AMD and OpenCL

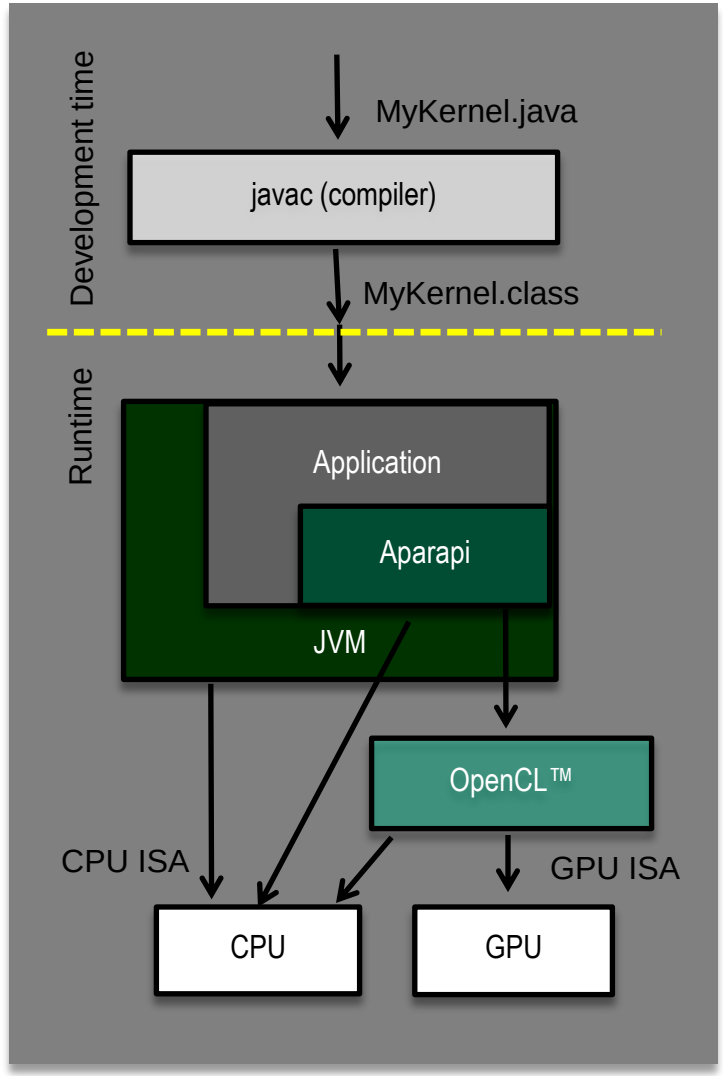
What is Aparapi?

At development time

- Aparapi offers an API for expressing data parallel workloads in Java™
  - Developer uses common Java patterns and idioms
    - extend Kernel base class and implements `run()` method
  - Java source compiled to (bytecode) using standard compiler (javac)
  - Classes packaged and deployed using traditional Java tool chain

