

5th STL workshop, June 2005

Hashtables based on linear hashing — Design

Jørgen Havsberg Seland

Topics

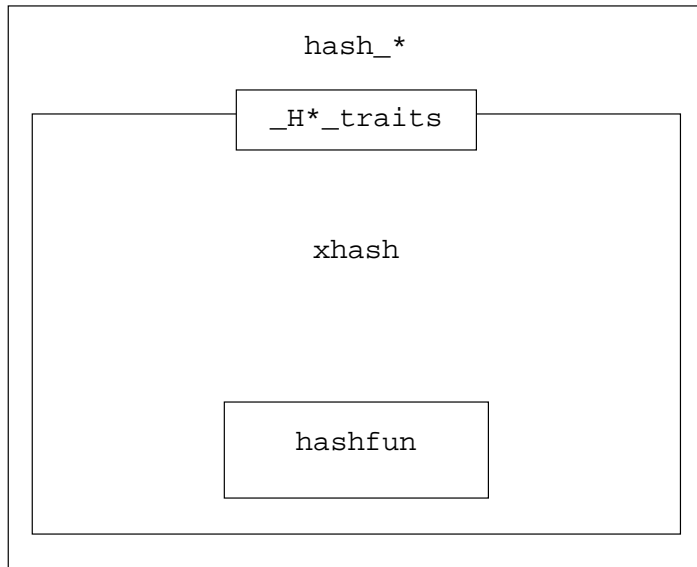
1. Modules
2. External interfaces
3. Internal interfaces
4. Implementations
5. Operations: expansion / contraction, bucket traversal, table traversal

Primary ingredients

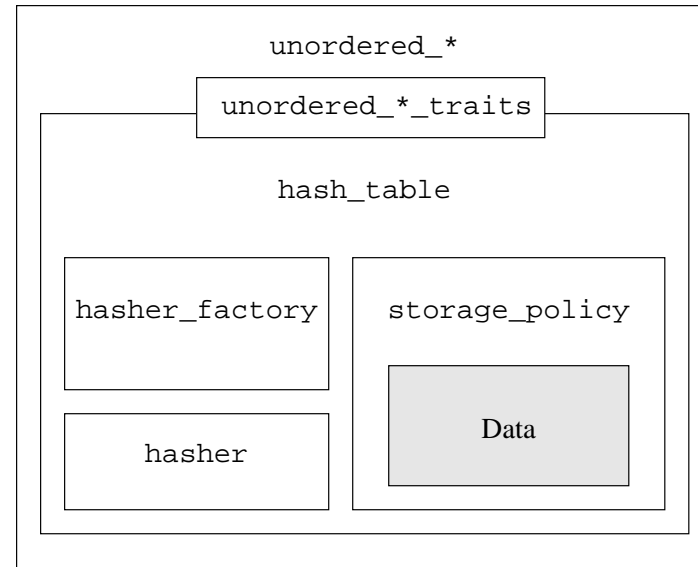
1. Linear hashing
2. Universal classes of hash functions
3. Modular design

