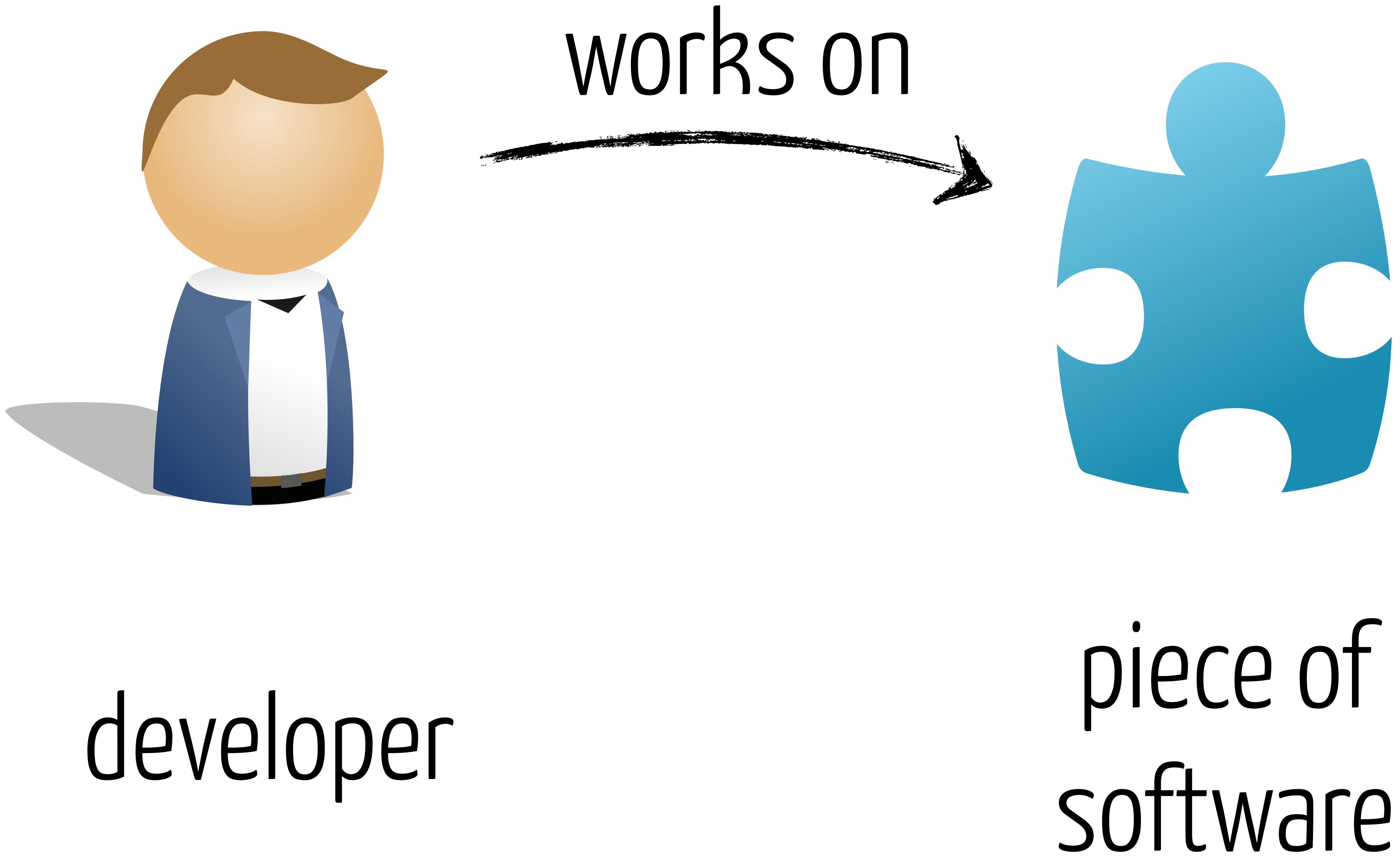
CPSC 310 – Software Engineering

Lecture 5

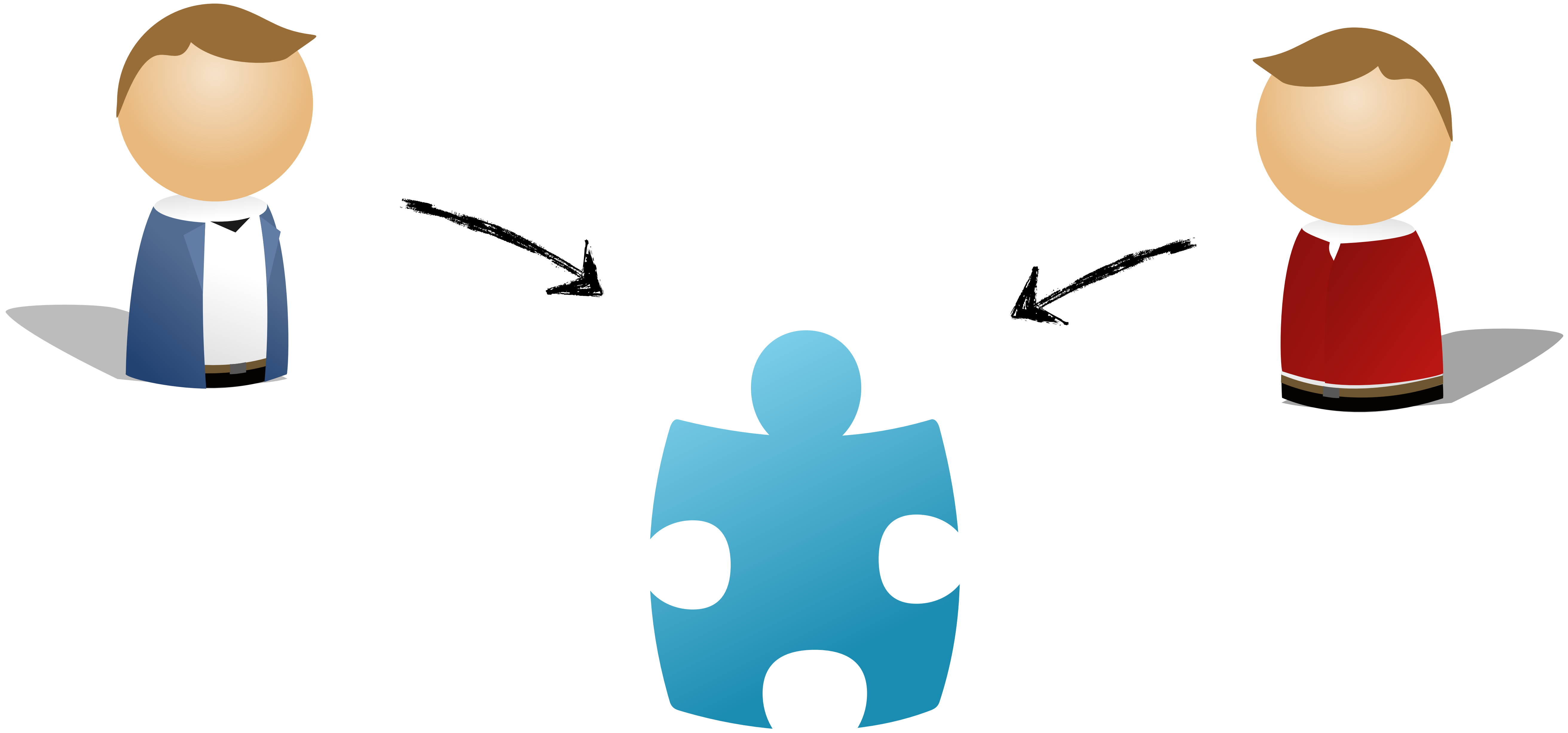
Collaborative Development & Source Code Versioning

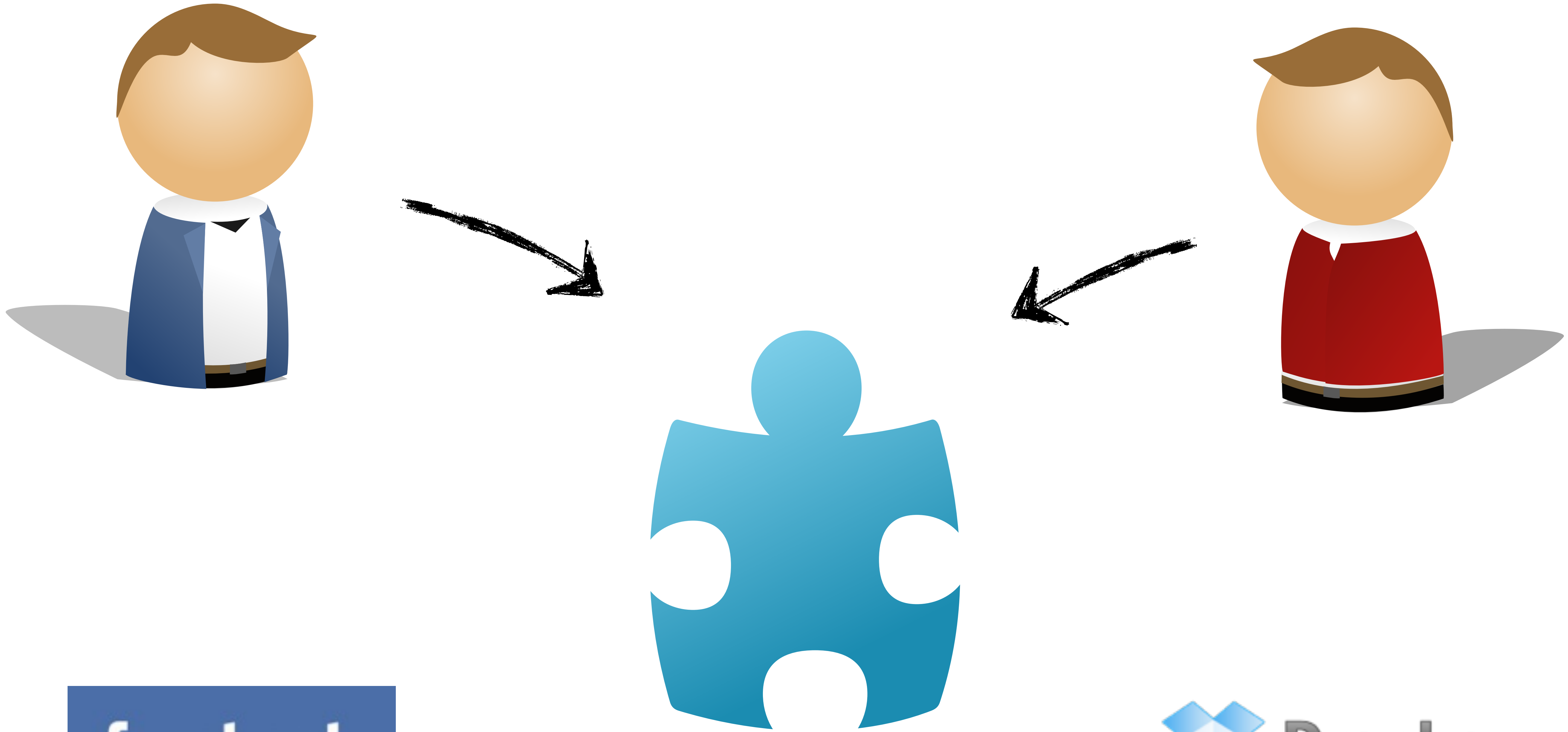
Huge thanks to Sebastien Mosser (sebastien.mosser@unice.fr) for the slides


works on
developer
piece of software

Collaborative

Development

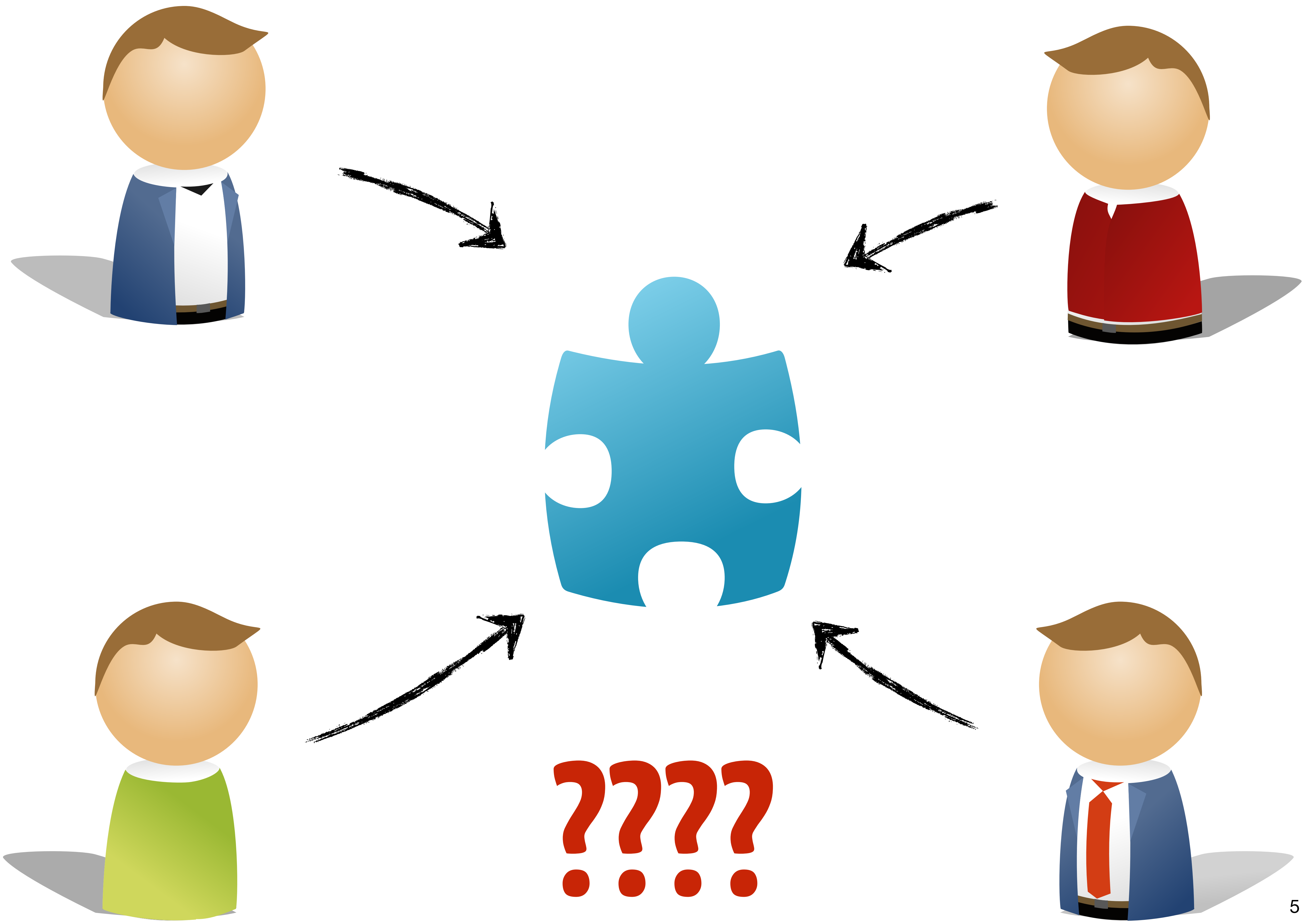


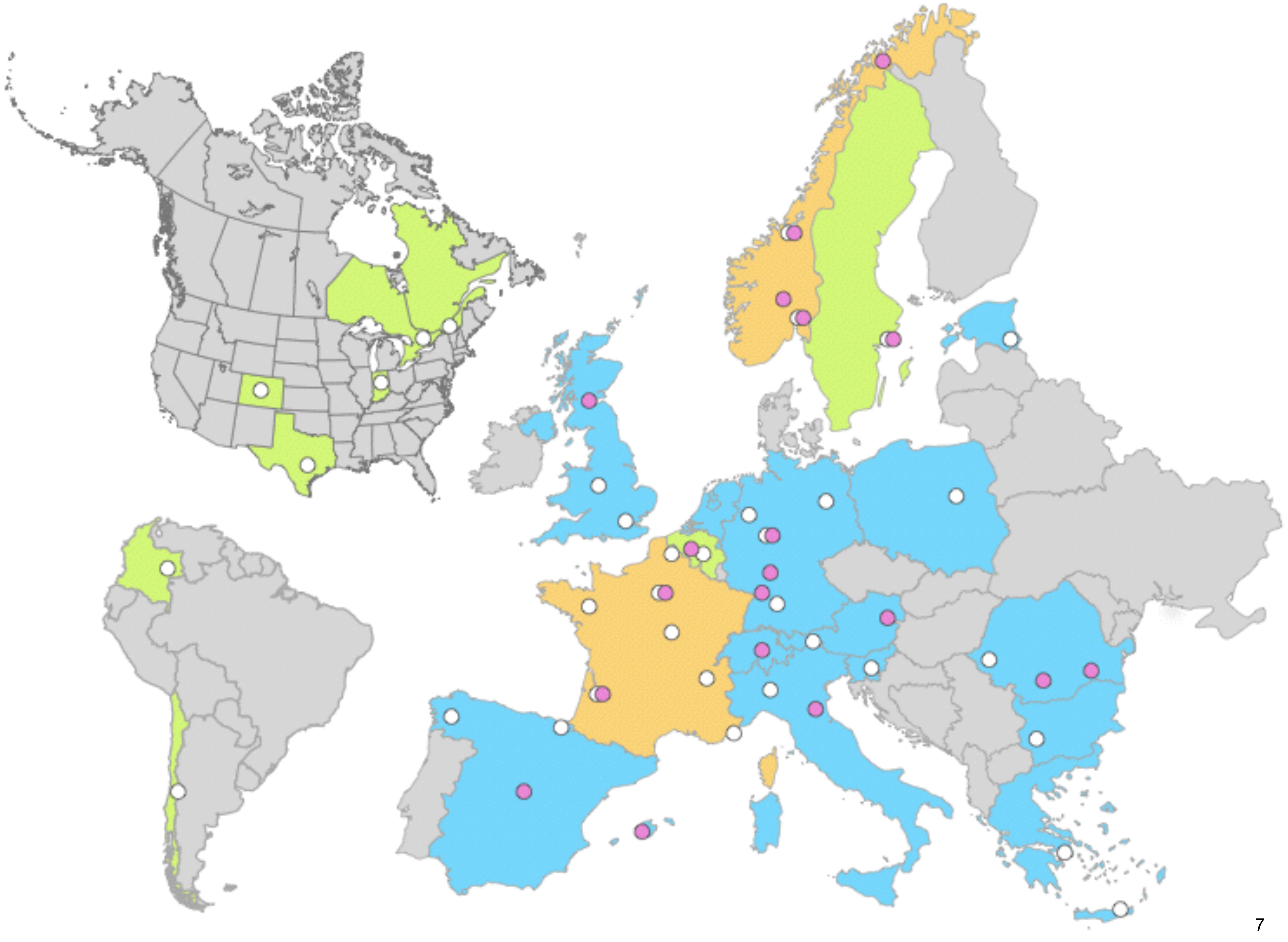


USB Key?

Email?

Shared directory?



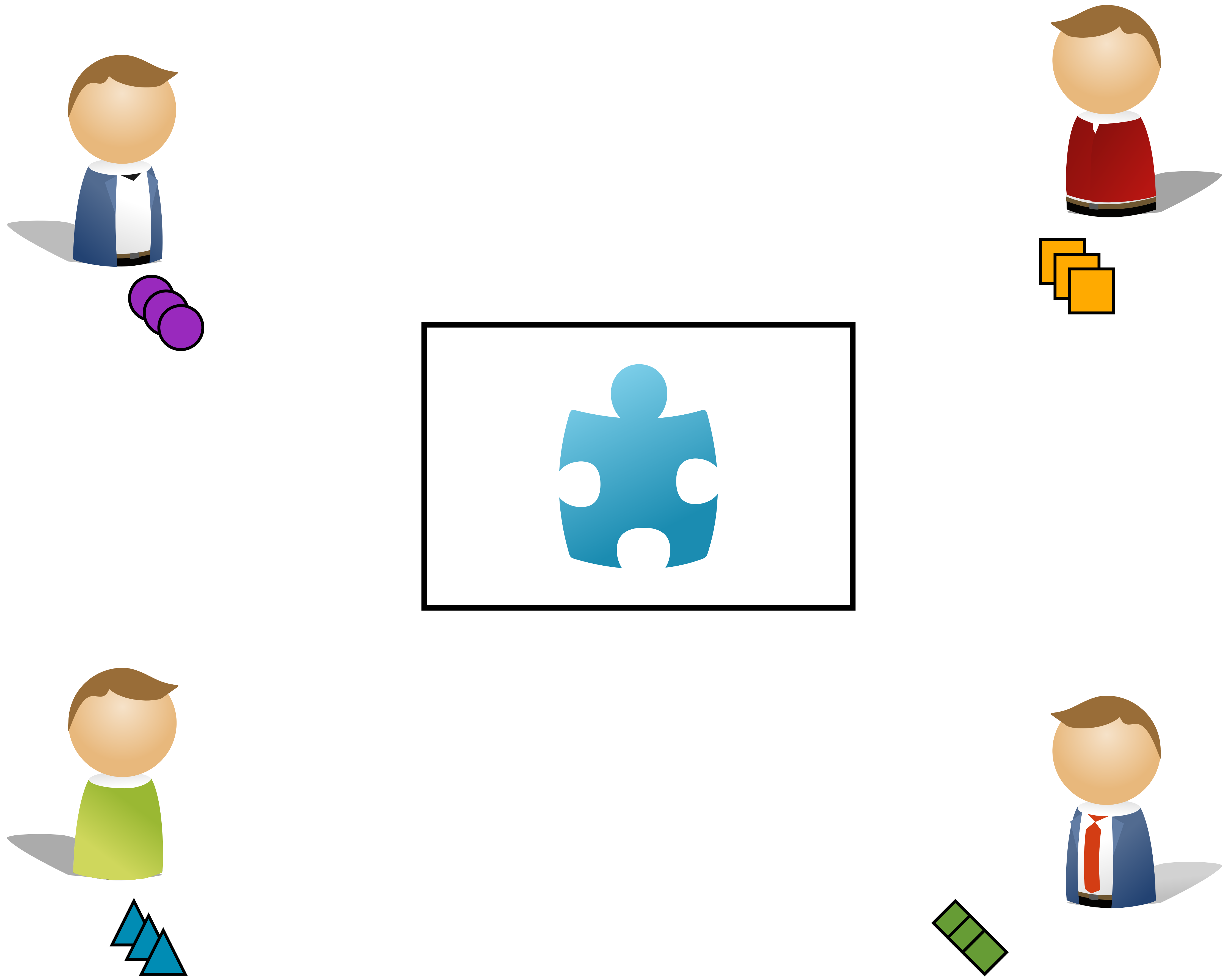


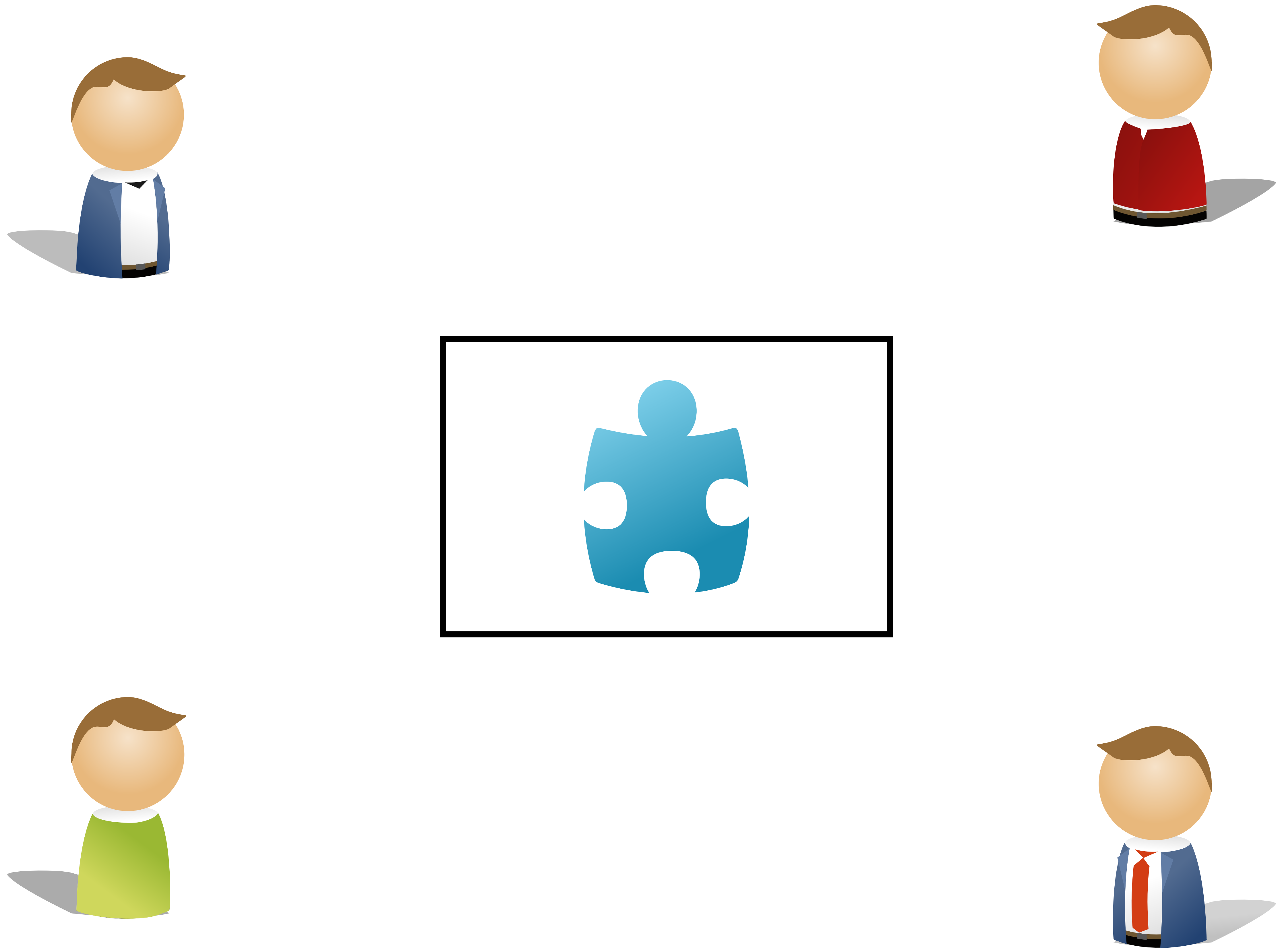
«**Why** do we version source code?»

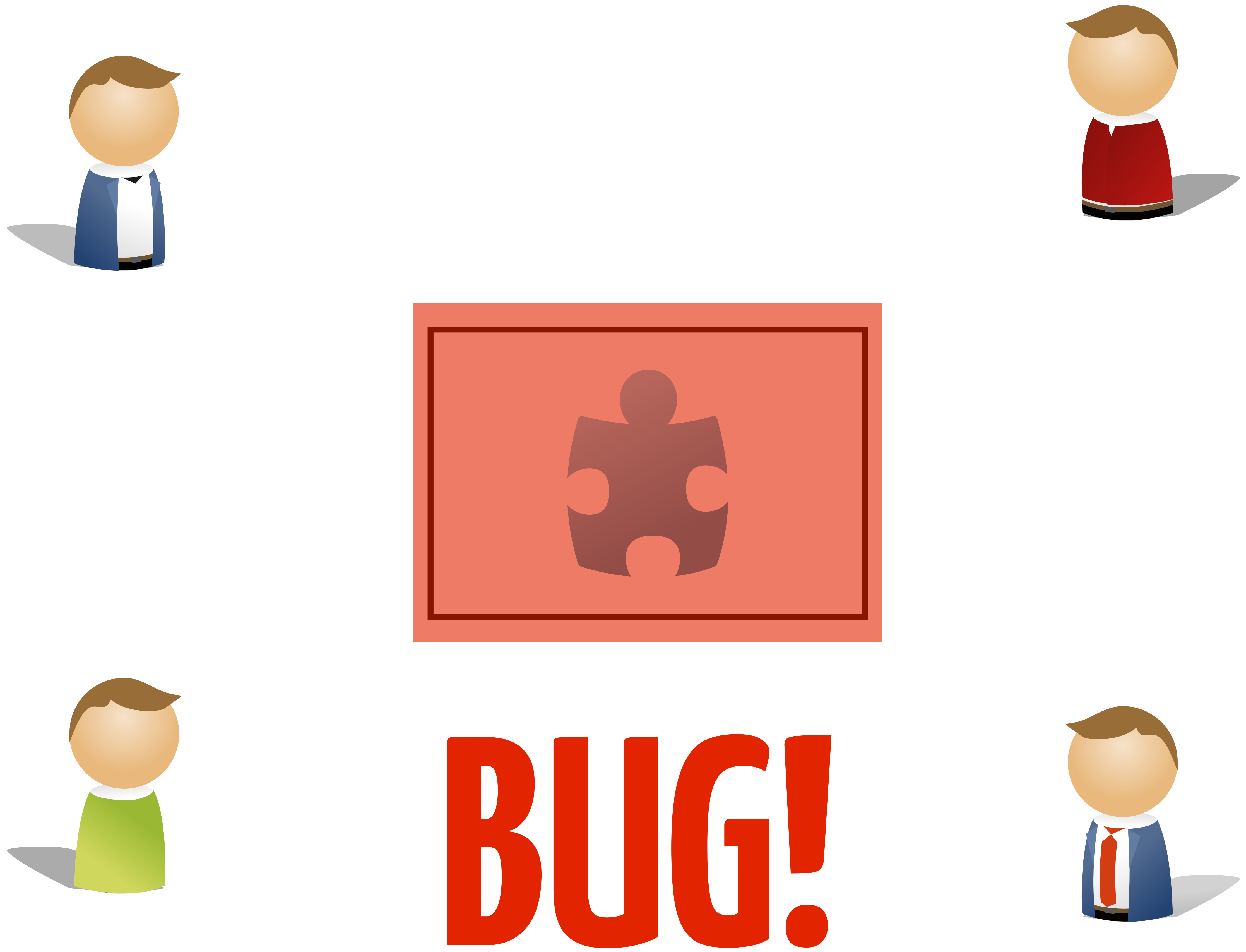
Motivations (among others)

