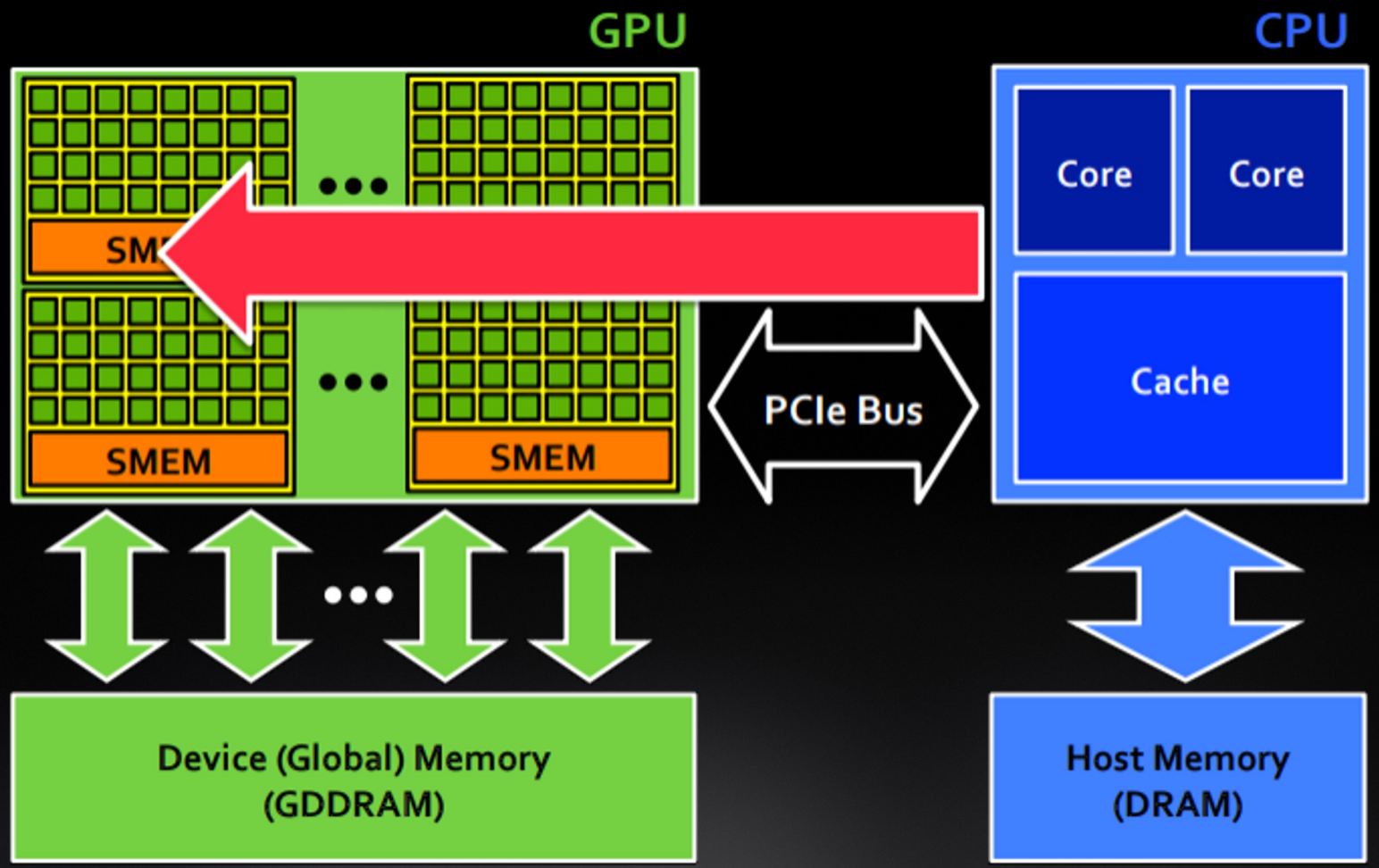
GPU Accelerator
Optimized for
Parallel Tasks



