Gestion de version avec GIT

Bruno Mermet
Université du Havre
2017

Préliminaires
Installation de git

- Allez à l'adresse suivante :
  - https://git-scm.com/download/linux
- Pour les « roots » :

  Exécuter la commande indiquée

  exemple : sous Ubuntu, apt-get install git
- Pour les autres :
  - Cliquer sur le lien « download a tarball »
  - Téléchargez la version de votre choix (`git-1.9.1.tar.gz`)
  - Décompresser l'archive dans un répertoire $GIT
  - `cd $GIT ; ./configure ; make`
  - `export PATH=$GIT:$PATH`

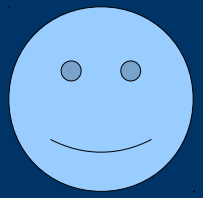
Plan

- Principes de la Gestion de Version
  - Gestion de version mono-document (T3) ⟹
  - Gestion de version multi-documents (T12) ⟹
- Fonctionnement de GIT (T21) ⟹
- Dépôt commun et application à Github
  - Les bases (T72) ⟹
  - Fork et pull request (T86) ⟹
- Bilan, conseils, astuces (T100) ⟹

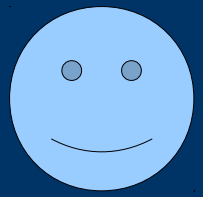
La gestion de version Mono-document

- Permettre d'avoir un historique des différentes versions d'un document avec :
  - Explications
  - Retour en arrière possible
  - Identification des versions
- Permettre le travail à plusieurs
- Centraliser le dépôt des documents (sauvegardes)

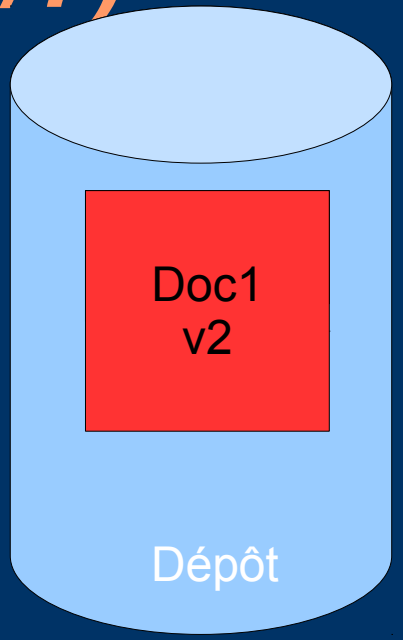
GV mono-document : Conflits (1/7)



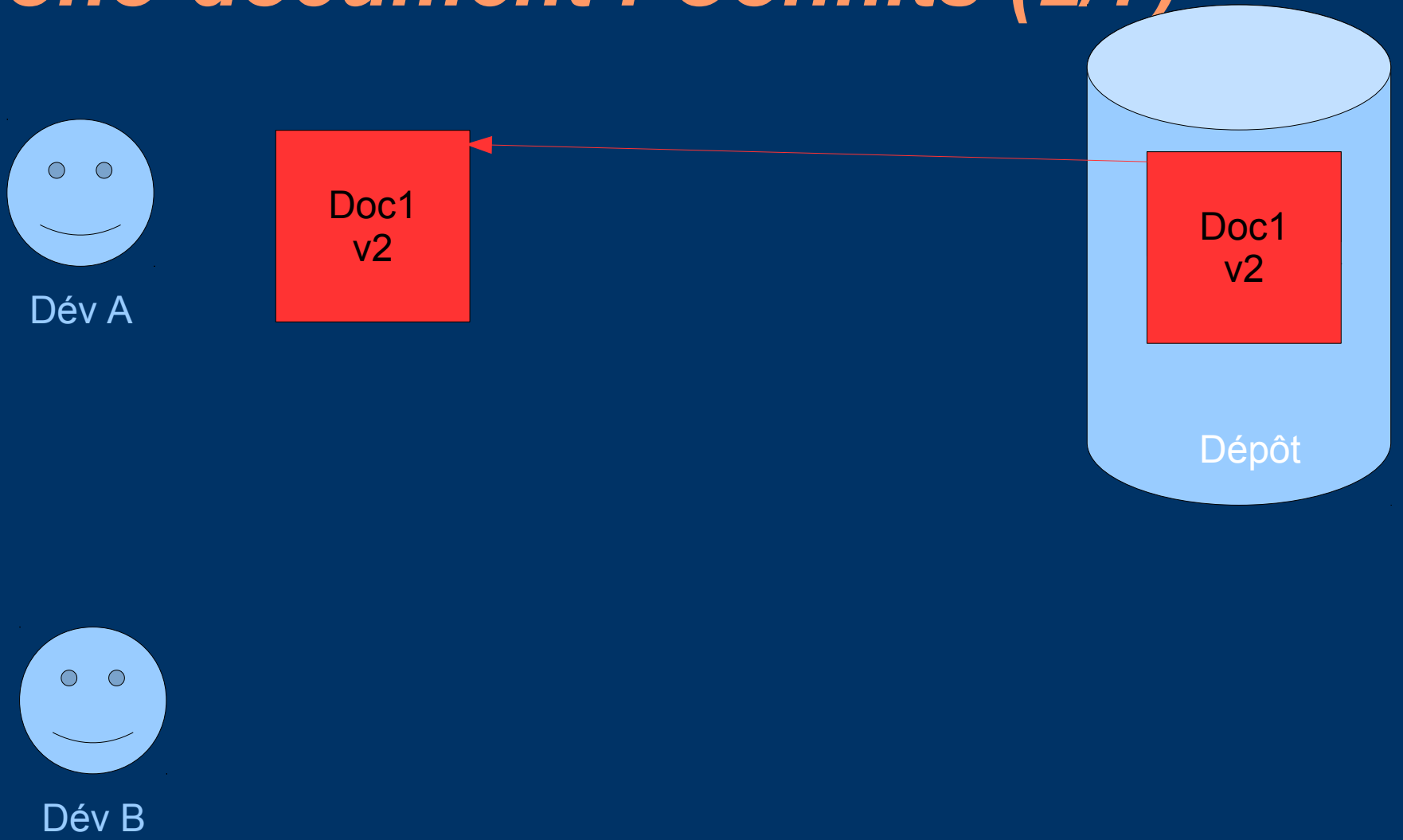
Dév A



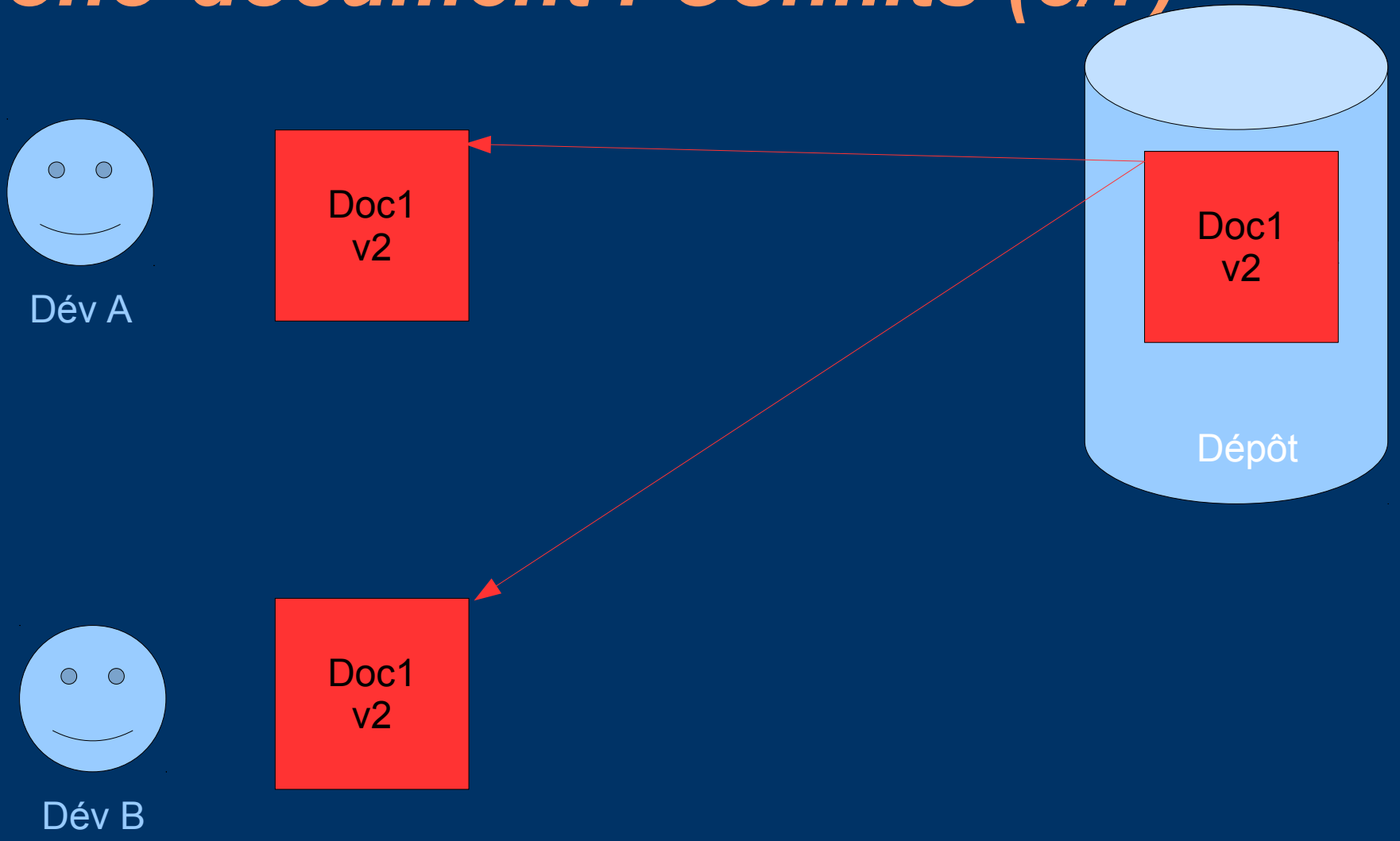
Dév B



GV mono-document : Conflits (2/7)

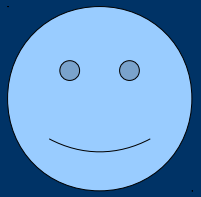


GV mono-document : Conflits (3/7)

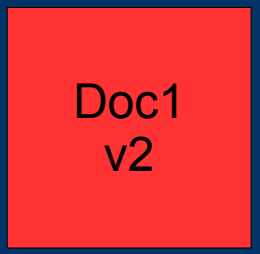


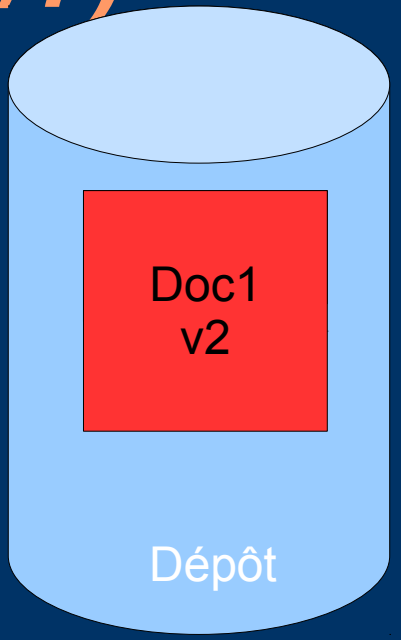
GV mono-document : Conflits (4/7)

Dév A



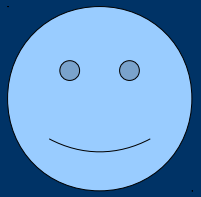
Dév B



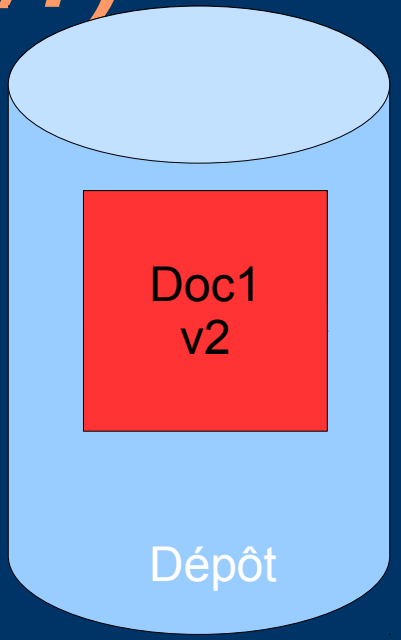


GV mono-document : Conflits (5/7)

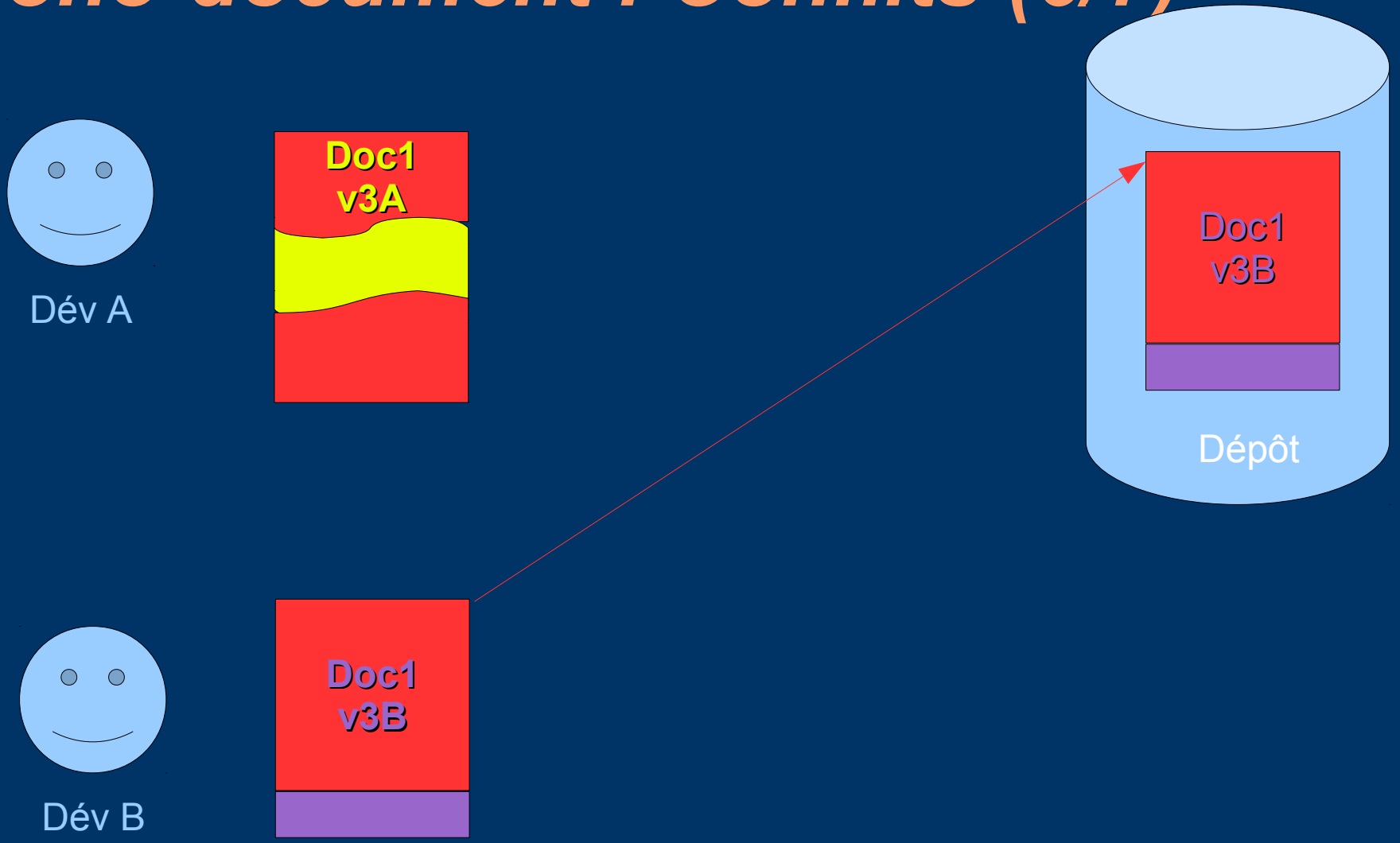
Dév A



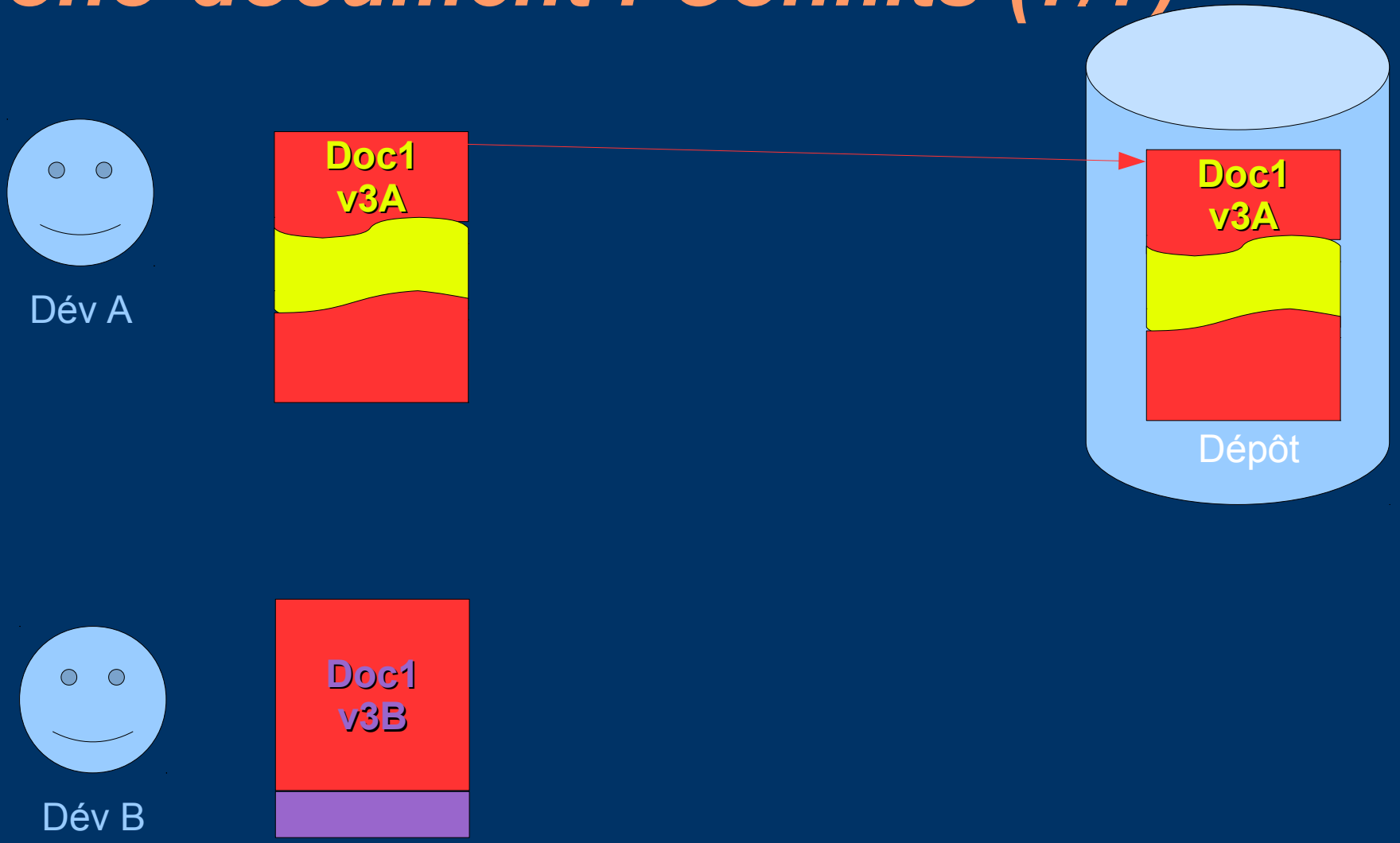
Dév B



GV mono-document : Conflits (6/7)



GV mono-document : Conflits (7/7)



GV mono-document : solution

- Gestion des conflits :

  Utilisation de verrous pour modifier un document :

  - Check-out document (si pas de verrou sur le document, met un verrou puis le récupère)
  - Édition du document en local
  - Check-in document (met la nouvelle version du document dans le dépôt, libère le verrou, et supprime le document de la zone de travail)

- Implantations :
  - SCCS
  - RCS
- Stockage :
  - Stockage différentiel des fichiers textes via l'utilitaire *diff*.

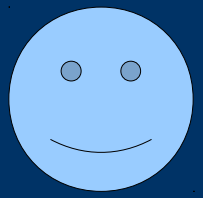
GV multi-documents

- Pourquoi :
  - Un projet = un ensemble de documents
  - Une version d'un projet peut correspondre à des versions différentes de chacun des documents le composant
  - Solution des verrous inexploitable
- Implantations :
  - CVS, reposant sur SCCS/RCS
  - Subversion (SVN), évolution de CVS
  - Git
  - Autres (Mercurial, ClearCase, …)

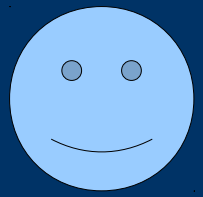
GV multi-documents : conflits

- Pas de verrous :
  - La récupération (update) d'une version est libre
  - Les conflits sont détectés à la sauvegarde (commit) sur le dépôt :
    - Pas de conflit, sauvegarde effectuée
    - Conflit, sauvegarde refusée
  - Les conflits sont corrigés en local, lors de la récupération d'une nouvelle version

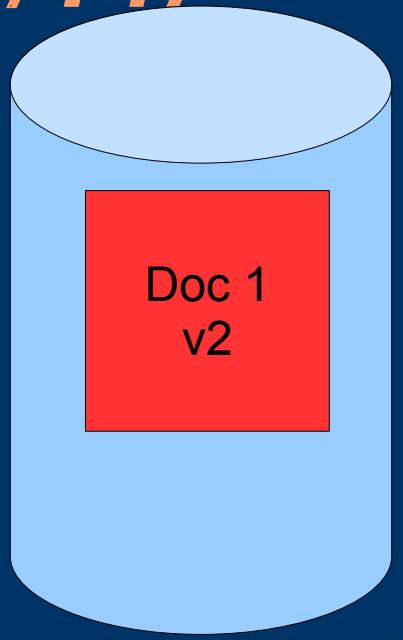
GV multi-documents : conflits (1/14)



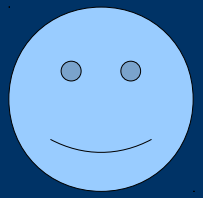
Dév A



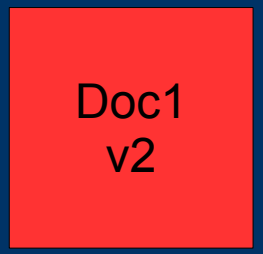
Dév B

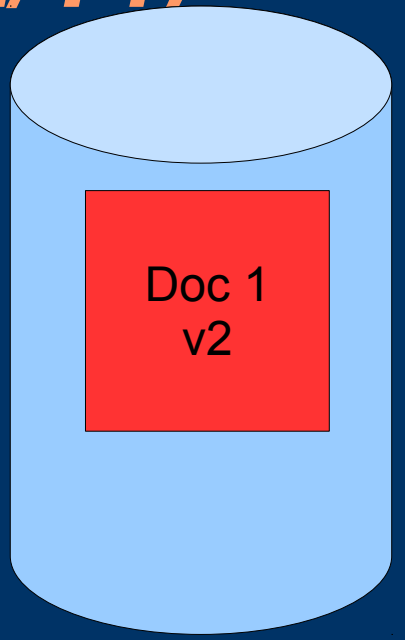


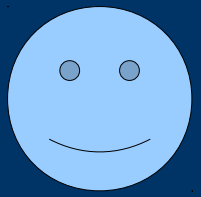
GV multi-documents : conflits (2/14)



Dév A

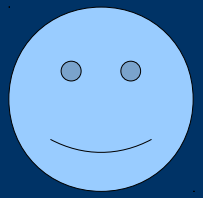




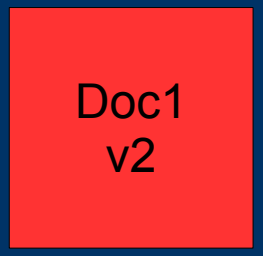


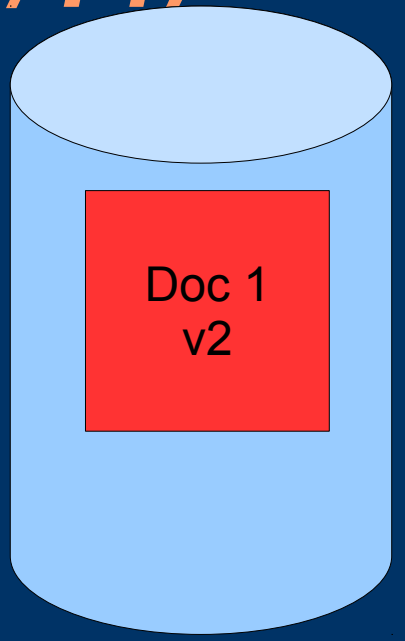
Dév B

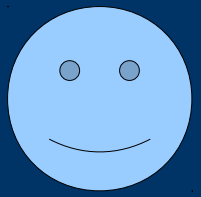
GV multi-documents : conflits (3/14)



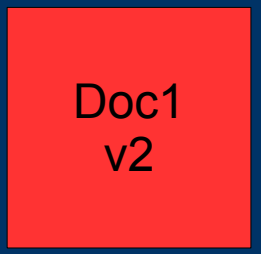
Dév A



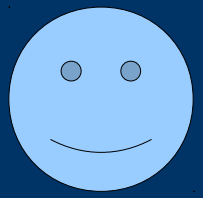




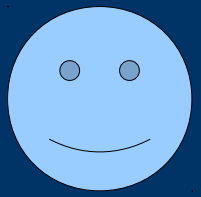
Dév B



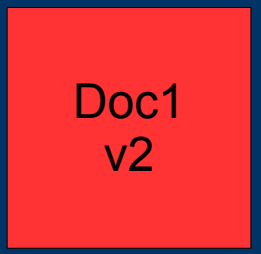
GV multi-documents : conflits (4/14)

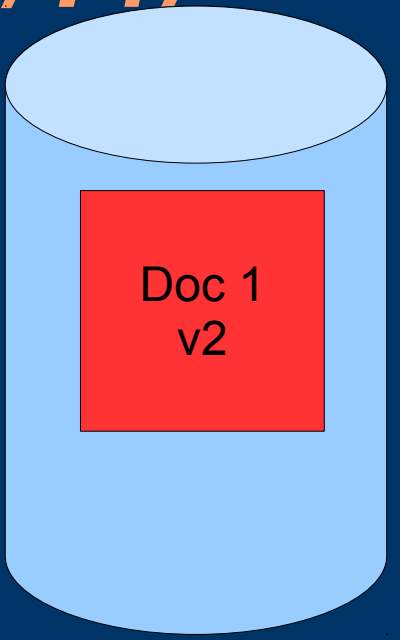


Dév A

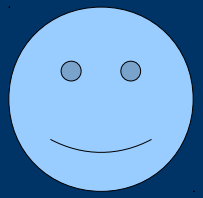


Dév B

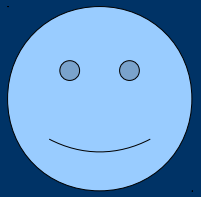




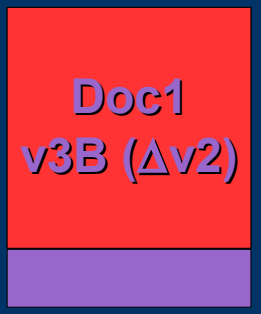
GV multi-documents : conflits (5/14)

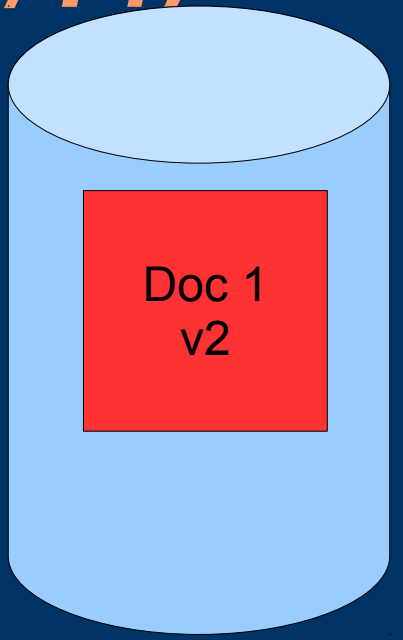


Dév A

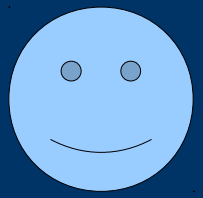


Dév B

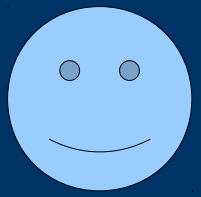




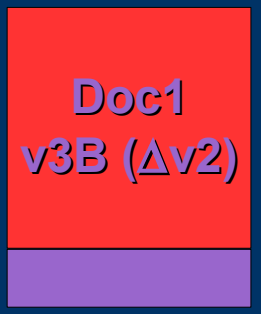
GV multi-documents : conflits (6/14)

