Peaceful conflict resolution in version control systems

`m.strova.dk/git`

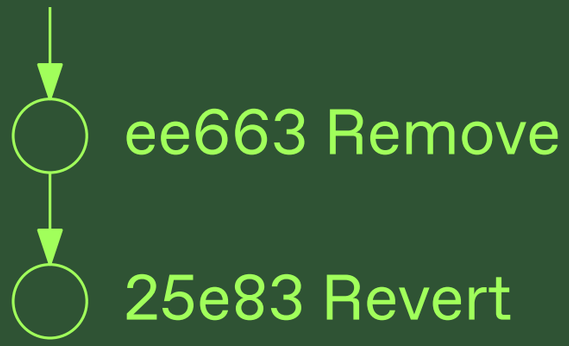
Mathias Rav
mathias@scalgo.com

SCALGO

Here’s a thing you can do

```
$ git commit -am ’Remove feature’
[parallelsort ee66395c] Remove feature
1 file changed, 8 insertions(+), 52 deletions(-)

$ git revert @
[parallelsort 25e83e1b] Revert ”Remove feature”
1 file changed, 52 insertions(+), 8 deletions(-)
```

Why though?



SCALGO

Let’s create space for water

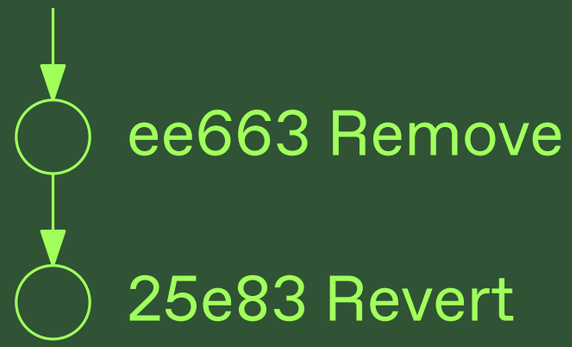
Here’s a thing you can do

```
$ git commit -am ’Remove feature’
[parallelsort ee
1 file changed,

$ git revert @
[parallelsort 25
1 file changed,
```

Why though?

Goal of this talk

- New workflow for git branches
- No more merge conflict markers
- Make “commit-and-revert” make sense



SCALGO

Let’s create space for water

About me

- Computer science PhD from AU (2019)

SCALGO

Let’s create space for water

About me

- Computer science PhD from AU (2019)

  — Advisor: Lars Arge (1967–2020)

- Software developer at Scalgo (2019–)

  — Co-founded by Lars Arge

- What does Scalgo do?

SCALGO

Let's create space for water

Scalgo Live (briefly)

- Web-based platform sold by Scalgo

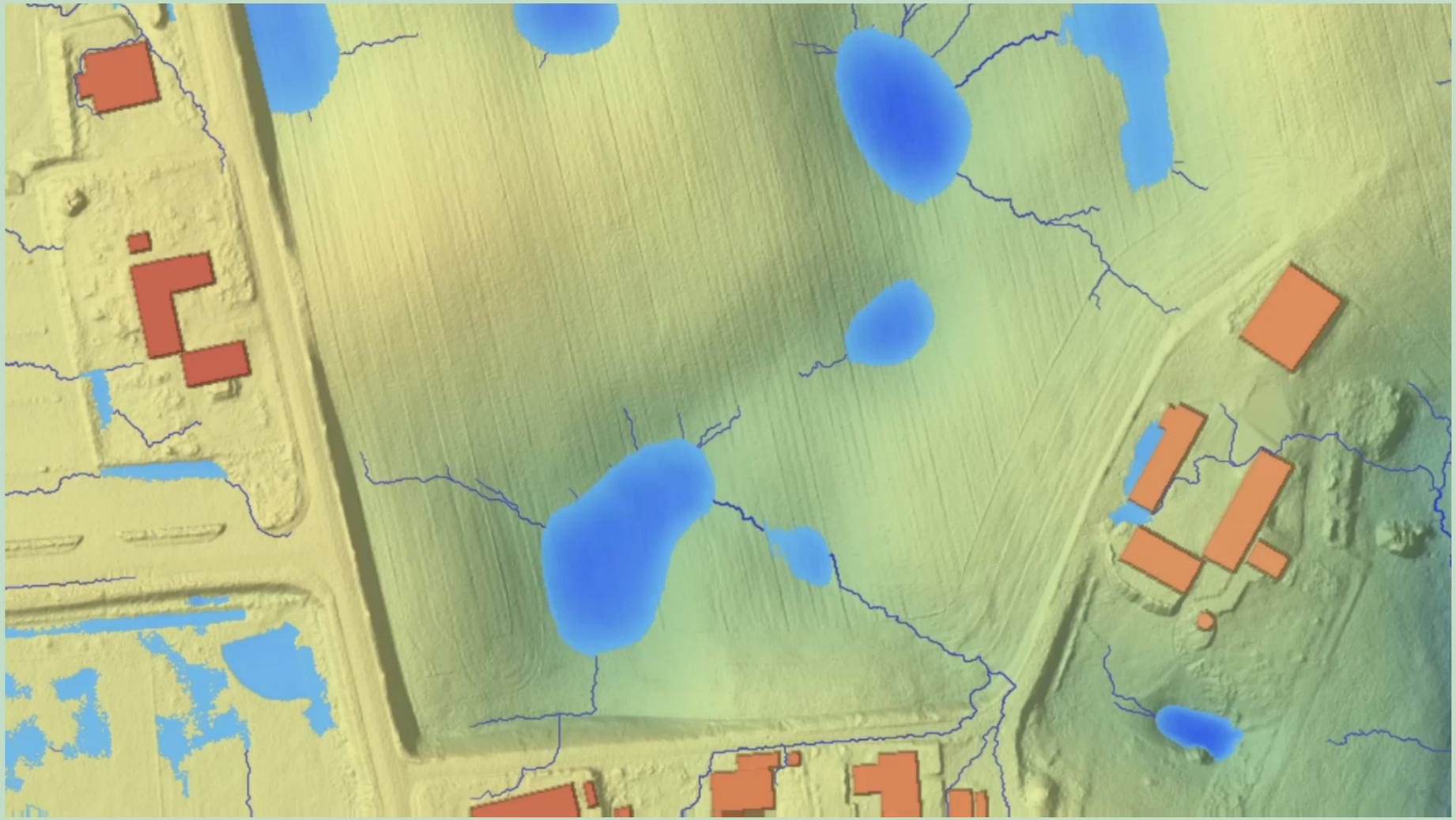
SCALGO

Let’s create space for water

- Precomputed analysis

SCALGO

Let’s create space for water



- Precomputed analysis
- Zoom in anywhere

SCALGO

Let’s create space for water

- Precomputed analysis
- Zoom in anywhere
- Plan your projects

SCALGO

Let’s create space for water

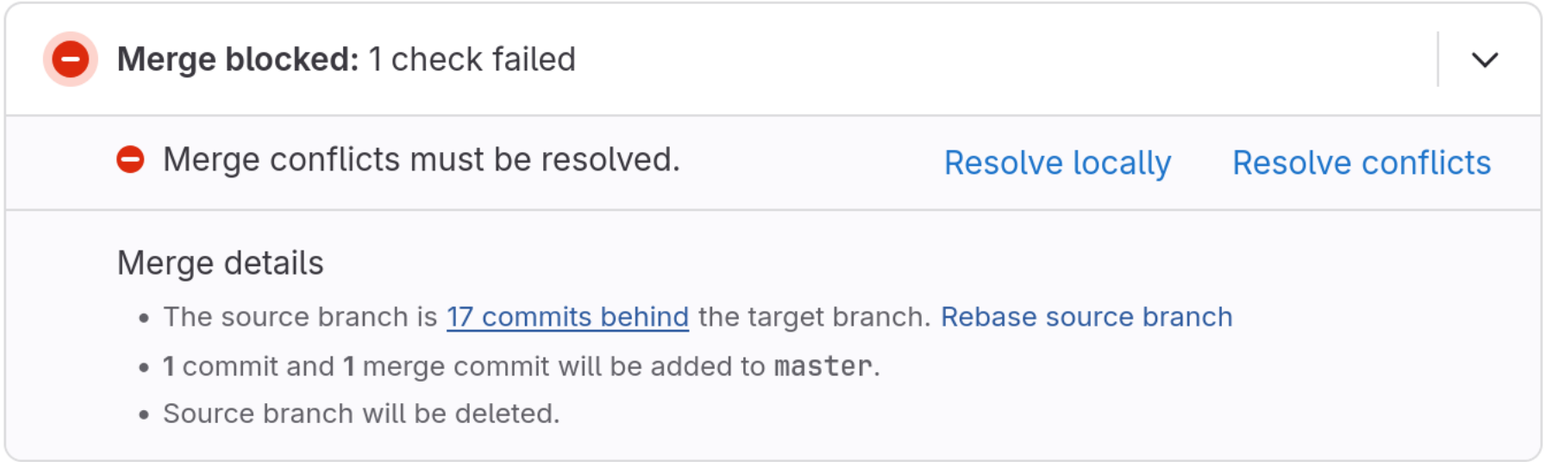
- Precomputed analysis
- Zoom in anywhere
- Plan your projects
- Get your feet wet

SCALGO

Let’s create space for water

What's a merge conflict?

Has this ever happened to you?



Automatic merge failed; fix conflicts and then commit the result

```
Auto-merging annotate.py
CONFLICT (content): Merge conflict in annotate.py
CONFLICT (modify/delete): cpp.py deleted in HEAD and modified in 0b49e1b (Implement docstring passthrough).  Version 0b49e1b (Implement docstring passthrough) of cpp.
Auto-merging parser.py
CONFLICT (content): Merge conflict in parser.py
CONFLICT (modify/delete): test/base.spr deleted in HEAD and modified in 0b49e1b (Implement docstring passthrough).  Version 0b49e1b (Implement docstring passthrough)
error: could not apply 0b49e1b... Implement docstring passthrough
hint: Resolve all conflicts manually, mark them as resolved with
hint: "git add/rm <conflicted_files>", then run "git rebase --continue".
hint: You can instead skip this commit: run "git rebase --skip".
hint: To abort and get back to the state before "git rebase", run "git rebase --abort".
hint: Disable this message with "git config set advice.mergeConflict false"
Could not apply 0b49e1b... # Implement docstring passthrough
```

SCALGO

Let's create space for water

What's a merge conflict?

Has this ever happened to you?

```
                    elif t2.type == TokenType.IDENTIFIER:
                        self.checkToken(self.token, [TokenType.LBRACE, TokenType.COLON])
                        if self.token.type == TokenType.COLON:
                            self.consumeToken([TokenType.COLON])
                            type_ = self.consumeToken([TokenType.IDENTIFIER])
                            members.append(Value(self.token, t2, None, type_, None, None, doccomment2))
                        else:
<<<<<<< HEAD
                            members.append(Table(t, t2, None, self.parseContent()))
=======
                            members.append(Table(t2, t2, None, self.parseContent(), doccomment2))
                        doccomment2 = None
>>>>>>> 0b49e1b (Implement docstring passthrough)
                    else:
                        assert(False)
                    if self.token.type in [TokenType.COMMA, TokenType.SEMICOLON]:
                        self.nextToken()
```

Implement docstring passthrough) of cpp.

```
CONFLICT (content): Merge conflict in parser.py
CONFLICT (modify/delete): test/base.spr deleted in HEAD and modified in 0b49e1b (Implement docstring passthrough).  Version 0b49e1b (Implement docstring passthrough)
error: could not apply 0b49e1b... Implement docstring passthrough
hint: Resolve all conflicts manually, mark them as resolved with
hint: "git add/rm <conflicted_files>", then run "git rebase --continue".
hint: You can instead skip this commit: run "git rebase --skip".
hint: To abort and get back to the state before "git rebase", run "git rebase --abort".
hint: Disable this message with "git config set advice.mergeConflict false"
Could not apply 0b49e1b... # Implement docstring passthrough
```

SCALGO

Let's create space for water

What's a merge conflict?

Has this ever happened to you?

Someone edited the same file as me!
How dare they!

