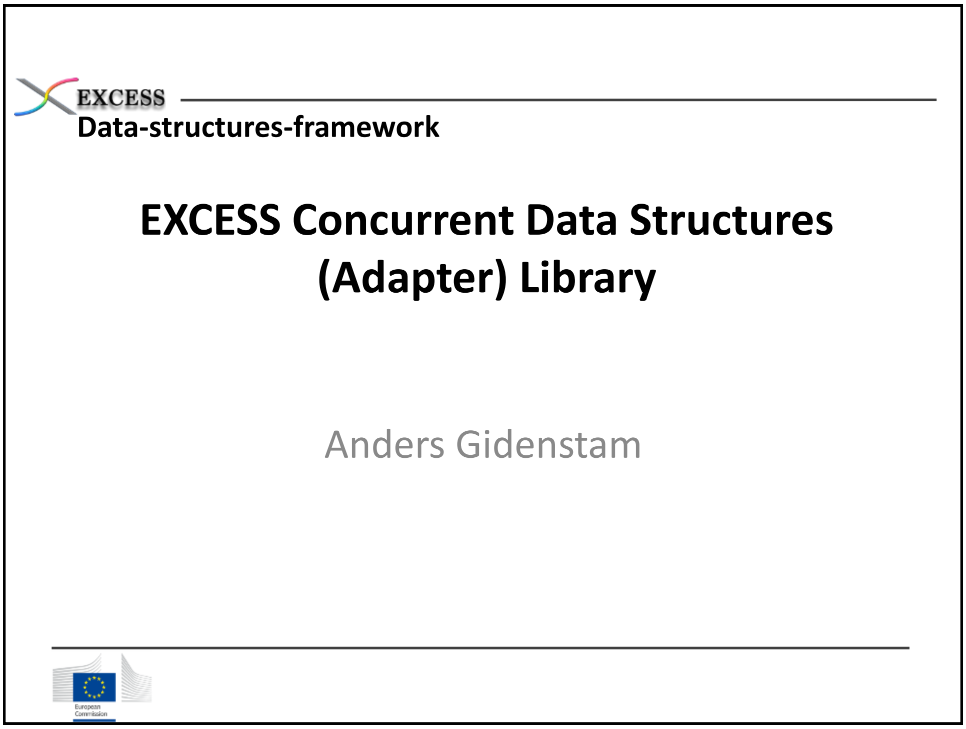
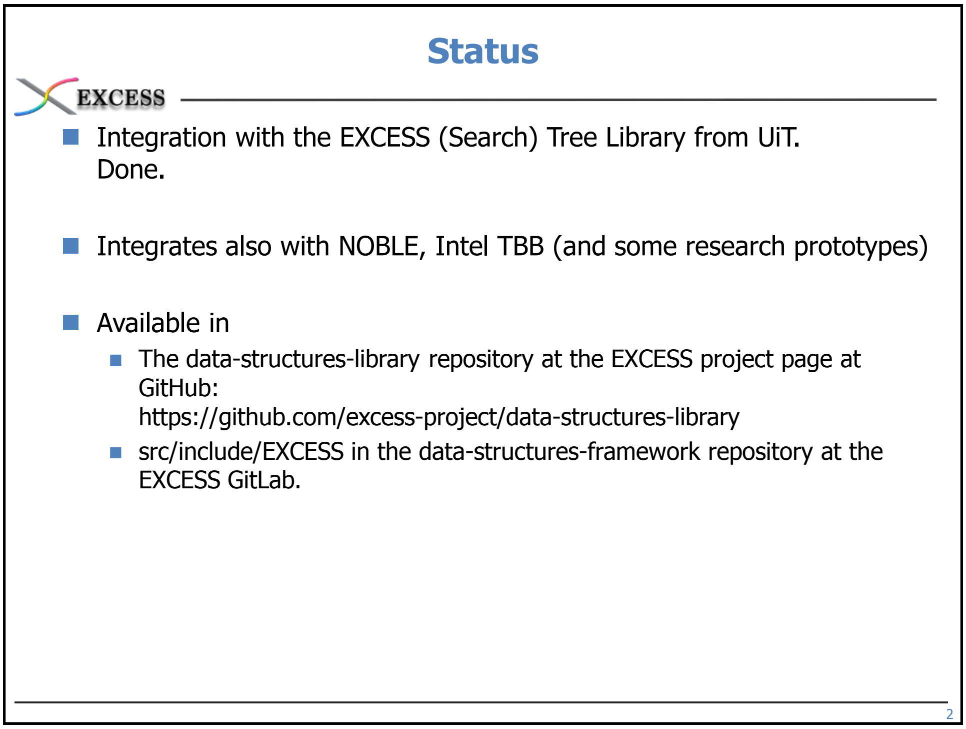


