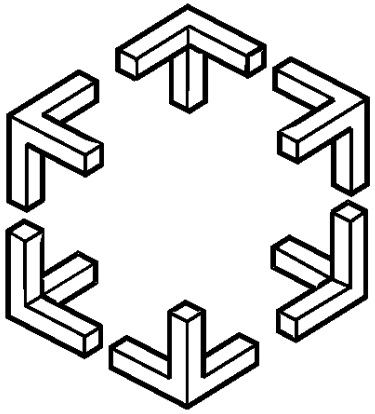
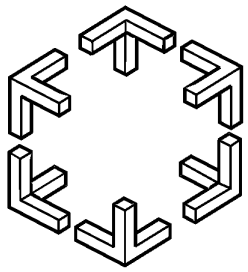
Distributed Computing and Systems
Chalmers university of technology



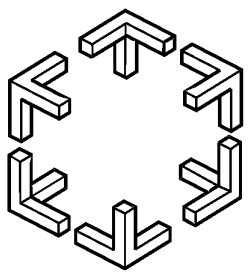
Fast and Lock-Free Concurrent Priority Queues for Multi-Thread Systems

Håkan Sundell
Philippas Tsigas



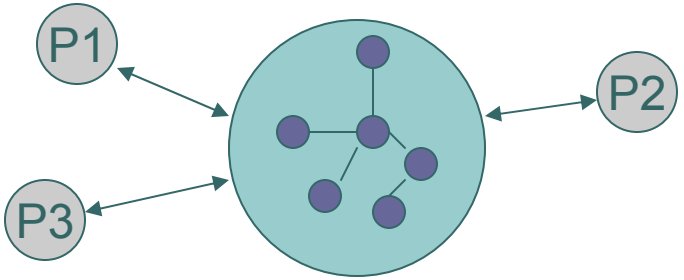
Outline

- Synchronization Methods
- Priority Queues
- Concurrent Priority Queues
  - Lock-Free Algorithm: Problems and Solutions
- Experiments
- Conclusions

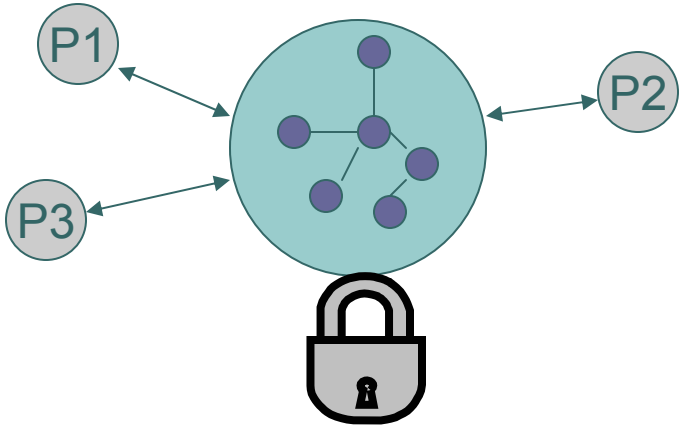


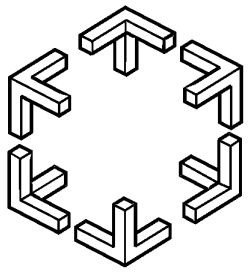
Synchronization

- Shared data structures needs synchronization



- Synchronization using Locks
  - Mutually exclusive access to whole or parts of the data structure

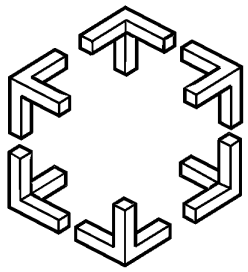




Blocking Synchronization

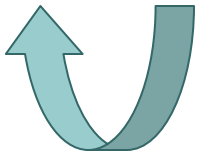
- Drawbacks
  - Blocking
  - Priority Inversion
  - Risk of deadlock

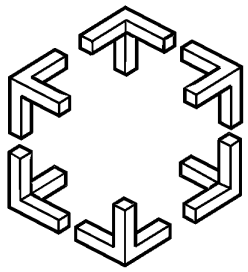
- Locks: Semaphores, spinning, disabling interrupts etc.
  - Reduced efficiency because of reduced parallelism



Non-blocking Synchronization

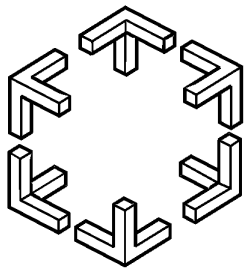
- Lock-Free Synchronization
  - Optimistic approach
    - Assumes it's alone and prepares operation which later takes place (unless interfered) in one atomic step, using hardware atomic primitives
    - Interference is detected via shared memory and the atomic primitives
    - Retries until not interfered by other operations
      - Can cause starvation



