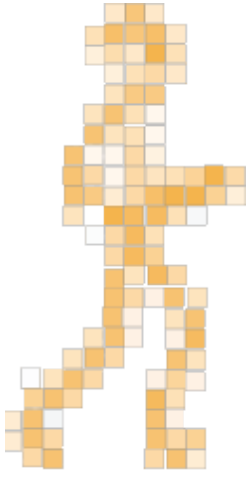
Vendor Branches in Mercurial: Elegant Management of Third-Party Code

Martin Geisler
⟨mg@aragost.com⟩

Mercurial Geek Night II
November 24th, 2010

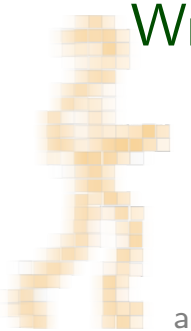


About the Speaker

Martin Geisler:

- core Mercurial developer:
  - reviews patches from the community
  - helps users in our IRC channel
- PhD in Computer Science from Aarhus University, DK
  - exchange student at ETH Zurich in 2005
  - visited IBM Zurich Research Lab in 2008
- now working at aragost Trifork, Zurich
  - offers professional Mercurial support
  - customization, migration, training
  - advice on best practices

Outline



Outline

What is Mercurial?

Main features:

- fast, distributed revision control system
- robust support for branching and merging
- very flexible and extensible

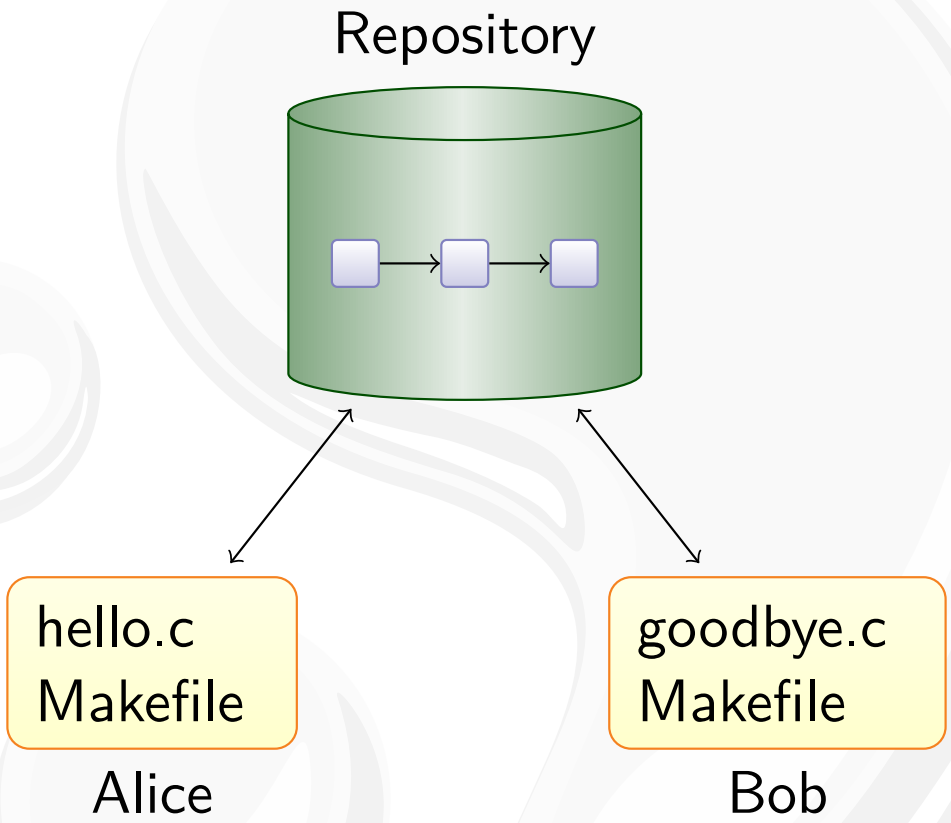
Who is Using it?

Mercurial is used by:

- Oracle for Java, OpenSolaris, NetBeans, OpenOffice, ...
- Mozilla for Firefox, Thunderbird, ...
- Google
- many more...

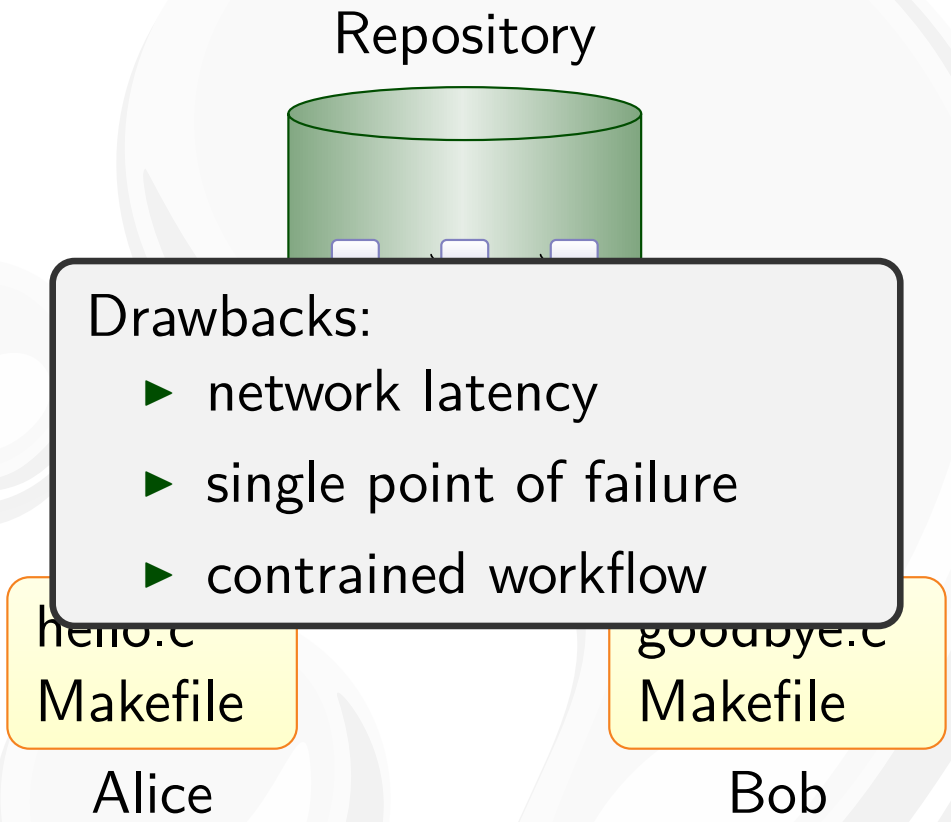
Centralized Revision Control

Single repository, multiple working copies:



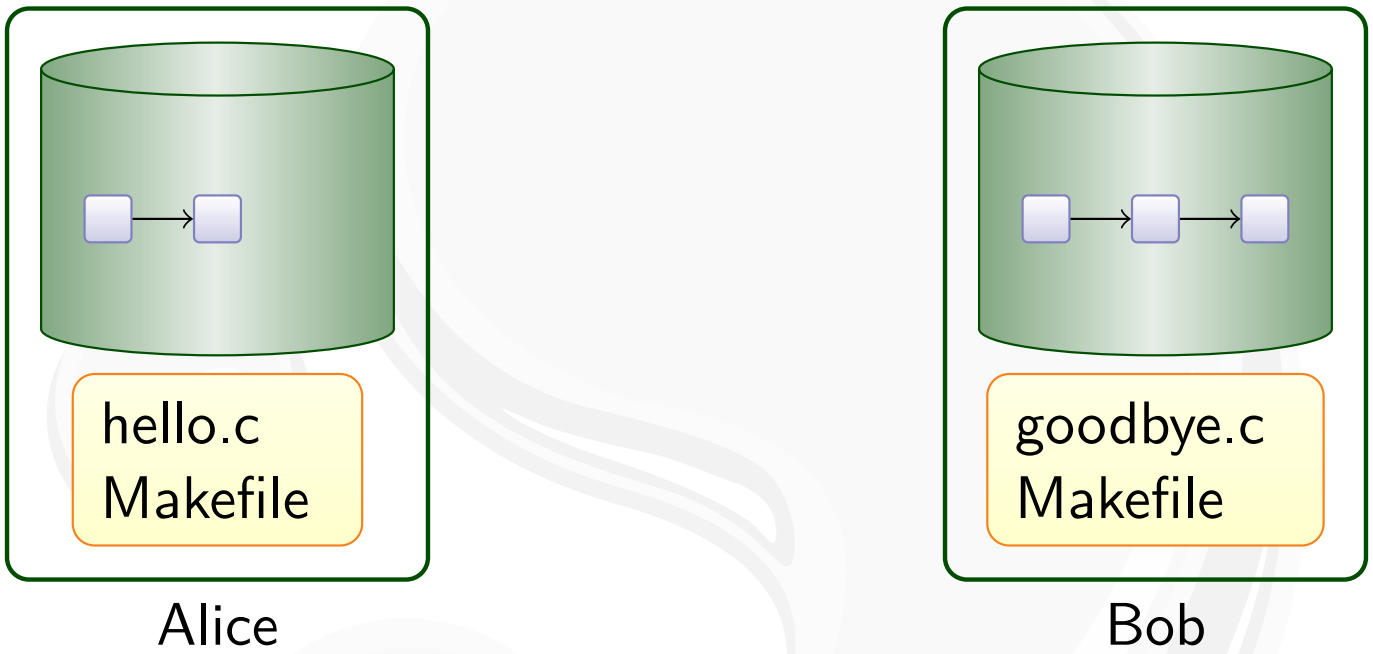
Centralized Revision Control

Single repository, multiple working copies:



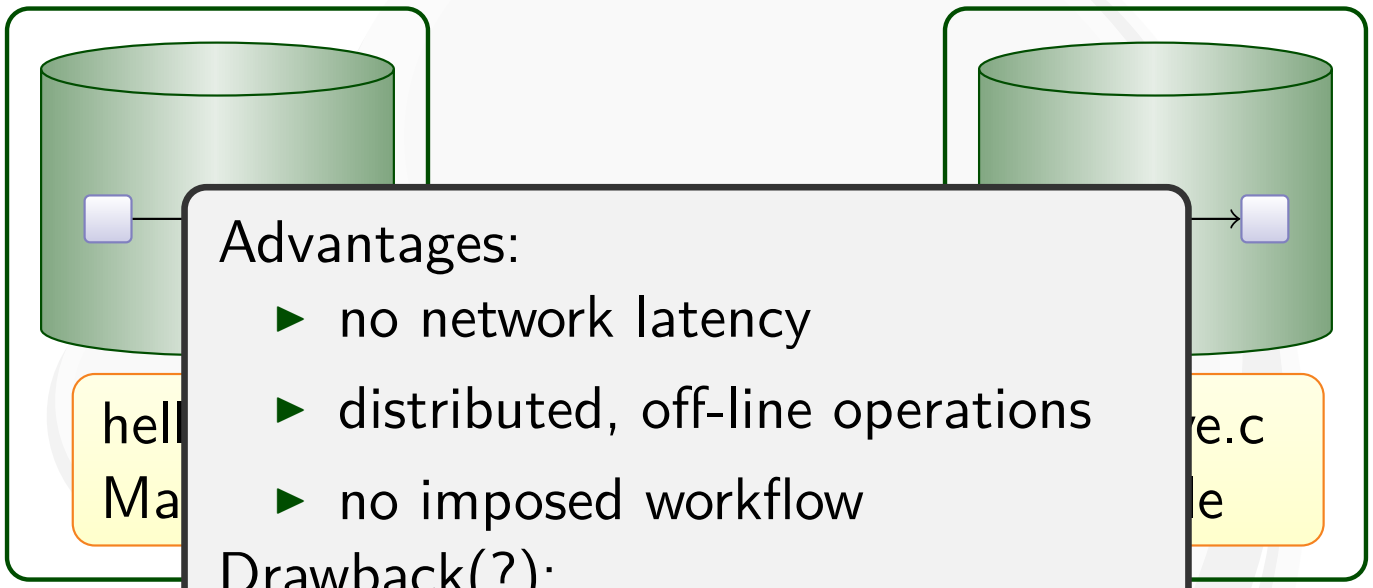
Distributed Revision Control

Mercurial duplicates the history on many servers:



Distributed Revision Control

Mercurial duplicates the history on many servers:



Advantages:

- no network latency
- distributed, off-line operations
- no imposed workflow

Drawback(?):

- must synchronize repositories

Moving Changesets Around

Pull and merge:

Alice

Bob

Moving Changesets Around

Pull and merge:

Moving Changesets Around

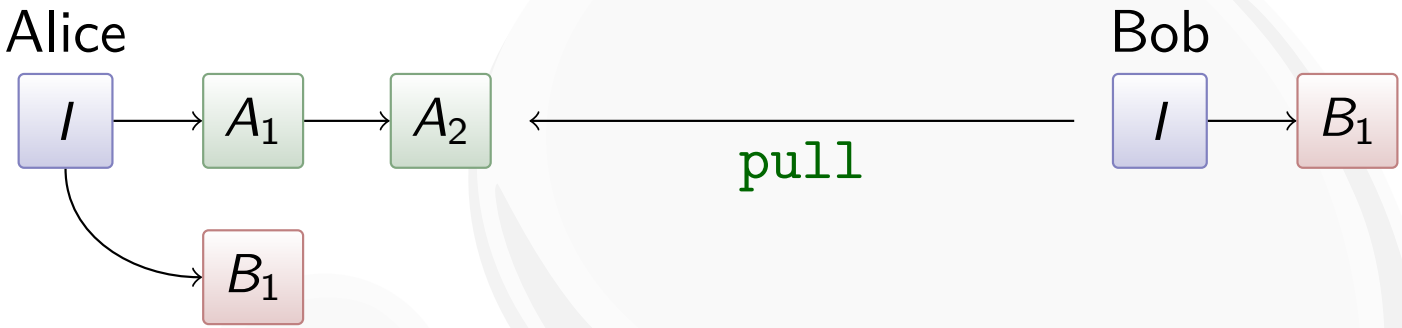
Pull and merge:

Moving Changesets Around

Pull and merge:

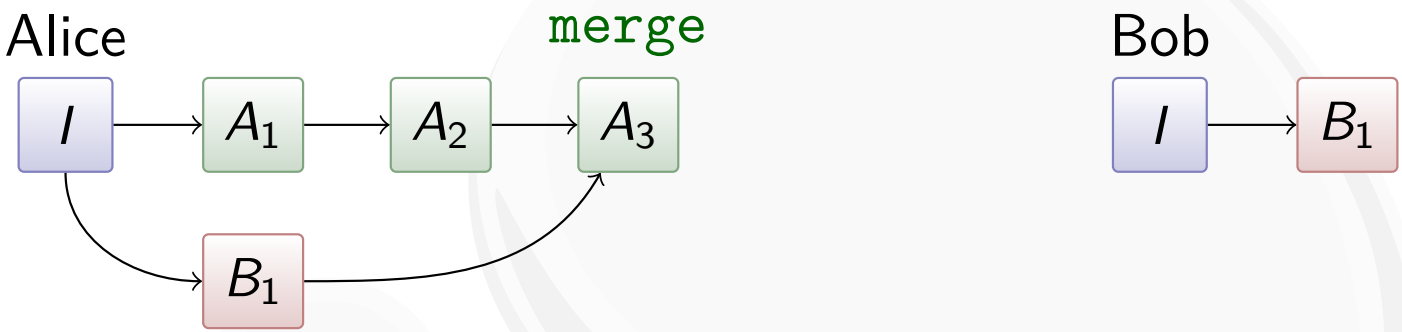
Moving Changesets Around

Pull and merge:



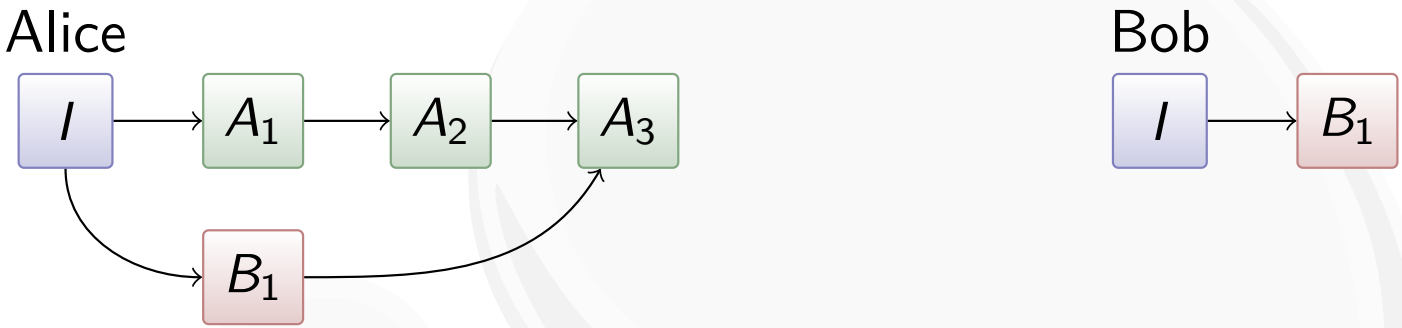
Moving Changesets Around

Pull and merge:



Moving Changesets Around

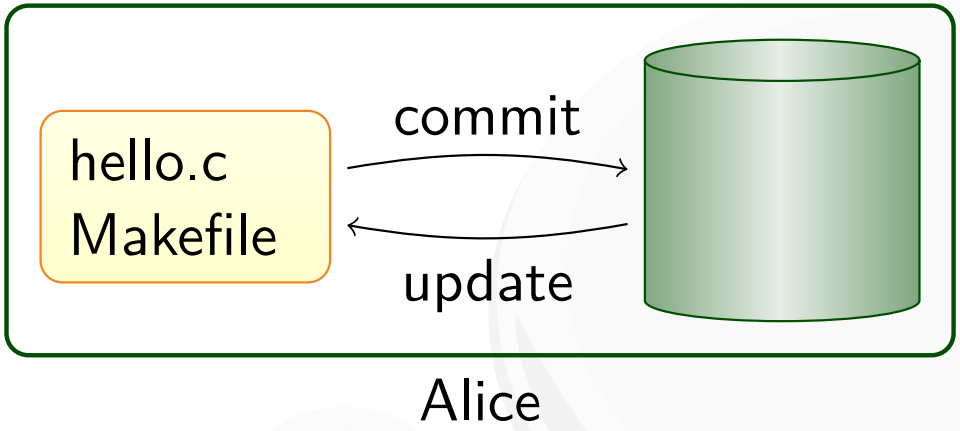
Pull and merge:



Merging:

- find common ancestor of A_2 and B_1: I
- compute differences between I and B_1
- apply them to A_2, taking renames into account

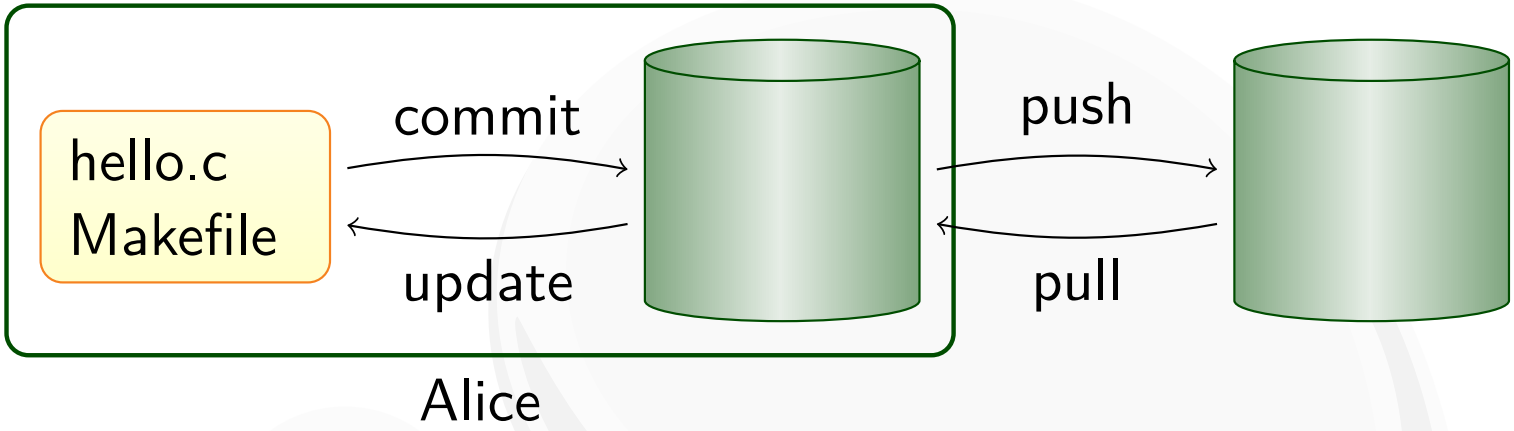
Key Mercurial Commands



Local commands:

- `hg commit`: save a snapshot into the current repository
- `hg update`: checkout revision into working directory
- `hg merge`: join different lines of history

Key Mercurial Commands



Local commands:

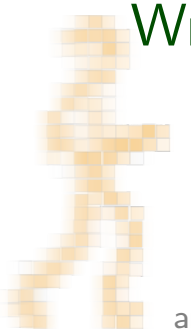
- `hg commit`: save a snapshot into the current repository
- `hg update`: checkout revision into working directory
- `hg merge`: join different lines of history

Network commands:

- `hg pull`: retrieve changesets from another repository
- `hg push`: send your changesets to another repository

