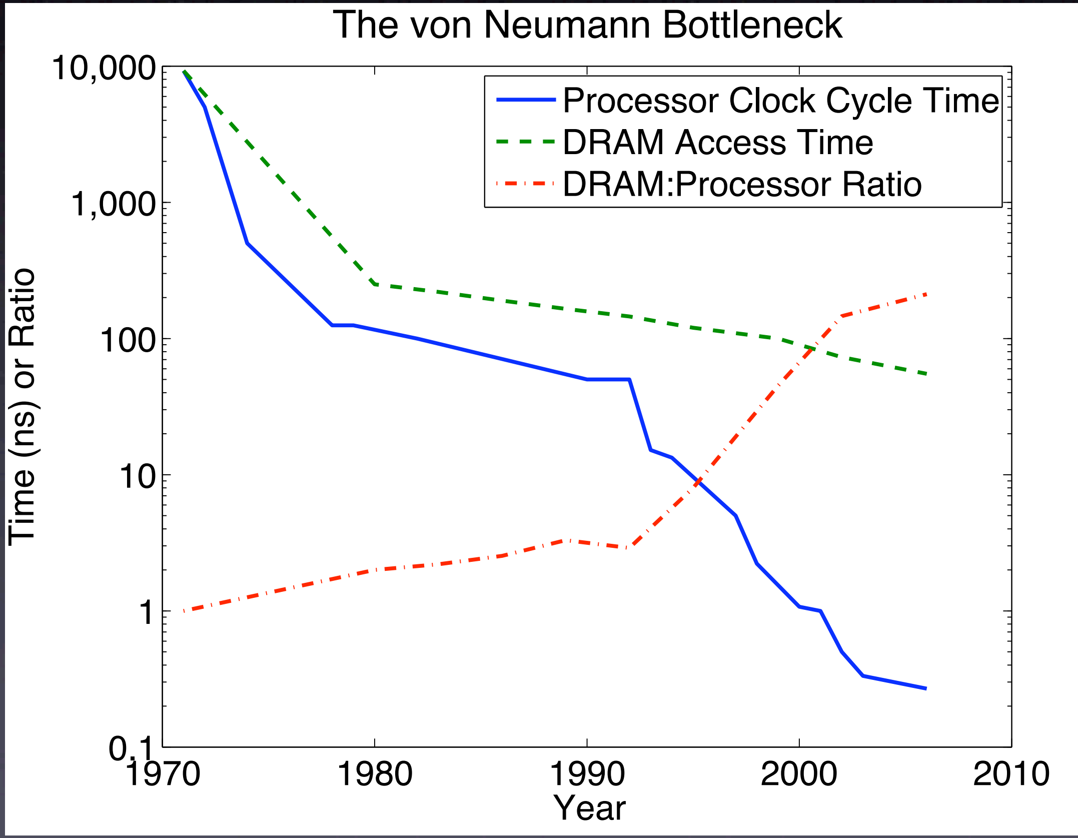
Qthreads

An API for Programming with
Millions of Lightweight Threads

The Future is Parallel

- Making CPUs faster is getting more difficult
- Running into the Memory Wall
- Two kinds of parallelism: big and small:
    - Simultaneous Multithreading, OOO Execution
    - Pthreads, MPI, OpenMP, etc.
    - Semantic Disconnect



