Séminaire SARI, le 8 avril 2008

SVN un gestionnaire de versions

Frédéric MELOT – LPSC / IN2P3 / CNRS (Grenoble)
melot@in2p3.fr

Objectifs de ce séminaire

- 1 Introduction – présentation de svn
- Côté utilisateur
  - 2 Description détaillée des mécanismes de base
  - 3 Survol des principales fonctionnalités (ce qu'il est possible de faire)
  - 4 Exemples concrets d'utilisation
- Côté administrateur
  - 5 Éléments d'installation d'un serveur
  - 6 Administration d'un serveur
- 7 Conclusion

PLAN

1 Introduction

- 1.1 Ce qu'est Subversion
- 1.2 Pourquoi utiliser Subversion ?
- 1.3 D'où vient Subversion ?
- 1.4 Améliorations par rapport à CVS

1.1 Ce qu'est Subversion (ou svn)

- Subversion est un système de gestion de versions
  Il gère les fichiers et les répertoires, ainsi que leurs modifications à travers le temps.

- Principale utilisation : gérer les modifications de code source mais également toute sorte d'information : documentation, images, fichiers de configuration, …
  Pour Subversion une données est une données quelle soit binaire (illisible humainement) ou non

- Logiciel gratuit et 'open source'

- Fonctionne sous Linux, Windows, Mac OS, …

- Concept similaire pour les autres logiciels de gestion de versions (RCS, clear case, VSS, CVS, …)

1.2 Pourquoi utiliser Subversion ?

- **Permet de conserver l'historique d'une hiérarchie de fichiers :**
  - Garde une trace de toute les modifications (qui, où, pourquoi)
  - L'ancien code reste disponible => récupération des versions anciennes de vos données et retour en arrière possible
  - Être à l'abri des fausses manœuvres
  - On peut examiner l'historique des modifications (ex : consulter les différences entre révisions de fichiers)

- **Permet de développer du code de façon collaborative par plusieurs personnes, équipes, ...**

  car Subversion peut agir à travers le réseau et ainsi permettre son utilisation à différentes échelles : un service, une laboratoire, une collaboration nationale ou internationale
  - Les modifications effectuées par une personne sont accessibles à tous
  - Un ou plusieurs développeur peut modifier un même fichier (les modifications d'un fichier peuvent provenir de plusieurs sources)

1.2 Pourquoi utiliser Subversion ?

- Peut servir de système de sauvegarde
- remplace très avantageusement les 'zip' :
  - classement plus simple
  - Ne sauvegarde que les différences entre révisions => gain de place
  - Permet la gestion globale de l'historique de projet et permet l'interrogation
- remplace très avantageusement les échanges de versions par mail
- Utile aussi bien pour les collaborations que pour les individus
- Subversion n'est pas un outil de gestion de configuration !
  Ces systèmes sont spécialement conçus pour gérer du code source et ont donc des fonctionnalités propres au développement d'applications
  Ex : CMT (http://www.cmtsite.org/), tag collector, …

1.3 D'où vient Subversion ?

- Dans le monde du logiciel libre, CVS (Concurrent Versions System) a été la référence pendant des années
- Corriger ses problèmes était très compliqué : conception et modèle de développement à revoir
- Début 2000 la société CollabNet (http://www.collab.net) a décidé d'écrire un nouveau système de gestion de versions :
  - En ré écrivant tout
  - En utilisant les idées de base de CVS mais en éliminant ses bugs et en comblant ses lacunes
  - Assez similaire à CVS pour permettre une transition aisée
- Mi 2001 première version
- Développé pour être le successeur de CVS, SVN a deux ambitions :
  - Être un système 'free' et 'open source' au design similaire à CVS
  - Éviter la plupart de ses défauts
- Maintenant presque tous les nouveaux utilisateurs de logiciels de gestion de Versions choisissent SVN et non CVS

1.4 Améliorations par rapport à CVS

- **Gestion des versions des répertoires**
- **Historique 'total' des versions**
  Copies, renommage ou déplacement de fichiers non supportées par CVS.
- **Les commits sont atomiques**
  Un ensemble de modifications sera soit entièrement validé dans le repository soit pas du tout.
- **Gestion des métadata**
  Il est possible de définir pour chaque fichier et répertoire un l'ensemble de propriétés voulues, gérées à travers le temps.
- **Choix des couches de communication**
  - peut s'intégrer à un serveur Web apache
  - possède également un serveur indépendant, avec son propre protocole
  - a une notion abstraite d'accès au repository, ce qui permet d'en développer facilement de nouveaux
- **Gestion des fichiers binaires**
  Subversion exprime les différences de contenu de version les fichiers à l'aide d'un algorithme spécifique, qui fonctionne de la même façon pour les fichiers texte que pour les fichiers binaires. Ils sont stockés avec la même compression dans le repository.
- **Branches et tags revisités (et plus efficaces)**
  Le coût des opérations de branches et de tag ne sont plus proportionnels à la taille du projet. Avec Subversion la création de branche et de tag s'effectuent à l'aide d'une simple copie du projet, ce qui revient à créer sur le repository un lien symbolique.
  => temps de l'opération faible et constant et plus de sticky tags !

PLAN

2 Mécanismes de base

- 2.1 Architecture (repository, working copy et communication)
- 2.2 Principe de fonctionnement
- 2.3 Différents modèles de systèmes de gestion de versions
  - La solution lock – modify - unlock
  - La solution copy – modify - merge
- 2.4 Accès au repository
- 2.5 Précisions sur la working copy
- 2.6 Révisions
- 2.7 Commandes SVN - généralités

2.1 Architecture

L'installation de Subversion permet l'utilisation de plusieurs programmes :

- svn : le programme client en ligne de commande
- svnversion : un programme permettant de connaître l'état d'une working copy
- svnlook : un outil d'interrogation directe du repository
- svnadmin : un outil permettant de créer, de paramétrer et de réparer le repository
- svndumpfilter : un programme de filtrage des dumps de repository svn
- mod_dav_svn : un plug-in pour le serveur apache, afin de rendre votre repository disponible sur le réseau
- svnserve : un serveur svn, lançable comme un démon ou invocable par ssh
- svnsync : un programme de recopie incrémentale de repository

2.1 Architecture

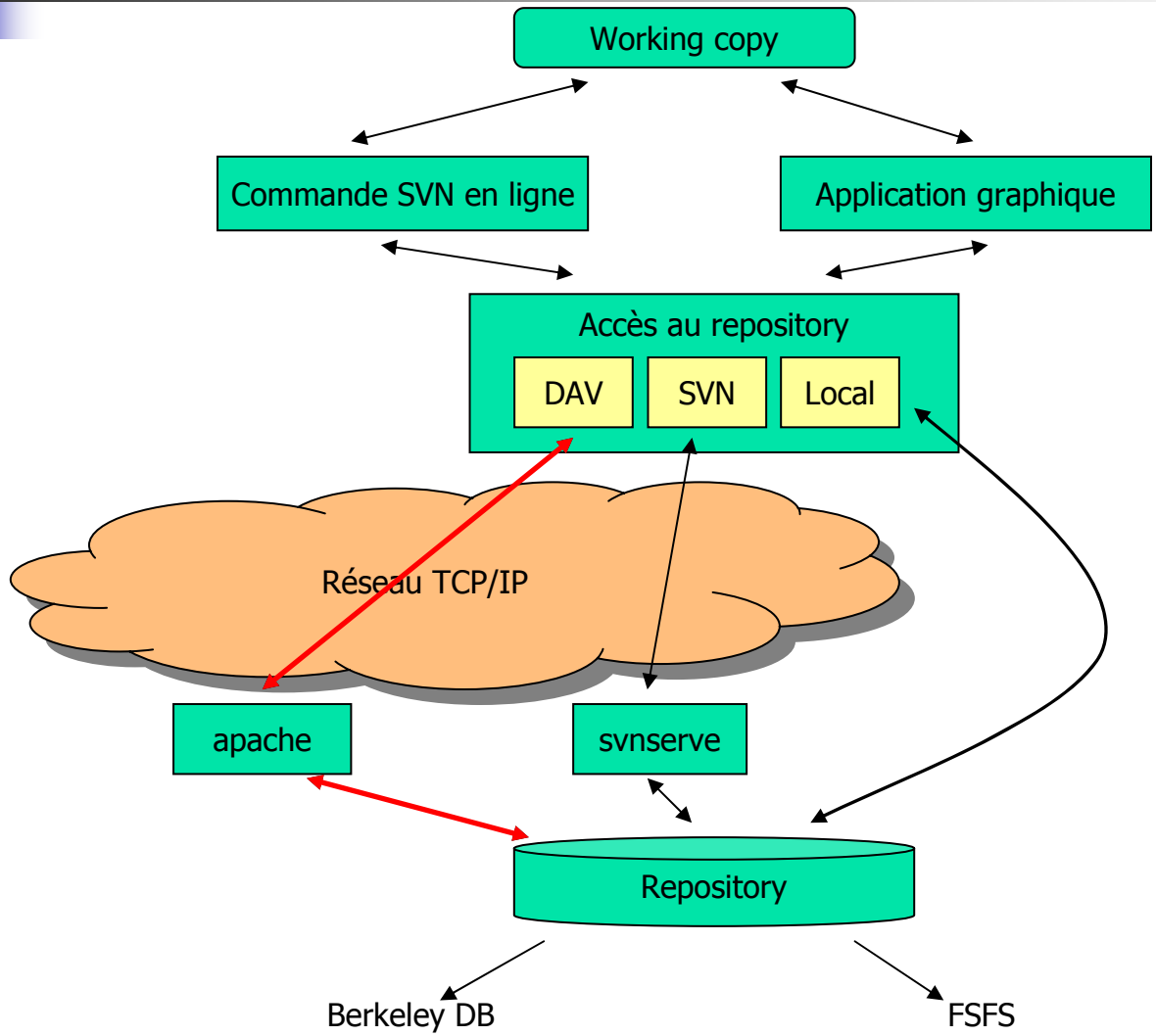
- Subversion est un système de gestion de versions de type “centralisé”. Son système central de données de référence s’appelle le repository
  Il contient toute la hiérarchie de fichiers et tout l’historique des modifications

- De l’autre côte se trouve votre client subversion qui gère une version locale d’une partie de ces données, la “working copy”
  - Tout ou partie de la hiérarchie
  - Une seule version à un instant donné

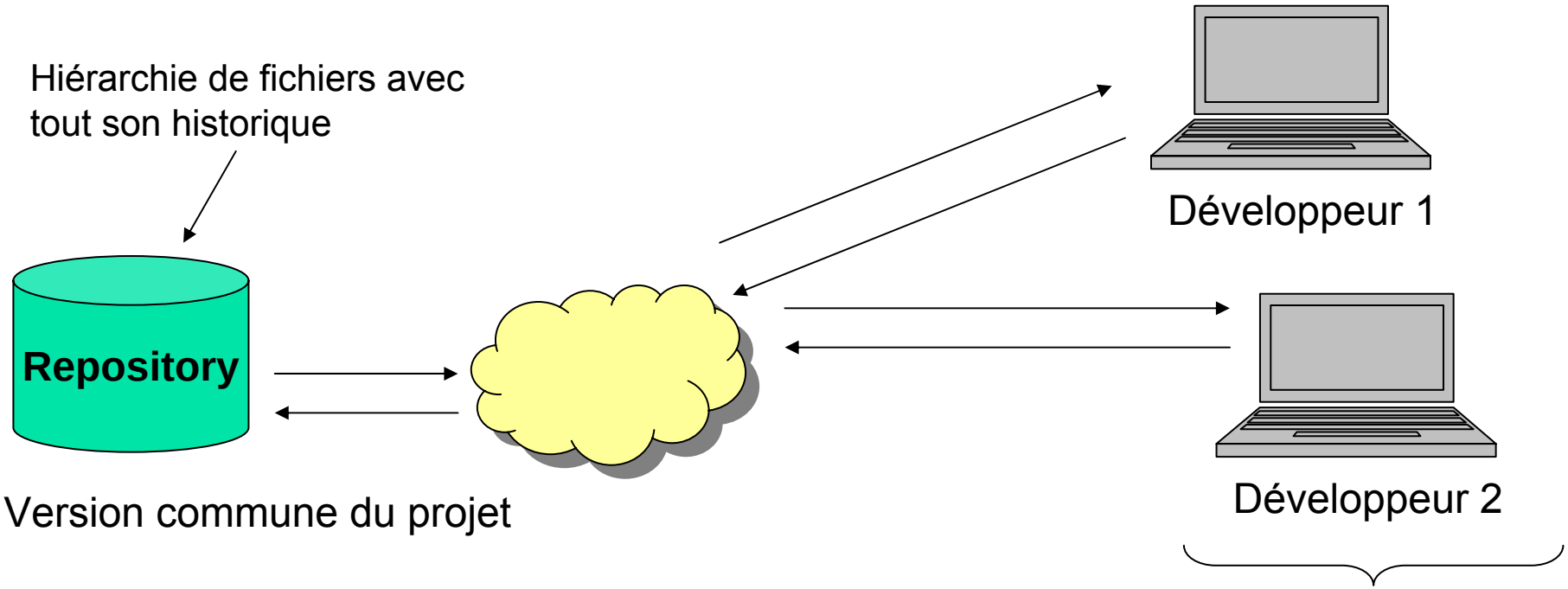
- Entre les deux, de nombreuses possibilités de communication existent à travers différentes couches d’accès. Certaines empreinte le réseau et à travers de servers accèdent au repository alors que d’autre ne se servent pas du réseau et accèdent directement au repository.

