XEON PHI BASICS

Reusing this material

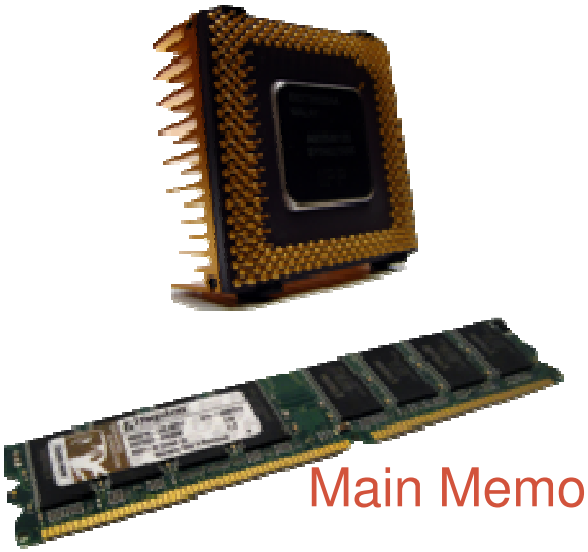
epcc | THE UNIVERSITY OF EDINBURGH

LESSON PLAN

- Programming models
- Parallelisation
- Compilers and Tools
- Performance Considerations

Programming models

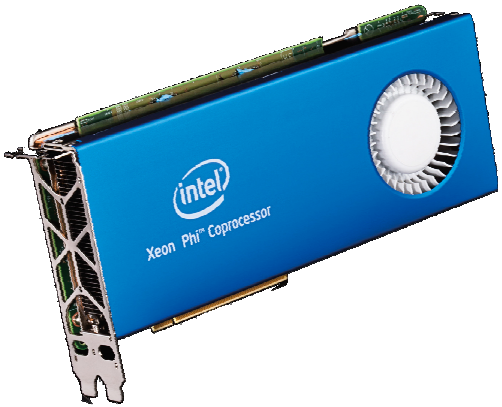
Host



Main Memory

+

Coprocessor

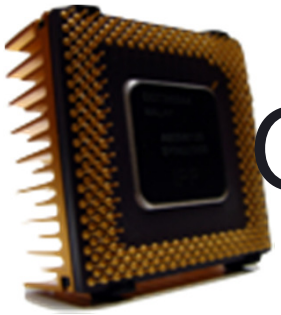


3 Basic Programming Models

Host

Native mode

Coprocessor

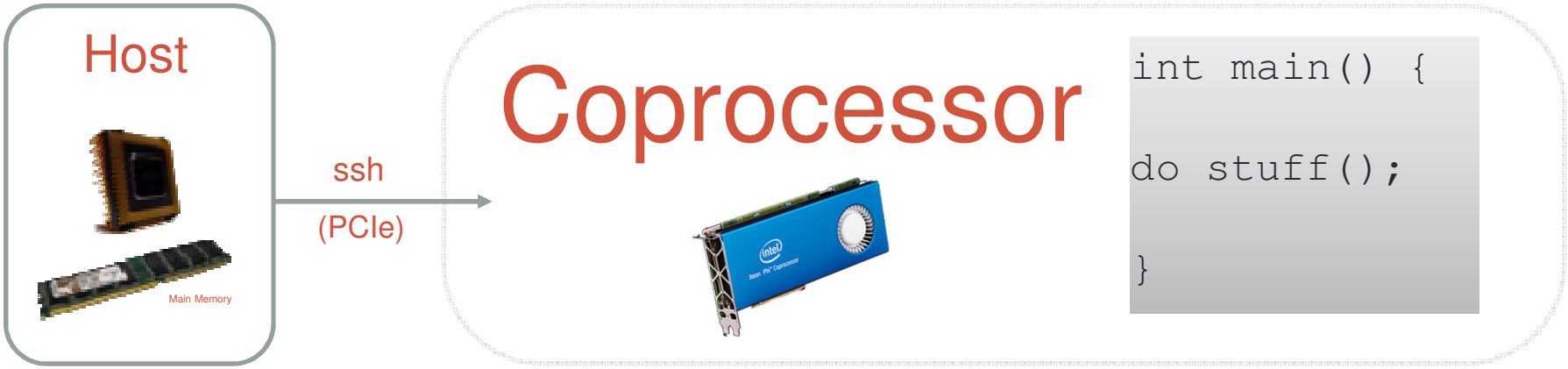


Offload

Symmetric execution

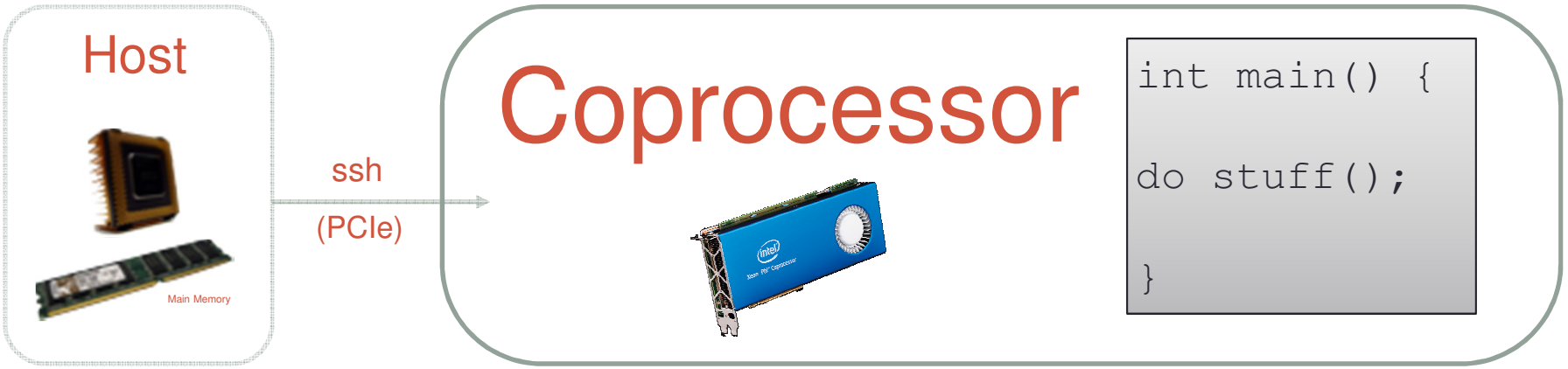
Main Memory

Native Mode: Xeon Phi only



- Host used for preparation work (e.g. compiling, data copy)
- User initiates run from host or can use host to connect to Xeon Phi via ssh

Native Mode: Xeon Phi only



- Host used for preparation work (e.g. compiling, data copy)
- User initiates run from host or can use host to connect to Xeon Phi via ssh
- **Programme runs on Xeon Phi from start to finish "as usual"**

Native Mode: Xeon Phi only

Pros:

- Requires minimal effort to “port”
- Works well with ‘flat profile’ applications
- No memory copy required

Native Mode: Xeon Phi only

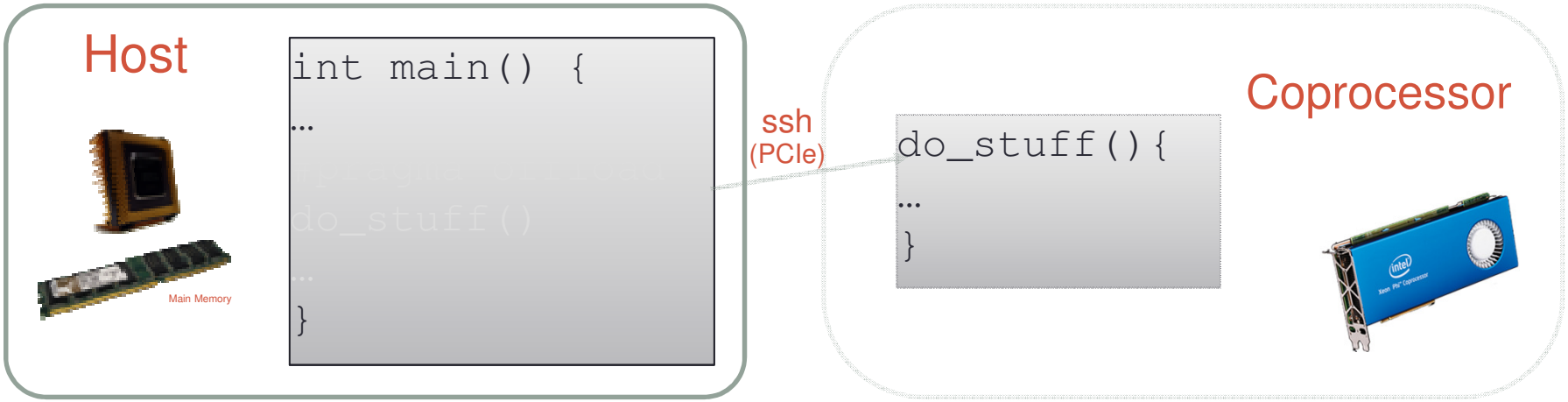
Pros:

- Requires minimal effort to “port”
- Works well with ‘flat profile’ applications
- No memory copy required

Cons:

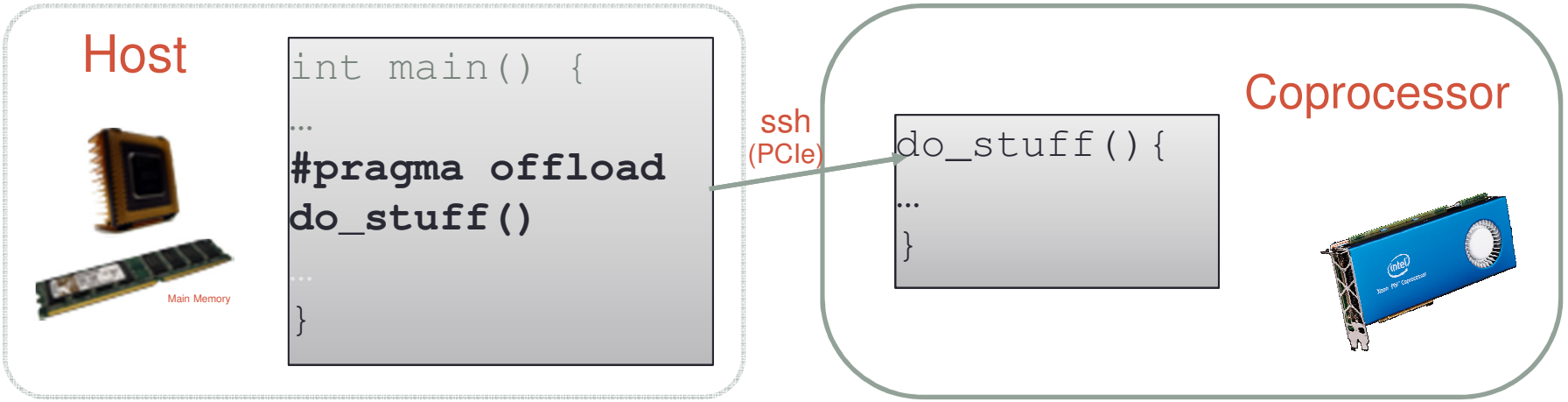
- Poor performance on codes with large serial regions and ‘complex codes’
- Limited Xeon Phi memory

Offload Execution: Hotspot eliminator



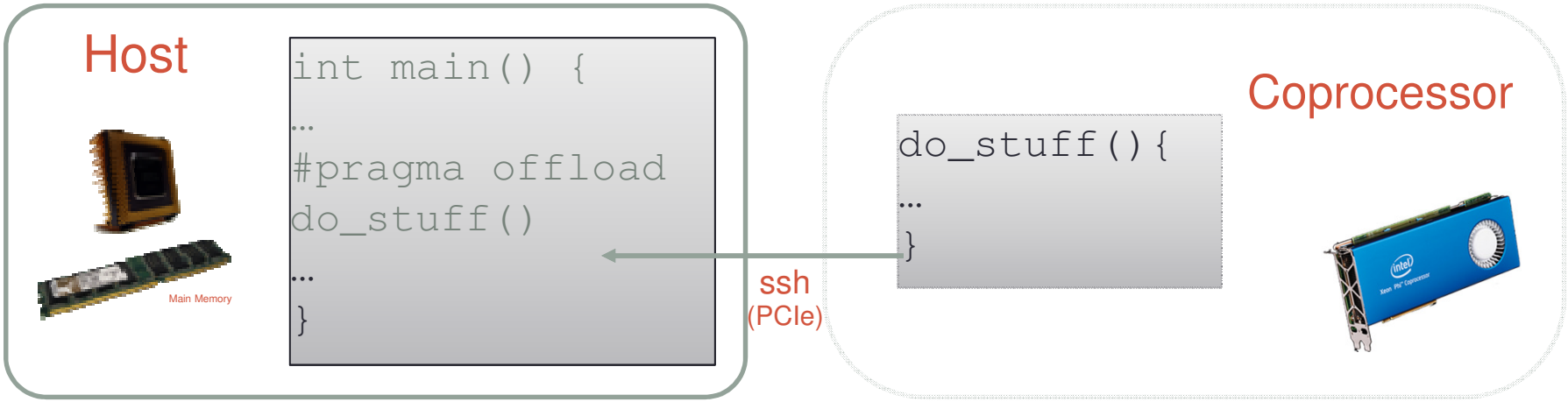
- **Application is initiated on host**

Offload Execution: Hotspot eliminator



- Application is initiated on host
- **Embarrassingly parallel hotspots are offloaded to Xeon Phi**

Offload Execution: Hotspot eliminator



- Application is initiated on host
- Embarrassingly parallel hotspots are offloaded to Xeon Phi
- **Results of offload region are returned to host where execution continues**

Offload Execution: Hotspot eliminator

Pros:

- Serial code handled by advanced CPU cores
- Embarrassingly parallel hotspots are executed efficiently on Xeon Phi
- More efficient use of (limited) Xeon Phi memory

Offload Execution: Hotspot eliminator

Pros:

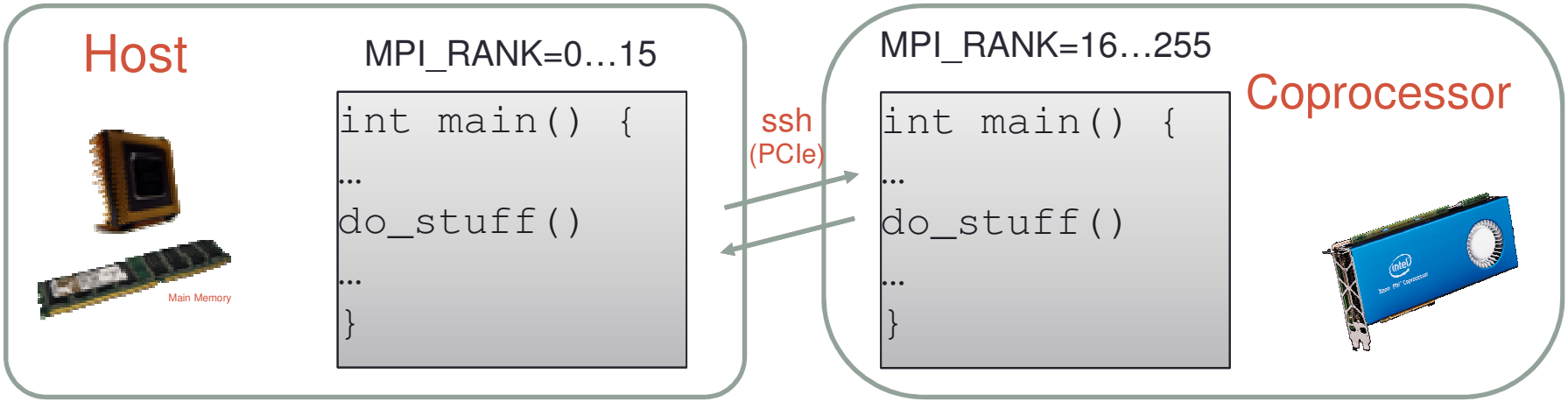
- Serial code handled by advanced CPU cores
- Embarrassingly parallel hotspots are executed efficiently on Xeon Phi
- More efficient use of (limited) Xeon Phi memory

Cons:

- Data must be copied to and from the Xeon Phi via (slow) PCIe Bus
- May lead to poor utilisation of CPU/XeonPhi (idle time)

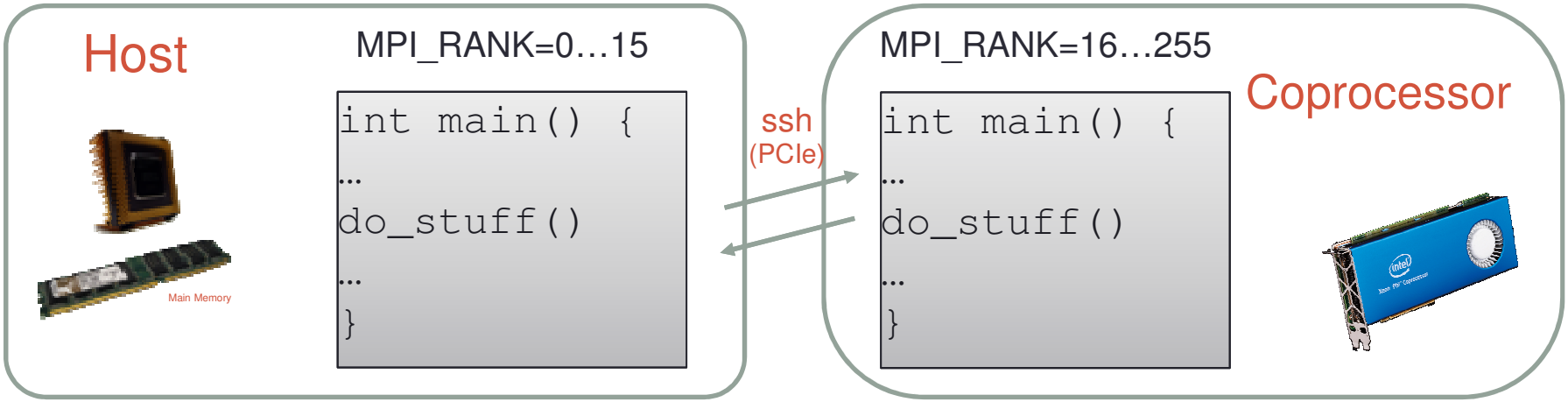
Can be alleviated by asynchronous execution and memory copies

