

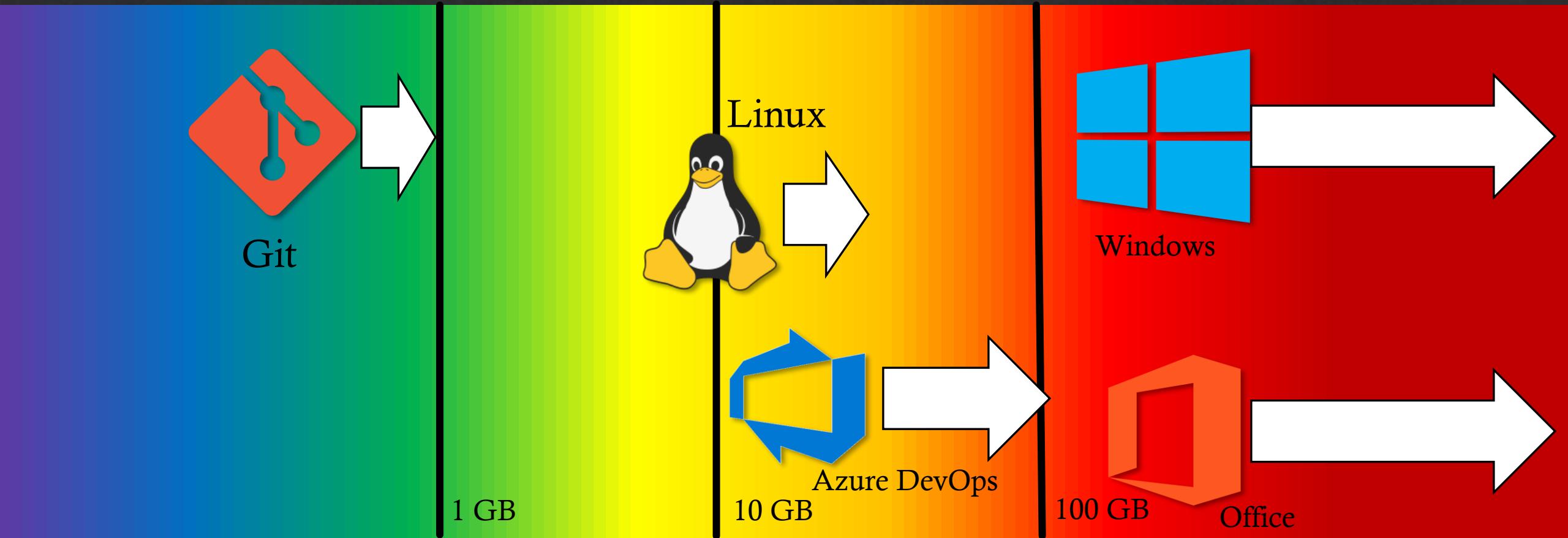
Git at Scale for Everyone

D. Stolee, Git Ecosystem Team, Microsoft

Twitter: @stolee GitHub: @derrickstolee

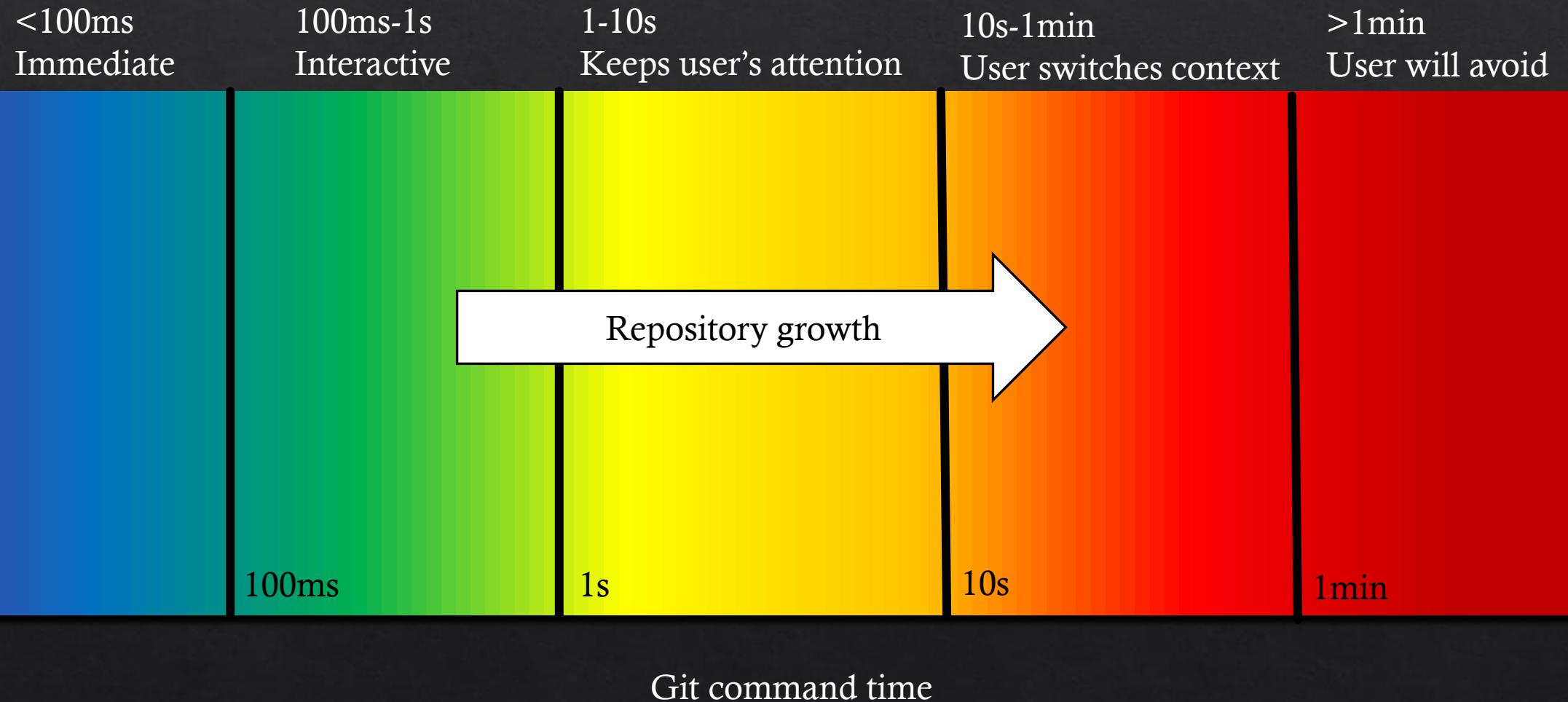
<https://stolee.dev/docs/git-merge-2020.pdf>

