12 Survey	Mon Apr 07	29. Software Engineering I [Optional] Recording		10. Graphs II, Tries Regular / Solutions / Video / Slides Exam Prep / Solutions / Video / Slides	Lab 10: Tetris (Optional)		Project 3A: World Generation (due 4/18)
	Wed Apr 09	30. Merge Sort, Insertion Sort, and Quick Sort Video / Recording / Pacing	Ch 30				
	Fri Apr 11	31. Software Engineering II [Optional]					
13 Survey	Mon Apr 14	32. Software Engineering III [Optional]	Ch 32	11. Sorting	Project 3 Workday	Homework 4 (due 05/04)	
	Wed Apr 16	33. More Quicksort, Quick Select, Stability	Ch 32				
	Fri Apr 18	34. Sorting and Algorithmic Bounds	Ch 34				
14 Survey	Mon Apr 21	35. Software Engineering IV [Optional]		12. Sorting II	Project 3 Workday		Project 3B: Interactivity (due 4/27)
	Wed Apr 23	36. Radix Sorts	Ch 35				
	Fri Apr 25	37. Sorting Conclusion, Algorithm Design Practice	Ch 36				

Lecture 31

Software Engineering II

CS61B, Spring 2025 @ UC Berkeley

Industry Practices for Collaboration

Introduction

Lecture 31, CS61B Spring 2025

Stella Kaval (she/her)

- 4th year, Computer Science
- 4th semester on CS61B staff
- Prev. intern at Microsoft, Oracle
- Favorite Taylor Swift album: Reputation

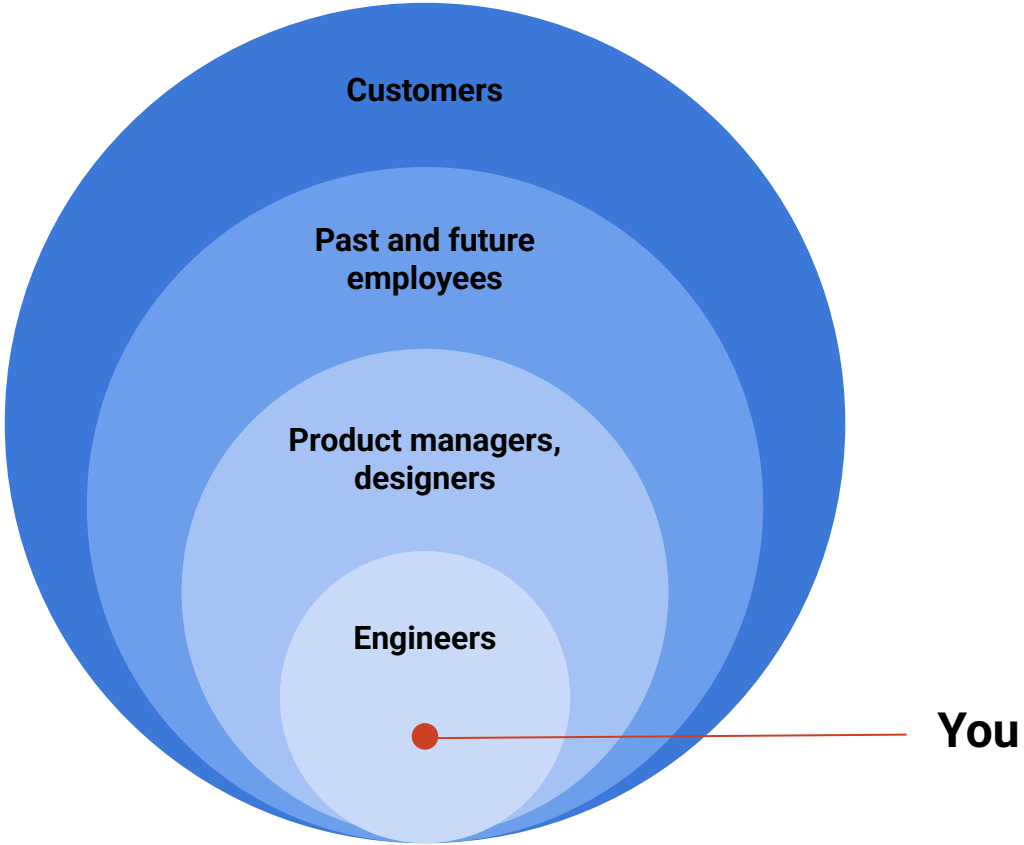
Kanav Mittal (he/him)

- 4th year, EECS
- 3rd semester on CS61B staff
- Prev. intern at a data privacy startup then at NVIDIA
- Favorite Taylor Swift album: Folklore

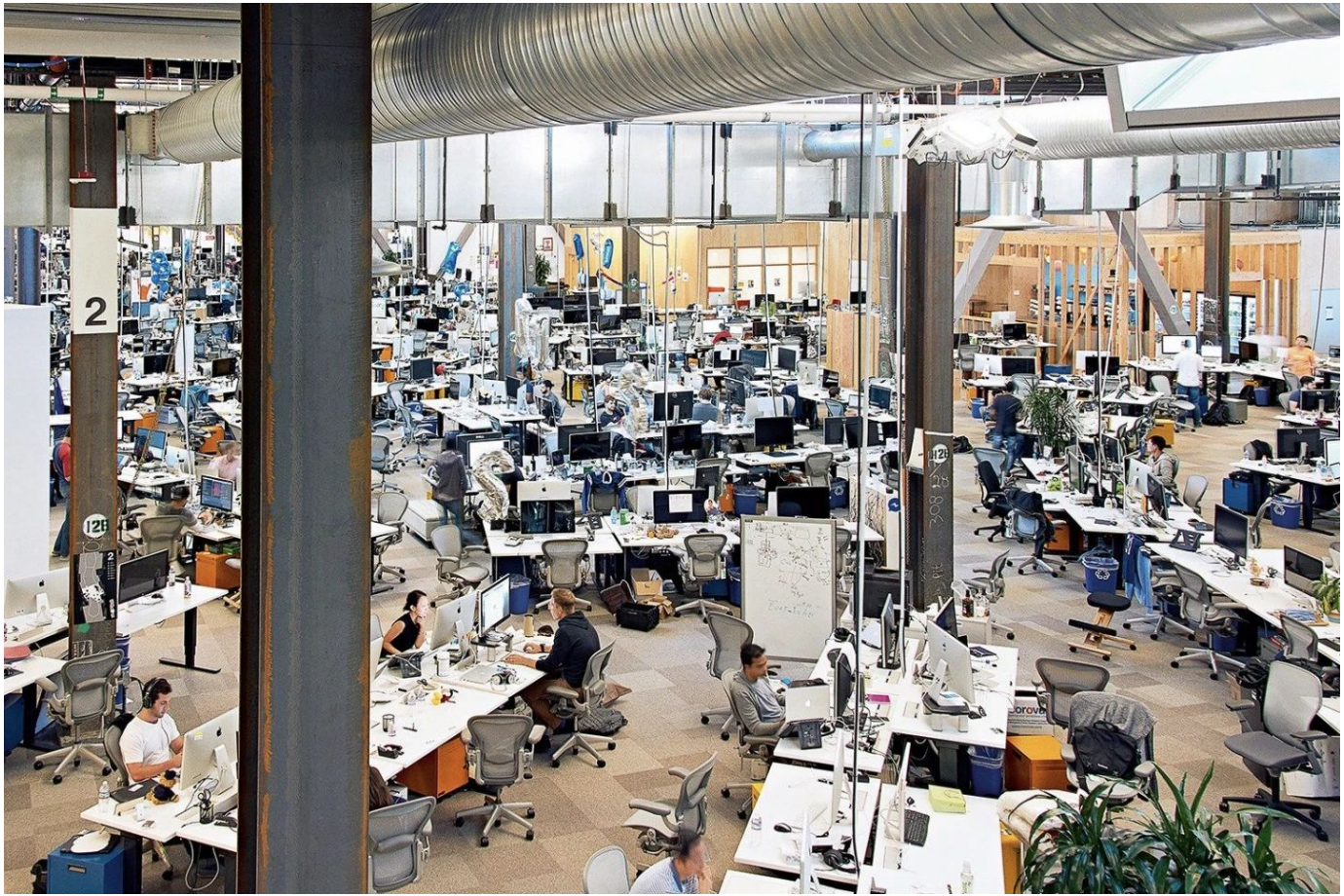
What comes to mind when you think about a software engineer’s day to day work?

Myth: Software engineering is mostly coding by yourself

Collaboration



Introduction to collaboration in software engineering: open office



Introduction to collaboration in software engineering: open office

Advantages

- Efficient collaboration
- Spontaneous conversations

Disadvantages

- Only works in real time
- Remote work

Today's topics

- Agile framework at a larger scale
- Git to collaborate with teammates

Agile Development

Lecture 31, CS61B Spring 2025

Collaborative Project

Name

- 61Belly

Purpose

- Rating foods and sharing culinary opinions

Key Features

- User reviews
- Numerical rating system
- Photo uploads
- Social sharing

Talking to Customers

Why Customer Input Matters

- Ensures product-market fit
- Identifies real user needs and pain points
- Reduces risk of developing unwanted features

Questions

- What features would you find most useful in a food rating app?
- How often do you use food rating apps, and why?
- What frustrates you about existing food rating platforms?

Findings

Top user suggestions

- Easy registration
- Customizable profile
- Reviews and ratings system
- High-quality photos
- Search and filter options for the menu
- Social sharing capabilities

Any class suggestions?

Next Steps

Given our findings, now what?

- How do we plan building out our app?
- How do we adapt to changing customer needs?
- How will we work together with limited time, how about asynchronously?

Agile Software Development Review

“Iterative approach to delivering a project, which focuses on continuous releases that incorporate customer feedback” (Atlassian)

Set of values that guide software development processes through 4 principles:

Agile Software Development Review

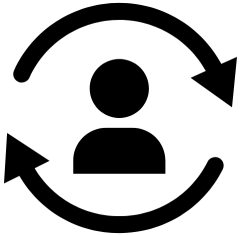
“Iterative approach to delivering a project, which focuses on continuous releases that incorporate customer feedback” (Atlassian)

Set of values that guide software development processes through 4 principles:

Individuals and interactions

Working software

Customer collaboration



Responding to change

Agile Development

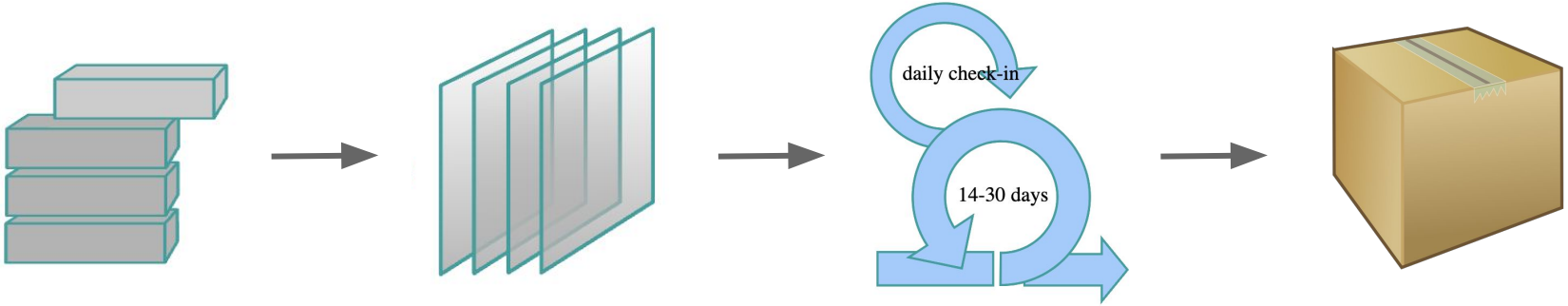
Scrum is an Agile framework for managing work

Product Backlog
List of tasks from the product roadmap, written as user stories

Sprint Backlog
Project broken into pieces called “sprints”

Sprint
1-4 weeks to complete an iteration, with a daily stand-up

Working increment of the product



Applying Agile to 61Belly

Product Backlog

1. Easy registration
2. Reviews and ratings system
3. Search and filter options for menu
4. Social sharing capabilities
5. Customizable profile
6. High-quality photos
7. Edit reviews after they're posted
8. Create shared ratings with friends
9. Push notifications

Sprint 1: 4/11 → 4/25

Sprint 2: 4/25 → 5/1

Sprint 3: 5/1 → 5/15

Applying Agile to 61Belly

Product Backlog

1. Search and filter options for menu
2. Social sharing capabilities
3. Customizable profile
4. High-quality photos
5. Edit reviews after they're posted
6. Create shared ratings with friends
7. Push notifications

Sprint 1: 4/11 → 4/25

- Easy registration
- Reviews and ratings system

Sprint 2: 4/25 → 5/1

Sprint 3: 5/1 → 5/15

Applying Agile to 61Belly

Product Backlog

1. Customizable profile
2. High-quality photos
3. Edit reviews after they're posted
4. Create shared ratings with friends
5. Push notifications

Sprint 1: 4/11 → 4/25

- Easy registration
- Reviews and ratings system

Sprint 2: 4/25 → 5/1

- Search and filter options for menu
- Social sharing capabilities

Sprint 3: 5/1 → 5/15

Applying Agile to 61Belly

Product Backlog

1. Edit reviews after they're posted
2. Create shared ratings with friends
3. Push notifications

Sprint 1: 4/11 → 4/25

- Easy registration
- Reviews and ratings system

Sprint 2: 4/25 → 5/1

- Search and filter options for menu
- Social sharing capabilities

Sprint 3: 5/1 → 5/15

- Customizable profile
- High-quality photos

Agile Development Tools

What tools do software engineers use to track and put these plans into action?

Jira is a software application developed by Atlassian for issue tracking and project management

- Sprint planning and tracking
- Customizable workflows and boards
- Backlog management
- Reporting and dashboards

Let’s create a Jira project for 61Belly!

Jira Demo

Projects / 61Belly

Backlog

Search | SK | Epic | Insights | View settings

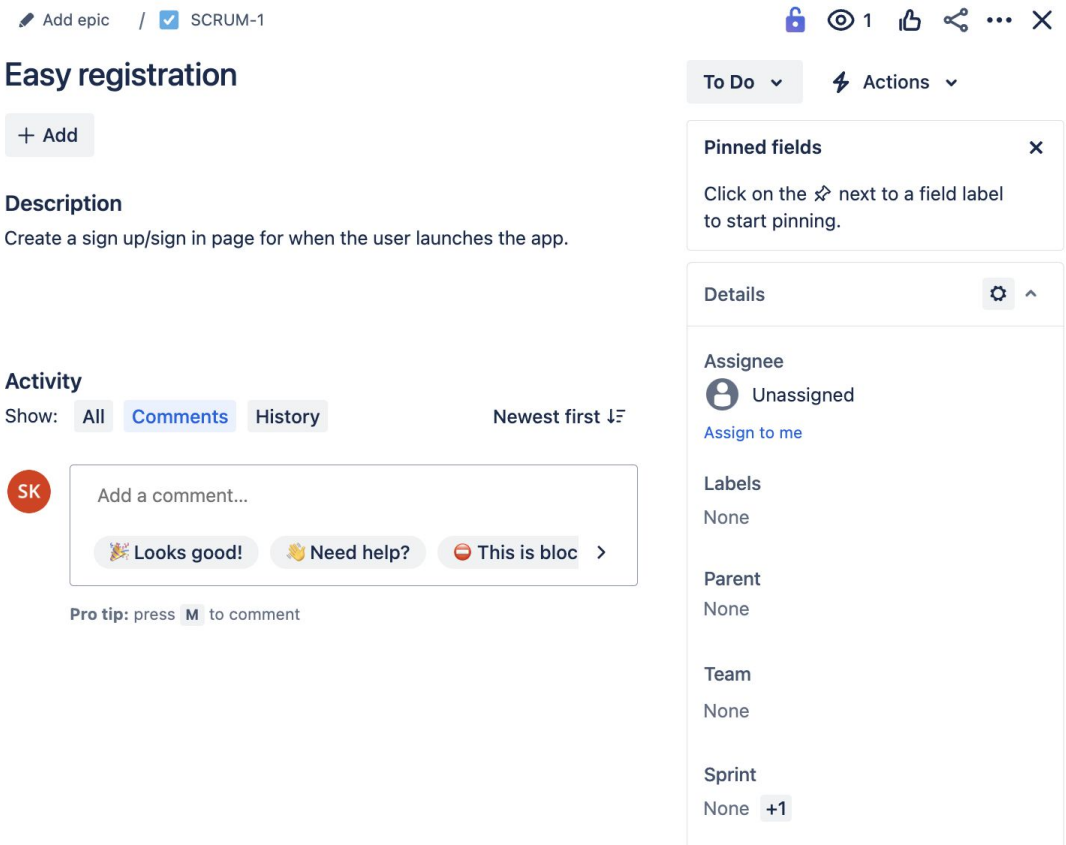
Backlog (8 issues) | 0 | 0 | 0 | Create sprint

SCRUM-1	Make signup and login pages	TO DO	-
SCRUM-4	Ratings not displaying correctly	TO DO	-
SCRUM-3	Search and filter options for menu	TO DO	-
SCRUM-5	Social sharing capabilities	TO DO	-
SCRUM-6	Customizable profile	TO DO	-
SCRUM-7	High-quality photos	TO DO	-
SCRUM-8	Edit reviews after they're posted	TO DO	-
SCRUM-9	Create shared ratings with friends	TO DO	-

+ Create issue

Jira Demo

A Jira **ticket** is a task, bug, or request within a project



Jira Demo

Backlog

Search | +2 | Epic | Insights | View settings

Sprint 1 18 Nov – 29 Nov (2 issues) — 0 0 0 — Start sprint

SCRUM-1	Easy registration	TO DO	- SK
SCRUM-4	Reviews and ratings system	TO DO	- KM

+ Create issue

2 issues | Estimate: 0

Backlog (6 issues) — 0 0 0 — Create sprint

SCRUM-3	Search and filter options for menu	TO DO	-
SCRUM-5	Social sharing capabilities	TO DO	-
SCRUM-6	Customizable profile	TO DO	-
SCRUM-7	High-quality photos	TO DO	-
SCRUM-8	Edit reviews after they're posted	TO DO	-
SCRUM-9	Create shared ratings with friends	TO DO	-

+ Create issue

Jira Demo

Projects / 61Belly

Sprint 1

13 days | Complete sprint

Search | SK KM

GROUP BY None | Insights | View settings

TO DO 2	IN PROGRESS	DONE
Easy registration SCRUM-1 SK		
Reviews and ratings system SCRUM-4 KM		
+ Create issue		

Jira Demo

Projects / 61Belly

Sprint 1

13 days | Complete sprint

Search

SK KM

GROUP BY None | Insights | View settings

TO DO	IN PROGRESS	DONE 2
+ Create issue		Reviews and ratings system ~~SCRUM-4~~ KM
		Easy registration ~~SCRUM-1~~ SK

Jira Demo

Projects / 61Belly

Backlog

Search | SK | Epic | Insights | View settings

Sprint 2 Add dates (2 issues) 0 0 0 Start sprint

SCRUM-3	Search and filter options for menu	TO DO	-
SCRUM-5	Social sharing capabilities	TO DO	-

+ Create issue

2 issues | Estimate: 0

Backlog (4 issues) 0 0 0 Create sprint

SCRUM-6	Customizable profile	TO DO	-
SCRUM-7	High-quality photos	TO DO	-
SCRUM-8	Edit reviews after they're posted	TO DO	-
SCRUM-9	Create shared ratings with friends	TO DO	-

Add a forgot password button and feature

Jira Demo

Projects / 61Belly

Backlog

Search | +2 | Epic | Insights | View settings

Sprint 2 Add dates (3 issues) 0 0 0 Start sprint

SCRUM-10	Add a forgot password button and feature	TO DO	SK
SCRUM-3	Search and filter options for menu	TO DO	- KM
SCRUM-5	Social sharing capabilities	TO DO	- KM

+ Create issue

3 issues | Estimate: 0

Backlog (4 issues) 0 0 0 Create sprint

SCRUM-6	Customizable profile	TO DO	-
SCRUM-7	High-quality photos	TO DO	-
SCRUM-8	Edit reviews after they're posted	TO DO	-
SCRUM-9	Create shared ratings with friends	TO DO	-

+ Create issue

Jira Demo

Projects / 61Belly

Sprint 2

13 days | Complete sprint

Search

GROUP BY None | Insights | View settings

TO DO	IN PROGRESS 3	DONE
+ Create issue	Add a forgot password button and feature SCRUM-10 SK	
	Search and filter options for menu SCRUM-3 KM	
	Social sharing capabilities SCRUM-5 KM	

Developing features and collaboration

Why do we use Jira? It helps communicate:

- **Who** will be doing which task
- **When** the tasks can be completed by
- **How** to adjust timelines to incorporate customer feedback

Developing features and collaboration

Why do we use Jira? It helps communicate:

- **Who** will be doing which task
- **When** the tasks can be completed by
- **How** to adjust timelines to incorporate customer feedback

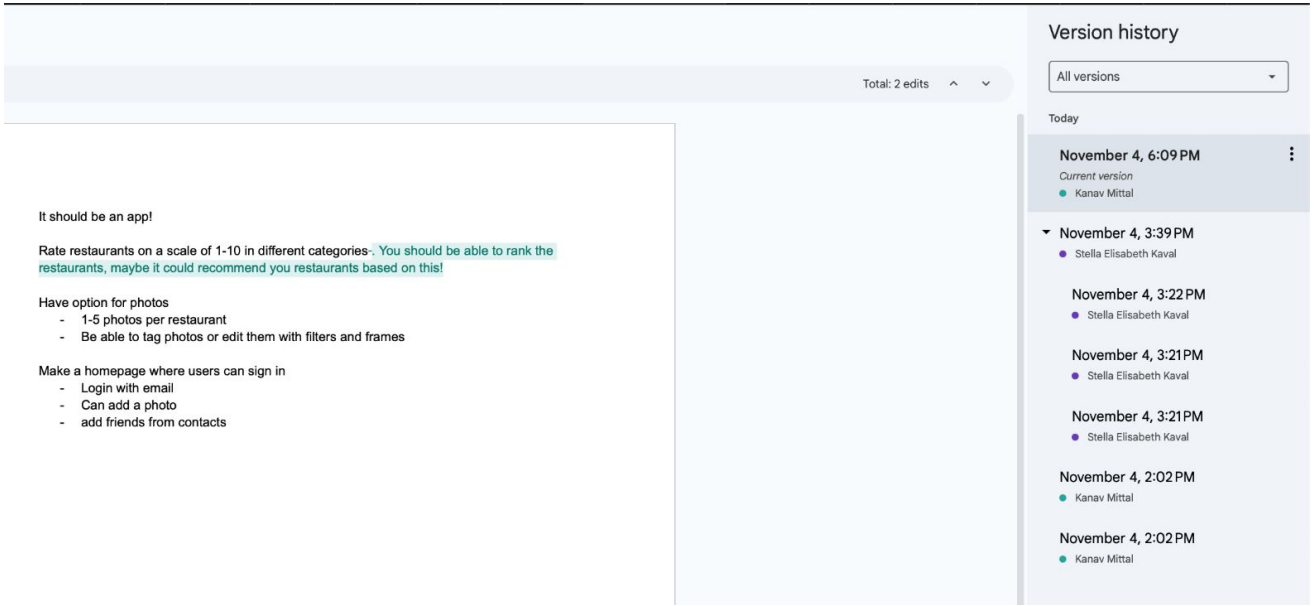
We talked about completing these tickets in the backlog on a high level, but how do we actually **code** them?

- Collaborate with developers (co-workers, future ones, yourself)
- See changes over time
- Revert to previous versions

Solution 1: Google Docs

Google Docs is great at working together on documents

- Easy-to-use
- Allows for real time collaboration
- Maintains a version history of documents so you can go back
- **So… what's the problem?**



Git is more powerful!

Advantages

- Git allows you to choose files and when to take a snapshot of your files to include in your version history
- Git allows you to collaborate on different variations of your project without interference from the others
  - e.g. partners working on different Project 3B ambition features
  - You don’t want errors from one variation affecting the other variation

Disadvantages

- Git is a more specialized tool for software development and not as helpful for non-developers

Review: Git Commands

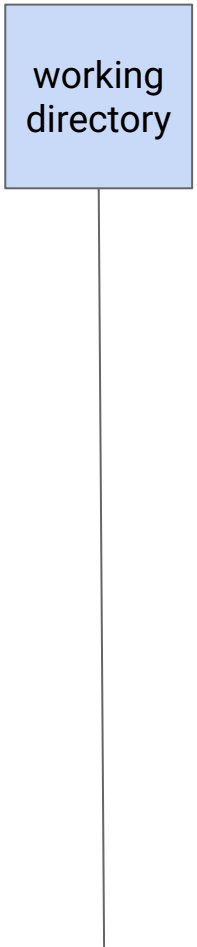
Lecture 31, CS61B Spring 2025

Git as a version control system

- Committing is like a stronger type of saving
  - Saves a snapshot of your files
  - You can go back to this snapshot if you would like

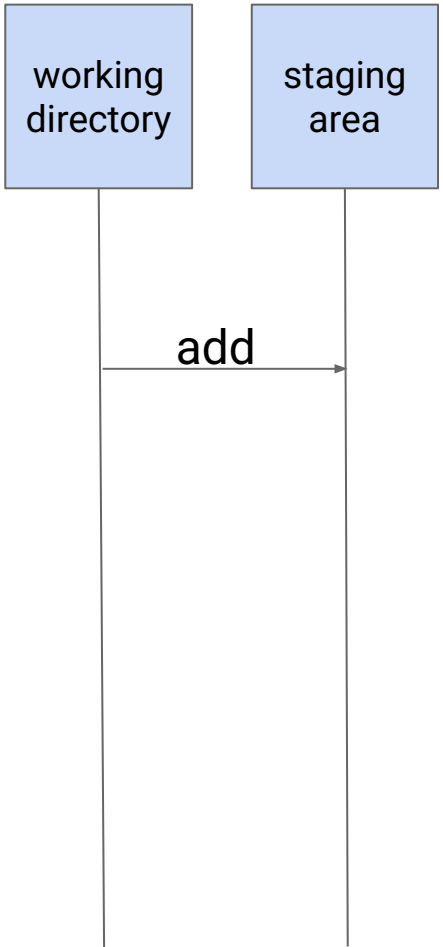
Git as a version control system

- Committing is like a stronger type of saving
  - Saves a snapshot of your files
  - You can go back to this snapshot if you would like



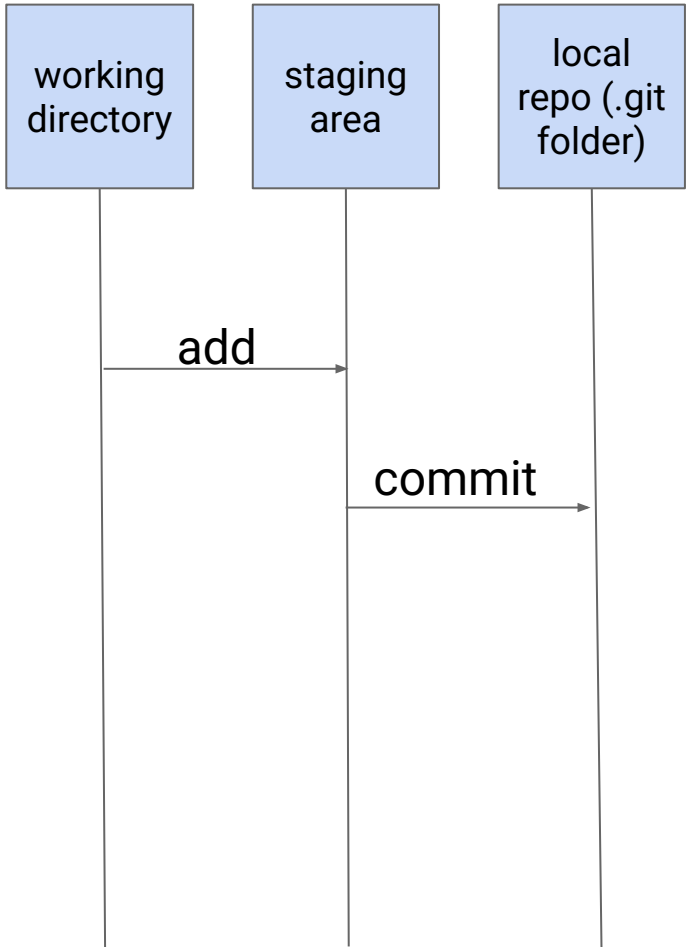
Git as a version control system

- Committing is like a stronger type of saving
  - Saves a snapshot of your files
  - You can go back to this snapshot if you would like
- `git add` tells Git which files to save (i.e., files in staging area)



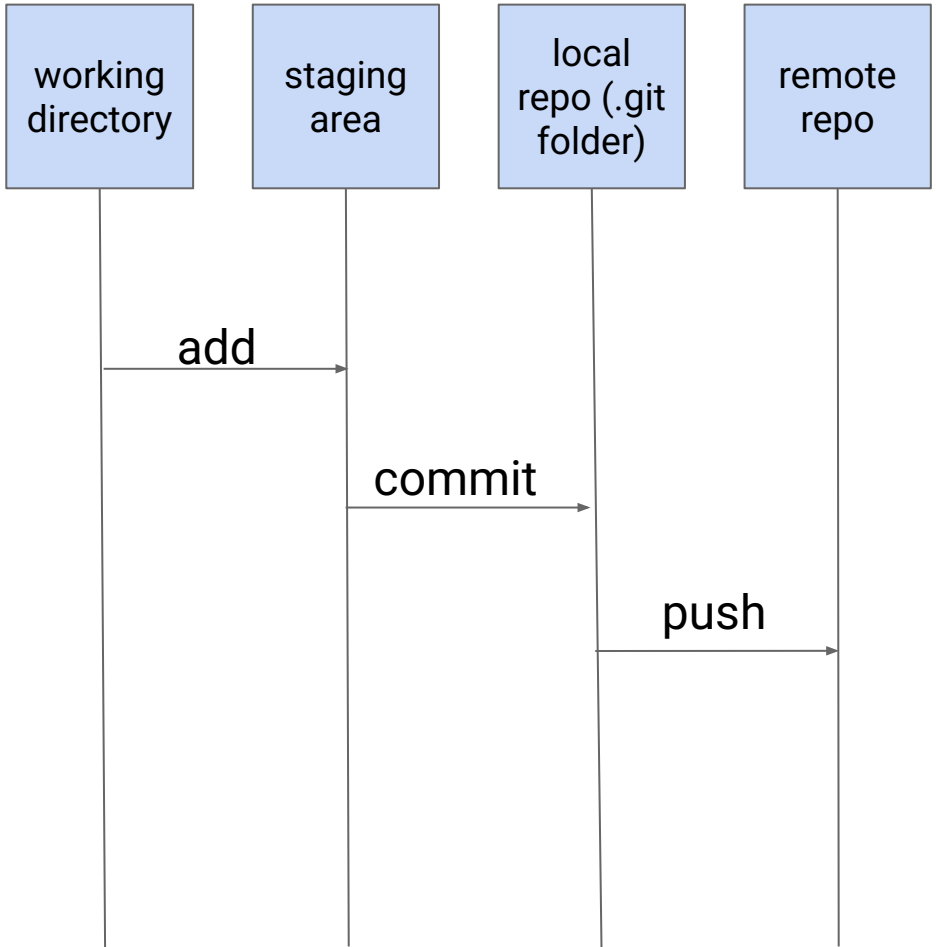
Git as a version control system

- Committing is like a stronger type of saving
  - Saves a snapshot of your files
  - You can go back to this snapshot if you would like
- `git add` tells Git which files to save (i.e., files in staging area)
- `git commit` executes the save
  - Adds commit to version history in your local repository



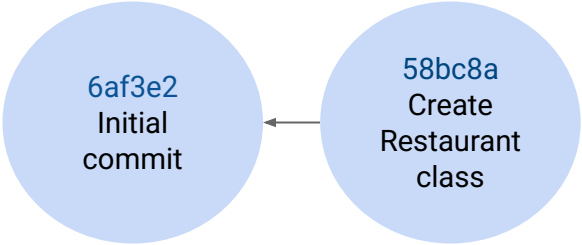
Git as a version control system

- Committing is like a stronger type of saving
  - Saves a snapshot of your files
  - You can go back to this snapshot if you would like
- `git add` tells Git which files to save (i.e., files in staging area)
- `git commit` executes the save
  - Adds commit to version history in your local repository
- `git push` sends your updated version history to a remote repository for backup
  - Remote repository: copy of your code stored on Internet



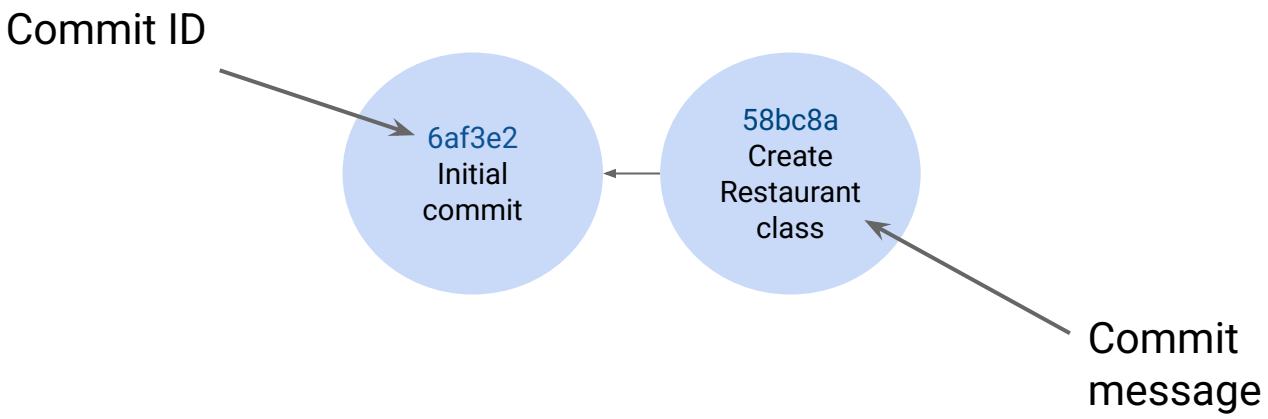
Commit History

- Commits contain pointers to the commits that came before them



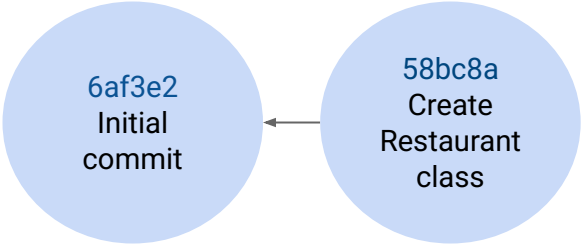
Commit History

- Commits contain pointers to the commits that came before them



Commit History

- Commits contain pointers to the commits that came before them
  - Commit history forms a linked list



Commit History

- Commits contain pointers to the commits that came before them
  - Commit history forms a linked list
  - When we make a commit, we add a new node to our linked list

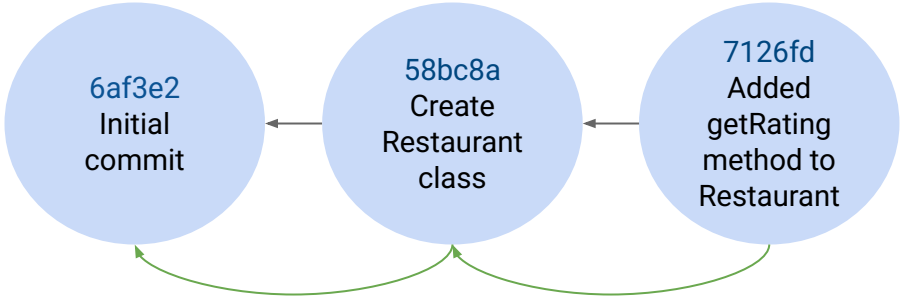
Commit History

- Commits contain pointers to the commits that came before them
  - Commit history forms a linked list
  - When we make a commit, we add a new node to our linked list

- `git log` gives you the commit history

Commit History

- Commits contain pointers to the commits that came before them
  - Commit history forms a linked list
  - When we make a commit, we add a new node to our linked list



- `git log` gives you the commit history
  - Starting from current commit, keep iterating through the previous commits until you reach the initial commit

Commit History

- Commits contain pointers to the commits that came before them
  - Commit history forms a linked list
  - When we make a commit, we add a new node to our linked list

- `git log` gives you the commit history
  - Starting from current commit, keep iterating through the previous commits until you reach the initial commit
  - What are all the changes that occurred to get the current code?

