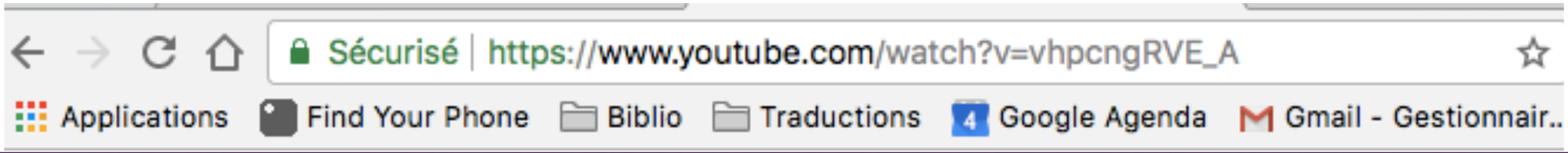


Cours à suivre (38 mn de video)

Génie Logiciel - Conduite de Projet (OverView)

26 532 vues 216 4 PARTAGER

https://www.youtube.com/watch?v=vhpcngRVE_A

Collaborative Development & Source Code Versioning

Sébastien Mosser
(modifié par M. Blay-Fornarino en intégrant des cours de M.Pallez, et M. Urli)

Avant de commencer, qui a utilisé … et pourquoi?

- ☐ Dropbox?
- ☐ GoogleDrive?
- ☐ Github?

- ☐ Jalon?
- ☐ Autre?

Et comment avez-vous géré les différentes versions de vos codes ?

Objectifs de ce cours

✓ Comprendre l'intérêt d'un gestionnaire de version.

✓ Donner une vue un peu plus concrète de la gestion de projet outillé

➡ En TD : Vous apprenez à utiliser le gestionnaire de versions connecté à un gitlab déployé au département.

➡ Ensuite vous gérez vos projets et vous vous auto-organizez. (*Les recruteurs peuvent apprécier d'avoir vos activités sous Github.)*

«Why do we version source code?»

Motivations (among others)

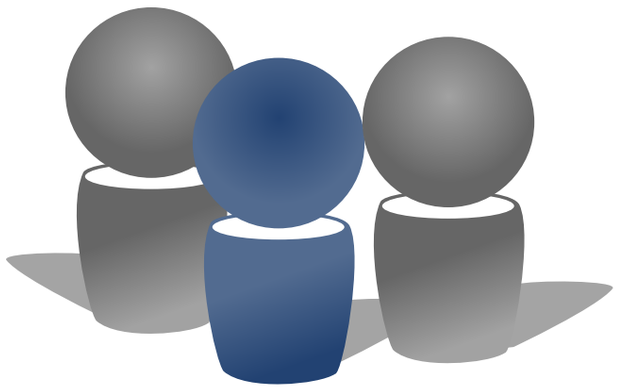
http://theplanetd.com/mongol-rally-back-in-time/

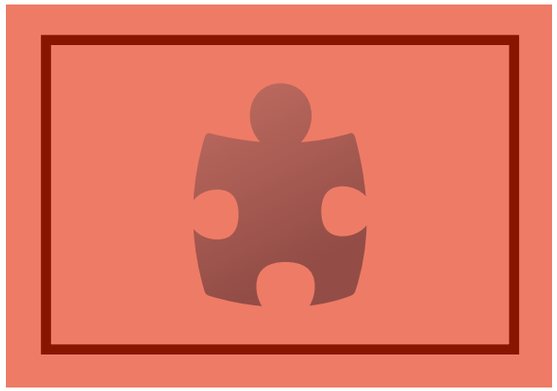
«Why do we version source code?»

To keep version history
To be able to rollback

Here is the new release!

???



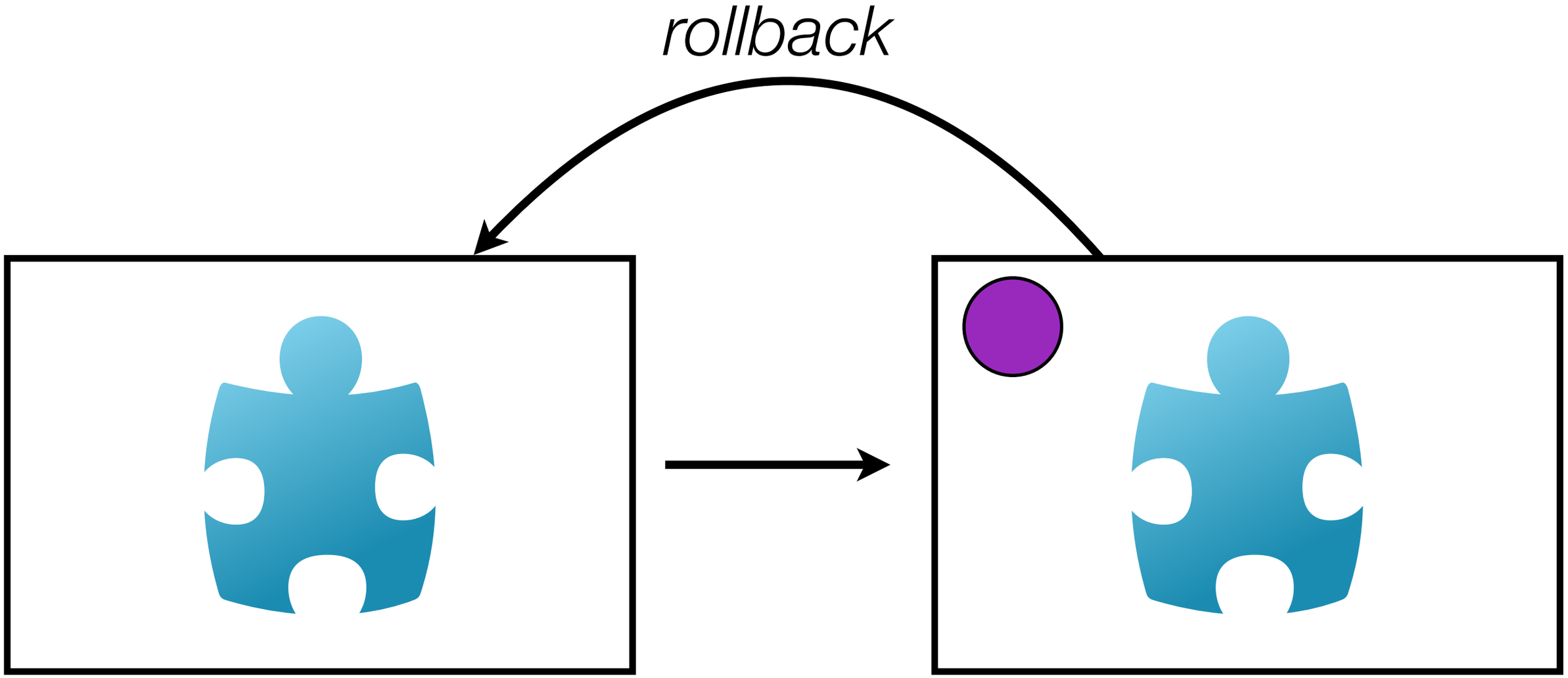


BUG!

It was working 2 days ago...

Can I see it?

To rollback changes!


rollback

Rôles d'un gestionnaire de version

Fonction *Gérer un historique* qui permet
D'enregistrer de nouvelles révisions à tout moment
De récupérer n'importe quelle révision enregistrée

http://blogs.wsj.com/photojournal/2012/03/13/photos-of-the-day-march-13/

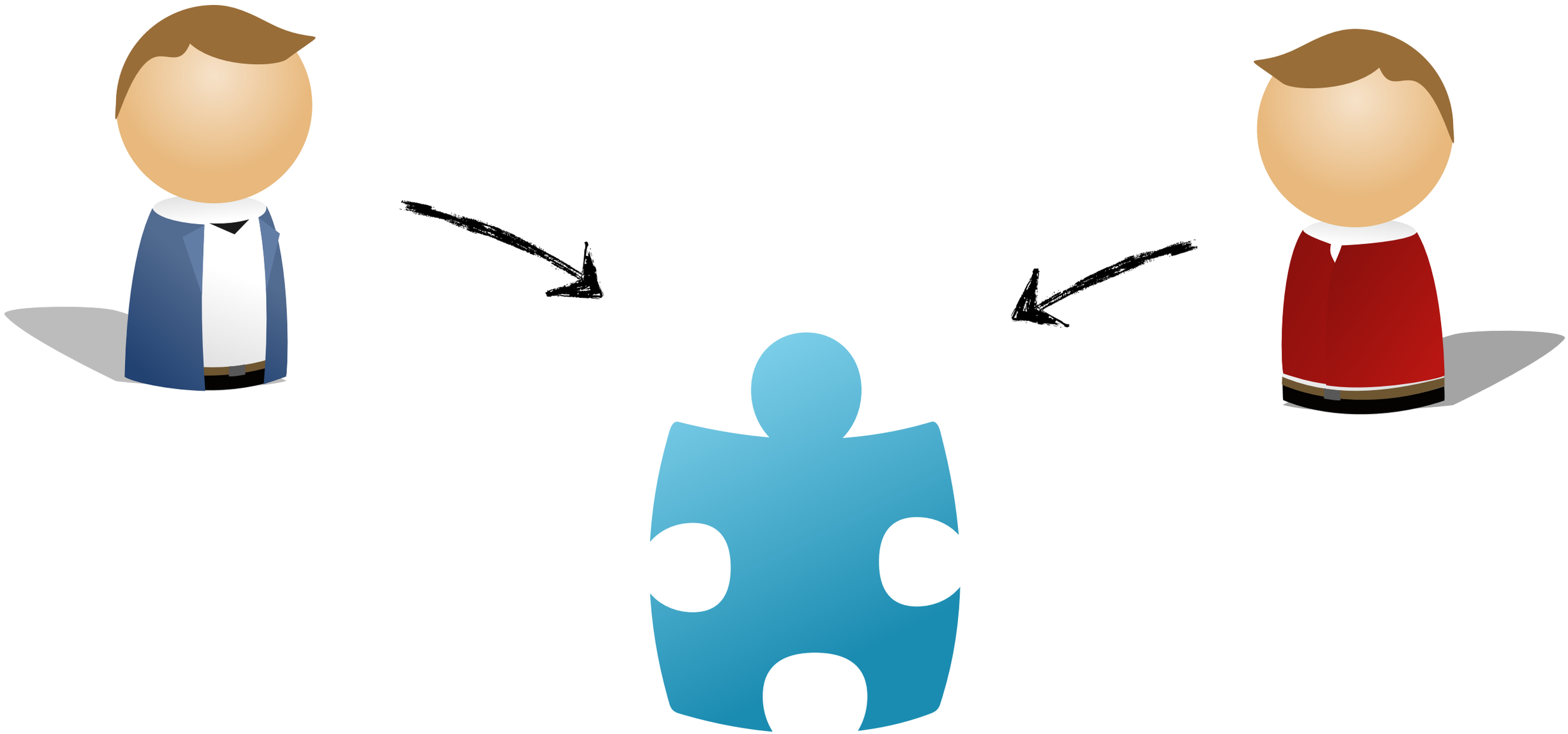
«Why do we version source code?»

To know who worked on...

