# Chapitre 3: Les systèmes de gestion de version

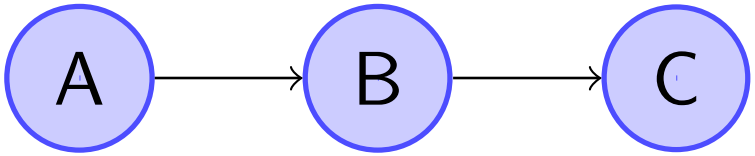
***Ce document doit être complété par les notes de cours !***

# C’est quoi ?

- Logiciels permettant de gérer l’historique des modifications d’un ensemble de documents.
- Typiquement : les codes source d’un logiciel.
- Mais aussi : documentation, site web, des fichiers de configuration etc.
- Fonctions de base :
  - conserver un historique des modifications
  - permettre travailler à plusieurs (verrous, gestion des conflits)
  - permettre les modifications en parallèle (branches)
  - garantir la sécurité (intégrité, disponibilité, confidentialité)

# Concepts de base

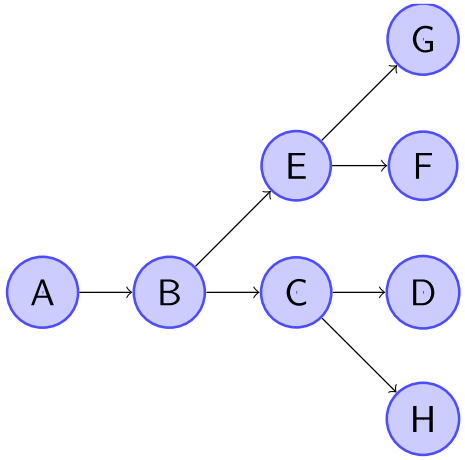
- **Dépôt** (repository)
  - Répertoire ou espace de stockage : conserve l'historique des modifications
- **Révision**
  - Chaque état des données a un identificateur unique → **révision**.
  - également appelée **commit** par abus de langage.




  - Dans un projet, on retrouve une séquence ordonnée (dans le temps) de révision

# Concepts de base (suite)

- **Branches**
  - une ligne d'évolution divergeant de la ligne d'évolution courante, celles-ci se poursuivant indépendamment des autres branches.
  - Pourquoi faire ?
    - corriger un problème sur une ancienne version
    - développer plusieurs idées en parallèle
    - Gérer sa propre version d'un logiciel
    - fusionner après une divergence.



# Concepts de base (suite)

- **Tags**
  - Marques symboliques sur une révision.
  - Permettent de définir les versions du projet.
  - Permettent de nommer des branches.

# Travailler à plusieurs

- Travailler à plusieurs sur le même code
  - Pas de verrou sur les sources.
  - Chacun a sa propre copie.
- SGV permet la gestion des conflits:
  - d'abord intégrer les modifications des autres
  - fusion automatique
  - détection des conflits → résolution à la main
  - Bloquer **commit** avant la résolution du conflit

# Autres fonctions d'un SGV

- Visualisation de l'historique sous diverses formes
- Exécution automatique de scripts avant/après commit
  - Tests de validation,
  - Envoi d'e-mail après commit.
- Annotation du code avec les contributions
- Import/export vers d'autres SGV

