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

- A Nix flake for benchmarking the Nix flake[1]
- Matrix 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] https://github.com/ConnorBaker/benchmarking-nix-eval

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

Examples 📊

Testbed setup

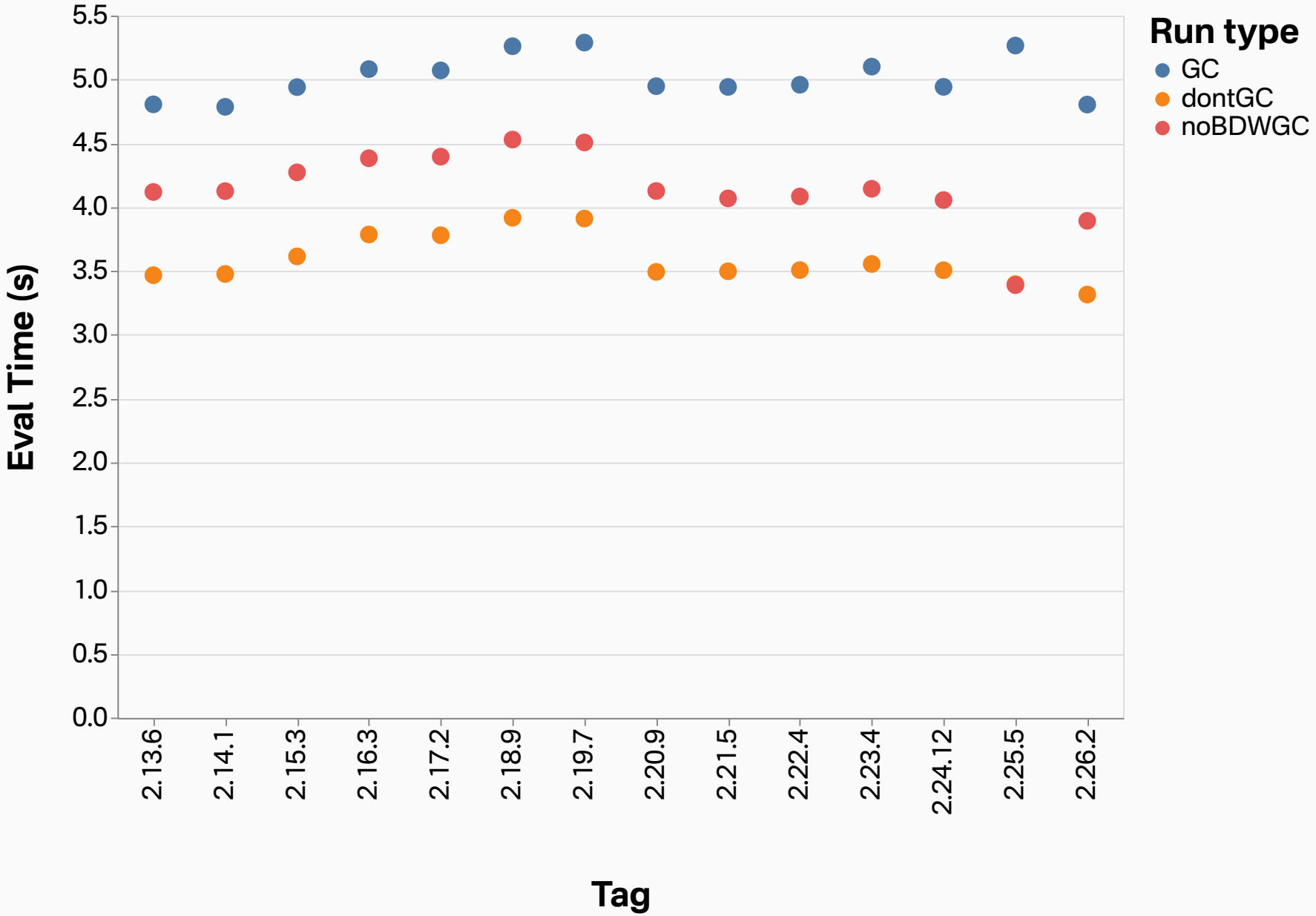
- Intel i9-13900K @ 3 GHz
  - Did not change niceness/pin to a favored core
- 96 GB DDR5 RAM
  - Did not attempt flushing caches
- Four-way ZFS RAID0
  - No deduplication/compression/integrity checking (just ARC)
  - Did not change IO niceness/flush caches
- Linux 6.12.13
- NixOS unstable @ `2ff53fe`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

