Méthodes agiles

- processus et outils -

UNIVERSITÉ DE LORRAINE

Analyse et Conception de Logiciels

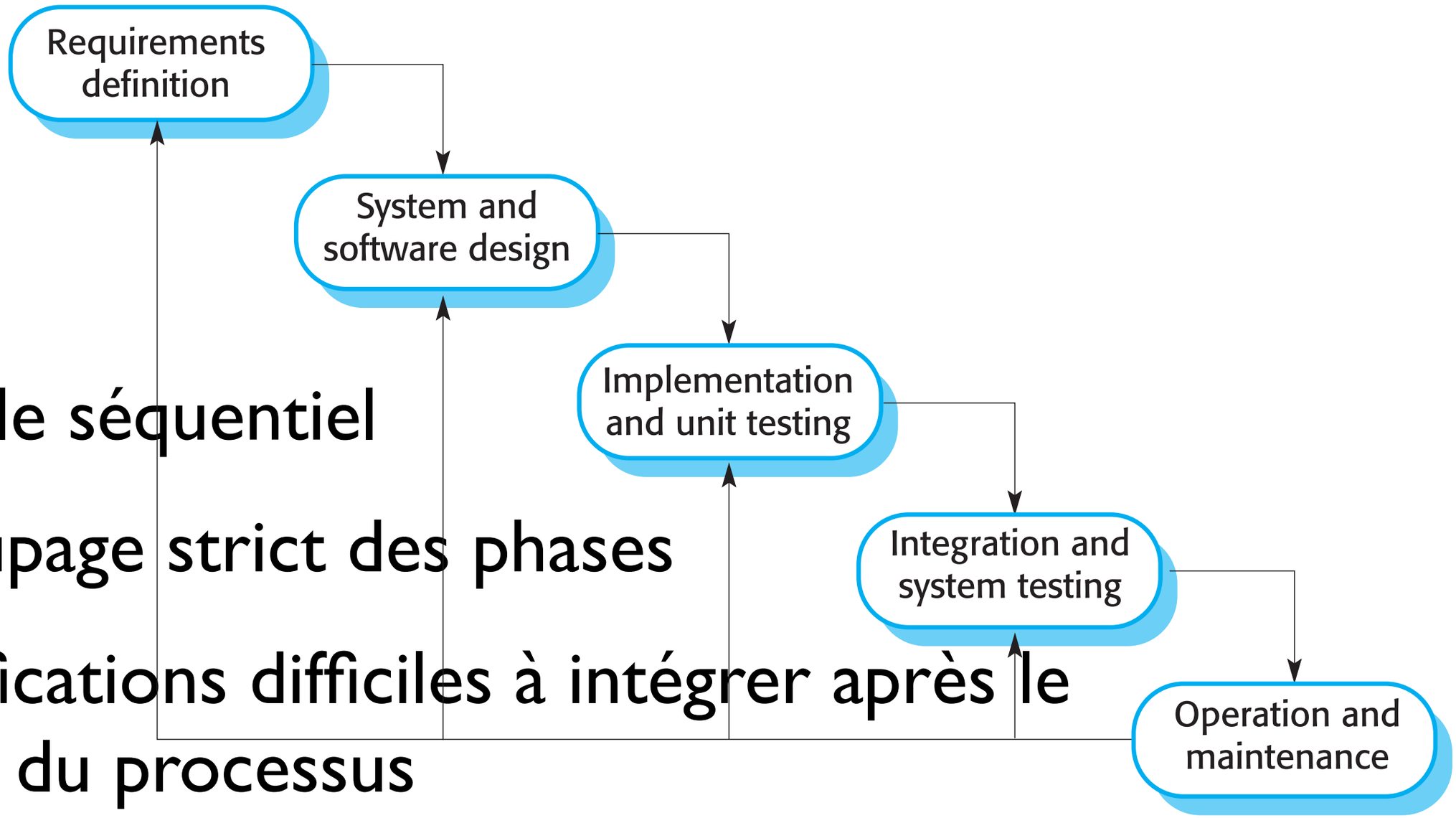
Horatiu CIRSTEA

Sources

- Cours Kären Fort, Jean-Christophe Bach
- Mountain Goat Software - www.mountaingoatsoftware.com
- cPrime - www.cprime.com
- The Apache Ant Project - ant.apache.org

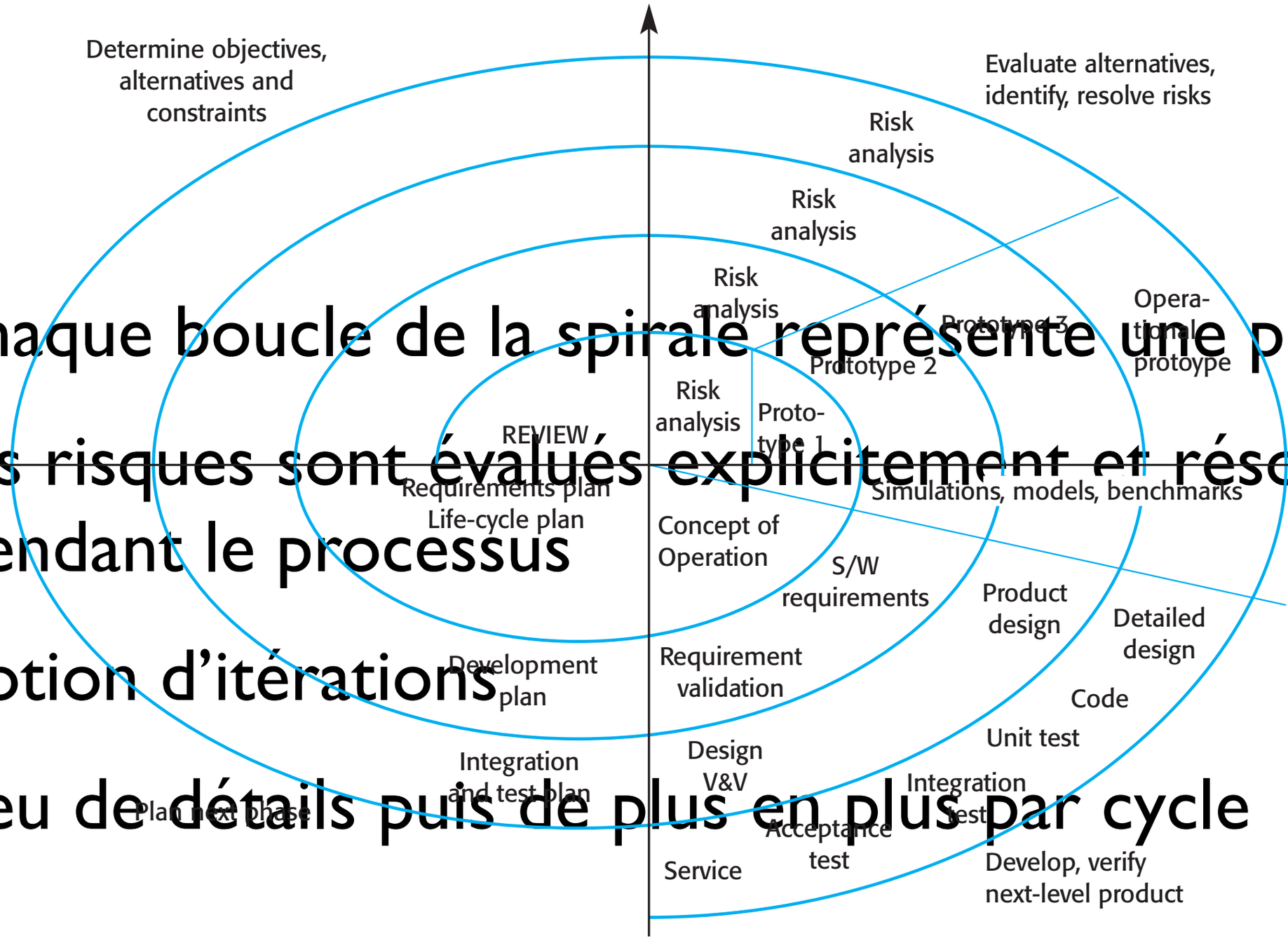
Modèles du processus de développement logiciel

Waterfall



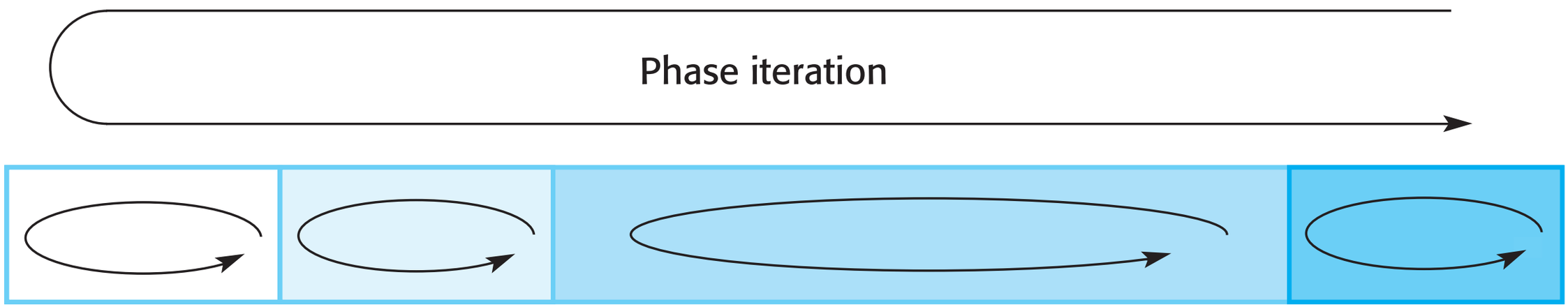
- modèle séquentiel
- découpage strict des phases
- modifications difficiles à intégrer après le début du processus

Modèle en spirale

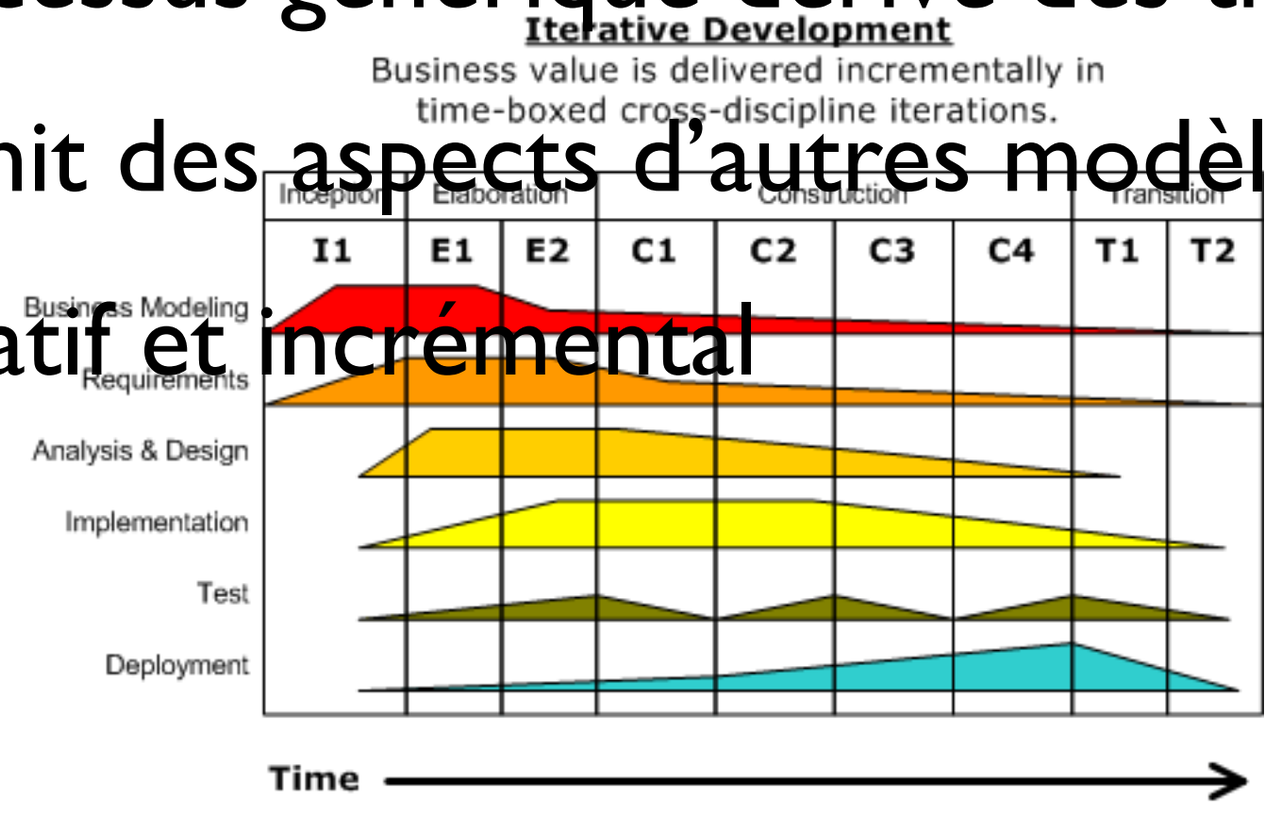


- chaque boucle de la spirale représente une phase
- ~~les risques sont évalués explicitement et résolus~~ pendant le processus
- notion d'itérations
- peu de détails puis de plus en plus par cycle

Rational Unified Process



- processus générique dérivé des travaux sur UML
- réunit des aspects d'autres modèles
- itératif et incrémental



wikipedia.org

Méthodes agiles

- RUP, Scrum, Crystal Clear, Extreme Programming, Adaptive Software Development, Feature Driven Development, Dynamic Systems Development Method,...

  ➢ Agile Manifesto - agilemanifesto.org

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Manifesto for Agile Software Development

Nous découvrons comment mieux développer des logiciels par la pratique et en aidant les autres à le faire. Ces expériences nous ont amenés à valoriser:

- Les individus et leurs interactions plus que les processus et les outils
- Des logiciels opérationnels plus qu'une documentation exhaustive
- La collaboration avec les clients plus que la négociation contractuelle
- L'adaptation au changement plus que le suivi d'un plan

Nous reconnaissons la valeur des seconds éléments, mais privilégions les premiers.

