Collaborative Development & Source Code Versioning

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

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 la forge.
- Dans la suite des TDs etc, vous utilisez la gestion de version

➡ Ensuite vous avez votre propre environnement pour gérer vos projets et vous vous auto-organizez.

Problématiques

- Travail en équipe
- Monitorer les changements
- Sauvegarde sur une machine distante
- Gestion de plusieurs «versions» d’un projet
  - Pouvoir revenir en arrière

https://momastery.com/blog/2012/05/09/van-tastic-mothers-day-love-flash-mob/

« Collaborative Development ?»

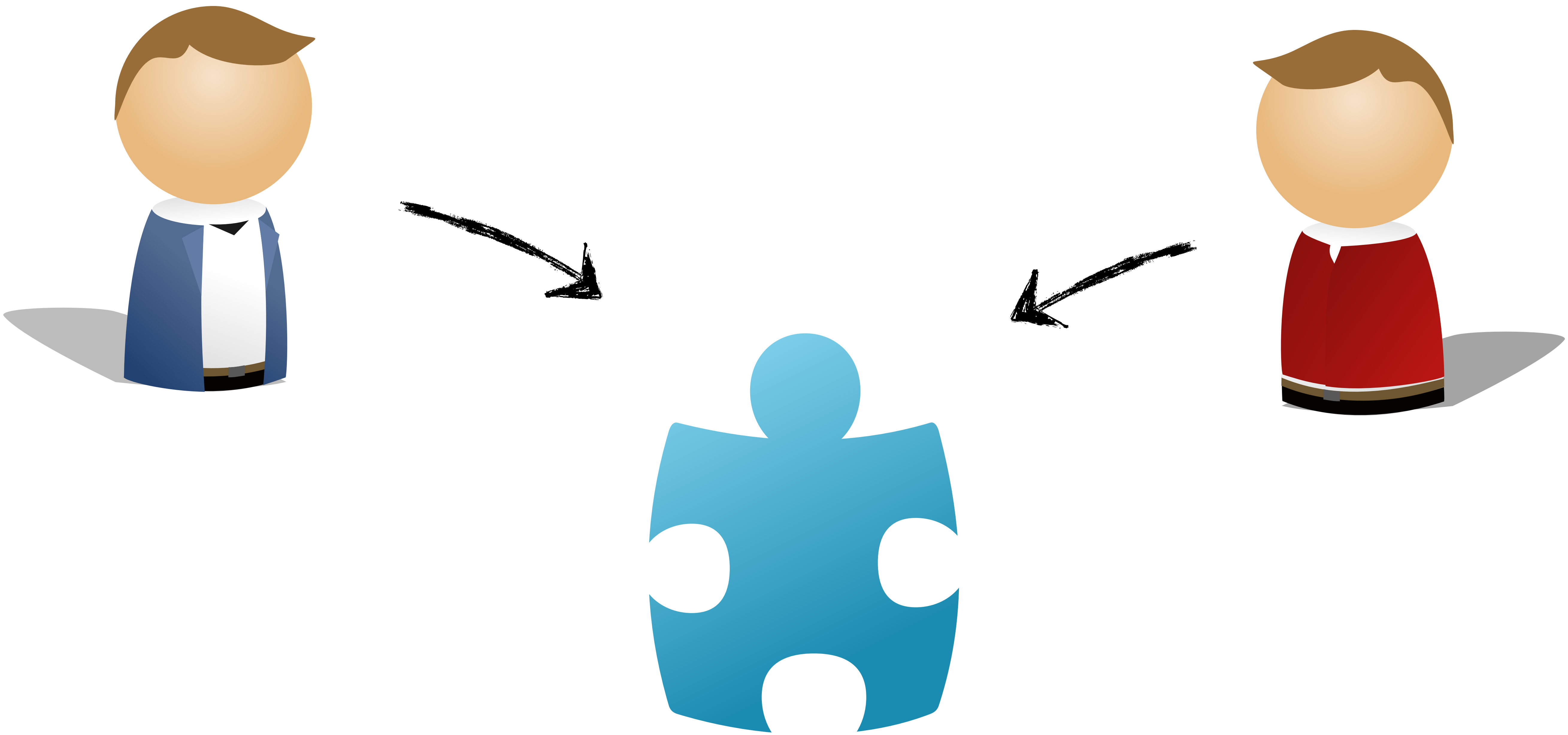
if you want to go fast, go alone.
If you want to go far, go together.

proverbe africain

works on

developer

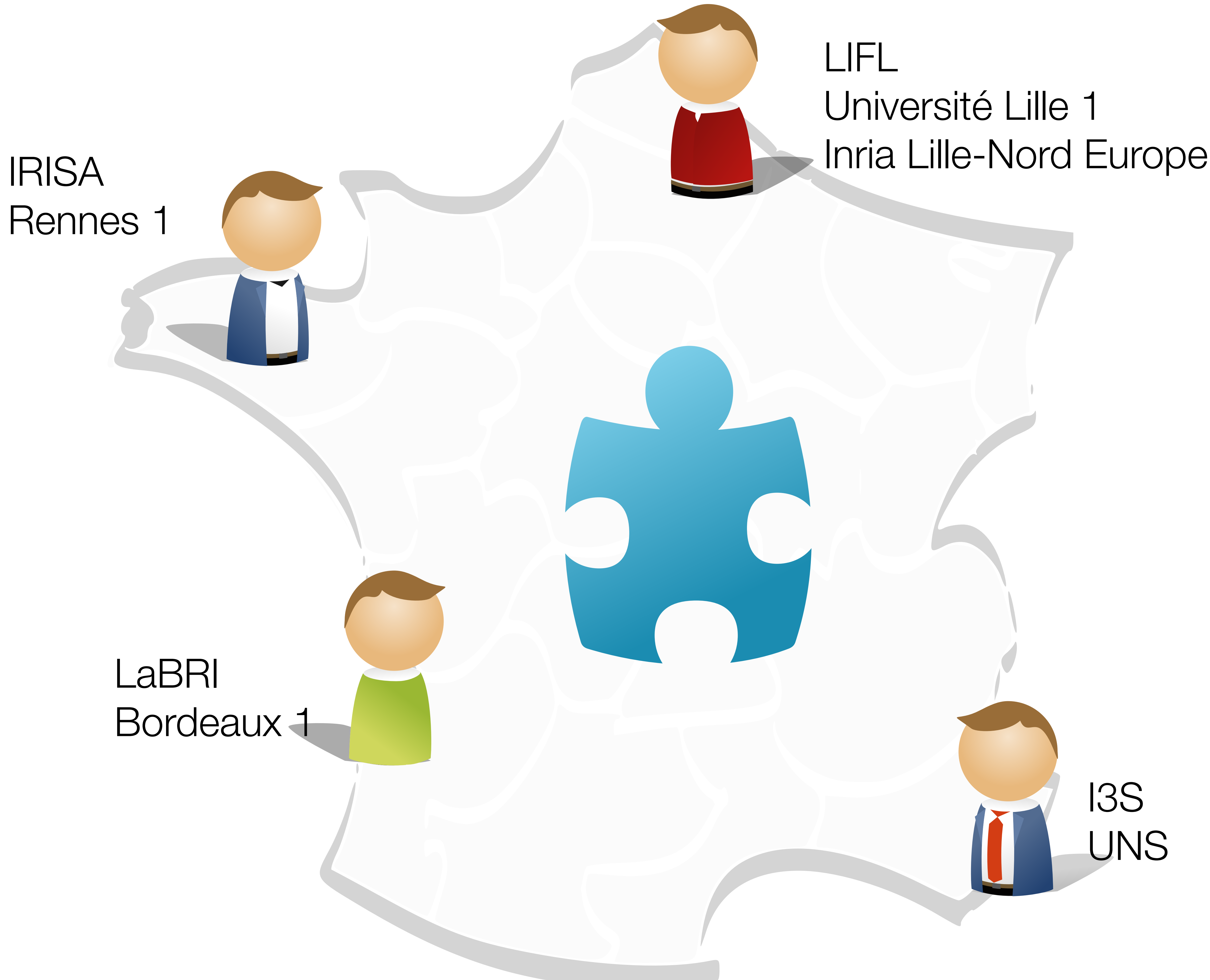
piece of
software

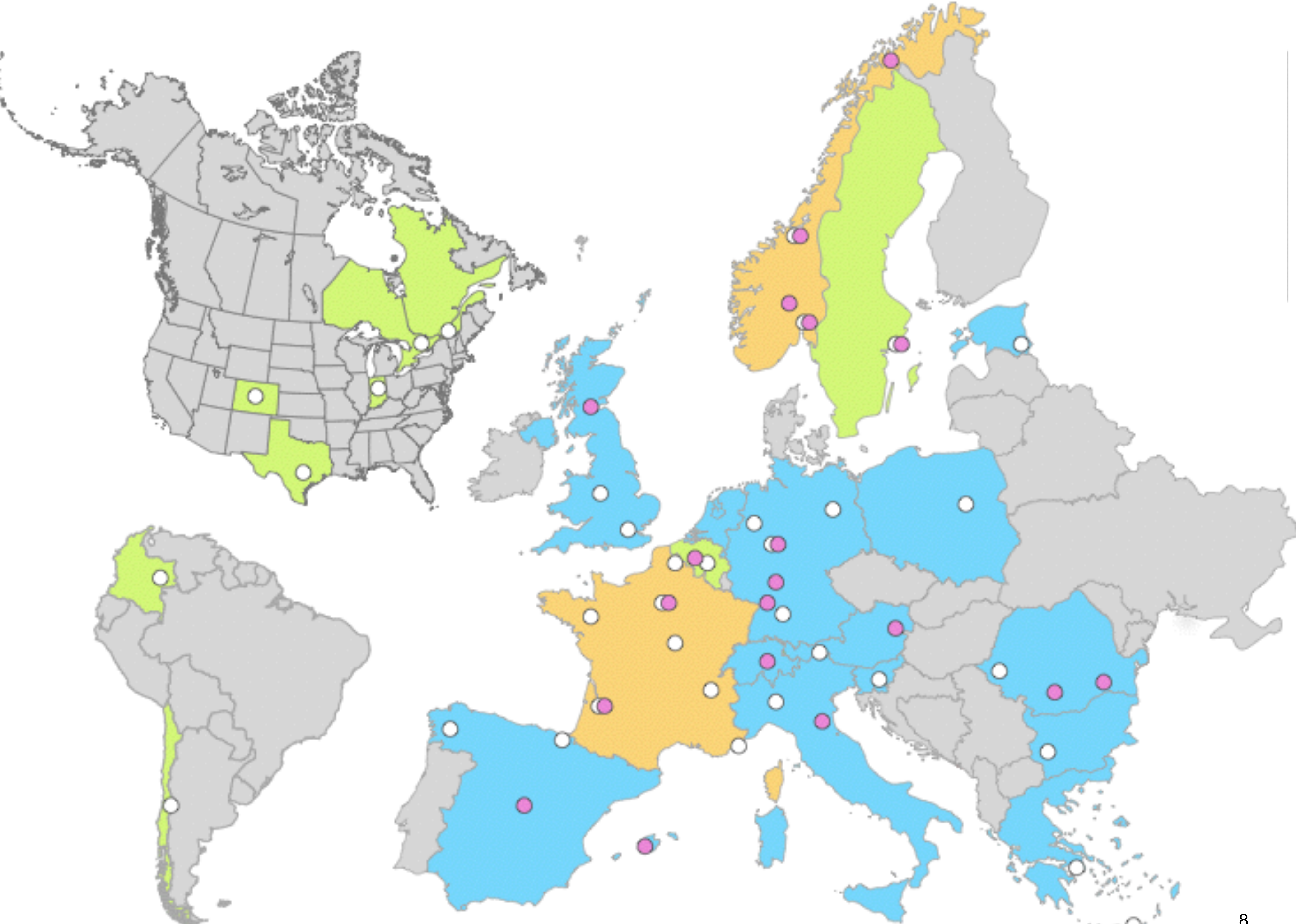


Email?

