Code Archeology

17-313: Foundations of Software Engineering

https://cmu-313.github.io

Michael Hilton and Josh Sunshine

Spring 2026

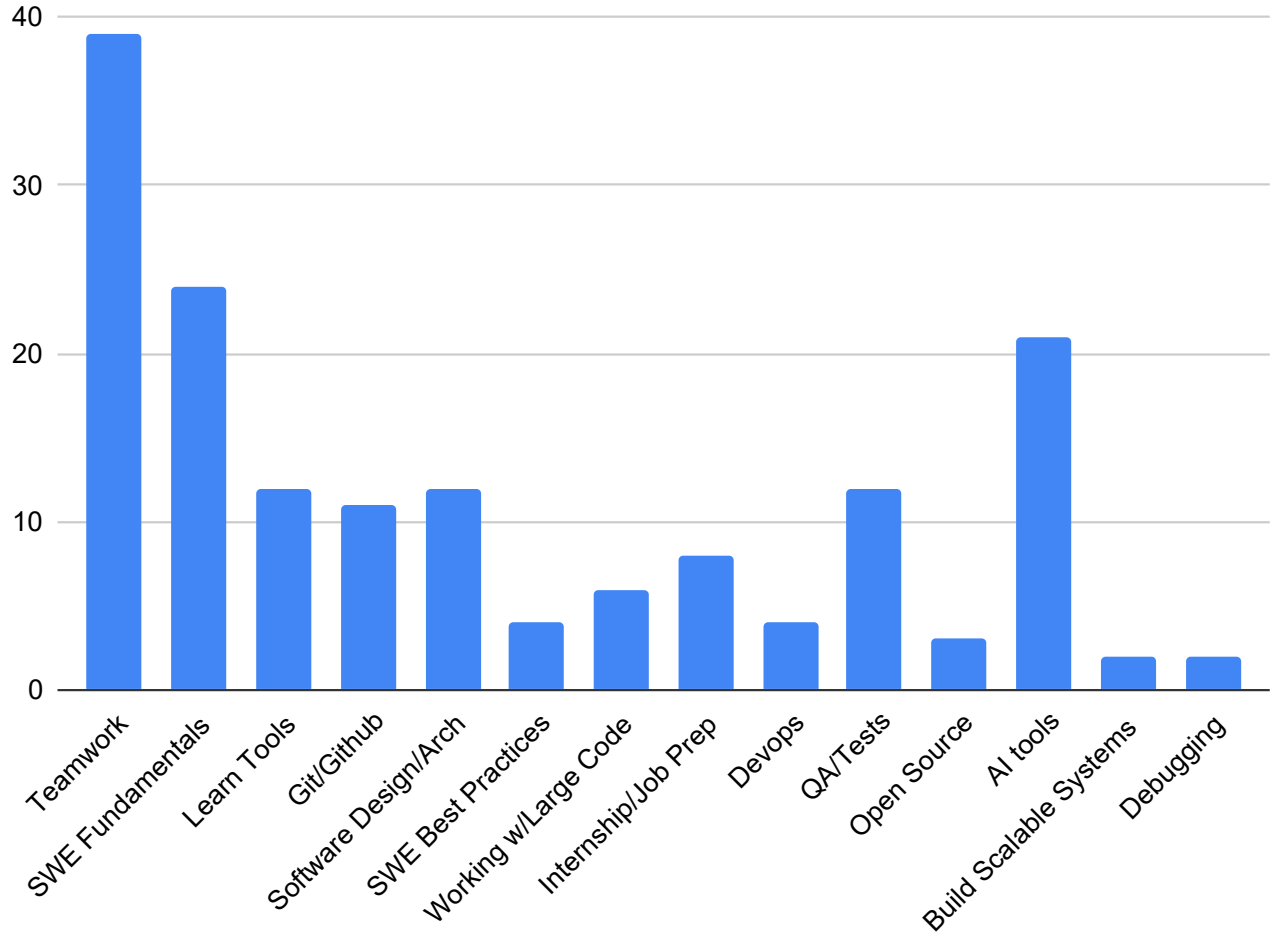
Administrivia

- **Project 1A - Build Checkpoint**
  Due Friday, January 16, 2026 @ 11:59PM
- If you haven't: **PLEASE FILL OUT TEAMWORK SURVEY!**
  - Posted to slack and canvas
- Get started early, ask for help, and check the #technical-support channel; chances are your questions have been asked by others!
- NO RECITATION MONDAY (MLK Day)

Smoking Section

- Last full row

Hope to learn (from last class)

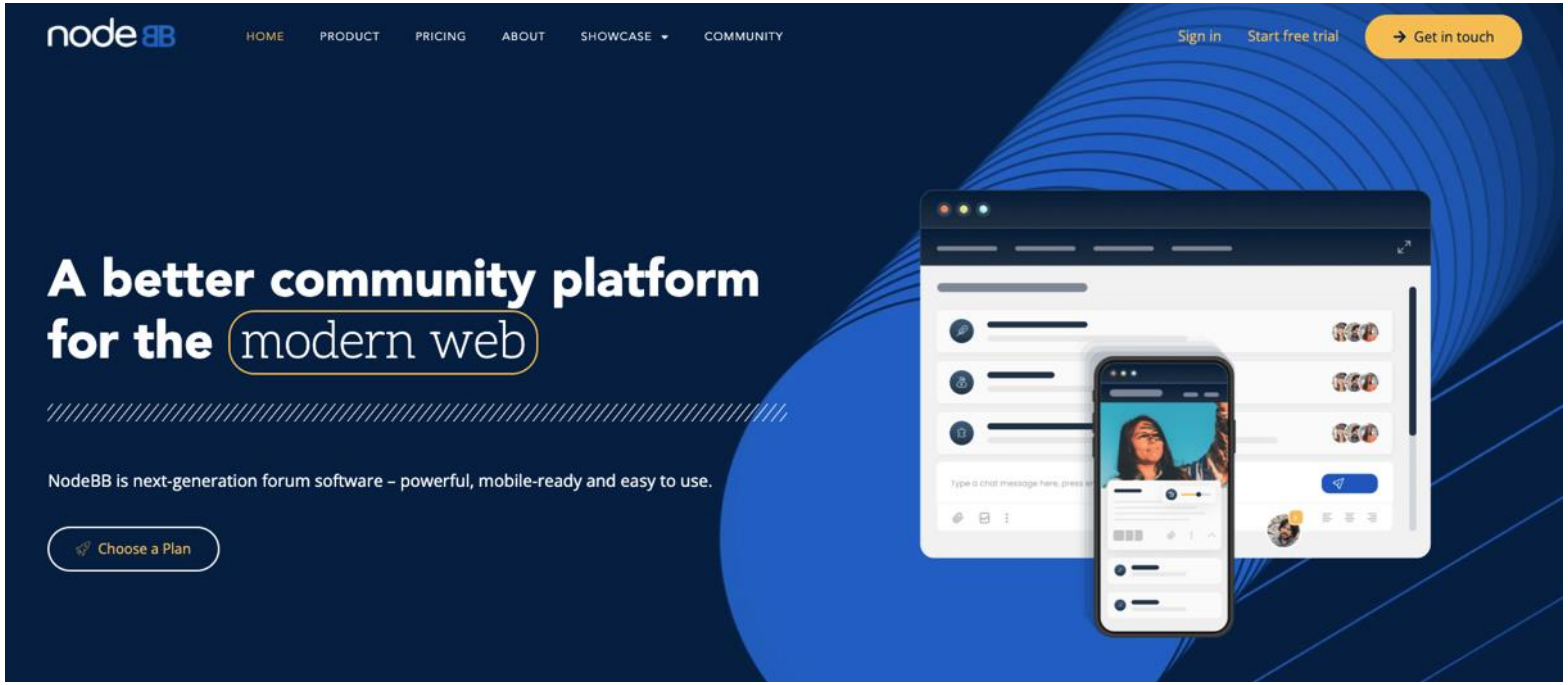


Slack Usage:

- Lots of great help for each other on #technicalsupport, keep up the good work!
  - use ✅ emoji to signal thread is answered
- Please Search before asking new questions
- Please put a picture of your face!!
- We don’t guarantee round the clock availability

Context: big ole pile of code

- … do something with it!



