A Study of Erlang ETS Table Implementations and Performance

Or: Judy Arrays Are Amazing Data Structures

Scott Lystig Fritchie

<slfritchie@snookles.com>

Snookles Music Consulting

Overview

- ETS table data structures
- Judy arrays
- “Contiguous Key Problem”
- Solving the “Contiguous Key Problem”
- Performance results

Audience

- Erlang community
  - Using ETS directly
  - Using ETS indirectly via Mnesia and other OTP applications
- C/C++ developers using hash tables and balanced trees
  - Performance gains by using “Judy arrays” can be impressive
  - Consider using Judy arrays in your applications

ETS Table Implementations

- Types included in Erlang/OTP:
  - AVL balanced binary tree: `ordered_set`
  - Resizable linear hash table: `set, bag, duplicate_bag`
- New research types:
  - In-memory B-tree: `btree`
  - Judy arrays (based on tries): `judysl, judyesl, judyeh`

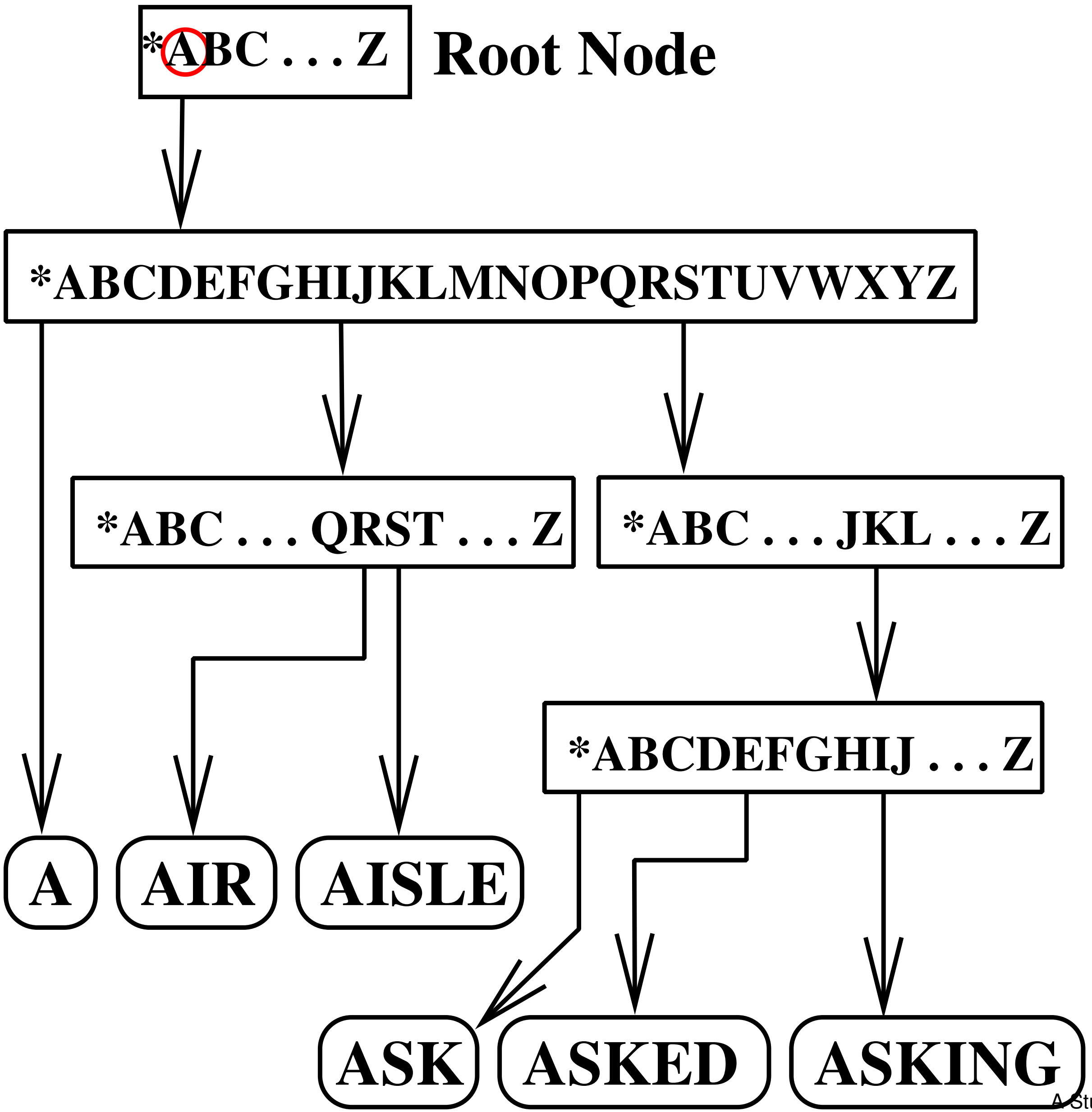
Judy Arrays

- Invented by Doug Baskins, implemented by Hewlett-Packard.
  - Named after Baskins's sister.
- Source code now available under GNU LGPL license.
- Source & docs at `http://judy.sourceforge.net/`

