



Licence Sciences, Technologies, Santé
Mention SPI, parcours Informatique

Introduction au Génie Logiciel
L3 / 175EN002

C7 – Outils de développement

Thierry Lemeunier
thierry.lemeunier@univ-lemans.fr
www-ic2.univ-lemans.fr/~lemeunie

Plan du cours

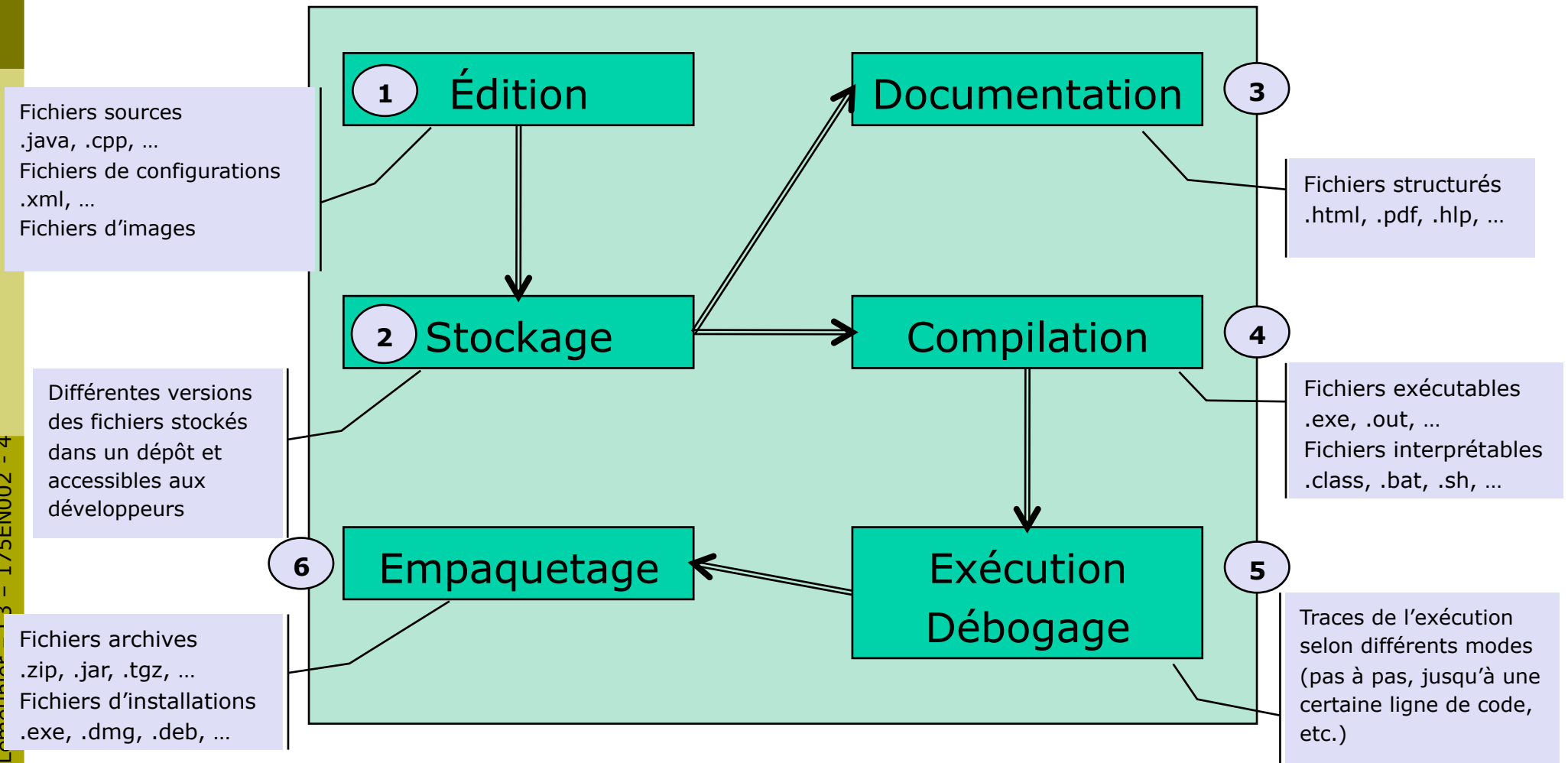
- La chaîne de développement
- L'édition
- La documentation
- La compilation
- Le débogage
- La gestion de version

La chaîne de développement (1 / 5)

- ❑ Chaque phase du processus technique nécessite l'usage d'outils spécifiques ou génériques :
 - Avant-projet :
 - ❑ Etablir le contrat et rédiger le cahier des charges : éditeur de texte (OpenOffice, ...)
 - Initialisation :
 - ❑ Planifier les tâches, affecter le personnel, calculer les coûts : éditeur de graphes PERT et de diagrammes de Gantt (GanttProject, OpenProj, ...)
 - ❑ Rédiger plan de développement : éditeur de texte (OpenOffice, ...)
 - Le développement (spécification/conception/réalisation/maintenance) :
 - ❑ Ecrire des modèles : outils de modélisation (Astah, Modelio, ...)
 - ❑ Ecrire du code documenté : outils de développement (isolés ou intégrés)
 - Editeur de code (emacs, ...)
 - Générateur de documentation (javadoc, doxygen, ...)
 - Stockage avec gestion de version (cvs, svn, git, ...)
 - Compilateurs (gcc pour le langage C, ...)
 - Outil de correction d'erreurs (les débogueurs)
 - Outil de création de tests (JUnit en Java, ...)
 - Environnement de développement intégré : intégration d'outils pour assurer la chaîne de développement

La chaîne de développement (2/5)

Environnement de développement intégré
(IDE : Integrated Development Environment)



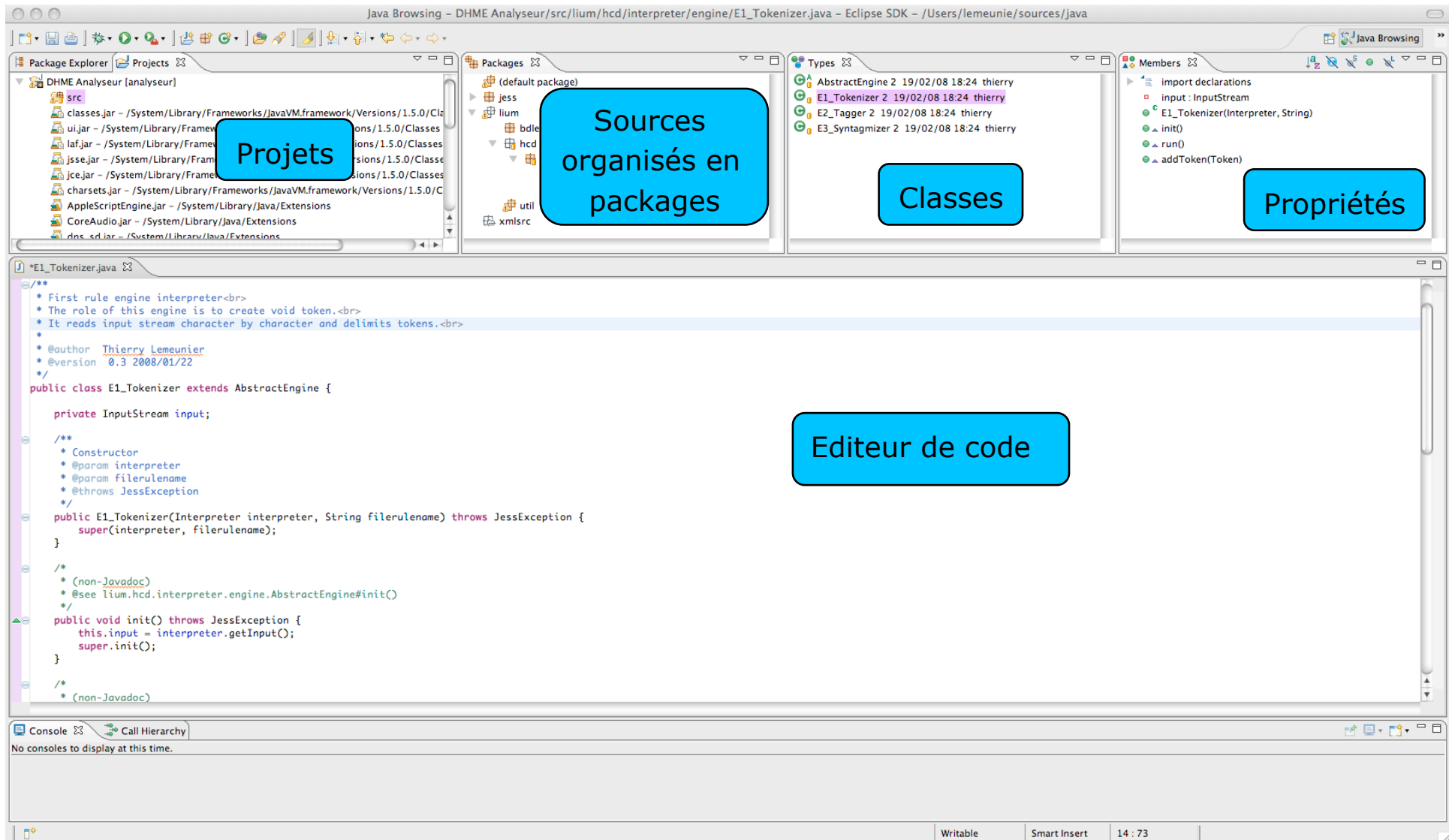
La chaîne de développement (3/5)