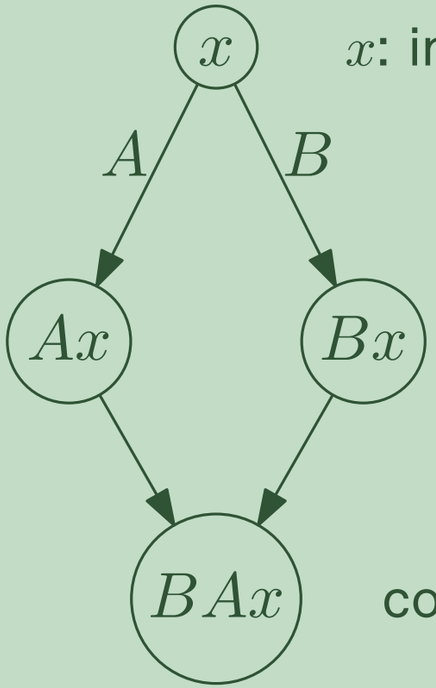
```
                    elif t2.type == TokenType.IDENTIFIER:
                        s
                        i
                            members.append(Table(t, t2, None, self.parseContent()))
<<<<<<< HEAD
=======
                            members.append(Table(t2, t2, None, self.parseContent(), doccomment2))
                        doccomment2 = None
>>>>>>> 0b49e1b (Implement docstring passthrough)
                    else:
                        assert(False)
                    if self.token.type in [TokenType.COMMA, TokenType.SEMICOLON]:
                        self.nextToken()
```

```
Implement docstring passthrough) of cpp.

CONFLICT (content): Merge conflict in parser.py
CONFLICT (modify/delete): test/base.spr deleted in HEAD and modified in 0b49e1b (Implement docstring passthrough).  Version 0b49e1b (Implement docstring passthrough)
error: could not apply 0b49e1b... Implement docstring passthrough
hint: Resolve all conflicts manually, mark them as resolved with
hint: "git add/rm <conflicted_files>", then run "git rebase --continue".
hint: You can instead skip this commit: run "git rebase --skip".
hint: To abort and get back to the state before "git rebase", run "git rebase --abort".
hint: Disable this message with "git config set advice.mergeConflict false"
Could not apply 0b49e1b... # Implement docstring passthrough
```

SCALGO
Let's create space for water

What's a merge?

- Two developers make different changes, A and B
- We want to apply both changes to the codebase



x: initial codebase contents

A, B: changes we want to apply

Ax, Bx: codebase with changes applied

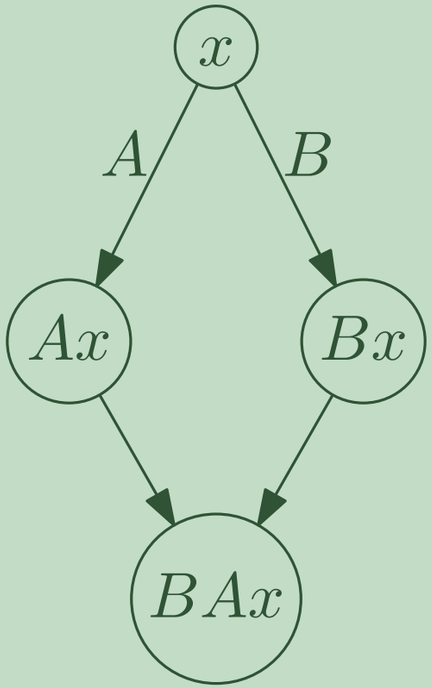
codebase with both A and B applied

SCALGO

Let's create space for water

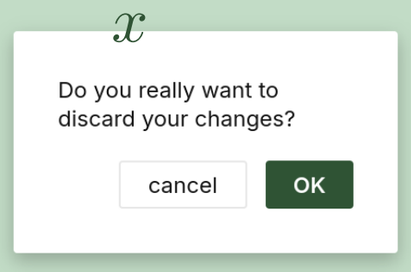
What's a merge?

- Two developers make different changes, A and B
- We want to apply both changes to the codebase

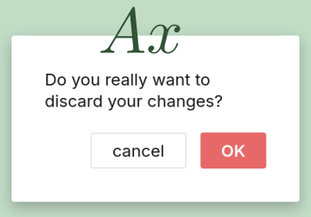


Example 1:

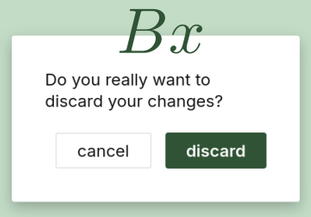
x: app with this dialog



A: make button red

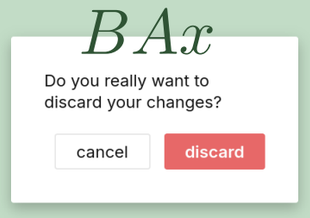


B: change label to "discard"



goal: apply both changes

Is it possible? Yes!

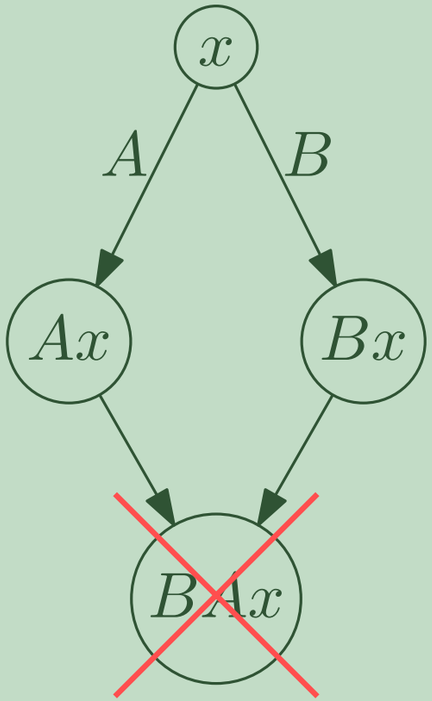


SCALGO

Let's create space for water

What's a merge?

- Two developers make different changes, A and B
- We want to apply both changes to the codebase



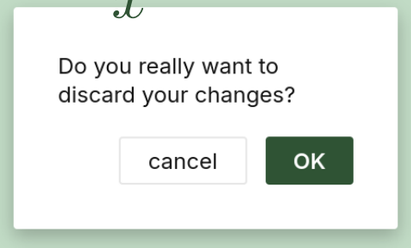
Example 2:

x: app with this dialog

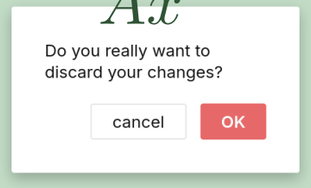
A: make button red

B: make button blue

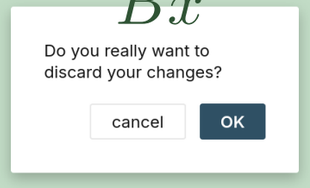
x



Ax



Bx



conflict: Not possible to do both

SCALGO

Let's create space for water

What's a merge?

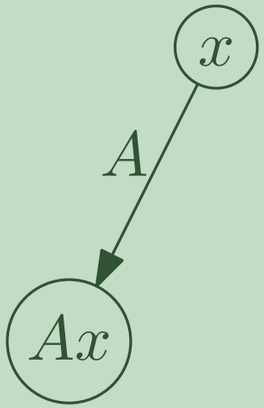
- Two developers make different changes, A and B
- We want to apply both changes to the codebase

x

```
def check_fizzbuzz(n, strict=False):
    if n % 3 == 0 or n % 5 == 0:
        print("WARNING: number not safe for printing")
        if strict:
            raise Exception("unsafe number")

check_fizzbuzz(16, strict=True)
n = int(input("Type a number: "))
check_fizzbuzz(n)

def check_numbers(numlist):
    for n in numlist:
        check_fizzbuzz(n)
```

Example 3:

A: Change default `strict` to `True`



Ax

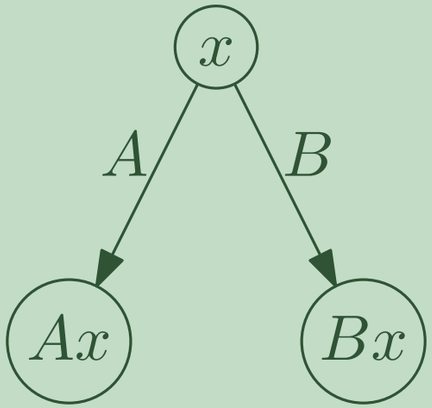
```
def check_fizzbuzz(n, strict=True):
    if n % 3 == 0 or n % 5 == 0:
        print("WARNING: number not safe for printing")
        if strict:
            raise Exception("unsafe number")

check_fizzbuzz(16, strict=True)
n = int(input("Type a number: "))
check_fizzbuzz(n, strict=False)

def check_numbers(numlist):
    for n in numlist:
        check_fizzbuzz(n, strict=False)
```

SCALGO

Let's create space for water

What's a merge?

- Two developers make different changes, A and B
- We want to apply both changes to the codebase



Example 3:

A: Change default `strict` to `True`

B: Add `strict` to `check_numbers`

x

```
def check_fizzbuzz(n, strict=False):
    if n % 3 == 0 or n % 5 == 0:
        print("WARNING: number not safe for printing")
        if strict:
            raise Exception("unsafe number")

check_fizzbuzz(16, strict=True)
n = int(input("Type a number: "))
check_fizzbuzz(n)

def check_numbers(numlist):
    for n in numlist:
        check_fizzbuzz(n)
```

Ax

```
def check_fi
    if n % 3
        prin
        if s


check_fizzbu
n = int(inpu
check_fizzbu

def check_nu
    for n in
        chec
```

Bx

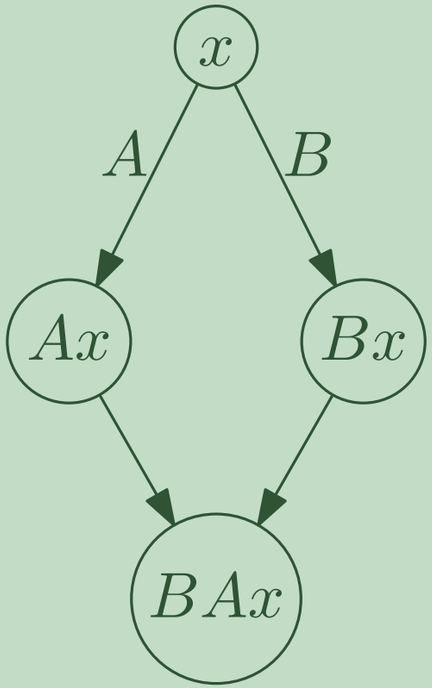
```
def check_fizzbuzz(n, strict=False):
    if n % 3 == 0 or n % 5 == 0:
        print("WARNING: number not safe for printing")
        if strict:
            raise Exception("unsafe number")

check_fizzbuzz(16, strict=True)
n = int(input("Type a number: "))
check_fizzbuzz(n)

def check_numbers(numlist, strict=False):
    for n in numlist:
        check_fizzbuzz(n, strict=strict)
```

SCALGO

Let's create space for water

What's a merge?

- Two developers make different changes, A and B
- We want to apply both changes to the codebase



Example 3:

A: Change default `strict` to T

B: Add `strict` to `check_num`

Conflict or no conflict?

git is not happy...

fizzbuzz.py

```
def check_fizzbuzz(n, strict=True):
    if n % 3 == 0 or n % 5 == 0:
        print("WARNING: number not safe for printing")
        if strict:
            raise Exception("unsafe number")

check_fizzbuzz(16, strict=True)
n = int(input("Type a number: "))
check_fizzbuzz(n, strict=False)

def check_numbers(numlist, strict=False):
    for n in numlist:
<<<<<<< HEAD
        check_fizzbuzz(n, strict=strict)
=======
        check_fizzbuzz(n, strict=False)
>>>>>>> stricttrue
```

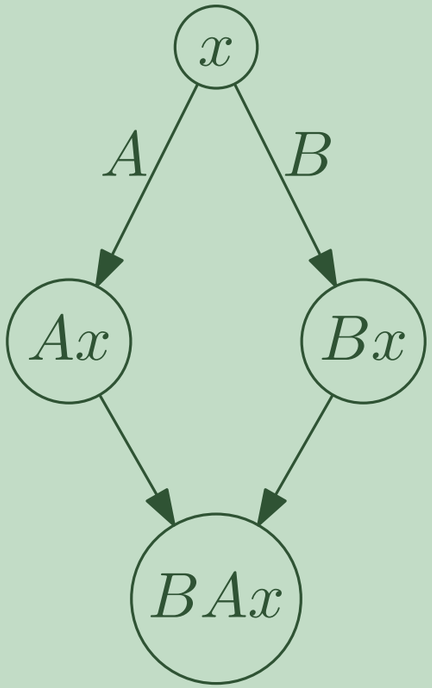
- Must "pick correct version"
- ... and find other errors

SCALGO

Let's create space for water

What's a merge?