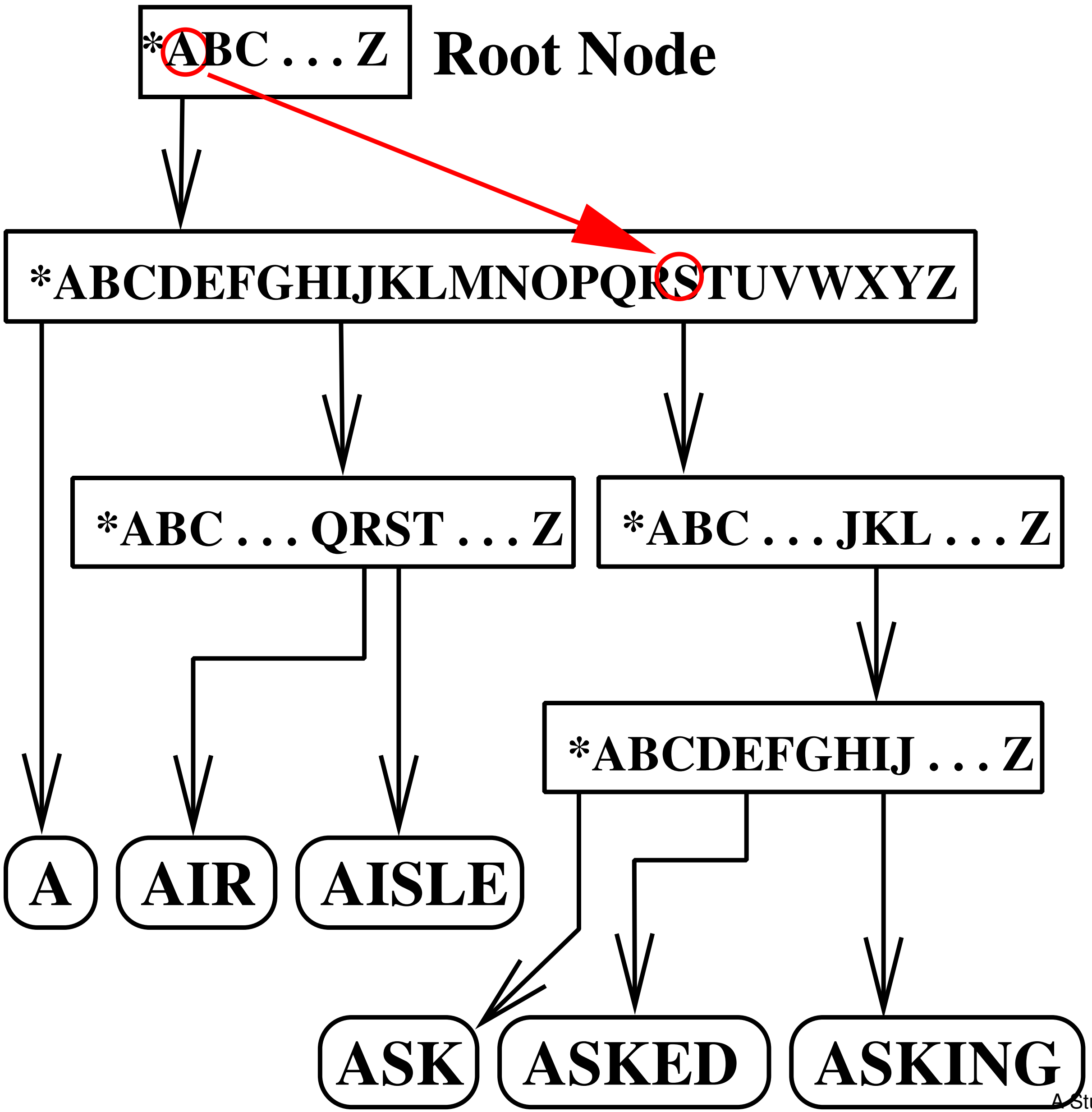
Judy Arrays (continued)

- Judy arrays are dynamic arrays
  - Index = 1 word, 32- or 64-bit
  - Value = 1 bit or 1 word
- Handles small & large populations, sparse & dense populations, *no tuning parameters!*
- Implemented as a logical 256-ary trie