The Dream

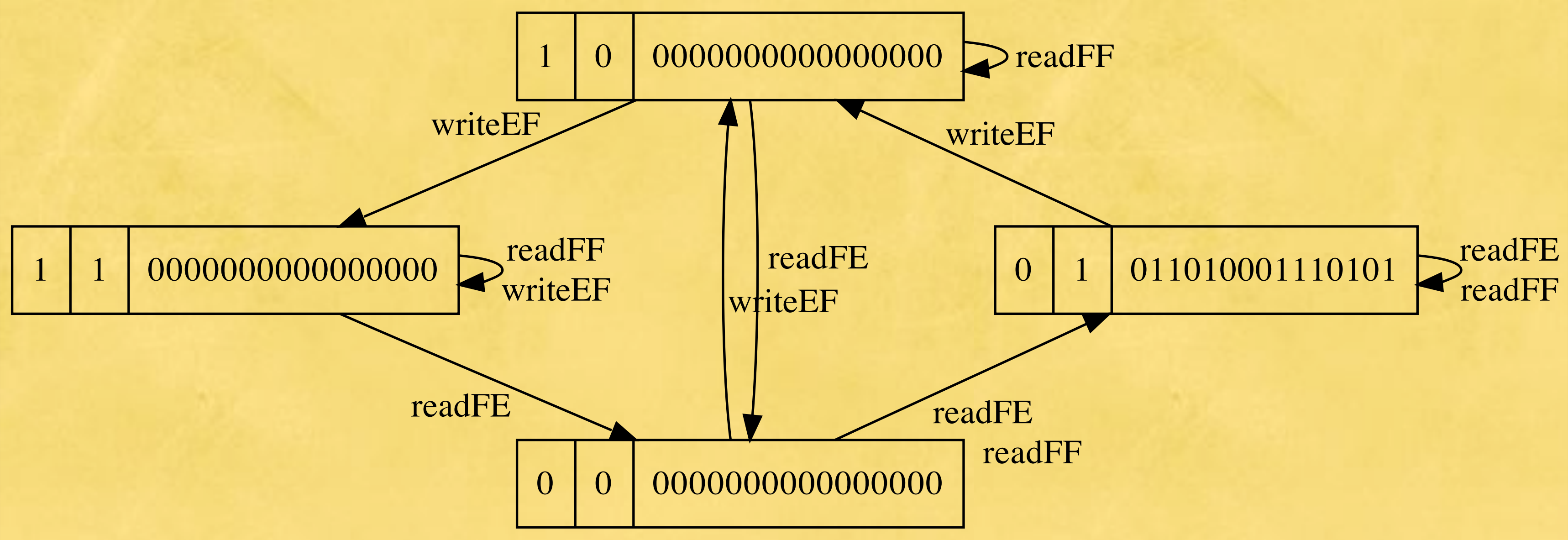
- Millions of threads! Massive Parallelism!
- BUT...
  - Per-thread overhead is a killer
  - Synchronization costs are high
    - Stop all interrupts? Trap into the kernel? Copy and send? Spin?
- Need lightweight threads & synchronization!

What is “Lightweight”?

Process	Heavy Thread	Light Thread
Flags	Flags	Flags?
Scheduler State	Scheduler State	Scheduler State?
CPU State	CPU State	CPU State
Timers	Timers	Parent Link
Timing Counters	Timing Counters	
Children list	Children list	
Parent link	Parent link	
Sibling list	Sibling list	
Loader hooks	Loader hooks	
TTY	TTY	
Priority	Priority	
PID	PID	
Groups	Groups	
UID/GID	UID/GID	
EUID/EGID	EUID/EGID	
Rlimits	Rlimits	
SysV Semaphore	SysV Semaphore	
Open File List	Open File List	
Signal Handlers	Signal Handlers	
Signal Masks	Signal Masks	
Memory Segments		
Namespace	Namespace	
...and more!...	...and more!...	

Hardware To the Rescue

- PIM/LWP and Cray MTA & XMT (ThreadStorm CPU)
  - Lightweight Threads
  - Lightweight Synchronization (FEBs)



Threading Models

- There's a Zillion
  - Some require a special compiler (OpenMP, MTA threads, nano-threads, Java pico-threads, Cilk)
  - Some aren't general-purpose (Python stackless threads, continuations, coroutines...)
  - Some simply don't scale (pthreads, TBB, GNU Pth)

Resource Management

- Failure modes: **what happens when the programmer asks for too many threads?**
- How to adapt algorithmic parallelism to available parallelism with low overhead?