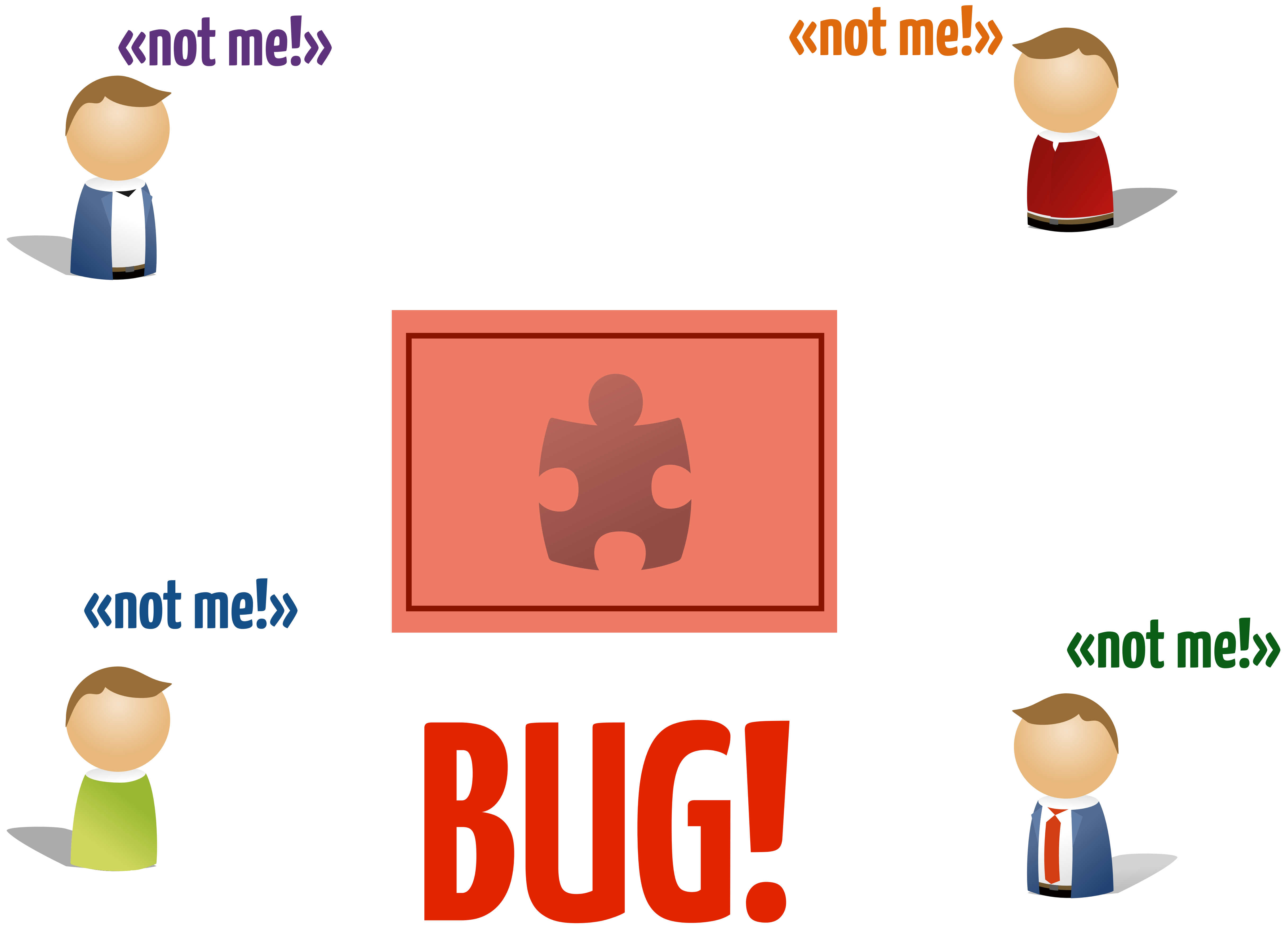
Before

After



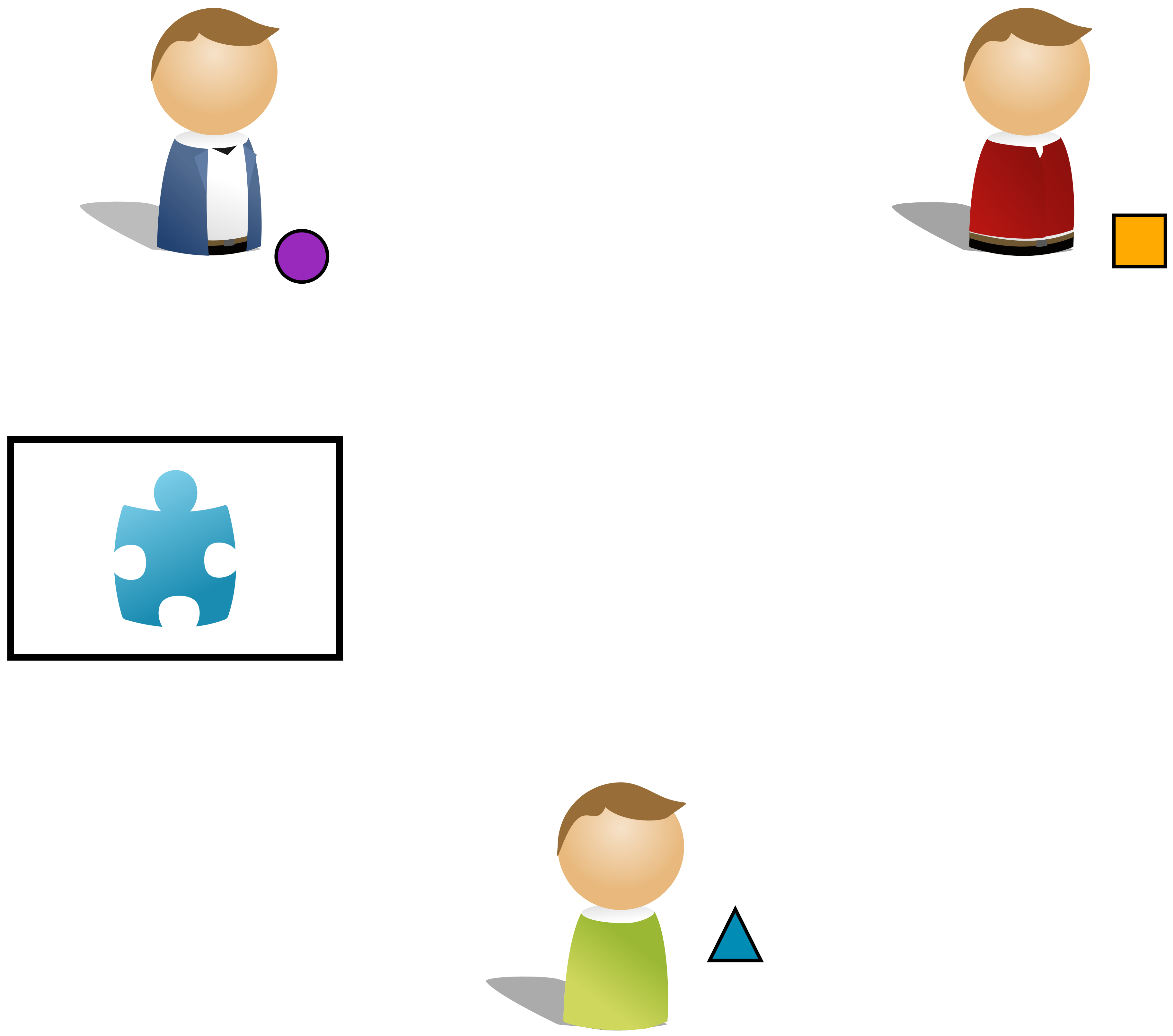


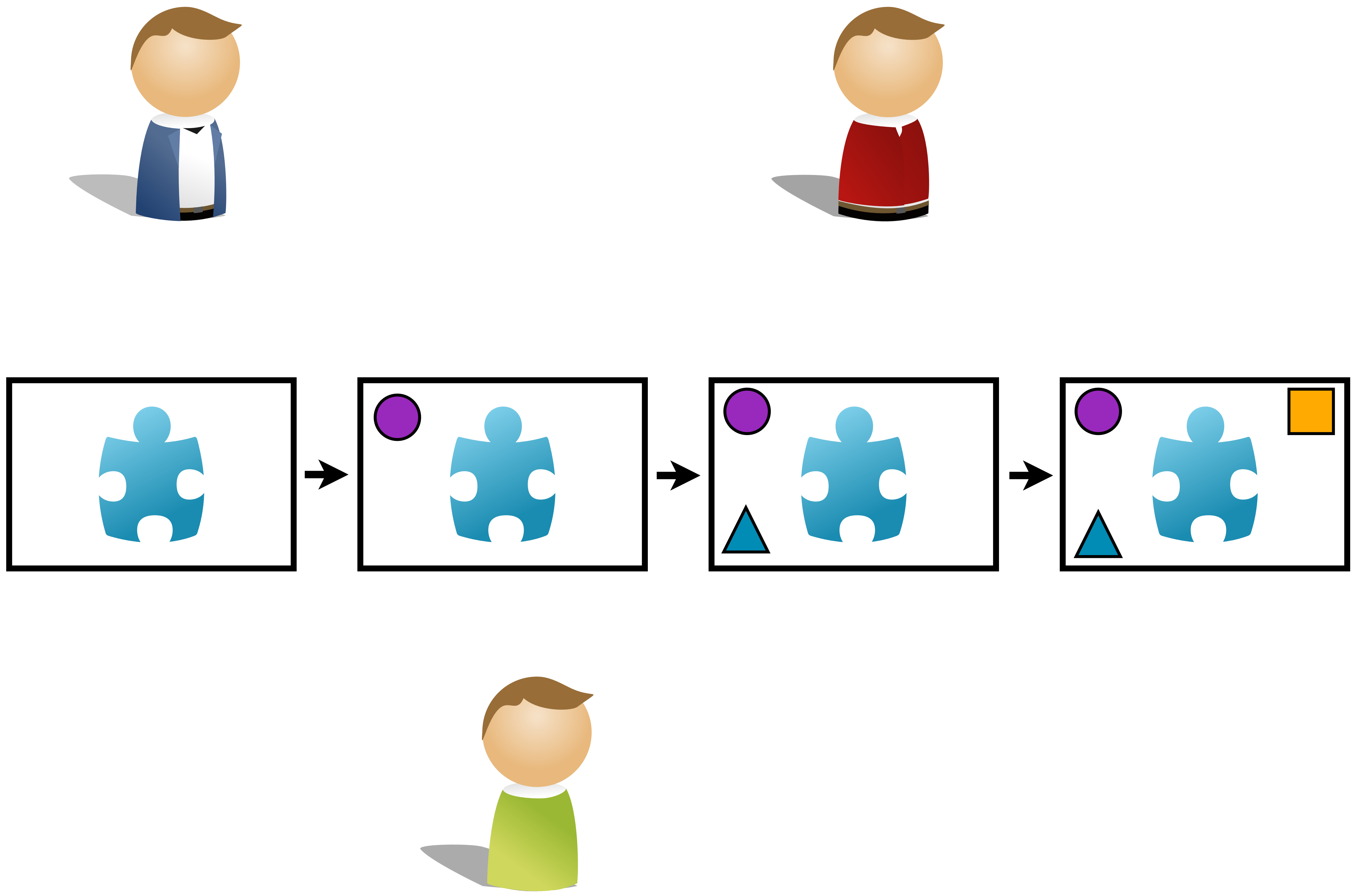

BUG!

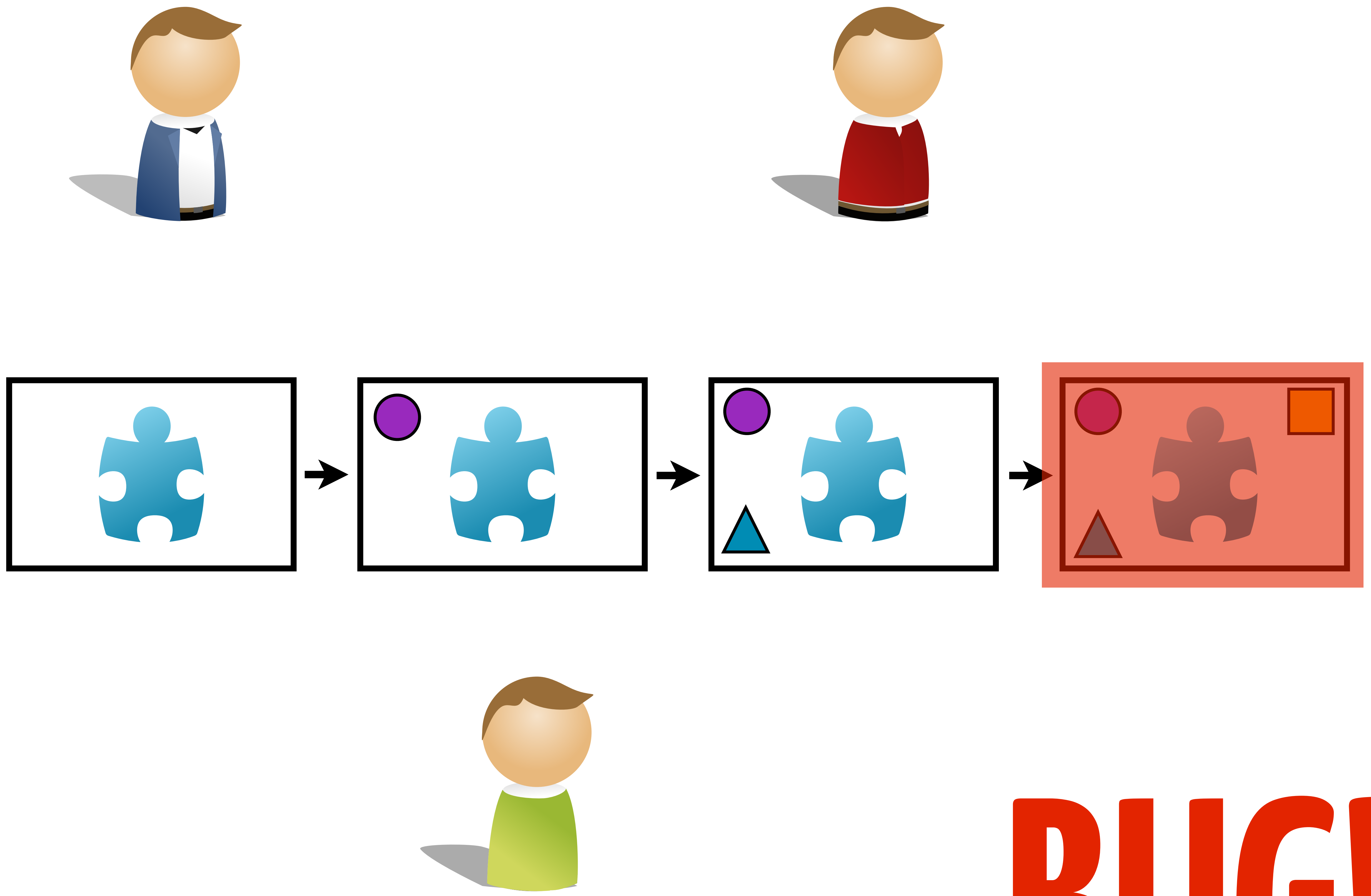

«not me!»
«not me!»
«not me!»
«not me!»
BUG!

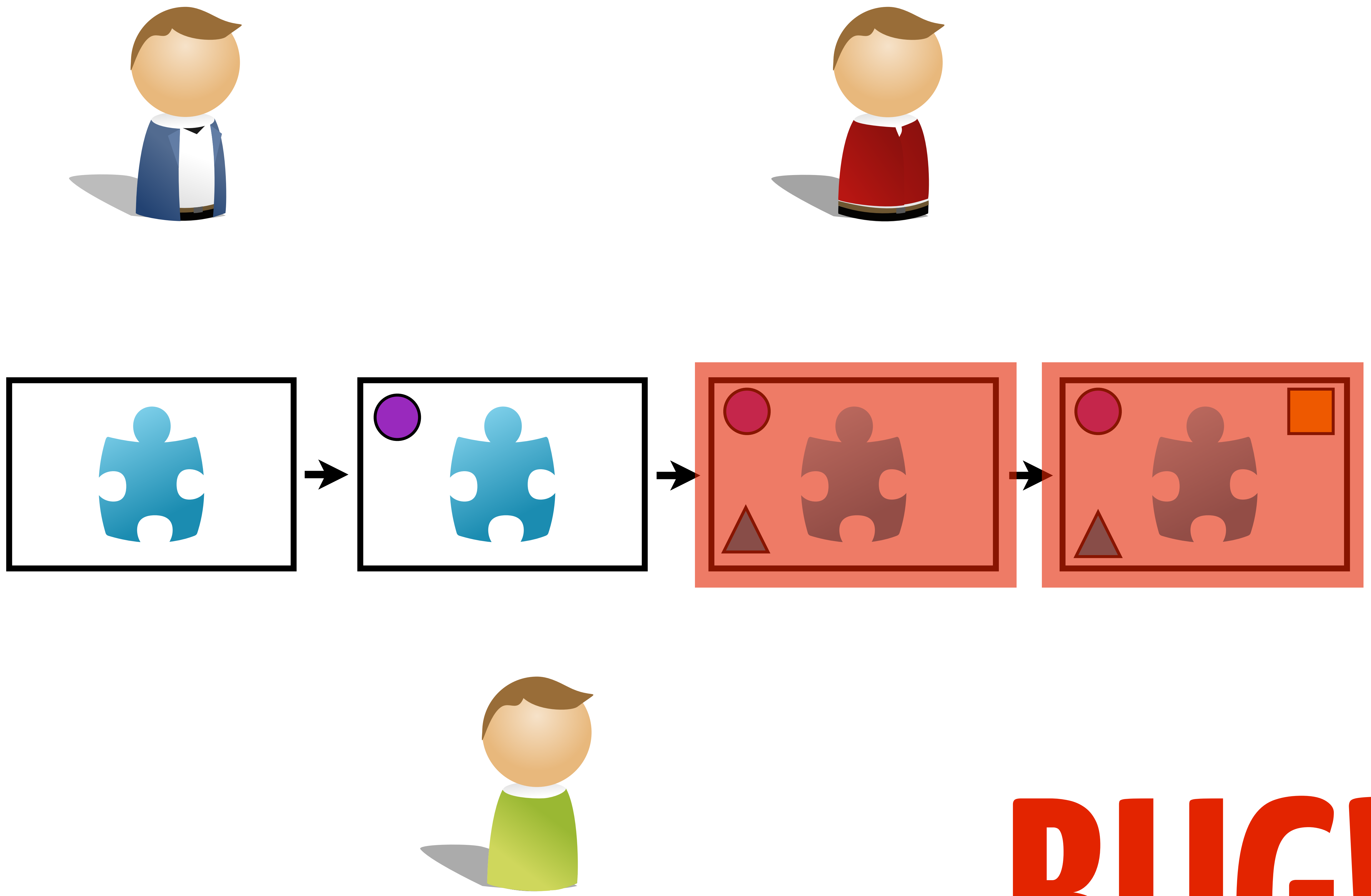
«**Why** do we version source code?»

To **trace** changes!

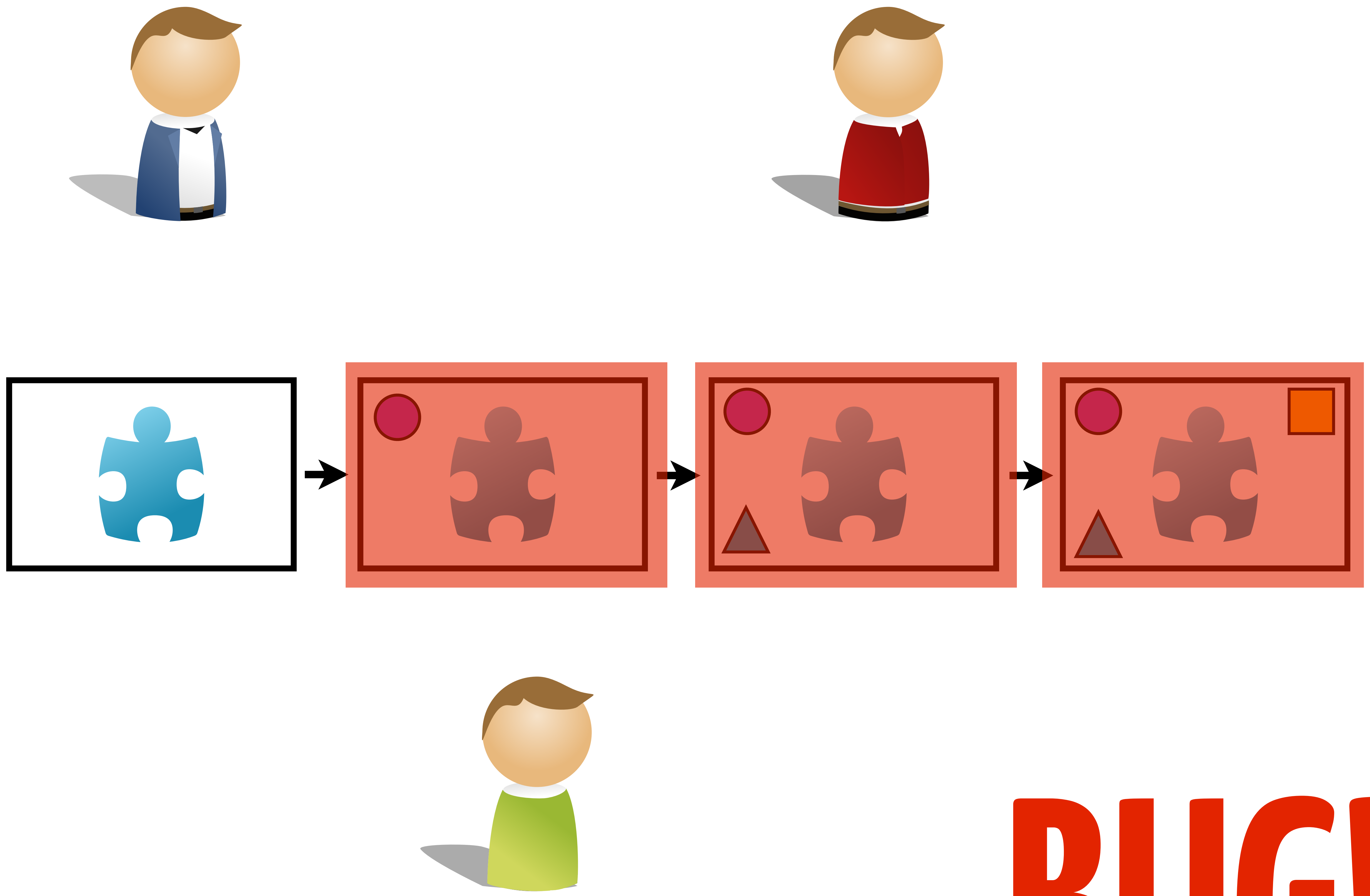




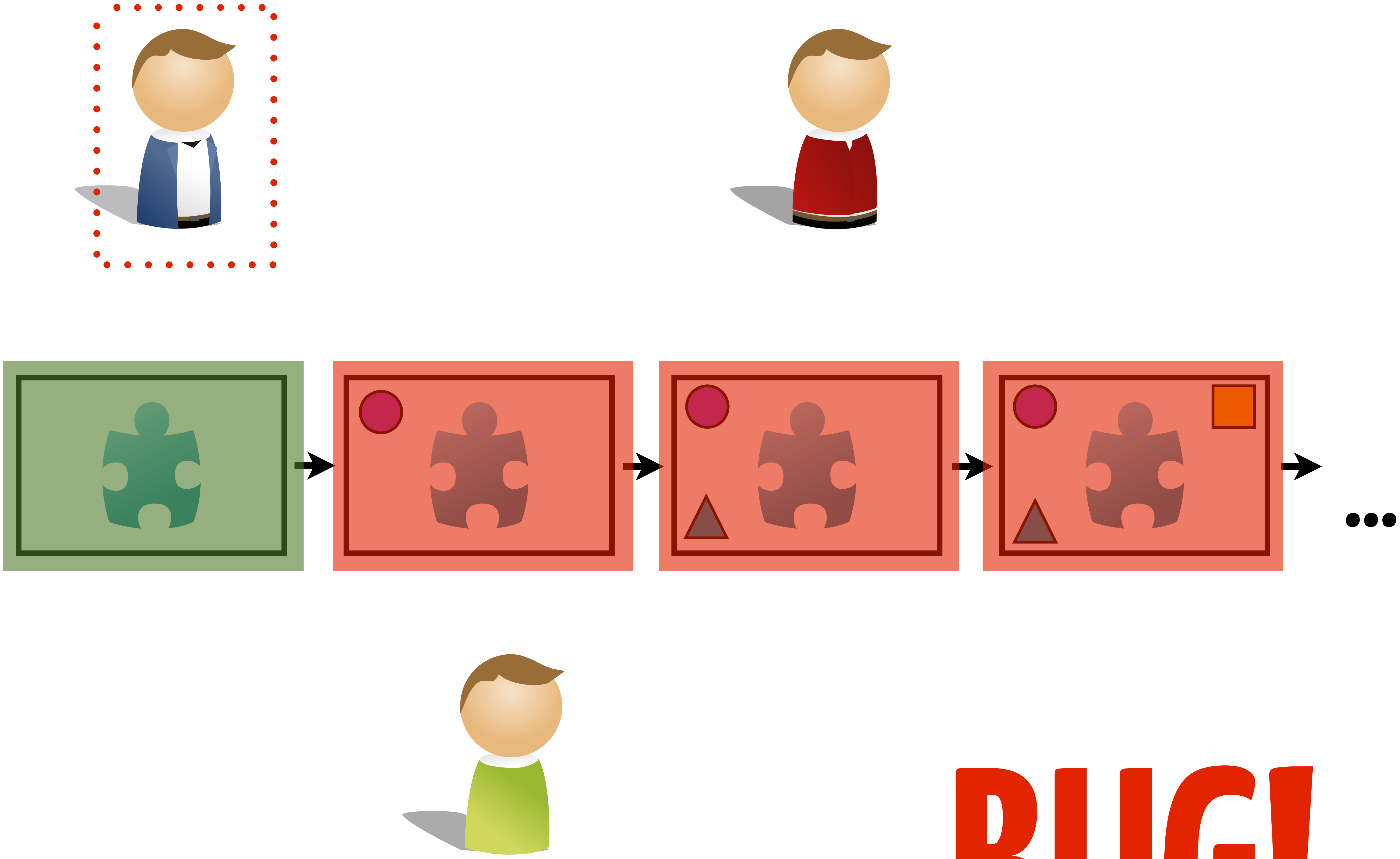

BUG!



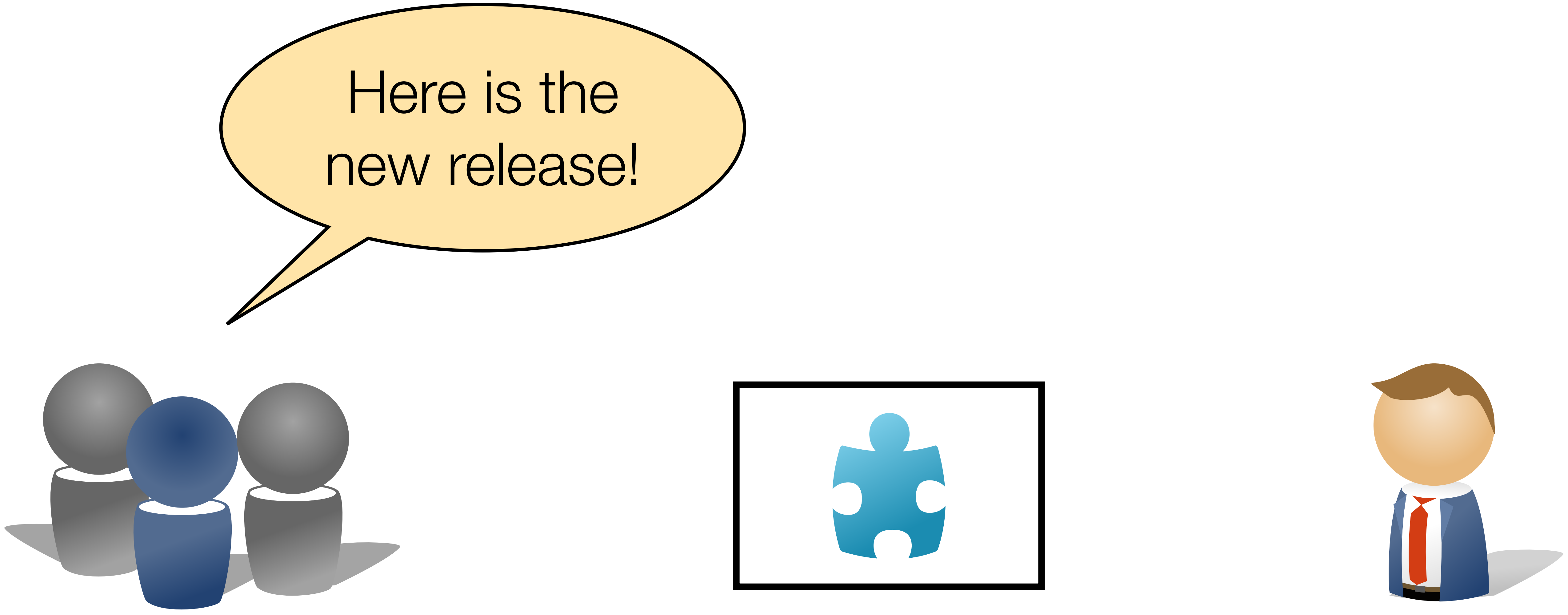
BUG!

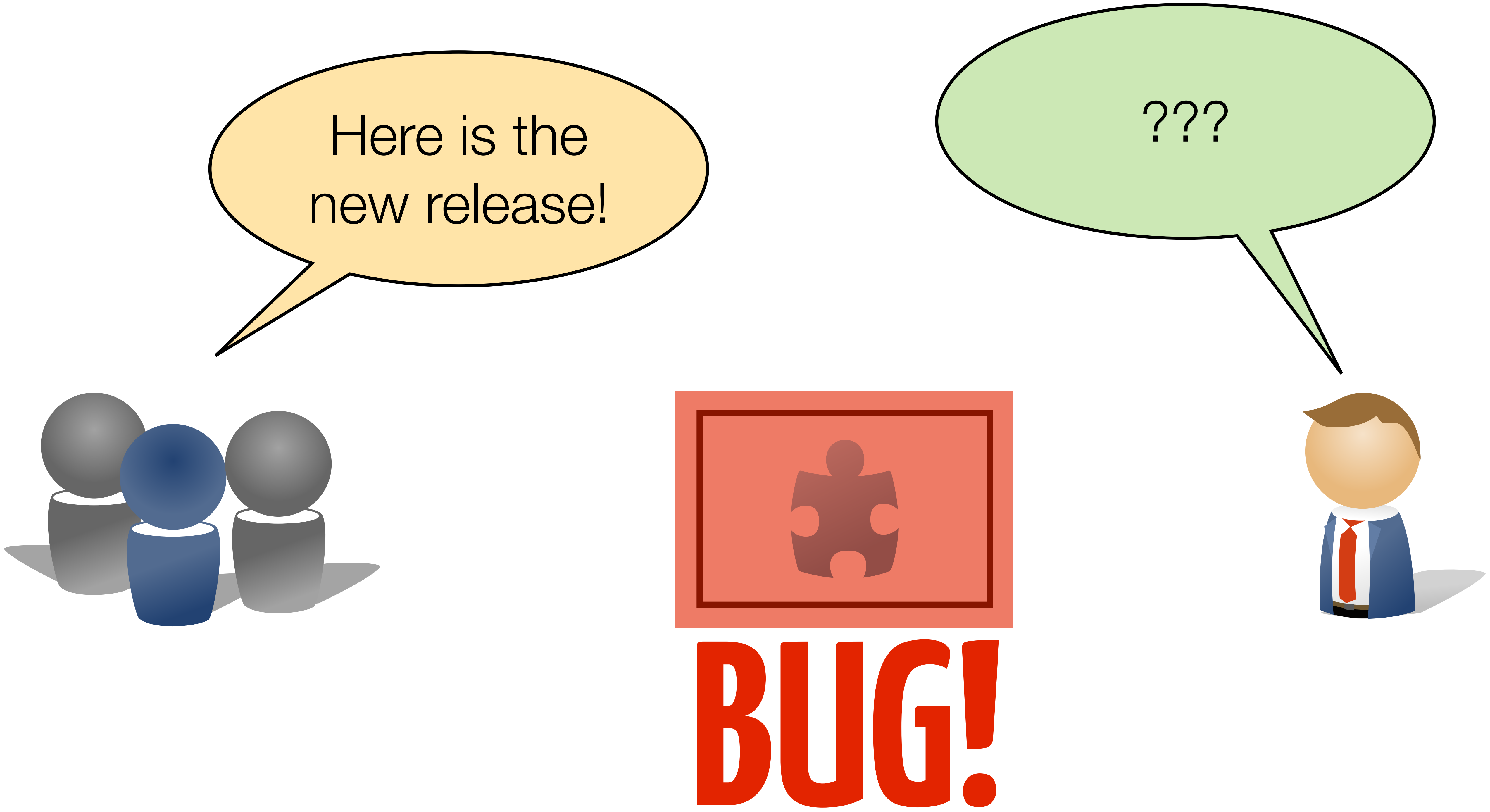


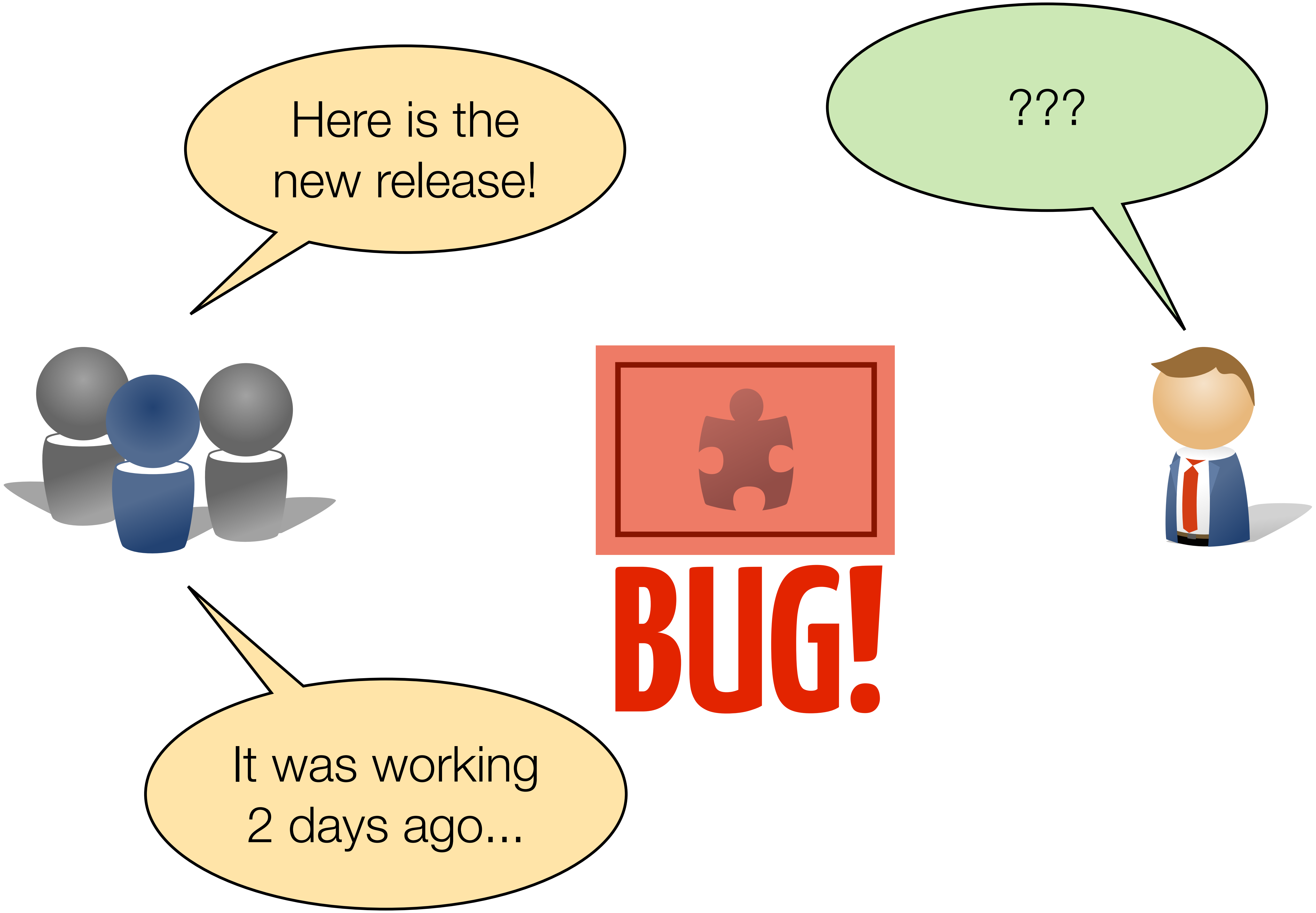
BUG!

