ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

502045

Software Engineering

Chapter 07

Lesson 10: Version Control

Contents

- What is version control
- History and evolution of version control
- Approaches
- Domain vocabulary

What is version control

- Place to store your source code
- Historical record of what you have done over time

VCS Classification (by repository model)

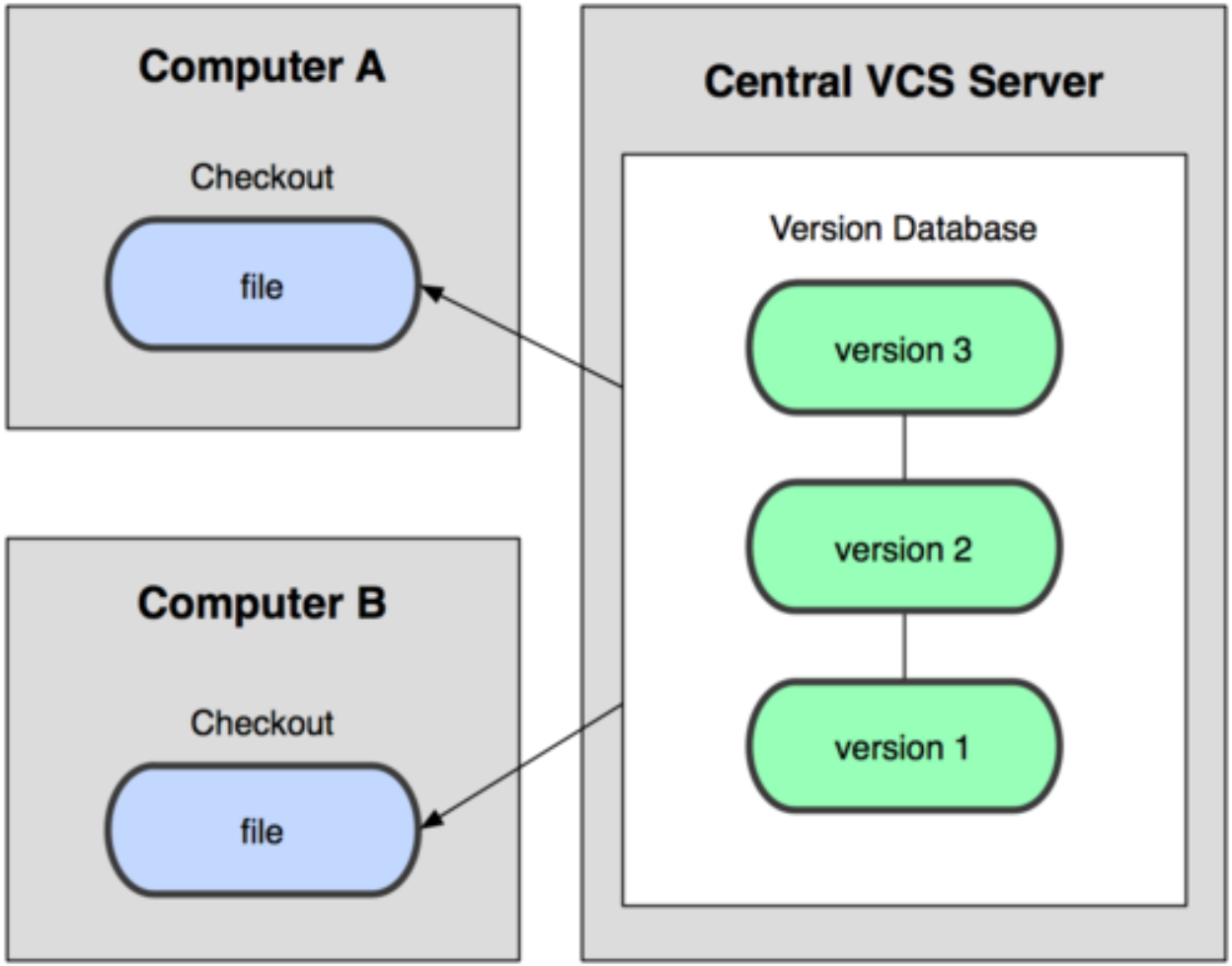
Centralized (client-server model)

- **Subversion**
- CVS
- VSS, TFS, Vault
- ClearCase
- AccuRev

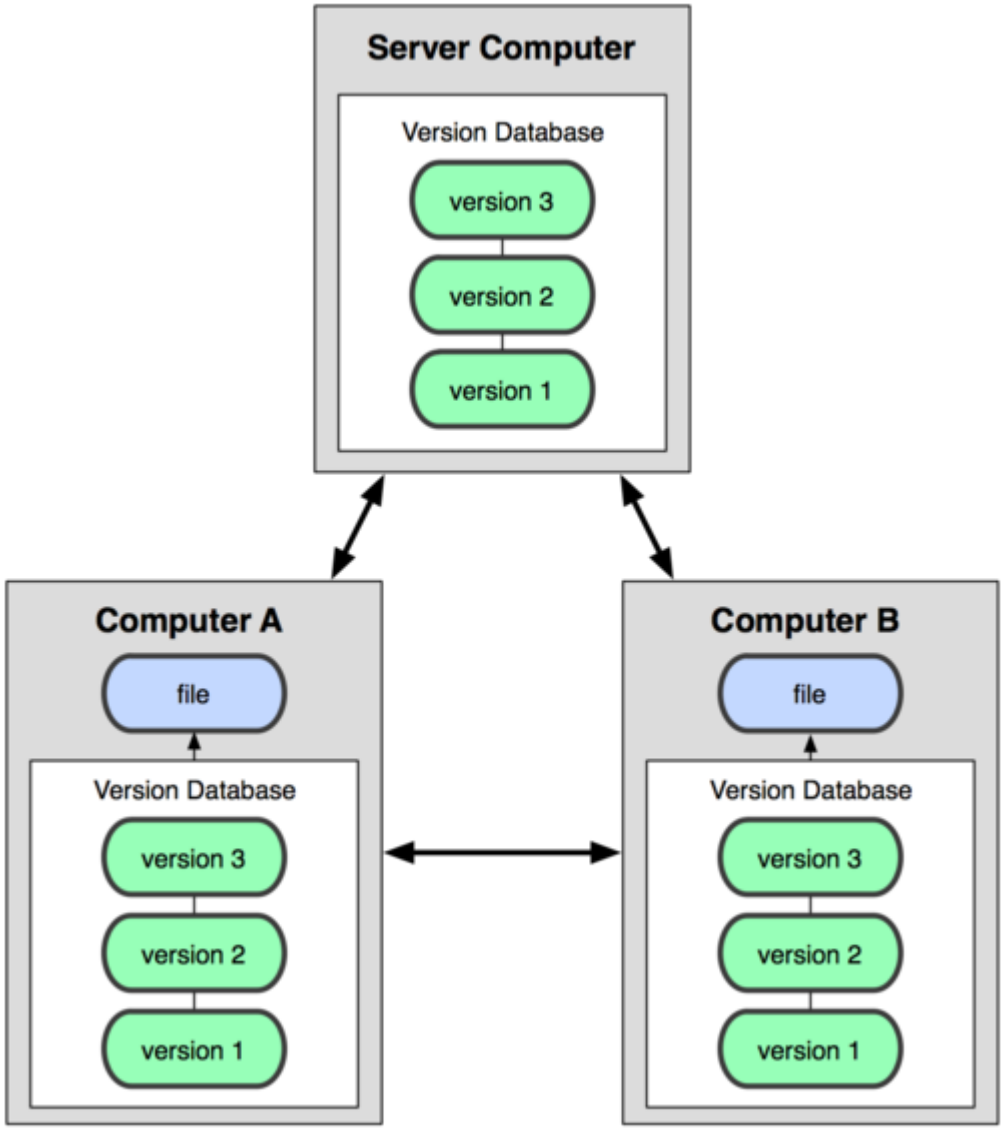
Distributed

- **Git**
- Mercurial
- Bazzar
- Perforce
- BitKeeper

Centralized model



Distributed model



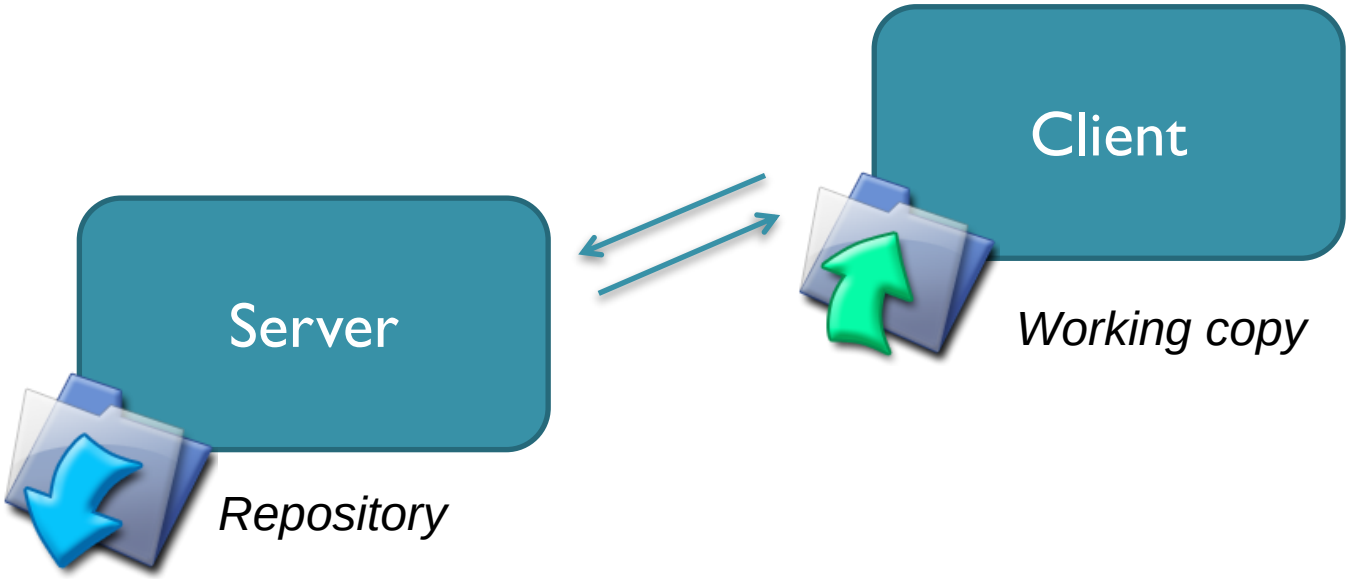
Centralized	Distributed
• Single repository • Commit requires connection • Impossible to commit changes to another user • All history in one place • Easy access management • Chosen for enterprise development	• Multiple repositories • Commit does not require connection • Possible to commit changes to another user • Impossible to get all history • Chosen for open source development

Distributed vs centralized. Differences

Environment

- Repository
  - Where files' current and historical data are stored.
- Server
  - A machine serving the repository.
- Client
  - The client machine connecting to the server.
- Working copy
  - Local copy where the developer changes the code.