Spectrum of Scale



Pack-file size for initial clone

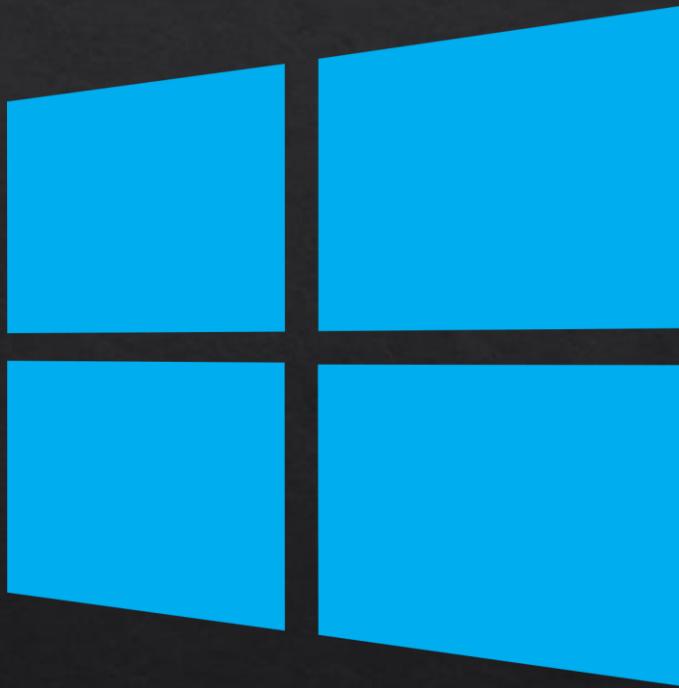
Spectrum of Perceived Performance





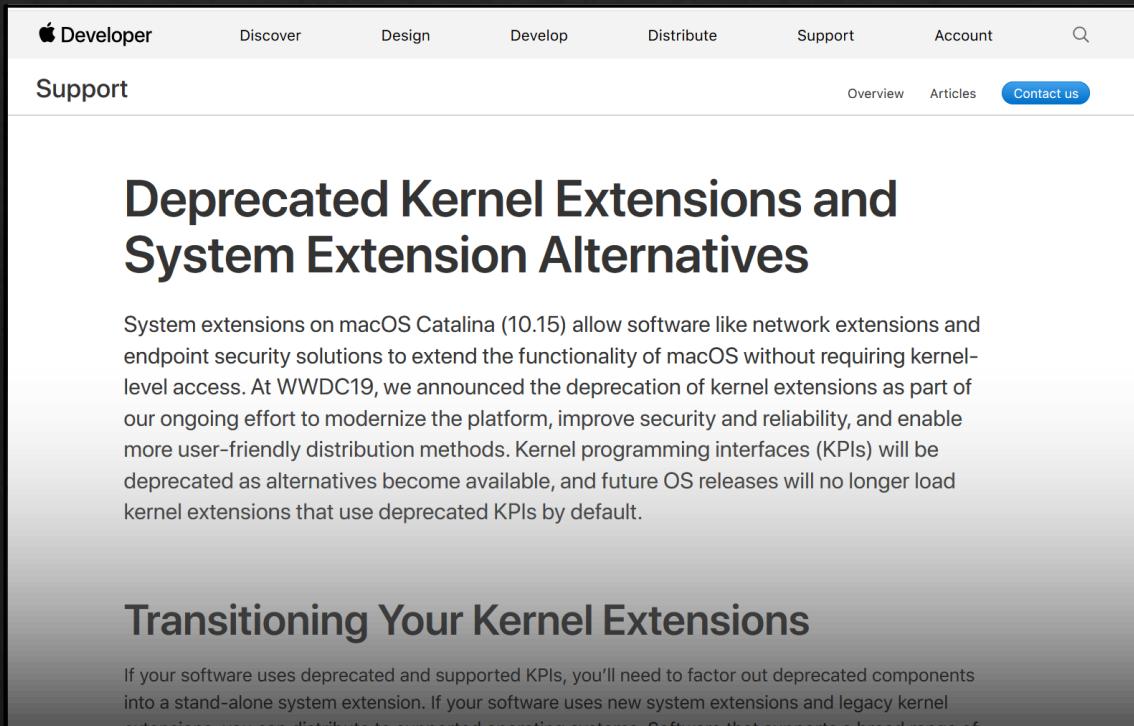
Success Story: Microsoft Windows

- ❖ Largest Git repository
- ❖ VFS for Git enabled using it *at all*
 - ❖ Virtualized filesystem “fakes” working directory updates
- ❖ Measuring real user interactions showed need for Git performance improvements
- ❖ Delivered most improvements as contributions to core Git client



Next Milestone: Microsoft Office

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



The screenshot shows a web browser window with the Apple Developer website. The navigation bar includes links for Developer, Discover, Design, Develop, Distribute, Support, and Account, along with a search icon. The main content area is titled "Support" and features a large heading "Deprecated Kernel Extensions and System Extension Alternatives". Below the heading, a paragraph explains the deprecation of kernel extensions on macOS Catalina (10.15) and the introduction of system extensions as alternatives. A section titled "Transitioning Your Kernel Extensions" is visible at the bottom.

Deprecated Kernel Extensions and System Extension Alternatives

System extensions on macOS Catalina (10.15) allow software like network extensions and endpoint security solutions to extend the functionality of macOS without requiring kernel-level access. At WWDC19, we announced the deprecation of kernel extensions as part of our ongoing effort to modernize the platform, improve security and reliability, and enable more user-friendly distribution methods. Kernel programming interfaces (KPIs) will be deprecated as alternatives become available, and future OS releases will no longer load kernel extensions that use deprecated KPIs by default.

Transitioning Your Kernel Extensions

If your software uses deprecated and supported KPIs, you'll need to factor out deprecated components into a stand-alone system extension. If your software uses new system extensions and legacy kernel extensions, you can distribute to supported execution systems. Software that supports a broad range of





Scalar

<https://github.com/microsoft/scalar>

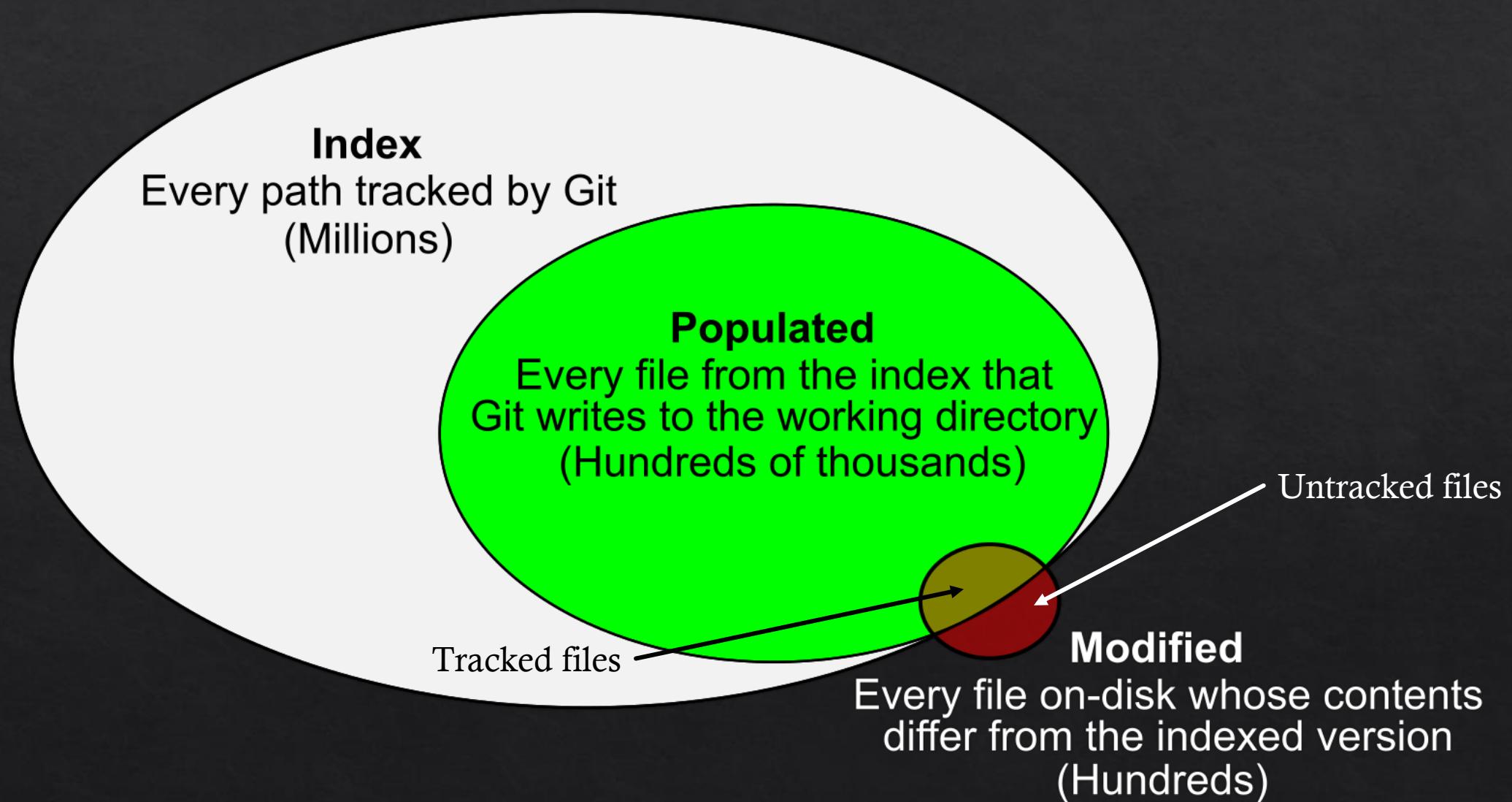
Lessons for Git at Scale

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

To control the number of files in your working directory, run

```
git sparse-checkout init --cone
```

