Evaluating the Nix Evaluator

Why Nix Performance Sometimes… Doesn’t

Connor Baker

2025-03-07

Planet Nix

Topics covered

- Benchmarking setup
- Nix evaluation performance over time
- Suggested areas for improvement

Assumptions 📕

Nix evaluation performance

Nix evaluation performance

1. Can improve?

Nix evaluation performance

1. Can improve?
   - Historically, yes!

Nix evaluation performance

1. Can improve?
   - Historically, yes!
2. Should improve?

Nix evaluation performance

1. Can improve?
    - Historically, yes!
2. Should improve?
    - It depends!

Benchmarking 🕐

Benchmarking is **difficult**.

What can we easily measure?

- Data reported by `NIX_SHOW_STATS`
  - CPU/GC time, number of certain operations, etc.
- Data reported by GNU `time`
  - IO: context switches, page faults, etc.
  - Memory: page size, maximum resident set size, etc.
  - Time: real, user, and sys time

benchmarking-nix-eval

- Allows matrixing Nix packages and configurations through flakes
- Runs `time nix eval` inside the sandbox n times
- Collects the results with some additional metadata into JSON
- Data is suitable for visualization with VegaLite

[1]

[1] https://github.com/ConnorBaker/benchmarking-nix-eval

This presentation uses **VegaLite** through **WASM** as a **Typst** package.

Examples 📊

Testbed setup

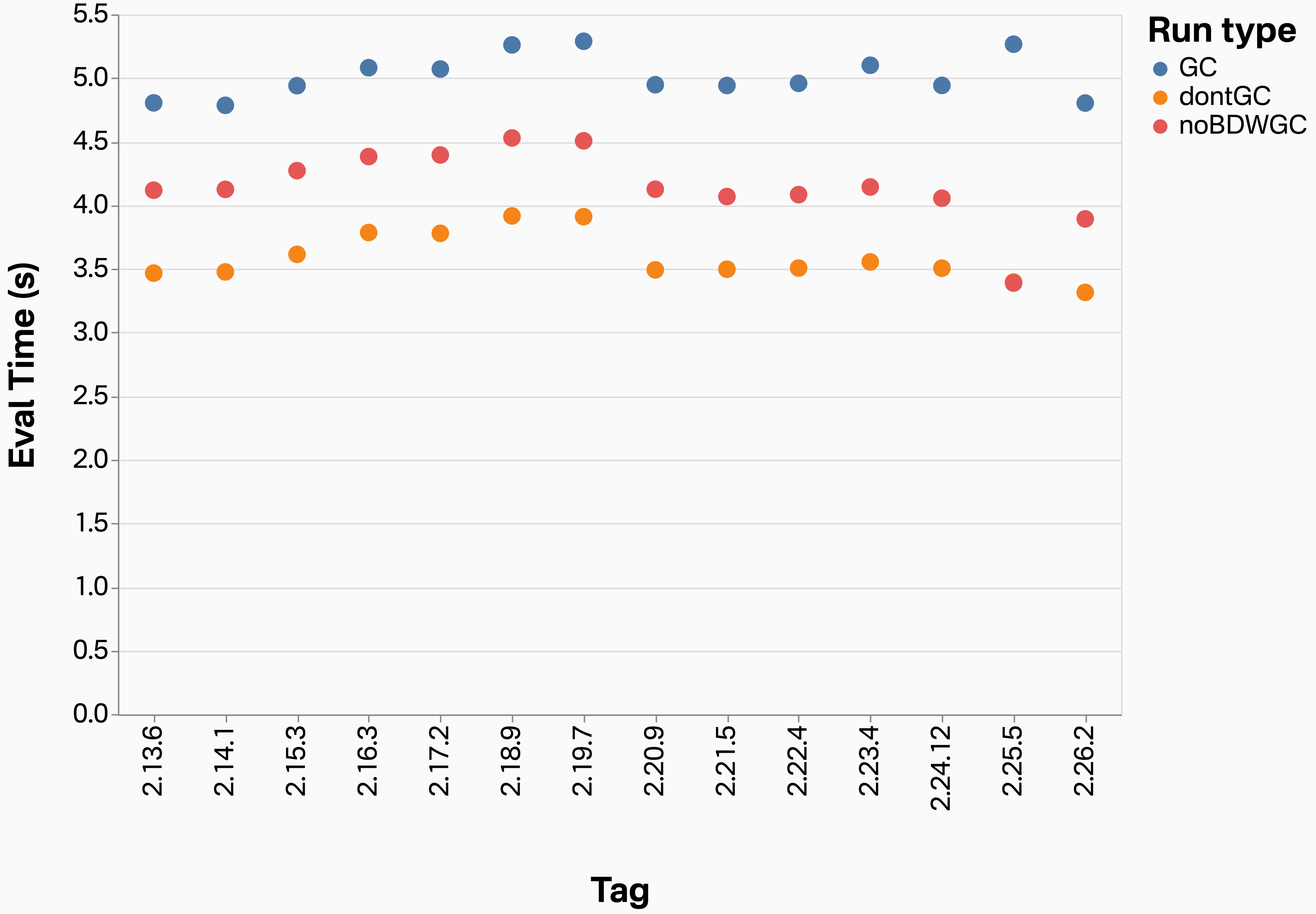
- Intel i9-13900K @ 3 GHz
  - Did not change niceness/pin to a favored core
- 96 GB DDR5 RAM
  - Did not attempt flushing caches
- Four-way ZFS RAID0
  - No deduplication/compression/integrity checking (just ARC)
  - Did not change IO niceness/flush caches
- Linux 6.12.13
- NixOS unstable @ `2ff53fe` (2025-02-13)
- `mimalloc` as the default allocator