At runtime

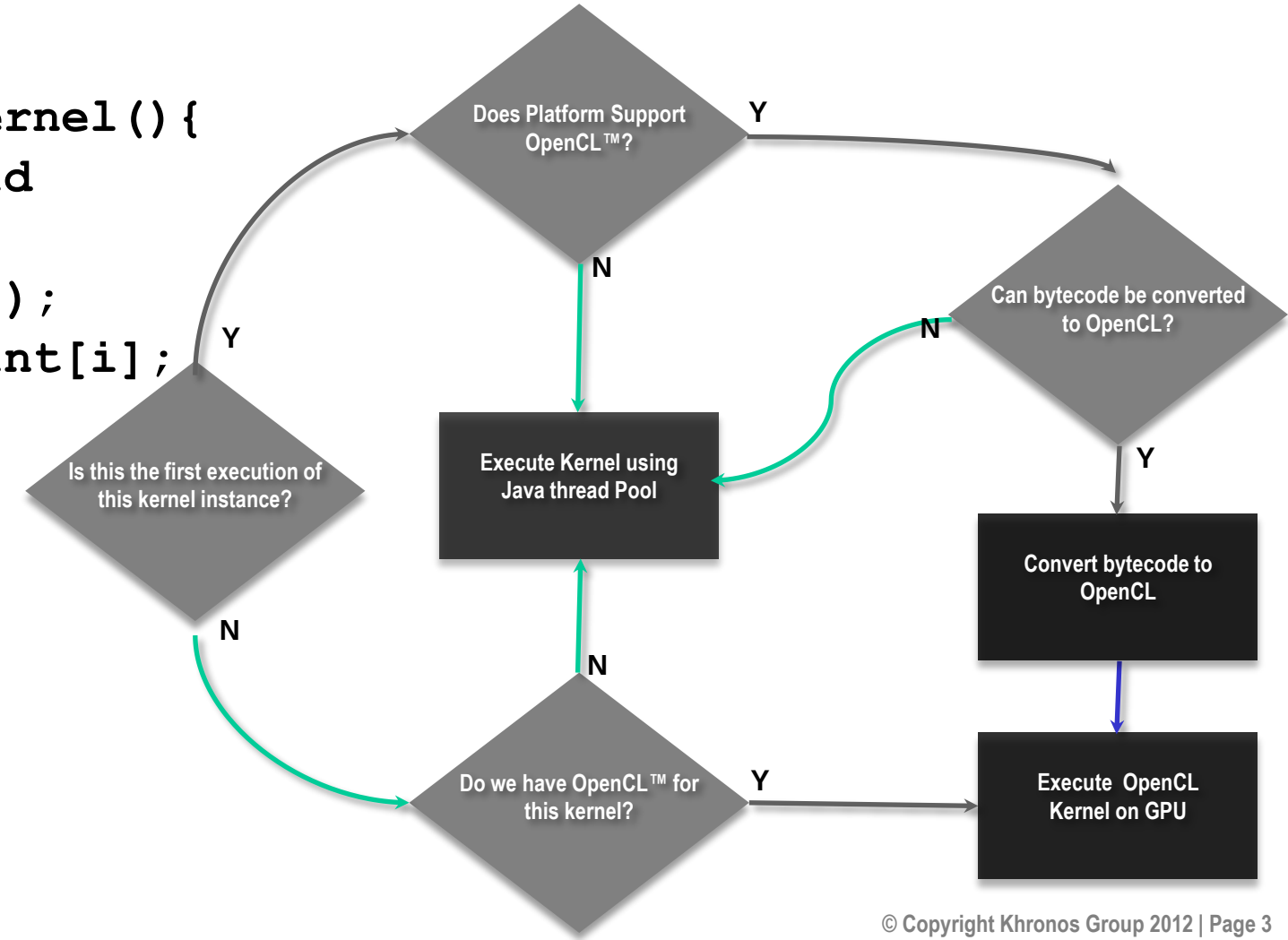
- Aparapi offers a runtime capable of converting bytecode to OpenCL™
  - For execution on GPU/APU (or any OpenCL 1.1+ capable device)
  - OR execute via a thread pool if OpenCL is not available