EXCESS
Data-structures-framework

EXCESS Concurrent Data Structures (Adapter) Library

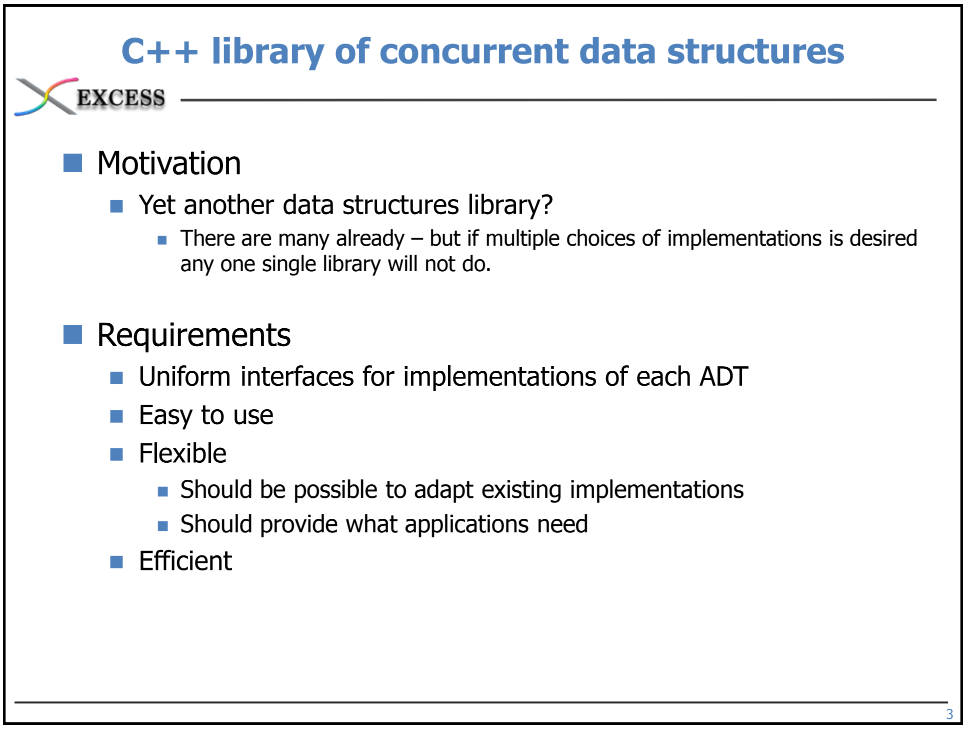
Anders Gidenstam



Status

EXCESS

- Integration with the EXCESS (Search) Tree Library from UiT.
  Done.
- Integrates also with NOBLE, Intel TBB (and some research prototypes)
- Available in
  - The data-structures-library repository at the EXCESS project page at GitHub:
    https://github.com/excess-project/data-structures-library
  - src/include/EXCESS in the data-structures-framework repository at the EXCESS GitLab.