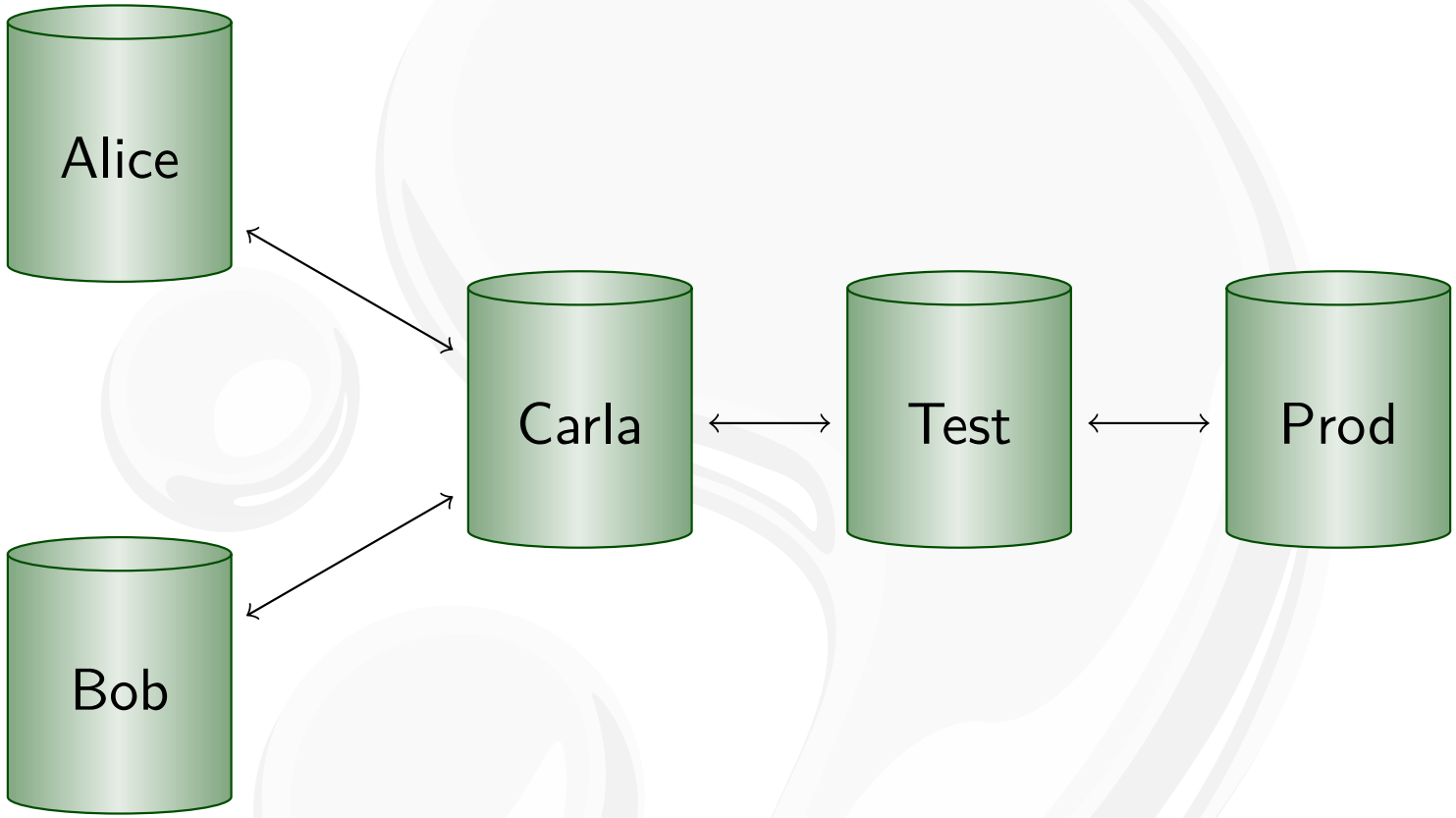
Outline

Outline



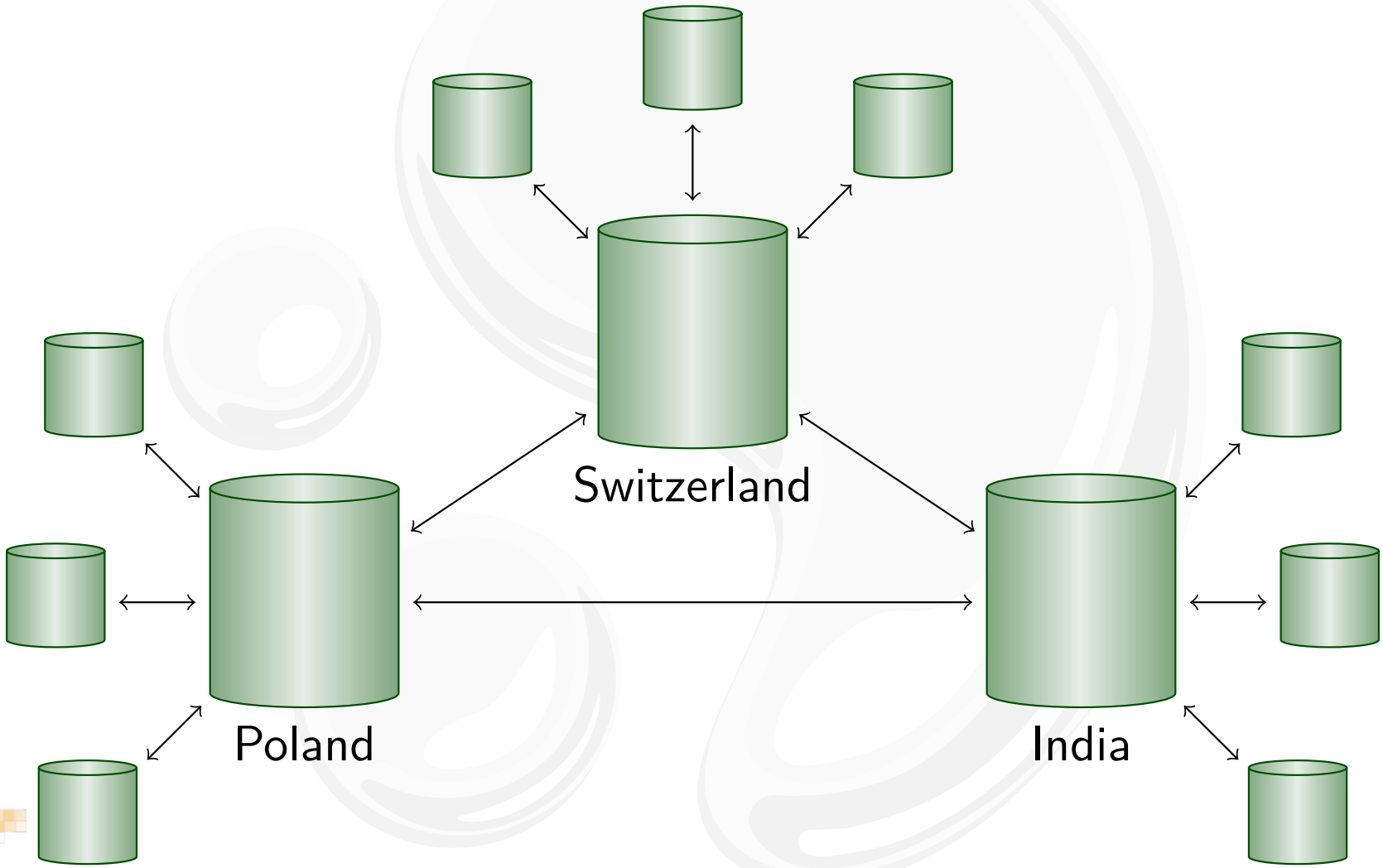
Workflow in a Team

Mercurial scales from a single team. . . :



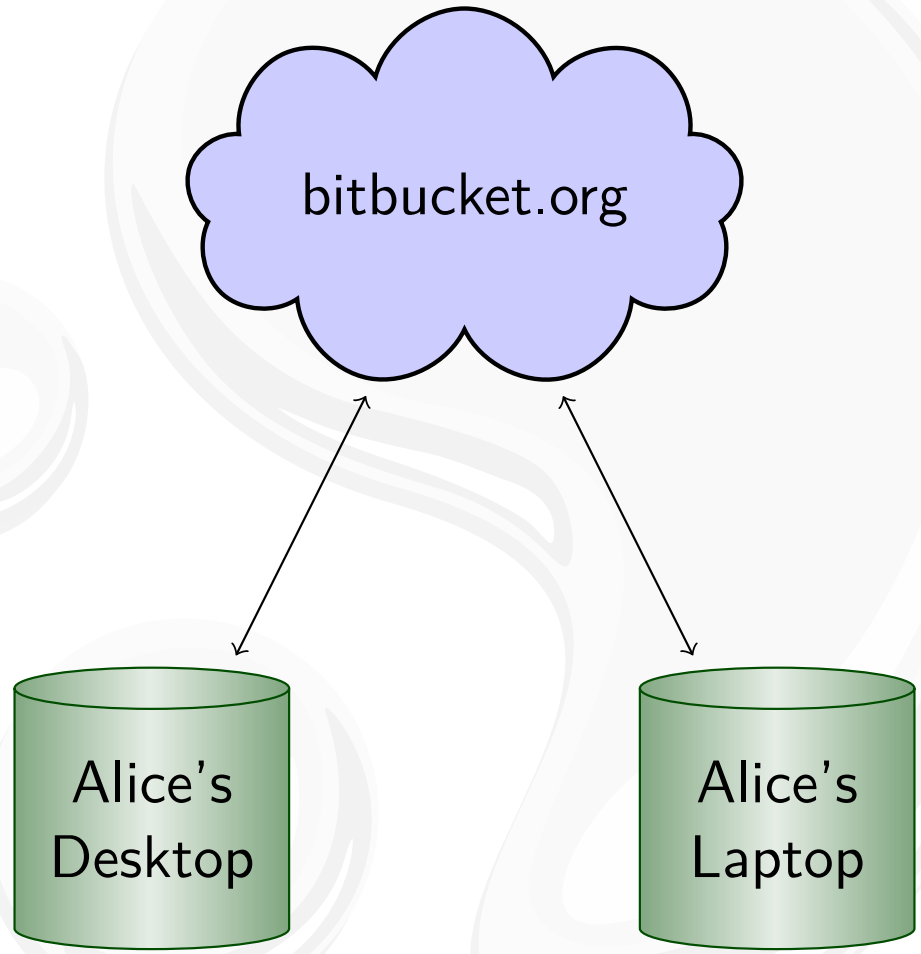
Workflow Between Company Divisions

. . . to enterprise-wide development. . . :

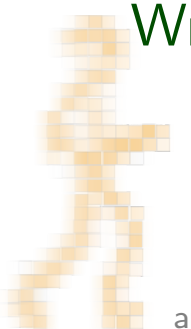


Workflow Between Two Computers

. . . to working with yourself:

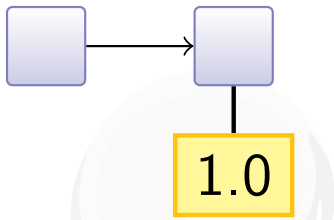


Outline

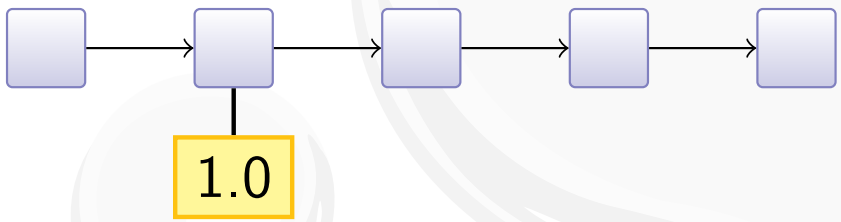


Release Branches

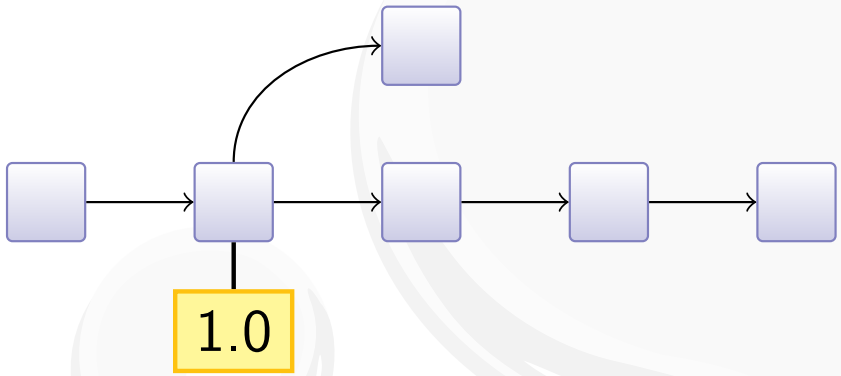
Release Branches



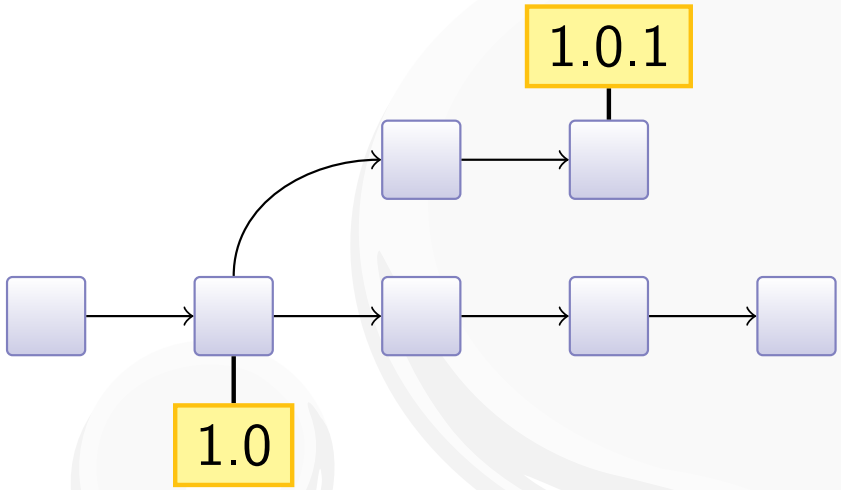
Release Branches



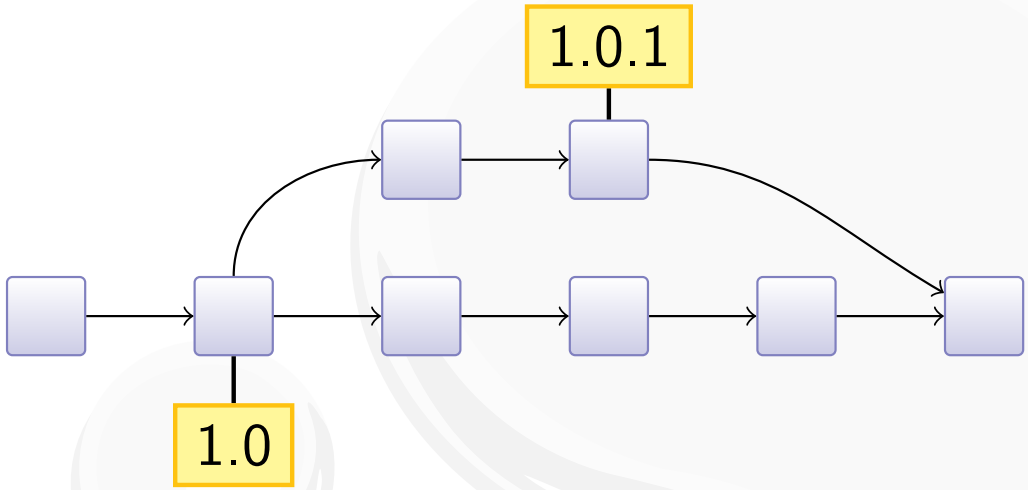
Release Branches



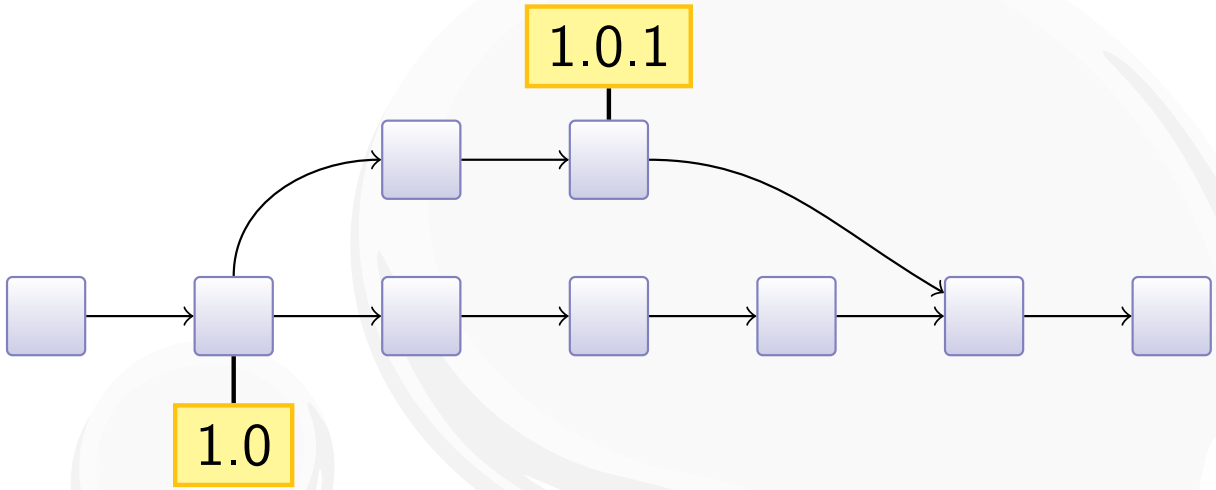
Release Branches



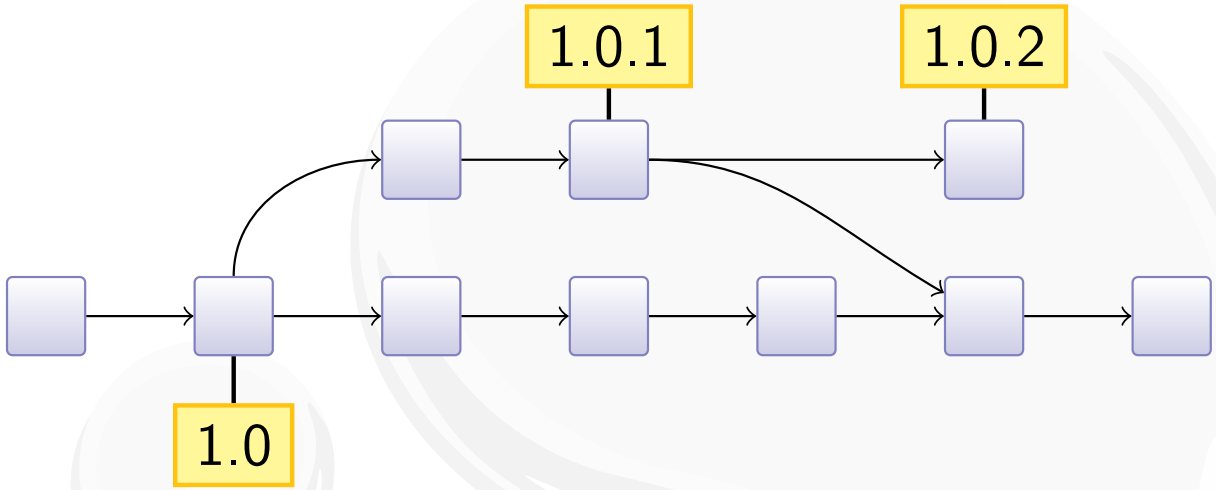
Release Branches



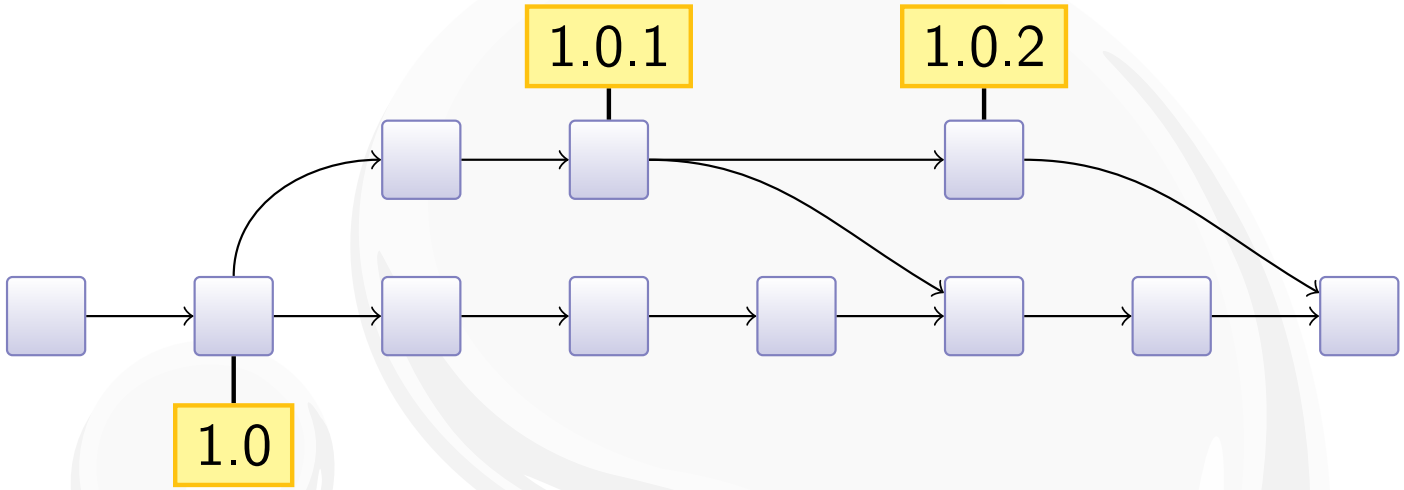
Release Branches



Release Branches



Release Branches



Outline

Bundling Third-Party Code

Storing third-party code in your repository is common:

- easy setup, developers need just one checkout

```
application/
-- src/
-- doc/
-- lib/
   -- libfoo/
      -- ...
   -- libbar/
      -- ...
```

- everybody has the same library versions
  - uniform across platforms
  - stable target for your application
- bugs can be fixed right away
  - no need to wait for a new upstream release

The Library Maintenance Problem

The situation:

- your application bundles `libfoo` version 1.0
- you discover and fix a bug in `libfoo`
- later, `libfoo` version 2.0 is released
- what now?

The Library Maintenance Problem

The situation:

- your application bundles `libfoo` version 1.0
- you discover and fix a bug in `libfoo`
- later, `libfoo` version 2.0 is released
- what now?

The problem:

- there might be many files in `libfoo`
- you might have changed several of them
- your changes were spread over many commits

The Library Maintenance Problem

The situation:

- your application bundles `libfoo` version 1.0
- you discover and fix a bug in `libfoo`
- later, `libfoo` version 2.0 is released
- what now?

