CO405H

Computing in Space with OpenSPL

Topic 15: Porting CPU Software to DFEs

Oskar Mencer **Georgi Gaydadjiev**

Department of Computing
Imperial College London

http://www.doc.ic.ac.uk/~oskar/
http://www.doc.ic.ac.uk/~georgig/

CO405H course page: http://cc.doc.ic.ac.uk/openspl14/
WebIDE: http://openspl.doc.ic.ac.uk
OpenSPL consortium page: http://www.openspl.org

o.mencer@imperial.ac.uk g.gaydadjiev@imperial.ac.uk

Porting N-Body to DFEs

- Very large N (~90,000 particles)
- Brute force approach
- Look at options, find optimal architecture

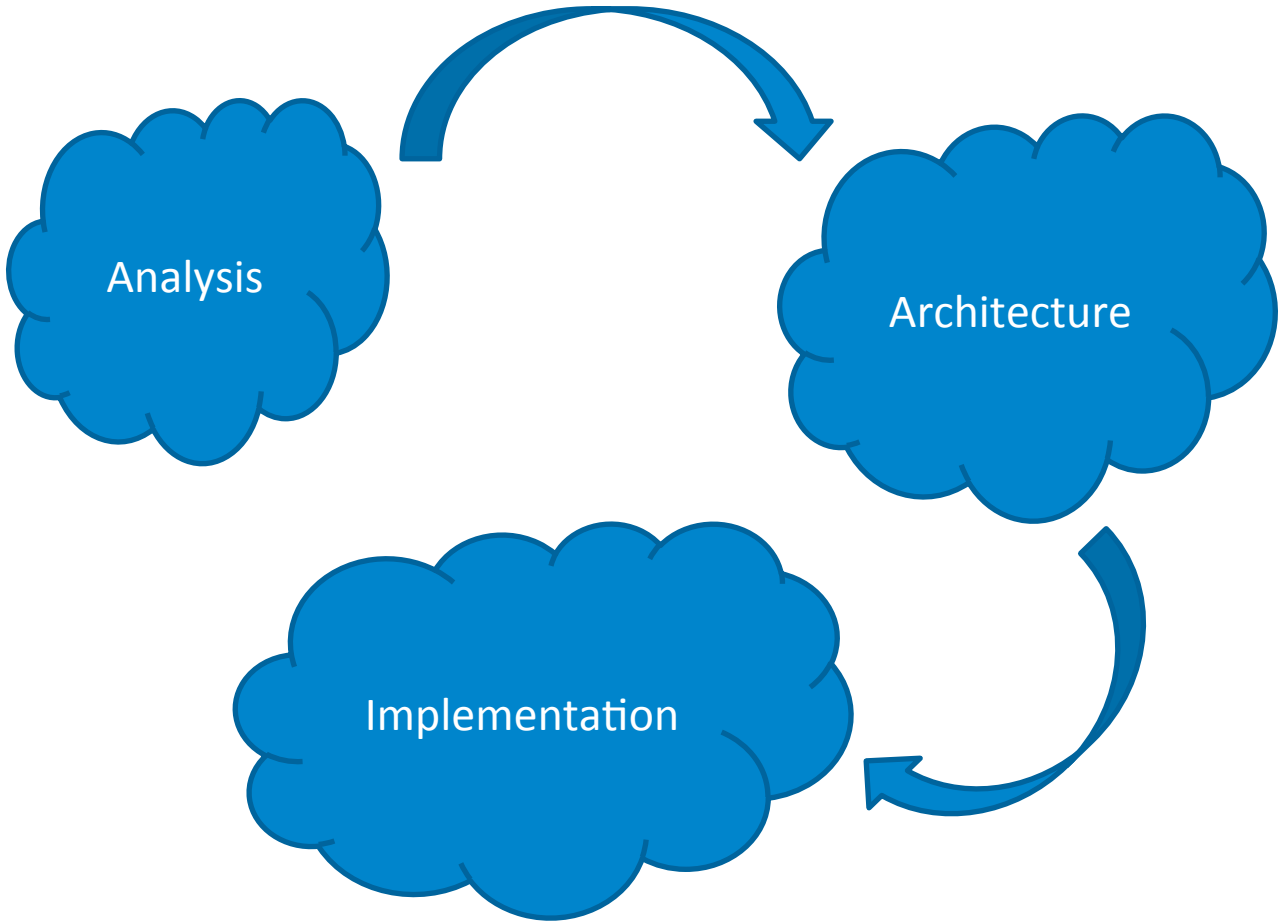
Problem to port to DFE

- Small code base

```
for (int t = 0; t < T; t++) {
    memset(a, 0, N * sizeof(coord3d_t));
    for (int q = 0; q < N; q++) {
        for (int j = 0; j < N; j++) {
                float rx = p[j].p.x - p[q].p.x;
                float ry = p[j].p.y - p[q].p.y;
                float rz = p[j].p.z - p[q].p.z;
                float dd = rx*rx + ry*ry + rz*rz + EPS;
                float d = 1/ sqrtf(dd * dd * dd);
                float s = m[j] * d;
                a[q].x += rx * s;
                a[q].y += ry * s;
                a[q].z += rz * s;
        }
    }
    for (int i = 0; i < N; i++) {
        p[i].p.x += p[i].v.x;
        p[i].p.y += p[i].v.y;
        p[i].p.z += p[i].v.z;
        p[i].v.x += a[i].x;
        p[i].v.y += a[i].y;
        p[i].v.z += a[i].z;
    }
}
```

- Very long running time: ~85 seconds per timestep, for 90,000 particles.

DFE Porting Process



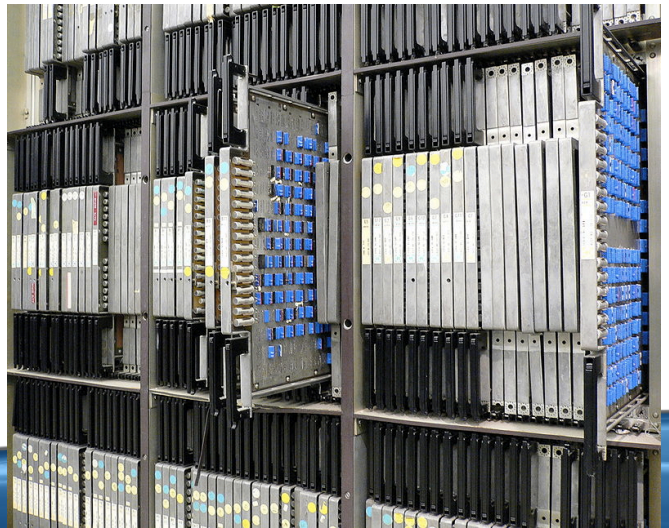
Slotnick's law (of effort)

"The parallel approach to computing does require that some original thinking be done about numerical analysis and data management in order to secure efficient use.

In an environment which has represented the absence of the need to think as the highest virtue this is a decided disadvantage."