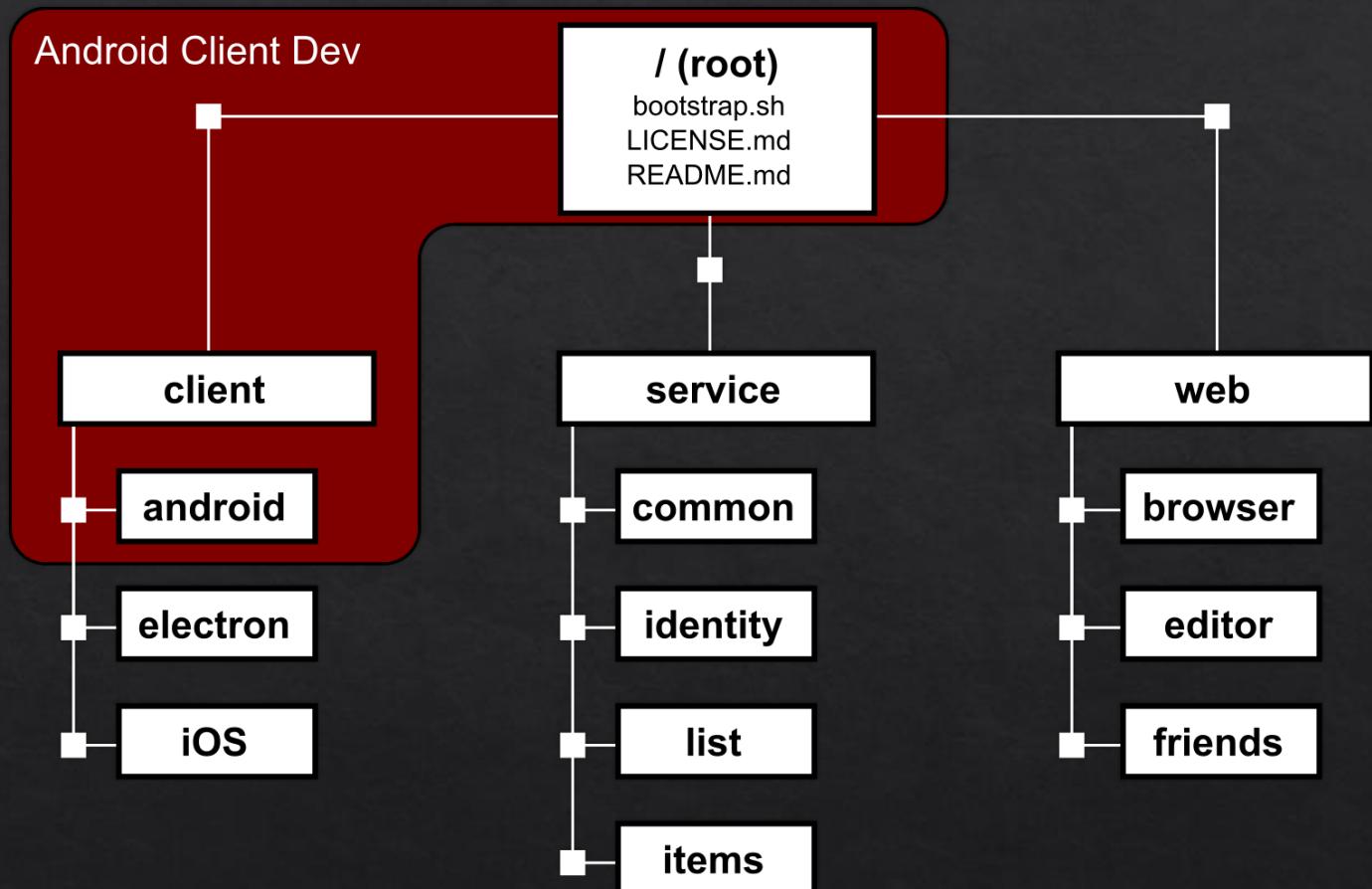
initializes sparse-checkout in “cone mode”. This starts with only the files at root.

Included paths can be expanded using

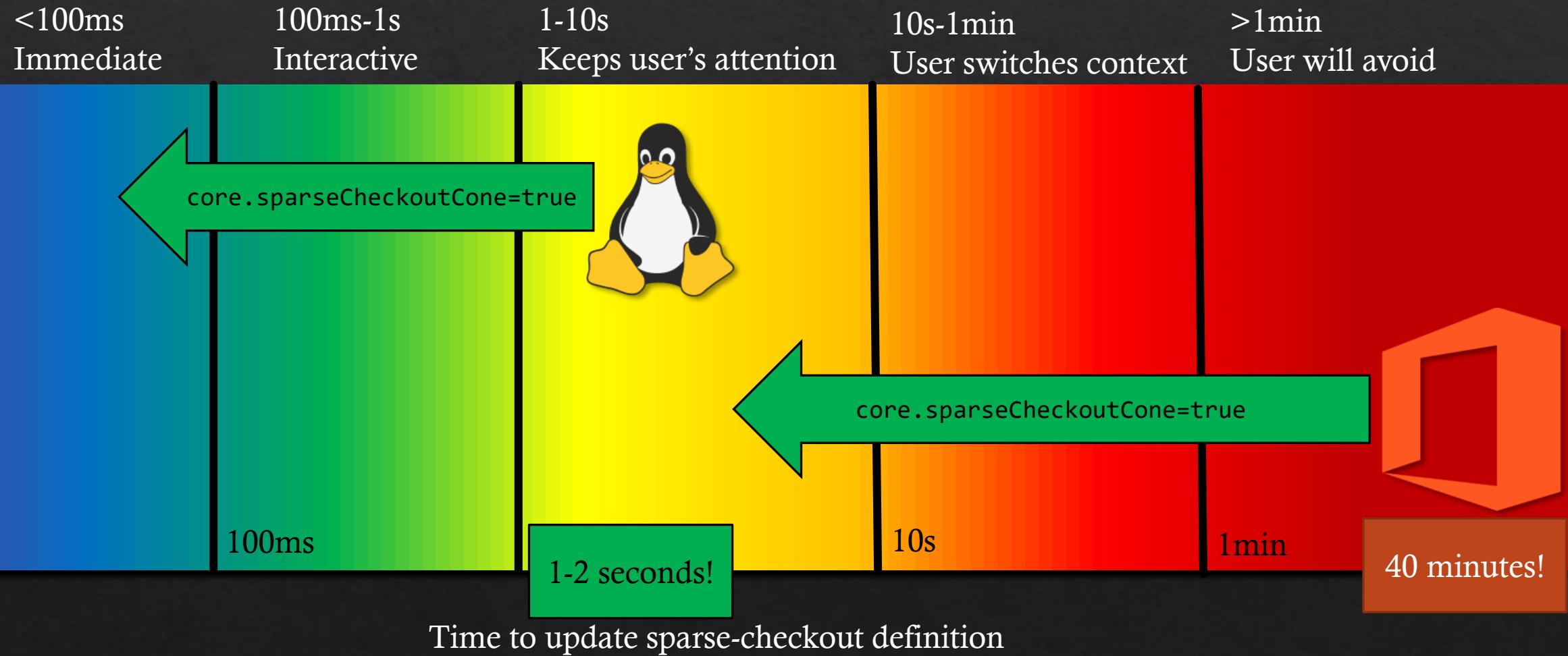
```
git sparse-checkout set <dir1> <dir2> ...
```

