# Whippet: A New GC for Guile

4 Feb 2023 – FOSDEM

Andy Wingo

## Guile is…

Mostly written in Scheme

Also a 30 year old C library

```
// API
SCM scm_cons (SCM car, SCM cdr);

// Many third-party users
SCM x = scm_cons (a, b);
```

## Putting the C into GC

```
SCM x = scm_cons (a, b);
```

Live objects: the *roots*, plus anything a live object refers to

How to include x into roots?

- Refcounting
- Register (& later unregister) &x with gc
- **Conservative roots**

## Conservative roots

Treat every word in stack as potential root; over-approximate live object set

1993: Bespoke GC inherited from SCM

2006 (1.8): Added pthreads, bugs

2009 (2.0): Switch to BDW-GC

BDW-GC: Roots also from `extern SCM foo;`, etc

## Conservative roots

+: Ergonomic, eliminates class of bugs (handle registration), no compiler constraints

-: Potential leakage, no compaction / object motion; no bump-pointer allocation, calcifies GC choice

## What if I told you

You can find roots conservatively *and*

- move objects and compact the heap
- do fast bump-pointer allocation
- incrementally migrate to precise roots

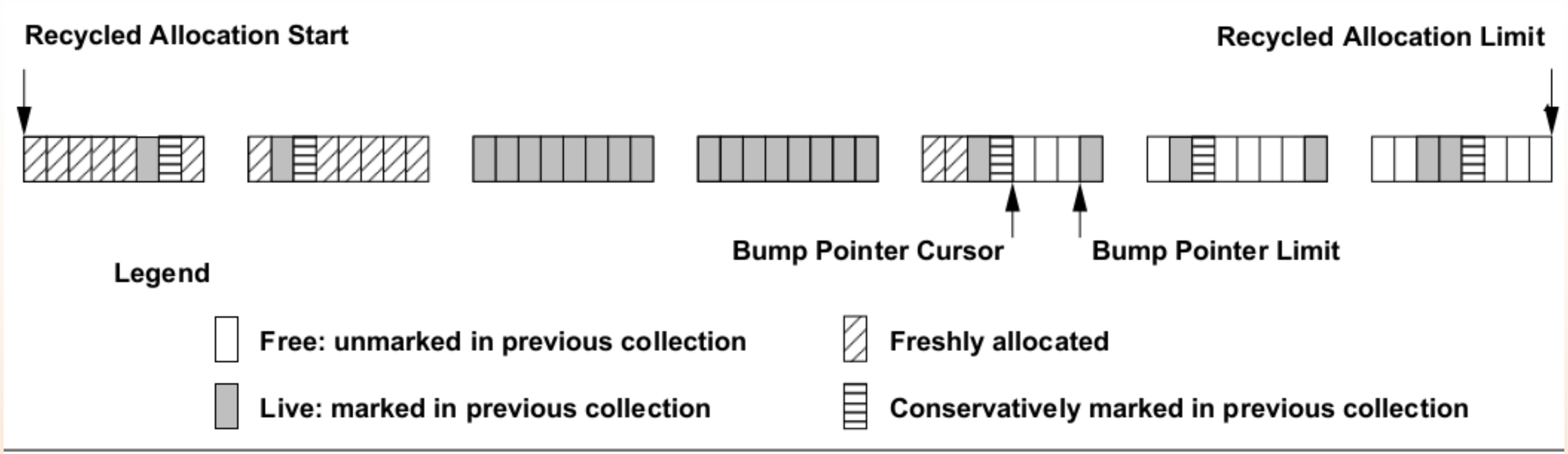
BDW is not the local maximum

## Immix

Fundamental GC algorithms

- mark-compact
- mark-sweep
- evacuation
- *mark-region*

Immix is a mark-region collector


Recycled Allocation Start
Recycled Allocation Limit
Bump Pointer Cursor
Bump Pointer Limit
Legend
Free: unmarked in previous collection
Freshly allocated
Live: marked in previous collection
Conservatively marked in previous collection

Allocate: Bump-pointer into holes in thread-local block, objects can span lines but not blocks

Trace: Mark objects *and lines*

Sweep: Coarse eager scan over line mark bytes

## Immix: Opportunistic evacuation

Before trace, determine if compaction needed. If not, mark as usual

If so, select candidate blocks and evacuation target blocks. When tracing in that block, try to evacuate, fall back to mark

## Immix: Guile

Opportunistic evacuation compatible with conservative roots!