-Daniel Slotnick,

Chief Architect, Illiac IV



DFE Porting Process Overview

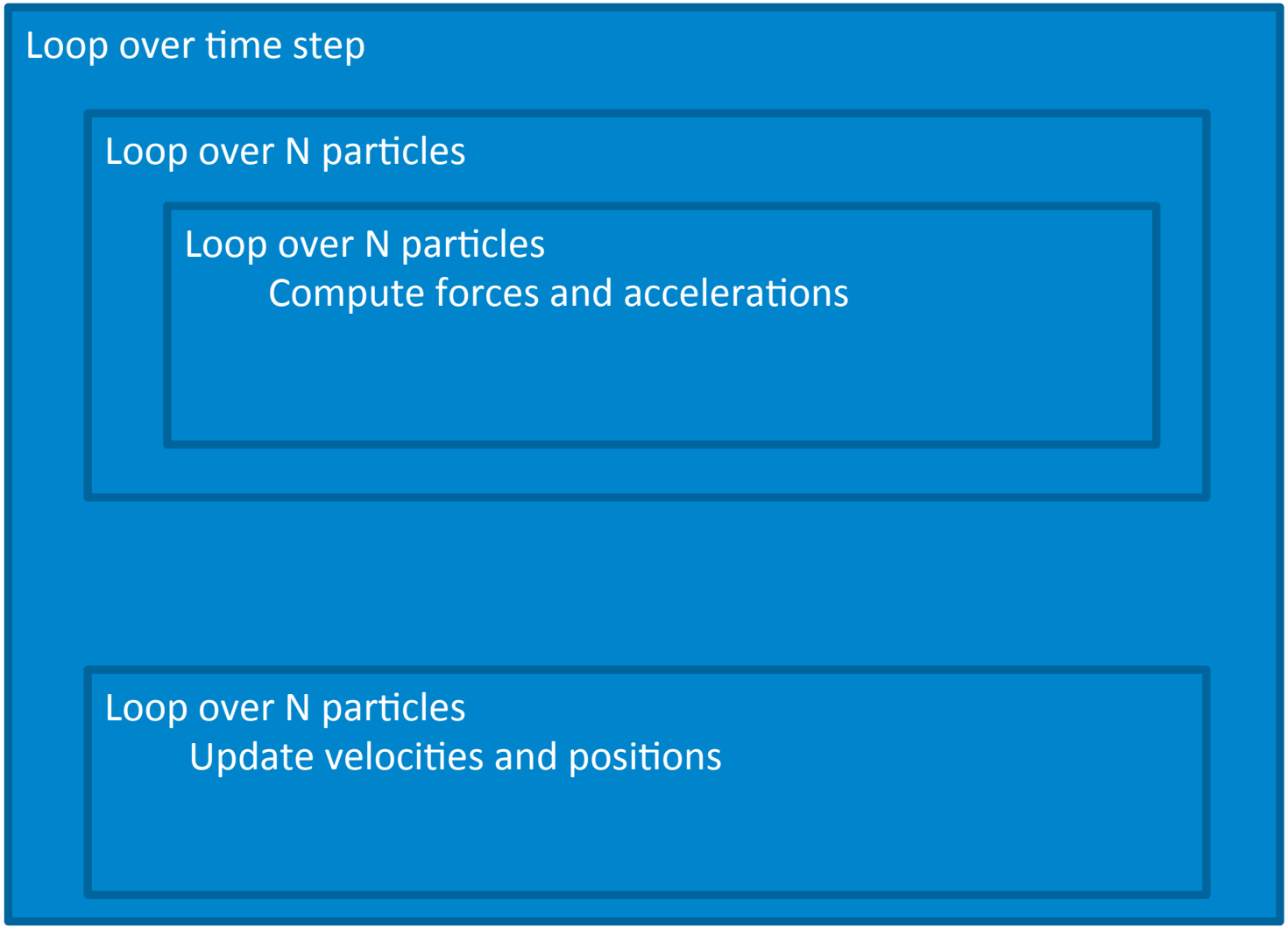
- **Step 1: Analyse Code**
  - Profile code, measure time taken
  - Measure memory requirements and working set size
  - Understand numerical requirements
- **Step 2: Architect Solution**
  - Evaluate and model partitioning options
  - Estimate speedup
- **Step 3: Implementation**
  - Transform code into partitioned architecture
  - Implement C models
  - Compile DFE (.max file)
  - Optimise and Achieve Speedup

Analysis: Step 1 – Dynamic Analysis

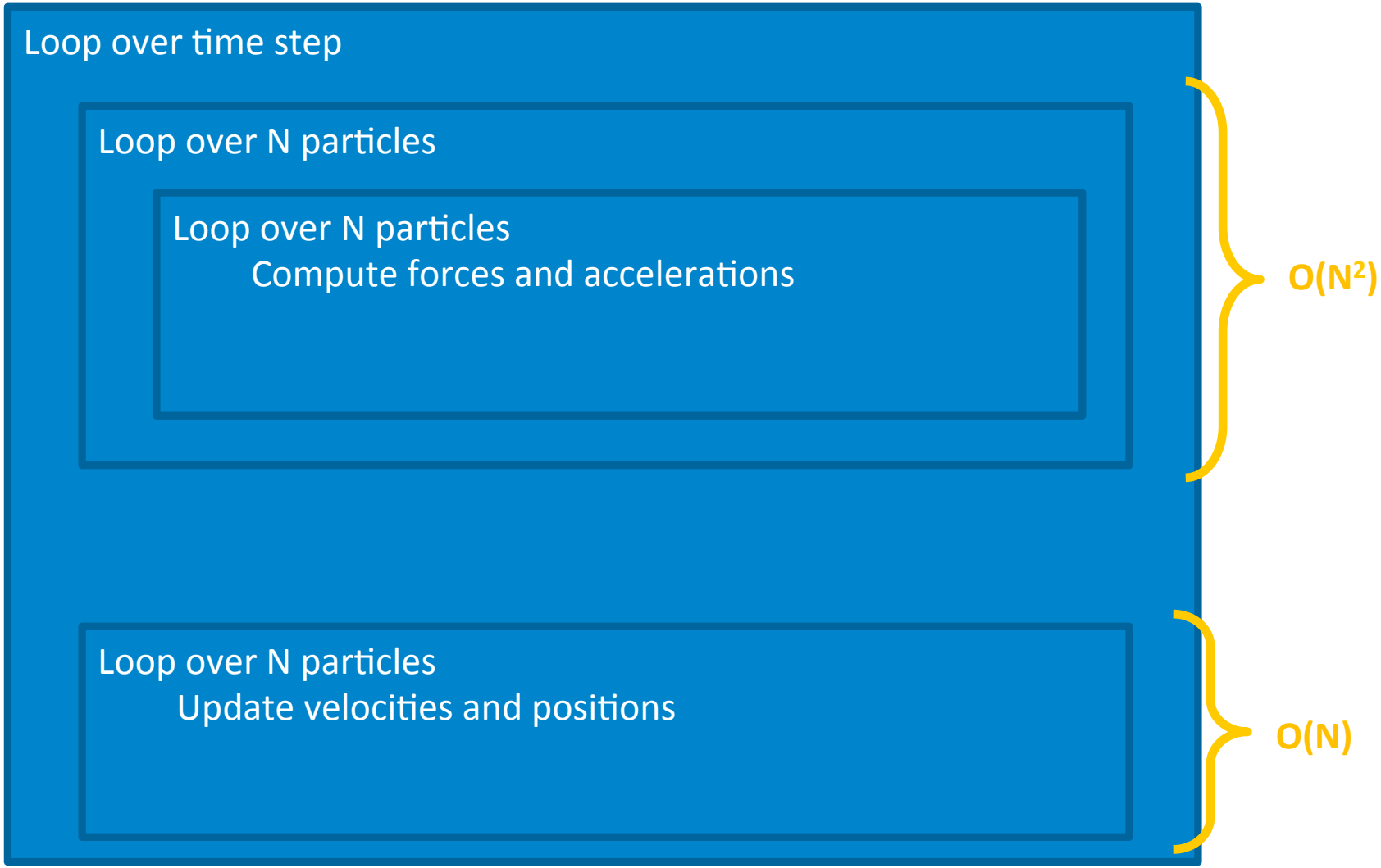
Aim: Have a complete map of all computation and dataflow, and timings for each block of computation.