In this example, we use:

```
git sparse-checkout set client/android
```



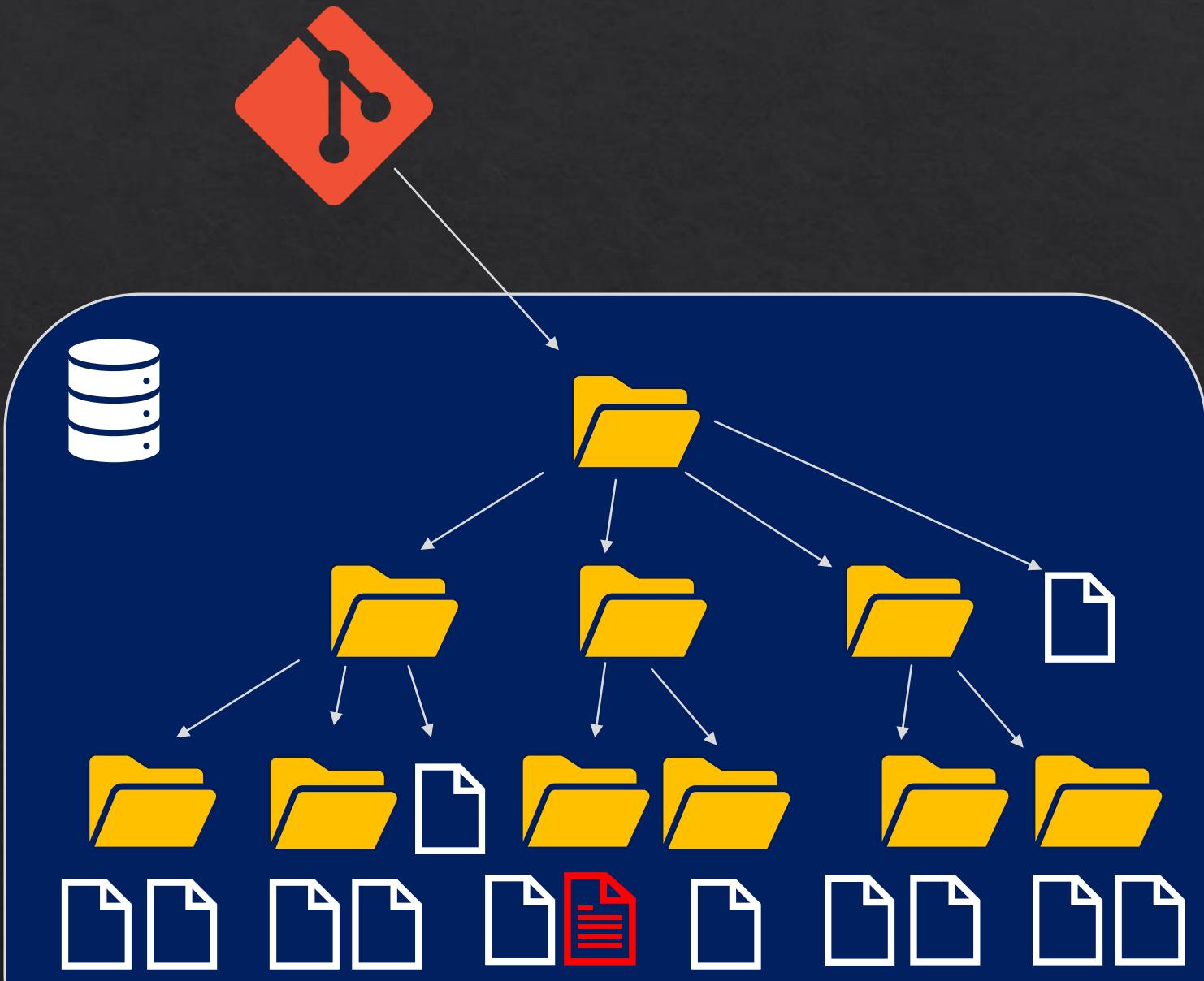
Spectrum of Perceived Performance



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.

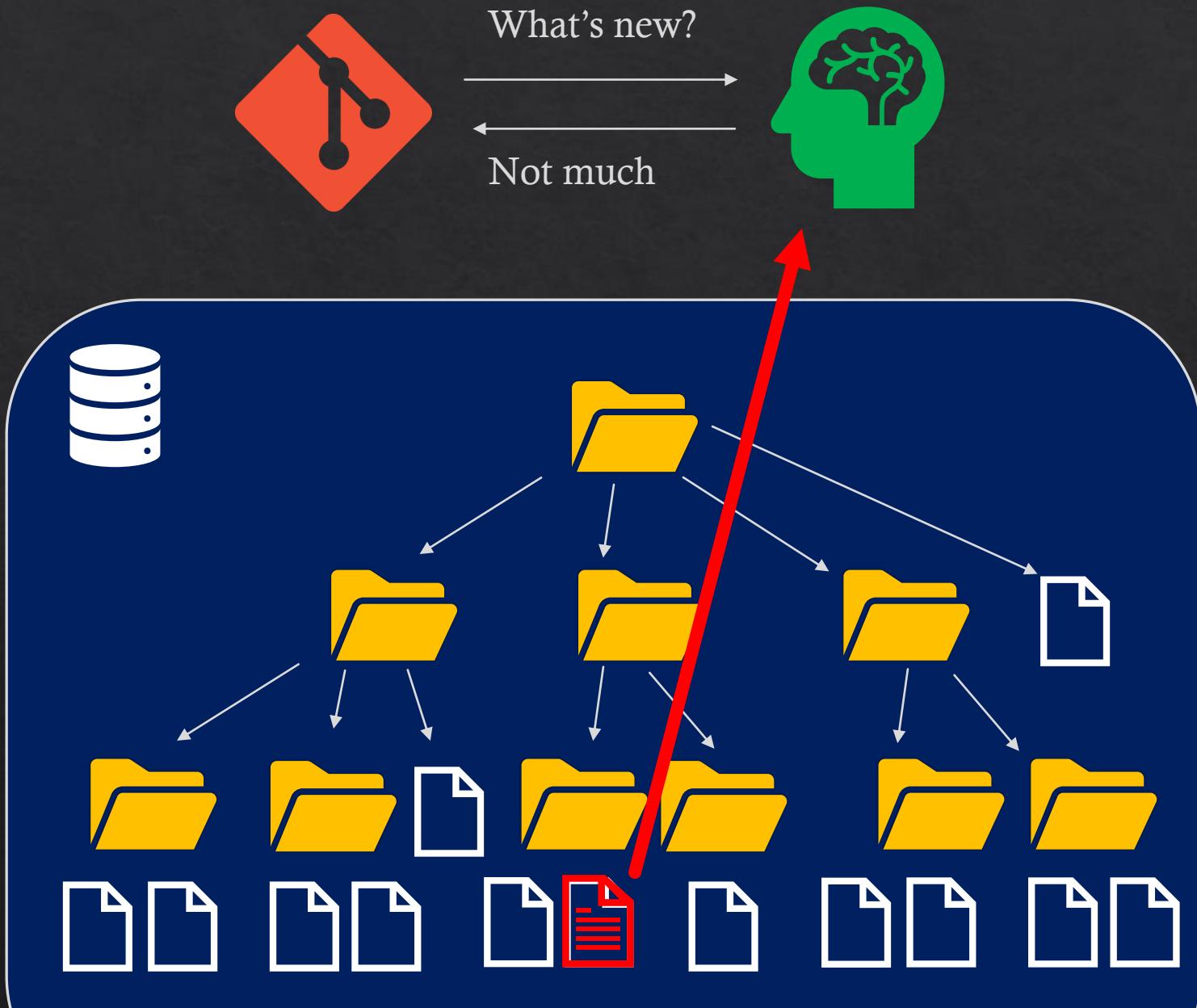
This usually results in scanning directories.