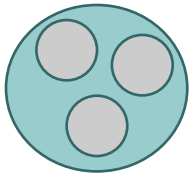


Non-blocking Synchronization

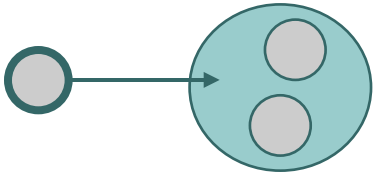
- Lock-Free Synchronization
  - Avoids problems with locks
  - Simple algorithms
  - Fast when having low contention
- Wait-Free Synchronization
  - Always finishes in a finite number of its own steps.
    - Complex algorithms
    - Memory consuming
    - Less efficient in average than lock-free



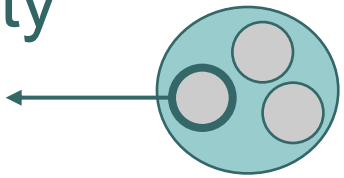
Priority Queues

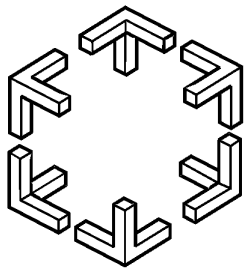


- Fundamental data structure
- Works on a set of <value,priority> pairs
- Two basic operations:
  - Insert(v,p): Adds a new element to the priority queue



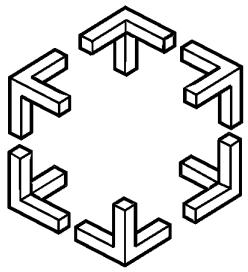
  - v=DeleteMin(): Removes the element <v,p> with the highest priority





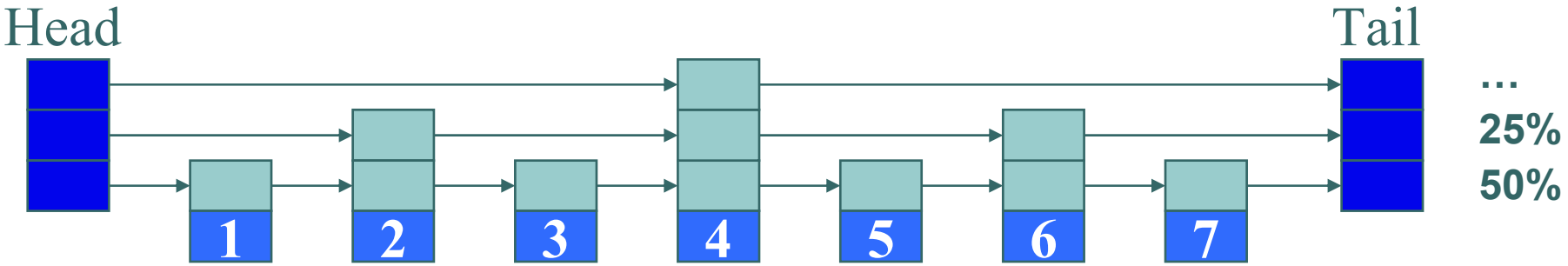
Sequential Priority Queues

- All implementations involves search phase in either Insert or DeleteMin
  - Arrays. Maximum complexity O(N)
  - Ordered Lists. O(N)
  - Trees. O(log N)
    - Heaps. O(log N)
  - Advanced structures (i.e. combinations)

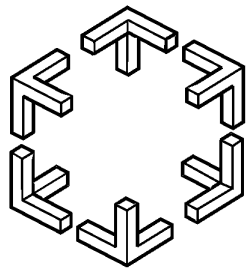


Randomized Algorithm: Skip Lists

- William Pugh: ”Skip Lists: A Probabilistic Alternative to Balanced Trees”, 1990
  - Layers of ordered lists with different densities, achieves a tree-like behavior

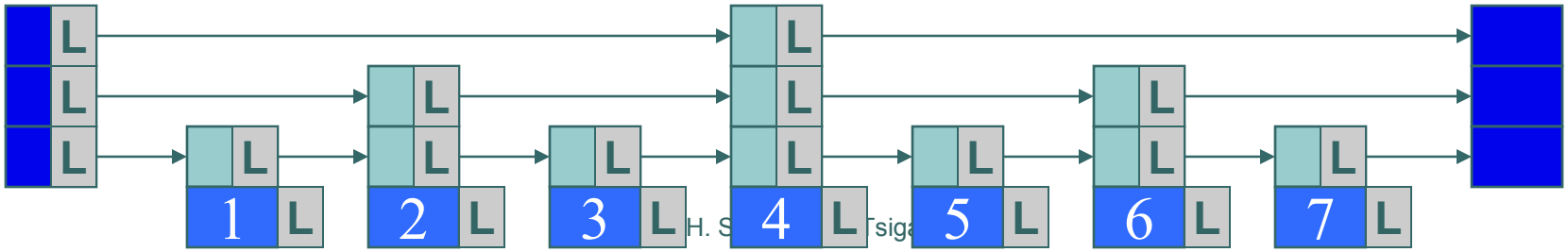


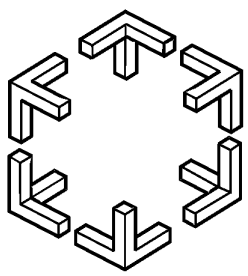
  - Time complexity: $O(\log_2 N)$ – probabilistic!