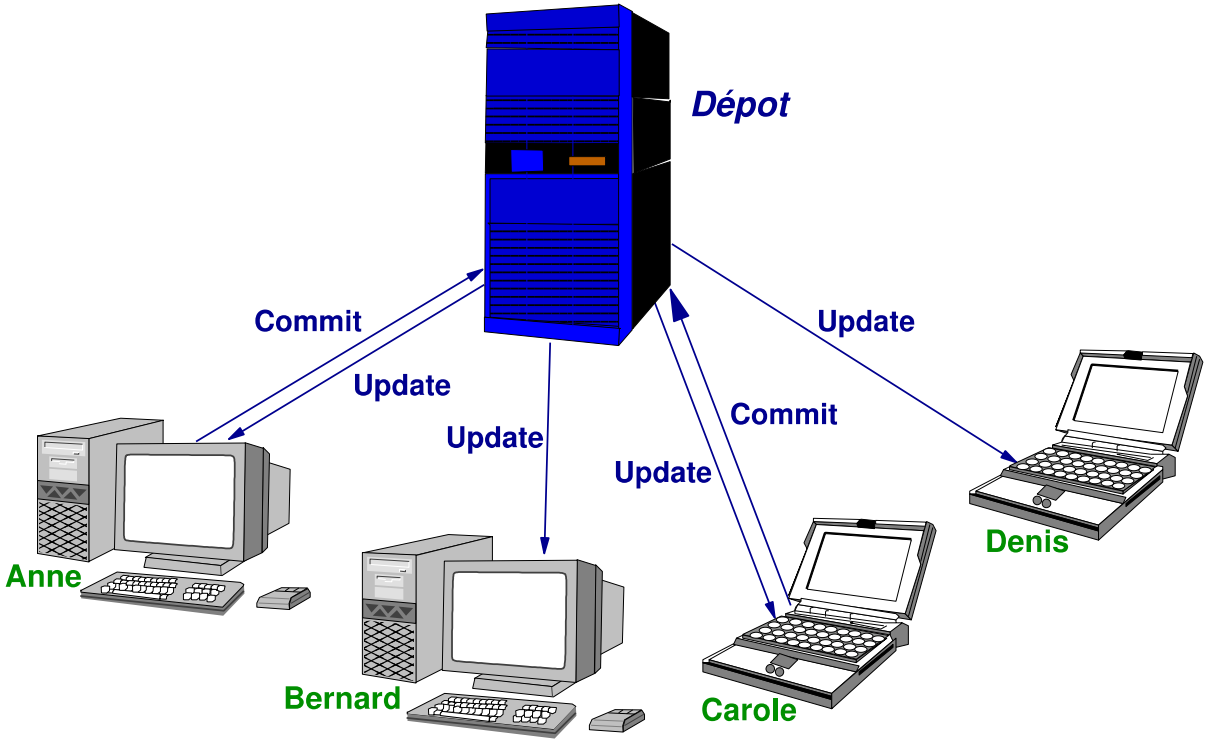
# Modèles de fonctionnement

- **Local**
  - Fonctionne dans un système de fichiers local. Pas de réseau.
    - SCCS, RCS,…
- **Client/Serveur (ou centralisé)**
  - Un serveur centralise le dépôt, accessible à distance.
    - CVS
    - Subversion
- **Distribué**
  - Multiples copies du dépots, branches locales.
    - bitkeeper, monotone, arch, darcs
    - mercurial, git, bazaar

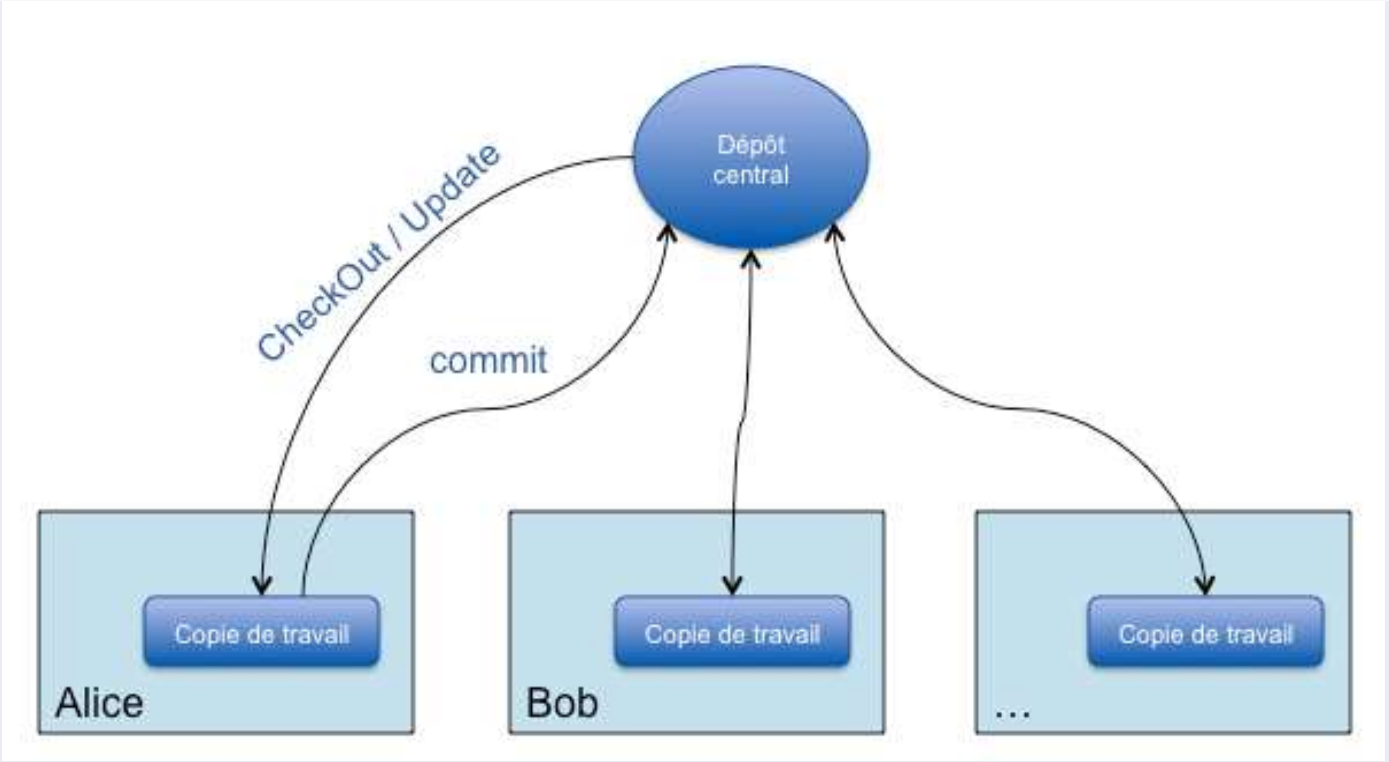
# Modéle centralisé

- Dépôt stocké dans un endroit partagé
  - par le système de fichiers
  - par un mécanisme réseau (rsh/ssh ou protocole dédié)
- Plusieurs copies de travail en parallèle : opérations de fusion.
- Nécessaire d'avoir la connexion au dépôt pour « committer ».

