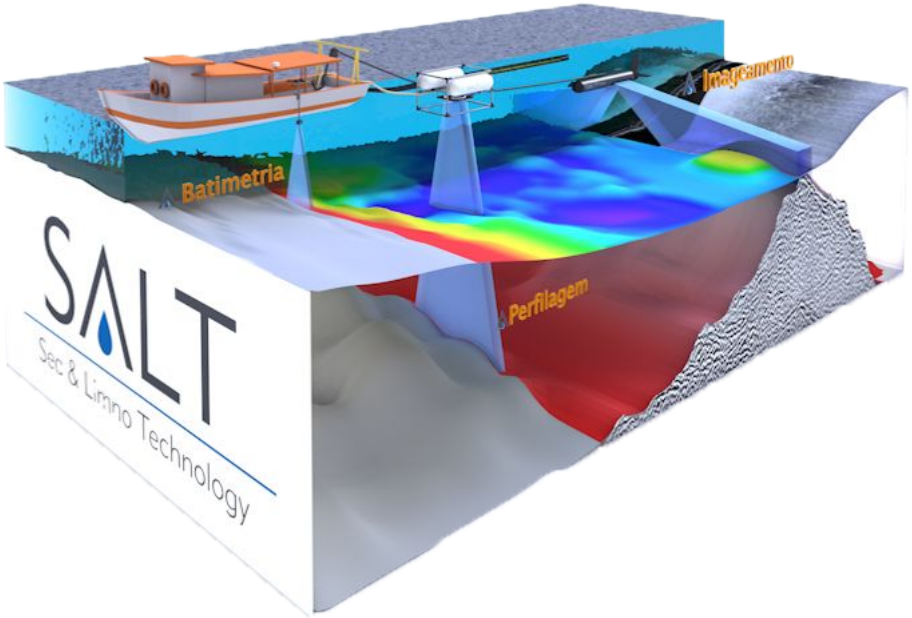
Enabling Multi-FPGA Clusters as an OpenMP Acceleration Device

Ramon Nepomuceno Renan Sterle Guido Araujo

Motivation: Seismic Imaging



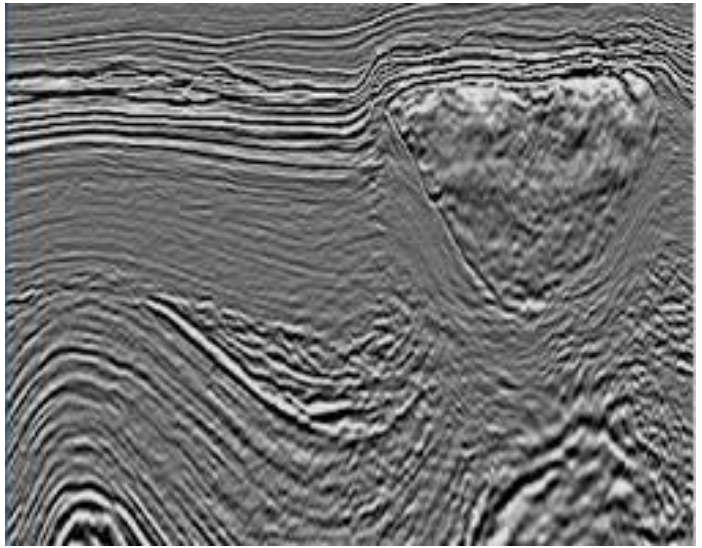
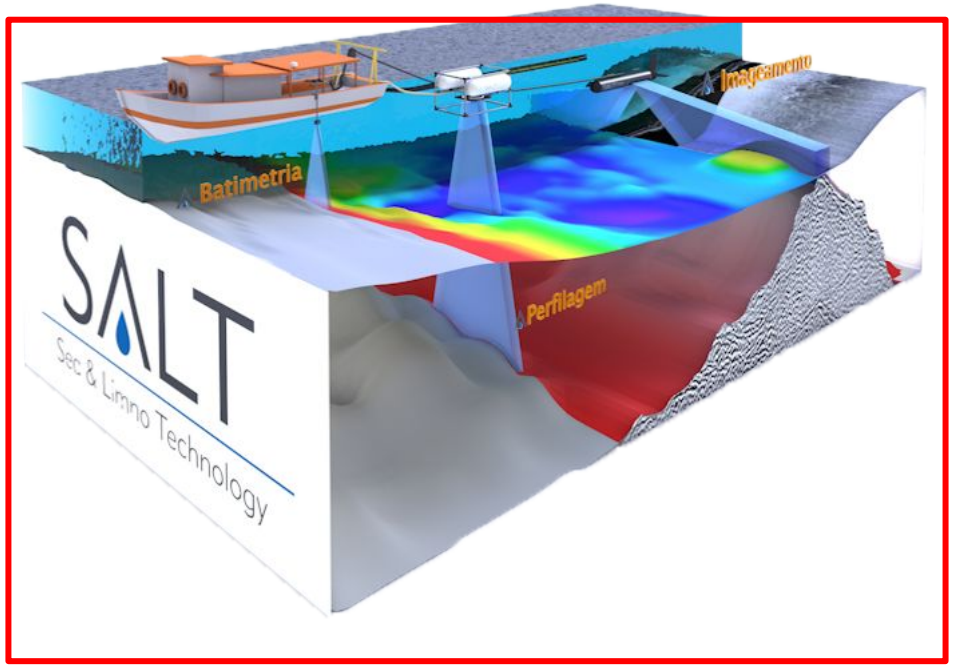


image taken from the internet:
https://www.saltambiental.com.br/geofisica/apresentacao/

Motivation: Seismic Imaging



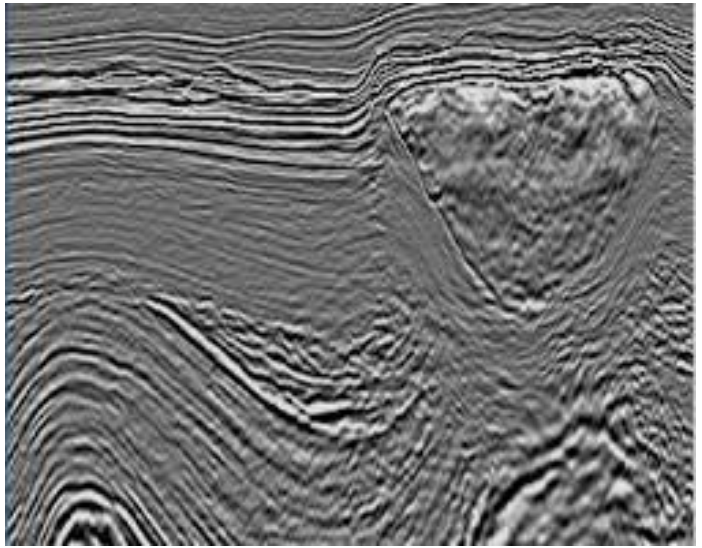
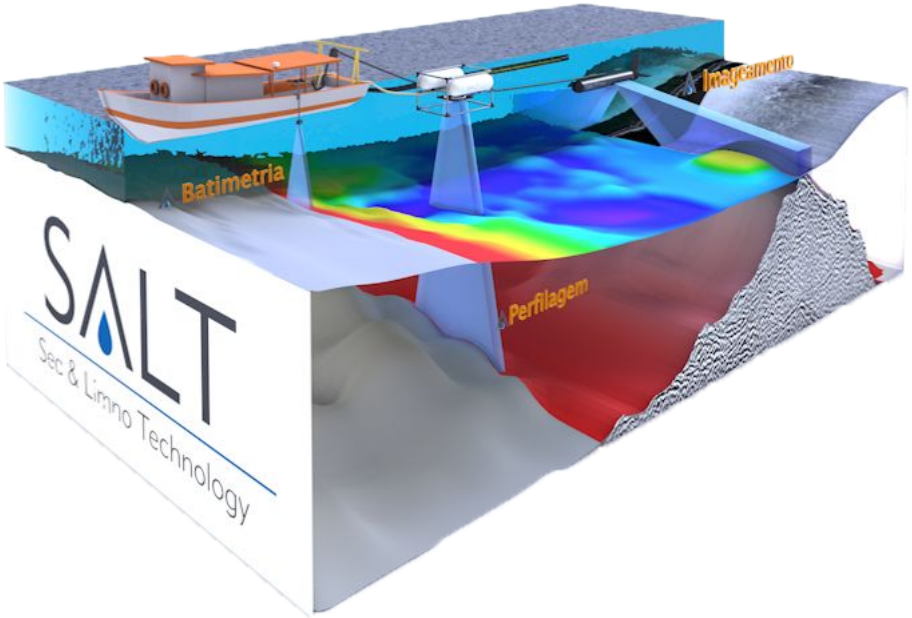


image taken from the internet:
https://www.saltambiental.com.br/geofisica/apresentacao/

Motivation: Seismic Imaging



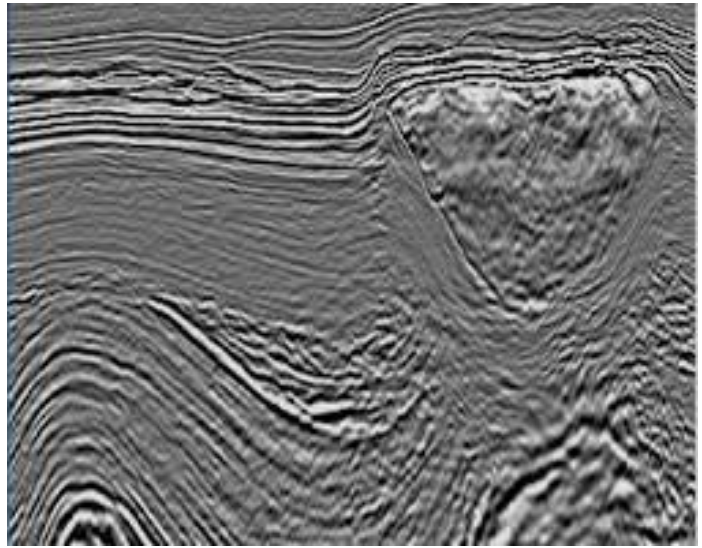
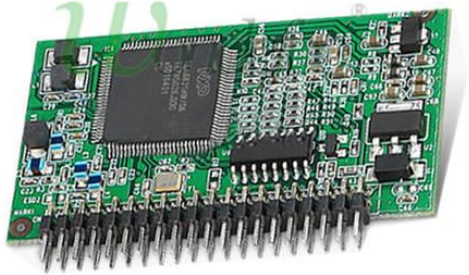
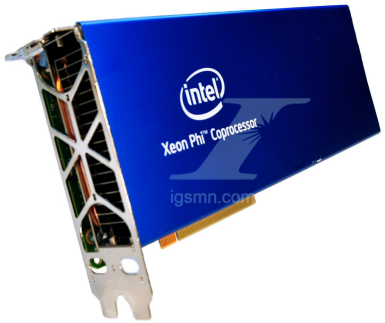
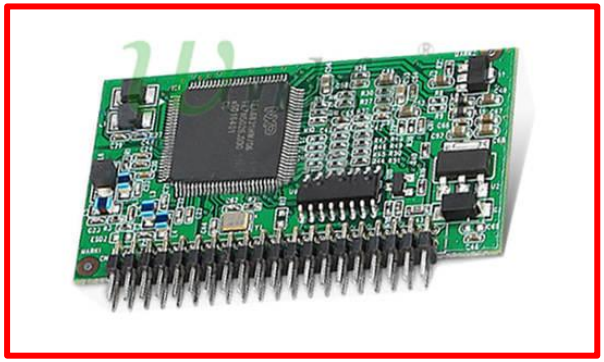


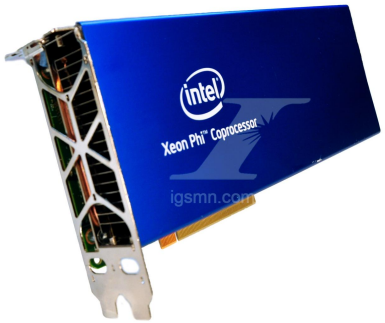
image taken from the internet:
https://www.saltambiental.com.br/geofisica/apresentacao/



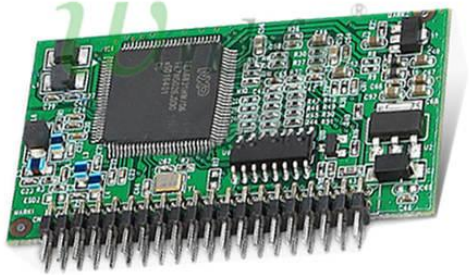


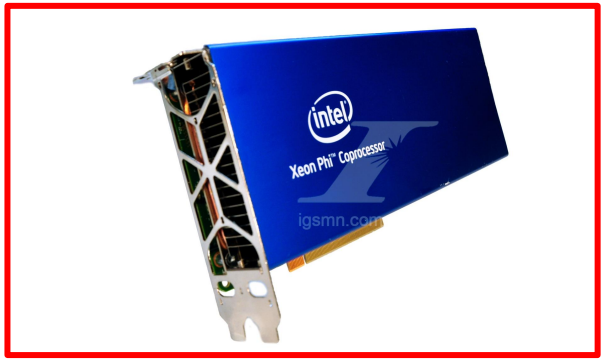
Heterogeneous Systems



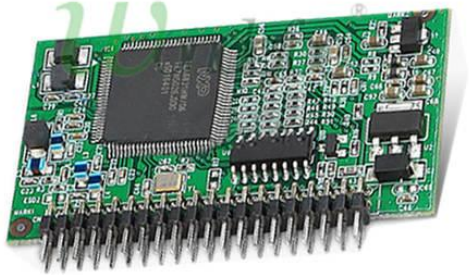


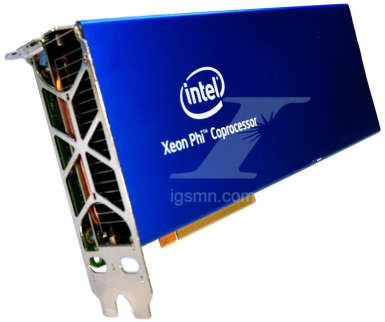
Heterogeneous Systems



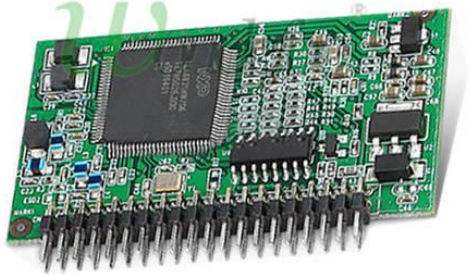


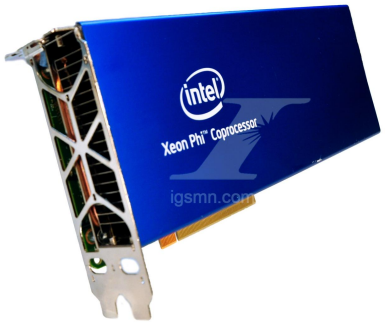
Heterogeneous Systems





Heterogeneous Systems





Heterogeneous Systems

Introduction

- An FPGA is a powerful device that can speedup pipelined applications

- But even a powerful FPGA cannot handle very large problems like Seismic Imaging

- Need Multi-FPGA architectures

Introduction

- Moreover....
- FPGA toolchains are hardware oriented and unfriendly to software designers

Introduction

- Moreover....
- FPGA toolchains are hardware oriented and unfriendly to software designers
- Need a simpler and transparent approach to embed FPGA accelerators into software applications

How to program such systems?