- ❑ Utiliser des outils ou développer tout « à la main » ?
 - Développer à la main :
 - ❑ Permet une maîtrise totale de toutes les étapes de développement contrairement à un IDE qui masque les détails
 - ❑ Apprentissage de la chaîne de développement
 - ❑ Quand aucun outil existe (à part un compilateur en ligne de commande)
 - ❑ Quand le projet est relativement simple
 - Développer avec des outils :
 - ❑ Dès que le projet devient plus important ou que l'on travaille à plusieurs, des outils sont nécessaires
 - Exemples :
 - Aide à la compilation : Makefile, ant, ...
 - Gestion et suivi des corrections : Bugzilla, ...
 - Gestion des versions du code source : CVS, SVN, ...

La chaîne de développement (4/5)

- Utiliser un IDE ou développer tout « à la main » ?
 - Utiliser un IDE :
 - Un IDE centralise et rend disponible tous les outils nécessaires
 - Un IDE permet l'automatisation de certaines étapes de la chaîne de développement
 - Par exemple : compilation et déploiement automatique
 - Un IDE permet donc un développement rapide d'application (*RAD : Rapid Application Development*)
 - Répondre rapidement à de nouvelles demandes dans un environnement d'entreprises concurrentielles
 - Aider à maintenir et faire évoluer une application existante
 - La notion de projet : tous les fichiers (sources, fichiers de configurations, ...) nécessaires à la compilation, au déploiement et à l'exécution
 - Exemples :
 - Eclipse (initié par IBM) : basé sur des plug-ins (UML, ...)
 - Netbeans de Oracle : Java version standard (JSE) et Java version entreprise (JEE)
 - Visual Studio de Microsoft : principalement C++ et C#

La chaîne de développement (5/5)



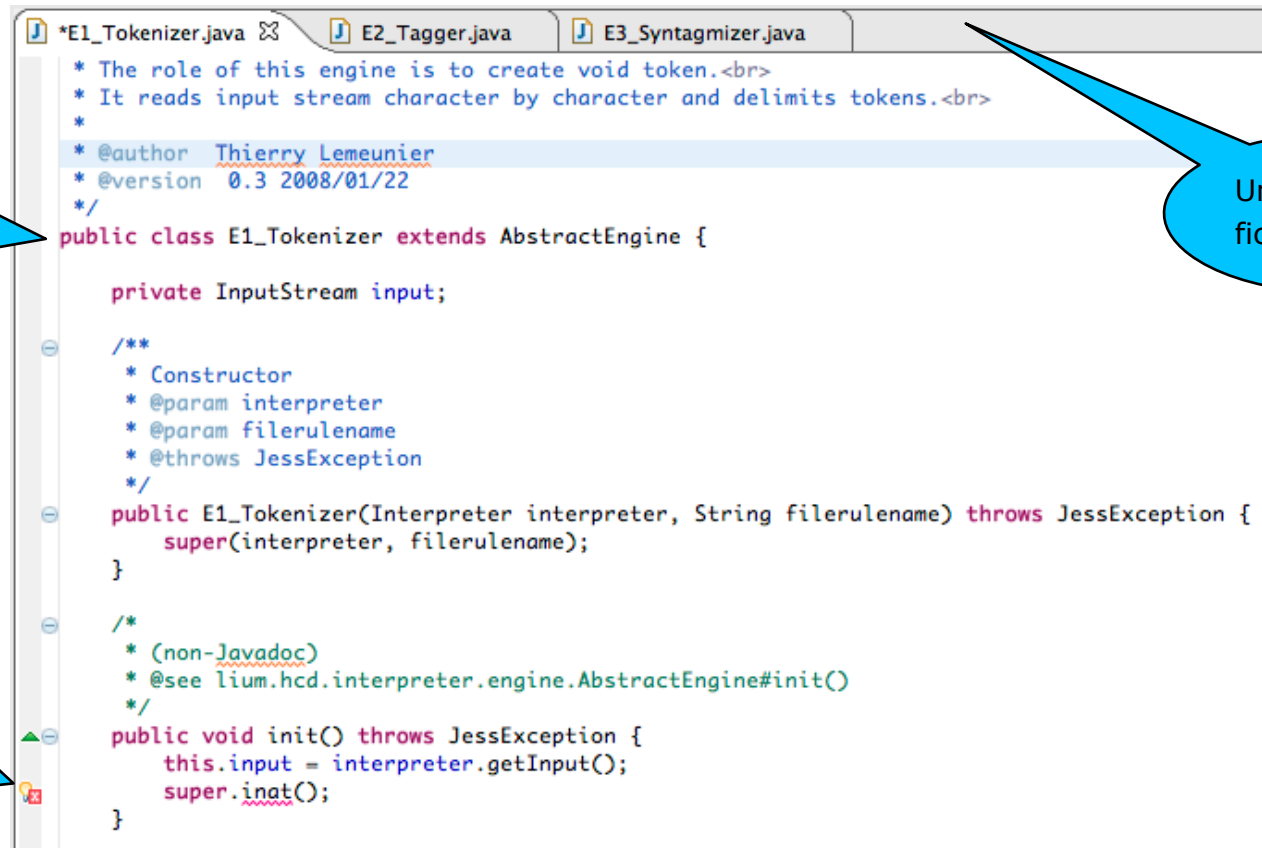
L'édition (1/2)

- ❑ Fonctionnalités de base d'un éditeur :
 - Création d'un fichier source
 - Modification du fichier source
 - Enregistrement dans un format texte universel (sans sauvegarde de mise en forme) encodé selon un certain codage des caractères (ASCII, UTF8, ISO-8859-1, ...)
- ❑ Fonctionnalités avancées d'un éditeur
 - Aide à la lisibilité
 - ❑ Indentation des lignes
 - ❑ Colorisation syntaxique (commentaires, mots-clés du langage, etc.)
 - ❑ Plusieurs vues disponibles, par exemples :
 - Vue pour le codage
 - Vue pour le débogage
 - Vue pour le dépôt des fichiers sources
 - Aide au développement
 - ❑ Compilation automatique à la volée
 - ❑ Matérialisation des erreurs par signe ou colorisation particulier
 - ❑ Auto-complétion : complétion du code en cours d'édition avec proposition de plusieurs possibilités
 - ❑ *Refactoring*, par exemples :
 - Aide à l'ajout de méthodes d'une classe (constructeurs, getters/setters, ...)
 - Aide à la création d'une nouvelle classe
 - Application/utilisation d'un patron de conception (*design pattern*)
 - ❑ Empaquetage : création de bibliothèques archivés
 - ❑ Création de la documentation (par exemple accès à javadoc sous Eclipse)
 - ❑ Aide à la recherche de fonctions/méthodes

