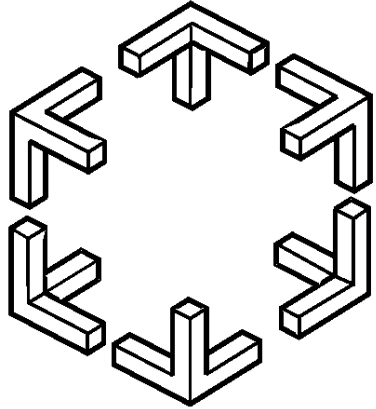




HÖGSKOLAN I BORÅS
VETENSKAP FÖR PROFESSION

Distributed Computing and Systems

Chalmers university of technology



NOBLE: Non-Blocking Programming Support via Lock-Free Shared Abstract Data Types



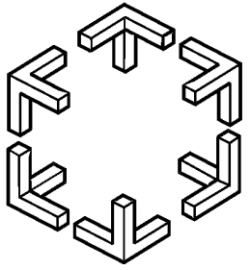
Håkan Sundell

University College of Borås

Parallel Scalable Solutions AB

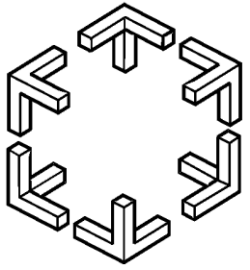
Philippas Tsigas

Chalmers University of Technology



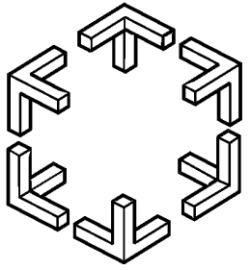
Outline

- Multicore in software development
 - Requirements
- Problems in parallel/multithread development
 - Description of core issues
 - Traditional solutions
- Our solutions
 - Technology
 - Products



Multicore: Important Issues

- Software Scalability
 - More logical CPU's increase overall performance?!
- Software Reliability
 - Correctness?
 - Fault-tolerance?
 - Predictable?

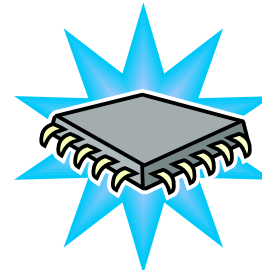
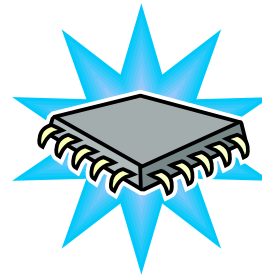
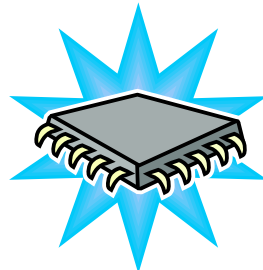


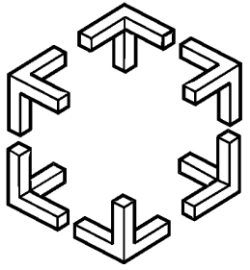
(Parallel) Software

- Programs consist of many tasks (threads)



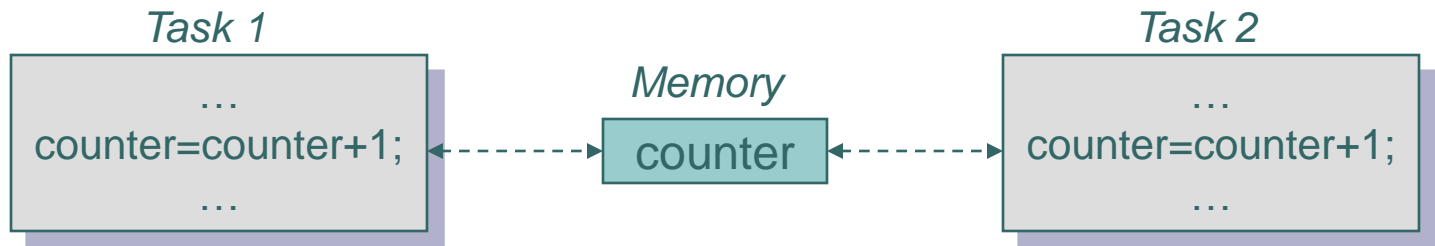
- That execute on one or more (logical) processors

