UiO : University of Oslo

Implementing Garbage Collection for Active Objects on Top of Erlang

Sigmund Hansen

Master thesis autumn 2014

Problem

- ABS is a modeling language for distributed systems
- ABS has an Erlang back end
- Back end cannot collect garbage processes

Goals

The garbage collector should be:

- Fast
  It should not considerably slow down simulations
- Comprehensive
  It should collect most or all garbage
- Correct
  It should not collect resources required later

ABS Overview

- Executable models
- Functional subset
- Object-oriented
- Concurrency:
  - Active objects
  - Scheduling in Concurrent Object Groups (COGs)
  - RPC with message passing

Erlang Overview

- Concurrency:
  - Actor model
  - Message passing
- Fault-tolerant
- Functional

Garbage Collection Overview

Automatic memory management for dynamically allocated memory.

- Reference counting
- Tracing
  - Object graph traversal
  - De-allocation of unmarked objects

Active Object Collection

- Kept alive by associated process
- Must kill the process
- High-level implementation

Choice of Algorithm

- Reference counting cannot collect cyclic garbage.
- Killing processes, not collecting memory,
  i.e. no moving collectors
- Greater risk with non-standard collection algorithms

Mark and sweep that stops the world implemented.

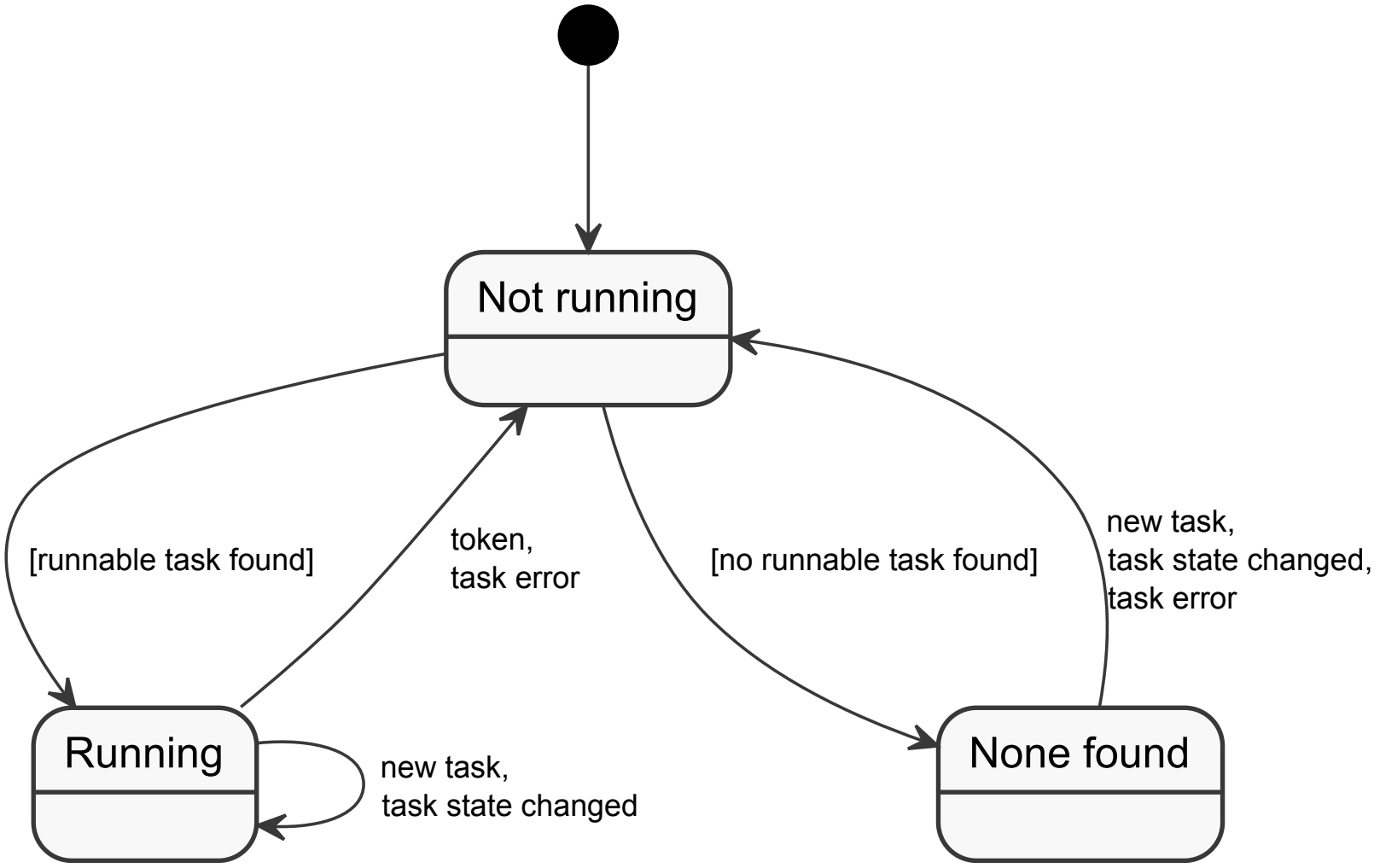
Stopping the World

- No interrupts
- Stop by messaging
- Tasks and scheduling must stop

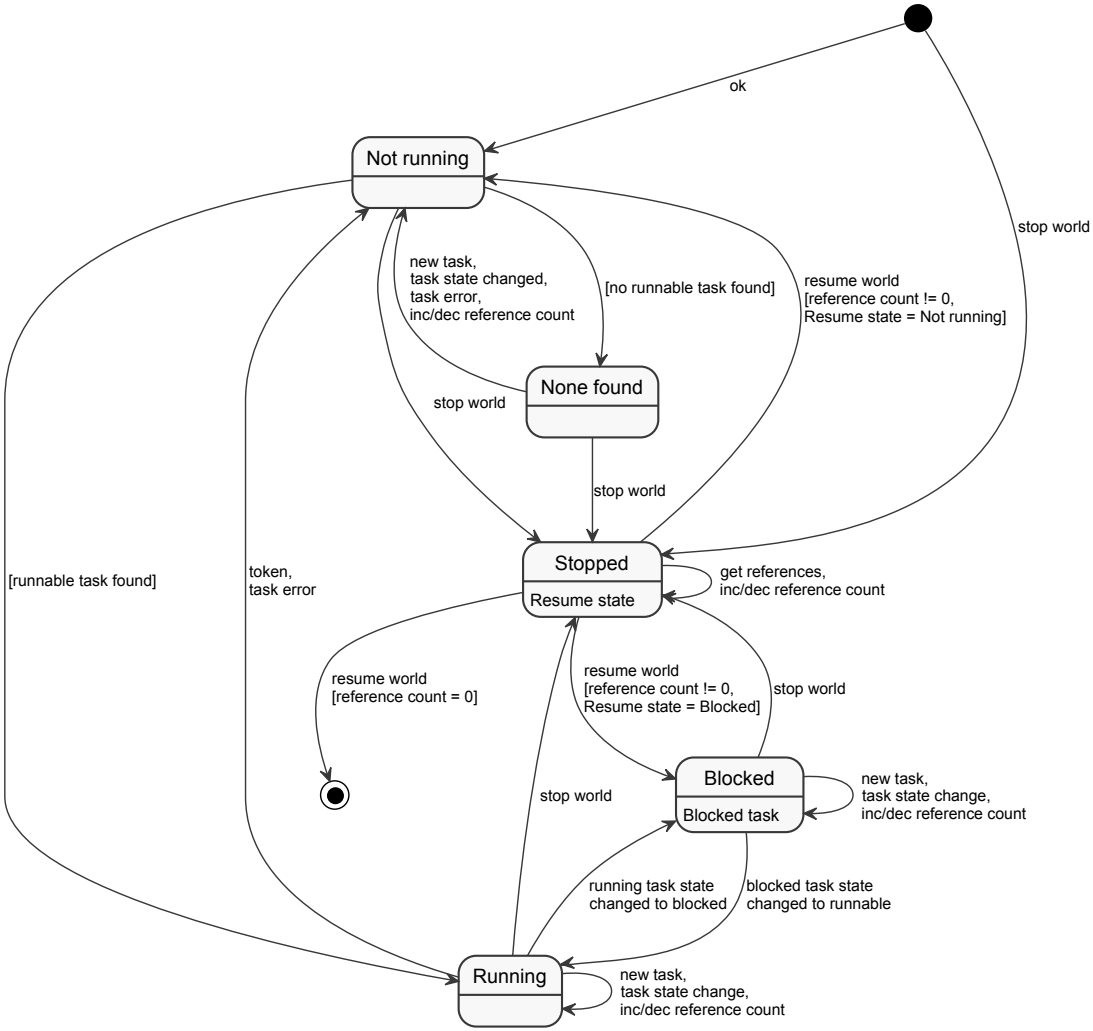
Blocking Operations

- Blocking on futures
- Blocking on object instantiation
- Inappropriate COG state during block

COG State Machine (Before)



COG State Machine (After)



Stopping Running Tasks

- Will reduce waiting
- COG messages task
- Task then blocks

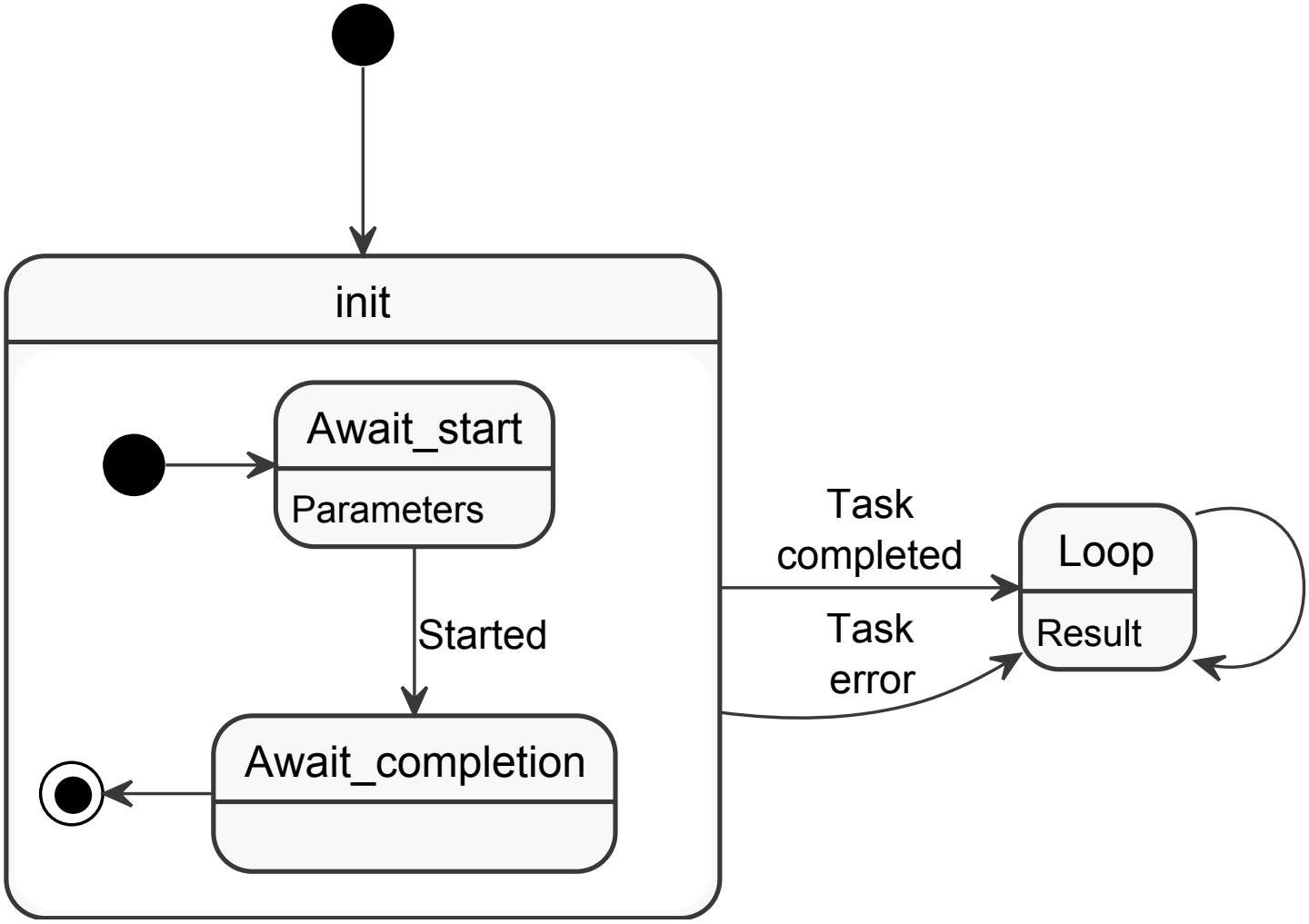
Asynchronous Method Calls on Inactive Objects