developer

works on

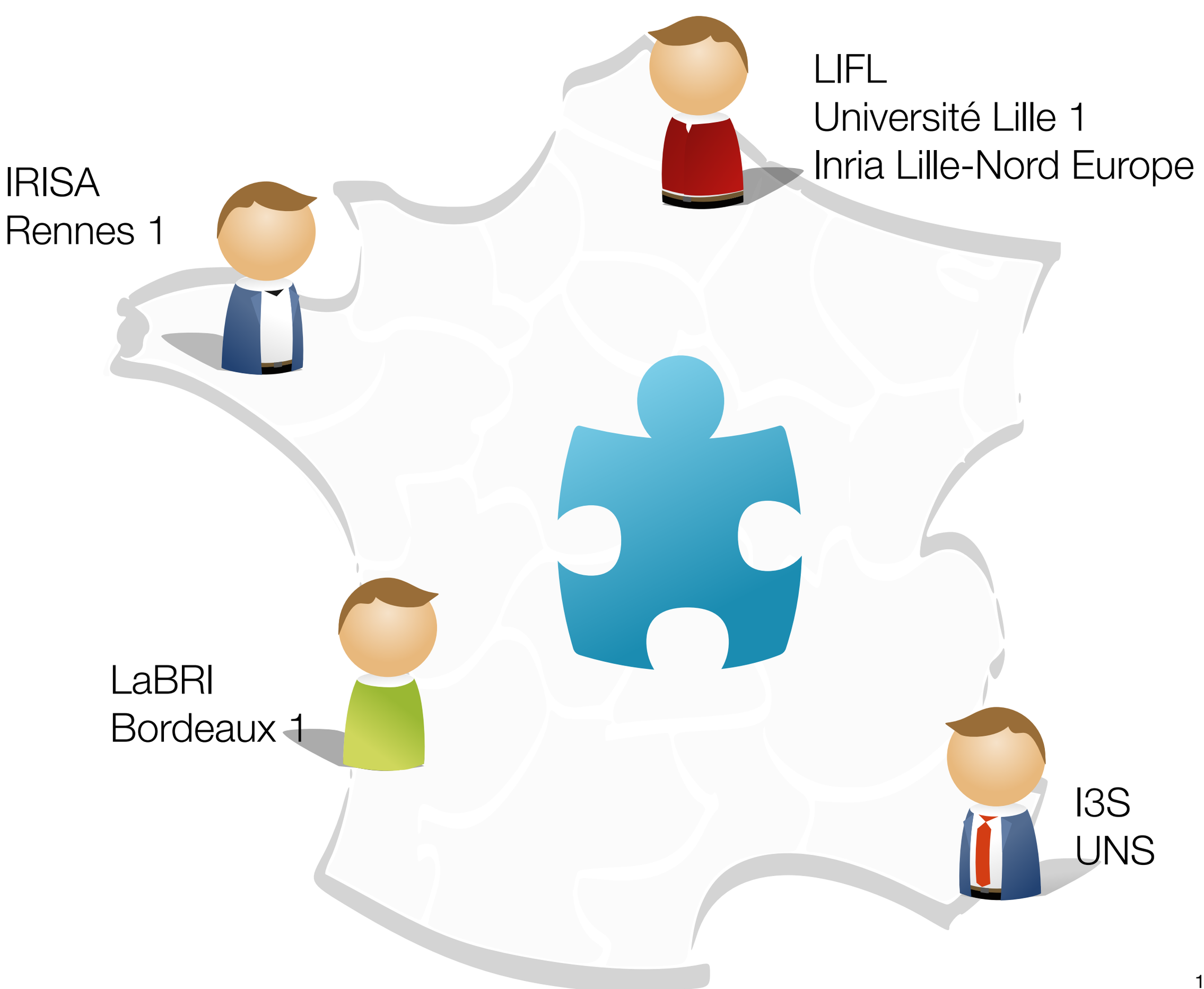
piece of software

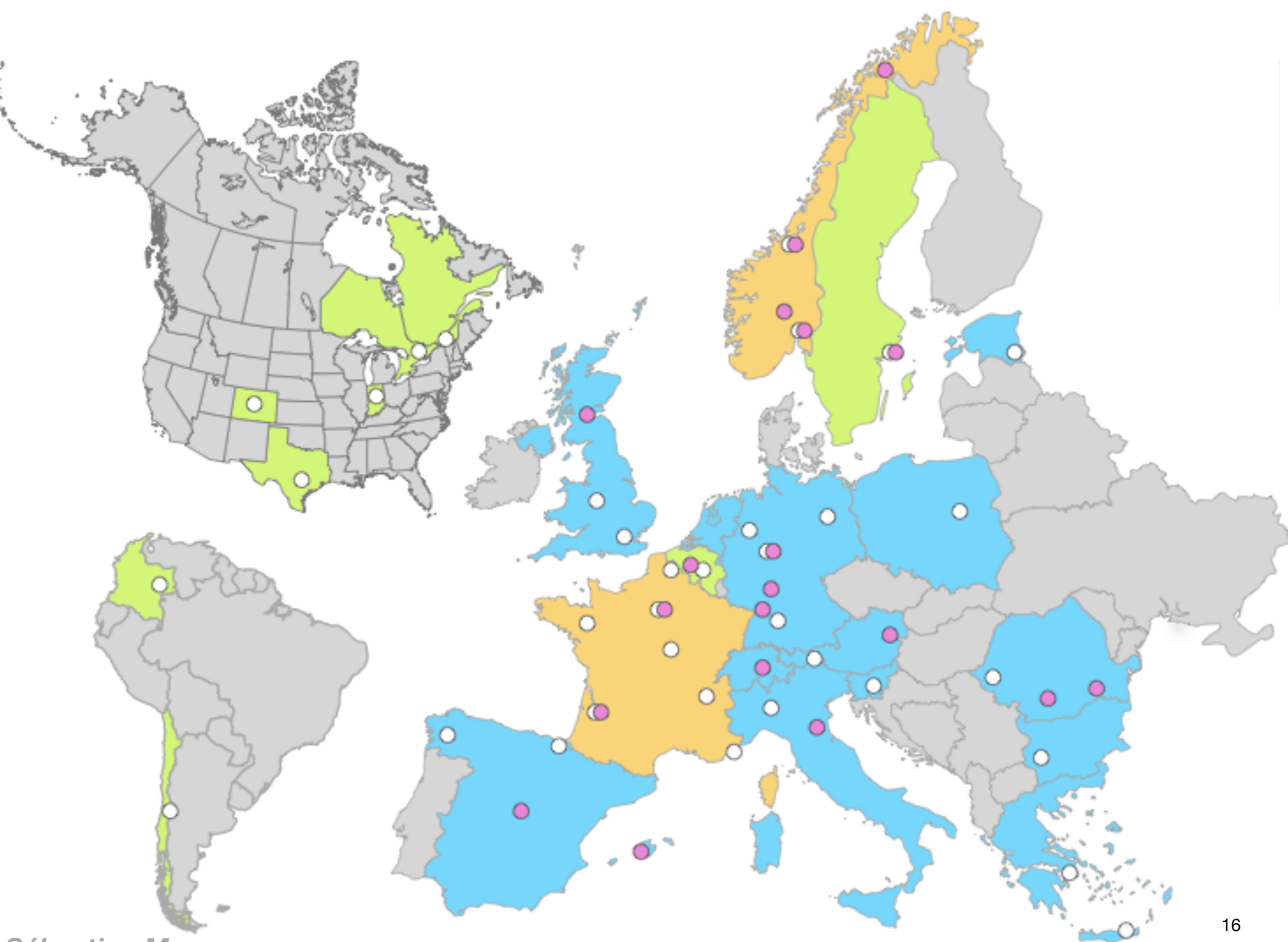


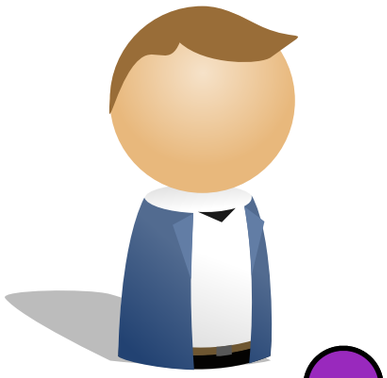
Email?

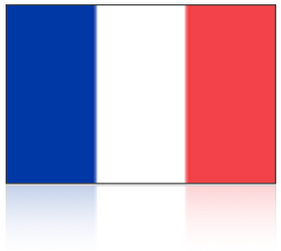
USB Key?

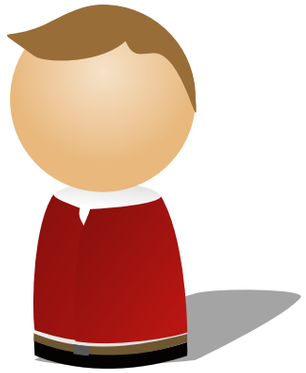
Shared directory?

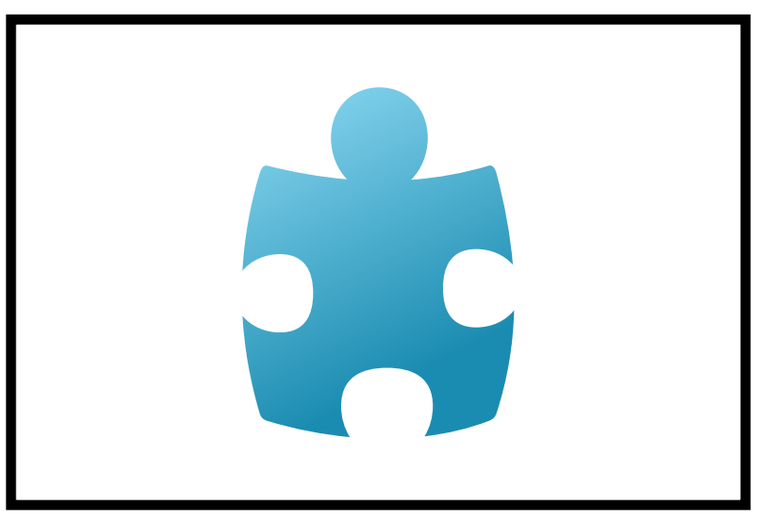

IRISA
Rennes 1
LIFL
Université Lille 1
Inria Lille-Nord Europe
LaBRI
Bordeaux 1
I3S
UNS

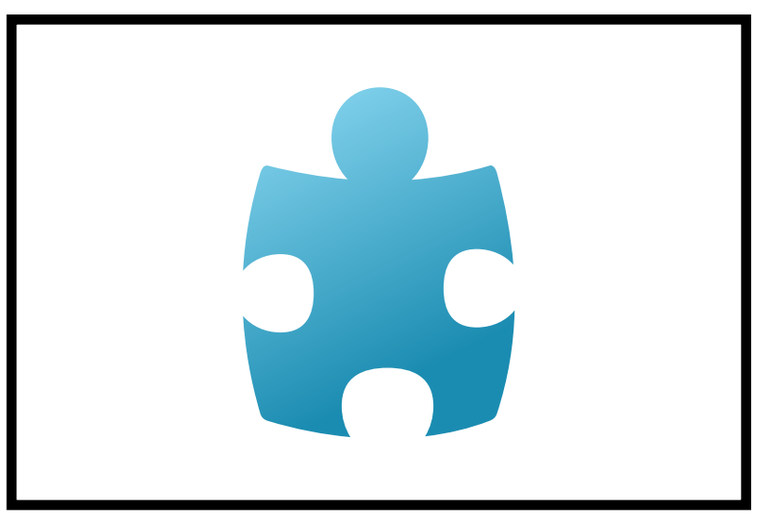




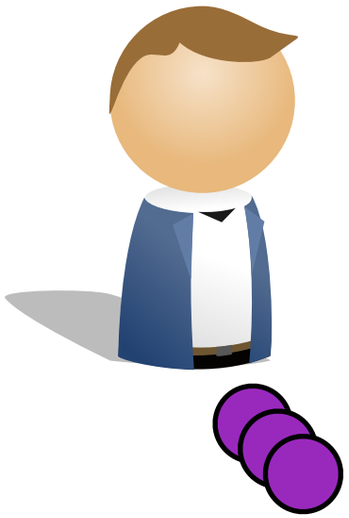




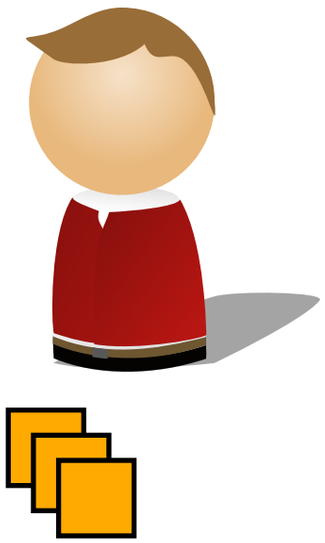


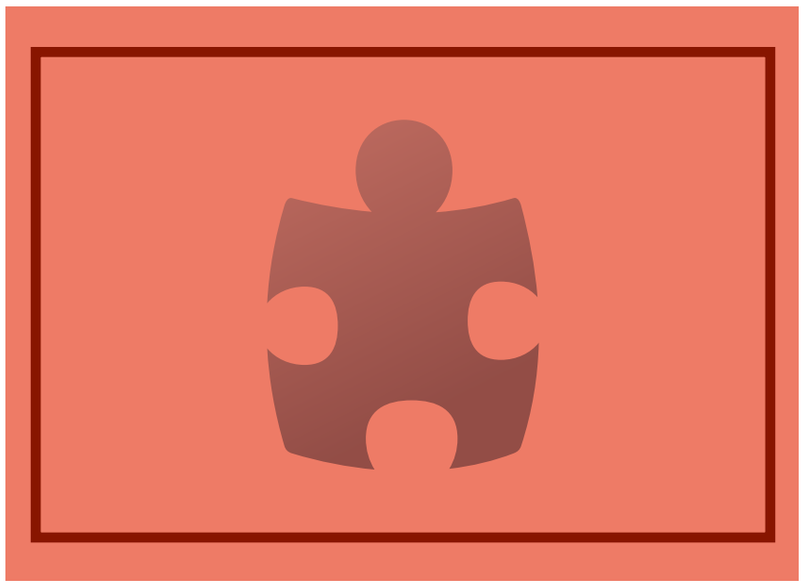


«not me!»



«not me!»

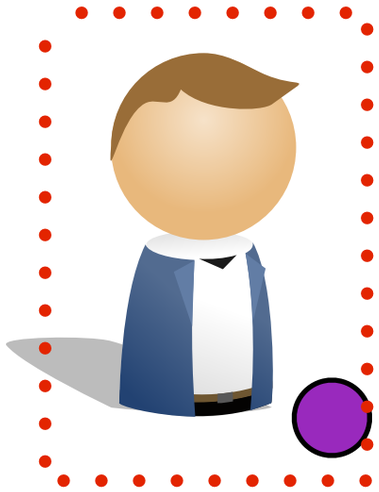


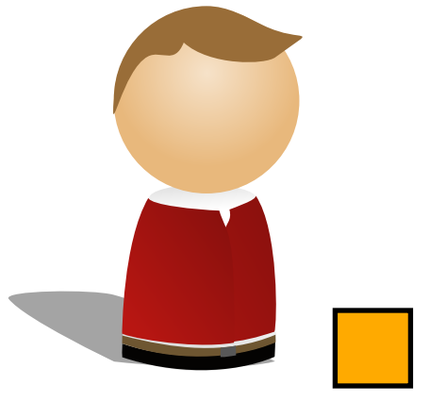


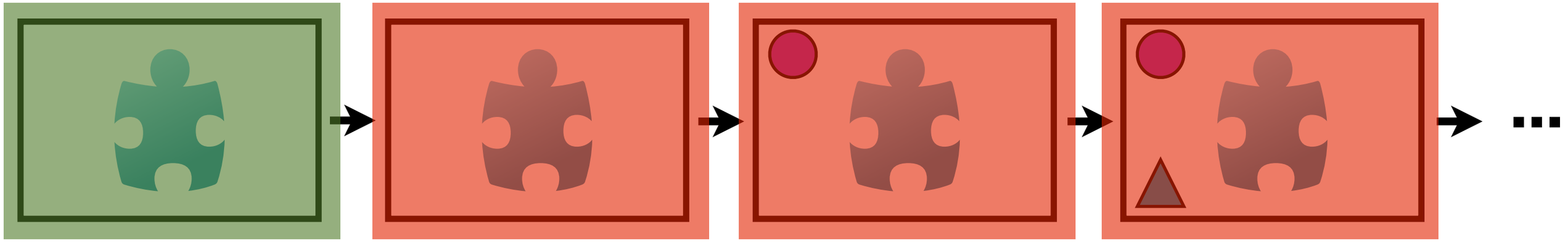
«not me!»

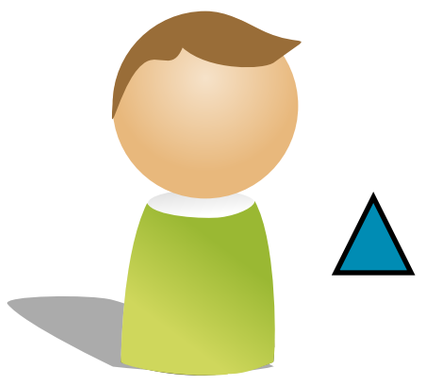
BUG!

«not me!»









BUG!