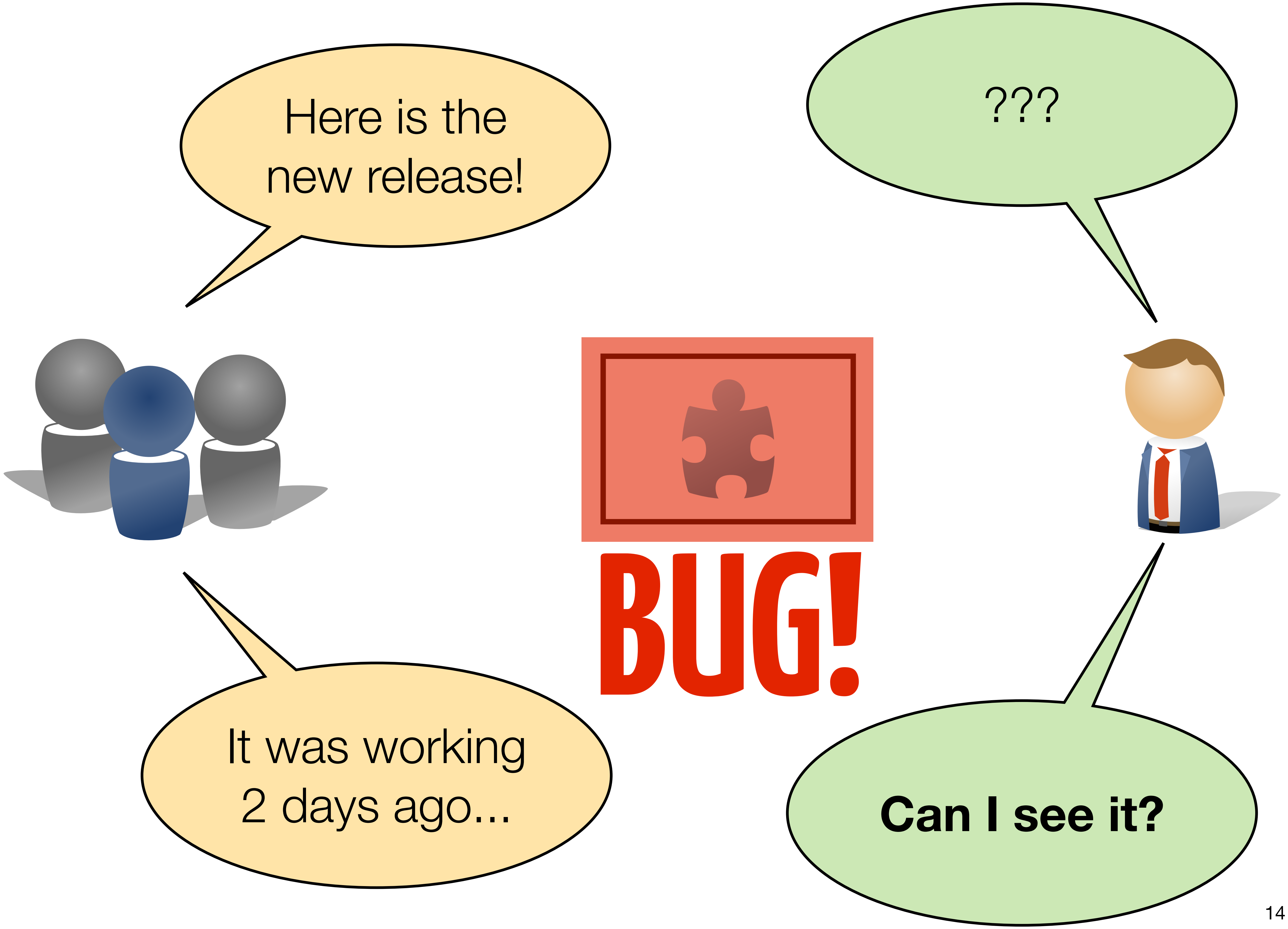


BUG!


Here is the new release!

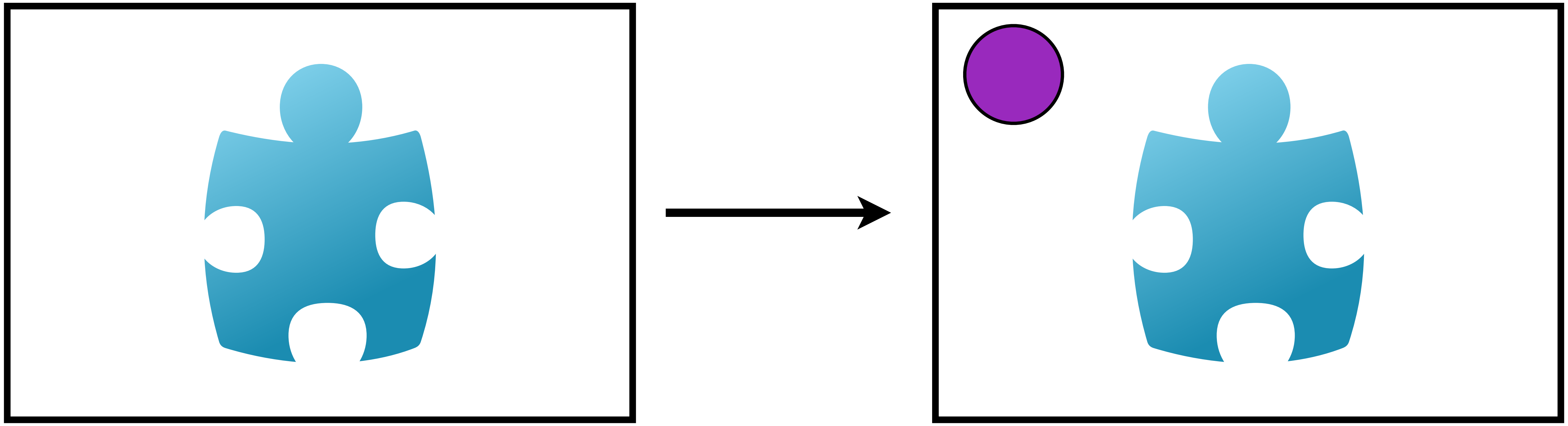

Here is the new release!
???
BUG!

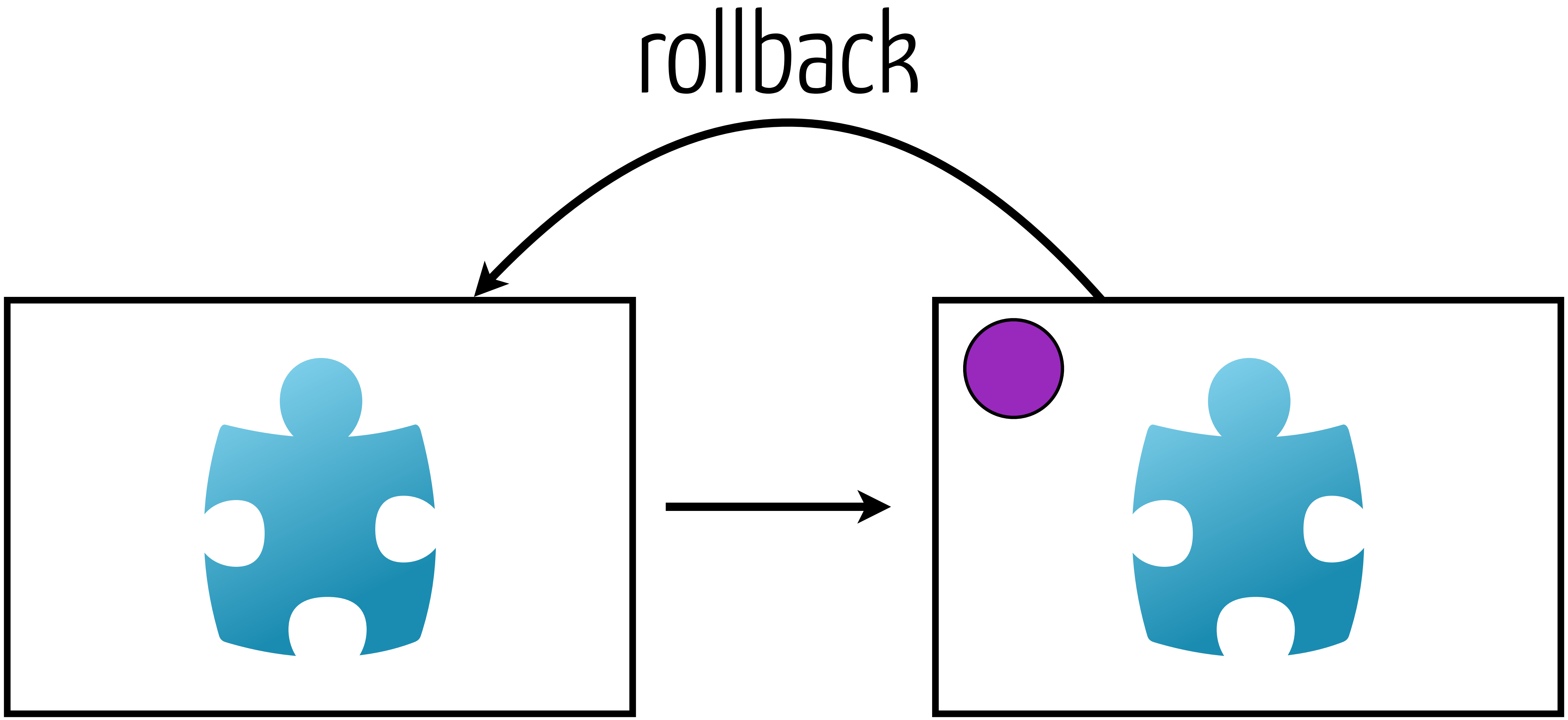

Here is the new release!
???
BUG!
It was working 2 days ago...

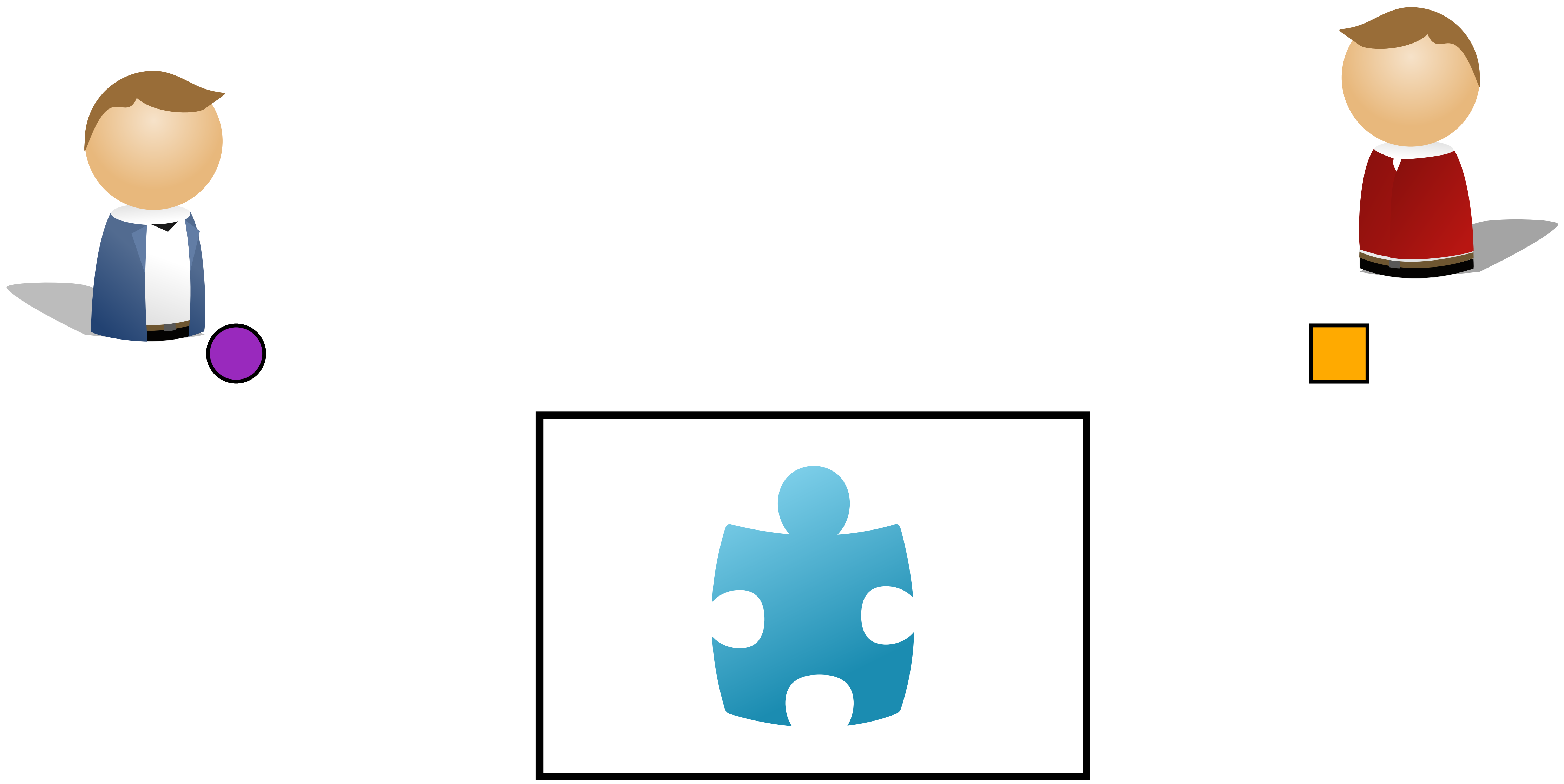

Here is the new release!
???
BUG!
It was working 2 days ago...
Can I see it?

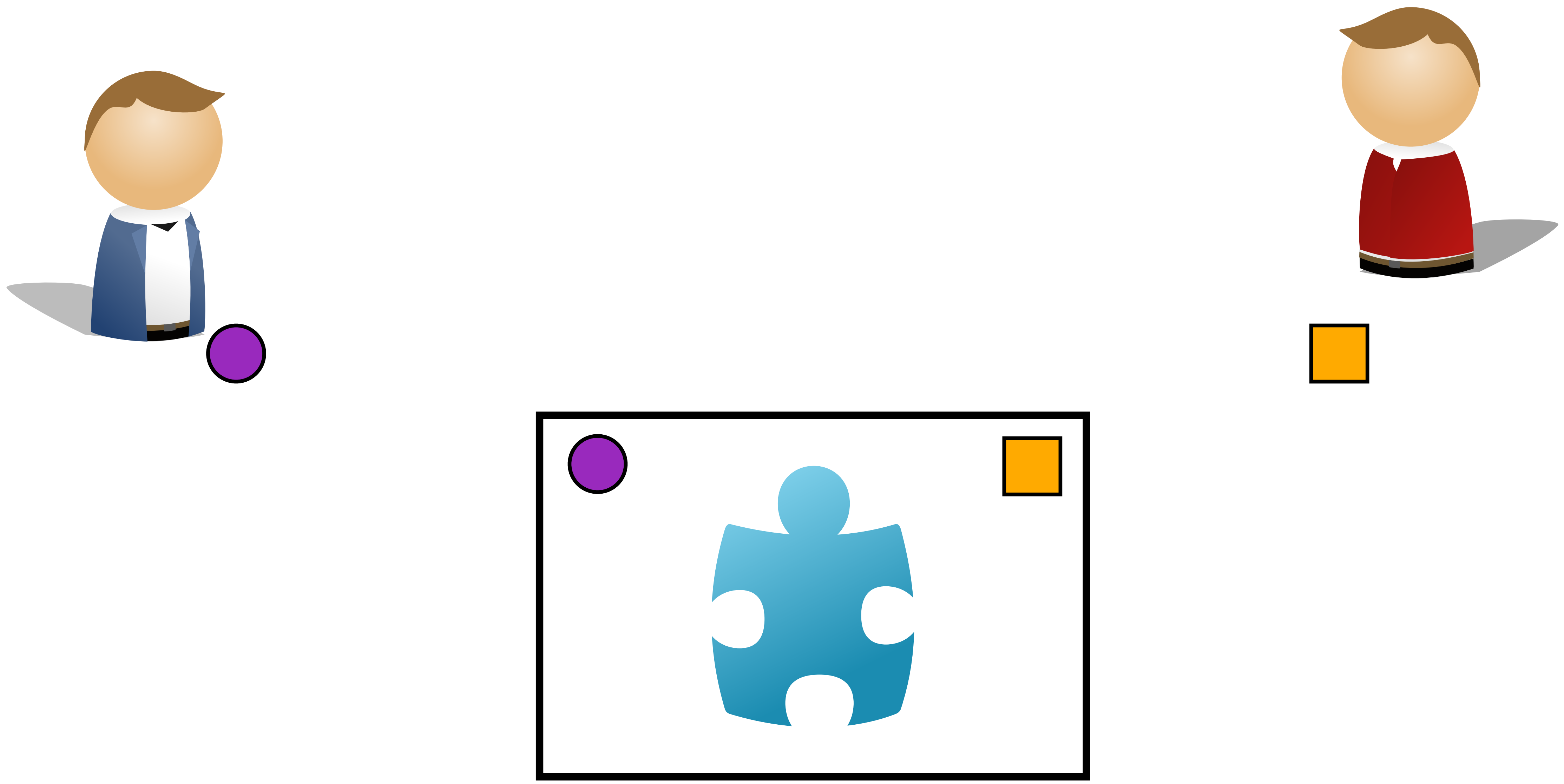
«**Why** do we version source code?»

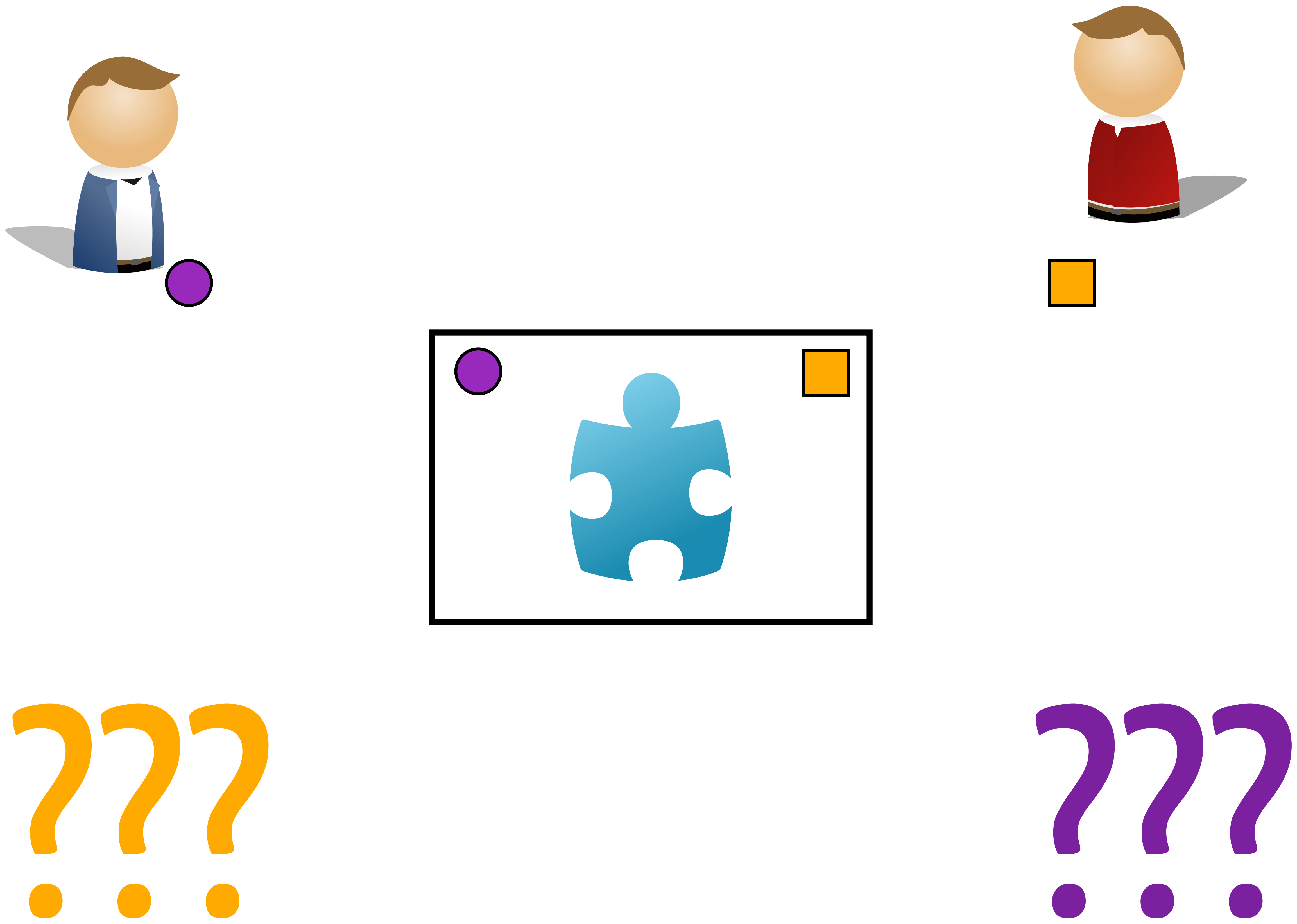
To **rollback** changes!

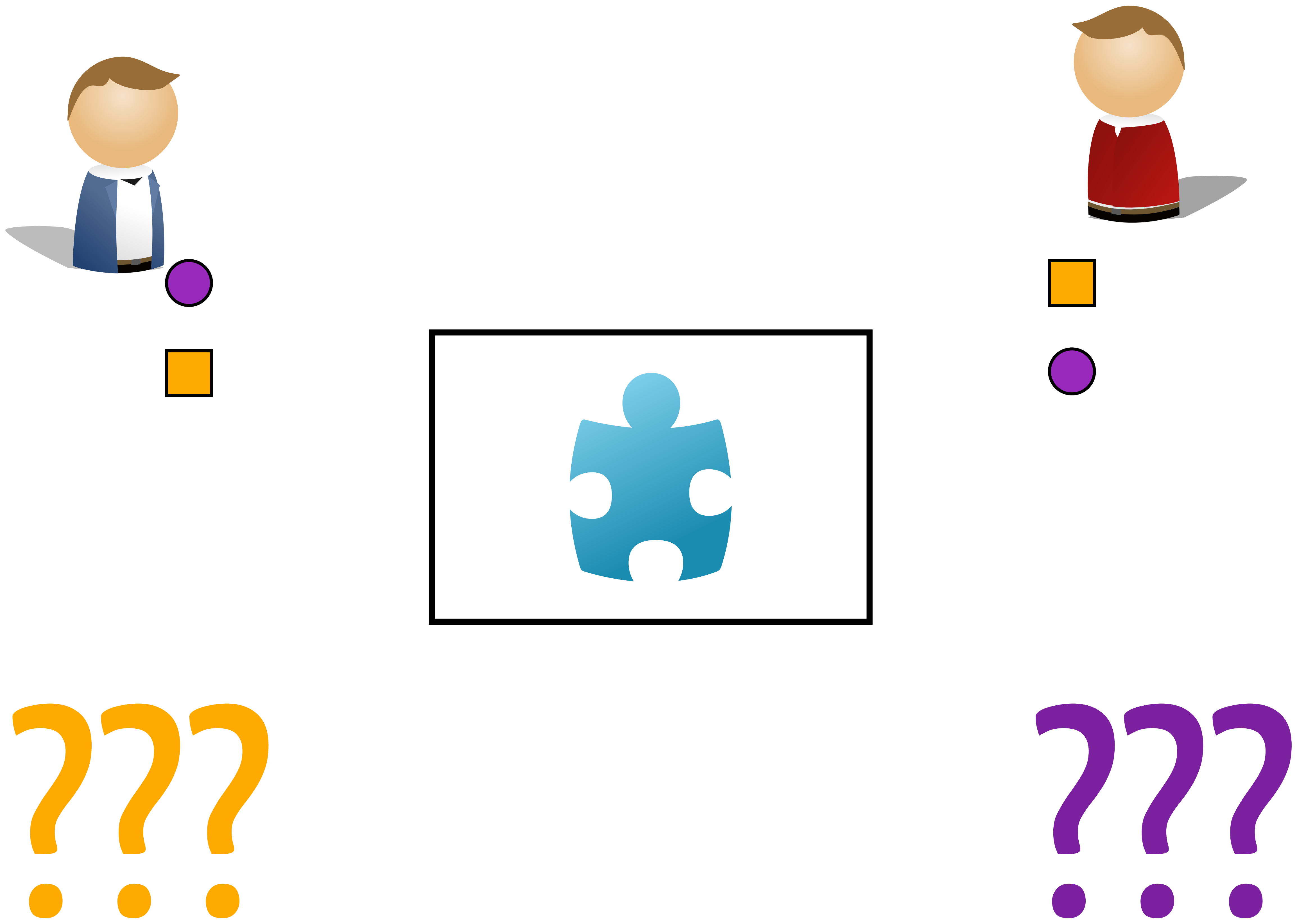



rollback



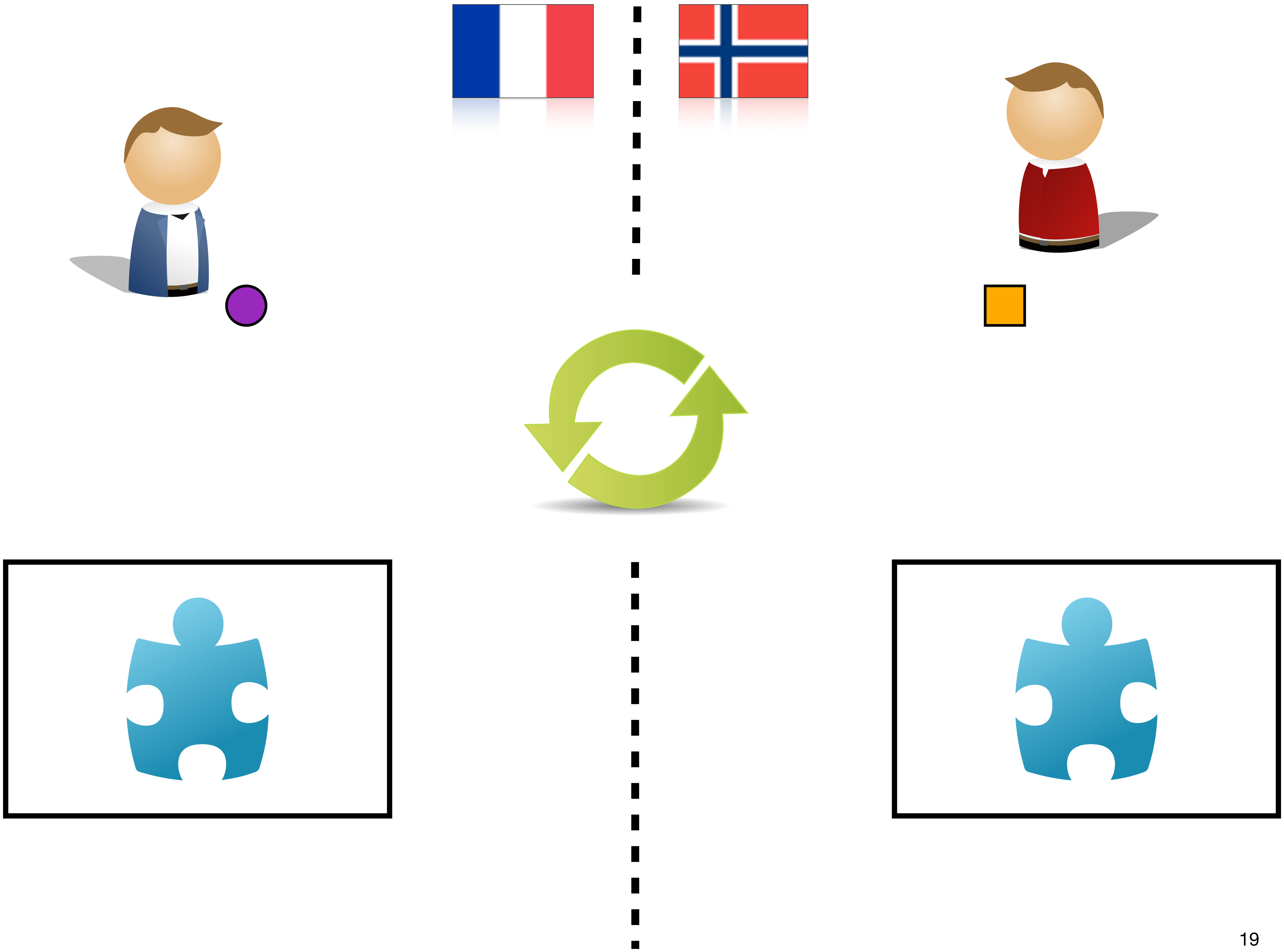


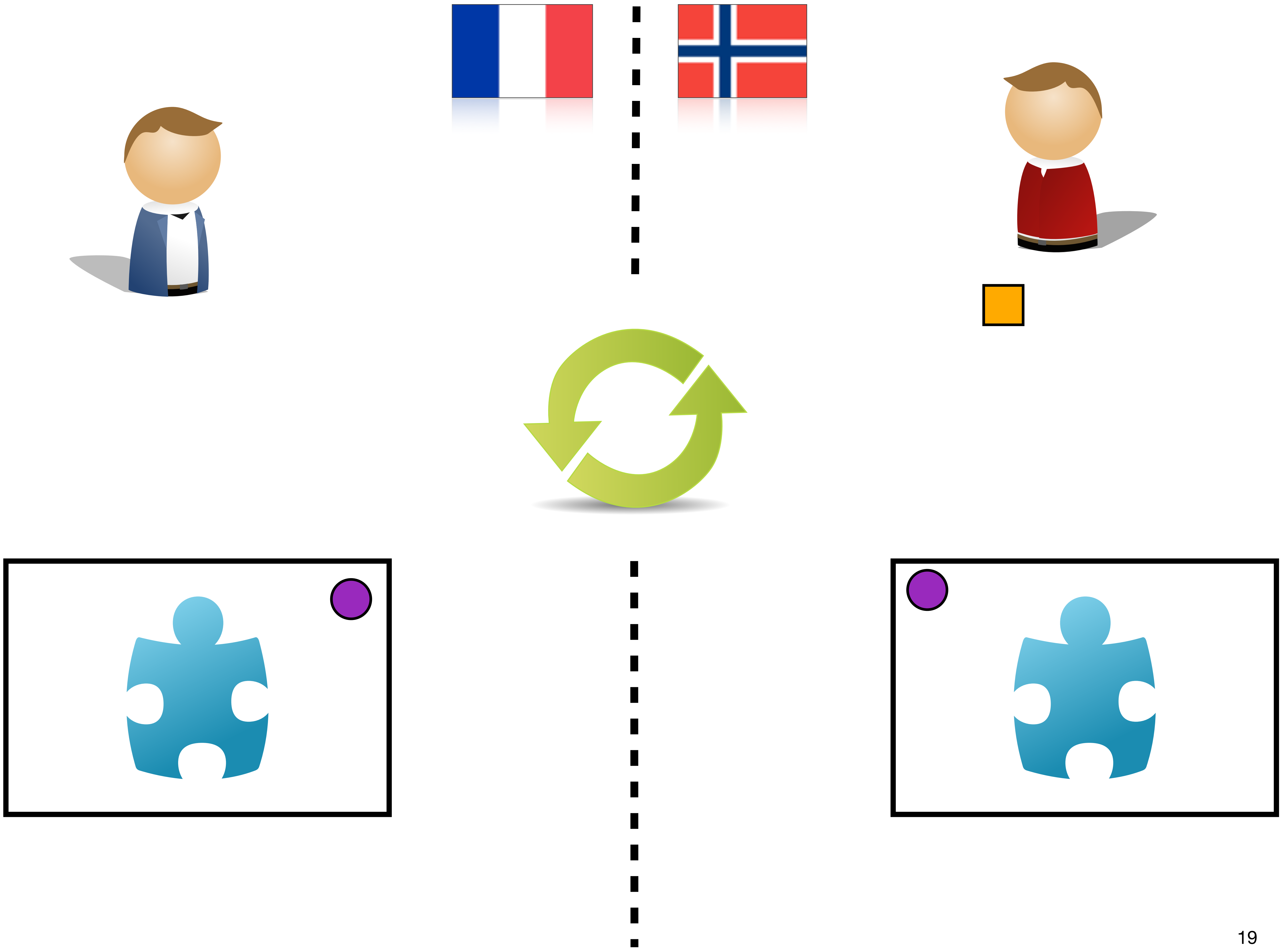


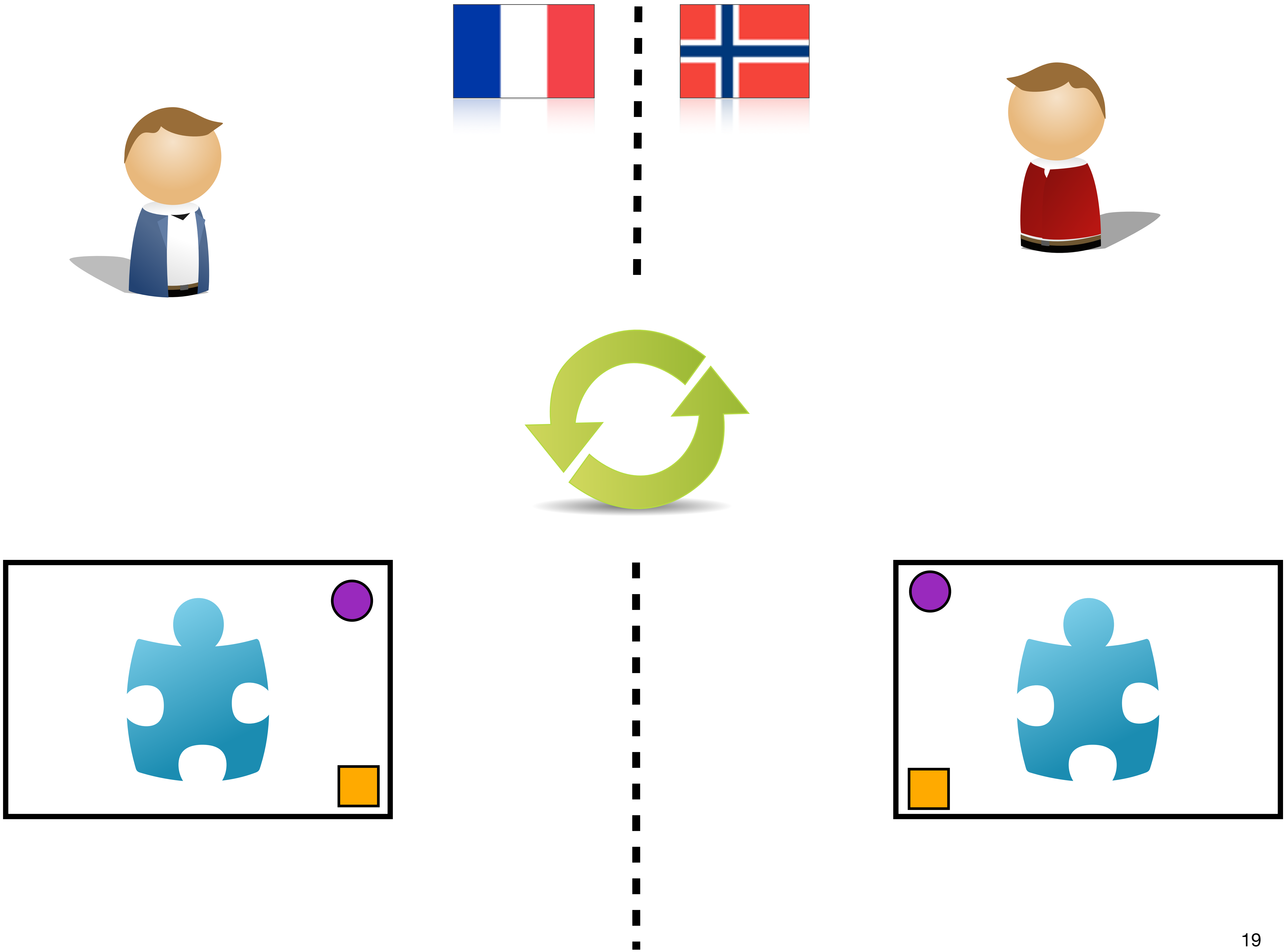


«**Why** do we version source code?»