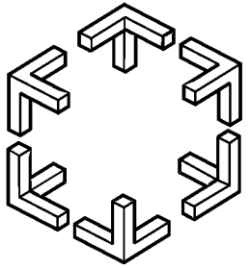




Communication

- Tasks need to communicate and work with shared resources.
 - Message Passing
 - High overhead and Abstract system design.
 - Shared Memory
 - Fast and Intuitive

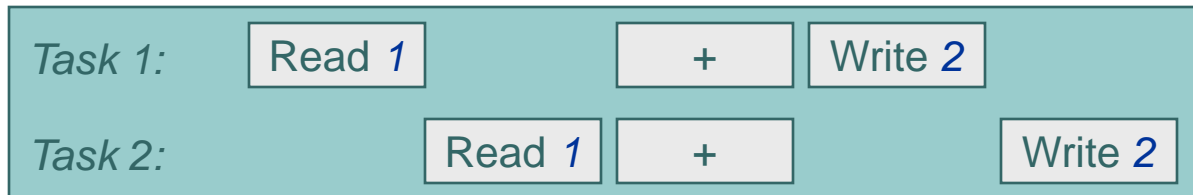




Critical Sections

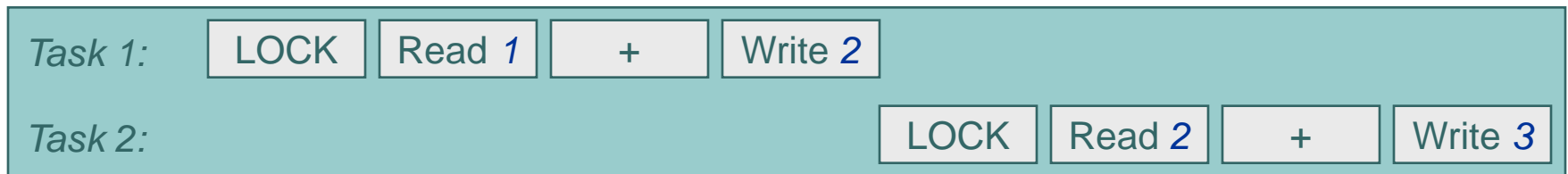
- Problem: operations on shared variables in programming languages are not atomic.

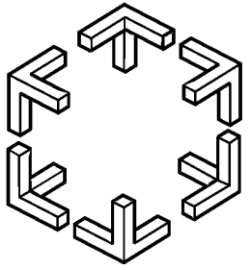
counter=counter+1; = Read + Write



! counter=2, but should be 3!

- Straightforward solution: Apply mutual exclusion

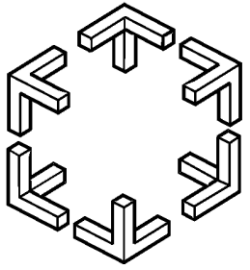




Scheduling

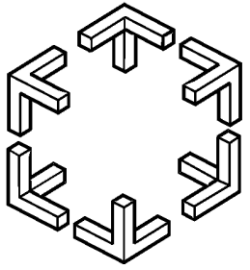
- Problem: We have many tasks and few processors.
 - Each task might have deadlines
 - Each task might have different importance (priorities)
- Solution: Tasks must be scheduled to alternately use the processors.
 - Scheduling algorithm (RM, EDF etc.)
 - Worst case execution time analysis
 - Schedulability analysis (i.e. deadlines must be met).





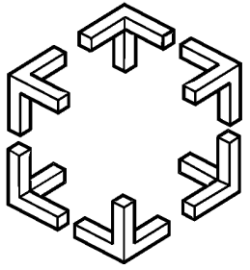
Multiprocessors

- Needs for performance are increasing
 - Single processor speed has reached its limit (due to power/heat)
 - Single processor in embedded systems might have even lower speed due to environmental constraints (temperature range, humidity etc.)
- We need multiple processors to increase performance!
- Multi-Core!



