TEMPO Workshop on Software Development

Michal Kvasnica

STU

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

“An expert is a person who has made all the mistakes that can be made in a very narrow field.”

Niels Bohr, Nobel price winner (1922)

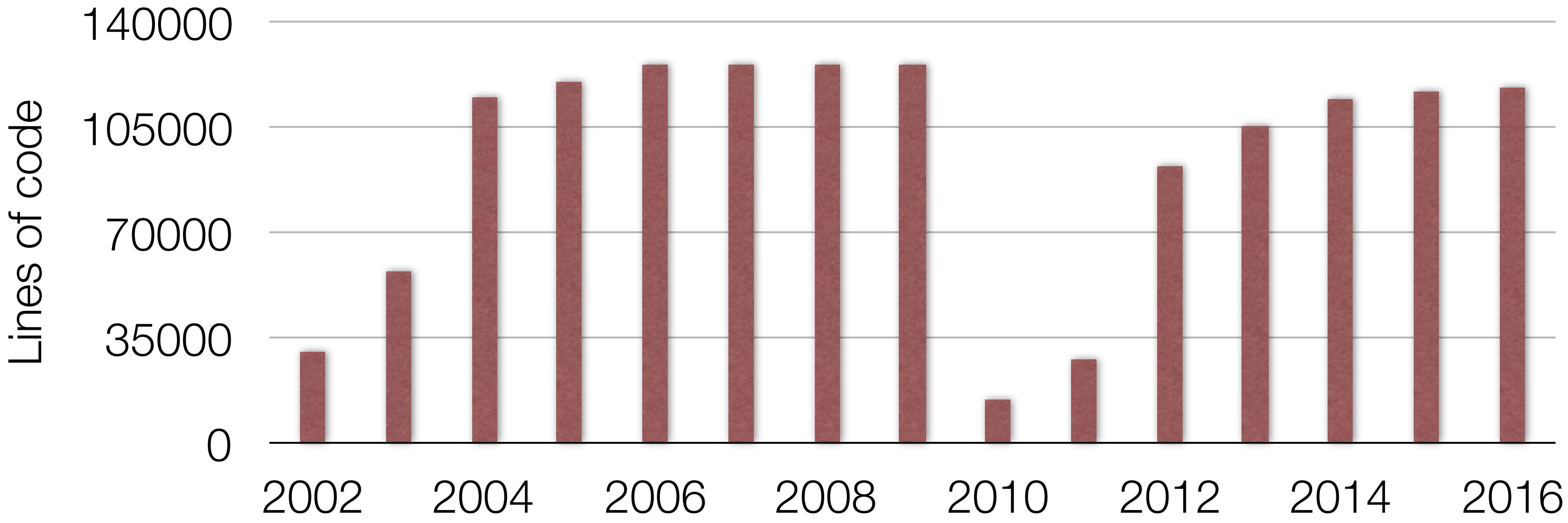
The Multi-Parametric Toolbox (MPT)

Developed since 2002

40,000+ installations world-wide

118,503 lines of code in 2,169 files

5 main developers



Agenda

Today

- version control systems and collaborative development
- Mercurial, Git, Bitbucket, GitHub
- providing support

Tomorrow

- unit testing
- documentation
- dissemination

Set of DOs and DONTs

Agenda

Today

- **version control systems and collaborative development**
- Mercurial, Git, Bitbucket, GitHub
- providing support

Tomorrow

- unit testing
- documentation
- dissemination

DONT #1: Do NOT Use Archives

```
mpt12.zip
mpt1_2_2.zip
mpt_ver04.zip
mpt_ver11R1.zip
mpt_ver141.zip
mpt13r1.zip
mpt_ver02.zip
mpt_ver05.zip
mpt_ver11R2.zip
mpt_ver144.zip
mpt14.zip
mpt_ver03.zip
mpt_ver11.zip
mpt_ver11R3.zip
mptv122.zip
```

Main problems:

- does not protect against accidental deletion
- linear development (cannot release until all new features are complete)
- single-user development
- difficult to find a change that introduced a bug

Archives do not give answers to:

- what has changed between versions?
- who made the changes?
- when and why were the changes made?
- which version is the latest stable release?

DO #1: Use a Version Control System

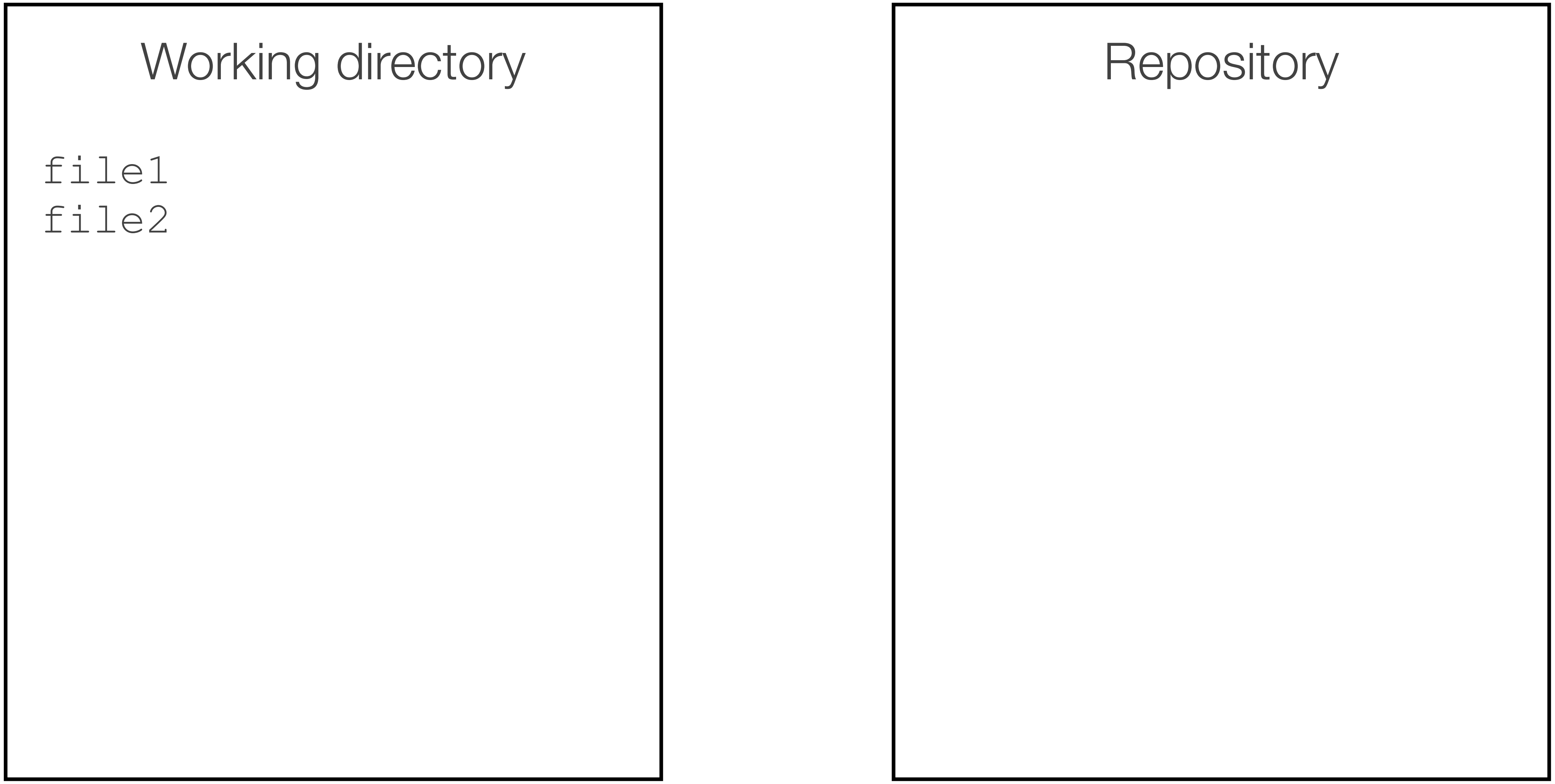
Main objectives:

- record changes over time
- recall a specific version later
- enable collaboration
- allow nonlinear development

DO #1: Use a Version Control System

Main objectives:

- **record changes over time**
- recall a specific version later
- enable collaboration
- allow nonlinear development



Working directory

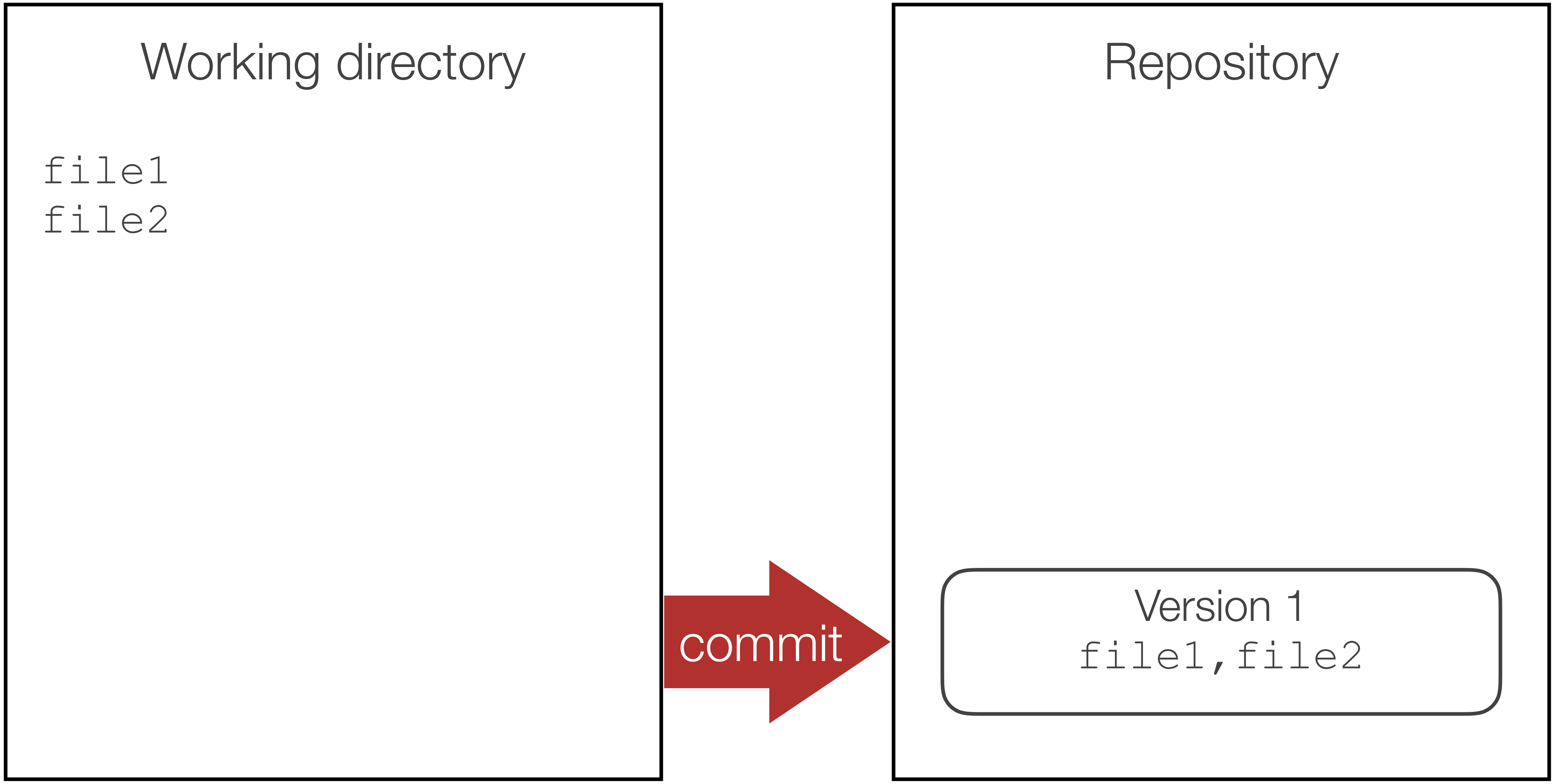
```
file1
file2
```

Repository

DO #1: Use a Version Control System

Main objectives:

- **record changes over time**
- recall a specific version later
- enable collaboration
- allow nonlinear development



Working directory

file1
file2

commit

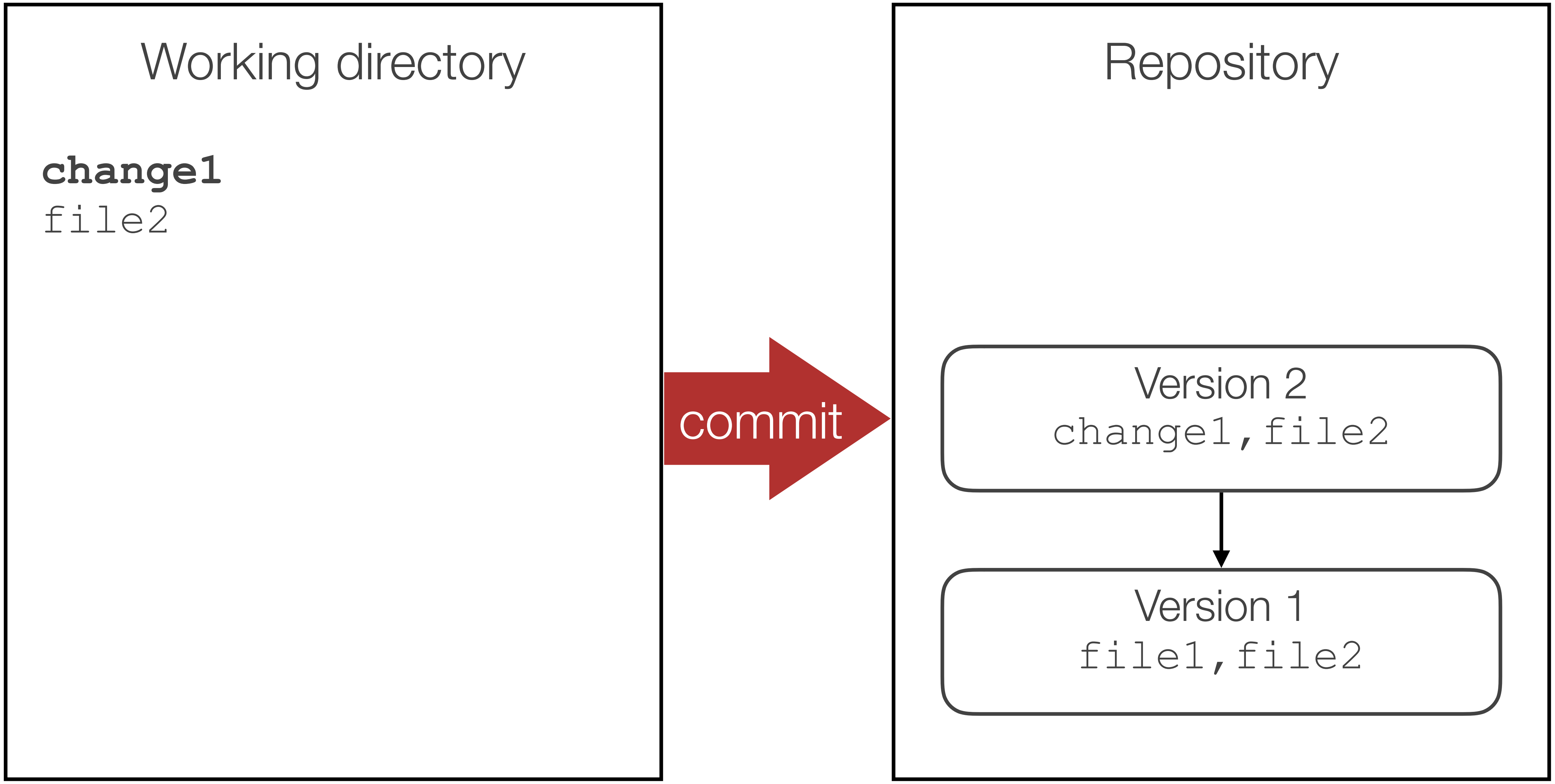
Repository

Version 1
file1,file2

DO #1: Use a Version Control System

Main objectives:

- **record changes over time**
- recall a specific version later
- enable collaboration
- allow nonlinear development



Working directory

change1
file2

commit

Repository

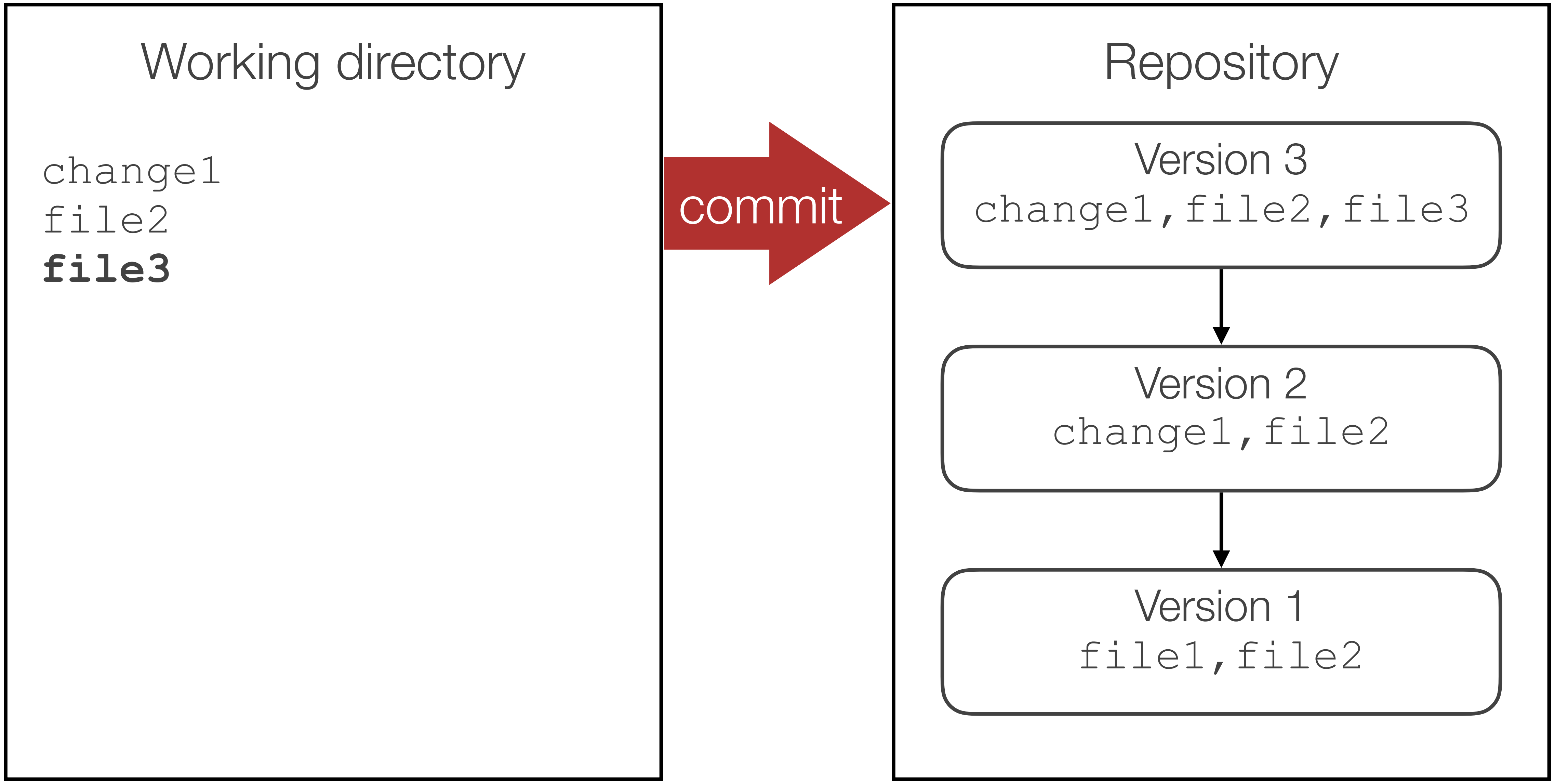
Version 2
change1,file2

Version 1
file1,file2

DO #1: Use a Version Control System

Main objectives:

- **record changes over time**
- recall a specific version later
- enable collaboration
- allow nonlinear development



Working directory

change1
file2
file3

commit

Repository

Version 3
change1,file2,file3

Version 2
change1,file2

Version 1
file1,file2

Sidenote: VCS is for Source Code

Typically stored in VCS:

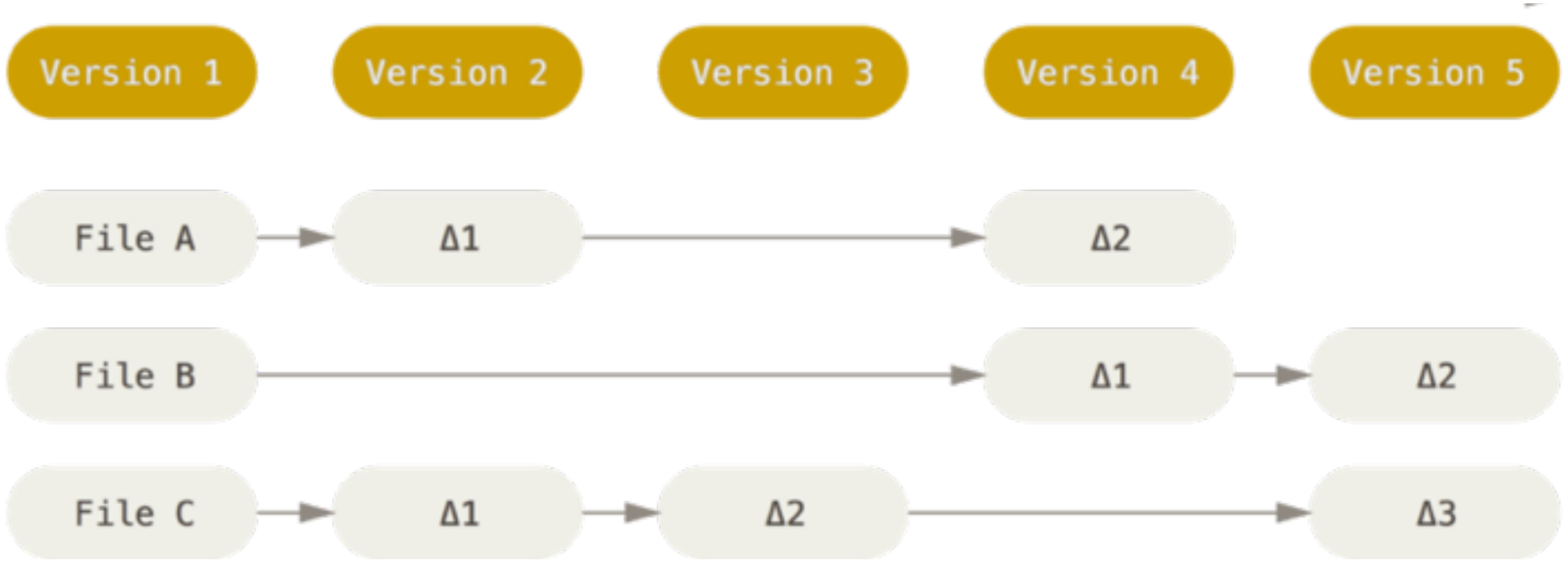
- source files
- documentation (markdown, html, LaTeX)
- binary files that do not change (e.g. logos)

Not to be stored:

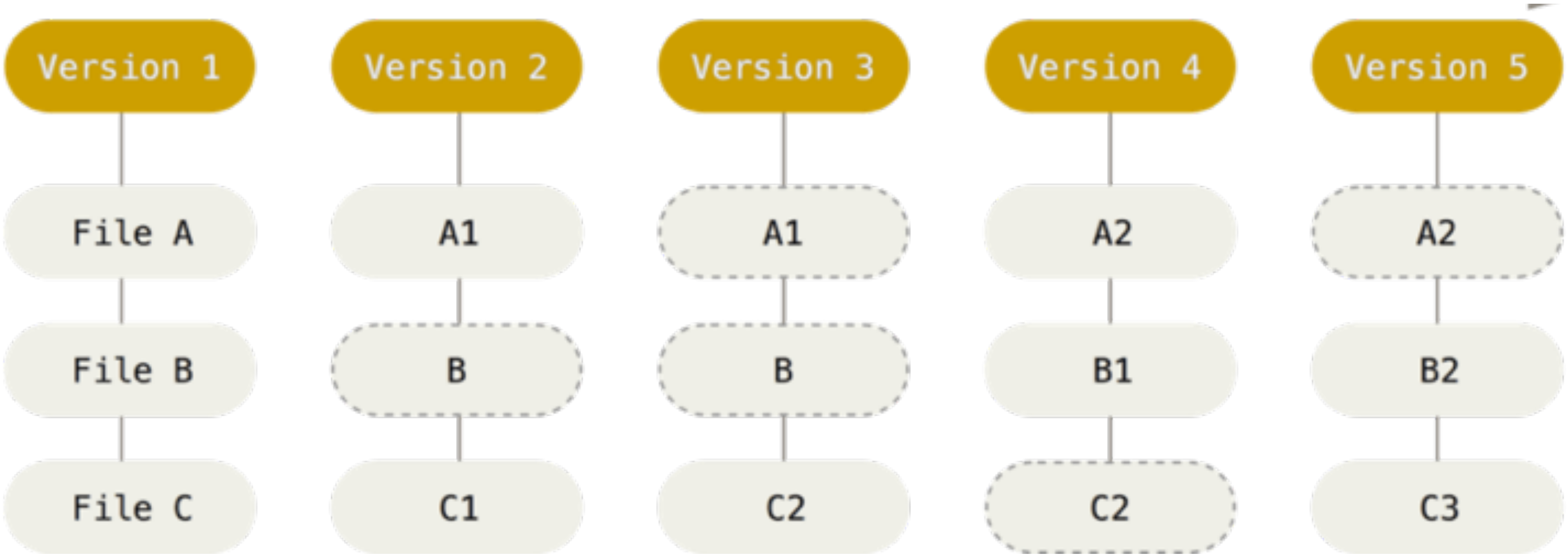
- automatically generated files (PDFs from LaTeX, compiled binaries, etc.)
- Word documents
- files containing passwords! (even when you remove the data from the top version, it stays in the repository forever in previous versions)

Sidenote: Different Storage Models

Deltas:
(Mercurial)



Snapshots:
(Git)

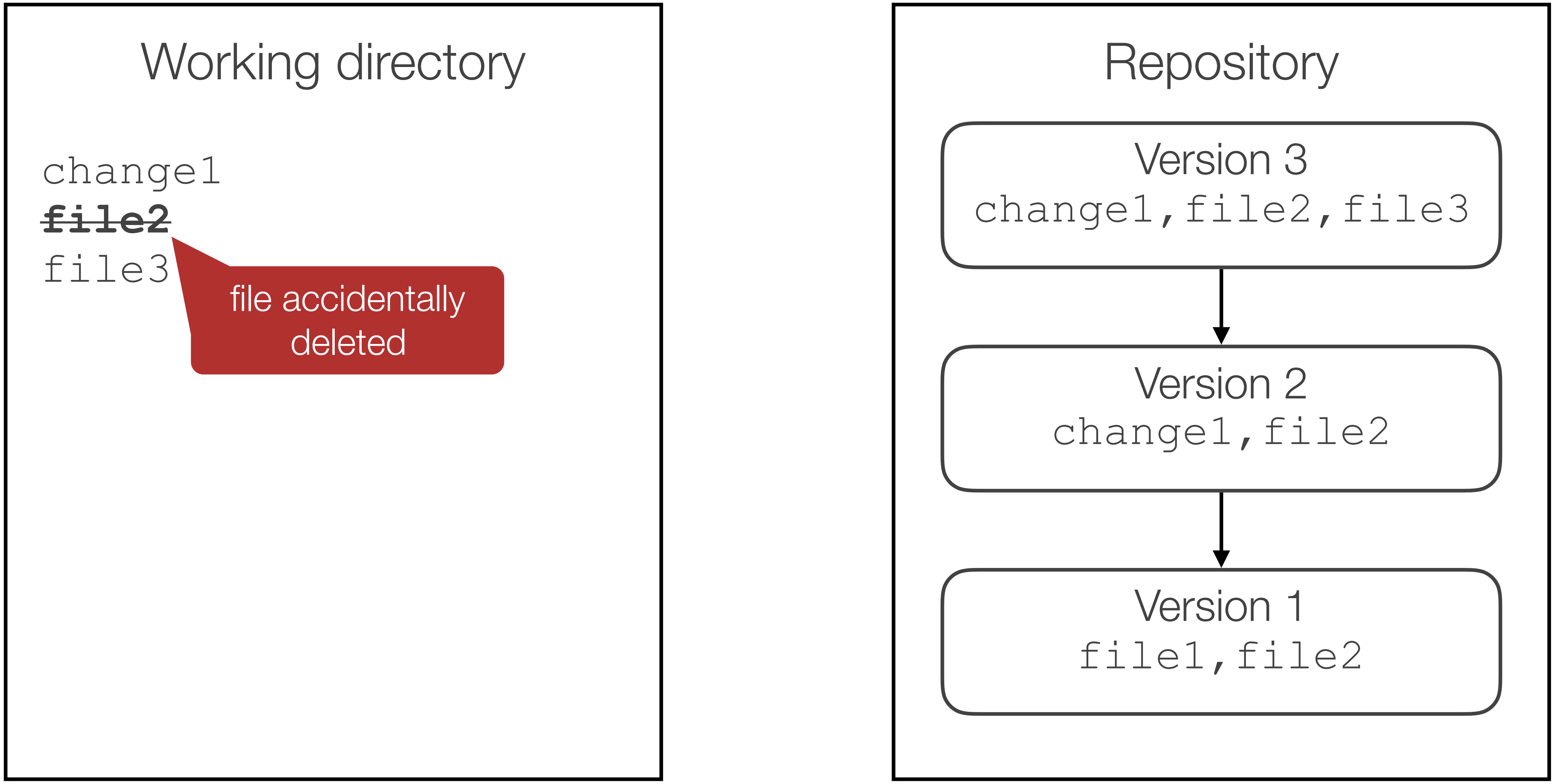


https://git-scm.com/book/en/v2/Getting-Started-Git-Basics

DO #1: Use a Version Control System

Main objectives:

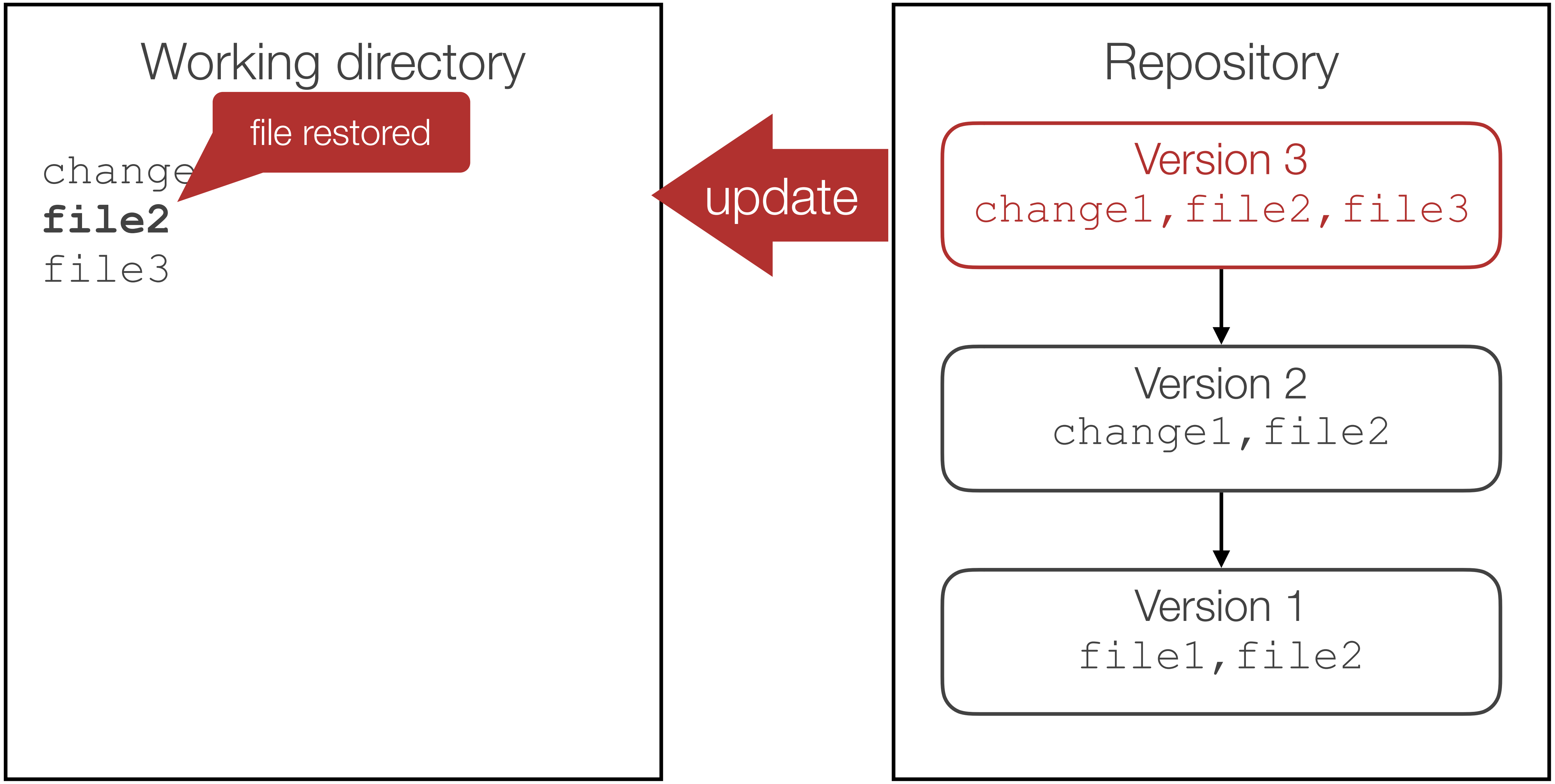
- record changes over time
- **recall a specific version later**
- enable collaboration
- allow nonlinear development



DO #1: Use a Version Control System

Main objectives:

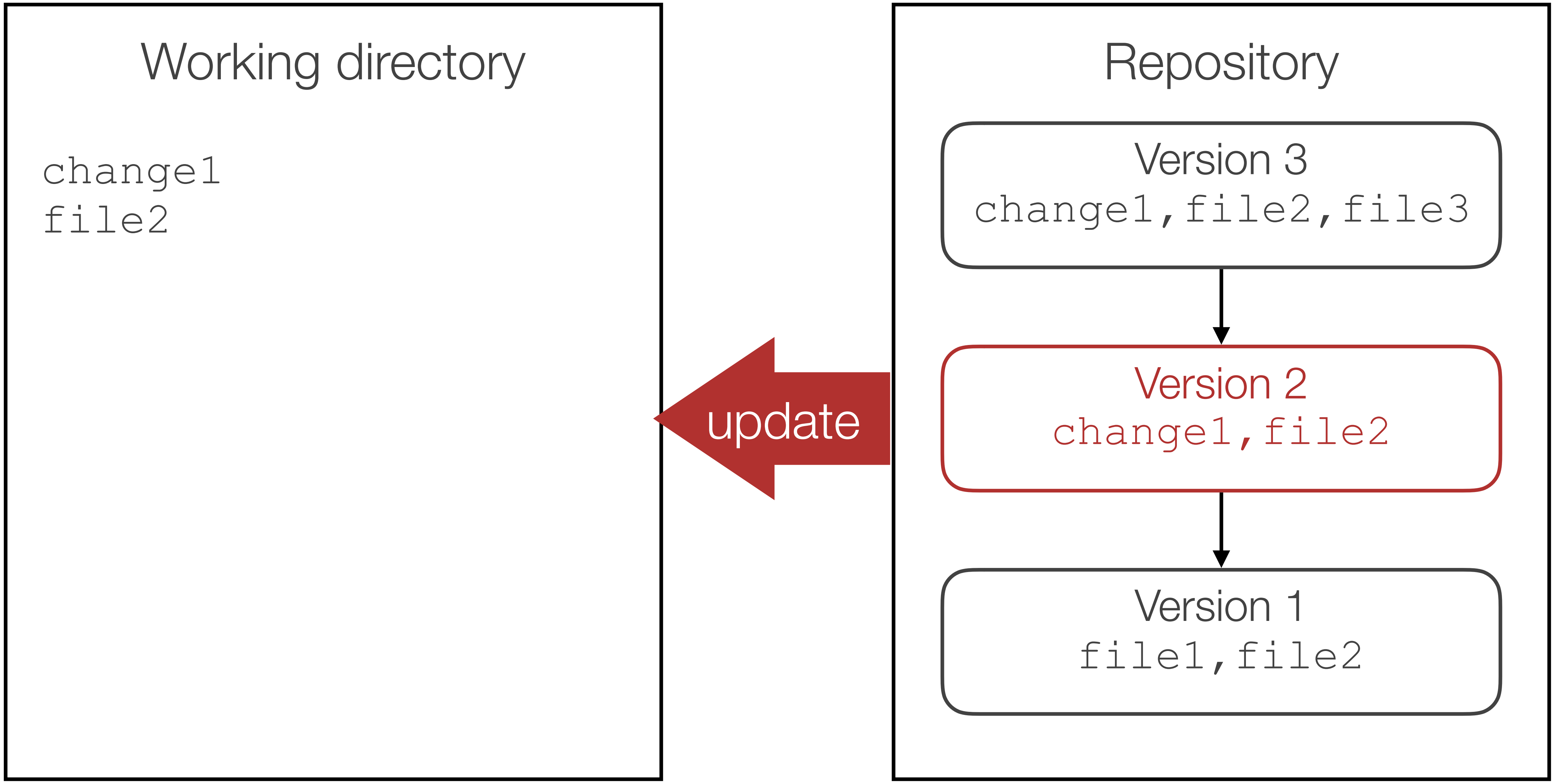
- record changes over time
- **recall a specific version later**
- enable collaboration
- allow nonlinear development



DO #1: Use a Version Control System

Main objectives:

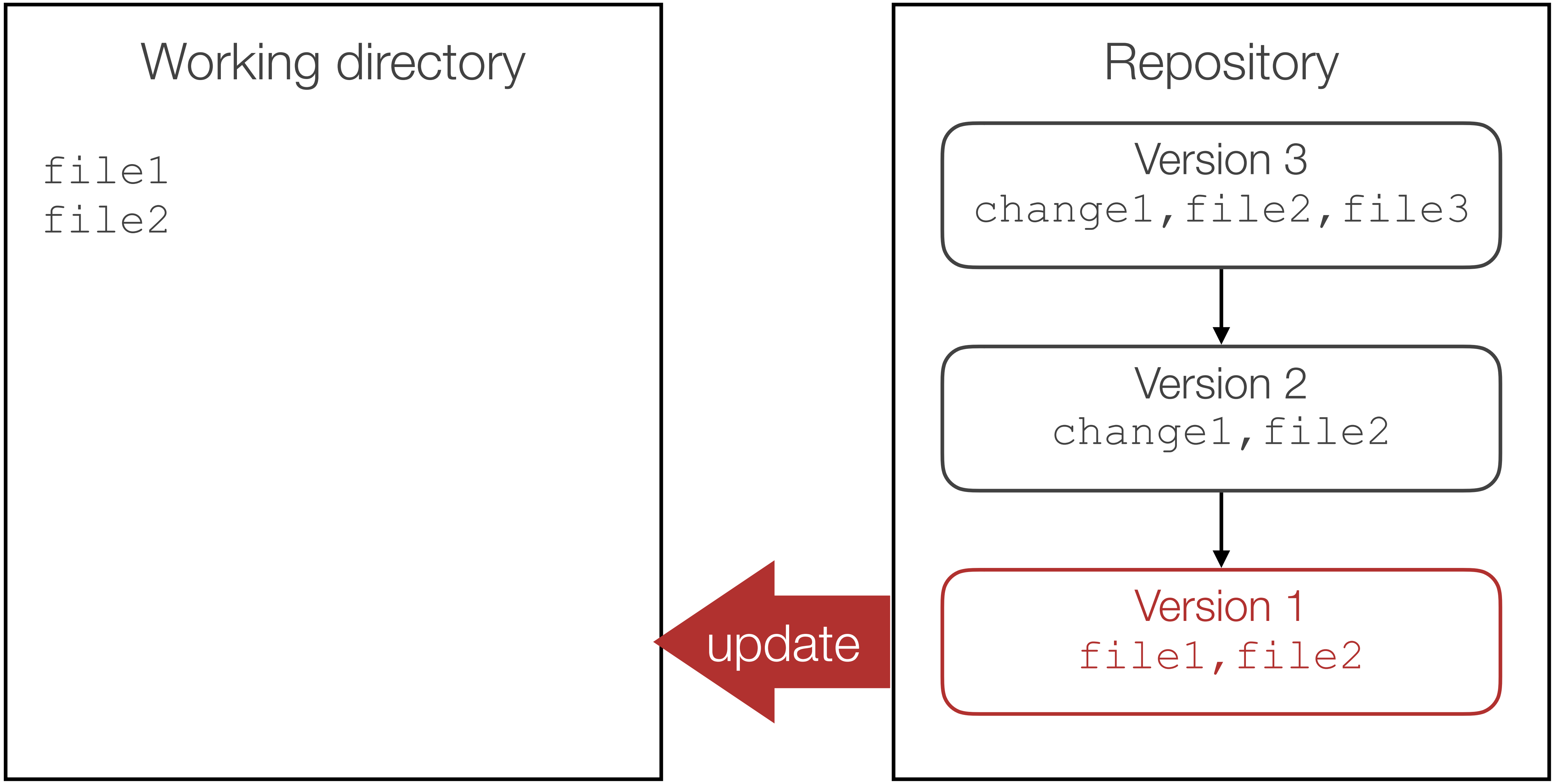
- record changes over time
- **recall a specific version later**
- enable collaboration
- allow nonlinear development



DO #1: Use a Version Control System

Main objectives:

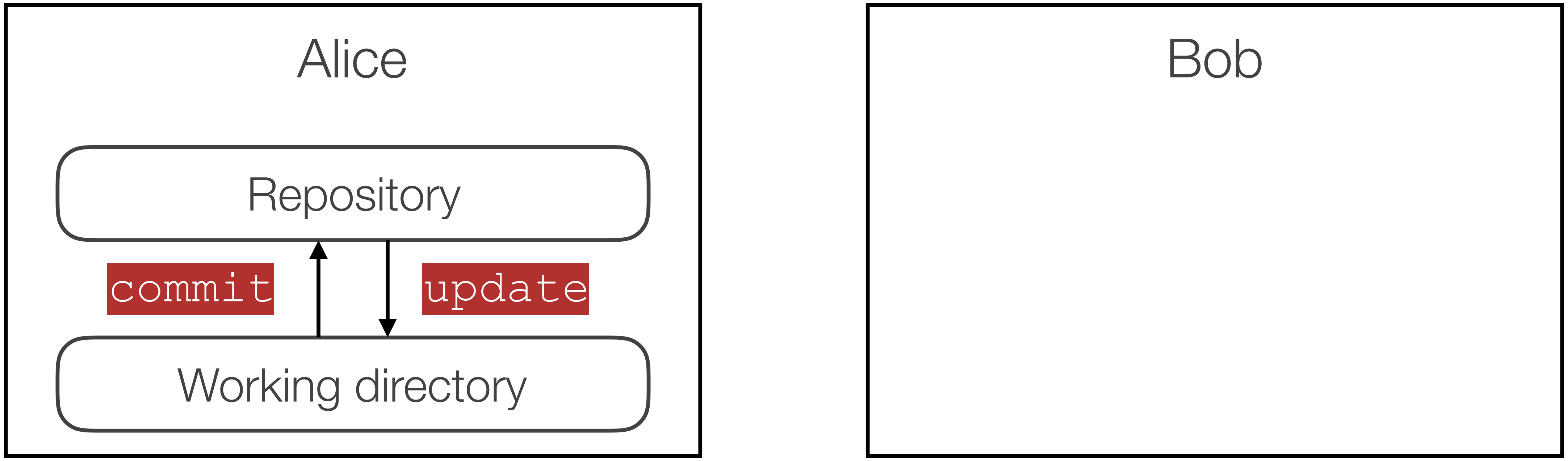
- record changes over time
- **recall a specific version later**
- enable collaboration
- allow nonlinear development



DO #1: Use a Version Control System

Main objectives:

- record changes over time
- recall a specific version later
- **enable collaboration**
- allow nonlinear development



Alice

Repository

`commit`

`update`

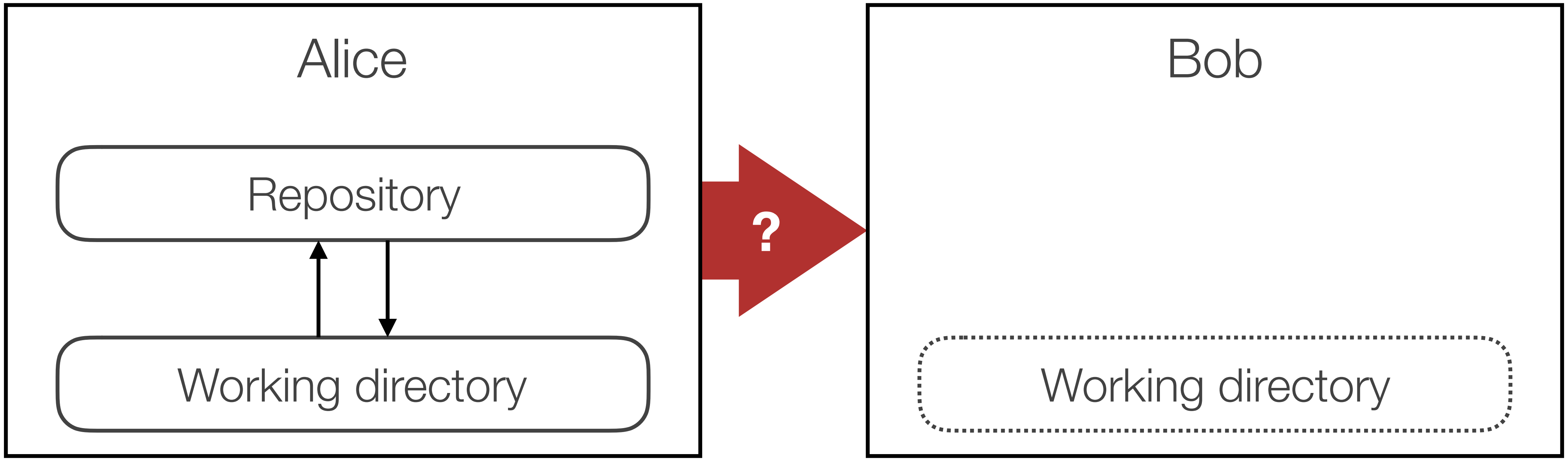
Working directory

Bob

DO #1: Use a Version Control System

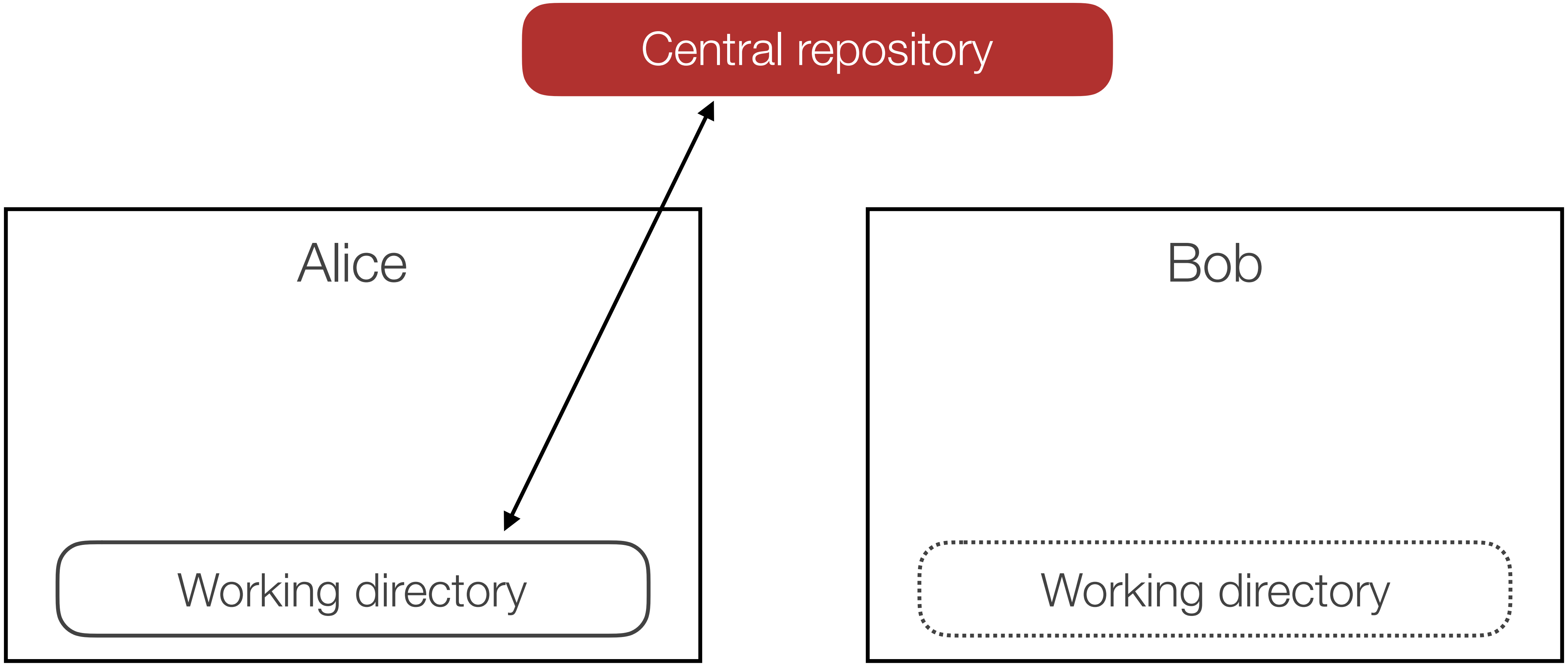
Main objectives:

- record changes over time
- recall a specific version later
- **enable collaboration**
- allow nonlinear development



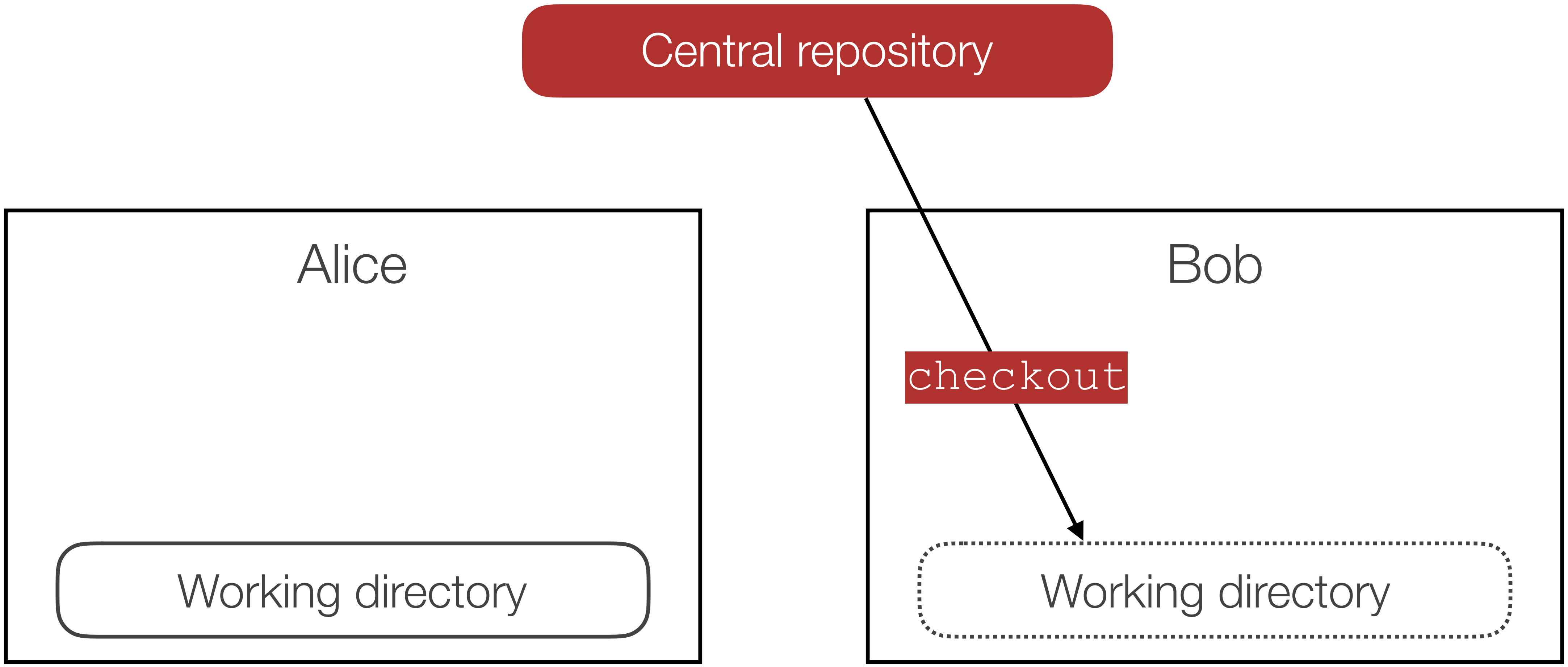
Centralized VCS (CVS, SVN, Perforce)

Main idea: use a centralized repository (public server)



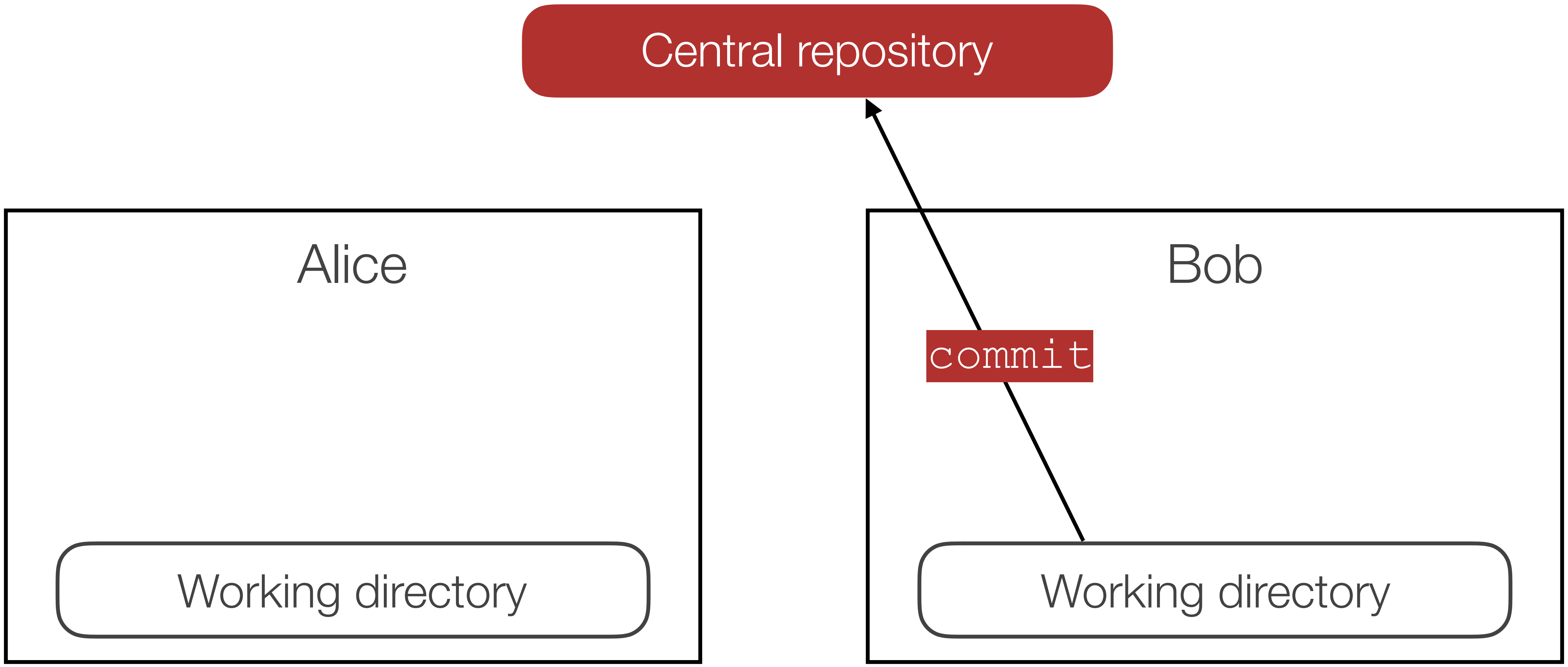
Centralized VCS (CVS, SVN, Perforce)

Main idea: use a centralized repository (public server)



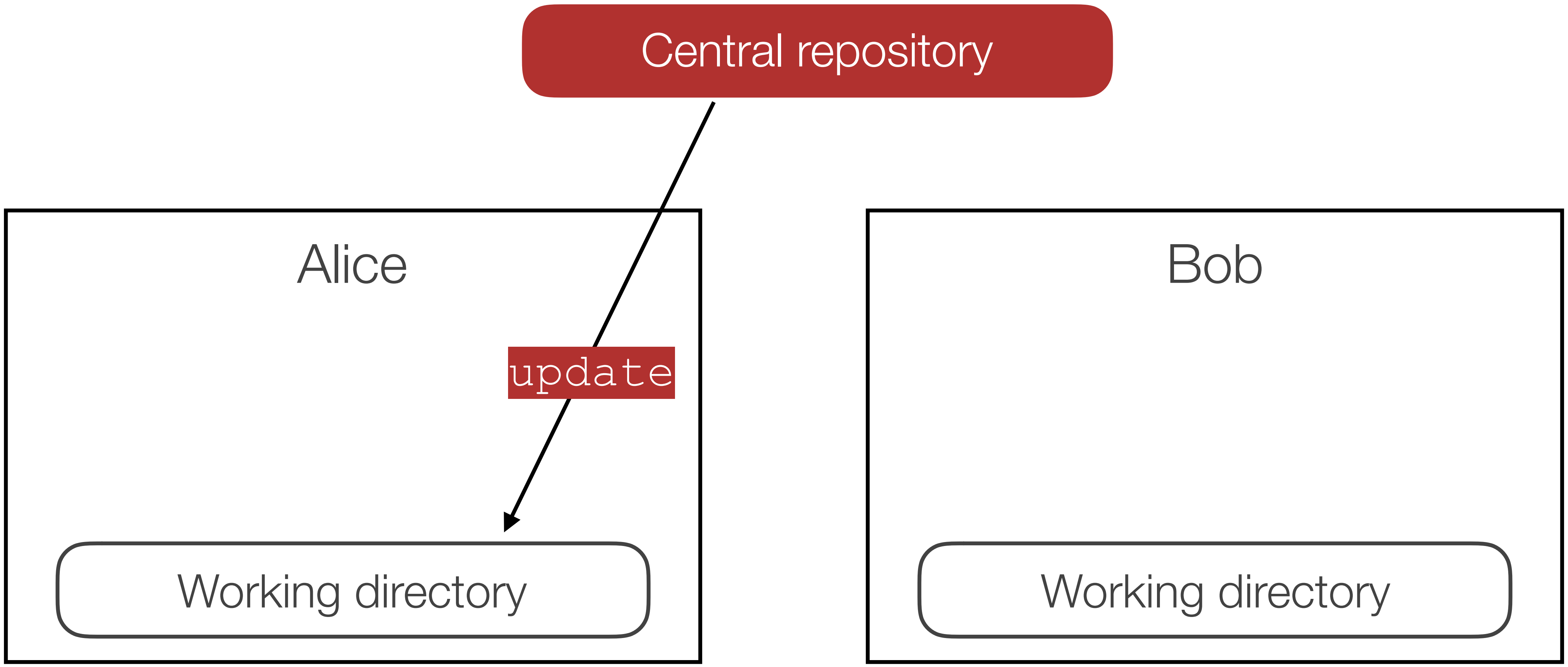
Centralized VCS (CVS, SVN, Perforce)

Main idea: use a centralized repository (public server)



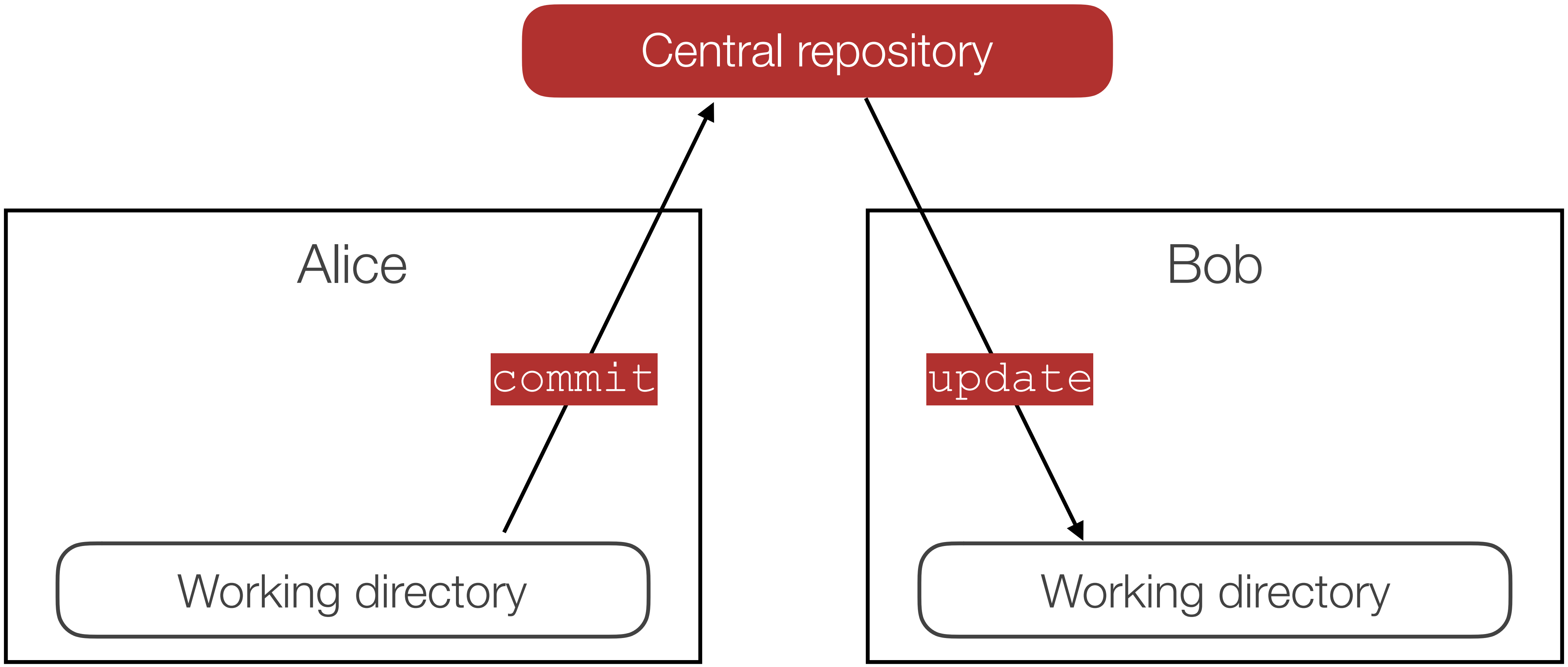
Centralized VCS (CVS, SVN, Perforce)

Main idea: use a centralized repository (public server)



Centralized VCS (CVS, SVN, Perforce)

Main idea: use a centralized repository (public server)

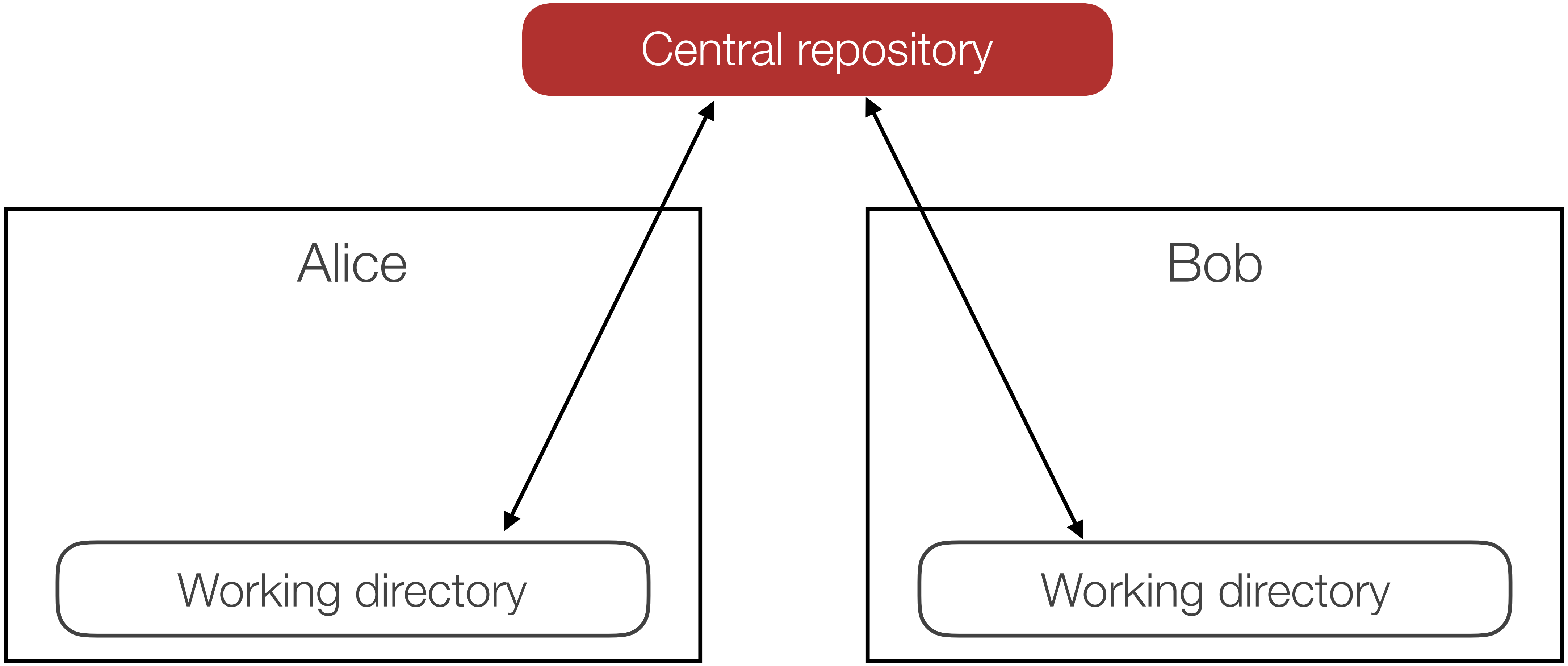


Centralized VCS (CVS, SVN, Perforce)

Main idea: use a centralized repository (public server)

Problems:

- requires permanent internet connection (+single point of failure)
- access rights management
- communication ("Hey, I made a change, you need to update")
- only one person can commit at a time

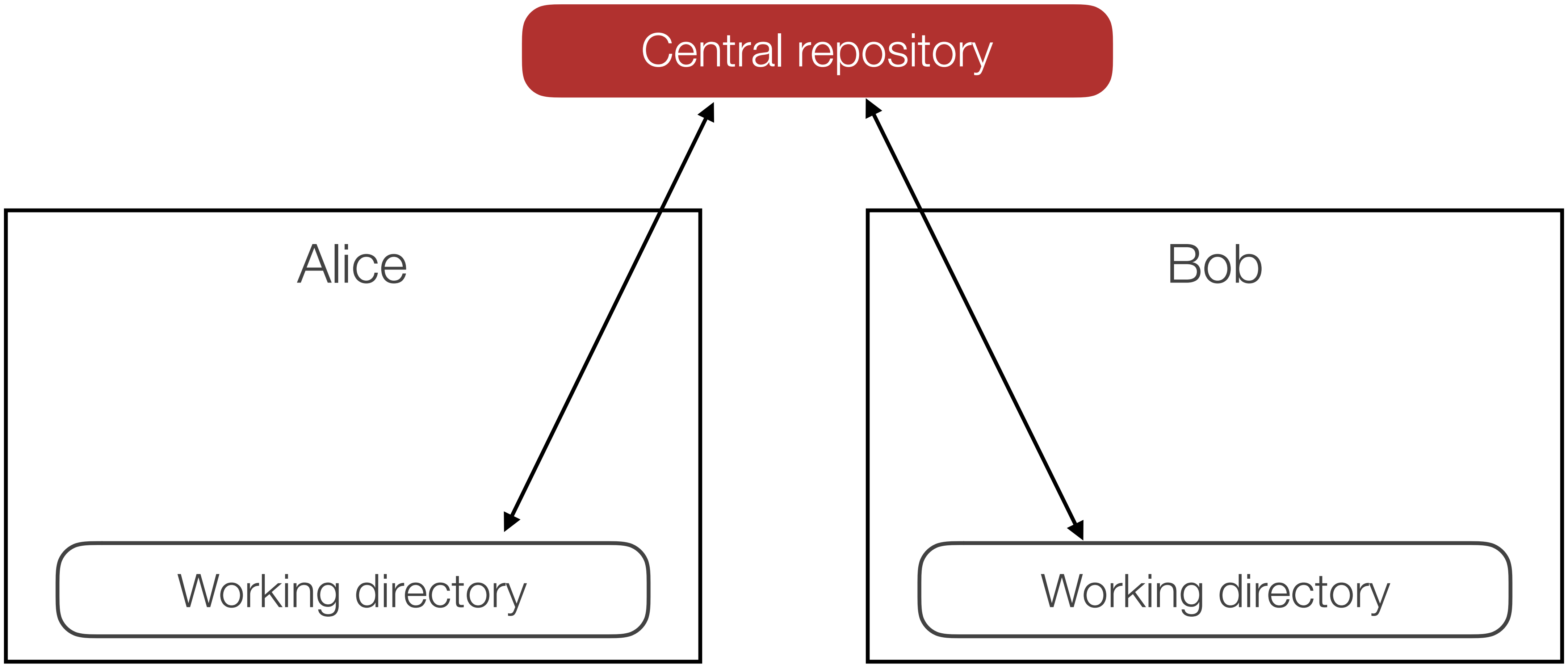


DONT #2: Never use a Centralized VCS

Main idea: use a centralized repository (public server)

Problems:

- requires permanent internet connection (+single point of failure)
- access rights management
- communication ("Hey, I made a change, you need to update")
- only one person can commit at a time

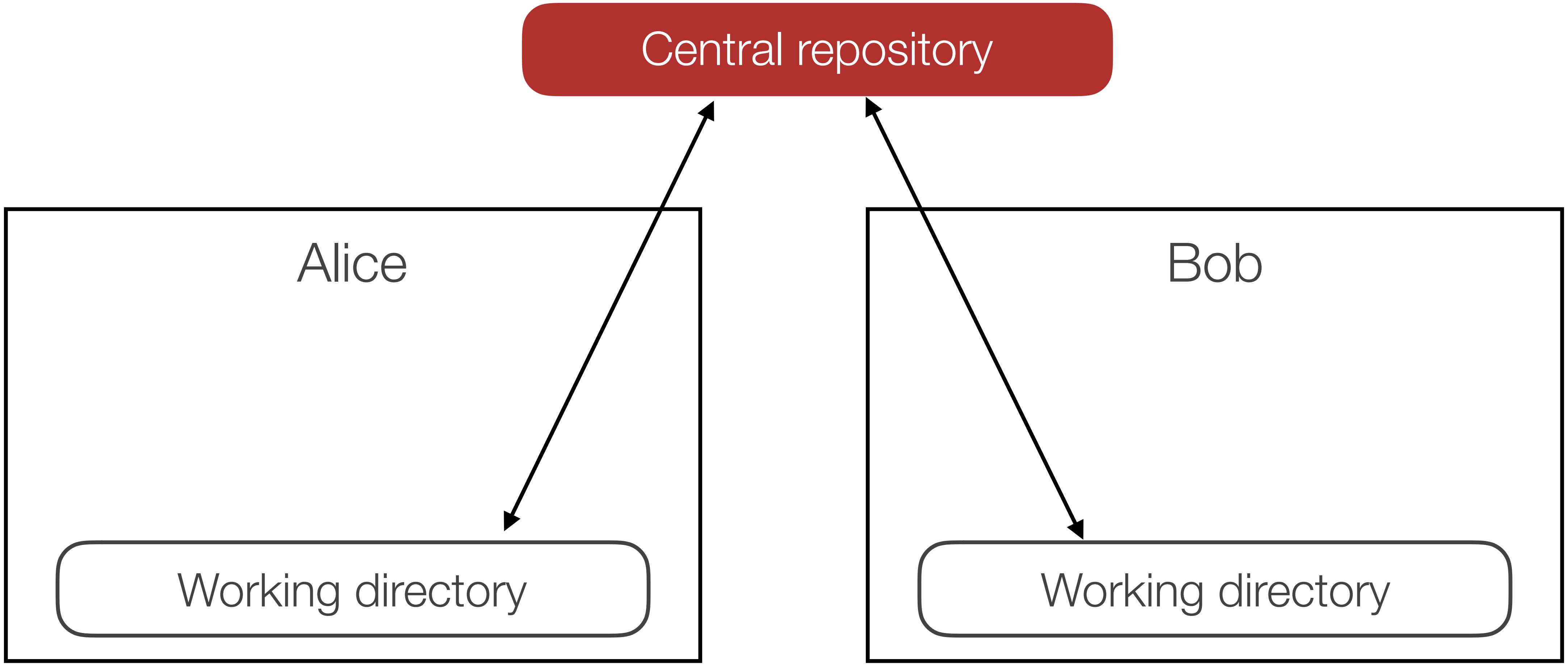


DONT #2: Never use a Centralized VCS

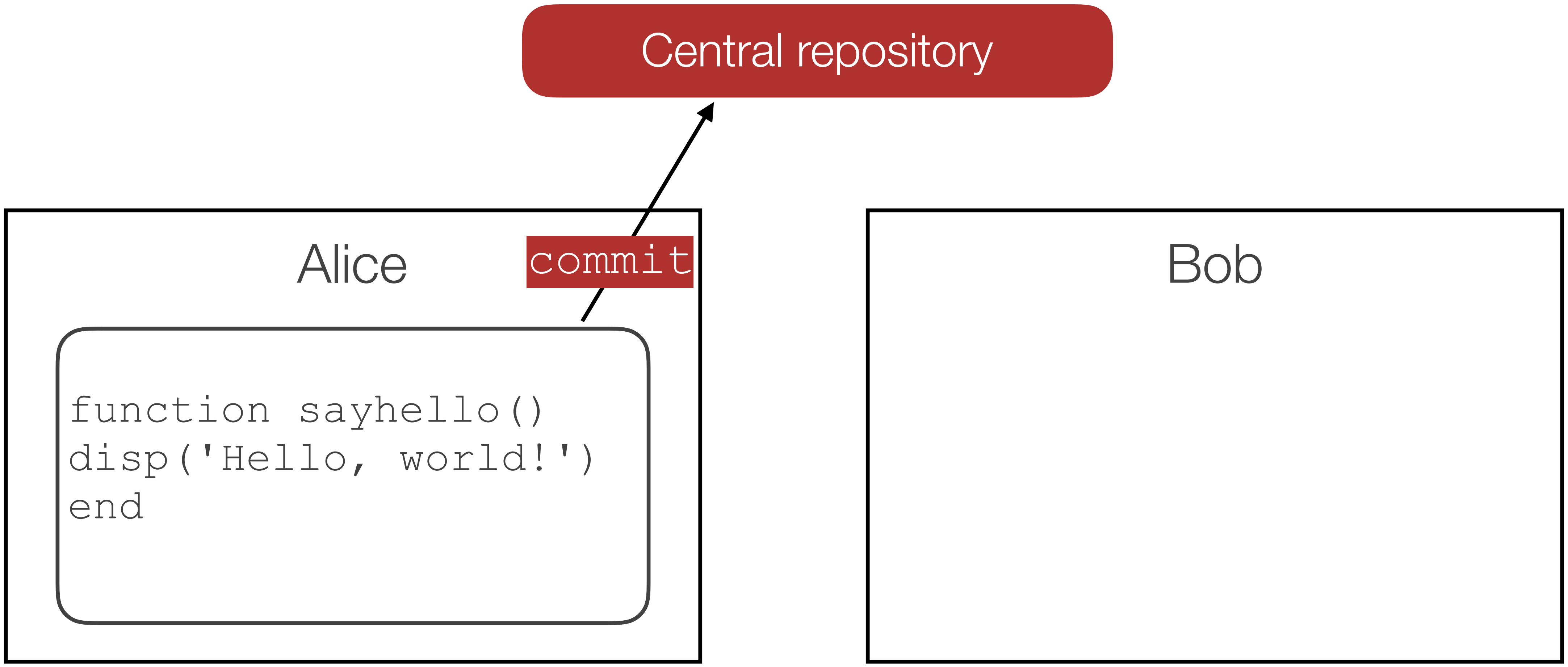
Main idea: use a centralized repository (public server)

Problems:

- requires permanent internet connection (+single point of failure)
- access rights management
- communication ("Hey, I made a change, you need to update")
- **only one person can commit at a time**



Sidenote: Merges



Central repository

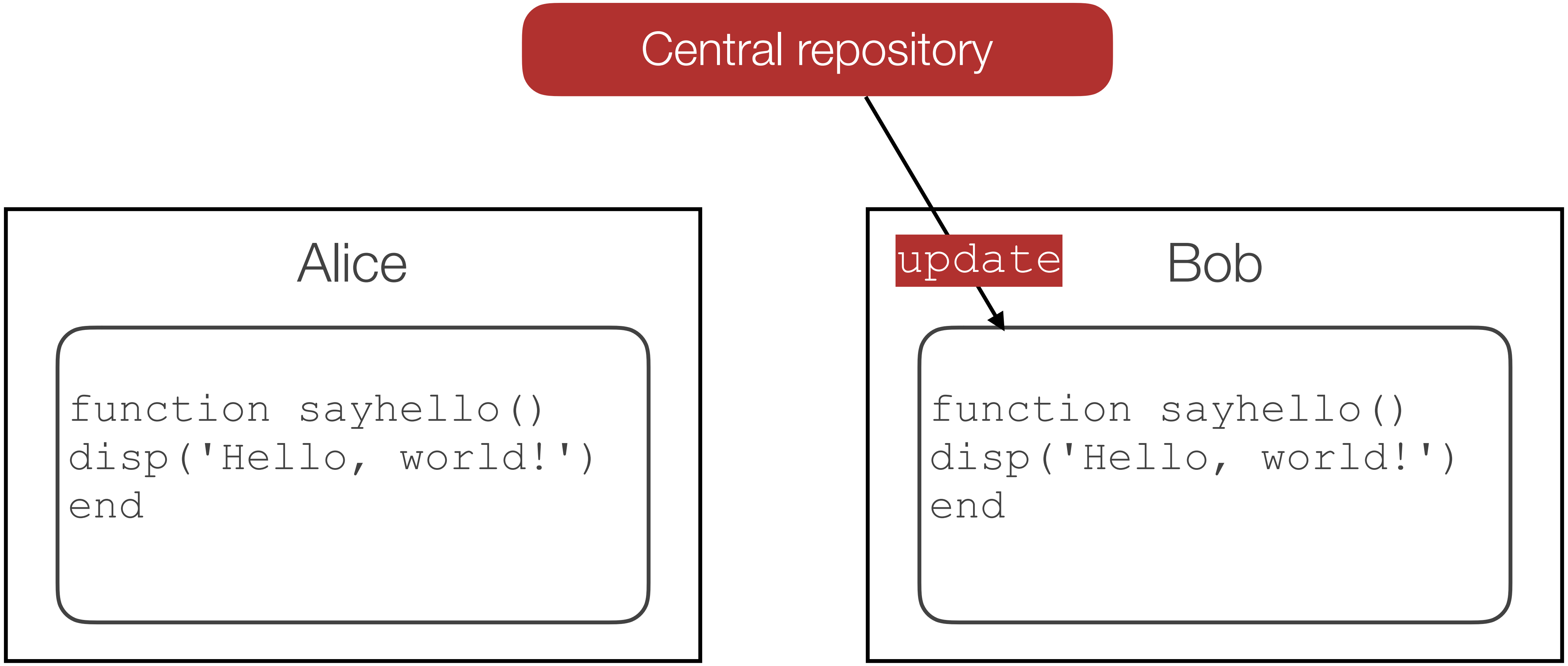
commit

Alice

```
function sayhello()
disp('Hello, world!')
end
```

Bob

Sidenote: Merges



Central repository

`update`

Alice

```
function sayhello()
disp('Hello, world!')
end
```

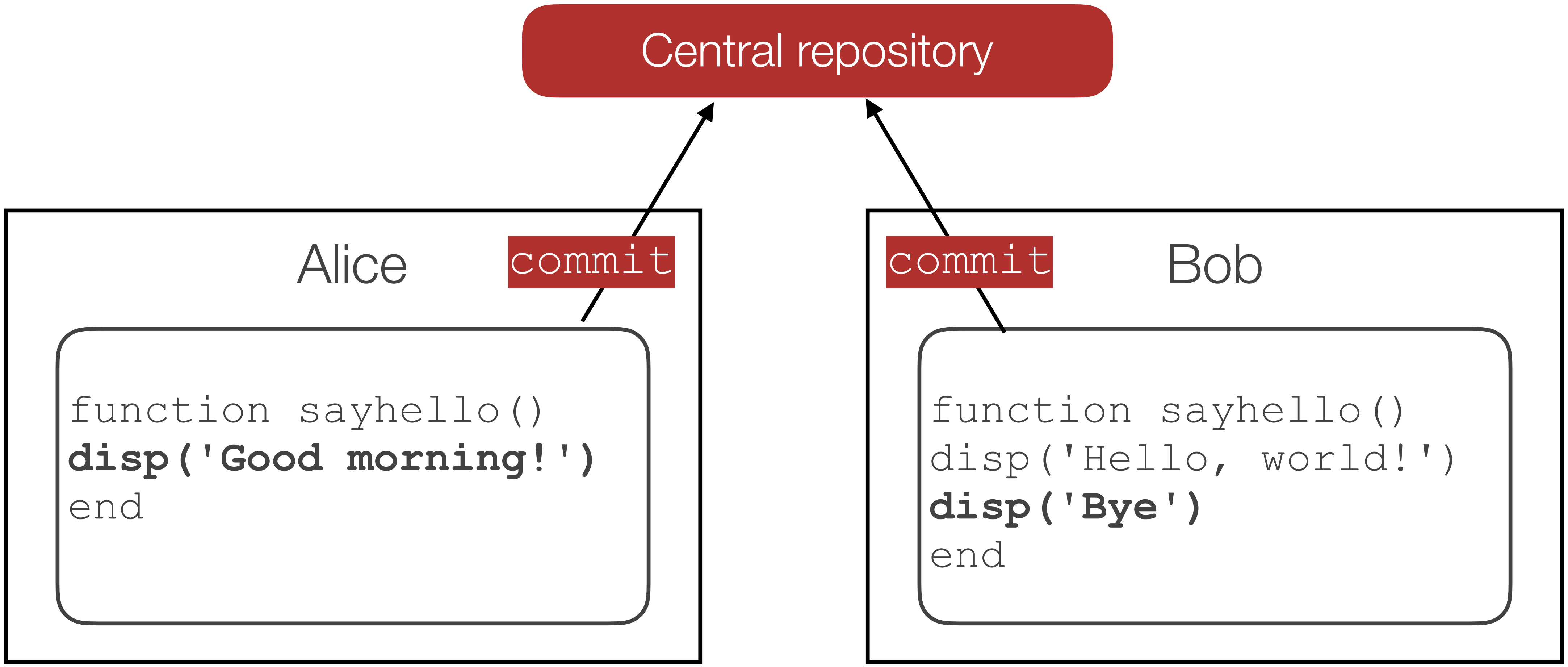
Bob

```
function sayhello()
disp('Hello, world!')
end
```

Sidenote: Merges

Centralized VCS will prevent simultaneous commits

- in fact, "update" locks the repository for a particular user



```
function sayhello()
disp('Good morning!')
end
```

```
function sayhello()
disp('Hello, world!')
disp('Bye')
end
```

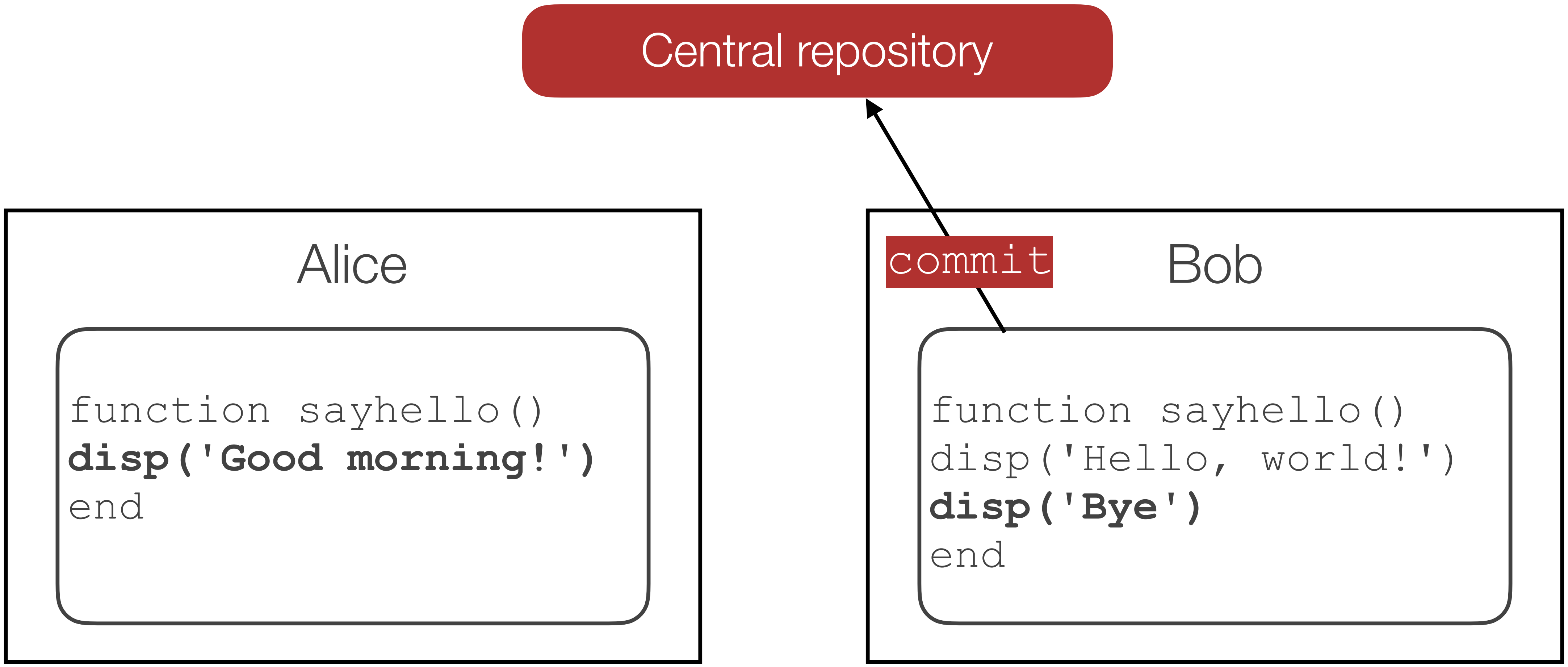
Sidenote: Merges

Centralized VCS will prevent simultaneous commits

- in fact, "update" locks the repository for a particular user

Resolution:

- Alice must wait for Bob to commit
- Alice then updates her working directory with Bob' changes
- Alice must "merge" Bob's changes with hers



```
function sayhello()
disp('Good morning!')
end
```

```
function sayhello()
disp('Hello, world!')
disp('Bye')
end
```

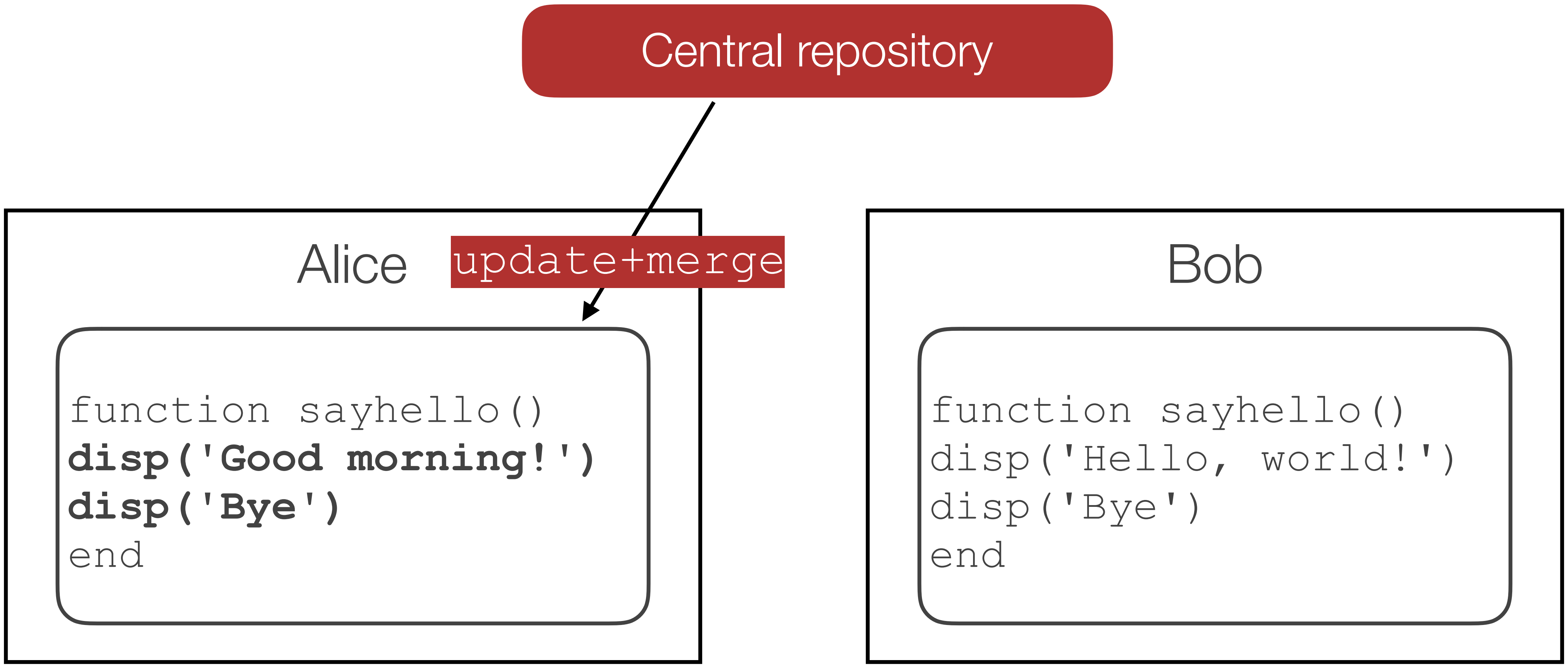
Sidenote: Merges

Centralized VCS will prevent simultaneous commits

- in fact, "update" locks the repository for a particular user

Resolution:

- Alice must wait for Bob to commit
- Alice then updates her working directory with Bob' changes
- Alice must "merge" Bob's changes with hers



Central repository

Alice

update+merge

```
function sayhello()
disp('Good morning!')
disp('Bye')
end
```

Bob

```
function sayhello()
disp('Hello, world!')
disp('Bye')
end
```

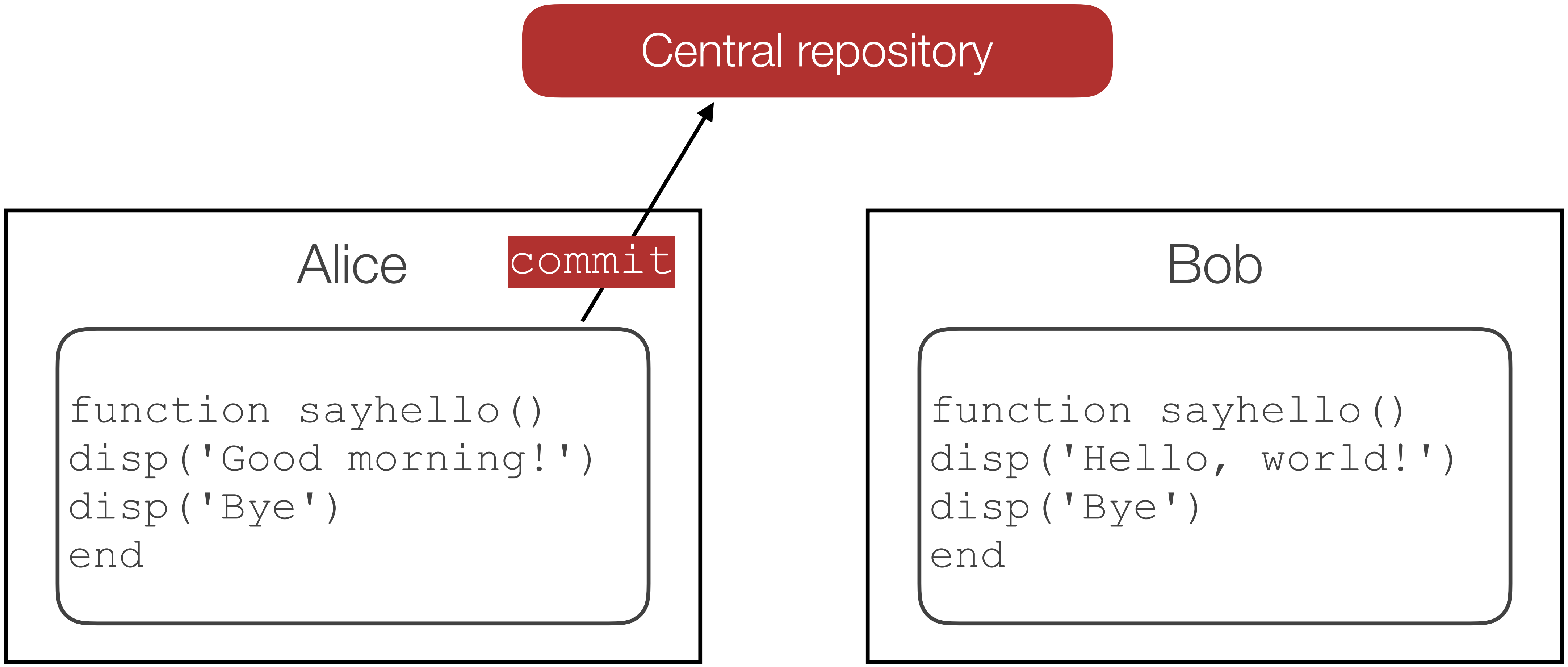
Sidenote: Merges

Centralized VCS will prevent simultaneous commits

- in fact, "update" locks the repository for a particular user

Resolution:

- Alice must wait for Bob to commit
- Alice then updates her working directory with Bob' changes
- Alice must "merge" Bob's changes with hers



```
function sayhello()
disp('Good morning!')
disp('Bye')
end
```

```
function sayhello()
disp('Hello, world!')
disp('Bye')
end
```

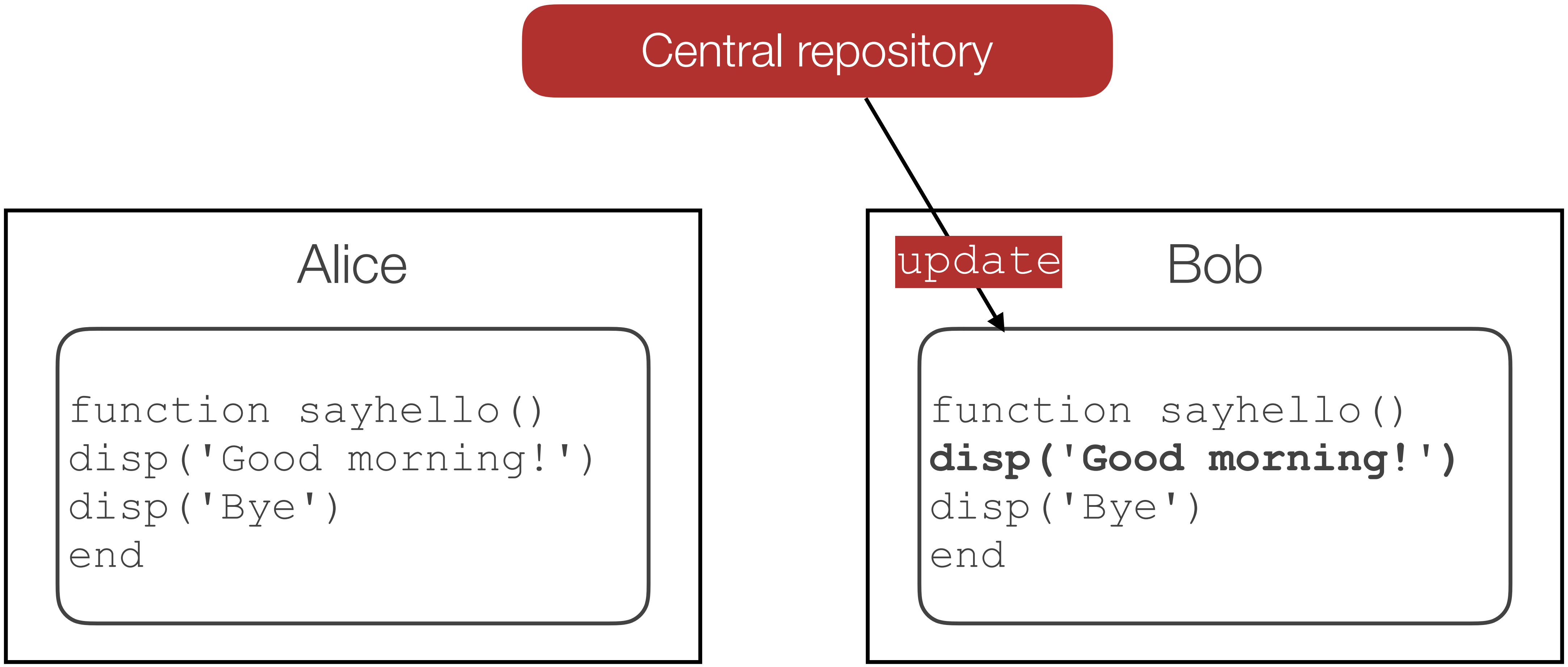
Sidenote: Merges

Centralized VCS will prevent simultaneous commits

- in fact, "update" locks the repository for a particular user

Resolution:

- Alice must wait for Bob to commit
- Alice then updates her working directory with Bob' changes
- Alice must "merge" Bob's changes with hers

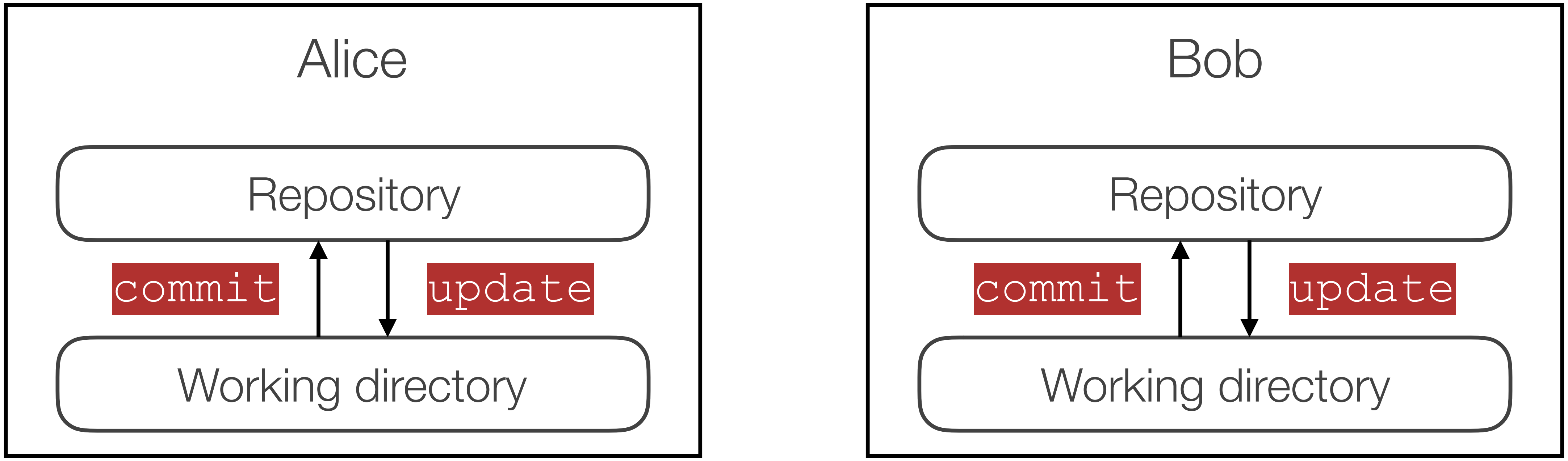


```
function sayhello()
disp('Good morning!')
disp('Bye')
end
```

```
function sayhello()
disp('Good morning!')
disp('Bye')
end
```

Distributed VCS (Mercurial, Git)

Main idea: everybody has a local copy of the repository



Alice

Repository

`commit`

`update`

Working directory

Bob

Repository

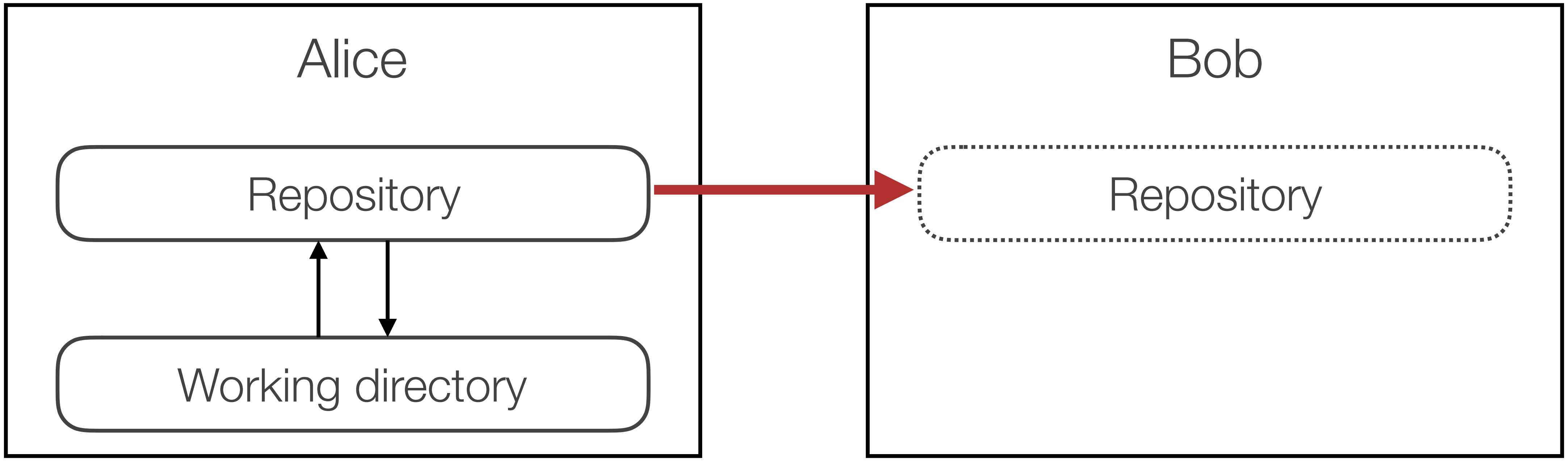
`commit`

`update`

Working directory

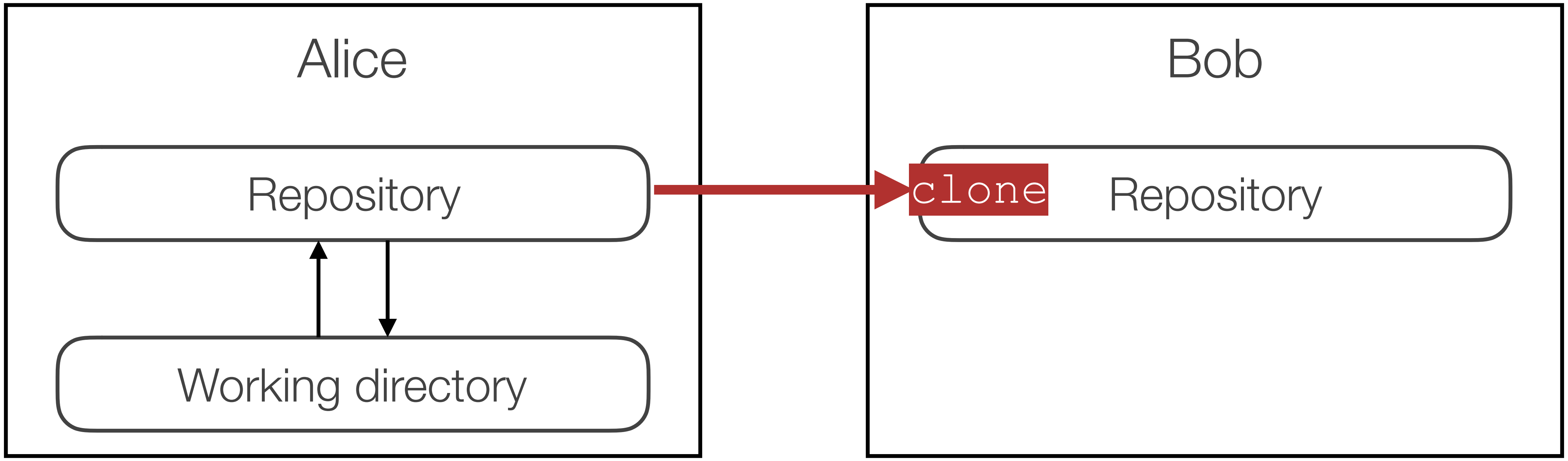
Distributed VCS (Mercurial, Git)

Main idea: everybody has a local copy of the repository



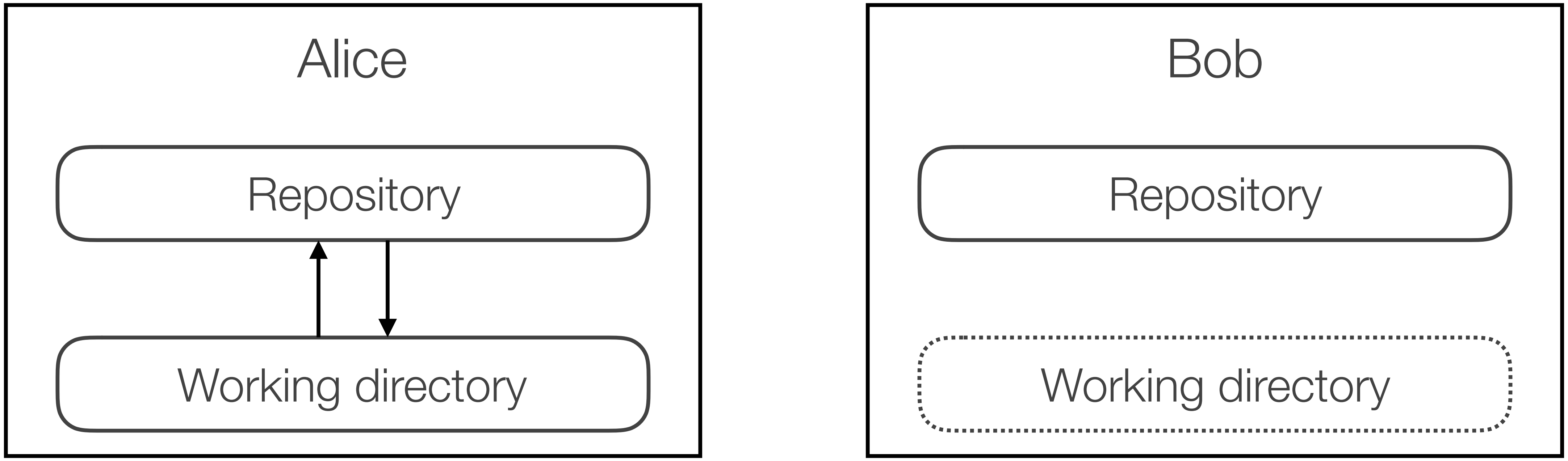
Distributed VCS (Mercurial, Git)

Main idea: everybody has a local copy of the repository



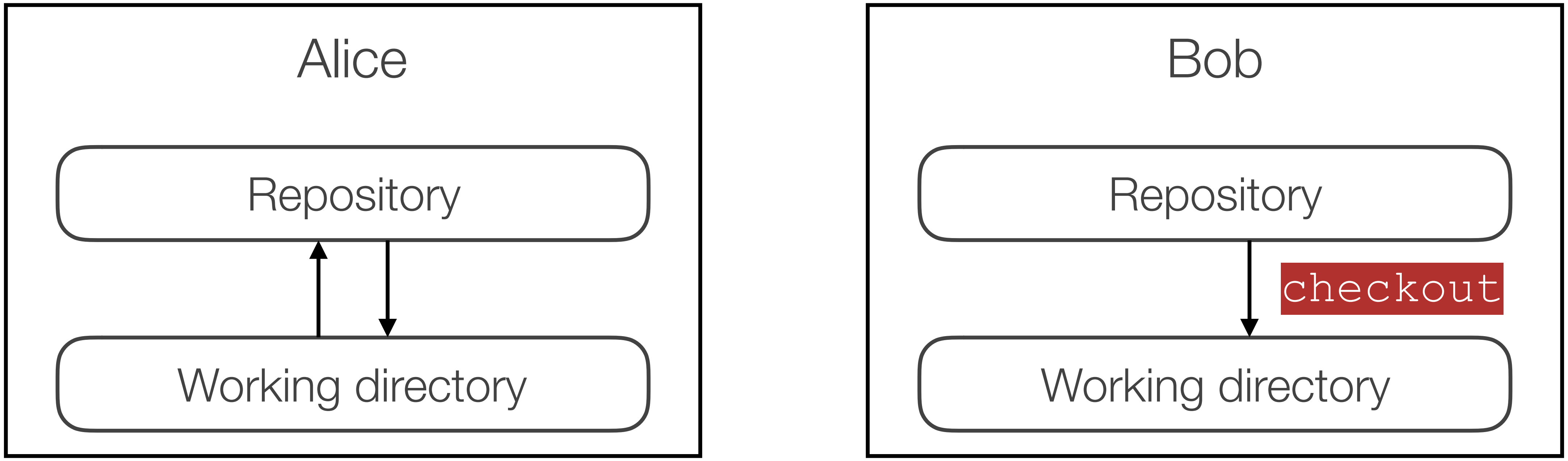
Distributed VCS (Mercurial, Git)

Main idea: everybody has a local copy of the repository



Distributed VCS (Mercurial, Git)

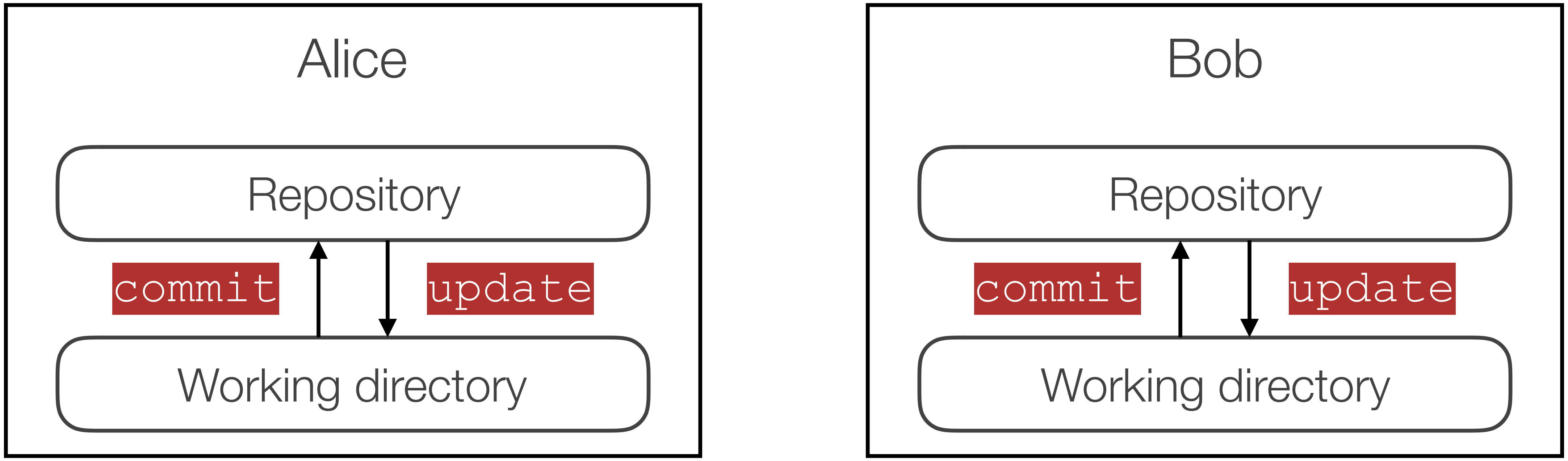
Main idea: everybody has a local copy of the repository



Distributed VCS (Mercurial, Git)

Main idea: everybody has a local copy of the repository

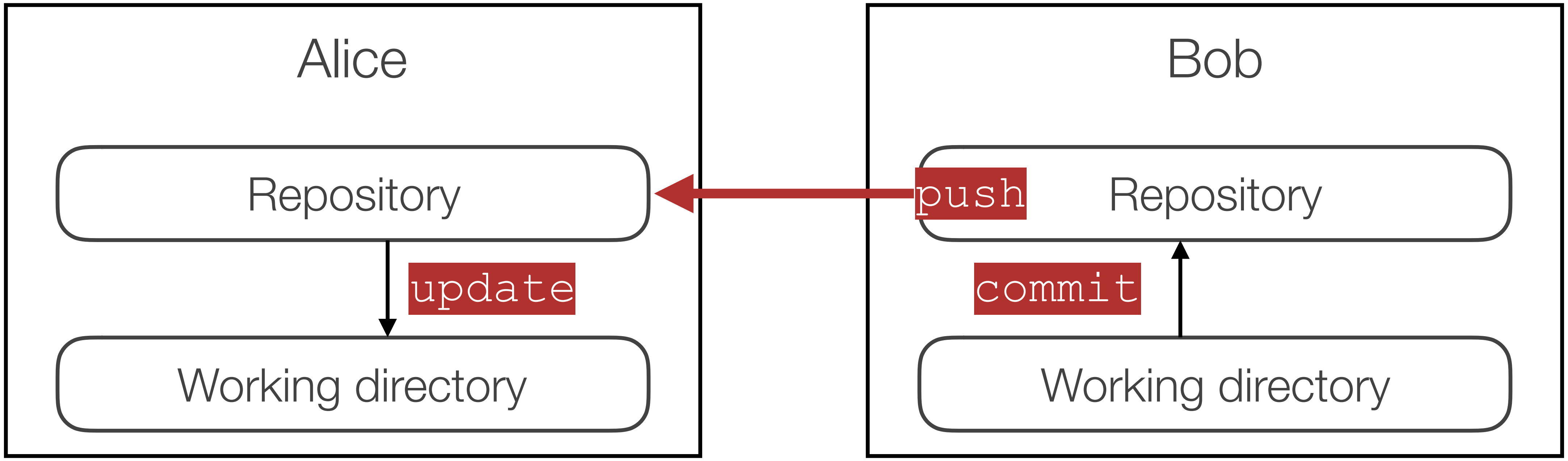
- every developer can commit/update at any time to his/her repository
- no permanent connection required



Distributed VCS (Mercurial, Git)

Main idea: everybody has a local copy of the repository

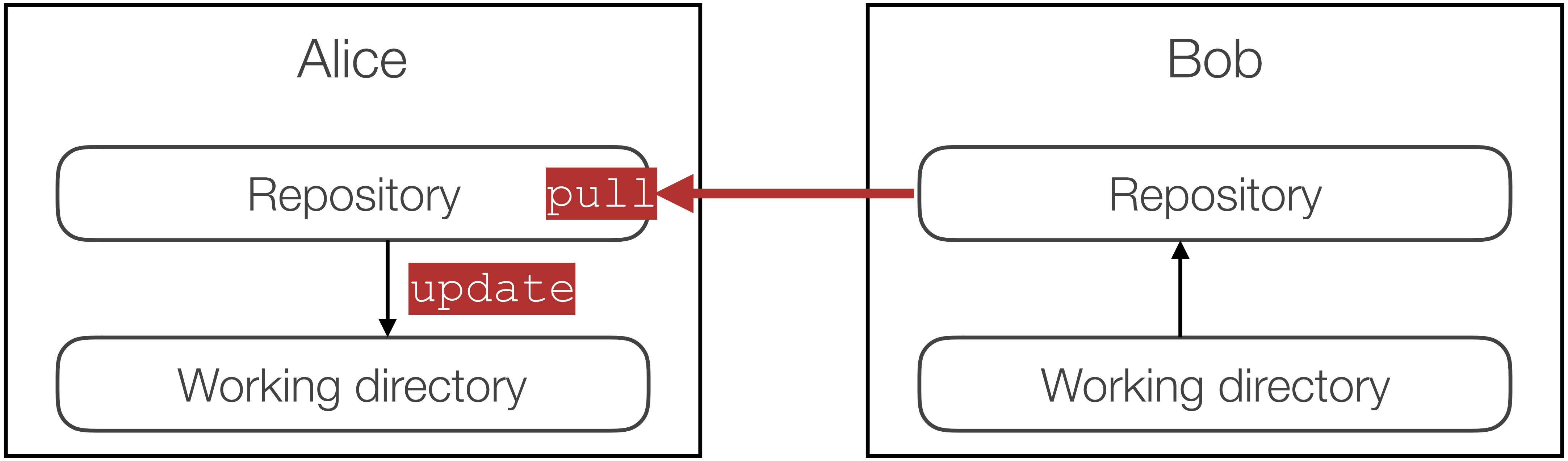
- every developer can commit/update at any time to his/her repository
- no permanent connection required



Distributed VCS (Mercurial, Git)

Main idea: everybody has a local copy of the repository

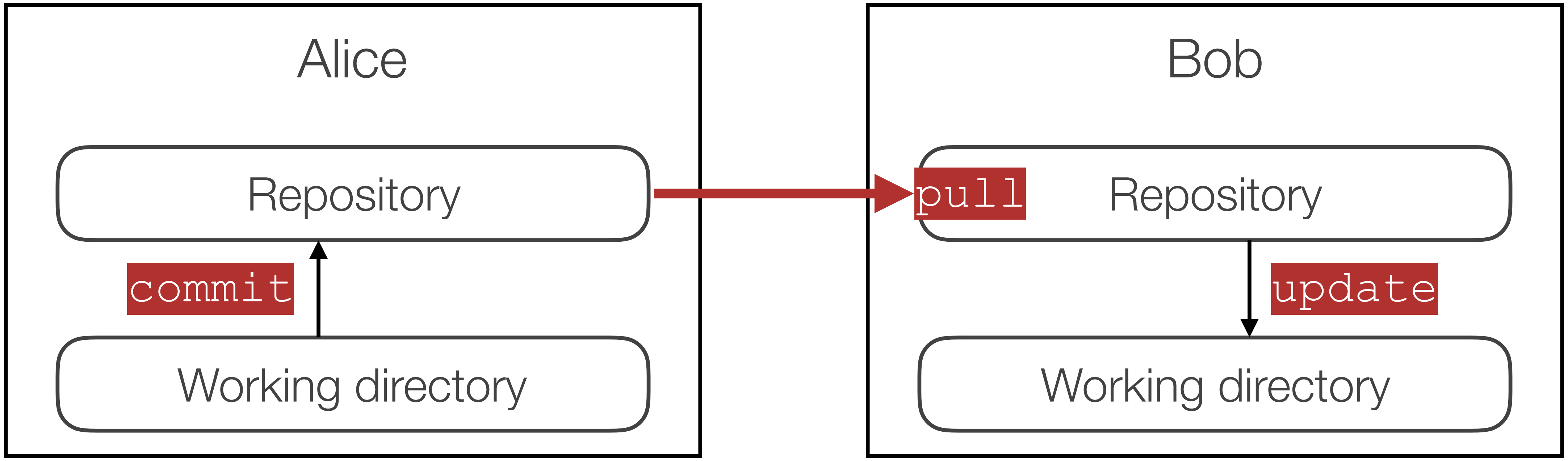
- every developer can commit/update at any time to his/her repository
- no permanent connection required



Distributed VCS (Mercurial, Git)

Main idea: everybody has a local copy of the repository

- every developer can commit/update at any time to his/her repository
- no permanent connection required

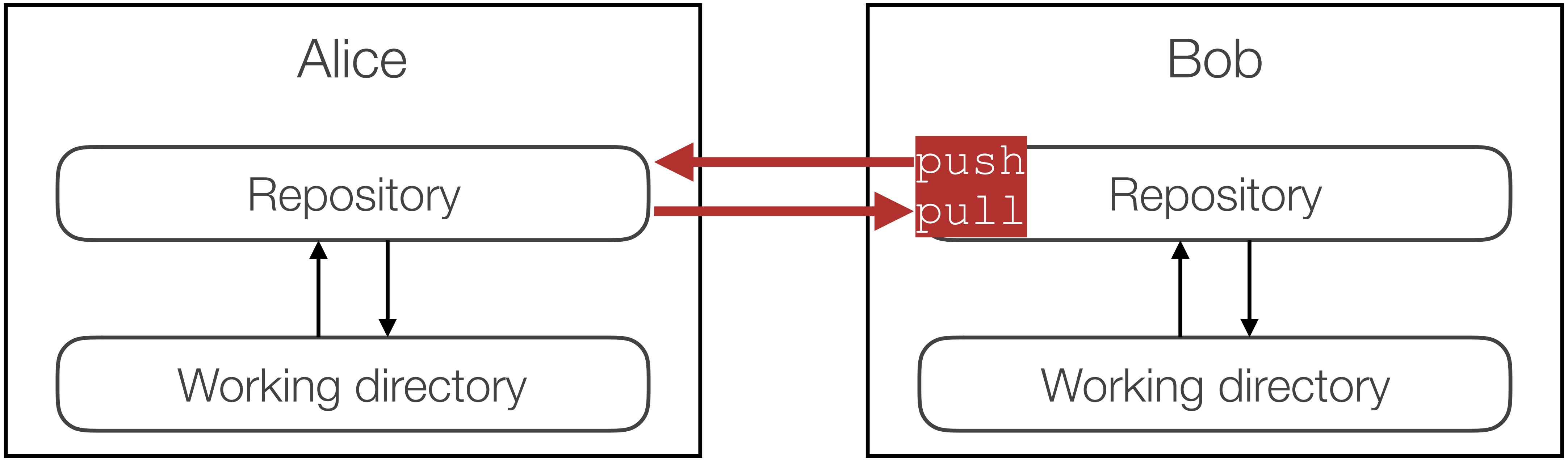


Distributed VCS (Mercurial, Git)

Main idea: everybody has a local copy of the repository

- every developer can commit/update at any time to his/her repository
- no permanent connection required

Rule of thumb: never push to a remote repository, let its owner pull from yours



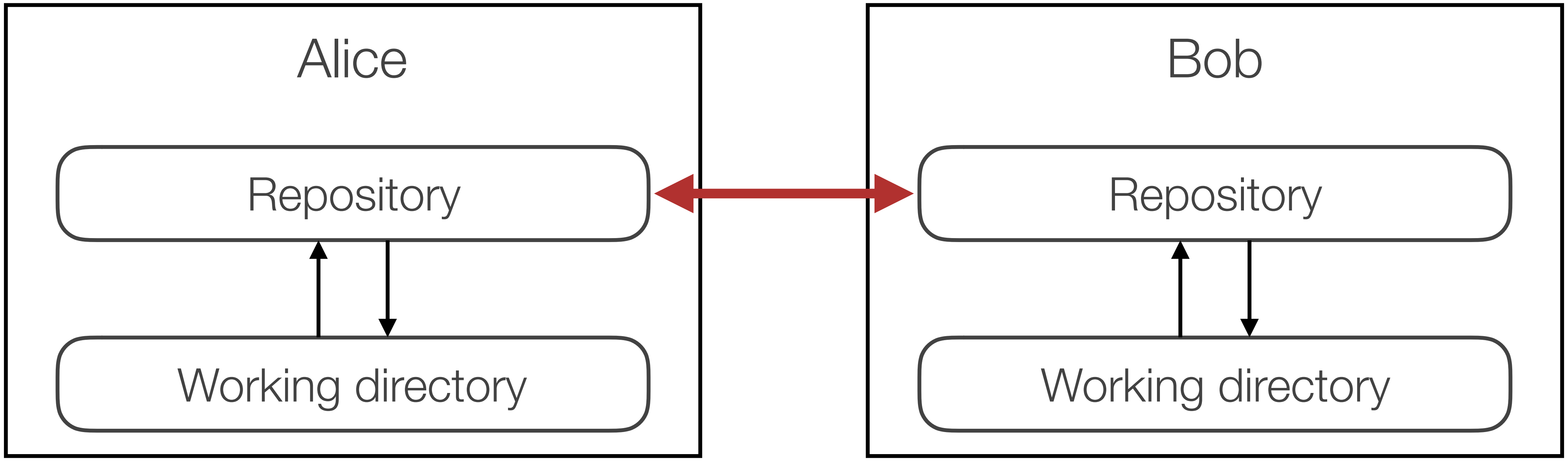
Distributed VCS (Mercurial, Git)

Main idea: everybody has a local copy of the repository

- every developer can commit/update at any time to his/her repository
- no permanent connection required

Questions:

- access rights management?
- communication ("Hey, I made a change, you need to update")?
- which repository is the "blessed" one for production code?



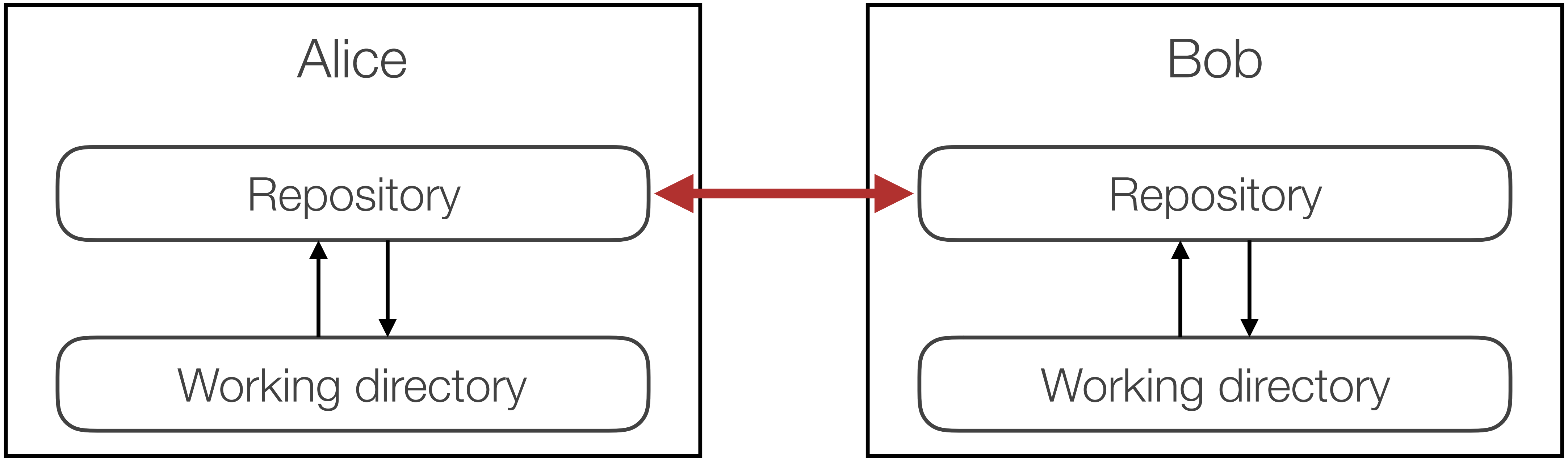
Distributed VCS (Mercurial, Git)

Main idea: everybody has a local copy of the repository

- every developer can commit/update at any time to his/her repository
- no permanent connection required

Questions:

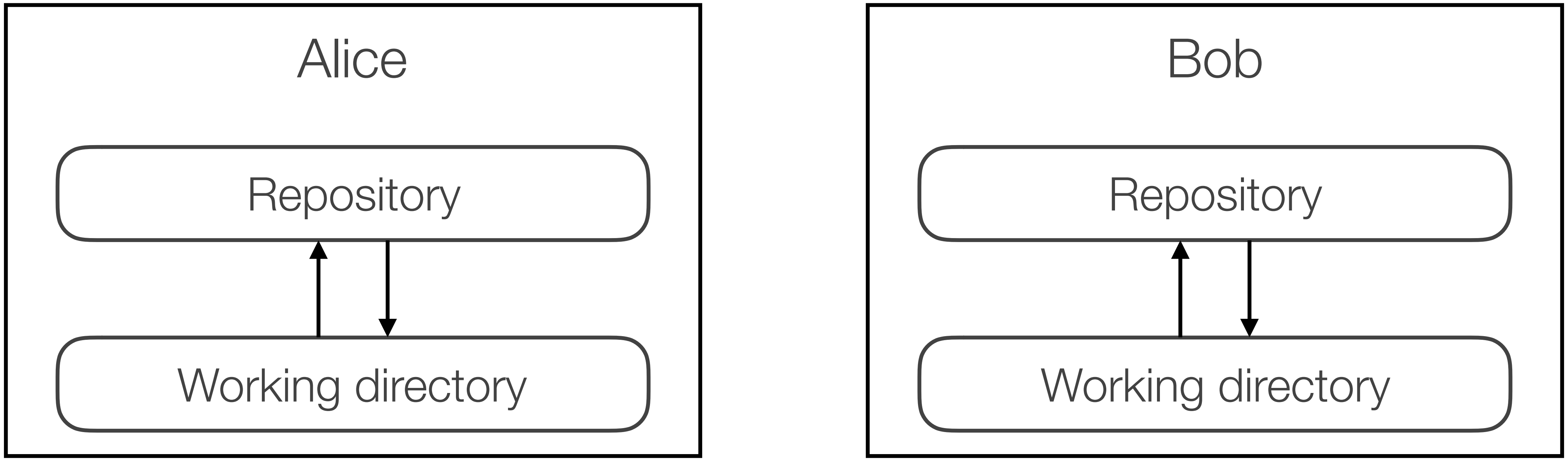
- access rights management?
- communication ("Hey, I made a change, you need to update")?
- **which repository is the "blessed" one for production code?**



Distributed VCS Workflows

Three possible workflows:

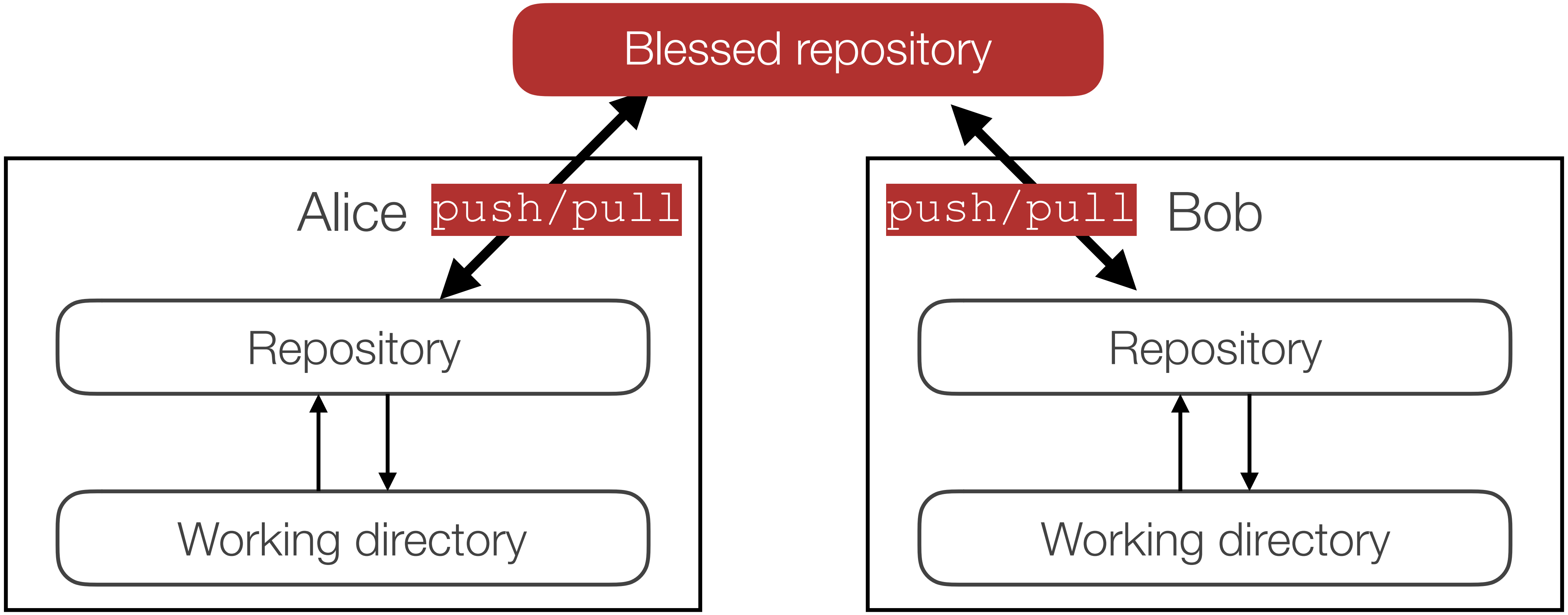
- centralized
- integration manager
- dictator / lieutenants



Centralized Workflow

Idea: use a central "blessed" repository

- but each developer has his/her own copy (no single point of failure)



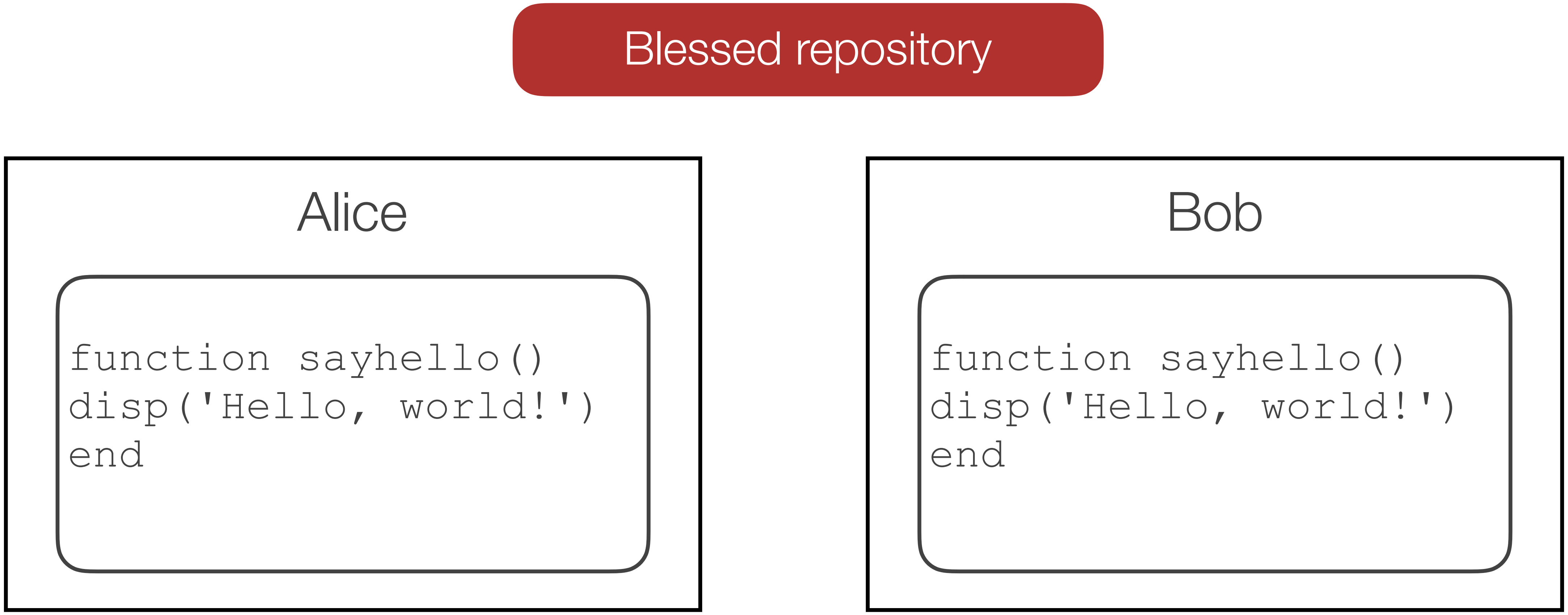
Centralized Workflow

Idea: use a central "blessed" repository

- but each developer has his/her own copy (no single point of failure)

Problems:

- cannot push conflicting changes



Blessed repository

Alice

```
function sayhello()
disp('Hello, world!')
end
```

Bob

```
function sayhello()
disp('Hello, world!')
end
```

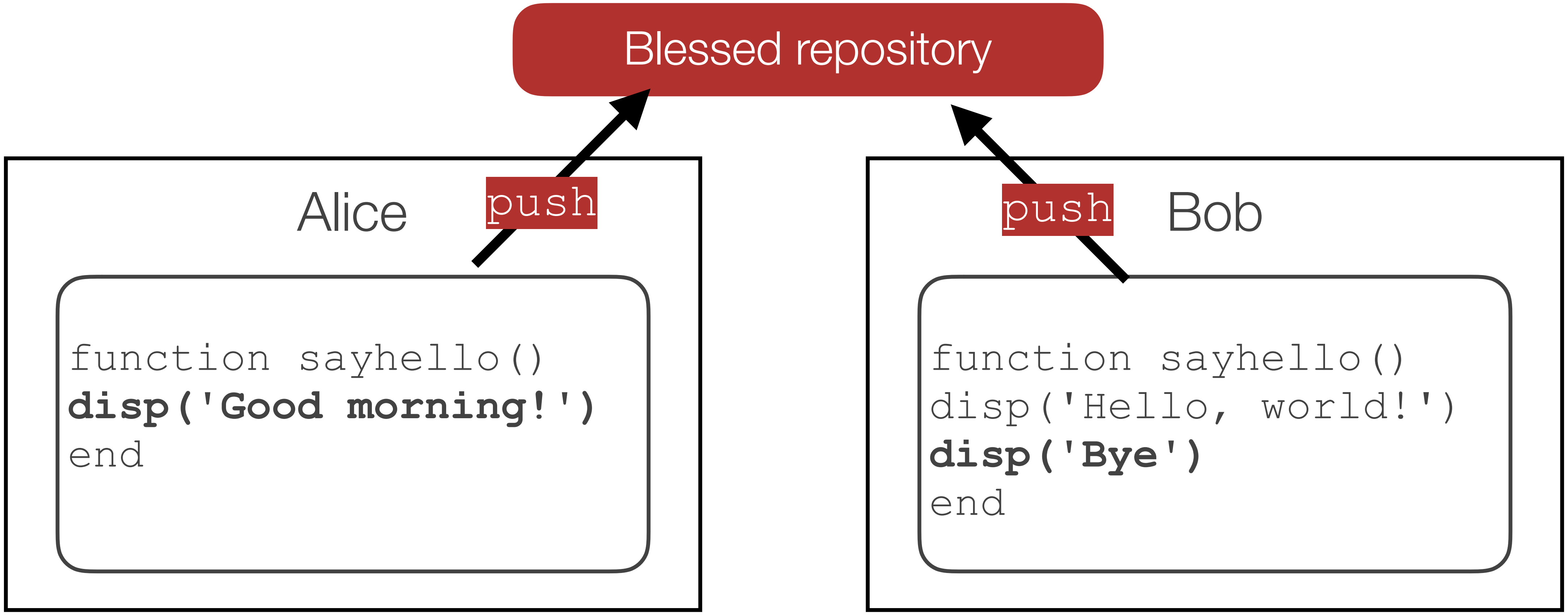
Centralized Workflow

Idea: use a central "blessed" repository

- but each developer has his/her own copy (no single point of failure)

Problems:

- cannot push conflicting changes



```
function sayhello()
disp('Good morning!')
end
```

```
function sayhello()
disp('Hello, world!')
disp('Bye')
end
```

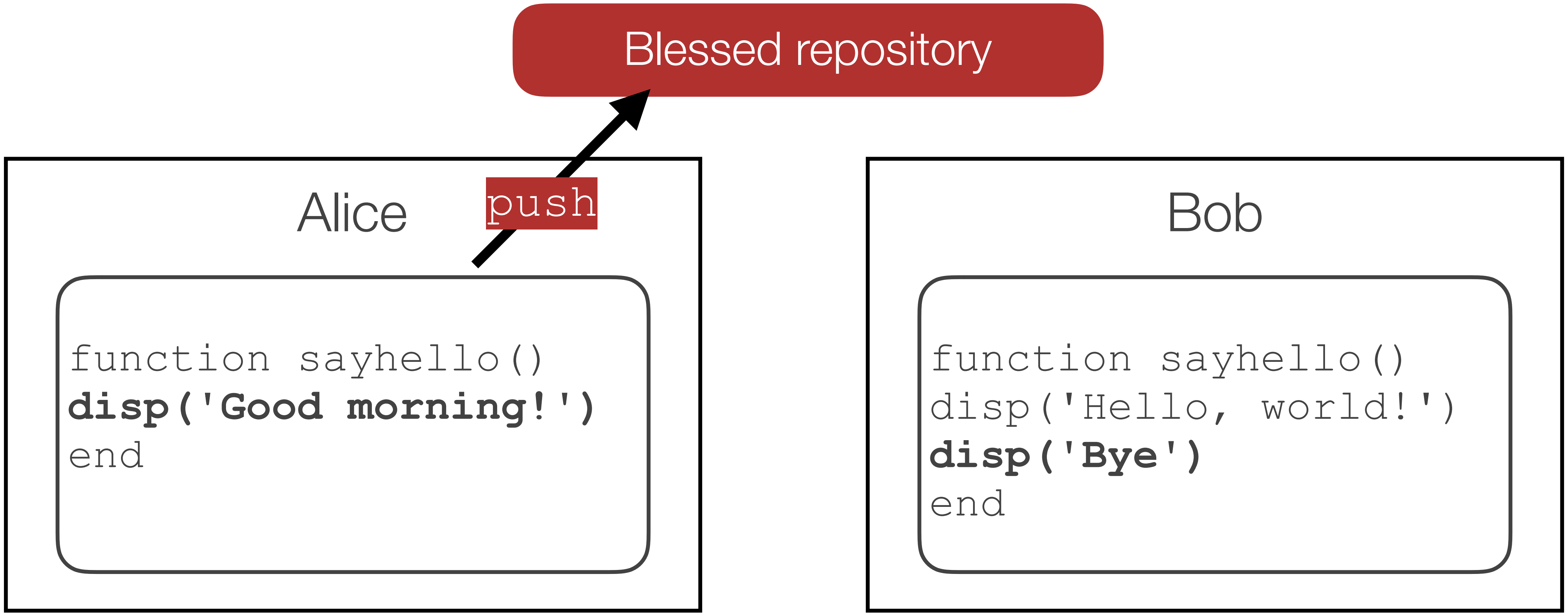
Centralized Workflow

Idea: use a central "blessed" repository

- but each developer has his/her own copy (no single point of failure)