Symmetric Execution: Phi-as-a-node



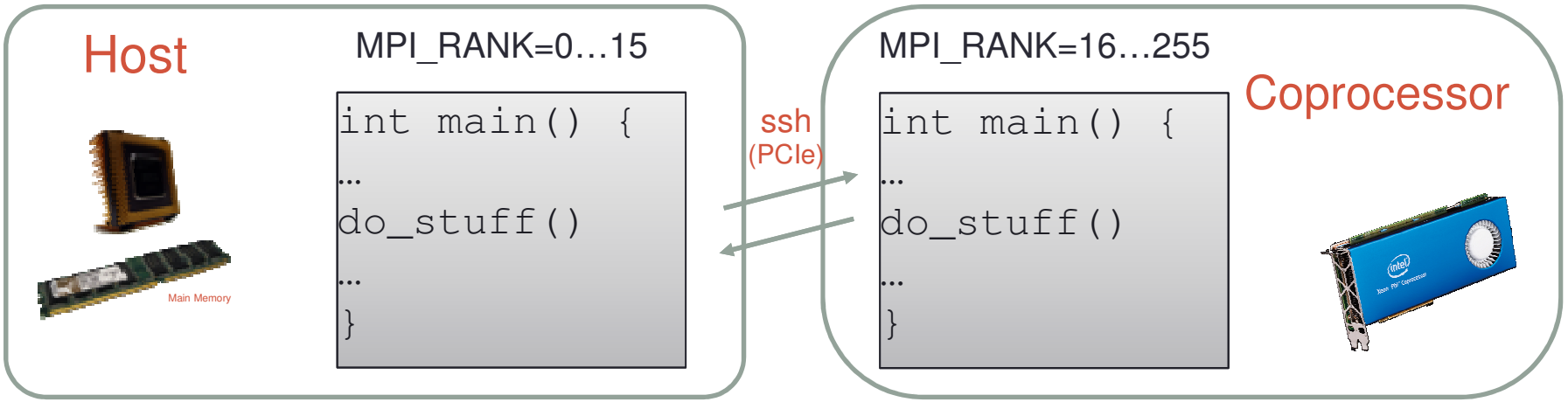
- **Application is initiated on host but…**

Symmetric Execution: Phi-as-a-node



- Application is initiated on host but…
- **Runs across both CPU and Xeon Phi cores**

Symmetric Execution: Phi-as-a-node



- Application is initiated on host but…
- Runs across both CPU and Xeon Phi cores
- **Effectively using Xeon Phi as just another node for MPI to use**

Symmetric Execution: Phi-as-a-node

Pros:

- Promise of full hardware utilisation
- No need for offloading pragmas and memory copies

Symmetric Execution: Phi-as-a-node

Pros:

- Serial code handled by advanced CPU cores
- Embarrassingly parallel hotspots are executed efficiently on Xeon Phi
- More efficient use of (limited) Xeon Phi memory

Cons:

- Tricky load-balancing
- Code is rarely optimal for both CPU and Xeon Phi

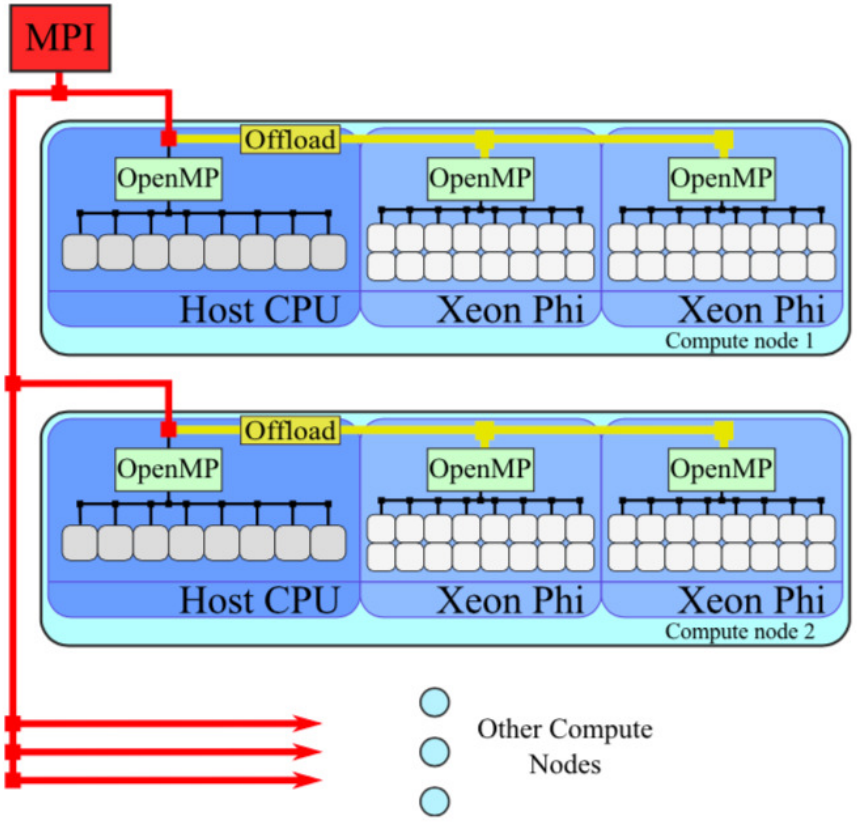
Parallelisation

MPI

and / or

OpenMP

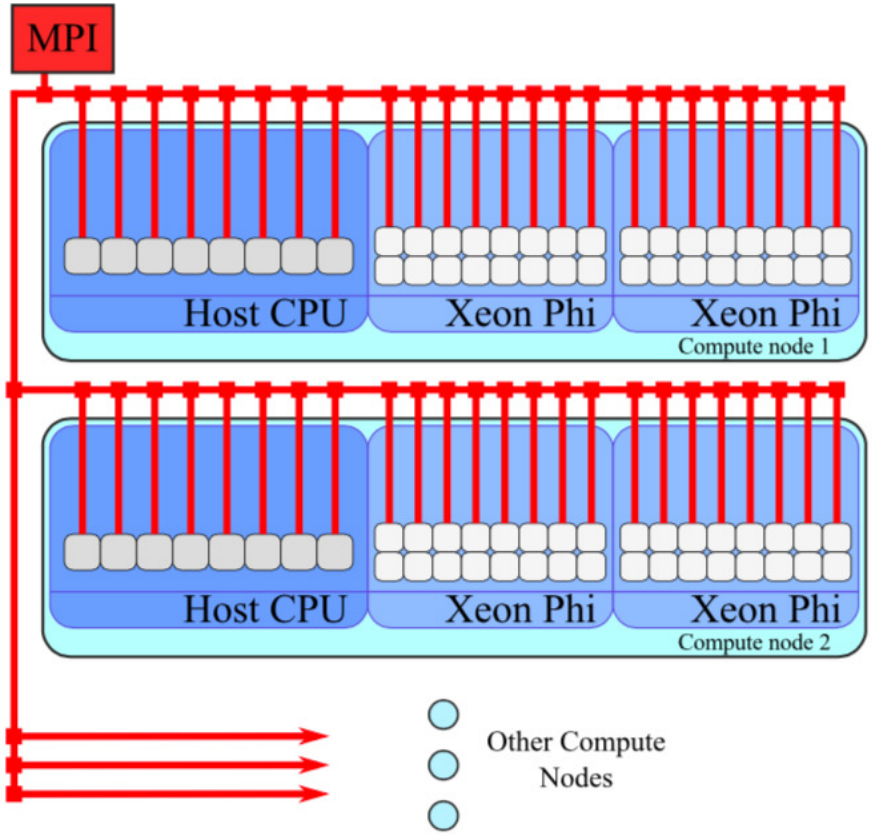
MPI+OpenMP with Offload



- MPI runs only on hosts
- MPI processes offload to Xeon Phi
- OpenMP in MPI processes
- OpenMP in offload regions