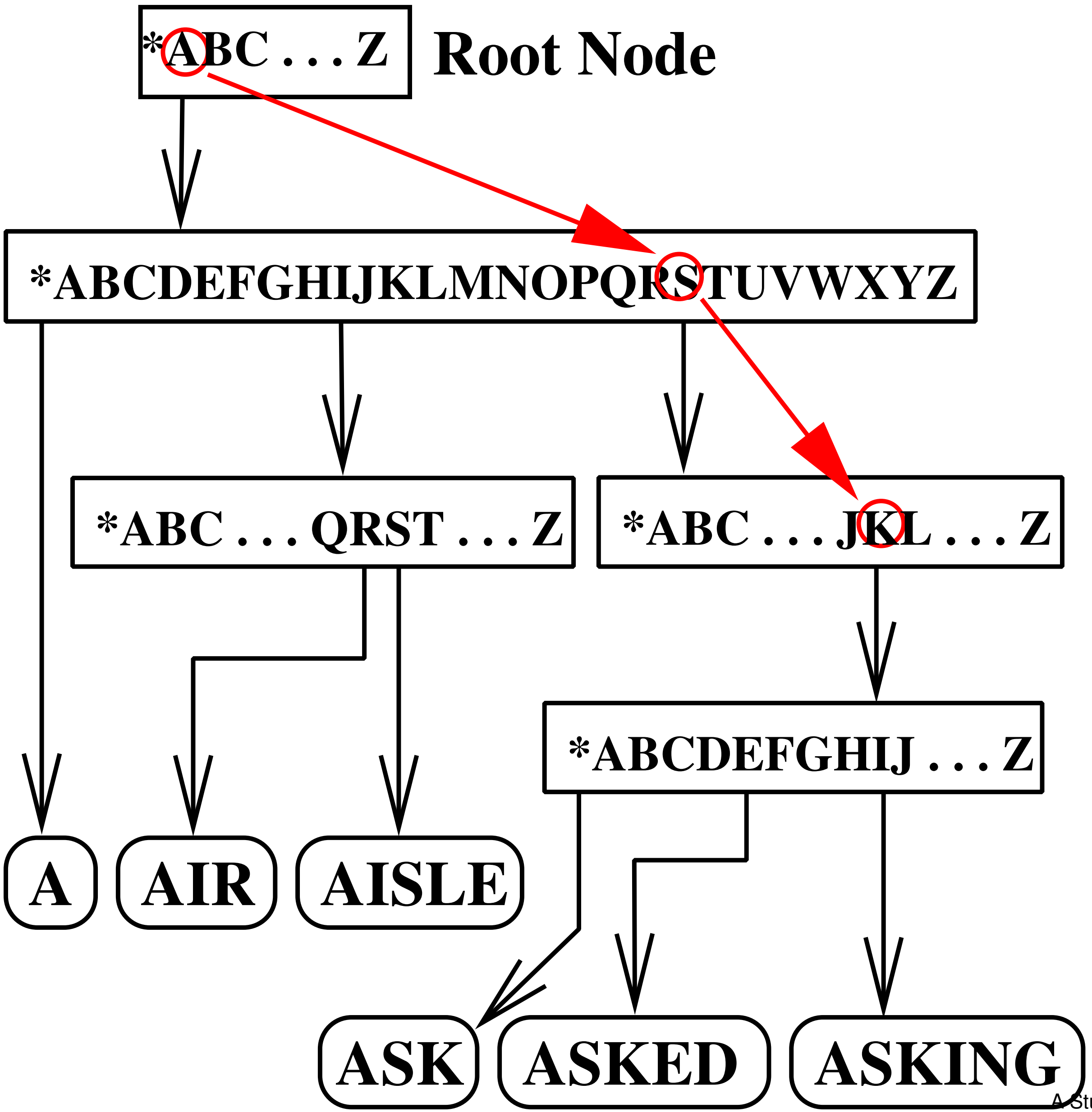
Data Structures Review: The Trie



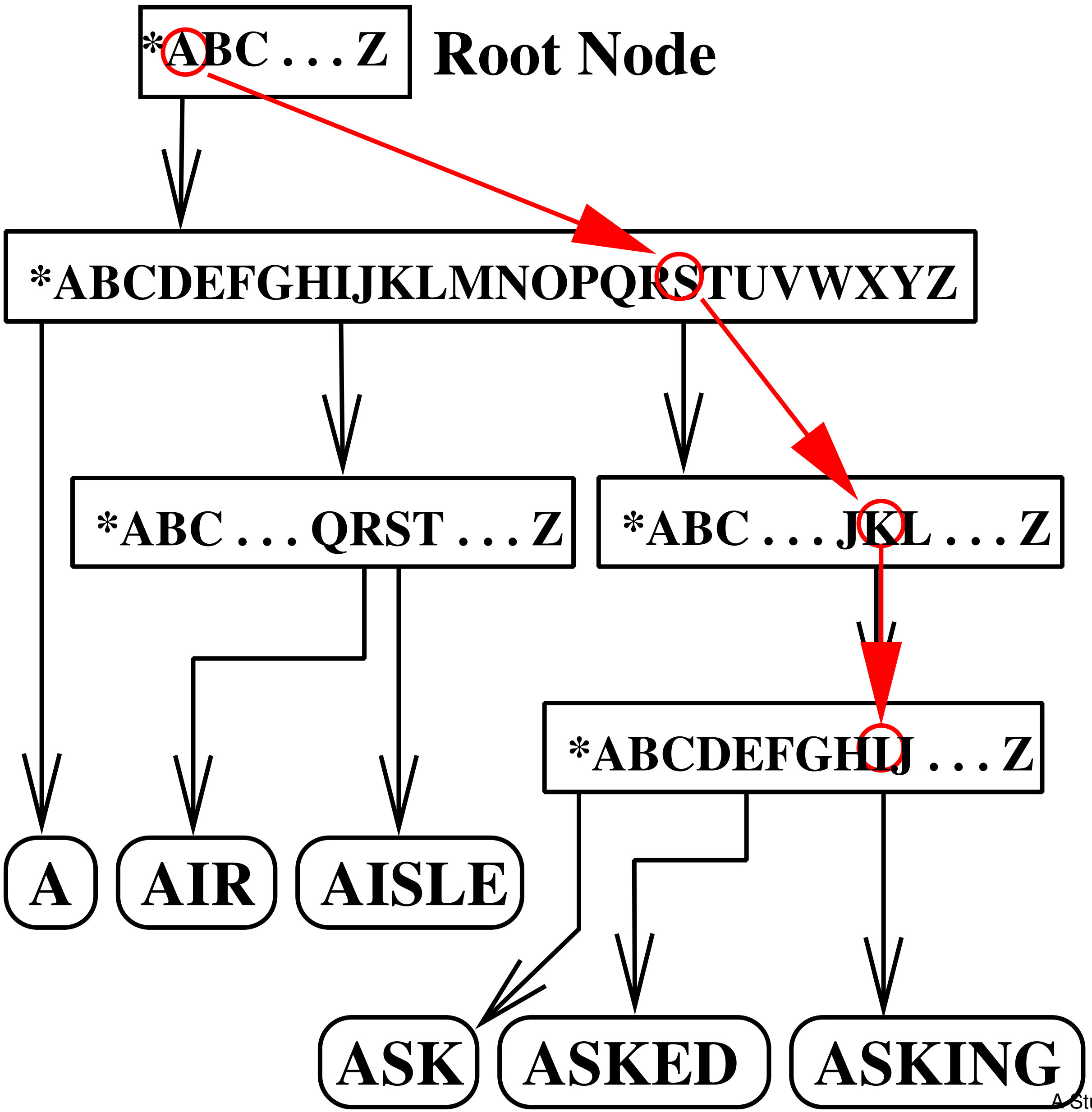
Data Structures Review: The Trie



