CSCI-GA.3033-012

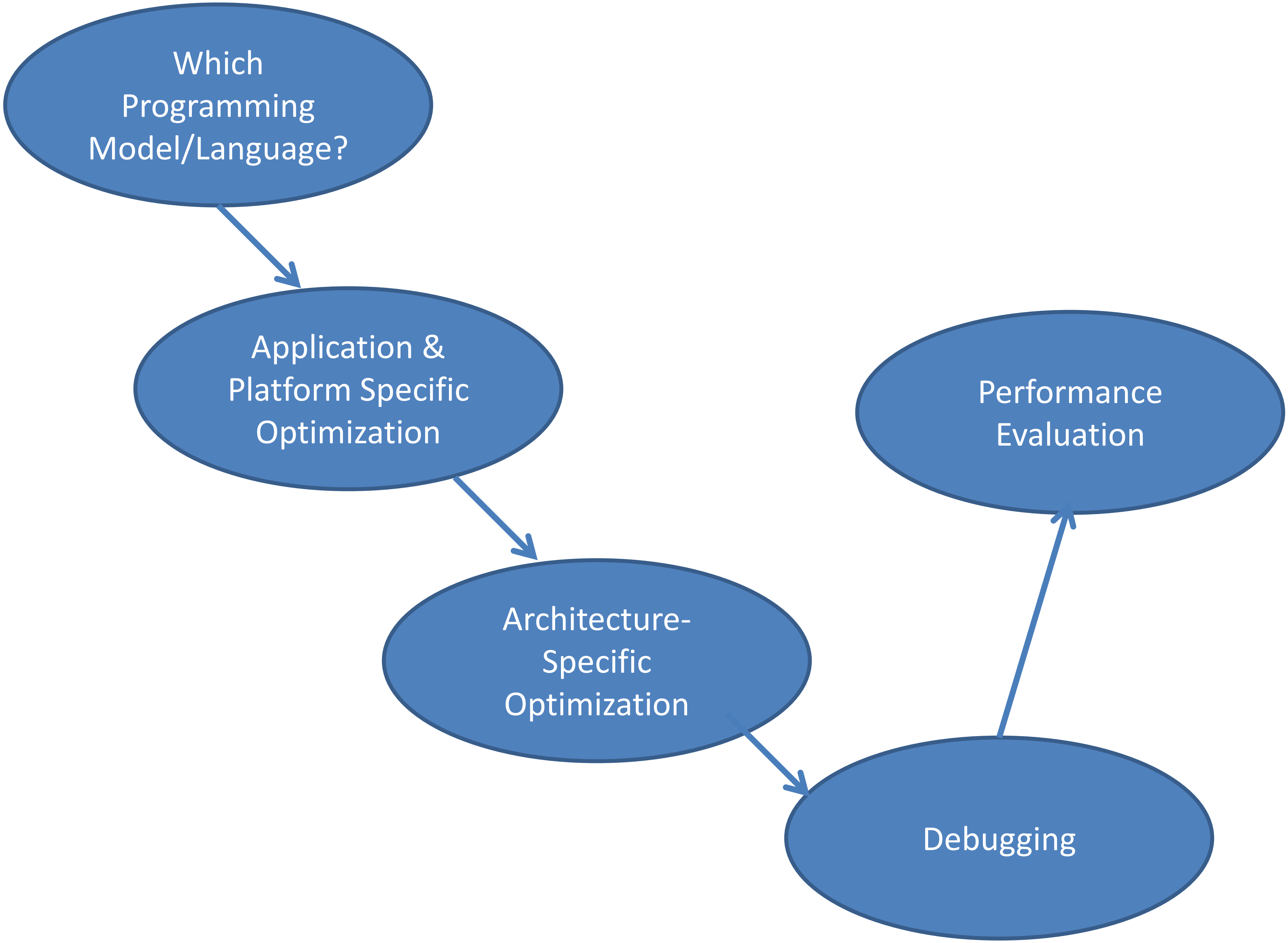
Multicore Processors: Architecture & Programming

Lecture 12: Putting It All Together

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com

Today's Lecture

- What to do with all what you heard about in this course?
- Challenges in hardware and how they affect the software
- Different type of parallelism
- Common problems in parallel programs
- A glimpse into the future
  - Software
  - Hardware



Which Programming Model/Language?

Application & Platform Specific Optimization

Architecture-Specific Optimization

Debugging

Performance Evaluation

TBB

OpenMP

Cilk/Cilk++

PThreads

Transactions

….

How to choose your programming language/model given an application?

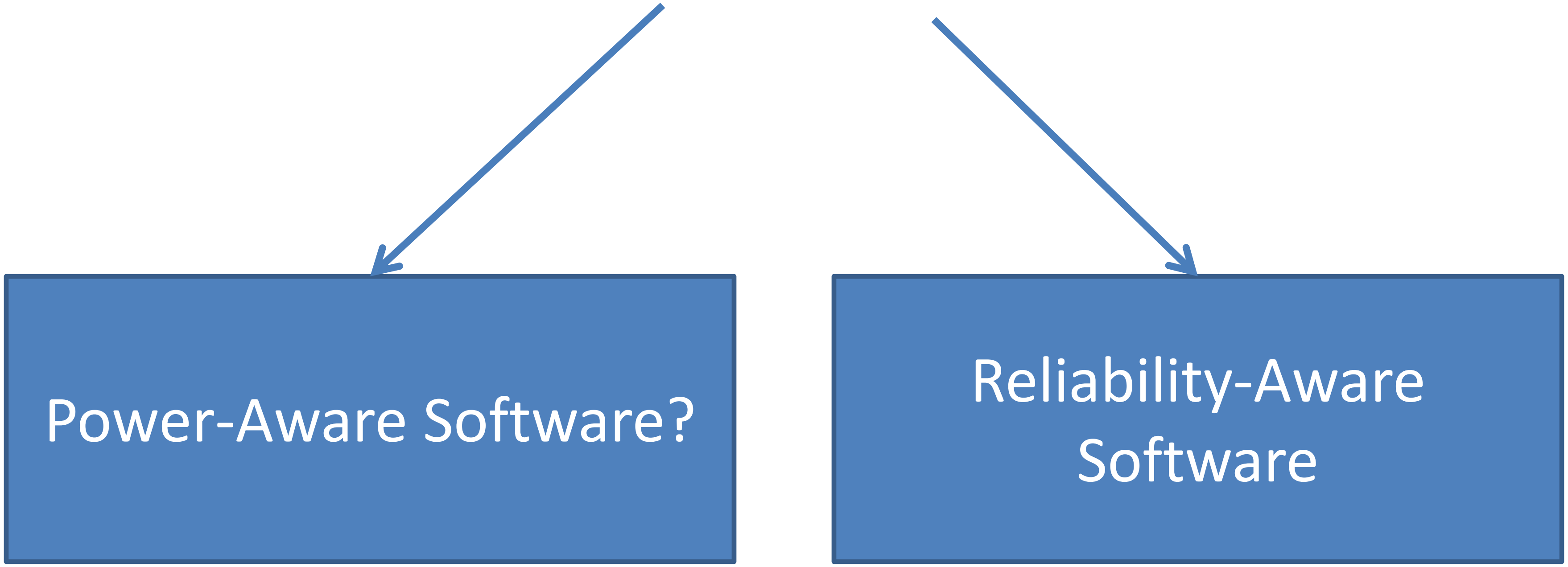
How to Choose?

- Your level of expertise
  - Less experience = higher abstraction
- The concurrency in the application
  - What type of parallelism we have?
- What can you do with the hardware?
- How does the platform interact with you?
- Are you starting from scratch? or from a sequential version of the program?

