Celebrating Guile 3

FOSDEM 2020, Brussels

Andy Wingo | wingo@igalia.com

wingolog.org | @andywingo

Lessons Learned from Guile, the Ancient & Spry

FOSDEM 2020, Brussels

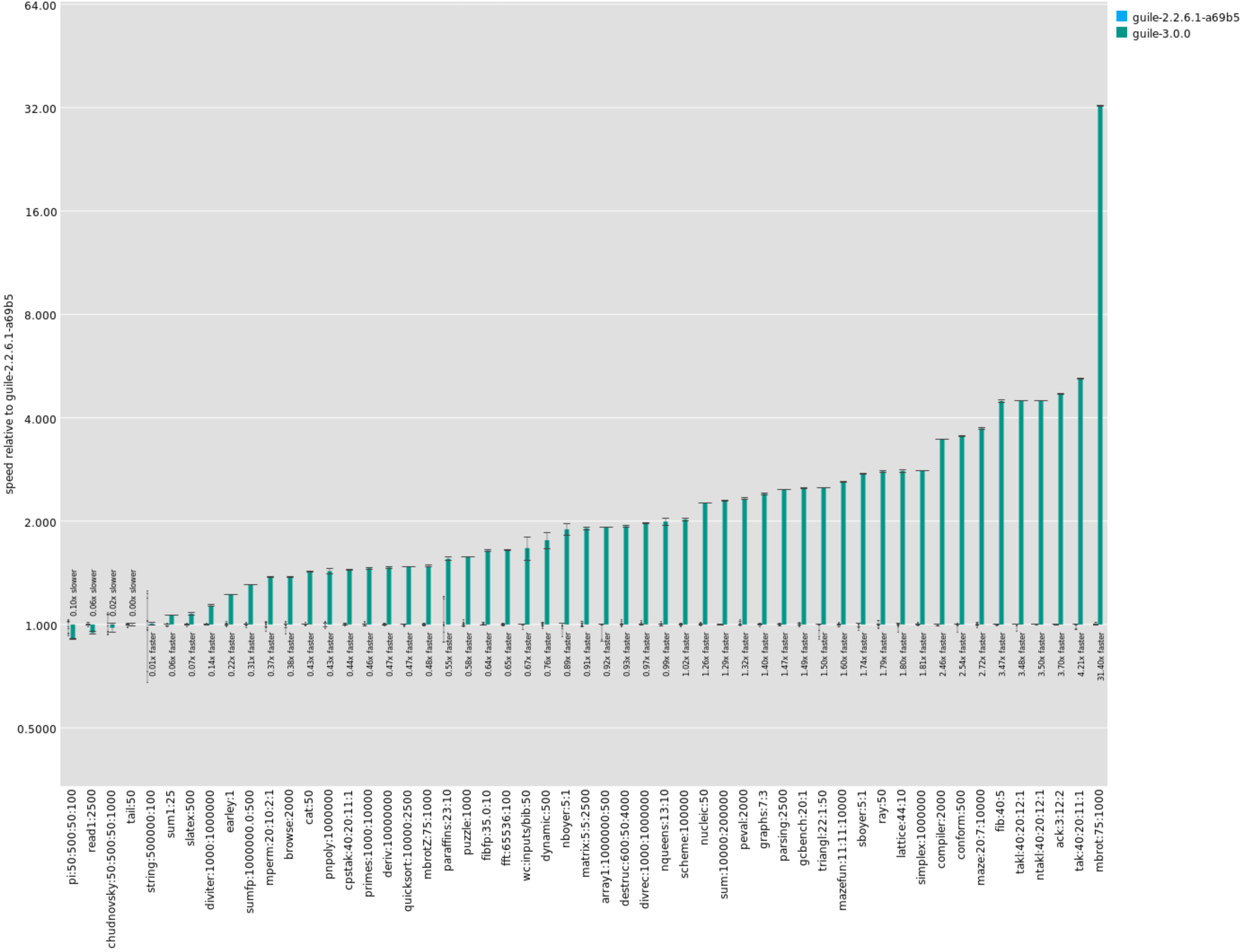
Andy Wingo | wingo@igalia.com

wingolog.org | @andywingo

