Version Control

Chung-Kil Hur

(Credit: Byung-Gon Chun & Many Slides from UCB CS169 taught by Armando Fox and David Patterson)

SWPP, CSE, SNU

Version Control:
Merge Conflicts, Effective Branching

Source & configuration management (SCM)

- What is it?
  - *Version* (snapshot) code, docs, config files, etc. at key points in time
  - Complete copy of every versioned file per snapshot
  - Implementation: deltas? complete file copy? symlink?
- Why do it?
  - Roll back if introduce bugs
  - Separate deployed from development version of code
  - Keep separate *branches* of development
  - Documented history of who did what when
  - Track what changed between revisions of a project

40 Years of Version Control

SCCS & RCS (1970s)

CVS (1986)

Subversion (2001)

Git (2005)

Image © TheSun.au

Social Coding

There is a really interesting group of people in the United States and around the world who do social coding now. The most interesting stuff is not what they do on Twitter, it's what they do on GitHub.

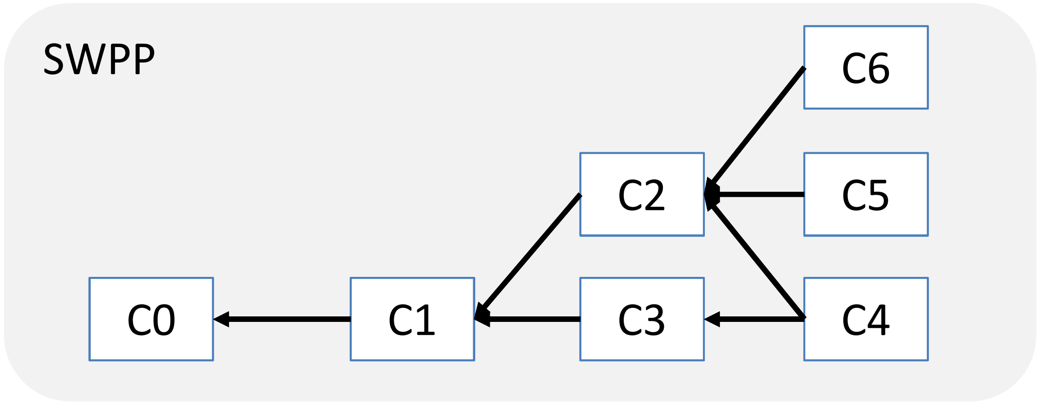
-- Al Gore, former US Vice President, 2013

github
SOCIAL CODING

Git

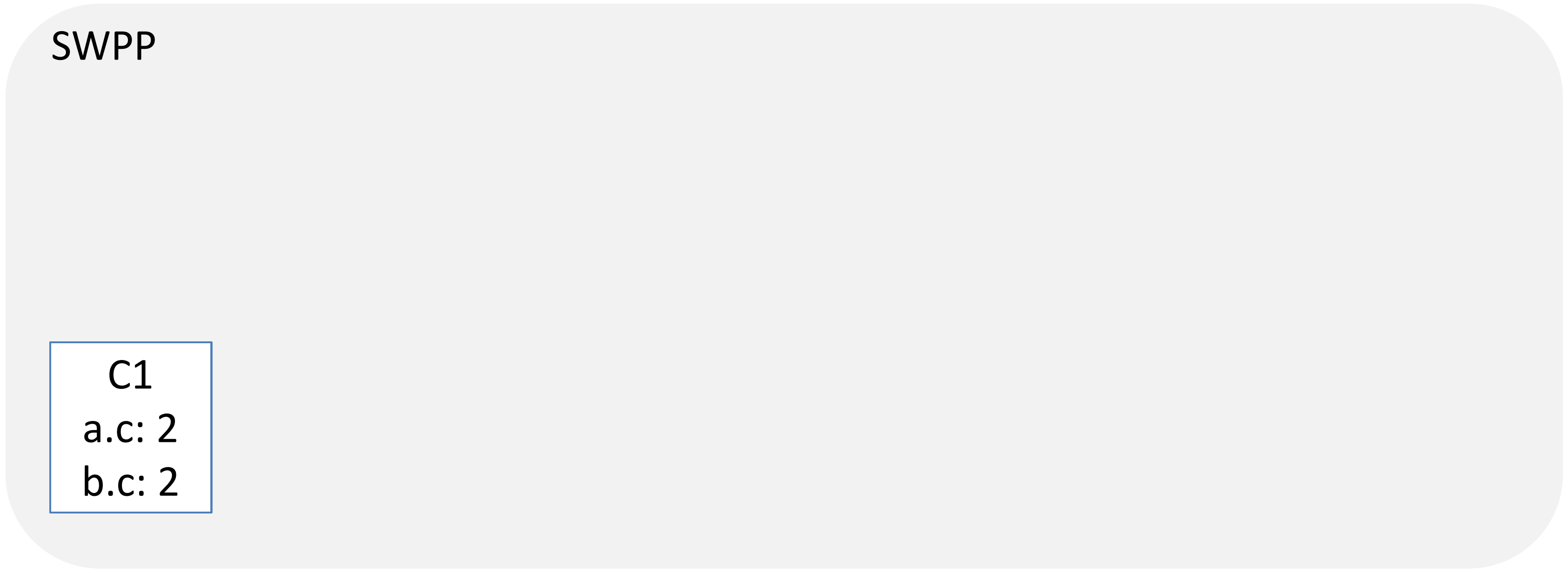
Repository

- A repository stores development histories (a rooted connected directed acyclic graph)
- Node (called Commit): A snapshot of development at some point
- Edge: Point to its parent nodes



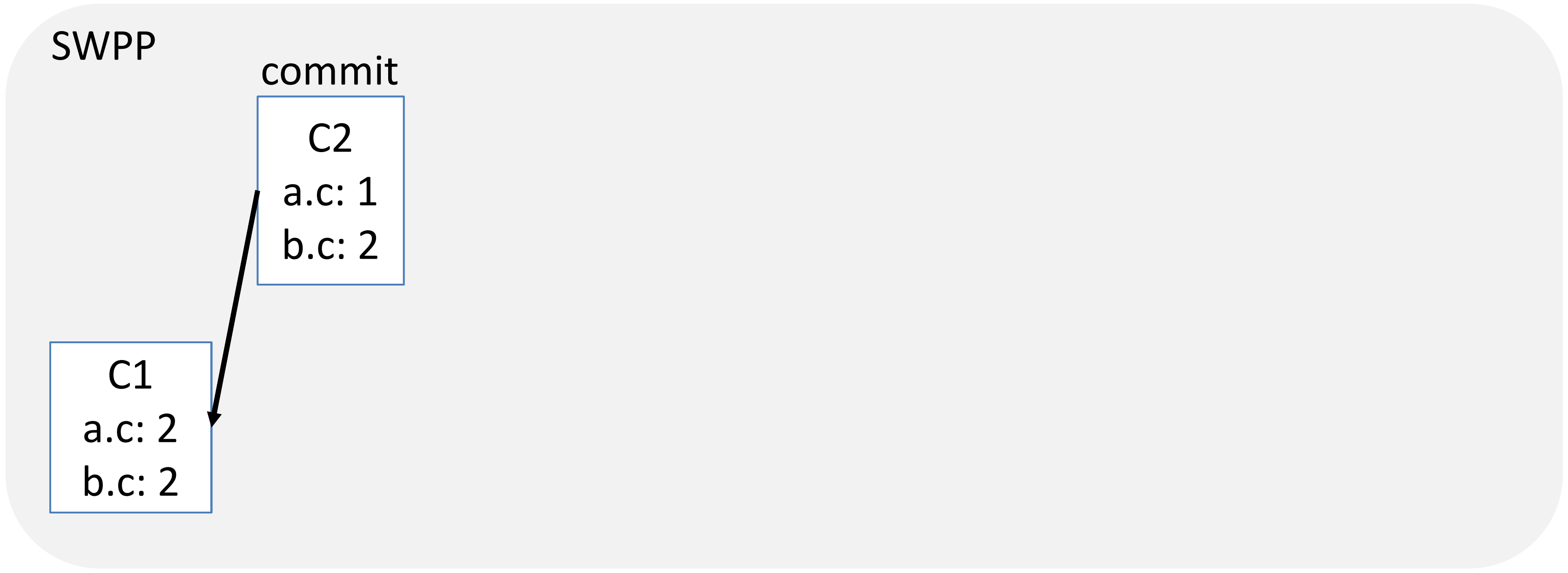
Basic Operations

- Commit:
  Add a fresh commit with a new development
- Merge (two parents):
  Add a commit by merging two commits
- Rebase:
  Add commits by applying existing changes to a commit



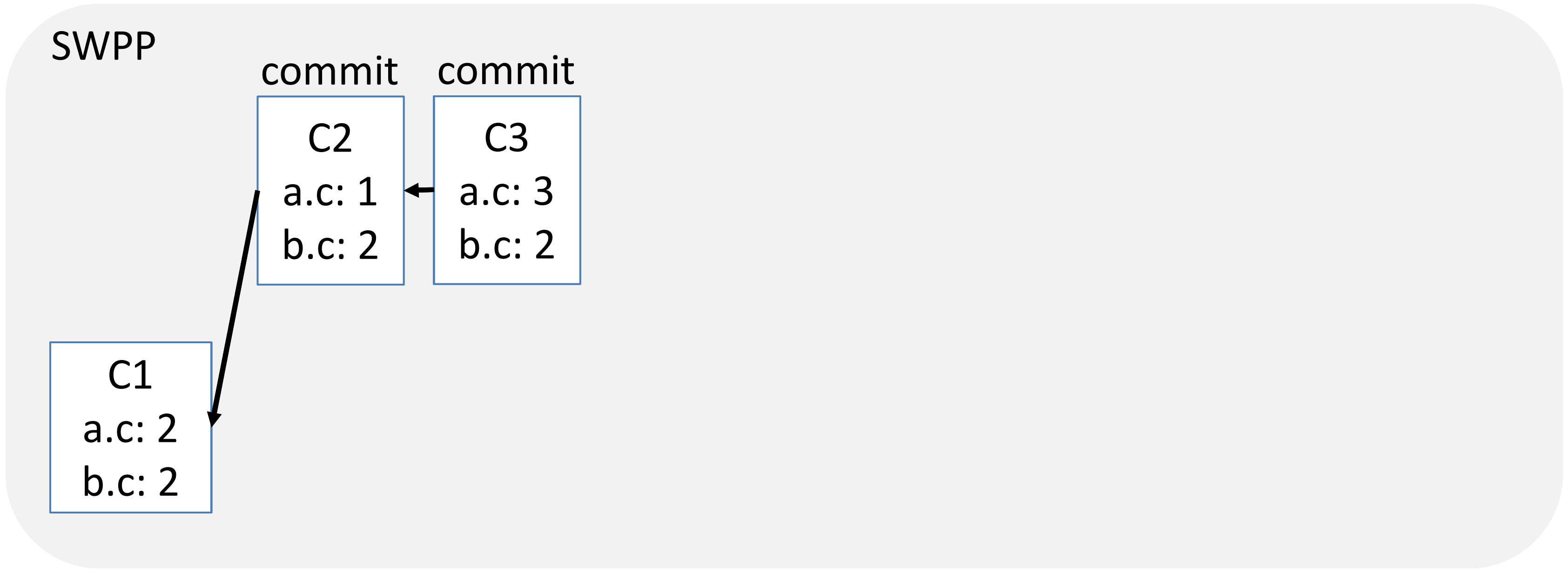
Basic Operations

- Commit:
  Add a fresh commit with a new development
- Merge (two parents):
  Add a commit by merging two commits
- Rebase:
  Add commits by applying existing changes to a commit



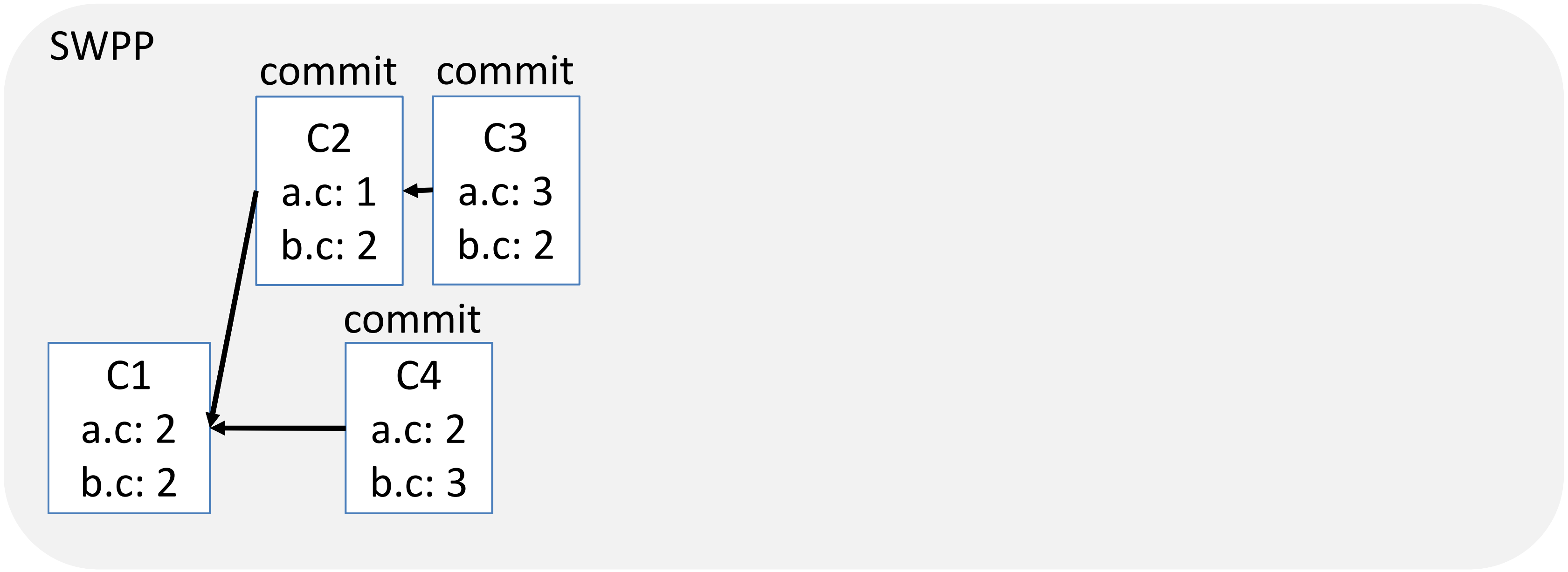
Basic Operations

- Commit:
  Add a fresh commit with a new development
- Merge (two parents):
  Add a commit by merging two commits
- Rebase:
  Add commits by applying existing changes to a commit



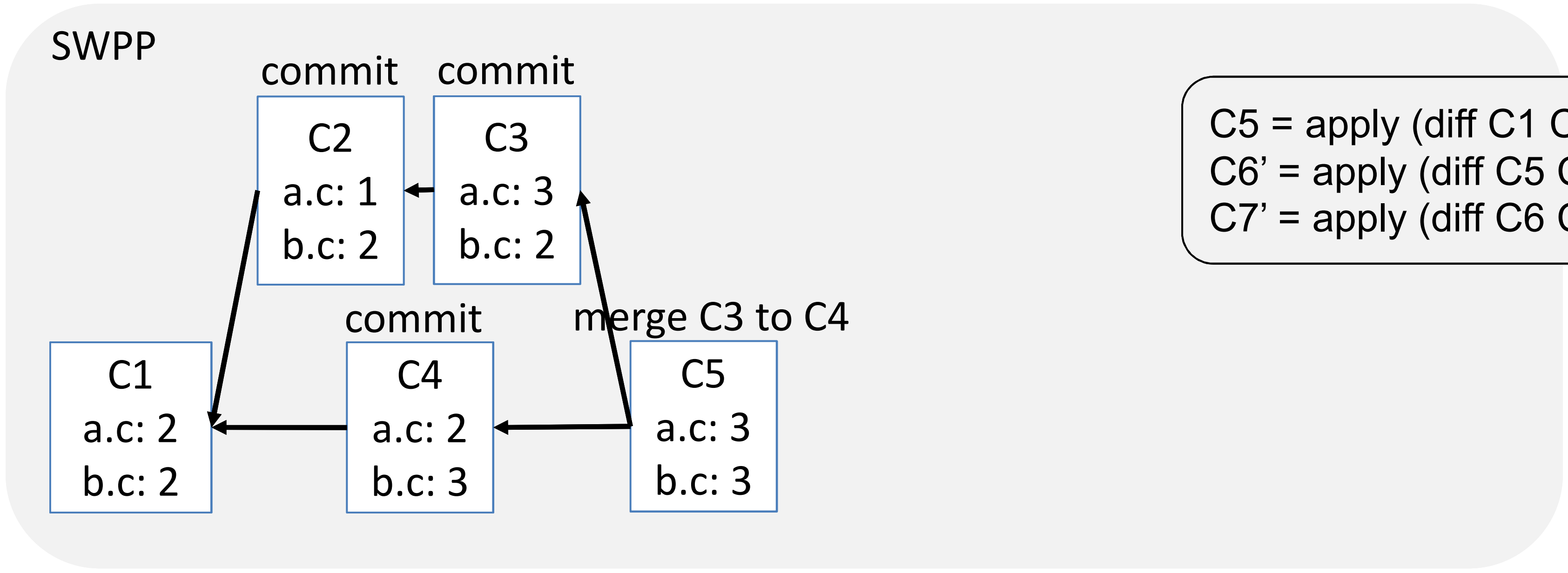
Basic Operations

- Commit:
  Add a fresh commit with a new development
- Merge (two parents):
  Add a commit by merging two commits
- Rebase:
  Add commits by applying existing changes to a commit



