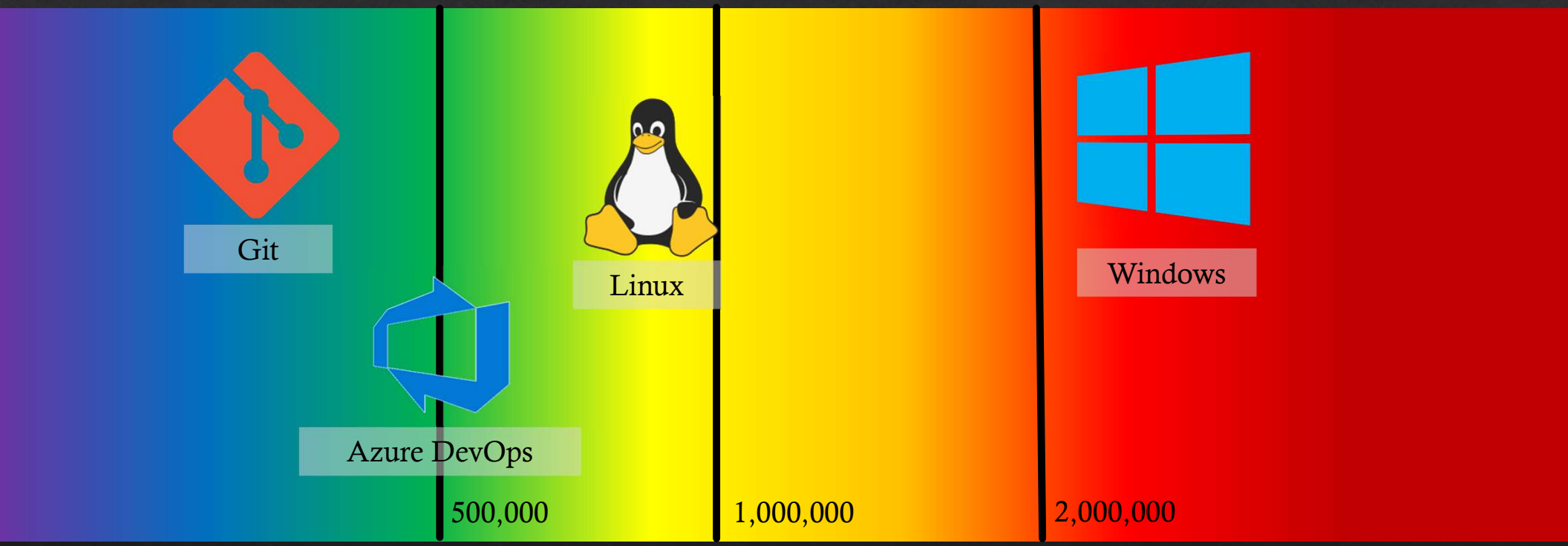
Git at Scale for Everyone

D. Stolee, Git Client Team, Microsoft

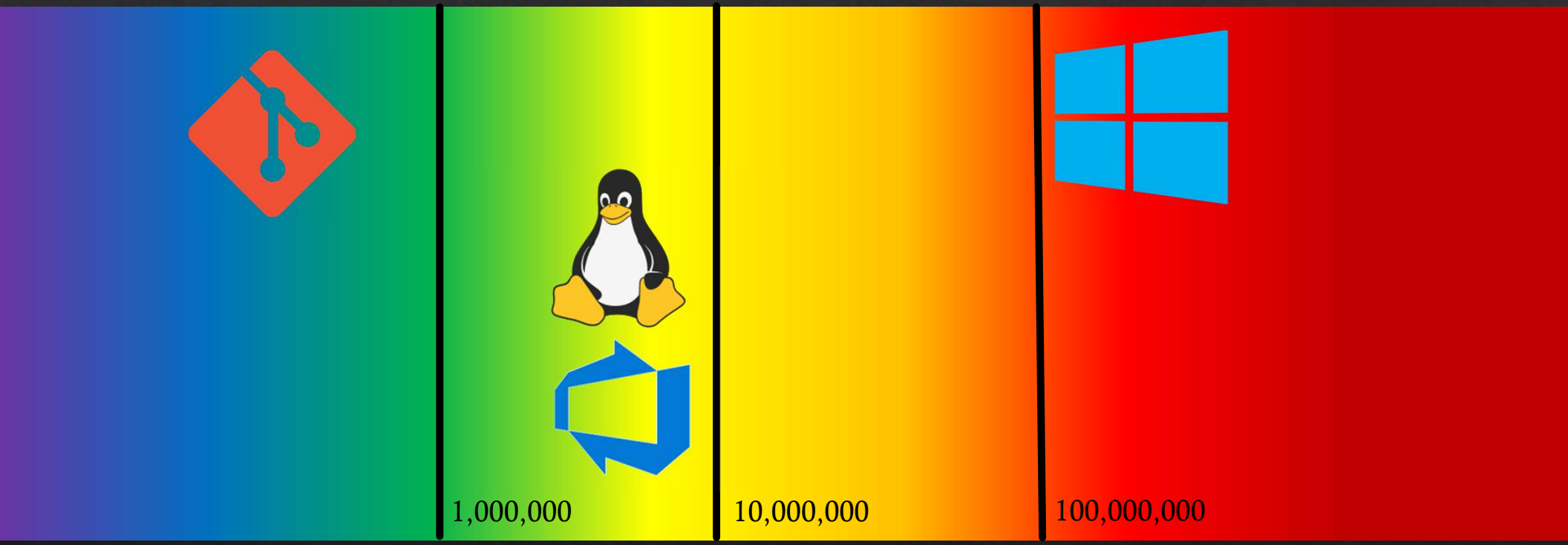
Twitter: @stolee GitHub: @derrickstolee

https://stolee.dev/docs/voss-2020.pdf

Spectrum of Scale

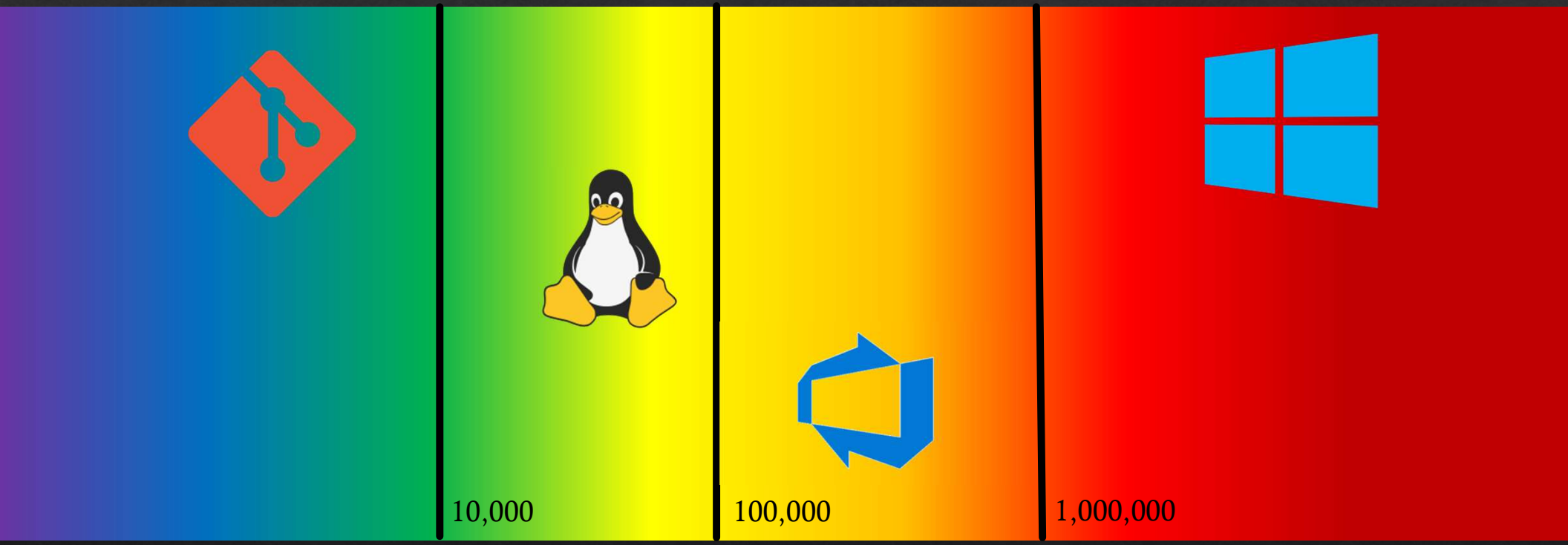


Spectrum of Scale



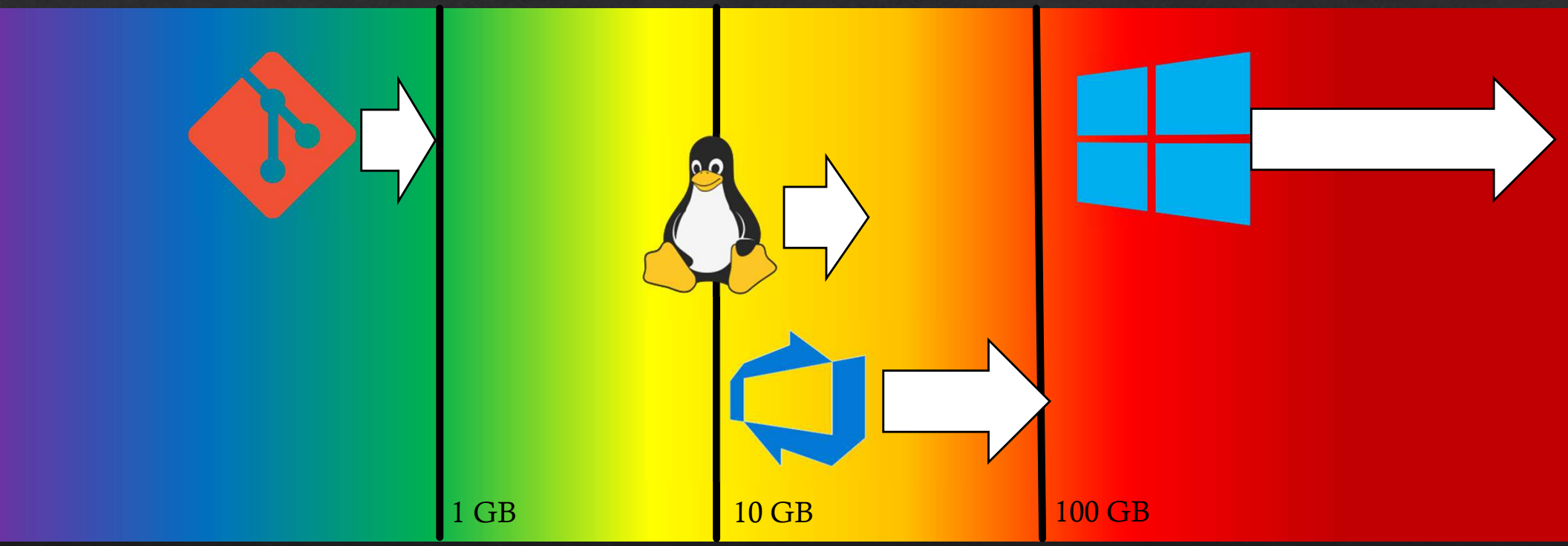
Total Object Count

Spectrum of Scale



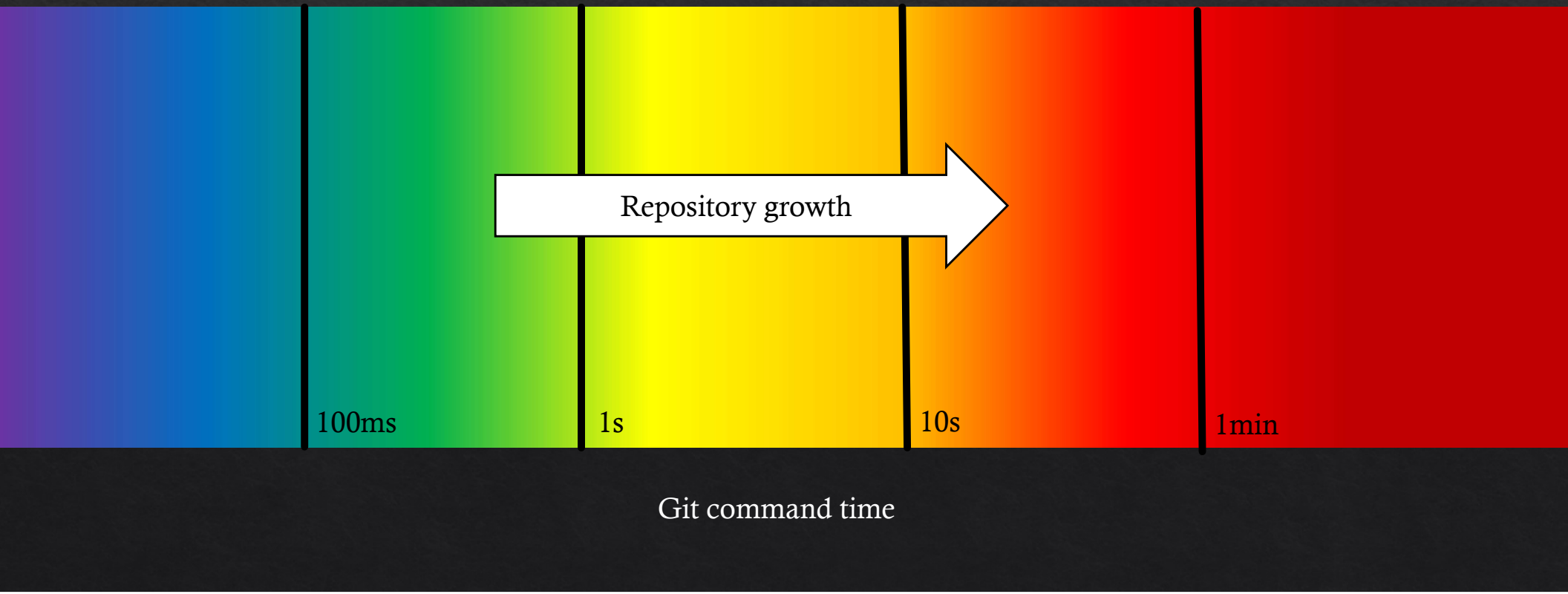
Files in Working Directory

Spectrum of Scale



Initial pack-file size

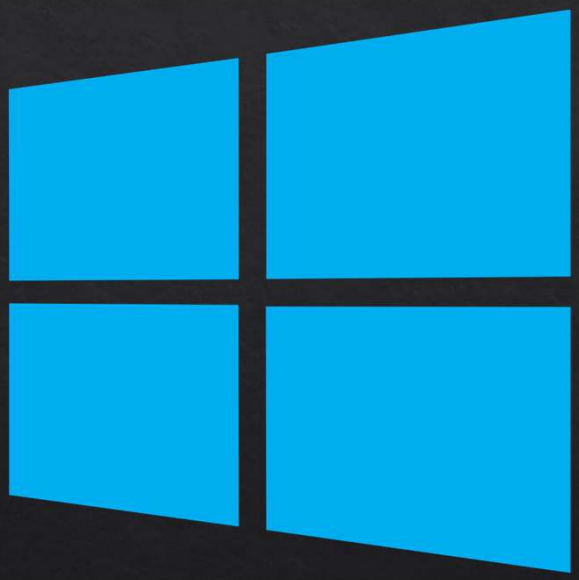
Spectrum of Perceived Performance



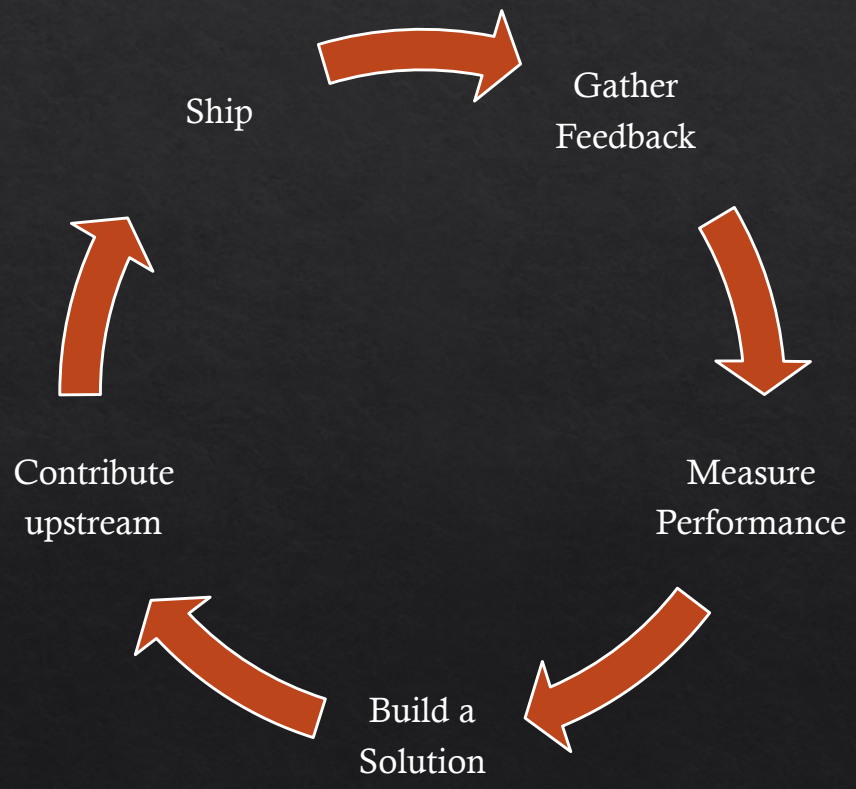
Git command time


http://gunshowcomic.com/648

Success Story: Microsoft Windows



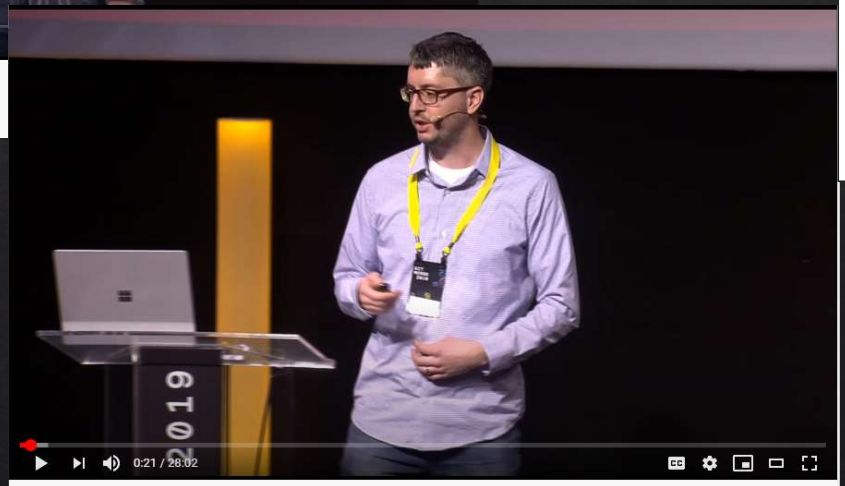
Improving Git Performance (Iteratively)



We make Git better *for everyone*

Making Git for Windows - Git Merge 2018

4,867 views • Apr 2, 2018



Technical contributions towards scaling for Windows - Git Merge 2019

1,109 views • Mar 11, 2019 20 1 SHARE SAVE

Supercharging the Git Commit Graph

Derrick

June 25th, 2018

Have you ev
window appe
history into a