spry /*sprī*/

- adjective: active; lively

Speedups in Guile 3: Microbenchmarks

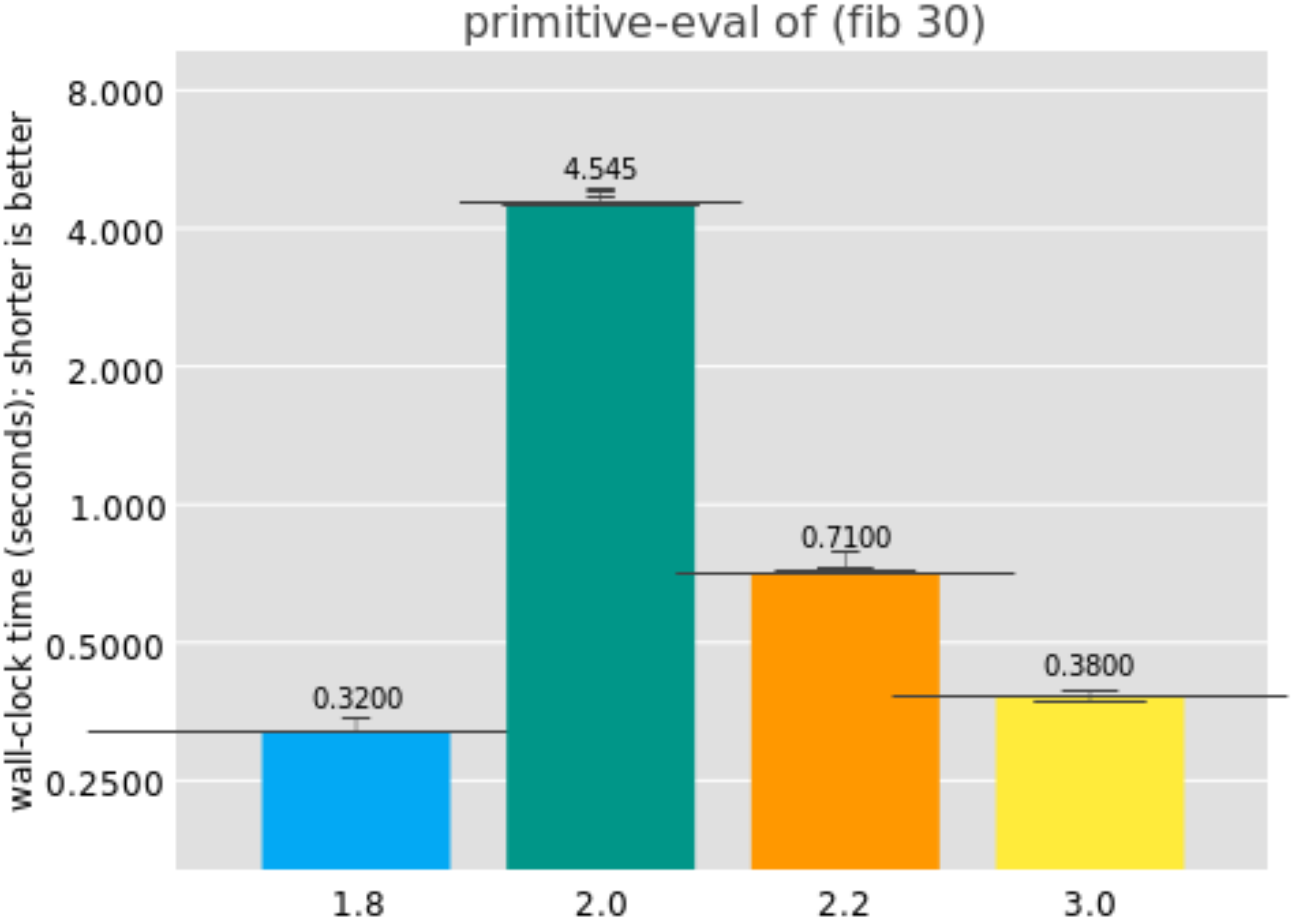


mini-benchmark: eval

```
(primitive-eval
 '(let fib ((n 30))
    (if (< n 2)
        n
        (+ (fib (- n 1)) (fib (- n 2))))))
```