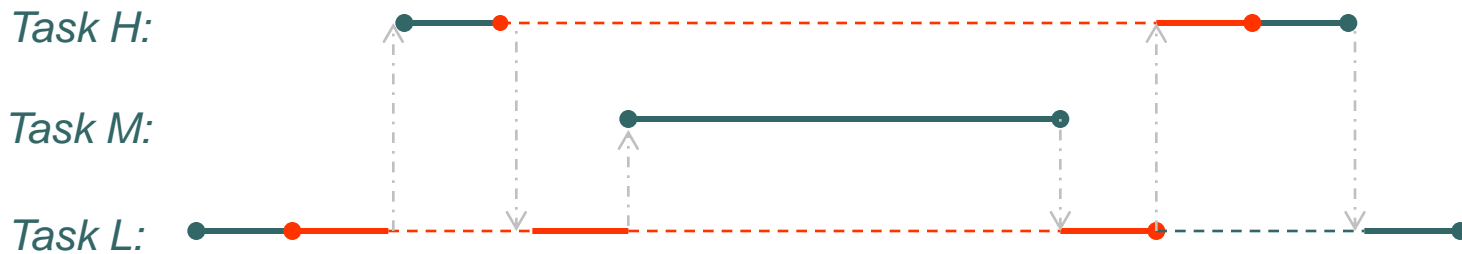
Solutions are not compositional!

- Critical Sections + Scheduling
 - **Blocking.** More advanced and pessimistic schedulability analysis.
 - **Deadlocks.** Reduced fault-tolerance, if one task fails, other (even all) might also fail.
 - **Priority Inversion.** Tasks might not execute with the proper priority even though it was set. Deadlines might be missed.

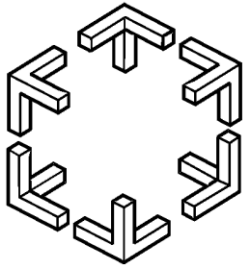


Priority Inversion

- A high priority task is delayed due to a low priority task holding a shared resource. The low priority task is delayed due to a medium priority task executing.

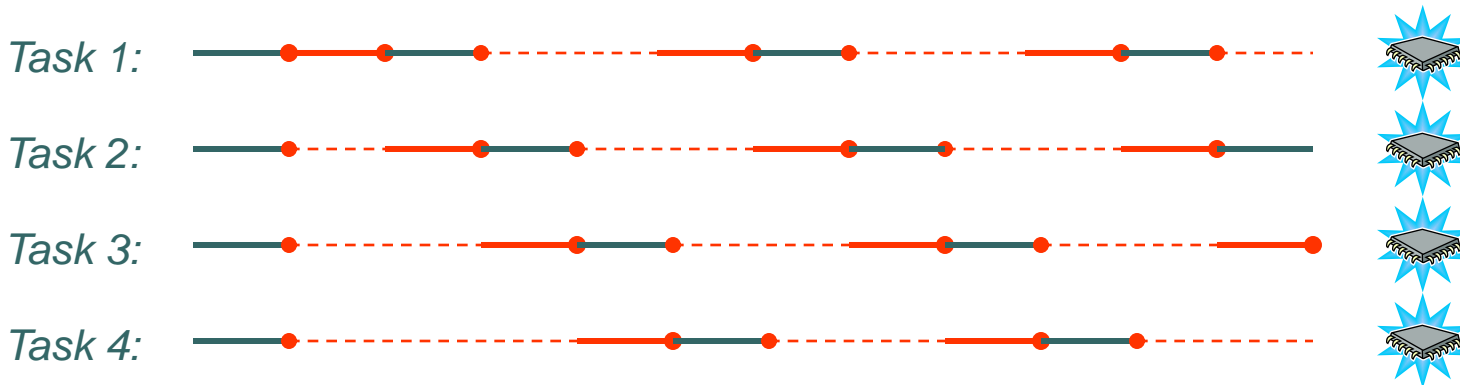


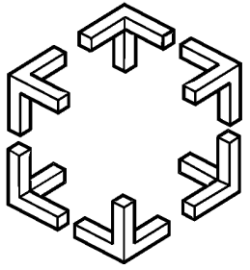
- Solutions: Priority inheritance protocols
 - Works ok for single processors, but definitely not for multiple processors!