- Find out where the computation is happening (Oprofile can help) and where the data is going
- Identify major loops / draw loop graph
- Measure time spent inside major loops

Analysis: Step 1 – Dynamic Analysis



Analysis: Step 1 – Dynamic Analysis

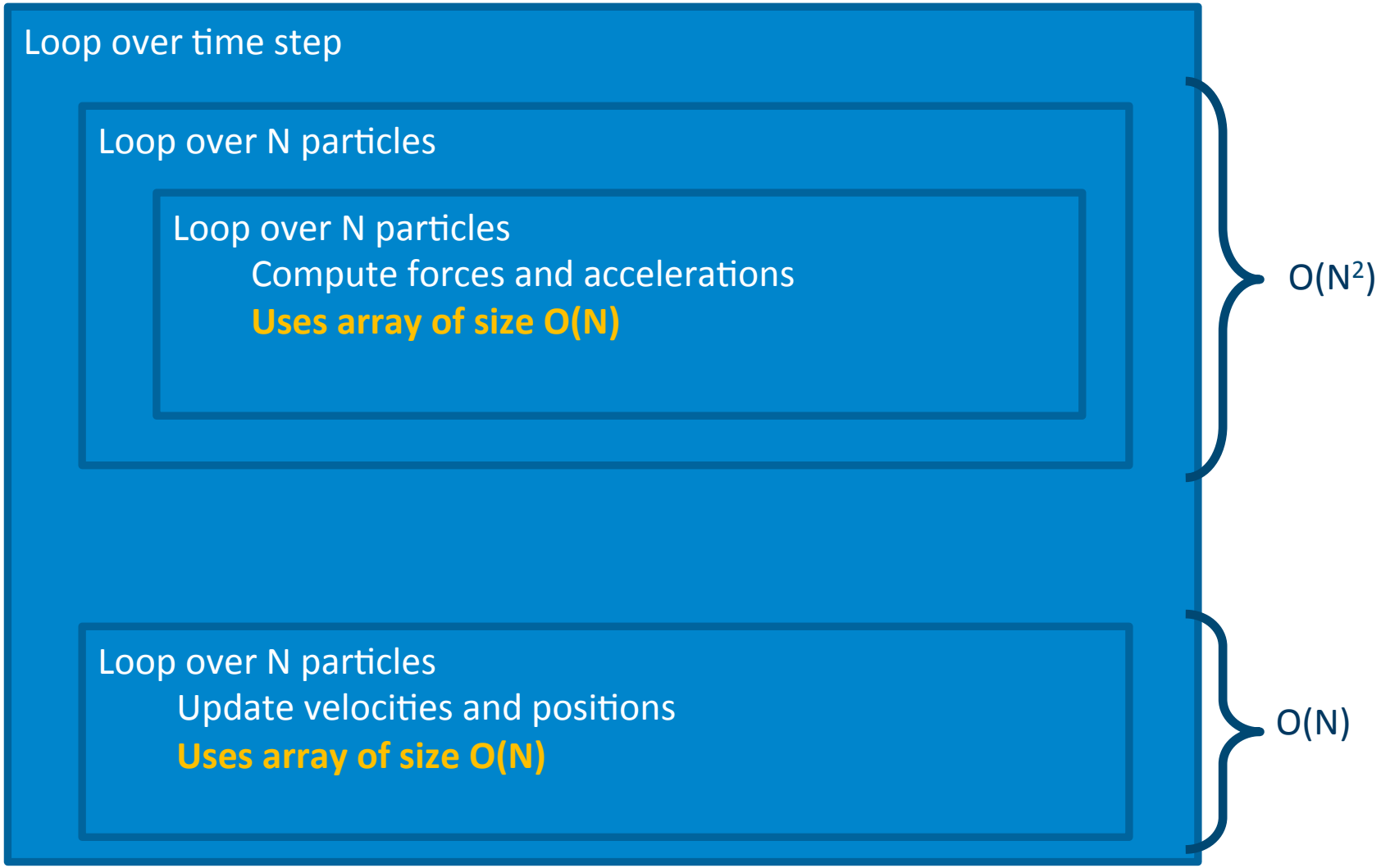


Analysis: Step 2 – Static Analysis

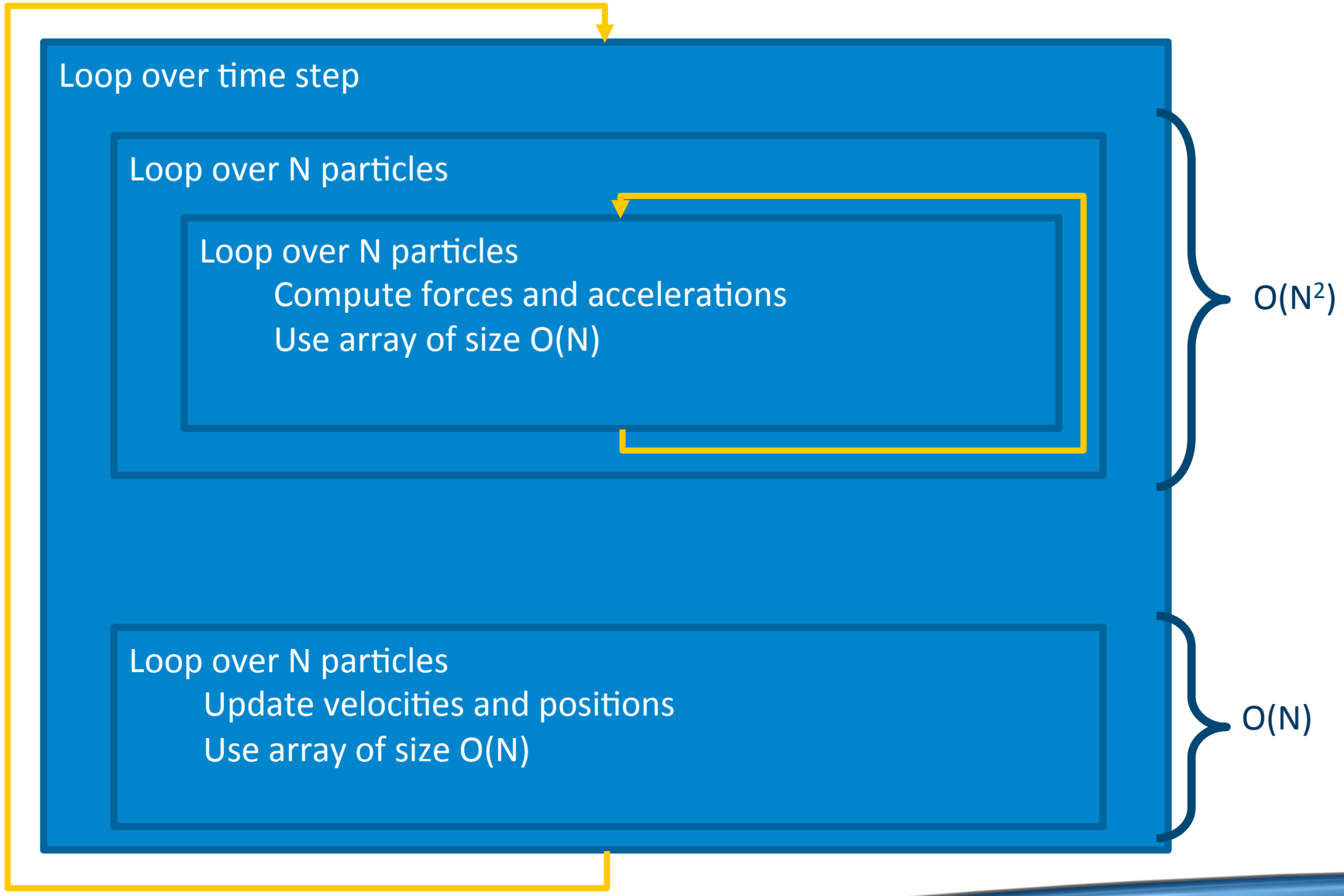
Aim: Understand amount of data being moved around and amount of compute to perform on it