Problems:

- cannot push conflicting changes



Blessed repository

push

Alice

```
function sayhello()
disp('Good morning!')
end
```

Bob

```
function sayhello()
disp('Hello, world!')
disp('Bye')
end
```

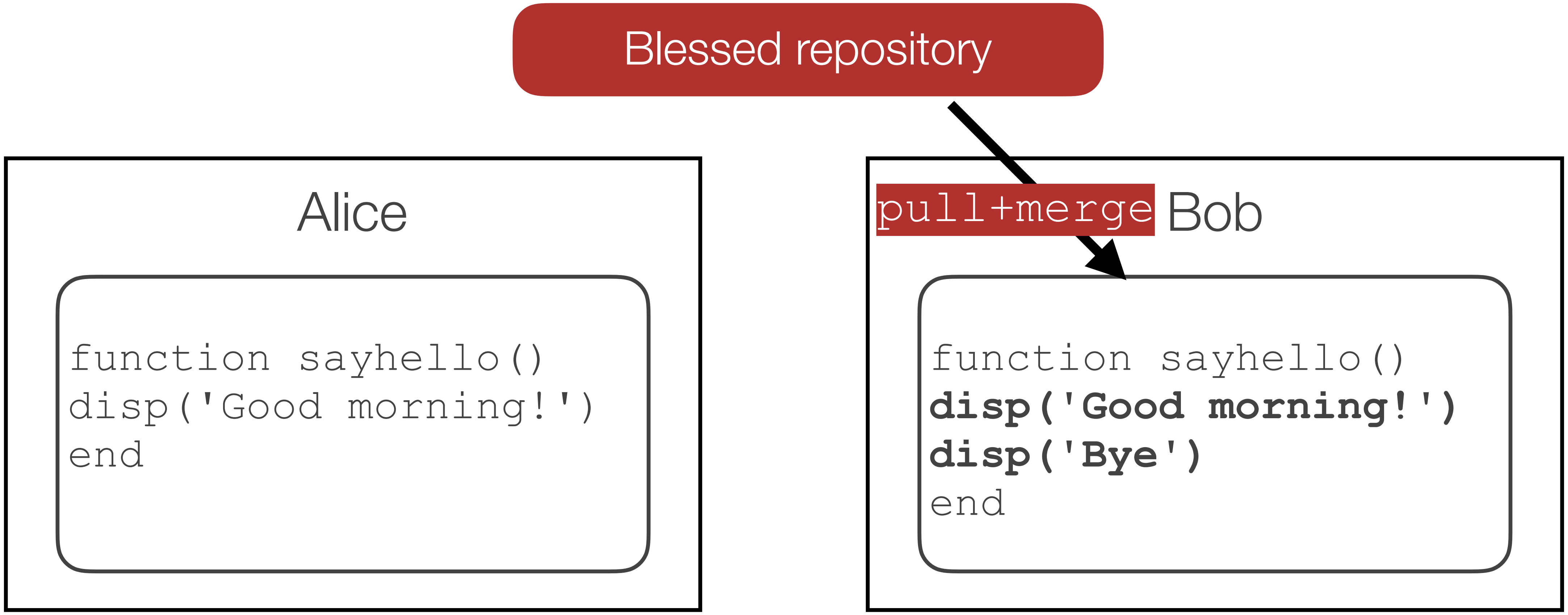
Centralized Workflow

Idea: use a central "blessed" repository

- but each developer has his/her own copy (no single point of failure)

Problems:

- cannot push conflicting changes



Blessed repository

Alice

```
function sayhello()
disp('Good morning!')
end
```

pull+merge Bob

```
function sayhello()
disp('Good morning!')
disp('Bye')
end
```

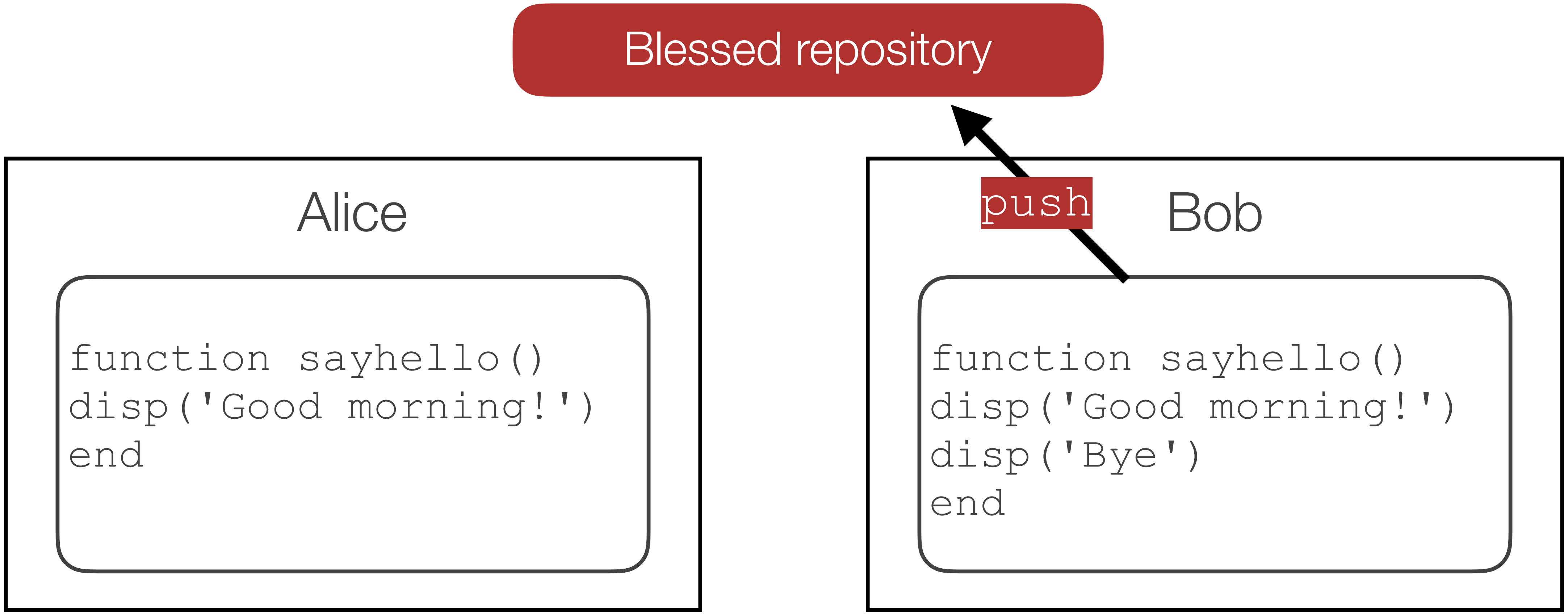
Centralized Workflow

Idea: use a central "blessed" repository

- but each developer has his/her own copy (no single point of failure)

Problems:

- cannot push conflicting changes



Blessed repository

push

Alice

```
function sayhello()
disp('Good morning!')
end
```

Bob

```
function sayhello()
disp('Good morning!')
disp('Bye')
end
```

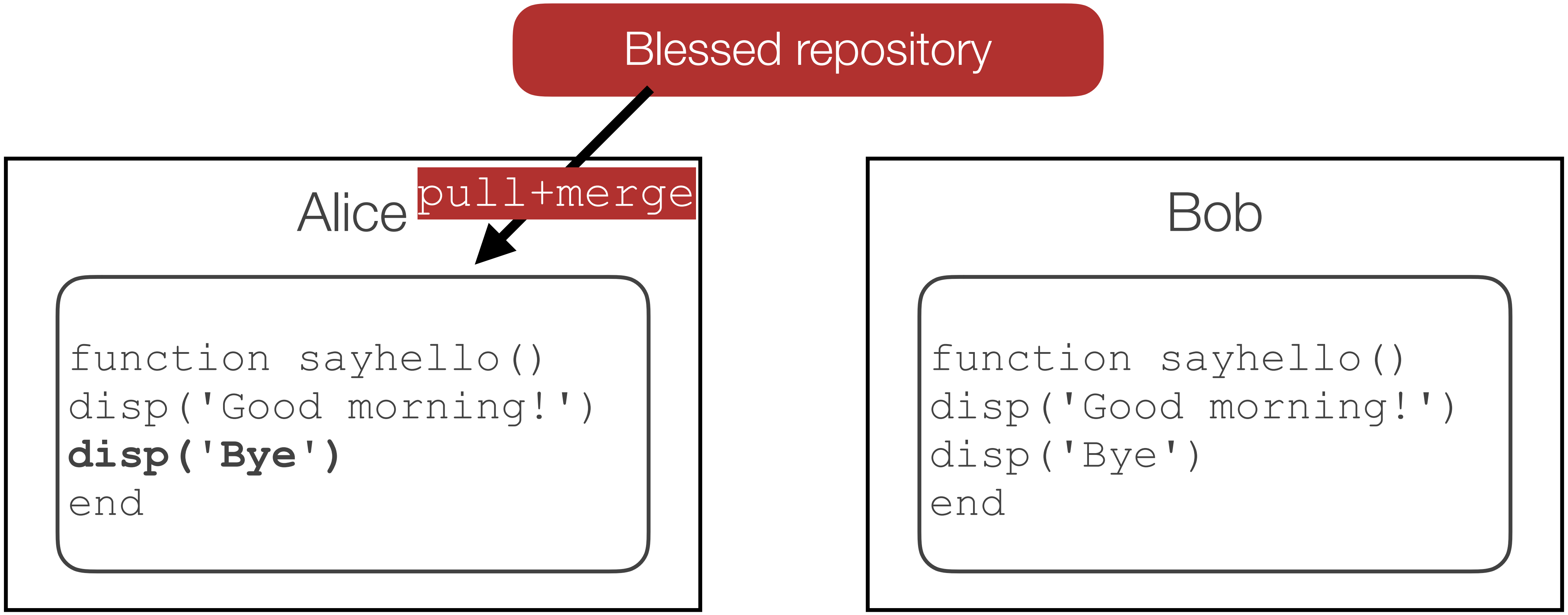
Centralized Workflow

Idea: use a central "blessed" repository

- but each developer has his/her own copy (no single point of failure)

Problems:

- cannot push conflicting changes



Blessed repository

pull+merge

Alice

```
function sayhello()
disp('Good morning!')
disp('Bye')
end
```

Bob

```
function sayhello()
disp('Good morning!')
disp('Bye')
end
```

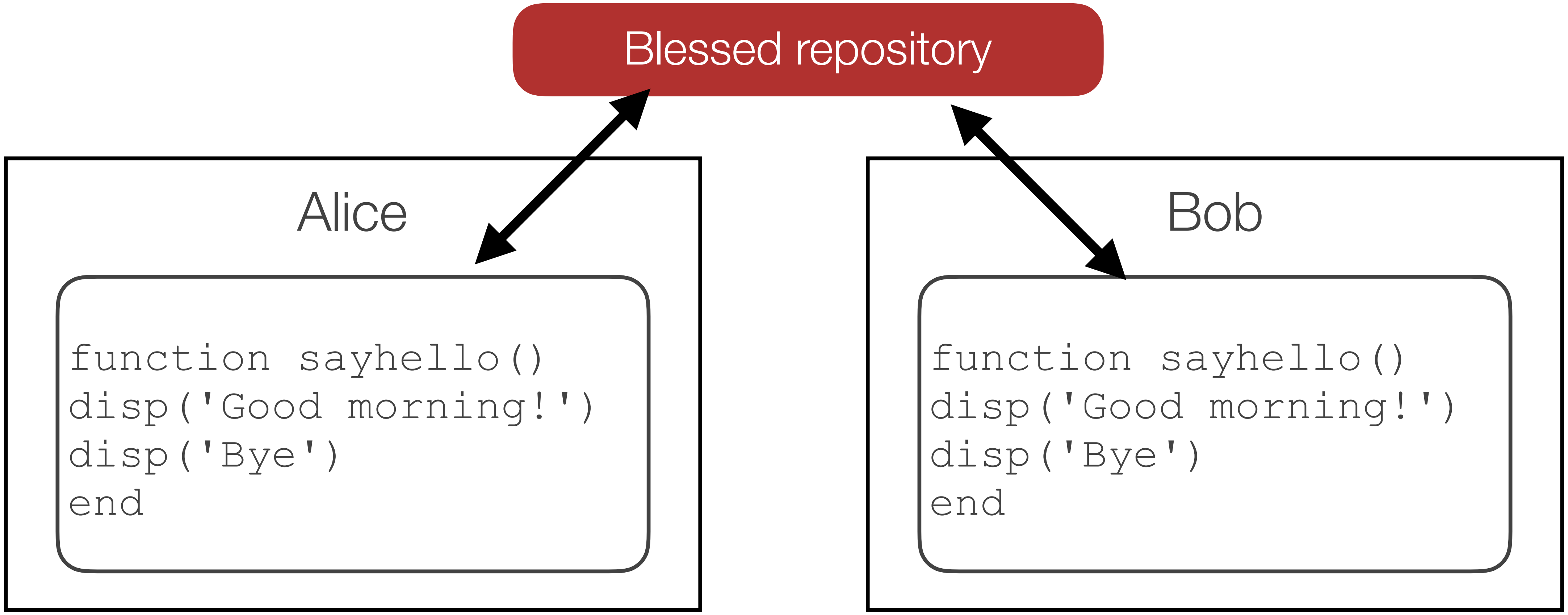
Centralized Workflow

Idea: use a central "blessed" repository

- but each developer has his/her own copy (no single point of failure)

Problems:

- cannot push conflicting changes (merging is required, often difficult!)
- only suitable for a small number of developers (<5)



Blessed repository

Alice

```
function sayhello()
disp('Good morning!')
disp('Bye')
end
```

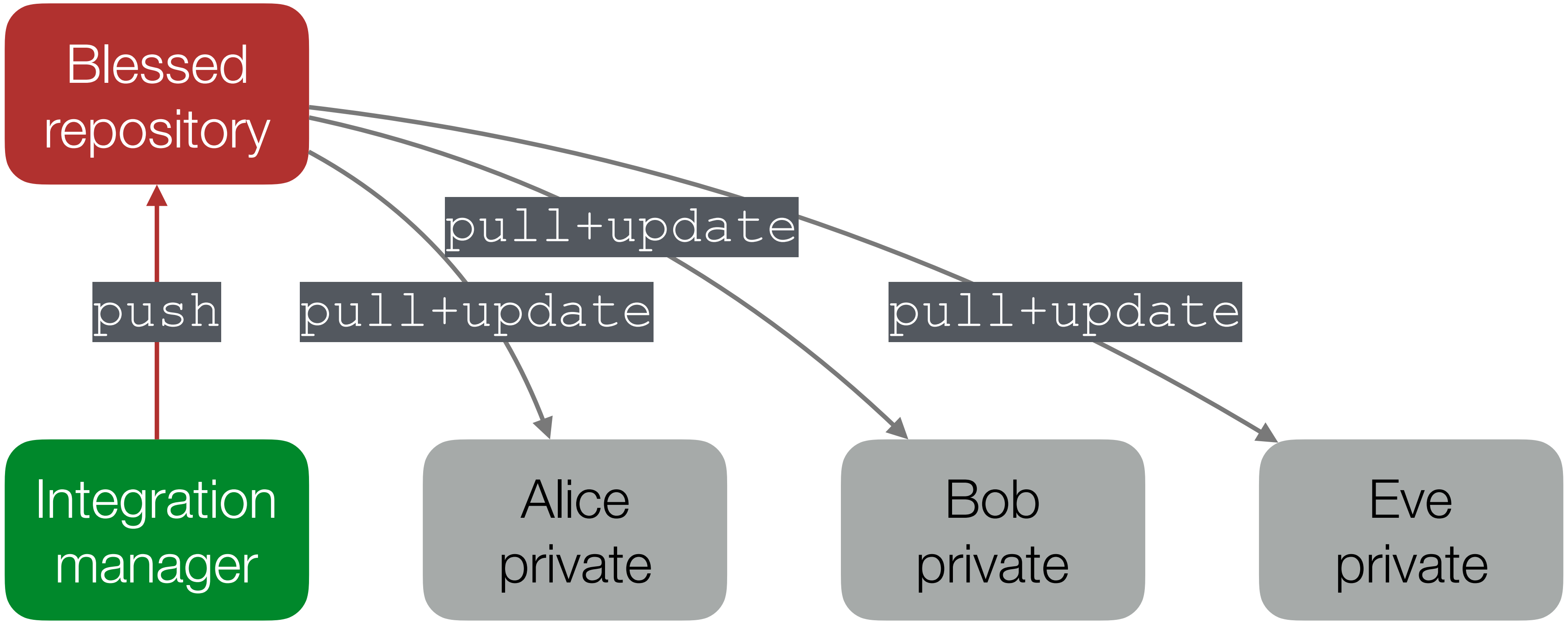
Bob

```
function sayhello()
disp('Good morning!')
disp('Bye')
end
```

Integration Manager Workflow

Idea: use a central "blessed" repository:

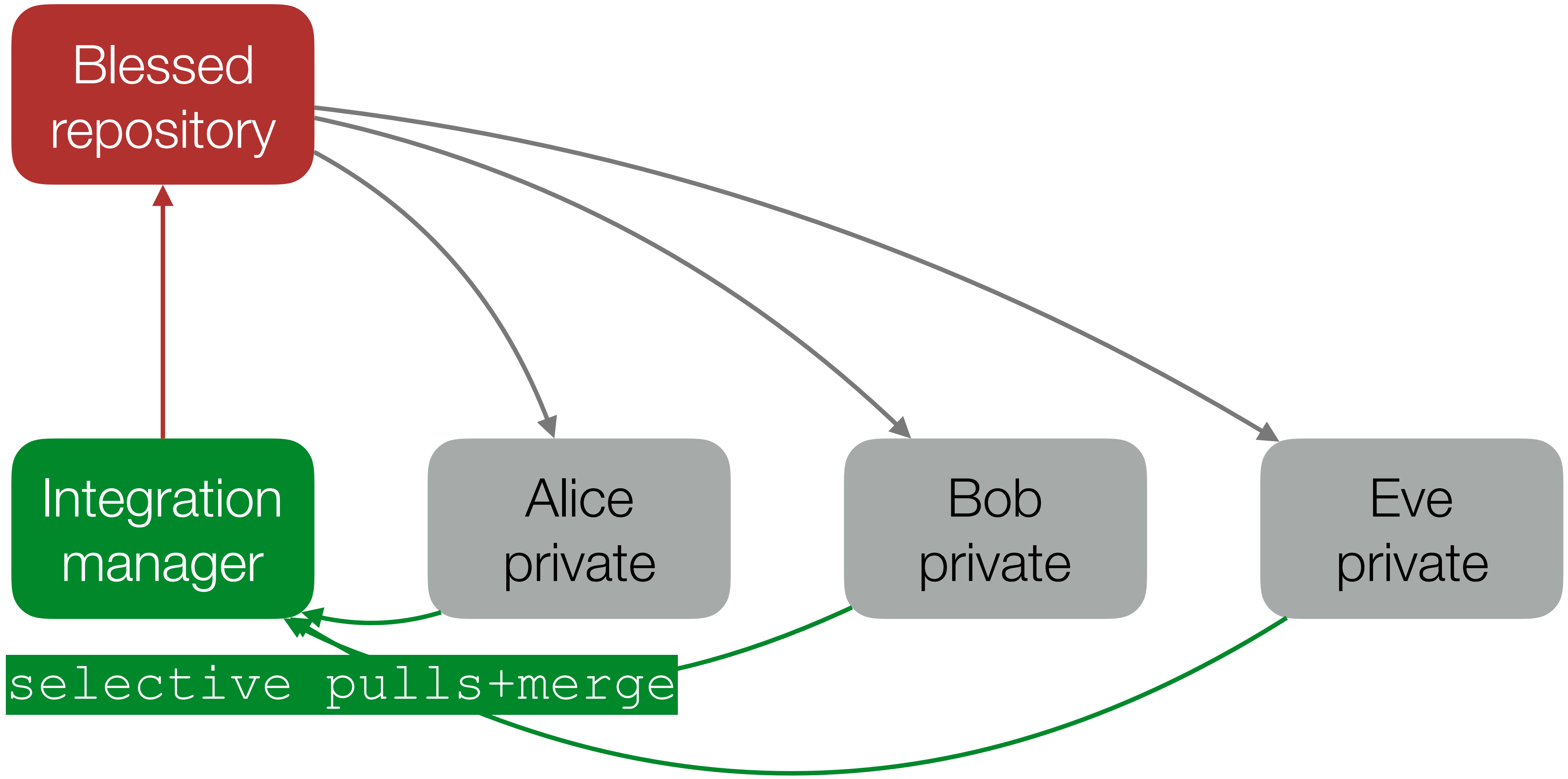
- but only the integration manager can push to it



Integration Manager Workflow

Idea: use a central "blessed" repository:

- but only the integration manager can push to it
- the manager cherry-picks changes and handles merges

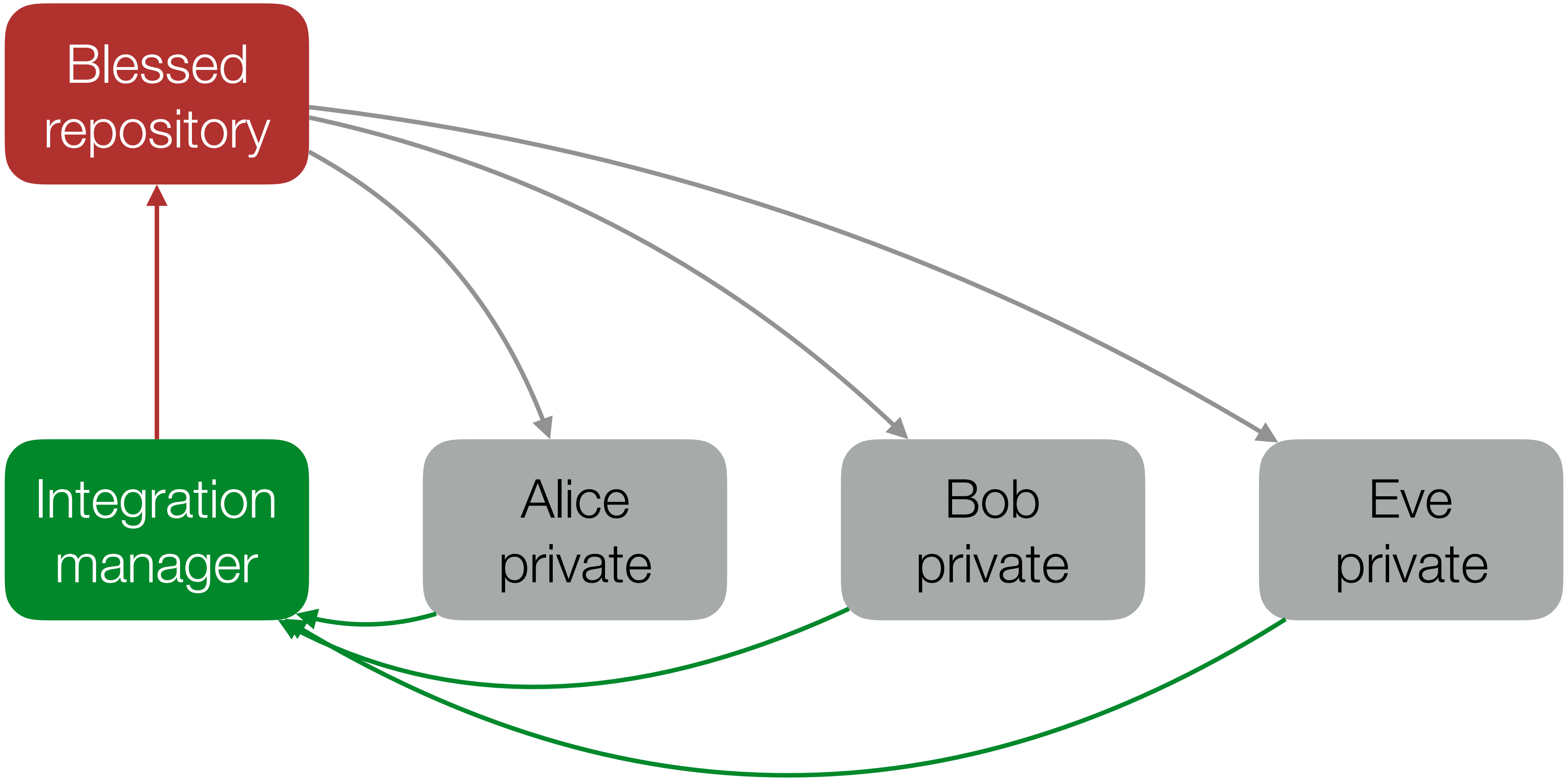


Integration Manager Workflow

Idea: use a central "blessed" repository:

- but only the integration manager can push to it
- the manager cherry-picks changes and handles merges

Problem: usually cannot pull from private repositories



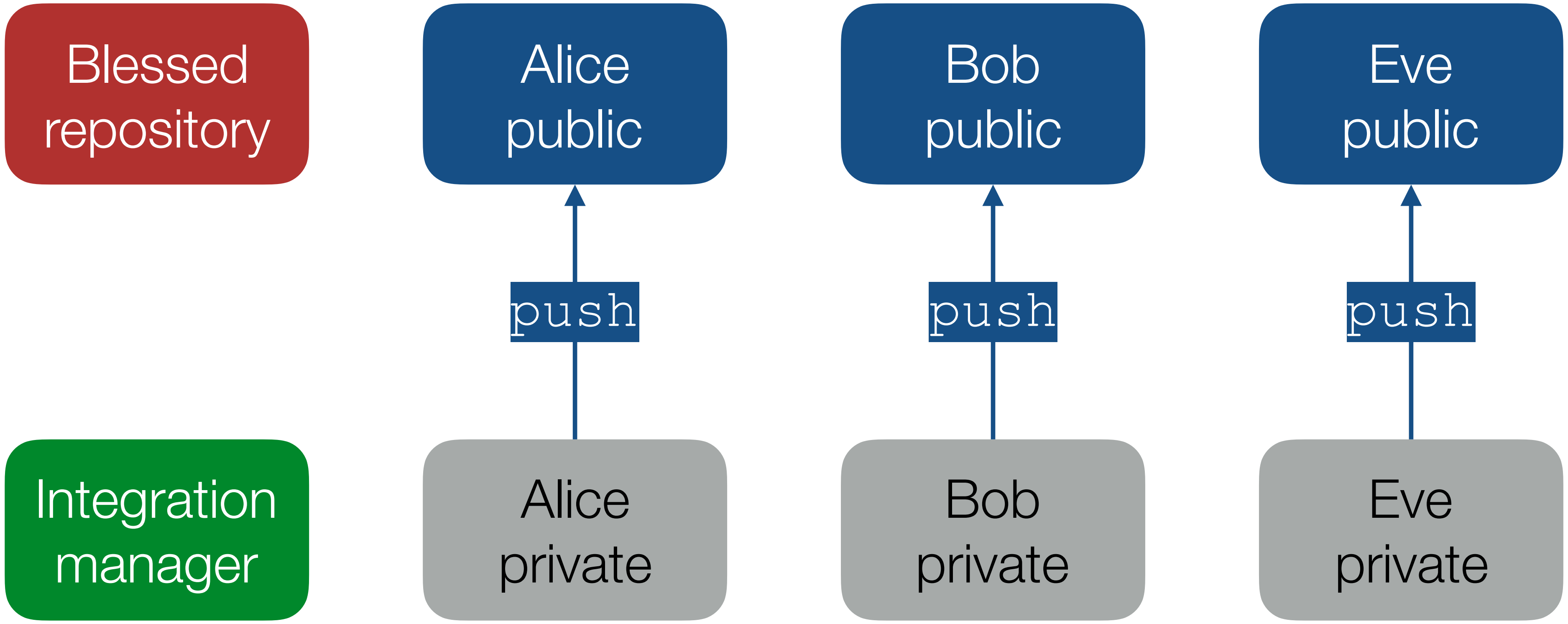
Integration Manager Workflow

Idea: use a central "blessed" repository:

- but only the integration manager can push to it
- the manager cherry-picks changes and handles merges

Problem: usually cannot pull from private repositories

Solution: every developer has a public version of his/her repository



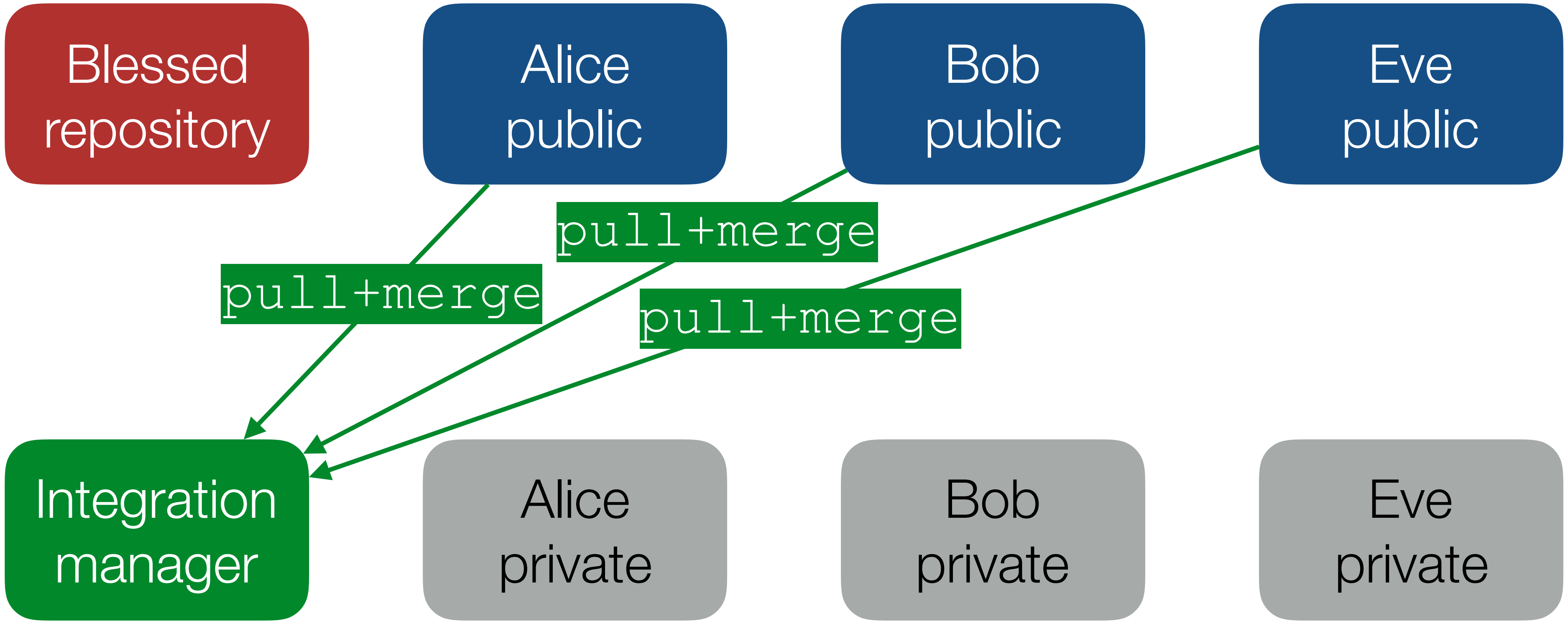
Integration Manager Workflow

Idea: use a central "blessed" repository:

- but only the integration manager can push to it
- the manager cherry-picks changes and handles merges

Problem: usually cannot pull from private repositories

Solution: every developer has a public version of his/her repository



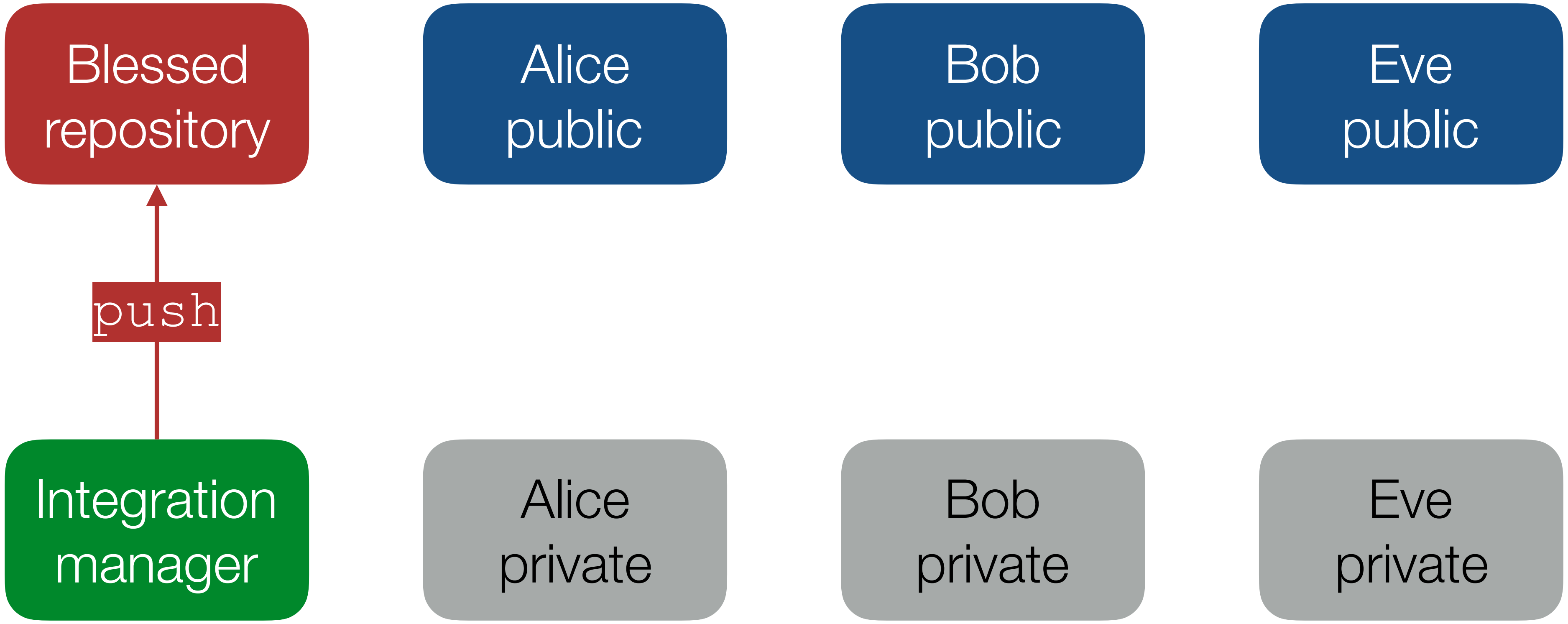
Integration Manager Workflow

Idea: use a central "blessed" repository:

- but only the integration manager can push to it
- the manager cherry-picks changes and handles merges

Problem: usually cannot pull from private repositories

Solution: every developer has a public version of his/her repository



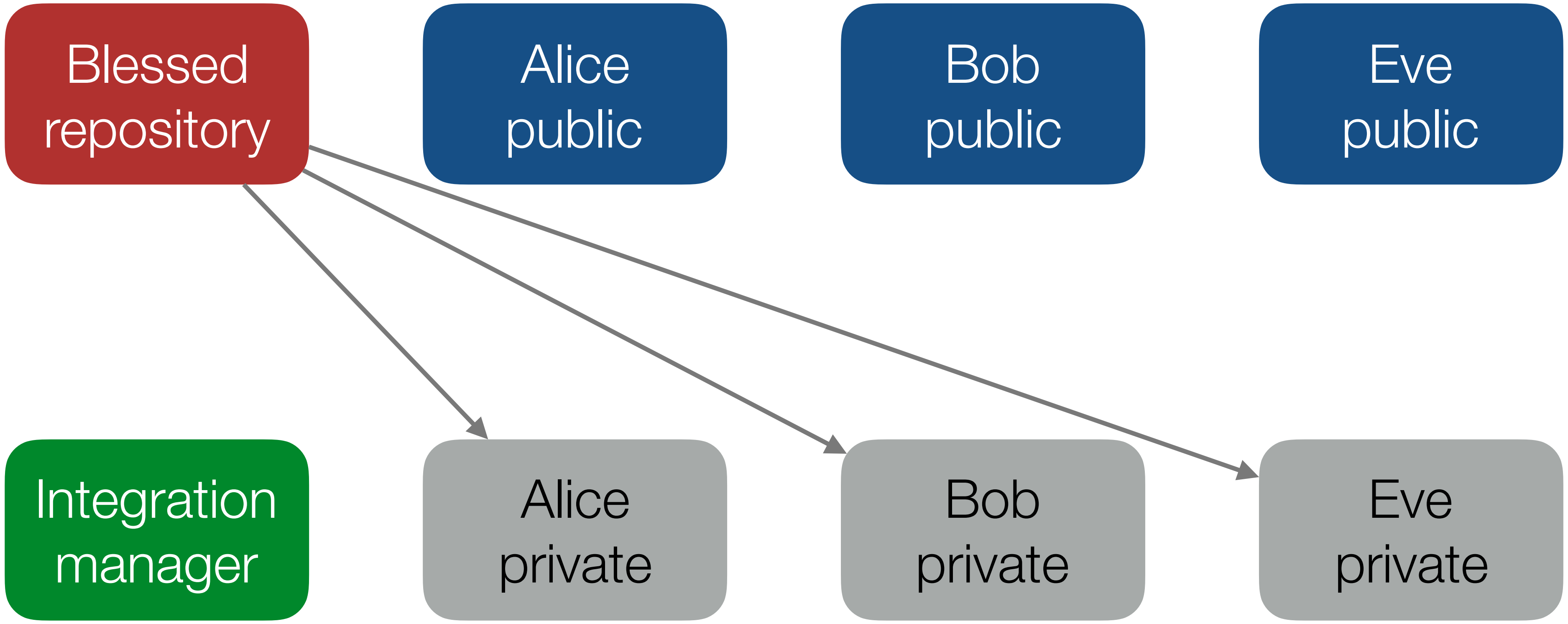
Integration Manager Workflow

Idea: use a central "blessed" repository:

- but only the integration manager can push to it
- the manager cherry-picks changes and handles merges

Problem: usually cannot pull from private repositories

Solution: every developer has a public version of his/her repository



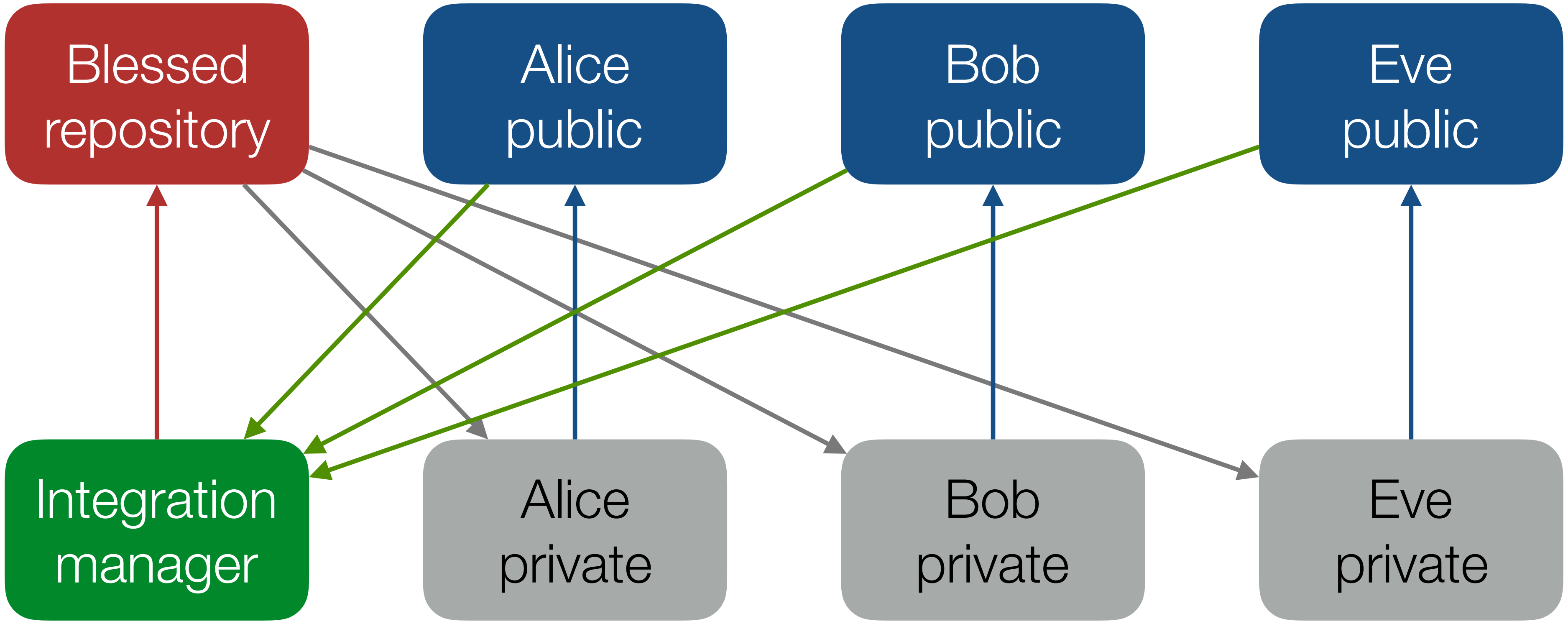
Integration Manager Workflow

Idea: use a central "blessed" repository:

- but only the integration manager can push to it
- the manager cherry-picks changes and handles merges

Problem: usually cannot pull from private repositories

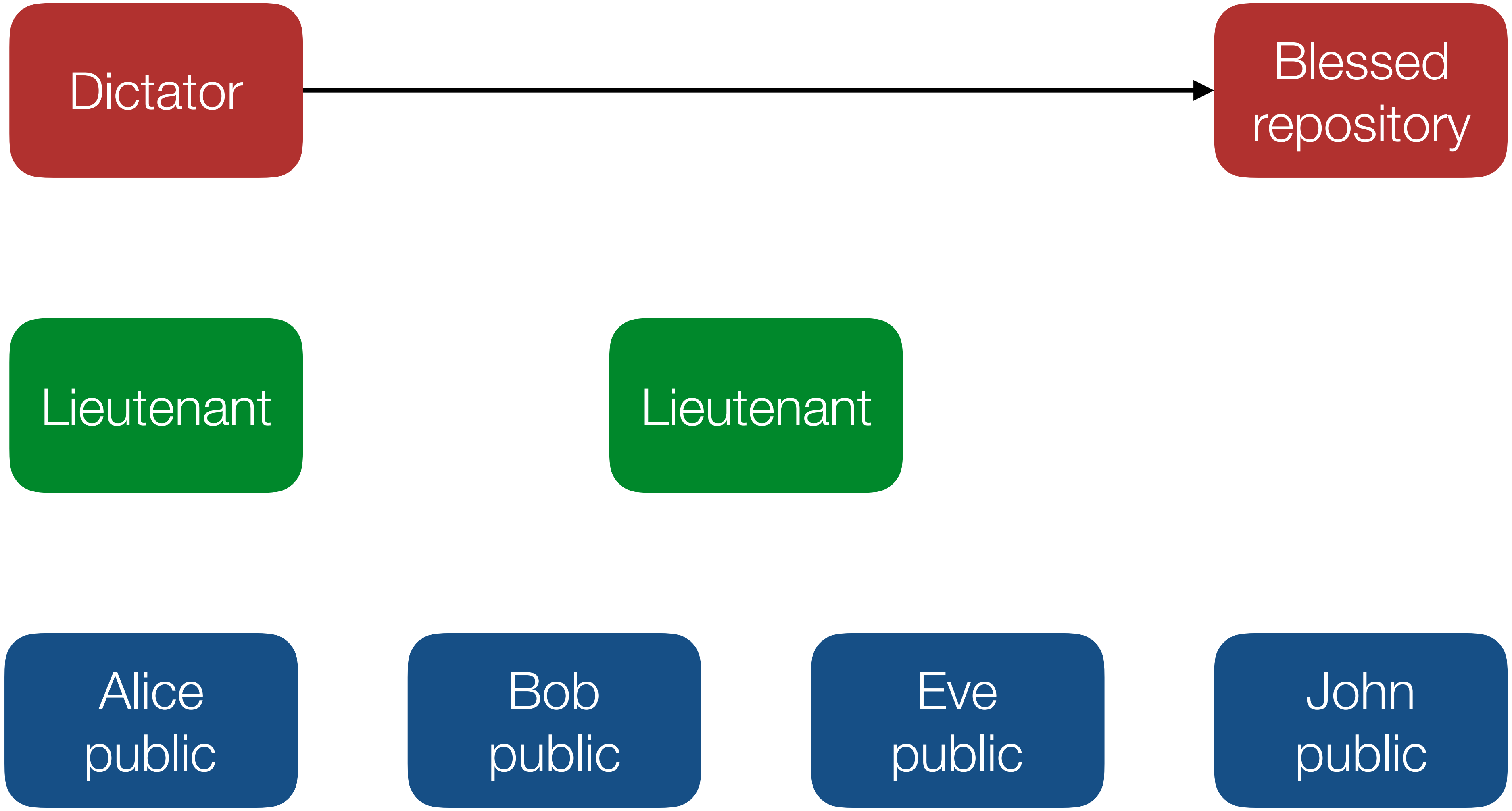
Solution: every developer has a public version of his/her repository



Dictator / Lieutenants Workflow

Idea: use multiple integration managers (aka lieutenants)