- Task initialization is synchronous
- Waits until object is activated
- Respond to GC messages while waiting

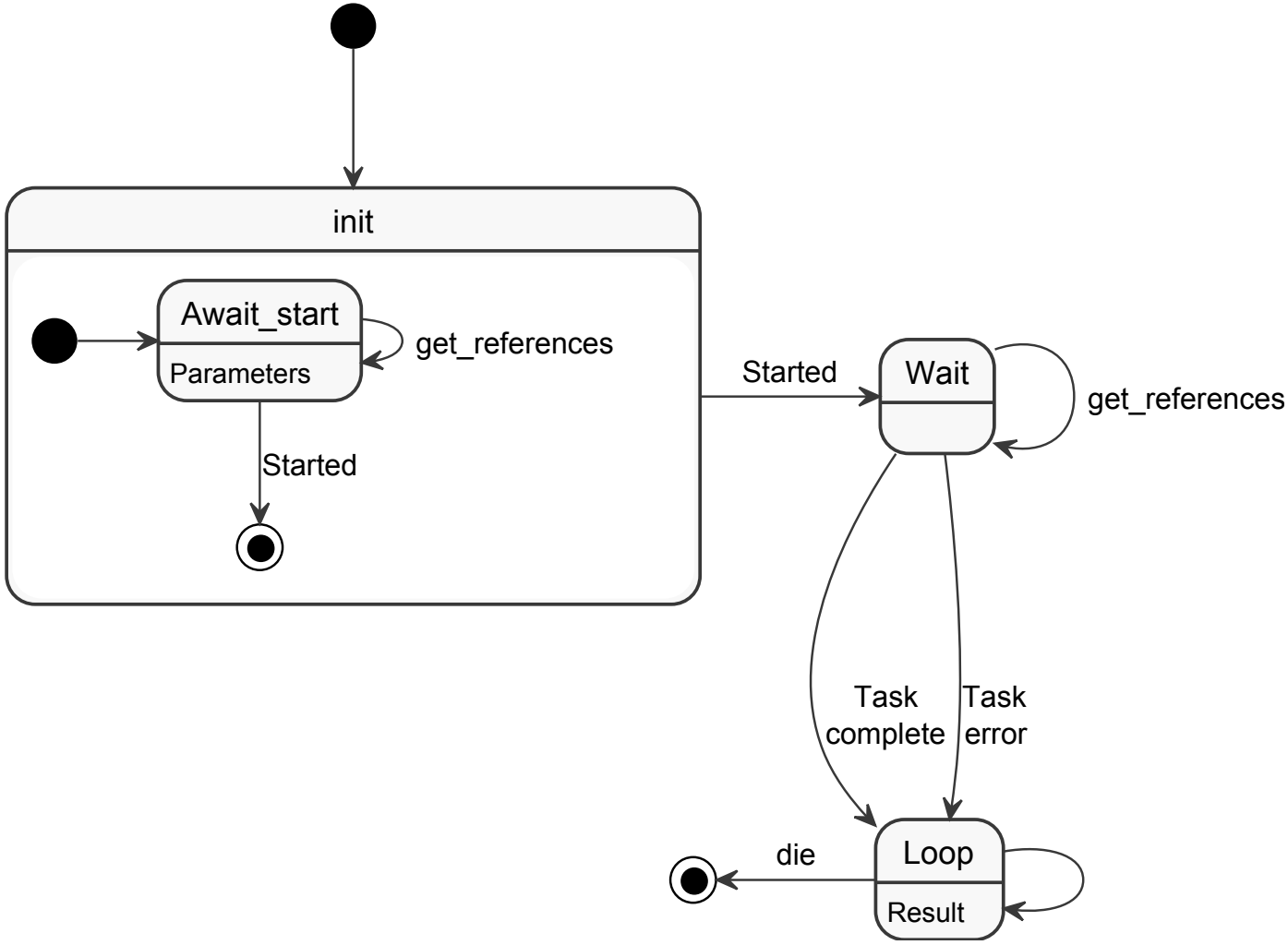
Tasks Created While the World is Stopping

- Task is created by a future
- Stopped COGs do not receive task creation messages
- Futures temporarily become roots and keep parameters

Future State Machine (Before)



Future State Machine (After)



A Complete View of COGs and Futures

Why is it required?

Marking Phase

- Messages all grays in parallel
- $B_{N+1} = B_N \cup G_N$
- $G_{N+1} = \mathit{References}(G_N) \setminus B_{N+1}$
- Sweeps when gray set is empty

Sweeping

- Identifies white objects
- Identifies white futures
- Sends message to kill them
- Resume message sent to COGs

Collecting COGs with Reference Counting

- Counts objects left in group
- COG stops if world resumes and it is empty
- Messages garbage collector

Static analysis

- Types that may contain references
- Reaching unawaited futures
- Other potential analyses:
  - Loops with synchronization points
  - Deep functions

Triggering Garbage Collection

When should garbage be collected:

- On time?
- Number of objects and futures?
- Process ratio?