L'édition (2/2)

■ Exemples d'éditeurs :

- Editeurs historiques sous Unix : vi, Emacs
- Editeurs plus ou moins complet (gestion de différents encodage, coloration, accès au compilateur)
 - Par exemples : TextMate (sous Mac), NotePad++ (sous Windows)
- Editeur intégré dans Eclipse :



```
*E1_Tokenizer.java  E2_Tagger.java  E3_Syntagmizer.java
* The role of this engine is to create void token.<br>
* It reads input stream character by character and delimits tokens.<br>
*
* @author  Thierry Lemeunier
* @version 0.3 2008/01/22
*/
public class E1_Tokenizer extends AbstractEngine {

    private InputStream input;

    /**
     * Constructor
     * @param interpreter
     * @param filerulename
     * @throws JessException
     */
    public E1_Tokenizer(Interpreter interpreter, String filerulename) throws JessException {
        super(interpreter, filerulename);
    }

    /**
     * (non-Javadoc)
     * @see lium.hcd.interpreter.engine.AbstractEngine#init()
     */
    public void init() throws JessException {
        this.input = interpreter.getInput();
        super.inat();
    }
}
```

Couleur
selon la
syntaxe du
langage

Un onglet par
fichier ouvert

Indication
d'une
erreur avec
aide pour la
résoudre

La documentation (1/2)

- ❑ Fonctionnalités des outils de documentation :
 - Avoir accès à la complétion de code durant l'édition
 - Avoir accès à la documentation (utilisation, explication) du code programmé par d'autres personnes :
 - ❑ Liste des fonctions/classes/méthodes utilisables : l'API (*Application Programming Interface*)
 - ❑ Explication sommaire de chaque fonctions/classes/méthodes de l'API
 - Génération automatique des API
- ❑ Exemples de documentations :
 - API : description pour utiliser le code mis à disposition
 - HOWTO : description de la manière de résoudre un problème spécifique.
 - FAQ (Foire Aux Questions, Frequently Asked Questions) : réponses aux questions courantes.
 - README : description générale à lire en premier
- ❑ Exemples d'outils :
 - Javadoc : outil du JDK pour générer les API Java (fichiers HTML) à partir des commentaires et des « *tags* » des fichiers sources .java
 - Doxygen : même principe mais indépendant du langage

La documentation (2/2)

Tags pour une classe

Tags pour une méthode

Commentaires :
- Pour chaque élément (classe, méthode, attribut)
- Formatage HTML
- Tags

```
package lium.hcd.interpreter.engine;

+import jess.JessException;

-/**
 *
 * A rule engine that tries to tag token.<br>
 *
 * @author Thierry Lemeunier
 * @version 0.3 2008/01/28
 */
public class E3_Syntagmizer extends AbstractEngine {

    /**
     * Constructor<br>
     * This method creates a new instance of E3_Syntagmizer and initializes it.<br>
     *
     * @param interpreter
     * @param filerulename
     * @throws JessException
     */
    public E3_Syntagmizer(Interpreter interpreter, String filerulename) throws JessException
    {
        super(interpreter, filerulename);
    }
}
```

Problems @ Javadoc Declaration

lium.hcd.interpreter.engine.E3_Syntagmizer.E3_Syntagmizer(Interpreter interpreter, String filerulename) throws JessException

Constructor
This method creates a new instance of E3_Syntagmizer and initializes it.

Parameters:
 interpreter
 filerulename

Throws:
 JessException

Javadoc qui sera générée

La compilation (1 / 3)

- Fonctionnalités d'un outil de compilation :
 - Génération d'un fichier exécutable par le système d'exploitation ou génération d'un fichier intermédiaire compréhensible pour un interpréteur
 - Détection des erreurs syntaxiques
- Exemples de compilateurs :
 - cc, gcc : compilateur de fichiers sources en langage C en exécutables systèmes
 - g++ : compilateur de fichiers sources en langage C++ en exécutables systèmes
 - ld : éditeur de liens entre fichiers intermédiaires et bibliothèques
 - javac : compilateur de fichiers sources en langage Java en classes interprétables par une machine virtuelle java (*JVM*)

La compilation (2/3)

□ Outils d'aide à la compilation :

■ Fonctionnalités :

□ Automatisation de la compilation

■ Par exemple pour le langage C :

1. Compilation séparée des fichiers sources
2. Edition des liens entre librairies et fichiers compilés
3. Création du fichier exécutable

□ Gestion des dépendances entre fichiers sources : permet une compilation « intelligente »

□ Gestion des chemins d'accès aux fichiers sources et aux librairies

■ Exemples :

- make, gmake : outils en ligne de contrôle de compilation qui lit des *makefiles* et exécute les directives
- ant : outil écrit en Java d'automatisation de tâches décrites dans un fichier build.xml

La compilation (3/3)

Extrait d'un makefile

```
OBJS = agenda.o analysis.o argaccs.o bload.o bmathfun.o bsave.o \  
      cardinaux.tab.o cardinaux.yy.o \  
      ...
```

Déclaration
d'une
variable

```
.c.o :  
    gcc -c -Wall $<  
  
clips: $(OBJS)  
    gcc -o clips $(OBJS) -lm -ltermcap  
    cp clips ../bin/yade  
  
clean:  
    rm *.o clips
```

Déclaration des directives de compilation :
1 – Comment faire pour transformer des .c en .o
2 – Comment faire pour exécuter la directive clips
3 – Comment faire pour exécuter la directive clean

```
# Dependencies generated using "gcc -MM *.c"
```

```
agenda.o: agenda.c setup.h envrnmnt.h symbol.h usrsetup.h argaccs.h ...
```

```
analysis.o: analysis.c setup.h envrnmnt.h symbol.h usrsetup.h constant.h ...
```

Déclaration des
dépendances entre
fichier sources

Le débogage (1/2)

□ Fonctionnalités de base d'un débogueur :

- Correspondance à l'exécution entre fichiers sources et fichiers exécutables/interprétables
- Récupération de l'état d'exécution du programme
 - Etat de la pile d'exécution : les fonctions/méthodes appelées jusqu'à l'instruction courante
 - Valeur des variables, des attributs, etc.
- Exécution en mode débogage :
 - Exécution pas à pas (instruction par instruction par défaut)
 - Exécution jusqu'à un point d'arrêt
 - Saut de code (boucle, méthode, etc.)
- Affichage des erreurs d'exécution (avec état de la pile d'exécution par exemple)

□ Exemples de débogueurs :

- `gdb` (*GNU project DeBugger*) : débogueur de programmes écrits en C, C++, Pascal, Objective C
- `jdb` : débogueur de programmes écrits en Java