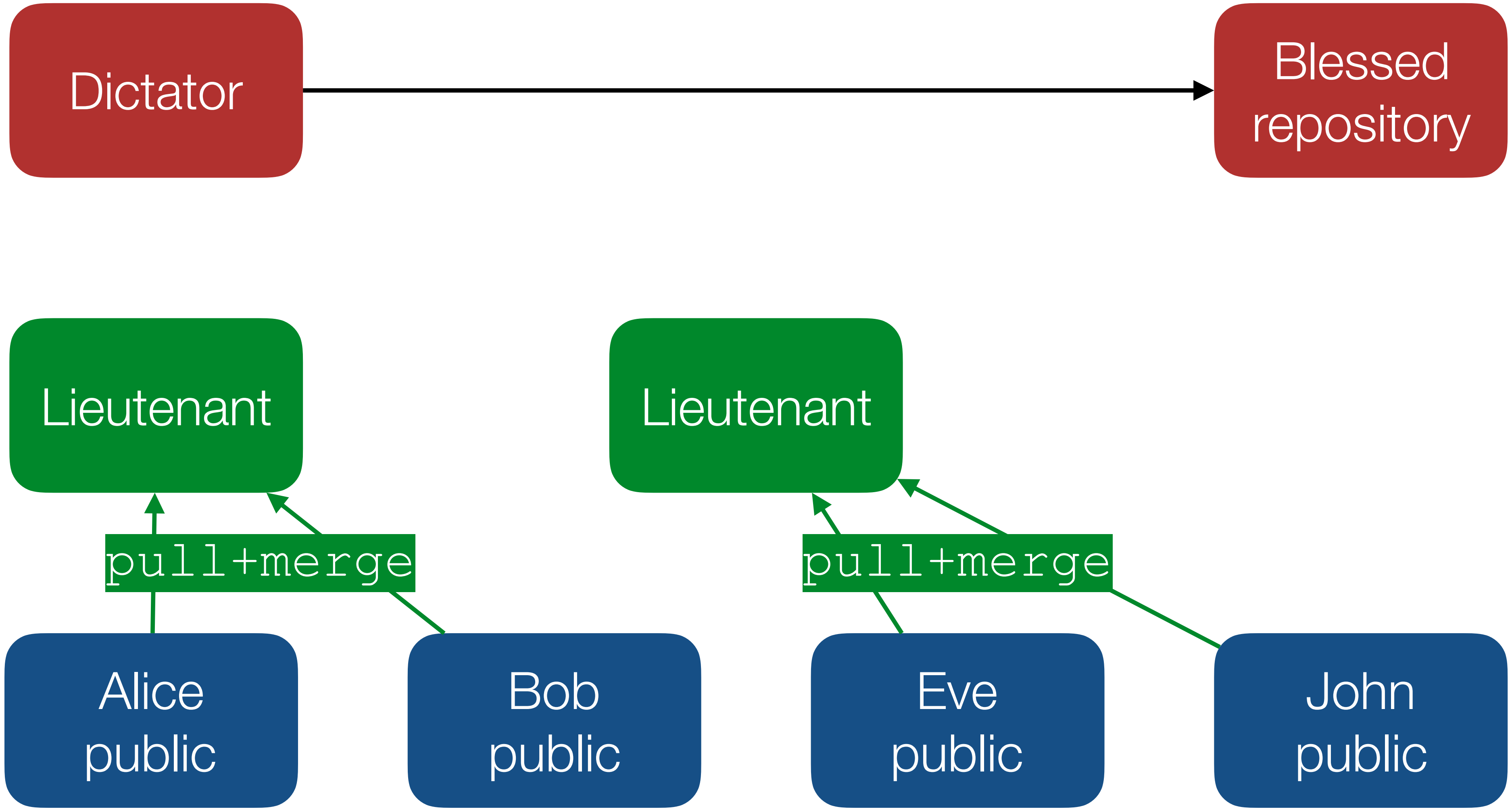
- each lieutenant is usually responsible for one module
- used for big projects (e.g. the Linux kernel, Python)



Dictator / Lieutenants Workflow

Idea: use multiple integration managers (aka lieutenants)

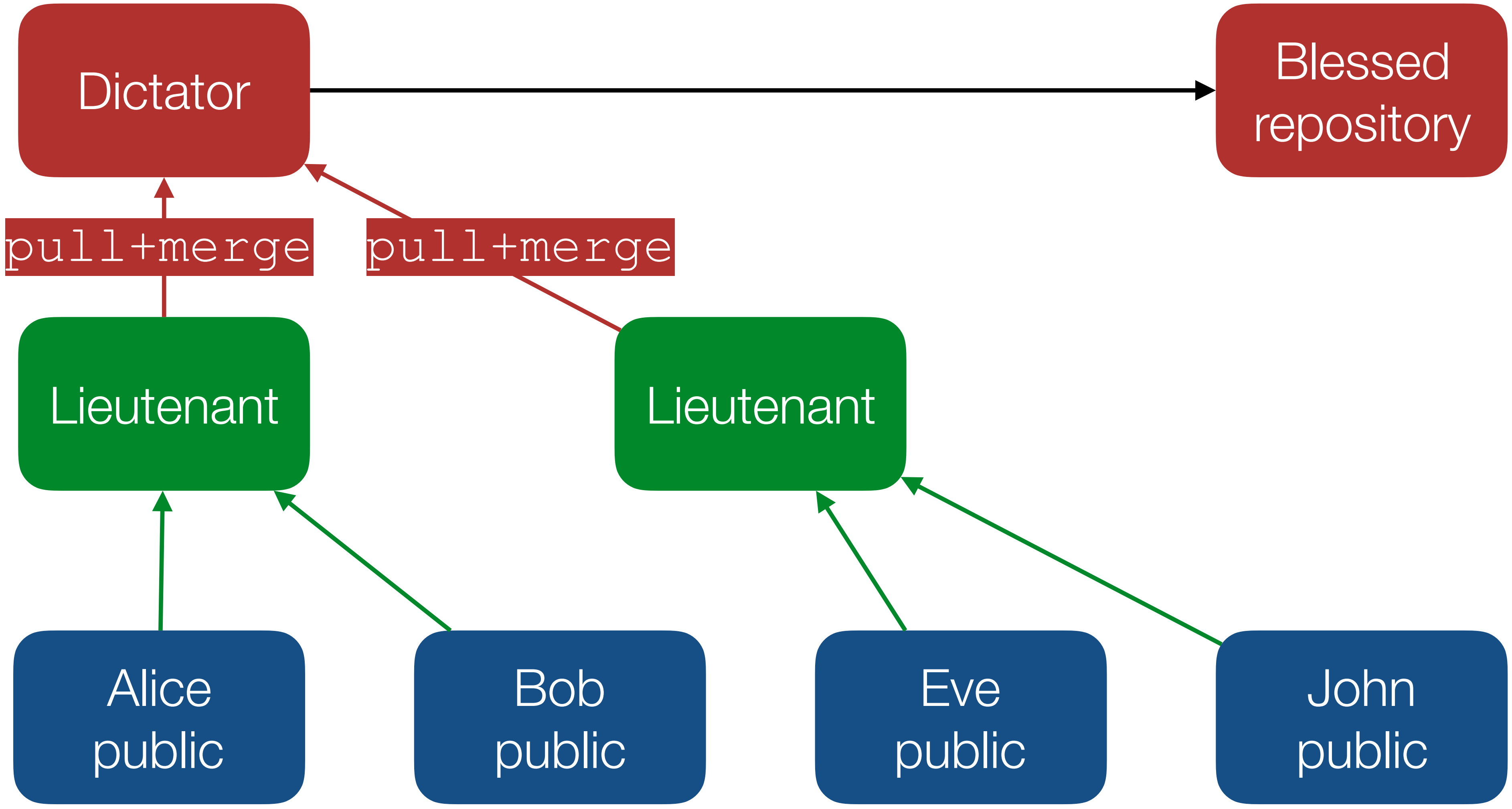
- each lieutenant is usually responsible for one module
- used for big projects (e.g. the Linux kernel, Python)



Dictator / Lieutenants Workflow

Idea: use multiple integration managers (aka lieutenants)

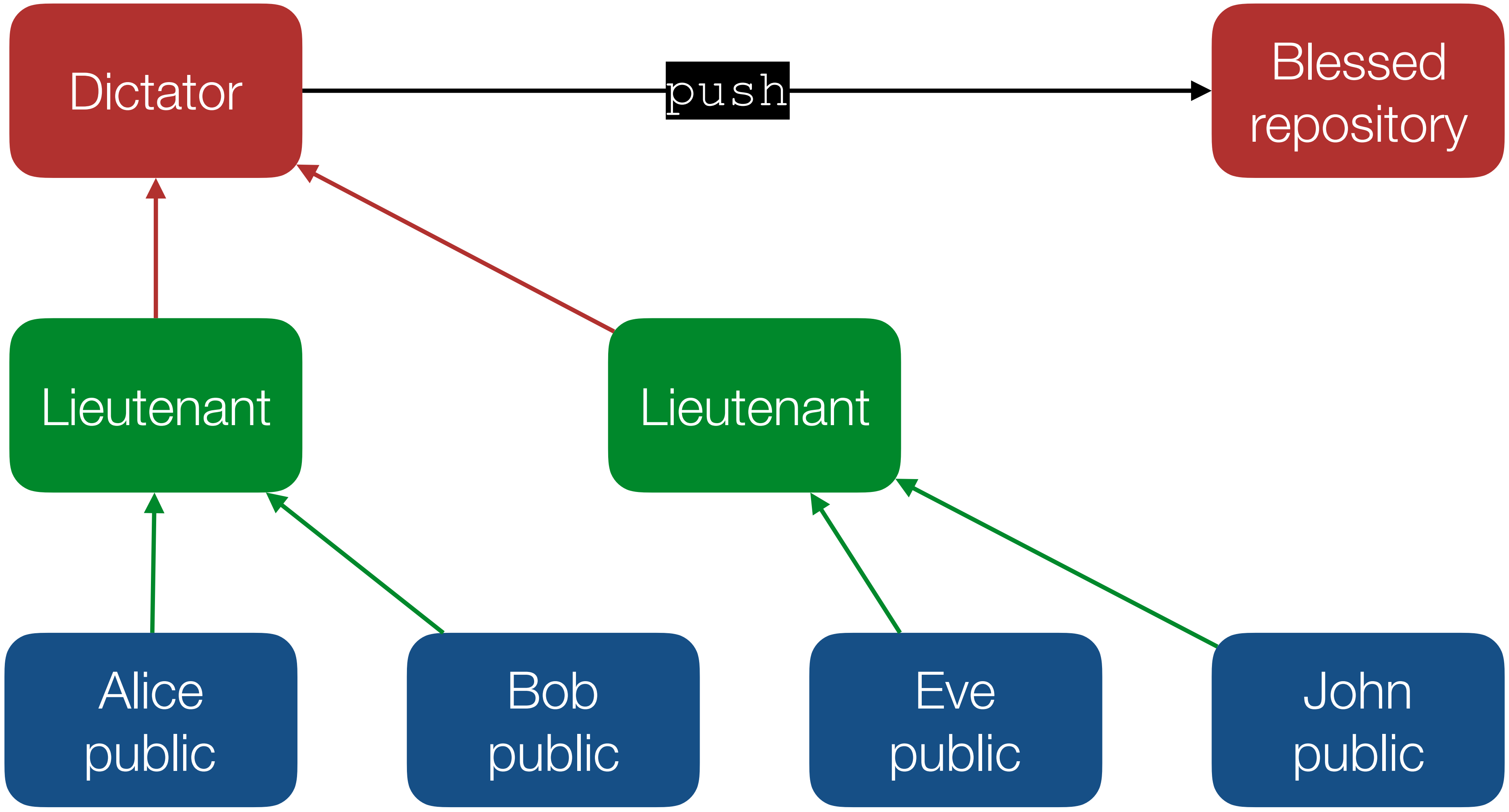
- each lieutenant is usually responsible for one module
- used for big projects (e.g. the Linux kernel, Python)



Dictator / Lieutenants Workflow

Idea: use multiple integration managers (aka lieutenants)

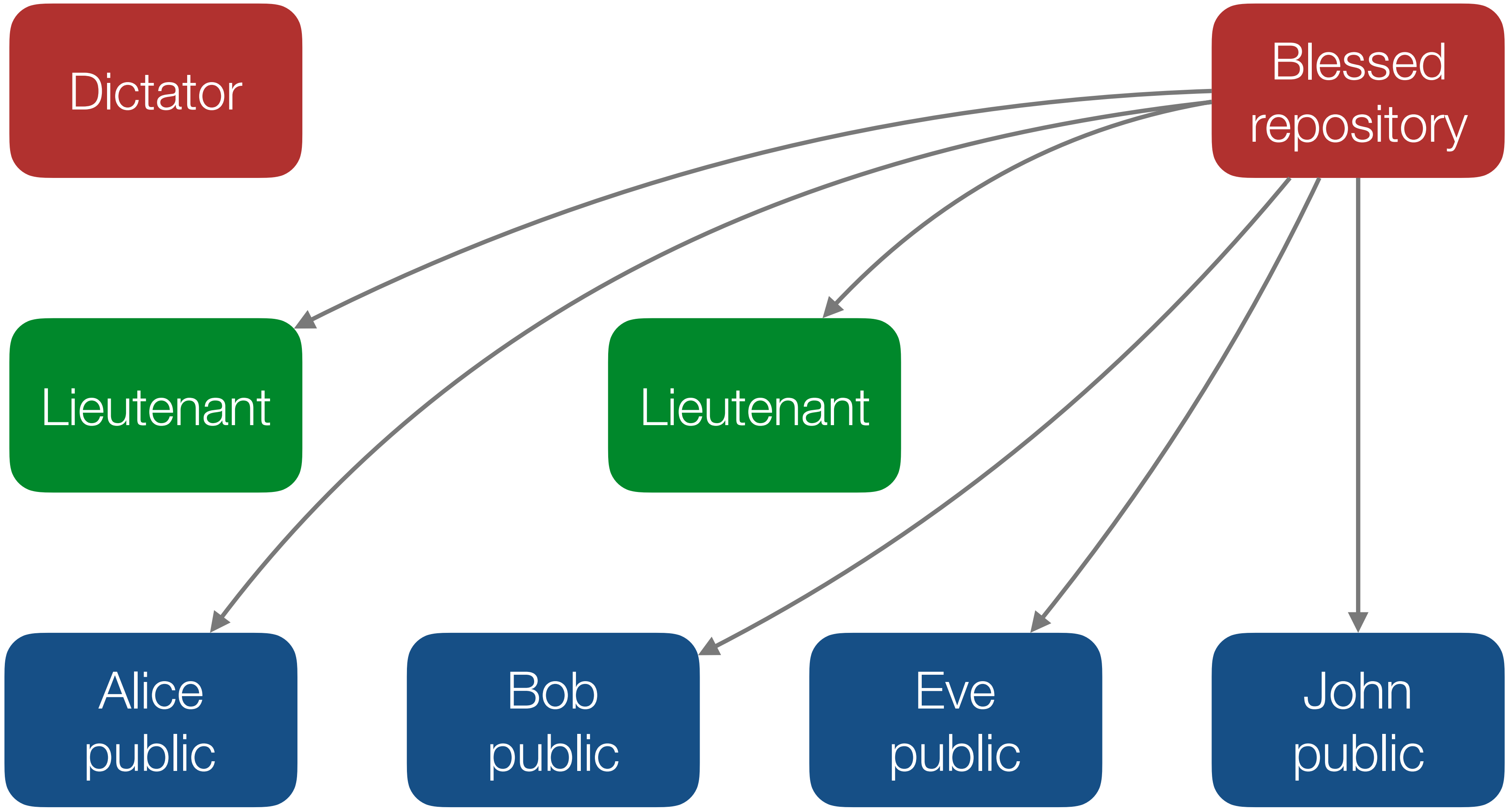
- each lieutenant is usually responsible for one module
- used for big projects (e.g. the Linux kernel, Python)



Dictator / Lieutenants Workflow

Idea: use multiple integration managers (aka lieutenants)

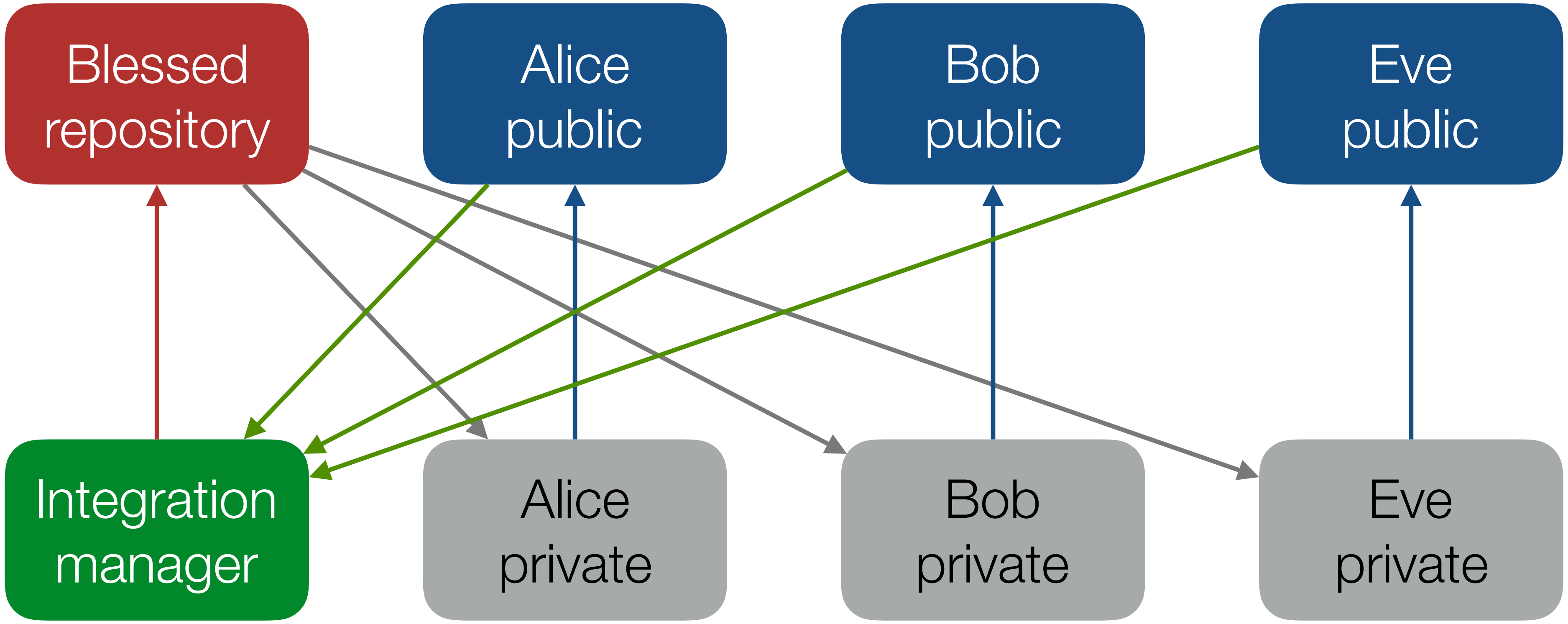
- each lieutenant is usually responsible for one module
- used for big projects (e.g. the Linux kernel, Python)



DO #2: Use the Integration Manager Workflow

Main objectives:

- record changes over time
- recall a specific version later
- **enable collaboration**
- allow nonlinear development



DO #3: Choose a Proper Development Model

Main objectives:

- record changes over time
- recall a specific version later
- enable collaboration
- **allow nonlinear development**

DO #3: Choose a Proper Development Model

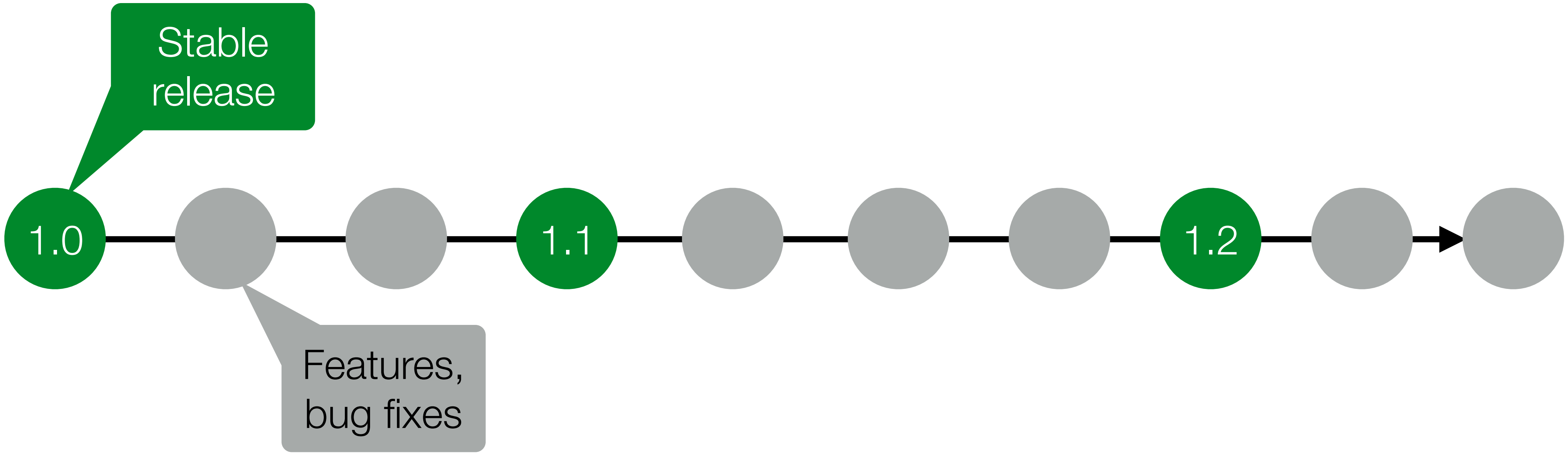
Development models:

- linear
- hotfix branches
- "master" trunk and "develop" branch
- Driessen's branching model

DO #3: Choose a Proper Development Model

Development models:

- **linear**
- hotfix branches
- "master" trunk and "develop" branch
- Driessen's branching model

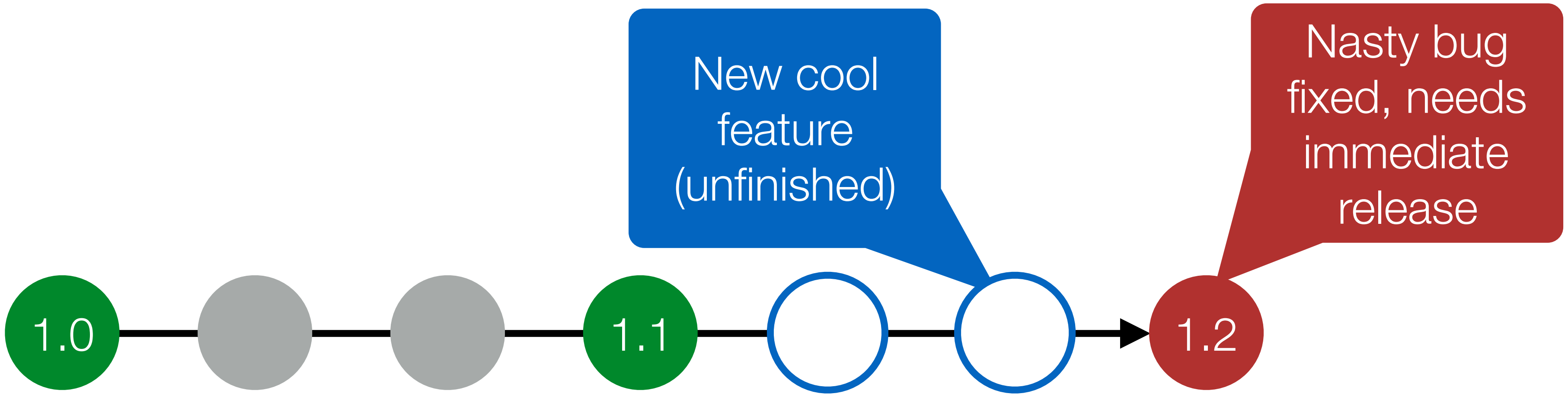


DO #3: Choose a Proper Development Model

Development models:

- **linear**
- hotfix branches
- "master" trunk and "develop" branch
- Driessen's branching model

Problem: release with unfinished features is not desired


New cool feature (unfinished)
Nasty bug fixed, needs immediate release
1.0
1.1
1.2

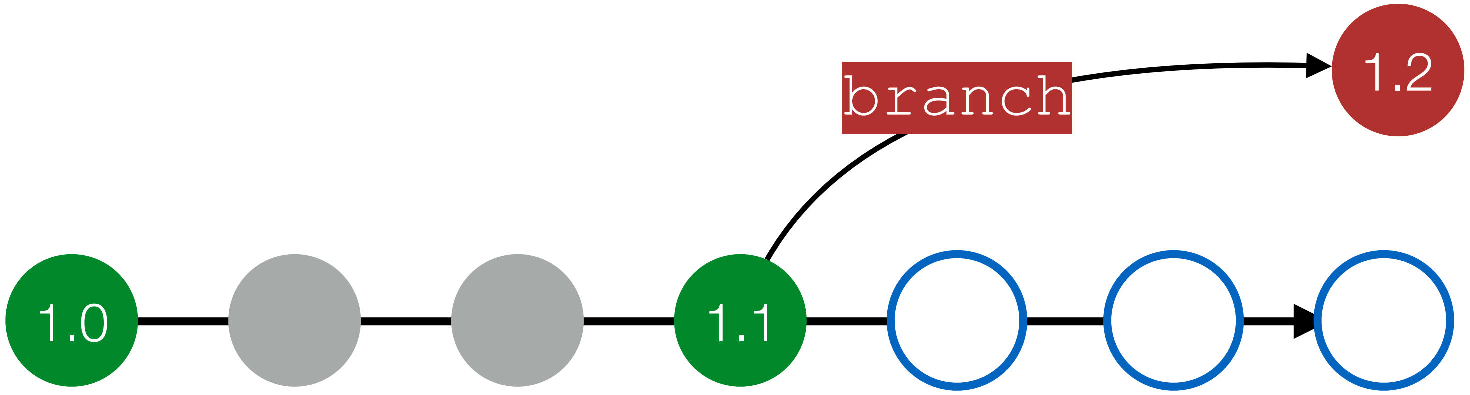
DO #3: Choose a Proper Development Model

Development models:

- ~~linear~~
- **hotfix branches**
- "master" trunk and "develop" branch
- Driessen's branching model

Problem: release with unfinished features is not desired

Solution: branch from latest stable release, only merge with finished features



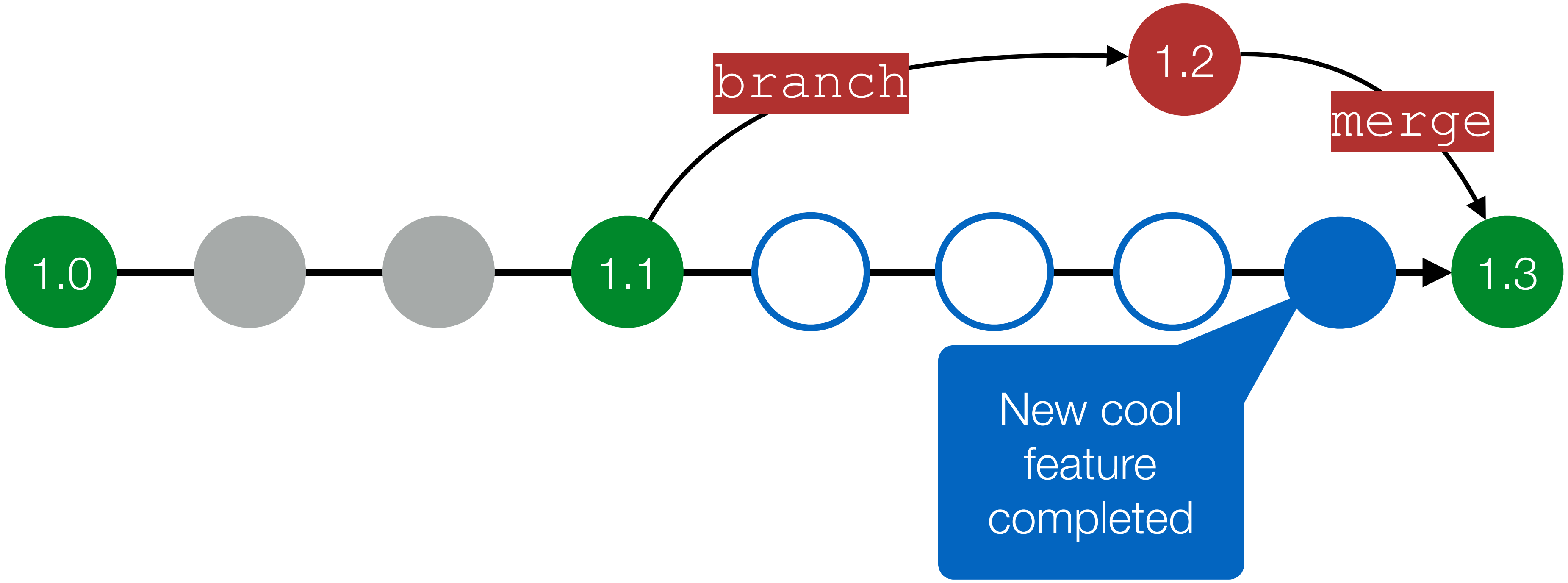
DO #3: Choose a Proper Development Model

Development models:

- ~~linear~~
- **hotfix branches**
- "master" trunk and "develop" branch
- Driessen's branching model

Problem: release with unfinished features is not desired

Solution: branch from latest stable release, only merge with finished features



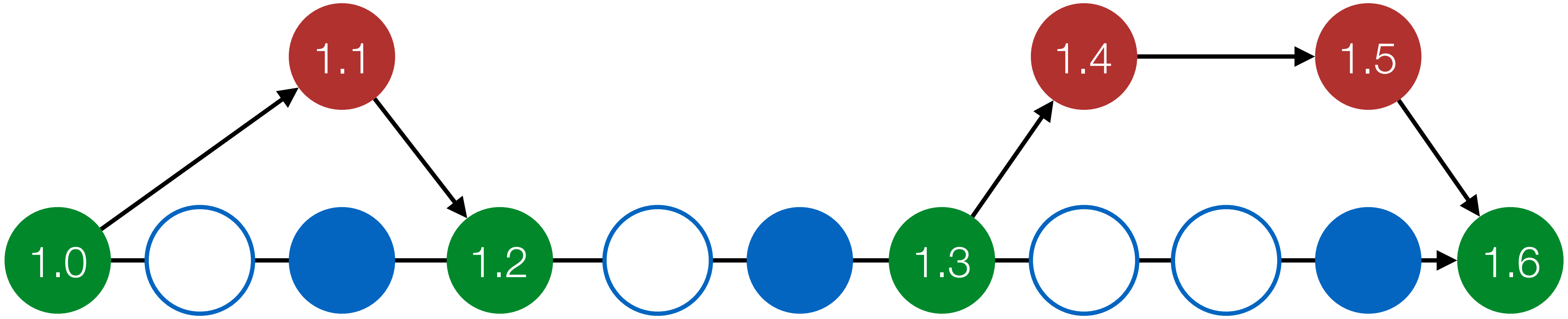
DO #3: Choose a Proper Development Model

Development models:

- ~~linear~~
- **hotfix branches**
- "master" trunk and "develop" branch
- Driessen's branching model

Problem: release with unfinished features is not desired

Solution: branch from latest stable release, only merge with finished features (but then the production code is the second-class citizen)

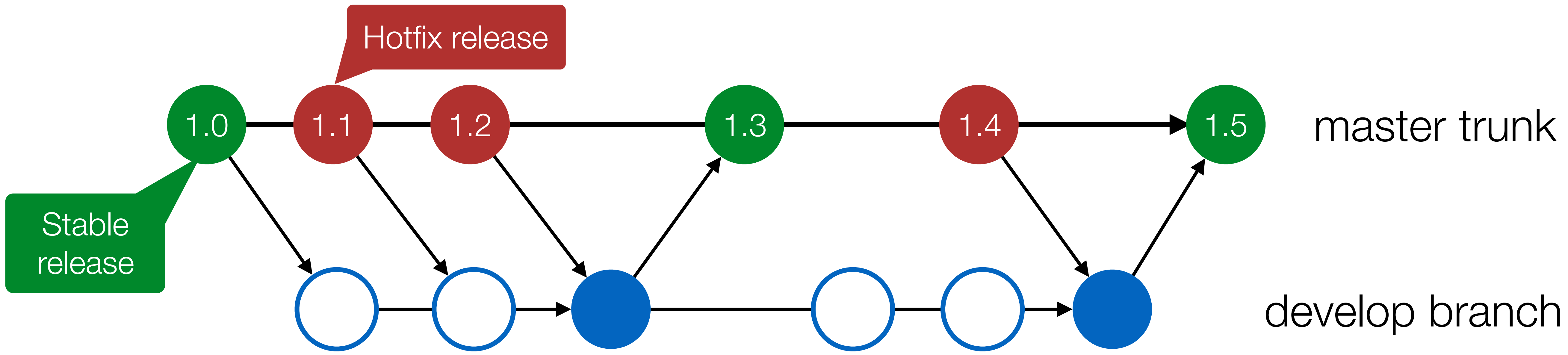


DO #3: Choose a Proper Development Model

Development models:

- ~~linear~~
- hotfix branches
- **"master" trunk and "develop" branch**
- Driessen's branching model

Idea: the trunk always contains production-ready code



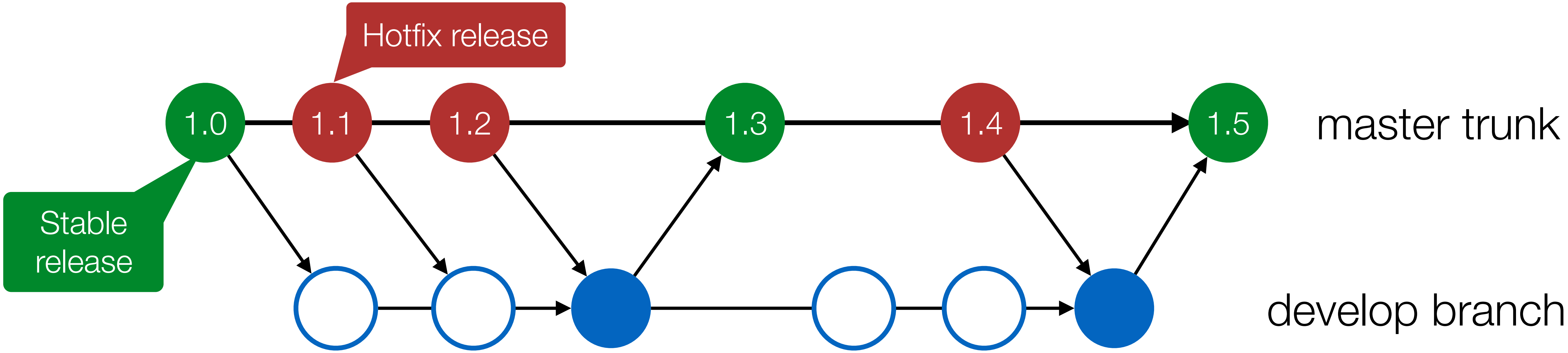
DO #3: Choose a Proper Development Model

Development models:

- ~~linear~~
- hotfix branches
- **"master" trunk and "develop" branch**
- Driessen's branching model

Idea: the trunk always contains production-ready code

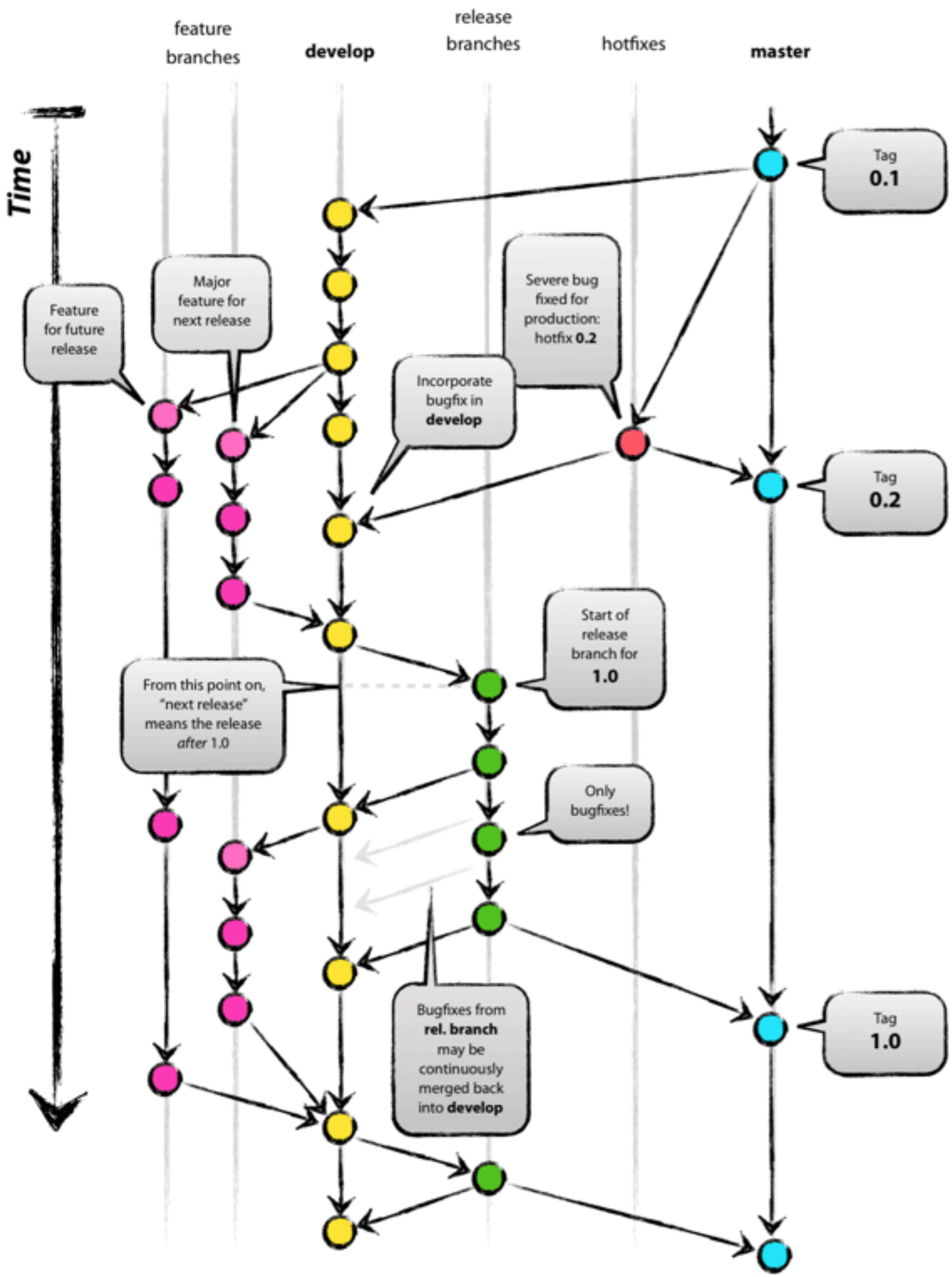
Problem: what if we work on multiple new features in parallel?



DO #3: Choose a Proper Development Model

Development models:

- ~~linear~~
- hotfix branches
- "master" trunk and "develop" branch
- **Driessen's branching model**


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

http://nvie.com/posts/a-successful-git-branching-model/

DO #3: Use a Proper Development model

Development models:

- ~~linear~~
- hotfix branches for projects with infrequent changes
- "master" trunk and "develop" branch for a single developer
- Driessen's branching model for projects with many contributors

Don't be afraid of branches, modern VCS's allow to "close and hide" branches (even unfinished ones)

Many open-source projects will only pull from you if your change is in a feature branch

DOs and DONTs

DONTs:

- do not use archives as a substitute for version control
- never use a centralized VCS (CVS, SVN)
- do not store automatically generated files (PDF, binaries, etc.) and sensitive data (e.g. passwords, logins, SSH keys, etc.)

DOs:

- use a distributed VCS (Mercurial, Git)
- use the "Integration Manager" workflow
  (although the "centralized" workflow works well for papers)
- use Driessen's branching model

Agenda

Today

- version control systems and collaborative development
- **Mercurial, Git, Bitbucket, GitHub**
- providing support

Tomorrow

- unit testing
- documentation
- dissemination

Mercurial vs Git

Mercurial vs Git

Both written in April 2005 after the big BitKeeper/Torvalds split

Both are:

- distributed version control systems
- fast (Git a tad faster)
- relatively easy to use once (Mercurial more so)
- multi-platform (Mercurial "more native" on Windows)

Selecting one is mostly a matter of taste

- Git is more popular mainly due to GitHub

Mercurial vs Git

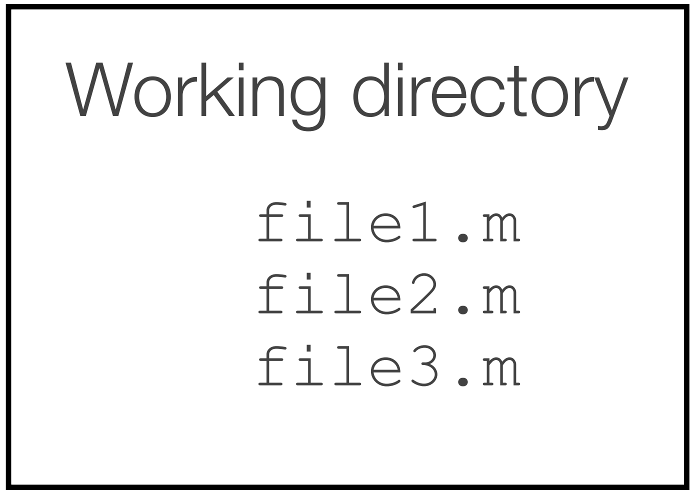
mercurial

Bitbucket
Dovecot
FuseSMB
Growl
MPT
mutt
NetBeans
nginx
Octave
OpenJDK
OpenOffice
OpenSolaris
Python
RabbitMQ
rpm.org
Sphinx

git

Android
Debian tools
Drupal
FFmpeg
GCC
GNOME
jQuery
Julia
KVM
Linux kernel
Maemo
OLPC
Perl
Samba
VLC
Yum

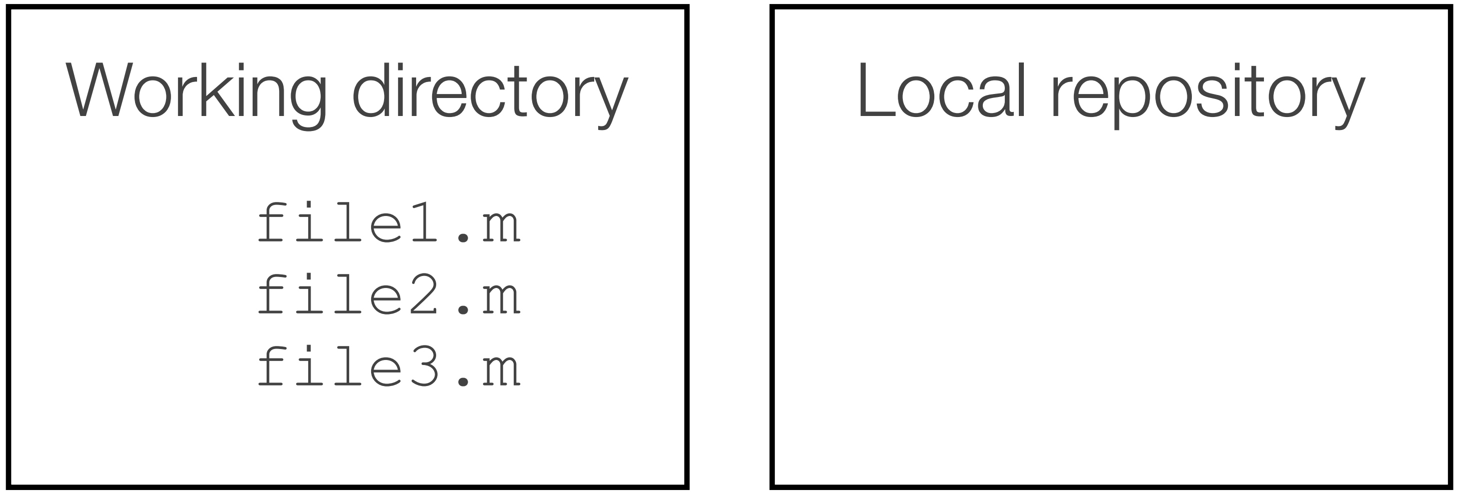
Mercurial: Basic Commands



Working directory

```
file1.m
file2.m
file3.m
```

Mercurial: Basic Commands



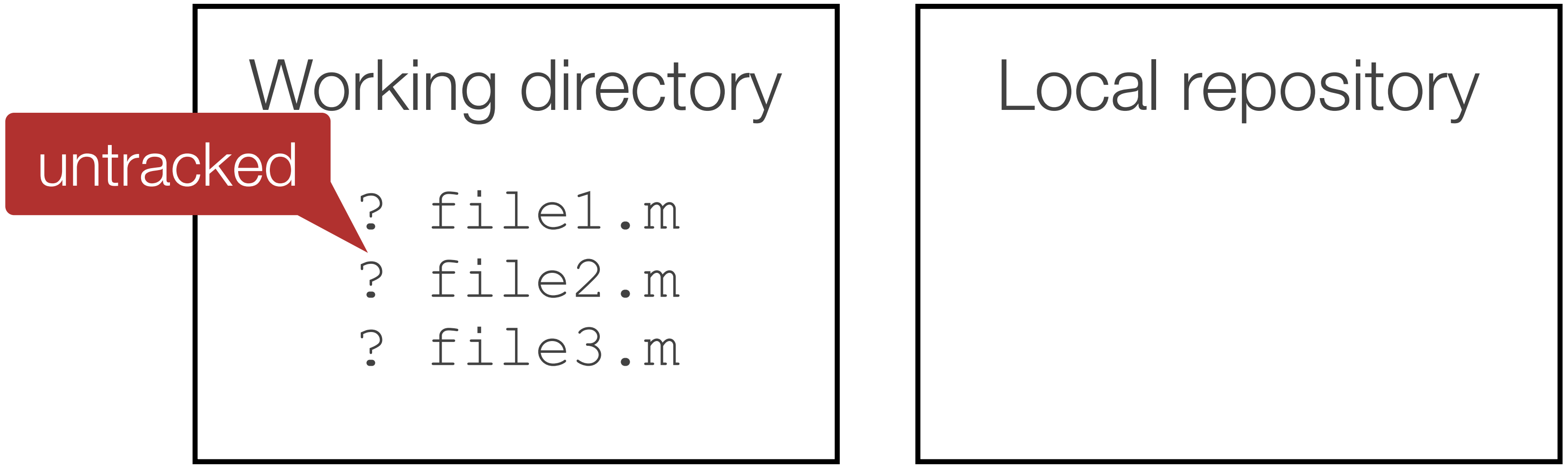
Working directory

file1.m
file2.m
file3.m

Local repository

Initialize a new local repository: `hg init`

Mercurial: Basic Commands



Working directory

untracked

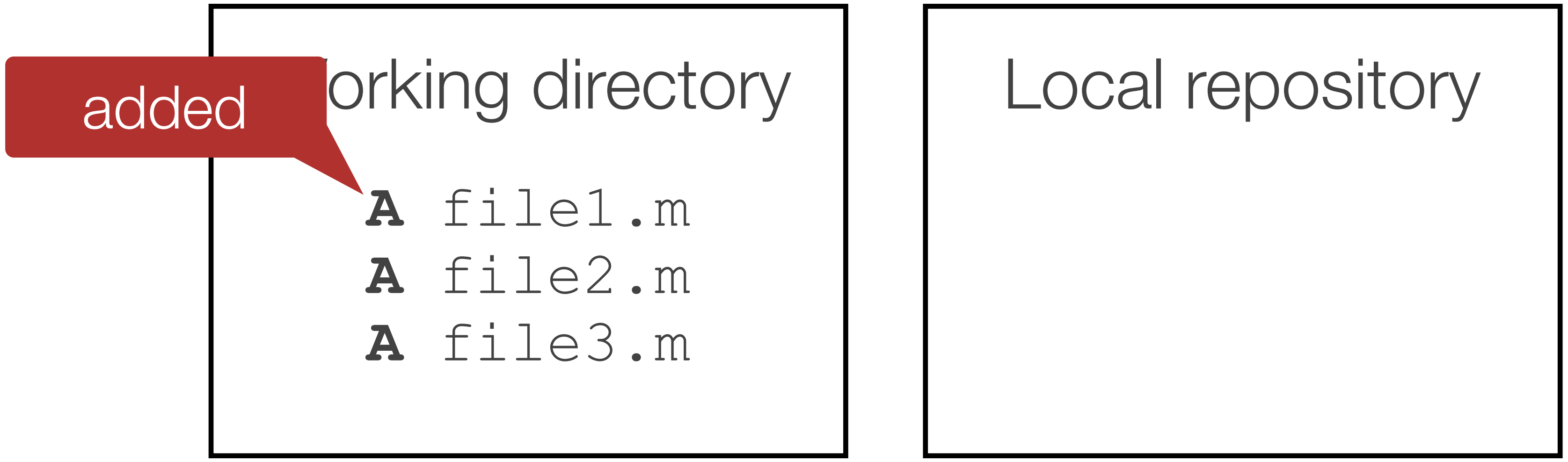
```
? file1.m
? file2.m
? file3.m
```

Local repository

Initialize a new local repository: `hg init`

Check status of files: `hg status`

Mercurial: Basic Commands



added

orking directory

```
A file1.m
A file2.m
A file3.m
```

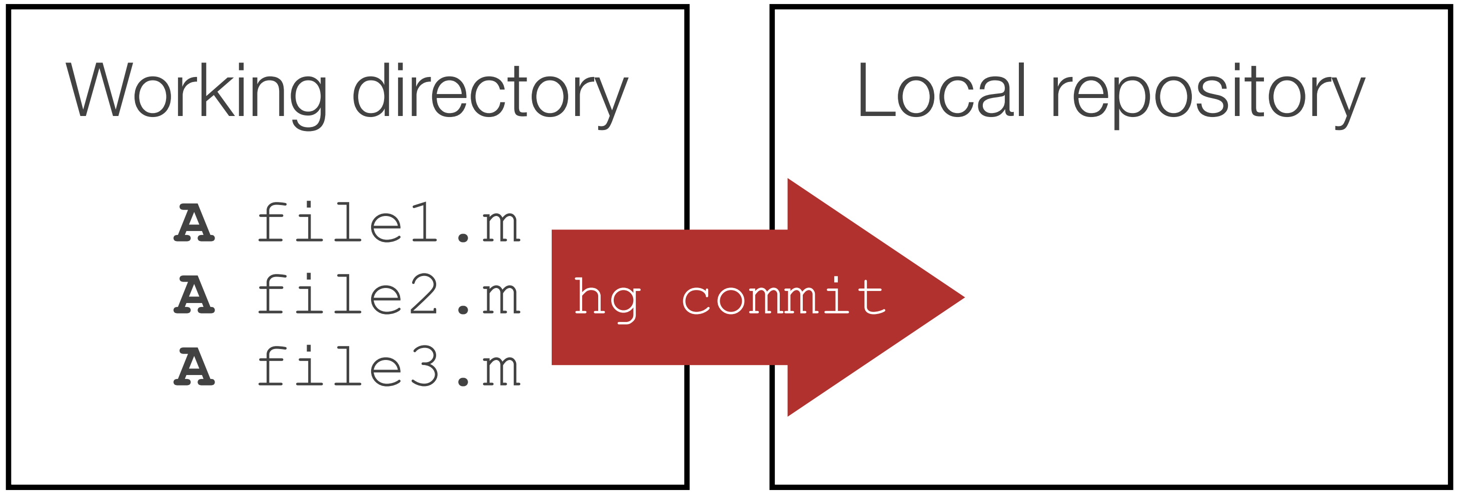
Local repository

Initialize a new local repository: `hg init`

Check status of files: `hg status`

Start tracking all files: `hg add`

Mercurial: Basic Commands



Working directory

```
A file1.m
A file2.m
A file3.m
```

hg commit

Local repository

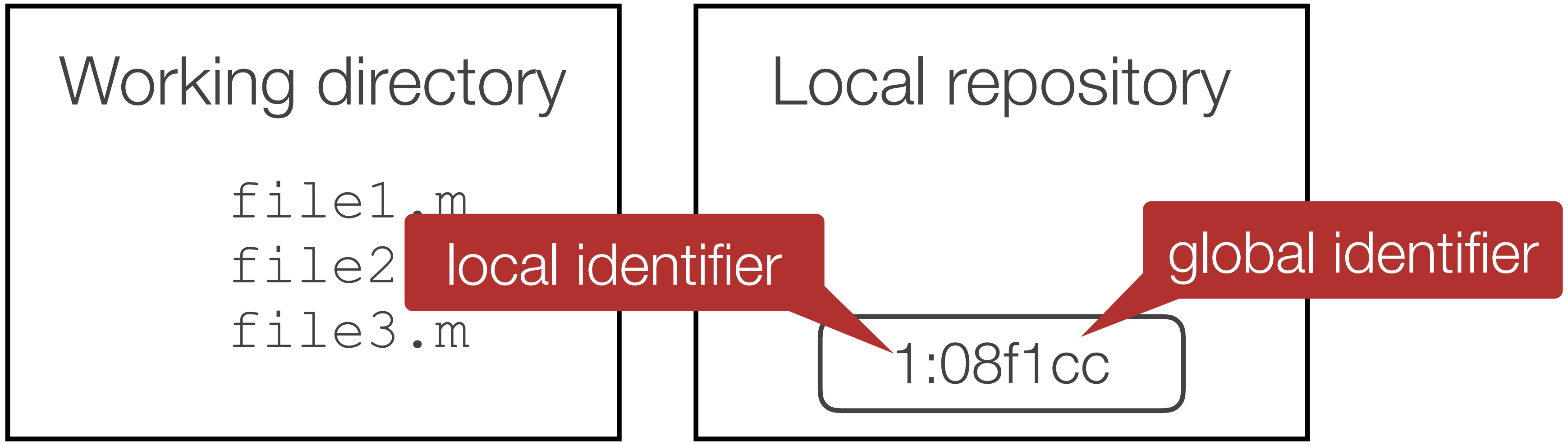
Initialize a new local repository: `hg init`

