CO405H

Computing in Space with OpenSPL
Topic 4: DataFlow Engines (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

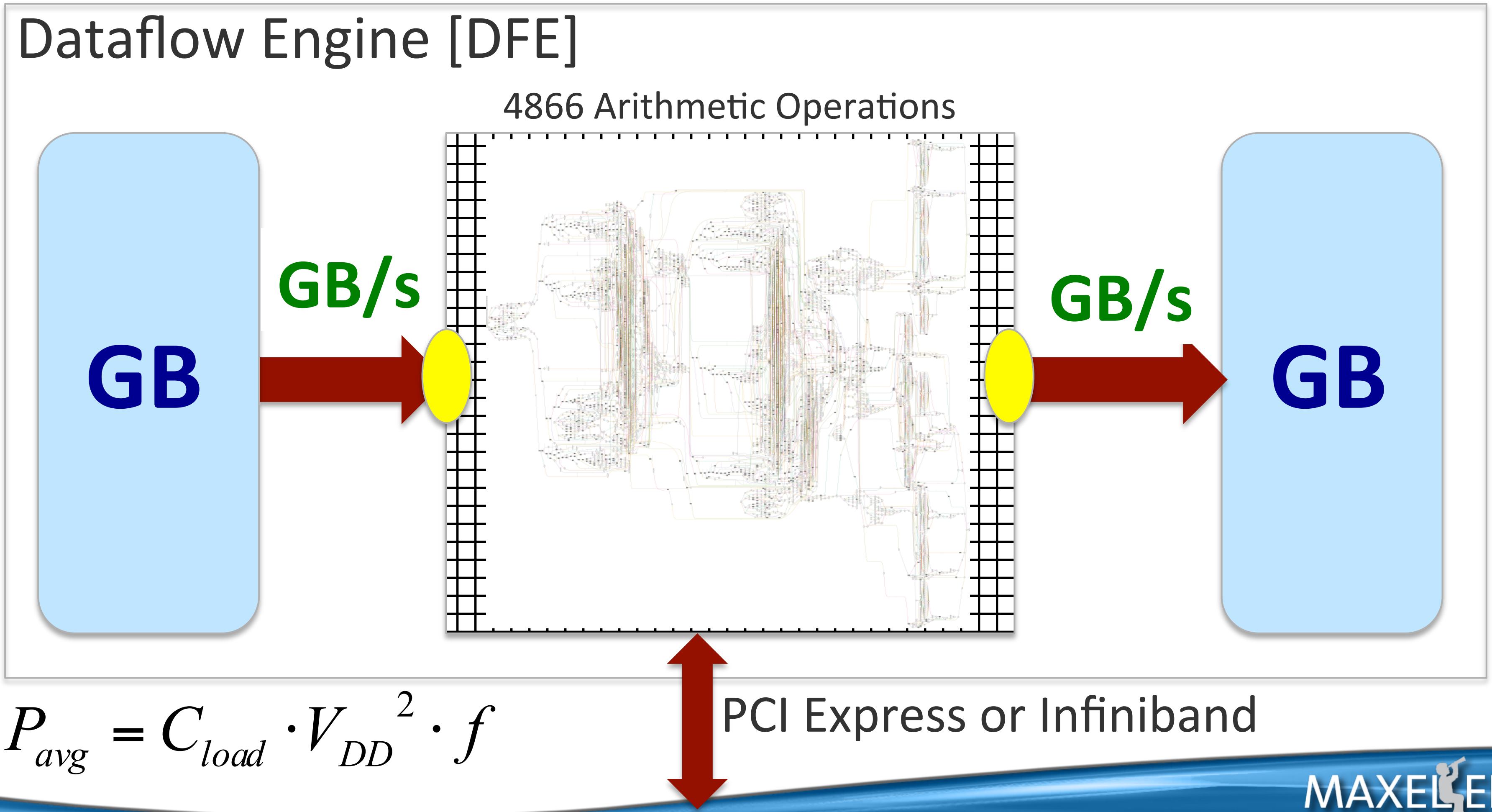
o.mencer@imperial.ac.uk g.gaydadjiev@imperial.ac.uk

Overview

- Real OpenSPL SCS instantiation – Maxeler's DFEs
- Architecture and Programming Model
- SLIC: Using DFEs from MatLab, Python and more

Maxeler Multiscale Dataflow Computing

$T_{compute}$=**GB**/[**GB/s**] for up to 10K operations
computing on the stream within a window of 7 MB