Rôles d'un gestionnaire de versions

Fonction *Documenter chaque révision en lui associant un message*

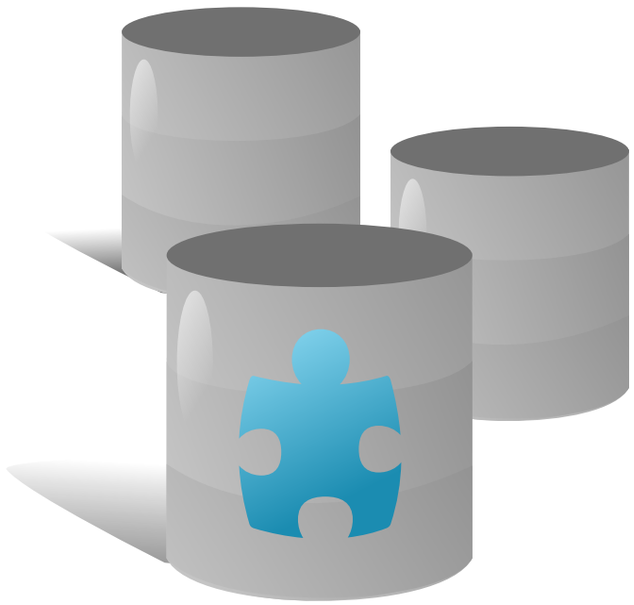
Fonction *Noter l'auteur de chaque révision*
Ce qui permet d'associer un responsable à chaque ligne de code dans chaque révision

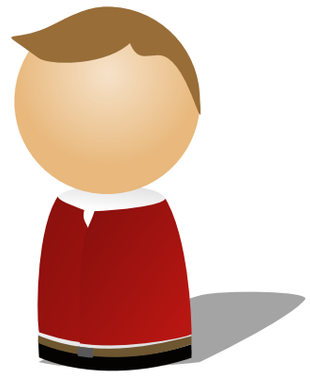
http://paysageislande.blogspot.fr/2013/06/periple-dans-le-sud-de-lislande-jour-5.html

«Why do we version source code?»

To share changes
To support merging

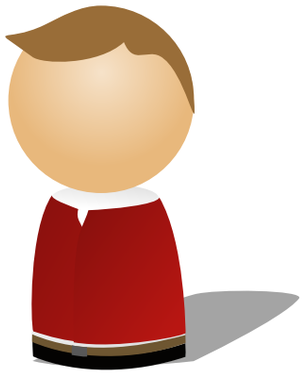
Shared Repository





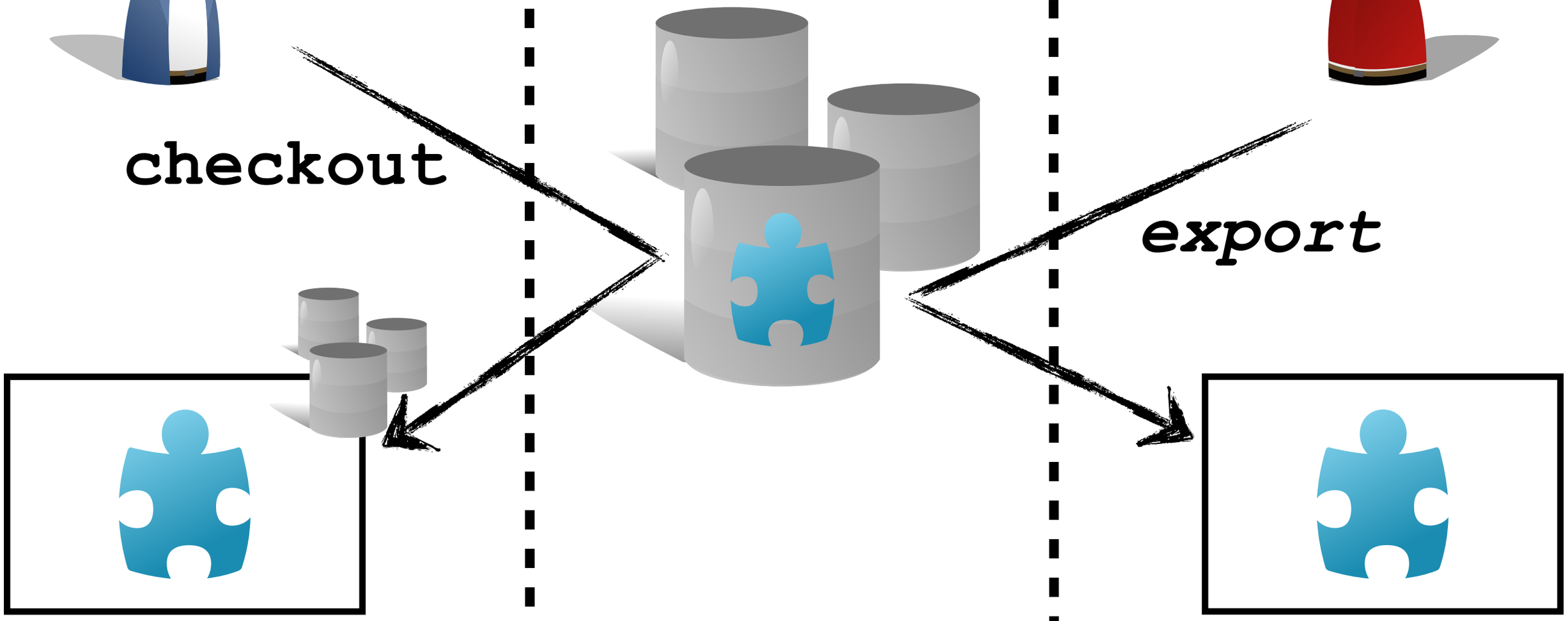
`create`

Shared
Repository



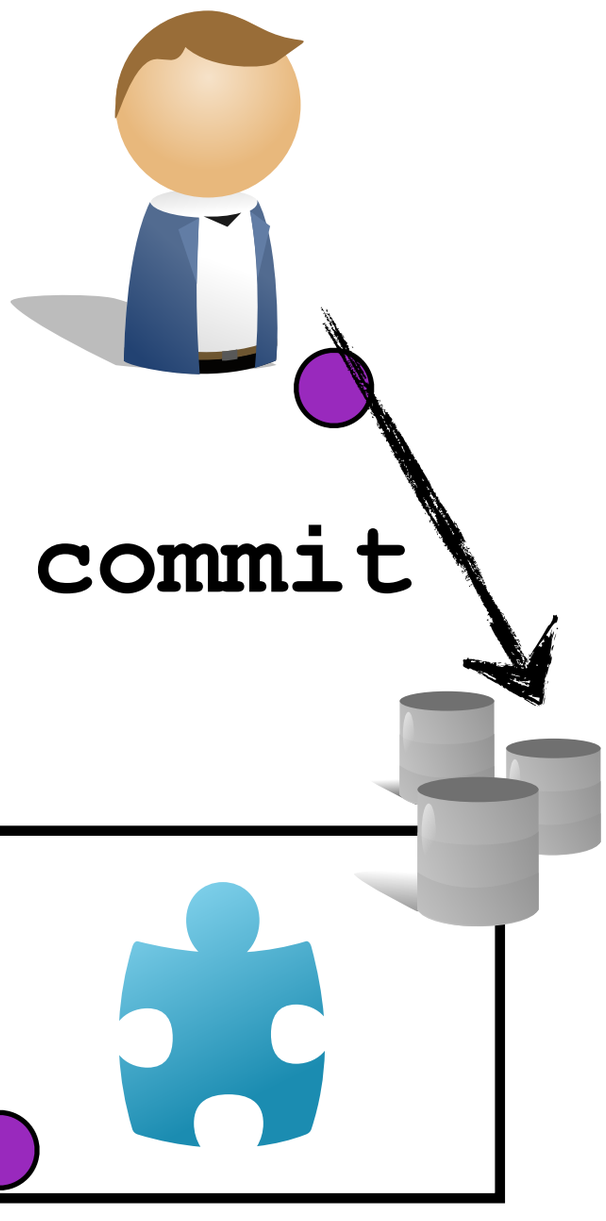
checkout

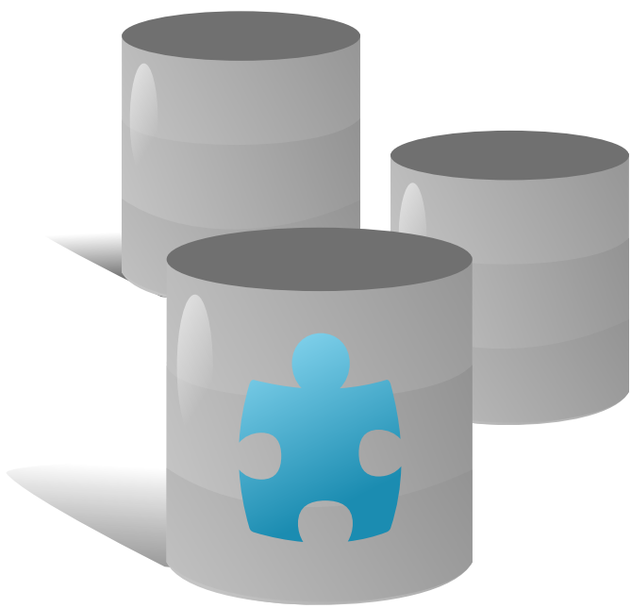
export

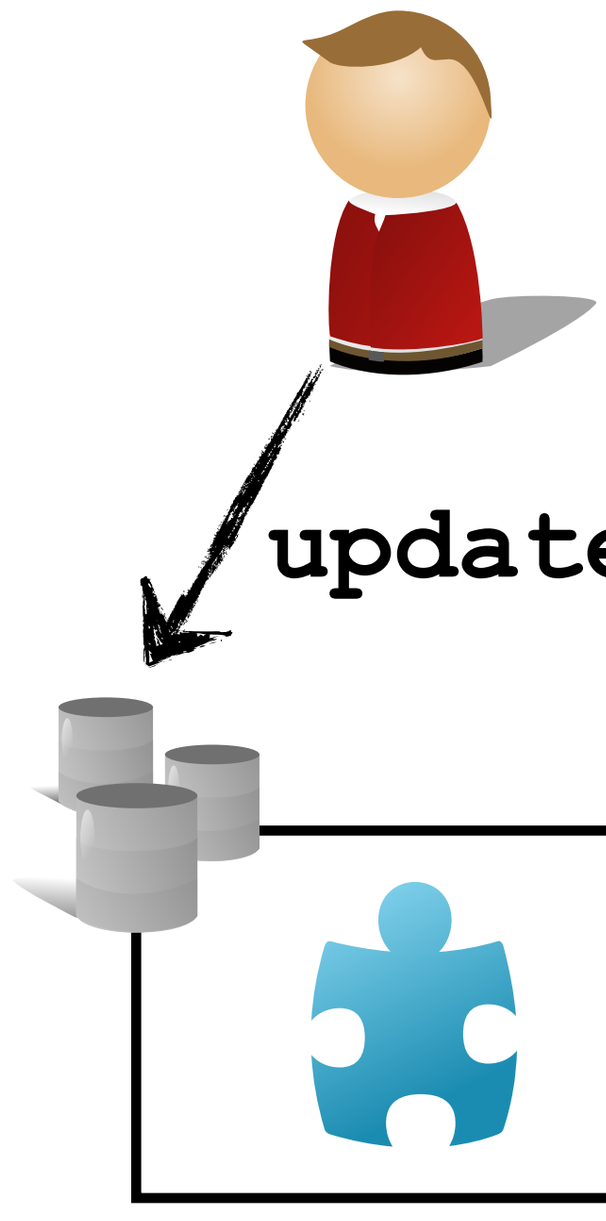


Récupérer un projet depuis le serveur (checkout)