Critical Sections + Multiprocessors

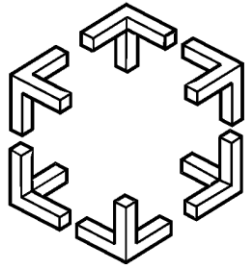
- **Reduced Parallelism.** Several tasks with overlapping critical sections will cause waiting processors to go idle.





Problems: Summary

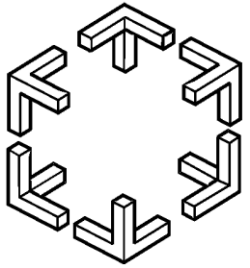
- Software Systems need:
 - Shared resources and Communication.
 - Dependency/Stability and Fault-tolerance.
 - Increased Performance and Utilization.
- Current solutions simply do not fit!
 - Communication can be a bottleneck as well as a source for difficult problems.



Communication+Scheduling+Uni/Multiprocessors:

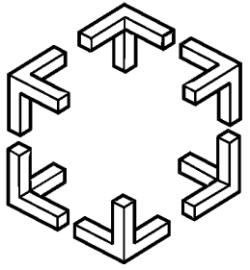
New Solution

- Avoid Critical Sections!
 - **Avoid Blocking.** Easier and more optimistic analysis, i.e. less hardware needed.
 - **Avoid Deadlocks.** Increased fault-tolerance as failed tasks can not affect others to fail.
 - **Avoid Priority Inversion.** Easier and more reliable analysis, and avoids complex and high-overhead solutions.
 - **Increased Parallelism.** Increased overall performance, more optimistic analysis, i.e. less hardware needed.



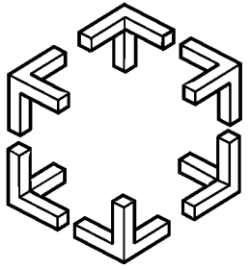
Non-Blocking Synchronization

- The key lies in how mutual exclusion (i.e. mutex, semaphore) is implemented in actual hardware (i.e. processors).
 - Atomic primitives in hardware can atomically update one memory word.
- Sophisticated solutions can exploit the same atomic primitives to support access to shared resources without locks, i.e. non-blocking.



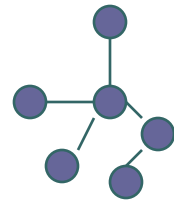
Non-Blocking Algorithms

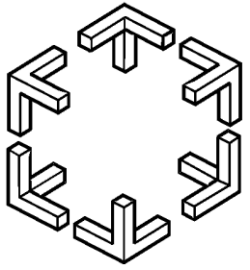
- **Obstruction-Free.**
 - Guarantees progress in absence of contention.
 - Need extra module for contention management.
- **Lock-Free.**
 - Guarantees that always one operation is making progress.
 - Combined with scheduling information, schedulability analysis can be done.
- **Wait-Free.**
 - Guarantees that any operation will finish in a finite time.
 - Schedulability analysis can be done directly.



Non-Blocking Algorithms

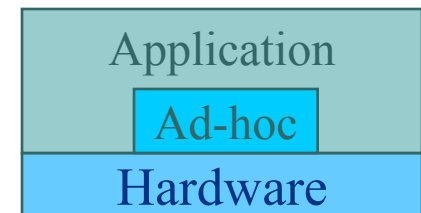
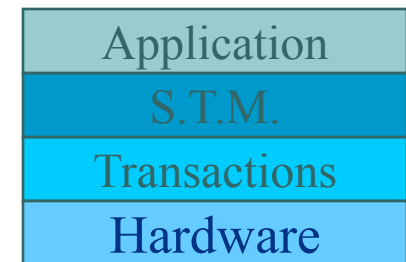
- The counter-problem can obviously be solved with atomic primitives.
 - Supported for example by Win32 API (Interlocked operations)
 - Supported atomic primitives are read-modify-write instructions, i.e. one-word transactions.
- However, programmers need to share more complex structures of data!
 - Can also more advanced data structures be constructed non-blocking?