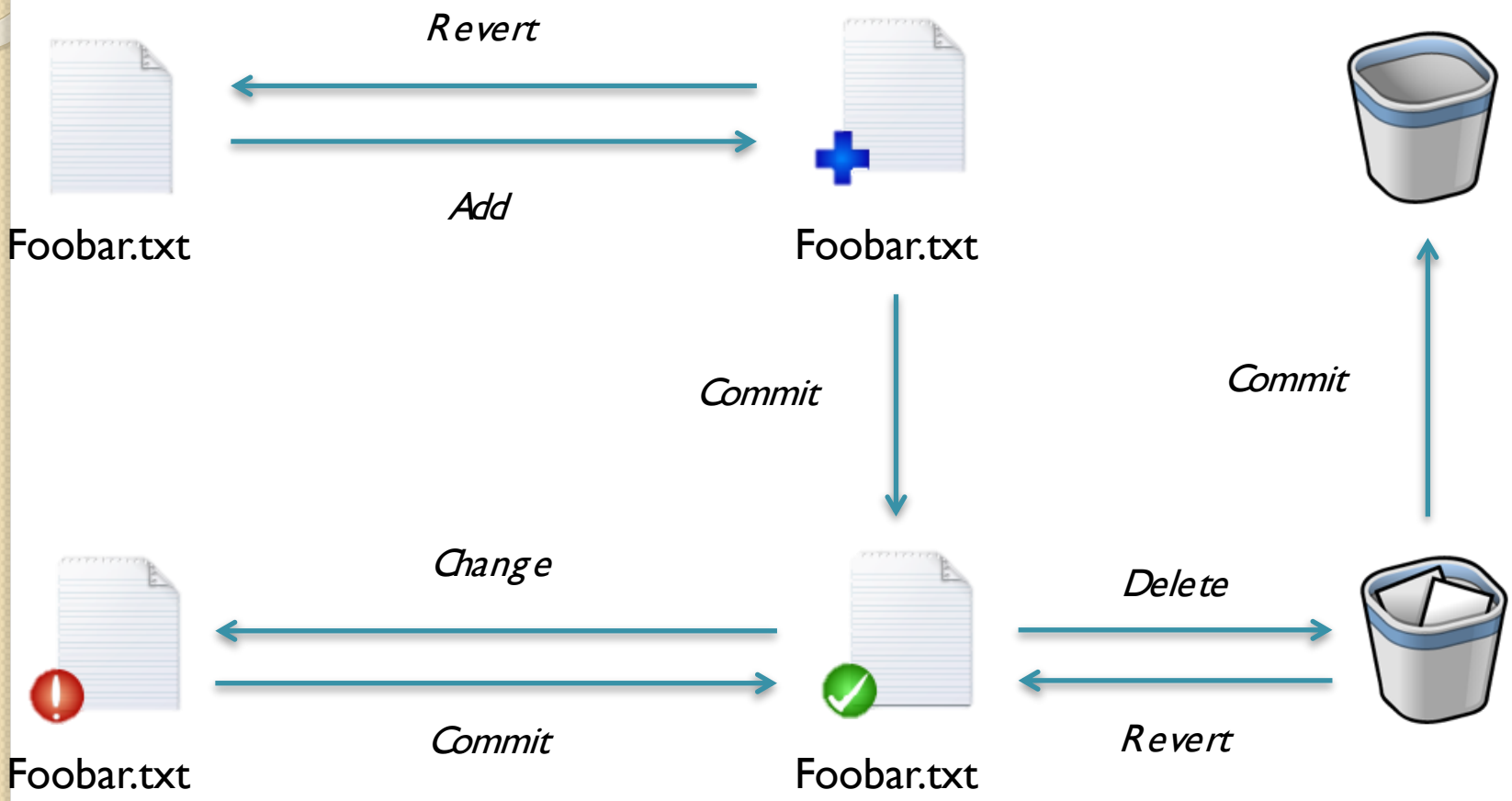
Environment



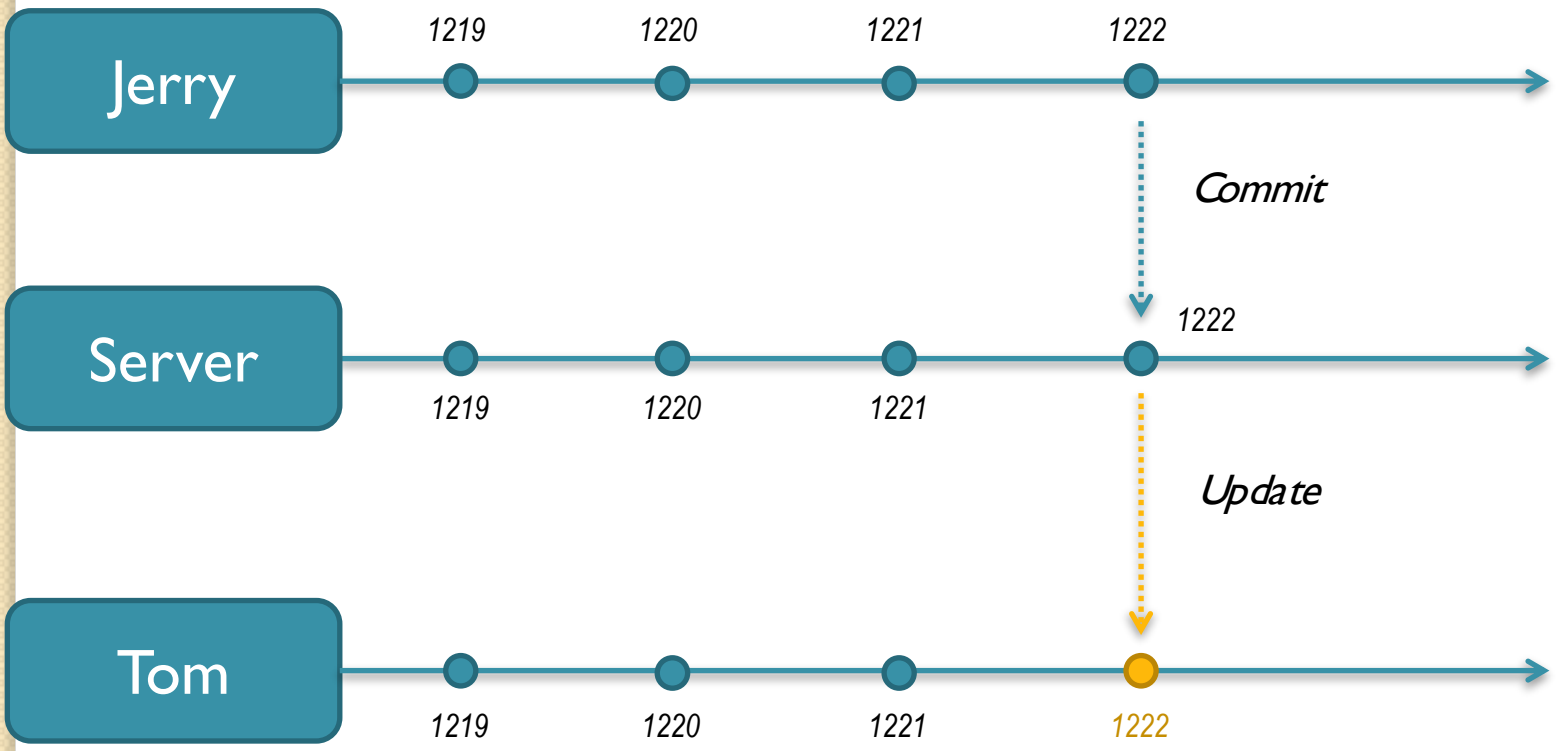
Basic operations

- Add
  - Mark a file or folder to be versioned
- Change
  - Create any changes in the local copy
- Commit
  - Send changes to the repository
- Revert
  - Discard local changes and go back to the same last known revision from the repository
- Update
  - Synchronize changes from the repository to the local copy

Basic operations



Basic operations



Domain vocabulary. CVCS workflow example

WC (working copy)