Data Collection

- Garbage collector state changes
- Number of objects and futures swept
- Memory usage and number of objects, futures and COGs

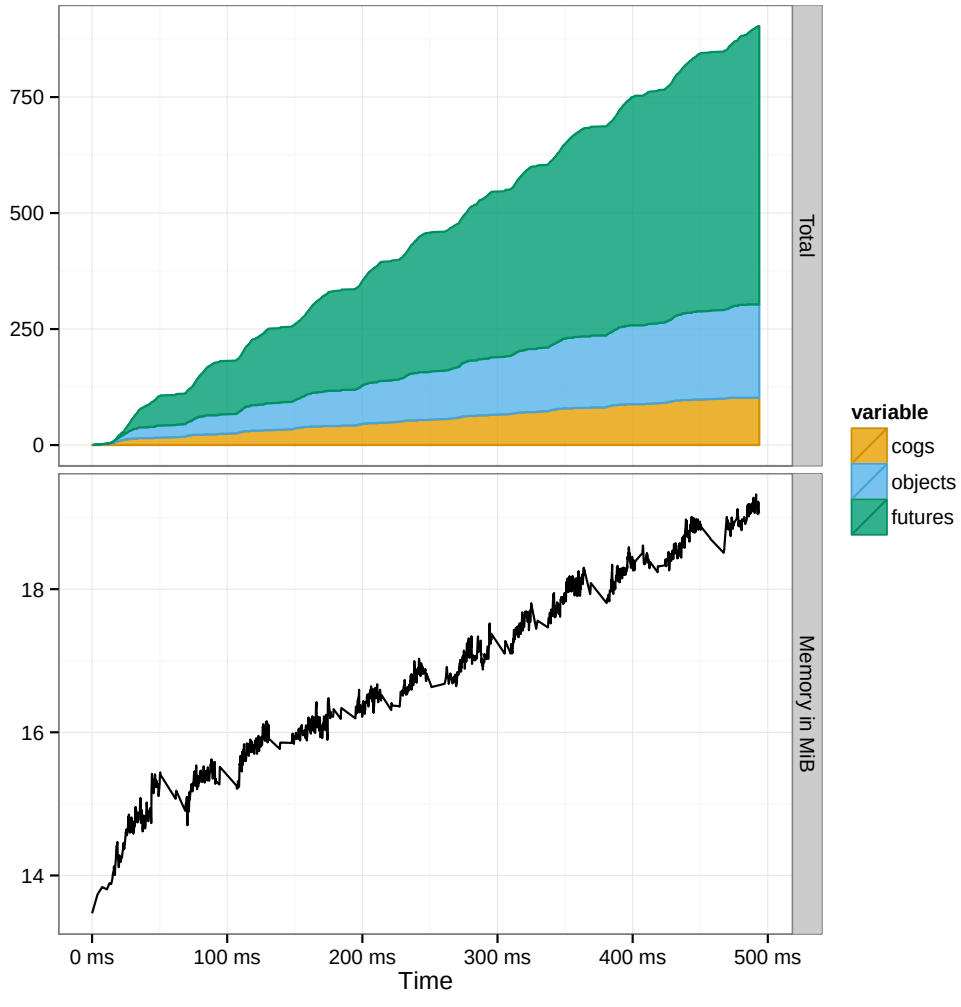
Test Cases

- Ping pong
  Basic test with cyclic garbage
- Sequences
  Unstoppable loop
- Prime Sieve
  Long-running tasks
- MapReduce - Indexing
  Real-world test case

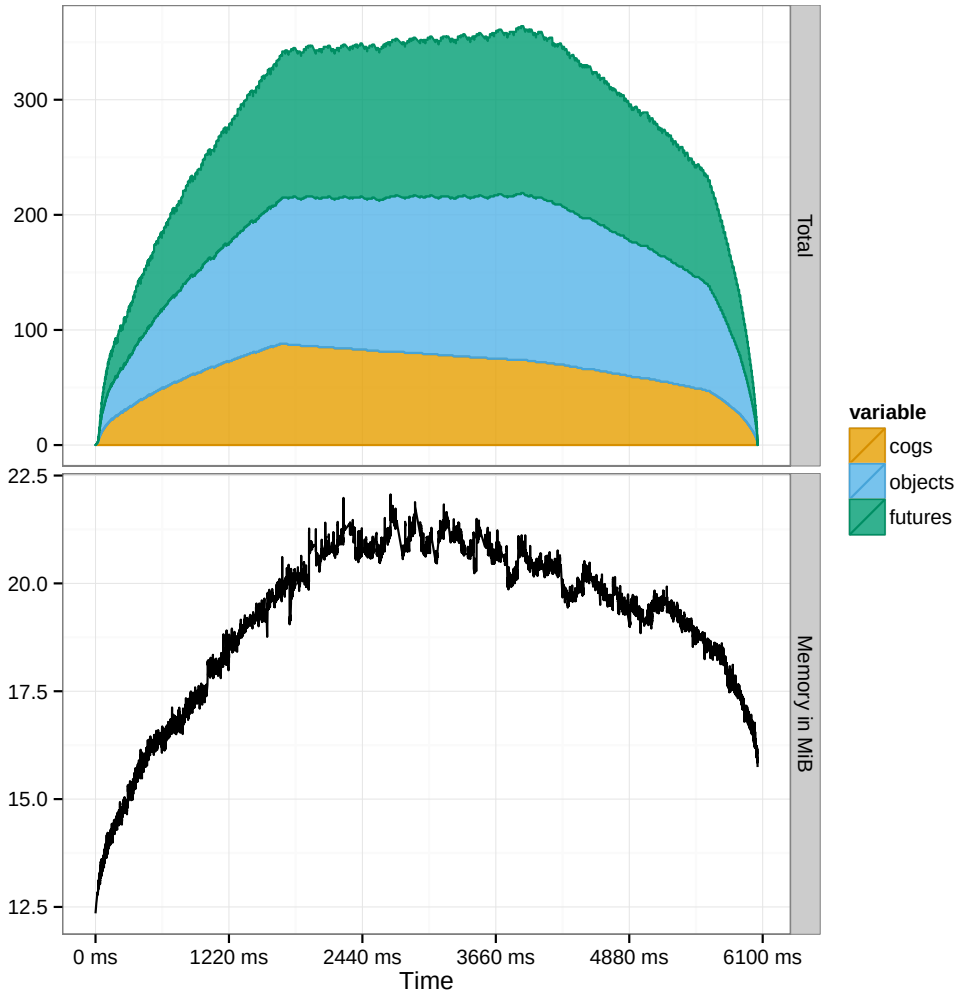
Ping Pong - Time

Collection scheme	Real time	Relative	Relative without idle
Before GC	1274 ms	100.0 %	100.0 %
Never collect	1257 ms	98.6 %	93.5 %
Always collect	1854 ms	145.5 %	311.3 %
Collect on count	1293 ms	101.4 %	106.7 %
Collect on time	1261 ms	98.9 %	94.9 %
+ Stop running	1265 ms	99.2 %	96.5 %
Collect on either	1290 ms	101.2 %	105.5 %
+ Stop running	1298 ms	101.8 %	108.4 %

