

AN INTRODUCTION TO VCS

OBJECTIVES

- At the end of this course, the learner should understand
 - What Version Control is all about and what problems it solves
 - Subversion (SVN)
 - GIT
 - Branching and Merging

WHY VERSION CONTROL?

- Scenario 1:
 - Your program is working
 - You change “just one thing”
 - Your program breaks
 - You change it back
 - Your program is still broken--*why?*
- Has this ever happened to you?



WHY VERSION CONTROL? (PART 2)

- Your program worked well enough yesterday
- You made a lot of improvements last night...
 - ...but you haven't gotten them to work yet
- You need to turn in your program *now*
- Has this ever happened to you?



VERSION CONTROL FOR TEAMS

- Scenario:
 - You change one part of a program--it works
 - Your co-worker changes another part--it works
 - You put them together--it doesn't work
 - Some change in one part must have broken something in the other part
 - What were all the changes?



TEAMS (PART 2)

- Scenario:
 - You make a number of improvements to a class
 - Your co-worker makes a number of *different* improvements to the *same* class
- How can you merge these changes?



VERSION CONTROL SYSTEMS

- A version control system (often called a source code control system) does these things:
 - Keeps multiple (older and newer) versions of everything (not just source code)
 - Requests comments regarding every change
 - Allows “check in” and “check out” of files so you know which files someone else is working on
 - Displays differences between versions



WHY DO WE CARE

- Software is written:
 - As a combination of modules
 - In one or more languages
 - For one or more applications
 - Using one or more libraries
- Things that may (will) change over time:
 - The modules required and used
 - The languages employed
 - The application requirements
 - The libraries
- Each change could cause the system to break.

YOU KNOW YOU HAVE A CM PROBLEM WHEN...

- “But the system was working fine yesterday! What happened?”
- “But I can’t reproduce your bug in my copy of the system!”
- “But I already corrected that code last week! What happened to my fix?”
- “I wonder if that bug was fixed in this copy too?”

CM DEFINED

- Configuration management attempts to identify and track all relevant elements of the configuration of a system, so that all possible errors can be identified, and possible solutions to the problems found.
- Version control is a special case of configuration management, where we are concerned with maintaining multiple versions of a software system.
- Defect tracking systems monitor the status of error reports from first identification to resolution. Defect tracking relates to CM because defects occur as a result of changes, and defect removal typically requires a change in configuration.