- Two developers make different changes, A and B
- We want to apply both changes to the codebase



Example 3:

A: Change default `strict` to `True`

B: Add `strict` to `check_numbers`

x

```
def check_fizzbuzz(n, strict=False):
    if n % 3 == 0 or n % 5 == 0:
        print("WARNING: number not safe for printing")
        if strict:
            raise Exception("unsafe number")

check_fizzbuzz(16, strict=True)
n = int(input("Type a number: "))
check_fizzbuzz(n)

def check_numbers(numlist):
    for n in numlist:
        check_fizzbuzz(n)
```

Ax

```
def check_fizzbuzz(n, strict=True):
    if n % 3 == 0 or n % 5 == 0:
        print("WARNING: number not safe for printing")
        if strict:
            raise Exception("unsafe number")

check_fizzbuzz(16, strict=True)
n = int(input("Type a number: "))
check_fizzbuzz(n, strict=False)

def check_numbers(numlist):
    for n in numlist:
        check_fizzbuzz(n, strict=False)
```

...but it's possible:

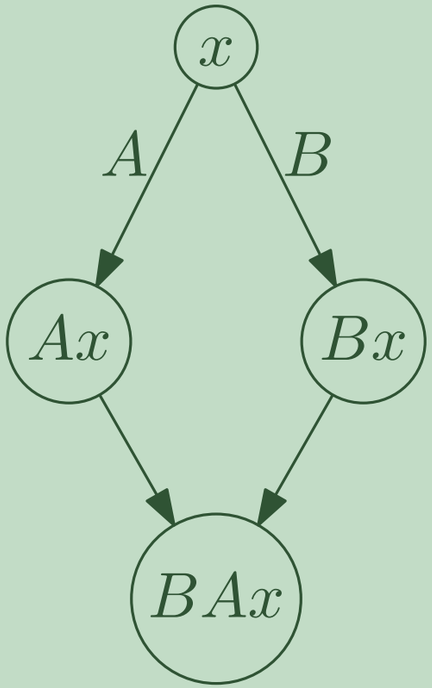
Just apply A, then apply B

SCALGO

Let's create space for water

What's a merge?

- Two developers make different changes, A and B
- We want to apply both changes to the codebase



Example 3:

A: Change default `strict` to `True`

B: Add `strict` to `check_numbers`

x

```
def check_fizzbuzz(n, strict=False):
    if n % 3 == 0 or n % 5 == 0:
        print("WARNING: number not safe for printing")
        if strict:
            raise Exception("unsafe number")

check_fizzbuzz(16, strict=True)
n = int(input("Type a number: "))
check_fizzbuzz(n)

def check_numbers(numlist):
    for n in numlist:
        check_fizzbuzz(n)
```

BAx

```
def check_fizzbuzz(n, strict=True):
    if n % 3 == 0 or n % 5 == 0:
        print("WARNING: number not safe for printing")
        if strict:
            raise Exception("unsafe number")

check_fizzbuzz(16~~, strict=True~~)
n = int(input("Type a number: "))
check_fizzbuzz(n, strict=False)

def check_numbers(numlist, strict=True):
    for n in numlist:
        check_fizzbuzz(n, strict=strict)
```

...but it's possible:

Just apply A, then apply B

SCALGO

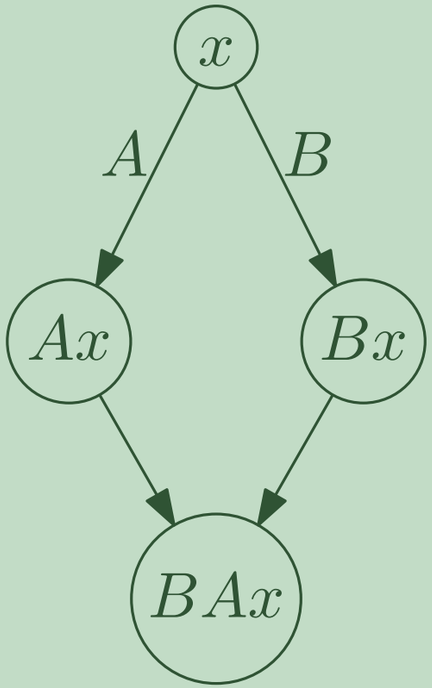
Let's create space for water

What's a merge?

- Two developers make different changes, A and B
- We want to apply both changes to the codebase

x

```
def check_fizzbuzz(n, strict=False):
    if n % 3 == 0 or n % 5 == 0:
        print("WARNING: number not safe for printing")
        if strict:
            raise Exception("unsafe number")

check_fizzbuzz(16, strict=True)
n = int(input("Type a number: "))
check_fizzbuzz(n)

def check_numbers(numlist):
    for n in numlist:
        check_fizzbuzz(n)
```



Example 3:

A: Change default `strict` to `True`

B: Add `strict` to `check_numbers`

A and B have a syntactic conflict
but there is no semantic conflict

"modify the same lines of code"

"change the same bit of behavior"
(e.g. change the color of the same button)

SCALGO
Let's create space for water

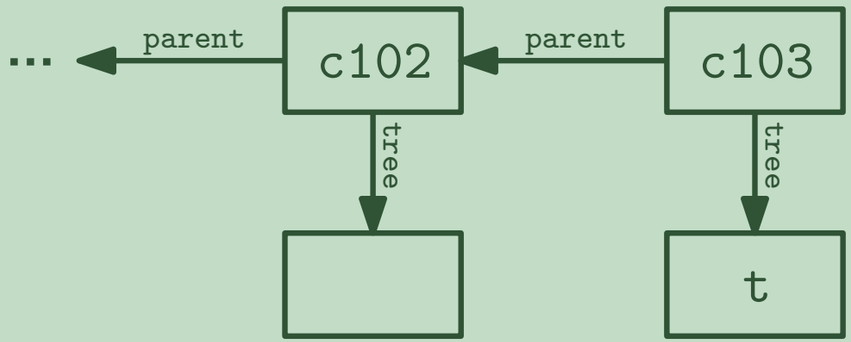
The git object graph

- git operates on so-called `tree` and `commit` objects
- A `tree` in git is a collection of files with contents, e.g.

```
t = {"file1.txt": "Hello world\nThis is the first file\n",
     "file2.txt": "This is the second file"}
```

- A `commit` in git is a point in a codebase's history

```
c103 = {"tree": t,
        "message": "Add file2.txt",
        "author": "Mathias <m@git.strova.dk> 2025-10-26 17:03:14 GMT",
        "parent": [c102]}
```



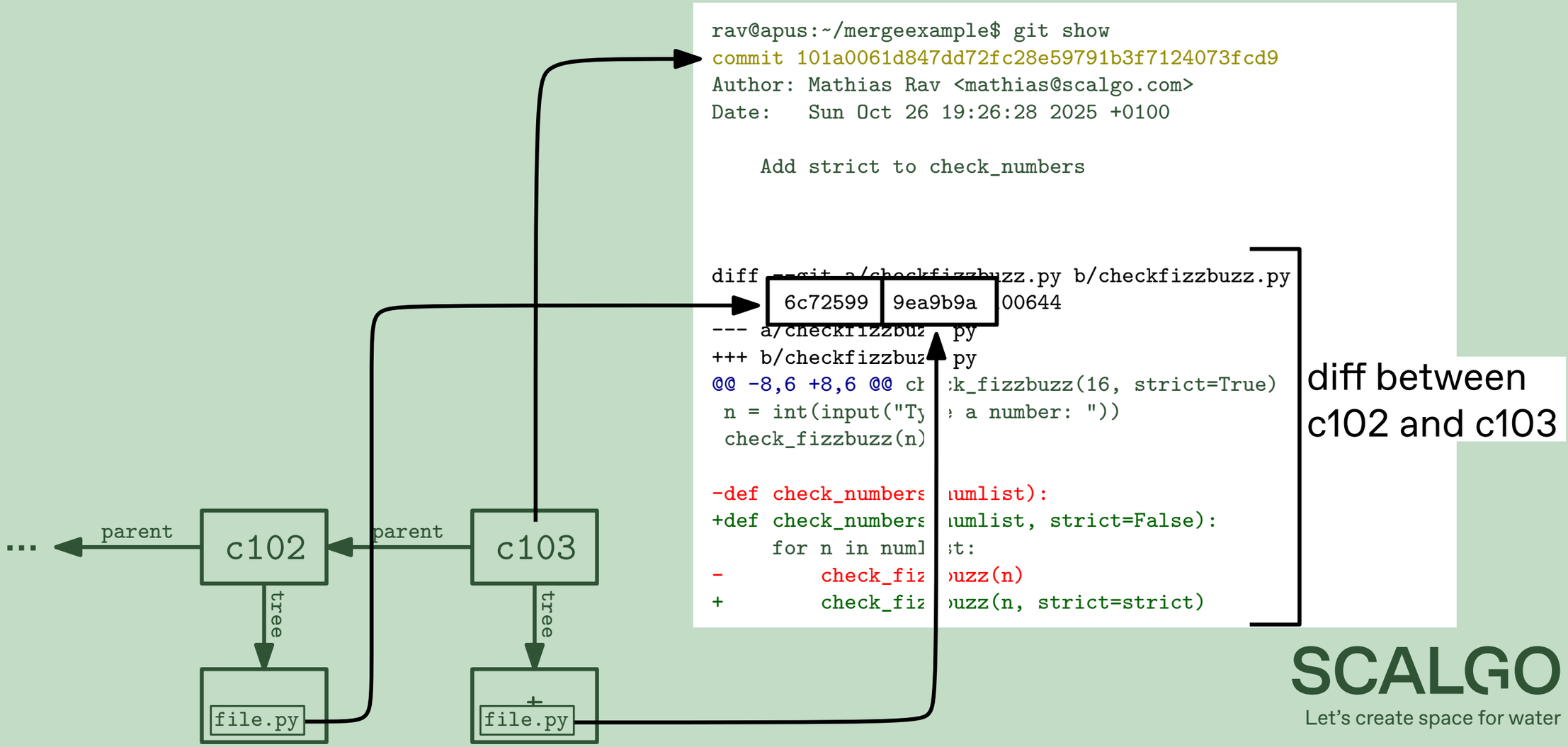
Merkle tree for content-based addressing:
Each object's id is a hash $h(\ldots)$ of its contents

$$h(\text{commit}) = h(\text{tree}) \oplus h(\text{message}) \oplus h(\text{author}) \oplus h(\text{parent})$$