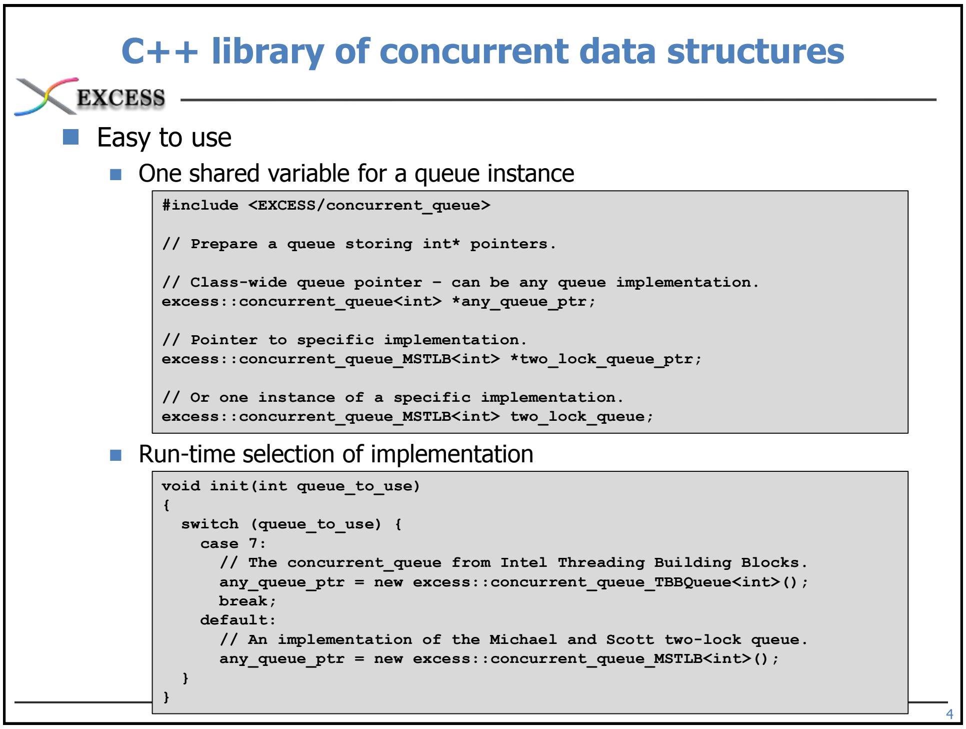


C++ library of concurrent data structures

EXCESS

- Motivation
  - Yet another data structures library?
    - There are many already – but if multiple choices of implementations is desired any one single library will not do.
- Requirements
  - Uniform interfaces for implementations of each ADT
  - Easy to use
  - Flexible
    - Should be possible to adapt existing implementations
    - Should provide what applications need
  - Efficient

3



C++ library of concurrent data structures

EXCESS

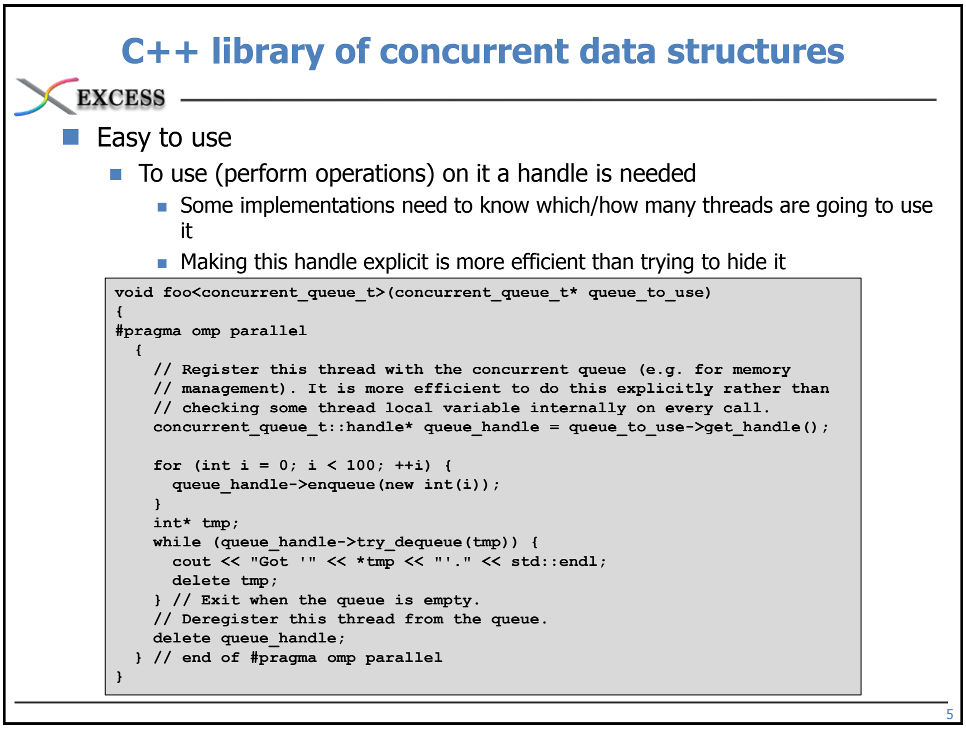
- Easy to use
  - One shared variable for a queue instance

```
#include <EXCESS/concurrent_queue>

// Prepare a queue storing int* pointers.

// Class-wide queue pointer – can be any queue implementation.
excess::concurrent_queue<int> *any_queue_ptr;

// Pointer to specific implementation.
excess::concurrent_queue_MSTLB<int> *two_lock_queue_ptr;

// Or one instance of a specific implementation.
excess::concurrent_queue_MSTLB<int> two_lock_queue;
```

