Wastrel: WebAssembly sans runtime

1 February 2026 – FOSDEM ’26

Andy Wingo

Igalia, S.L.

Demo

Storytime

Longtime Guile co-maintainer

Guile uses BDW-GC; very reliable but suboptimal

- Bad multi-threaded mutator perf
- High baseline memory usage
- Can’t compact

Always wanted to write a gc

Heard of Immix (2008): GC algorithm that can evacuate or leave objects in place

In which a Whippet

An embeddable GC library inspired by Immix

Vendor it in your C projects

No deps

Specialized to needs of embedder at compile-time

- Object representation
- Tracing procedures
- Root finding

Collection of collectors

Whippet demo

```
wastrel compile --gc=help
```

Whippet's collector collection

`pcc`: Parallel copying collector (possibly generational not)

`mmc`: Mostly-marking collector (possibly generational, conservative...)

`bdw`: Boehm-Demers-Weiser (BDW) (conservative)

Why Whippet

Pitch: `mmc` improves upon `bdw`

- access tighter heaps
- scale better for multiple mutator threads
- can compact

Thanks to NLnet, Igalia for support

`https://nlnet.nl/project/Whippet/`

`https://arxiv.org/abs/2503.16971`

Whippet Where

Whippet was made for Guile

`wip-whippet`, ~350 commits, April to August 2025

Needs incremental merging via Codeberg, for Guile 4.0 push

Strategy:

- Rebase on Whippet API; `bdw`
- heap-conservative `mmc`
- heap-precise `mmc`

Talk tomorrow at Guix days?

Wastrel

Remember Wastrel

This a talk about Wastrel

Wastrel

Wastrel is a Wasm-to-C compiler, with Whippet

Wasm-to-C

Prior art: `wasm2c`, `w2c2`

Core compiler builds on Guile Hoot's Wasm libs

Performance: Best-in-class

Memory safety: 8 GB `PROT_NONE` region

Aside: C best practices

Lean into type system and optimizer

Never cast/convert implicitly. rarely cast at all

Use memcpy to read and write

Static inline functions, data structs

Avoid bare integers (`uintptr_t`, `int`, etc)

If you do it right, generated code bears proof of behavior embodied by source wasm types; GCC checks your work

(also `-Wconversion`, UBSAN, etc)

Standard library

```
clang --target=wasm32-wasip1 -o main main.c
```

Requires WASI-SDK

Implementing WASI in Wastrel: 2 weeks. Awful. Aggregate data encoding terrible, have to be very careful about memory reads/writes

Filesystem sandbox via Linux namespaces

GC

Wasm 1.0: Linear memory

Wasm 3.0: Linear memory + exceptions + managed memory + …

```
(type $pair (struct (ref eq) (ref eq)))
(func $cons (param $a (ref eq))
            (param $b (ref eq))
            (result (ref $pair))
  (struct.new $pair
    (local.get $a) (local.get $b)))
```

Arrays, structs, subtyping, type recursion, typed function references, 31-bit immediates, static and dynamic casts, memory safety

Adding GC to Wastrel

Hell: difficult to translate Wasm types to C in presence of subtyping, three type lattices.

Heaven too: generated C carries types from source.

Base types

```
// Assume typedefs, e.g. "typedef struct anyref {...} anyref"
struct anyref { uintptr_t value; };
struct eqref { anyref p; };
struct i31ref { eqref p; };
struct arrayref { eqref p; };
struct structref { eqref p; };

struct externref { uintptr_t value; };

struct funcref { void *value; uint32_t type; };

struct wasm_obj { uintptr_t tag_word; };
struct wasm_array { struct wasm_obj p; uint32_t len; };
struct wasm_struct { struct wasm_obj p; };
```

Base types

```
// Assume static inline
int is_null(anyref ref) { return ref.value == 0; }
int is_imm(anyref ref) { return (ref.value & 1) == 1; }

wasm_obj* any_obj(anyref ref) {
  assert(!is_null(ref) && !is_imm(ref));
  return (wasm_obj*) ref.value;
}
```

Let’s step through generated C

```
(type $type_0
  (struct i32))

(rec
 (type $type_1
   (sub $type_0 (struct i32 (ref $type1)))))
```

Generated types; tags assigned via depth-first search

```
struct type_0ref { structref p; };
struct type_1ref { type_0ref p; };

struct type_0 { wasm_struct p; int32_t f0; };
struct type_1 { type_0 p; type_1ref f1; };

int is_type_0(anyref ref) {
  wasm_obj *obj = any_obj(ref);
  uint32_t tag = ((uint32_t) obj->tag_word) >> 1;
  return 0 <= tag && tag < 2;
}
```

No handles; stack/registers conservatively traced, pinned

```
static type_1ref pack_type_1(type_1 *obj) {
  // Oh yeah baby
  return (type_1ref){{{{{(uintptr_t)obj}}}}};
}

static type_1ref
make_type_1(struct gc_mutator *mut, int32_t f0, type_1ref f1) {
  type_1 *ret =
    gc_allocate(mut, sizeof(type_1), GC_ALLOCATION_TAGGED);
  ret->p.p.p.tag_word = 1 << 1;
  ret->p.f0 = f0;
  ret->f1 = f1;
  return pack_type_1(ret);
}
```

Capabilities

Ad-hoc selection of stdlib imports (e.g. `(import "debug" "debug_str")`);

Going to build out Hoot stdlib shortly: bignums, etc

Wastrel can implement extensions, e.g. stringref or stringref support

Can still access WASI capability, but impedance mismatch with linear memory

Status

Demo-ware

`codeberg.org/andywingo/wastrel/issues/`

What's next?

Happy to implement more stdlib for OCaml, etc

Guile-to-native

Benchmarking: prove mostly-marking collector performance against real programs

Feature completeness (e.g. exceptions)

Fix to enable FS sandbox with GC

Implement new Wasm standards (stack switching!!!)

Help evolve Wasm standards

Call to action™

Writing a new language? Consider targetting WasmGC!

- `wingolog.org/archives/2014/11/27/scheme-workshop-2014/`

Targetting WasmGC? Let's make native binaries together!

- `codeberg.org/andywingo/wastrel/`

Need a GC? Let's talk!

- `github.com/wingo/whippet/`