parallel work
seconds for
performance
repository.

If you have a

Exploring new frontiers for Git push performance

Derrick

May 15th, 2019

In previous posts
that our team co
we've been push
Git repositories
sending these im
and later you can
algorithm that w
push operation o

A Deep Dive into Git Performance using Trace2

Jeff

August 6th, 2019

One of the cardinal rules when attempt
performance is to measure rather than
into the trap of attempting a performan
root-causing the real performance bott

Our team at Microsoft has been workin
performance of Git to scale up to extre
such as the Office and Windows reposit

Updates to the Git Commit Graph Feature

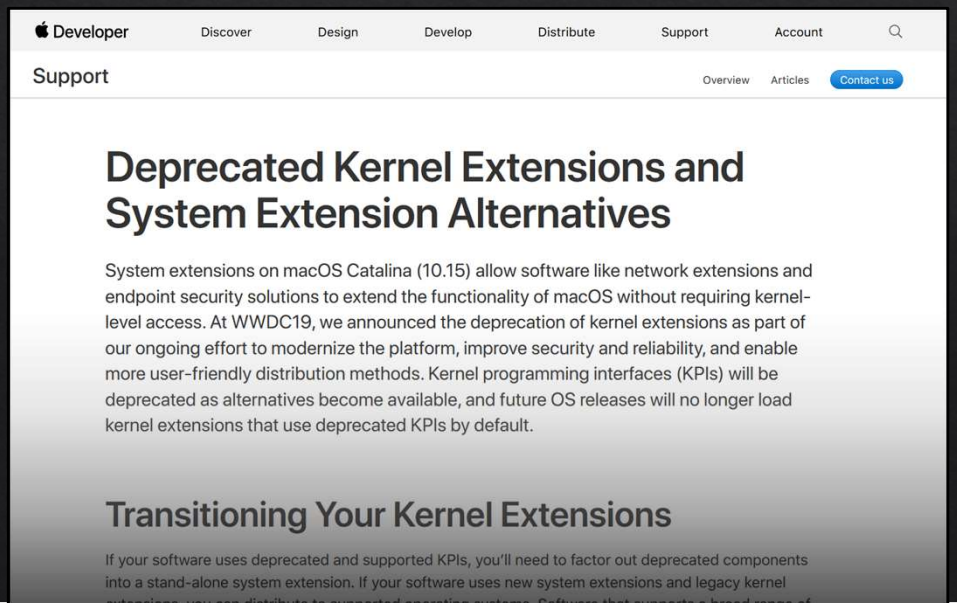
Derrick

November 11th, 2019

In a previous blog series, we announced that Git has a new *commit-graph* feature, and described some future directions. Since then, the commit-graph feature has grown and evolved. In the recently released Git version 2.24.0, the commit-graph is enabled by default! Today, we discuss what you should know about the feature, and what you can expect when you upgrade.