  - Run-time selection of implementation

```
void init(int queue_to_use)
{
  switch (queue_to_use) {
    case 7:
      // The concurrent_queue from Intel Threading Building Blocks.
      any_queue_ptr = new excess::concurrent_queue_TBBQueue<int>();
      break;
    default:
      // An implementation of the Michael and Scott two-lock queue.
      any_queue_ptr = new excess::concurrent_queue_MSTLB<int>();
  }
}
```

4



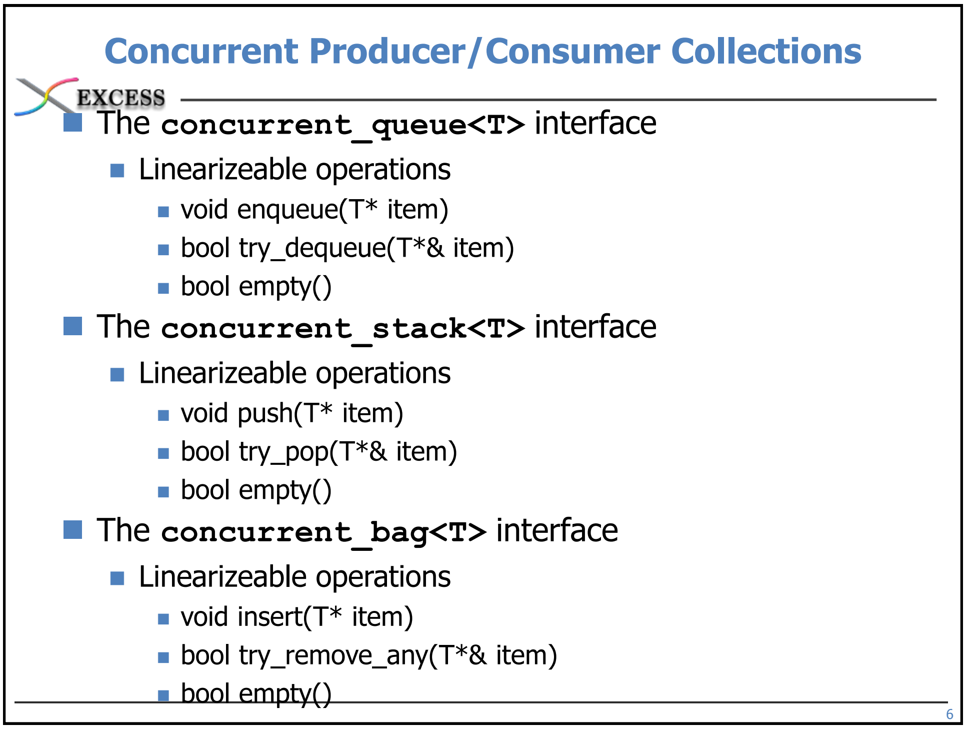
C++ library of concurrent data structures

