

# Gestionnaires de versions concurrentes

Subversion, Git,...

Olivier Barais,

D'après une partie des slides de Didier  
Vojtisek



# SOMMAIRE

1. Contexte
2. Survol des fonctionnalités
3. Subversion

- Utilisation (Gestion de versions, Gestion de la concurrence
- Utilisation avancée (Lock, Gestion de la base, Branches, Co  
Macro, Alertes, Aliases)



4. Gestion de version décentralisée (Git)
5. Aller plus loin

–

Outils connexes

•

version

Structurer son code pour aider la gestion de la version

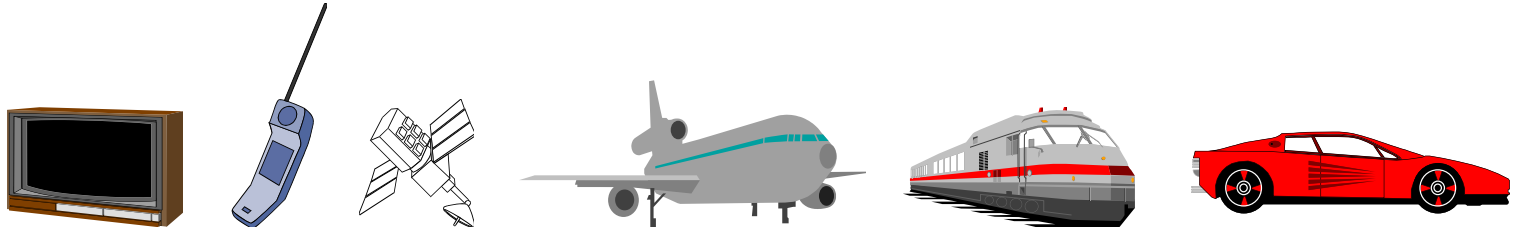


6. Conclusion

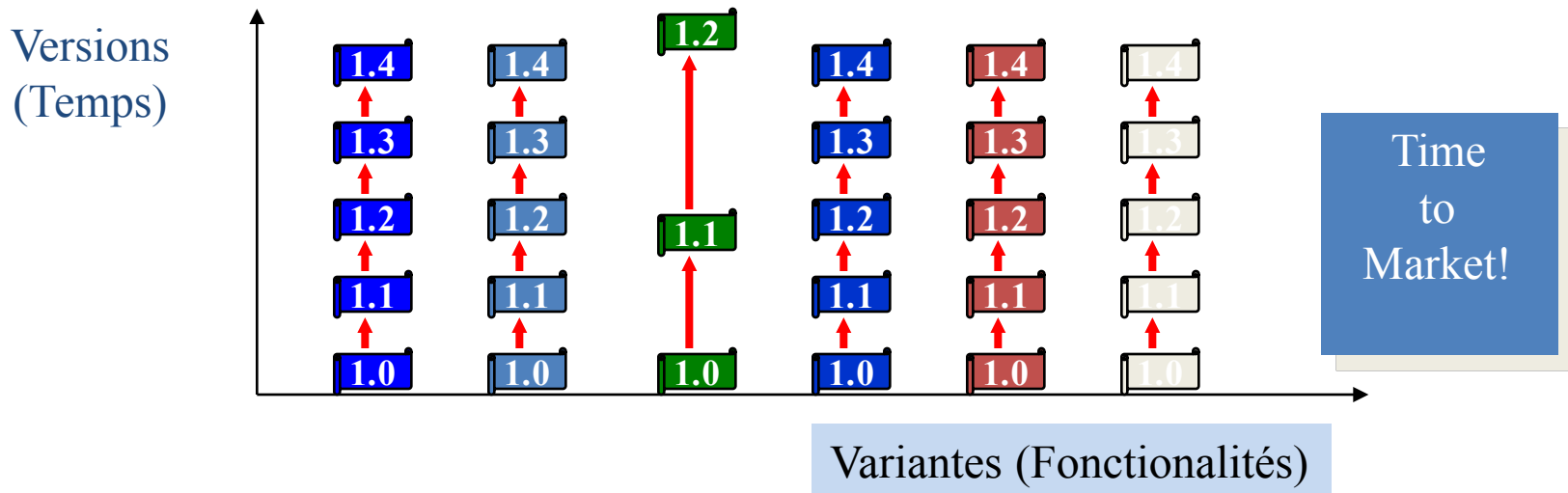


# Contexte

# Problématique du logiciel « moderne »



- Importance des aspects non fonctionnels
  - systèmes répartis, parallèles et asynchrones
  - qualité de service : fiabilité, latence, performances...
- Flexibilité accrue des aspects fonctionnels
  - notion de *lignes de produits* (espace, temps)



# The 3 Dimensions of Software Configuration Management: [Estublier et al. 95]

- The Revision dimension
  - Evolution over time
- The Concurrent Activities dimension
  - Many developers are authorized to modify the same configuration item
- The Variant dimension
  - Handle environmental differences
- Even with the help of sophisticated tools, the complexity might be daunting
  - Try to simplify it by reifying the variants of an OO system

# Variants in Software Systems

- Hardware Level
  - Heterogeneous Distributed Systems
  - Peculiarities in Target Operating System
  - Compiler Differences
  - Range of Products
  - User Preferences for GUI
  - Internationalization
- $V_i = 16$
  - $V_p = 4$
  - $V_n = 8$
  - $V_g = 5$
  - $V_l = 24$

- Number of Variants =  $V_p * V_n * V_g * 2^{V_i + V_l - 2}$
- That's 43,980,465,111,040 possible variants

# 2

## Survol des fonctionnalités

# Outils existants

- ClearCase
- Continuus
- PVCS
- Visual SourceSafe
- CVS : Concurrent Versions System
- Subversion
- Svk, git+cogito, Mercurial, bazaar
- ...

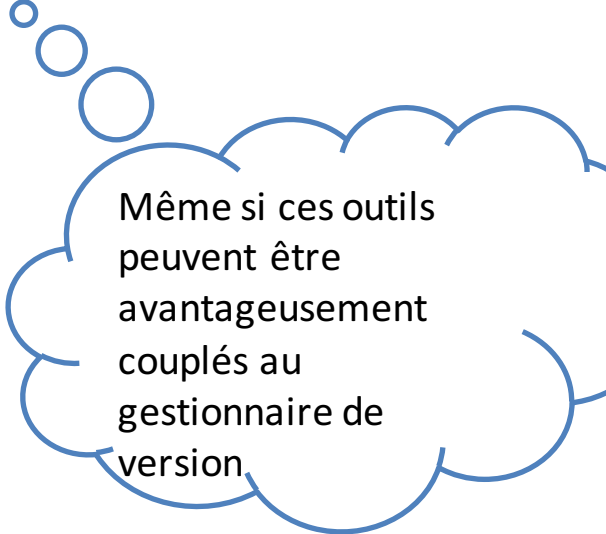


# Pourquoi faire ?

- Travailler à plusieurs sur les mêmes fichiers en même temps (concurrency)
- Gérer les versions des sources d'un développement
  - Tags symboliques
  - Comparaisons entre versions
  - Multiple lignes de développement dans une seule base
  - Gestion des branches de développement
  - Support pour les fichiers binaires
  - Envois d'évènements
  - Garder des copies de sauvegarde

# Ce que ca ne fait pas

- Système de construction
- Suivi de bugs
- Suivi de dépendances
- Procédure de test
- Gestion de la documentation du code



Même si ces outils  
peuvent être  
avantageusement  
couplés au  
gestionnaire de  
version.

# Disponibilité

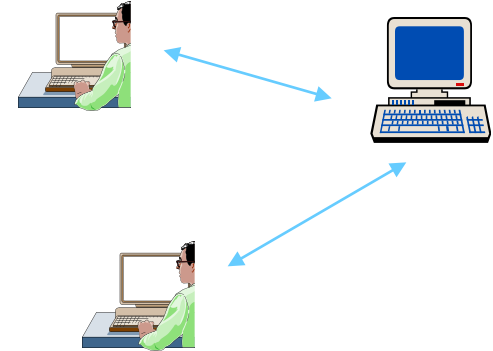
- Serveurs Subversion, CVS, Git ou Mercurial sont souvent disponibles via les services de forges
  - ex: sourceForge, google code, ...
- Logiciel libre disponible pour toute plate-forme
- Client en ligne de commande
- Clients intégrés à des outils
  - Eclipse, Netbeans, visual studio,...
- Plusieurs clients graphiques
  - Ex pour cvs : Eclipse, WinCVS, tkCVS, jCVS, WebCvs, Jalindi igloo, TortoiseCVS ...
  - Ex pour svn : RapidSVN, eSvn, Qt client, kdesvn, ViewCVS, Subversive, TortiseSVN, Ankhsvn, ...

# 3

## Subversion (SVN)

Utilisation de base

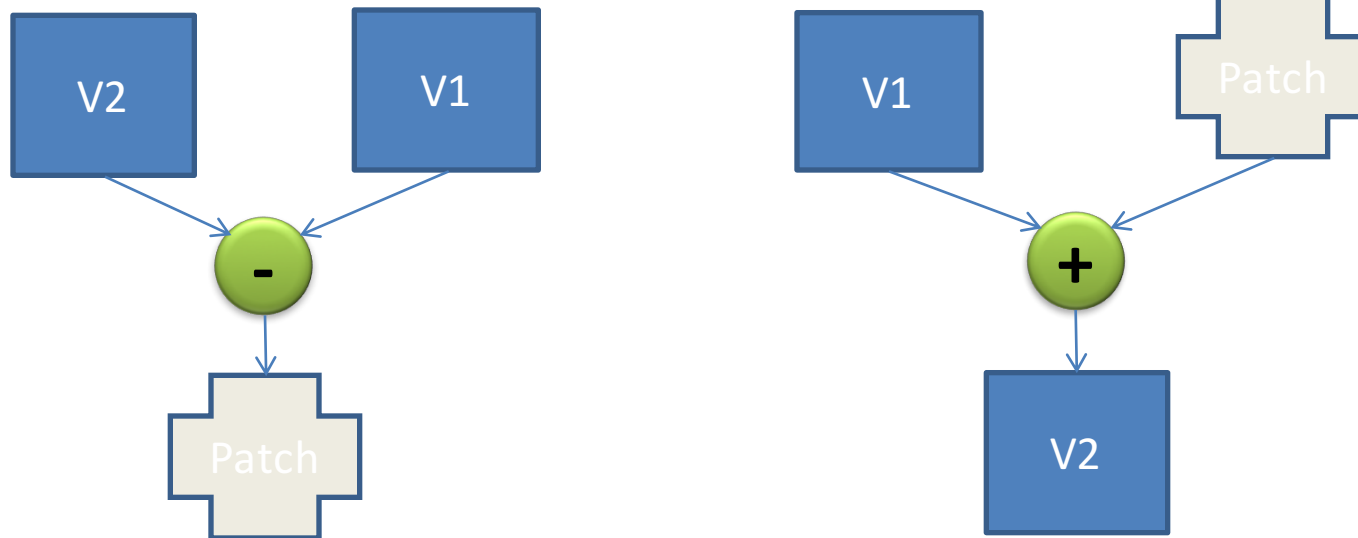
# Principes de base



- Modèle clients/serveur
  - Votre copie de travail contient un répertoire `.svn` (généralement caché)
- Une fois envoyé sur le serveur, les modifications ultérieures peuvent être inversées (i.e. un fichier n'est jamais vraiment supprimé, il part dans une zone cachée)

# Diff et patch

- Economise l'espace disque en ne conservant que les différences



# Modèle d'utilisation

## *Checkout, Update, Commit*

- “On édite d’abord et on fusionne après”

- **Checkout**

Crée une copie privée dans un répertoire de travail. Copies multiples, version multiples possibles.

- **Update**

Met à jour une copie de travail à partir de la version dans la base

- **Commit**

Valide les changements d’une copie de travail pour les ajouter à la base. La copie de travail doit être à jour avec la base

# Révision et tag

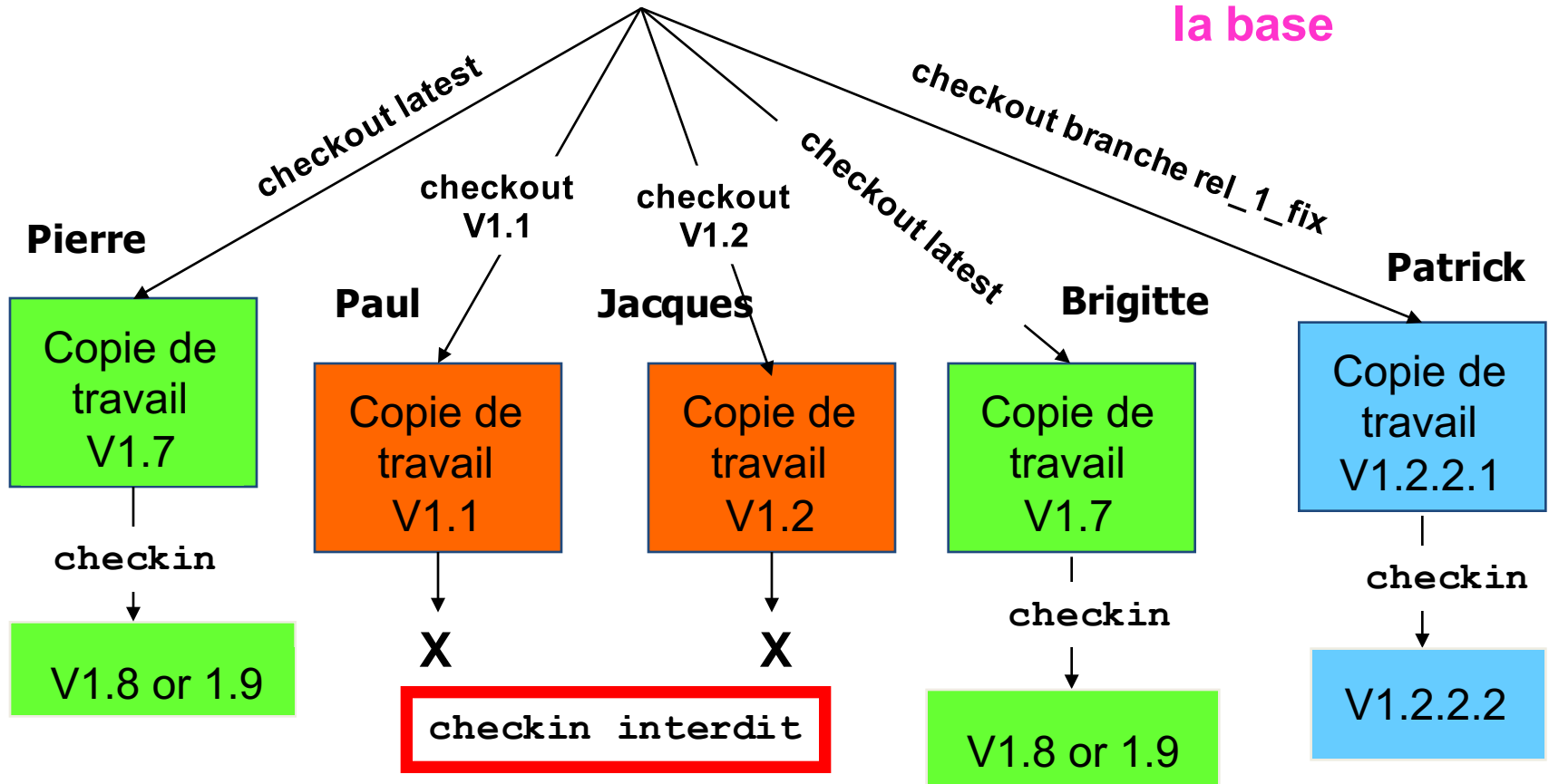
- Révision
  - Numéro qui s'incrémente à chaque changement validé dans la base.
- Tag (étiquette)
  - Pour marquer des étapes où l'on souhaite pouvoir revenir ultérieurement
  - Pour marquer une version distribuée.
- Remarque :
  - Les numéros de révisions ne sont pas liés aux versions du logiciel.