Bump-pointer allocation

Compaction!

1 year ago: start work on *WIP* GC implementation

## Whippet vs Immix: Tiny lines

Immix: 128B lines + mark bit in object

Whippet: 16B “lines”; mark *byte* in side table

More size overhead: 1/16 vs 1/128

Less fragmentation (1 live obj = 2 lines retained)

More alloc overhead?  More small holes

## Whippet vs Immix: Lazy sweeping

Immix: “cheap” eager coarse sweep

Whippet: just-in-time lazy fine-grained sweep

Corrolary: Data computed by sweep available when sweep complete

Live data at previous GC only known before next GC

Empty blocks discovered by sweeping

## Whippet vs BDW

Compaction/defrag/pinning, heap shrinking, sticky-mark generational GC, threads/contention/allocation, ephemerons, precision, tools

## Whippet vs BDW: Motion

Heap-conservative tracing: no object moveable

Stack-conservative tracing: stack referents pinned, others not

Whippet: If whole-heap fragmentation exceeds threshold, evacuate most-fragmented blocks

Stack roots scanned first; marked instead of evacuated, implicitly pinned

Explicit pinning: bit in mark byte

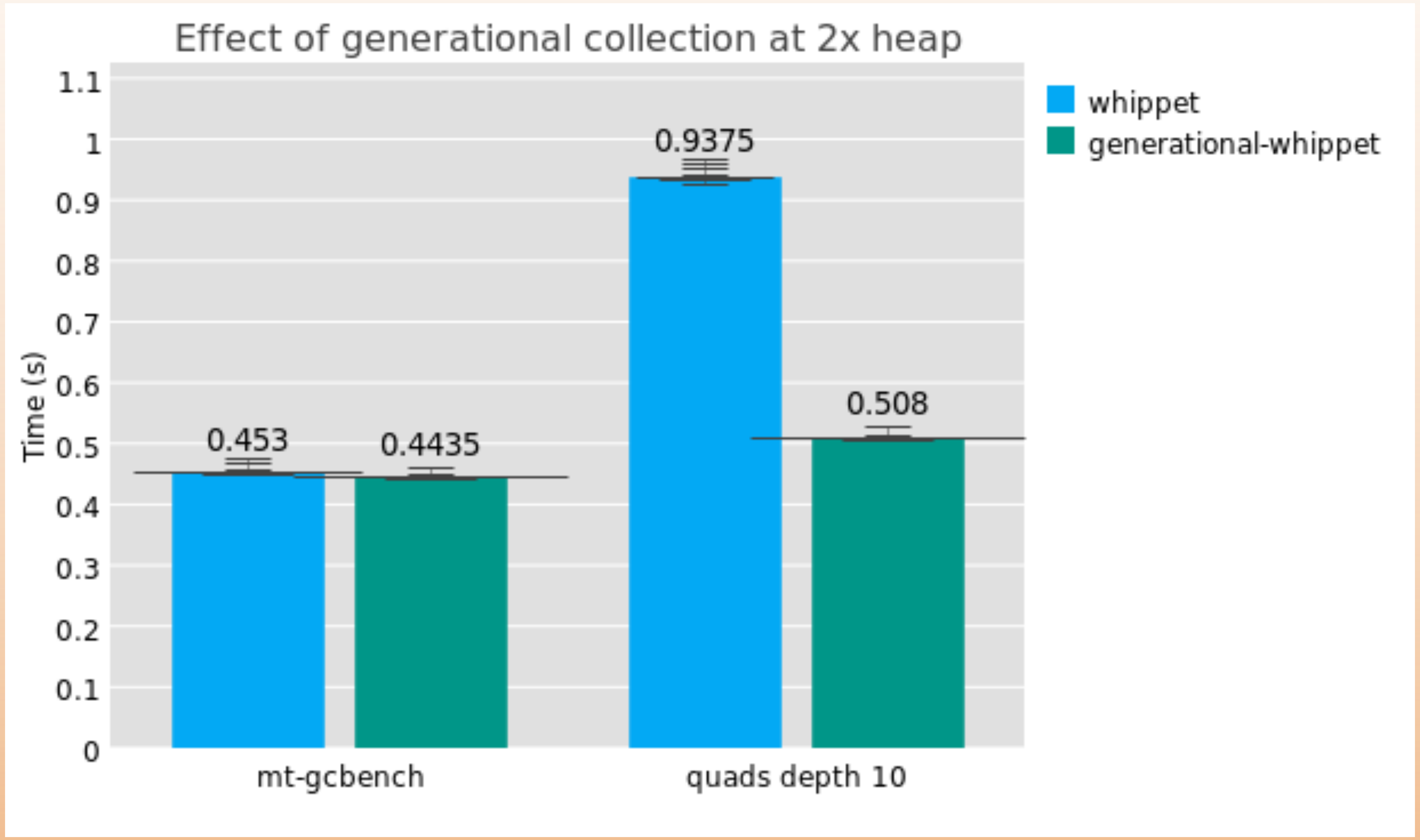
## Whippet vs BDW: Shrinking

Lazy sweeping finds empty blocks: potentially give back to OS

Need empty blocks? Do evacuating collection

Possibility to do adaptive heap size management (`http://marisa.moe/balancer.html`) `https://wingolog.org/archives/2022/10/22/the-sticky-mark-bit-algorithm`