EXCESS

- Easy to use
  - To use (perform operations) on it a handle is needed
    - Some implementations need to know which/how many threads are going to use it
    - Making this handle explicit is more efficient than trying to hide it

```
void foo<concurrent_queue_t>(concurrent_queue_t* queue_to_use)
{
#pragma omp parallel
  {
    // Register this thread with the concurrent queue (e.g. for memory
    // management). It is more efficient to do this explicitly rather than
    // checking some thread local variable internally on every call.
    concurrent_queue_t::handle* queue_handle = queue_to_use->get_handle();

    for (int i = 0; i < 100; ++i) {
      queue_handle->enqueue(new int(i));
    }
    int* tmp;
    while (queue_handle->try_dequeue(tmp)) {
      cout << "Got '" << *tmp << "'." << std::endl;
      delete tmp;
    } // Exit when the queue is empty.
    // Deregister this thread from the queue.
    delete queue_handle;
  } // end of #pragma omp parallel
}
```

5

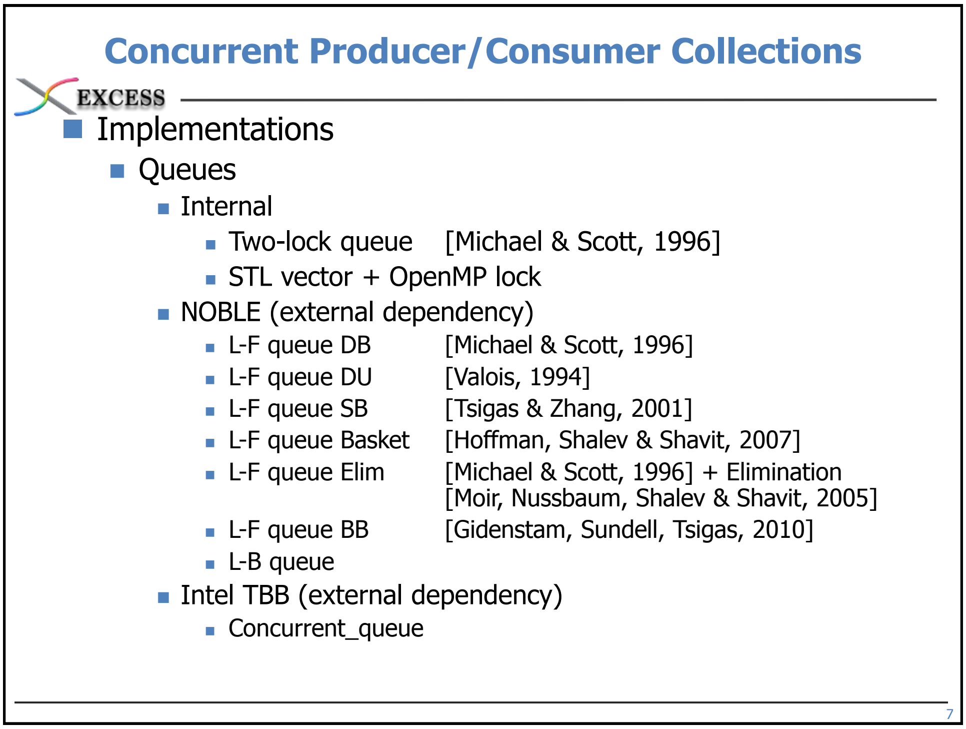


Concurrent Producer/Consumer Collections

EXCESS

- The `concurrent_queue<T>` interface
  - Linearizeable operations
    - void enqueue(T* item)
    - bool try_dequeue(T*& item)
    - bool empty()
- The `concurrent_stack<T>` interface
  - Linearizeable operations
    - void push(T* item)
    - bool try_pop(T*& item)
    - bool empty()
- The `concurrent_bag<T>` interface
  - Linearizeable operations
    - void insert(T* item)
    - bool try_remove_any(T*& item)
    - bool empty()

