Projet Informatique

2020

Steve Hostettler

Software Modeling and Verification Group

University of Geneva

Gestion du cycle de développement

Environnement de développement

Environnement de développement intégré:

Edition du code
Compilation
Aide au débogage
Déploiement
Test unitaire

Eclipse ou IDEA

Présentation d’Eclipse

Installation d’Eclipse (ou IDEA)

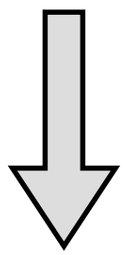
Gestion des versions

Partager les sources d’un projet

Archiver et sauvegarder les différentes versions

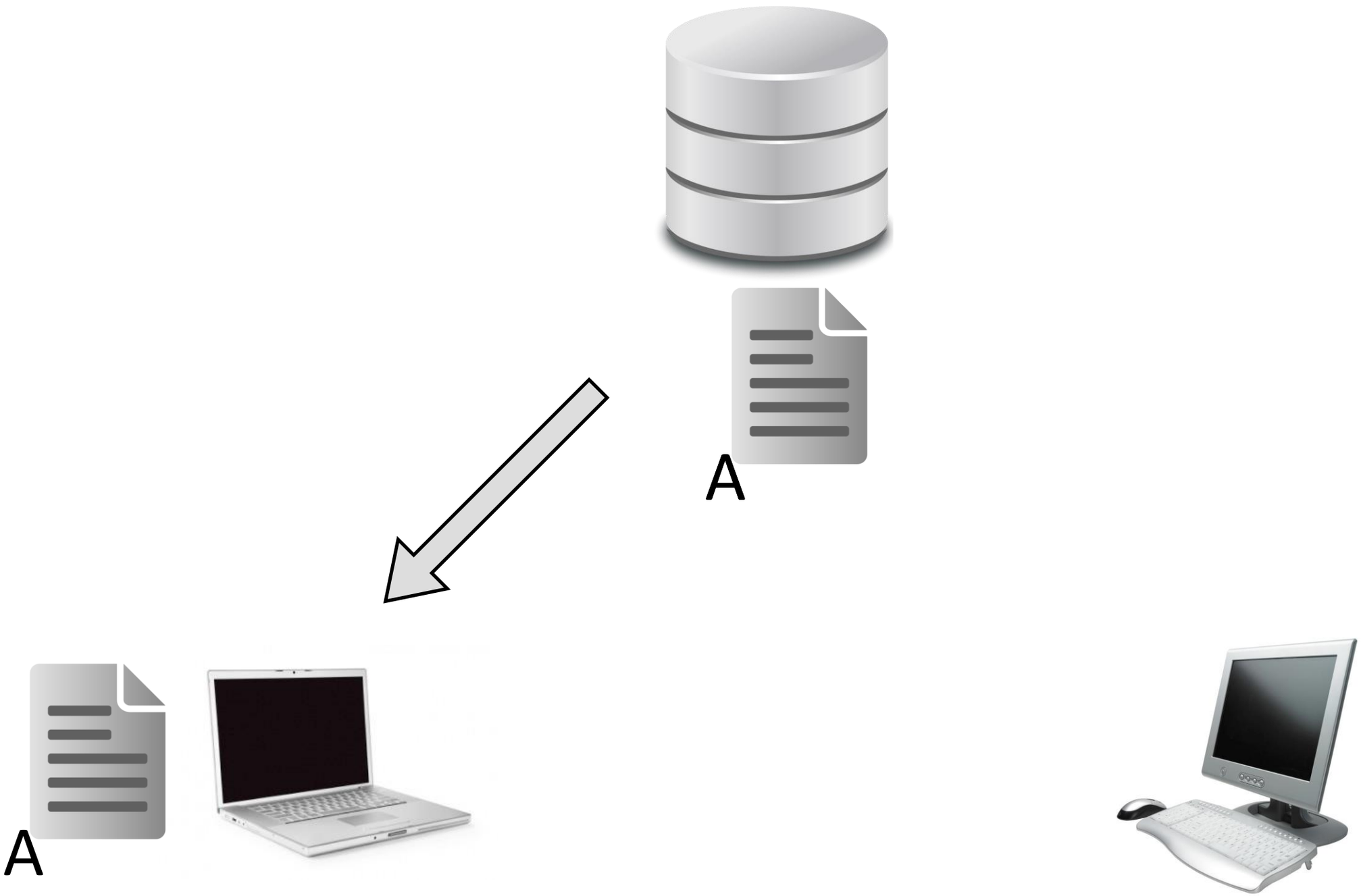
Revenir en arrière facilement

Pouvoir travailler hors-ligne

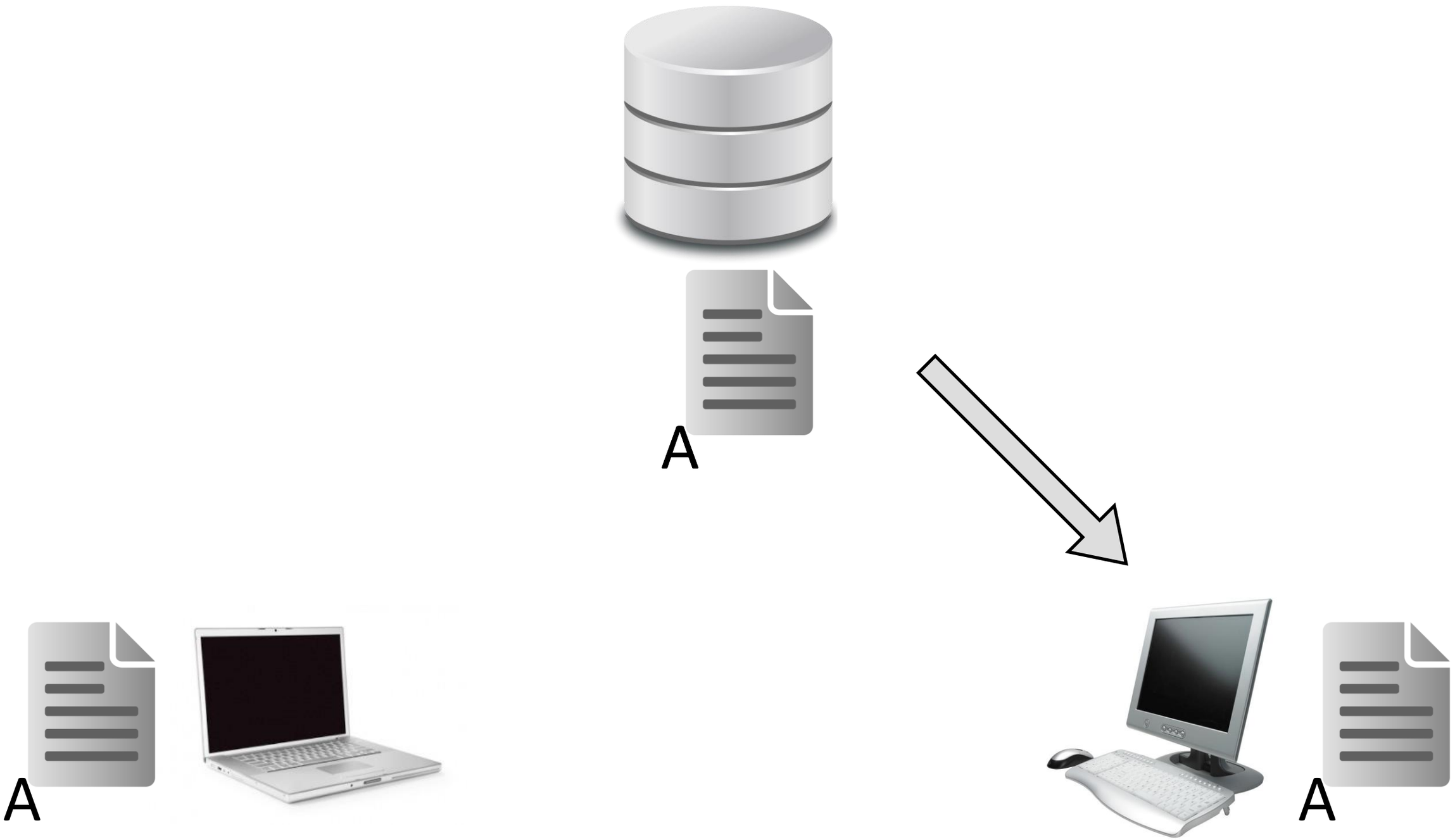


Système de gestion de versions

A



1. Lit A


A
A
A
2. Lit A

A

A’

A

3. Modifie A en A’

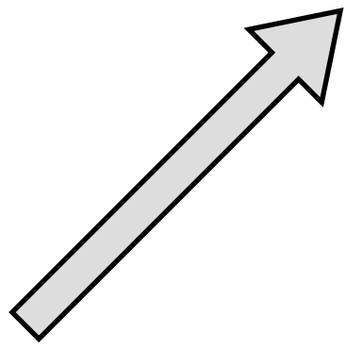
A

A’

A’’

4. Modifie A en A’’

A’

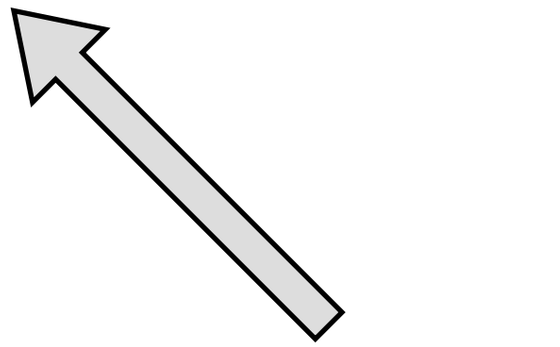


A’

5. Sauve A’

A’’

A’’



A’

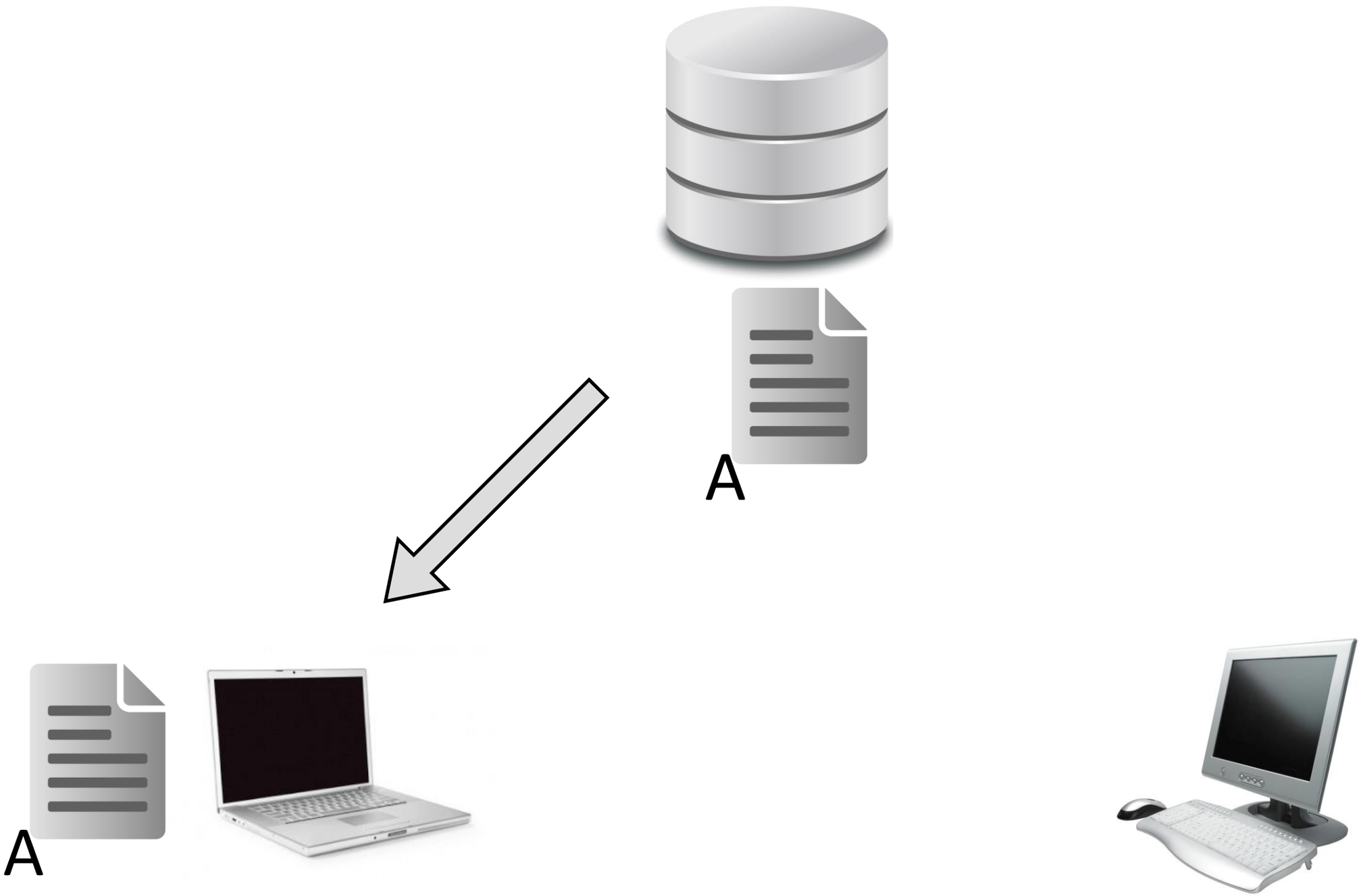
A’’