Going back to a previous version of your code

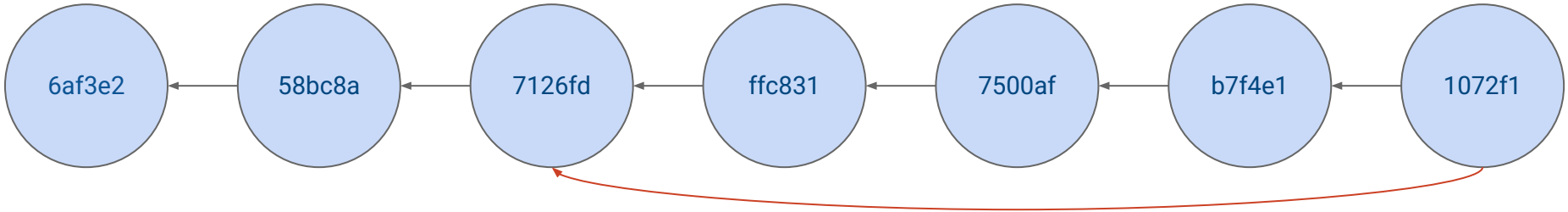
- Why do we care about storing a version history?
  - What if you break something in your code and want to go back?
  - Or what if you want to build off a past checkpoint of your code?

Going back to a previous version of your code

- Why do we care about storing a version history?
  - What if you break something in your code and want to go back?
  - Or what if you want to build off a past checkpoint of your code?
- Git allows you to go back to a previous version of your code saved in your commit history
  - See a history and get commit ID using `git log`
  - Inspect past version with `git checkout`
  - Undo changes with `git revert` or `git reset`

Going back to a previous version of your code

- Why do we care about storing a version history?
  - What if you break something in your code and want to go back?
  - Or what if you want to build off a past checkpoint of your code?
- Git allows you to go back to a previous version of your code saved in your commit history
  - See a history and get commit ID using `git log`
  - Inspect past version with `git checkout`
  - Undo changes with `git revert` or `git reset`

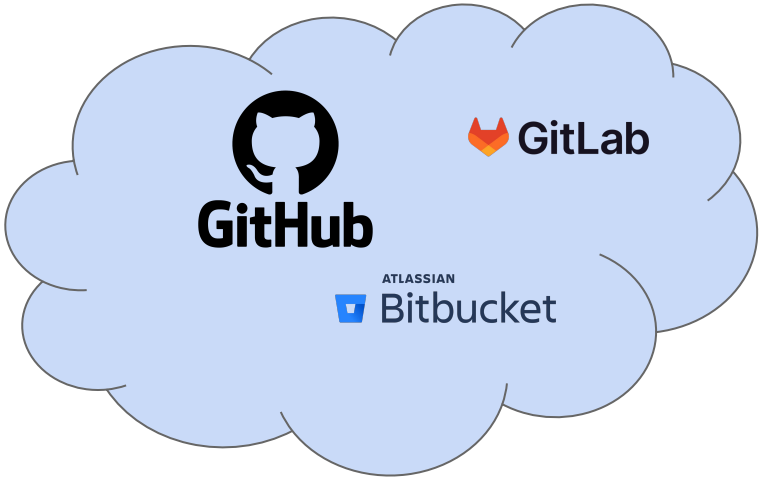


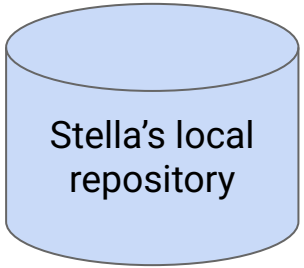
Going back to a previous version of your code

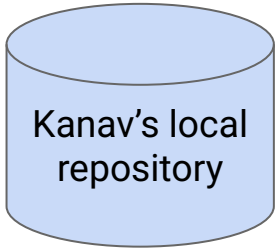
- Why do we care about storing a version history?
  - What if you break something in your code and want to go back?
  - Or what if you want to build off a past checkpoint of your code?
- Git allows you to go back to a previous version of your code saved in your commit history
  - See a history and get commit ID using `git log`
  - Inspect past version with `git checkout`
  - Undo changes with `git revert` or `git reset`
- Tips
  - Write brief but descriptive commit messages
  - Commit frequently!

Git as a tool for collaboration

- Remote repositories can be shared by multiple people

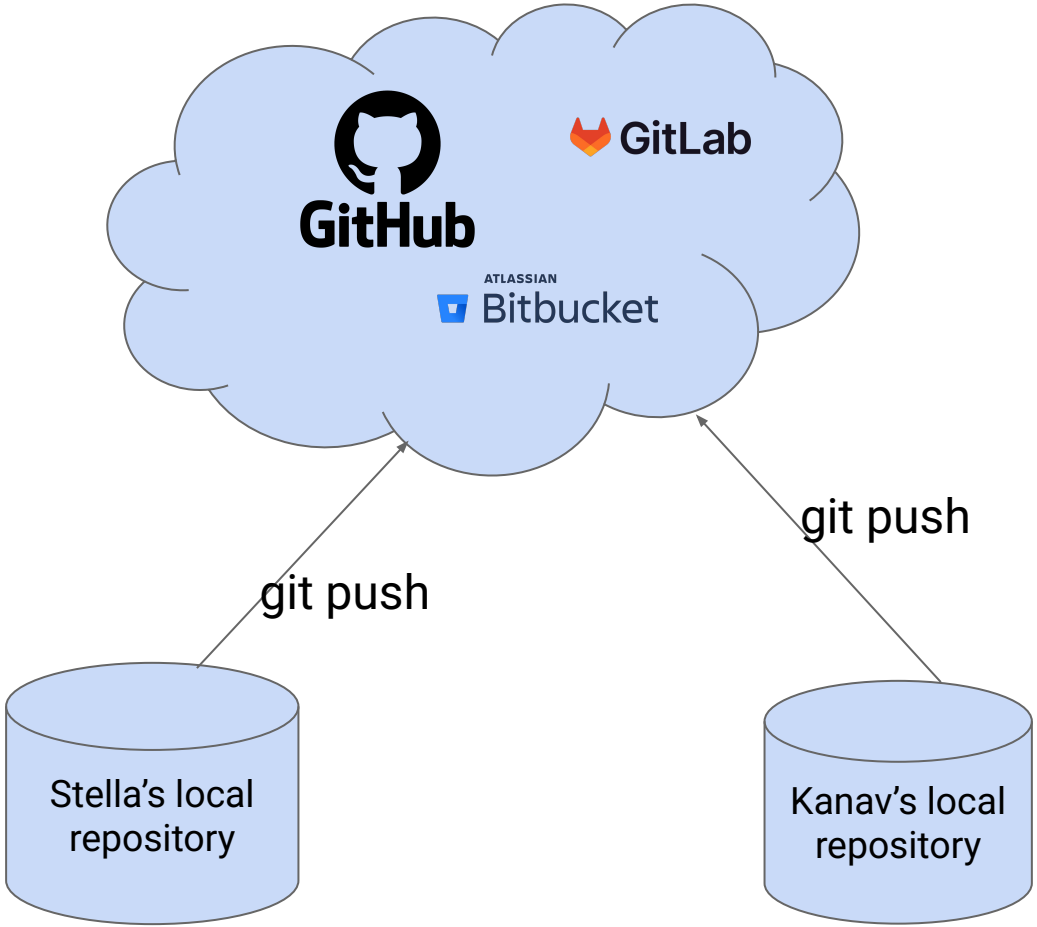






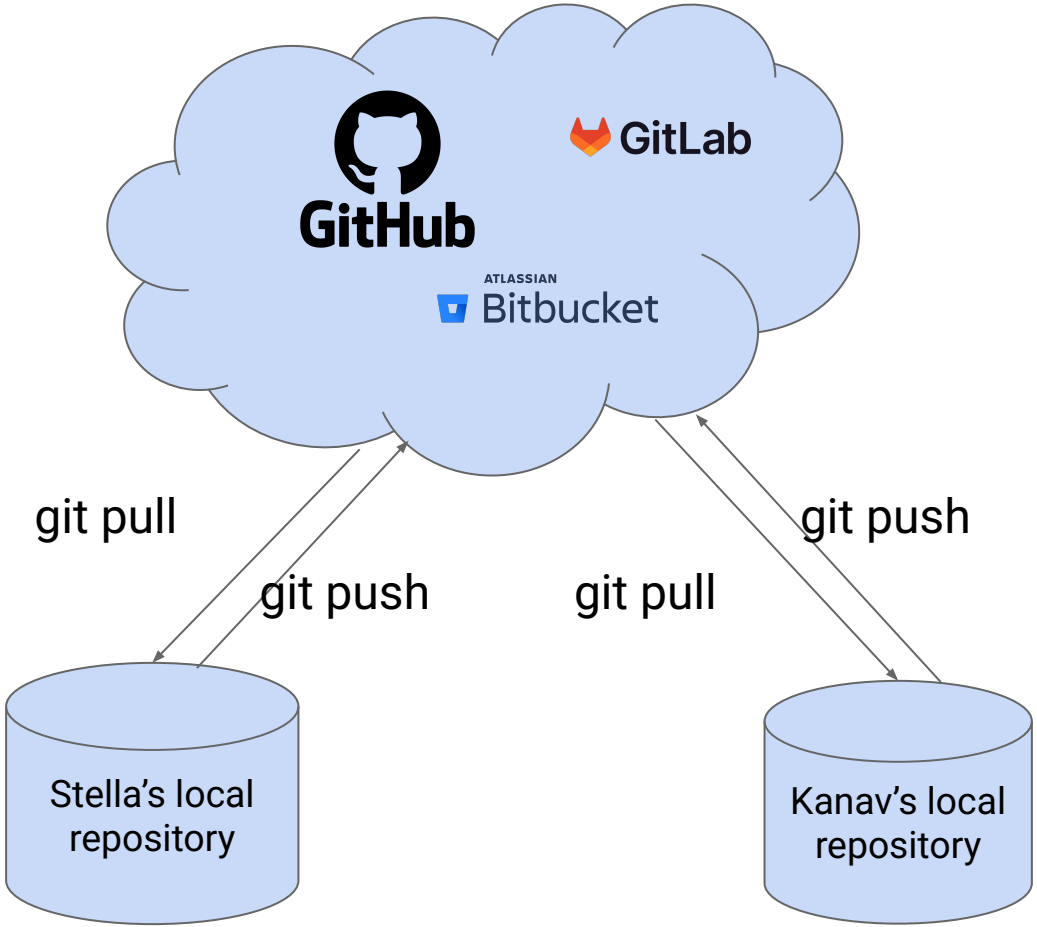
Git as a tool for collaboration

- Remote repositories can be shared by multiple people
- `git push` updates the remote with your local changes



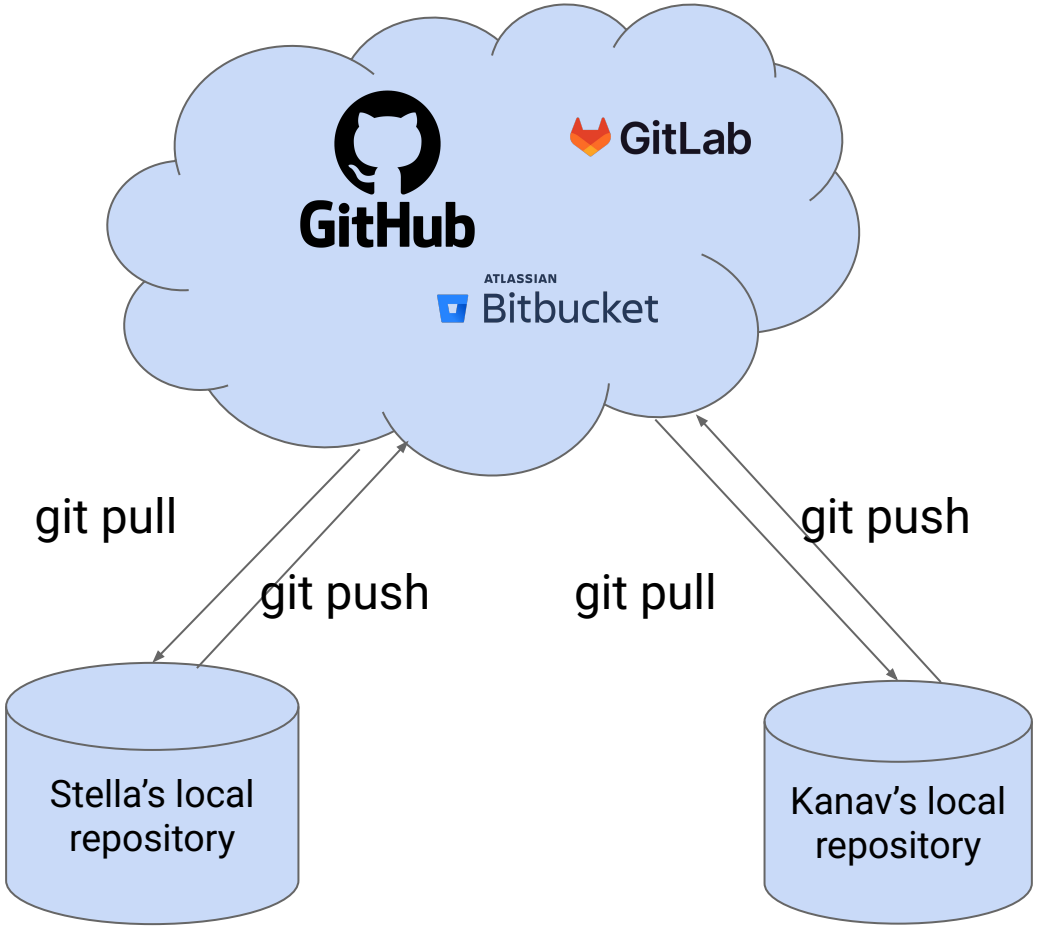
Git as a tool for collaboration

- Remote repositories can be shared by multiple people
- `git push` updates the remote with your local changes
- `git pull` incorporates changes from remote into local



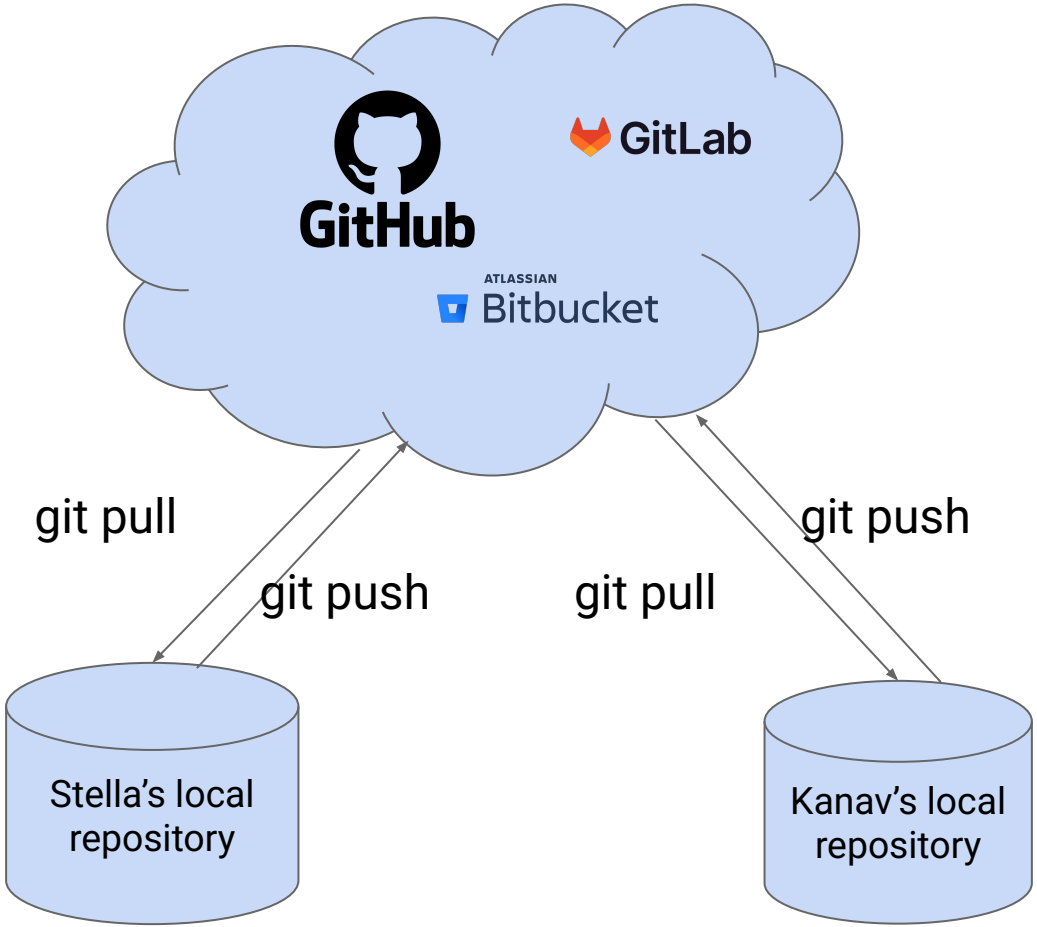
Git as a tool for collaboration

- Remote repositories can be shared by multiple people
- `git push` updates the remote with your local changes
- `git pull` incorporates changes from remote into local
- One developer can push their changes, and the other developer can pull them in to work off them.



Git as a tool for collaboration

- Remote repositories can be shared by multiple people
- `git push` updates the remote with your local changes
- `git pull` incorporates changes from remote into local
- One developer can push their changes, and the other developer can pull them in to work off them.
- Remember to pull before pushing



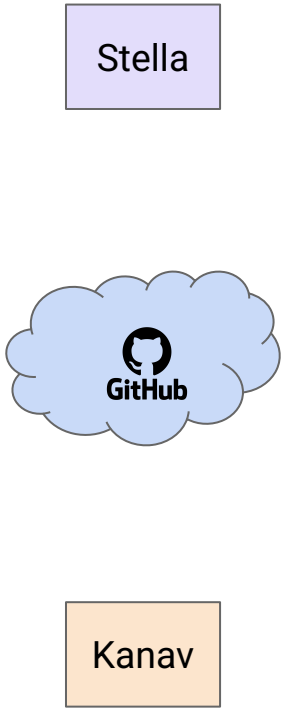
Collaborating on the same commit history

- Remember: commit history forms a linked list-like structure

Collaborating on the same commit history

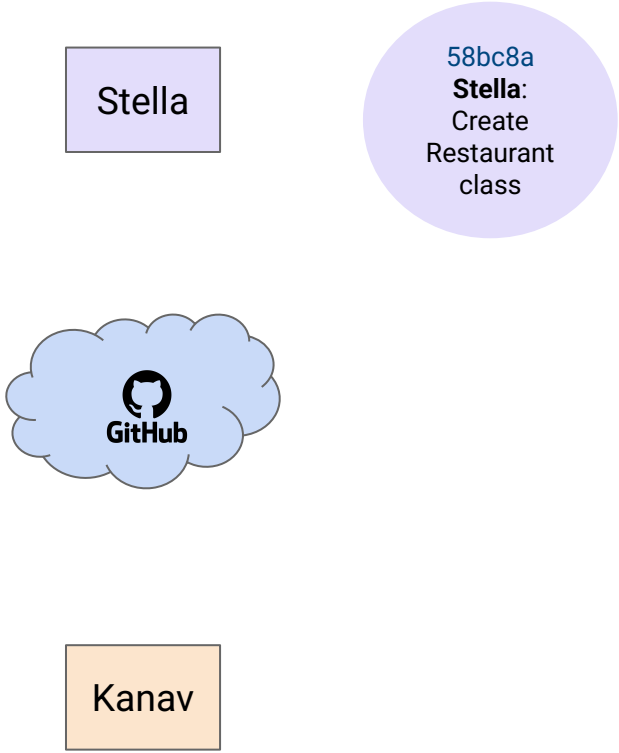
- Remember: commit history forms a linked list-like structure
- `git push` and `git pull` try to copy this linked list of commits back and forth between local and remote repositories
  - Multiple people can work on the same line of development

Collaborating on the same commit history



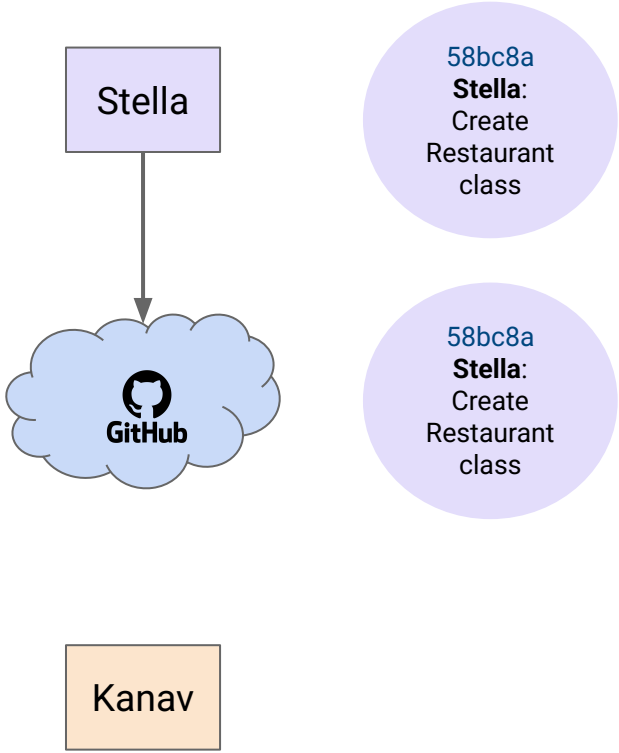
Collaborating on the same commit history

Stella creates a Restaurant class and commits it



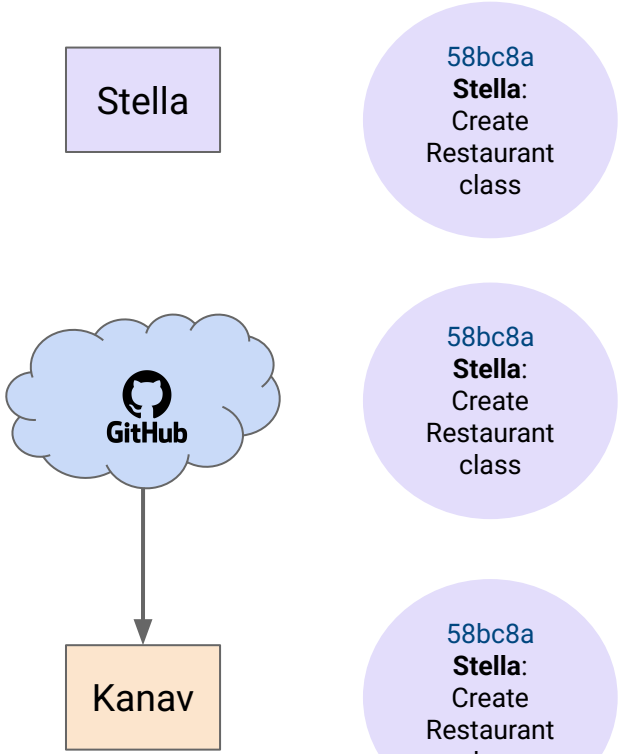
Collaborating on the same commit history

Stella pushes her changes to GitHub so others can access them



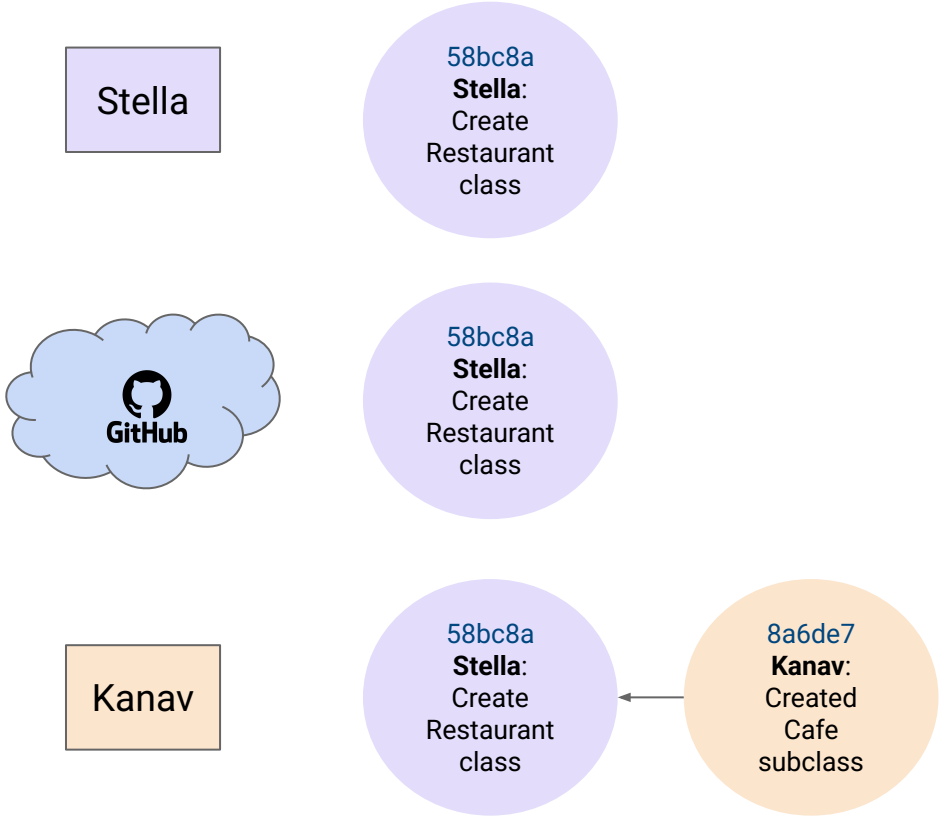
Collaborating on the same commit history

Kanav pulls Stella's changes from GitHub



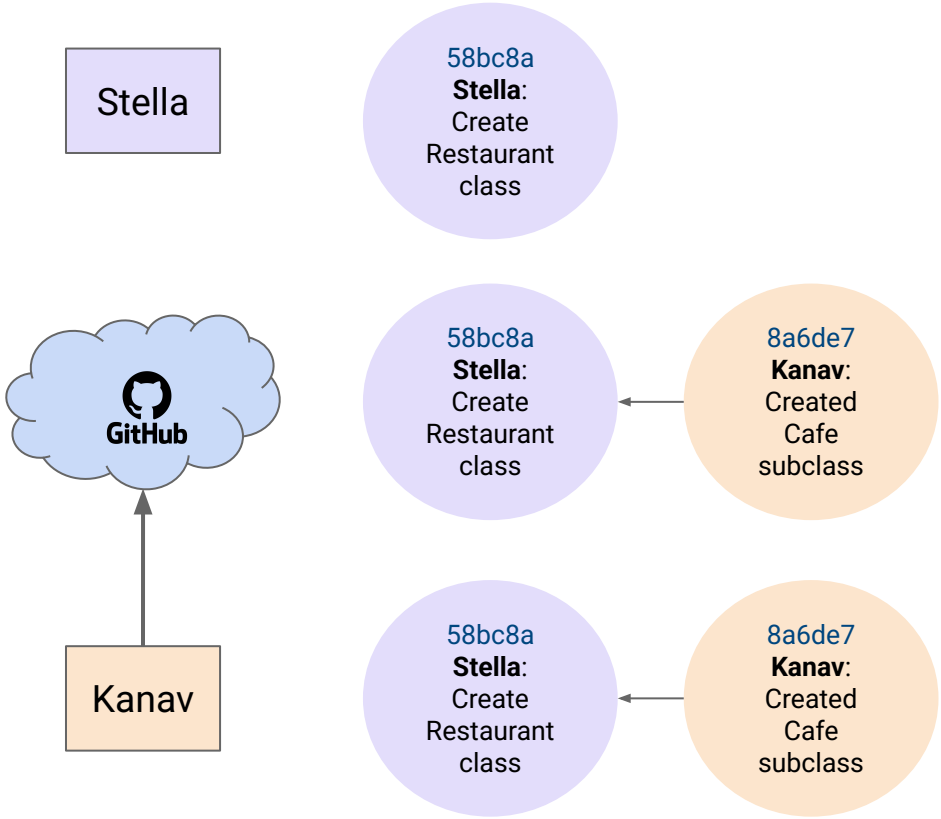
Collaborating on the same commit history

Kanav works on creating a Cafe subclass and commits it



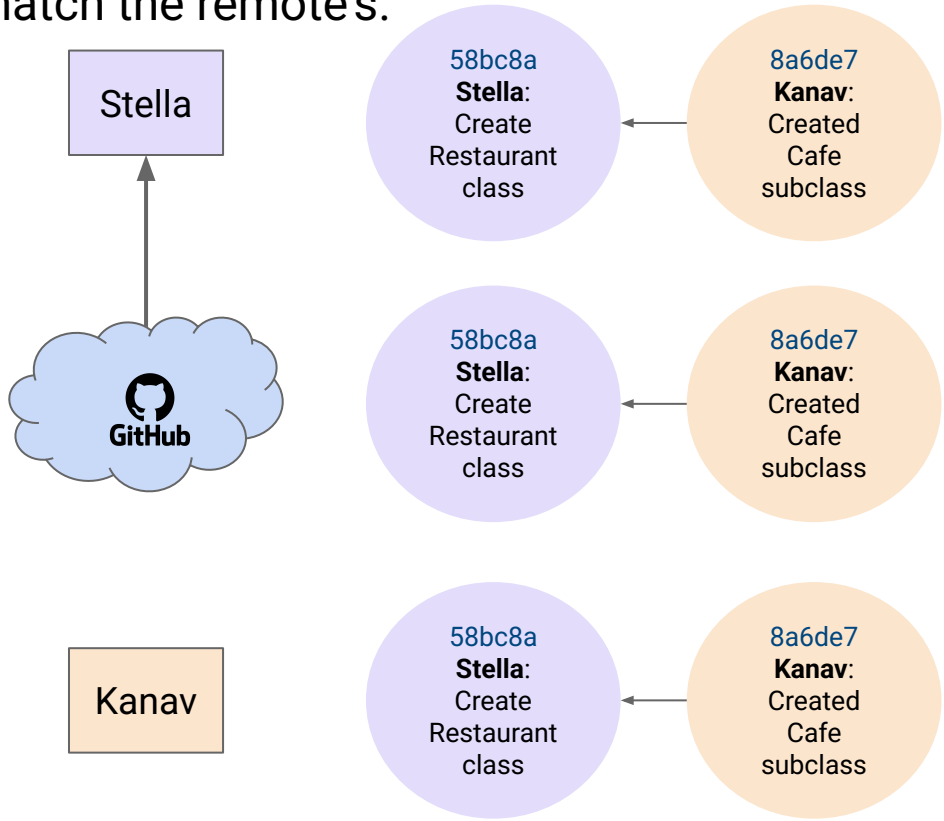
Collaborating on the same commit history

Kanav pushes his commit history to GitHub



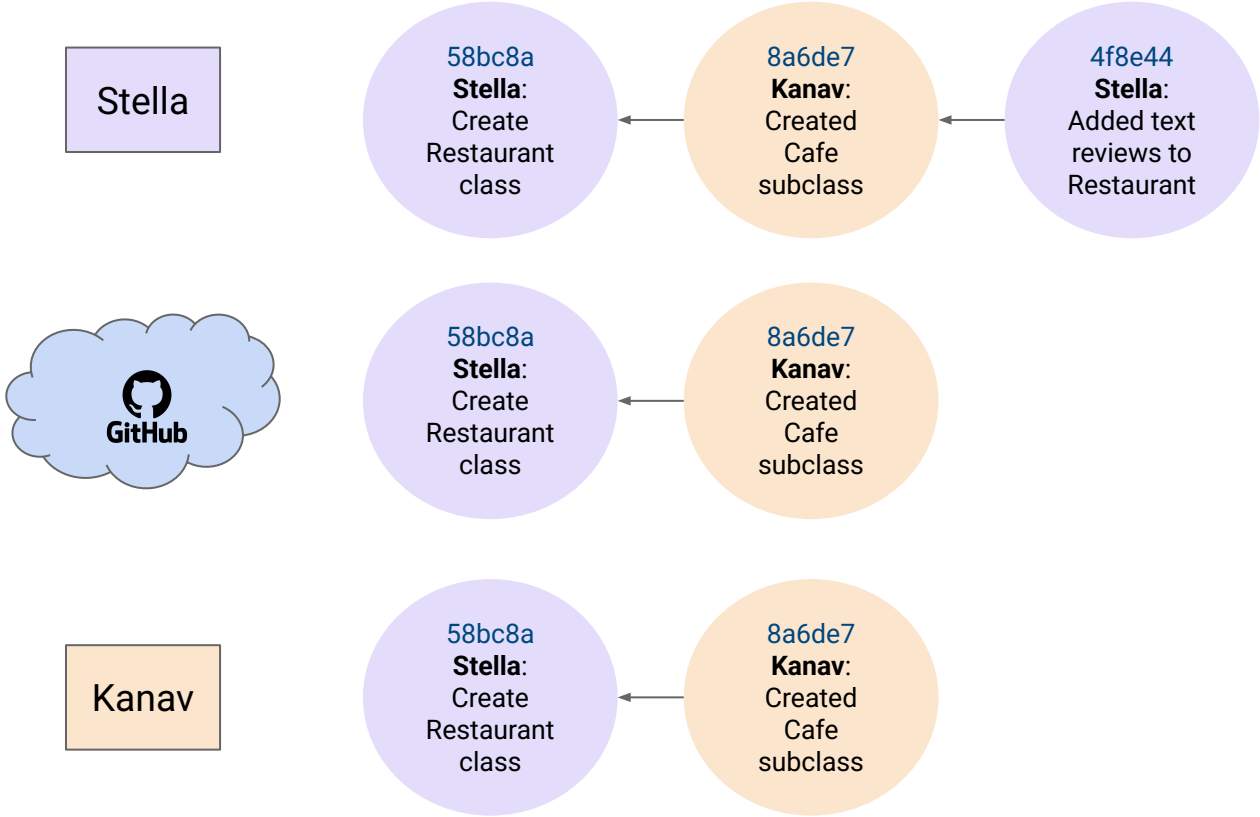
Collaborating on the same commit history

Stella pulls the commit history from GitHub. Her commit history is fast-forwarded to match the remote's.