# Modèle centralisé (suite)



# Modèle centralisé (suite)

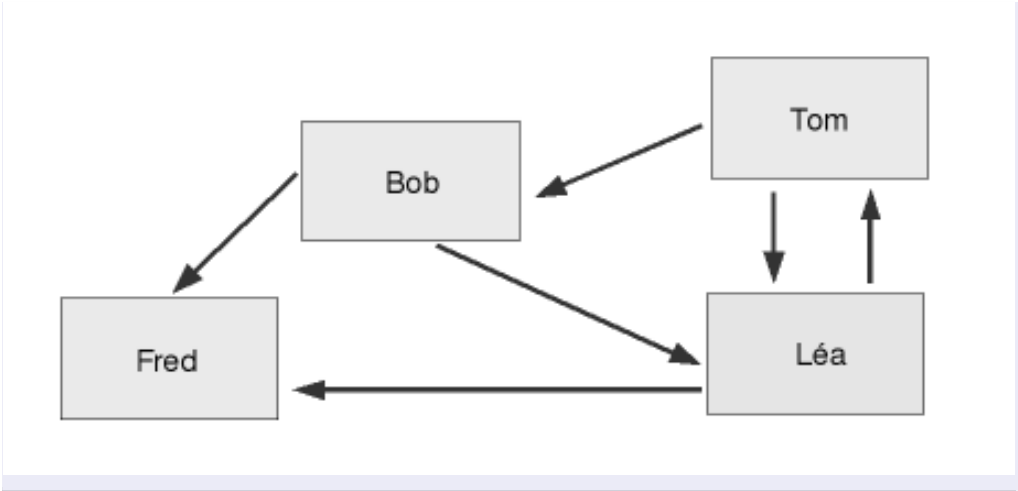


# Modèle centralisé (suite)

- Avantages
  - Technologie éprouvée
  - Largement disponible
- Inconvénients
  - Échange entre les dépôts impossible
  - Échange entre les copies locales impossible
  - Travail hors connexion impossible
  - Temps de mise à jour long pour de gros projets
  - Et si le serveur tombe en panne ?

# Modèle décentralisé

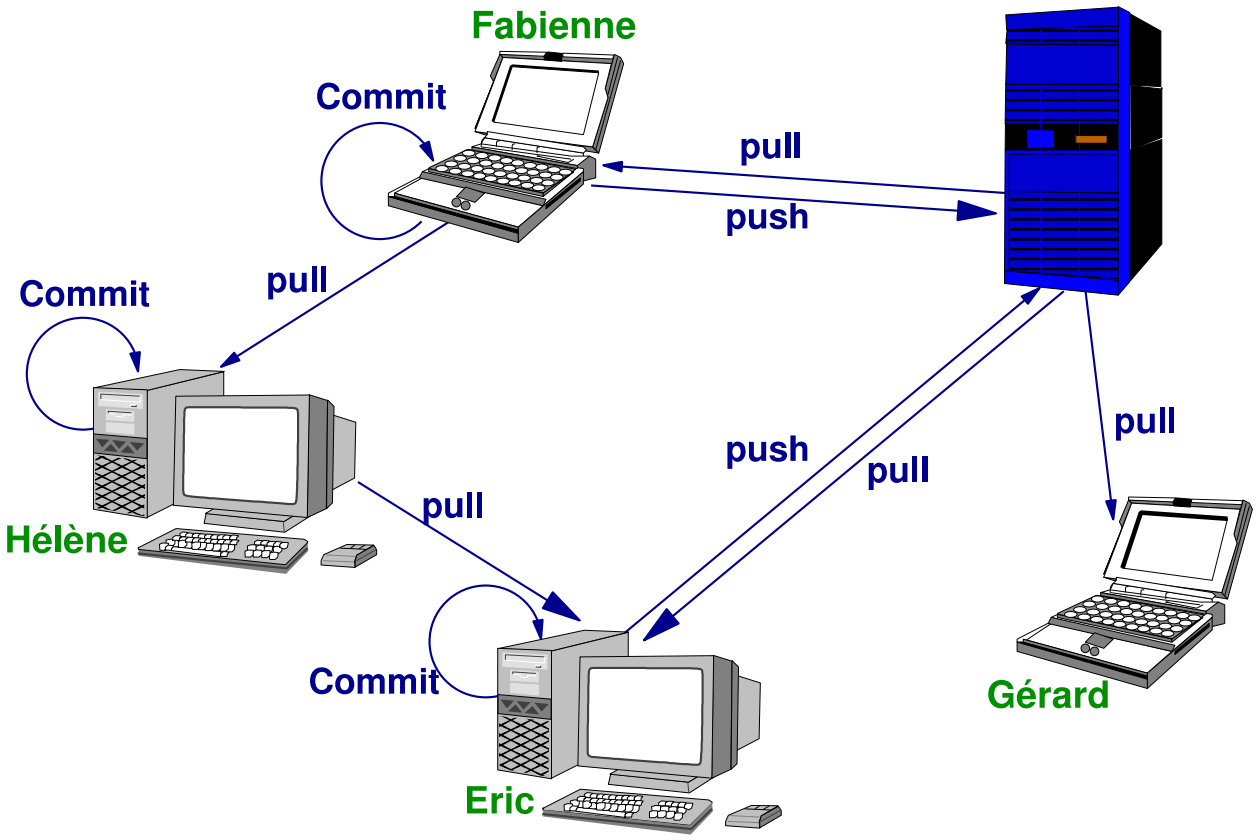
- Objectifs : pallier les limites/problèmes des systèmes centralisés
  - Pouvoir utiliser ce système *hors connexion*
  - Ne pas être dépendant d'un dépôt centralisé (panne, temps, . . . )
  - Pouvoir échanger ses fichiers avec une partie des développeurs
- Chaque développeur possède son propre dépôt (et sa copie de travail)