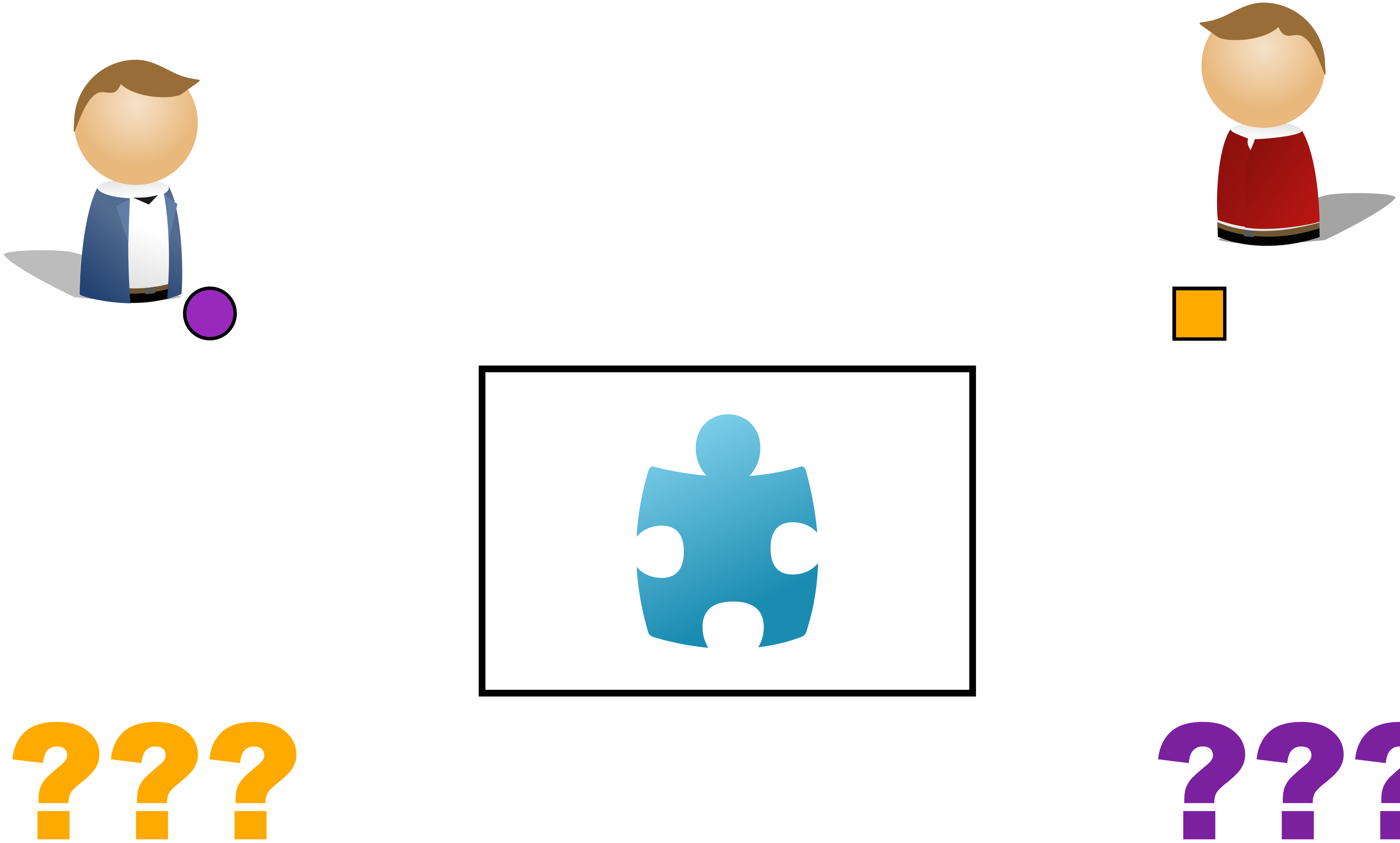
Shared directory?

USB Key?

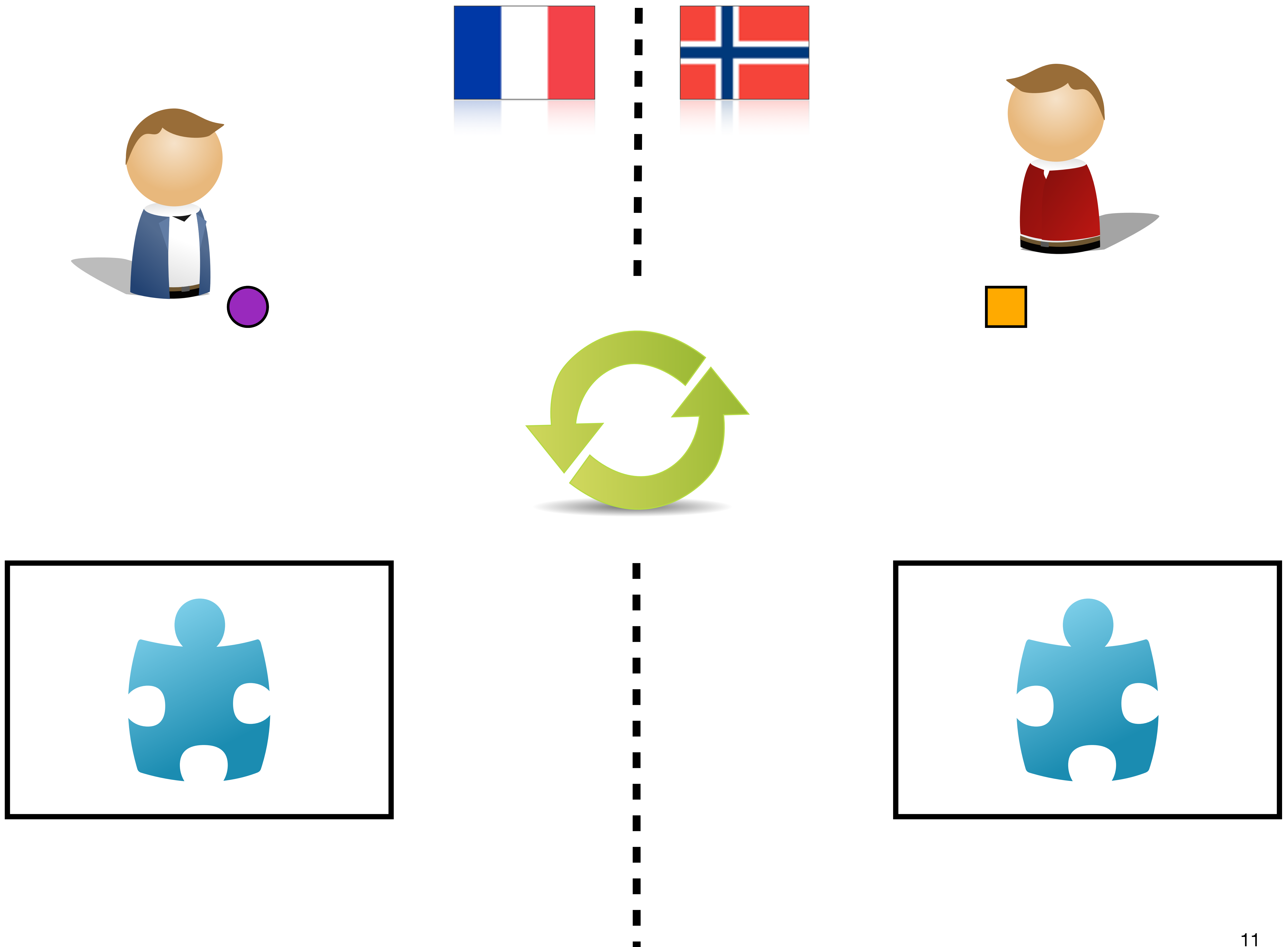

LIFL
Université Lille 1
Inria Lille-Nord Europe
IRISA
Rennes 1
LaBRI
Bordeaux 1
I3S
UNS



Sébastien Mosser



To share changes!



Sébastien Mosser

«Why do we version source code?»