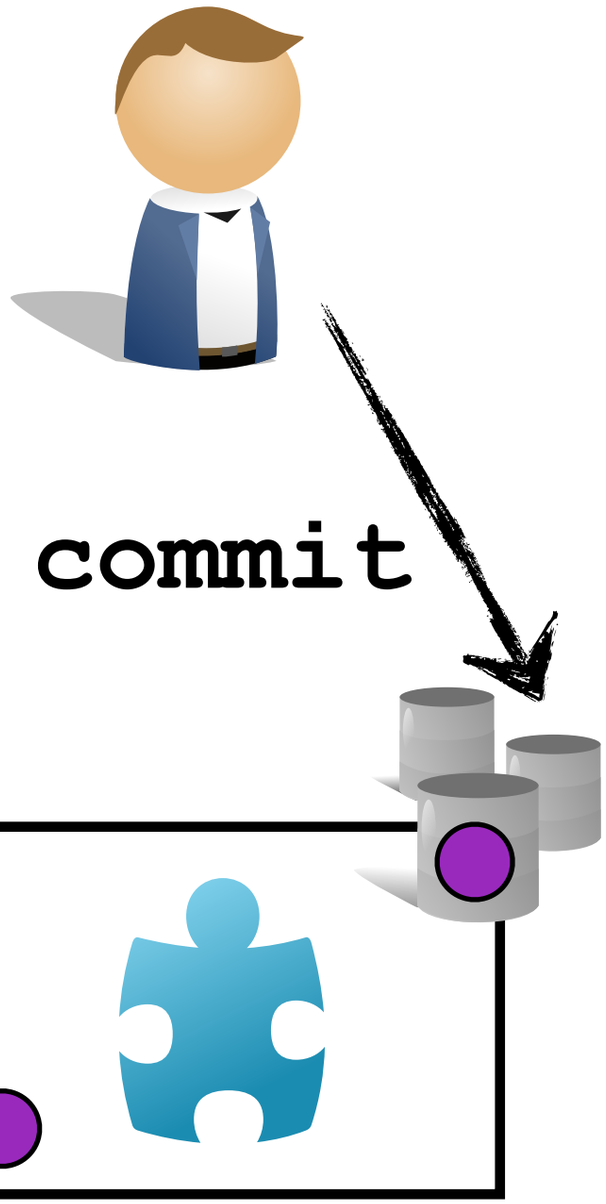
Shared Repository

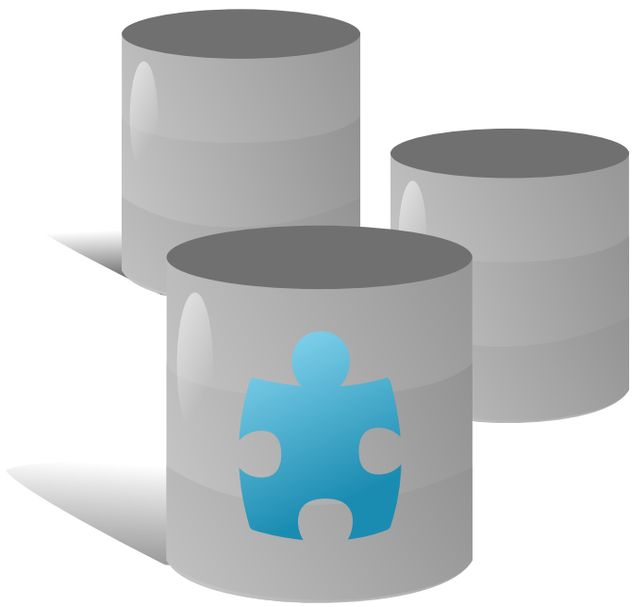


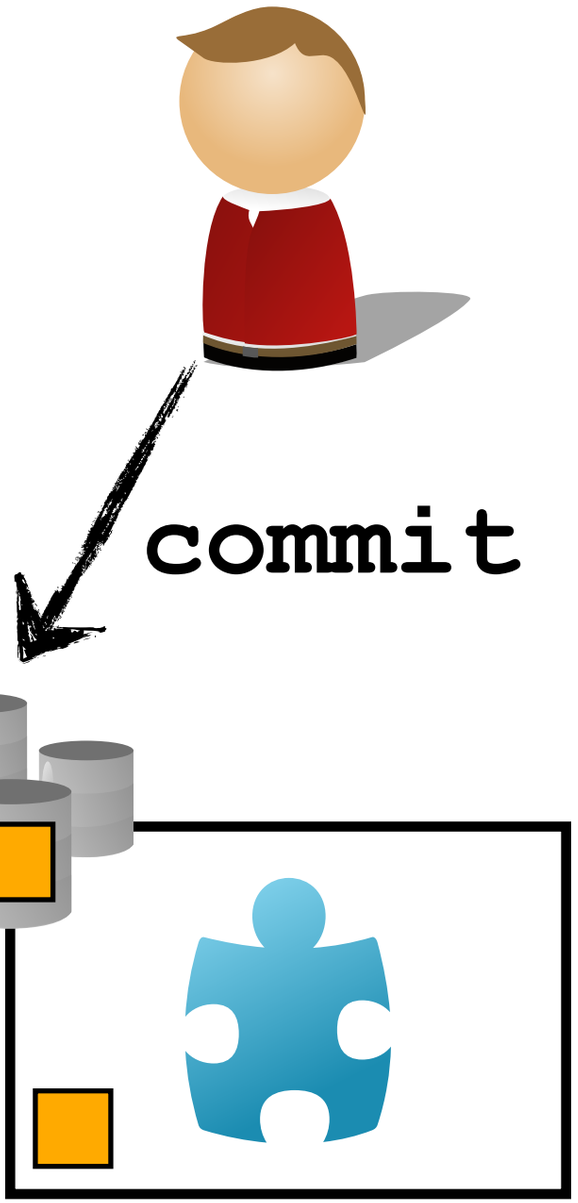




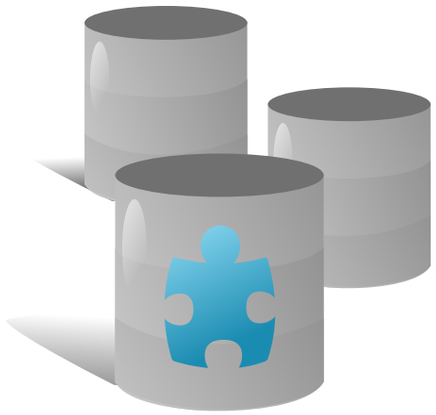
Shared Repository



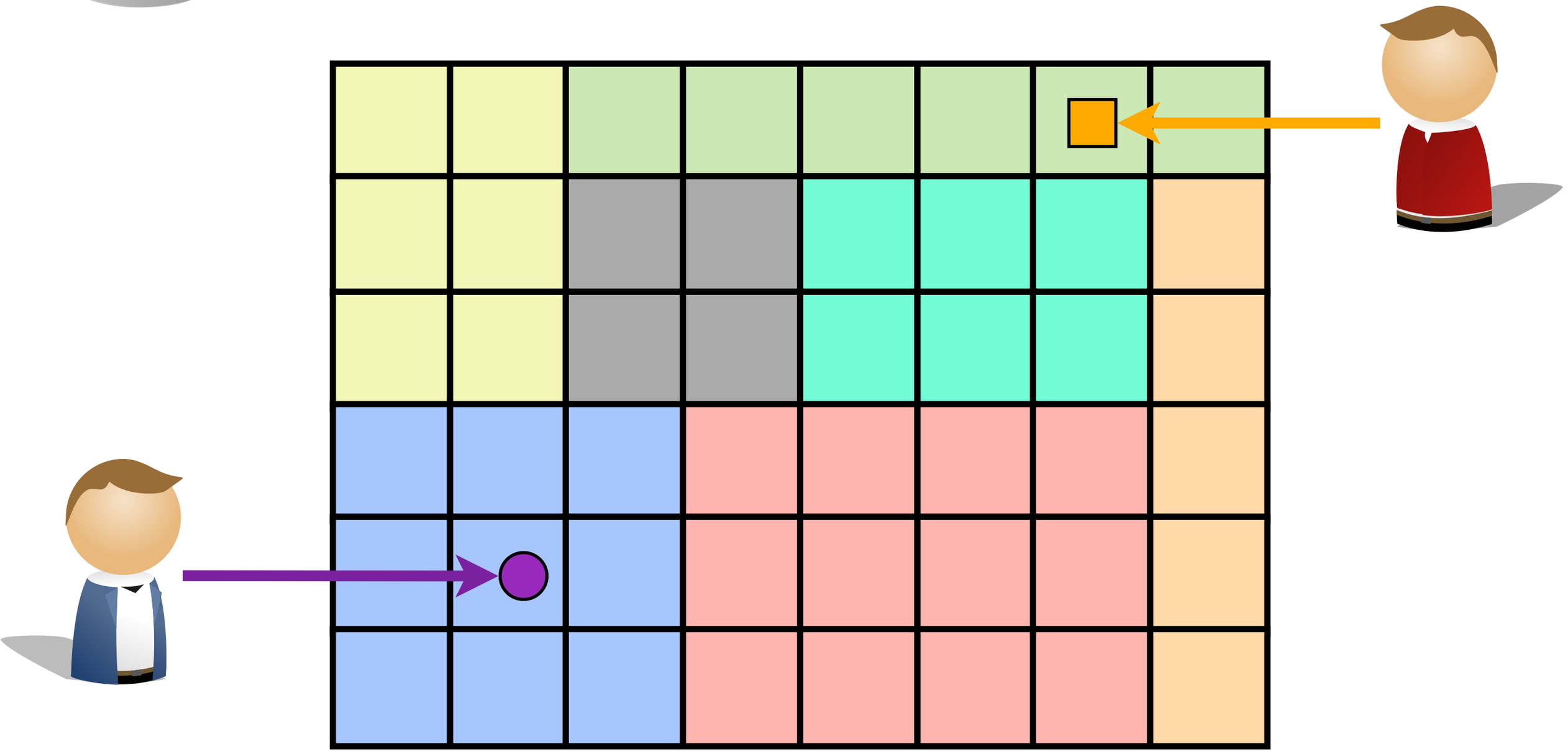




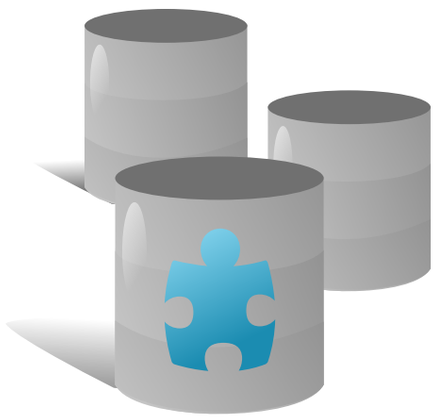
????



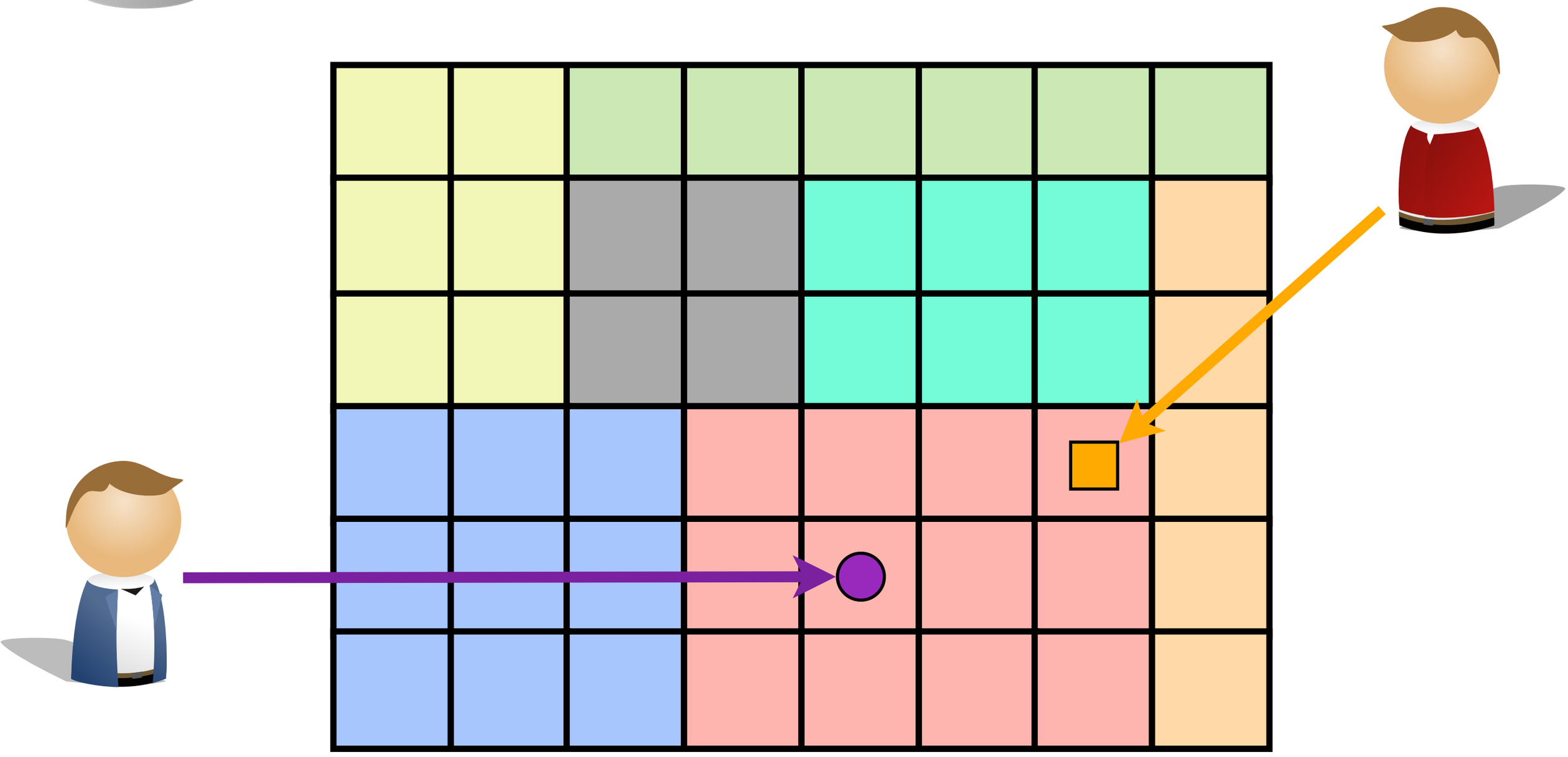
Case #1: different files



Atomic operations. No problem at all!



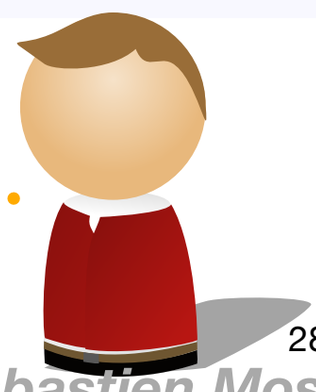
Case #2: different part of the same file