SCALGO
Let's create space for water

The git object graph

- Each commit represents the change from its parent's tree to its own tree

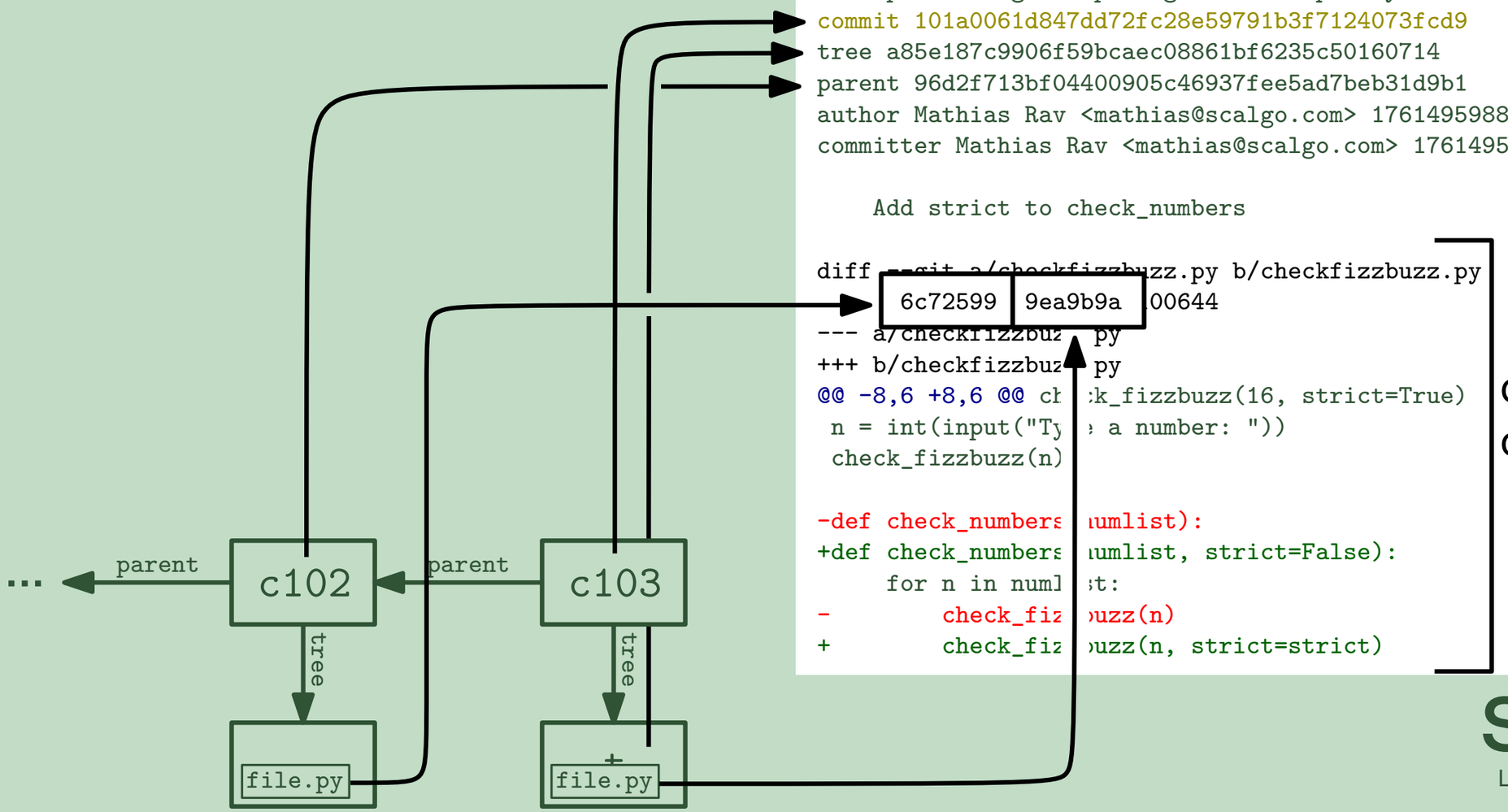


SCALGO

Let's create space for water

The git object graph

- Each commit represents the change from its parent's tree to its own tree

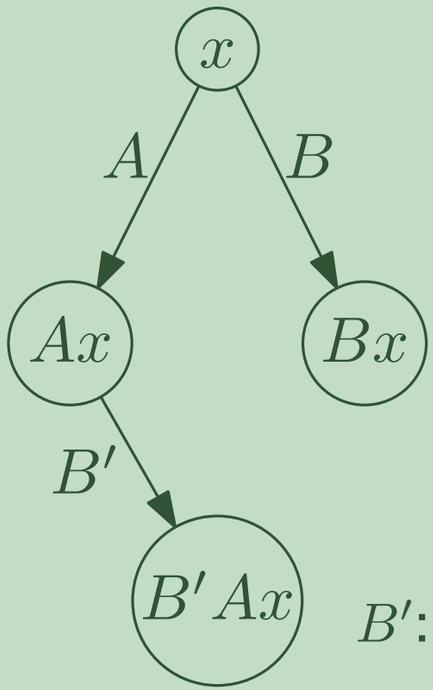


SCALGO

Let's create space for water

The git object graph

- Each commit represents the change from its parent's tree to its own tree
- A merge BAx is an attempt to apply diff of B on top of the codebase Ax



`git` operates on real files in the real world ...

... but programmers like to think about features and behaviors

$\varphi(x)$: an app with this and that button

x: a codebase with foo.c and bar.py

$\varphi(A)$: change the color of this button
$\varphi(B)$: move the button to the left side

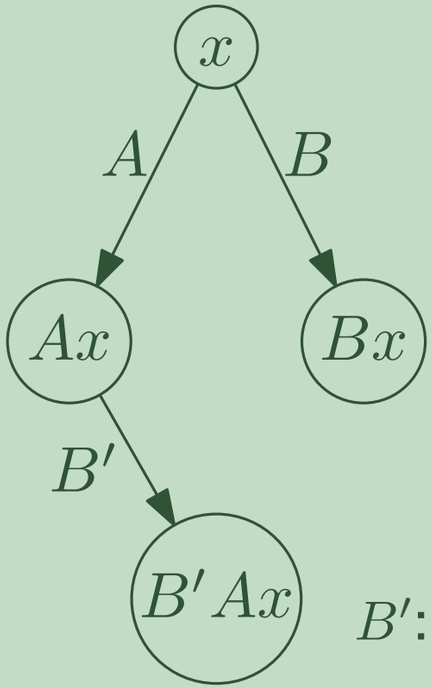
A: change these class methods
B: change those function parameters

B': Same change as B, but on top of Ax instead of x

SCALGO
Let's create space for water

The git object graph

- Each commit represents the change from its parent's tree to its own tree
- A merge BAx is an attempt to apply diff of B on top of the codebase Ax



`git` operates on real files in the real world ...

... but programmers like to think about features and behaviors

φ : Concrete programs $\rightarrow$ Abstract programs

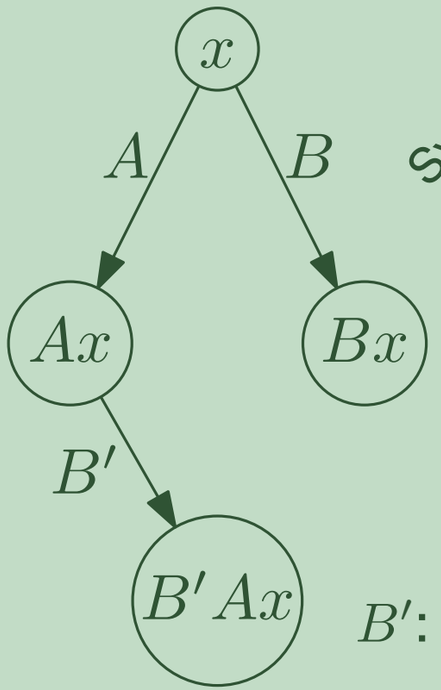
Conflicts in concrete program $\neq$ Conflicts in abstract programs

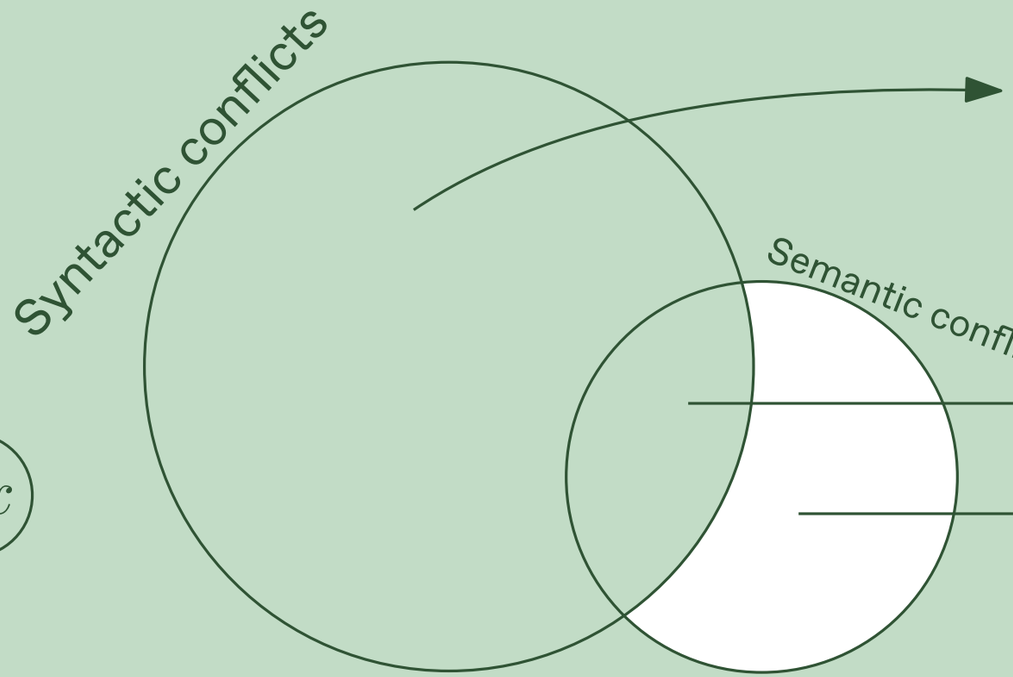
B': Same change as B, but on top of Ax instead of x

SCALGO
Let's create space for water

The git object graph

- Each commit represents the change from its parent's tree to its own tree
- A merge BAx is an attempt to apply diff of B on top of the codebase Ax





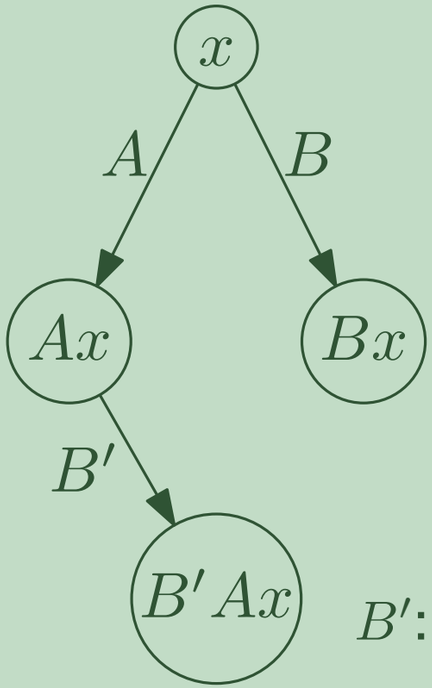
- can be merged, but not automatically
- cannot be merged
- will merge and then fail checks (or break prod if you're unlucky)

B': Same change as B, but on top of Ax instead of x

SCALGO
Let's create space for water

The git object graph

- Each commit represents the change from its parent's tree to its own tree
- A merge BAx is an attempt to apply diff of B on top of the codebase Ax



Casual git notation: changes and states omitted

B': Same change as B, but on top of Ax instead of x

SCALGO
Let's create space for water

Remainder of this talk

- The business perspective on merge conflicts
- Refresher: The existing `git rebase` toolbox
- New tool: The Hammer (commit-and-revert)

Business perspective (or lack thereof)

Two options to handle a merge conflict:

- Go over all merge conflict markers
- Throw out the branch and start over

Tedious and error-prone, unusual task

```
@@ -165,13 +169,17 @@ class AMI_stack {
165 169       ///
166 170       ////////////////////////////////////////////////////////////////////
167 171
168     -     AMI_err main_memory_usage(TPIE_OS_SIZE_T *usage,
169     -             MM_stream_usage usage_type) const;
    172 +     err main_memory_usage(TPIE_OS_SIZE_T *usage,
    173 +             MM_stream_usage usage_type) const;
170 174
171 175
172 176       //  This should go as soon as all old code has been migrated.
173 177       TPIE_OS_OFFSET stream_len() const {
    178 + <<<<<<< .mine
    179 +     cerr << "Using stack<T>::stream_len() is deprecated." << endl;
    180 + =======
174 181       std::cerr << "Using AMI_stack<T>::stream_len() is deprecated." << std::endl;
    182 + >>>>>>> .r1580
175 183       return m_size;
176 184       }
```

Programmer accidentally included conflict markers in commit

SCALGO

Let's create space for water

Business perspective (or lack thereof)

Two options to handle a merge conflict:

- Go over all merge conflict markers
- Throw out the branch and start over

Tedious and error-prone, unusual task

```
@@ -165,13 +169,17 @@ class AMI_stack {
165 169       ///
166 170       ////////////////////////////////////////////////////////////////////
167 171
168     -     AMI_err main_memory_usage(TPIE_OS_SIZE_T *usage,
169     -             MM_stream_usage usage_type) const;
    172 +     err main_memory_usage(TPIE_OS_SIZE_T *usage,
    173 +             MM_stream_usage usage_type) const;
170 174
171 175
172 176       //  This should go as soon as all old code has been migrated.
173 177       TPIE_OS_OFFSET stream_len() const {
    178 + <<<<<<< .mine
    179 +     cerr << "Using stack<T>::stream_len() is deprecated." << endl;
    180 + =======
174 181       std::cerr << "Using AMI_stack<T>::stream_len() is deprecated." << std::endl;
    182 + >>>>>>> .r1580
175 183       return m_size;
176 184       }
```

Boring code monkey task

By Takeshi Miyazawa, from comicsalliance.com
©2012 Pak Man Productions and 10 Print JoCo

SCALGO
Let's create space for water

Business perspective (or lack thereof)

Two options to handle a merge conflict:

- Go over all merge conflict markers
- Throw out the branch and start over

Not fun, not a good use of time

Factors contributing to merge conflicts:

- Long-lived feature branches
- Large feature branches

Today's goals:

- Tools to keep feature branches small
- Tools to solve merge conflicts in more fun ways
- Tools to reorganize branch commits effectively

SCALGO

Let's create space for water

Refresher: `git rebase`

- Core function: replay a sequence of commits or commands on top of a base commit
- Commits fail to apply cleanly ⇒ merge conflict

  — User can fix merge conflicts manually (edit `; git add ; git rebase --continue`)

  — … or give up and type `git rebase --abort`
- Merge conflicts in `git rebase` can often snowball

  ⇒ usually best to admit defeat, give up, and try something else
- Remember to squash before rebasing
- What do you actually use `git rebase` for?