To **share** changes!

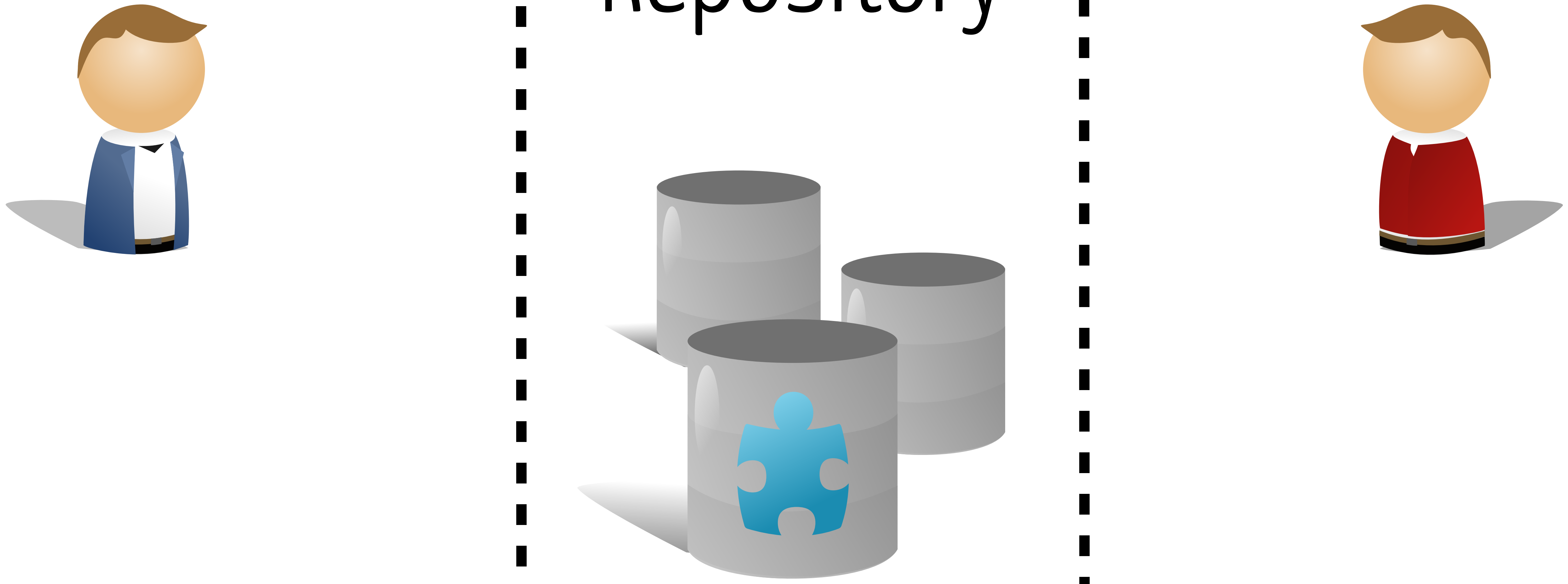




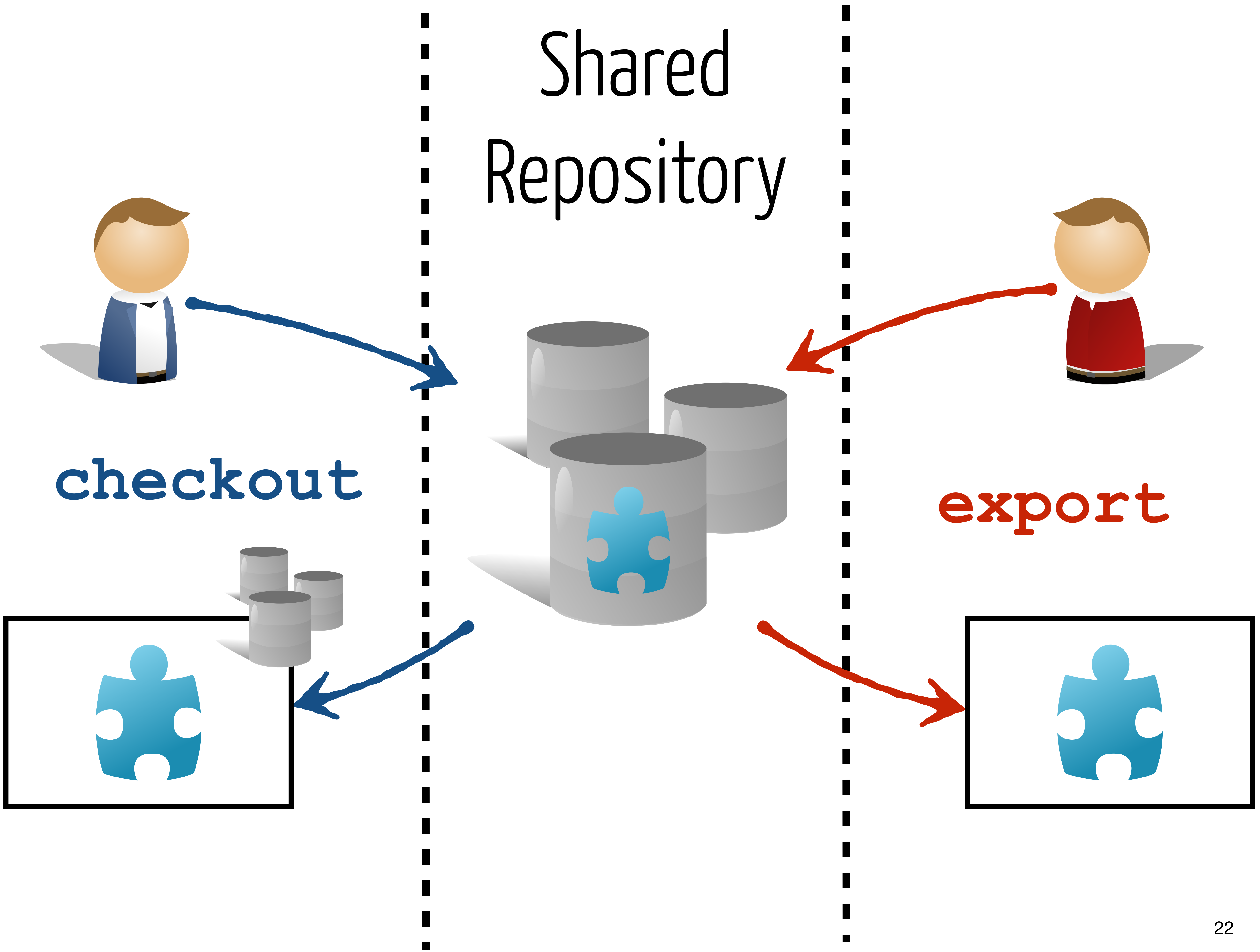


Centralized Model | (e.g., CVS, Subversion)