Handling Exhaustion

- If we can’t create any more threads:
  - Die!
  - Finish the rest serially!
  - Wait & Hope!
- Punt to the programmer

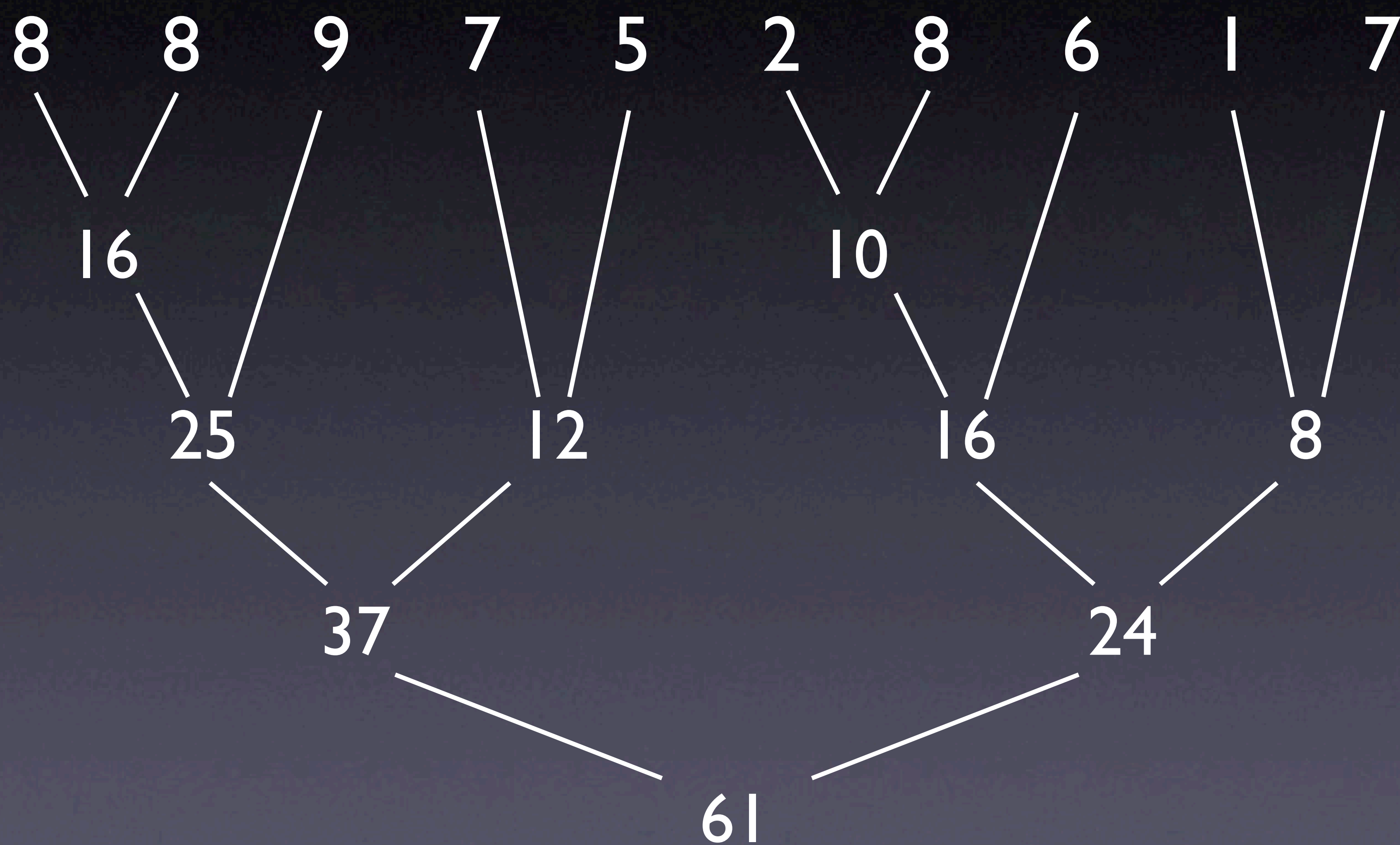
Simple Problem

- Sum an Array in Parallel
  - Recursive Tree
  - Equal Distribution (Barrier)
  - Lagging-Loop

Efficient?

Adaptive?