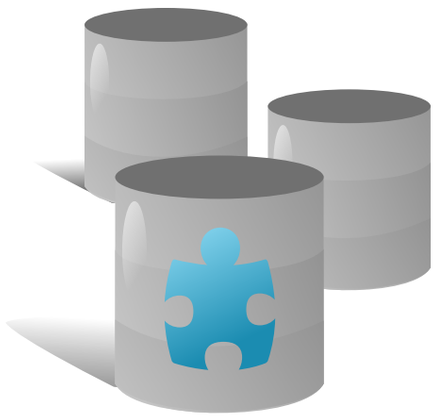


Automatic merge

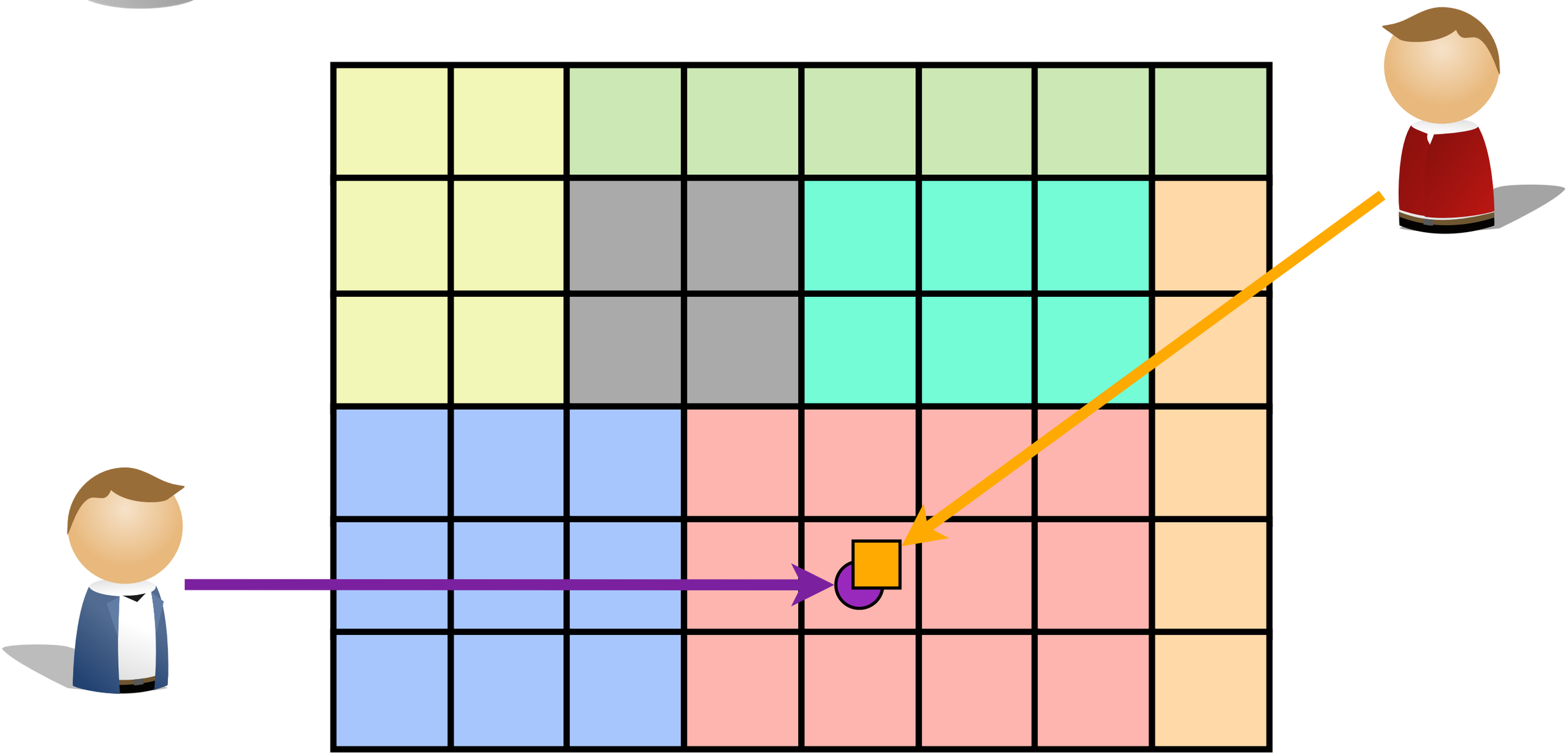
ScalaMOF

```
...  ...  @@ -45,8 +48,8 @@ class Operation  extends TypedElement with MultiplicityElement {
45   48       */
46   49      def `class`: Class = _class
47   50      def class_=(c: Class) {
48        -     require(c != null)
49        -     require(c.ownedOperations contains this)
     51   +     require(c != null, "`class` attribute cannot be null")
     52   +     require(c.ownedOperations contains this, "`class` must contain this operation")
50   53        _class = c
51   54      }
52   55      private[this] var _class: Class = _
...  ...  @@ -54,7 +57,7 @@ class Operation  extends TypedElement with MultiplicityElement {
54   57      /**
55   58       * <em>"The parameters to the operation."</em>
56   59       */
57        -   def ownedParameters: Seq[Parameter] = _ownedParameters
     60   +   def ownedParameters: Seq[Parameter] = _ownedParameters.reverse
58   61      private[this] var _ownedParameters = List[Parameter]()
```





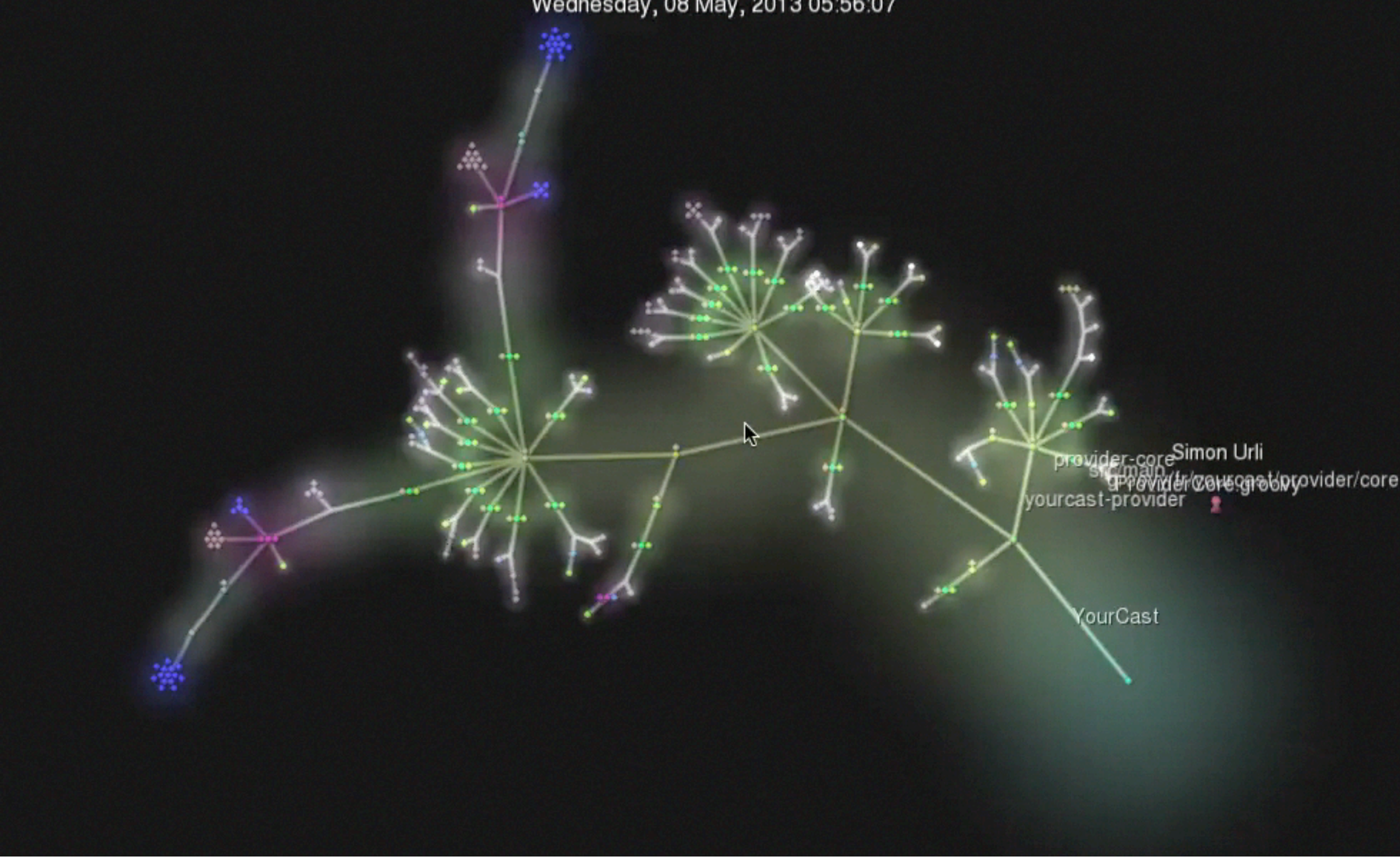
Case #3: **same** part of the **same** file



Conflict!

Rôles d'un gestionnaire de version

Fonction *Faciliter la fusion des modifications en gérant les conflits*



Visualisation des changements

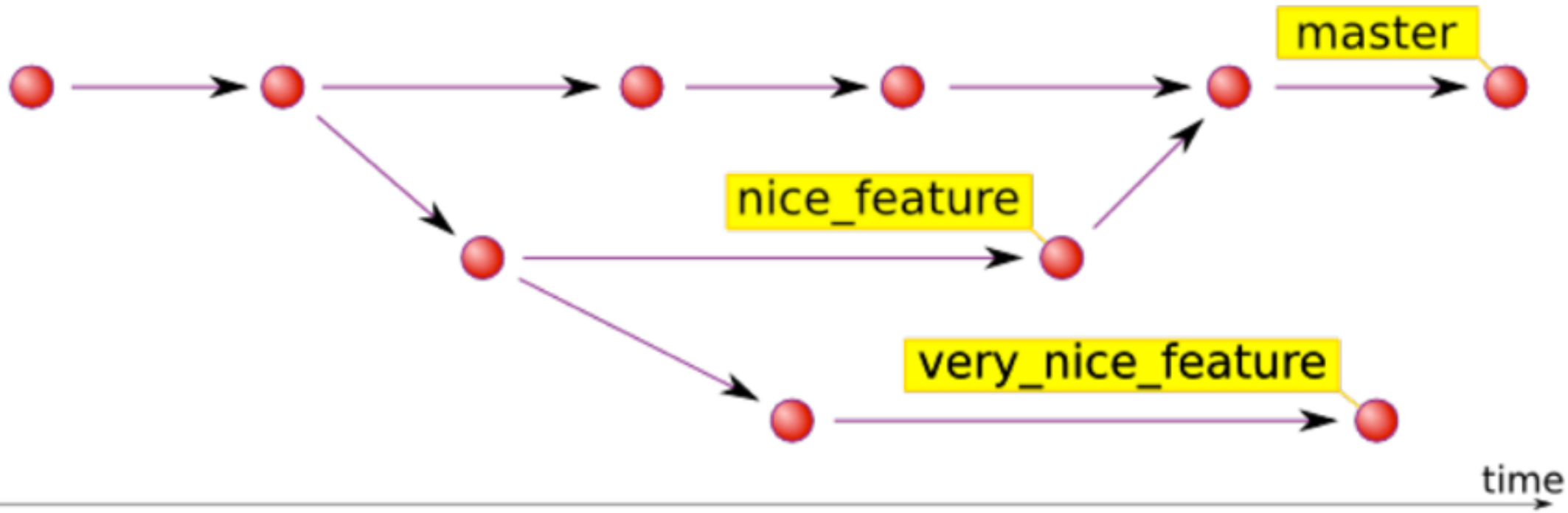
«Why do we version source code?»

To support multiple versions (Branches)

Problématique

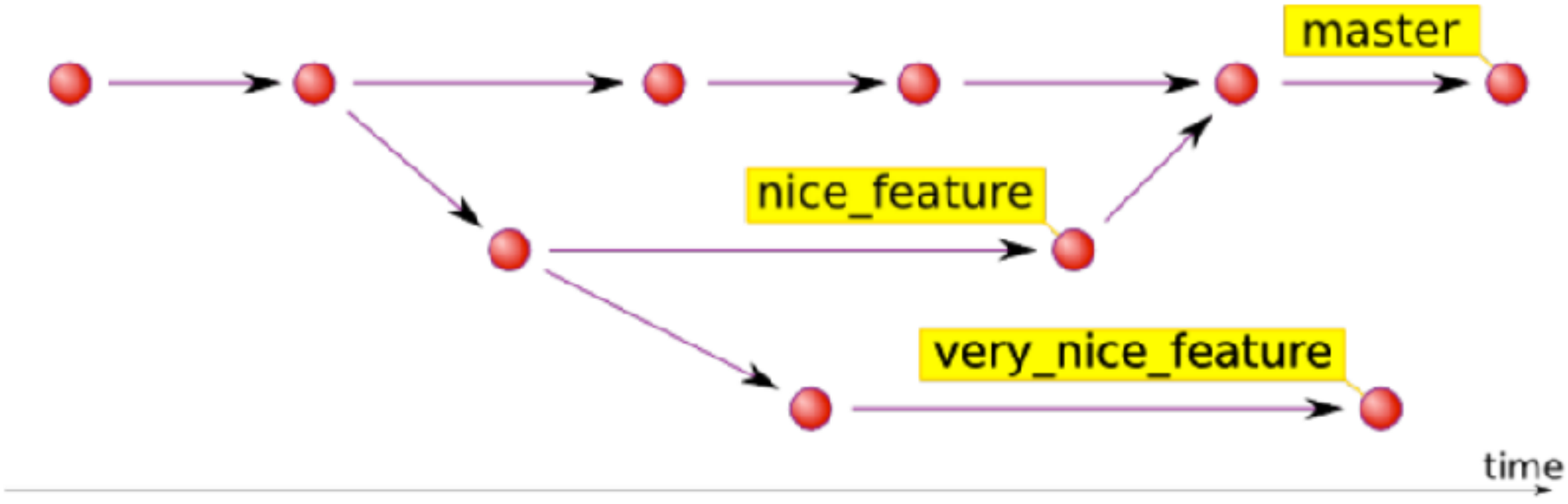
- Cas : même logiciel / différentes tâches
  - Plusieurs systèmes d'exploitation
    - Linux, Mac, Windows
  - Plusieurs lignes de logiciels
    - Produit commercialisé (stable)
    - Nouvelle idée (instable, en cours de développement)
  - Plusieurs cibles
    - Logiciel bridé (version de démo)
    - Logiciel « pour particulier »
    - Logiciel « pour entreprise »
  - Plusieurs bogues
    - Logiciel commercialisé
    - Repérage du bogue 277
    - Repérage du bogue 389

Les branches !



Chaque rond est un commit

Les branches !