- Quand les clients se connectent au repository :
  - En écrivant, ils rendent leurs modifications disponibles pour les autres
  - En lisant, ils reçoivent les informations des autres
  - Ils peuvent aussi récupérer les états anciens du système de fichiers
  - Ils peuvent questionner le repository (ex : que contenait le repository lundi dernier ? , quels fichiers ont été modifiés depuis 2 mois ? , qui a effectuer les dernières modifications dans le repository pour un fichier donné ?, …)

2.1 Architecture



2.2 Principe de fonctionnement

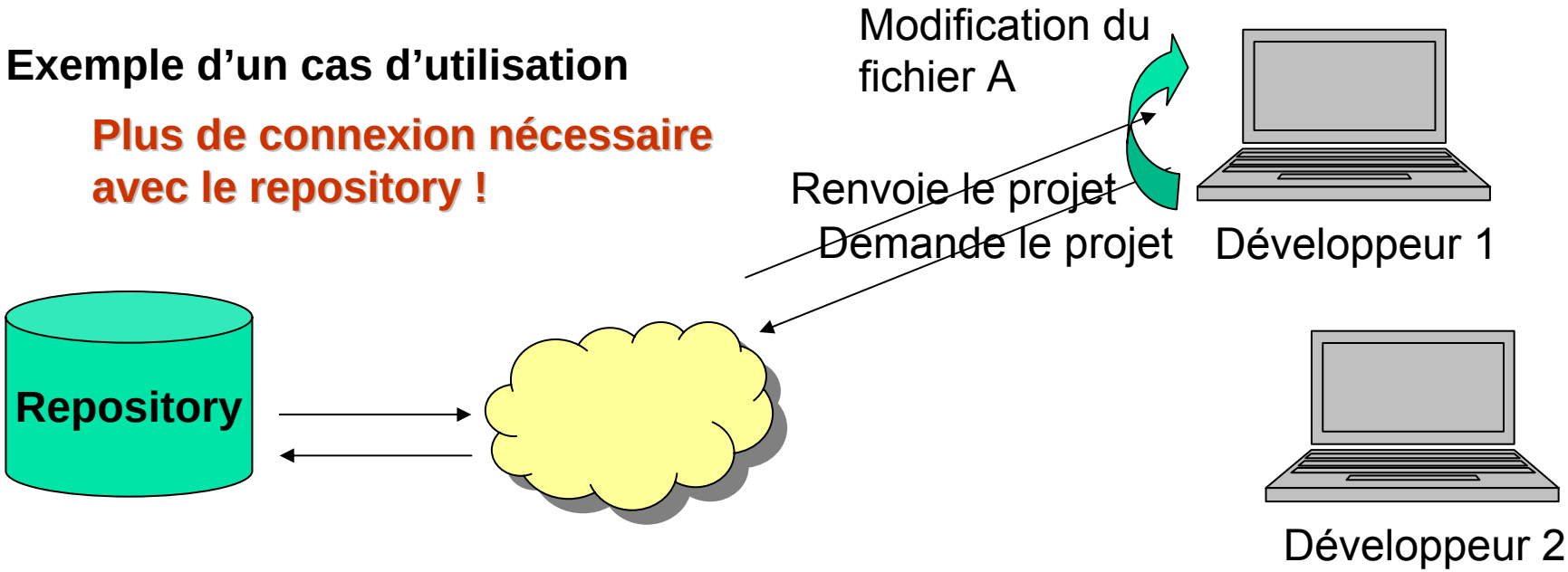


A un instant donné, chaque personne possède une version et une seule de chaque fichier

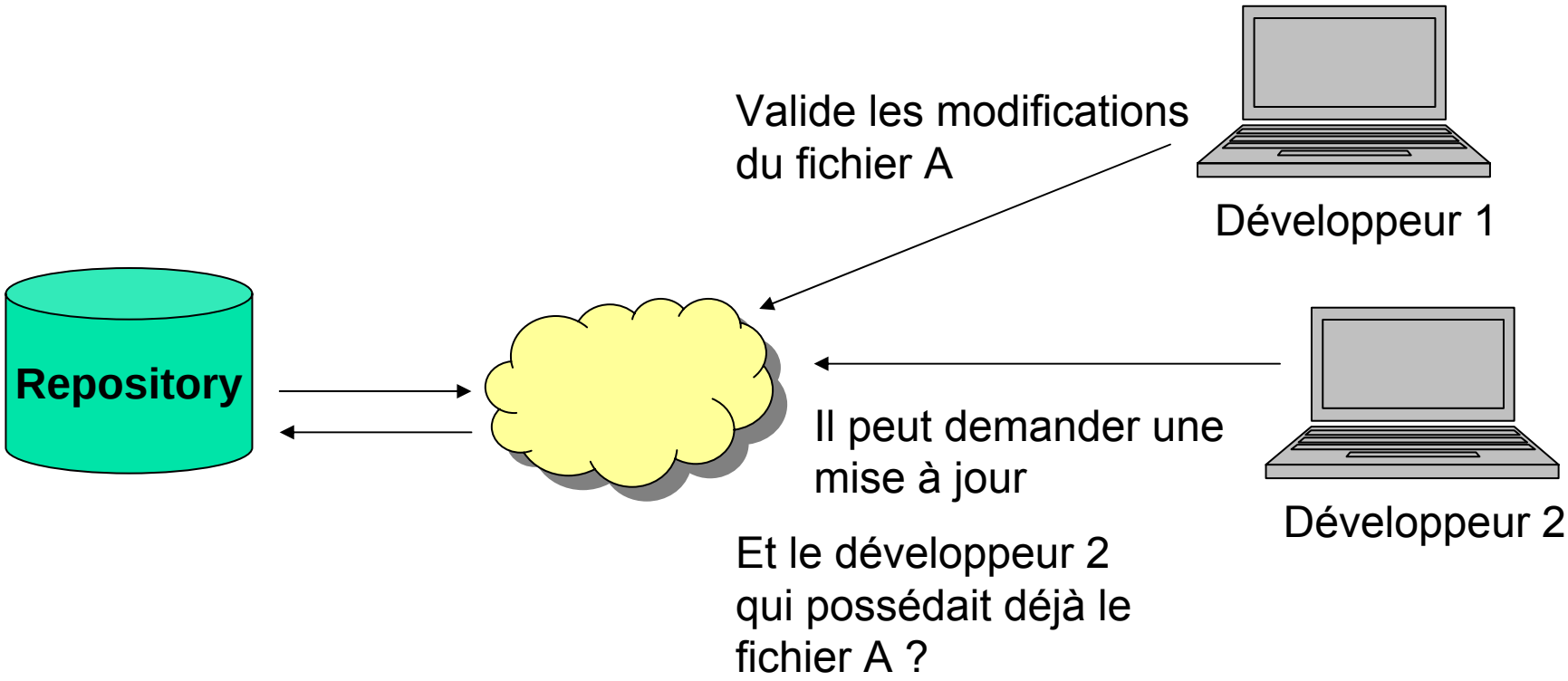
2.2 Principe de fonctionnement

Exemple d'un cas d'utilisation

Plus de connexion nécessaire avec le repository !



2.2 Principe de fonctionnement

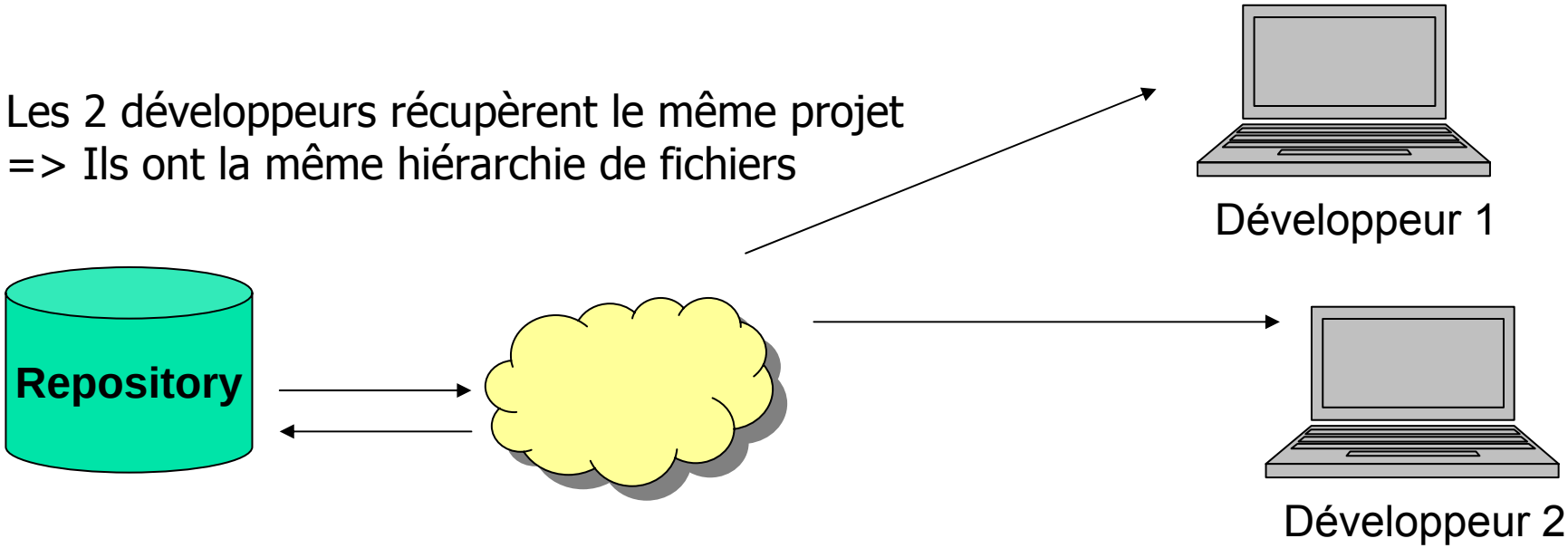


2.3 Différents modèles de systèmes de gestion de versions

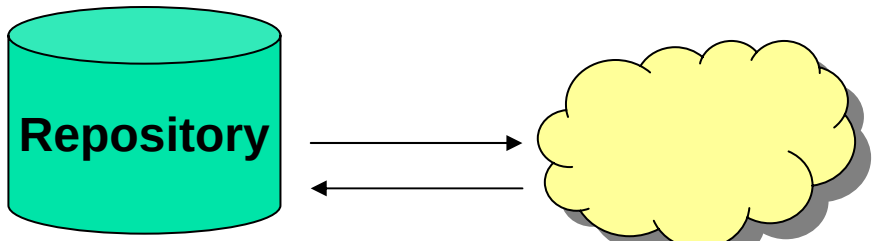
- Un système de gestion de versions permet :
  - L'édition collaborative
  - Le partage d'information
- Tous les systèmes de gestion de versions ont le problème fondamental : permettre le partage d'informations sans que personne n'écrase les données des autres
- A ce problème deux solutions :
  - La solution lock – modify – unlock
  - La solution copy – modify - merge

2.3 Différents modèles de systèmes de gestion de versions

Les 2 développeurs récupèrent le même projet
=> Ils ont la même hiérarchie de fichiers



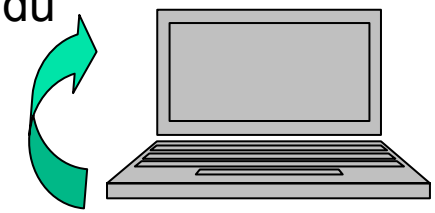
2.3 Différents modèles de systèmes de gestion de versions



Fichier A

Modification du fichier A

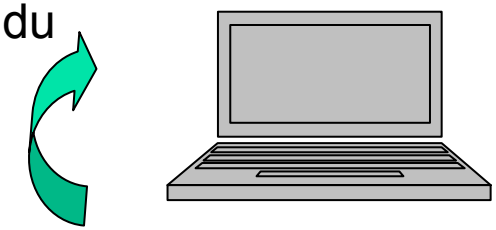
=> A'



Développeur 1

Modification du fichier A

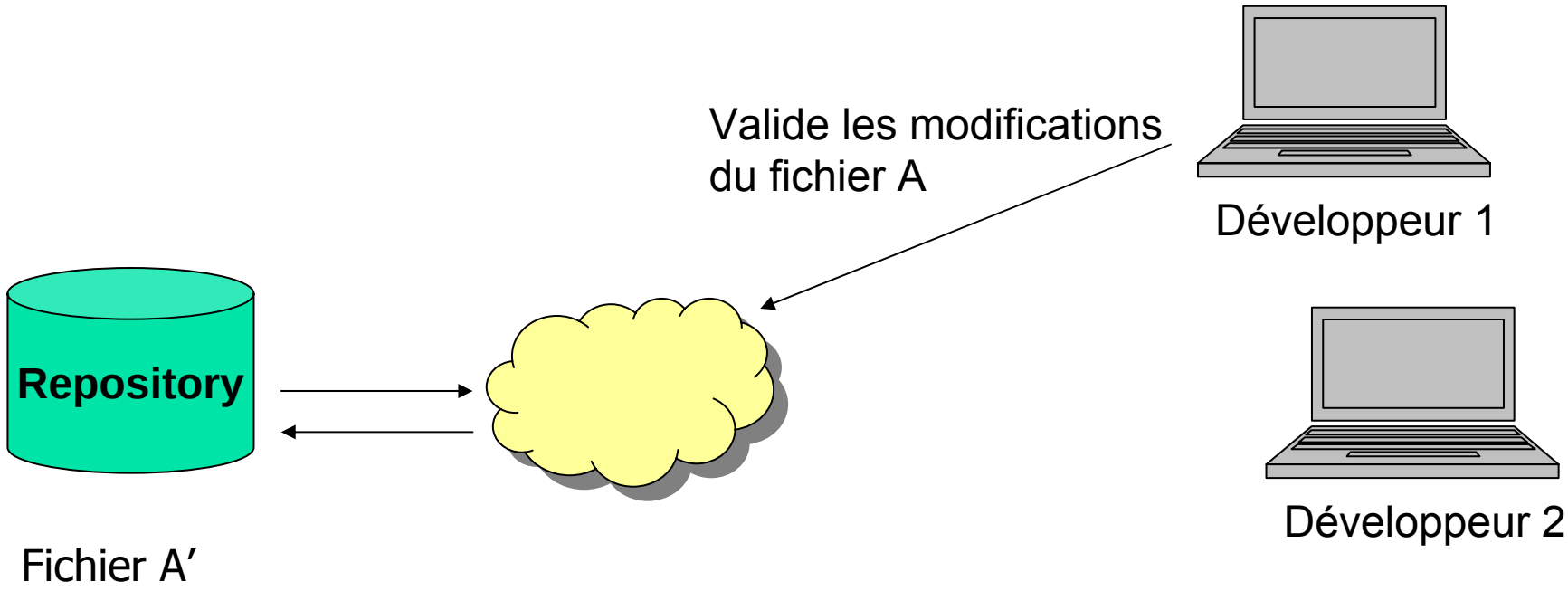
=> A''