# Modèle décentralisé (suite)

- Pas de dépôt centralisé
  - Chaque développeur a sa copie avec ses branches privées
- Opérations push/pull : synchronisation avec les autres dépôts.
- Simplification de la fusion de branches en gardant l'historique des fusions.
- À une Influence sur la philosophie de développement :
  - plus de liberté,
  - Mais il y a un risque de dispersion...

# Modèle décentralisé (suite)



# Modèle décentralisé (suite)

- Un système distribué peut devenir anarchique :
  - pas de notion de branche “principale” ou “de référence”
  - chacun résout les conflits à sa manière...

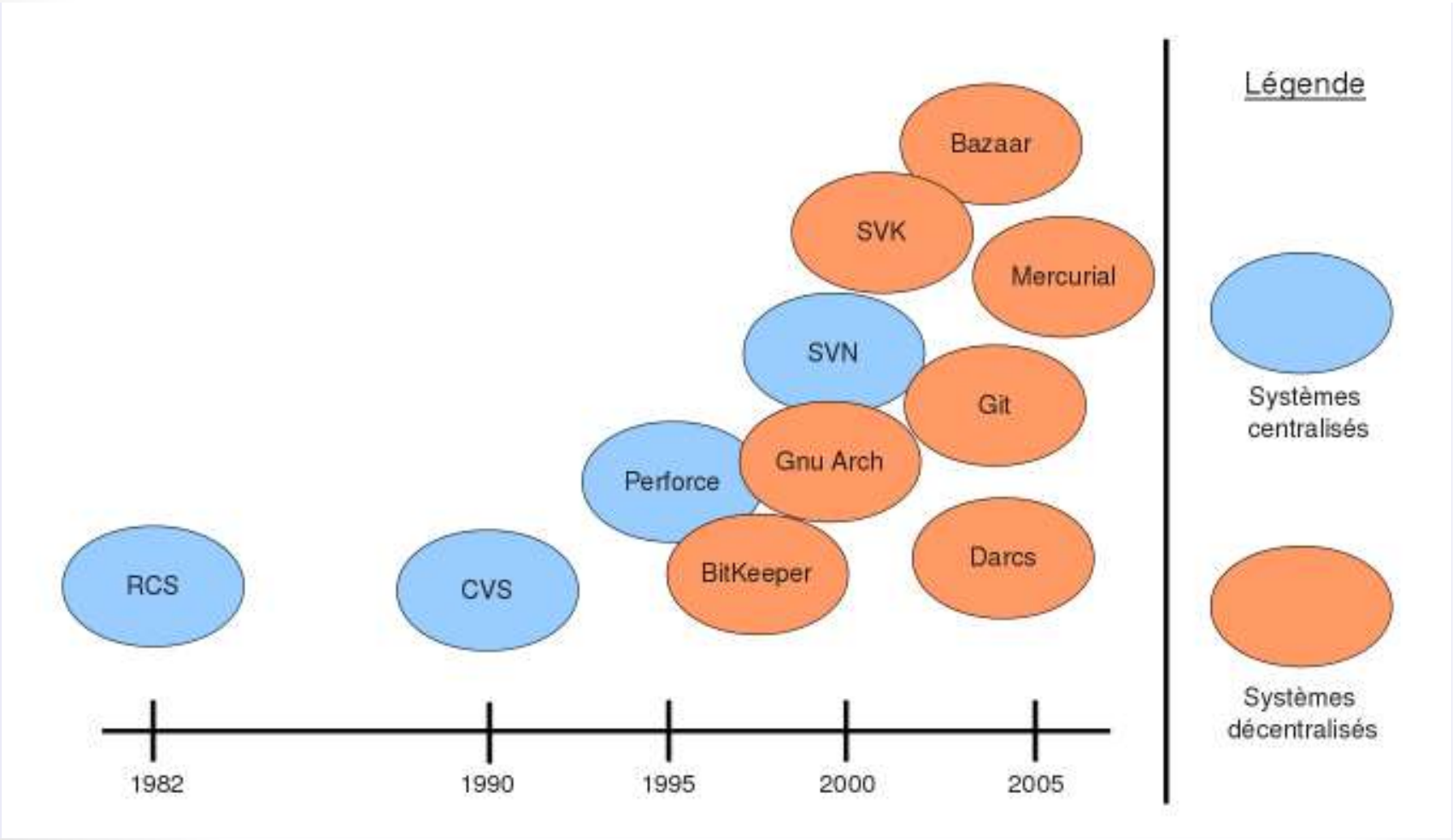
→Nécessite une politique de management :