Valeurs

- L'équipe
  - les individus (leur créativité, leur motivation) sont plus importants que les méthodes formalisées ou les outils.
- L'application
  - un code bien commenté qui marche est préférable à une documentation technique abondante. Mais la maîtrise de l'application doit être partagée (transfert des compétences).
- La collaboration
  - doit être étroite avec le client qui donne régulièrement ses retours d'expérience.
- L'acceptation du changement
  - les premières *release* provoquent inévitablement des changements. La planification du projet doit être très flexible.

Les méthodes agiles

Fonctionnent seulement si l'équipe est motivée pour l'adopter

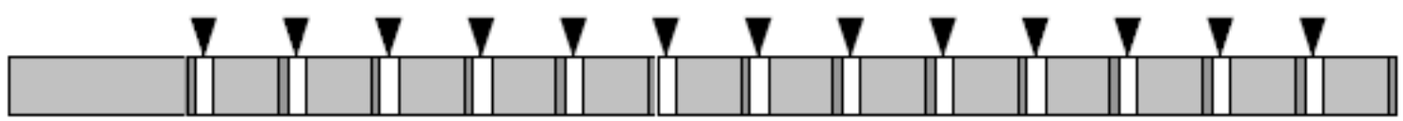
eXtreme Programming

- Modélisation optionnelle, informelle, sommaire et itérative
- Processus itératif, rythmé
- Développement dirigé par les tests (*test-driven development*)
- Programmation par binômes (*peer programming*), revue de code immédiate
- Rotation des rôles

XP - processus

- Cycle de décision habituel

- Cycle de décision XP



XP - processus

- Le client assiste l'avancement du projet et utilise le logiciel
- Mise en production rapide, puis régulière
- Fonctionnalités développées dans l'ordre choisi par le client (et non en fonction des contraintes techniques)

XP - processus

- Une itération =
  - Planification des fonctionnalités à implémenter (client + équipe)
  - Organisation de l'équipe en fonction de l'objectif
- Stand-up meeting quotidien
- **Pas d'heures supplémentaires deux semaines de suite :** un problème qui dure est un problème de fond à résoudre autrement qu'en travaillant davantage

XP - tests

- Font office de spécification (informelle)
  - *fonctionnelle* : écrits par le client et le testeur, contraignent le client à savoir précisément son besoin
  - *technique* : écrits par le développeur, limitent les anomalies, une ligne de code = une ligne de test
- Tests écrits **avant** le code (*test-driven development*)
- Automatisation extrême des tests nécessaire

XP - rôles

- *Fixes*
  - expert XP / coach
  - manager
  - client
  - consultant technique / tracker
- *Rotatifs*
  - testeur
  - développeur
  - contrôleur

Une personne peut jouer plusieurs rôles à la fois.

XP - rôles

- *Développeur* = *artisan-créateur*
  - produit du code simple
  - rédige des tests unitaires
  - est en contact le plus possible avec le client
- *Client*
  - attribue des priorités aux tâches
  - rédige les tests de recette
  - participe à la planification
- *Testeur*
  - possède des compétences techniques et métier
  - est proche du client
  - automatise les tests de recette

XP - rôles

- *Expert XP / coach*
  - organise le processus XP
  - anime la planification et les *stand-up meetings*
  - gère les dérives
- *Tracker*
  - assure le suivi des tâches en cours
  - détecte les difficultés avant qu'il ne soit trop tard
  - a des qualités relationnelles importantes
- *Manager*
  - gère l'environnement matériel et l'équipe XP
  - contrôle le processus XP
  - ne fait pas forcément partie de l'équipe

XP - points clé

- *Hommes* : des compétences techniques et des qualités humaines
- *Communication* : entre le développeur et le client
- *Simplicité* : dans le processus, dans la communication, dans la conception, dans le codage

  - pas de conception a priori
  - tests de non-regression primordiaux, automatisation des tests, qualité extrême du code (*coding style*, lisibilité), etc.

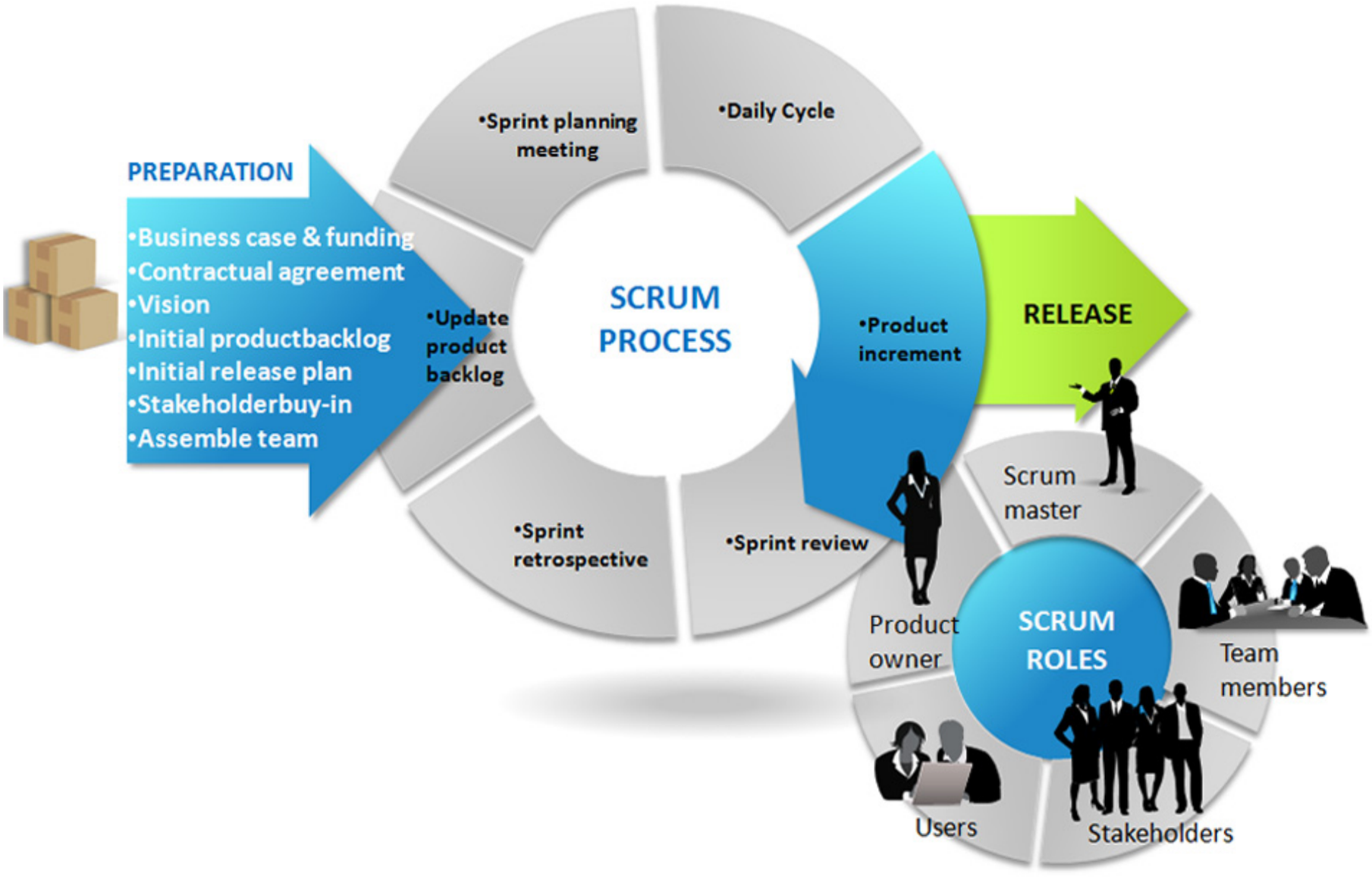
Scrum

- processus agile qui permet de se focaliser sur la livraison *rapide* de produits de *qualité*
- le *business* fixe les priorités et l'équipe s'organise pour livrer les *fonctionnalités de haute priorité*
- permet l'*inspection rapide et répétée* (chaque 2 à 4 semaines) de *code fonctionnel*
- un logiciel fonctionnel est *disponible régulièrement* (chaque 2 à 4 semaines) pour *tout le monde* (développeurs, clients, utilisateurs)
  - on peut décider la *livraison* ou l'*amélioration*

Scrum - Principes

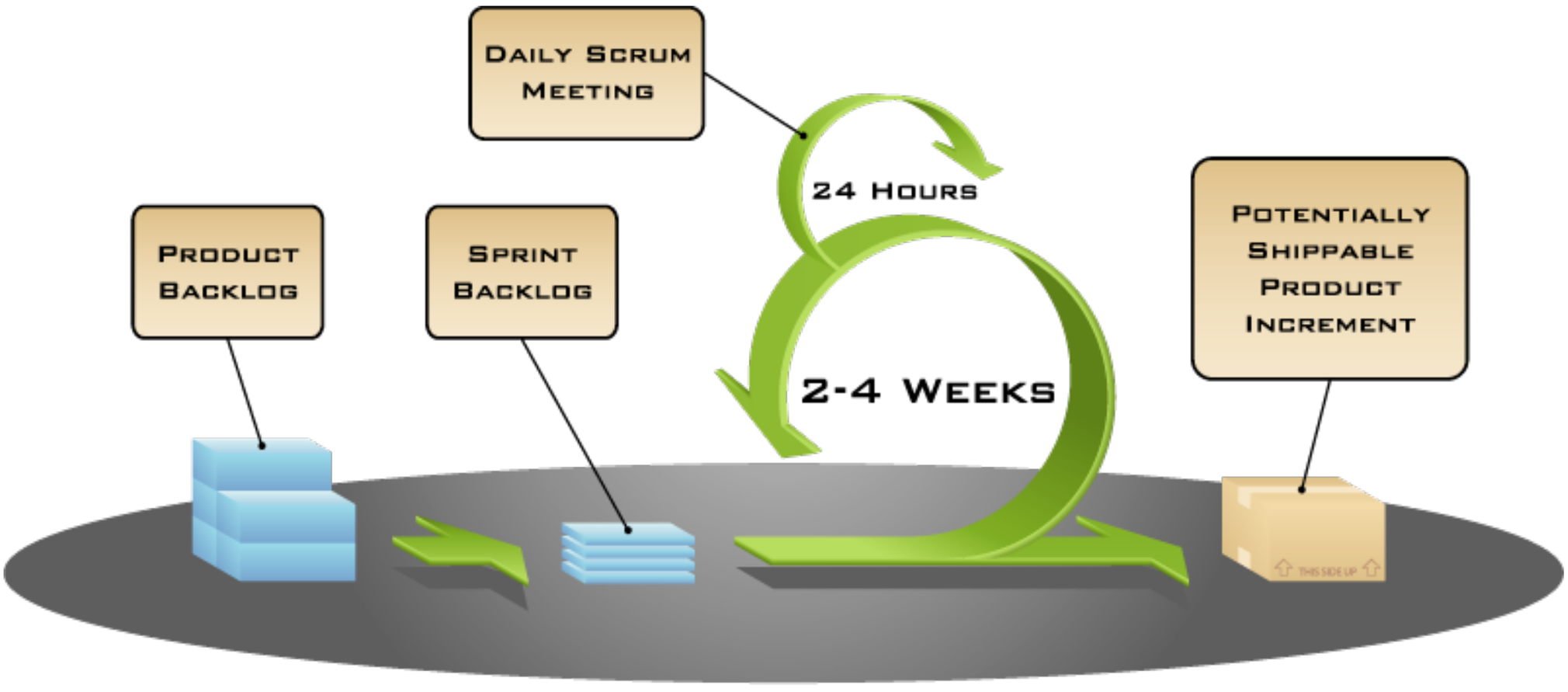
- Transparence : progression visible par tout le monde (développeurs, clients, utilisateurs)
- Inspection : opportunités fréquentes pour analyser l'état courant du travail
- Adaptation : corriger et adapter suite à l'inspection

Scrum - Processus



www.cPrime.com

Scrum - Processus



Scrum - Processus

- Les projets Scrum progressent par une série de *sprints*
  - équivalents aux itérations d'XP
  - durée d'un sprint est de 2 à 4 semaines
- Le produit (partiel) est conçu, codé et testé pendant le sprint

Scrum

Rôles

- Scrum master
- Product owner
- Equipe SCRUM

Artefacts

- Backlog produit
- Backlog de sprint
- Burndown chart

Cérémonies

- Sprint planning
- Sprint review
- Sprint retrospective
- Daily scrum meeting

Scrum - Roles

Product owner ➢ détient la vision du produit

- connaît les demandes et priorités des clients
- définit les fonctionnalités (le backlog produit)
- ajuste les fonctionnalités et priorités à chaque itération
- valide le bon fonctionnement du produit

Scrum - Roles

Scrum master ➢ Représente le management du projet

- facilite le déroulement du SCRUM
- élimine les perturbations/obstacles
- facilite une coopération poussée entre tous les rôles et fonctions

Scrum - Roles

Equipe SCRUM ➢ 5-10 personnes

- auto-organisée (choisit la façon de travailler)
- pas de rôles fixes définis
- tous les rôles (architecte, concepteur, développeur, testeur, etc.) sont représentés
- à plein temps sur le projet, de préférence
- la composition ne doit pas changer pendant un Sprint

Scrum - Artefacts

Backlog produit

- Liste de fonctionnalités à réaliser (user stories)
- Chaque item
  - une priorité assignée par le PO (réévalué au début de chaque sprint)
  - un titre compréhensible par tout le monde
  - une estimation en points
- Idéalement chaque item ajoute de la valeur pour les clients/utilisateurs du produit