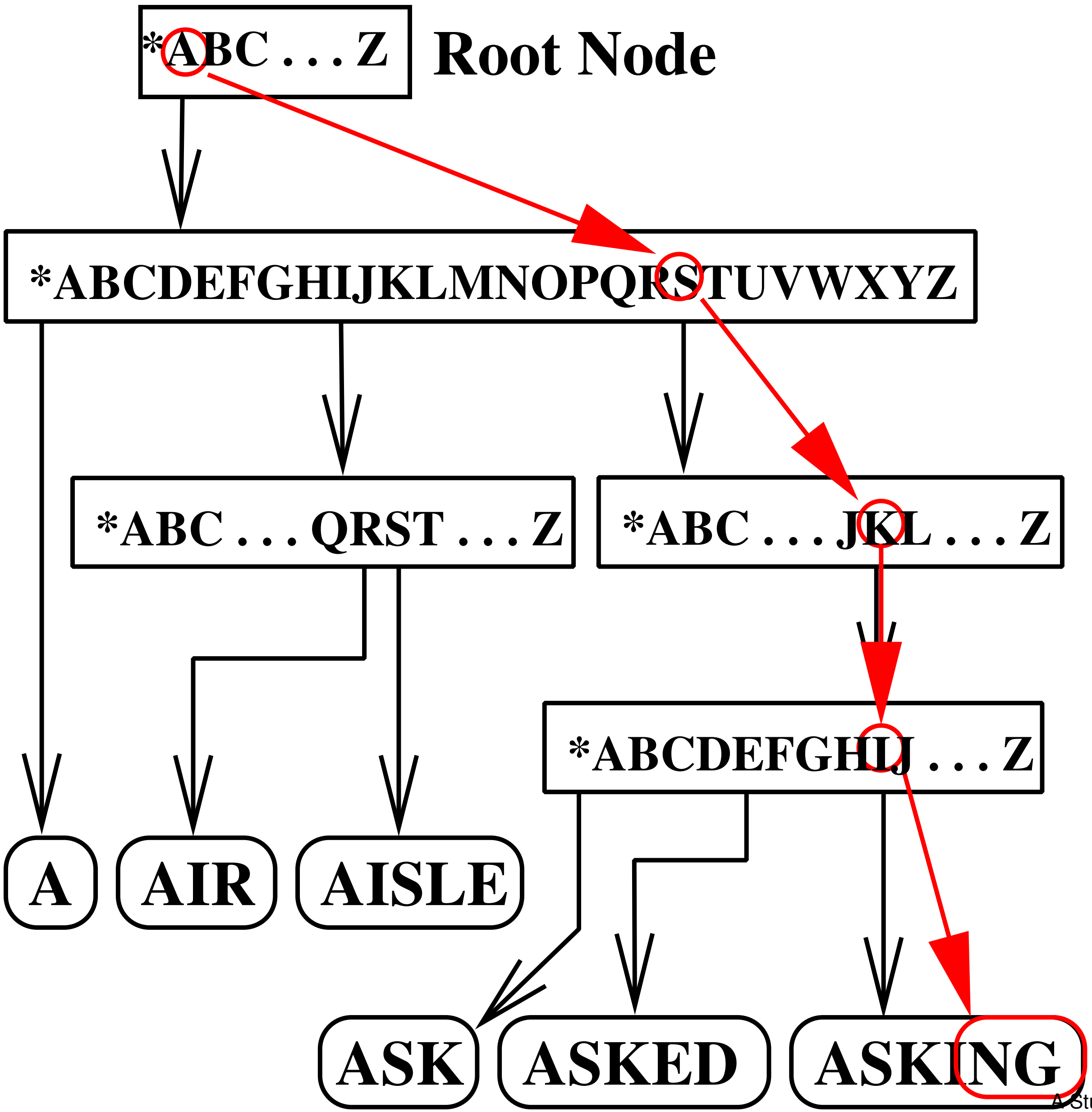
Data Structures Review: The Trie



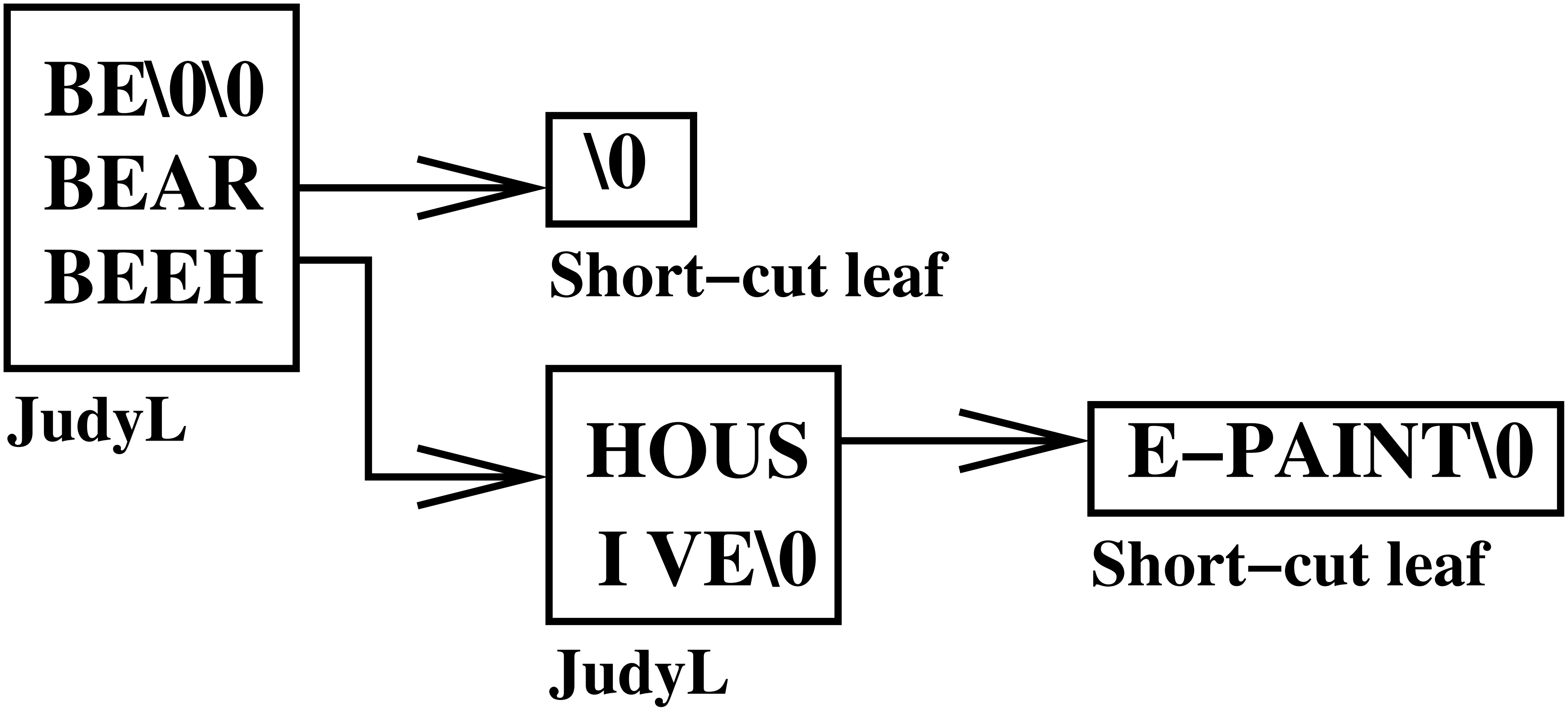
Data Structures Review: The Trie



Data Structures Review: The Trie