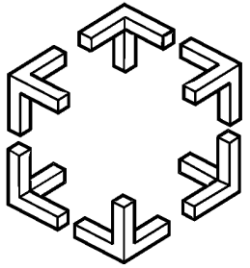




Shared Data Structures

- Several approaches without locks exist.
- Software Transactional Memory
 - General solutions.
 - In research.
 - High overhead, low performance.
- Ad-hoc solutions
 - Specific solutions for each particular type of data structure.
 - Best possible performance.





Ad-hoc Shared Data Structures

- **NOBLE**

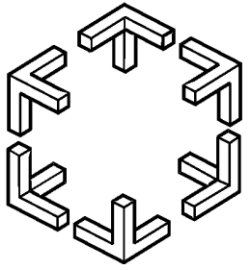
- Large commercial software library for C/C++

- **Real-Time Java**

- Uses Wait-Free Queues for communication.

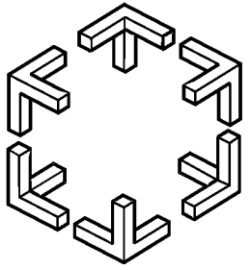
- **Java 1.6, Concurrency package**

- Contains several non-blocking data structures for developers.



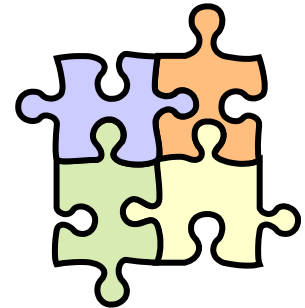
Commercial Tools

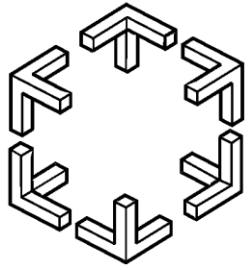
- Intel Threading Building Blocks
 - C++
 - Windows, Linux, Mac
- Microsoft Parallel Extensions to .NET Framework 3.5 (Preview)
 - C#
- NOBLE
 - C/C++
 - Windows, Linux, Mac, Unix



NOBLE Professional Edition: Contents

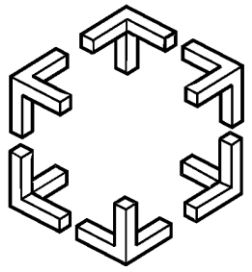
- Memory Management
 - Memory allocation
 - Memory reclamation (garbage collection)
- Atomic primitives
 - Single-word and Multi-word transactions.
- Common shared data structures
 - Stack
 - Queue
 - Deque
 - Priority Queue
 - Dictionary
 - Linked Lists
 - Snapshots





Properties of ad-hoc data structure algorithms

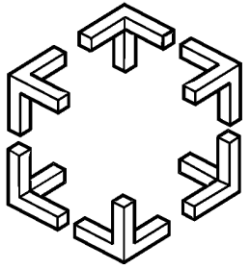
- Sequential Time Complexity
- Space Complexity
- Scalability
- Overall Performance
- Semantics
- Locality
- Dependencies and Limitations



NOBLE Professional Edition: Design

- **Easy to use.** Hides the underlying complexity and shows a common and simple interface.
- **Versatile.** Contains lock-based as well as lock-free/wait-free implementations of each component.
- **Efficient.** Designed for best possible performance.
- **Object-oriented.** Designed in C, but can easily be encapsulated in C++ or other language constructs.
- **Configurable.** A lot of optional parameters and functionalities that can be set/tuned to meet specific needs.