Basic Operations

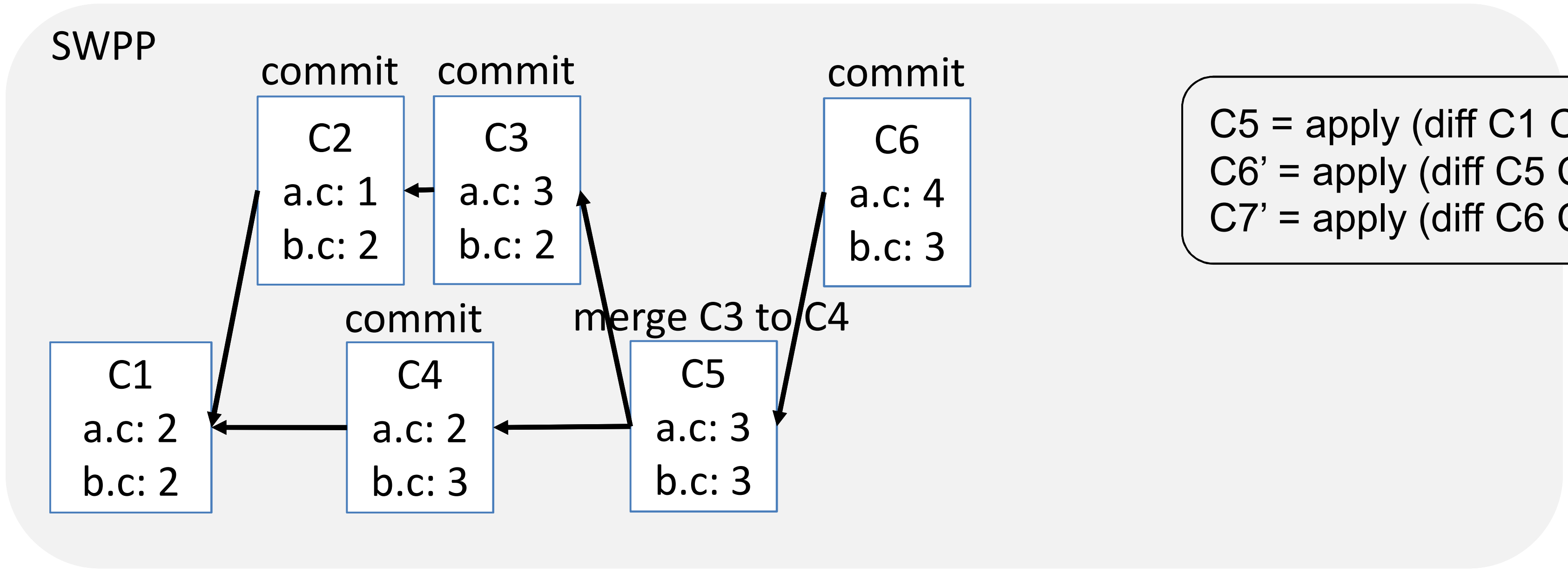
- Commit:
  Add a fresh commit with a new development
- Merge (two parents):
  Add a commit by merging two commits
- Rebase:
  Add commits by applying existing changes to a commit



C5 = apply (diff C1 C3) C4
C6’ = apply (diff C5 C6) C8
C7’ = apply (diff C6 C7) C6’

Basic Operations

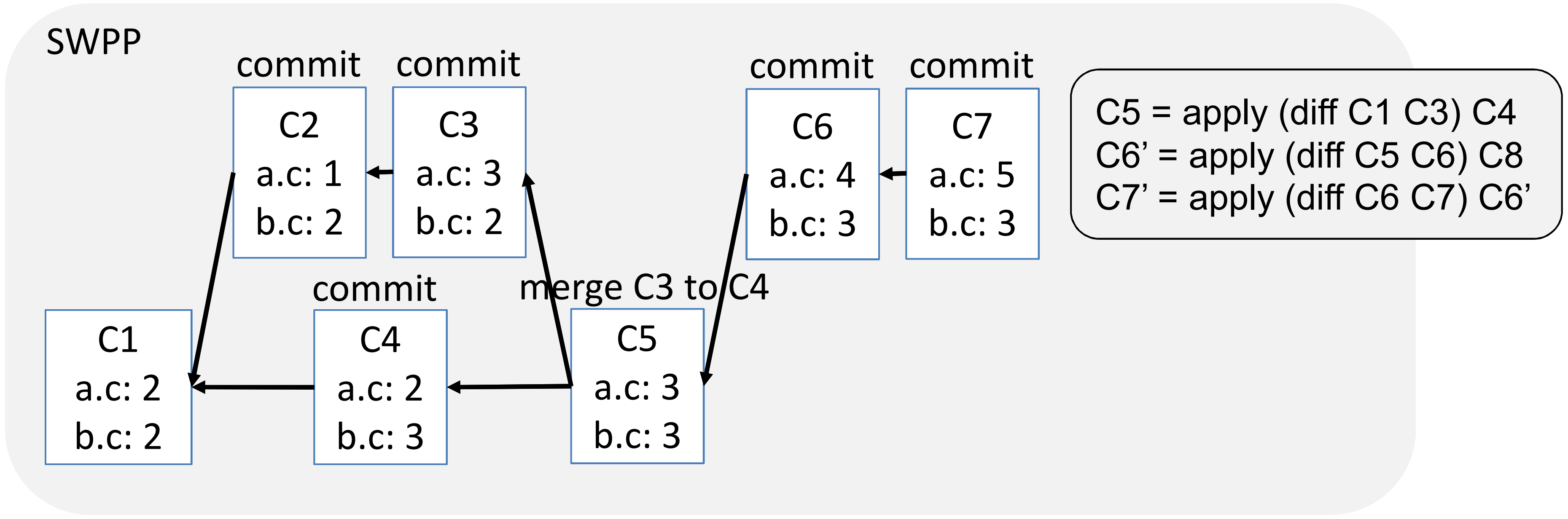
- Commit:
  Add a fresh commit with a new development
- Merge (two parents):
  Add a commit by merging two commits
- Rebase:
  Add commits by applying existing changes to a commit



C5 = apply (diff C1 C3) C4
C6’ = apply (diff C5 C6) C8
C7’ = apply (diff C6 C7) C6’

Basic Operations

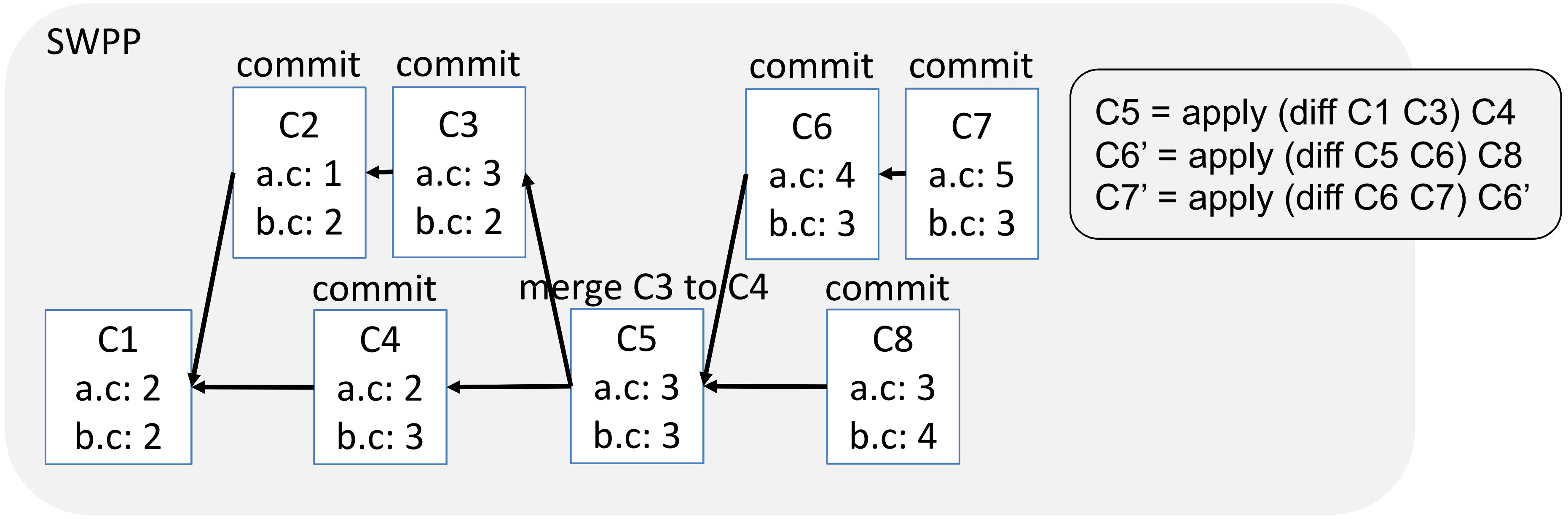
- Commit:
  Add a fresh commit with a new development
- Merge (two parents):
  Add a commit by merging two commits
- Rebase:
  Add commits by applying existing changes to a commit



C5 = apply (diff C1 C3) C4
C6’ = apply (diff C5 C6) C8
C7’ = apply (diff C6 C7) C6’

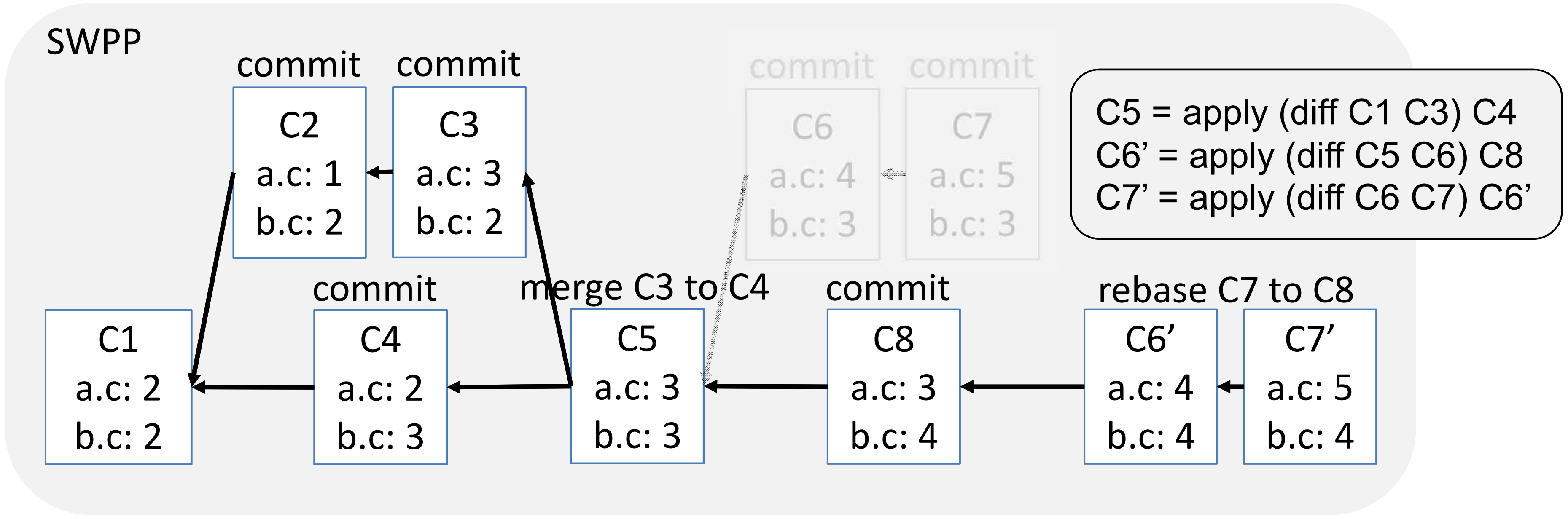
Basic Operations

- Commit:
  Add a fresh commit with a new development
- Merge (two parents):
  Add a commit by merging two commits
- Rebase:
  Add commits by applying existing changes to a commit