- Définir une branche de référence et nommer un responsable
- Définir une nomenclature pour les branches partagées
- Inciter les développeurs à fusionner leurs travaux

# Quel SGV choisir ?

- Technologie en pleine évolution
- De nouveaux systèmes apparaissent régulièrement → Vaste choix
- Critéres à considerer
  - Pérénité : systèmes *leaders* vs. systèmes *émergents*
  - Interfaces graphiques
  - Portabilité (multi OS)
  - Sécurité
  - Documentations abondantes

# Quel SGV choisir ?



# SVN et Git

- Logiciels libres
- Multi OS (linux, windows, MacOS, ...)
- Très répandus (documentations abondantes, support, ...)
- Sécurisés (ssh, https, ...)


# Subversion (SVN)

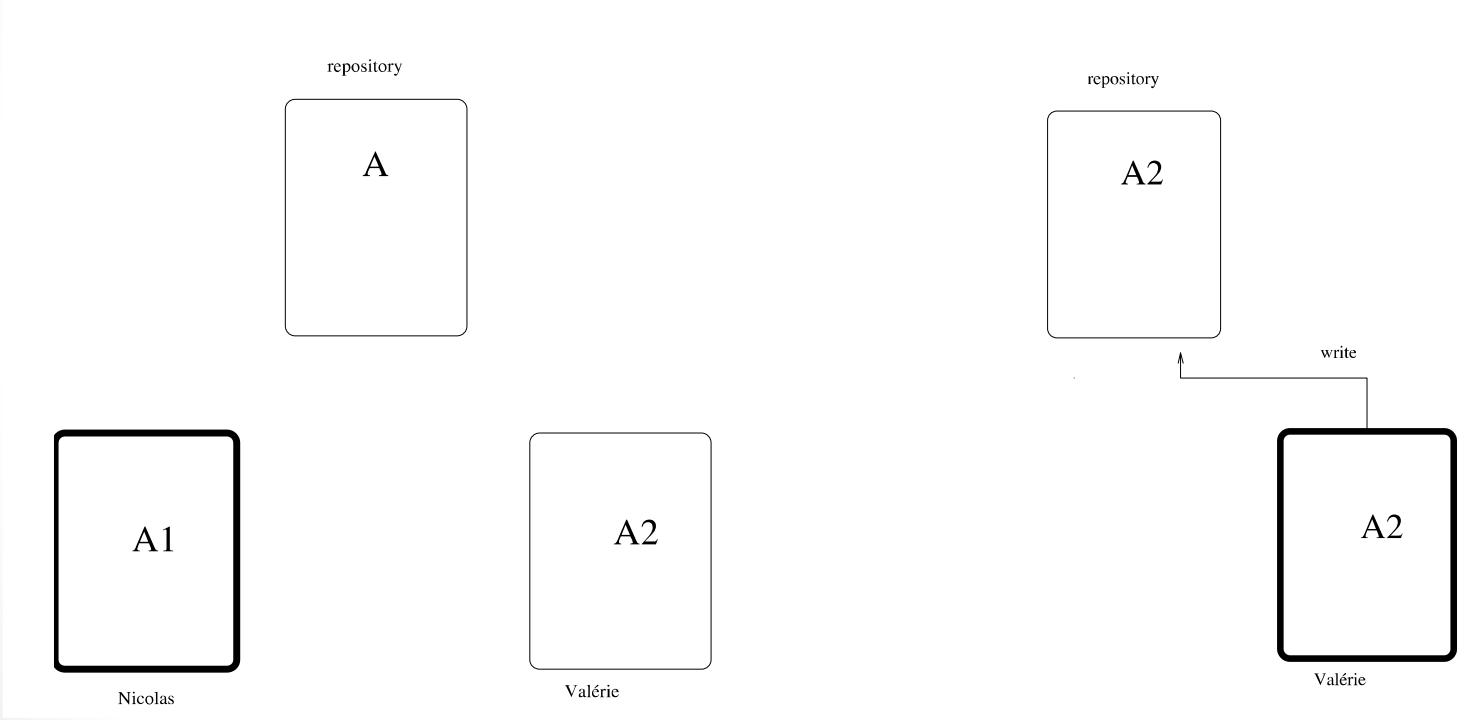
- Un des SGV les plus utilisé
- **Systéme de gestion de version centralisé**
- Documentations très riches et forums actifs
- Interfaces graphiques
  - Linux : rapidsvn, kdesvn, esvn, Qsvn, ...
  - Windows : intégré à l'explorateur via le plugin TortoiseSVN
- Proposé dans les Forges et intégré dans certains IDE (Eclipse, Kdevelop)

