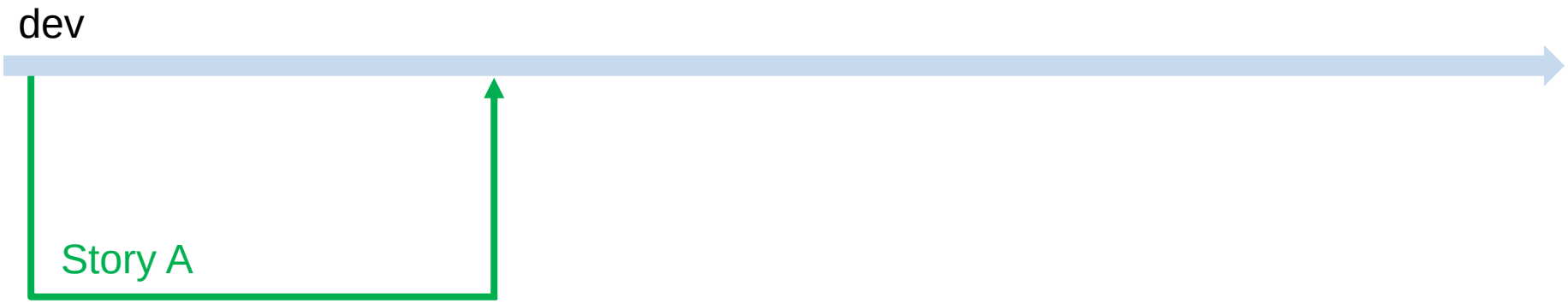
Story-Branch Workflow

- AKA Feature-Branch
- Each User Story (rather than each Sprint) has its own branch
- One Developer creates a Story-Branch
- Many Developers may implement Tasks for that Story and push their commits

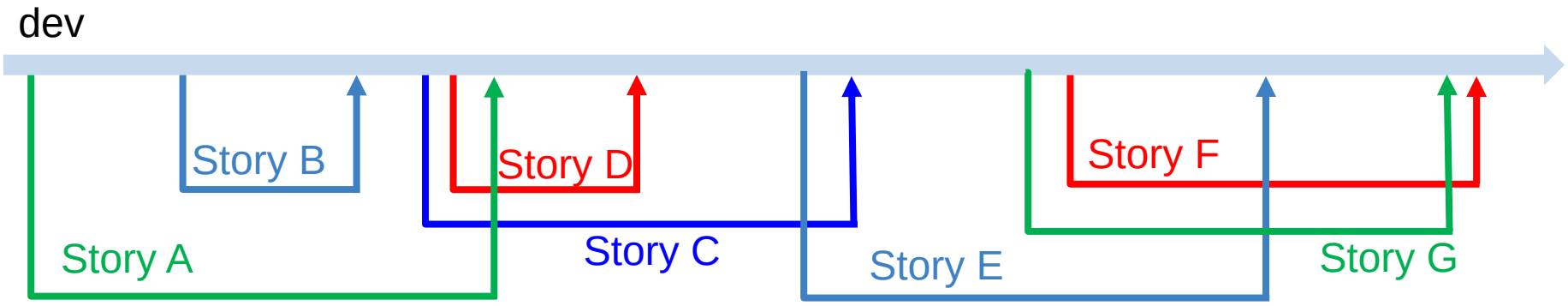
Story-Branch Workflow

- AKA Feature-Branch
- Each User Story (rather than each Sprint) has its own branch
- One Developer creates a Story-Branch
- Many Developers may implement Tasks for that Story and push their commits



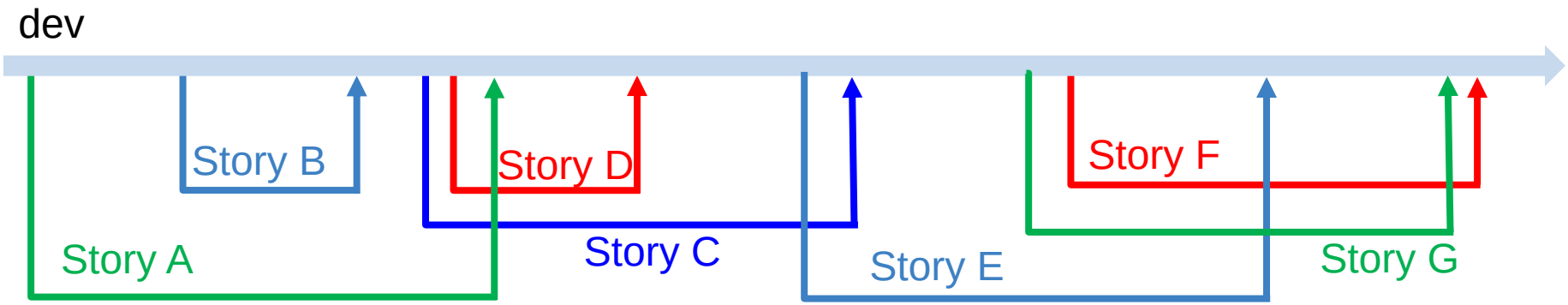
Story-Branch Workflow

- AKA Feature-Branch
- Each User Story (rather than each Sprint) has its own branch
- One Developer creates a Story-Branch
- Many Developers may implement Tasks for that Story and push their commits



Story-Branch Workflow

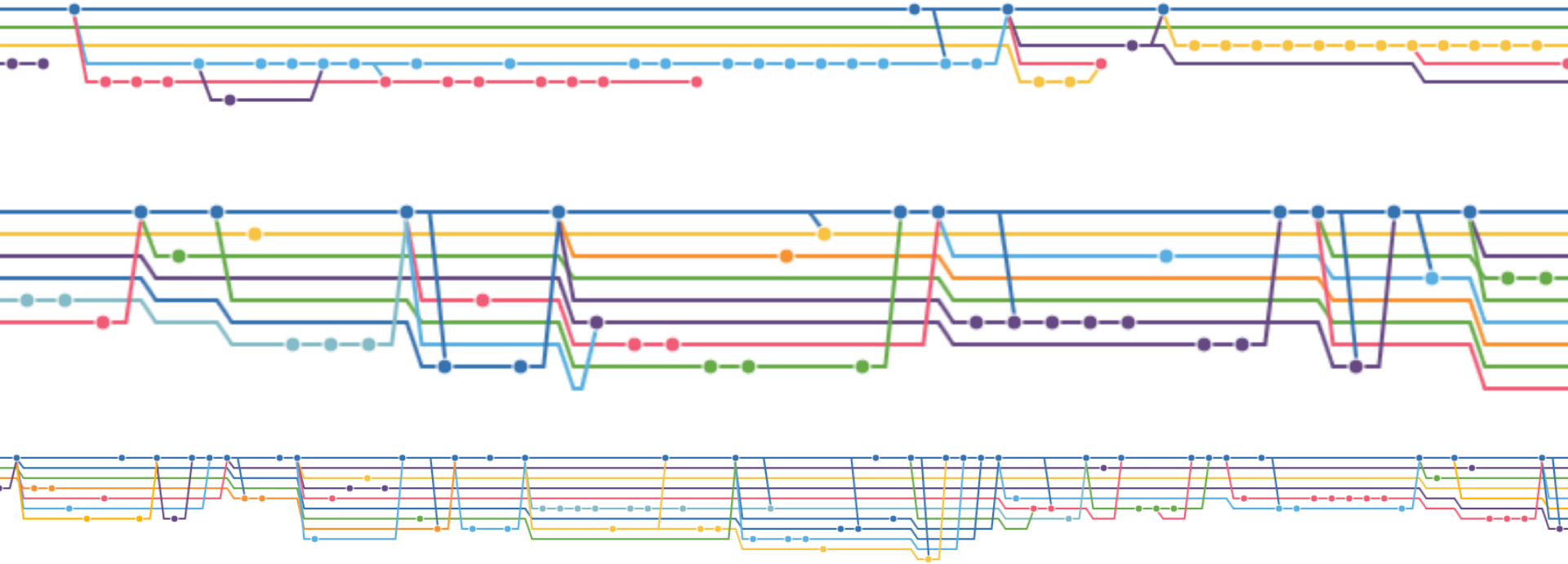
- Advantages:
  - ???
- Disadvantages:
  - ???
- When to Use:
  - ???