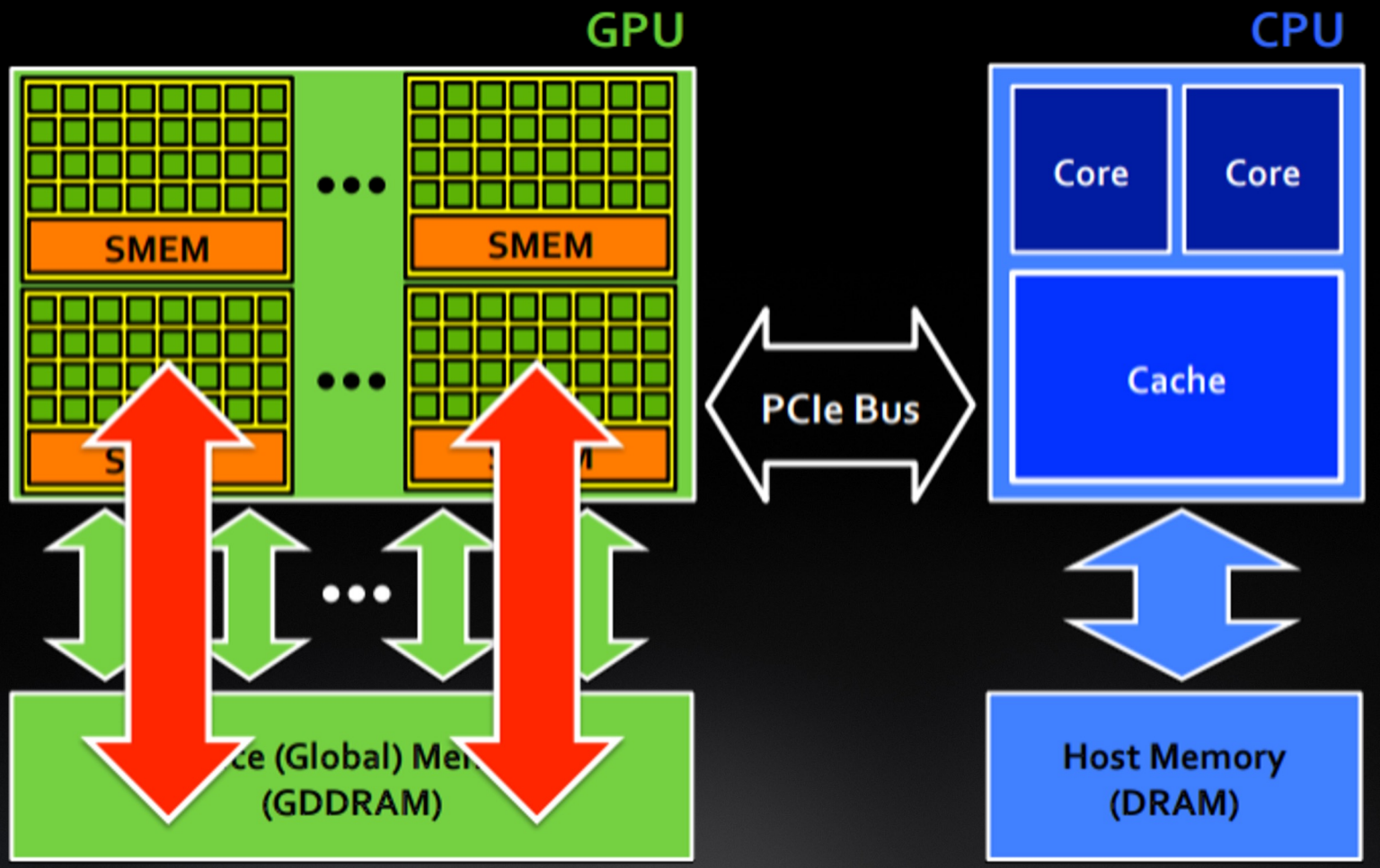
HETEROGENEOUS PROGRAMMING



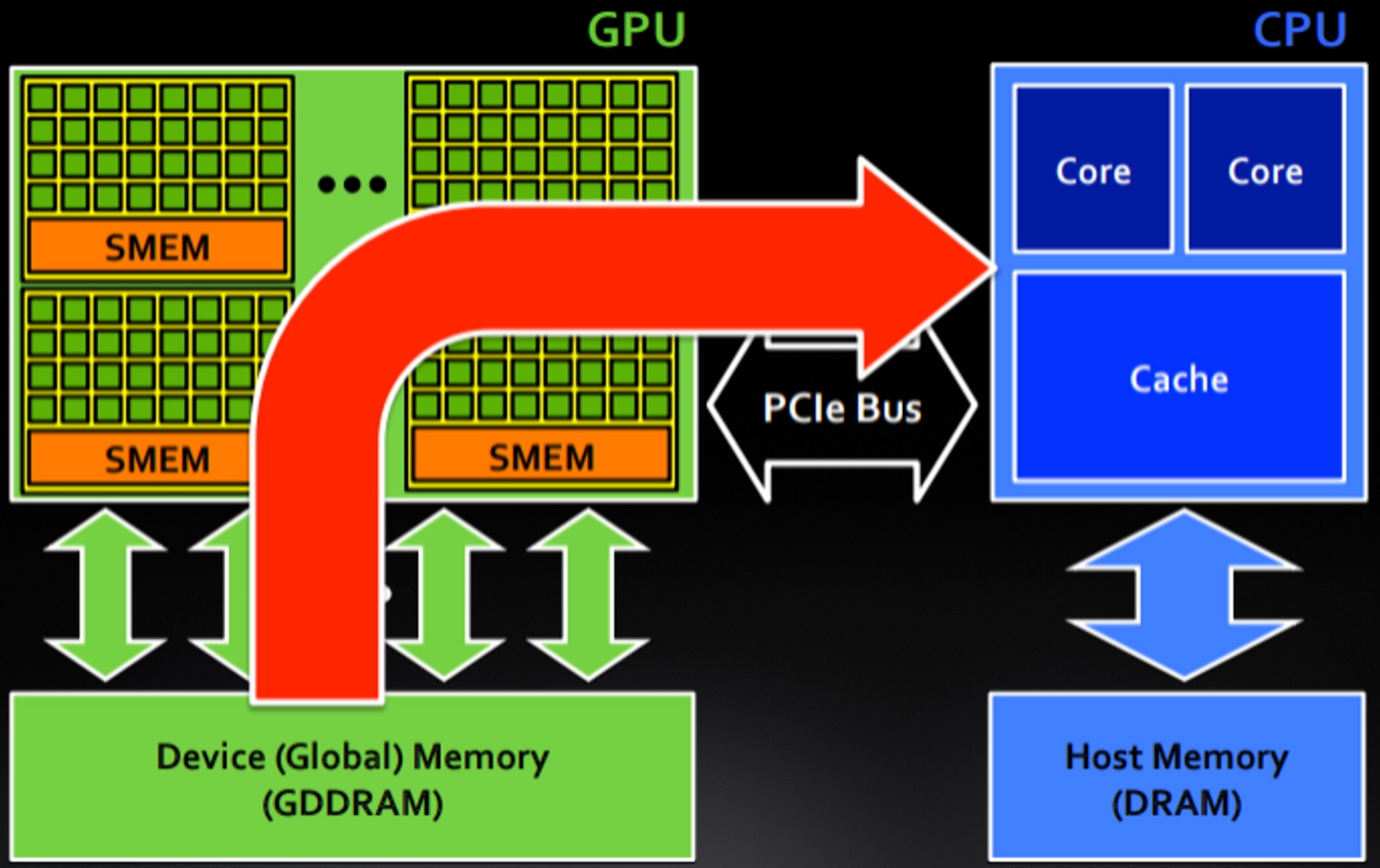
EXECUTION FLOW - H2D



EXECUTION FLOW - KERNEL LAUNCH



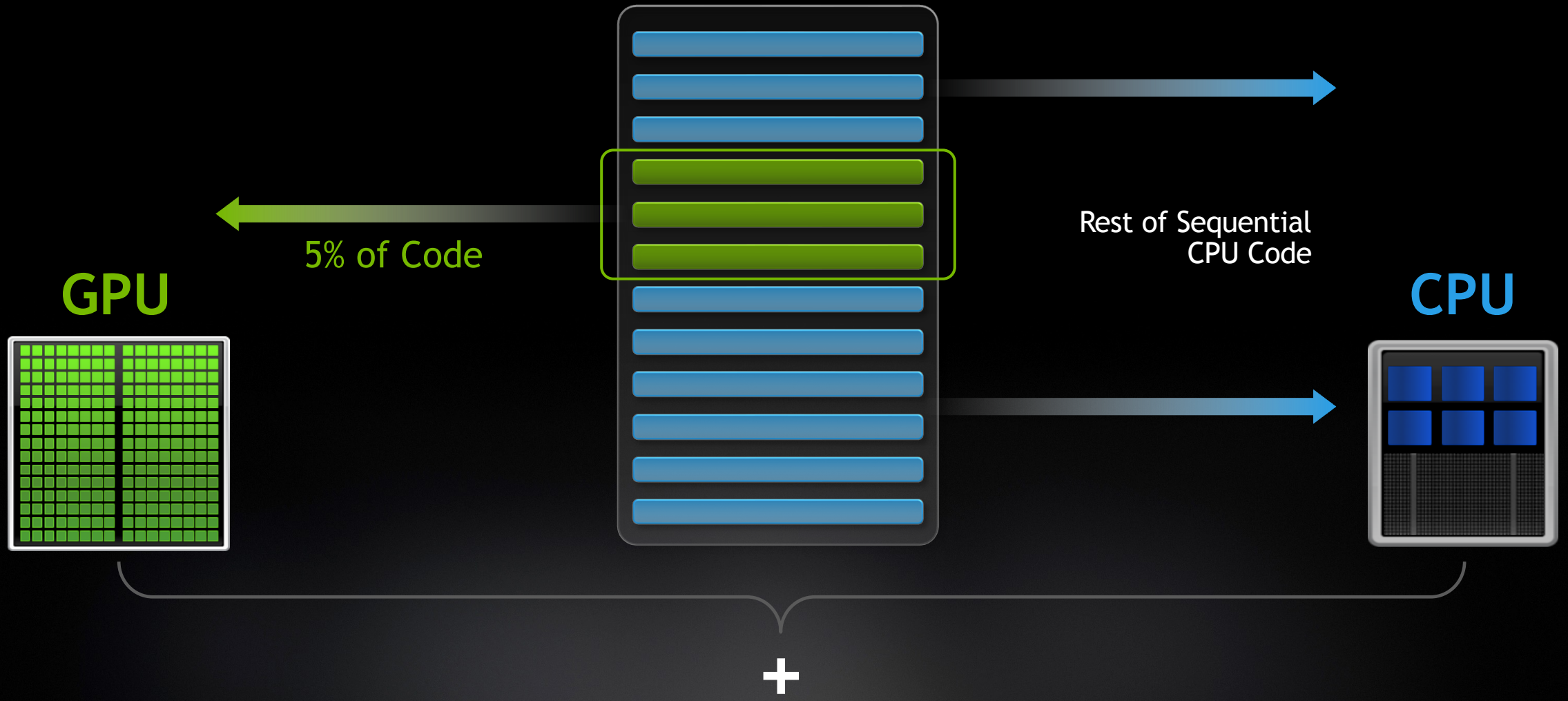
EXECUTION FLOW - D2H



MANY WAYS TO PROGRAM A GPU

(PROGRAMMING MODELS)

HOW GPU ACCELERATION WORKS



GPU PROGRAMMING IN 2020 AND BEYOND

Math Libraries | Standard Languages | Directives | CUDA

```
std::transform(par, x, x+n, y, y,
    [=](float x, float y) {
        return y + a*x;
});
```

```
do concurrent (i = 1:n)
   y(i) = y(i) + a*x(i)
enddo
```

GPU Accelerated C++ and Fortran

```
#pragma acc data copy(x,y)
{

...

std::transform(par, x, x+n, y, y,
    [=](float x, float y) {
        return y + a*x;
});

...

}
```