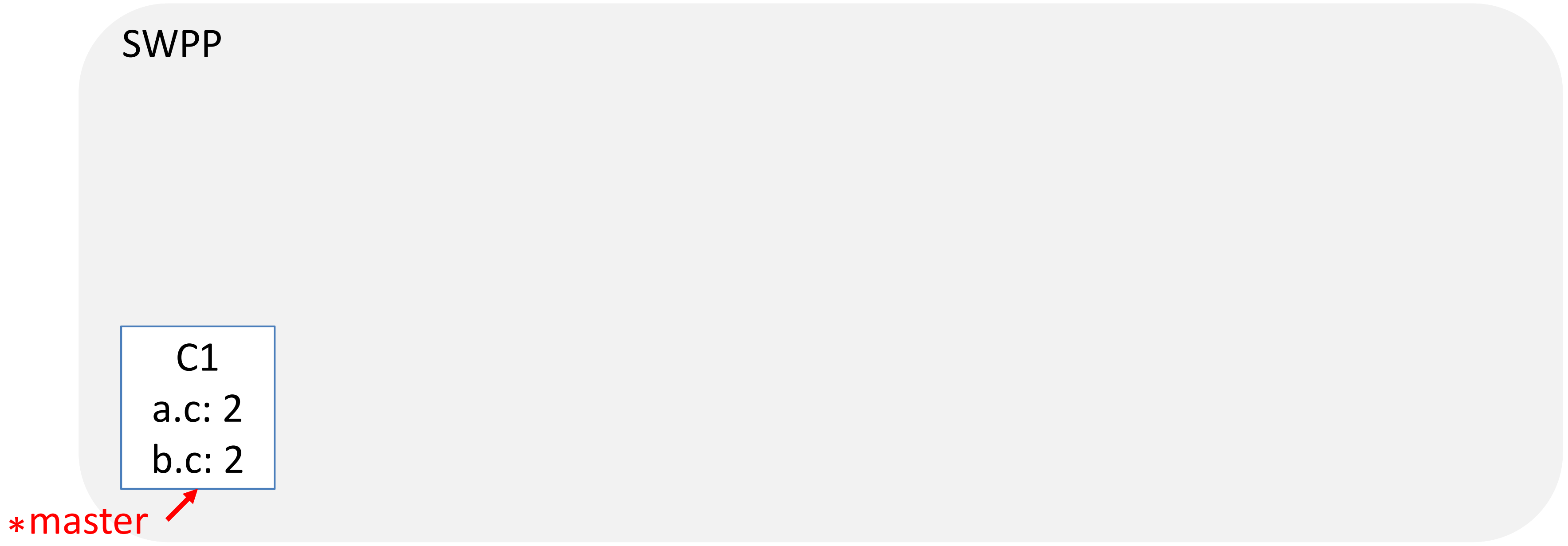


Basic Operations

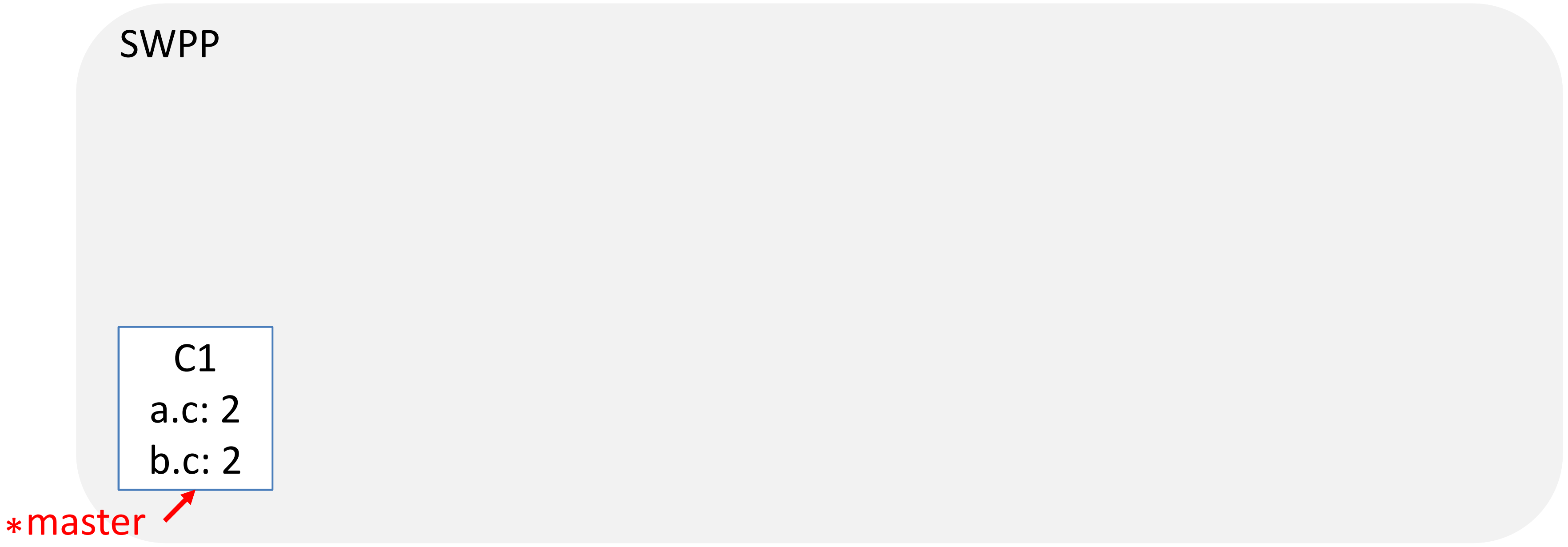
- Commit:
  Add a fresh commit with a new development
- Merge (two parents):
  Add a commit by merging two commits
- Rebase:
  Add commits by applying existing changes to a commit



Branch: Cursor to a commit

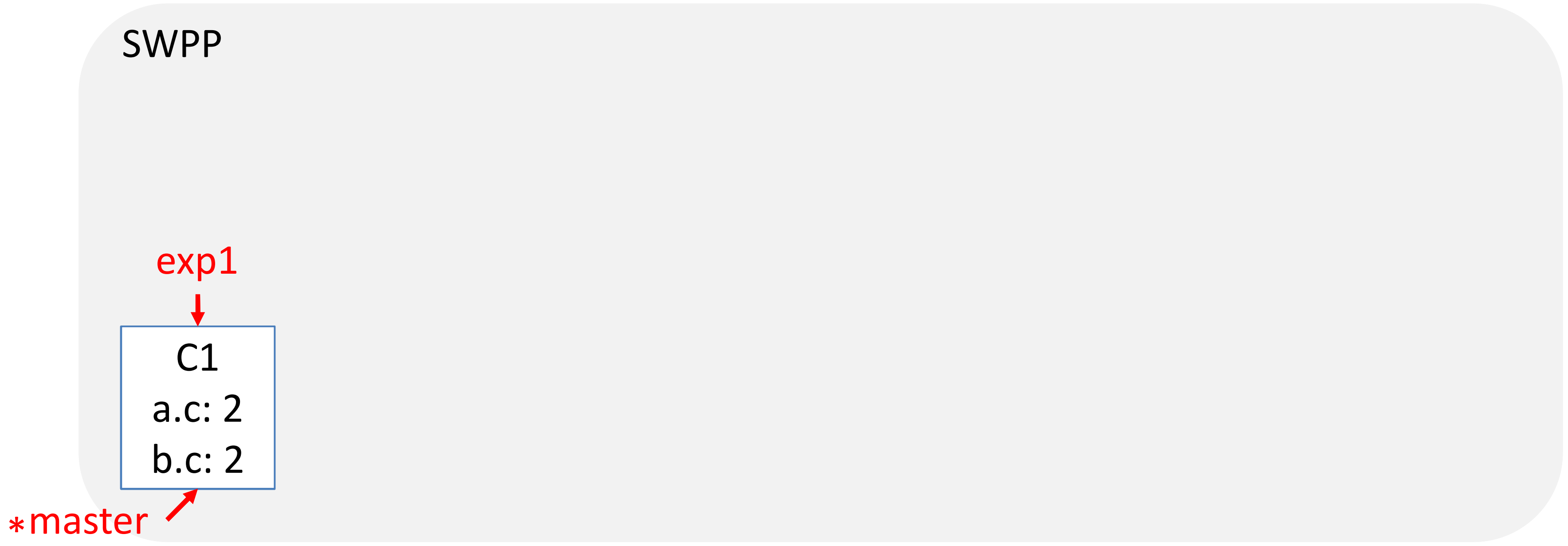


Branch: Cursor to a commit



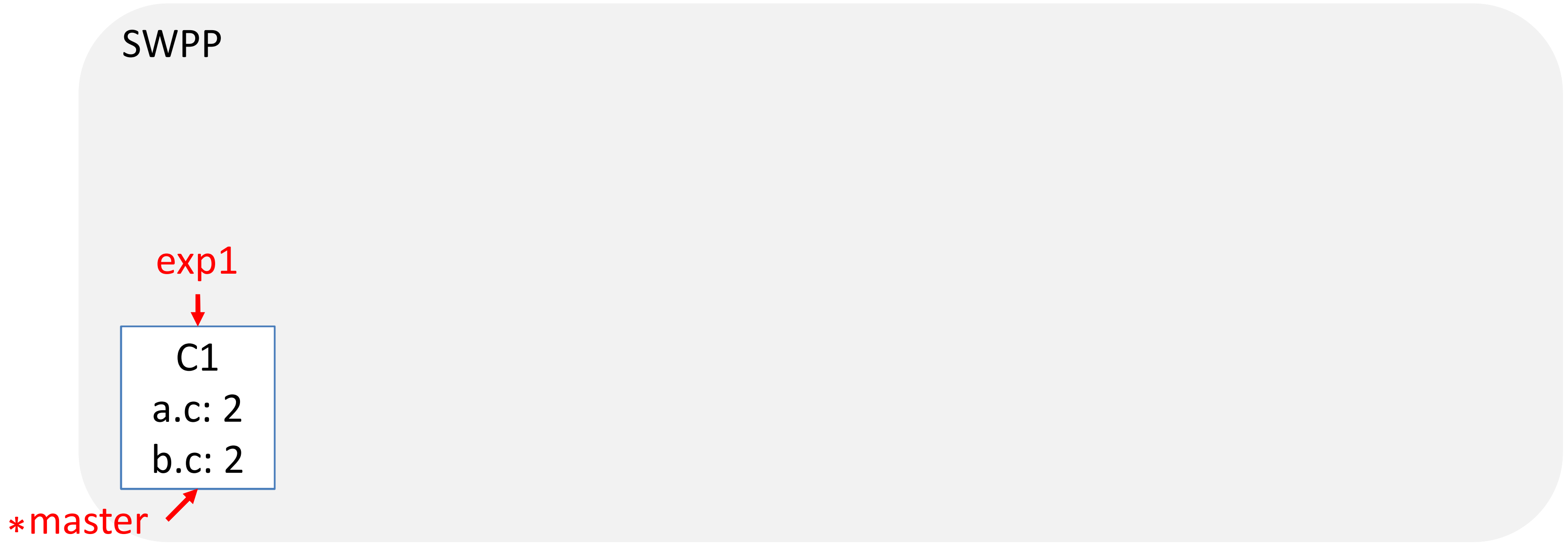
git branch exp1

Branch: Cursor to a commit



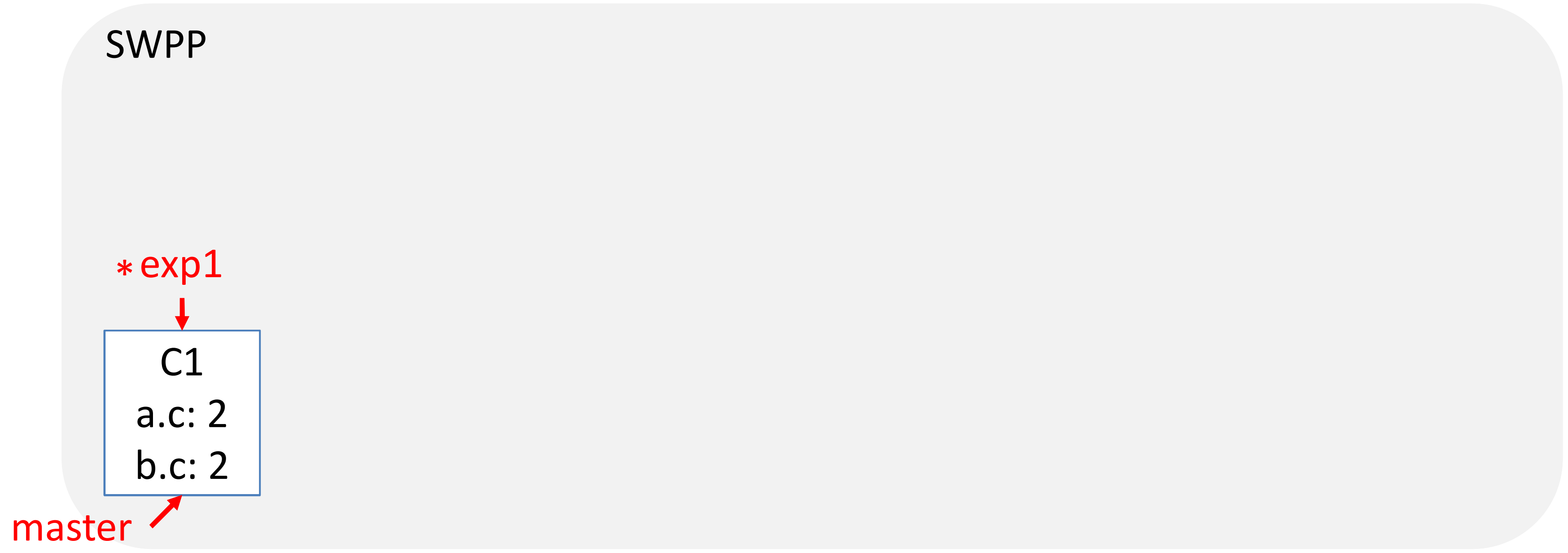
```
git branch exp1
```

Branch: Cursor to a commit



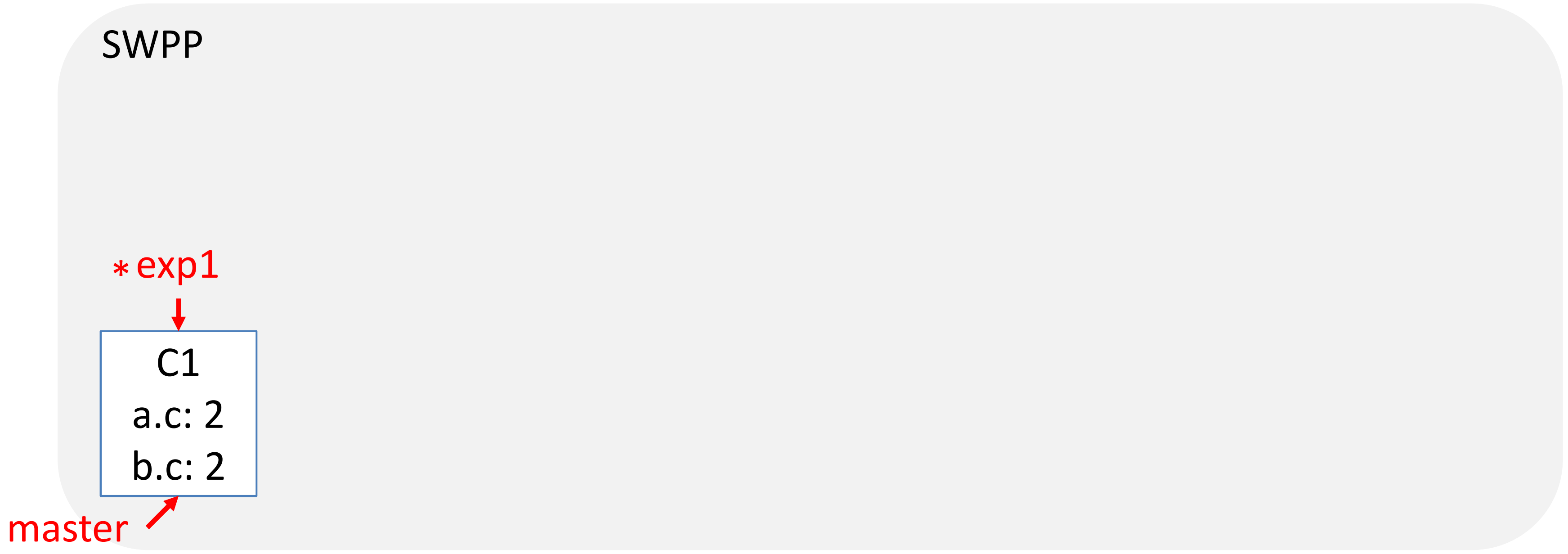
```
git branch exp1
git checkout exp1
```

Branch: Cursor to a commit



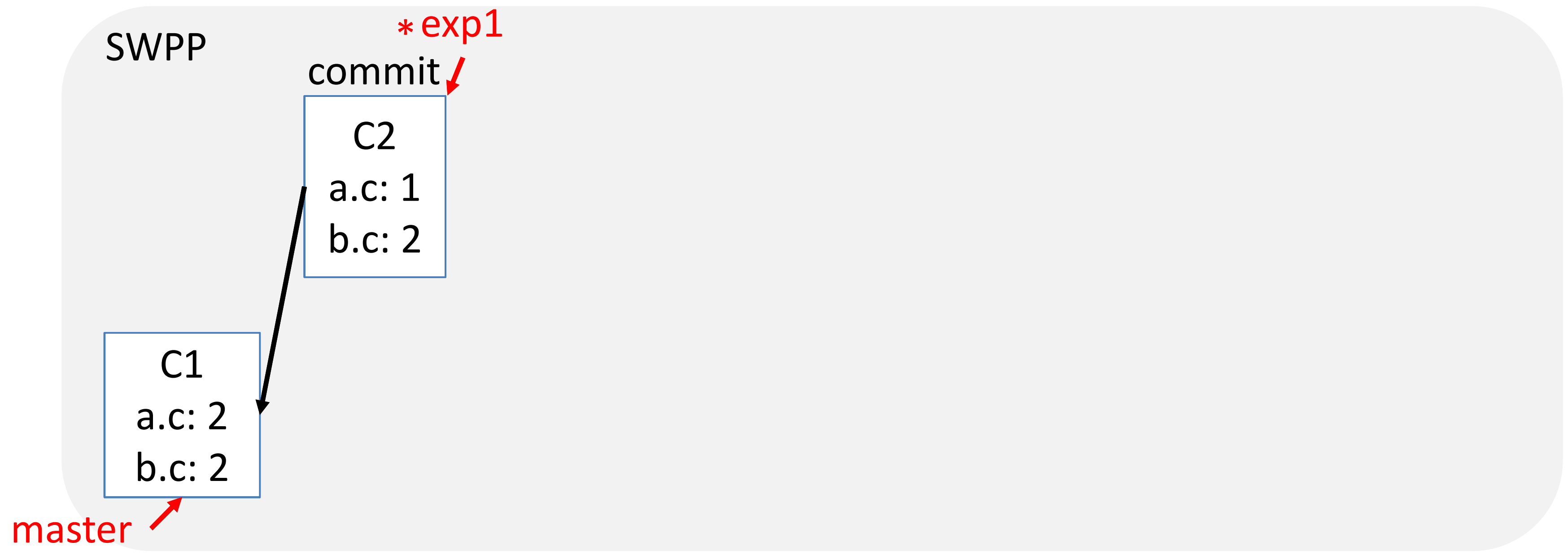
```
git branch exp1
git checkout exp1
```

