csinparallel.org

An Overview Of The PDC Landscape

Suzanne J. Matthews
West Point

Slides Prepared By:
Joel Adams
Calvin College

Overview

Let's explore two related-but-different areas:

- Shifts in the hardware landscape
  - Distributed computing
  - Parallel computing
- Corresponding changes on the software side

Distributed Hardware Landscape

- Cloud computing services:
  - Amazon's Elastic Compute Cloud (EC2)
  - Microsoft's Azure
  - Google's App Engine
  - …
- Devices for the "Internet of Things":
  - Raspberry PI
  - Intel Galileo
  - …

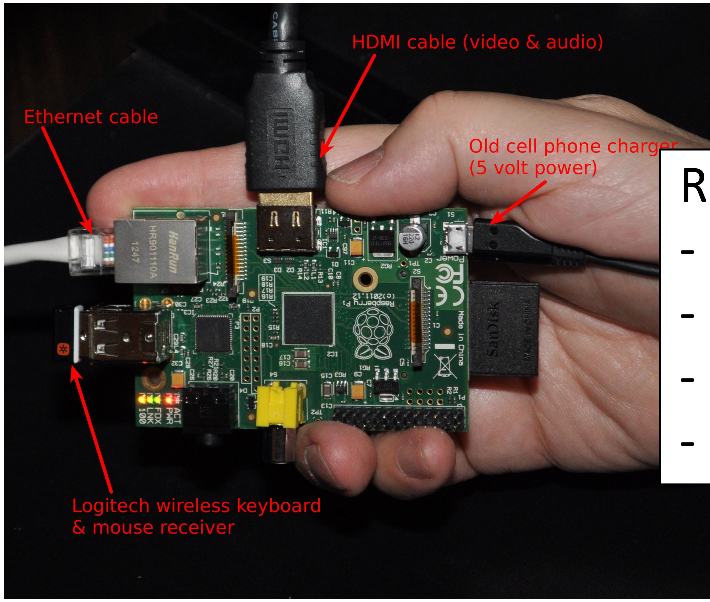
Distributed Computing: Hardware

amazon webservices™

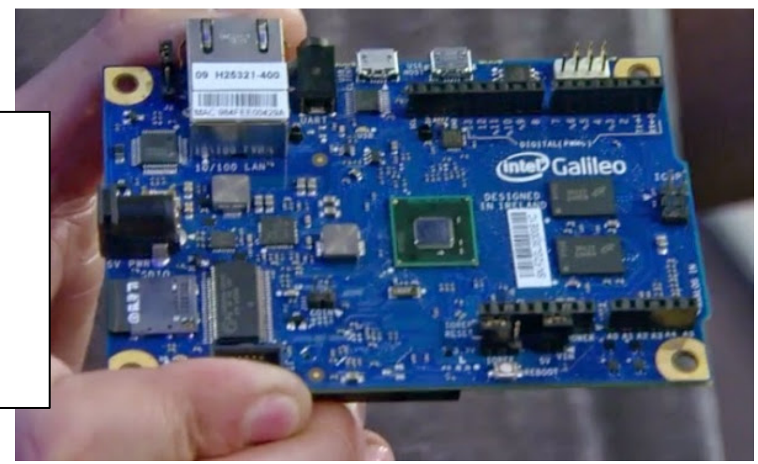
Azure

Google App Engine

Distributed Computing: Hardware (2)