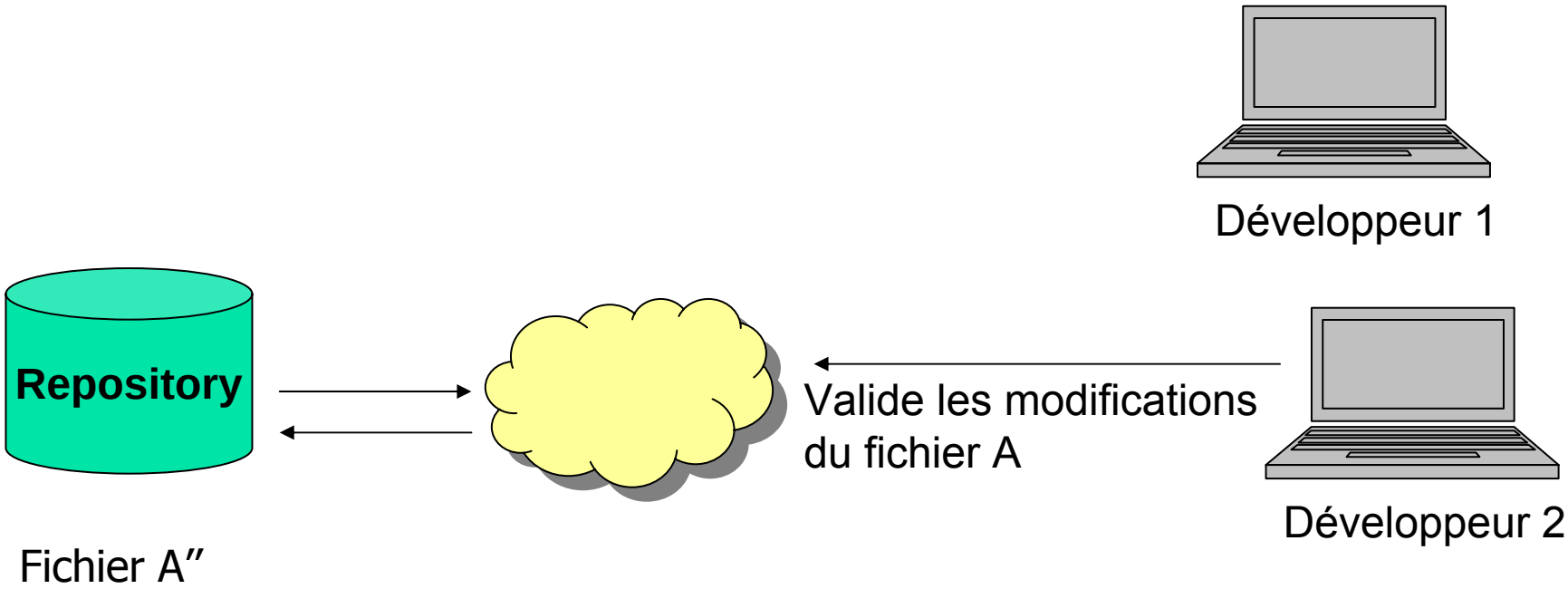
Développeur 2

Les fichiers A' et A'' sont incompatibles

2.3 Différents modèles de systèmes de gestion de versions



2.3 Différents modèles de systèmes de gestion de versions

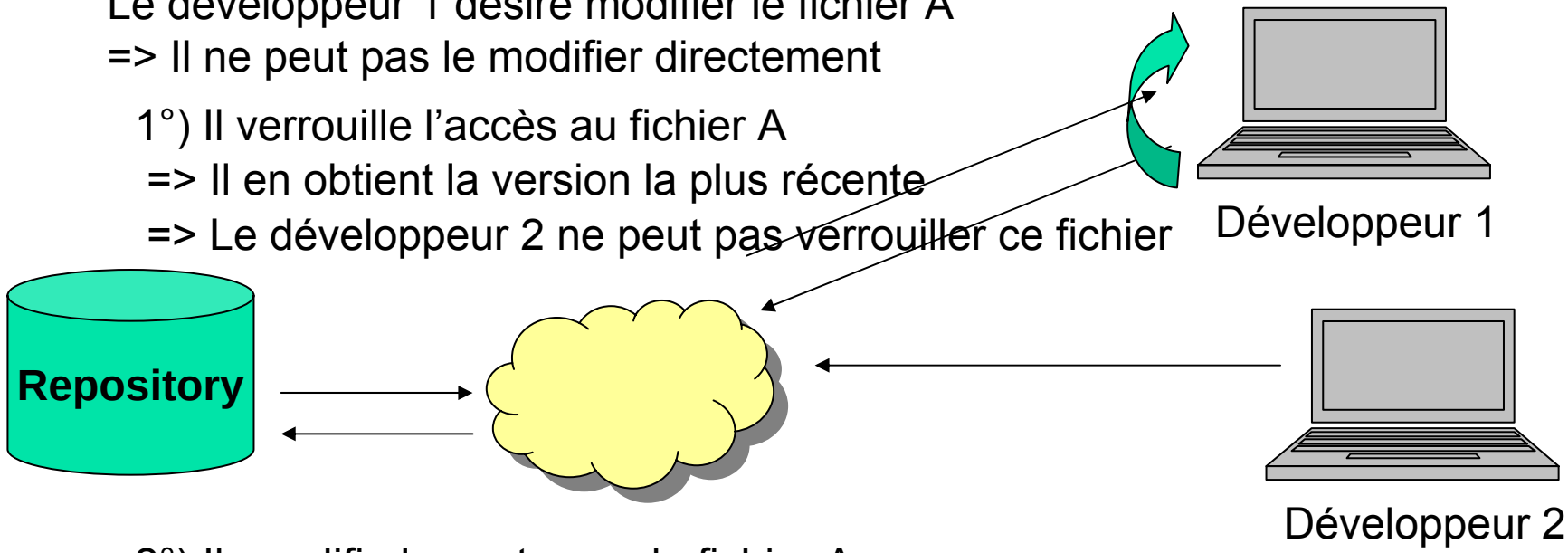


Si aucun système de vérification n’existe le développeur 2 écrase les modifications du développeur 1 !

2.3 La solution lock – modify – unlock

Le développeur 1 désire modifier le fichier A
=> Il ne peut pas le modifier directement

1°) Il verrouille l'accès au fichier A
=> Il en obtient la version la plus récente
=> Le développeur 2 ne peut pas verrouiller ce fichier



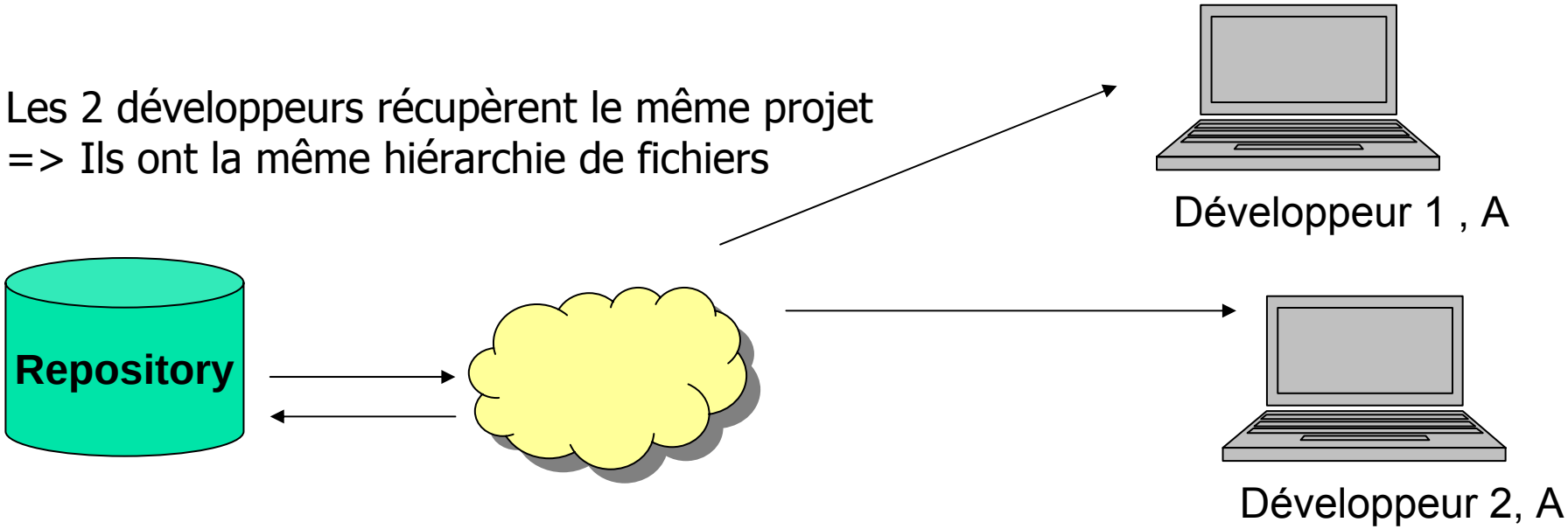
2°) Il modifie le contenue du fichier A
3°) Il valide ses modifications
4°) Il déverrouille l'accès au fichier A
=> le développeur 2 peut verrouiller ce fichier et ainsi récupérer sa version la plus récente

2.3 La solution lock – modify – unlock

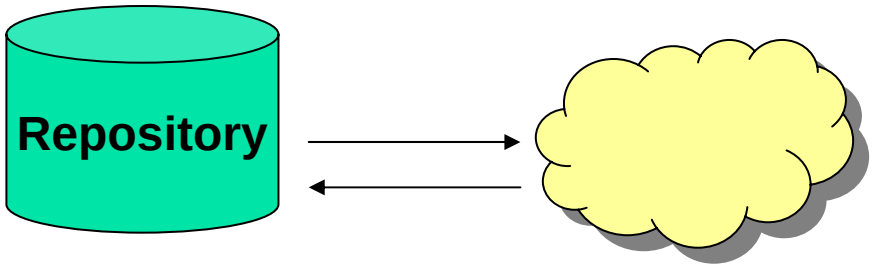
- Solution la plus restrictive : une seule personne à la fois peut modifier un fichier
- Peut être bloquant
  - On doit attendre, peut être pour rien
  - Si quelqu'un oublie de déverrouiller …
- Donne une fausse sensation de sécurité
  - Deux personnes peuvent effectuer simultanément des modifications incompatibles sur des fichiers différents
  - Rien n'empêche un utilisateur de supprimer les modifications d'un autre
  - => Ce n'est pas le système de gestion de versions qui remplace la communication

2.3 La solution copy – modify - merge

Les 2 développeurs récupèrent le même projet
=> Ils ont la même hiérarchie de fichiers



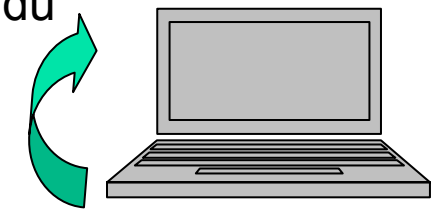
2.3 La solution copy – modify - merge



Fichier A

Modification du fichier A

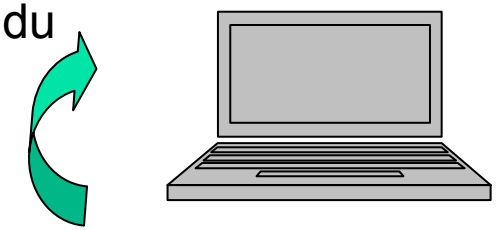
=> A'



Développeur 1 , A'

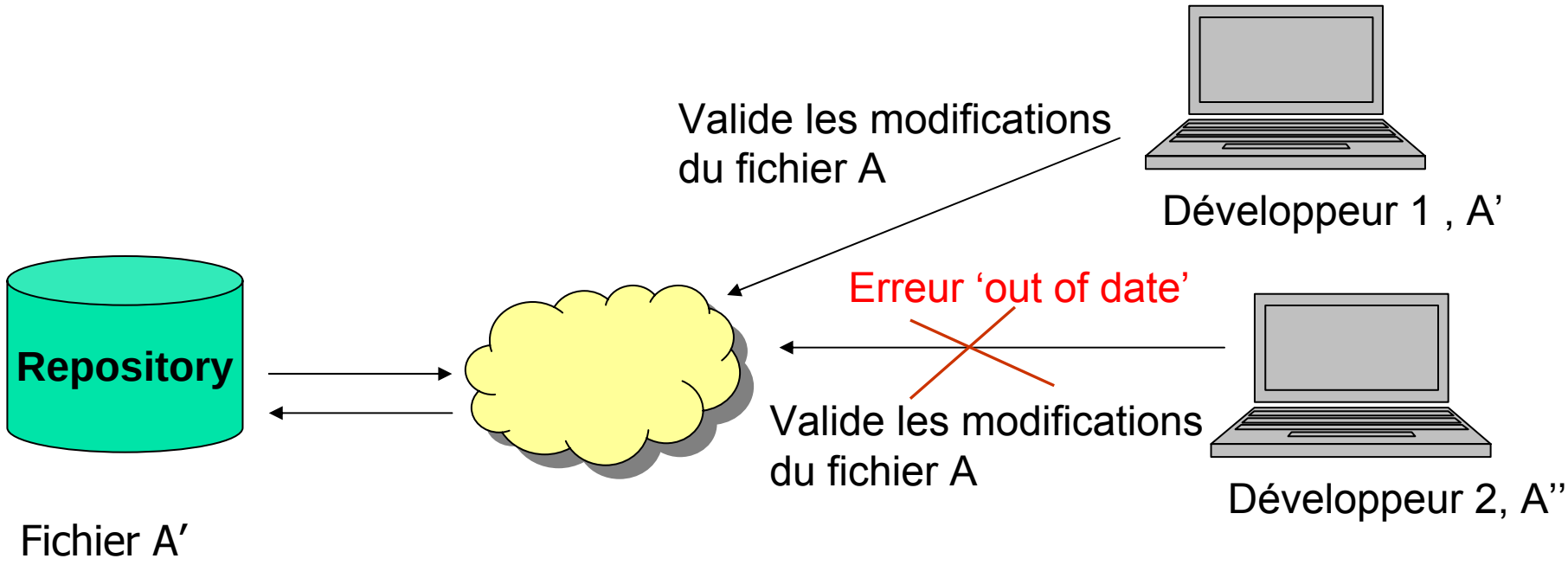
Modification du fichier A

=> A''



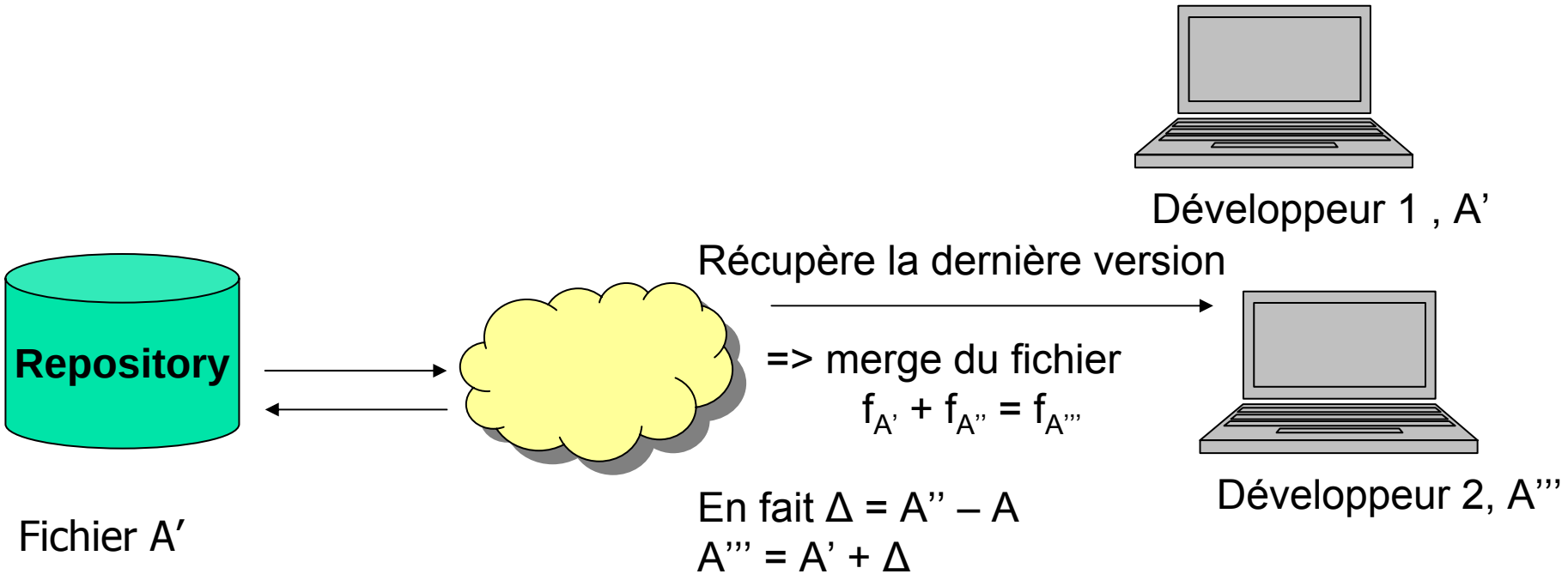
Développeur 2, A''

2.3 La solution copy – modify - merge



La version la plus récente du repository a changé depuis la création de la working copy du développeur 2