Software setup

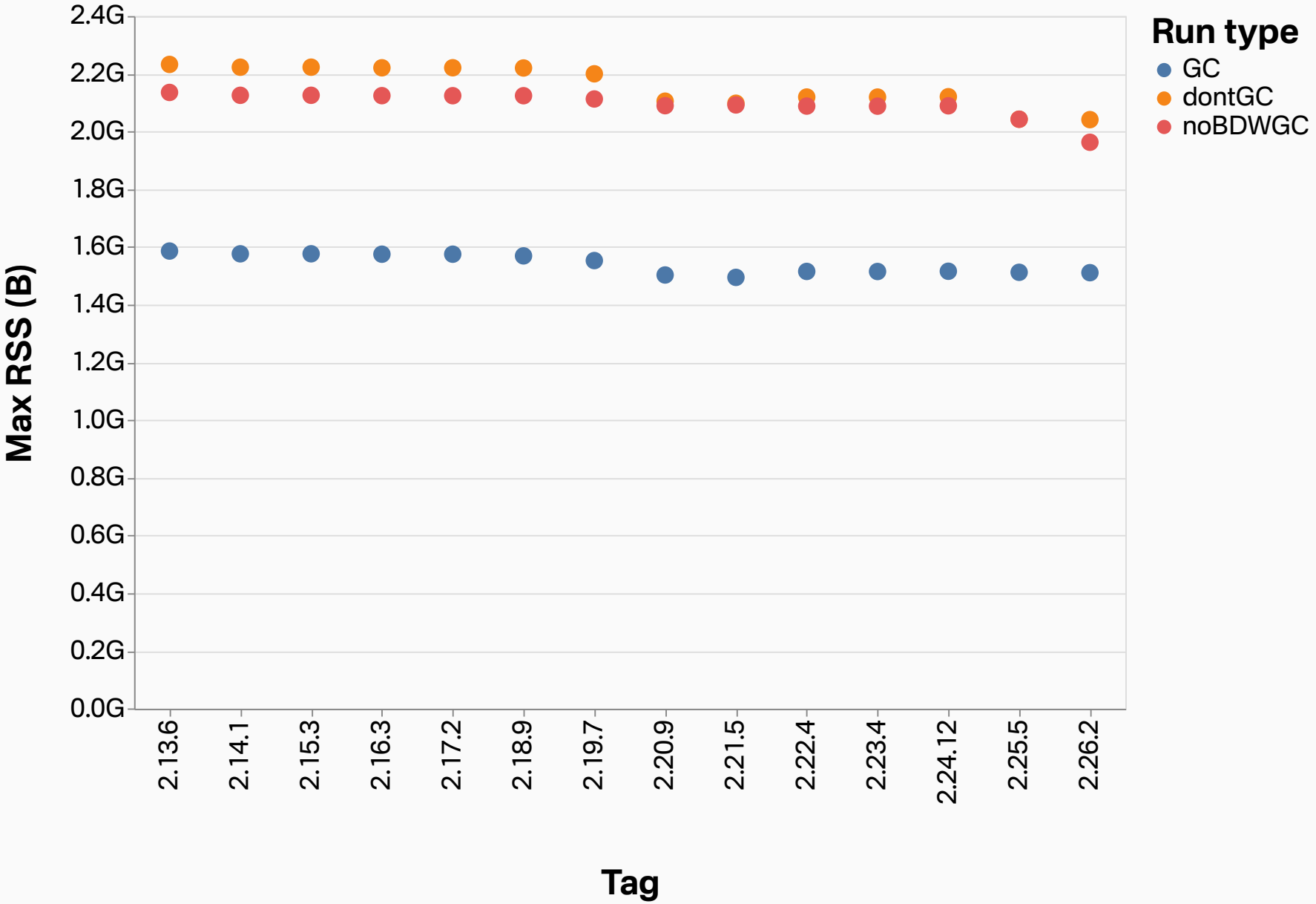
- Latest minor versions of Nix (2.13-2.26)
- Benchmarks run one at a time, 20 times each for each config
  - With collection (GC)
  - Without collection (dontGC)
  - Without BDWGC (noBDWGC)
- Median values are plotted
  - Observed little variation between runs
- Generated data is available[1]