Story-Branch Workflow

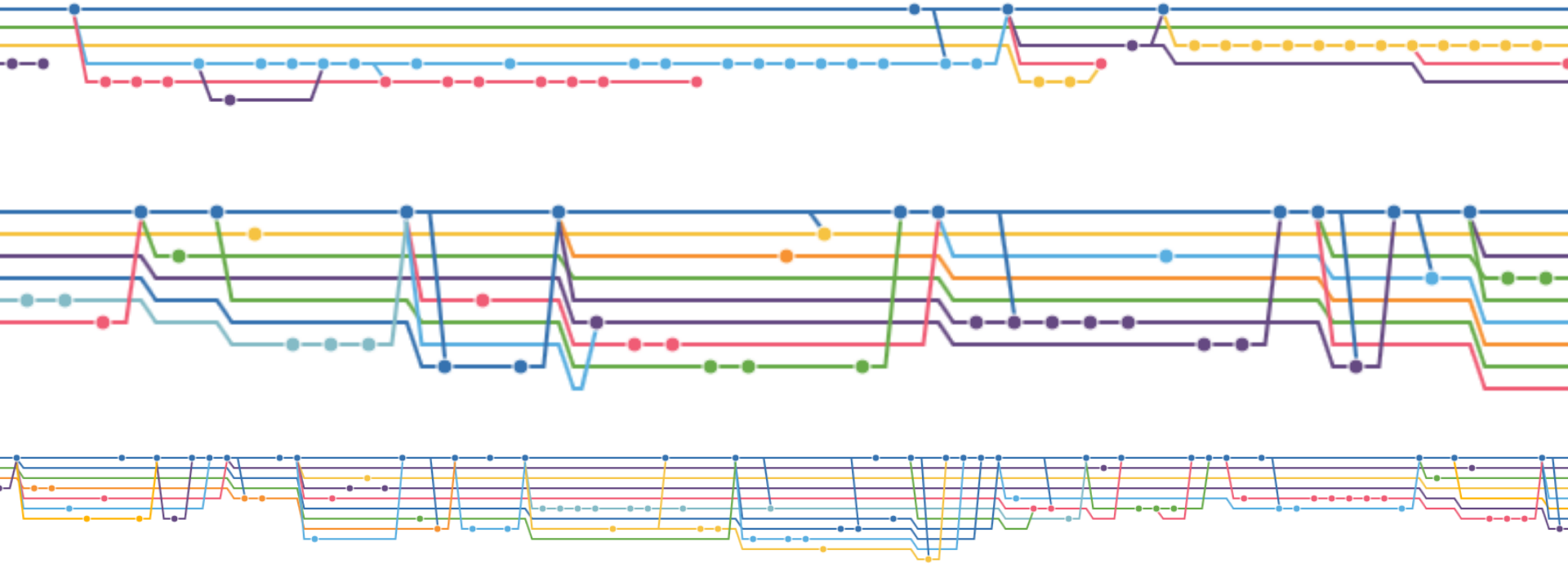
- Advantages:
  - Allows strong collaboration on each story
  - Isolates one story's changes from those of other stories
- Disadvantages:
  - Developers have to switch branches a lot
  - Potential integration issues during the sprint and at the end of the sprint (when multiple stories may be merged to master)
- When to Use:
  - For implementing risky stories whose outcome is uncertain

Story-Branch Workflow Examples

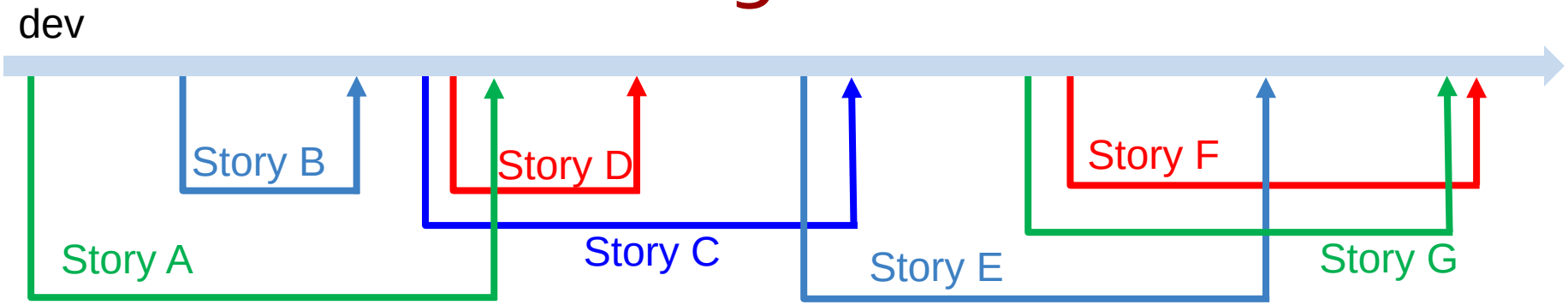


Story-Branch Workflow Examples

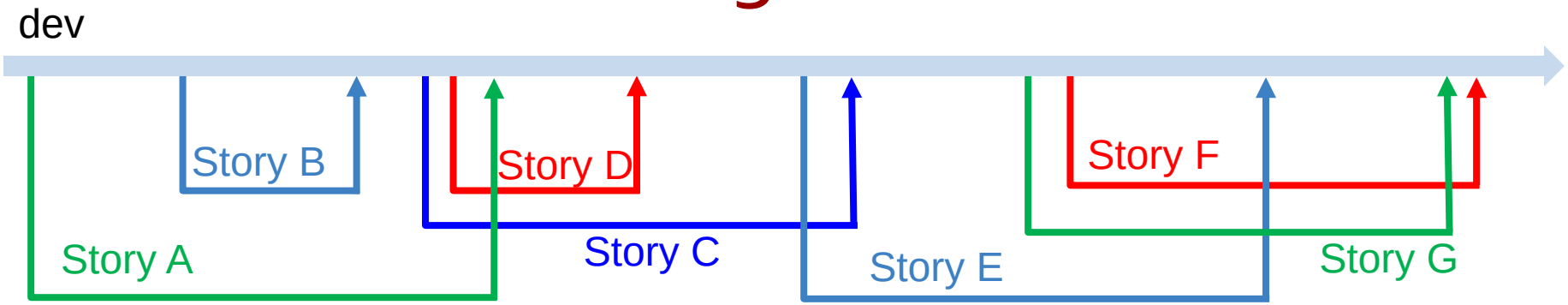
- Multiple "parallel" versions of the code in different branches
- Developers have to switch branches a lot
- Non-linear (cluttered) history