OpenMP
Enabling HPC since 1997

- We propose using OpenMP task-based programming model for Multi-FPGA architectures. We are extending it to:
  - Enable automatic bitstream/data offloading to FPGAs
  - Transparently map data dependencies to IPs/boards interconnects
  - Allow a seamless flow of data to/from FPGA.

Background

OpenMP Task Directive


```
#pragma omp parallel
  #pragma omp single
    #pragma omp task depend (out:A)
      A = foo();
    for(int i = 0; i < 2; i ++){
      #pragma omp task depend(in:A)\\
       depend(out: B[i])
            B[i]=bar(a)
     }
     for(int i = 0; i < 4; i ++){
       #pragma omp task depend(in:B[i/2])
          fun(B[i/2]);
     }
```

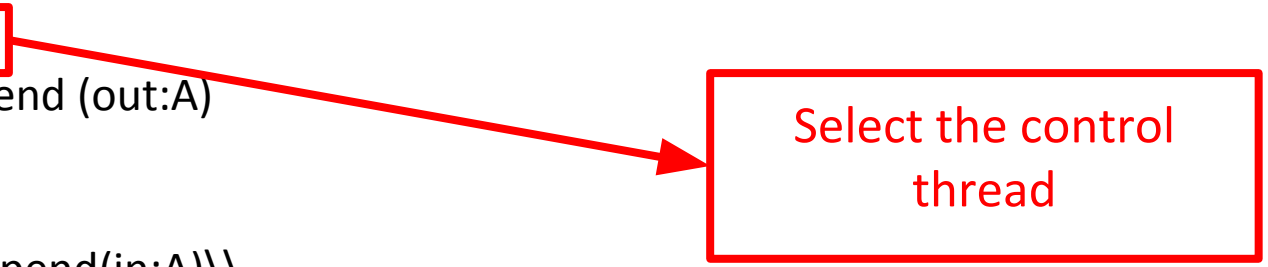
OpenMP Task Directive

```
#pragma omp parallel
  #pragma omp single
    #pragma omp task depend (out:A)
      A = foo();
    for(int i = 0; i < 2; i ++){
      #pragma omp task depend(in:A)\\
       depend(out: B[i])
            B[i]=bar(a);
     }
     for(int i = 0; i < 4; i ++){
       #pragma omp task depend(in:B[i/2])
          fun(B[i/2]);
     }
```

Create a pool of worker threads

OpenMP Task Directive

```
#pragma omp parallel
 #pragma omp single
    #pragma omp task depend (out:A)
      A = foo();
    for(int i = 0; i < 2; i ++){
      #pragma omp task depend(in:A)\\
       depend(out: B[i])
            B[i]=bar(a);
     }
     for(int i = 0; i < 4; i ++){
       #pragma omp task depend(in:B[i/2])
          fun(B[i/2]);
     }
```

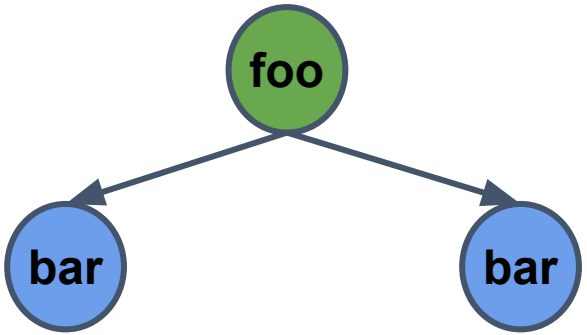


OpenMP Task Directive

```
#pragma omp parallel
  #pragma omp single
    #pragma omp task depend (out:A)
      A = foo();
    for(int i = 0; i < 2; i ++){
      #pragma omp task depend(in:A)\\
       depend(out: B[i])
            B[i]=bar(a)
    }
    for(int i = 0; i < 4; i ++){
      #pragma omp task depend(in:B[i/2])
          fun(B[i/2]);
    }
```

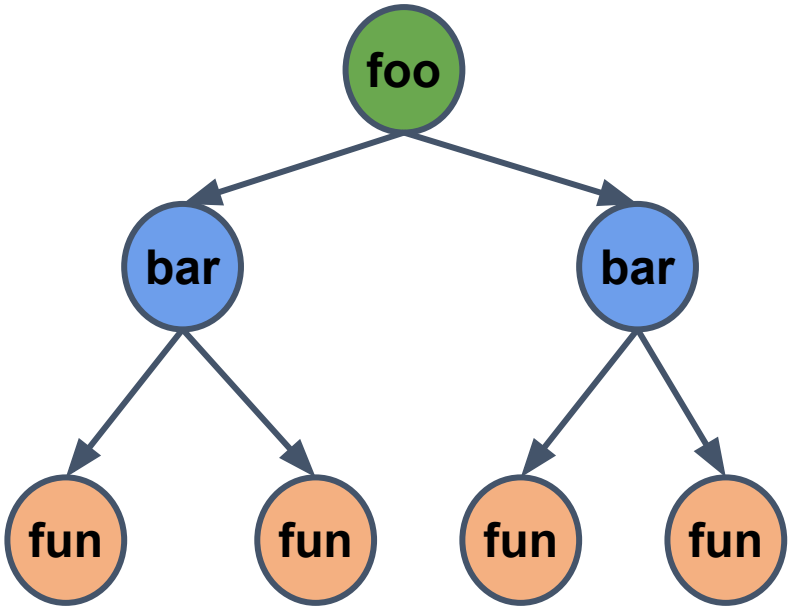
OpenMP Task Directive

```
#pragma omp parallel
  #pragma omp single
    #pragma omp task depend (out:A)
      A = foo();
    for(int i = 0; i < 2; i ++){
      #pragma omp task depend(in:A)\\
       depend(out: B[i])
            B[i]=bar(a)
    }
     for(int i = 0; i < 4; i ++){
       #pragma omp task depend(in:B[i/2])
           fun(B[i/2]);
     }
```

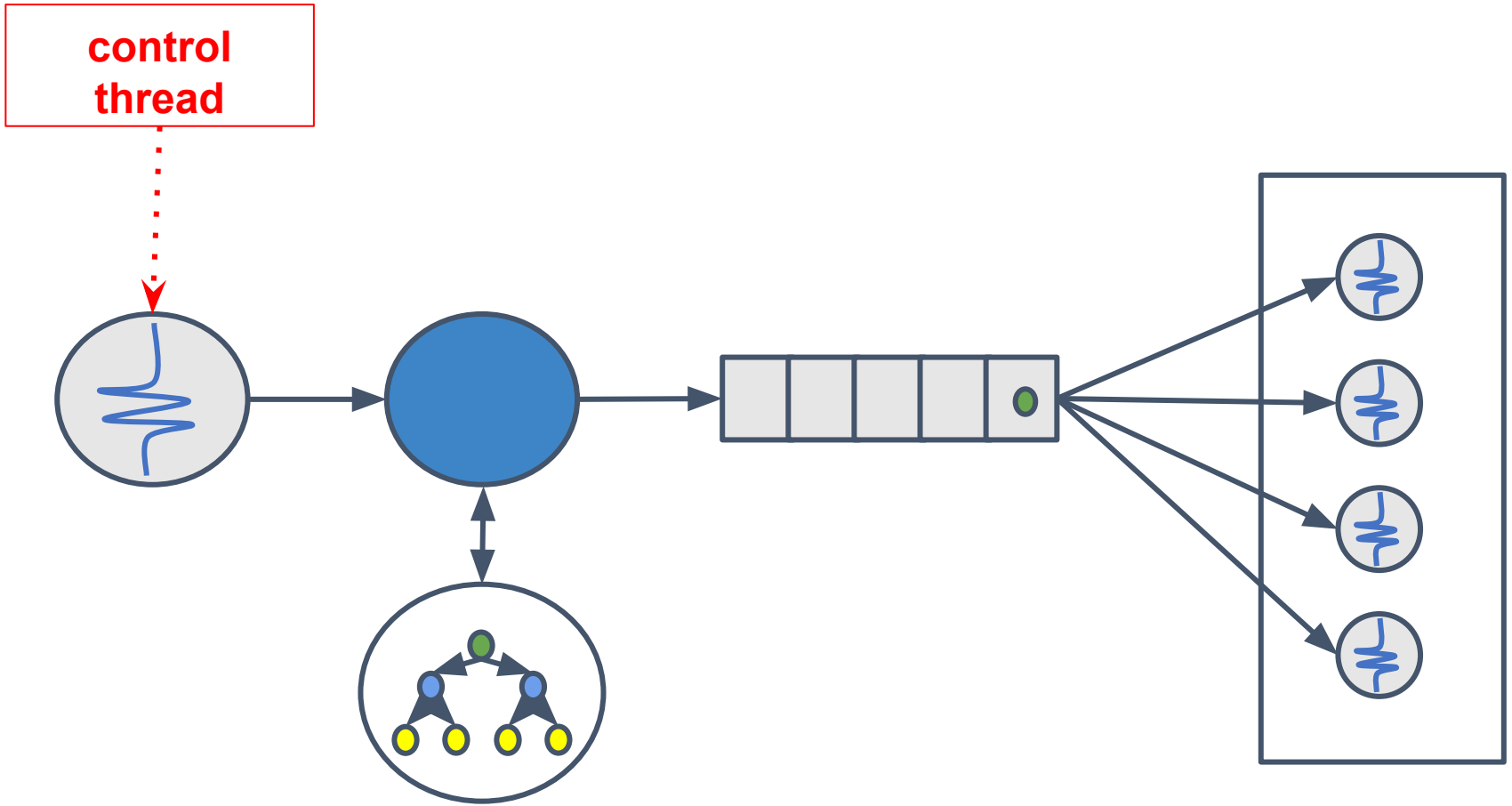


OpenMP Task Directive

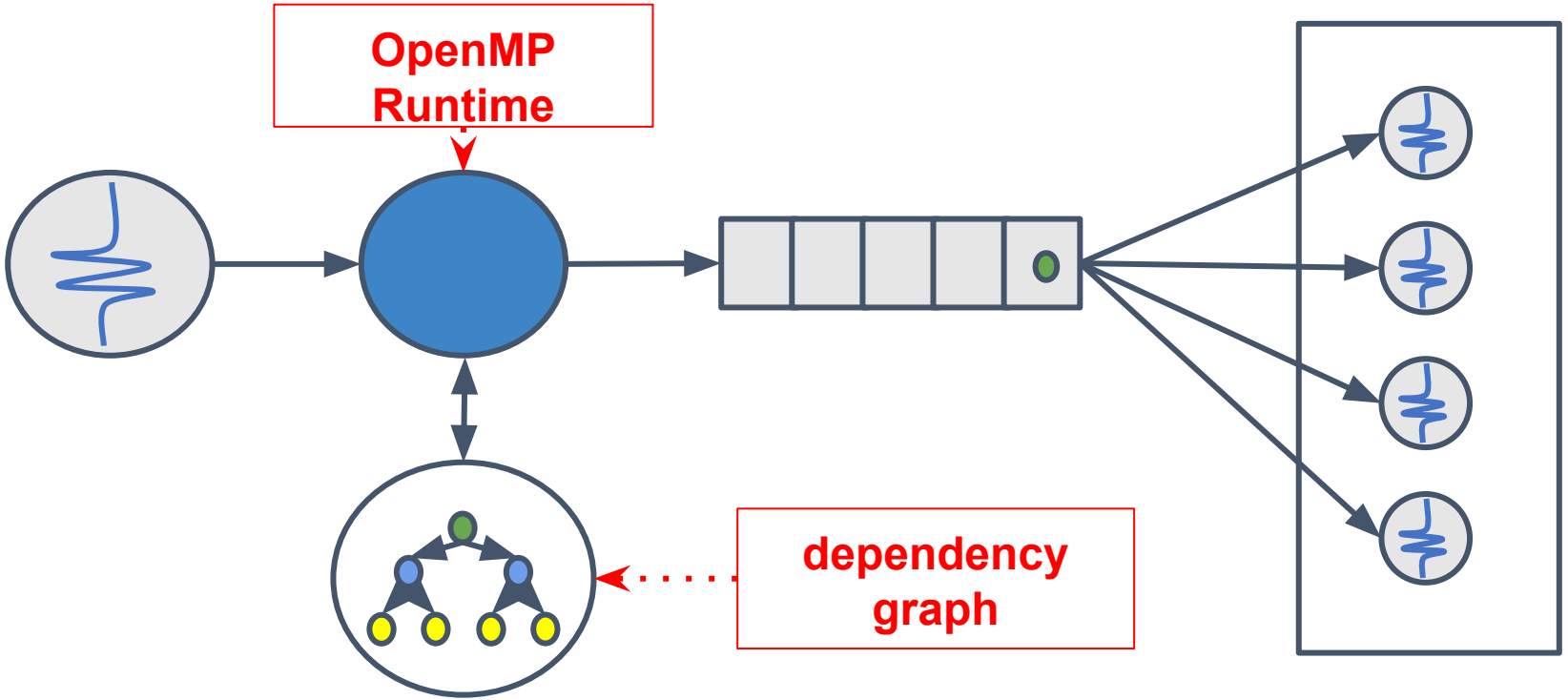
```
#pragma omp parallel
  #pragma omp single
    #pragma omp task depend (out:A)
      A = foo();
    for(int i = 0; i < 2; i ++){
      #pragma omp task depend(in:A)\\
       depend(out: B[i])
            B[i]=bar(a)
    }
    for(int i = 0; i < 4; i ++){
      #pragma omp task depend(in:B[i/2])
          fun(B[i/2]);
    }
```