Scrum - Backlog produit

Item	Estimate
Permettre à un utilisateur de s'enregistrer	3
Un utilisateur fait une réservation	5
Un utilisateur consulte ses réservations	1
Améliorer la journalisation	3
...	8

Scrum - Artefacts

Backlog sprint

- Une déclaration courte de l'*objectif* du sprint
- Sous-ensemble d'items du backlog produit
- Les développeurs choisissent le travail à faire (jamais attribué par un autre)
- N'importe qui peut ajouter, supprimer ou changer la liste des tâches
- Le travail du sprint émerge progressivement
- Si un travail n'est pas clair, définir une tâche avec plus de temps et la décomposer après

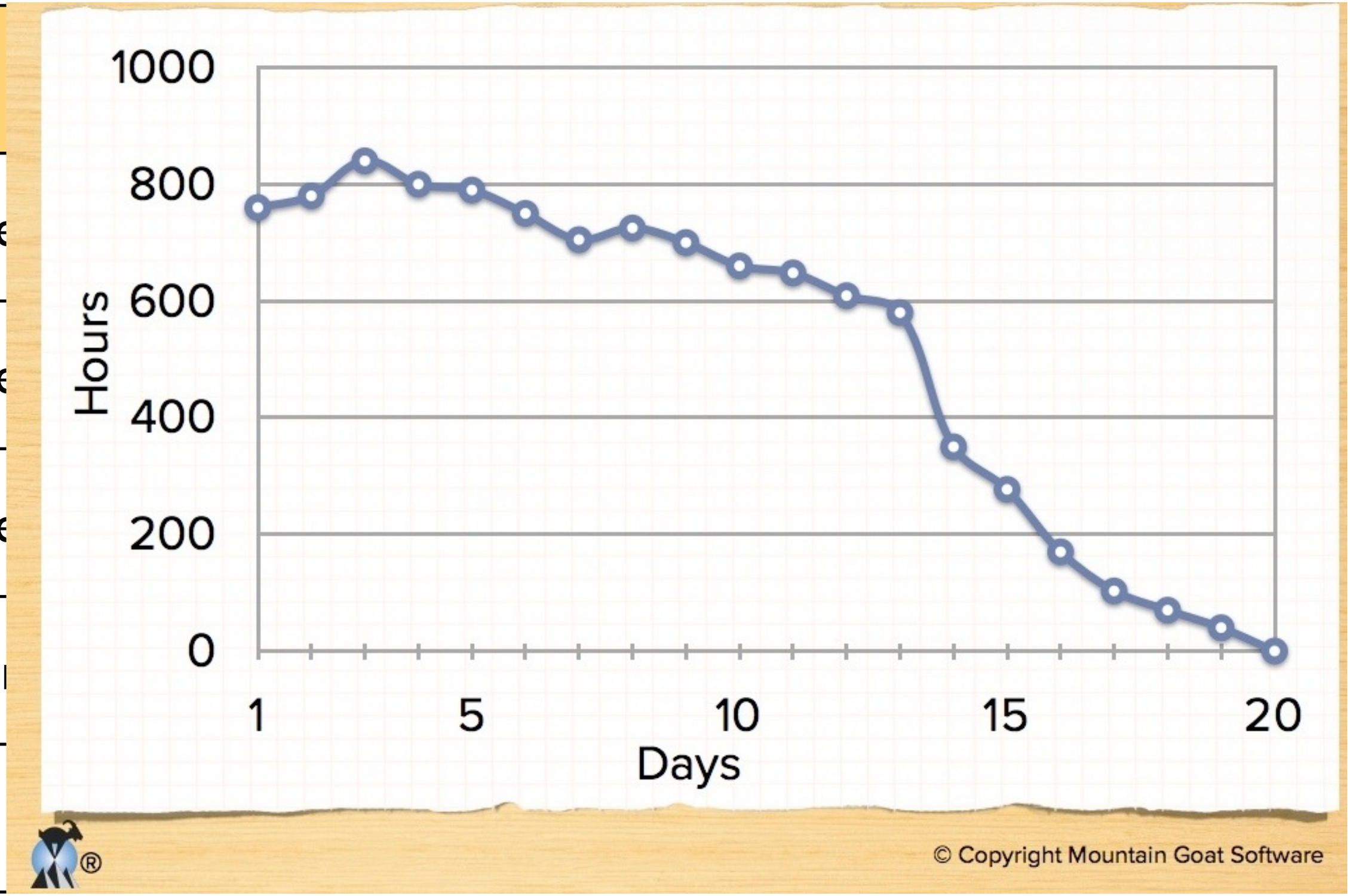
Scrum - Artefacts

Burndown chart

- Indique le travail qui reste à faire
- L'estimation du travail qui reste à faire est actualisée tous les jours

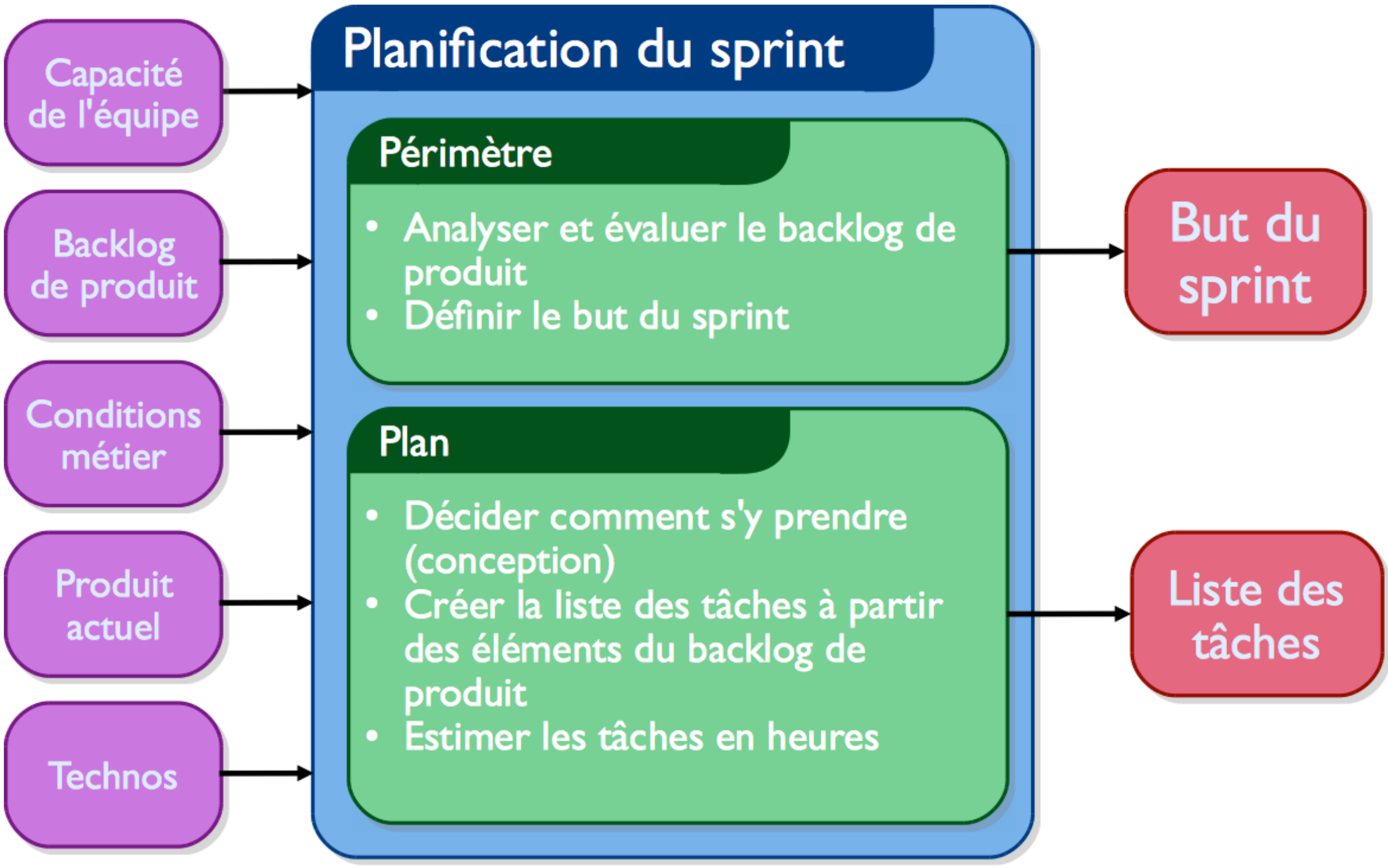
Scrum - Backlog sprint

Codage
Codage
Ajouter
..



Scrum - Cérémonies

Sprint planning

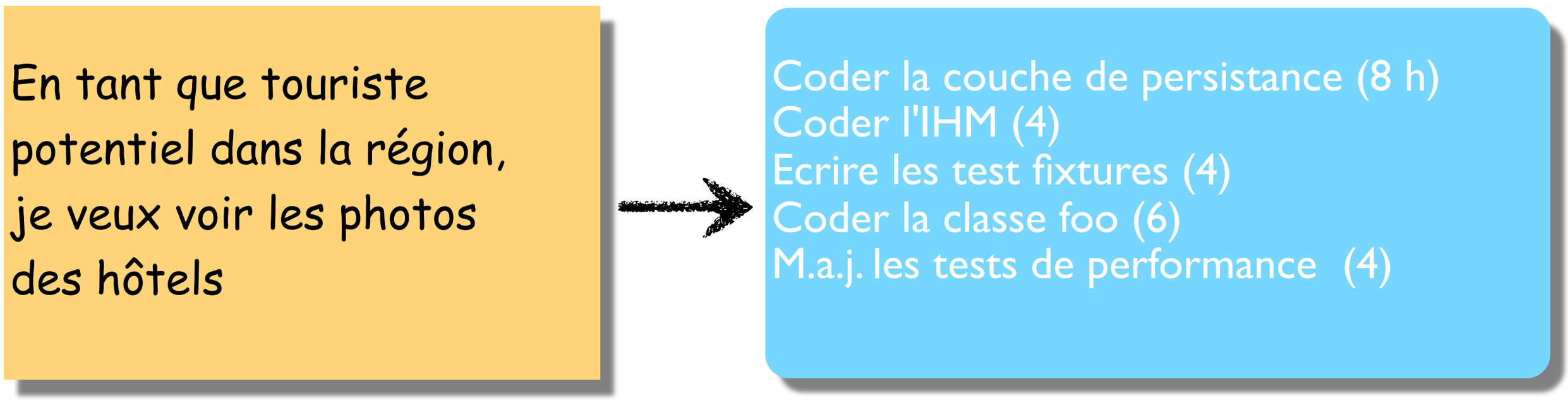


Source: Mountain Goat Software, LLC

Scrum - Cérémonies

Sprint planning

- L'équipe construit le backlog du sprint en fonction des capacités et de la vélocité
- Les tâches sont identifiées et estimées (1-16 heures)
- La conception de haut niveau est abordée



Source exemple: Mountain Goat Software, LLC

Scrum - Artefacts

Daily scrum meeting

- Tous les jours
- 15 minutes
- Debout
- Pas fait pour résoudre les problèmes
- Tout le monde est invité
- Seuls les membres de l'équipe peuvent parler
- Permet d'éviter l'organisation d'autres réunions

Scrum - Artefacts

Daily scrum meeting

- Qu'as-tu fait hier ?
- Que vas-tu faire aujourd'hui ?
- Y a t-il un obstacle qui te freine ?

Scrum - Artefacts

Sprint review

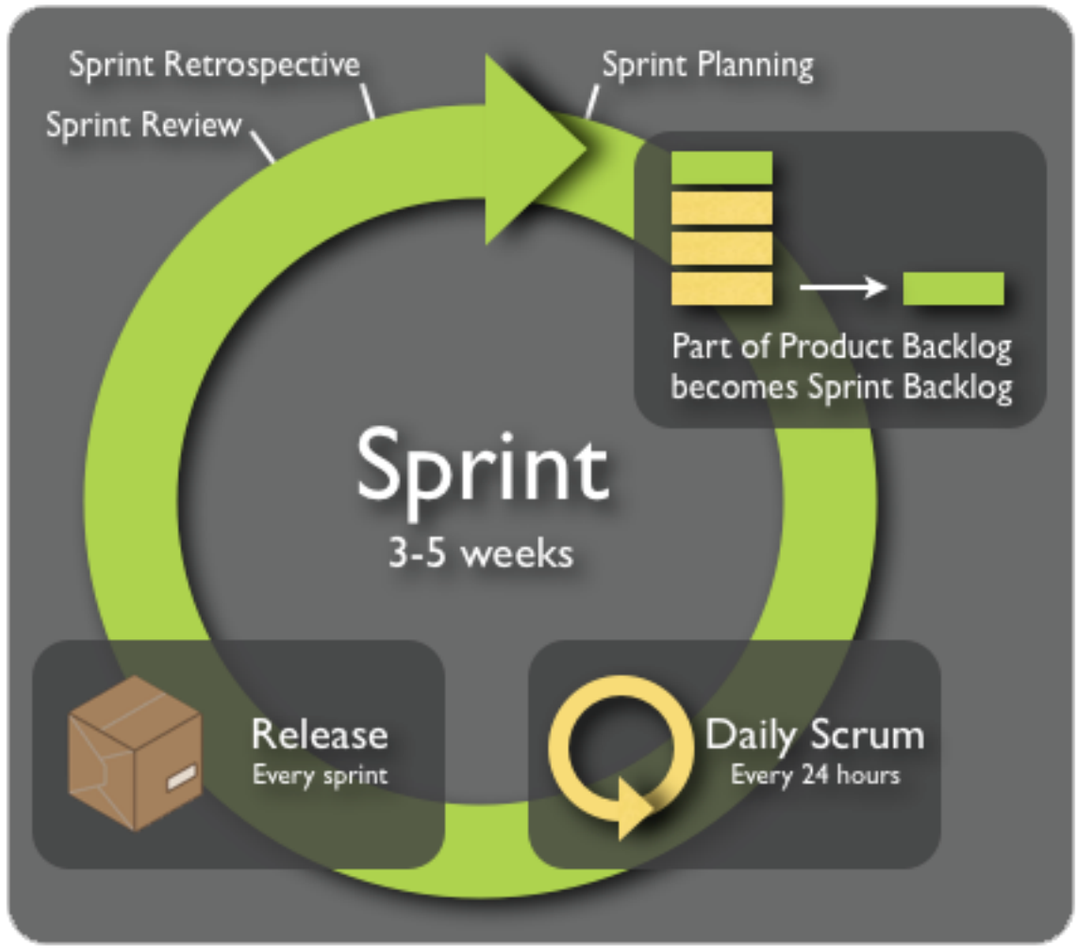
- Réunion informelle de maximum 4 heures (préparation < 2h)
- *Objectif*: revue du travail (produit) réalisé par l'équipe pendant le sprint et démo
- Le PO valide les fonctionnalités
- Toute l'équipe participe
- On invite du monde

Scrum - Artefacts

Sprint retrospective

- *Objectif*: Réfléchir régulièrement à ce qui marche et ce qui ne marche pas
- Fait à la fin de chaque sprint
- Toute l'équipe participe
  - ScrumMaster
  - Product Owner
  - Equipe

Scrum



Source: Martin Christensen, http://www.kaeru.se/

Outils pour le développement logiciel

Comment travailler?