Next Milestone: Microsoft Office

- Similar size and shape to Windows OS repo
- Hosted on Azure Repos
- Client **must** work on Windows & macOS



https://github.com/microsoft/scalar

MIT License

Built in the open. Contributions welcome!

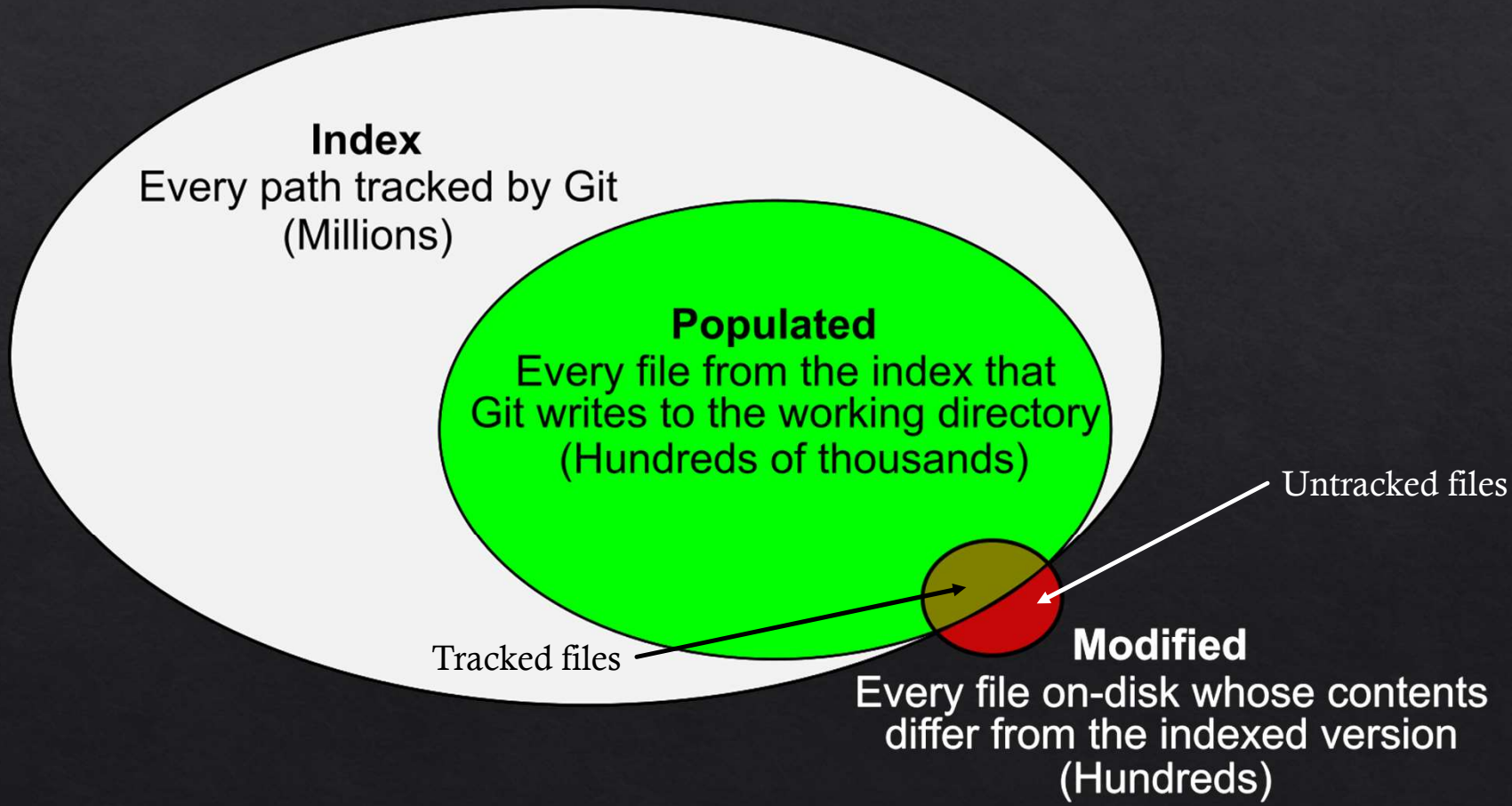
Lessons for Git at Scale

Lesson 1: Focus on the files that matter

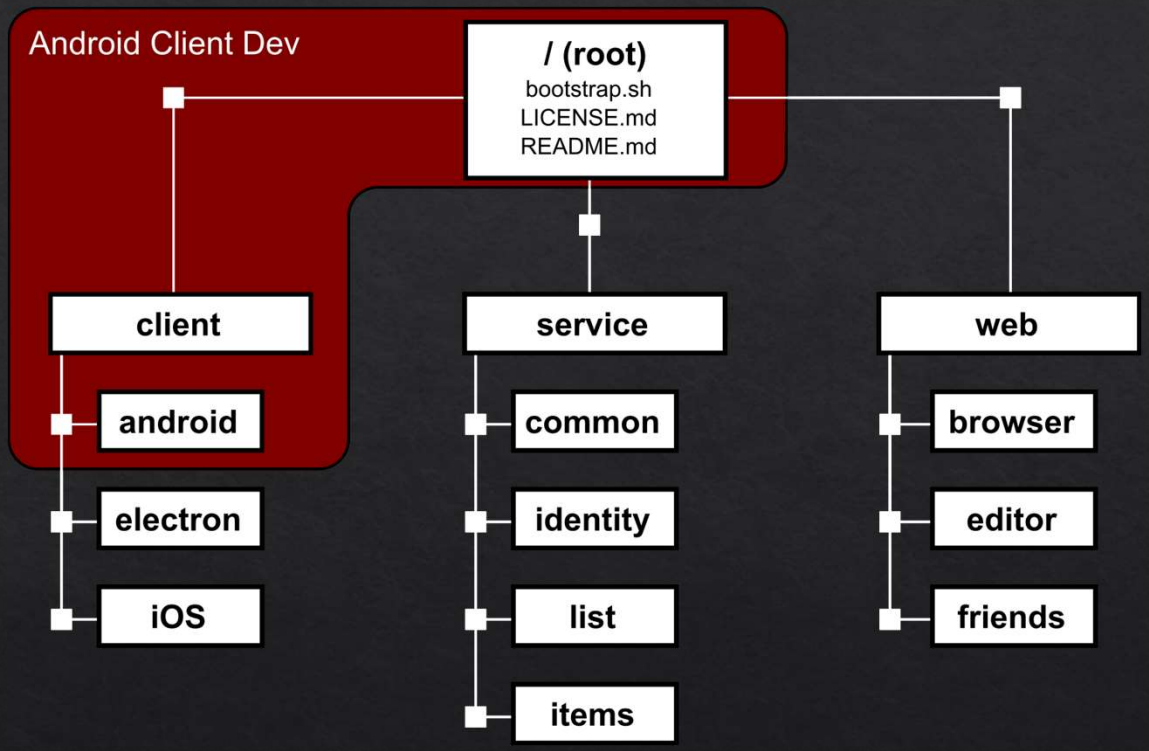
Lesson 2: Reduce object transfer

Lesson 3: Don’t wait for expensive operations

Lesson 1: Focus on the files that matter



Reduce Populated Size: Sparse-checkout



Demo: `git sparse-checkout`

https://github.blog/2020-01-17-bring-your-monorepo-down-to-size-with-sparse-checkout/

Sparse-checkout (default mode)

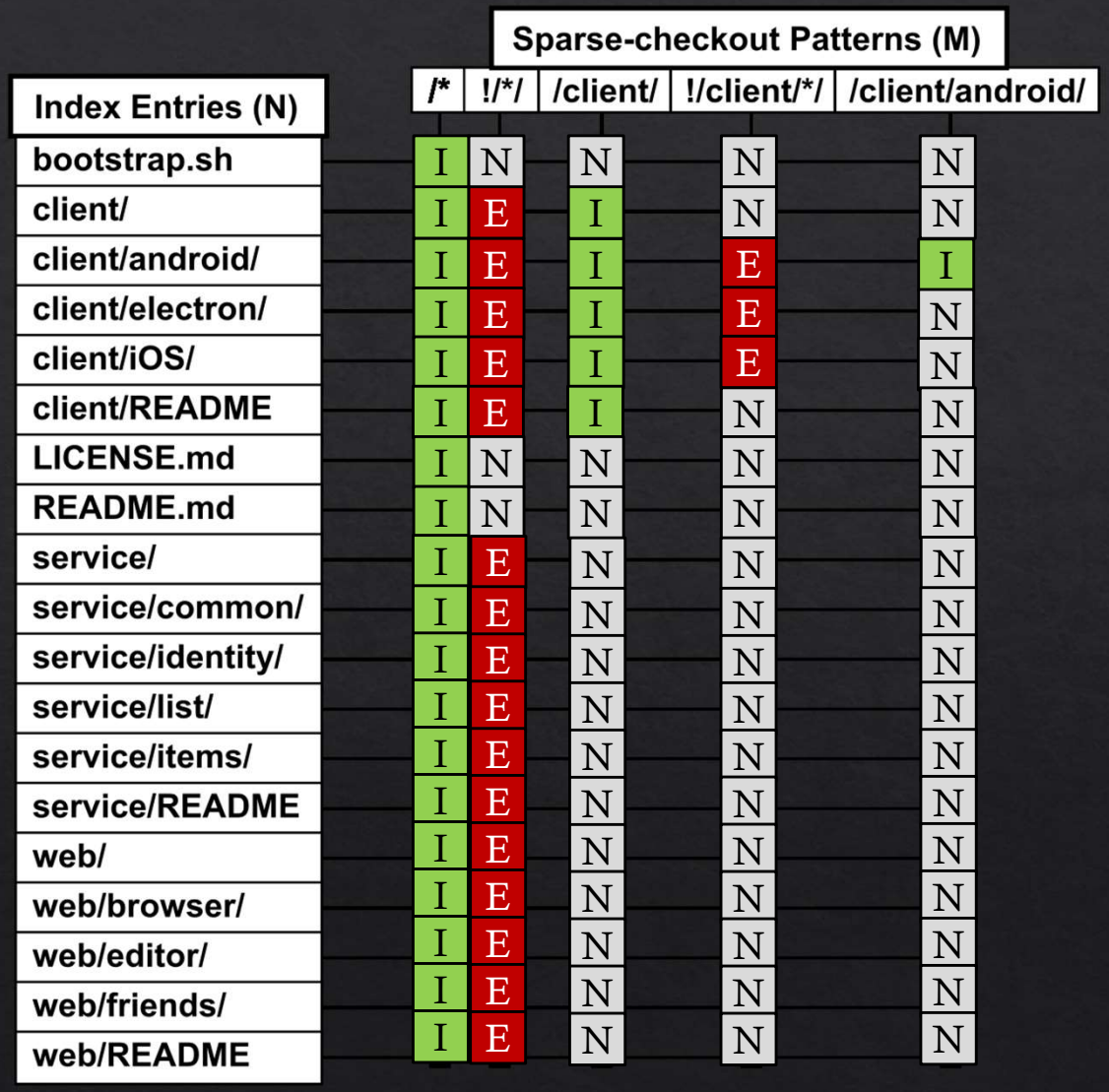
Patterns are verified in order against every path in the index.