Image from Colfax training material

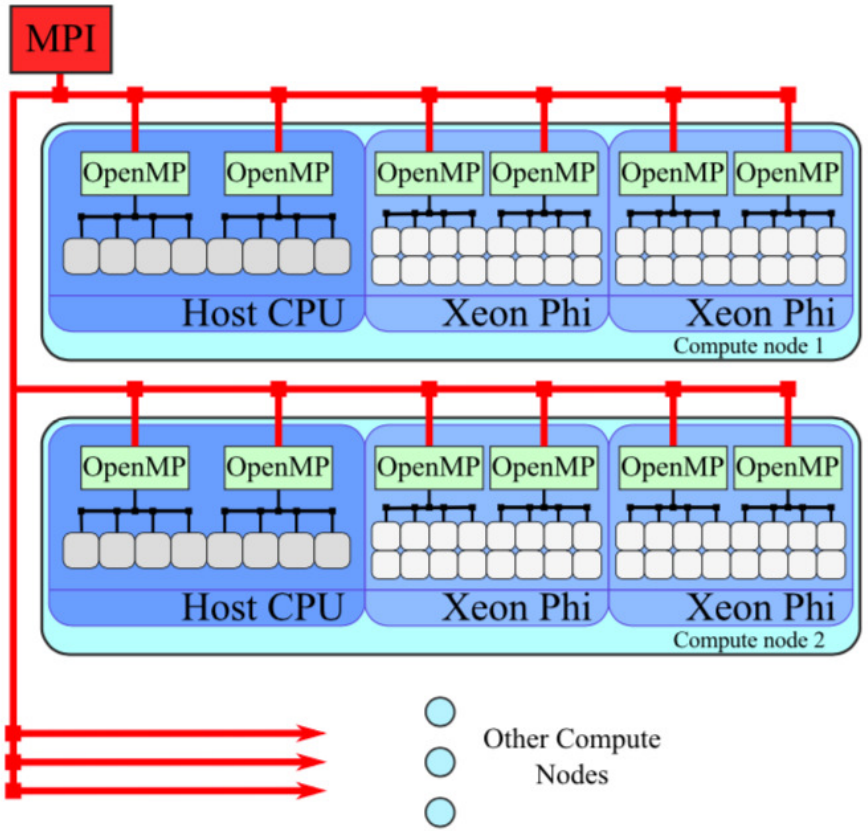
Symmetric Pure MPI



- MPI processes on host
- MPI processes (native) on Xeon Phi
- No OpenMP

Image from Colfax training material

Symmetric hybrid MPI+OpenMP



- MPI processes on host
- MPI processes (native) on Xeon Phi
- All MPI processes use OpenMP multithreading

Image from Colfax training material

What is best?

- What is your goal?
- What is your system?
- What is your application?

- Generally OpenMP faster than MPI on Xeon Phi
  - Poor performance of MPI on Xeon Phi
  - Less memory (especially important on Xeon Phi)

- Worth checking affinity settings (more later)

Compilers & Tools

Compilers

In a word: **Intel**

Compilers

In a word: **Intel**

- Intel **C** Compiler
- Intel **C++** Compiler
- Intel **Fortran** Compiler

Tools

In two words:

Intel & Allinea

(but mainly Intel)

Tools

Intel Parallel Studio XE

- Intel C, C++ and Fortran compilers (MIC-capable)
- Intel Math Kernel Library (MKL)
- Intel MPI Library (only in Cluster Edition)
- Intel Trace Analyzer and Collector / ITAC (MPI profiler)
- Intel VTune Amplifier XE (multi-threaded profiler)
- Intel Inspector XE (memory and threading debugging)
- Intel Threading Building Blocks / TBB (threading library)
- Intel Performance Primitives / IPP (media and data)
- Intel Advisor XE (guided parallelism design)

Allinea

- Map (lightweight profiler)
- DDT (debug)
- Forge (unified UI for DDT & Map)

Tools → Runtime

Tools → Runtime

MPSS

(Intel Manycore Platform Software Stack)

Environment Variables

Linux Commands

Tools → Runtime

MPSS

- `micnativeloadex`
- `micinfo`
- `micccheck`
- `micsmc` (GUI)
- `micrasd` (root)

…

Environment Variables

- `MKL_MIC_ENABLE`
- `MIC_ENV_PREFIX`
- `MIC_LD_LIBRARY_PATH`
- `I_MPI_MIC`
- `I_MPI_MIC_POSTFIX`
- `OFFLOAD_REPORT`
- `KMP_AFFINITY`
- `KMP_BLOCKTIME`
- `MIC_USE_2MB_BUFFERS`

…

Linux Commands

- `lspci | grep Phi`
- `cat /etc/hosts | grep mic`
- `cat /proc/cpuinfo | grep proc | tail -n 3`

…

For more details:

http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/xeon-phi-software-configuration-users-guide.pdf

https://software.intel.com/sites/products/documentation/doclib/iss/2013/compiler/cpp-lin/GUID-E1EC94AE-A13D-463E-B3C3-6D7A7205F5A1.htm

Performance Considerations

Four things to consider first:

Execution mode

Vectorisation

Alignment

Affinity

Application Design

Mode of execution

- Native
- Offload
- Symmetric

Mode chosen should depend on the application and system configuration (as discussed previously)

Vectorisation

- **Xeon Phi performance is greatly dependant on vector units.**
- Intel Xeon CPUs also use (smaller) vector units → Code optimised for Intel Xeon will run faster on Intel Xeon Phi
- KNL (next generation Xeon Phi) will also use 512-AVX vector units → Code optimised for Intel Xeon Phi KNC will also run faster on Intel Xeon Phi KNL

*(KNC-KNL not binary compatible)

Vectorisation

- Xeon Phi performance is greatly dependant on vector units.
- **Intel Xeon CPUs also use (smaller) vector units → Code optimised for Intel Xeon will run faster on Intel Xeon Phi**
- KNL (next generation Xeon Phi) will also use 512-AVX vector units → Code optimised for Intel Xeon Phi KNC will also run faster on Intel Xeon Phi KNL

*(KNC-KNL not binary compatible)

Vectorisation

- Xeon Phi performance is greatly dependant on vector units.
- Intel Xeon CPUs also use (smaller) vector units → Code optimised for Intel Xeon will run faster on Intel Xeon Phi
- **KNL (next generation Xeon Phi) will also use 512-AVX vector units → Code optimised for Intel Xeon Phi KNC will also run faster on Intel Xeon Phi KNL**

*(KNC-KNL not binary compatible)

Data Alignment

- **"Loop is vectorised" != faster**
  - Data alignment is critical for vectorisation to be beneficial
  - Remember to not only align data, but also to tell the compiler that data is aligned at loop.

Data Alignment

- "Loop is vectorised" != faster
- **Data alignment is critical for vectorisation to be beneficial**
- Remember to not only align data, but also to tell the compiler that data is aligned at loop.

Data Alignment

- "Loop is vectorised" != faster
- Data alignment is critical for vectorisation to be beneficial
- **Remember to not only align data, but also to tell the compiler that data is aligned at loop.**

Affinity

- **All data moves over high-speed ring interconnect**
  - Affinity critical for good performance
- Default settings are not always optimal
- In offload mode, may accidentally use poor settings.

e.g. 240 threads competing for the use of 30 cores, while 30 other cores are idle.

Affinity

- All data moves over high-speed ring interconnect
- **Affinity critical for good performance**
- Default settings are not always optimal
- In offload mode, may accidentally use poor settings.

e.g. 240 threads competing for the use of 30 cores, while 30 other cores are idle.

Affinity

- All data moves over high-speed ring interconnect
- Affinity critical for good performance
- **Default settings are not always optimal**
- In offload mode, may accidentally use poor settings.

e.g. 240 threads competing for the use of 30 cores, while 30 other cores are idle.

Affinity

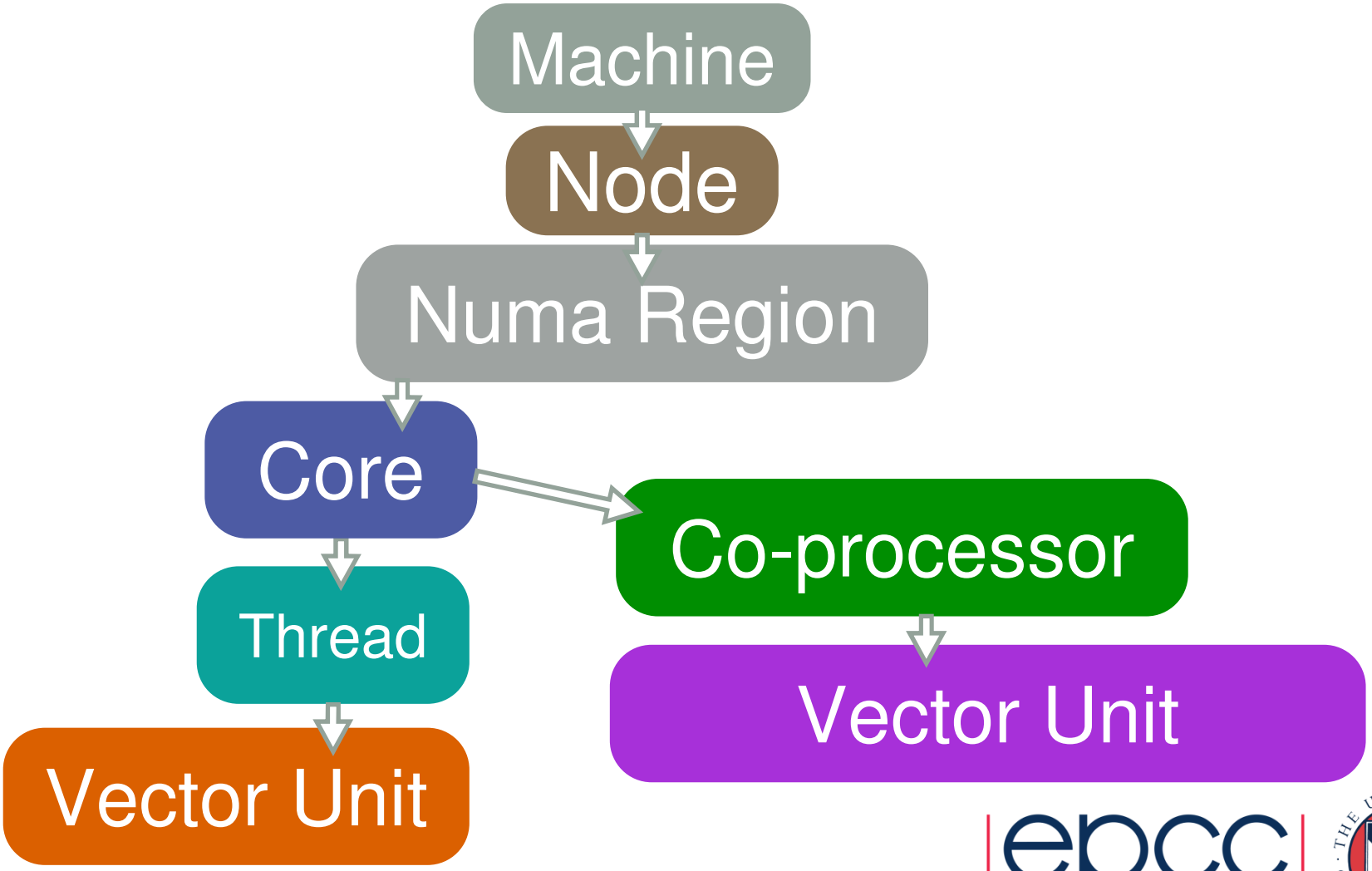
- All data moves over high-speed ring interconnect
- Affinity critical for good performance
- Default settings are not always optimal
- **In offload mode, may accidentally use poor settings.**