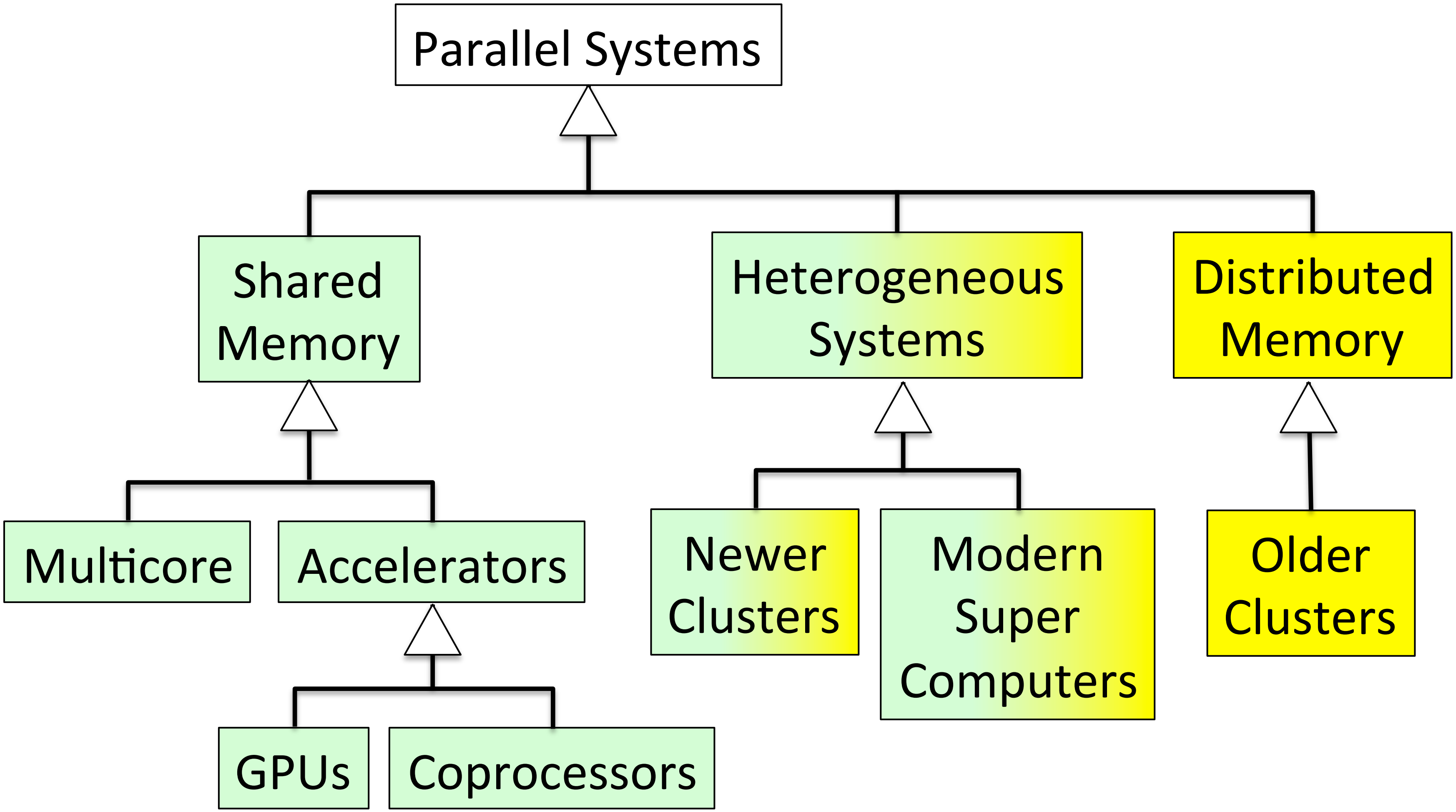


Raspberry PI
- ARM 1176 CPU
- VideoCore GPU
- Linux
- $25



Intel Gallileo
- Quark X1000 CPU
- Arduino IDE
- ~$50

Parallel Hardware Landscape



History / Timeline

Decade	Parallel Hardware Platforms	Memory
1980s	Vector supercomputers	Shared
	Multiprocessors (networked)	Distributed
1990s	Cluster supercomputers	Distributed
	Internet	Distributed
	Symmetric multiprocessors	Shared
2000s	GPUs	Shared
	Multicore processors	Shared
2010s	Hybrid supercomputers/clusters	Both
	Coprocessors (w. vector units)	Shared

History / Timeline (2)

Decade	Parallel Hardware Platforms	Software
1980s	Vector supercomputers	Proprietary
	Multiprocessors (networked)	Proprietary
1990s	Cluster supercomputers	MPI
	Internet	Sockets,BOINC
	Symmetric multiprocessors	Various
2000s	GPUs	CUDA,OpenCL
	Multicore processors	Various
2010s	Hybrid systems	Combinations
	Coprocessors (w. vector units)	Various

Today's Software Landscape

- The software generally varies with the hardware platform it is intended to run on:
  - Distributed memory systems
  - Shared memory systems
    - Vanilla shared-memory systems
    - Shared-memory systems with Accelerators
      - Manycore GPUs and/or coprocessors
  - Heterogeneous systems
- No "one size fits all" software solution (yet)
- Let's explore these one at a time…

Distributed Memory Systems

Two broad categories; both use standalone *compute nodes*, each with their own memory:

- Local-area distributed-memory systems
  - Nodes are connected via a *local area network* (the faster the better)
- Wide-area distributed-memory systems
  - Nodes are connected via a *wide area network* (such as the Internet – comparatively slow)

Distributed Memory System Software

Local-area dist-mem. systems use *multiprocessing*:

- Remote *processes* are launched on compute nodes
- The message passing interface (MPI) is the industry standard platform for such systems
  - Implementations for C, C++, Fortran, Python, …
  - Generality: Works well on *shared-memory systems* too
- MapReduce is a Google platform for *reliably* solving some kinds of distributed problems
  - Hadoop is an open-source version of MapReduce
  - WebMapReduce is a browser-based Hadoop front end developed by Dick Brown & his St. Olaf students

MACALESTER COLLEGE

CALVIN
MINDS IN THE MAKING

UNITED STATES MILITARY ACADEMY
WEST POINT.

ST · OLAF COLLEGE

Distributed Mem. System Software (2)

For *wide-area* distributed-memory systems:

- Remote processes communicate via *sockets*:
  - Client-server systems are most common
  - Peer to peer systems are a decentralized alternative…
- The Berkeley Open Infrastructure for Network Computing (BOINC) is a widely used platform for coordinating distributed computing tasks:
  - SETI@Home, Folding@Home, LHC@Home, …

The relative slowness of wide-area communication limits this approach to embarrassingly parallel problems.

Shared Memory Systems

Three broad categories:

- Vanilla shared-memory systems
  - Multicore / Multisocket CPU-based systems
  - Cores share a common memory
- Accelerated shared-memory systems
  - Vanilla systems plus many-core accelerator(s) (GPGPU, Coprocessor)
- Heterogenous systems: CPU + Accelerator

Most shared-memory systems use *multithreading*

Vanilla Shared Mem. Software

Vanilla shared-memory systems are ubiquitous:

- Open MultiProcessing (OpenMP) is an industry standard for multithreading
  - Non-proprietary open standard
  - Multilanguage support (C, C++, Fortran)
  - Pragma-based programming; relatively easy
- Language-based multithreading options:
  - C (pthreads), C++11 (Boost), C# (.NET), Java, …
- Vendor-specific (proprietary) libraries/languages
  - Intel's Thread Building Blocks (TBB), Google's Go*, …

iPad 3: quad-core A5X chip

iPhone 6: dual-core A8 chip

Vanilla Shared Mem. Software (2)

Vanilla shared-memory systems can also be programmed via *message-passing*:

- MPI also works well on these systems
- Some languages utilize *message-passing tasks* to avoid multithreading's *race conditions*:
  - Erlang, Scala, …
  - Programs written in these languages port easily to distributed-memory systems