## Card marking barrier (256B); compare to BDW mprotect / SIGSEGV

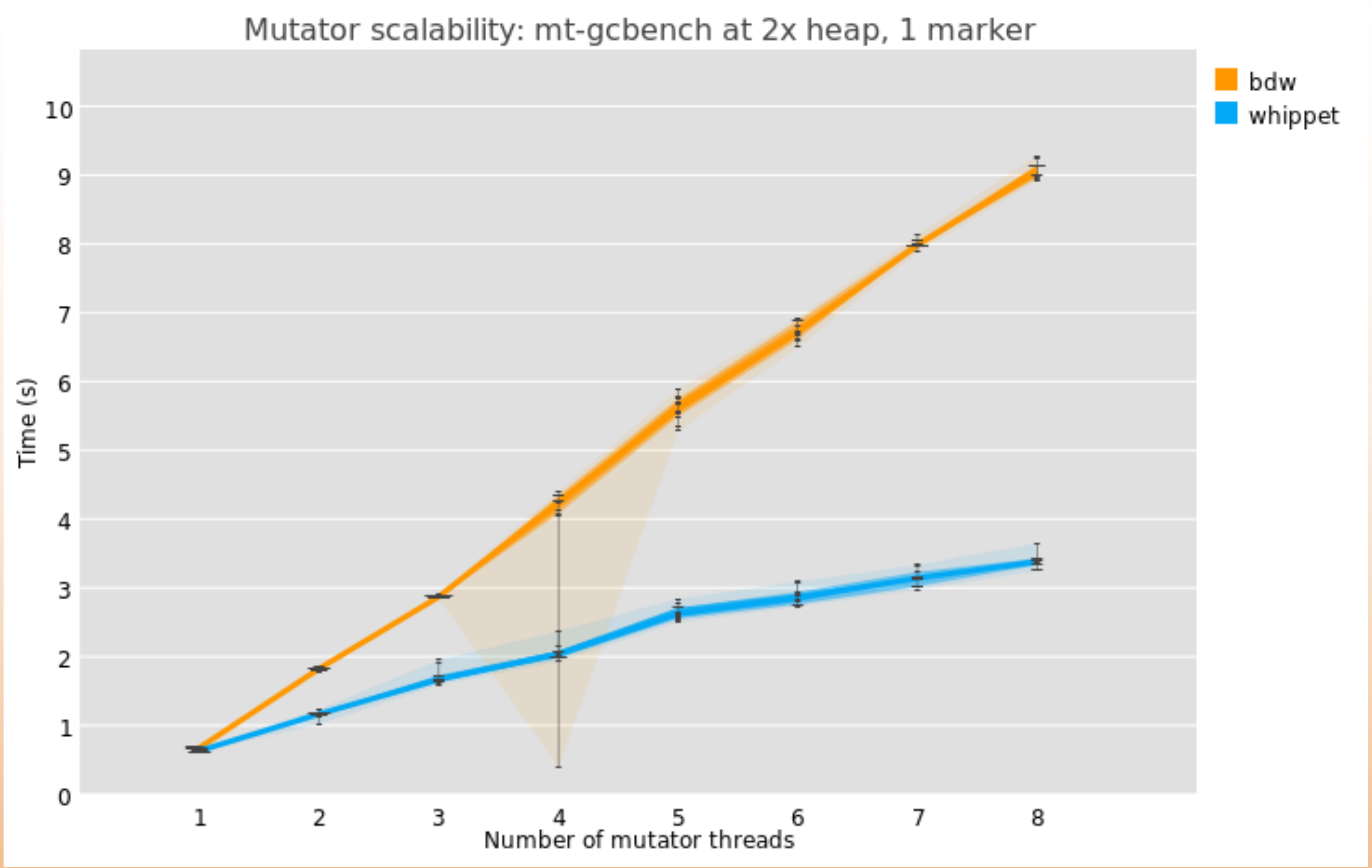