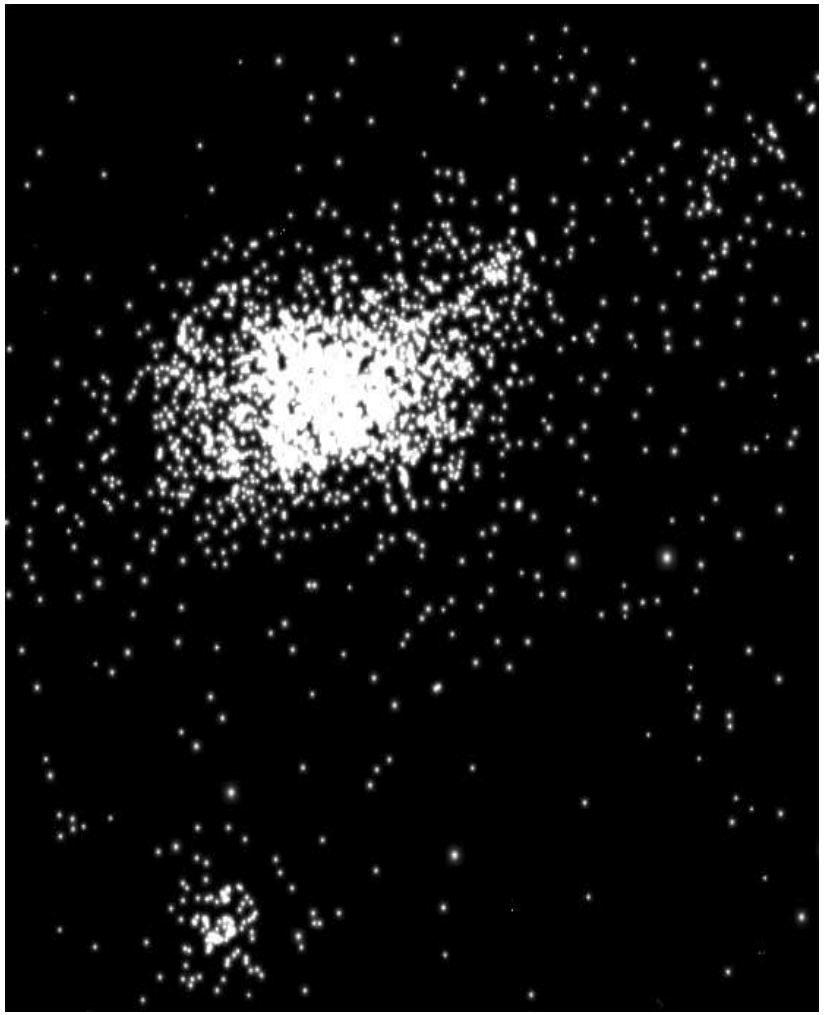
What happens when we RUN kernel.EXECUTE(<range>)?

```
Kernel kernel = new Kernel(){
   @Override public void
  run(){
     int i=getGlobalID();
     square[i]=int[i]*int[i];
   }
};

kernel.execute(size);
```