Decluttering the Branches

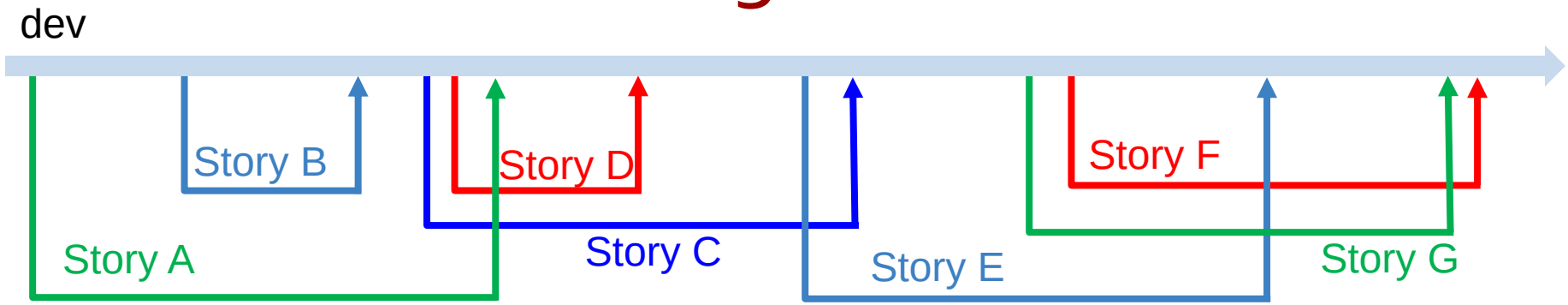


Decluttering the Branches

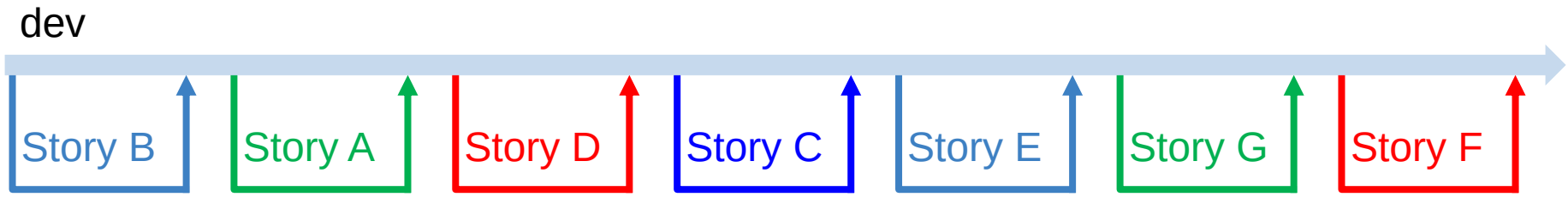


Using the squashing and rebasing functionality of git, we can transform the cluttered history into…

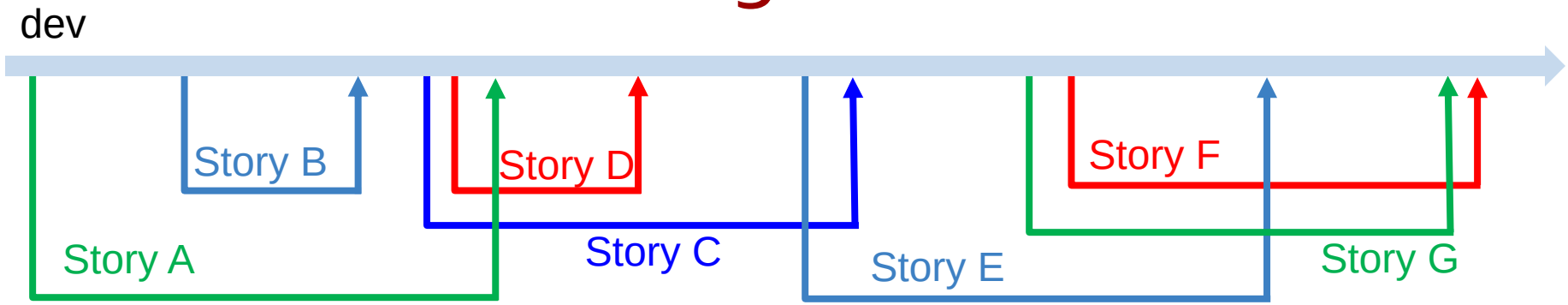
Decluttering the Branches



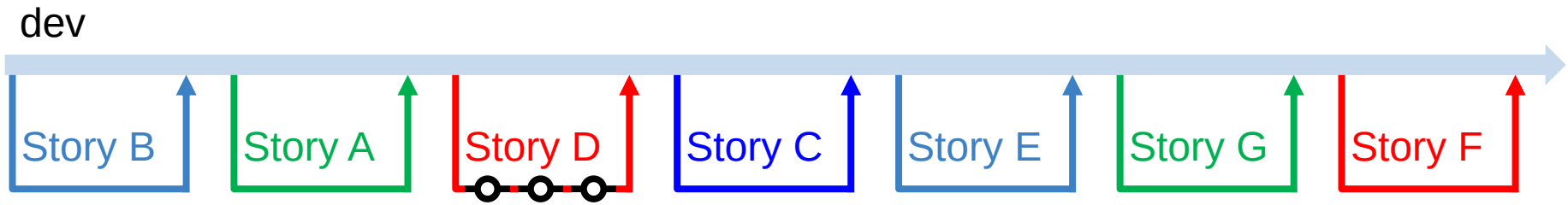
Using the squashing and rebasing functionality of git, we can transform the cluttered history into…



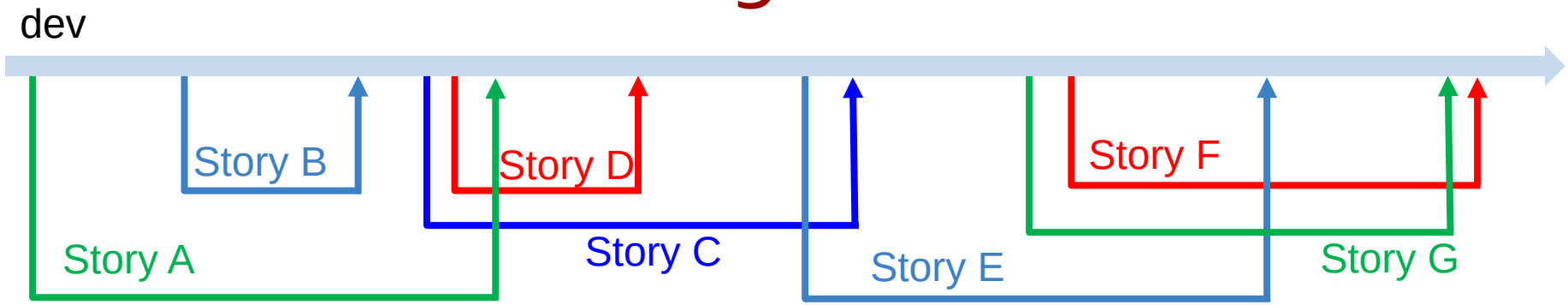
Decluttering the Branches



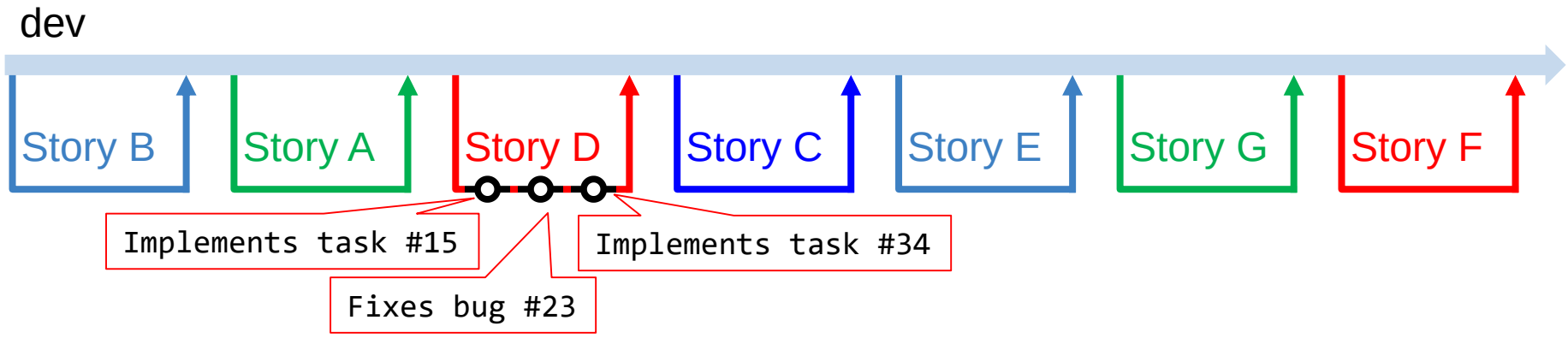
Using the squashing and rebasing functionality of git, we can transform the cluttered history into…