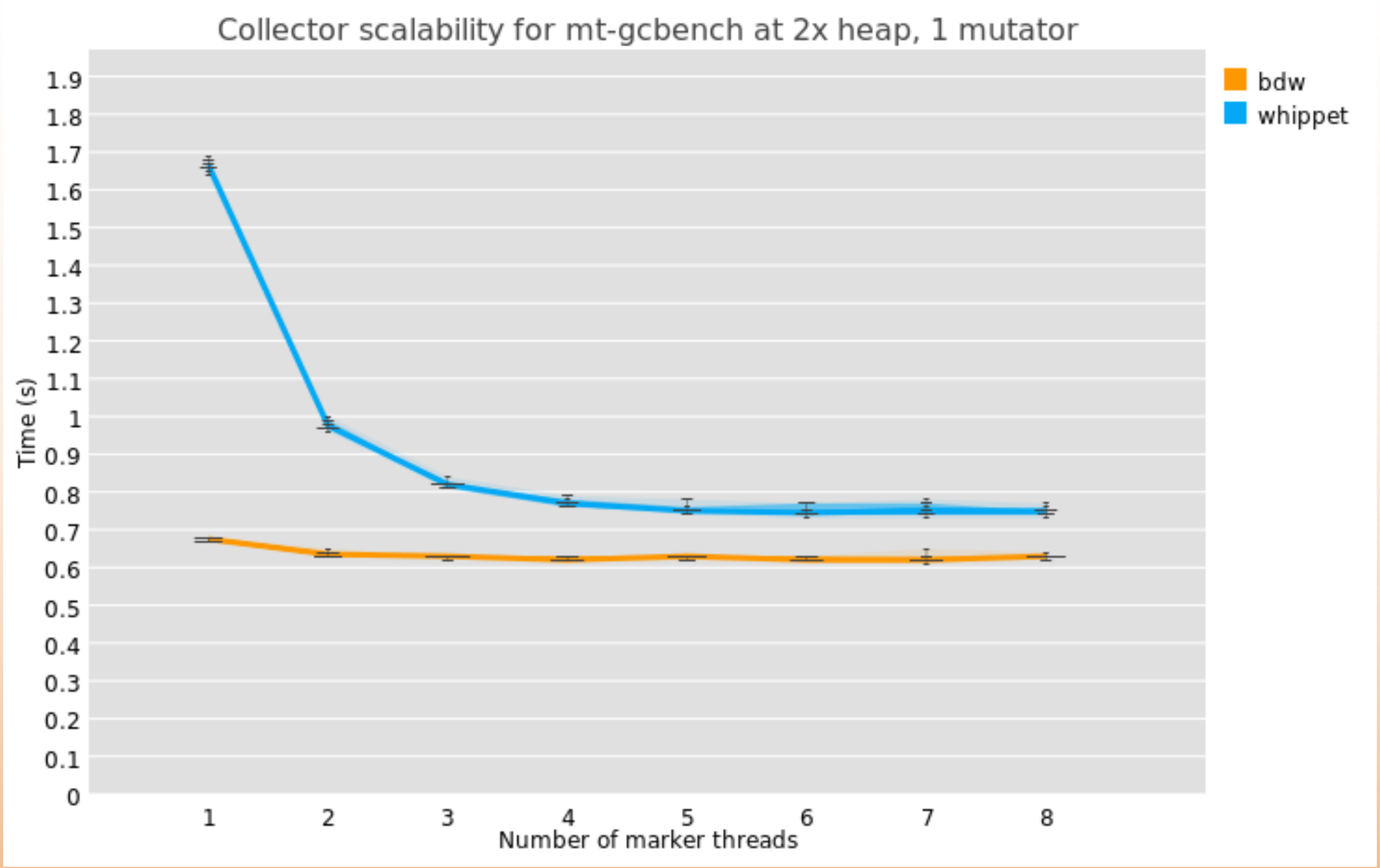
## Whippet vs BDW: Scale

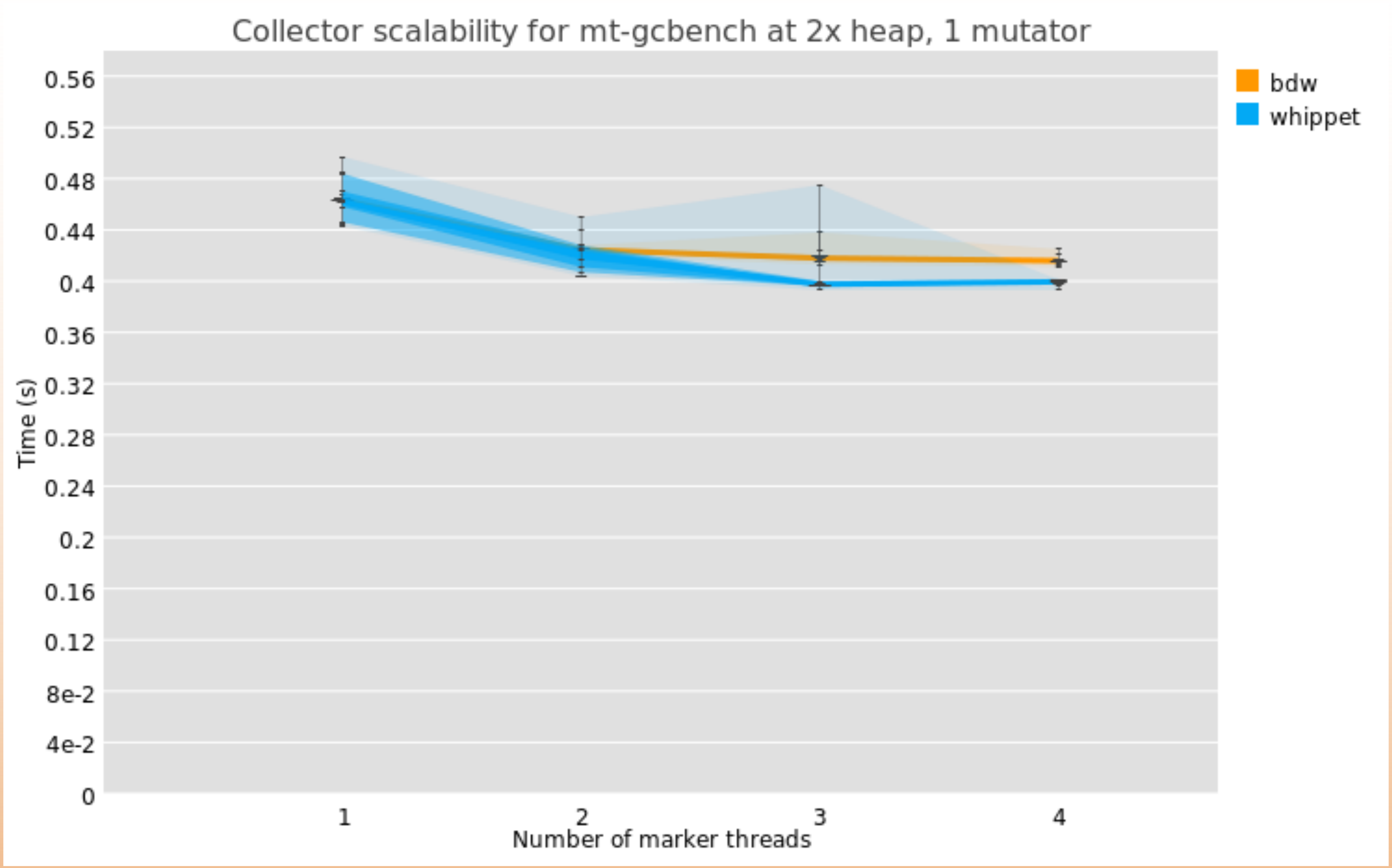
BDW: TLS segregated-size freelists, lock to refill freelists, SIGPWR for stop