NBODY EXAMPLE



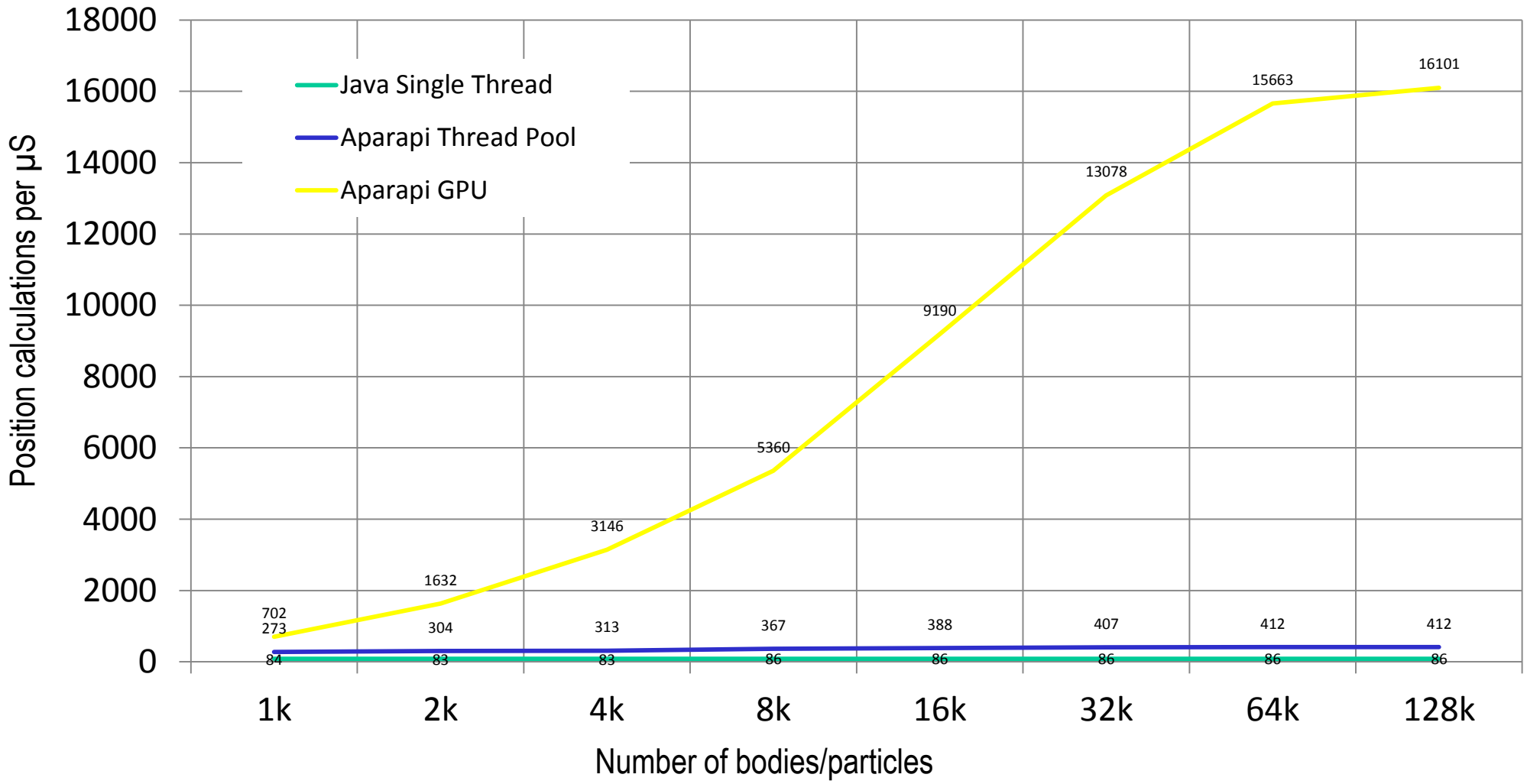
```
@Override public void run() {
    int body = getGlobalId();
    int count = bodies * 3;
    int globalId = body * 3;

    float accx = 0.f;
    float accy = 0.f;
    float accz = 0.f;

    float myPosx = xyz[globalId + 0];
    float myPosy = xyz[globalId + 1];
    float myPosz = xyz[globalId + 2];
    for (int i = 0; i < count; i += 3) {
       float dx = xyz[i + 0] - myPosx;
       float dy = xyz[i + 1] - myPosy;
       float dz = xyz[i + 2] - myPosz;
       float invDist = rsqrt((dx * dx) + (dy * dy) + (dz * dz) + espSqr);
       float s = mass * invDist * invDist * invDist;
       accx = accx + s * dx;
       accy = accy + s * dy;
       accz = accz + s * dz;
    }
    accx = accx * delT;
    accy = accy * delT;
    accz = accz * delT;
    xyz[globalId + 0] = myPosx + vxyz[globalId + 0] * delT + accx * .5f * delT;
    xyz[globalId + 1] = myPosy + vxyz[globalId + 1] * delT + accy * .5f * delT;
    xyz[globalId + 2] = myPosz + vxyz[globalId + 2] * delT + accz * .5f * delT;

    vxyz[globalId + 0] = vxyz[globalId + 0] + accx;
    vxyz[globalId + 1] = vxyz[globalId + 1] + accy;
    vxyz[globalId + 2] = vxyz[globalId + 2] + accz;
}
```


NBODY PERFORMANCE:
Calculations PER µSEC vs. Number of Bodies



Directive Based Programming Using CAPS Technology

OpenACC and OpenHMPP directives

- Incremental GPU programming
- Ease of use
- Automatic OpenCL code generation
- C and Fortran languages

Example

Image Processing using a Sobel Filter Code achieved 50x speedup by inserting 6 OpenACC directives

```
#pragma acc kernels copyin(original_image[0:rows*cols]) copyout(edge_image[0:
  {
    …
    #pragma acc loop independent
    for (iy = 1; iy < rows-1; iy++) {
      #pragma acc loop independent
      for (ix = 1; ix < cols-1; ix++) {
        int sum_x=0, sum_y=0, sum=0;
        …
        edge_image[ ix + iy * cols] = 255 - (unsigned char)(sum);
      }
    }
  }//end of OpenACC kernels region
```

www.caps-entreprise.com

Fabric Engine GPU Support

Goal: Fabric Engine's High-Performance language, KL, able to compile and run the same code on CPU and GPU

Already used LLVM in the CPU case

Needed a means to compile to LLVM-IR and a means to control the GPU at runtime

OpenCL API (not the language) was a stepping stone to control the GPU: provided all we needed to get the code and the GPU ready and fire off computation