6

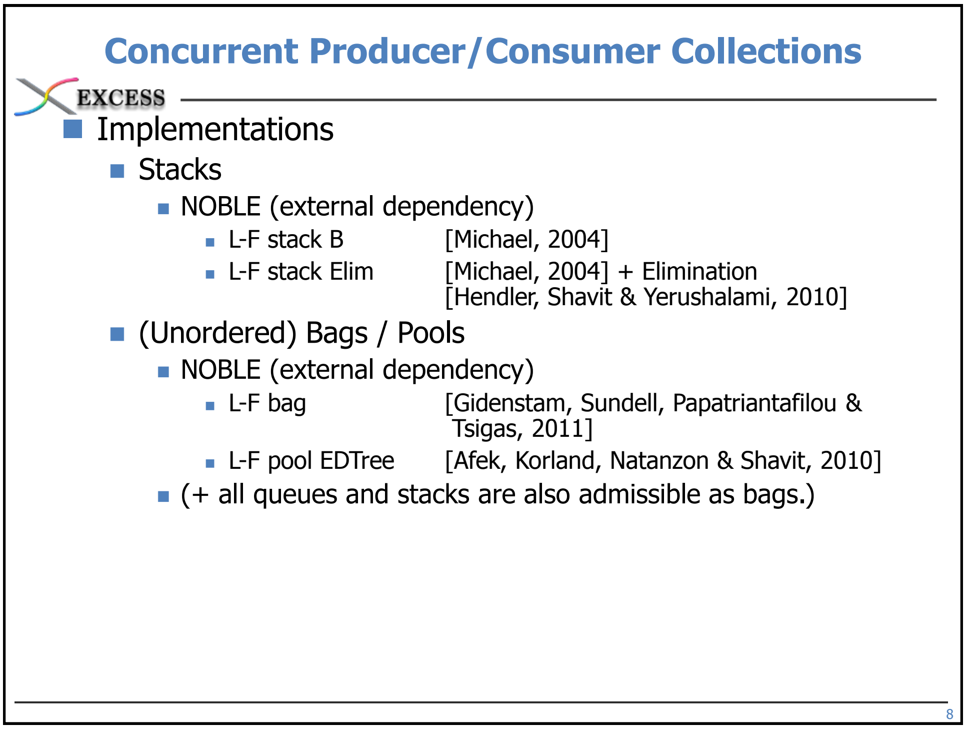


Concurrent Producer/Consumer Collections

EXCESS

- Implementations
  - Queues
    - Internal
      - Two-lock queue [Michael & Scott, 1996]
      - STL vector + OpenMP lock
    - NOBLE (external dependency)
      - L-F queue DB [Michael & Scott, 1996]
      - L-F queue DU [Valois, 1994]
      - L-F queue SB [Tsigas & Zhang, 2001]
      - L-F queue Basket [Hoffman, Shalev & Shavit, 2007]
      - L-F queue Elim [Michael & Scott, 1996] + Elimination [Moir, Nussbaum, Shalev & Shavit, 2005]
      - L-F queue BB [Gidenstam, Sundell, Tsigas, 2010]
      - L-B queue
    - Intel TBB (external dependency)
      - Concurrent_queue