- Pouvoir travailler en parallèle sur plusieurs features en même temps
- Pouvoir switcher entre les features, les versions etc
- Fusionner les modifications sur une même branche à la fin

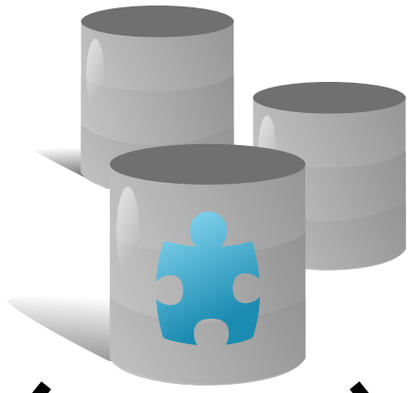
Rôles d'un gestionnaire de versions

Fonction *Faciliter le développement parallèle de multiples branches et le transfert de modifications entre branches*

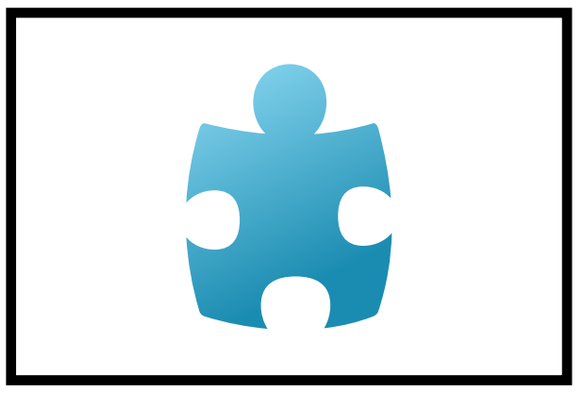
Distributed Model

(e.g., Bazaar, Git)

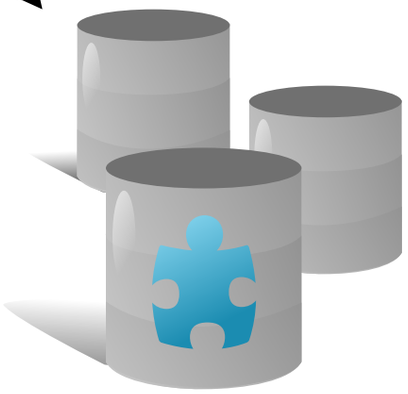
repository

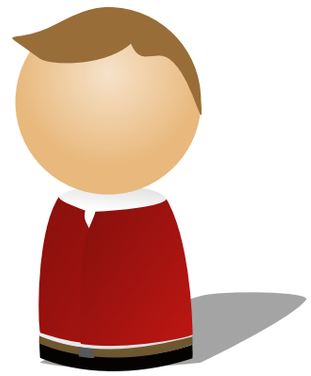


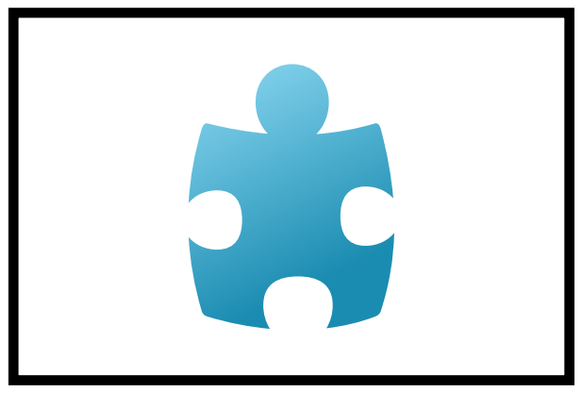
clone



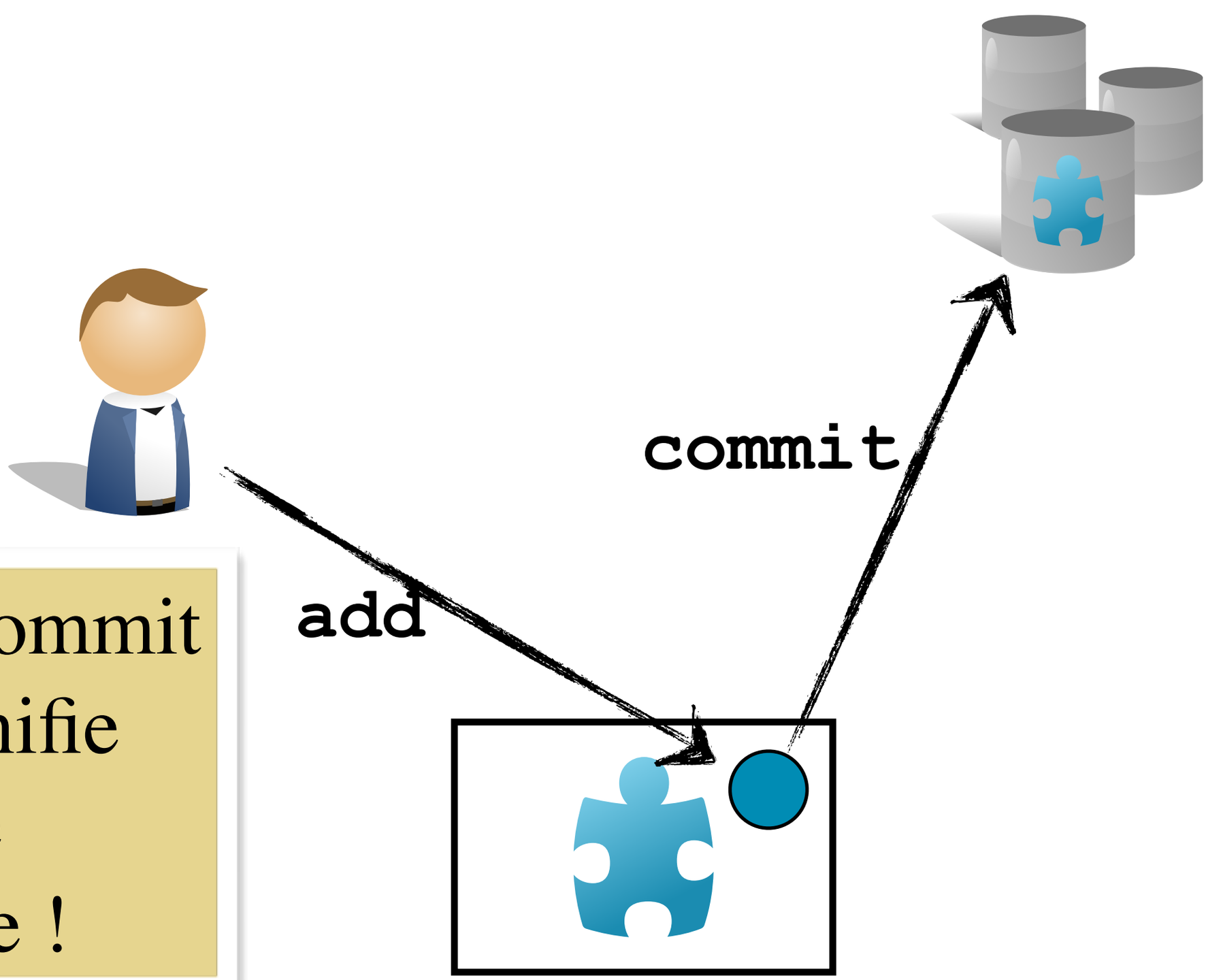
clone



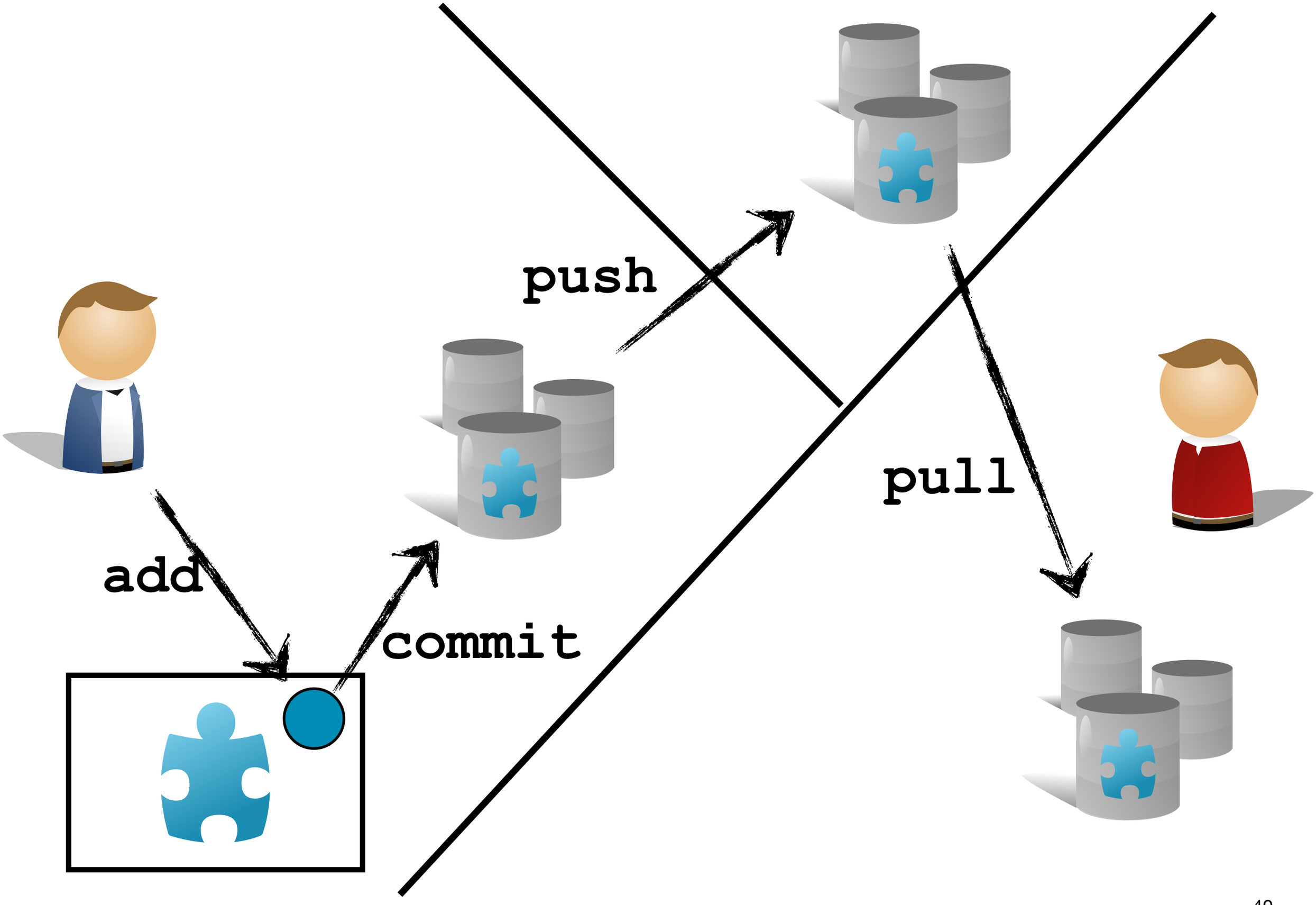


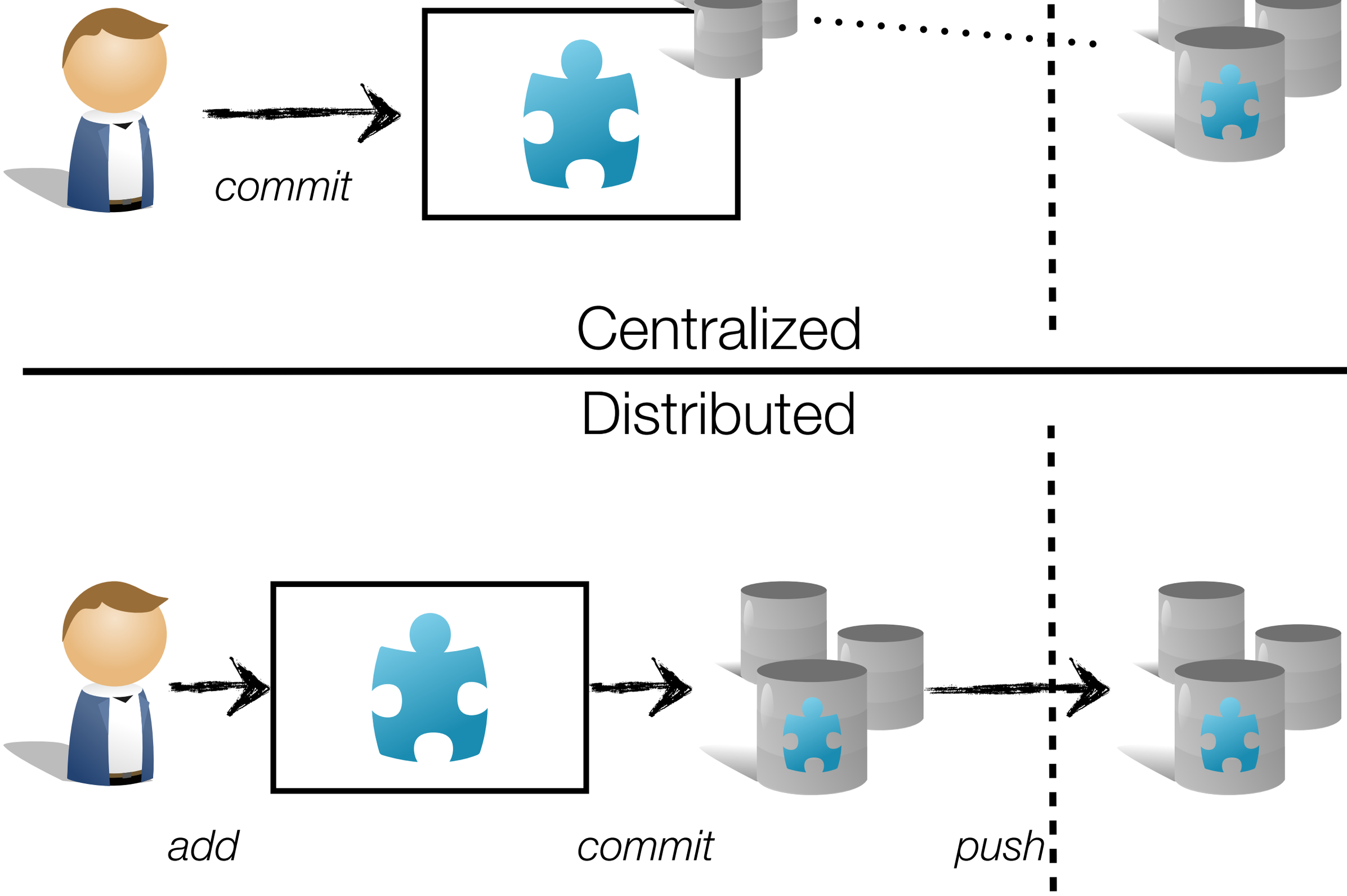


completely offline!



- Un commit ne signifie PAS le partage !


push
pull
add
commit


commit
Centralized
Distributed
add
commit
push

Systèmes de version décentralisés

- Autant de dépôts que d'utilisateurs
- Mais des dépôts hébergés sur des serveurs (ex: GitHub, BitBucket, ...)
- Un commit ne signifie PAS le partage !
- Obligation de faire commit ET push !
- Git, Mercurial, ...