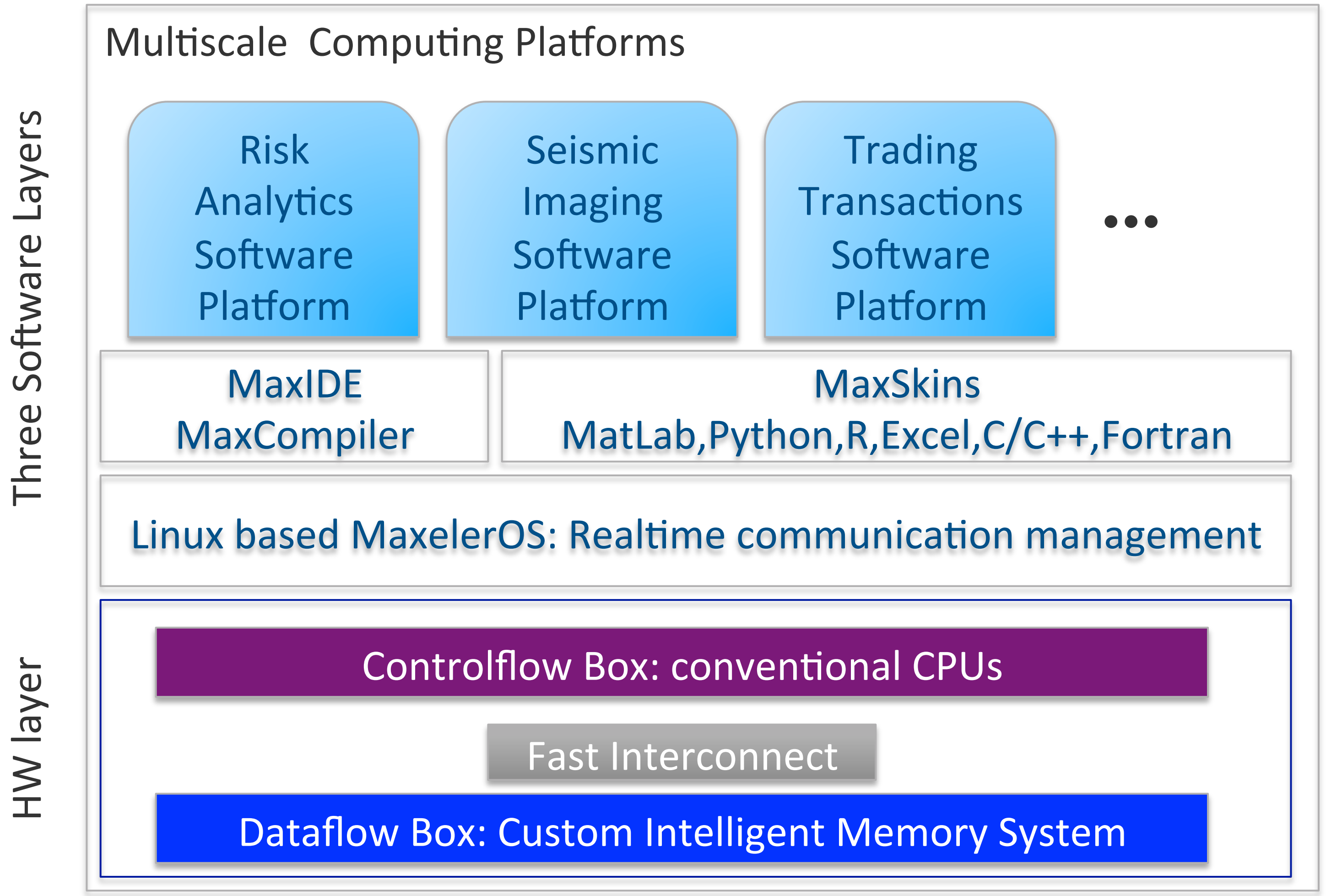
$$P_{avg} = C_{load} \cdot V_{DD}^{2} \cdot f$$

Mapping SCS to DFE

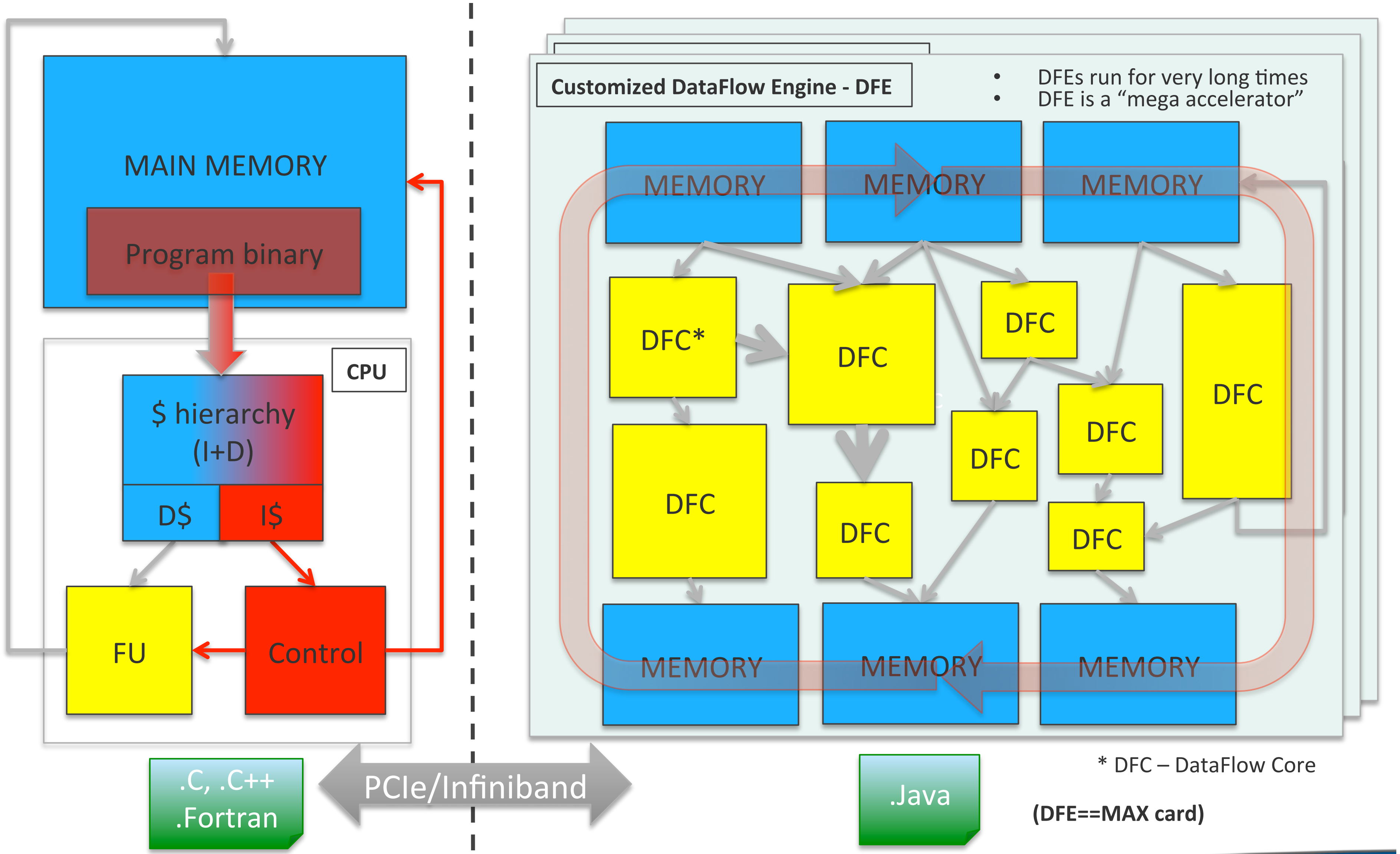
- SCS → DFE for Multiscale Computing Platforms
- Spatial Configurable Technology → (largest conf. dev.)
- High-level language used Java
- OpenSPL Compiler → MaxCompiler
- OpenSPL OS → MaxOS
- OpenSPL data movement managed by "Manager.maxj"
- Domain specific support (too big to mention here), e.g., Finite Differences