Software setup

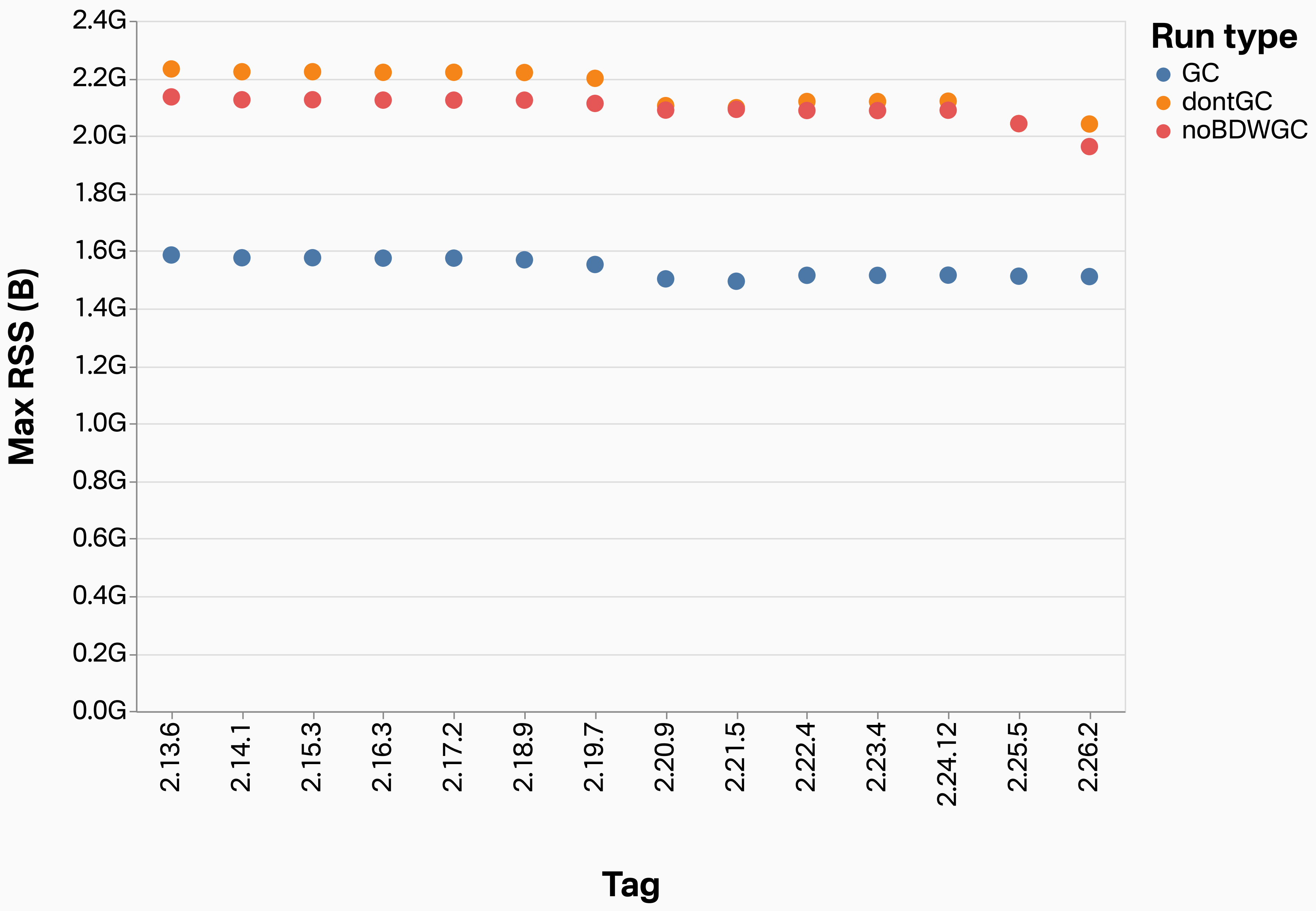
- Latest minor versions of Nix (2.13-2.26)
- 20 runs of each benchmark, one at a time, with and without GC
- Median values are plotted
  - Observed little variation between runs
- Generated data is available[1]

[1] https://github.com/ConnorBaker/benchmarking-nix-eval/releases/download/v0.0.1/aggregated-nixos-desktop-20-runs-1-job-no-boost.json