Step 1:
Get Code

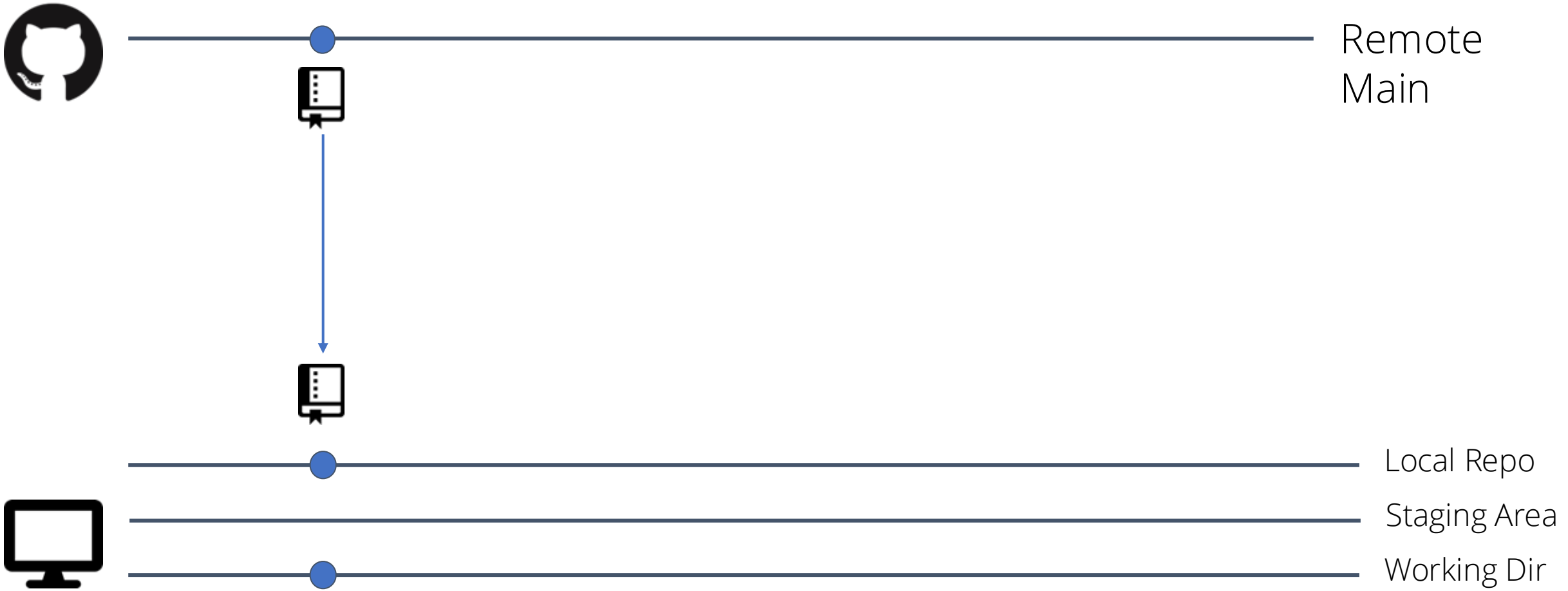
Step 1:
~~Get Code~~ Git Code

Version Control

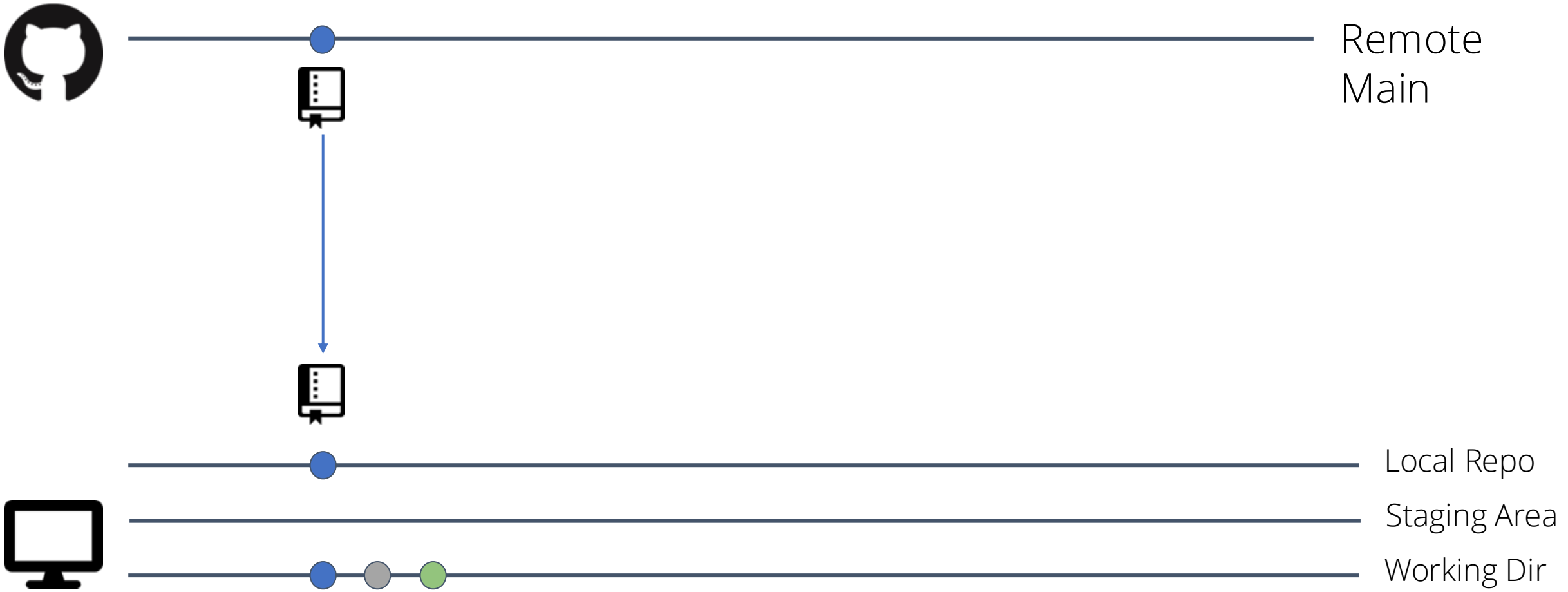
Git Fundamentals

> git clone REPO

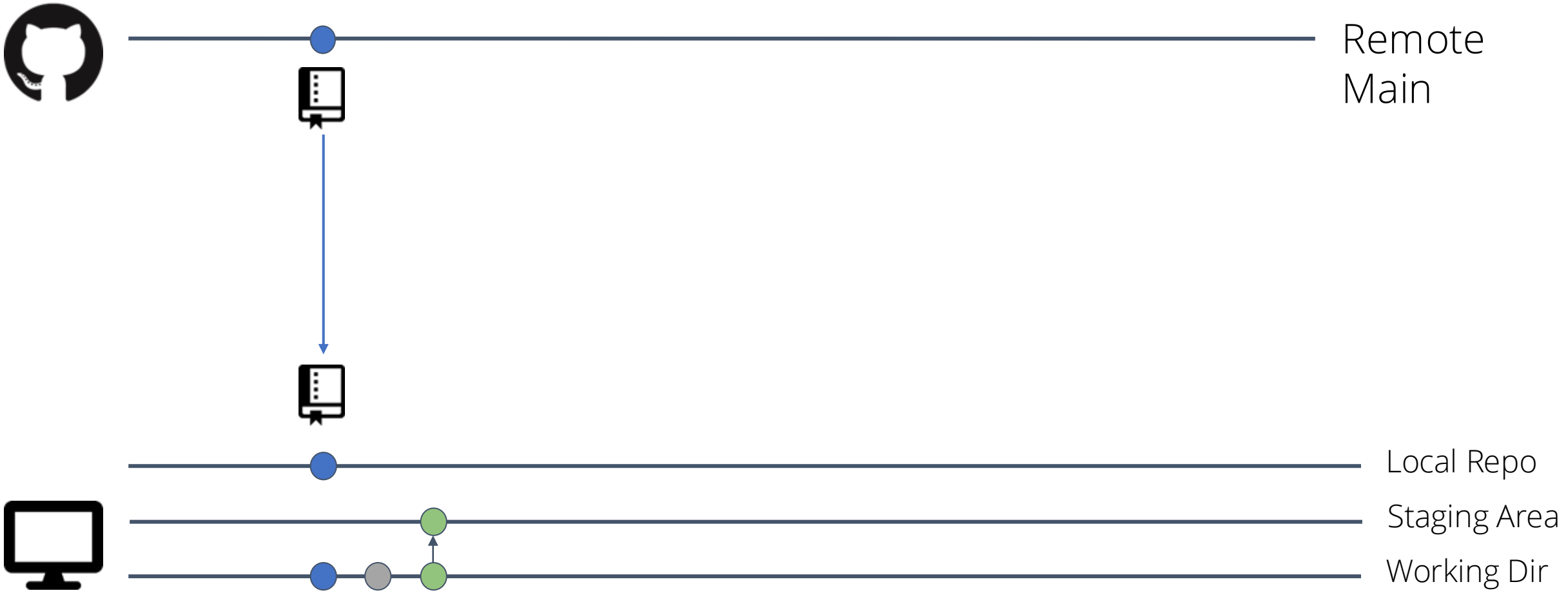
> git clone REPO



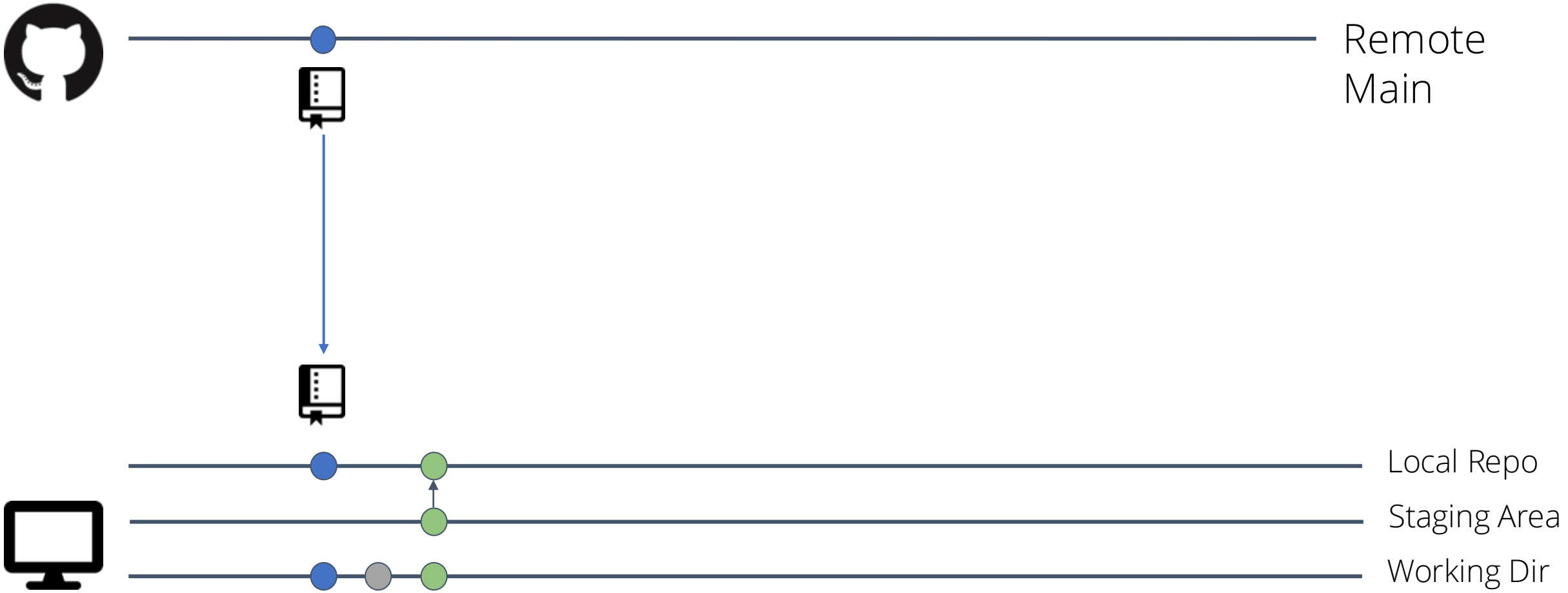
Edit file in working directory



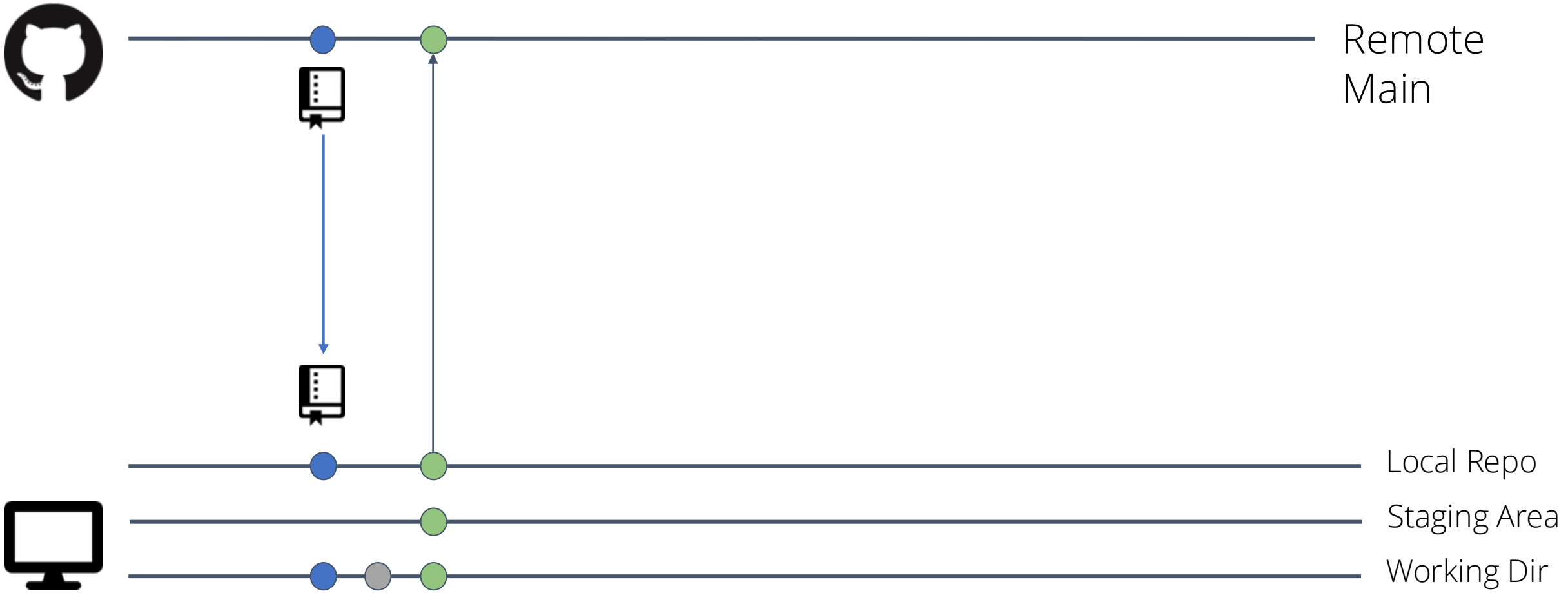
> git add File



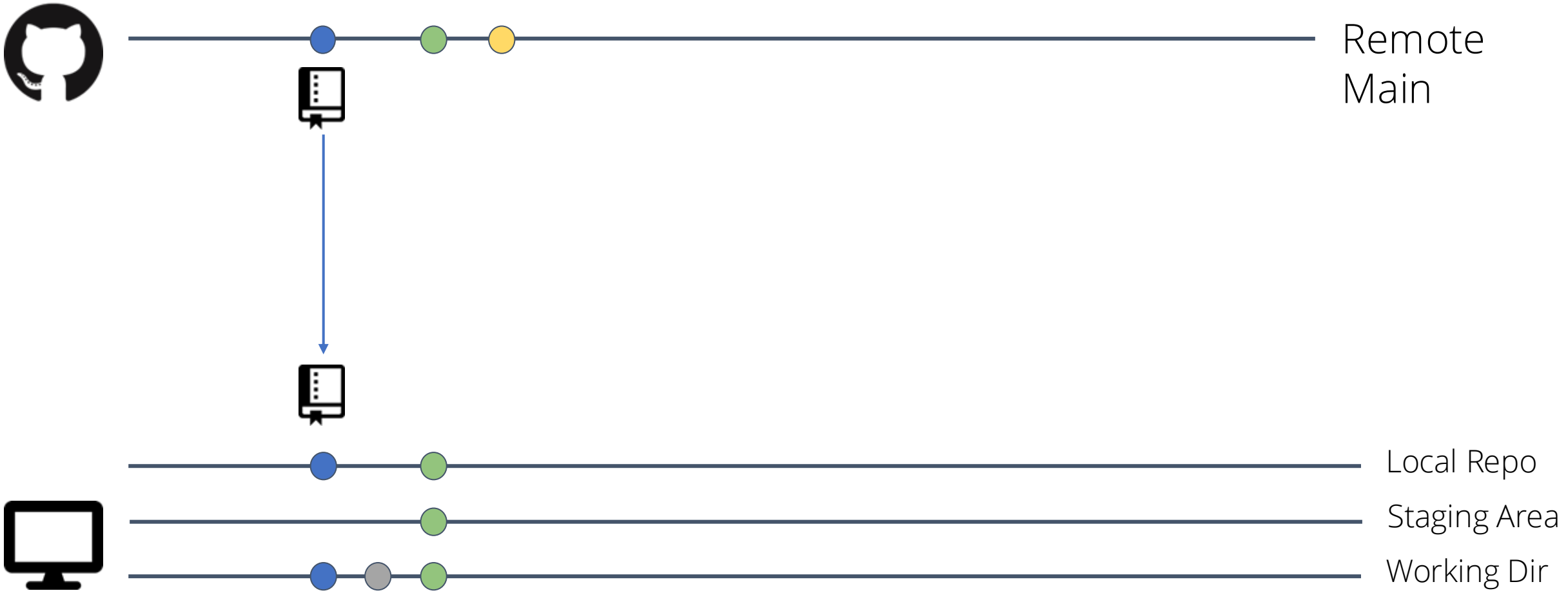
> git commit -m “added green edit”



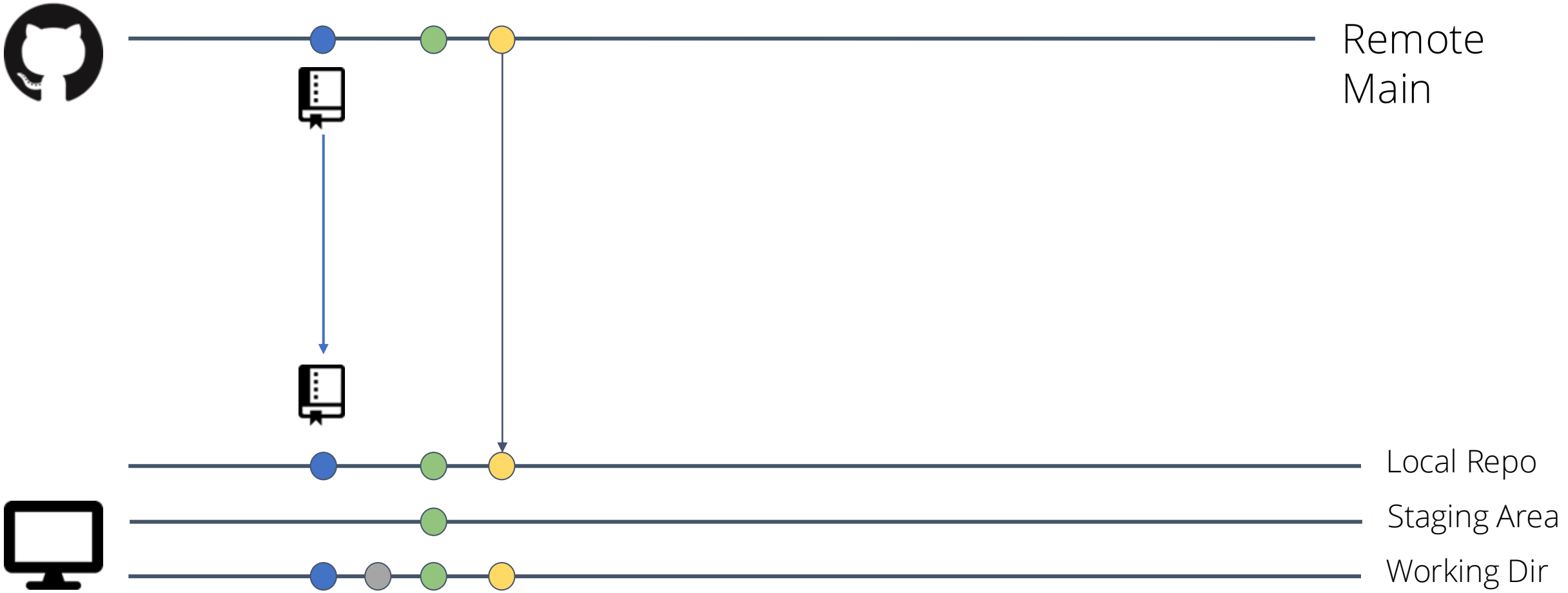
> git push origin main



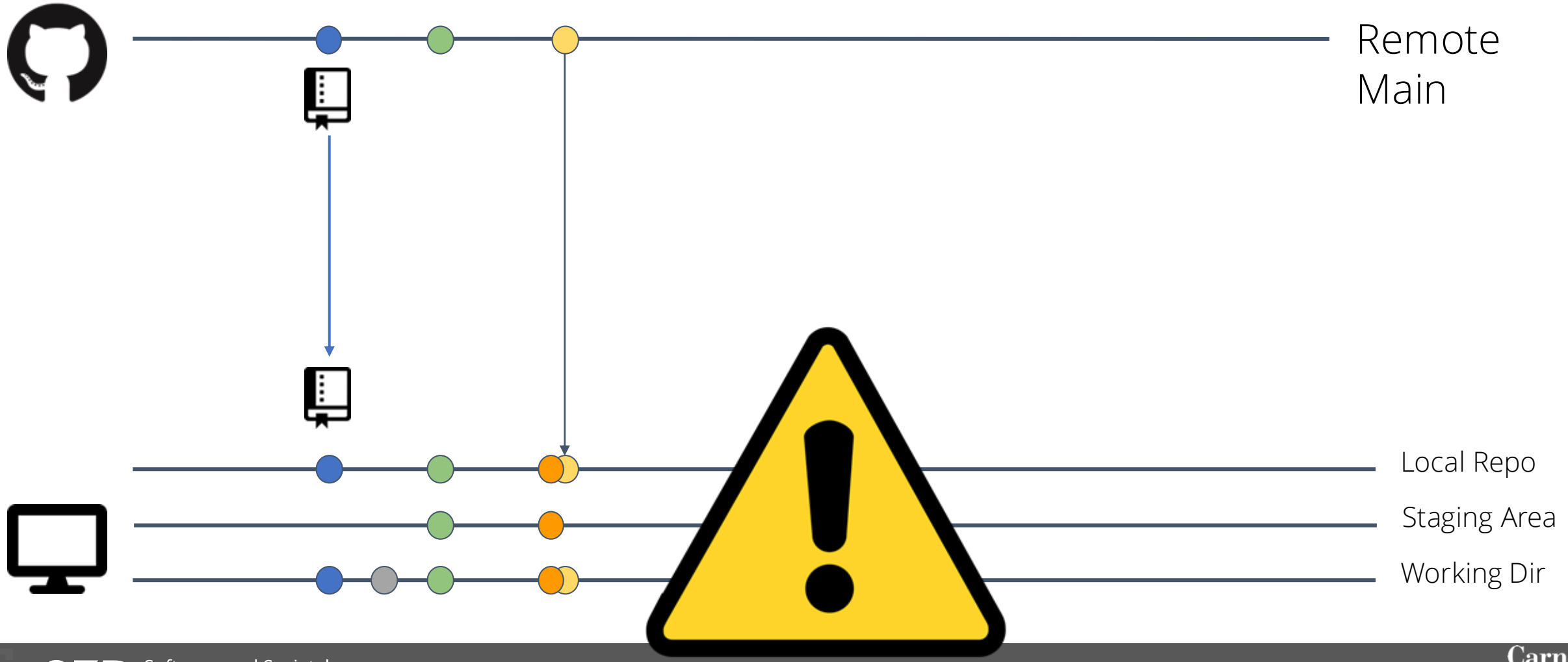
> git fetch (check for changes on remote)



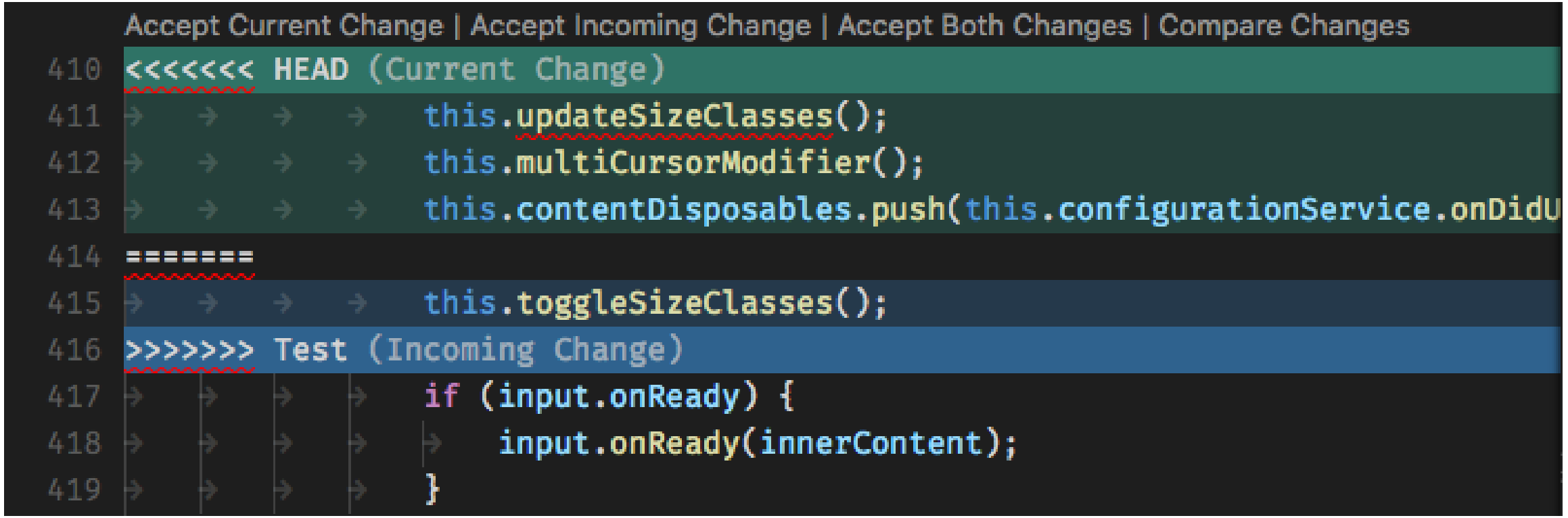
> git pull



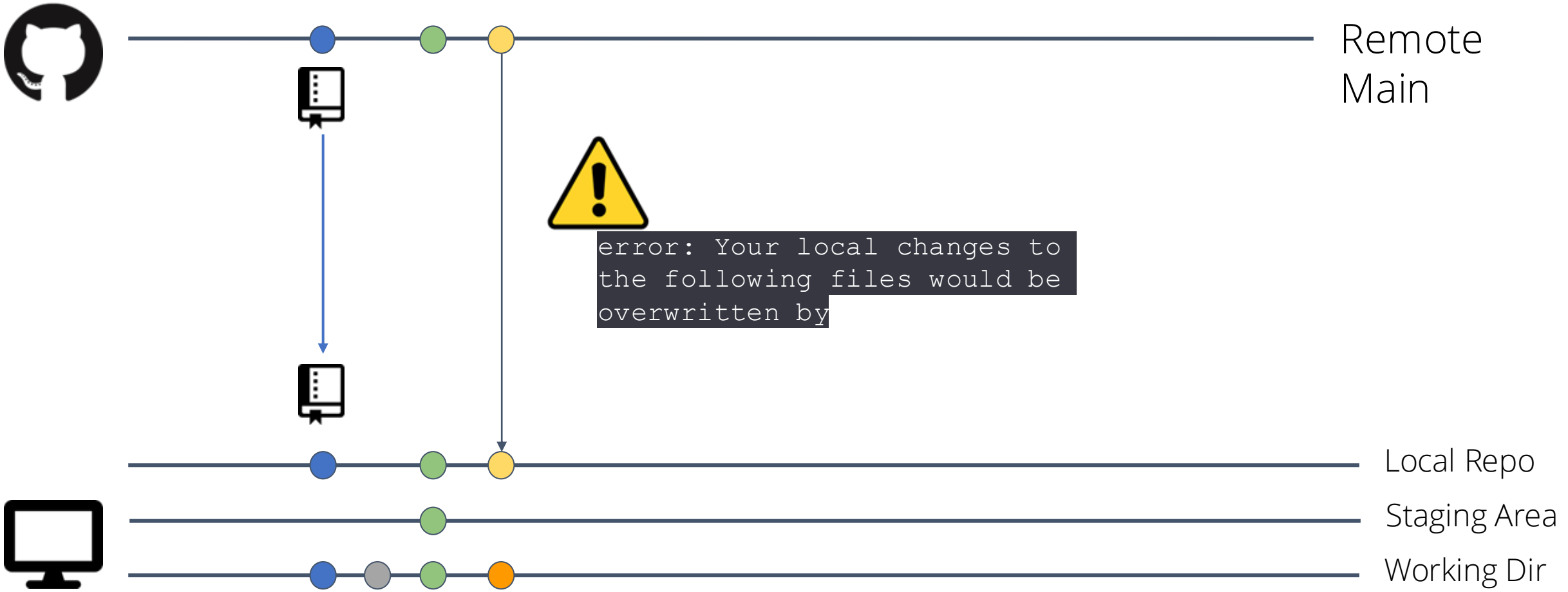
> git pull can lead to Merge Conflicts



Merge Conflict (in VS Code)