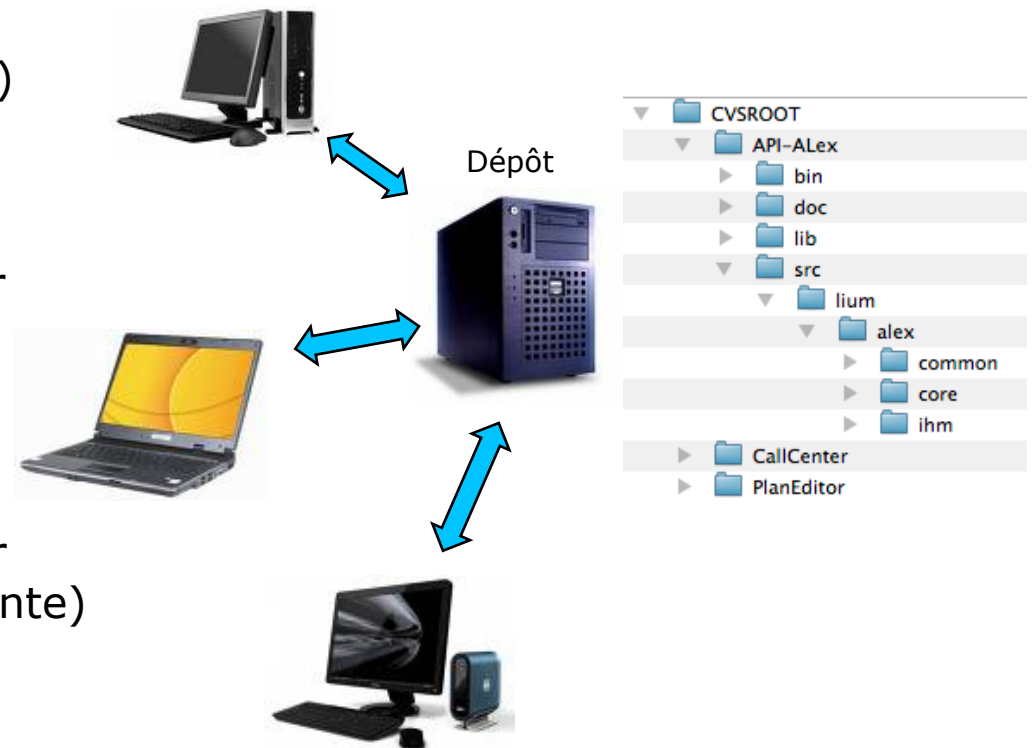
Le débogage (2/2)

The screenshot displays the Eclipse IDE interface during a debug session. The top toolbar includes standard IDE icons and a 'Debug' button. The main workspace is divided into several panels:

- Debug Console (Top Left):** Shows the execution stack. The top entry is 'DiagramAppWindow [Java Application]' with a sub-entry 'lium.diagram.diagramapp.DiagramAppWindow at localhost:50003'. Below this, the 'Thread [main]' is shown as 'Suspended (breakpoint at line 164 in DiagramAppWindow)'. The stack trace includes 'DiagramAppWindow.<init>() line: 164' and 'DiagramAppWindow.main(String[]) line: 681'. Other threads like 'Daemon Thread [AWT-AppKit]' and 'Thread [AWT-Shutdown]' are also listed. A blue box labeled 'Pile d'exécution' points to this panel.
- Variables View (Top Right):** Displays the current state of variables. The 'this' variable is expanded, showing attributes like 'accessibleContext' (null), 'alwaysOnTop' (false), 'appContext' (AppContext (id=35)), and 'background' (CColorPaintUIResource (id=37)). A blue box labeled 'Valeurs des attributs et variables' points to this panel.
- Source Editor (Bottom Left):** Shows the source code of 'DiagramAppWindow.java'. The line 'Login logDialog = new Login(this);' is highlighted, indicating the current line of execution. A blue box labeled 'Code en cours d'interprétation' points to this line.
- Outline View (Bottom Right):** Lists the classes and methods in the project. The 'DiagramAppWindow()' method is highlighted, indicating it is the current method being debugged.
- Console (Bottom):** Displays the output of the application. The text 'DiagramAppWindow [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.5.0/Home/bin/java (17 nov. 08 17:49:49)' is visible.

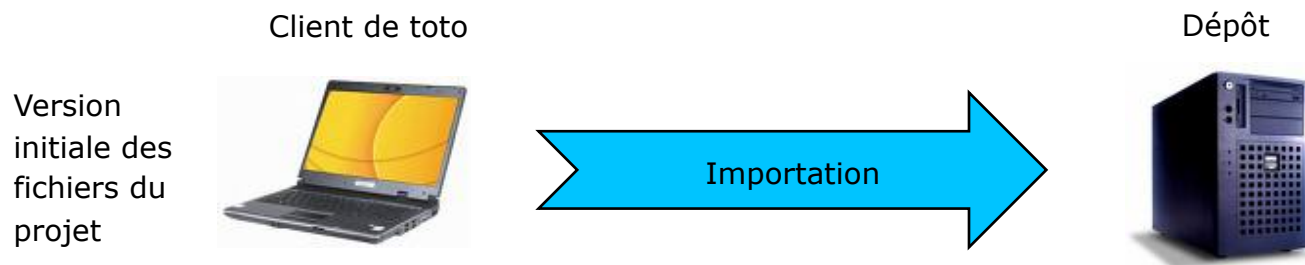
La gestion de version (1/13)

- ❑ Fonctionnalités d'un système de gestion de version :
 - Permanence des fichiers sources
 - Gestion des droits d'accès aux fichiers sources
 - Accès concurrent aux fichiers sources pour travailler à plusieurs sur le projet
 - Gestion de différentes versions d'un fichier source
 - Peut être aussi utilisé pour gérer les versions des documents du projet
- ❑ Exemples d'outils libres :
 - CVS (*Concurrent Versions System*)
 - SVN (*Subversion*)
- ❑ Principes de fonctionnement :
 - Un dépôt sur une machine serveur
 - Des utilisateurs se connectent sur le serveur avec des logiciels clients en ligne de commande ou graphique
 - Remarque : peut aussi fonctionner en local (dépôt sur la machine cliente)



La gestion de version (2/13)

- Les opérations d'un système de gestion de version
 - Opérations d'administration :
 - Création du dépôt
 - Gestion des utilisateurs
 - Création des login, mot de passe et droits de l'utilisateur
 - Associer un répertoire à l'utilisateur (partage de données)
 - Opérations d'utilisation du dépôt :
 - Transfert initial des fichiers du projet vers le dépôt (*repository*)
 - Exemple avec CVS :
`cvs import -m "sources" ./src toto v1`
`cvs import -m message file user tag`

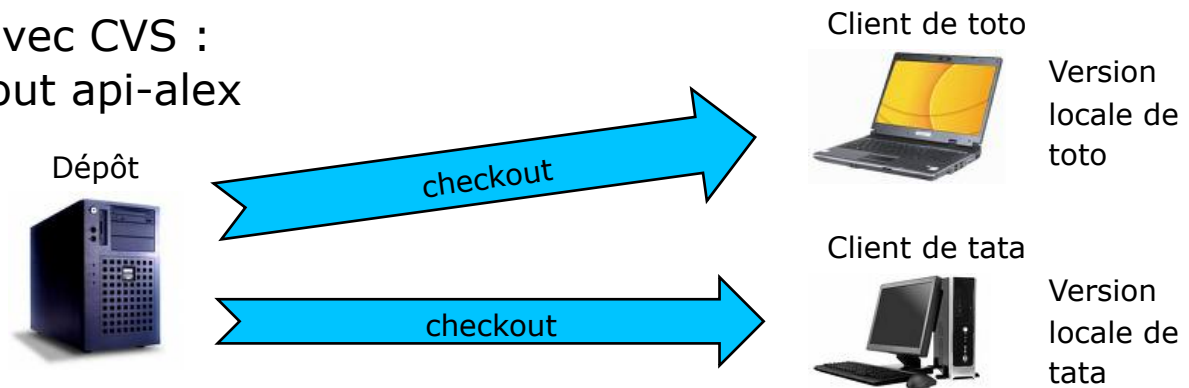


La gestion de version (3/13)

■ Opérations d'utilisation du dépôt (suite) :

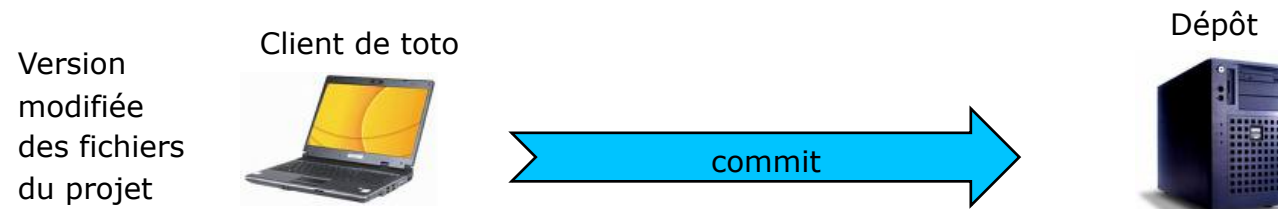
□ Création d'une copie privée locale (*working copy*)