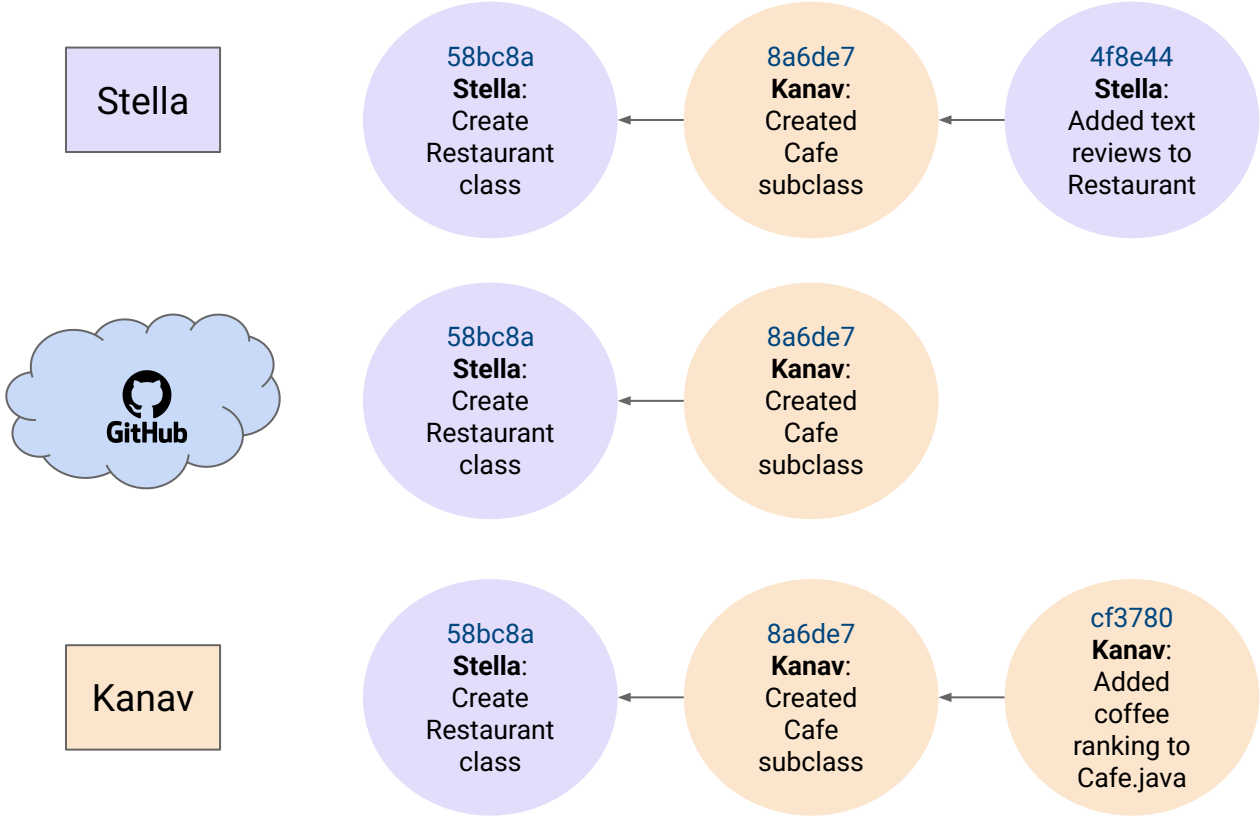
Collaborating on the same commit history

Stella works on adding text reviews to Restaurant.java



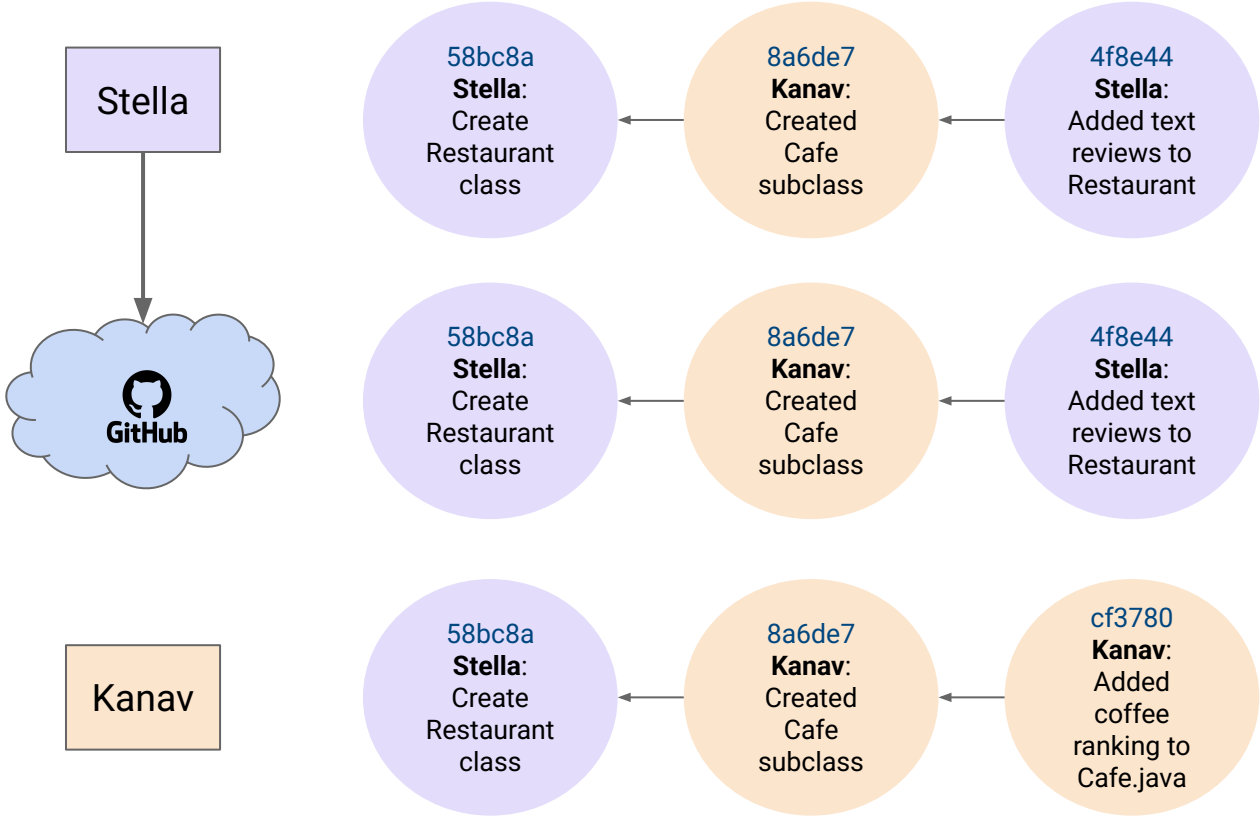
Collaborating on the same commit history

Kanav works on adding a coffee ranking feature to Cafe.java



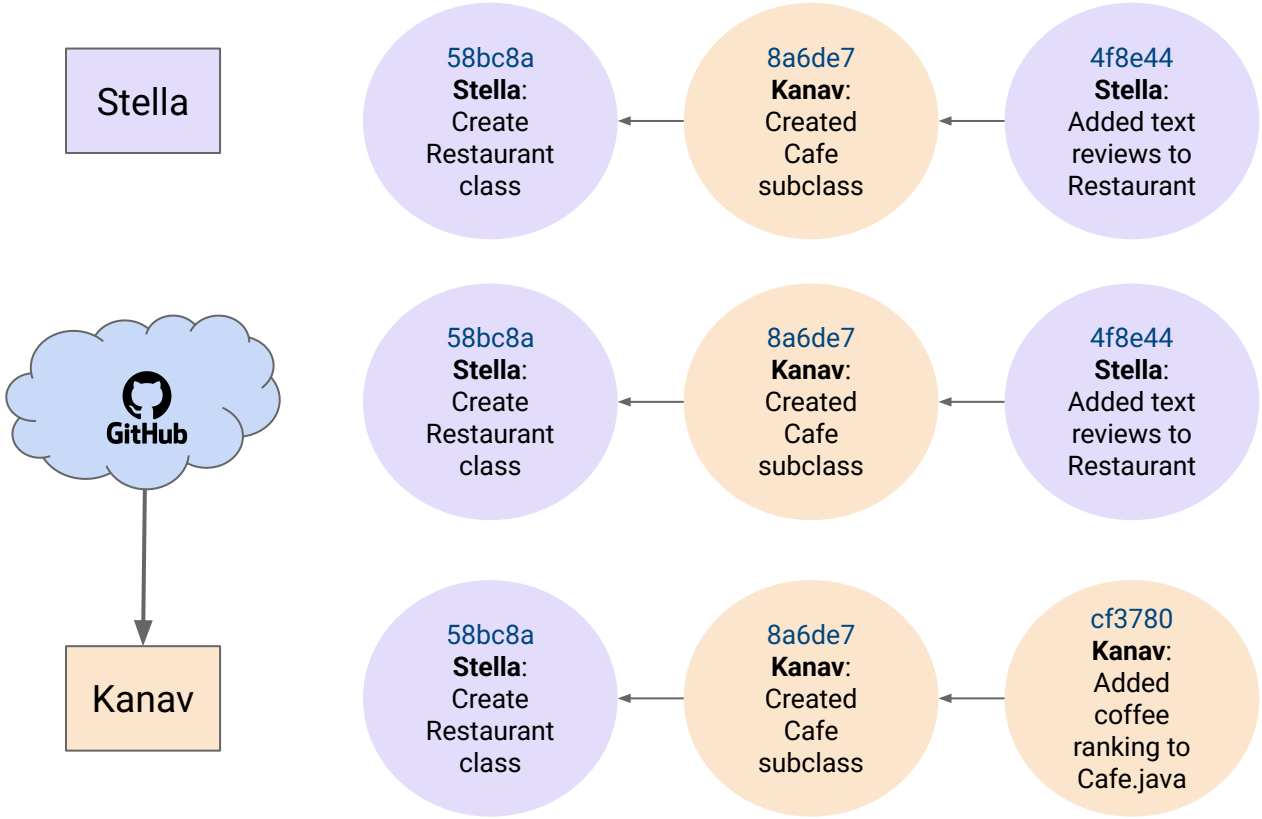
Collaborating on the same commit history

Stella pushes her work to GitHub



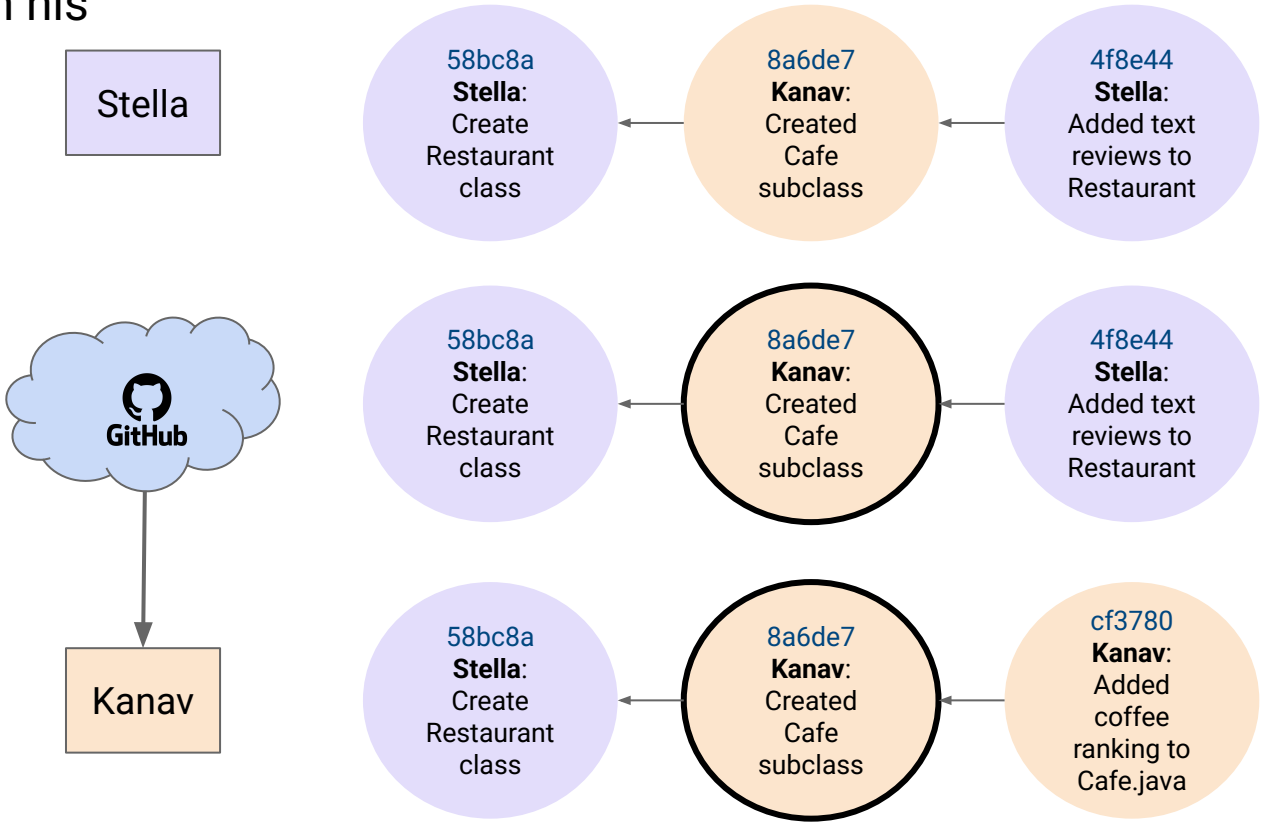
Collaborating on the same commit history

Kanav wants to push to GitHub, but first he tries to pull from GitHub…



Collaborating on the same commit history

Kanav tries to pull from GitHub, but he now finds that the remote's history diverges from his



Merging

- If the remote and local commit histories at one point shared a common ancestor commit but have since then gone their own way, we say that the two histories **diverge** from each other

Merging

- If the remote and local commit histories at one point shared a common ancestor commit but have since then gone their own way, we say that the two histories **diverge** from each other
- Solution: **merge** the remote and local commit history together
  - We add a merge commit to signify the merge

Merging

- If the remote and local commit histories at one point shared a common ancestor commit but have since then gone their own way, we say that the two histories **diverge** from each other
- Solution: **merge** the remote and local commit history together
  - We add a merge commit to signify the merge
- The merge finds the Closest Common Ancestor (point of divergence) of the two histories
  - Combines the changes made from the closest common ancestor to each of the latest commits

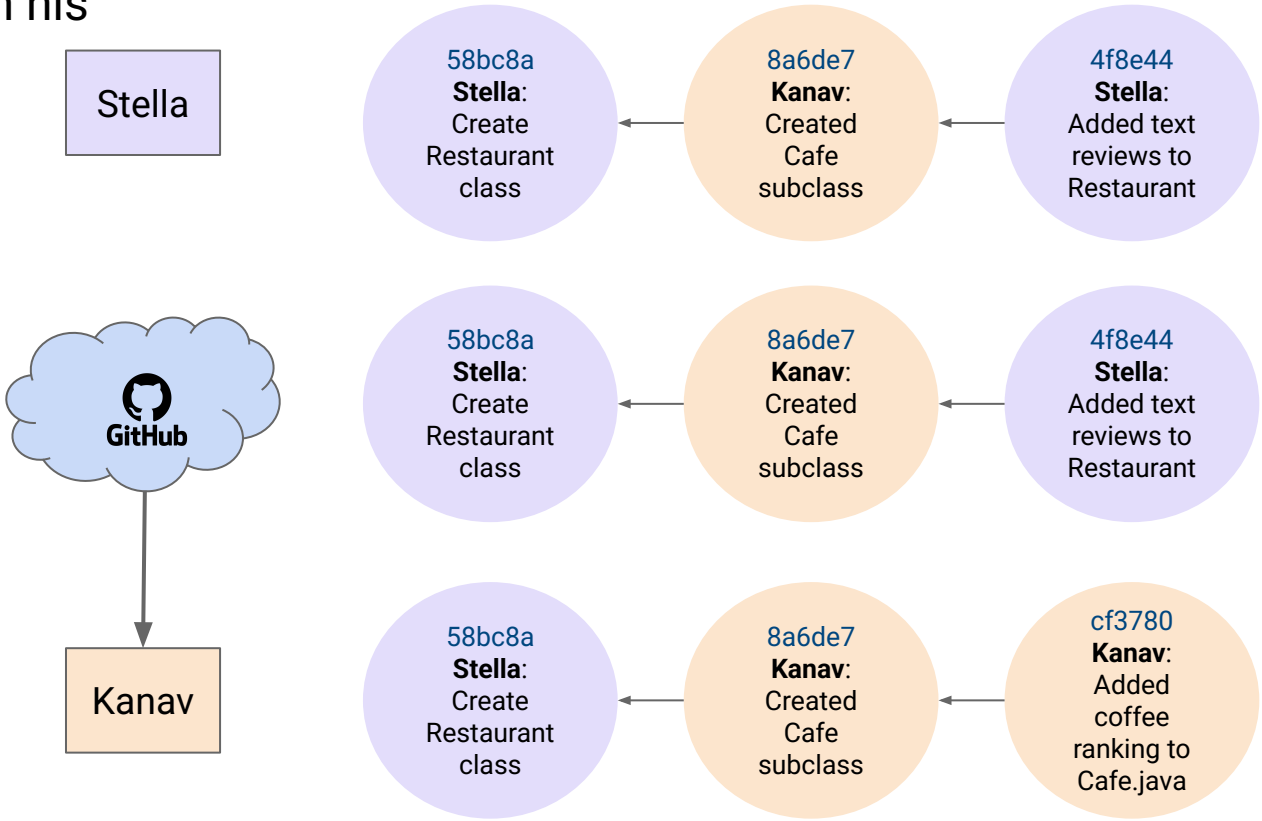
Merging

If you've seen a message like this, this is executing a merge

```
Merge branch 'main' of github.com:kanavmittal314/61belly
# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
~
~
~
~
~
~
~
~
```

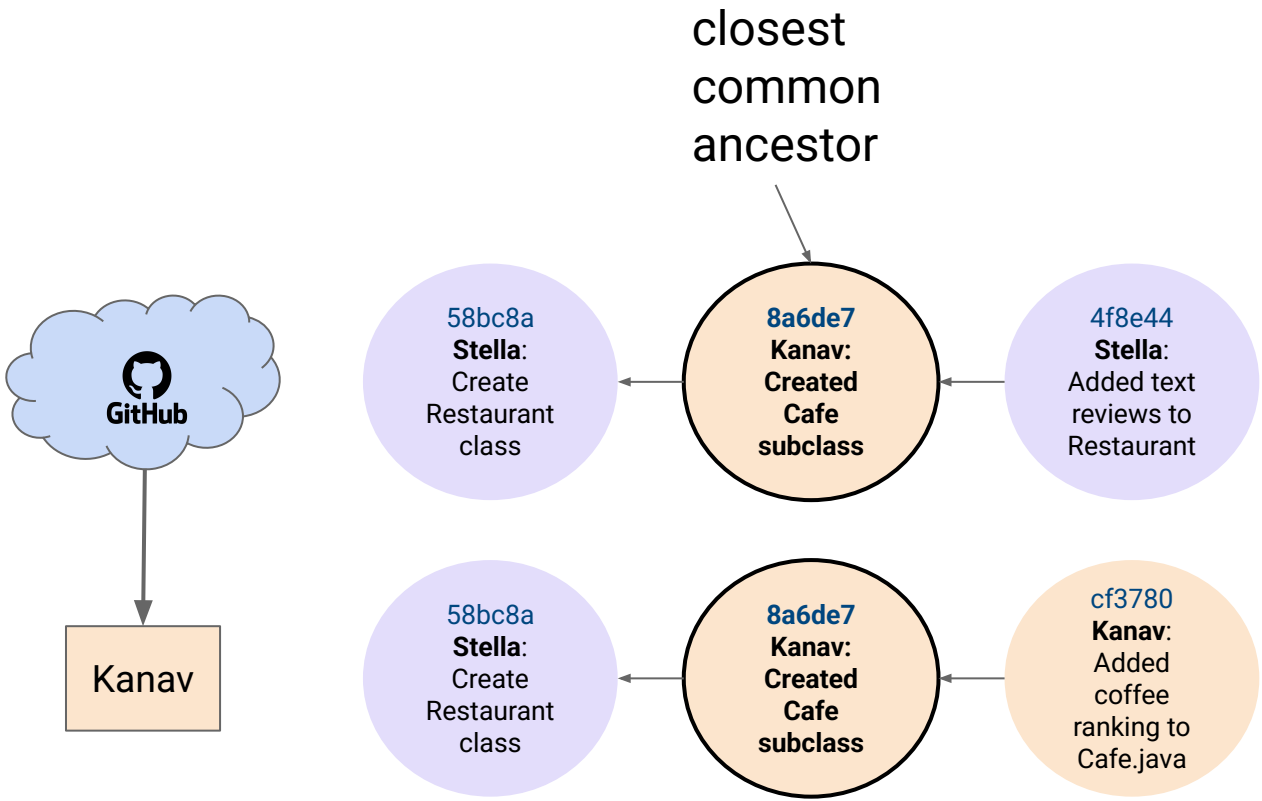
Collaborating on the same commit history

Kanav tries to pull from GitHub, but he now finds that the remote's history diverges from his



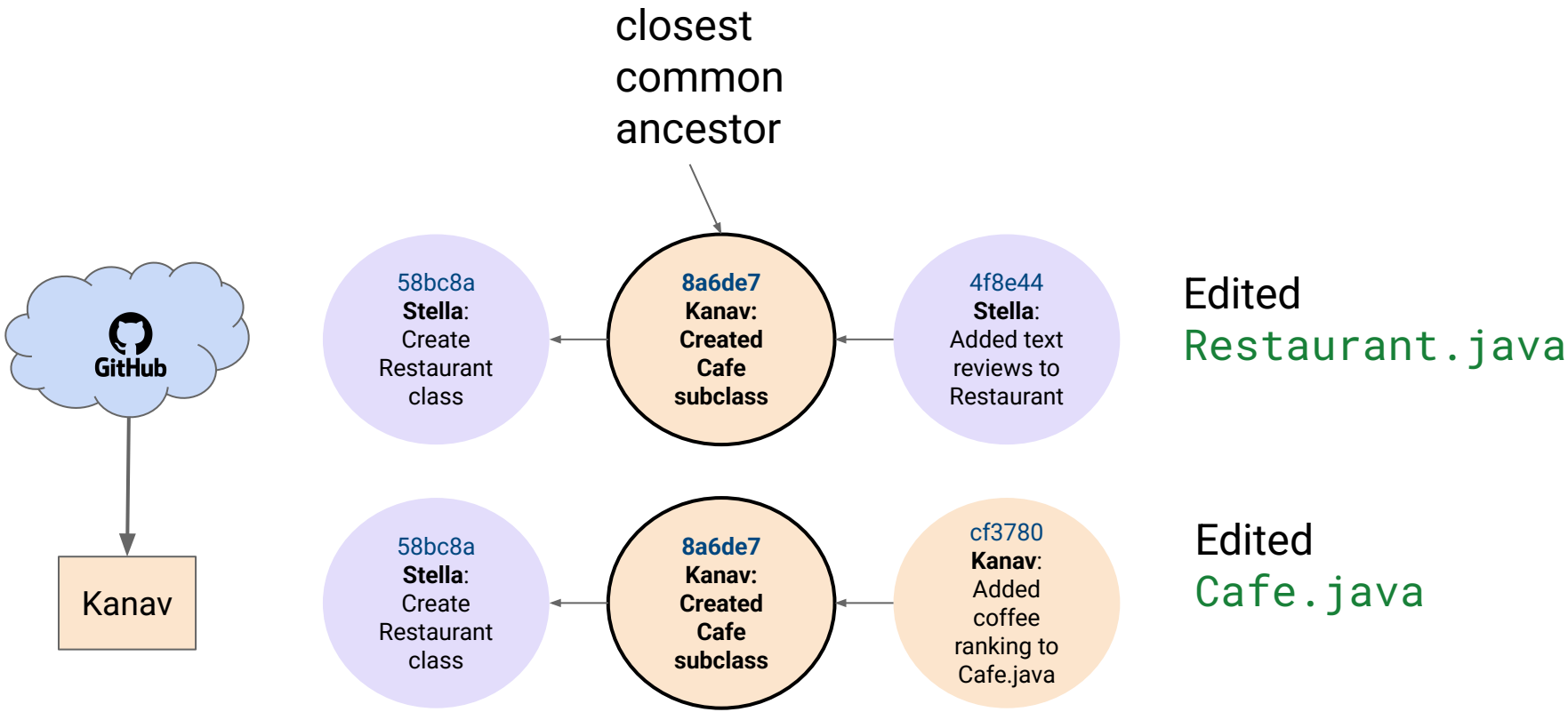
Collaborating on the same commit history

First step of the merge: identify the closest common ancestor



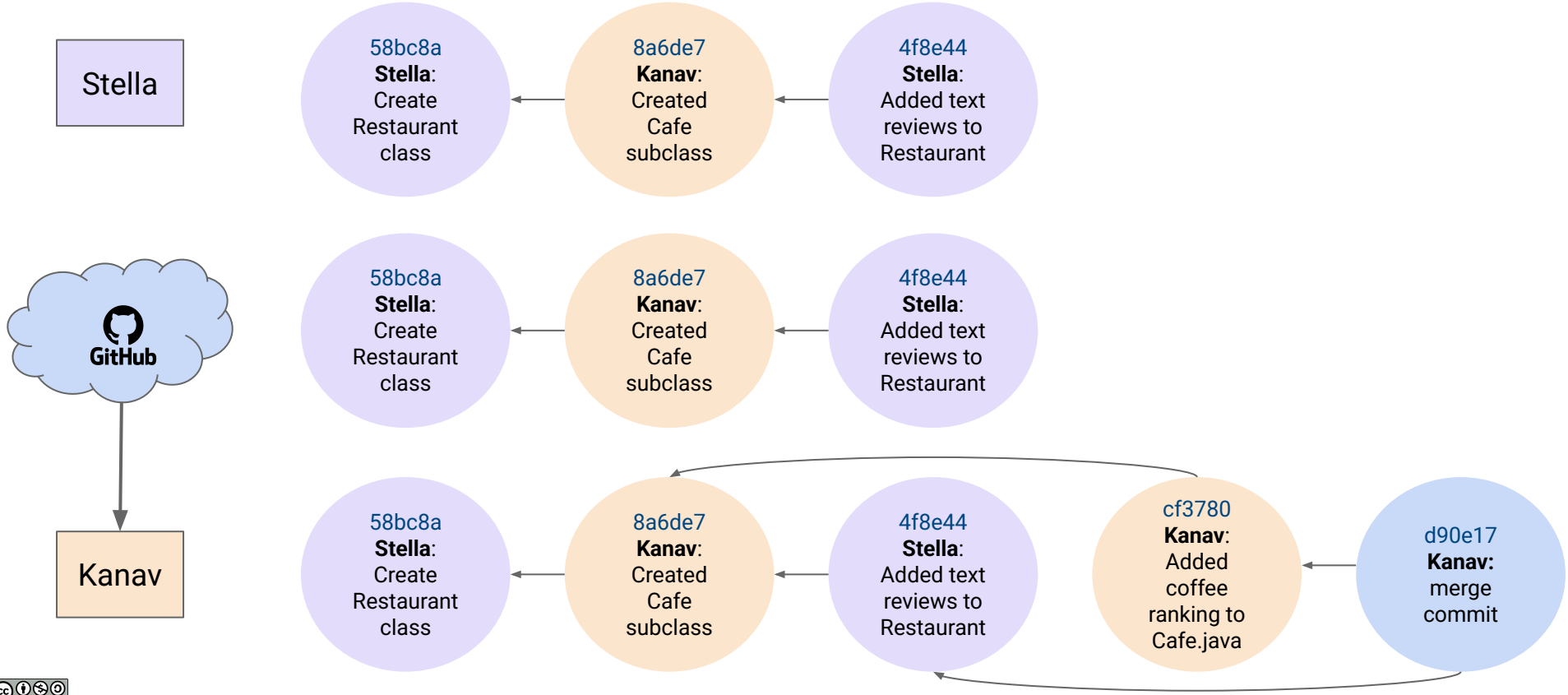
Collaborating on the same commit history

Second step of the merge: combine changes and add a merge commit



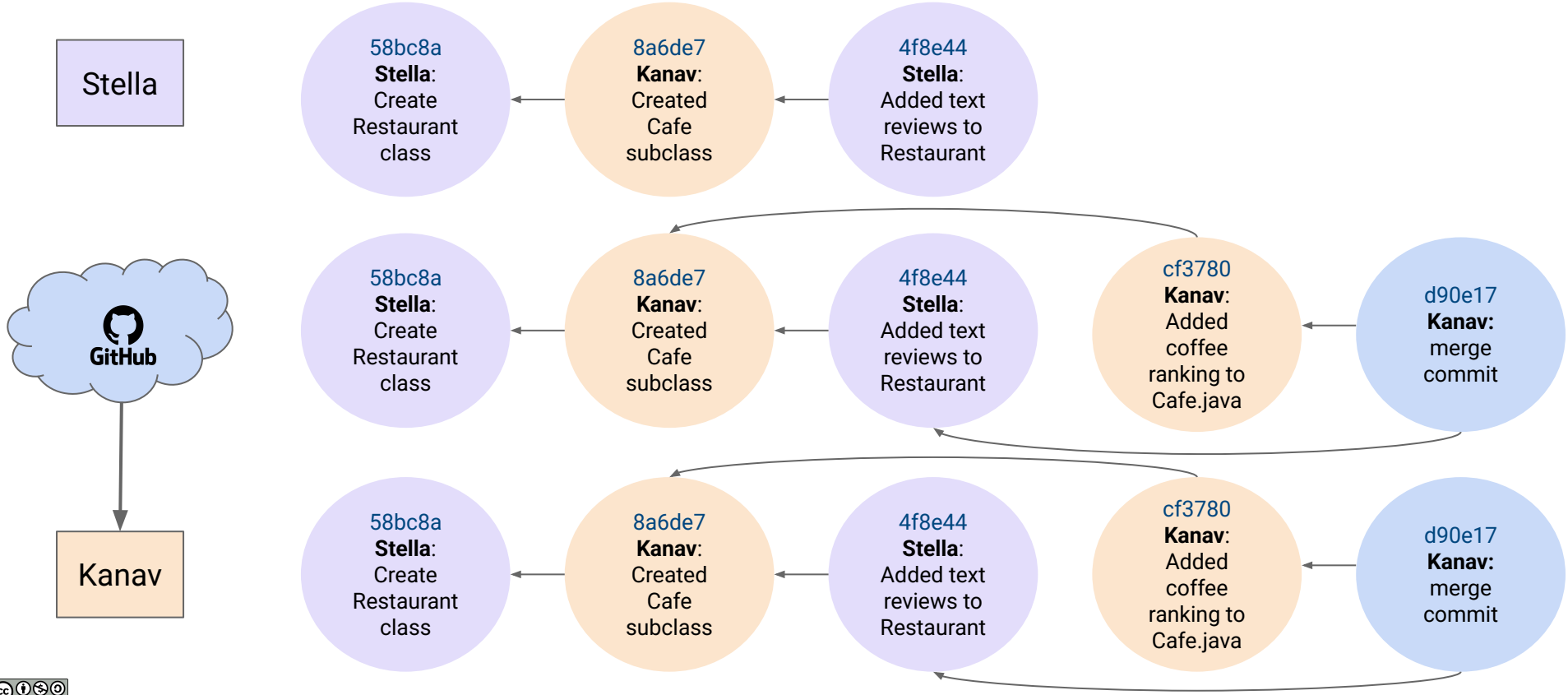
Collaborating on the same commit history