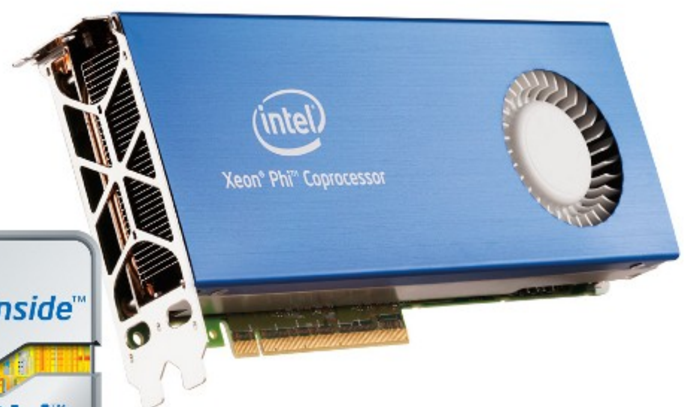
Every CS undergraduate student should learn how to program vanilla shared-memory parallel systems

Accelerated Shared Mem. Systems

Software for shared-memory systems with accelerators varies with the accelerator:

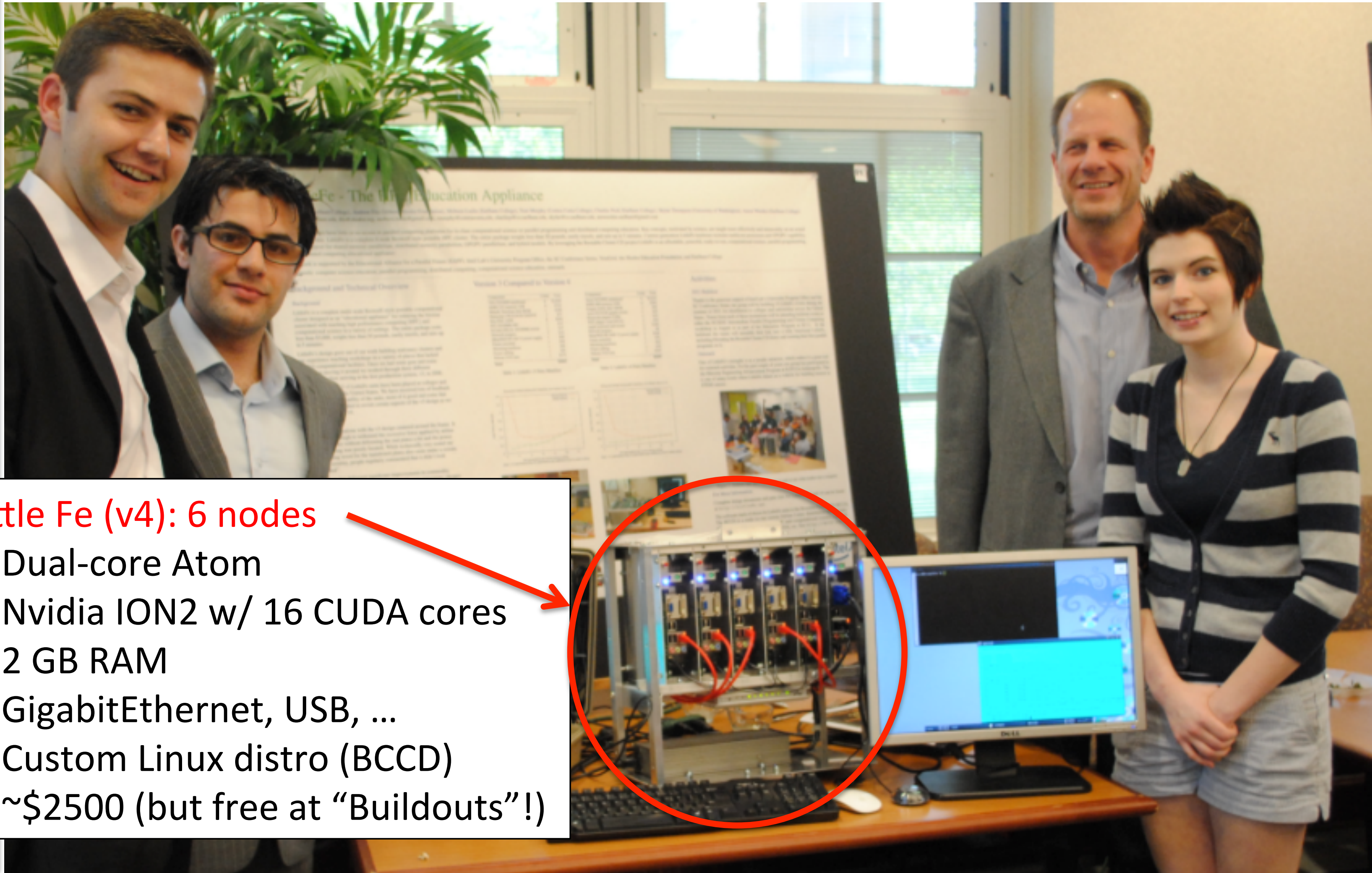
- General Purpose Graphics Processing Unit (GPGPU) systems
  - Nvidia
  - AMD's ATI/Radeon
- Coprocessor systems
  - Intel's *Xeon Phi* (61 cores, 4 hw threads/core), available to us on Intel's Manycore Testing Lab (MTL)
  - Parallella's *Epiphany* (16 cores)