NOBLE Professional Edition: Example



Globals

```
#include <Noble.h>
NBLQueueRoot* queue;
```

Main

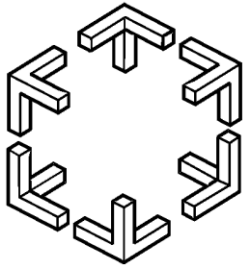
```
queue=NBLQueueCreateLF();
/* Create and run the threads */
NBLQueueFree(queue);
```

Thread 1

```
NBLQueue *handle=
NBLQueueGetHandle(queue);
...
NBLQueueEnqueue(handle, item);
...
NBLQueueFreeHandle(handle);
```

Thread 2

```
NBLQueue *handle=
NBLQueueGetHandle(queue);
...
item=NBLQueueDequeue(handle);
...
NBLQueueFreeHandle(handle);
```



C++ Example (Dictionary + Memory)

Globals

```
class MyObject {  
public:  
    int x;  
    int y;  
    int z;  
};  
NBL::Memory<MyObject> * memory;  
NBL::Dictionary<MyObject,int> *dictionary;
```

Thread 1

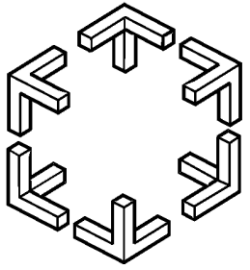
```
MyObject * obj1 = memory->AllocBlock();  
obj1->x = 1;  
obj1->y = 2;  
obj1->z = 3;  
dictionary->Insert(1,obj1);  
memory->ReleaseRef(obj1);
```

Thread 2

```
MyObject * obj1 = dictionary->Find(1);  
...  
memory->ReleaseRef(obj1);
```

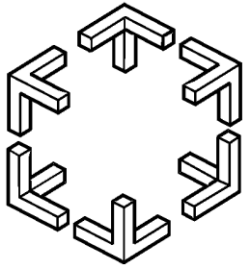
Main

```
NBL::Memory<MyObject> * memory = NBL::Memory<MyObject>::CreateLF_SUU();  
dictionary = NBL::Dictionary<MyObject,int>::CreateLF_EB();  
dictionary->SetValueMemoryHandler(memory);
```



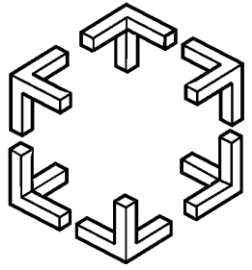
Obstacles

- Semantics
 - Various Semantics used in Literature
 - Extend, Modify and Optimize
- Memory Consistency
 - Memory Barriers needed
 - Excessive Care
- Application Integration
 - User Level Objects