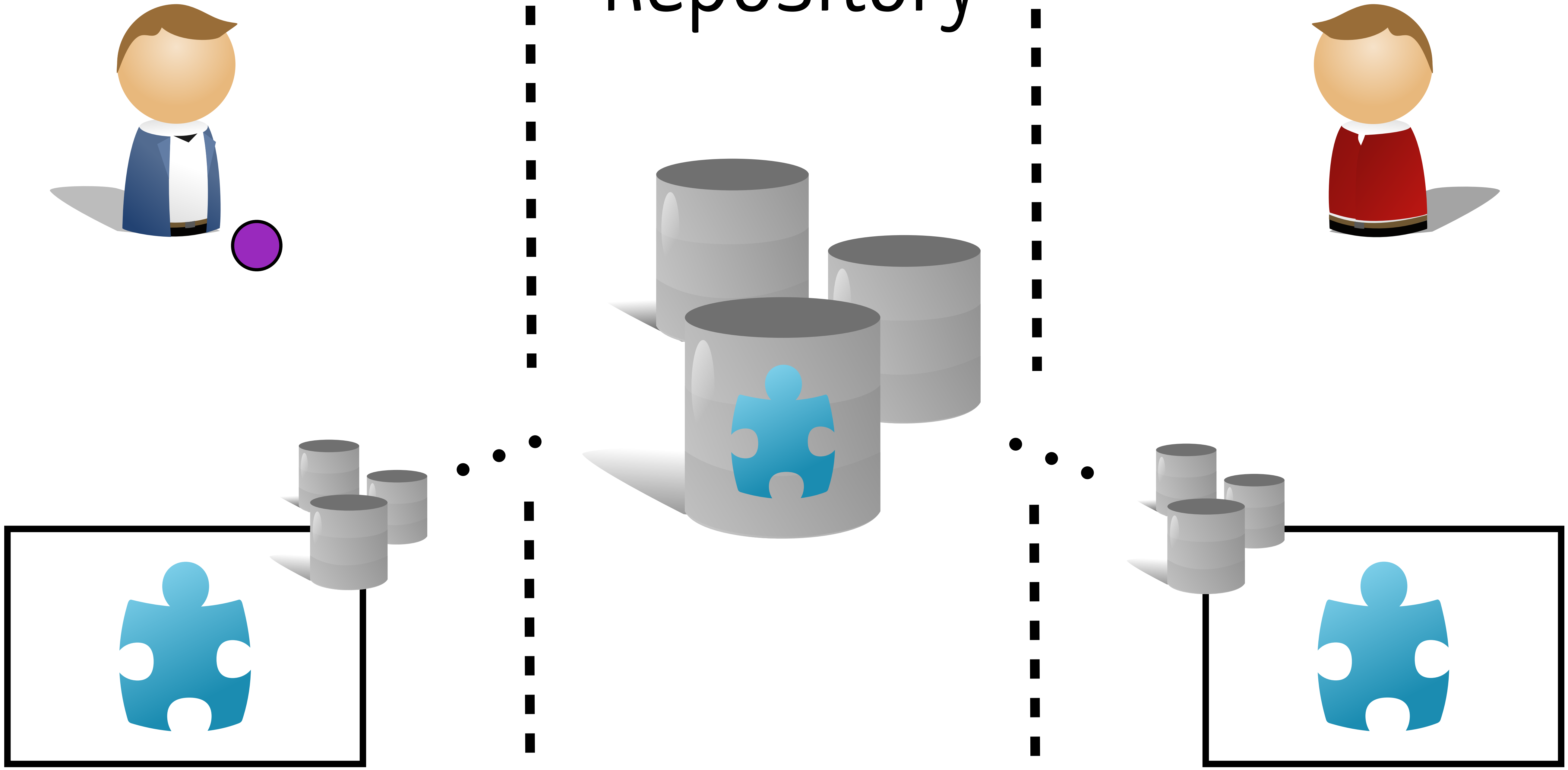
Shared Repository



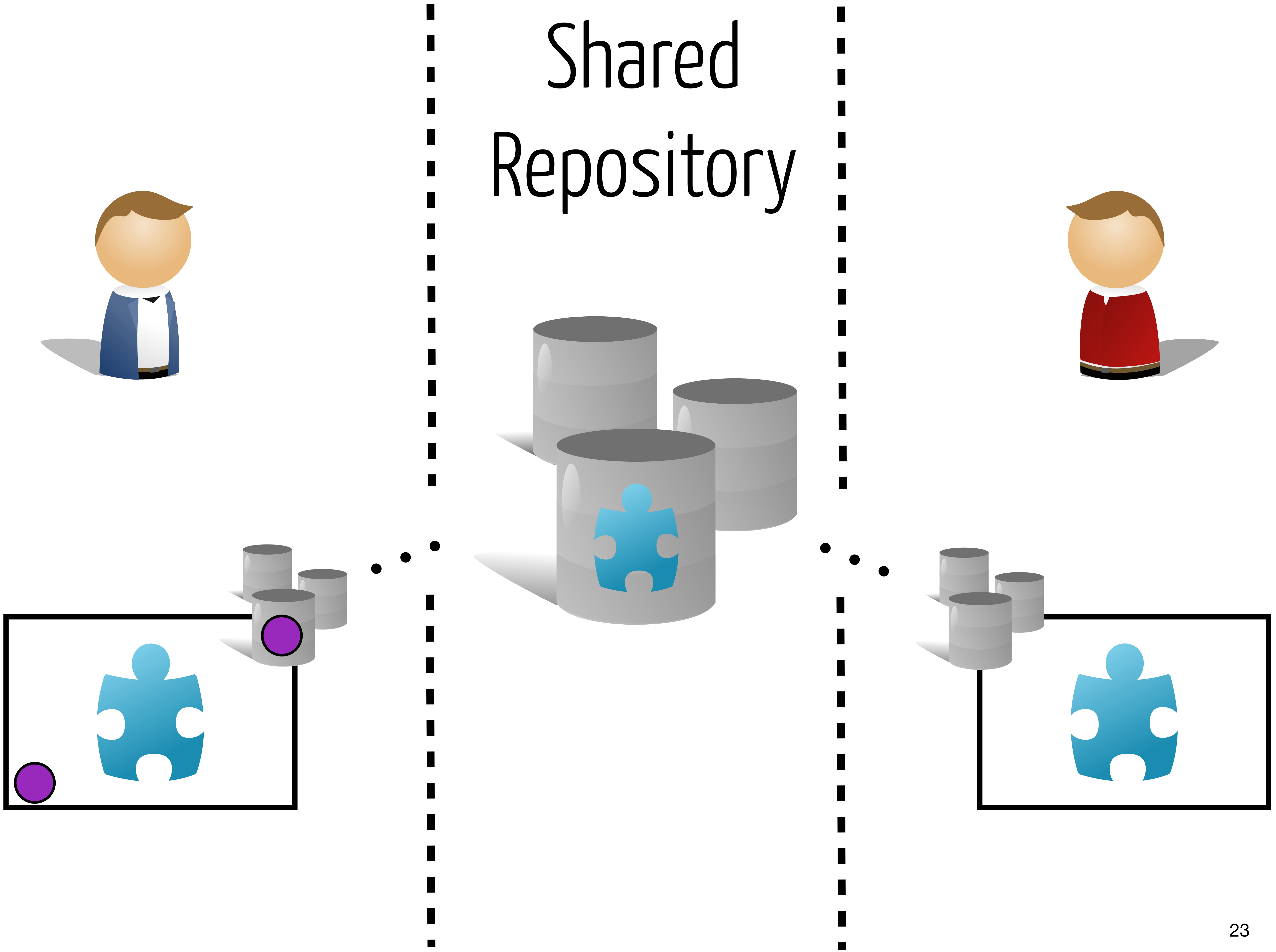
`create`


Shared Repository
checkout
export

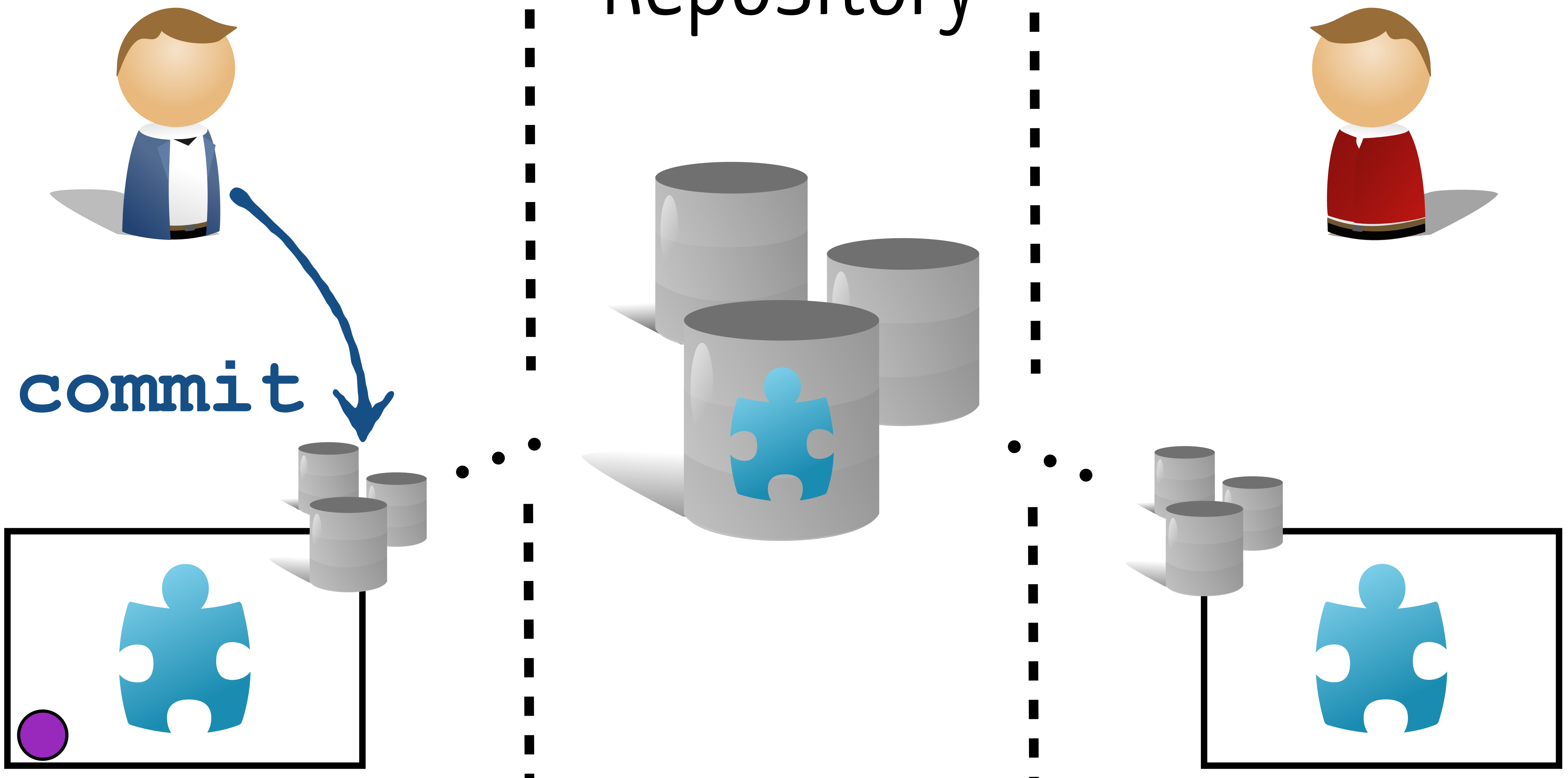
Shared Repository



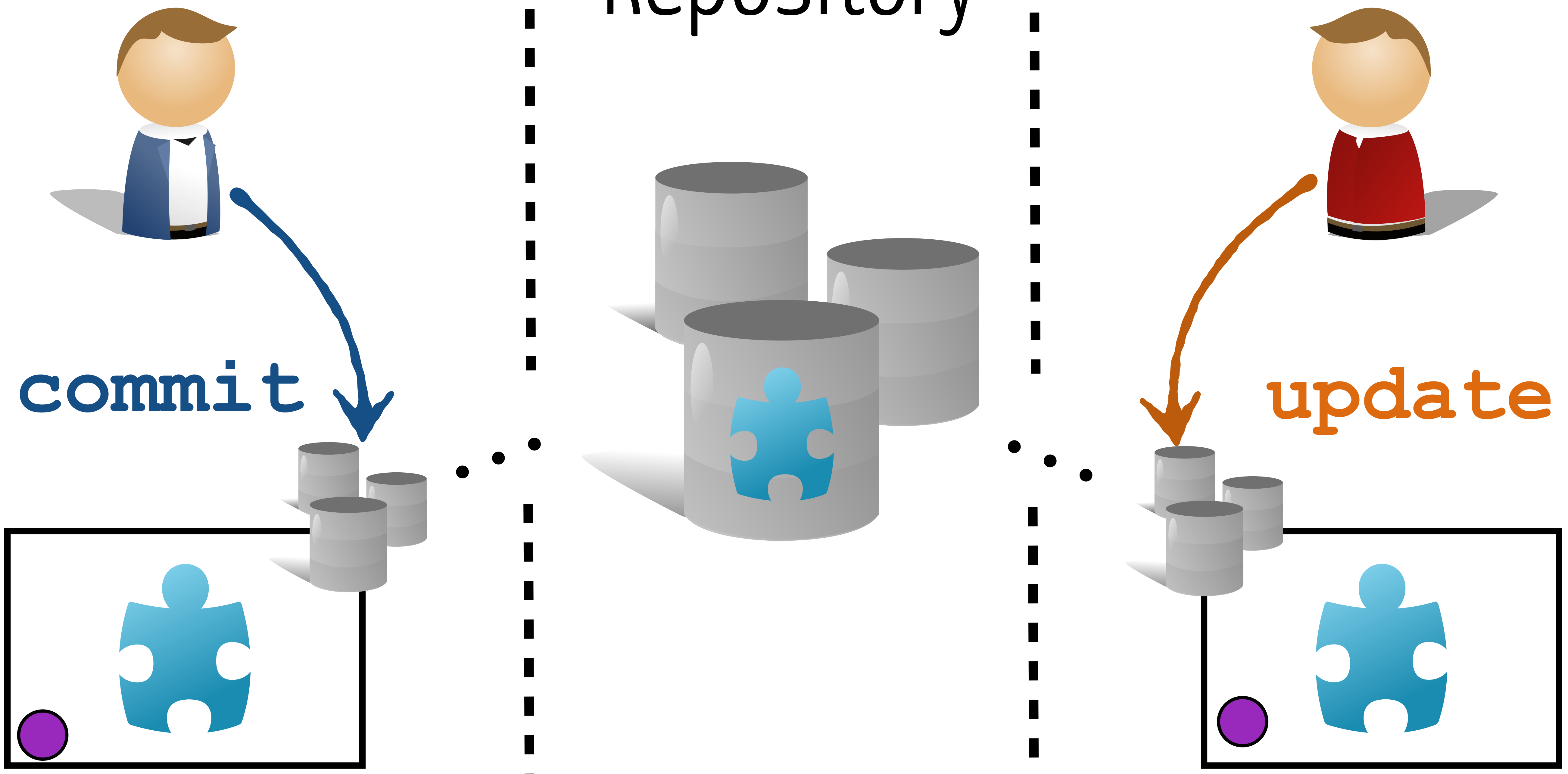
Shared Repository



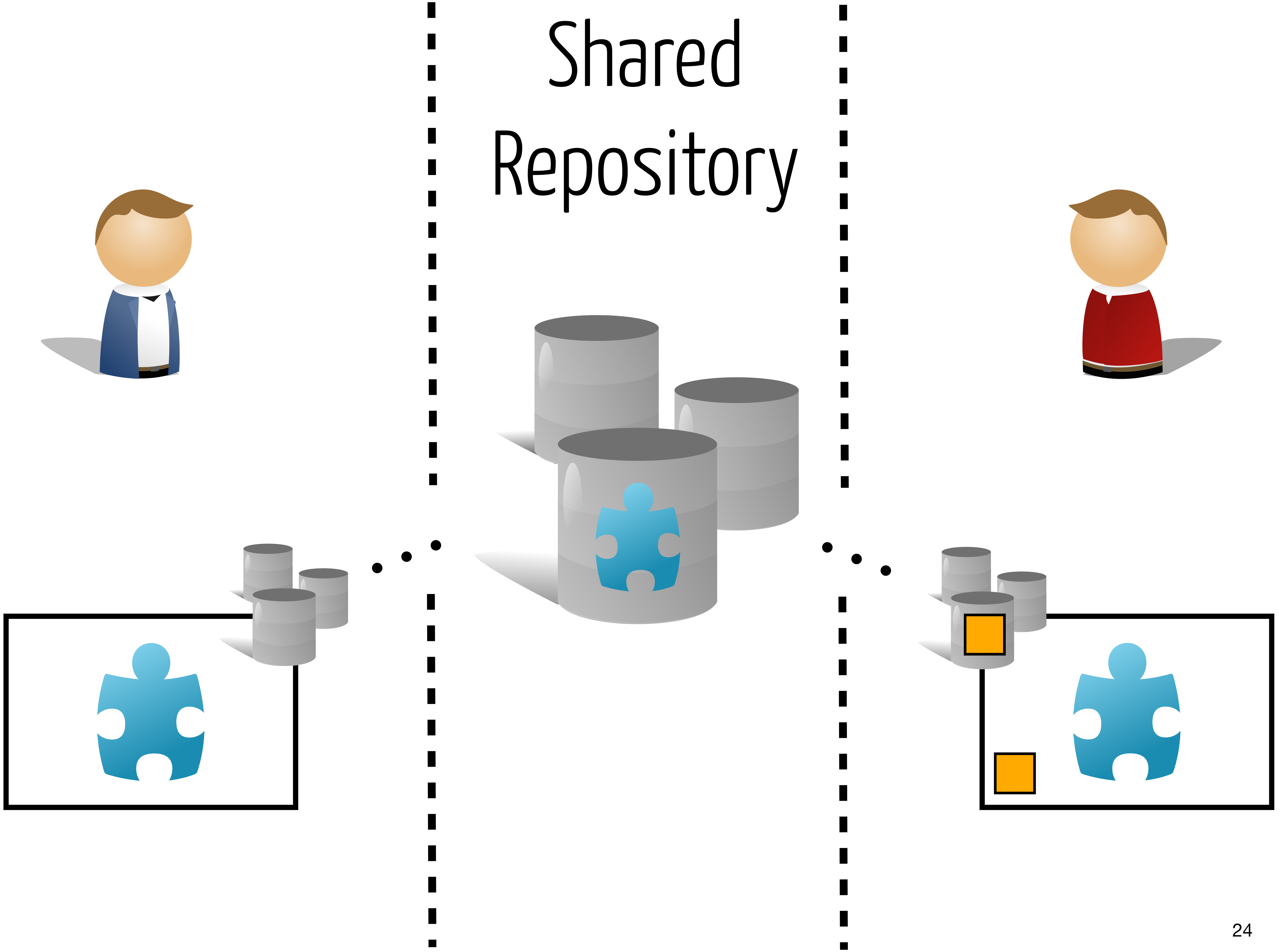
Shared Repository



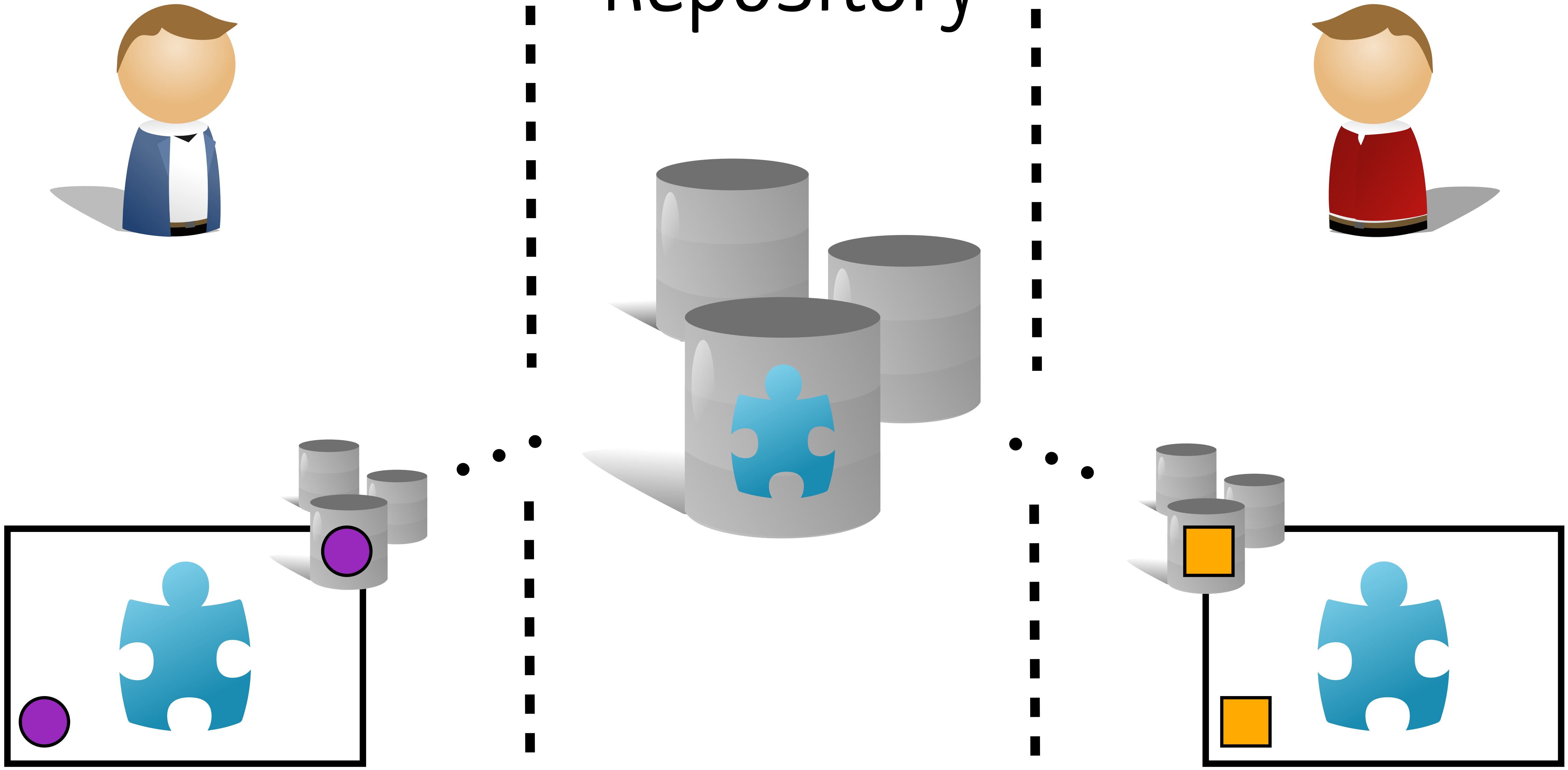
Shared Repository



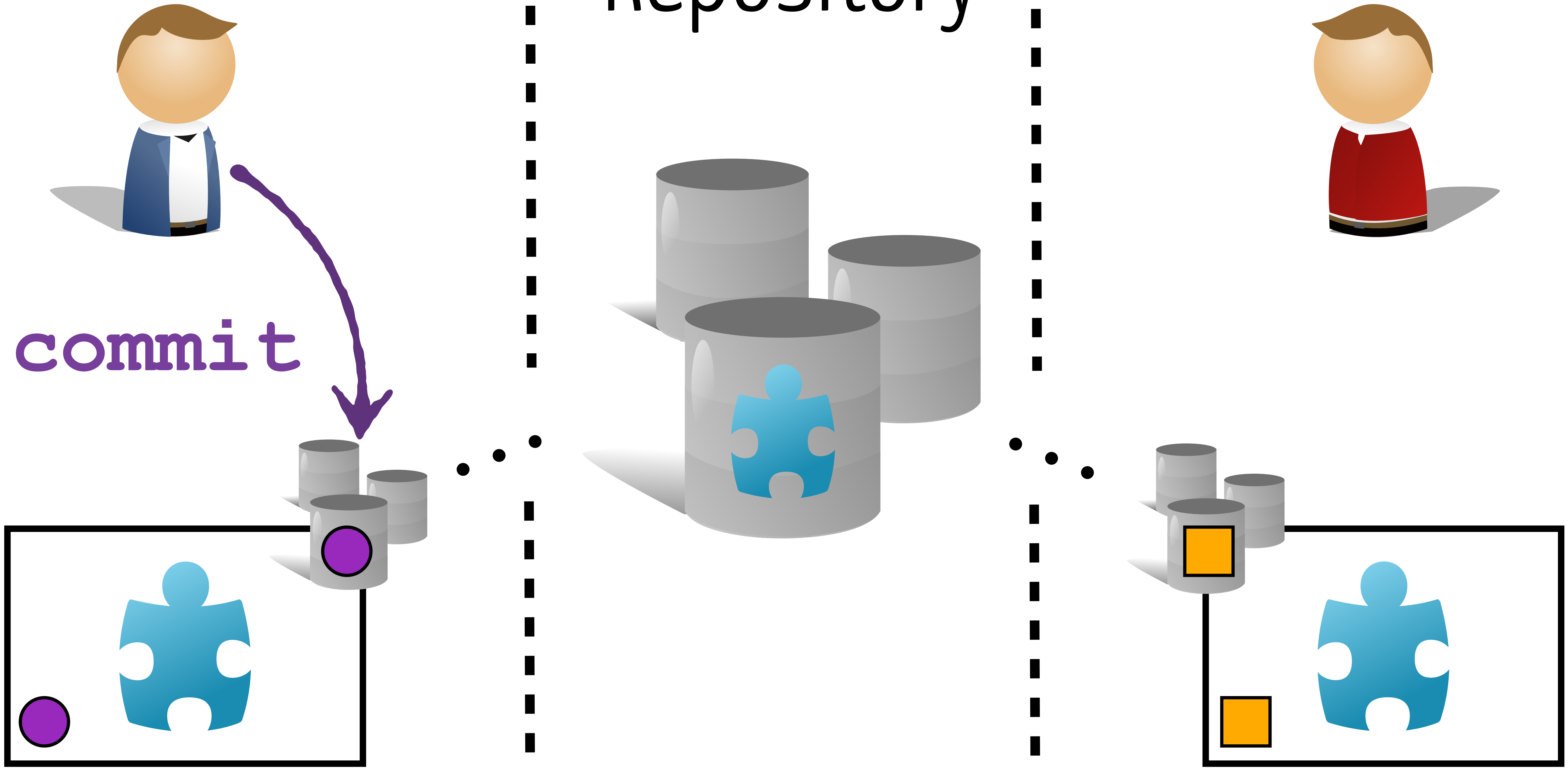
Shared Repository



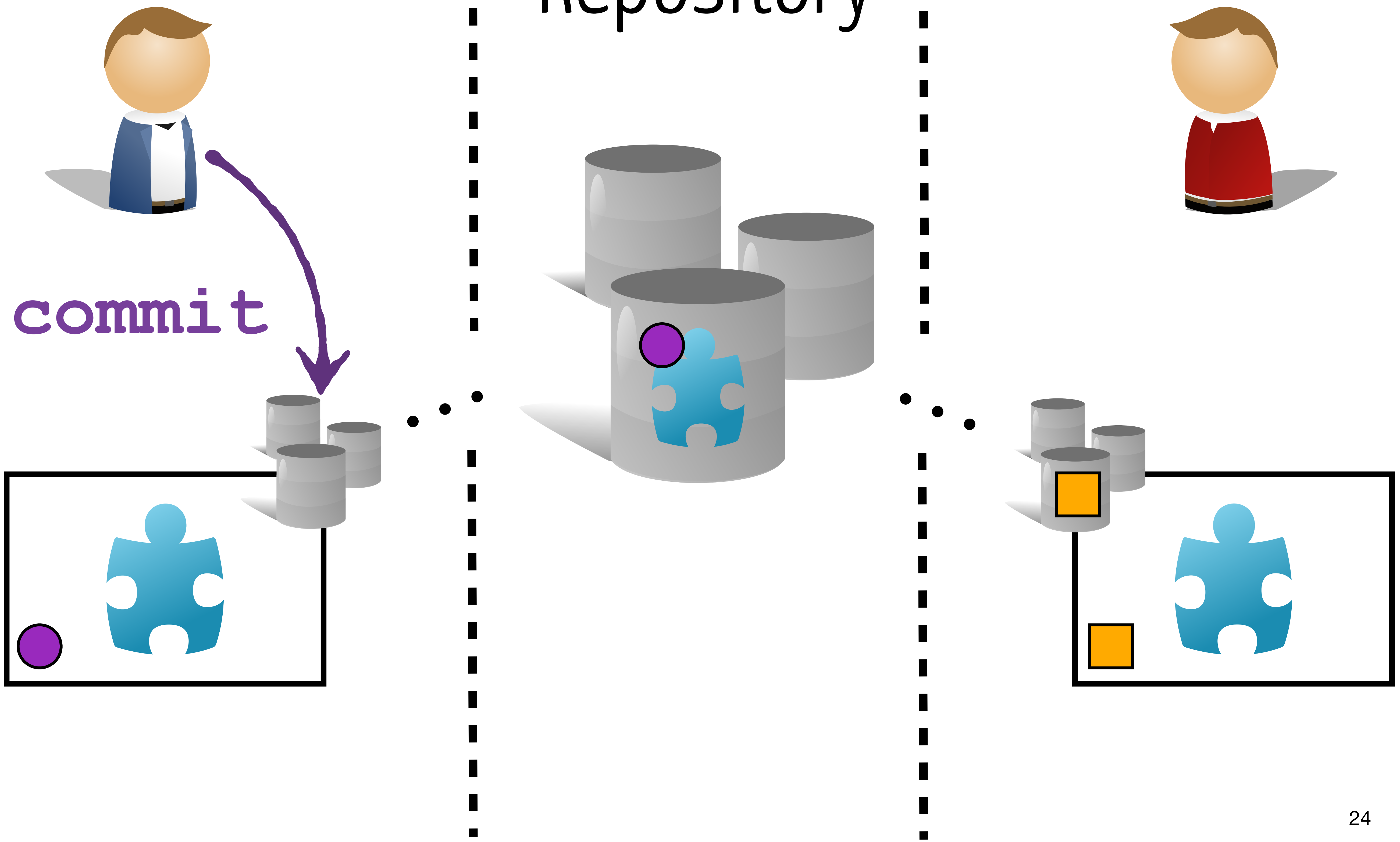
Shared Repository



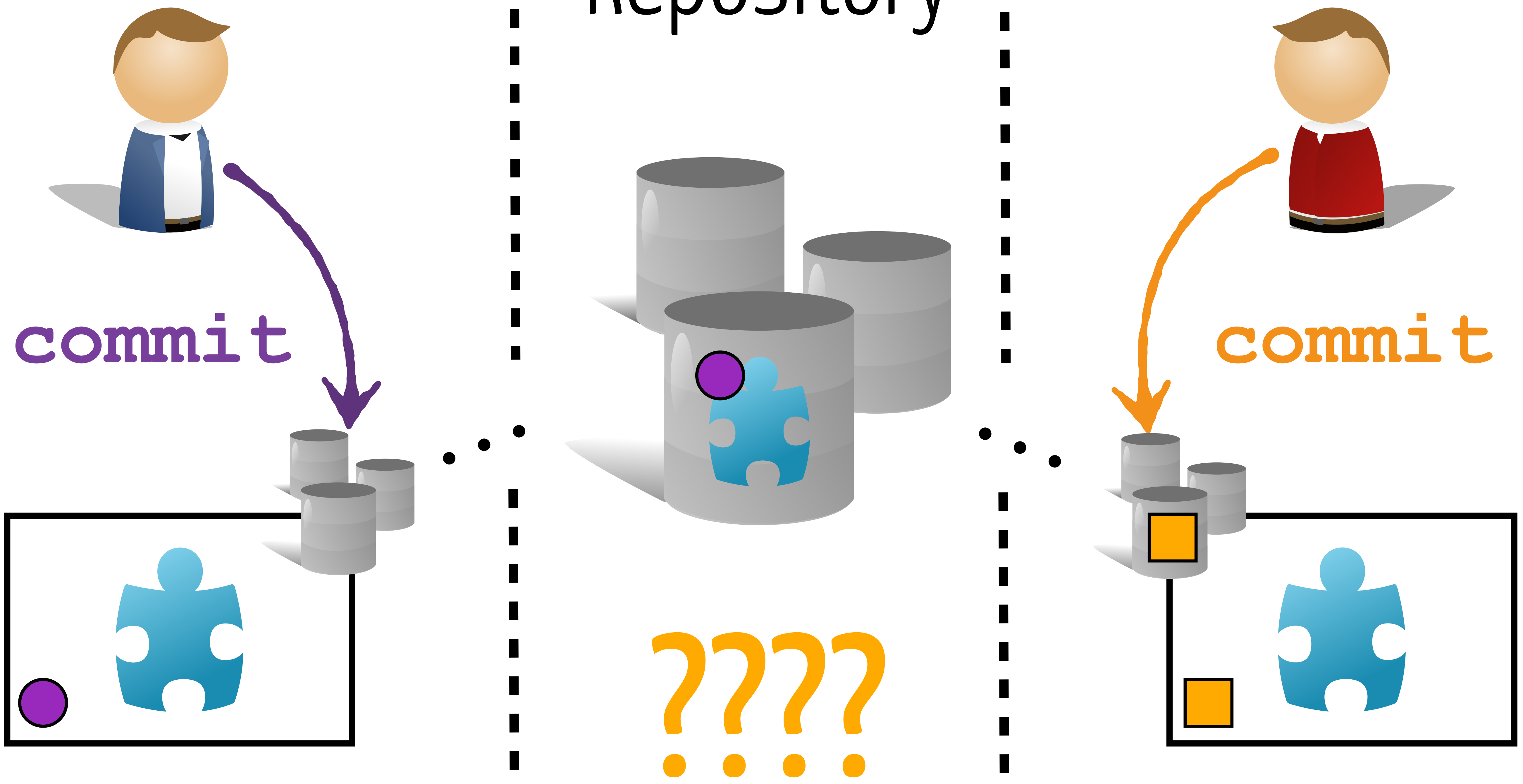
Shared Repository



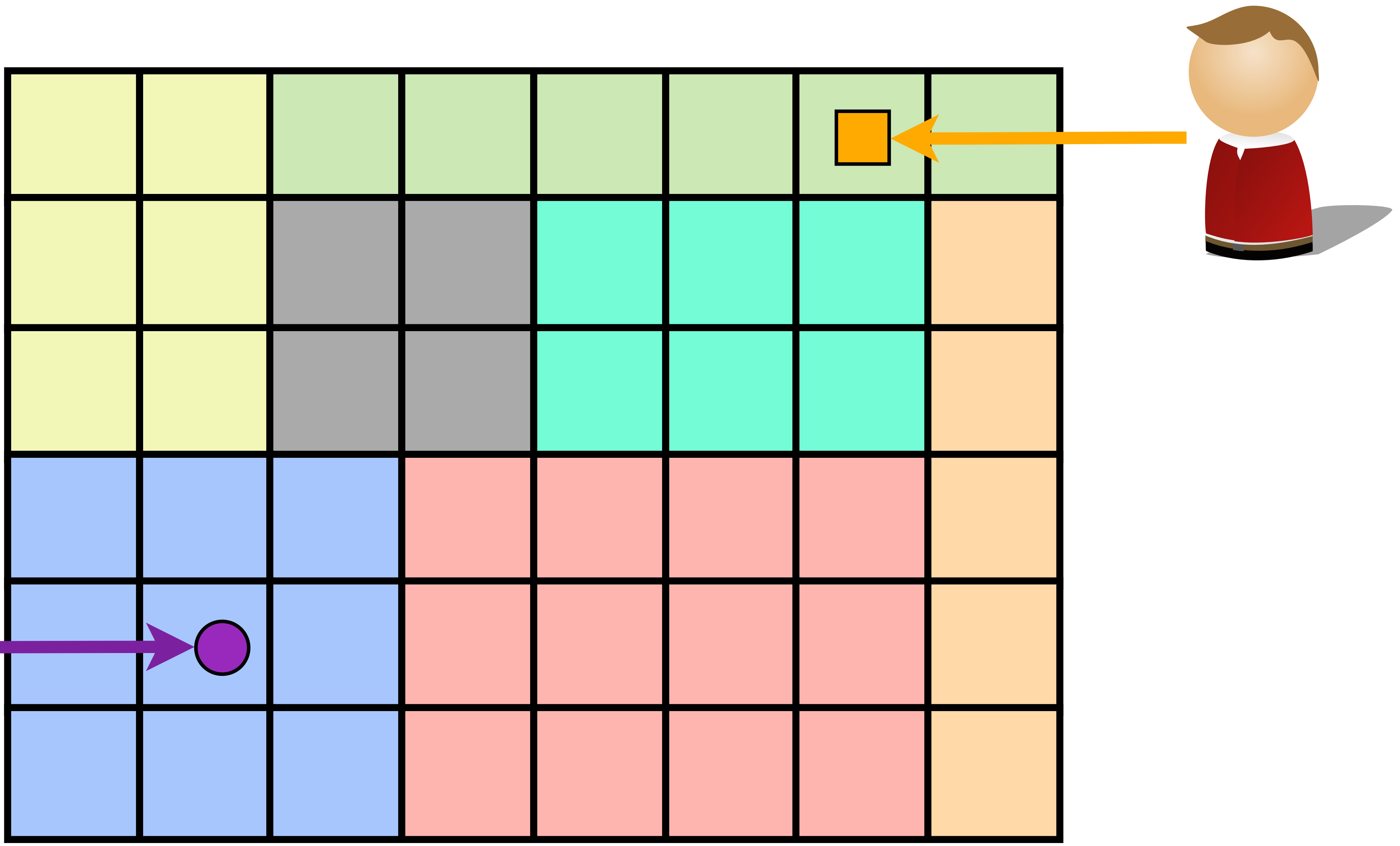
Shared Repository



Shared Repository

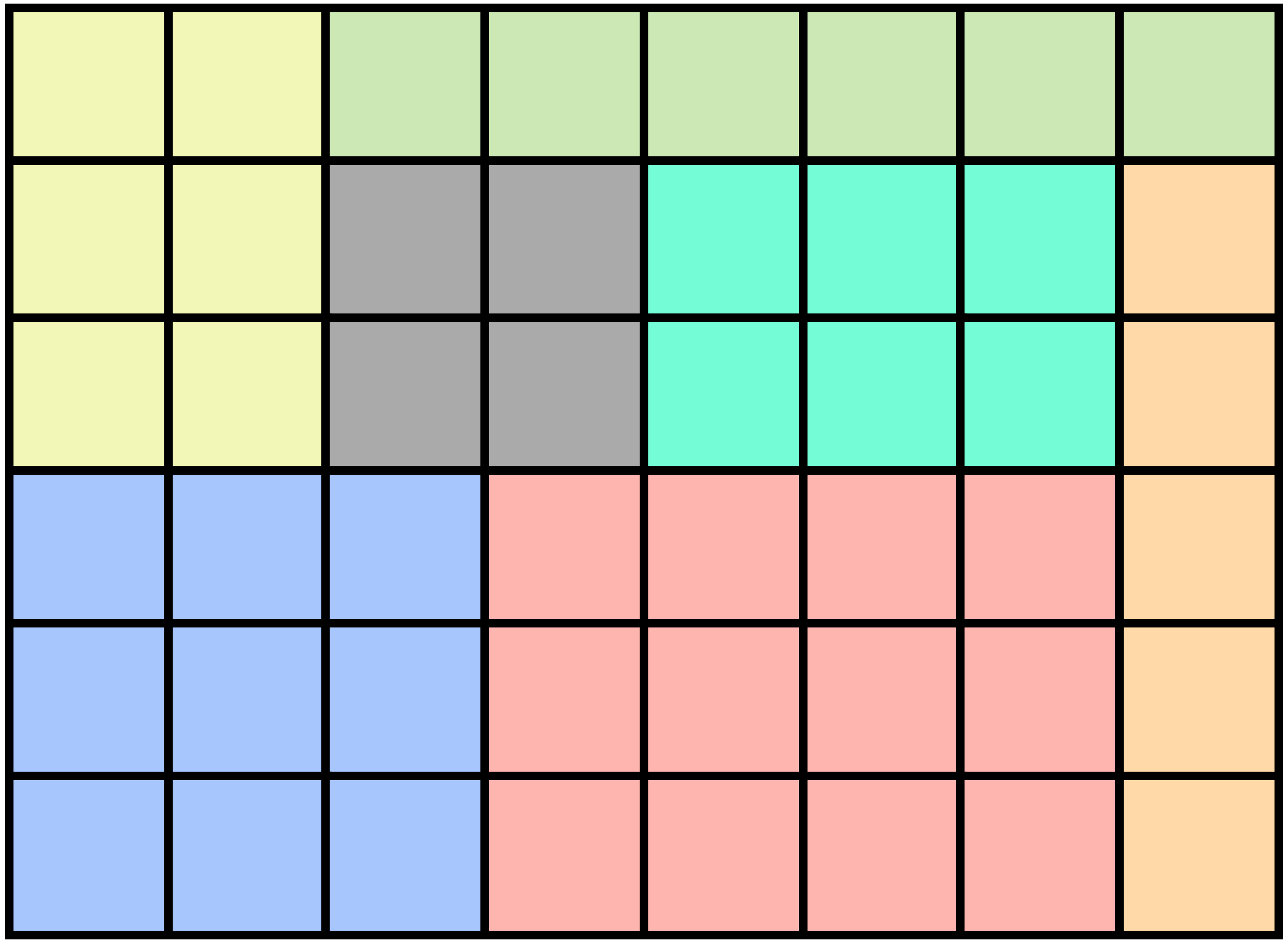


#1: different files



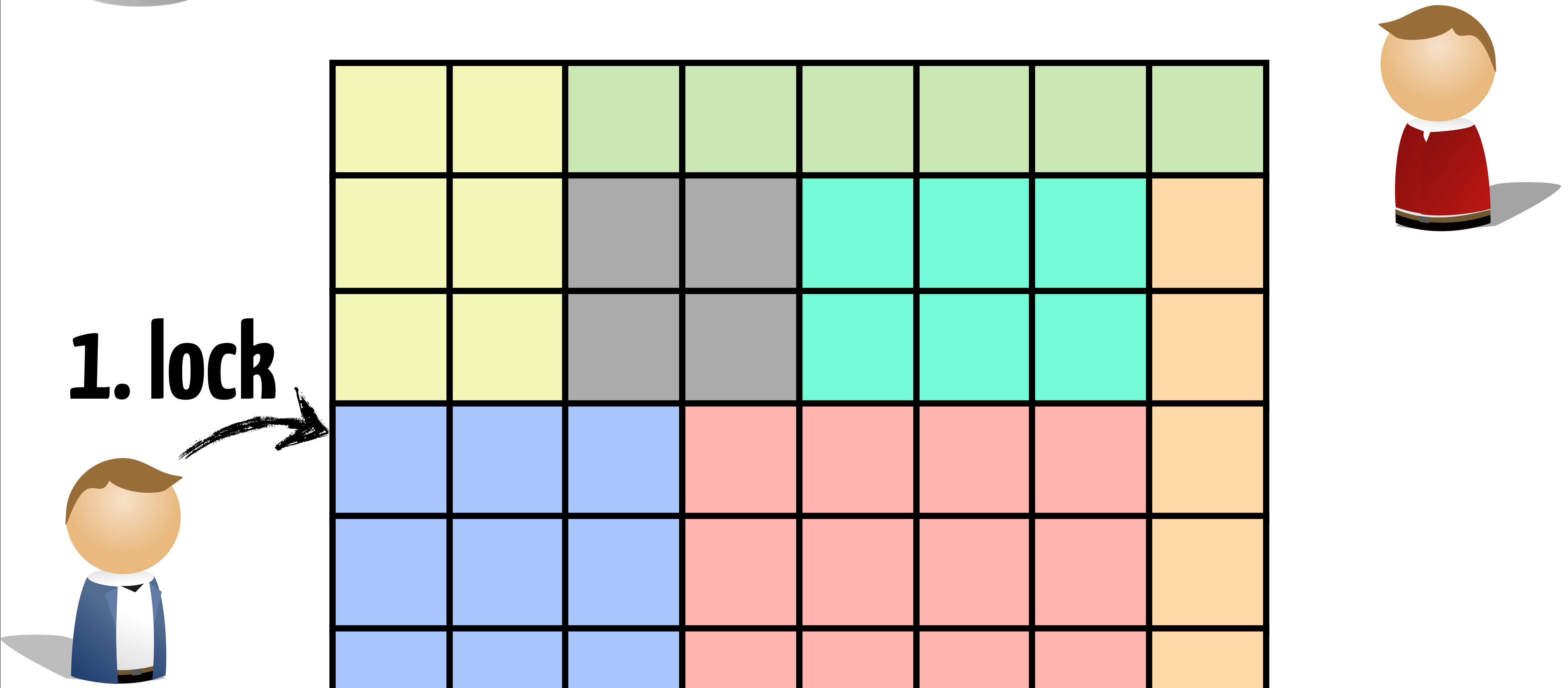
Atomic operations. No problem at all!

#2: different part of the same file



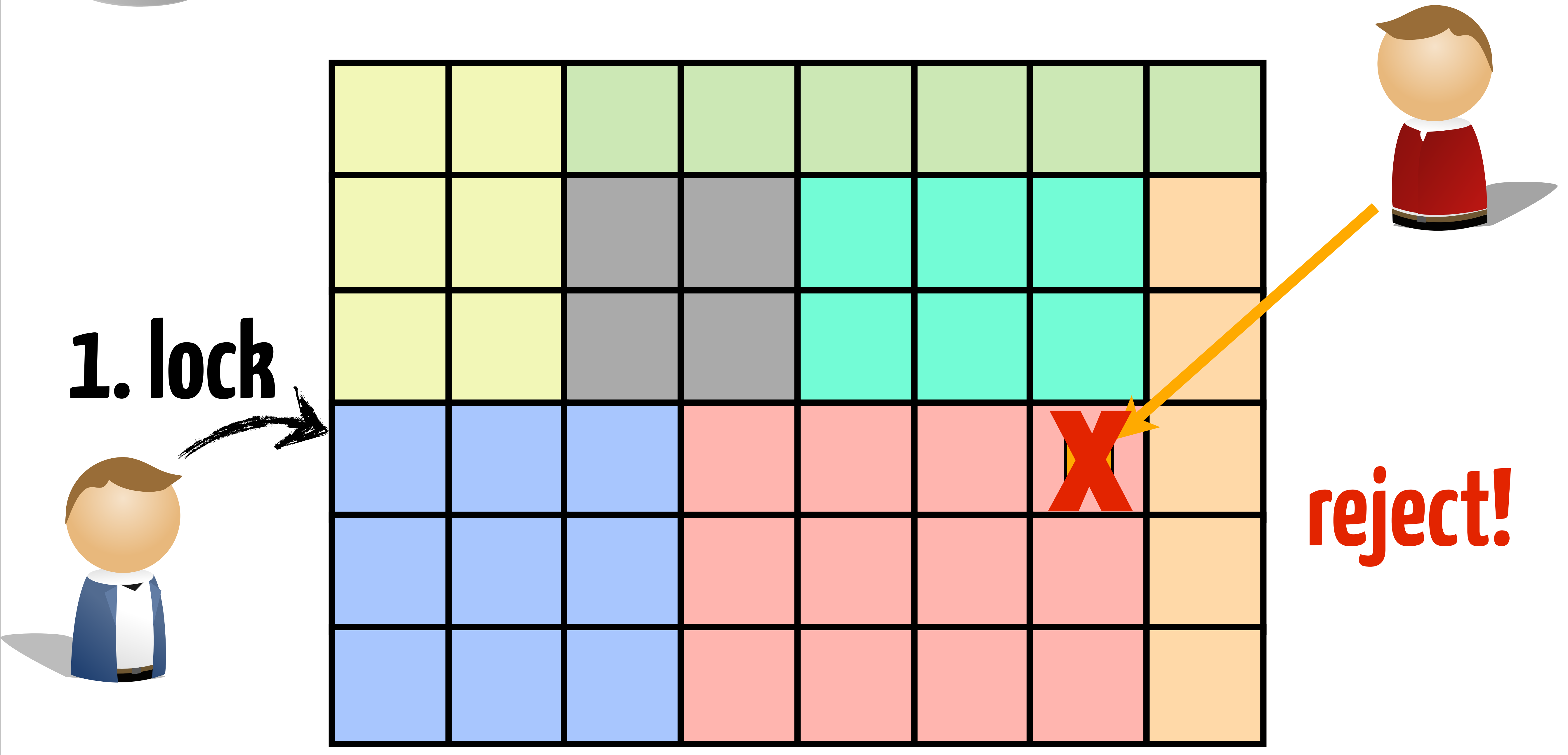
File Locking (old school)

#2: different part of the same file



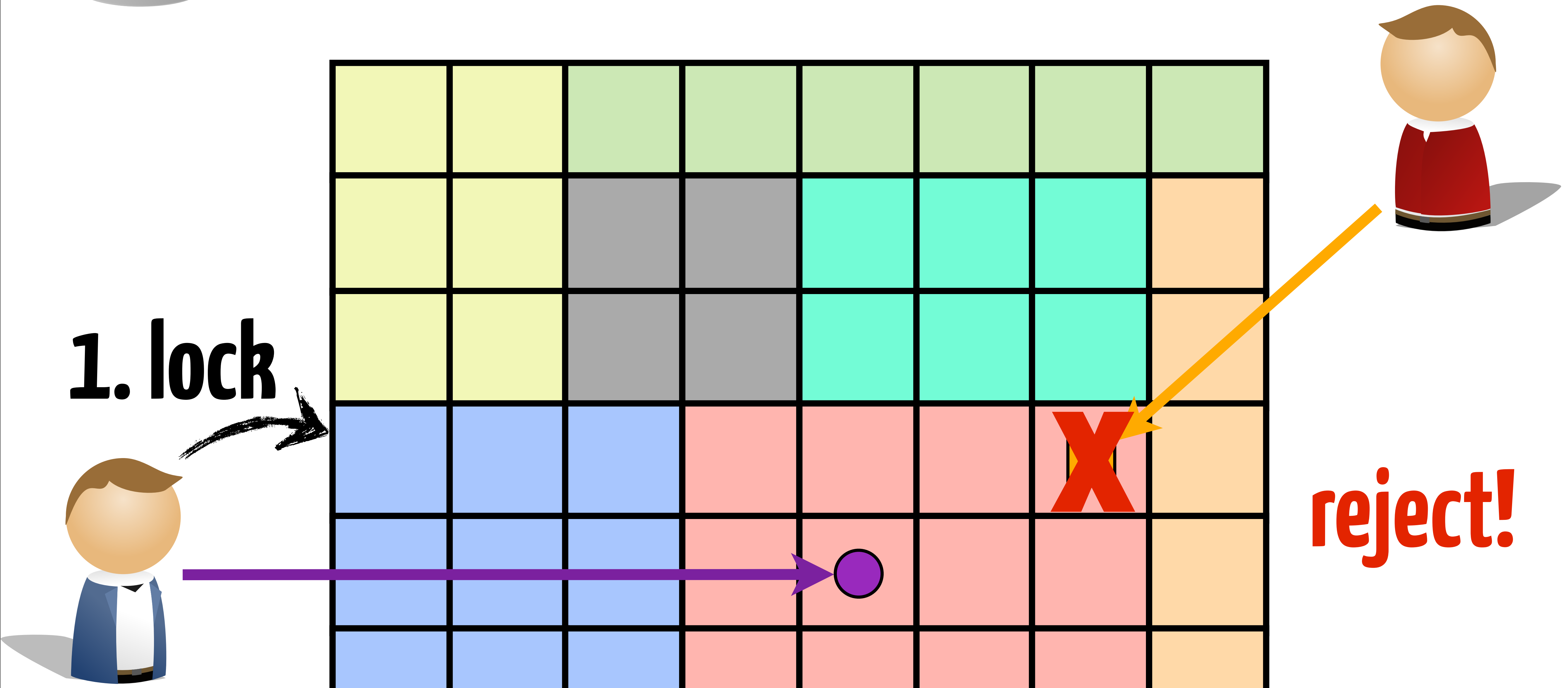
File Locking (old school)

#2: different part of the same file



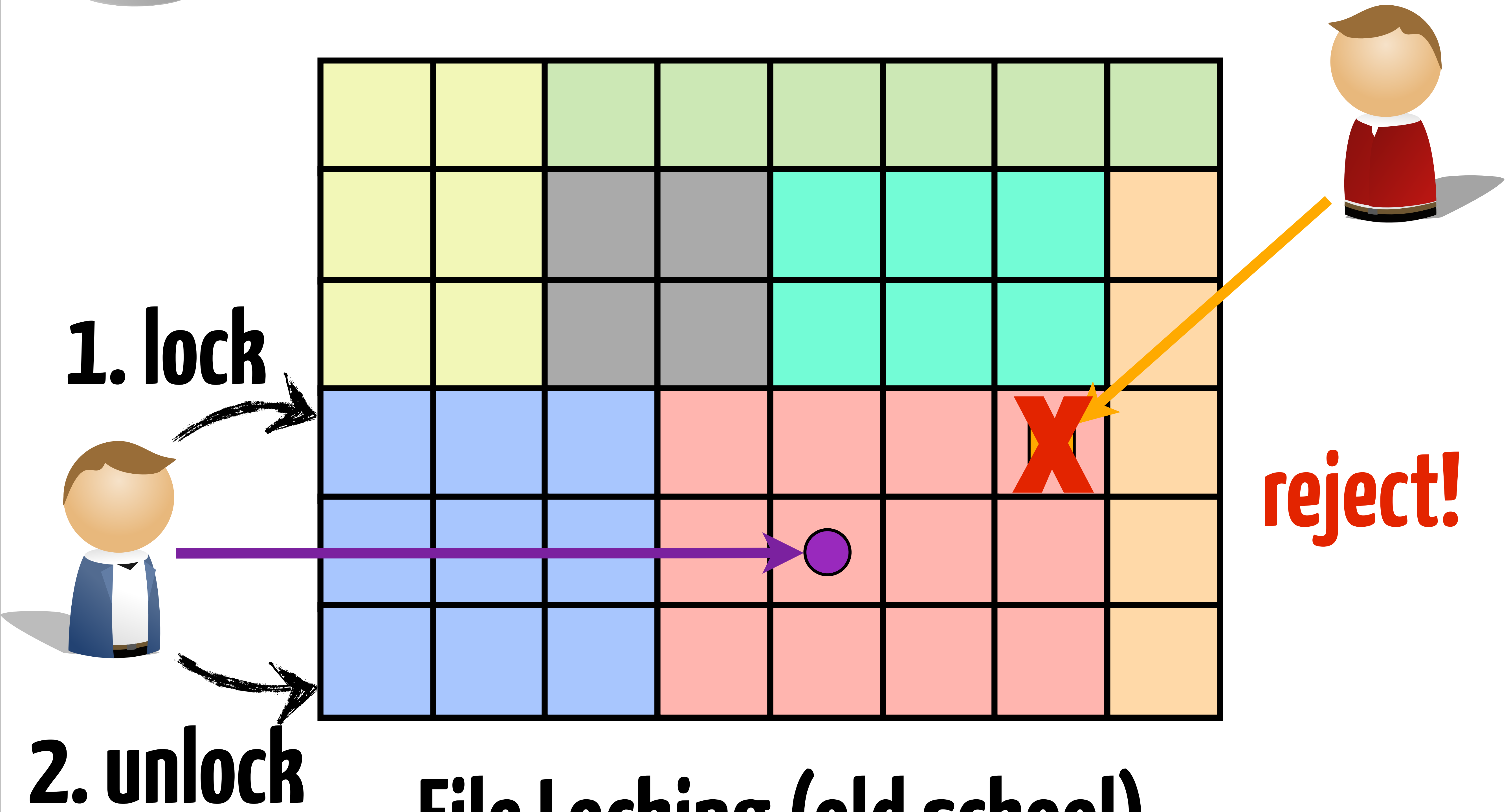
File Locking (old school)