Why Skip Lists for Concurrent Priority Queues?

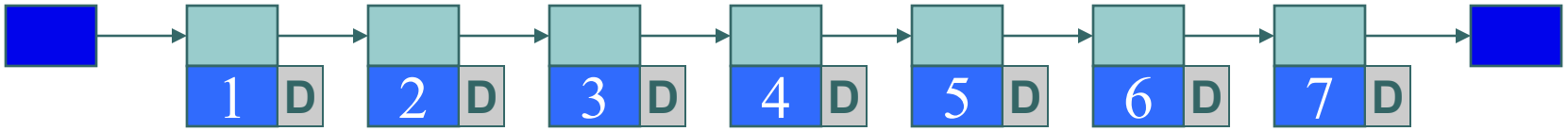
- Ordered Lists is simpler than Trees
  - Easier to make efficient concurrently
- Search complexity is important
  - Skip Lists is an alternative to Trees
- Lotan and Shavit: “Skiplist-Based Concurrent Priority Queues”, 2000
  - Implementation using multiple locks

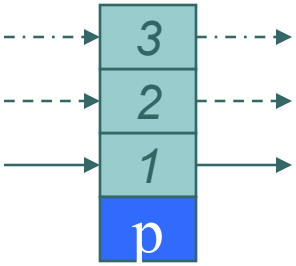




Our Lock-Free Concurrent Skip List

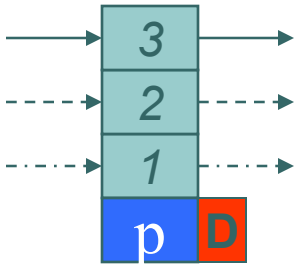
- Define node state to depend on the insertion status at lowest level as well as a deletion flag

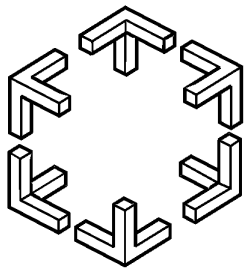




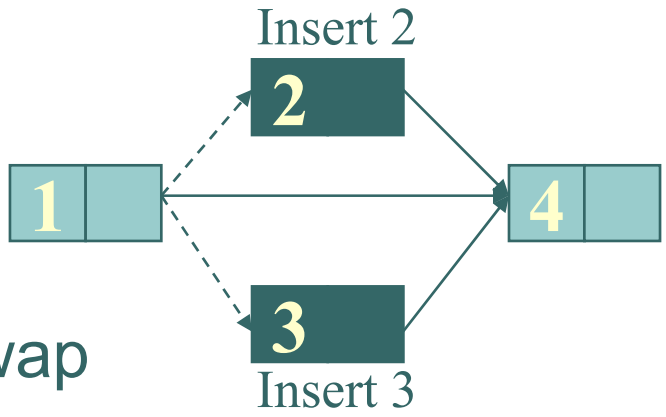
- Insert from lowest level going upwards