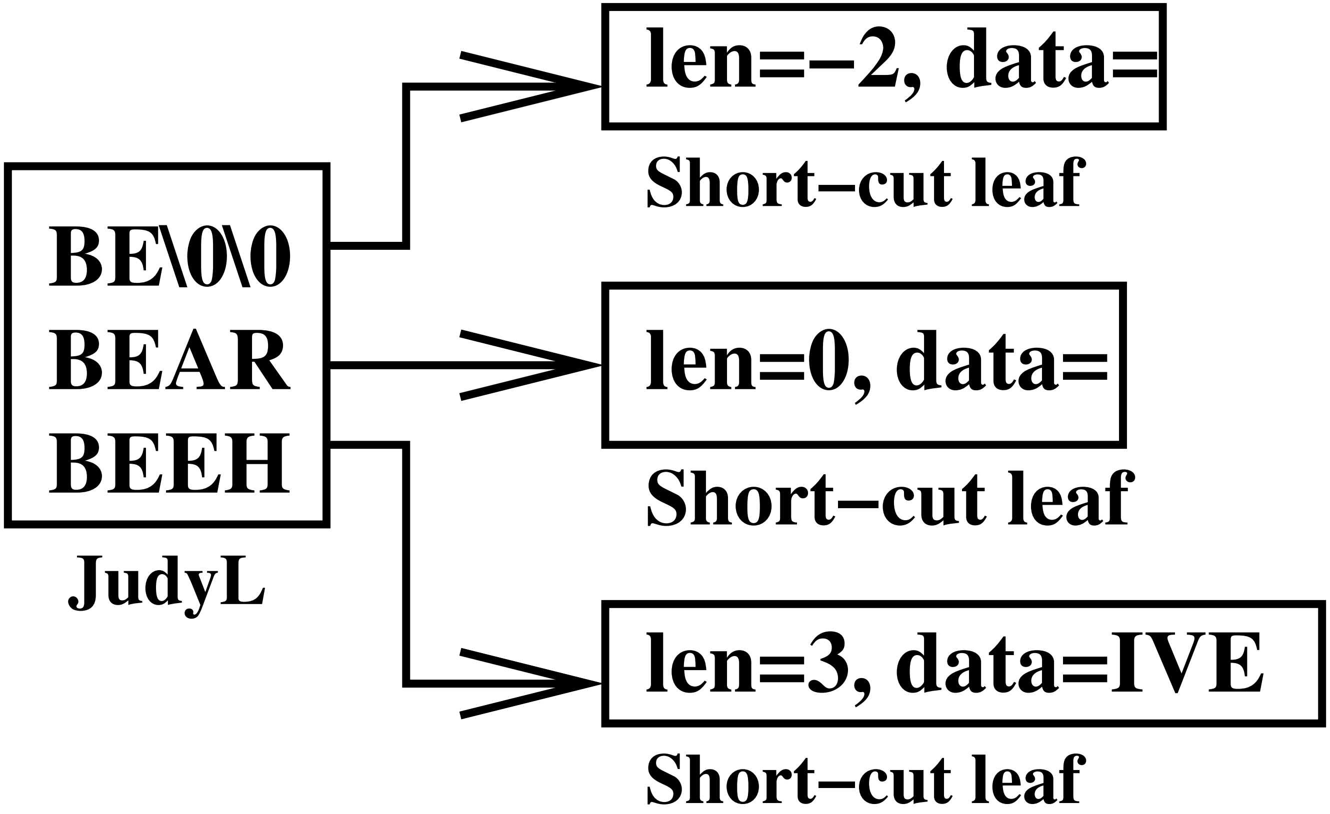


JudySL: A Trie of JudyL Arrays



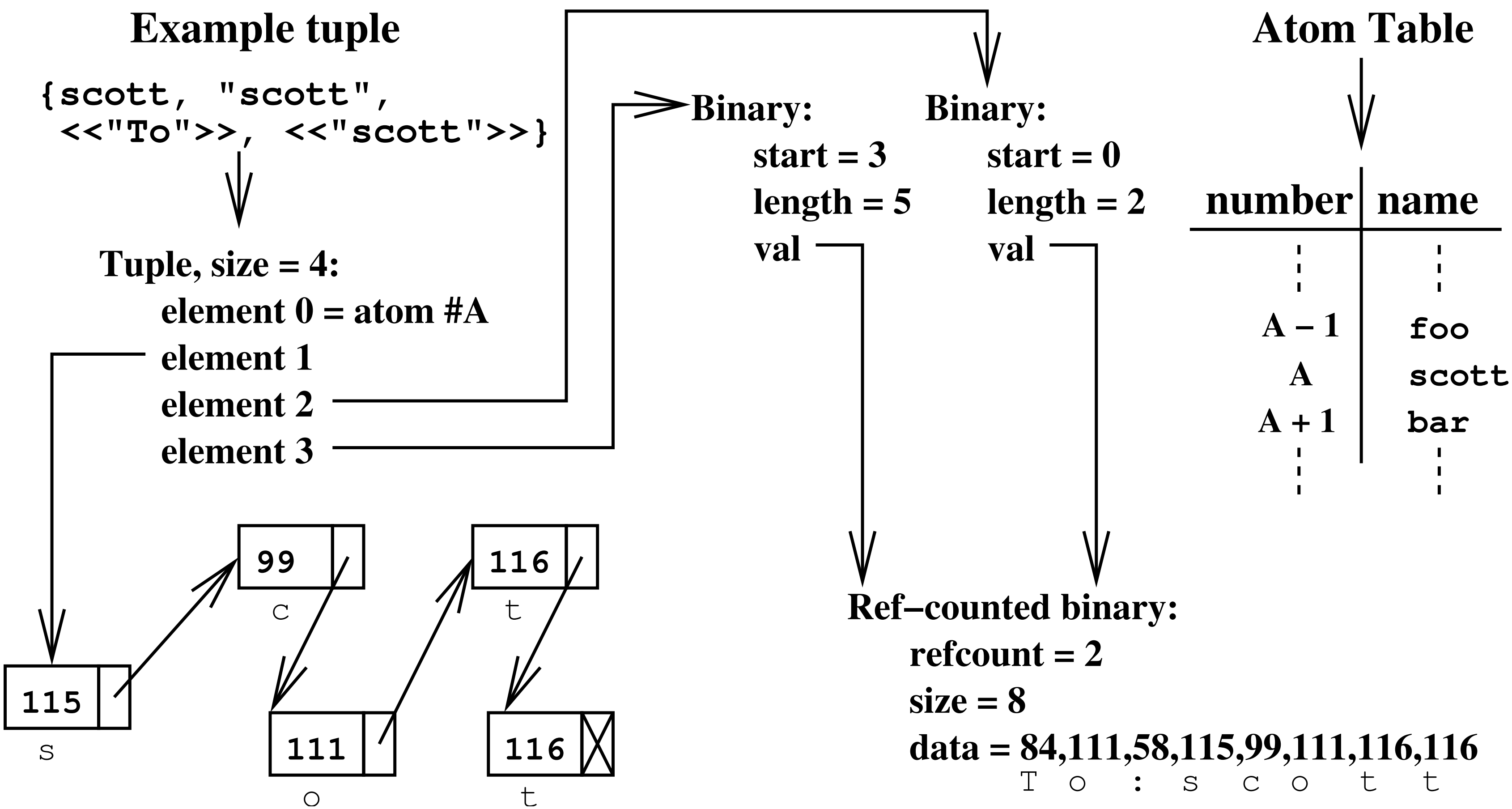
Words: BE, BEAR, BEEHOUSE–PAINT, BEEHIVE