6. Sauve A’’

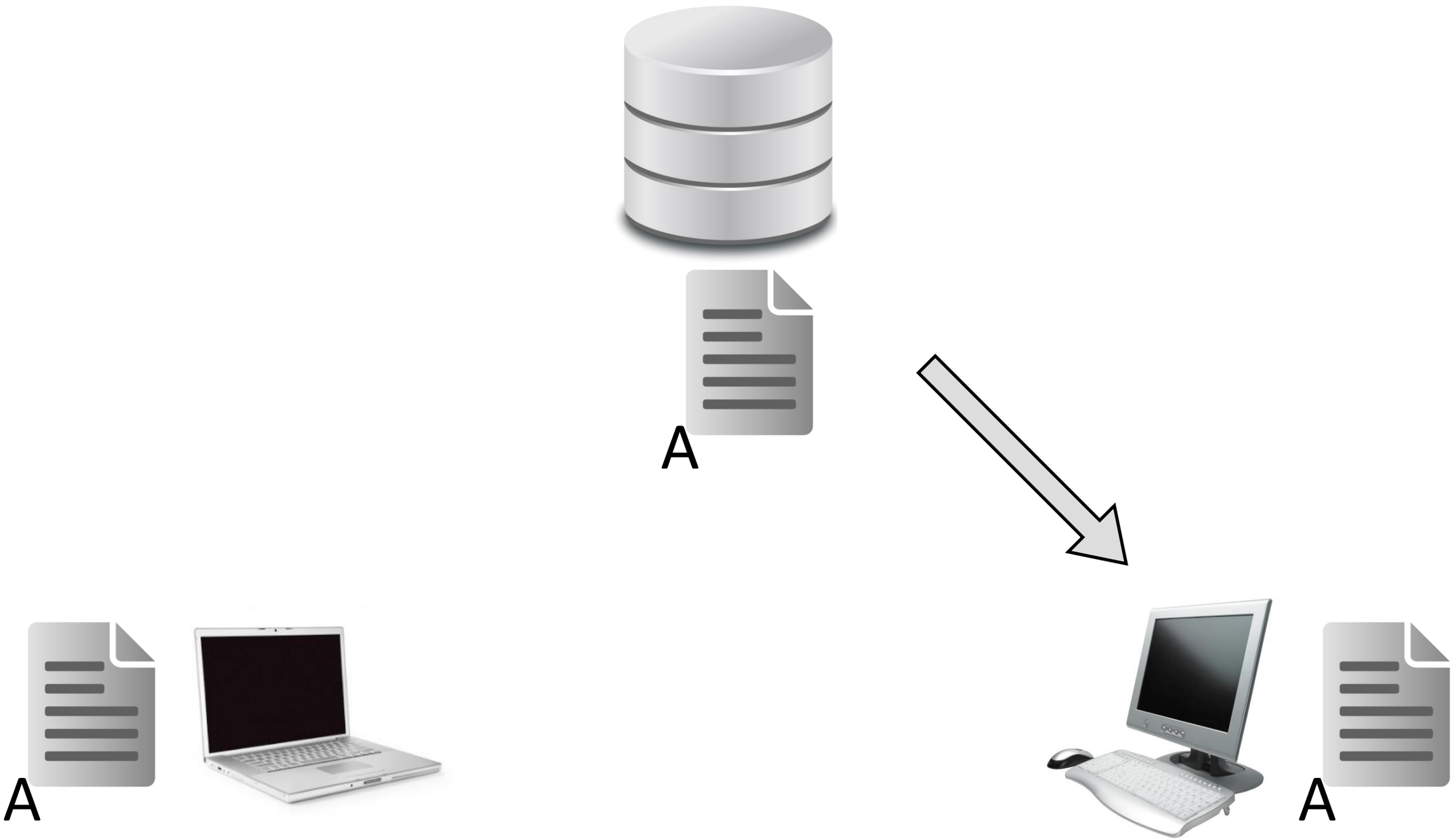

Conflit de version
A’
A’’