Today's Lecture

- What to do with all what you heard about in this course?
- Challenges in hardware and how they affect the software
- Different type of parallelism
- Common problems in parallel programs
- A glimpse into the future
  - Software
  - Hardware

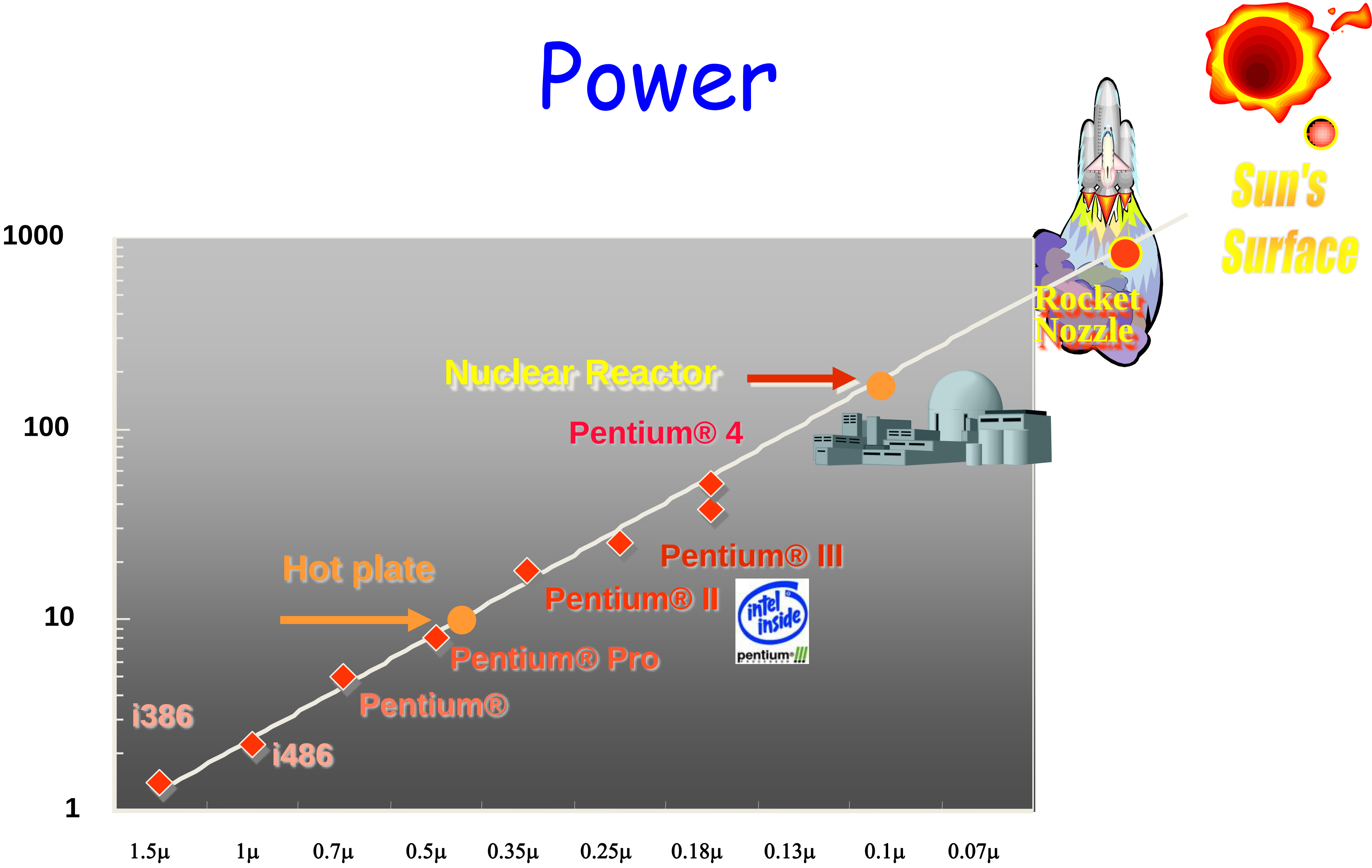
Challenges



Power-Aware Software?

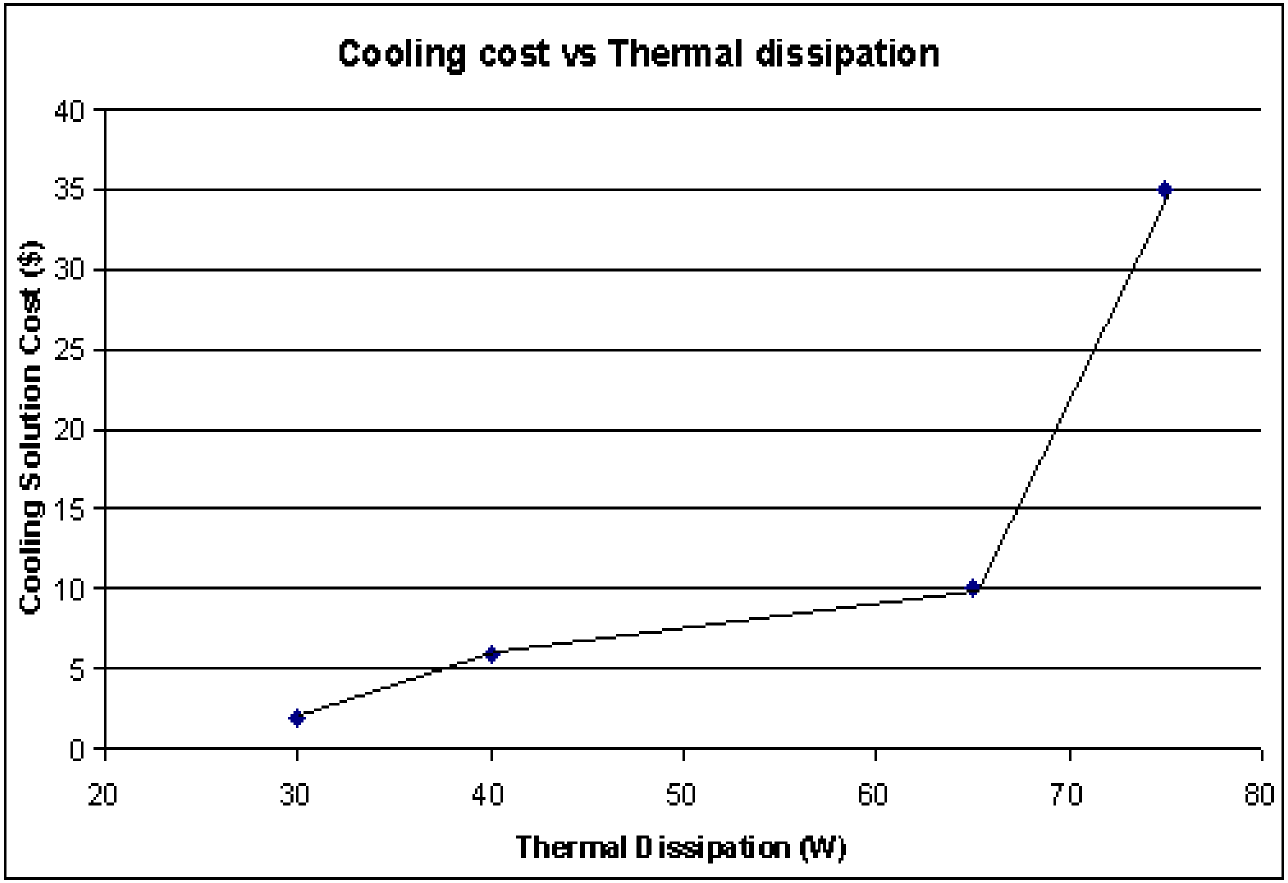
Reliability-Aware Software

Power



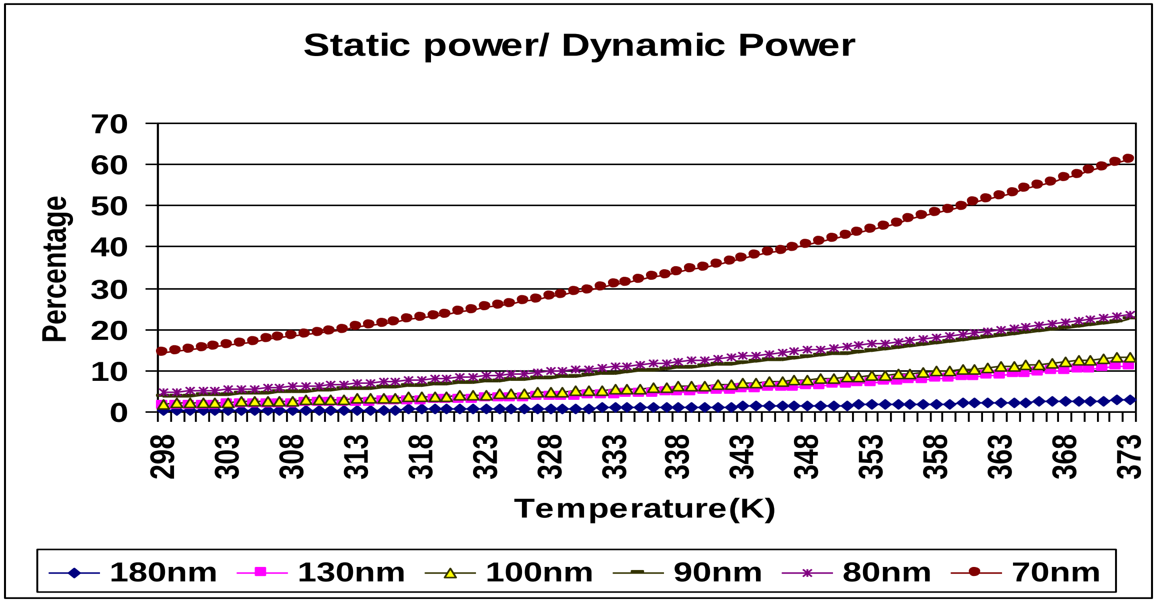
* "New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies" – Fred Pollack, Intel Corp. Micro32 conference key note - 1999.