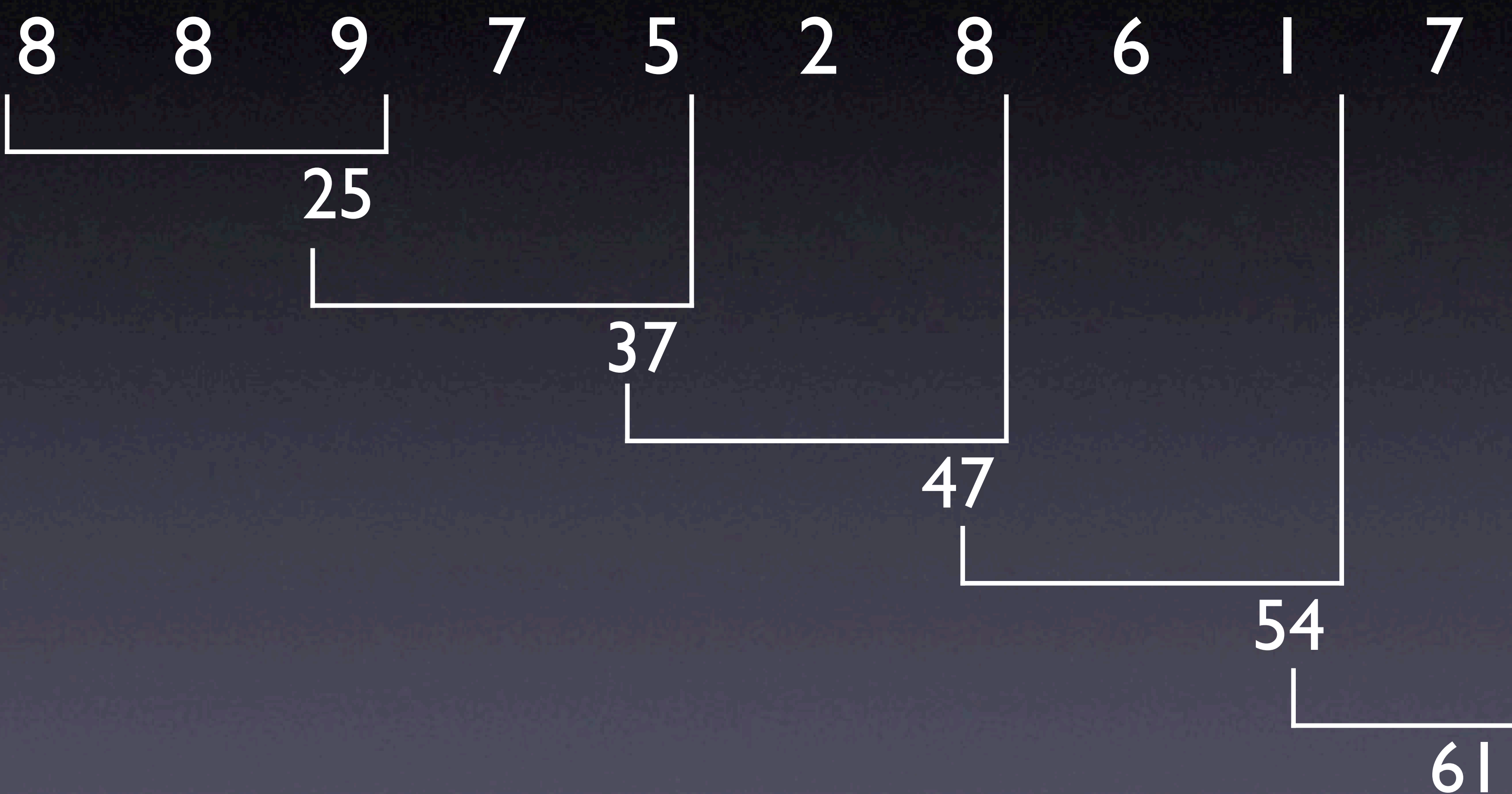
Recursive Tree



Equal Distribution

$$\underbrace{\underbrace{8 \quad 8}_{16} \quad \underbrace{9 \quad 7 \quad 5}_{21} \quad \underbrace{2 \quad 8}_{10} \quad \underbrace{6 \quad 1 \quad 7}_{14}}_{61}$$

Lagging Loop



What do we want in a threading library?

- Good failure handling
- Handle locality like hardware
- Portability (no specialized compiler)
- Debuggability?