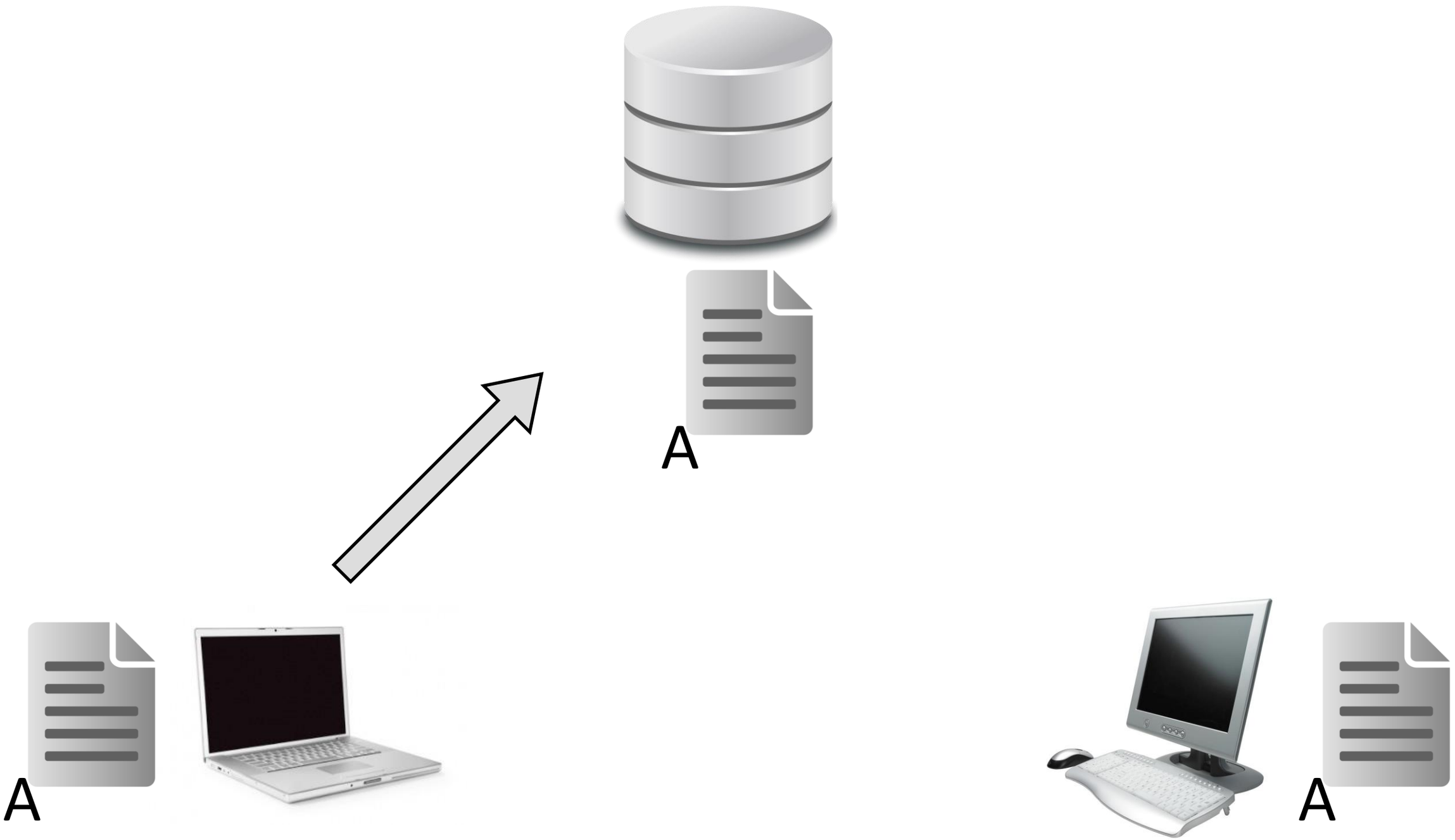
Solution 1: Protéger le fichier par un verrou

A



1. Lit A


A
A
A
2. Lit A

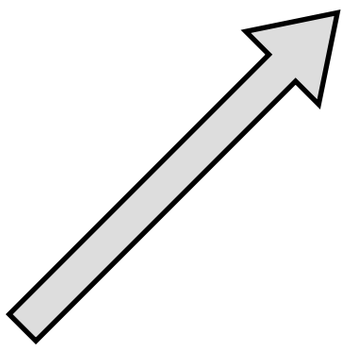


3. Verrouille A

3. Modifie A en A’

4. Verrouille A

A’

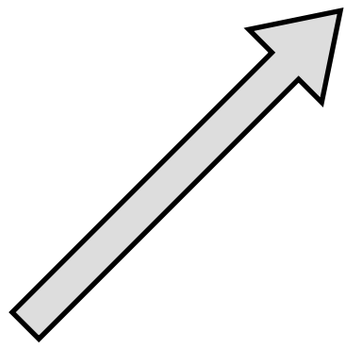


A’

A

5. Sauve A’

A’



A’