7

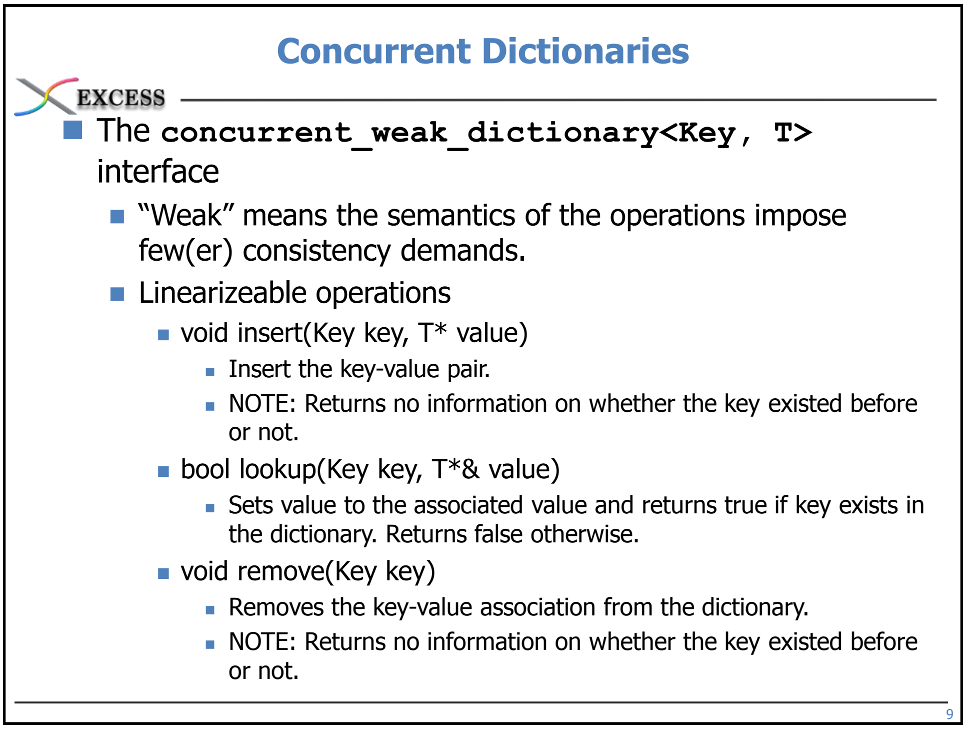


Concurrent Producer/Consumer Collections

EXCESS

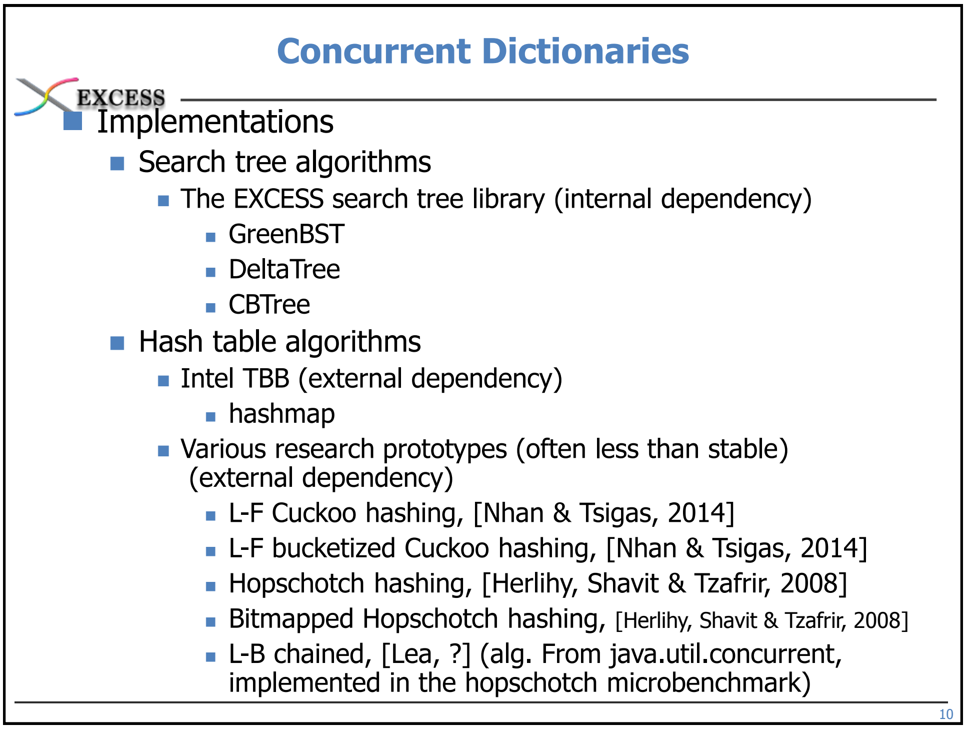
- Implementations
  - Stacks
    - NOBLE (external dependency)
      - L-F stack B [Michael, 2004]
      - L-F stack Elim [Michael, 2004] + Elimination [Hendler, Shavit & Yerushalami, 2010]
  - (Unordered) Bags / Pools
    - NOBLE (external dependency)
      - L-F bag [Gidenstam, Sundell, Papatriantafilou & Tsigas, 2011]
      - L-F pool EDTree [Afek, Korland, Natanzon & Shavit, 2010]
    - (+ all queues and stacks are also admissible as bags.)