#2: different part of the same file



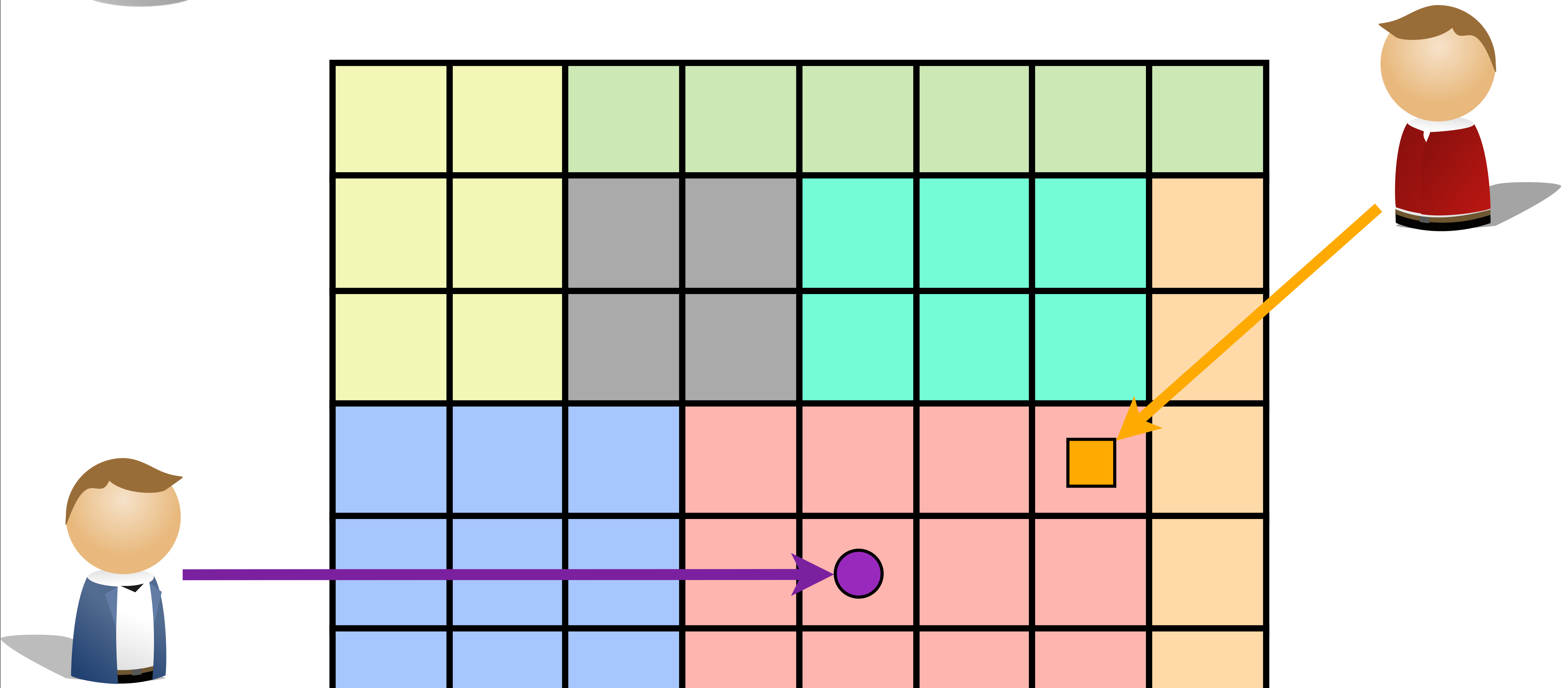
File Locking (old school)

#2: different part of the same file

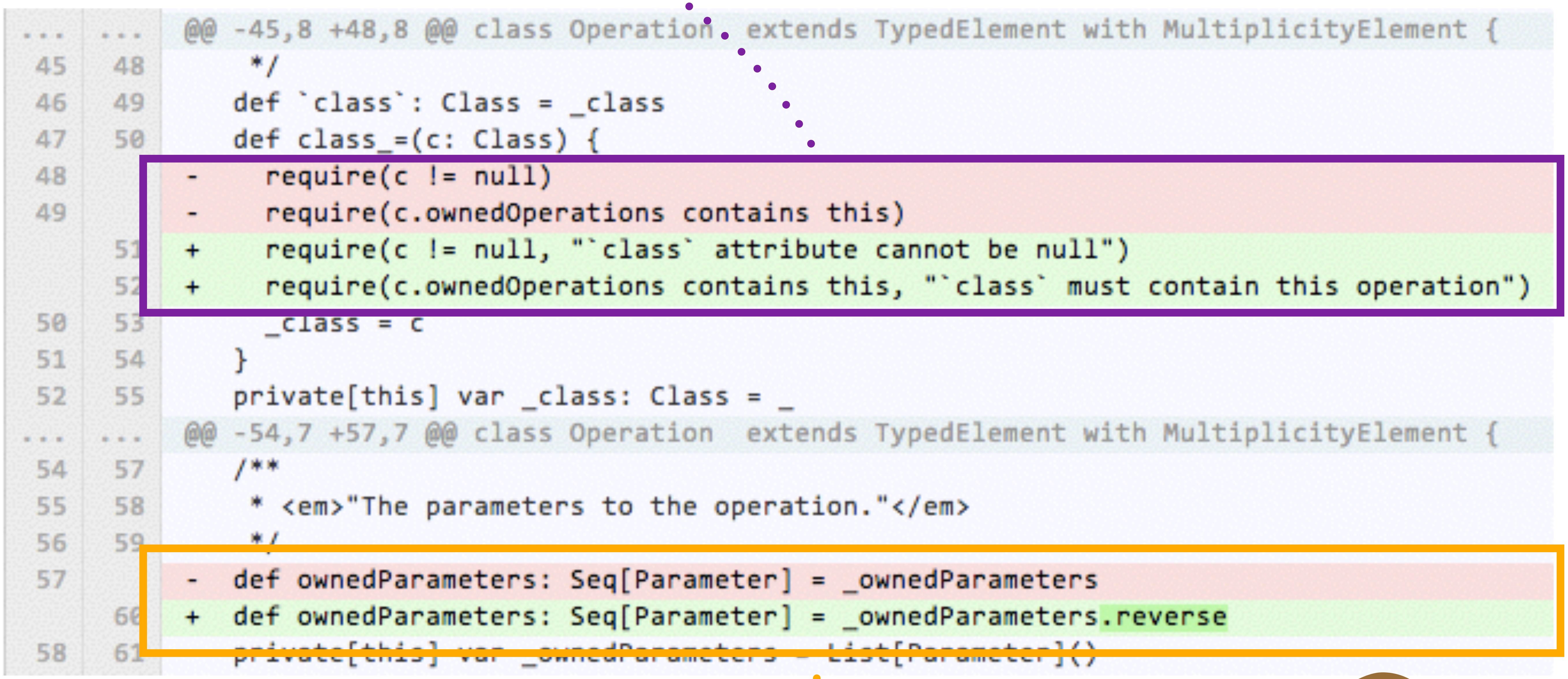


File Locking (old school)

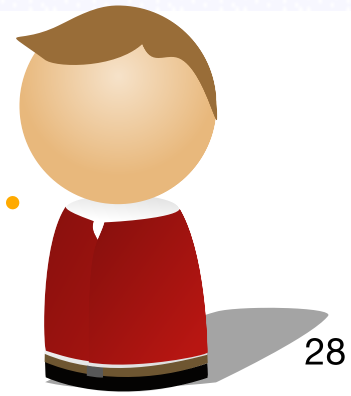
#2: different part of the same file

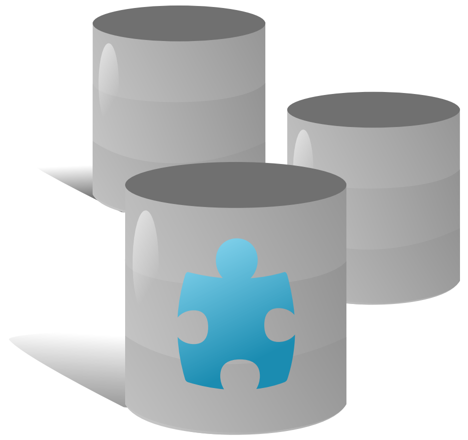


Automatic merge

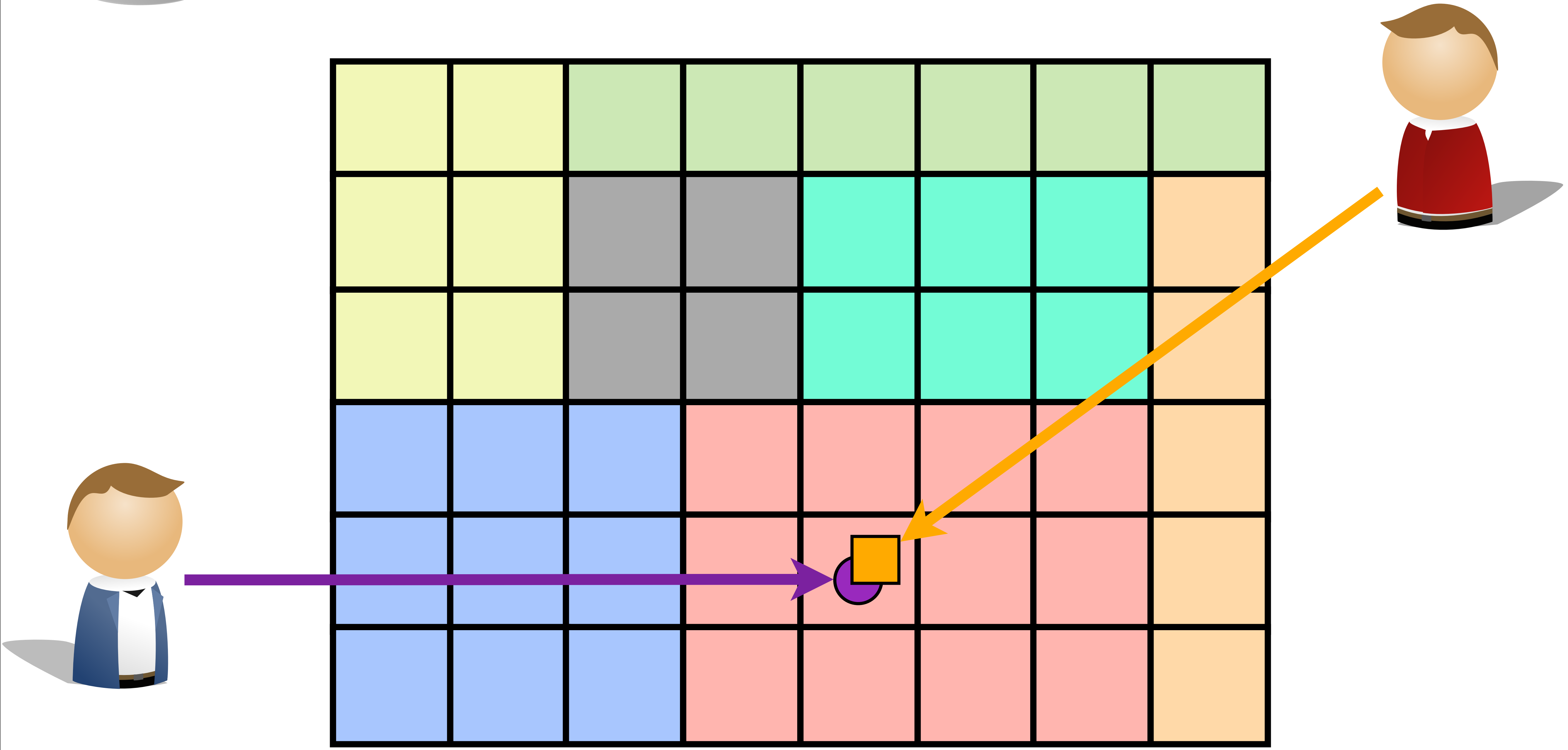


```
@@ -45,8 +48,8 @@ class Operation  extends TypedElement with MultiplicityElement {
       */
     def `class`: Class = _class
     def class_=(c: Class) {
-      require(c != null)
-      require(c.ownedOperations contains this)
+      require(c != null, "`class` attribute cannot be null")
+      require(c.ownedOperations contains this, "`class` must contain this operation")
       _class = c
     }
     private[this] var _class: Class = _
@@ -54,7 +57,7 @@ class Operation  extends TypedElement with MultiplicityElement {
     /**
      * <em>"The parameters to the operation."</em>
      */
-    def ownedParameters: Seq[Parameter] = _ownedParameters
+    def ownedParameters: Seq[Parameter] = _ownedParameters.reverse
     private[this] var _ownedParameters = List[Parameter]()
```

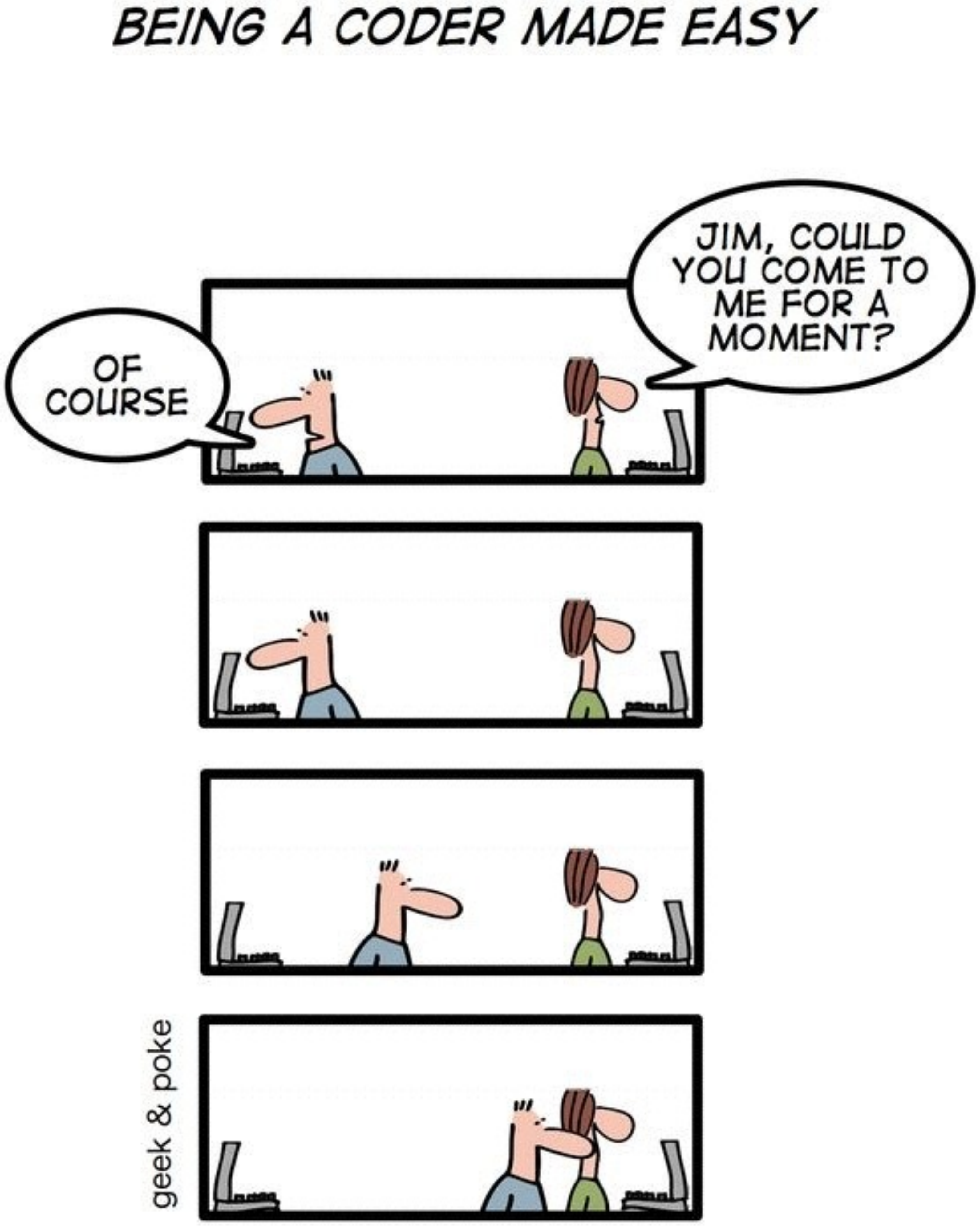


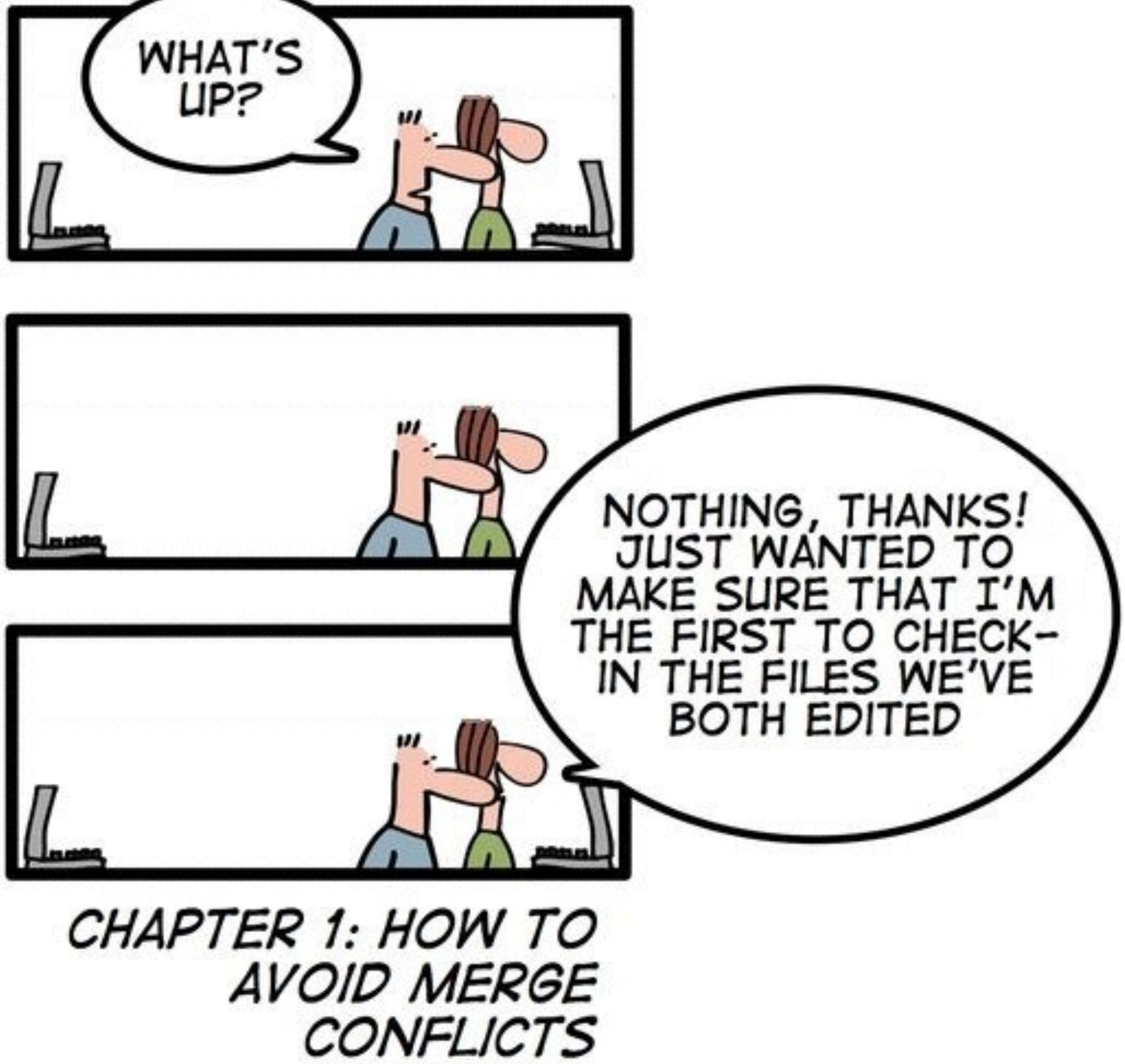


#3: same part of the same file



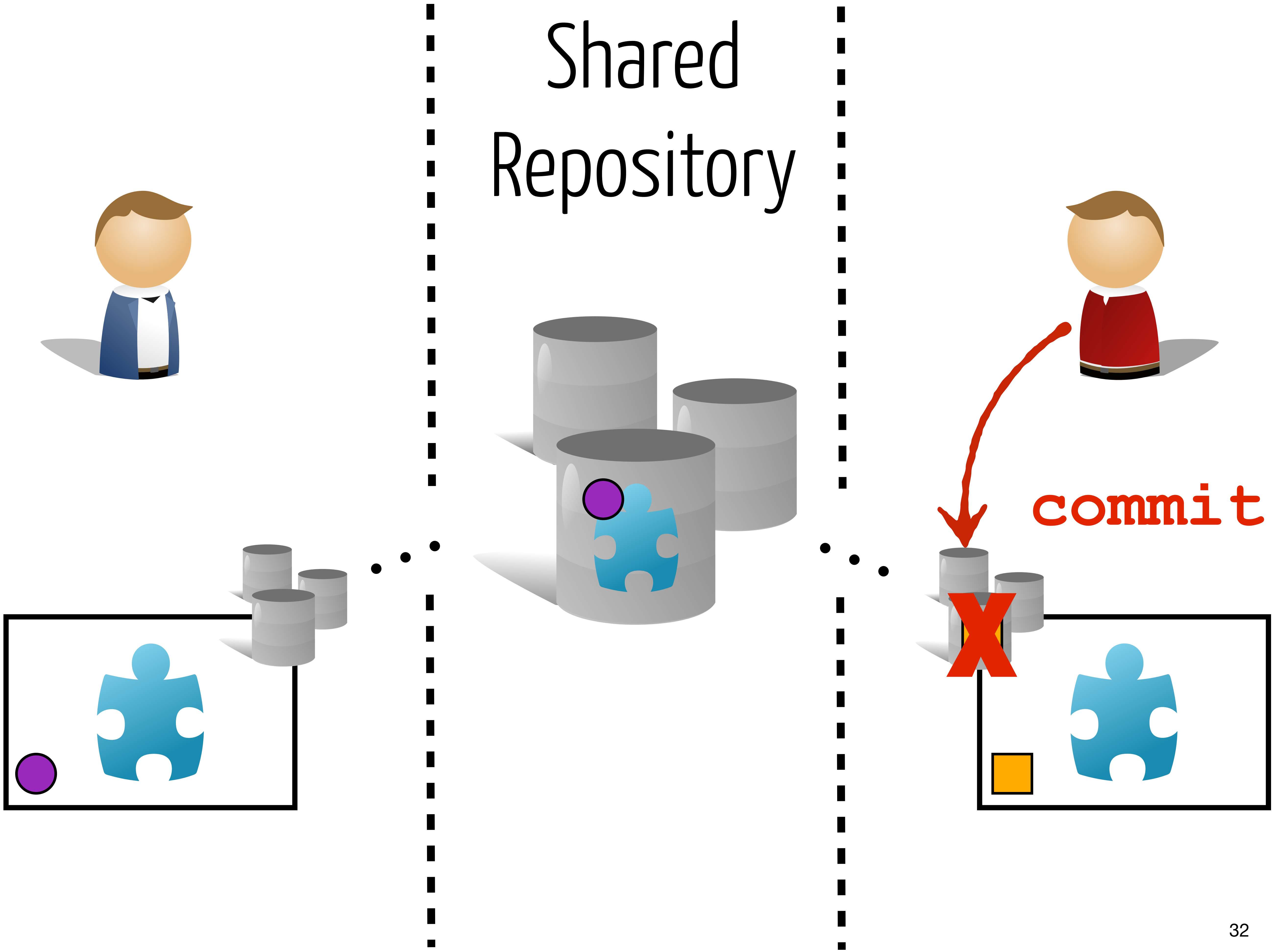
Conflict!


BEING A CODER MADE EASY
JIM, COULD YOU COME TO ME FOR A MOMENT?
OF COURSE
geek & poke

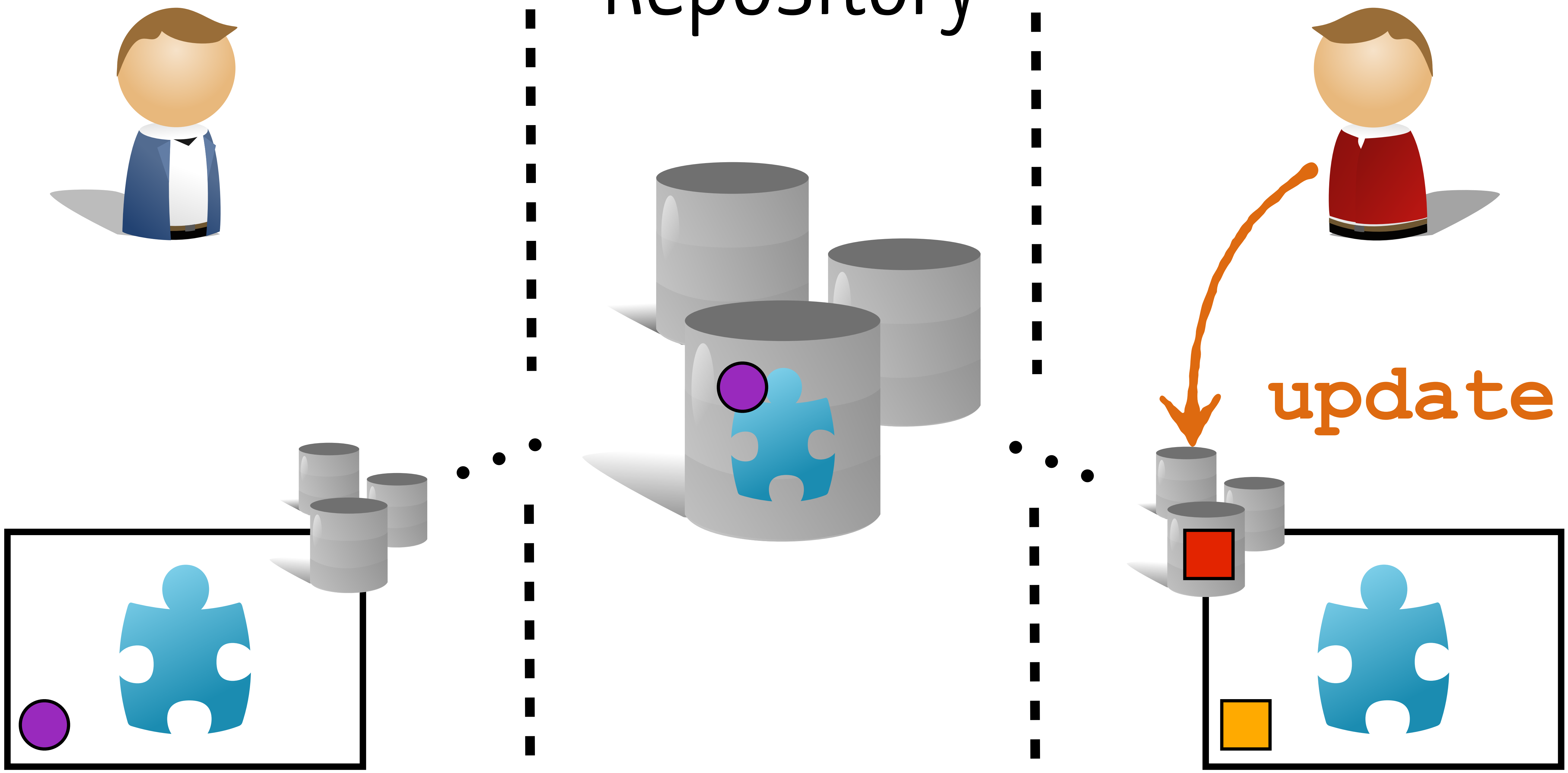

WHAT'S UP?
NOTHING, THANKS! JUST WANTED TO MAKE SURE THAT I'M THE FIRST TO CHECK-IN THE FILES WE'VE BOTH EDITED
CHAPTER 1: HOW TO AVOID MERGE CONFLICTS

http://geekandpoke.typepad.com/geekandpoke/2010/10/being-a-code-made-easy-chapter-1.html