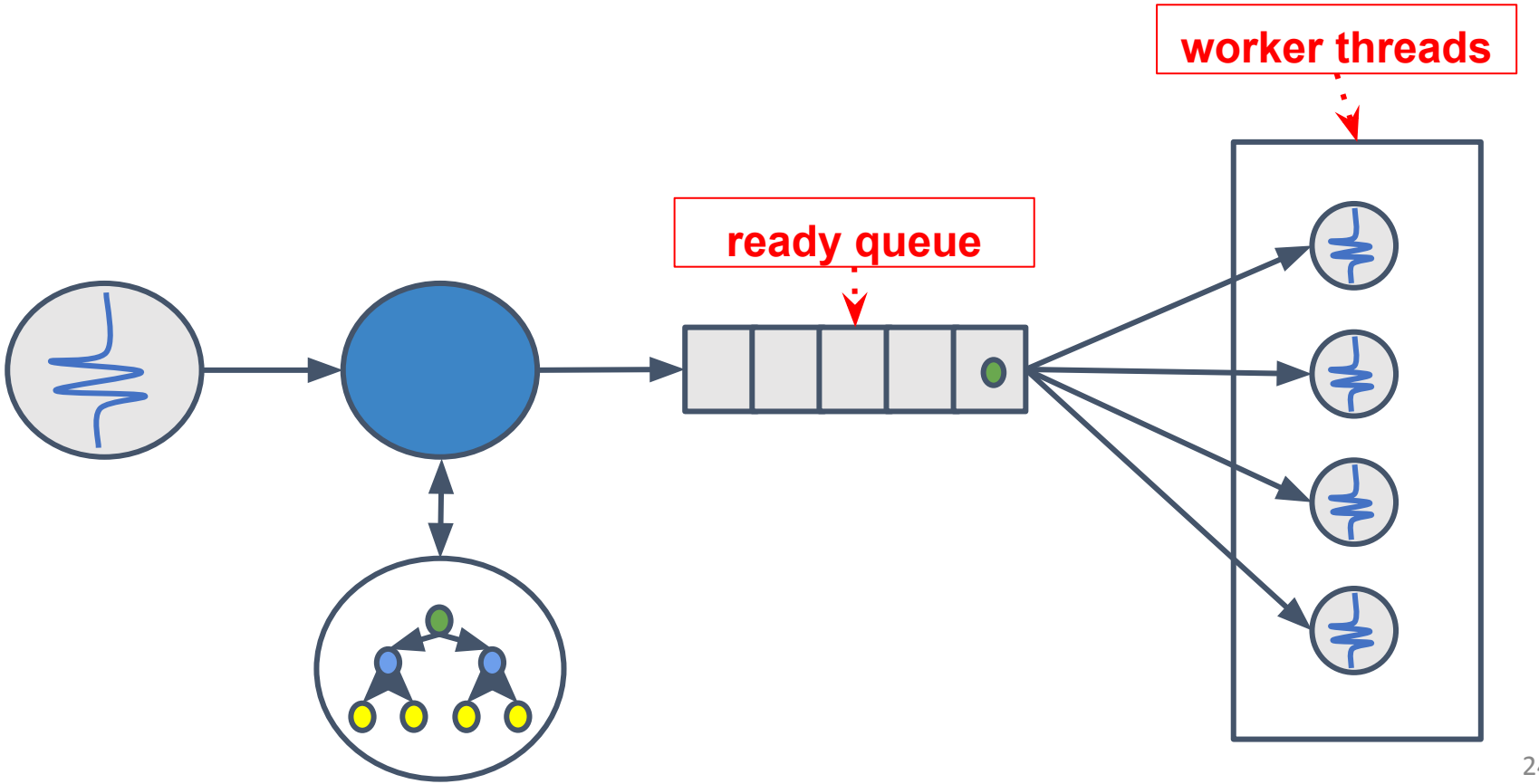
OpenMP Task Runtime



OpenMP Task Runtime

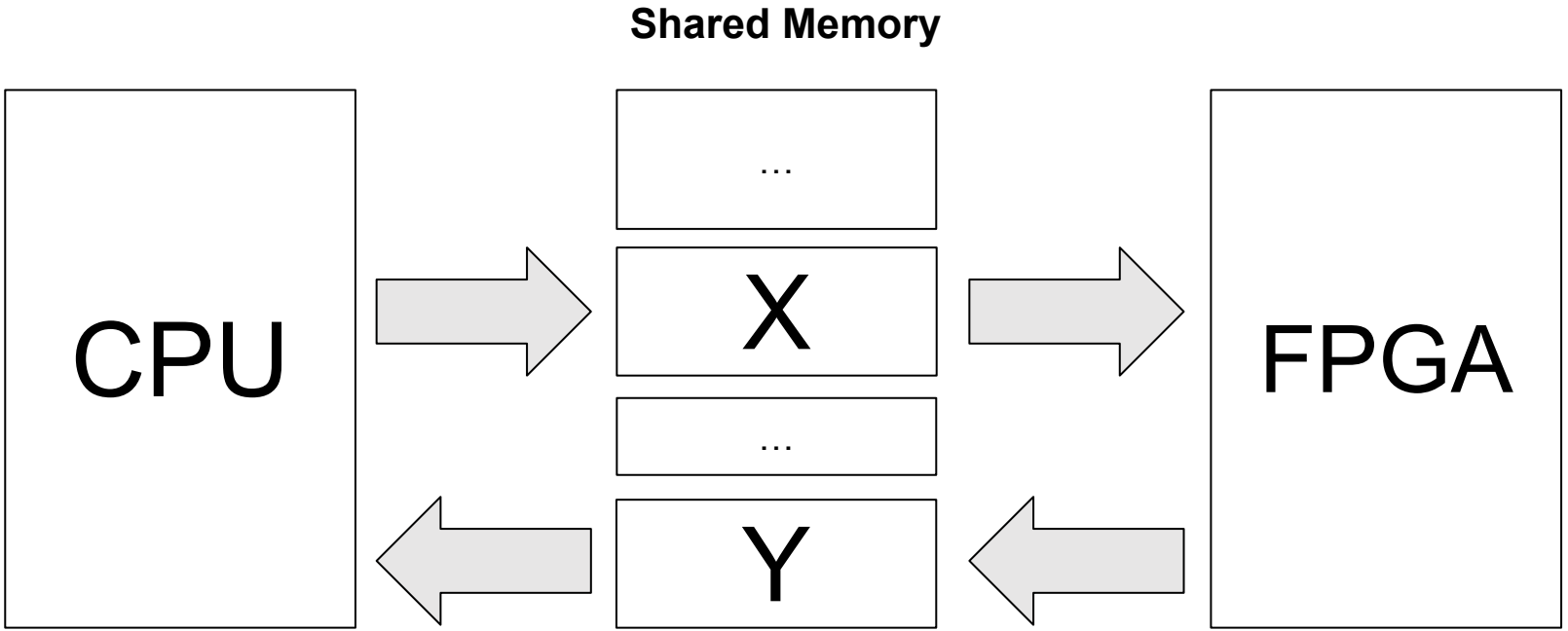


OpenMP Task Runtime



OpenMP Target Directive

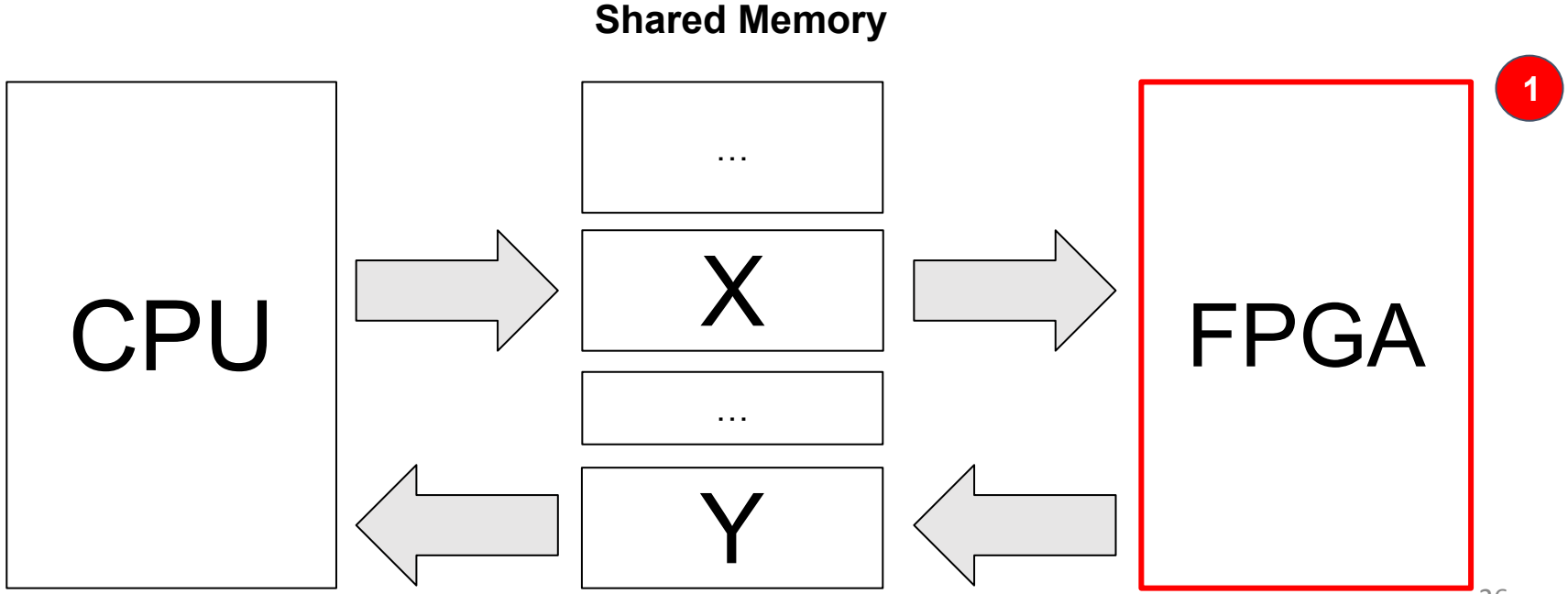
#pragma omp target device(FPGA) map(to: X) map(from: Y)



OpenMP Target Directive

1

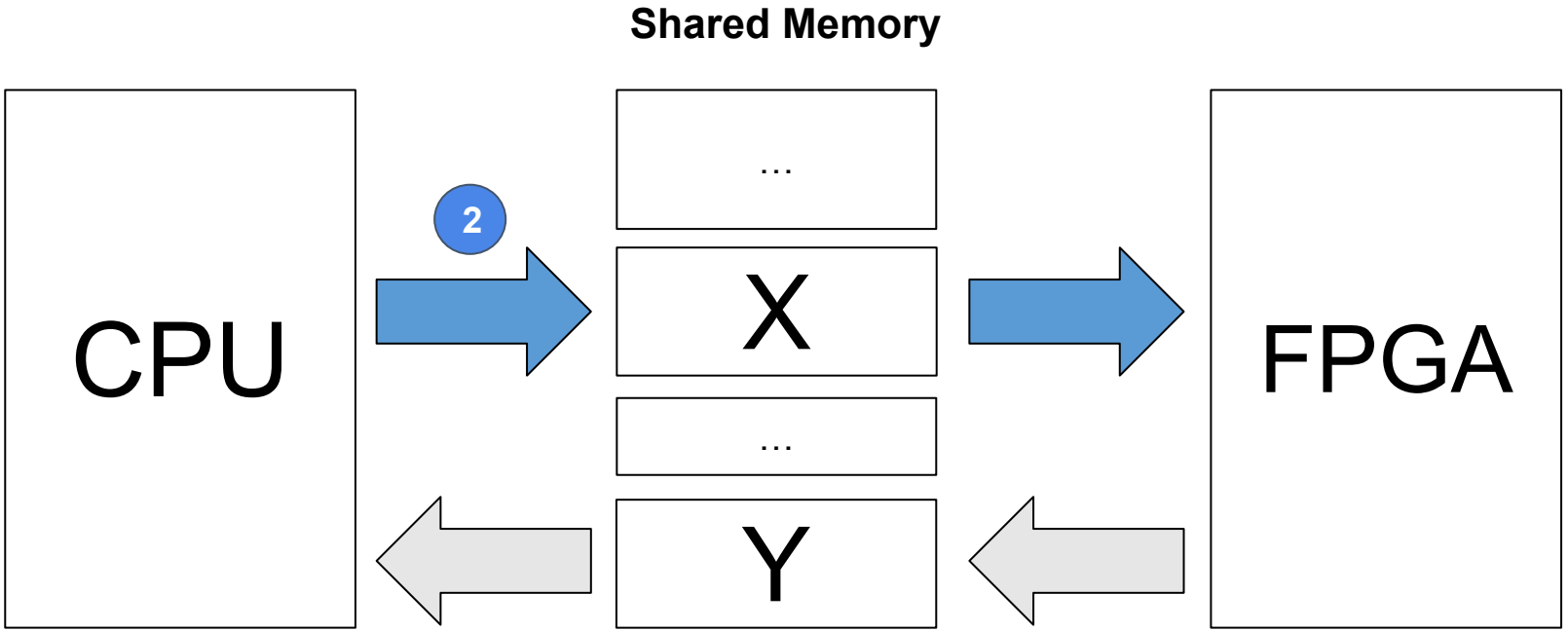
```
#pragma omp target device(FPGA)  map(to: X)  map(from: Y)
```