Whippet: thread-local block, sweep without contention, wait-free acquisition of next block, safepoints to stop with ragged marking

Both: parallel markers


Mutator scalability: mt-gcbench at 2x heap, 1 marker
Time (s)
0
1
2
3
4
5
6
7
8
9
10
Number of mutator threads
1
2
3
4
5
6
7
8
bdw
whippet


Collector scalability for mt-gcbench at 2x heap, 1 mutator
Time (s)
Number of marker threads
bdw
whippet
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
2
3
4
5
6
7
8


Collector scalability for mt-gcbench at 2x heap, 1 mutator
Time (s)
0.56
0.52
0.48
0.44
0.4
0.36
0.32
0.28
0.24
0.2
0.16
0.12
8e-2
4e-2
0
1
2
3
4
Number of marker threads
bdw
whippet

## Whippet vs BDW: Ephemerons

BDW: No ephemerons (link)

Whippet: Yes

## Whippet vs BDW: Precision

BDW: ~Always stack-conservative, often heap-conservative

Whippet: Fully configurable (at compile-time)

Guile in mid/near-term: C-stack-consrvative, Scheme stack precise, heap-precise

Possibly fully precise: unlock semi-space nursery

## Whippet vs BDW: Tools?

Can build heap tracers and profilers moer easily

More hackable

(BDW-GC has as many preprocessor directives as whippet has source lines)

## Engineering Whippet

Embed-only, abstractions, migration, modern; timeline

## Engineering Whippet: Embed-only

`https://github.com/wingo/whippet-gc/`

Semi: 6 kB; Whippet: 22 kB; BDW: 184 kB

Compile-time specialization:

- ❧ for embedder (e.g. how to forward objects)
- ❧ for selected GC algorithm (e.g. semi-space vs whippet)