- Set deletion flag. Delete from highest level going downwards



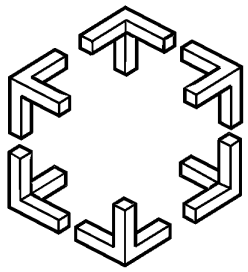


Overlapping operations on shared data



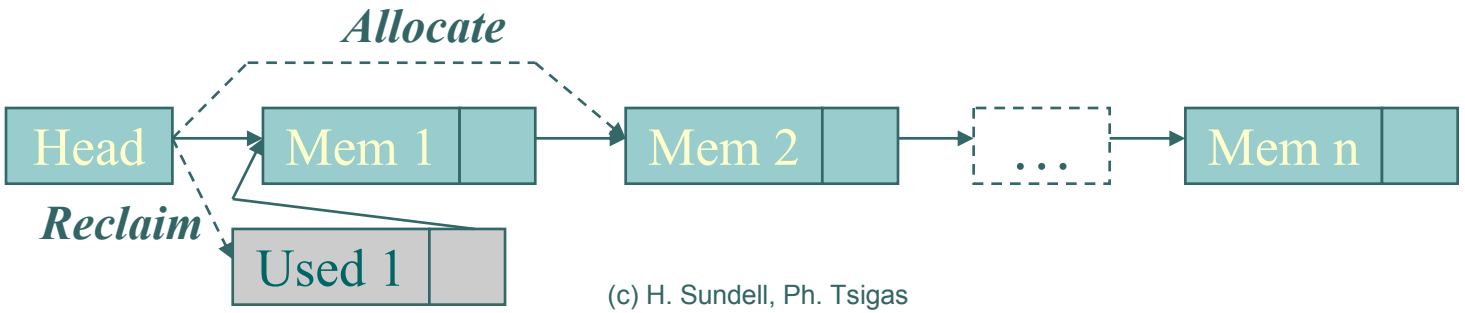
- Example: Insert operation
  - which of 2 or 3 gets inserted?
- Solution: Compare-And-Swap atomic primitive:

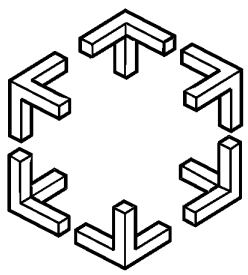
```
CAS(p:pointer to word, old:word,
new:word):boolean
atomic do
        if *p = old then
                *p := new;
                return true;
        else return false;
```



Dynamic Memory Management

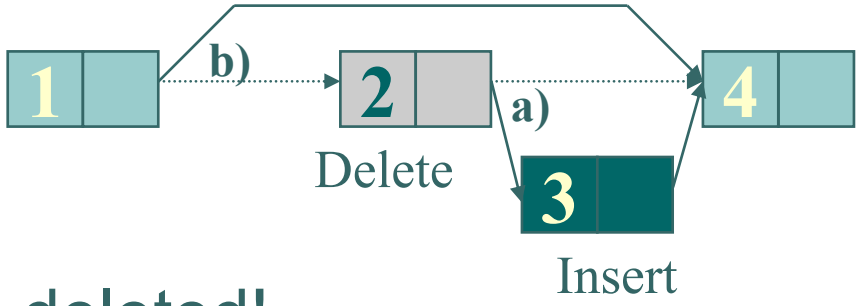
- Problem: System memory allocation functionality is blocking!
- Solution (lock-free), IBM freelists:
  - Pre-allocate a number of nodes, link them into a dynamic stack structure, and allocate/reclaim using CAS





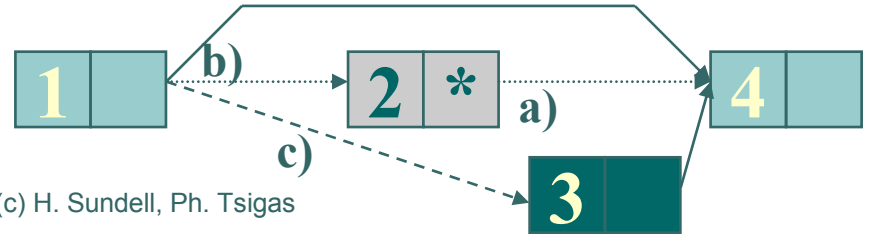
Concurrent Insert vs. Delete operations

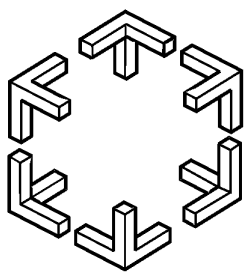
- Problem:



  - both nodes are deleted!

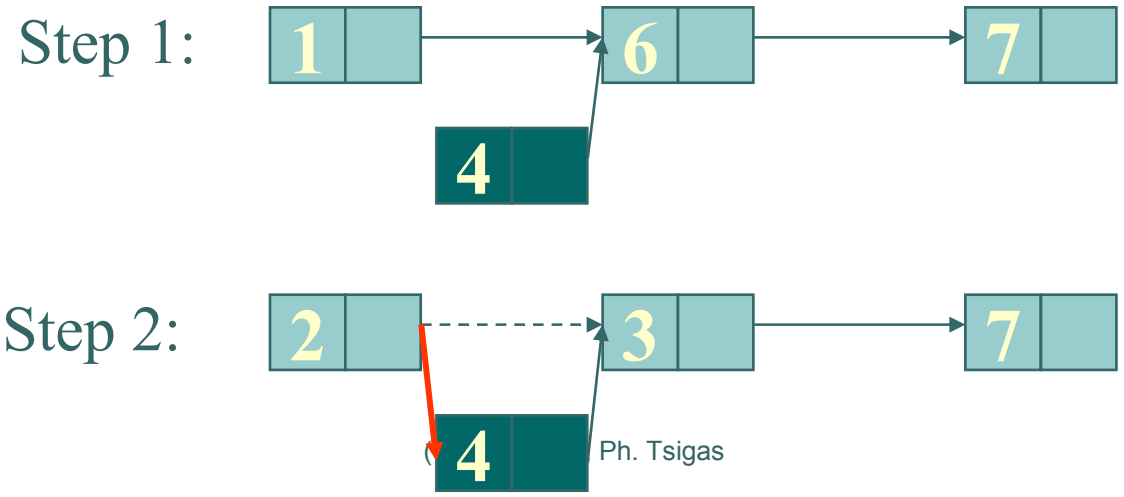
- Solution (Harris et al): Use bit 0 of pointer to mark deletion status