- Copie de la version actuelle des fichiers sur la machine cliente
- L'utilisateur peut ensuite travailler localement sur les fichiers
- Exemple avec CVS :
`cvs checkout api-alex`



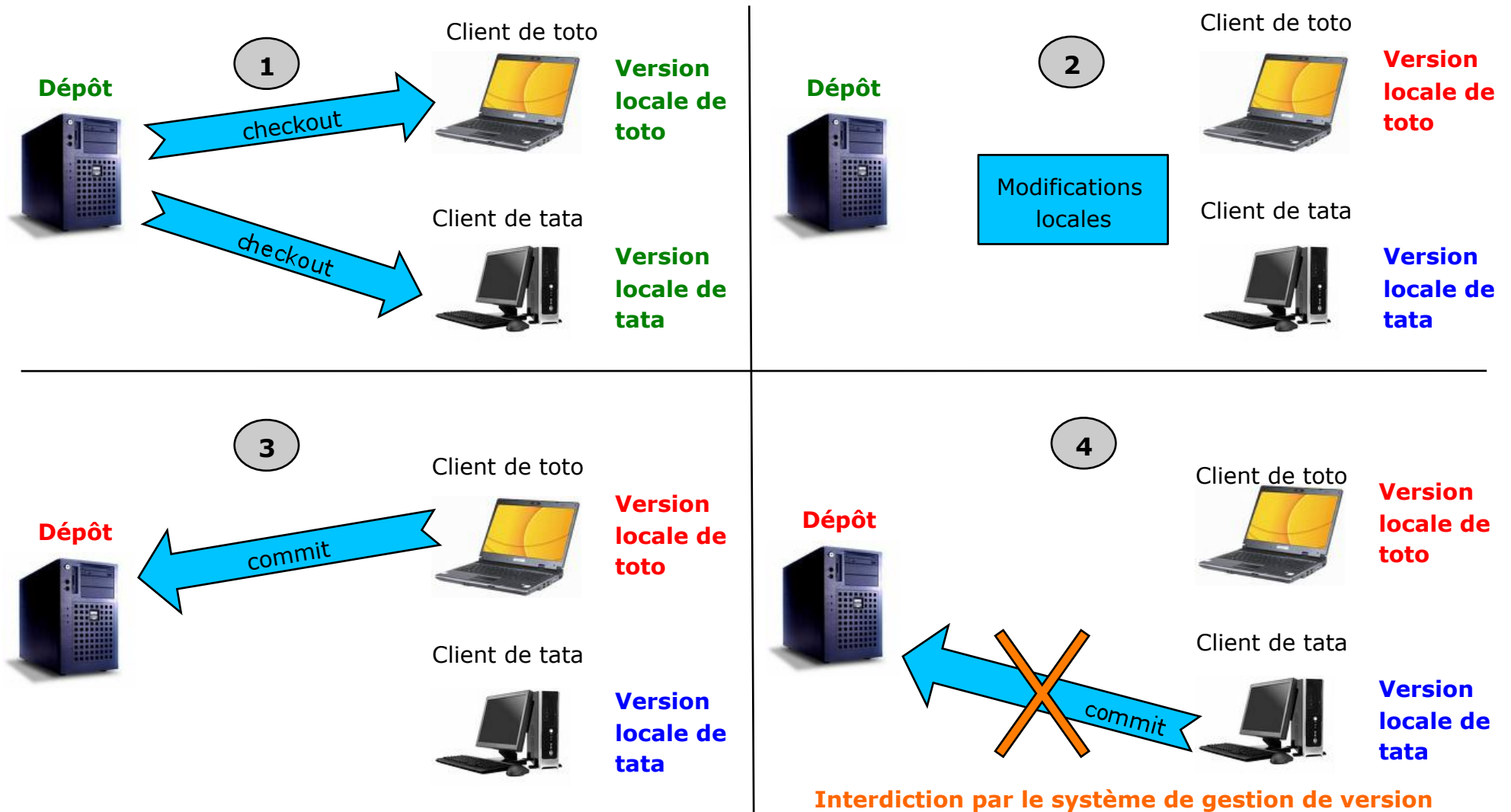
□ Enregistrer les modifications dans le dépôt

- Une nouvelle version du fichier existe alors sur le dépôt
- Exemple avec CVS :
`cvs commit -m "Optimisation de la méthode de tri" tri_externe.c`



La gestion de version (4/13)

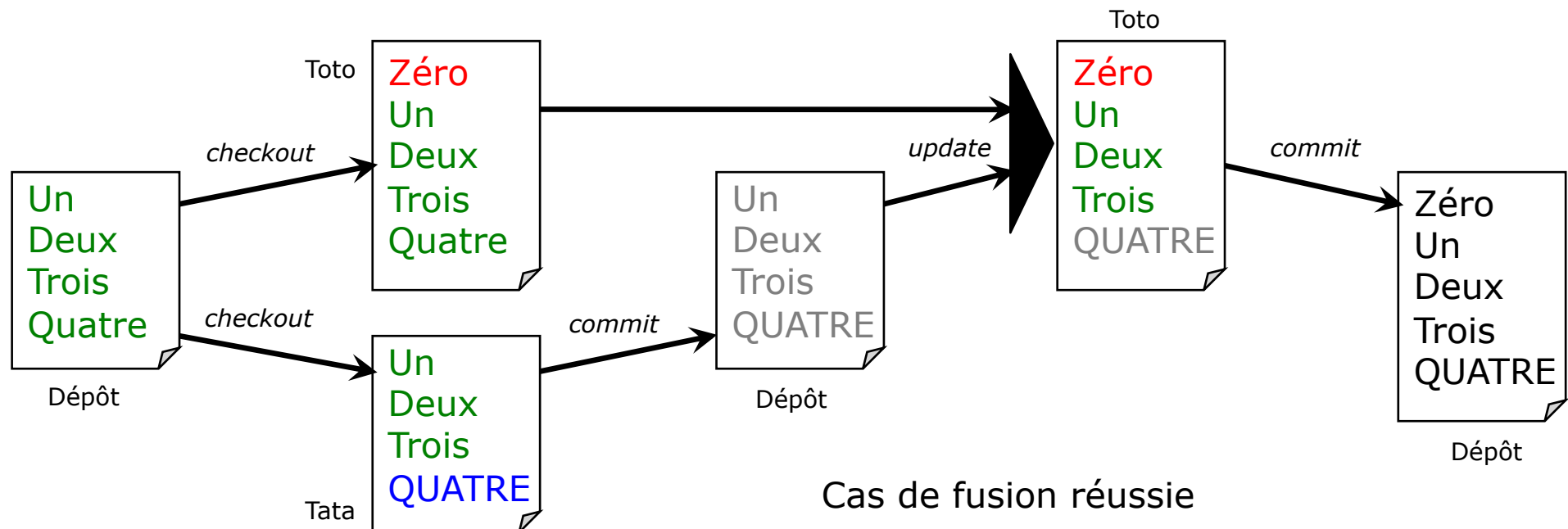
Interdiction du commit après modifications locales simultanées