Cooking Aware Computing


Cooling cost vs Thermal dissipation
Cooling Solution Cost ($)
0
5
10
15
20
25
30
35
40
Thermal Dissipation (W)
20
30
40
50
60
70
80

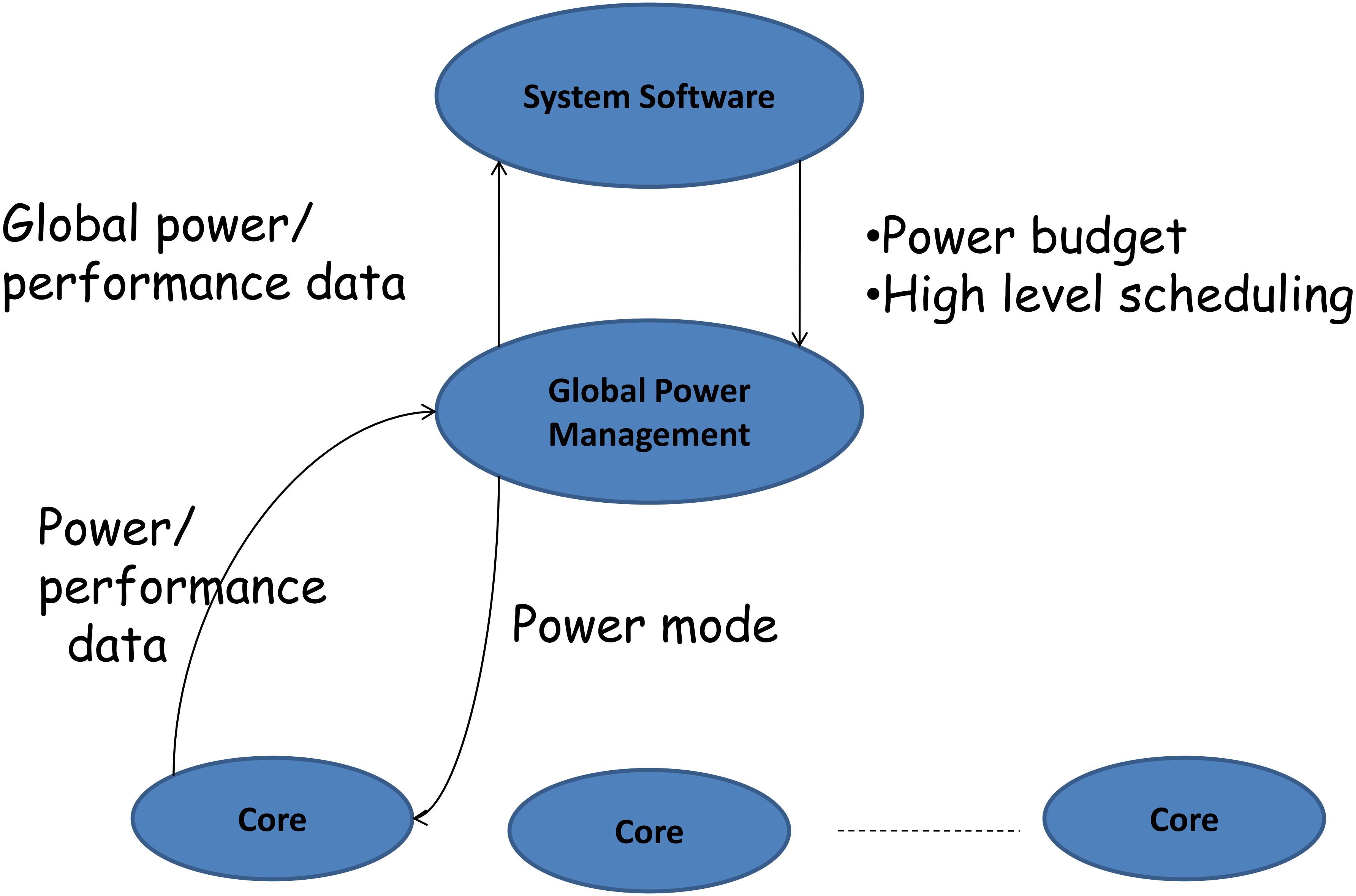
Power-Aware Computing

- Definition: reducing power without loosing performance
- Must deal with:
  - dynamic power
  - static power
  - temperature



What To Do About Dynamic Power

- Stop and go
- DFVS (Dynamic Frequency and Voltage Scaling)
  - At OS level
    - idle time represents energy waste
    - deadlines for interactive programs
  - Offline compiler analysis
    - insert mode-set instructions
    - depends on program phases
    - lowers the voltage for memory-bound sections
  - Online dynamic compiler analysis
    - phase detection
    - binary instrumentation
- Reducing switching activity