8



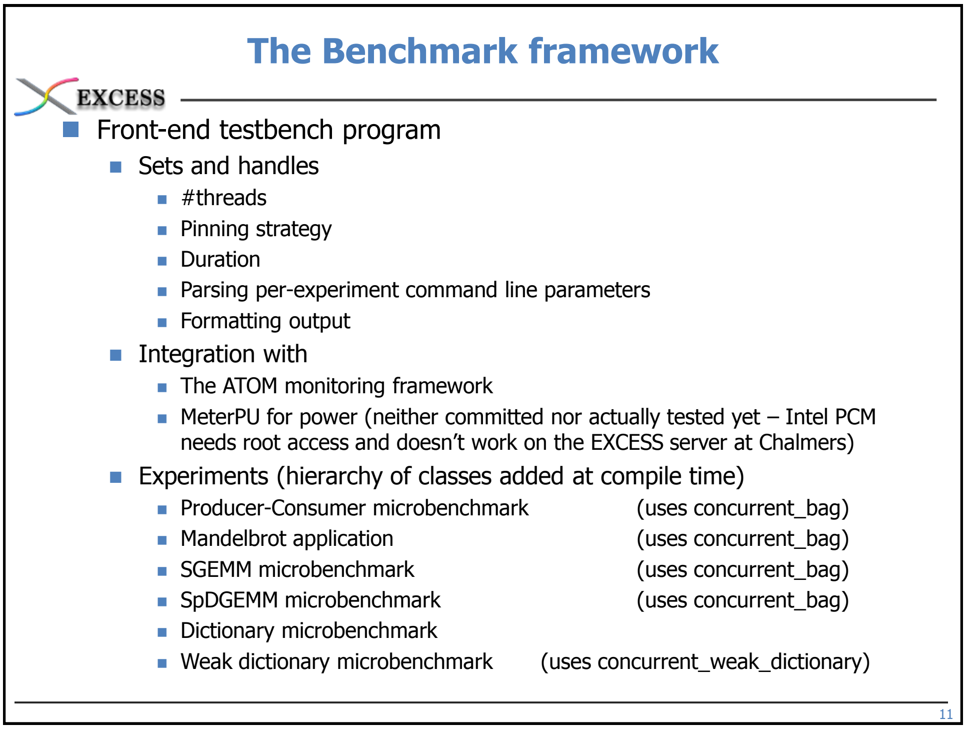
Concurrent Dictionaries

- The `concurrent_weak_dictionary<Key, T>` interface
  - "Weak" means the semantics of the operations impose few(er) consistency demands.
  - Linearizeable operations
    - void insert(Key key, T* value)
      - Insert the key-value pair.
      - NOTE: Returns no information on whether the key existed before or not.
    - bool lookup(Key key, T*& value)
      - Sets value to the associated value and returns true if key exists in the dictionary. Returns false otherwise.
    - void remove(Key key)
      - Removes the key-value association from the dictionary.
      - NOTE: Returns no information on whether the key existed before or not.



Concurrent Dictionaries

- Implementations
  - Search tree algorithms
    - The EXCESS search tree library (internal dependency)
      - GreenBST
      - DeltaTree
      - CBTree
  - Hash table algorithms
    - Intel TBB (external dependency)
      - hashmap
    - Various research prototypes (often less than stable) (external dependency)
      - L-F Cuckoo hashing, [Nhan & Tsigas, 2014]
      - L-F bucketized Cuckoo hashing, [Nhan & Tsigas, 2014]
      - Hopschotch hashing, [Herlihy, Shavit & Tzafrir, 2008]
      - Bitmapped Hopschotch hashing, [Herlihy, Shavit & Tzafrir, 2008]
      - L-B chained, [Lea, ?] (alg. From java.util.concurrent, implemented in the hopschotch microbenchmark)



The Benchmark framework

EXCESS

- Front-end testbench program
  - Sets and handles
    - #threads
    - Pinning strategy
    - Duration
    - Parsing per-experiment command line parameters
    - Formatting output
  - Integration with
    - The ATOM monitoring framework
    - MeterPU for power (neither committed nor actually tested yet – Intel PCM needs root access and doesn't work on the EXCESS server at Chalmers)
  - Experiments (hierarchy of classes added at compile time)
    - Producer-Consumer microbenchmark (uses concurrent_bag)
    - Mandelbrot application (uses concurrent_bag)
    - SGEMM microbenchmark (uses concurrent_bag)
    - SpDGEMM microbenchmark (uses concurrent_bag)
    - Dictionary microbenchmark
    - Weak dictionary microbenchmark (uses concurrent_weak_dictionary)

11