Branch: Cursor to a commit



```
git branch exp1
git checkout exp1
git commit –a –m “C2” // after wr 1 to a.c
```

Branch: Cursor to a commit

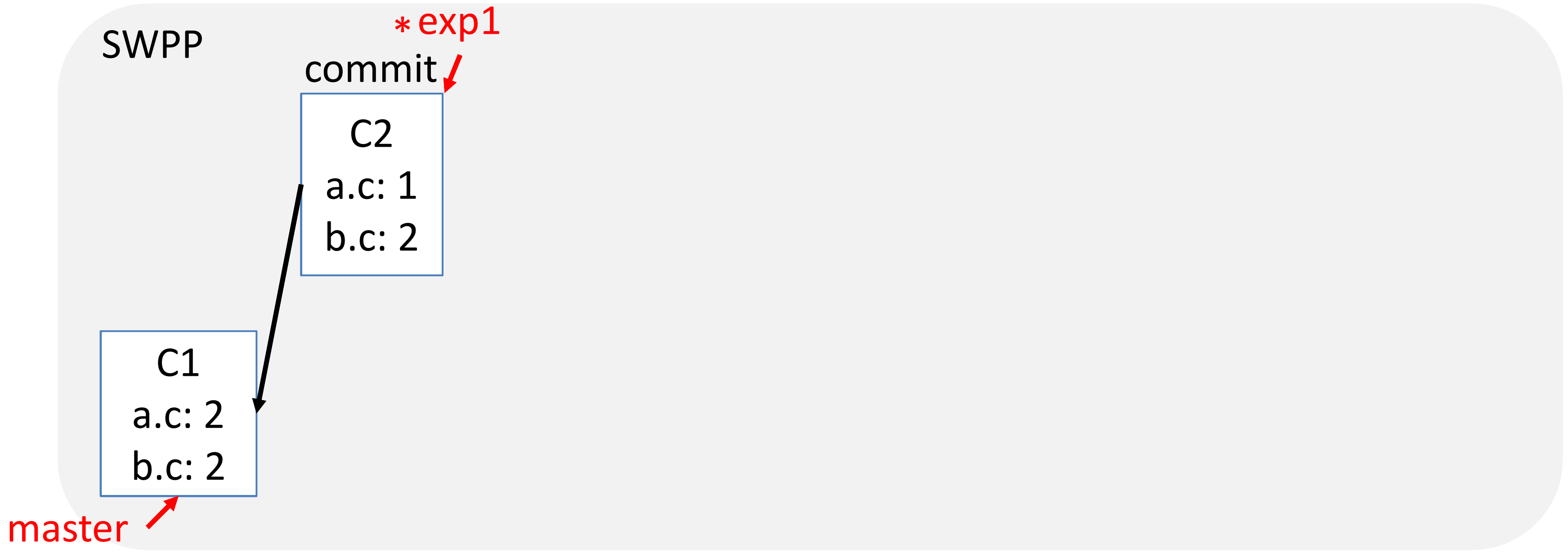


git branch exp1

git checkout exp1

git commit –a –m “C2” // after wr 1 to a.c

Branch: Cursor to a commit



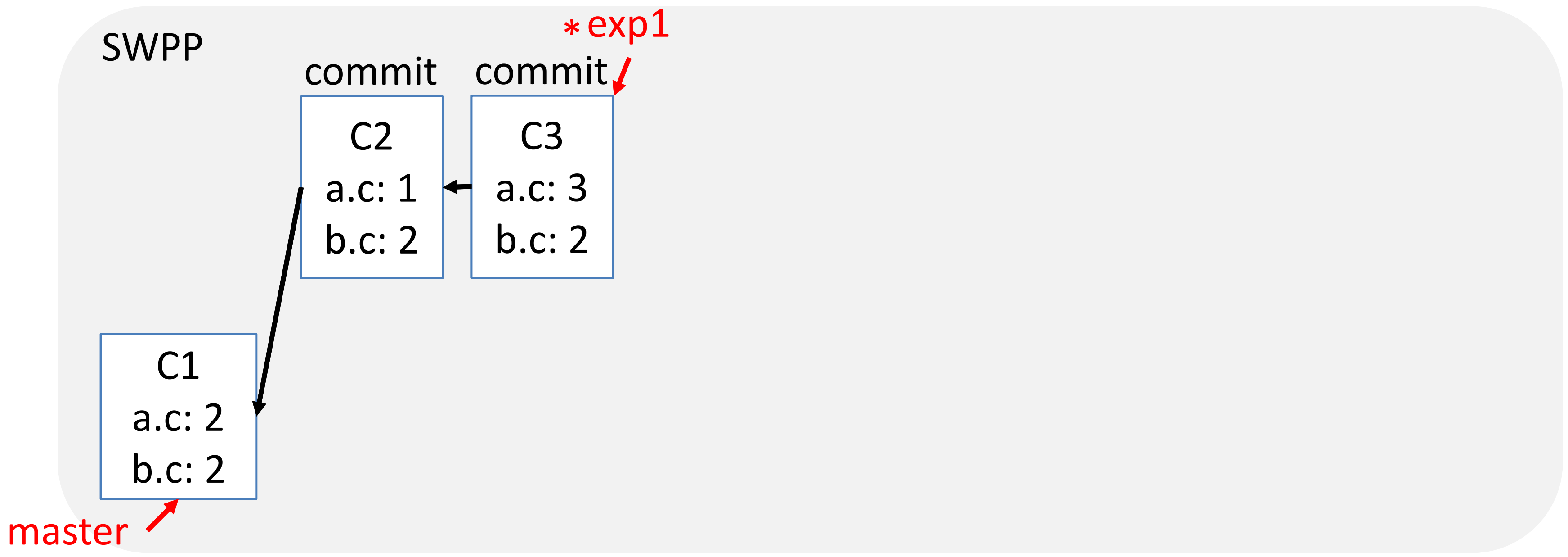
git branch exp1

git checkout exp1

git commit –a –m "C2" // after wr 1 to a.c

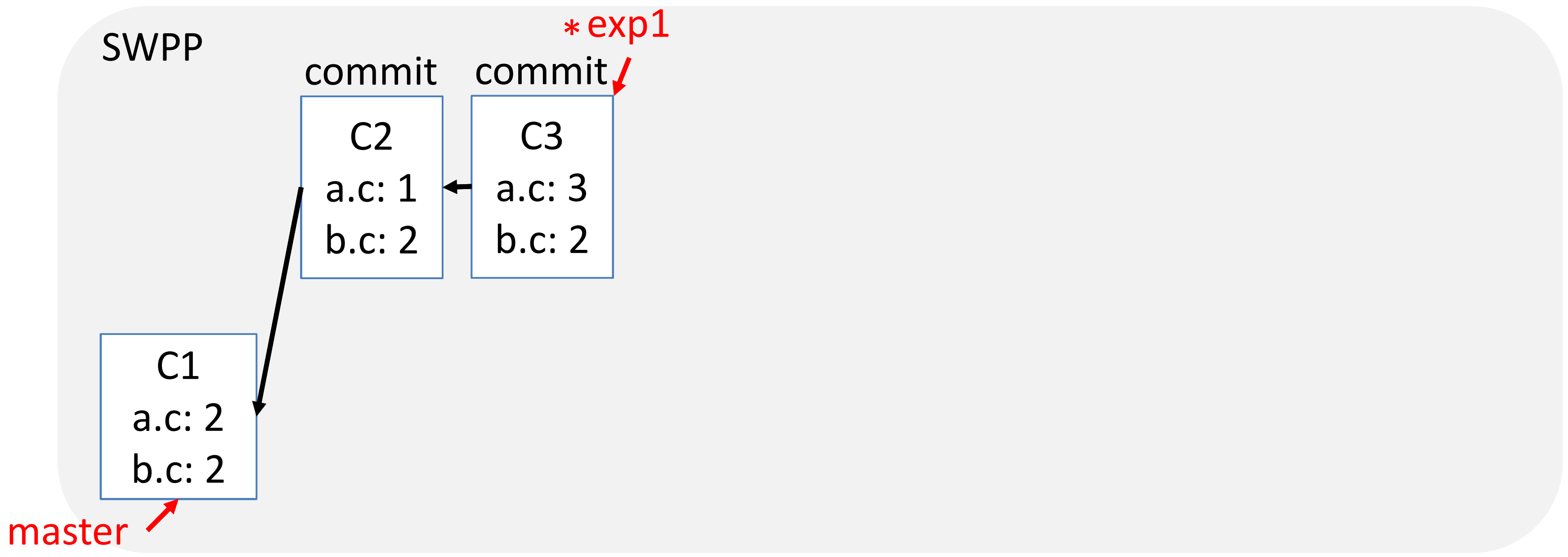
git commit –a –m "C3" // after wr 3 to a.c

Branch: Cursor to a commit



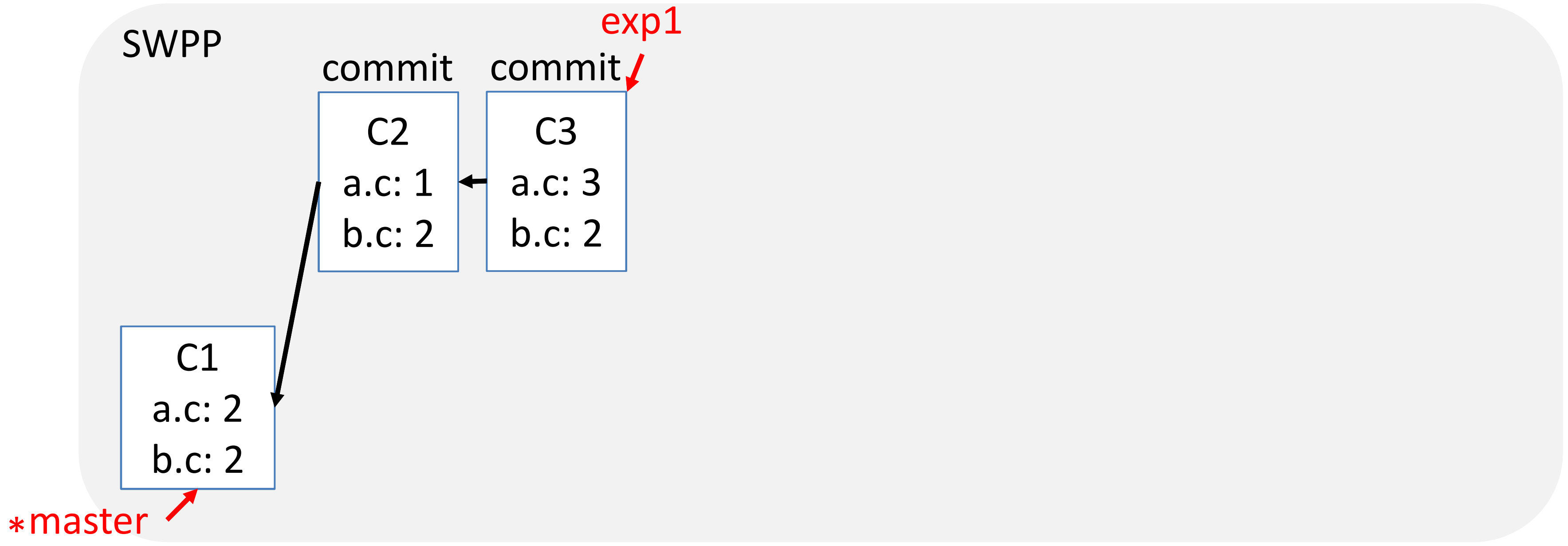
```
git branch exp1
git checkout exp1
git commit –a –m "C2" // after wr 1 to a.c
git commit –a –m "C3" // after wr 3 to a.c
```

Branch: Cursor to a commit



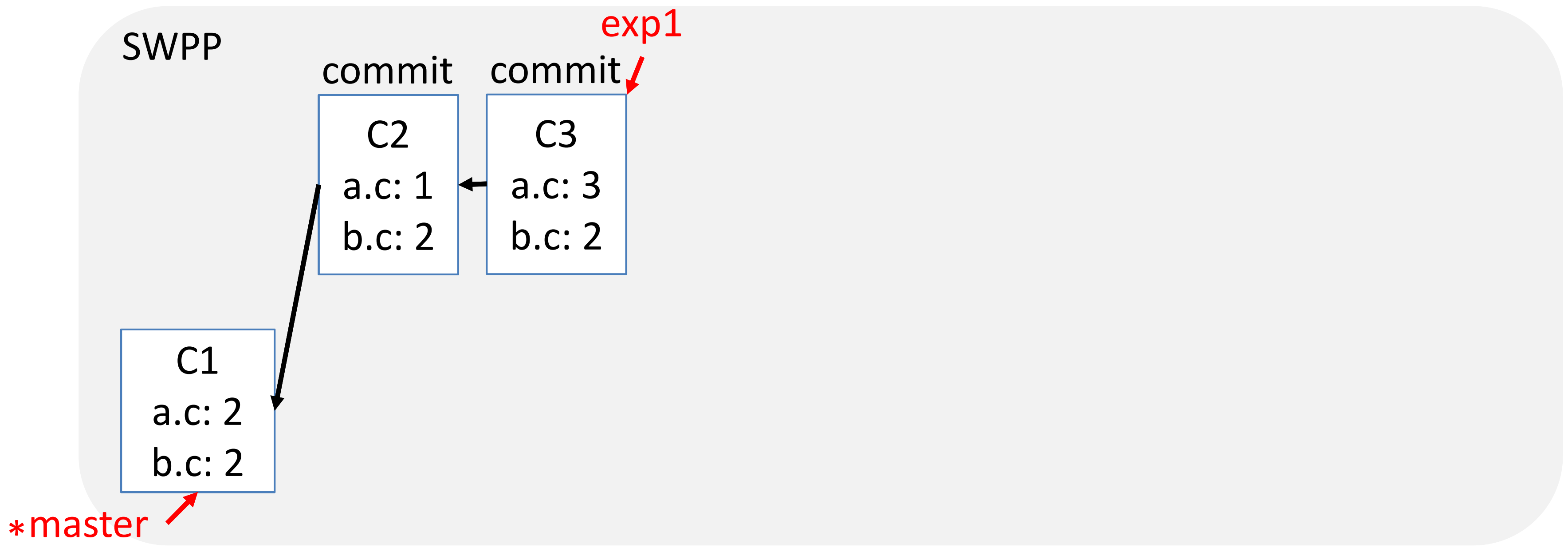
```
git branch exp1
git checkout exp1
git commit –a –m “C2” // after wr 1 to a.c
git commit –a –m “C3” // after wr 3 to a.c
git checkout master
```