Guile 1.8: `primitive-eval` written in C

Guile 2.0+: `primitive-eval` in Scheme


primitive-eval of (fib 30)
wall-clock time (seconds); shorter is better
8.000
4.000
2.000
1.000
0.5000
0.2500
4.545
0.7100
0.3200
0.3800
1.8
2.0
2.2
3.0

macro-benchmark: guix

```
guix build libreoffice ghc-pandoc guix \
  --dry-run --derivation
```

7% faster

```
guix system build config.scm \
  --dry-run --derivation
```

10% faster

spry /*sprī*/

- adjective: (especially of an old person) active; lively

guile is ancient

2010: Rust

2009: Go

2007: Clojure

1995: Ruby

1995: PHP

1995: JavaScript

1993: Guile (3^3 years before 3.0!)

built from ancient parts

1991: Python

1990: Haskell

1990: SCM

1989: Bash

1988: Tcl

1988: SIOD

written in an ancient language

1987: Perl

1984: C++

1975: Scheme

1972: C

1958: Lisp

1958: Algol

1954: Fortran

1958: Lisp

1930s: λ-calculus (3^4 years ago!)

ancient & spry

Men make their own history, but they do not make it as they please; they do not make it under self-selected circumstances, but under circumstances existing already, given and transmitted from the past.

The tradition of all dead generations weighs like a nightmare on the brains of the living. […]

Eighteenth Brumaire of Louis Bonaparte, Marx, 1852

ancient & spry

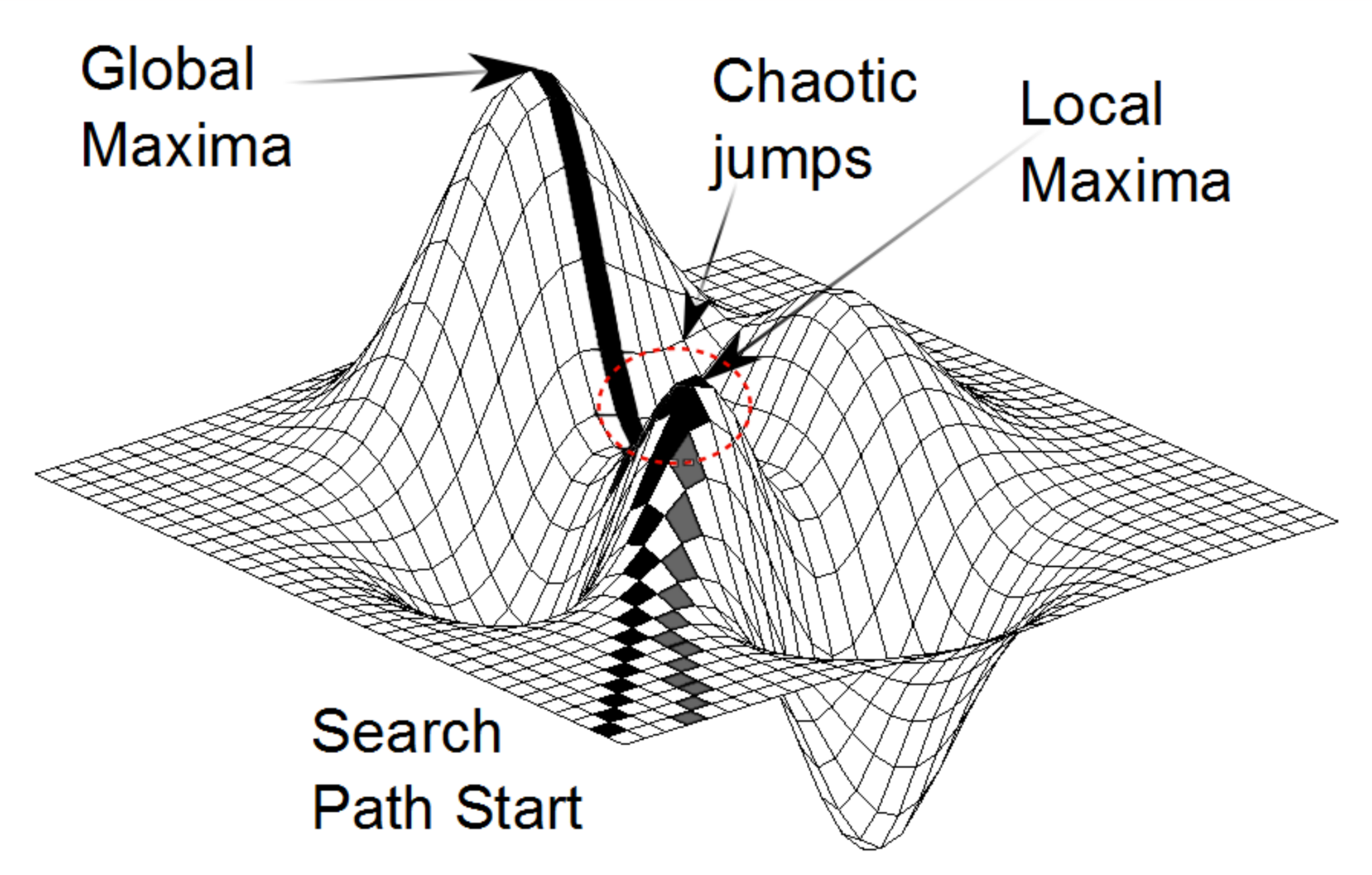
Languages evolve; how to remain **minimal**?