Accelerators



intel inside
Xeon Phi

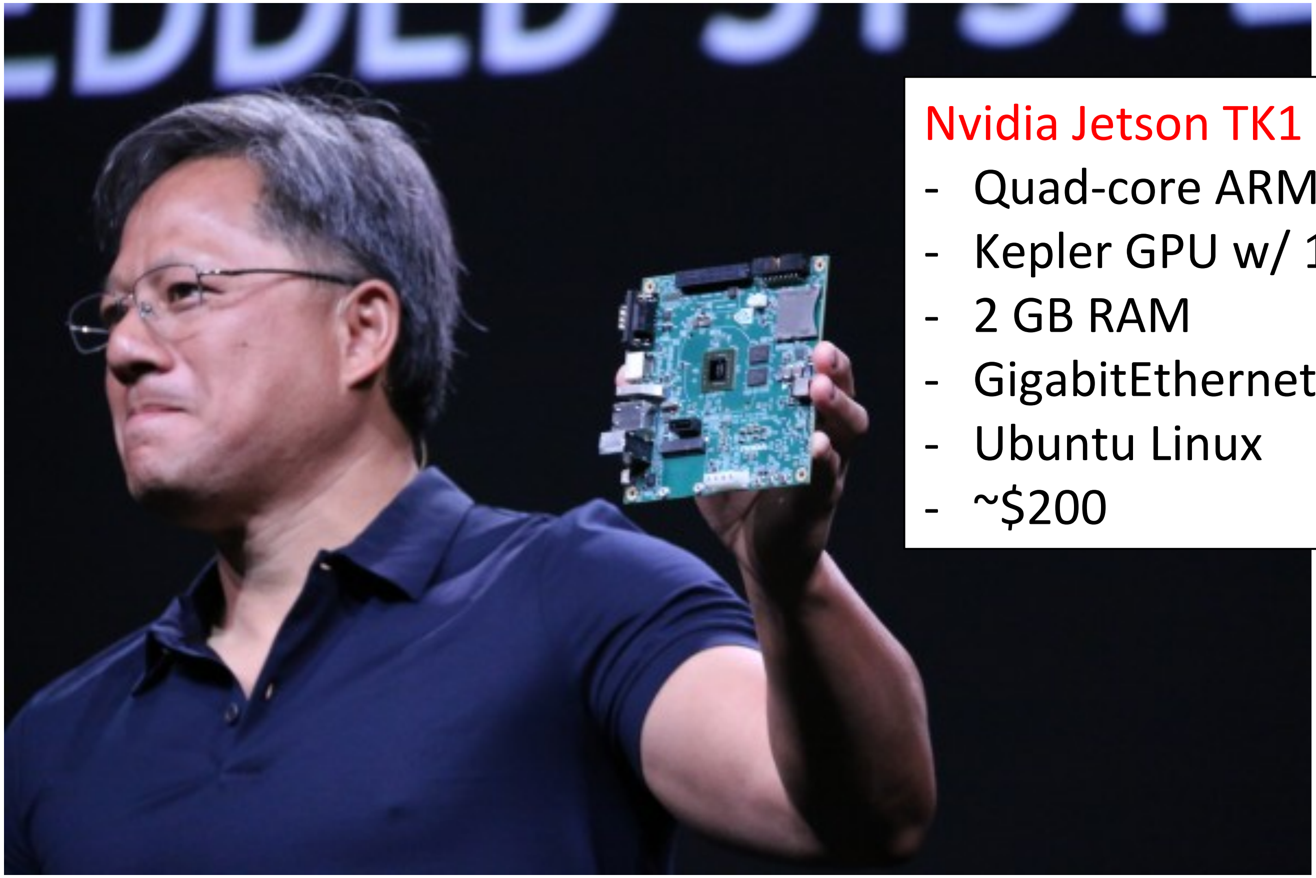
Accelerator System: Little Fe



Little Fe (v4): 6 nodes

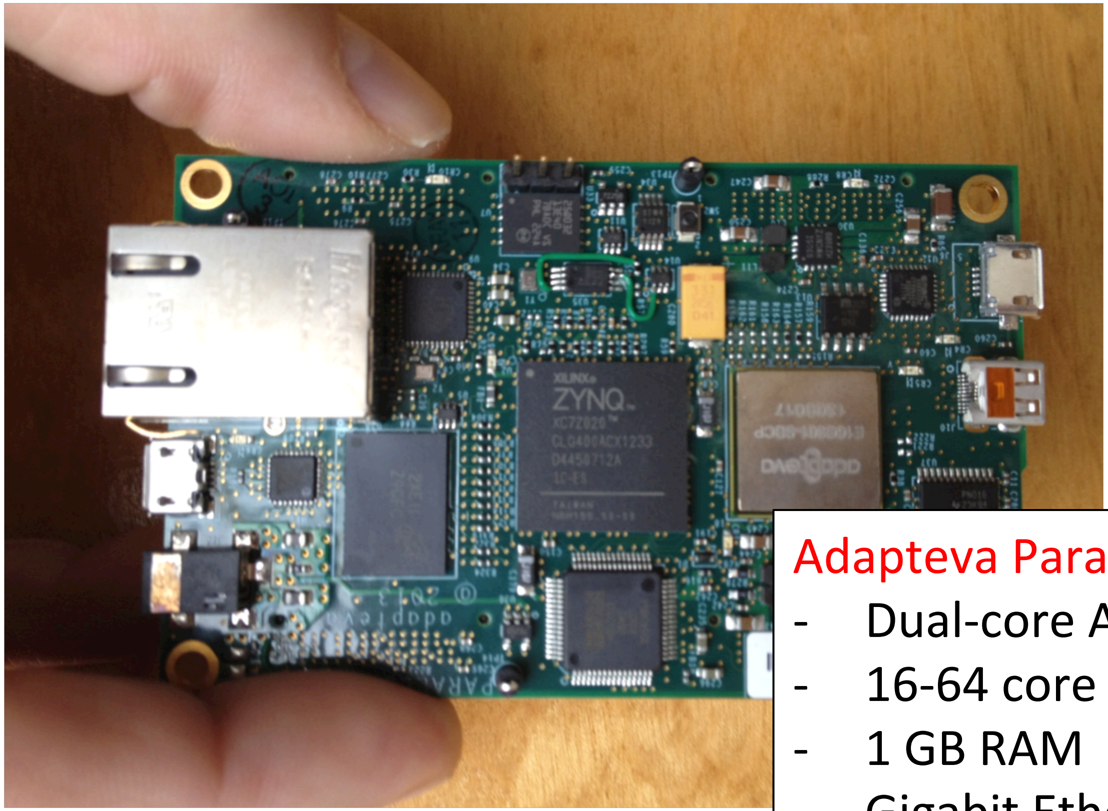
- Dual-core Atom
- Nvidia ION2 w/ 16 CUDA cores
- 2 GB RAM
- GigabitEthernet, USB, …
- Custom Linux distro (BCCD)
- ~$2500 (but free at “Buildouts”!)