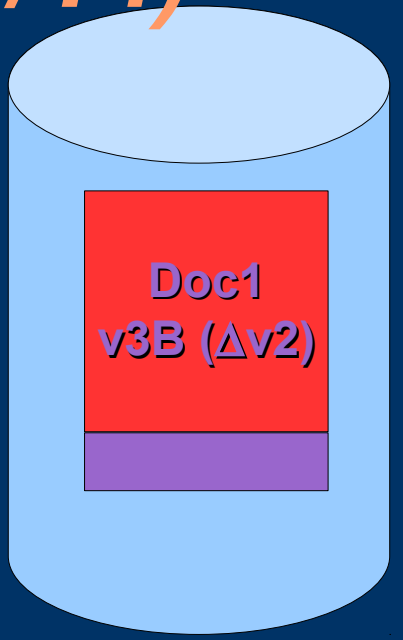


Dév A

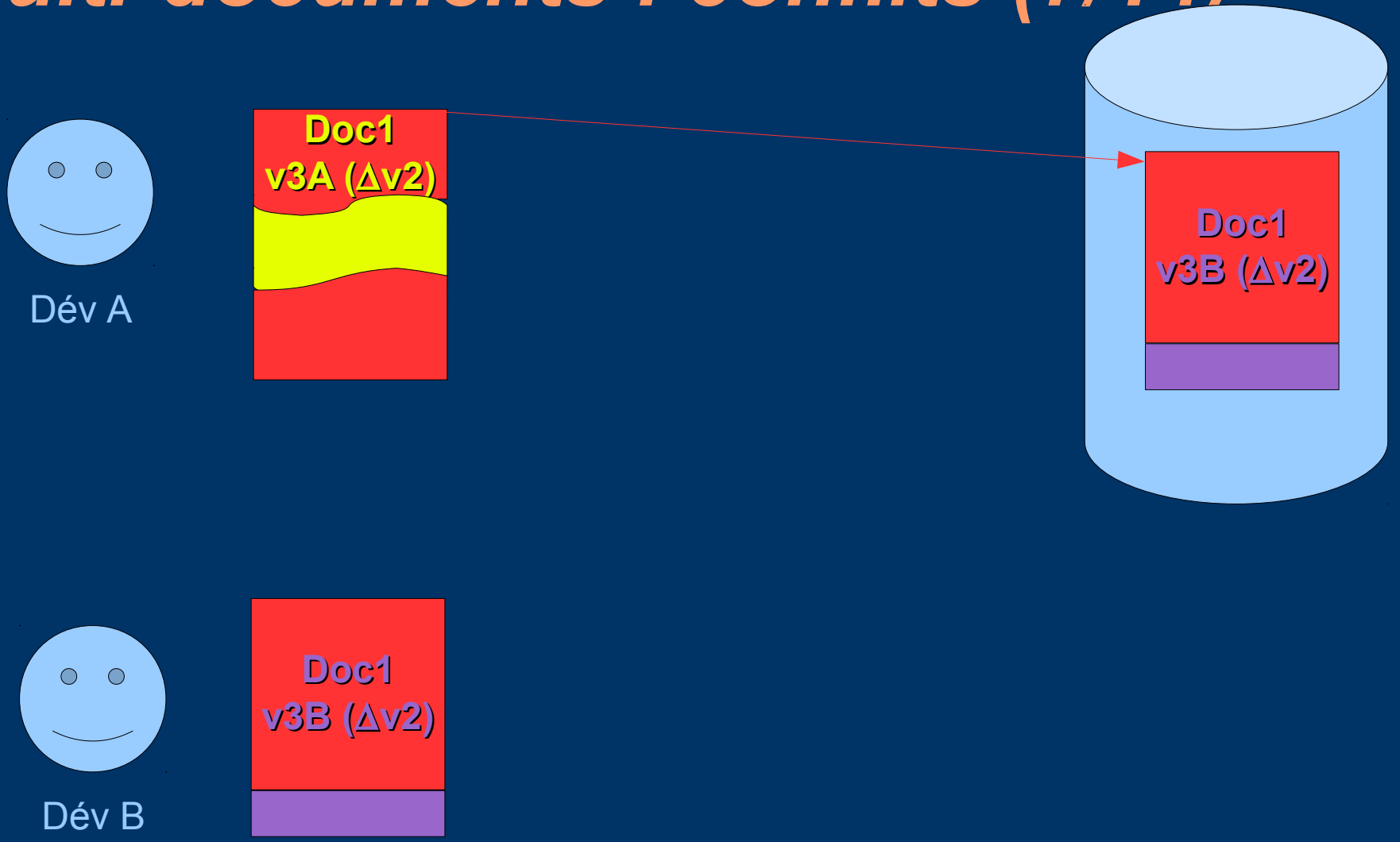


Dév B

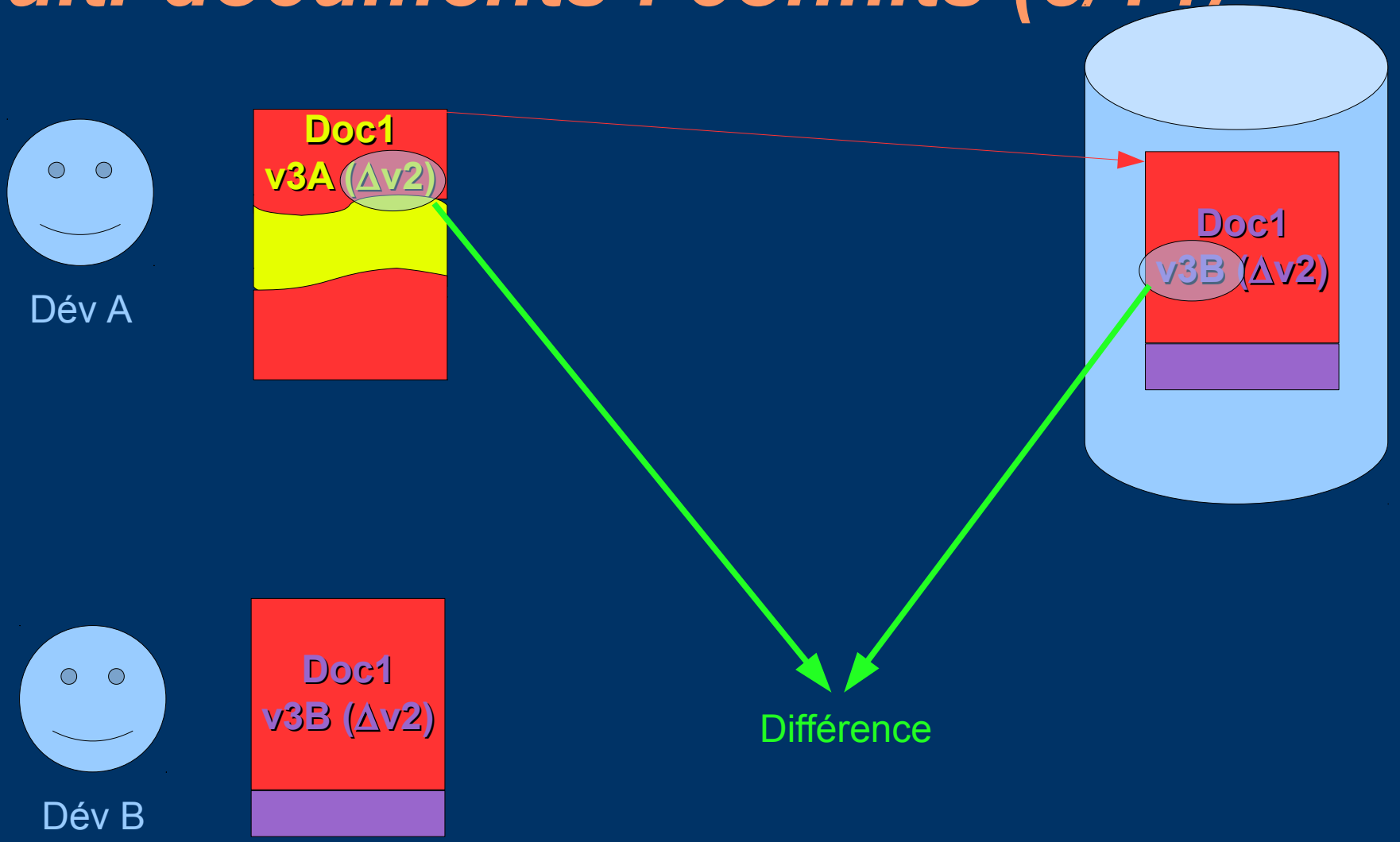




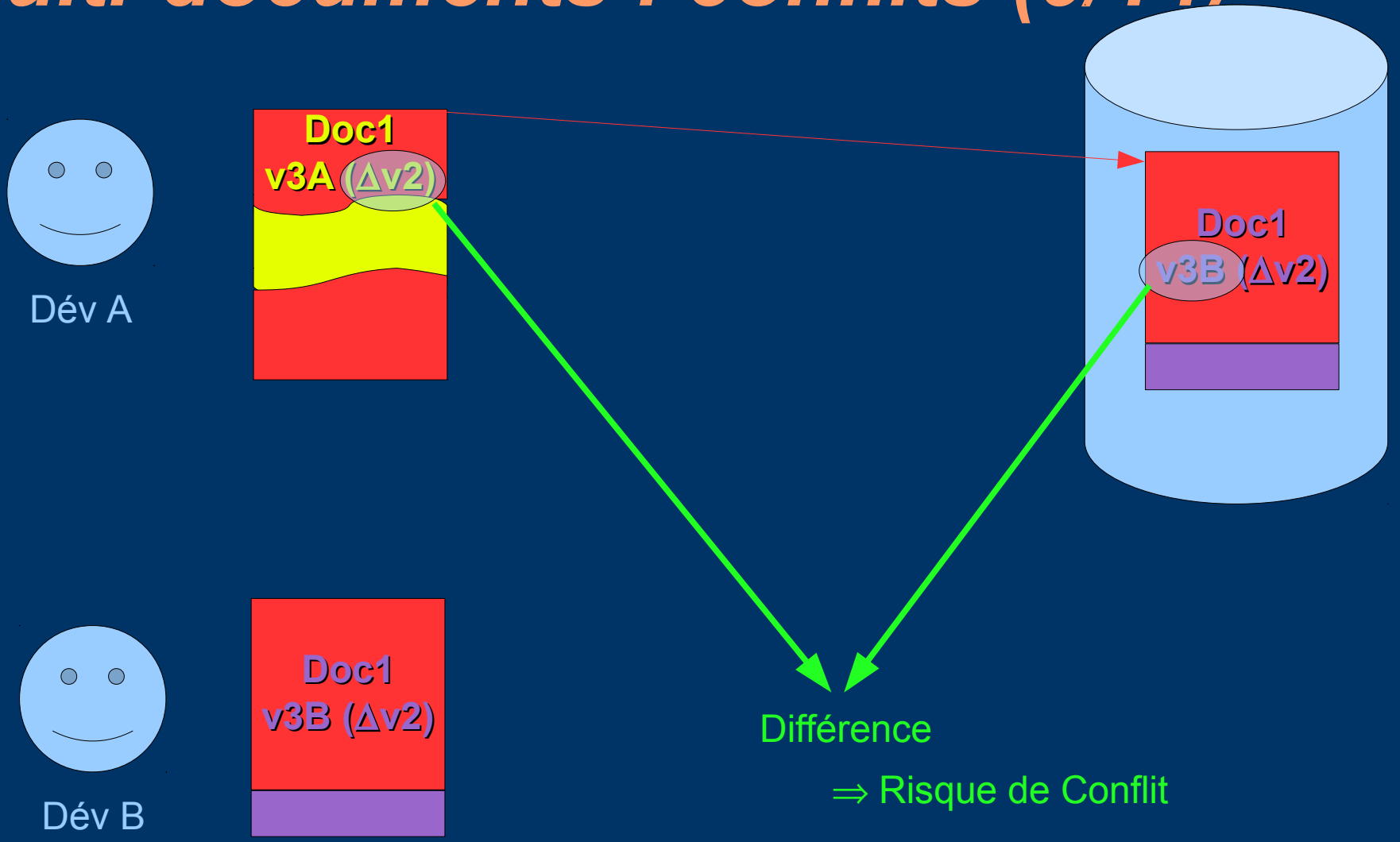
GV multi-documents : conflits (7/14)



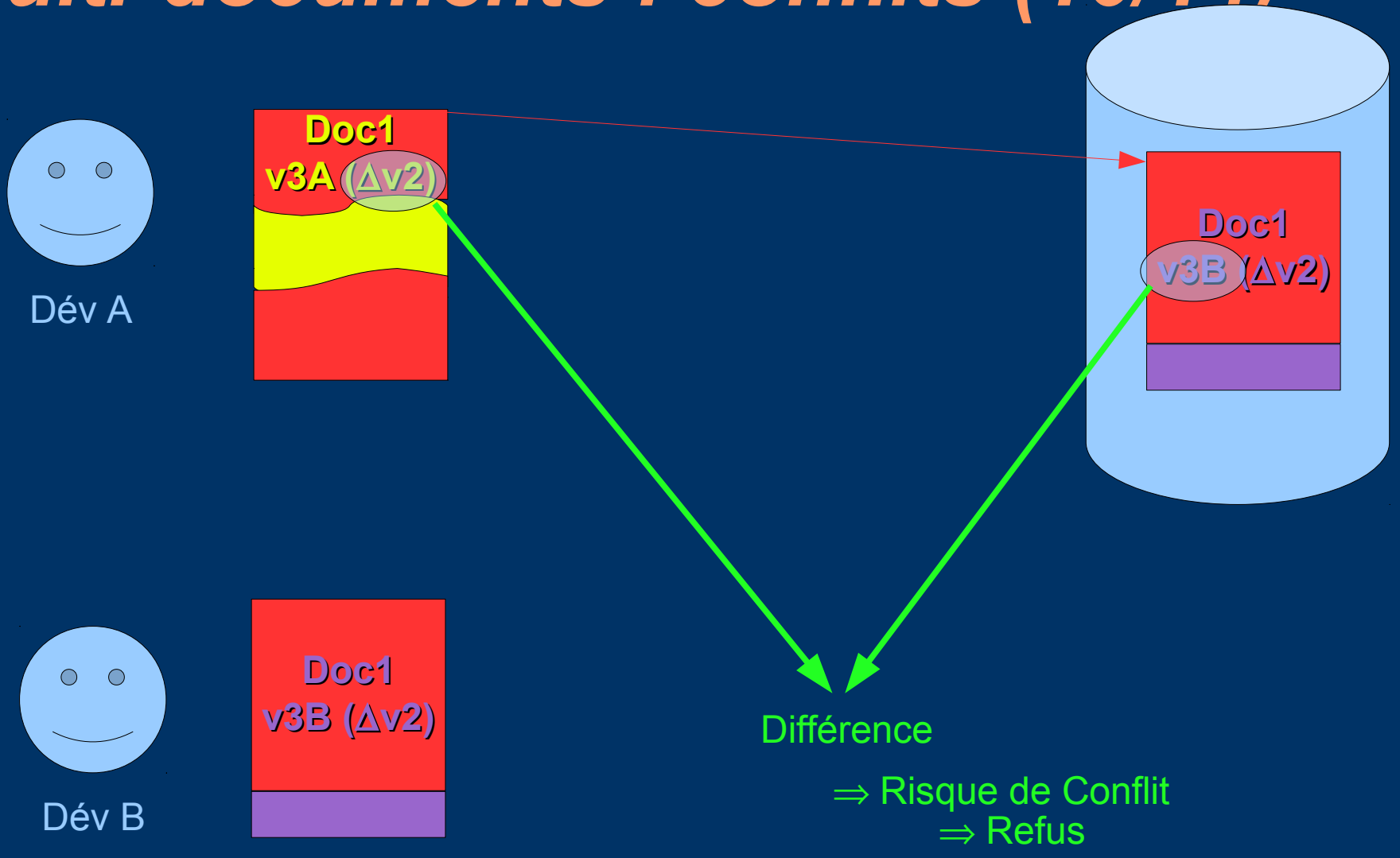
GV multi-documents : conflits (8/14)



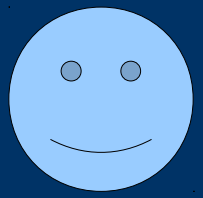
GV multi-documents : conflits (9/14)



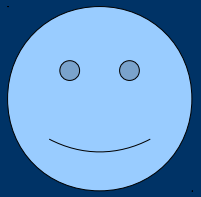
GV multi-documents : conflits (10/14)



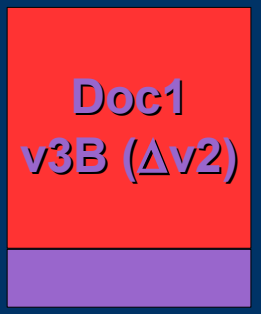
GV multi-documents : conflits (11/14)

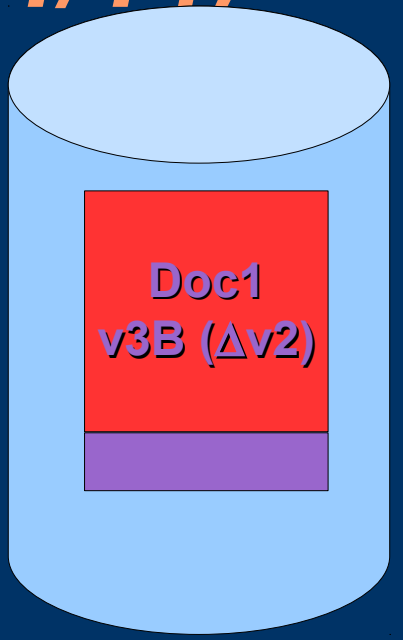


Dév A

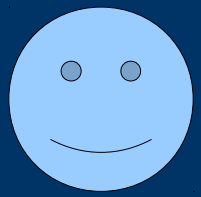


Dév B

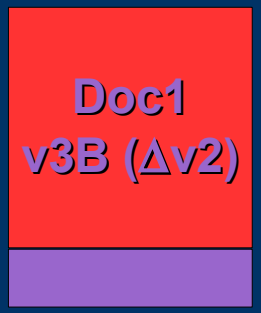


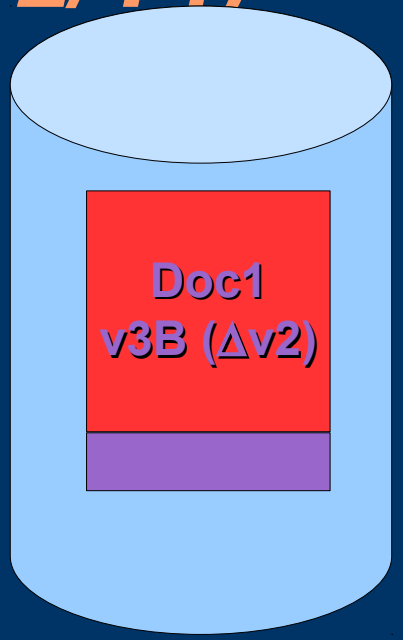


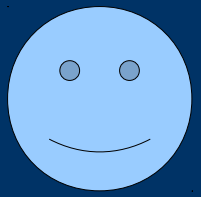
GV multi-documents : conflits (12/14)



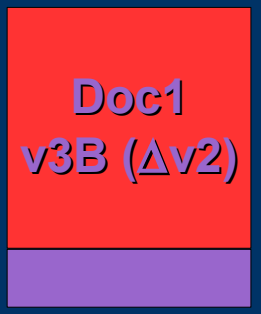
Dév A



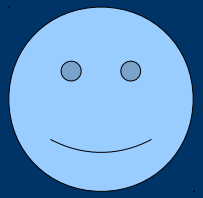




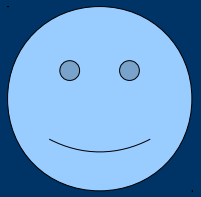
Dév B



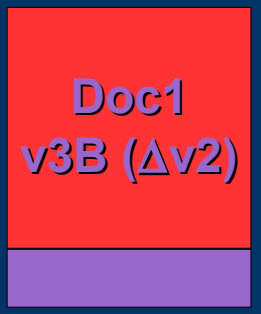
GV multi-documents : conflits (13/14)

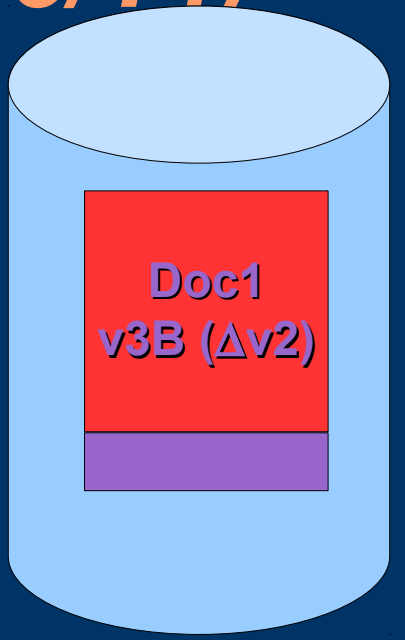


Dév A

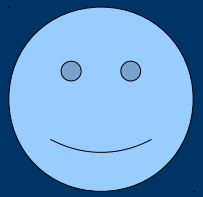


Dév B

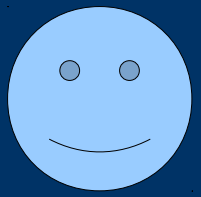




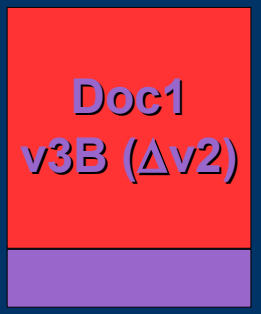
GV multi-documents : conflits (14/14)

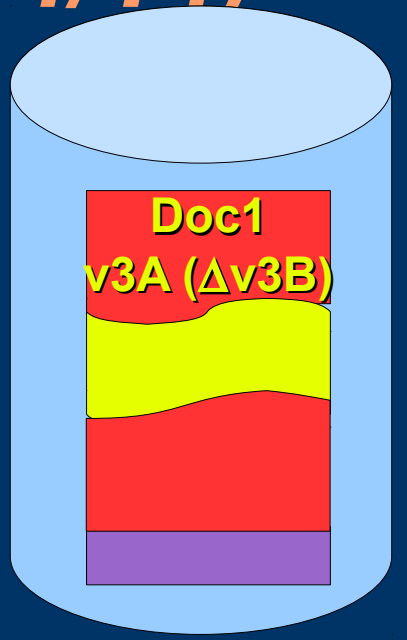


Dév A



Dév B





Gestion de version décentralisée
Principes généraux

- Chaque développeur possède
  - Une version courante
  - Sa propre version du dépôt
- Un (ou plusieurs) serveur(s) héberge(nt) une version du projet
- Le développeur utilise sont dépôt local pour gérer les versions
- Régulièrement, le développeur synchronise son dépôt local avec le(s) dépôt(s) distants

Gestion de version décentralisée
Opérations principales

- Récupérer un projet depuis un serveur
  - Clone
- Récupérer une version depuis son dépôt
  - Checkout
- Sauvegarder une nouvelle version dans son dépôt
  - Commit
- Transférer la version actuelle de son dépôt sur le serveur
  - Push
- Récupérer une nouvelle version depuis le dépôt
  - Fetch/Pull
- Fusionner de versions
  - Merge

Git : un système de gestion de versions décentralisé

- Créé en 2005 par Linus Torvalds pour le développement du noyau de Linux
- Utilisation

  `git commandeGit paramètres`
- Aide
  - `git help commandeGit`
  - Livre Pro Git : https://git-scm.com/book/fr/v2
- Utilisation
  - Auparavant surtout pour les logiciels libres
  - Depuis très récemment, adopté massivement par les entreprises (en lieu et place de subversion)

Git : Plan

- Principes généraux (T23) ⟹
- Fonctionnement de base
  - Travail avec un seul fichier (T27) ⟹
  - Travail avec plusieurs fichiers et fichier .gitignore (T37) ⟹
  - Étiqueter une version (T41) ⟹
- Travailler avec les branches (T43) ⟹
- Retour sur le fonctionnement de git en images (T64) ⟹

Travailler avec Git

- Travail en local
  - Les différents états d'un fichier
  - Créer son répertoire de travail
  - Configurer git
  - Exemple introductif avec un seul fichier
  - L'historique
  - Les étiquettes
  - La notion de branche
- Synchronisation avec un autre dépôt

Créer son répertoire de travail

- Créer le répertoire

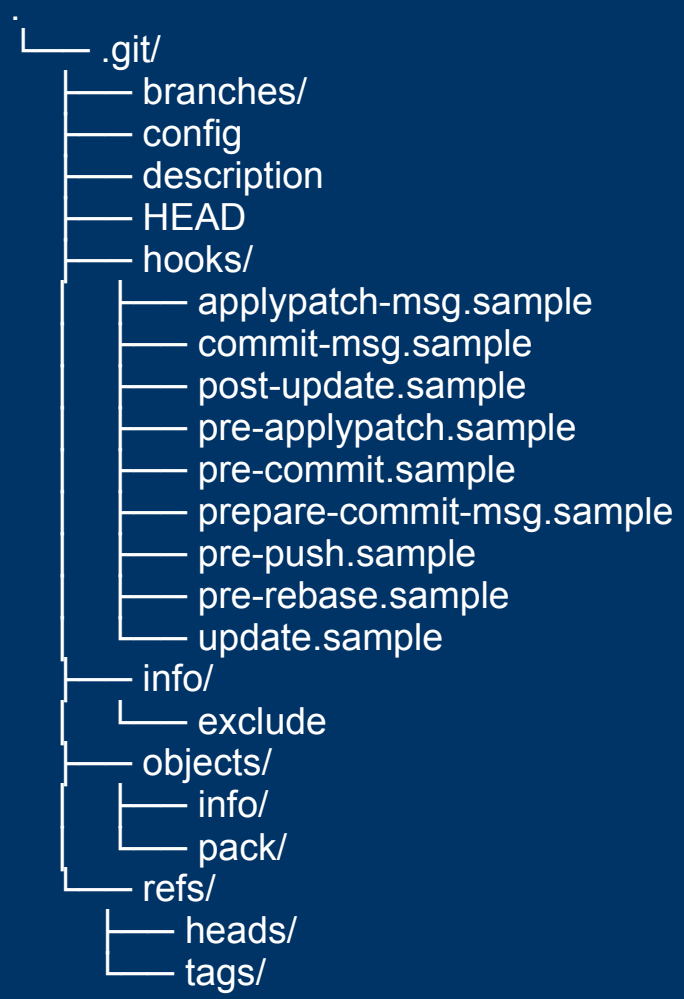
  mkdir Projet

- Aller dans le répertoire

  cd Projet

- Initialiser le dépôt git

  git init



Configurer git

- Configuration : les bases
  - Configuration globale : git config --global
  - Configuration pour le projet courant : git config
- Configuration : le minimum
  - git config --global user.name "Prénom Nom"
  - git config --global user.email "adresseElectronique"
- Configuration : le petite plus
  - git config --global core.editor *emacs*
- Affichage
  - git config -l

Statut d'un fichier

- Statuts possibles :
  - Modifié (ou créé) localement
  - Prêt à être transféré dans le dépôt
  - Tel que dans le dépôt
- Connaître le statut
  - git status

```
Sur la branche master

Validation initiale

rien à valider (créez/copiez des fichiers et utilisez "git add" pour les suivre)
```

Statut d'un fichier

- Statuts possibles :
  - Modifié (ou créé) localement
  - Prêt à être transféré dans le dépôt
  - Tel que dans le dépôt
- Connaître le statut
  - git status

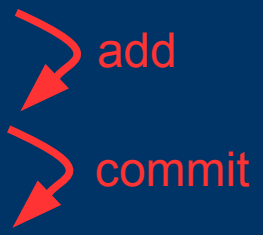

```
Sur la branche master

Validation initiale

rien à valider (créez/copiez des fichiers et utilisez "git add" pour les suivre)
```

Statut d'un fichier

- Statuts possibles :
  - Modifié (ou créé) localement
  - Prêt à être transféré dans le dépôt
  - Tel que dans le dépôt
- Connaître le statut
  - git status




```
Sur la branche master

Validation initiale

rien à valider (créez/copiez des fichiers et utilisez "git add" pour les suivre)
```

Travail avec un fichier Exemple (1)

- emacs Hello.java
- ls

```
Hello.java
```

- git status

```
Sur la branche master

Validation initiale

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

        Hello.java

aucune modification ajoutée à la validation mais des fichiers non suivis sont présents (utilisez
"git add" pour les suivre)
```

Travail avec un fichier Exemple (2)

- javac Hello.java
- ls

```
Hello.class    Hello.java
```

- git status

```
Sur la branche master

Validation initiale

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

        Hello.class
        Hello.java

aucune modification ajoutée à la validation mais des fichiers non suivis sont présents (utilisez
"git add" pour les suivre)
```

Travail avec un fichier Exemple (3)

- git add Hello.java
- git status

```
Sur la branche master

Validation initiale

Modifications qui seront validées :
  (utilisez "git rm --cached <fichier>..." pour désindexer)

        nouveau fichier: Hello.java

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

        Hello.class
```

Travail avec un fichier Exemple (4)

- git commit -m "Création du Hello standard"

```
[master (commit racine) fd577c3] Création du Hello standard
 1 file changed, 5 insertions(+)
 create mode 100644 Hello.java
```

- git status

```
Sur la branche master
Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

        Hello.class

aucune modification ajoutée à la validation mais des fichiers non suivis sont présents (utilisez
"git add" pour les suivre)
```

Travail avec un fichier Exemple (5)

- git log

```
commit fd577c31871eb2c938a5d772dd5a13cd7734fc83
Author: Bruno Mermet <Bruno.Mermet@univ-lehavre.fr>
Date:   Tue Jun 6 12:50:26 2017 +0200

    Création du Hello standard
```

Travail avec un fichier Exemple (6)

- emacs Hello.java
- javac Hello.java
- git status

```
Sur la branche master
Modifications qui ne seront pas validées :
  (utilisez "git add <fichier>..." pour mettre à jour ce qui sera validé)
  (utilisez "git checkout -- <fichier>..." pour annuler les modifications dans la copie de travail)

        modifié:        Hello.java

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

        Hello.class

aucune modification n'a été ajoutée à la validation (utilisez "git add" ou "git commit -a")
```

Travail avec un fichier Exemple (7)

- git add Hello.java
- git status

```
Sur la branche master
Modifications qui seront validées :
  (utilisez "git reset HEAD <fichier>..." pour désindexer)

        modifié:        Hello.java

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

        Hello.class
```

Travail avec un fichier Exemple (8)

- git commit -m "message plus long" ; git status

```
Sur la branche master
Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)
        Hello.class

aucune modification ajoutée à la validation mais des fichiers non suivis sont présents (utilisez
"git add" pour les suivre)
```

- git log

```
commit 18ef331bbe4ef1b36cf11e2a712da8945f8c7f00
Author: Bruno Mermet <Bruno.Mermet@univ-lehavre.fr>
Date:   Tue Jun 6 18:23:11 2017 +0200
    message plus long

commit fd577c31871eb2c938a5d772dd5a13cd7734fc83
Author: Bruno Mermet <Bruno.Mermet@univ-lehavre.fr>
Date:   Tue Jun 6 12:50:26 2017 +0200

    Création du Hello standard
```

Simplifier la prise en comptes des fichiers modifiés

- Pour les fichiers modifiés (pas pour les fichiers créés), on peut remplacer :
  - git add fic1 ; git add fic2
  - git commit
- Par
  - git commit -a

Travail avec un fichier Exemple (9)