Qthread API Provides

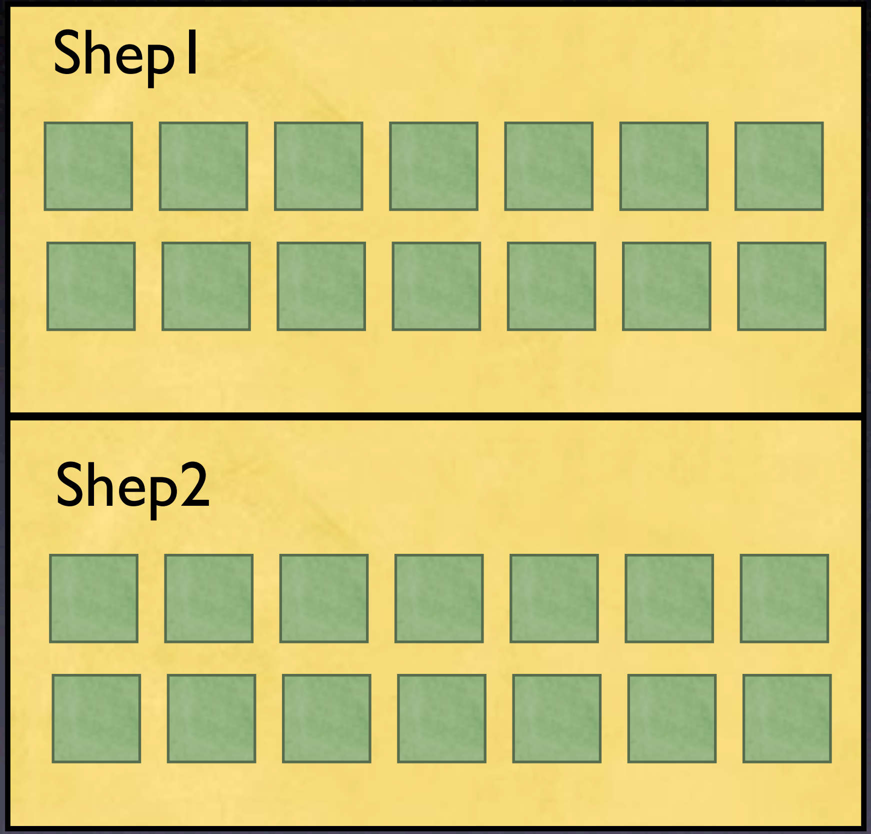
- Lightweight Threading & Synchronization
  - Efficient memory synchronization primitives
  - Explicit loop parallelization
- Thread creation fails safely & configurably
- Portable - POSIX Threads, Sandia Structural Simulation Toolkit, and soon: MTA
  - PPC, x86, IA64, Niagara2
- Debuggable

Basic Qthreading

- Resource Awareness: Qthreads vs Futures
  - qthread_init(), qthread_fork()
  - future_init(), future_fork()
- Locking
  - Mutex (lock/unlock)
  - FEB (readFE/readFF/writeF/writeEF)
- Shepherds

Shepherds

- Thread Mobility Domains
  - Threads are assigned a shepherd and do not transfer
- Unix implementation only: hierarchical threads



Conveniences

- Qutil & Qloop
  - C & C++ bindings
  - Threaded loop constructs
  - Simple examples (sum, findmin, etc.)
  - QuickSort implementation

Threaded Loops

```c
unsigned int i;
for (i = start; i < end; i += step) {
        func(NULL, argptr);
}
```

```c
qt_loop(start, stop, step, func, argptr);
```

Balanced Threaded Loops

```
size_t i;
for (i = start; i < end; i++) {
        func(argptr);
}
```

```
void func_loop(qthread_t *me, const size_t startat,
          const size_t stopat, void *arg)
{
    size_t i;
    for (i = startat; i < stopat; i++) {
        func(arg);
    }
}

qt_loop_balance(start, stop, 1, func_loop, argptr);
```