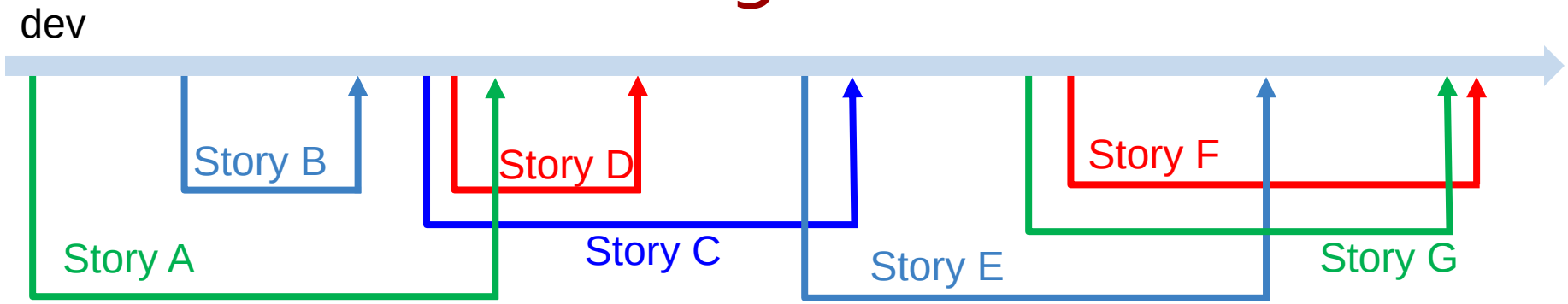
Decluttering the Branches



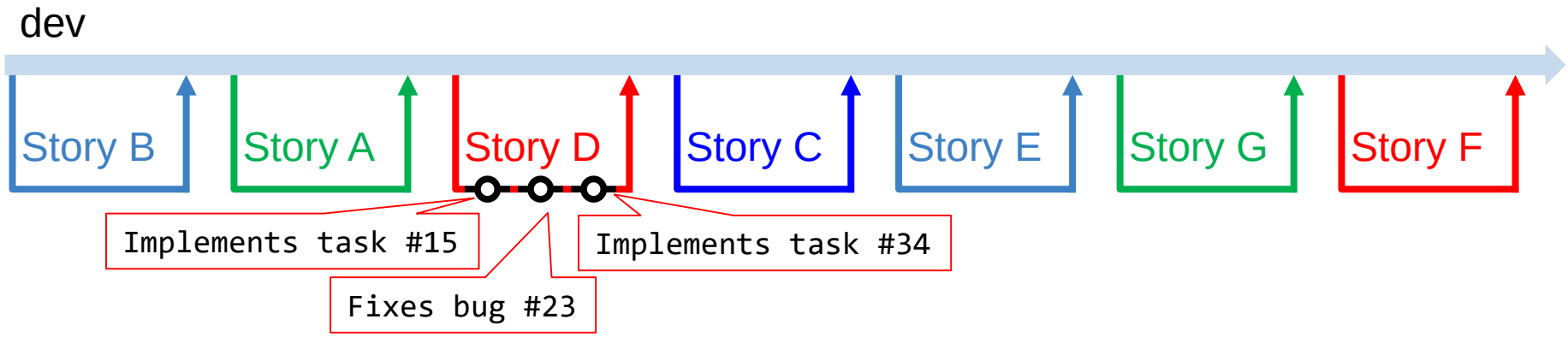
Using the squashing and rebasing functionality of git, we can transform the cluttered history into…



Decluttering the Branches



Using the squashing and rebasing functionality of git, we can transform the cluttered history into…



Each branch contains only atomic commits that implement tasks/bugs related to the story (e.g., #15, #23 and #34 belong to Story D)

Workflow trends discussed so far…

Workflow trends…

- "Module-branch" workflow (Waterfall)
- Sprint-branch workflow
- Story-branch workflow

Workflow trends…

- "Module-branch" workflow (Waterfall)
  - "Big-Bang" integration issues
- Sprint-branch workflow
  - Potential mini "Big-Bang" integration at end of sprint
- Story-branch workflow
  - Potential (smaller) integration issues through the sprint
- Tendency was for developers to focus on smaller changes ⇒ rapid feedback ⇒ increased productivity

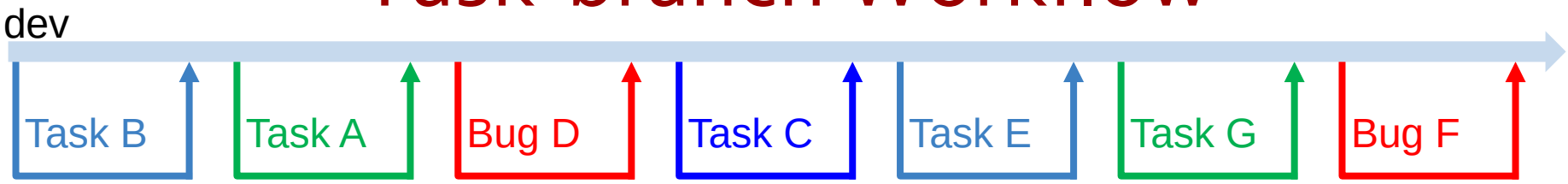
Workflow trends…

- "Module-branch" workflow (Waterfall)
  - "Big-Bang" integration issues
- Sprint-branch workflow
  - Potential mini "Big-Bang" integration at end of sprint
- Story-branch workflow
  - Potential (smaller) integration issues through the sprint
- ? workflow

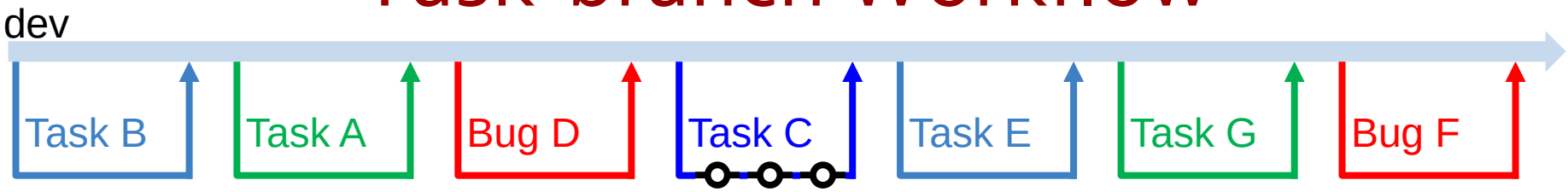
Workflow trends…

- "Module-branch" workflow (Waterfall)
  - "Big-Bang" integration issues
- Sprint-branch workflow
  - Potential mini "Big-Bang" integration at end of sprint
- Story-branch workflow
  - Potential (smaller) integration issues through the sprint
- Task-branch? workflow