- Analyse the data flow between the critical loops.
  - Examine what data structures are being created.
  - Identify which loops are going to work with very large arrays.
- Analyse computation inside the critical loops.
  - Count the number of floating point operations per data point
  - Analyse loop dependencies
- Understand the mathematical algorithms being used.
  - Relationship between input and runtime, memory use.
  - Understand precision requirements of each part of the algorithm

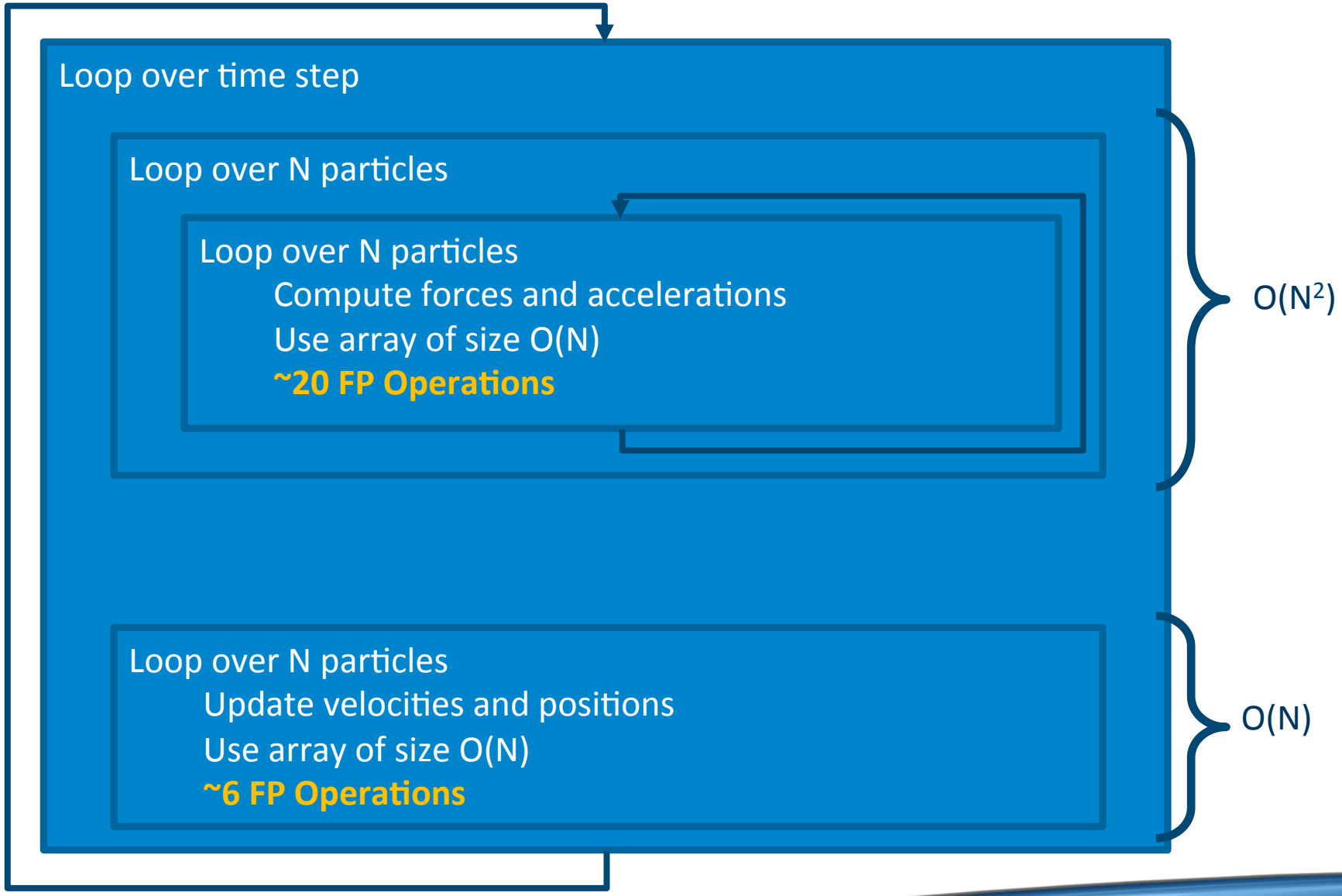
Analysis: Step 2 – Static Analysis



Analysis: Step 2 – Static Analysis



Analysis: Step 2 – Static Analysis



Analysis: Step 3 – Data Access plan

Aim: Consider various architecture choices and understand the pros and cons of each choice

- Examine volume of data flowing through algorithm.
  - How large is the working set, i.e. does it need to be stored in LMEM or FMEM?
  - Is data access pattern known statically or calculated dynamically?
  - How much computation would be done with each loaded data value?
  - Consider the ratio of Computation to Communication!

Computation-to-Communication

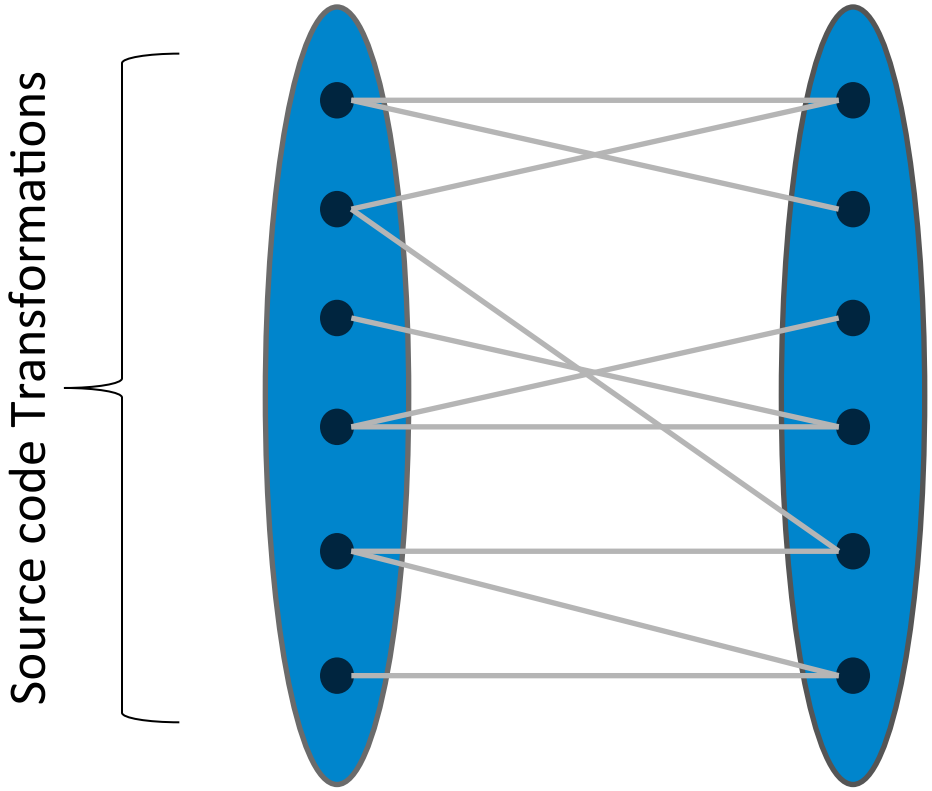
- LMEM access and CPU⇔DFE transfers are transactions
- It's like a trip to NY: you need to justify the flight by visiting enough sights to make the trip worthwhile