Check status of files: `hg status`

Start tracking all files: `hg add`

Commit the added files to the repository: `hg commit -m "message"`

Mercurial: Basic Commands



Working directory

file1.m
file2
file3.m

local identifier

Local repository

global identifier

1:08f1cc

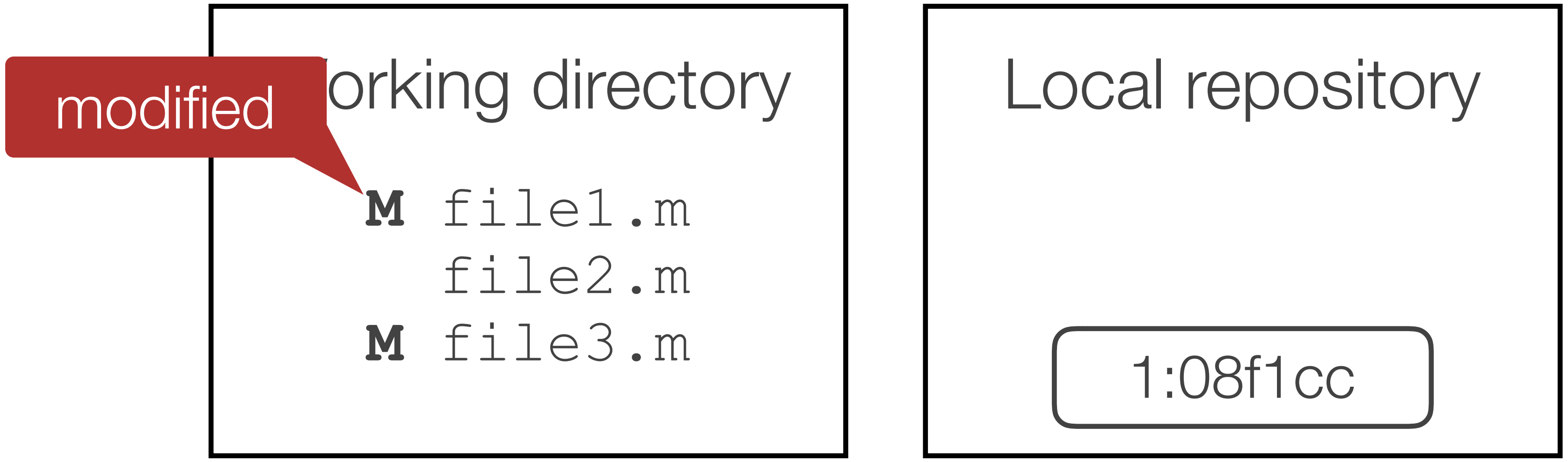
Initialize a new local repository: `hg init`

Check status of files: `hg status`

Start tracking all files: `hg add`

Commit the added files to the repository: `hg commit -m "message"`

Mercurial: Basic Commands



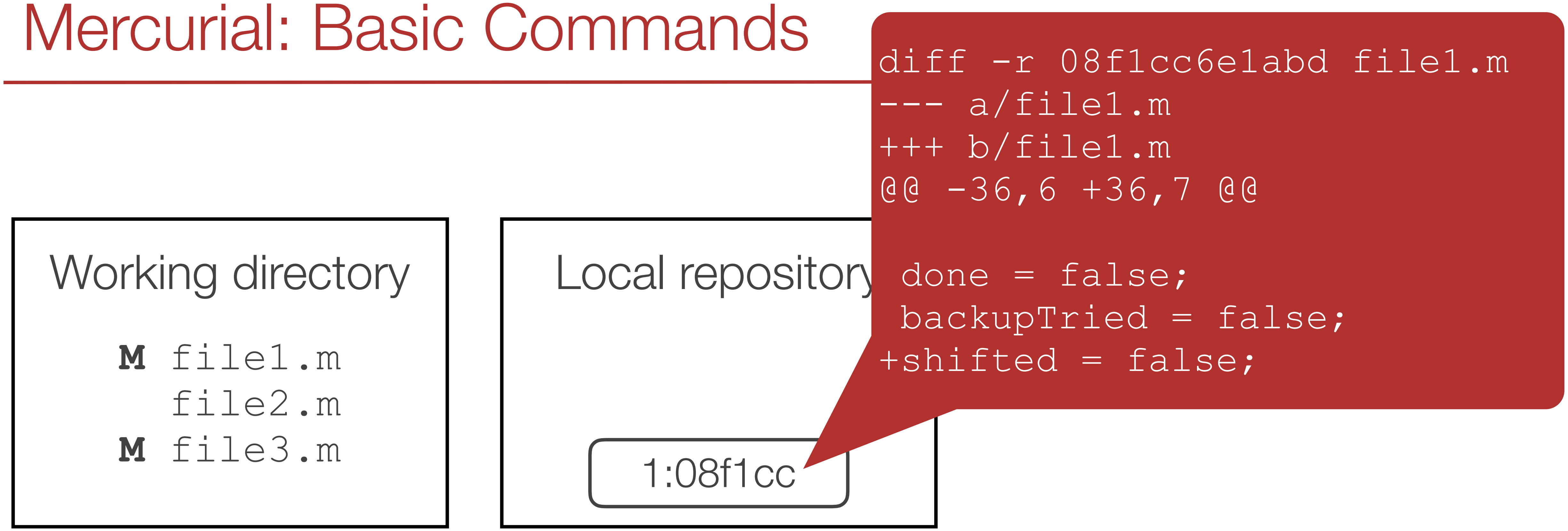
Initialize a new local repository: `hg init`

Check status of files: `hg status`

Start tracking all files: `hg add`

Commit the added files to the repository: `hg commit -m "message"`

Mercurial: Basic Commands



Working directory

```
M file1.m
  file2.m
M file3.m
```

Local repository

```
diff -r 08f1cc6e1abd file1.m
--- a/file1.m
+++ b/file1.m
@@ -36,6 +36,7 @@

 done = false;
 backupTried = false;
+shifted = false;
```

Initialize a new local repository: `hg init`

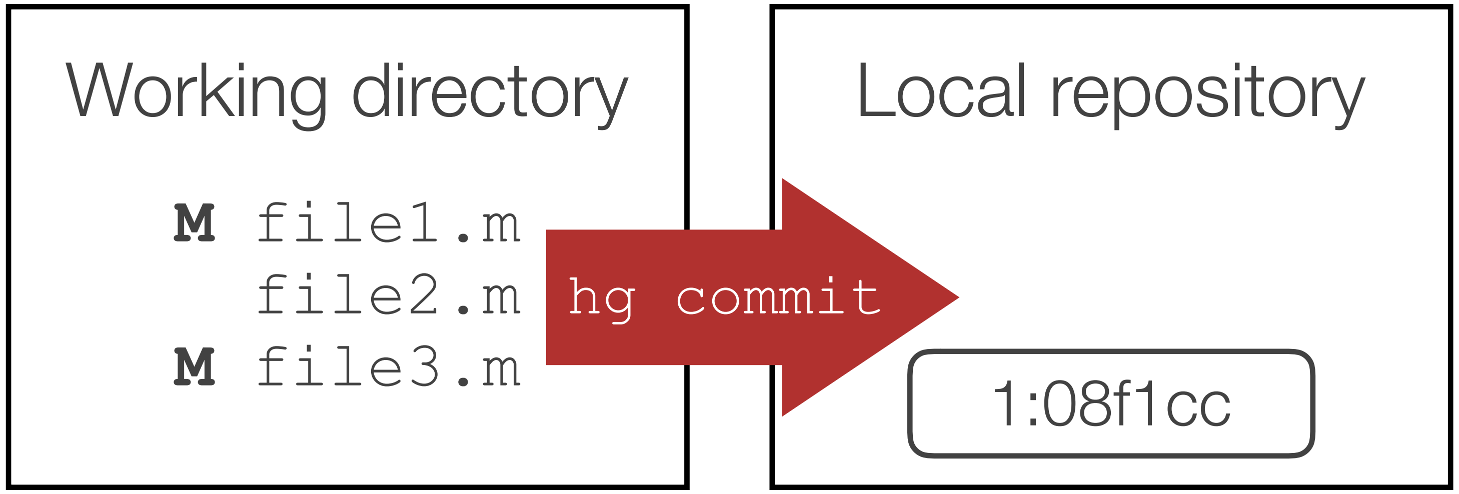
Check status of files: `hg status`

Start tracking all files: `hg add`

Commit the added files to the repository: `hg commit -m "message"`

View differences to repository: `hg diff`

Mercurial: Basic Commands



Initialize a new local repository: `hg init`

Check status of files: `hg status`

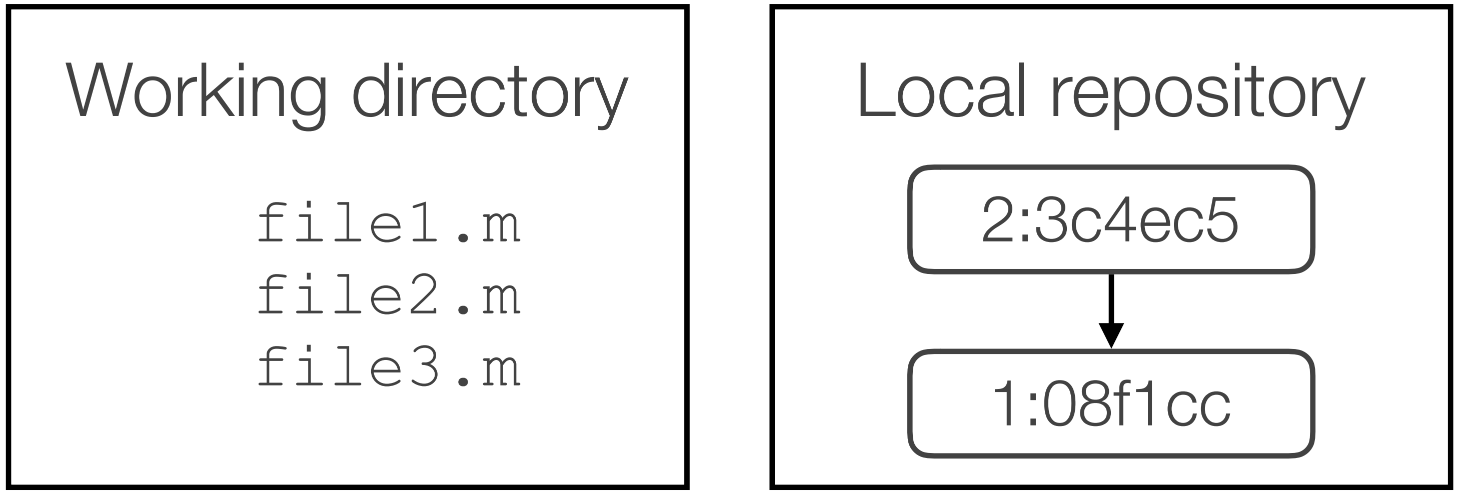
Start tracking all files: `hg add`

Commit the added files to the repository: `hg commit -m "message"`

View differences to repository: `hg diff`

Commit the modified files to the repository: `hg commit -m "message"`

Mercurial: Basic Commands



Working directory

file1.m
file2.m
file3.m

Local repository

2:3c4ec5

1:08f1cc

Initialize a new local repository: `hg init`

Check status of files: `hg status`

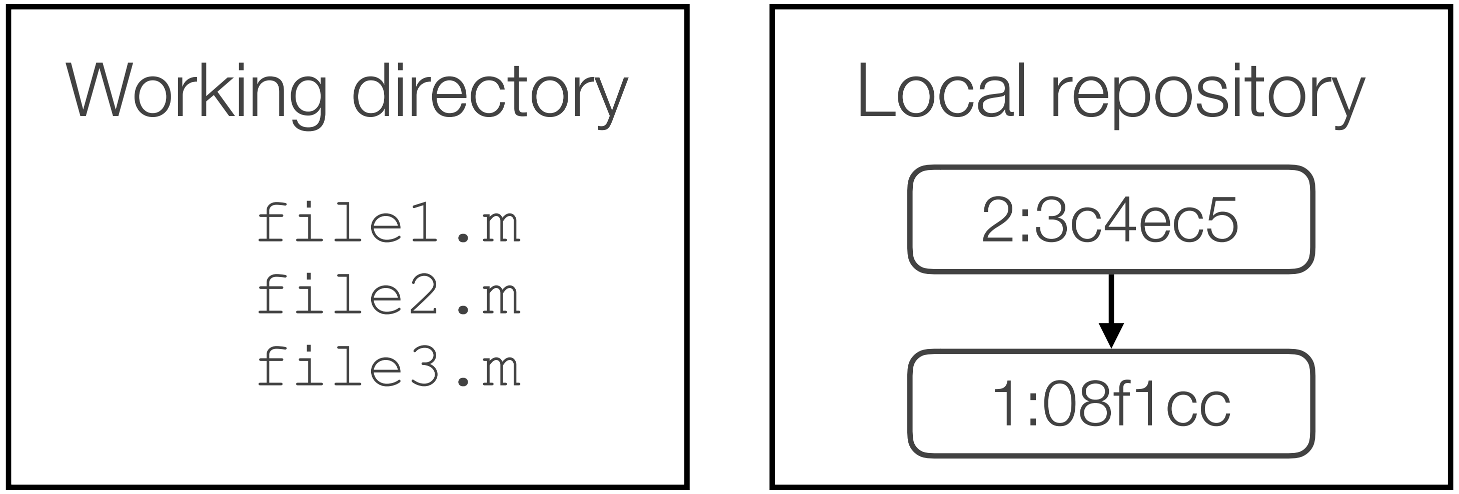
Start tracking all files: `hg add`

Commit the added files to the repository: `hg commit -m "message"`

View differences to repository: `hg diff`

Commit the modified files to the repository: `hg commit -m "message"`

Mercurial: Basic Commands



Working directory

```
file1.m
file2.m
file3.m
```

Local repository

2:3c4ec5

1:08f1cc

Initialize a new local repository: `hg init`

Check status of files: `hg status`

Start tracking all files: `hg add`

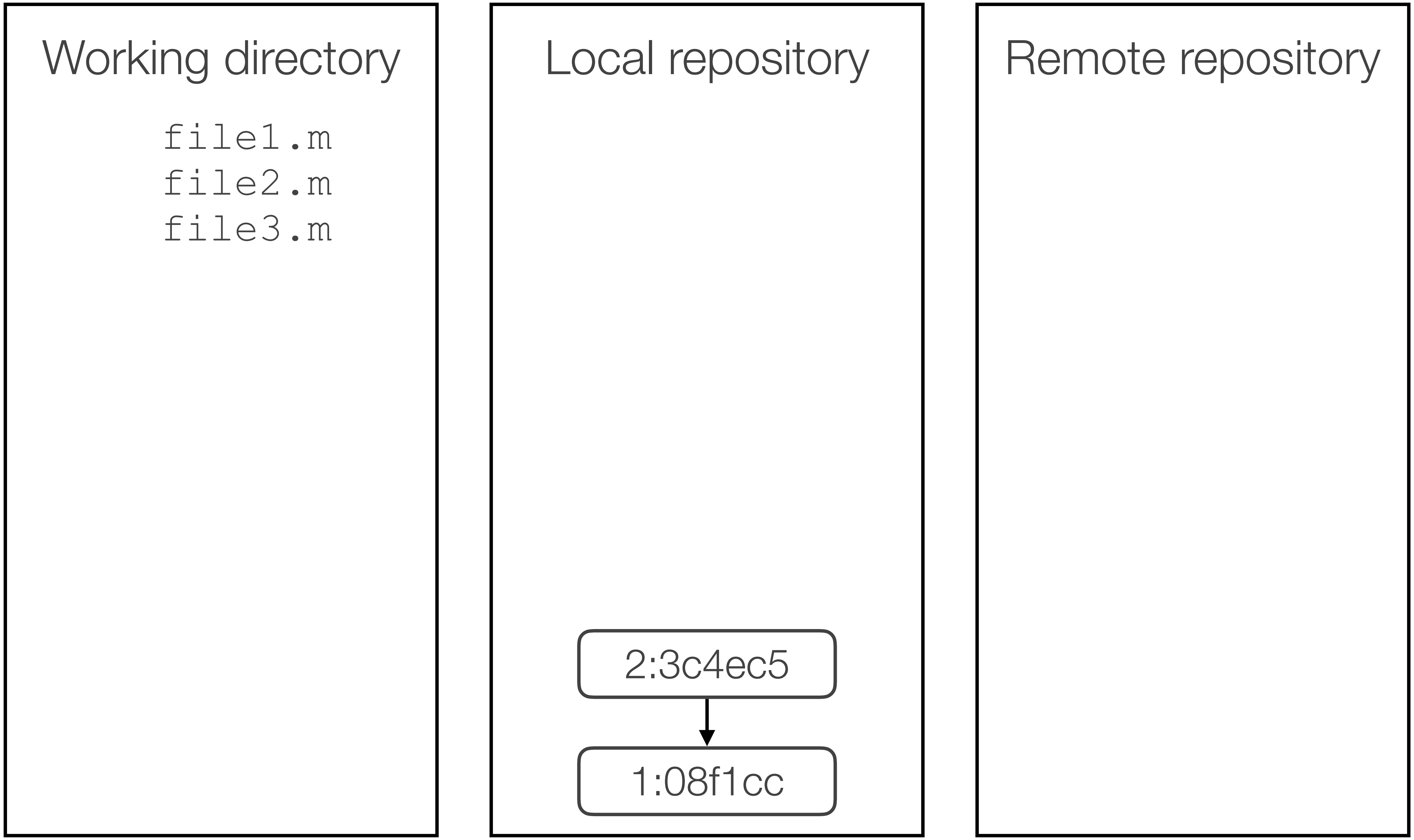
Commit the added files to the repository: `hg commit -m "message"`

View differences to repository: `hg diff`

Commit the modified files to the repository: `hg commit -m "message"`

View the log: `hg log`

Mercurial: Basic Commands



Working directory

file1.m
file2.m
file3.m

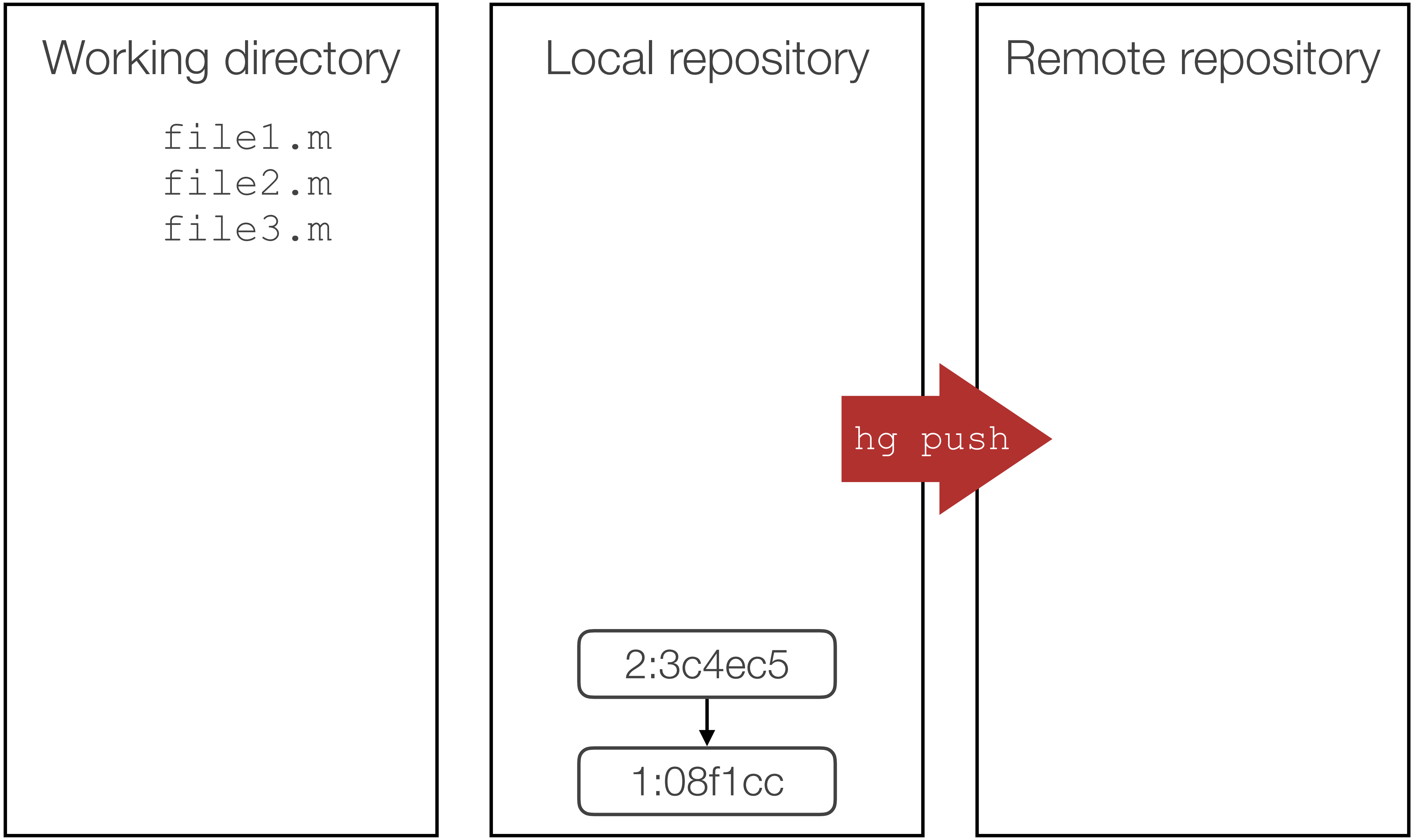
Local repository

2:3c4ec5

1:08f1cc

Remote repository

Mercurial: Basic Commands



Working directory

file1.m
file2.m
file3.m

Local repository

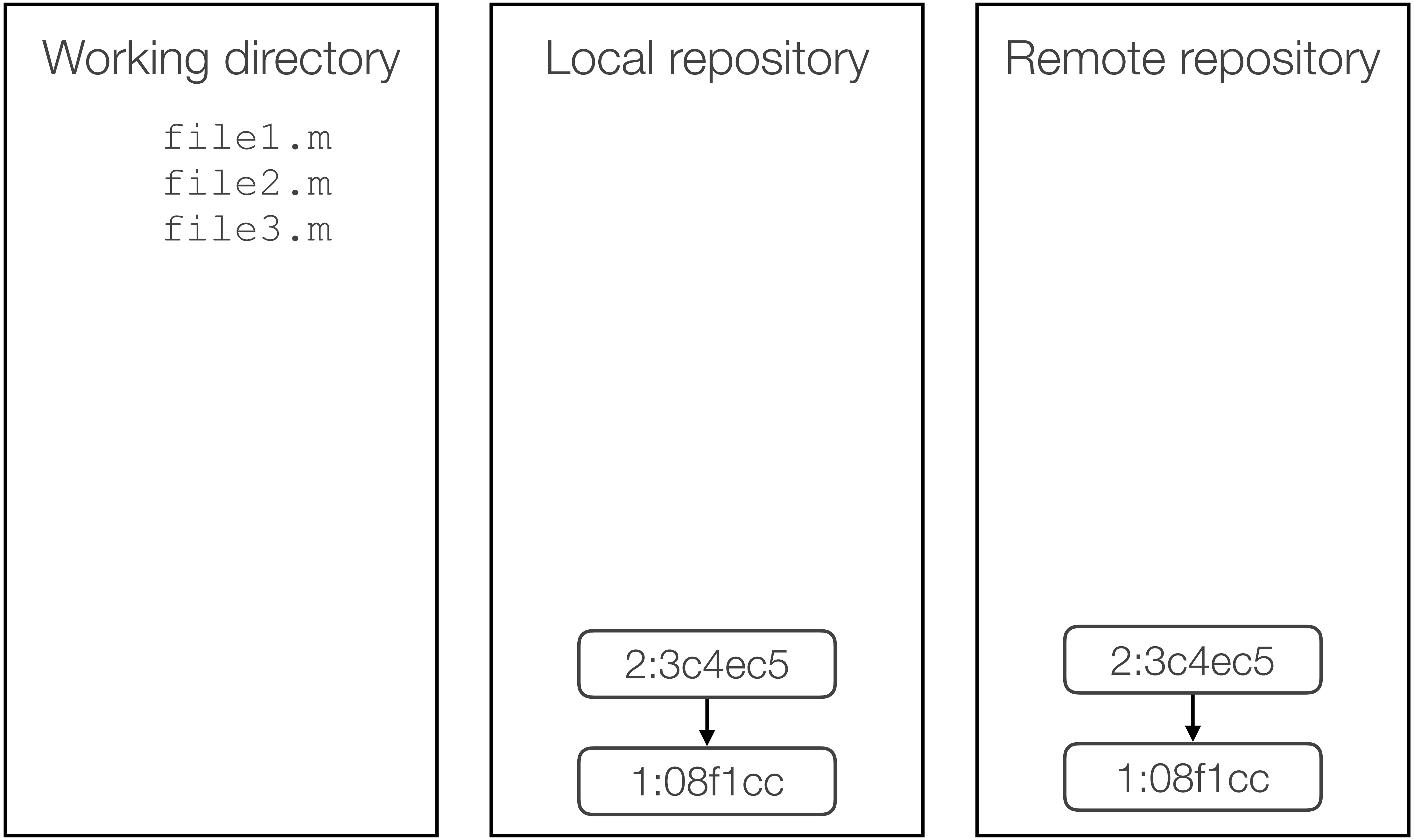
2:3c4ec5

1:08f1cc

`hg push`

Remote repository

Mercurial: Basic Commands



Working directory

file1.m
file2.m
file3.m

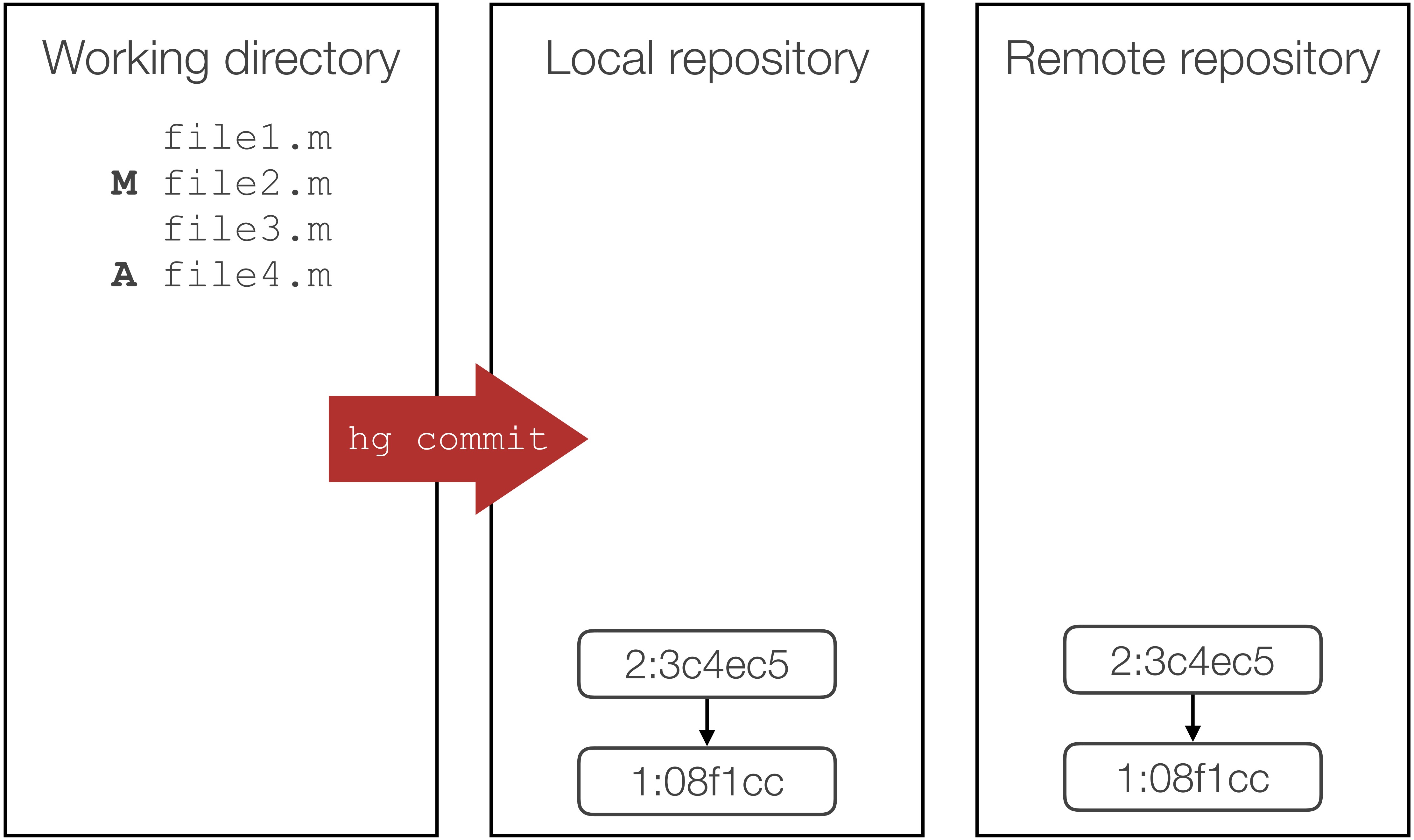
Local repository

2:3c4ec5
1:08f1cc

Remote repository

2:3c4ec5
1:08f1cc

Mercurial: Basic Commands



Working directory

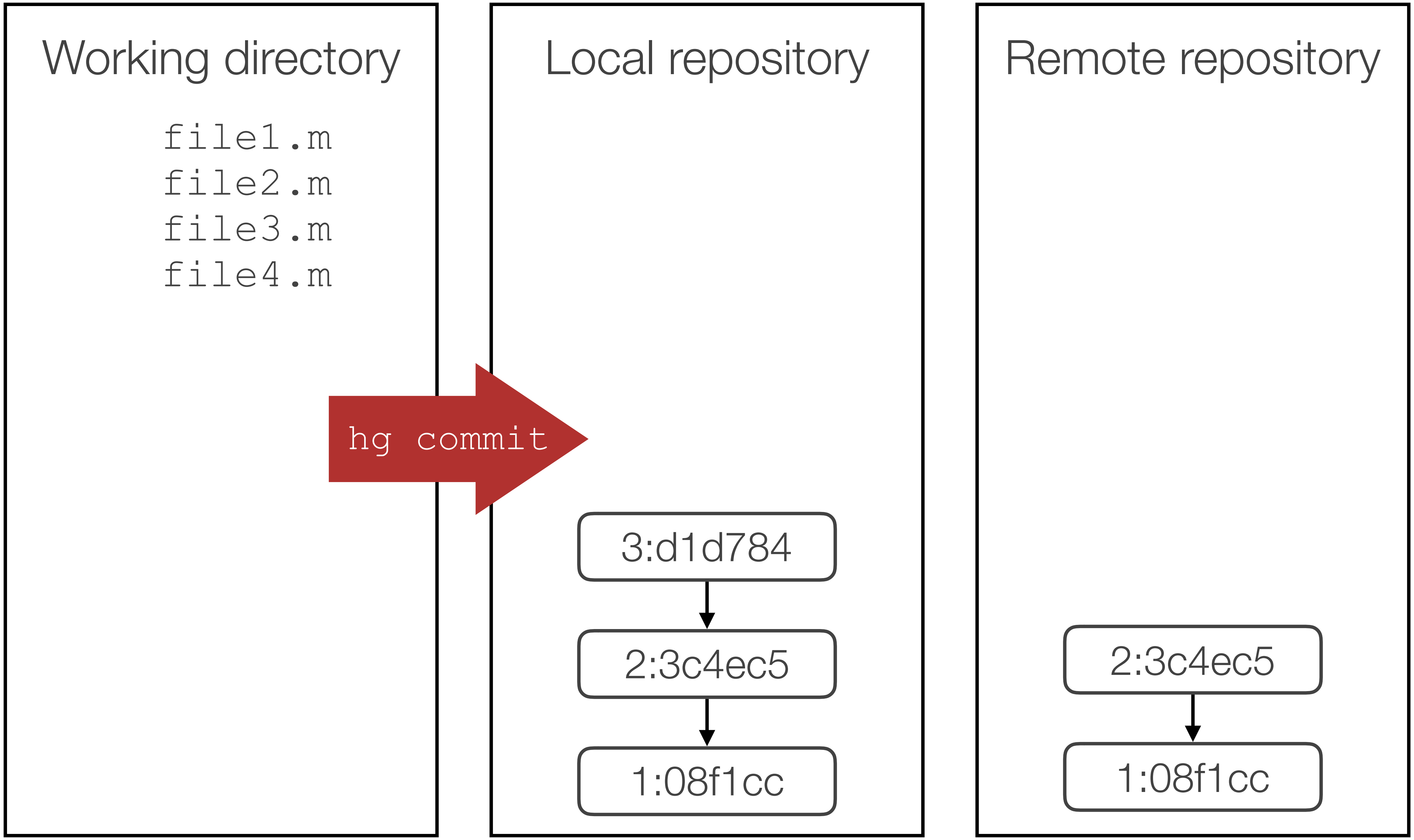
```
  file1.m
M file2.m
  file3.m
A file4.m
```

`hg commit`

Local repository

Remote repository

Mercurial: Basic Commands



Working directory

file1.m
file2.m
file3.m
file4.m

```
hg commit
```

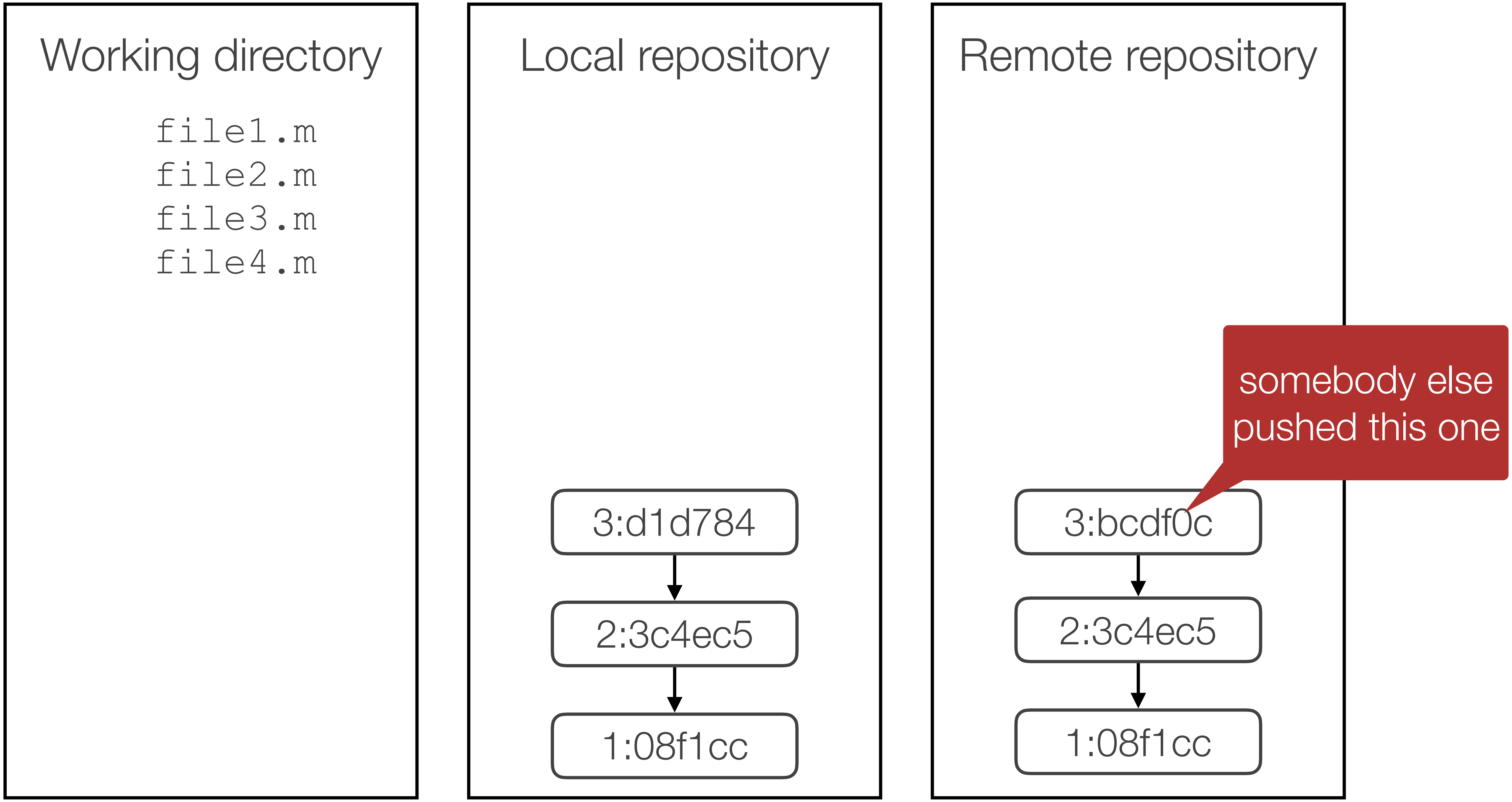
Local repository

3:d1d784
2:3c4ec5
1:08f1cc

Remote repository

2:3c4ec5
1:08f1cc

Mercurial: Basic Commands



Working directory

```
file1.m
file2.m
file3.m
file4.m
```

Local repository

3:d1d784

2:3c4ec5

1:08f1cc

Remote repository

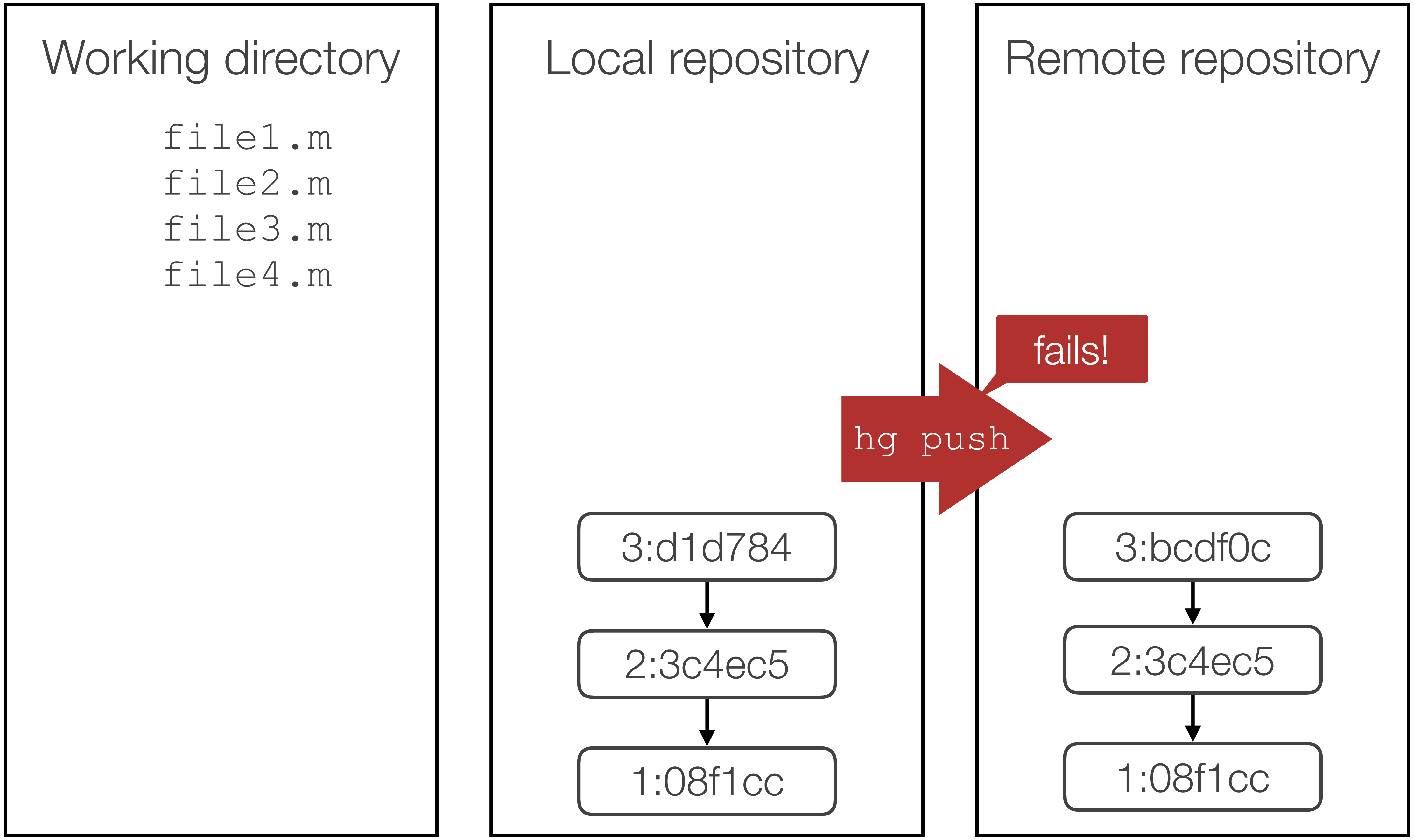
3:bcdf0c

2:3c4ec5

1:08f1cc

somebody else pushed this one

Mercurial: Basic Commands



Working directory

```
file1.m
file2.m
file3.m
file4.m
```

Local repository

3:d1d784

2:3c4ec5

1:08f1cc

`hg push`

fails!