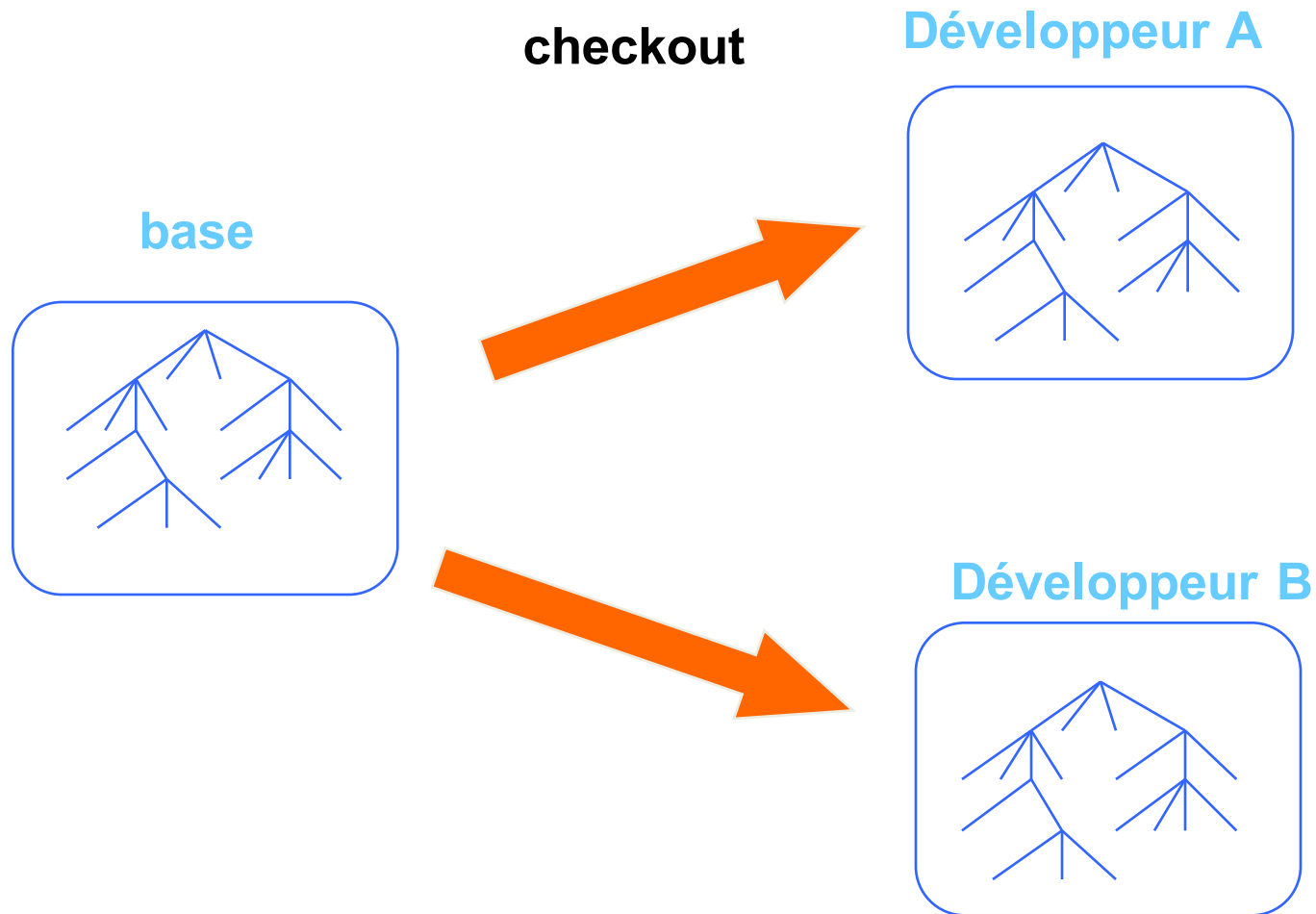
# Checkout concurrents

Base CVS/SVN

Par défaut, le checkout ne bloque pas le fichier dans la base



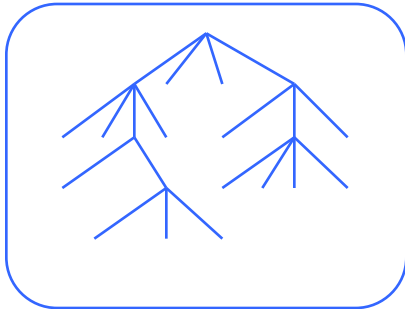
# Développement idéal (1/4)



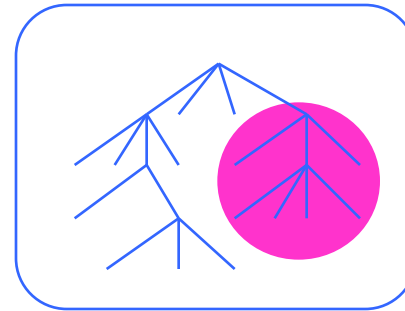
# Développement idéal (2/4)