Refresher: `git rebase`

Basically two workflows:

1. Reapply commits from one base commit to another
   — `git rebase` *oldbase* `--onto` *newbase*
   — `git rebase` *newbase*

2. Edit a series of commits
   — `git rebase -i @~`n (edit most recent n commits)
   — `git rebase -i` *oldbase* (edit all commits since *oldbase*)
   — `git rebase -i` `` `git merge-base @` *newbase*`` (edit commits not in *newbase*)

It is possible, but rarely a good idea, to do both:

- Edit commits and reapply onto another base commit
  — `git rebase -i` *oldbase* `--onto` *newbase*
  — `git rebase -i` *newbase*

Refresher: `git rebase`

- `git rebase -i` first asks the user to plan before it applies the plan

```
# Rebase 7ff43ce..6b2b60c onto 7ff43ce (5 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup [-C] <commit> = like "squash" but keep only the previous
#                  commit's log message, unless -C is used, in which case
#                  keep only this commit's message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
```

- Note: Only difference between `fixup`, `fixup -C` and `squash` is the commit message
- `fixup`: use first commit/”squash up”; `fixup -C`: use second commit/”squash down”
- `squash`: open editor to combine first and second commit message

SCALGO

Let’s create space for water

Refresher: `git rebase`

- `git rebase -i` first asks the user to plan before it applies the plan

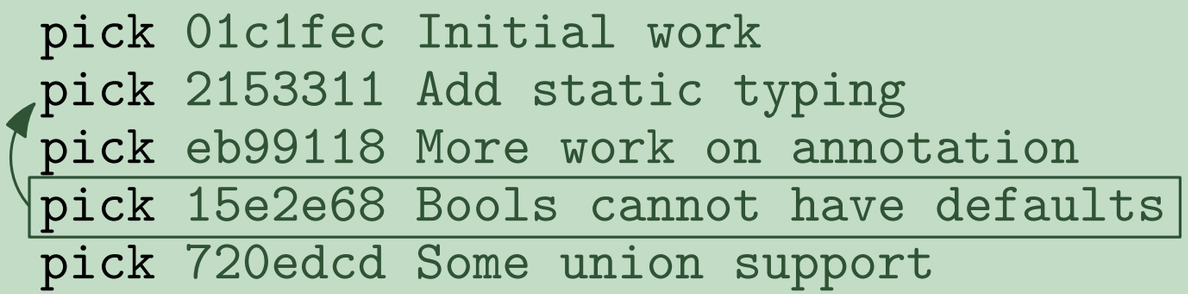
```
pick 01c1fec Initial work
pick 2153311 Add static typing
pick eb99118 More work on annotation
pick 15e2e68 Bools cannot have defaults
pick 720edcd Some union support
```

SCALGO

Let's create space for water

Refresher: `git rebase`

- `git rebase -i` first asks the user to plan before it applies the plan



Combine commit with a previous commit

SCALGO

Let's create space for water

Refresher: `git rebase`

- `git rebase -i` first asks the user to plan before it applies the plan

```
pick 01c1fec Initial work
pick 2153311 Add static typing
fixup 15e2e68 Bools cannot have defaults
pick eb99118 More work on annotation
pick 720edcd Some union support
```

Combine commit with a previous commit

Move line and change `pick` to `fixup`

Common tactics when using `git rebase -i` to edit a series of commits:

- Drop a commit. (Remove `pick` lines)
- Reorder commits. (Move `pick` lines around)
- Combine commits. (Change `pick` to `fixup`)
- Change commit message. (Change `pick` to `reword`)
- Split a commit. (Change `pick` to `edit`, then use `git reset`, `git add -p`)

SCALGO

Let's create space for water

Refresher: `git rebase`

Conflicts usually mean the rebase failed, requiring `git rebase --abort`

First rule of `git rebase -i`:

- Do one thing at a time!

— Failed rebases are frustrating if you planned a lot of changes
— Failed rebases with only one change can quickly be retried

Common tactics when using `git rebase -i` to edit a series of commits:

- Drop a commit. (Remove `pick` lines)
- Reorder commits. (Move `pick` lines around)

Can potentially cause conflicts

- Combine commits. (Change `pick` to `fixup`)
- Change commit message. (Change `pick` to `reword`)
- Split a commit. (Change `pick` to `edit`, then use `git reset`, `git add -p`)

No conflicts(!)

- Why is it only the first two bullets that can cause conflicts?

Let's think about those `tree` objects...

SCALGO
Let's create space for water

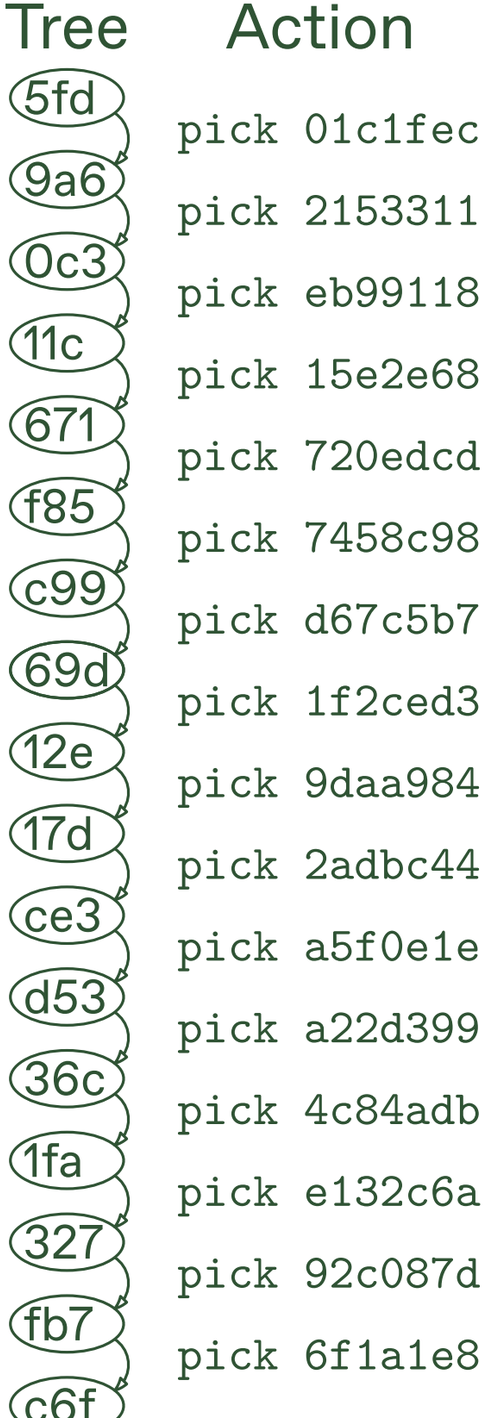
- pick 01c1fec Initial work
- pick 2153311 Add static typing
- pick eb99118 More work on annotation
- pick 15e2e68 Bools cannot have defaults
- pick 720edcd Some union support
- pick 7458c98 Some work on cpp output
- pick d67c5b7 Some more annotating
- pick 1f2ced3 More work on cpp input
- pick 9daa984 Implement and test numerics
- pick 2adbc44 Cleanup
- pick a5f0e1e Add support for members
- pick a22d399 Add support for text
- pick 4c84adb Add support for pods
- pick e132c6a Add support for lists
- pick 92c087d Add support for text
- pick 6f1a1e8 Finish up list support

- `pick 01c1fec Initial work`
- `pick 2153311 Add static typing`
- `pick eb99118 More work on anno`
- `pick 15e2e68 Bools cannot have`
- `pick 720edcd Some union support`
- `pick 7458c98 Some work on cpp output`
- `pick d67c5b7 Some more annotating`
- `pick 1f2ced3 More work on cpp input`
- `pick 9daa984 Implement and test numerics`
- `pick 2adbc44 Cleanup`
- `pick a5f0e1e Add support for members`
- `pick a22d399 Add support for text`
- `pick 4c84adb Add support for pods`
- `pick e132c6a Add support for lists`
- `pick 92c087d Add support for text`
- `pick 6f1a1e8 Finish up list support`

Useful to think about the `tree` between each line

`tree`: git concept for file and directory contents

Trees have object ids (hashes), just like commits



Useful to think about the `tree` between each line

`tree`: git concept for file and directory contents

Trees have object ids (hashes), just like commits

Conflicts happen when picked commit cannot be applied to current tree

Splitting/combining results in the same trees

Dropping/reordering results in different trees

⇒ Dropping/reordering can cause conflicts
... except for dropping the last commit

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

Here's another thing in `git rebase -i` that can never fail:

- Commit an arbitrary change and immediately revert it
  `git commit -am WIP && git revert @`

```
pick 01c1fec
pick 2153311
pick eb99118
break
pick 15e2e68
pick 720edcd
pick 7458c98
```

```
$ git rebase -i base
(opens editor)
Stopped at eb99118 (More work on annotation)
$ edit src/foo.c
$ git commit -am 'Delete feature'
[detached HEAD a8bec8e] Delete feature
 1 file changed, 6 deletions(-)
$ git revert @
[detached HEAD 35fae16] Revert "Delete feature"
 1 file changed, 6 insertions(+)
$ git rebase --continue
Successfully rebased and updated refs/heads/featurebranch.
```

SCALGO

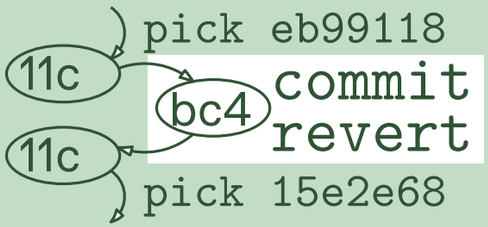
Let's create space for water

New tool in the toolbox: The Hammer

Here's another thing in `git rebase -i` that can never fail:

- Commit an arbitrary change and immediately revert it
  `git commit -am WIP && git revert @`

```
pick 01c1fec
pick 2153311
pick eb99118
commit
revert
pick 15e2e68
pick 720edcd
pick 7458c98
```



What happened during the `break`?

commit, then revert ⇒ same `tree` as before

Okay, but when would you actually do this?