Positive patterns *include* files

Negative patterns *exclude* files

On a typical sparse-checkout pattern for the **Microsoft Office** repo, this calculation takes

40 minutes



Sparse-checkout (cone mode)

Create two hashsets:

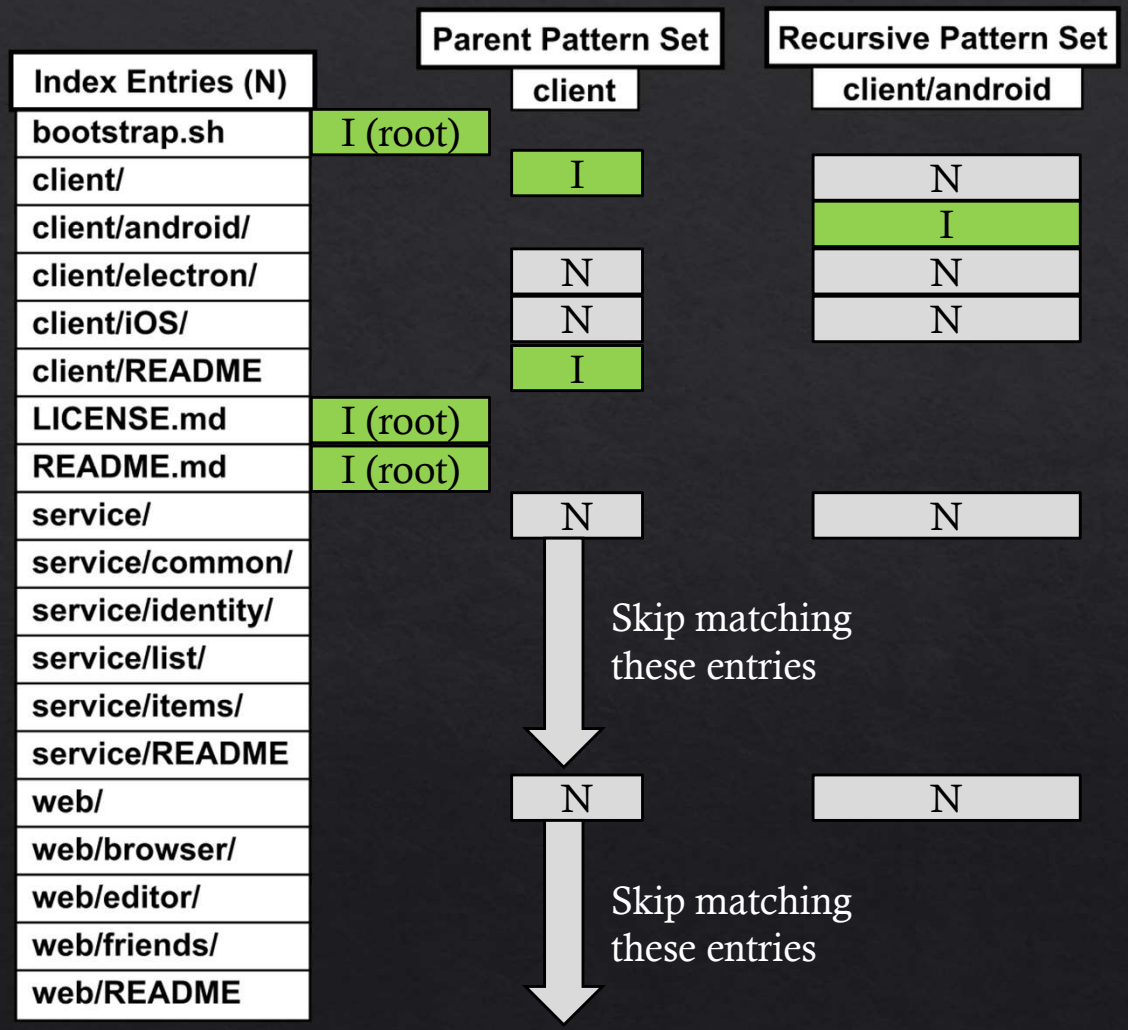
- Parent patterns
- Recursive patterns

Match based on exact strings

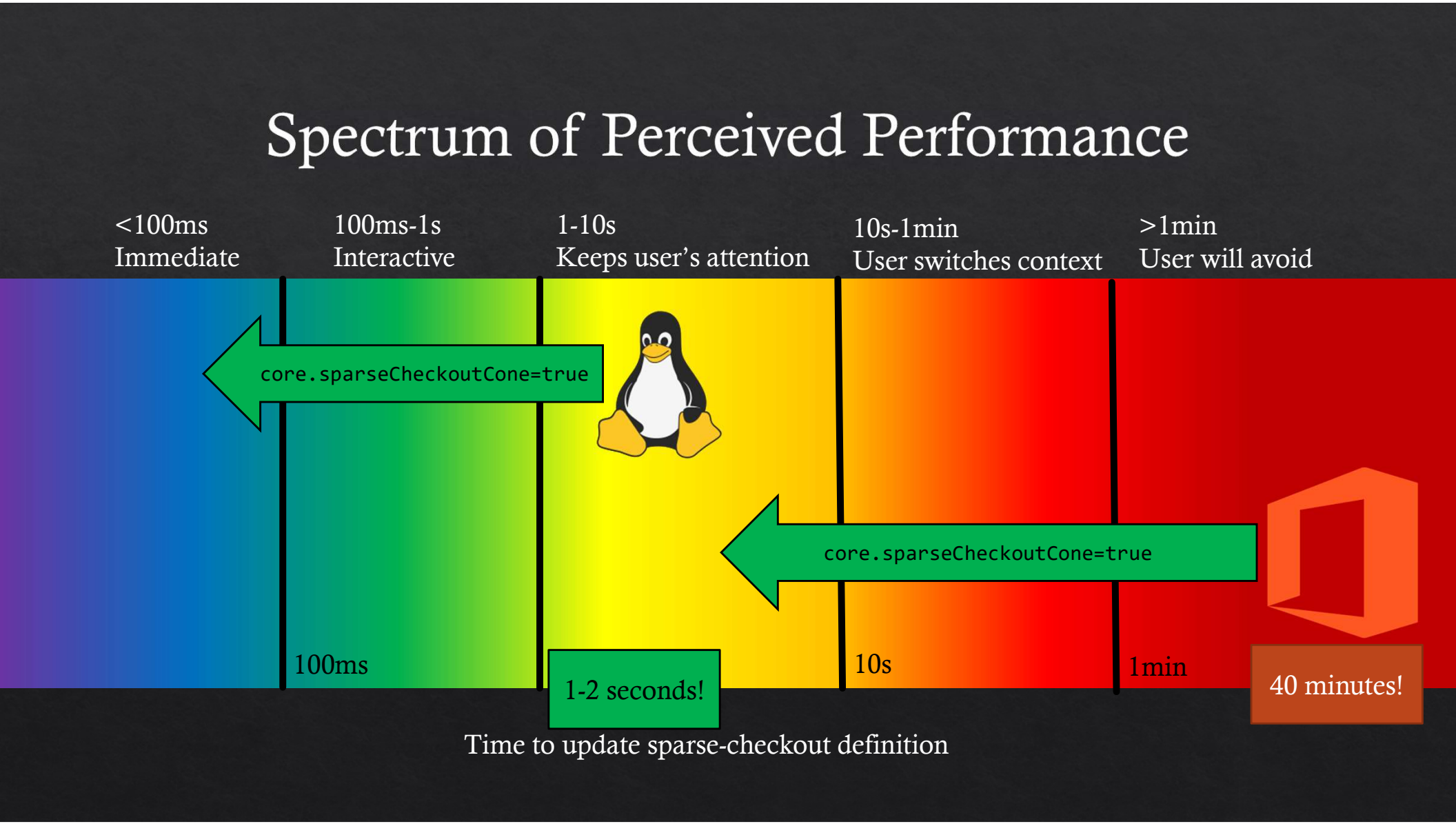
(remove filename if not directory)

On all known sparse-checkout patterns for the **Microsoft Office** repo, this calculation takes