développement

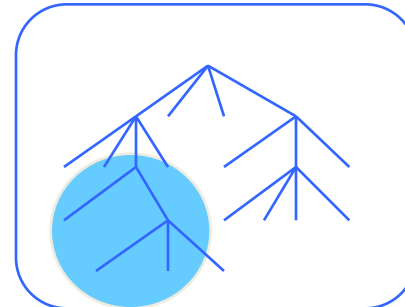
base



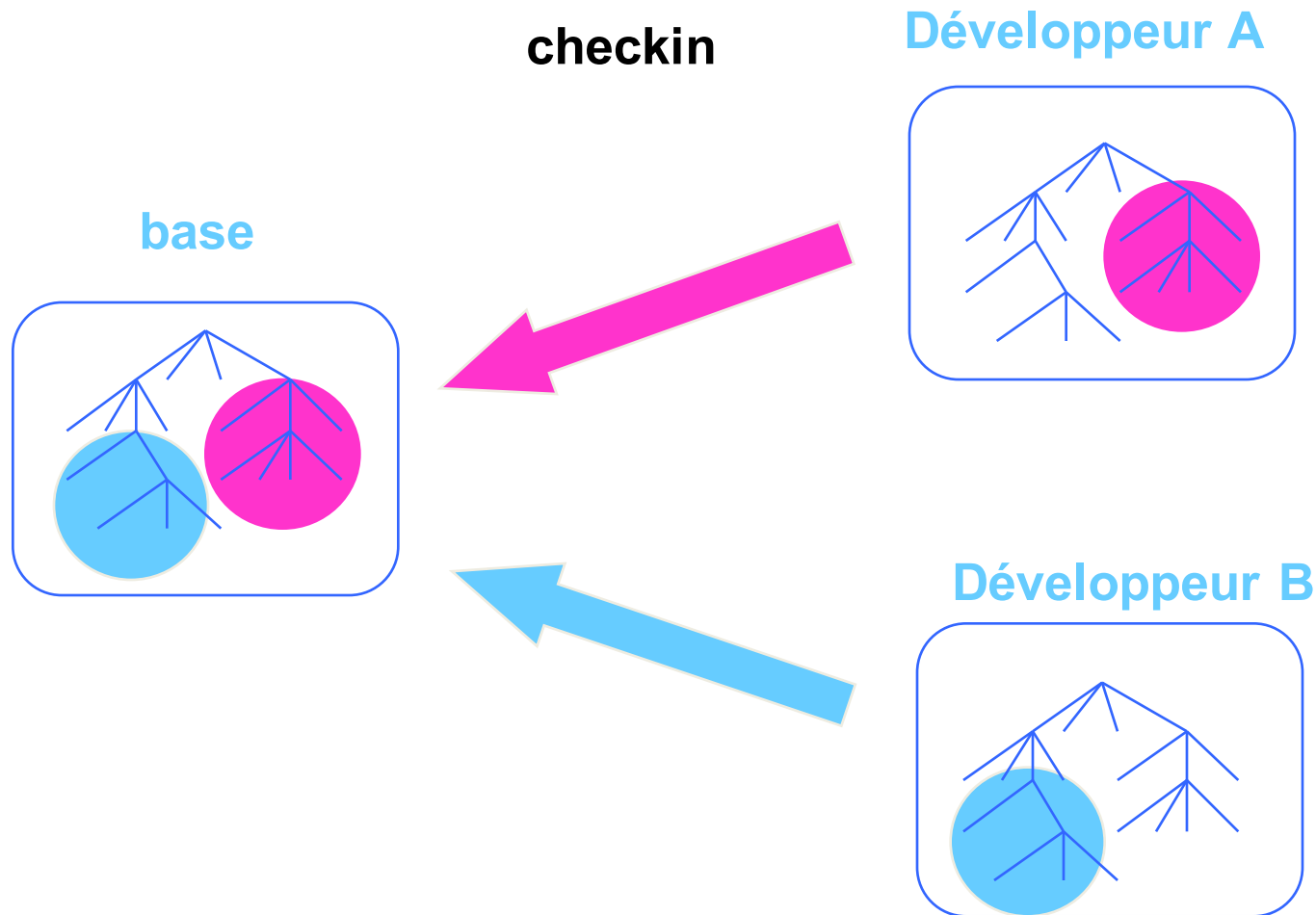
Développeur A



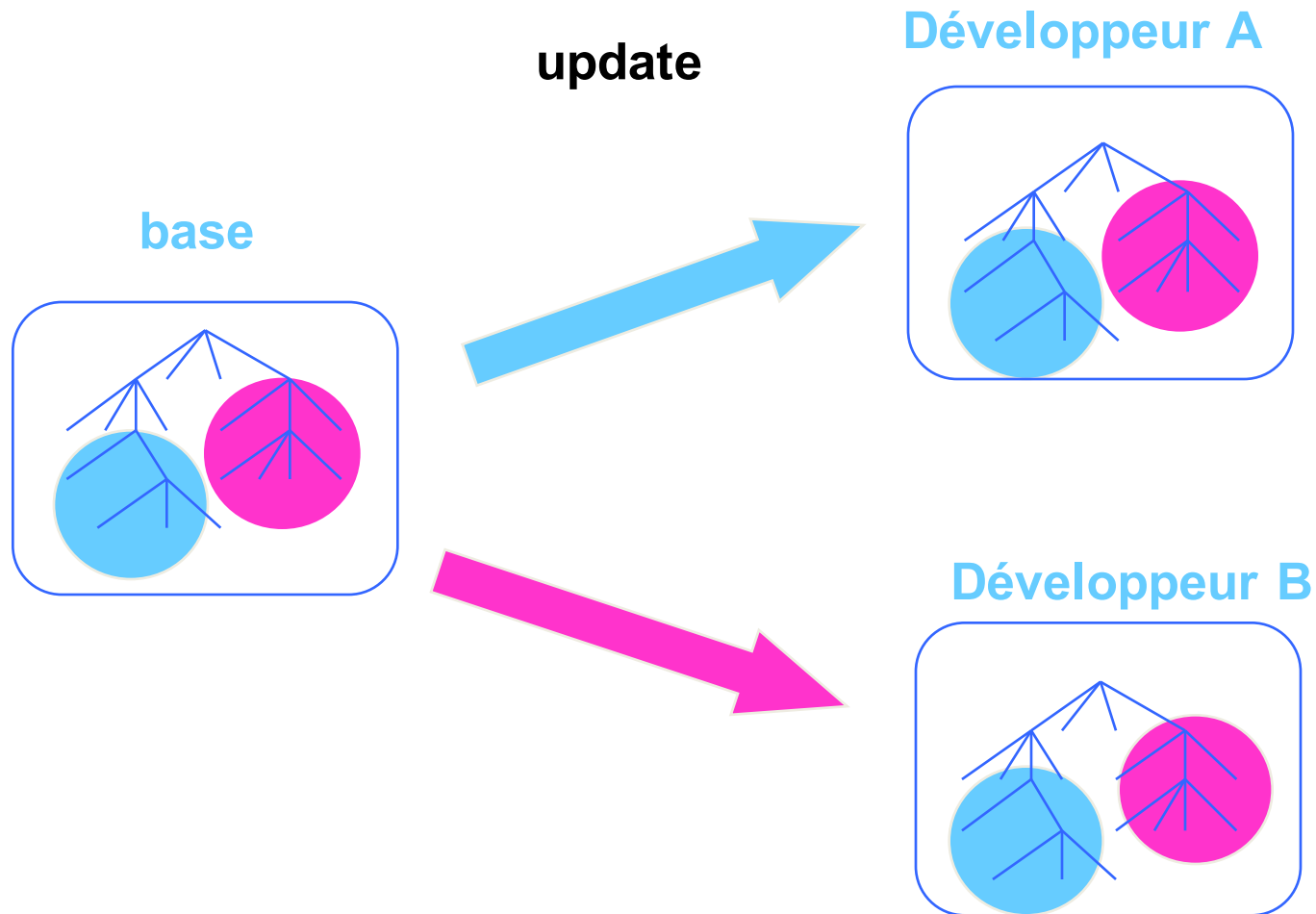
Développeur B



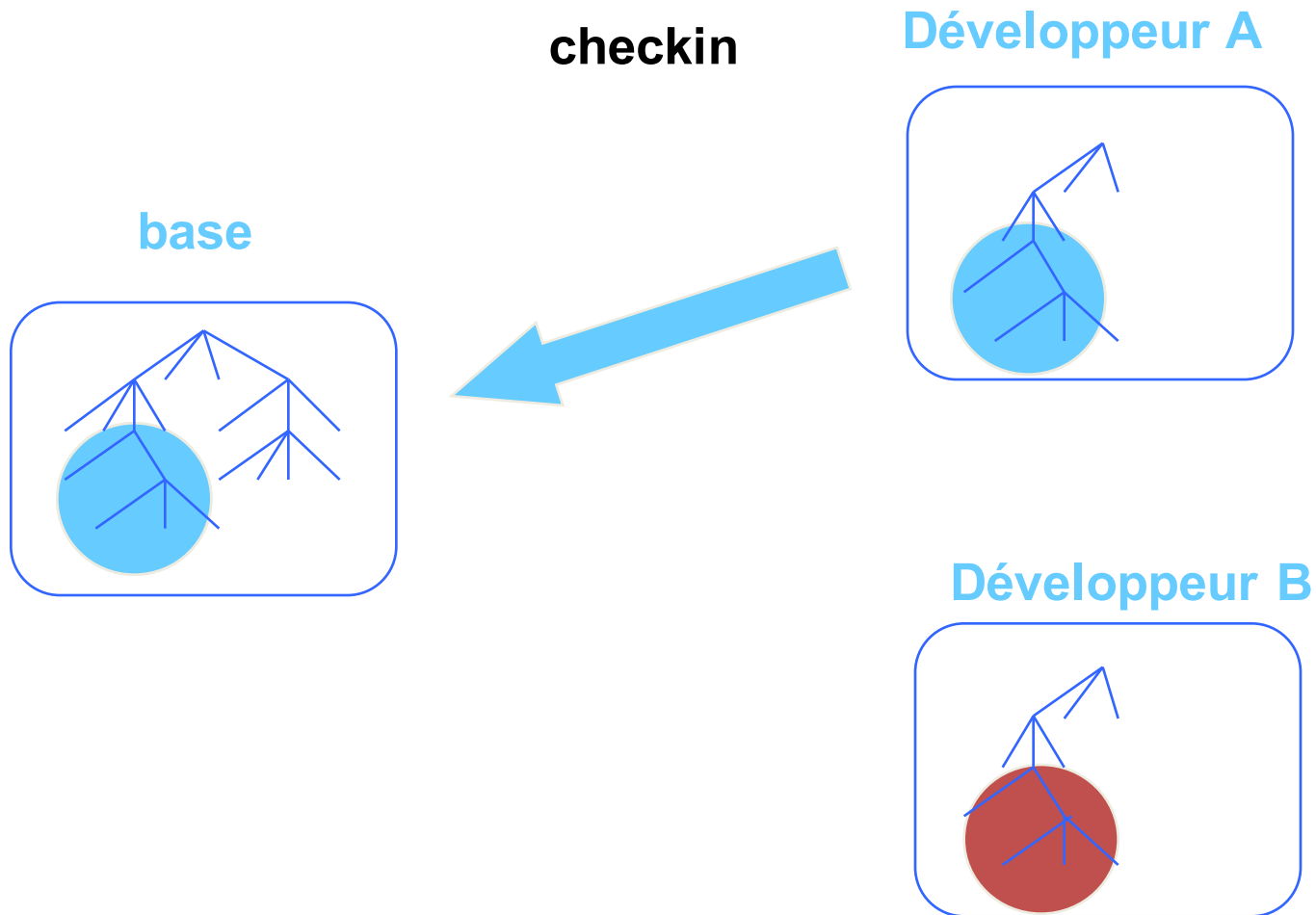
# Développement idéal (3/4)



# Développement idéal (4/4)



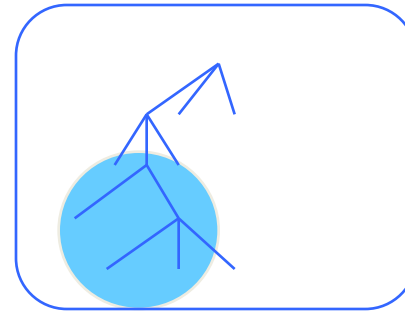
# Développement réel (1/5)



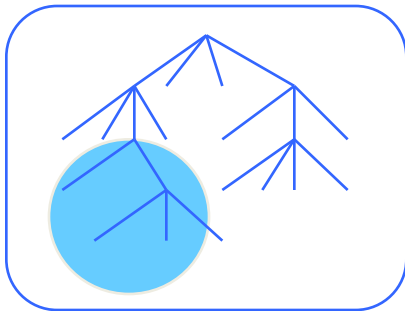
# Développement réel (2/5)

checkin

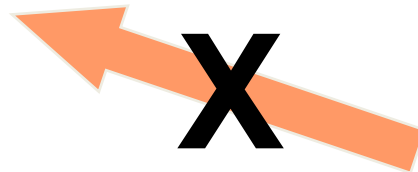
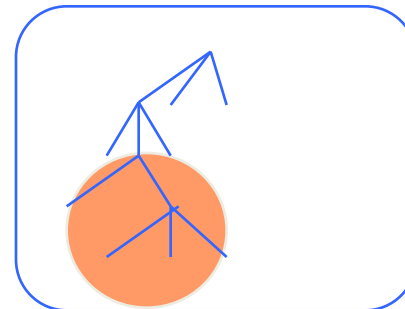
Développeur A



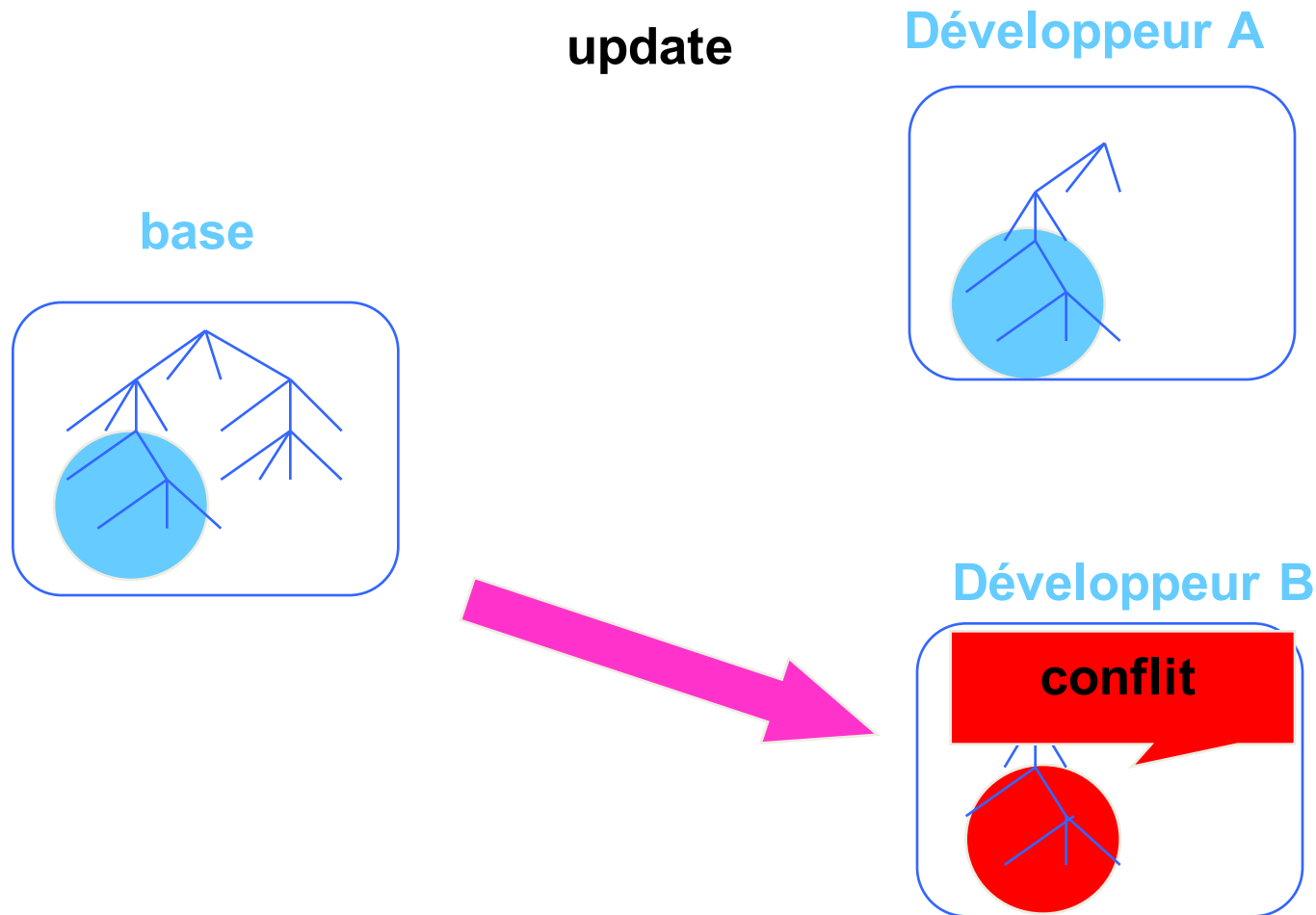
base



Développeur B



# Développement réel (3/5)

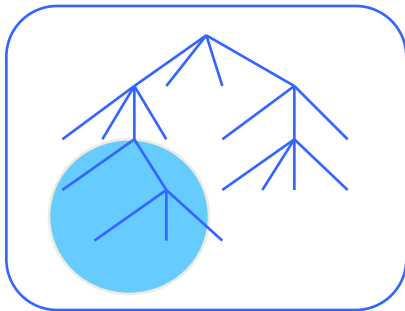




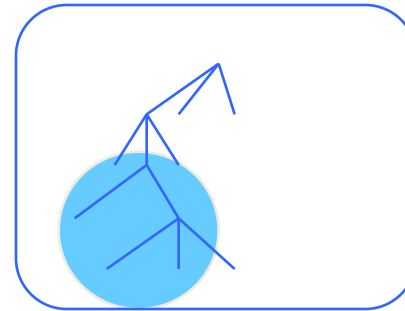
# Développement réel (4/5)

## Résolution du conflit

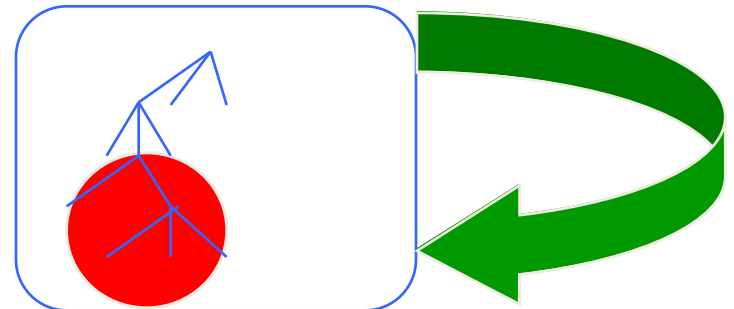
base



Développeur A



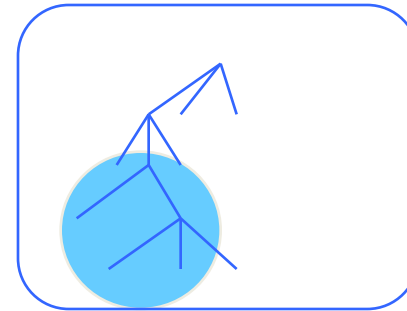
Développeur B



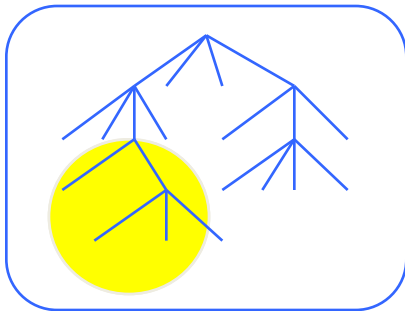
# Développement réel (5/5)

checkin

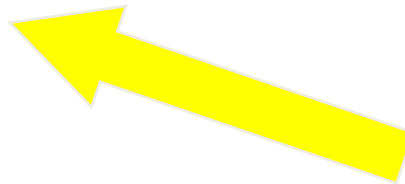
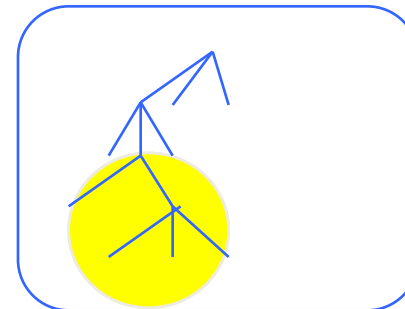
Développeur A



base



Développeur B



# Etats possibles après une synchro avec update (1/2)

- **(?) unknown** : le fichier n'est pas connu sur le serveur. Utiliser cvs add.
- **(A)dded** : l'utilisateur a ajouté un nouveau fichier mais il n'est pas encore rendu public (commit)
- **(U)pdated** : le fichier n'existait pas dans la copie de travail ou n'avait pas été modifié localement. La nouvelle version remplace l'ancienne.
- **(R)emoved** : Le fichier a été supprimé de la base, il doit être détruit de la copie locale.
- **(M)odified** : le fichier a été modifié localement, mais il n'a pas encore été rendu public

# Etats possibles après une synchro avec update (2/2)

- (M) **erged** : Le fichier a été modifié par un autre utilisateur mais aussi localement. Cependant CVS/SVN a réussi à effectuer une fusion sans conflit. Une copie de sauvegarde est créée dans la copie privée.
- (C) **onflict** : Le fichier a été modifié par un autre utilisateur mais aussi localement. Un conflit a été détecté lors de la fusion. Une copie de sauvegarde est créée.