System Software

Global power/
performance data

- Power budget
- High level scheduling

Global Power
Management

Power/
performance
data

Power mode

Core

Core

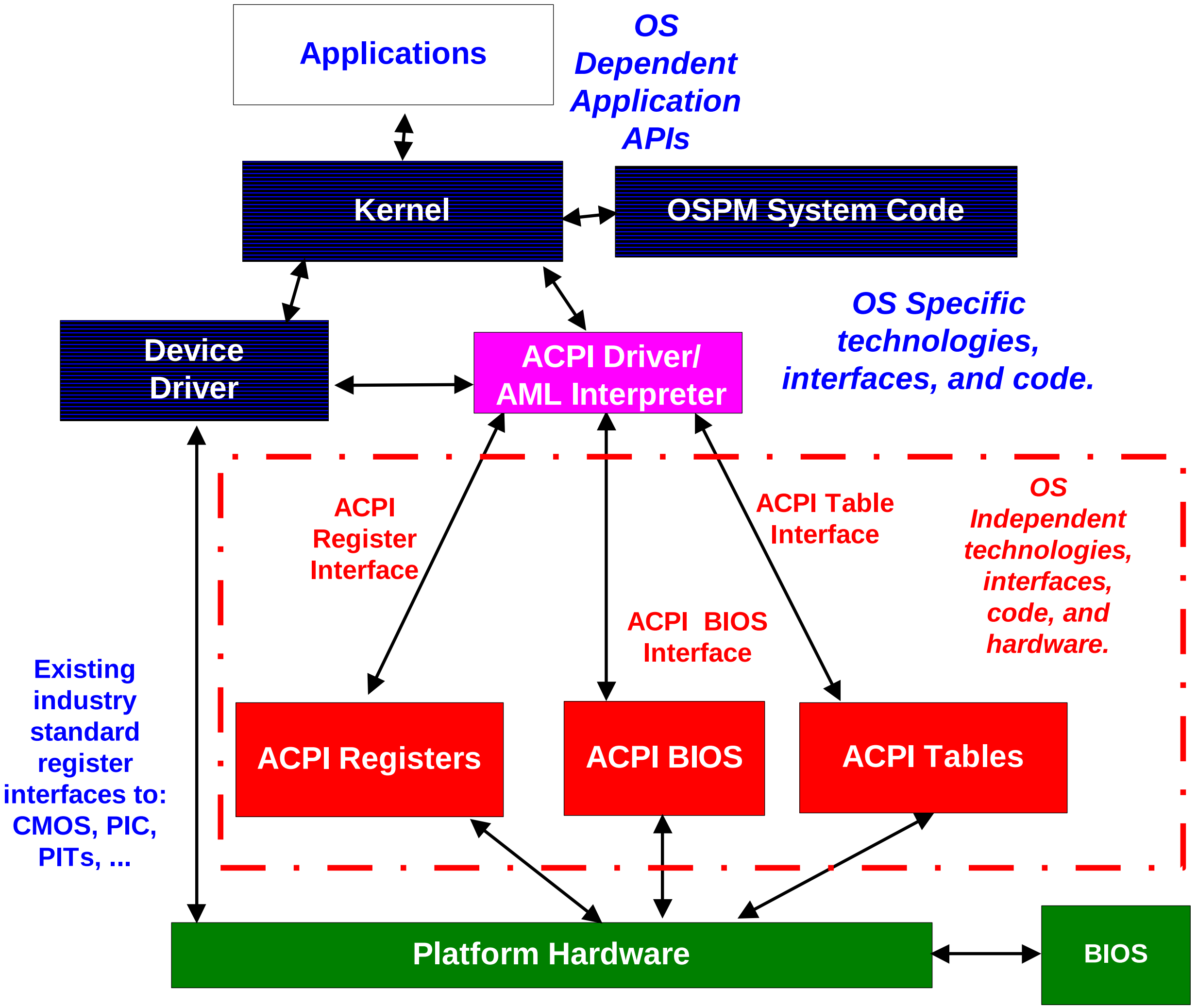
Core

Other Techniques for the Multicore

- Migration
  - Moving threads among cores
  - Timescale on order of a millisecond, much slower than DVFS
  - Migration can be used with DVFS
- Migrate critical thread
  - Measure criticality with heat sensor
  - Or with cache misses as a proxy

As A Programmer

- Try to loose less-expensive operations (i.e. help the compiler)
- Locality to help caches
- Control DVFS
  - Advanced Configuration and Power Interface (ACPI)



Applications

OS Dependent Application APIs

Kernel

OSPM System Code

Device Driver

ACPI Driver/ AML Interpreter

OS Specific technologies, interfaces, and code.

ACPI Register Interface

ACPI Table Interface

OS Independent technologies, interfaces, code, and hardware.

ACPI BIOS Interface

Existing industry standard register interfaces to: CMOS, PIC, PITs, ...

ACPI Registers

ACPI BIOS

ACPI Tables

Platform Hardware

BIOS

🟥 - ACPI Spec Covers this area.

🟦 - OS specific technology

🟩 - Hardware/Platform

🟪 - Provided by ACPI CA

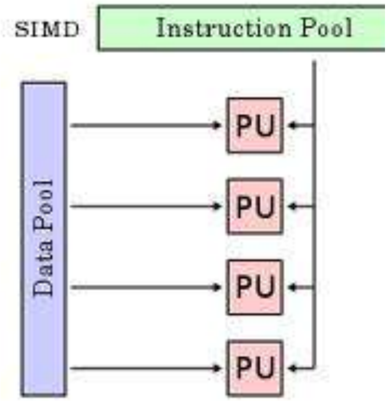
Reliability

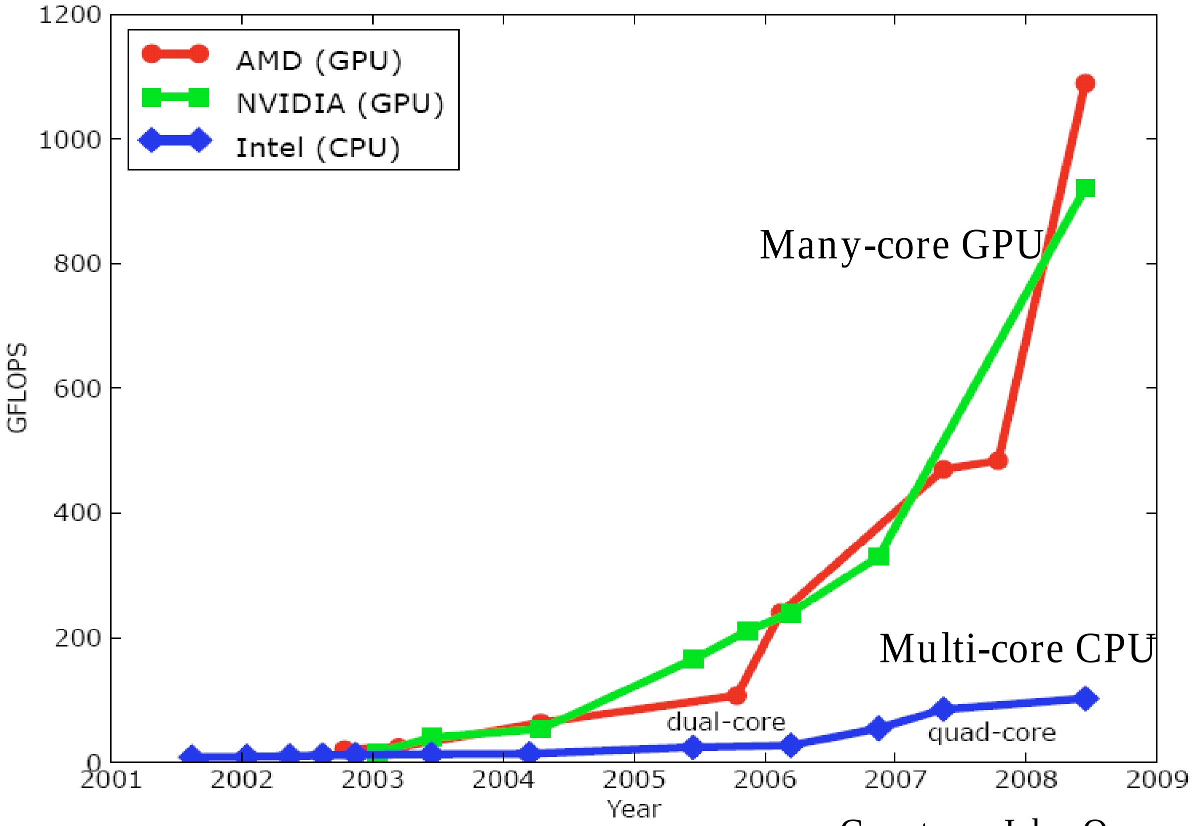
- Transistors are becoming unreliable
- What will your application do if a core becomes unavailable?
- Can you duplicate some computations for very critical operations?

Today's Lecture

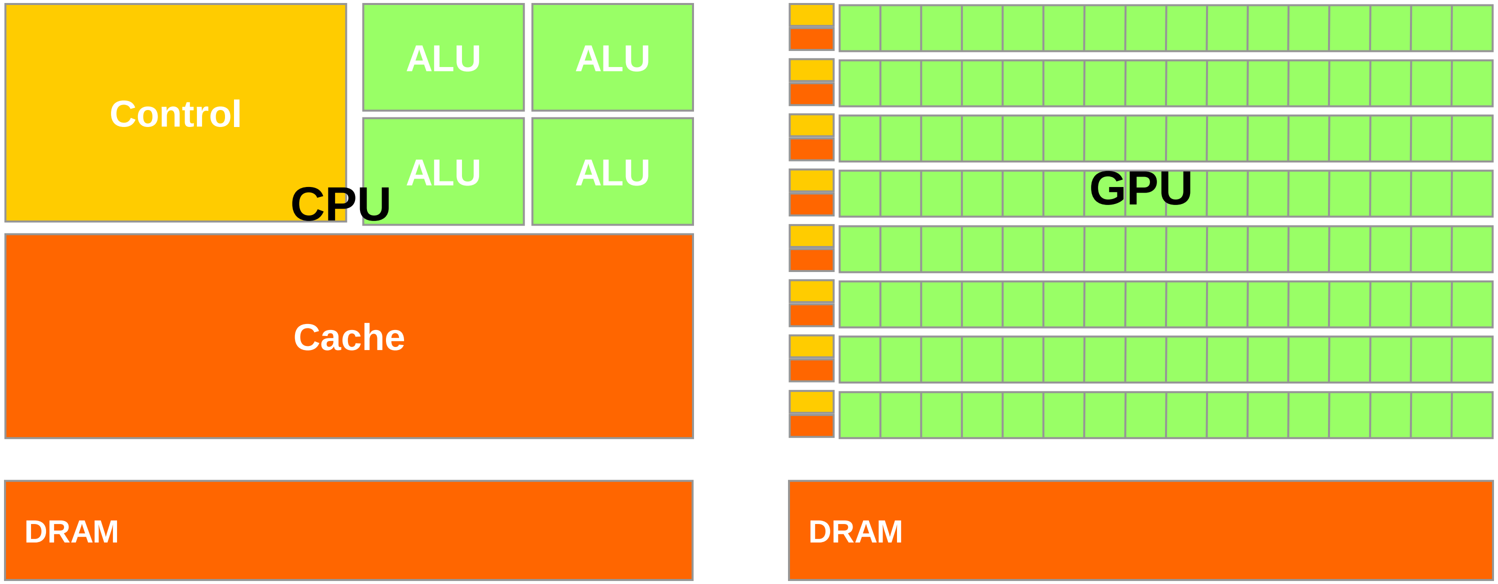
- What to do with all what you heard about in this course?
- Challenges in hardware and how they affect the software
- Different type of parallelism
- Common problems in parallel programs
- A glimpse into the future
  - Software
  - Hardware