The Modules

Dinkumware

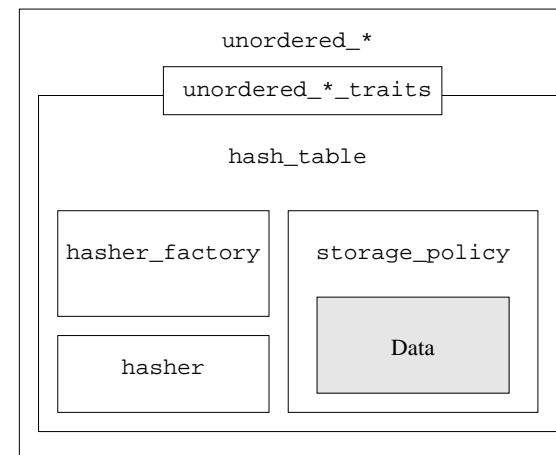


CPHSTL



Pros/cons of a modular design

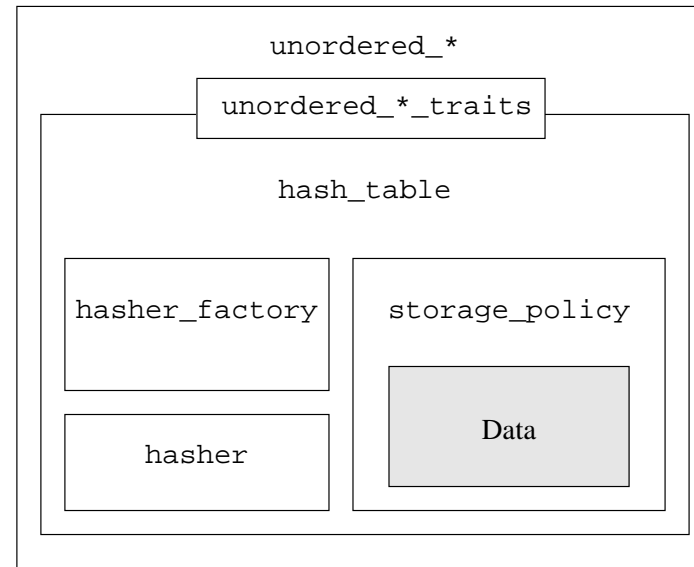
- ÷ Layers create dependency on compiler
- + Extensions are plug'n'play
- + We won't need a Silver Bullet



- `unordered_map`, `unordered_set`, `unordered_multimap`, `unordered_multiset`.
- Close to the proposal of Matthew Austern (of the C++ standard working group), with the following exceptions:
 - Bucket interface made optional
 - Universal hashing support added
 - `min_load_factor`