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

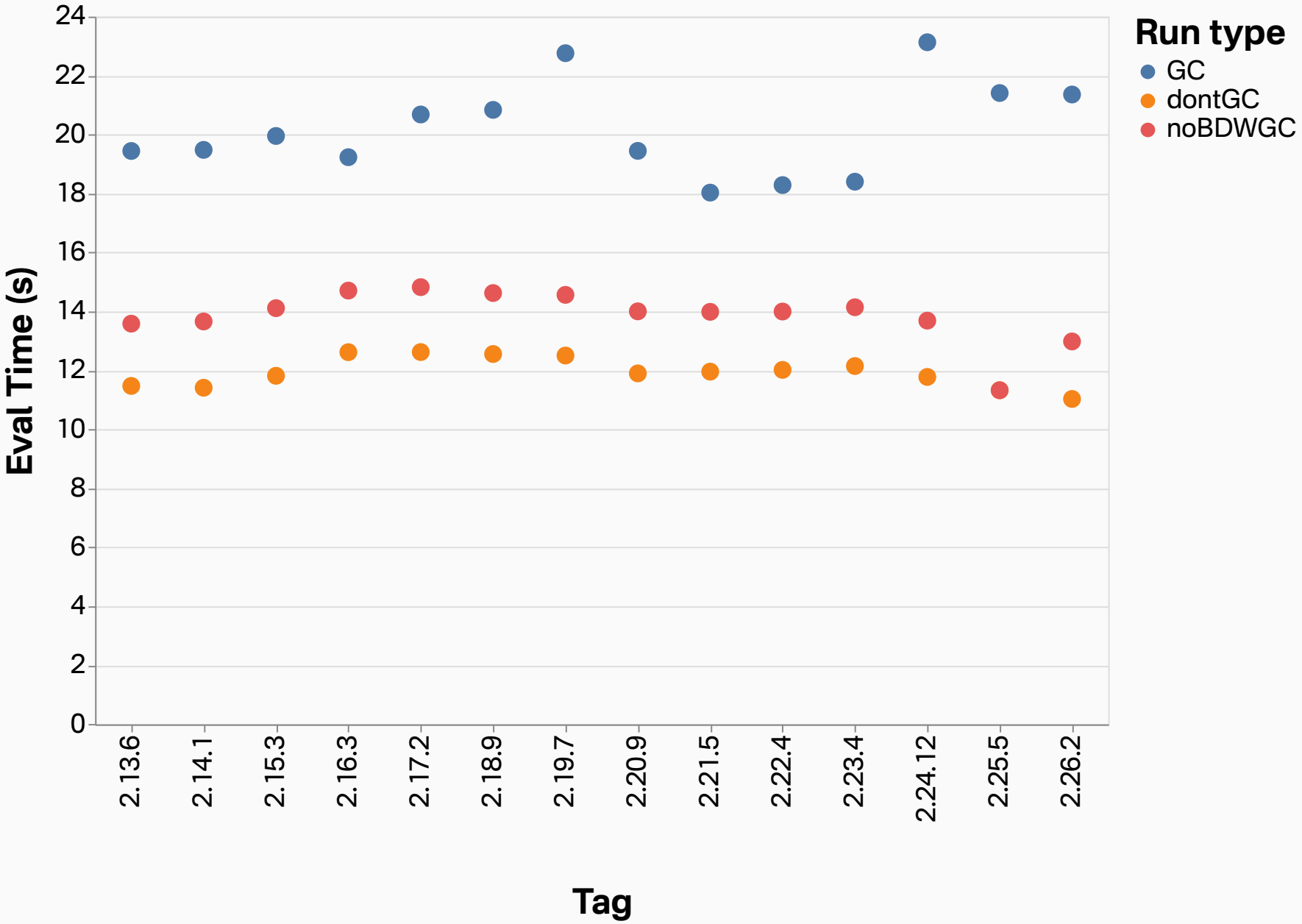
firefox-unwrapped eval time



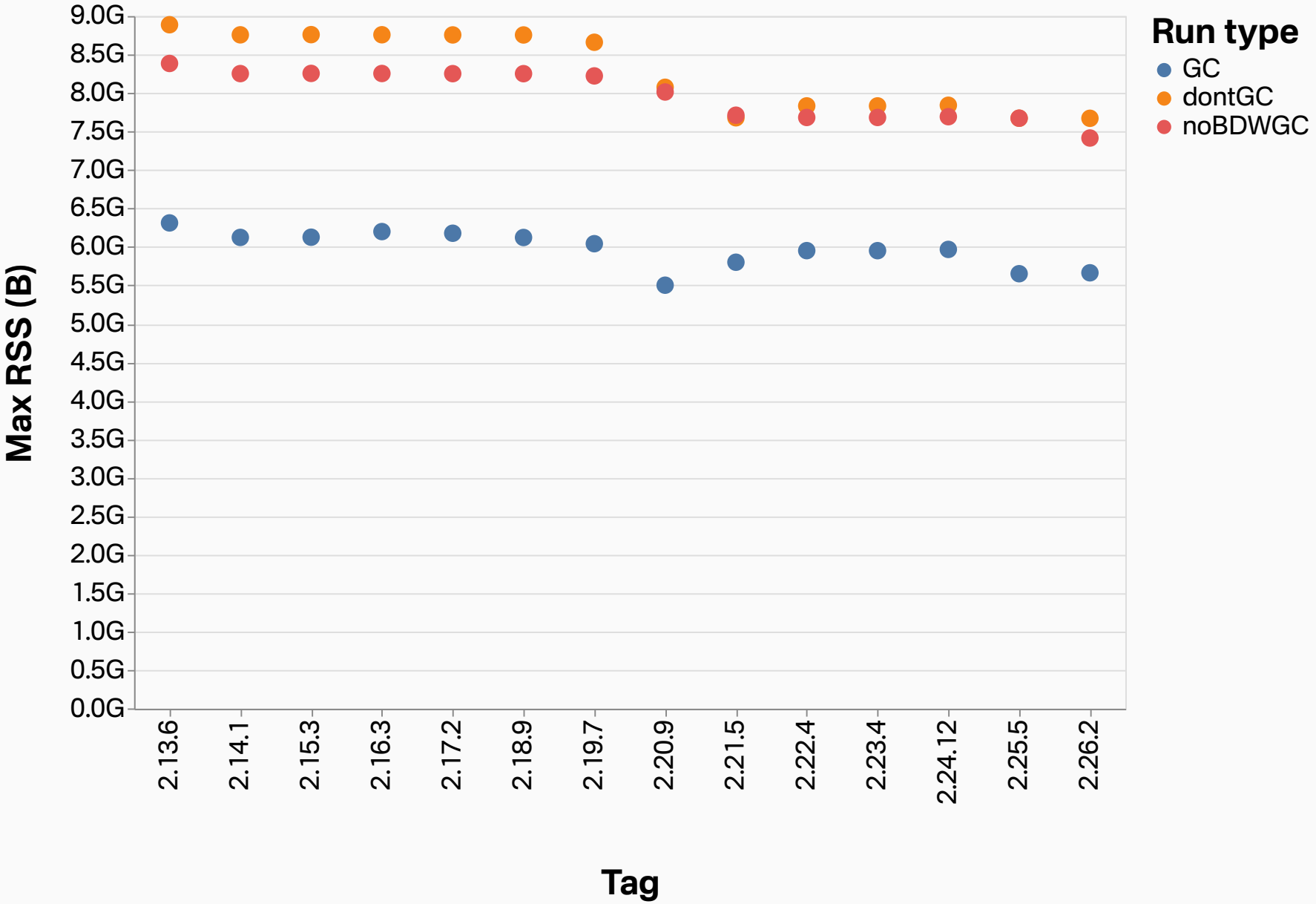
firefox-unwrapped eval space