1-2 seconds



Spectrum of Perceived Performance

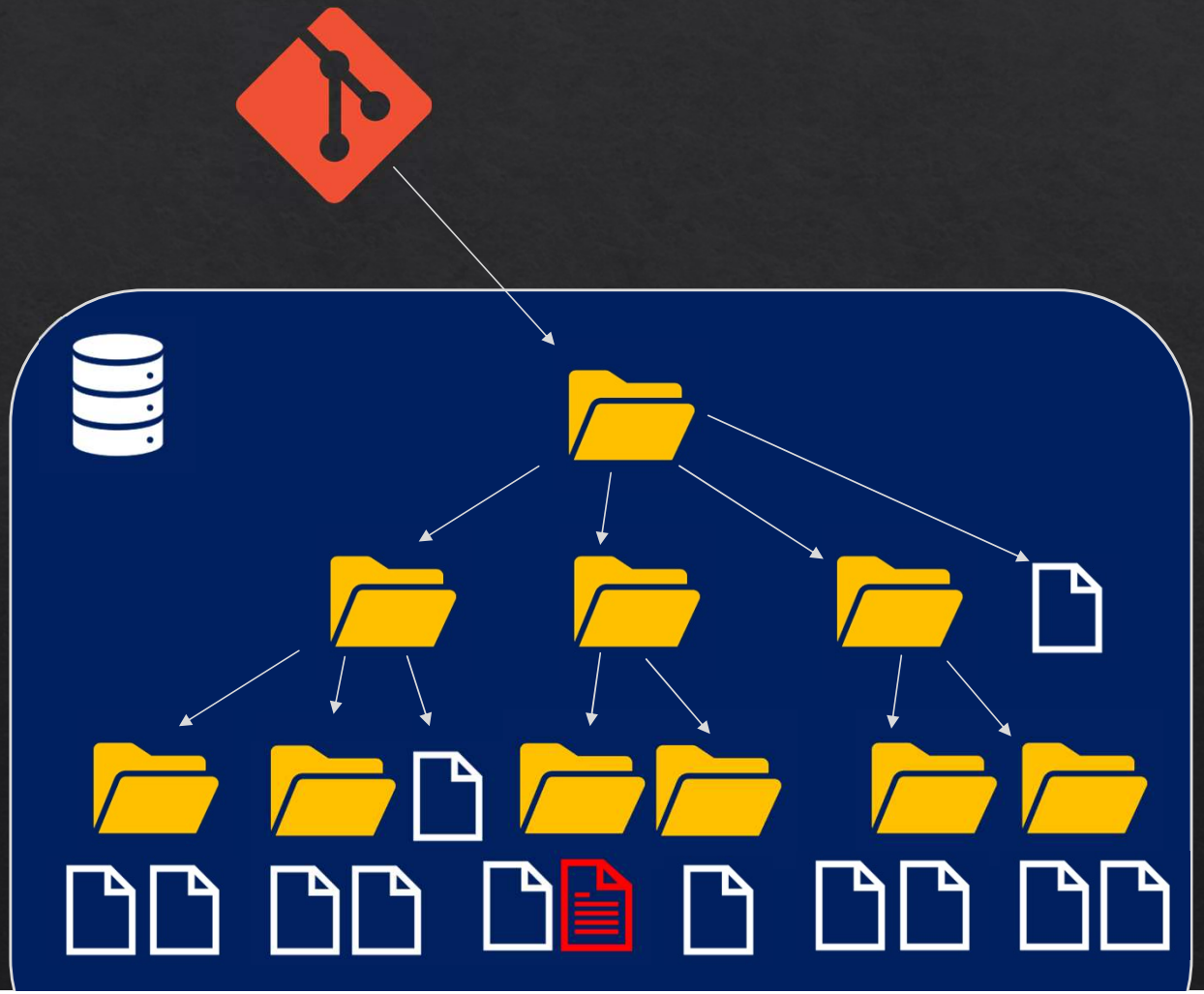


Time to update sparse-checkout definition

Finding Modified Files with Filesystem Monitor

Commands like `git status` or `git add` need to know which files were modified since the last checkout.

This usually results in scanning directories.



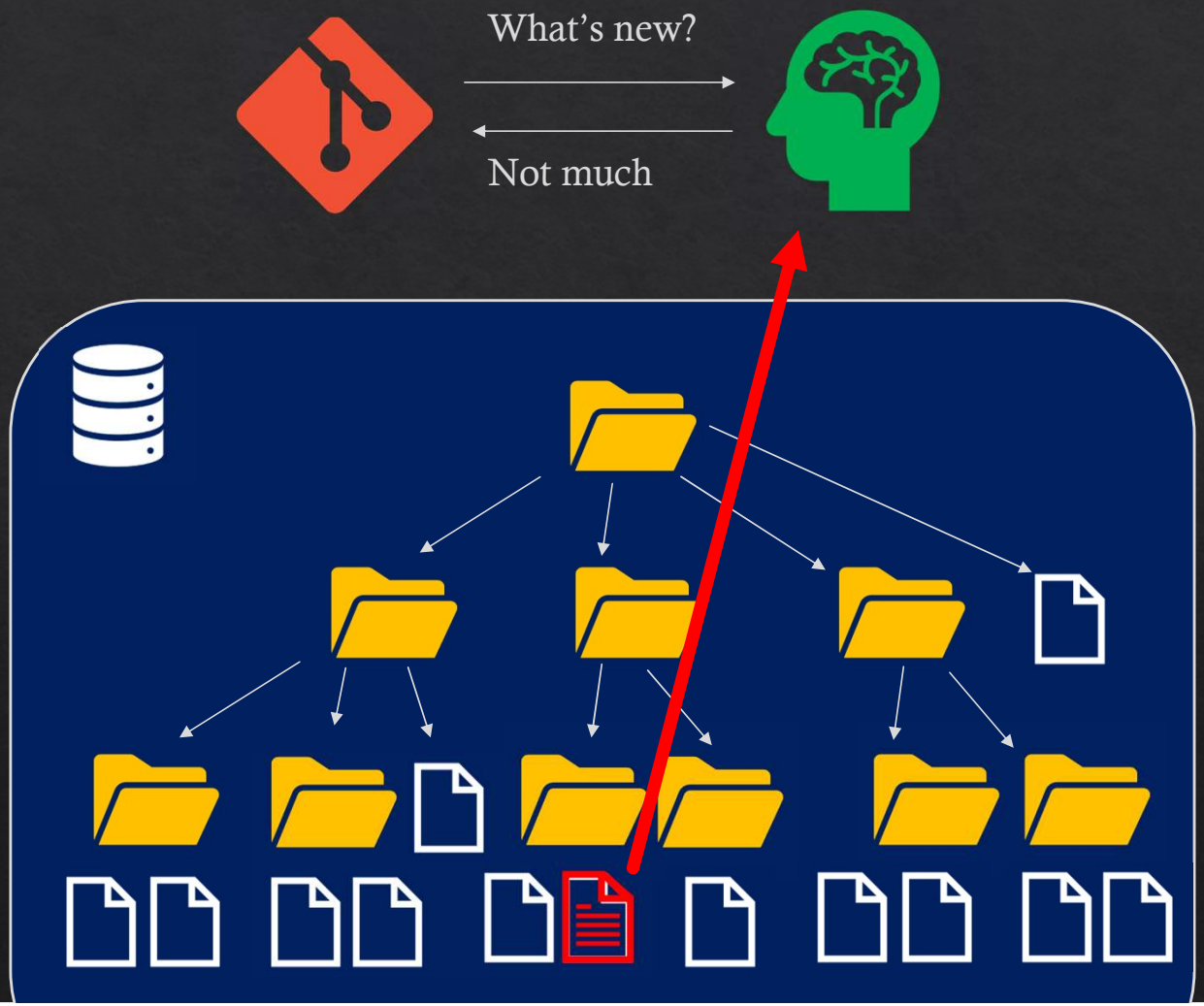
Finding Modified Files with Filesystem Monitor

Commands like `git status` or `git add` need to know which files were modified since the last checkout.

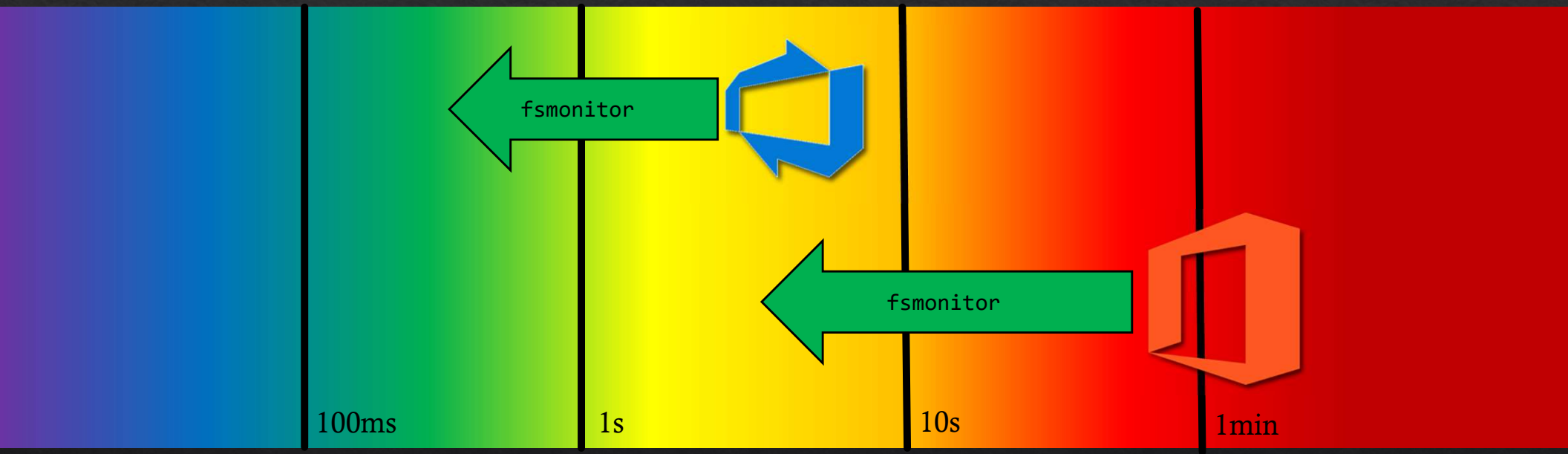
This usually results in scanning directories.

With the `fsmonitor` hook, Git can get a list from a specialize filesystem watcher, such as

`https://github.com/facebook/watchman`



Spectrum of Perceived Performance



`git status` command time

How can Git better focus on files that matter?

Sparse-Checkout

- Continued UX improvements
  - `git sparse-checkout add <dir>`
  - `git sparse-checkout remove <dir>`
  - `git sparse-checkout stats`
  - Update with non-empty `git status`

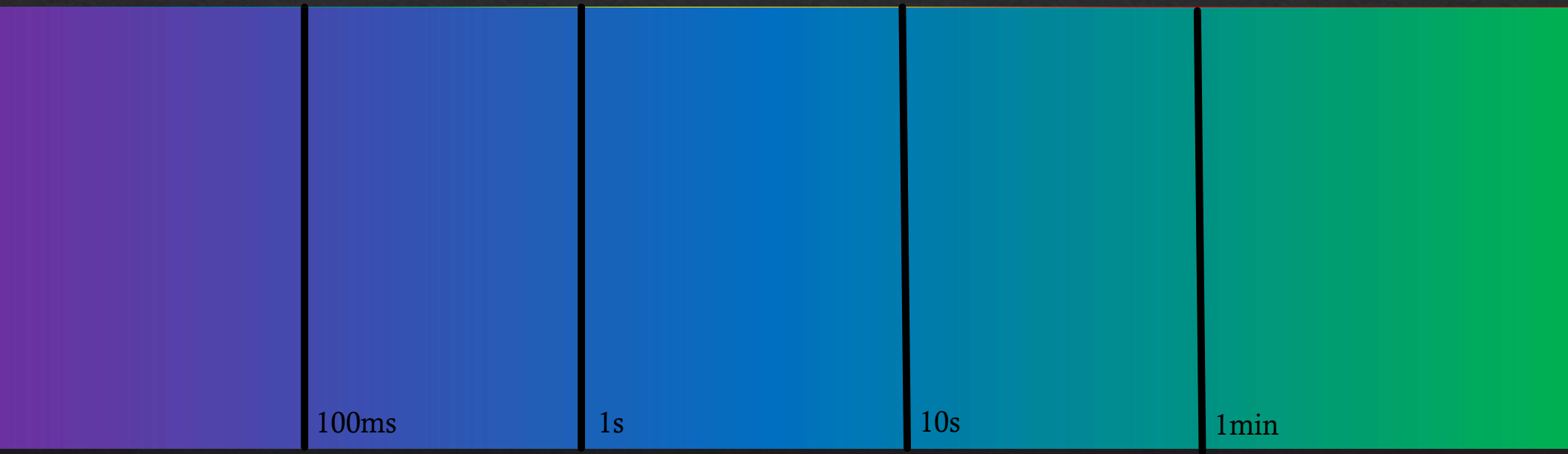
Filesystem Monitor

- Make the hook more robust, faster
- We are preparing a **Git-aware** filesystem monitor.