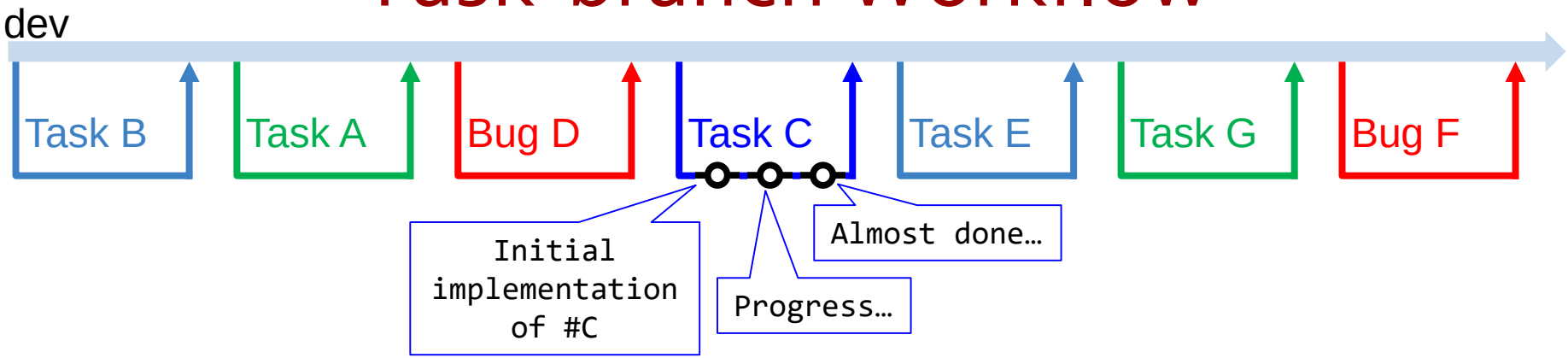
Task-branch Workflow



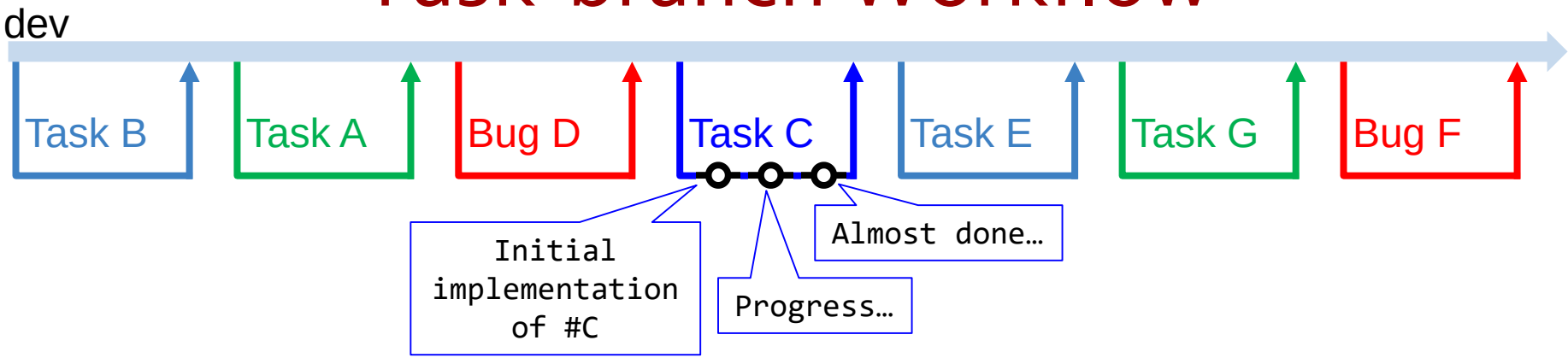
Task-branch Workflow



Task-branch Workflow

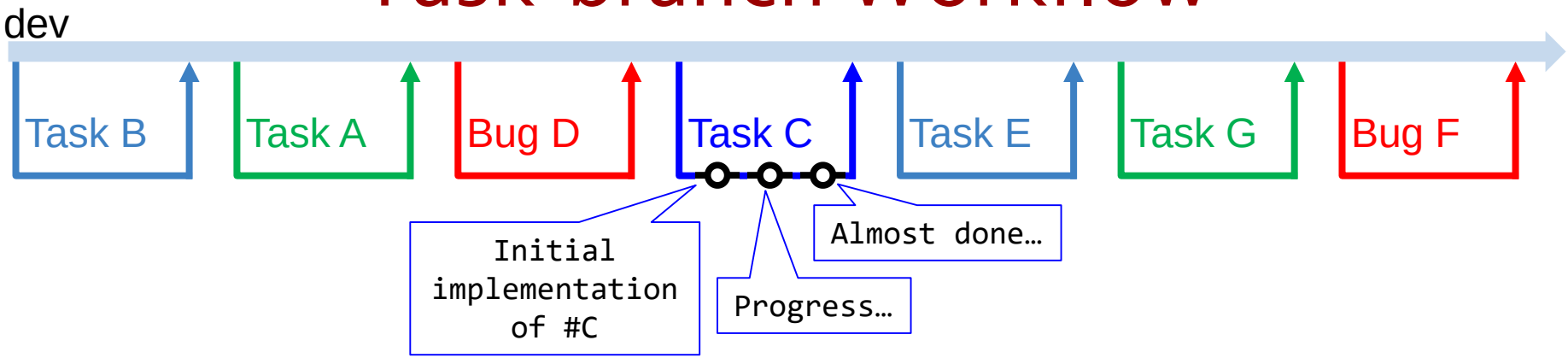


Task-branch Workflow

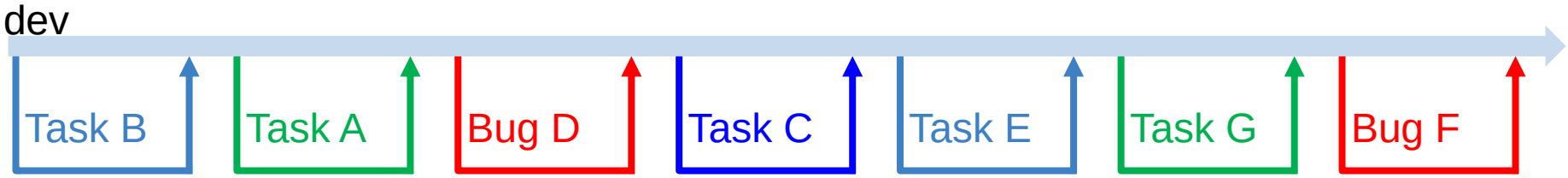


After squashing the 3 commits into one commit with a precise commit log message

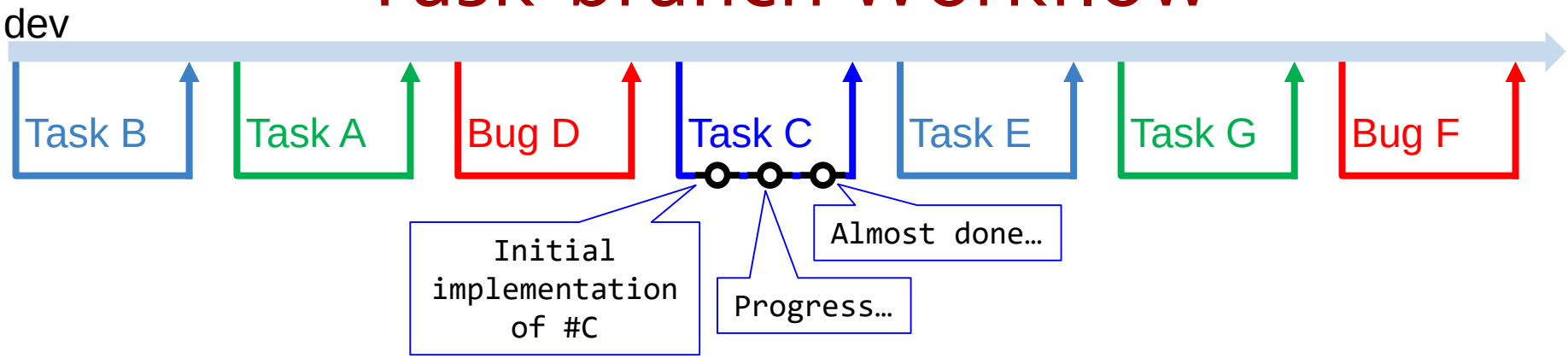
Task-branch Workflow



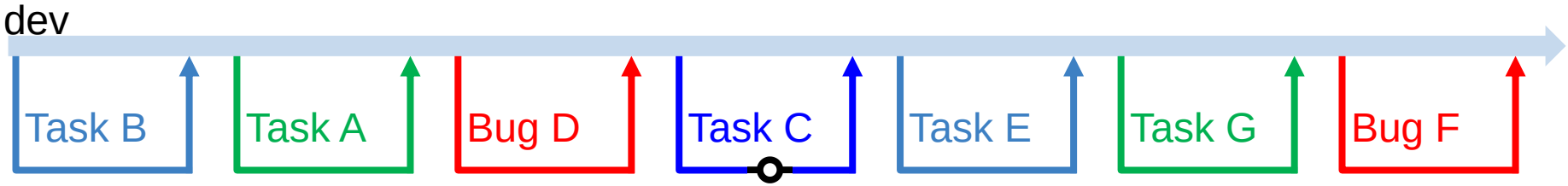
After squashing the 3 commits into one commit with a precise commit log message