- Sur des projets complexes
  - Gestion de versions
  - Outils de build
  - Intégration continue

Intégration continue

- Ensemble de bonnes pratiques en développement logiciel :
  - Dépôt versionné
  - Compilation automatique
  - Automatisation des tests
  - Déploiement automatisé

➢ La version du dépôt est toujours utilisable (compilable, exécutable, testable)

Utiliser

- des systèmes de gestion de versions
  - `Subversion / GIT /...`
- des outils de build
  - `Make / Ant / Maven /...`
- des plate-formes d'intégration continue
  - `(Hudson) / Jenkins /...`

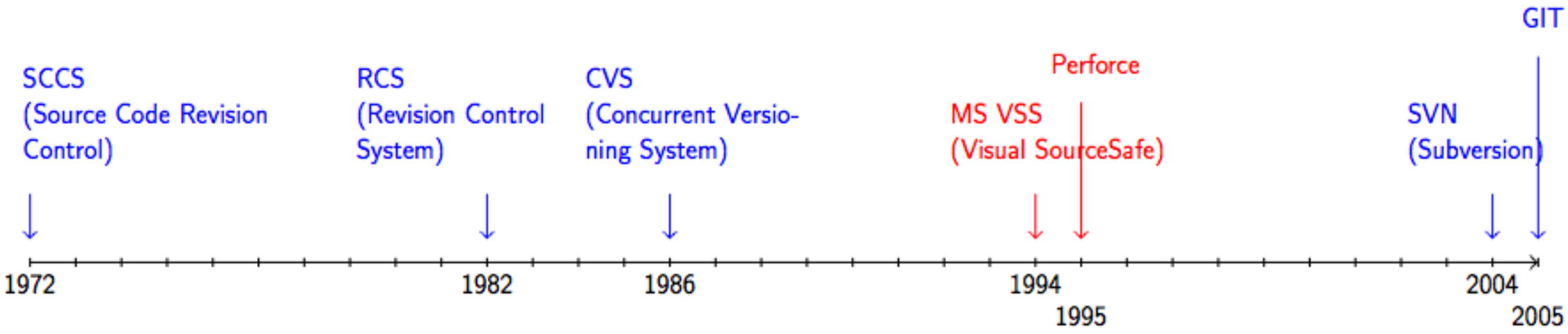
Systèmes de gestion de versions

Version Control Systems (VCS)

Problématiques

- Sécurité (sauvegarde avec historique)
  - Sauvegarder ses données
  - Revenir à une version antérieure d'un fichier/projet
- Parallélisme (multi-machines, multi-développeurs)
  - Travailler sur plusieurs machines
  - Travailler à plusieurs sur un même fichier/projet

Histoire



Source: Karën Fort

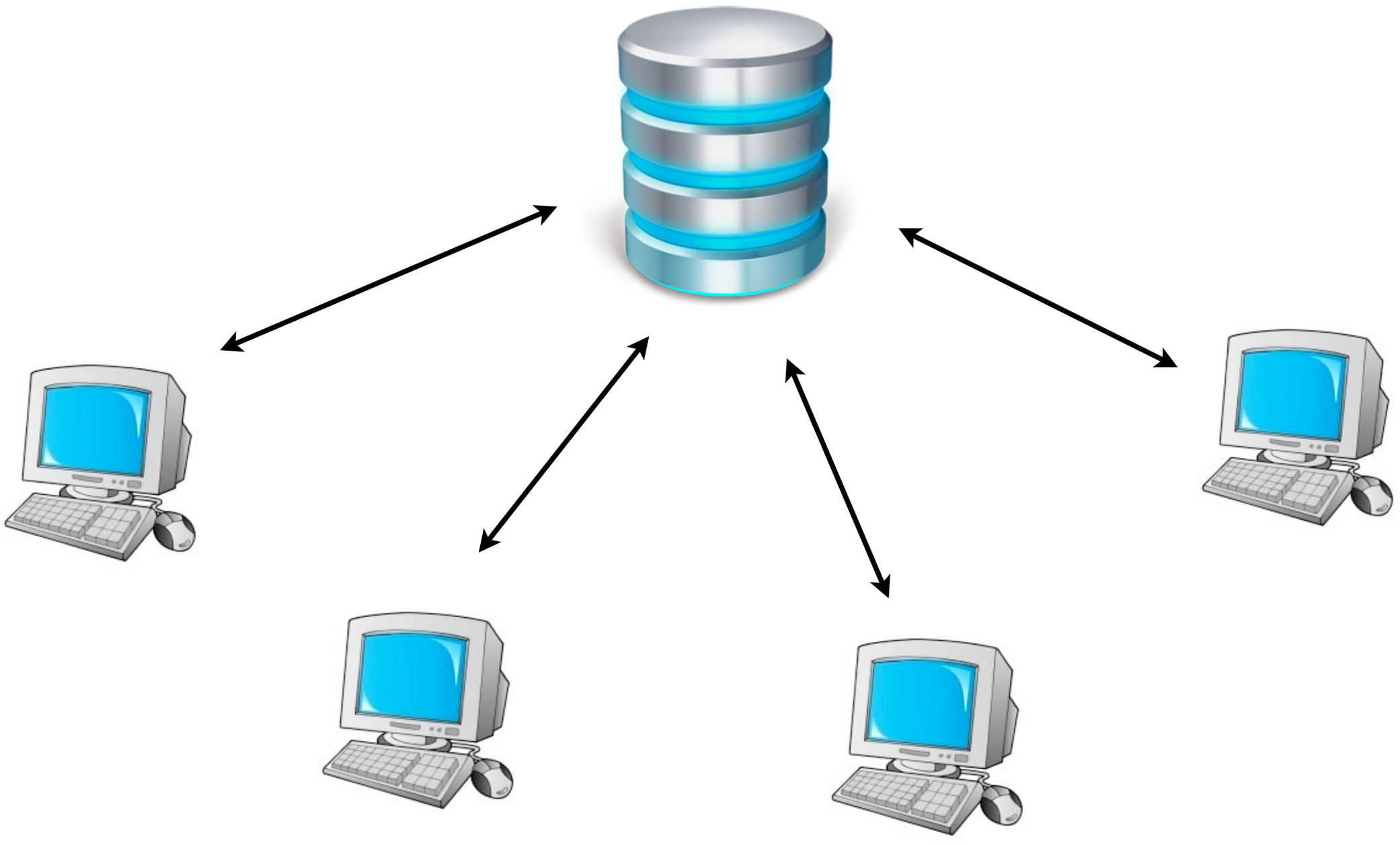
Generation	Networking	Operations	Examples
First	None	One file at a time	RCS, SCCS
Second	Centralized	Multi-file	CVS, SourceSafe, Subversion,
Third	Distributed	Changesets	Bazaar, Git, Mercurial

Source: http://ericsink.com/

Familles

- **Centralisé** `Subversion, CVS,...`
  - Chaque collaborateur travaille sur une copie locale
  - Synchronisation des copies via un dépôt centralisé
- **Décentralisé** `Git, Mercurial, Darcs,...`
  - Travail sur une copie locale
  - Synchronisation sur le dépôt local
  - Synchronisation sur des dépôts distants