# Résolution d'un conflit

- Commande " diff "
- Outils graphiques montrant les deux versions.
- Eviter les conflits :
  - penser à faire un update régulièrement
  - Chaque intervenant doit travailler sur des domaines fonctionnels séparés.
  - Communiquer

# Concurrence sur des fichiers non texte

- Pas de diff disponible, les conflits sont à éviter à tout prix !
- Dans ce cas, SVN sert surtout pour l'historique.
- Une solution si les accès concurrents sont absolument nécessaires:
  - Utiliser une copie locale dans un répertoire d'équipe et laisser gérer le problème de concurrence par l'outil qui ouvre le fichier
  - Nécessite de bien informer les utilisateurs sur les zones qu'ils peuvent manipuler depuis leur copie locale et depuis la copie locale d'équipe.



# Subversion

## Utilisation de avancée

# Utilisation des Lock

- Permet de traiter les cas des conflits sur fichiers binaires
- Par défaut, non obligatoire :
  - Découvre le lock lorsque l'on veut faire un commit !
  - Possibilité de voler un lock !! (communication avec le propriétaire vivement recommandée)
- Utilisation recommandée de
  - `svn:needs-lock`
  - La copie locale est en lecture seule
- Disponible à partir de la version 1.5 de SVN



# Commandes simples habituelles

svn checkout

Récupère des fichiers pour édition.

svn commit

Valide les modifications vers la base

svn update

Met à jour la copie locale avec la base.

svn add

Ajoute des fichiers/répertoires dans la base.

svn copy

copie des ressources (avec historique) .

(permet les tags et les branches)

svn delete

Supprime un élément dans la base.

svn status

Montre le status des fichiers locaux.

(ajouter --show-updates pour aussi voir les modifications sur le serveur)

svn log

Affiche l'historique des révisions.

svn diff

Compare les fichiers de la copie de travail avec les versions de la base.

Extra:

svn blame

Identifie le dernier auteur d'une ligne.

# Base type

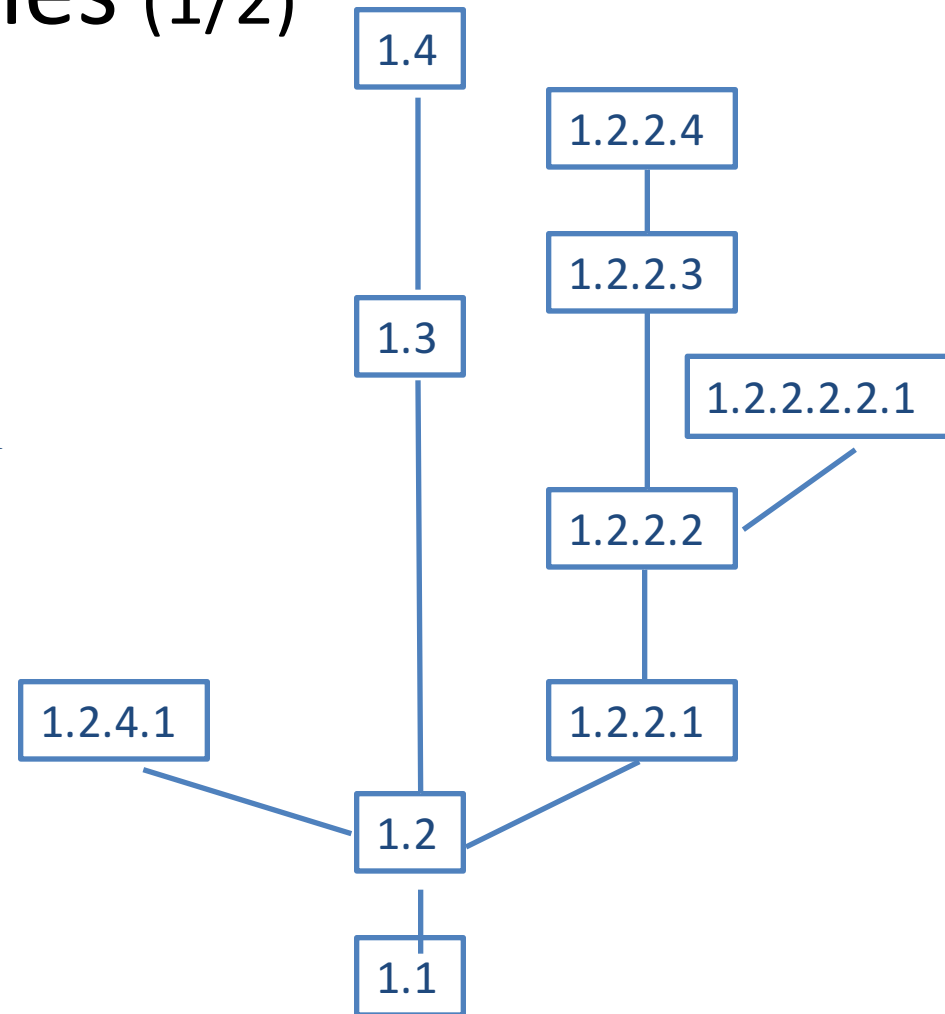
- Forme minimale recommandée
  - trunk
  - branches
  - tags
  - (releases)

# Organisation de la base

- Même si SVN permet de déplacer les répertoires, ne négligez pas une bonne organisation de vos sources dès le démarrage
- Ex: eclipse tend à mettre tous les "projet eclipse" à la racine du dépôt
  - => forcer un sous répertoire correspondant à "votre" projet pour pouvoir regrouper les projets eclipse
  - Ex:
  - Monoutil
    - trunk
      - fr.irisa.monoutils
      - fr.irisa.monoutils.ui
      - fr.irisa.monoutils.texteditor
      - fr.irisa.monoutils.graphicaleditor
      - ...
    - branches
    - tags
    - releases
      - V1.0
        - » fr.irisa.monoutils
        - » fr.irisa.monoutils.ui
        - » fr.irisa.monoutils.texteditor
        - » fr.irisa.monoutils.graphicaleditor
        - » ...
  - Mon2iemFuturUtil
    - trunk
    - branches
    - tags
    - releases
  - LesPartiesCommunesQuiNeFontPartidAucunOutils
    - ...

# Branches (1/2)

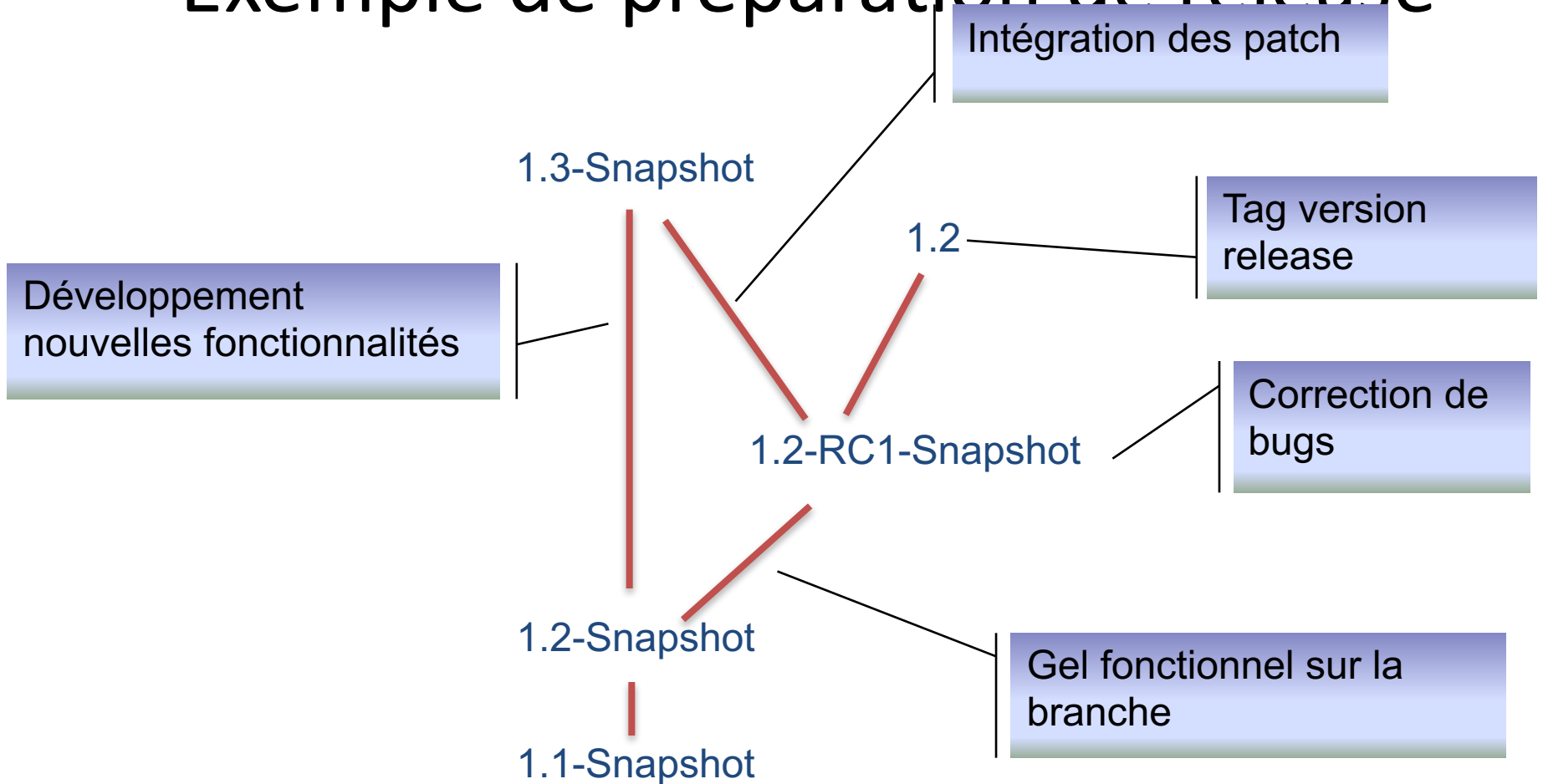
- correction urgente d'un bug sur une version livrée
- commit et update sans troubler la branche principale
- Dans SVN, c'est en fait une copie d'un ensemble de ressources dans le répertoire dédié (typiquement, on copie le contenu de "trunk" dans "branches")



# Branches (2/2)

- Toujours planifier de joindre ou d'abandonner une branche
- A utiliser avec précaution ...
  - La jonction de branches pose pas mal de problèmes de conflits, en particulier lors de branches multiples
  - Il est préconisé que les développeurs d'une branche récupèrent régulièrement le contenu de la branche principale dans leur branche

# Exemple de préparation de release

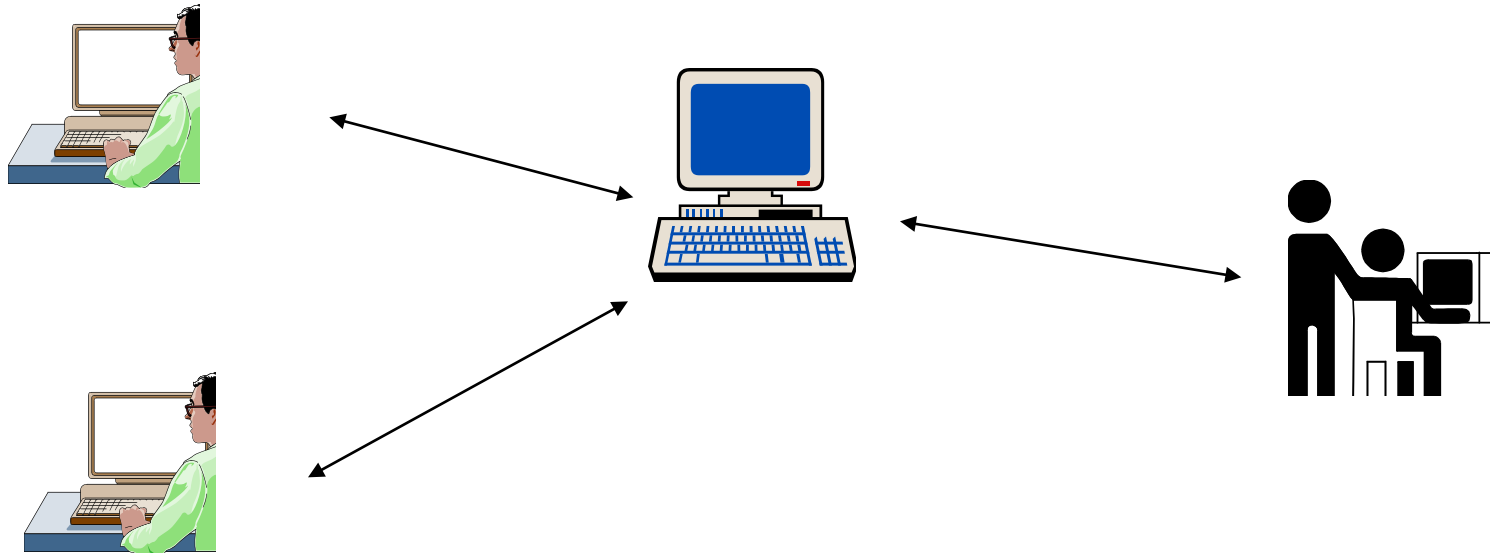


# Configuration de subversion

- Des propriétés attachées aux ressources (fichier ou répertoire) permettent d'affiner le comportement
  - `svn:ignore` permet d'ignorer des ressources
  - `svn:eol-style` permet de préciser le type de retour chariot pour les fichiers texte.
    - Native,
    - CRLF, LF, CR

# Problème Unix/Windows

- Spécificité de l'Inria : travail en environnements multiples





# Unix/Windows

Cas Normal

Texte initial (Windows) :

Mon texte {CR}{NL}

Import avec un client Windows

Texte sur le serveur (Unix) :

Mon texte {NL}

Export à partir d'une station

Texte obtenu (Unix) :

Mon texte {NL}

# Exemple de problème Unix/Windows

Texte initial (Windows) :

Mon texte {CR}{NL}

Import à partir d'une station Unix

Texte sur le serveur (Unix) :

Mon texte {CR}{NL}

Export à partir d'un PC

Texte obtenu ( ??? ) :

Mon texte {CR}{CR}{NL}

# Keyword substitution

- Mots clés à insérer dans les sources
  - \$Author\$
  - \$Date\$
  - \$Revision\$
  - \$Id\$
  - \$HeadURL\$
- Permet d'aider à la gestion des fichiers sans avoir nécessairement accès au web pour retrouver l'historique
- Pas mis par défaut ! Doit être activé grâce à la propriété `svn:keywords` sur chaque fichier

```
$ svn propset svn:keywords "Date Author" weather.txt  
property 'svn:keywords' set on 'weather.txt'
```

## Substitutions (2)

- À utiliser généralement dans une partie en « commentaire »
- Attention aux fichiers dont le format est inconnu
- Solution:
  - désactiver l'expansion
  - le passer en tant que fichier binaire
    - `svn:mime-type`

# Choix du mode de connexion (ex : SVN)

- https://
  - Simple à utiliser (mot de passe de la forge)
  - Passe généralement à travers les parefeux
  - Relativement lent (multiplie les connexions, transite par le serveur apache sous jacent)
- svn+ssh://
  - Rapide
  - Certaines entreprises bloquent le port 22 de ssh
  - Pas toujours disponible sur certaines forges
  - Nécessite de gérer des clés ssh et de poster la clé publique sur la forge
    - Si le client SVN insiste pour avoir le mot de passe trop souvent (n'arrive pas à le conserver correctement pour une session):
      - Utiliser un ssh-agent
      - Ou utiliser une clé sans passphrase mais n'utiliser cette clé que pour un seul serveur !