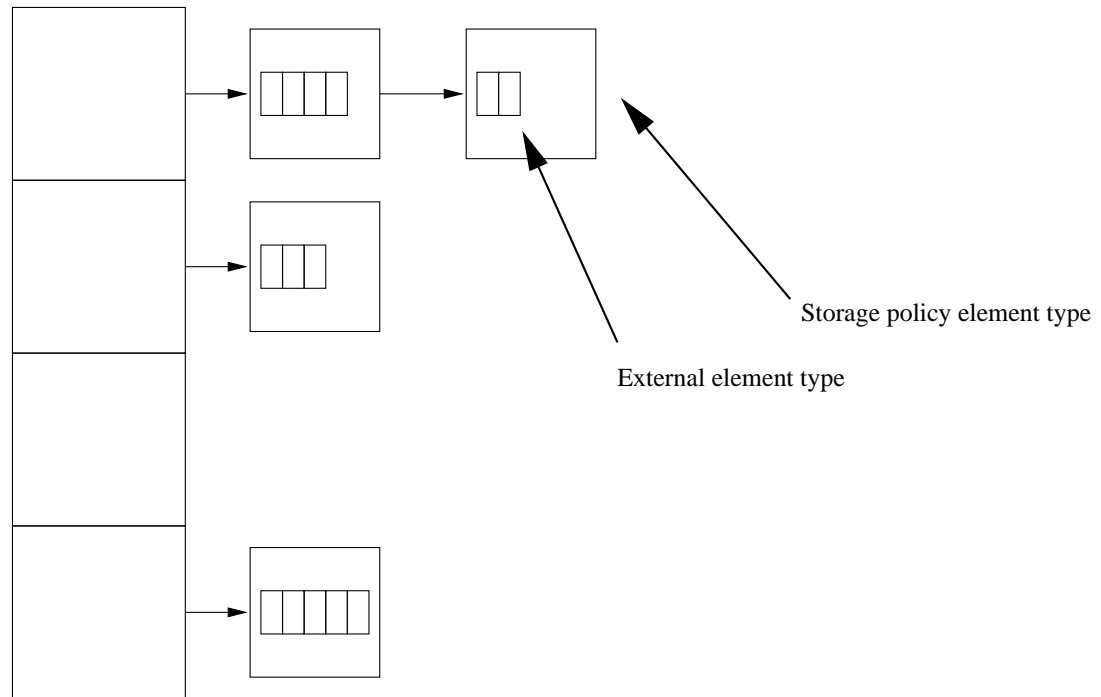
The Glue

1. `hash_table`
2. `value_type` and `internal_value_type`



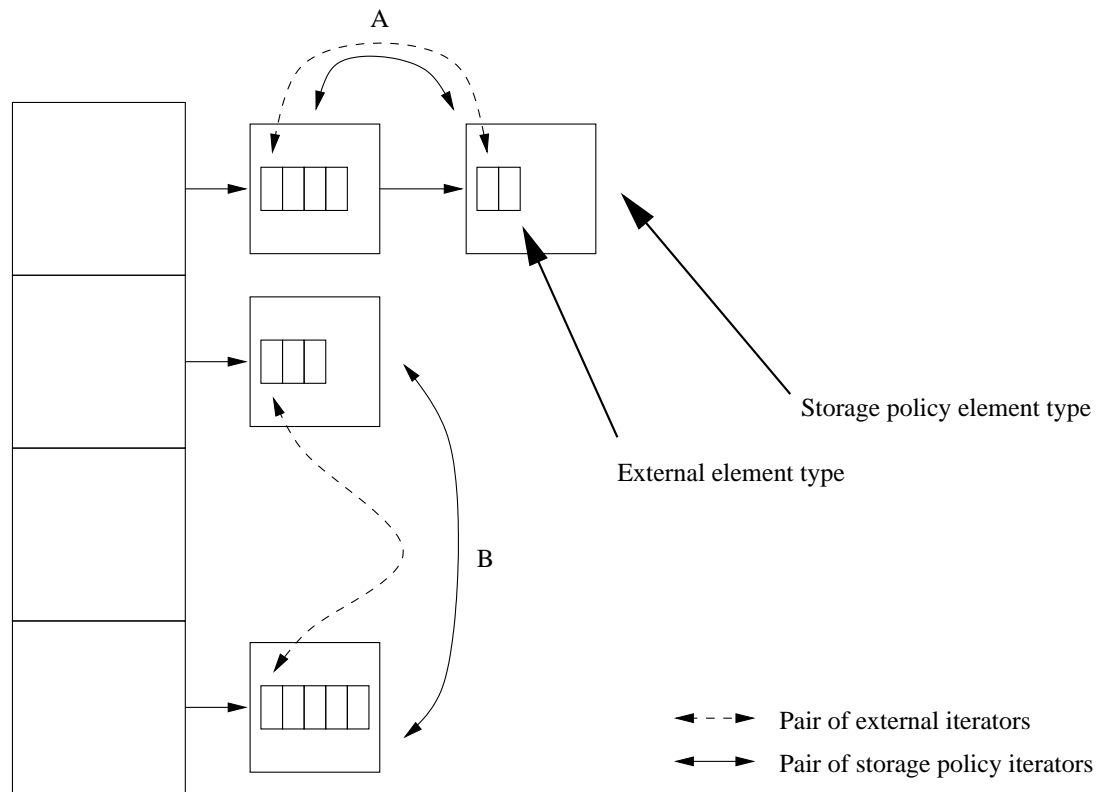
`unordered_multimap` **and** `unordered_multiset` **at no extra cost.**