- git diff fd577c

```
diff --git a/Hello.java b/Hello.java

index 4593c25..5f3508d 100644
--- a/Hello.java
+++ b/Hello.java
@@ -1,5 +1,5 @@
 public class Hello {
     public static void main(String... args) {
-        System.out.println("Hello");
+        System.out.print("Hello World");
     }
 }
```

Travailler avec plusieurs fichiers

- Ajouter plusieurs fichiers modifiés
  - git add *repertoire* (et notamment git add .)
- Ne pas faire gérer certains fichiers
  - Utilisation du fichier .gitignore
- Déplacer un fichier en gardant l'historique
  - git mv
- Faire ignorer un fichier jusqu'à présent pris en compte
  - git rm

Format du fichier .gitignore (à faire gérer par git)

- Ref : https://git-scm.com/docs/gitignore
- 1 motif par ligne
- Les lignes vides ne comptent pas
- Les lignes commençant par un # sont des commentaires
- Principales formes de motif (* = jocker sauf '/')
  - chemin
  - chemin/
  - !motif
  - chemin/**/chemin
  - /chemin

Format du fichier .gitignore (à faire gérer par git)

- Ref : https://git-scm.com/docs/gitignore
- 1 motif par ligne
- Les lignes vides ne comptent pas
- Les lignes commençant par un # sont des commentaires
- Principales formes de motif (* = jocker sauf '/')
  - chemin
  - chemin/
  - !motif
  - chemin/**/chemin
  - /chemin

*.class

bin/

bin/**/*.class

Travail avec un fichier
Exemple (10) : fichier .gitignore

- Avant création du .gitignore : git status

```
Sur la branche master
Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)
        Hello.class

aucune modification ajoutée à la validation mais des fichiers non suivis sont présents (utilisez
"git add" pour les suivre)
```

- Après création du .gitignore

```
Sur la branche master
Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)
        .gitignore

aucune modification ajoutée à la validation mais des fichiers non suivis sont présents (utilisez
"git add" pour les suivre)
```

Travail avec un fichier
Exemple (11) : fichier .gitignore

- Faire gérer le .gitignore à git

  git commit -a -m "ajout du fichier .gitignore pour ignorer les fichiers .class"

  git status

```
Sur la branche master
rien à valider, la copie de travail est propre
```

- git log --oneline

```
9da7eb1 ajout du fichier .gitignore pour ignorer les fichiers .class
18ef331 message plus long
fd577c3 Création du Hello standard
```

Étiquetage d'une version

- Étiqueter une version permet de donner un nom à une version particulière pour s'y retrouver.
- Utilisation
  - git tag -a "nomVersion"
  - git tag -a "nomVersion" numVersion

Travail avec un fichier
Exemple (12) : étiquetage

- git tag -a "v1.0" ; git log --decorate --oneline

```
9da7eb1 (HEAD, tag: v1.0, master) ajout du fichier .gitignore pour ignorer les fichiers .class
18ef331 message plus long
fd577c3 Création du Hello standard
```

- emacs Hello.java ; git commit -a -m "au revoir"
- git log --decorate --oneline

```
4712b0f (HEAD, master) au revoir
9da7eb1 (tag: v1.0) ajout du fichier .gitignore pour ignorer les fichiers .class
18ef331 message plus long
fd577c3 Création du Hello standard
```

- git tag -a "debut" 9da7eb1 ; git log --decorate --oneline

```
4712b0f (HEAD, master) au revoir
9da7eb1 (tag: v1.0, tag: debut) ajout du fichier .gitignore pour ignorer les fichiers .class
18ef331 message plus long
fd577c3 Création du Hello standard
```

Notion de branche

- Les Systèmes de Gestion de version permettent de créer des branches
- La création de branches permet de mener des développements différents à partir d'un point donné, sans qu'il y ait interaction entre les 2 développements
- La fusion de branches est une fonction fondamentale, permettant de ré-intégrer dans une branche le travail fait dans une autre
- Principales utilisations des branches
  - Développement d'une nouvelle fonctionnalité
  - Correctif à une version antérieure

Développer une fonctionnalité avec les branches : démarche générale

- Créer une nouvelle branche
- Basculer sur la branche créée
- Développer sa fonctionnalité

  (Régulièrement, intégrer les modifications éventuellement apportées dans la branche de départ)
- Lorsque la fonctionnalité est terminée (développée, documentée, testée)
  - Réintégrer la fonctionnalité dans la branche de départ
  - Supprimer la branche utilisée

Git et les branches

- Nom de la branche principale : master
- Connaître la liste des branches
  - git branch
- Créer une branche (sans aller dedans)
  - git branch nomBranche
- Aller sur une branche donnée
  - git checkout nomBranche
- Créer une branche et aller dedans
  - git checkout -b nomBranche
- Supprimer une branche
  - git branch -d nomBranche
- Intégrer dans la branche courante le contenu d'une autre branche
  - git merge nomBranche

Travail avec les branches
Exemple (1) : situation de départ

- git branch

```
* master
```

- git log --decorate --oneline

```
4712b0f (HEAD, master) au revoir
9da7eb1 (tag: v1.0, tag: debut) ajout du fichier .gitignore pour ignorer les fichiers .class
18ef331 message plus long
fd577c3 Création du Hello standard
```

Travail avec les branches
Exemple (2) : créer une fonctionnalité

- git branch "addition" ; git branch

```
addition
* master
```

- git log --decorate --oneline

```
4712b0f (HEAD, master, addition) au revoir
9da7eb1 (tag: v1.0, tag: debut) ajout du fichier .gitignore pour ignorer les fichiers .class
18ef331 message plus long
fd577c3 Création du Hello standard
```

- git checkout addition ; git branch

```
* addition
master
```

Travail avec les branches
Exemple (3) : créer une fonctionnalité

- emacs Operation.java ; emacs Hello.java
- javac Hello.java ; git status

```
Sur la branche addition
Modifications qui ne seront pas validées :
  (utilisez "git add <fichier>..." pour mettre à jour ce qui sera validé)
  (utilisez "git checkout -- <fichier>..." pour annuler les modifications dans la copie de travail)

        modifié:        Hello.java

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

        Operation.java

aucune modification n'a été ajoutée à la validation (utilisez "git add" ou "git commit -a")
```

- git add . ; git commit -m "addition, premier essai"

Travail avec les branches
Exemple (4) : état de l'historique

- emacs Operation.java ; emacs Hello.java
- git add . ; git commit
- javac Hello.java
- git log --decorate --oneline

```
45eaaa2 (HEAD, addition) addition, premier essai
4712b0f (master) au revoir
9da7eb1 (tag: v1.0, tag: debut) ajout du fichier .gitignore pour ignorer les fichiers .class
18ef331 message plus long
fd577c3 Création du Hello standard
```

Travail avec les branches
Exemple (5) : évolution du tronc

- git checkout master
- git log --decorate --oneline --all

```
45eaaa2 (HEAD, addition) addition, premier essai
4712b0f (master) au revoir
9da7eb1 (tag: v1.0, tag: debut) ajout du fichier .gitignore pour ignorer les fichiers .class
18ef331 message plus long
fd577c3 Création du Hello standard
```

- emacs Hello.java ; git commit -a -m "francisation"
- git log --decorate --oneline --all --graph

```
* 0e08bc5 (HEAD, master) francisation
| * 45eaaa2 (addition) addition, premier essai
|/
* 4712b0f au revoir
* 9da7eb1 (tag: v1.0, tag: debut) ajout du fichier .gitignore pour ignorer les fichiers .class
* 18ef331 message plus long
* fd577c3 Création du Hello standard
```

Travail avec les branches
Exemple (6) : reprise du travail dans la branche

- git checkout addition
- git log --decorate --oneline --all –graph

```
* 0e08bc5 (master) francisation
| * 45eaaa2 (HEAD, addition) addition, premier essai
|/
* 4712b0f au revoir
* 9da7eb1 (tag: v1.0, tag: debut) ajout du fichier .gitignore pour ignorer les fichiers .class
* 18ef331 message plus long
* fd577c3 Création du Hello standard
```

- emacs Operation.java ; emacs Hello.java
- javac Hello.java
- git commit -a -m "passage par une méthode d'instance"

Travail avec les branches
Exemple (7) : état des lieux

- git log --decorate --oneline --all --graph

```
* ca919a2 (HEAD, addition) passage par une méthode d'instance
* 45eaaa2 addition, premier essai
| * 0e08bc5 (master) francisation
|/
* 4712b0f au revoir
* 9da7eb1 (tag: v1.0, tag: debut) ajout du fichier .gitignore pour ignorer les fichiers .class
* 18ef331 message plus long
* fd577c3 Création du Hello standard
```

Travail avec les branches
Exemple (8) : synchro avec le tronc

- git merge master

```
Fusion automatique de Hello.java
CONFLIT (contenu) : Conflit de fusion dans Hello.java
La fusion automatique a échoué ; réglez les conflits et validez le résultat.
```

- cat Hello.java

```
public class Hello {
    public static void main(String... args) {
<<<<<<< HEAD
        System.out.print("Hello World");
        Operation op = new Operation(2,3);
        System.out.println("2+3="+op.addition());
        System.out.print("Au revoir");
=======
        System.out.println("Bonjour");
        System.out.println("Au revoir");
>>>>>>> master
    }
}
```

Travail avec les branches
Exemple (8) : synchro avec le tronc

- git merge master

```
Fusion automatique de Hello.java
CONFLIT (contenu) : Conflit de fusion dans Hello.java
La fusion automatique a échoué ; réglez les conflits et validez le résultat.
```

- cat Hello.java

```
public class Hello {
   public static void main(String... args) {
<<<<<<< HEAD
      System.out.print("Hello World");
      Operation op = new Operation(2,3);
      System.out.println("2+3="+op.addition());
      System.out.print("Au revoir");
=======
      System.out.println("Bonjour");
      System.out.println("Au revoir");
>>>>>>> master
   }
}
```

Travail avec les branches
Exemple (8) : synchro avec le tronc

- git merge master

```
Fusion automatique de Hello.java
CONFLIT (contenu) : Conflit de fusion dans Hello.java
La fusion automatique a échoué ; réglez les conflits et validez le résultat.
```

- cat Hello.java

```
public class Hello {
   public static void main(String... args) {
<<<<<<< HEAD
      System.out.print("Hello World");
      Operation op = new Operation(2,3);
      System.out.println("2+3="+op.addition());
      System.out.print("Au revoir");
=======
      System.out.println("Bonjour");
      System.out.println("Au revoir");
>>>>>>> master
   }
}
```

Travail avec les branches
Exemple (8) : synchro avec le tronc

- git merge master

```
Fusion automatique de Hello.java
CONFLIT (contenu) : Conflit de fusion dans Hello.java
La fusion automatique a échoué ; réglez les conflits et validez le résultat.
```

- cat Hello.java

```
public class Hello {
   public static void main(String... args) {
<<<<<<< HEAD
      System.out.print("Hello World");
      Operation op = new Operation(2,3);
      System.out.println("2+3="+op.addition());
      System.out.print("Au revoir");
=======
      System.out.println("Bonjour");
      System.out.println("Au revoir");
>>>>>>> master
   }
}
```

Travail avec les branches
Exemple (8) : résolution du conflit

- emacs Hello.java ; cat Hello.java

```
public class Hello {
    public static void main(String... args) {
       System.out.println("Bonjour");
       Operation op = new Operation(2,3);
       System.out.println("2+3="+op.addition());
       System.out.println("Au revoir");
    }
}
```

- git commit -a -m "intégration des modifs du tronc"

```
*   ddb2758 (HEAD, addition) intégration des modifs du tronc
|\
| * 0e08bc5 (master) francisation
* | ca919a2 passage par une méthode d'instance
* | 45eaaa2 addition, premier essai
|/
* 4712b0f au revoir
* 9da7eb1 (tag: v1.0, tag: debut) ajout du fichier .gitignore pour ignorer les fichiers .class
* 18ef331 message plus long
* fd577c3 Création du Hello standard
```

Résolution de conflit
Utilisation d'outils externes (1)

- Principe

  Lorsqu'un conflit est détecté, git mergetool permet d'utiliser un outil externe pour aider à la résolution du conflit

- Mise en œuvre avec Meld
  - sudo apt-get install meld
  - git config --global merge.tool meld
  - Après un « git merge » indiquant un conflit, lancer

    git mergetool

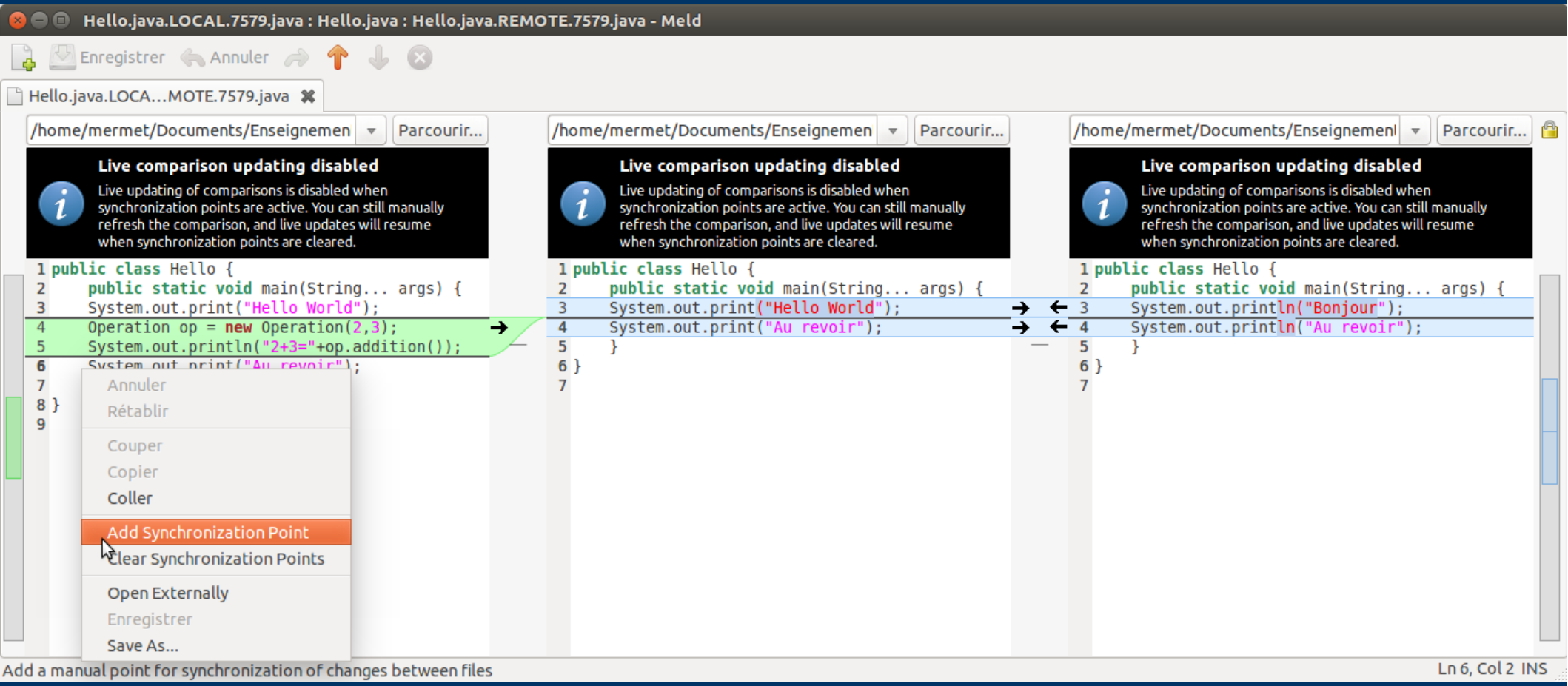
```
Merging:
Hello.java

Normal merge conflict for 'Hello.java':
  {local}: modified file
  {remote}: modified file
Hit return to start merge resolution tool (meld):
```

Résolution de conflit
Utilisation d'outils externes (2)

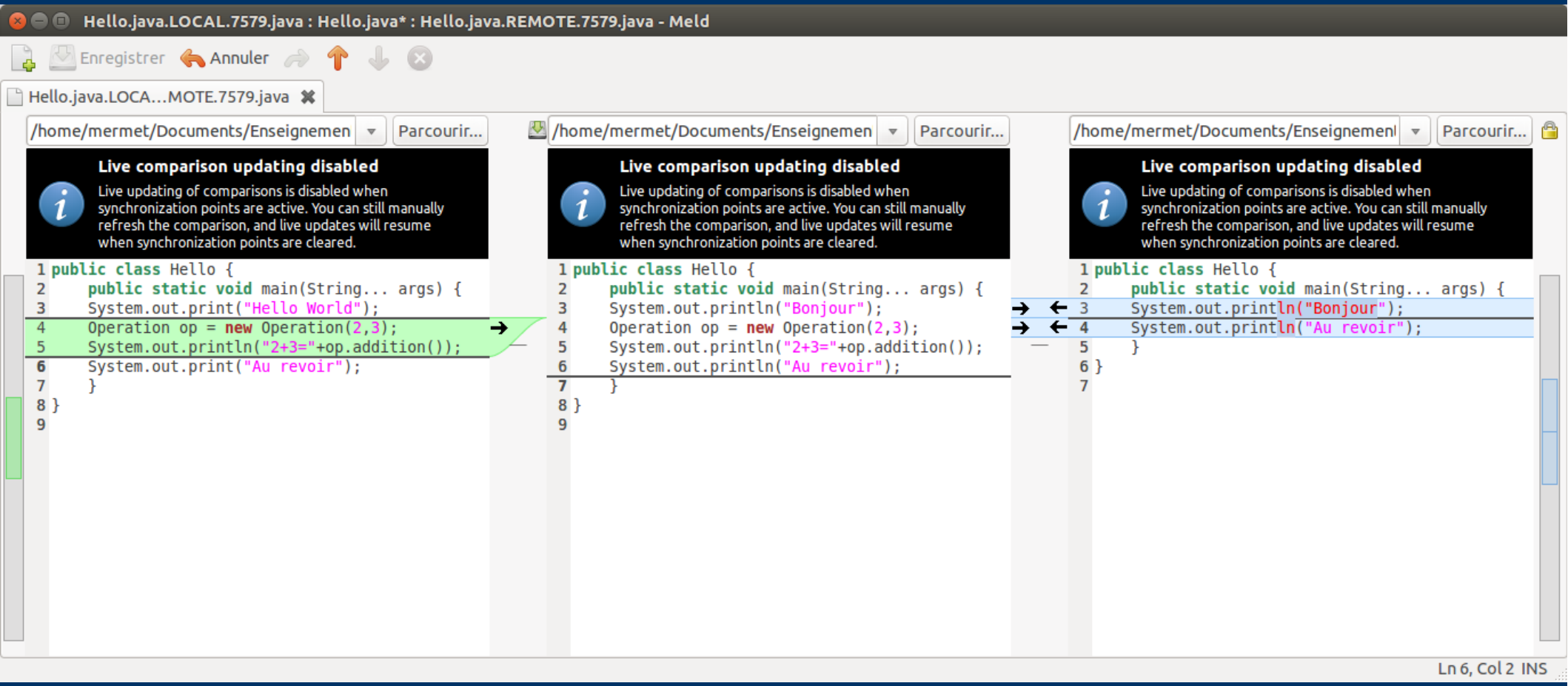
Résolution de conflit
Utilisation d'outils externes (3)

- Définition de points de synchronisation



Résolution de conflit
Utilisation d'outils externes (3)

- Après clic /Shift-clic/Ctrl-clic sur les bonnes flèches



Résolution de conflit
Utilisation d'outils externes (4)

- Après sauvegarde, git status

```
Tous les conflits sont réglés mais la fusion n'est pas terminée.
  (utilisez "git commit" pour terminer la fusion)

Modifications qui seront validées :

        modifié:        Hello.java

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

        Hello.java.orig
```

- rm Hello.java.orig
- git commit

Travail avec les branches
Exemple (9) : fin du dvlpt de la fonc.

- emacs Hello.java
- git commit -a -m "finalisation addition"
- git log --decorate --oneline --all --graph

```
* ed75c55 (HEAD, addition) finalisation addition
*   ddb2758 intégration des modifs du tronc
|\
| * 0e08bc5 (master) francisation
* | ca919a2 passage par une méthode d'instance
* | 45eaaa2 addition, premier essai
|/
* 4712b0f au revoir
* 9da7eb1 (tag: v1.0, tag: debut) ajout du fichier .gitignore pour ignorer les fichiers .class
* 18ef331 message plus long
* fd577c3 Création du Hello standard
```

Travail avec les branches
Exemple (10) : retour sur le tronc

- ls

```
Hello.class  Hello.java  Operation.class  Operation.java
```

- git checkout master
- ls

```
Hello.class  Hello.java  Operation.class
```

- git log --decorate --oneline --all --graph

```
* ed75c55 (addition) finalisation addition
*   ddb2758 intégration des modifs du tronc
|\
| * 0e08bc5 (HEAD, master) francisation
* | ca919a2 passage par une méthode d'instance
* | 45eaaa2 addition, premier essai
|/
* 4712b0f au revoir
* 9da7eb1 (tag: v1.0, tag: debut) ajout du fichier .gitignore pour ignorer les fichiers .class
* 18ef331 message plus long
* fd577c3 Création du Hello standard
```

Travail avec les branches
Exemple (11) : intégration de la fonc.

- Git merge addition

```
Mise à jour 0e08bc5..ed75c55
Fast-forward
 Hello.java     |  4 ++++
 Operation.java | 12 ++++++++++++
 2 files changed, 16 insertions(+)
 create mode 100644 Operation.java
```

- git log --decorate --oneline --all --graph

```
* ed75c55 (HEAD, master, addition) finalisation addition
*   ddb2758 intégration des modifs du tronc
|\
| * 0e08bc5 francisation
* | ca919a2 passage par une méthode d'instance
* | 45eaaa2 addition, premier essai
|/
* 4712b0f au revoir
* 9da7eb1 (tag: v1.0, tag: debut) ajout du fichier .gitignore pour ignorer les fichiers .class
* 18ef331 message plus long
* fd577c3 Création du Hello standard
```

Travail avec les branches
Exemple (12) : suppression branche

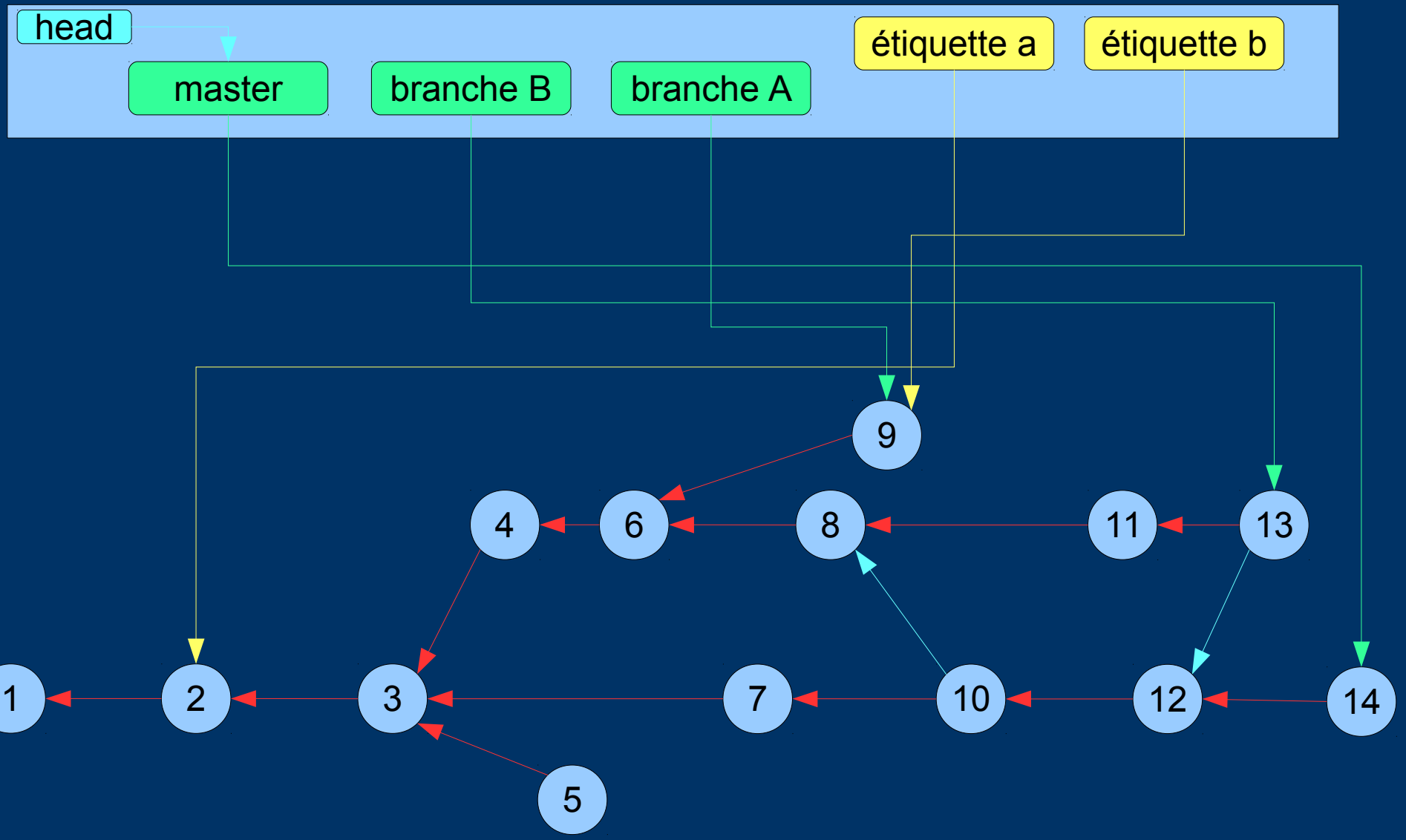
- git branch -d addition
- git branch -a

```
* master
```

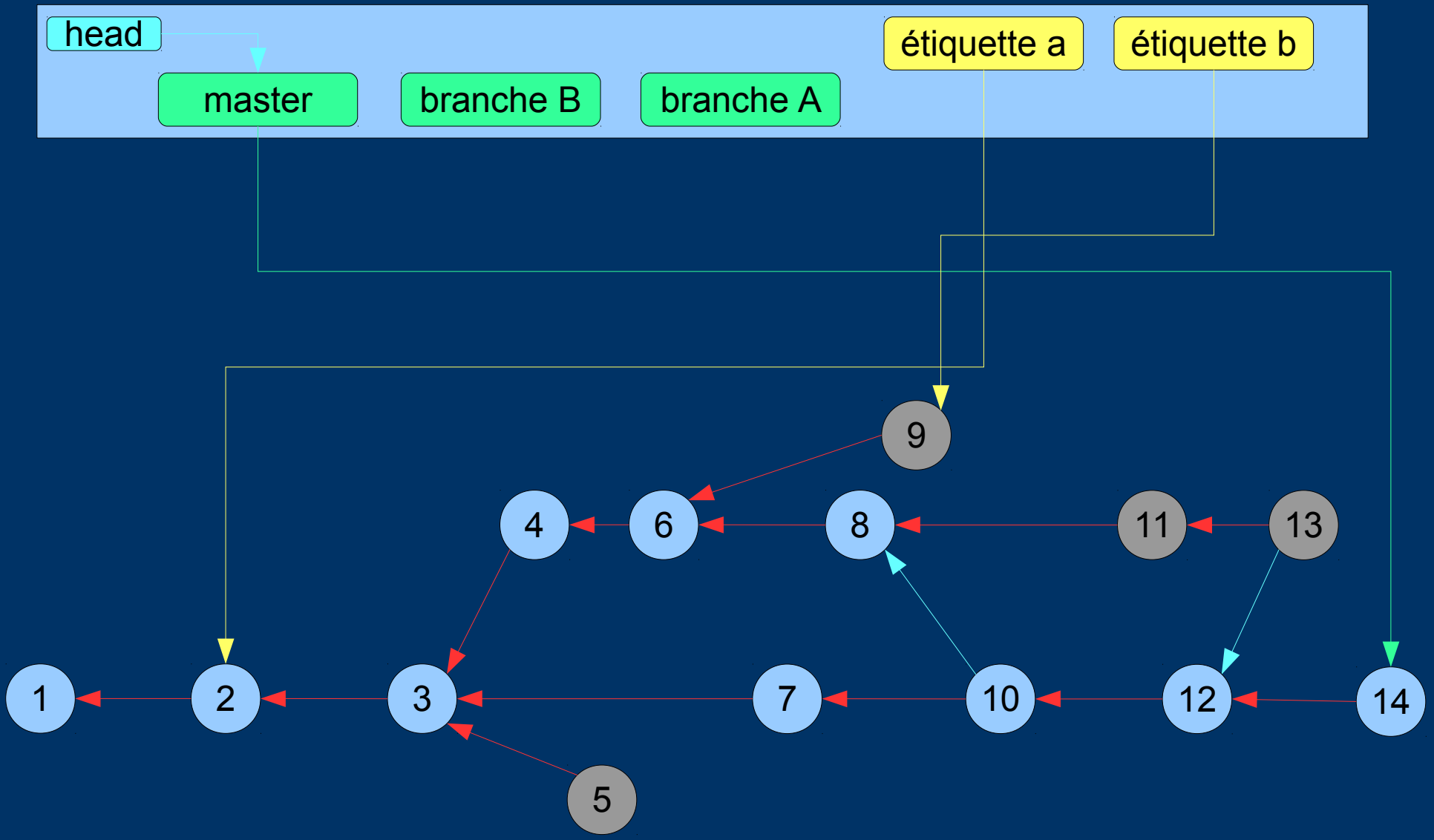
- git tag -a "v1.1"
- git log --decorate --oneline --all --graph

```
* ed75c55 (HEAD, tag: v1.1, master) finalisation addition
*   ddb2758 intégration des modifs du tronc
|\
| * 0e08bc5 francisation
* | ca919a2 passage par une méthode d'instance
* | 45eaaa2 addition, premier essai
|/
* 4712b0f au revoir
* 9da7eb1 (tag: v1.0, tag: debut) ajout du fichier .gitignore pour ignorer les fichiers .class
* 18ef331 message plus long
* fd577c3 Création du Hello standard
```