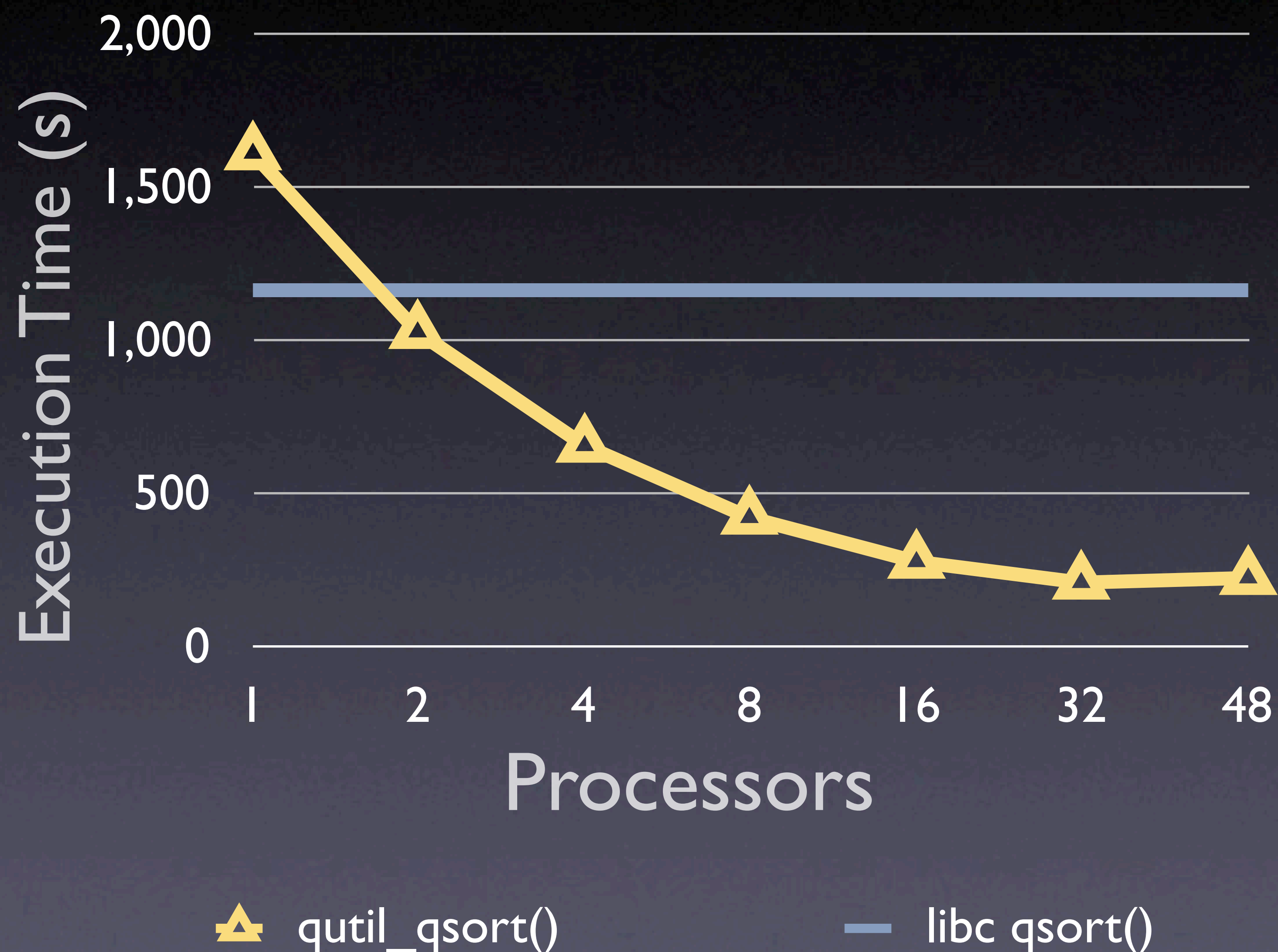
Performance

- From the Unix implementation
  - Hierarchical Threads created in Userland
  - Relies on Pthread mutexes for locking datastructure (hash)