The Platforms

decoupling the data plane and control plane


Multiscale Computing Platforms
Three Software Layers
HW layer
Risk Analytics Software Platform
Seismic Imaging Software Platform
Trading Transactions Software Platform
…
MaxIDE MaxCompiler
MaxSkins MatLab,Python,R,Excel,C/C++,Fortran
Linux based MaxelerOS: Realtime communication management
Controlflow Box: conventional CPUs
Fast Interconnect
Dataflow Box: Custom Intelligent Memory System

Maxeler Accelerator Architecture



- DFEs run for very long times
- DFE is a “mega accelerator”

* DFC – DataFlow Core

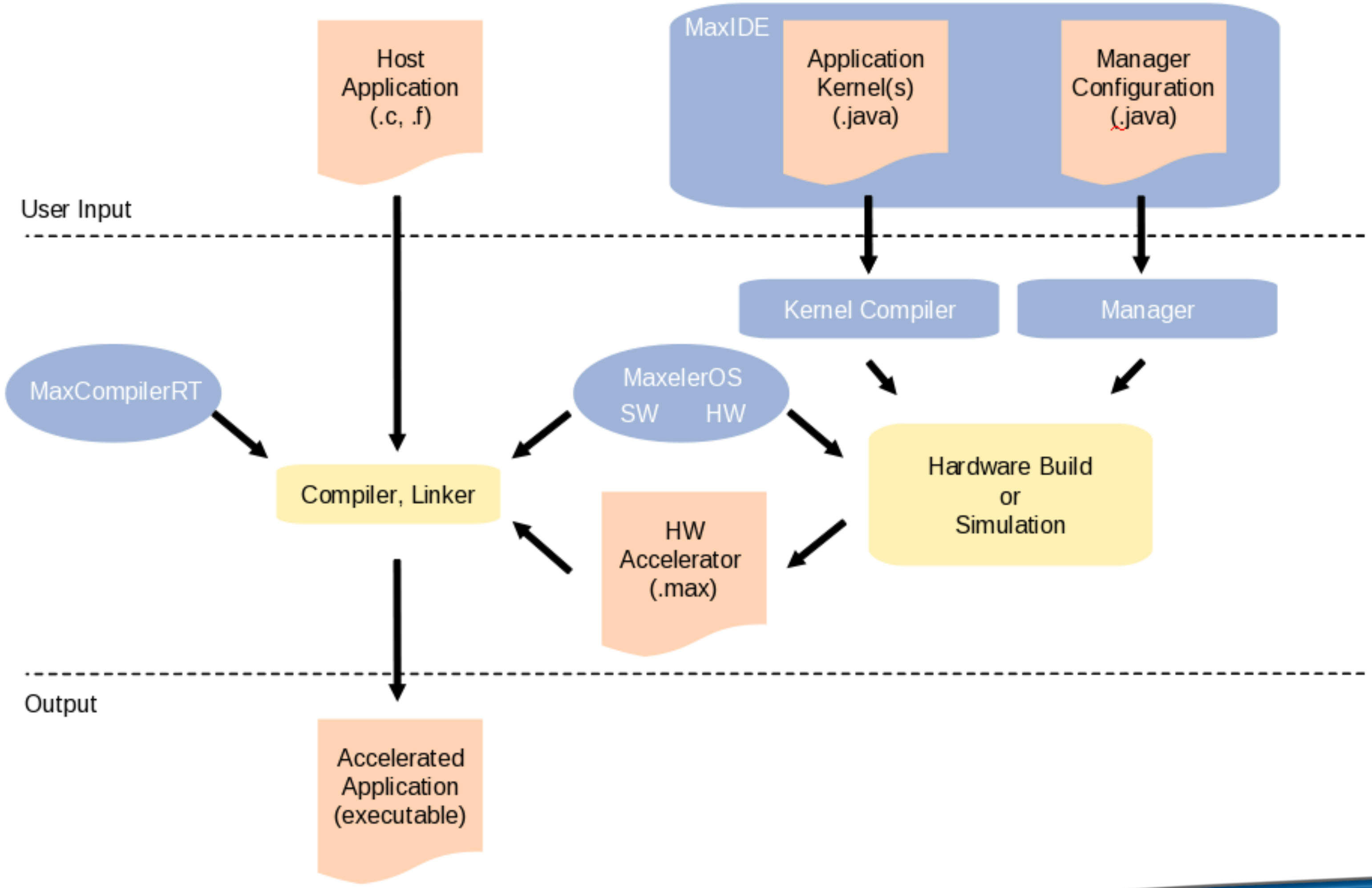
(DFE==MAX card)

MaxCompiler

- Complete development environment for Maxeler DFE accelerator platforms
- Write *MaxJ* code to describe the dataflow accelerator
  - *MaxJ* is an extension of Java for MaxCompiler
  - *Execute* the Java → *generate* the accelerator
- C software on CPUs *uses* the accelerator