Avantages du décentralisé

- Mode «déconnecté» : possibilité de travail en local
- Prise en charge des branches beaucoup plus évoluée
- Plus fiable car serveur central est seulement une AUTRE copie des versions
- Beaucoup de plateformes le supporte

ATTENTION

- GitHub n’est PAS Git !
- GitHub est une **plateforme d’hébergement** de dépôts Git offrant en plus la possibilité d’annoter le code etc.

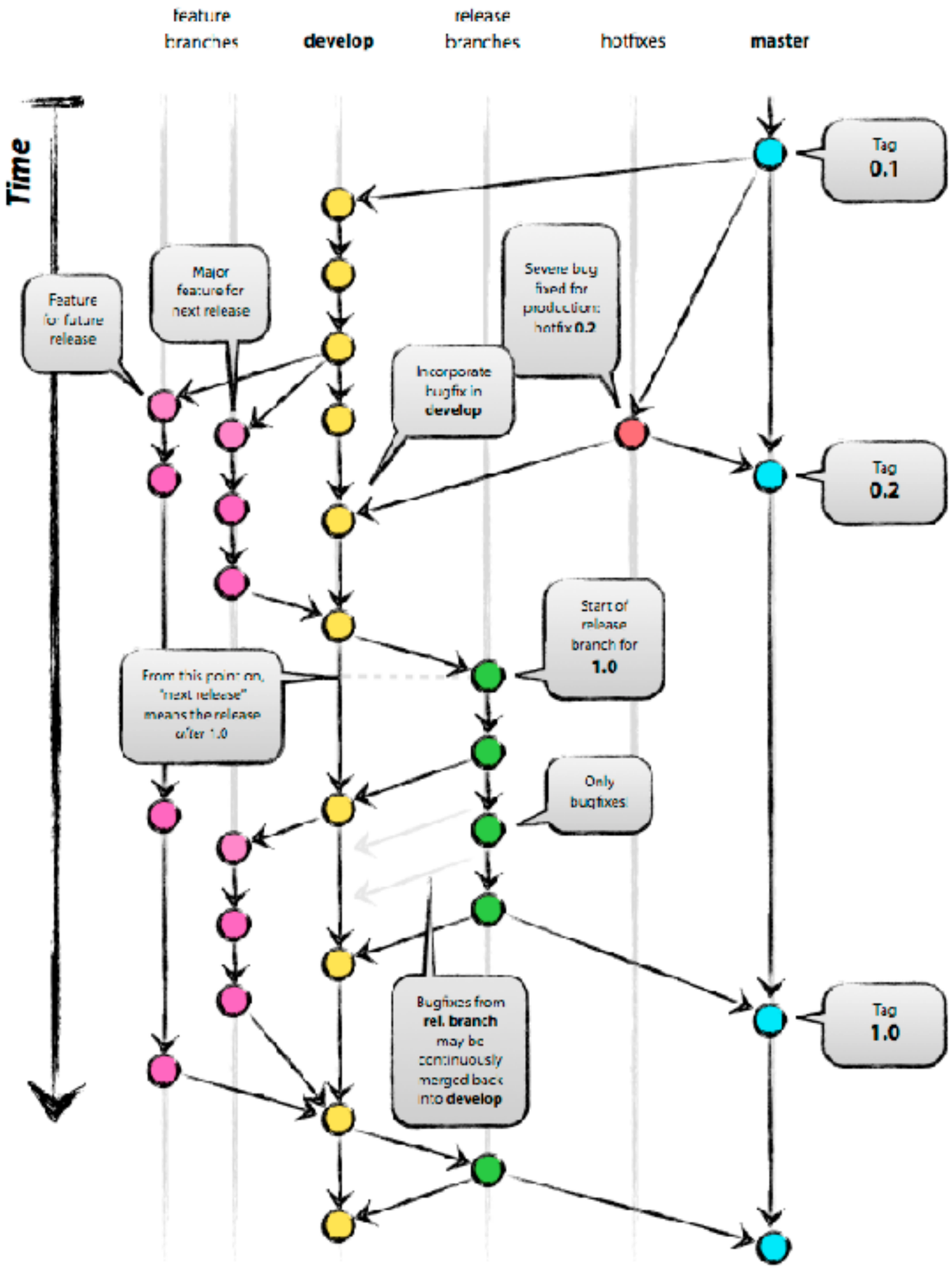
Quelques commandes de Git

- **init** : initialisation d'un dépôt vide
- **clone** : récupération d'une copie d'un dépôt
- **add** : ajout d'un fichier nouveau ou modifié pour le commit
- **commit** : enregistrement des modifications sur le dépôt
- **push** : envoie les modifications sur un serveur
- **pull** : récupère les modifications d'un serveur
- **status** : voir l'état du repository
- **branch** : gérer les branches
- **checkout** : switcher sur une autre version / branche
- **log** : afficher les infos des précédents commits
- ... : https://git-scm.com/

Quelle gestion des branches ?

Git Flow : un processus d'utilisation de Git

- Idée de Git Flow : **exploiter les branches au maximum mais pas n'importe comment !**
- Principe :
  - Une **branche « master »** de releases
    - récupérée par défaut du répertoire distant
      → est une branche utilisée comme pointeur sur une version de release
  - Une branche « **develop** » où tout se passe
  - Des branches « **features** » par fonctionnalité (à partir de **develop**)
  - Des branches pour les patchs…
  - Des branches « **release** »: sortie d'une version stable "officielle" → est créée à partir de la branche "develop"


feature branches
develop
release branches
hotfixes
master
Time
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.1
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

Un outil pour « Git Flow »

... qui s'appelle «Git Flow»

- Un ensemble de commandes pour créer automatiquement les branches en fonction de ce qu'on fait
- Et fermer/fusionner les branches !

➡Sucre syntaxique au dessus de git !

Git Flow : quelques liens

- **https://jeffkreeftmeijer.com/git-flow/**
- http://nvie.com/posts/a-successful-git-branching-model/
- https://github.com/nvie/gitflow

Attention, ...

Anti-informative Commit Messages

master

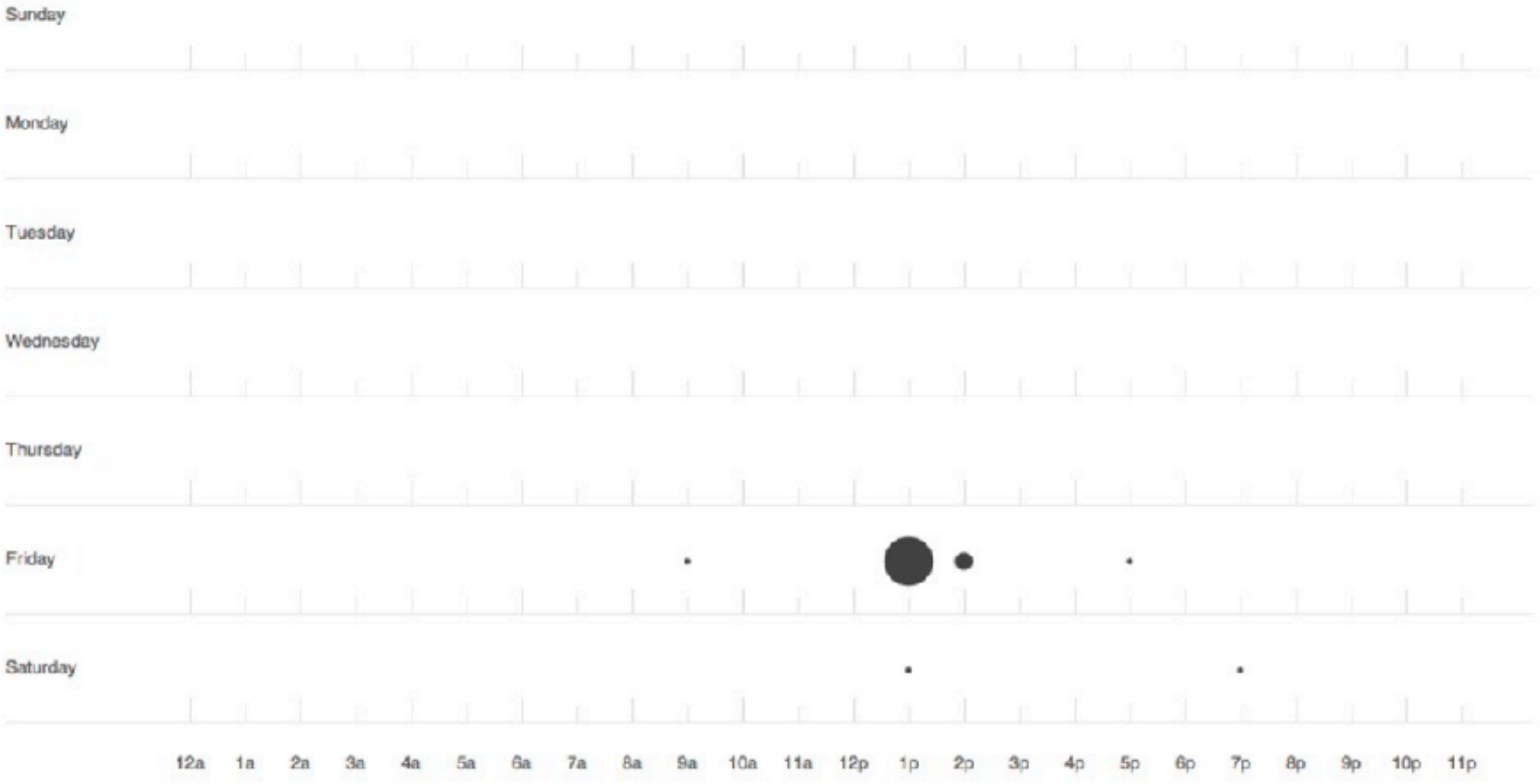
Author	Commit	Message
[redacted]	2285889	Version 22/02/2013
[redacted]	4e06542	Version 21/02/2013
[redacted]	5798097	Version 2 16/02/2013
[redacted]	775f97a	Version 16/02/2013

Commit message = Intention of the version

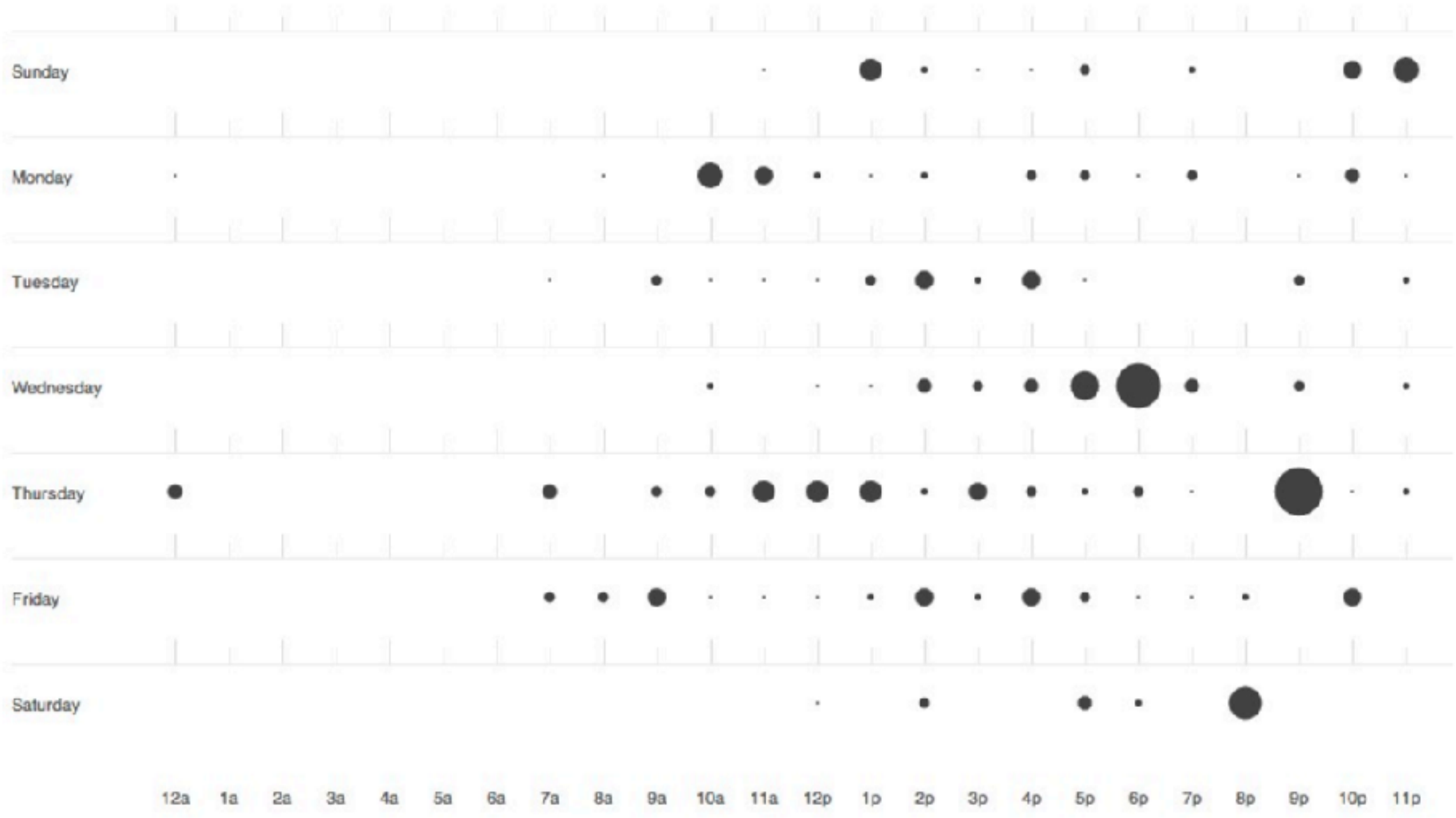
master

Author	Commit	Message
[redacted]	6ad5d7a	Correction des warnings
[redacted]	a2bd26c	Correction des erreures de nom de methode
[redacted]	7a3e625	Suppression de fourmiliere.java dans le pack

One-shot activity



Expected: Continuous Development



The Archive Commit

master ▾ | public /

Myrmes[redacted].zip

test.txt

Versioned Binary Files

- src
  - fourmiliere
    - Evenement.java
    - Fourmiliere.java
    - Salle.java
    - SalleLarve.java
    - SalleOuvriere.java
    - SalleSoldat.java
    - SalleStock.java
- bin
  - fourmiliere
    - Fourmiliere.class
    - Salle.class
    - SalleLarve.class
    - SalleOuvriere.class
    - SalleSoldat.class
    - SalleStock.class

.gitignore

Commits must be related to tickets!