Dialectic opposites

- world and guile
- stable and active
- …

Lessons learned from inside Hegel’s motor of history

hill-climbing is insufficient



Ex: Guile 1.8; Extend vs Embed

users
stay
unless
pushed
away

Inertial factor: interface

- Source (API)
- Binary (ABI)
- Embedding (API)
- CLI
- …

Ex: Python 3; `local-eval`; R6RS syntax; `set!`, `set-car!`

you can’t keep all users

What users say: don’t change or remove existing behavior

But: sometimes losing users is OK. Hard to know when, though

No change at all == death

- ❧ Natural result of hill-climbing

Ex: `psyntax`; BDW-GC mark & finalize; compile-time; Unicode / locales

every interface is a cost

Guile binary ABI: libguile.so; compiled Scheme files

Make compatibility easier: **minimize** interface

Ex: `scm_sym_unquote`, GOOPS, Go, Guix

parallel installs for the win

Highly effective pattern for change

- `libguile-2.0.so`
- `libguile-3.0.so`

`https://ometer.com/parallel.html`

Changed ABI is new ABI; it should have a new name

Ex: `make-struct/no-tail`, `GUILE_PKG([2.2])`, libtool

deprecation facilitates migration

```
__attribute__ ((__deprecated__))
```

```
(issue-deprecation-warning
 "(ice-9 mapping) is deprecated."
 "  Use srfi-69 or rnrs hash tables instea
```

```
scm_c_issue_deprecation_warning
  ("Arbiters are deprecated.  "
   "Use mutexes or atomic variables instea
```

```
begin-deprecated,
SCM_ENABLE_DEPRECATED
```

the arch-pattern

Replace, Deprecate, **Remove**

All change is possible; question is only length of deprecation period

Applies to all interfaces

Guile deprecation period generally one stable series

Ex: `scm_t_uint8`; `make-struct`; Foreign objects; uniform vectors

change produces a new stable point

Stability within series: only additions

Corollary: dependencies must be at least as stable as you!

- ❧ for your definition of stable
- ❧ social norms help (GNU, semver)

Ex: libtool; unistring; gnulib

who can crank the motor of history?

All libraries define languages

Allow user to evolve the language

- User functionality: modules (Guix)
- User syntax: macros (yay Scheme)

Guile 1.8 perf created tension

- incorporate code into Guile
- large C interface “for speed”

Compiler removed pressure on C ABI

Empowered users need less from you

contributions and risk

From maintenance point of view, all interface is legacy

Guile: Sometimes OK to accept user modules when they are more stable than Guile

In-tree users keep you honest

Ex: SSAX, fibers, SRFI

sticky bits

Memory management is an ongoing thorn

Local maximum: Boehm-Demers-Weiser conservative collector

How to get to precise, generational GC?

Not just Guile; e.g. CPython `__del__`

future

We are here: stability

And then?

- Parallel-installability for source languages: `#lang`
- Sediment idioms from Racket to evolve Guile user base

Remove myself from “holding the crank”

dialectic, boogie woogie woogie

`https://gnu.org/s/guile`

`https://wingolog.org/`

`#guile` on freenode

`@andywingo`

`wingo@igalia.com`

Happy hacking!