e.g. 240 threads competing for the use of 30 cores, while 30 other cores are idle.

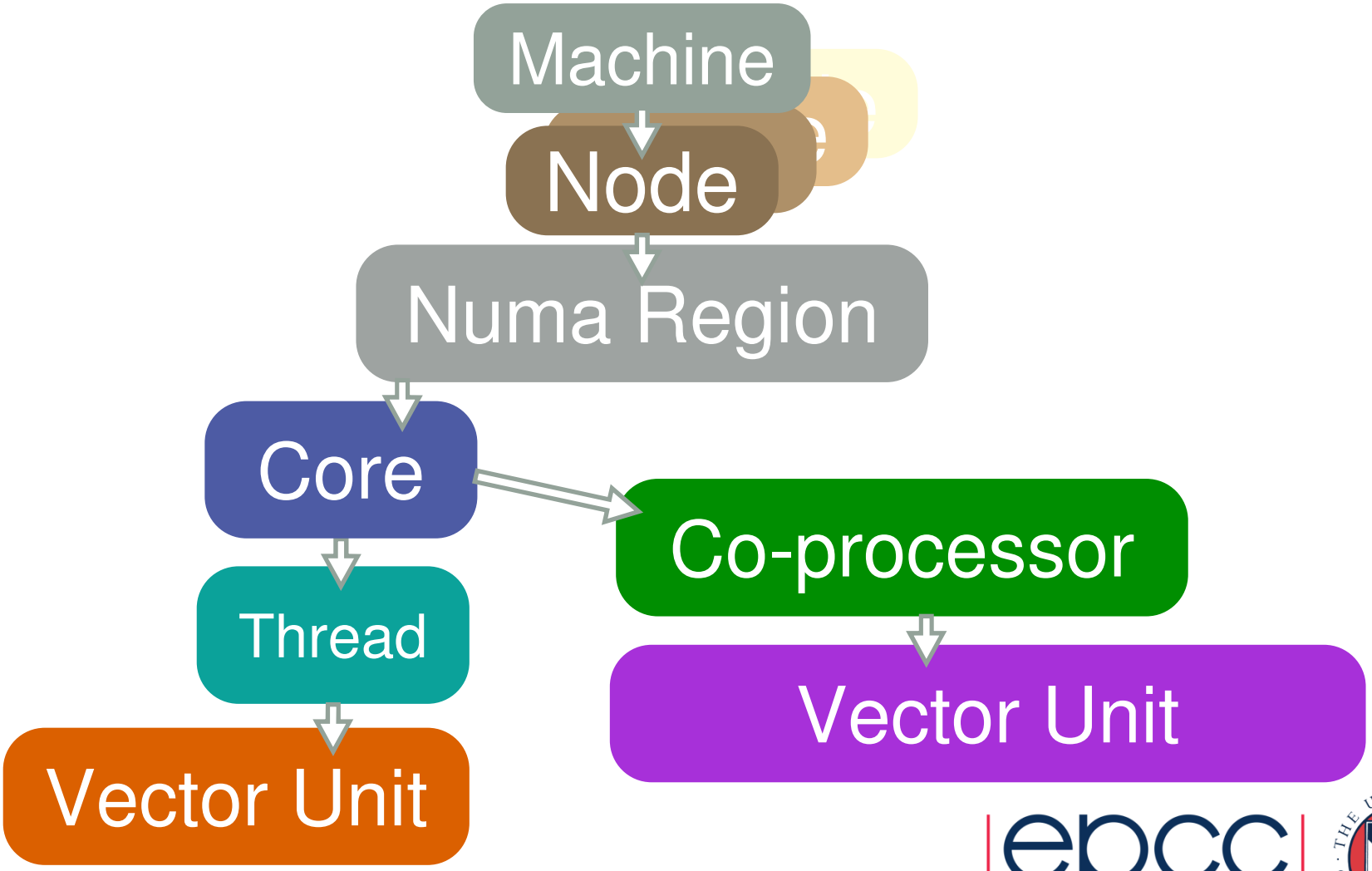
Application Design

- **Design** >> *Optimisation*
- Consider all levels of parallelism available and adapt your algorithm to exploit as many and as much as possible