JudyESL: A Variation of JudySL



Words: BE, BEAR, BEEHIVE

The Contiguous Key Problem



Judy-Based Tables

- `judysl` table type
  - Serialized key = `encode_NUL_bytes(term_to_binary(Key))`
- `judyesl` table type
  - The JudyESL library uses explicit string length, not NUL termination.
  - Serialized key = `term_to_binary(Key)`
- **NOTE:** JudySL and JudyESL preserve lexigraphic sort order of serialized keys, *not of original Erlang key terms*.

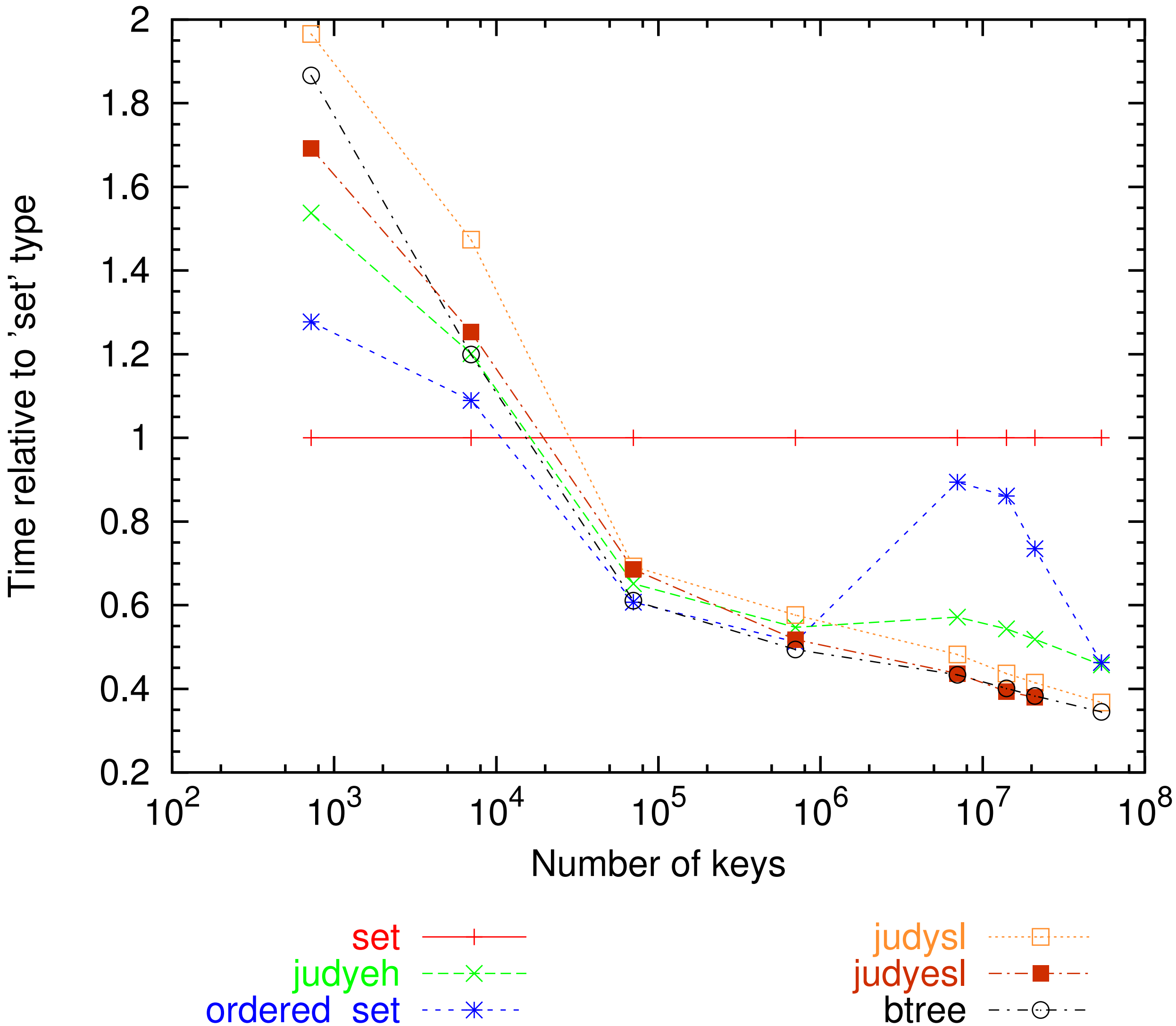
Judy-Based Tables (continued)

- `judyeh` table type
  - JudyL array for hash table: 2^{32} hash buckets!
  - No serialization, unlike `judysl` and `judyesl`
  - No meta-trie: search one JudyL array, not several
  - Hash collision rate $< 0.2\%$ for 7 million items