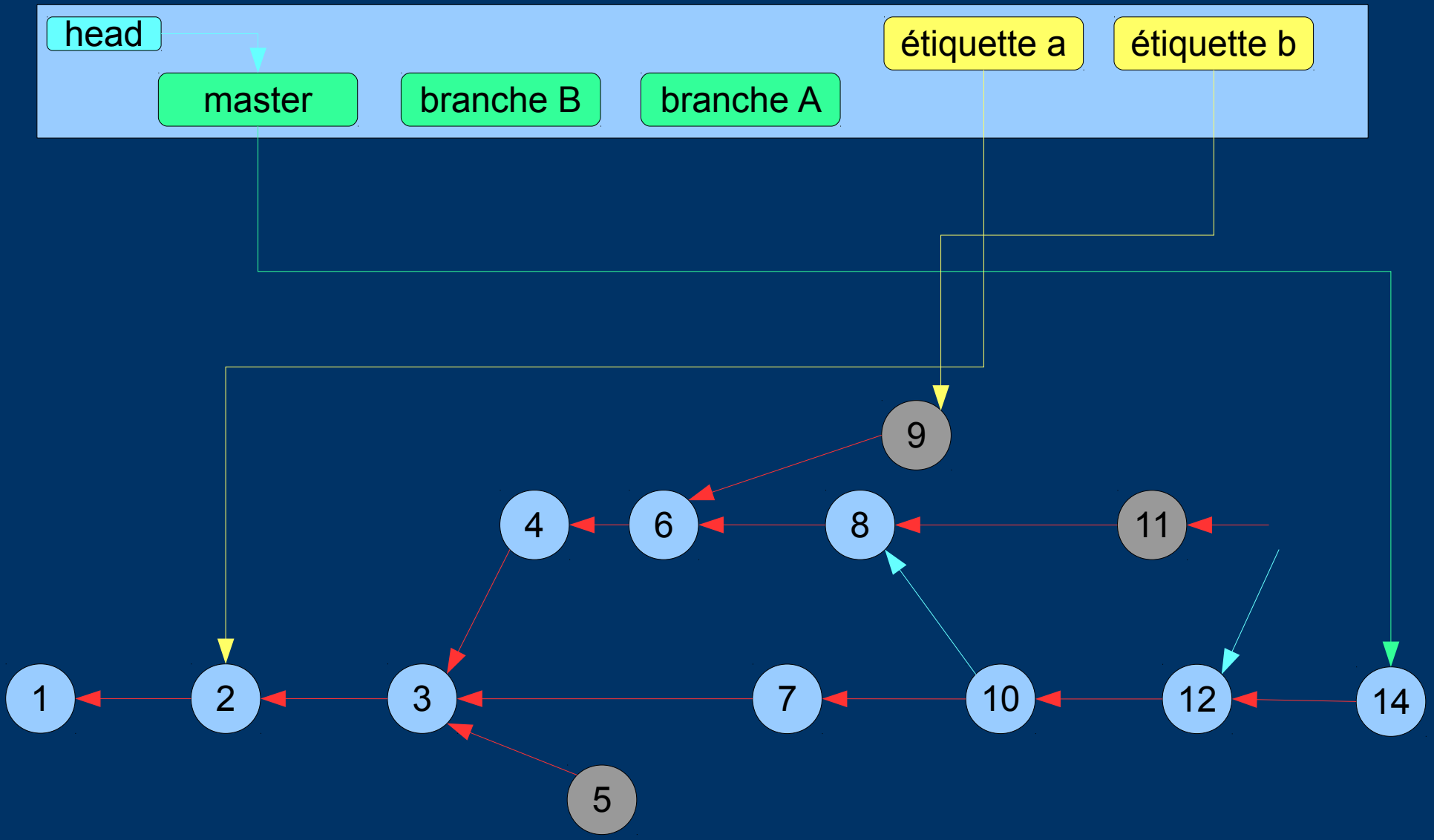
gestion des révisions par GIT
Une affaire de pointeurs



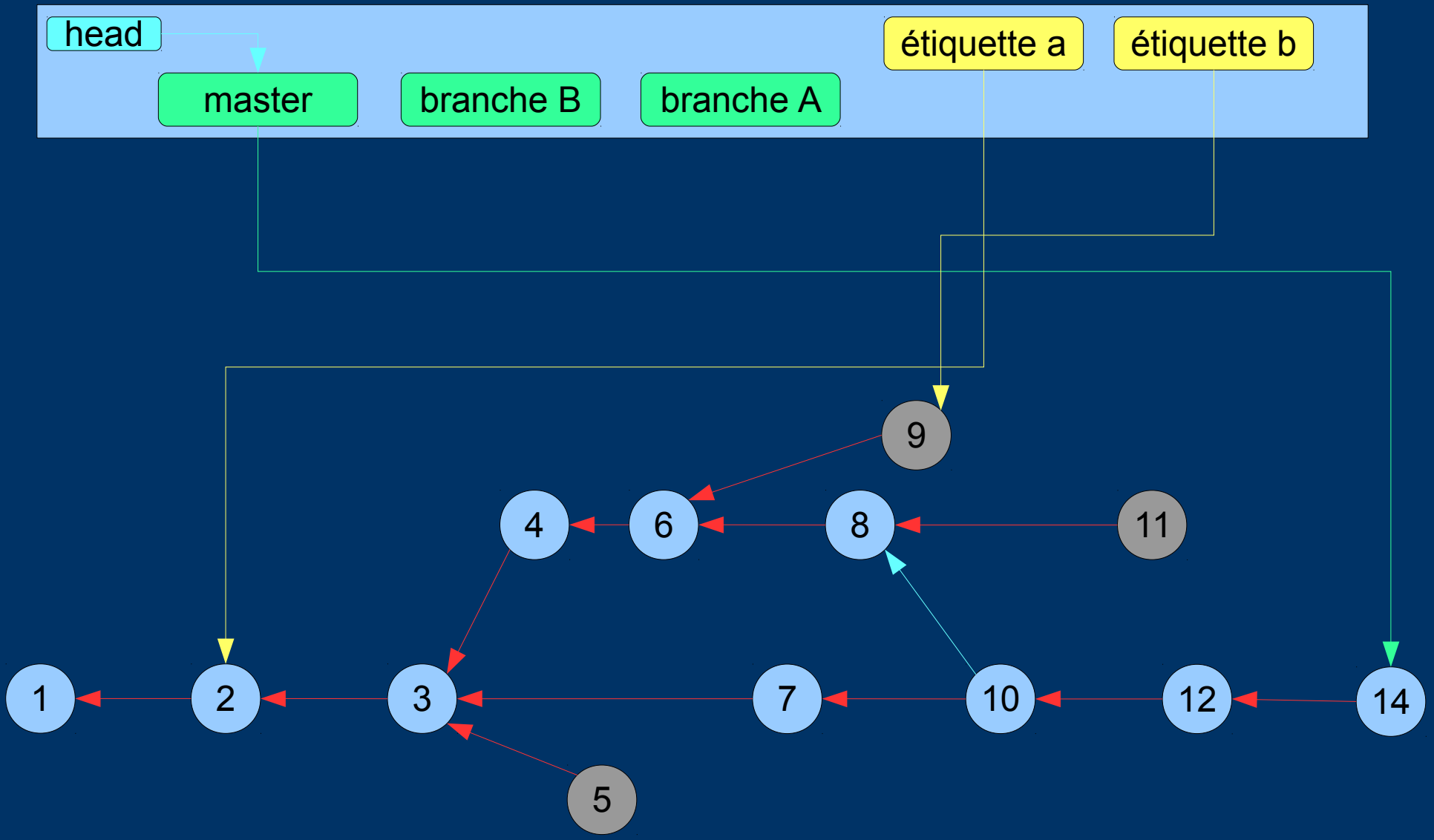
Gestion des révisions par GIT
Point de vue depuis « master » (1/6)



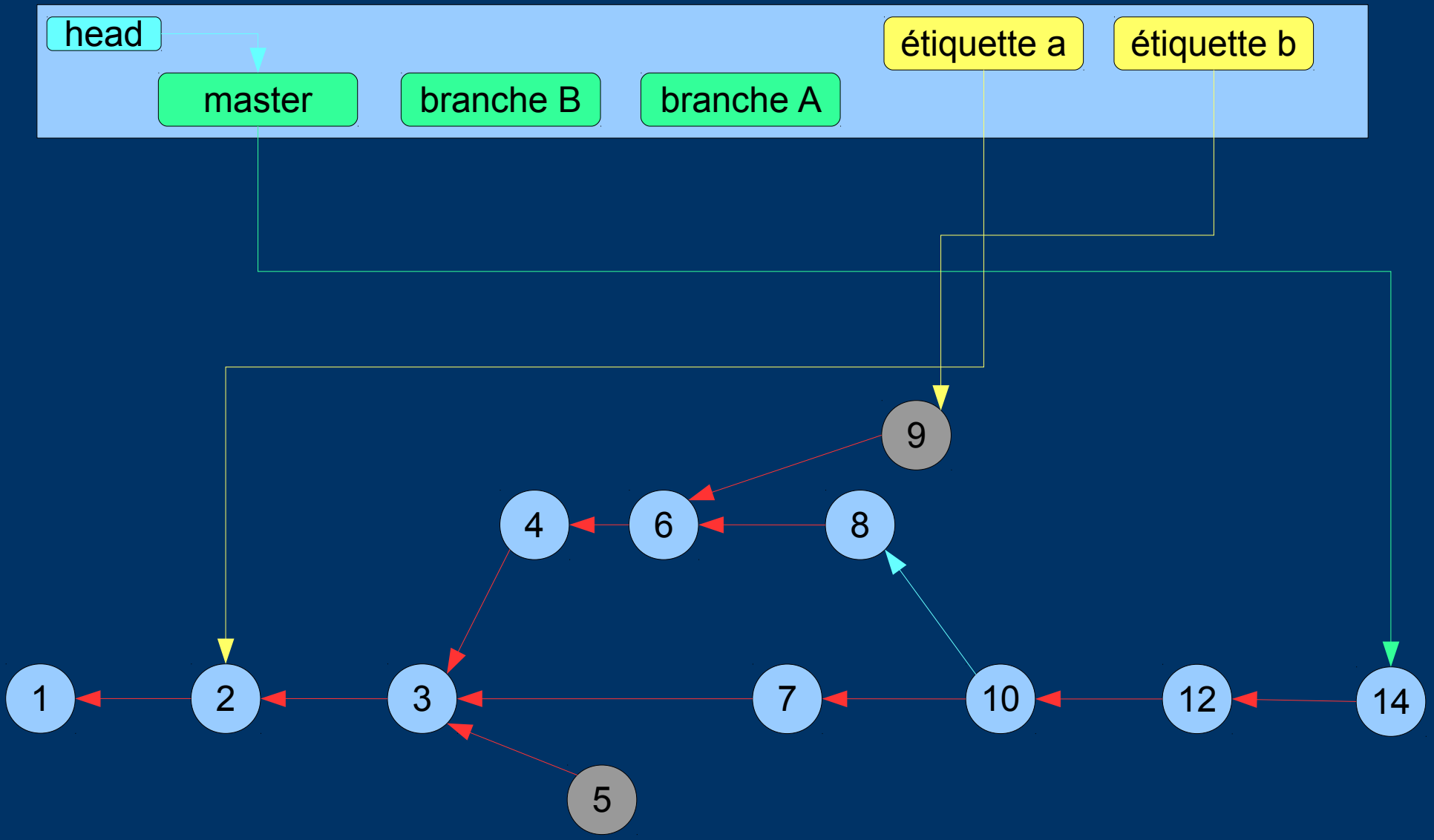
Gestion des révisions par GIT
Point de vue depuis « master » (2/6)



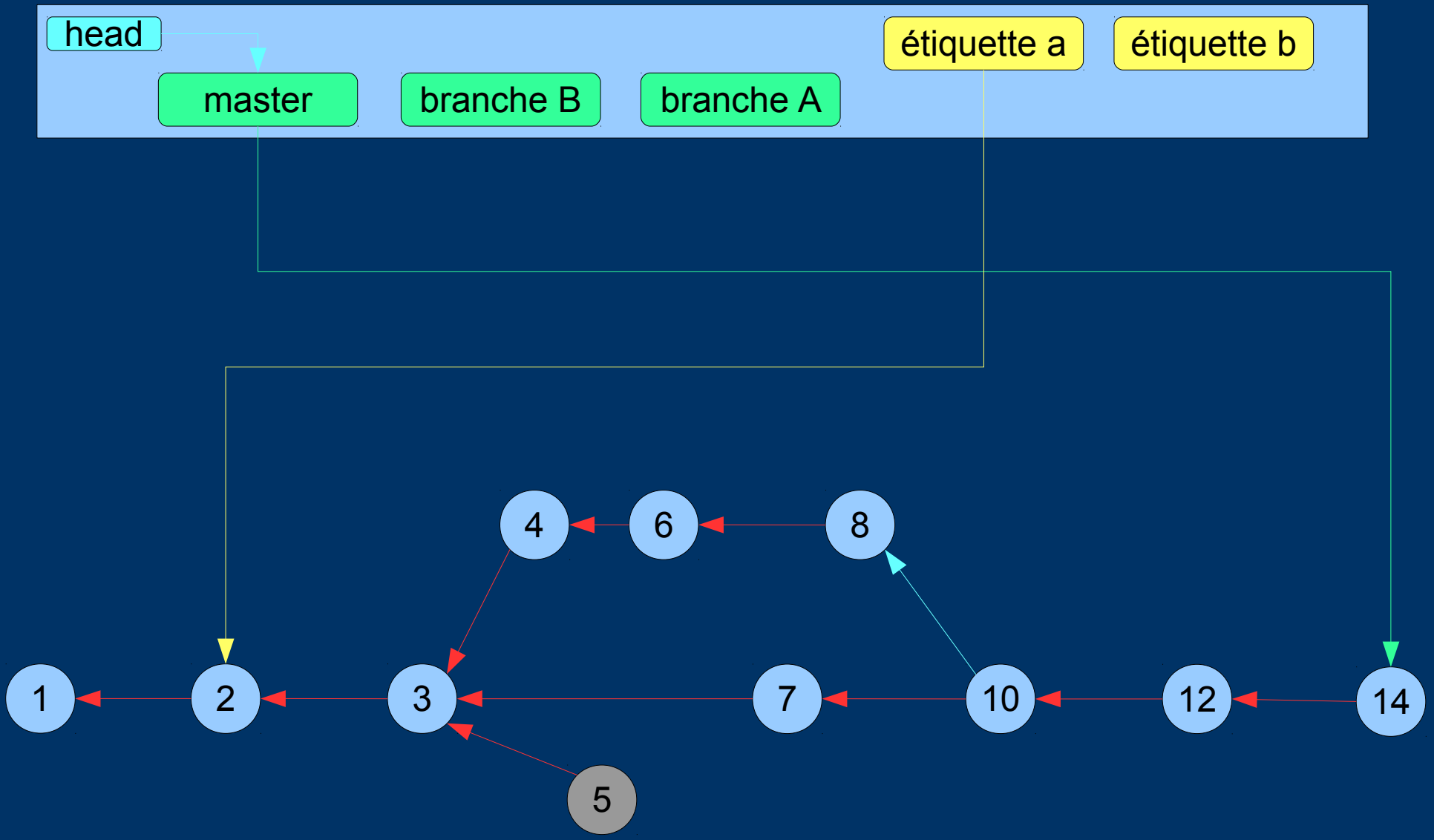
Gestion des révisions par GIT
Point de vue depuis « master » (3/6)



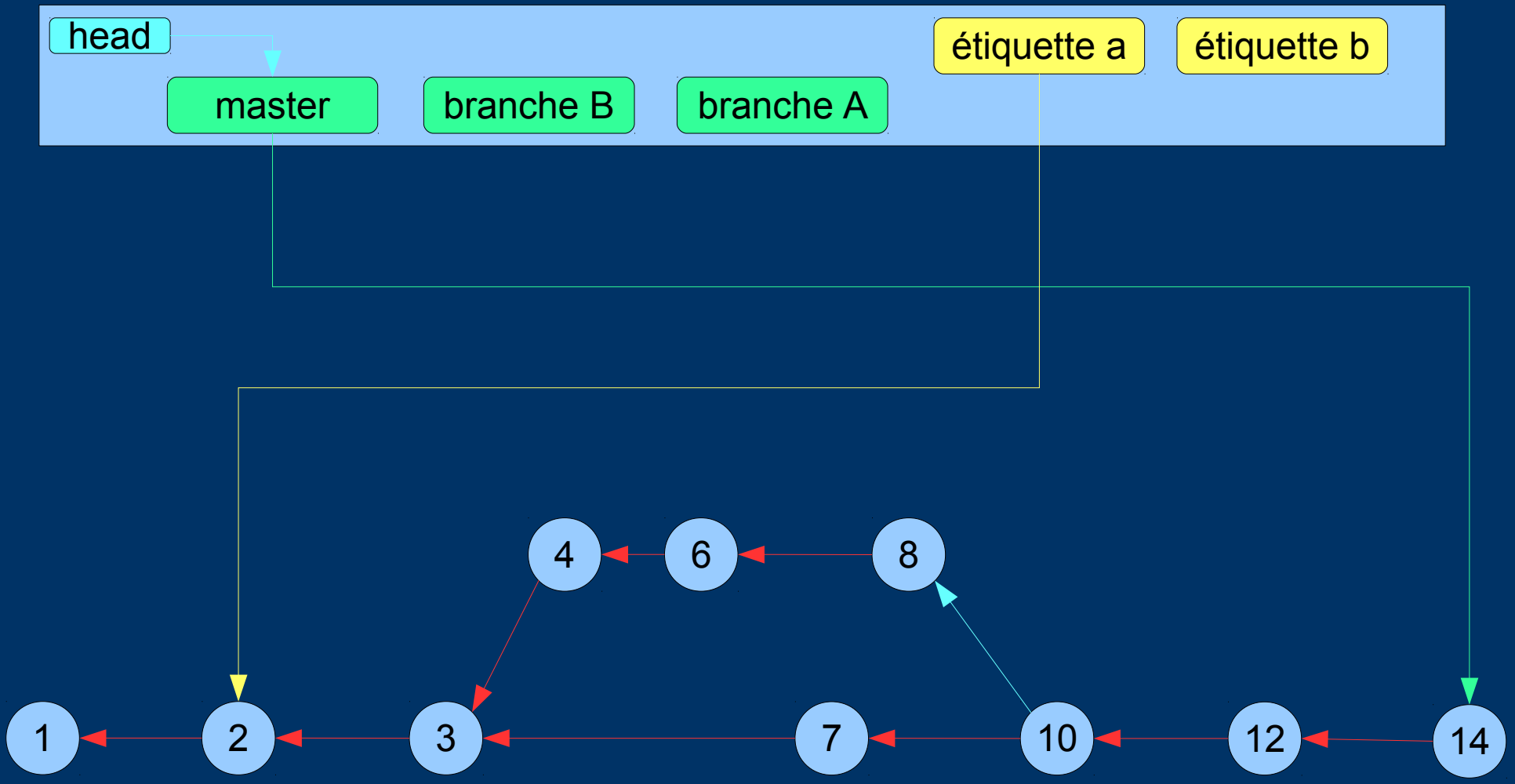
Gestion des révisions par GIT
Point de vue depuis « master » (4/6)



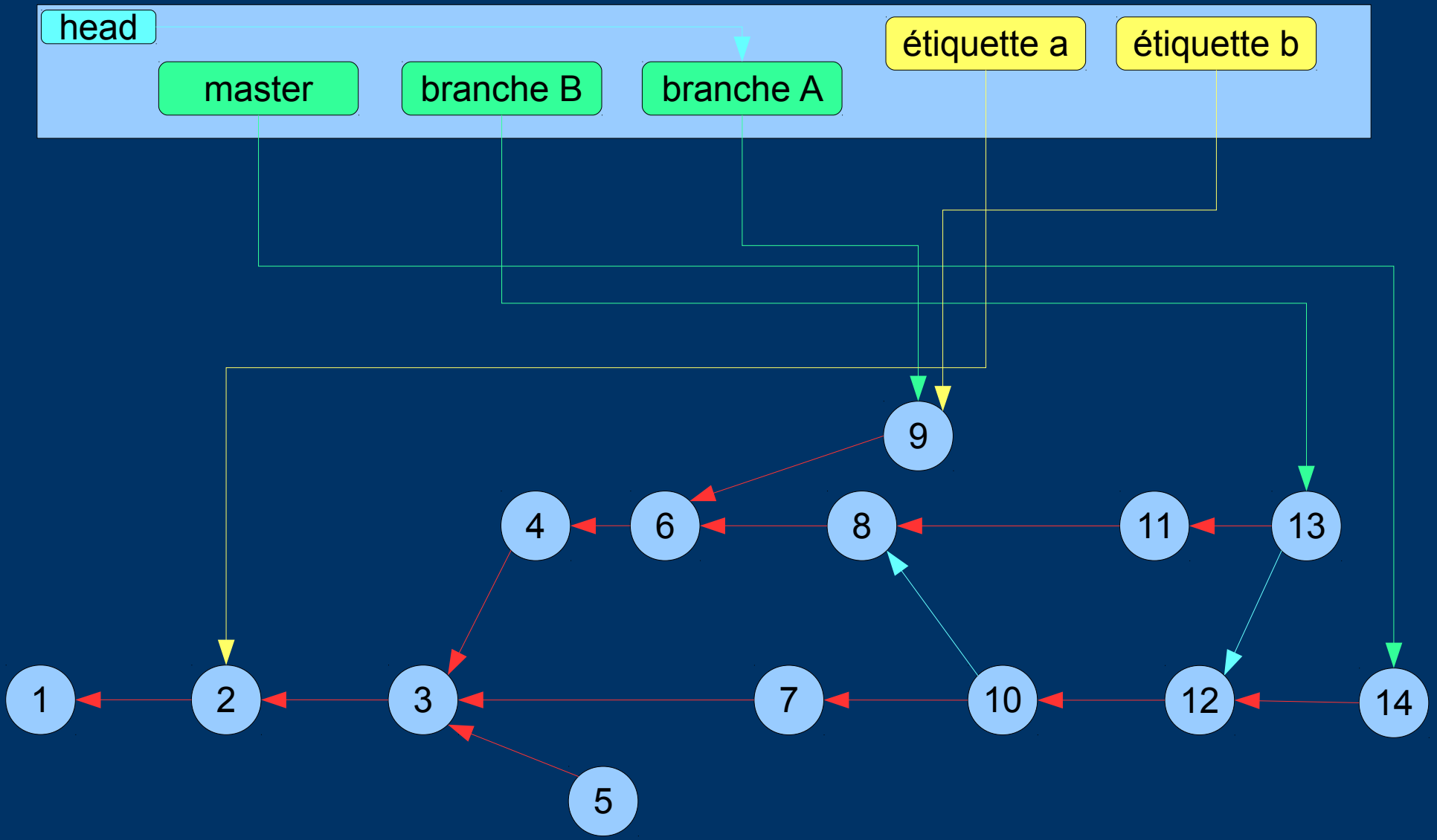
Gestion des révisions par GIT
Point de vue depuis « master » (5/6)



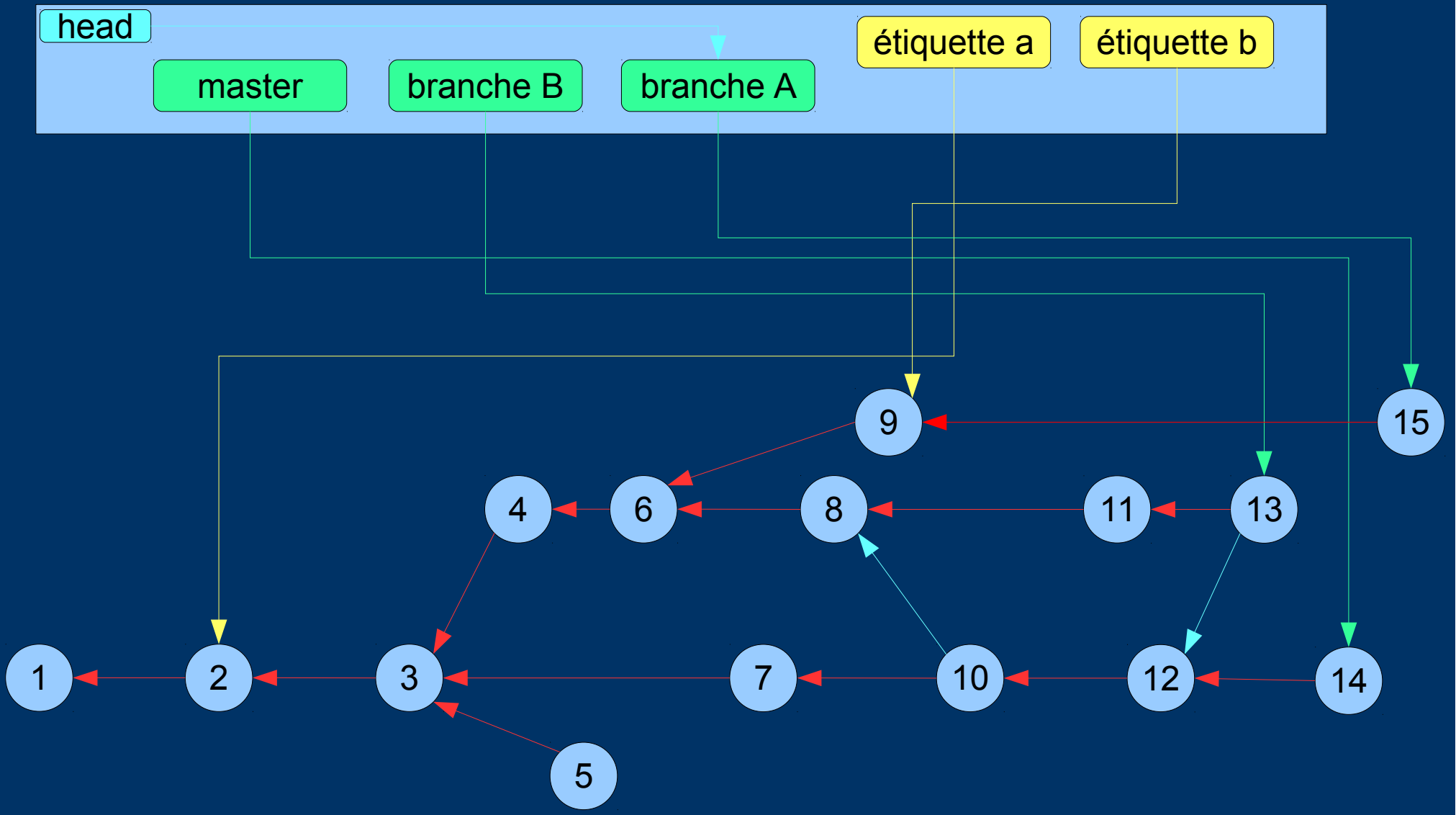
Gestion des révisions par GIT
Point de vue depuis « master » (6/6)



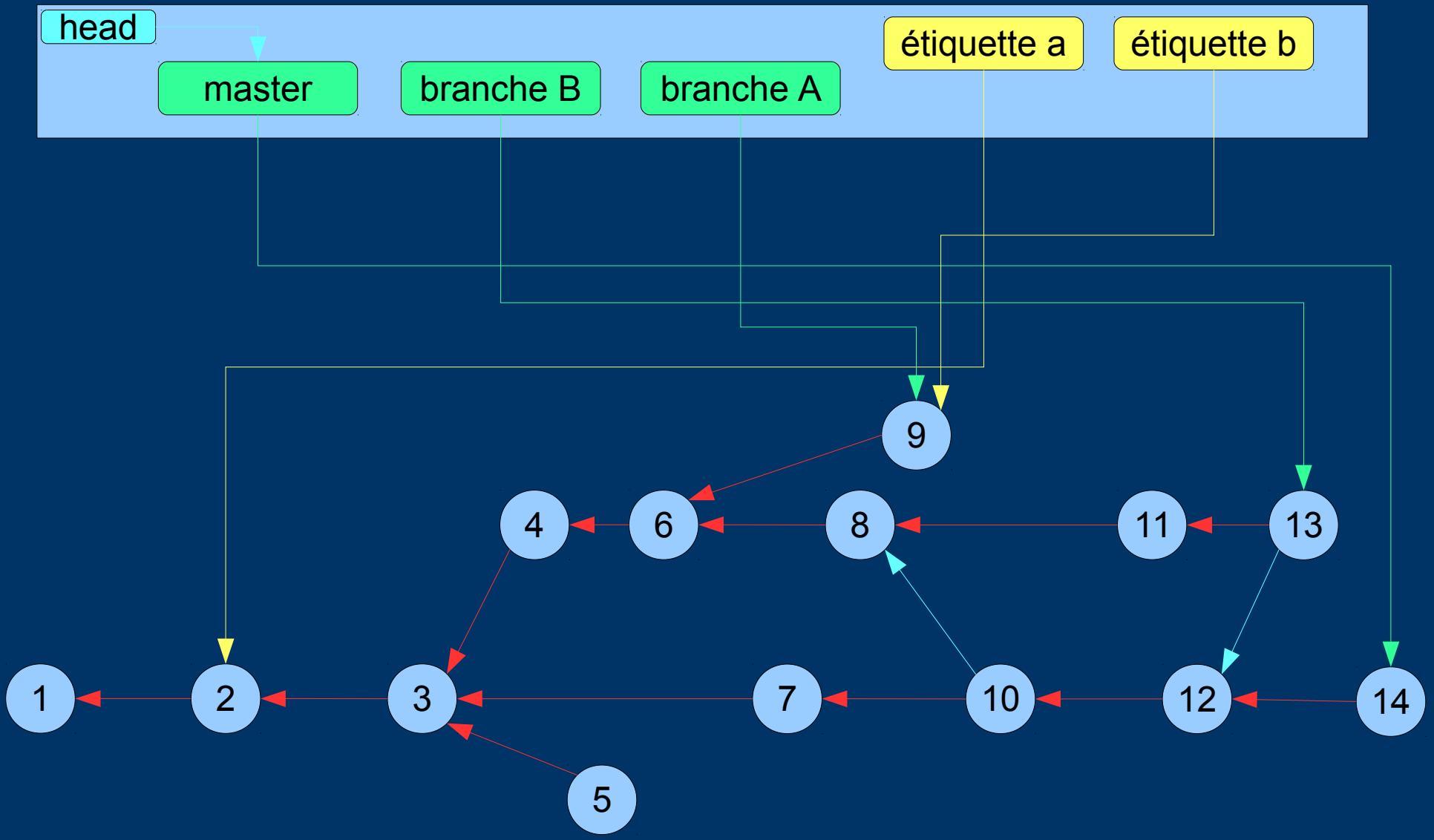
gestion des révisions par GIT
Nouveau commit sur la branche A (1/2)