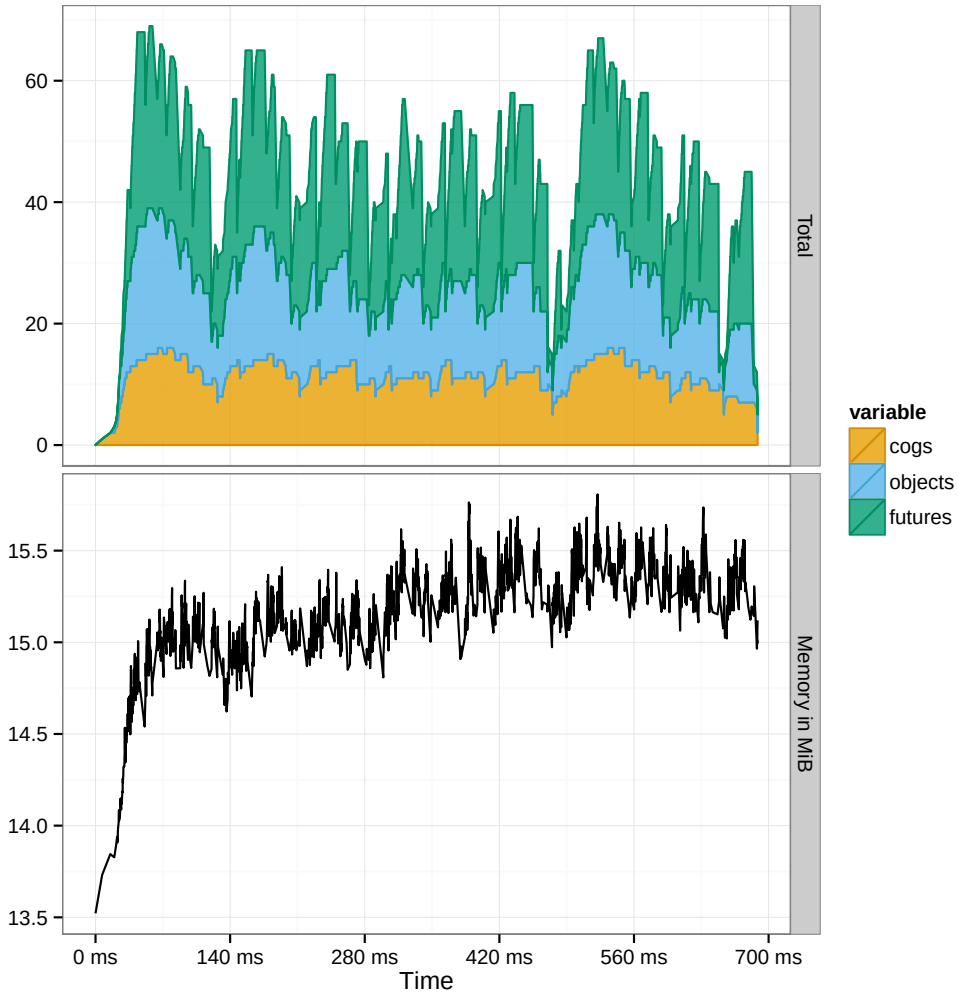
Ping Pong - No GC



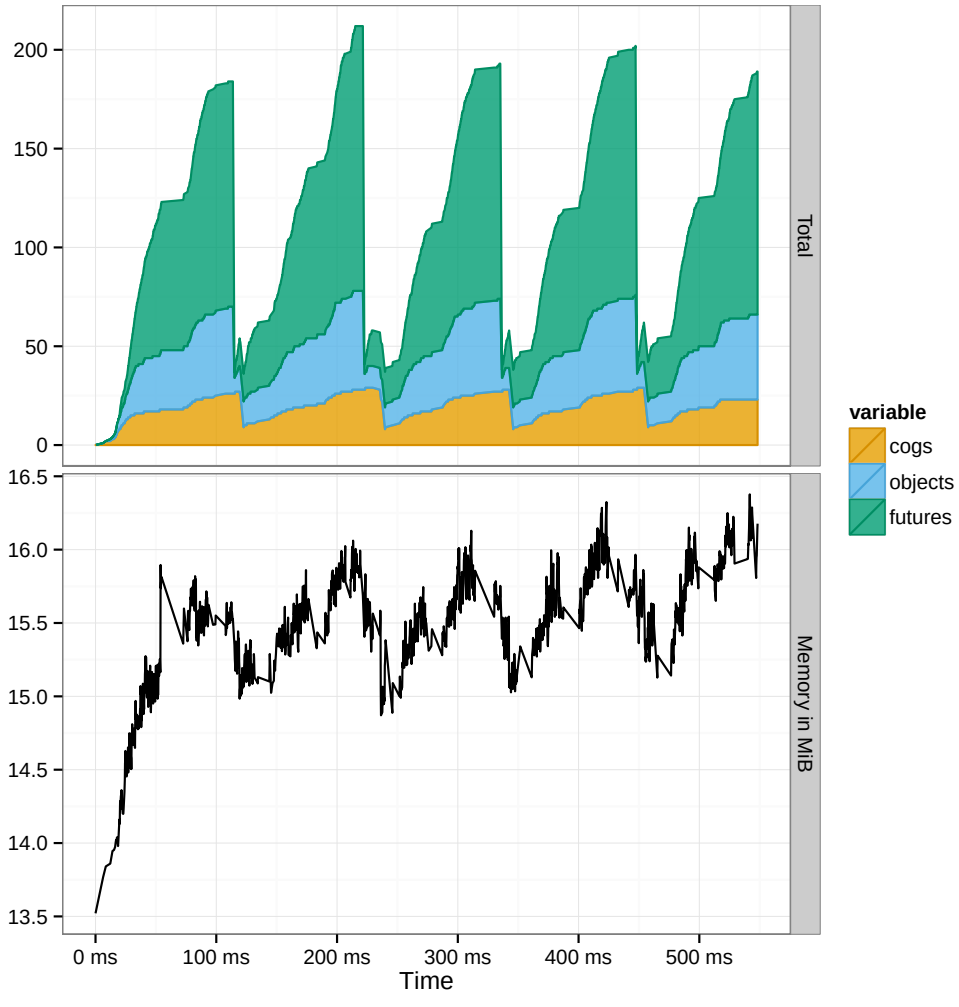
Ping Pong - Always collecting



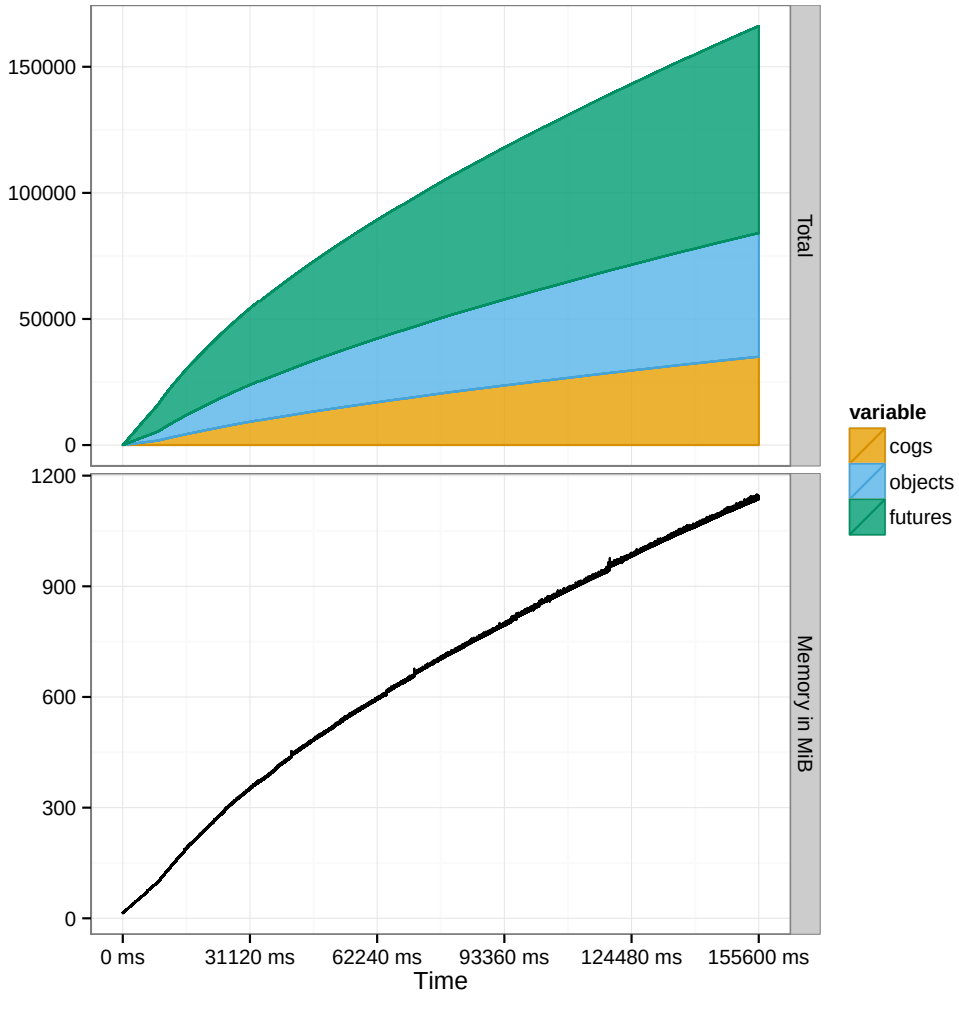
Ping Pong - By count



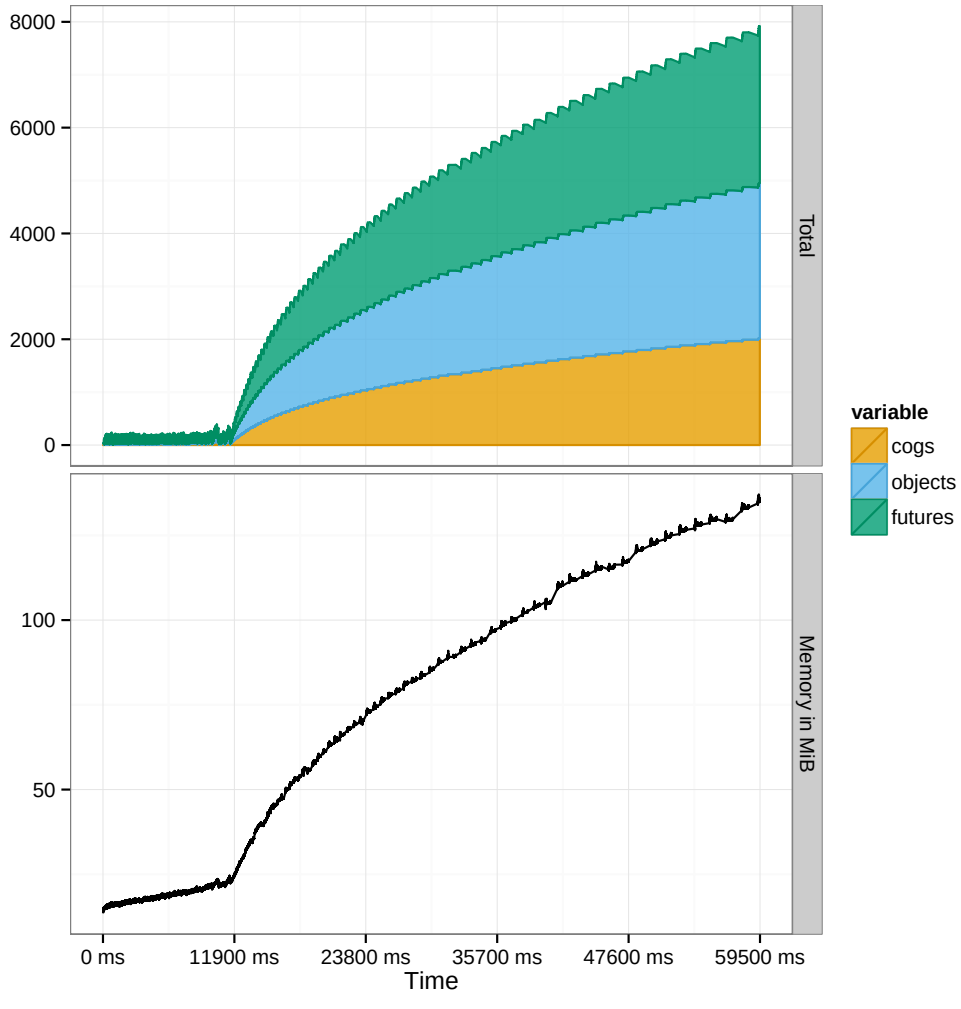
Ping Pong - Timed



Infinite Ping Pong - No GC



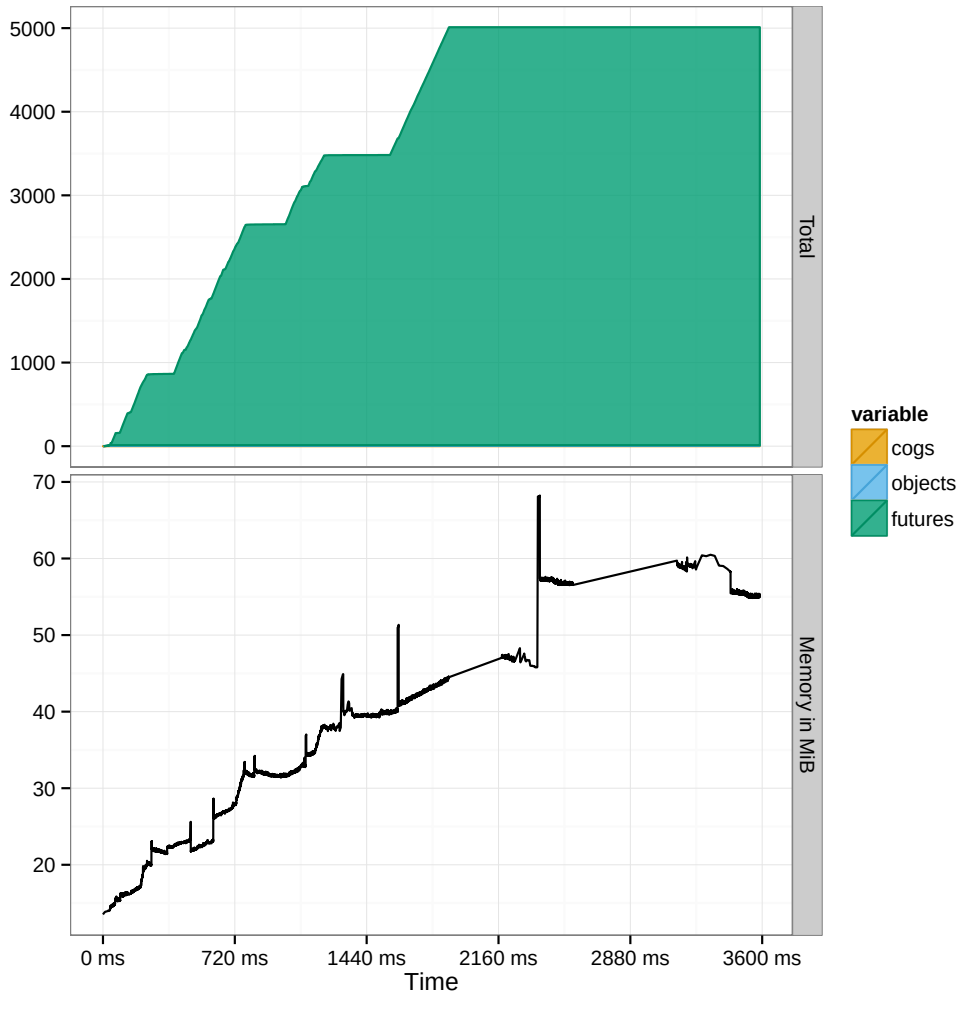
Infinite Ping Pong - Timed



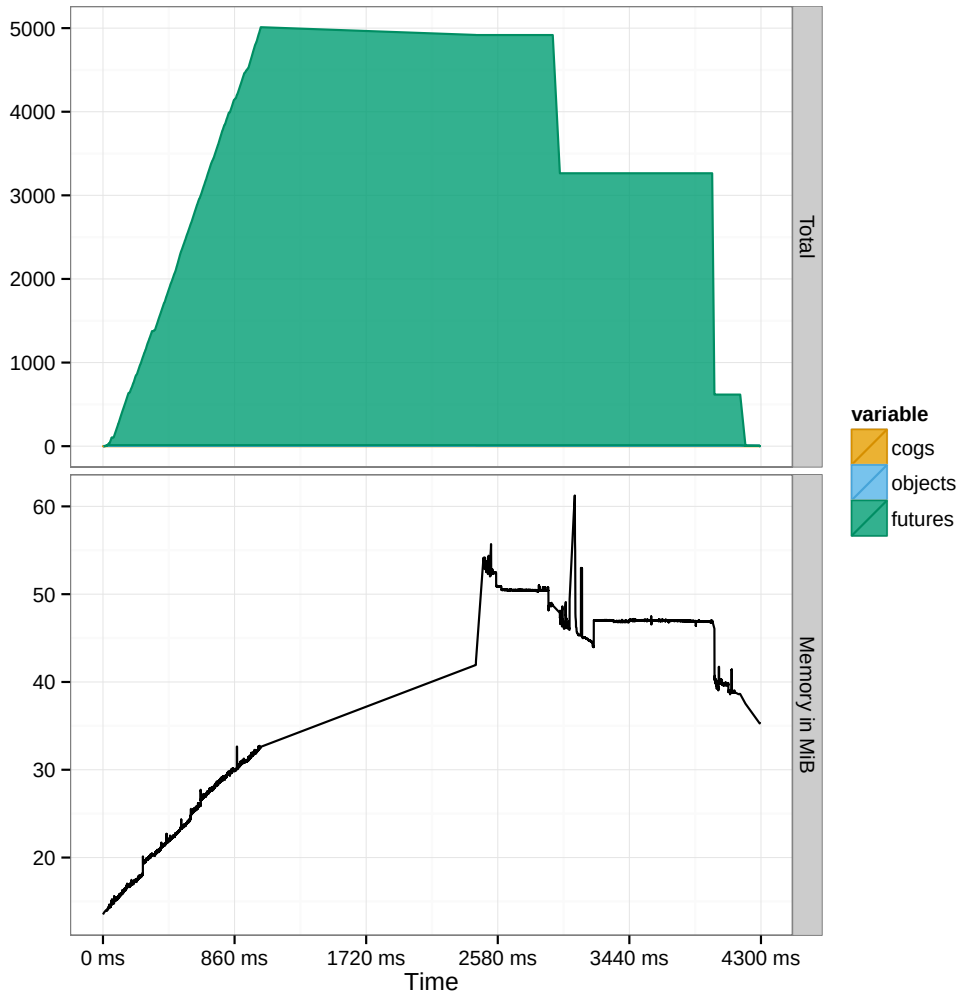
Sequences - Time

Collection scheme	Real time	Relative	Relative without idle
Before GC	1395 ms	100.0 %	100.0 %