Incremental Performance Optimization with Directives

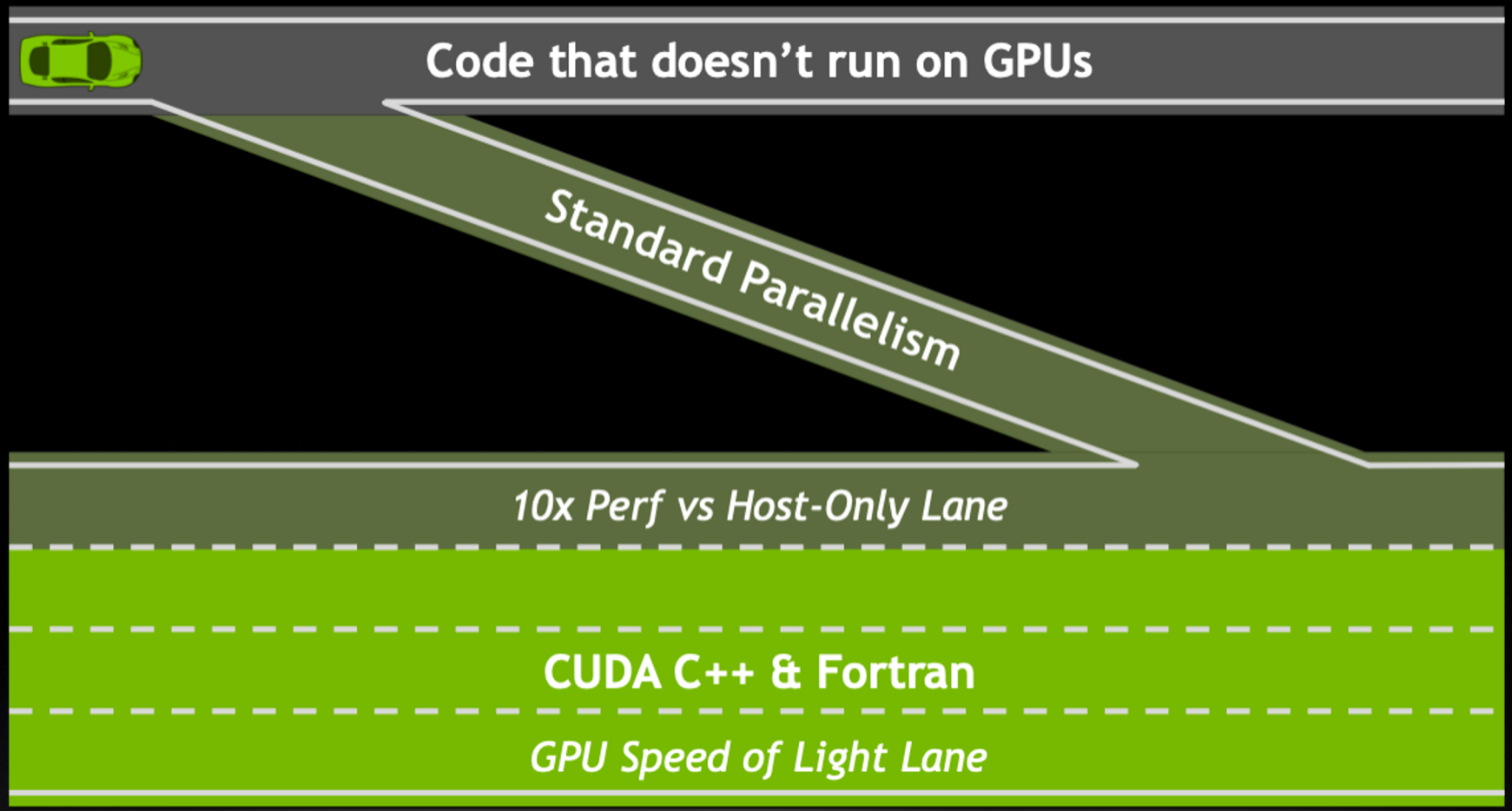
```
__global__
void saxpy(int n, float a,
           float *x, float *y) {
  int i = blockIdx.x*blockDim.x +
          threadIdx.x;
  if (i < n) y[i] += a*x[i];
}

int main(void) {
  cudaMallocManaged(&x, ...);
  cudaMallocManaged(&y, ...);
  ...
  saxpy<<<(N+255)/256,256>>>(...,x, y)
  cudaDeviceSynchronize();
  ...
}
```