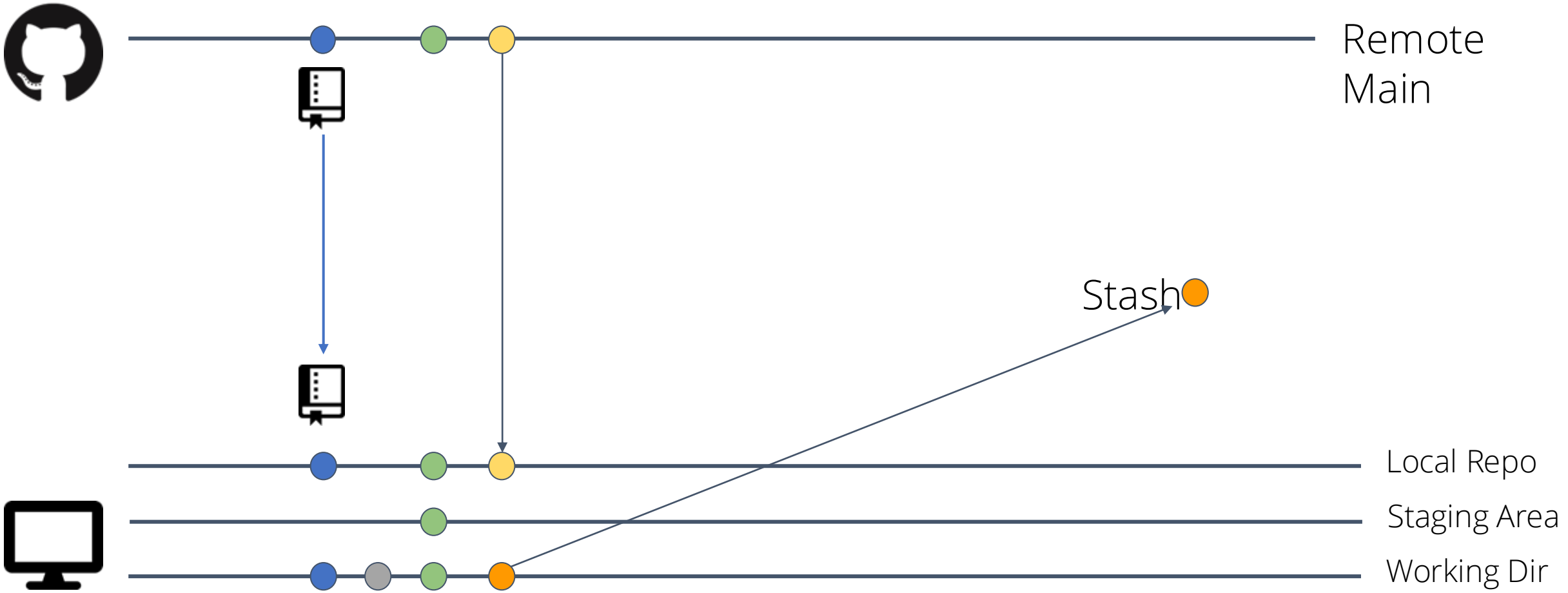


> git pull

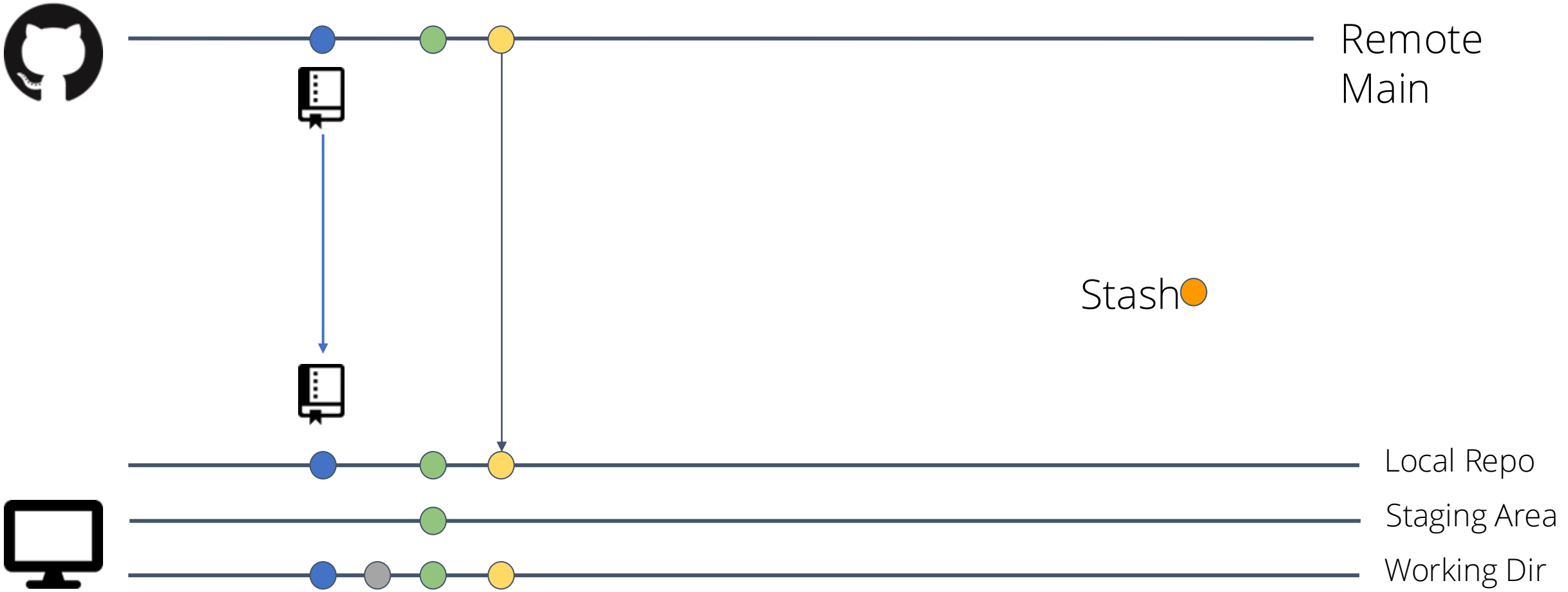

Remote
Main
error: Your local changes to the following files would be overwritten by
Local Repo
Staging Area
Working Dir

Keep Changes

> git stash

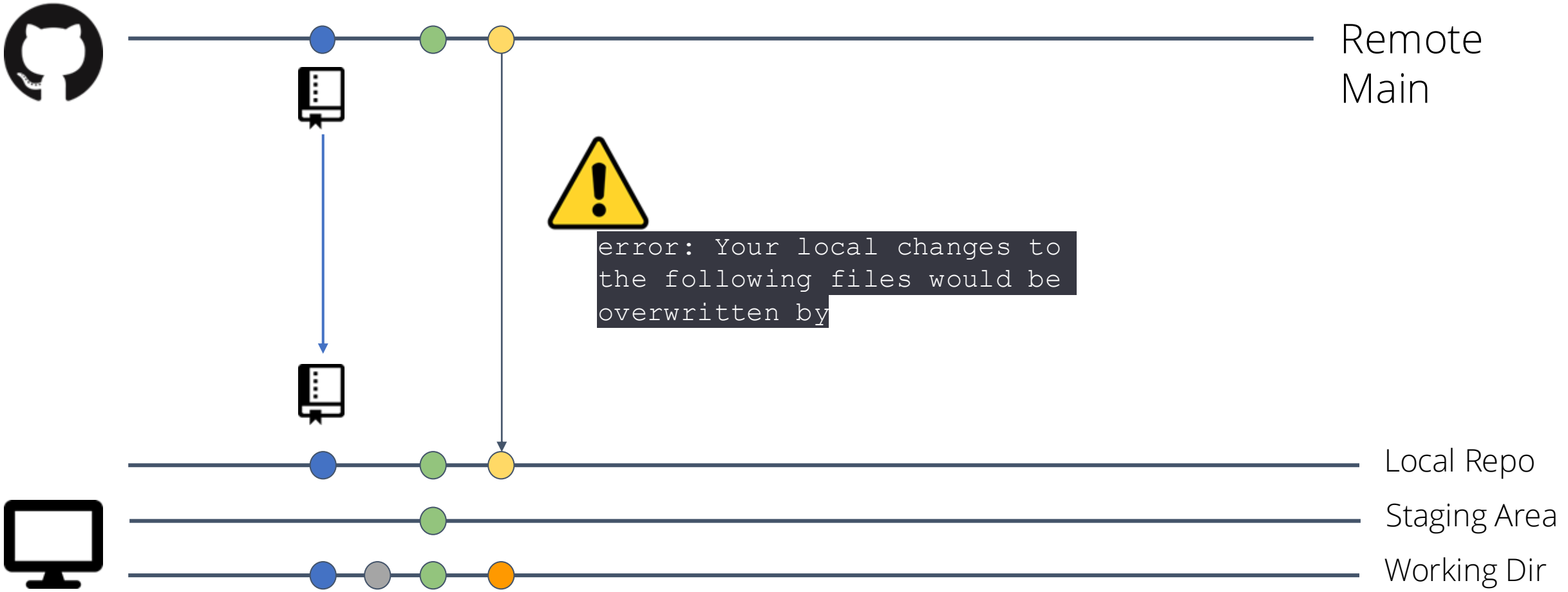


> git pull

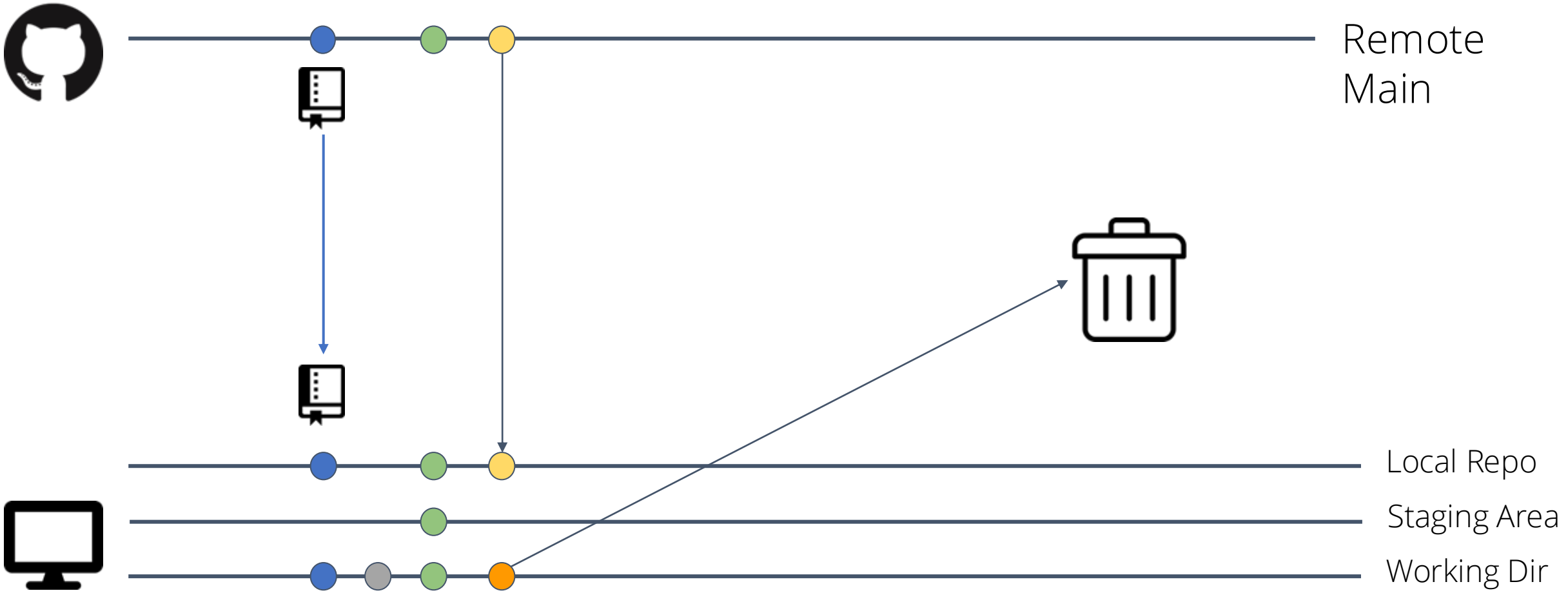


Discard Changes

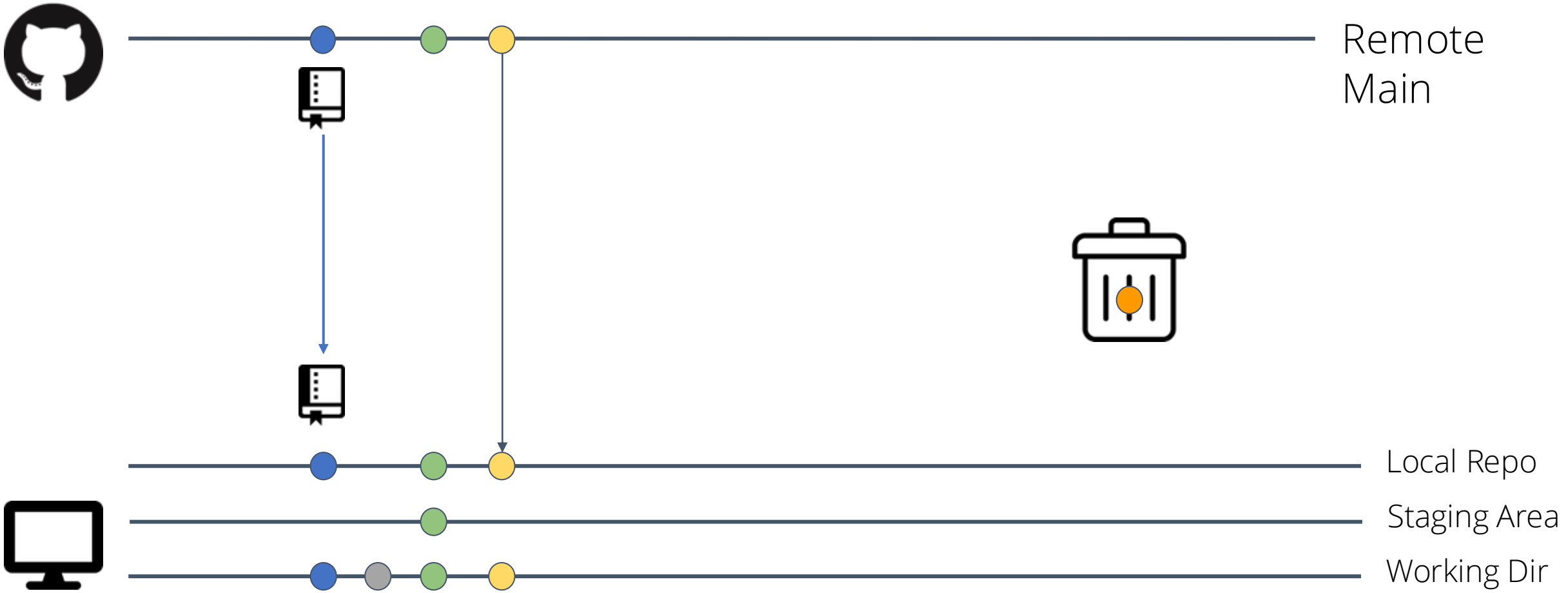
> git pull


Remote
Main
error: Your local changes to the following files would be overwritten by
Local Repo
Staging Area
Working Dir