Never collect	1439 ms	103.2 %	111.2 %
Always collect	39159 ms	2807.2 %	9662.1 %
Collect on count	16801 ms	1204.4 %	4001.0 %
Collect on time	1622 ms	116.3 %	157.5 %
+ Stop running	1530 ms	109.7 %	134.1 %
Collect on either	17054 ms	1222.6 %	4065.1 %
+ Stop running	12614 ms	904.3 %	2940.8 %

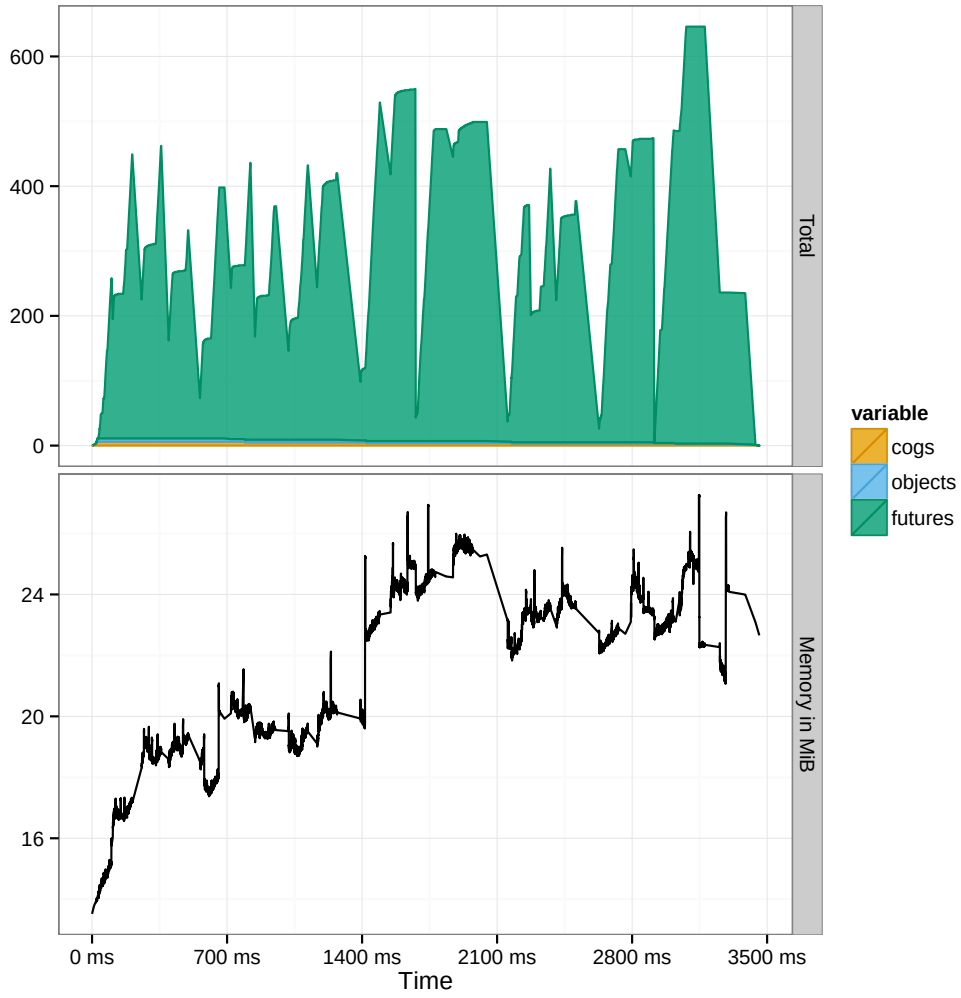
Sequences - No GC



Sequences - Timed



Sequences -
Timed stopping running processes



Prime Sieve - Time

Collection scheme	Real time	Relative	Relative without idle
Before GC	1448 ms	100.0 %	100.0 %