1. `internal_value_type`
as a list
2. `twolevel_iterator`

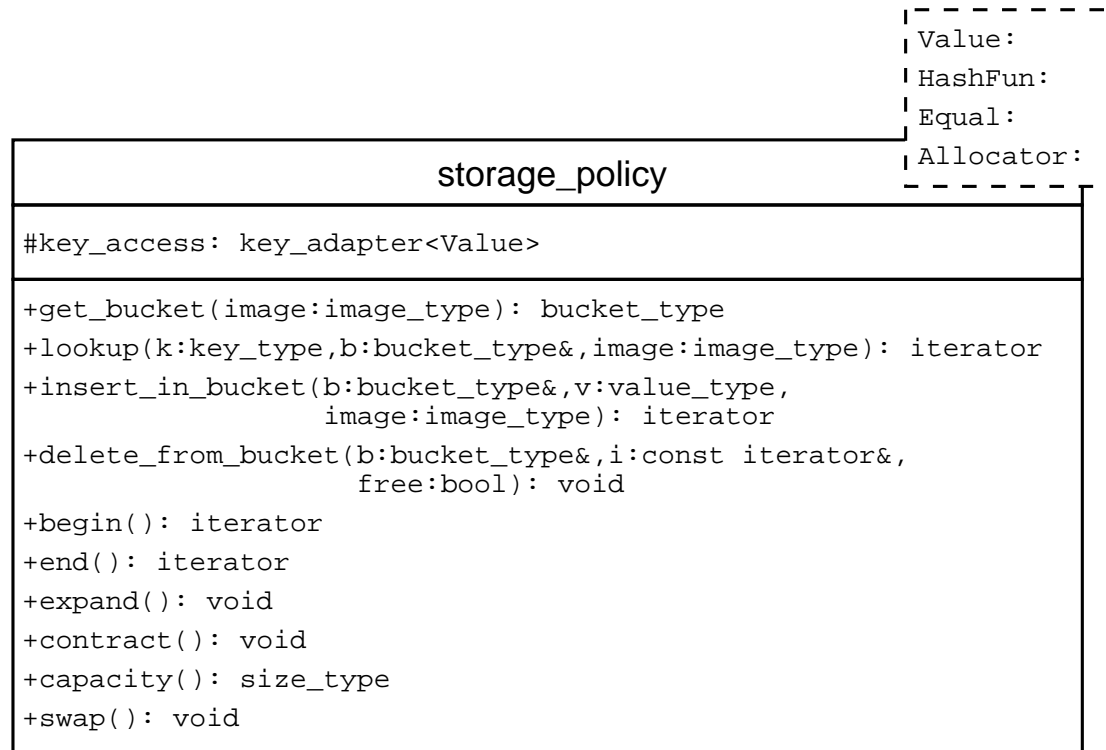


`unordered_multimap` **and** `unordered_multiset` **at no extra cost.**

1. `internal_value_type`
as a list
2. `twolevel_iterator`



The Storage Policy



Aim: Reduced number of methods without reduction in performance

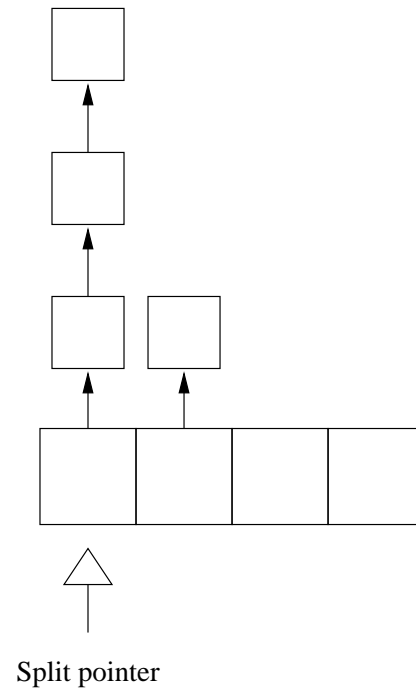
The tables

One-table linear hashing Closely following Per-Åke Larson's paper

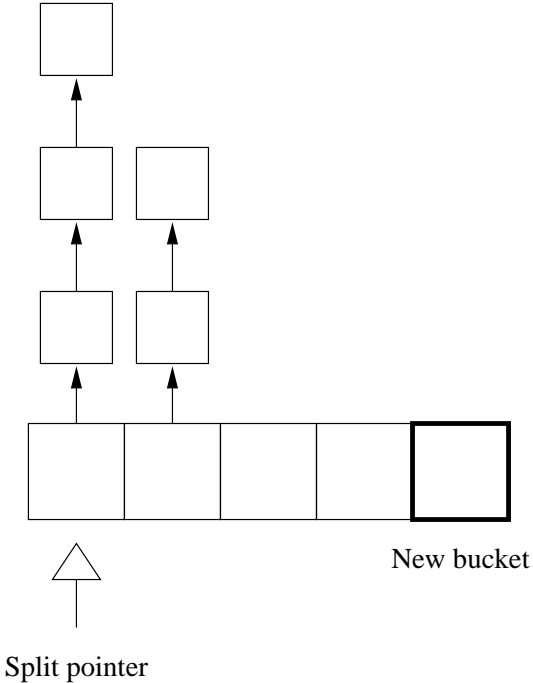
Two-table linear hashing Linear hashing / gradual expansion hybrid

Chained hashing An (almost) regular chained hash table.