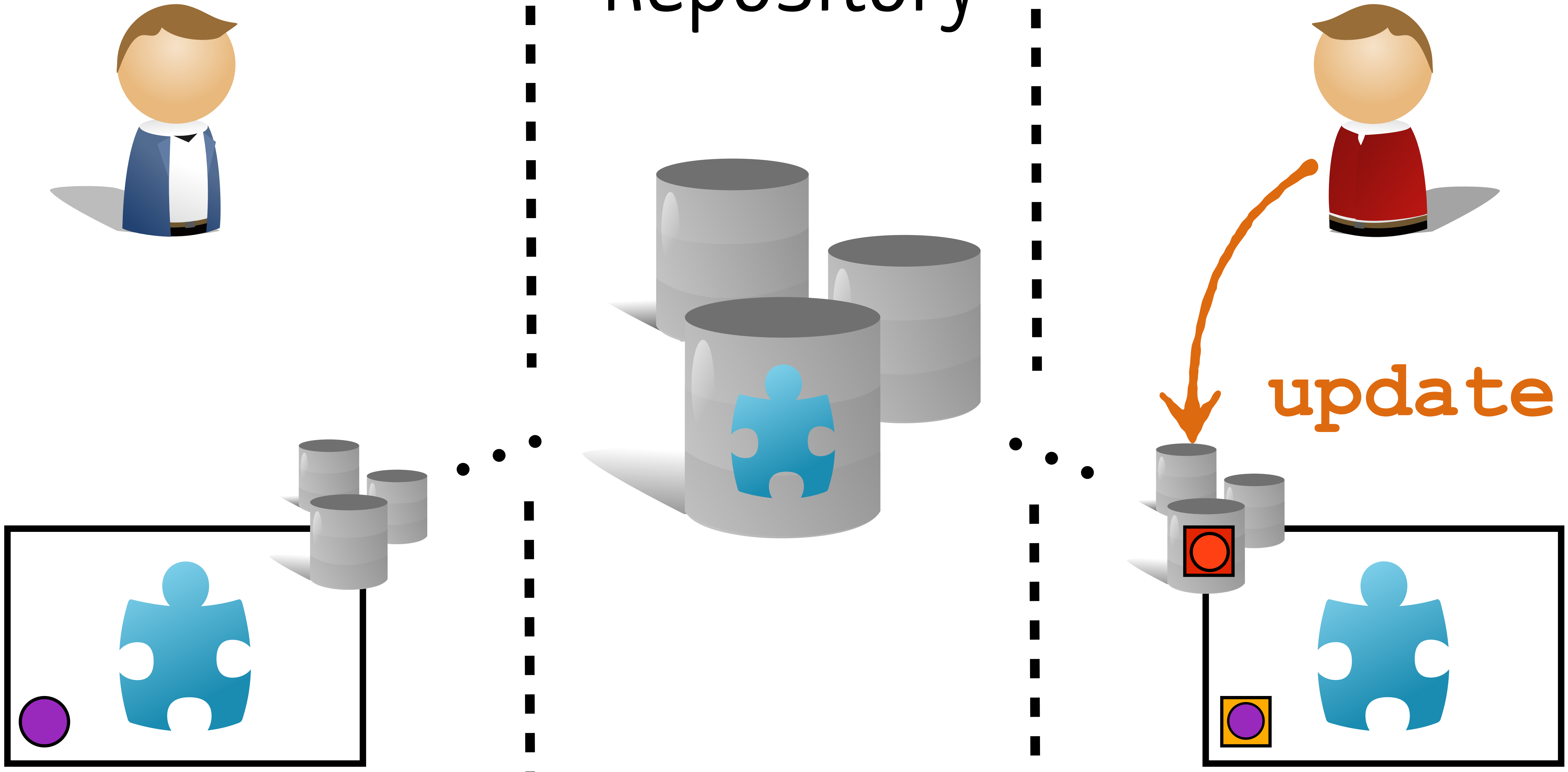
Shared Repository



Shared Repository

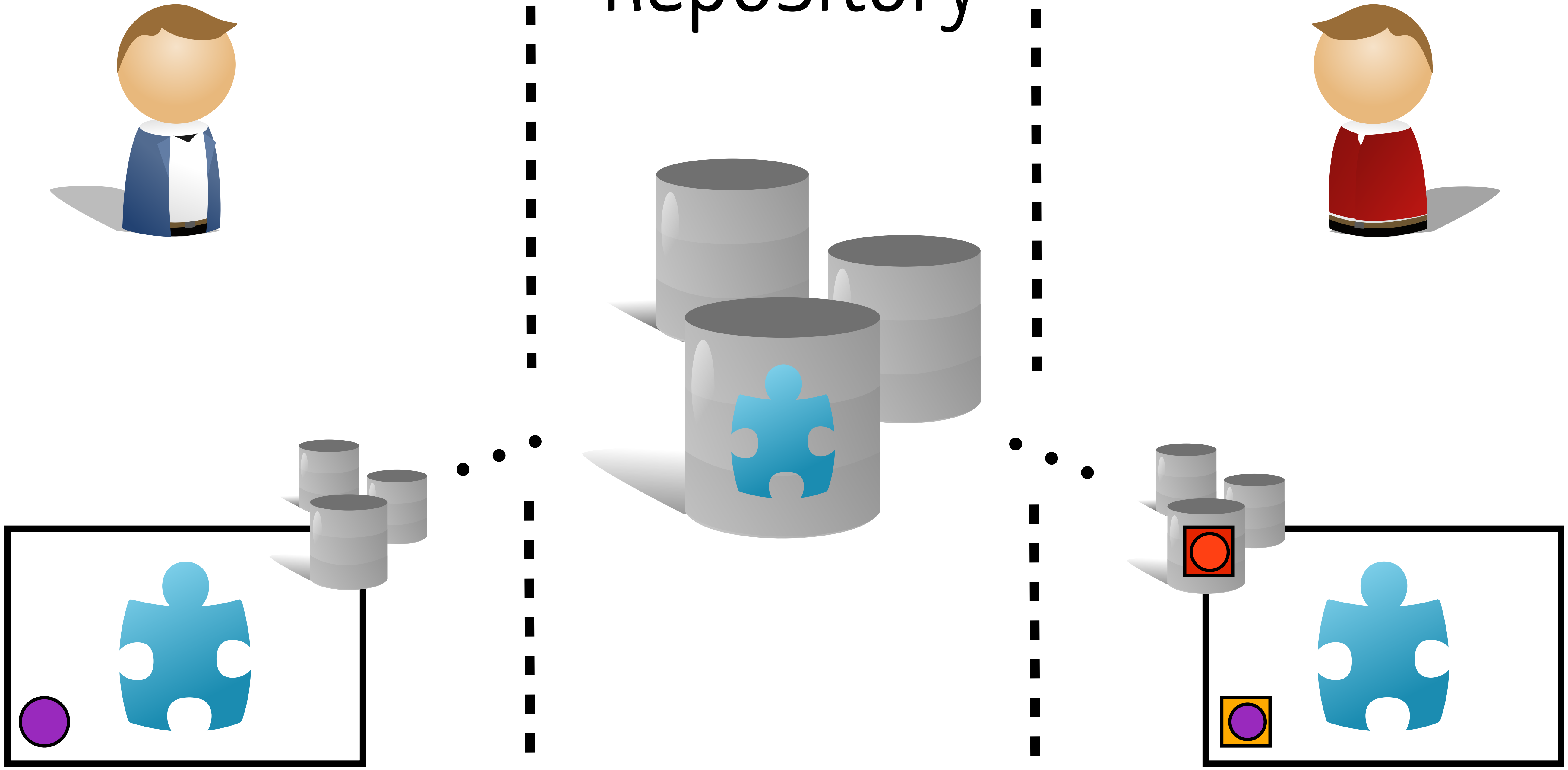


Shared Repository

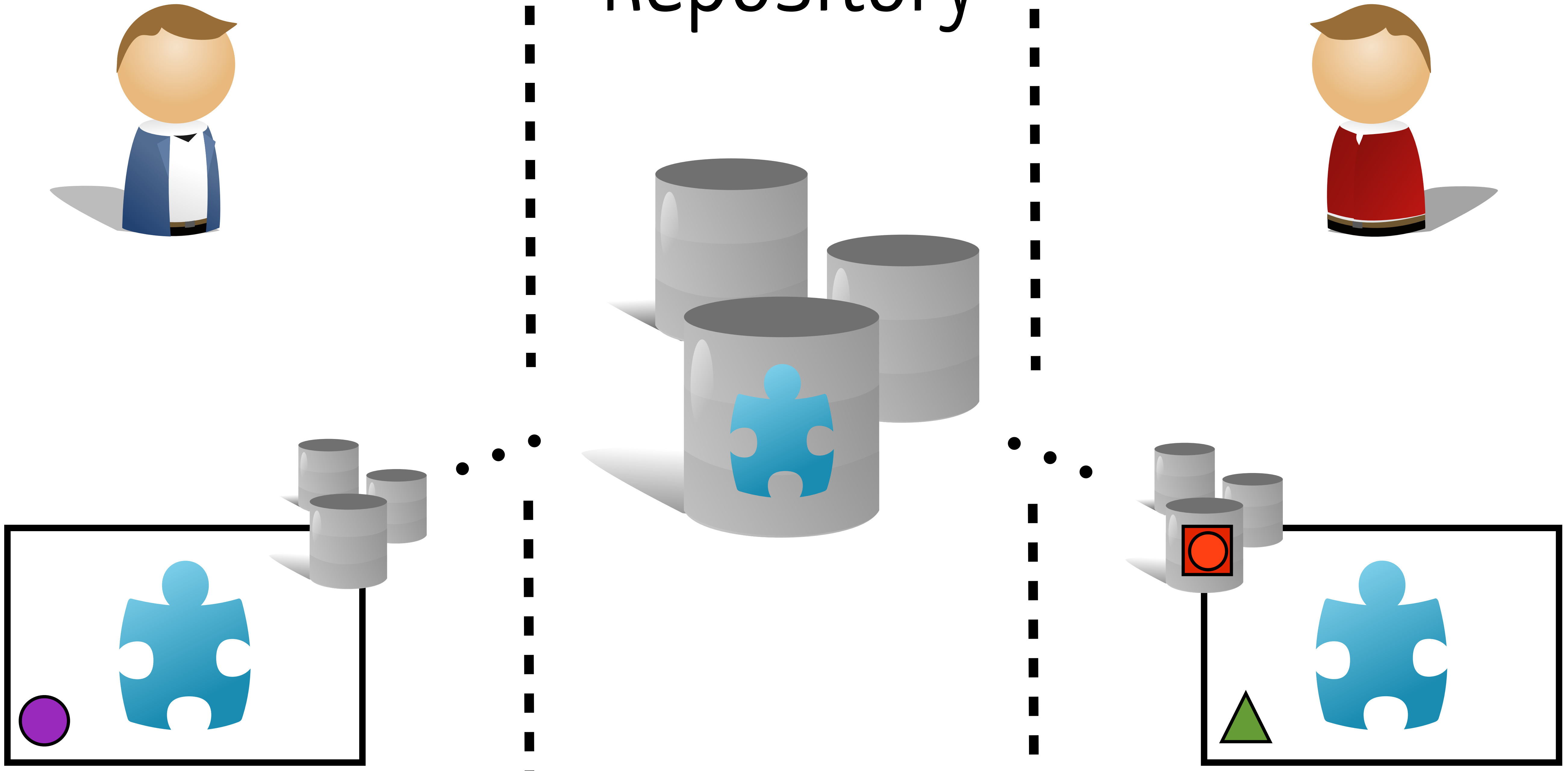


Conflict!

Shared Repository

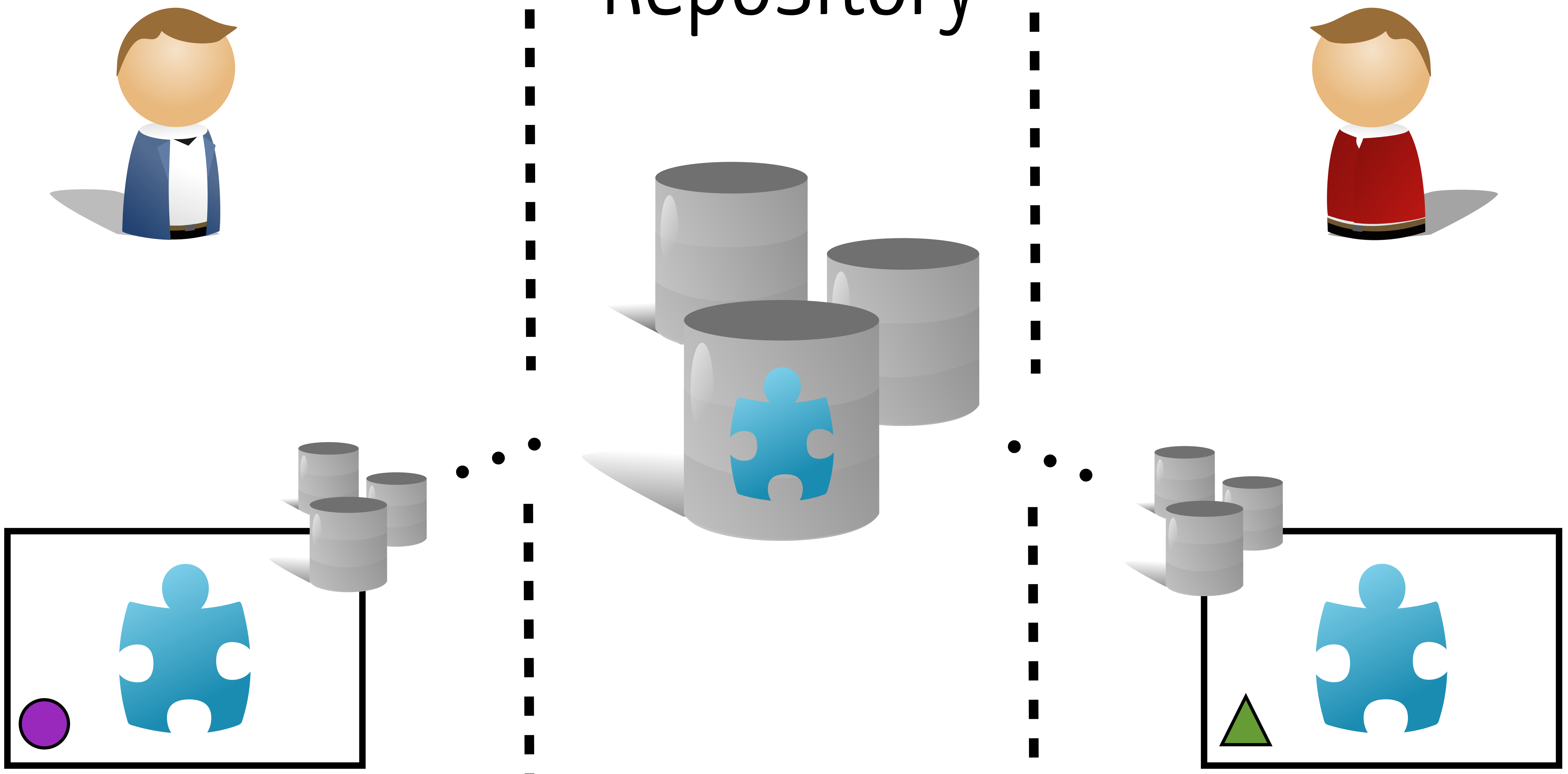


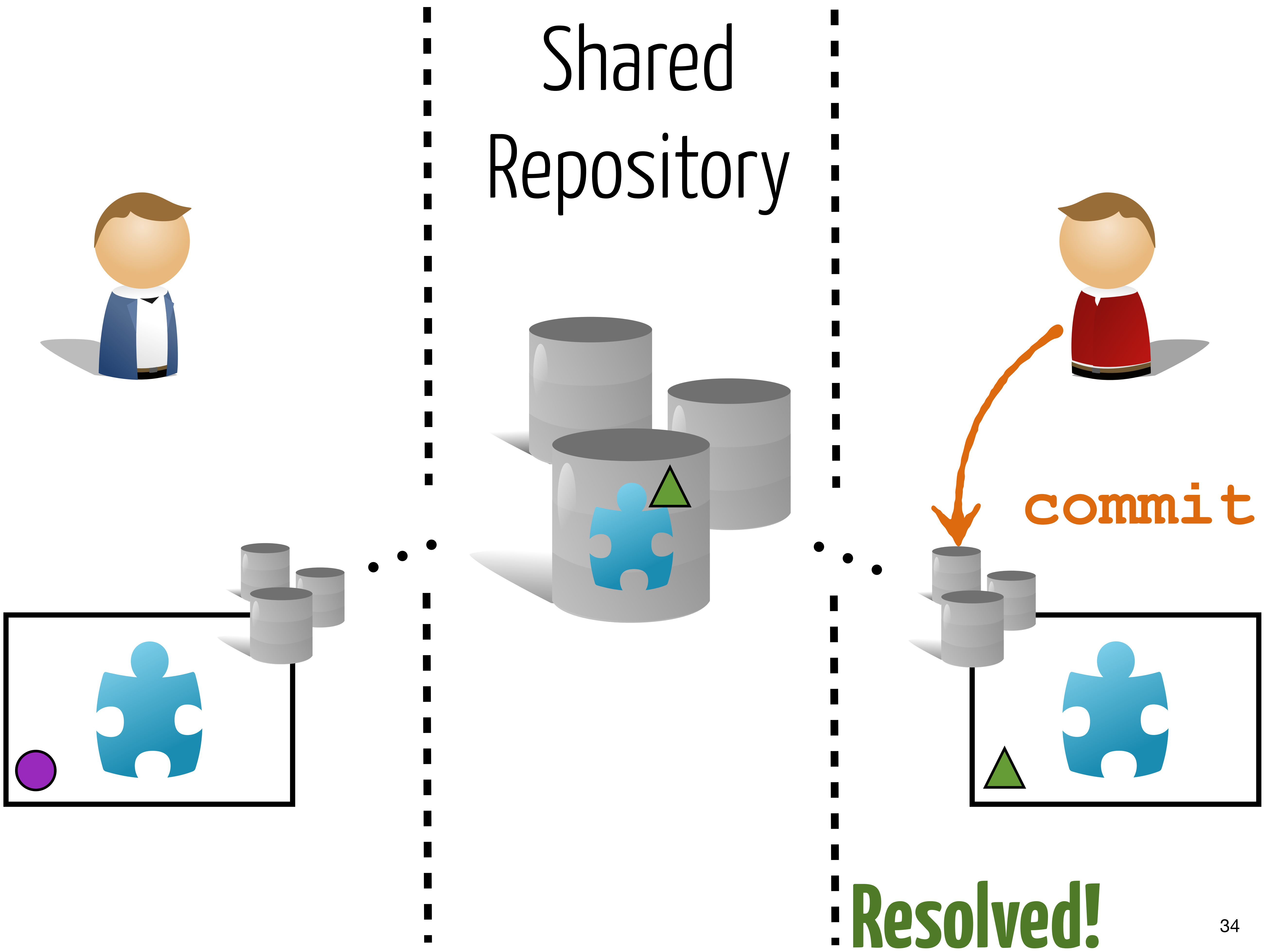
Shared Repository



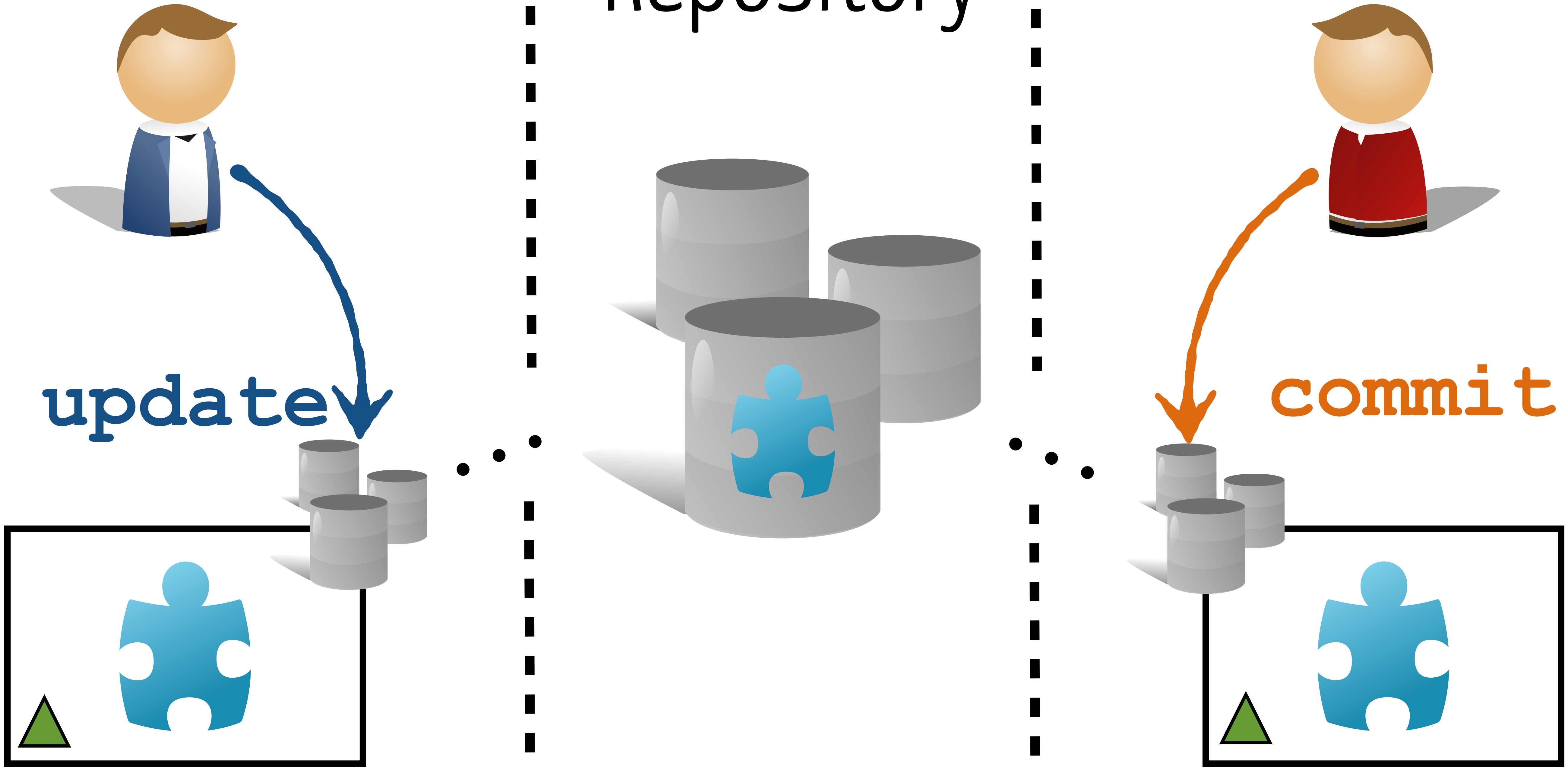
Conflict!

Shared Repository




Shared Repository
commit
Resolved!

Shared Repository



Resolved!

Distributed Model

(e.g., Bazaar, Git)

Centralized = 1 repository

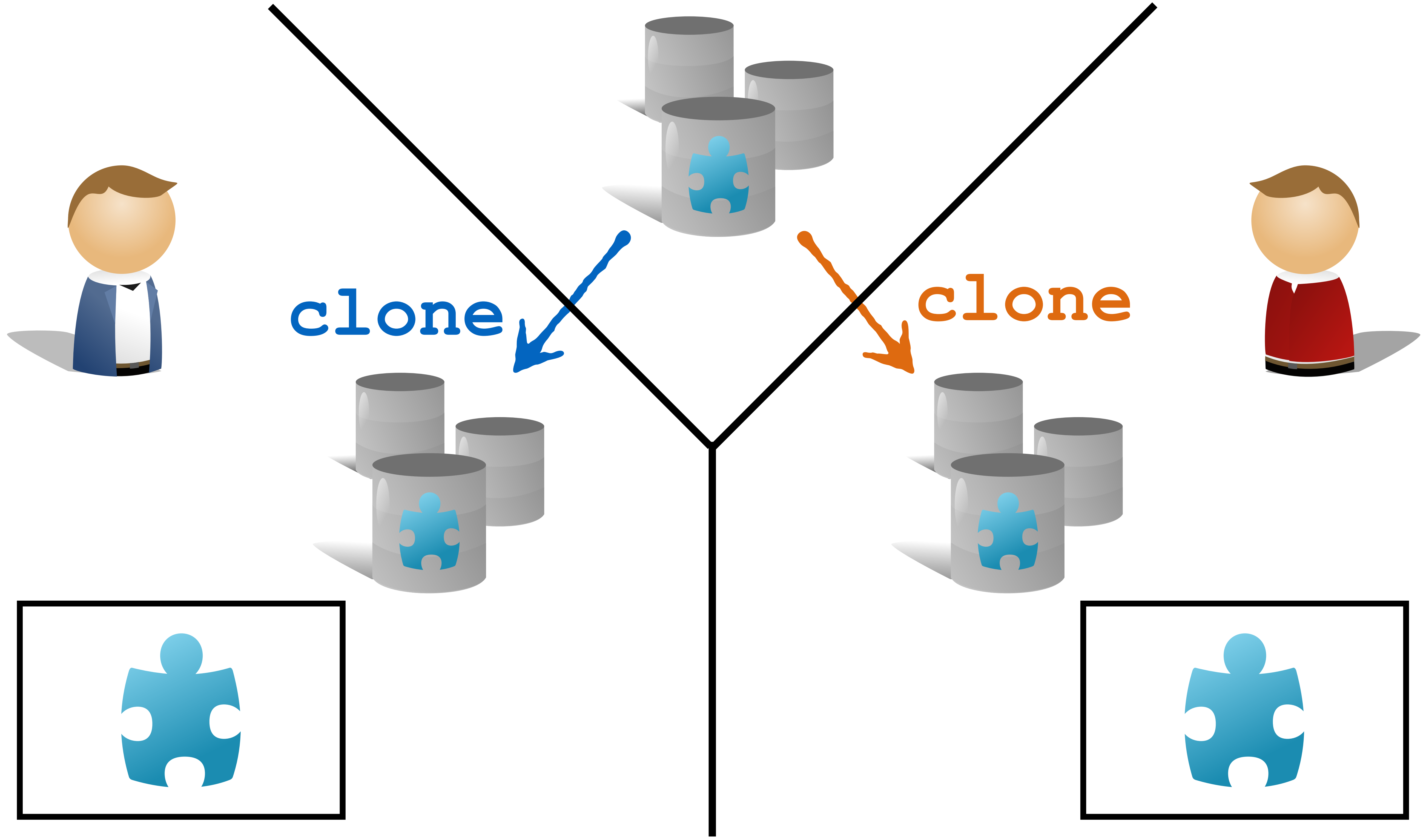
Distributed = N repository

when **N = 1**, Centralized = Distributed

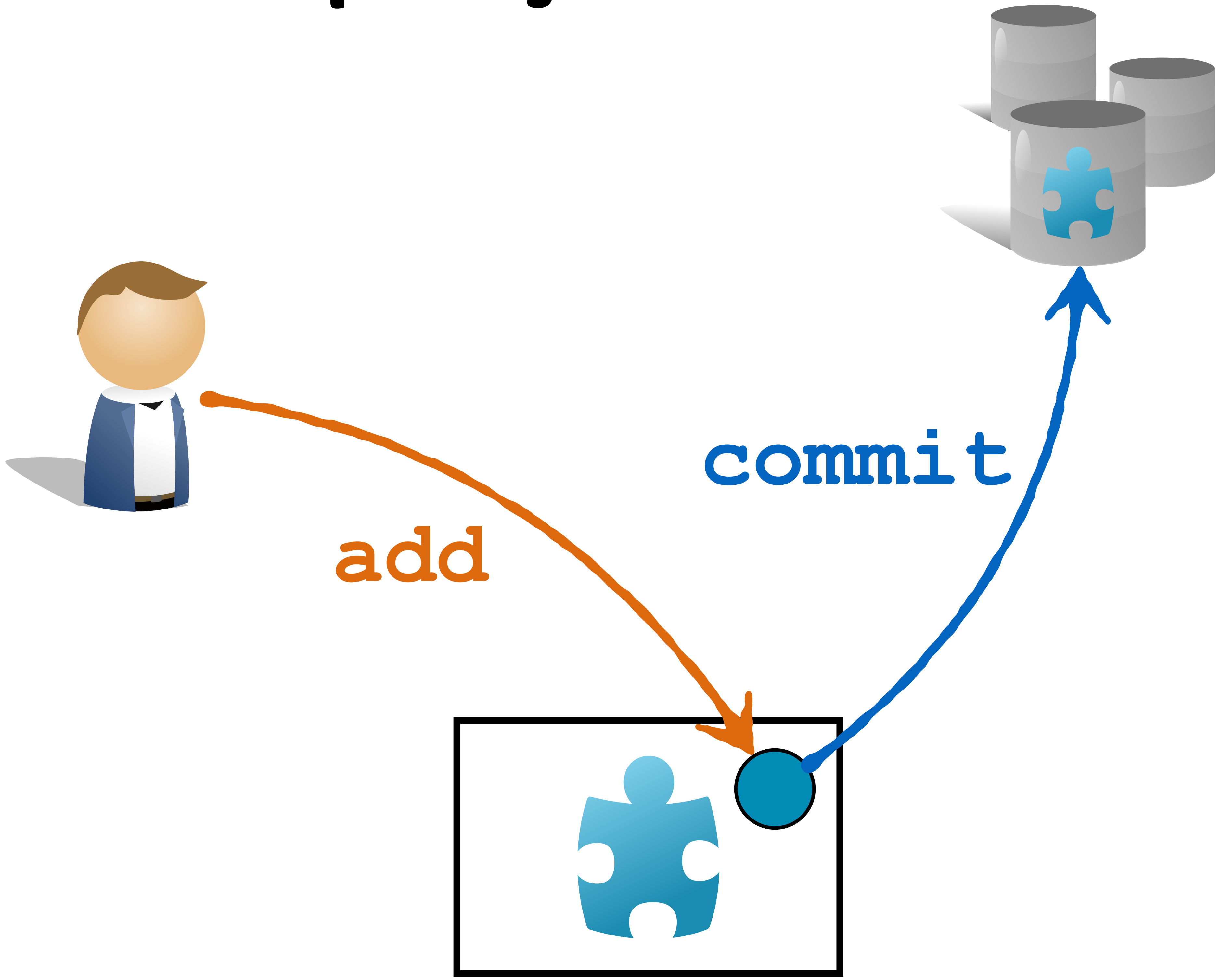
He who can **do more**

can **do less**

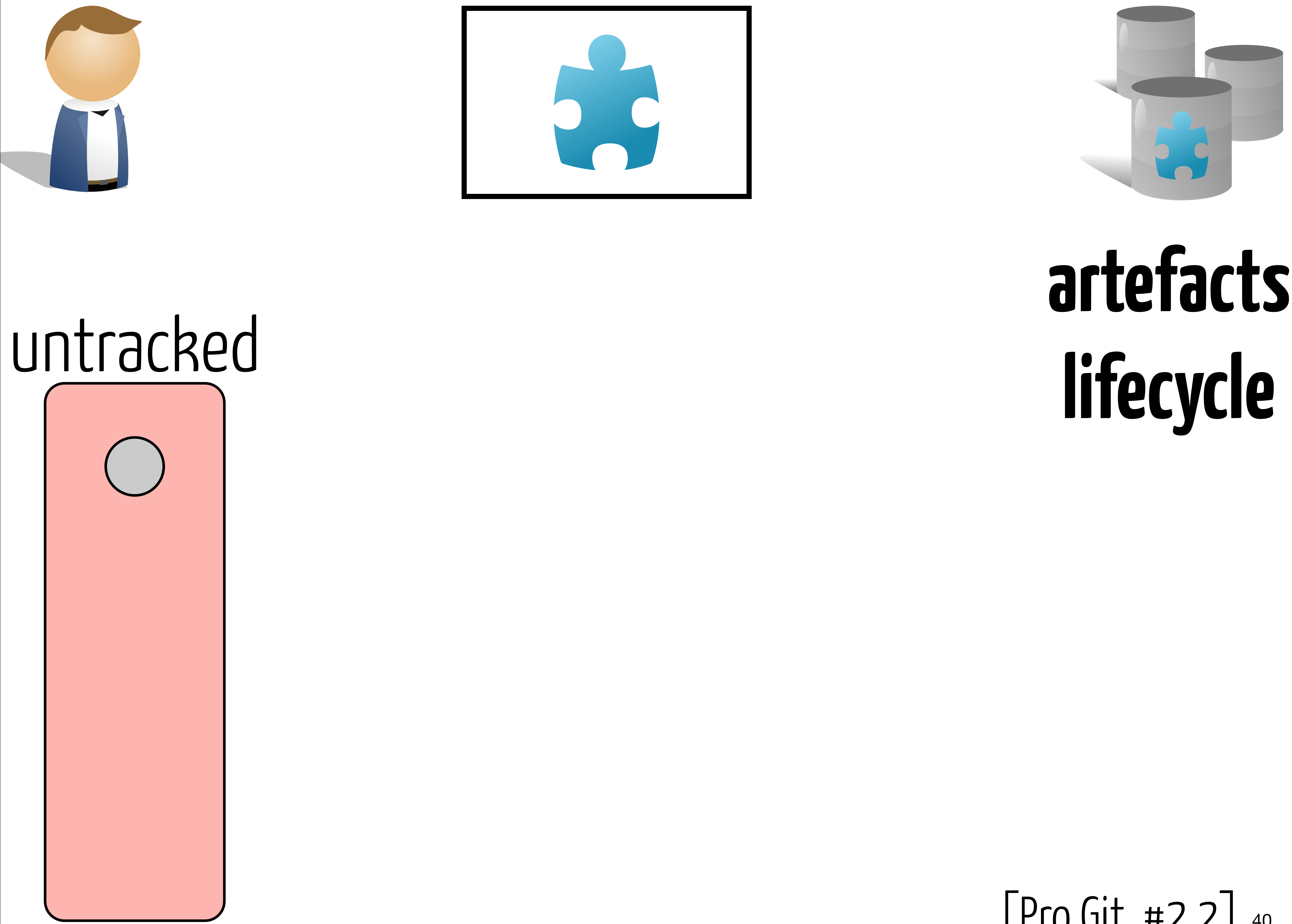
repository



completely offline!