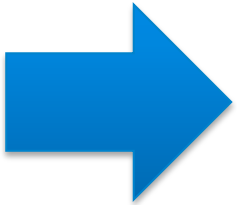
Define Your Options: move data first, then move compute

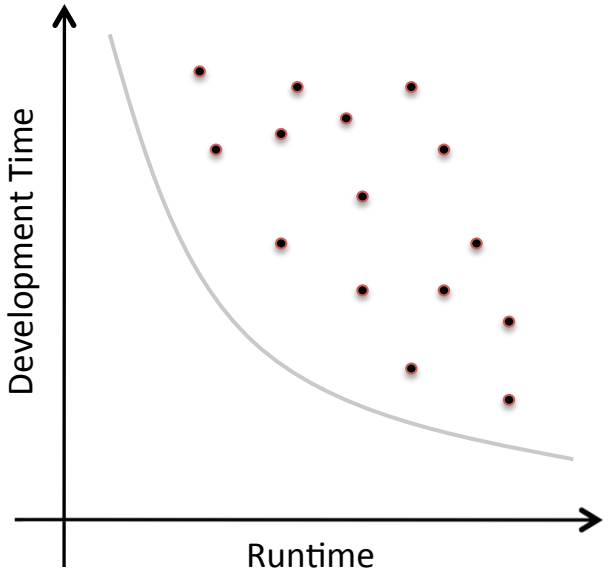
Data Access Plans

Code Partitioning

Pareto Optimal Options







Try to minimise runtime and development time, while maximising flexibility and precision.

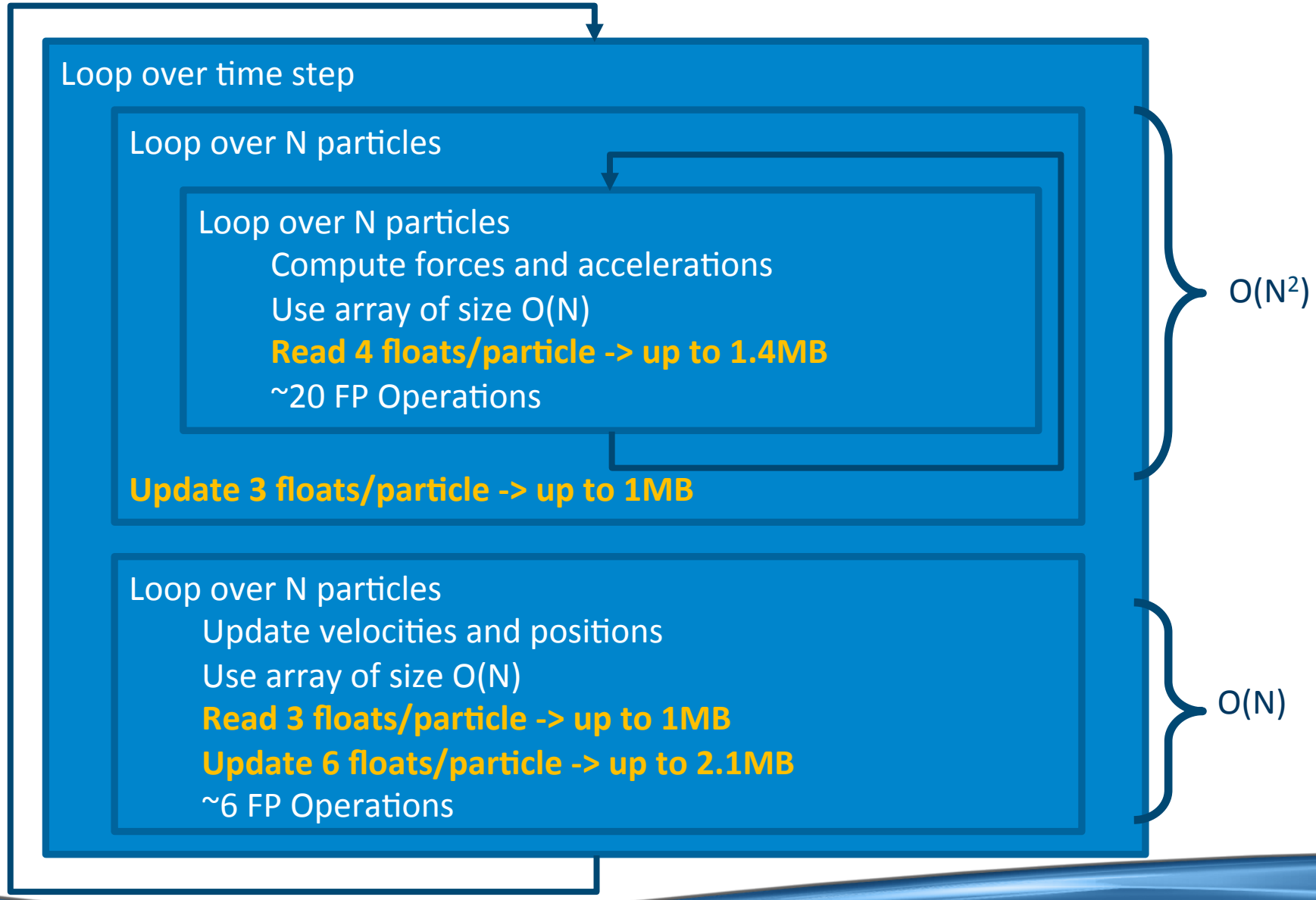
Analysis: Step 3 – Architecture Options

- Examine volume of data flowing through algorithm.
  - How large is the working set,
    i.e. does it need to be stored in LMEM or FMEM?

On a MAX3 card, you have around 4.5MB of available ultra fast access (>10TB/s) of storage in FMEM*. If you need more than that, then you will have to use LMEM which offers 12GB, 24GB, 48GB or 96GB of storage per DFE.

* Some of this FMEM will be used by MaxCompiler for automatic buffering (for example for scheduling, or in FIFOs between Kernels). How much, varies widely from one design to another.

Analysis: Step 3 – DFE Architecture Options



Analysis: Step 3 – DFE Architecture Options

- Examine volume of data flowing through algorithm.
  - Is data access pattern known statically?

    If the pattern is static then you can either use one of the command generators provided (LINEAR1D, ...) or generate commands on the CPU and stream them in.
  - Is data access pattern computed dynamically?

    If the address of the data you need to read or write needs to be computed on the Dataflow Engine, then your access pattern is dynamic and you will have to generate the LMEM command inside a Kernel.