6f09558	TWTWGMM-38 Modification de la salle stock en fonction des ressources	2 days ago	TWTWGMM-38
336da47	TWTWGMM-40 Creation et implementation de la classe	2 days ago	TWTWGMM-40
daa568d	TWTWGMM-39 Creation et implementation de la classe	2 days ago	TWTWGMM-39
7084534	TWTWGMM-25 Modification de methode	2 days ago	TWTWGMM-25
eab5591	TWTWGMM-26 Modification de l'enum	2 days ago	TWTWGMM-26
b876abf	TWTWGMM-5 Modification des methodes	2 days ago	TWTWGMM-5
af70391	TWTWGMM-8 Definition d'un des	2 days ago	TWTWGMM-8
903a5f5	TWTWGMM-21 Definition d'un soldat	2 days ago	TWTWGMM-21
d7ca130	TWTWGMM-20 Definition d'une ouvriere	2 days ago	TWTWGMM-20
84be5fb	TWTWGMM-3 Definition d'une nourrice	2 days ago	TWTWGMM-3
399e9b0	TWTWGMM-22 Definition d'une larve	2 days ago	TWTWGMM-22
802fb3c	TWTWGMM-2 Definition d'une fourmi	2 days ago	TWTWGMM-2

Définition

git est un logiciel de gestion de versions décentralisé.

voir https://fr.wikipedia.org/wiki/Git

GitHub est un service web d'hébergement et de gestion de développement de logiciels
voir https://github.com/

GitLab is a web-based Git-repository manager with wiki, issue-tracking and CI/CD pipeline features, using an open-source license, developed by GitLab Inc.

https://about.gitlab.com/

GitLab Continuous Integration & Deployment

L'intégration continue est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier *à chaque modification* de code source que le résultat des modifications ne produit *pas de régression* dans l'application développée.

Le déploiement continu est une approche de génie logiciel dans lequel les équipes produisent des logiciels dans des cycles courts et veille à ce que le logiciel soit *fiable à tout moment*. Il vise à la construction, les essais et la diffusion des logiciels plus *rapidement* et plus *fréquemment*.

« wikipedia »

Intégration Continue

Vérifier que chaque modification de code n'engendre pas de régression.

- A chaque commit :

1. Compiler le code (standard, rapide, ...),
2. Lancer les tests unitaires,
3. Analyser le code,
4. Générer des rapports, ..., un exécutable ou une archive

GitLab

Gestion du code source

Intégration continue

Déploiement continu

Support à la gestion de projets :
issues, milestones, wiki,

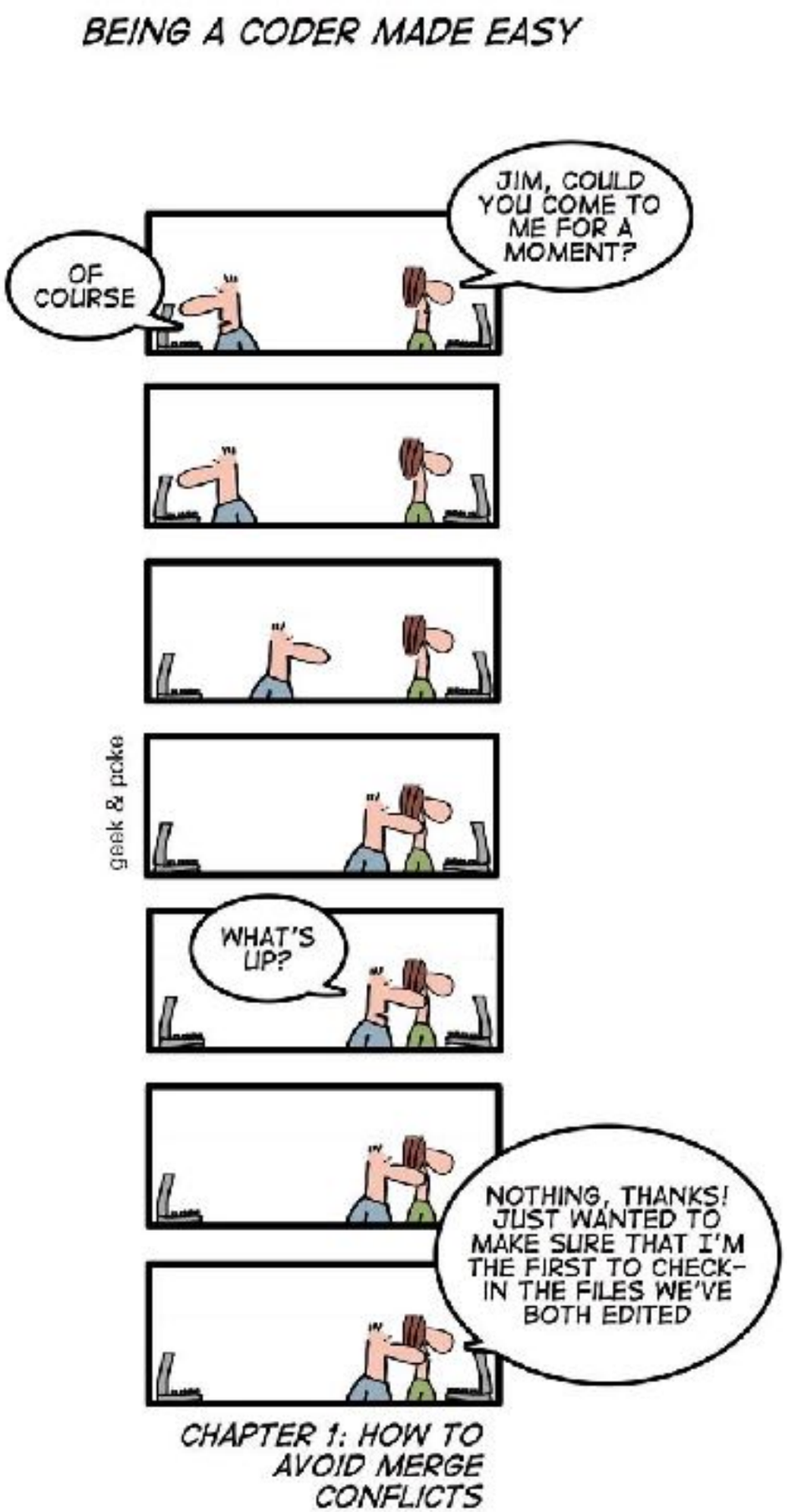
https://pxhere.com/fr/photo/1214452

Conclusions

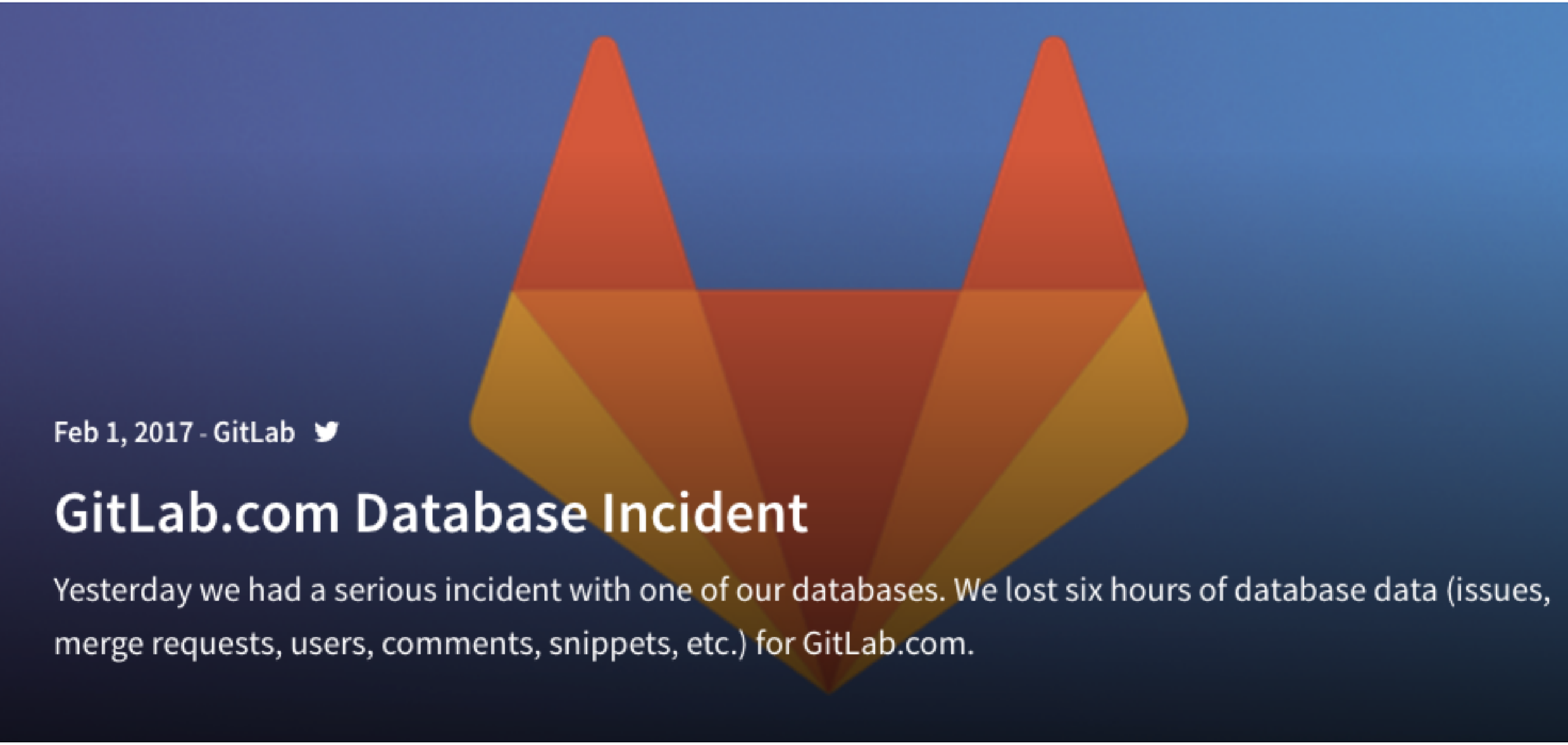
Rôles d’un gestionnaire de versions

- Fonction 1
  - Gérer un historique qui permette
    - D’enregistrer de nouvelles révisions à tout moment
    - De récupérer n’importe quelle révision enregistrée
- Fonction 2
  - Documenter chaque révision en lui associant un message
- Fonction 3
  - Noter l’auteur de chaque révision
  - Ceci permet d'associer un responsable à chaque ligne de code dans chaque révision
- Fonction 4
  - Faciliter le travail en multipostes en permettant l'accès à distance à un dépôt partagé par tousles postes
- Fonction 5
  - Faciliter la fusion des modifications en gérant les conflits
- Fonction 6
  - Faciliter le développement parallèle de multiples branches et le transfert de modifications entre branches


In case of fire
1. git commit
2. git push
3. leave building
EXTINCTEUR
CLASSES AB
EXTINCTEUR
CLASSE B



credits: C.Line design & sxc.hu



credits: C.Line design & sxc.hu

References

http://www.git-tower.com/files/cheatsheet/Git_Cheat_Sheet_grey.pdf

https://fr.slideshare.net/VincentComposieux/gitlab-ci-integration-et-dploiement-continue

http://git-scm.com/book/fr

Appendix : Basic git commands

Git task	Notes	Git commands
Tell Git who you are	Configure the author name and email address to be used with your commits. Note that Git strips some characters (for example trailing periods) from user.name.	git config --global user.name "Sam Smith" git config --global user.email sam@examp
Create a new local repository		git init
Check out a repository	Create a working copy of a local repository:	git clone /path/to/repository
	For a remote server, use:	git clone username@host:/path/to/reposit

Appendix : Basic git commands

Git task	Notes	Git command
Add files	Add one or more files to staging (index):	git add <filename> git add *
Commit	Commit changes to head (but not yet to the remote repository):	git commit -m "Commit message"
	Commit any files you've added with git add, and also commit any files you've changed since then:	git commit -a

https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html

Appendix : Basic git commands

Git task	Notes	Git commands
Status	List the files you've changed and those you still need to add or commit:	git status
Connect to a remote repository	If you haven't connected your local repository to a remote server, add the server to be able to push to it:	git remote add origin <server>
	List all currently configured remote repositories:	git remote -v

https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html

Appendix : Basic git commands

Git task	Notes	Git comman
Branches	Create a new branch and switch to it:	git checkout -b <branchna
	Switch from one branch to another:	git checkout <branchname
	List all the branches in your repo, and also tell you what branch you're currently in:	git branch
	Delete the feature branch:	git branch -d <branchname
	Push the branch to your remote repository, so others can use it:	git push origin <branchnar
	Push all branches to your remote repository:	git push --all origin
	Delete a branch on your remote repository:	git push origin :<branchna

Appendix : Basic git commands

Git task	Notes	Git commands
Update from the remote repository	Fetch and merge changes on the remote server to your working directory:	git pull
	To merge a different branch into your active branch:	git merge <branchname>
	View all the merge conflicts: View the conflicts against the base file: Preview changes, before merging:	git diff git diff --base <filename> git diff <sourcebranch> <targetbranch>
	After you have manually resolved any conflicts, you mark the changed file:	git add <filename>

https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html

Appendix : Basic git commands

Git task	Notes	Git commands
Tags	You can use tagging to mark a significant changeset, such as a release:	git tag 1.0.0 <commitID>
	CommitId is the leading characters of the changeset ID, up to 10, but must be unique. Get the ID using:	git log
	Push all tags to remote repository:	git push --tags origin

https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html

Appendix : Basic git commands

Git task	Notes	Git commands
Undo local changes	If you mess up, you can replace the changes in your working tree with the last content in head: Changes already added to the index, as well as new files, will be kept.	git checkout -- <filename>
	Instead, to drop all your local changes and commits, fetch the latest history from the server and point your local master branch at it, do this:	git fetch origin git reset --hard origin/master
Search	Search the working directory for foo():	git grep "foo()"

https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html