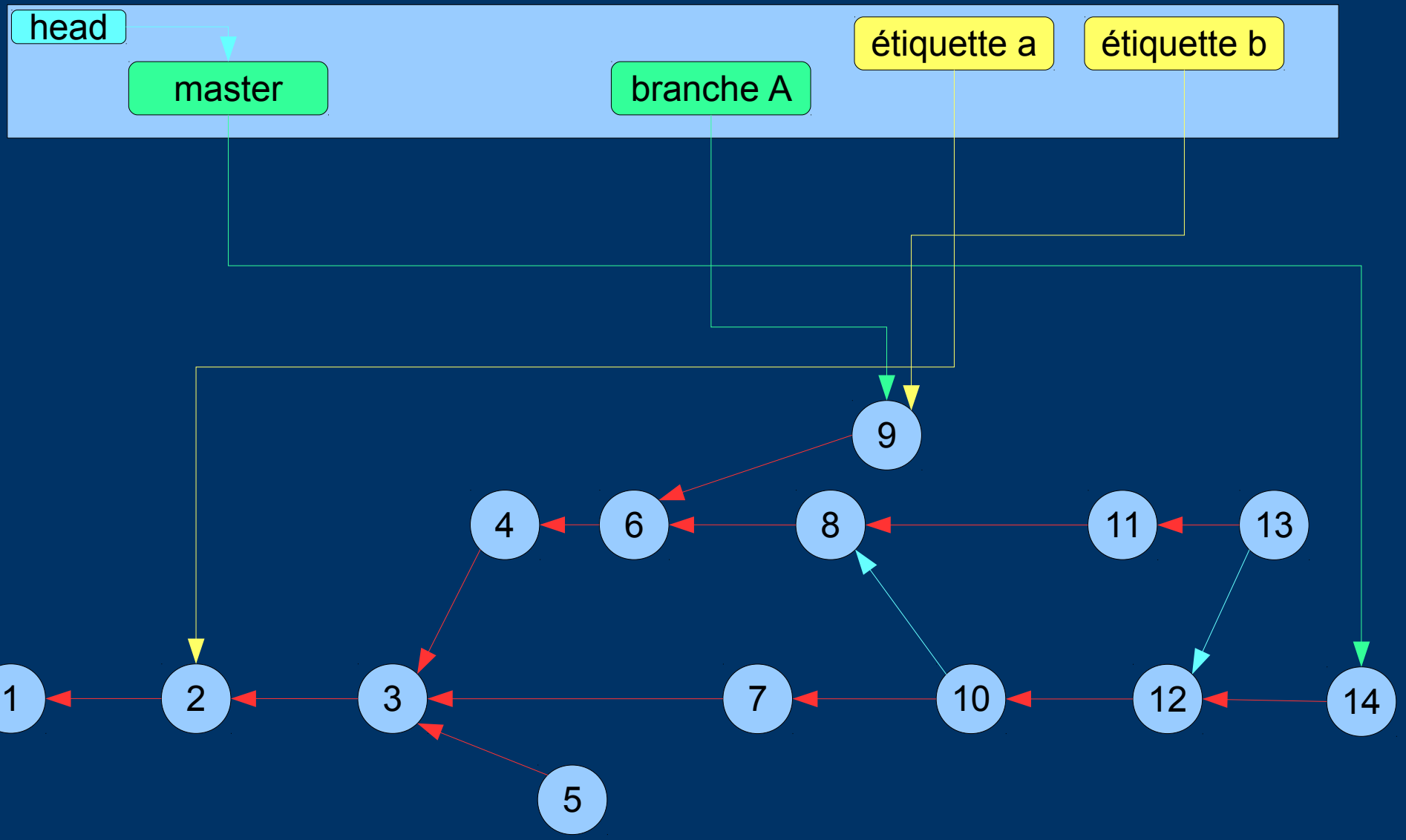
gestion des révisions par GIT
Nouveau commit sur la branche A (2/2)



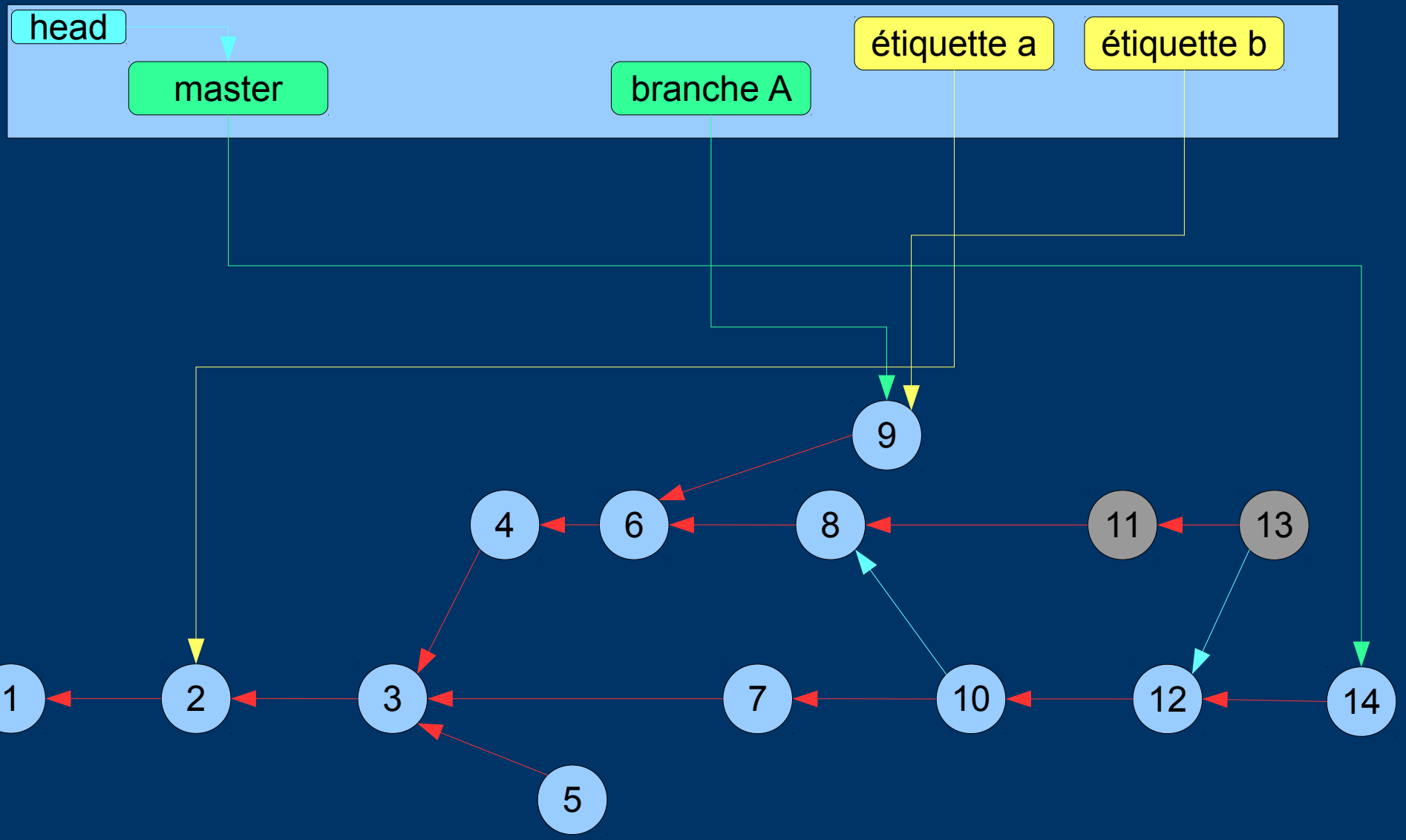
gestion des révisions par GIT
Suppression de la branche B (1/3)



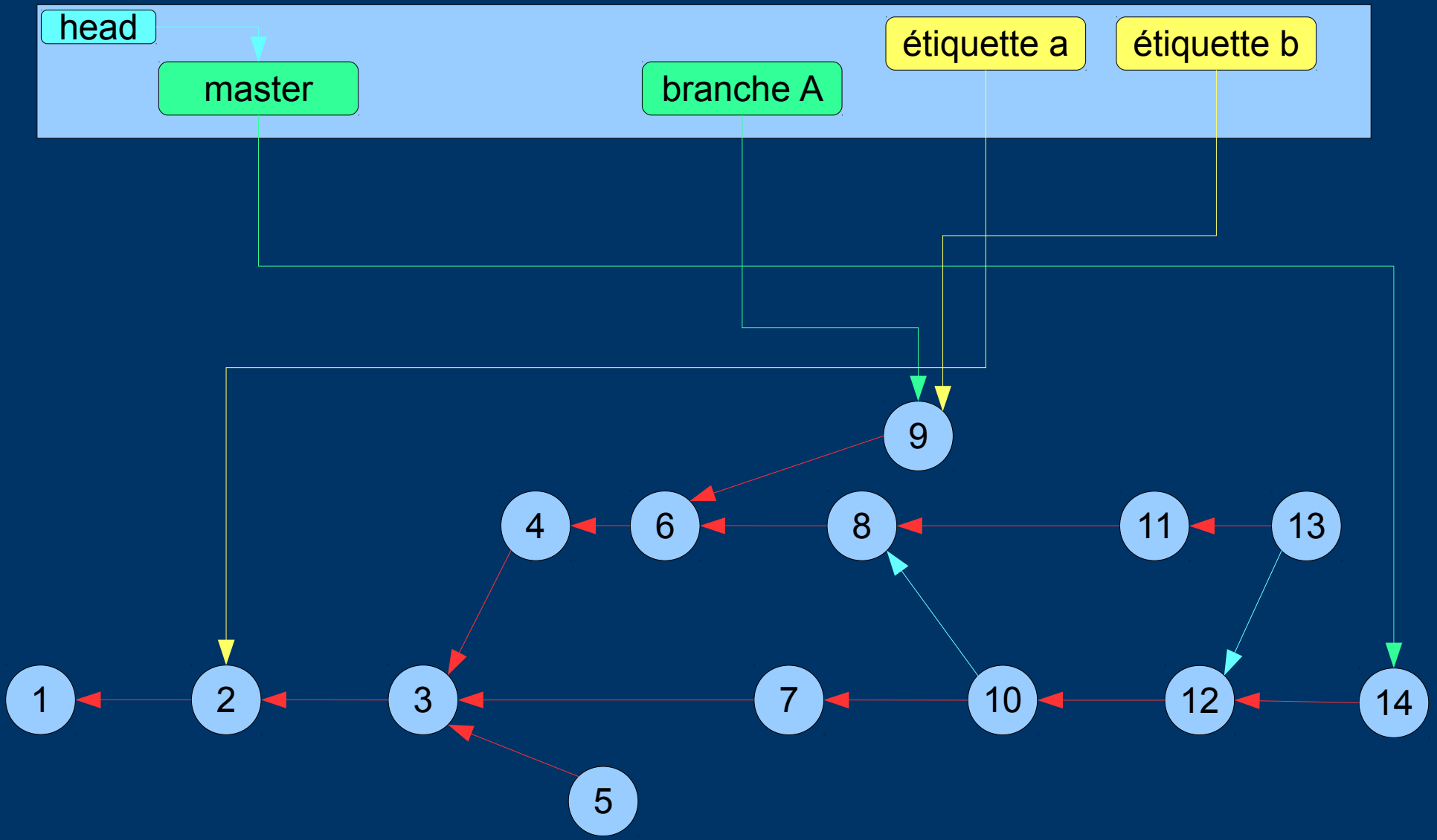
gestion des révisions par GIT
Suppression de la branche B (2/3)



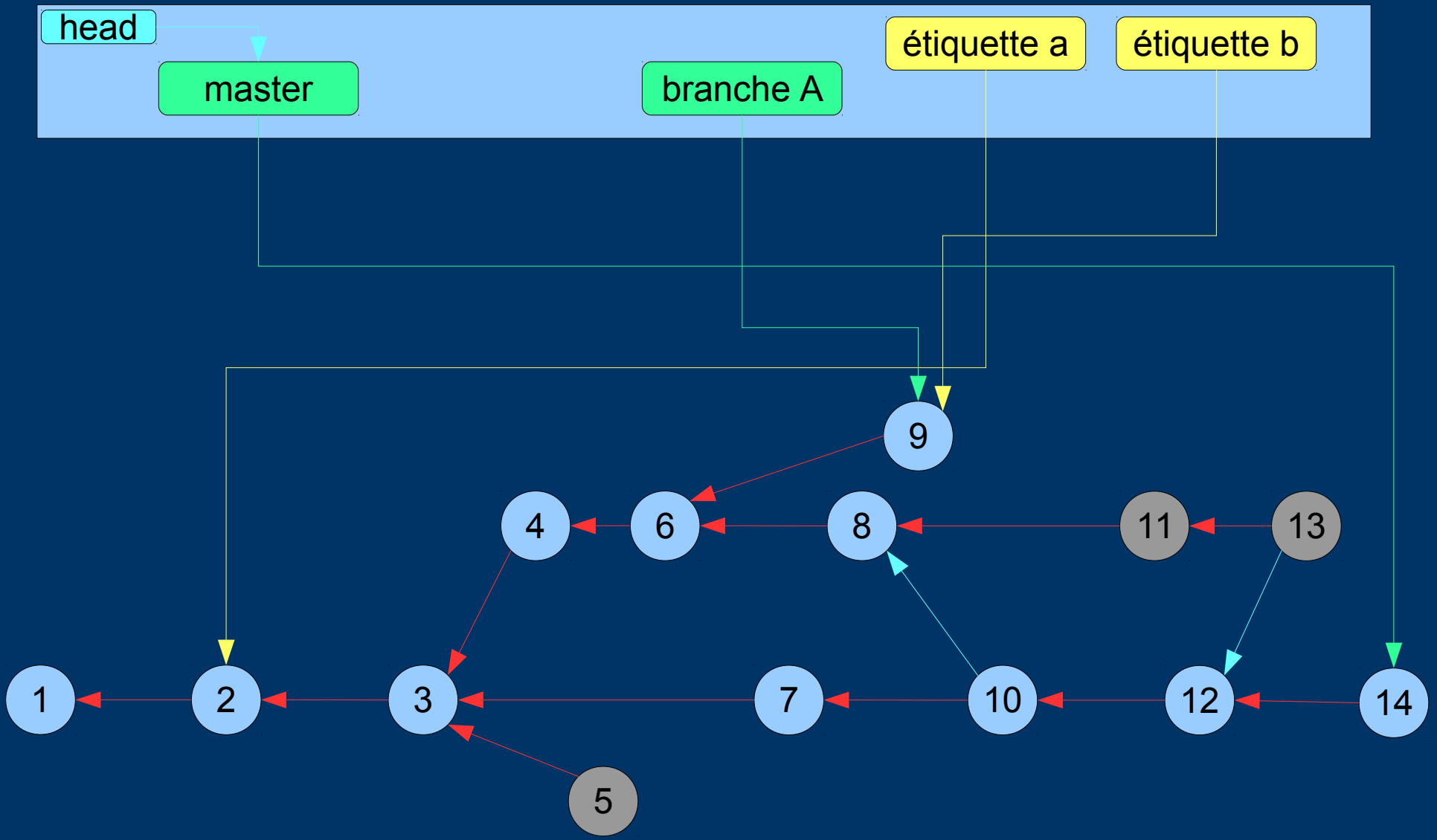
gestion des révisions par GIT
Suppression de la branche B (3/3)



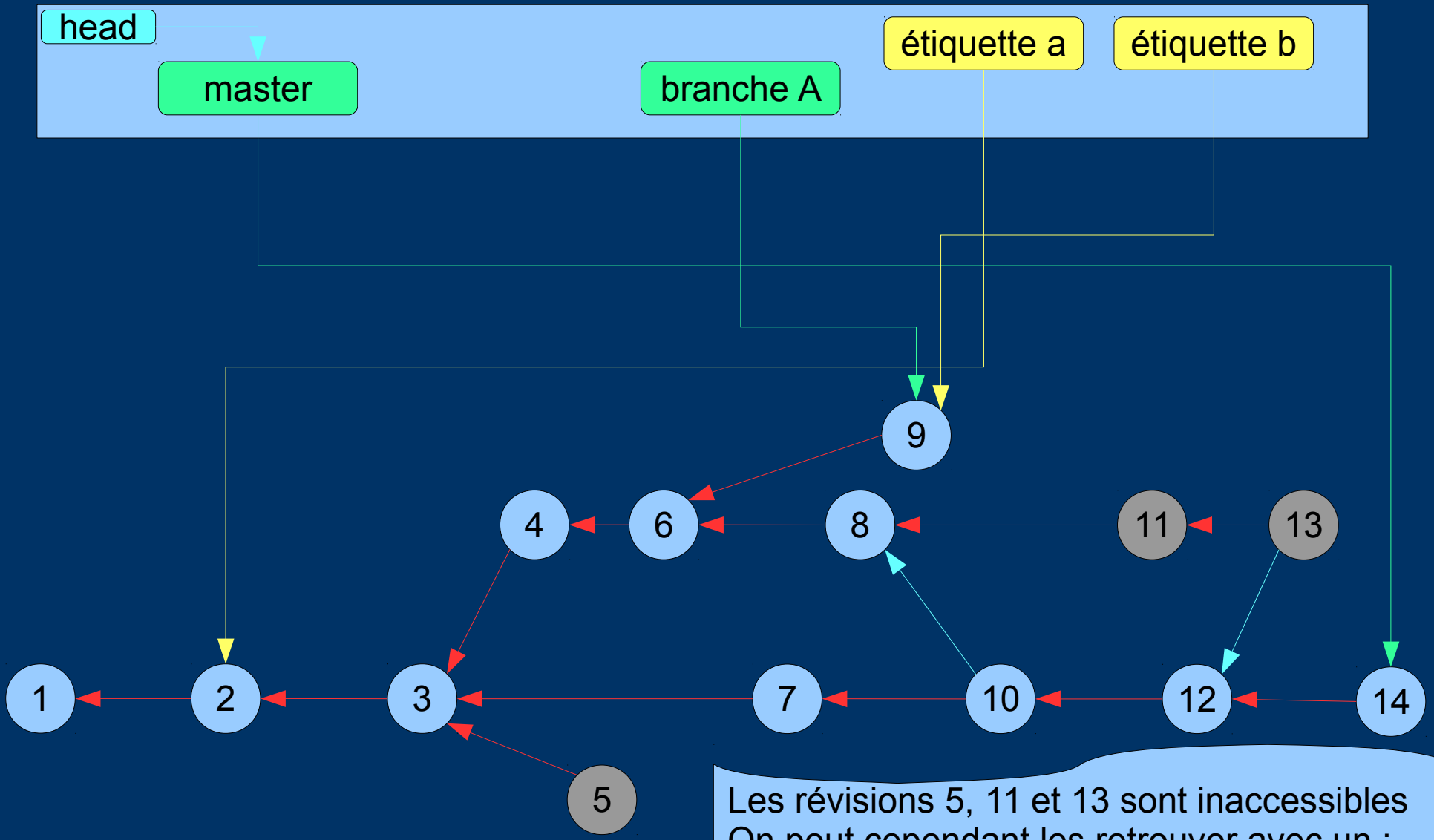
gestion des révisions par GIT
Révisions inaccessibles (1/3)



gestion des révisions par GIT
Révisions inaccessibles (2/3)



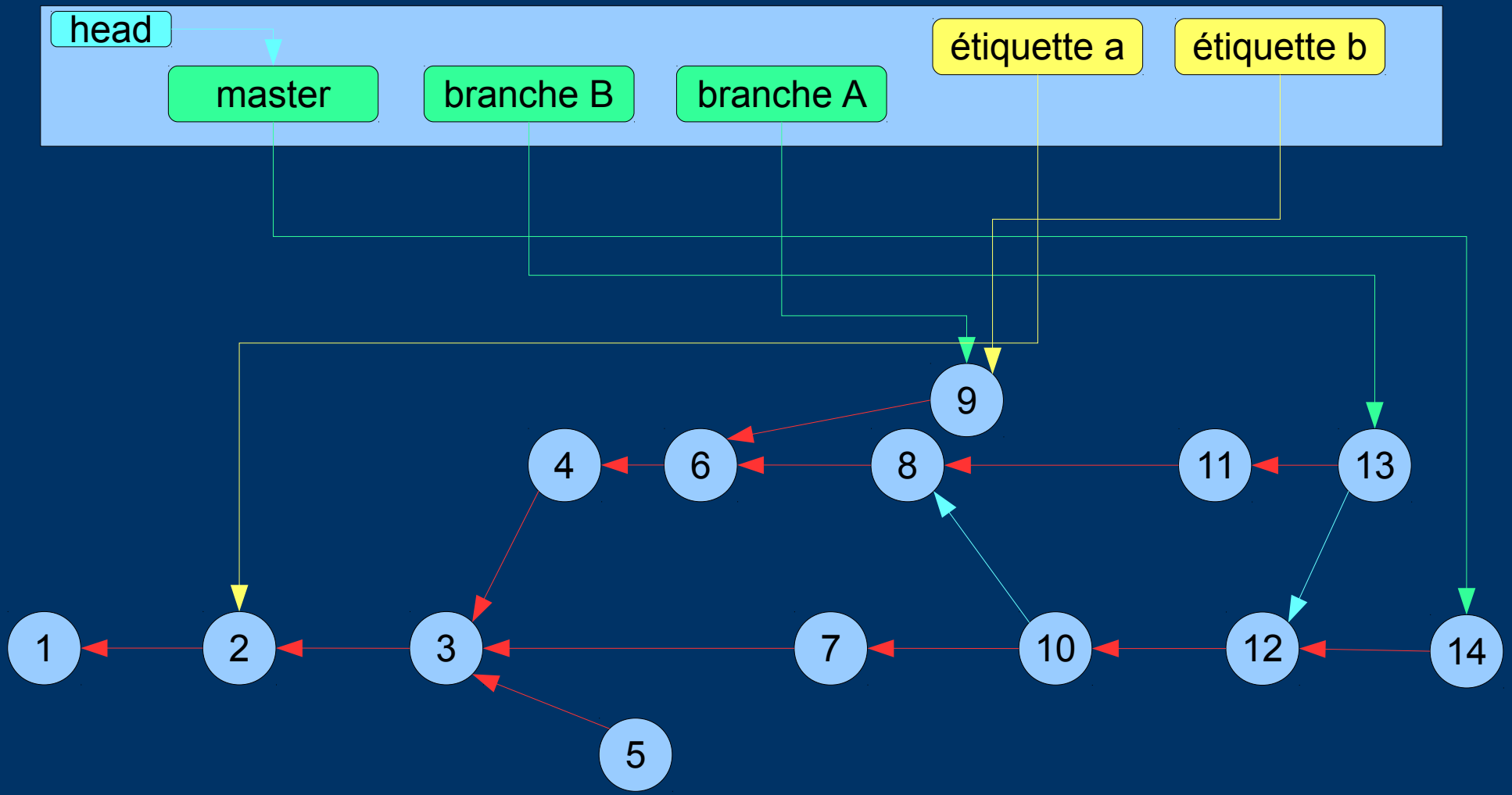
gestion des révisions par GIT
Révisions inaccessibles (3/3)



Les révisions 5, 11 et 13 sont inaccessibles
On peut cependant les retrouver avec un :
`git log --reflog`

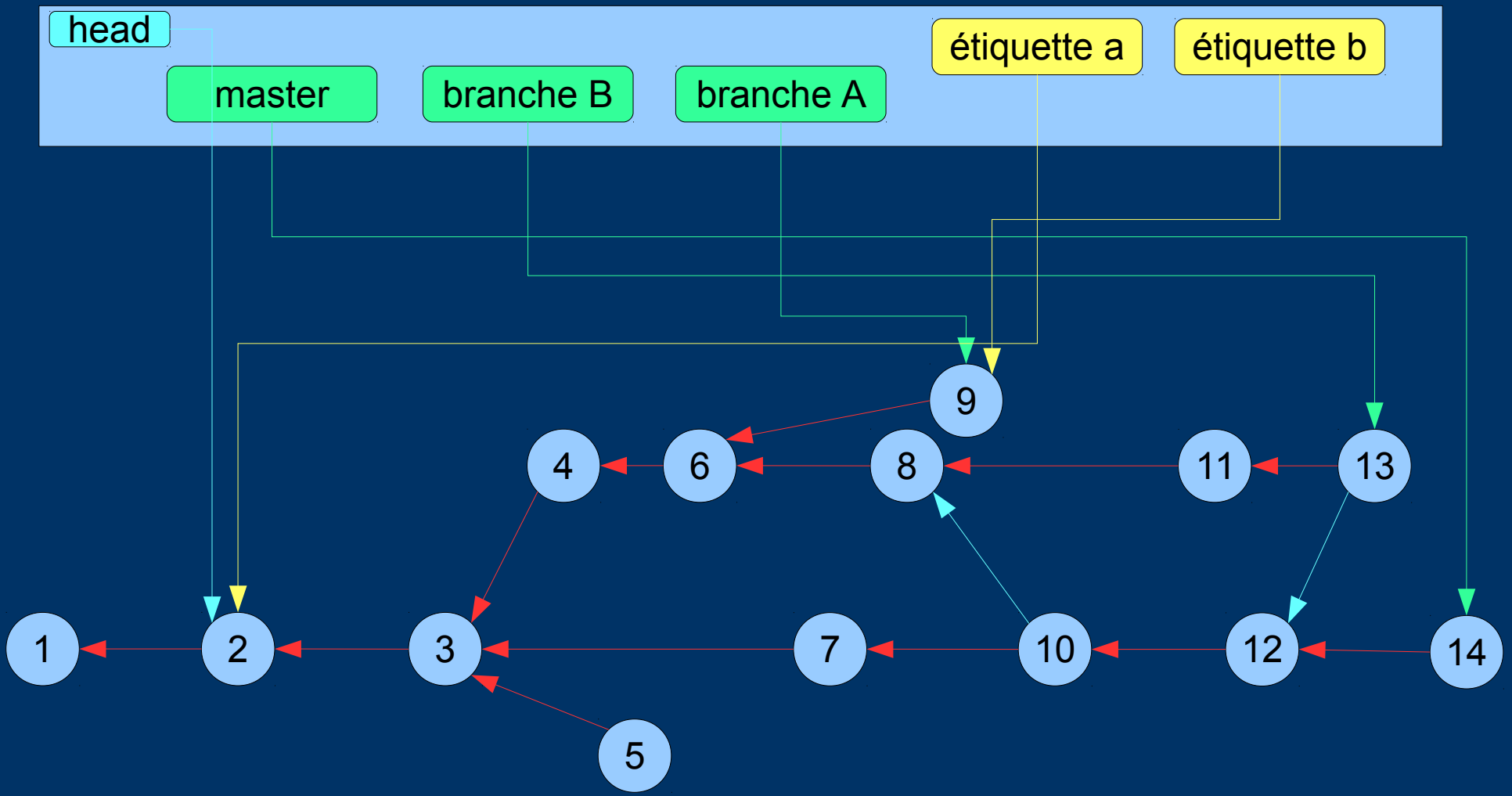
gestion des révisions par GIT
Récupération d'une version spécifique

```
git checkout ''étiquette a'' ou git checkout 2
```



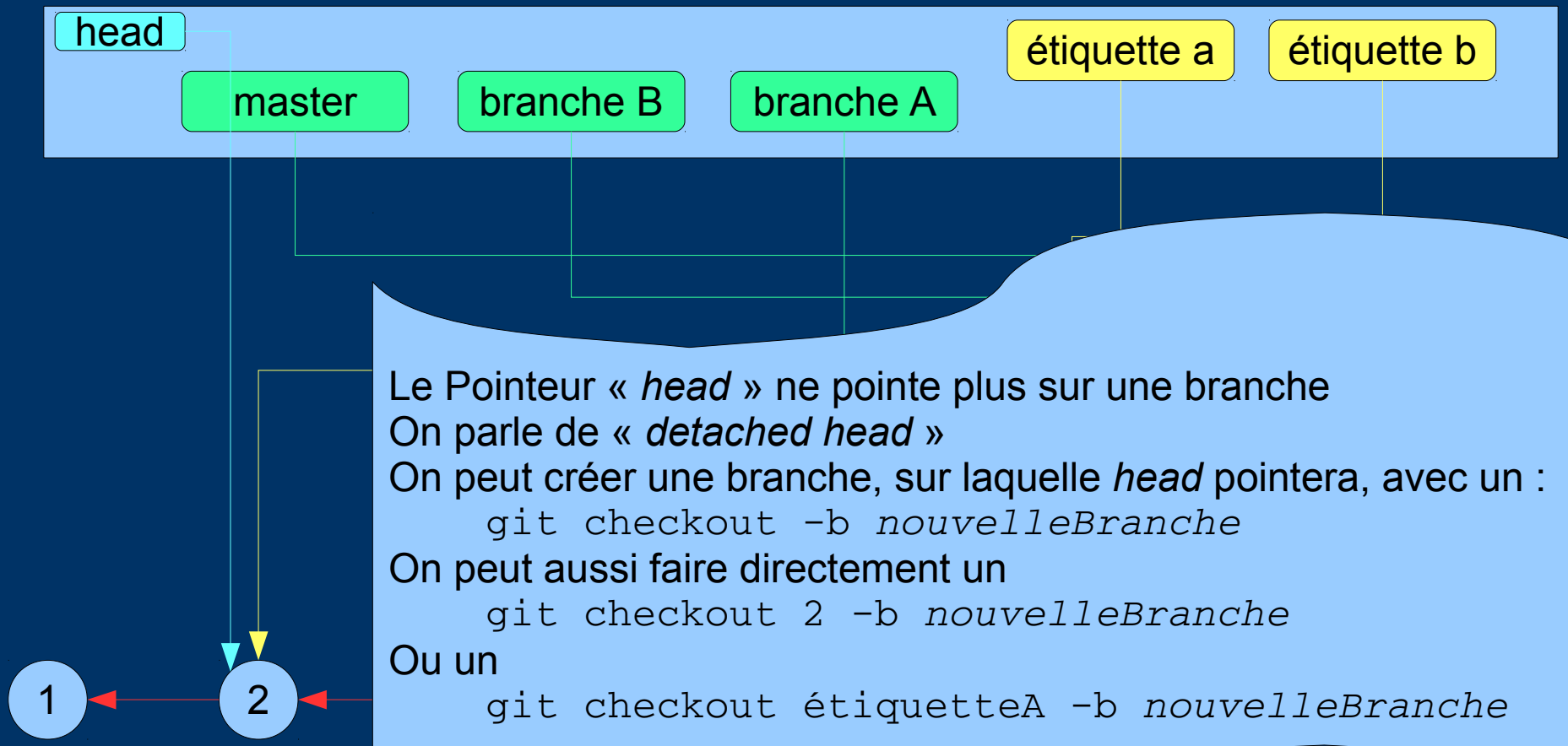
gestion des révisions par GIT
Récupération d'une version spécifique

```
git checkout ''étiquette a'' ou git checkout 2
```



gestion des révisions par GIT
Récupération d'une version spécifique

```
git checkout ''étiquette a'' ou git checkout 2
```