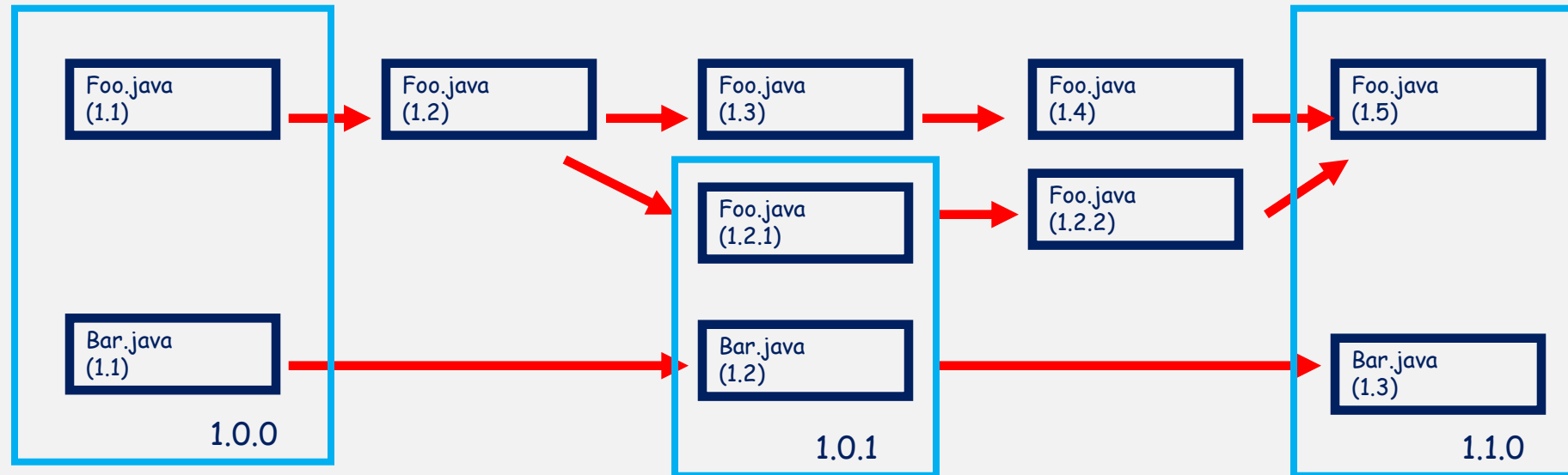
THE THREE CLASSIC CM PROBLEMS

- Any configuration management approach must address at least the following “classic” problems:
- **The double maintenance problem**
 - Must prevent occurrence of multiple copies of the same file that must be independently updated.
- **The shared data problem**
 - Must allow two or more developers to access the same file/data.
- **The simultaneous update problem**
 - Must prevent “clobbering” when two developers update the same file at the same time.
 - “clobbering”: only the second developer’s changes survive.

VERSIONS VS. CONFIGURATIONS (TRADITIONAL)

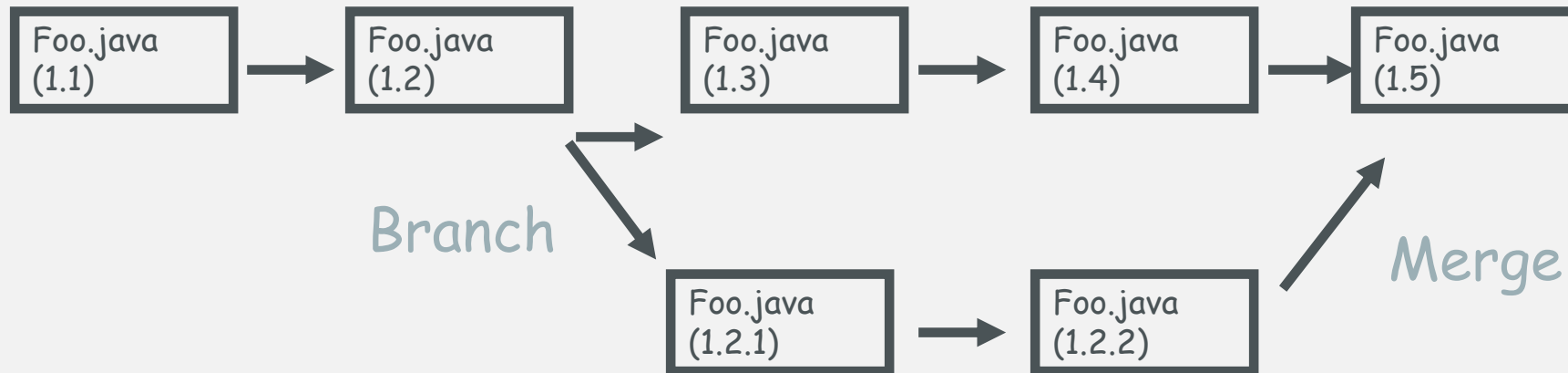
- Files exist in multiple **versions**
 - Sequences of updates over time
 - Parallel variants
- Systems exist in multiple **configurations**
 - For different platforms
 - For different customers
 - For different functionality/pricing levels

VERSIONS VS. CONFIGURATIONS (TRADITIONAL)



VERSION CONTROL

- Version control systems support:
 - Multiple versions of a file
 - Multiple paths of file revision
 - Locking to prevent two people from modifying the same file at the same time
 - Recovery of any prior version of a file
 - Efficient storage via “deltas” (forward or backward)