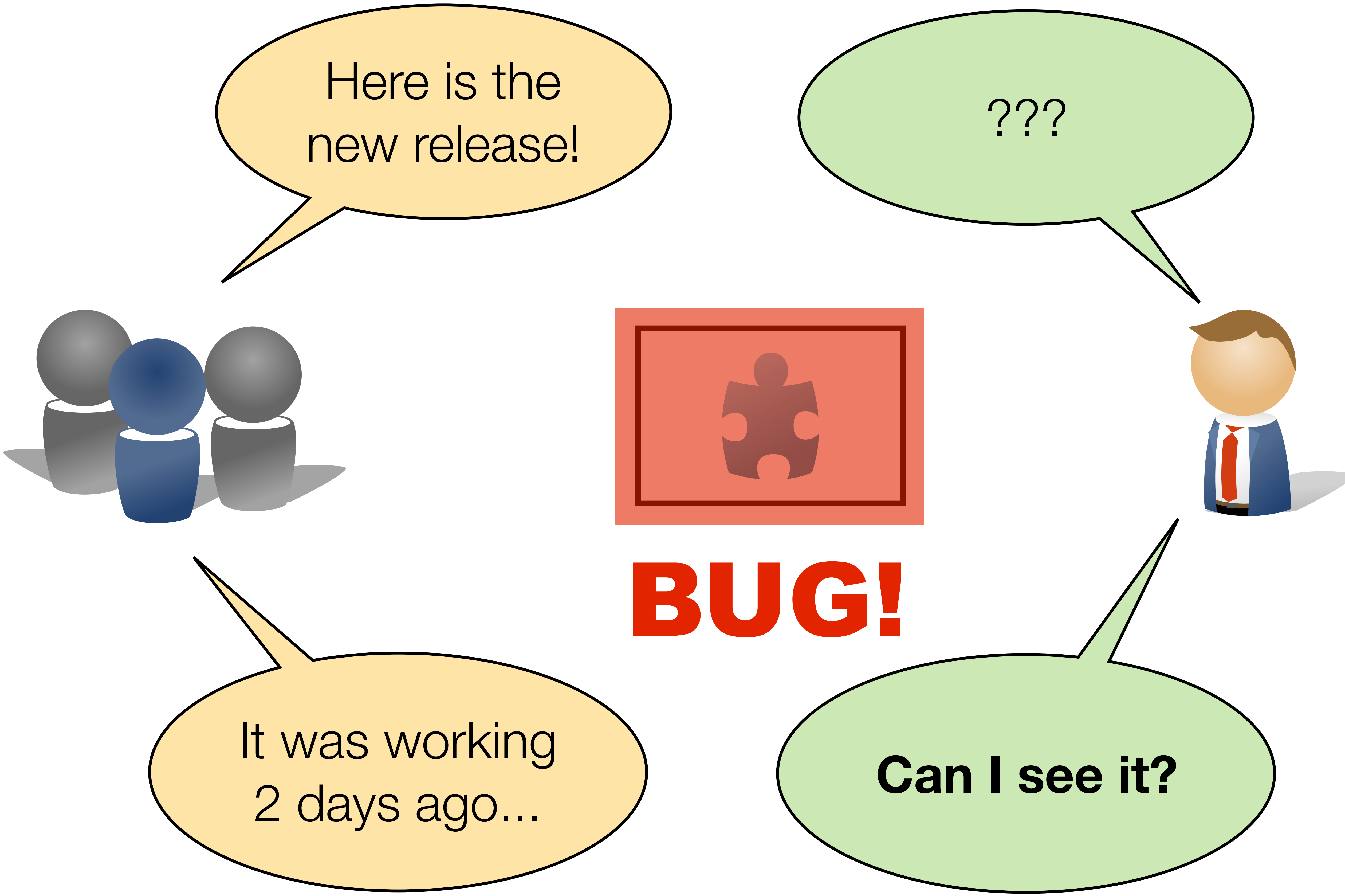
Motivations
(among others)

Sébastien Mosser

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

«Why do we version source code?»

To keep version history
To be able to rollback

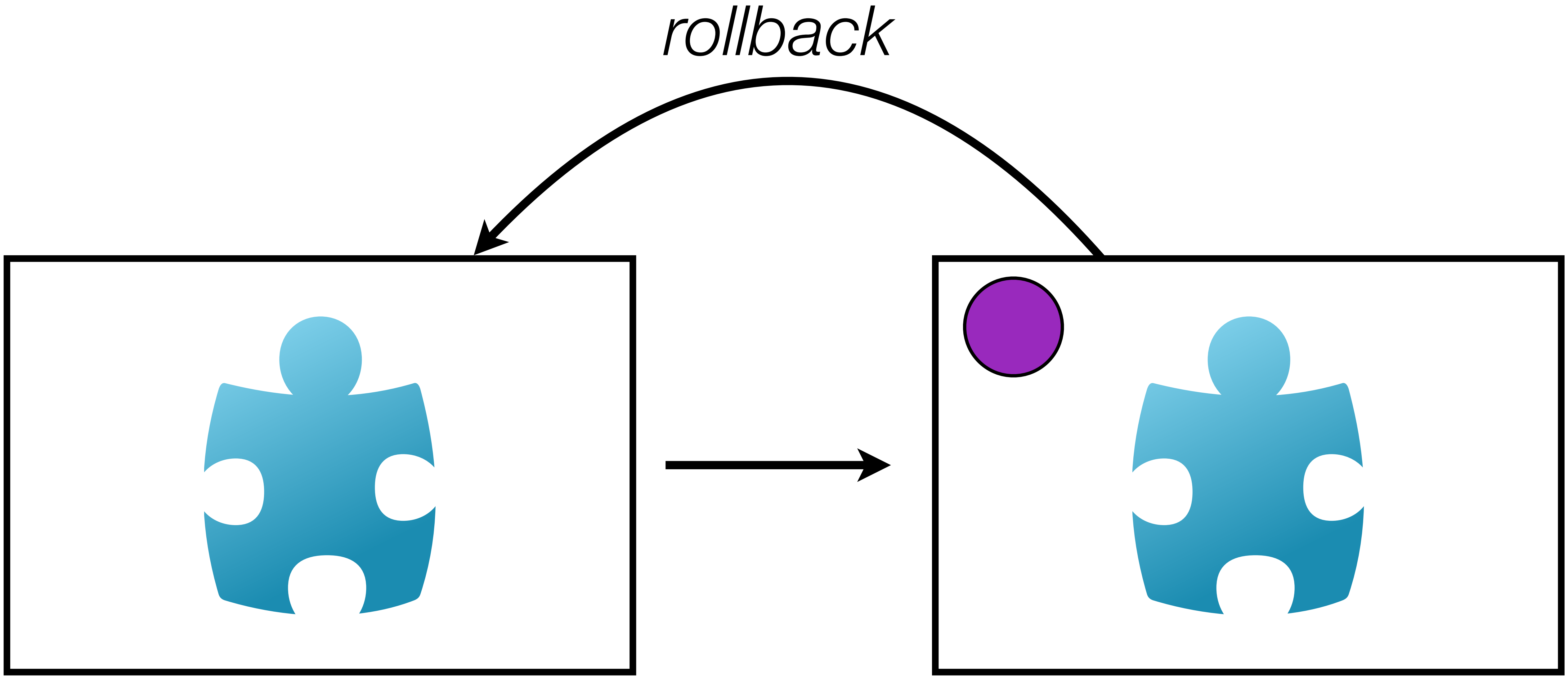

Here is the new release!
???
BUG!
It was working 2 days ago...
Can I see it?

Problématique

- Cas : édition d'un document

- Erreur de manipulation ?
- Comment récupérer ?
  – Voyager dans le temps pour visiter une version antérieure du document
- **Version/Révision**
  – Une révision d'un document ou d'un projet est un *instantané* pris à un instant donné et sauvegardé dans l'historique du projet

To rollback changes!


rollback

Rôles d’un gestionnaire de version

- Fonction 1
  - Gérer un historique qui permet
    - D’enregistrer de nouvelles révisions à tout moment
    - De récupérer n’importe quelle révision enregistrée

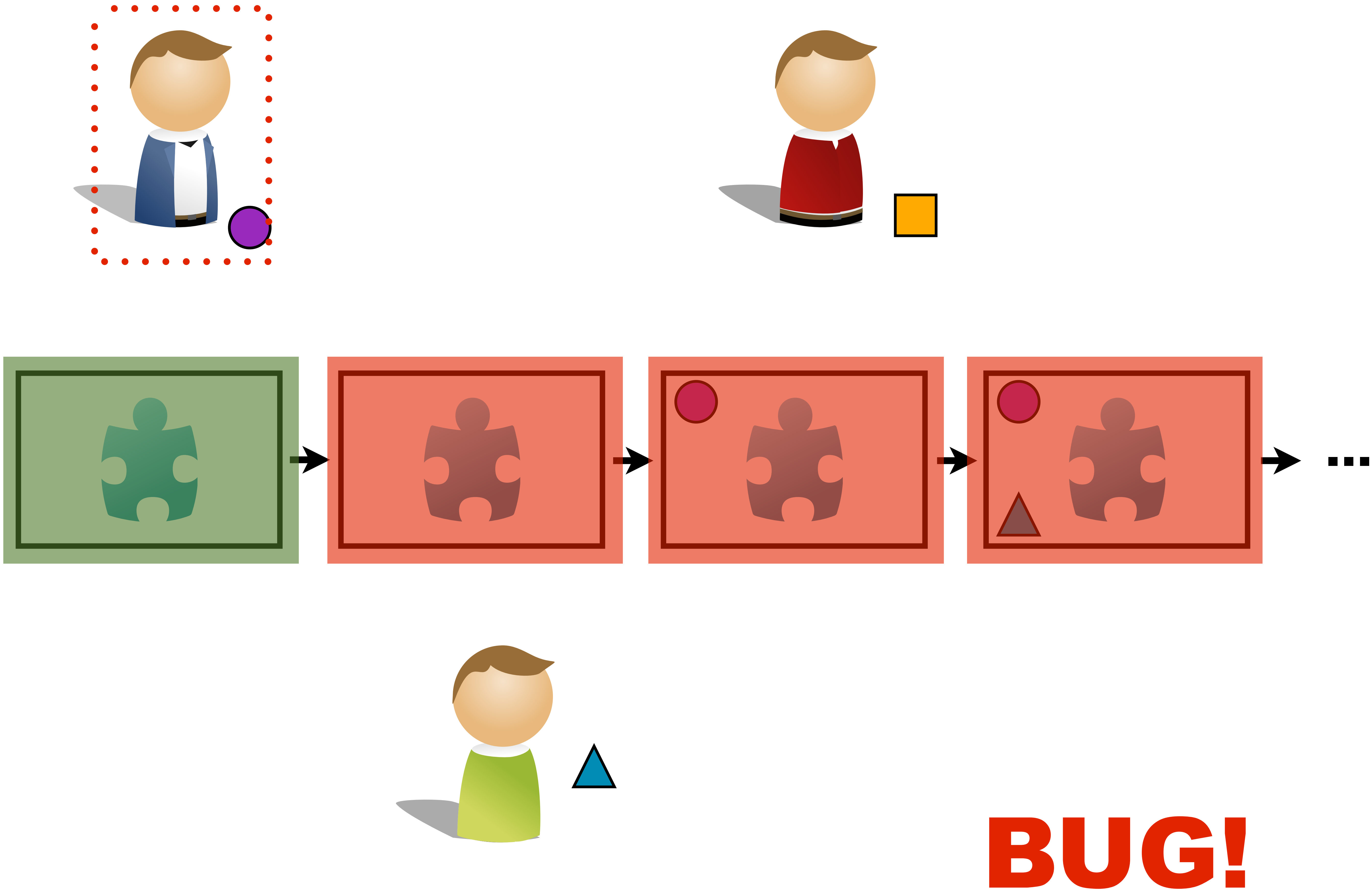
http://blogs.wsj.com/photojournal/2012/03/13/photos-of-the-day-march-13/

«Why do we version source code?»