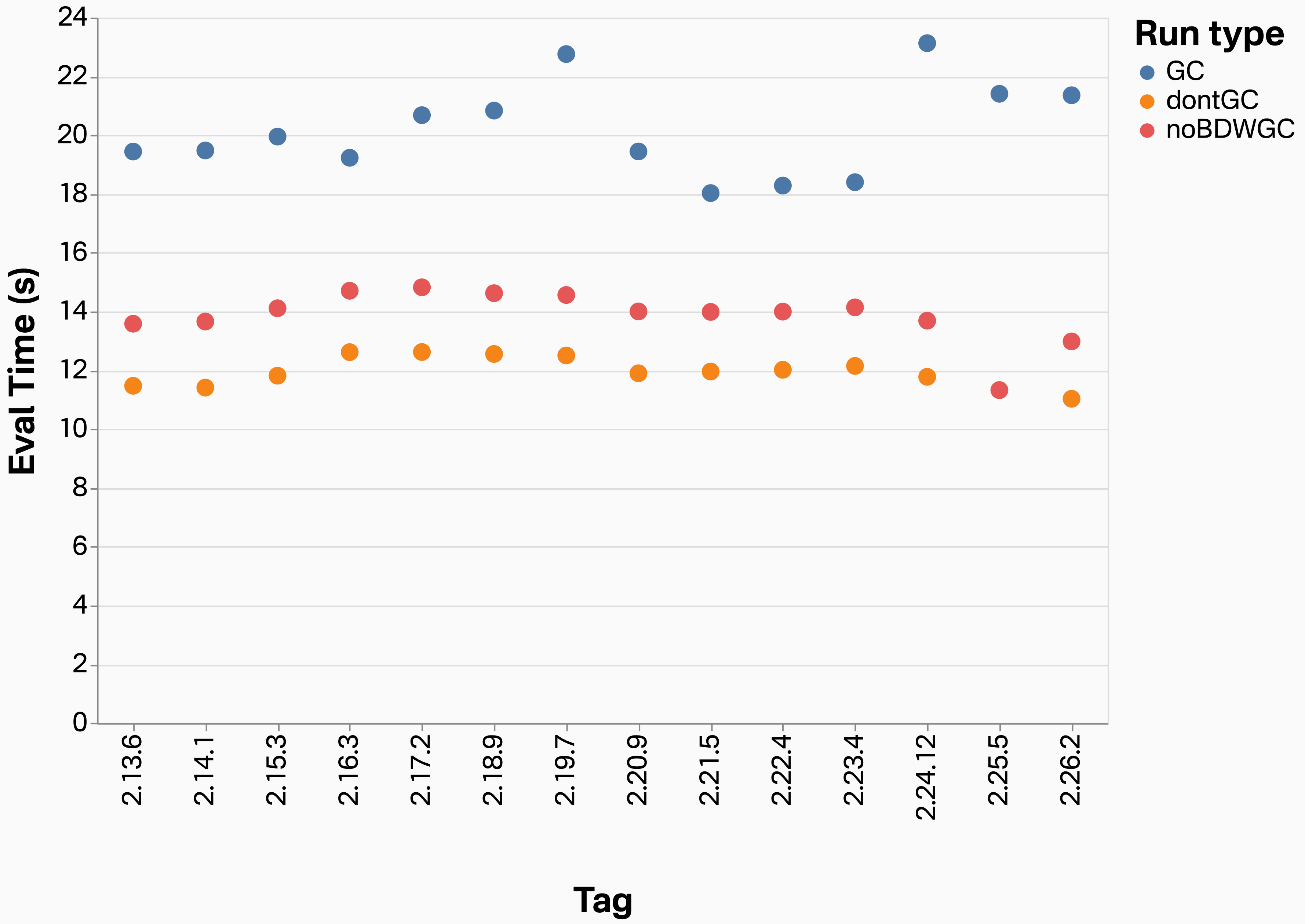
firefox-unwrapped eval time



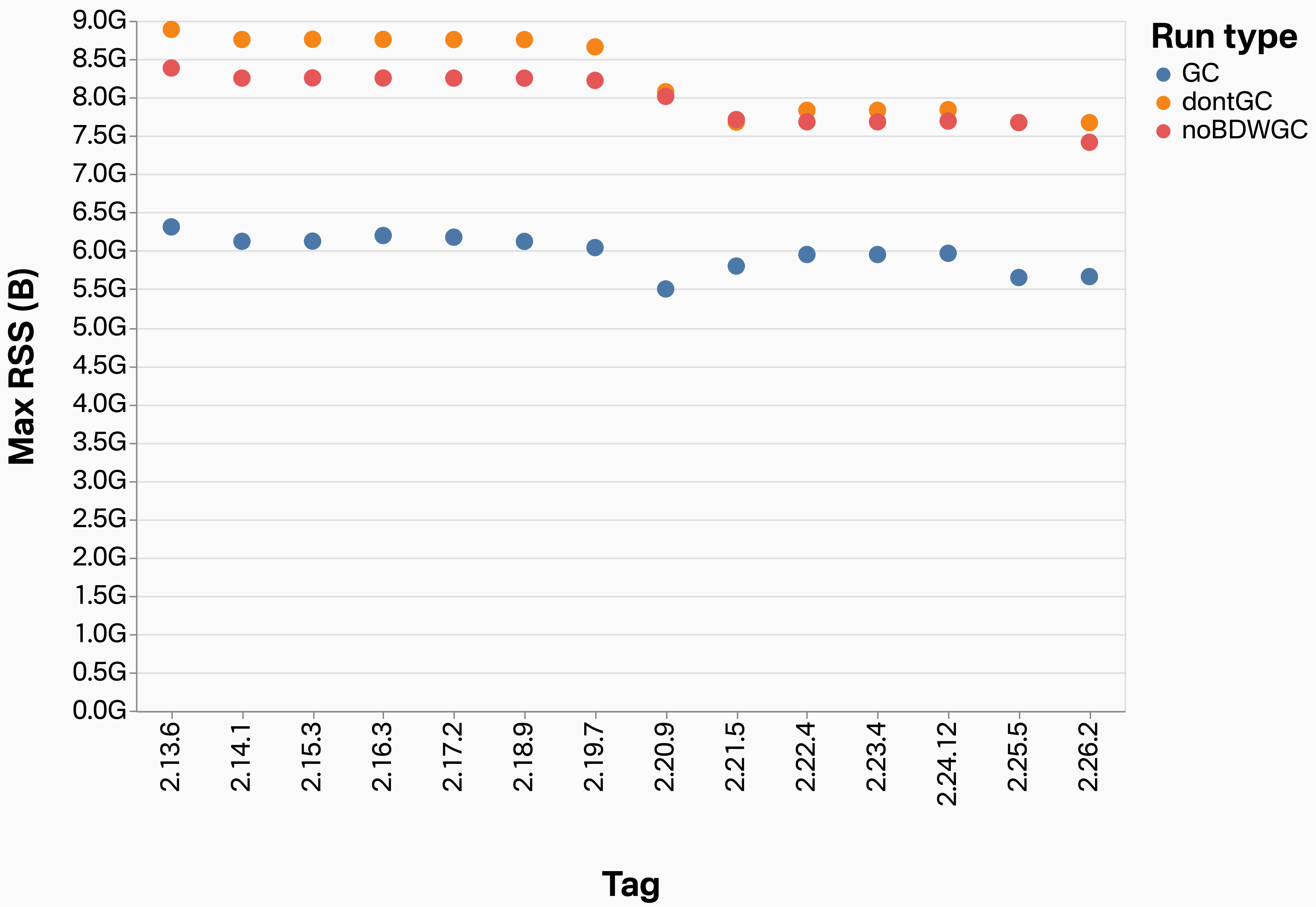
firefox-unwrapped eval space