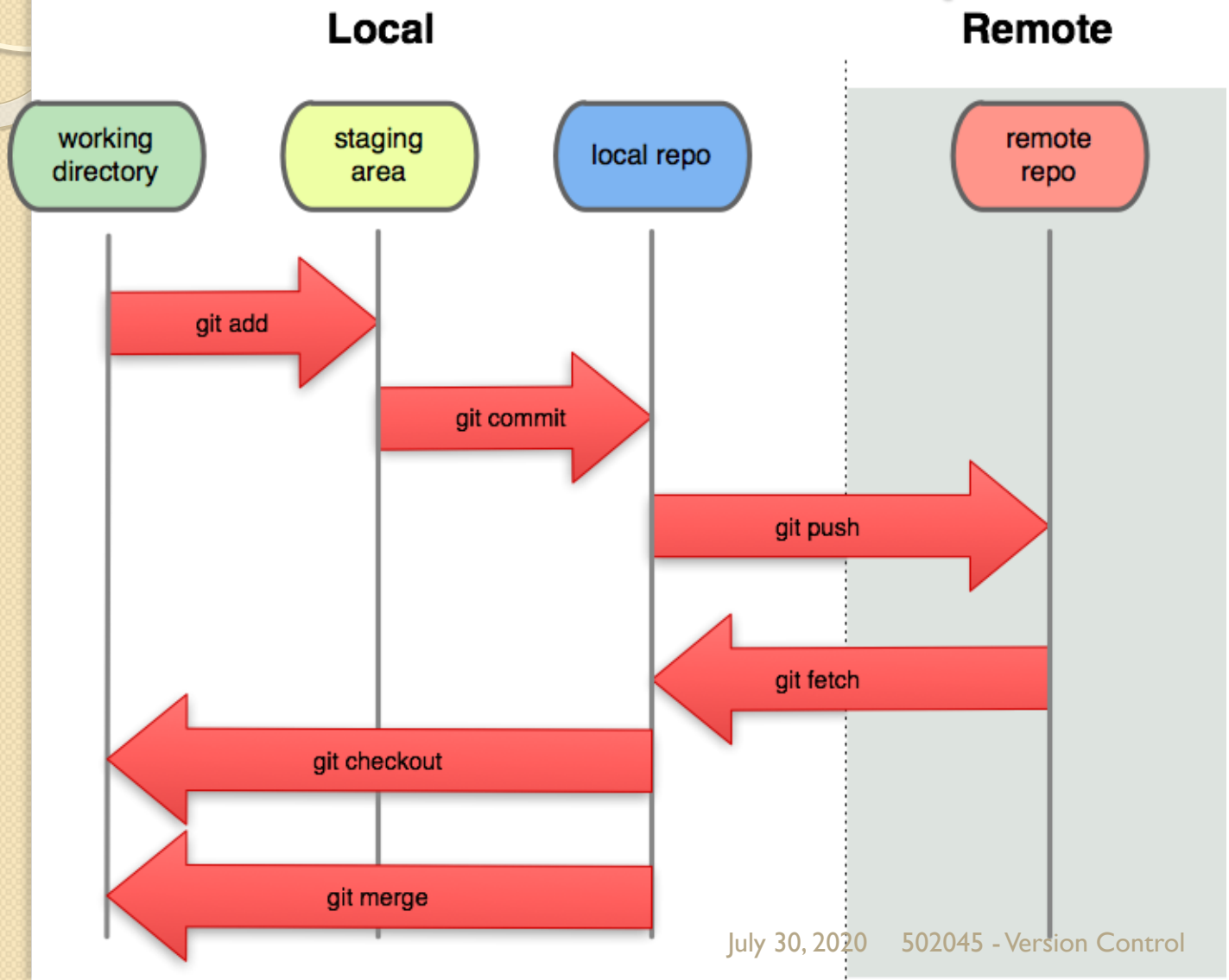
Repository

`svn add`

`svn commit`

`svn update`

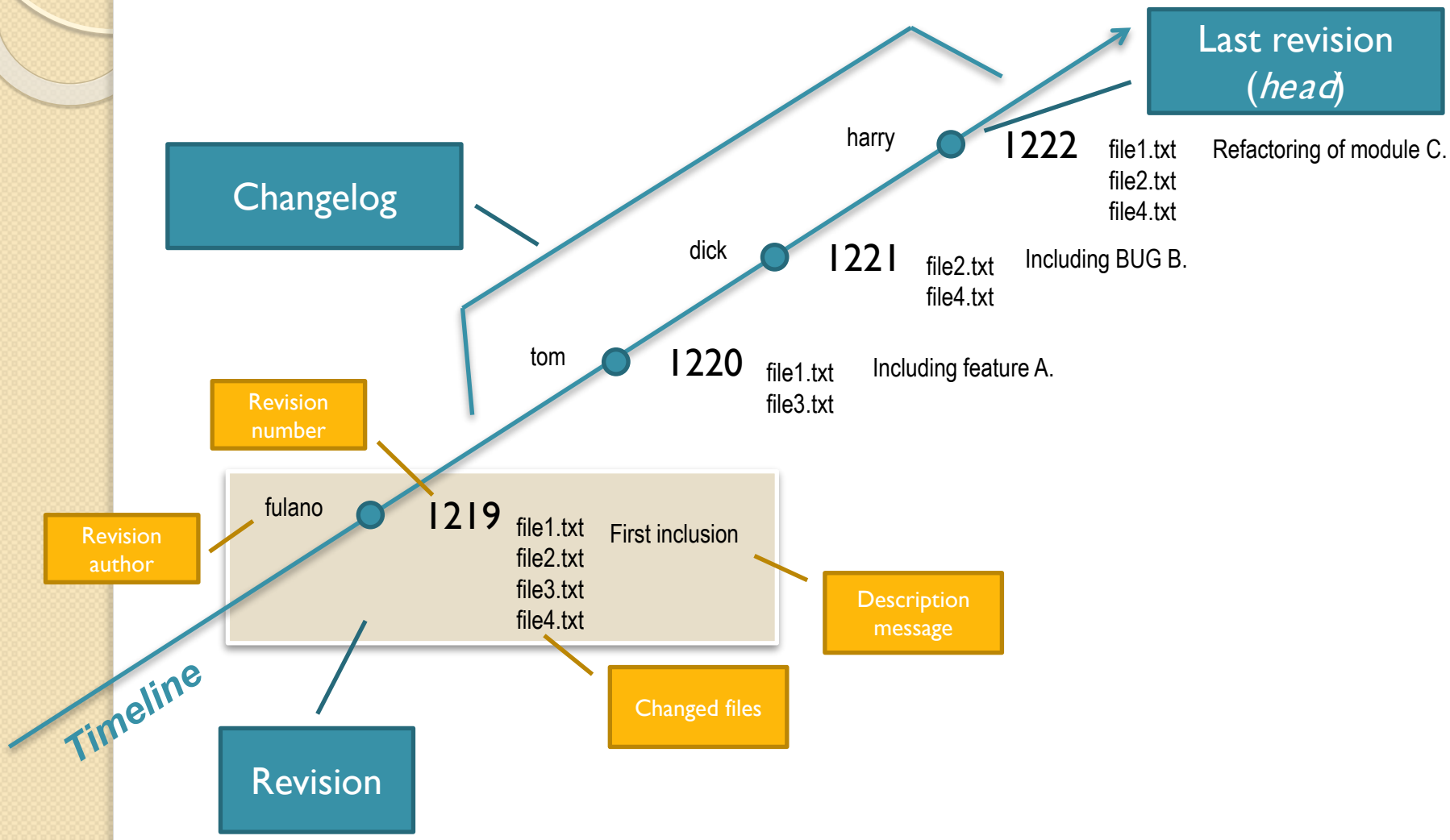
Domain vocabulary. DVCS workflow example



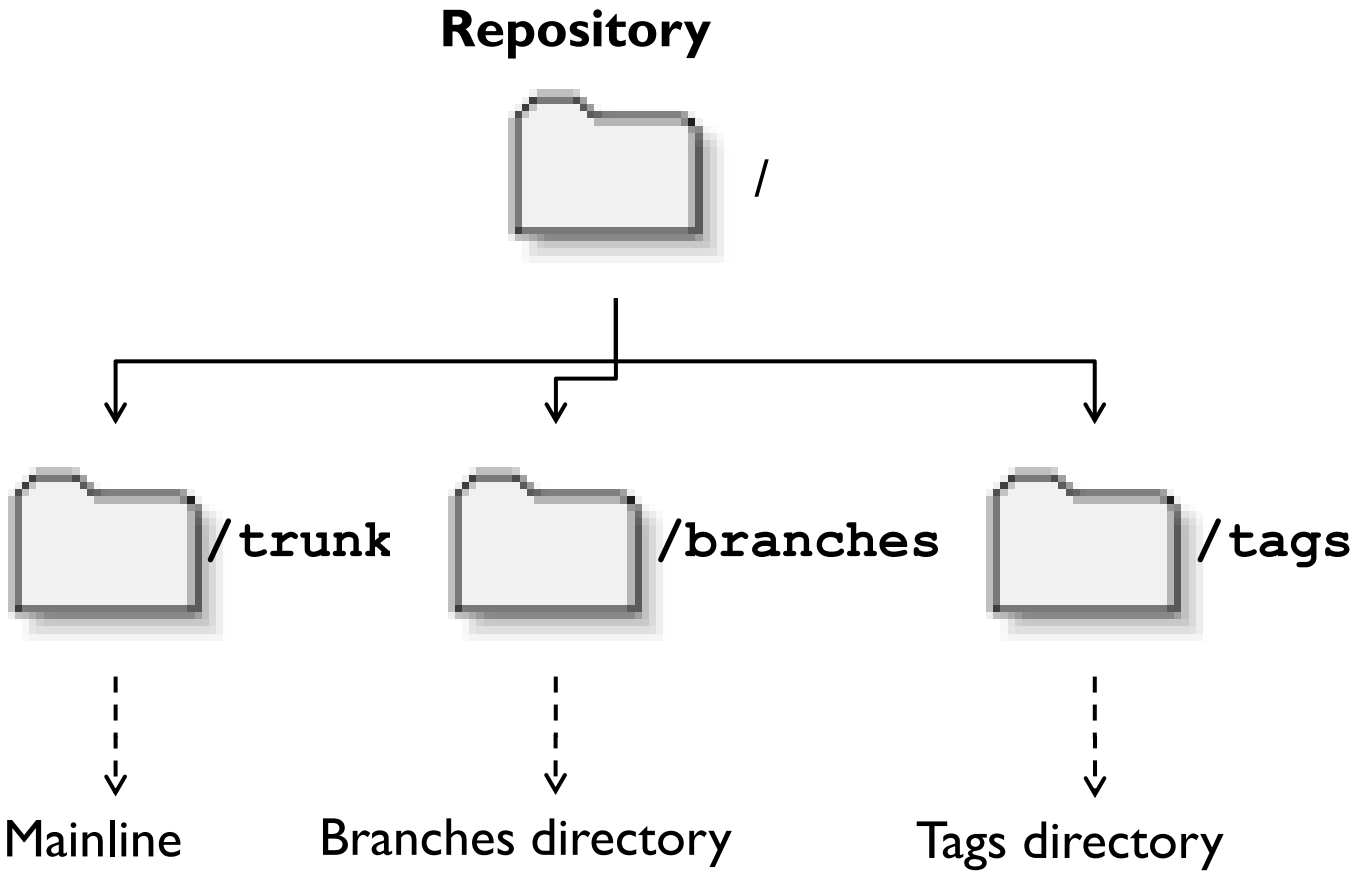
Basic artifacts

- Revision
  - Set of changes, state of the code in a point of time
- Changelog
  - Sequential view of the changes done to the code, stored in the repository

Revision


Last revision
(head)
Changelog
harry
1222
file1.txt
file2.txt
file4.txt
Refactoring of module C.
dick
1221
file2.txt
file4.txt
Including BUG B.
tom
1220
file1.txt
file3.txt
Including feature A.
Revision number
fulano
1219
file1.txt
file2.txt
file3.txt
file4.txt
First inclusion
Revision author
Description message
Changed files
Timeline
Revision

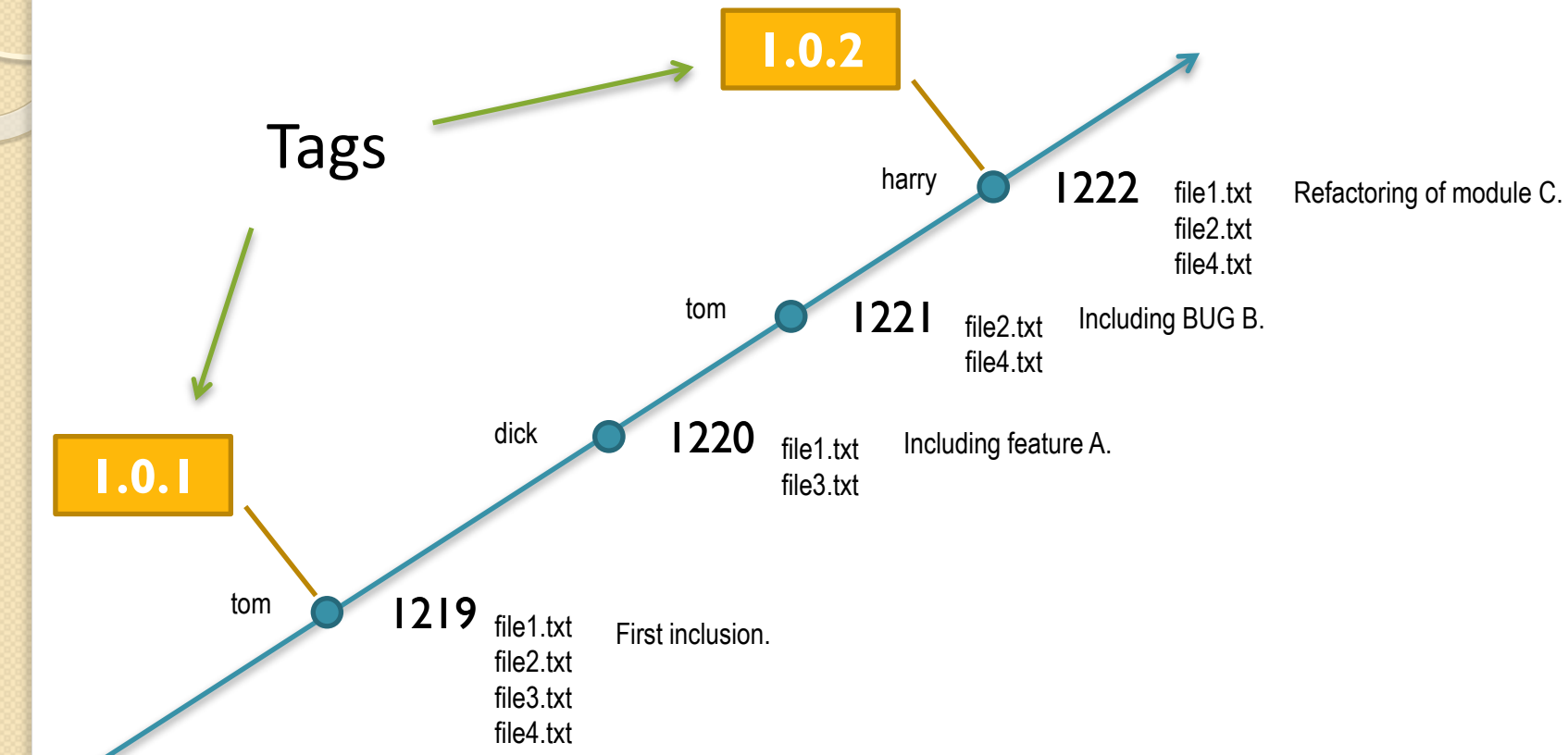
Domain vocabulary. Repository layout



Labels

- Tags
  - Comprehensive alias for existing revisions, marking an important snapshot in time

Labels


1.0.2
Tags
harry
1222
file1.txt
file2.txt
file4.txt
Refactoring of module C.
tom
1221
file2.txt
file4.txt
Including BUG B.
dick
1220
file1.txt
file3.txt
Including feature A.
1.0.1
tom
1219
file1.txt
file2.txt
file3.txt
file4.txt
First inclusion.

Changes

- Diffs and Patches
  - The comparison between two text files is done by an application *diff-like* that feeds an application *patch-like* capable of redo the changes from the originating state to the destination state.
  - This operation of computing the differences is normally referred as *diff*, while the "file containing the differences", that could be exchanged in e-mails and other eletronic media, is denominated *patch*.