Le Pointeur « *head* » ne pointe plus sur une branche
On parle de « *detached head* »
On peut créer une branche, sur laquelle *head* pointera, avec un :

```
git checkout -b nouvelleBranche
```

On peut aussi faire directement un

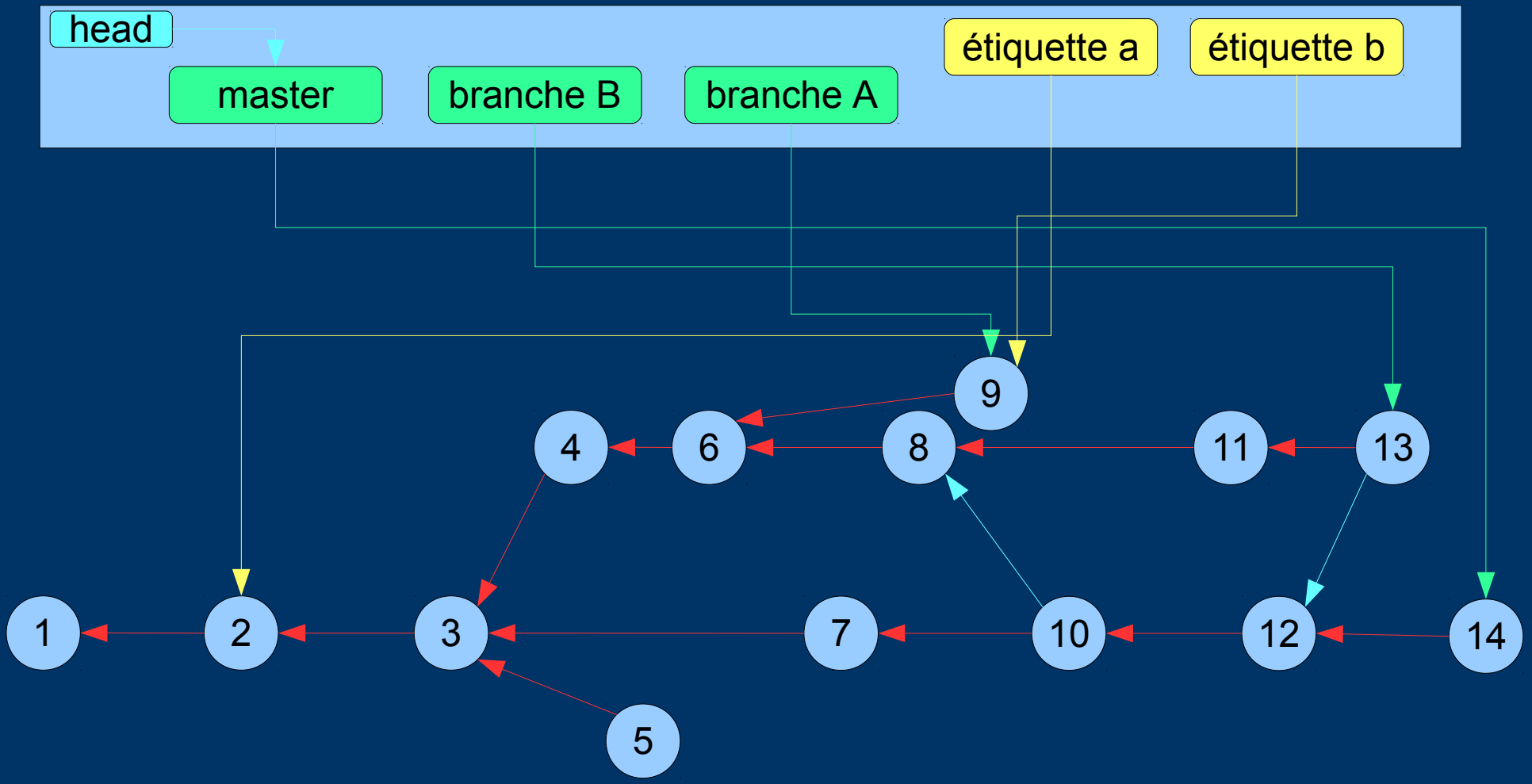
```
git checkout 2 -b nouvelleBranche
```

Ou un

```
git checkout étiquetteA -b nouvelleBranche
```

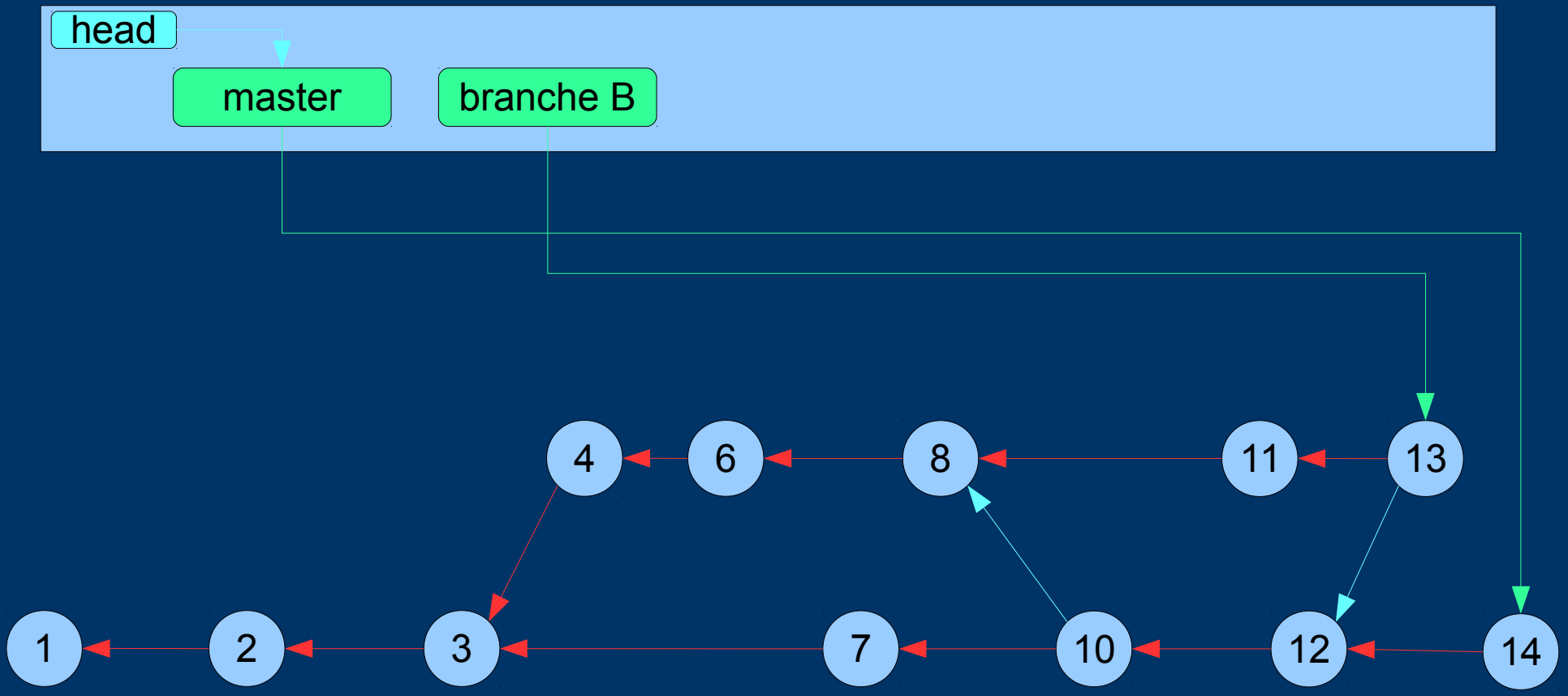
Principe de la fusion de branche
1. Plus jeune ancêtre commun (1/3)

git checkout master ; git merge brancheB



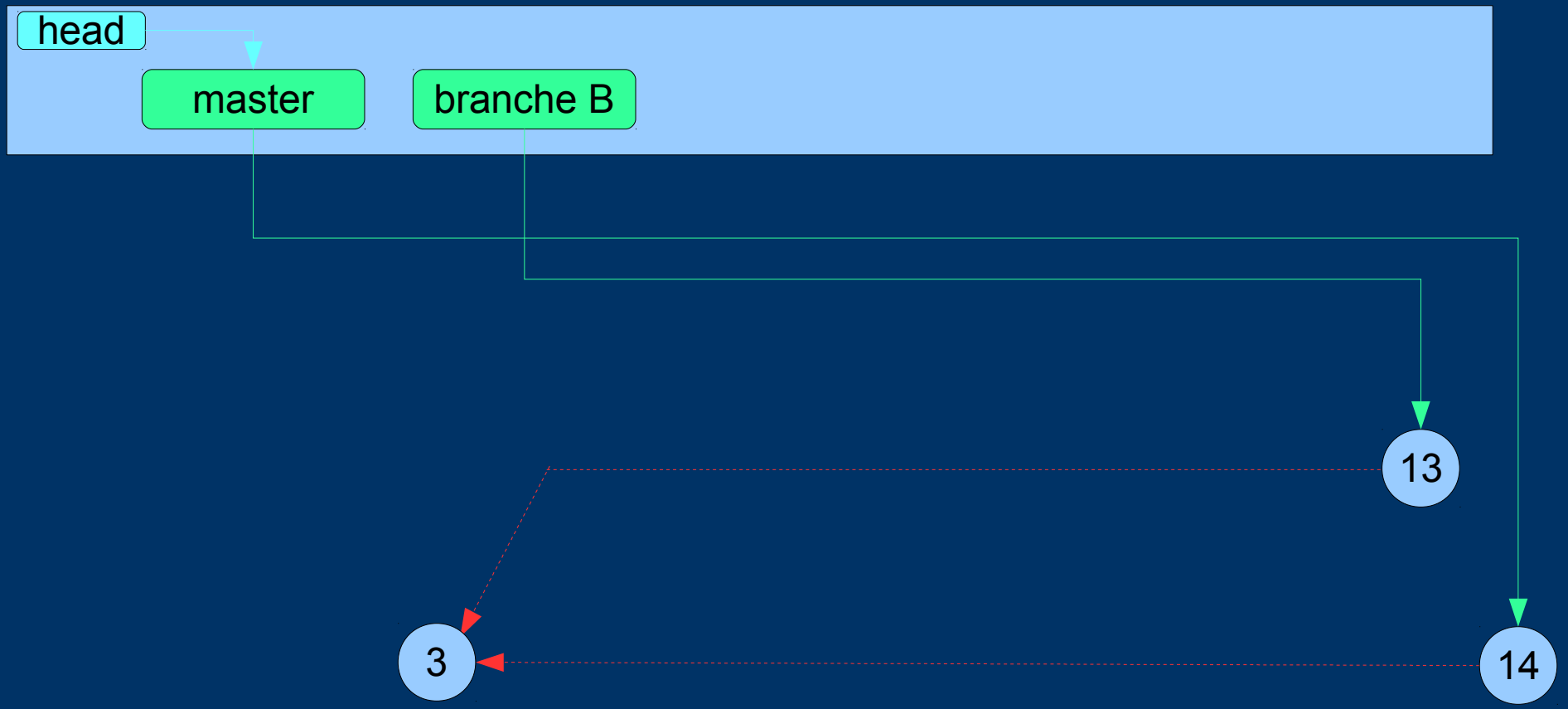
Principe de la fusion de branche
1. Plus jeune ancêtre commun (2/3)

git checkout master ; git merge brancheB



Principe de la fusion de branche
1. Plus jeune ancêtre commun (3/3)

git checkout master ; git merge brancheB



Principe de la fusion de branche
2. Recherche des différences dans le code

Ancêtre commun	Dernier master	Dernier brancheB	Fusion
ligne 1	ligne 1	ligne 1	
ligne 2	ajout a1	ligne 2	
ligne 3	ligne 2	ligne 3	
ligne 4	ligne 3	ligne 5	
ligne 5	ligne 4	ajout a3	
ligne 6	ligne 5	ligne 6	
ligne 7	ajout a2	ligne 7	
ligne 8	ligne 6	ligne 8	
ligne 9	ligne 7	ligne 9	
ligne 10	ligne 9	ligne 10	
ligne 11	ligne 10	ligne 11	
ligne 12	ligne 12	ajout a4	
		ligne 12	

Principe de la fusion de branche
2. Recherche des différences dans le code

Ancêtre commun	Dernier master	Dernier brancheB	Fusion
ligne 1	ligne 1	ligne 1	
ligne 2	ajout a1	ligne 2	
ligne 3	ligne 2	ligne 3	
ligne 4	ligne 3	ligne 5	
ligne 5	ligne 4	ajout a3	
ligne 6	ligne 5	ligne 6	
ligne 7	ajout a2	ligne 7	
ligne 8	ligne 6	ligne 8	
ligne 9	ligne 7	ligne 9	
ligne 10	ligne 9	ligne 10	
ligne 11	ligne 10	ligne 11	
ligne 12	ligne 12	ajout a4	
		ligne 12	

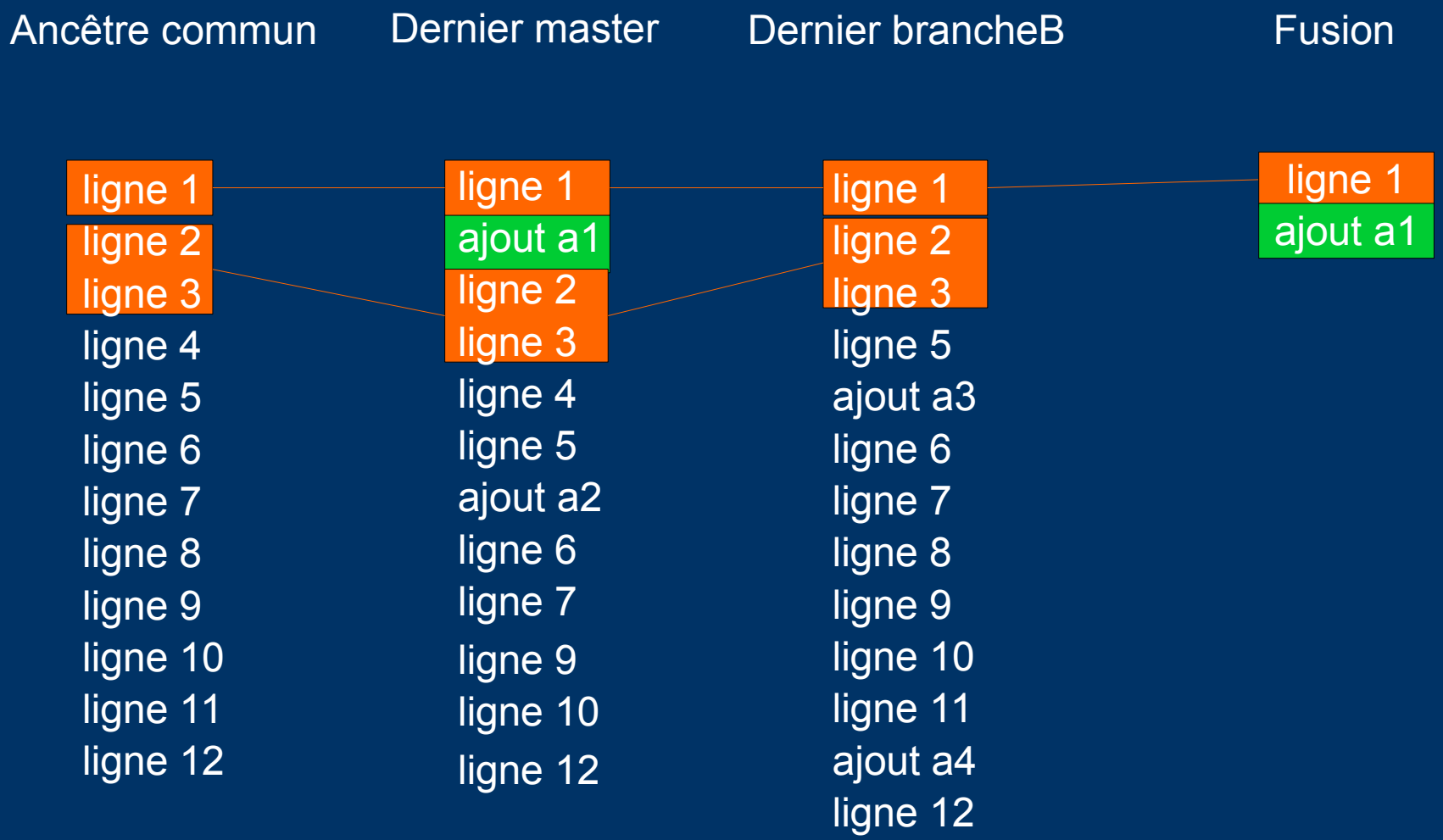
Principe de la fusion de branche
2. Recherche des différences dans le code

Ancêtre commun	Dernier master	Dernier brancheB	Fusion
ligne 1	ligne 1	ligne 1	ligne 1
ligne 2	ajout a1	ligne 2	
ligne 3	ligne 2	ligne 3	
ligne 4	ligne 3	ligne 5	
ligne 5	ligne 4	ajout a3	
ligne 6	ligne 5	ligne 6	
ligne 7	ajout a2	ligne 7	
ligne 8	ligne 6	ligne 8	
ligne 9	ligne 7	ligne 9	
ligne 10	ligne 9	ligne 10	
ligne 11	ligne 10	ligne 11	
ligne 12	ligne 12	ajout a4	
		ligne 12	

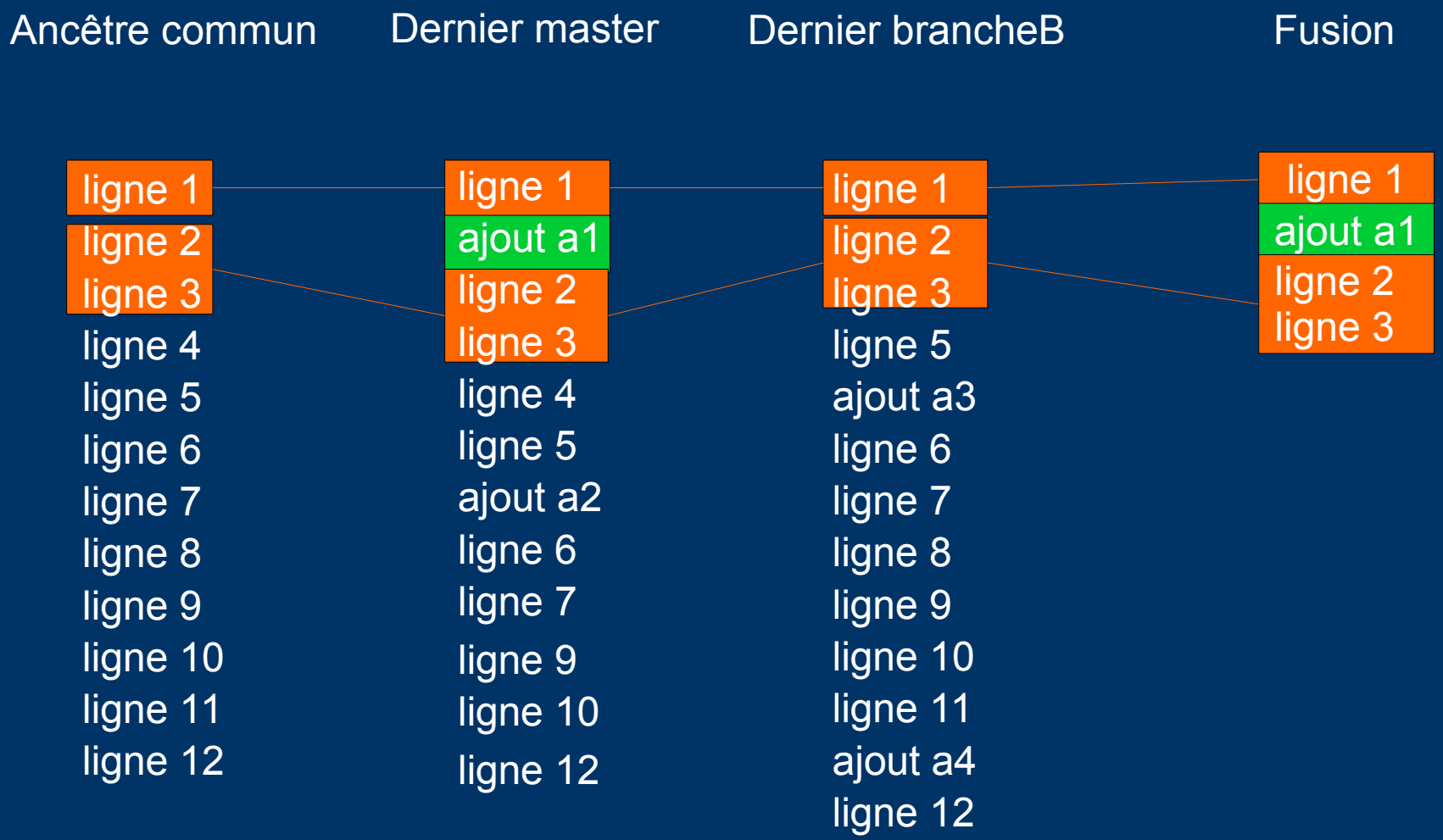
Principe de la fusion de branche
2. Recherche des différences dans le code

Principe de la fusion de branche
2. Recherche des différences dans le code

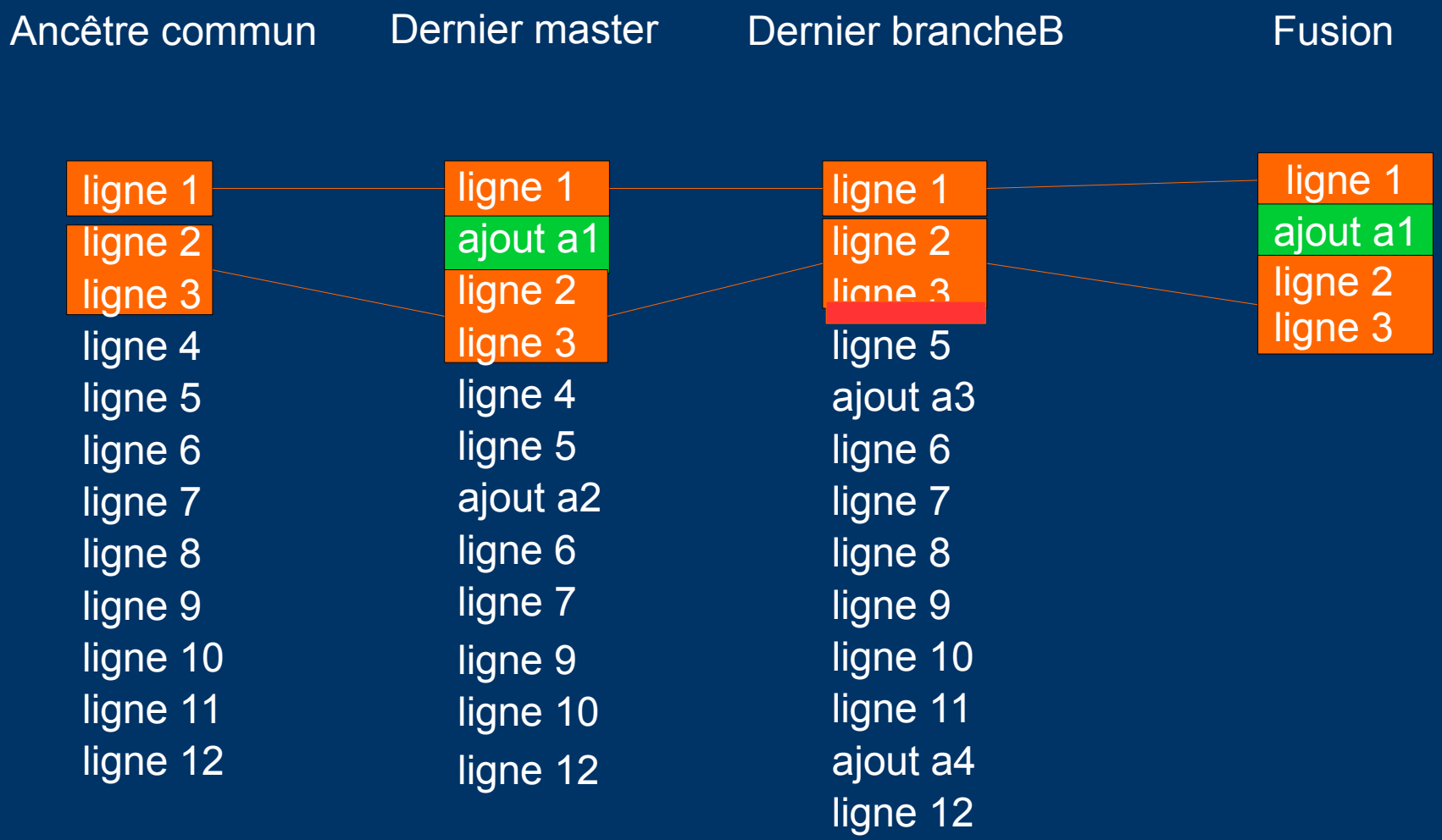
Principe de la fusion de branche
2. Recherche des différences dans le code



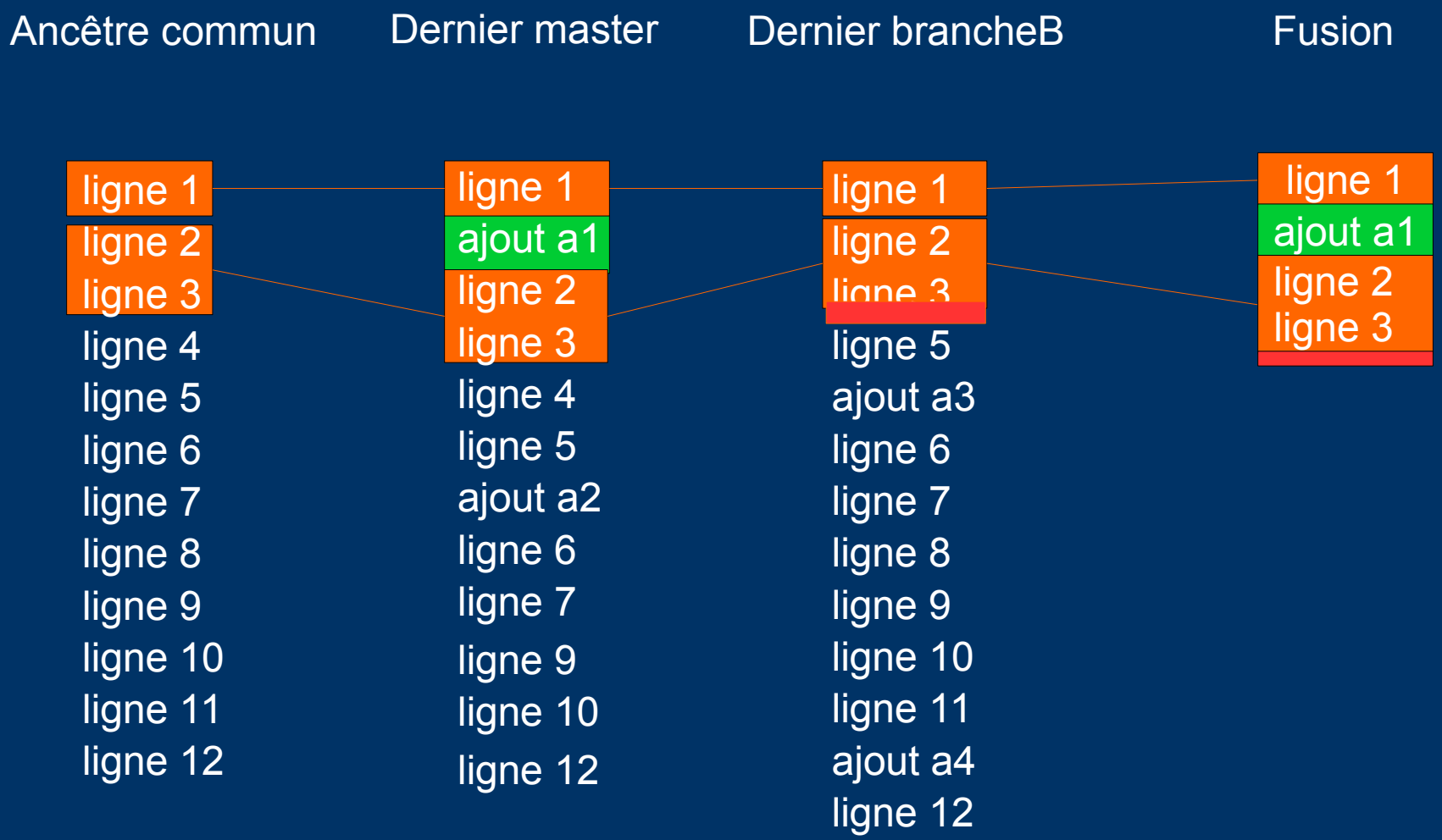
Principe de la fusion de branche
2. Recherche des différences dans le code