Let's look at some use cases

SCALGO

Let's create space for water

New tool in the toolbox: The Hammer

Example: Splitting refactors from feature-additions

- Commit an arbitrary change and immediately revert it

```
pick eb99118 Refactor and add feature
break  # commit, then revert
```

New tool in the toolbox: The Hammer

Example: Splitting refactors from feature-additions

- Commit an arbitrary change and immediately revert it

```
pick eb99118 Refactor and add feature
pick 58f52a0 Remove feature
pick 0615bbc Revert "Remove feature"
```

SCALGO

Let’s create space for water

New tool in the toolbox: The Hammer

Example: Splitting refactors from feature-additions

- Commit an arbitrary change and immediately revert it

```
pick eb99118 Refactor and add feature
squash 58f52a0 Remove feature
reword 0615bbc Revert "Remove feature"
```

New tool in the toolbox: The Hammer

Example: Splitting refactors from feature-additions

- Commit an arbitrary change and immediately revert it

```
pick 2934552 Refactor

pick 12759ba Add feature
```

- Useful when `git reset` + `git add -p` is not enough
- Allows refactors to be merged early

New tool in the toolbox: The Hammer

Example: Splitting refactors from feature-additions

- Commit an arbitrary change and immediately revert it

```
pick 2934552 Refactor
break
pick 12759ba Add feature
```

- Can be repeated to move more changes from first to second commit

SCALGO

Let's create space for water

New tool in the toolbox: The Hammer

Example: Splitting refactors from feature-additions

- Commit an arbitrary change and immediately revert it

```
pick 2934552 Refactor
pick 6d5b167 WIP
pick a29cbd7 Revert "WIP"
pick 12759ba Add feature
```

- Can be repeated to move more changes from first to second commit

SCALGO

Let's create space for water

New tool in the toolbox: The Hammer

Example: Splitting refactors from feature-additions

- Commit an arbitrary change and immediately revert it

squash up

```
pick 2934552 Refactor
fixup 6d5b167 WIP
```

squash down

```
pick a29cbd7 Revert "WIP"
fixup -C 12759ba Add feature
```

- Can be repeated to move more changes from first to second commit
- Recall: Only difference between `fixup`, `fixup -C` and `squash` is the commit message
- `fixup`: use first commit/"squash up"; `fixup -C`: use second commit/"squash down"
- `squash`: open editor to combine first and second commit message

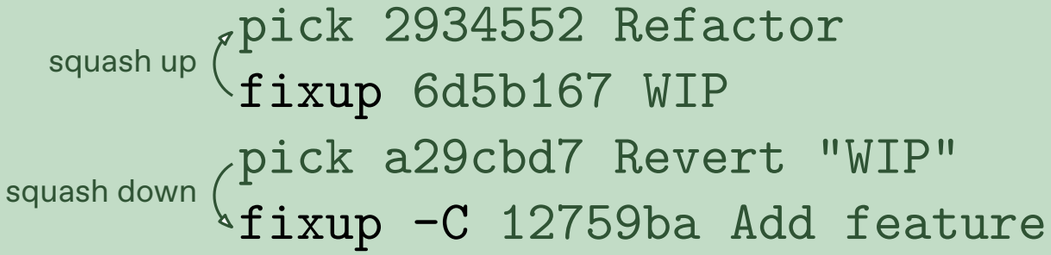
SCALGO

Let's create space for water

New tool in the toolbox: The Hammer

Example: Splitting refactors from feature-additions

- Commit an arbitrary change and immediately revert it



- Can be repeated to move more changes from first to second commit

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

Example: Splitting refactors from feature-additions

- Commit an arbitrary change and immediately revert it

```
pick da65620 Refactor


pick 6a43fca Add feature
```

- Can be repeated to move more changes from first to second commit

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

Example: Splitting reformatting from manual edits

```
break
pick c906cf7 Reformat and add feature
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO

Let's create space for water

New tool in the toolbox: The Hammer

Example: Splitting reformatting from manual edits

```
pick 5a66972 Reformat
pick ef3778a Revert "Reformat"
pick c906cf7 Reformat and add feature
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

Example: Splitting reformatting from manual edits

```
pick 5a66972 Reformat
pick ef3778a Revert "Reformat"
squash c906cf7 Reformat and add feature
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO

Let's create space for water

New tool in the toolbox: The Hammer

Example: Splitting reformatting from manual edits

```
pick 5a66972 Reformat

pick 8158507 Add feature
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO

Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit

- Suppose we want to fix a bug in the first commit

```
edit 01c1fec Initial work
pick ef3778a First separate feature
pick bb06683 Second separate feature
pick 8158507 Third separate feature

$ edit src/foo.c
$ git commit -a --amend && git rebase --continue
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

New tool in the toolbox: The Hammer

```
CONFLICT (content): Merge conflict in src/foo.c
error: could not apply bb06683... Second separate feature
hint: Resolve all conflicts manually, mark them as resolved with
hint: "git add/rm <conflicted_files>", then run "git rebase --continue".
hint: You can instead skip this commit: run "git rebase --skip".
hint: To abort and get back to the state before "git rebase", run "git rebase --abort".
hint: Disable this message with "git config set advice.mergeConflict false"
Could not apply bb06683... # Second separate feature
```

$\Rightarrow$ give up, abort, try to amend the commit in a different way...

There's got to be a better way!

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO

Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit

- Suppose we want to fix a bug in the first commit

```
edit 01c1fec Initial work
pick ef3778a First separate feature
pick bb06683 Second separate feature
pick 8158507 Third separate feature
```

```
$ edit src/foo.c  # fix the bug
$ git commit -am bugfix && git revert @ && git rebase --continue
$ git rebase -i  # start another rebase
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit

- Suppose we want to fix a bug in the first commit

```
pick 01c1fec Initial work
pick 720edcd bugfix
pick 7458c98 Revert "bugfix"
pick ef3778a First separate feature
pick bb06683 Second separate feature
pick 8158507 Third separate feature
```

Hammer time

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit

- Suppose we want to fix a bug in the first commit

```
pick 01c1fec Initial work
pick 720edcd bugfix
pick ef3778a First separate feature
pick 7458c98 Revert "bugfix"
pick bb06683 Second separate feature
pick 8158507 Third separate feature
```

move down

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit

- Suppose we want to fix a bug in the first commit

```
pick 01c1fec Initial work
pick 720edcd bugfix
pick ef3778a First separate feature
pick bb06683 Second separate feature
pick 7458c98 Revert "bugfix"
pick 8158507 Third separate feature
```

move down

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit

- Suppose we want to fix a bug in the first commit

```
pick 01c1fec Initial work
pick 720edcd bugfix
pick e
pick 1
pick 7458c98 Revert "bugfix"
pick 8158507 Third separate feature
```

move down

```
Could not apply bb06683... # Second separate feature
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit

- Suppose we want to fix a bug in the first commit

```
pick 01c1fec Initial work
pick 720edcd bugfix
pick e
pick 1
pick 7
pick 81585
```

move down

```
Could not apply bb06683... # Second separate feature
```

```
$ git rebase --abort
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit

- Suppose we want to fix a bug in the first commit

```
pick 01c1fec Initial work
pick 720edcd bugfix
pick ef3778a First separate feature
pick 7458c98 Revert "bugfix"
pick bb06683 Second separate feature
break
pick 8158507 Third separate feature
edit src/foo.c && git commit && git revert && git rebase --continue
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit

- Suppose we want to fix a bug in the first commit

```
pick 01c1fec Initial work
pick 720edcd bugfix
pick ef3778a First separate feature
pick 7458c98 Revert "bugfix"
pick bb06683 Second separate feature
pick d65b008 bugfix
pick 406e859 Revert "bugfix"
pick 8158507 Third separate feature
```

Hammer time

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit

- Suppose we want to fix a bug in the first commit

```
pick 01c1fec Initial work
pick 720edcd bugfix
pick ef3778a First separate feature
pick 7458c98 Revert "bugfix"
pick bb06683 Second separate feature
pick d65b008 bugfix
pick 406e859 Revert "bugfix"
pick 8158507 Third separate feature
```

Same semantic bugfix implemented twice, but on top of different codebases

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

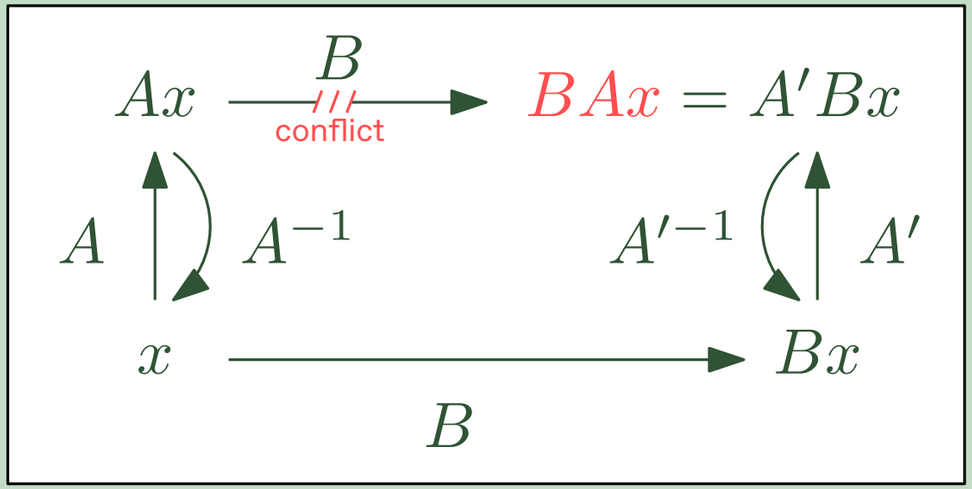
New tool in the toolbox: The Hammer

Example: Amending a previous commit

- Suppose we want to fix a bug in the first commit

x	`pick 01c1fec Initial work`
A	`pick 720edcd bugfix`
	`pick ef3778a First separate feature`
A^{-1}	`pick 7458c98 Revert "bugfix"`
B	`pick bb06683 Second separate feature`
A'	`pick d65b008 bugfix`
A'^{-1}	`pick 406e859 Revert "bugfix"`
	`pick 8158507 Third separate feature`

Same semantic bugfix implemented twice, but on top of different codebases



The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit

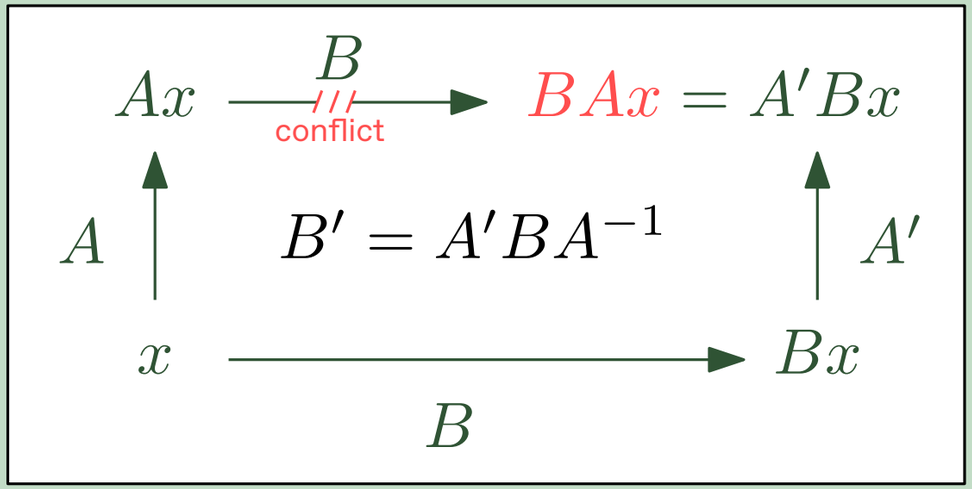
- Suppose we want to fix a bug in the first commit