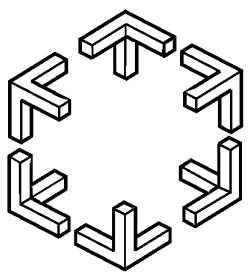




The ABA problem

- Problem: Because of concurrency (pre-emption in particular), same pointer value does not always mean same node (i.e. CAS succeeds)!!!

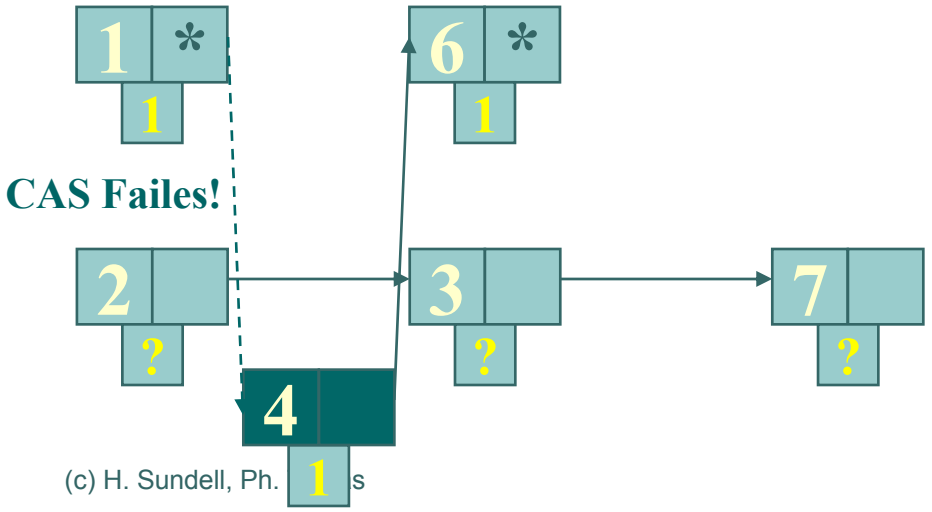


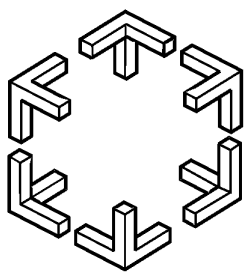


The ABA problem

- Solution: (Valois et al) Add reference counting to each node, in order to prevent nodes that are of interest to some thread to be reclaimed until all threads have left the node

New Step 2:

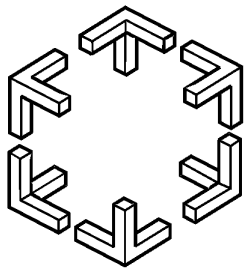




Helping Scheme

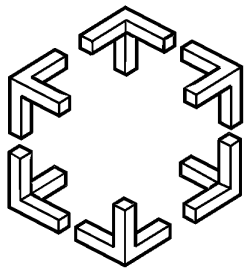
- Threads need to traverse safely

- Need to remove marked-to-be-deleted nodes while traversing – Help!
- Finds previous node, finish deletion and continues traversing from previous node

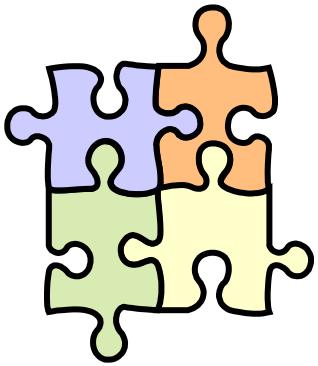


Back-Off Strategy

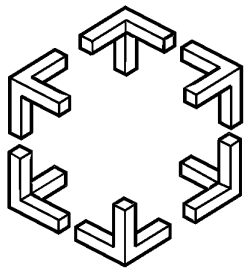
- For pre-emptive systems, helping is necessary for efficiency and lock-freeness
- For really concurrent systems, overlapping CAS operations (caused by helping and others) on the same node can cause heavy contention
- Solution: For every failed CAS attempt, back-off (i.e. sleep) for a certain duration, which increases exponentially