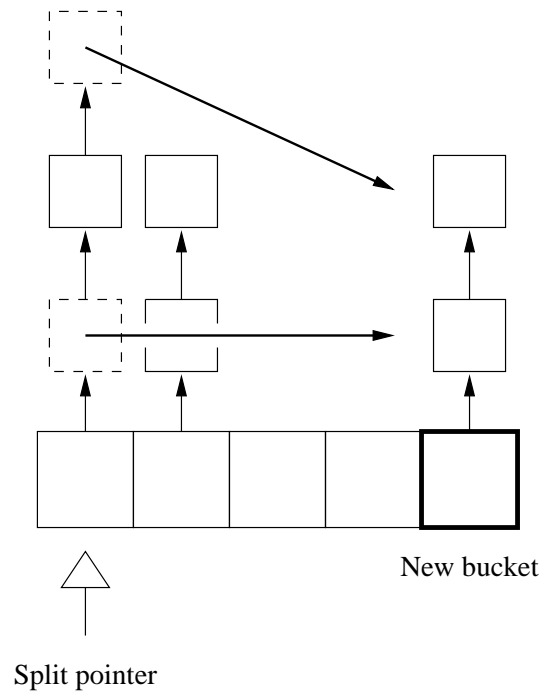
Before the split. α has passed α_{max} .



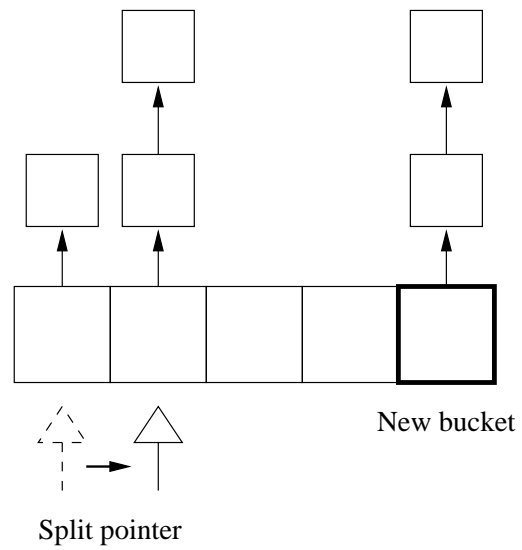
We establish a new bucket. α is again below or equal to α_{max} .



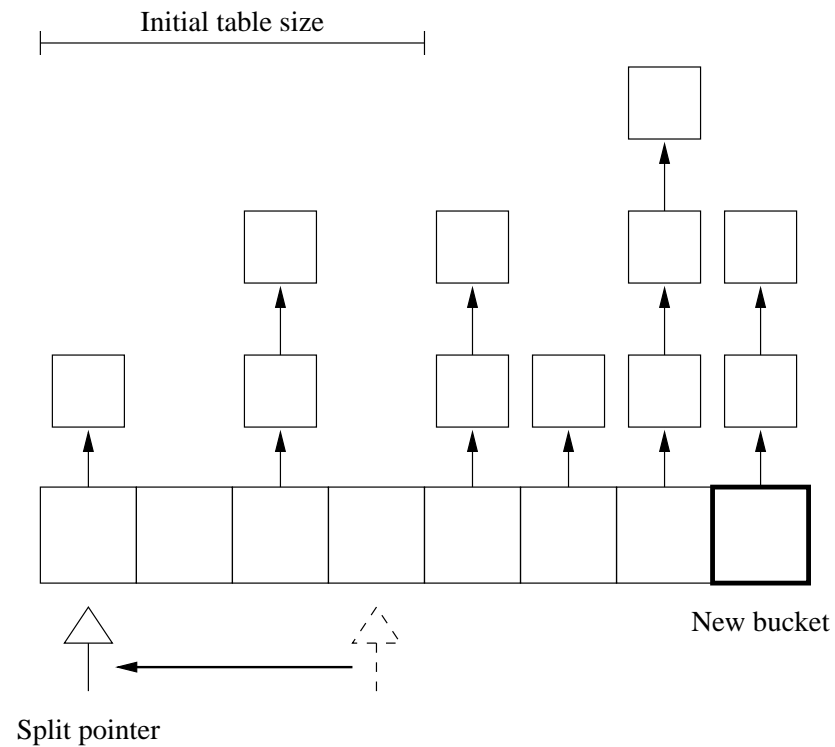
We pick a suitable new hash function, and the bucket is split.