Changes – Example

original:

```
1  This part of the
2  document has stayed the
3  same from version to
4  version.  It shouldn't
5  be shown if it doesn't
6  change.  Otherwise, that
7  would not be helping to
8  compress the size of the
9  changes.
10
11 This paragraph contains
12 text that is outdated.
13 It will be deleted in the
14 near future.
15
16 It is important to spell
17 check this dokument. On
18 the other hand, a
19 misspelled word isn't
20 the end of the world.
21 Nothing in the rest of
22 this paragraph needs to
23 be changed. Things can
24 be added after it.
```

new:

```
1  This is an important
2  notice! It should
3  therefore be located at
4  the beginning of this
5  document!
6
7  This part of the
8  document has stayed the
9  same from version to
10 version.  It shouldn't
11 be shown if it doesn't
12 change.  Otherwise, that
13 would not be helping to
14 compress anything.
15
16 It is important to spell
17 check this document. On
18 the other hand, a
19 misspelled word isn't
20 the end of the world.
21 Nothing in the rest of
22 this paragraph needs to
23 be changed. Things can
24 be added after it.
25
26 This paragraph contains
27 important new additions
28 to this document.
```

Diff file or patch

The command **`diff original new`** produces the following *normal diff output*:

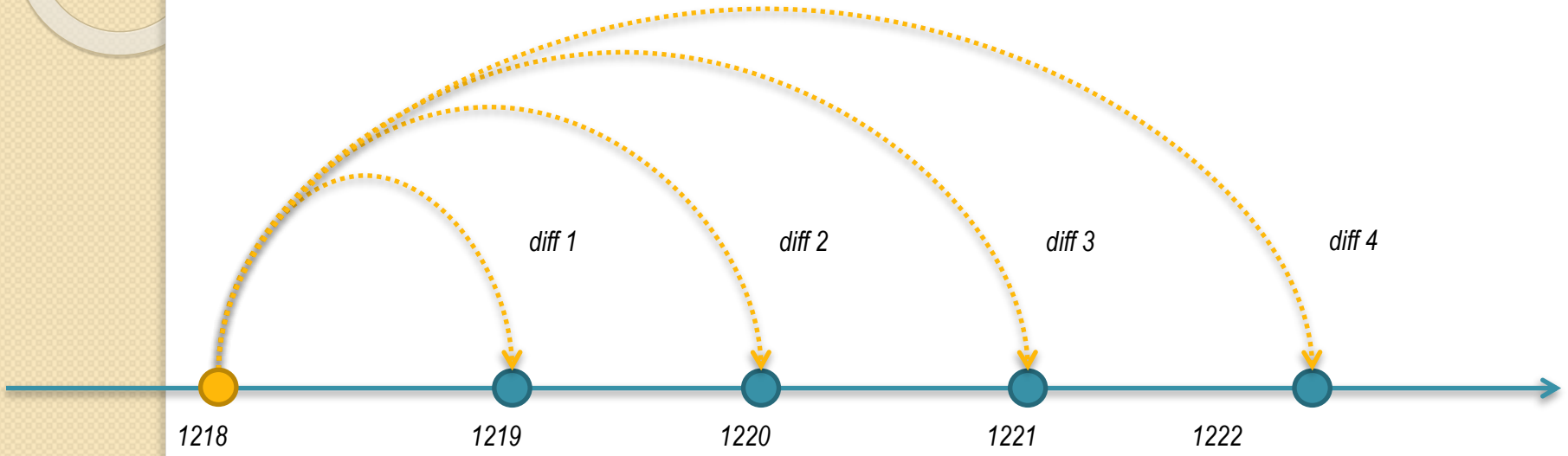
```
0a1,6
> This is an important
> notice! It should
> therefore be located at
> the beginning of this
> document!
>
8,14c14
< compress the size of the
< changes.
<
< This paragraph contains
< text that is outdated.
< It will be deleted in the
< near future.
---
> compress anything.
17c17
< check this dokument. On
---
> check this document. On
24c24,28
< be added after it.
---
> be added after it.
>
> This paragraph contains
> important new additions
> to this document.
```

Diff file or patch

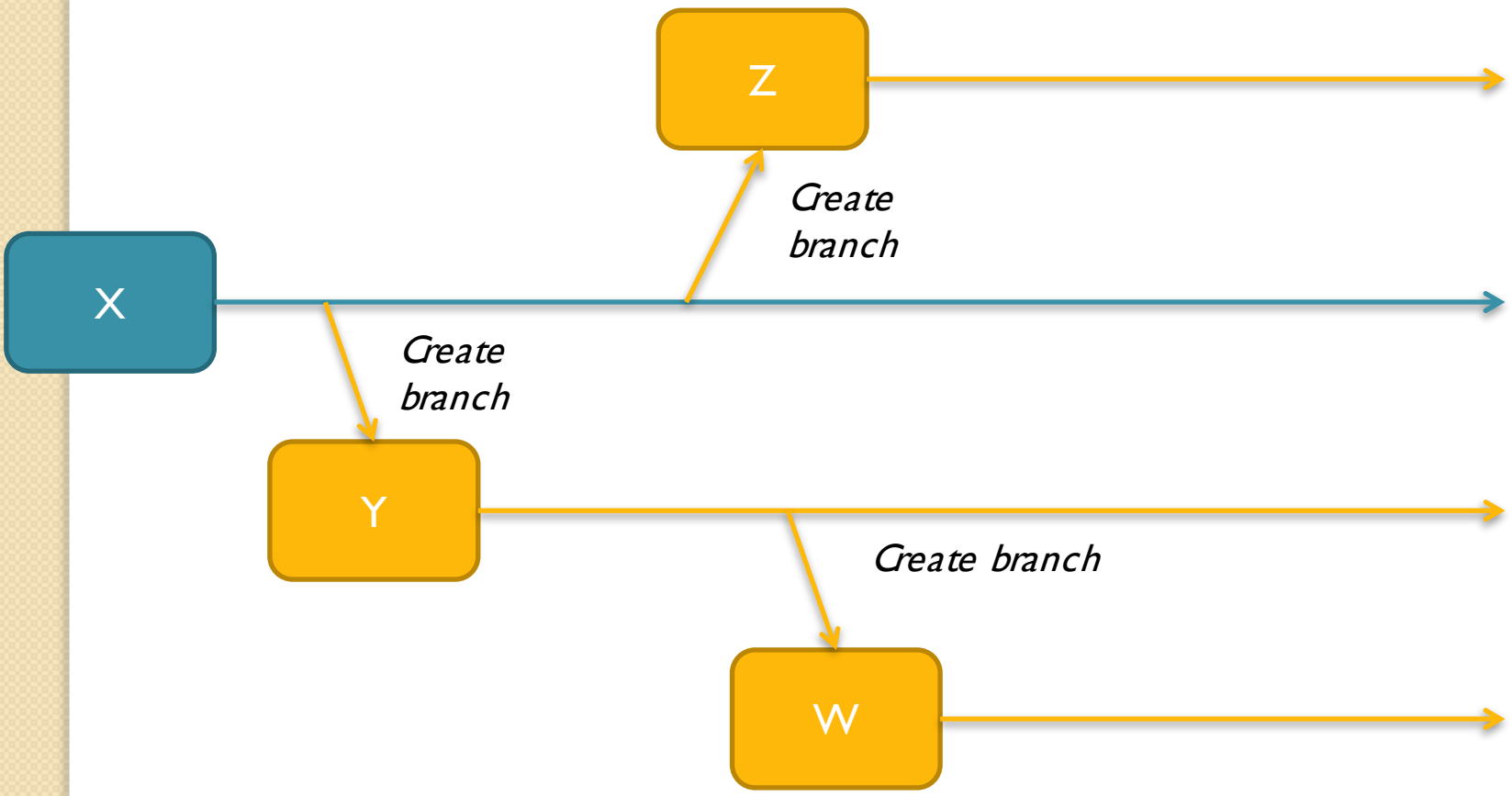
The command `diff -c original new` produces the following output:

```
*** /path/to/original ''timestamp''
--- /path/to/new      ''timestamp''
***************
*** 1,3 ****
--- 1,9 ----
+ This is an important
+ notice! It should
+ therefore be located at
+ the beginning of this
+ document!
+
  This part of the
  document has stayed the
  same from version to
***************
*** 5,20 ****
  be shown if it doesn't
  change.  Otherwise, that
  would not be helping to
! compress the size of the
! changes.
!
! This paragraph contains
! text that is outdated.
! It will be deleted in the
! near future.

  It is important to spell
! check this dokument. On
  the other hand, a
  misspelled word isn't
  the end of the world.
--- 11,20 ----
  be shown if it doesn't
  change.  Otherwise, that
  would not be helping to
! compress anything.

  It is important to spell
! check this document. On
  the other hand, a
  misspelled word isn't
  the end of the world.
***************
*** 22,24 ****
--- 22,28 ----
  this paragraph needs to
  be changed. Things can
  be added after it.
+
+ This paragraph contains
+ important new additions
+ to this document.
```

Changes



Branches



Merge

- In some point of the development, the code must be merged.

Merge

Conflicts

- When the changes are done in the same source file lines, you can have conflicts.

How to solve conflicts?

ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

Question:

Is it possible to solve these conflicts automatically?

How to solve conflicts?

ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

Answer: No!!!

Even when don't have changes in the same source line, you can still have semantic problems in the code. So, always double check before to commit.

How to solve conflicts?

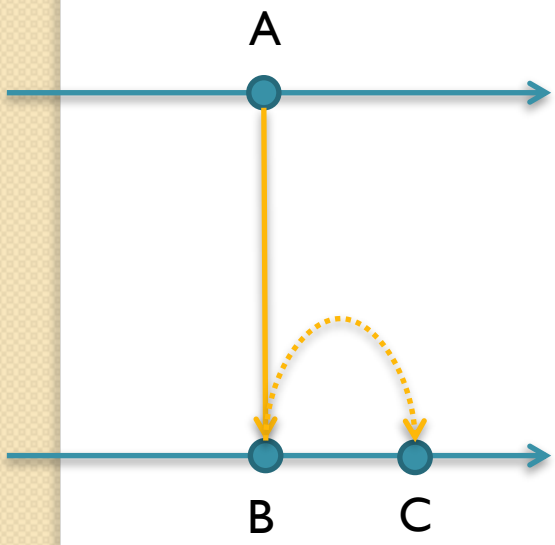
Anote before continuing:

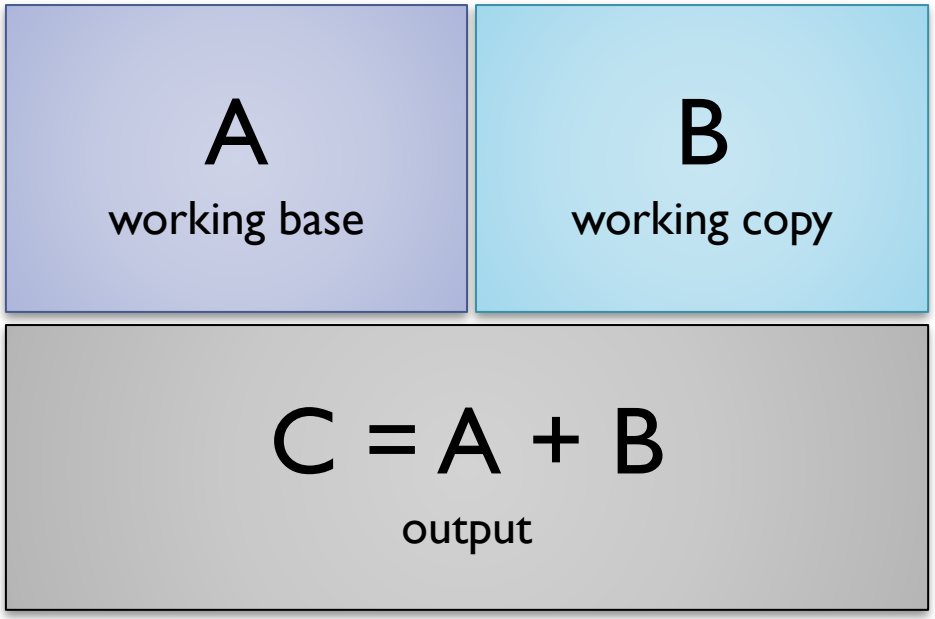
It is always better to avoid unnecessary conflicts.