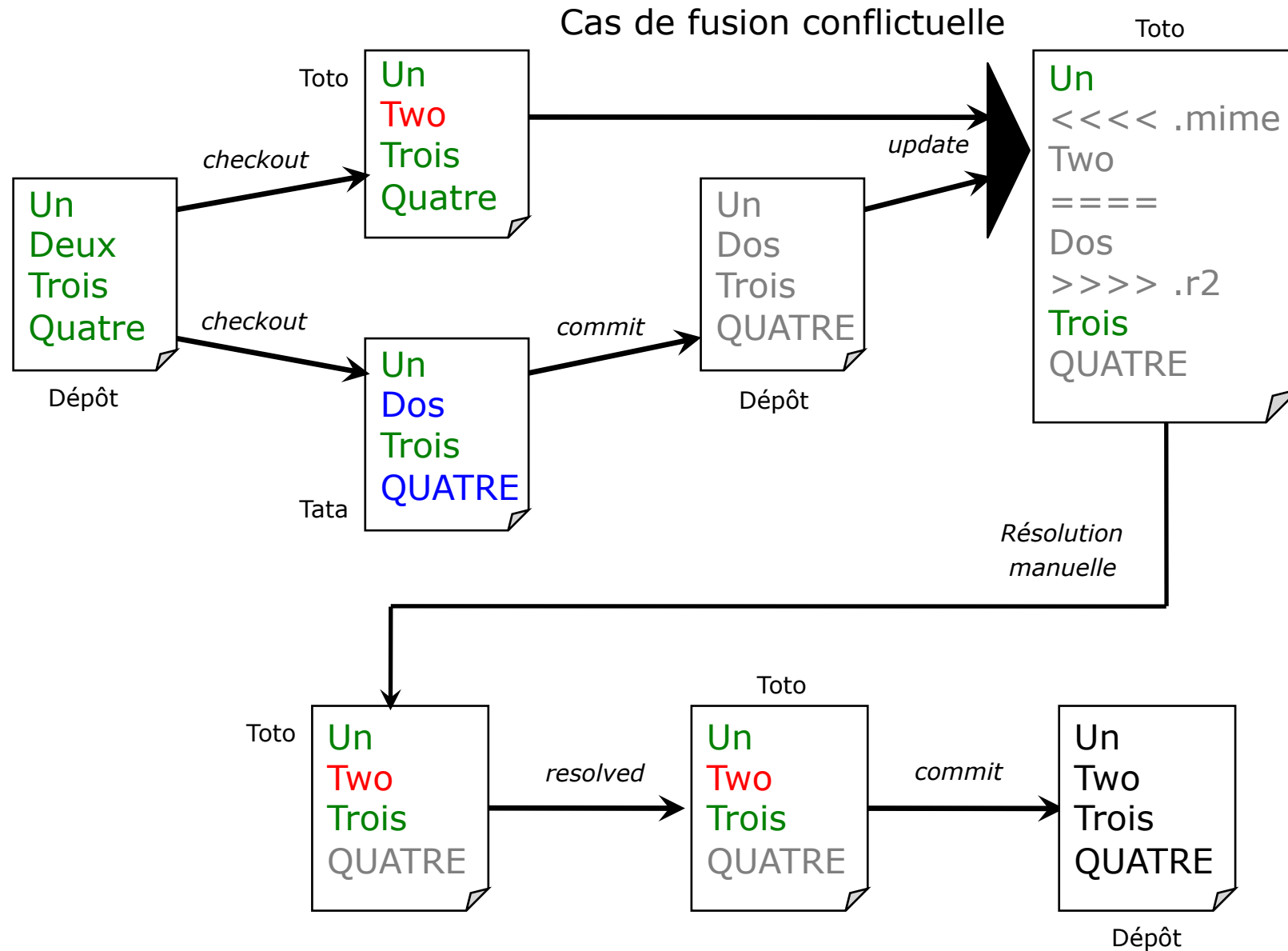
La gestion de version (5/13)

■ Opérations d'utilisation du dépôt (suite) :

- Mettre à jour une copie locale (avant de faire un commit)
 - Le système de gestion de version fusionne localement la copie locale avec la version du fichier du dépôt
 - La fusion marche si les modifications concernent des lignes différentes
 - La fusion échoue pour les même lignes modifiées : l'utilisateur doit alors intervenir manuellement pour résoudre le conflit
 - Exemple avec CVS :
`cvs update tri_externe.c`

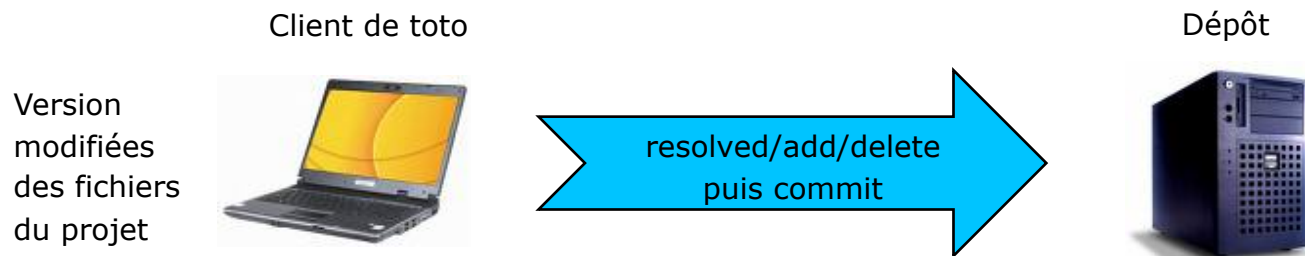


La gestion de version (6/13)



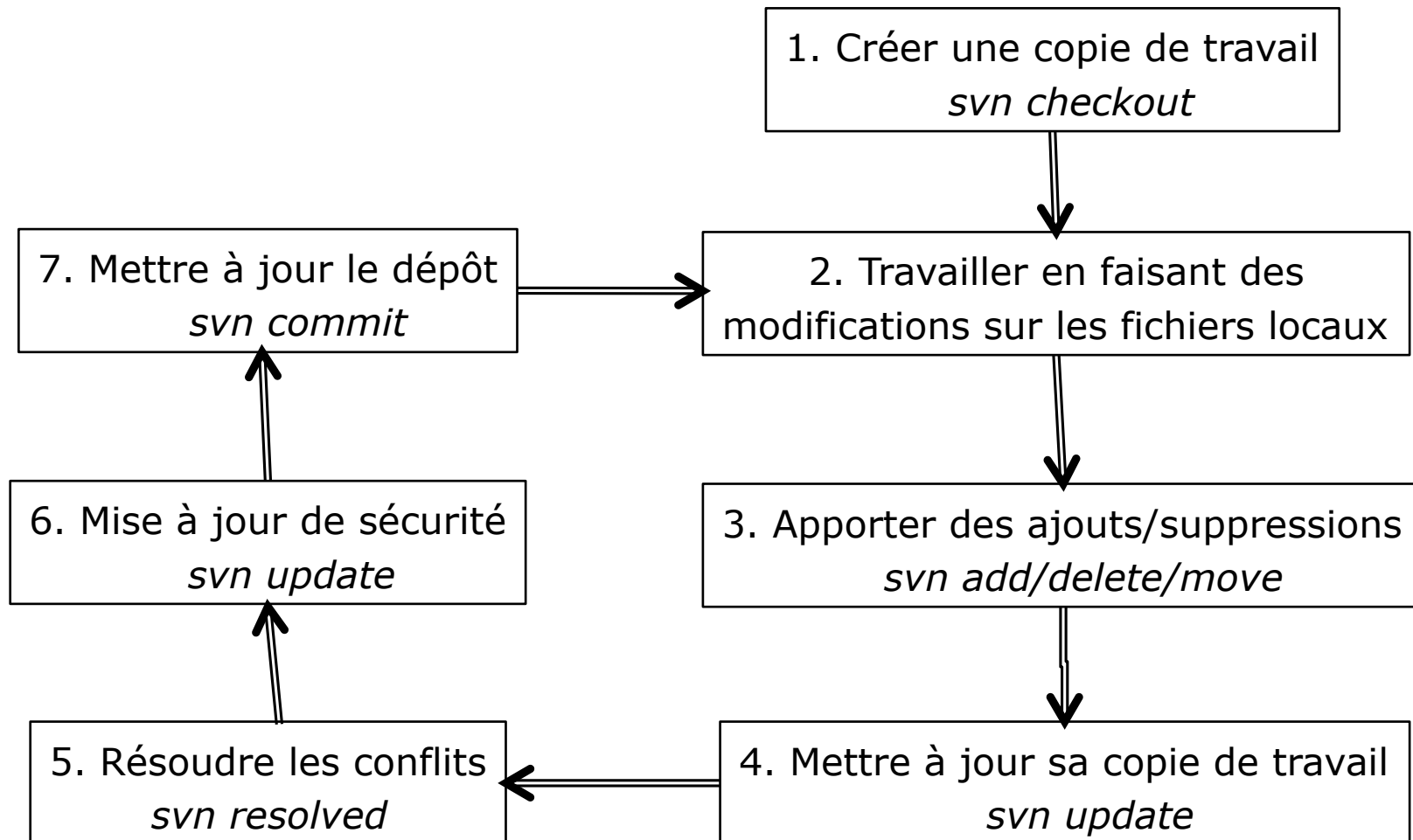
La gestion de version (7/13)