Maximize GPU Performance with CUDA C++/Fortran

GPU Accelerated Math Libraries

HPC DEVELOPERS NEED A ON-RAMP



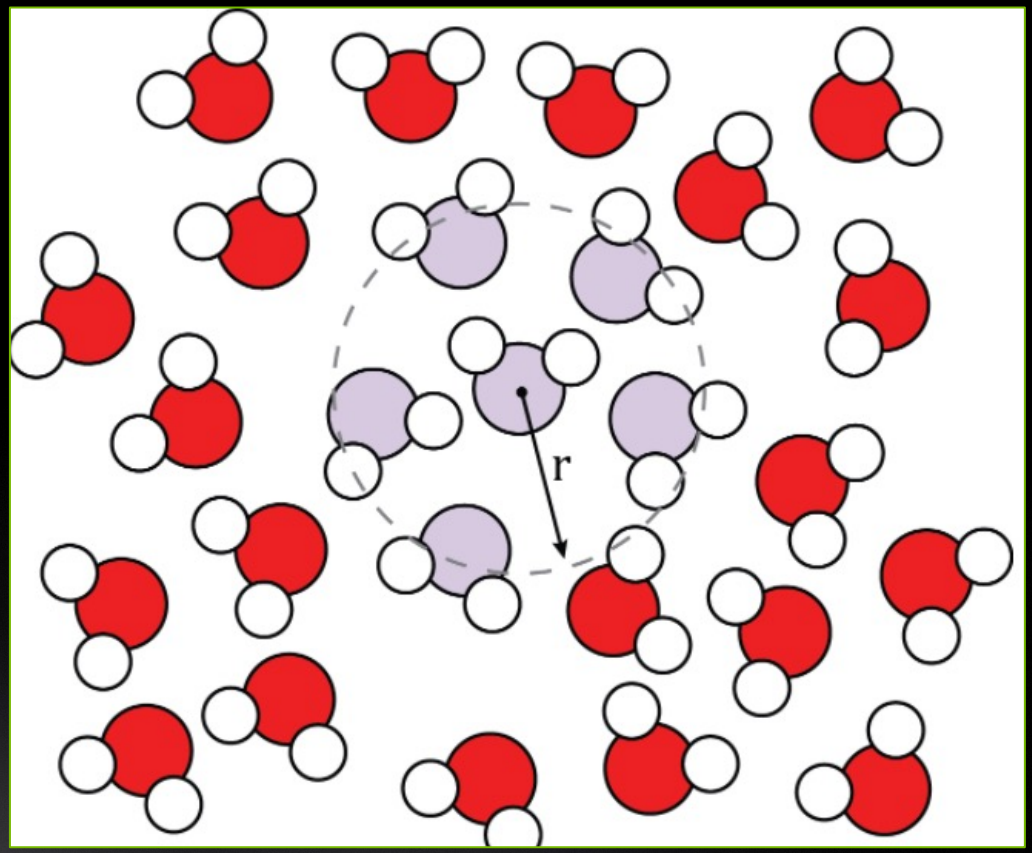
BOOTCAMP MINI-APP

APPLICATION

Molecular Simulation

RDF

The radial distribution function (RDF) denoted in equations by g(r) defines the probability of finding a particle at a distance r from another tagged particle.



RDF

Pseudo Code - C

```
for (int frame=0;frame<nconf;frame++){

      for(int id1=0;id1<numatm;id1++)
      {
            for(int id2=0;id2<numatm;id2++)
            {
                  dx=d_x[]-d_x[];
                  dy=d_y[]-d_y[];
                  dz=d_z[]-d_z[];
                  r=sqrtf(dx*dx+dy*dy+dz*dz);

                  if (r<cut) {
                        ig2=(int)(r/del);
                        d_g2[ig2] = d_g2[ig2] +1 ;
                  }
            }
      }
}
```