To know who worked on...


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

To trace changes!


...
BUG!

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**

Rôles d’un gestionnaire de version

- 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 révision**

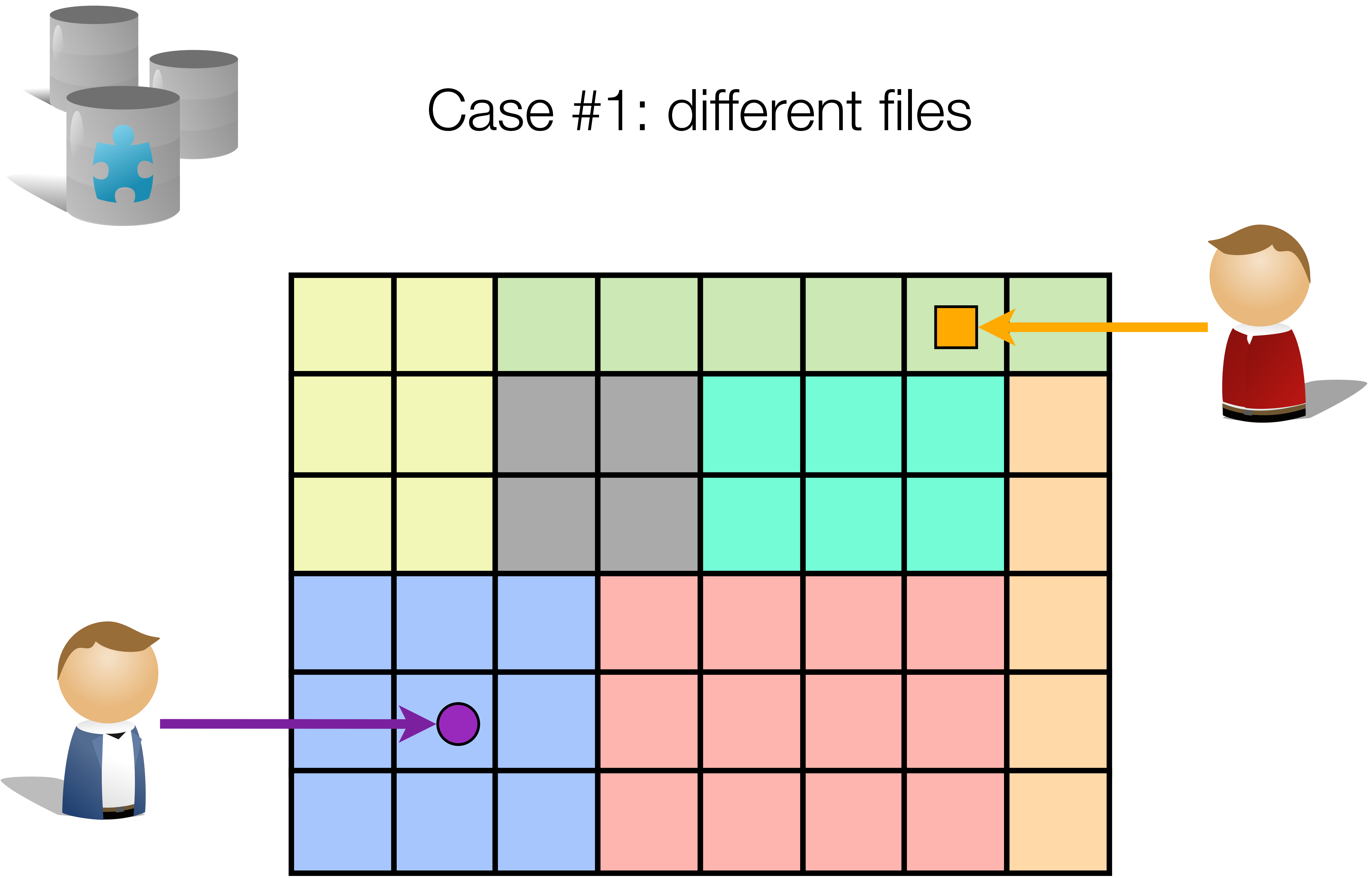
http://paysageislande.blogspot.fr/2013/06/periple-dans-le-sud-de-lislande-jour-5.html

«Why do we version source code?»

To share changes
To support merging

MBF

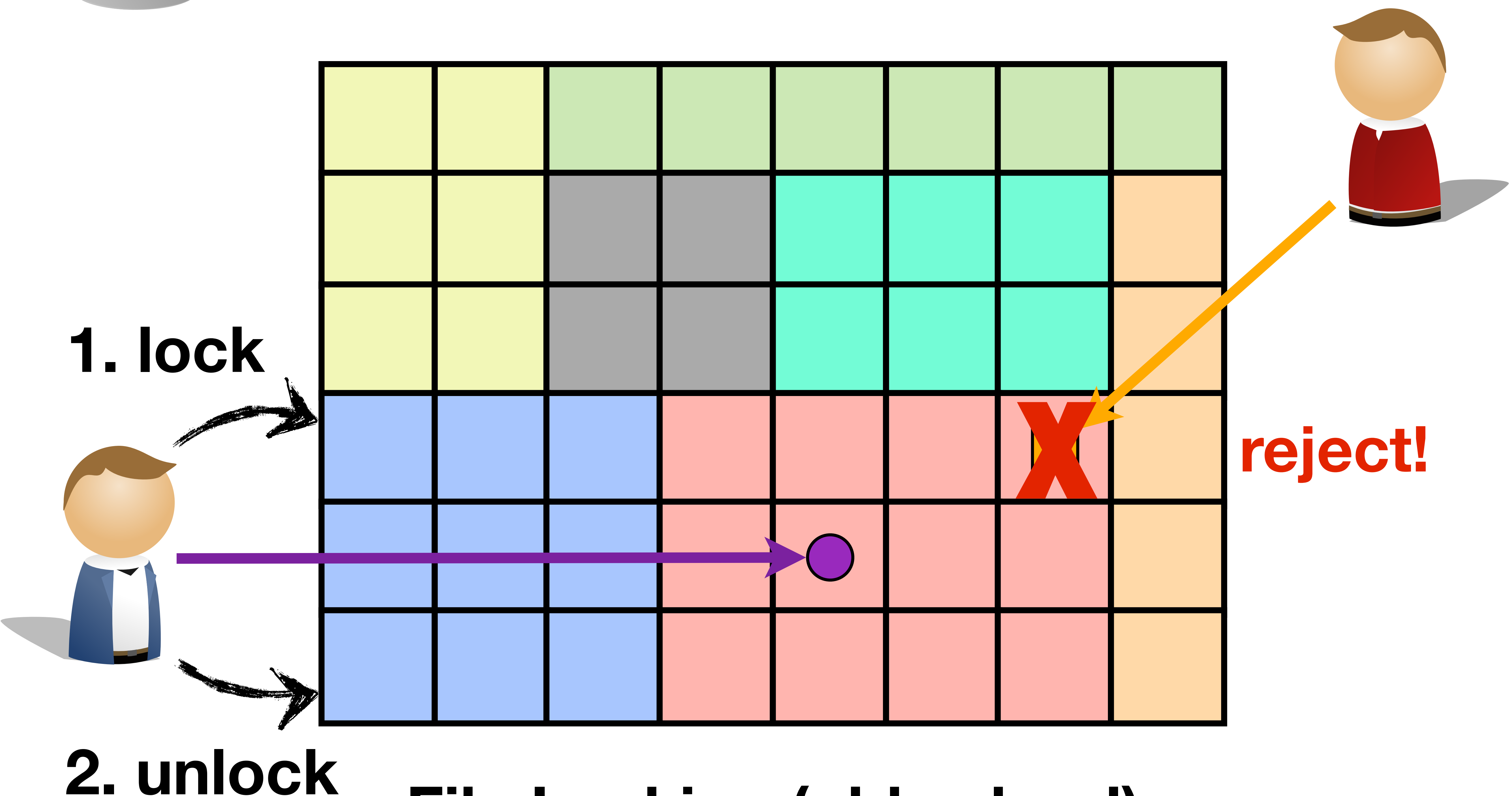
Case #1: different files



Atomic operations. No problem at all!