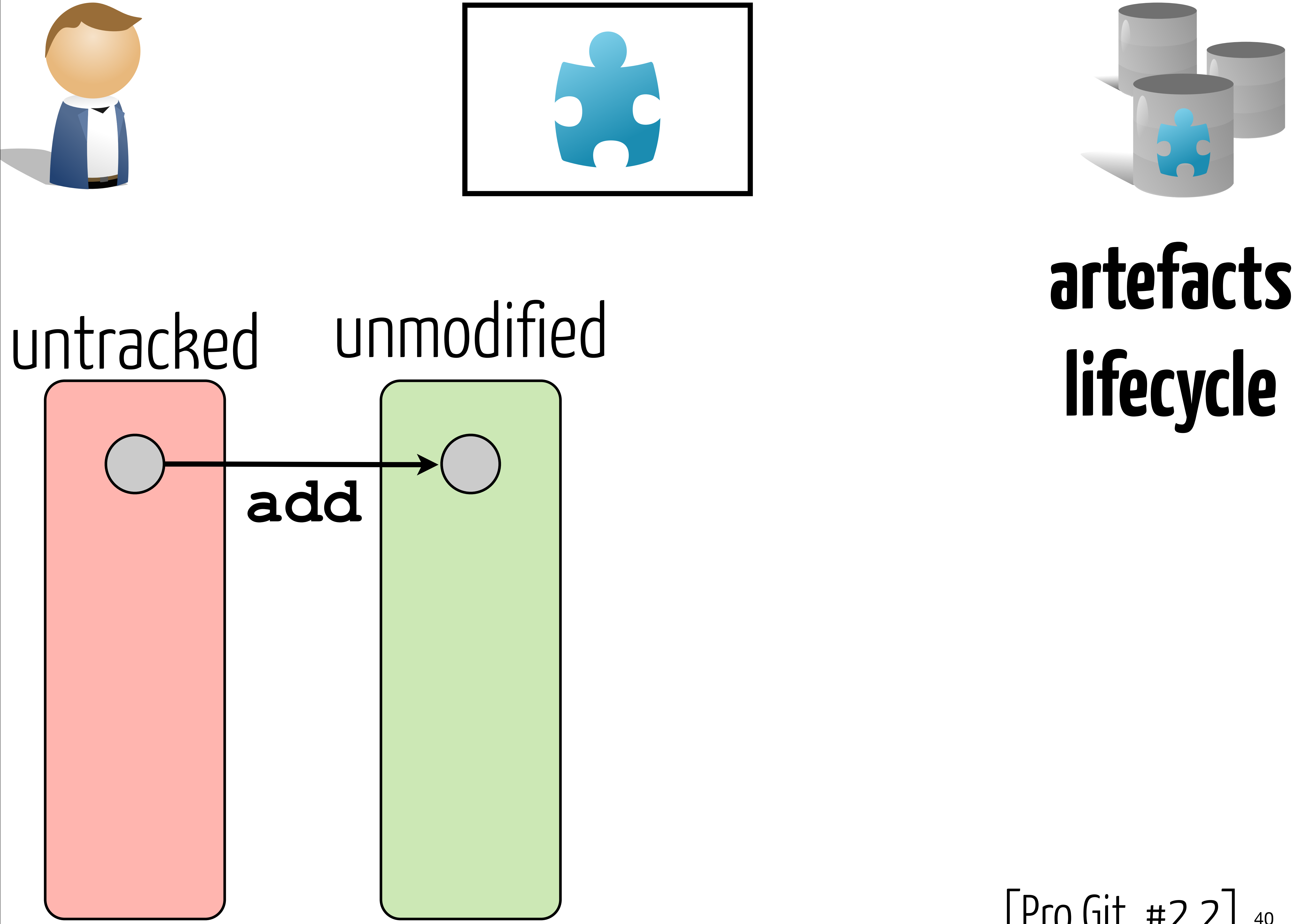


artefacts lifecycle

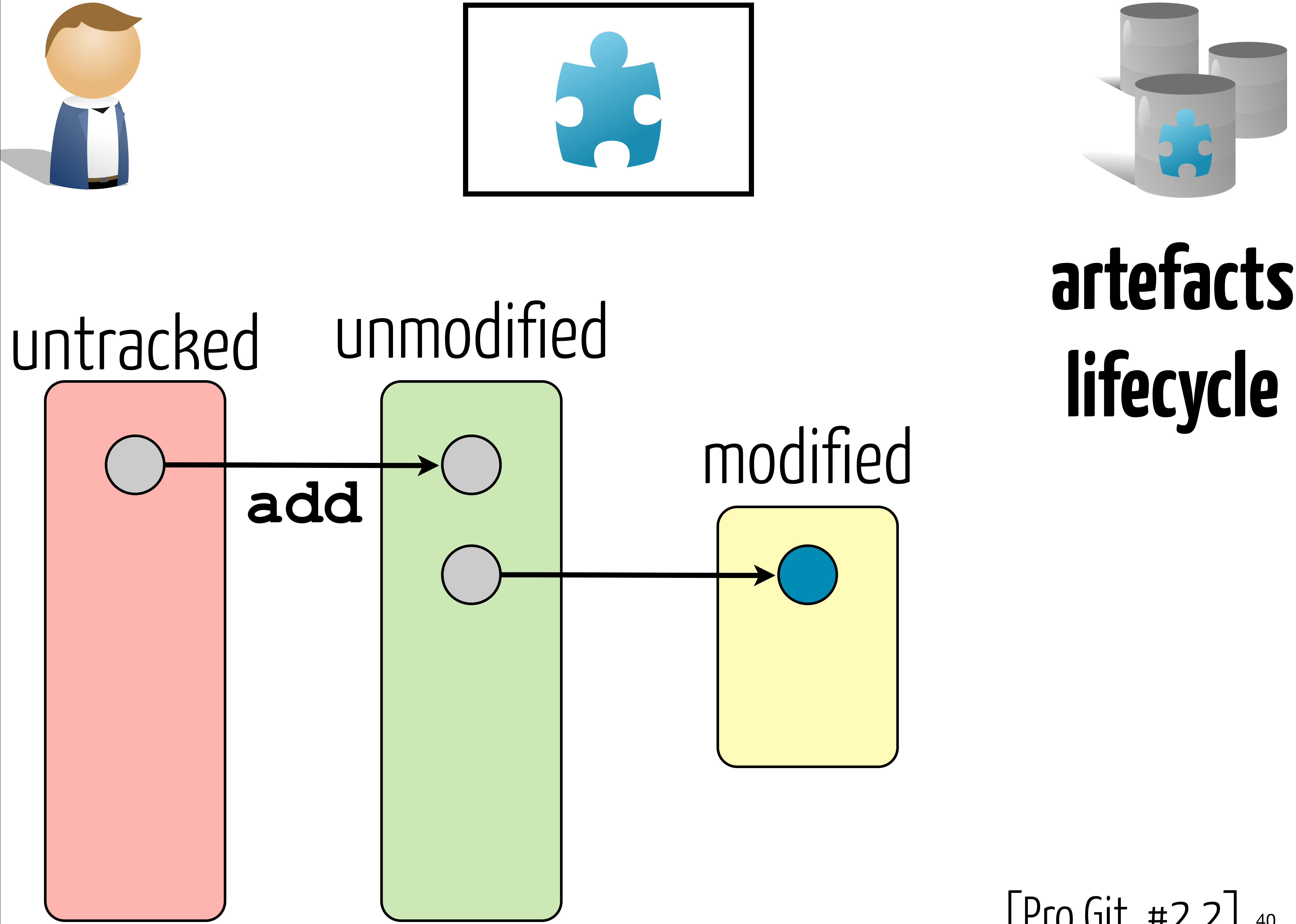


[Pro Git, #2.2]

artefacts lifecycle

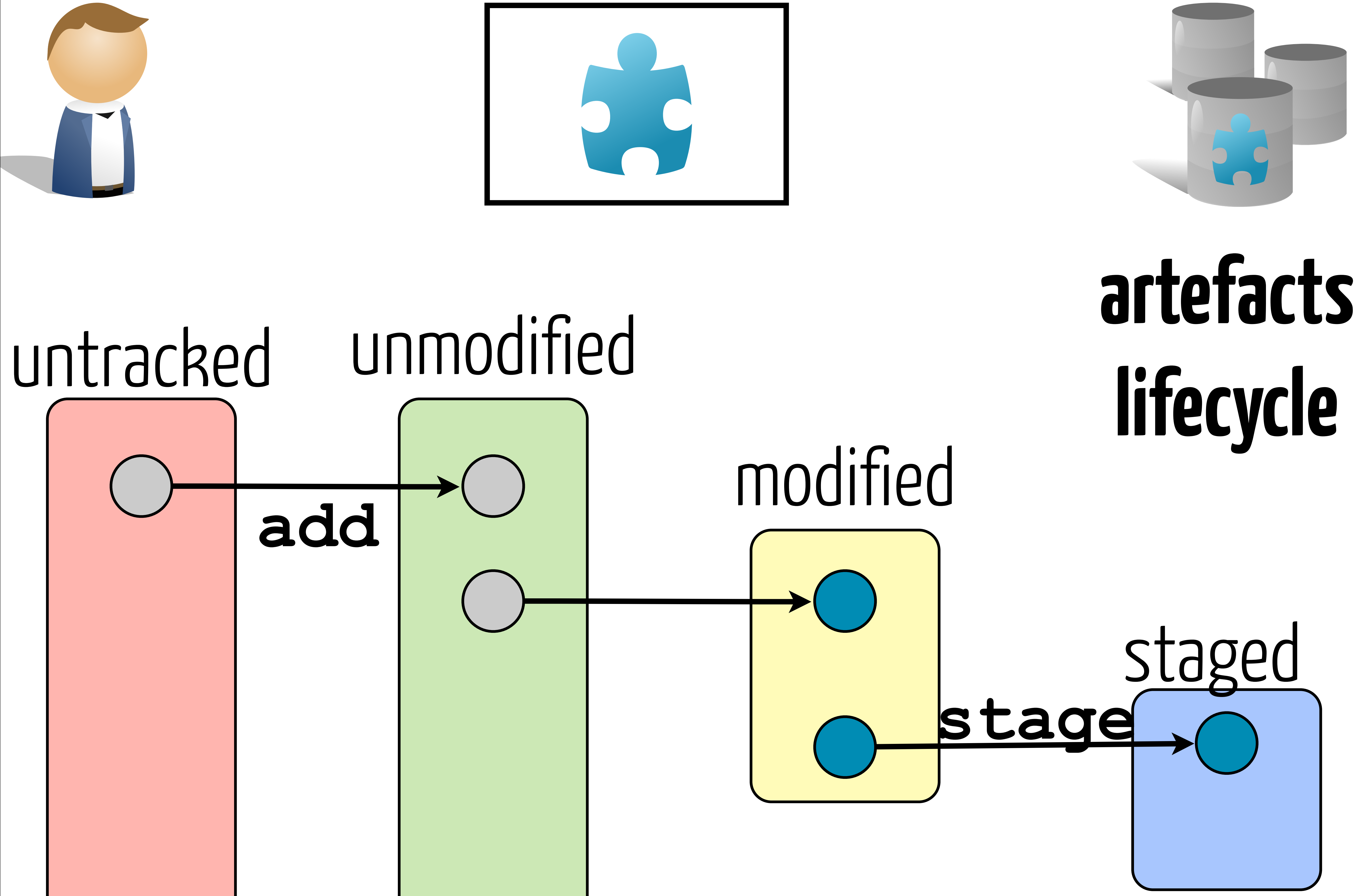


artefacts lifecycle



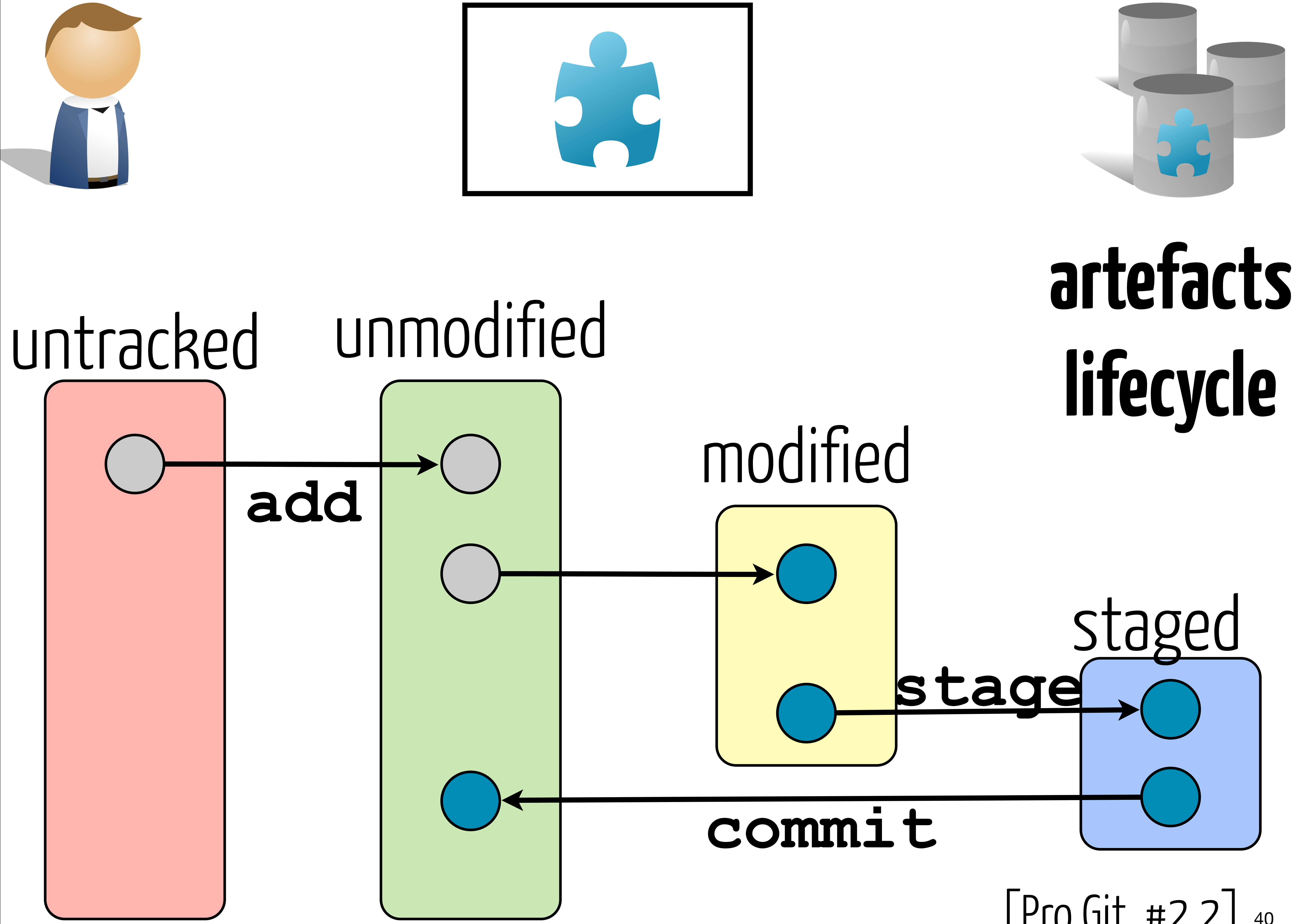
[Pro Git, #2.2]

artefacts lifecycle



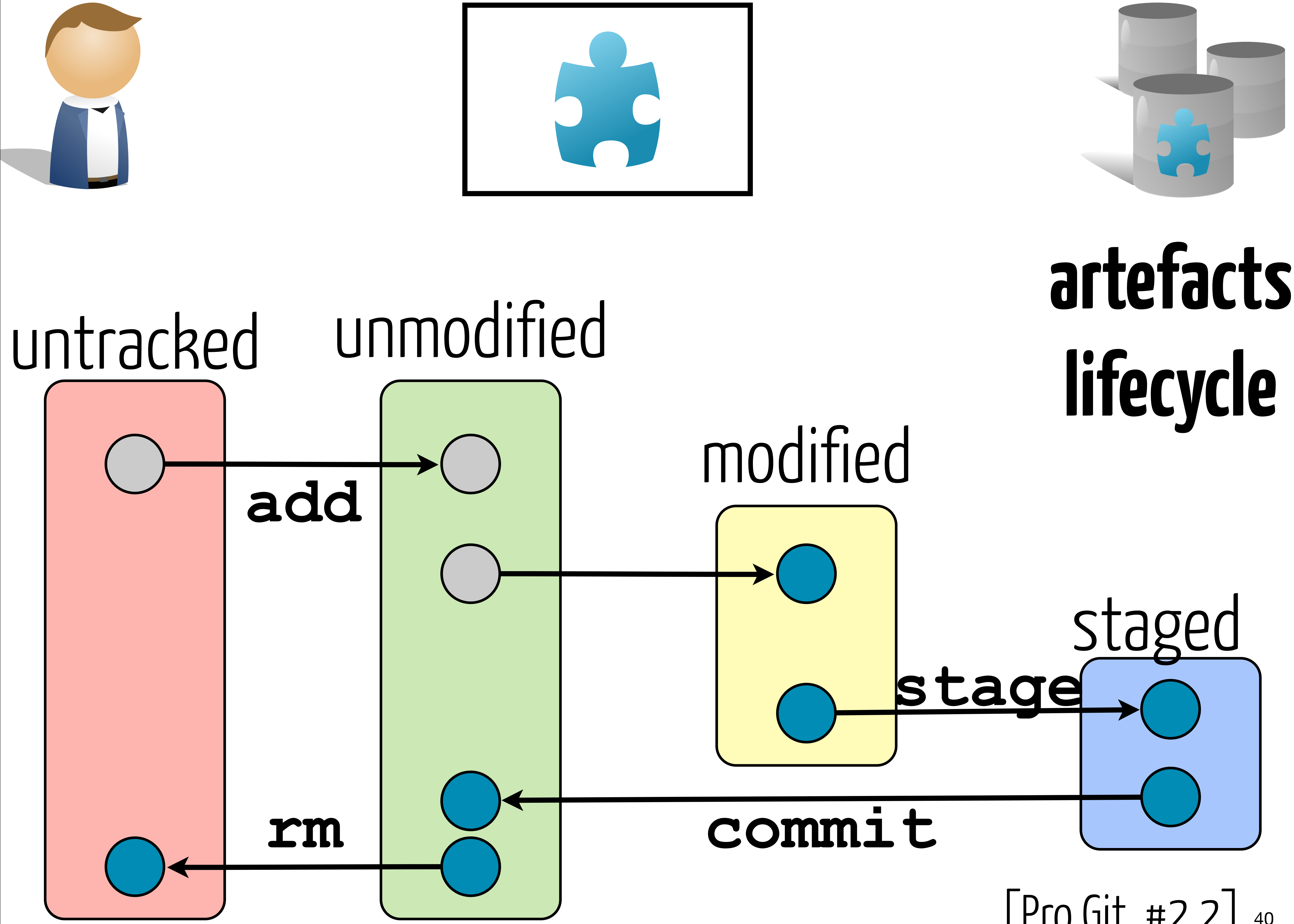
[Pro Git, #2.2]

artefacts lifecycle

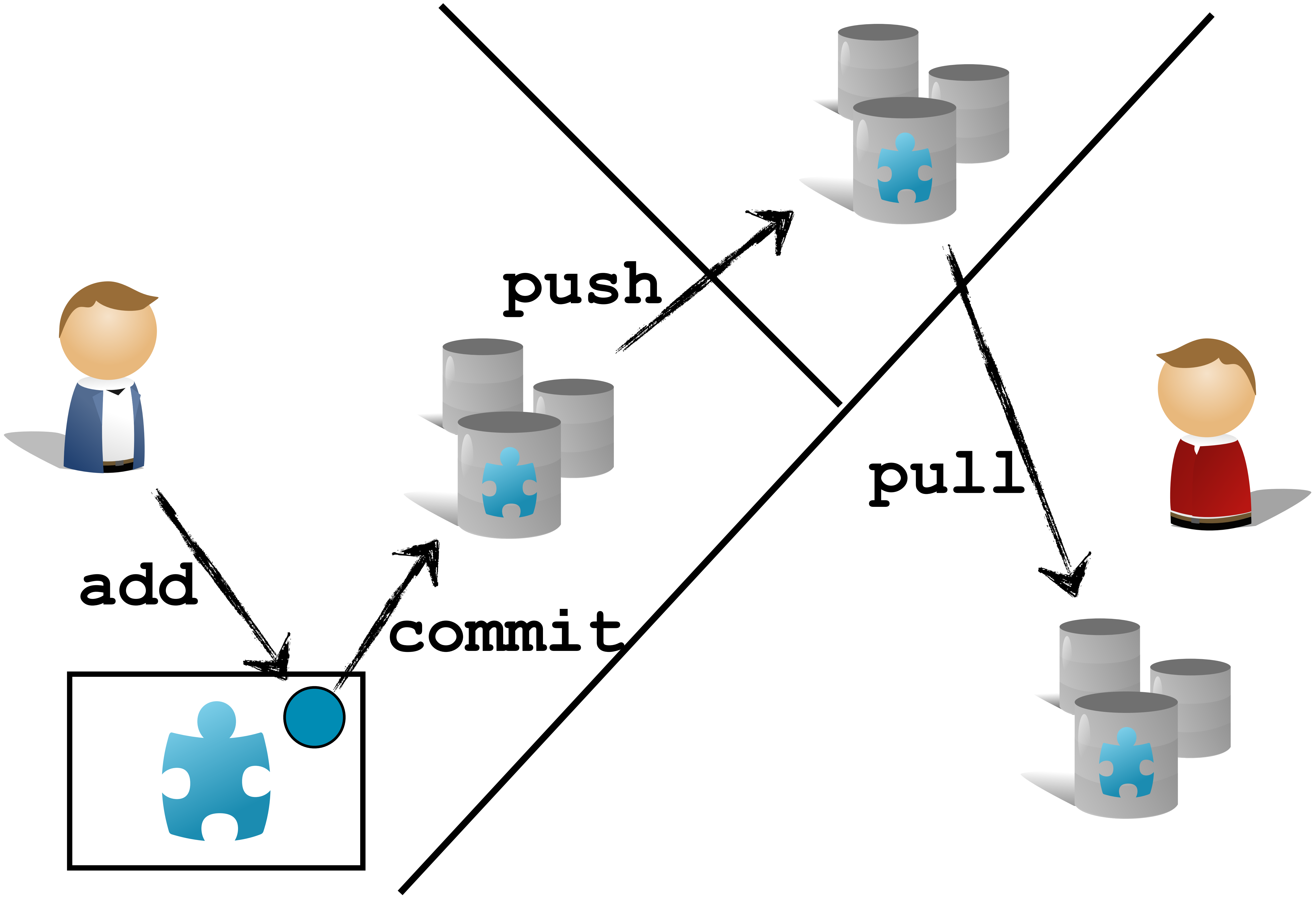


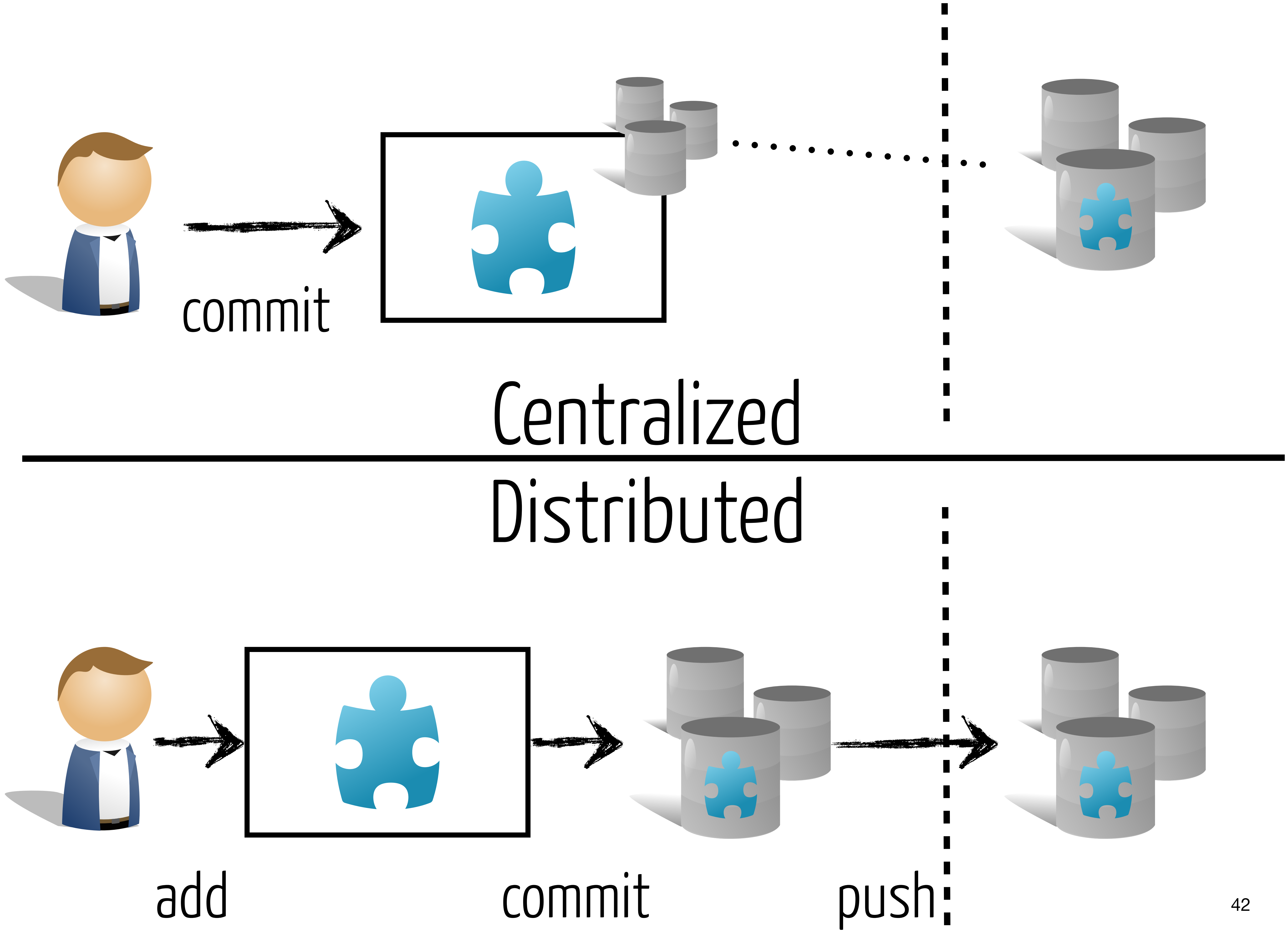
[Pro Git, #2.2]

artefacts lifecycle



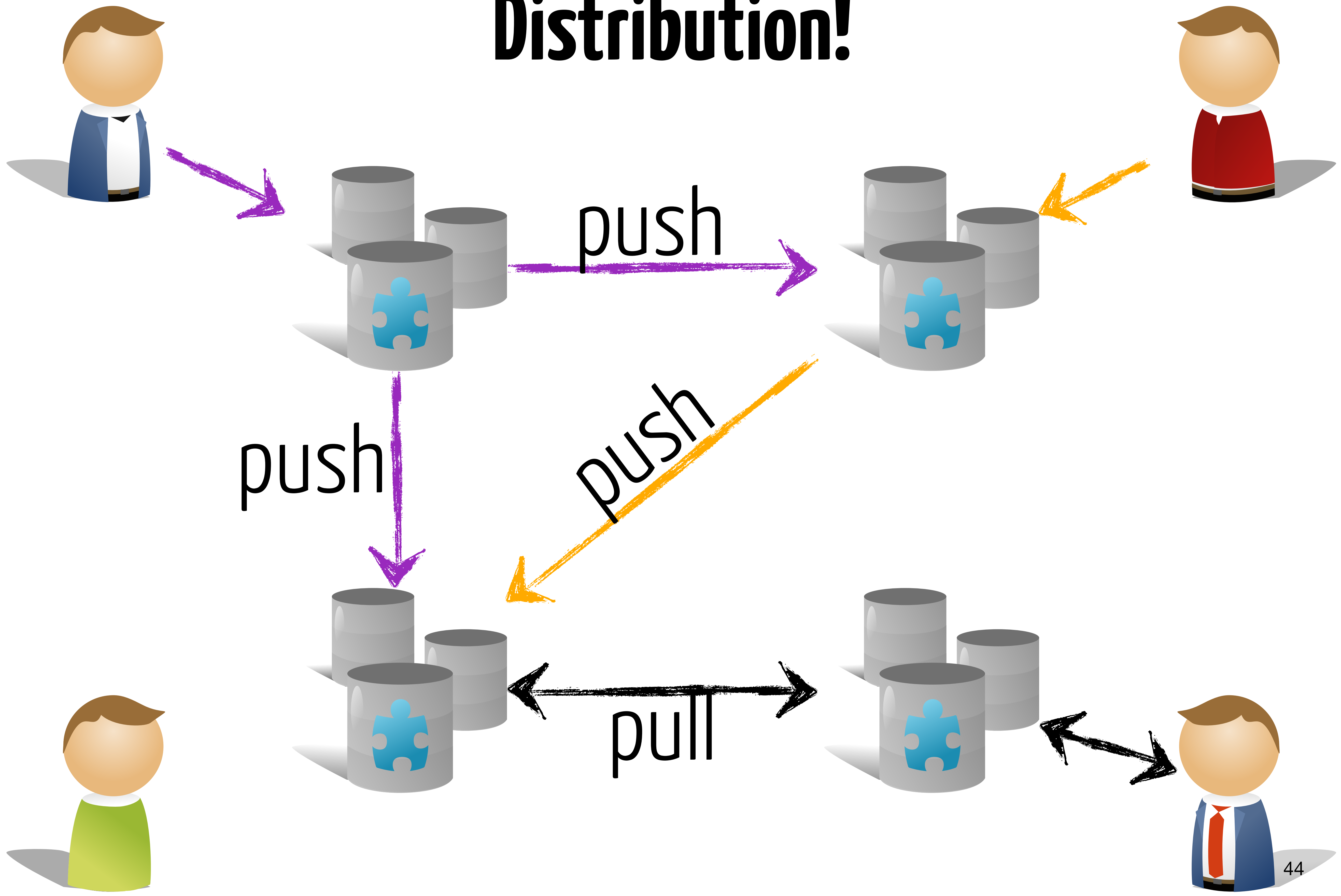
[Pro Git, #2.2]


push
add
commit
pull


commit
Centralized
Distributed
add
commit
push

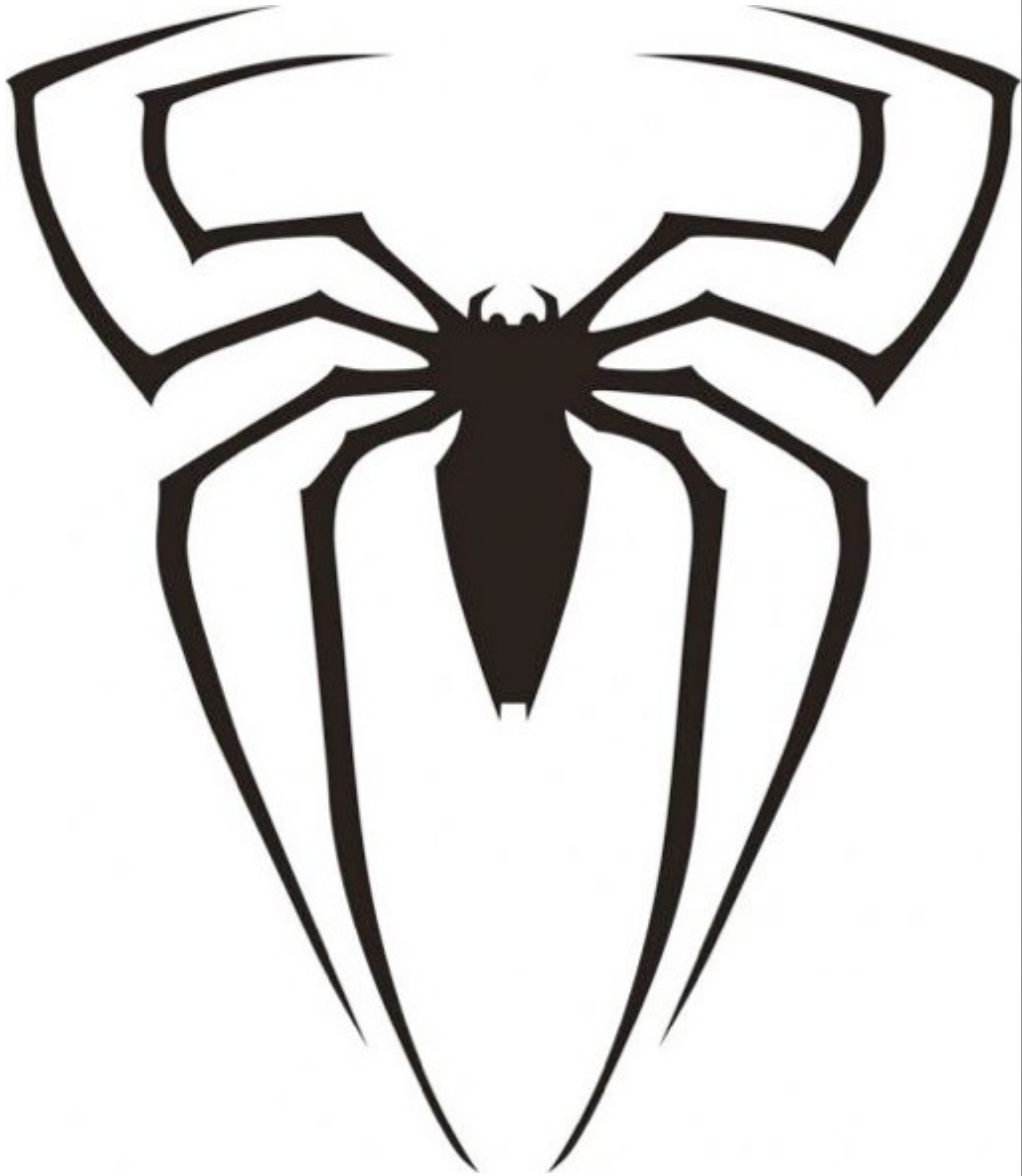
Seriously?

Distribution!



Spiderman's Theorem

«With great power comes great responsibility»



Best Practices

A commit should be a logical unit and have a descriptive message (avoid http://whatthecommit.com/)

Commit/Update frequently

Inspect your changes before committing

Don't break the build (unit tests) if not expected by the others

WE WILL BE ADOPTING THE BEST PRACTICES IN OUR INDUSTRY, JUST LIKE EVERYONE ELSE.
BEST PRACTICES
IF EVERYONE IS DOING IT, BEST PRACTICES IS THE SAME THING AS MEDIOCRE.
STOP MAKING MEDIOCRITY SOUND BAD!
SORRY.
www.dilbert.com scottadams@aol.com
4-3-08 © 2008 Scott Adams, Inc./Dist. by UFS, Inc.

DO VERSION

Source code of any sort (Java, HTML,CSS, etc.)

Images

Configuration files

Documentation (related to process and product)

Automated Tests

Files related to the project

DO NOT VERSION

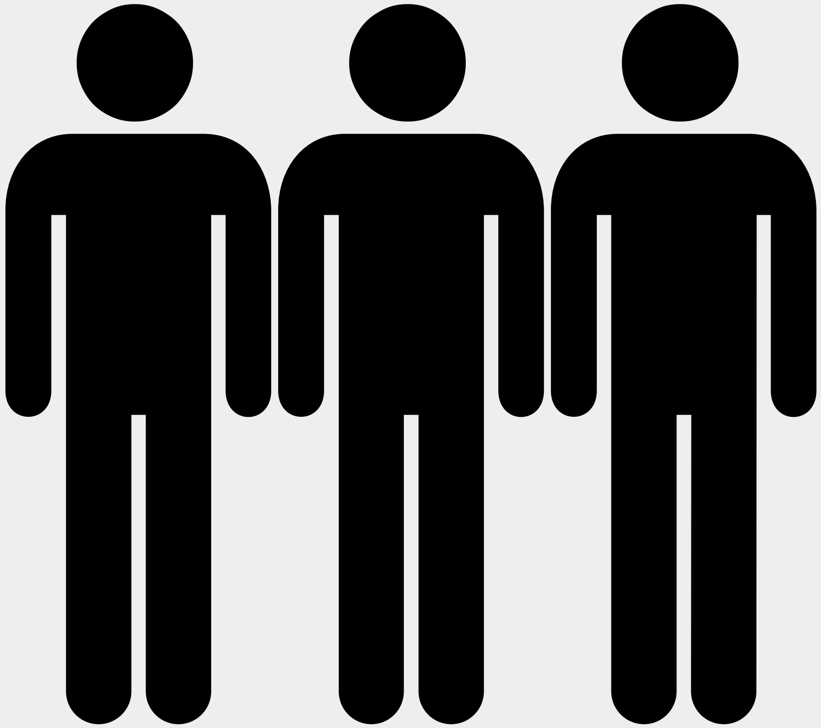
Generated Artifacts

| compiled code, documentation, etc.

Local build environment information

Secured information

Use ignore mechanism provided by VCS
For Git see: https://github.com/github/gitignore



Version control strategy for your team

?

Conclusions

Why do we version code?

To **trace** changes!

To **rollback** changes!

To **share** changes!

(among others)

Why do we **version** code?

To **trace** **changes**!

To **rollback** **changes**!

To **share** **changes**!

(among others)

Different models for code versioning

Centralized

versus

Distributed

when **N = 1**, Centralized = Distributed

He who can **do more**
can **do less**

(also works for P = NP)