A Glimpse at Another Type of Parallelism: SIMD/SPMD/STMD



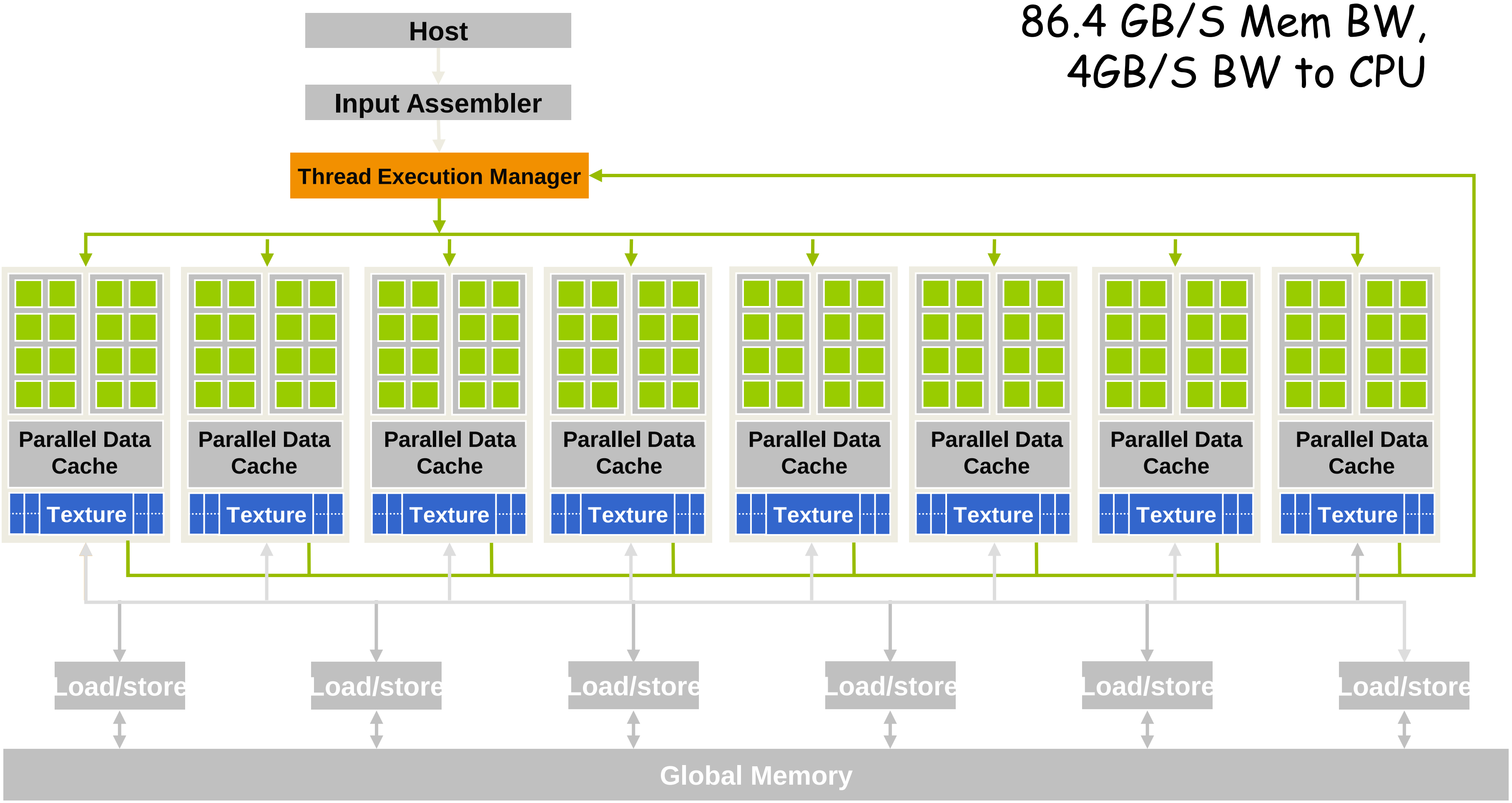


Courtesy: John Owens


Control
ALU
ALU
ALU
ALU
CPU
Cache
DRAM
GPU
DRAM

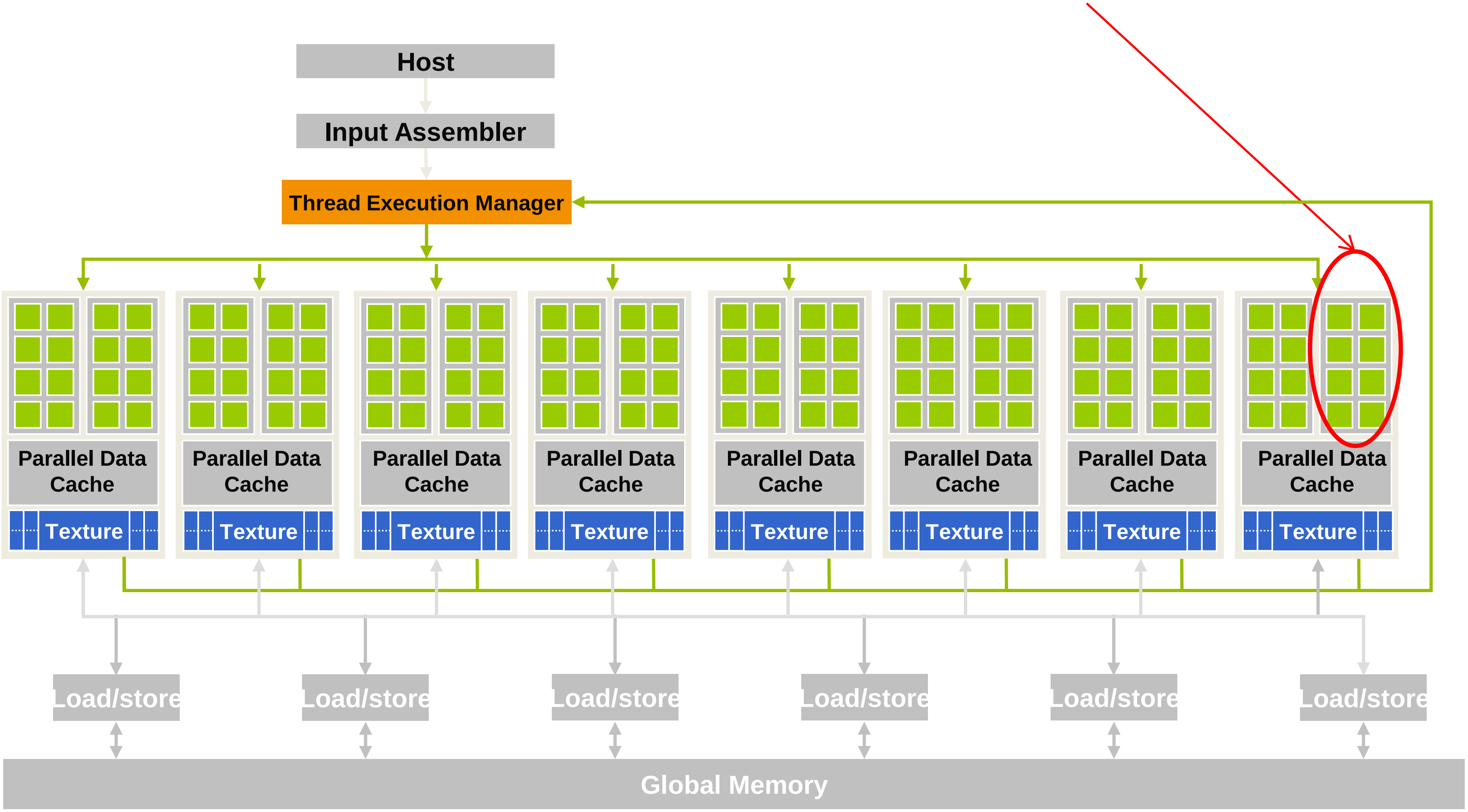
A Glimpse at A Typical GPU: GeForce 8800 (2007)