Accelerator Systems: Small



Nvidia Jetson TK1
- Quad-core ARM A15
- Kepler GPU w/ 192 CUDA cores
- 2 GB RAM
- GigabitEthernet, HDMI, USB, …
- Ubuntu Linux
- ~$200

Accelerator Systems: Even Smaller



Adapteva Parallella

- Dual-core ARM A7
- 16-64 core Epiphany Coprocessor
- 1 GB RAM
- Gigabit Ethernet, USB, HDMI, …
- Ubuntu Linux
- ~$99 (but free via university program!)

Accelerated Shared Mem. Software

Software for shared-memory systems with *GPUs*:

- Nvidia's Compute Unified Device Architecture (CUDA):
  - \+ Well established; extensive examples/documentation available
  - – Proprietary; works only on Nvidia GPU cores
- Open Compute Language (OpenCL):
  - \+ Platform independent open standard
  - \+ Can use every core in a system (Nvidia or not)
  - – Significantly more complicated than CUDA
  - – Fewer examples/tutorials/documentation available
- OpenACC (Open Acceleration?):
  - \+ Pragmas (*a la* OpenMP) to simplify GPU computing
  - – Promising, but still in development
- Intel's Array Building Blocks (ArBB):
  - \+ C++ library for vectorized parallel computing
  - – Proprietary; C++ only

Accelerated Shared Mem. Software (2)

Software for shared-memory systems with *coprocessors* (cluster on a chip):

- MPI
- OpenMP
- OpenCL
- Intel's ArBB

Coprocessors are fairly new, so other software platforms for them will likely appear…

Heterogeneous Systems

Complications:

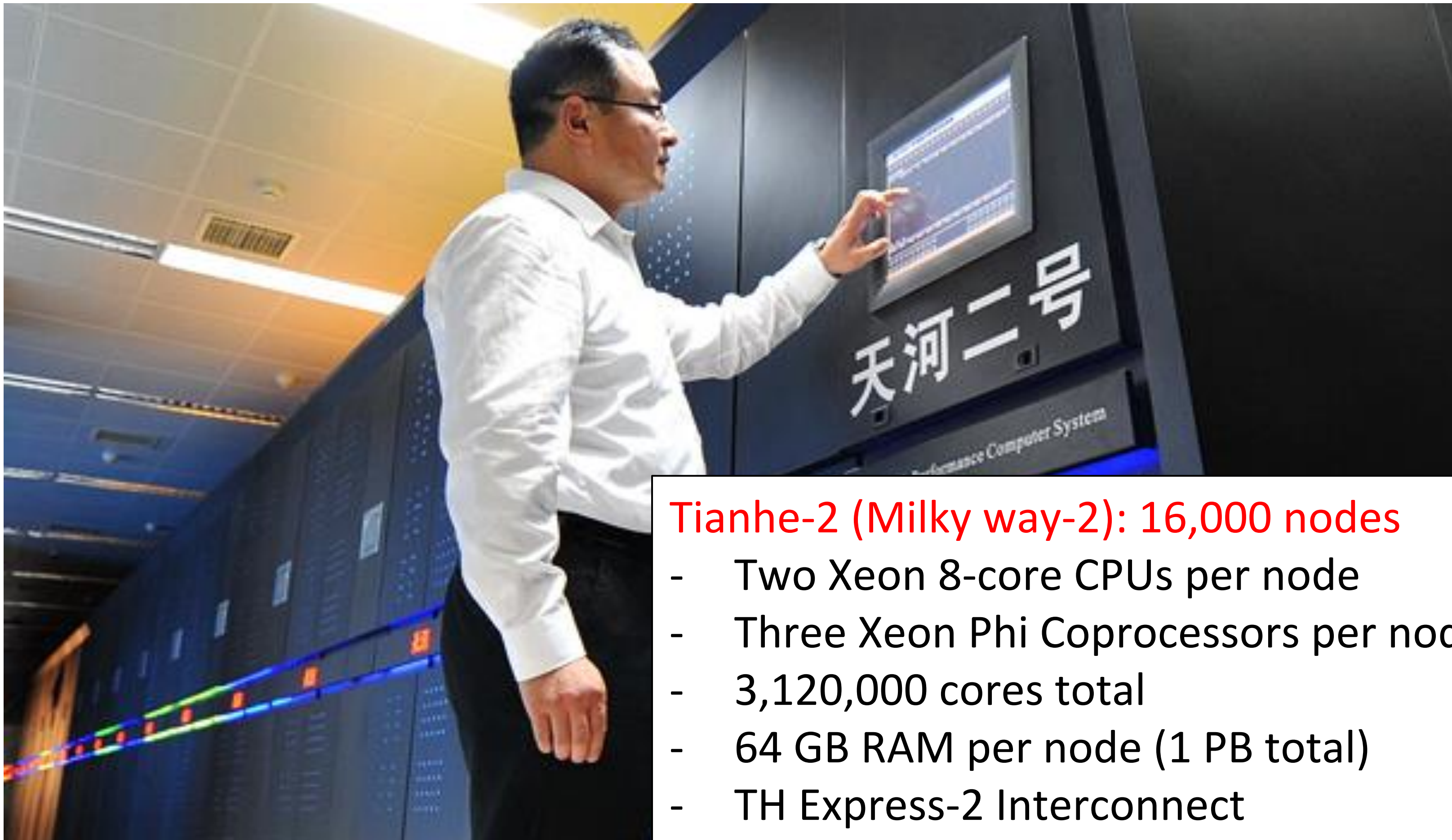
1. Unicore CPUs are nearly extinct…
   – All recent clusters have multicore CPUs

2. Accelerators can be added to cluster nodes

This creates lots of options for heterogeneity:

– Distributed + shared memory
– Distributed + shared memory + GPUs
– Distributed + shared memory + coprocessors