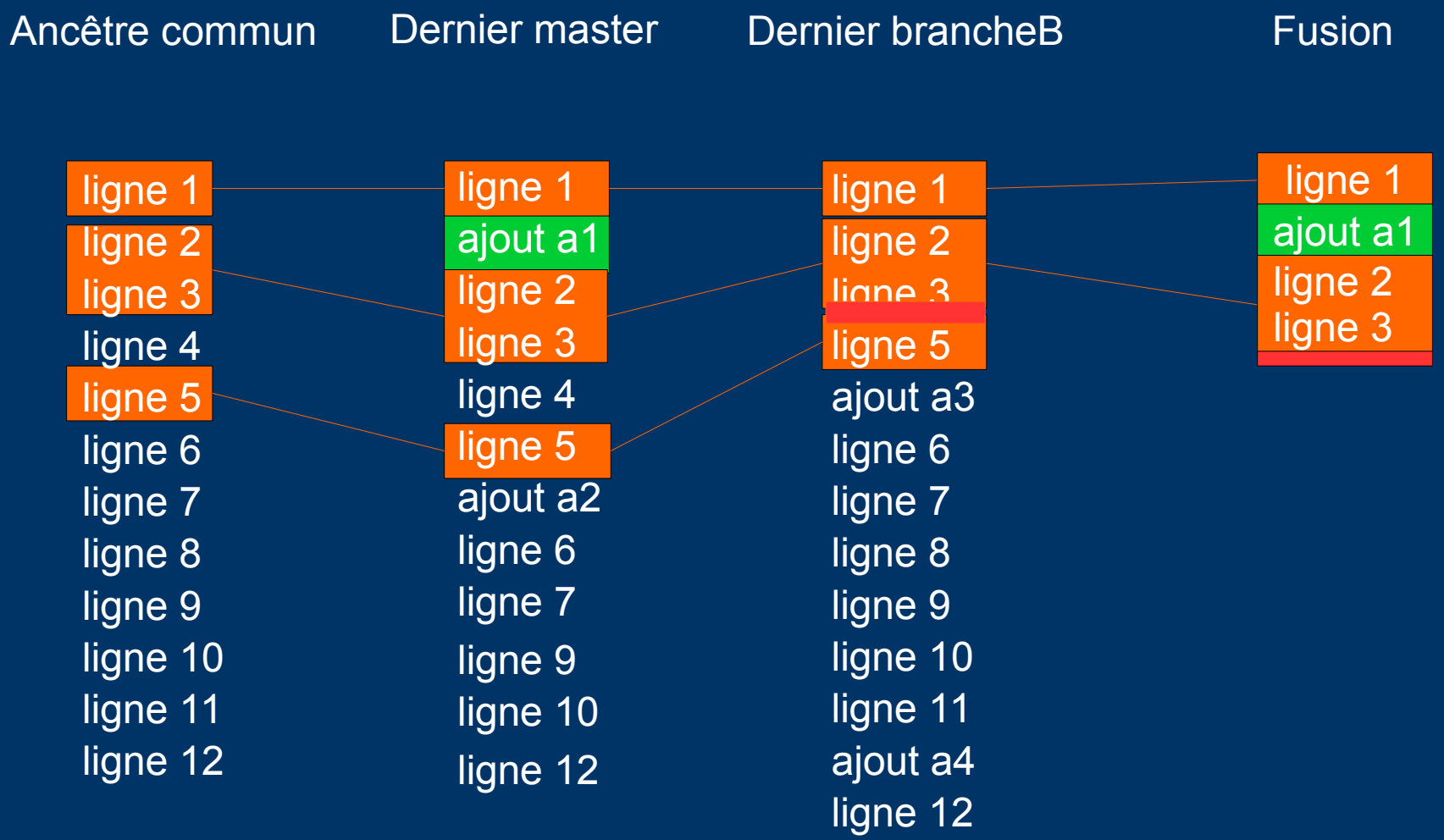
Principe de la fusion de branche
2. Recherche des différences dans le code



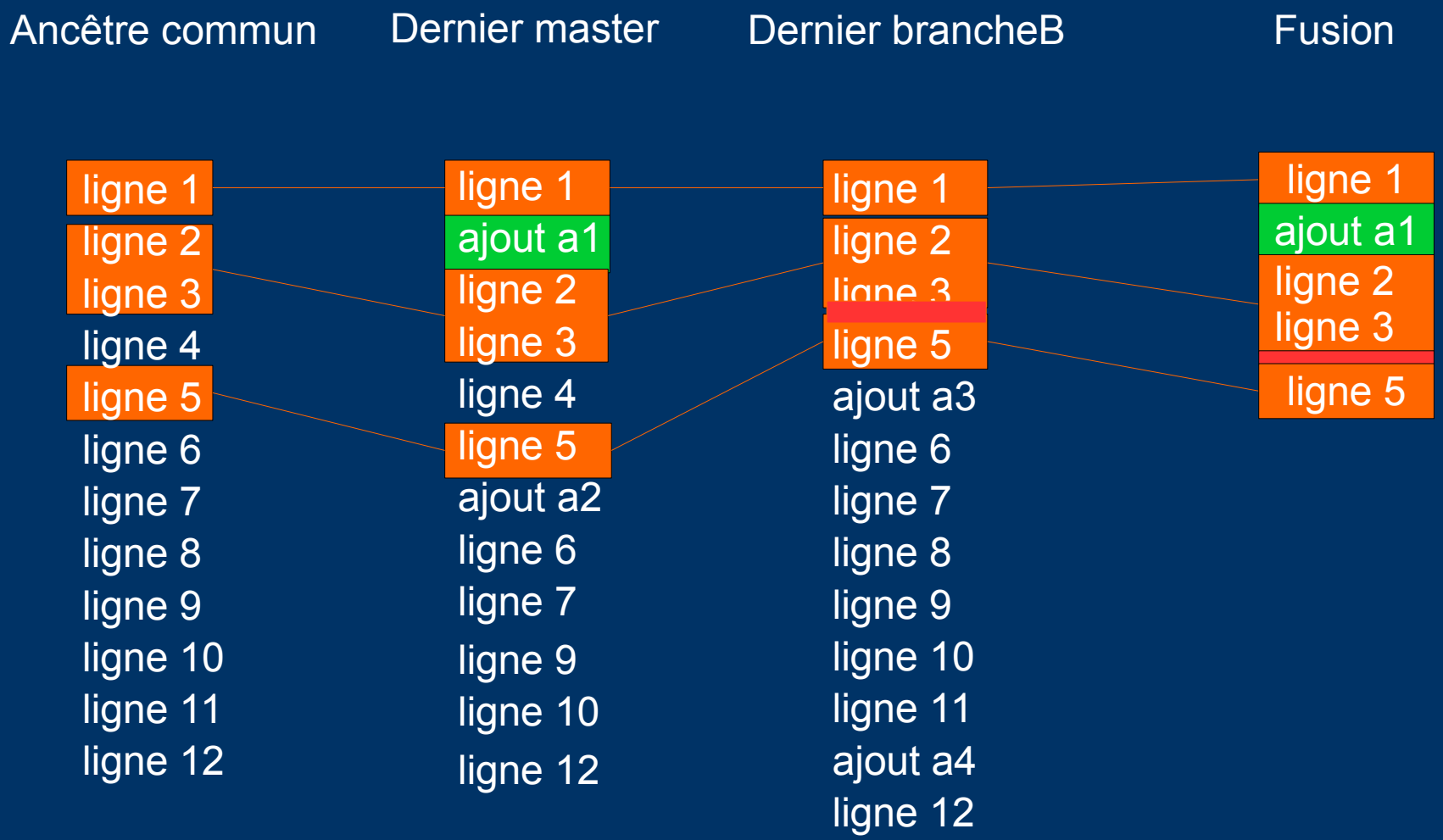
Principe de la fusion de branche
2. Recherche des différences dans le code



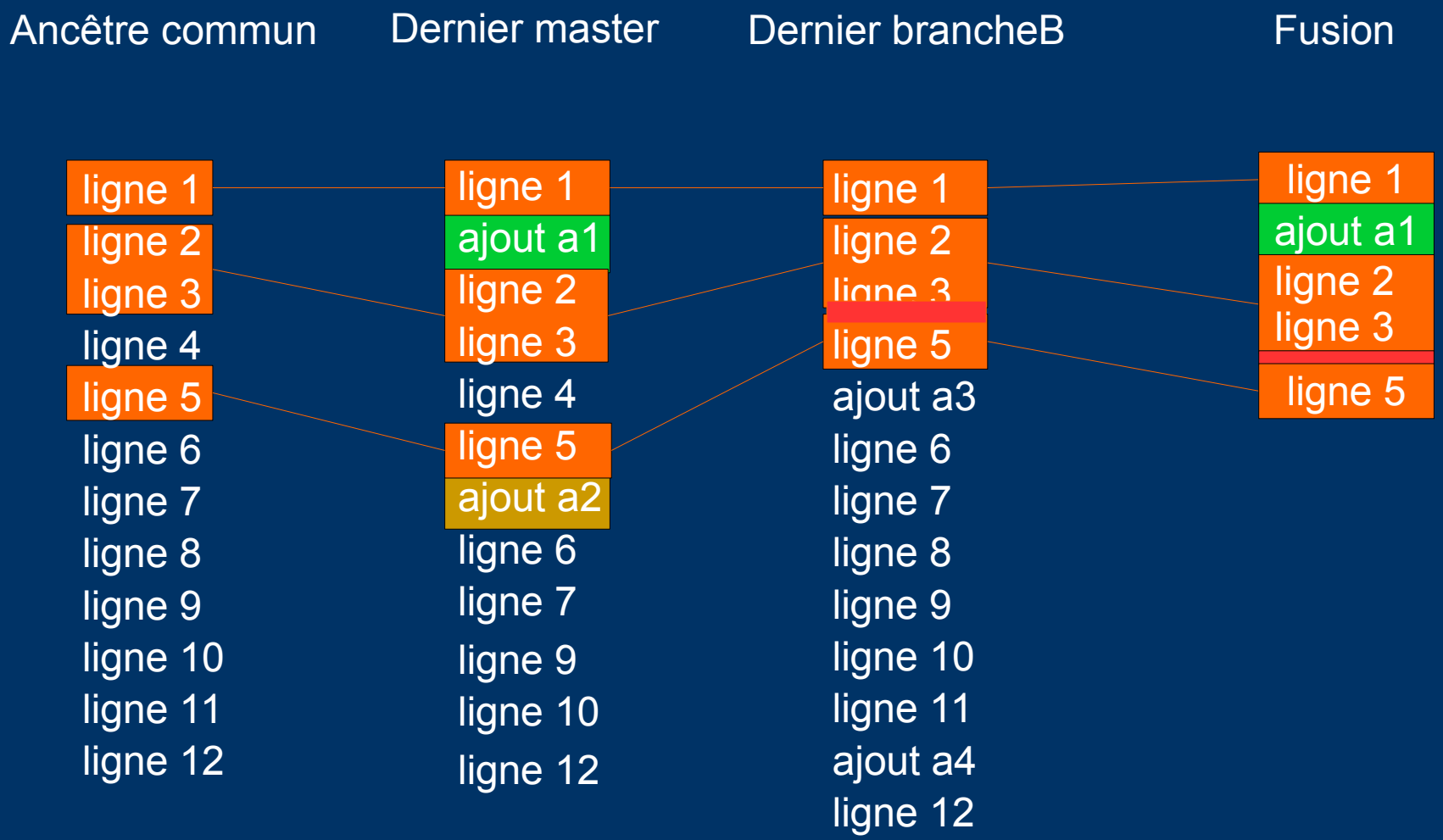
Principe de la fusion de branche
2. Recherche des différences dans le code



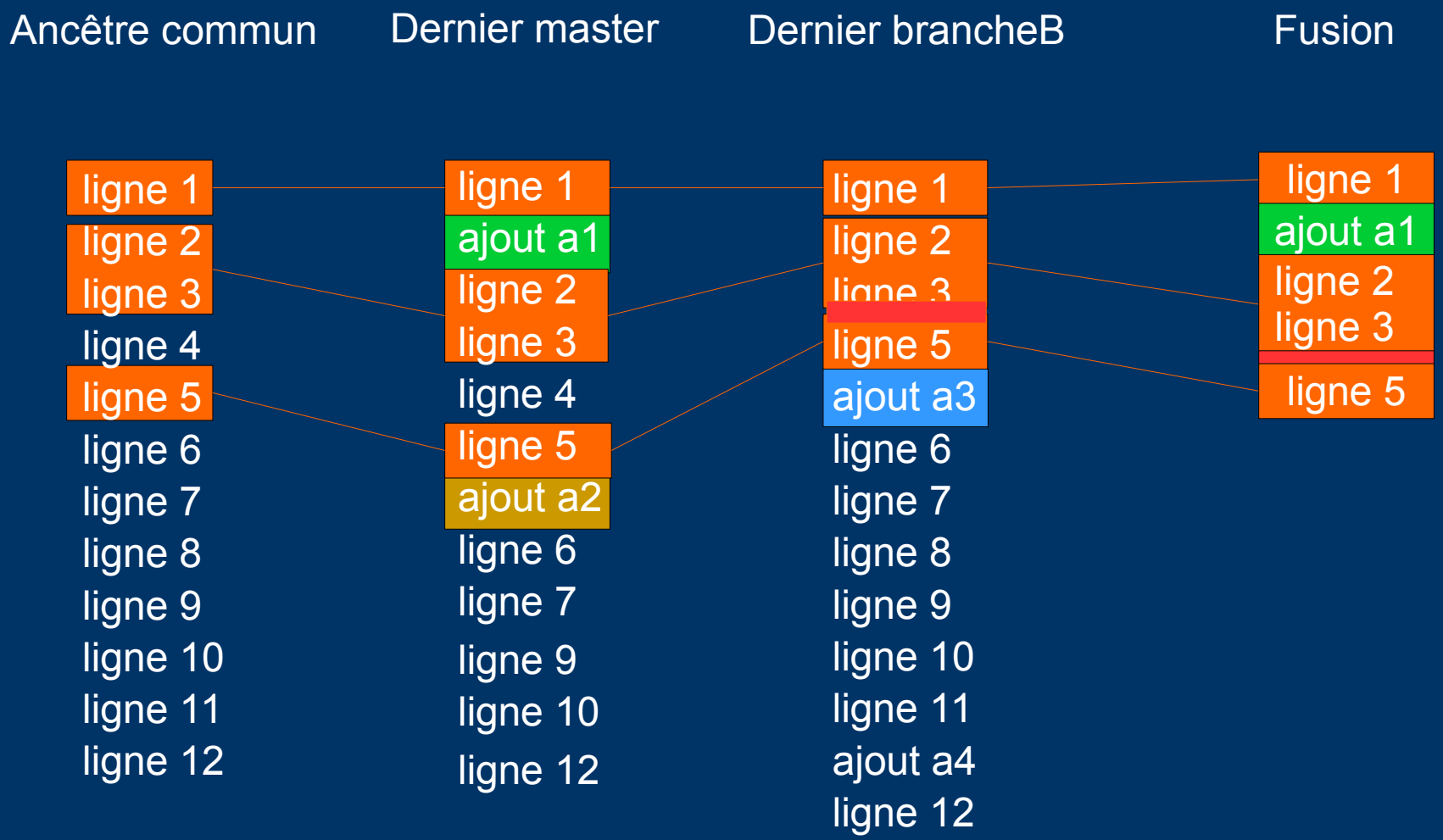
Principe de la fusion de branche
2. Recherche des différences dans le code



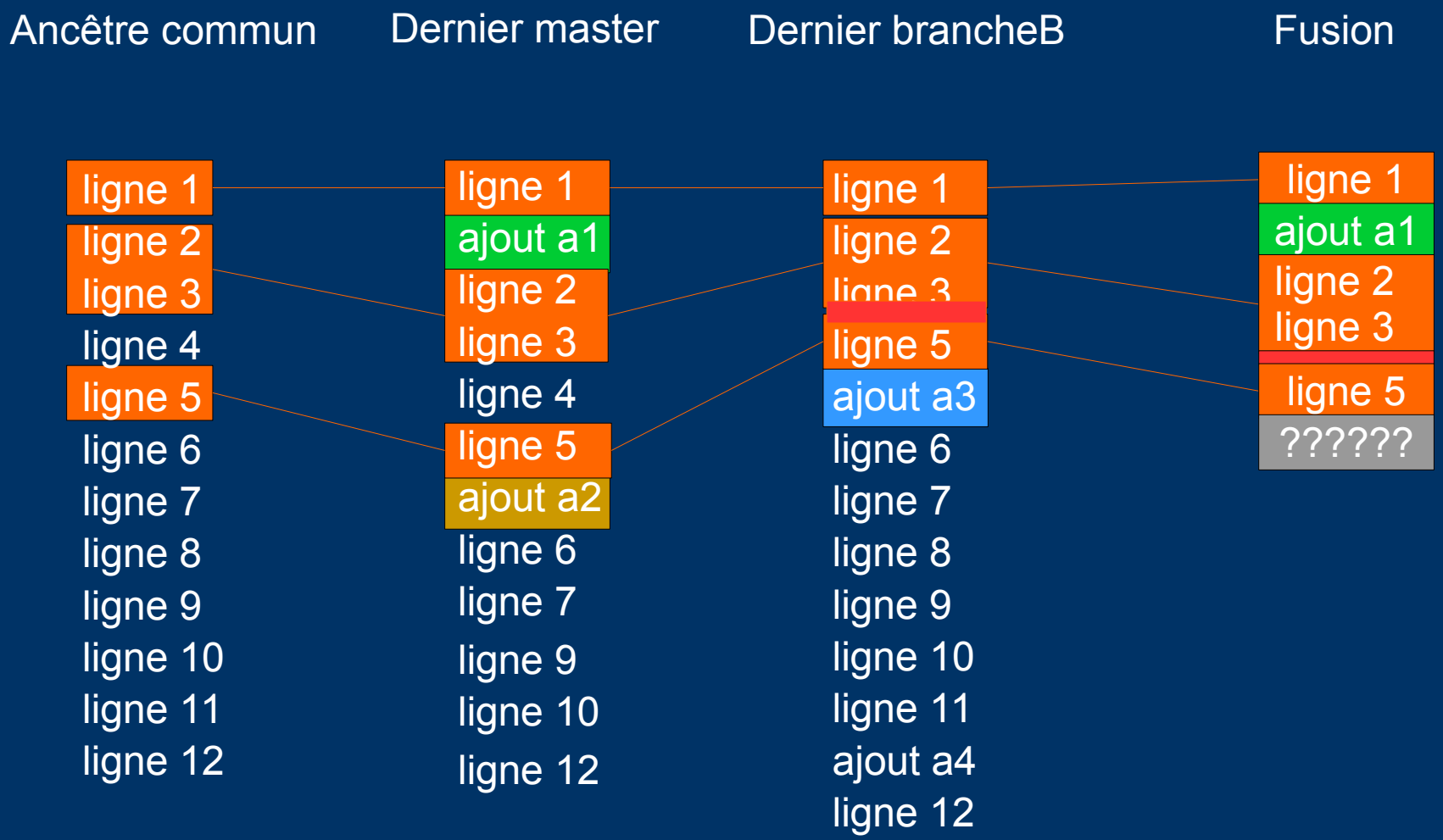
Principe de la fusion de branche
2. Recherche des différences dans le code



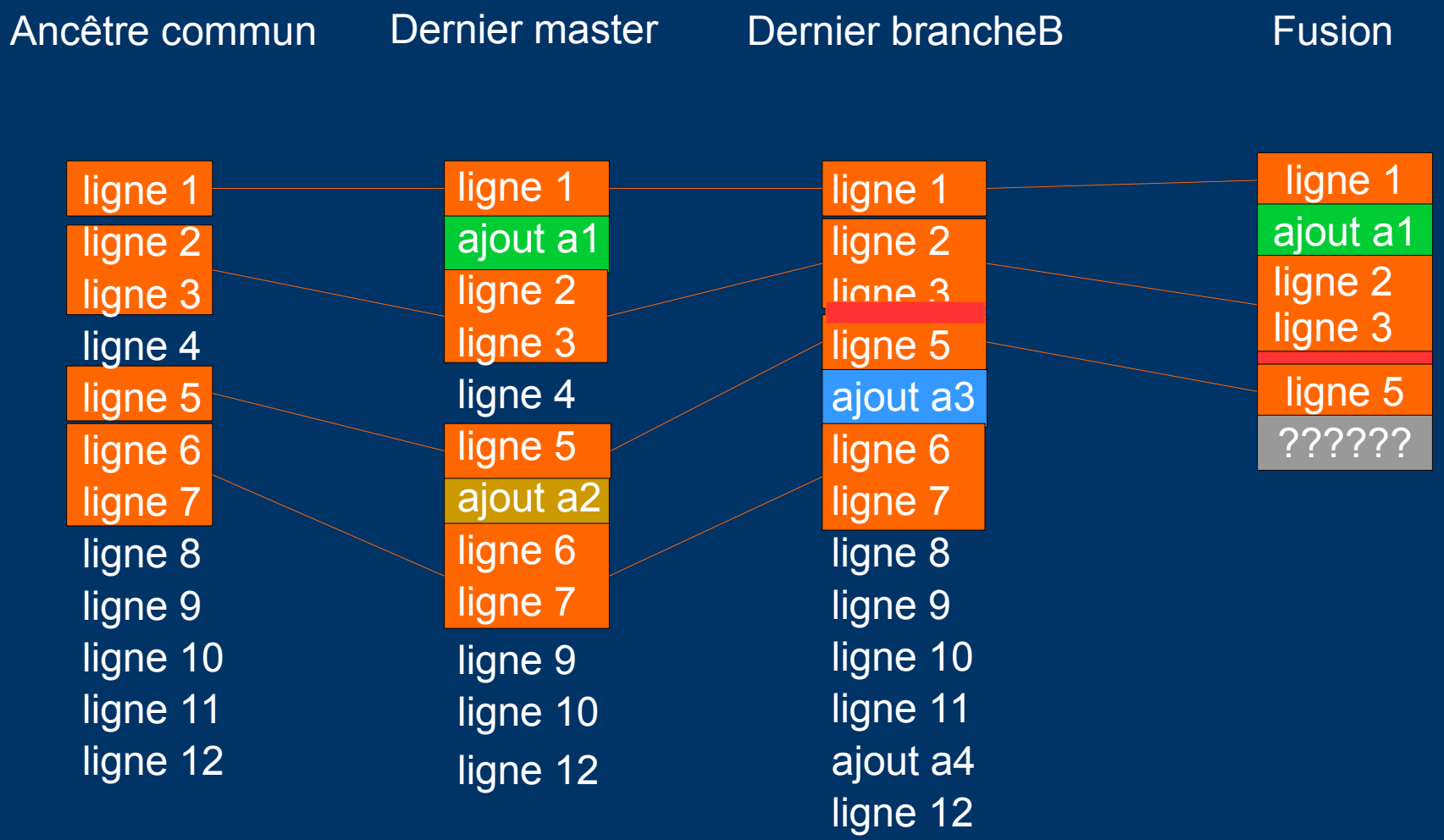
Principe de la fusion de branche
2. Recherche des différences dans le code



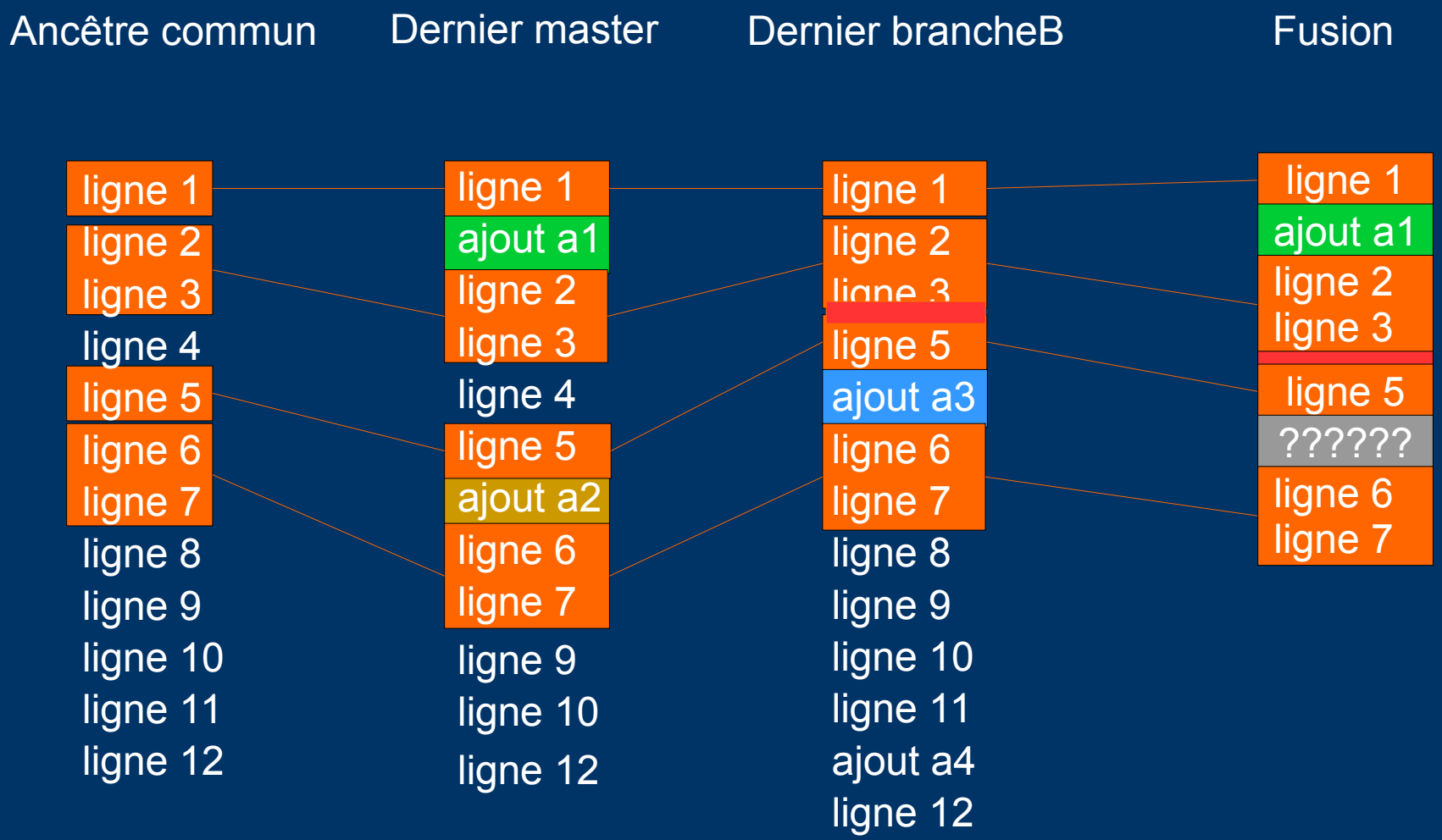
Principe de la fusion de branche
2. Recherche des différences dans le code



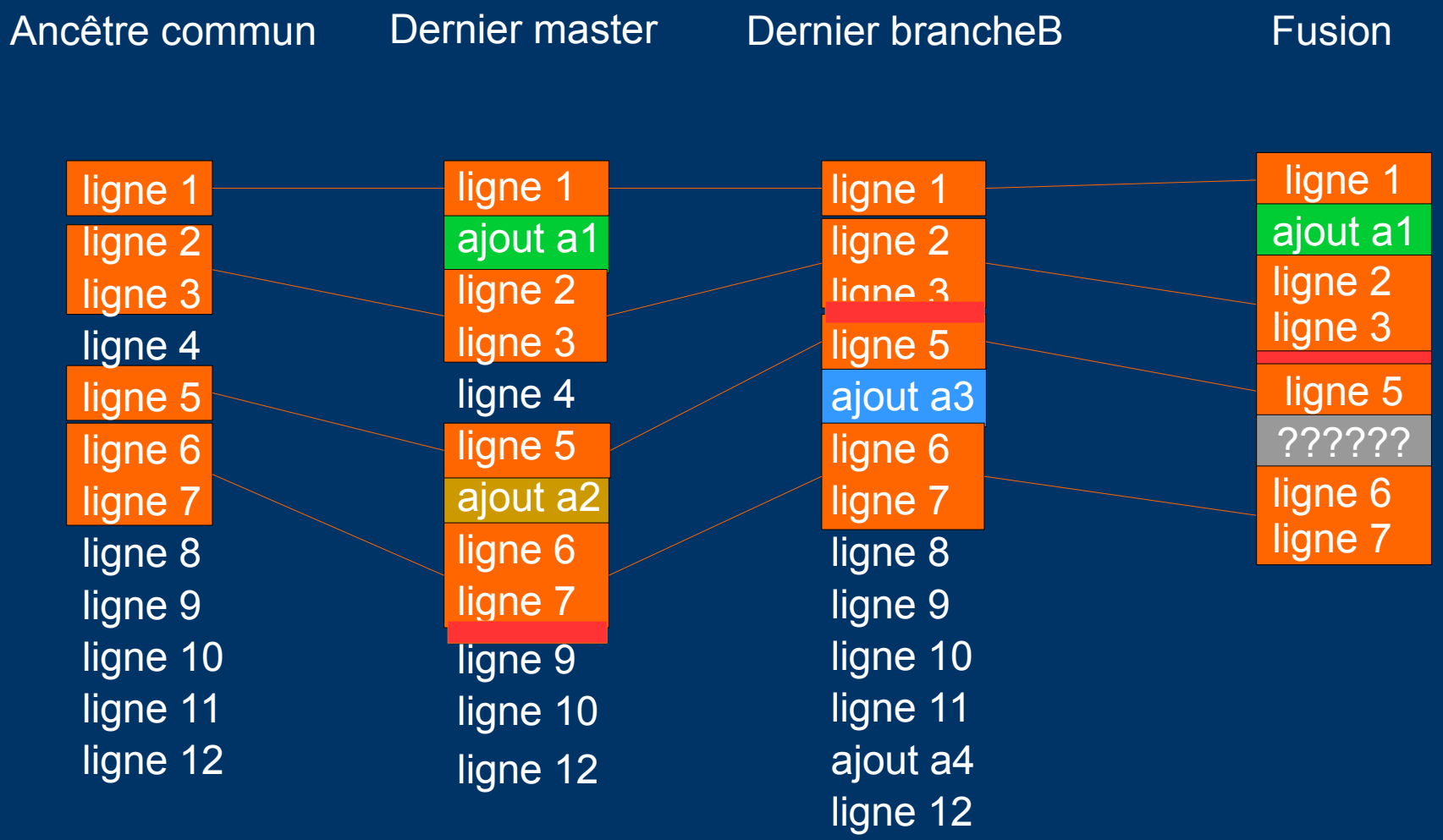
Principe de la fusion de branche
2. Recherche des différences dans le code