The problem:

- there might be many files in `libfoo`
- you might have changed several of them
- your changes were spread over many commits
- you might have changed `libfoo` when you imported it

The Library Maintenance Problem

The situation:

- your application bundles `libfoo` version 1.0
- you discover and fix a bug in `libfoo`
- later, `libfoo` version 2.0 is released
- what now?

The problem:

- there might be many files in `libfoo`
- you might have changed several of them
- your changes were spread over many commits
- you might have changed `libfoo` when you imported it
- files were renamed in `libfoo` 2.0!

Handling Library Upgrades

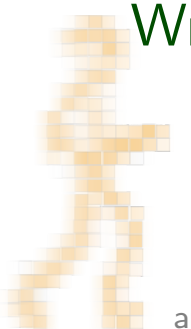
The goal:

- make the same fixes to `libfoo` 2.0 as you did to 1.0

Vendor branches help you here:

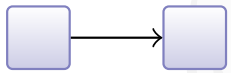
- gives you clear distinction between
  - changes made by the vendor
  - your own changes

Outline



Vendor Branches in Mercurial

High-level view of vendor branches:

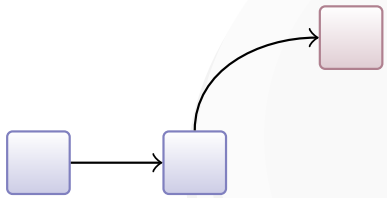
default: 

Vendor Branches in Mercurial

High-level view of vendor branches:

libfoo:

default:

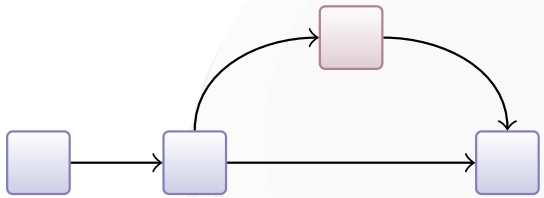


Vendor Branches in Mercurial

High-level view of vendor branches:

libfoo:

default:

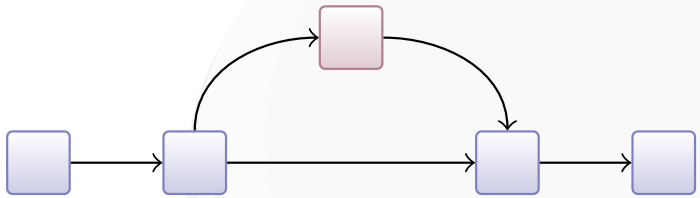


Vendor Branches in Mercurial

High-level view of vendor branches:

libfoo:

default:

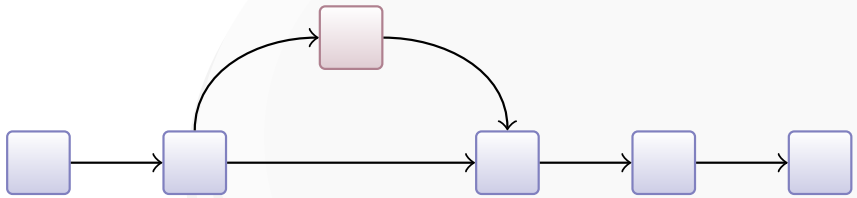


Vendor Branches in Mercurial

High-level view of vendor branches:

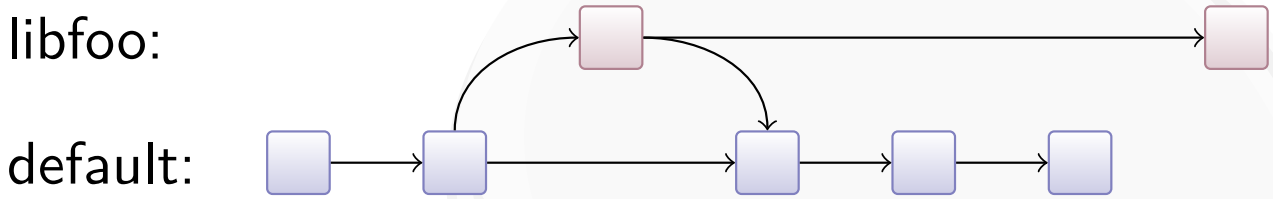
libfoo:

default:



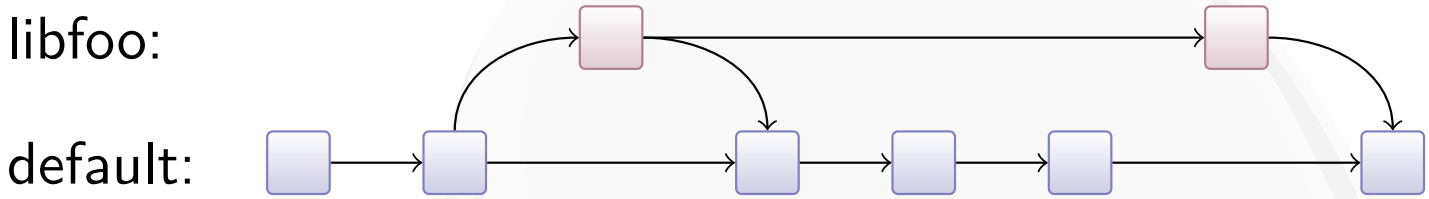
Vendor Branches in Mercurial

High-level view of vendor branches:



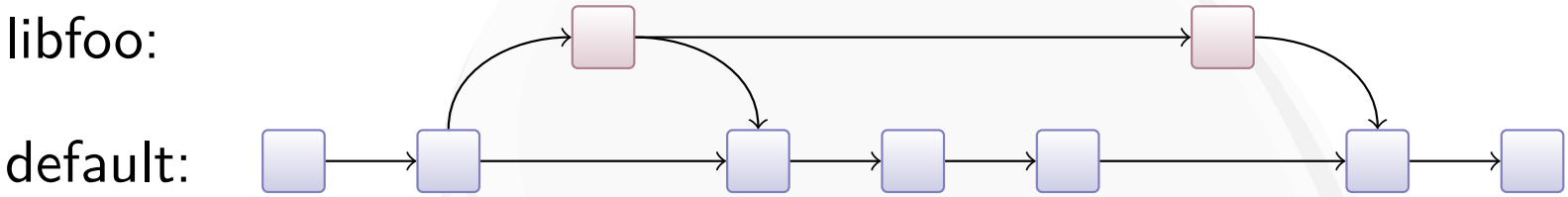
Vendor Branches in Mercurial

High-level view of vendor branches:



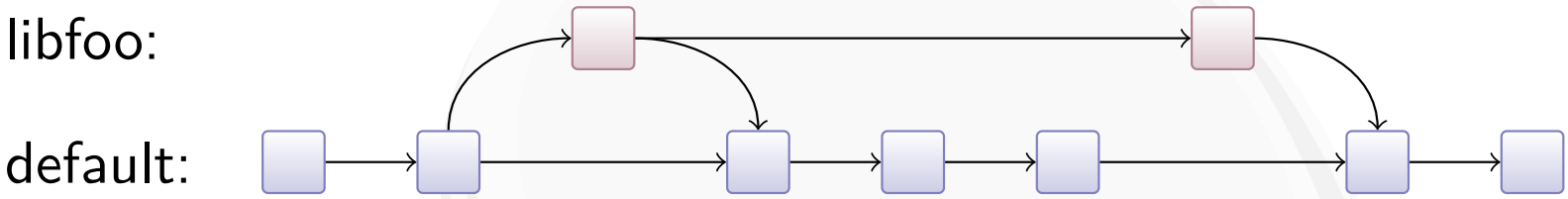
Vendor Branches in Mercurial

High-level view of vendor branches:



Vendor Branches in Mercurial

High-level view of vendor branches:

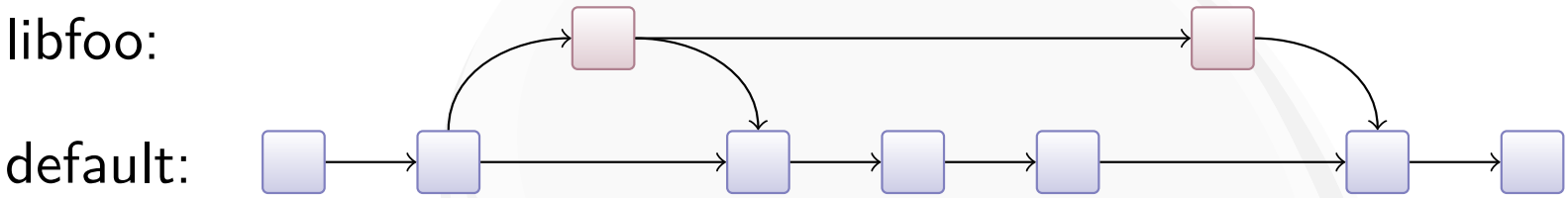


This workflow lets you:

- clearly distinguish between upstream code and your code
- directly modify libraries in your codebase

Vendor Branches in Mercurial

High-level view of vendor branches:



This workflow lets you:

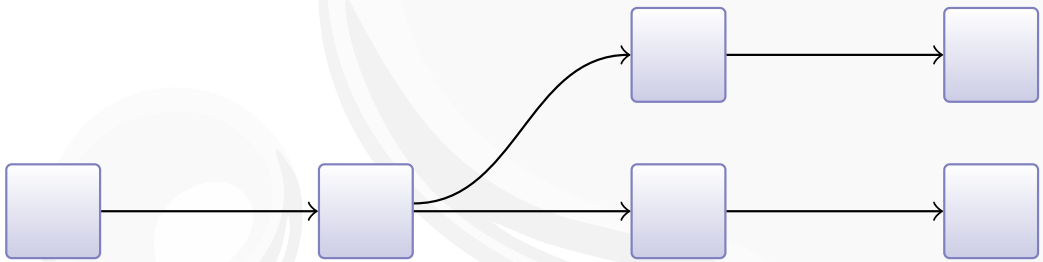
- clearly distinguish between upstream code and your code
- directly modify libraries in your codebase
- Mercurial knows exactly what to merge

What Happens in a Merge?

Or: Why does distributed revision control work?

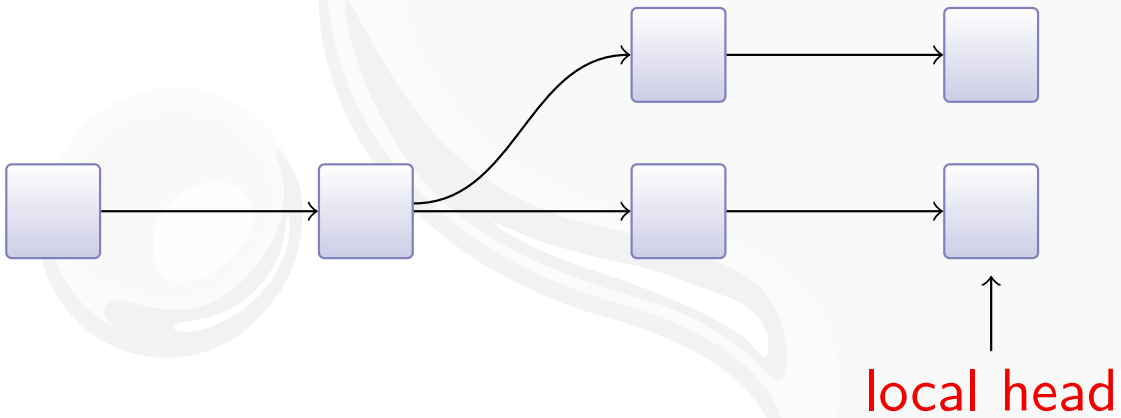
What Happens in a Merge?