Second step of the merge: combine changes and add a merge commit



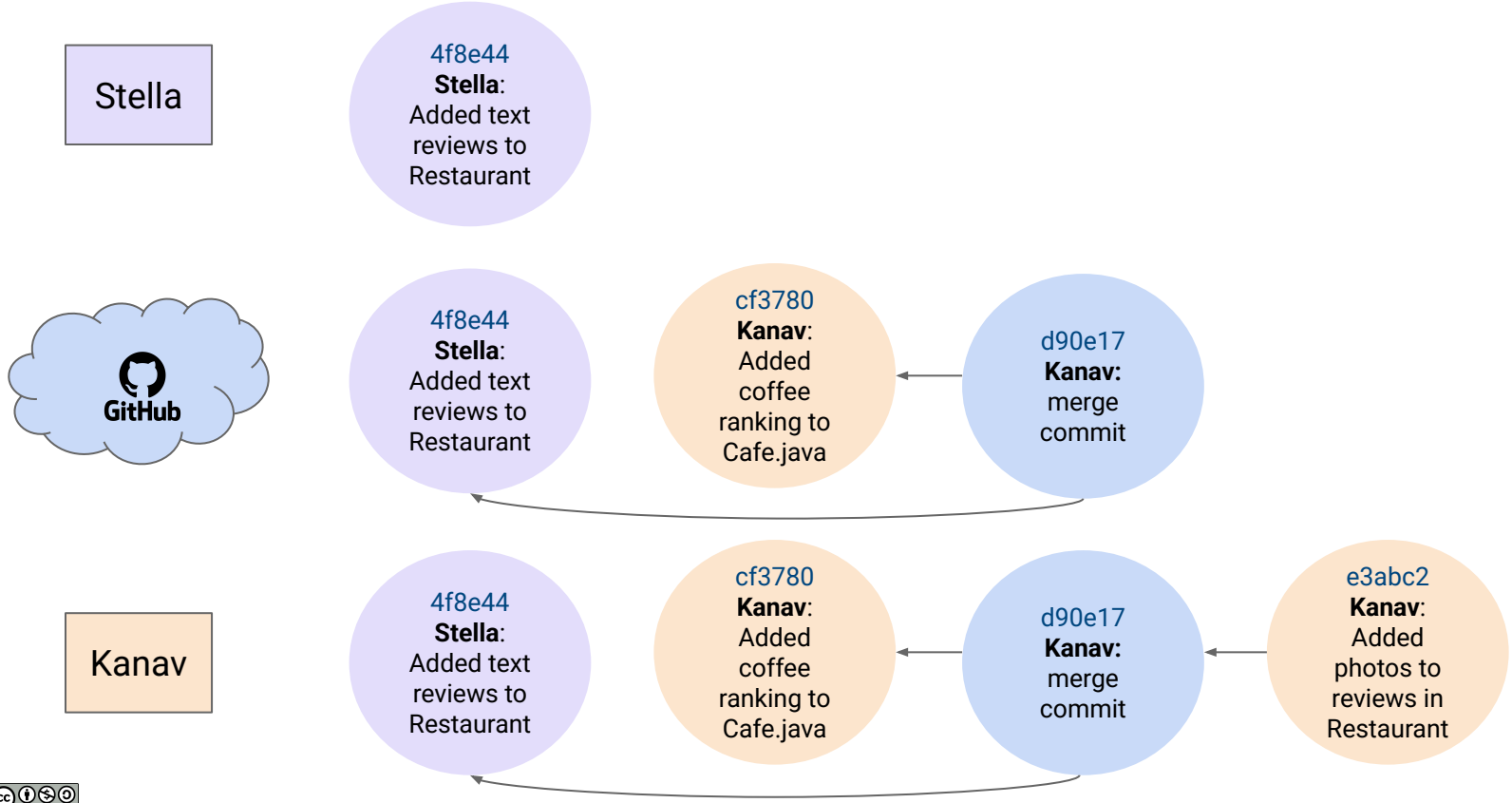
Collaborating on the same commit history

Kanav pushes his commit history



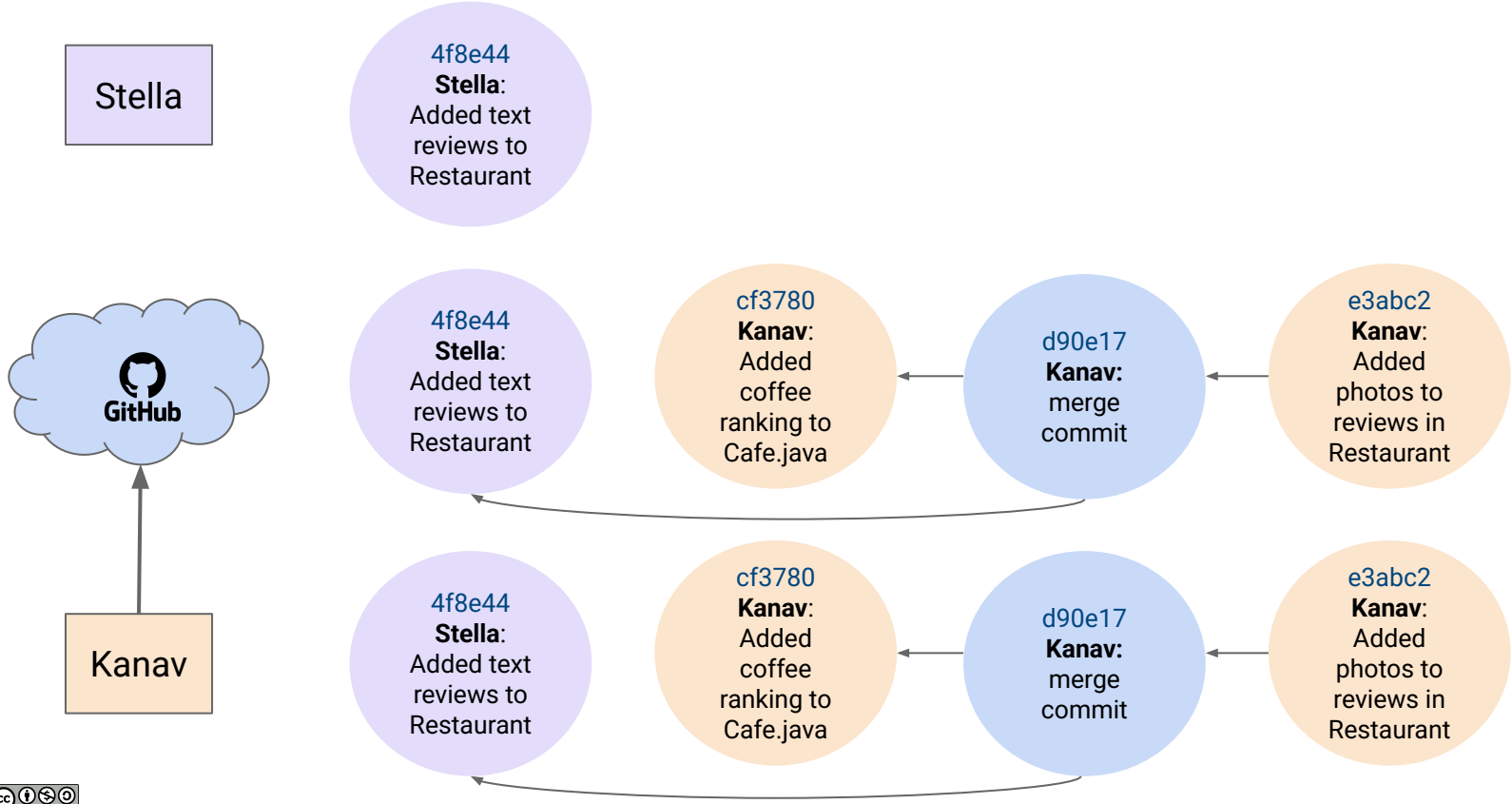
Collaborating on the same commit history

Kanav edits `Restaurant.java` to add photos to reviews and makes a commit



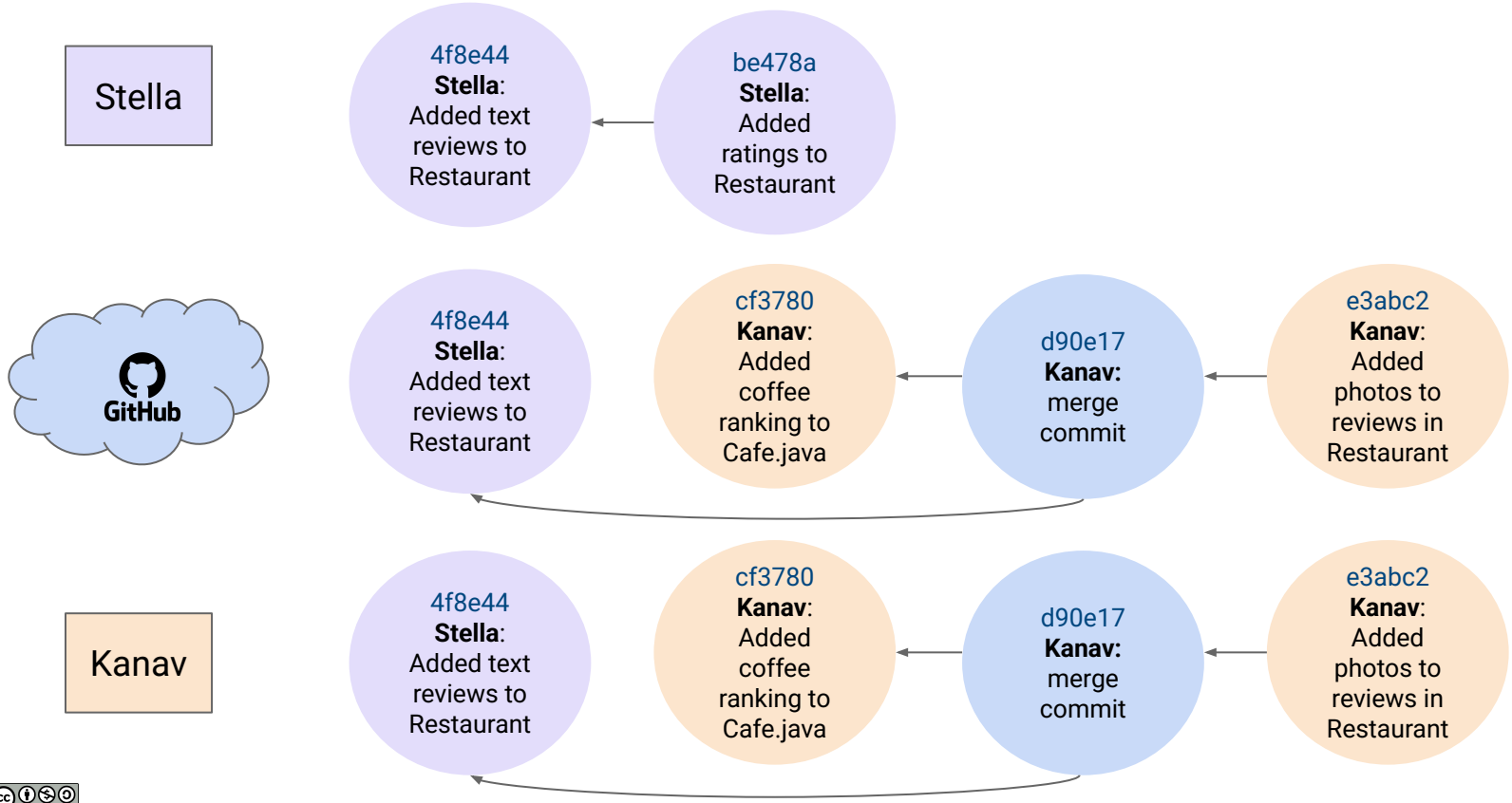
Collaborating on the same commit history

Kanav pushes his changes to GitHub



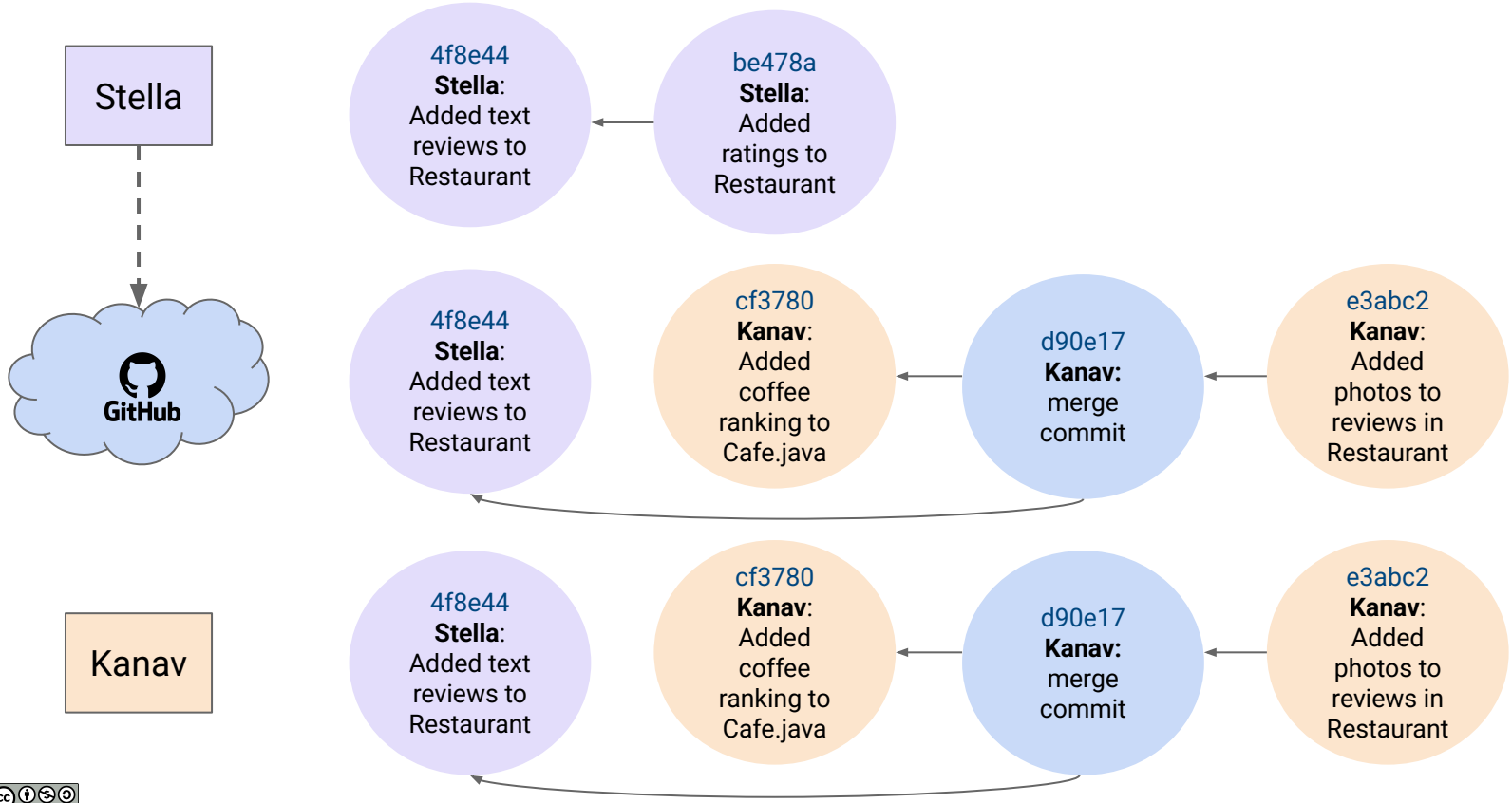
Collaborating on the same commit history

Stella edits `Restaurant.java` to add star ratings to reviews and makes a commit



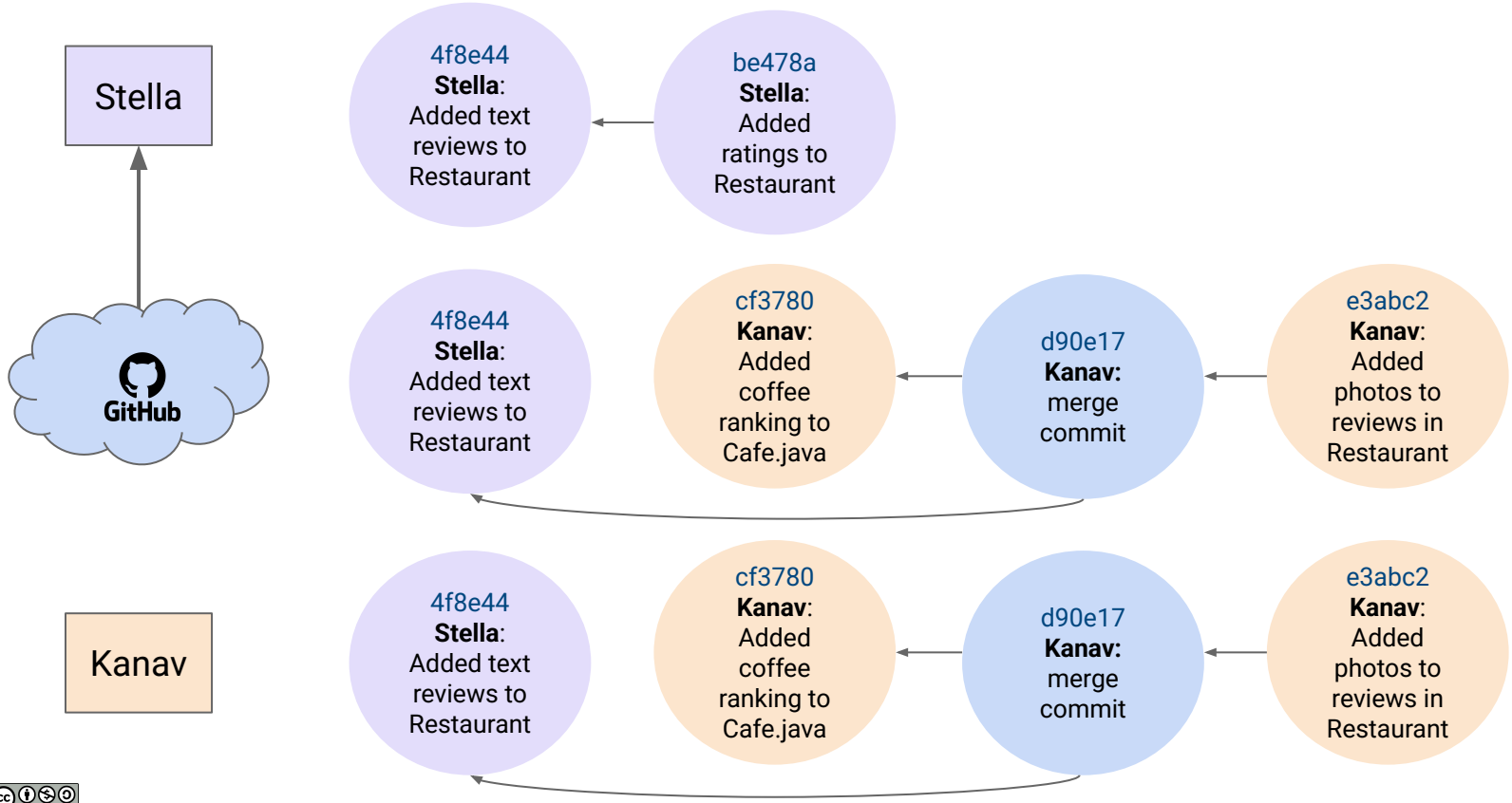
Collaborating on the same commit history

Stella wants to push her changes to GitHub, but…



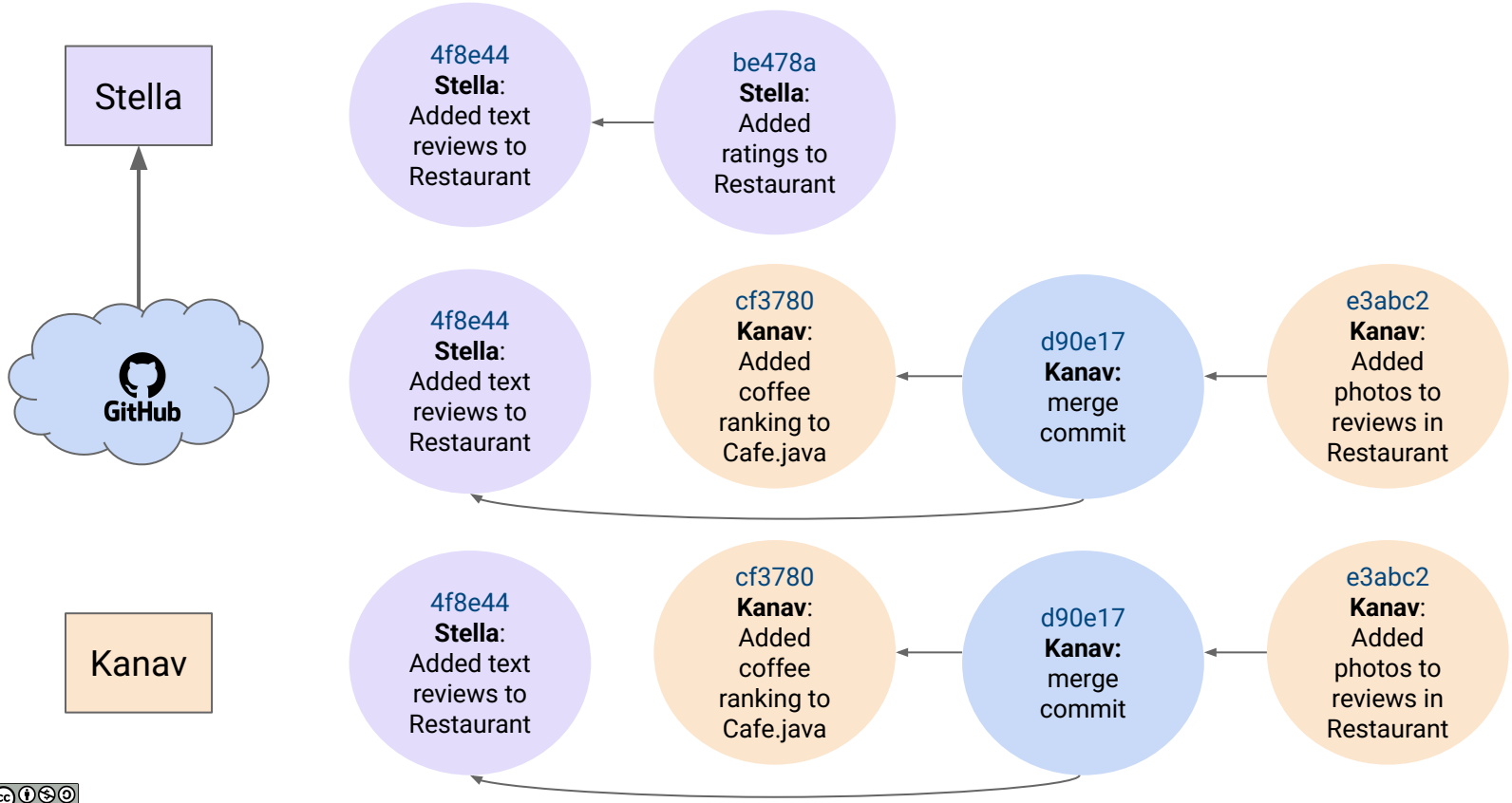
Collaborating on the same commit history

Stella wants to push her changes to GitHub, but she first pulls (pull before push!)



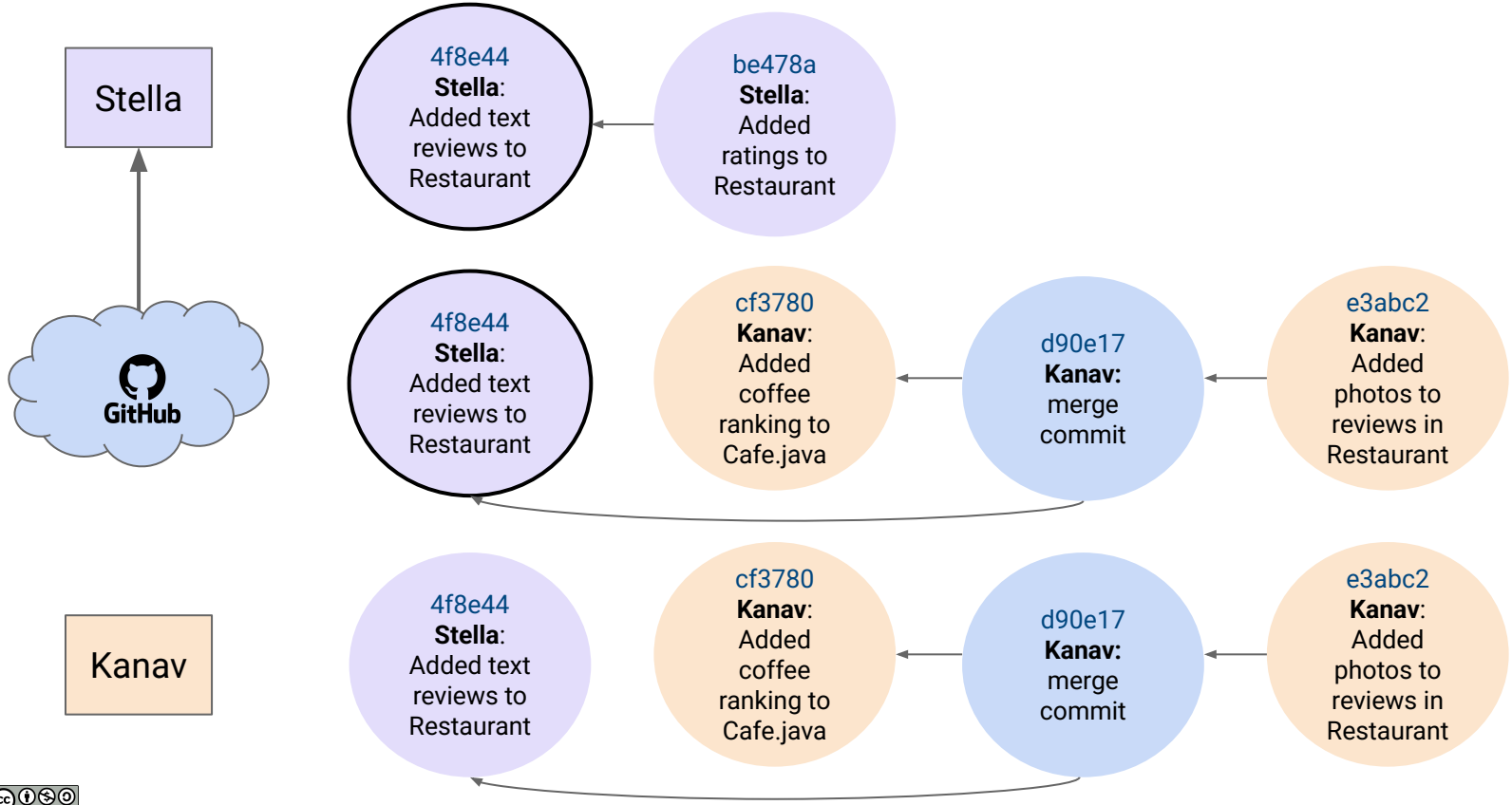
Collaborating on the same commit history

Stella pulls from GitHub, but local and remote histories diverge.



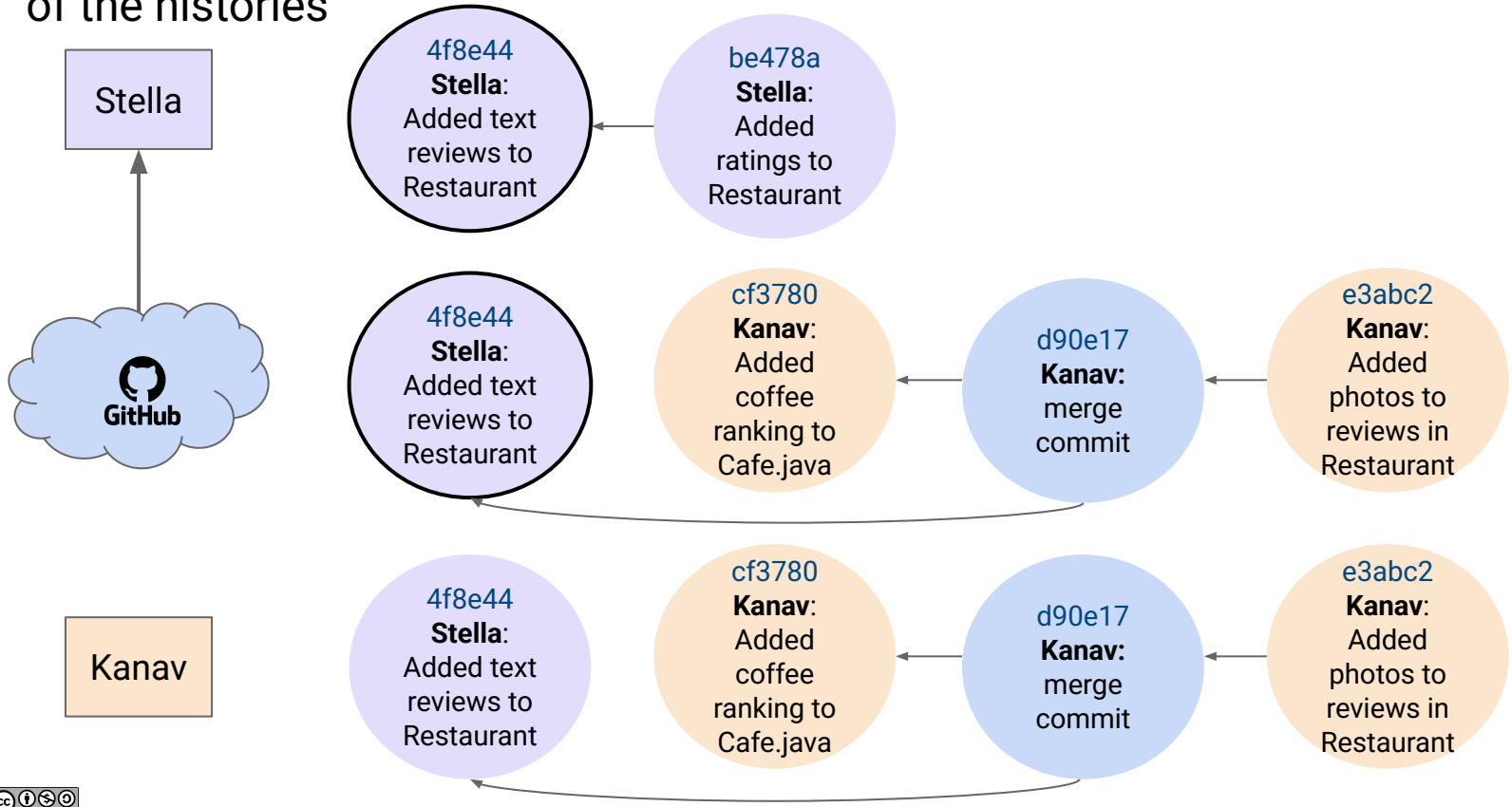
Collaborating on the same commit history

Step 1: Git identifies Closest Common Ancestor



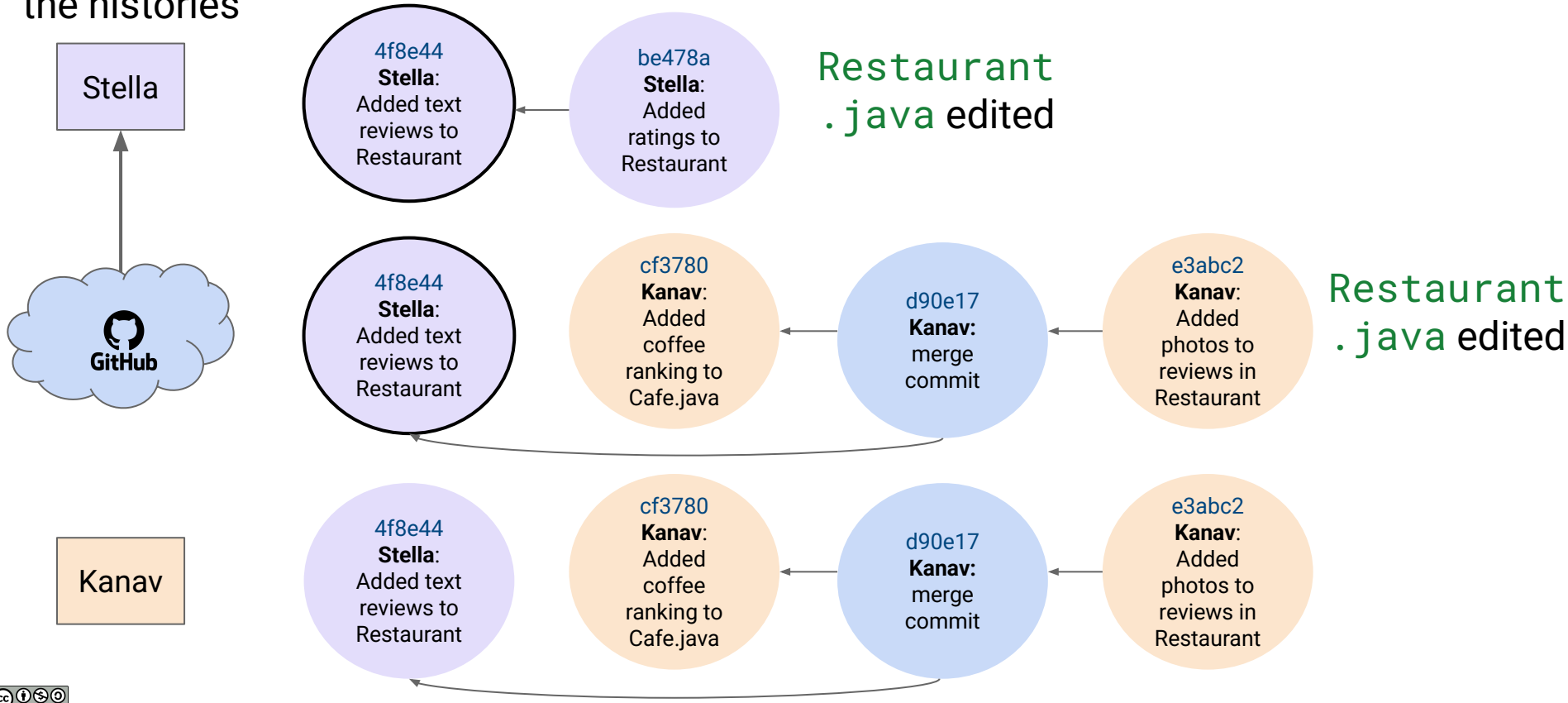
Collaborating on the same commit history

Step 2: Combine changes from the Closest Common Ancestor to the latest commits of the histories



Collaborating on the same commit history

Step 2: Combine changes made from Closest Common Ancestor to latest commits of the histories



Merge conflicts

- If both histories modified the same file in different ways, which version do we use in our merge commit?
  - Git cannot automatically compute the merge
  - Answer: let the developer decide by resolving a merge conflict

Merge conflicts

- If both histories modified the same file in different ways, which version do we use in our merge commit?
  - Git cannot automatically compute the merge
  - Answer: let the developer decide by resolving a merge conflict

```
public class Restaurant {
    <<<<<<< HEAD
    private int[] ratings;
    public Restaurant (int[]
    ratings) {
        this.ratings = ratings;
    =======
    private Photo[] photos;
    public Restaurant (Photo[]
    photos) {
        this.photos = photos
    >>>>>>> e3abc2
    }
}
```

Merge conflicts

- If both histories modified the same file in different ways, which version do we use in our merge commit?
  - Git cannot automatically compute the merge
  - Answer: let the developer decide by resolving a merge conflict

```
public class Restaurant {
    <<<<<<< HEAD
    private int[] ratings;
    public Restaurant (int[] ratings) {
        this.ratings = ratings;
    =======
    private Photo[] photos;
    public Restaurant (Photo[] photos) {
        this.photos = photos
    >>>>>>> e3abc2
    }
}
```

Local version

Merge conflicts

- If both histories modified the same file in different ways, which version do we use in our merge commit?
  - Git cannot automatically compute the merge
  - Answer: let the developer decide by resolving a merge conflict

```
public class Restaurant {
    <<<<<<< HEAD
    private int[] ratings;
    public Restaurant (int[] ratings) {
        this.ratings = ratings;
    =======
    private Photo[] photos;
    public Restaurant (Photo[] photos) {
        this.photos = photos
    >>>>>>> e3abc2
    }
}
```

Local version

Remote version

Merge conflicts

- If both histories modified the same file in different ways, which version do we use in our merge commit?
  - Git cannot automatically compute the merge
  - Answer: let the developer decide by resolving a merge conflict
- Resolve merge conflicts by choosing the correct version (or combining parts of both) and removing all the extra text. Test, then, finish the commit with `git commit`

```
public class Restaurant {
    <<<<<<< HEAD
    private int[] ratings;
    public Restaurant (int[] ratings) {
        this.ratings = ratings;
    =======
    private Photo[] photos;
    public Restaurant (Photo[] photos) {
        this.photos = photos
    >>>>>>> e3abc2
    }
}
```

Local version

Remote version

Merge conflicts are messy!