> git reset –hard



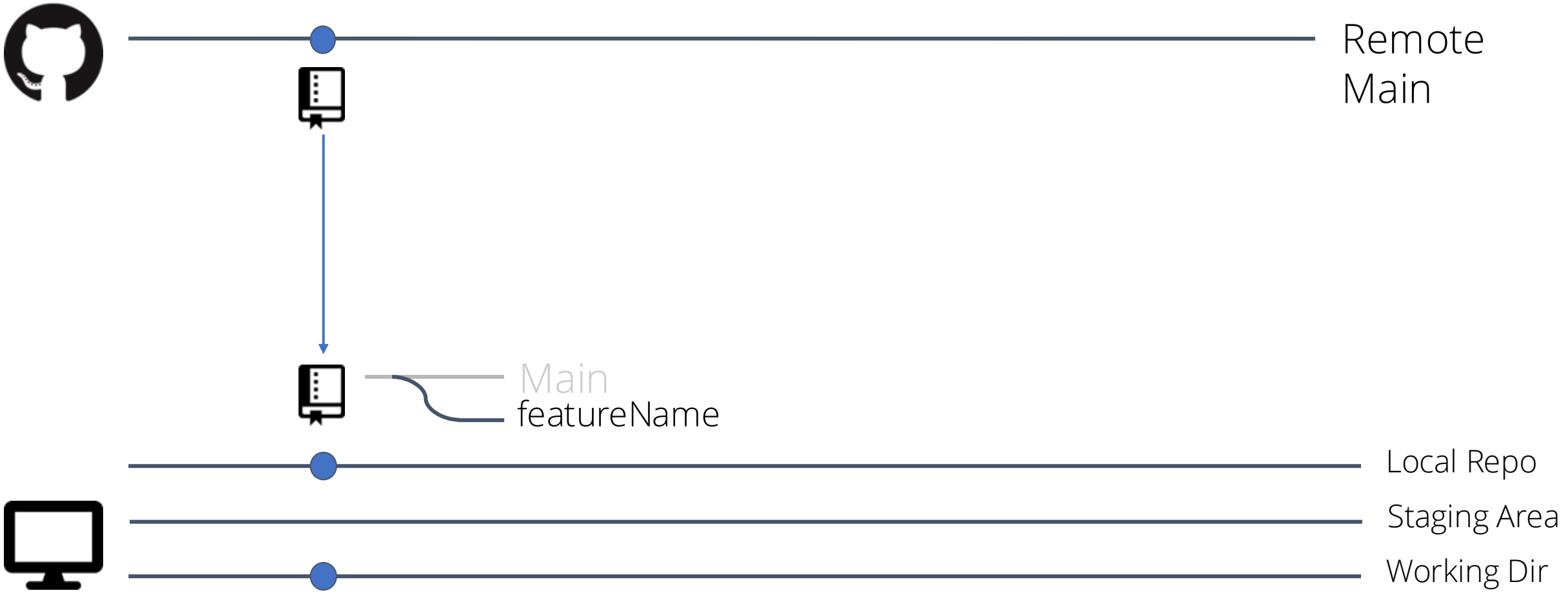
> git pull



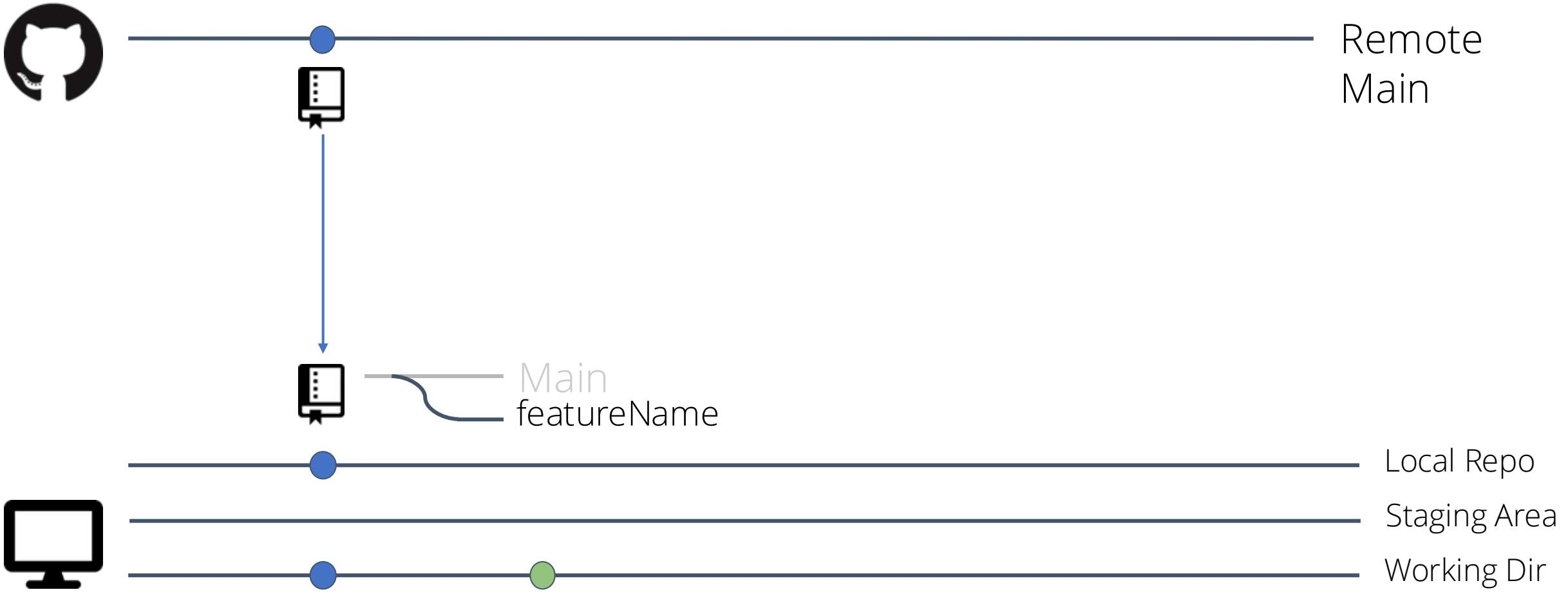
What if you want to work on multiple things at once?

Branches

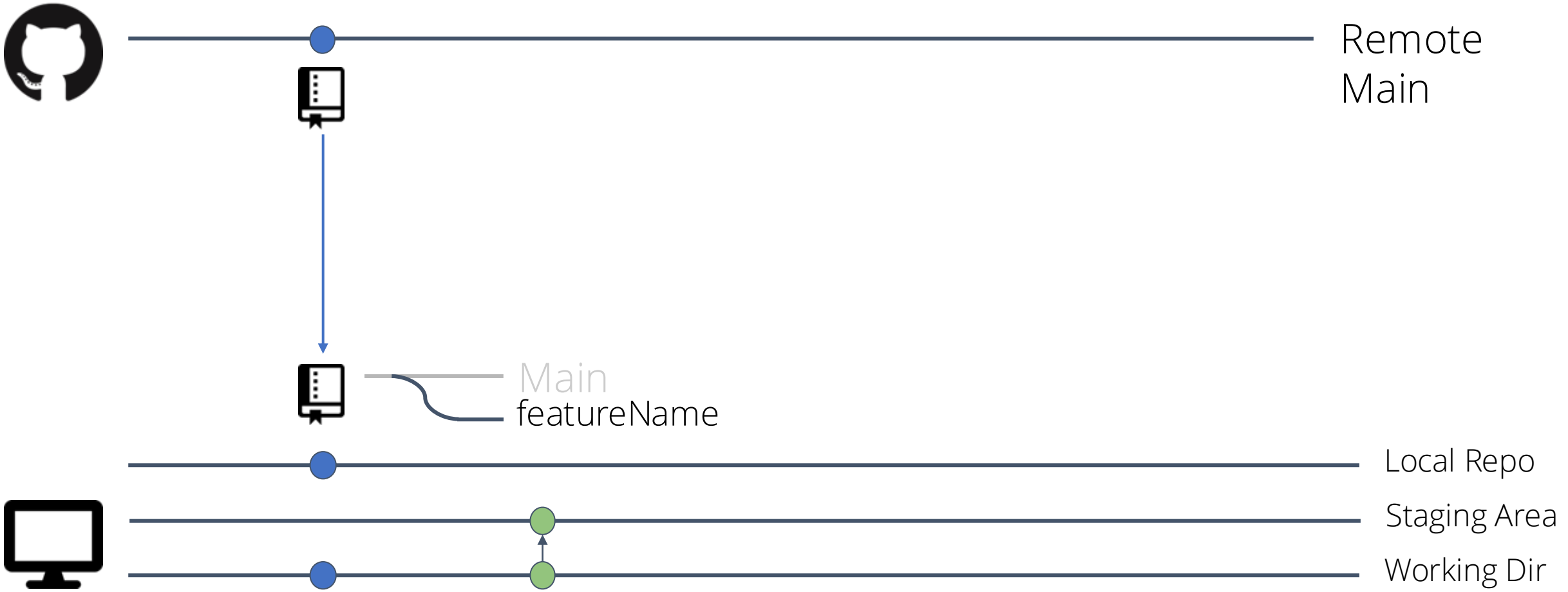
> git checkout -b featureName



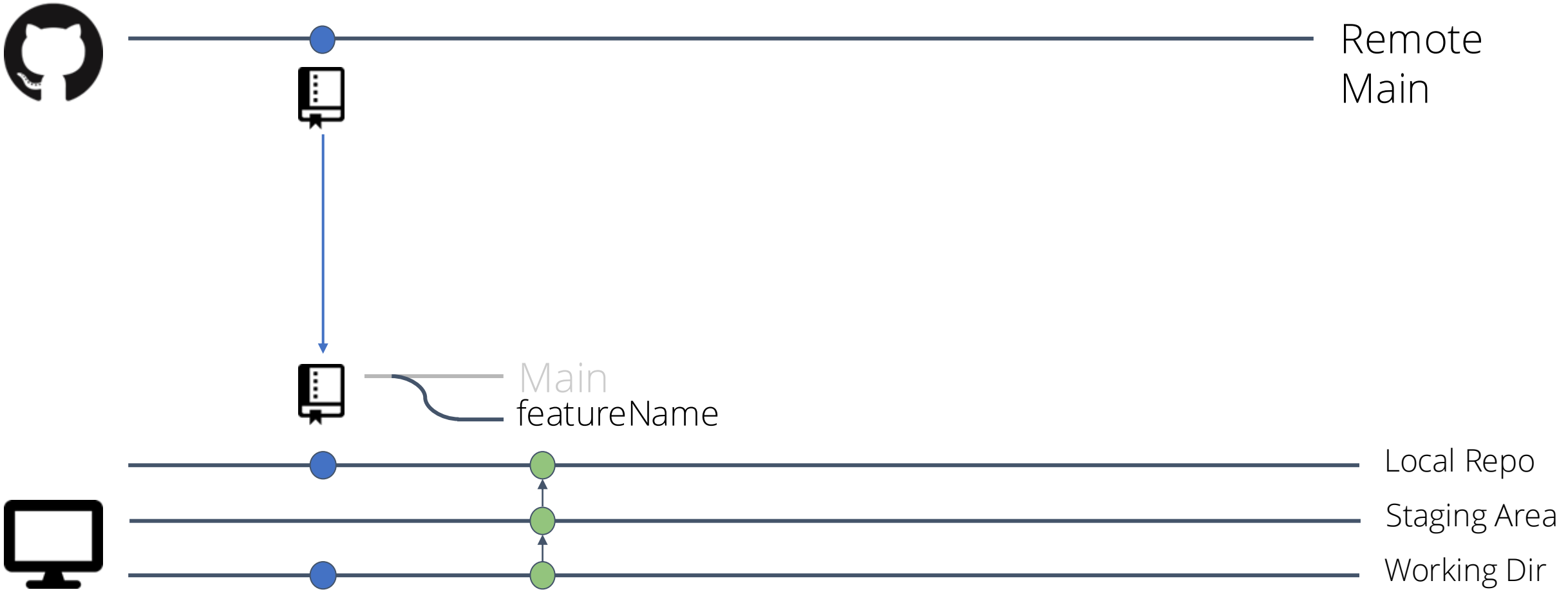
edit file(s) in working directory



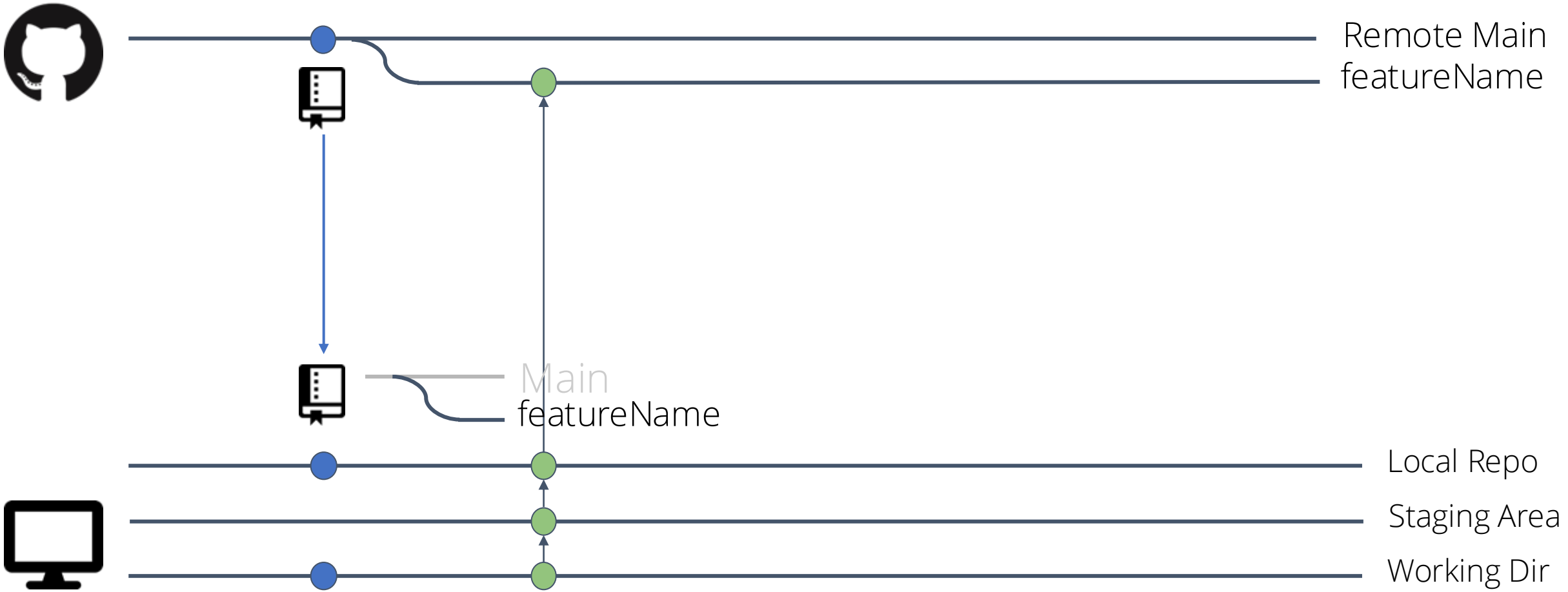
> git add File


Remote
Main
Main
featureName
Local Repo
Staging Area
Working Dir

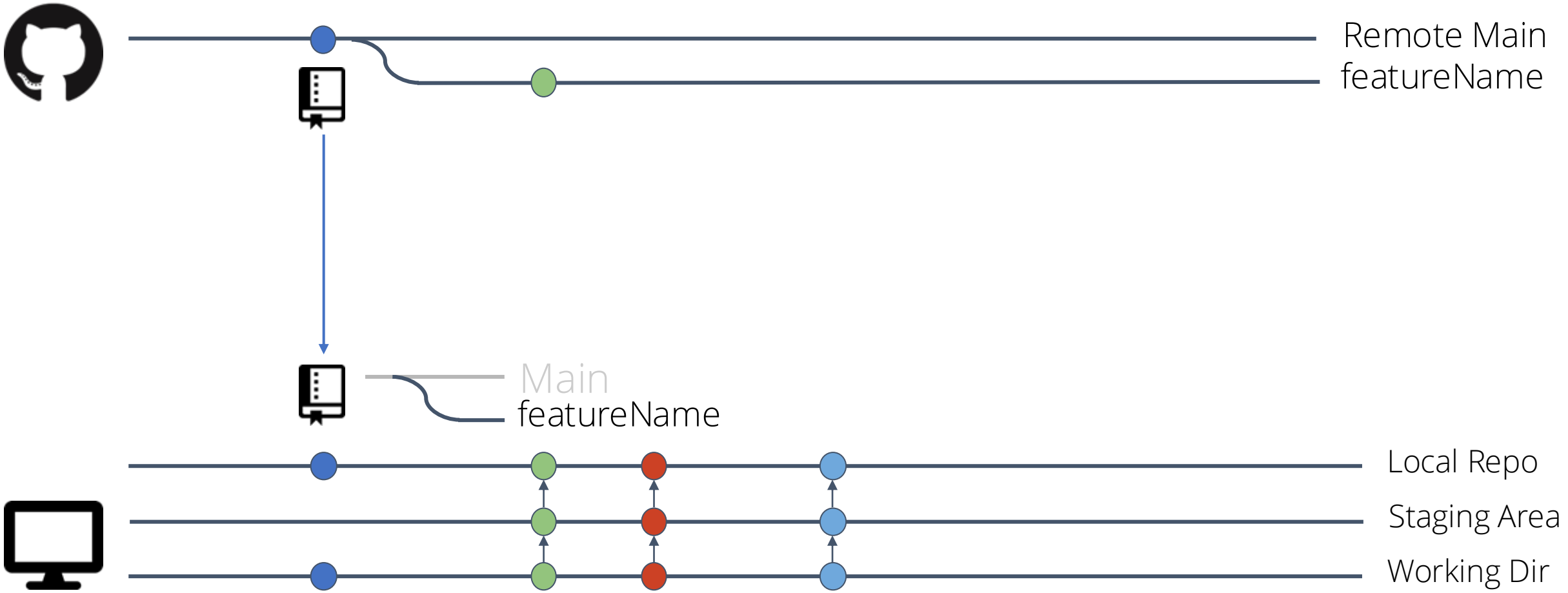
> git commit -m “added green edit”