Built apart, but with LTO to remove library overhead

## Engineering Whippet: Abstract performance

User API abstracts over GC algorithm, e.g. semi-space or whippet

Expose enough info to allow JIT to open-code fast paths

Inspired by `https://mmtk.io`

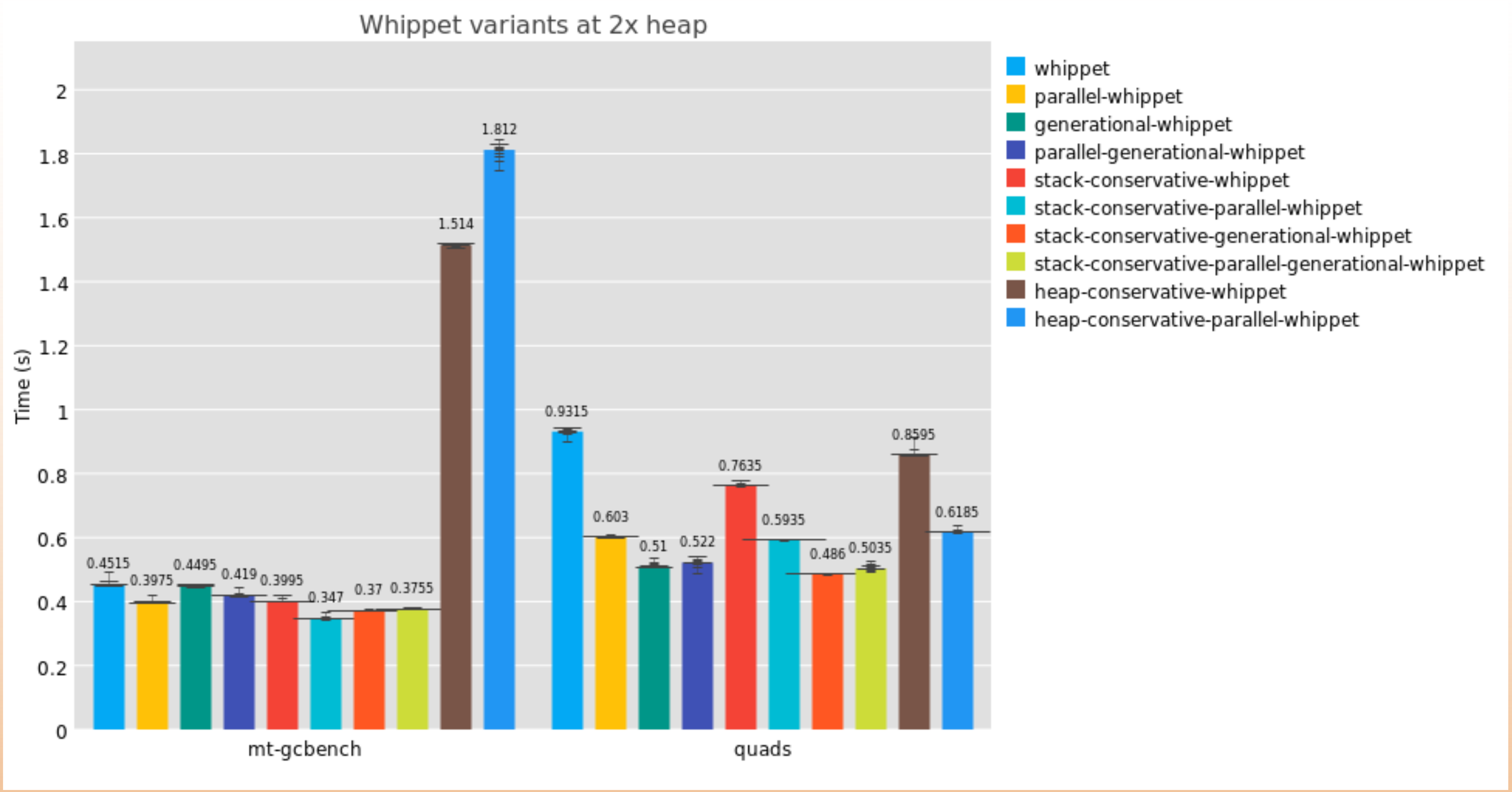
Abstractions permit change: of algorithm, over time

## Engineering Whippet: Migration

API implementable by BDW-GC (except ephemerons)

First step for Guile: BDW behind Whippet API

Then switch to whippet/immix (by default)


Whippet variants at 2x heap
Time (s)
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
mt-gcbench
quads
0.4515
0.3975
0.4495
0.419
0.3995
0.347
0.37
0.3755
1.514
1.812
0.9315
0.603
0.51
0.522
0.7635
0.5935
0.486
0.5035
0.8595
0.6185
whippet
parallel-whippet
generational-whippet
parallel-generational-whippet
stack-conservative-whippet
stack-conservative-parallel-whippet
stack-conservative-generational-whippet
stack-conservative-parallel-generational-whippet
heap-conservative-whippet
heap-conservative-parallel-whippet

## Engineering Whippet: Modern

stdatomic

`constexpr`-ish

pthreads (for parallel markers)

No `void*`; instead `struct` types: `gc_ref`, `gc_edge`, `gc_conservative_ref`, etc

Embed-only lib avoids any returns-struct-by-value ABI issue

Rust? MMTk; supply chain concerns

Platform abstraction for conservative root finding

## Engineering Whippet: Timeline

As time permits

Whippet TODO: heap growth/ shrinking, finalizers, safepoint API

Guile TODO: safepoints; heap-conservative first

Precise heap TODO: `gc_trace_object`, SMOBs, user structs with raw ptr fields, user `gc_malloc` usage; 3.2

6 months for 3.1.1; 12 for 3.2.0 ?