OpenMP Target Directive

```
#pragma omp target device(FPGA)  map(to: X)  map(from: Y)
```

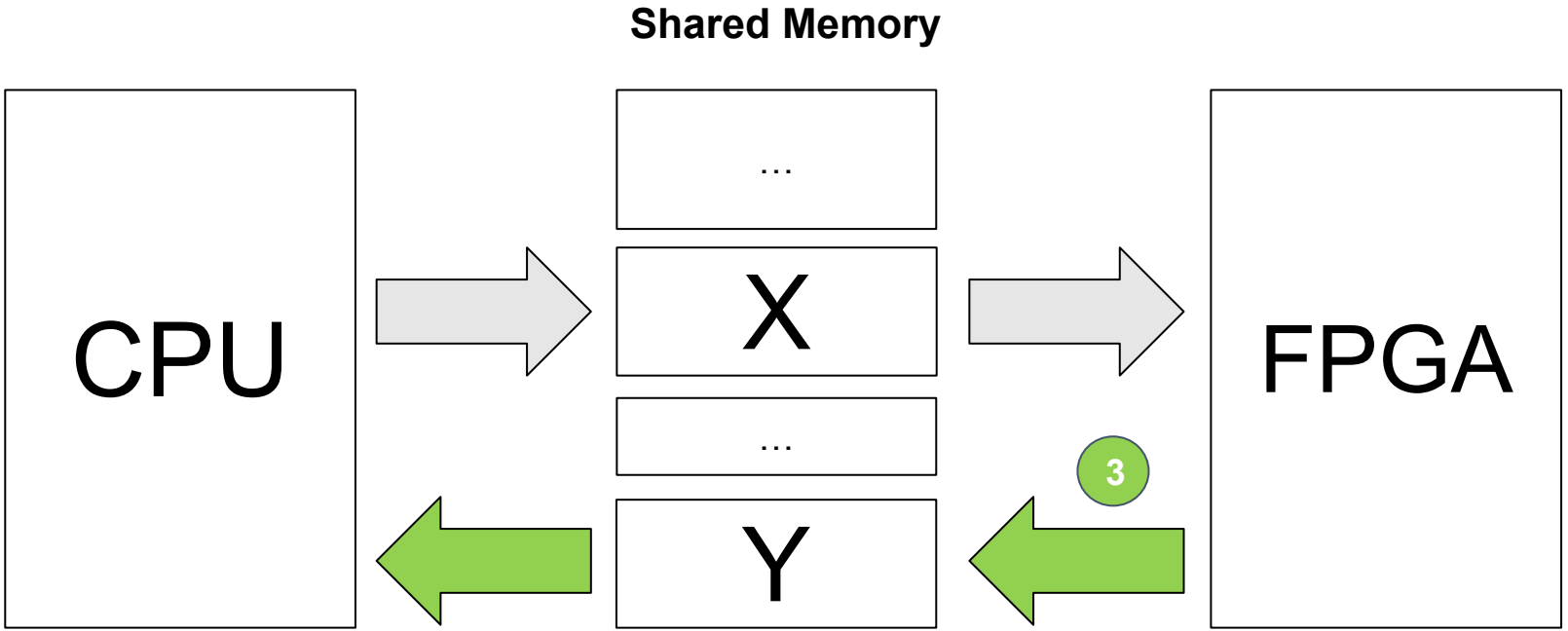
2



OpenMP Target Directive

3

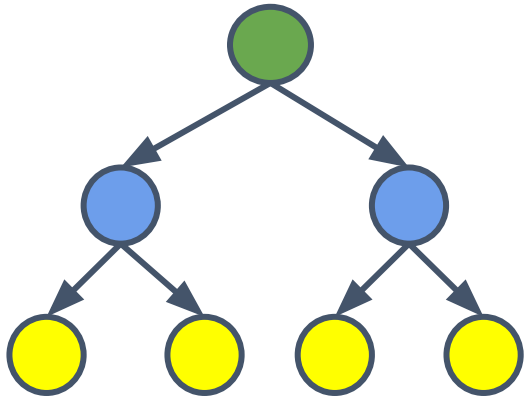
#pragma omp target device(FPGA) map(to: X) map(from: Y)



OpenMP Target Depend Directive

```
#pragma omp parallel
  #pragma omp single
    #pragma omp target device(GPU)\\
    map(from: A)
    depend (out: A) nowait
      A = foo();
    for(int i = 0; i < 2; i ++){
      #pragma omp target device(GPU)\\
       map(to: A) map(from: B[i]) \\
       depend (in: A) depend (out: B[i]) nowait
              B[i]=bar(a)
    }
    for(int i = 0; i < 4; i ++){
       #pragma omp target device(GPU)\\
       map(to: B[i/2]) \\
       depend (in: B[i/2])  nowait
         fun(B[i/2]);
    }
```

```
#pragma omp target device(GPU) \\
  map(from: A)                 \\
  depend(out: A) nowait
```



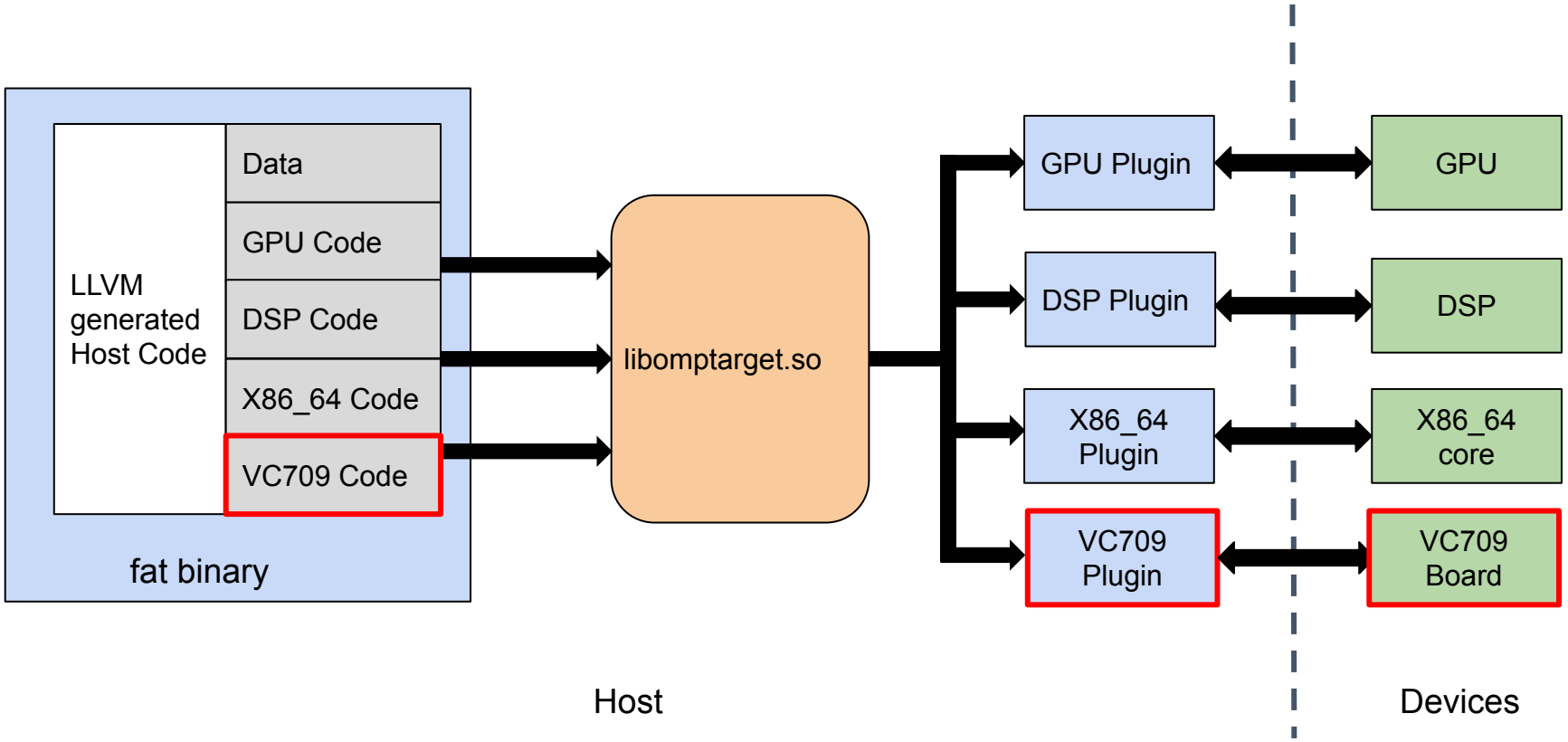
But LLVM/OpenMP cannot handle FPGAs

LLVM/OpenMP FPGA Limitations:

- Does not support for bitstream offloading;
- Does not support for mapping data dependencies to IP/boards interconnects;
- Does not allow a seamless flow of data to/from multiple-FPGA.

The Software

Our modules: LLVM/OpenMP VC709 Plugin



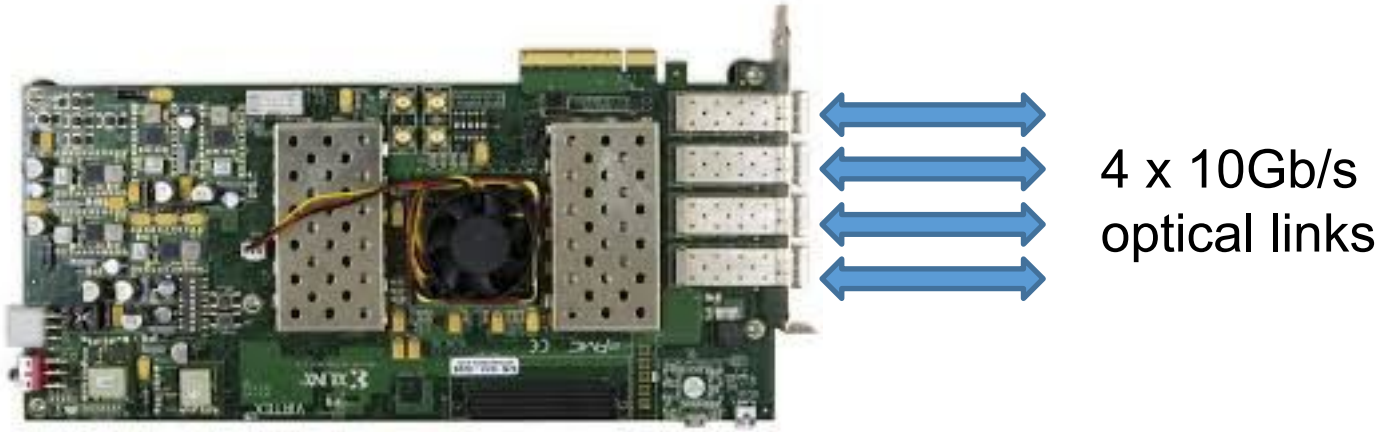
The Hardware

Single Node

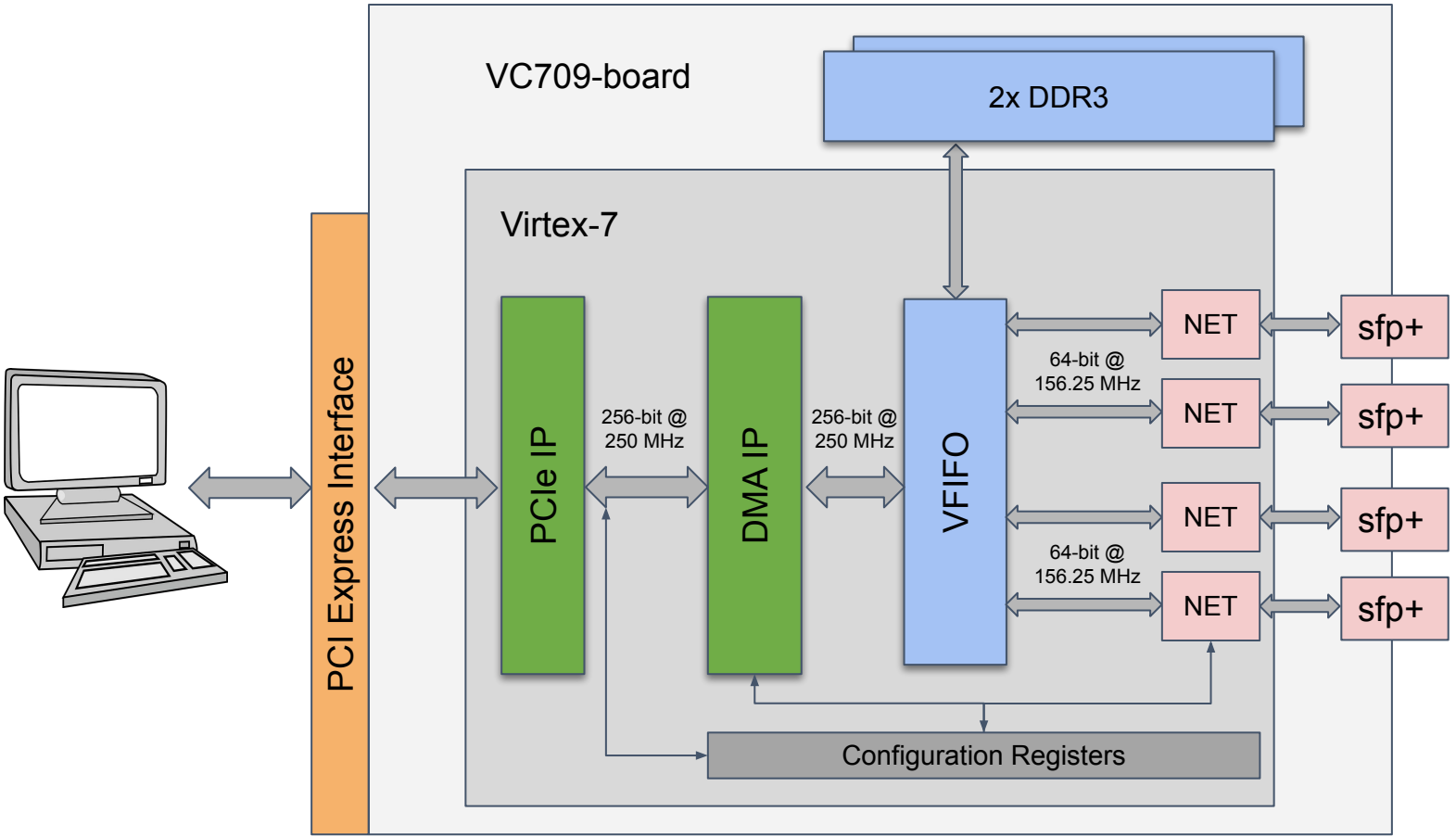
The FPGA

- Field Programmable Gate Array.
  - Xilinx VC709 Board
    - 2x 4GB DDR3 SODIMM memories;
    - 8-lane PCIe;
    - 4x SFP;