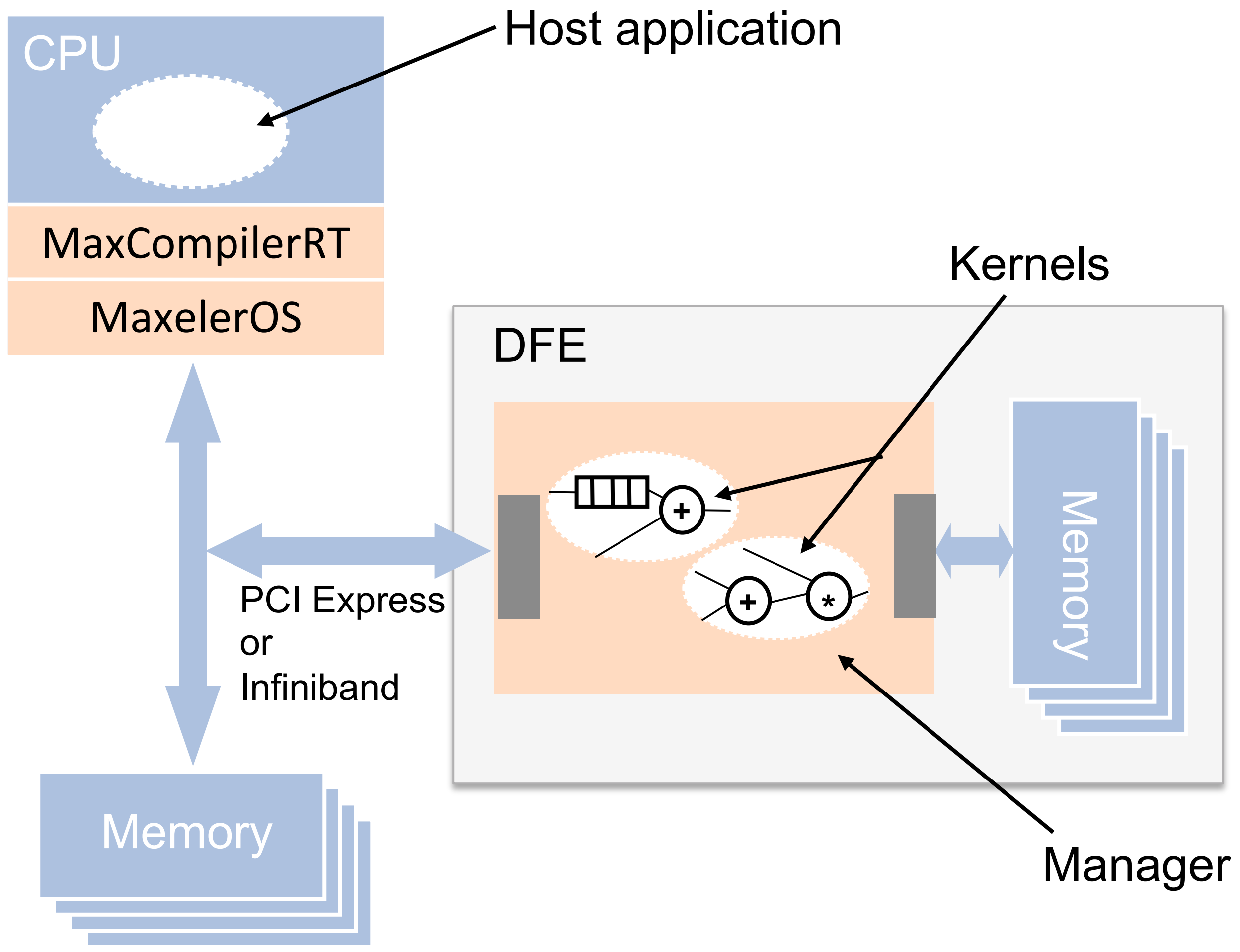
```
class MyAccelerator extends Kernel {
    public MyAccelerator(…) {
        DFEVar x = io.input("x", dfeFloat(8, 24));
        DFEVar y = io.input(“y", dfeFloat(8, 24));

        DFEVar x2 = x * x;
        DFEVar y2 = y * y;
        DFEVar result = x2 + y2 + 30;

        io.output(“z", result, dfeFloat(8, 24));
    }
}
```

MaxCompiler Architecture



MaxIDE
Host Application (.c, .f)
Application Kernel(s) (.java)
Manager Configuration (.java)
User Input
Kernel Compiler
Manager
MaxCompilerRT
MaxelerOS SW HW
Compiler, Linker
Hardware Build or Simulation
HW Accelerator (.max)
Output
Accelerated Application (executable)

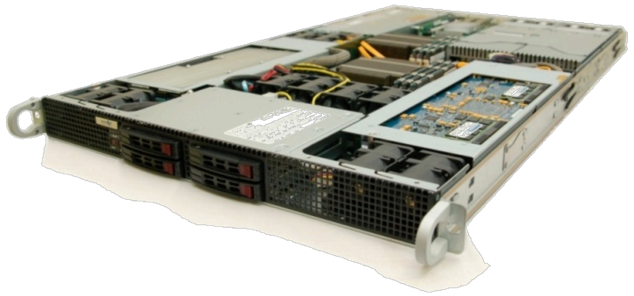
Application Components



Dataflow Engines (DFEs)

MPC C Series

High Density DFEs
Intel Xeon CPU cores and up to 6 DFEs with 288GB of RAM



MPC X Series

The Dataflow Appliance
Dense compute with 8 DFEs, 384GB of RAM and dynamic allocation of DFEs to CPU servers with zero-copy RDMA access