Communicate your development, always include a description in your committed revisions, ask for help if you don't understand anything, etc.

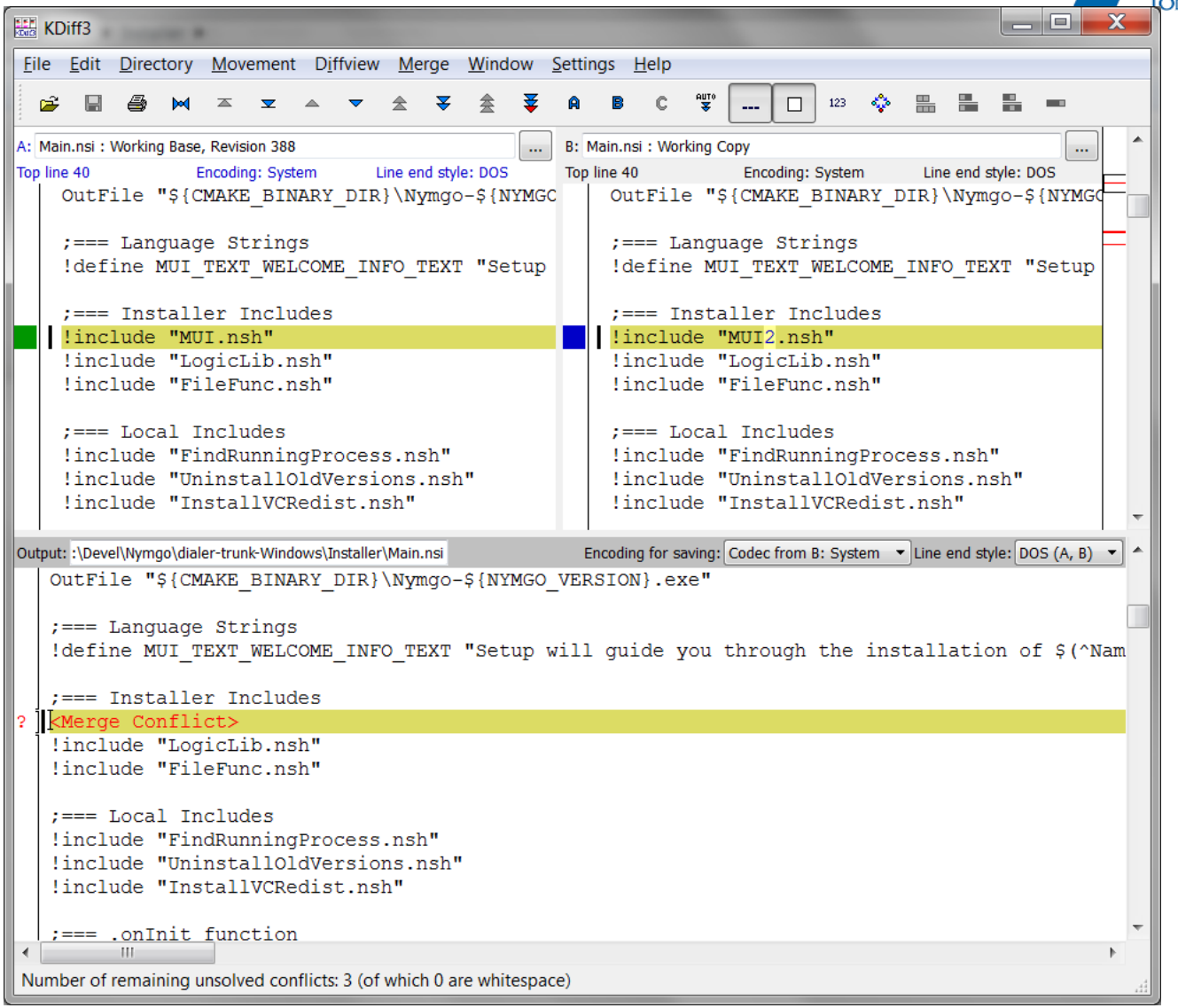
How to solve conflicts?

Two way merge



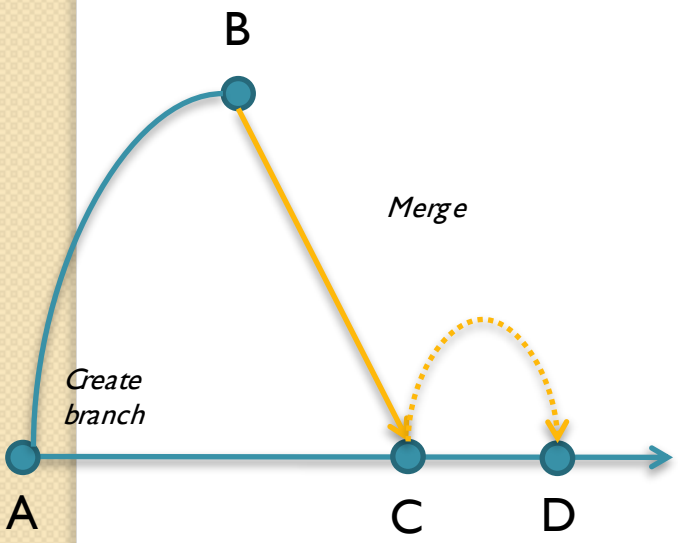


Tools: KDiff3



How to solve conflicts?