```
pick 01c1fec Initial work
pick 720edcd bugfix
pick ef3778a First separate feature
pick 7458c98 Revert "bugfix"
fixup -C bb06683 Second separate feature
fixup d65b008 bugfix
pick 406e859 Revert "bugfix"
pick 8158507 Third separate feature
```

squash down

squash up

Same semantic bugfix implemented twice, but on top of different codebases



The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO

Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit

- Suppose we want to fix a bug in the first commit

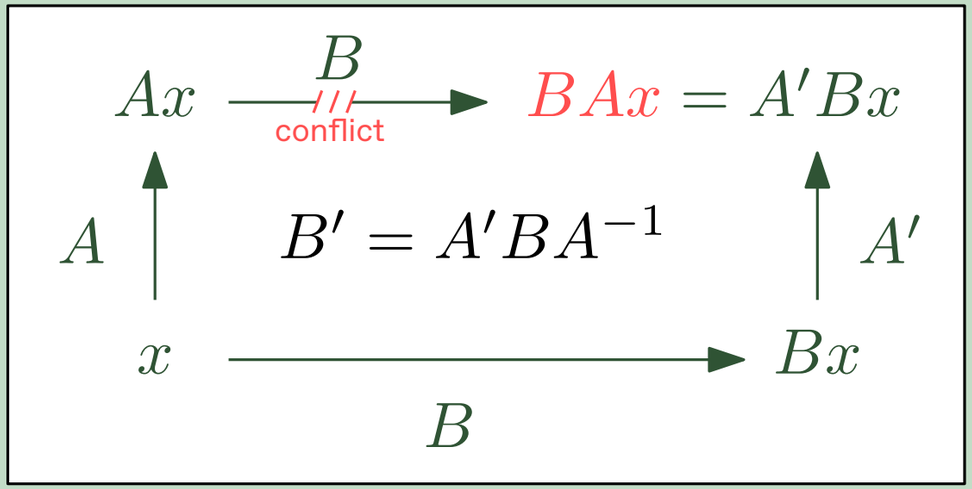
  B, B' are the same semantic change. Proof:

- A, A' same semantic change: $\varphi(A) = \varphi(A')$
- Thus: $\varphi(B') = \varphi(A'BA^{-1}) = \varphi(B)$

  $\Rightarrow B, B'$ same semantic change

! All of this assumes that $\varphi(A)$ and $\varphi(B)$ commute

Same semantic bugfix implemented twice, but on top of different codebases



The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit

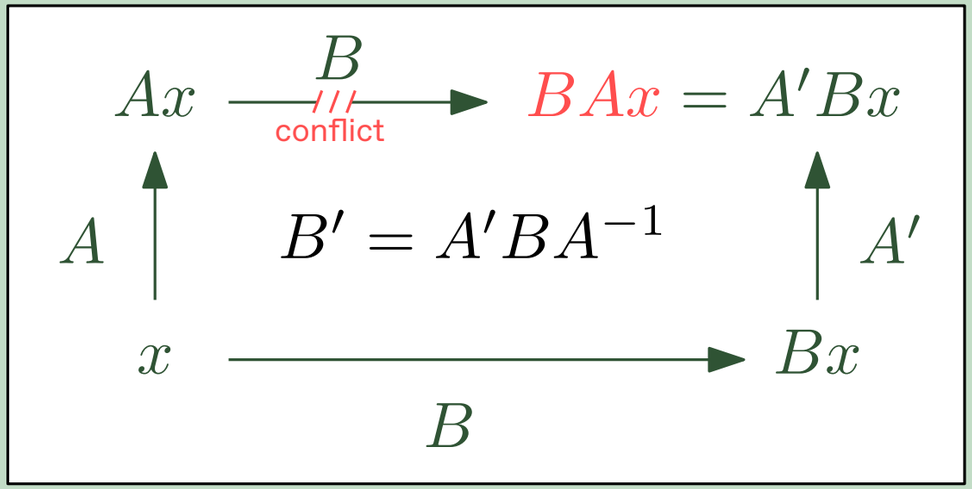
- Suppose we want to fix a bug in the first commit

```
pick 01c1fec Initial work
pick 720edcd bugfix
pick ef3778a First separate feature
pick 7458c98 Revert "bugfix"
fixup -C bb06683 Second separate feature
fixup d65b008 bugfix
pick 406e859 Revert "bugfix"
pick 8158507 Third separate feature
```

squash down

squash up

Same semantic bugfix implemented twice, but on top of different codebases



The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO

Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit

- Suppose we want to fix a bug in the first commit

```
pick 01c1fec Initial work
pick 720edcd bugfix
pick ef3778a First separate feature

pick bb06683 Second separate feature

pick 8158507 Third separate feature
pick 406e859 Revert "bugfix"
```

move down

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit

- Suppose we want to fix a bug in the first commit

```
pick 01c1fec Initial work
pick 720edcd bugfix
pick ef3778a First separate feature

pick bb06683 Second separate feature

pick 8158507 Third separate feature
pick 406e859 Revert "bugfix"
```

and then drop it (always safe at the end)

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit

- Suppose we want to fix a bug in the first commit

squash up

```
pick 01c1fec Initial work
fixup 720edcd bugfix
pick ef3778a First separate feature
pick bb06683 Second separate feature
pick 8158507 Third separate feature
```

and then we're done!

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO

Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit — alternate way

- Suppose we want to fix a bug in the first commit

```
pick 01c1fec Initial work
pick ef3778a First separate feature
pick bb06683 Second separate feature
pick 8158507 Third separate feature
pick a209fc7 bugfix
```

move up

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit — alternate way

- Suppose we want to fix a bug in the first commit

```
pick 01c1fec Initial work
pick ef3778a First separate feature
pick bb06683 Second separate feature
pick a209fc7 bugfix
pick 8158507 Third separate feature
```

move up

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit — alternate way

- Suppose we want to fix a bug in the first commit

```
pick 01c1fec Initial work
pick ef3778a First separate feature
pick b
pick a
pick 8158307 third separate feature
```

move up

```
Could not apply a209fc7... # bugfix
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit — alternate way

- Suppose we want to fix a bug in the first commit

```
pick 01c1fec Initial work
pick ef3778a First separate feature
pick
pick a
pick 81583
```

move up

```
Could not apply a209fc7... # bugfix
```

```
$ git rebase --abort
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

New tool in the toolbox: The Hammer

Example: Amending a previous commit — alternate way

- Suppose we want to fix a bug in the first commit

```
pick 01c1fec Initial work
pick ef3778a First separate feature
pick bb06683 Second separate feature
pick a209fc7 bugfix
pick 8158507 Third separate feature
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

New tool in the toolbox: The Hammer

Example: Amending a previous commit — alternate way

- Suppose we want to fix a bug in the first commit

x `pick 01c1fec Initial work`
`pick ef3778a First separate feature`
A' `pick 1219524 bugfix`
A'^{-1} `pick 0fbd0d2 Revert "bugfix"`
B `pick bb06683 Second separate feature`
A `pick a209fc7 bugfix`
`pick 8158507 Third separate feature`

Hammer time

Same semantic bugfix implemented twice, but on top of different codebases

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

New tool in the toolbox: The Hammer

Example: Amending a previous commit — alternate way

- Suppose we want to fix a bug in the first commit

```
pick 01c1fec Initial work
pick ef3778a First separate feature
pick 1219524 bugfix
pick 0fbd0d2 Revert "bugfix"
fixup -C bb06683 Second separate feature
fixup a209fc7 bugfix
pick 8158507 Third separate feature
```

squash down

squash up

Same semantic bugfix implemented twice, but on top of different codebases

Squash ABA'^{-1}

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO

Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit — alternate way

- Suppose we want to fix a bug in the first commit

move up (using The Hammer)

```
pick 01c1fec Initial work
pick ef3778a First separate feature
pick 1219524 bugfix
pick 9cafda4 Second separate feature
pick 8158507 Third separate feature
```

Despite the syntactic conflict, since there's no semantic conflict, we could move the `bugfix` commit up

... without ever looking at merge conflict markers!

(Instead, we implemented the same bugfix twice)

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO

Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit — alternate way

- Suppose we want to fix a bug in the first commit

```
pick 01c1fec Initial work
pick 1219524 bugfix
pick ef3778a First separate feature
pick 9cafda4 Second separate feature
pick 8158507 Third separate feature
```

move up

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

Example: Amending a previous commit — alternate way

- Suppose we want to fix a bug in the first commit

squash up

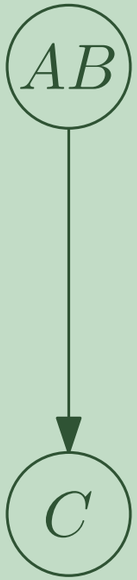
```
pick 01c1fec Initial work
fixup 1219524 bugfix
pick ef3778a First separate feature
pick 9cafda4 Second separate feature
pick 8158507 Third separate feature
```

and then we're done!

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

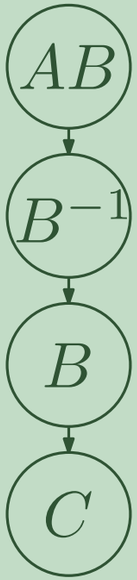



```
pick b48659f # Do A and B

break   # remove “B” and immediately revert

pick 028d6e2 # Do C
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let’s create space for water

New tool in the toolbox: The Hammer




```
pick b48659f # Do A and B
pick da65620 # Undo B
pick a0101e0 # Revert "Undo B"
pick 028d6e2 # Do C
```

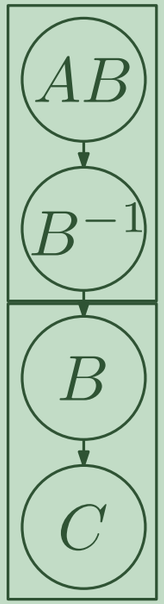
Done by hand

Done using `git revert`

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

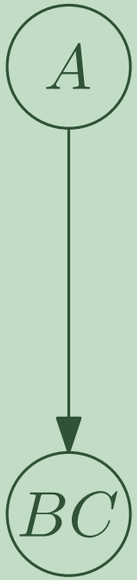



```
pick b48659f # Do A and B

squash da65620 # Undo B

pick a0101e0 # Revert "Undo B"

squash 028d6e2 # Do C
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

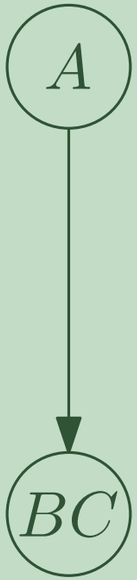



```
pick 6a43fca # Do A

pick 768196d # Do B and C
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

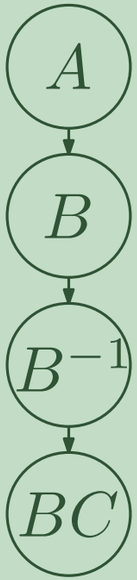



```
pick 6a43fca # Do A

break   # add “B” and immediately revert

pick 768196d # Do B and C
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let’s create space for water

New tool in the toolbox: The Hammer

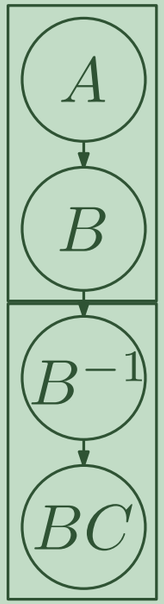



```
pick 6a43fca # Do A
pick 972de9f # Do B
pick 627a9c6 # Revert "Do B"
pick 768196d # Do B and C
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

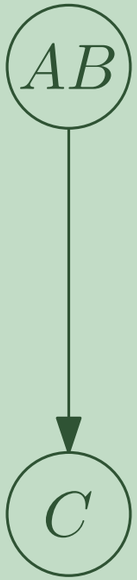



```
pick 6a43fca # Do A
squash 972de9f # Do B
pick 627a9c6 # Revert "Do B"
squash 768196d # Do B and C
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer



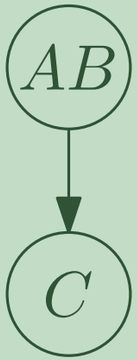

```
pick b48659f # Do A and B

pick 028d6e2 # Do C
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

```
break   # add “A” and immediately revert
```

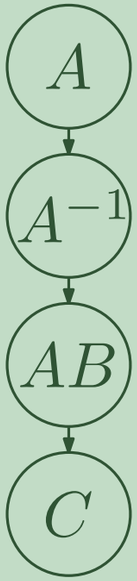



```
pick b48659f # Do A and B

pick 028d6e2 # Do C
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let’s create space for water

New tool in the toolbox: The Hammer

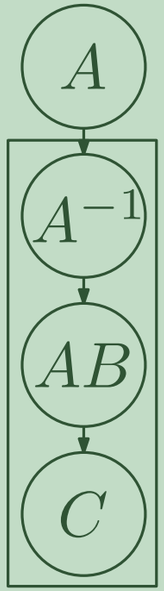



```
pick 6a43fca # Do A
pick da65620 # Revert "Undo A"
pick b48659f # Do A and B
pick 028d6e2 # Do C
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

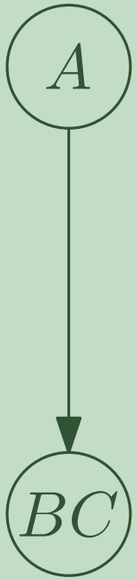