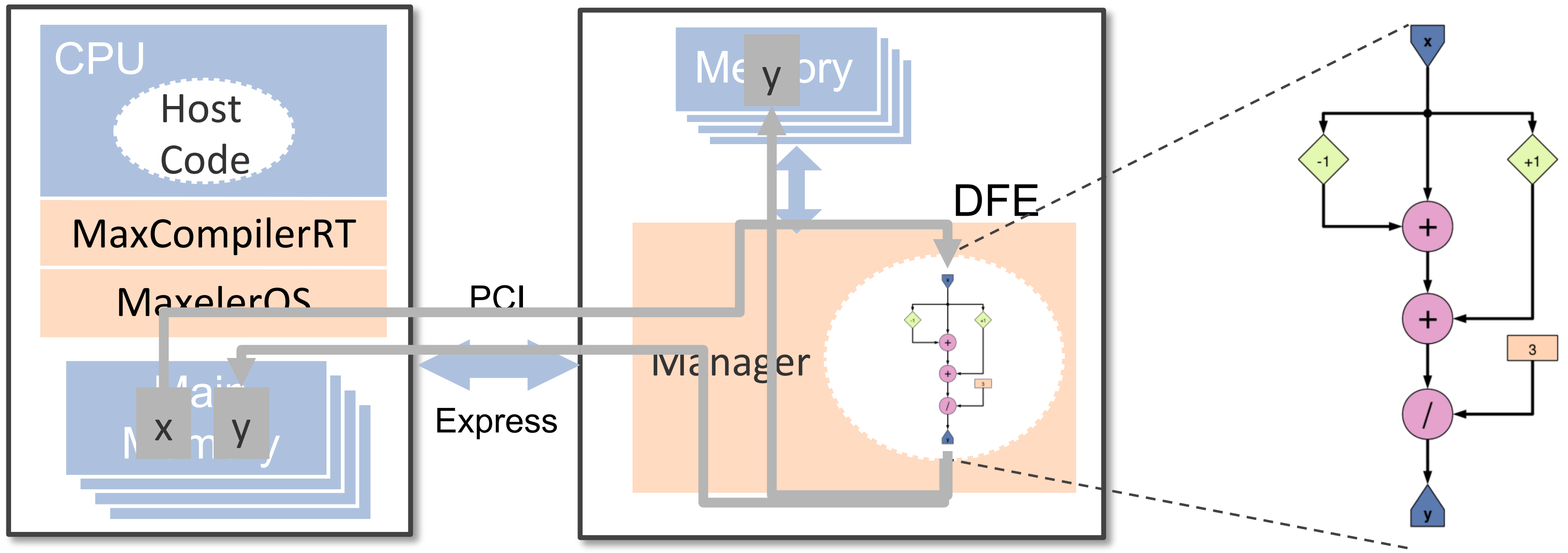
MPC N Series

The Low Latency Appliance
Intel Xeon CPUs and 4 DFEs with direct links to up to six 10Gbit Ethernet connections

MaxWorkstation
Desktop dataflow development system

Dataflow Engines
48GB DDR3, high-speed connectivity and dense configurable logic

Development Process



Host Code (.c)

```
float *x, *y;

for (int i =1; i < DATA_SIZE-1; i++)
   y[i]=(x[i-1]+x[i]+x[i+1])/3;
```

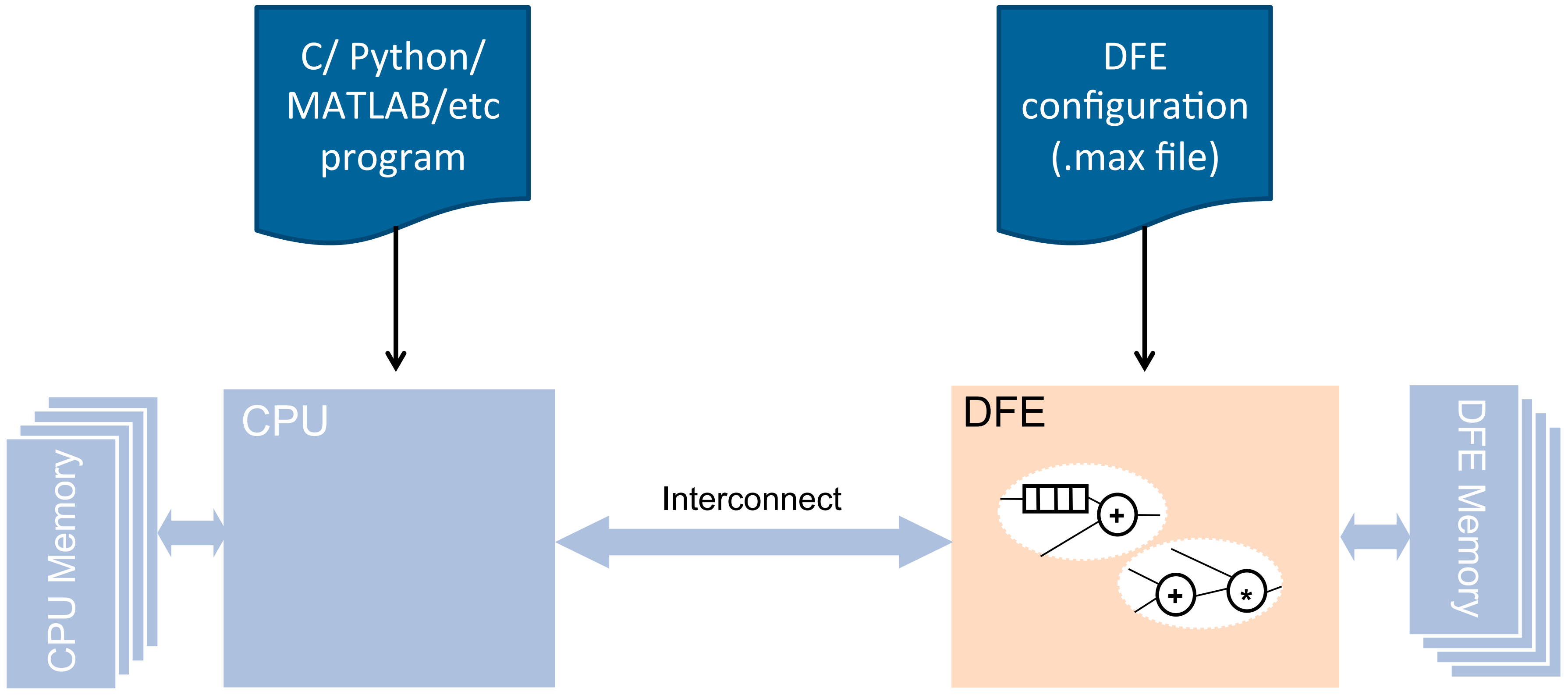
Manager (.java)

```
Manager m = new Manager(“Eg");
Kernel k = new EgKernel();

m.setKernel(k);
m.setIO(
   link(“x", PCIE),
   link(“y", PCIE), DRAM_LINEAR1D));
m.build();
```