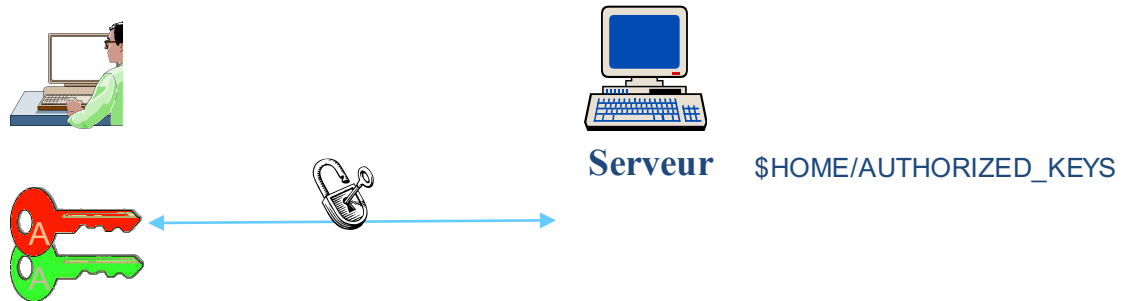
# Rappel clés ssh



- Clé publique à mettre sur le serveur
  - id\_rsa.pub
  - Ex: via l'interface web de la forge
- Clé privée à conserver dans un endroit sûr dans votre machine



- id\_rsa ou id\_rsa.ppk
- Droits sur le répertoire rwx - -
- Utilisation d'une passphrase pour crypter le fichier local



# Notifications

- Vous pouvez customiser votre dépôt sur différents événements
  - par ex pour envoyer des mails sur certains événements

- Utilise la notion de "Hooks"

<code>post-commit.tmpl</code>	<code>post-</code>
<code>unlock.tmpl</code>	
<code>pre-revprop-change.tmpl</code>	<code>post-</code>
<code>lock.tmpl</code>	
<code>pre-commit.tmpl</code>	<code>pre-</code>
<code>unlock.tmpl</code>	
<code>post-revprop-change.tmpl</code>	<code>pre-</code>
<code>lock.tmpl</code>	
<code>start-commit.tmpl</code>	

- Attention les actions ne doivent généralement pas être trop longues !
  - Ex: faire une compilation pour vérifier que le commit est valide est généralement une mauvaise idée sauf si vous êtes seul à utiliser le dépôt ...

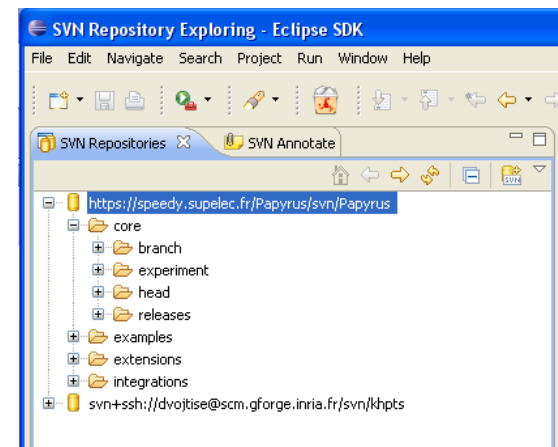
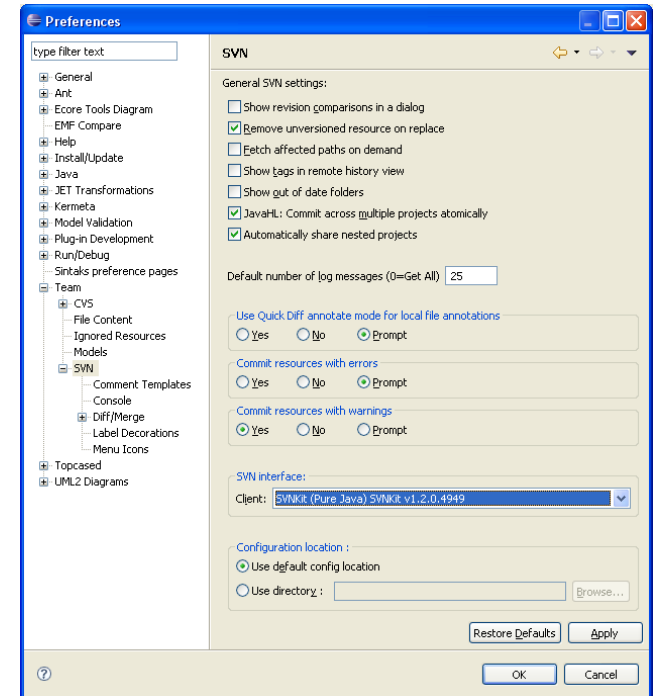
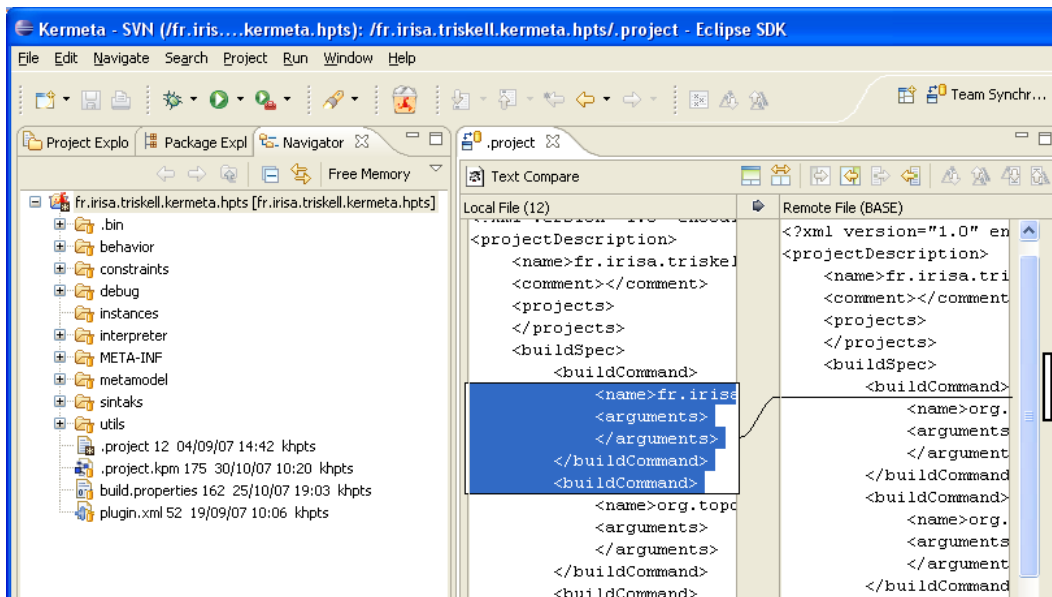
# Définitions externes (aliases)

- Pour créer une copie locale à partir de différentes sources
- Peut pointer une version particulière
  - Créer un répertoire pour héberger "l'alias"
  - Utiliser la propriété :
    - svn:externals
    - Une ligne par "external"
  - Lors d'une checkout, les externals seront mis sous ce répertoire (en plus des ressources qui y seraient aussi)



# SVN et Eclipse

- Un vue supplémentaire (gestion des serveurs et fenêtre de synchronisation), des menus contextuels

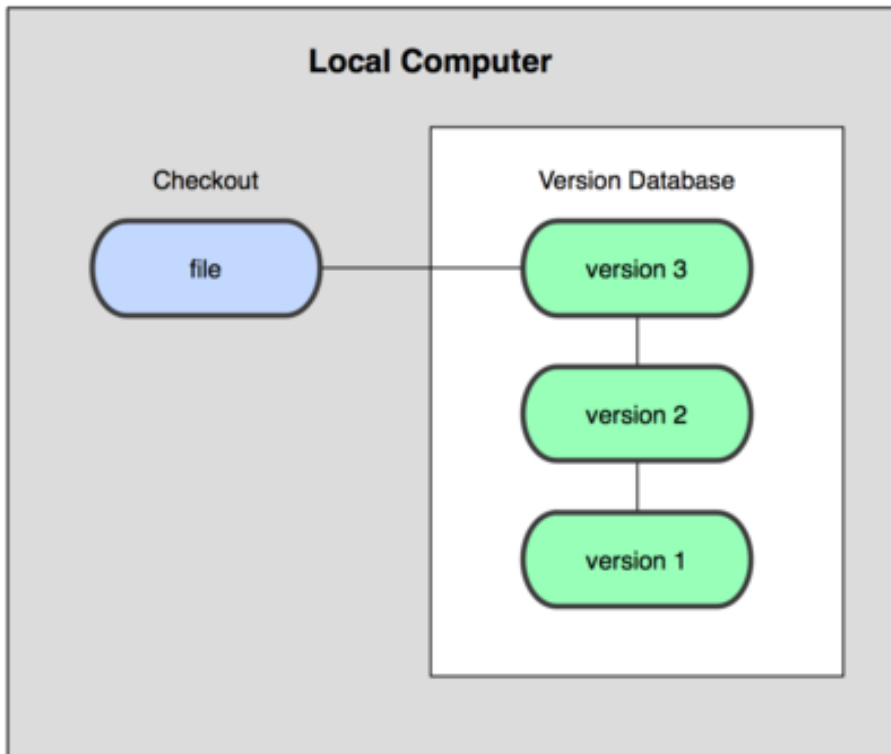


# 4

## Gestion de version décentralisée

# Local Version Control

- Gestion de version locale à l'ancienne (RCS)



# Centralized Version Control

- Exemple subversion

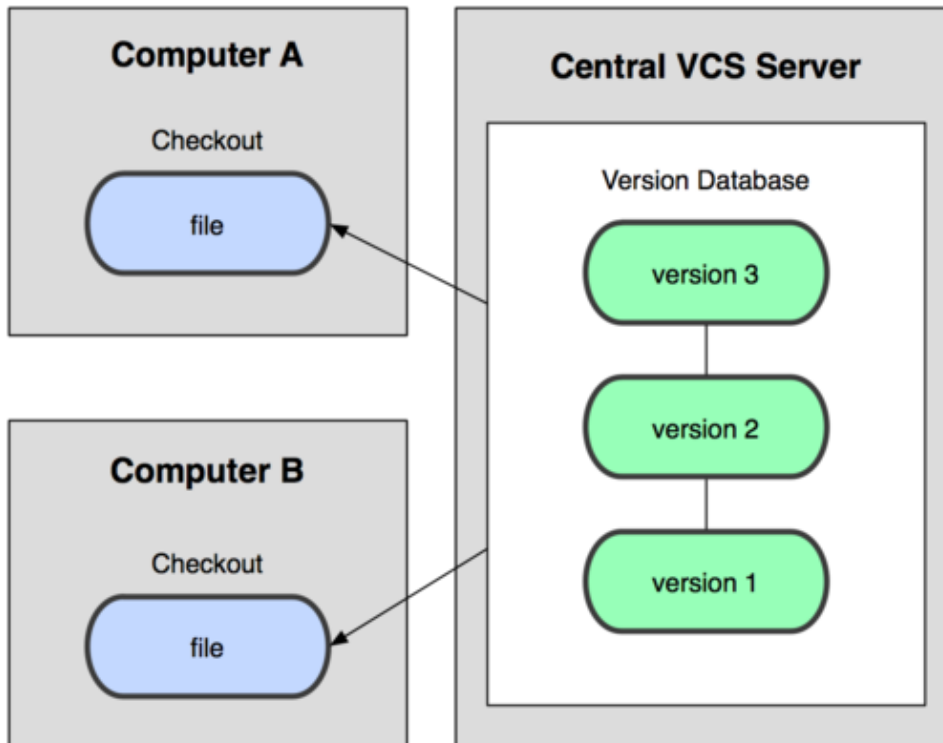


Image stolen from <http://progit.org/book/> which is really good, go read it.