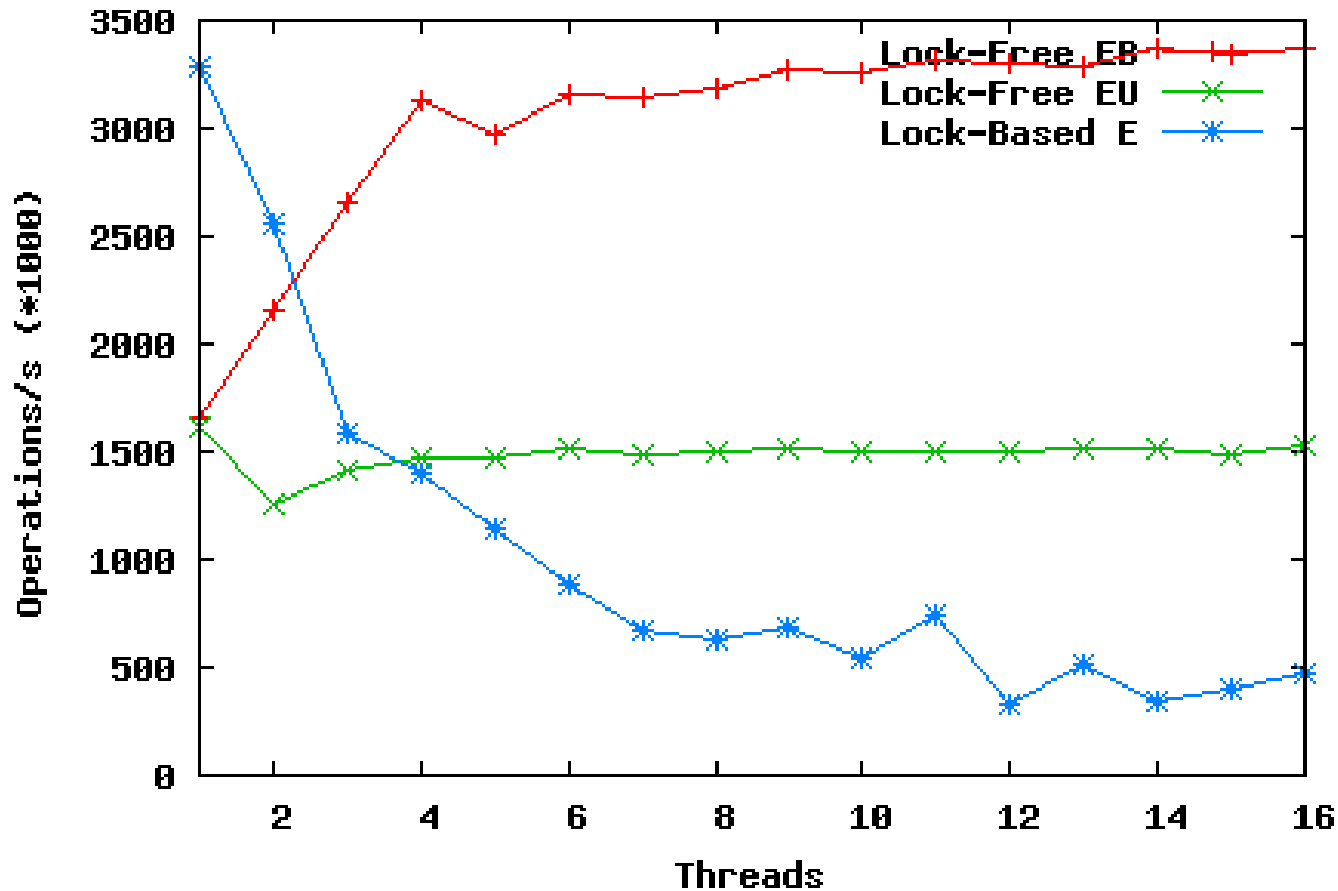
Obstacles

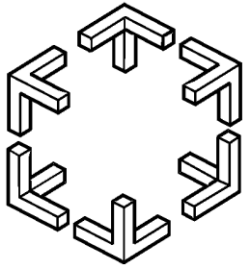
- Languages
 - C
 - C++
 - Thread Local Storage is Limited
- Process vs Threads
 - Code vs Data address ranges on different CPU's
- Architectural Differences
 - Performance Tweaking (Back-off, ...)