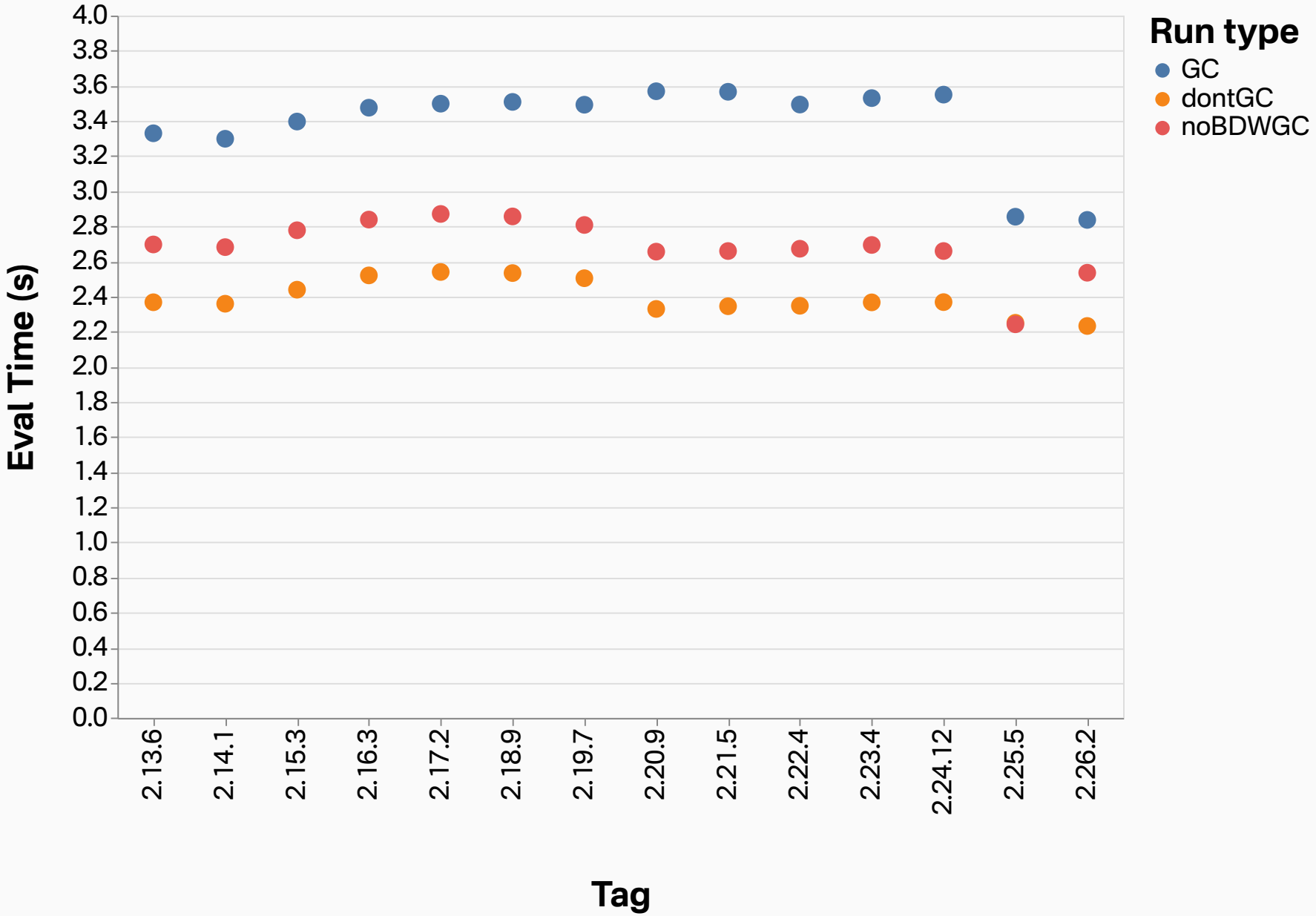
release-attrpaths-superset.names eval time



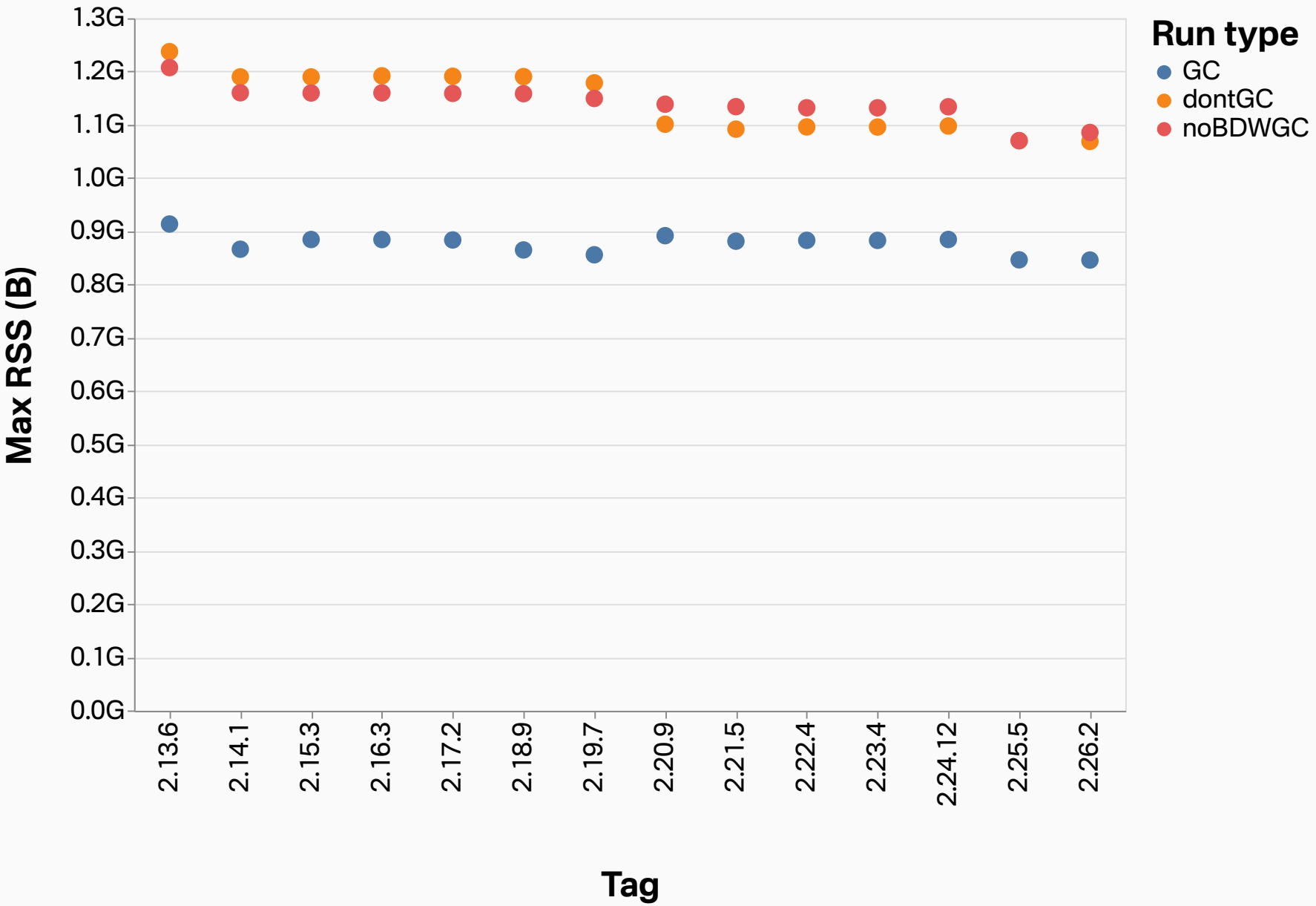
release-attrpaths-superset.names eval space