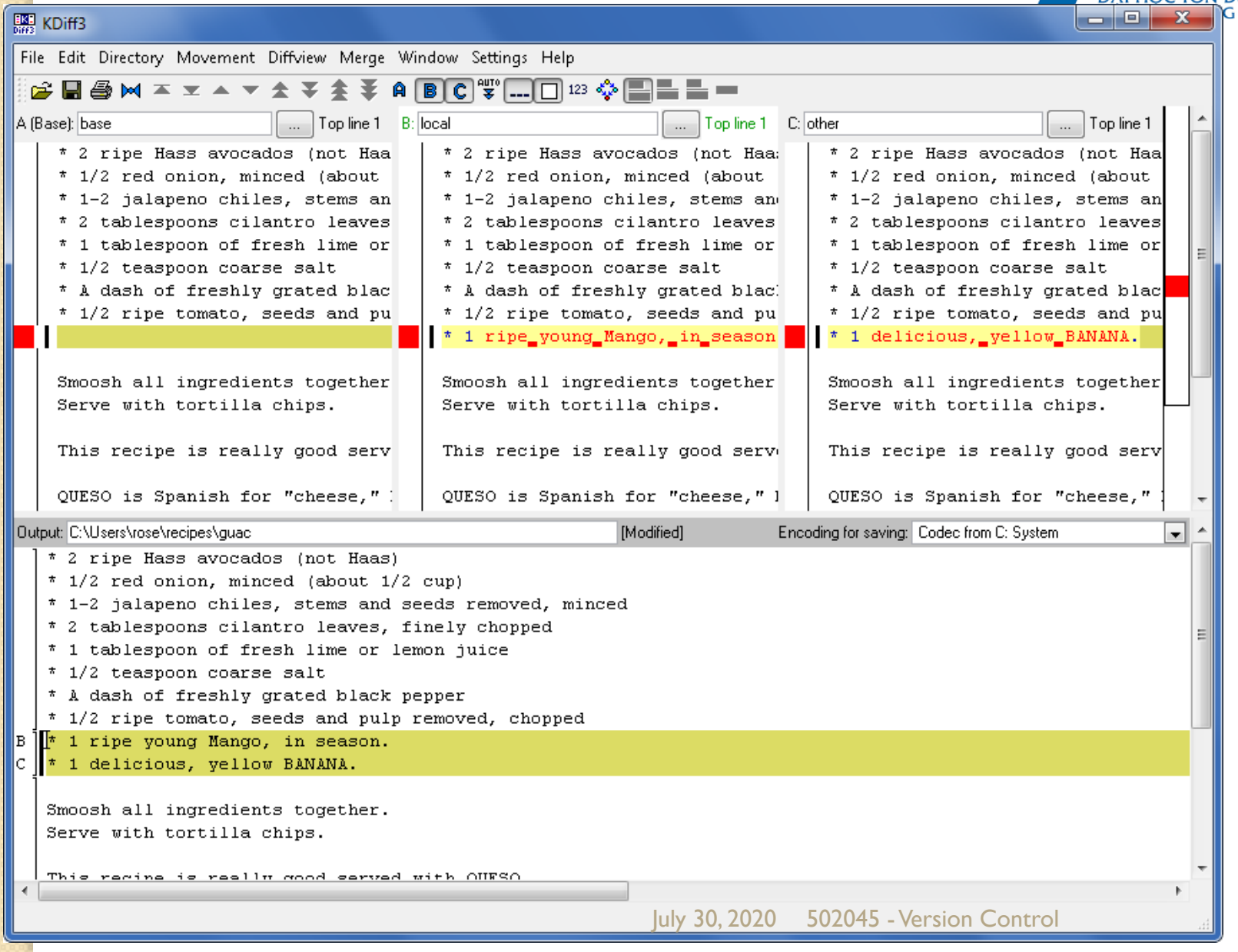
Three way merge



A	B	C
origin	branch #1	branch #2

D = A + B + C

Tools: KDiff3

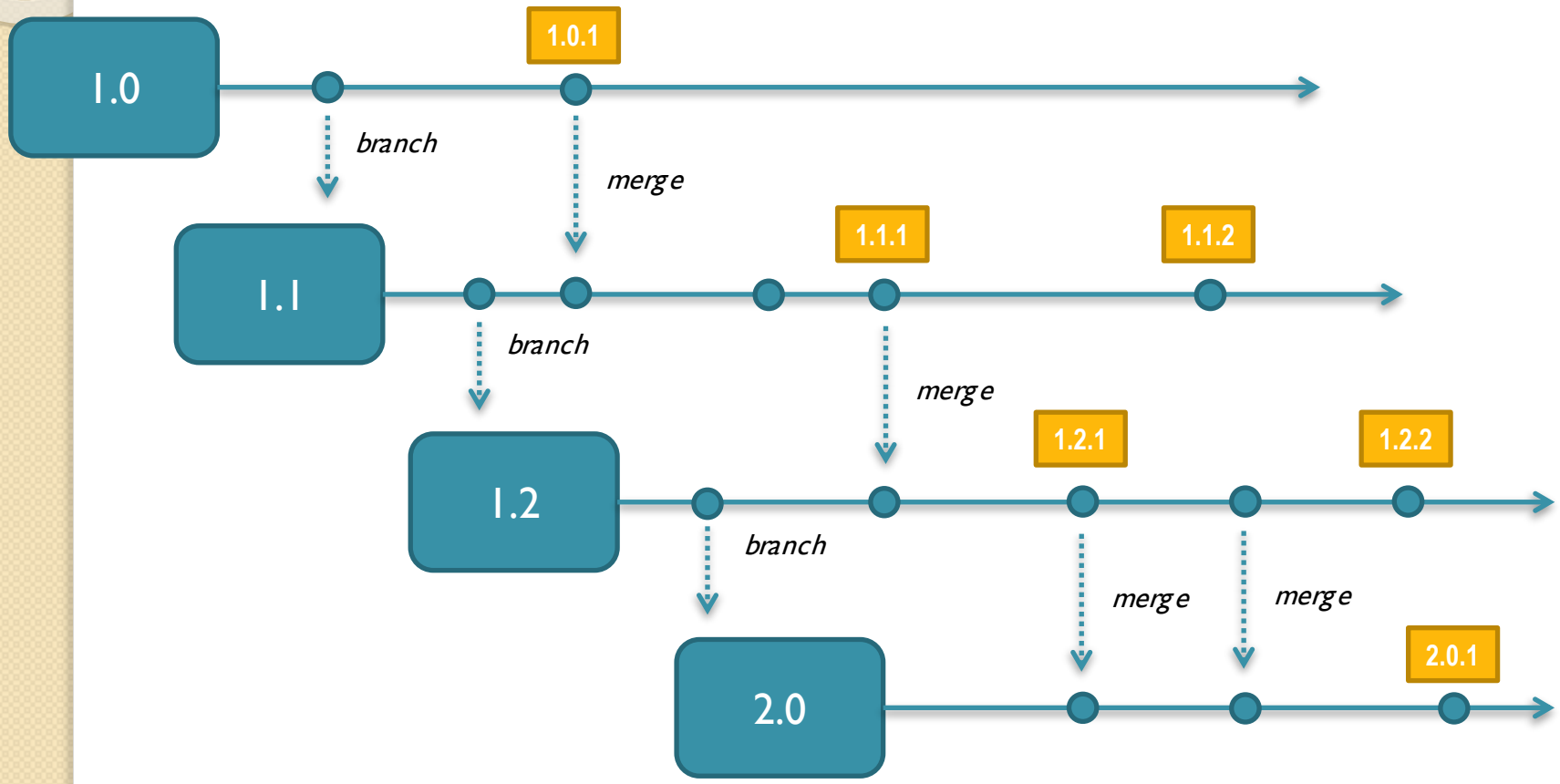


ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

How to manage the branches during the software lifecycle

BRANCHING PATTERNS

Mainline


1.0
1.0.1
branch
merge
1.1
1.1.1
1.1.2
branch
merge
1.2
1.2.1
1.2.2
branch
merge
merge
2.0
2.0.1

Mainline

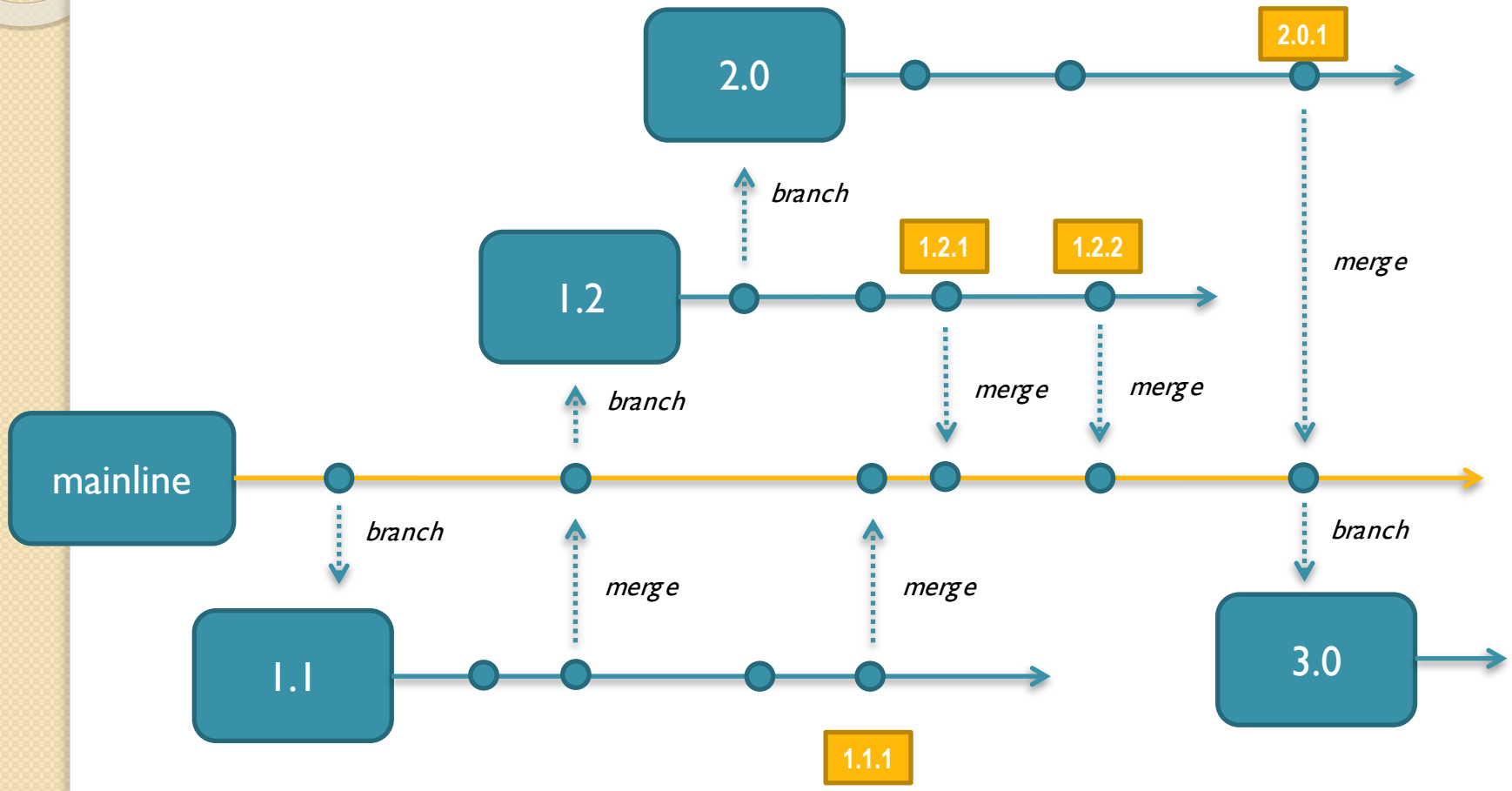
Bring a fix from branch *n* to branch *m* require *m-n* merges (*linear complexity with the number of branches*).

Mainline

Learned lesson:

Perform merges the early and frequently as possible.

Mainline / Variant



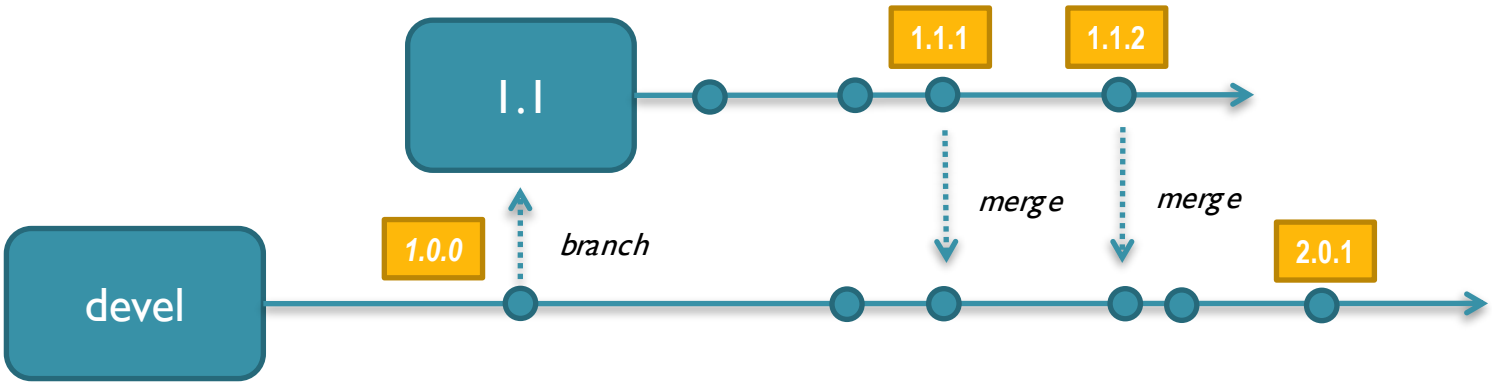
ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

Scenario:

You just developed a stable version of the software and you need to create a new version with new features and still provide small fixes for the last stable version.

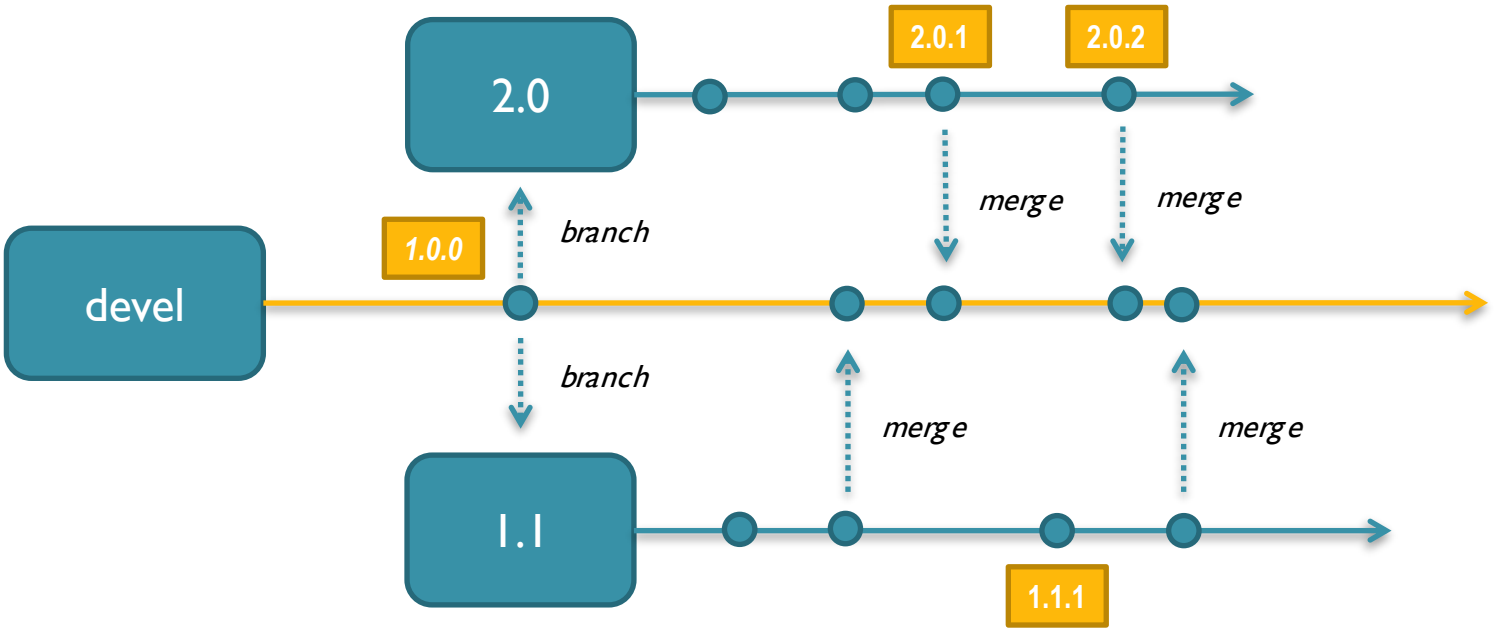
Parallel Maintenance / Development lines