Subversion

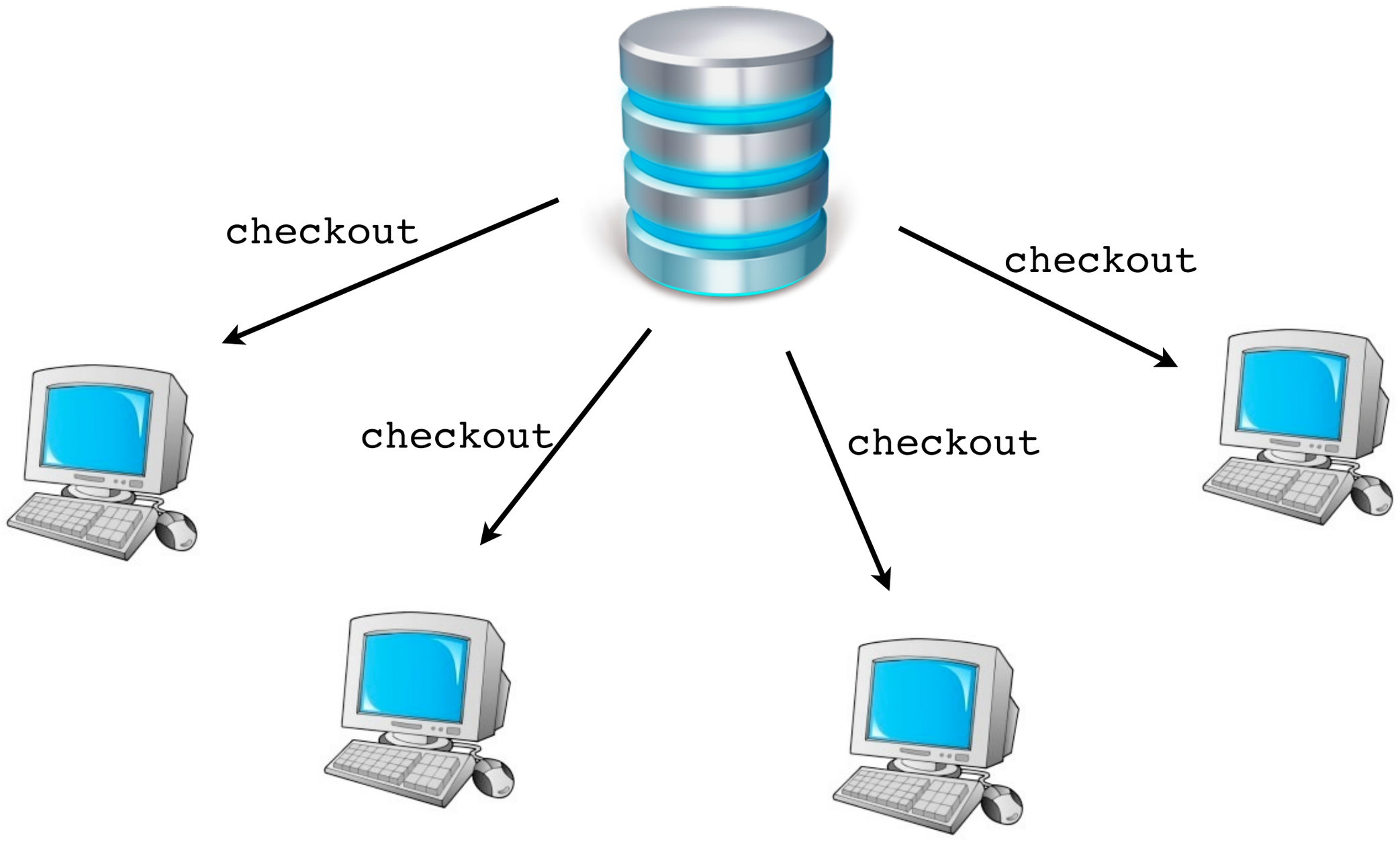


Subversion

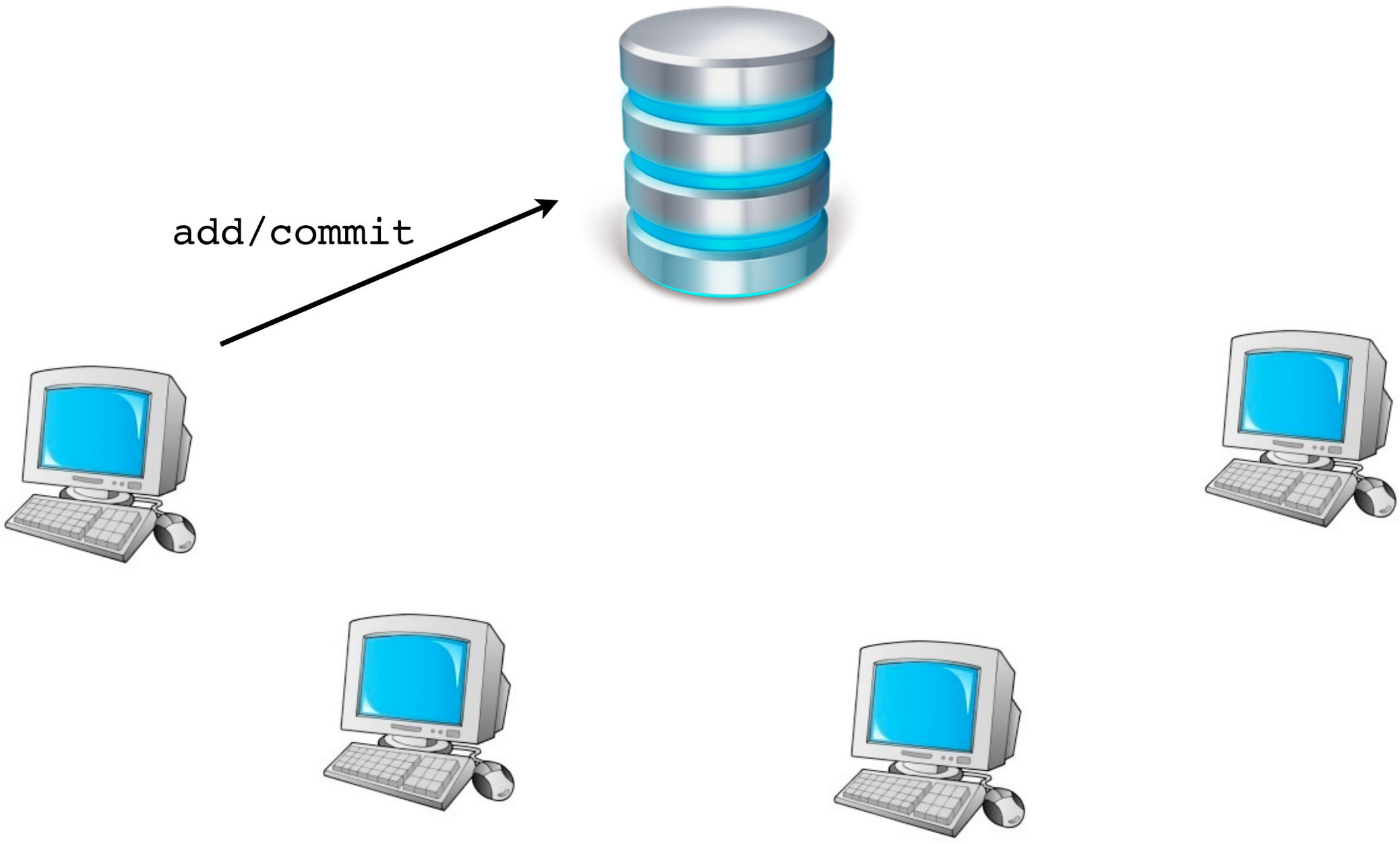
Commande **`svn`**

- **`checkout`**: récupère localement la version du dépôt
- **`commit`**: valide les changements faits localement et les ajoute au dépôt
- **`update`**: met à jour la copie locale
- **`add`** : ajoute un fichier dans le dépôt
- **`status`**: vérifie l'etat local par rapport à l'état du dépôt distant

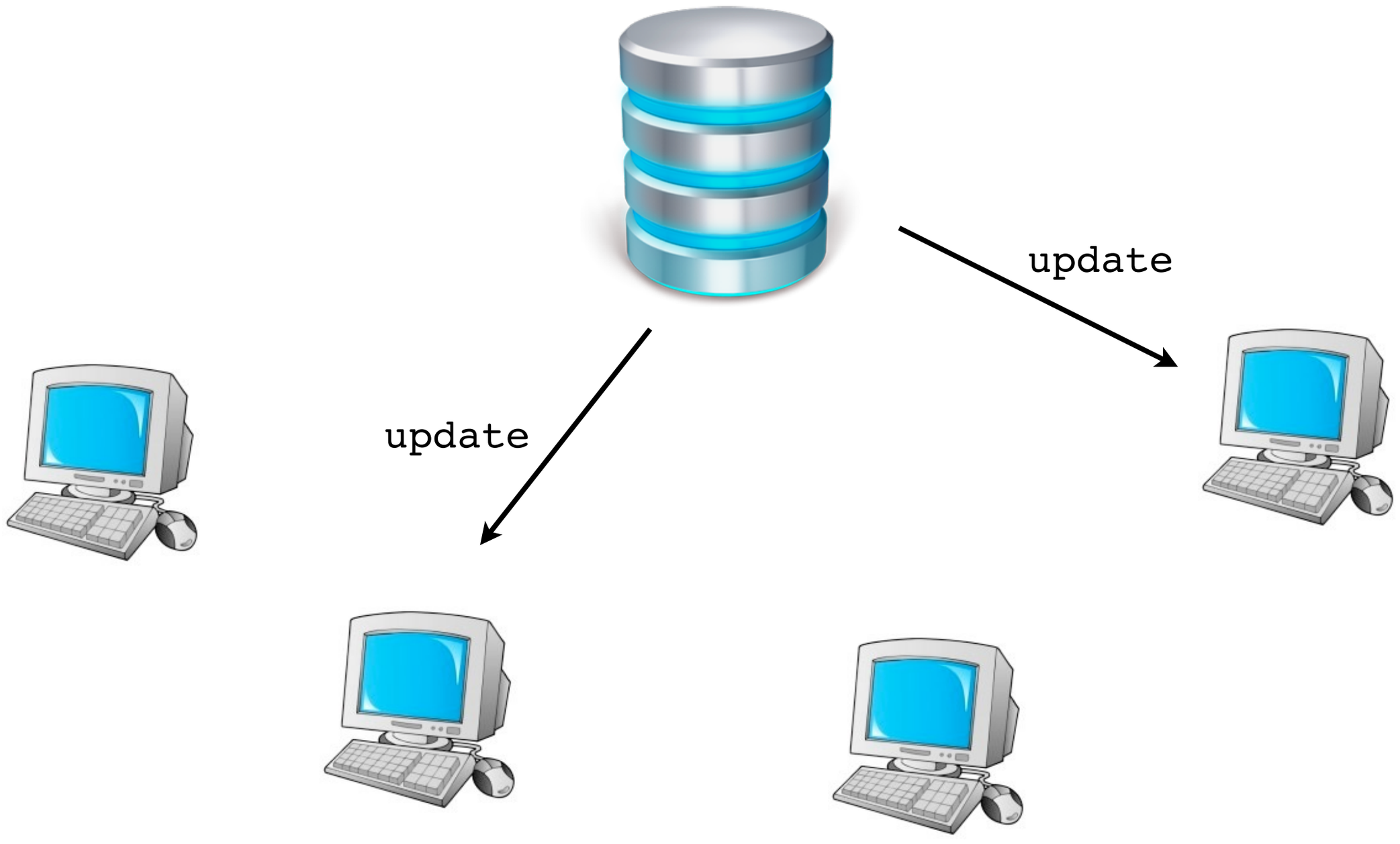
Récupérer dépôt



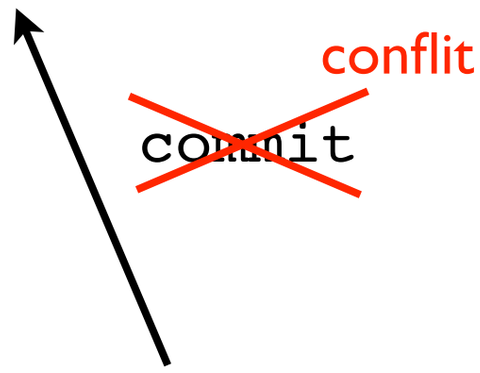
Ajouter/modifier fichier(s)