- Merge conflicts require a lot of manual work
- Occur when we have multiple developers editing same file in different ways
  - Almost inevitable!

Merge conflicts are messy!

- Merge conflicts require a lot of manual work
- Occur when we have multiple developers editing same file in different ways
  - Almost inevitable!
- Can we reduce the frequency of merges and merge conflicts?

Merge conflicts are messy!

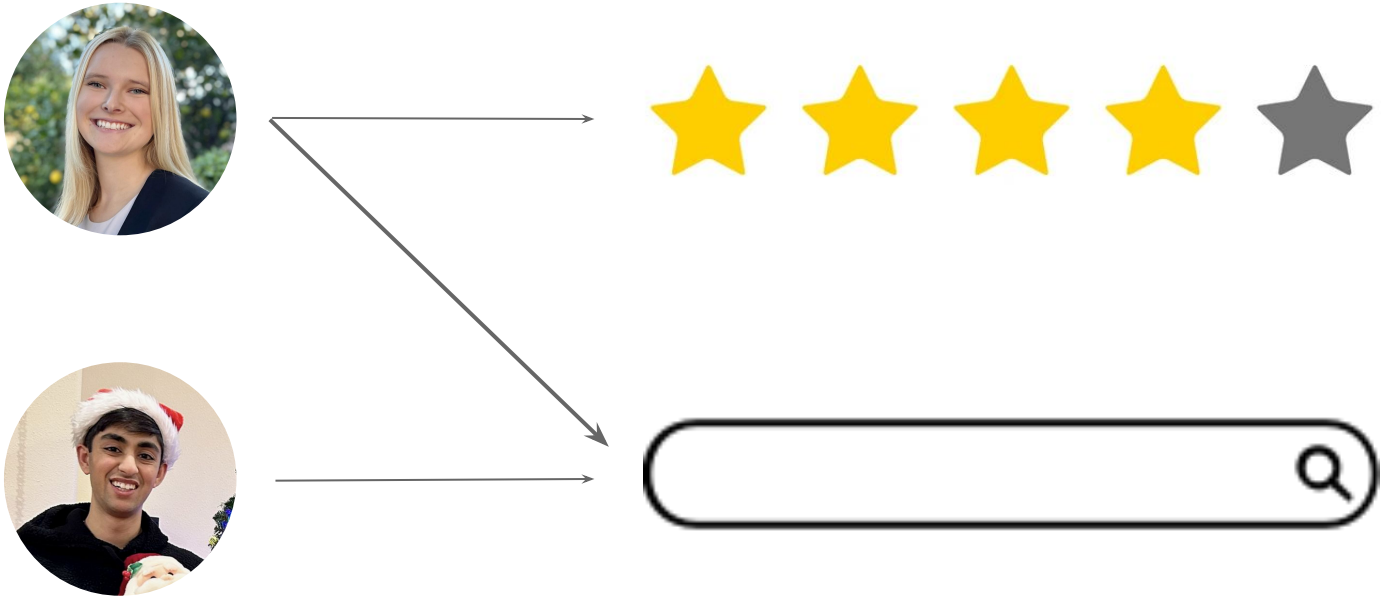
- Merge conflicts require a lot of manual work
- Occur when we have multiple developers editing same file in different ways
  - Almost inevitable!
- Can we reduce the frequency of merges and merge conflicts?
- Can we let developers edit files independently and then only merge when necessary?

Solution 1: Just push and pull less

- Can we let developers edit files independently and then only merge when necessary?
- Solution: push and pull only when you have completed a feature.

Solution 1: Just push and pull less

- Can we let developers edit files independently and then only merge when necessary?
- Solution: push and pull only when you have completed a feature.
- But…what if two people wanted to work on the same feature together?
  - We need pushing and pulling to work off each others' changes



From linear to branches

- With a linked list linear commit history, everyone adds their commits to the same line of development
  - This means that your changes will affect your collaborators' changes
  - Can we isolate changes from each other until they're ready?

From linear to branches

- With a linked list linear commit history, everyone adds their commits to the same line of development
  - This means that your changes will affect your collaborators' changes
  - Can we isolate changes from each other until they're ready?
- Solution: we can create other lines of development (**branches**)
  - Developers can work independently on in-progress features without interference from other features
  - This may be useful for ambition features in Project 3B

Git Branching and Merging

Lecture 31, CS61B Spring 2025

Branches

- Branches are pointers to commits
  - Generally, they point to the latest commit in a line of development

Branches

- Branches are pointers to commits
  - Generally, they point to the latest commit in a line of development
- Can create new branches with `git branch <branch-name>`

Branches

- Branches are pointers to commits
  - Generally, they point to the latest commit in a line of development
- Can create new branches with `git branch <branch-name>`
  - All repositories start off with one branch (normally `main` branch)

Branches

- Branches are pointers to commits
  - Generally, they point to the latest commit in a line of development
- Can create new branches with `git branch <branch-name>`
  - All repositories start off with one branch (normally `main` branch)
- Can set the active branch (HEAD) with `git switch <branch-name>`

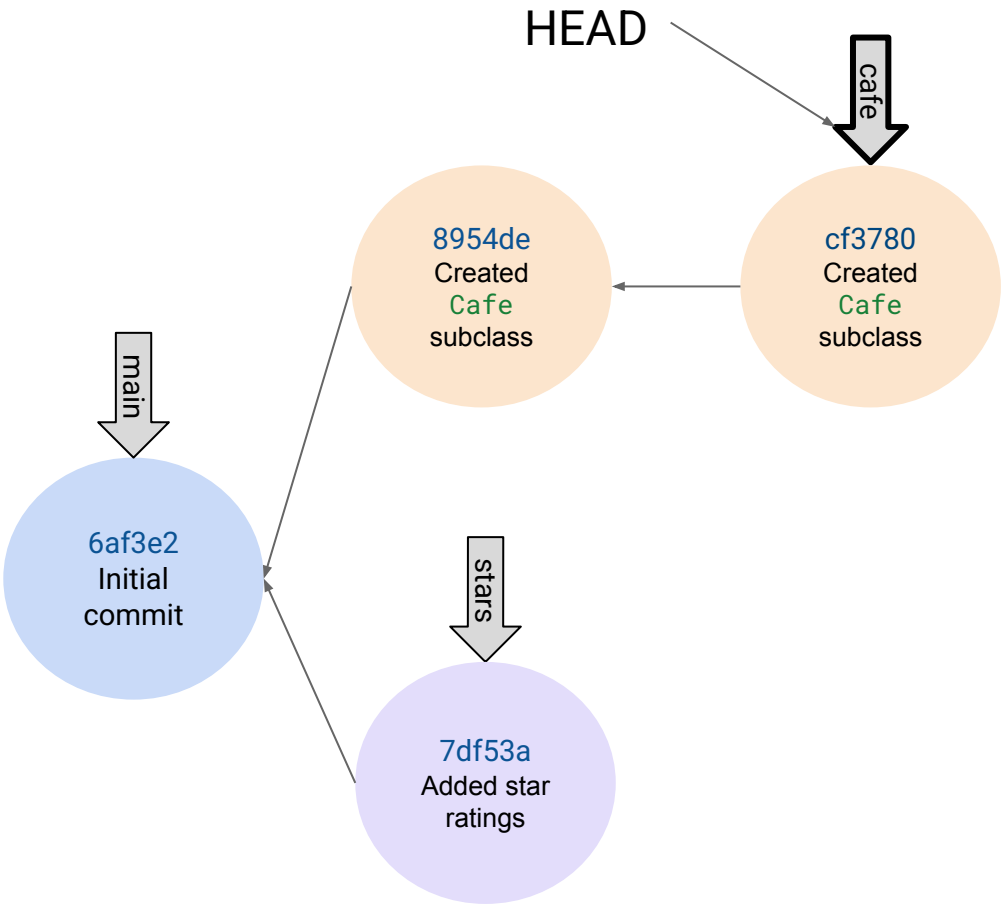
Branches

- Branches are pointers to commits
  - Generally, they point to the latest commit in a line of development
- Can create new branches with `git branch <branch-name>`
  - All repositories start off with one branch (normally `main` branch)
- Can set the active branch (HEAD) with `git switch <branch-name>`
  - When we create a commit, we move *only* the active branch forward to point to the new commit

Branches

- Branches are pointers to commits
  - Generally, they point to the latest commit in a line of development
- Can create new branches with `git branch <branch-name>`
  - All repositories start off with one branch (normally `main` branch)
- Can set the active branch (HEAD) with `git switch <branch-name>`
  - When we create a commit, we move *only* the active branch forward to point to the new commit
  - This means development on the active branch does not affect other branches

Branching allows you to have other lines of development



Merging

- Branching allows us to have lines of development that diverge from `main`

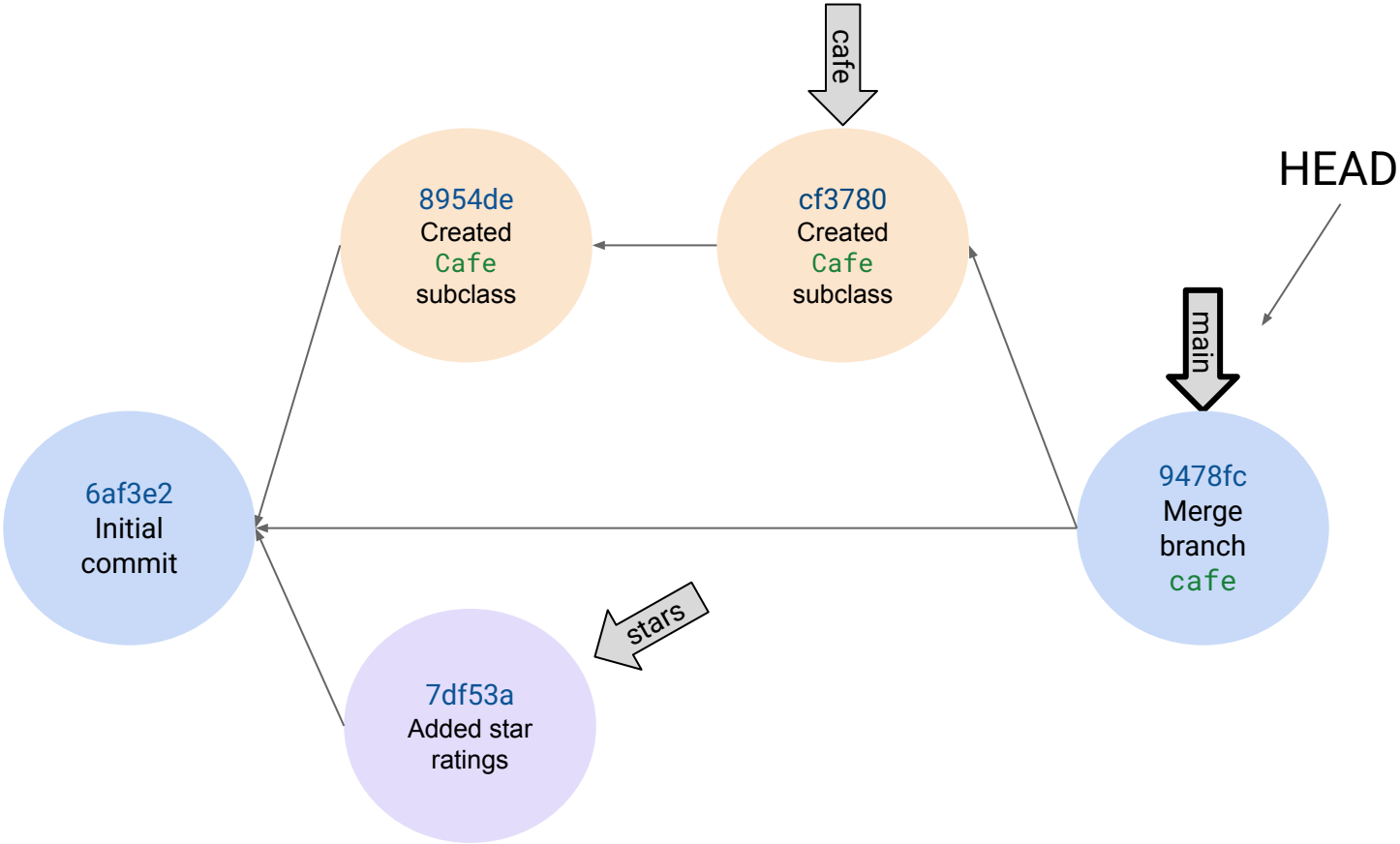
Merging

- Branching allows us to have lines of development that diverge from `main`
- Join two lines of development back together by merging them!

Merging

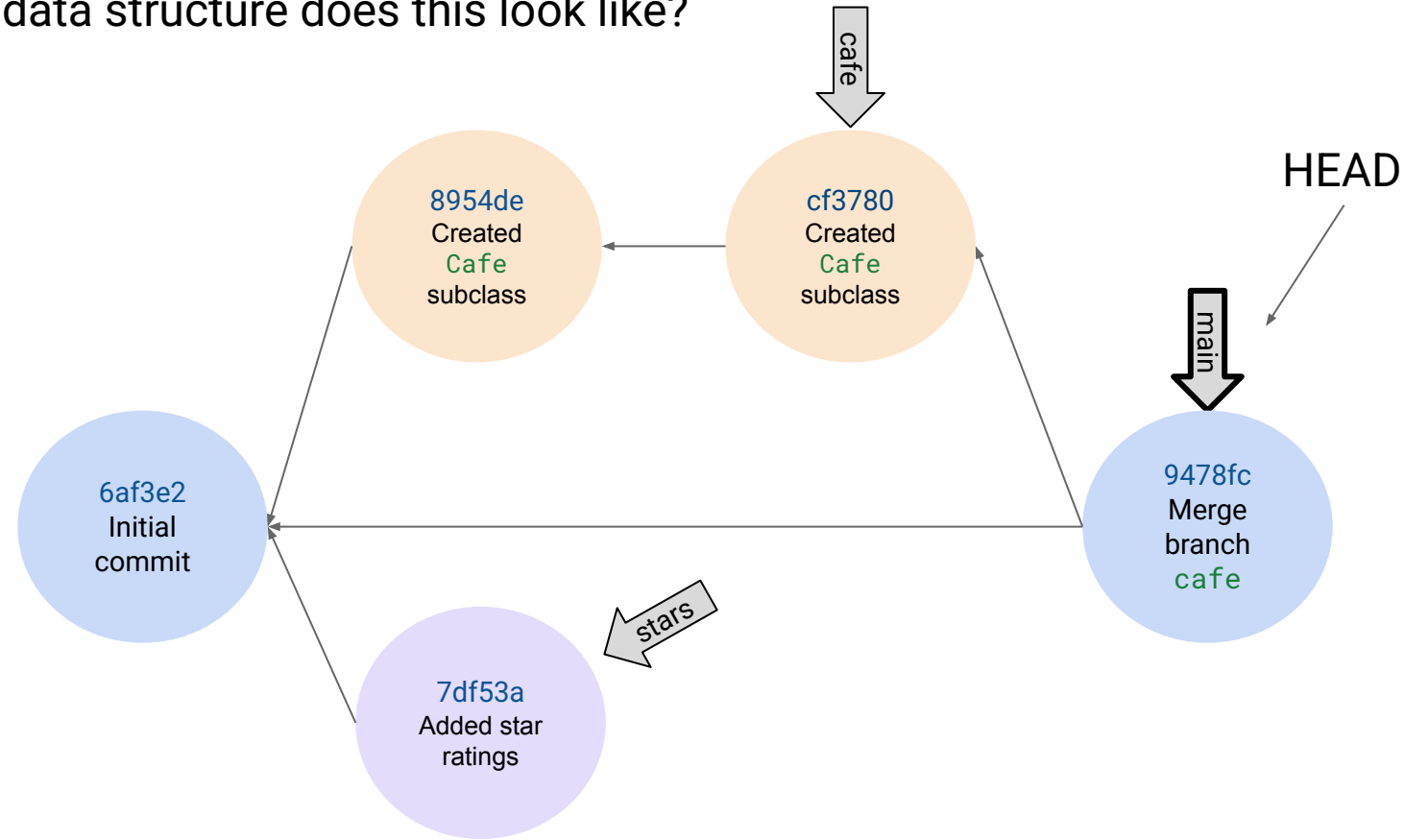
- Branching allows us to have lines of development that diverge from `main`
- Join two lines of development back together by merging them!
- `git merge <branch_name>` merges the specified branch into the active branch and creates a merge commit
  - Merge commits have two parents (commits at the tips of the branches)

Merging joins together two lines of development



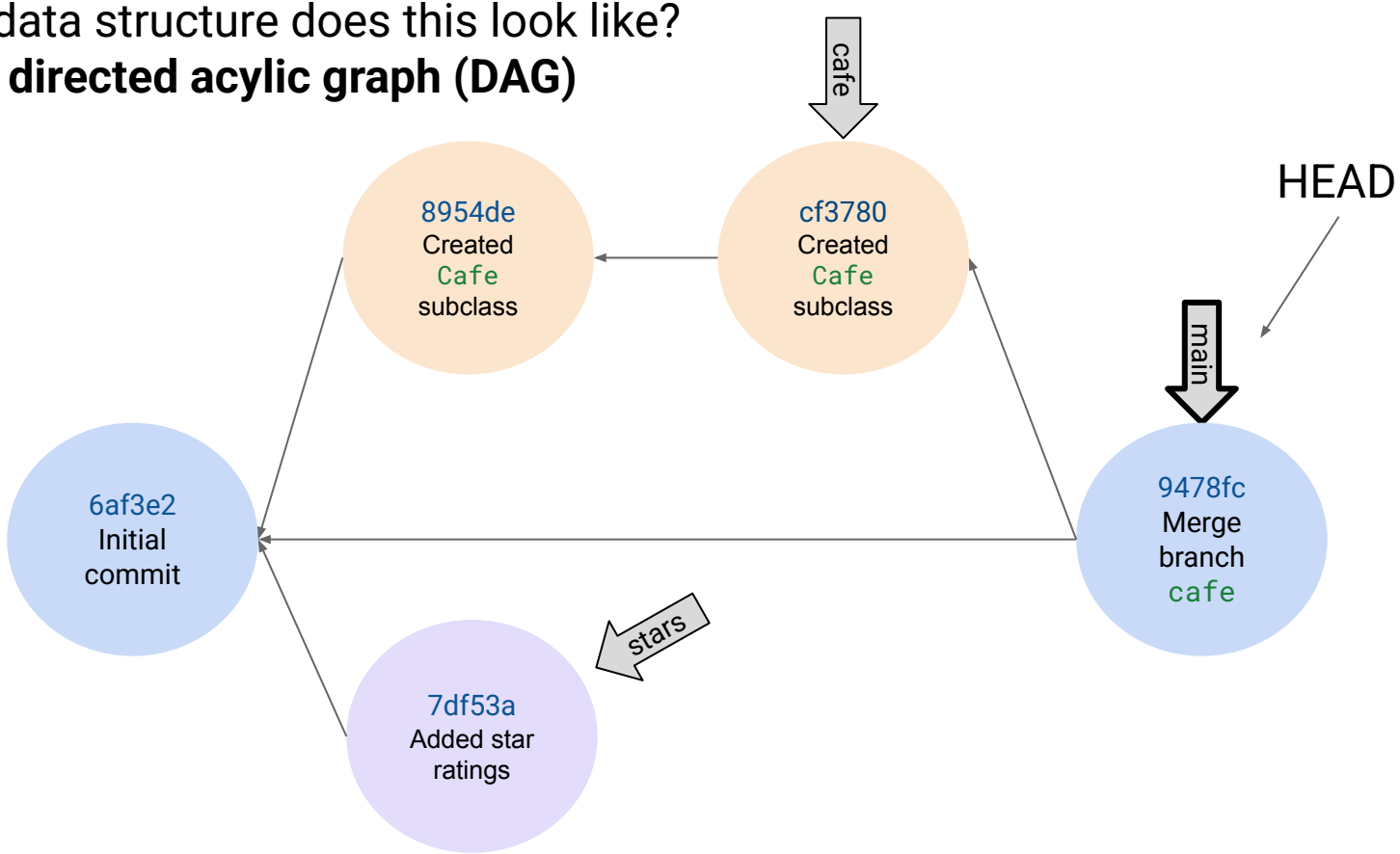
Merging joins together two lines of development

- What data structure does this look like?



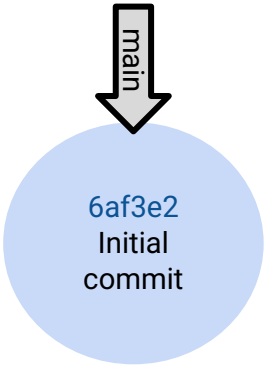
Merging joins together two lines of development

- What data structure does this look like?
  - **A directed acylic graph (DAG)**



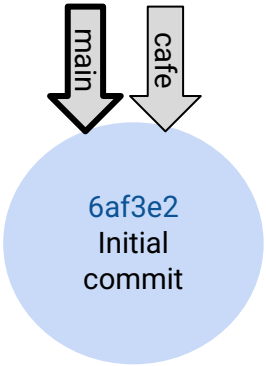
Branching demo

Initially, our `main` branch starts off by pointing to our initial commit.



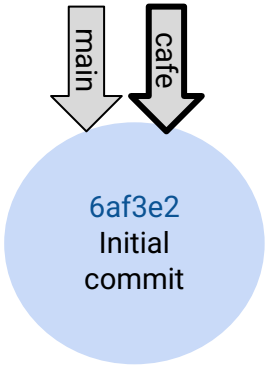
Branching demo

```
git branch cafe
```



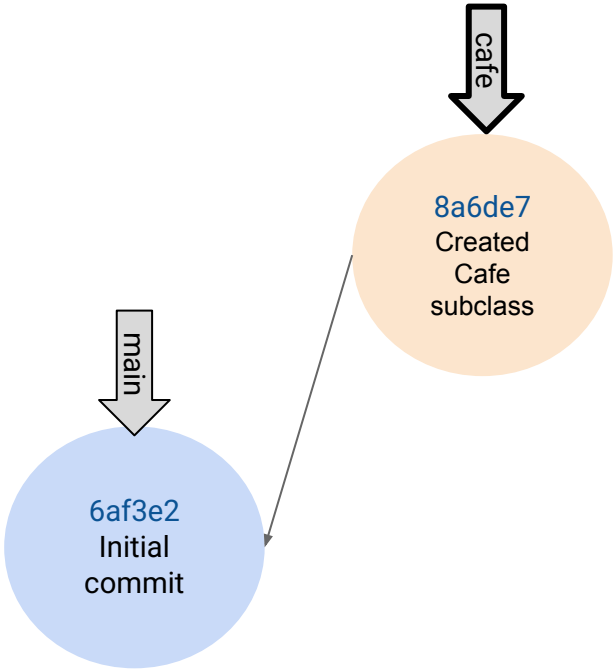
Branching demo

```
git switch cafe
```



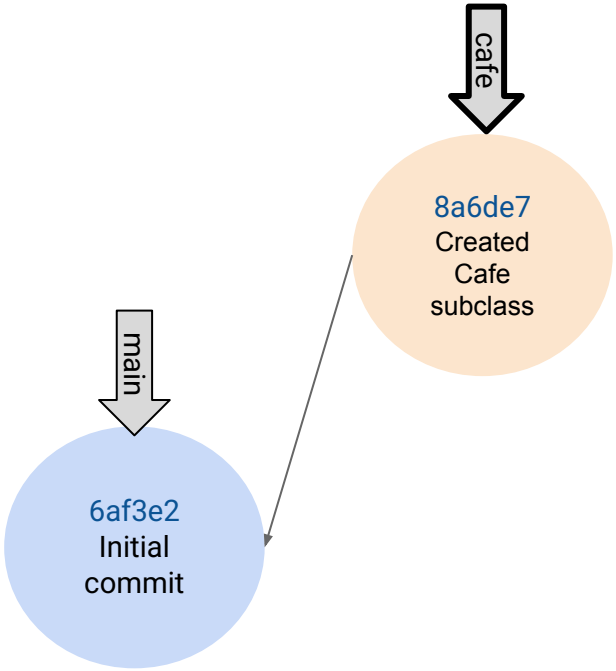
Branching demo

```
git commit -m “Created Cafe subclass”
```



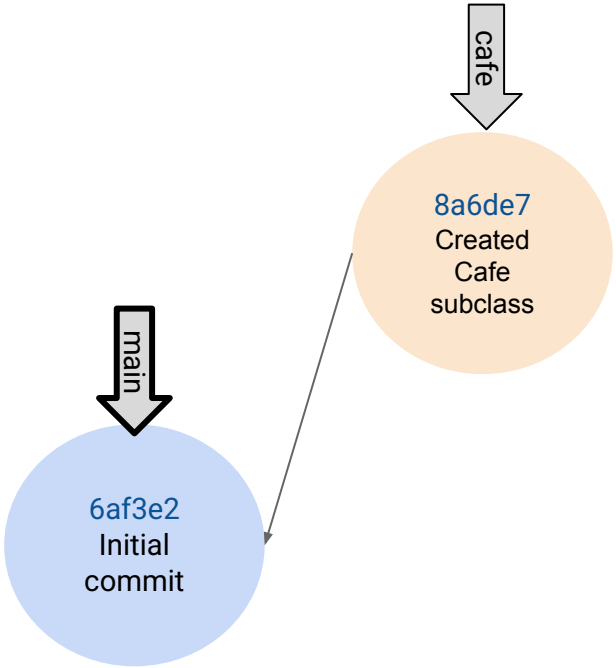
Branching demo

```
git commit -m “Created Cafe subclass”
```



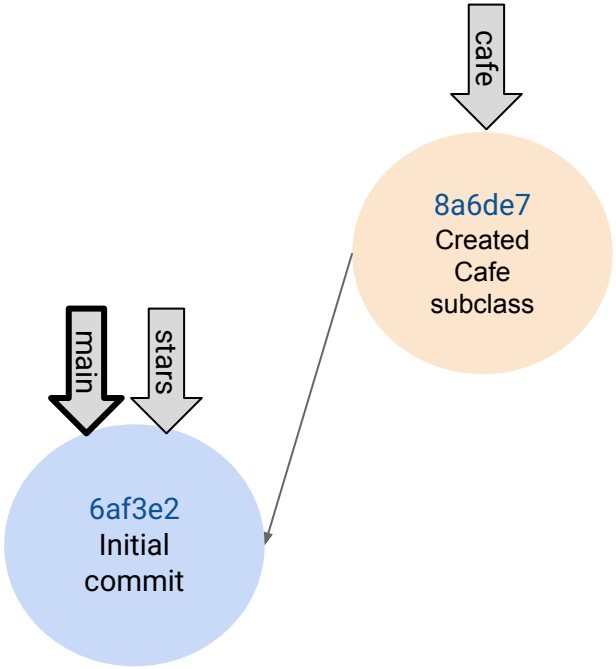
Branching demo

```
git switch main
```



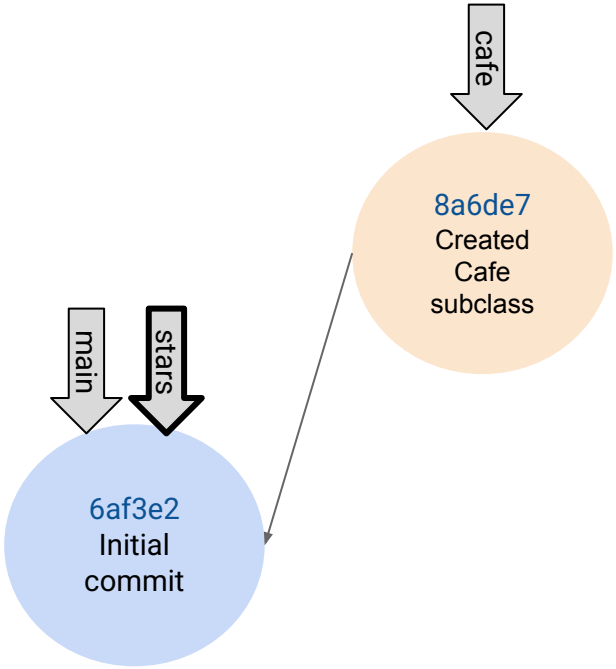
Branching demo

```
git branch stars
```



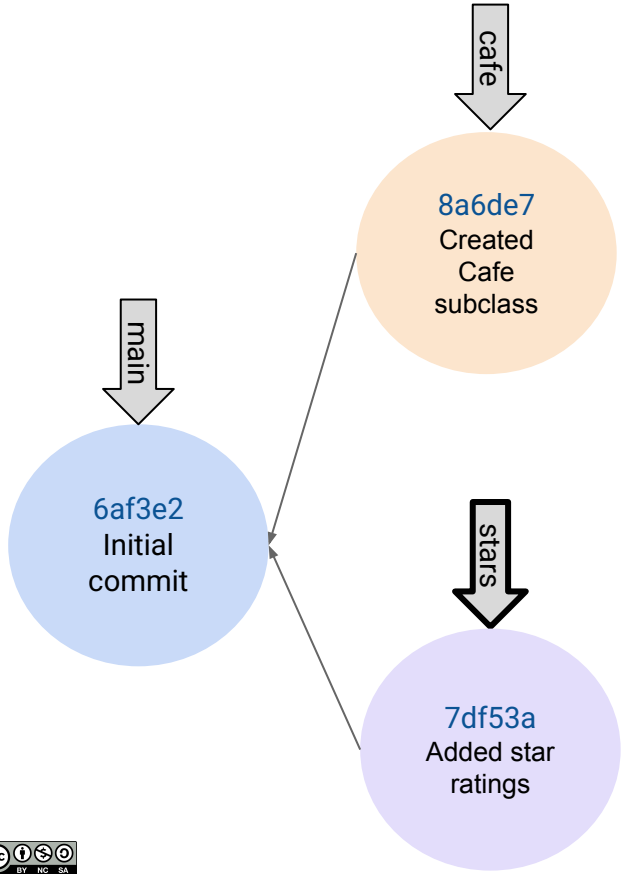
Branching demo

```
git switch stars
```



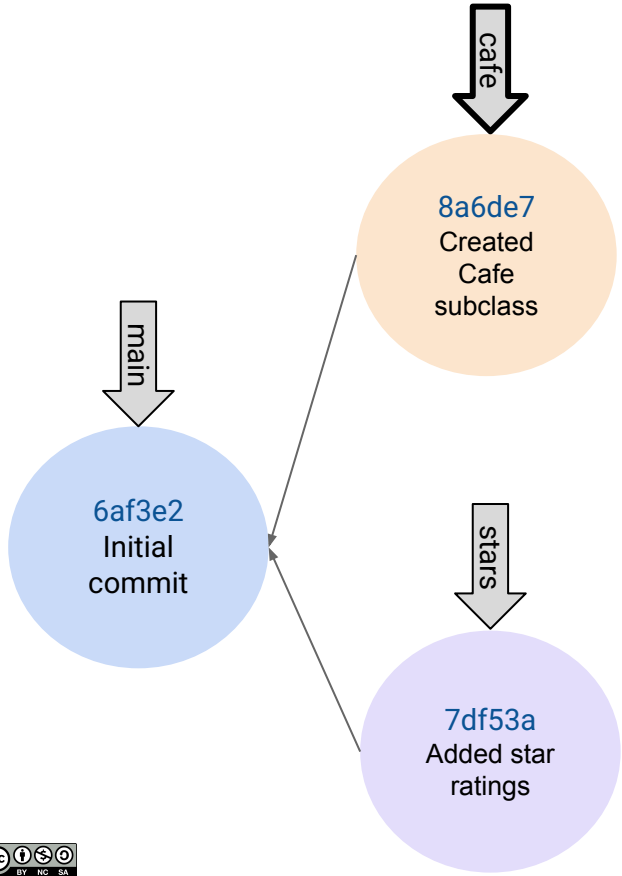
Branching demo

```
git commit -m “Added star ratings”
```