Never collect	1453 ms	100.3 %	101.1 %
Always collect	1903 ms	131.4 %	201.6 %
Collect on count	1482 ms	102.4 %	107.6 %
Collect on time	1454 ms	100.4 %	101.3 %
+ Stop running	1542 ms	106.5 %	120.9 %
Collect on either	1472 ms	101.6 %	105.3 %
+ Stop running	1545 ms	106.7 %	121.6 %

Indexing - Time

Collection scheme	Real time	Relative	Relative without idle
Before GC	1356 ms	100.0 %	100.0 %
Never collect	1373 ms	101.2 %	104.7 %
Always collect	5043 ms	371.8 %	1134.2 %
Collect on count	1408 ms	103.9 %	114.7 %
Collect on time	1398 ms	103.1 %	111.7 %
+ Stop running	1405 ms	103.6 %	113.9 %
Collect on either	1410 ms	104.0 %	115.3 %
+ Stop running	1434 ms	105.7 %	121.9 %

Conclusion

Timed trigger gives acceptable results.

- Acceptably fast
- Mark and sweep is complete
- Measures taken stopping the world to ensure correctness

Discussion

Testing with infinite loops have discouraging results. Future work to lower overhead may be needed:

- Optimize stopping the world
- Optimize marking
- Concurrent collection
- Change of algorithm