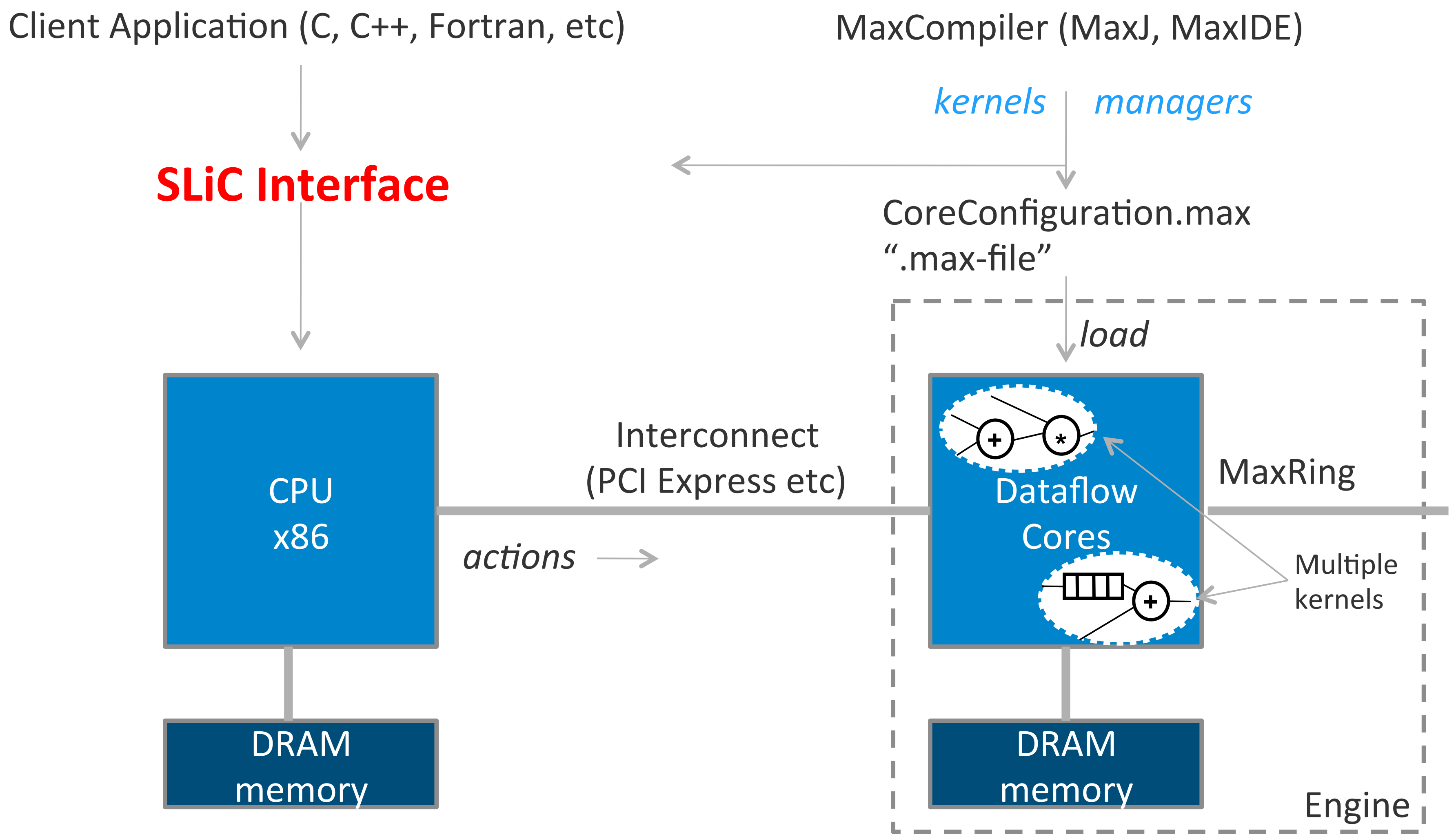
Kernel (.java)

```
DFEVar x = io.input("x", dfeFloat(8, 24));
DFEVar prev = stream.offset(x, -1);
DFEVar next = stream.offset(x, 1);

DFEVar sum = prev+x+next;
DFEVar result = sum/3;

io.output("y", result, dfeFloat(8, 24));
```

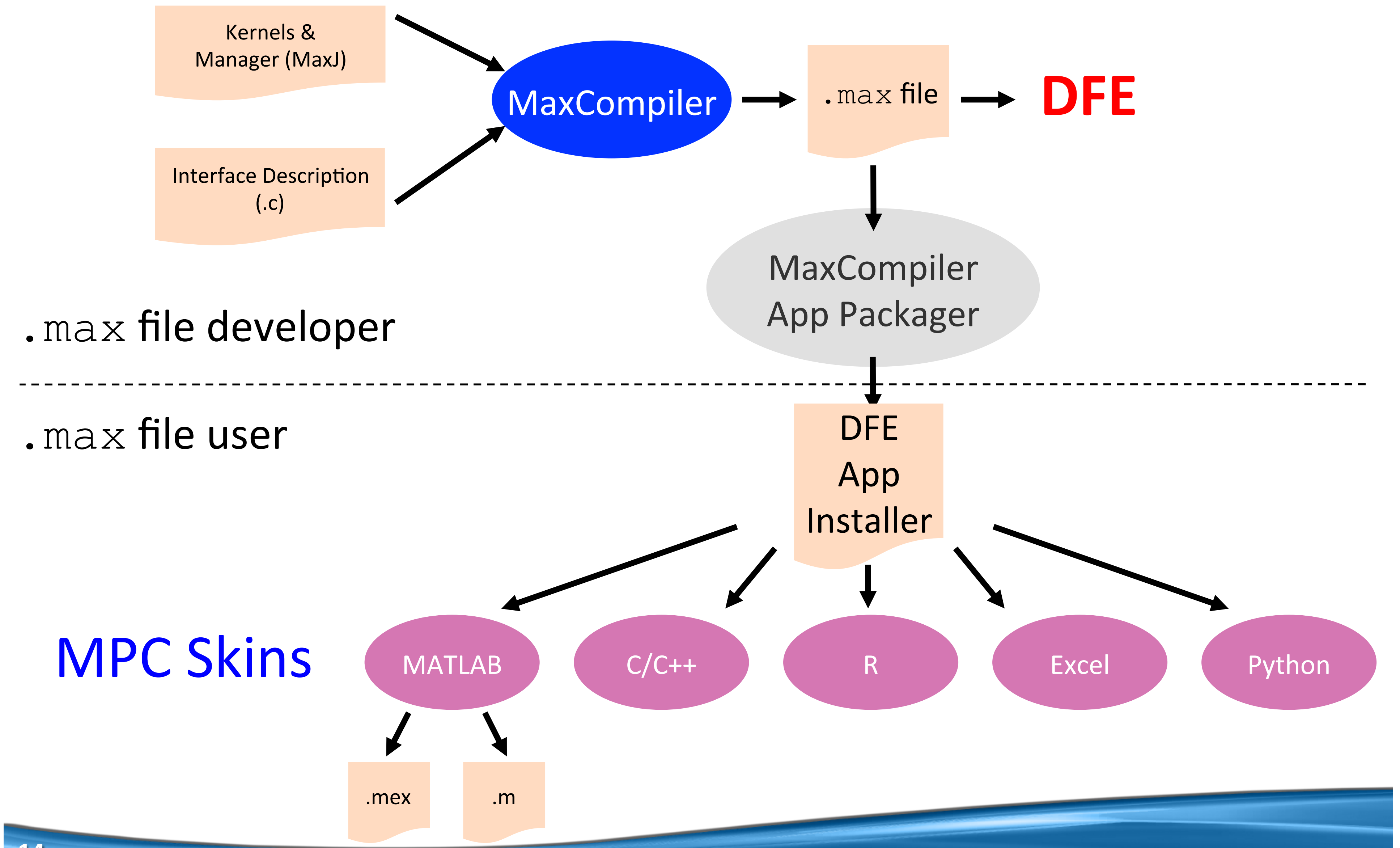
Using DFEs


C/ Python/ MATLAB/etc program
DFE configuration (.max file)
CPU Memory
CPU
Interconnect
DFE
+
+
*
DFE Memory