Mis à jour copie locale



(Résolution de) conflits



(Résolution de) conflits

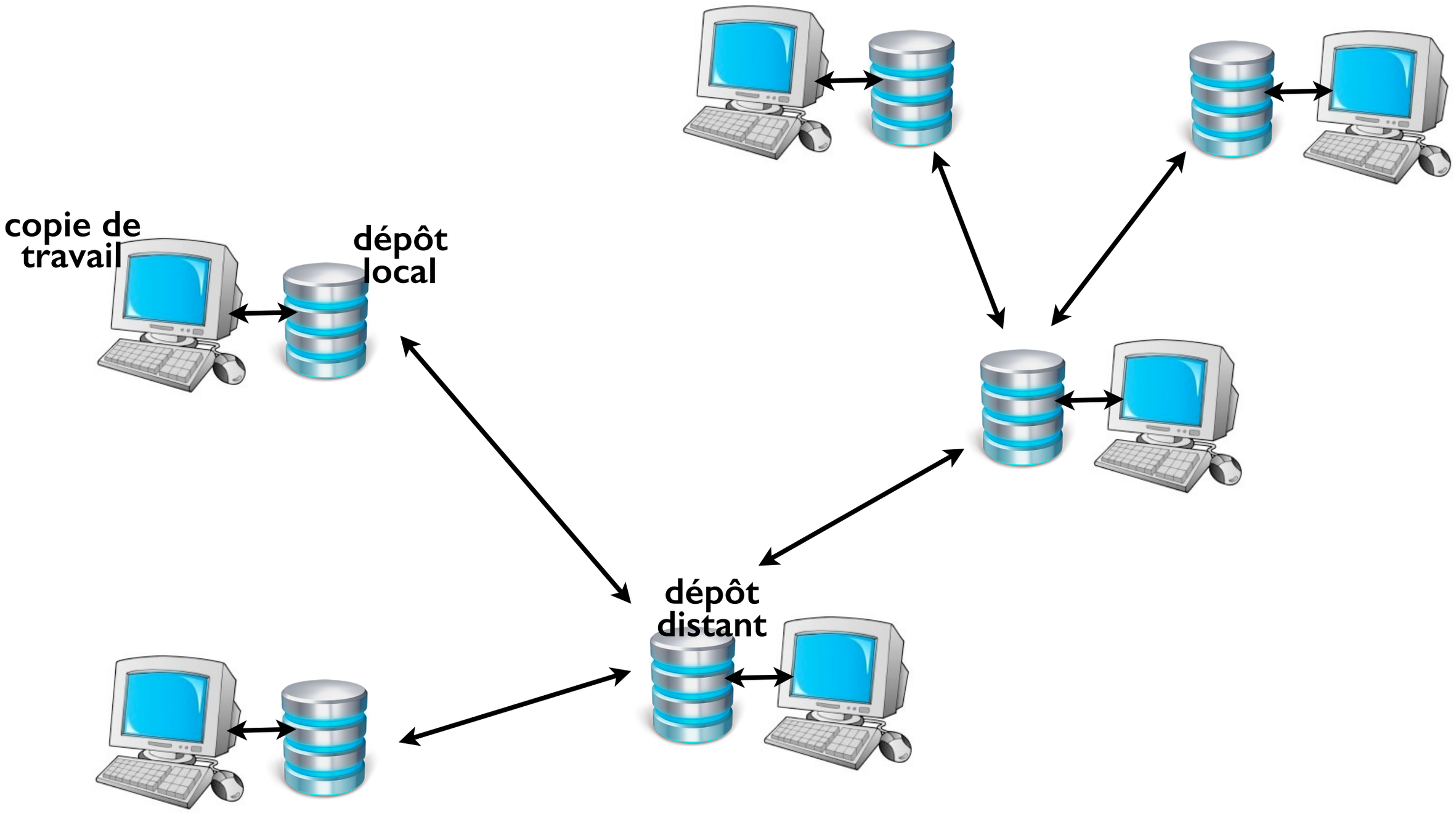
`update`

résolution conflits

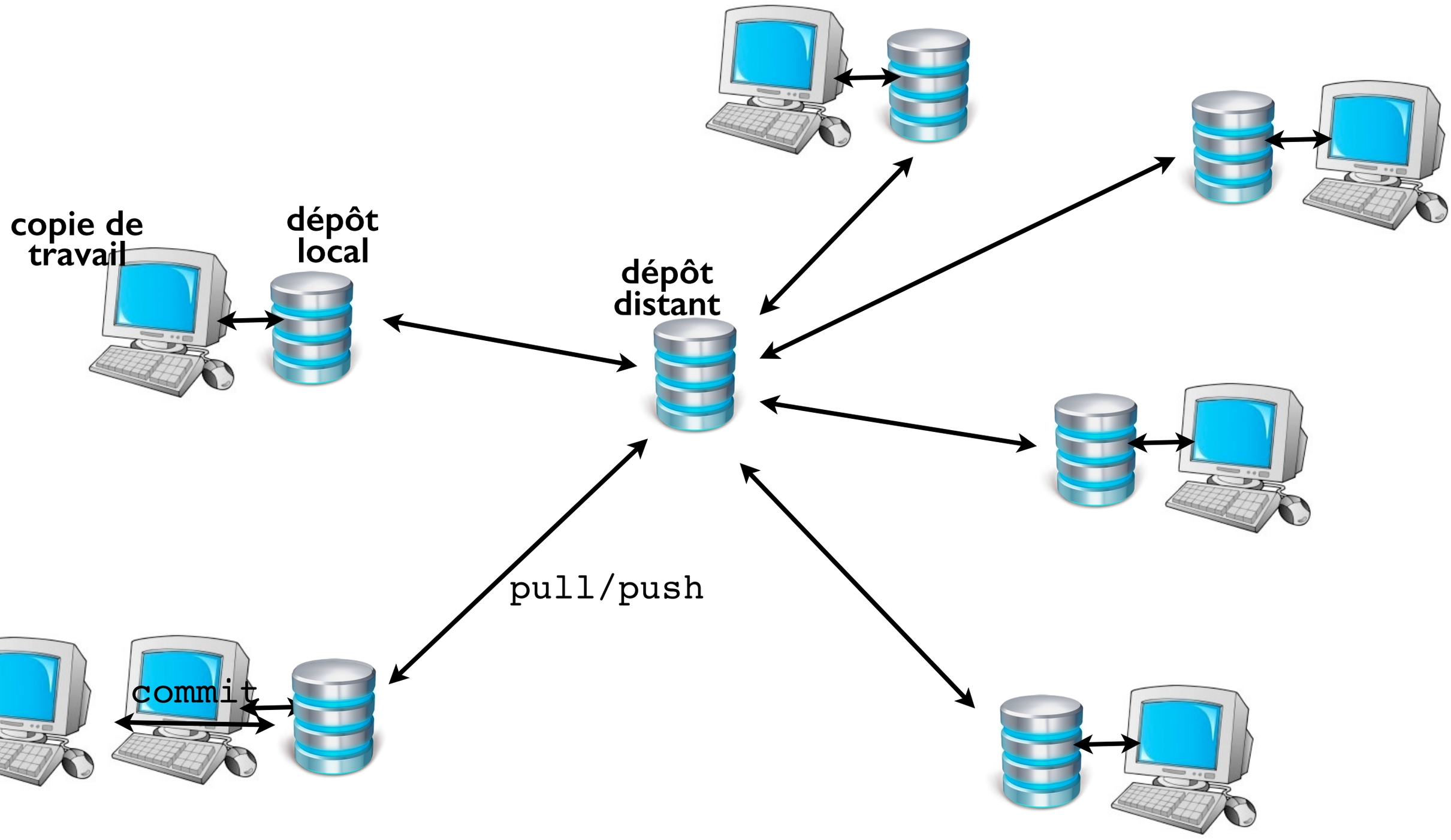
`commit`

Git

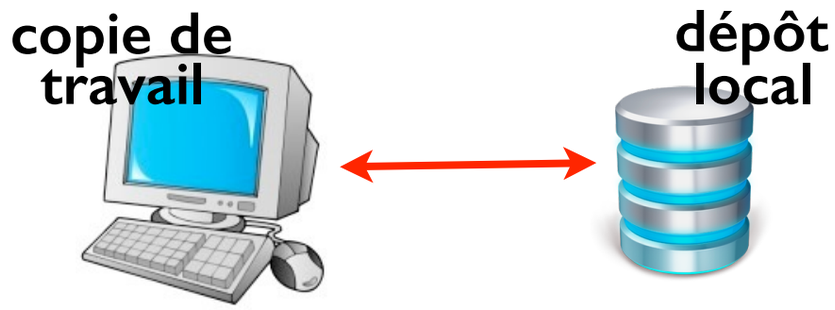
http://git-scm.com/



Git avec dépôt central



Git

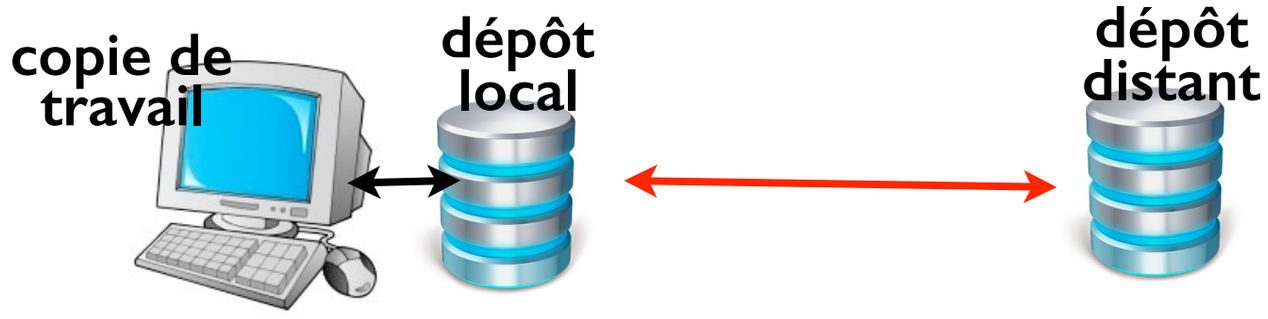


Commande **`git`**

- Commandes principales pour le dépôt local :
  - **`checkout`**: récupère la version du dépôt local (et écrase les modifications non validées)
  - **`commit`**: valide les changements faits localement et les intègre au dépôt local
  - **`add`** : ajoute un fichier au suivi pour le dépôt local

Git

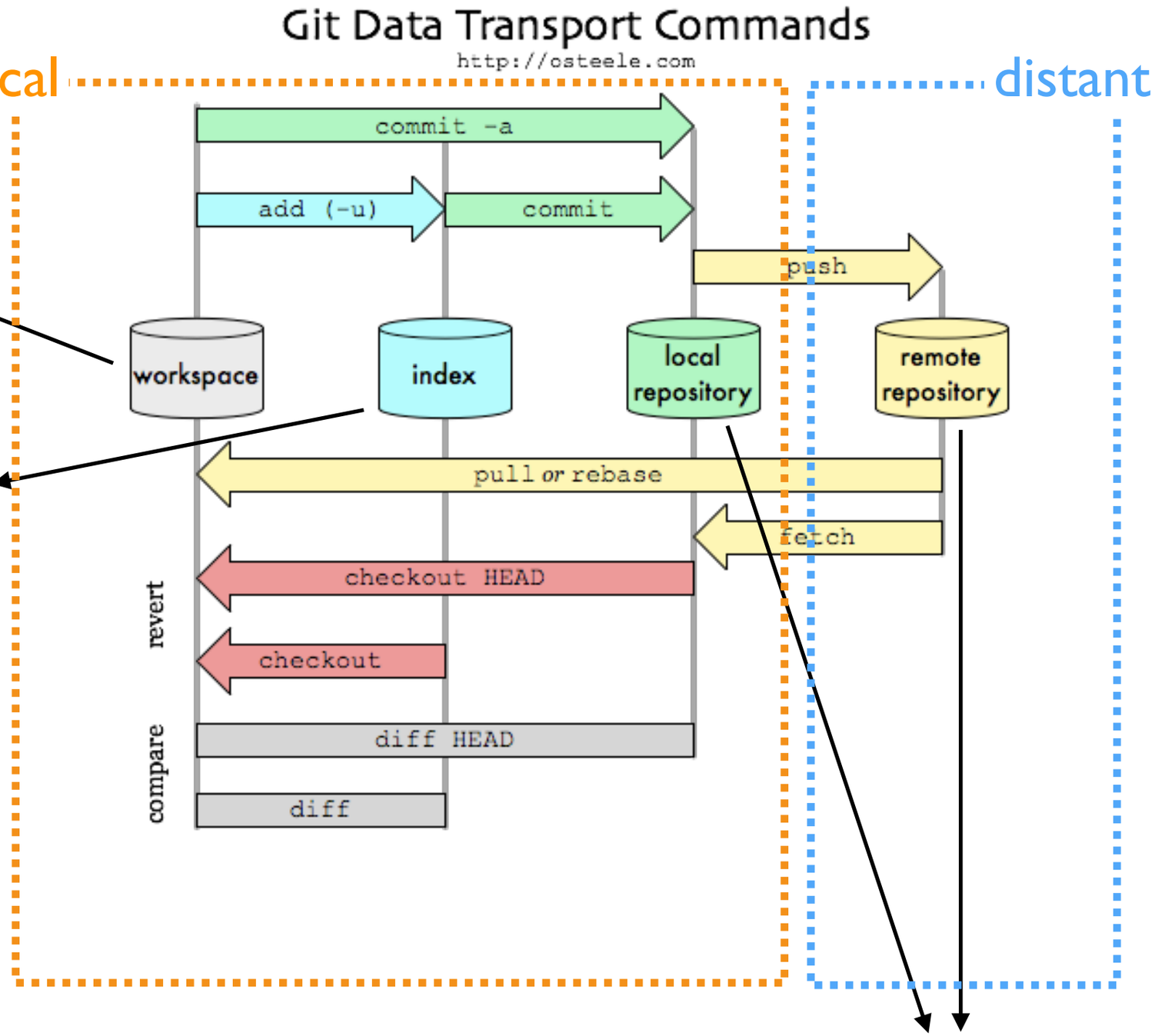
Commande **`git`**



- Commandes principales pour synchroniser avec le(s) dépôt(s) distant(s) :
  - **`clone`**: récupère localement un dépôt distant
  - **`pull`**: récupère les mises à jour sur les dépôts distants
  - **`push`**: envoie les mises à jour locales vers les dépôts distants

copie locale
des sources

espace temporaire qui permet de spécifier le sous-ensemble de modification du workspace à répercuter dans le dépôt local lors d'un commit



dépôt(s) contenant toutes les versions commitées ainsi des méta-informations (logs, tags, etc.)

- **Création**
  - dépôt local vide : `git init`
  - à partir d'un dépôt existant : `git clone`
- **Ajout**
  - au suivi : `git add`
  - au dépôt local : `git commit`
- **Synchronisation**
  - récupérer changements
    - dans dépôt local : `git fetch`
    - dans répertoire travail : `git pull`
  - propager changements : `git push`

- **Informations**
  - infos répertoire travail et index : `git status`
  - historique des commits : `git log`
  - détails d'un commit : `git show`
  - différences avec
    - ✦ index : `git diff`
    - ✦ dépôt : `git diff head`

- **Opérations**
  - associer une balise : `git tag`
  - déplacer fichier(s) : `git mv`
  - supprimer un fichier : `git rm`

Gestion conflits

Jason

```
> vi README
> git add README
> git commit -m « JB »
> git pull
> git push
```

Austin

```
> vi README
> git add README
> git commit -m « AP »
> git pull
```

```
CONFLICT (content): Merge conflict in README
Automatic merge failed; fix conflicts and then commit the result.
```

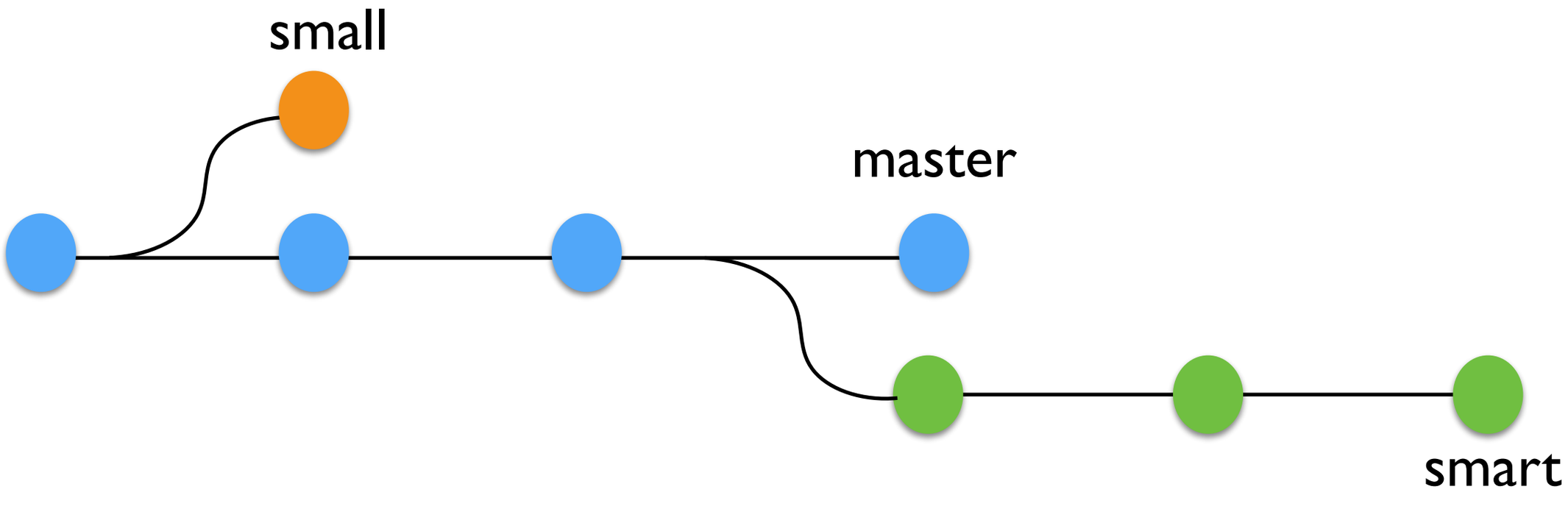
```
> vi README
> git commit a –m « fix AP »
```

Branches (création)

- lignes indépendante de développement
  - `git branch`
    - listes les branches
  - `git branch <branch>`
    - création de `branch`
  - `git branch -d <branch>`
    - suppression (safe) de `branch`
  - `git branch -D <branch>`
    - suppression (force) de `branch`

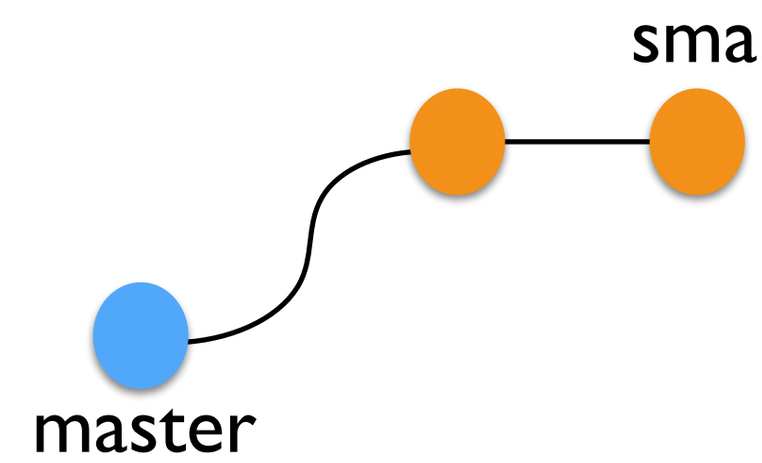
Branches (branchement)

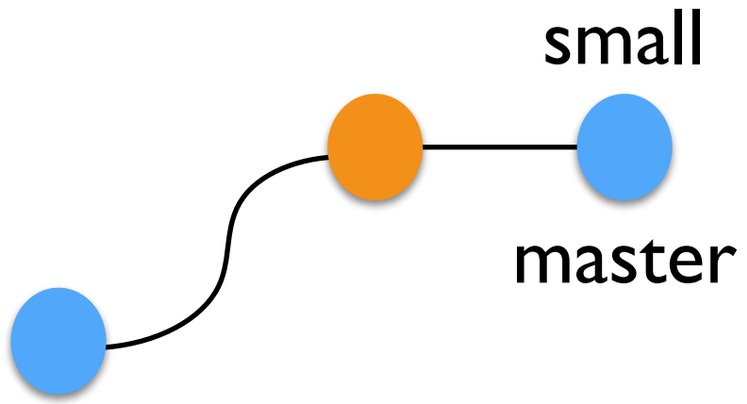
- `git checkout <branch>`
  - selection de `branch`
- `git checkout -b <branch>`
  - création et sélection de `branch`