Tianhe-2



Tianhe-2 (Milky way-2): 16,000 nodes

- Two Xeon 8-core CPUs per node
- Three Xeon Phi Coprocessors per node
- 3,120,000 cores total
- 64 GB RAM per node (1 PB total)
- TH Express-2 Interconnect
- Kylin Linux

Heterog. System Software Options

- Distributed + shared memory
  - MPI (1 MPI process/core)
  - MPI + OpenMP (1 MPI process/node)
  - MapReduce (1 or more MR process/node)
- Distributed + shared memory + GPUs
  - MPI + CUDA
  - MPI + OpenMP + CUDA
  - MPI + OpenCL
- Distributed + shared memory + coprocessors
  - MPI
  - MPI + OpenMP
  - MPI + OpenCL

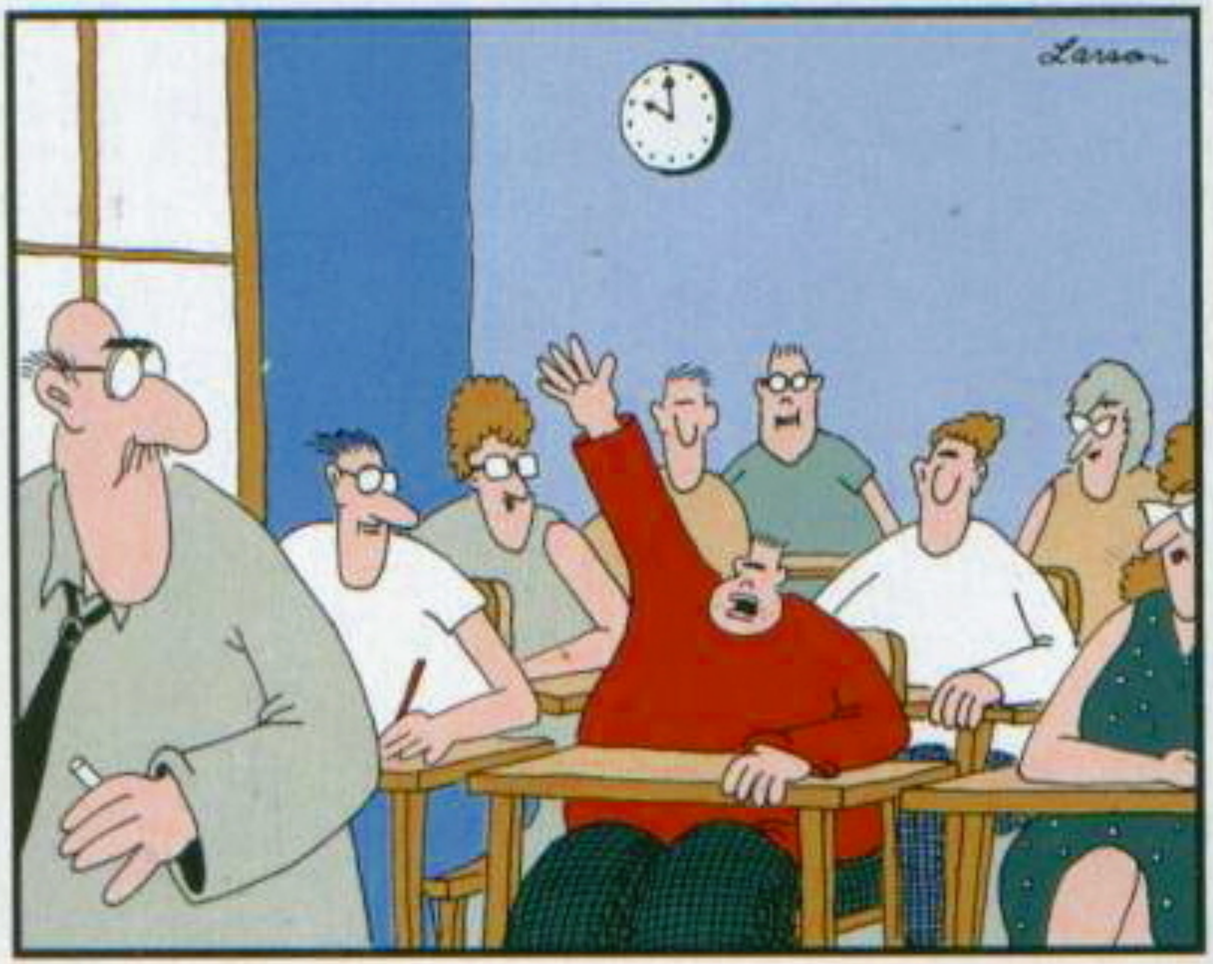
Problems

- MPI, OpenMP, etc. have carried us this far, but the experts say they are *insufficient* to let us reach *exascale* computing
- MPI is relatively low-level
- Newer *high level languages* are being developed to make it easier to develop scalable programs (at least for distributed+shared mem. hybrids):
  - Chapel: APGAS language from Cray
  - Scala: immutable OO Actors, message passing, JVM
  - …

APGAS Languages

… asynchronous partitioned global address space

- All tasks share a global address space / memory
- All tasks can access the entire space, but the address space is logically partitioned, so a task may have *affinity* for a particular partition:
  - Thread-local memory on a shared mem. system
  - Process memory on a distributed mem. system
  - …
- Merges strengths of shared+distributed systems
  - Chapel, Unified Parallel C (UPC), X10, Fortress, …



"Mr. Osborne, may I be excused? My brain is full."

MACALESTER COLLEGE

CALVIN
MINDS IN THE MAKING

UNITED STATES MILITARY ACADEMY
WEST POINT.

ST · OLAF COLLEGE

Information Overload

- If you are saying to yourself:
  - *"This is overwhelming!"*
  - *"PDC is changing so fast; is there any content that is worth my time to learn / not ephemeral?"*

  Don't feel bad; you're not alone!

- One of our goals is to establish a supportive community for PDC educators!

Parallel Patterns

- … are industry-standard techniques and best-practices that have proven useful in many different parallel contexts.
- … are built into popular platforms like MPI and OpenMP.
- … are likely to be useful for the long-term, regardless of future PDC developments.
- … provide a way to organize PDC concepts.