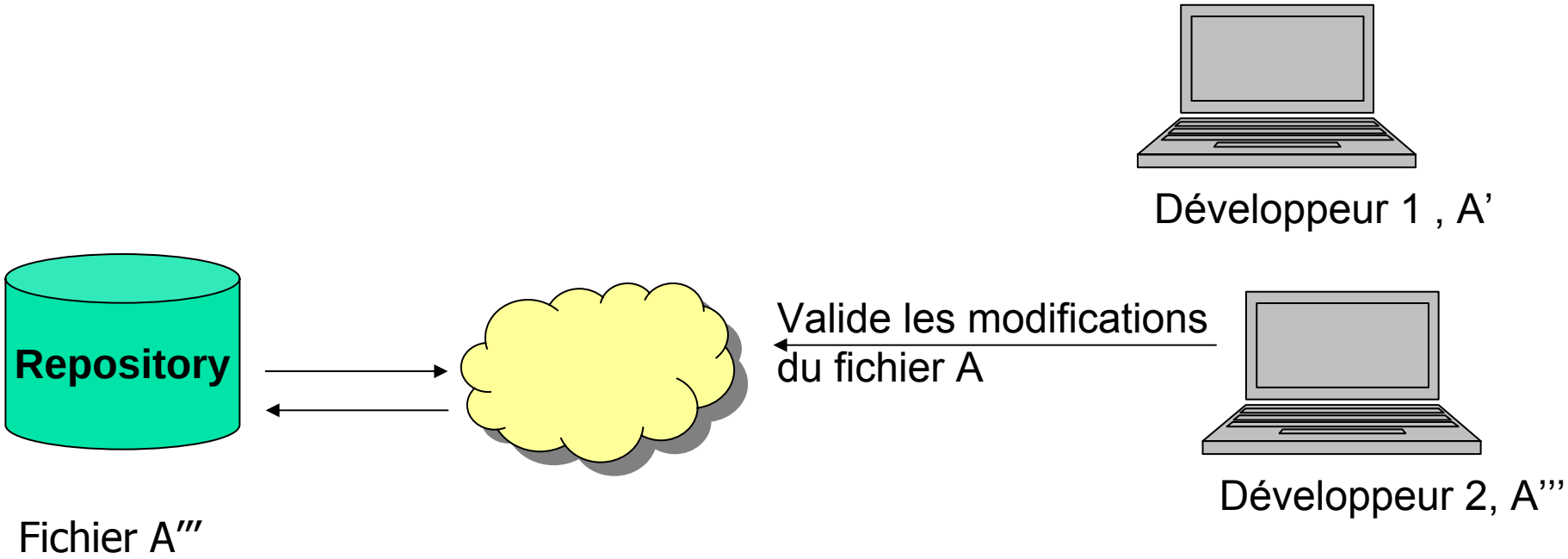
2.3 La solution copy – modify - merge



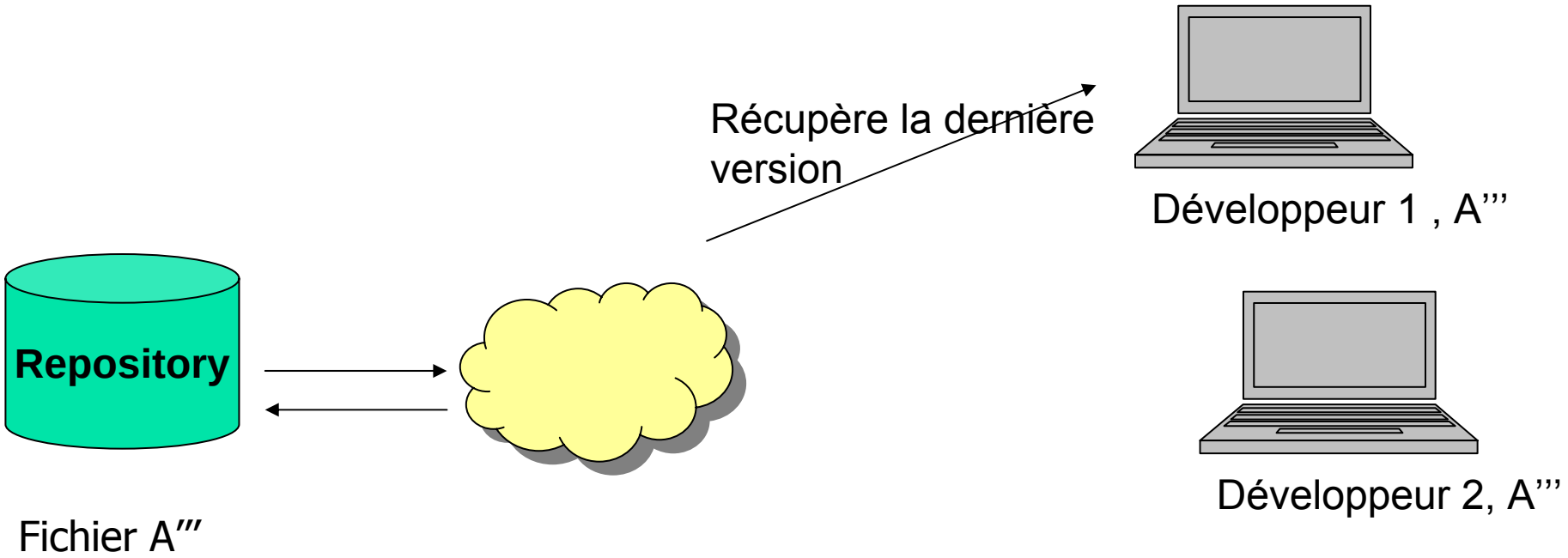
2 cas :

- Modifications compatibles => merge automatique
- Modifications concurrentes => situation de conflit
  => merge manuel

2.3 La solution copy – modify - merge



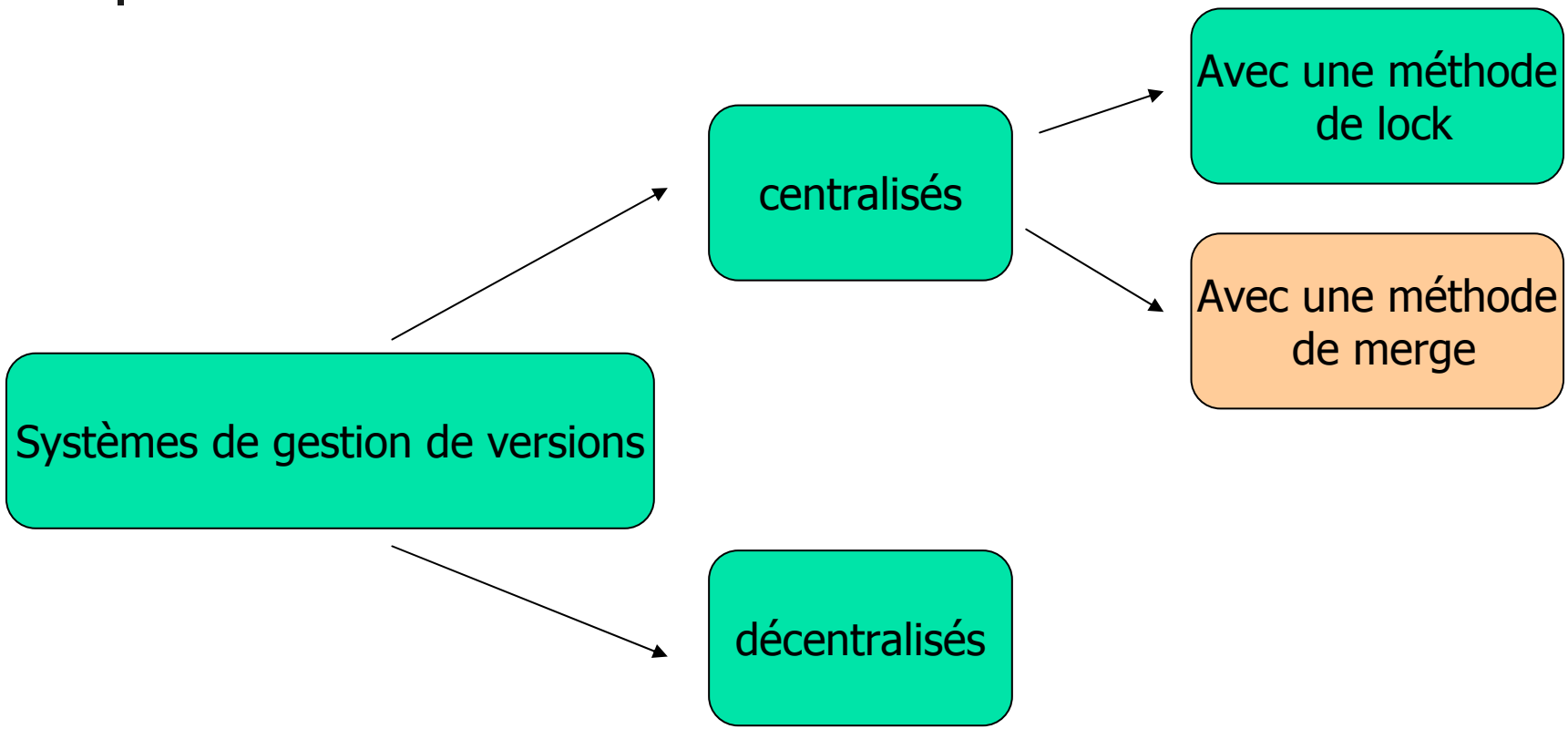
2.3 La solution copy – modify - merge



2.3 La solution copy – modify - merge

- Solution proposée par CVS, Subversion, …
- Chacun a sa propre working copy et peut y travailler indépendamment et simultanément. A la fin les versions sont mergées ensemble dans une version finale (de façon automatique ou manuelle)
- Peut sembler chaotique mais fonctionne extrêmement bien :
  - Pas besoin d'attendre
  - La plupart des modifications sur un même fichier ne donne pas lieues à un conflit
  - De plus résoudre un conflit est souvent plus rapide qu'attendre un déverrouillage
- Les conflits peuvent être évités en améliorant la communication entre utilisateurs
- Mais ne convient pas aux fichiers non 'mergeable' => mécanisme de lock toujours disponible

2.3 Différents modèles de systèmes de gestion de versions



2.4 Accès au repository

- SVN utilise des URL pour identifier les fichiers et les répertoires gérés dans le repository
  - Ex : https://lpsc.in2p3.fr:80/svn/test/web/index.html
  - Ex : file:///Z:/repository/SVN/test
  - Ex : svn info 'https://lpsc.in2p3.fr/tests pour Planck/dev'

- Différentes méthodes d'accès au repository :

Préfixe de l'url	**Méthode d'accès au repository**
file://	Accès direct
http://	Via apache
https://	Via apache avec une encryption ssl
svn://	Via un protocole spécifique vers un svnserve démon
svn+ssh://	Via un protocole spécifique vers un svnserve démon à travers un tunnel ssh

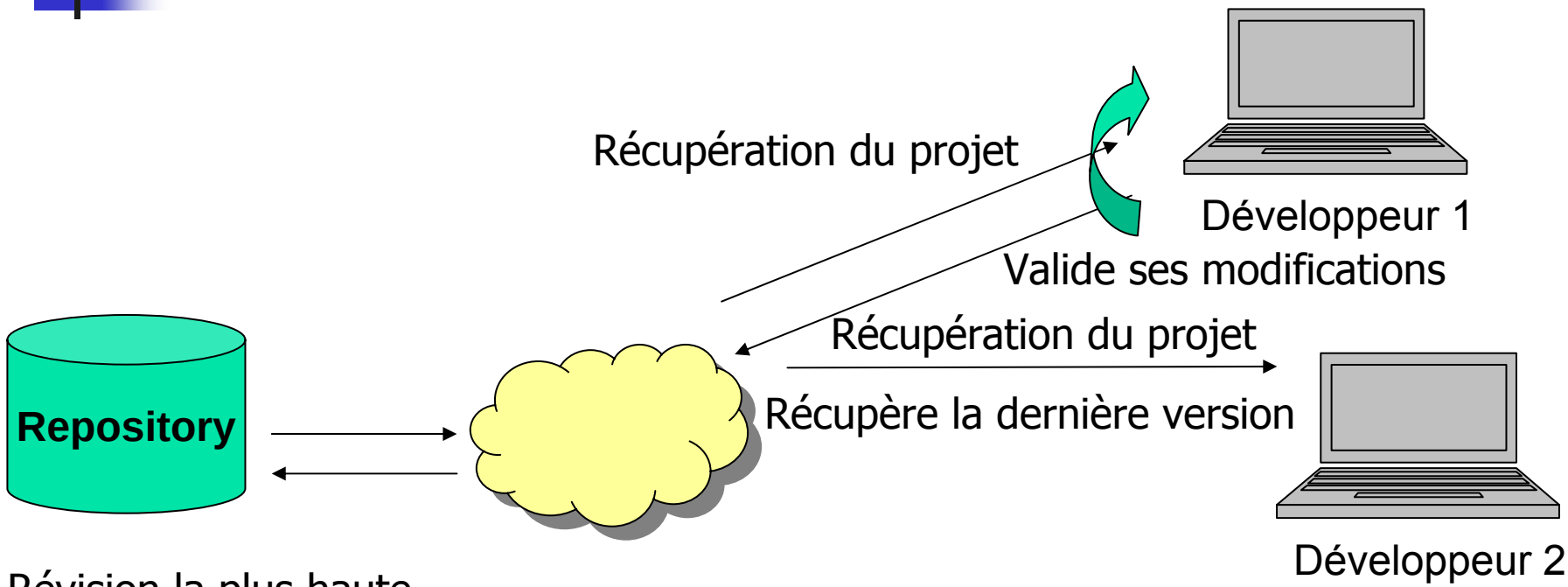
- C'est l'administrateur SVN qui fournit cette URL aux utilisateurs

2.5 Précisions sur la Working copy

- Une working copy est une hiérarchie de fichiers ordinaire dans votre système de fichiers
- Vous pouvez éditer, modifier l'intérieur d'une working copy comme si elle n'en était pas une, avec n'importe quel outil (vi, nedit, emacs, word, excel, gimp, …). On peut aussi le compiler, debogguer, tester, … avec ou sans connexion réseau
- Votre working copy est un 'espace privé'. Vous n'obtiendrez ni les modifications des autres ni ne soumettrez les vôtres si vous ne le demandez pas explicitement à l'aide de commandes SVN dédiées à cela
- Une working copy contient également des fichiers 'supplémentaires', créés et maintenus par SVN. Ils se trouvent dans des répertoires .svn, créés dans chaque répertoire de la hiérarchie.