# SVN: fonctionnement

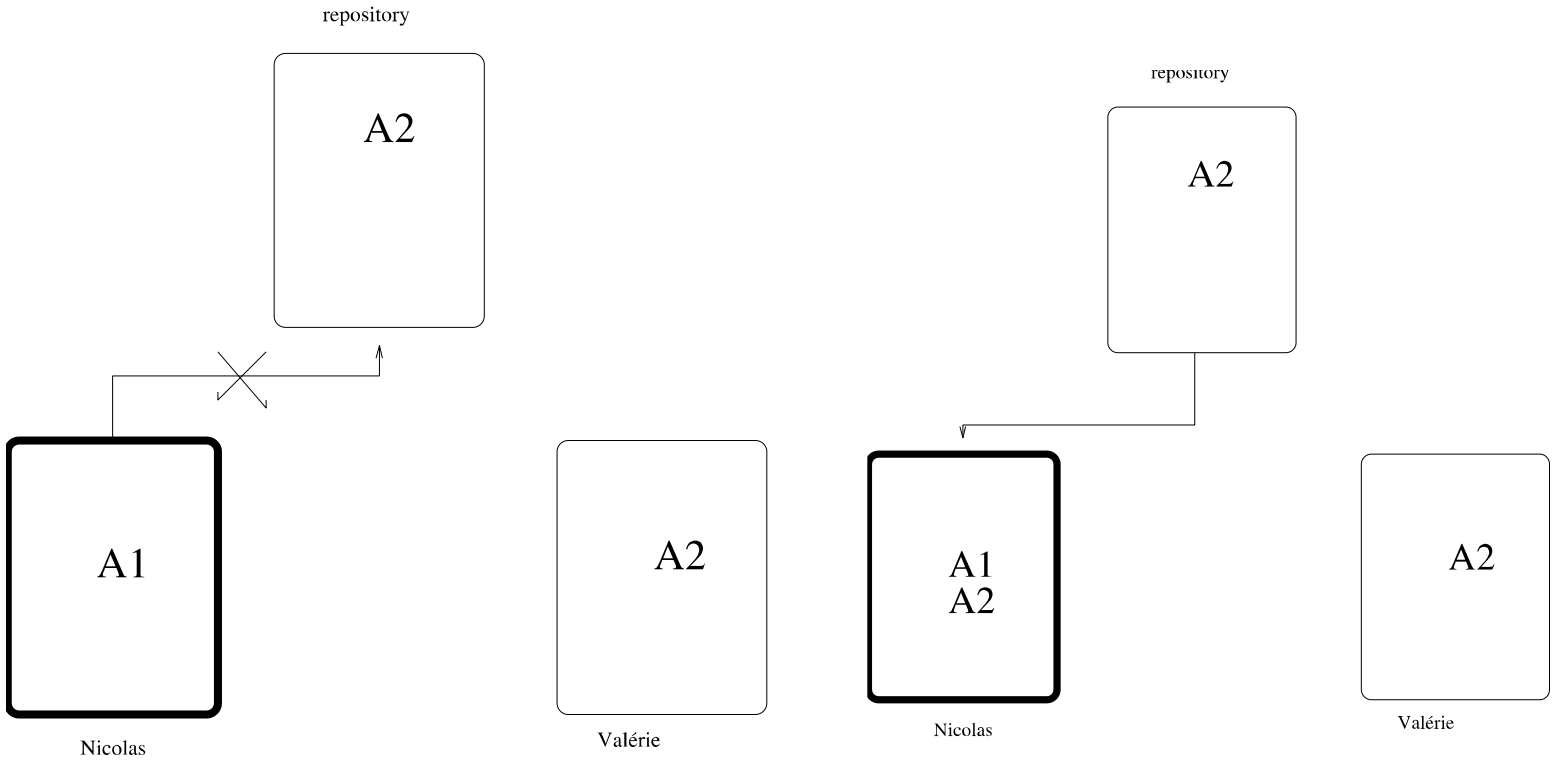
- svn opére sur des fichiers texte
  - Pas nécessairement avec l'extension .txt
  - Des fichiers de code java ou C (.java, .c)
  - mais pas des fichiers word !
- Les développeurs doivent travailler en concertation
- Ils ne modifient pas la même partie du même fichier
- svn fusionne deux modifications si celles-ci concernent des parties différentes du fichier ;
  - Sinon, il détecte un conflit
  - Réglé par concertation entre les développeurs.