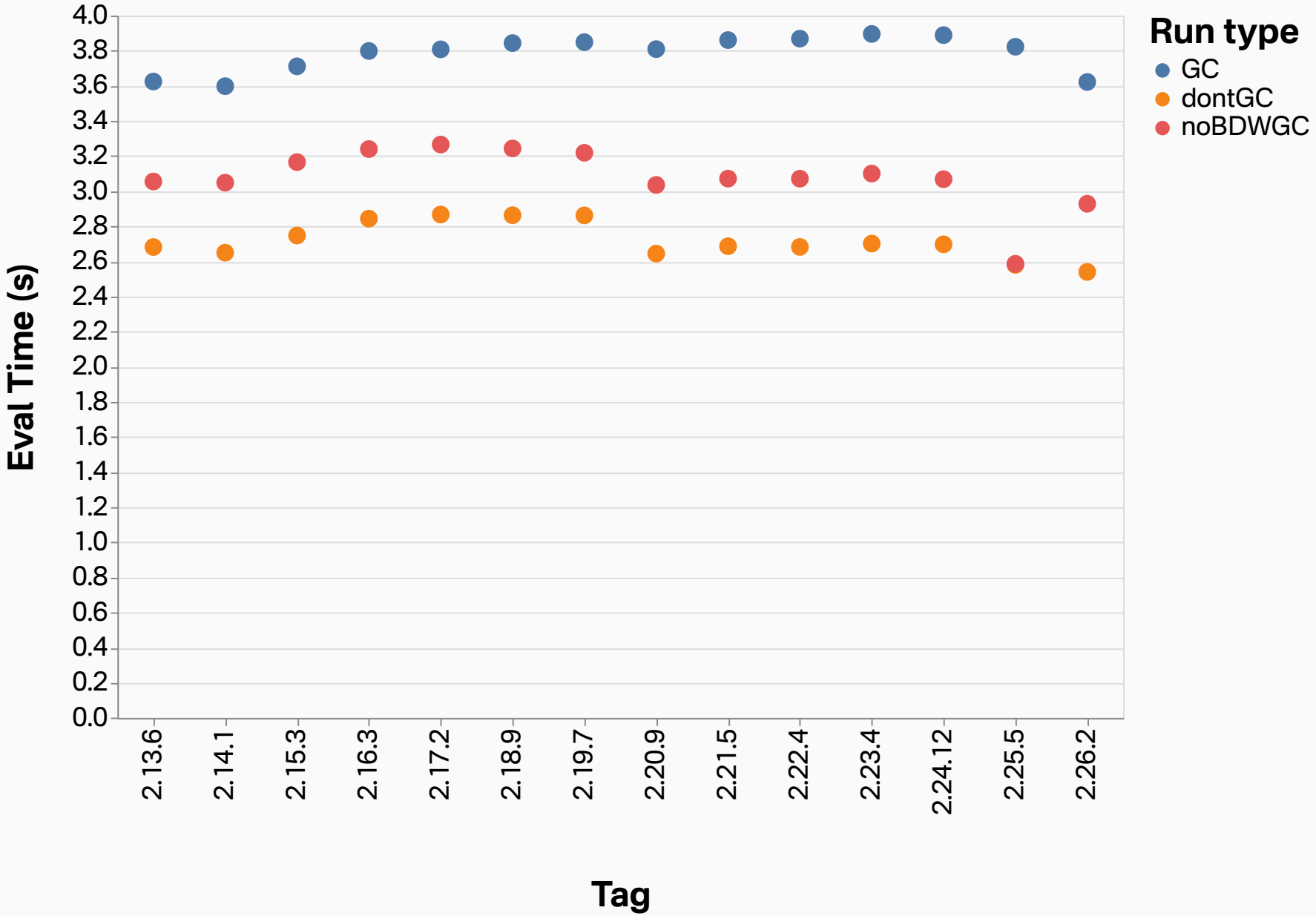
closures.smallContainer.x86_64-linux eval time



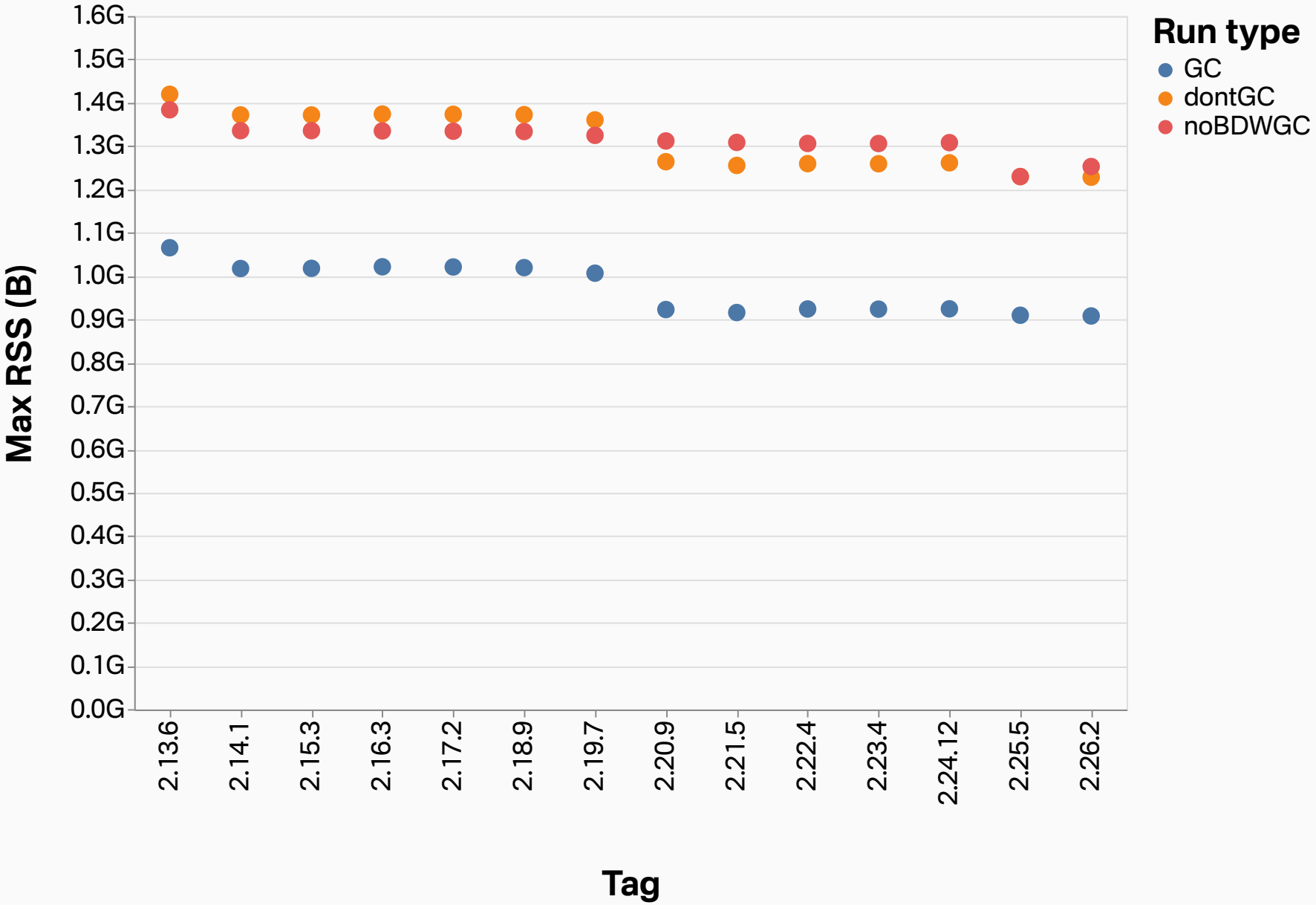
closures.smallContainer.x86_64-linux eval space