# Distributed Version Control

- Gestion de version distribué

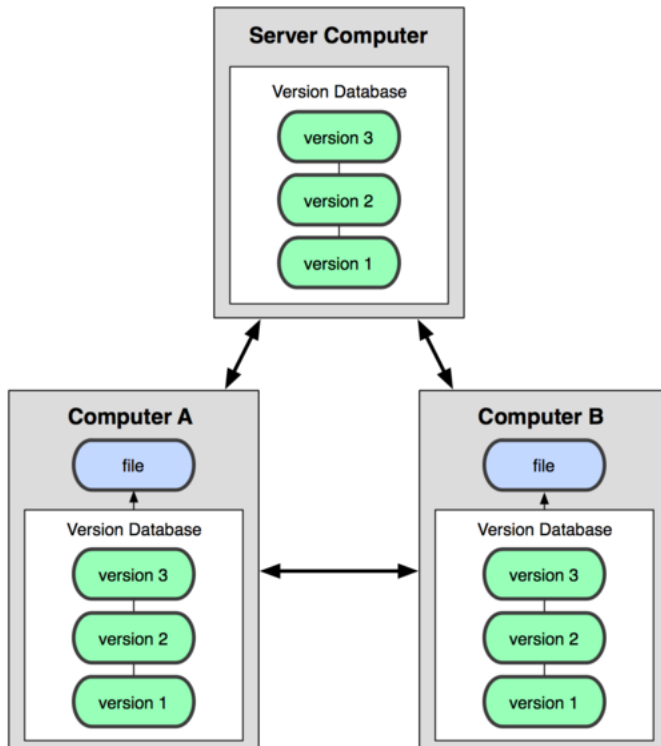
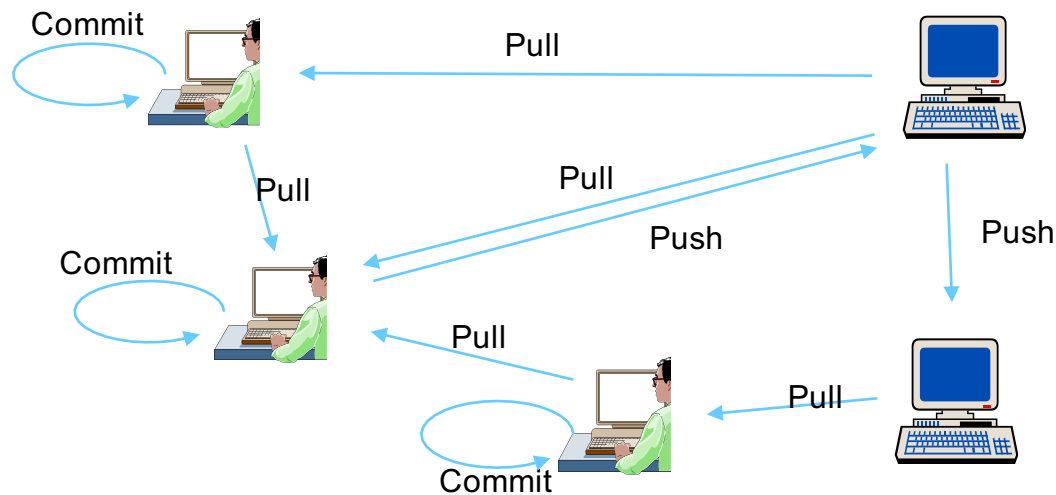


Image stolen from <http://progit.org/book/> which is really good, go read it.

# Autres outils: Systèmes décentralisés

- Chaque développeur peut avoir son dépôt dans lequel il peut commiter
  - 2 commit parfois : 1 dans le dépôt local, 1 dans le dépôt de livraison
  - Peer to peer
- Théorie des patches
  - Patch == ensemble de changement sur un arbre de fichiers
  - Utile pour gérer en configuration des modification personnelle alors que l'on n'a qu'un accès limité au SCM officiel



# Avantages des dépôts décentralisés

- Gros projets :
  - Facilité de gestion des branches et tags,
  - Notion de cercles de confiance
- Projets moyen :
  - possibilité de travailler "offline "
  - Possibilité d'avoir des branches personnelles
- Facilite certains scénarios tels que :
  - Travail synchronisé avec des dépôts officiels pour lesquels on n'a qu'un accès anonyme (modifications = uniquement des patch soumis à la communauté)

# Inconvénients des dépôts décentralisés

- Organisation des dépôts à spécifier
  - Choix d'avoir ou pas un dépôt de référence est laissé à l'organisation du projet
- Maturité de l'intégration dans les IDE
  - Mais cela évolue vite ...



# Systèmes décentralisés : QQ exemples

- GIT (*supporté par sourceForge, github, bitbucket...*)
- Bazaar (*supporté par sourceForge, ...*)
- *Mercurial* (*supporté par google code, sourceForge, ...*)
- DARCS
- SVK
- ARCH/TLA
- Monotone

# Git

## Présentation

# A Brief History of Git

- Linus uses BitKeeper to manage Linux code
- Ran into BitKeeper licensing issue
  - Liked functionality
  - Looked at CVS as how not to do things
- April 5, 2005 - Linus sends out email showing first version
- June 15, 2005 - Git used for Linux version control

# Git is Not an SCM

*Never mind merging. It's not an SCM, it's a distribution and archival mechanism. I bet you could make a reasonable SCM on top of it, though. Another way of looking at it is to say that it's really a content-addressable filesystem, used to track directory trees.*

*Linus Torvalds, 7 Apr 2005*

<http://lkml.org/lkml/2005/4/8/9>

# Git Advantages

- Resilience
  - No one repository has more data than any other
- Speed
  - Very fast operations compared to other VCS (I'm looking at you CVS and Subversion)
- Space
  - Compression can be done across repository not just per file
  - Minimizes local size as well as push/pull data transfers
- Simplicity
  - Object model is very simple
- Large userbase with robust tools

# Some GIT Disadvantages

- Definite learning curve, especially for those used to centralized systems
  - Can sometimes seem overwhelming to learn
- Documentation mostly through man pages

# Git Architecture

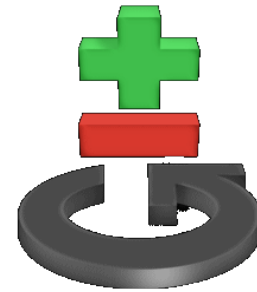
- Index
  - Stores information about current working directory and changes made to it
- Object Database
  - Blobs (files)
    - Stored in .git/objects
    - Indexed by unique hash
    - All files are stored as blobs
  - Trees (directories)
  - Commits
    - One object for every commit
    - Contains hash of parent, name of author, time of commit, and hash of the current tree
  - Tags

# Some Commands

- Getting a Repository
  - git init
  - git clone
- Commits
  - git add
  - git commit
- Getting information
  - git help
  - git status
  - git diff
  - git log
  - git show



# Git (bilan)



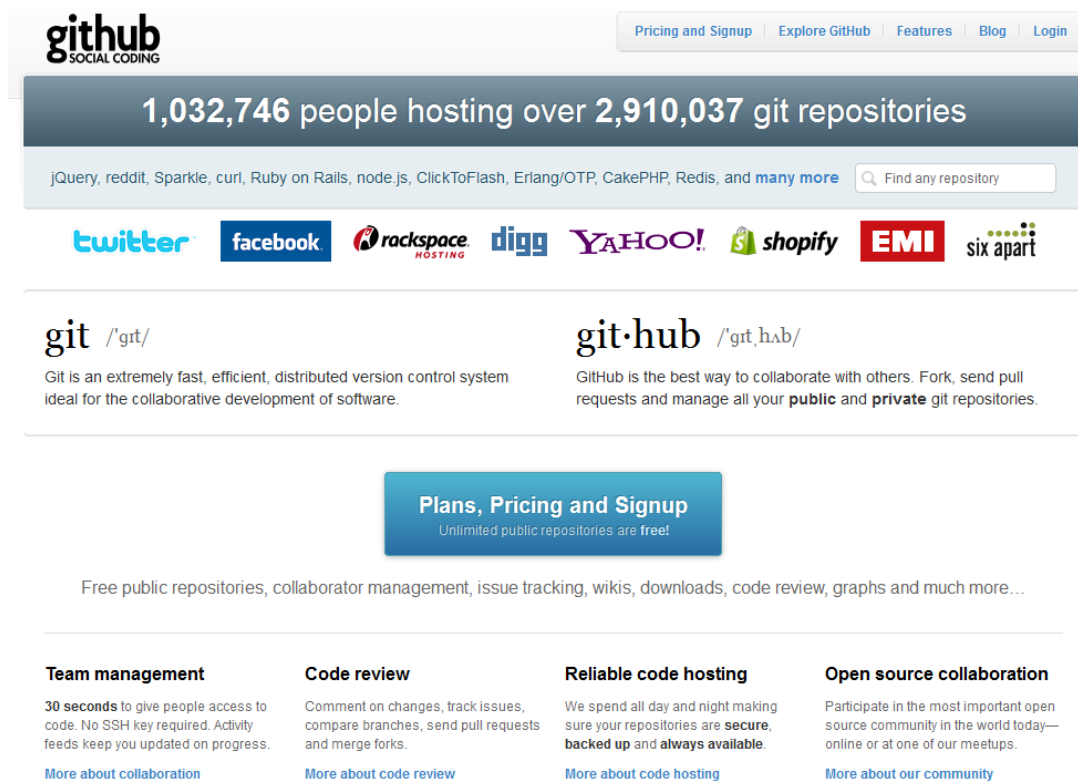
- Linus Torval (2005)
- " Pas de serveur " (gitHub)
- Support du merge plus efficace que SVN
- Historique complet en local
  - Backup "gratuit"
    - Car le dépôt est dupliqué sur tous les postes !
    - (attention au premier checkout sur les gros dépôts)
- Fonctions avancées puissantes
  - Bisect : dichotomie pour retrouver un commit particulier

# Git pour utilisateurs de CVS

CVS	Git
checkout	clone
update	pull
commit	commit -a + push
add	add
remove	rm
diff	diff
log	log

# Github

- Un peu plus qu'un dépôt de source ou une forge logicielle : un réseau social



The screenshot shows the GitHub homepage. At the top, the GitHub logo (a stylized octocat) is on the left, and navigation links for 'Pricing and Signup', 'Explore GitHub', 'Features', 'Blog', and 'Login' are on the right. Below the navigation bar, a large blue banner displays the text '1,032,746 people hosting over 2,910,037 git repositories'. Underneath the banner is a search bar with the text 'jQuery, reddit, Sparkle, curl, Ruby on Rails, node.js, ClickToFlash, Erlang/OTP, CakePHP, Redis, and many more' and a search icon. Below the search bar is a row of logos for various companies and services: Twitter, Facebook, Rackspace Hosting, Digg, Yahoo!, Shopify, EMI, and Six Apart. The main content area is divided into two columns. The left column is for 'git' (with the path '/git/') and describes it as 'an extremely fast, efficient, distributed version control system ideal for the collaborative development of software.' The right column is for 'git·hub' (with the path '/git,hab/') and describes it as 'the best way to collaborate with others. Fork, send pull requests and manage all your public and private git repositories.' Below this content is a large blue button that says 'Plans, Pricing and Signup' with the text 'Unlimited public repositories are free!' underneath. Below the button is a line of text: 'Free public repositories, collaborator management, issue tracking, wikis, downloads, code review, graphs and much more...'. At the bottom, there are four columns of text describing GitHub's features: 'Team management' (30 seconds to give people access to code), 'Code review' (comment on changes, track issues), 'Reliable code hosting' (secure, backed up, and always available), and 'Open source collaboration' (participate in the most important open source community).

github  
SOCIAL CODING

Pricing and Signup | Explore GitHub | Features | Blog | Login

1,032,746 people hosting over 2,910,037 git repositories

jQuery, reddit, Sparkle, curl, Ruby on Rails, node.js, ClickToFlash, Erlang/OTP, CakePHP, Redis, and many more

Find any repository

twitter facebook rackspace hosting digg YAHOO! shopify EMI six apart

git /'git/  
Git is an extremely fast, efficient, distributed version control system ideal for the collaborative development of software.

git·hub /'git,həb/  
GitHub is the best way to collaborate with others. Fork, send pull requests and manage all your **public** and **private** git repositories.

Plans, Pricing and Signup  
Unlimited public repositories are free!

Free public repositories, collaborator management, issue tracking, wikis, downloads, code review, graphs and much more...

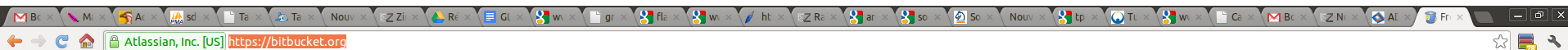
**Team management**  
30 seconds to give people access to code. No SSH key required. Activity feeds keep you updated on progress.  
[More about collaboration](#)

**Code review**  
Comment on changes, track issues, compare branches, send pull requests and merge forks.  
[More about code review](#)

**Reliable code hosting**  
We spend all day and night making sure your repositories are **secure**, **backed up** and **always available**.  
[More about code hosting](#)

**Open source collaboration**  
Participate in the most important open source community in the world today—online or at one of our meetups.  
[More about our community](#)

- <https://bitbucket.org/>



[Atlassian Home](#) [Documentation](#) [Support](#) [Blog](#) [Forums](#)

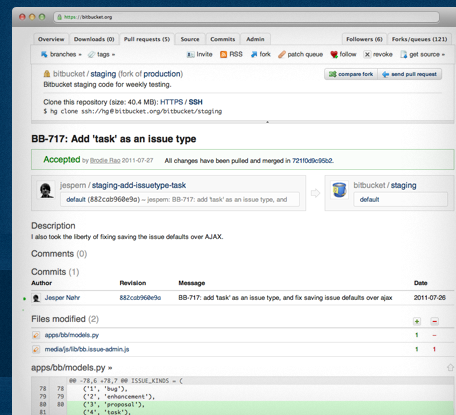
[Pricing & signup](#) [Explore Bitbucket](#) [Log in](#) [owner/repo](#)

## Unlimited DVCS Code Hosting, Free

Store all of your Git and Mercurial source code in one place with unlimited private repositories. Includes issue tracking, wiki, and pull requests.

Secure hosting with flexible permissions for your repositories. Integrates with JIRA, Jenkins, Pivotal, Cloud9 IDE and other developer tools.