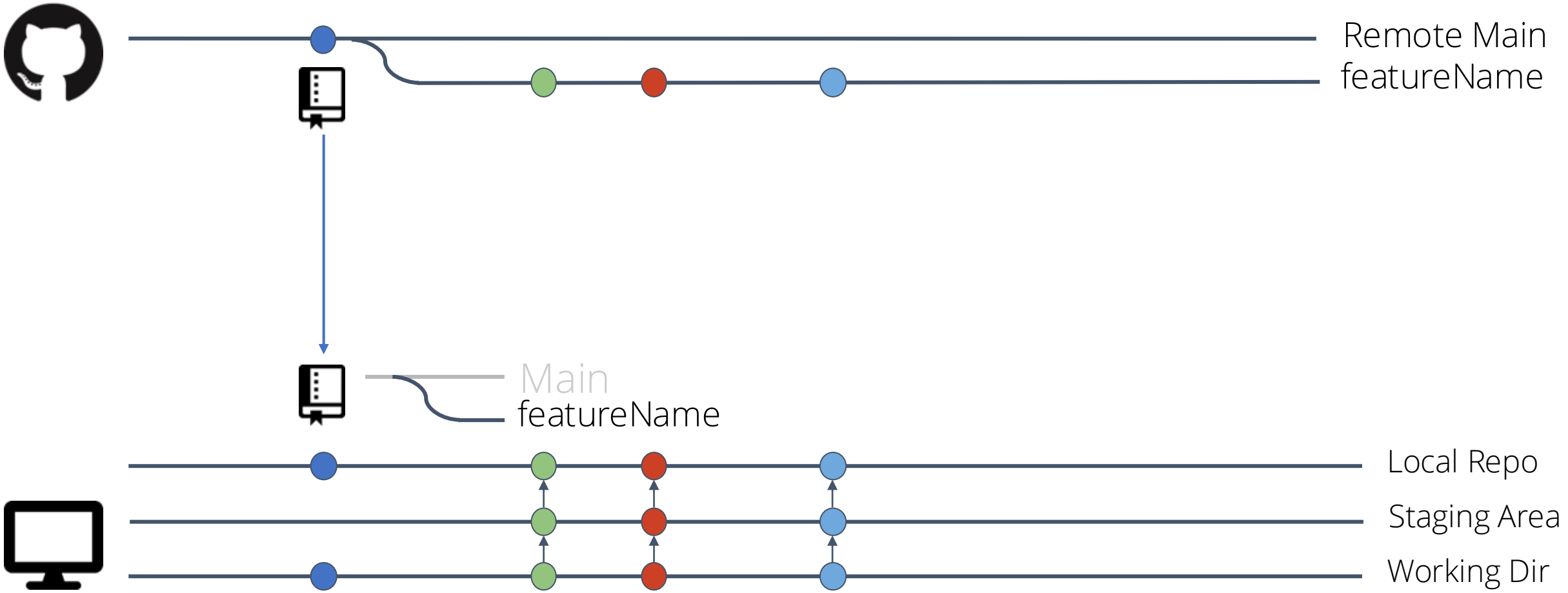
> git push origin featureName



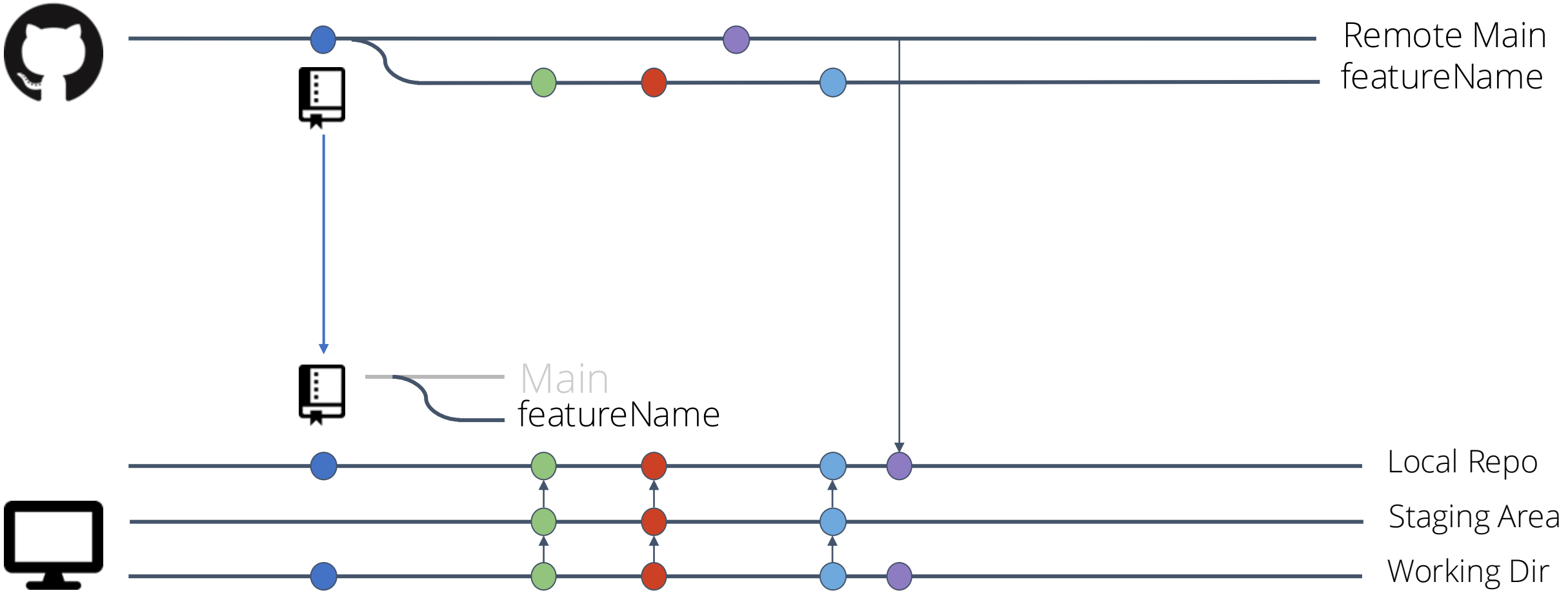
Continue work



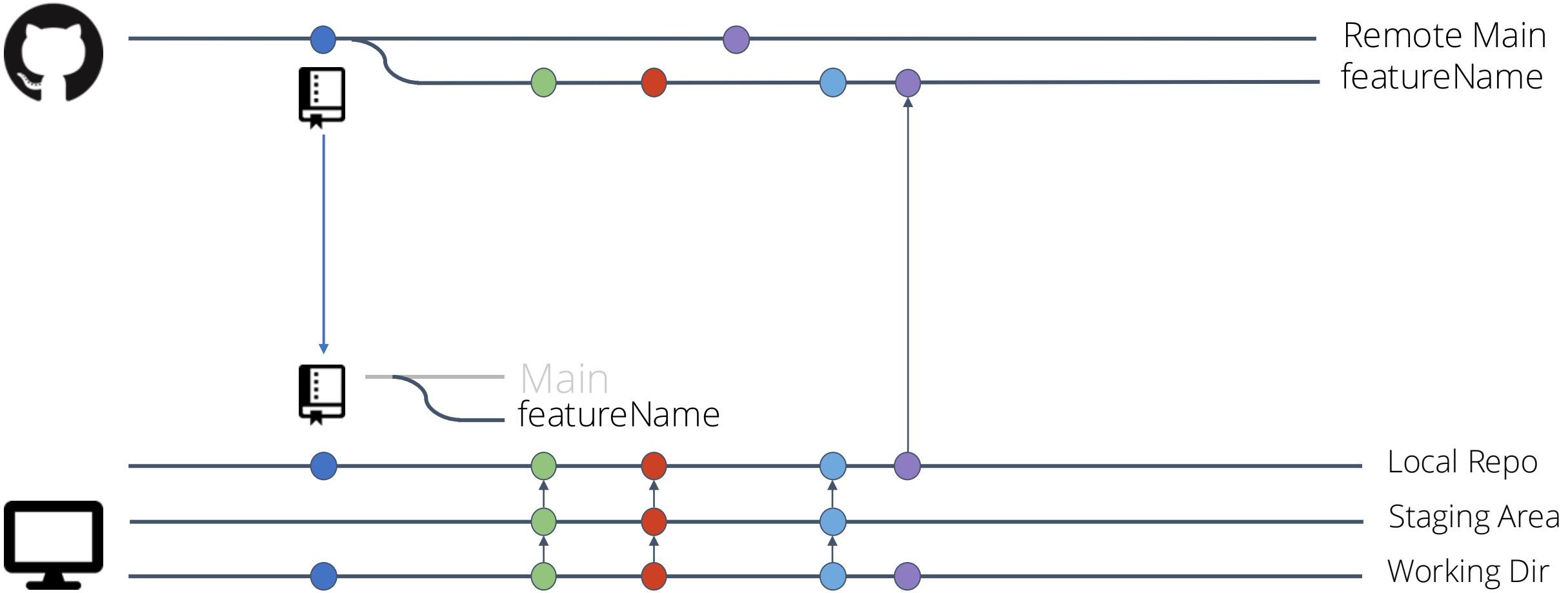
> git push remote featureName



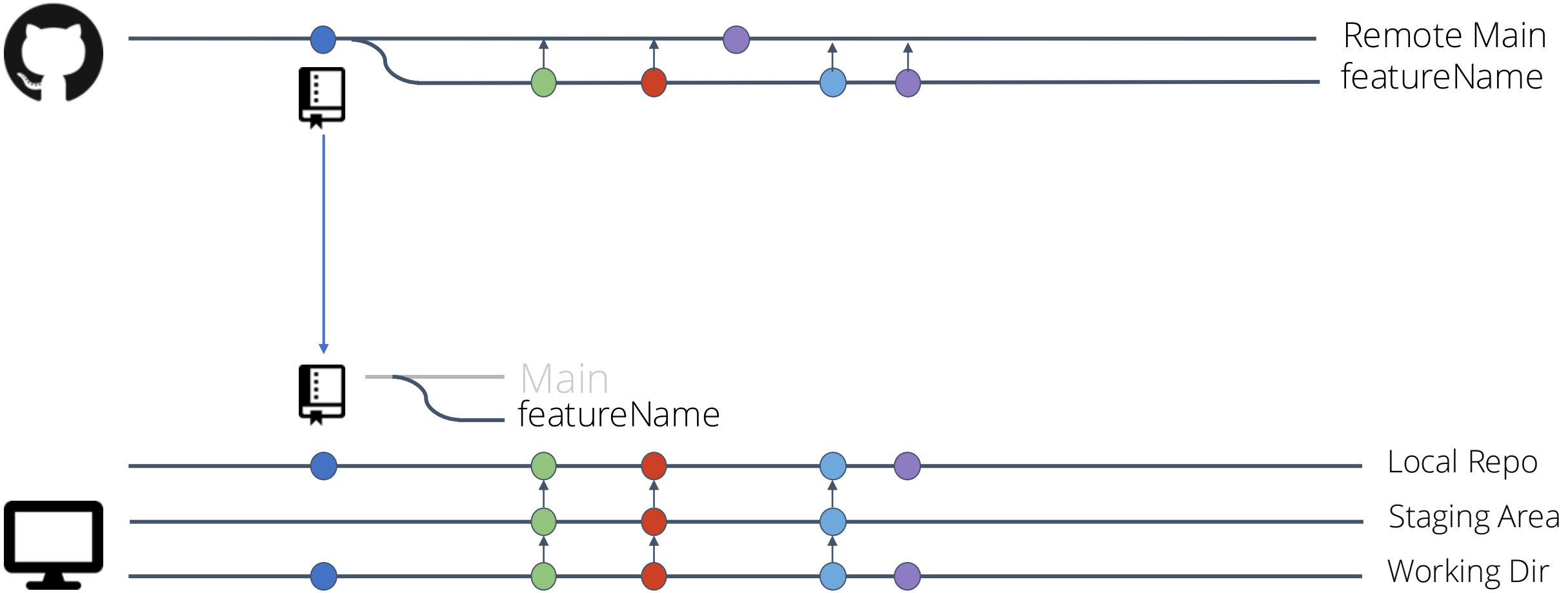
> git pull origin main //Resolve any conflicts



> git push featureName



Open Pull Request on Github



Setup the environment

Problems in running code?

- What version of python are you running? Node? npm?
- What if you have different software that relies on different versions?
- How can you run same code on different platforms?
- How can you automate setup?

Docker

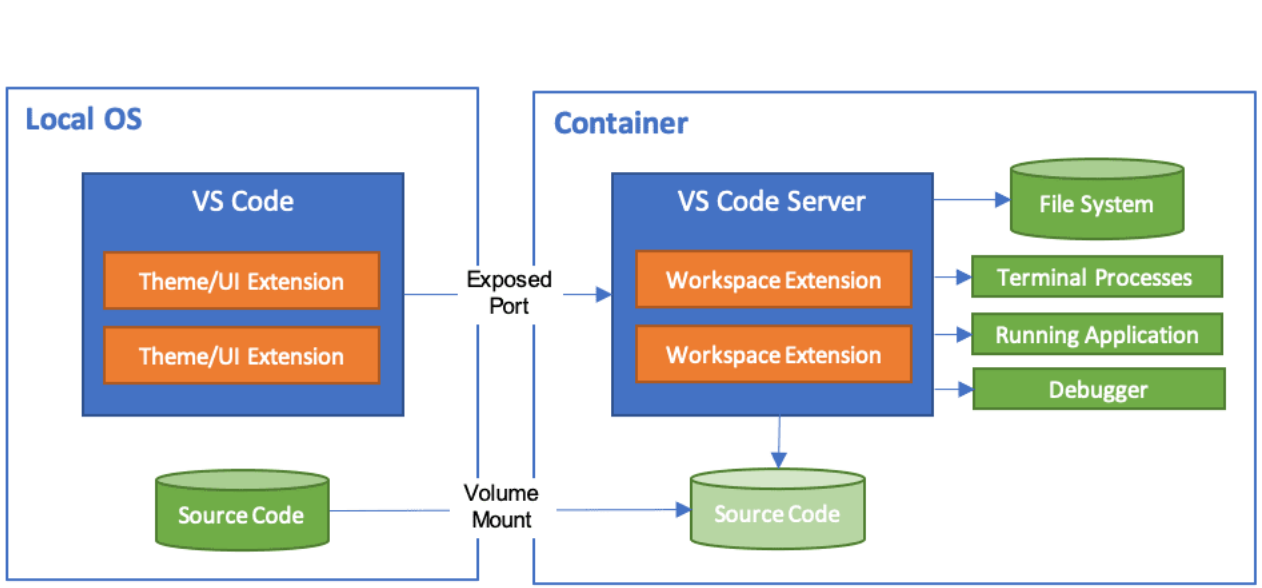
- Docker allows packaging applications and their dependencies into lightweight, portable containers.
- Each container includes everything needed to run the application (code, runtime, libraries, and system tools) in a self-contained environment.
- Containers share the host operating system's kernel but run in isolation from each other, making them much lighter than traditional virtual machines.
- Docker uses images as blueprints for containers, which can be version-controlled and shared through registries like Docker Hub.

Why we use Docker

- Docker solves the "it works on my machine" problem by ensuring applications run consistently across development, testing, and production environments.
- Docker simplifies dependency management by bundling everything an application needs, avoiding conflicts between different projects or system configurations.
- Containers start in seconds and use minimal resources compared to VMs, making development faster and enabling efficient scaling in production.
- HOWEVER: They are complicated to create and manage, and add complexity to running the application.

VSCode Development Containers

- Allow VS Code to handle the development container for us.
- Hides the complexity of containers, while providing all the benefits.
- What we use in this course.



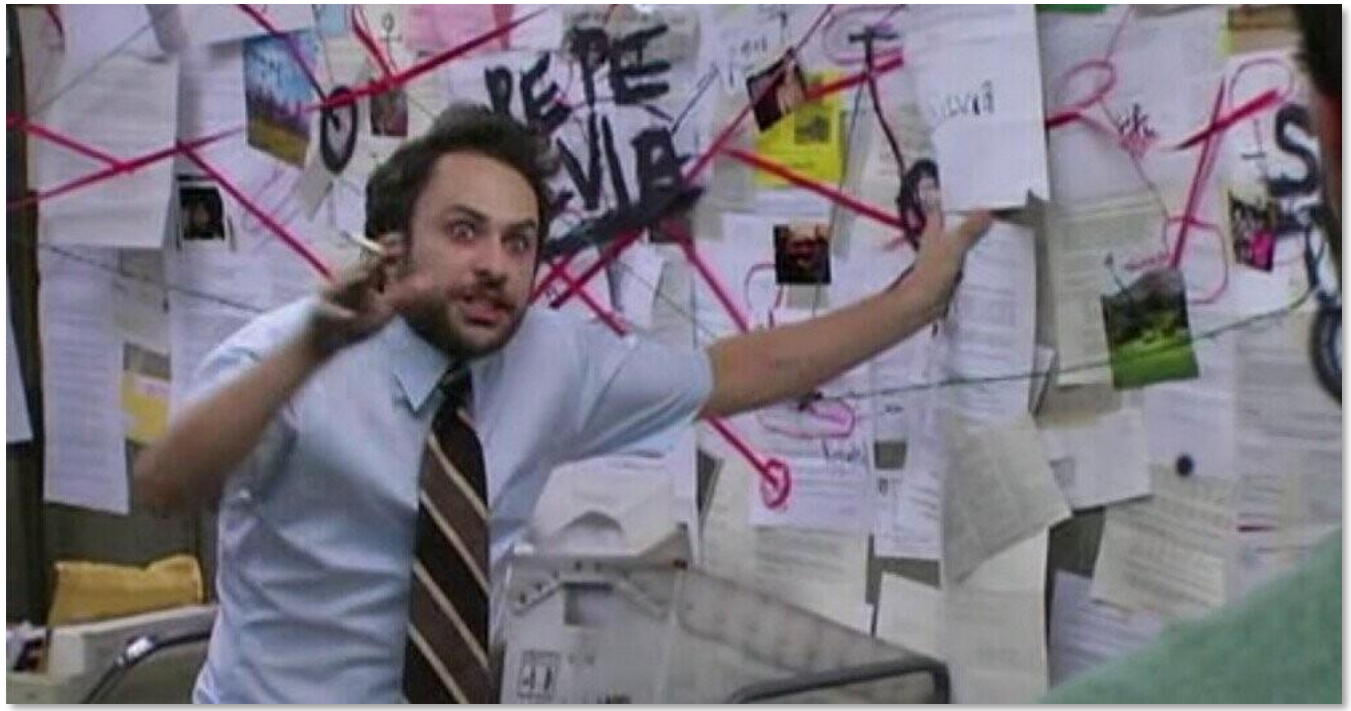
Dive into the code

Participation Activity: Part 1

- Take out a piece of paper (or ask for one)
- Write down the **challenges you've faced trying to understand someone else's code**
- Pair with your neighbor and discuss your answers. Do you agree?
- Share with the class!
- Write your own andrewID on the paper; leave it at the end of class.

You will never understand the entire system!

Challenge: How do I tackle this codebase?



Participation Activity: Part 2

- Write down **strategies to understand a large codebase that is unfamiliar to you**

Challenge: How do I tackle this codebase?

- Leverage your previous experiences (languages, technologies, patterns)
- Consult documentation, whitepapers
- Talk to experts, code owners
- **Follow best practices to build a working model of the system**

Bad news: There are few helpful resources!