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

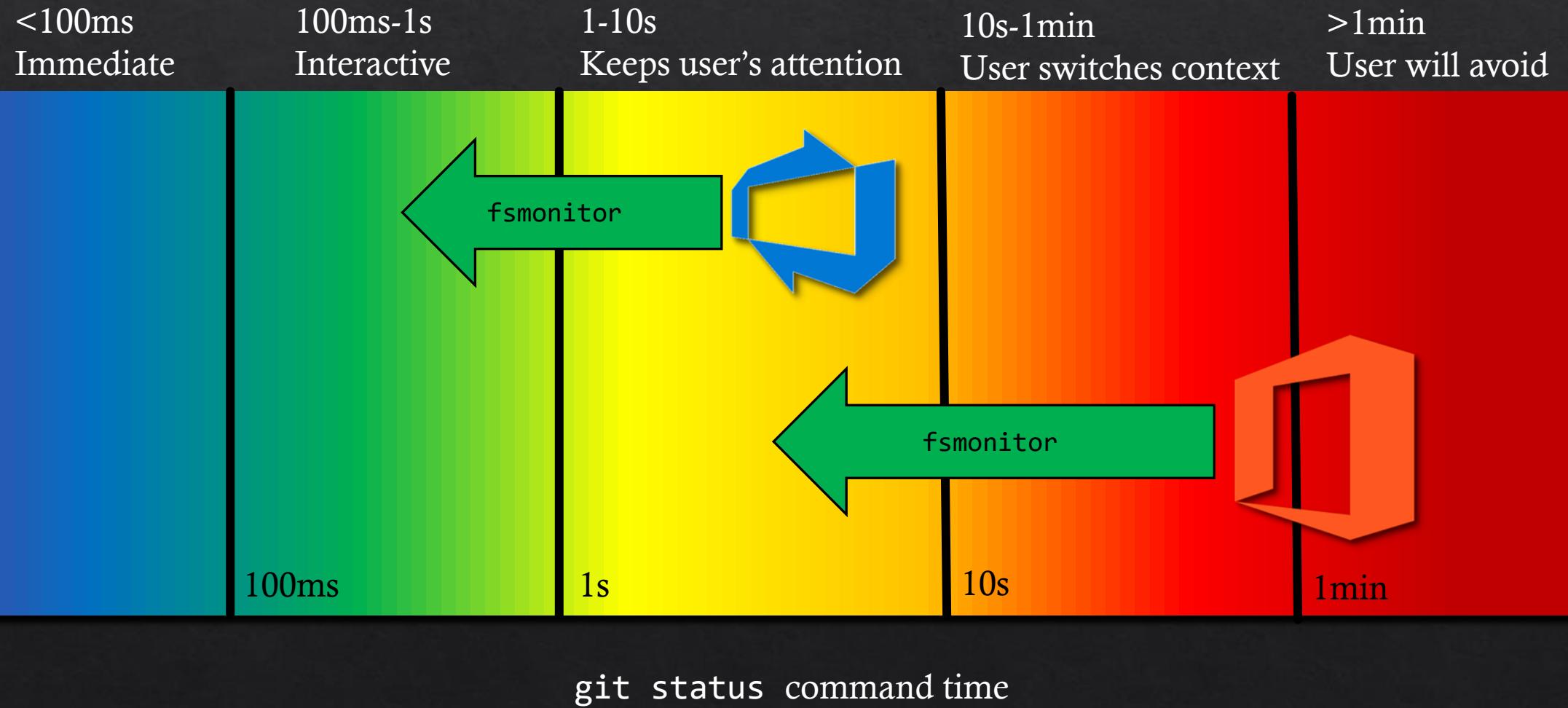
<https://github.com/facebook/watchman>

What's new?

Not much



Spectrum of Perceived Performance



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@dstolee-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

<100ms	100ms-1s	1-10s	10s-1min	>1min
Immediate	Interactive	Keeps user's attention	User switches context	User will avoid

100ms

1s

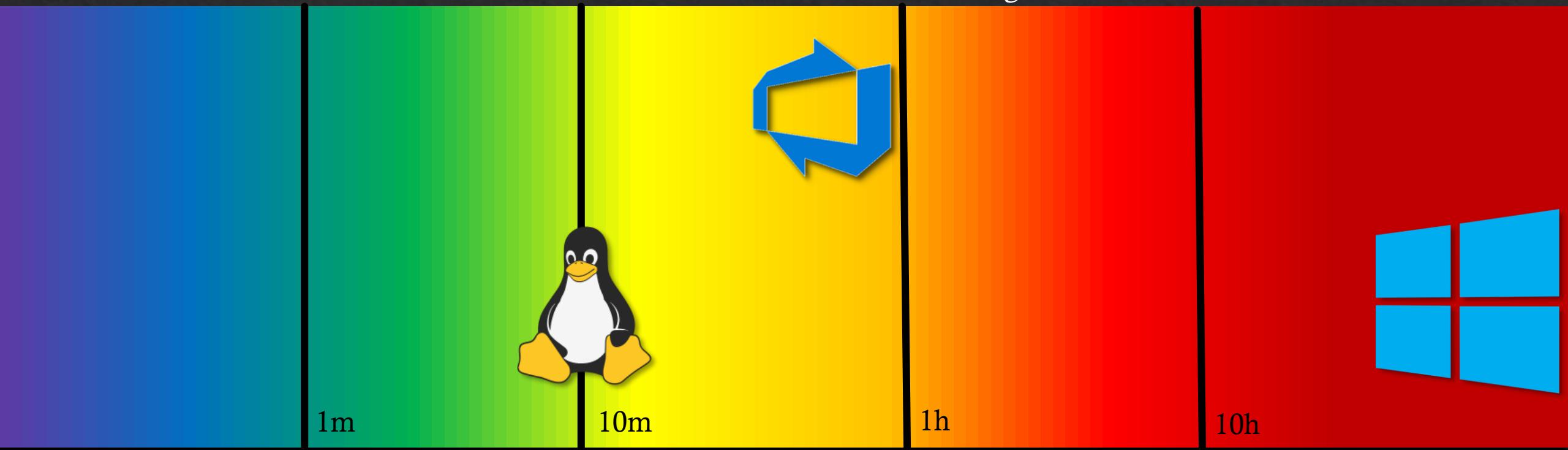
10s

1min

git clone time

Spectrum of Perceived Performance

<1m	1m-10m	10m-1h	1h-10h	>10h
Feels fast	Feels slow	Over lunch break	Overnight	User will avoid



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

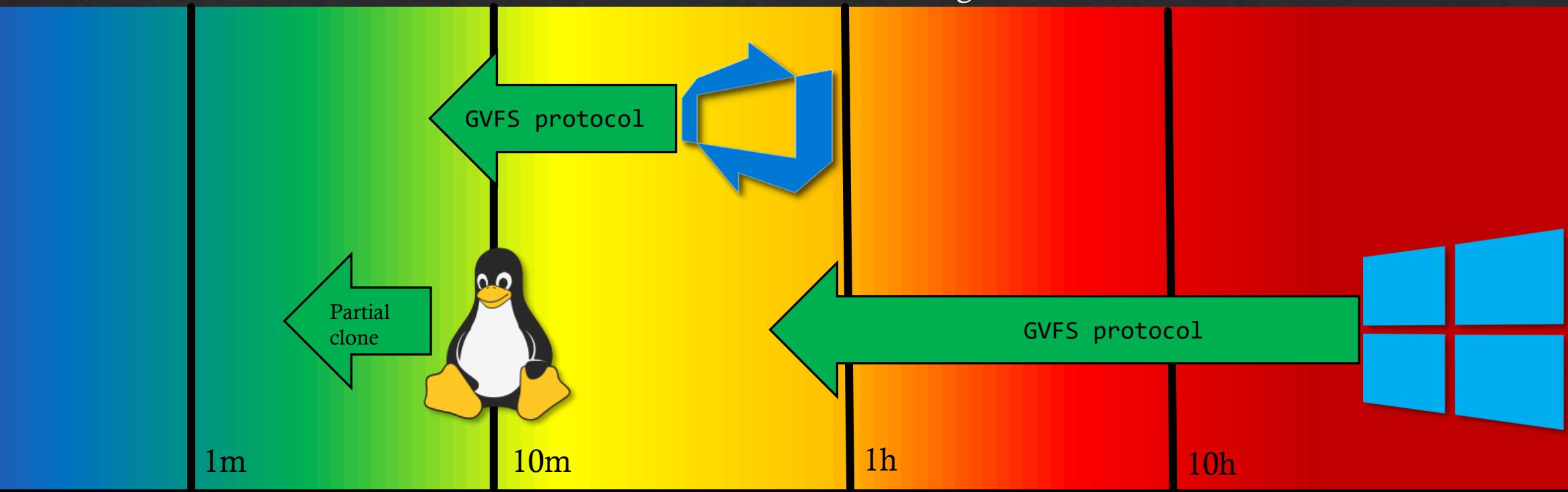
Now available on all GitHub.com repositories!