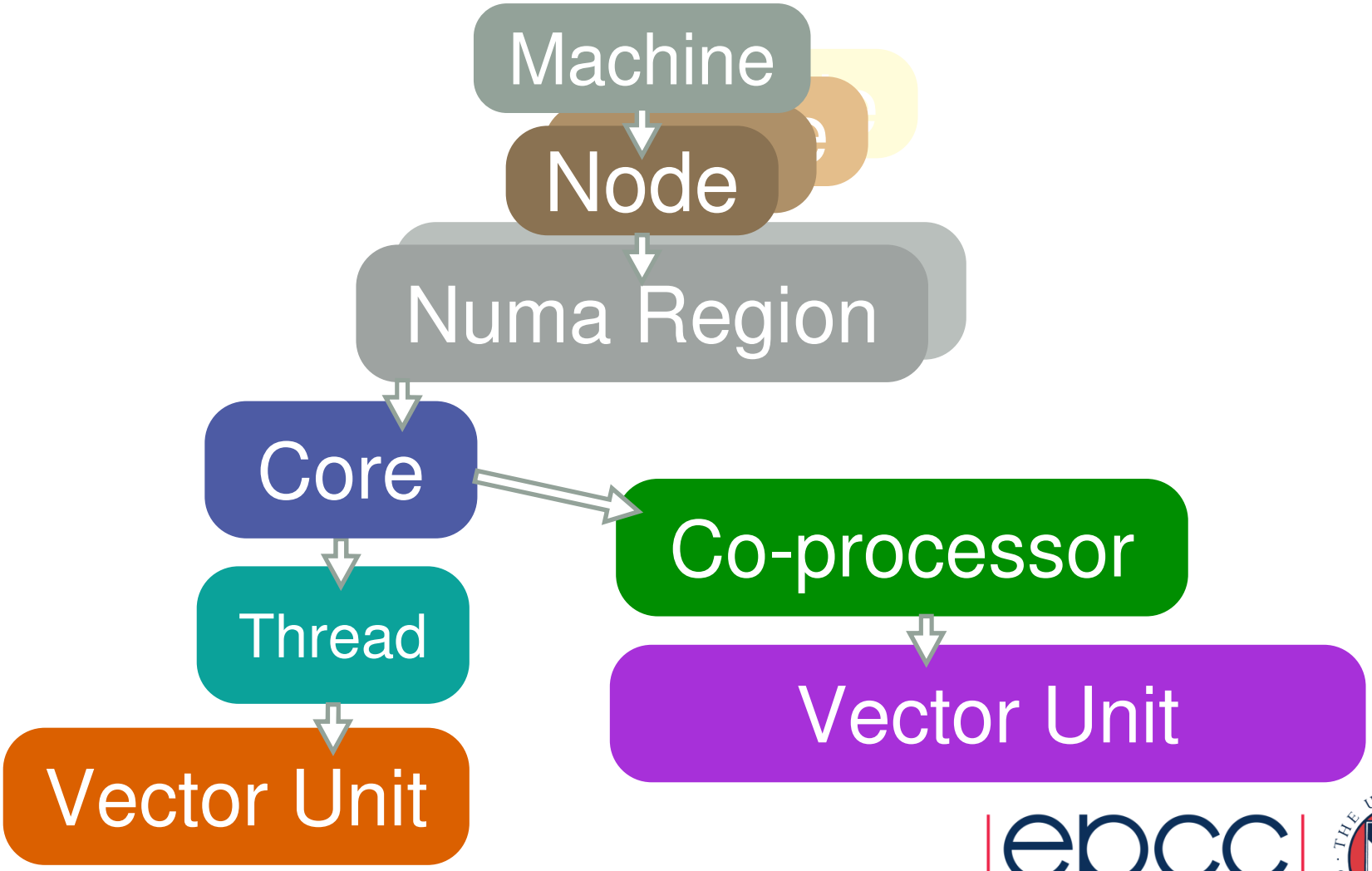
Levels of parallelism



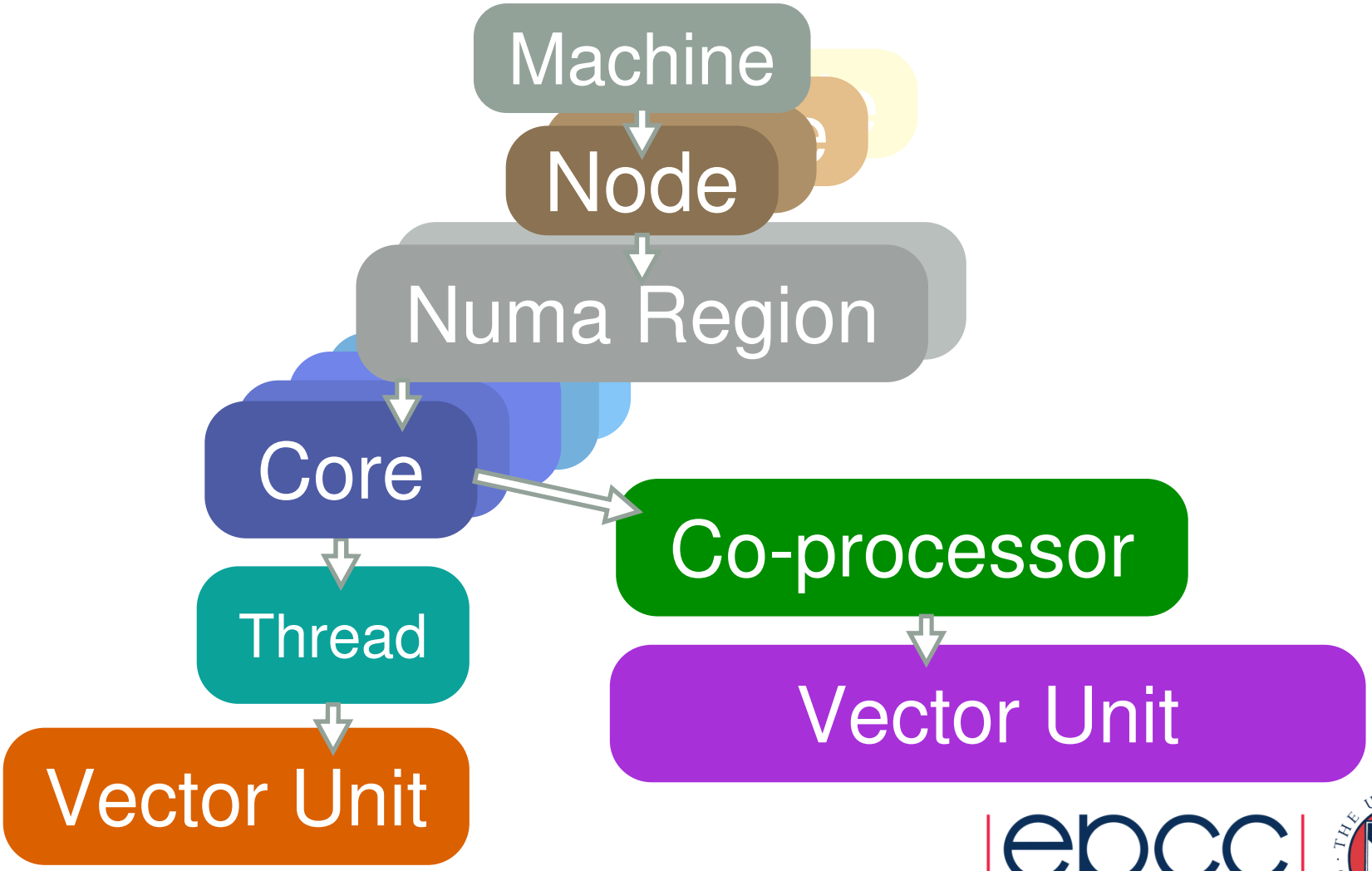
Levels of parallelism



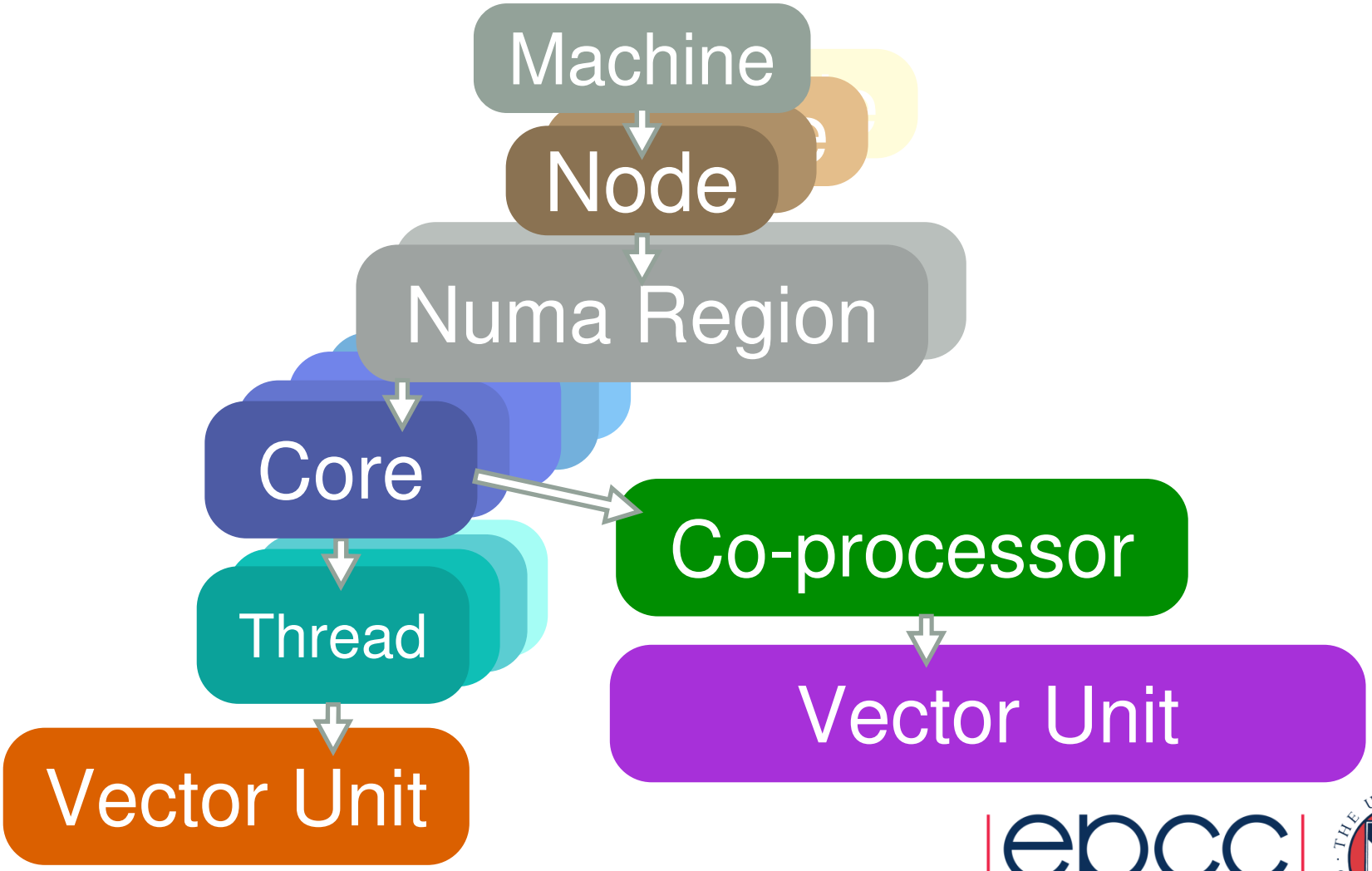
Levels of parallelism



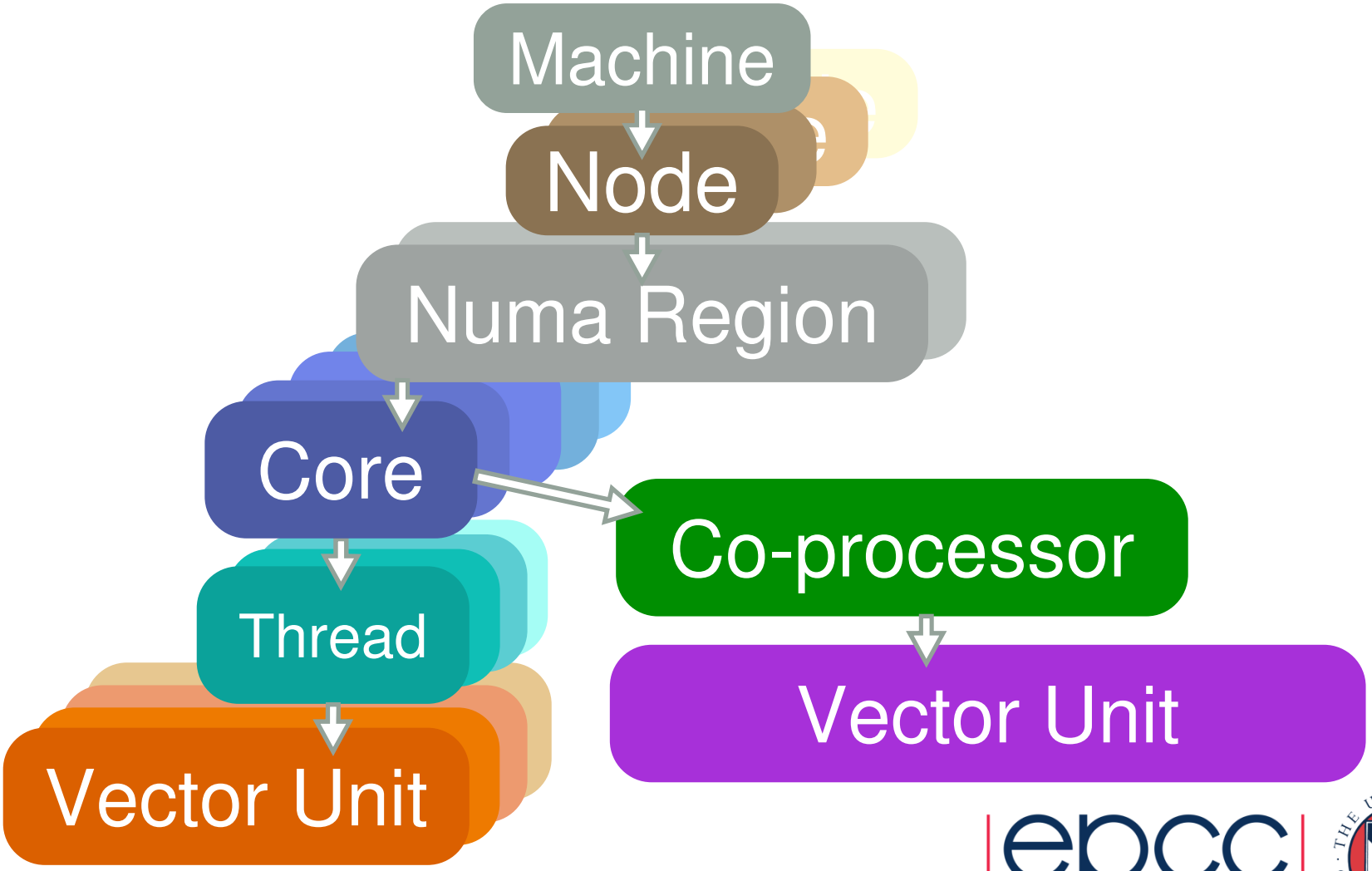
Levels of parallelism



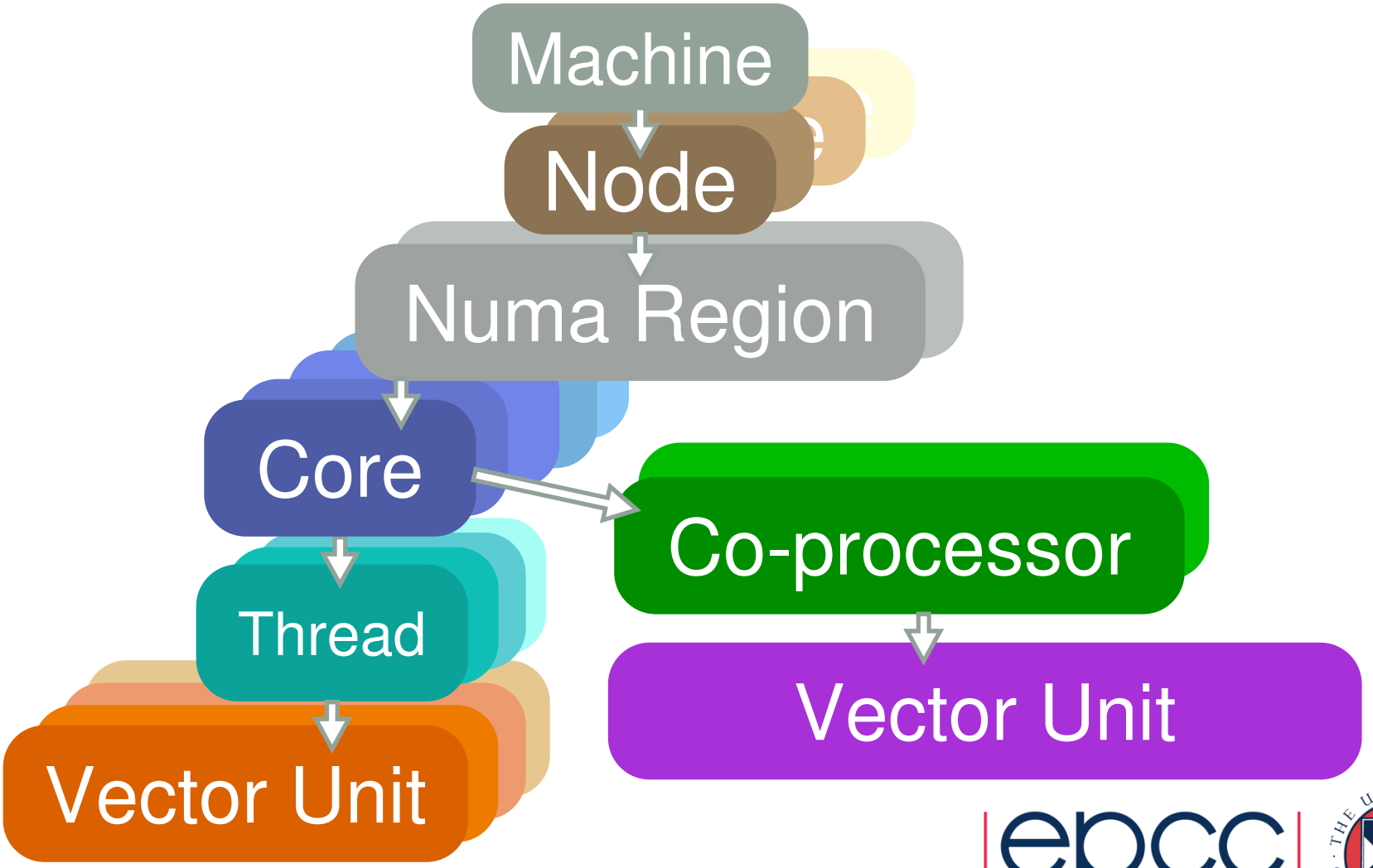
Levels of parallelism



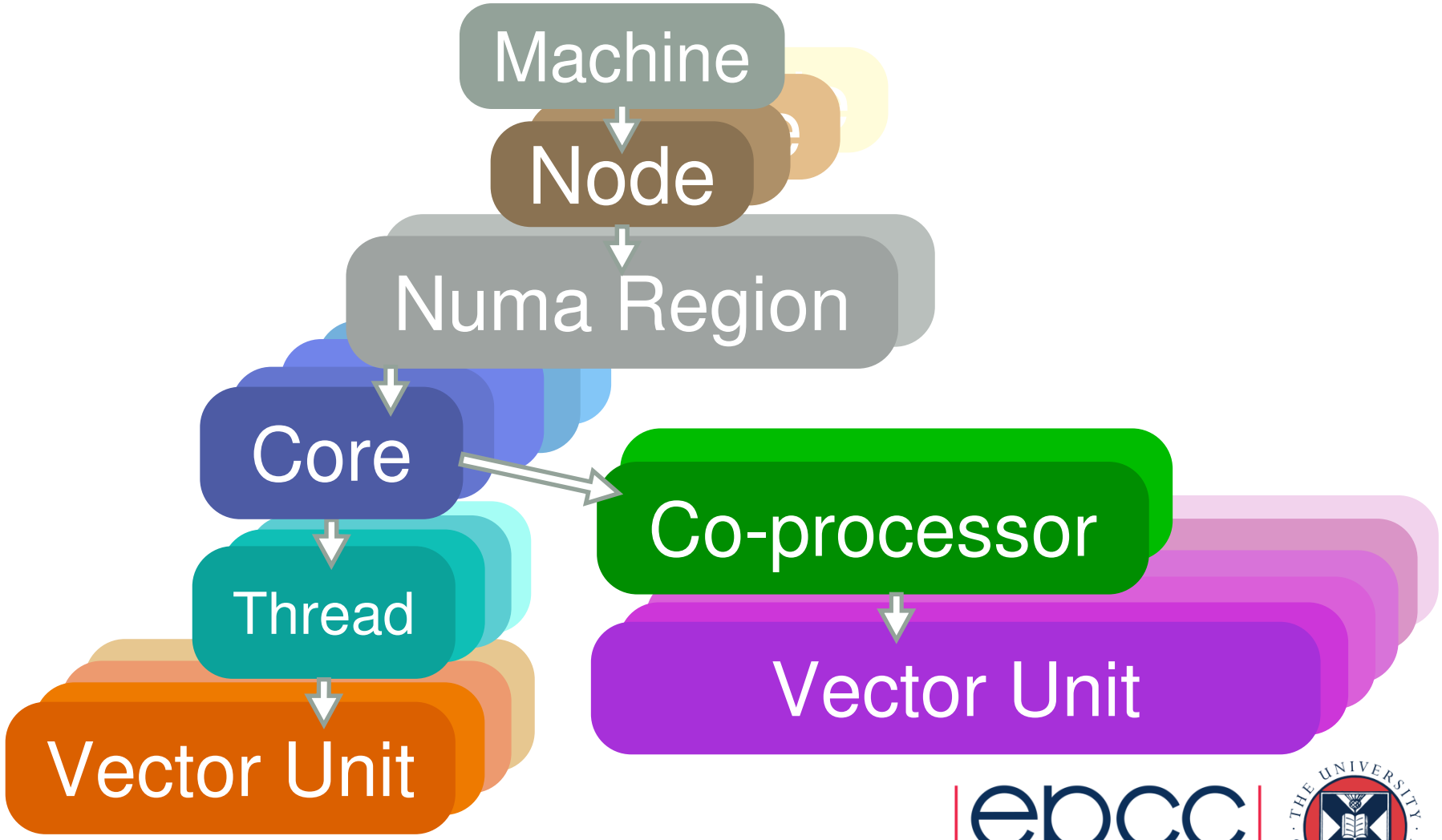
Levels of parallelism



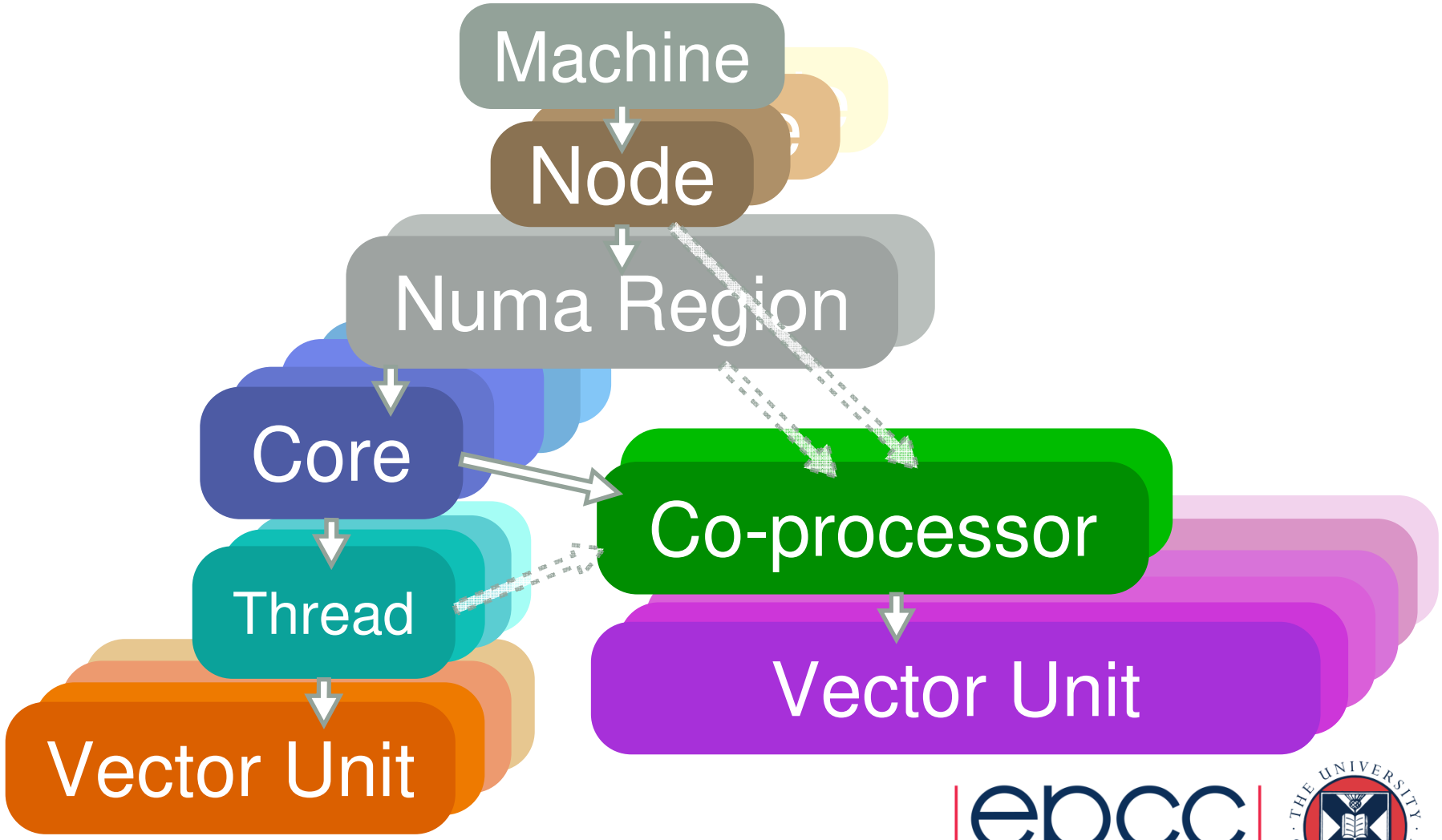
Levels of parallelism



Levels of parallelism



Levels of parallelism

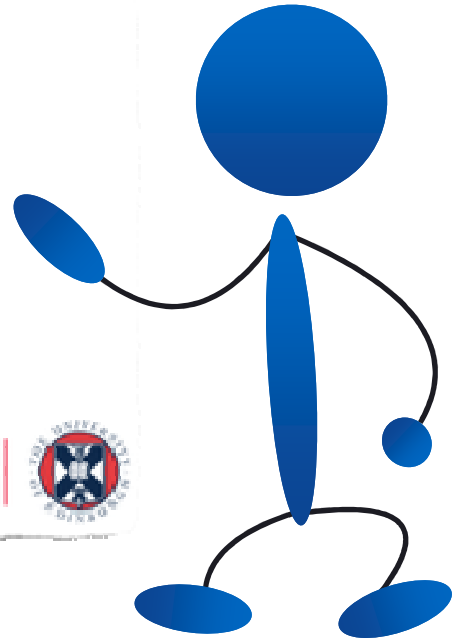


Summary

LESSON PLAN

- Programming models
- Parallelisation
- Compilers and Tools
- Performance Considerations

epcc



epcc

- **Programming models**
  - Native, Offload, Symmetric - what's best for you.
- **Parallelisation**
  - MPI, OpenMP -> OpenMP better on Xeon Phi
  - Many ways to mix and match
- **Compilers and Tools**
  - Use Intel compilers (C, C++, Fortran)
  - Intel and Allinea tools: VTune, Map, etc.
  - Wide variety of runtime tools and environment variables: micinfo, KMP_AFFINITY
- **Performance Considerations**
  - Programming model