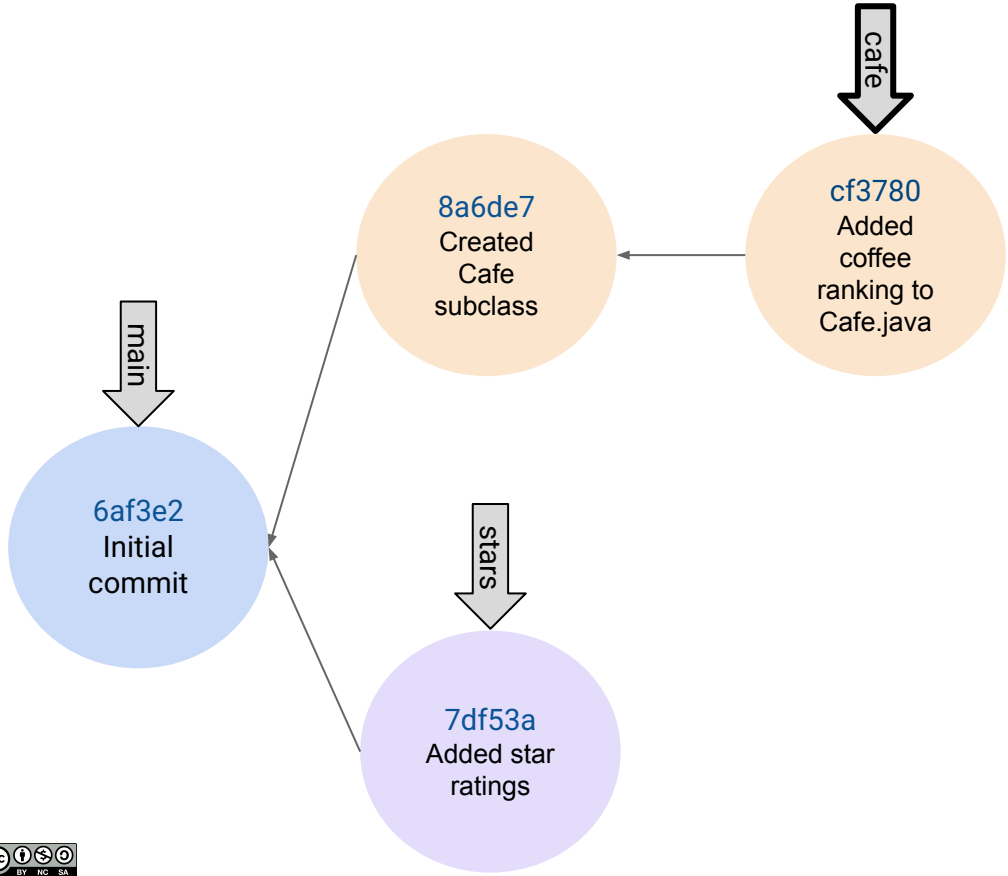
Branching demo

```
git switch cafe
```



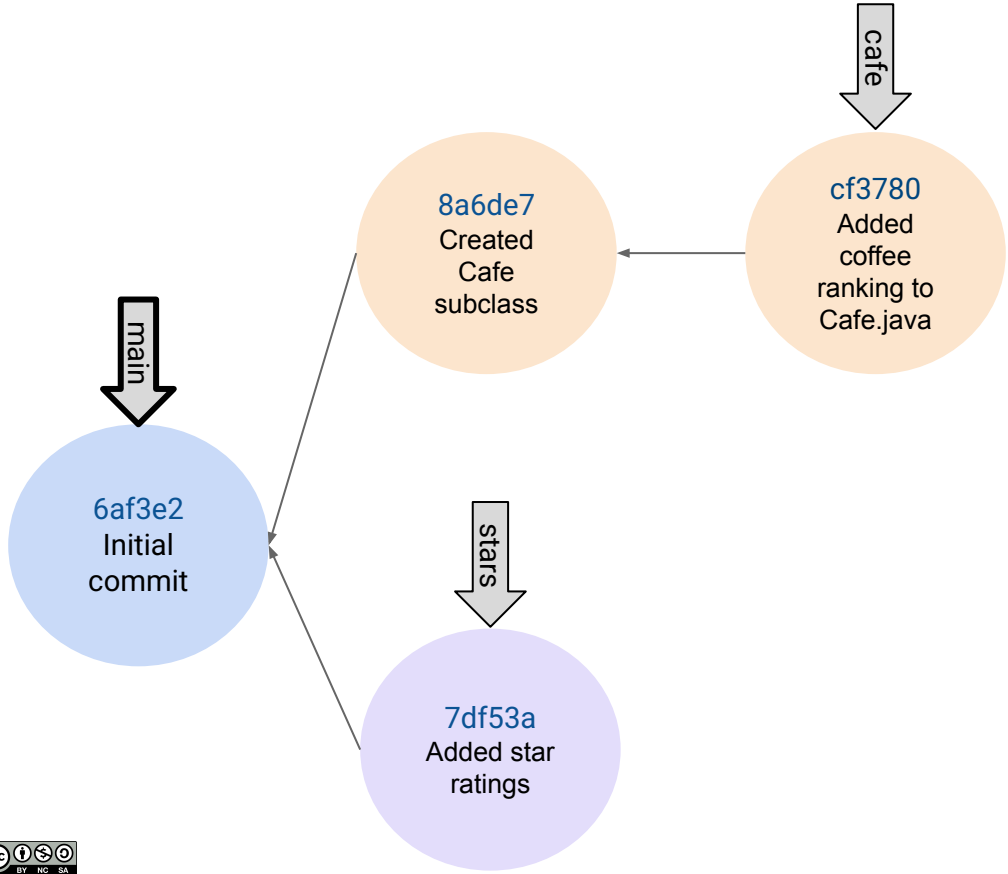
Branching demo

```
git commit -m “Added coffee ranking to Cafe.java”
```



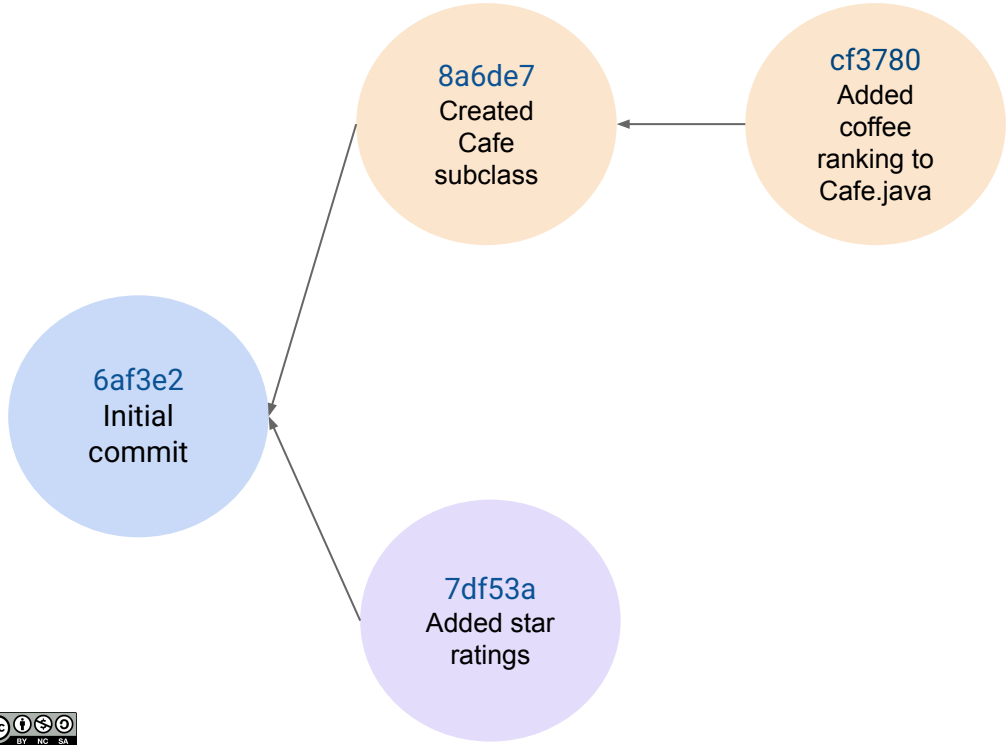
Branching demo

```
git switch main
```



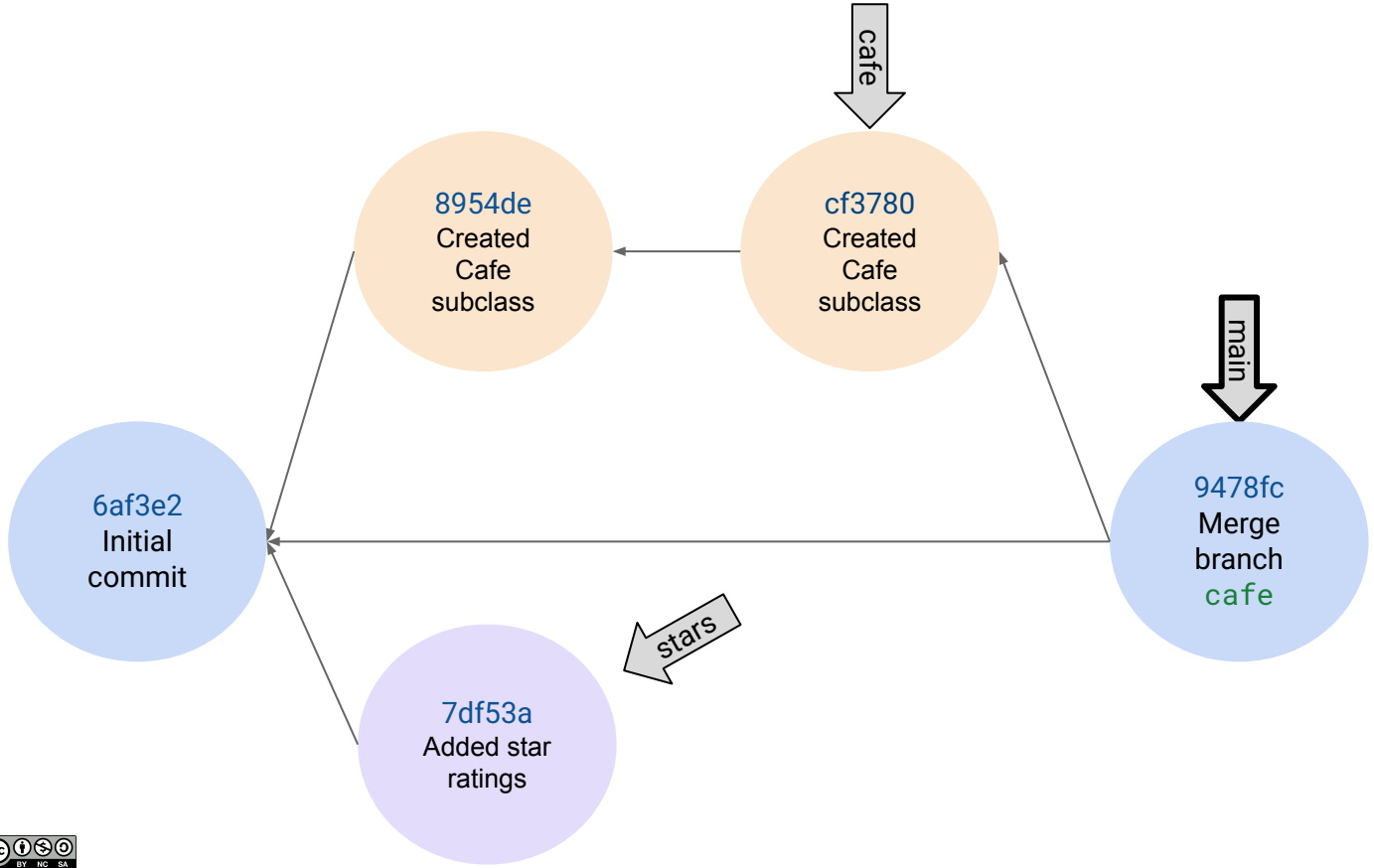
From linked list to trees

- By adding the ability to create branches and other lines of development, our commit history begins to resemble a tree



Branching demo

```
git merge main
```



Feature branches

- Feature branch workflow means that ALL development occurs in branches apart from `main`

Feature branches

- Feature branch workflow means that ALL development occurs in branches apart from `main`
- Name feature branches according to the feature being developed
  - May also include main developer's username (e.g.,: kanav/login-page) or category (e.g., bugfix/ratings-display)

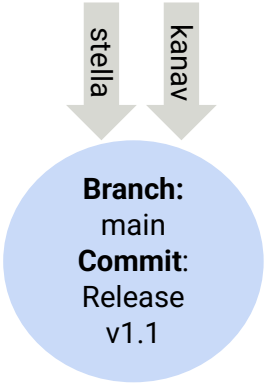
Feature branches

- Feature branch workflow means that ALL development occurs in branches apart from `main`
- Name feature branches according to the feature being developed
  - May also include main developer's username (e.g.,: kanav/login-page) or category (e.g., bugfix/ratings-display)
- Once we finish implementing and testing on a feature branch, we can merge into the `main` branch.

Feature branches

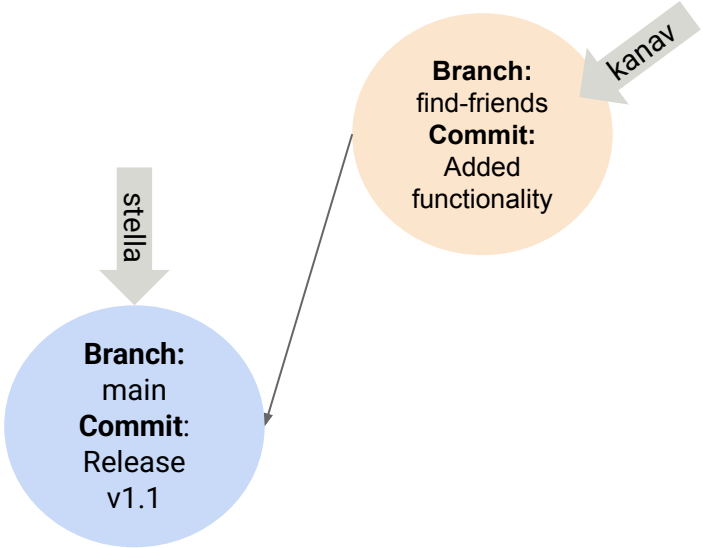
- Feature branch workflow means that ALL development occurs in branches apart from `main`
- Name feature branches according to the feature being developed
  - May also include main developer's username (e.g.,: kanav/login-page) or category (e.g., bugfix/ratings-display)
- Once we finish implementing and testing on a feature branch, we can merge into the `main` branch.
- **Important: Test your changes on your feature branch as much as possible!**

Feature branch demo with multiple developers



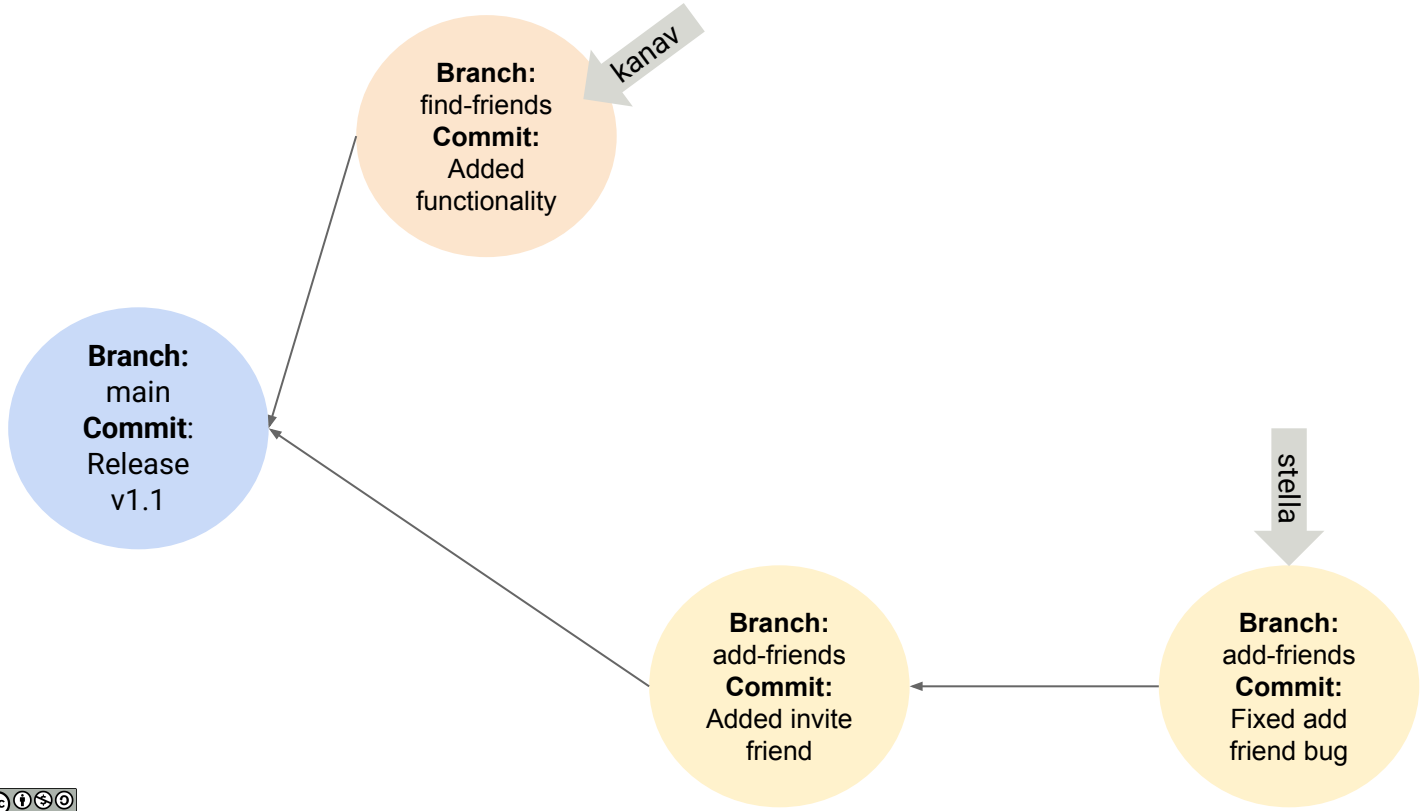
Feature branch demo with multiple developers

Kanav: creates `find-friends` branch



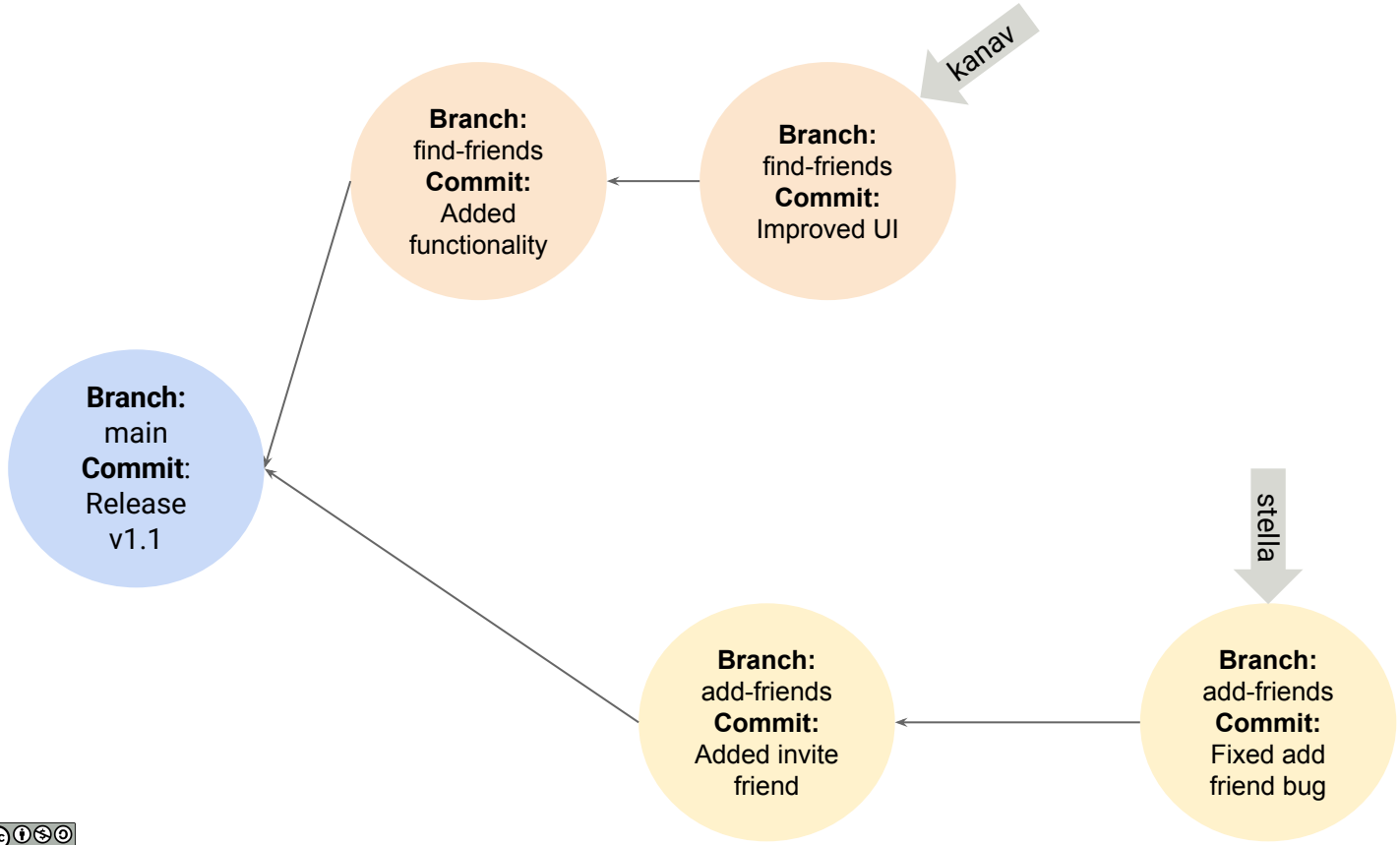
Feature branch demo with multiple developers

Stella: creates `add-friends` branch and works on it



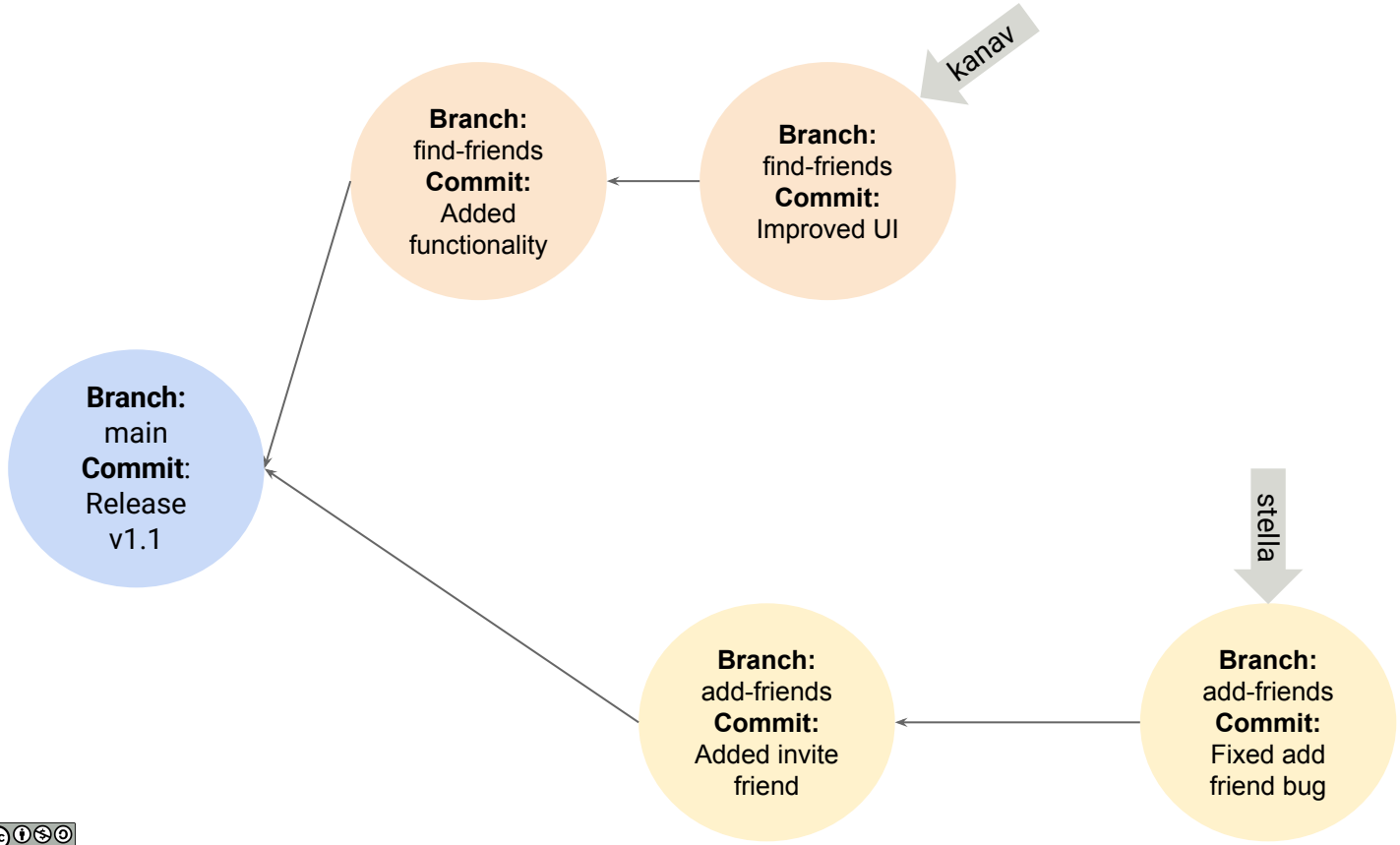
Feature branch demo with multiple developers

Kanav: works some more on `find-friends` branch



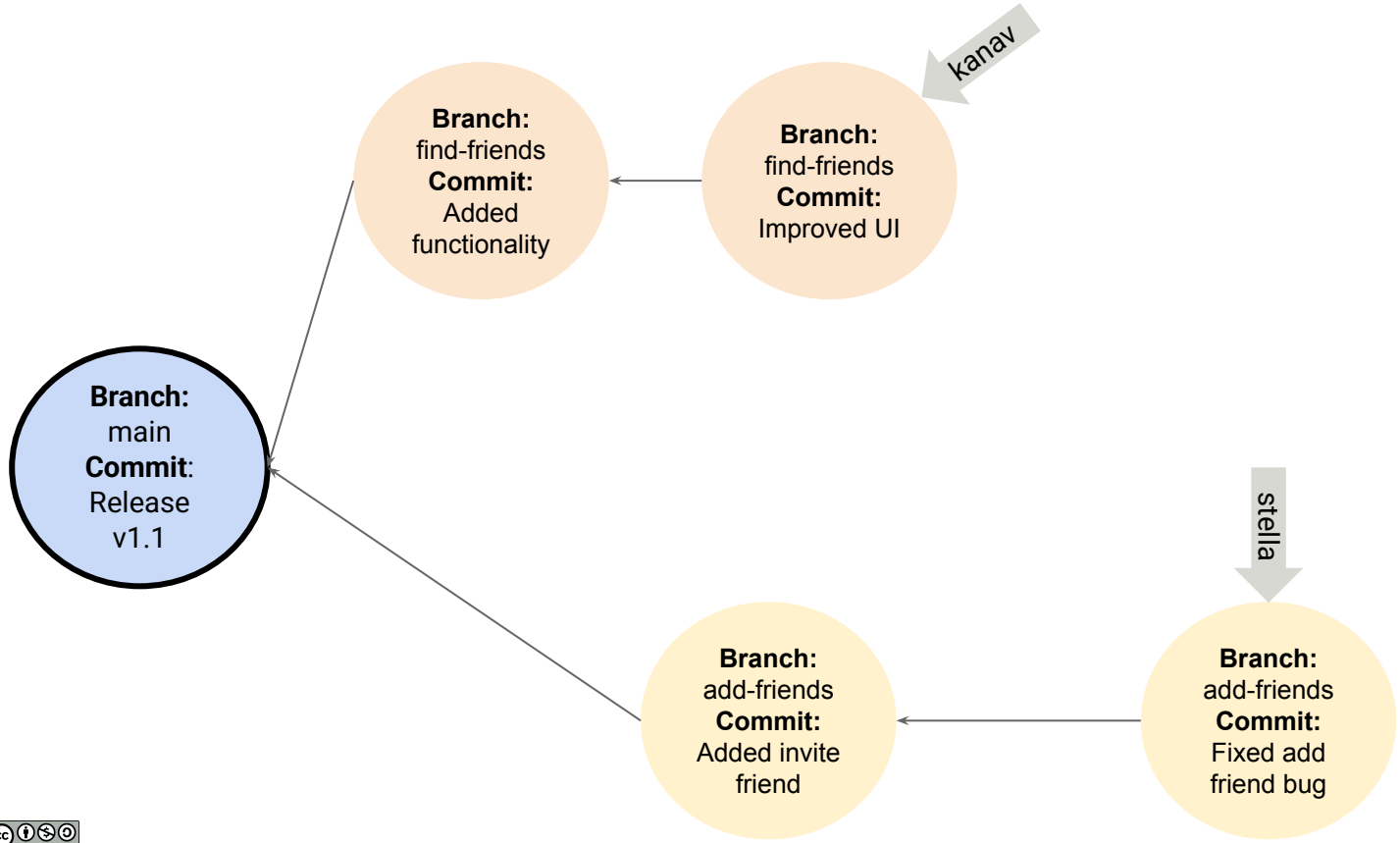
Feature branch demo with multiple developers

Kanav: merges `find-friends` branch into `main`



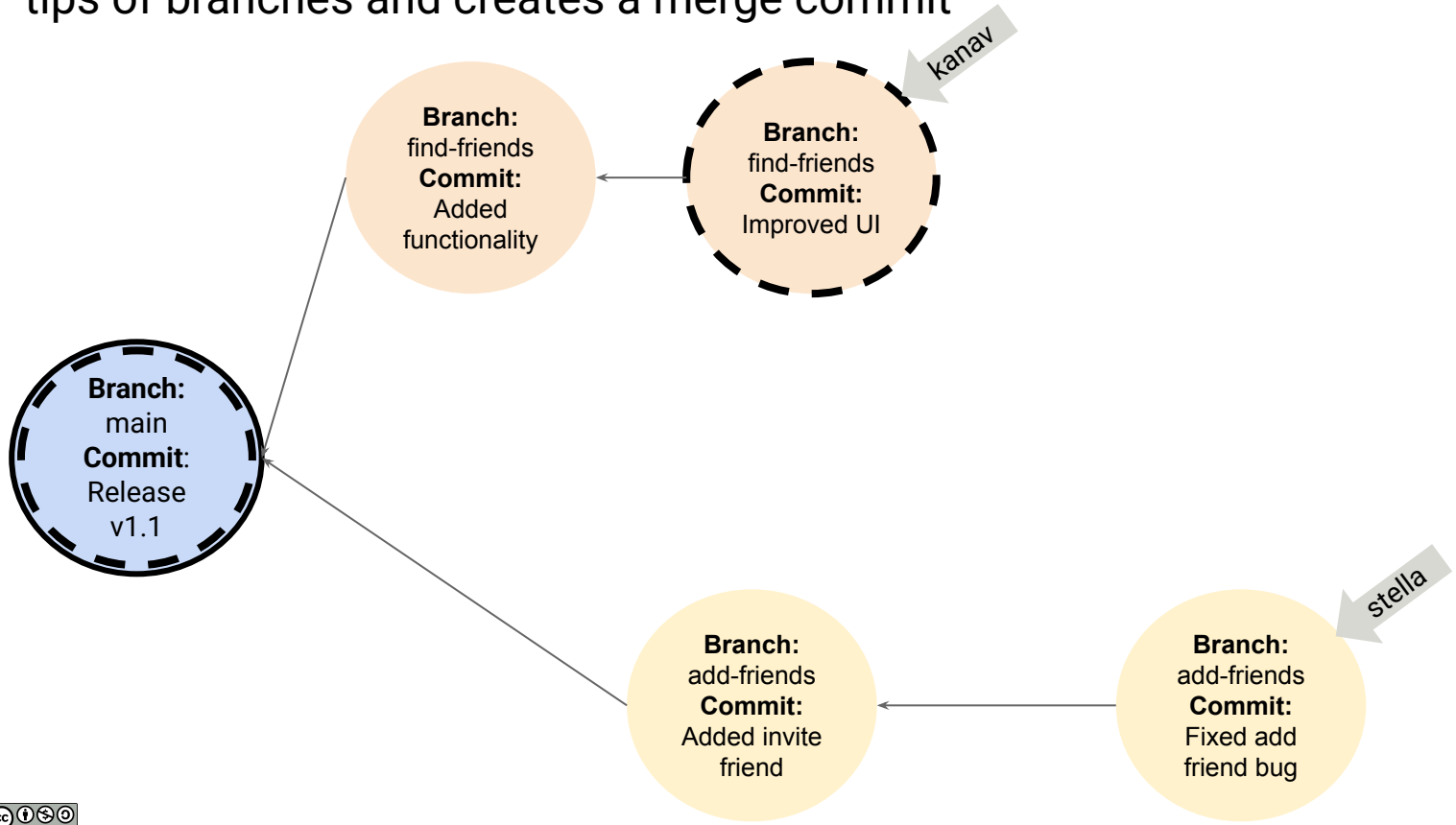
Feature branch demo with multiple developers