Branch: Cursor to a commit



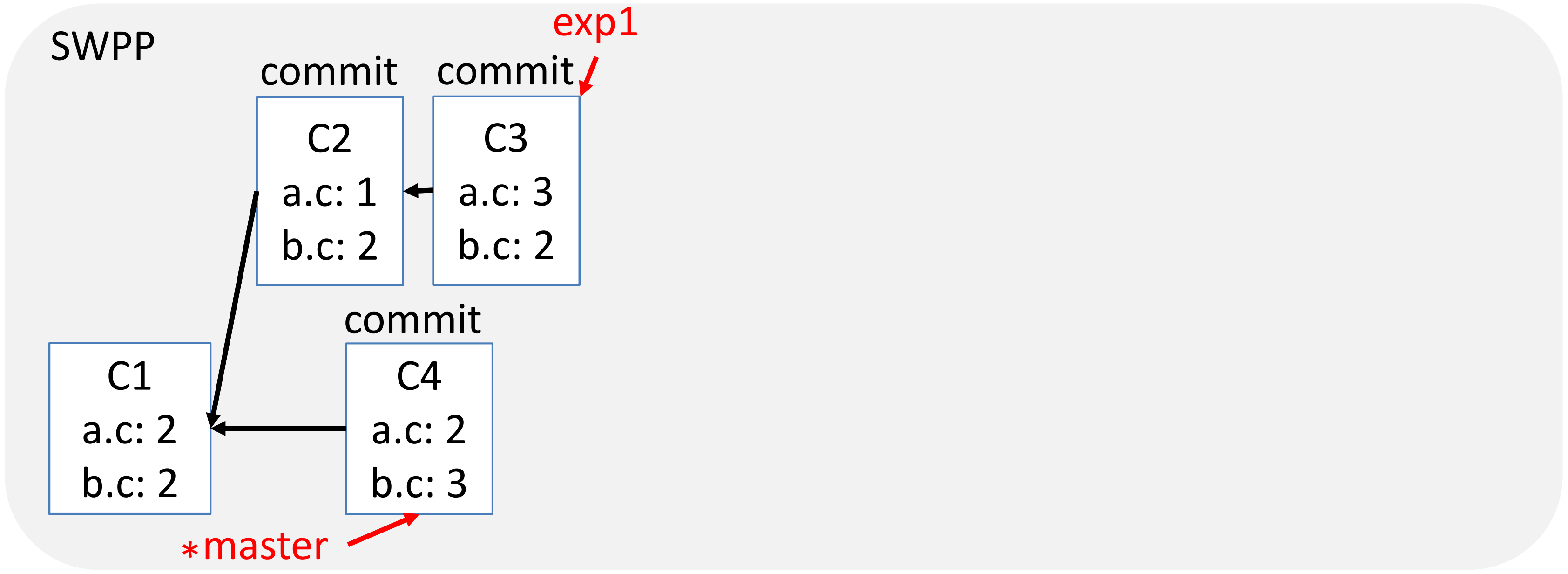
```
git branch exp1
git checkout exp1
git commit –a –m “C2” // after wr 1 to a.c
git commit –a –m “C3” // after wr 3 to a.c
git checkout master
```

Branch: Cursor to a commit



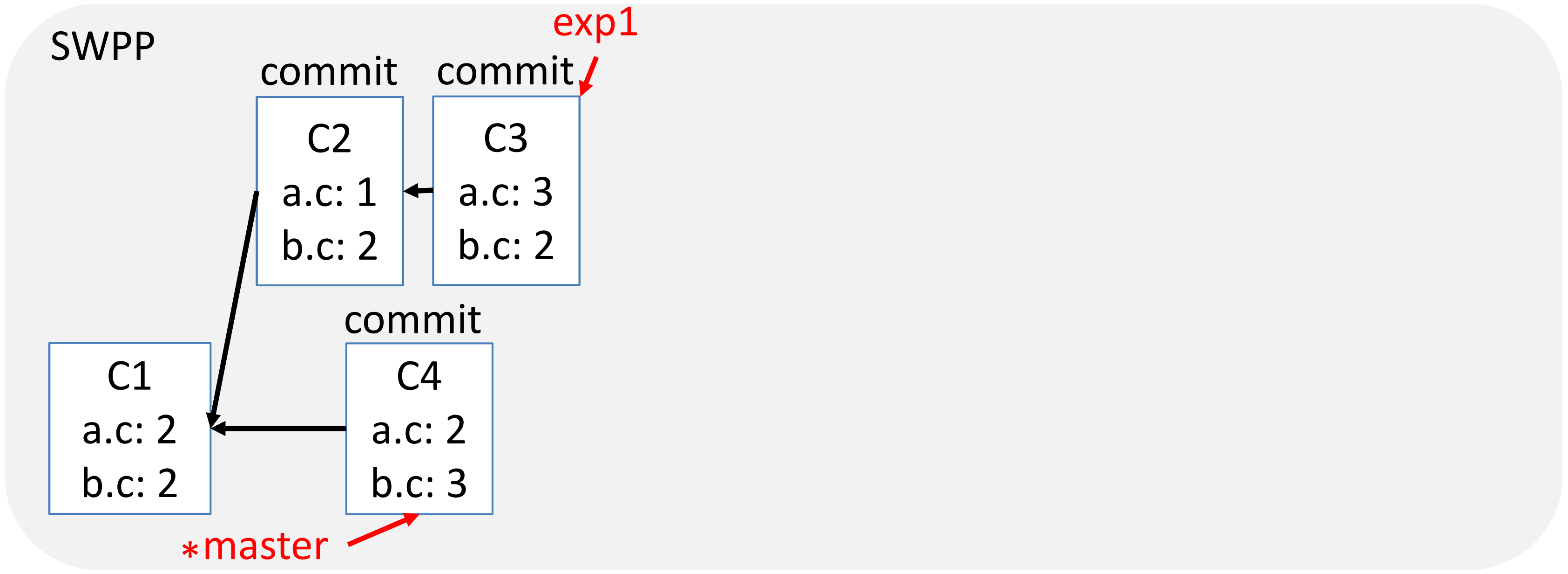
```
git branch exp1
git checkout exp1
git commit –a –m “C2” // after wr 1 to a.c
git commit –a –m “C3” // after wr 3 to a.c
git checkout master
git commit –a –m “C4” // after wr 3 to b.c
```

Branch: Cursor to a commit



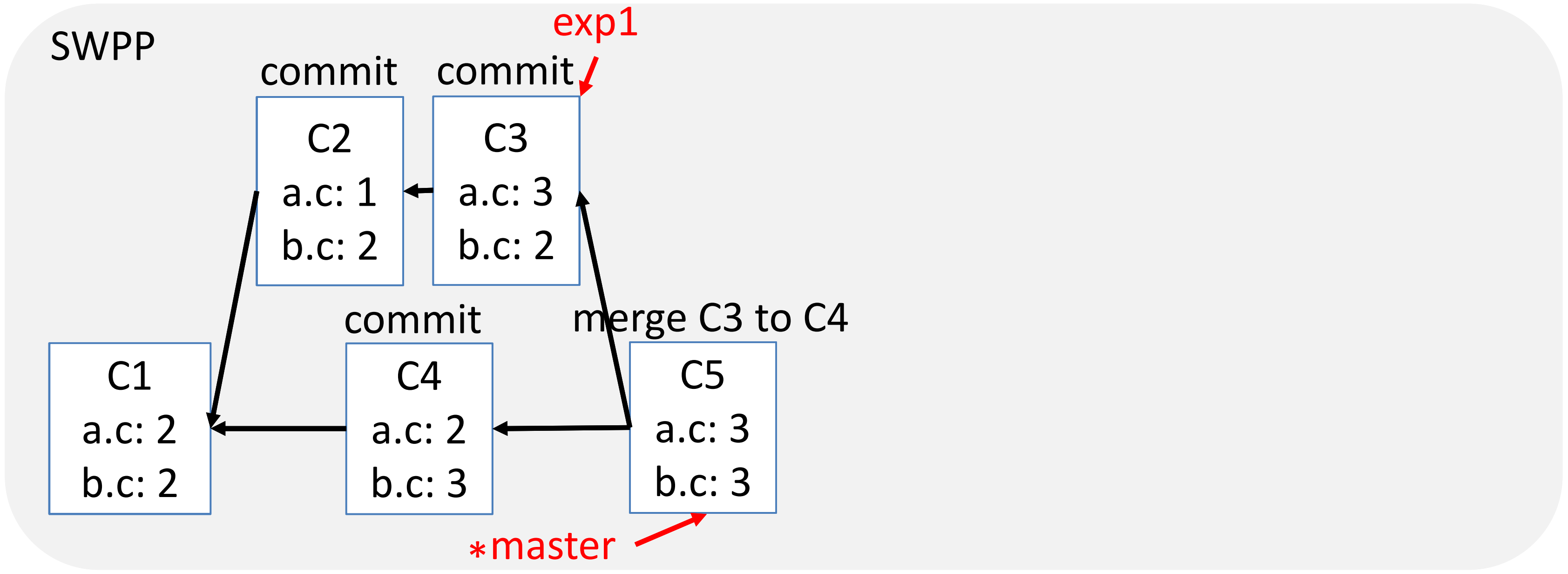
```
git branch exp1
git checkout exp1
git commit –a –m “C2” // after wr 1 to a.c
git commit –a –m “C3” // after wr 3 to a.c
git checkout master
git commit –a –m “C4” // after wr 3 to b.c
```

Branch: Cursor to a commit



```
git branch exp1
git checkout exp1
git commit –a –m “C2” // after wr 1 to a.c
git commit –a –m “C3” // after wr 3 to a.c
git checkout master
git commit –a –m “C4” // after wr 3 to b.c
git merge exp1
```

Branch: Cursor to a commit



git branch exp1

git checkout exp1

git commit –a –m “C2” // after wr 1 to a.c

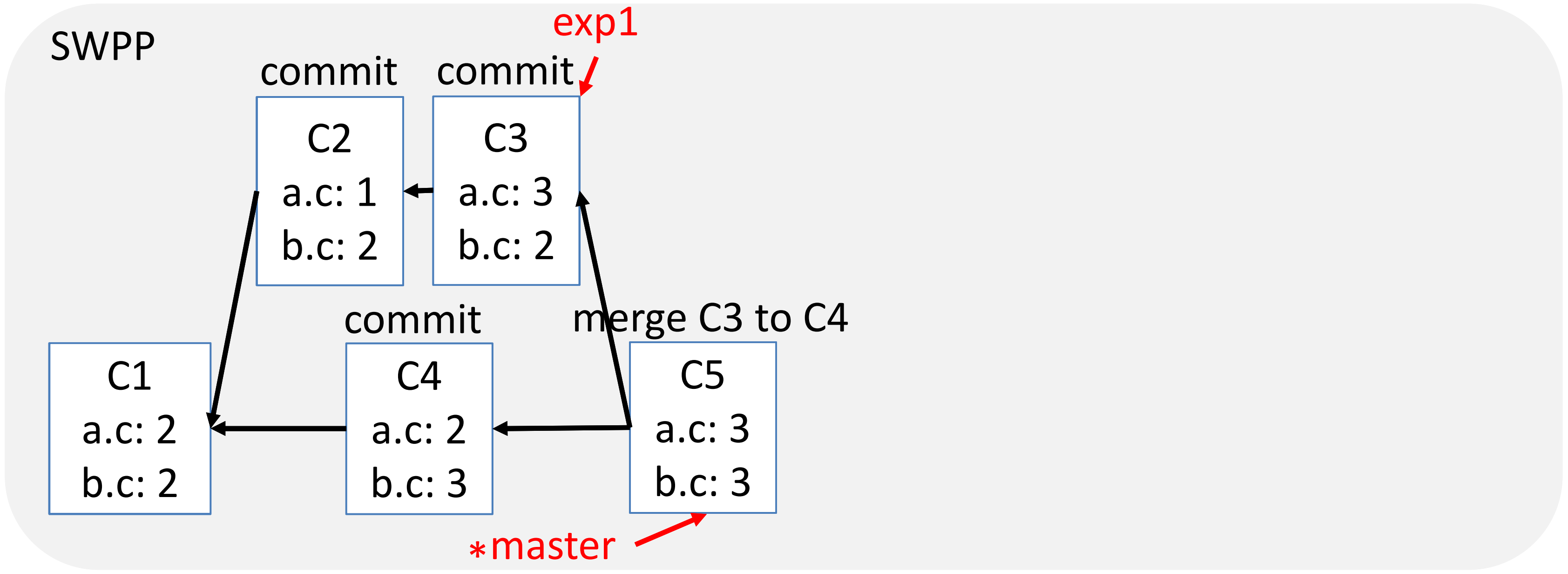
git commit –a –m “C3” // after wr 3 to a.c

git checkout master

git commit –a –m “C4” // after wr 3 to b.c

git merge exp1

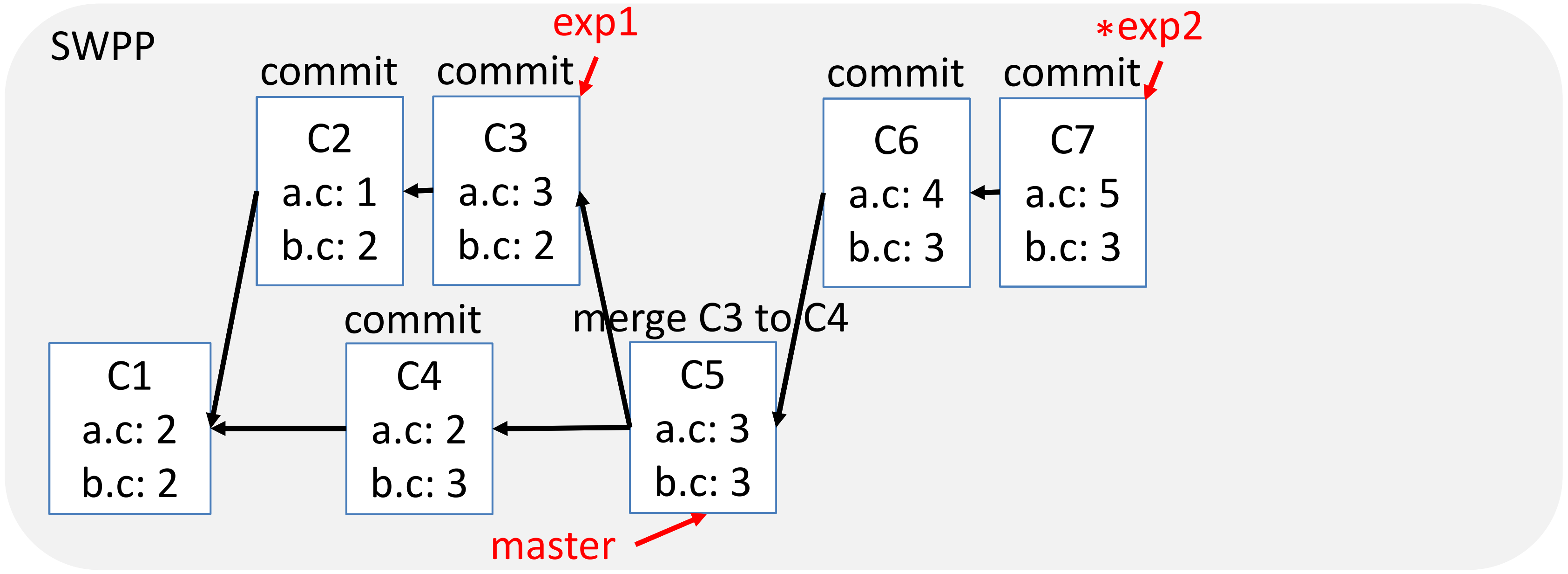
Branch: Cursor to a commit



```
git branch exp1
git checkout exp1
git commit –a –m “C2” // after wr 1 to a.c
git commit –a –m “C3” // after wr 3 to a.c
git checkout master
git commit –a –m “C4” // after wr 3 to b.c
git merge exp1
```

```
git branch exp2
git checkout exp2
git commit –a –m “C6” // after wr 4 to a.c
git commit –a –m “C7” // after wr 5 to a.c
```