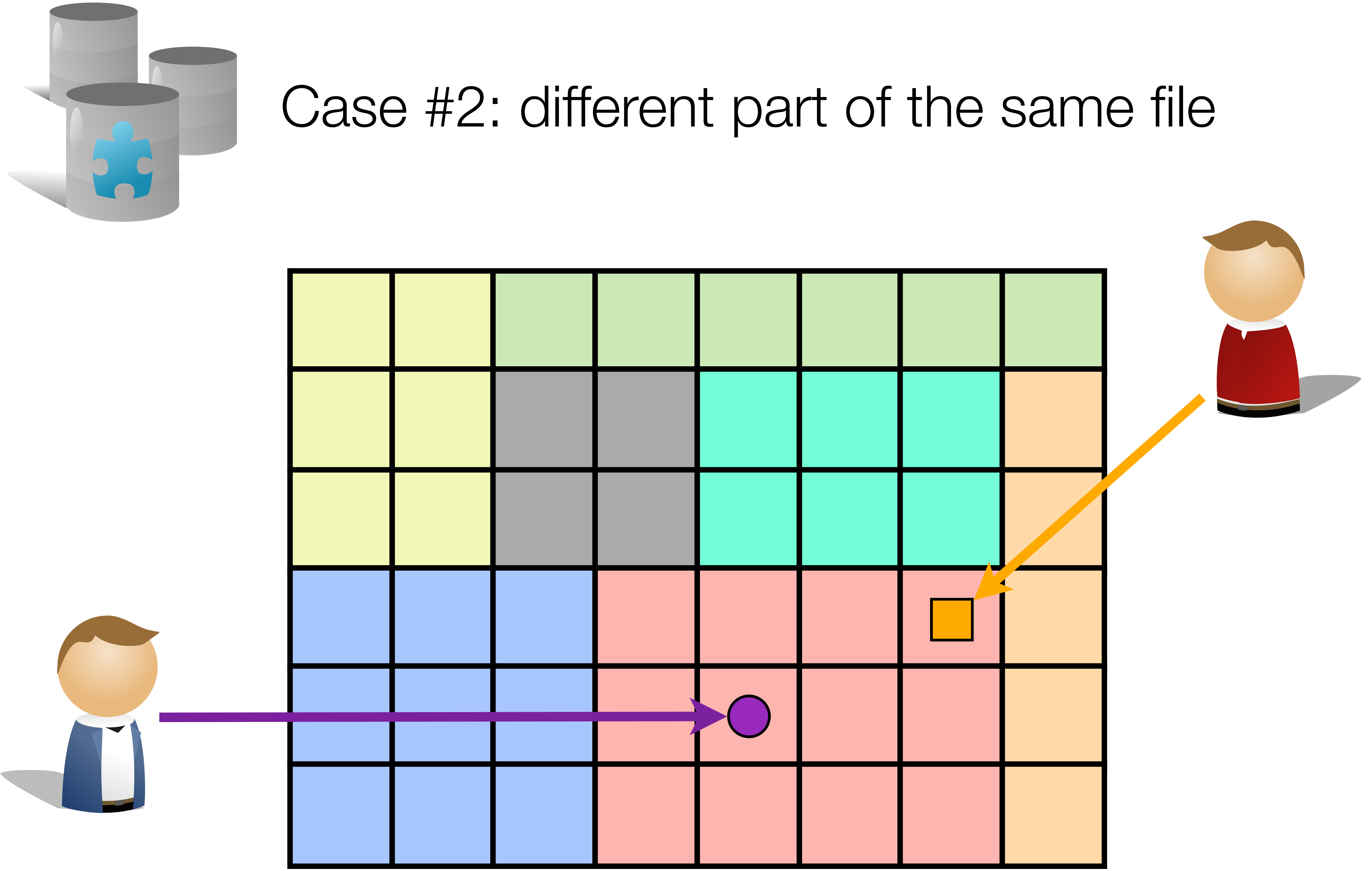
Sébastien Mosser

Case #2: different part of the same file



File Locking (old school)

Case #2: different part of the same file



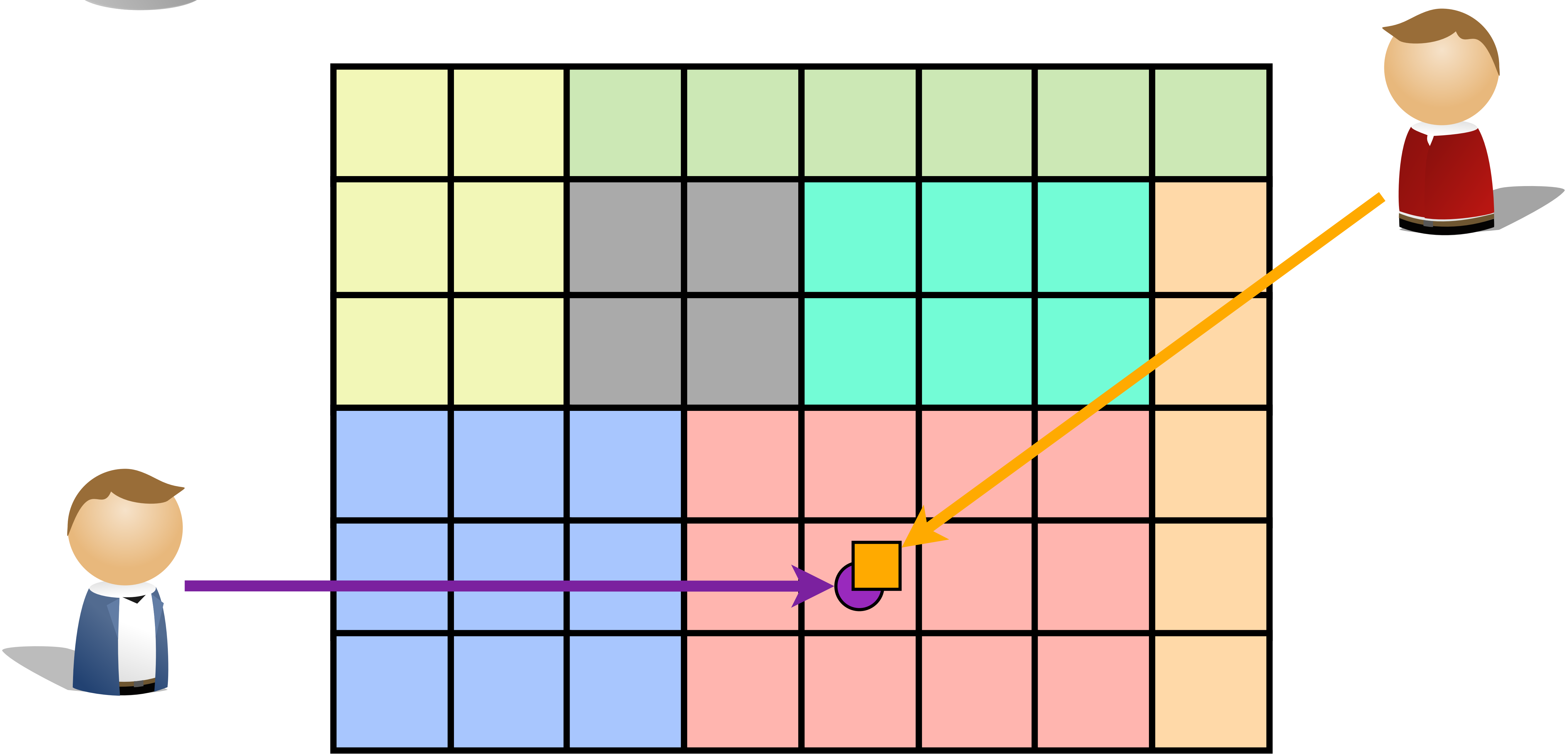
Automatic merge

Sébastien Mosser

ScalaMoF

```
@@ -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]()
```

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



Conflict!

Problématique

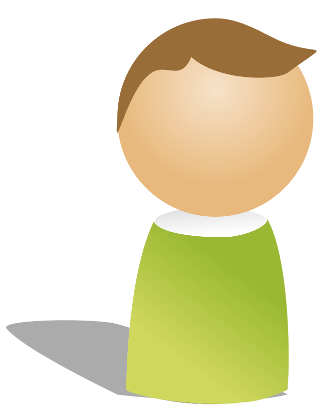
- Cas : travail en équipe

Dépôt initial

jeu.h jeu.c

A fait une copie de travail

jeu.h jeu.c



B fait une copie de travail

jeu.h jeu.c

Problématique

- Cas : travail en équipe

Dépôt initial

A modifie jeu.h

B modifie jeu.c

- A & B doivent échanger leurs modifs

➡Le gestionnaire de version va servir d'intermédiaire

Problématique

- Cas : travail en équipe (autre cas)

Dépôt initial

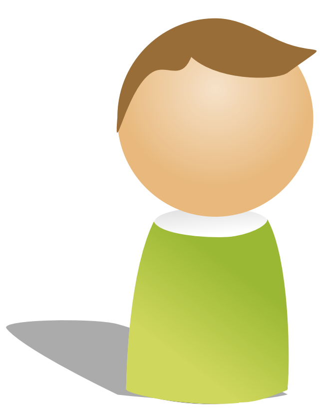
jeu.h

jeu.c

A fait une copie de travail

jeu.h

jeu.c



B fait une copie de travail

jeu.h

jeu.c

Problématique

- Cas 4 : travail en équipe (autre cas)

Dépôt initial

A modifie jeu.c

B modifie jeu.c

- A doit intégrer les modifs de B
- B doit intégrer les modifs de A

➡ Problème de la fusion (merge)

Rôles d'un gestionnaire de version

- 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**

«Why do we version source code?»

To support multiple versions (Branches)