CPU↔DFE communication in MaxelerOS



Using and programming MPC machines

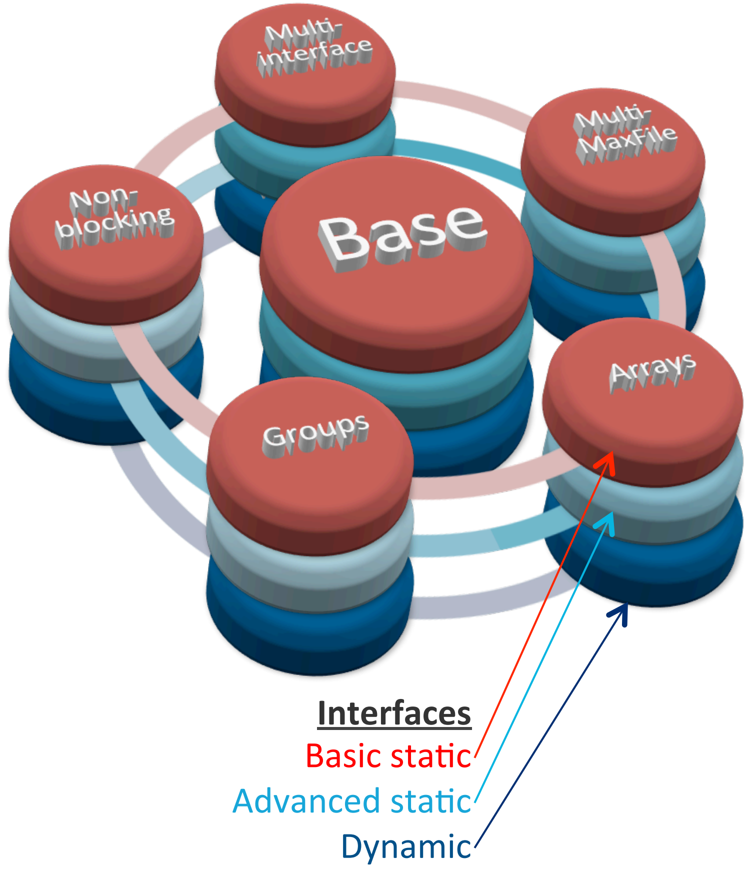


SLiC API Overview

- Engine Code (MaxJ): Specification of “interfaces”
  - Specify CPU interface to .max-file
    - Encapsulates scalar-input values, LMEM addresses, stream offsets, etc...
  - Multiple interfaces per .max-file (with a default)

- CPU Code: Application/DFE integration
  - Interface generated by MaxCompiler in the .max-file
    - Basic static interface – single function call
    - Advanced static interface – run actions on individual engines
  - Dynamic interface – for maximum flexibility

SLiC Programming Concepts

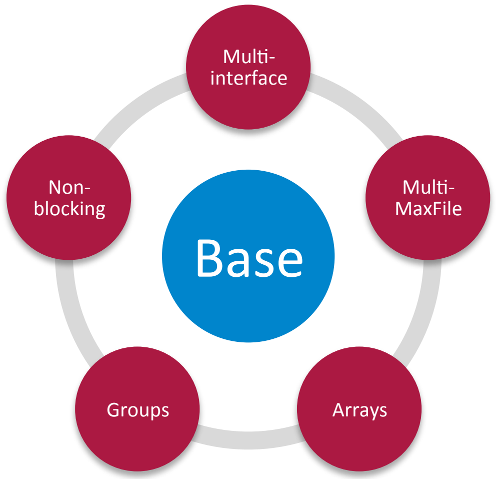


- Base
  - single maxfile, single interface, and single user/process
- Multi-interface
  - multiple use cases in one MaxFile
- Multi-MaxFile
  - multiple MaxFiles in one application
- Arrays
  - for using MaxRing
- Groups
  - share engines across multiple processes
- Non-blocking
  - asynchronous action runs

Types of Interface

- Basic Static:
  - single-function-call interface for using Maxeler engines
- Advanced Static:
  - fine-grain control over which DFEs to use
  - enables use of groups + arrays
  - asynchronous execution
- Dynamic:
  - all functionality of advanced static interface
  - doesn't require use of statically generated API
  - kernels, scalar I/Os, etc. identified with strings

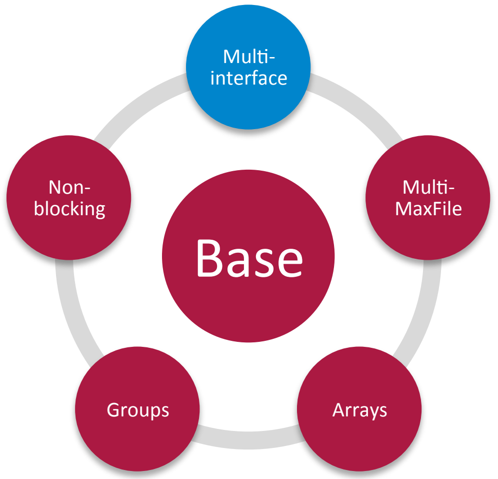
Base interface



```
#include "MaxSLiCInterface.h"
#include "AVG.h" // or #include "AVG.max"

#define N 16

int main(...) {
   float x[N], y[N];
   // Fill-in x array
   AVG(N, x, y);
   // Use y array
}
```