  NE JAMAIS LES MODIFIER !
- Ils servent à SVN pour connaître l'état des fichiers, permettre l'exécution de commande SVN sans avoir à spécifier l'URL du repository, …

2.5 Précisions sur la Working copy



Révision la plus haute du repository (HEAD) :

~~17~~
~~18~~
19

Révision de référence du développeur 1 : ~~17~~ ~~18~~ 19

Révision de référence du développeur 2 : ~~17~~ 19

2.5 Précisions sur la Working copy

- Pour chacun des fichiers, Subversion enregistre deux informations importantes dans les répertoires .svn:
  - Sur quelle révision est basée votre working copy (version de référence), Vref
  - L'heure et la date à laquelle votre copie locale a été mise à jour par le repository Tref

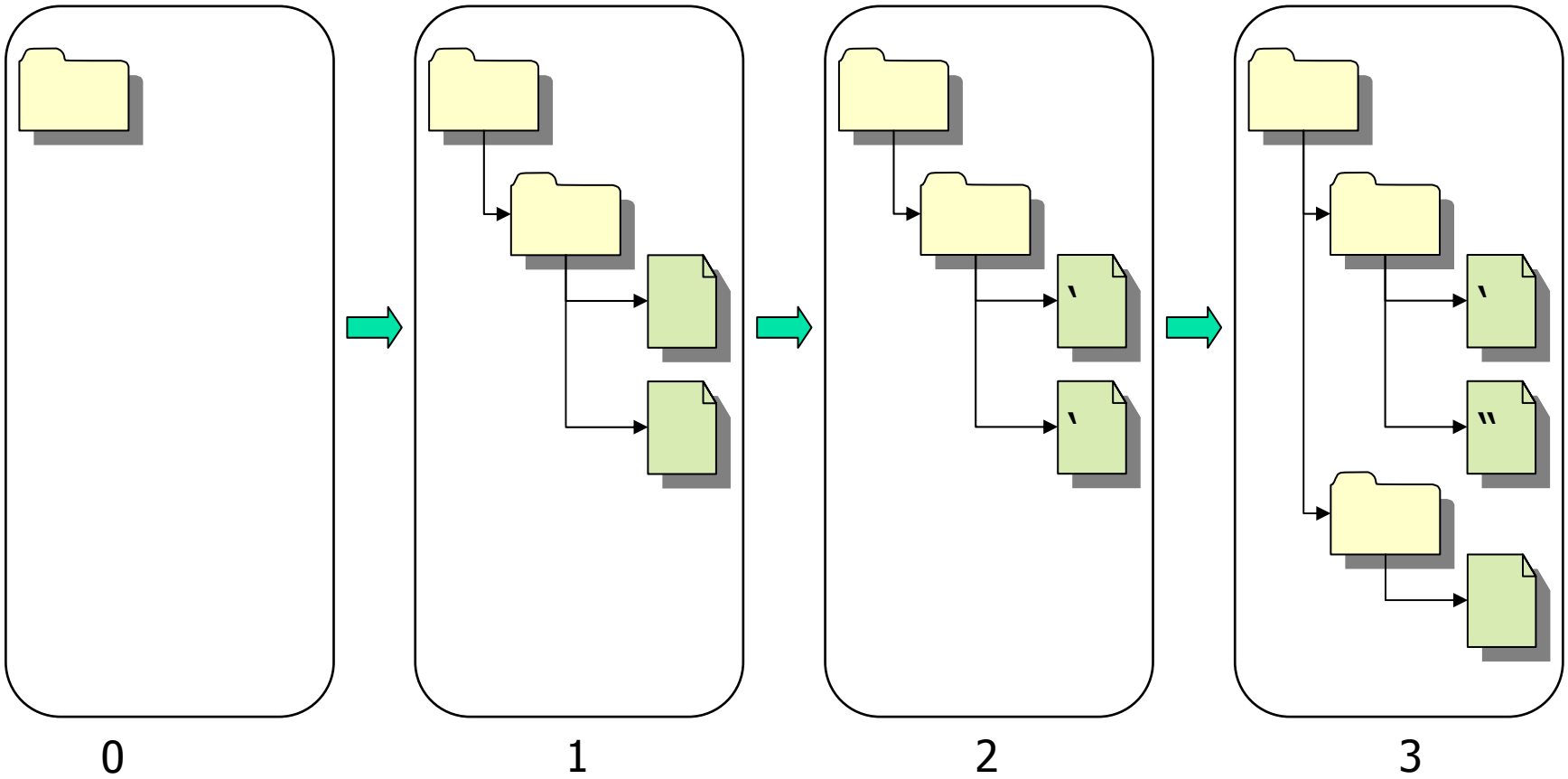
- A partir de ces informations ainsi que de la version la plus haute dans le repository, Vrep et de la date de dernière modification Tmod, SVN peut trouver 4 états différents pour chacun de vos fichiers :

	Observation	**Que fait une validation de modification ?**	**Que fait une mise à jour de working copy ?**
Inchangé localement et à jour	$T_{ref} = T_{mod}$ et $V_{ref} = V_{rep}$	Rien	Rien
Changé localement et à jour	$T_{ref} \neq T_{mod}$ et $V_{ref} = V_{rep}$	Met à jour le repository	Rien
Inchangé localement et 'out of date'	$T_{ref} = T_{mod}$ et $V_{ref} \neq V_{rep}$	Rien	Met à jour votre working copy
Changé localement et 'out of date'	$T_{ref} \neq T_{mod}$ et $V_{ref} \neq V_{rep}$	Réclame une mise à jour de working copy	Met à jour votre working copy

2.6 Révisions

- Une validation de modifications (commit) sur le repository provoque une transaction atomique qui concerne un certain nombre de fichiers et/ou répertoires (modification, ajout, suppression, renommage, déplacement, …)
- A chaque commit, un nouvel état du repository est créé, nommé révision
- A chaque révision est assigné un numéro entier, incrémenté de 1 vis-à-vis de la révision précédente
- Un repository tout juste créé (cad vide) a pour numéro de révision 0

2.6 Révisions

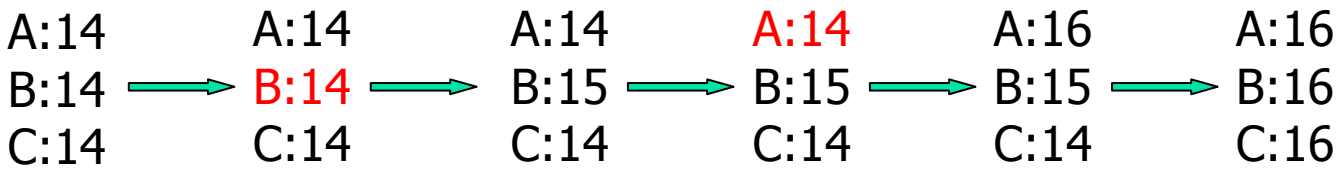


2.6 Révisions

- Les numéros de révisions sont globaux
  - contrairement aux autres systèmes de gestion de versions, les numéros de révision de SVN s'appliquent à toute la hiérarchie de fichiers et non pas fichier par fichier
  - Une révision correspond à un état global de tout le repository, après un commit
  - La révision 5 correspond donc à l'état du repository après le 5ème commit
  - La révision 5 du fichier λ correspond donc à l'état du fichier λ à la révision 5
  - Le contenu de deux révisions différentes d'un même fichier ne diffère pas nécessairement

2.6 Révisions

- Dans une working copy on a souvent un mixage de numéros de révisions



- Après une mise à jour, tous les fichiers ont le même numéro de révision de dernière synchronisation
- Après une validation de modifications, généralement il y a différents numéros de révision de dernière synchronisation
- Les mises à jour et les validation de modifications sont des opérations totalement indépendantes

2.7 Commandes SVN – généralités

- Syntaxe générale : svn commande [options] [arguments]
- L'ordre où apparaissent les options n'est pas important
- Hors d'une working copy il est obligatoire de spécifier l'URL du repository
  Dans une working copy, il n'est plus nécessaire de la préciser
- Les commandes peuvent agir sur :
  - L'URL d'un repository,
  - un fichier,
  - un ensemble de fichiers,
  - Un lien symbolique (sauf sous windows)
  - un répertoire,
  - ou encore être sans cible
    => le répertoire courant ('.') est la cible par défaut
- Si une commande agit sur un répertoire, elle agit récursivement sur tous les sous répertoires et fichiers qu'il contient (sauf quelques exceptions)

2.7 Commandes SVN – généralités

- Pour obtenir de l'aide :
  - svn help => pour obtenir la liste des commandes
  - svn help commande => pour connaître les arguments et les options d'une commande particulière
- Les commandes ont souvent des alias (ex : svn checkout = svn co)
- Il est souvent nécessaire d'identifier des révisions particulières dans une commande :
  - Une particulière : --revision REV ou -r REV
  - Un intervalle : --revision REV1:REV2 ou -r REV1:REV2
- REV peut être :
  - Un numéro de révision : --revision 3
  - Un mot clé :
    - HEAD, la plus haute révision du repository
    - BASE, la révision de référence de votre working copy
    - ..
  - Une date : "{2007-04-23}", "{2005-12-17 15:36}"

PLAN

3 Principales fonctionnalités

- 3.1 Checkout / Update / Commit
  - 3.1.1 Commande checkout
  - 3.1.2 Comment SVN effectue ses comparaisons (rappel)
  - 3.1.3 Commande update
  - 3.1.4 Commande commit
- 3.2 Autres commandes utiles
  - 3.2.1 Modifications de la working copy – commandes import, export, add, delete, copy, move, mkdir et revert
  - 3.2.2 Examiner vos modifications - commandes status et diff
  - 3.2.3 Résoudre les conflits - commande resolved
  - 3.2.4 Connaître l'historique du repository - commandes log, diff, list et cat
- 3.3 Gestion des méta données
  - Exemple d'utilisation des méta données : portabilité de fichiers
- 3.4 Branches et tags
  - 3.4.1 Branches – définition et utilité
  - 3.4.2 Branches – création
  - 3.4.3 Branches – svn merge
  - 3.4.4 Tags
  - 3.4.5 Divers

3.1 Checkout / update / commit

	Action	Modification du repository	Modification de la working copy
Checkout (co)	Récupérer localement un projet depuis le repository, pour le modifier (Working Copy)	NON	OUI
Update (up)	Mettre à jour localement un projet à partir du repository (Synchronisation) => Récupérer les modifications des autres	NON	OUI
Commit (ci)	Valider des modifications locales dans le repository => Rend disponible les modifications pour les autres	OUI	NON

3.1.1 Commande checkout

- Afin d'utiliser un repository la plupart du temps, on récupère un projet déjà existant :

  svn co https://lpsc.in2p3.fr/svn/projetA

  => création d'une working copy à partir de la version la plus récente du projet

- Cela produit un répertoire projetA, la working copy, à l'intérieur de laquelle les commandes SVN vont fonctionner

- Ce répertoire est alors entièrement modifiable, avec n'importe quel éditeur ou outil (vi, nedit, emacs, word, excel, gimp, …). On peut aussi le compiler, tester, … avec ou sans connexion réseau. On peut même le supprimer !

- Cette commande peut agir sur n'importe quel répertoire du repository

- Checkout est exécuté rarement : une **seule** fois pour chaque projet

  => pour se mettre à jour (récupérer les modifications des autres utilisateurs), il faut utiliser la commande update

3.1.2 Comment SVN effectue ses comparaisons (rappel)

- 3 origines de comparaisons pour chaque fichier :
  - La version courante dans la working copy
  - La révision de dernière synchronisation avec le repository (checkout ou update)
  - La révision de référence => toujours la plus récente révision du repository (la HEAD révision)

- SVN calcule deux types de changement :
  - entre le second et le troisième, il trouve les ***changements dans le repository*** => ceux des autres utilisateurs
  - entre le premier et le second, il trouve vos ***changements en local***

3.1.3 Commande update

- Pour mettre à jour votre working copy avec la version la plus récente du repository :
  svn up
  => Se synchroniser avec la version du repository
  => Pour obtenir les modifications apportées par les autres
  => On ne spécifie plus l'url du repository

- Car SVN ne va vous laisser modifier que la HEAD révision d'un fichier et non pas une version ancienne
  => obligatoire avant de valider vos modifications dans le repository

- **Il s'agit d'un merge (combinaison) des modifications des autres avec votre version courante (changements dans le repository et changements en local)**
  => possible automatiquement ou non

- Quand le serveur effectue des mises à jour dans votre working copy, il vous l'indique à l'aide de la convention suivante :
  - U toto : le fichier toto a été mis à jour sans merge (**U**pdate)
  - A toto : le fichier ou le répertoire toto a été ajouté à votre working copy (**A**dded)
  - D toto : le fichier ou le répertoire toto a été supprimé de votre working copy (**D**eleted)
  - G toto : le fichier toto a été mis à jour avec un merge (mer**G**ed)
  - C toto : le fichier toto n'a pas pu être mis à jour à cause d'un conflit (**C**onflict)
  - …

3.1.4 Commande commit

- Vous mettre à jour :
  svn update
  obligatoire si d'autres personnes ont modifié le code

- Valider vos modifications locale :
  svn commit => ouvrira votre éditeur préféré
  ou *svn commit --message "message"* pour un message court
  ou *svn commit --file logmessage* sinon
  => Envoie vos modifications au repository, quel que soit leur signification
  => Renvoie un message 'out of date' si vous n'êtes pas à jour
  => Cela rend vos modifications disponible pour les autres
  => Le message doit décrire vos modifications

- **Règle importante :** aucun changement en local n'est jamais effectué dans le repository avant un commit

- Fréquence des 'svn ci' : très dépendant du projet

3.2.1 Modifications de la working copy

- **svn import –m 'mes' projet https://lpsc.in2p3.fr/svn/web**
  Copier une hiérarchie de fichiers non gérés par svn dans un repository (ne nécessite pas de working copy)

- **svn export https://lpsc.in2p3.fr/svn/Web/resa/trunk reservation**
  Pour obtenir une copie de la hiérarchie de fichiers
  - pas de création de working copy !

  => pour mettre en production ou diffuser un logiciel

- **svn add fichier.c**
  => Planifie l'ajout de fichier, répertoire ou lien symbolique dans le repository

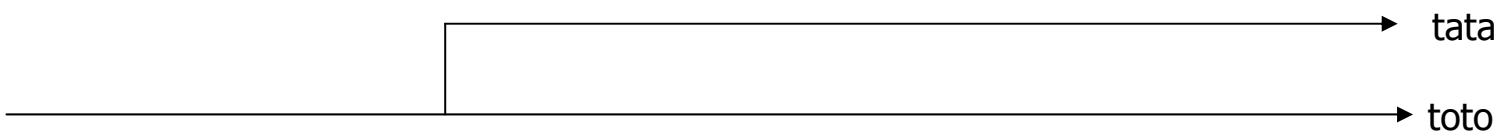
- **svn delete fichier.c**
  => Planifie la suppression de fichier, répertoire ou lien symbolique dans le repository
  => Bien sûr toto n'est supprimé que de la HEAD révision et reste disponible dans l'historique

- **svn revert fichier.c**
  => Pour annuler vos modifications en local

3.2.1 Modifications de la working copy

- **svn copy toto tata (cp)**
  => Crée un nouveau fichier ou répertoire tata identique à toto dans la working copy et planifie cette copy dans le repository
  => Lors de son ajout dans le repository, au commit suivant, son origine est sauvegardée dans son historique (il provient d'une copie de toto)

Cette commande, très utilisée, est très peu coûteuse en temps et en espace disque (équivalent d'un lien symbolique)



- **svn move toto tata (mv, rename, ren)**
  => Équivalent à svn copy toto tata puis svn del toto

- **svn mkdir initialisations**
  => Équivalent à mkdir toto puis svn add toto

3.2.2 Examiner vos modifications

- Avant d'effectuer un svn commit, vous pouvez vouloir vérifier vos modifications
  - Pour voir ce qui a été modifié : svn status
    (Après *svn checkout*, *svn update* et *svn commit*, la commande la plus utile !)
  - Pour voir les modifications en détail : svn diff

  => Permet de voir par exemple qu'un fichier a été modifié par inadvertance

  => Permet d'affiner le message de log

- Ces opérations sont possibles sans connexion au repository si elles ne concernent que vos modifications en local
  - Car svn conserve une version cachée de votre révision de référence
    => cela lui permet également lors d'un commit de n'envoyer que les modifications et non tous les contenus

3.2.3 Résoudre les conflits

- Rappel (modèle copy – modify - merge) :

svn up

	Modifications locales	Modifications sur le serveur
U globus.h	Non	Oui
G globus.c	Oui	Oui
C server.c	Oui	Oui

- Les conflits, dus à des modifications sur une même portion de code, sont rares :
  - Une seule personne travaille sur un morceau particulier de code
  - Les personnes travaillant sur la même fonctionnalité devraient se parler
- SVN gère les révisions de hiérarchie de fichiers, il ne remplace ni la gestion de projet ni la communication

  Mais il propose une aide à la résolution manuelle de conflit

3.2.4 Connaître l'historique du repository

- Le repository conserve toutes les modifications validées par chaque commit

- Il vous permet de 'remonter le temps' et d'examiner toutes les révisions :
  - En récupérant une révision donnée (*svn co -r REV* ou *svn update -r REV*)
  - En questionnant l'historique du repository :
    - svn log => affiche tous les messages des log
    - svn diff => affiche le détail des modifications
    - svn cat => affiche le contenu d'un fichier pour une révision
    - svn list => liste le contenu d'un répertoire pour une révision

3.3 Gestion des méta données

- En plus de la gestion de versions des fichiers et des répertoires, svn gère :
  - les méta données (ou propriétés) sur ces fichiers et répertoires
  - leur gestion de versions

- Une propriété est un couple (nom de propriété , valeur de propriété)
  Ex : ('copyright', 'Paul et André'),
  ('tests unitaires effectués','partiellement'), …

- svn utilise également de façon interne des méta données :
  - Pour sauvegarder des caractéristiques de fichiers ou de répertoires
    ex : svn:executable, svn:mime-type, svn:ignore, svn:needs-lock, svn:eol-style, svn:keywords, …
  - Pour conserver les caractéristiques des commit, des méta données associées à chaque révision
    ex : svn:author, svn:date, svn:log, …

- Vous pouvez également modifier ces méta données, si vous savez ce que vous faites !