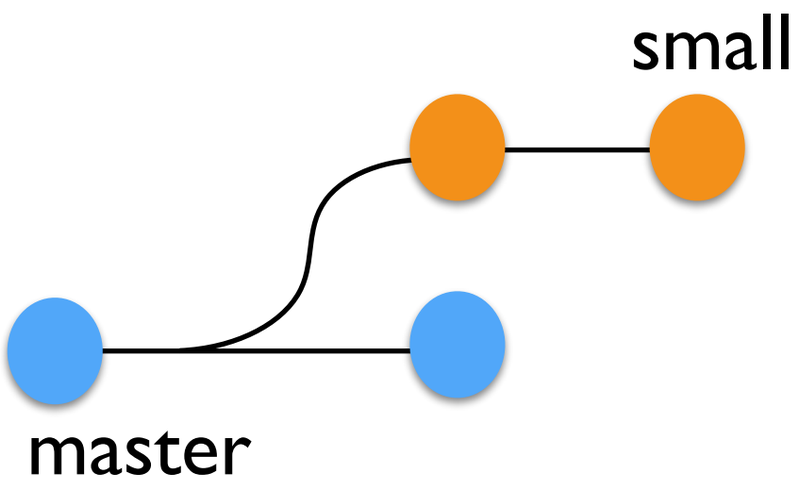


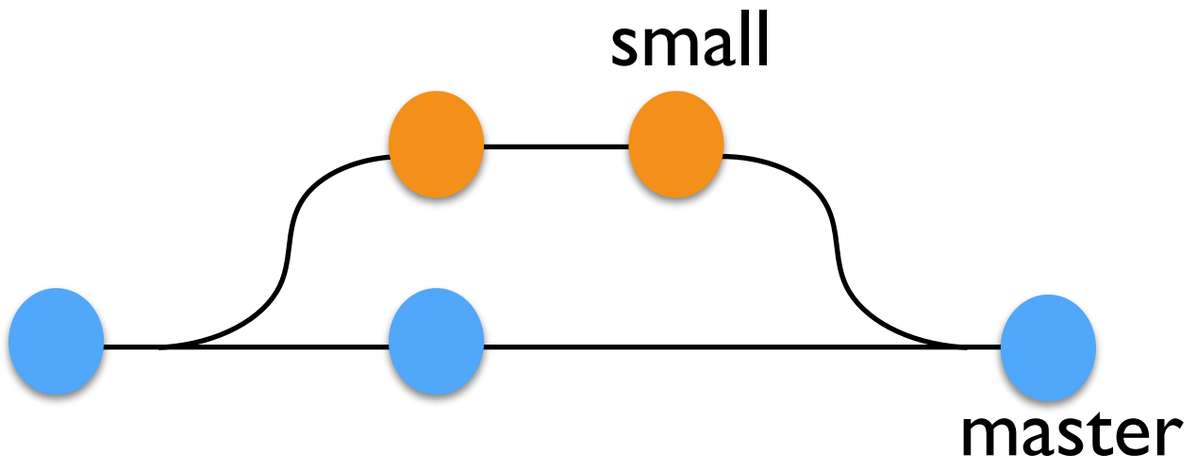
Branches (intégration)

- `git merge <branch>`
  - intégration de `branch` dans la branche courante (sélectionnée avec `checkout`)









Branches (exemple)

```
# Nouvelle branche
git checkout -b smart master
# Faire des ajouts/modifications
git add myFile
...
git commit -m "Finished smart feature"
# Revenir sur branche master branch
git checkout master
# Faire des ajouts/modifications
git add newFile
...
git commit -m "Make changes to master"
# Intégrer la smart feature
git merge smart
git branch -d smart
```

Numérotation versions

Subversion :

- numéros de révisions créés automatiquement s'appliquant à l'arborescence toute entière et non à chaque fichier individuellement (≠ CVS)
- numéros de versions créés manuellement (`svn tag`)

GIT :

- numéros de révisions créés automatiquement et s'appliquent à chaque commit
- numéros de versions créés manuellement (`git tag`)

Numérotation versions

`Major.Minor.Patch`

- **`Major`** : *Backwards incompatible* breaking changes
- **`Minor`** : New features *backwards compatible*
- **`Patch`** : *Backwards compatible* bug fixes only

Pre-release versions :

- `M.m.p-`**`alpha`** : version interne (bugs potentiels)
- `M.m.p-`**`beta`** : version publique (tests clients)

Bonnes pratiques

- mettre à jour (update/pull) avant de tester
- ne pas casser le build : tester avant de committer
- committer souvent, par petits incréments
- mettre des commentaires utiles pour chaque commit (commit -m “mon commentaire”)

Outils de build

Outils de build

- Simplifier/automatiser le processus de compilation/test/construction/déploiement d'un projet
- Exemples d'outils:
  - `Make`
  - `Ant`
  - `Maven`

make

- Ancêtre de `Ant`
- Très utilisé dans le monde `C/C++`
- Configuration par un fichier `Makefile` qui décrit des règles:

```
cible : dépendance1, dépendance2,...
        commande
simple : hello.c hellolib.c
    gcc -o hello hello.c hellolib.c -I.
```

cible : dépendances

commande

```
all : hello.c hellolib.c
    gcc -o hello hello.c hellolib.c -I.

hello.o : hello.c hellolib.h
  gcc -c -o hello hello.c -I.

hellolib.o : hellolib.c hellolib.h
  gcc -c -o hellolib hellolib.c -I.
```

variables, motifs, macros

```
CC = gcc
CFLAGS = -I.
DEPS = hellolib.h
OBJ = hello.o hellolib.o

%.o : %.c $(DEPS)
  $(CC) -c -o $@ $< $(CFLAGS)

all : $(OBJ)
  gcc -o $@ $^ $(CFLAGS)

clean :
    rm -rf *.o
    rm hello
```

make

- prend comme paramètre une cible :

  `make all`

- si pas de paramètre, `make` applique la première cible du fichier `Makefile`

- ne compile que ce qui doit l'être

⇨ Difficile à utiliser pour un gros projet sans l'aide d'outils annexes de génération

Apache Ant

- Séquenceur de tâches
- Equivalent de `make` pour le monde Java
- Configuration des tâches (<`target`>) par un fichier XML (`build.xml`)
- http://ant.apache.org

`build.xml`

- fichier XML conforme à la grammaire définie par `Ant`
- racine <`project`> contenant
  - des <`target`> = des tâches à accomplir
  - des <`property`> = des variables
  - ...

Exemple

```
<project default="hello">
  <target name="hello">
    <echo message="Hello World"/>
  </target>
</project>
```

⇨ attributs `project`

- `default`
- `name`
- `basedir`

```
 > ant
Buildfile: build.xml

hello:
     [echo] Hello World

BUILD SUCCESSFUL
Total time: 1 seconds
```

Exemple

```
<project default="hello">
  <target name="hello">
    <echo message="Hello World"/>
  </target>
  <target name="goodbye">
    <echo message="Goodbye World"/>
  </target>
</project>
```

```
 > ant
Buildfile: build.xml

hello:
     [echo] Hello World

BUILD SUCCESSFUL
Total time: 1 seconds
```

```
 > ant goodbye
Buildfile: build.xml

hello:
     [echo] Goodbye World

BUILD SUCCESSFUL
Total time: 1 seconds
```

Exemple

```
<project default="compile">
  <target name="compile">
      <javac srcdir="." />
  </target>
</project>
```

```
> ant
Buildfile: build.xml
compile:
    [javac] Compiling 1 source file
BUILD SUCCESSFUL
Total time: 1 seconds

> ls
Hello.class  Hello.java  build.xml
```

```
<project default="hello">
  <target name="hello">
    <echo message="Hello World"/>
  </target>
  <target name="goodbye">
    <echo message="Goodbye World"/>
  </target>
  <target name="all" depends="hello,goodbye">
    <echo message="Done”/>
  </target>
</project>
```

Attention aux cycles !

```
 > ant all
Buildfile: build.xml
hello:
     [echo] Hello World
goodbye:
     [echo] Goodbye World
all:
     [echo] Done
BUILD SUCCESSFUL
Total time: 1 seconds
```

Propriétés

- définition de variables utilisables dans le fichier
- attributs
  - `name` : nom de la variable
  - `value` : valeur de la variable

```
<property name="src.dir" value="src" />
```

- utilisation

```
<javac srcdir="${src.dir}" ...>
```

Tâches

- la plus petite unité de travail
- peuvent opérer sur plusieurs fichiers avec la même opération appliquée à chaque fichier
  - `echo` : afficher un message (attribut : message)
  - `delete` : supprimer un dossier/fichier (dir/file)
  - `mkdir` : créer un dossier (dir)
  - `javac` : compiler (srcdir et destdir)
  - `jar` : créer une archive (destfile, basedir)
  - `...`

Exemple projet Java

basedir
- src
  - acl
    - Hello.java
- build
  - classes
    - acl
      - Hello.class
      - _____.class
  - jar
    - HelloApp.jar

```
<project name="HelloApp" basedir="." default="main">
    <property name="src.dir"     value="src"/>
    <property name="build.dir"   value="build"/>
    <property name="class.dir"   value="${build.dir}/classes"/>
    <property name="jar.dir"     value="${build.dir}/jar"/>

    <property name="main-class" value="acl.Hello"/>

    <target name="compile">
        <mkdir dir="${class.dir}"/>
        <javac srcdir="${src.dir}" destdir="${class.dir}"/>
    </target>
```

propriétés génériques

propriété spécifique

Exemple projet Java

```
<project name="HelloApp" basedir="." default="main">
    ...
    <target name="jar" depends="compile">
        <mkdir dir="${jar.dir}"/>
        <jar destfile="${jar.dir}/${ant.project.name}.jar"
             basedir="${class.dir}">
            <manifest>
                <attribute name="Main-Class" value="${main-class}"/>
            </manifest>
        </jar>
    </target>

    <target name="run" depends="jar">
        <java jar="${jar.dir}/${ant.project.name}.jar" fork="true"/>
    </target>
```

java -jar build/jar/HelloApp.jar

attribut du tag **project**

Exemple projet Java

```
<project name="HelloApp" basedir="." default="main">
    ...
    <target name="jar" depends="compile">
        <mkdir dir="${jar.dir}"/>
        <jar destfile="${jar.dir}/${ant.project.name}.jar"
             basedir="${class.dir}">
        </jar>
    </target>

    <target name="run" depends="jar">
        <java classname="${main-class}" fork="true">
          <classpath path="${jar.dir}/${ant.project.name}.jar" />
        </java>
    </target>
```

```
java -cp build/jar/HelloApp.jar acl.Hello
```

Exemple projet Java

```
> ls
build.xml  src

> ant compile
Buildfile: <bd>/build.xml
compile:
    [mkdir] Created dir: <bd>/build/classes
    [javac] Compiling 1 source file to <bd>/build/classes
jar:
    [mkdir] Created dir: <bd>/build/jar
      [jar] Building jar: <bd>/build/jar/HelloApp.jar
Total time: 1 second

> ls
build   build.xml  src

> ls build
classes jar
```

Exemple projet Java

```
<project name="HelloApp" basedir="." default="main">
 ...
 <target name="run" depends="jar">
    <java jar="${jar.dir}/${ant.project.name}.jar" fork="true"/>
 </target>

 <target name="clean">
     <delete dir="${build.dir}"/>
 </target>
 <target name="clean-jar">
     <delete file="${jar.dir}/${ant.project.name}.jar" />
 </target>
 <target name="clean-compile">
    <delete>
       <fileset dir="${class.dir}" includes="**/*.class" />
    </delete>
 </target>

 <target name="clean-build" depends="clean,jar"/>

 <target name="main" depends="clean,run"/>
</project>
```

Tests

```
<project name="HelloApp" basedir="." default="main">
    ...
    <property name="test.dir" value="test"/>
    <property name="test.report.dir" value="${build.dir}/testrep"/>

    <target name="compile">
        <mkdir dir="${class.dir}"/>
        <javac srcdir="${src.dir}" destdir="${class.dir}"/>
        <javac srcdir="${test.dir}" destdir="${class.dir}"/>
</target>
...
```

Tests

```
...
<target name="junit" depends="compile">
    <mkdir dir="${test.report.dir}"/>
    <junit printsummary="yes" fork="yes" haltonfailure="off">
        <classpath>
          <pathelement location="${class.dir}"/>
        </classpath>

        <formatter type="plain"/>

        <!-- Test by test -->
        <test name="acl.HelloTest" todir="${test.report.dir}"/>

        <!-- Same with batch testing -->
        <batchtest fork="yes" todir="${test.report.dir}">
             <fileset dir="${class.dir}">
                 <include name="acl/*Test.class" />
             </fileset>
        </batchtest>
   </junit>
</target>
```

Apache Ant

- outil puissant souvent intégré dans les IDE description de tâches (<target>) dans un fichier XML
- chaque <target> correspond à une étape du processus de construction
- extensible : il est possible d'ajouter d'autres tâches et éléments.

Maven

- Outil pour faciliter la gestion de projets Java
  - utilise une structure ***standard*** de projet
  - suit un cycle de vie de projet ***standard***
  - utilise un modèle abstrait de projet (POM)

Maven

- Automatisation
  - compilation
  - récupération de dépendances
  - déploiement (génération jar, war, ...)
  - test unitaires
  - génération de la documentation
  - production site web du projet (membres, état d'avancement, rapports sur les tests, ...)

Structure projet standard

`/src` : sources du projet

`/src/main` : fichiers source principaux

`/src/main/java` : code source Java

`/src/main/resources` : fichiers de ressources

...

`/src/test` : fichiers de test

`/src/test/java` : code source de test

...