```
pick 6a43fca # Do A

pick da65620 # Revert "Undo A"

squash b48659f # Do A and B

squash 028d6e2 # Do C
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

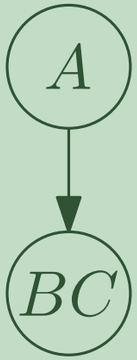



```
pick 6a43fca # Do A

pick 768196d # Do B and C
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer




```
pick 6a43fca # Do A

pick 768196d # Do B and C

break   # undo “C” and immediately revert
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let’s create space for water

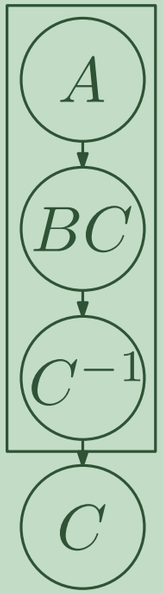
New tool in the toolbox: The Hammer


```
pick 6a43fca # Do A
pick 768196d # Do B and C
pick 028d6e2 # Undo C
pick 627a9c6 # Revert "Undo C"
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

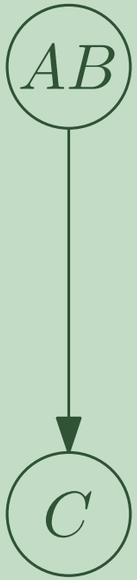



```
pick 6a43fca # Do A

squash 768196d # Do B and C

squash 028d6e2 # Undo C

reword 627a9c6 # Revert "Undo C"
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer




```
pick b48659f # Do A and B

pick 028d6e2 # Do C
```

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

New tool in the toolbox: The Hammer

- In summary, there are four ways to split AB with The Hammer:
  — Do A by hand: $A, (A^{-1}, AB) \Rightarrow A, B$
  — Do B by hand: $B, (B^{-1}, AB) \Rightarrow B, A$
  — Undo B by hand: $(AB, B^{-1}), B \Rightarrow A, B$
  — Undo A by hand: $(AB, A^{-1}), A \Rightarrow B, A$
- Do you need A first or B first in the resulting branch?
- Is it easier/faster to redo the one change or undo the other change?
- Do you need to split with `git reset` and `git add -p` first?
- … Is this really faster than just dealing with the merge conflict markers?

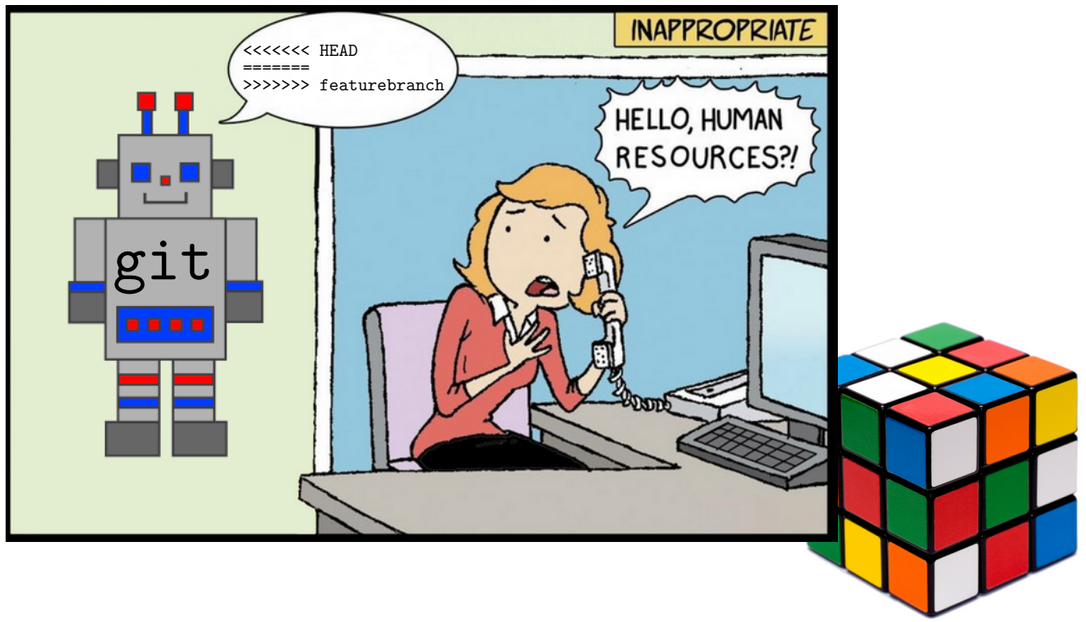
(shouldn't take more than a couple minutes to do)

4 combinations

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

SCALGO
Let's create space for water

Aside: Why even bother?



- Is The Hammer really faster than just dealing with the conflict markers?
- Well, to be honest … not always …
- … but it can be more relaxing
- Merge conflicts: A puzzle, a riddle, a trap, git making your life hard

Aside: Why even bother?

Claim:
Being good at writing code and doing code reviews does not make you good at solving merge conflicts, but it does make you good at using The Hammer.

- Is The Hammer really faster than just dealing with the conflict markers?
- Well, to be honest … not always …
- … but it can be more relaxing
- Merge conflicts: A puzzle, a riddle, a trap, git making your life hard
- The Hammer: Reading code, writing code

— "Redoing A by hand": Read through the AB commit's diff, and apply the A parts to the codebase
…with a calm mind and your favorite beverage at hand

— You can take a break and `git checkout` another branch at any time, with no `git reset --hard` and no messy repo state

SCALGO
Let's create space for water

Bonus: utility aliases and commands

- `git h` for commit+revert, `git hc` for commit+revert+continue

Put this in your .gitconfig:

```
[alias]
 h = "!/bin/bash -c \"git commit -nam \\\"HAMMER \\$*\\\" && git revert --no-edit @\""
 hc = "!/bin/bash -c \"git commit -nam \\\"HAMMER \\$*\\\" && git revert --no-edit @ && git rebase --continue\""
```

- `x s` to split a rebased commit by modified files

Run this to install:

```
mkdir -p ~/.local/lib &&
git clone https://github.com/Mortal/git-rebase-utils ~/.local/lib/git-rebase-utils &&
cat >> ~/.bashrc <<'EOF'
alias git='PATH=~/.local/lib/git-rebase-utils:$PATH \git'
EOF
```

```
pick 01c1fec Initial work
x s ef3778a First separate feature
pick bb06683 Second separate feature
```

SCALGO

Let's create space for water

Bonus: utility aliases and commands

- `git h` for commit+revert, `git hc` for commit+revert+continue

Put this in your .gitconfig:

```
[alias]
 h = "!/bin/bash -c \"git commit -nam \\\"HAMMER \\$*\\\" && git revert --no-edit @\""
 hc = "!/bin/bash -c \"git commit -nam \\\"HAMMER \\$*\\\" && git revert --no-edit @ && git rebase --continue\""
```

- `x s` to split a rebased commit by modified files

Run this to install:

```
mkdir -p ~/.local/lib &&
git clone https://github.com/Mortal/git-rebase-utils ~/.local/lib/git-rebase-utils &&
cat >> ~/.bashrc <<'EOF'
alias git='PATH=~/.local/lib/git-rebase-utils:$PATH \git'
EOF
```

```
pick 01c1fec Initial work
pick 798415b src/foo.c First separate feature
pick 447921e src/foo.h First separate feature
pick 8ac0094 README First separate feature
pick bb06683 Second separate feature
```

SCALGO

Let's create space for water

Bonus: utility aliases and commands

- `git h` for commit+revert, `git hc` for commit+revert+continue

Put this in your .gitconfig:

```
[alias]
 h = "!/bin/bash -c \"git commit -nam \\\"HAMMER \\$*\\\" && git revert --no-edit @\""
 hc = "!/bin/bash -c \"git commit -nam \\\"HAMMER \\$*\\\" && git revert --no-edit @ && git rebase --continue\""
```

- `x s` to split a rebased commit by modified files

Run this to install:

```
mkdir -p ~/.local/lib &&
git clone https://github.com/Mortal/git-rebase-utils ~/.local/lib/git-rebase-utils &&
cat >> ~/.bashrc <<'EOF'
alias git='PATH=~/.local/lib/git-rebase-utils:$PATH \git'
EOF
```

- Vim plugin to quickly rearrange `pick` lines in `git rebase -i`

Run this to install:

```
mkdir -p ~/.local/lib &&
git clone https://github.com/Mortal/hammertime ~/.local/lib/hammertime &&
echo >> ~/.vimrc <<'EOF'
source ~/.local/lib/hammertime/vimplugin.vim
EOF
```

Use case summary

Example: Splitting refactors from feature-additions

```
pick eb99118 Refactor and add feature
fixup 58f52a0 Remove feature
reword 0615bbc Revert "Remove feature"
```

squash up

Hammer time

Example: Iteratively move changes from one commit to another

```
pick 2934552 Refactor
fixup 6d5b167 WIP
pick a29cbd7 Revert "WIP"
fixup -C 12759ba Add feature
```

squash up

squash down

Hammer time

Example: Moving a fixup past a conflicting change

```
x   pick 01c1fec Initial work
A'  pick 1219524 bugfix
A'-1 pick 0fbd0d2 Revert "bugfix"
B   pick bb06683 Additional work
A   pick a209fc7 bugfix
```

squash down

squash up

Hammer time

Same semantic fixup implemented twice, but on top of different codebases

SCALGO

Let's create space for water

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

No merge conflicts or your money back!*

*Assuming you follow the instructions correctly.
Remember to squash before rebasing.
Only do one thing at a time when rebasing.
Consult a doctor before using The Hammer.

Example: Splitting refactors from feature-additions

```
squash up ↱ pick eb99118 Refactor and add feature
          ↳ fixup 58f52a0 Remove feature
            reword 0615bbc Revert "Remove feature"
```

Hammer time

Example: Iteratively move changes from one commit to another

```
squash up   ↱ pick 2934552 Refactor
            ↳ fixup 6d5b167 WIP
squash down ↱ pick a29cbd7 Revert "WIP"
            ↳ fixup -C 12759ba Add feature
```

Hammer time

Example: Moving a fixup past a conflicting change

```
  x   pick 01c1fec Initial work
 A'   pick 1219524 bugfix
A'⁻¹  pick 0fbd0d2 Revert "bugfix"
 B    pick bb06683 Additional work
 A    pick a209fc7 bugfix
```

squash down: A'^{-1} → B

squash up: B → A

Hammer time

Same semantic fixup implemented twice, but on top of different codebases

SCALGO
Let's create space for water

The Hammer: Insert A and A^{-1} at some point in a commit sequence, then squash A up and/or squash A^{-1} down

No merge conflicts or your money back!*

*Assuming you follow the instructions correctly.
Remember to squash before rebasing.
Only do one thing at a time when rebasing.
Consult a doctor before using The Hammer.

Example: Splitting refactors from feature-additions

```
squash up ↱ pick eb99118 Refactor and add feature
          ↳ fixup 58f52a0 Remove feature
            reword 0615bbc Revert "Remove feature"
```

Hammer time

Example: Iteratively move changes from one commit to anoth

```
squash up   ↱ pick 2934552 Refactor
            ↳ fixup 6d5b167 WIP
squash down ↱ pick a29cbd7 Revert "WIP"
            ↳ fixup -C 12759ba Add feature
```

Hammer time

Thank you!

Any questions?

Mathias Rav
mathias@scalgo.com

Example: Moving a fixup past a conflicting change

```
squash down ↱ $A'^{-1}$ pick 0fbd0d2 Revert "bugfix"
            ↳ $B$ pick bb06683 Additional work
squash up   ↱ $A$ pick a209fc7 bugfix
            ↳ $x$ pick 01c1fec Initial work
              $A'$ pick 1219524 bugfix
```

Hammer time

Same semantic fixup implemented twice, but on top of different codebases

SCALGO
Let's create space for water

Peaceful conflict resolution in version control systems

`m.strova.dk/git`

Mathias Rav, Chief Version Control Engineer
mathias@scalgo.com

SCALGO