**SIGN UP NOW, FREE**  
UNLIMITED PRIVATE REPOS



## Bitbucket Features

[SEE OUR PLANS & PRICING](#) 

**Old repository**

Source  
Subversion

URL (required)  
<https://svn.atlassian.com/svn/private/atlassian/crowd/trunk>

☒ Requires authorization

Username Password  
jstepka \*\*\*\*\*

### Unlimited repositories

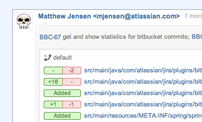
Host your code online with unlimited repositories. Share work with colleagues, collaborators, or potential employers.

Contractors	delete	Default access
Brodie Rao (brao)		
David (davidchambers)		
Dylan Etkin (etkin)		
Erik van Zijst (evzijst)		
jnoehr		
Mehmet S Catalbas (mcatalbas)		
Nicolas Venegas (nvenegas)		

### Work as a team

A centralized team account makes managing your repositories easy for you and your fellow developers. Simplify user management with groups.

```
hgsubversion / setup.py
f28e0f54a6ef 134 loc 4.6 KB
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3 import os
4 import re
5 import subprocess
6 import sys
7 import time
8 if not hasattr(sys, 'version_info') or sys.
9     raise SystemExit("Mercurial requires py
```



## Secure code collaboration

Delegate administration and grant users read/write access to your repository. Share power with your developers.

## Key integrations

Use the tools you love. Bitbucket has tight integrations with popular tools like JIRA, Crucible, Jenkins, and more!

# 5

Aller plus loin

Outils connexes

# Outils complémentaires



- Gestionnaire de bug / tâches (ex: bugzilla)
  - Permet le suivi des bugs et des demandes d'évolution entre les utilisateurs et les développeurs
  - Certains outils permettent de relier un commit à la résolution d'un bug

- Intégration continue (ex: Hudson/Jenkins)
  - Un robot récupère régulièrement les sources et les vérifie (compilation/test/métriques/...)
    - Automatise toute sorte de tâches
  - Notifie les développeurs en cas de problème



# Forge logicielles

- Rassemblent un outil de gestion de version avec des services complémentaires utiles à la vie d'un projet de développement.
- Par exemple, trackers, gestion de projet, hébergement web, forums, mailing lists, ...
- Exemple de forges :
  - <http://sourceforge.net>
  - <http://code.google.com>
  - <https://github.com/>
  - <https://bitbucket.org/>
  - <http://www.berlios.de/>
  - Redmine
  - Trac

Et bien entendu

- <http://forge.istic.univ-rennes1.fr/>

# Outils de complémentaires (2)

- Outils de build
  - Permet l'enchaînement des différentes activités de développement
    - Compilation, link, tests (unitaires, fonctionnels, intégrations, qualité,...),
  - Aident à la gestion de dépendance entre vos modules
  - Souvent généraliste mais orientés pour une famille de langage
- Ex: Maven, Gradle, Ant, Make, Cmake, ...
- Certains permettent d'aider aux changements de version pour les releases.

**maven**





# Gestion de binaires/package : exemple repository Maven

- Couplé à l'outil de build (ou à une distribution linux)
  - Conserve les versions liées aux releases du produit
- Permet de faire des dépendances entre binaires plutôt qu'entre sources
  - Pas besoin de récupérer tous les sources du projet pour travailler sur un de ses composants.

**maven**

# 6.2

Aller plus loin

Structurer son code pour aider la  
gestion de version

# Back to the 3 Dimensions of Software Configuration Management: [Estublier et al. 95]

- The Revision dimension
  - Evolution over time
- The Concurrent Activities dimension
  - Many developers are authorized to modify the same configuration item
- The Variant dimension
  - Handle environmental differences
- Even with the help of sophisticated tools, the complexity might be daunting
  - Try to simplify it by reifying the variants of an OO system



# Variants in Software Systems

- Hardware Level
  - Heterogeneous Distributed Systems
  - Peculiarities in Target Operating System
  - Compiler Differences
  - Range of Products
  - User Preferences for GUI
  - Internationalization
- $V_i = 16$
  - $V_p = 4$
  - $V_n = 8$
  - $V_g = 5$
  - $V_l = 24$

- **Number of Variants =  $V_p * V_n * V_g * 2^{V_i + V_l - 2}$** 
  - That's 43,980,465,111,040 possible variants

# Traditional Approaches

- Patch the executable
- Device Drivers
  - source level, link time, boot time, on demand at runtime
- Static Configuration Table
- Conditional Compilation / Runtime Tests

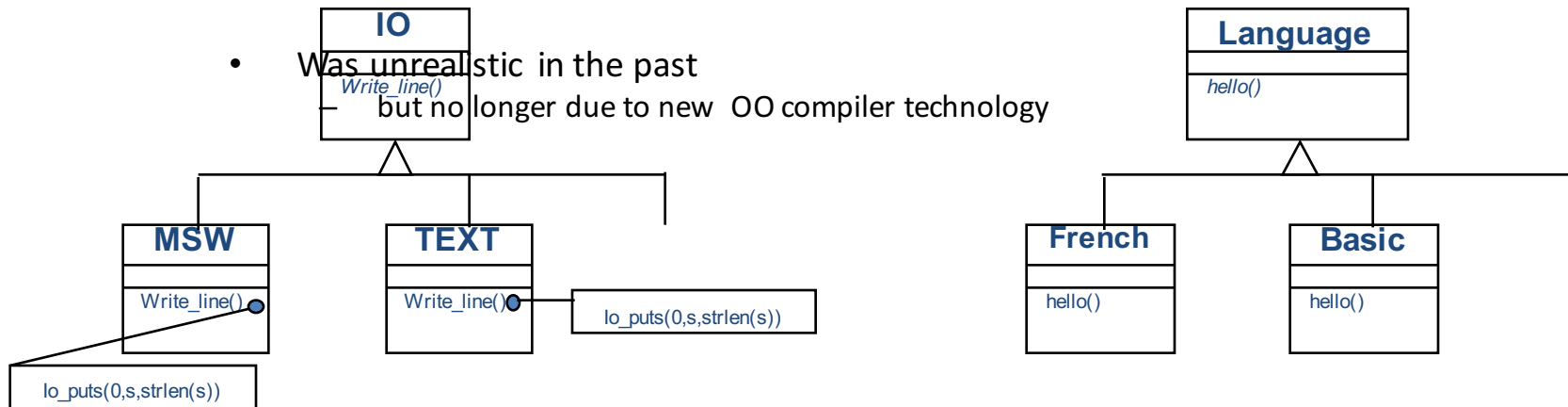
```
If (language == french) {  
#ifdef MSW  
    io_puts(0,"Bonjour",7);  
#elif TEXT  
    printf("Bonjour\n");  
#endif  
} else {  
#ifdef MSW  
    io_puts(0,"Hello",5);}  
#elif TEXT  
    printf("Hello\n");}  
#endif
```

- **Static and Dynamic configuration information intermingled**
- **Hard to change your mind on what should be static or dynamic...**

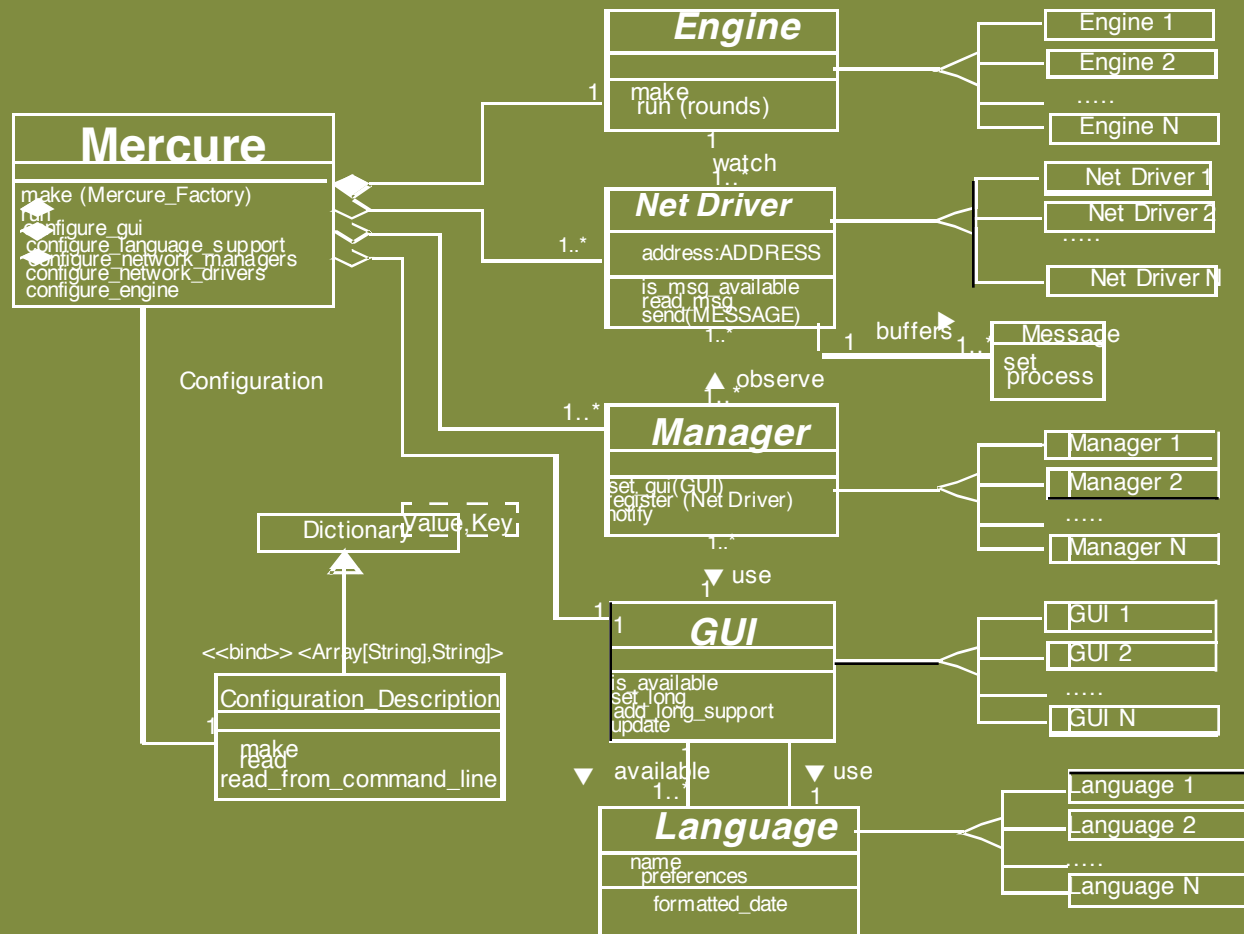
# Basic Idea

- Abstract the Intent
  - `io.write_line(language.hello)`
- Rely on Dynamic Binding for the Details
  - Don't care now for static/dynamic distinction

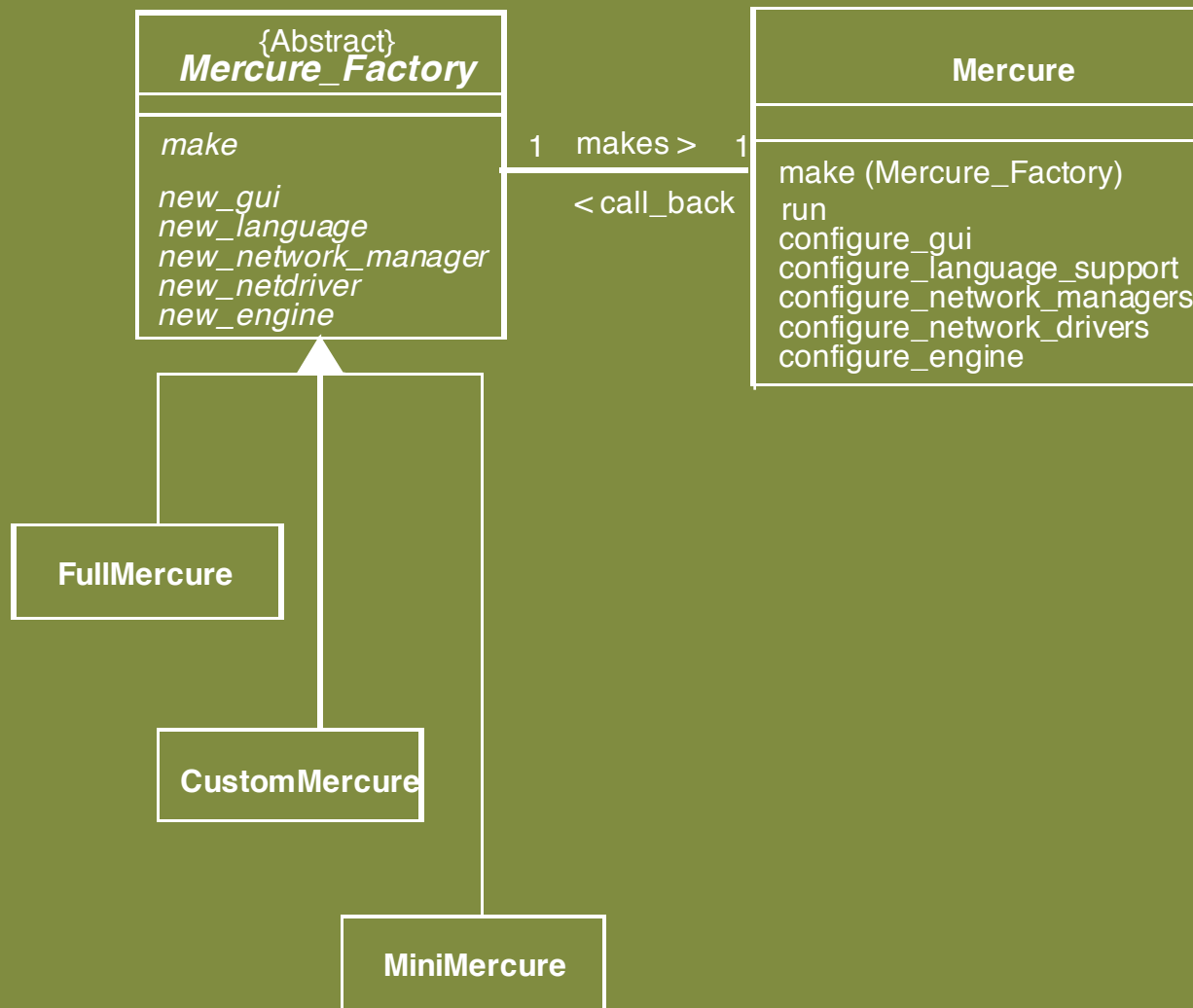
- Was unrealistic in the past
  - but no longer due to new OO compiler technology