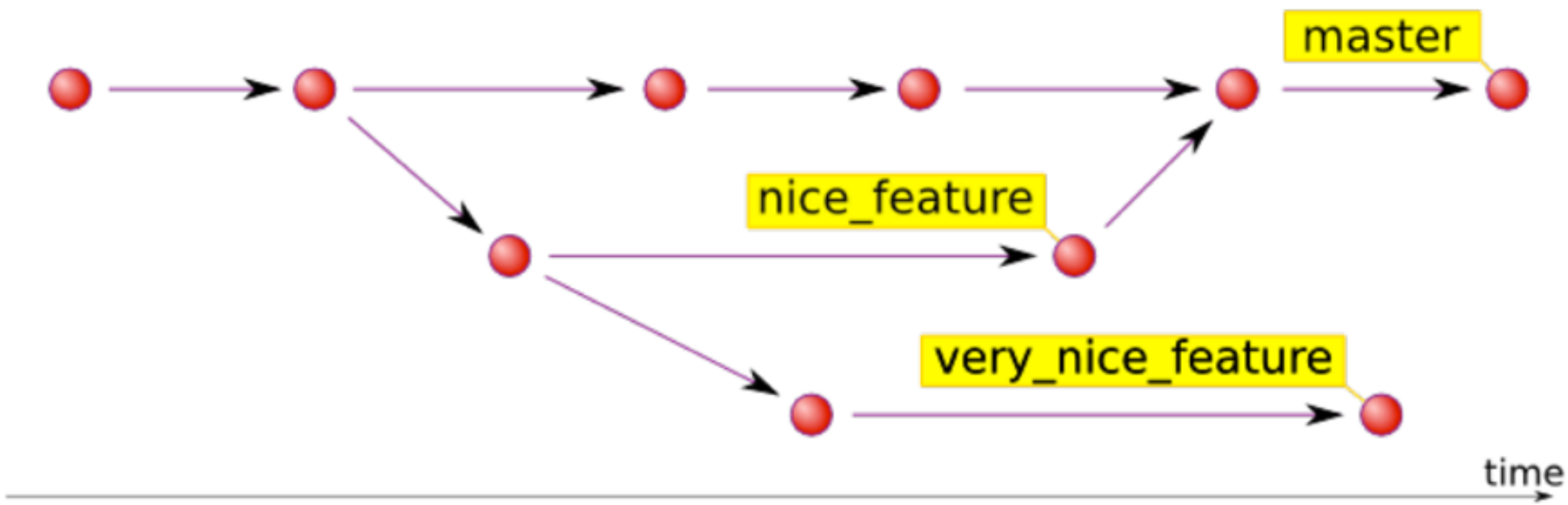
Problématique

- 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

Rôles d'un gestionnaire de versions

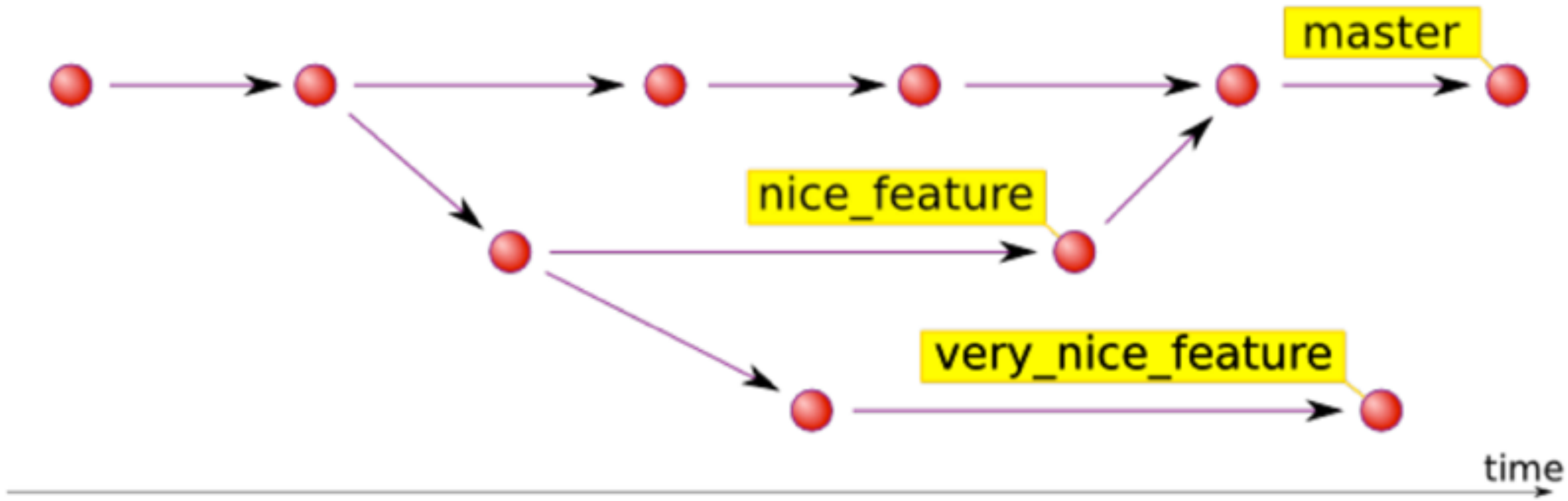
- Fonction 1
  - Gérer un historique qui permette
    - D'enregistrer de nouvelles révisions à tout moment
    - De récupérer n'importe quelle révision enregistrée
- Fonction 2
  - Documenter chaque révision en lui associant un message
- Fonction 3
  - Noter l'auteur de chaque révision
  - Ceci permet 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**

Les branches !



Chaque rond est un commit

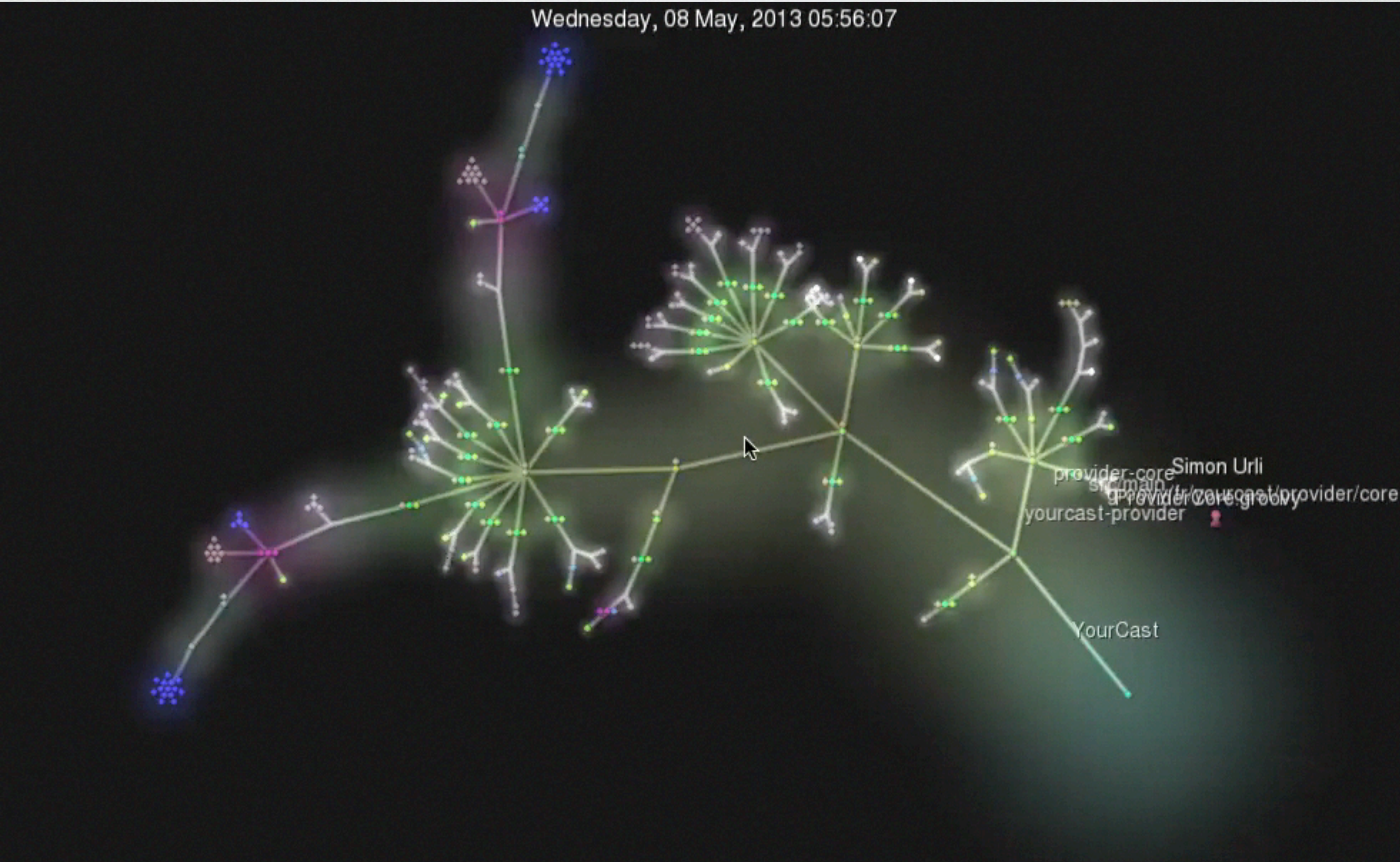
Les branches !



- 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

Les systèmes

Système	**Archive où ?**	**Archive quoi ?**	**Quand ?**
CVS	Centralisé	Fichiers	1989
SVN (Subversion ou tortoiseSVN)	Centralisé	Arborescence	2000
Git ou BZR	Décentralisé	Arborescence	2005


Wednesday, 08 May, 2013 05:56:07
provider-core
Simon Urli
yourcast-provider
YourCast

Attention, ...

Sébastien Mosser

Anti-informative Commit Messages

master

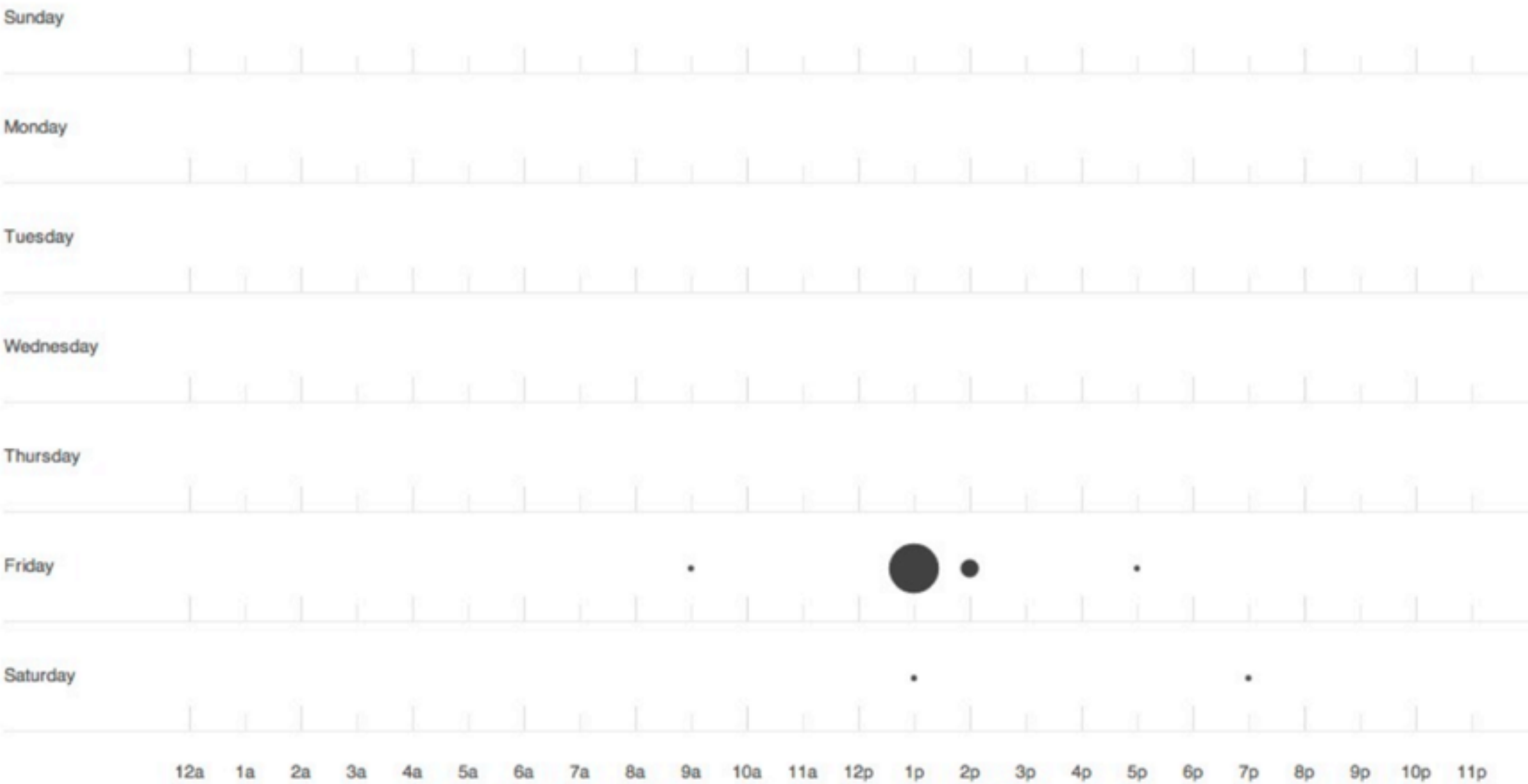
Author	Commit	Message
[redacted]	2285889	Version 22/02/2013
[redacted]	4e06542	Version 21/02/2013
[redacted]	5798097	Version 2 16/02/2013
[redacted]	775f97a	Version 16/02/2013