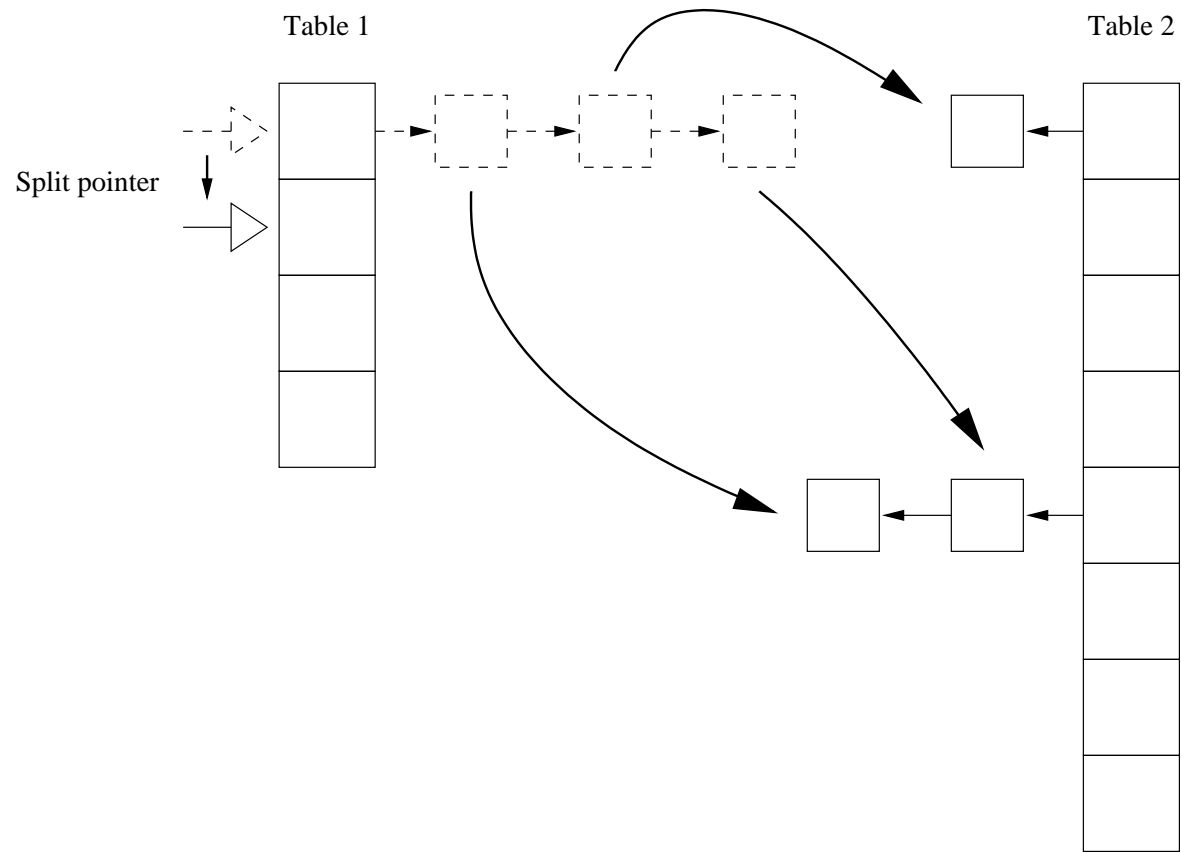
Finally, we update the *split pointer*.



All the buckets in the original table (size 4) have been split. We reset the pointer and continue this time to 8.



The Two-Table Variation



Operations: Expansion / Contraction: Points of interest

1. `expand()` and `contract()` vs. `resize(n)`
2. Linear hashing: Importance of *storing the hash value*.
3. One-table: The SED tree and friends.
4. Two-table: Function Switching
5. Chained: Copy minimization

Operations: Bucket Traversal / Searching: Optimizations

1. Rabin-Karp: Importance of *storing the hash value*.
2. Chained Hash: The Tight Inner Loop

Operations: Table Traversal / Iteration: The Methods

1. One- and two-table: Traversing table chains
2. Chained: Traversing bucket array
3. Dinkumware: The “Hybrid” (not implemented)