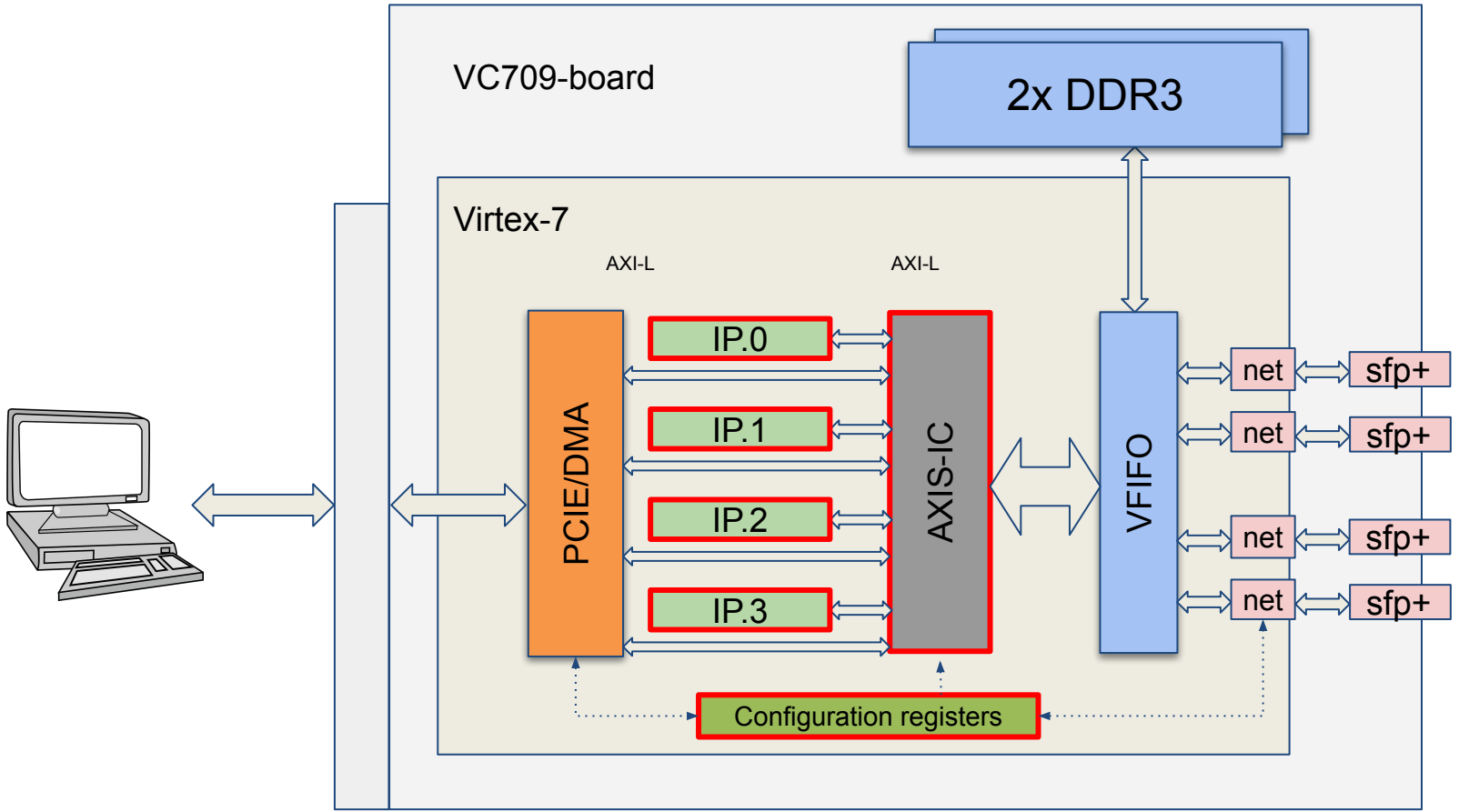
The Board



VC709 Target Reference Design



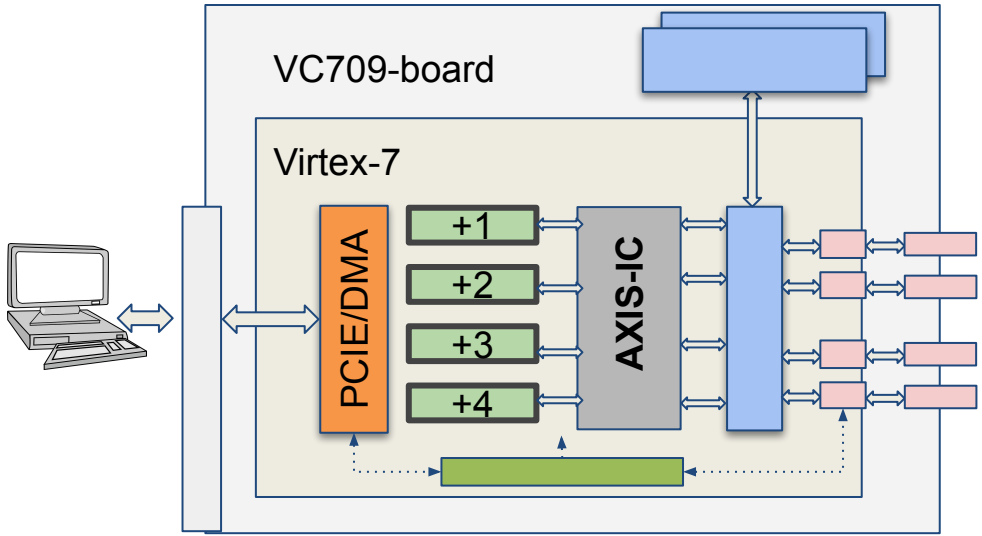
Our modules


VC709-board
2x DDR3
Virtex-7
AXI-L
AXI-L
PCIE/DMA
IP.0
IP.1
IP.2
IP.3
AXIS-IC
VFIFO
net
net
net
net
sfp+
sfp+
sfp+
sfp+
Configuration registers

Single node: How it works

```
#pragma omp parallel{
  #pragma omp single{
    for(int i=0; i < 4 ; i++){
      #pragma omp target device(VC709 + i)\\
       map(tofrom: A[:N])                  \\
       depend(inout: A[:N]) nowait{
        for(int j = 1; j <= N ; j++ )
          A[j] = A[j] + i;
      }
    }
  }
}
```

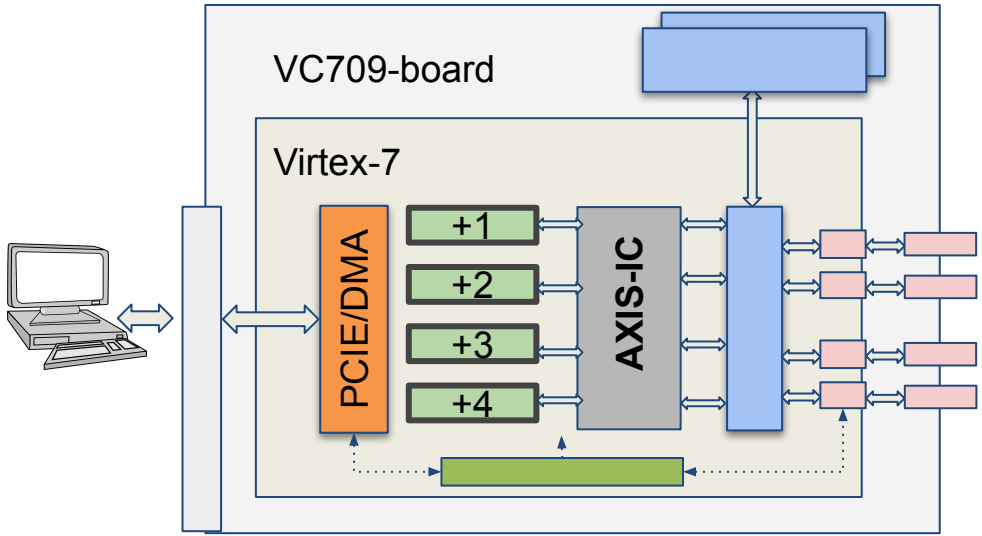
Create pool of threads



Single node: How it works

```
#pragma omp parallel{
  #pragma omp single{
    for(int i=0; i < 4 ; i++){
      #pragma omp target device(VC709 + i)\\
       map(tofrom: A[:N])                  \\
       depend(inout: A[:N]) nowait{
        for(int j = 1; j <= N ; j++ )
          A[j] = A[j] + i;
      }
    }
  }
}
```

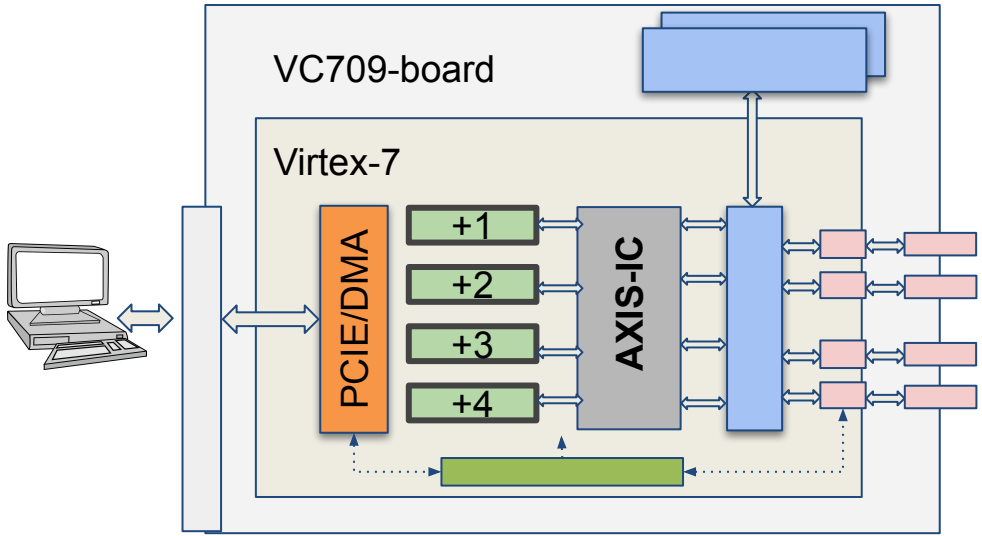
Select the control thread



Single node: How it works

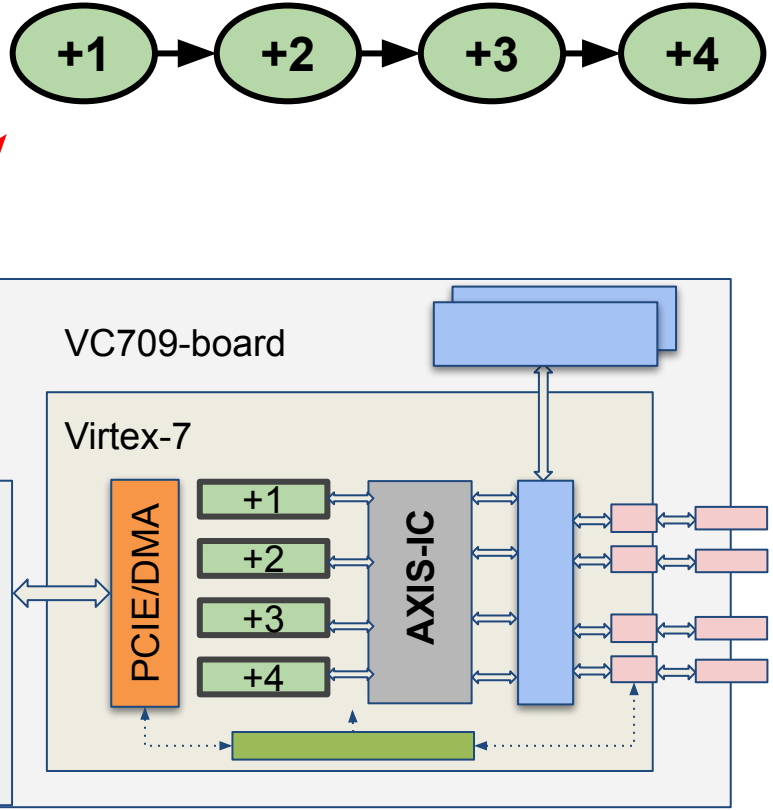
```
#pragma omp parallel{
  #pragma omp single{
    for(int i=0; i < 4 ; i++){
      #pragma omp target device(VC709 + i)\\
       map(tofrom: A[:N])                  \\
       depend(inout: A[:N]) nowait{
        for(int j = 1; j <= N ; j++ )
          A[j] = A[j] + i;
      }
    }
  }
}
```