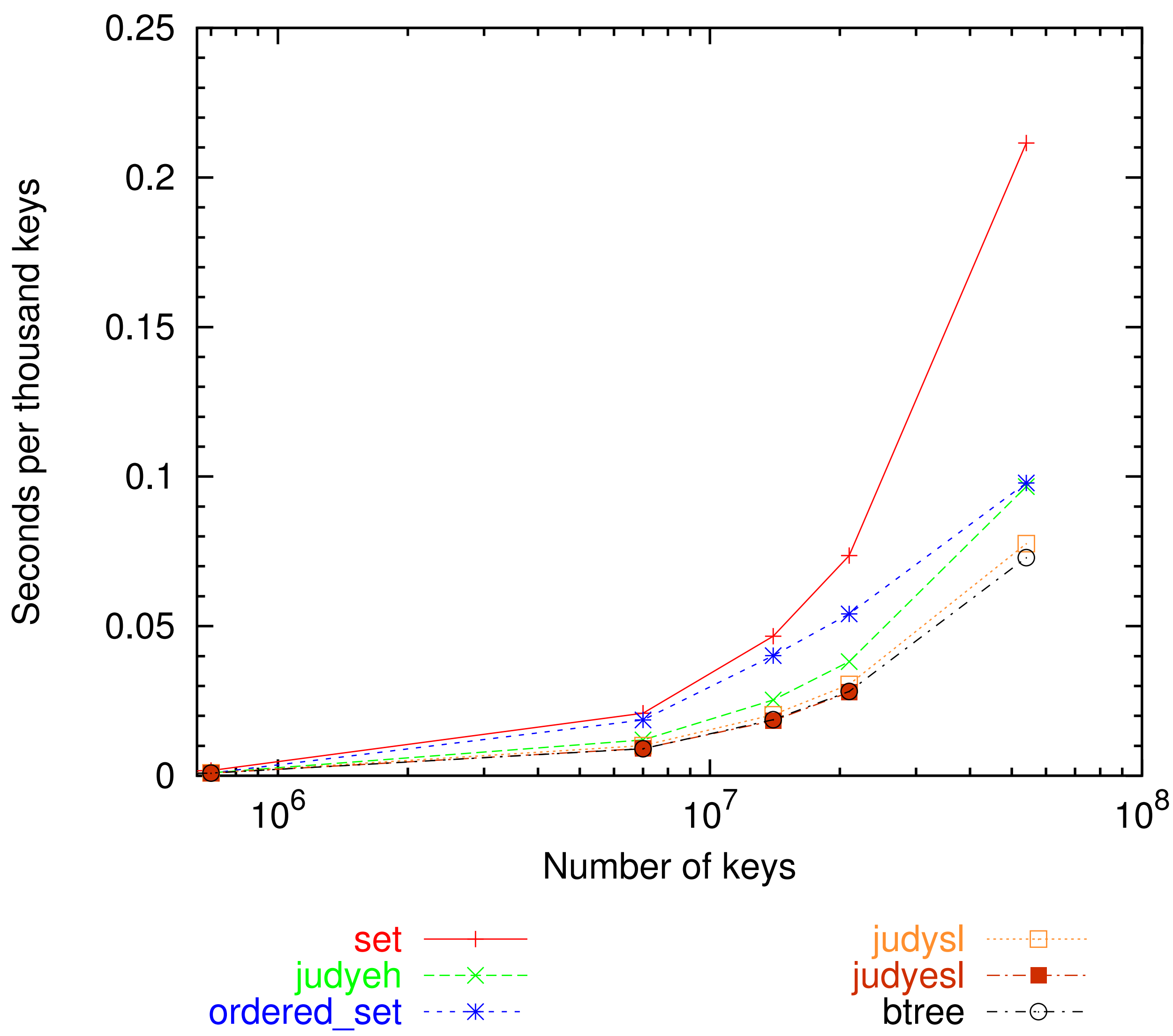
Experiment Design

- Intentionally maximize time executing ETS-related code.
  - Show table differences as much as possible.
  - Benchmark time in ETS-related code: 35-70%
    - SCCT time in ETS-related code: 18%
  - All other parts of VM unchanged.
- Benchmark result graphs
  - Overall, `set` is fastest "old" table type.
  - All run times normalized against `set`'s time.
  - Run time $<$ 1.0 $\rightarrow$ better
- CPU cache size reflected between 10^4 and 10^5 keys.