Our Lock-Free Algorithm

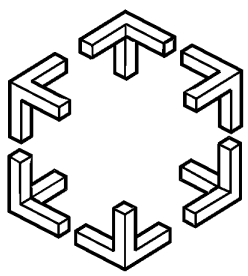


- Based on Skip Lists
  - Treated as layers of ordered lists
- Uses CAS atomic primitive
- Lock-Free memory management
  - IBM Freelists
  - Reference counting
- Helping scheme
- Back-Off strategy
- All together proved to be linearizable

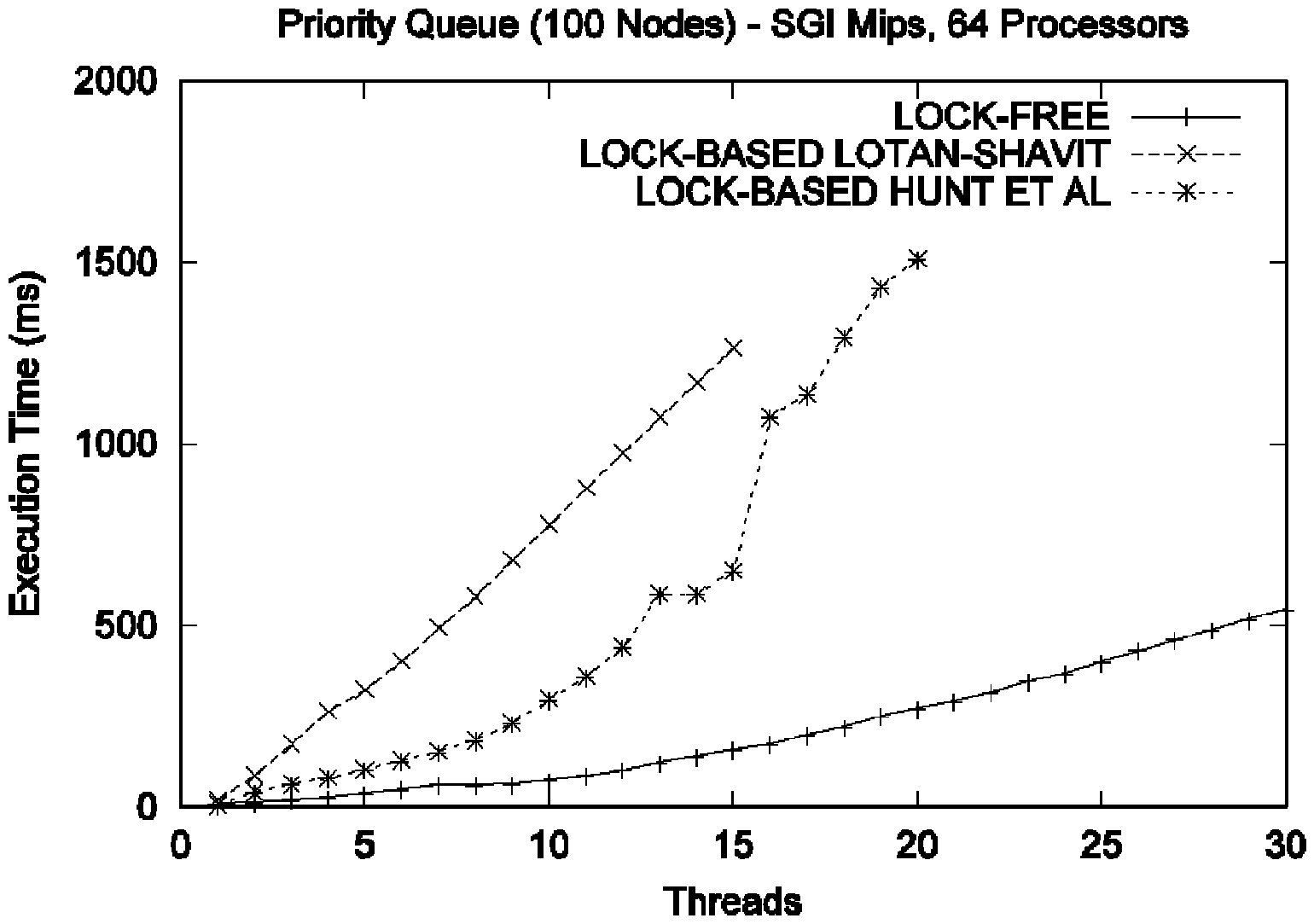


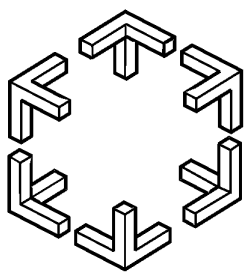
Experiments

- 1-30 threads on platforms with different levels of real concurrency
- 10000 Insert vs. DeleteMin operations by each thread. 100 vs. 1000 initial inserts
- Compare with other implementations:
  - Lotan and Shavit, 2000
  - Hunt et al “An Efficient Algorithm for Concurrent Priority Queue Heaps”, 1996