Reduced Object Transfer + Sparse-Checkout = Success!

<https://git-scm.com/docs/partial-clone>

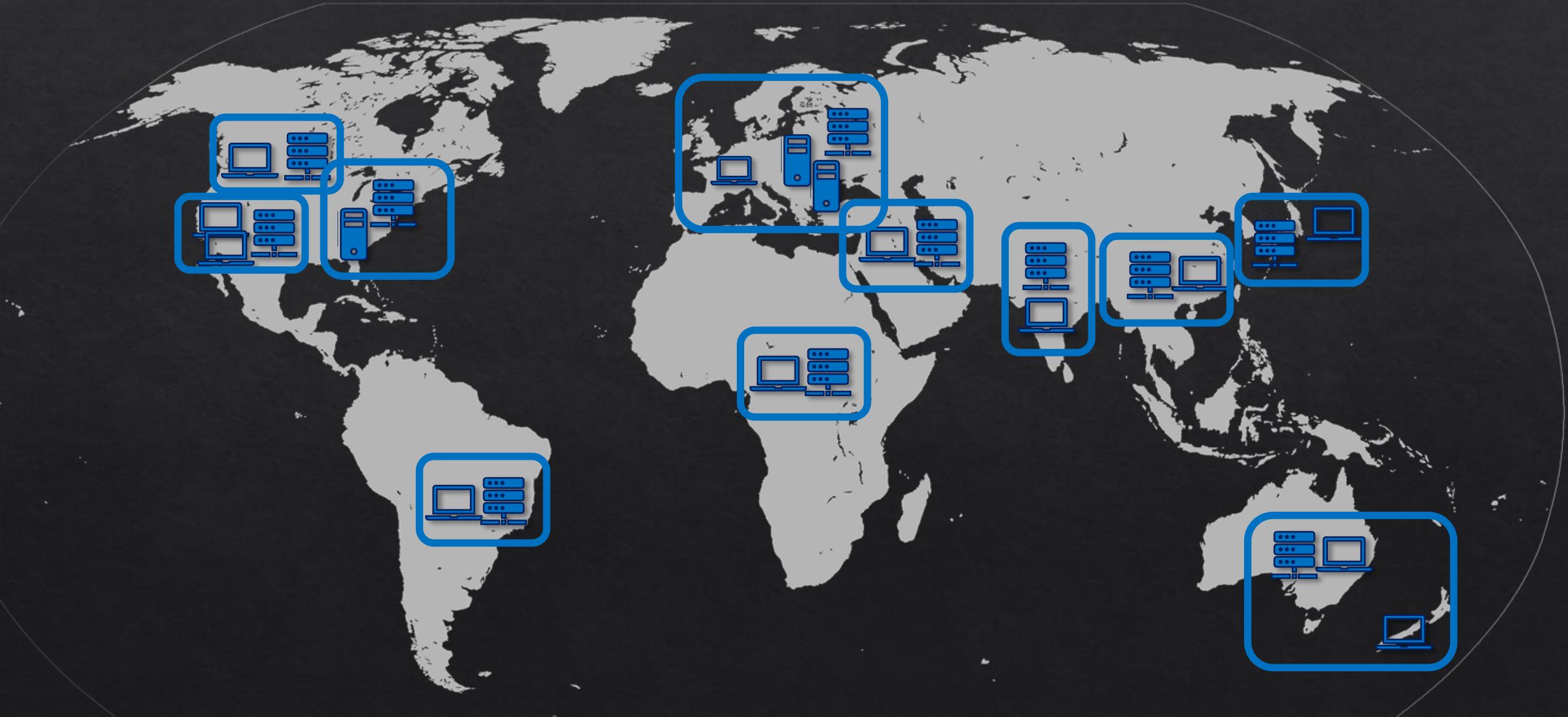
Spectrum of Perceived Performance

<1m	1m-10m	10m-1h	1h-10h	>10h
Feels fast	Feels slow	Over lunch break	Overnight	User will avoid



Time for git clone vs partial clone or GVFS protocol

GVFS Cache Servers and Git Promisor Remotes



Recommended Updates to Partial Clone

1. Extend multiple promisor remotes to do commit and tree fetches.
2. Extend Git protocol to assist auto-discovery of nearby promisor remotes