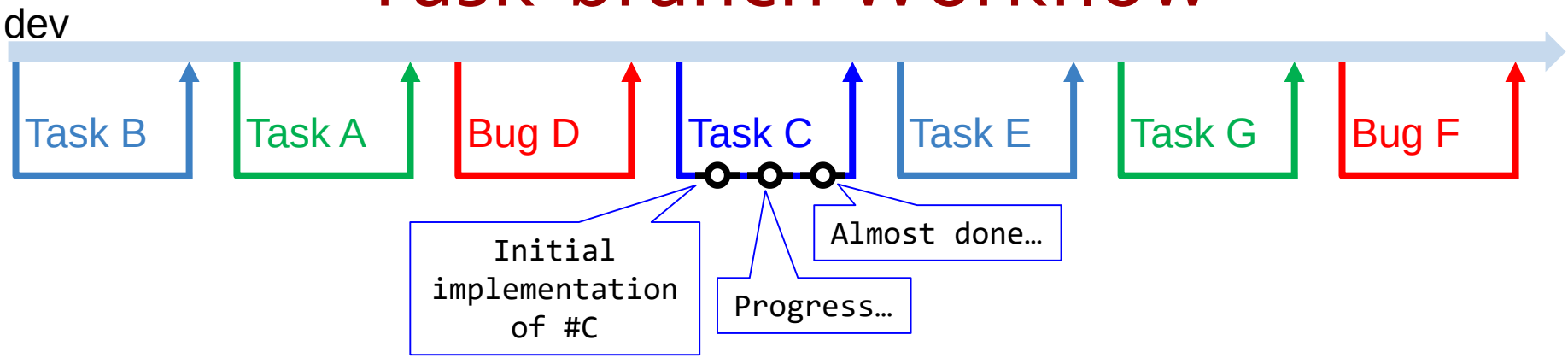
Task-branch Workflow



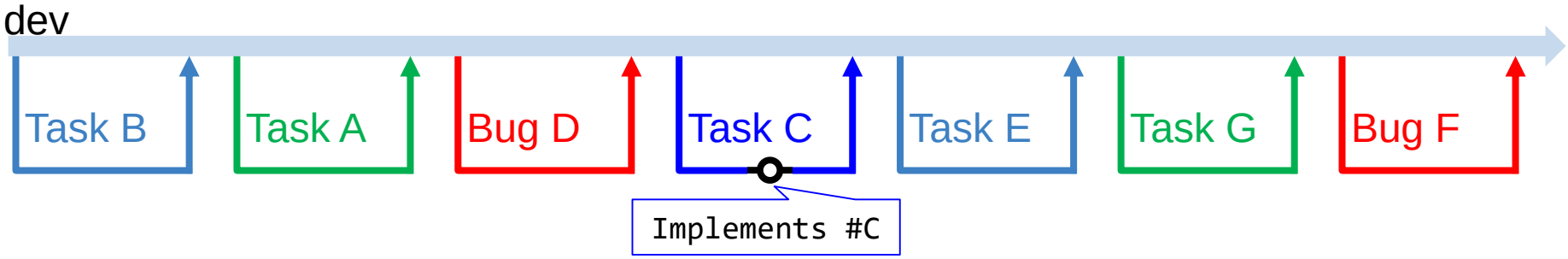
After squashing the 3 commits into one commit with a precise commit log message



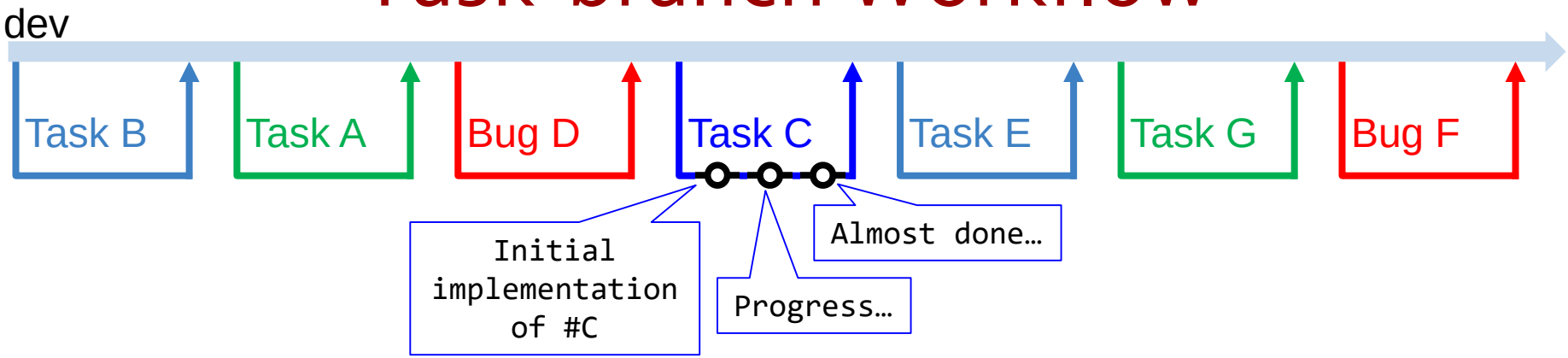
Task-branch Workflow



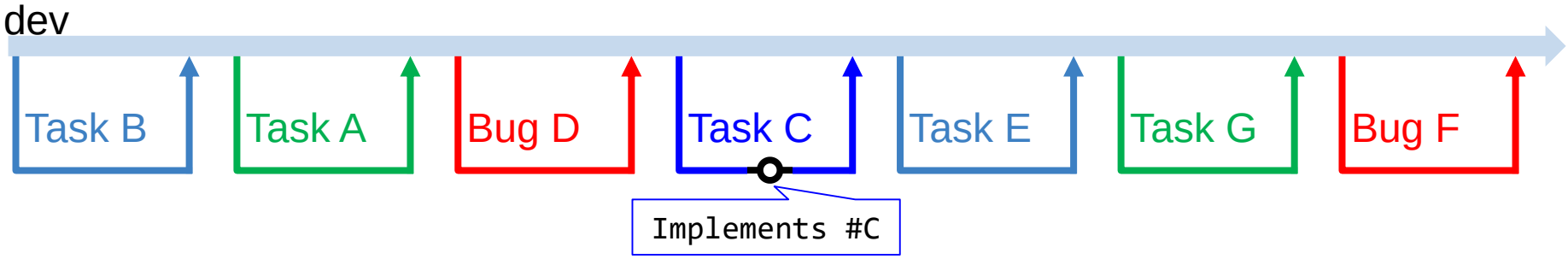
After squashing the 3 commits into one commit with a precise commit log message