- Across Frames
- Find Distance
- Reduction

RDF

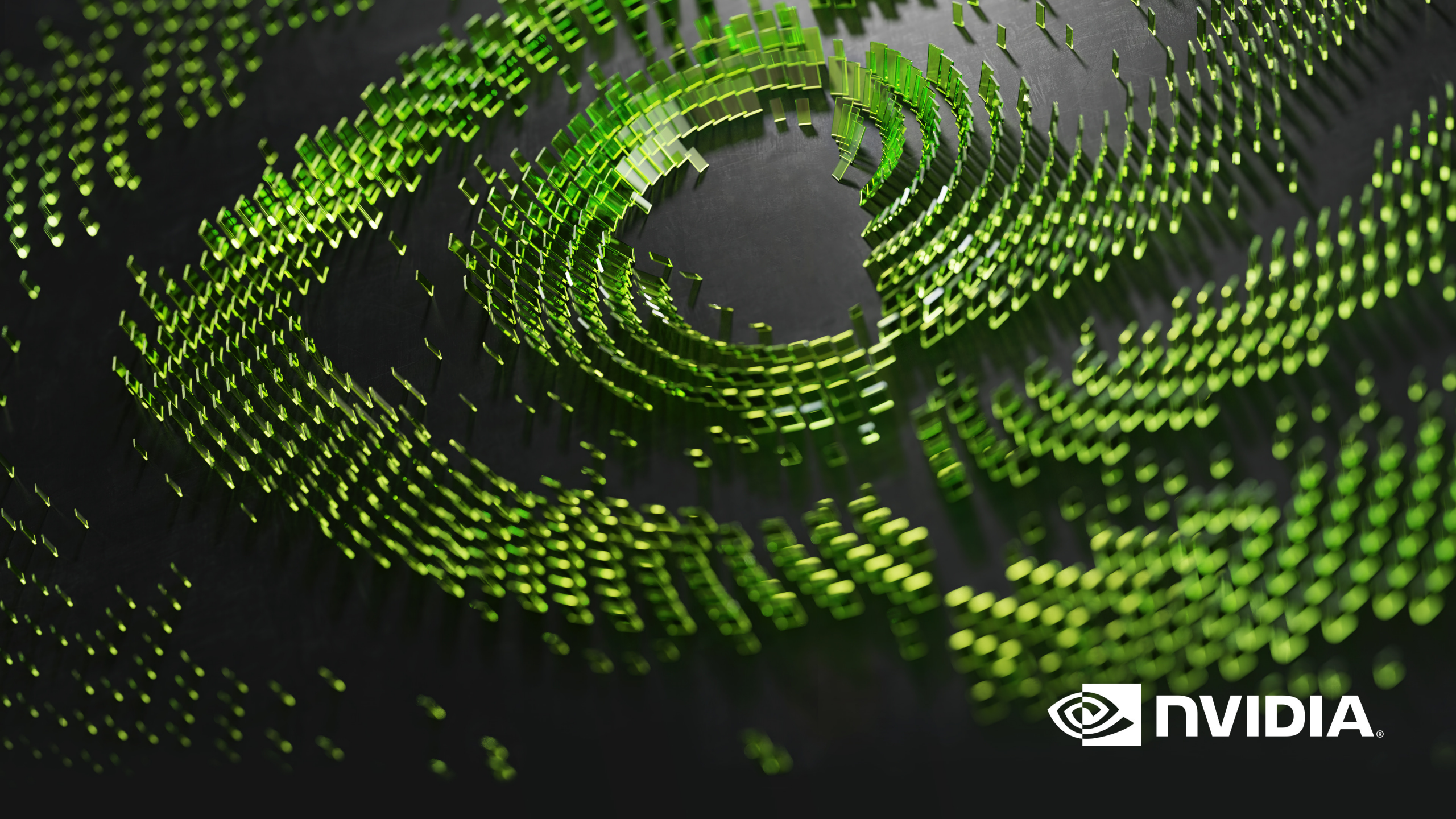
Pseudo Code - Fortran

```
do iconf=1,nframes
   if (mod(iconf,1).eq.0) print*,iconf

   do i=1,natoms
      do j=1,natoms
            dx=x(iconf,i)-x(iconf,j)
            dy=y(iconf,i)-y(iconf,j)
            dz=z(iconf,i)-z(iconf,j)

            r=dsqrt(dx**2+dy**2+dz**2)
            if(r<cut)then
                  g(ind)=g(ind)+1.0d0
            endif
      enddo
   enddo
enddo
```

- Across Frames
- Find Distance
- Reduction


NVIDIA