create the tasks



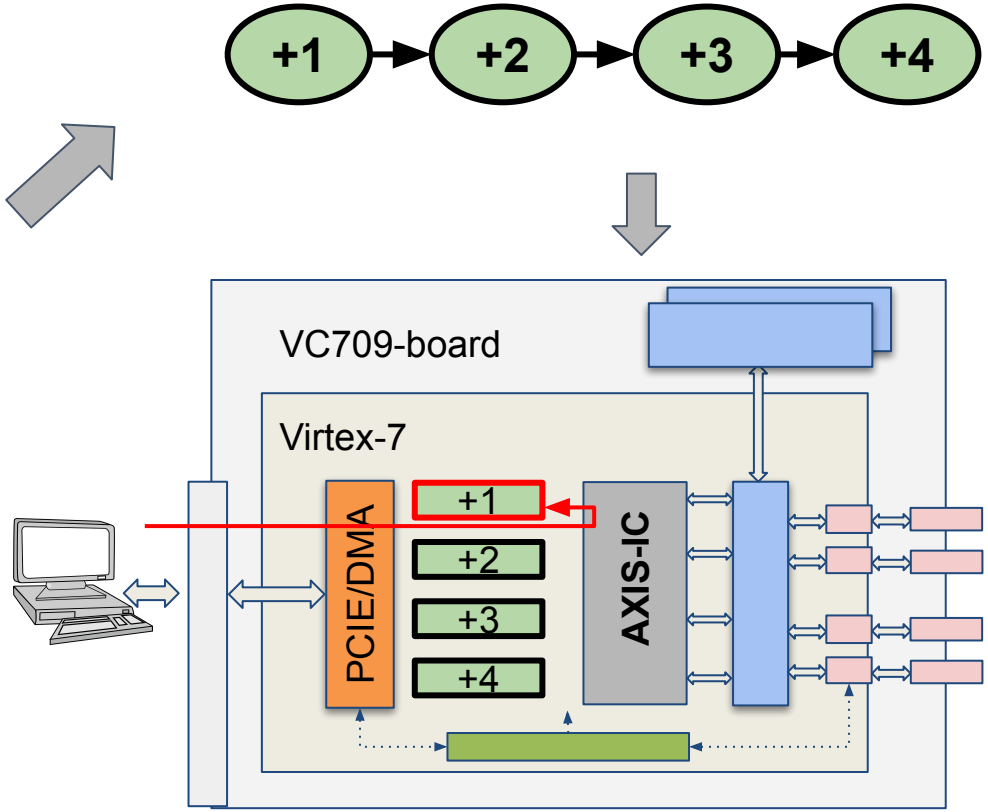
Single node: How it works

```
#pragma omp parallel{
  #pragma omp single{
    for(int i=0; i < 4 ; i++){
     #pragma omp target device(VC709 + i)\\
      map(tofrom: A[:N])                  \\
      depend(inout: A[:N]) nowait{
       for(int j = 1; j <= N ; j++ )
         A[j] = A[j] + i;
     }
    }
  }
}
```



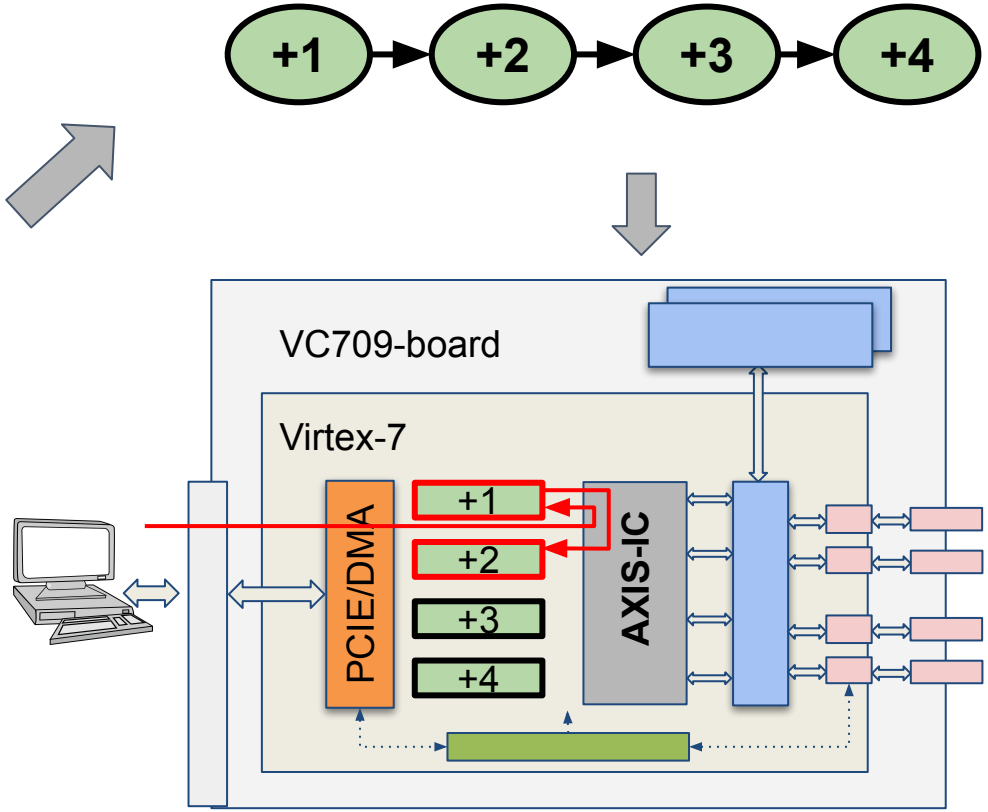
Single node: How it works

```
#pragma omp parallel{
  #pragma omp single{
    for(int i=0; i < 4 ; i++){
      #pragma omp target device(VC709 + i)\\
       map(tofrom: A[:N])                  \\
       depend(inout: A[:N]) nowait{
        for(int j = 1; j <= N ; j++ )
          A[j] = A[j] + i;
      }
    }
  }
}
```



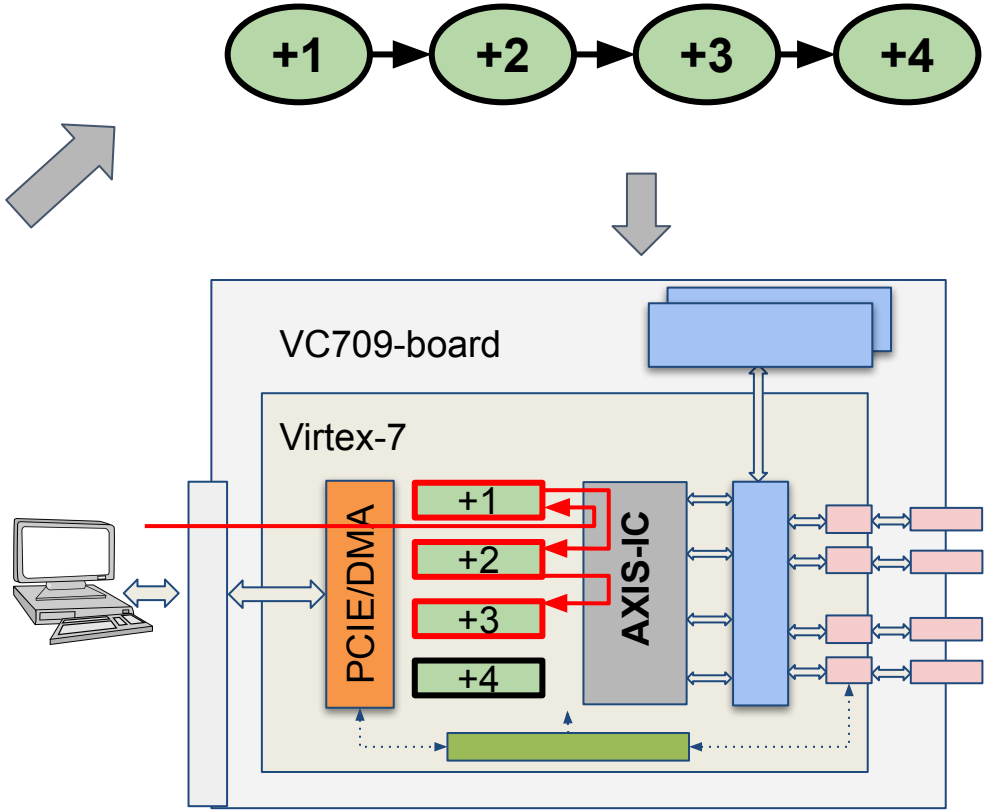
Single node: How it works

```
#pragma omp parallel{
  #pragma omp single{
    for(int i=0; i < 4 ; i++){
      #pragma omp target device(VC709 + i)\\
       map(tofrom: A[:N])                  \\
       depend(inout: A[:N]) nowait{
        for(int j = 1; j <= N ; j++ )
          A[j] = A[j] + i;
      }
    }
  }
}
```



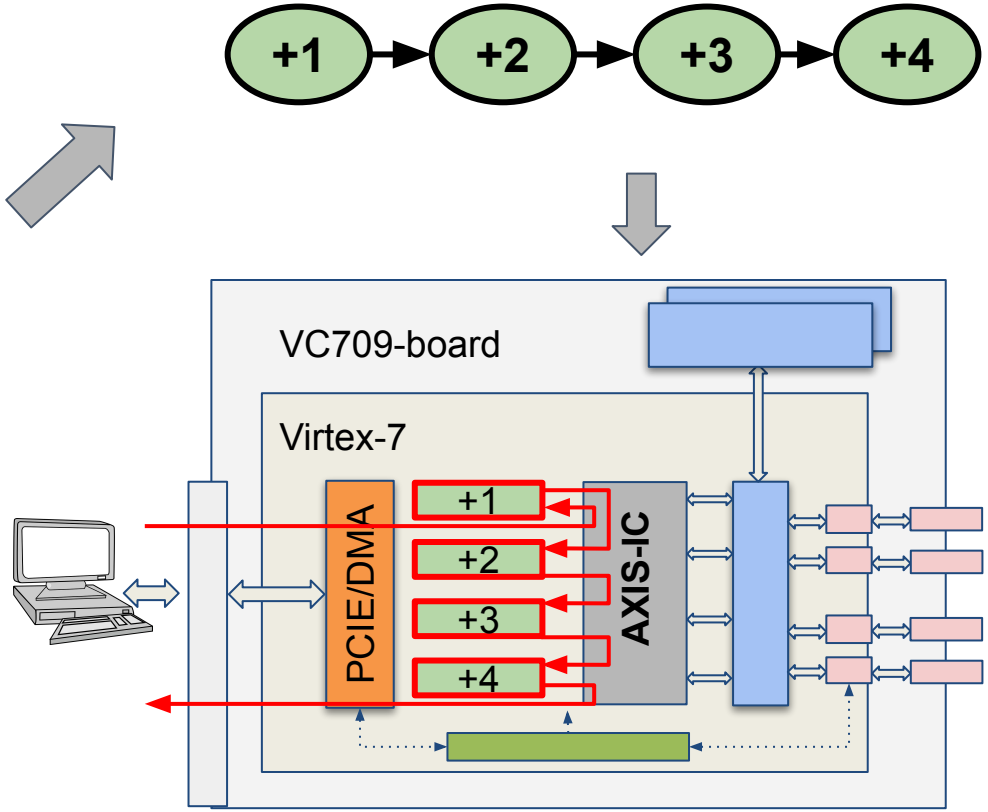
Single node: How it works

```
#pragma omp parallel{
  #pragma omp single{
    for(int i=0; i < 4 ; i++){
      #pragma omp target device(VC709 + i)\\
       map(tofrom: A[:N])                  \\
       depend(inout: A[:N]) nowait{
        for(int j = 1; j <= N ; j++ )
          A[j] = A[j] + i;
      }
    }
  }
}
```