Task-branch Workflow

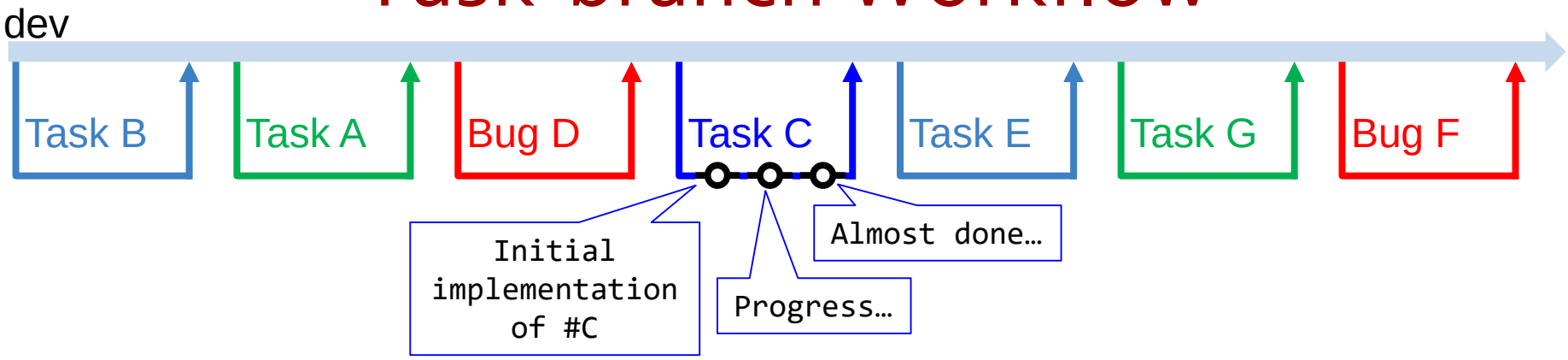


After squashing the 3 commits into one commit with a precise commit log message

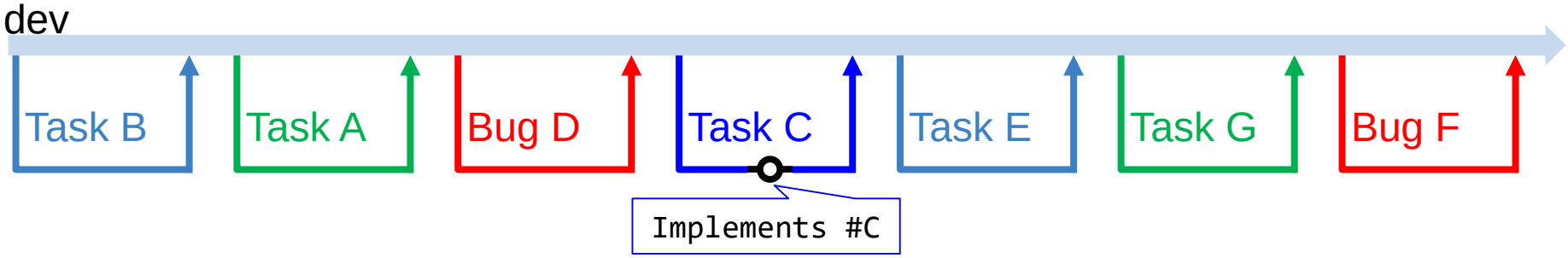


After rebase-ing the commit and deleting the branch

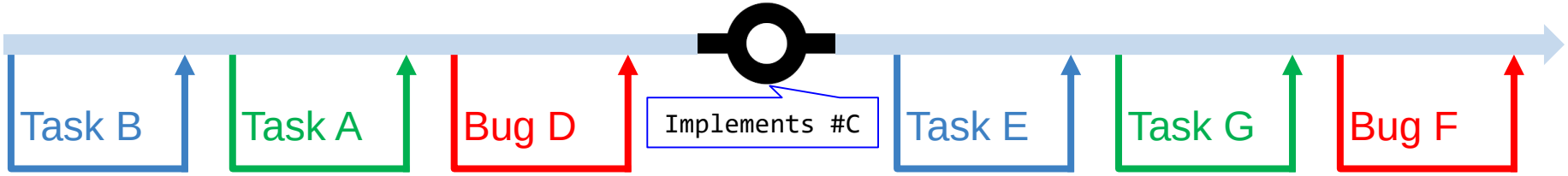
Task-branch Workflow



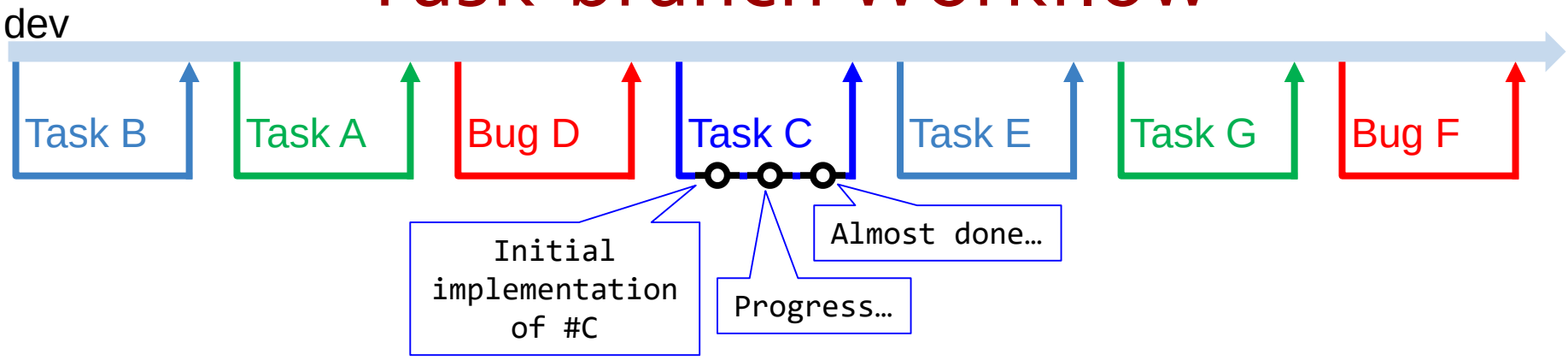
After squashing the 3 commits into one commit with a precise commit log message



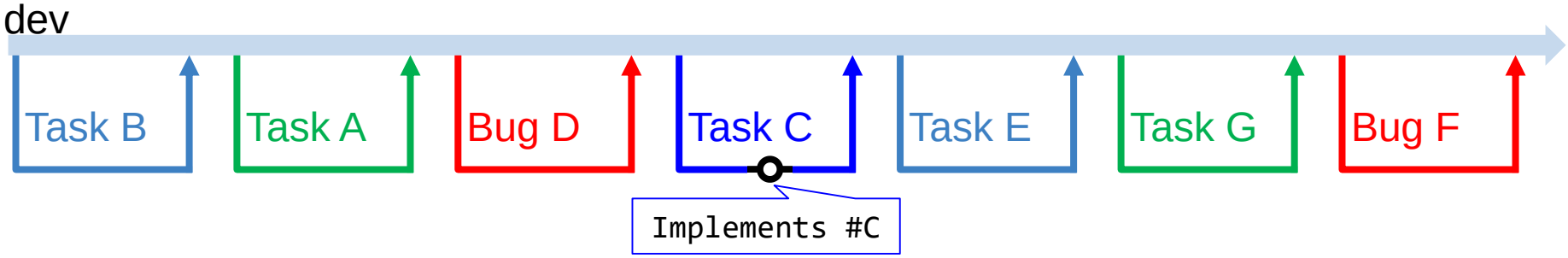
After rebase-ing the commit and deleting the branch



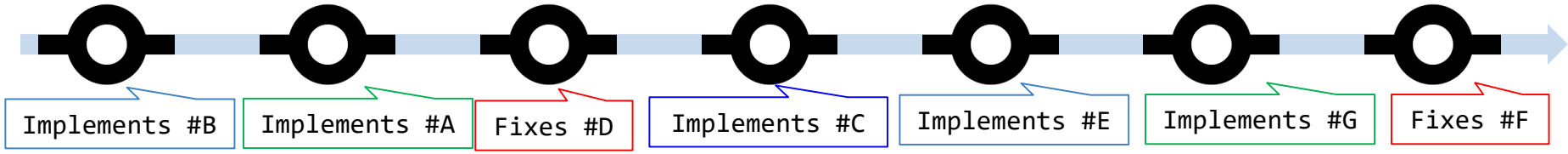
Task-branch Workflow



After squashing the 3 commits into one commit with a precise commit log message



After rebase-ing the commit and deleting the branch



Workflow Questions

- Why don't all the Developers simply commit their changes to the *master branch*?

Workflow Questions

- Why don't all the Developers simply commit their changes to the *master branch*?