Lesson 3: Don't wait for expensive operations

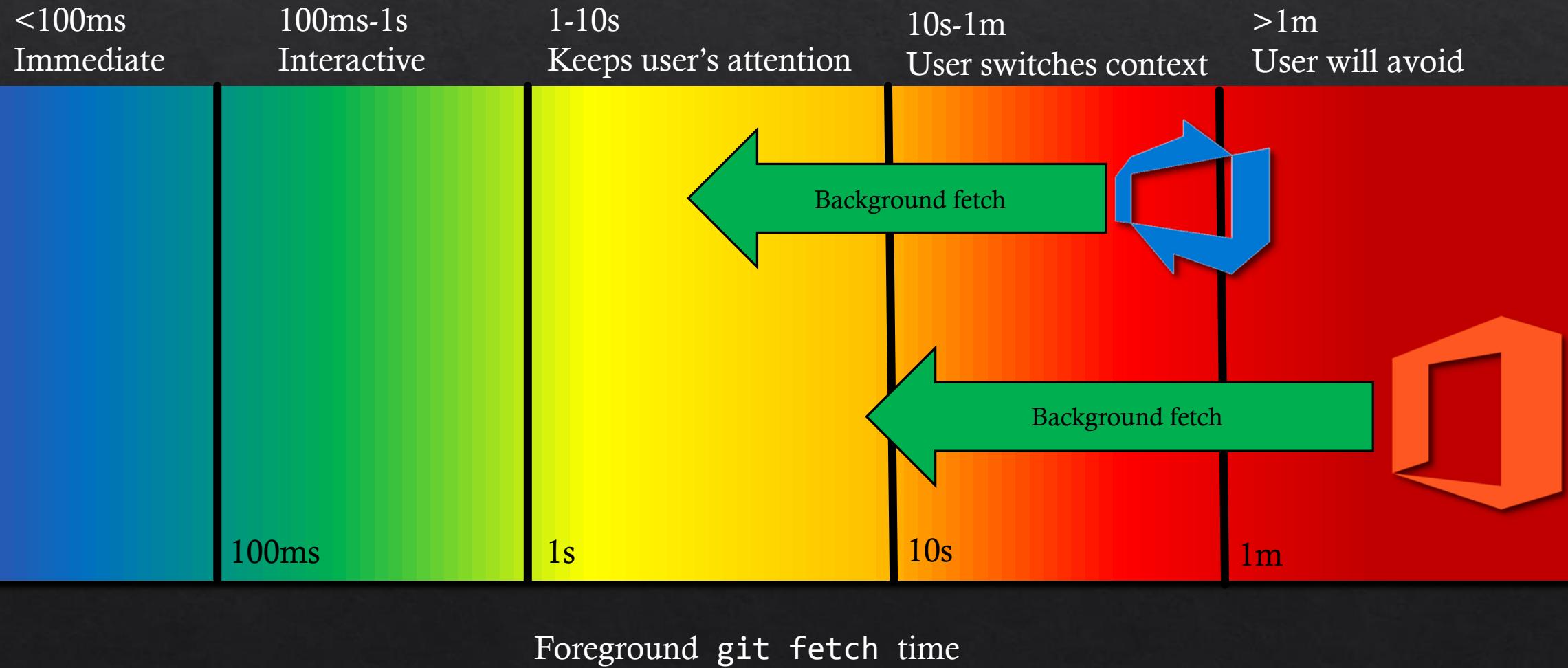


Background Maintenance

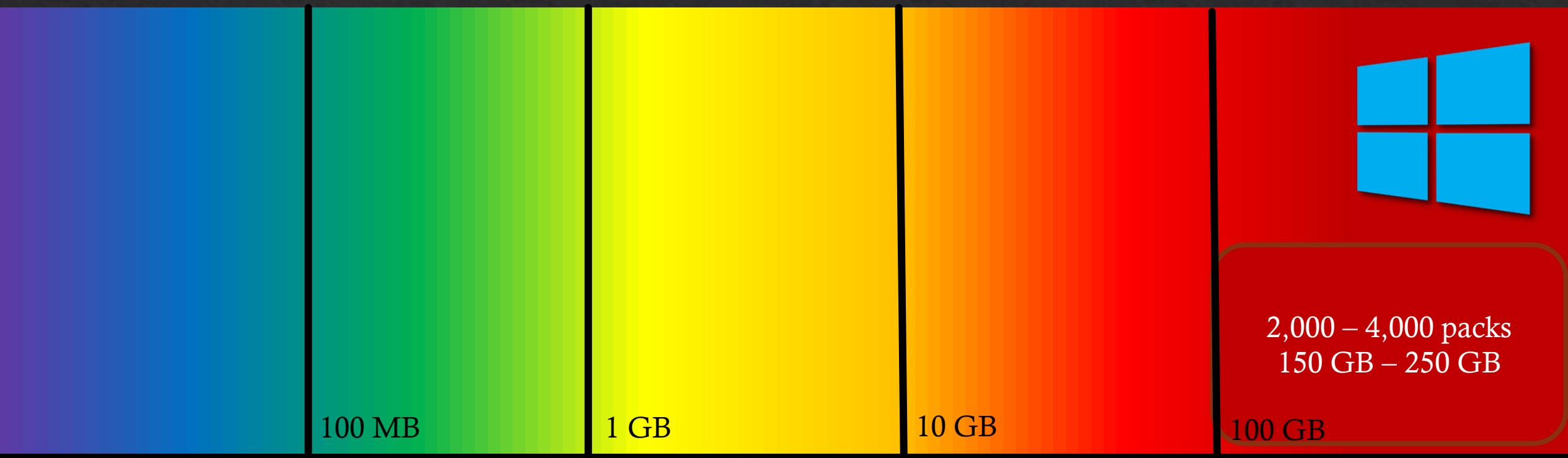
The following can be done in the background, reducing user-blocking time:

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

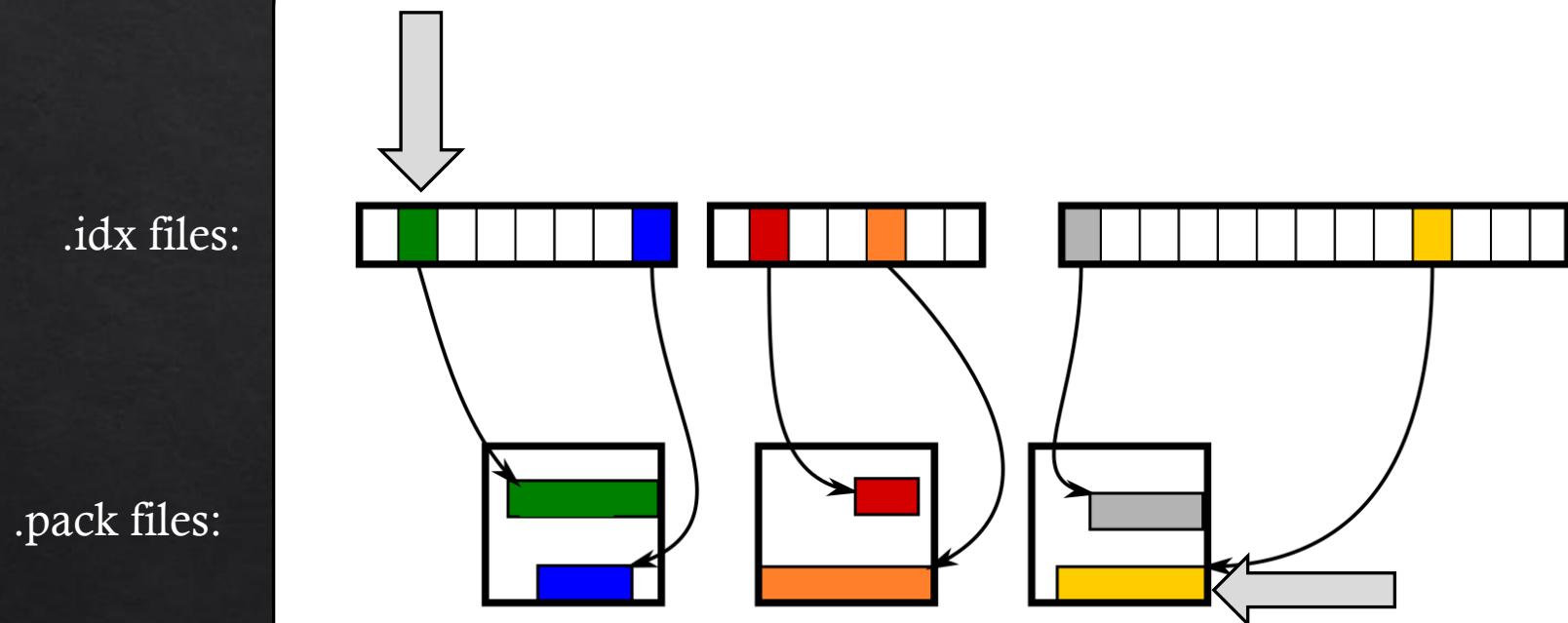
Spectrum of Perceived Performance



Too Many Packs?



Too Many Packs?



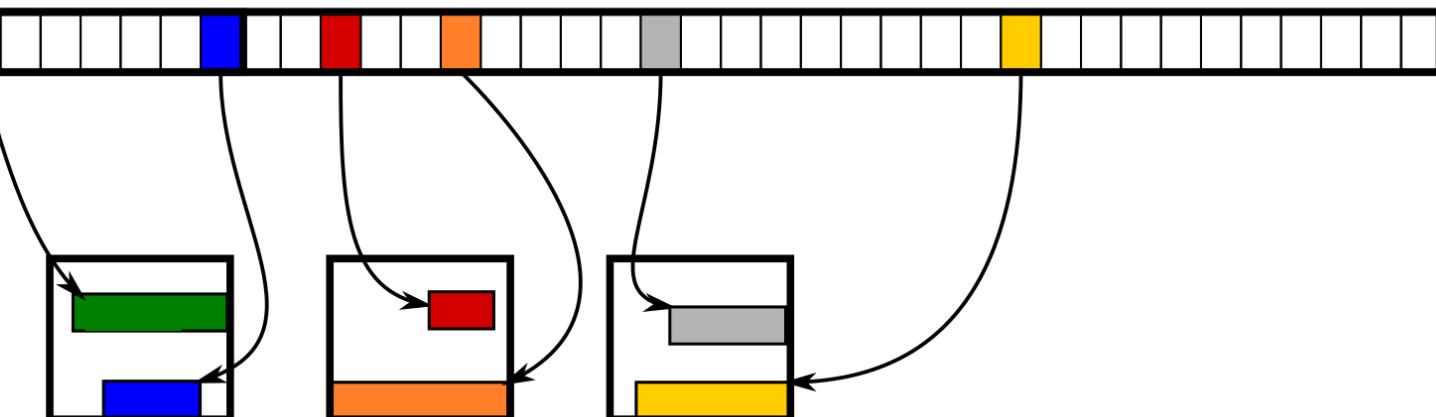
Too Many Packs?