A

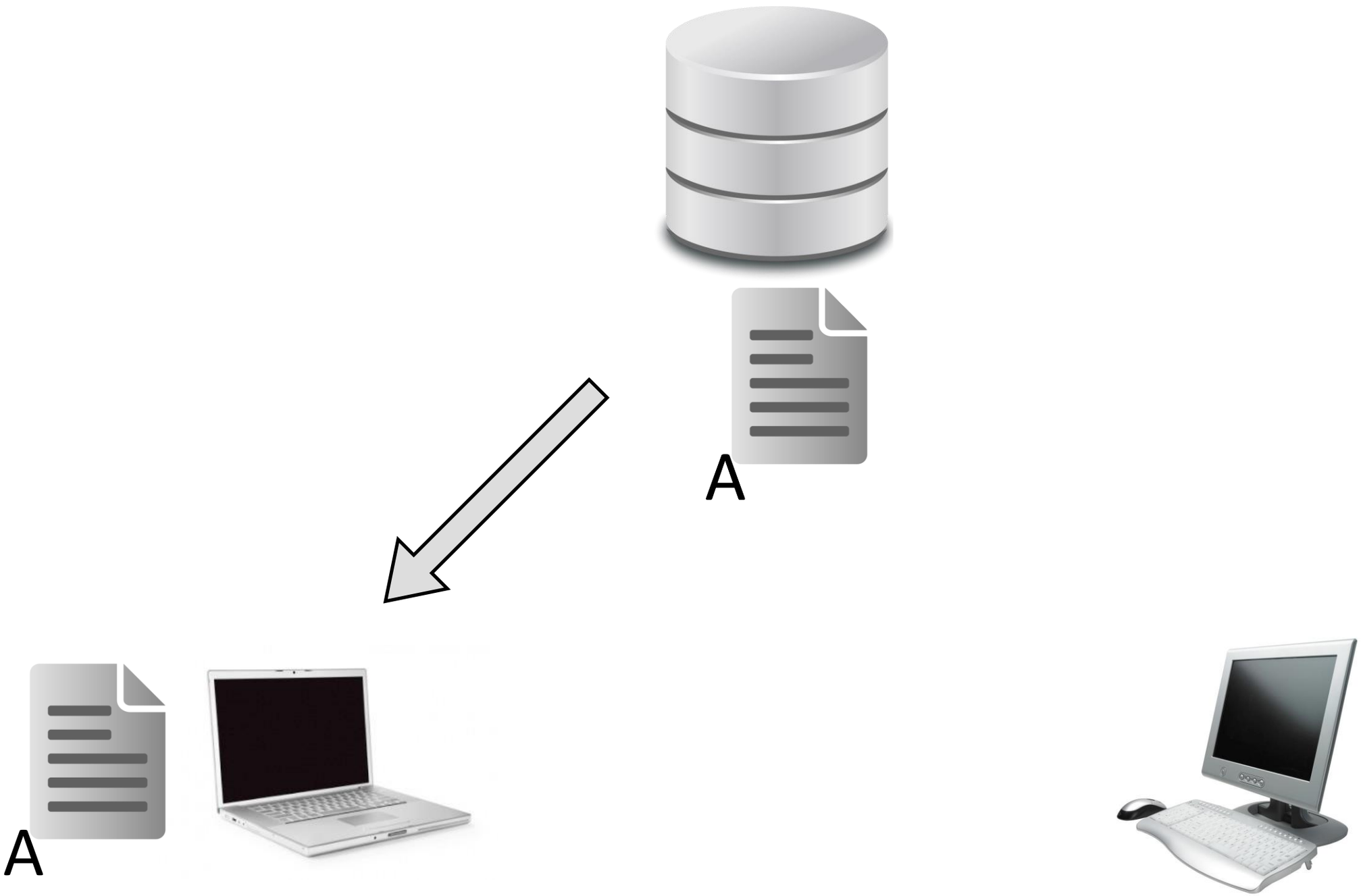
6. Déverouille A’

Verrou: Pas efficace pour travailler en groupe sur les même fichiers

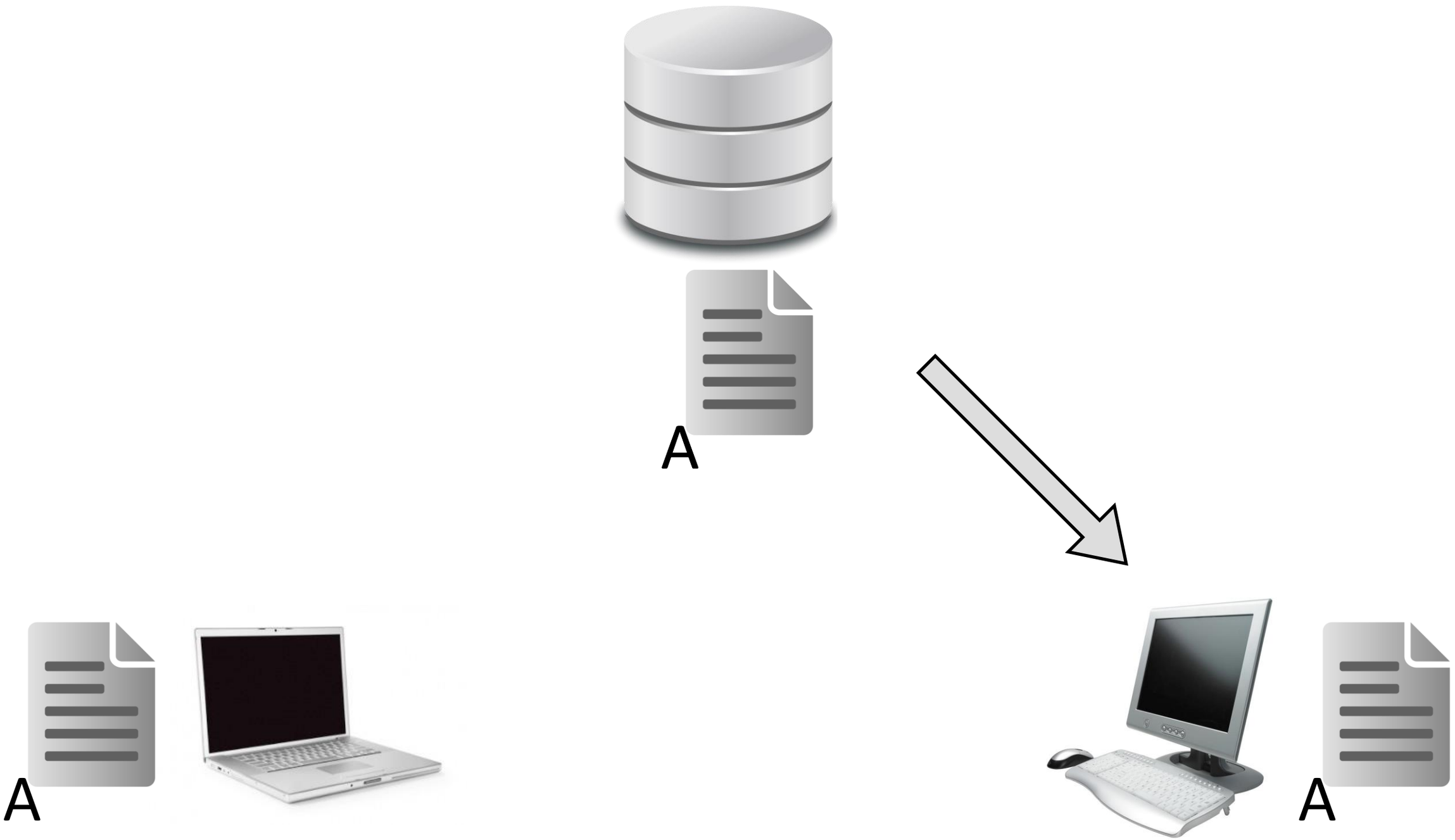
Solution 2:

Copier - Modifier - Fusionner

A



1. Lit A


A
A
2. Lit A

A

A’

A

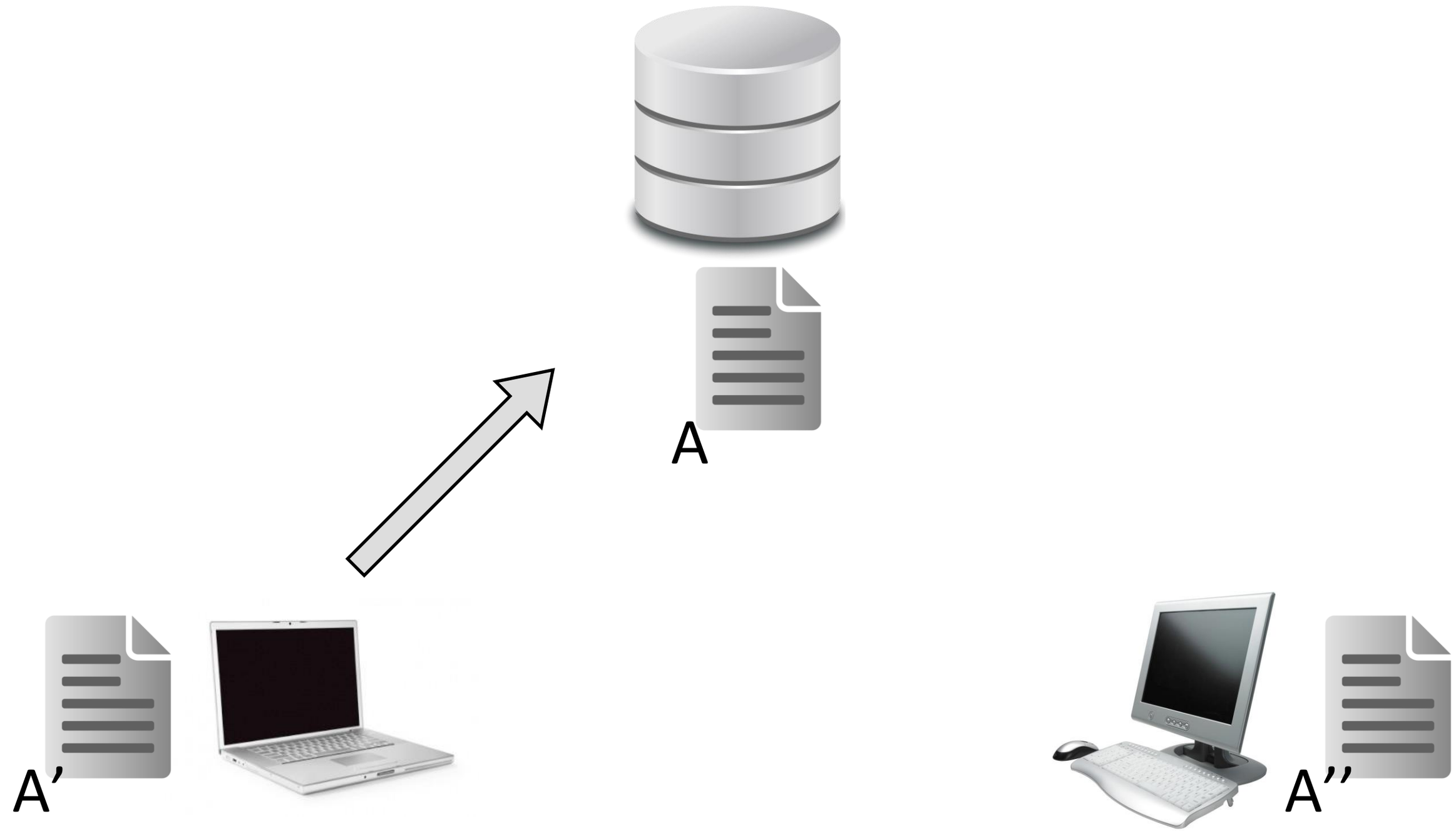
3. Modifie A en A’