`/target` : fichiers résultat, binaires, packages générés, résultats des tests

➢ Convention plutôt que configuration

Phases standard

`validate` : vérification projet correct et info dispo
`compile` : compilation du code source
`test` : exécution tests unitaires
`package` : assemblage binaires dans un jar, war, ...
`integration-test` : test du code dans un clone de l'environnement de déploiement
`verify` : vérification intégrité/qualité de l'archive
`install` : ajouter package sur le dépôt local
`deploy` : ajouter package sur le dépôt public

➢ Phases mappées à des *goals*

Project Object Model

```
<project xmlns="..." xmlns:xsi="..." ...>
  <modelVersion>4.0.0</modelVersion>
  <groupId>fr.ul.acl</groupId>
  <artifactId>acl-maven-app</arti
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>acl-maven-app</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

version de l'object model

Project Object Model

```
<project xmlns="..." xmlns:xsi=".
  <modelVersion>4.0.0</modelVers
  <groupId>fr.ul.acl</groupId>
  <artifactId>acl-maven-app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>acl-maven-app</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

identifiant organisation à l'origine du projet

identifiant de l'artefact généré par le projet

Project Object Model

```
<project xmlns="..." xmlns:xsi="..." ...>
  <modelVersion>4.0.0</modelVersion>
  <groupId>fr.ul.acl</groupId>
  <artifactId>acl-maven-app</ar
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>acl-maven-app</name>
  <url>http://maven.apache.org<
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

type packaging

version de l'artefact généré

Project Object Model

```
<project xmlns="..." xmlns:xsi="..." ...>
  <modelVersion>4.0.0</modelVersion>
  <groupId>fr.ul.acl</groupId>
  <artifactId>acl-maven-app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>acl-maven-app</name>
  <url>ht                    n.apache.org</url>
  <dependen
    <depende
      <group
      <artif
      <versi
      <scope
    </depend
  </dependen
</project>
```

acl-maven-app

|

|---src/main/java/fr/ul/acl/...

...

|---target/acl-maven-app-1.0-SNAPSHOT.jar

...

Project Object Model

```
<project xmlns="..." xmlns:xsi="..." ...>
  <modelVersion>4.0.0</modelVersion>
  <groupId>fr.ul.acl</groupId>
  <artifactId>acl-maven-app</
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>acl-maven-app</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</grou
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

nom du projet utilisé pour l'affichage

adresse du site du projet

Project Object Model

```
<project xmlns="..." xmlns:xsi="..." ...>
  <modelVersion>4.0.0</modelVersion>
  <groupId>fr.ul.acl</groupId>
  <artifactId>acl-maven-app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>acl-maven-app</name>
  <url>http://maven.apache.o
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

dépendances utilisées par la projet

Dans la pratique

- Création projet

```
> mvn3 archetype:generate
        -DgroupId=fr.ul.acl -DartifactId=acl-maven-app
```

- Compiler

```
acl-maven-app > mvn3 compile
```

- Tester

```
acl-maven-app > mvn3 test
```

- Packager

```
acl-maven-app > mvn3 package
```

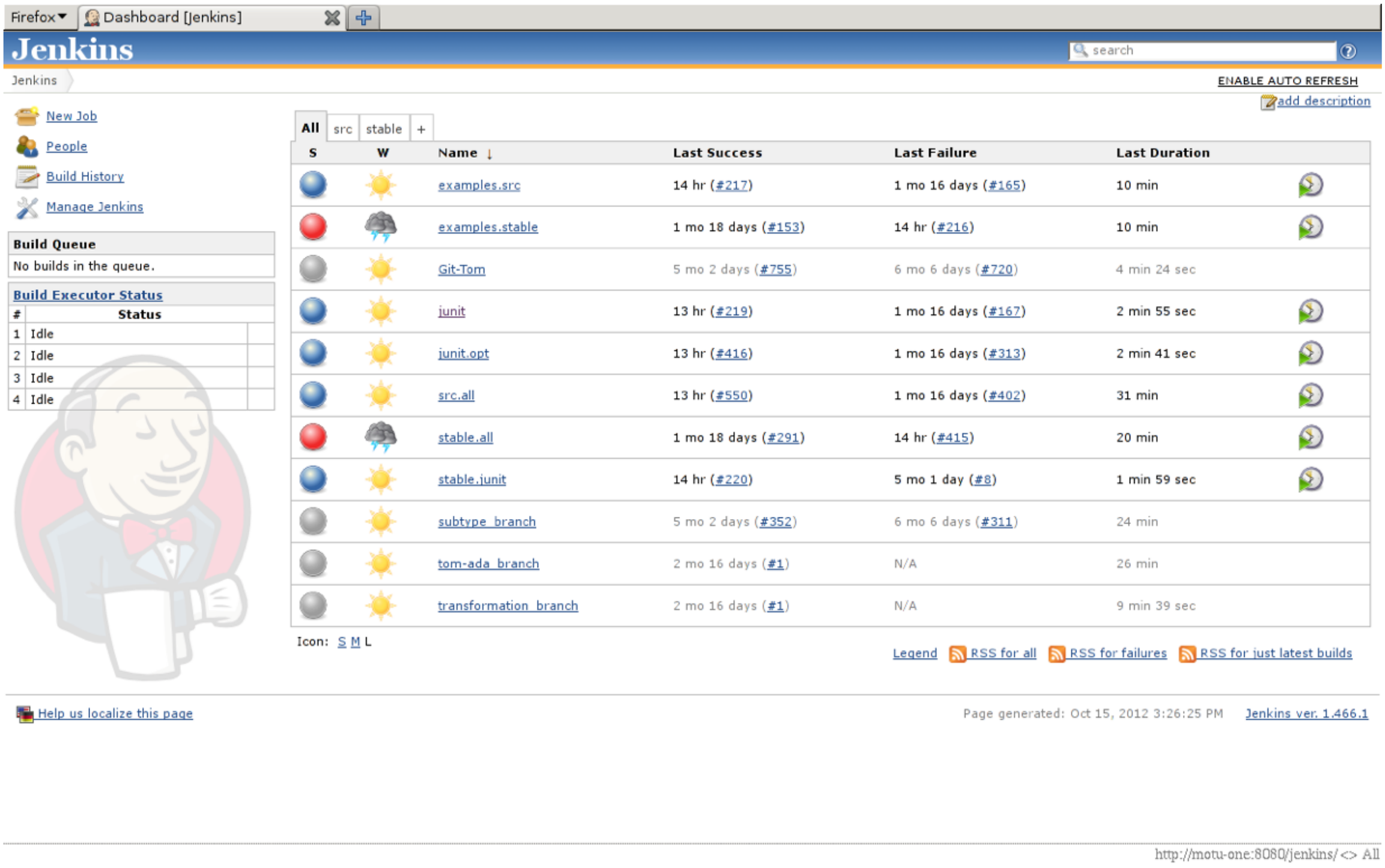
Maven

- Comprendre un projet Maven permet de comprendre tous les projets Maven
- Quand ces conventions sont respectées, tout est quasi-automatique

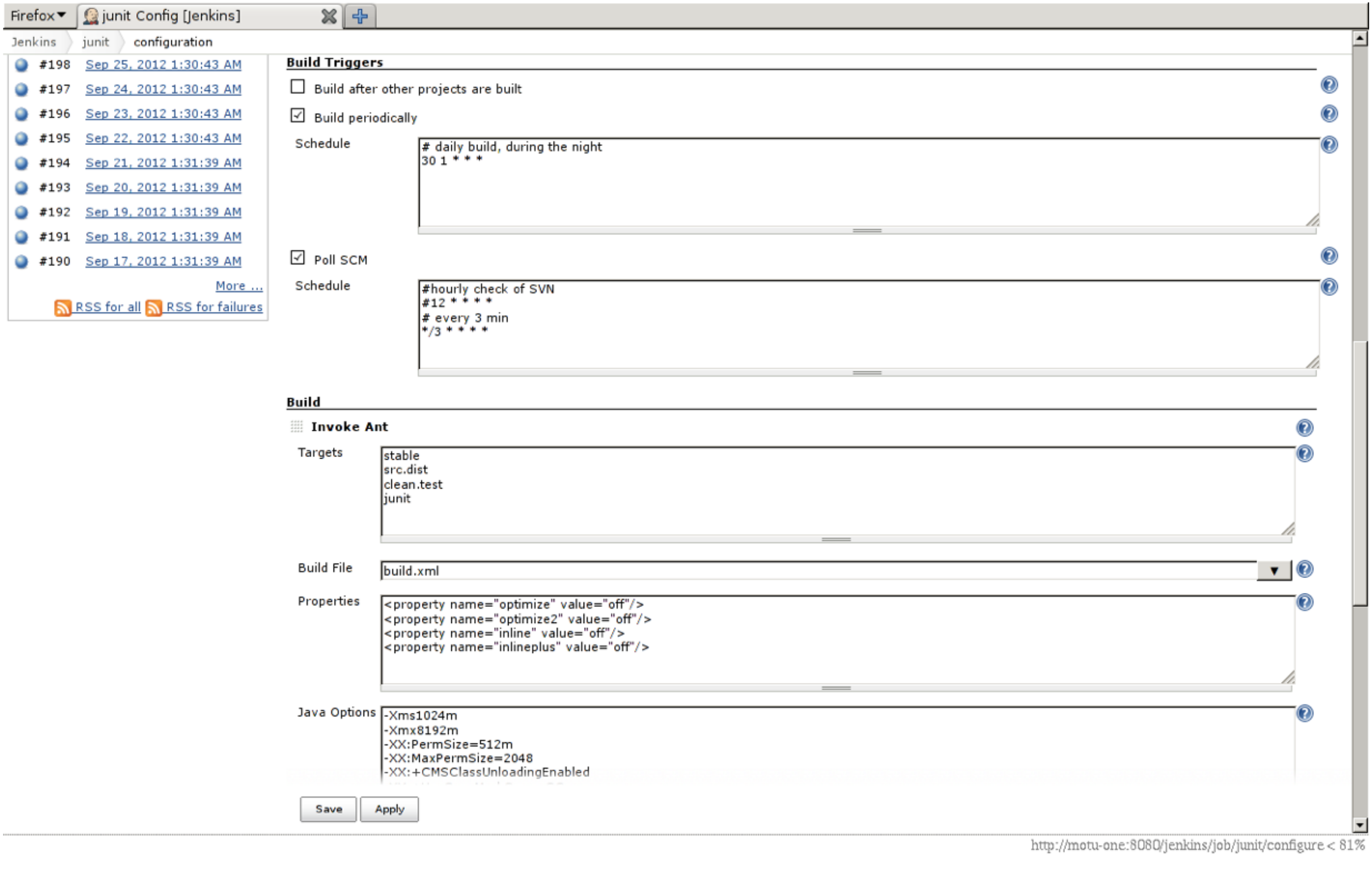
Framework d'intégration continue

- Planification, automatisation et intégration des tâches de compilation, de tests
- Gestion des dépendances entre tâches (ex. : les tests unitaires sont lancés lorsque la compilation s'est bien passée)
- Notifications
- Utilise des outils de build, de versionning, etc.
- Exemples : Hudson/Jenkins, Tinderbox, etc.

Jenkins


Firefox
Dashboard [Jenkins]
Jenkins
search
Jenkins
ENABLE AUTO REFRESH
add description
New Job
People
Build History
Manage Jenkins
Build Queue
No builds in the queue.
Build Executor Status
Status
1 Idle
2 Idle
3 Idle
4 Idle
All
src
stable
+
S
W
Name ↓
Last Success
Last Failure
Last Duration
examples.src
14 hr (#217)
1 mo 16 days (#165)
10 min
examples.stable
1 mo 18 days (#153)
14 hr (#216)
10 min
Git-Tom
5 mo 2 days (#755)
6 mo 6 days (#720)
4 min 24 sec
junit
13 hr (#219)
1 mo 16 days (#167)
2 min 55 sec
junit.opt
13 hr (#416)
1 mo 16 days (#313)
2 min 41 sec
src.all
13 hr (#550)
1 mo 16 days (#402)
31 min
stable.all
1 mo 18 days (#291)
14 hr (#415)
20 min
stable.junit
14 hr (#220)
5 mo 1 day (#8)
1 min 59 sec
subtype_branch
5 mo 2 days (#352)
6 mo 6 days (#311)
24 min
tom-ada_branch
2 mo 16 days (#1)
N/A
26 min
transformation_branch
2 mo 16 days (#1)
N/A
9 min 39 sec
Icon: S M L
Legend
RSS for all
RSS for failures
RSS for just latest builds
Help us localize this page
Page generated: Oct 15, 2012 3:26:25 PM
Jenkins ver. 1.466.1
http://motu-one:8080/jenkins/ <> All

Jenkins


Firefox
junit Config [Jenkins]
Jenkins
junit
configuration
#198 Sep 25, 2012 1:30:43 AM
#197 Sep 24, 2012 1:30:43 AM
#196 Sep 23, 2012 1:30:43 AM
#195 Sep 22, 2012 1:30:43 AM
#194 Sep 21, 2012 1:31:39 AM
#193 Sep 20, 2012 1:31:39 AM
#192 Sep 19, 2012 1:31:39 AM
#191 Sep 18, 2012 1:31:39 AM
#190 Sep 17, 2012 1:31:39 AM
More ...
RSS for all
RSS for failures
Build Triggers
Build after other projects are built
Build periodically
Schedule
daily build, during the night
30 1 * * *
Poll SCM
Schedule
#hourly check of SVN
#12 * * * *
every 3 min
*/3 * * * *
Build
Invoke Ant
Targets
stable
src.dist
clean.test
junit
Build File
build.xml
Properties
<property name="optimize" value="off"/>
<property name="optimize2" value="off"/>
<property name="inline" value="off"/>
<property name="inlineplus" value="off"/>
Java Options
-Xms1024m
-Xmx8192m
-XX:PermSize=512m
-XX:MaxPermSize=2048
-XX:+CMSClassUnloadingEnabled
Save
Apply
http://motu-one:8080/jenkins/job/junit/configure < 81%

Exercices