Step 1: Git identifies Closest Common Ancestor of `find-friends` and `main`



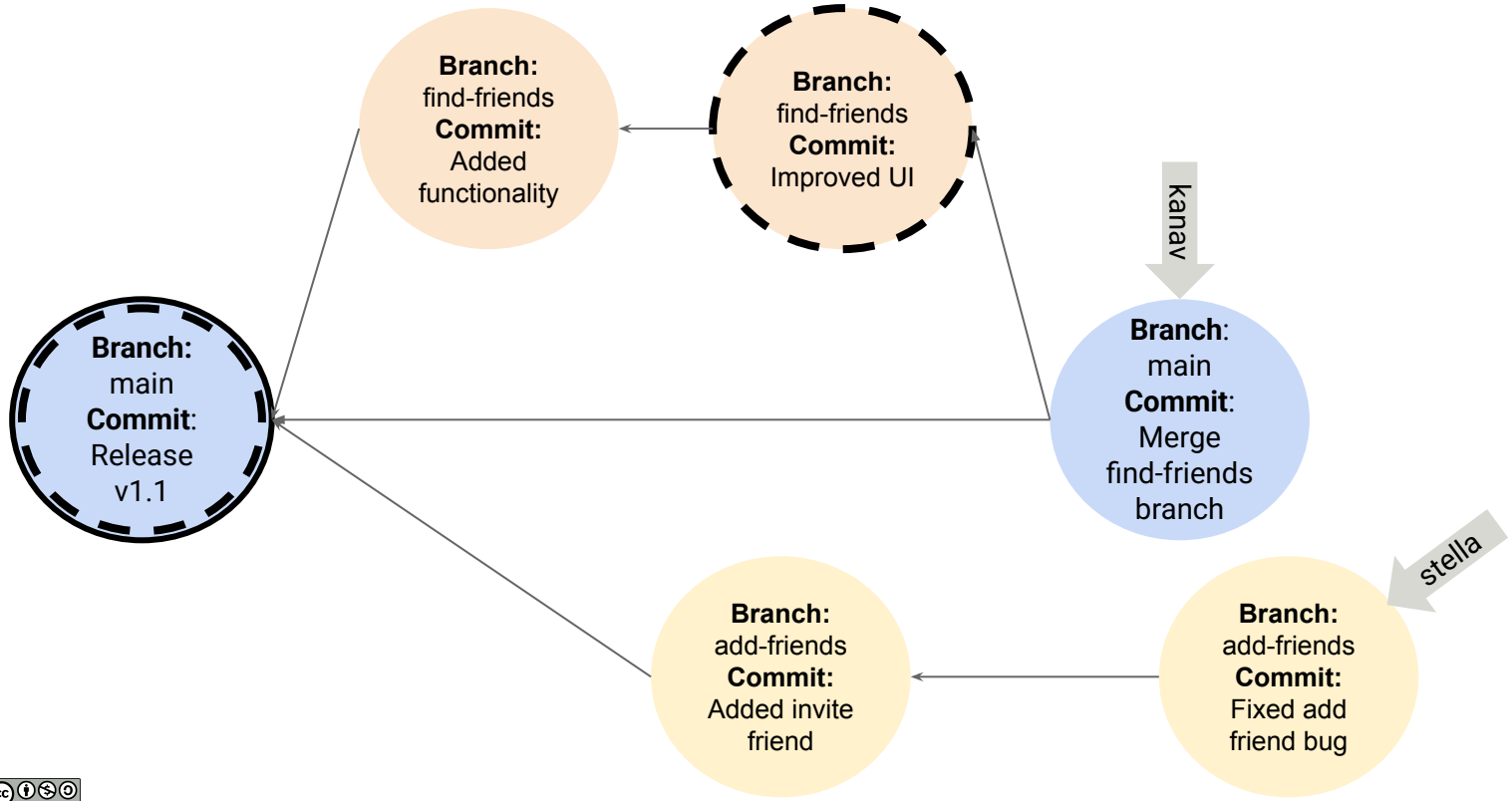
Feature branch demo with multiple developers

Step 2: Git combines changes made between Closest Common Ancestor and the tips of branches and creates a merge commit



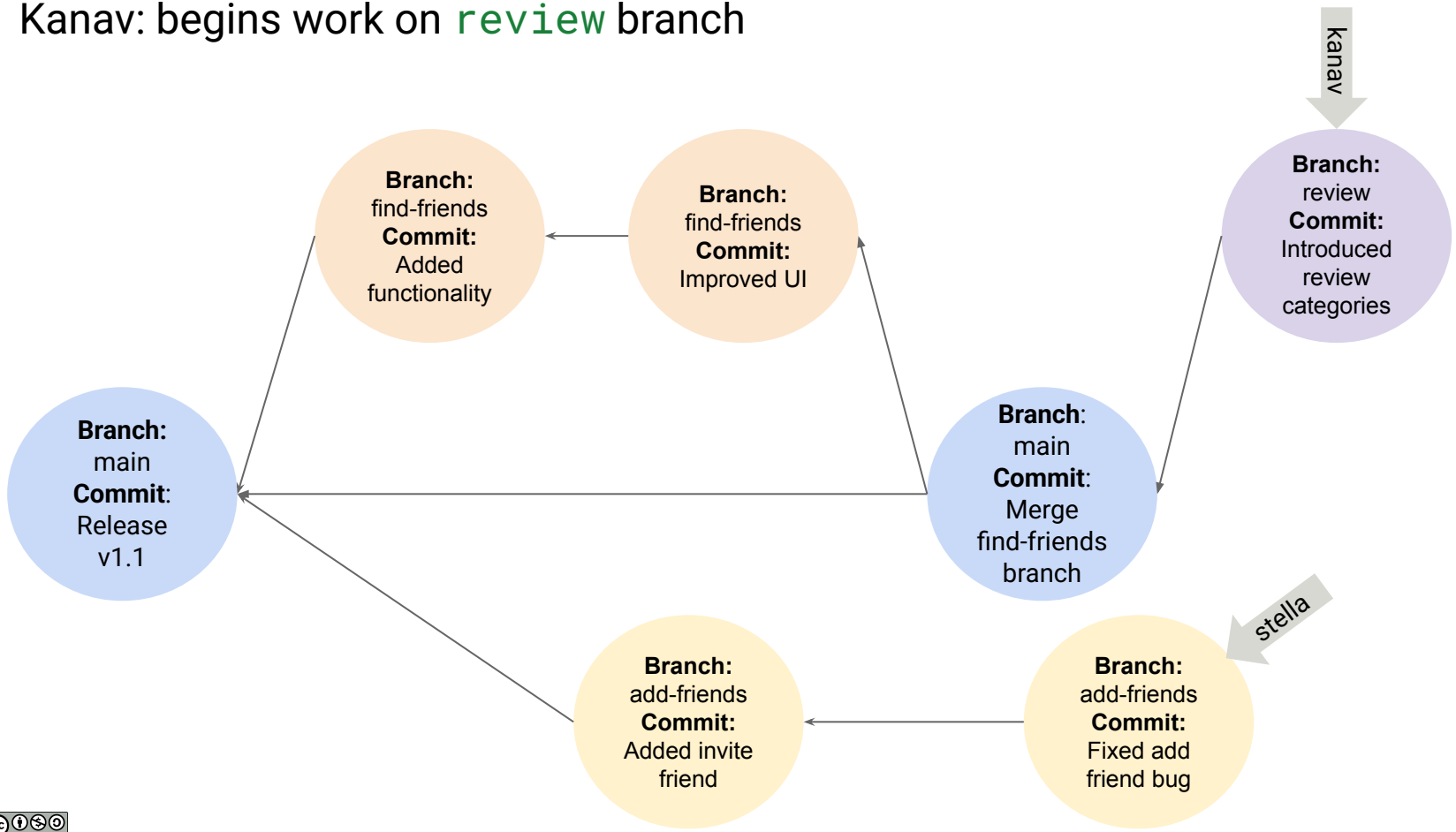
Feature branch demo with multiple developers

Step 2: Git combines changes made between Closest Common Ancestor and the tips of branches and creates a merge commit



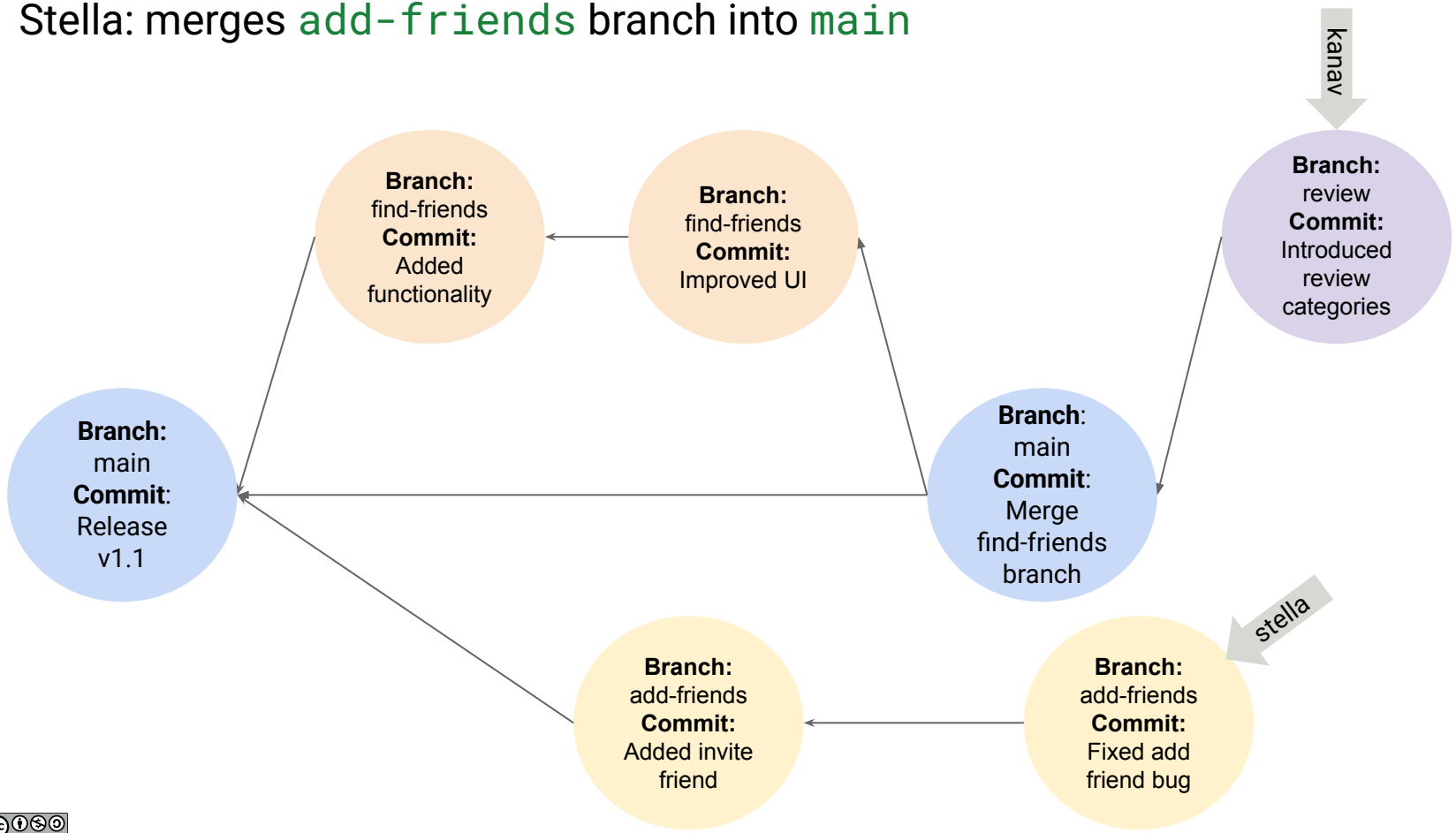
Feature branch demo with multiple developers

Kanav: begins work on `review` branch



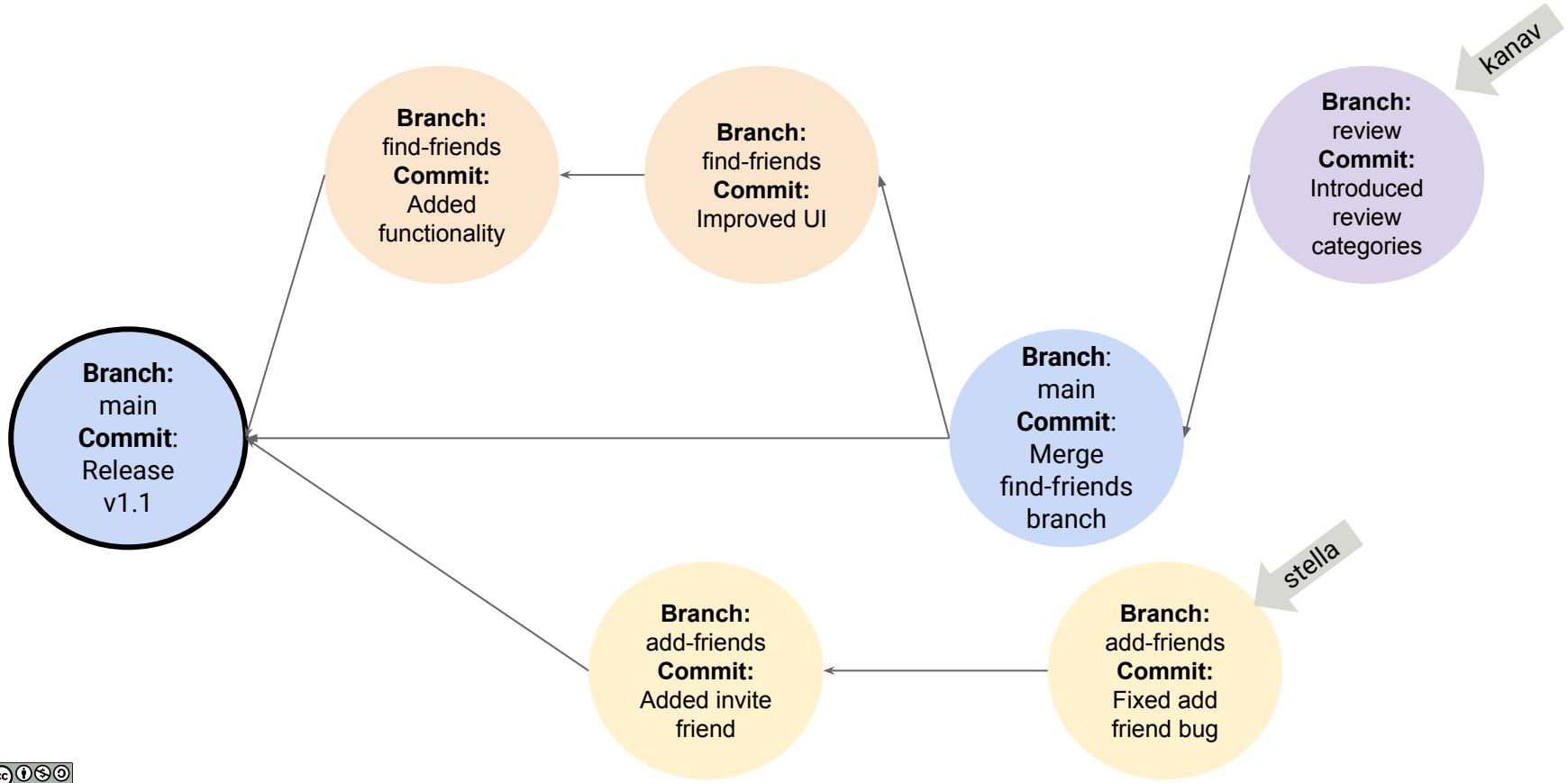
Feature branch demo with multiple developers

Stella: merges `add-friends` branch into `main`



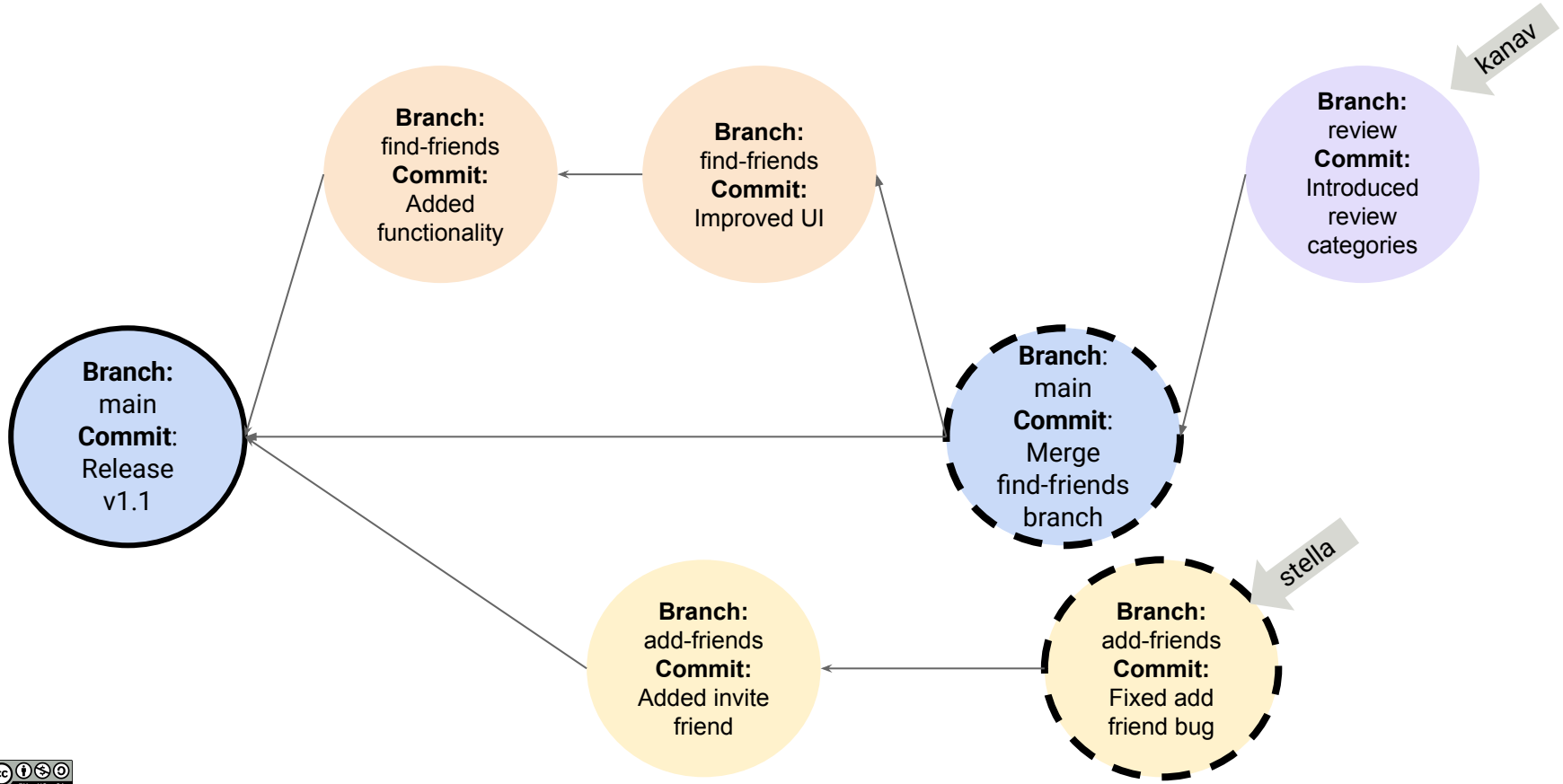
Feature branch demo with multiple developers

Step 1: Git identifies Closest Common Ancestor of `add-friends` and `main`



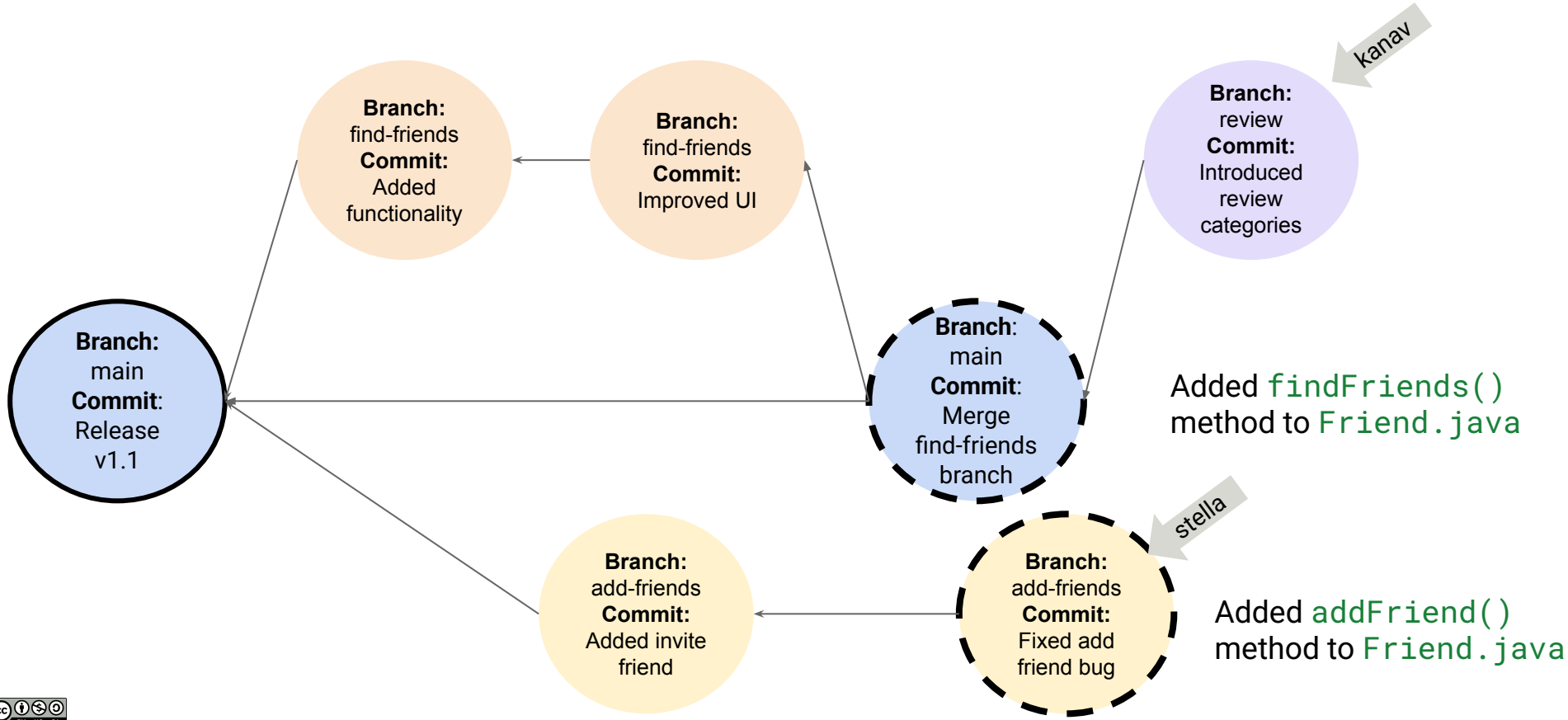
Feature branch demo with multiple developers

Step 2: Git combines changes made since the Closest Common Ancestor and…



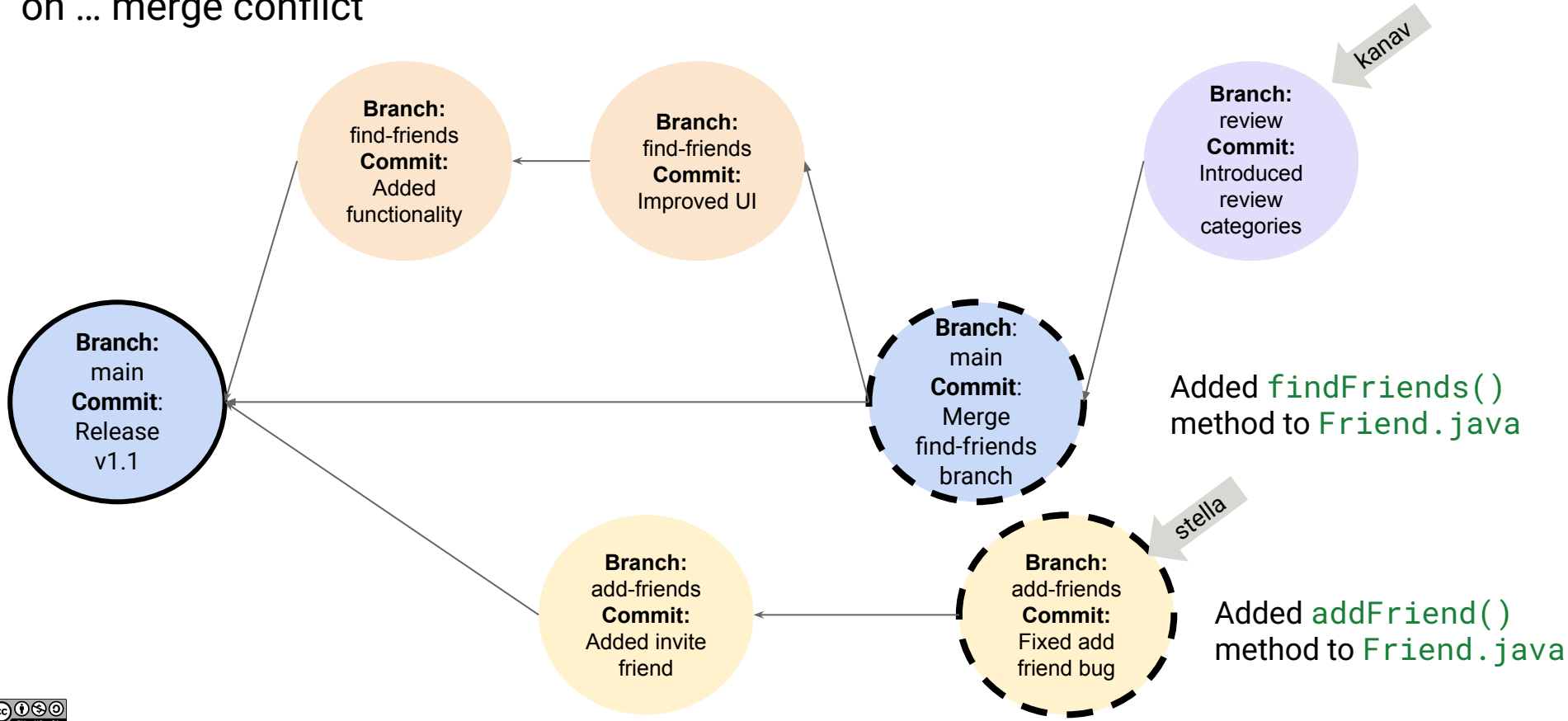
Feature branch demo with multiple developers

Step 2: Git combines changes made since the Closest Common Ancestor and…

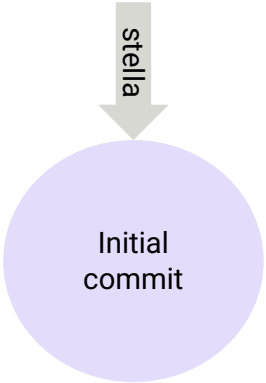


Feature branch demo with multiple developers

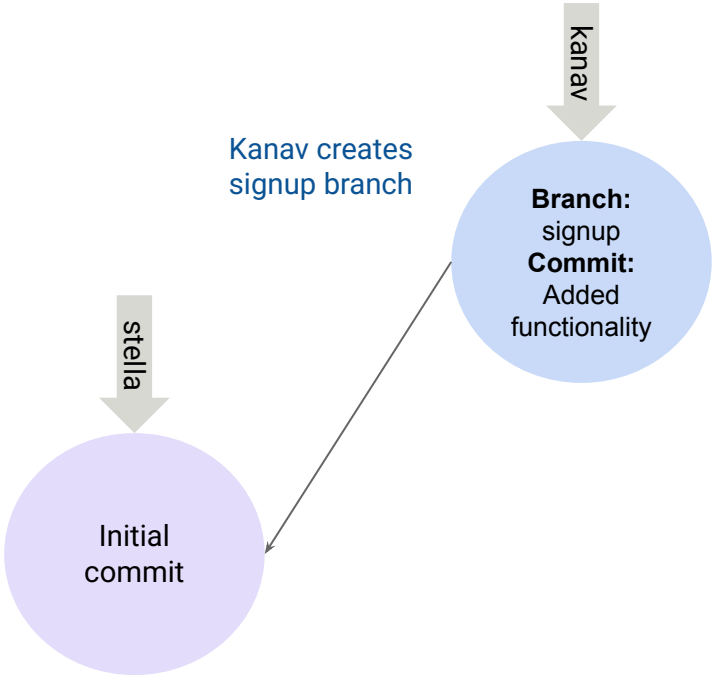
Step 2: Git combines changes made since the Closest Common Ancestor and… uh oh … merge conflict



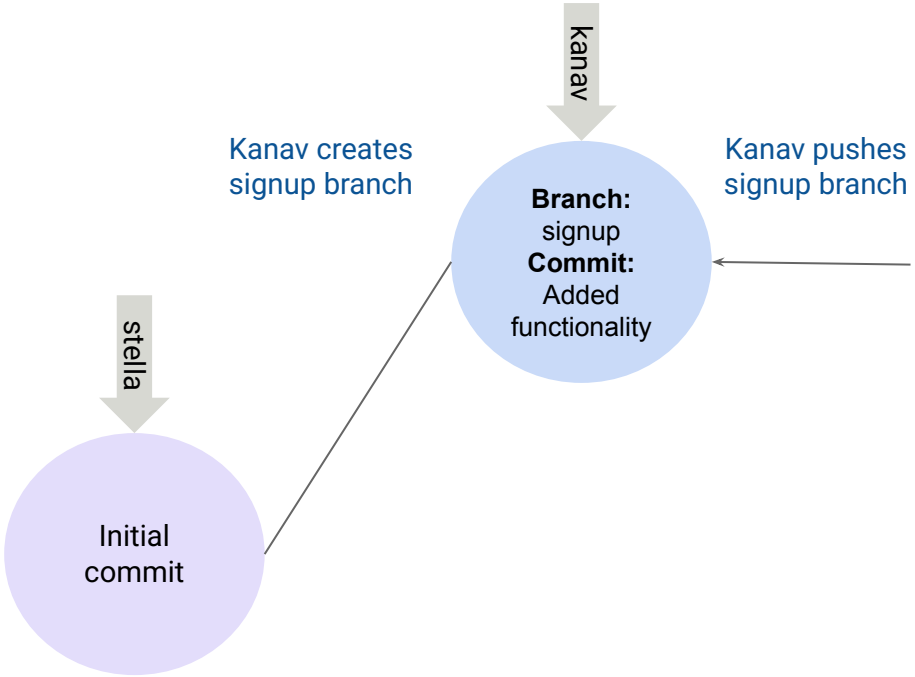
Collaboration on feature branches demo



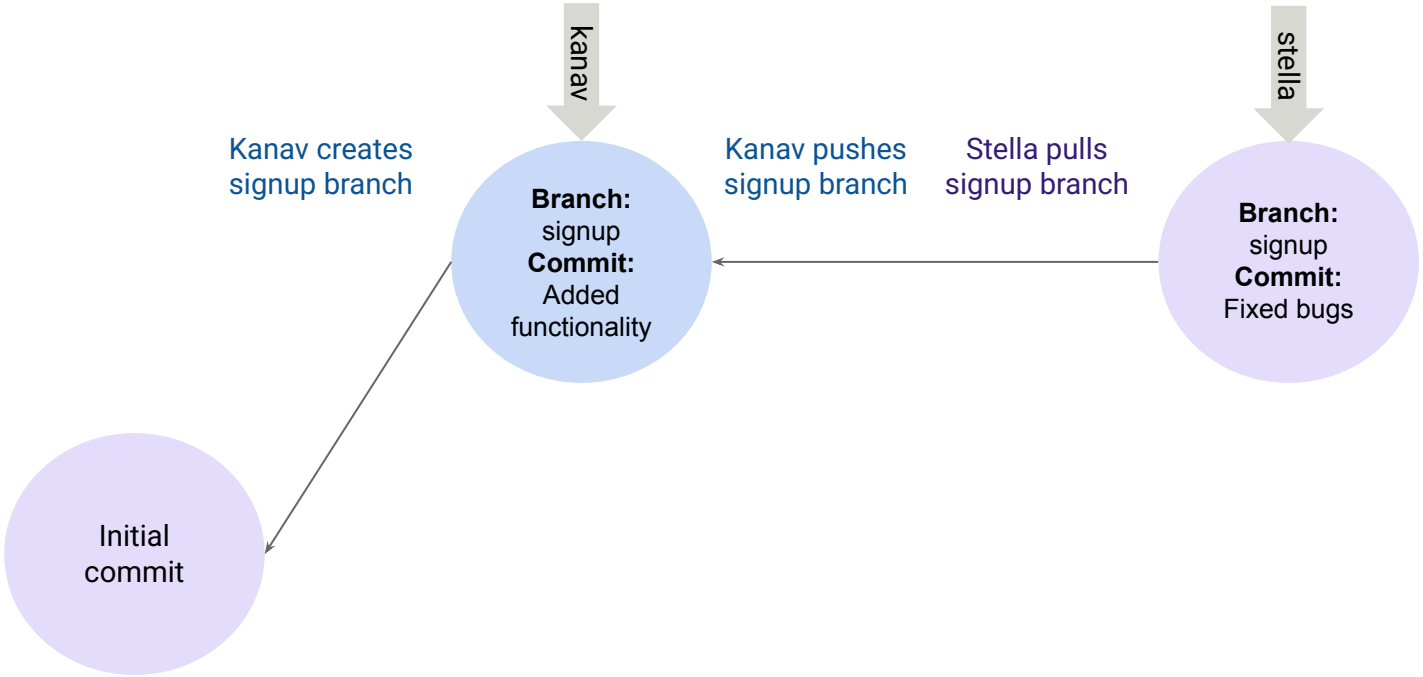
Collaboration on feature branches demo



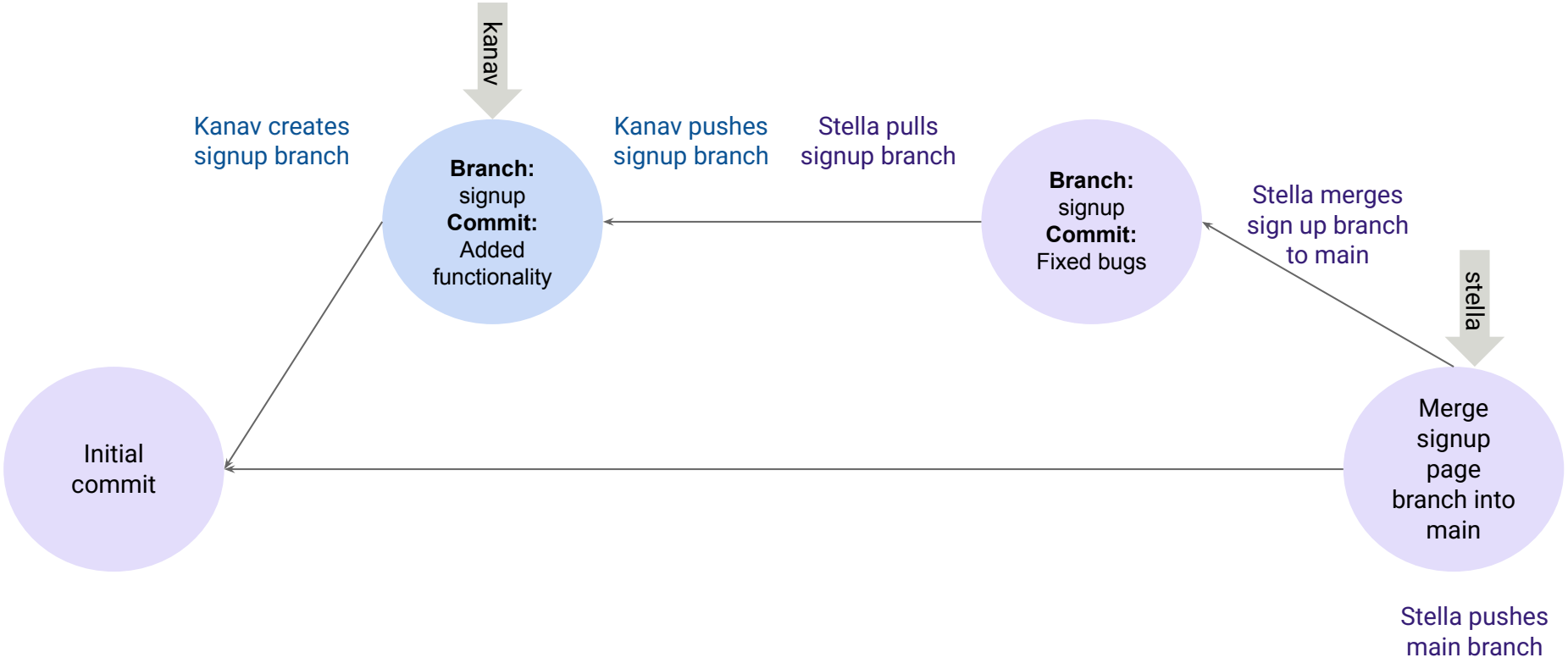
Collaboration on feature branches demo



Collaboration on feature branches demo

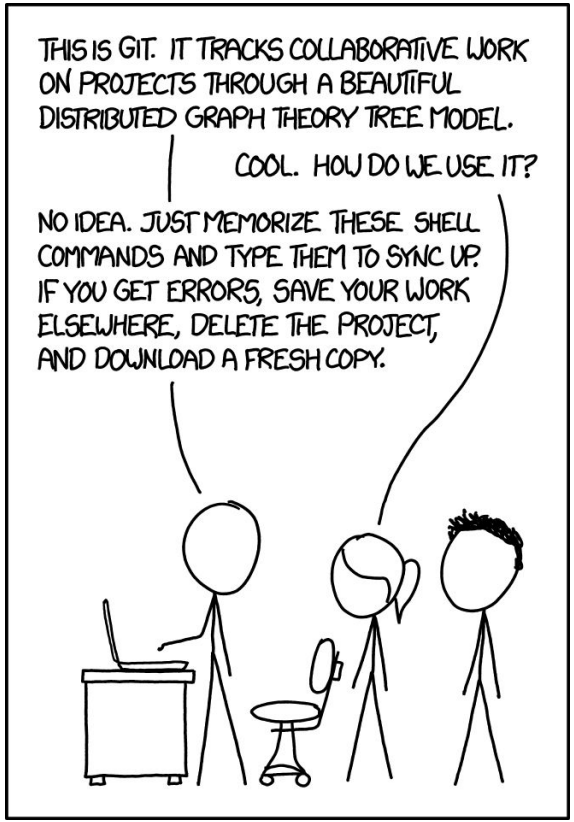


Collaboration on feature branches demo



General advice with Git

- Be careful with Git commands
  - Use `git status` and `git log` to check status of commits and history
  - Beware “shortcuts”
    i. `git add .` or `git add *`
    ii. `git push --force`
- Google is your friend. There are a lot of tips online about Git
- If Git terminal commands fail, Git will often tell you how to fix them
  - Don’t just blindly follow, understand what they are saying
  - Again, use Google to help you



https://xkcd.com/1597/

Pull Requests

Lecture 31, CS61B Spring 2025

Main branch

- The `main` branch is often reserved for deployment
  - Deployment is the process of making software available for use by users
  - CI/CD pipelines use the `main` branch to test and deploy