- **Working Effectively with Legacy Code**

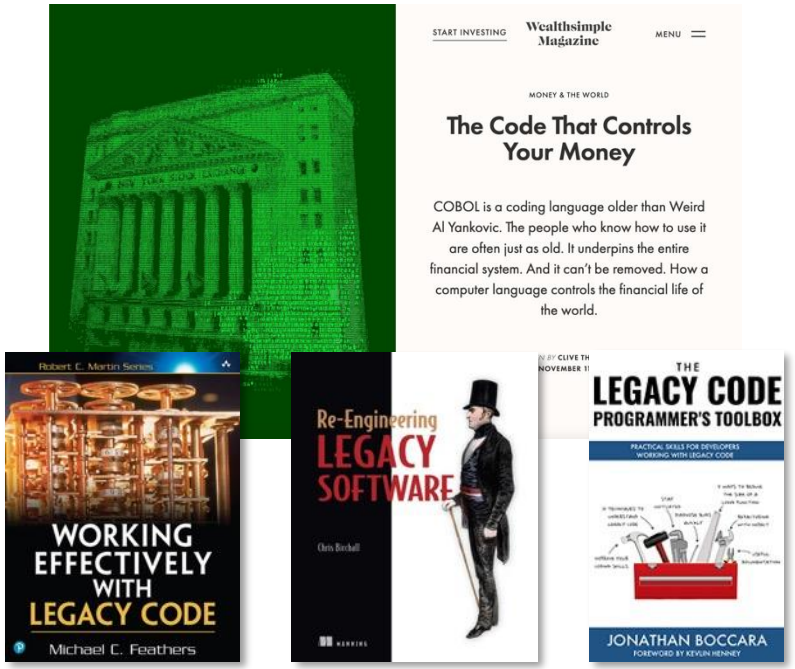
  Michael C. Feathers. 2004

- **Re-Engineering Legacy Software**

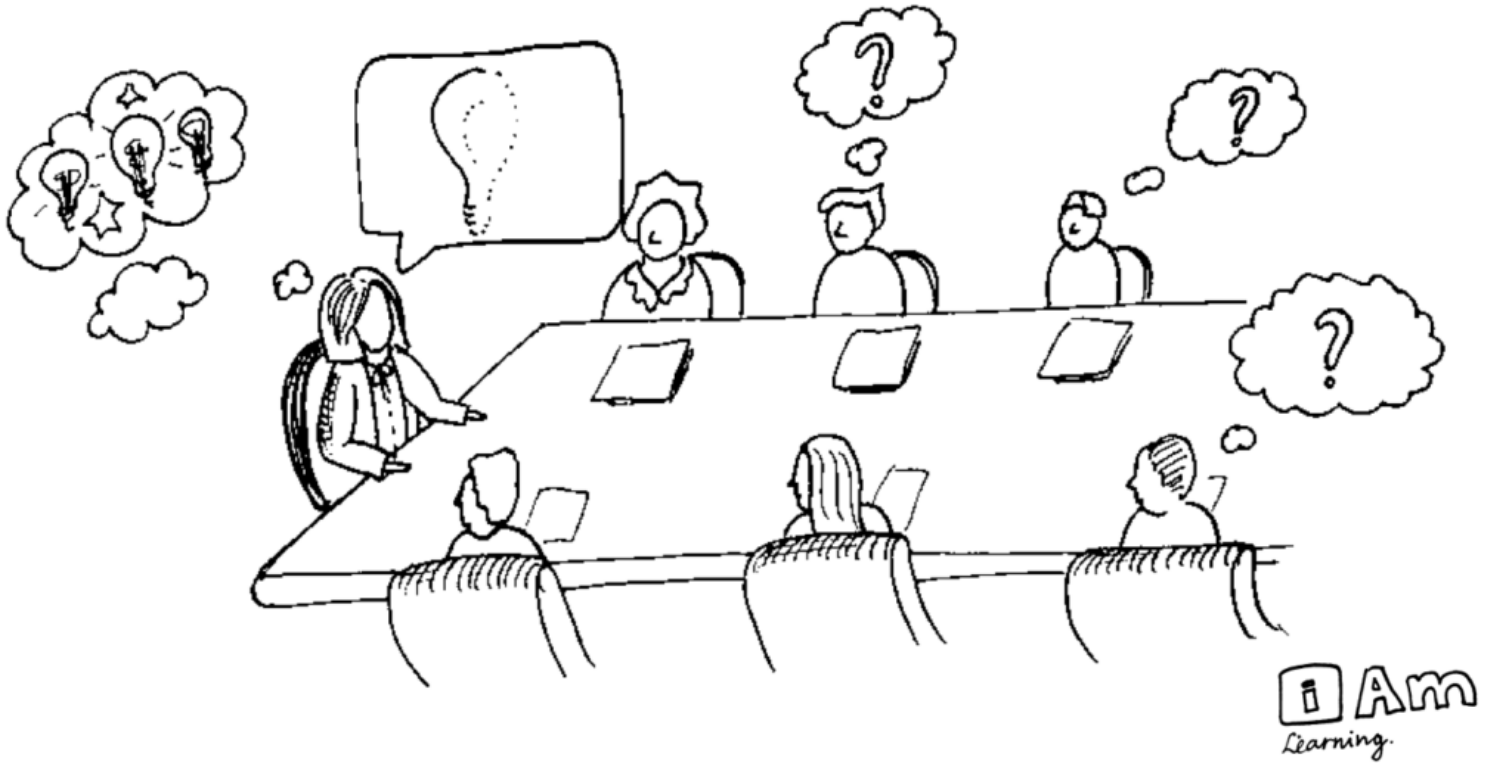
  Chris Birchall. 2016

- **The Legacy Code Programmer's Toolbox**

  Jonathan Boccara. 2019

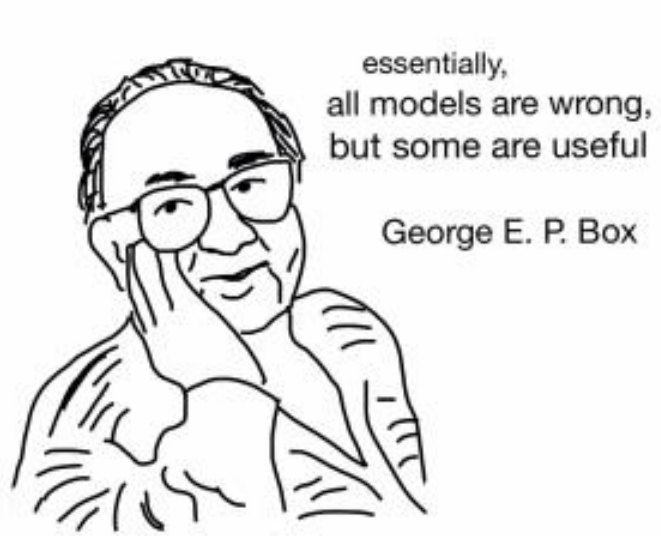


Why? Because of Tacit Knowledge



How to Tackle New Codebases

- **Goal:** develop and test a working model about how (part of) a system works



- **Working Model:** an understanding of the pieces of the system (components), and their interactions (connections)

- How to quickly **build, test and refine** models
  - explore various tools, tips, and techniques

Program comprehension strategies

Novice

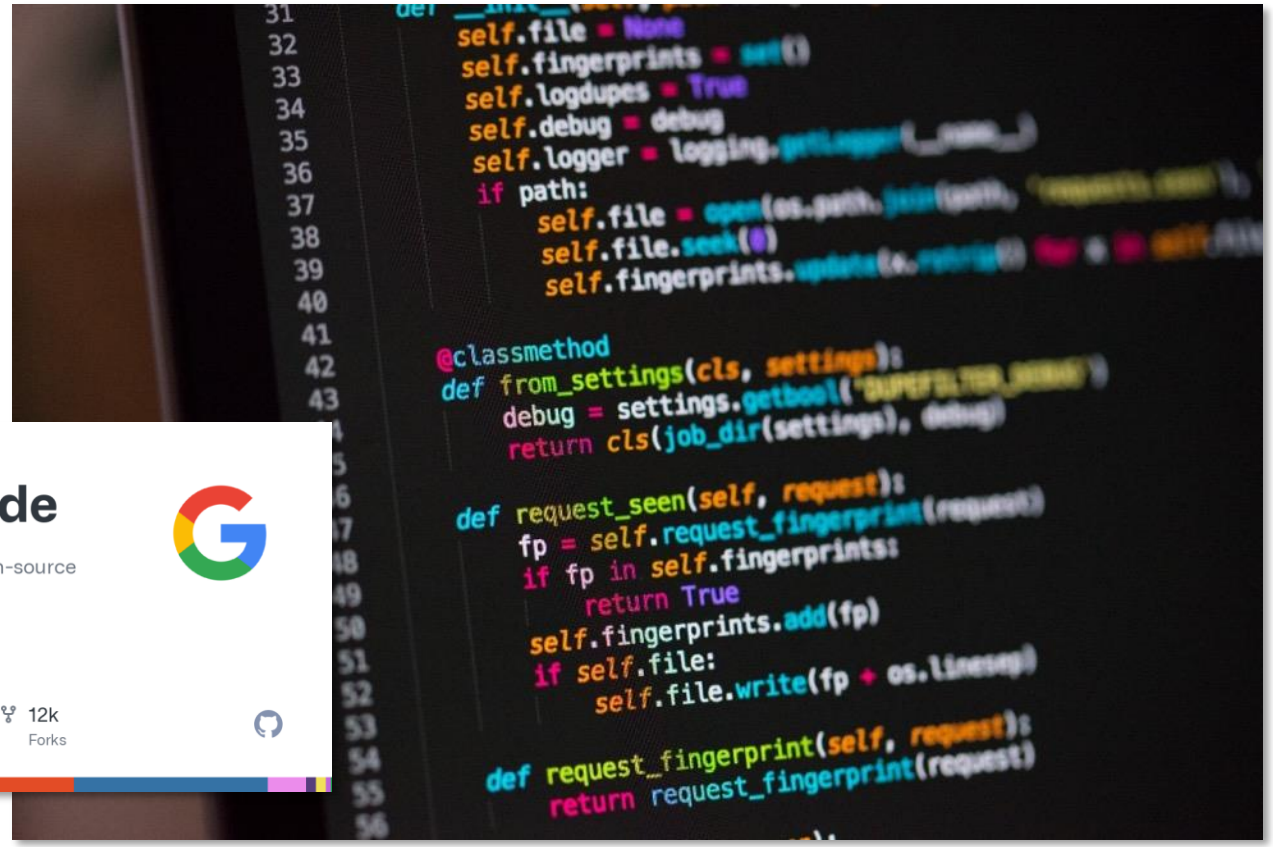
- Reads code line by line
- Revisits same code repeatedly
- Trial and error
- Only tests “happy path”

Expert

- “Top down”
- Recognizes patterns
- Forms hypotheses
- Checks up/downstream consequences

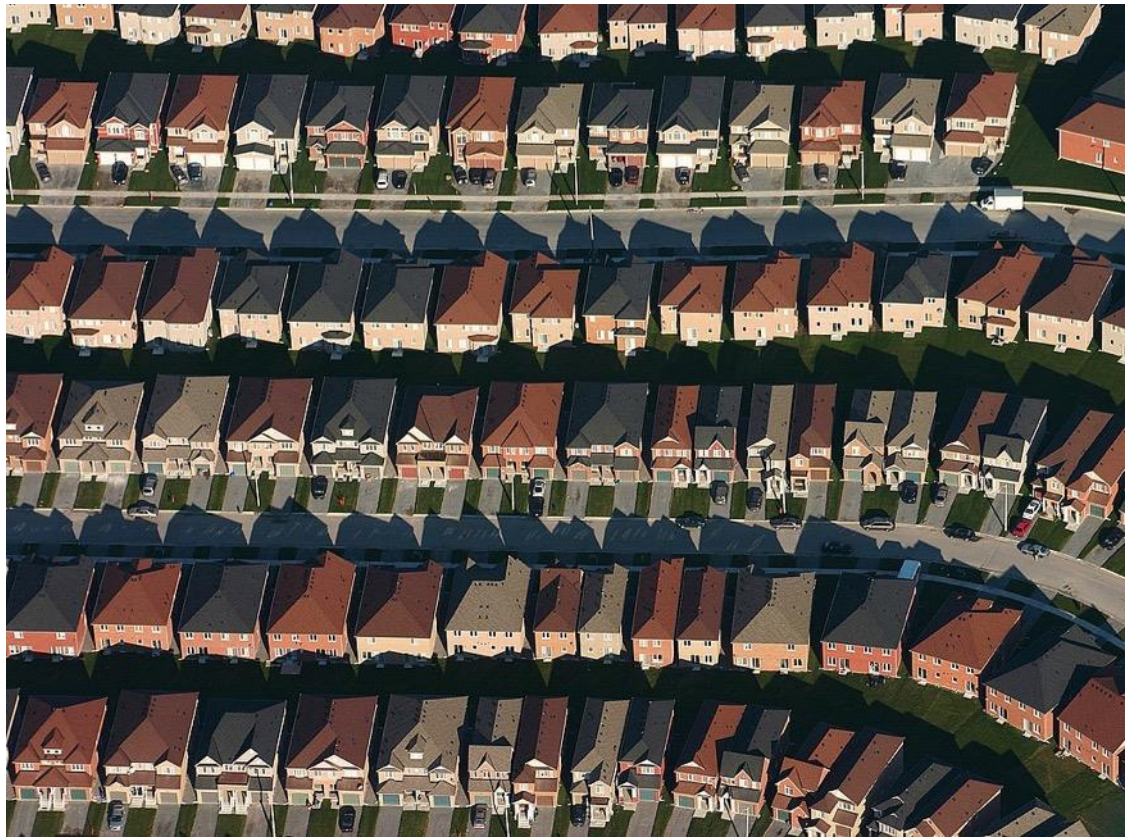
Observation: Software is full of patterns

- File structure
- System architecture
- Code structure
- Names
- …

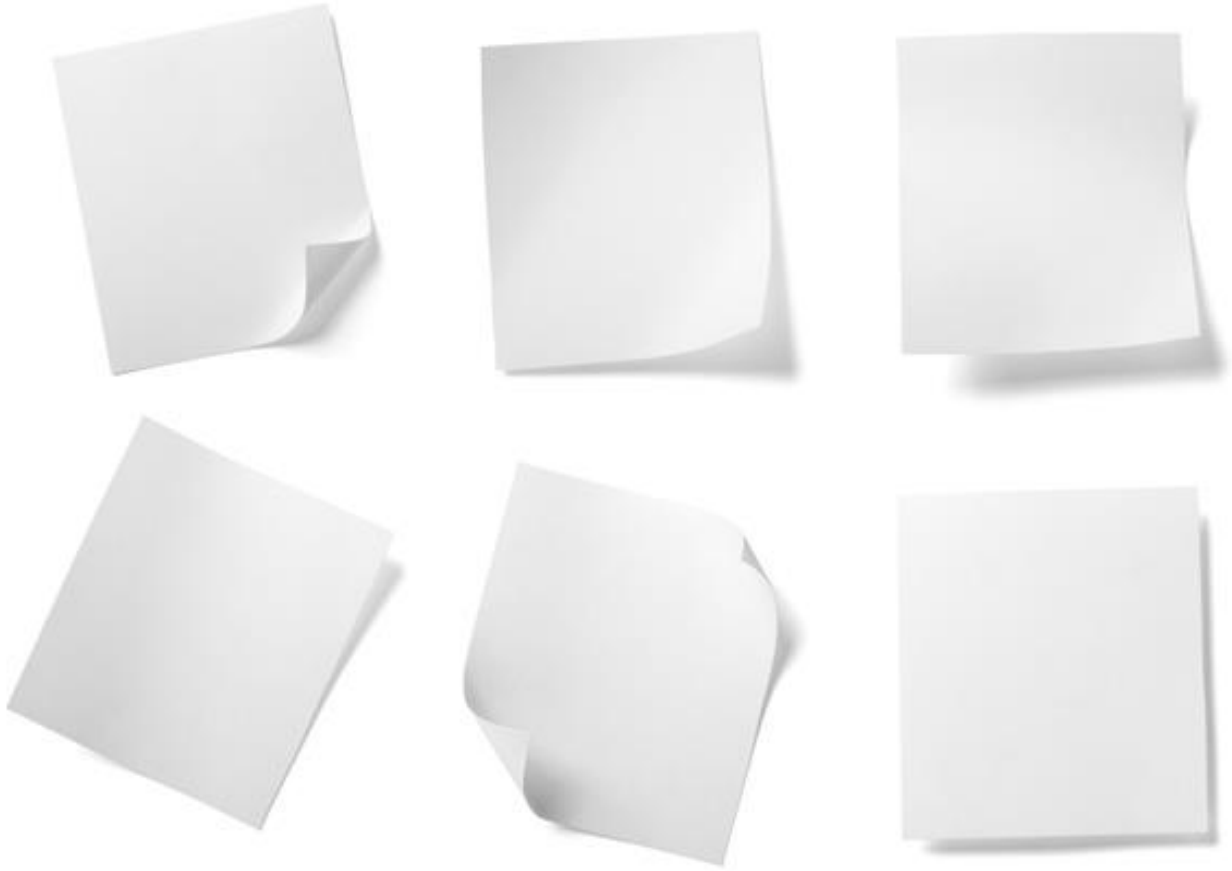


Observation: Software is massively redundant

- There's always something to copy/use as a starting point!

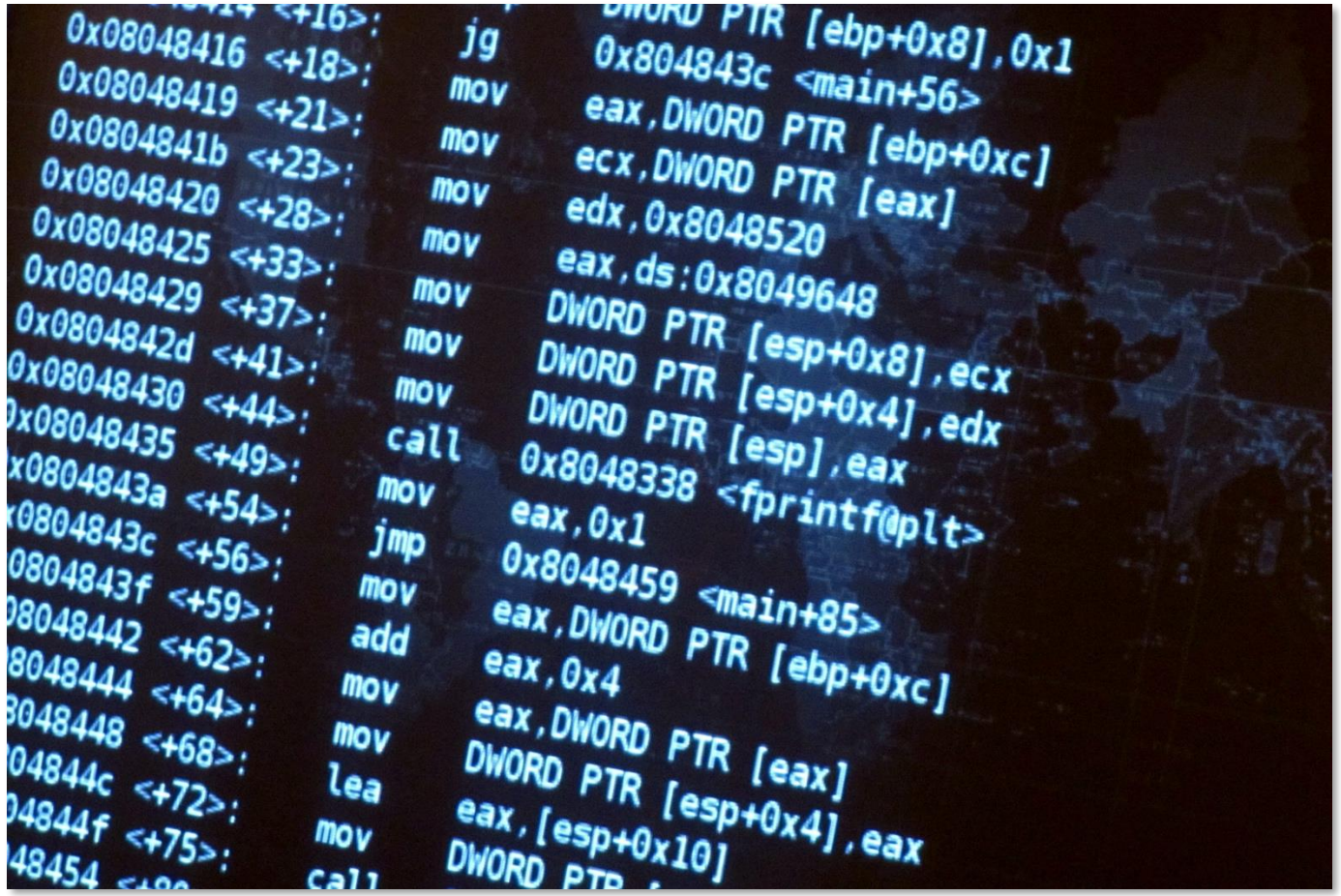


Observation: Code must run to do stuff…



Observation: If code runs, it must have a beginning...