Lesson 2: Reduce Object Transfer

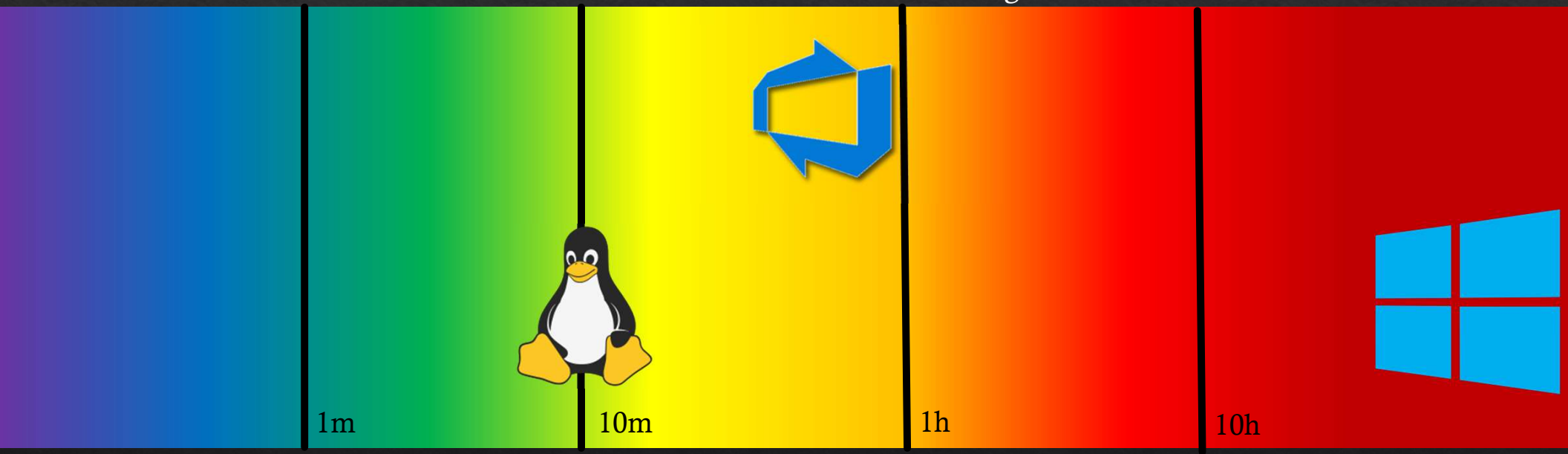
```
dstolee@stolee-book MINGW64 /c/_git/t
$ git clone --single-branch https://dev.azure.com/mseng/_git/AzureDevOps
Cloning into 'AzureDevOps'...
remote: Azure Repos
remote: Found 6938156 objects to send. (1090 ms)
Receiving objects:   0% (18433/6938156), 3.13 MiB | 1.17 MiB/s
```

Spectrum of Perceived Performance



`git clone` time

Spectrum of Perceived Performance



`git clone` time

GVFS Protocol → Partial Clone

GVFS protocol (Created 2015-16)

- Uses these REST API endpoints:
  - `GET <url>/gvfs/config`
  - `GET <url>/gvfs/objects/{objectid}`
  - `POST <url>/gvfs/objects`
  - `GET <url>/gvfs/prefetch`
  - `POST <url>/gvfs/sizes`

Git Partial Clone (Created 2018)

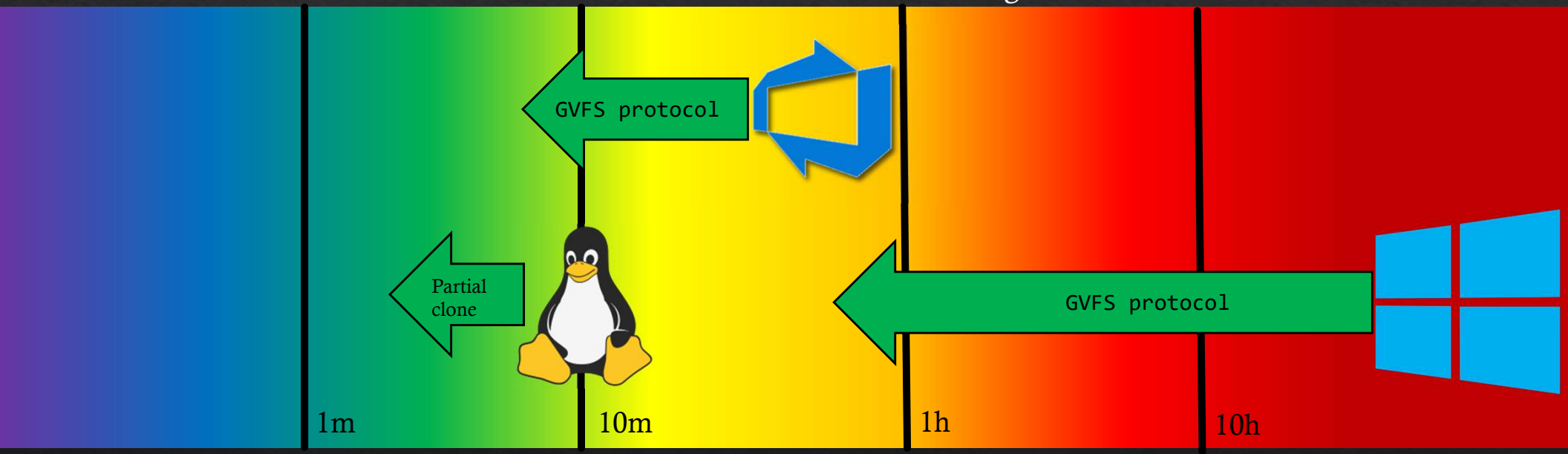
- `git clone --filter=blob:none <url>`
- Fetches only commits and trees
- Blobs are fetched in a batch request during `git checkout` and similar requests

Now available on all GitHub.com repositories!

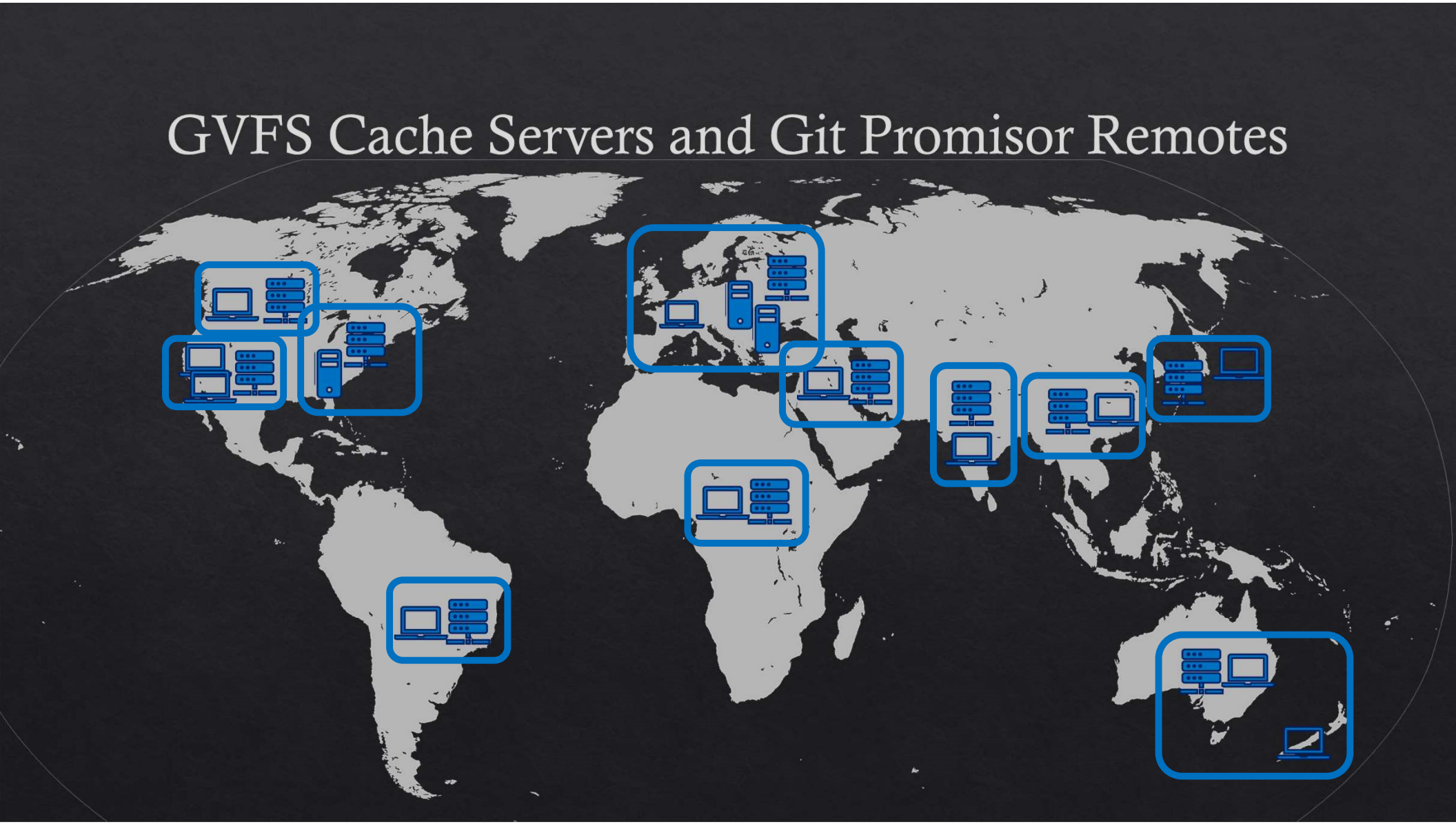
Reduced Object Transfer + Sparse-Checkout = Success!