Single node: How it works

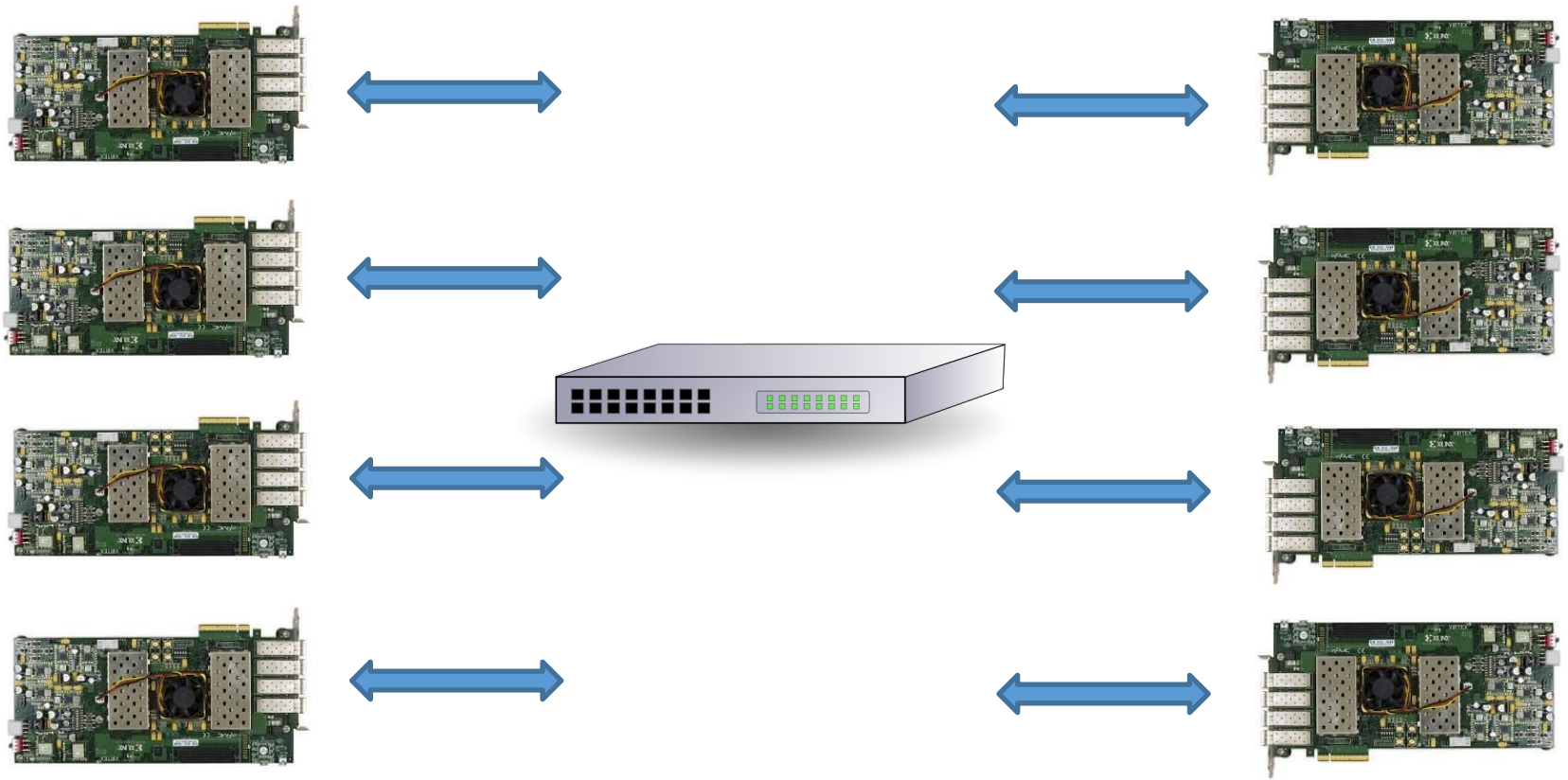
```
#pragma omp parallel{
  #pragma omp single{
    for(int i=0; i < 4 ; i++){
      #pragma omp target device(VC709 + i)\\
       map(tofrom: A[:N])                  \\
       depend(inout: A[:N]) nowait{
        for(int j = 1; j <= N ; j++ )
          A[j] = A[j] + i;
      }
    }
  }
}
```



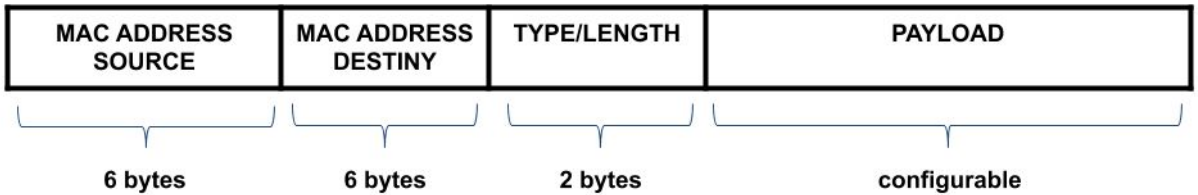
The Hardware

Multiple Nodes

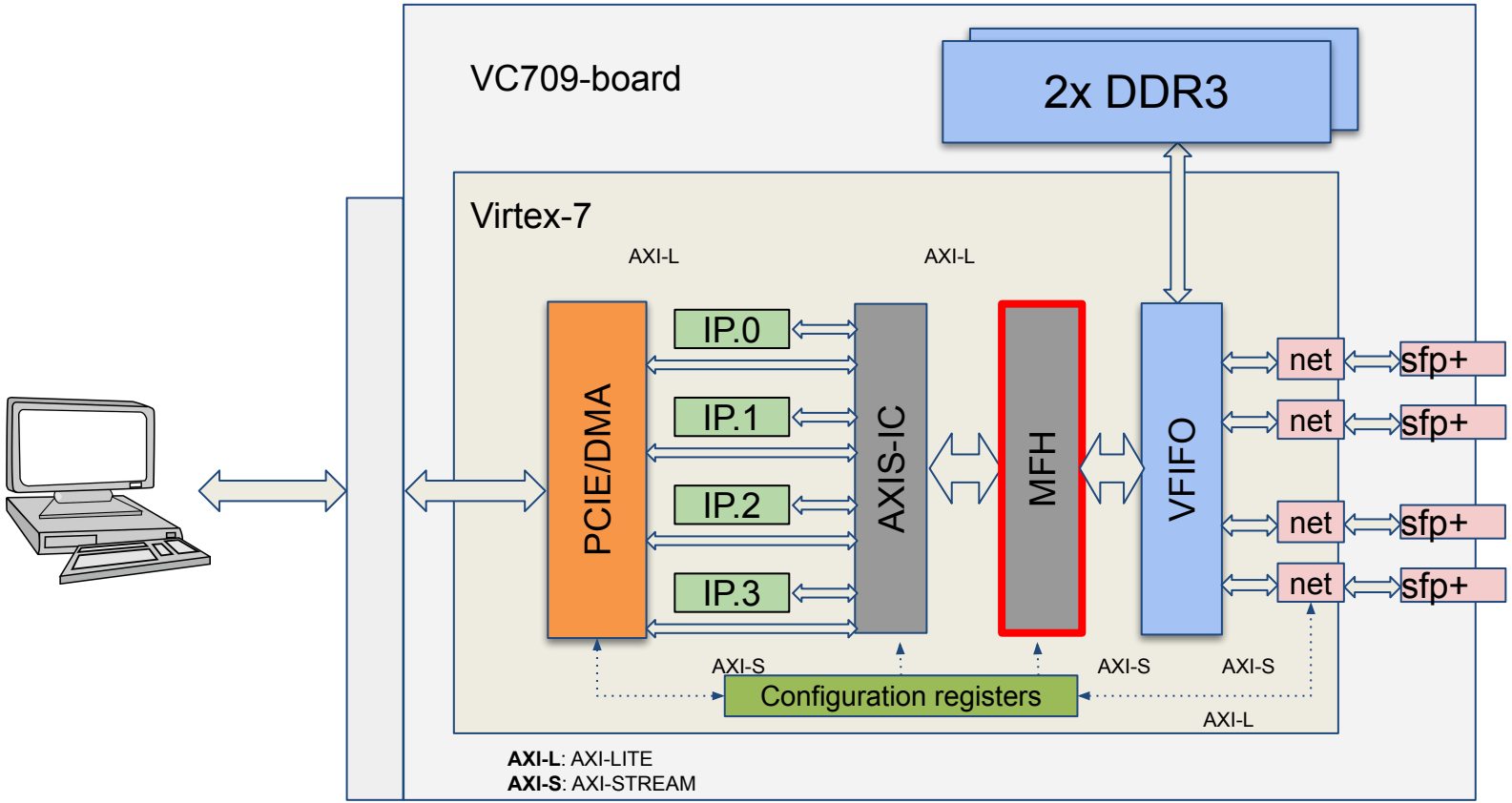
Multiple Nodes



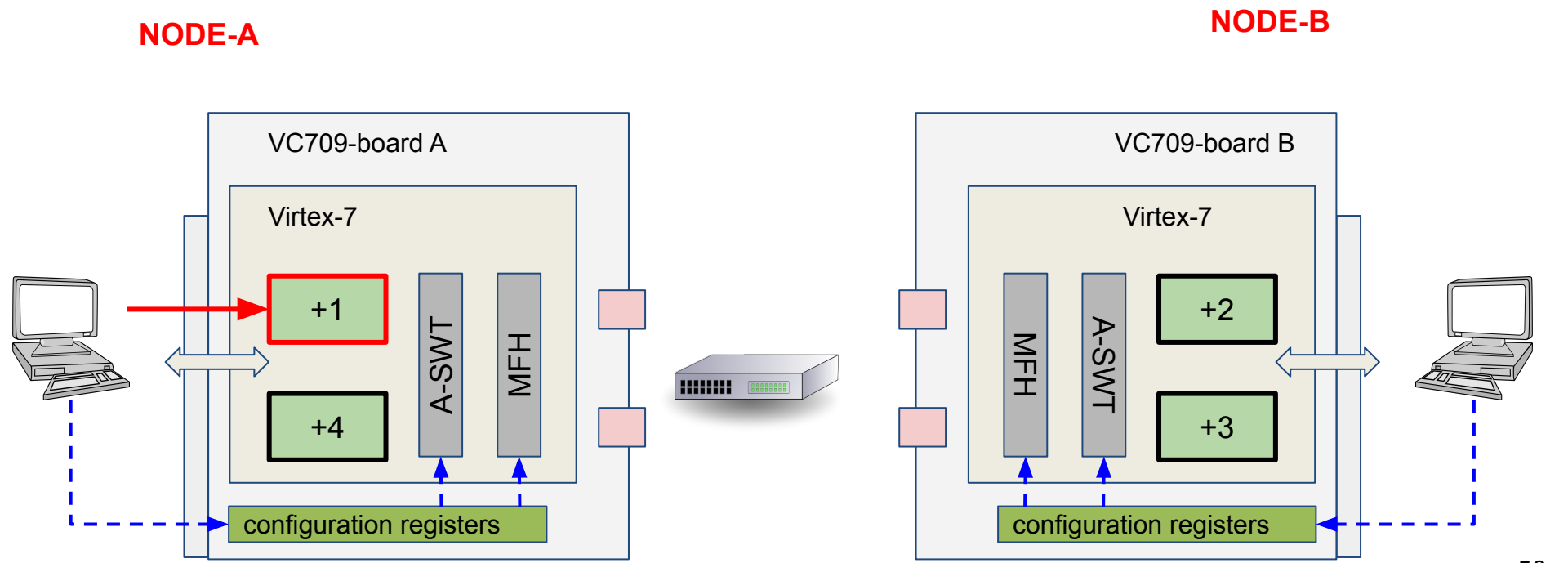
- Need to forward dependency data across optical links
  - Designed a MAC Frame Handler (MFH)
  - Mount and unmount a MAC frame.



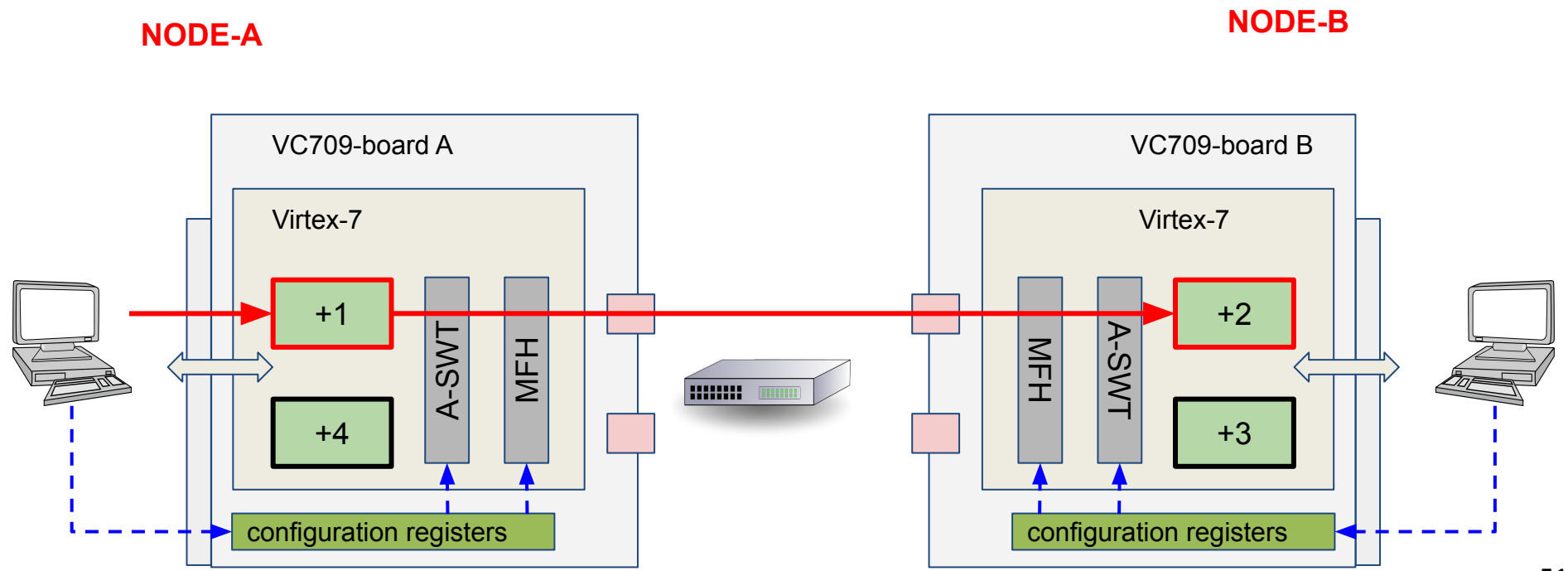
Adding MFH


VC709-board
2x DDR3
Virtex-7
AXI-L
AXI-L
PCIE/DMA
IP.0
IP.1
IP.2
IP.3
AXIS-IC
MFH
VFIFO
net
net
net
net
sfp+
sfp+
sfp+
sfp+
AXI-S
AXI-S
AXI-S
Configuration registers
AXI-L
AXI-L: AXI-LITE
AXI-S: AXI-STREAM