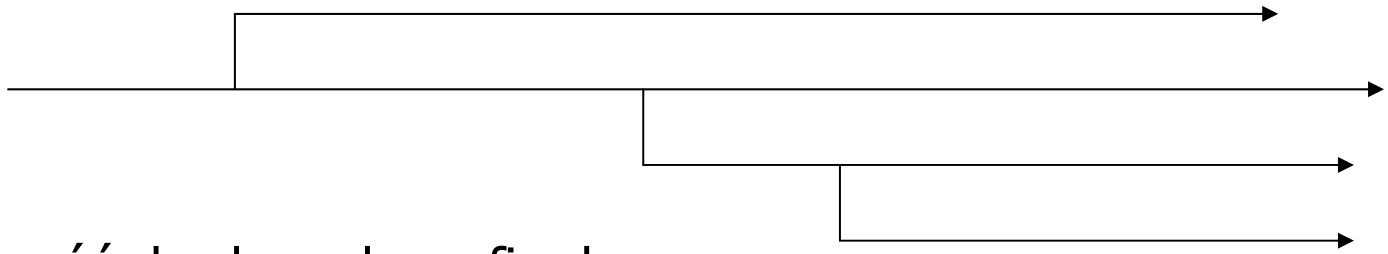
3.3 Utilisation des méta données : portabilité de fichiers

- Portabilité des fichiers
  - La séquence de caractères permettant de repérer les fins de lignes est différente sous Windows (CRLF) et sous Linux (LF)
  - Linux utilise les liens symboliques, pas Windows
  - L'exécutabilité d'un fichier est défini par un bit dans le système de fichiers sous linux, par une extension de nom de fichier différente sous Windows
  - …
- Svn gère ces différences et donc la portabilité des fichiers à l'aide des méta données :
  - svn:executable (existe ou non)
  - svn:mime-type (rien, 'application/octet-stream' ou mime-type plus précis : 'image/png', 'application/x-shockwave-flash', 'text/plain', …)
  - svn:eol-style ('native', 'CRLF' ou 'LF')
  - svn:special (pour repérer les liens symboliques)

3.4.1 Branches – définition et utilité

- Une branche est une ligne de développement indépendante des autres mais qui partage un historique commun

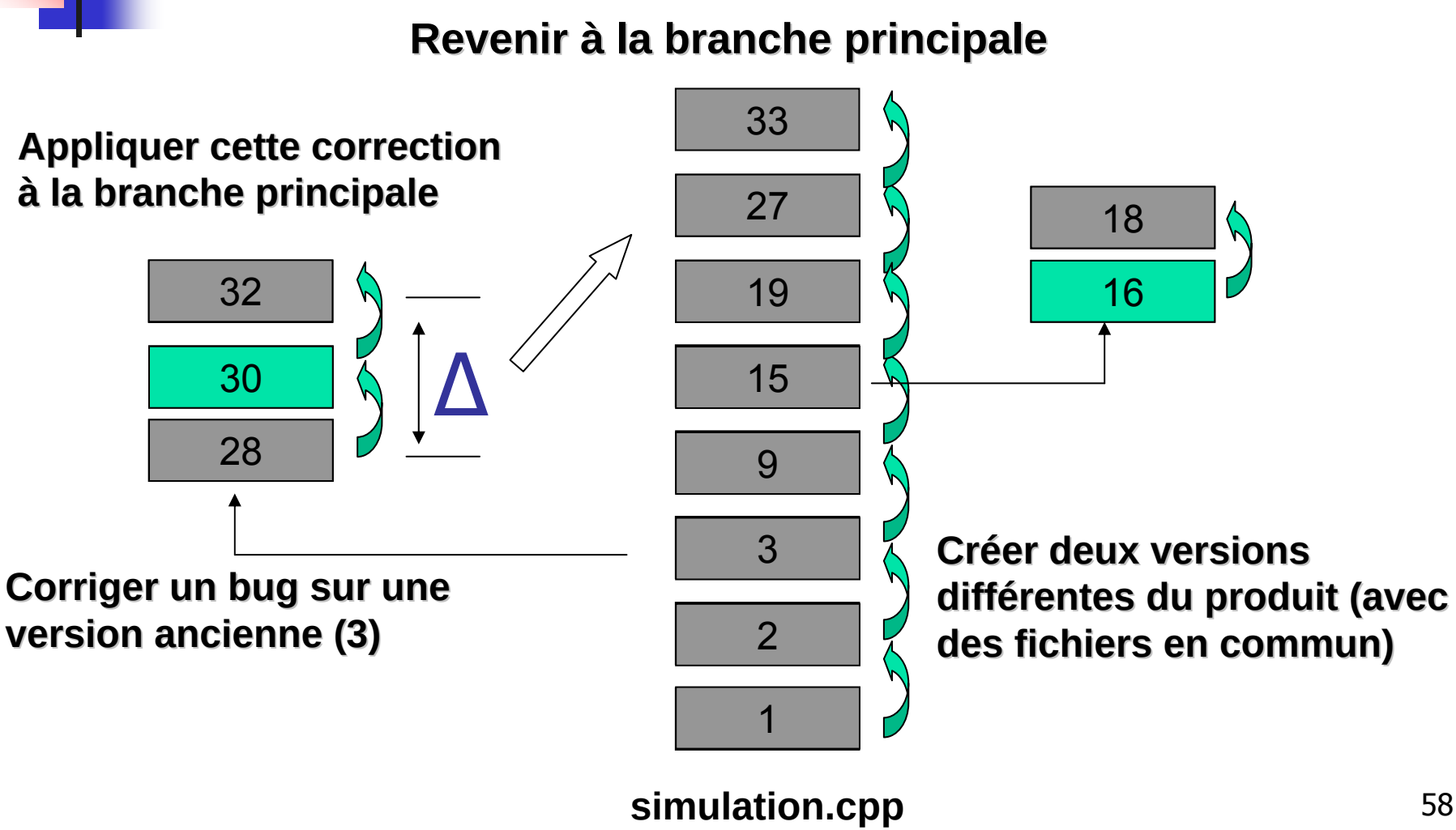
  Elle est toujours créée à partir d'une copie d'un répertoire déjà existant et possède ensuite son propre historique



- On créé des branches afin de :
  - maintenir plusieurs versions très similaires en même temps du même logiciel => branches de fonctionnalité

    Ex : un programme de gestion de ventes de PC et d'imprimantes partageant 90% de code commun
  - corriger des erreurs sur des versions anciennes
  - développer temporairement dans un environnement différent de celui des autres développeurs

3.4.1 Branches – définition et utilité



3.4.2 Branches – création

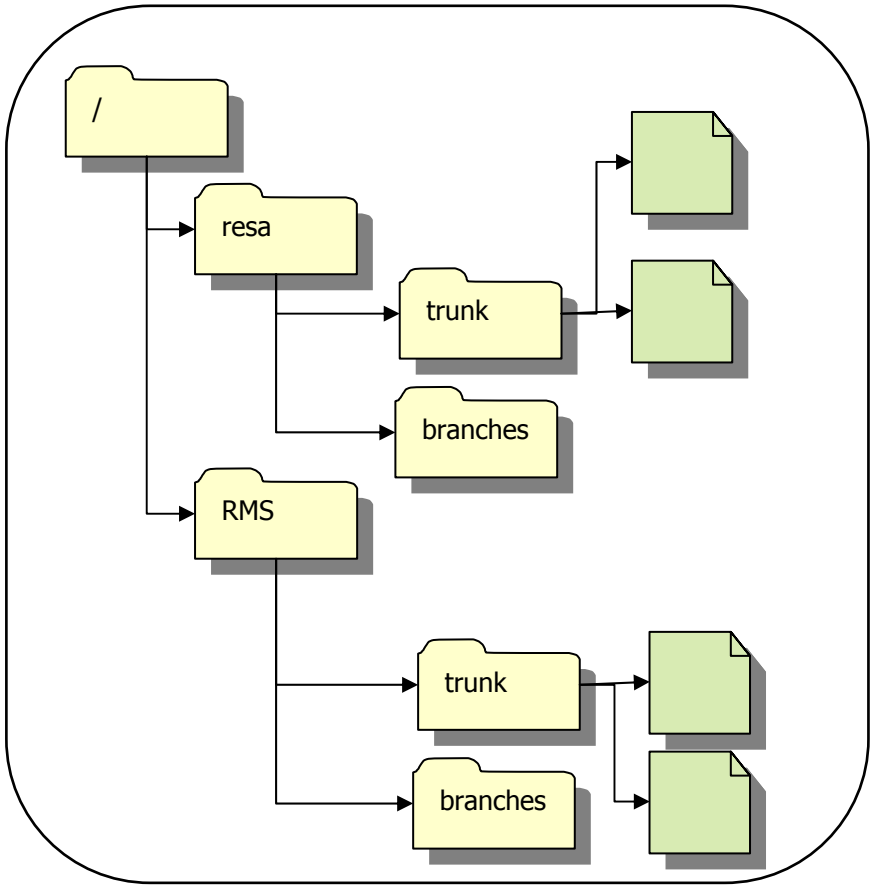
- Trunk est une branche particulière : c'est la branche d'origine, qui reste souvent la branche 'principale'
- Pour créer une branche, vous effectuez une copie de la partie du repository qui vous intéresse, en utilisant la commande *svn copy*
- Cette copie peut s'effectuer n'importe où mais il est de coutume de l'effectuer dans le répertoire '*branches*'
- Une copie est une opération à temps et espace disque constant

  => Vous pouvez en créer aussi souvent que vous le désirez

3.4.2 Branches – création

- Les branches ne partagent pas le même emplacement dans la hiérarchie de fichiers du repository mais partagent le même référentiel de temps : les mêmes numéros de révision
- Les autres utilisateurs qui travaillent dans le trunk, lors de leurs *'svn update'* ne récupéreront pas vos mises à jour effectuées dans la branche
- Concepts clé : svn n'a pas de concept interne de branche.
  - Le résultat d'une copie est une branche car vous lui associez cette signification
  - Une branche est en fait un répertoire comme les autres, que svn gère comme n'importe quel répertoire
  - Cela est contraire aux autres systèmes de gestion de versions qui utilisent une dimension supplémentaire

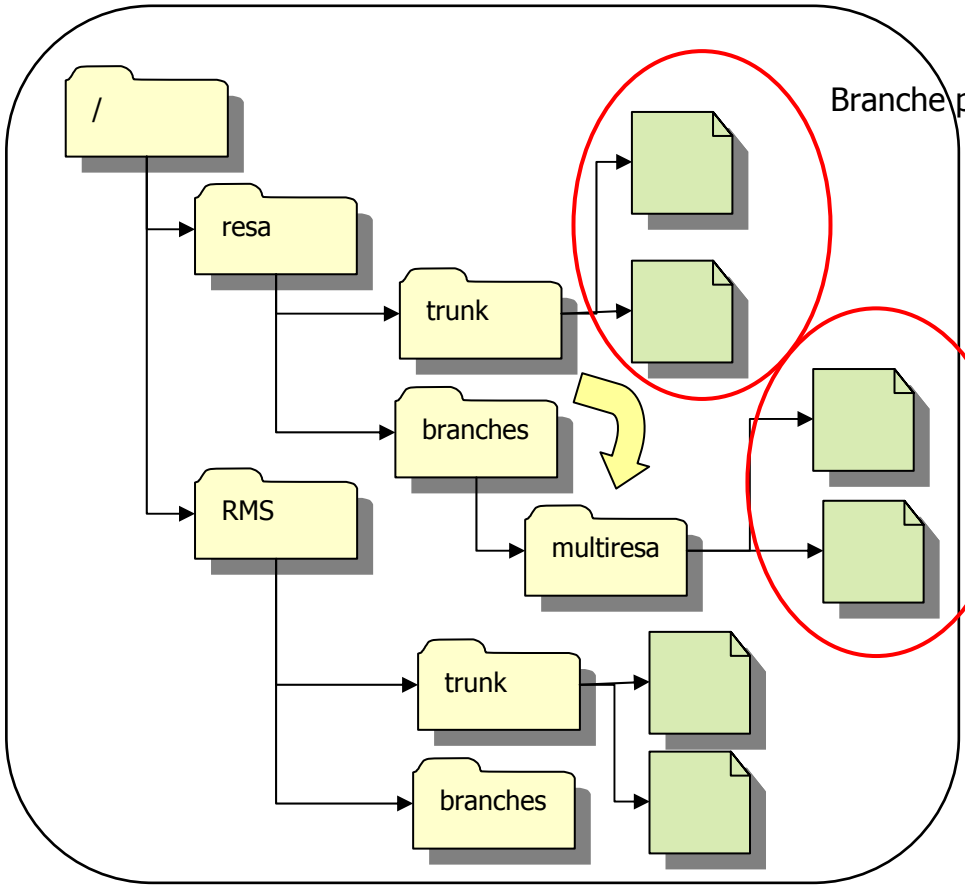
3.4.2 Branches – création



- Vous désirez écrire une nouvelle fonctionnalité, qui va affecter tous les fichiers du projet pendant plusieurs semaines
- Vous ne voulez pas déranger vos collègues qui développent d'autres fonctionnalités
- Vous désirez effectuer des *svn commit* régulièrement sans pour autant finaliser vos modifications
  - Car vos modifications sont très importante en taille
  - Vous désirez pouvoir sauvegarder vos modifications et les diffuser (avec vos collègues ou si vous avez plusieurs working copy)
  - Vous désirez pouvoir récupérer les modifications effectuées par les autres dans la branche principale