Most used option



Parallel Maintenance / Development Lines

Note the internal branch



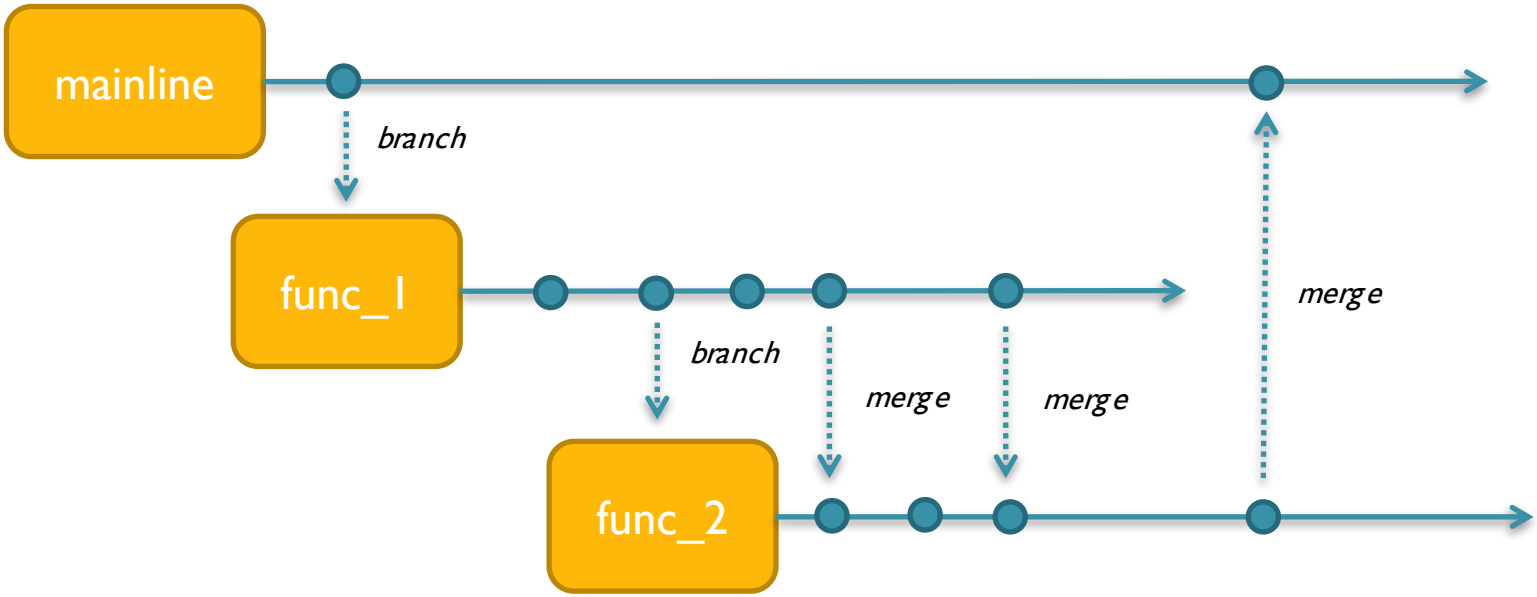
ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

Scenario:

You need to develop two different features in a short period of time.

(laughts)

Overlapping Release Lines

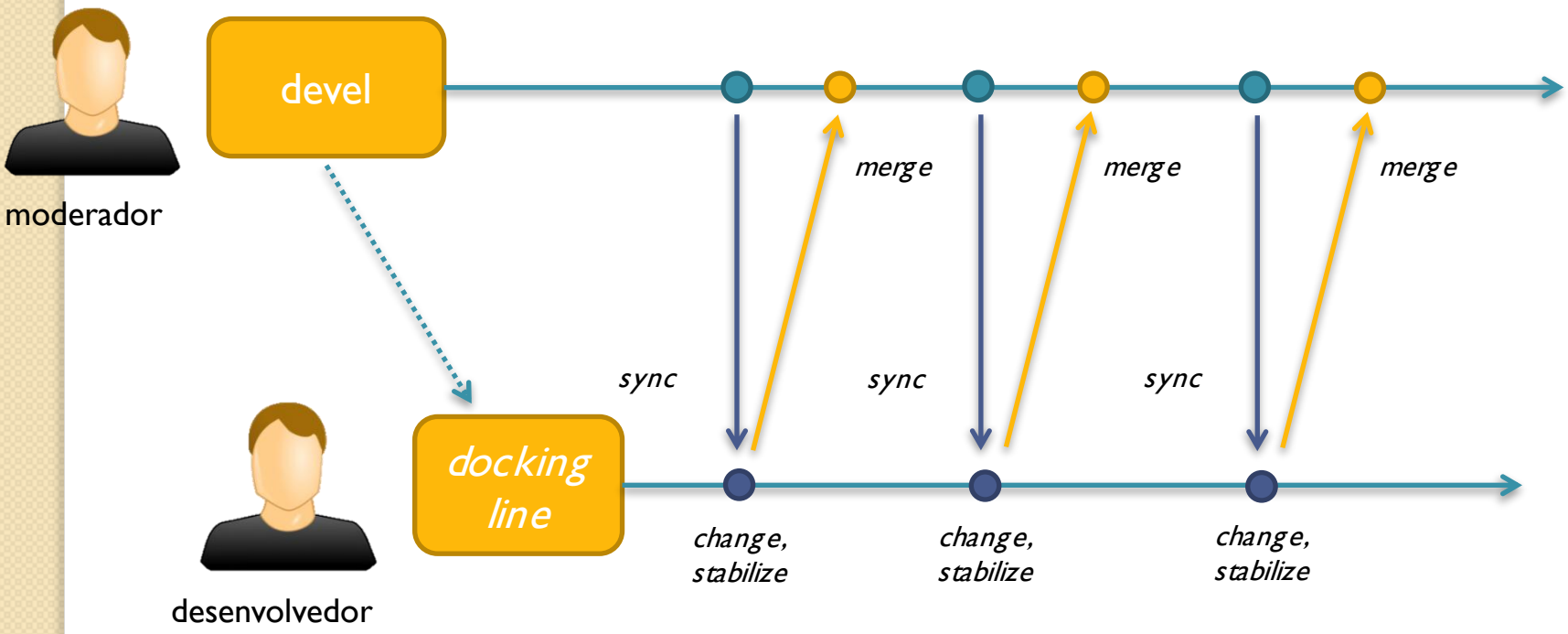


ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

Scenario:

You allocated a new developer to work in a too risky code base and you want to moderate his/her changes for a while.

Docking Line

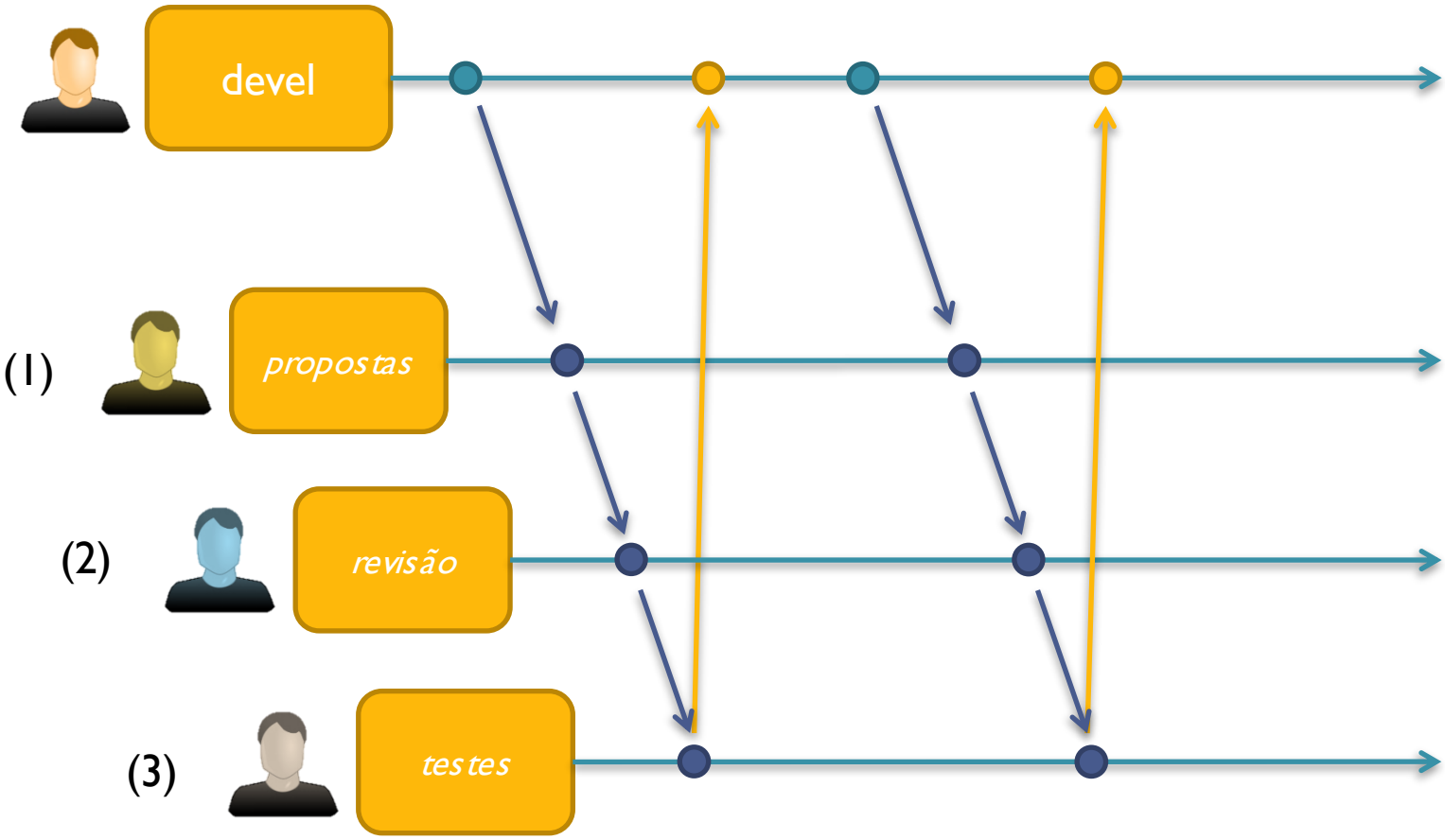


ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

Scenario:

Your development tasks need to progress in discrete levels of maturity: (1) change proposals, (2) analysis, (3) review, (4) unit tests, (5) integration tests, (6) system tests, etc.