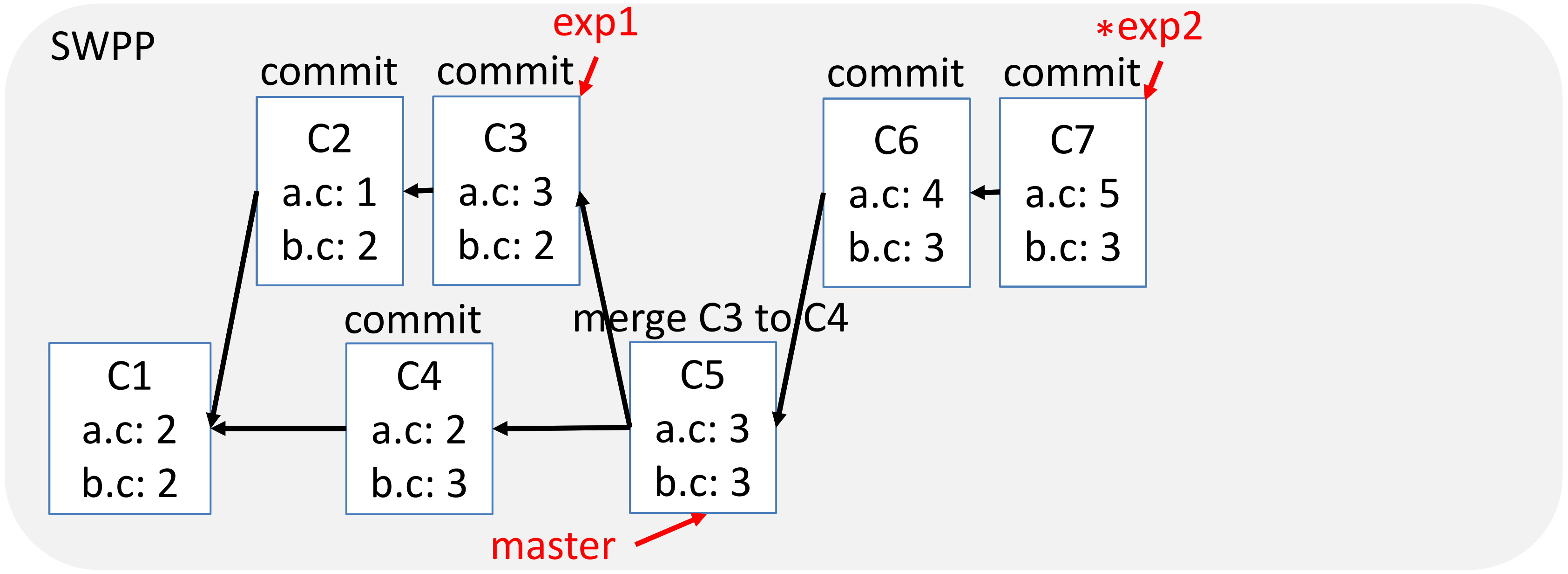
Branch: Cursor to a commit



```
git branch exp1
git checkout exp1
git commit –a –m “C2” // after wr 1 to a.c
git commit –a –m “C3” // after wr 3 to a.c
git checkout master
git commit –a –m “C4” // after wr 3 to b.c
git merge exp1
```

```
git branch exp2
git checkout exp2
git commit –a –m “C6” // after wr 4 to a.c
git commit –a –m “C7” // after wr 5 to a.c
```

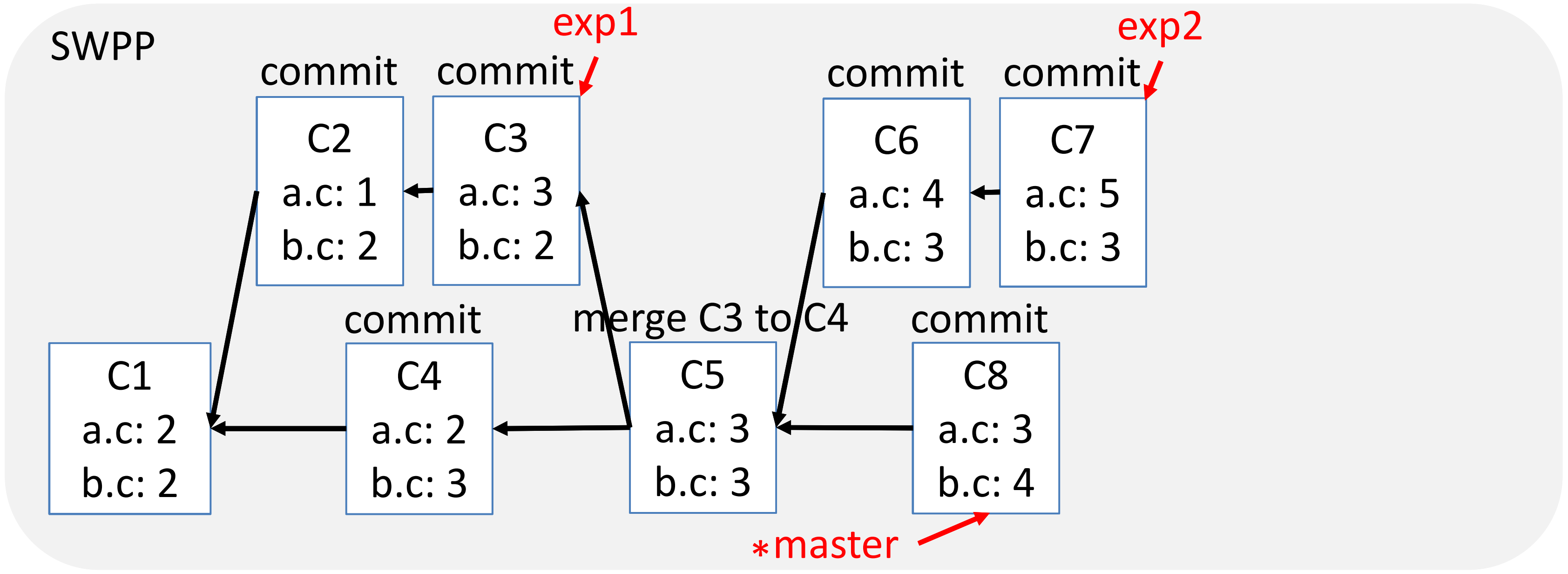
Branch: Cursor to a commit



```
git branch exp1
git checkout exp1
git commit –a –m “C2” // after wr 1 to a.c
git commit –a –m “C3” // after wr 3 to a.c
git checkout master
git commit –a –m “C4” // after wr 3 to b.c
git merge exp1
```

```
git branch exp2
git checkout exp2
git commit –a –m “C6” // after wr 4 to a.c
git commit –a –m “C7” // after wr 5 to a.c
git checkout master
git commit –a –m “C8” // after wr 4 to b.c
```

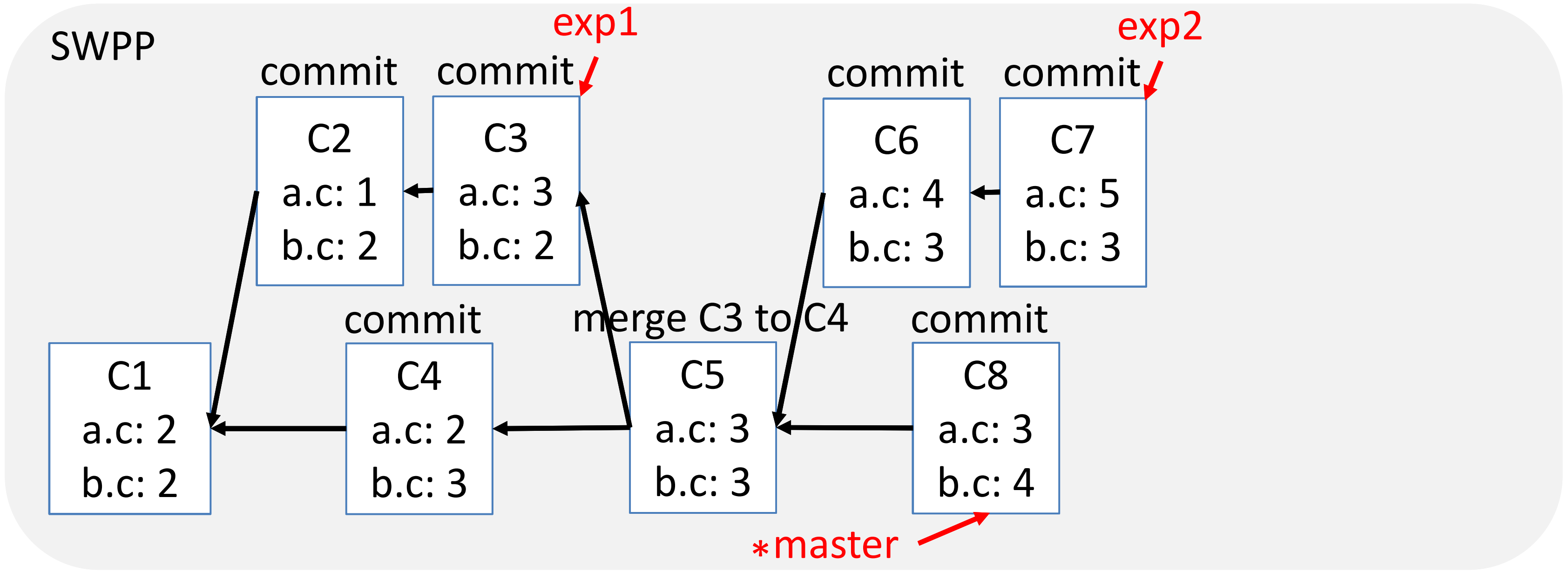
Branch: Cursor to a commit



```
git branch exp1
git checkout exp1
git commit –a –m “C2” // after wr 1 to a.c
git commit –a –m “C3” // after wr 3 to a.c
git checkout master
git commit –a –m “C4” // after wr 3 to b.c
git merge exp1
```

```
git branch exp2
git checkout exp2
git commit –a –m “C6” // after wr 4 to a.c
git commit –a –m “C7” // after wr 5 to a.c
git checkout master
git commit –a –m “C8” // after wr 4 to b.c
```

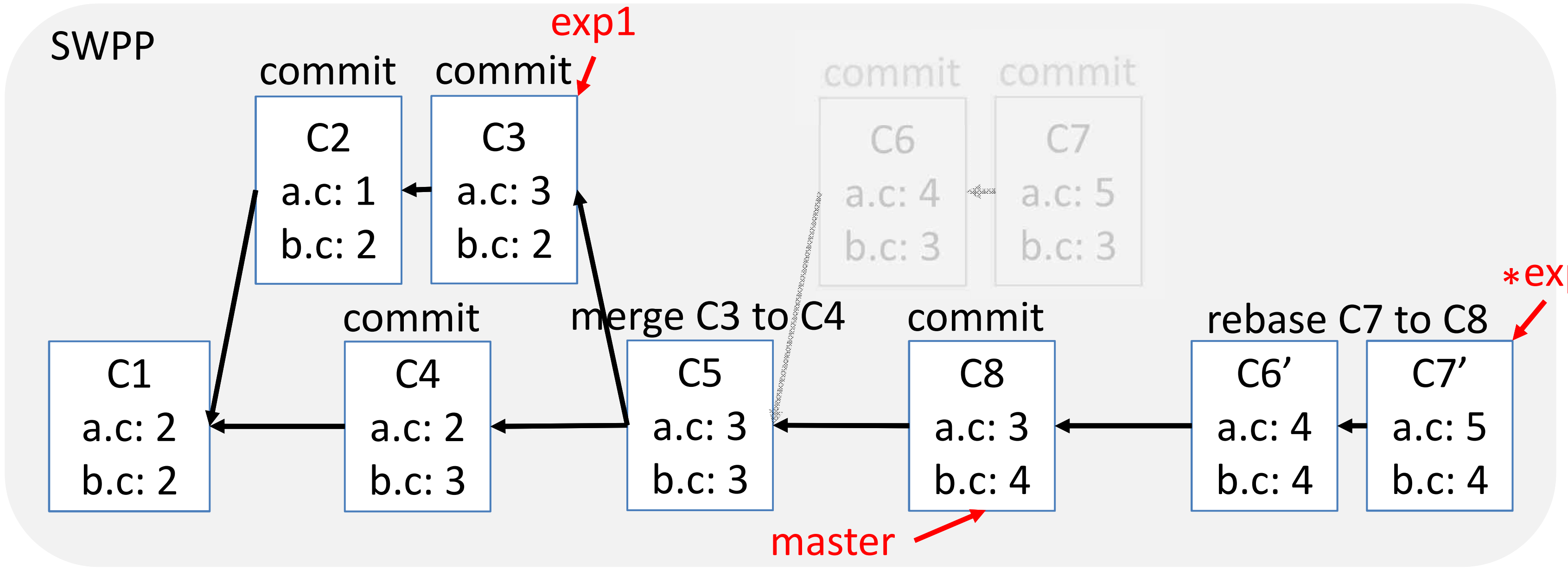
Branch: Cursor to a commit



```
git branch exp1
git checkout exp1
git commit –a –m “C2” // after wr 1 to a.c
git commit –a –m “C3” // after wr 3 to a.c
git checkout master
git commit –a –m “C4” // after wr 3 to b.c
git merge exp1
```

```
git branch exp2
git checkout exp2
git commit –a –m “C6” // after wr 4 to a.c
git commit –a –m “C7” // after wr 5 to a.c
git checkout master
git commit –a –m “C8” // after wr 4 to b.c
git rebase master exp2
```

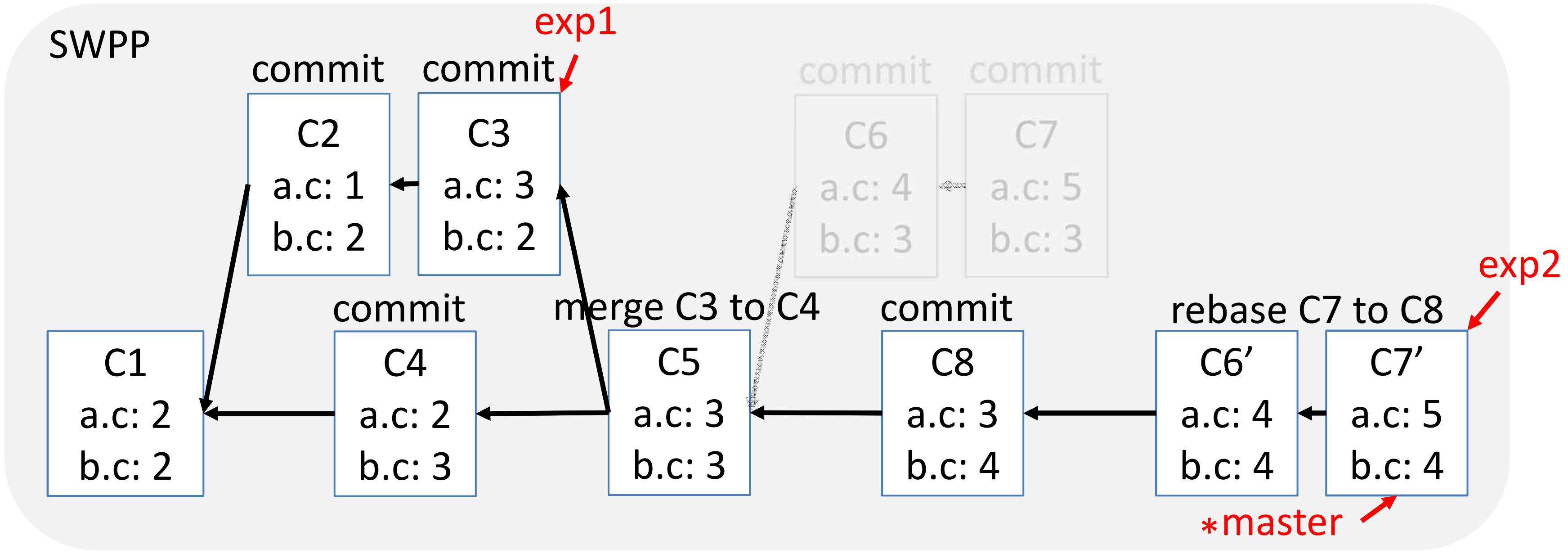
Branch: Cursor to a commit



```
git branch exp1
git checkout exp1
git commit –a –m “C2” // after wr 1 to a.c
git commit –a –m “C3” // after wr 3 to a.c
git checkout master
git commit –a –m “C4” // after wr 3 to b.c
git merge exp1
```

```
git branch exp2
git checkout exp2
git commit –a –m “C6” // after wr 4 to a.c
git commit –a –m “C7” // after wr 5 to a.c
git checkout master
git commit –a –m “C8” // after wr 4 to b.c
git rebase master exp2
```