=> création d'une branche

3.4.2 Branches – création



Branche principale

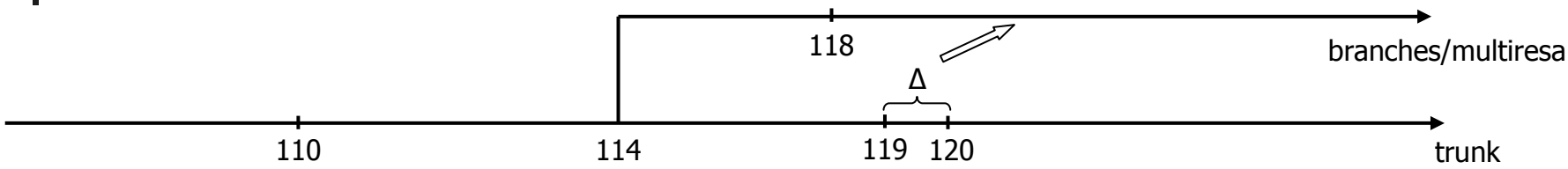
Branche pour le développement parallèle

svn copy trunk branches/multiresa

svn commit -m 'création de la branche multiresa'

Vous allez donc créer un nouveau répertoire 'multiresa' dans le répertoire resa/branches, qui va commencer d'exister en tant que copie du répertoire resa/trunk

3.4.3 Branches – svn merge



- Afin de ne pas trop diverger, il peut être utile de récupérer de temps en temps les modifications des autres d'une branche à une autre

- Pour voir les différences :

  svn diff -r 119:120 https://xxx/resa/trunk

- Pour les récupérer :

  cd resa/branches/multiresa

  svn merge -r 119:120 https://xxx/resa/trunk

  Cette commande est très similaire à svn diff mais au lieu d'afficher les différences, elle les applique à votre working copy, en tant que modifications locales :

  svn status

  M calendar.php

3.4.4 Tags

- Un tag est un snapshot du repository à un instant donné
- Une révision est déjà un snapshot de la hiérarchie de fichiers à un instant donné
- Mais un tag permet de lui donner un nom plus attractif qu'un entier et peut ne concerner qu'une partie de la hiérarchie de fichiers
  Ex : il est plus simple de se souvenir du tag 'release-1.9' que de retenir que la version finalisée est un répertoire particulier de la révision 8842
- Pour créer un tag :
  svn copy https://xxx/resa/trunk https://xxx/resa/tag/version-1.1 -m 'tag de la version 1.1 de resa'
  => Il s'agit de la même méthode que pour créer une branche !!!
- Pour subversion une branche ou un tag sont des répertoires comme les autres, créés par une copie. La différence entre les deux, c'est vous qui la faites !
- Si vous ne modifiez pas un répertoire copié, il s'agit d'un tag ; dés que vous modifiez pas un répertoire copié, il s'agit d'une branche
- C'est pour les distinguer que l'on créé par défaut des répertoires trunk, tags et branches

3.4.5 Test d'une connexion à un repository svn

- Installer la dernière version de svn et mettre l'exécutable dans le PATH
- Demander à votre administrateur svn l'url du repository
- Vous authentifier avec la méthode adéquate (mises au point préalables souvent nécessaires)
- Essayer une commande d'interrogation non liée à une working copy
  ex :
  - svn list https://xxx
  - svn info https://xxx
- Installer une interface graphique

3.4.5 Svn et les authentifications

- *svn --version* permet de connaître :
  - La version du client svn
  - Les protocoles de communication que svn connaît (http, https, svn ou file)
- Pour la plupart des opérations svn n'a pas besoin de contacter le serveur => pas d'authentification nécessaire
- svn ne demande de s'authentifier que lors d'opérations le nécessitant
- Après une authentification, svn met en cache (dans des fichiers dans ~/.subversion/auth) vos paramètres d'authentification et ne vous les redemandera pas

  => sauf en cas d'authentification par certificat

  => sauf si vous utilisez --no-auth-cache ou store-auth-creds (voir après)

  - Sous windows, protection par encryption avec une clé par utilisateur
  - Sous linux, protection par droits sur les fichiers

3.4.5 Configuration du client svn

- Les utilisateurs peuvent vouloir utiliser de façon systématique de nombreuses options des commandes svn
  - Ex : --no-auth-cache pour chaque commande
- Pour cela svn permet de stocker vos 'préférences' dans des fichiers de configuration
  - Fichier servers (relatif aux couches réseau)
  - Fichier config (tout ce qui ne concerna pas le réseau)
  - Stockés dans la zone de configuration de svn
    - Une propre à chaque machine
    - Une propre à chaque utilisateur
- Ces fichiers sont modifiables à l'aide de n'importe quel éditeur et ne sont pas versionnés !

3.4.5 Localisation

- Permet d'obtenir une internationalisation de svn :
  - Pour tous les messages émis par svn
    - En fonction de la variable d'environnement LC_MESSAGES affiche tous les messages dans une langue donnée
    - Cela n'est possible que si subversion possède le fichier de traduction pour la langue spécifique

      ex : /usr/share/locale/de/LC_MESSAGES/subversion.mo
  - Pour les données saisis vers svn
    - svn convertit les noms de fichiers, les chemins et les messages de log depuis votre langue vers de l'unicode, en UTF-8
- Pour le français : *setenv LC_ALL fr_FR*

PLAN

4 Exemples d'utilisation de SVN

Rappel

- Subversion gère les fichiers et les répertoires, ainsi que leurs modifications à travers le temps.

  Ainsi tout document mis au point et maintenu par un ou plusieurs acteurs seront à l'abri des fausses manœuvres et toujours disponibles de la première à la dernière version

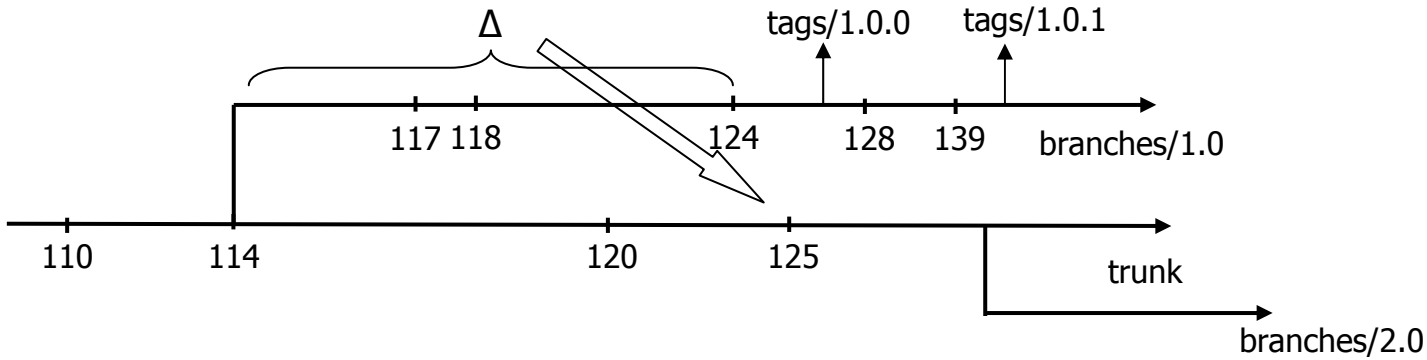
- Principale utilisation : gérer les modifications de code source mais également toute sorte d'information : documentation, images, fichiers de configuration, …

  Pour Subversion une données est une données quelle soit son type

- 3 exemples d'utilisation hors développement de code à plusieurs :
  - Tout travail collaboratif en tirera du bénéfice : rédaction d'un article à plusieurs à l'aide de LaTex
  - Utile aussi bien pour les collaborations que pour les individus
  - Dans un contexte ASR : sauvegarde de l'évolution des fichiers de configuration

4 Exemples d'utilisation de SVN

- Développement de logiciel distribué sous forme de release
  - Éviter les fausses manœuvres
  - Utilisation de branches, pour corriger les bugs des anciennes versions et gérer l'évolution des versions



  - Fin des développements de la version 1.0 du code dans le trunk, en 114
    => Il faut la tester et commencer à développer la version 2.0
  - Création d'une branche de release
    - Correction de bug
    - Développement de la version 2.0
    - Portage(s) des corrections de bugs dans la branche principale
    - La branche est taggée 1.0.0 et délivrée
    - Elle est par la suite maintenue et peut donner lieu à d'autres releases
  - Création d'une nouvelle branche de release pour la version 2.0

4 Exemples d'utilisation de SVN

Facilité de création des releases à l'aide d'un script shell :

```
#!/bin/tcsh
cd /tmp
rm -rf resa*
svn export https://lpsc.in2p3.fr/.../resa/trunk resa
cd resa
mv commun/config_ini.php commun/config.php
mv commun/database_ini.php commun/database.php
rm doc/*.doc
rm img/logo.*
cd ..
chmod -R 500 resa
...
chmod 600 resa/commun/database.php
chmod 600 resa/commun/config.php
chmod 700 resa
tar cvf - resa | gzip > resa.tar.gz
chmod -R 700 resa
rm -rf resa
```

4 Exemples d'utilisation de SVN

Contexte ASR : gestion des fichiers / scripts de configuration (ex : apache)

- La configuration d'un système évolue
  - Intéressant de conserver l'historique
  - Pouvoir revenir en arrière
    - Pour des modifications simples
    - Ou pour des modifications complexes (ex : impliquant plusieurs fichiers)
  - Plusieurs administrateurs impliqués
    - Détecter les conflits
    - Identifier les changements des autres
  - Configuration d'un service partagé par plusieurs machines
    - Propager facilement les modifications
    - Garantir la synchronisation des différentes machines
  - Pouvoir changer rapidement de version

PLAN

5 Eléments d'installation d'un serveur

- 5.1 Quelques pistes de réflexion
- 5.2 Création d'un repository
- 5.3 svnserve versus apache
- 5.4 svnserve
- 5.5 svnserve + ssh
- 5.6 apache

5.1 Quelques pistes de réflexion

- Quelles données mettre dans le repository ?
- Comment les organiser ?
  - Un ou plusieurs repository ?
    - un seul repository avec plusieurs projets
      - ne pas multiplier la maintenance (un backup, un set de hooks, de dump/load pour des mises à jour incompatibles, …)
      - mais les numéros de révisions vont dépendre de tous les projets
      - les triggers peuvent être différents => plus de code à écrire
    - plusieurs repository avec un seul projet par repository
    - plusieurs repository avec un ou plusieurs projets par repository
      - un compromis
  - Architecture à l'intérieur de chaque repository ?
    - (trunk, tags, branches) avant ou après les projets ?
      - les projets avant permet d'accéder à tout le projet avec une seule url
    - Bien que svn nous permet de modifier facilement la structure de la hiérarchie de fichiers, il est préférable de penser son organisation avant (trop de *svn move* sur les répertoires principaux d'un repository peuvent être gênant pour les utilisateurs)

5.1 Quelques pistes de réflexion

- Comment accéder au repository ?
  - En local, sur le réseau ?
  - Via http ou svnserve ?
- Quels types de contrôle d'accès prévoir ?
  - Authentification par login / mot de passe, par certificat, …
  - Autorisation par projet, par répertoire, …
  - Autorisation en lecture seule, en lecture /écriture
- Faut-il prévoir une gestion d'événements ?
  - Ex : envoi de mail automatique après chaque commit, test avant un commit pour savoir si on l'exécute, en fonction du résultat de la compilation, …
  - À l'aide des *'hooks'*
- Quel type de stockage utiliser ?
  - Berkeley DB
  - FSFS, les changements associés à une révision sont stockés dans un fichier ayant pour nom le numéro de la révision. Lors d'un commit les changements sont écrits dans un fichier de transactions, qui est ensuite converti en révision, ce qui garantit l'atomicité des commit

5.2 Création d'un repository

- Créer un repository de type FSFS :

  svnadmin create /usr/local/SVN/electronique

  et c'est tout !

- Ensuite la maintenance s'effectue avec des commandes d'administration qui s'utilise sur le serveur

- Ne modifier **jamais** les fichiers du repository manuellement => perdre les habitudes prises avec CVS, sauf pour les hooks

- Vous pouvez tout de même visualiser (pour information) le répertoire /usr/local/SVN/electronique/, en particulier le répertoire db qui contient :
  - les révisions (répertoire revs)
  - les propriétés (répertoire revprops)
  - les éventuelles transactions (répertoire transactions)

5.3 svnserve versus apache

- Une fois le repository créé, la question suivante est sa mise à disposition sur le réseau

- **Apache** est un serveur web très utilisé et très bien connu
  En utilisant le module mod_dav_svn, Apache peut accéder à un repository et le rendre disponible aux clients svn, via le protocole WebDAV/DeltaV, qui est une extension d'HTTP
  => on récupère toutes les possibilités d'Apache : encryption via SSL, authentification HTTP, …

- **Svnserve** est un programme serveur léger qui communique avec ses clients svn à l'aide d'un protocole propre.
  Il est plus rapide, mais en contrepartie possède moins de fonctionnalités : que les authentifications MD5, pas de log, pas d'encryption
  Il est cependant très simple à mettre en place et est souvent la meilleure solution pour les petites équipes qui veulent démarrer très vite

- Une autre solution consiste à utiliser svnserve à travers un **tunnel ssh**
  => avantages de svnserve + encryption mais contreparties …

5.3 svnserve versus apache

	Apache + mod_dav_svn	**svnserve**	**svnserve over SSH**
Authentification	HTTP(S) basic auth, certificats, LDAP, …	MD5	Utilisateurs ssh
Comptes utilisateurs	Fichier propre d'utilisateurs	Fichier propre d'utilisateurs	Comptes systèmes
Autorisation	Accès en lecture et/ou écriture répertoire par répertoire	Accès en lecture et/ou écriture répertoire par répertoire	Accès en lecture et/ou écriture répertoire par repository
Encryption	via SSL	N/A	Tunnel SSH
Logging	Log d'apache + possibilité de log 'haut niveau' sur les commandes passées	N/A	N/A
Navigation Web	Par défaut mais limité à la HEAD révision	Pas par défaut	Pas par défaut
Rapidité	Plutôt plus lent	Plutôt plus rapide	Plutôt plus rapide
Mise en œuvre	Plutôt complexe	Extrêmement simple	Assez simple

5.4 svnserve

- Pour lancer le service:
  - En tant que service 'standalone'
    - svnserve –d
      => écoute le port 3690
      => svn list svn://lpsc.in2p3.fr/usr/local/repository/projet1
    - svnserve –d –r /usr/local/repository
      => svn list svn://lpsc.in2p3.fr/projet1
  - En tant que service géré par inetd
  - Cas de svnserve à travers un tunnel
    - svnserve -t
      => dans ce cas c'est un programme annexe qui a déjà effectué l'authentification (rsh, ssh, …)
  - En tant que service géré par Windows

- Pour chaque requête d'un client :
  - Le client spécifie le repository
  - Le serveur vérifie le contenu du répertoire conf/svnserve.conf de ce repository
    - Soit la requête est permise de façon anonyme
    - Soit un challenge d'authentification est proposé (les mots de passe ne passent jamais sur le réseau)
    - En mode tunnel, le client est considéré déjà authentifié

5.4 svnserve

- Le fichier conf/svnserve.conf est le point central pour définir les authentifications et les autorisations

- Exemple de contenu de ce fichier:
  [general]
  password-db = /usr/local/SVN_users
  realm = SVN namespace
  anon-access = read // none, read or write
  auth-access = write
  authz-db = /usr/local/SVN_auth (voir plus loin son contenu)

- Contenu d'un fichier SVN_users :
  [users]
  john = password
  marc = password
  => les mots de passe sont en clairs sur le serveur

5.5 svnserve + ssh

- Si tous les utilisateurs svn possèdent déjà un compte système sur le serveur svn, qui possède également un service ssh opérationnel => peut être une solution rapide …

- Utilisation :
  whoami
  durand
  svn list svn+ssh://lpsc.in2p3.fr/projet1
  durand@lpsc.in2p3.fr's password: xxxxx
  ≈ ssh lpsc.in2p3.fr => svnserve –t <= svn list file://projet1

- Pas de démon svnserve qui tourne

- Ce n'est pas svn qui authentifie => pas de mise en cache de la passphrase

- Autorisation par appartenance à un groupe (attention à l'umask) + les variables auth-access et auth-access du fichier svnserve.conf

- Il est possible de restreindre l'accès ssh à la seule utilisation de svn

5.6 apache - installation

- Plus de possibilités que svnserve => plus de complexité à installer et configurer

- Apache2
- Le module mod_dav
  installé par défaut dans apache2 mais il faut l'activer :
  LoadModule dav_svn_module modules/mod_dav_svn.so (httpd.conf)
- Subversion
- L'extension mod_dav_svn (contenu le package de subversion)
  LoadModule dav_svn_module modules/mod_dav_svn.so (httpd.conf)

- Configurer httpd.conf pour donner l'accès au repository
  <Location /repo> => http://hostname/repo
  DAV svn
  SVNPath /usr/local/repository
  </Location>
  ou
  <Location /svn> => http://hostname/svn/projet1
  DAV svn (pas besoin de redémarrer le serveur web pour ajouter un repository)
  SVNParentPath /usr/local/repository
  /Location>

5.6 apache - authentification

⇒ Votre repository est alors anonymement accessible en rw !