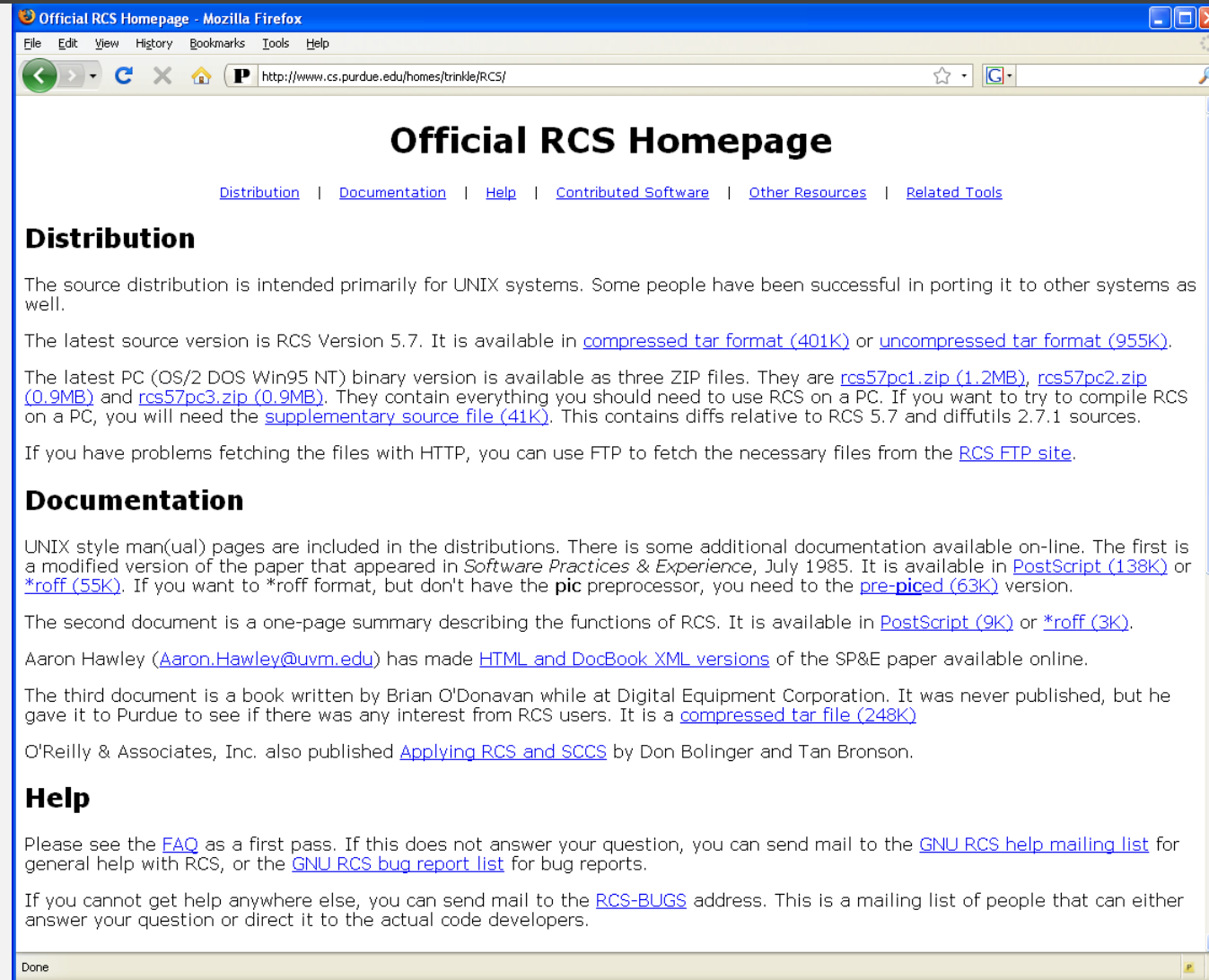


SCCS

- **SCCS** is Source Code Control System (UNIX)
- **SCCS** keeps multiple versions of a complete directory
- Storage requirements are small, because **SCCS**:
 - keeps the *original* documents
 - keeps the *changes* needed to go from one version to the next
 - *generates* any version when you ask for it