Branch: Cursor to a commit

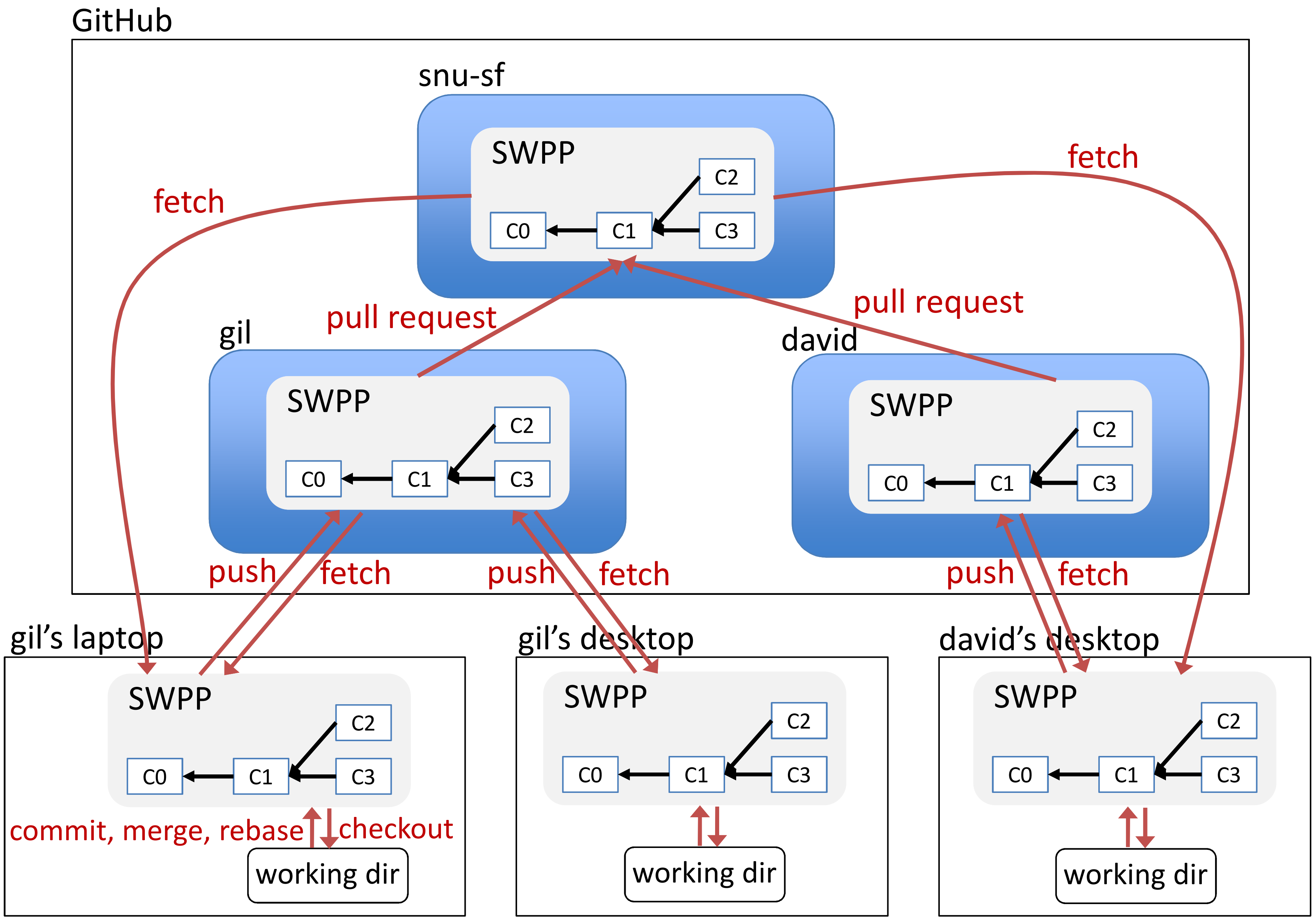


```
git branch exp1
git checkout exp1
git commit –a –m “C2” // after wr 1 to a.c
git commit –a –m “C3” // after wr 3 to a.c
git checkout master
git commit –a –m “C4” // after wr 3 to b.c
git merge exp1
```

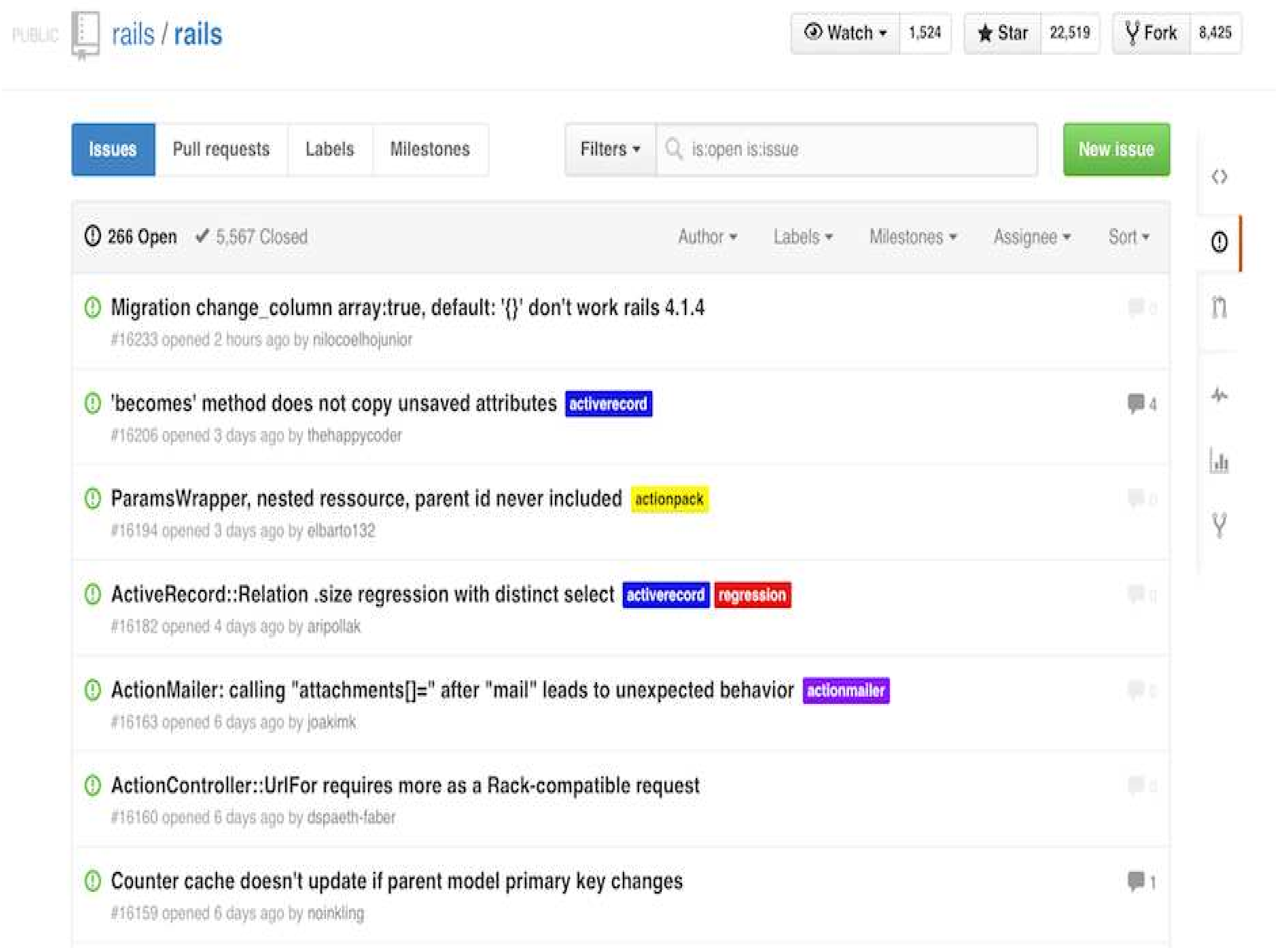
```
git branch exp2
git checkout exp2
git commit –a –m “C6” // after wr 4 to a.c
git commit –a –m “C7” // after wr 5 to a.c
git checkout master
git commit –a –m “C8” // after wr 4 to b.c
git rebase master exp2
git checkout master; git merge exp2
```

GitHub

Main Workflow



GitHub – Issue Tracking



PUBLIC rails / rails

Watch 1,524 | Star 22,519 | Fork 8,425

Issues | Pull requests | Labels | Milestones | Filters | is:open is:issue | New issue

266 Open ✓ 5,567 Closed | Author | Labels | Milestones | Assignee | Sort

Migration change_column array:true, default: '{}' don't work rails 4.1.4
#16233 opened 2 hours ago by nilocoelhojunior — 0

'becomes' method does not copy unsaved attributes `activerecord`
#16206 opened 3 days ago by thehappycoder — 4

ParamsWrapper, nested ressource, parent id never included `actionpack`
#16194 opened 3 days ago by elbarto132 — 0

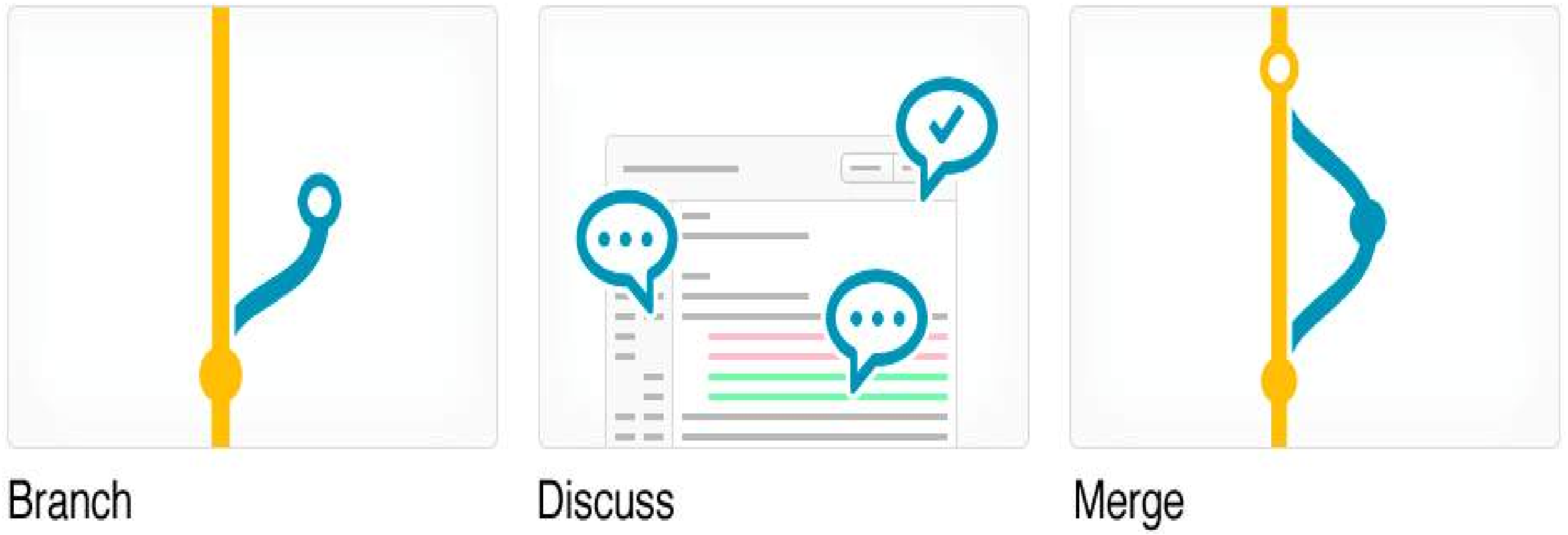
ActiveRecord::Relation .size regression with distinct select `activerecord` `regression`
#16182 opened 4 days ago by aripollak — 0

ActionMailer: calling "attachments[]=" after "mail" leads to unexpected behavior `actionmailer`
#16163 opened 6 days ago by joakimk — 0

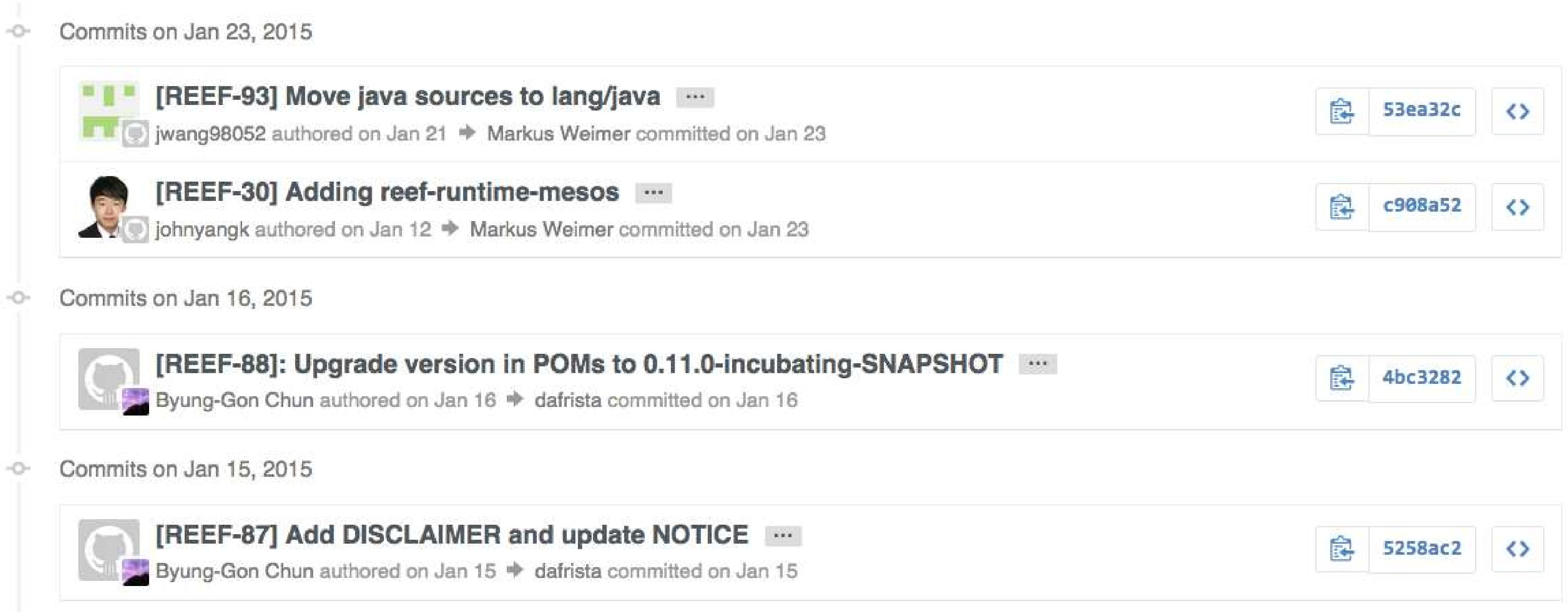
ActionController::UrlFor requires more as a Rack-compatible request
#16160 opened 6 days ago by dspaeth-faber — 0

Counter cache doesn't update if parent model primary key changes
#16159 opened 6 days ago by noinkling — 1

GitHub – Code Review with Pull Request



GitHub – Commit Log



Commits on Jan 23, 2015

[REEF-93] Move java sources to lang/java ···
jwang98052 authored on Jan 21 → Markus Weimer committed on Jan 23
53ea32c

[REEF-30] Adding reef-runtime-mesos ···
johnyangk authored on Jan 12 → Markus Weimer committed on Jan 23
c908a52