- Opérations d'utilisation du dépôt (suite) :
 - Indiquer la résolution d'un fichier en conflit
 - Il faut faire un commit après !
 - Exemple avec SVN : `svn resolved tri_extern.c`
 - Ajouter au dépôt un fichier ou un répertoire qui a été créé localement
 - Il faut faire un commit après !
 - Exemple avec SVN : `svn add tri_interne.c`
 - Supprimer un fichier ou un répertoire du dépôt (et aussi localement)
 - Il faut faire un commit après !
 - Exemple avec SVN : `svn delete tri.c`



La gestion de version (8/13)

Exemple de cycle de travail avec SVN :

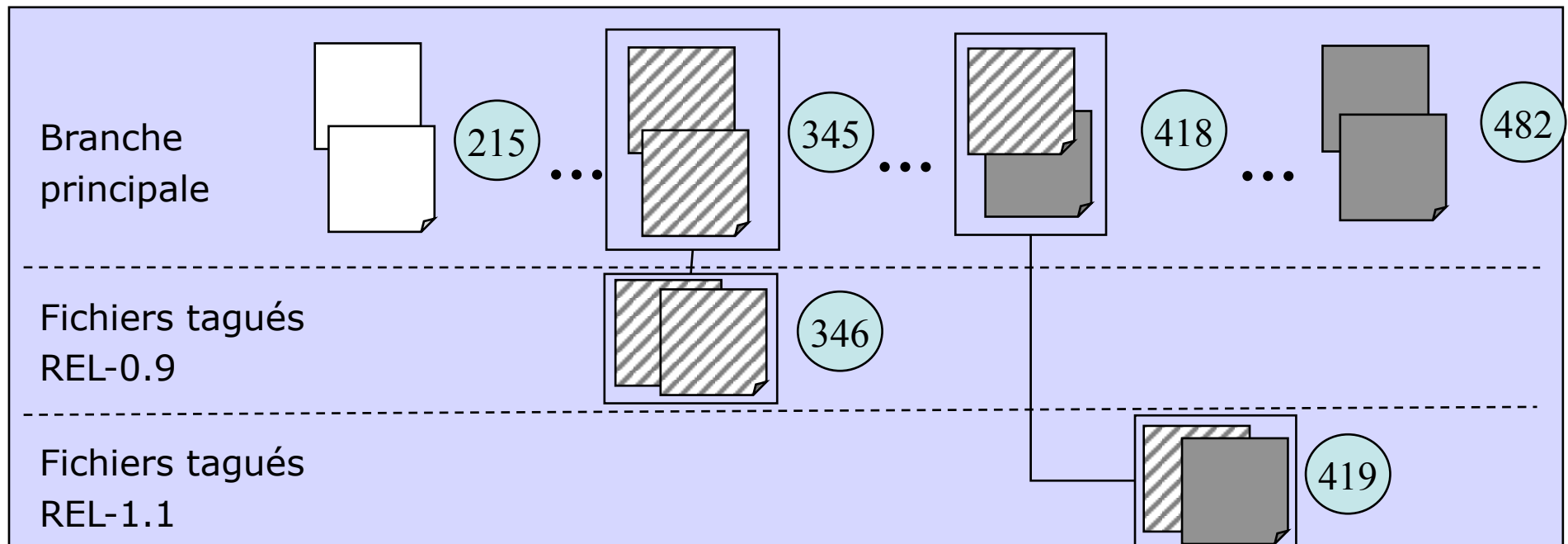


La gestion de version (9/13)

■ Les étiquettes (*tags*)

- Chaque version d'un fichier du dépôt porte un numéro de révision qui est incrémenté automatiquement
- Pour distinguer une version particulière, par exemple une version du projet qui est livrée (une *release*), on marque les fichiers avec un *tag*
- On peut ensuite récupérer les fichiers du *tag*

Dépôt



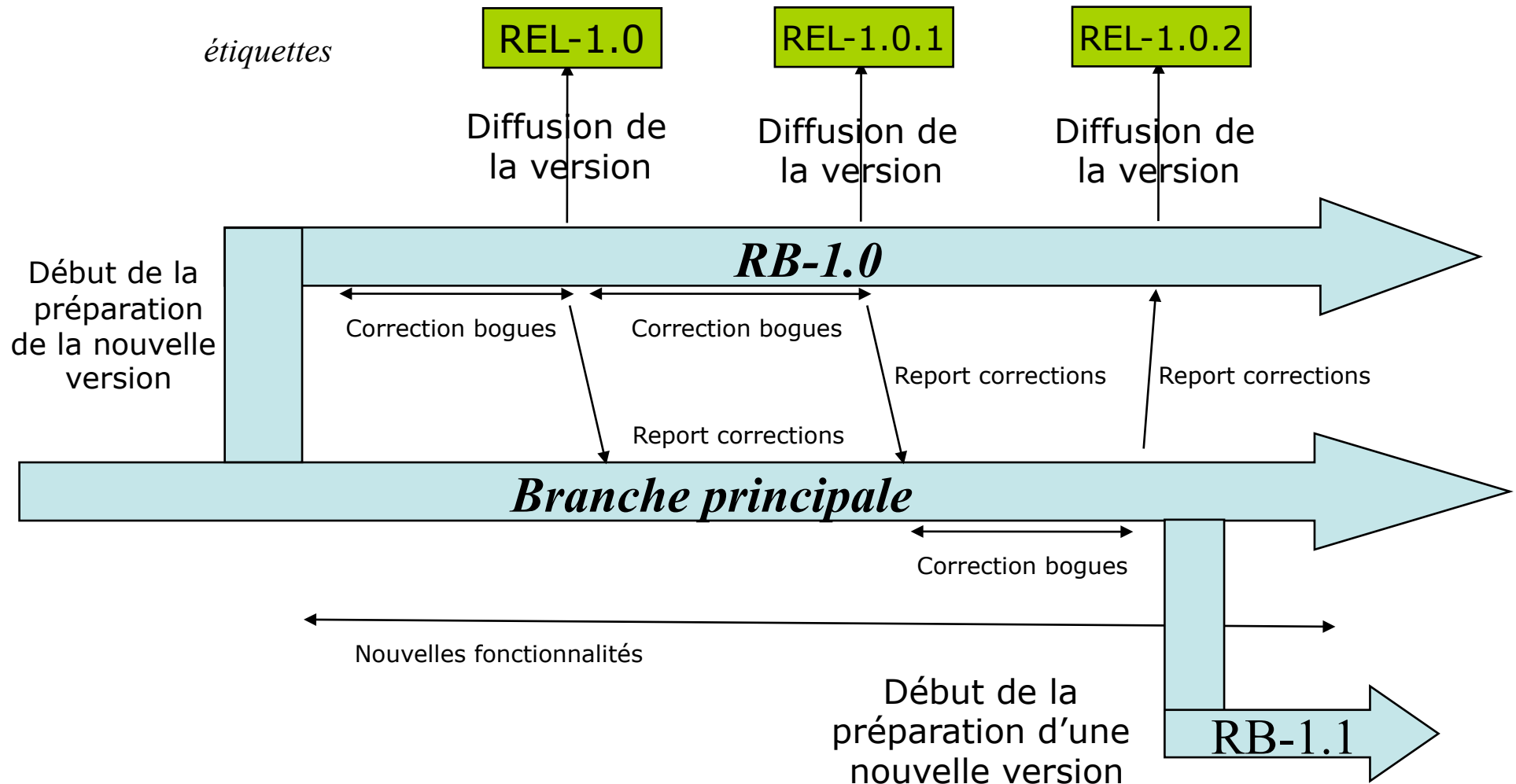
La gestion de version (10/13)