Or: Why does distributed revision control work?



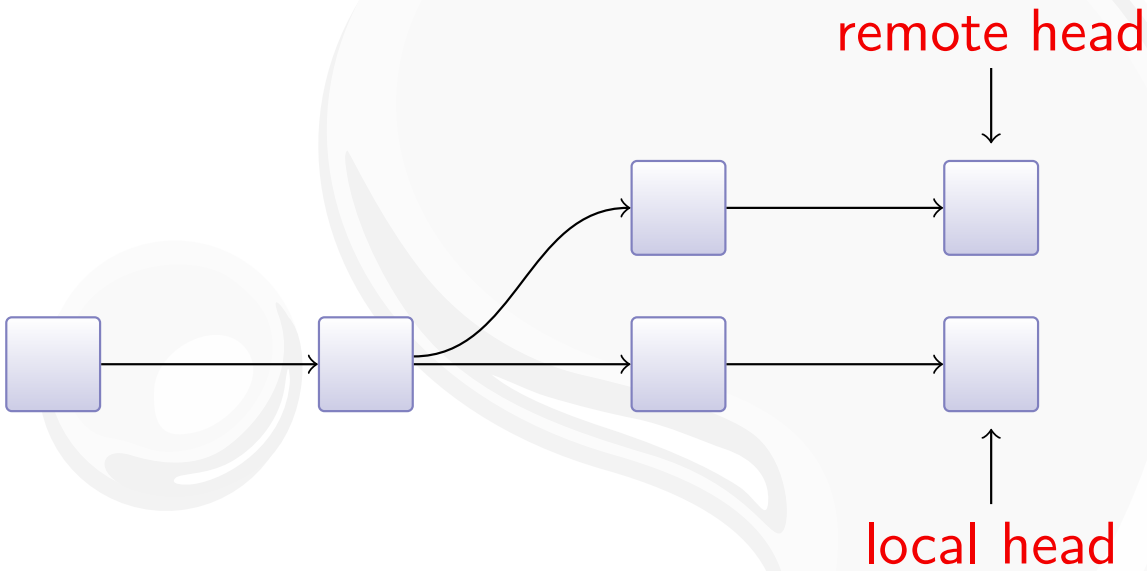
What Happens in a Merge?

Or: Why does distributed revision control work?



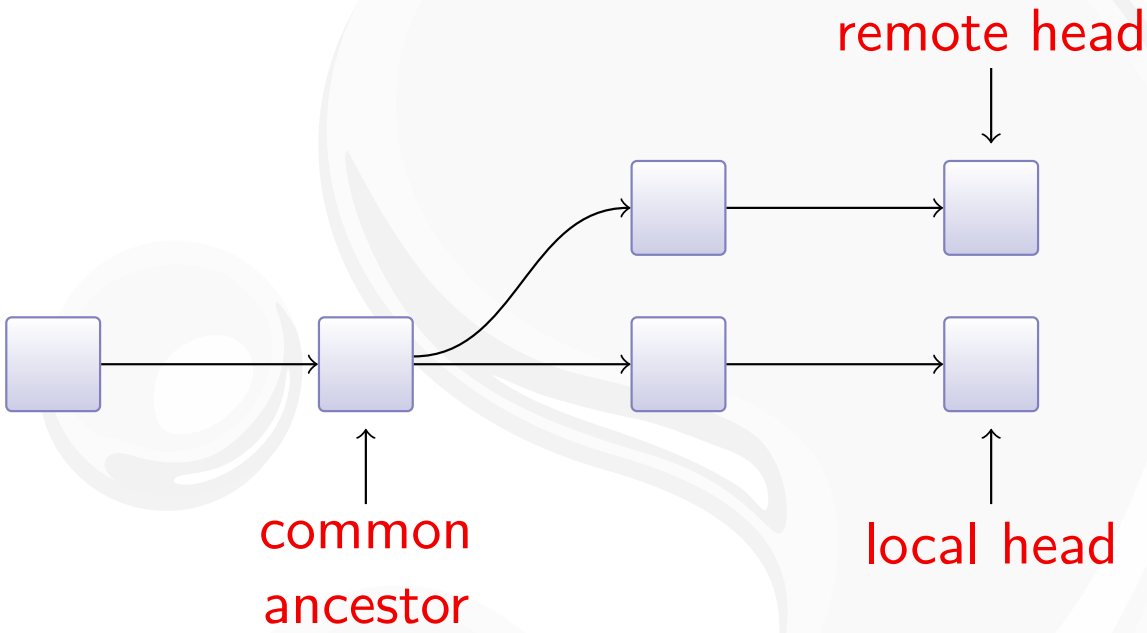
What Happens in a Merge?

Or: Why does distributed revision control work?



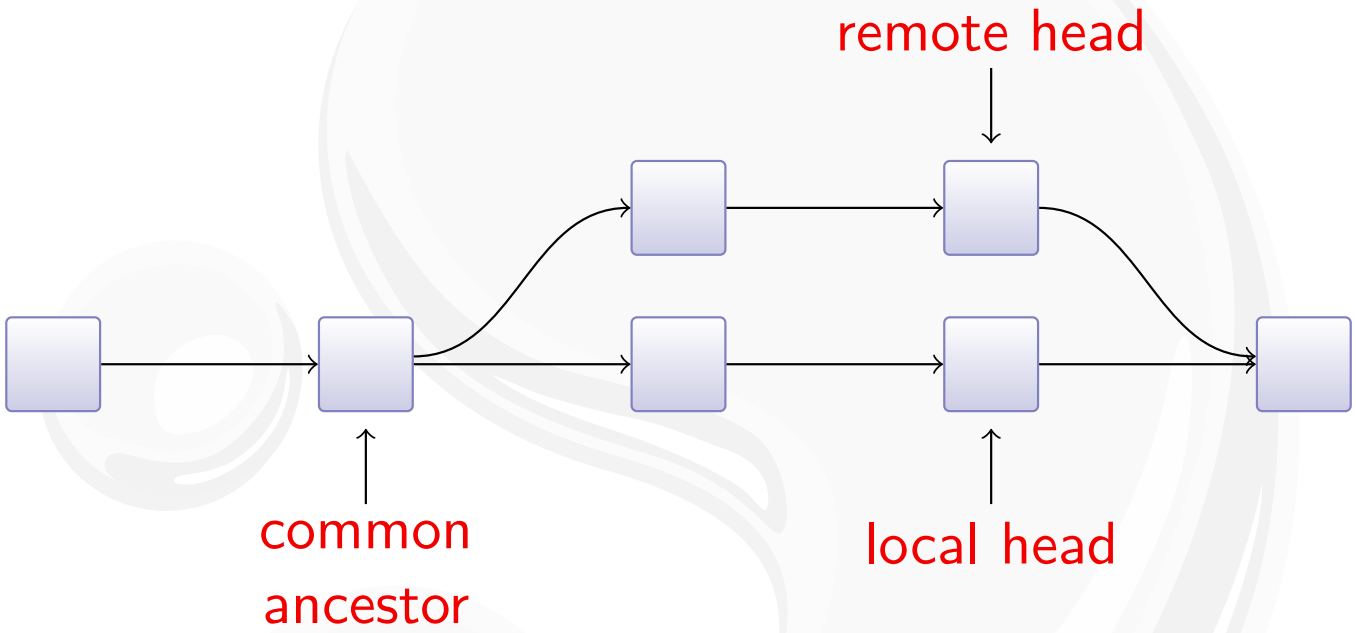
What Happens in a Merge?

Or: Why does distributed revision control work?



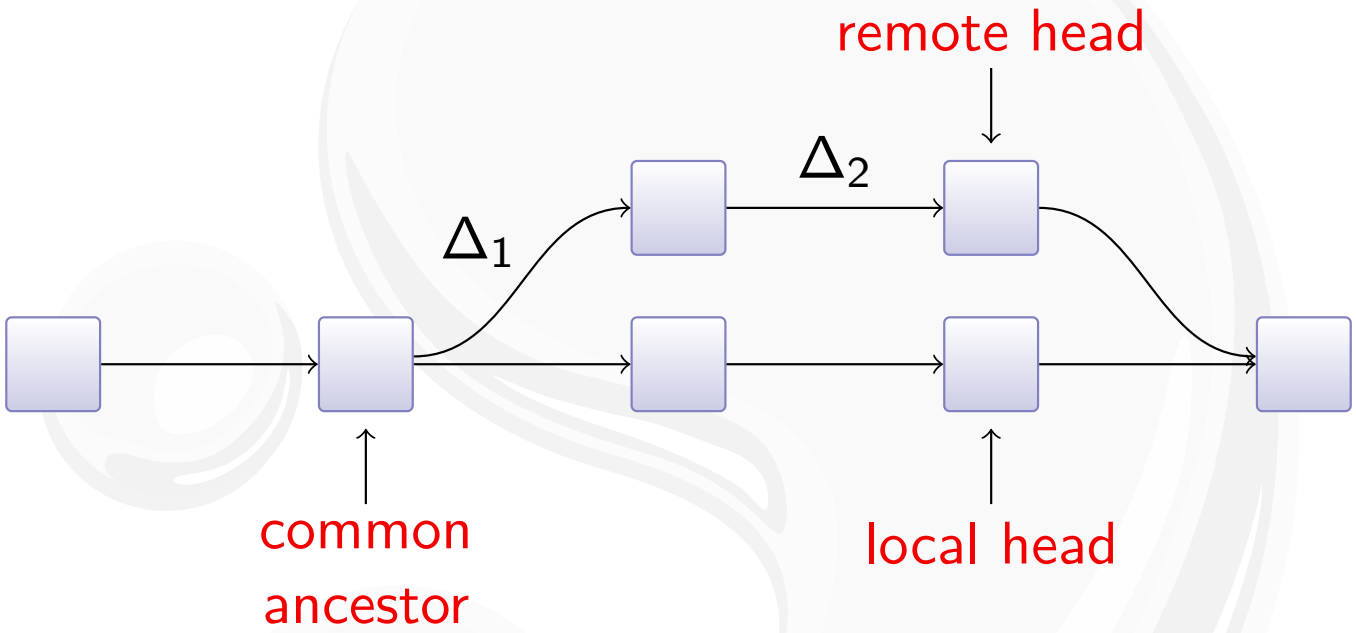
What Happens in a Merge?

Or: Why does distributed revision control work?