Analysis: Step 3 – DFE Architecture Options

- For N-Body problem, access pattern is static and linear 1D

```
for (int q = 0; q < N; q++) {
    for (int j = 0; j < N; j++) {
        ...
    }
}
```

Analysis: Step 3 – DFE Architecture Options

- For N-Body problem, access pattern is static and linear 1D

```
for (int q = 0; q < N; q++) {
    for (int j = 0; j < N; j++) {
        ...
    }
}
```

Analysis: Step 3 – Architecture Options

- For N-Body problem, access pattern is static and linear 1D

```
for (int q = 0; q < N; q++) {
    for (int j = 0; j < N; j++) {
        ...
    }
}
```

Analysis: Step 3 – Architecture Options

- For N-Body problem, access pattern is static and linear 1D

```
for (int q = 0; q < N; q++) {
    for (int j = 0; j < N; j++) {
        ...
    }
}
```

Analysis: Step 3 – Architecture Options

- For N-Body problem, access pattern is static and linear 1D

```
for (int q = 0; q < N; q++) {
    for (int j = 0; j < N; j++) {
        ...
    }
}
```

Analysis: Step 3 – Architecture Options

- For N-Body problem, access pattern is static and linear 1D

```
for (int q = 0; q < N; q++) {
    for (int j = 0; j < N; j++) {
        ...
    }
}
```

Analysis: Step 3 – Architecture Options

- Examine volume of data flowing through algorithm.
  - How much computation would be done with each loaded data value?

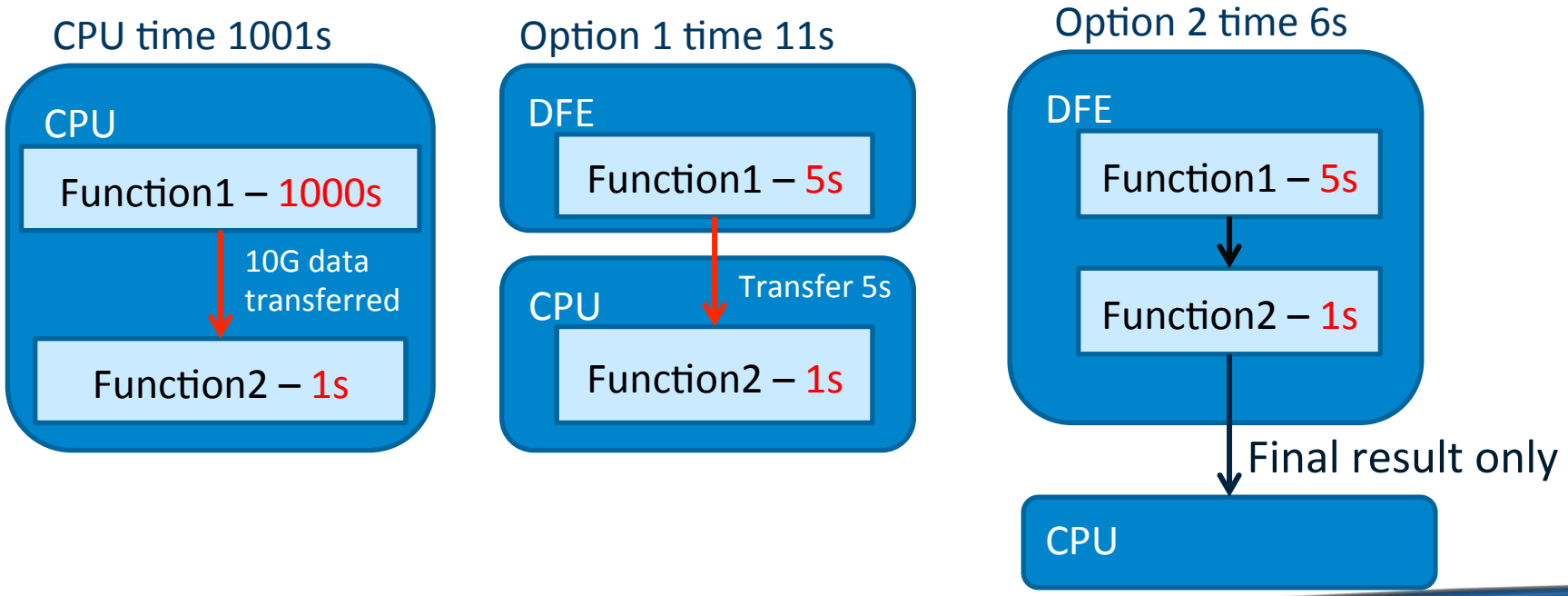
By carefully choosing your memory access pattern, you can increase data reuse and decrease memory bandwidth requirement. LMEM has a limit which depends on the platform and its frequency. For some DFE at 350MHz this is about 33.5 GB/s.

With complex access patterns and many streams, actual bandwidth could be different.

Analysis: Step 3 – Architecture Options

- What needs to be on the DFE, and what can stay on the CPU?
  - How many functions require access to the largest arrays?
  - Do the functions that use the large arrays also have long runtime?

  Moving the bulk of the compute to the DFE might not be the right answer.



Analysis: Step 3 – Architecture Options

- Examine what data can be pre-computed.
  - Which functions actually need to be run inside the loops?

  Consider the following loops:

```
for i = 0..99 do
    double a = cos(i*2*PI/100)
    for j = 0..9999
        // do some compute
```

Assume that we wish to put these loops onto a DFE and that each iteration of j takes one cycle. Putting the computation of `a` onto the DFE as well means that we will be using hardware resources to compute a cosine that is needed only once every 10,000 cycles. This is very wasteful. Instead, it would be better to compute the 100 different values of `a` and store them into an FMEM on the DFE.

Analysis: Step 3 – Multiple Kernels

NOTE: There is a high overhead(*) to create a new kernel (each running in their own clock domain), so keep the number of kernels low.

- Your design can have one or more kernels. How do you decide how many kernels to build:

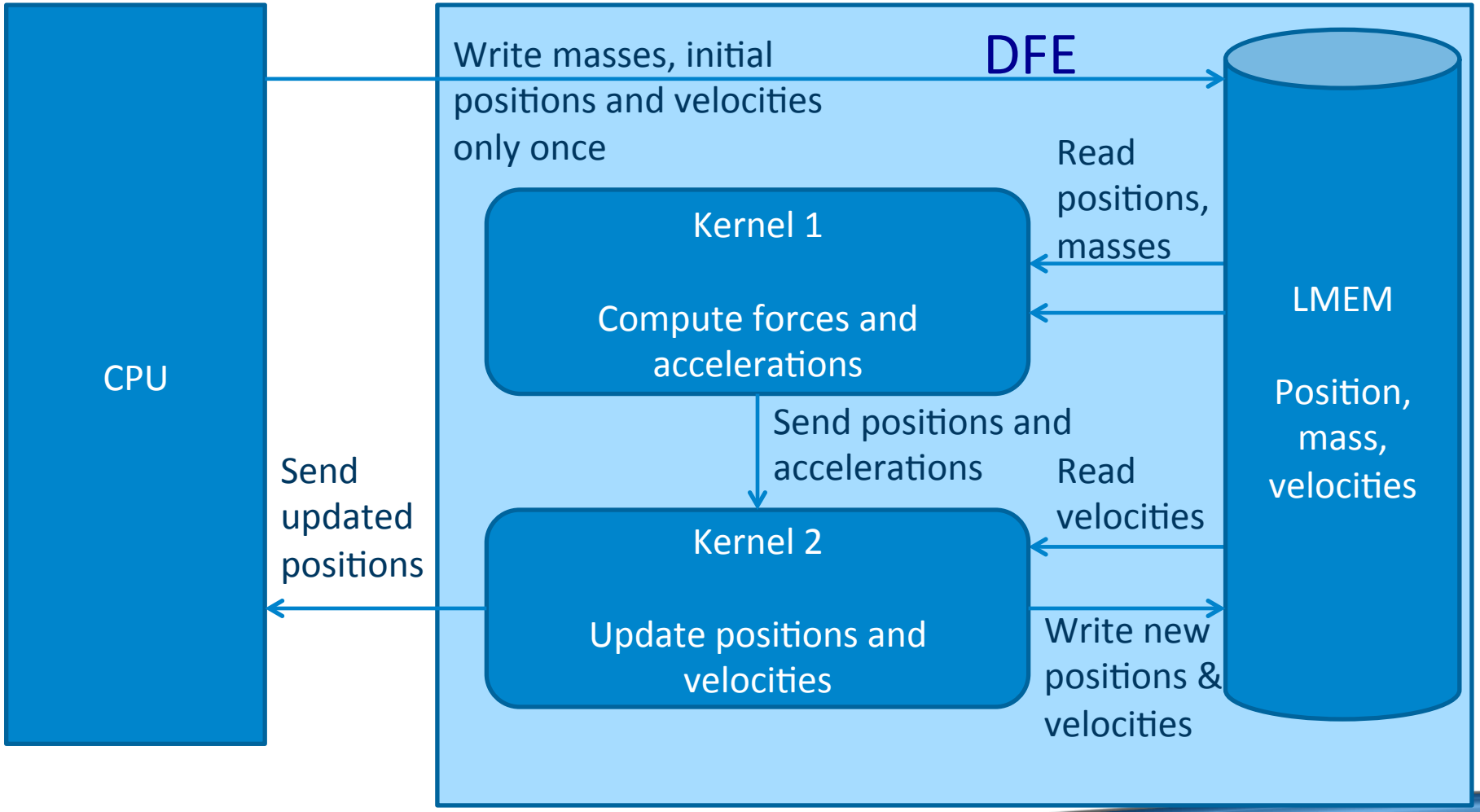
1. Your design may have multiple passes. Each pass could have a separate kernel.

2. You may be able to partition your design into pieces with dynamic and/or different input and output bandwidth requirements

(*) A Maxeler architecture is a Globally Asynchronous Locally Synchronous (GALS) architecture