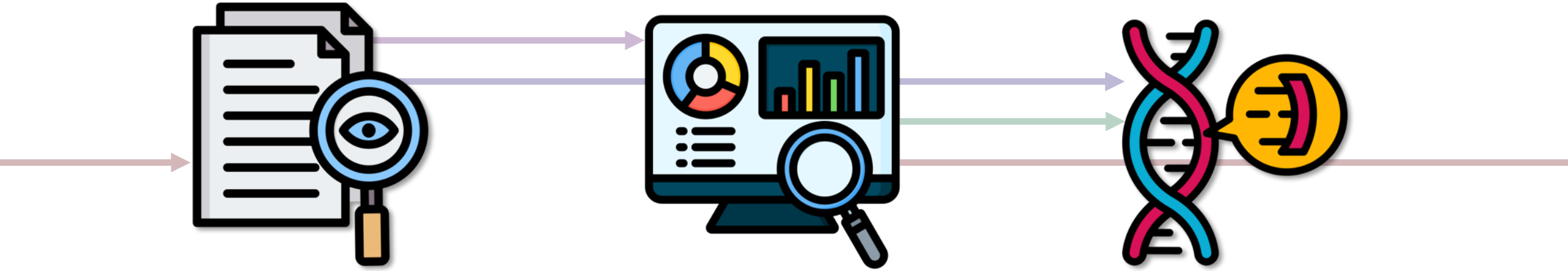
Observation: If code runs, it must exist…



⚠ WARNING ⚠

AVOID SLOT MACHINE PROGRAMMING

How to build, test, and refine mental models

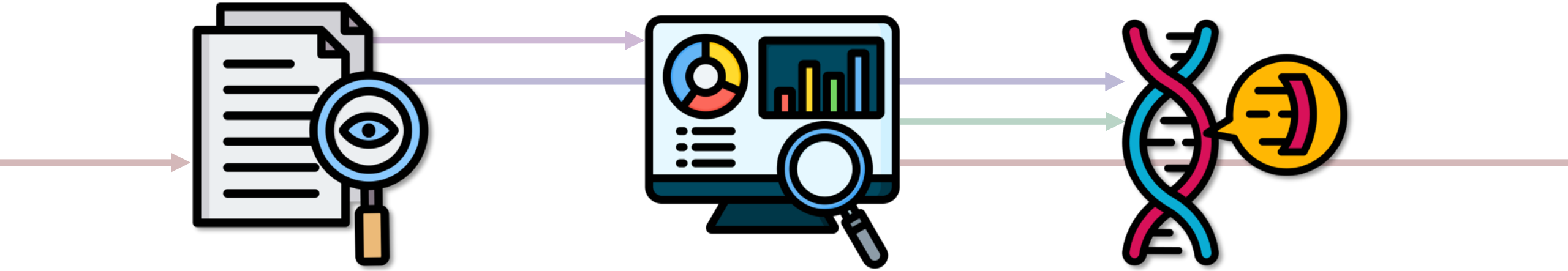


Examine
artifacts without running code

Probe
running system to observe behavior

Modify
code, rebuild, and assess impact

How to build, test, and refine mental models



Examine
artifacts without running code

Probe
running system to observe behavior

Modify
code, rebuild, and assess impact

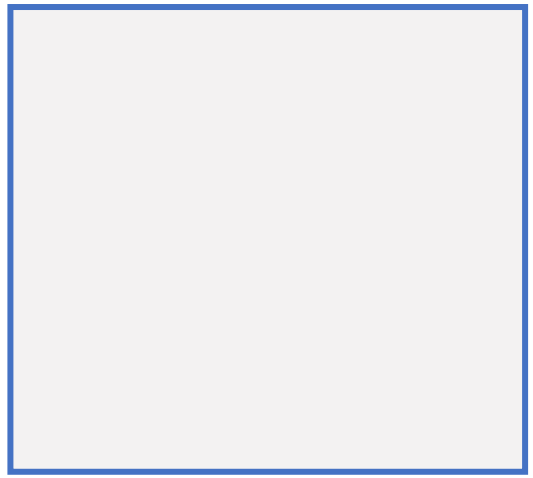
Can code be examined, probed, and modified?

White-box

Source code built locally

- examine
- probe
- modify

Grey-box



Binaries running locally

Open Source	Closed Source
examine	examine
probe	probe
modify *	modify *

Black-box

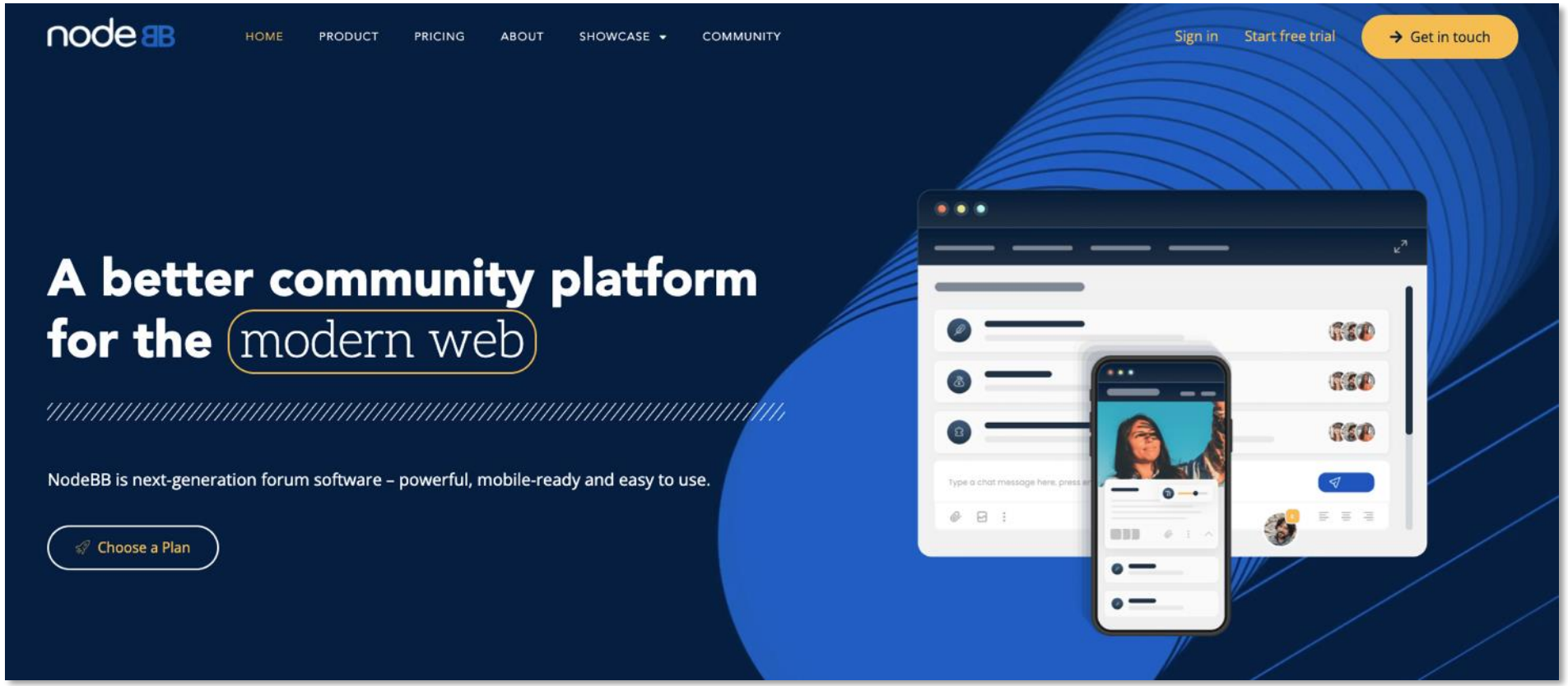
Server-side apps running remotely

Open Source	Closed Source
examine	
probe	Talk to NSA
modify	

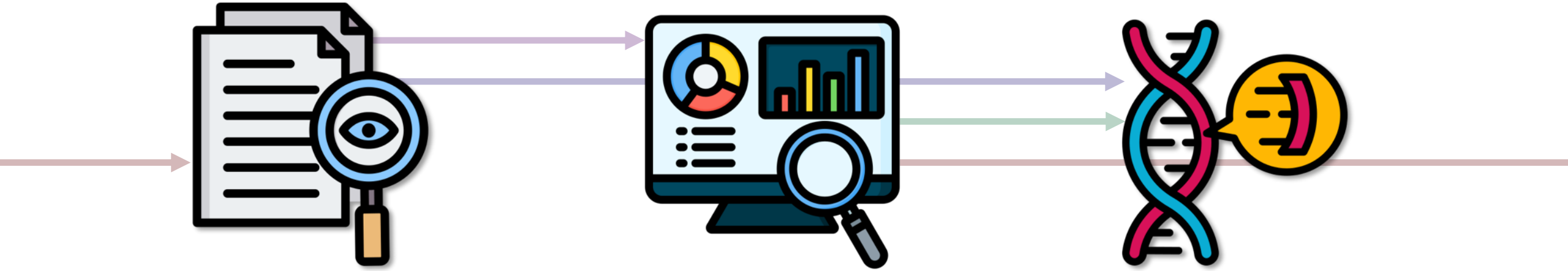
Creating a model of unfamiliar code

Source code built locally

Live Demonstration: NodeBB



How to build, test, and refine mental models



Examine
artifacts without running code

Probe
running system to observe behavior

Modify
code, rebuild, and assess impact

How to build, test, and refine mental models

Examine
artifacts without running code

Probe
running system to observe behavior

Modify
code, rebuild, and assess impact

Examine artifacts to build a mental model

🧐 Ask

- How do we build / test / run it?
- How is this system structured?
  - Where are the entrypoints?
  - Where are the seams? Can we probe them?
  - Where is data persisted?
- What technologies does it use?
- What are its stated features? Limitations?
- Is the project active?

🔎 Scan

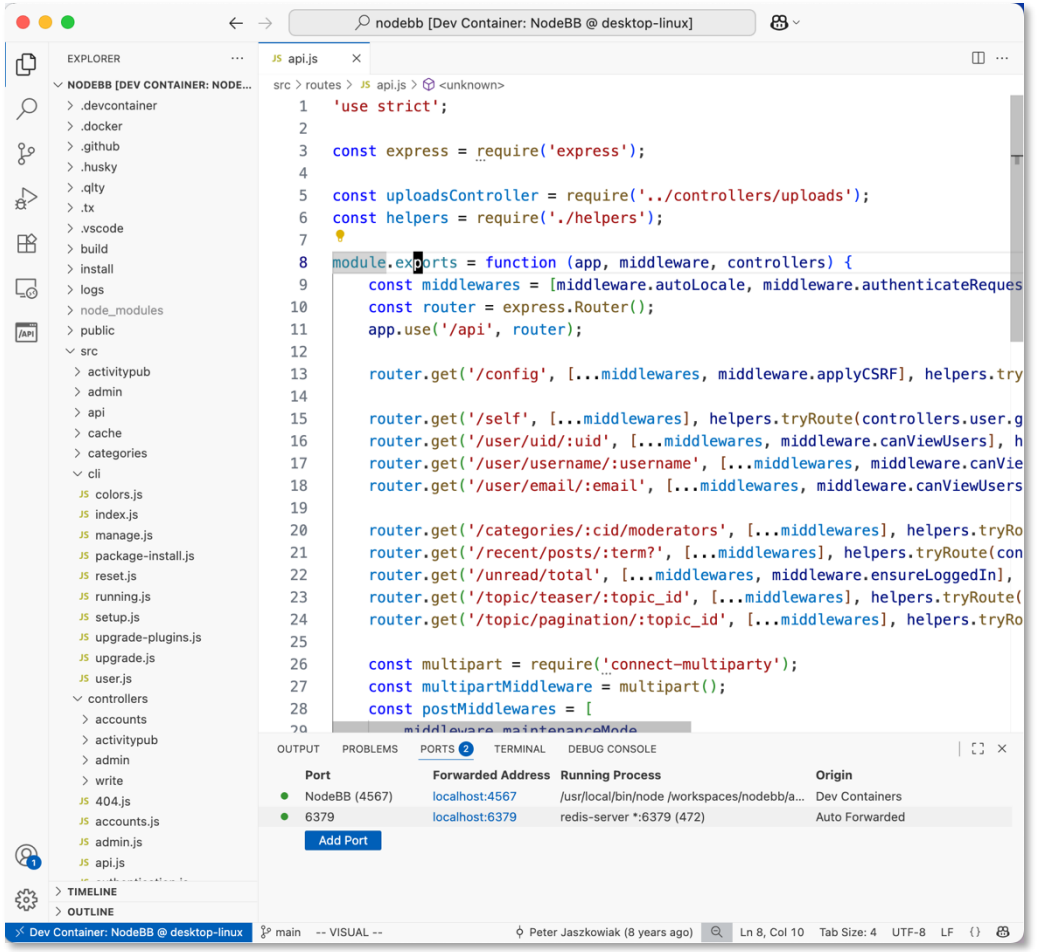
- Source: code
- Build/CI: package.json, Docker, workflows
- Config: env vars, config.json, ...
- Docs: README, Documentation
- History: commits, issues, PRs, projects

🏆 Goal

- a **build/run** command
- an **entry point** that you can target
- a **seam** that you can probe

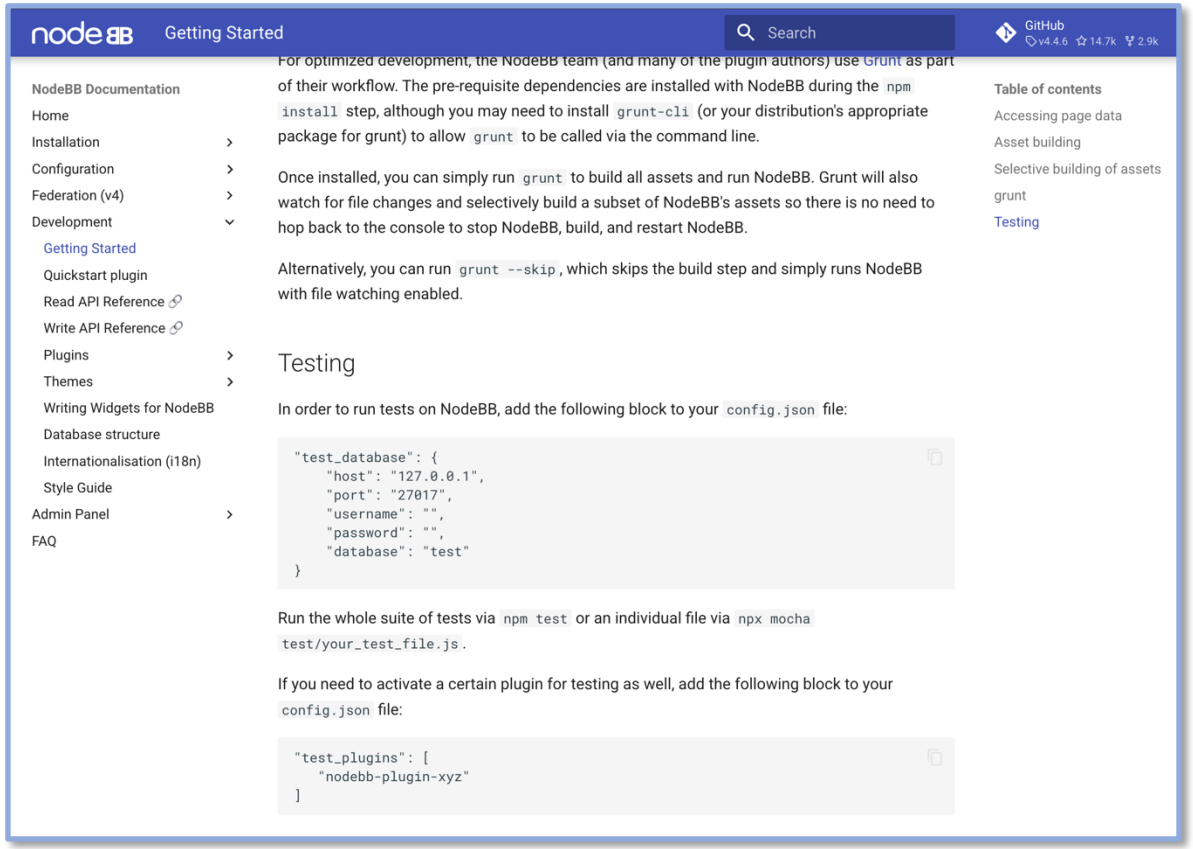
Tip: Configure and use your IDE to its full potential

- We will provide support for **DevContainers** in VSCode in this course
  - bundles together everything you need into a Docker image that behaves like a native install
- **Right click** on code to learn more
  - variables, functions, classes, modules, …
  - Go to Definition, Go to References, Rename Symbol, Refactor, …
- Install and explore IDE **Extensions**
  - Redis, ESLint, OpenAPI Editor, LiveShare, …



Tip: Consider documentation and tutorials judiciously

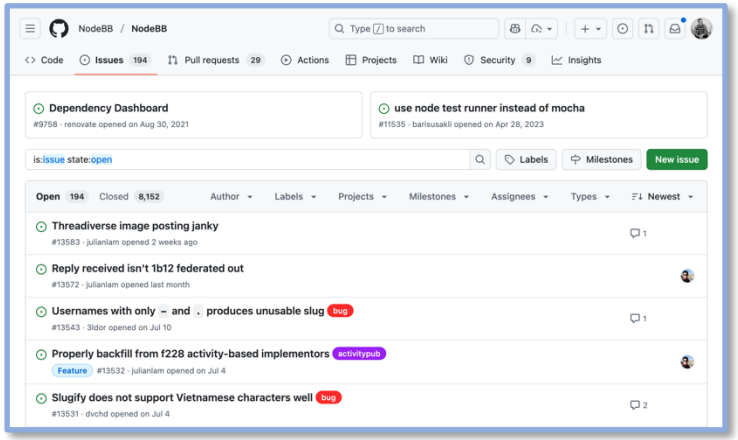
- Info on how to build the system, its dependencies, and how to use it
- Great for finding entry points
- Can tell you about the overall system architecture; more on that topic later in the semester



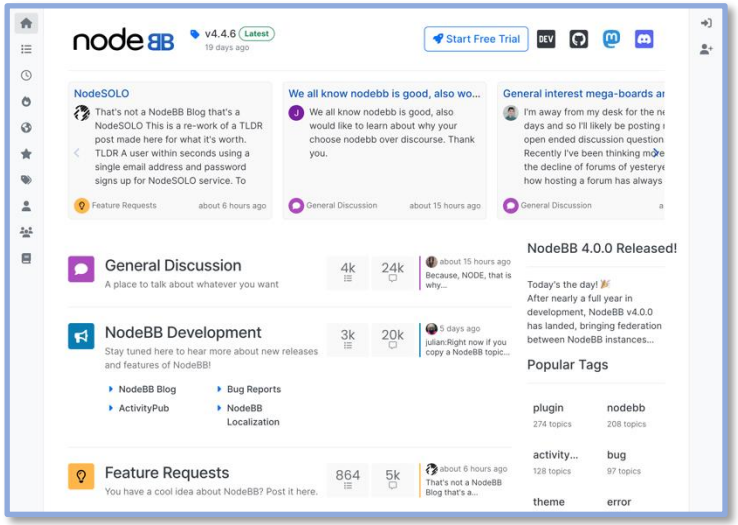
- ⚠ **Often out of date!** Treat as a starting point rather than truth

Tip: Use discussion boards and issue trackers

- Are features unimplemented?
- Is the project still being maintained?
- Is someone else having the same issue?