`git multi-pack-index write`

multi-pack-index:



.pack files:



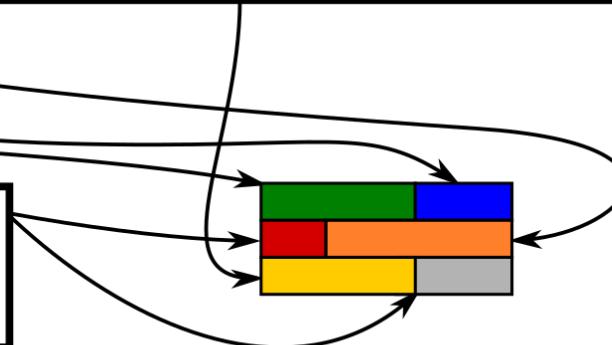
Incremental Repack

`git multi-pack-index repack`

multi-pack-index:



.pack files:



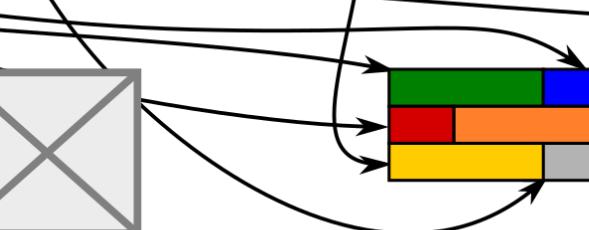
Incremental Repack

`git multi-pack-index expire`

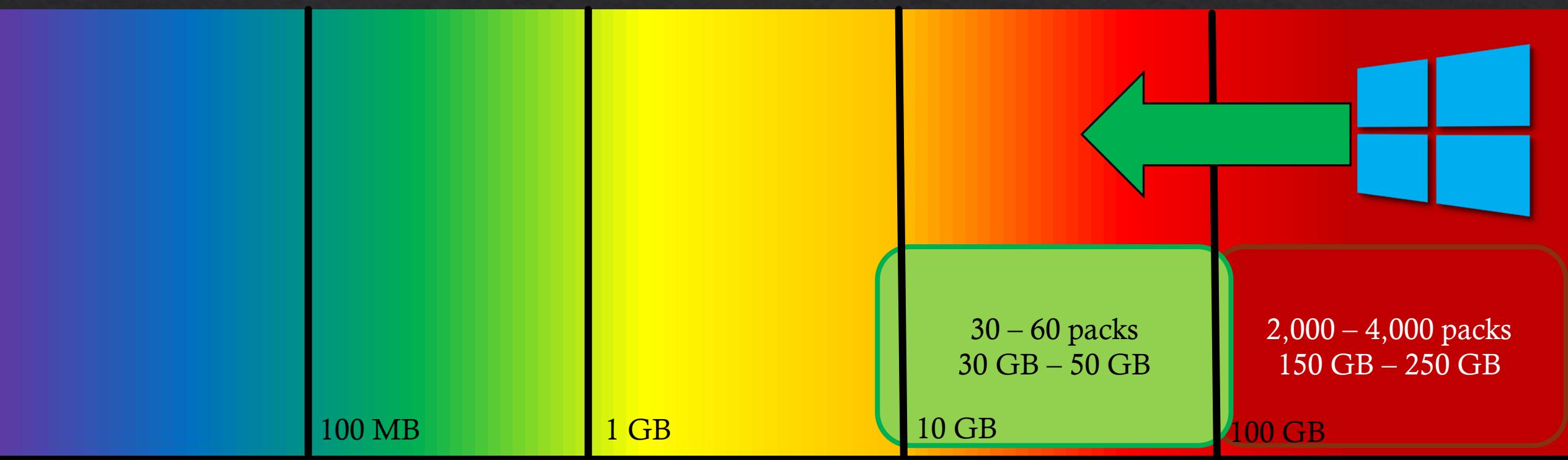
multi-pack-index:



.pack files:



Spectrum of Scale



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.)



Scalar

<https://github.com/microsoft/scalar>

Installers available for Windows and macOS

Scalar Quick Start

```
$ git version
```

```
git version 2.25.1.vfs.1.2
```

```
$ scalar version
```

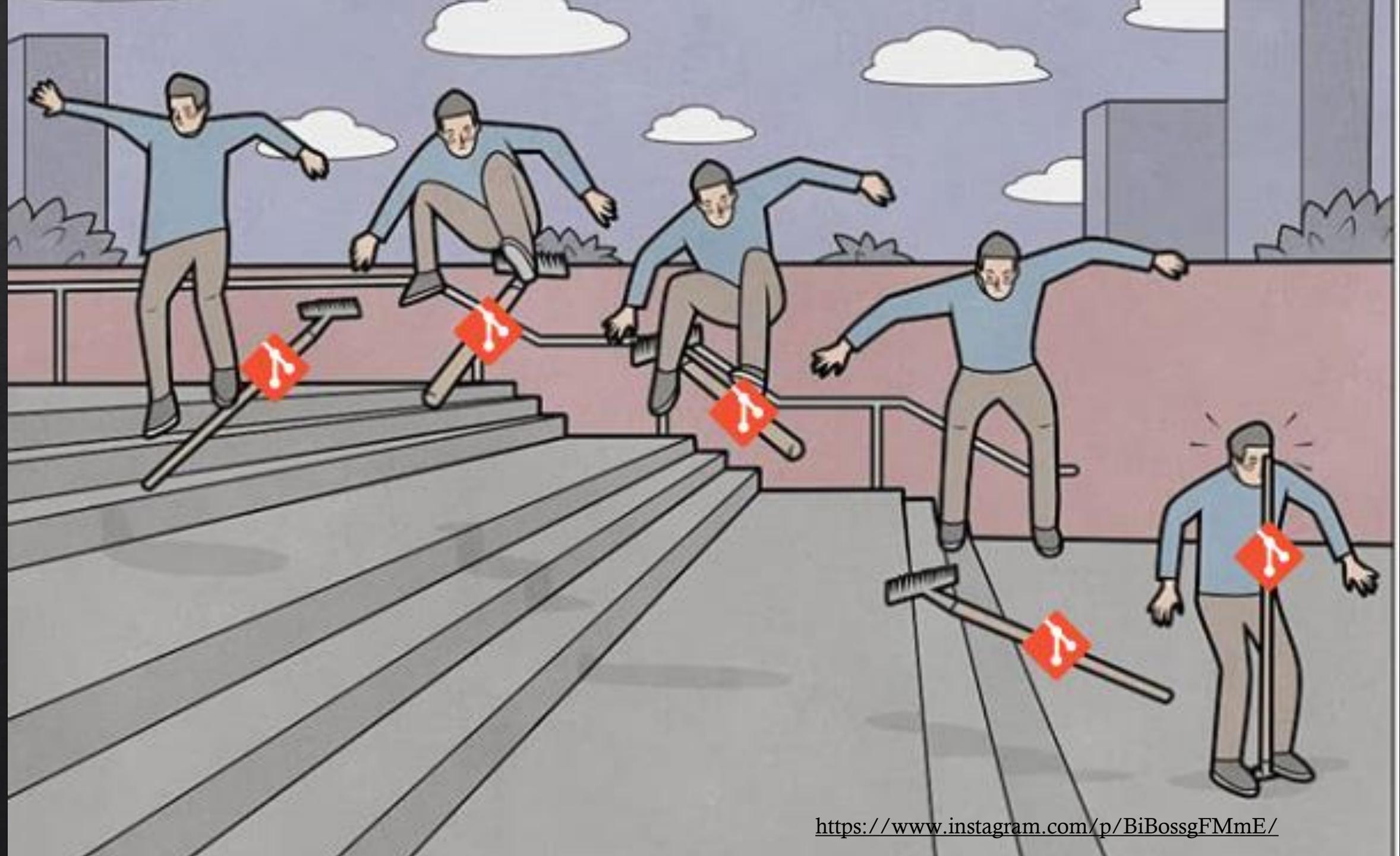
```
scalar 20.03.167.1
```

```
$ scalar register
```

```
Successfully registered repo at '/Users/stolee/_git/vscode'
```

What does scalar register do?

1. Sets advanced Git config settings for optimal performance
2. Initializes filesystem monitor hook, if Watchman is installed
<https://github.com/facebook/watchman>
3. Starts background maintenance
 1. Background fetch
 2. Write commit-graph
 3. Clean up loose objects
 4. Clean up pack-files



What does `scalar clone` do?

1. Creates new repository with working directory `<name>/src`
2. If remote supports **GVFS protocol**, then configure to use it.
3. Otherwise, configures Git to use **partial clone**.
4. Downloads all commits and trees.
5. `git sparse-checkout init --cone`
6. Everything from `scalar register`

Scalar bridges
the gap *for now*

Hopefully, one day Scalar will set
recommended config *and that's it.*



Scalar

<https://github.com/microsoft/scalar>

Installers available for Windows and macOS