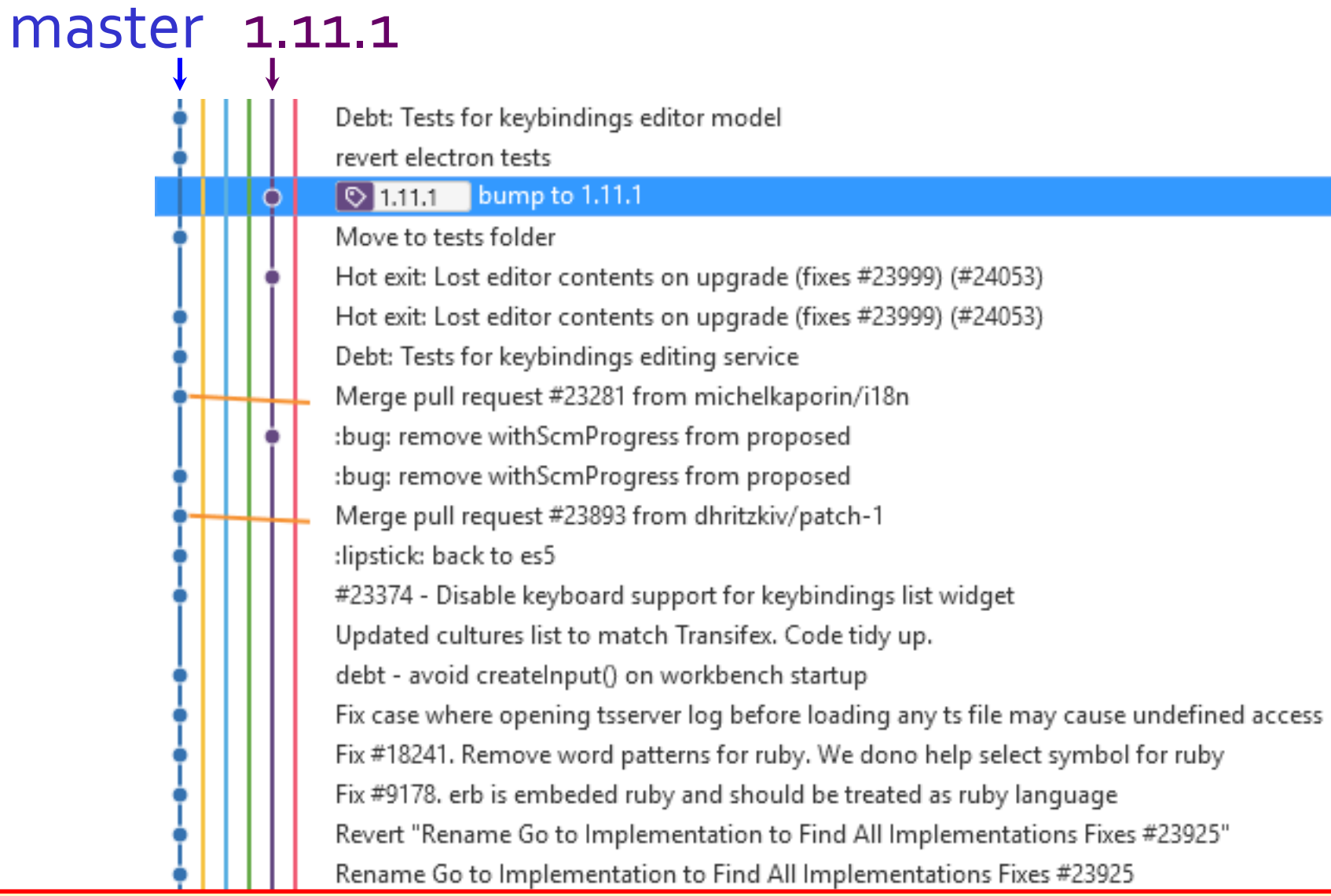
- How does the workflow used in your internship differ from the one we discussed? Advantages, disadvantages?

Workflow of Visual Studio Code: Submit to master / polish release branches

- Visual Studio Code* uses a workflow where:
  - developers test their newly implemented code in their local sandboxes
  - submit tested code to master (triggering CI)
  - create a "release" branch at regular intervals and the release branch is "synced" with the master
    - some last minute commits to master are incorporated in the branch
      - any changes submitted to the master, but not incorporated in the release branch will be used in future releases
    - fixes in the branch are incorporated back into the master

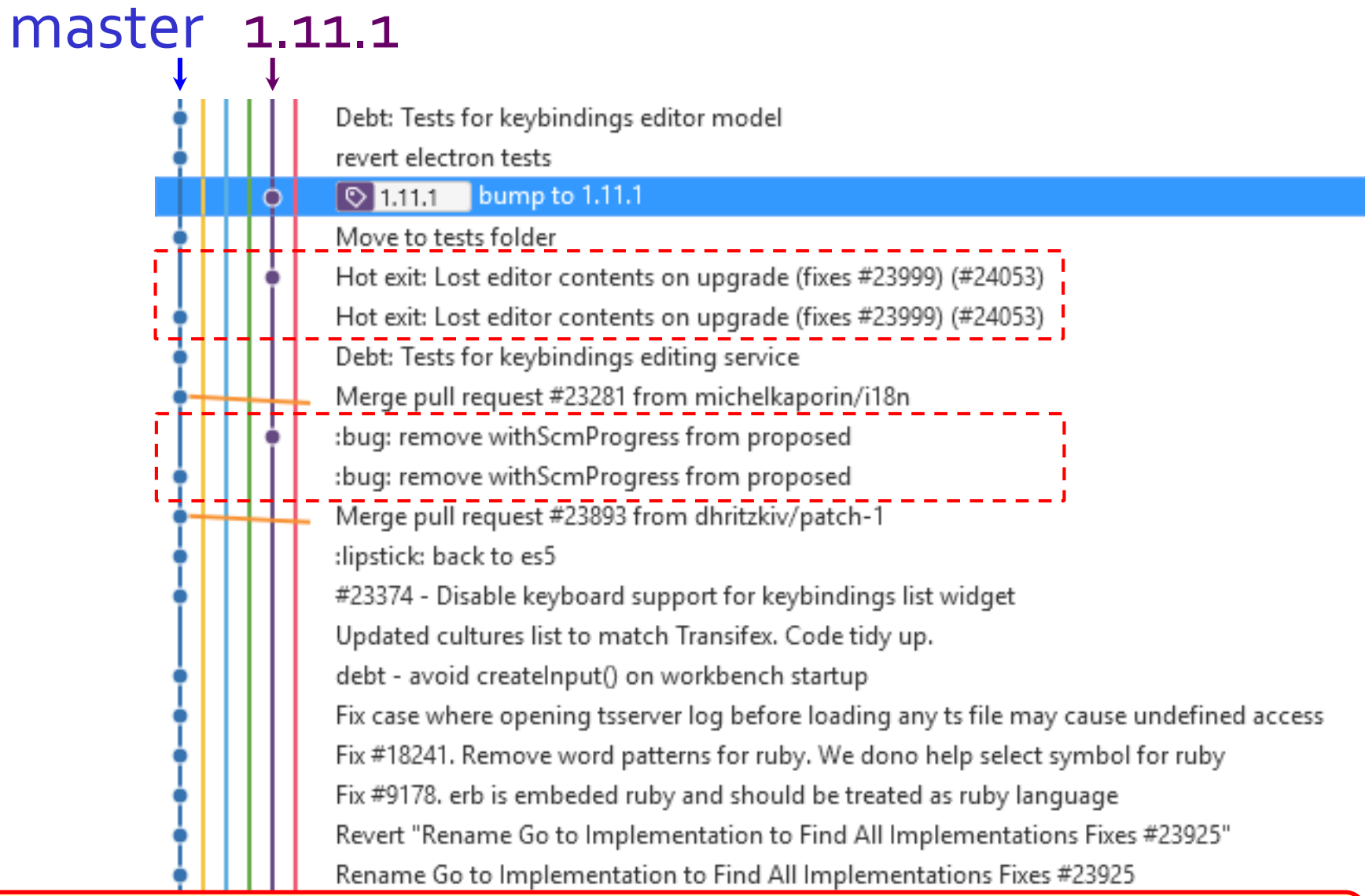
https://github.com/Microsoft/vscode

Workflow of Visual Studio Code



`1.11.1 is a release branch` created from `master`

Workflow of Visual Studio Code



`1.11.1 is a release branch created from master`
`(1.11.1 and master are kept in sync for the most part)`