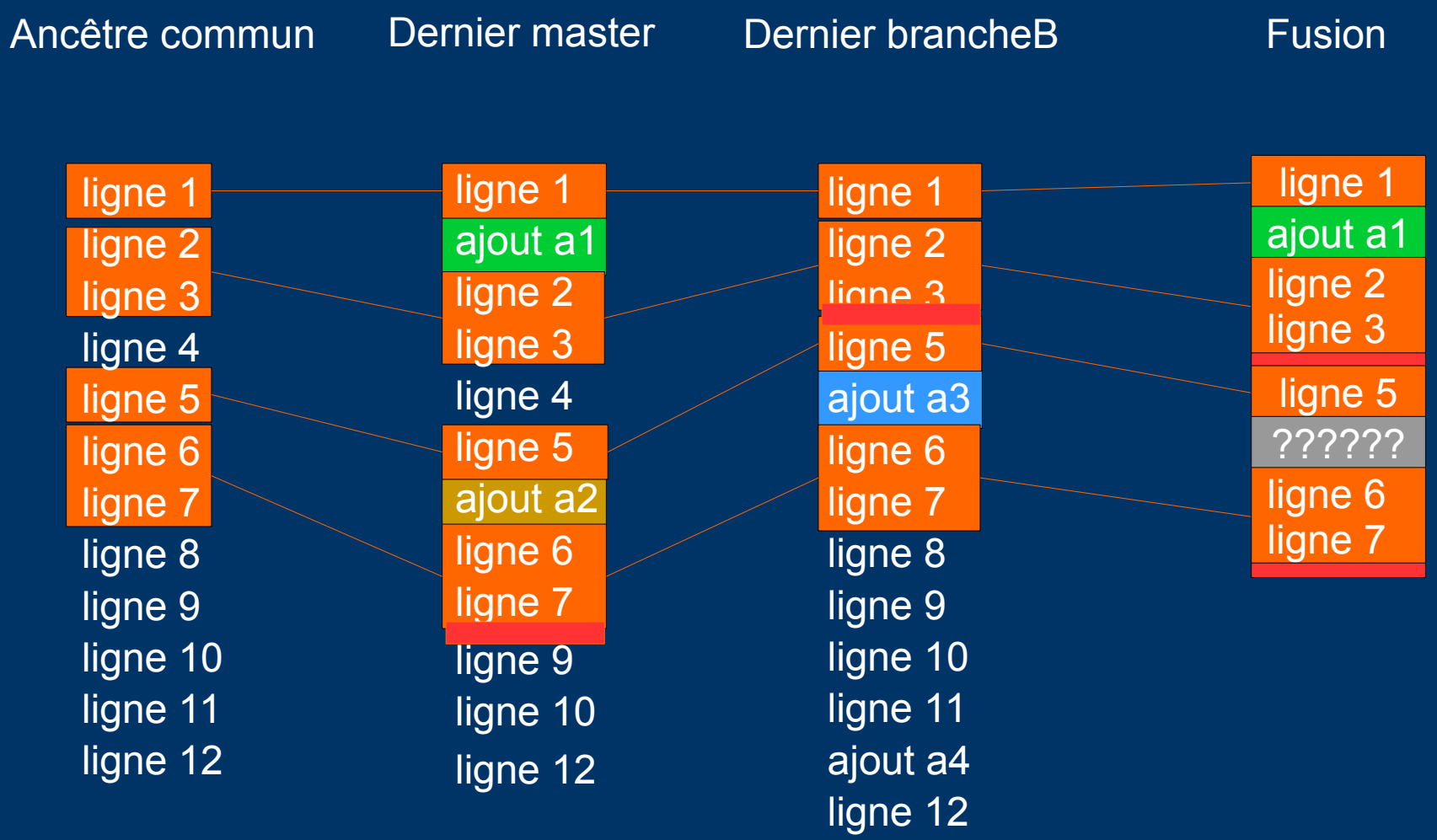
Principe de la fusion de branche
2. Recherche des différences dans le code



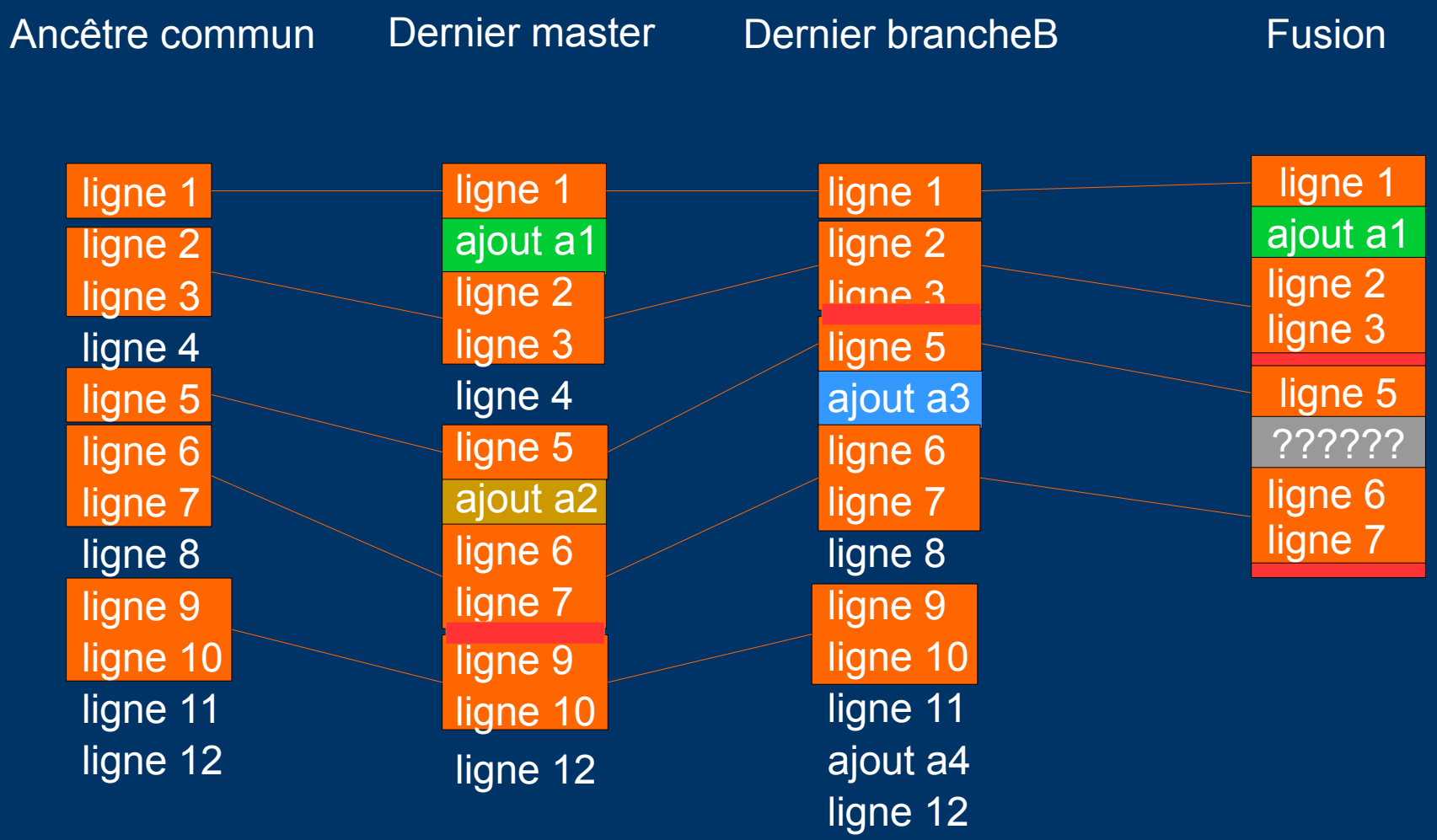
Principe de la fusion de branche
2. Recherche des différences dans le code



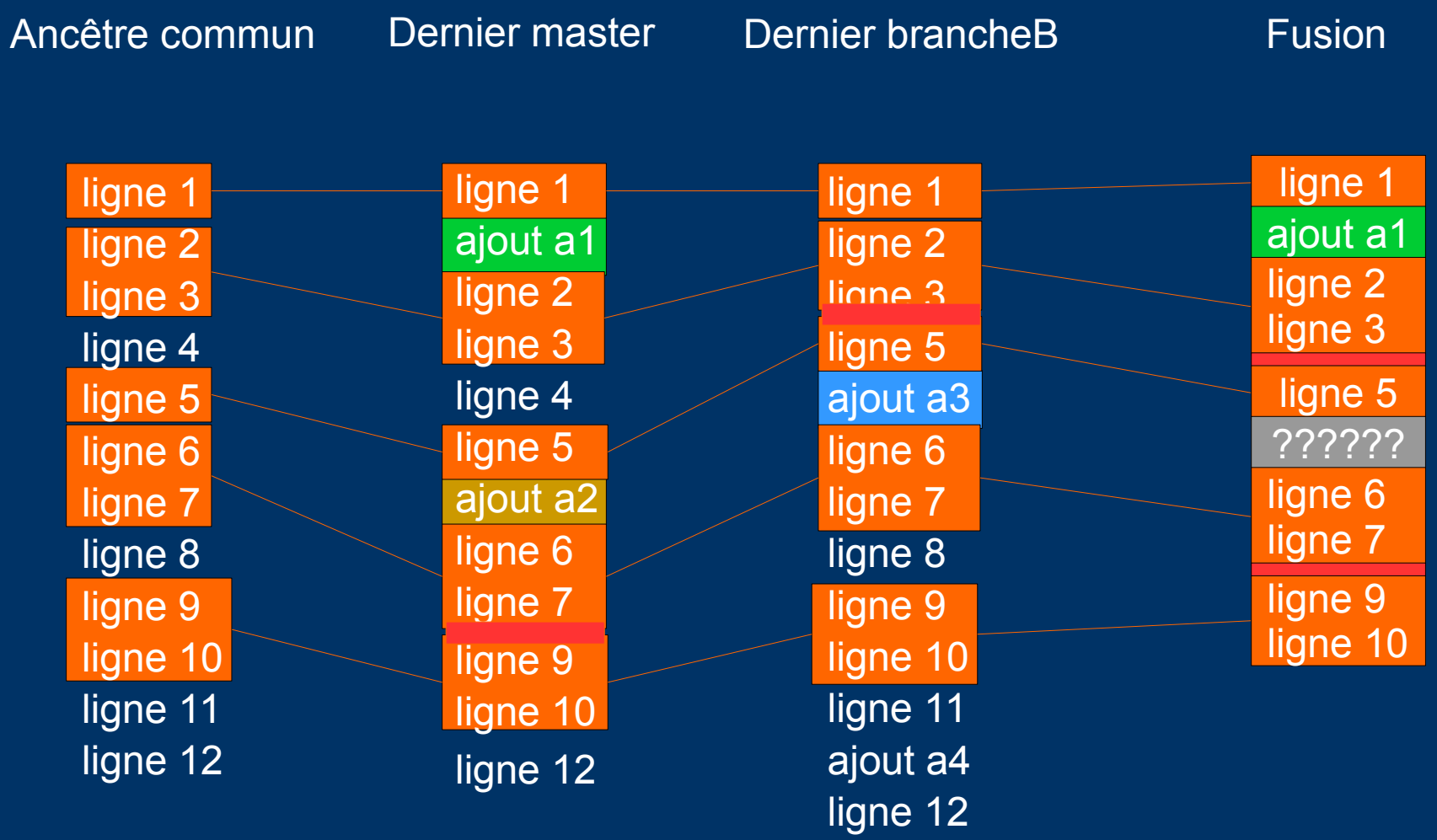
Principe de la fusion de branche
2. Recherche des différences dans le code



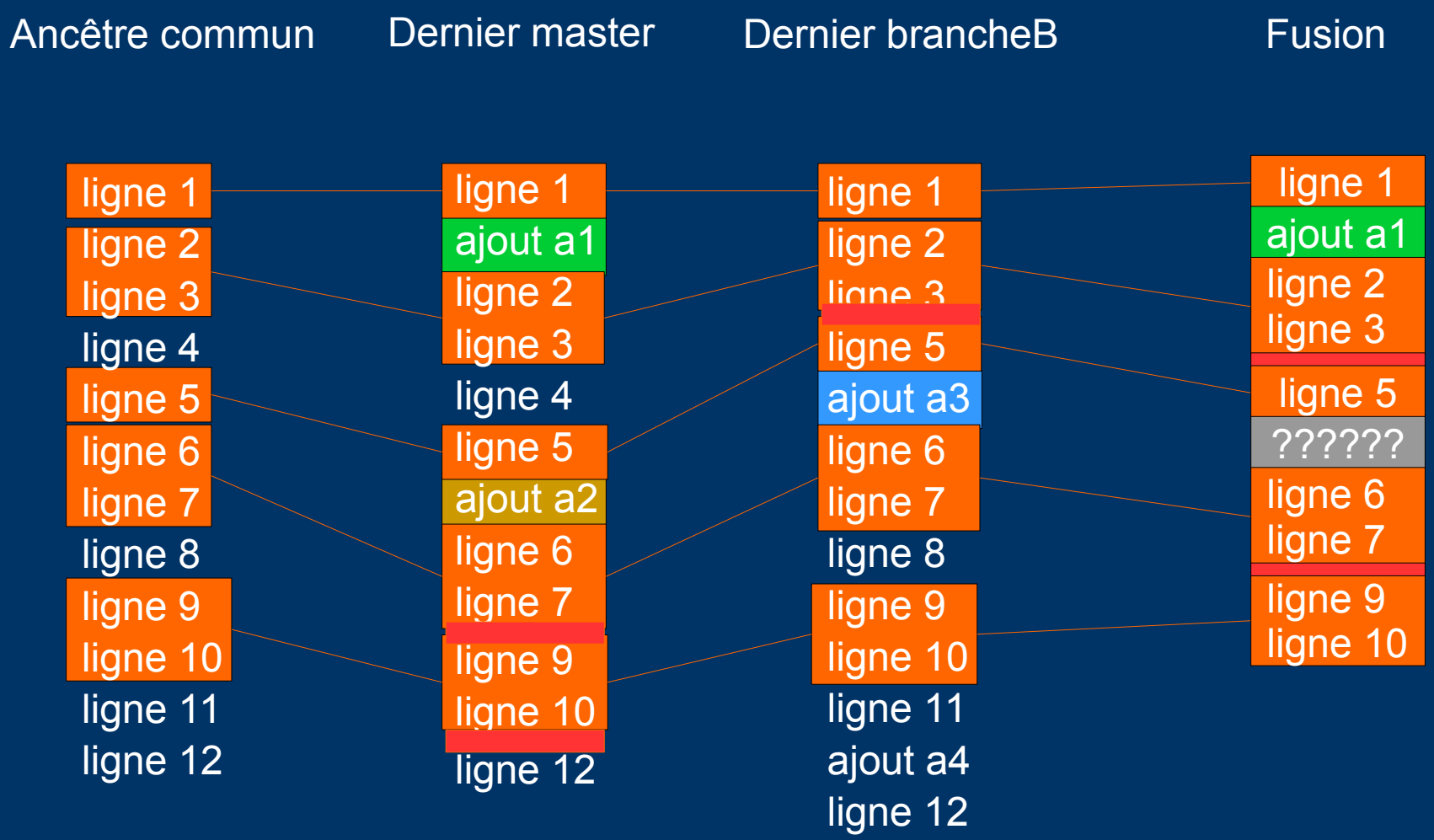
Principe de la fusion de branche
2. Recherche des différences dans le code



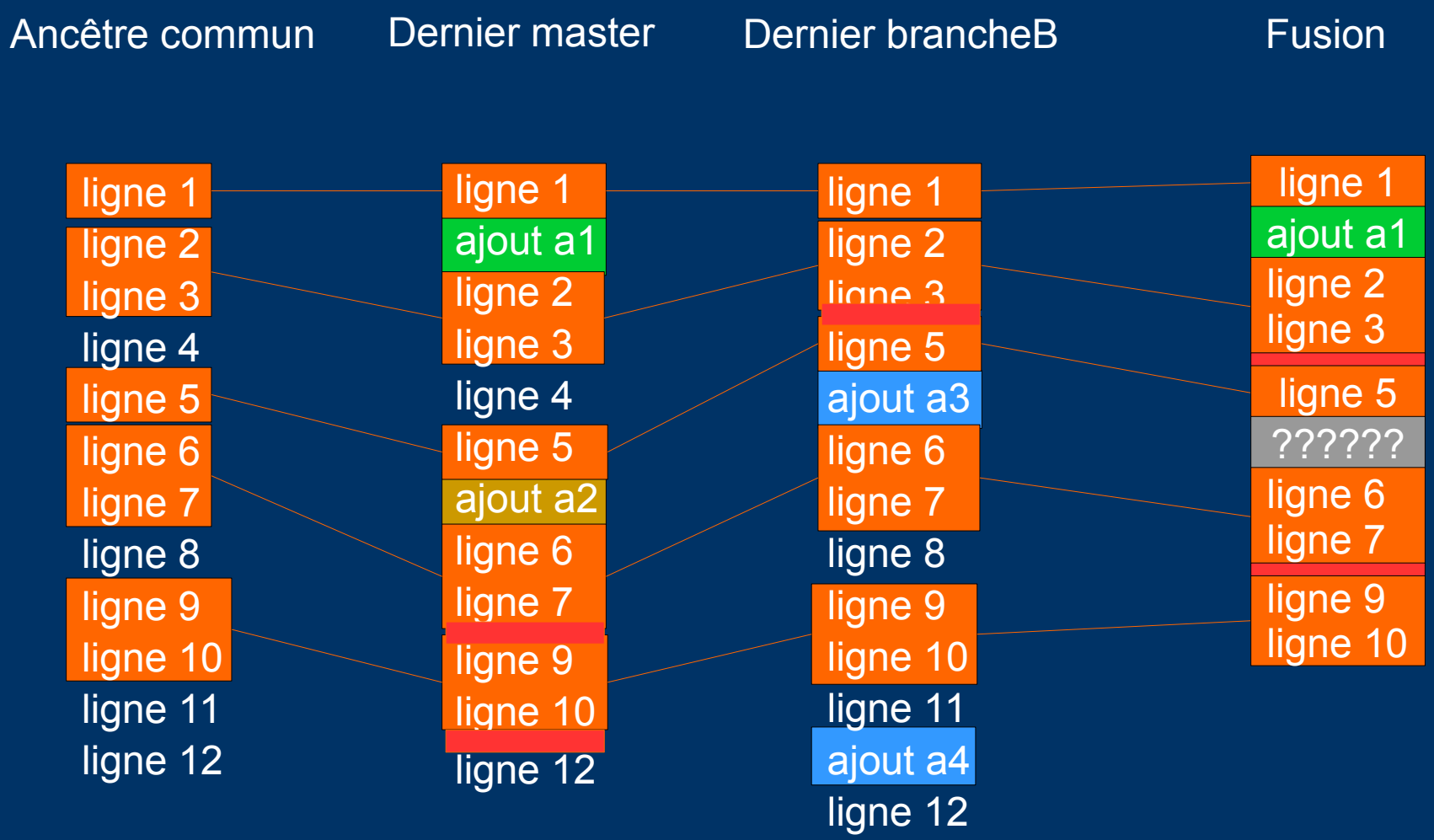
Principe de la fusion de branche
2. Recherche des différences dans le code



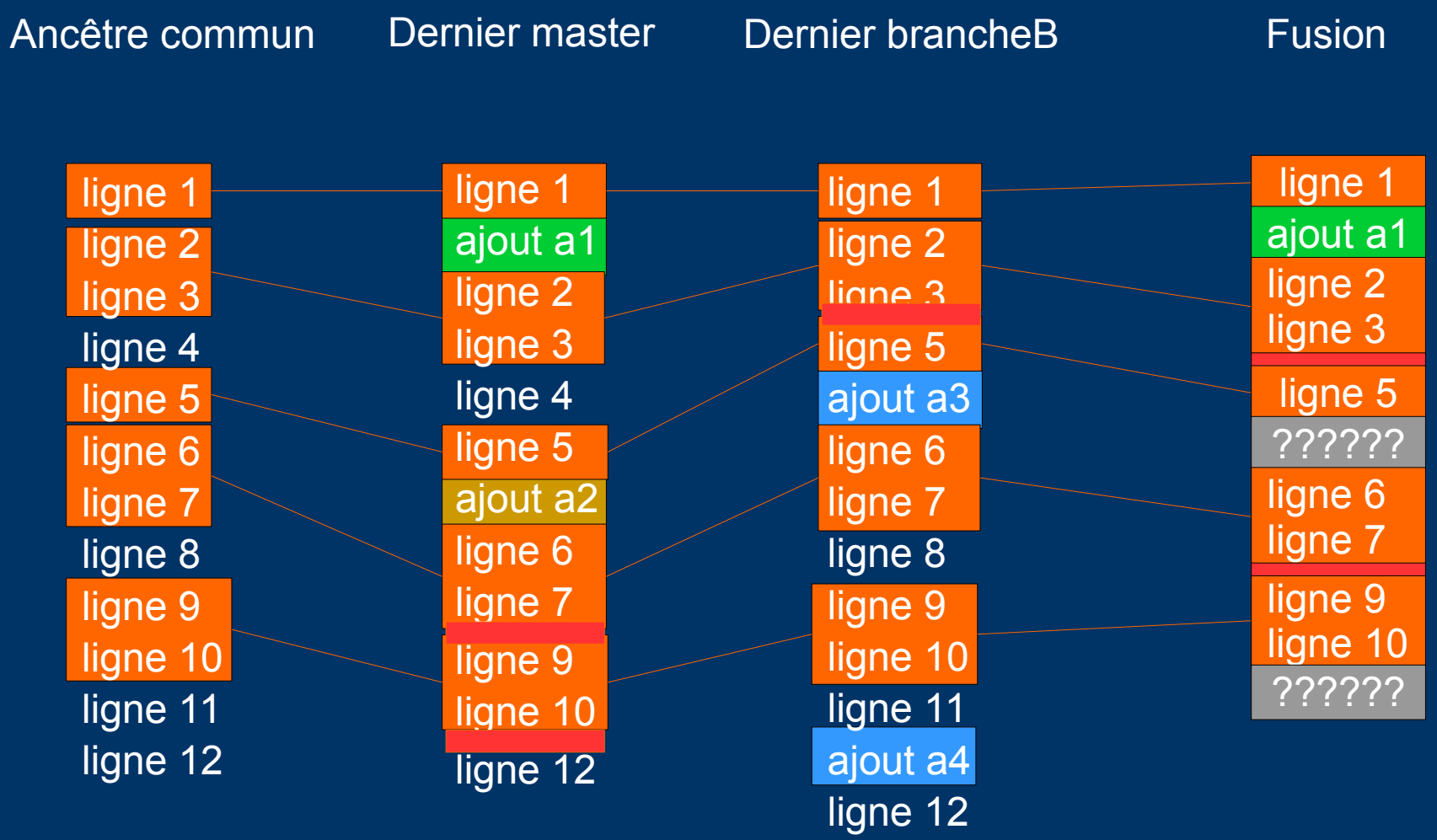
Principe de la fusion de branche
2. Recherche des différences dans le code



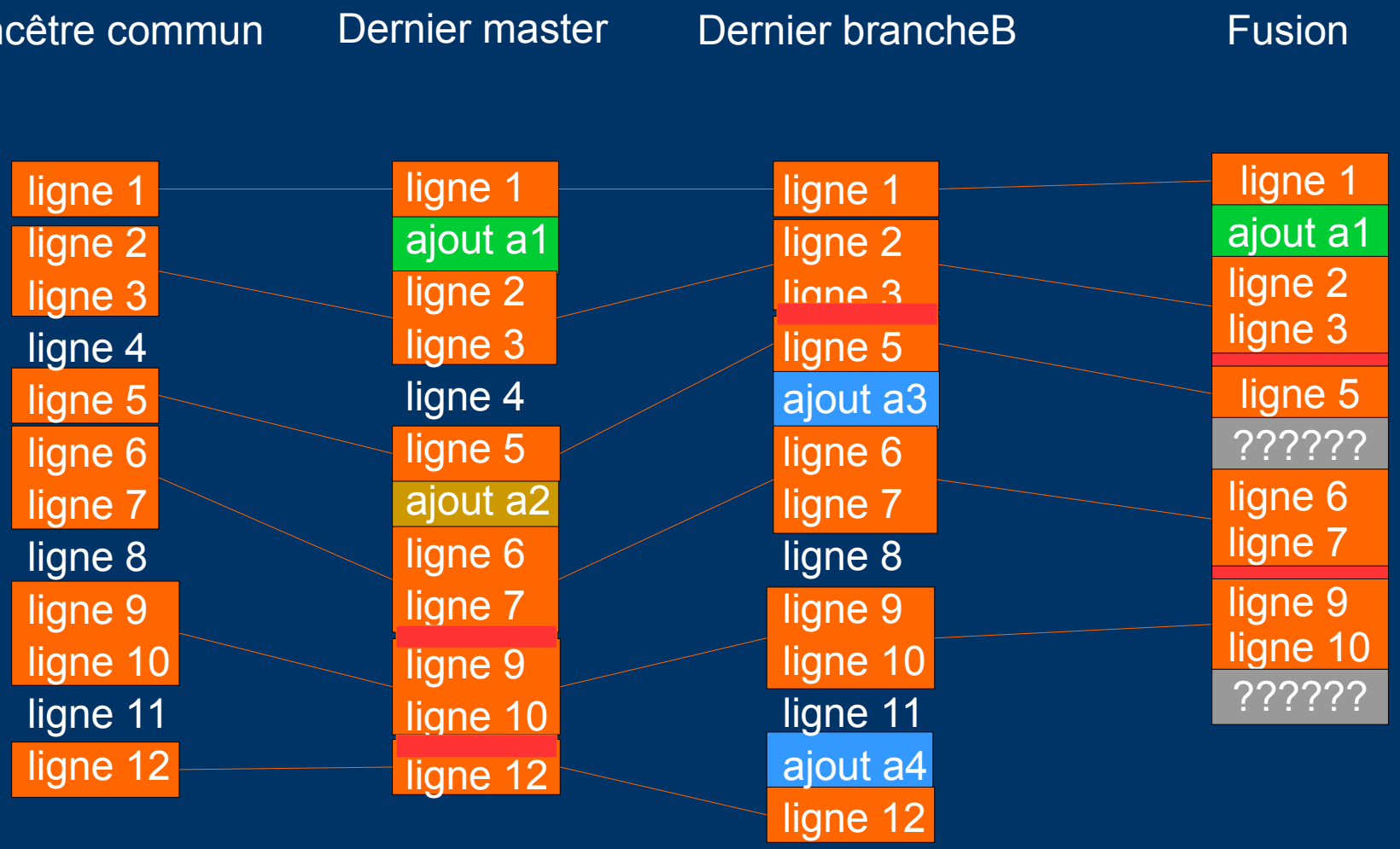
Principe de la fusion de branche
2. Recherche des différences dans le code



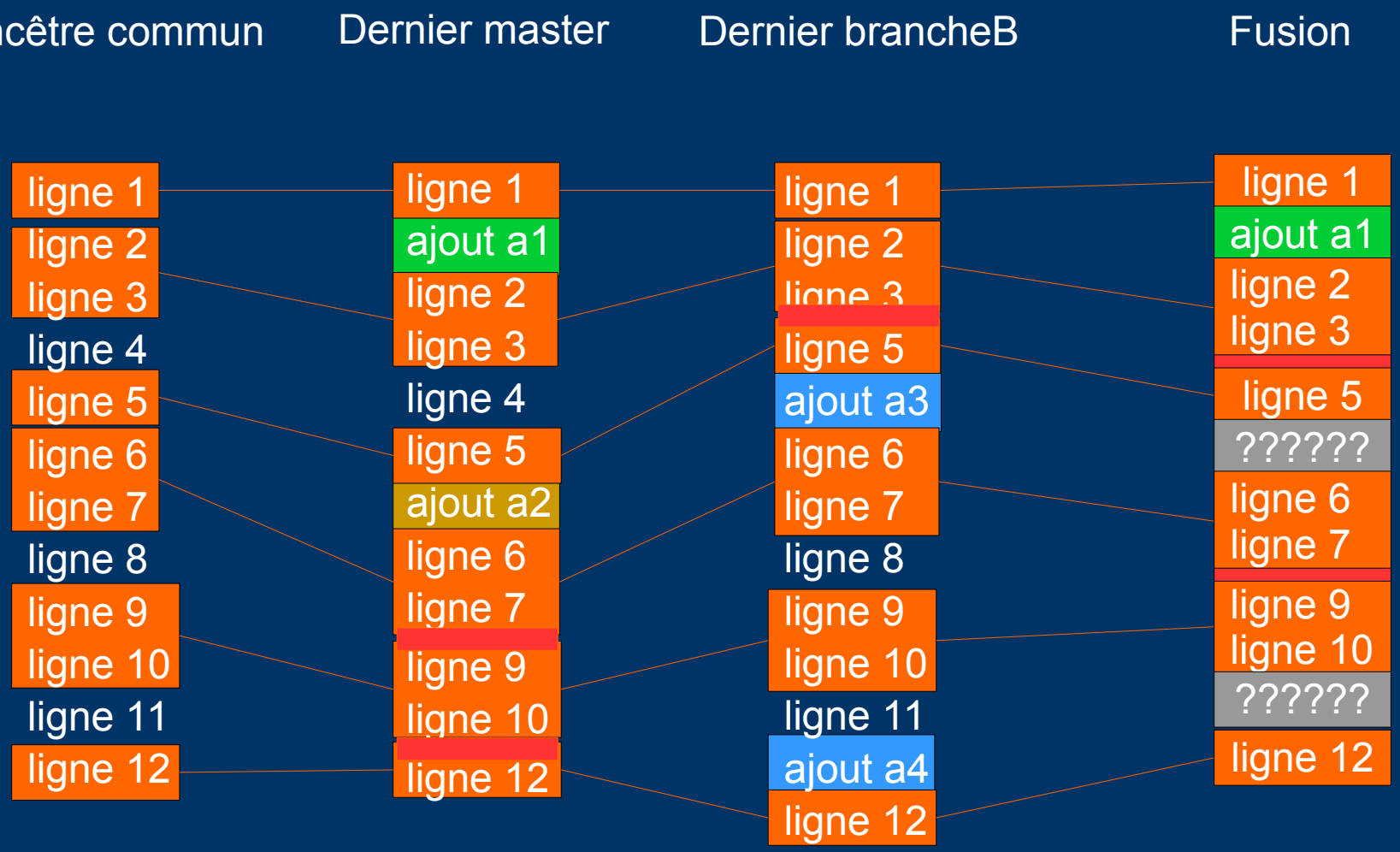
Principe de la fusion de branche
2. Recherche des différences dans le code



Principe de la fusion de branche
2. Recherche des différences dans le code



Principe de la fusion de branche
2. Recherche des différences dans le code



Principe de la fusion de branche
2. Recherche des différences dans le code

Principe de la fusion de branche
2. Recherche des différences dans le code

Fusion