## Whippet: A Better GC?

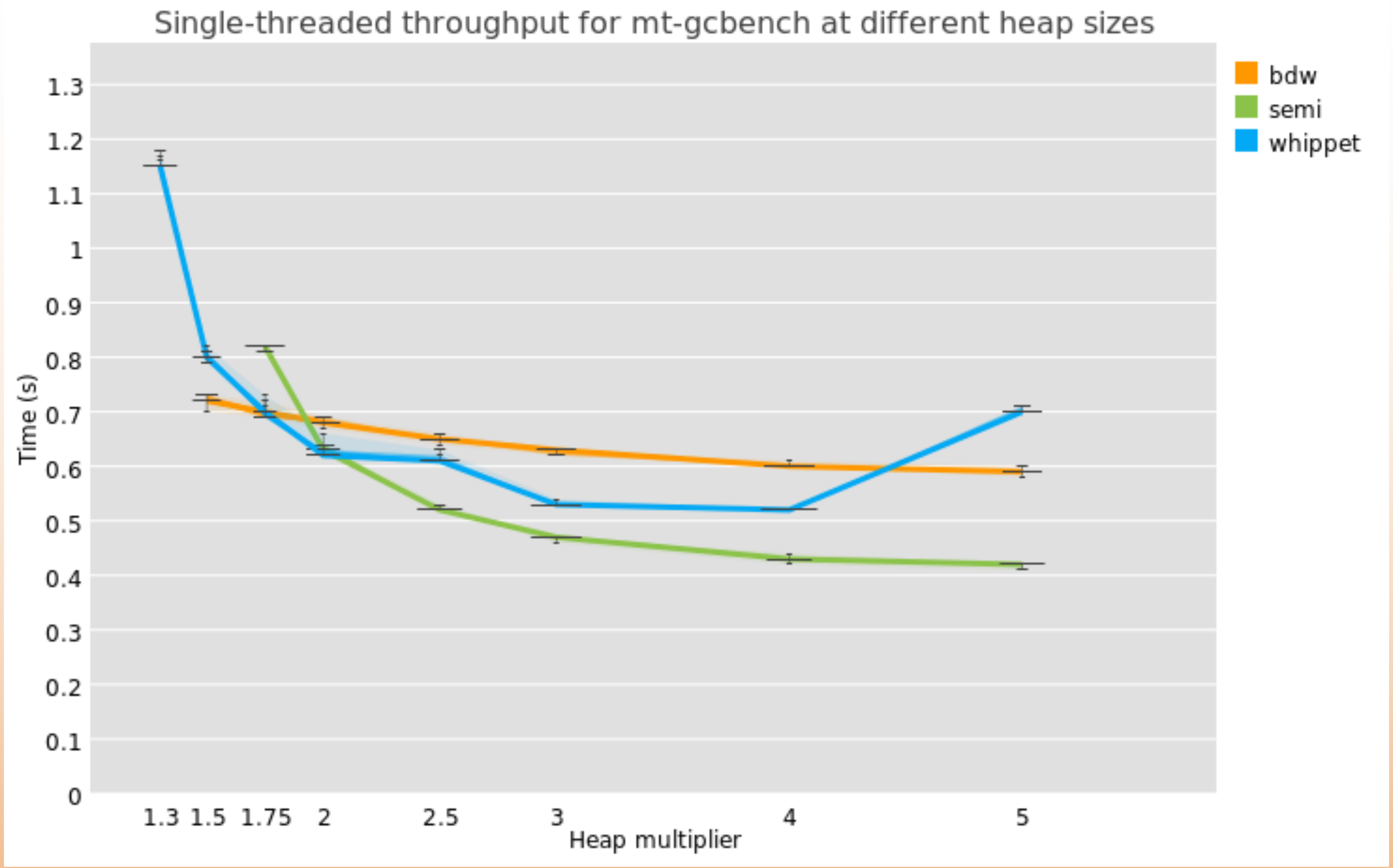
An Immix-derived GC

`https://github.com/wingo/whippet-gc/`

`https://wingolog.org/tags/gc/`

Guile 3.2 ?

Thanks to MMTk authors for inspiration!


Single-threaded throughput for mt-gcbench at different heap sizes
Time (s)
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1.3
1.5
1.75
2
2.5
3
4
5
Heap multiplier
bdw
semi
whippet