Commit message = Intention of the version

master

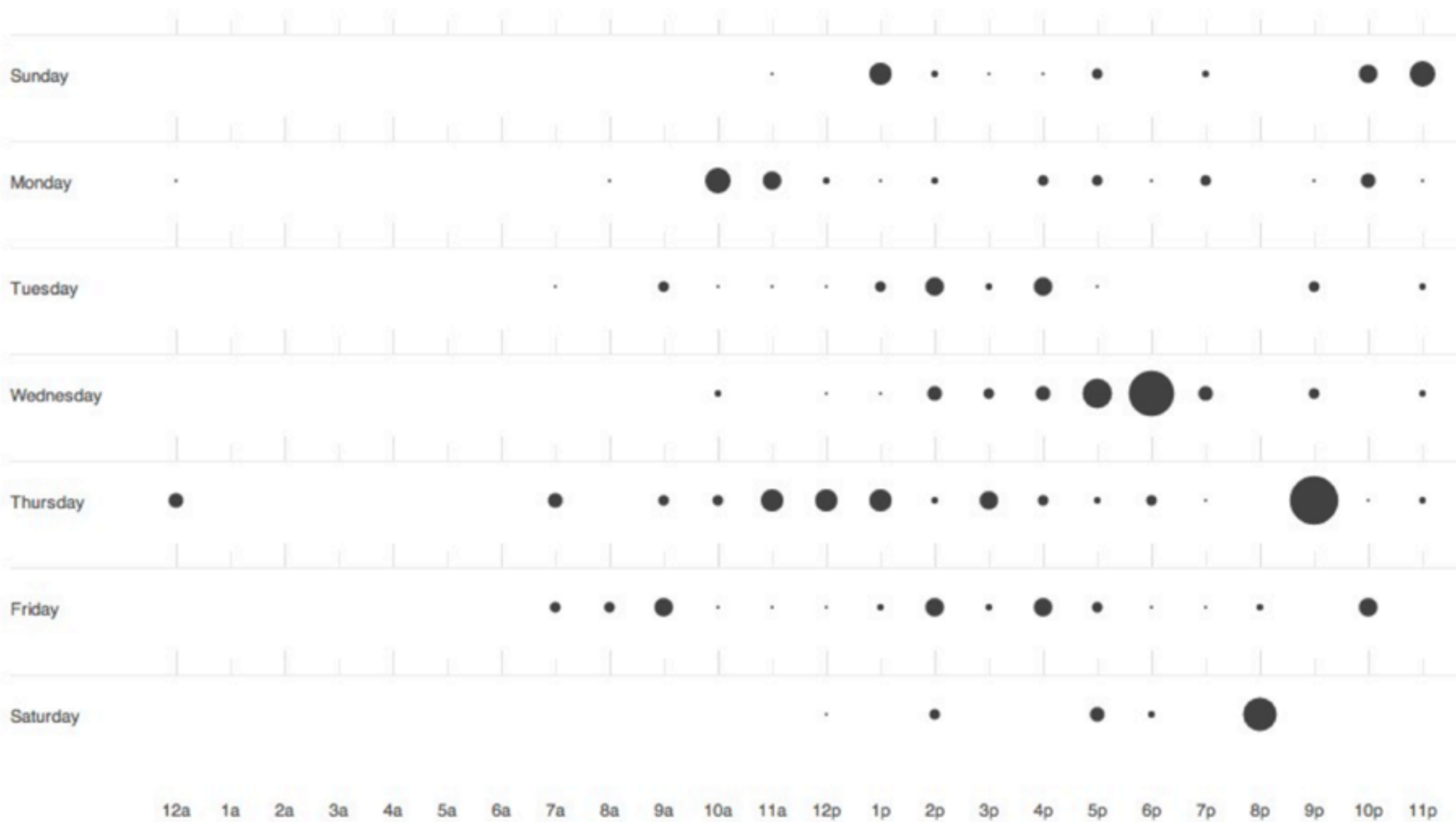
Author	Commit	Message
[redacted]	6ad5d7a	Correction des warnings
[redacted]	a2bd26c	Correction des erreures de nom de methode
[redacted]	7a3e625	Suppression de fourmiliere.java dans le pack

One-shot activity



Sébastien Mosser

Expected: Continuous Development



The Archive Commit

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~~

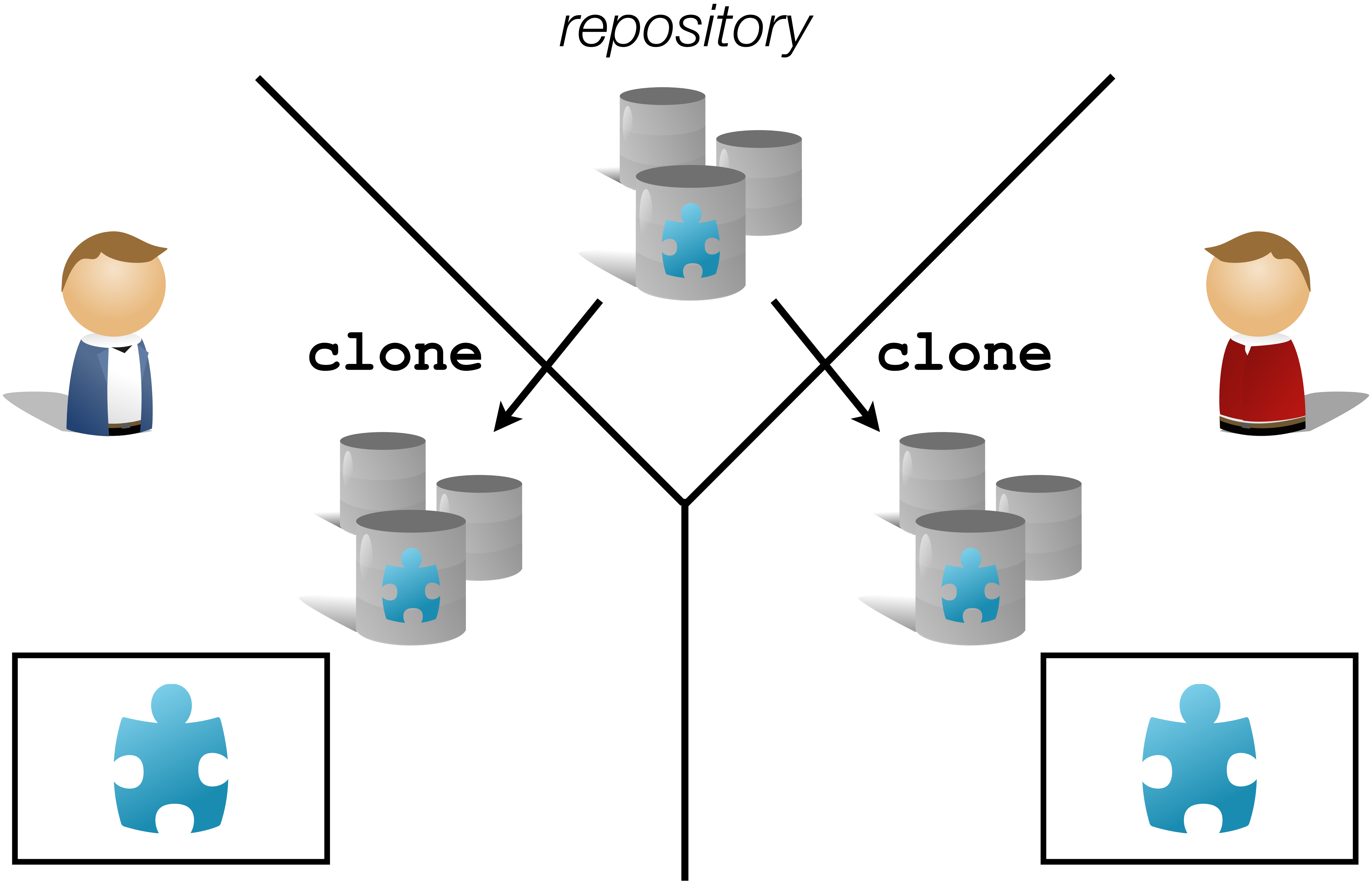
.gitignore

Commits must be related to tickets!