Main branch

- The `main` branch is often reserved for deployment
  - Deployment is the process of making software available for use by users
  - CI/CD pipelines use the `main` branch to test and deploy
- What does this mean?
  - We don't want development to occur on the `main` branch
  - Need safeguards to ensure that we only merge *complete, functional, and well-tested* changes to `main`
  - Paranoia!

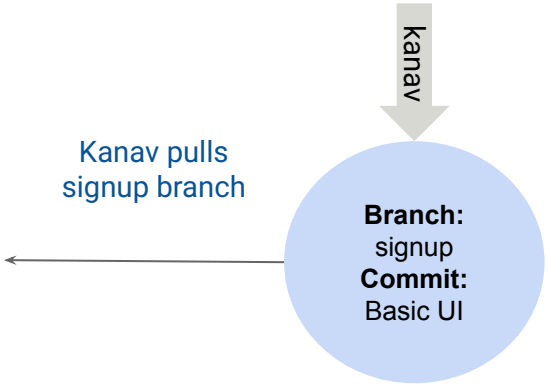
Pull requests

- *How do we ensure that we only merge complete, functional, and well-tested changes?*
- **Pull request (PR)**: way to request that your changes be merged into `main`
  - Developers can review your changes, leave feedback, and approve the request
  - Good practice for others to review your code before it's merged!
  - Most major tech companies mandate that at least one additional person review your code

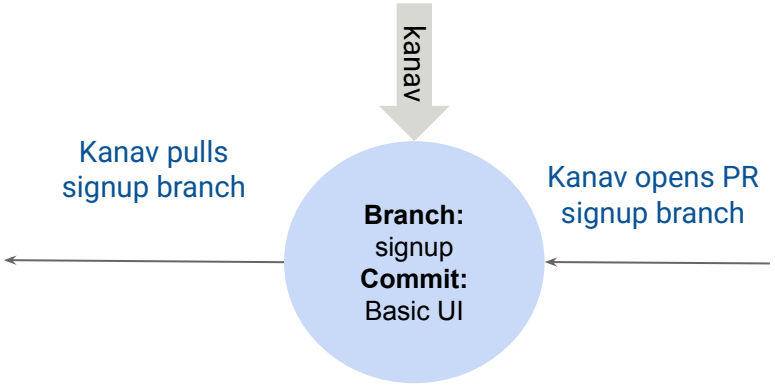
Pull requests

- *How do we ensure that we only merge complete, functional, and well-tested changes?*
- **Pull request (PR)**: way to request that your changes be merged into `main`
  - Developers can review your changes, leave feedback, and approve the request
  - Good practice for others to review your code before it's merged!
  - Most major tech companies mandate that at least one additional person review your code
- You can contribute to open-source projects (e.g. Firefox) by making a pull request
  - A maintainer will review your changes and provide feedback

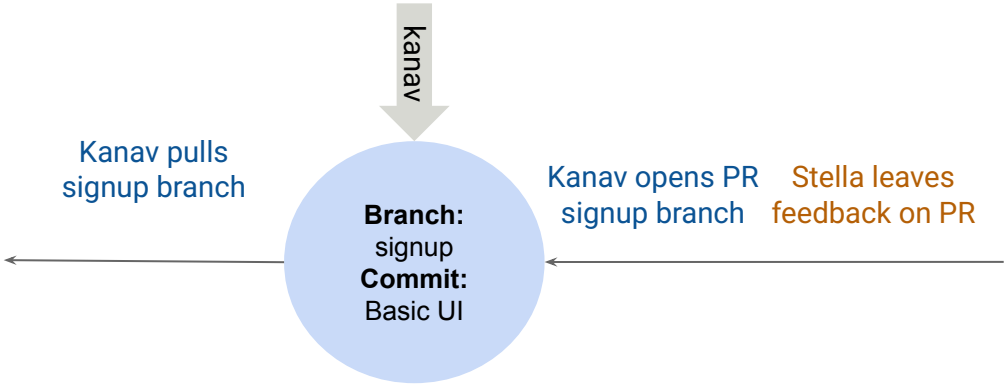
Pull requests example



Pull requests example

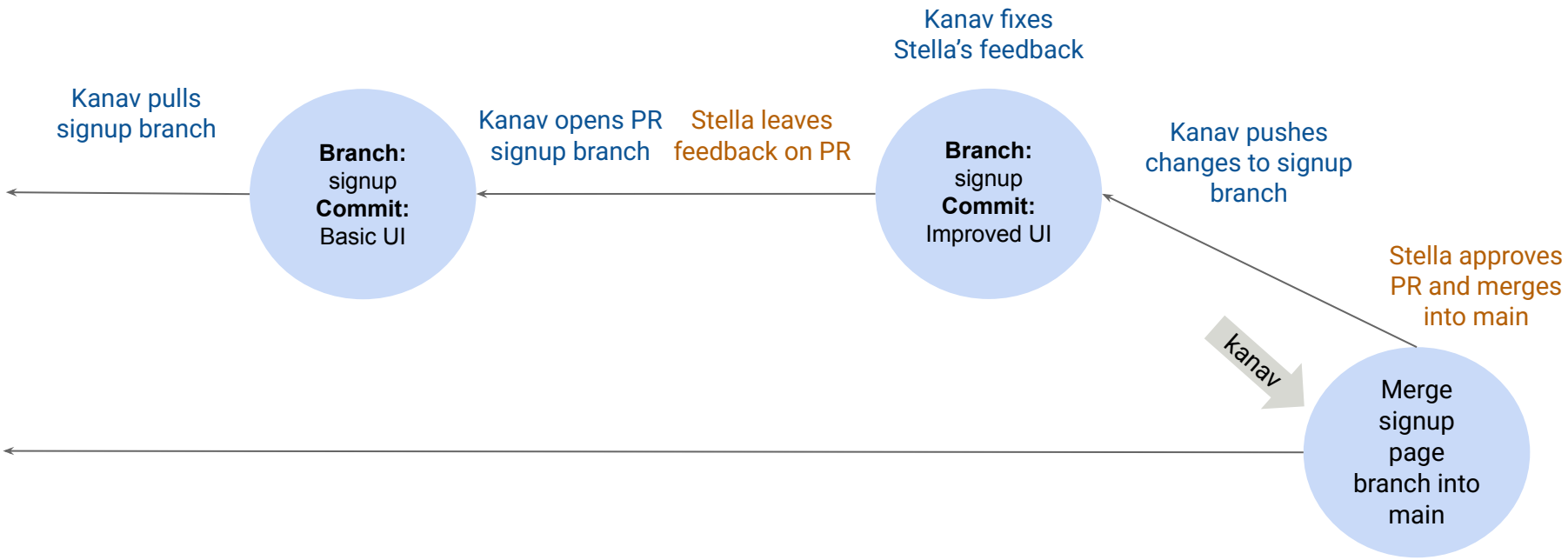


Pull requests example



Pull requests example

Pull requests example



Writing good PRs

- The goal of a PRs is to get feedback on your code
  - Be explicit about what feedback you want
  - @mention individuals involved
  - Keep PR descriptions and commit messages professional and informative
- Important: keep the code changes in a PR small
  - Easier to review a small code change than a big one!
  - Average PR size is 50-200 lines of code (GitHub)
- Storytime

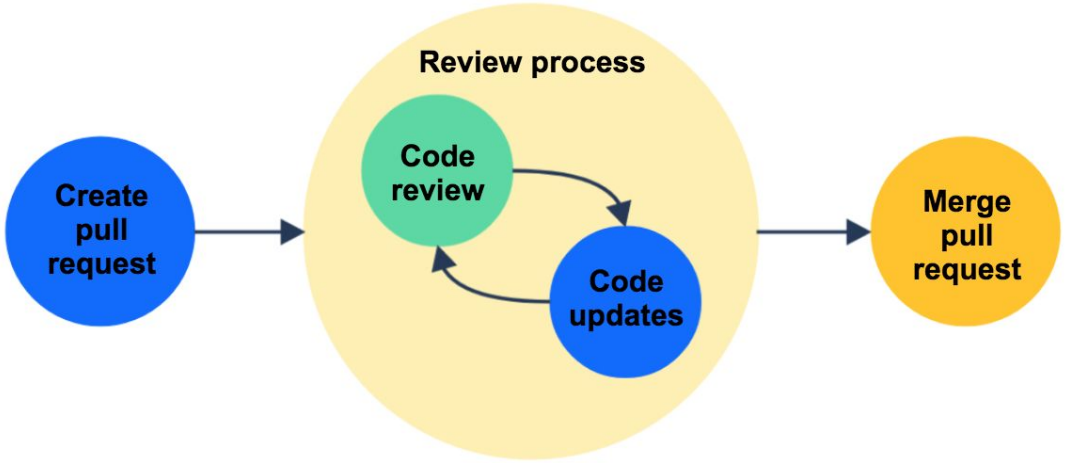
```
## What?
## Why?
## How?
## Testing?
## Screenshots (optional)
## Anything Else?
```

GitHub PR template in Markdown

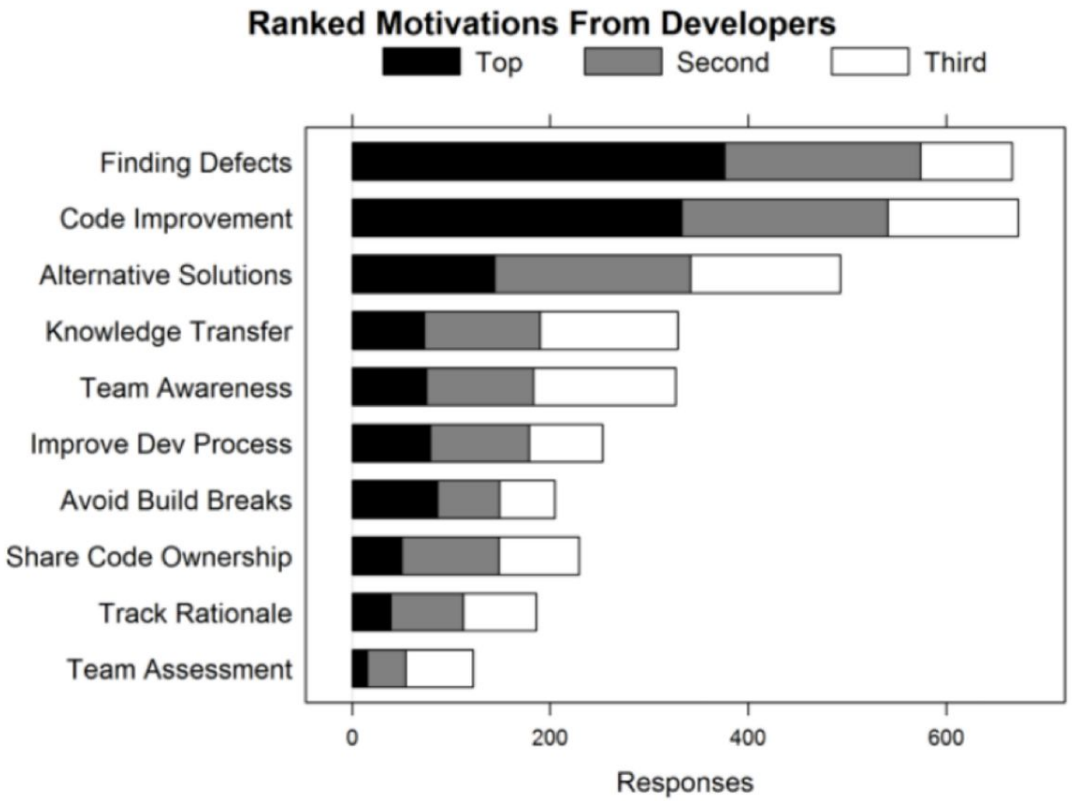
Code reviews

Key part of pull requests: someone can leave feedback on your code! This is called a **code review**.

- Reviewers can comment on specific lines of code
- Can cover both high-level concerns and low-level issues
- Reviewers provide actionable suggestions for improvement
- Back-and-forth discussion between author and reviewers
- Author can push updates to address feedback, PR automatically refreshes



Why Code Review?



Bacchelli, Alberto and Christian Bird. "Expectations, outcomes, and challenges of modern code review." Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013.

GitHub Pull Request Demo

Reviewer assigned

PR description

Added signup page #4

Edit | Code

Open | kanavmittal314 wants to merge 1 commit into `main` from `kanav/signup-page`

Conversation 0 | Commits 1 | Checks 0 | Files changed 1 | +13 −0

kanavmittal314 commented now — Owner

Created `SignupPage.java` class. Includes username and password.

signup page added — 469928e

kanavmittal314 requested a review from **stellakaval** now

Review requested
Review has been requested on this pull request. It is not required to merge.
Learn more about requesting a pull request review.
Show all reviewers

1 pending reviewer

Require approval from specific reviewers before merging
Rulesets ensure specific people approve pull requests before they're merged.
Add rule

Continuous integration has not been set up
GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request | You can also open this in GitHub Desktop or view command line instructions.

Reviewers
stellakaval
Still in progress? Convert to draft

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

Development
Successfully merging this pull request may close these issues.
None yet

Notifications — Customize
Unsubscribe
You're receiving notifications because you're watching this repository.

GitHub Pull Request Demo

Conversation 0 | Commits 1 | Checks 0 | Files changed 1 | +13 −0

kanavmittal314 commented 3 minutes ago — Owner

Created `SignupPage.java` class. Includes username and password.

signup page added — 469928e

kanavmittal314 requested a review from **stellakaval** 3 minutes ago

stellakaval commented 1 minute ago — Collaborator

You need to implement the `addAccount` method to finish this implementation. Great work so far.

stellakaval requested changes now — View reviewed changes

SignupPage.java

```
+        }
+
+        public void addAccount() {
+                # TODO
```

stellakaval now — Collaborator

You left this as a TODO, please complete so I can approve.

Reply...

Resolve conversation

Reviewers
stellakaval

Still in progress? Convert to draft

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

Development
Successfully merging this pull request may close these issues.
None yet

Notifications — Customize
Unsubscribe
You're receiving notifications because you're watching this repository.

1 participant

Lock conversation

Reviewer comments

GitHub Pull Request Demo

Added signup page #4

Edit | Code

Open **kanavmittal314** wants to merge 2 commits into `main` from `kanav/signup-page`

Conversation 3 | Commits 2 | Checks 0 | Files changed 1 | +16 −0

Changes from 1 commit | File filter | Conversations | Jump to | 0 / 1 files viewed | Review in codespace | Review changes

5 SignupPage.java | Viewed

```
@@ -1,13 +1,16 @@
+ import java.util.*;
+
  public class SignupPage {
          private String username;
          private String password;
+         private static Map<String, String> accounts = new HashMap<>();

          public SignupPage(String username, String password){
                  this.username = username;
                  this.password = password;
          }

          public void addAccount() {
-                 # TODO
+                 accounts.put(this.username, this.password);
          }
  }
```

Updated PR lines shown in green, removed in red, auto refresh update once pushed!

GitHub Pull Request Demo

Open **Added signup page** #4
kanavmittal314 wants to merge 1 commit into `main` from `kanav/signup-page`

stellakaval requested changes 4 minutes ago — View reviewed changes

`SignupPage.java` Outdated — Show resolved

Finished `addAccount` method in `SignupPage.java` — `ec733f1`

stellakaval approved these changes now — View reviewed changes

stellakaval left a comment — Collaborator

Great work, thanks.

Final reviewer comment

Changes approved — Show all reviewers
1 approving review Learn more about pull request reviews.

1 approval

Require approval from specific reviewers before merging — Add rule
Rulesets ensure specific people approve pull requests before they're merged.

Continuous integration has not been set up
GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request — You can also open this in GitHub Desktop or view command line instructions.

PR ready for merge with main

No milestone

Development
Successfully merging this pull request may close these issues.
None yet

Notifications — Customize
Unsubscribe
You're receiving notifications because you're watching this repository.

1 participant

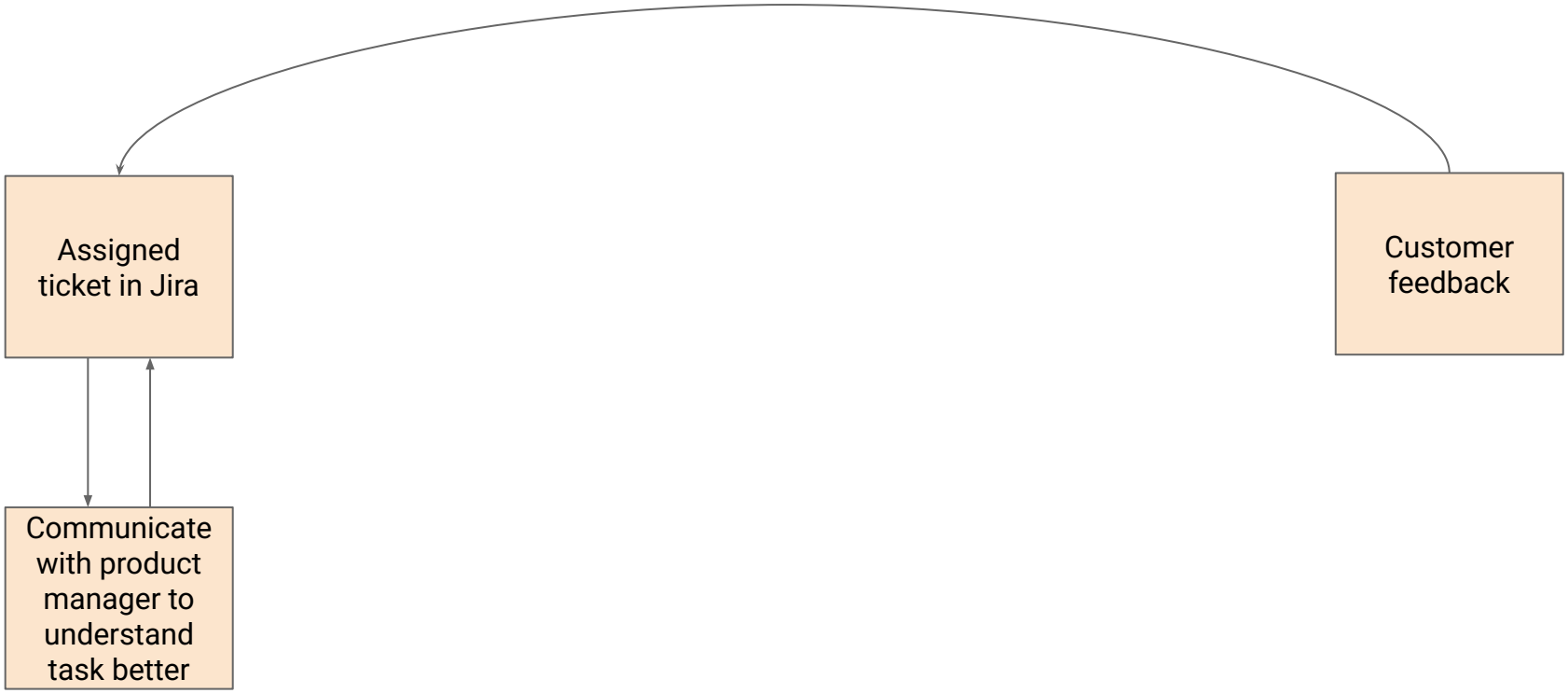
Lock conversation

Summary

Lecture 31, CS61B Spring 2025

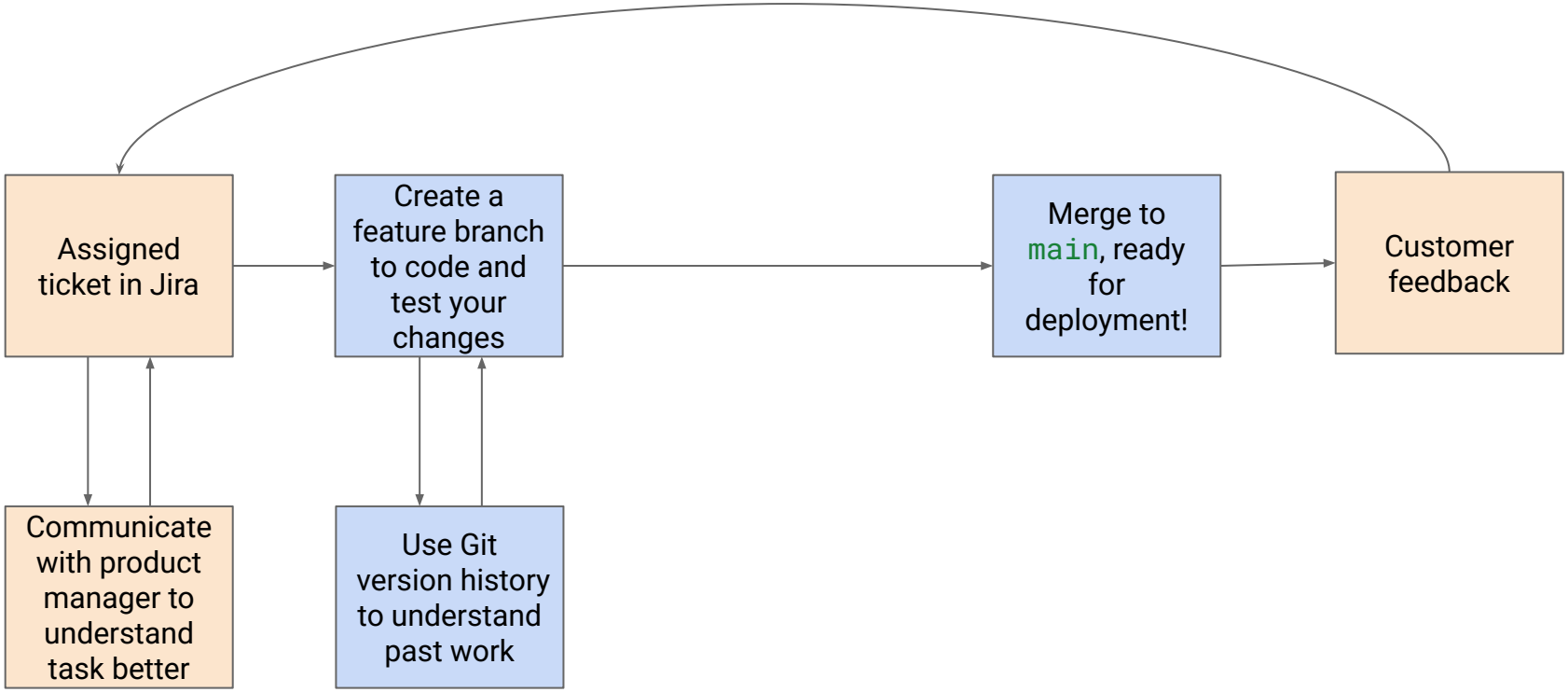
Summary (Git + Jira + Customer)

1. First, we talked about using Jira for organizing Agile/Scrum development



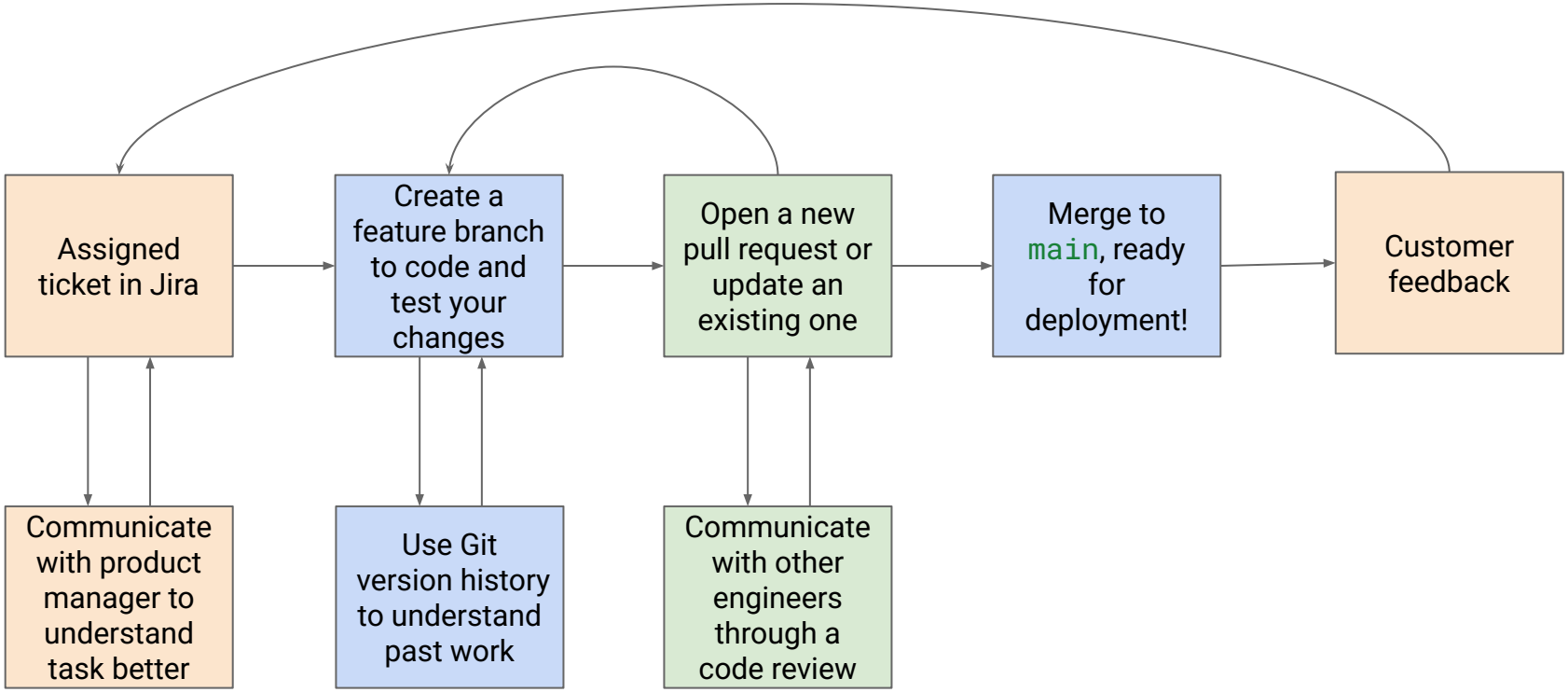
Summary (Git + Jira + Customer)

2. Second, we talked about git workflows like feature branches to develop these items

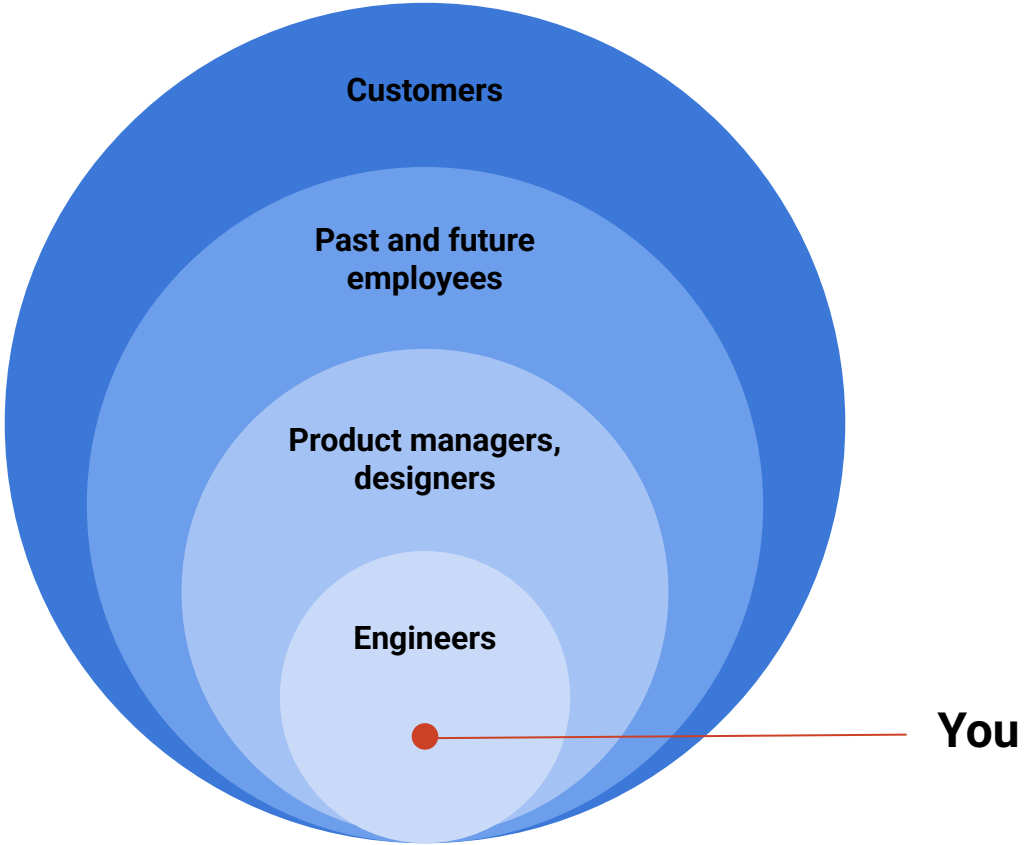


Summary (Git + Jira + Customer)

3. Third, we discussed how the PR and code review process was key for code quality and releasing to main



Collaboration



Summary (Collaboration is key)

Lessons: **Collaboration is key.** Opportunities to collaborate with customers, product managers, other engineers, and past/future employees exist throughout the process!