# Case Study: The Mercure Software

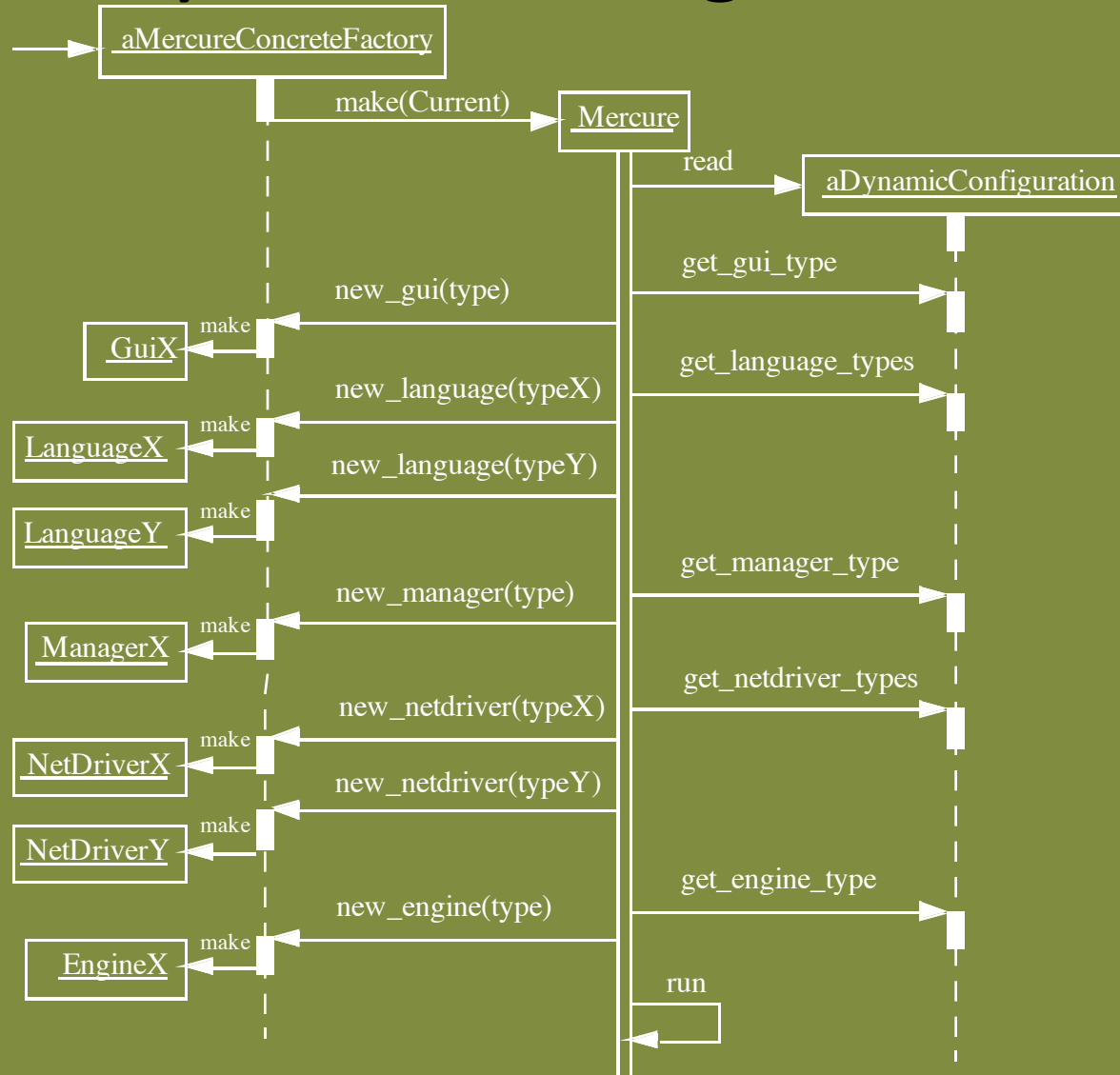


# Reifying the Variants





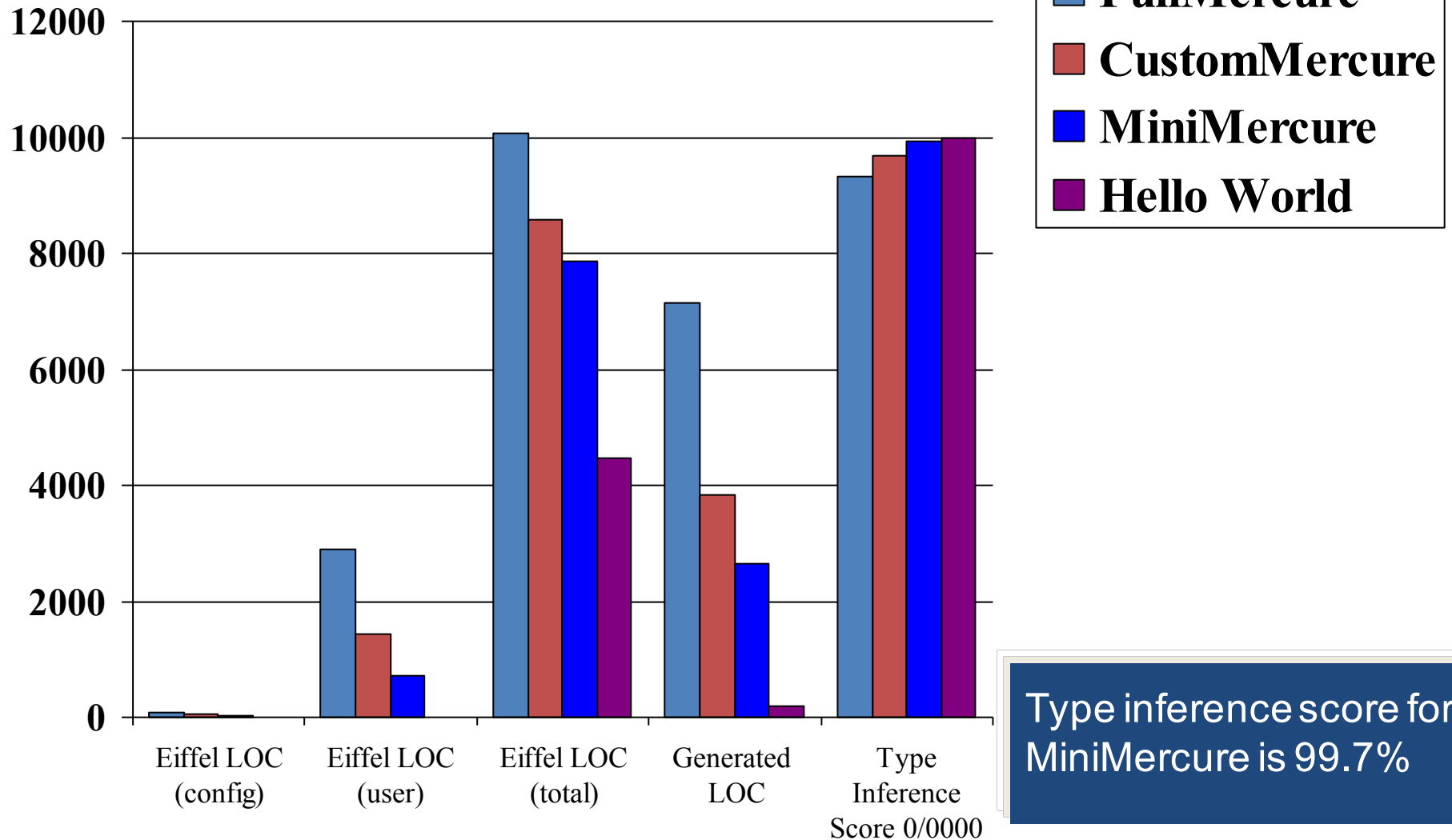
# Dynamic Configuration



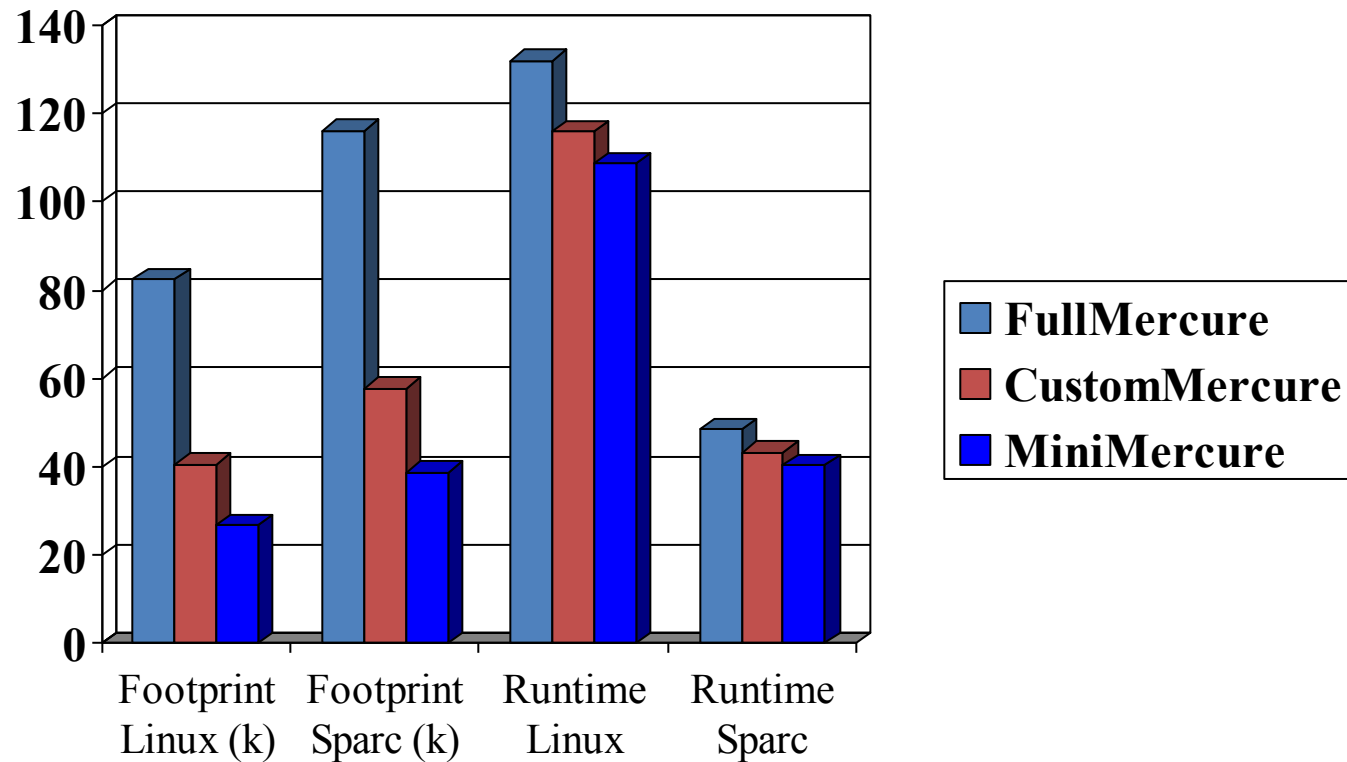
# Compilation Technology

- By limiting the range of variants available from a given Concrete Factory:
  - The compiler may know the set of *living* classes
    - Type Inference (special case of Partial Evaluation)
  - Generate specialized code for the product
    - When only one *living* class for an abstract varying part:
      - Dynamic binding replaced with direct call (and even inlining)
    - When only a few *living* classes
      - Dynamic binding replaced by *if then ... else*
  - Implemented in e.g., GNU SmallEiffel
- All static configuration issues kept encapsulated in the Concrete Factory

# Taille de code



# Performances d'exécution



Dead code removal

Automatic static binding

# Impact on SCM

- Limitations
  - Only one SCM dimension is dealt with
    - At least revision control is still needed (+cooperative versioning)
    - e.g. SVN
  - Compilers may do optimizations based on type inference only if granted access to the full code
    - is this really a problem? Modularity, compilation speed and source protection can be dealt with other techniques...
- Advantages
  - notion of product family *much more* concrete than with *diff*
  - concrete factories easy (albeit tedious) to program
  - configuration done within the target OO language
    - no need to learn another complex language just for SCM

# Conclusion

# Conseils complémentaires

- Ne pas utiliser un outil de gestion de version est une faute professionnelle !
- Ce ne sont que des outils de gestion de versions, ils ne dispensent pas d'une bonne organisation du projet et du développement
- Ne pas attendre trop longtemps pour se resynchroniser
- quelques idées :
  - noter la version des outils utilisés (historique ou fichier spécial)
  - Toujours identifier les versions distribuées (tag, ...)
  - avoir en permanence une version extraite en lecture seule (éventuellement compilée) utile pour la consultation des documentations, tests de non régression, ...

# Limitations de tous les outils de gestion de version

- Les formats spécifiques sont peu ou mal gérés sur les accès concurrents
  - Pb des conflits sur des fichiers "binaires" (ex: word !!)
  - Différence entre le diff syntaxique et le diff sémantique



# Quelques pointeurs

- Subversion
  - <http://subversion.tigris.org/>
  - <http://svnbook.red-bean.com/>
- Gestionnaires distribués
  - [http://en.wikipedia.org/wiki/Distributed\\_revision\\_control](http://en.wikipedia.org/wiki/Distributed_revision_control)
  - <http://www.jres.org/paper/2.pdf>
- Git
  - <http://git.or.cz/>
  - <http://git.or.cz/course/cvs.html>
  - <http://git.or.cz/course/svn.html>
  - <http://www.kernel.org/pub/software/scm/git/docs/user-manual.html>
  - [http://jonas.iki.fi/git\\_guides/HTML/git\\_guide/](http://jonas.iki.fi/git_guides/HTML/git_guide/)





# Questions



# Questions



Merci