RCS



RCS COMMANDS

- Create a directory for your **RCS** files
- `co -l file -- check out a file and lock it`
 - Locking means you can check the file back in
- `ci file -- check in a revision (put file under rcs control)`
- `rcs -l file -- lock a file you already checked out`
 - (Needed when you checked it out and forgot the `-l`)
- `rcsdiff files -- report differences between files`
- `merge files -- merges two files into original file`
 - Not magic--you have to check the results



EXAMPLE RCS COMMANDS

- `% ci foo.java`
 - Submit foo.java for version control, or submit updated version.
 - Creates RCS/foo.java,v
 - Deletes foo.java
 - Prompts you to supply a string documenting updates.
- `% co foo.java`
 - Obtain read only copy of latest version of foo.java.
- `% co -l foo.java`
 - Request a lock so file is read/write
- `% rcsmerge -r1.4 -r1.2.2 foo.java`
 - Merge multiple versions into one file
- `% rcsdiff -r1.4 -r1.2.2 foo.java`
 - See differences between versions 1.4 and 1.2.2

RCS: REVISION CONTROL SYSTEM

- When foo.java is put under RCS control, a new file called RCS/foo.java,v is created that represents:
 - The most recent version of foo.java
 - Backward deltas allowing reconstruction of any older version of the file in space-efficient manner.
 - Information on who has a lock on the file (if anyone)
- For details, download, etc:
 - <http://www.gnu.org/software/rcs/>