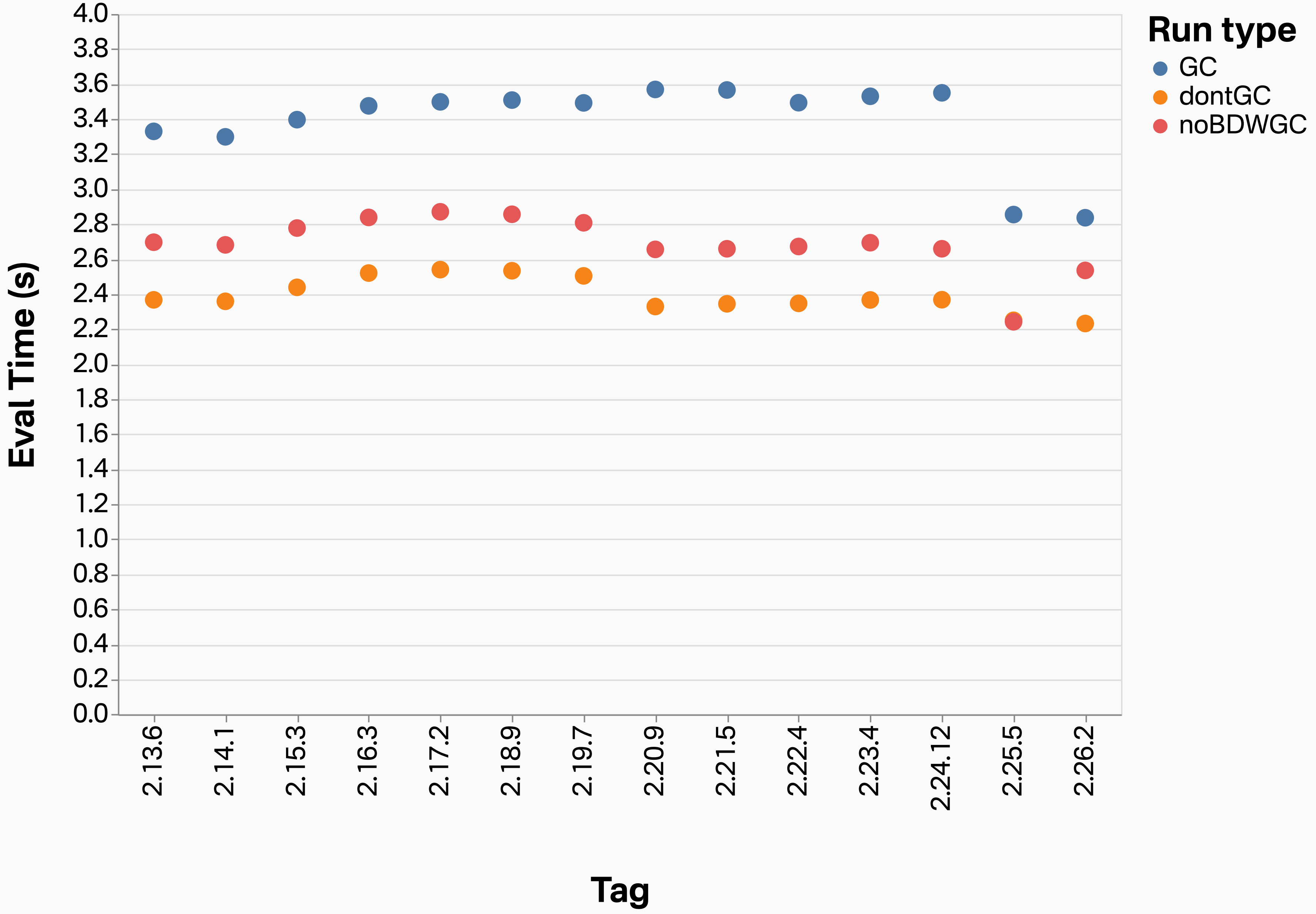
release-attrpaths-superset.names eval time



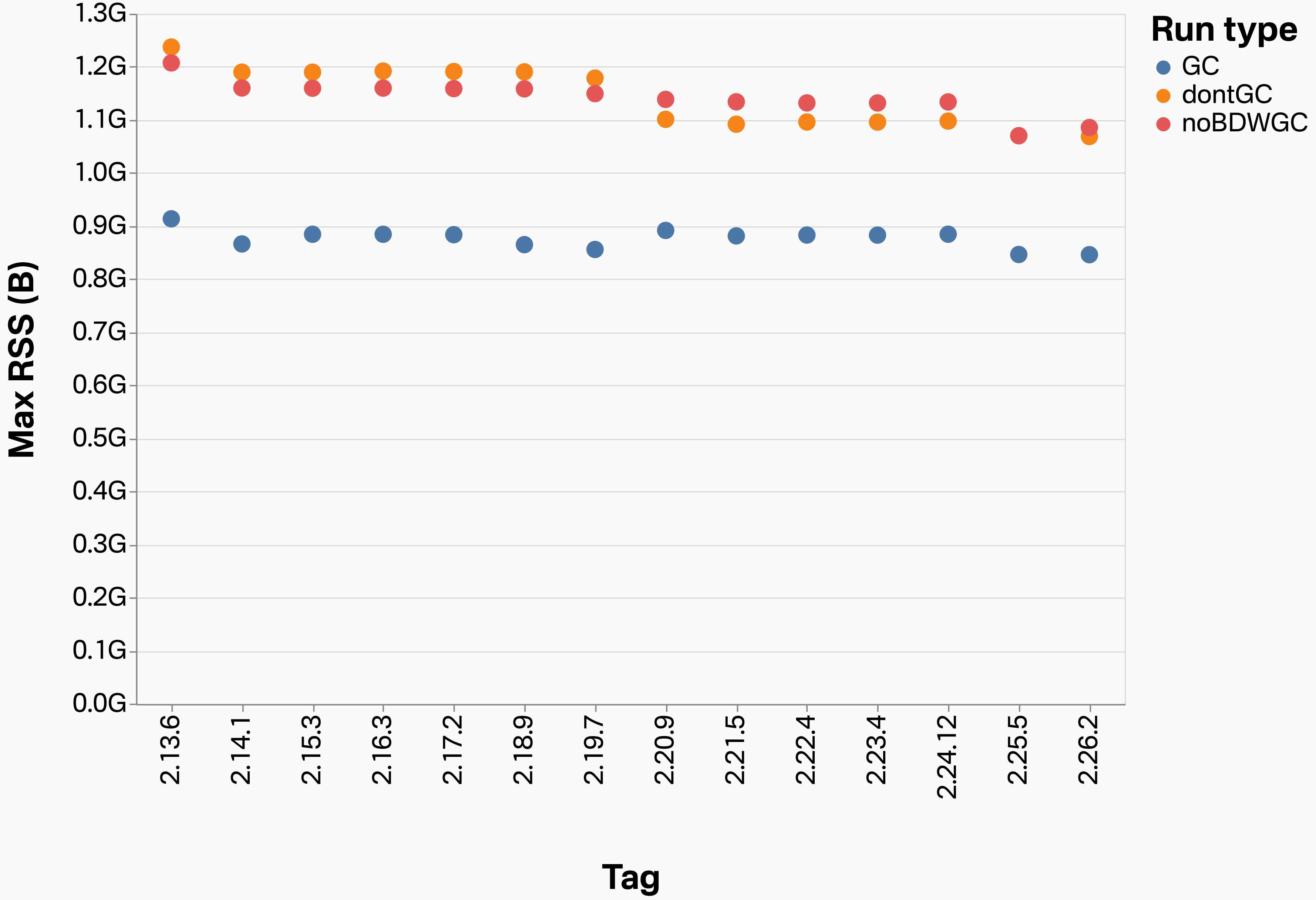
release-attrpaths-superset.names eval space