  - Vectorisation - needed to exploit Xeon Phi compute
  - Data alignment - needed to make vectorisation useful
  - Thread/process affinity - can be critical for performance
  - Application design: Consider levels of parallelism

Summary

- **Programming models**
  - Native, Offload, Symmetric - what's best for you.
- **Parallelisation**
  - MPI, OpenMP -> OpenMP better on Xeon Phi
  - Many ways to mix and match
- **Compilers and Tools**
  - Use Intel compilers (C, C++, Fortran)
  - Intel and Allinea tools: VTune, Map, etc.
  - Wide variety of runtime tools and environment variables: micinfo, KMP_AFFINITY
- **Performance Considerations**
  - Programming model
  - Vectorisation - needed to exploit Xeon Phi compute
  - Data alignment - needed to make vectorisation useful
  - Thread/process affinity - can be critical for performance
  - Application design: Consider levels of parallelism

Summary

- **Programming models**
  - Native, Offload, Symmetric - what's best for you.
- **Parallelisation**
  - MPI, OpenMP -> OpenMP better on Xeon Phi
  - Many ways to mix and match
- **Compilers and Tools**
  - Use Intel compilers (C, C++, Fortran)
  - Intel and Allinea tools: VTune, Map, etc.
  - Wide variety of runtime tools and environment variables: micinfo, KMP_AFFINITY
- **Performance Considerations**
  - Programming model
  - Vectorisation - needed to exploit Xeon Phi compute
  - Data alignment - needed to make vectorisation useful
  - Thread/process affinity - can be critical for performance
  - Application design: Consider levels of parallelism

Summary

- **Programming models**
  - Native, Offload, Symmetric - what's best for you.
- **Parallelisation**
  - MPI, OpenMP -> OpenMP better on Xeon Phi
  - Many ways to mix and match
- **Compilers and Tools**
  - Use Intel compilers (C, C++, Fortran)
  - Intel and Allinea tools: VTune, Map, etc.
  - Wide variety of runtime tools and environment variables: micinfo, KMP_AFFINITY
- **Performance Considerations**
  - Programming model
  - Vectorisation - needed to exploit Xeon Phi compute
  - Data alignment - needed to make vectorisation useful
  - Thread/process affinity - can be critical for performance
  - Application design: Consider levels of parallelism

Thank You!