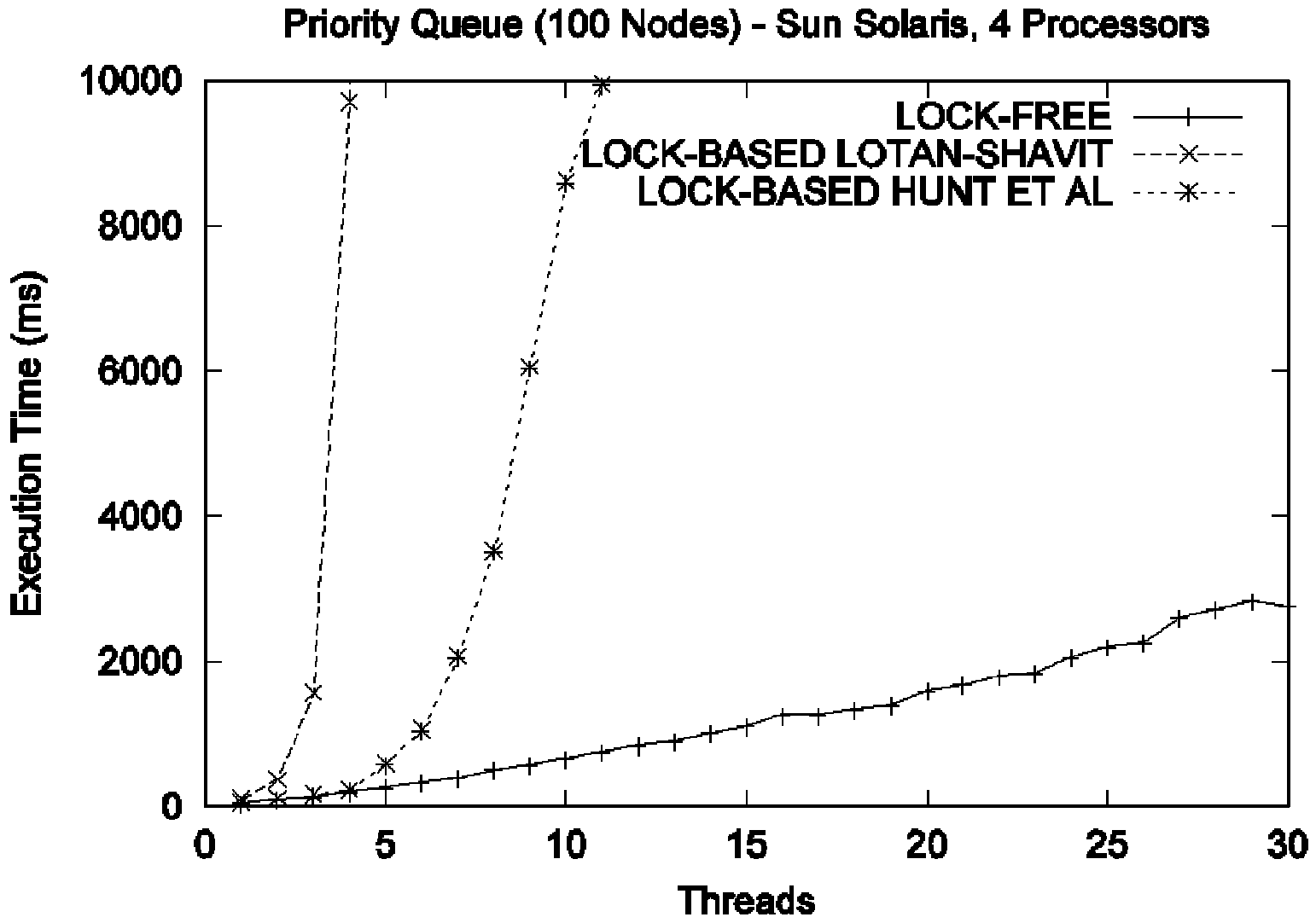


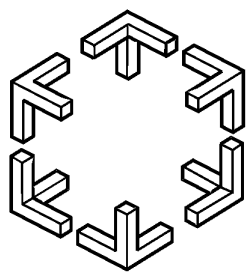
Full Concurrency



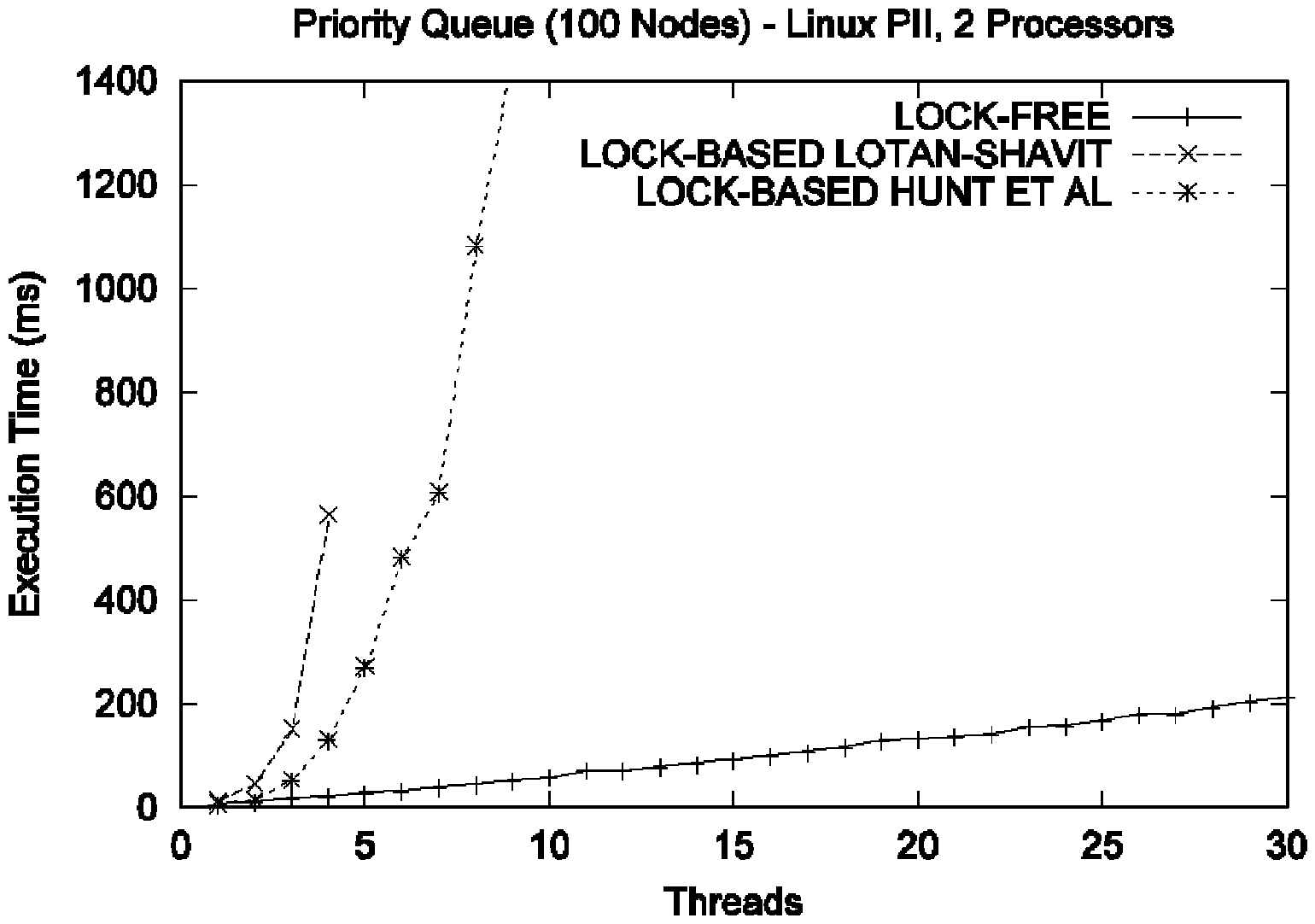


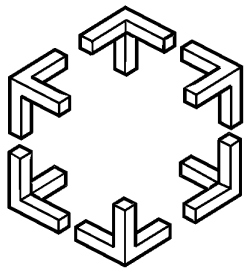
Medium Pre-emption





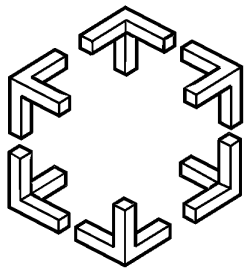
High Pre-emption





Conclusions

- Our work includes a Real-Time extension of the algorithm, using time-stamps and a time-stamp recycling scheme
- Our lock-free algorithm is suitable for both pre-emptive as well as systems with full concurrency
  - Will be available as part of NOBLE software library, http://www.noble-library.org
- See Technical Report for full details, http://www.cs.chalmers.se/~phs



Questions?

- Contact Information:
  - Address:

    Håkan Sundell vs. Philippas Tsigas
    Computing Science
    Chalmers University of Technology
  - Email:

    <phs , tsigas> @ cs.chalmers.se
  - Web:

    http://www.cs.chalmers.se/~phs/warp