Atom Garbage Collection

Thomas Lindgren
thomasl_erlang@yahoo.com

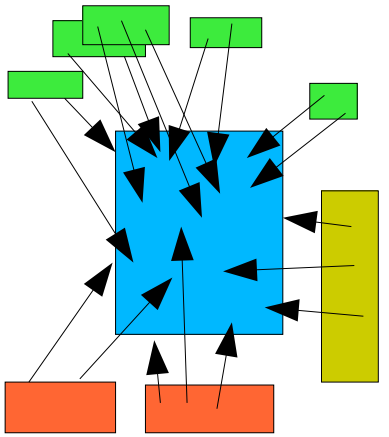
Why?

- Atoms are simple, useful, ubiquitous, dangerous
- Not collecting them introduces a **space leak**
  - Problem for long-running systems
  - Reclaim atom storage only by stopping node
- Programmer gets responsibility
  - But has few tools for managing atoms
- Unintuitive performance model
  - Space used ~ sum of all atom names **ever in system**

Characteristics of atom collector

- Each module has O(100) atoms
- A mid-sized system has O(20,000) atoms
- Atom table is normally not huge
  - Ets tables and process data may be much larger
- An atom collector probably *runs seldom*
  - Far less often than ordinary memory management
  - Should have *low overhead for common case*
- But: atoms may be used more aggressively than today if there is an atom collector

Garbage collecting atoms



- Atoms are a centralized resource
  - Appear everywhere: process data, ets tables, code, ...
- Stop-the-world can be used
  - Mark all atoms reachable from ets, process, code
  - Deallocate unmarked atoms
- **Problem**: long pause (needs to traverse everything)
- **Our solution**: incremental steps, *usually* short
  - But not *guaranteed* to be short

Incremental atom collector

- Migrate atoms from old to new *epoch/atom table*
  - When loading/unloading code, refcount atoms
  - At start of AGC, move all atoms refcount > 0
  - Before running a process, convert its atoms (~ gc)
  - When accessing ets, convert atoms in term
- **Eventually** all data in the system uses new atom table; then deallocate old table
- If incrementalism takes too long, hurry up the collector by doing more work (longer pauses)

Conclusion

- Implementation still to come
  - What policy should be used? When to run vs increase atom table size?
  - First step: retain atom info for each loaded module
- Proposed algorithm is hybrid copy/refcount
  - Common case: some space overhead, small runtime overhead when accessing ets, else pay as you go
  - Collector reasonably simple, incremental