- Found an issue with the code? **File a GitHub issue**
- Having a hard time getting some to work? Trying to change something? **Post to the NodeBB forums**
- Have a question about {Node, Redis, Express, ...}? **Post to StackOverflow or Slack.**



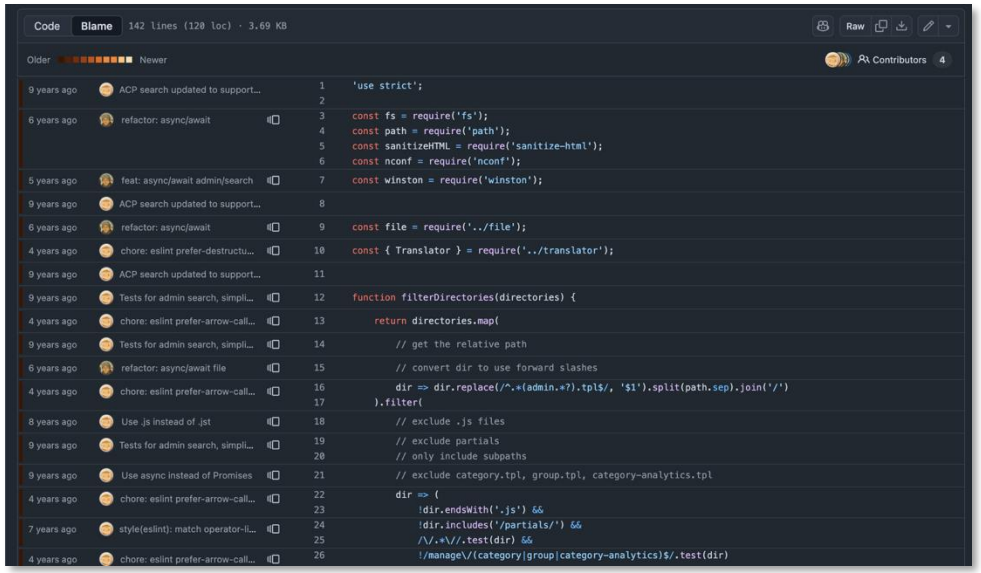
Tip: Use AI to explain parts of the code — but be careful

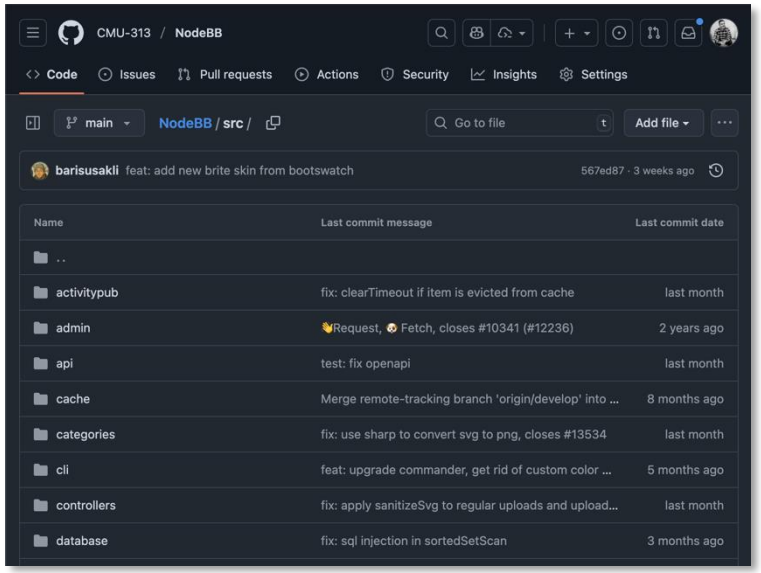
- Used carefully, AI tools can help you quickly tackle new codebases
- These tools fail confidently; expect errors and omissions, and cross-check against code, docs, and tests before trusting results.

- We will have a **whole lecture** on this new, emerging skill later in the course — for now, **experiment with AI**, **but don't rely on it**

Tip: Look at file structure, ownership, and history

- Files are not randomly named and organized. Directory structures and naming conventions reveal patterns.
- Inspect history to learn ownership and stability: identify contributors, recency of changes, and churn. Treat stale or recently rewritten files with caution.





How to build, test, and refine mental models

Examine
artifacts without running code

Probe
running system to observe behavior

Modify
code, rebuild, and assess impact

Probe to test your mental model

🧪 Hypothesis → Experiment

- Introduce a probe to observe the system at a given seam or entry point
- Use the observed behavior to confirm or refute your hypothesis
- Gradually build confidence in your understanding of system behavior

- **Example:**
  When I click X, handler Y runs
  → Set a breakpoint in Y then trigger X

🔬 Probes & Triggers

- Add breakpoints, logpoints, and step
- Logging: ./nodebb dev
- Print statements
- Bruno / Postman / curl / httpie
- Database viewers

🏆 Goal

- One confirmed or refuted hypothesis
- One short note (trigger → code path → signal)
- One next probe or modification

Tip: Instrument the source code

- **Print debugging**

```
console.log('Administrator found, skipping Admin setup');
```

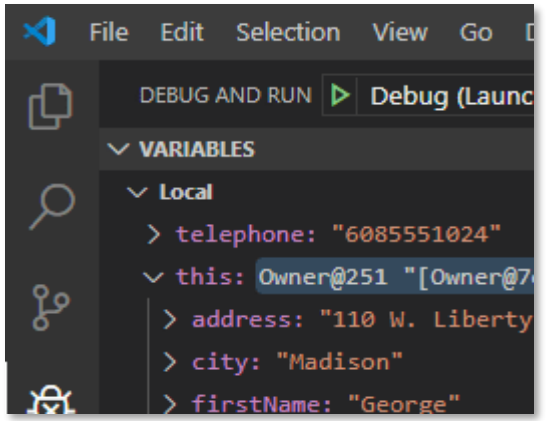
  - Quick and easy
  - Cons: need to rebuild + restart; easy to commit by accident

- **Structured logging**

```
winston.warn(`Flooding detected! Calls : ${socket.callsPerSecond}, Duration : ${socket.elapsedTime}`);
```

  - Add levels, timestamps, and context; better for collecting data in deployment
  - Cons: need to rebuild + restart

- **Debuggers**
  - Inspect locals, call stack, evaluate expressions
  - Add breakpoints as you go; no need to rebuild + restart
  - No changes to the code means no risk of accidental changes
  - We will explore the debugger in more depth later in the course

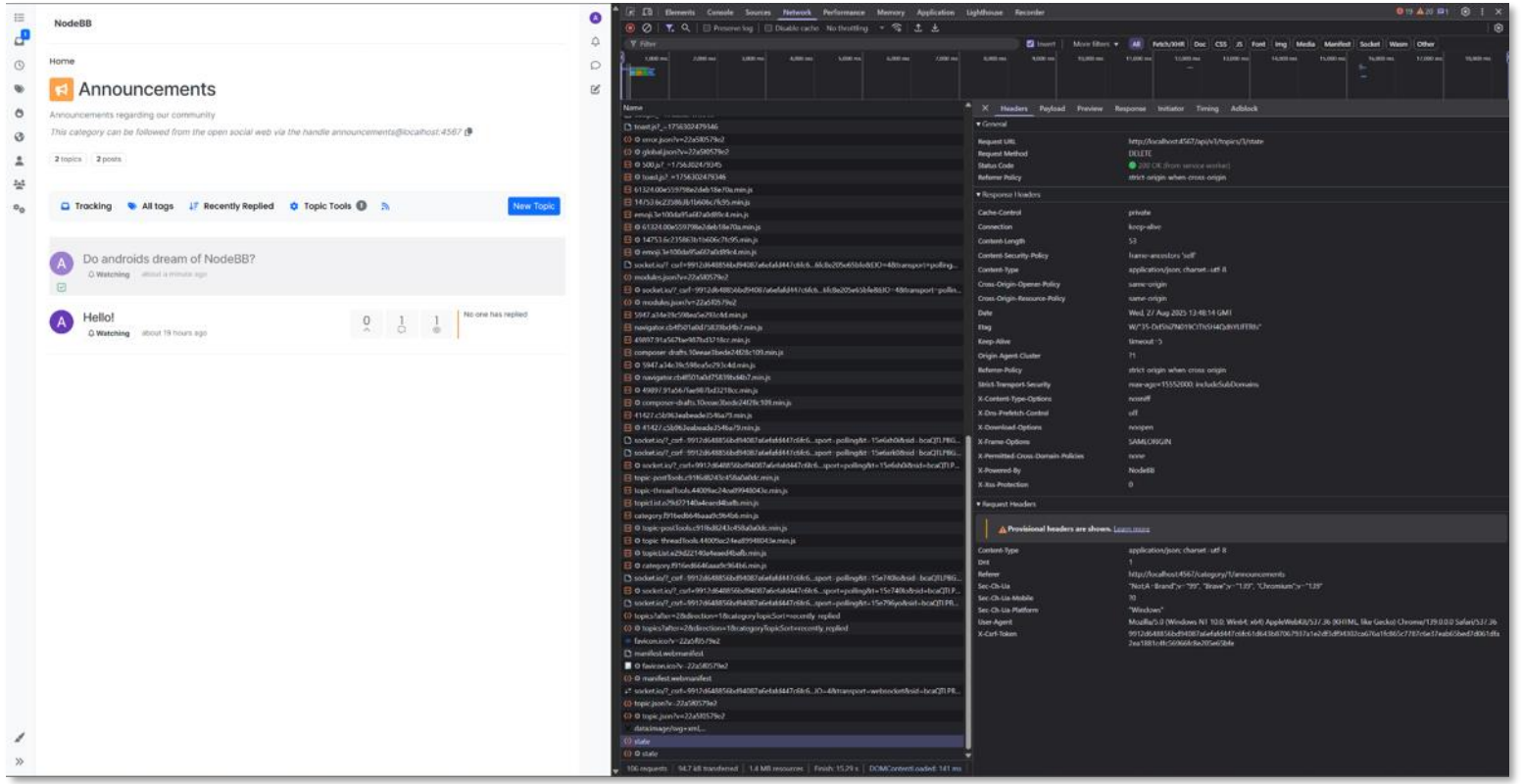


Tip: Use developer tools in the browser to spy on traffic

- Spy on web traffic while you use the app

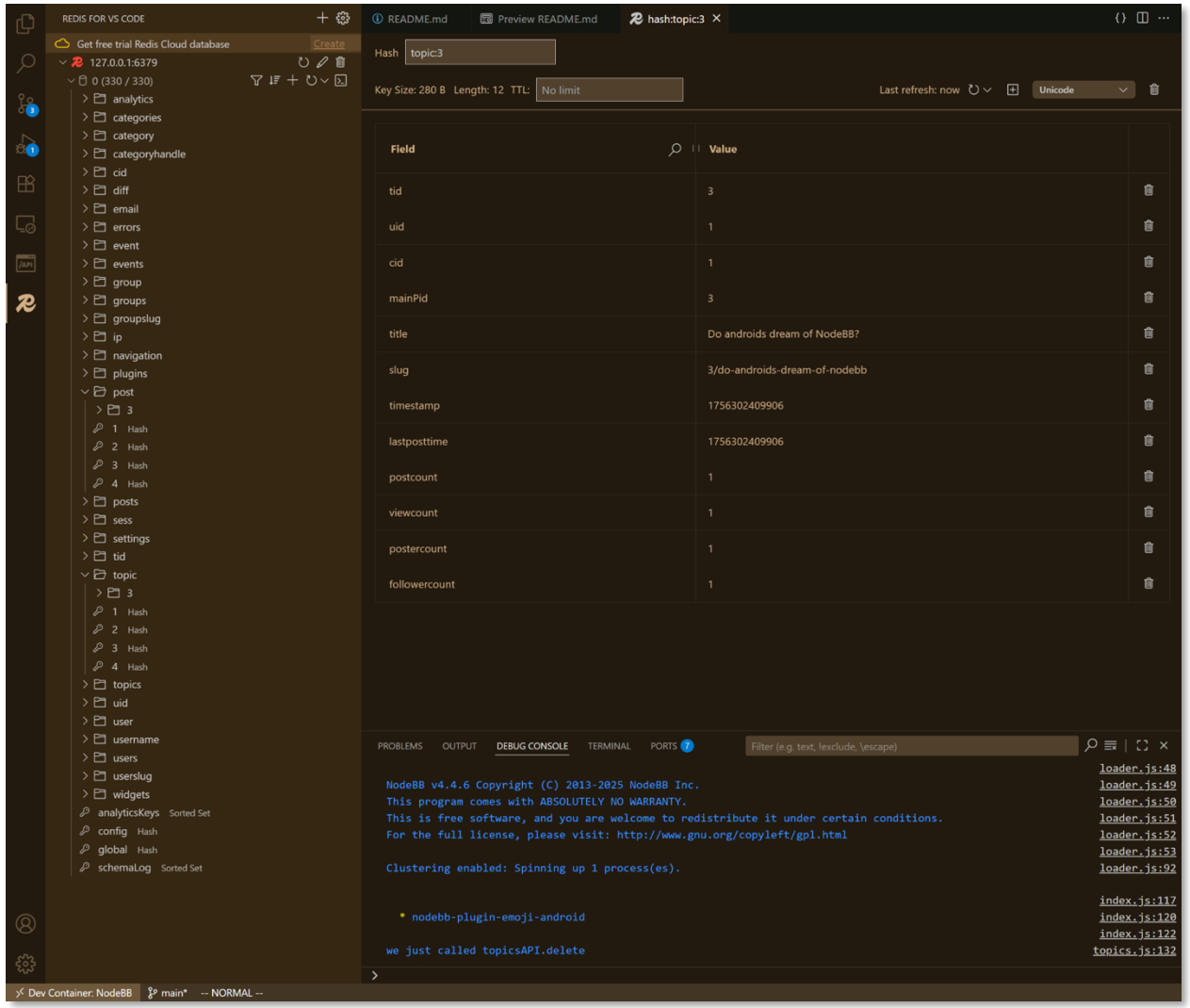
- Chrome DevTools (also used by Brave)
- Firefox Dev Tools
- Safari Web Inspector

- **Bonus:** Use Bruno, Postman, httpie, or curl to trigger API requests



Tip: Peek at the database

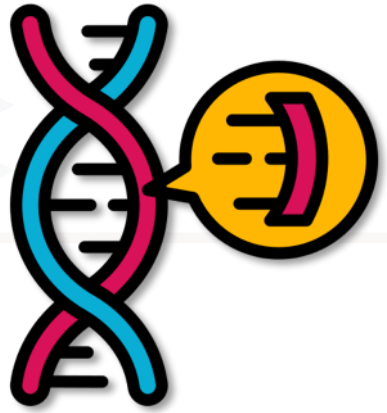
- Use the **Redis extension** that's provided with the DevContainer
- Perform an action (e.g., create or delete a topic) and watch which keys / fields change
  - filter by **prefix** to keep things manageable (topic:*, post:*, user:*)
- Use to confirm or refute your hypotheses about data flow



How to build, test, and refine mental models

Examine
artifacts without running code

Probe
running system to observe behavior



Modify
code, rebuild, and assess impact

Modify code to validate your model

🏗️ Plan and execute your change

- What **behavior** should change if your model is correct?
- What's the **simplest change** that you can make?
- What **signal** can you observe? (user interface, API, logs, database, test case)

- Rebuild the code and see what happens!
- Tip: **delete debugging** is a powerful tool

📈 Assess impact

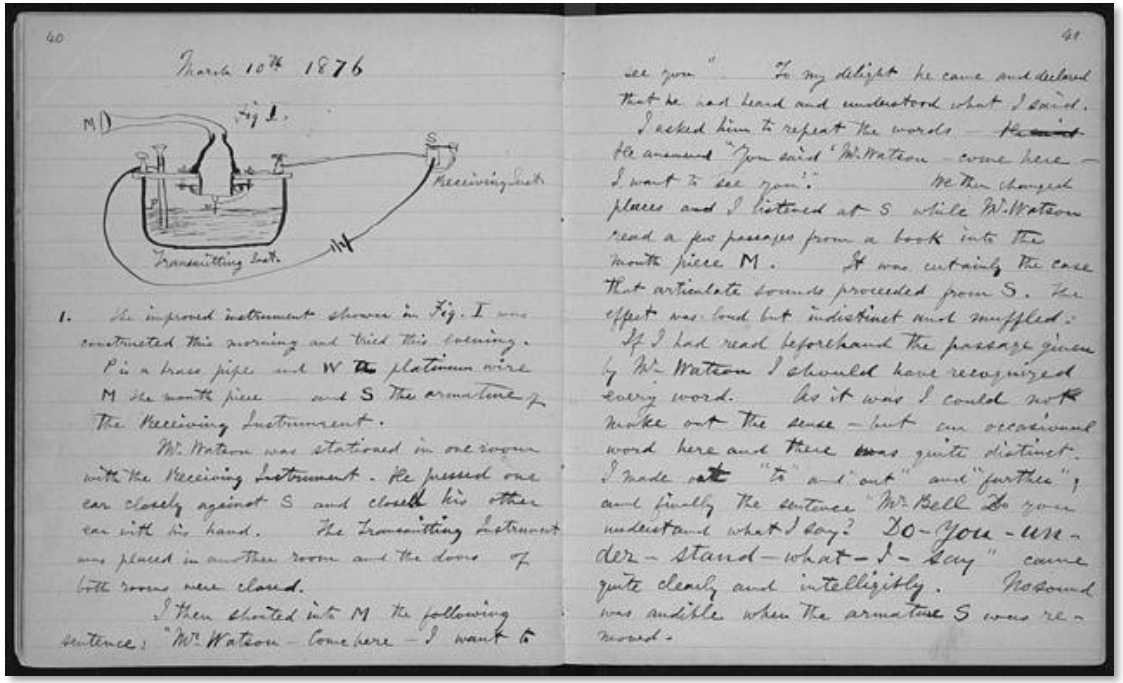
- Did the **predicted signal** change?
- If yes, your model **holds** for now.
- If not, you need to **revise** it.

🏆 Goal

- One change with a clear effect
- A note of what it confirms or refutes
- A next step (examine, probe, modify)

Document and share your findings!

- Update README and docs
  - Or better: use a **Developer Wiki**
  - Use Mermaid for diagrams
- Collaborate with others
  - use LiveShare to debug, explore, and program collaboratively
- Include negative results, too!



Next Time: 737-MAX Case Study