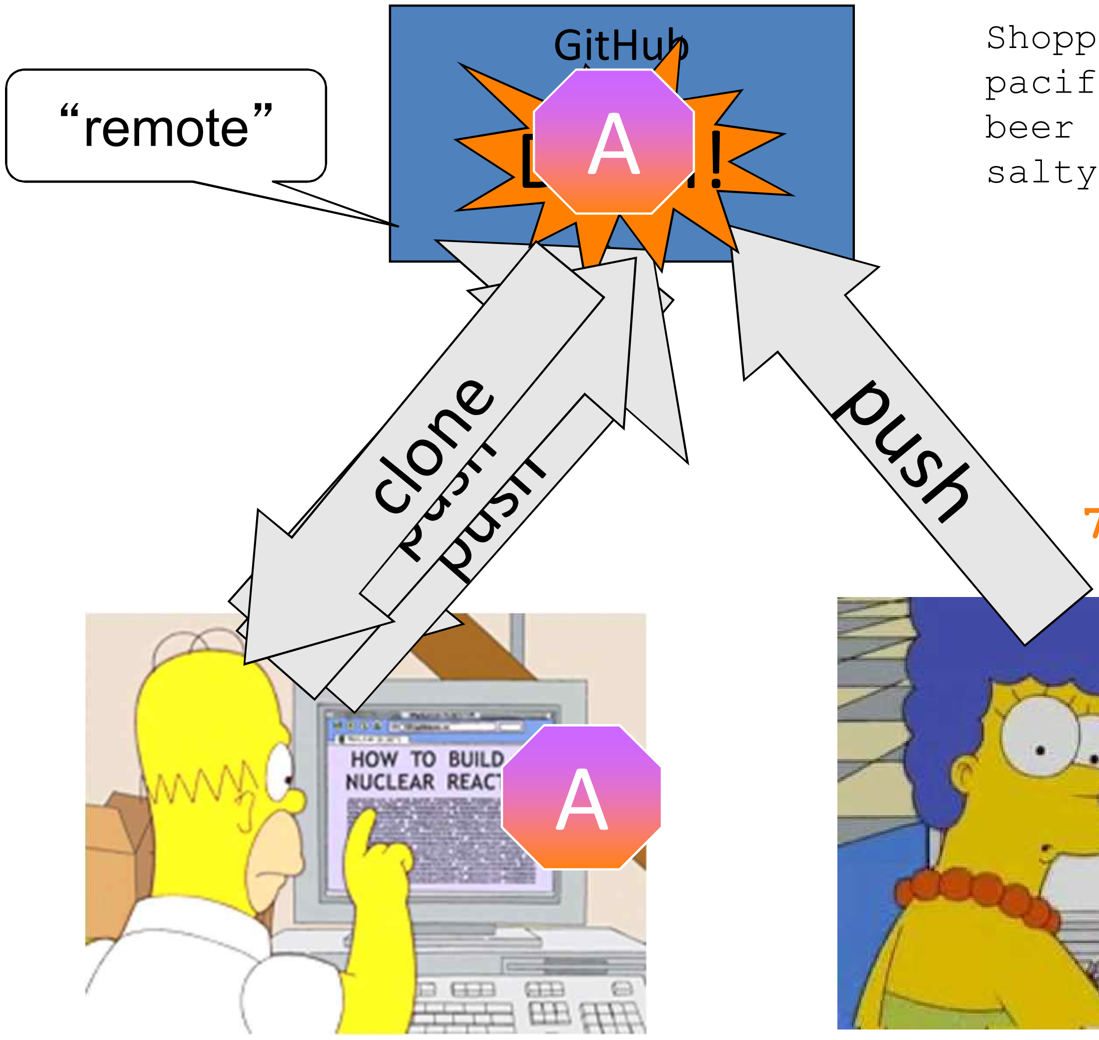
Commits on Jan 16, 2015

[REEF-88]: Upgrade version in POMs to 0.11.0-incubating-SNAPSHOT ···
Byung-Gon Chun authored on Jan 16 → dafrista committed on Jan 16
4bc3282

Commits on Jan 15, 2015

[REEF-87] Add DISCLAIMER and update NOTICE ···
Byung-Gon Chun authored on Jan 15 → dafrista committed on Jan 15
5258ac2

Merge Conflict



```
Shopping list:
pacifiers for Maggie
beer for Homer
salty snacks...mmm, snacks
```

770dfb346ace24

Pull = Fetch + Merge

- Fetch = copies new commits from the origin
- Merge two repos = try to apply commits in both
  - Conflict if different changes to same file "too close" together
  - `git pull` = `git pull origin master`
- Successful merge implies commit!
- Always commit your changes before merging/pulling
- Commit early & often—small commits OK!
  `git commit` (and `push`) when all done

Commit: a *tree snapshot* identified by a commit-ID

- 40-digit hex hash (SHA-1), unique in the universe…but a pain
  - use unique (in this repo) prefix, eg `770dfb`
  - `HEAD`: most recently committed version on current branch
  - `ORIG_HEAD`: right after a merge, points to pre-merged version
  - `HEAD~`***n***: n' th previous commit
  - `770dfb~2`: 2 commits before `770dfb`
  - `"master@{01-Sep-2012}"`: last commit on master branch prior to 1-Sep-2012

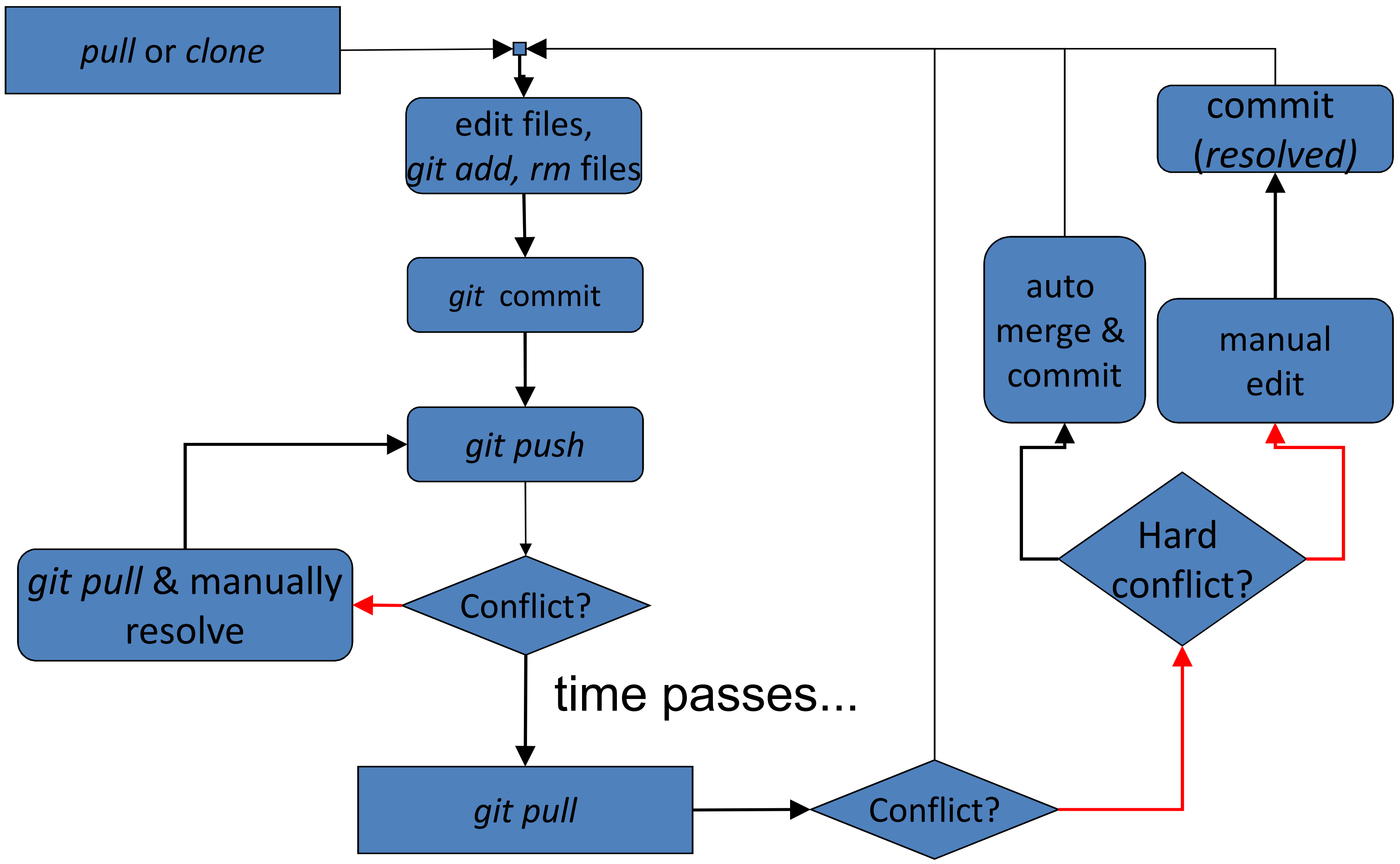
Undo!

```
git reset --hard ORIG_HEAD
git reset --hard HEAD
git checkout commit-id – files...
```

Track who changed what file and when

```
git blame files
git diff files
git diff branch files
git diff "master@{01-Sep-12}" files
git log ref..ref files
git log -since="date" files
```

Version control with conflicts



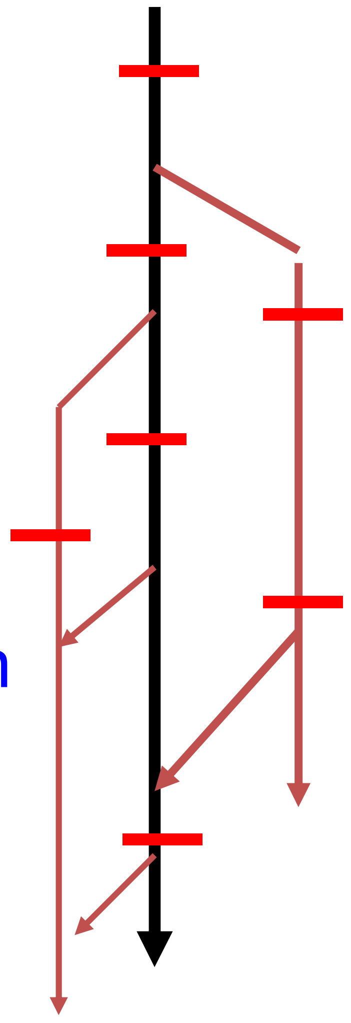
If you try to push to a remote and get a "non-fast-forward (error): failed to push some refs", which statement is FALSE?

- ☐ Some commits present at remote are not present on your local repo
- ☐ You need to do a merge/pull before you can complete the push
- ☐ You need to manually fix merge conflicts in one or more files
- ☐ Your local repo is out-of-date with respect to the remote

Branches

- Development **master** vs. **branches**
  - Creating branch is *cheap!*
  - switch among branches: *checkout*
- Separate commit histories per *branch*
- *Merge* branch back into master
  - ...or with *pushing* branch changes
  - Most branches eventually die
- Two common branch management strategies: feature branch, release branch
- Killer use case for agile SaaS: *branch per feature*

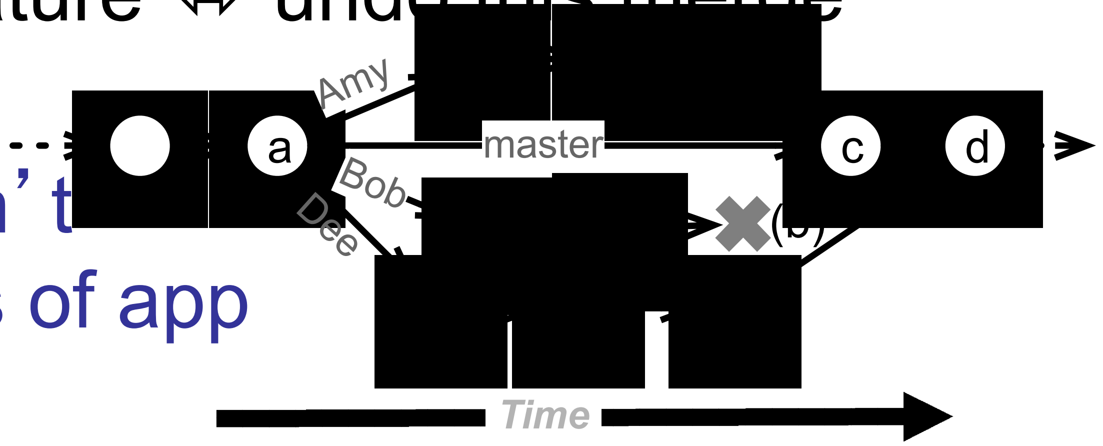
  release branch uncommon in SaaS



Creating new features without disrupting working code

1. To work on a new feature, create new branch *just for that feature*
   – many features can be in progress at same time
2. Use branch *only* for changes needed for *this feature*, then merge into master
3. Back out this feature ⇔ undo this merge

In well-factored app,
  1 feature shouldn’t t
  touch many parts of app



Mechanics

- Create new branch & switch to it

```
git branch CoolNewFeature
git checkout CoolNewFeature
```
🡐*current branch*

- Edit, add, make commits, etc. on branch
- Push branch to origin repo (optional):

```
git push origin CoolNewFeature
```

  - creates *tracking branch* on remote repo

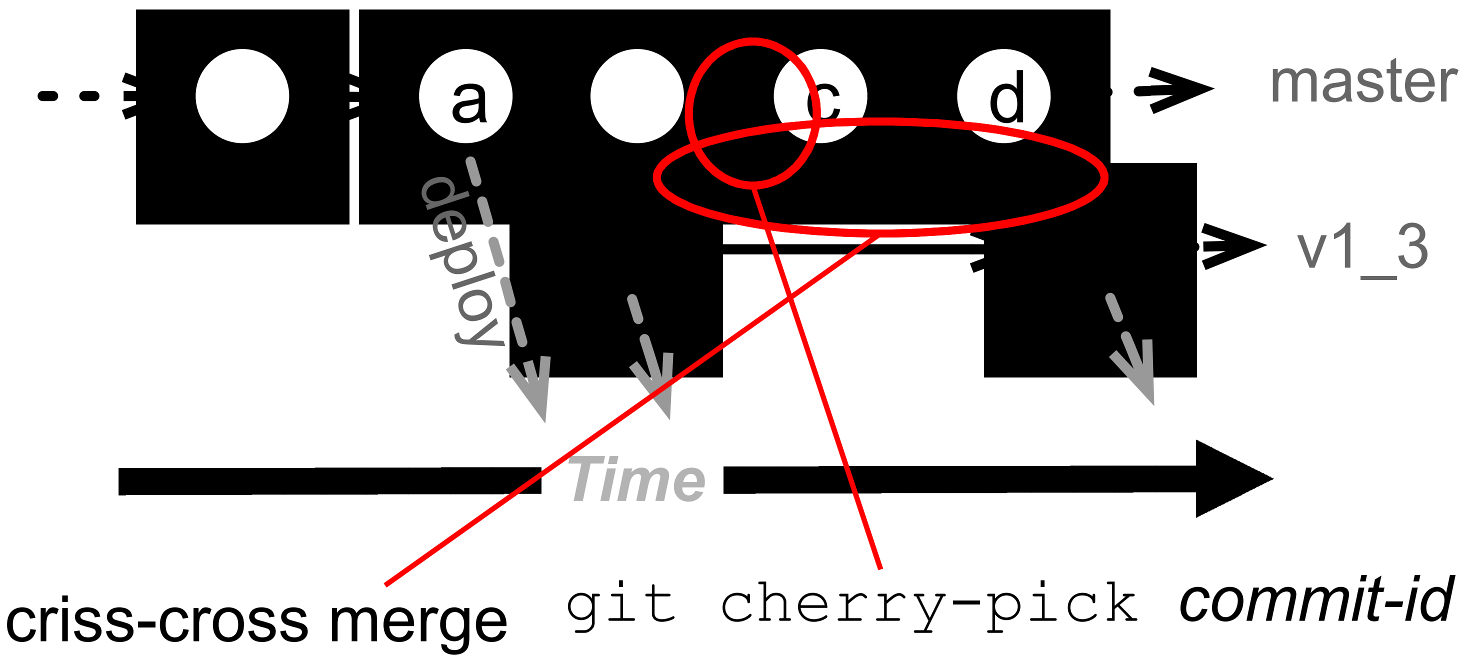
- Create a pull request, do code review, and merge into master in origin repo
- Switch back to master, and pull:

```
git checkout master
git pull
```

Branches & Deployment

- Feature branches should be short-lived
  - otherwise, drift out of sync with master, and hard to reconcile
  - `git rebase` can be used to “incrementally” merge
  - `git cherry-pick` can be used to merge only specific commits
- “Deploy from master” is most common

Release/bugfix branches and cherry-picking commits



Rationale: release branch is a stable place to do incremental bug fixes

Branch vs. Fork

- Git supports *fork & pull* collaboration model
- If you have push/admin access on repo:
  - branch: create branch in *this repo*
  - merge: fold branch changes into master (or into another branch)
  - Create a pull request from the branch for code review rather than folding changes to master
- If you don't:
  - fork: clone *entire repo* on GitHub to one that you can branch, push, etc.
  - Finalize your work on its own branch
  - pull request asks owner of original repo to pull specific commits from my forked repo

If separate sub-teams are assigned to work on *release bug fixes* and *new features,* you will need to use:

- ☐ Branch per release
- ☐ Branch per feature
- ☐ Branch per release + Branch per feature
- ☐ Any of these will work