`https://git-scm.com/docs/partial-clone`

Spectrum of Perceived Performance



Time for git clone vs partial clone or GVFS protocol


GVFS Cache Servers and Git Promisor Remotes

Lesson 3: Don’t wait for expensive operations

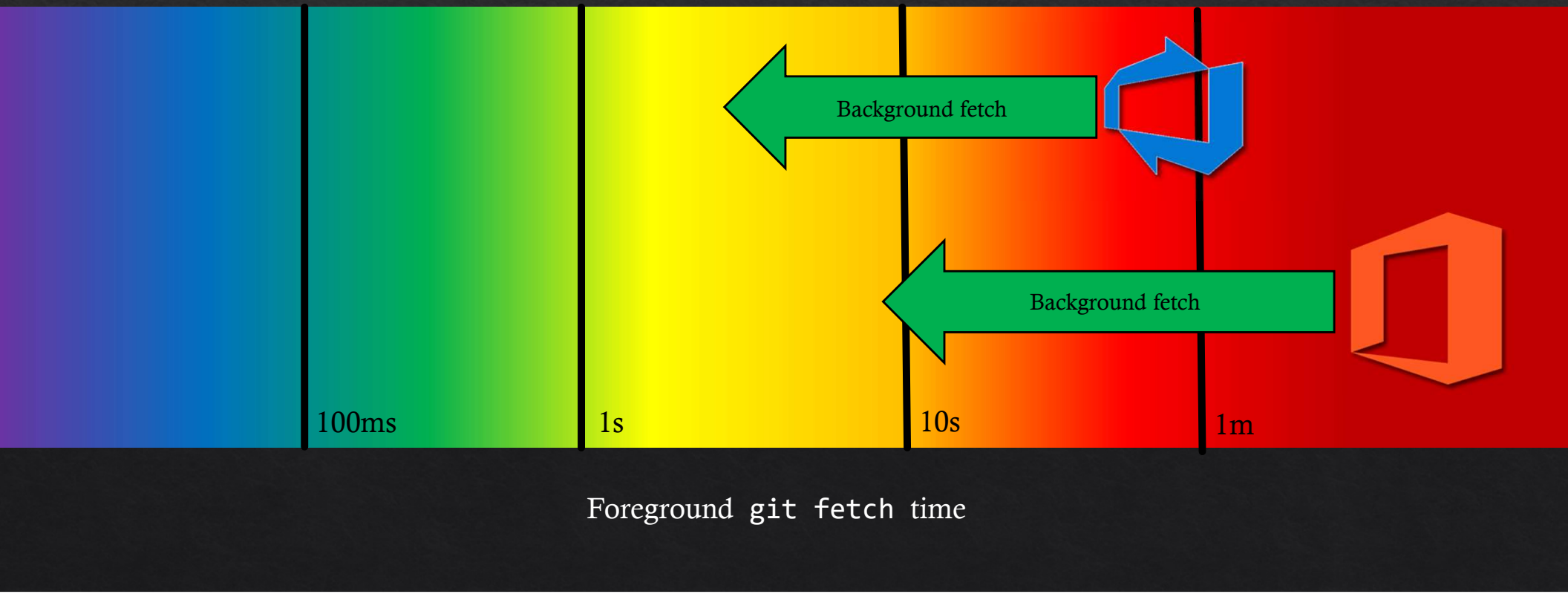
https://xkcd.com/303/

Background Maintenance

The following can be done in the background, reducing user-blocking time:

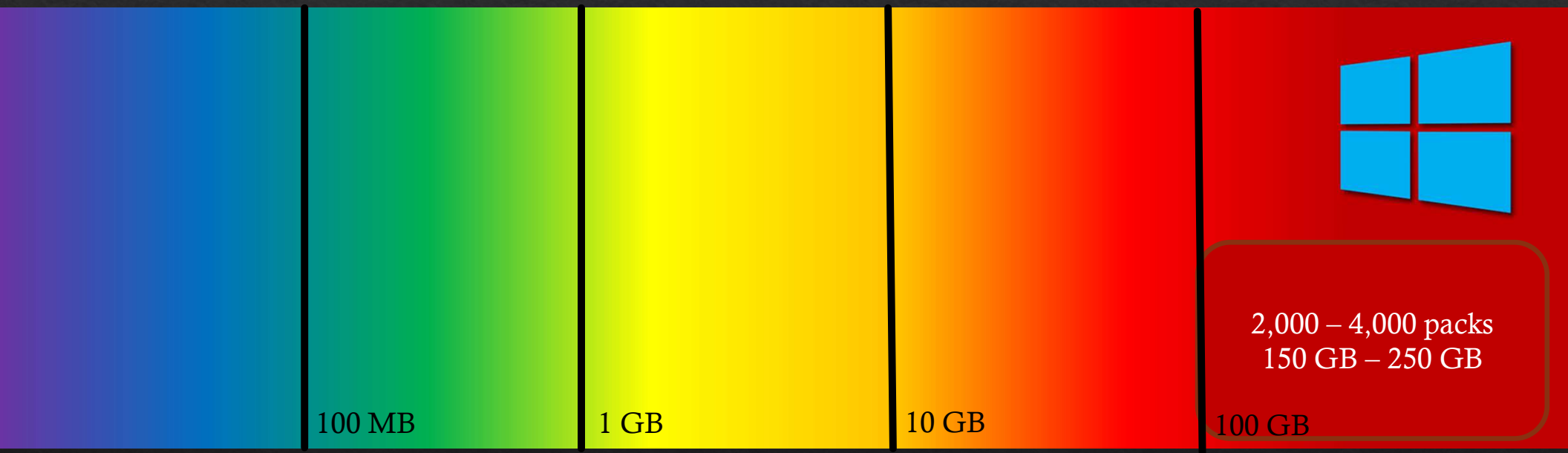
- **Background fetch:** get latest objects from remotes
- **Loose Objects:** Clean up loose objects safely
- **Pack-files:** Index and repack pack-files incrementally

Spectrum of Perceived Performance

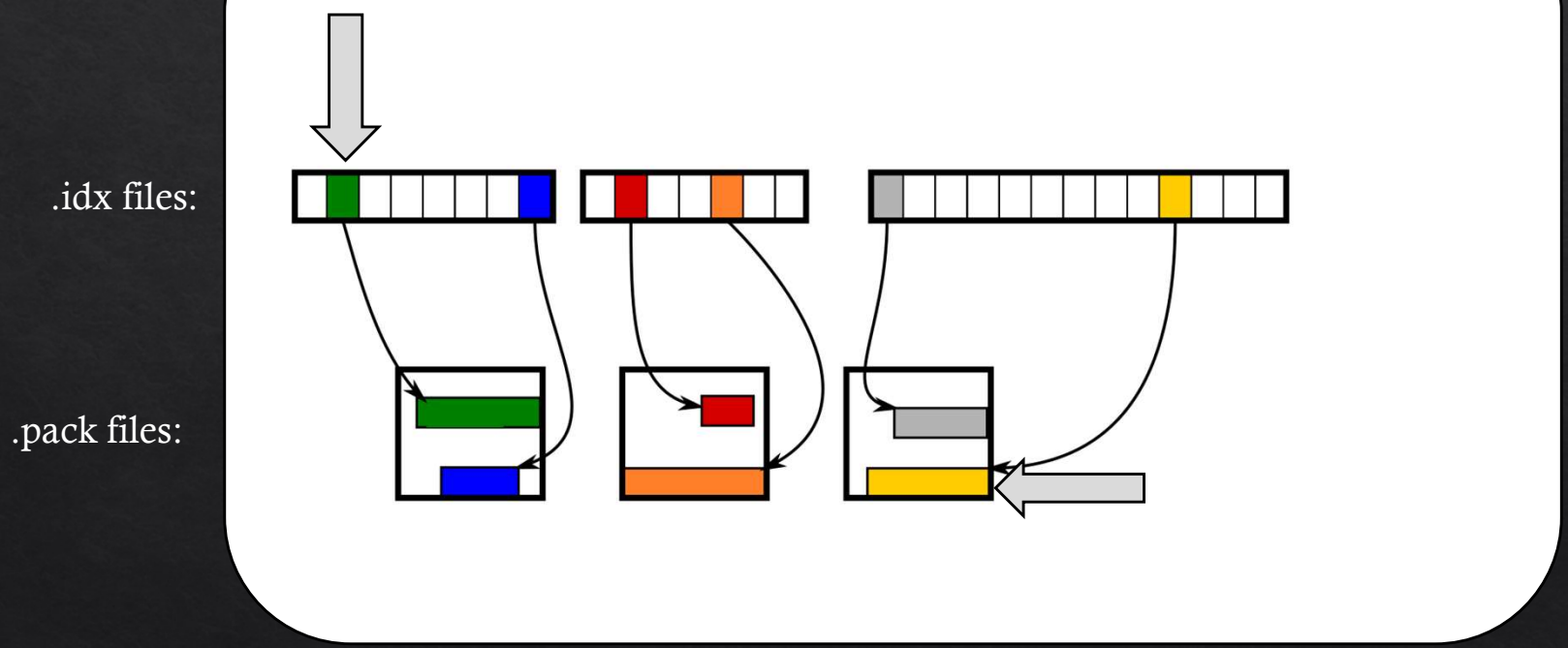


Foreground `git fetch` time

Too Many Packs?

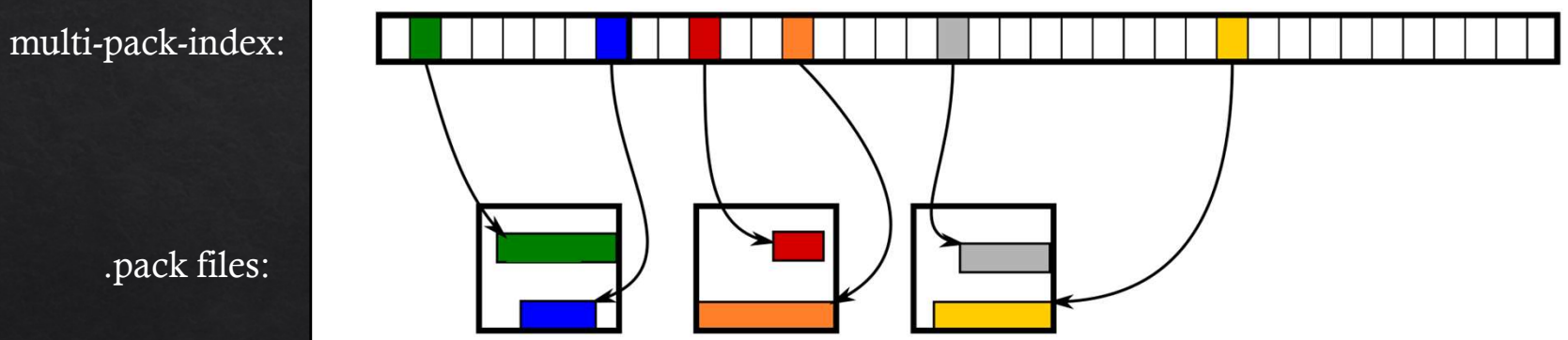


Too Many Packs?