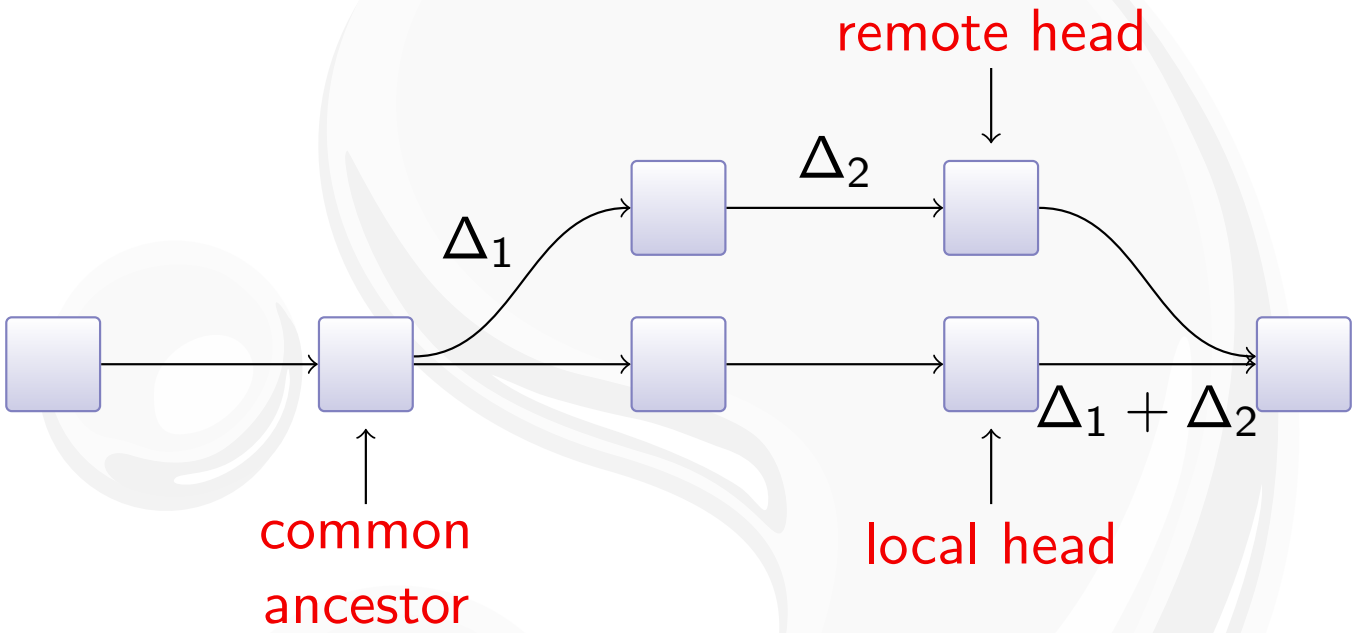
What Happens in a Merge?

Or: Why does distributed revision control work?



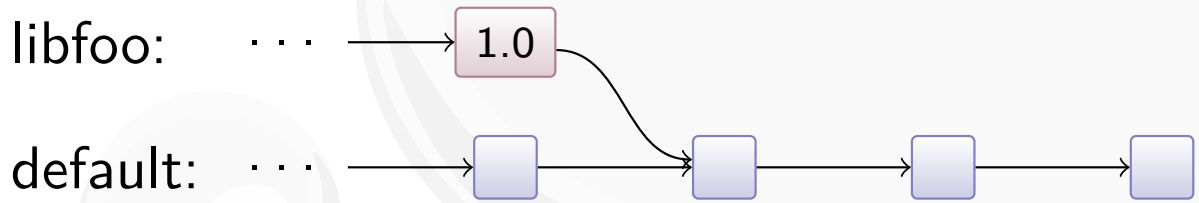
What Happens in a Merge?

Or: Why does distributed revision control work?



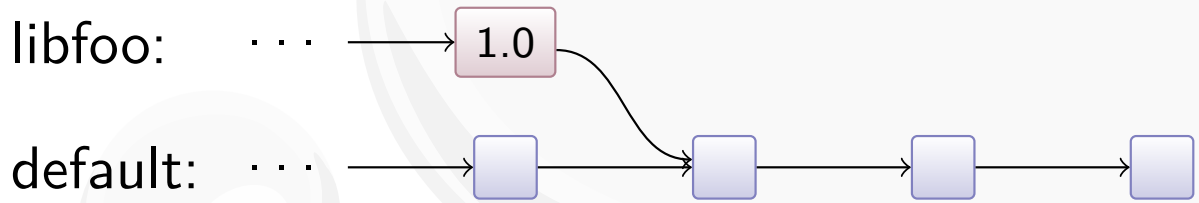
Merging Vendor Branches

Upgrading from `libfoo` version 1.0 to version 2.0:



Merging Vendor Branches

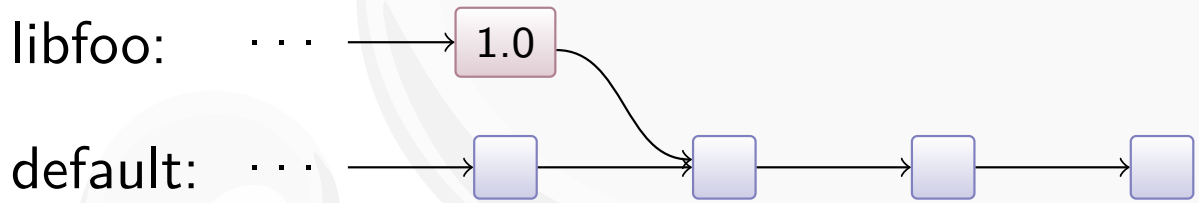
Upgrading from `libfoo` version 1.0 to version 2.0:



- `hg update libfoo`

Merging Vendor Branches

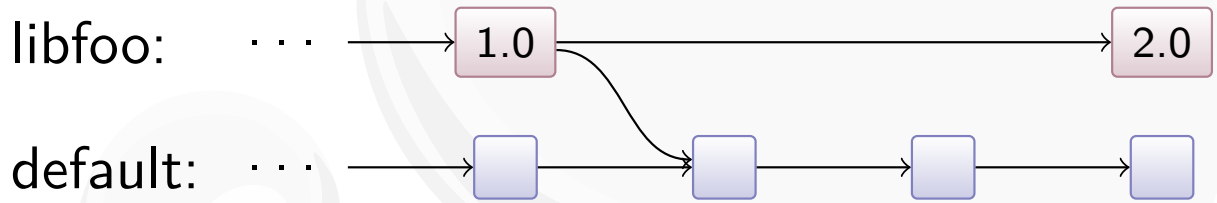
Upgrading from `libfoo` version 1.0 to version 2.0:



- `hg update libfoo`
- unpack and import `libfoo` version 2.0

Merging Vendor Branches

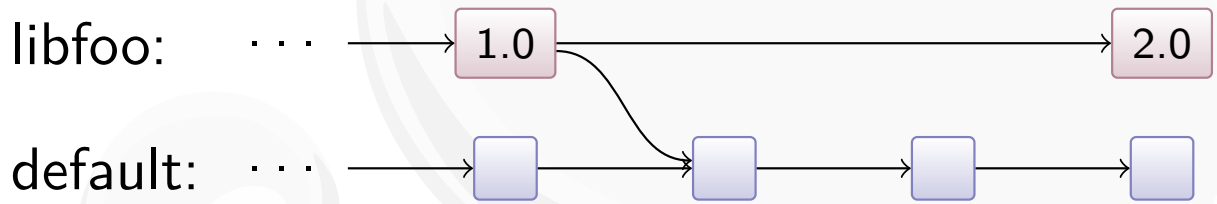
Upgrading from `libfoo` version 1.0 to version 2.0:



- `hg update libfoo`
- unpack and import `libfoo` version 2.0
- `hg commit -m 'Import of libfoo 2.0'`

Merging Vendor Branches

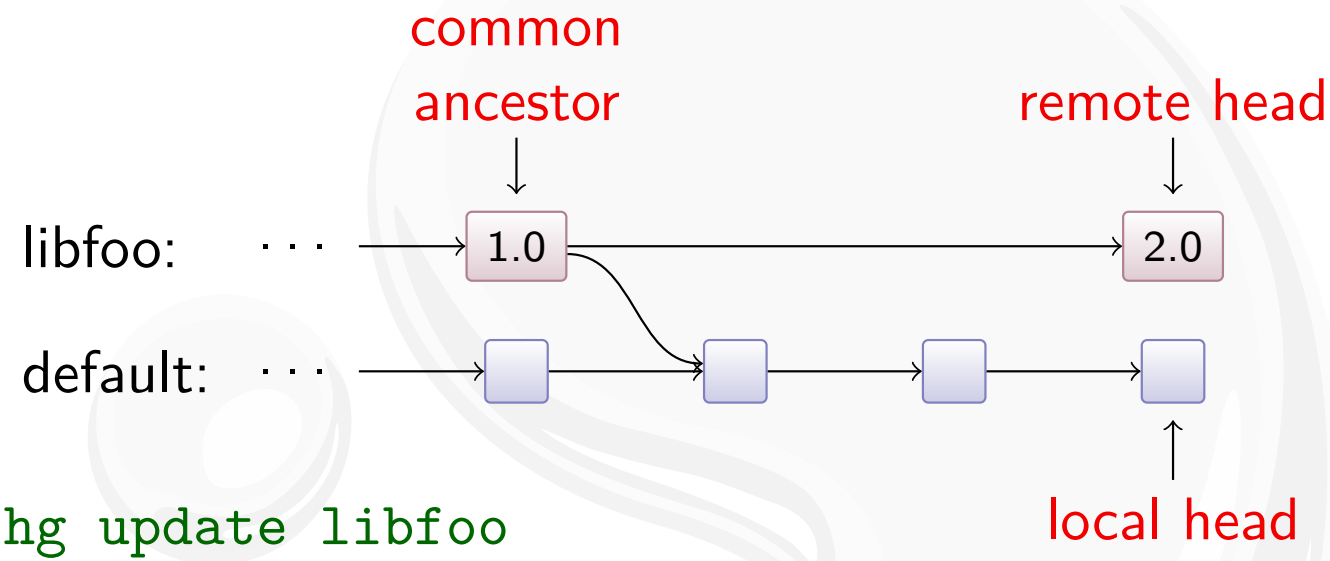
Upgrading from `libfoo` version 1.0 to version 2.0:



- `hg update libfoo`
- unpack and import `libfoo` version 2.0
- `hg commit -m 'Import of libfoo 2.0'`
- `hg update default`