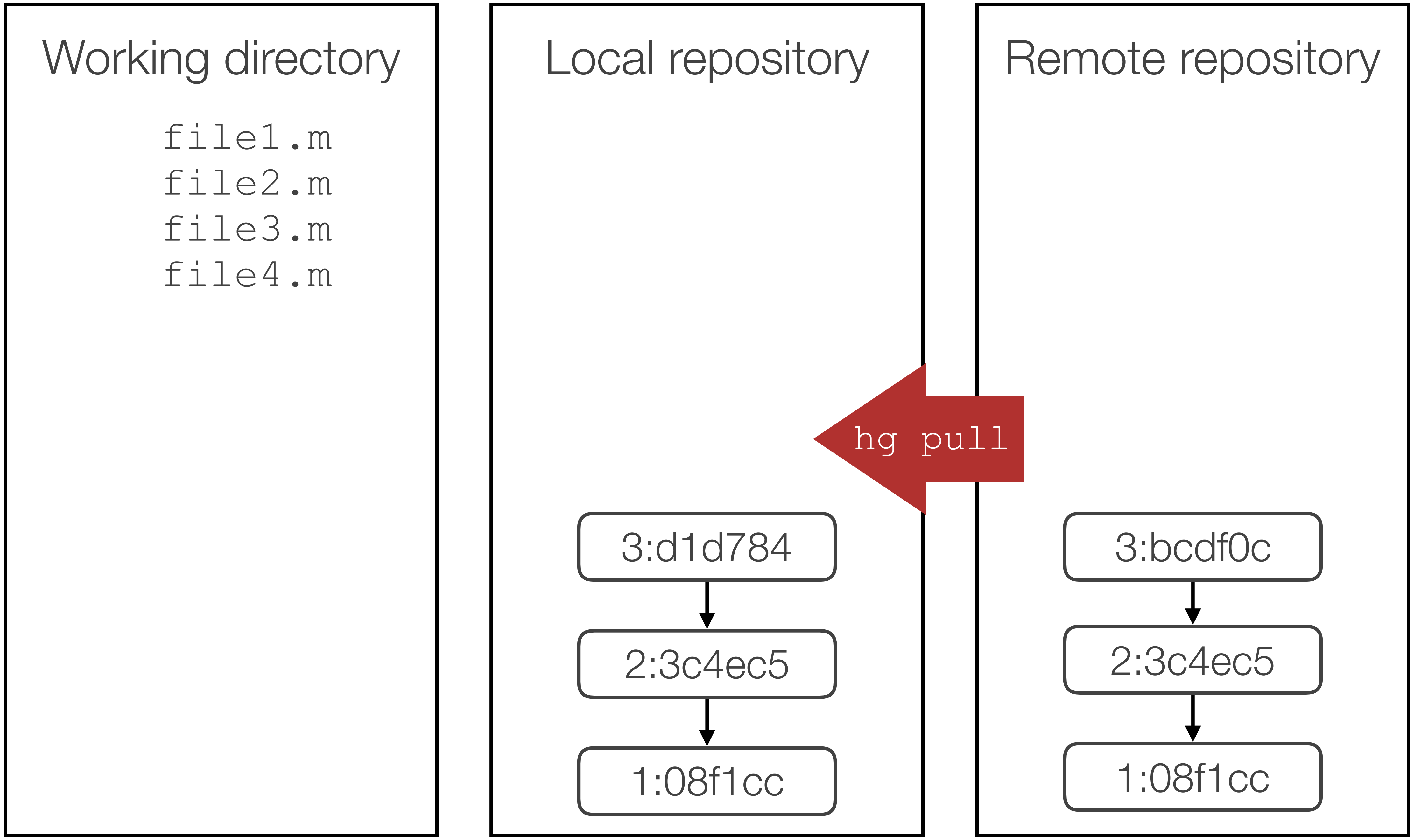
Remote repository

3:bcdf0c

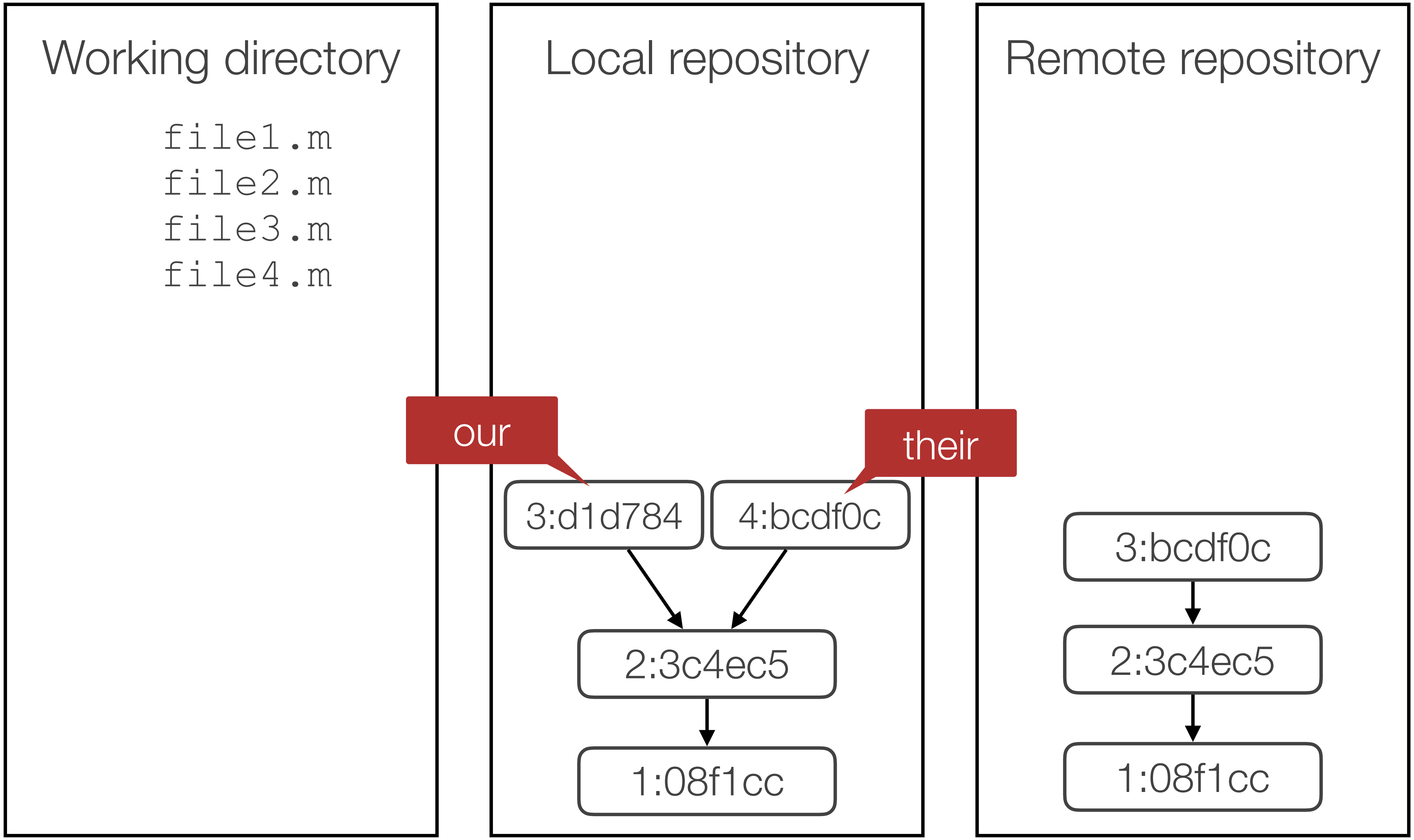
2:3c4ec5

1:08f1cc

Mercurial: Basic Commands



Mercurial: Basic Commands



Working directory

```
file1.m
file2.m
file3.m
file4.m
```

Local repository

our

their

3:d1d784

4:bcdf0c

2:3c4ec5

1:08f1cc

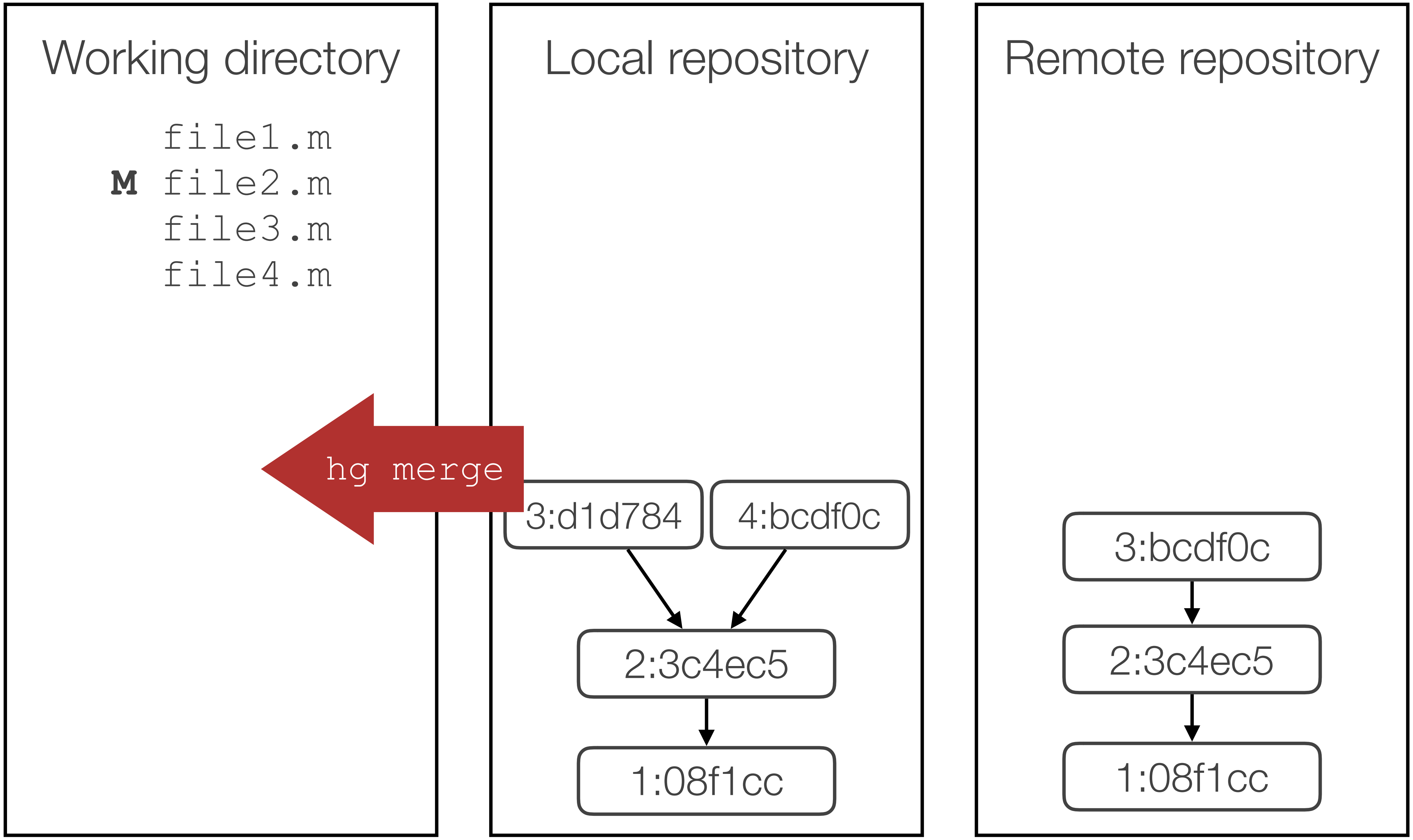
Remote repository

3:bcdf0c

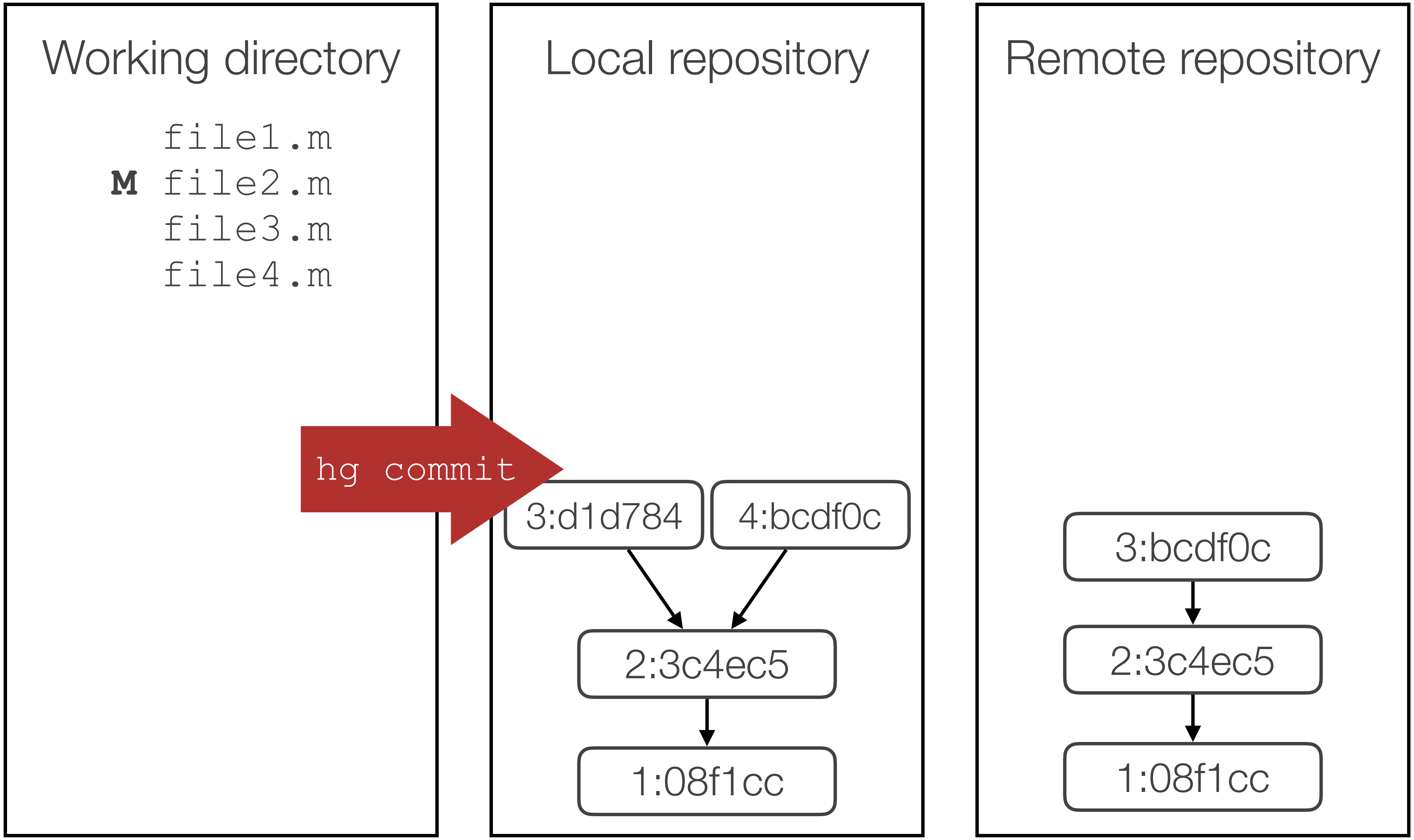
2:3c4ec5

1:08f1cc

Mercurial: Basic Commands



Mercurial: Basic Commands



Working directory

```
   file1.m
M  file2.m
   file3.m
   file4.m
```

`hg commit`

Local repository

3:d1d784

4:bcdf0c

2:3c4ec5

1:08f1cc

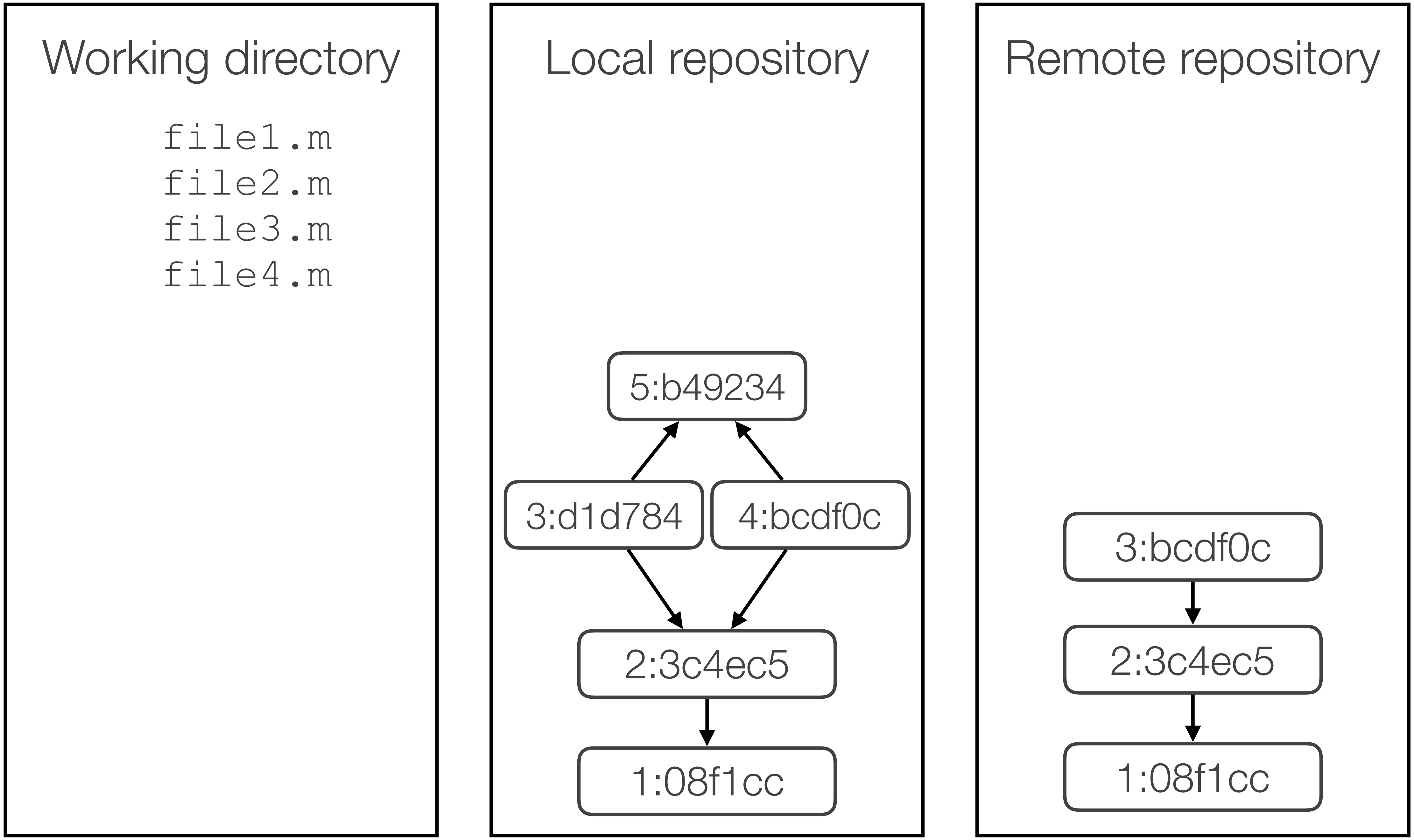
Remote repository

3:bcdf0c

2:3c4ec5

1:08f1cc

Mercurial: Basic Commands



Working directory

file1.m
file2.m
file3.m
file4.m

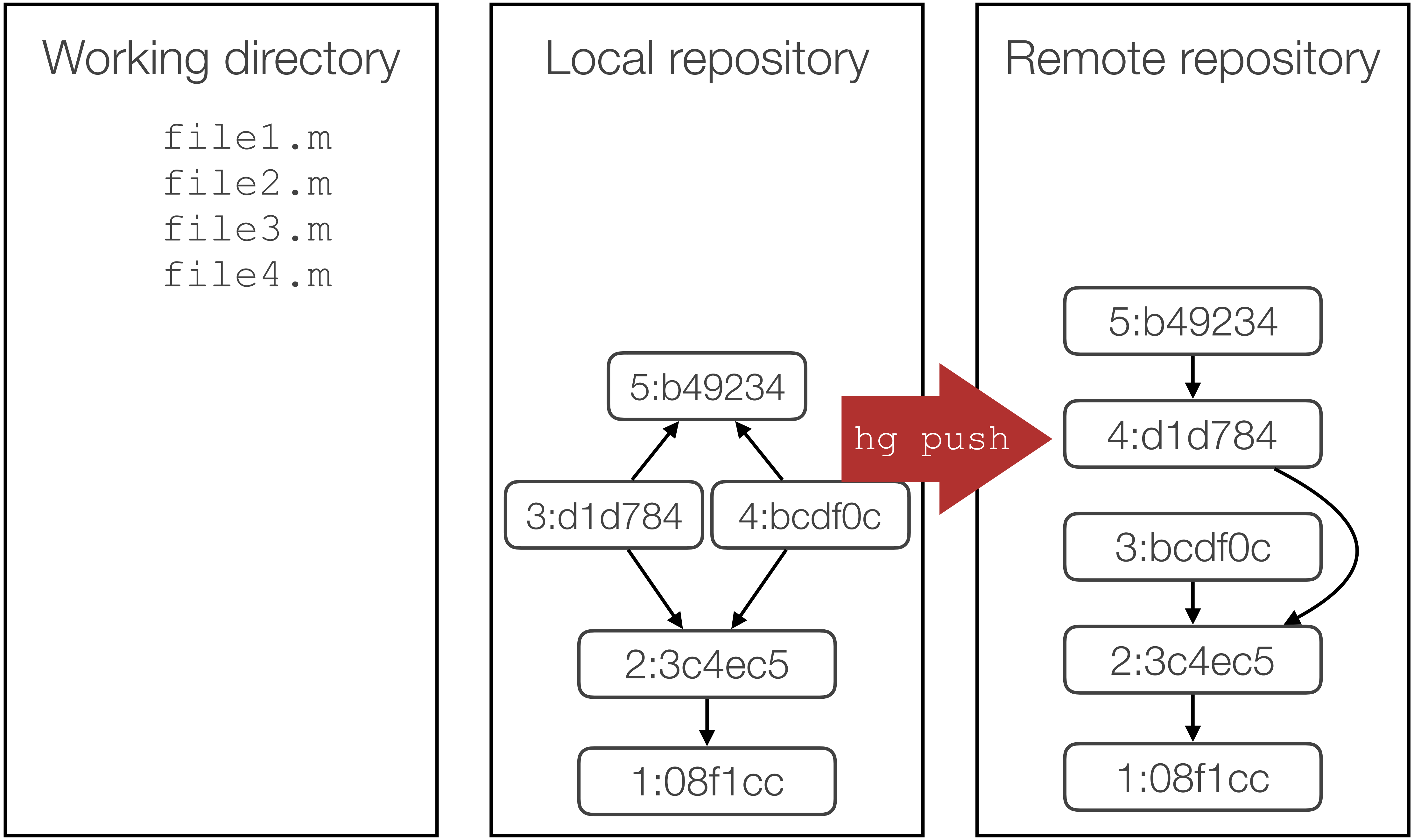
Local repository

5:b49234
3:d1d784
4:bcdf0c
2:3c4ec5
1:08f1cc

Remote repository

3:bcdf0c
2:3c4ec5
1:08f1cc

Mercurial: Basic Commands



Working directory

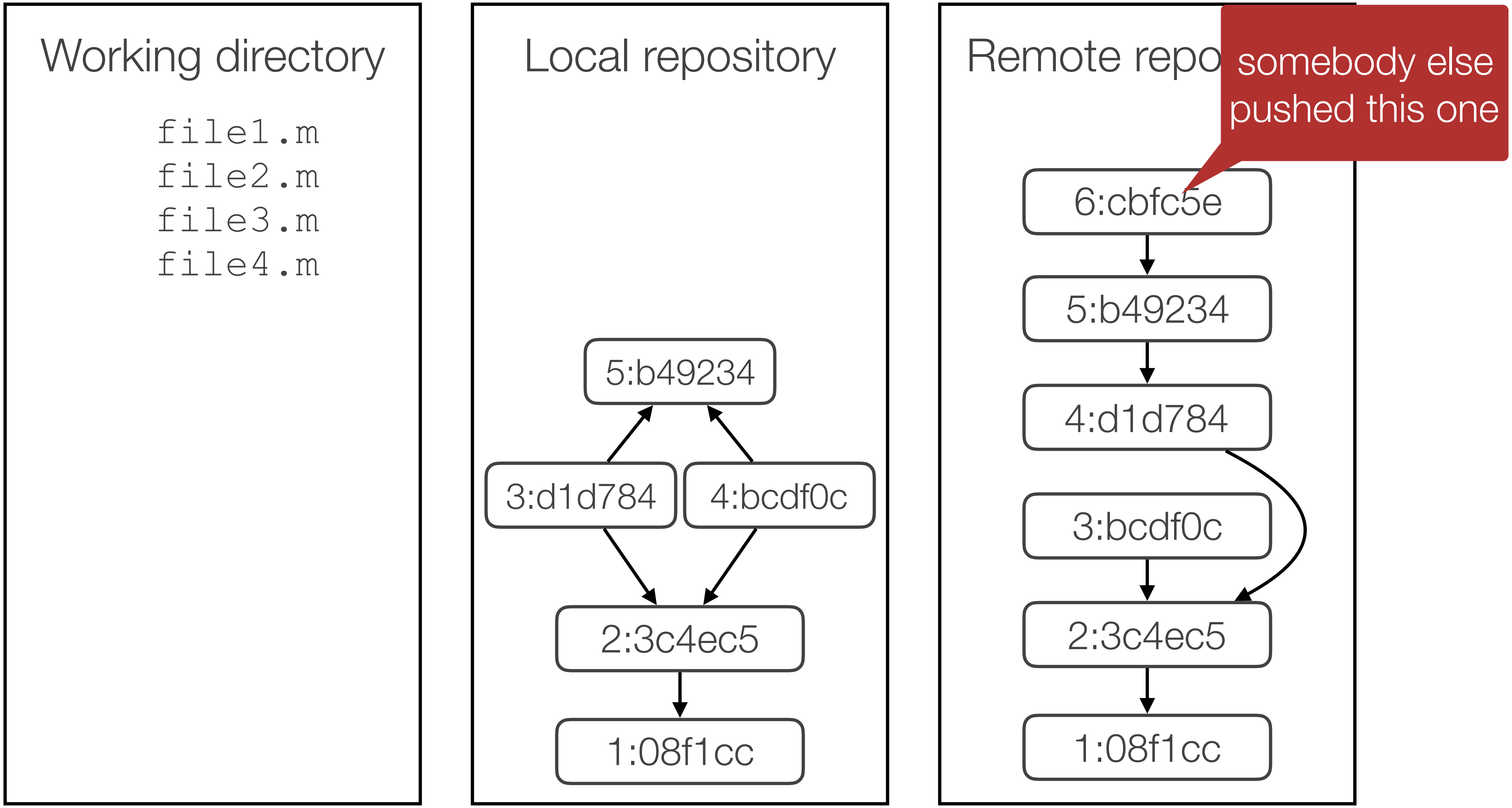
file1.m
file2.m
file3.m
file4.m

Local repository

hg push

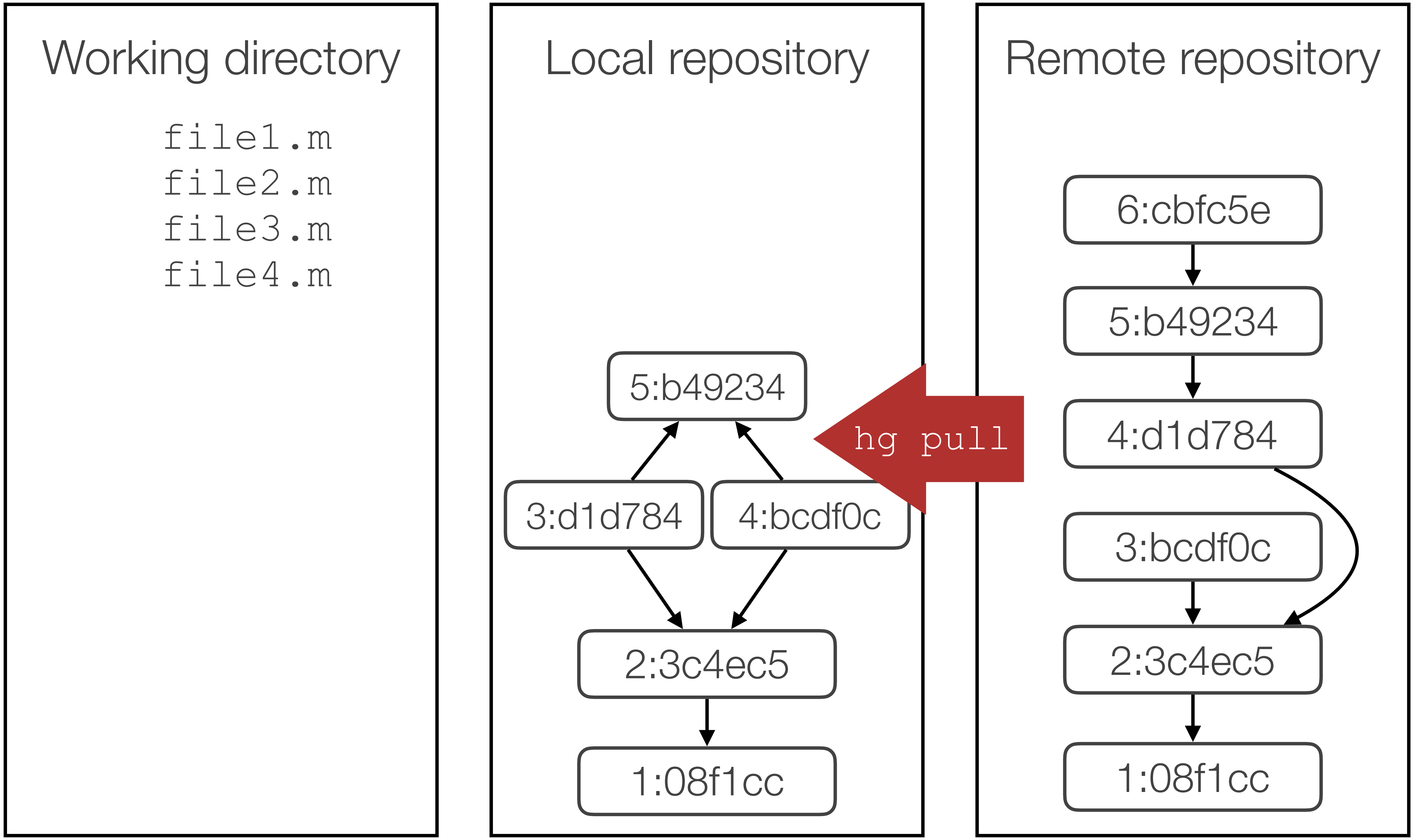
Remote repository

Mercurial: Basic Commands

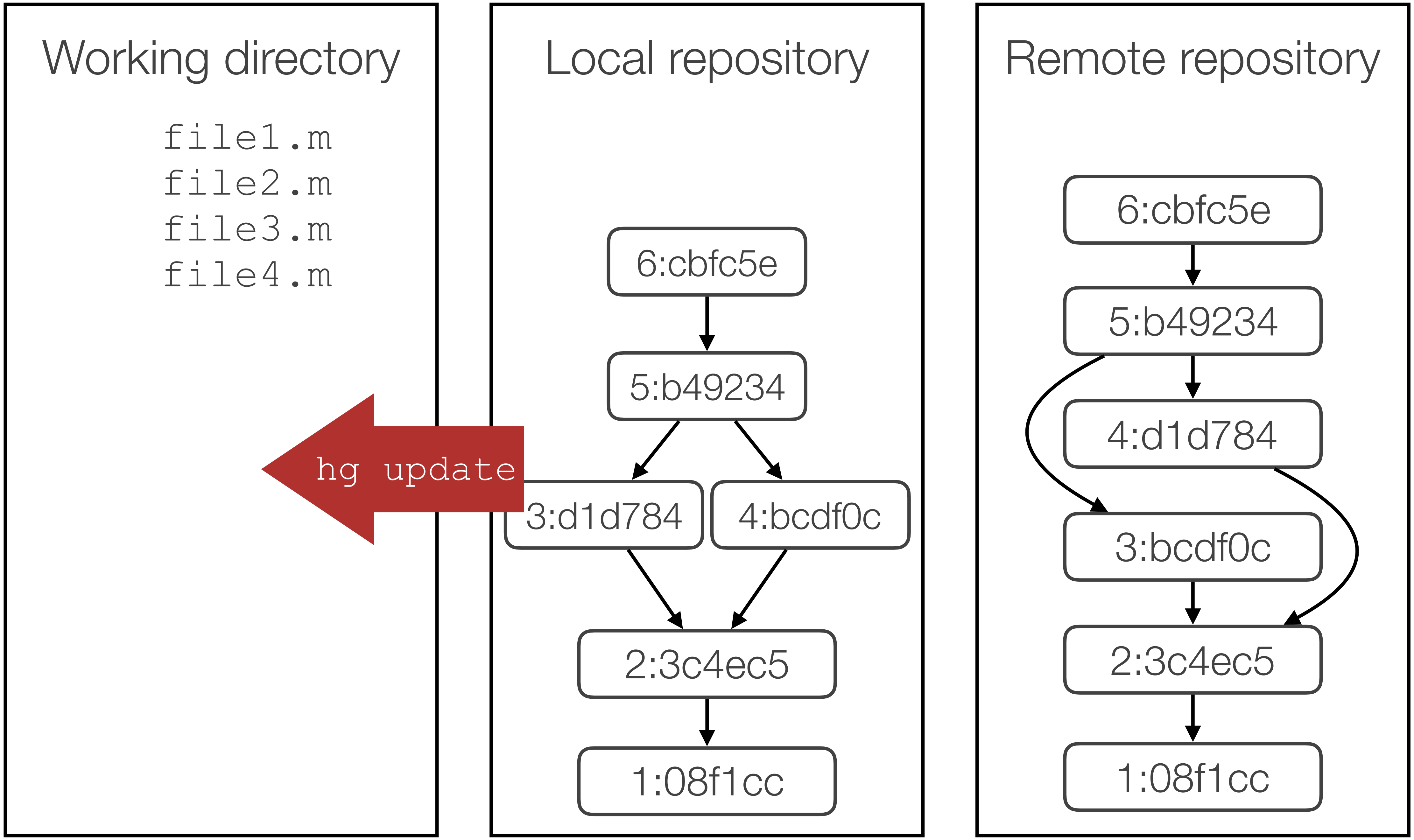


somebody else pushed this one

Mercurial: Basic Commands



Mercurial: Basic Commands



Sometime Merging is Fully Automatic…

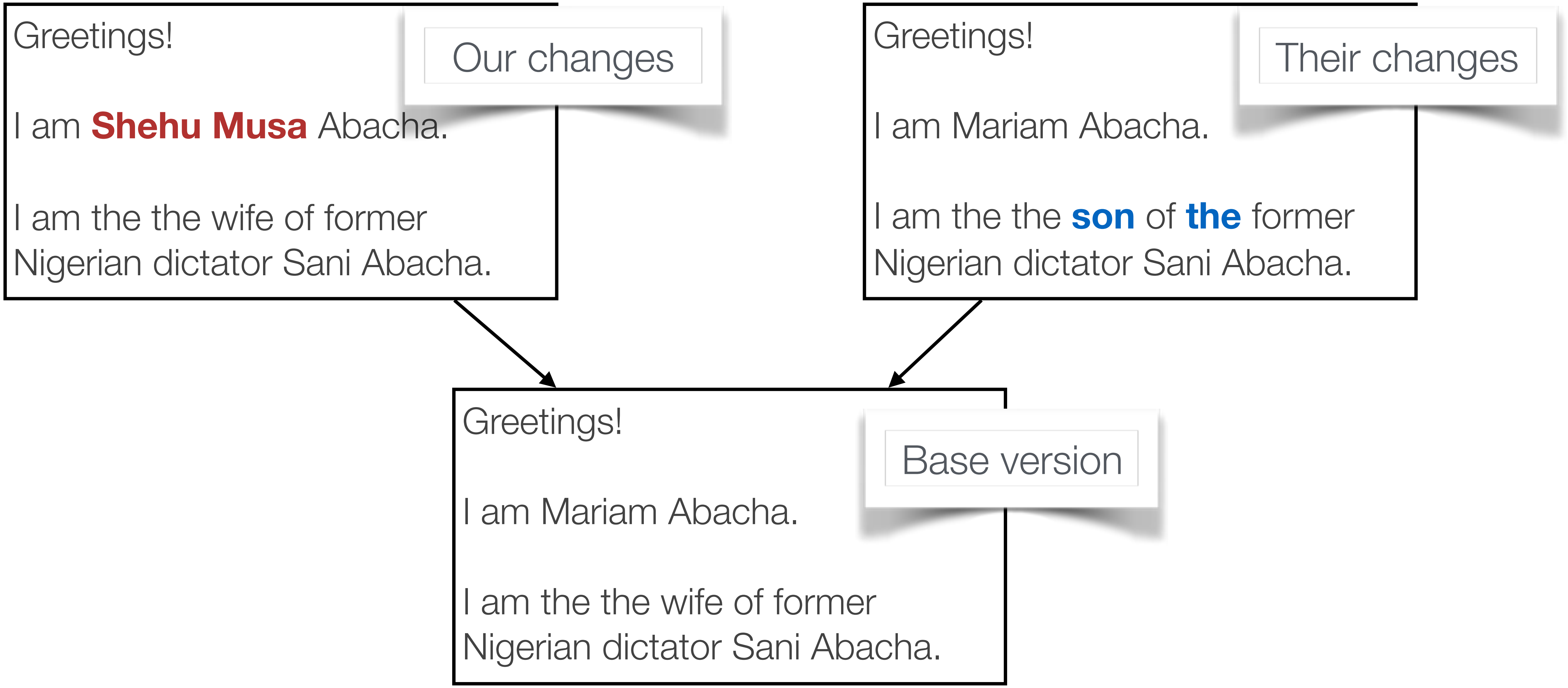
Greetings!

I am Mariam Abacha.

I am the the wife of former
Nigerian dictator Sani Abacha.

Base version

Sometime Merging is Fully Automatic…



Our changes

Greetings!

I am **Shehu Musa** Abacha.

I am the the wife of former Nigerian dictator Sani Abacha.

Their changes

Greetings!

I am Mariam Abacha.

I am the the **son** of **the** former Nigerian dictator Sani Abacha.

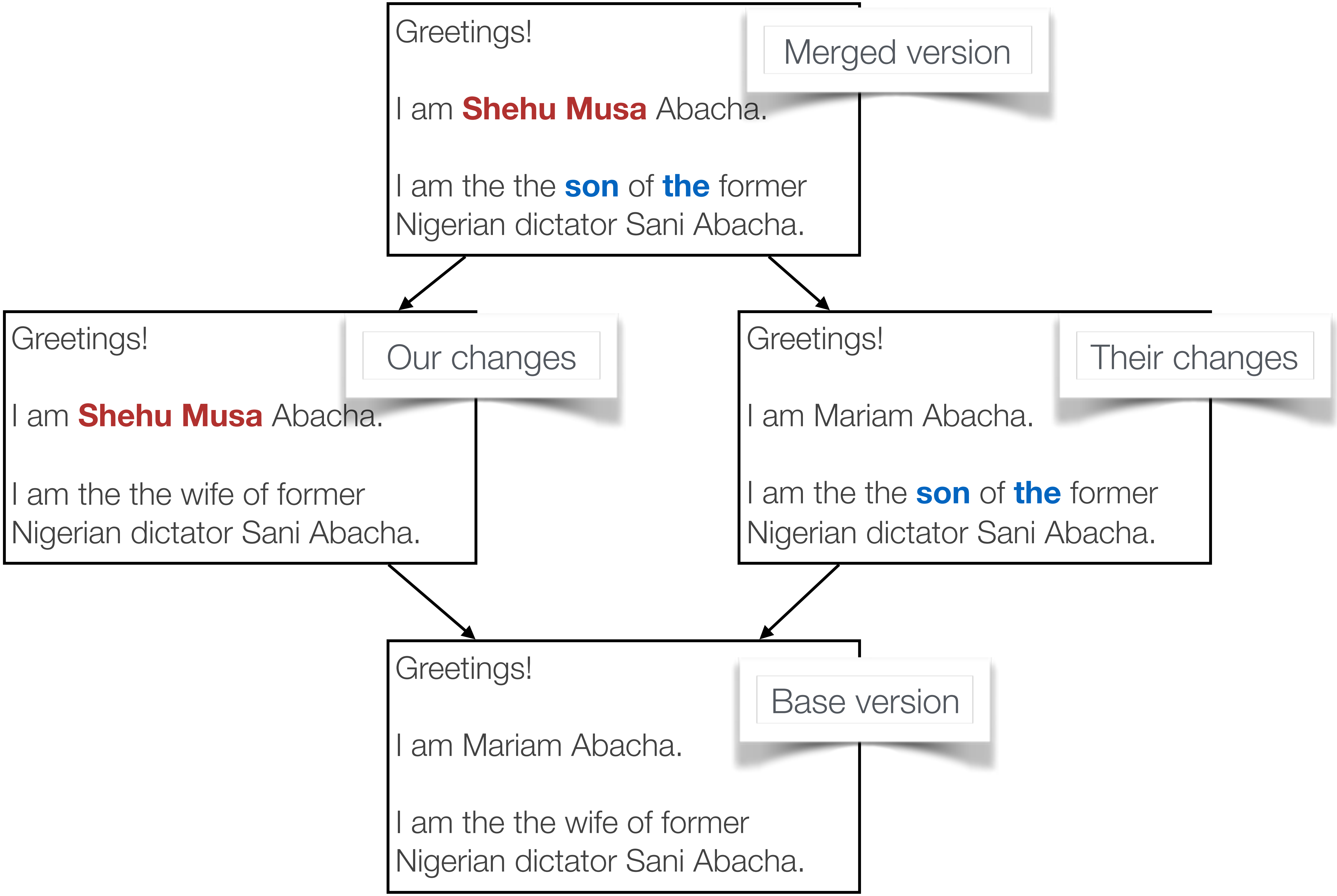
Base version

Greetings!

I am Mariam Abacha.

I am the the wife of former Nigerian dictator Sani Abacha.

Sometime Merging is Fully Automatic…



Merged version

Greetings!

I am **Shehu Musa** Abacha.

I am the the **son** of **the** former Nigerian dictator Sani Abacha.

Our changes

Greetings!

I am **Shehu Musa** Abacha.

I am the the wife of former Nigerian dictator Sani Abacha.

Their changes

Greetings!

I am Mariam Abacha.

I am the the **son** of **the** former Nigerian dictator Sani Abacha.

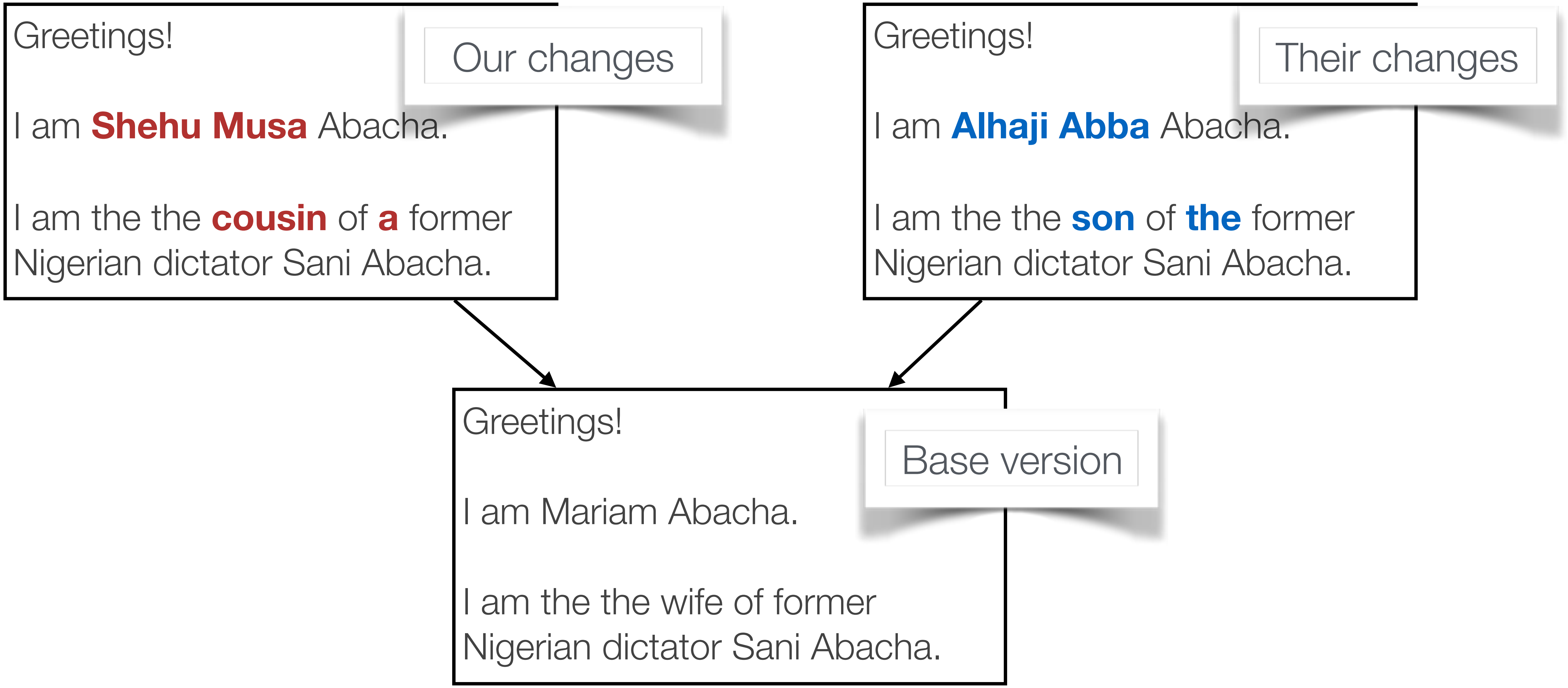
Base version

Greetings!

I am Mariam Abacha.

I am the the wife of former Nigerian dictator Sani Abacha.

…and Sometimes Must be Resolved Manually



Our changes

Greetings!

I am **Shehu Musa** Abacha.

I am the the **cousin** of **a** former Nigerian dictator Sani Abacha.

Their changes

Greetings!

I am **Alhaji Abba** Abacha.

I am the the **son** of **the** former Nigerian dictator Sani Abacha.