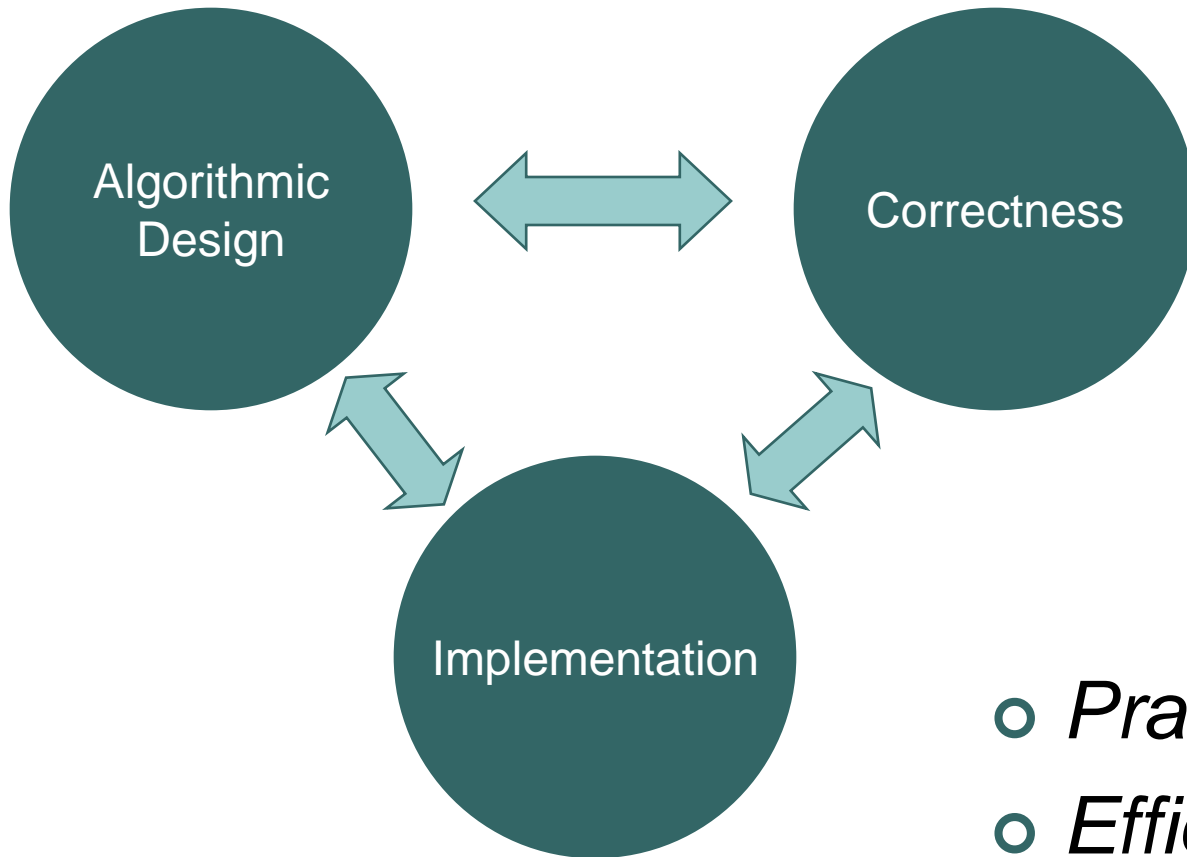
Benchmarks and Improvements

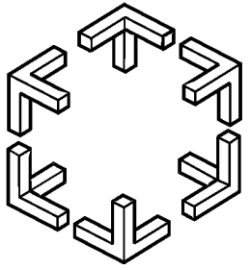
Dictionary (1000) - AMD 2x2 2.2 GHz, MinXP
Maximum Contention





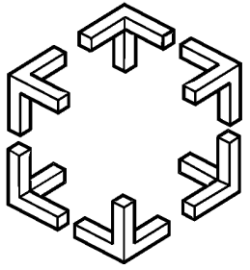
How we work





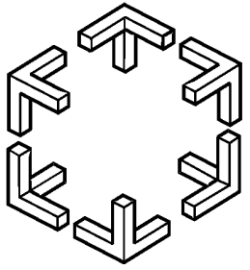
Correctness

- Parallel software have infinitely many running scenarios and interleavings.
 - Testing is not enough for being sure.
 - Needs proofs of correctness.
 - Machine-made proofs can be very time-consuming and difficult for humans to read.
- Our approach
 - Intuitive readable-text proofs, using analytical and mathematical oriented methods.



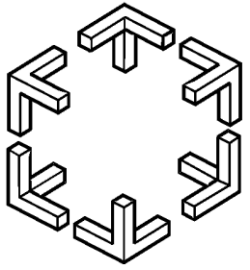
Products and Availability

- NOBLE Professional Edition
 - Details described in the Reference Manual (>250 pages)
 - License model
 - Commercial, Based on number of Developers
 - Available for modern platforms
 - x86, sparc, powerpc, mips etc.
 - Win32 (32/64-bit), Linux etc.
 - Custom



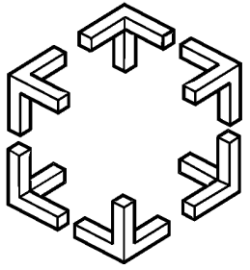
Availability

- NOBLE Professional Edition
 - Freely available to academia for educational purposes
 - Platforms of choice
 - Licensed by application
 - Free Demo Version
 - Windows
 - Visual Studio C++



Future Work

- More functionality
- Improved support for object-oriented languages and environments
 - Managed environments
- More experiments
- Dissipation



Questions?

Thank You for listening!

www.pss-ab.com

www.adm.hb.se/~hsu

www.cse.chalmers.se/~tsigas