Merging Vendor Branches

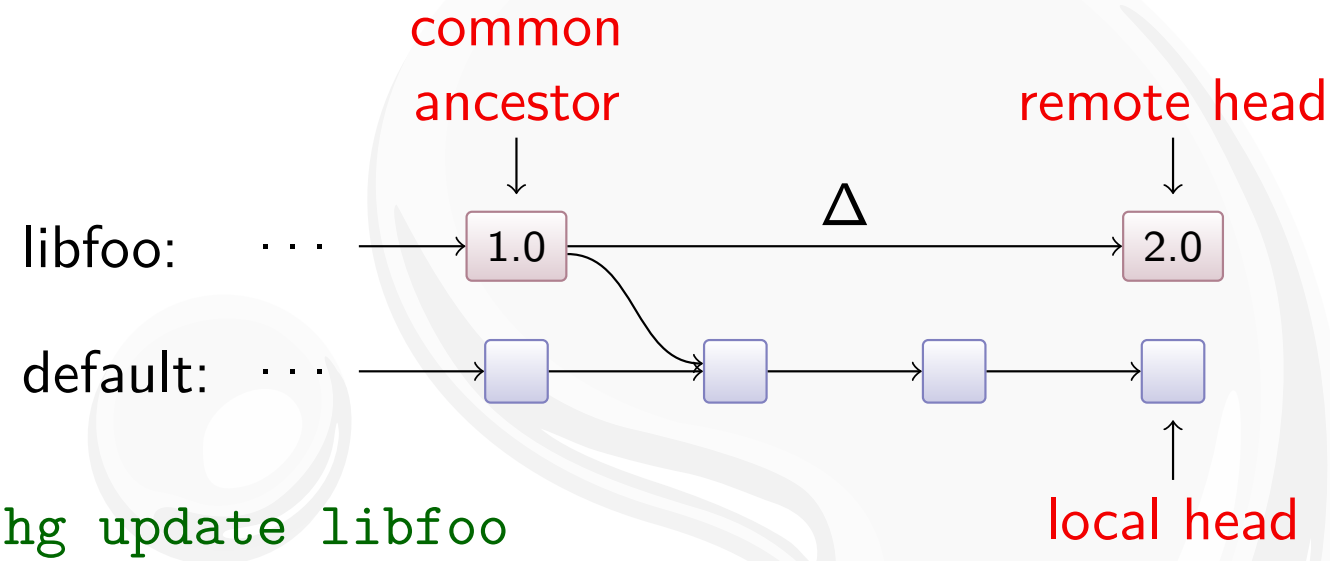
Upgrading from `libfoo` version 1.0 to version 2.0:



- `hg update libfoo`
- unpack and import `libfoo` version 2.0
- `hg commit -m 'Import of libfoo 2.0'`
- `hg update default`

Merging Vendor Branches

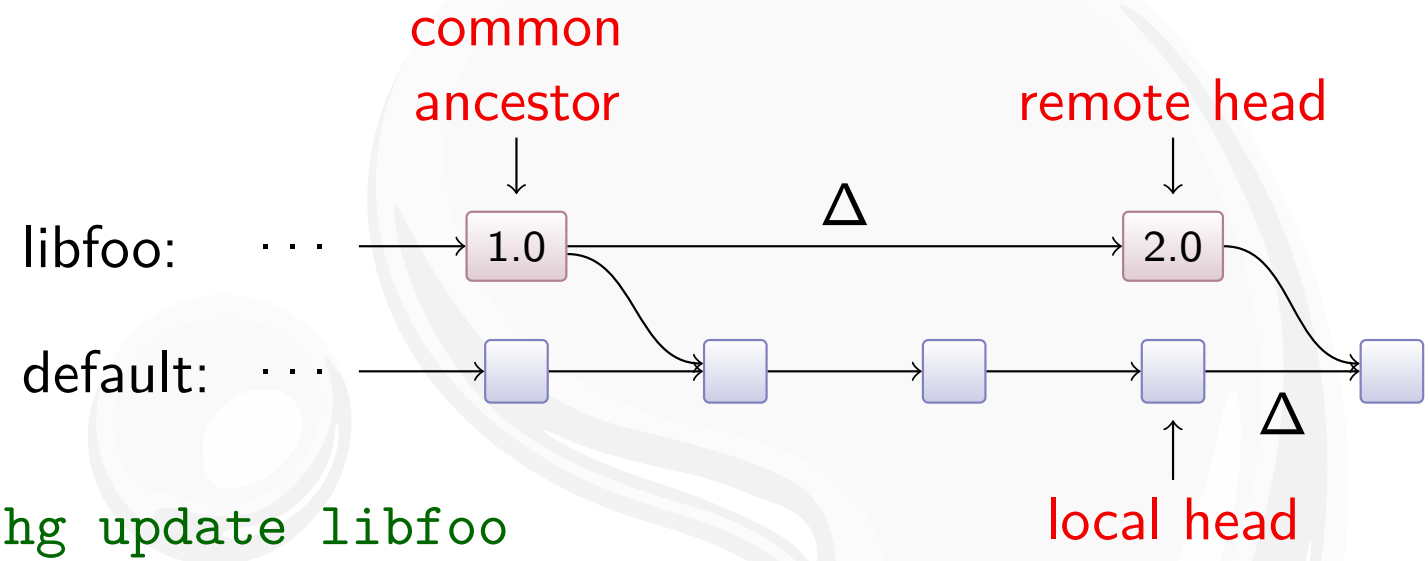
Upgrading from `libfoo` version 1.0 to version 2.0:



- `hg update libfoo`
- unpack and import `libfoo` version 2.0
- `hg commit -m 'Import of libfoo 2.0'`
- `hg update default`
- `hg merge libfoo`

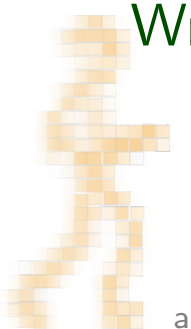
Merging Vendor Branches

Upgrading from `libfoo` version 1.0 to version 2.0:



- `hg update libfoo`
- unpack and import `libfoo` version 2.0
- `hg commit -m 'Import of libfoo 2.0'`
- `hg update default`
- `hg merge libfoo`
- `hg commit -m 'Merged with libfoo 2.0'`

Outline



Importing a Code Drop

Mercurial can help you:

```
$ rm -r lib/libfoo
$ unzip libfoo-2.0.zip -d lib/libfoo
$ hg status
M lib/libfoo/modified.txt
! lib/libfoo/deleted.txt
? lib/libfoo/new.txt
```

Importing a Code Drop

Mercurial can help you:

```
$ rm -r lib/libfoo
$ unzip libfoo-2.0.zip -d lib/libfoo
$ hg status
M lib/libfoo/modified.txt
! lib/libfoo/deleted.txt
? lib/libfoo/new.txt
```

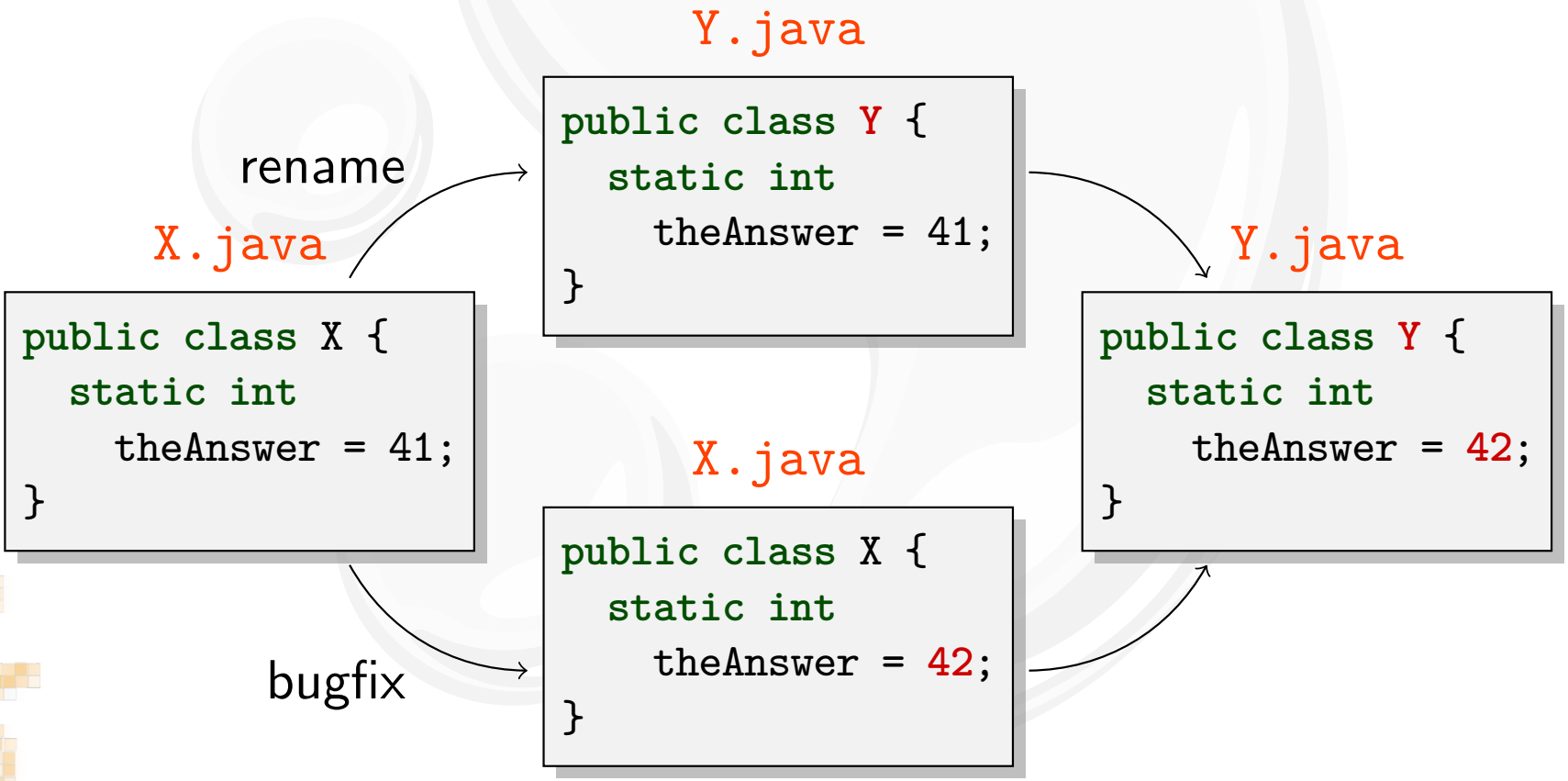
Question: has `deleted.txt` been renamed to `new.txt`?

```
$ hg addremove --similarity 90
removing deleted.txt
adding new.txt
recording removal of deleted.txt as rename to new.txt (94% similar)
```

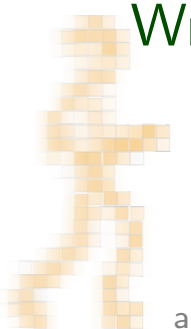
Finding Renamed Files

Tracking renames is important:

- you fix a bug in `X.java` in version 1.0
- version 2.0 now uses `Y.java` instead of `Y.java`
- Mercurial does the right thing with rename tracking



Outline



Conclusion

Mercurial can help manage vendor branches:

- simple workflow
- relies on every-day merge techniques

More Information

- Mercurial homepage:
  http://mercurial.selenic.com/
- *Mercurial: The Definitive Guide*:
  http://hgbook.red-bean.com/
- Getting Started:
  http://mercurial.aragost.com/kick-start/
  http://mercurial.ch/
  http://hginit.com/
- Some free Mercurial hosting sites:
  http://bitbucket.org/
  http://code.google.com/
  http://sourceforge.net/
  http://www.codeplex.com/

Contact

Please get in touch if you have more questions or have considered using Mercurial in your organization:

- Email: `mg@aragost.com`
- IRC: `mg` in `#mercurial` on `irc.freenode.net`

Mercurial Contributors

From `http://ohloh.net/p/mercurial/map`:

Showing 50 of 325 contributors.

Mercurial Contributors

From `http://ohloh.net/p/mercurial/map`:

Showing 50 of 325 contributors.