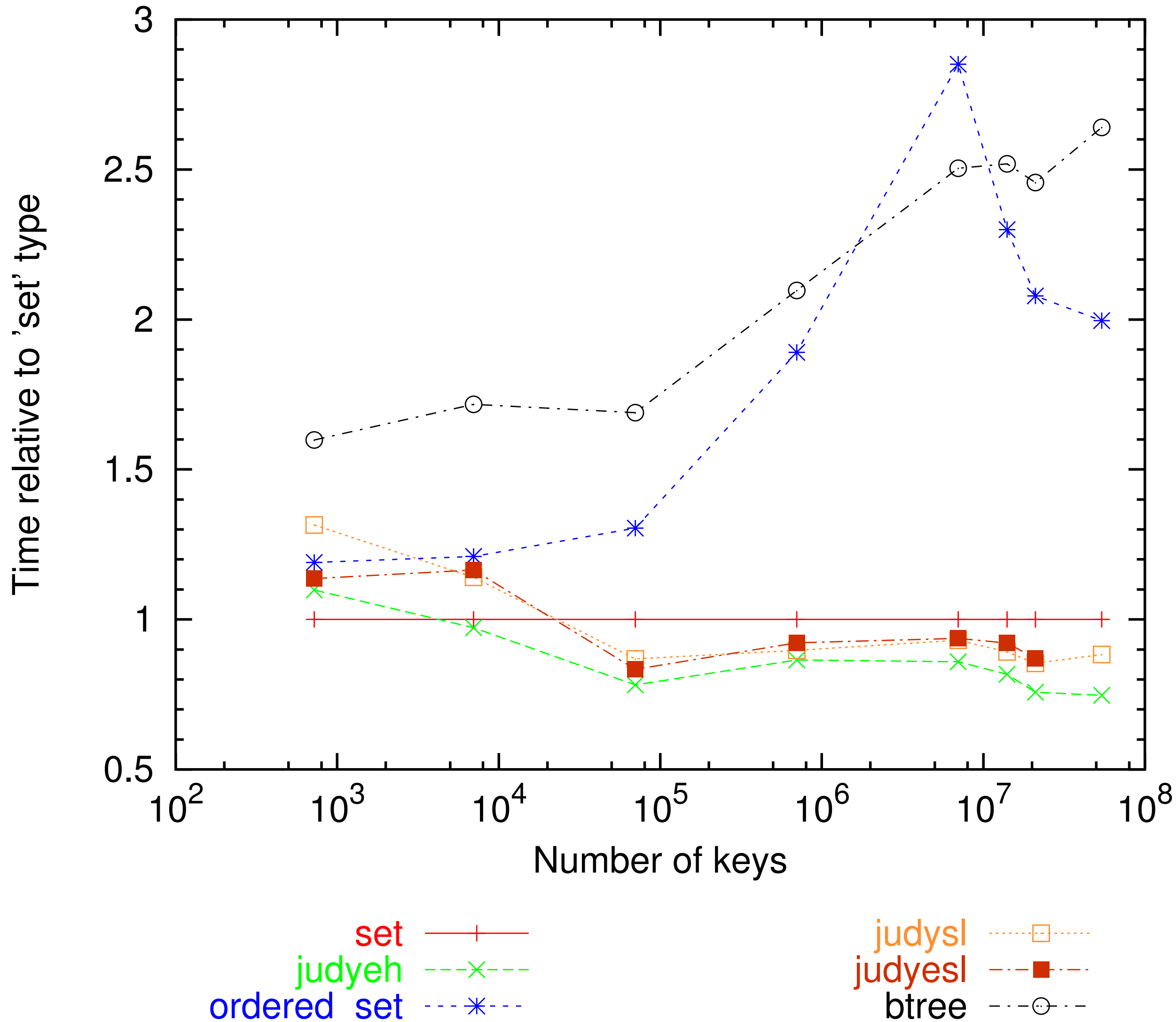
Sequential Insertion Into Empty Table



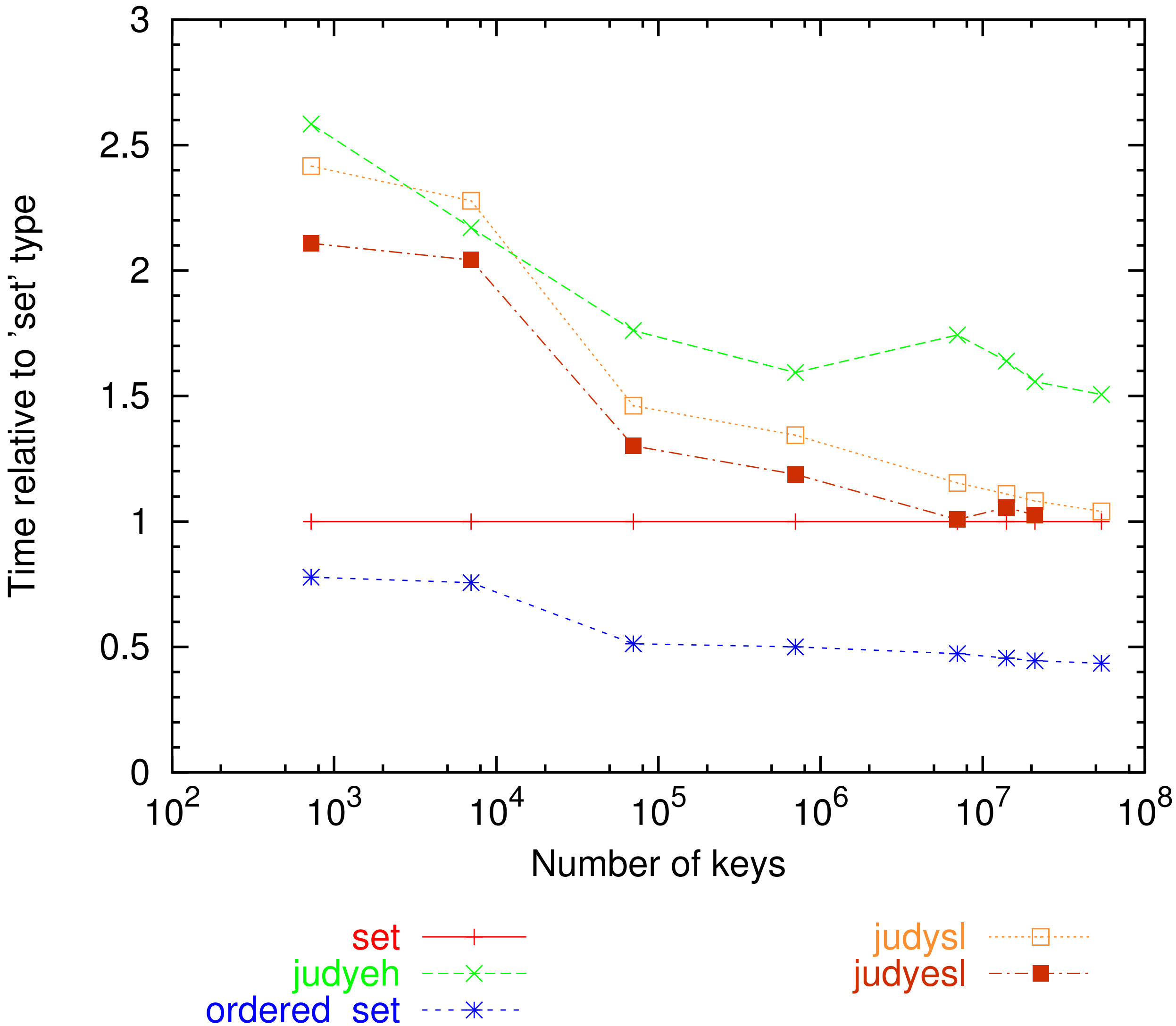
Sequential Insertion, Per 1K Keys



Random Lookup in Full Table



Forward Traversal of Full Table



Memory Utilization

Table type	Memory used by 70K keys	Memory used by 21M keys	Difference from set
btree	10.4MB	1,055MB	7.7%
judyeh	10.4MB	1,036MB	5.7%
judysl	10.4MB	1,033MB	5.4%
judyesl	11.3MB	1,324MB	35%
ordered_set	10.7MB	1,129MB	15%
set	10.2MB	980MB	—

Conclusion

- Judy array-based ETS tables perform very well for ETS table sizes that exceed CPU cache size.
  - Table traversal performance is probably fixable.
- Performance gain of Judy-based tables far exceeds extra memory consumption.
- JudySL- or JudyESL-based technique could perform better than `set` *and* still preserve key sort order.
- Using Judy arrays in a “real world” application can improve performance. Your application can probably benefit, too.