A

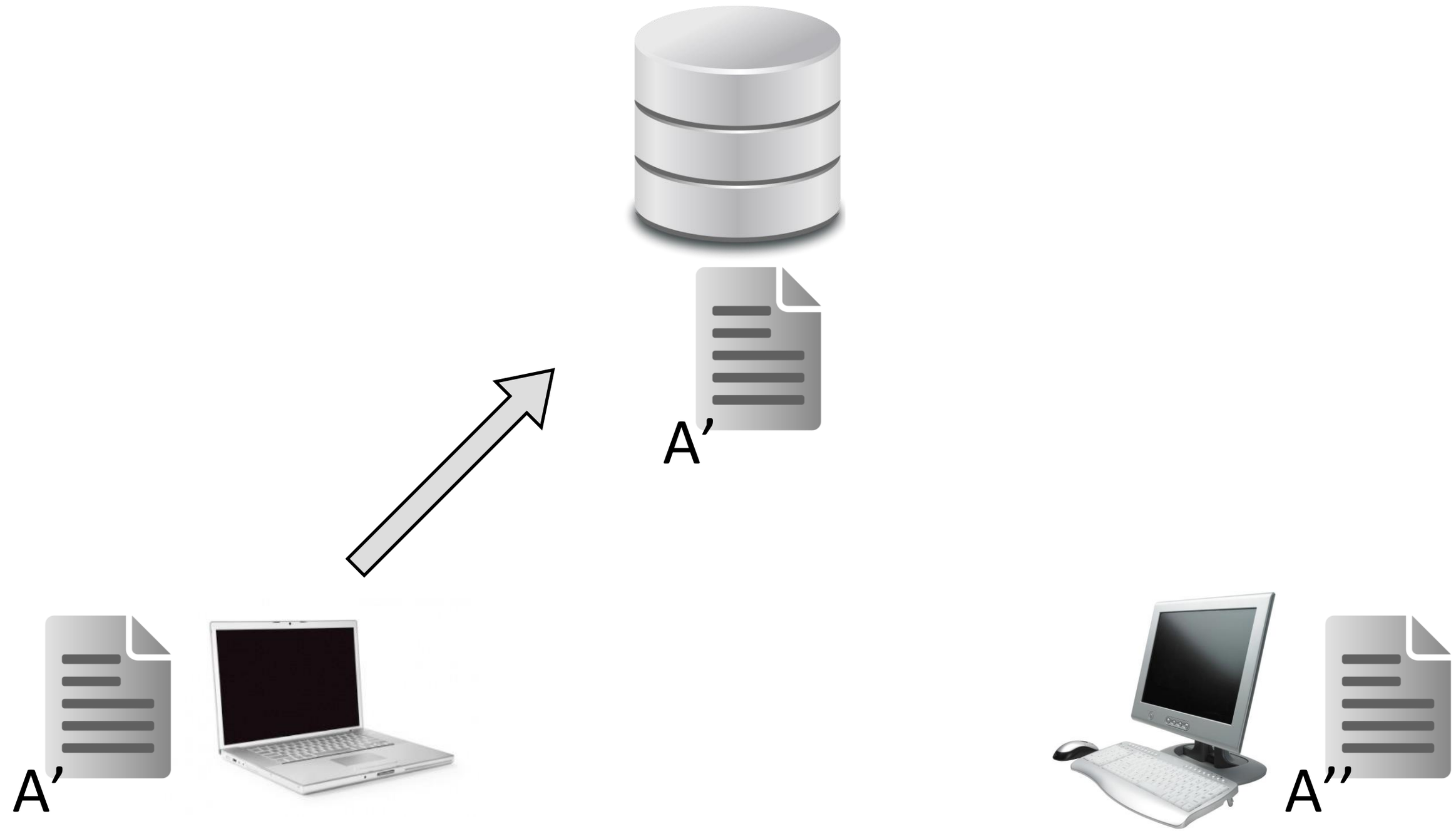
A’

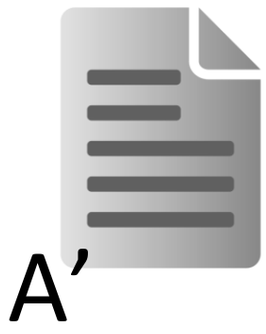
A’’

4. Modifie A en A’’

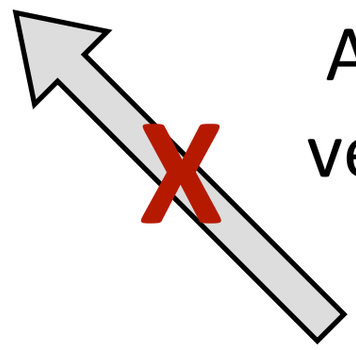


5. Sauve A’ à la place de A


A’
A’
A’’



A’



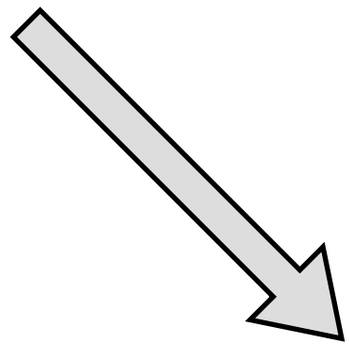
A n’est plus la version de base

A’

A’’

6. Sauve A’’ à la place de A

A’



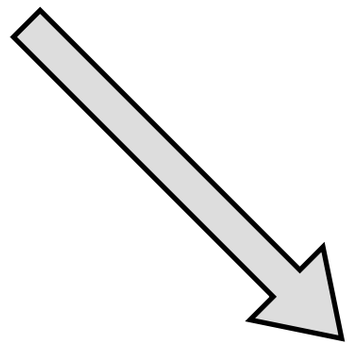
A’

A’’

A’

6. Lit la dernière version

A’

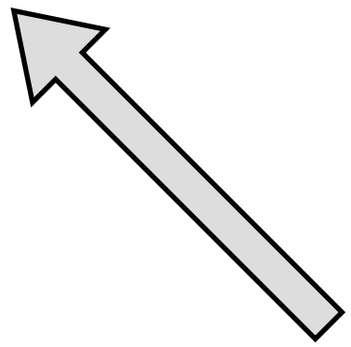


A’

A’’’

7. Fusionne A’ et A’’

A’’’



A’

A’’’

6. Sauve A’’’ à la place de A’

La solution utilisant copie-modifier-fusionner ne fonctionne efficacement que si les fichiers sont modularisés (méthode, classes) sinon il faut utiliser la technique du verrou

SCM

- Nombreux SCM (SVN, CVS, Git, Mercurial)
- Nous utiliserons Git.