Staged Integration Lines

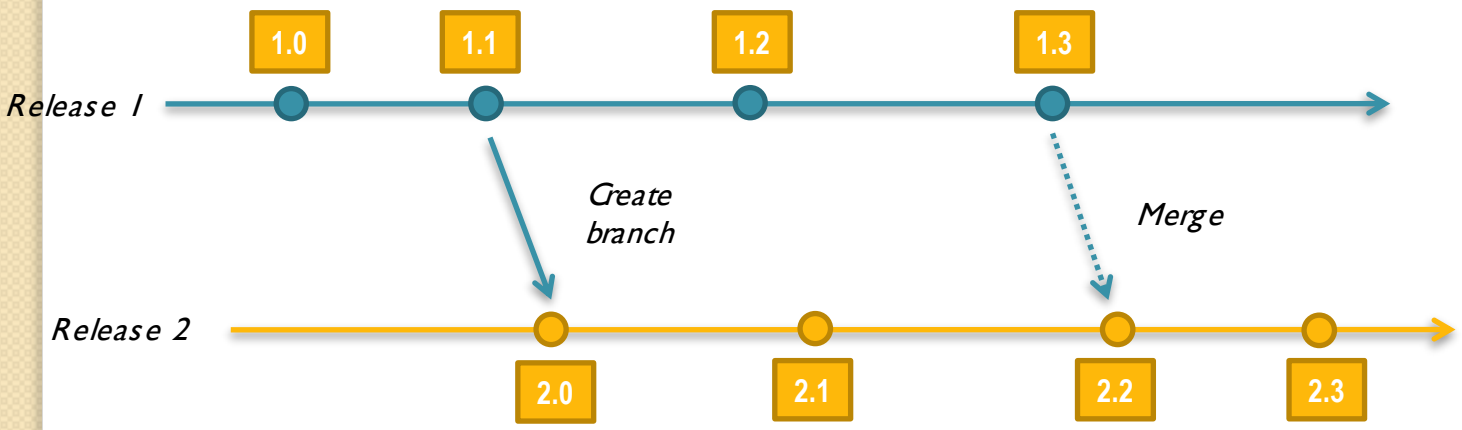


ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

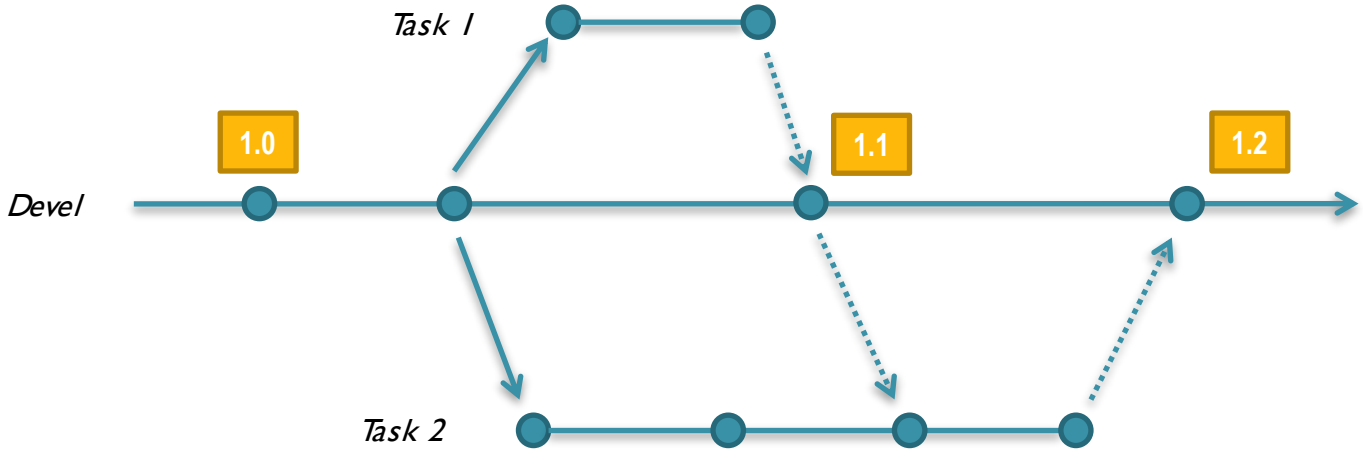
Some most common patterns.

WHEN TO CREATE A BRANCH?

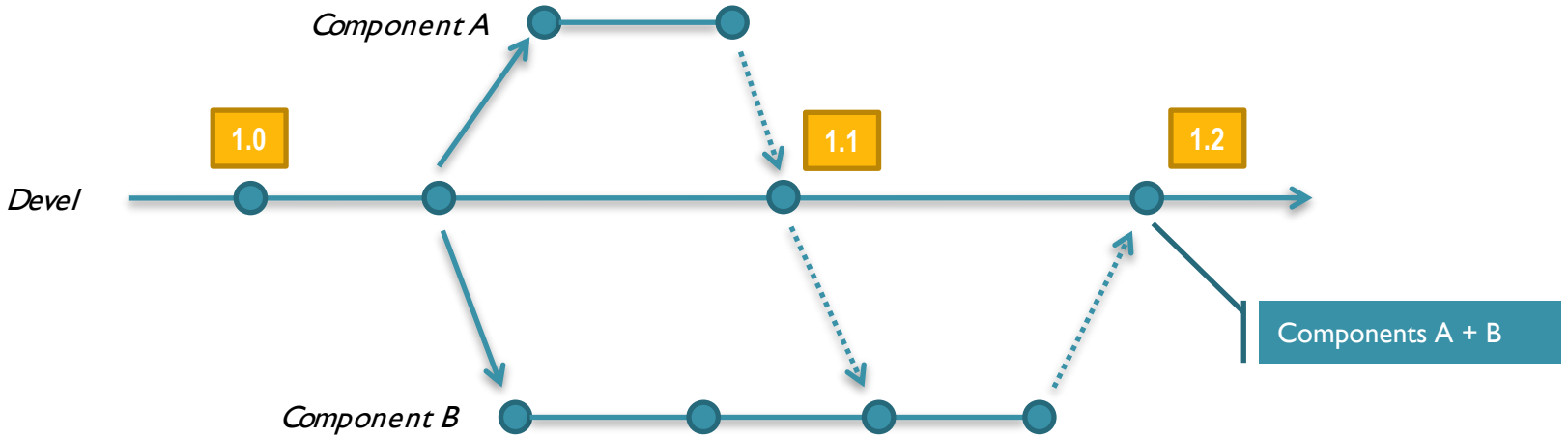
Branch per Delivery



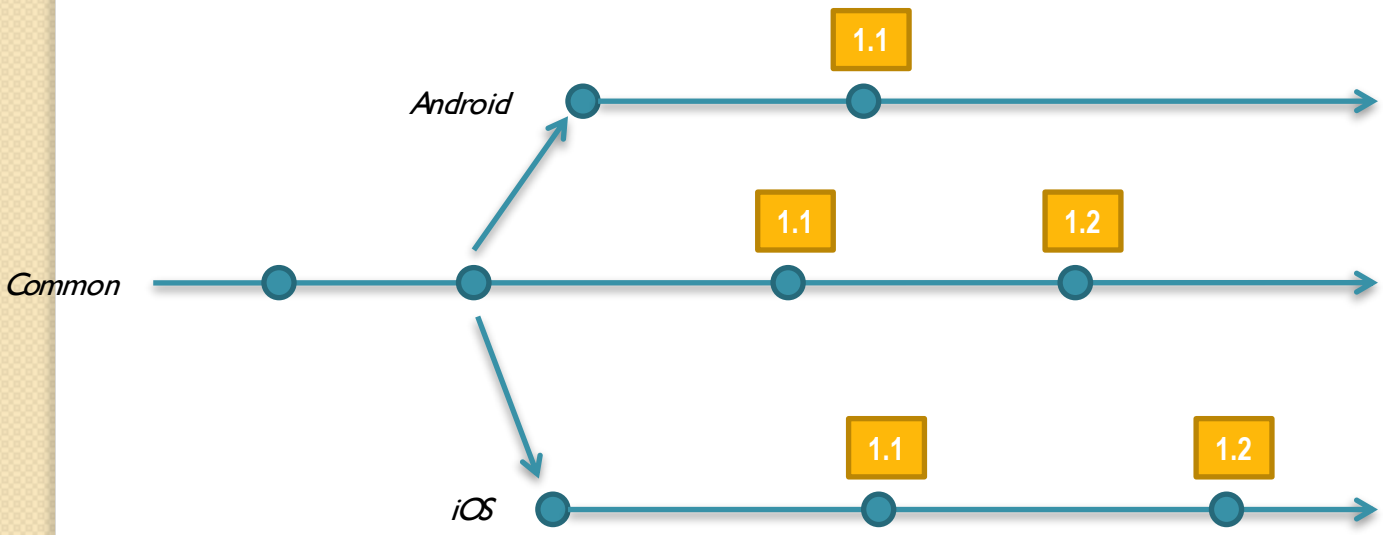
Branch per Task



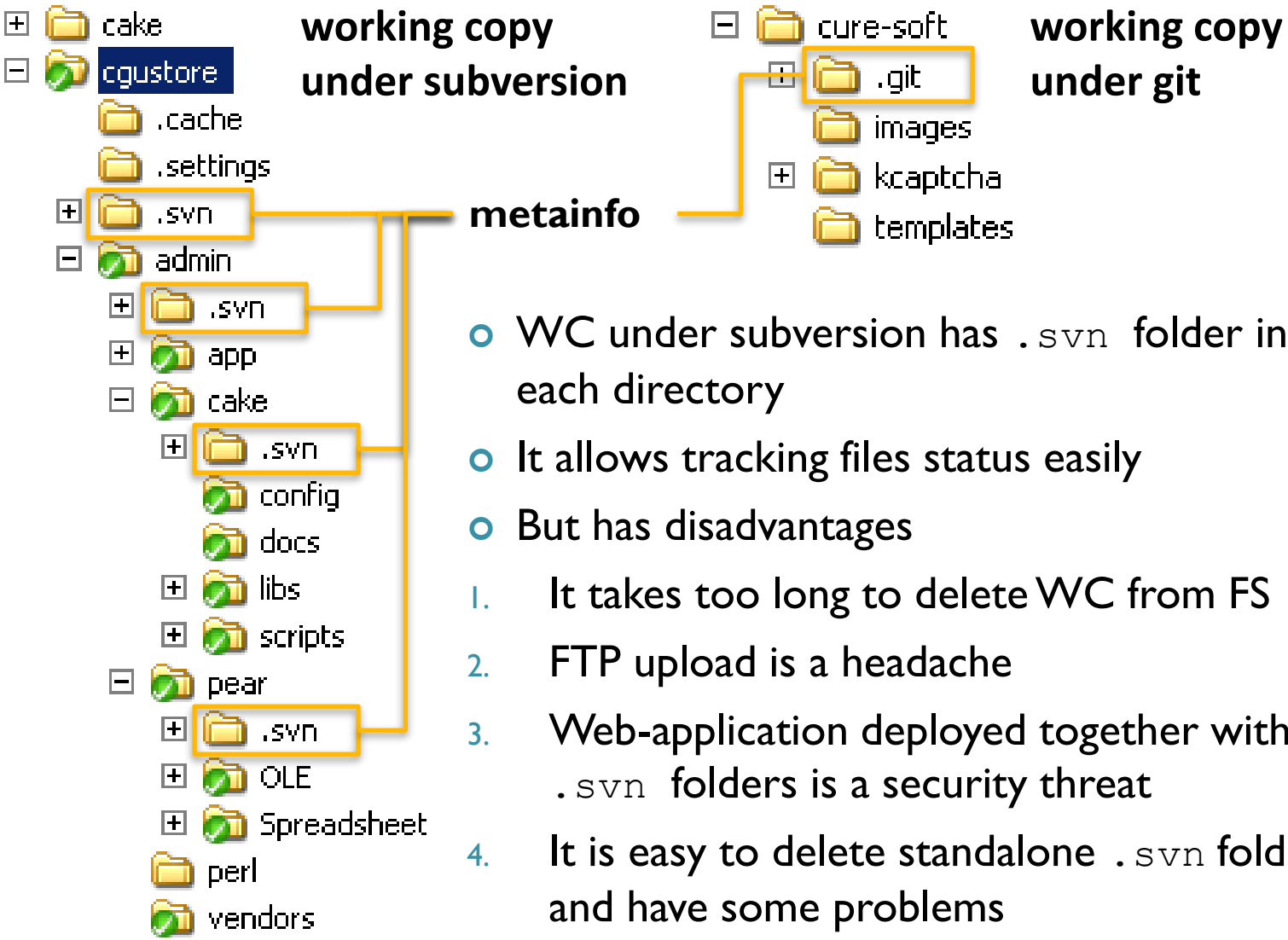
Branch per Component



Branch per Technology



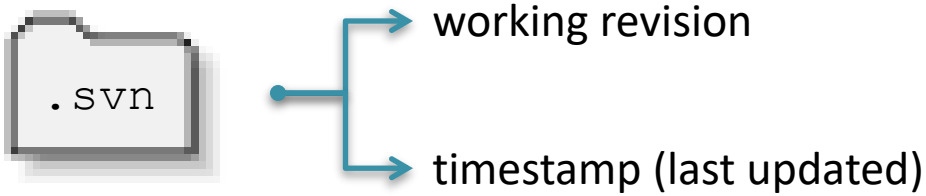
Domain vocabulary - Metainfo



- WC under subversion has `.svn` folder in each directory
- It allows tracking files status easily
- But has disadvantages

1. It takes too long to delete WC from FS
2. FTP upload is a headache
3. Web-application deployed together with `.svn` folders is a security threat
4. It is easy to delete standalone `.svn` folder and have some problems

Working copy file status



Unchanged, and current

Unchanged, and out of date

Locally changed, and current

Locally changed, and out of date