16 highly threaded SM's, >128 FPU's,
367 GFLOPS, 768 MB DRAM,
86.4 GB/S Mem BW,
4GB/S BW to CPU



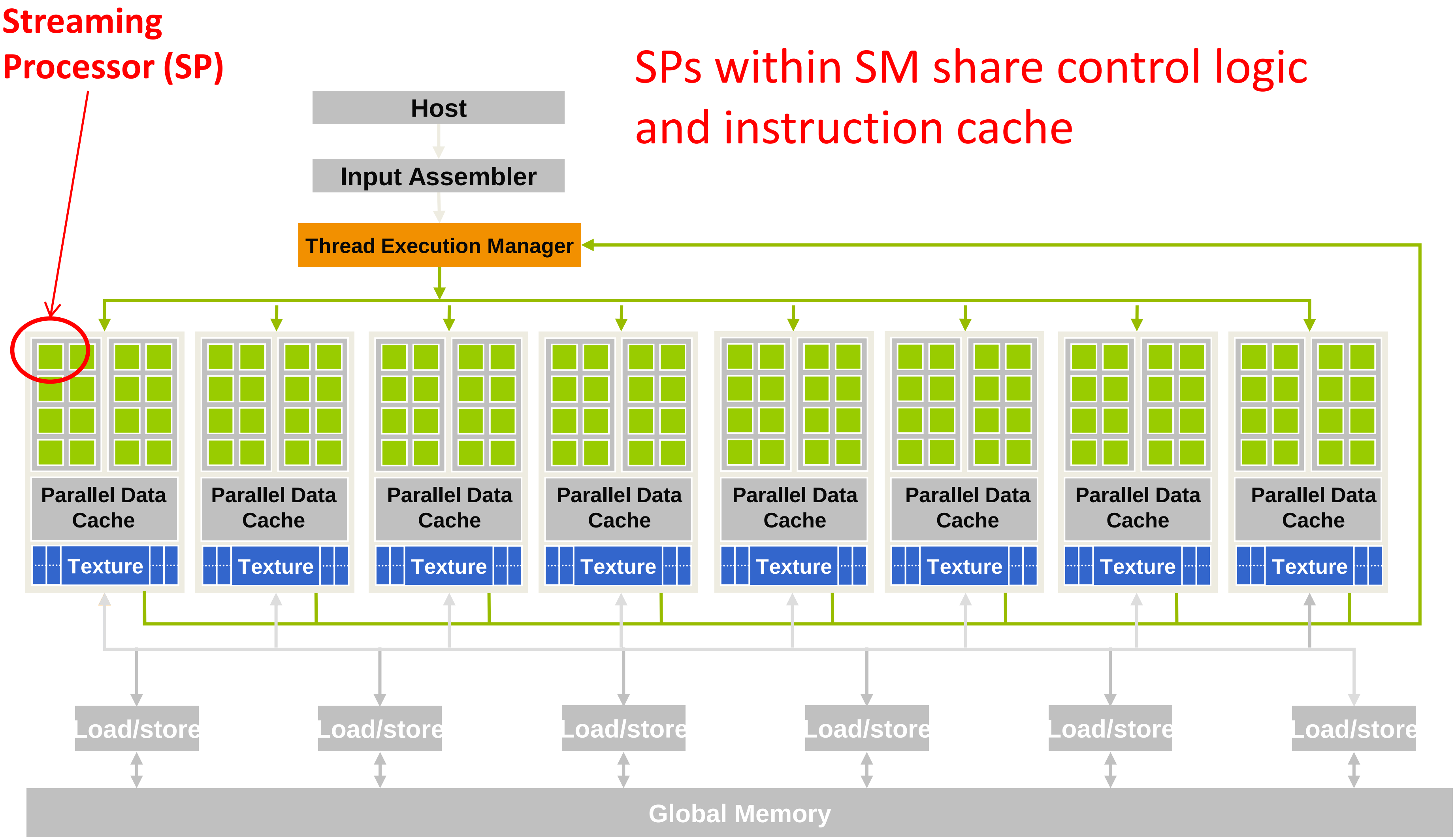
A Glimpse at A Typical GPU

Streaming Multiprocessor (SM)



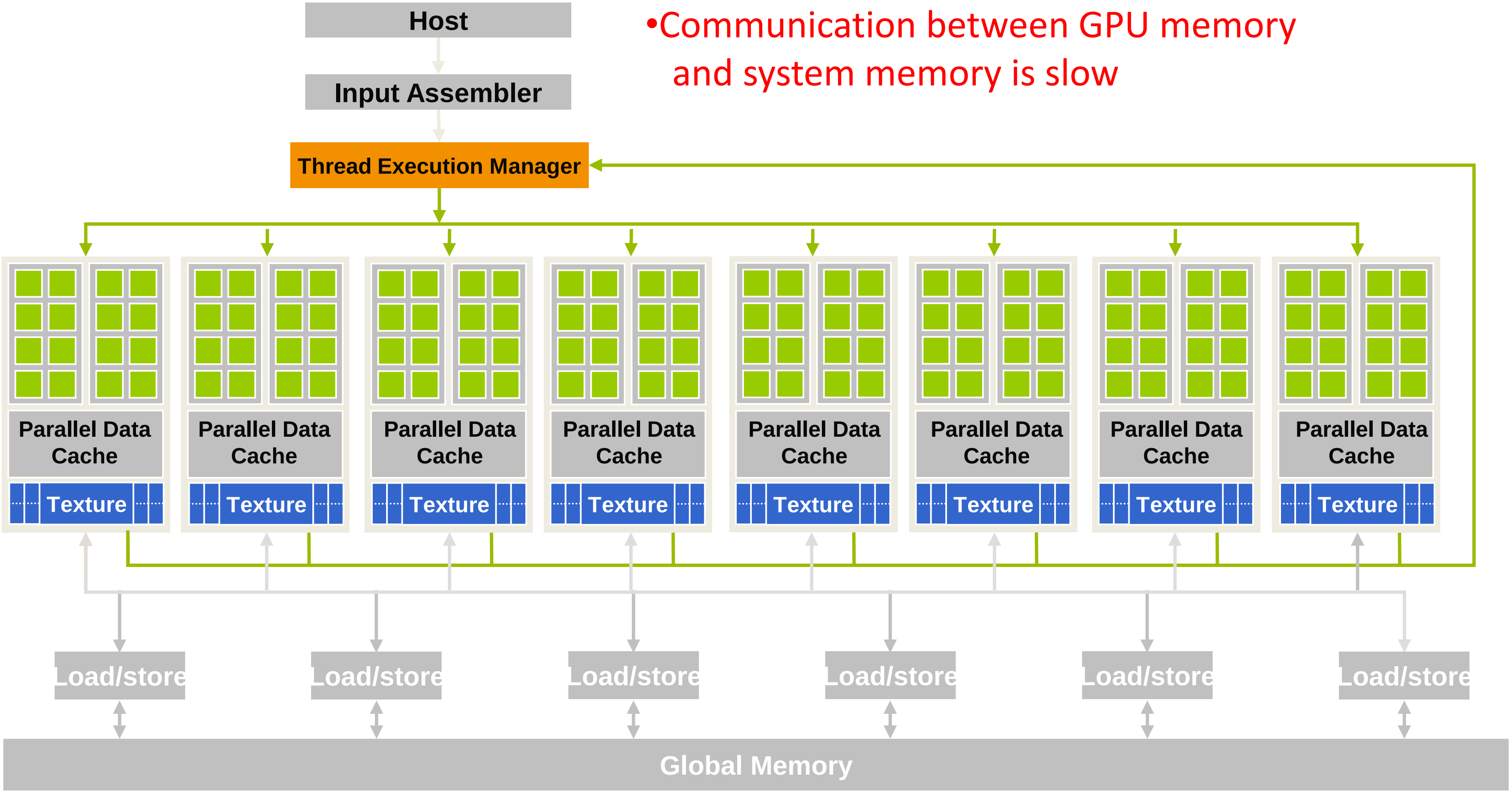
A Glimpse at A Typical GPU

SPs within SM share control logic and instruction cache



A Glimpse at A Typical GPU

- Much higher bandwidth than typical system memory
- A bit slower than typical system memory
- Communication between GPU memory and system memory is slow