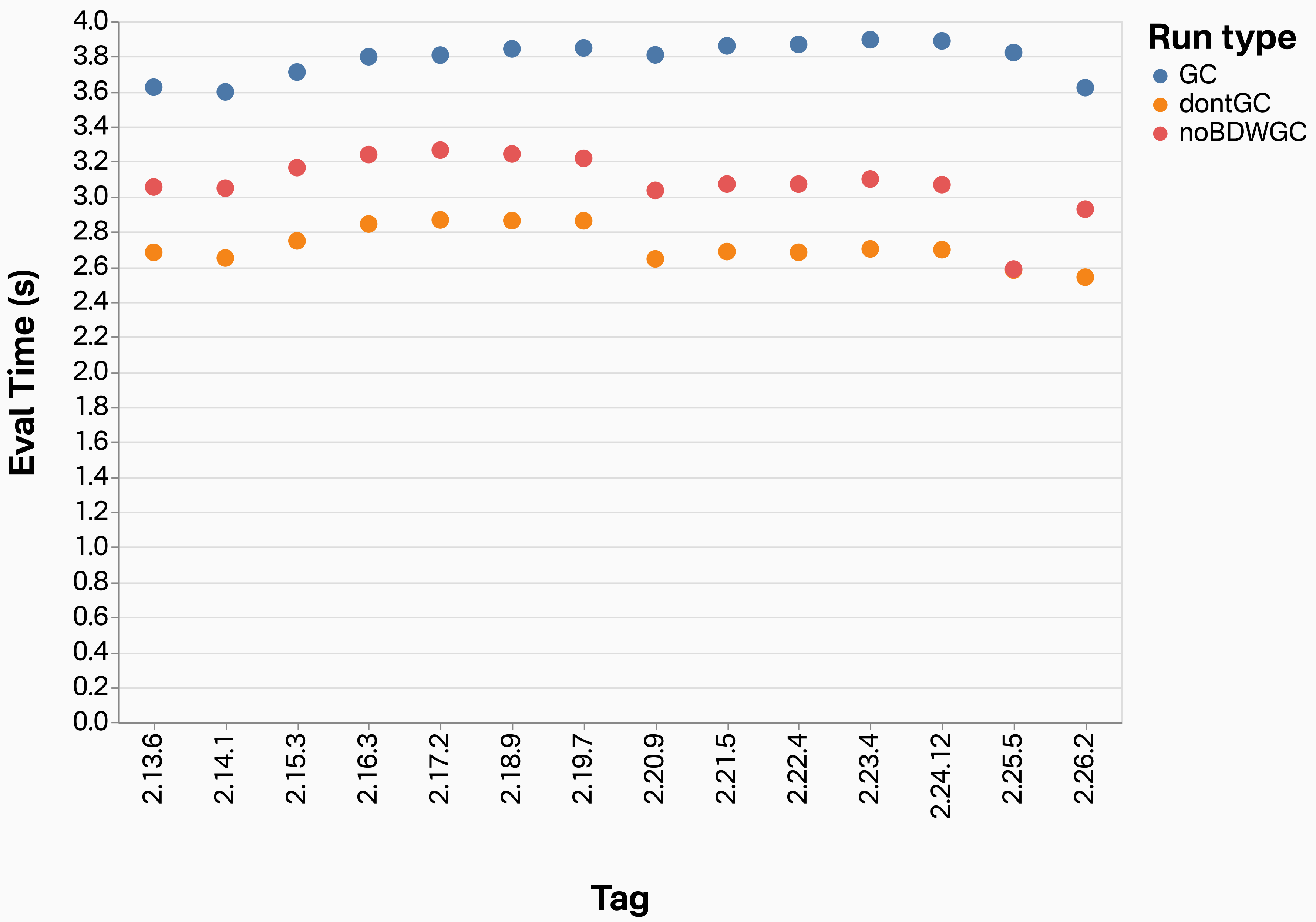
closures.smallContainer.x86_64-linux eval time



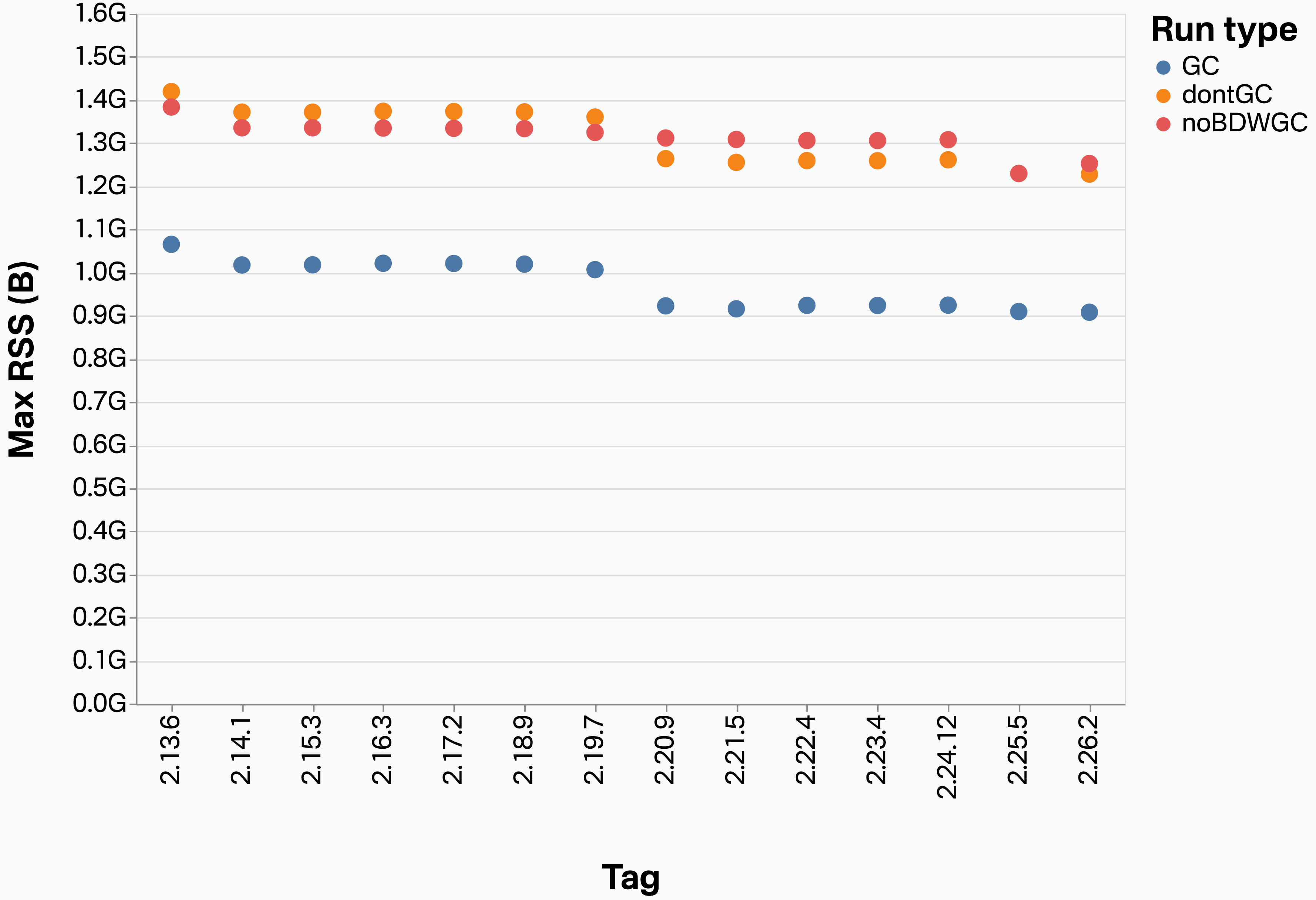
closures.smallContainer.x86_64-linux eval space