□ Les branches :

- Dans certains cas, il est nécessaire de développer plusieurs branches en parallèle, par exemples :
 - Pour préparer la sortie d'une nouvelle version, on ne travaille plus que sur la correction de bogues mais pendant ce temps d'autres développeurs peuvent vouloir introduire de nouvelles fonctionnalités pour les futures versions
 - Certaines fonctionnalités d'une *release* ont peut-être disparues dans des releases ultérieures mais tant que cette *release* est maintenue, il faut pouvoir corriger les bogues de ces fonctionnalités
- Une branche permet de faire évoluer le projet contrairement à un *tag* qui n'est qu'une étiquette
- Il peut y avoir autant de branches que nécessaires (comme les *tags*)
- On peut faire des fusions de branches

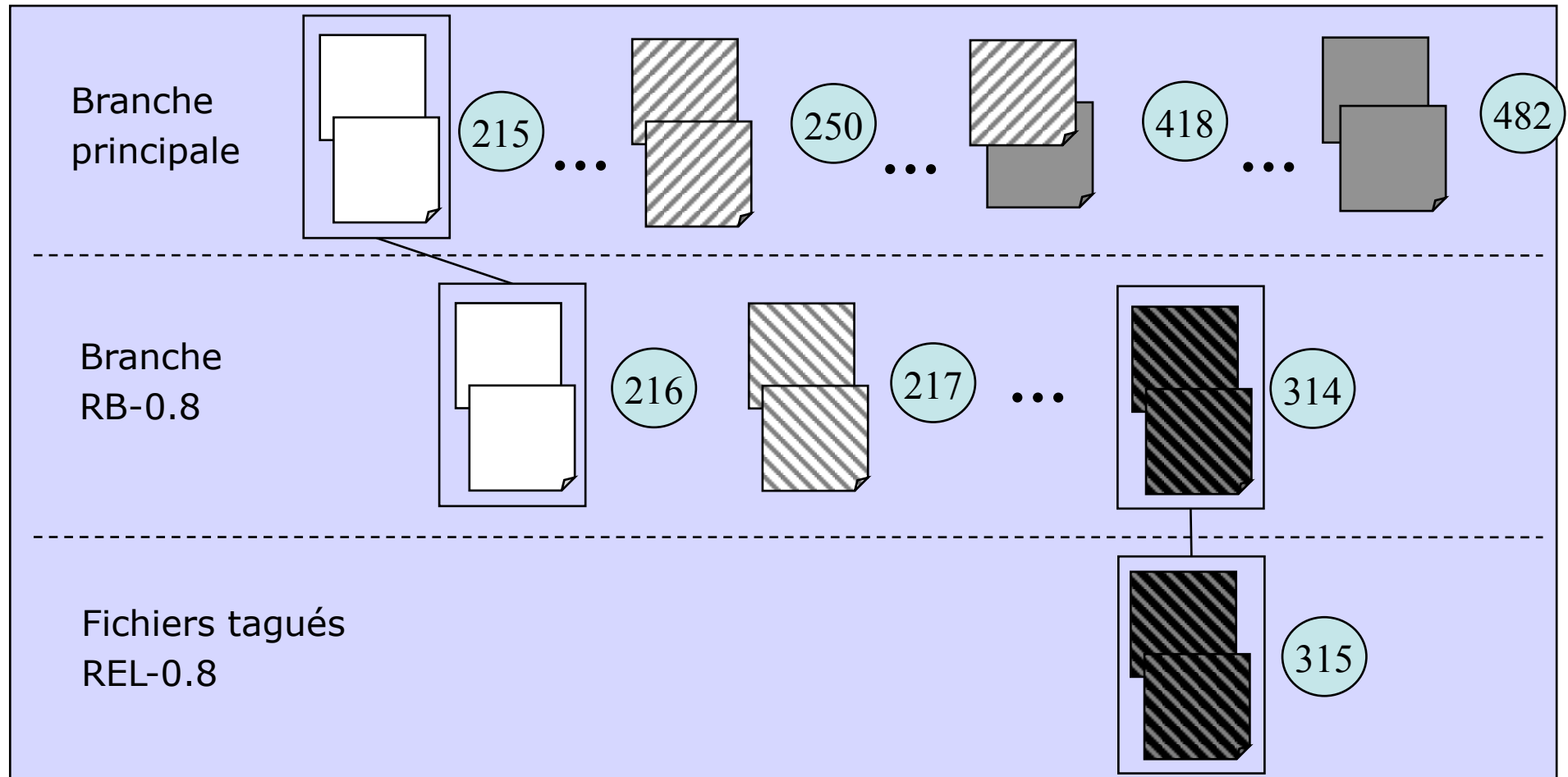
La gestion de version (11/13)

Exemple du cycle de préparation, diffusion et maintien d'une version



La gestion de version (12/13)

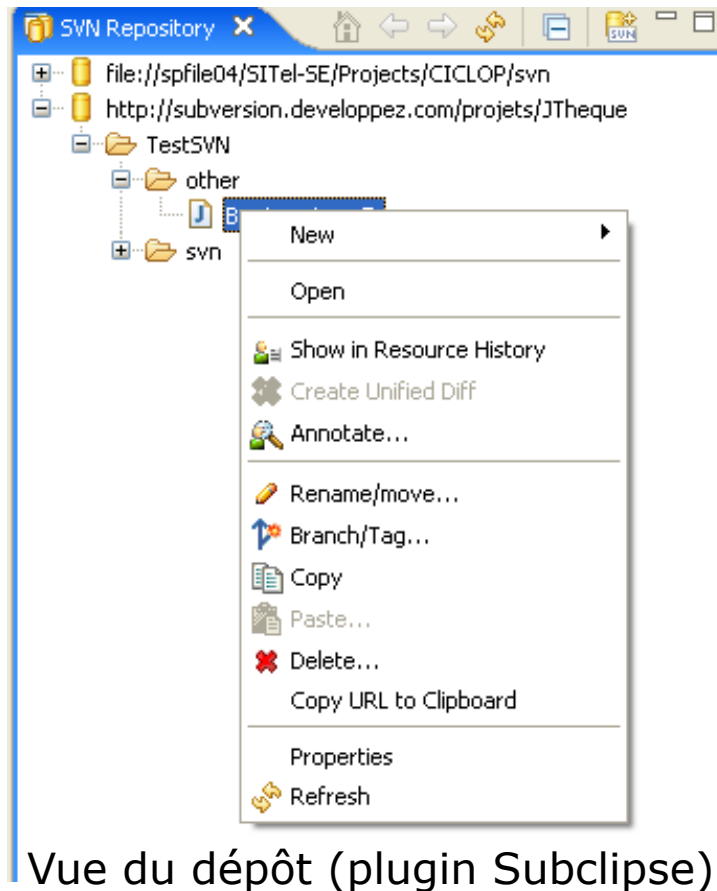
Dépôt



La gestion de version (13/13)

■ Exemples de clients :

- Client CVS : WinCVS, TortoiseCVS
- Client SVN : TortoiseSVN
- Plugins CVS et SVN pour les EDI



Menu du plugin Subclipse (SVN)