BUT …

- GPU is not standalone, it needs CPU
- How to divide your program among multicore and GPU?
- Can you put part of your program into STMD/SPMD/SIMD?
- GPU → OpenCL and CUDA

Today's Lecture

- What to do with all what you heard about in this course?
- Challenges in hardware and how they affect the software
- Different type of parallelism
- Common problems in parallel programs
- A glimpse into the future
  - Software
  - Hardware

Common problems in parallel programs

- Parallel programs are subject to the usual bugs
- Plus: new timing and synchronization bugs (race condition, deadlocks, livelocks, …)
- parallel bugs often disappear when you add code to try to identify the bug ☹

Common problems in parallel programs: Too Many Threads

- (Fixed amount of work)/(large number of threads) → each thread will do too little + the overhead of threading
- overhead of threading:
  - starting ending threads
  - contention on shared resources → especially when software threads are more than hardware threads

Common problems in parallel programs: Too Many Threads

- The best strategy: limit the number of software threads to:
  - Number of hardware threads
  - Number of outer-level caches
- Also, separate your threads into I/O threads and compute threads
  - Blocked threads are not fighting for time-slice by the OS
- OpenMP takes this burden from the programmer
- Thread pool is another option (no standard one for POSIX threads though)

Common problems in parallel programs: Data Race, Deadlocks, and Live Locks

- Sometimes races are hidden by the language syntax (What may seem like a single instruction may actually be several ones.)
- Several ways to deal with that:
  - Use tools like Intel Thread checker
  - Locks
  - Transactions (which may go down to locks in some implementations!).

Common problems in parallel programs: Data Race, Deadlocks, and Live Locks

- Locks can lead to deadlocks
- Deadlocks occur when the following 4 conditions exist:
  - Access to each resource is exclusive
  - A thread is allowed to hold one resource while requesting another
  - No thread is willing to relinquish a resource it has acquired
  - There is a cycle of threads trying to acquire resources
- Deadlocks can be avoided by breaking anyone of the above four conditions, for example:
  - Replicate a resource if possible
  - Always let threads acquire locks (resources) in the order

Common problems in parallel programs: Heavily Contended Locks

- Locks can become contended → performance degradation
- What to do?
  - Replicating the resource (hence spreading the contention) can help.
  - Consider partitioning the resource and use locks for each part

Today's Lecture

- What to do with all what you heard about in this course?
- Challenges in hardware and how they affect the software
- Different type of parallelism
- Common problems in parallel programs
- A glimpse into the future
  - Software
  - Hardware

How Will the Future Look Like?

- **"I think there is a world market for maybe five computers.“**

– Thomas Watson, chairman of IBM, 1949

- **"There is no reason in the world anyone would want a computer in their home. No reason.”**

– Ken Olsen, Chairman, DEC, 1977

- **"640K of RAM ought to be enough for anybody.”**

– Bill Gates, 1981

Predicting the Future is not easy!!

How Will the Future Look Like?

- **Evolution**: just interpolating the current trend
- **Revolution**: a new technology, a paradigm shift → very hard to predict!

The Future of Software: Evolution

- Application types
- Languages
- MPI for multicore?
- Auto-parallelization
- Functional-programming
- Transactional memories

What Kind of Apps Need 100s Cores?

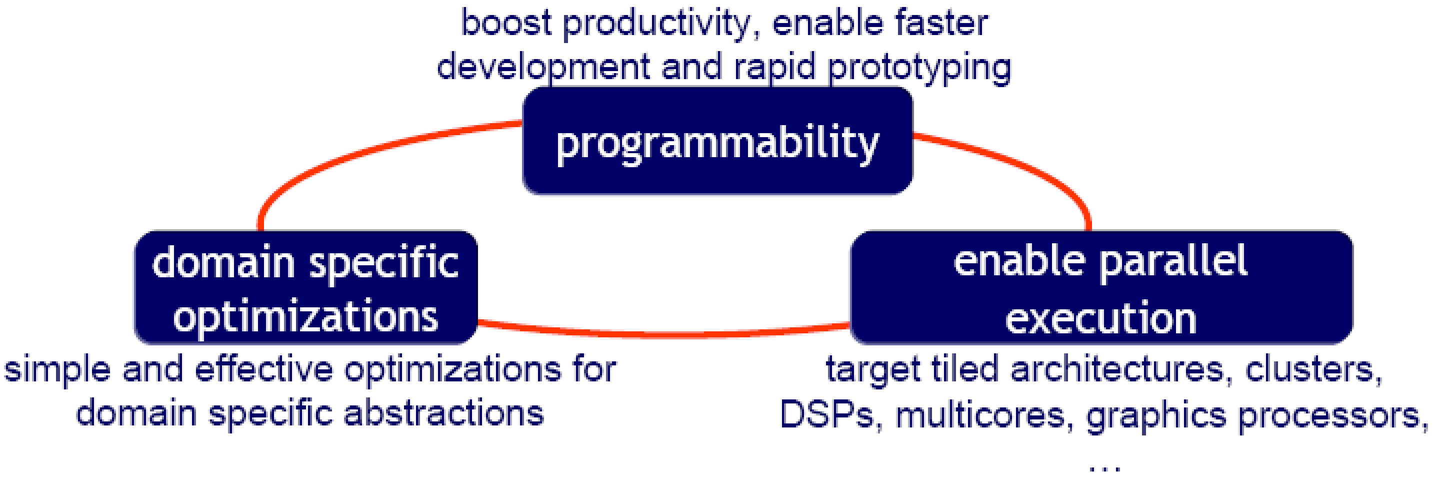
- "Who needs 100 cores to run M/S Word?"
  - Need compelling apps that use 100s of cores
- Compelling in terms of likely market or social impact, with longer term potential

Example of Evolving Applications

- Content-based image retrieval
- Health-record management
- NLP
- Teleconferences
- Heavy multimedia contents
- More realistic graphics and user interface
- Event-driven (real-time)
- …

Evolving Languages

- Scripting
- Domain specific languages (e.g. StreamIt, ...)



boost productivity, enable faster development and rapid prototyping

programmability

domain specific optimizations

simple and effective optimizations for domain specific abstractions

enable parallel execution

target tiled architectures, clusters, DSPs, multicores, graphics processors, …

Message Passing Interface (MPI)

- Message Passing Model
- Allows communication between processes (threads) using specific message-passing system calls.
- Shared data is communicated through messages
- Does not assume shared memory

Message Passing Interface (MPI)

- Allows for asynchronous events
- Does not require programmer to write in terms of loop-level parallelism
- Operates on multicore AND is scalable to very large systems
- Extremely flexible

Why then it was not adopted for multicore as the de facto?