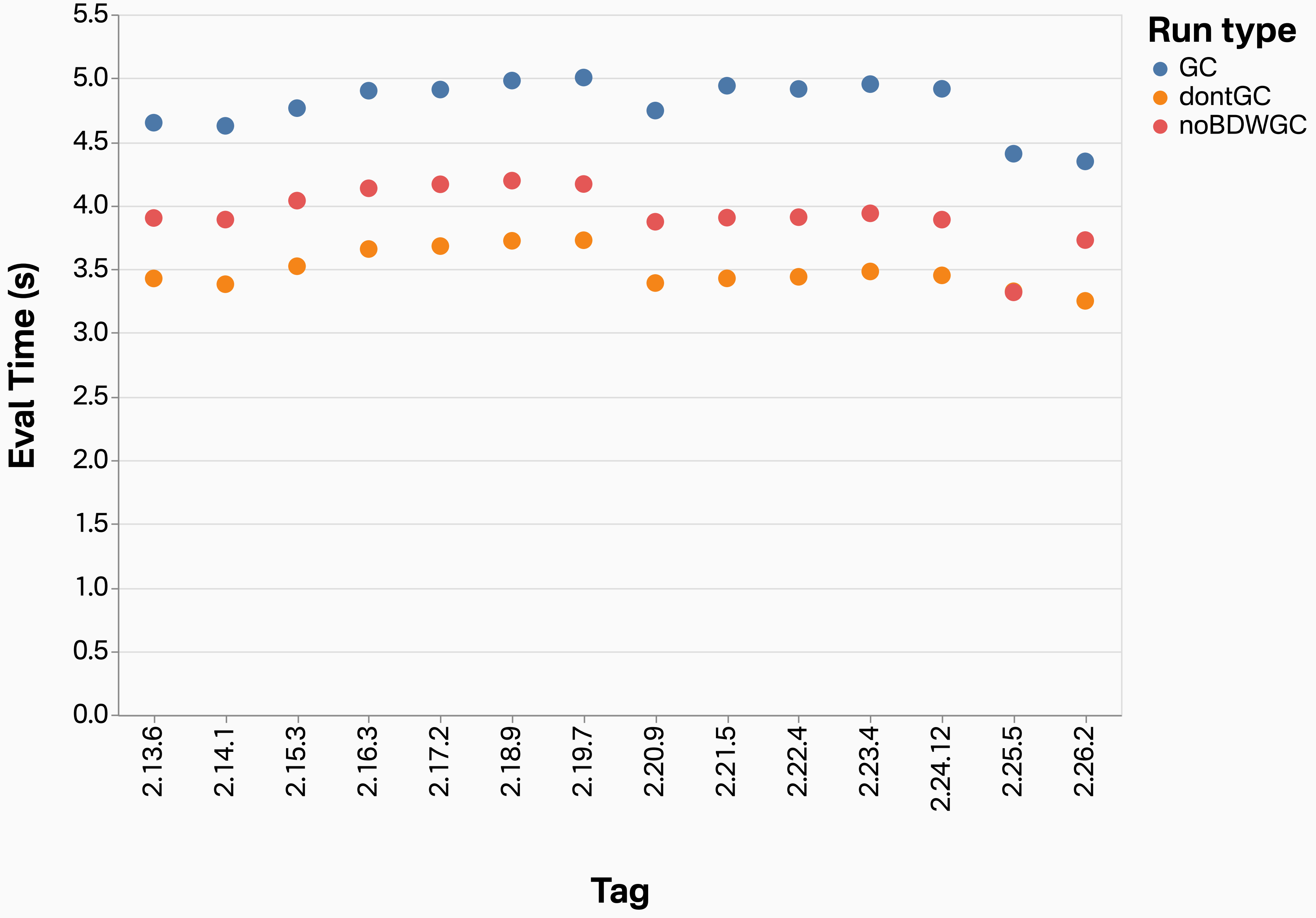
closures.lapp.x86_64-linux eval time



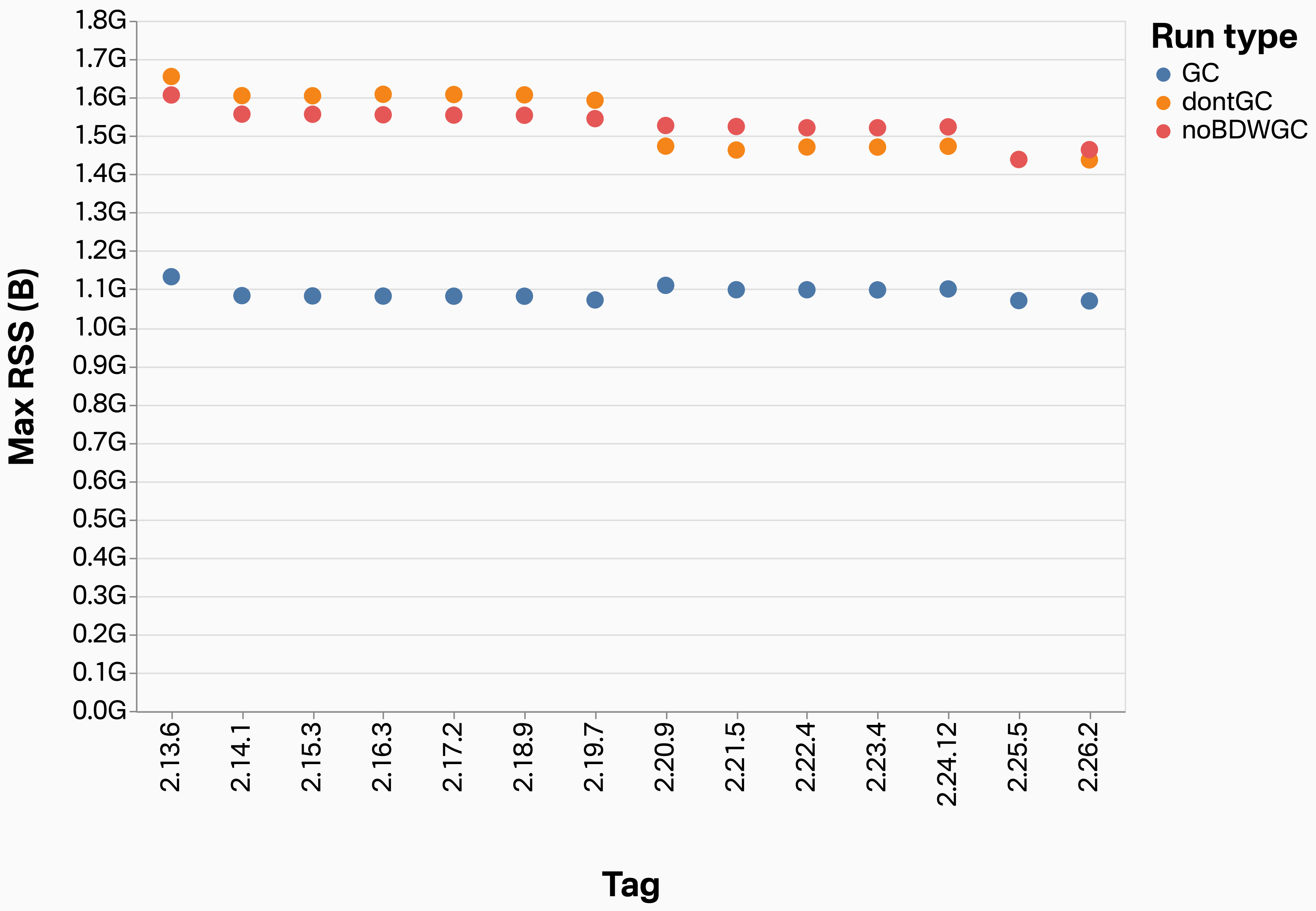
closures.lapp.x86_64-linux eval space



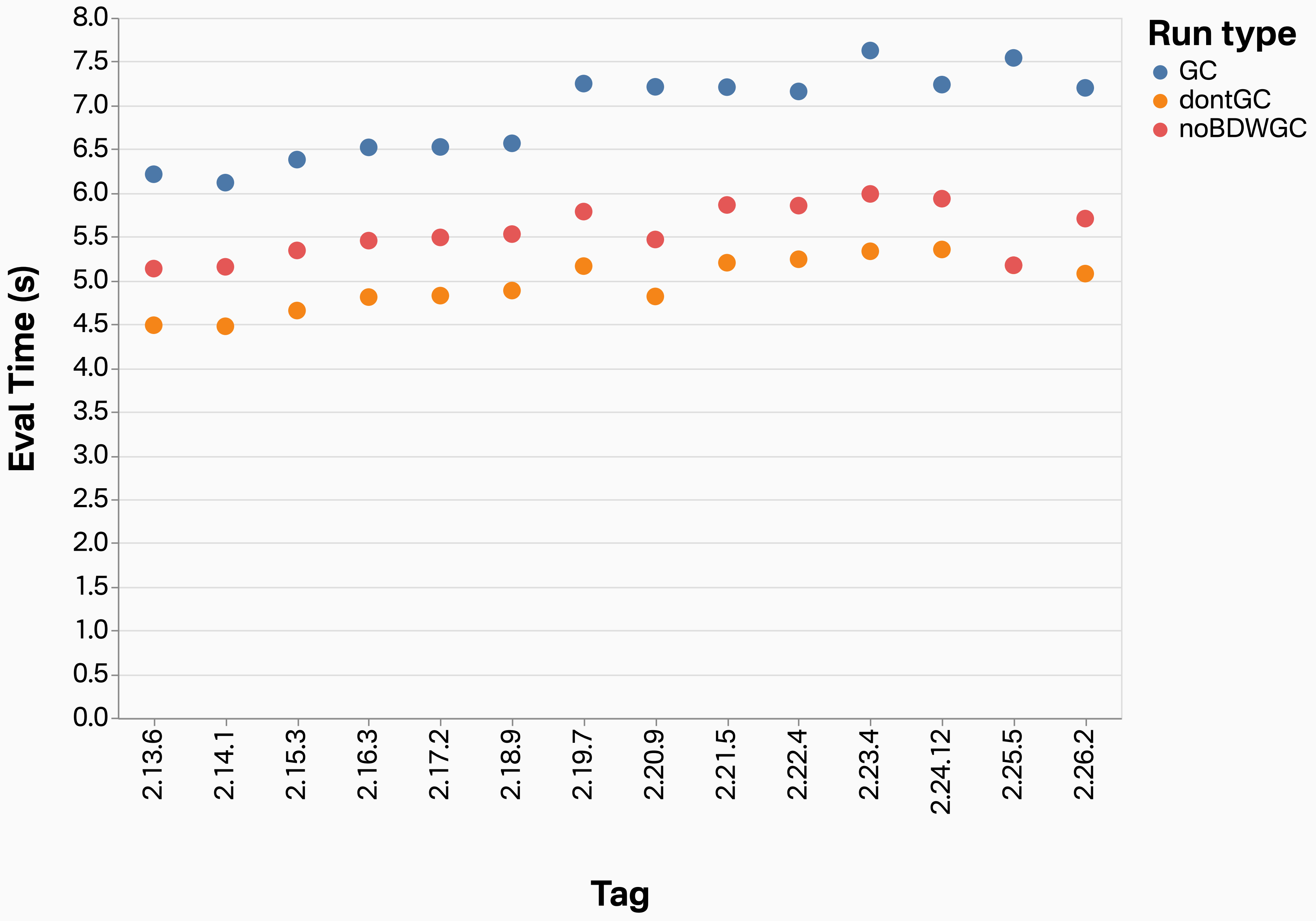
closures.kde.x86_64-linux eval time



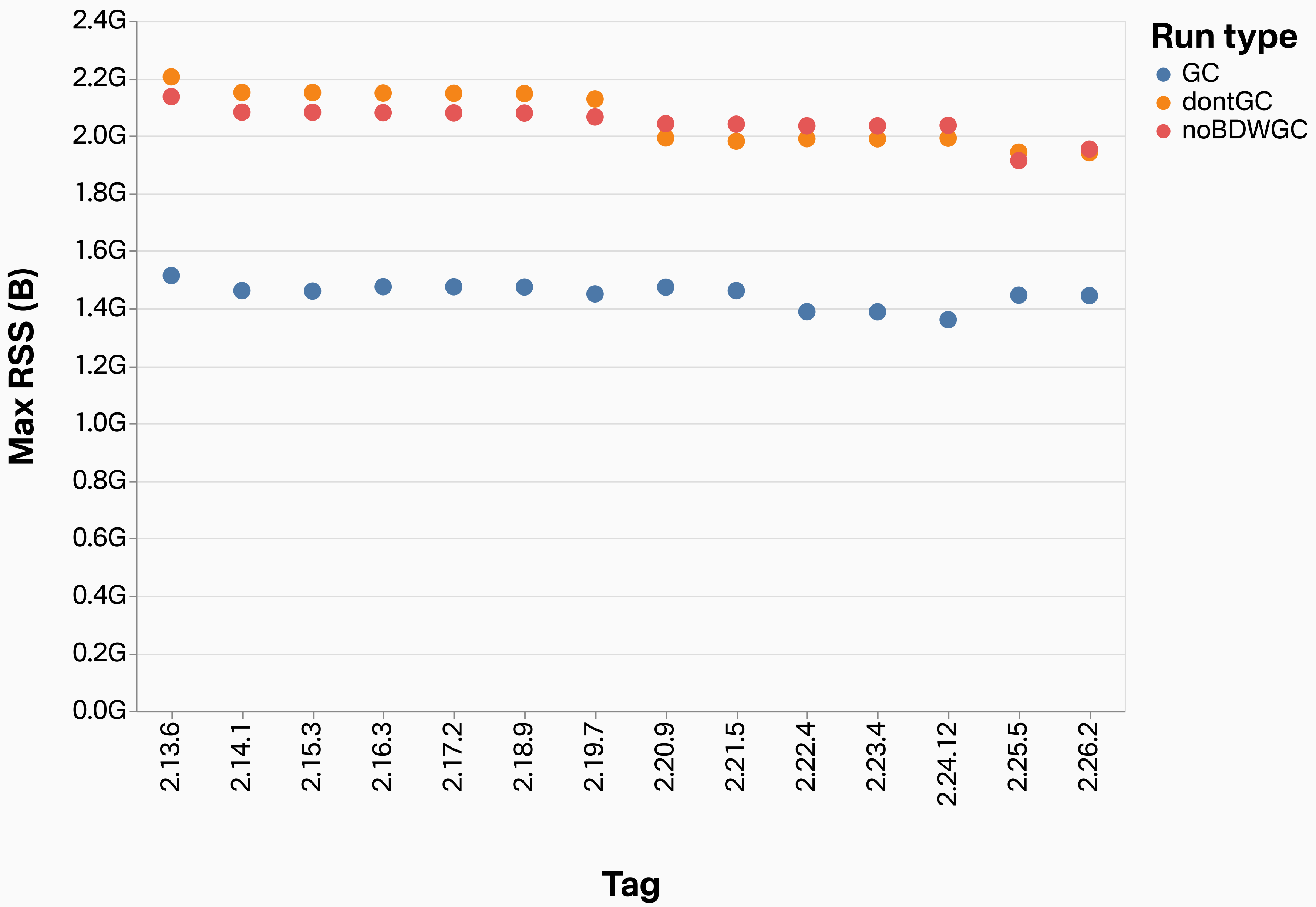
closures.kde.x86_64-linux eval space



iso_gnome.x86_64-linux eval time



iso_gnome.x86_64-linux eval space



Summary

- If you need faster evaluation, set `GC_DONT_GC`
  - `nix-eval-jobs` does this[1]
- `GC_DONT_GC` is faster than no BDWGC
  - BDWGC has

[1] https://github.com/nix-community/nix-eval-jobs/blob/4b392b284877d203ae262e16af269f702df036bc/src/nix-eval-jobs.cc#L421-L422

What's with all the garbage?

- TODO: benchmarks without GC running and without Boehm entirely
- Transition to looking at the actual implementations

Evaluator structures 🧬

Value

- Padding, etc.

List

- Special-cased for lists of size 0, 1, and 2, which can fit in a Value
- Implemented as a C-style array, so great data locality

Attribute set

- TODO: has it changed? I remember there being two arrays (one for names, one for values), but now it seems to be a vector of tuples.

Improvements 🔮

Suggested improvements should be **orthogonal** to those an **optimizing** or **parallel interpreter** would provide.

Persistent data structures

- TODO
- I mean, functional programming language with immutable values so why not benefit from sharing?
- Describe Immer library

Shrinking structures

- TODO: Link to branch I have with these changes

Future work 🔎

Future work

- Modularizing `benchmarking-nix-eval`
- Adding more benchmarks
- Building a web dashboard to visualize the data
- Integration into CI to detect regressions