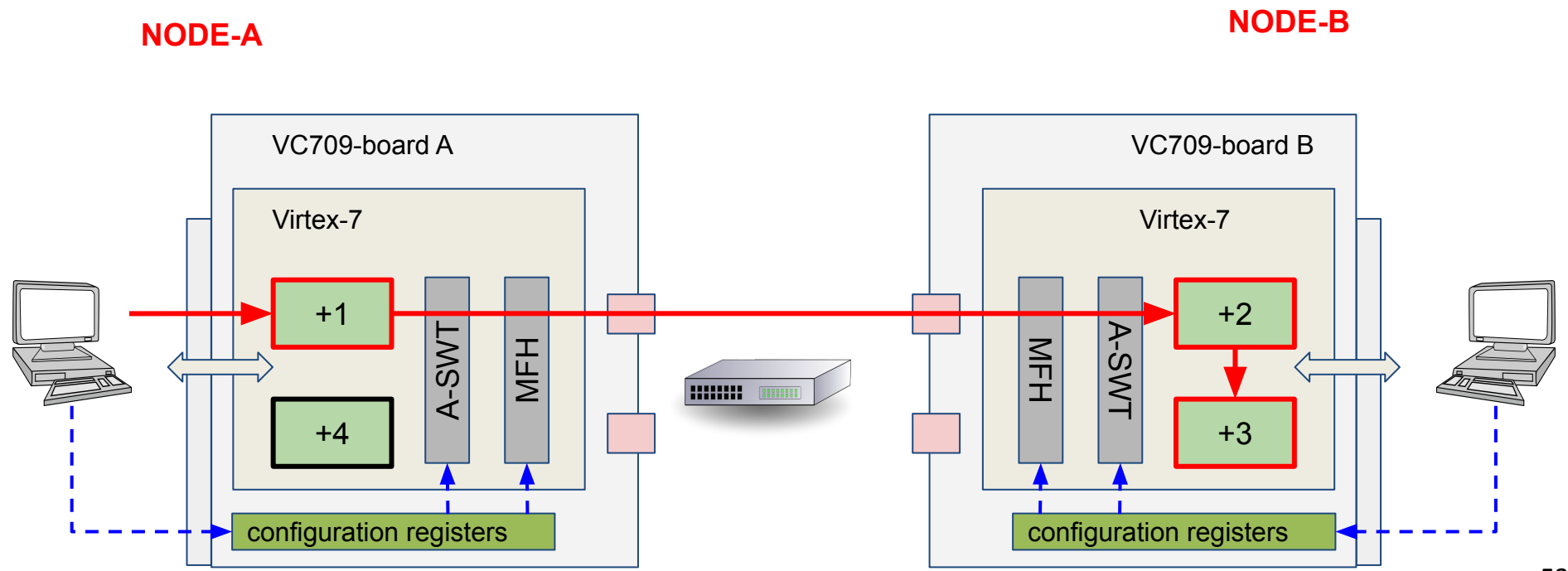
Multiple nodes: How it works



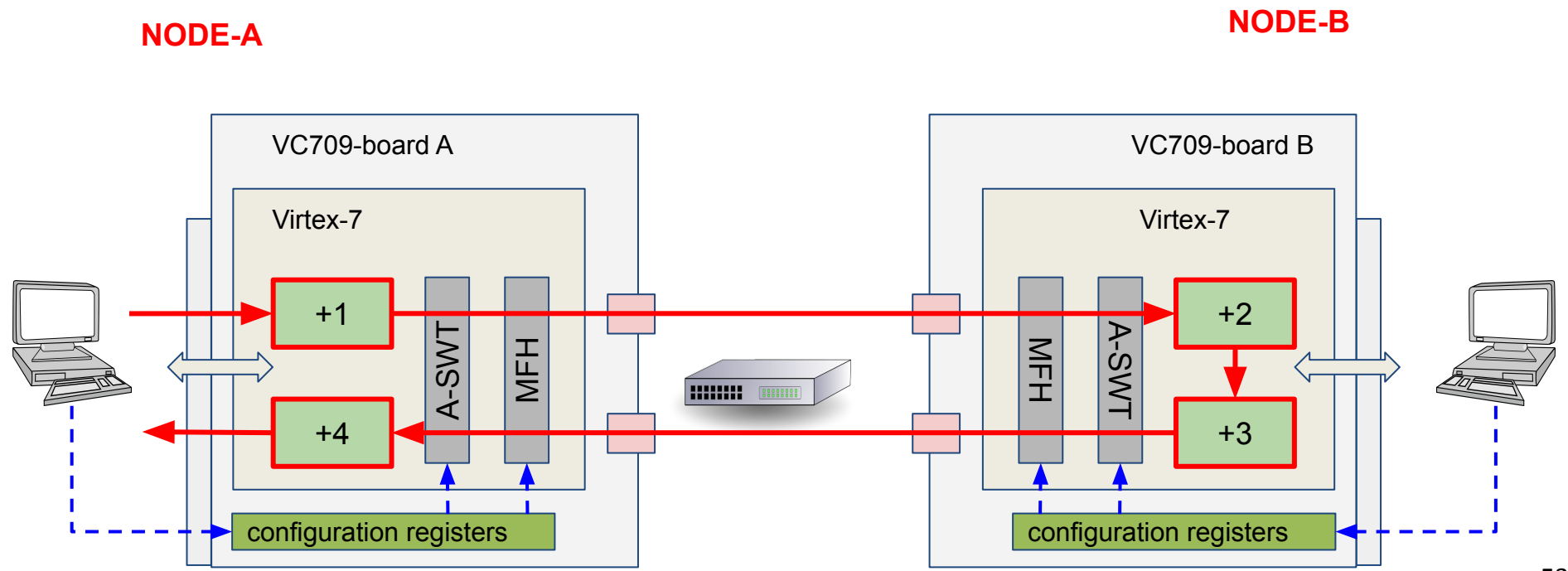
Multiple nodes: How it works



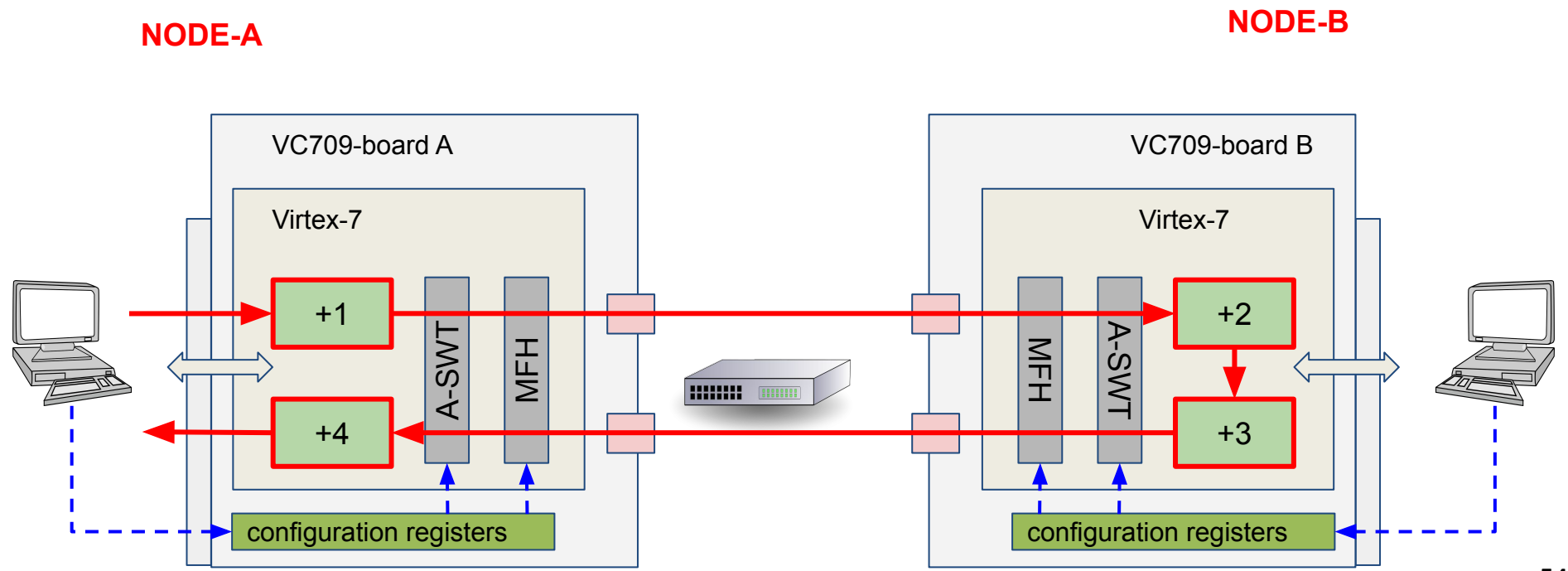
Multiple nodes: How it works



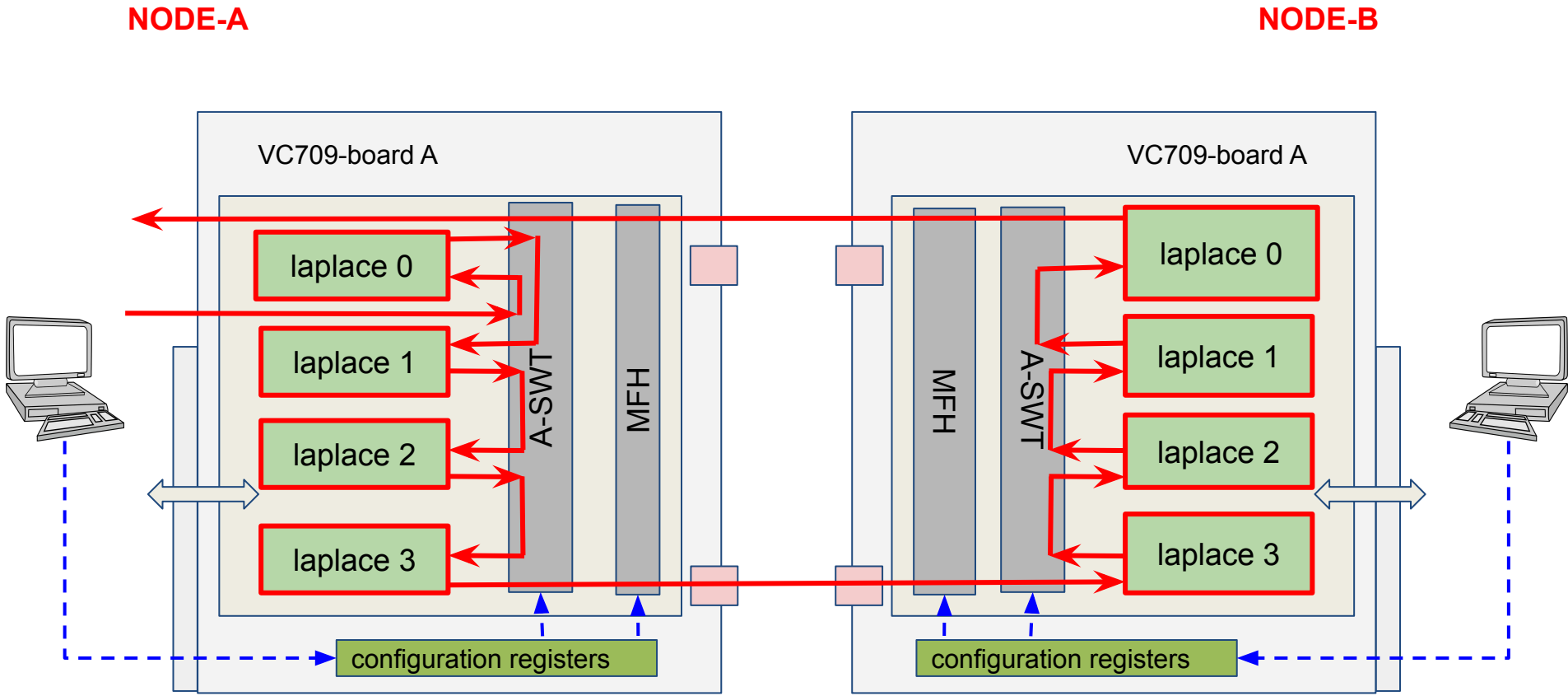
Multiple nodes: How it works



Multiple nodes: How it works



Ongoing Experiments



Related Works

OpenMP for FPGAs

	Year	Multi-FPGA	Compiler ToolKit	Target System
Leow	2006	no	C-Breeze	Celoxica RC100 Board-Spartan
Cabrera	2009	no	Mercurium, GCC, SGI RASClib	SGI RASC 2.2 Board-Virtex 4
Cilardo	2013	no	Custom OpenMP, Xilinx EDK	Xilinx EDK Boards
Choi	2013	no	LLVM GCC	Altera Board with Stratix IV
Filgueras	2014	no	Mercurium, Nanos++	Xilinx Board with Zynq-700
Podobas	2014	no	Custom C89 Compiler	Altera ED5 Stratix V
Sommer	2017	no	Clang, LLVM, libomptarget, TPC	Xilinx VC709 Board
Ceissler (FCCM)	2018	no	Clang, LLVM, libomptarget	Intel HARP2 and Amazon AWS
Bosh	2018	no	Mercurium, Nanos++	Zynq Ultrascale+
Knaust	2019	no	Clang LLVM	Intel Board Arria 10 GX
Our work	2021	yes	Clang LLVM, libomptarget	Cluster of Xilinx VC709 Boards

OpenMP for FPGAs

	Year	Multi-FPGA	Compiler ToolKit	Target System
Leow	2006	no	C-Breeze	Celoxica RC100 Board-Spartan
Cabrera	2009	no	Mercurium, GCC, SGI RASClib	SGI RASC 2.2 Board-Virtex 4
Cilardo	2013	no	Custom OpenMP, Xilinx EDK	Xilinx EDK Boards
Choi	2013	no	LLVM GCC	Altera Board with Stratix IV
Filgueras	2014	no	Mercurium, Nanos++	Xilinx Board with Zynq-700
Podobas	2014	no	Custom C89 Compiler	Altera ED5 Stratix V
Sommer	2017	no	Clang, LLVM, libomptarget, TPC	Xilinx VC709 Board
Ceissler (FCCM)	2018	no	Clang, LLVM, libomptarget	Intel HARP2 and Amazon AWS
Bosh	2018	no	Mercurium, Nanos++	Zynq Ultrascale+
Knaust	2019	no	Clang LLVM	Intel Board Arria 10 GX
Our work	2021	yes	Clang LLVM, libomptarget	Cluster of Xilinx VC709 Boards

Thank you!

email: ramon.nepomuceno@ic.unicamp.br