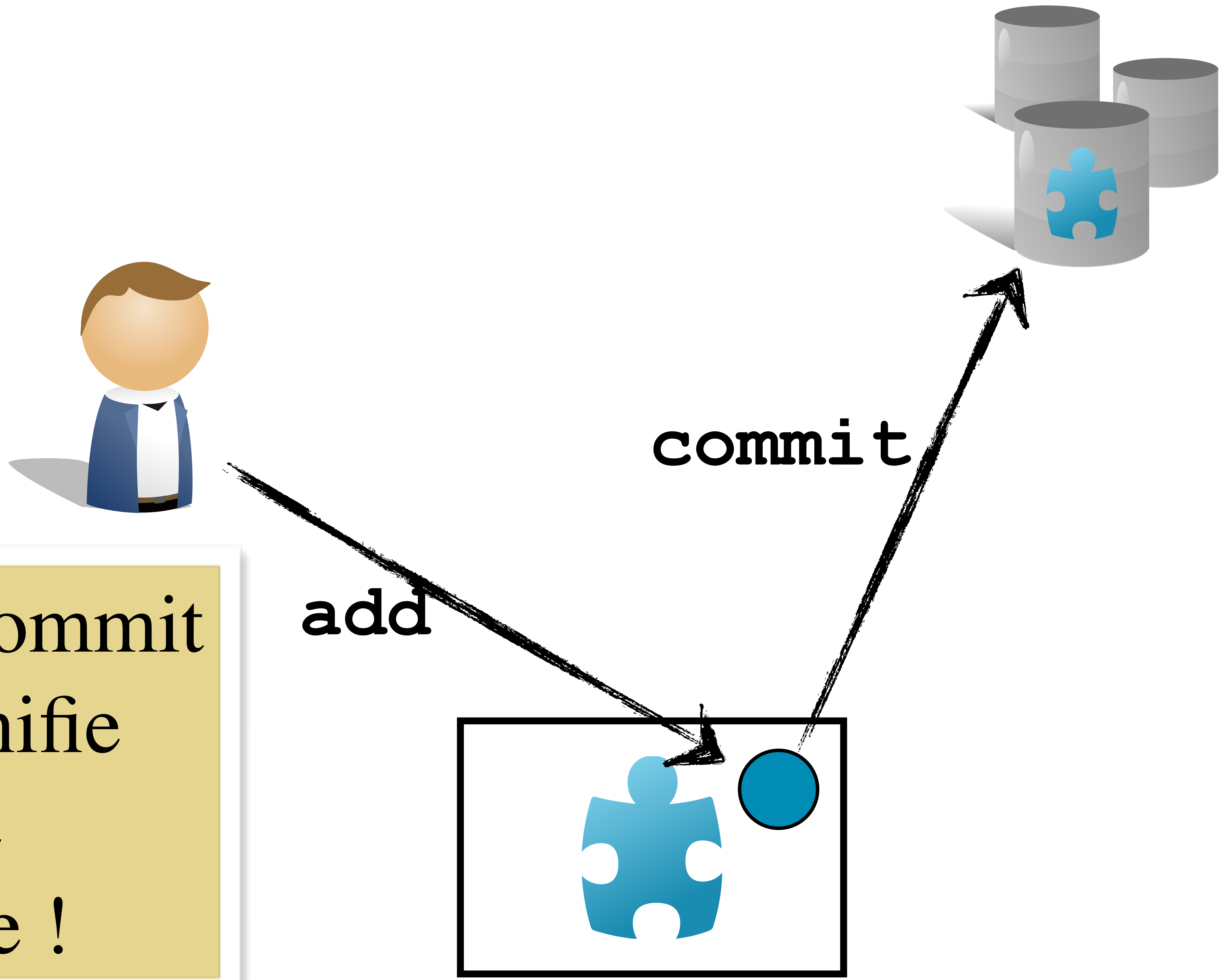
6f09558	TWTWGMM-38 Modification de la salle stock en fonction des ressources	2 days ago	TWTWGMM-38
336da47	TWTWGMM-40 Creation et implementation de la classe	2 days ago	TWTWGMM-40
daa568d	TWTWGMM-39 Creation et implementation de la classe	2 days ago	TWTWGMM-39
7084534	TWTWGMM-25 Modification de methode	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

Distributed Model

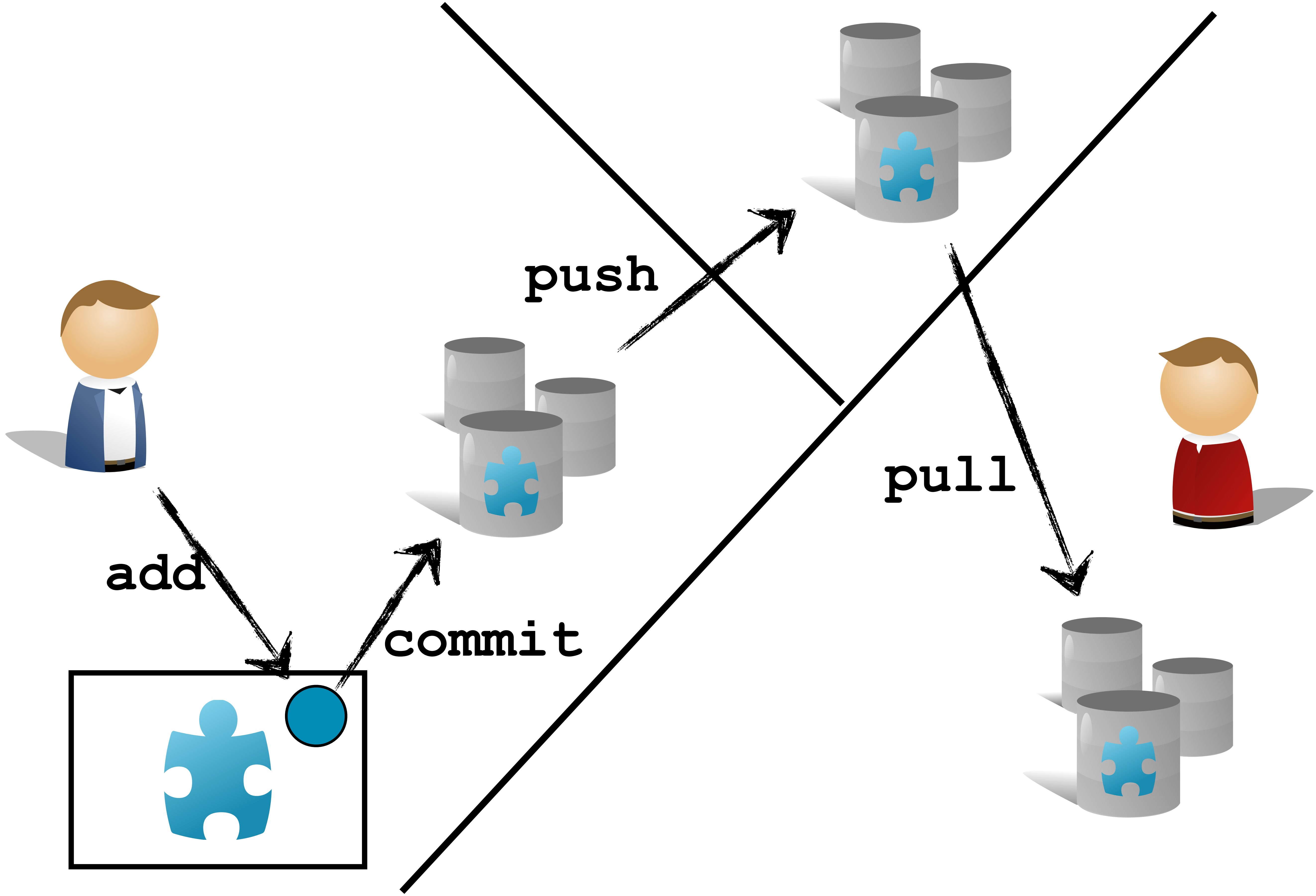
(e.g., Bazaar, Git)

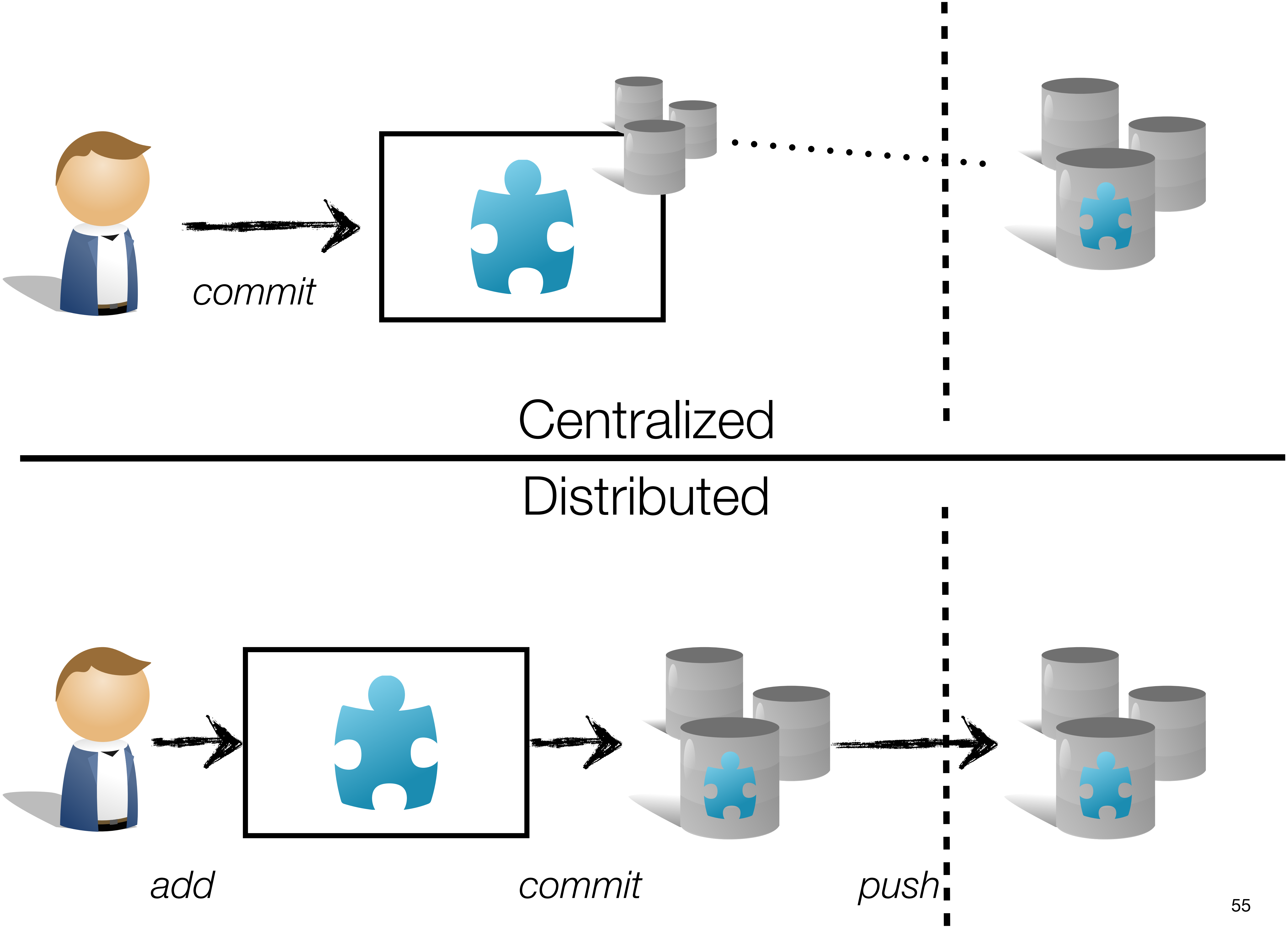

repository
clone
clone

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/

Conclusions

Why do we version code?

To trace changes!

To rollback changes!

To share changes!

(among others)

Different models for code versioning

Centralized

versus

Distributed

Distributed model to be used during labs

add

commit

push

https://try.github.io

References

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

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