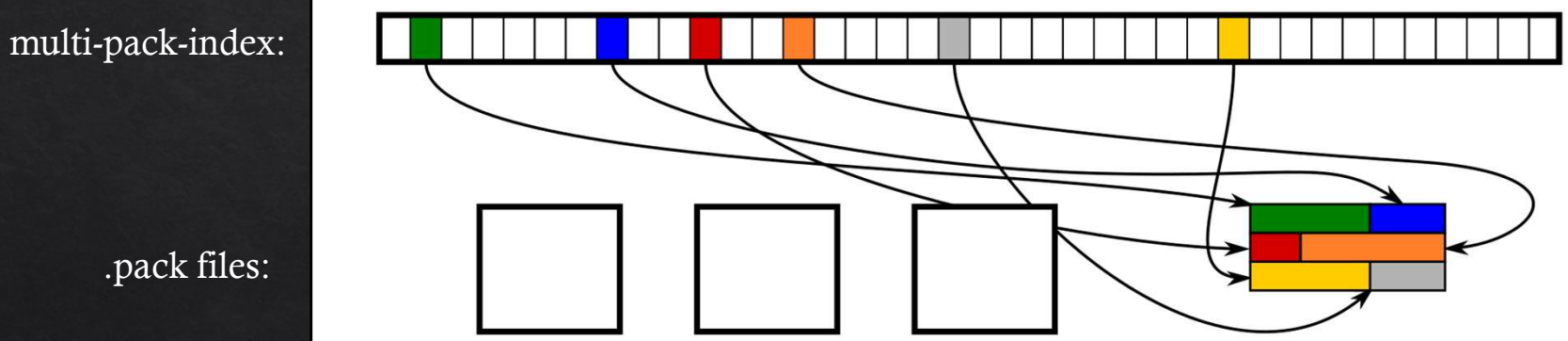
Too Many Packs?

```
git multi-pack-index write
```



Incremental Repack

```
git multi-pack-index repack
```

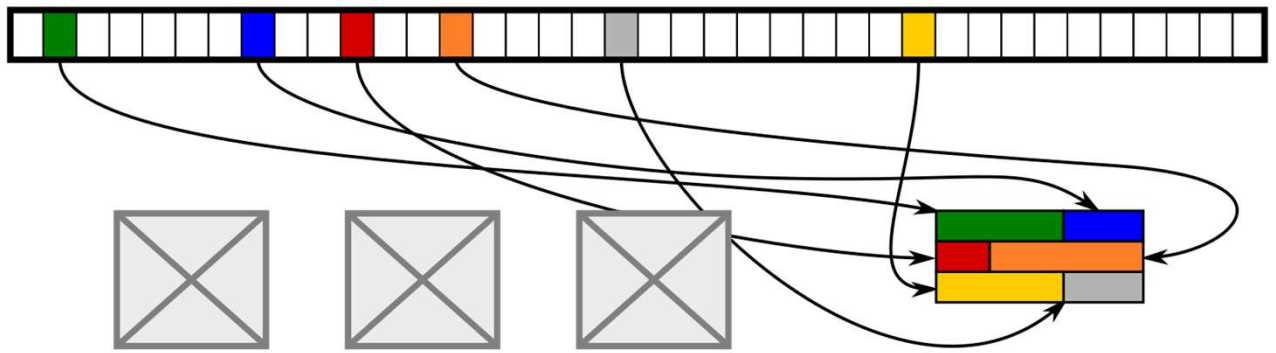


Incremental Repack

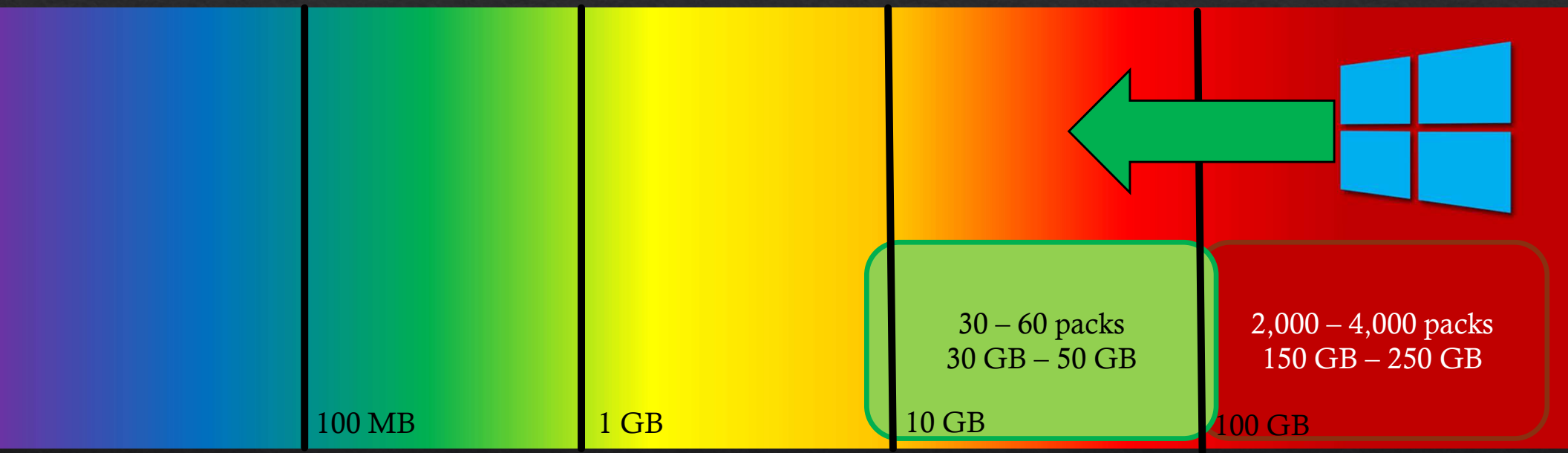
```
git multi-pack-index expire
```

multi-pack-index:

.pack files:



Spectrum of Scale



Effect of background `git multi-pack-index repack`

Background Maintenance in Git?

- *Should* Git do background maintenance?
- What of these background jobs make sense for most users?
- How might expert users want to customize these jobs? (Frequency, batch sizes, etc.)

https://github.com/microsoft/scalar

Installers available for Windows and macOS

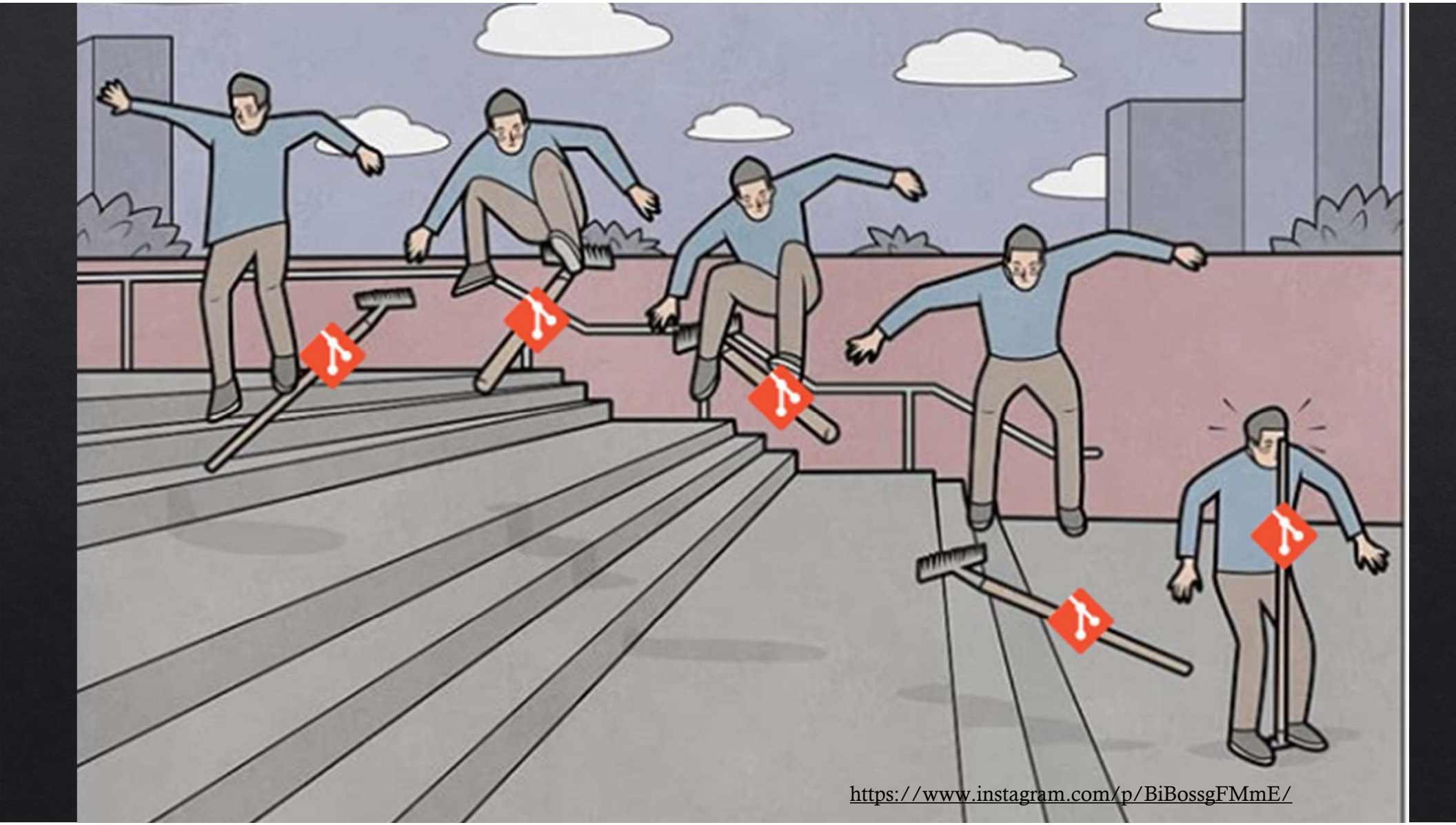
Scalar Quick Start

```
$ git version
git version 2.25.0.vfs.1.3

$ scalar version
scalar 20.01.165.4

$ scalar register
Successfully registered repo at ‘/Users/stolee/_git/vscode’
```

Demo: `scalar register`


https://www.instagram.com/p/BiBossgFMmE/

Demo: `scalar clone`

Scalar bridges the gap *for now*

Features coming to Git:

- Git-native filesystem monitor
- Git-native cache servers
- Background maintenance

https://github.com/microsoft/scalar

Installers available for Windows and macOS