# SVN: fonctionnement

- Modification simultanée d'un même fichier

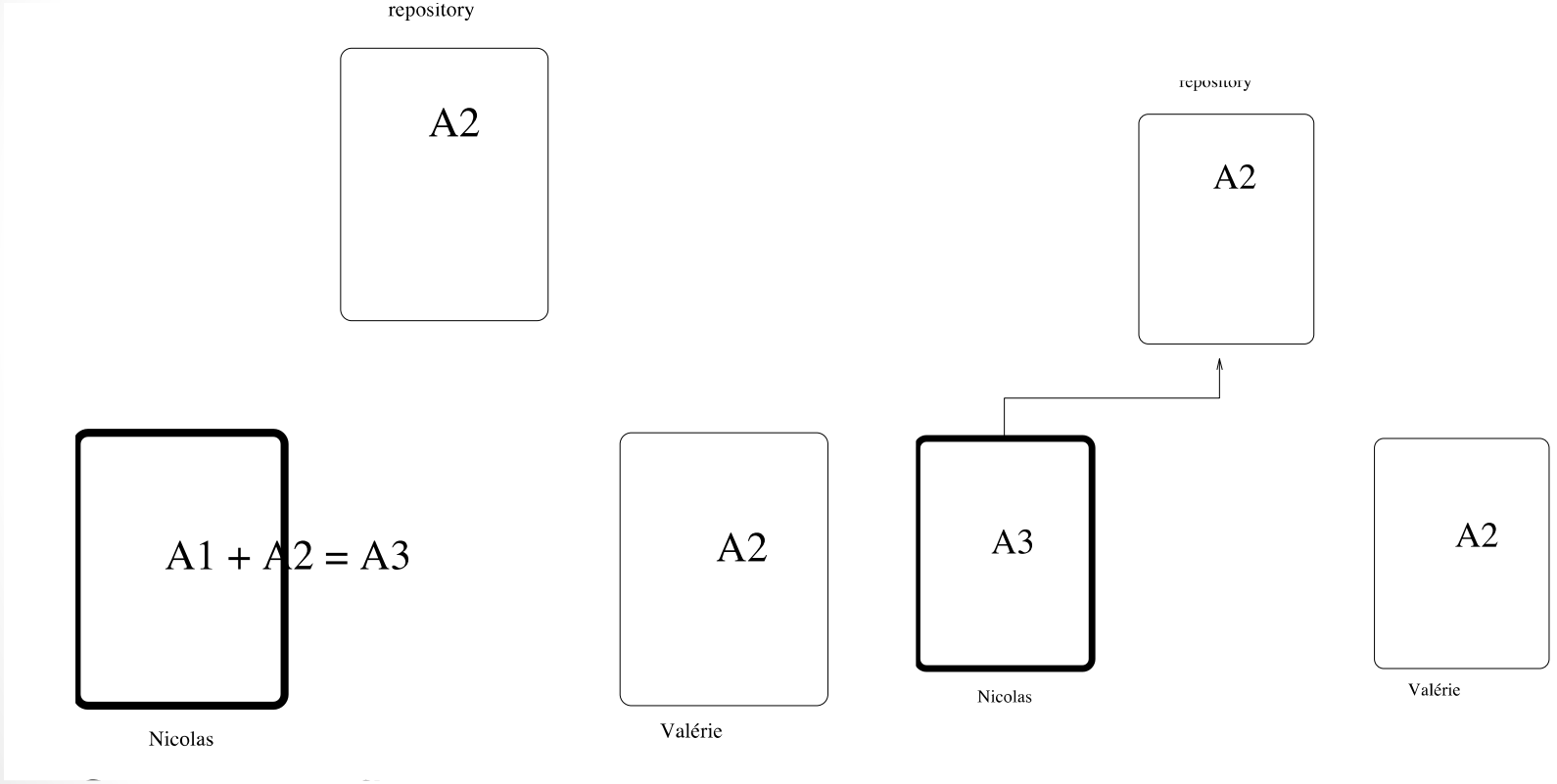


# SVN: fonctionnement




(3) (4)