Parallel Pattern Categories

These patterns can be categorized:

- Algorithmic Strategies: general approaches to devising parallel algorithms.
- Implementation Strategies: patterns used to implement a given algorithmic strategy.
- Communication & Synchronization: patterns for synchronizing/communicating between the tasks in a given strategy.

Parallel Algorithm Strategies

Most parallel programs use one of just three *parallel algorithm strategy patterns*:

- Data decomposition: divide up the data and process it in parallel.
- Task decomposition: divide the algorithm into functional tasks that are performed in parallel (to the extent possible).
- Pipeline: divide the algorithm into linear stages, through which we "pump" the data.

Data Decomposition: 1 Thread

Thread 0

Data Decomposition: 2 Threads

Thread 0

Thread 1

Data Decomposition: 4 Threads

Thread 0

Thread 1

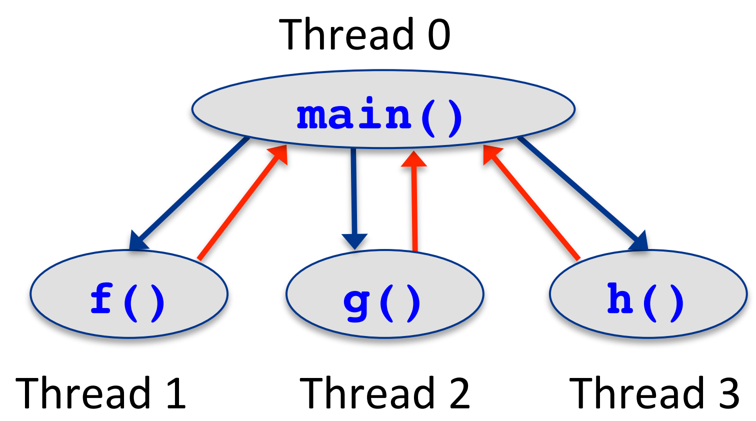
Thread 2

Thread 3

Algor. Strategy: Task Decomposition

– The independent functions in a sequential computation can be "parallelized":

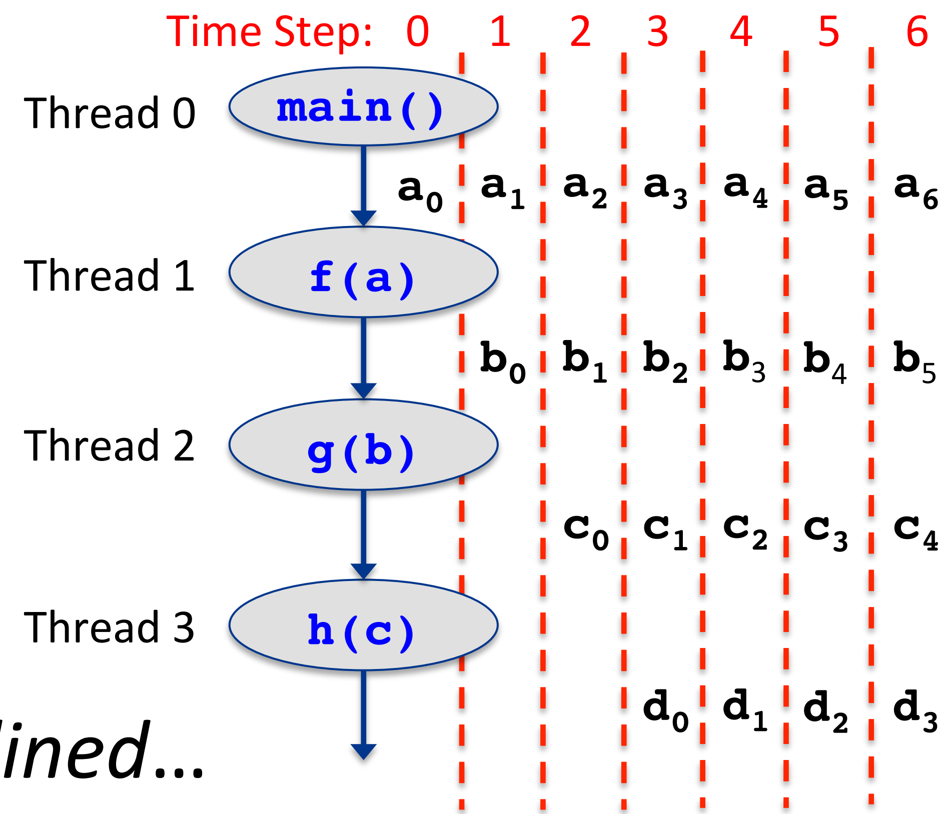
```
int main() {
    x = f();
    y = g();
    z = h();
    w = x + y + z;
}
```



Algorithmic Strategy: Pipeline

- When functions are not independent:

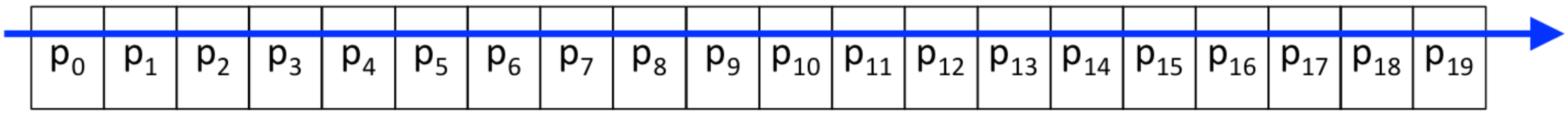
```
int main() {
    ...
    while (fin) {
        fin >> a;
        b = f(a);
        c = g(b);
        d = h(c);
        fout << d;
    }
    ...
}
```