- Authentification HTTP, en http ou https :

```
<Location /svn>
        DAV svn
        SVNParentPath /usr/local/repository

        AuthType Basic
        AuthName ”Subversion repository”
        AuthUserFile /etc/svn-auth-file
        Require valid-user
 </Location>
```

- Authentification par certificat, en https :

```
<Location /svn>
        DAV svn
        SVNParentPath /usr/local/repository

        SSLRequireSSL
        SSLVerifyClient require
        SSLVerifyDepth 3
        SSLOptions +StrictRequire
        SSLUserName SSL_CLIENT_S_DN
 </Location>
```

- …

5.6 apache - autorisations

- LoadModule authz_svn_module modules/mod_authz_svn.so

- <Location /repos>
  DAV svn
  SVNParentPath /usr/local/svn
  …
  AuthzSVNAccessFile /usr/local/SVN_auth
  </Location>

- Contenu du fichier :
  [groups]
  melot= /C=FR/O=CNRS/OU=UMR5821/CN=Frederic Melot/emailAddress=melot@in2p3.fr
  theo=@melot, durand, dupont

  [repo1:/]
  *=r
  durand=rw
  melot=

  [repo1:/prive]
  *=
  durand=rw

  [repo2:/]
  @theo=rw

- => va dégrader les performances du serveur !

PLAN

6 Quelques mots sur l'administration d'un serveur svn

- 6.1 Hooks
- 6.2 Maintenance de repository
  - 6.2.1 Le programme svnlook
  - 6.2.2 Le programme svnadmin
  - 6.2.3 Le programme svndumpfilter
    - Migration de repository
  - 6.2.4 Le programme svnsync
  - 6.2.5 Backup de repository

6.1 Hooks

- Un hook est un programme déclenché par un événement sur le repository. Ils sont localisés dans le répertoire hooks du repository
- Il en existe deux types :
  - Les pre-hooks : ils sont appelés avant un événement et permettent :
    - De rapporter l'événement qui va arriver
    - Et / ou de l'empêcher en fonction de conditions
  - Les post-hooks : ils sont appelés après la fin de l'événement
    - Ils exécutent des tâches qui examinent le repository mais ne le modifient pas
- Chaque hook appelé reçoit des paramètres sur la nature de l'événement, les modifications et son l'auteur
- Le hook pre-revprop-change est un peu spécial : il doit exister pour permettre aux utilisateurs de changer les propriétés des révisions.
  Par défaut il n'existe pas !

6.1 Hooks

- Par défaut dans le répertoire hooks, on trouve un fichier template pour chaque hook prédéfini : pre-commit.tmpl, pre-lock.tmpl, pre-revprop-change.tmpl, pre-unlock.tmpl, start-commit.tmpl, post-commit.tmpl, post-lock.tmpl, post-revprop-change.tmpl, post-unlock.tmpl et start-commit.tmpl

- Ces fichiers template sont très utile car ils contiennent toute l'information utile pour écrire les hooks (liste des paramètres passés, …)

- Pour créer un hook,
  - placez un fichier dans le répertoire hooks, avec un des noms prédéfini
  - Rendez-le exécutable (droit x pour Linux, extension .exe pour Windows)
  - Le programme peut être écrit dans n'importe quel langage
  - Le hook peut contenir l'action à exécuter ou appeler d'autres programmes
  - Des scripts tout prêts existent déjà dans subversion (en python et en perl) dans /xxx/subversion/tools/hook-script
  - L'utilisateur qui accède au repository doit avoir le droit d'exécution sur le hook
  - Les hooks sont exécutés dans un environnement vide (attention aux PATH)
  - Ne jamais modifier une transaction dans un hook !

6.2 Maintenance de repository

- Une fois le repository créé et correctement paramétré, il y a très peu de maintenance
- La maintenance du repository s'effectue principalement à l'aide de 2 commandes :
  - *svnadmin* pour la plupart des opérations de modification du repository
  - *svnlook* pour l'examiner sans le modifier
    - faire des diagnostics
    - agit sur des révisions (--revision) ou des transactions (--transaction), par défaut sur la HEAD révision
  - Mais aussi *svndumpfilter* et *svnsync*
- Les utiliser sur le serveur lui-même et pour spécifier le repository utiliser un chemin du système de fichiers local et surtout pas une url (sauf pour svnsync)

6.2.1 Le programme svnlook

- *svnlook author /xxx/…*
- *svnlook cat /xxx/…*
- *svnlook changed /xxx/…*
- *svnlook date /xxx/…*
- *svnlook diff /xxx/…*
- *svnlook info /xxx/…*
- *svnlook log /xxx/…*
- *svnlook propget /xxx/…*
- *svnlook proplist /xxx/…*
- *svnlook tree /xxx/…*
- *svnlook youngest /xxx/…*
- *…*

6.2.2 Le programme svnadmin

- *svnadmin create* => créer un repository
- *svnadmin dump* => faire une sauvegarde du contenu d'un repository (dans un format de dump portable et en mode texte)
  - Dans un but de sauvegarde
  - Pour déplacer tout ou partie d'un repository vers un autre
  - Pour supprimer des éléments d'un repository (voir plus loin)
  - Pour certains changements majeurs de version de subversion
  - Passer d'une Berkeley DB vers un FSFS
  - Dans un dump chaque révision contient les informations minimales pour être reconstruite à partir des révisions précédentes => que les fichiers concernés et leur delta, sauf la première révision du dump
  - *svnadmin dump -r 1000:2000 > dump-r1000:2000*
    => la révision 1000 contient l'entièreté des fichiers
  - *svnadmin dump --incremental -r 1000:2000 > dump-r1000:2000*
    => la révision 1000 contient le delta depuis la révision 999

6.2.2 Le programme svnadmin

- *svnadmin load* => charge une sauvegarde (fichier dump) et crée une nouvelle révision
  - ne déclenche pas les hook, sauf si spécifié :
    --use-pre-commit-hook ou –use-post-commit-hook mais lance le hook pour chaque révision chargée !
  - On peut effectuer le chargement dans un répertoire défini (et existant) avec l'option --parent-dir
- *svnadmin hotcopy* => fait une sauvegarde du repository complet (données, hooks, …)
- *svnadmin verify* => vérifie l'intégrité du contenu d'un repository (checksum, …)
- *svnadmin lslocks* => affiche tous les verrous du repository
- *svnadmin rmlocks* => permet de supprimer des verrous du repository

6.2.2 Le programme svnadmin

- *svnadmin lstxns*
  => liste le nom des transactions courantes => celles qui ont échouées ! (ne se sont pas transformées en révision)
  pour regarder leur contenu, qui l'a fait, … : *svnlook --transaction …*

- *svnadmin rmtxns* => détruit des transactions

- *svnadmin recover* => équivalent du svn cleanup mais pour le repository

- *svnadmin setlog /xxx/… file*
  => remplace le contenu du log d'une révision (propriété svn:log)
  car par défaut les utilisateurs ne peuvent pas le faire (voir le hook pre-revprop-change)
  Utiliser l'option --bypass-hooks avec précaution car il n'y a plus l'exécution des hooks (on peut perdre par exemple l'envoi de mail)

6.2.3 Le programme svndumpfilter

- Subversion sauvegarde toutes les données dans un format non lisible et donc inchangeable manuellement

  => 2 soucis :
  - Éliminer un fichier non désiré
  - Migrer une partie d'un repository à un autre (et non l'entièreté)

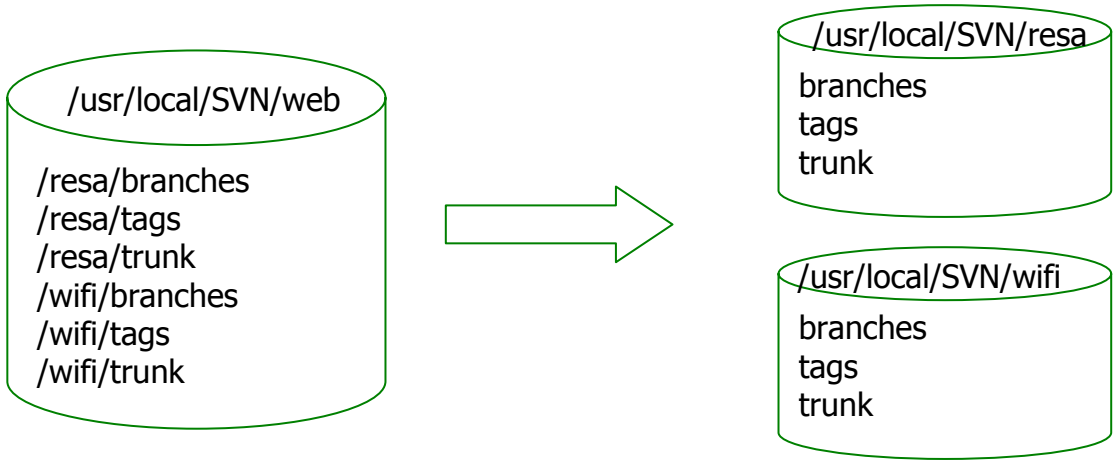
- Les fichiers de dump sont humainement lisibles => il est possible de les modifier manuellement mais
  - Attention à ce que votre éditeur ne convertisse pas les fins de lignes en un format natif => fichier dump inutilisable
  - s'ils sont trop gros …
  - => préférer svndumpfilter

6.2.3 Le programme svndumpfilter

- *svndumpfilter* permet de manipuler un fichier de dump sur la base de chemins de répertoires ou de fichiers
  - En donnant la liste des répertoires / fichiers à garder
    => commande *svndumpfilter include*
  - En donnant la liste des répertoires / fichiers à exlure
    => commande *svndumpfilter exclude*

- Options disponibles :
  - *--drop-empty-revs* : ne génère pas les révisions vides
  - *--renumber-revs* : si les révisions vides ne sont pas générées, change les numéros de révision afin d'éviter les 'trous'
  - *--preserve-revprops* : si les numéros de révisions vides sont conservés, conserver également leurs propriétés (sinon timestamp + commentaire générique)

6.2.3 Exemple d'utilisation : Migration de repository



- svnadmin dump /usr/local/SVN/web > repos-web

- cat repos-web | svndumpfilter include resa > repos-resa
- svnadmin create /usr/local/SVN/resa
- svnadmin load /usr/local/SVN/resa < repos-resa

- svndumpfilter include wifi < repos-web > repos-wifi
- svnadmin create /usr/local/SVN/wifi
- svnadmin load /usr/local/SVN/wifi < repos-wifi

- La hiérarchie de fichiers est conservée => vous pouvez vouloir supprimer le répertoire principal :
  - Manuellement dans le dump : champ Node-path à changer
  - Dans une working copy à l'aide de svn move (s) => solution à préférer

6.2.4 Le programme svnsync

- Svnsync (pour subversion 1.4 minimum) permet de maintenir une copie d'un repository, à l'aide de commandes de synchronisation
- La source et la destination peuvent être sur des machines séparées
- La synchronisation ne peut s'effectuer que sur l'entièreté d'un repository
- Le repository de synchronisation ne doit pas être modifié par d'autres commandes que celles de synchronisation
- Opérations préliminaires :
  - svnsync doit avoir accès en lecture au repository d'origine
  - svnsync doit avoir accès en lecture/écriture au repository de destination
  - Le repository de destination doit être accessible en lecture seul aux utilisateurs
  - svnsync a besoin de modifier les propriétés de révision du repository de destination => y placé le hook pre-revprop-change

6.2.4 Le programme svnsync

- svnadmin create /usr/local/SVN/mirror
- Créer pre-revprop-change et start-commit pour permettre à l'utilisateur du svnsync et lui seul d'effectuer des modifications
- svnsync init https://xxx/mirror https://xxx/web --username usersync --password userpassword
  le repository de destination sera maintenant lié à celui d'origine (plus besoin de le spécifier)
- svnsync synchronize https://xxx/mirror --username usersync --password userpassword
- svnsync copy-revprops https://xxx/mirror 143 --username usersync --password userpassword
  => permet de repliquer à nouveau une révision déjà synchronisée (si le log a été changé par exemple)
- svnsync synchronize peut être ajouté dans le post-commit du repository source

6.2.5 Backup de repository

- 3 solutions :
  - ~~Recopie de répertoire~~
  - svnadmin hot-copy
  - svnadmin dump
  - svnsync

  - Toutes ont des avantages et des inconvénients :
    - Full ou incrémental
    - Que les données ou tout le repository
    - Modifications des propriétés de révision prises en compte ?
    - Temps de restauration et de changement de serveur
    - Place sur les disques

PLAN

Utilisation d'interface graphique

- TkSVN
  - Utilisable sous Linux, Windows et Mac/OS
  - Basé sur Tcl/Tk
- TortoiseSVN
  - Pour Windows
- TRAC
- …
- Un grand avantage des GUI : la visualisation graphique du résultat d'un diff
  - avec TkSVN : TkDiff
  - il existe des logiciels permettant de visualiser des différences entre fichiers word, excel, …
- Mais les interfaces graphiques ne permettent pas de tout faire
  - toutes les options des commandes ne sont généralement pas implémentées
  - bugs

Conclusion

- Documentation disponible :
  - http://subversion.tigris.org et http://svnbook.red-bean.com
  - http://svn.haxx.se/users
  - http://www.svnforum.org
  - Groupe de discussion sur http://lpsc.in2p3.fr/elog/SVN
- Aucun code ne devrait être développé sans un outil de gestion de versions
- Si on est tout seul, l'utilisation en est simplifié => jamais d'update nécessaire, ni de conflit mais on peut créer des branches
- SVN est censé vous faire gagner du temps, pas en perdre
- Issu d'un cours SVN dispensé par la formation permanente de la DR11
  - Pris les définitions importantes
  - Tout n'y est pas – certaines notions resteront floues
- Questions