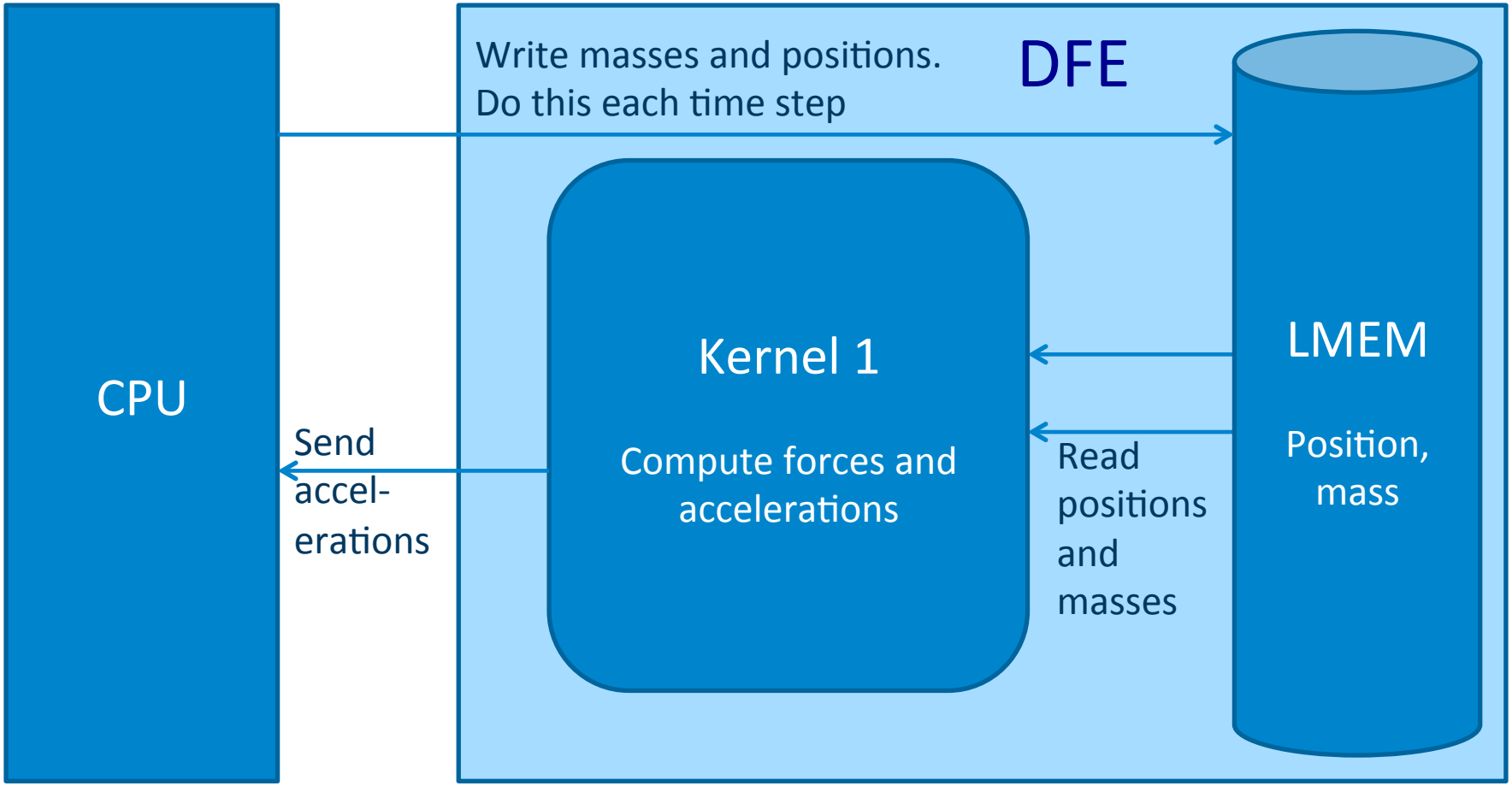
Analysis: Step 3 – Architecture Options

- NBody Option 1



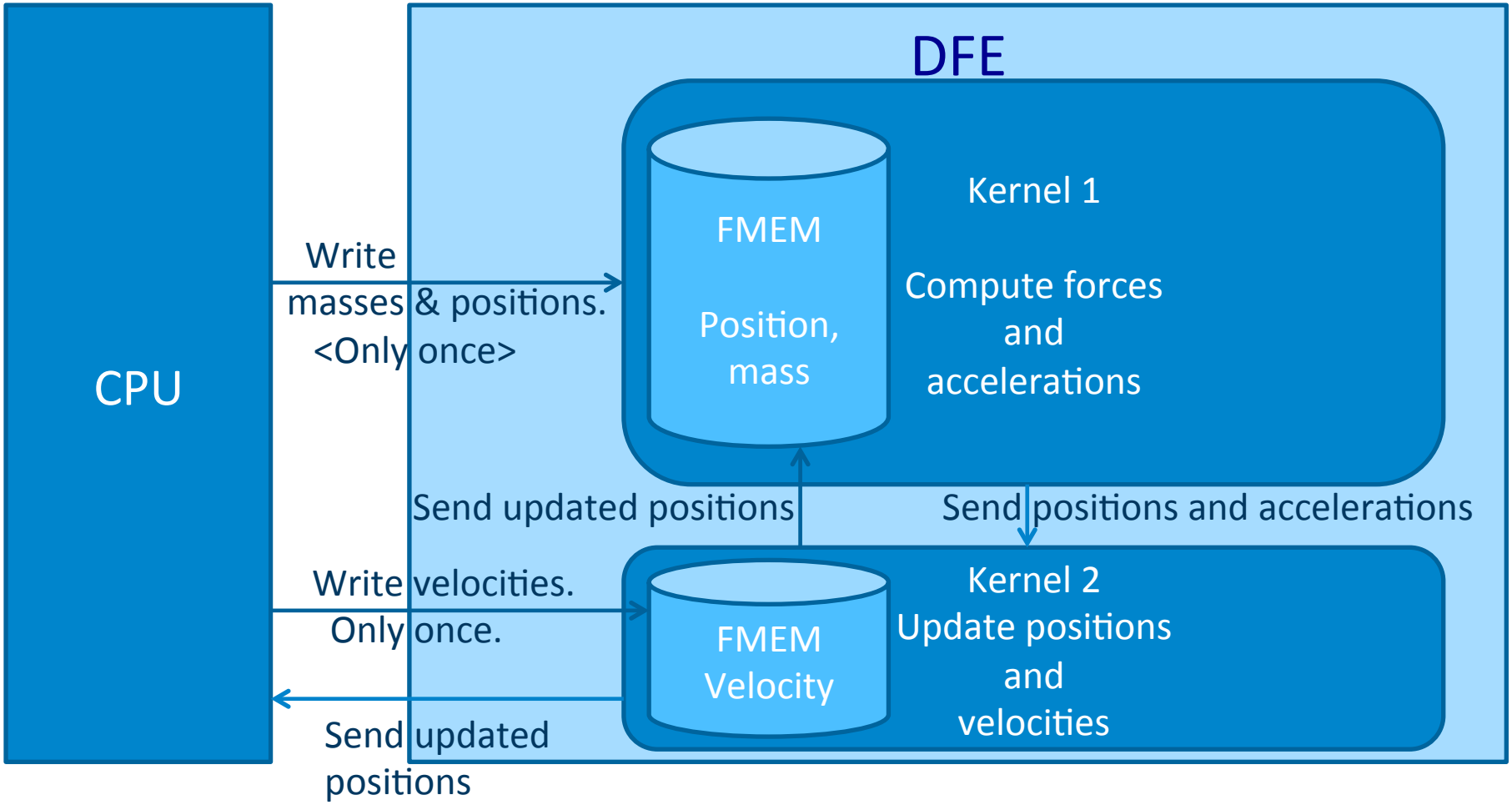
Analysis: Step 3 – Architecture Options

- NBody Option 2



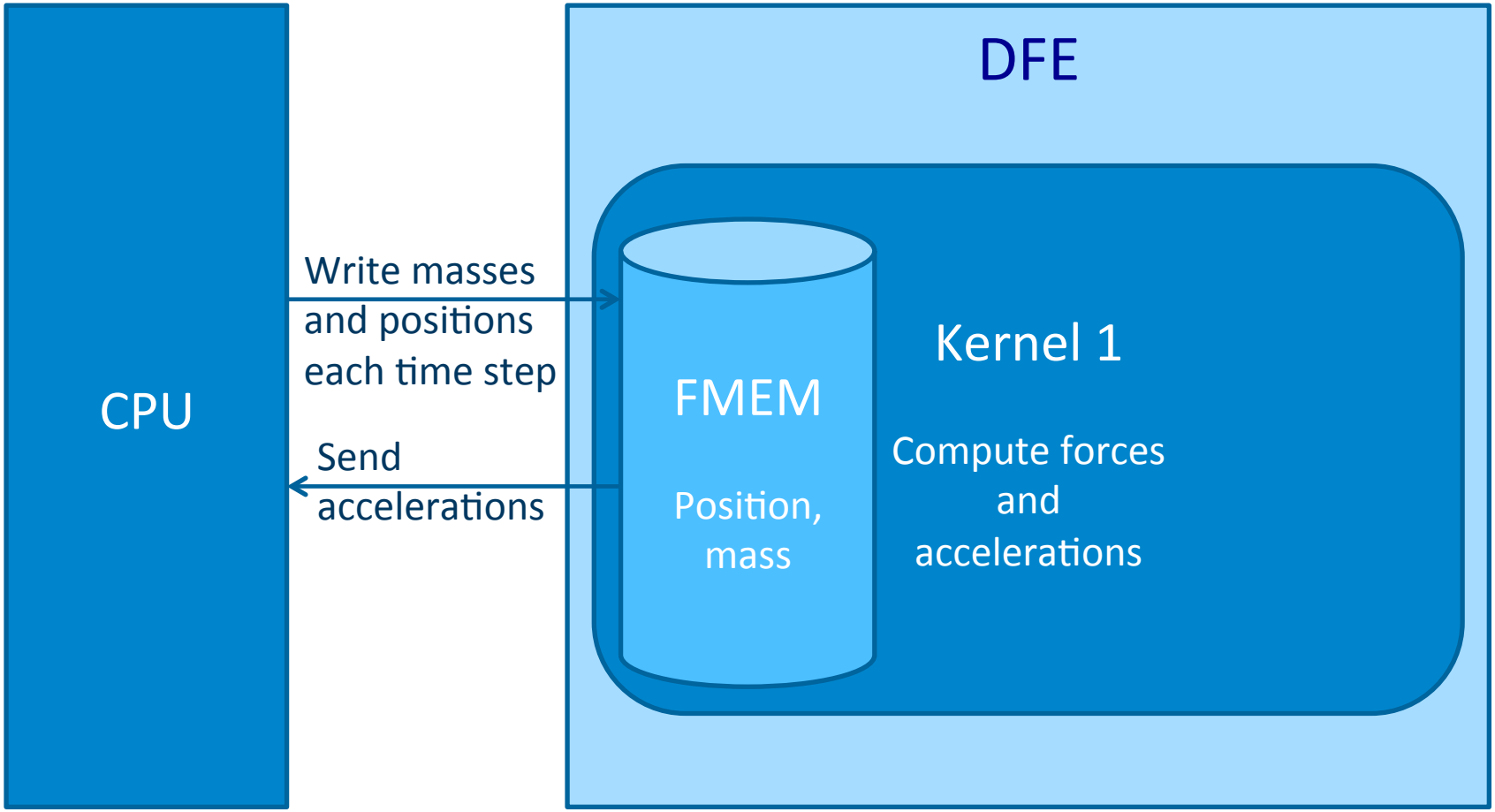
Analysis: Step 3 – Architecture Options

- NBody Option 3



Analysis: Step 3 – Architecture Options

- NBody Option 4



Conclusions – Porting CPU Software to DFEs

- Look at Options
- Process: Analysis, Architecture, Implementation
- Carefully minimise the number of kernels needed.
- First move data from CPU to DFE and then consider which computations need to move with the data