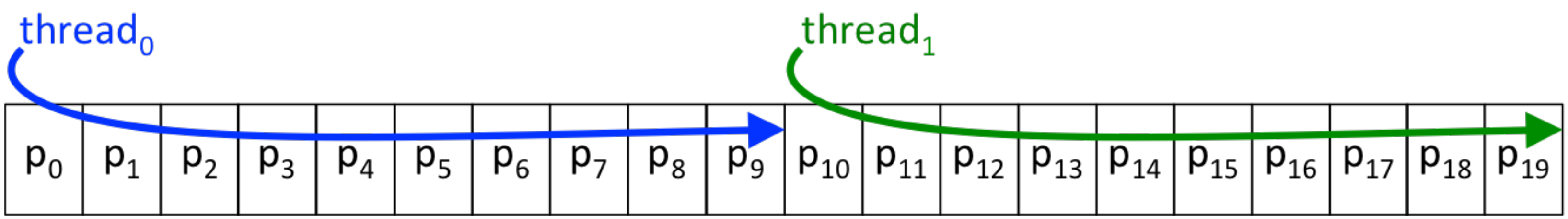
they can still be *pipelined*…

Implement. Strategy: Parallel Loop

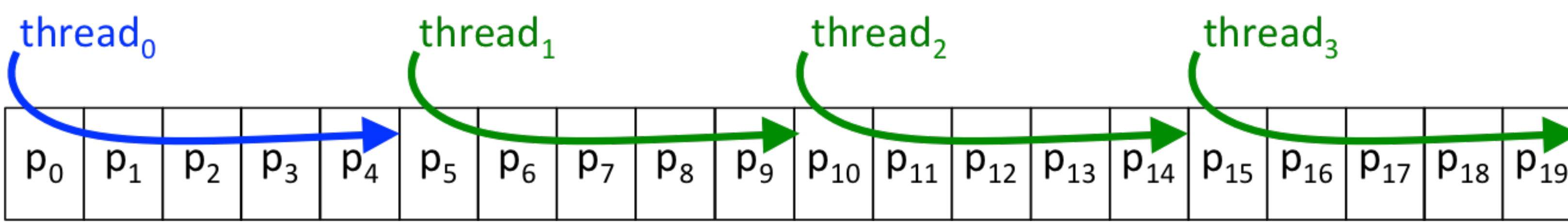
One Thread:



Two Threads:



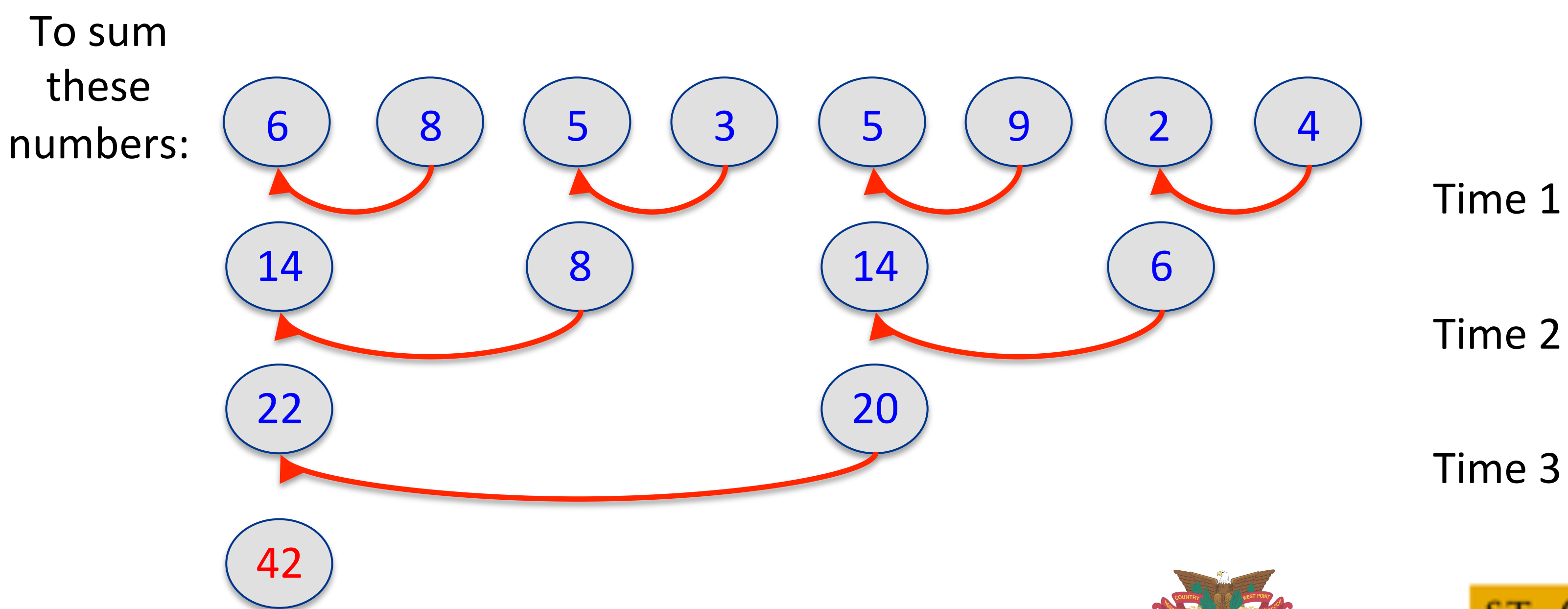
Four Threads:



Communication Pattern: Reduction

Parallel programs often need to combine the local results of N parallel tasks.

- When N is in the millions, *O(N)* time is too slow
- The reduction pattern does it in *O(lg(N))* time:



Conclusions

- There are many possible starting points in PDC
  - There are *many* hardware and software options
  - Getting started is more important than where
  - Choose a software platform(s) that works best at your institution/department:
    - C/C++: MPI+OpenMP
    - Java: Scala
    - Language agnostic: Chapel
    - …
- Patterns offer a much-needed source of stability.
- CSinParallel is here to help!

Thank You!

Macalester College · Calvin – Minds in the Making · United States Military Academy West Point · St. Olaf College