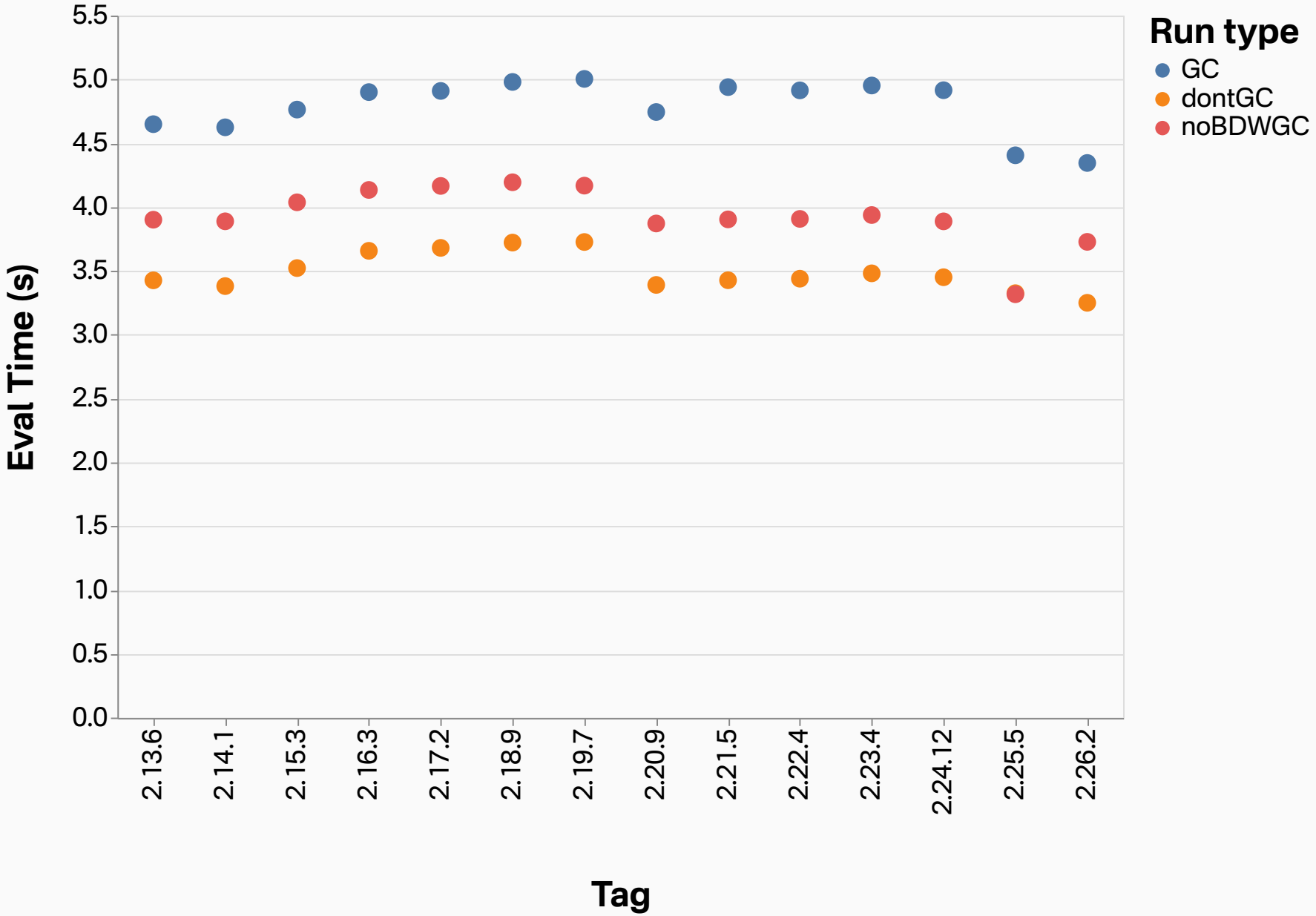
closures.lapp.x86_64-linux eval time



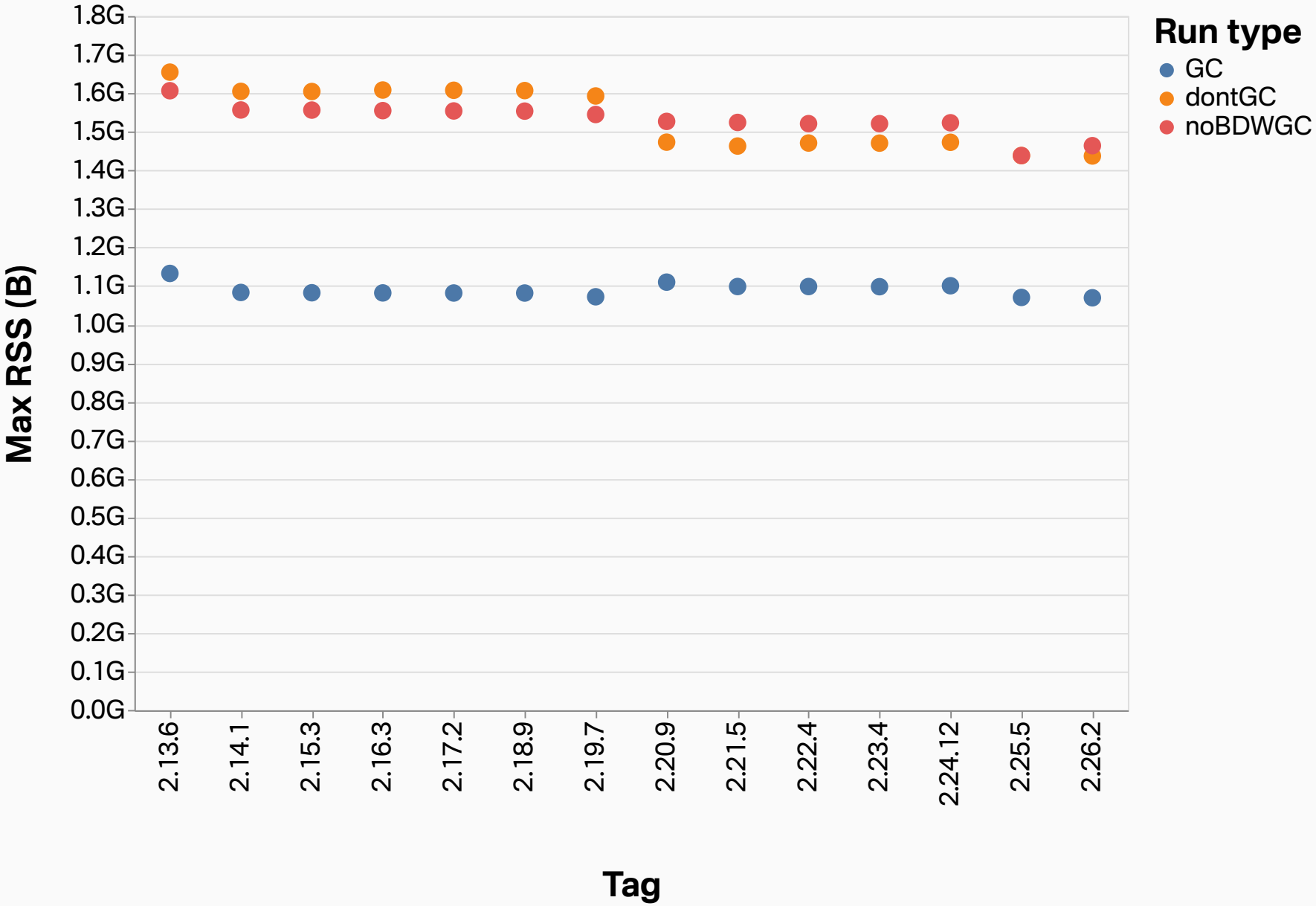
closures.lapp.x86_64-linux eval space



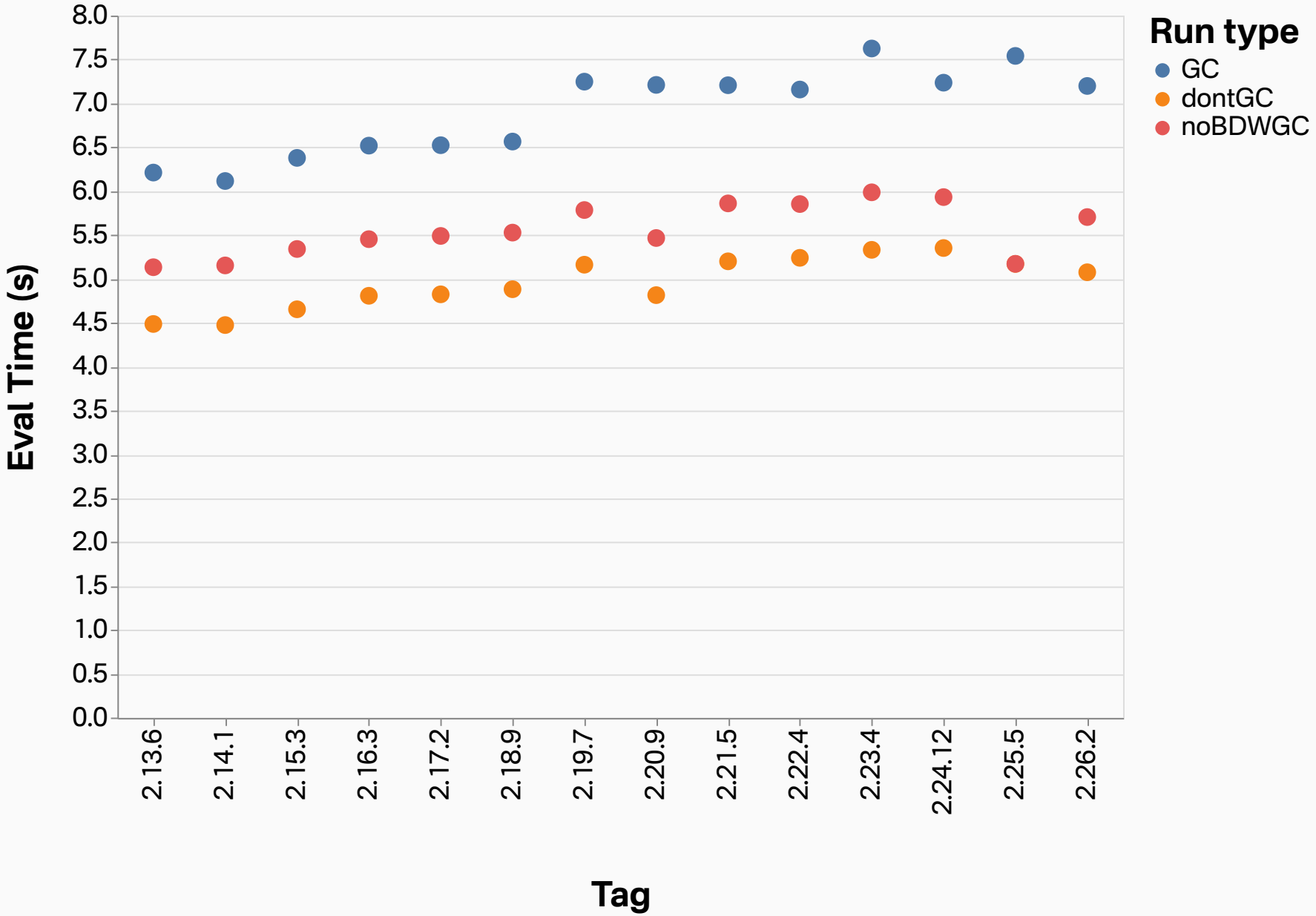
closures.kde.x86_64-linux eval time



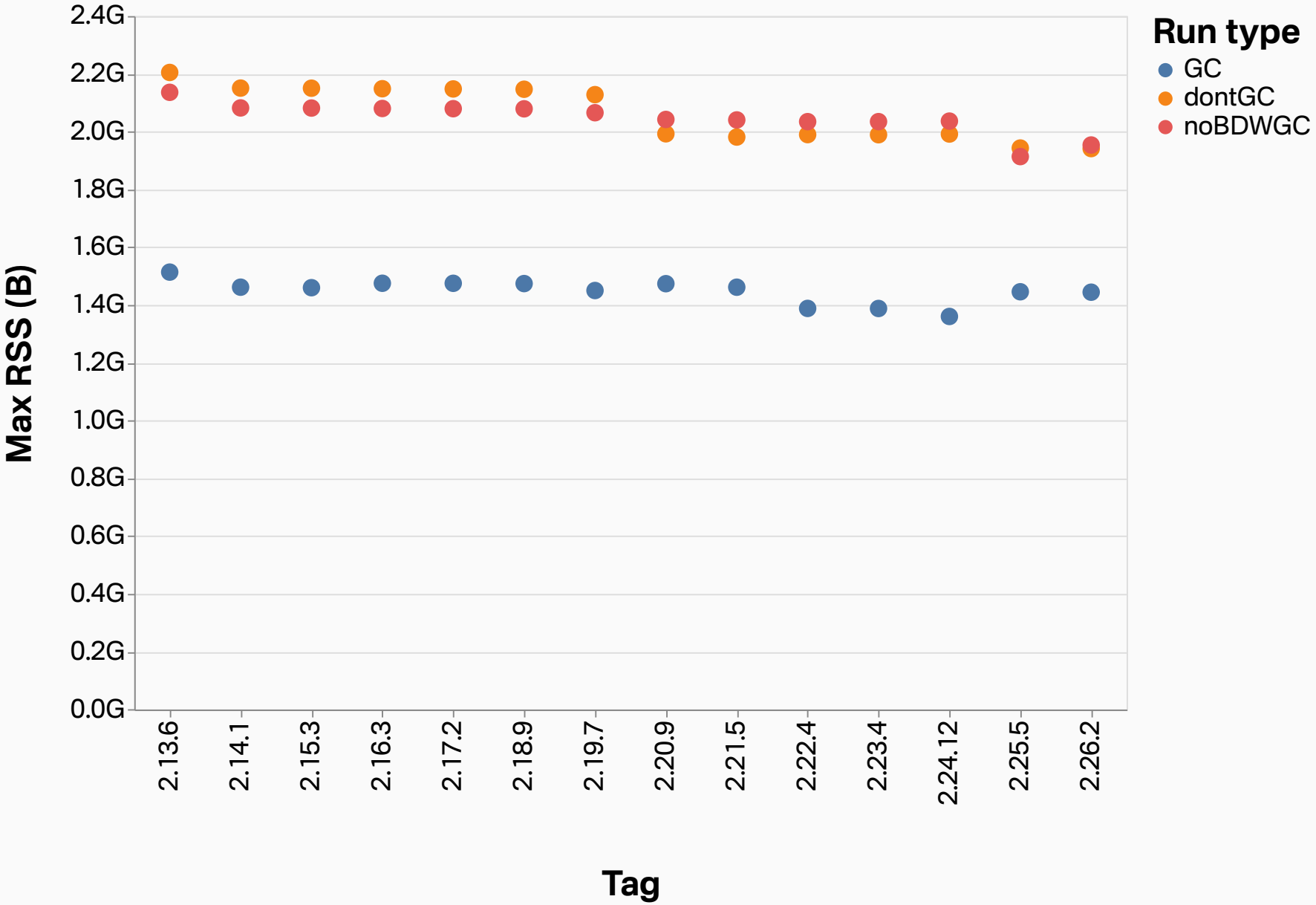
closures.kde.x86_64-linux eval space



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` (and Hydra) do this[1]
- No GC is slower than using BDWGC
  - Individual allocations vs. batched allocations
- No GC uses less memory than BDWGC
  - No bookkeeping overhead
- Evaluation should be separate from builds
  - Cache derivations
  - Build separately to avoid resource contention

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

Evaluator structures 🧬

Value

- Less than 20 possible (internal) types
- Structure is 24 bytes
  - 8 bytes (due to padding) for the type
  - 16 bytes for the actual content
- Created everywhere during evaluation

List

- 0/1/2 element lists are inlined into a `Value`
- Otherwise, a C-style array of `Value *`
  - Fantastic data locality
  - No sharing of existing values

Attribute set

- C-style array of `Attr`, a structure with three fields
  - `Symbol name` (4 bytes)
  - `PosIdx pos` (4 bytes)
  - `Value * value` (8 bytes)

Design

- Primitives for operating on values should:
  - be performant
  - be composable
- Data structure implementation should make the primitives exposed

Improvements 🔮

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

Data structures supporting **sharing**.

Shrinking the **Value** struct.

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