- String "AVG" comes from .max-file name as specified in the MaxJ code.
- AVG() function generated by MaxCompiler using an interface named "default"
- First suitable, available engine will be used
- Engine will be held open for program life-time

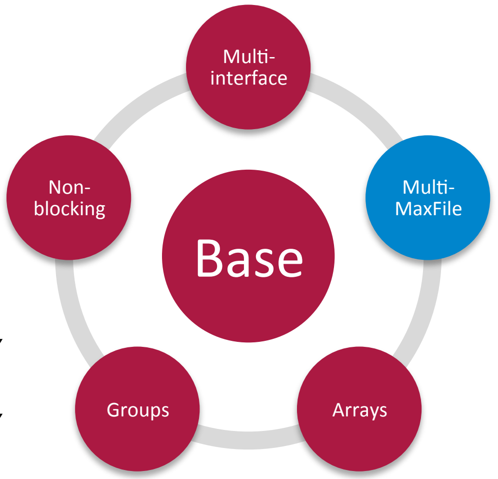
Multi-interface



```
#include "MaxSLiCInterface.h"
#include "AVG.h" // or #include "AVG.max"

#define N 16

int main(...) {
   float x[N], y[N];

   AVG(N, x, y);
   AVG_special(N, x, y);
   AVG_user(N, 10, x, y);
}
```

- MaxCompiler generates a function for each interface.
- Function prototypes appear in AVG.h with auto-generated doxygen comments.

Multi-maxfile



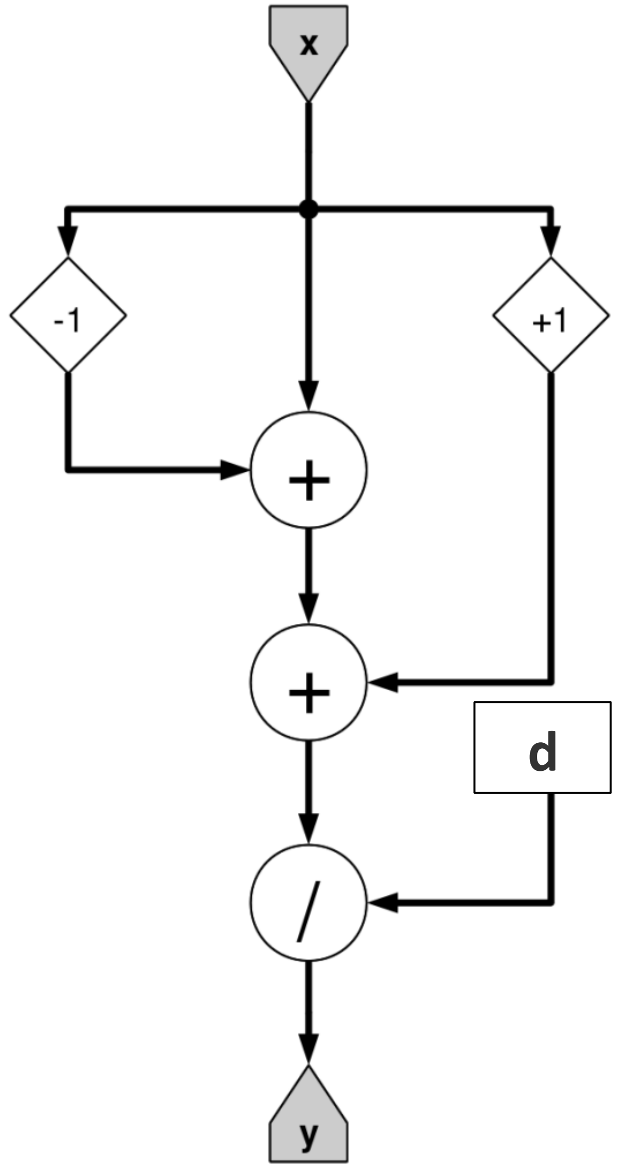
```
#include “MaxSLiCInterface.h”
#include “AVG.h” // or #include “AVG.max”
#include “SUM.max” // or #include “SUM.h”

#define N 16

int main(...) {
   float x[N], y[N];
   ...
   AVG(N, x, y);
   SUM(N, y, x);
   ...
}
```

Arrays and Groups

- Arrays are for using MaxRing connected engines
  - whole array can be run with a single command

- Groups allow engines to be shared between multiple processes/users
  - static or dynamic resource balancing of numbers of engines allocated to particular applications

The AVG Example



- 3-point moving average
- Stream input x
- Stream output y
- Scalar input d
- x, y connected to CPU
- .max-file name: “AVG”
- Kernel name: “AVGKernel”

AVG CPU Interface

- Actual information needed to run .max-file:
  - data pointers for X and Y streams
  - size of X and Y streams
  - value of scalar-input ‘d’
  - number of ‘ticks’ to run AVGKernel for

- A good CPU interface for AVG engine would be:
  - pointers to data for X and Y streams
  - integer N – used for: size of X and Y, number of ticks
  - Scalar ‘d’ **not specified** as a sensible default is 3

Interface Specification for AVG

```
void build(…) {
   Manager m = new Manager(“AVG”, DFEParameters);
   m.setKernel(new AVGKernel(...));
   m.setIO(link(“x”, CPU), link(“y”, CPU));
   m.createSLiCInterface(myInterface());
   // or “m.createSLiCInterface();” to create default
   m.build();
}

EngineInterface myInterface() {
   EngineInterface interface = new EngineInterface();
   InterfaceParam N = interface.addParam(“N”, CPUTypes.INT32);
   interface.setTicks(“AVGKernel”, N);
   interface.setScalar(“AVGKernel”, “d”, 3.0);
   interface.setStream(“x”, CPUTypes.FLOAT, N*4);
   interface.setStream(“y”, CPUTypes.FLOAT, N*4);
   return interface;
}
```

Elements Configurable in an Interface

- Scalar input values
- Stream offset values
- Mapped memory data
- Kernel ‘ticks’
- Routing information for manager-level muxes, demuxes and fanouts
- DRAM address generator configurations

- Settings can also be “ignored” per-interface

Summary

- DFEs are the first SCS instantiation
- MaxCompiler and MaxOS together with the Manager and the Chip vendor specific tools (Xilinx or Altera) provide a complete OpenSPL implementation.
- Interfaces to Python, MatLab, C, C++, R and even Fortran…