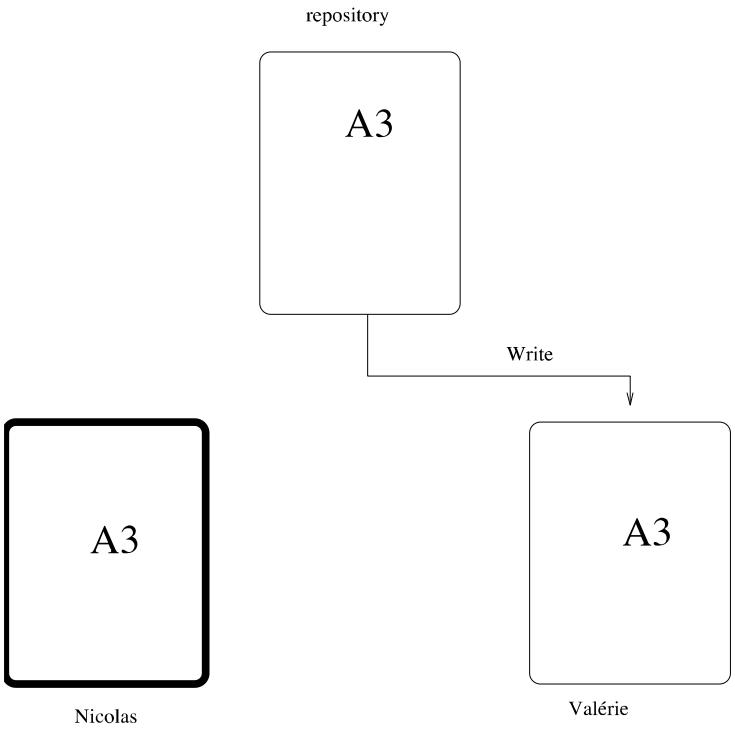
# SVN: fonctionnement




(5)

(6)

# SVN: fonctionnement

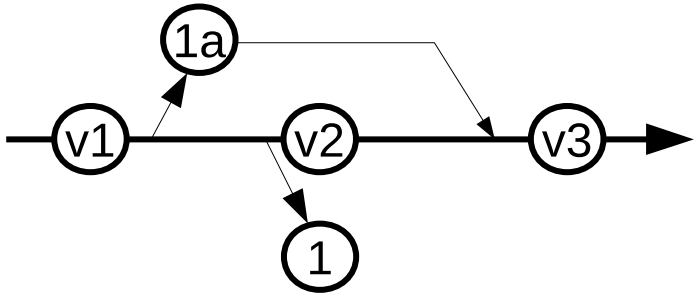



(7)

# SVN: structure

- SVN contient un ensemble de révisions de fichiers.
- Le contenu du répertoire projet contient les sous-répertoires suivants :
  - **trunk**
    - contient le code stable représentant la version destinée aux utilisateurs
    - C'est la branche principale
  - **branches**
    - contient les codes pour les nouvelles versions pendant les tests et développements de nouvelles fonctions.
    - Lorsqu'une branche est créée, aucun fichier n'est déplacé
    - des liens symboliques permettent d'accéder au code de la branche.
    - Dès l'édition d'un code, le fichier est copié puis modifié dans la branche correspondante.
  - **Tags**
    - contient une version figée des fichiers du trunk
    - Même si le trunk avance, la version contenue dans un tag ne sera pas modifiée.
    - Permet de conserver une version particulière d'un ensemble de fichiers et par conséquent, une version stable de l'application.
    - Il ne faut donc effectuer aucune modification dans les tags

# SVN: structure




- **Le trunk** est représenté par les chiffres de la version stable v1, v2, v3
- **Une branche** peut-être créée pour résoudre un bug sur l'application (v1a).
  - Lorsque le bug est résolu, on réintègre les développements de cette branche dans le trunk
- **Tag** permet de conserver une trace des fichiers utilisés dans la version v1
  - en cas de problème sur des fonctions qui étaient opérationnelles et qui dans la version v2 ne marchent plus.

# Logiciels serveurs SVN

- Svnserve
  - un serveur léger qui utilise un protocole TCP/IP spécifique.
  - Destiné aux petites installations
- Serveur HTTP d'Apache
  - les dépôts sont accessibles aux clients via l'installation d'un module WebDAV.
  - utilisation du protocole SSL qui permet de sécuriser l’acces via l'Internet.
    - SSH peut être plus contraignant et moins sécurisé puisqu'il crée des utilisateurs de plein droit sur le serveur.
    - Permet de tracer les connexions des clients.

# Logiciels serveurs SVN (suite)

- Sur windows VisualSvn Server
  - permet un packaging de Apache + Subversion
  - avec un paramétrage automatique
  - une console d'administration graphique du serveur pour gérer les droits

# Logiciels clients SVN

- L'accès au serveur subversion peut se faire en utilisant
  - son protocole natif (port TCP par défaut : 3690),
  - une connexion HTTP (le serveur doit alors prendre en charge les extensions WebDAV).
- Les deux modes d'accès peuvent être sécurisés au niveau de la couche de transport (tunnel SSH ou HTTPS)
  - recommandé dans le cas de serveurs sur un réseau public comme Internet.
- Plusieurs logiciels avec une interface graphique existent
  - sous forme d'exécutable standalone
  - et également de plugins pour Environnement de Développement Intégré (EDI) comme Eclipse.