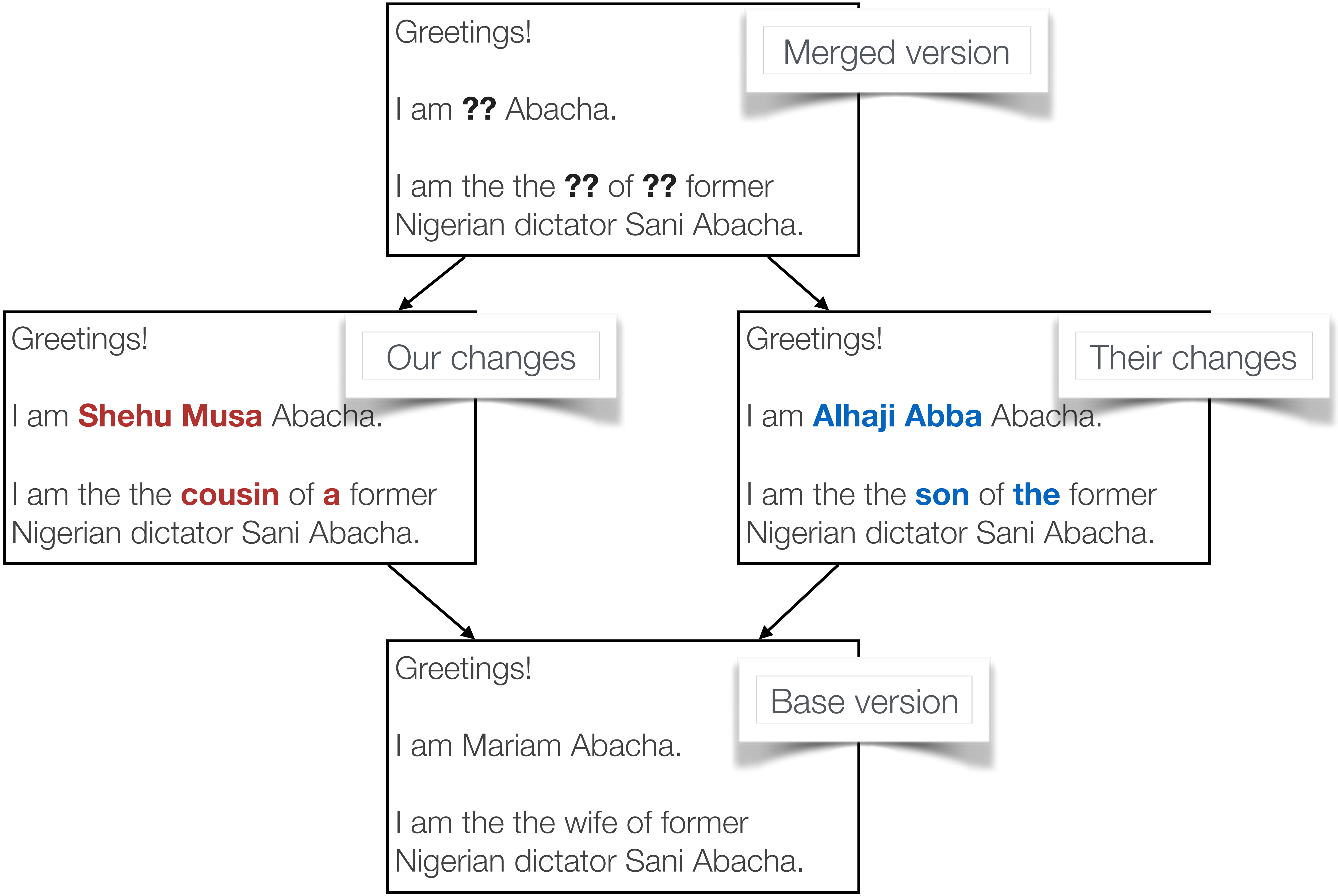
Base version

Greetings!

I am Mariam Abacha.

I am the the wife of former Nigerian dictator Sani Abacha.

…and Sometimes Must be Resolved Manually



Merged version

Greetings!

I am **??** Abacha.

I am the the **??** of **??** former Nigerian dictator Sani Abacha.

Our changes

Greetings!

I am **Shehu Musa** Abacha.

I am the the **cousin** of **a** former Nigerian dictator Sani Abacha.

Their changes

Greetings!

I am **Alhaji Abba** Abacha.

I am the the **son** of **the** former Nigerian dictator Sani Abacha.

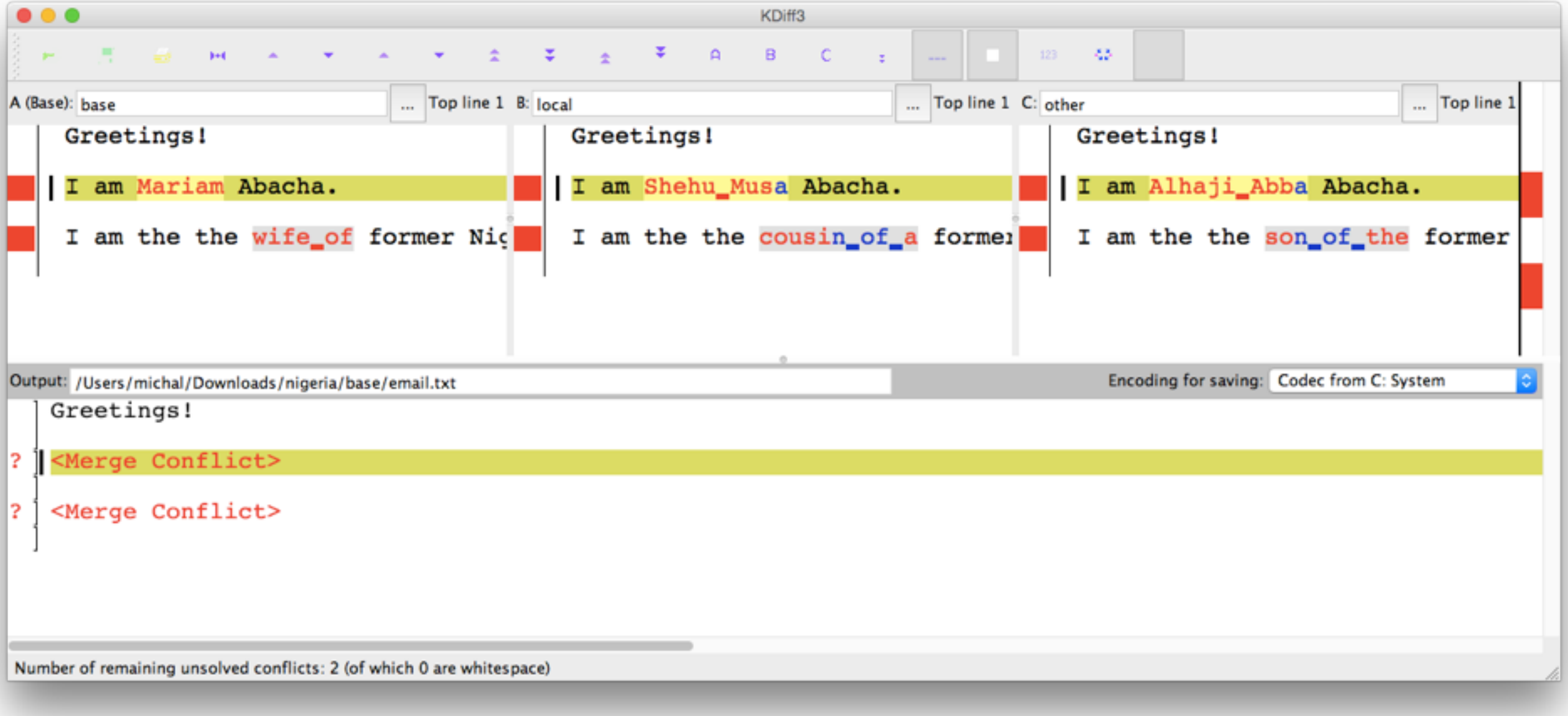
Base version

Greetings!

I am Mariam Abacha.

I am the the wife of former Nigerian dictator Sani Abacha.

…and Sometimes Must be Resolved Manually


KDiff3
A (Base): base
...
Top line 1
B: local
...
Top line 1
C: other
...
Top line 1
Output: /Users/michal/Downloads/nigeria/base/email.txt
Encoding for saving: Codec from C: System

A (base):

```
Greetings!
I am Mariam Abacha.
I am the the wife_of former Nig
```

B (local):

```
Greetings!
I am Shehu_Musa Abacha.
I am the the cousin_of_a formei
```

C (other):

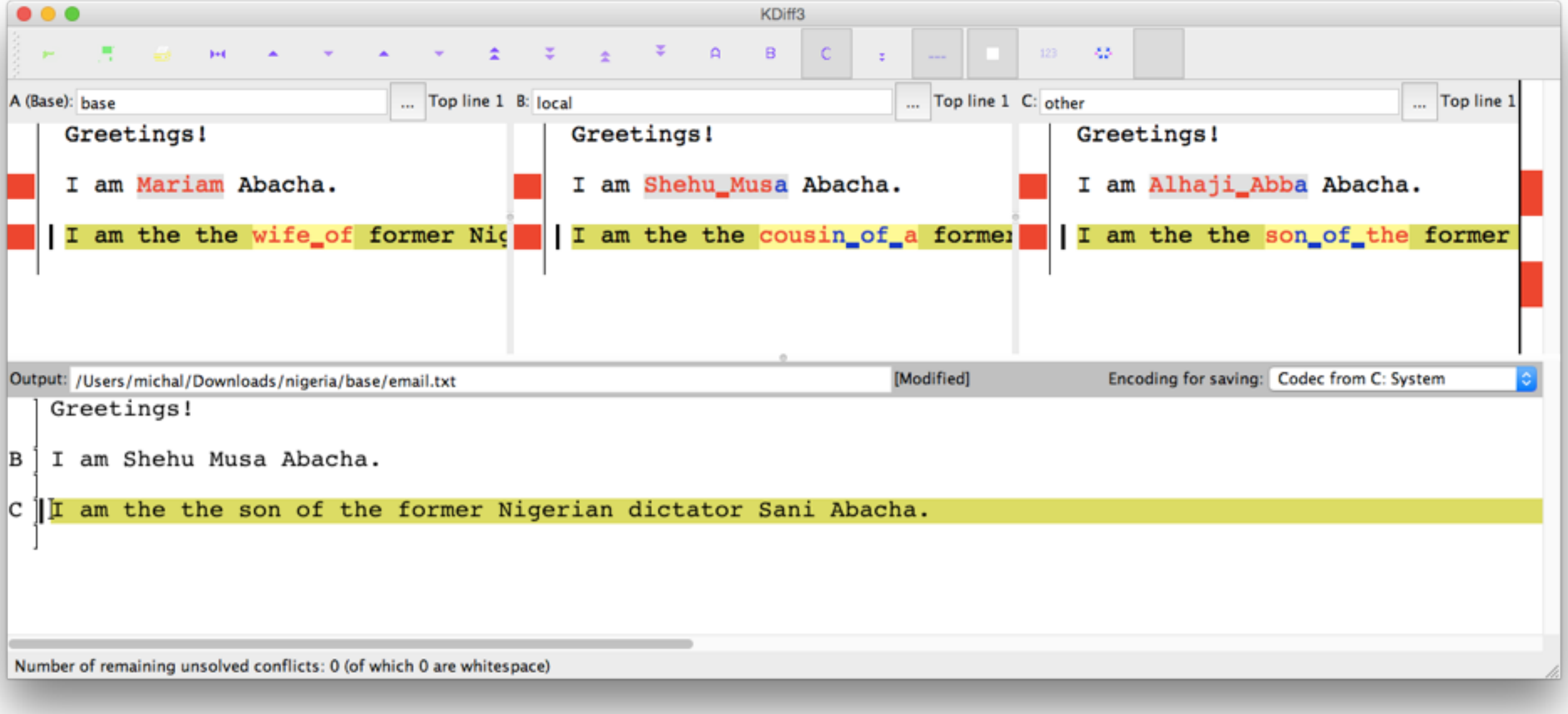
```
Greetings!
I am Alhaji_Abba Abacha.
I am the the son_of_the former
```

Output:

```
  Greetings!
? <Merge Conflict>
? <Merge Conflict>
```

Number of remaining unsolved conflicts: 2 (of which 0 are whitespace)

…and Sometimes Must be Resolved Manually


KDiff3
A (Base): base
... Top line 1
B: local
... Top line 1
C: other
... Top line 1
Output: /Users/michal/Downloads/nigeria/base/email.txt
[Modified]
Encoding for saving: Codec from C: System

A (Base):

```
Greetings!

I am Mariam Abacha.

I am the the wife_of former Nig
```

B: local

```
Greetings!

I am Shehu_Musa Abacha.

I am the the cousin_of_a former
```

C: other

```
Greetings!

I am Alhaji_Abba Abacha.

I am the the son_of_the former
```

Output:

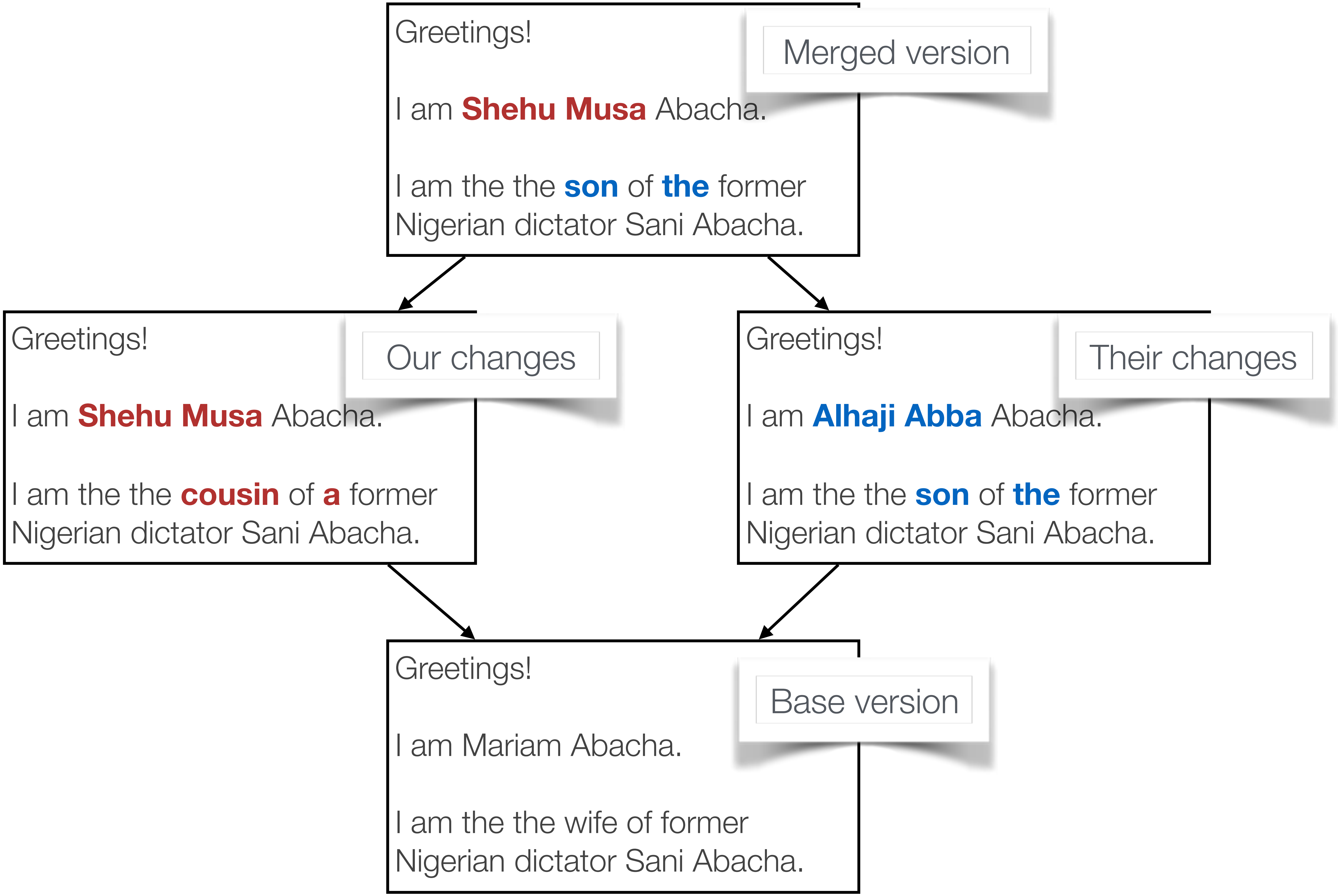
```
  Greetings!

B I am Shehu Musa Abacha.

C I am the the son of the former Nigerian dictator Sani Abacha.
```

Number of remaining unsolved conflicts: 0 (of which 0 are whitespace)

…and Sometimes Must be Resolved Manually



Merged version

Greetings!

I am **Shehu Musa** Abacha.

I am the the **son** of **the** former Nigerian dictator Sani Abacha.

Our changes

Greetings!

I am **Shehu Musa** Abacha.

I am the the **cousin** of **a** former Nigerian dictator Sani Abacha.

Their changes

Greetings!

I am **Alhaji Abba** Abacha.

I am the the **son** of **the** former Nigerian dictator Sani Abacha.

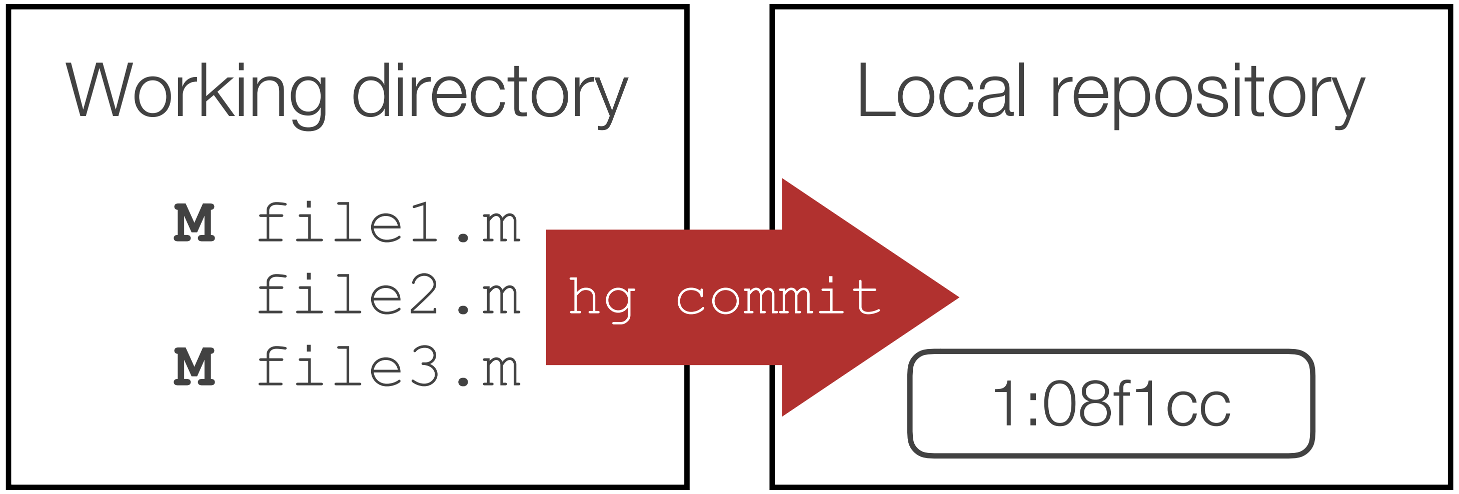
Base version

Greetings!

I am Mariam Abacha.

I am the the wife of former Nigerian dictator Sani Abacha.

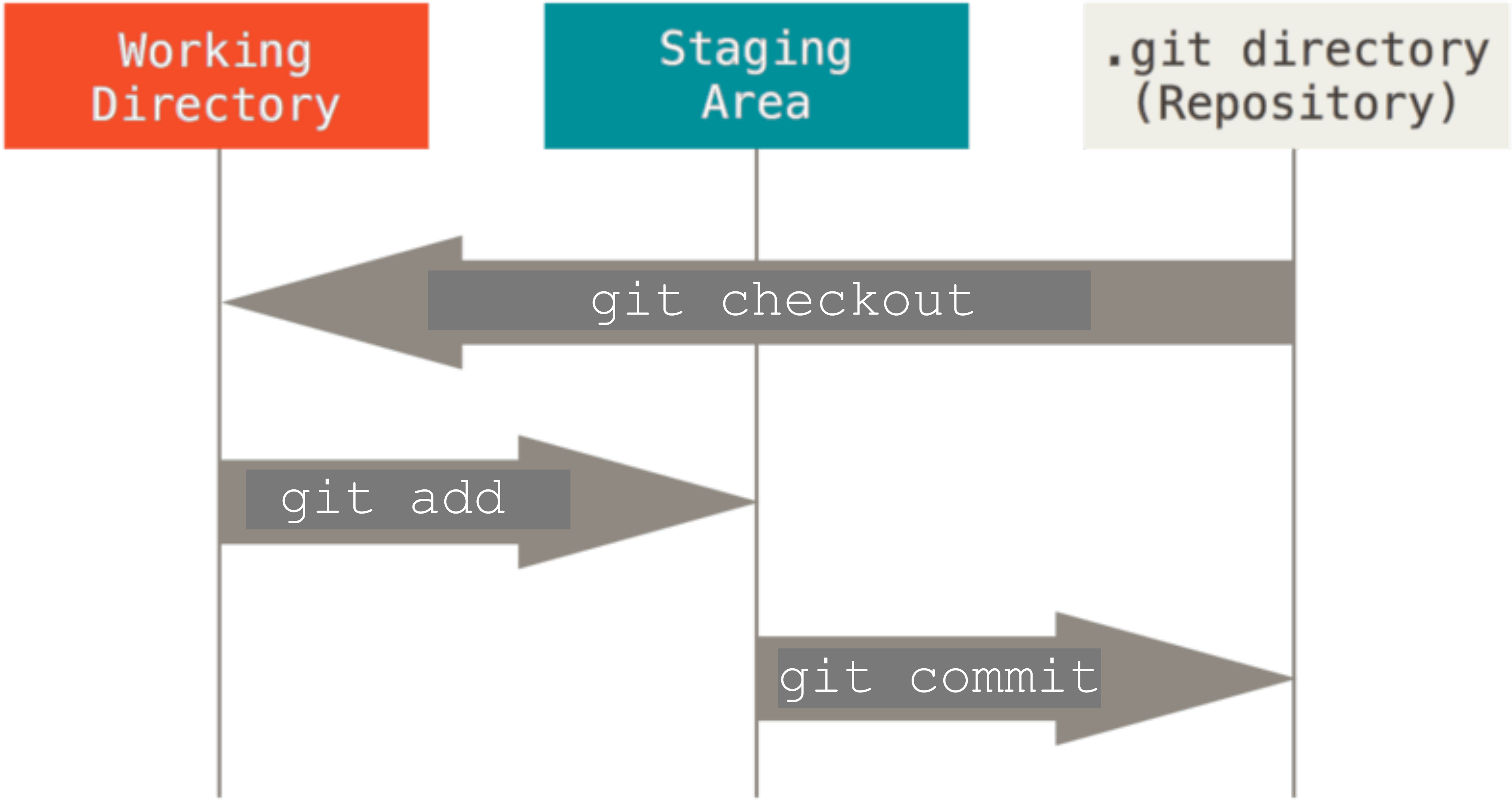
Sidenote: Committing in Git



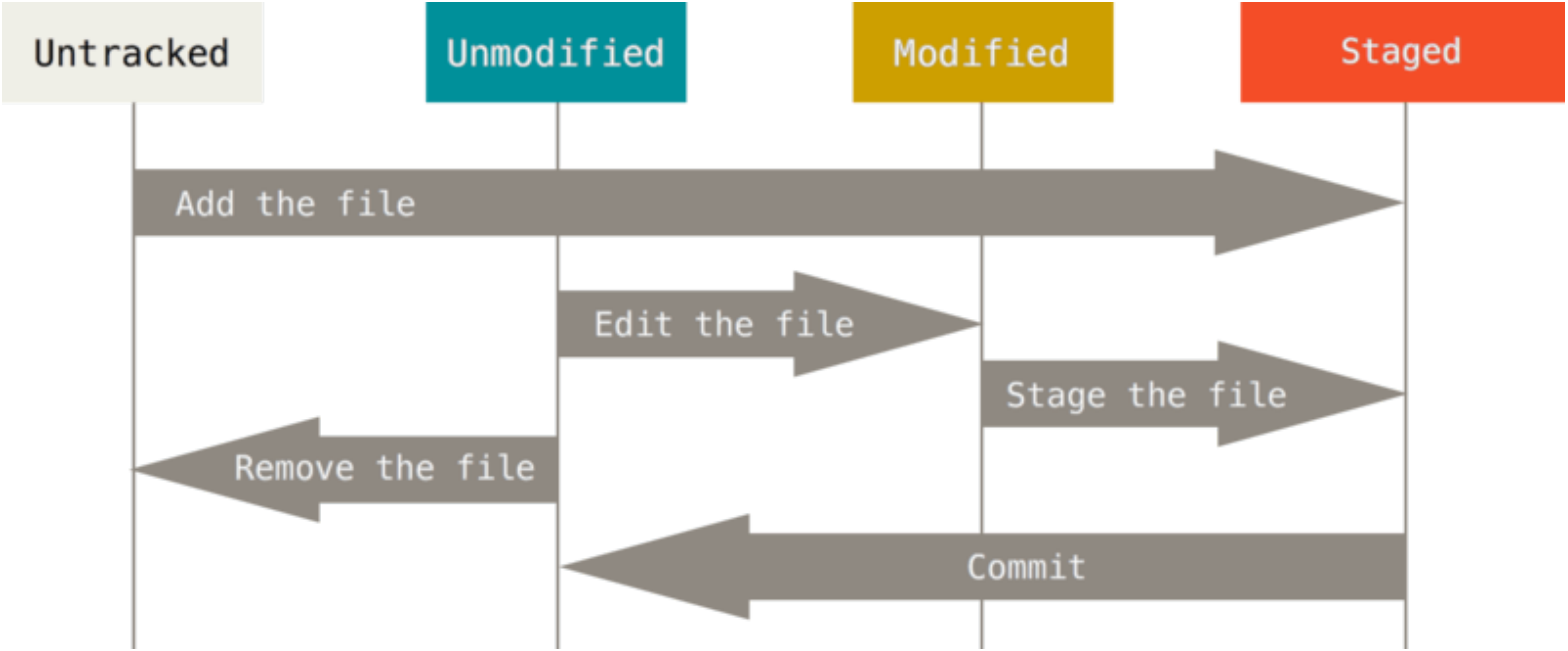
In Mercurial, `hg commit` automatically commits all modified files

In Git, things a bit more complicated…

Sidenote: Committing in Git



Sidenote: Committing in Git

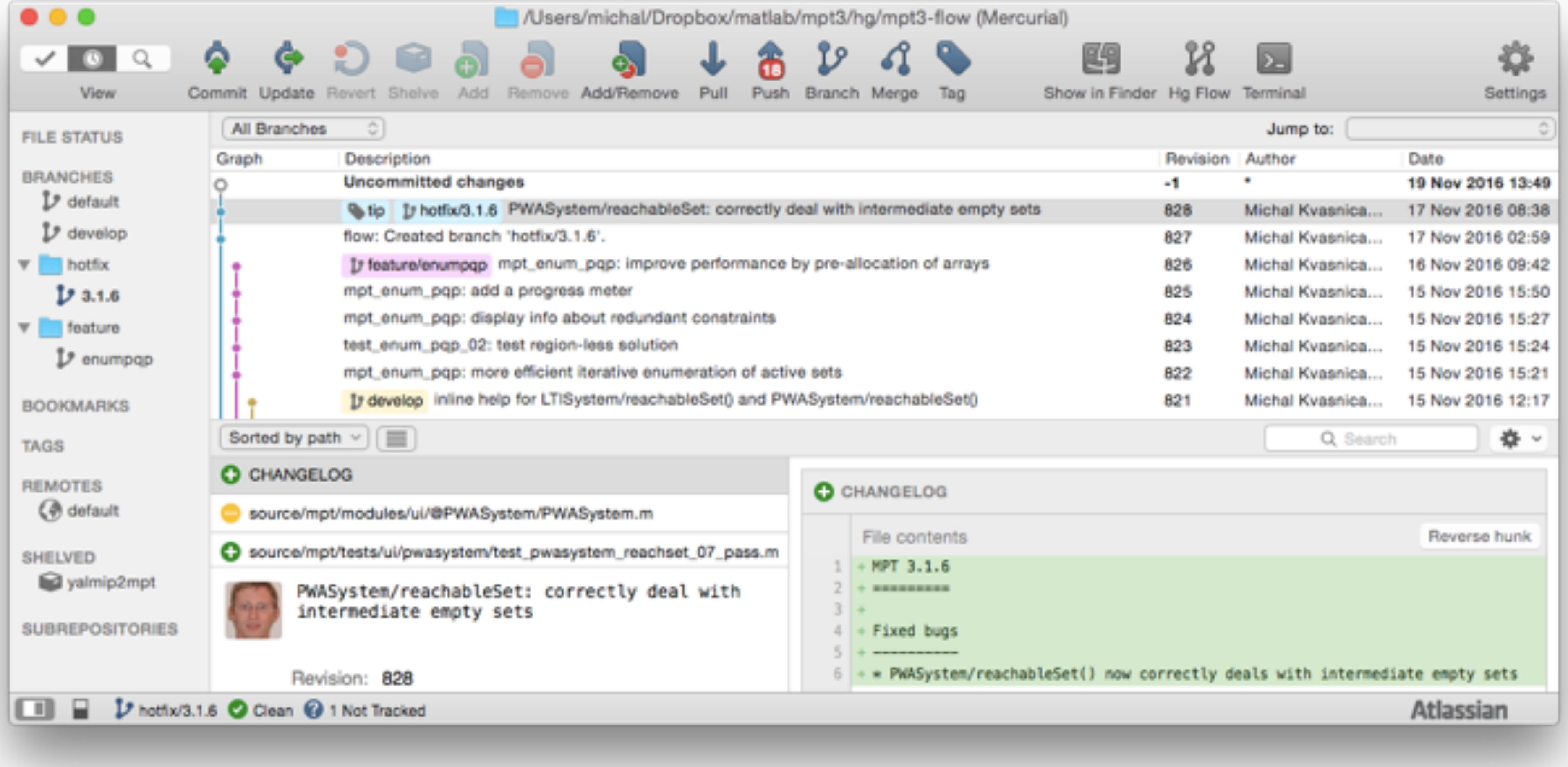


GUIs

Learn the command-line syntax first to understand concepts

Then use GUIs:

- my choice: SourceTree (hg, git, win, mac, free)
- other options: GitHub Desktop, GitEye, SmartGit, …



Graph	Description	Revision	Author	Date
	Uncommitted changes	**-1**	*****	**19 Nov 2016 13:49**
	tip hotfix/3.1.6 PWASystem/reachableSet: correctly deal with intermediate empty sets	828	Michal Kvasnica...	17 Nov 2016 08:38
	flow: Created branch 'hotfix/3.1.6'.	827	Michal Kvasnica...	17 Nov 2016 02:59
	feature/enumpqp mpt_enum_pqp: improve performance by pre-allocation of arrays	826	Michal Kvasnica...	16 Nov 2016 09:42
	mpt_enum_pqp: add a progress meter	825	Michal Kvasnica...	15 Nov 2016 15:50
	mpt_enum_pqp: display info about redundant constraints	824	Michal Kvasnica...	15 Nov 2016 15:27
	test_enum_pqp_02: test region-less solution	823	Michal Kvasnica...	15 Nov 2016 15:24
	mpt_enum_pqp: more efficient iterative enumeration of active sets	822	Michal Kvasnica...	15 Nov 2016 15:21
	develop inline help for LTISystem/reachableSet() and PWASystem/reachableSet()	821	Michal Kvasnica...	15 Nov 2016 12:17

CHANGELOG

source/mpt/modules/ui/@PWASystem/PWASystem.m

source/mpt/tests/ui/pwasystem/test_pwasystem_reachset_07_pass.m

```
PWASystem/reachableSet: correctly deal with
intermediate empty sets
```

Revision: 828

CHANGELOG

File contents Reverse hunk

```
+ MPT 3.1.6
+ =========
+
+ Fixed bugs
+ ----------
+ * PWASystem/reachableSet() now correctly deals with intermediate empty sets
```

SourceTree

Demo

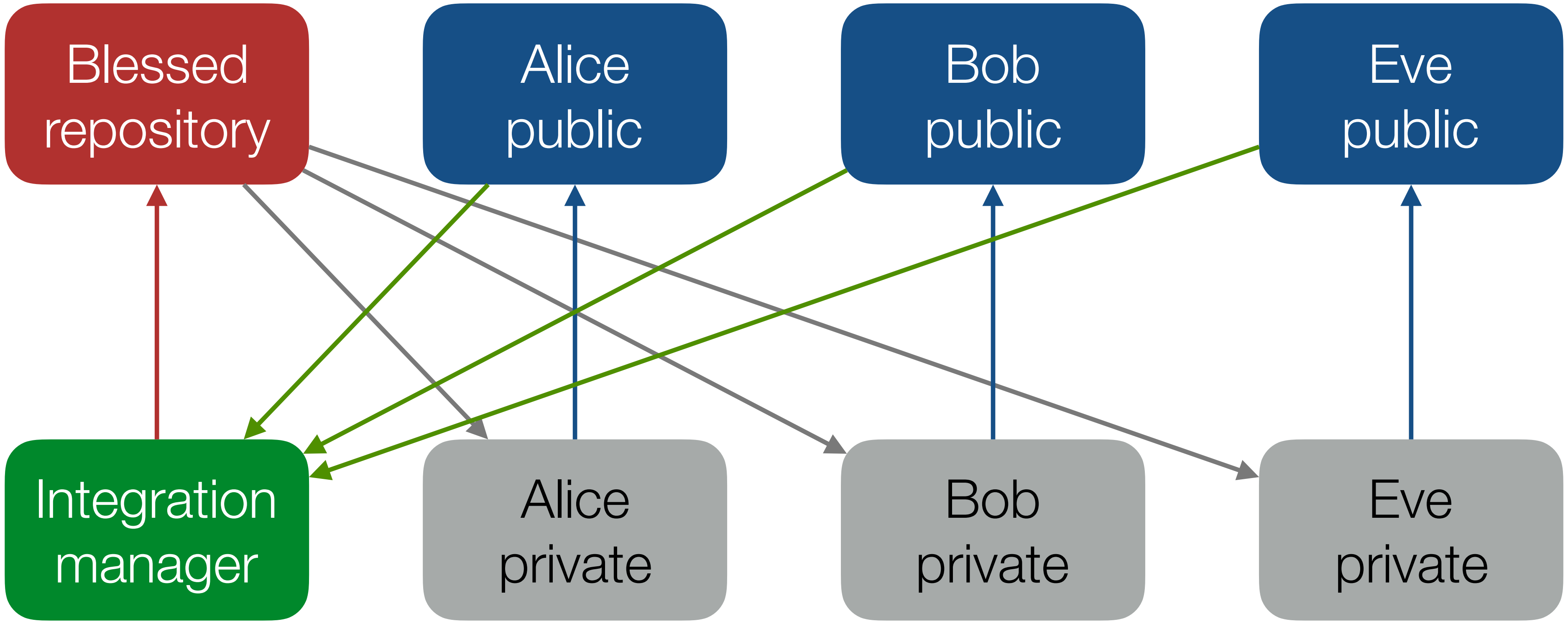
Public Repository Hosting

Cloud-based:

- Bitbucket (Mercurial+Git)
- GitHub (only Git)
- GitLab (only Git)

Self-hosted:

- Rhodecode (Mercurial+Git)
- Bitbucket Server (only Git)
- GitLab (only Git)



Bitbucket vs GitHub

Both:

- are free for academia (Bitbucket even free for small teams)
- offer code hosting, code review, issue tracking, wikis
- support pull requests

My choice: Bitbucket

But GitHub is more popular and visually more appealing

Bitbucket

Demo

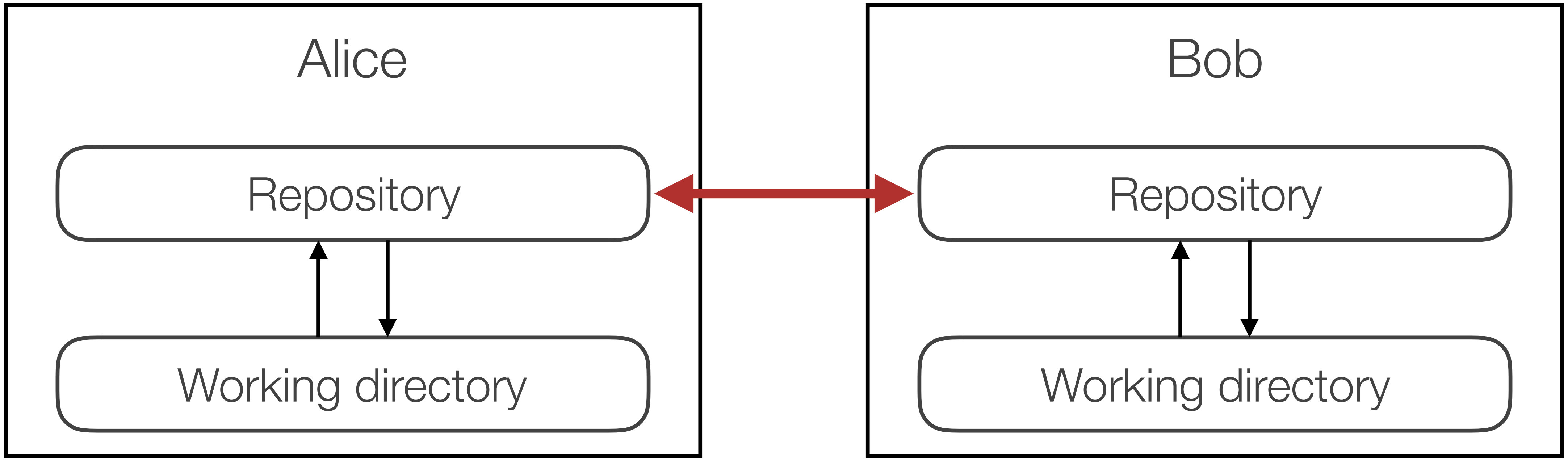
Pull Requests

Main idea: everybody has a local copy of the repository

- every developer can commit/update at any time to his/her repository
- no permanent connection required

Question: **communication? ("Hey, I made a change, you need to update")**

Answer: **pull requests**



Pull Requests in Bitbucket

Demo

Mercurial vs Git: Summary

DOs:

- learn the command line syntax to understand the concepts
- install a merging tool (e.g. kdiff3)
- pick a GUI that supports both systems
- write meaningful commit messages
- use pull requests

DONTs:

- do not lead religious wars about your choices, write code

Agenda

Today

- version control systems and collaborative development
- Mercurial, Git, Bitbucket, GitHub
- **providing support**

Tomorrow

- unit testing
- documentation
- dissemination

Providing Support

Request tracking:

- email + a request tracking system

Bug tracking:

- Bitbucket/GitHub

Discussion groups/fora:

- Google Groups

Request tracking

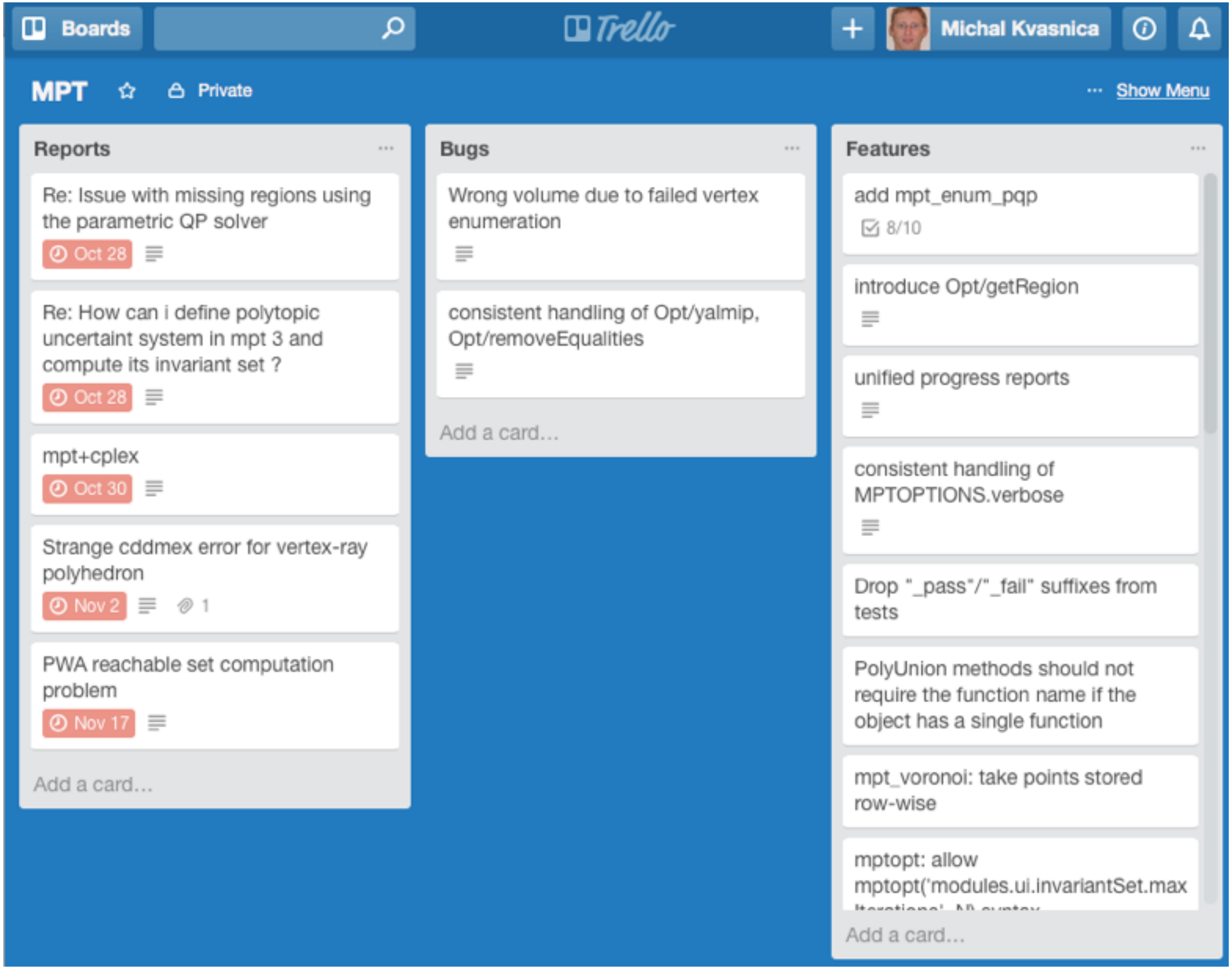
Create a generic email (e.g. `mpt@`) instead of using `username@`

- allows to dispatch the email to a support group
- lets you to change your affiliation later

Keep track of which emails have been answered:

- sounds easy, but difficult if we have multiple persons providing support
- use Trello as a simple request tracking system
- or set up a more complex system (Request Tracker, Roundup, …)

Trello



Reports

Re: Issue with missing regions using the parametric QP solver
Oct 28

Re: How can i define polytopic uncertaint system in mpt 3 and compute its invariant set ?
Oct 28

mpt+cplex
Oct 30

Strange cddmex error for vertex-ray polyhedron
Nov 2 1

PWA reachable set computation problem
Nov 17

Add a card…

Bugs

Wrong volume due to failed vertex enumeration

consistent handling of Opt/yalmip, Opt/removeEqualities

Add a card…

Features

add mpt_enum_pqp
☑ 8/10

introduce Opt/getRegion

unified progress reports

consistent handling of MPTOPTIONS.verbose

Drop "_pass"/"_fail" suffixes from tests

PolyUnion methods should not require the function name if the object has a single function

mpt_voronoi: take points stored row-wise

mptopt: allow mptopt('modules.ui.invariantSet.maxIterations', N) syntax

Add a card…

Bug Tracking

10 years ago:

- many options: Bugzilla, Redmine, Trac, Roundup
- tried them all, liked Roundup the most

Today:

- use issue tracking modules in Bitbucket and GitHub

Persuade your users to submit bug reports

How to Write Good Bug Reports

Check if you are using the latest version and whether the bug is known

File each issue separately

- bad: "fun1 errors, fun2 misbehaves, and, btw, there is a typo in fun3"

Give the report a descriptive name

- bad: "IT DOES NOT WORK!"
- good: "fun1 breaks when called with a string"

State what is the expected result and what is the actual behavior

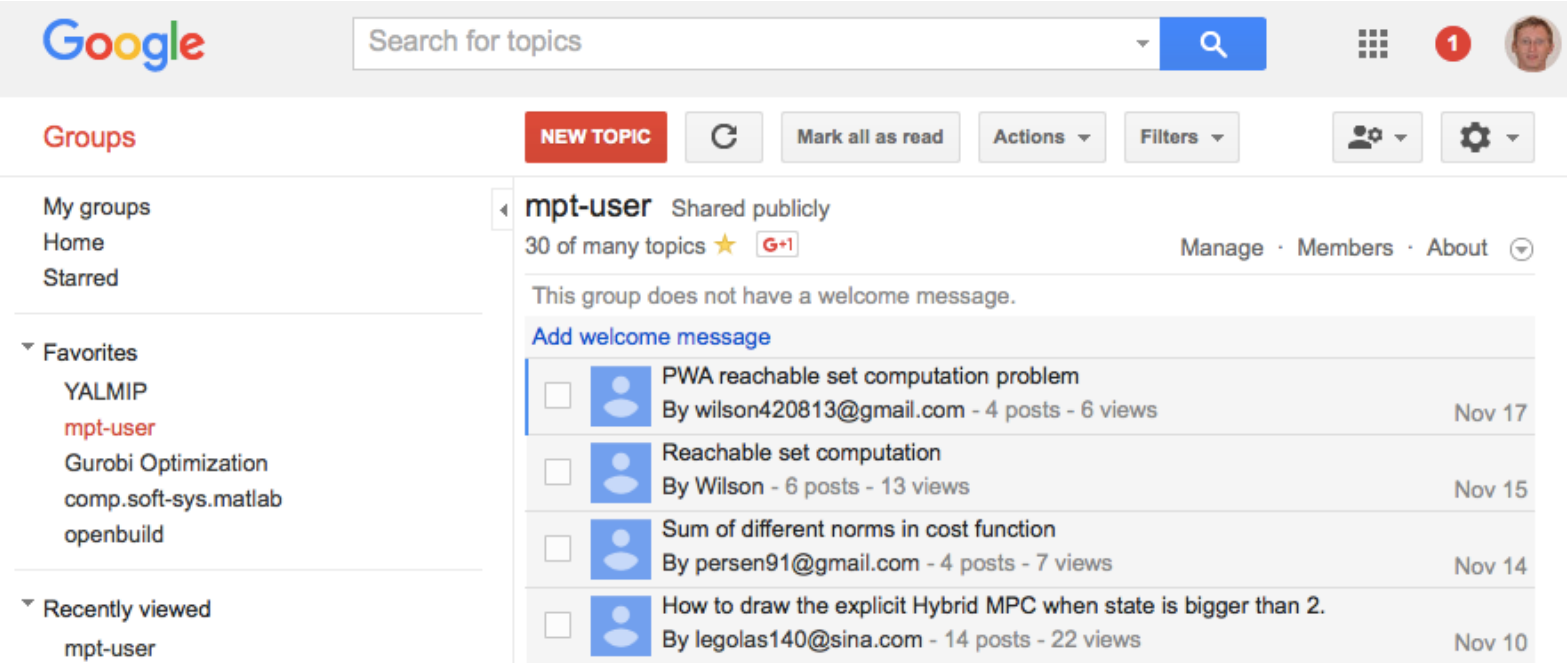
Give steps to reproduce the issue

- give information about your setup (OS/Matlab version, etc.)
- provide a minimal code!

Discussion Groups

Fact: same questions are asked over and over again

Idea: let users provide support to each other



mpt-user Shared publicly

30 of many topics

This group does not have a welcome message.

Add welcome message

PWA reachable set computation problem
By wilson420813@gmail.com - 4 posts - 6 views Nov 17

Reachable set computation
By Wilson - 6 posts - 13 views Nov 15

Sum of different norms in cost function
By persen91@gmail.com - 4 posts - 7 views Nov 14

How to draw the explicit Hybrid MPC when state is bigger than 2.
By legolas140@sina.com - 14 posts - 22 views Nov 10

Side note: redirect the support mail to the group

Google Groups

Demo

Agenda

Today

- version control systems and collaborative development
- Mercurial, Git, Bitbucket, GitHub
- providing support

Tomorrow

- unit testing
- documentation
- dissemination