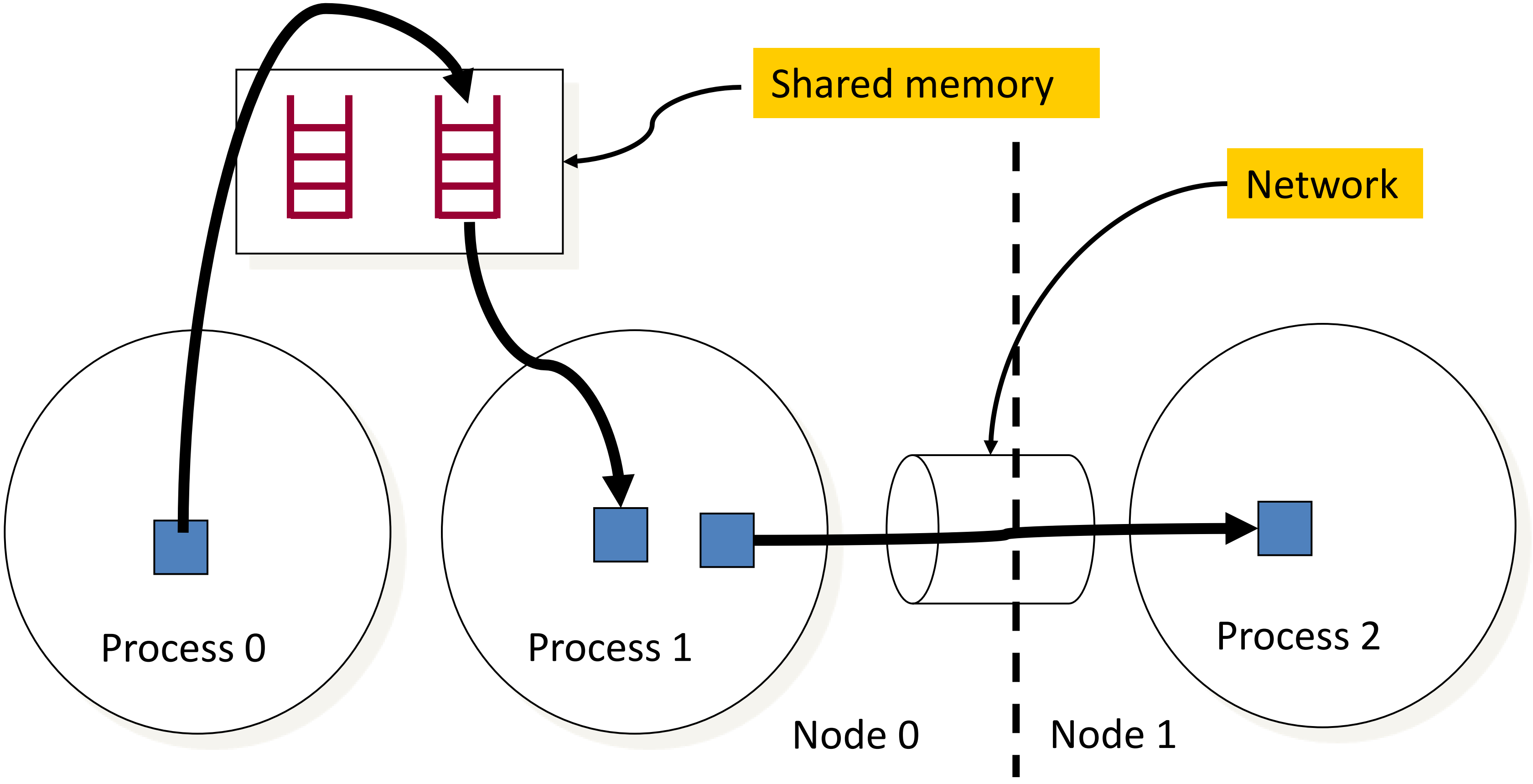
Message Passing Interface (MPI)

- Considered extremely difficult to write
  - Shared-memory models seem more intuitive
- Difficult to incrementally increase parallelism

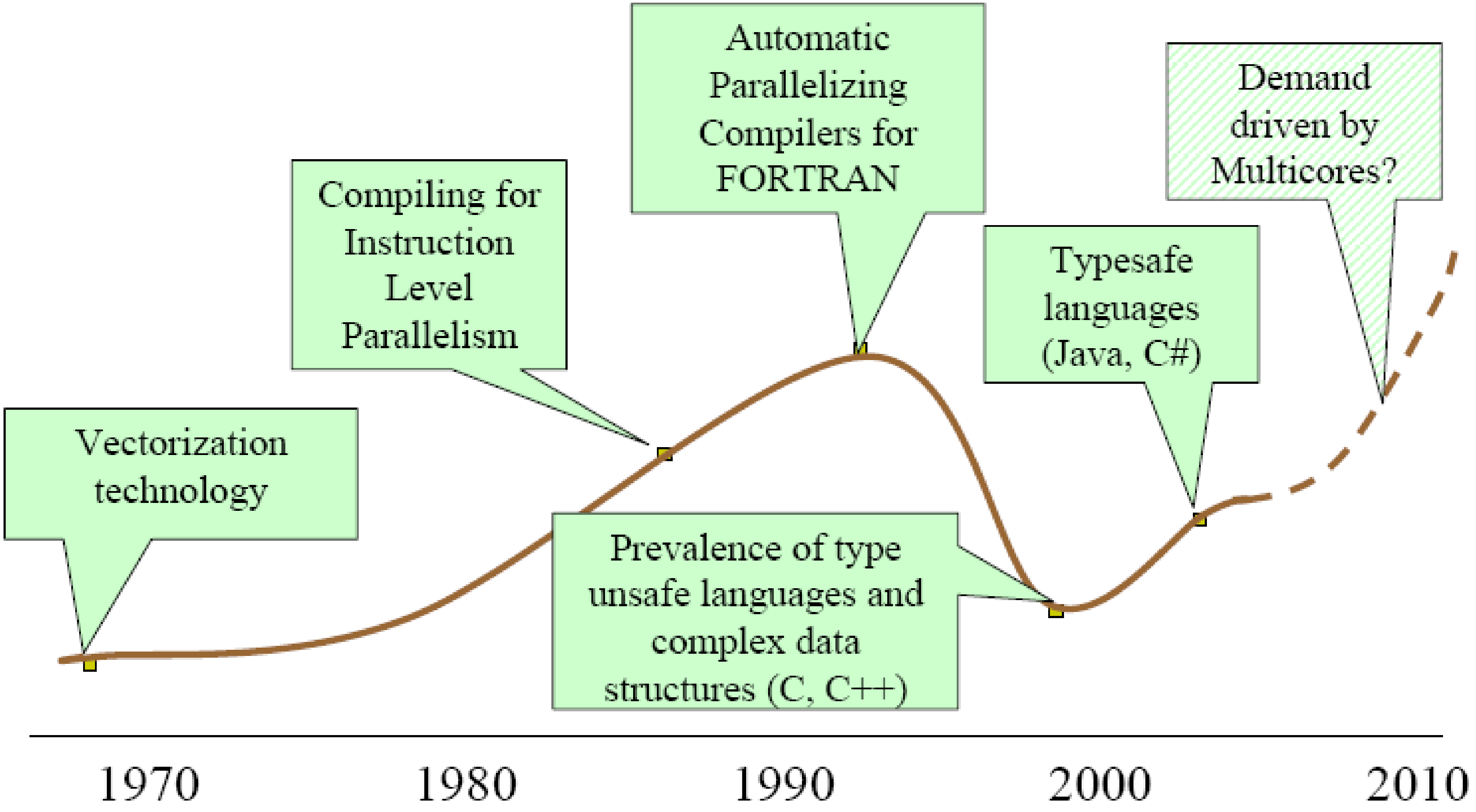
MPI on Multicore

- One MPI process per core
  - Each MPI process is a single thread
- One MPI process per node
  - MPI processes are multithreaded
  - One thread per core
  - aka Hybrid model

MPICH2 Features to Support Multicore Architechture



Improvement in Automatic Parallelization



Source: Saman Amarasinghe, MIT

What Do We Need From the Compiler?

- Compilers are critical in reducing the burden on programmers
  - Identification of data parallel loops can be easily automated, but many current systems require the programmer to do it.
- Reviving the push for automatic parallelization
  - Best case: totally automated parallelization hidden from the user
  - Worst case: simplify the task of the programmer

The Future of Software: Revolution

- Program design methodology
  - Sketching?
- New programming paradigm

The Future of Hardware: Evolution

- More cores on-chip
- Constraints increase
  - Dark-silicon becomes more substantial
  - Wire delay
  - Power
  - Reliability
- Clustered architecture on-chip
- More widespread heterogeneous multicore

The Future of Hardware: Revolution

- Brain-inspired machines
- Reconfigurable processors
- DNA computing
- Quantum computing
- Non Von-Neumann model
- …

Conclusions

- Multicore and manycore processors is a work in progress → new techniques/developments almost daily
- No magical recipe → must keep track of the software and hardware

Thank You!