Subversion

- Transactions atomiques
- Chaque transaction crée une nouvelle version (numéro de révision)
- Plus le n° de révision est grand, plus la version est récente
- Un numéro de révision est valable pour l'abre complet

Subversion

- Open source
- Administration facile
- Indépendant de la plateforme

Subversion
(vocabulaire)

- Le “repository” est l’emplacement central où sont stockés les sources et leurs historiques
- “working copy” est la copie locale sur laquelle le développeur travaille

Subversion
(vocabulaire)

- “checkout” ramène le projet pour la première fois. Le résultat est la copie de travail
- “import” est l’opération inverse de “checkout”
- “update” ramène la dernière version
- “commit” sauvegarde l’état sur le serveur

Subversion
(vocabulaire)

- Une “revision” est un numéro de version. Ce numéro augmente de 1 à chaque opération
- Un “tag” copy l’état du projet à un moment donné sous un nom particulier

Git

- Plus orienté development distribué que SVN
- Un peu plus compliqué de SVN
- Tant à remplacé SVN

Git vs SVN

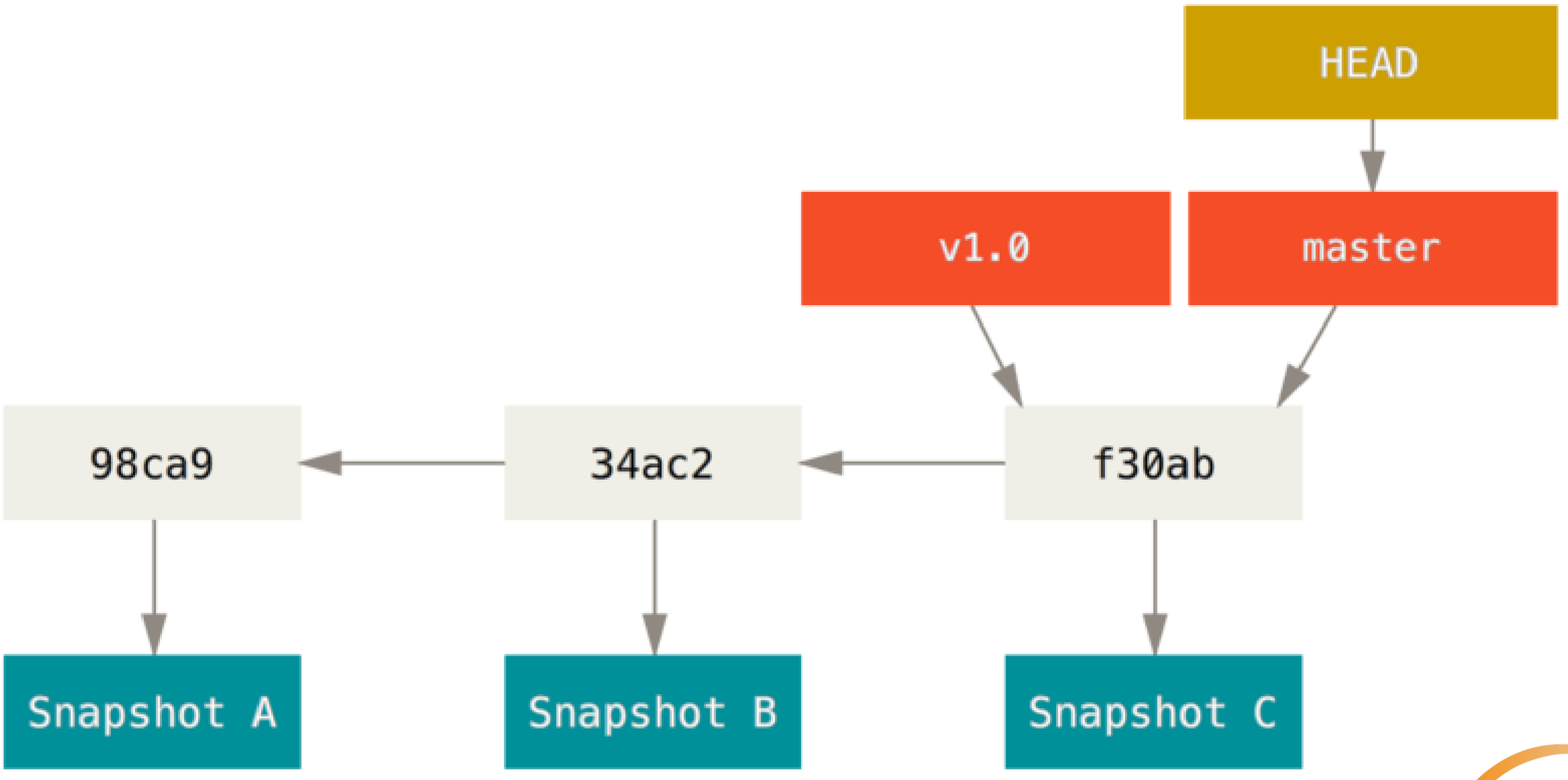
GIT

- Gets the whole repository
- Push online
- Revision is a hash code
- Several new features (e.g., Bisect)
- Faster than SVN

SVN

- Gets the current directory and sub directories
- Checkout/Commit/Diff online
- Revision is a number
- Cannot amend/cancel a commit
- Less complex than Git

Git


HEAD
v1.0
master
98ca9
34ac2
f30ab
Snapshot A
Snapshot B
Snapshot C

Git (Manuel de survie)

HEAD pointe vers le dernier commit de la branche courante (checkoutée)

master pointe, par convention, vers le dernier commit de la branche principale. HEAD = master quand on est sur la branche principale. Cela pointer vers un commit$

```
$cat .git/HEAD
ref: refs/heads/master
$ cat .git/refs/heads/master
4c461f529b169c34e49c17f20be7c4f9883dd4f8
```

origin est, par convention, le nom du repository distant

Git (Manuel de survie)

- Cloner un repository en local : `git clone https://....`
- Créer une nouvelle branche: `git checkout my_branch`
- Ramener les changements du remote: `git pull`
- Le statut actual: `git status`
- Rajouter tous les changements: `git add –all`
- Reprendre la version committé: `git checkout un_fichier`
- Créer un commit: `git commit –m “message de commit”`
- Pousser les changements: `git push`

Bonnes pratiques

- “Committer” plusieurs fois par jour (2-3x)
- “Committer” des changements cohérents
- “pusher” régulièrement
- Amender un commit si necessaire pour garder des changements cohérents

Bonnes pratiques

- Avant de committer, toujours executer les tests unitaires
- Pas de commit si:
  - le code ne compile pas
  - les tests unitaires ne sont pas verts

http://www.vogella.com/articles/Git/article.html#versionc
ontrolssystems