Micro-Benchmarks

48-Node Altix

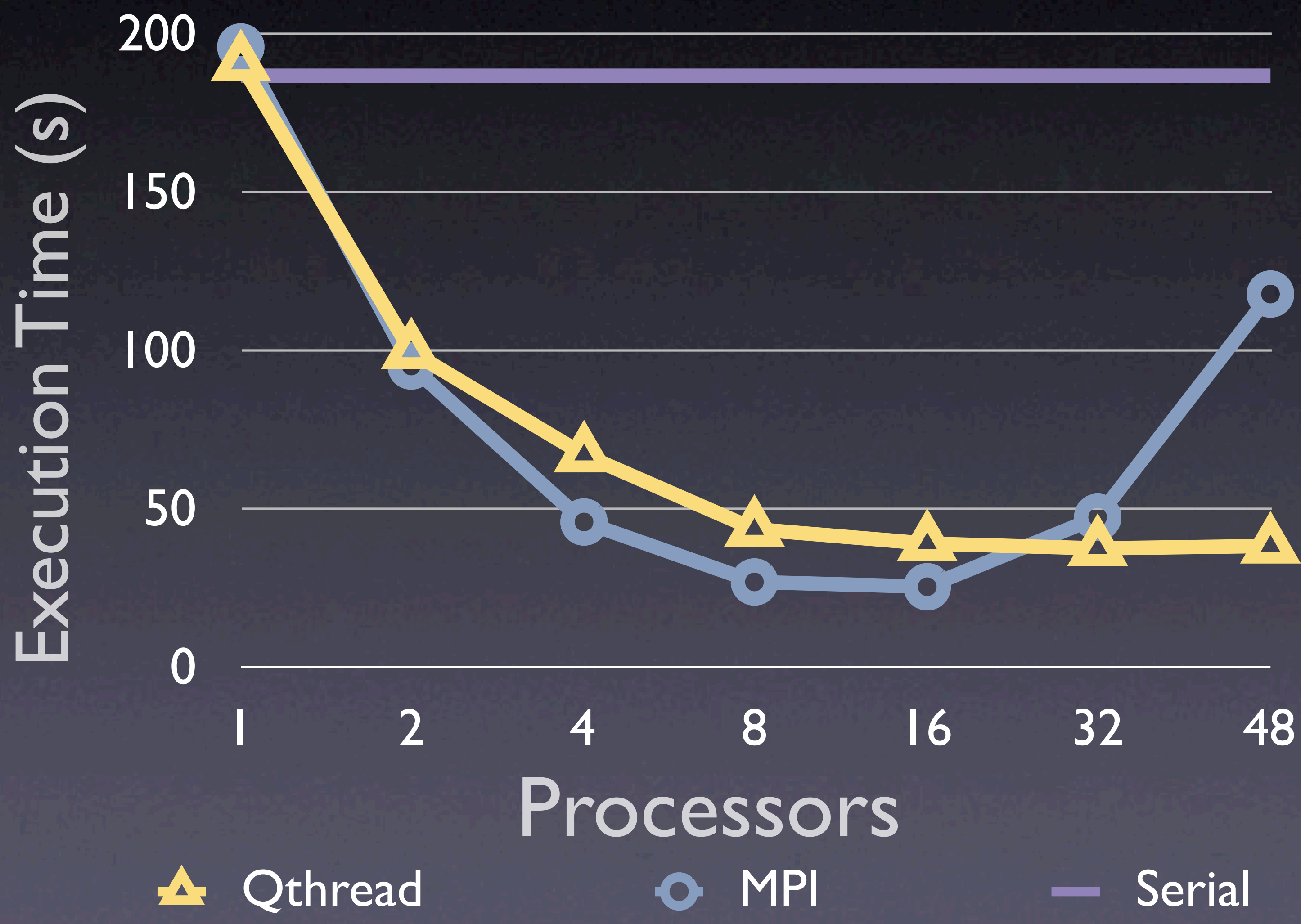
Parallel QuickSort



HPCCG

High Performance Computing Conjugate Gradient

Mantevo Project



Coming Attractions

- More benchmarks
  - MTGL
  - Mantevo (phdMesh, Epetra, LAMMPS)
- More portability
  - Niagara2, MTA

Download Now!

http://www.cs.sandia.gov/qthreads/

Many thanks to Dr. Murphy, Dr. Thain, Megan Vance, Dr. Rodriguez, Dr. Kogge, the University of Notre Dame, and Sandia National Laboratories