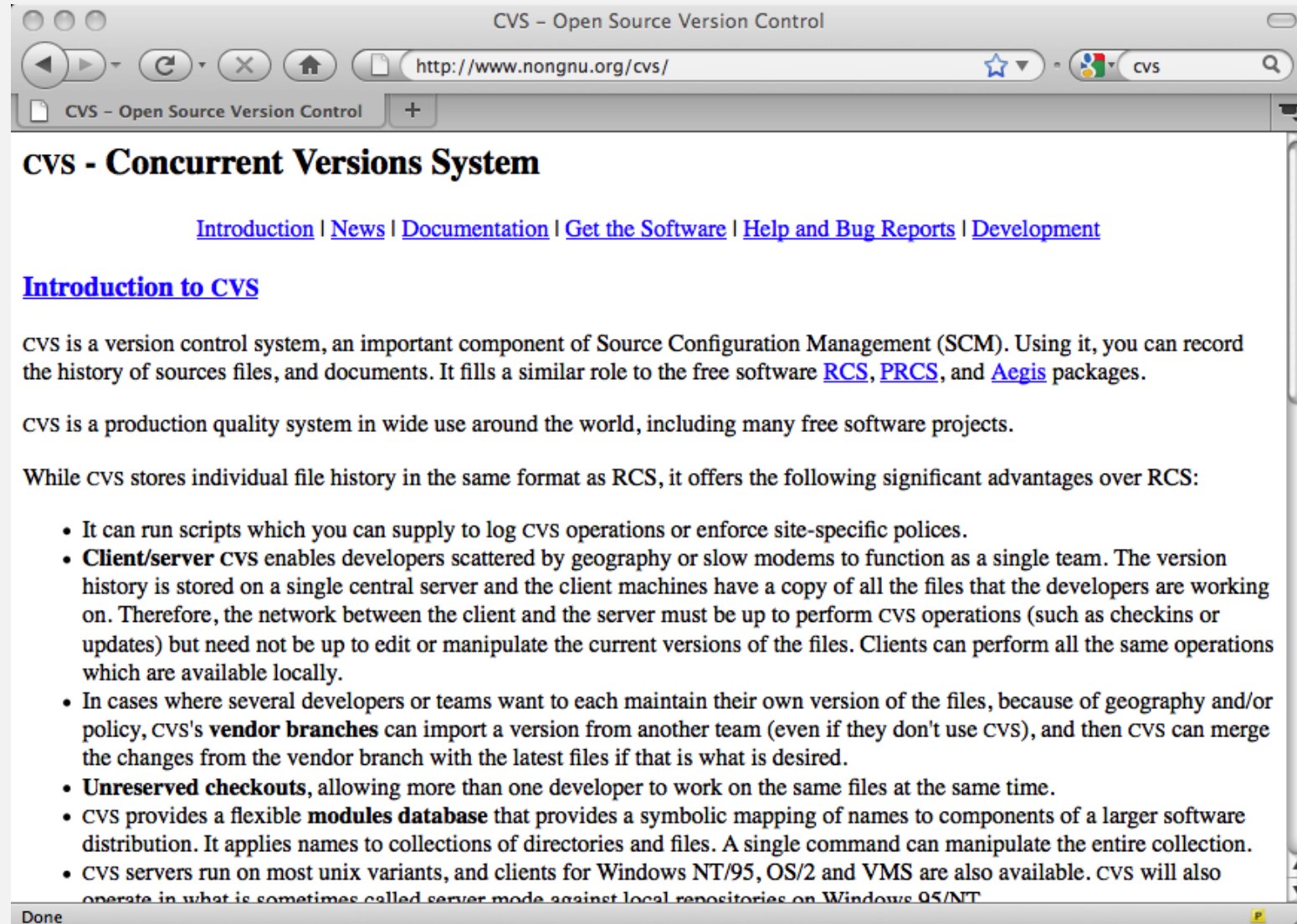
RCS AND THE 3 CM PROBLEMS

- The double maintenance problem
 - RCS provides a repository with a single “master copy” of each file.
- The shared data problem
 - Multiple developers can access the directory containing the checked out version of the file.
- The simultaneous update problem
 - To edit a file, a developer must obtain a lock.
 - RCS changes file permissions/ownership to implement the lock.

RCS SUMMARY

- Free, open source, former “industry standard”
- Easy to install and setup.
- Non-GUI, unix-based, command line interface.
- Provides version control, not configuration management.
- All developers must access the same (Unix) file system to solve double maintenance and shared data problems.
- Locking is “pessimistic”
 - Leads to many emails with “Please release lock on file foo.java.”

CVS



CVS – CONCURRENT VERSION SYSTEMS

- **CVS** (Concurrent Versions System)
- **CVS** is built on top of rcs
- **CVS** is built into JBuilder and Eclipse
- See the below for additional information on CVS/Windows
-- <http://www.cvshome.org/dev/codewindow.html>



CVS COMMANDS

- **cv**s checkout *file*
- **cv**s commit *file*
- **cv**s diff *file*
- These are CVS commands under UNIX
- Popular GUIs on Windows:
 - tkCVS
 - jCVS
 - WinCVS



CVS: CONCURRENT VERSIONS SYSTEM

- Uses RCS as backend for version control, adds configuration management support.
- Client-server architecture enables:
 - checkin and checkout over Internet
 - Developers do not need to access single (Unix) file system
- “Optimistic” locking
 - Multiple developers can modify same file simultaneously.
 - At checkin, any “merge conflicts” are detected and given to developers to resolve.
- Versions and configurations still on a “per-file” basis
 - SVN will take a different approach.

BASIC STEPS FOR CVS

- Administrator sets up CVS server and provides developers with CVS accounts and passwords.
- Developers configure their workstations with CVS repository location, account, and password information.
- Basic development cycle:
 - Check out a copy of system to your computer.
 - Edit files locally.
 - Commit your changes to repository when ready.
 - Update your local code tree to synchronize it with repository when desired.
 - Address merge conflicts as they occur.
 - Tag a set of file versions to create a configuration when ready to release.

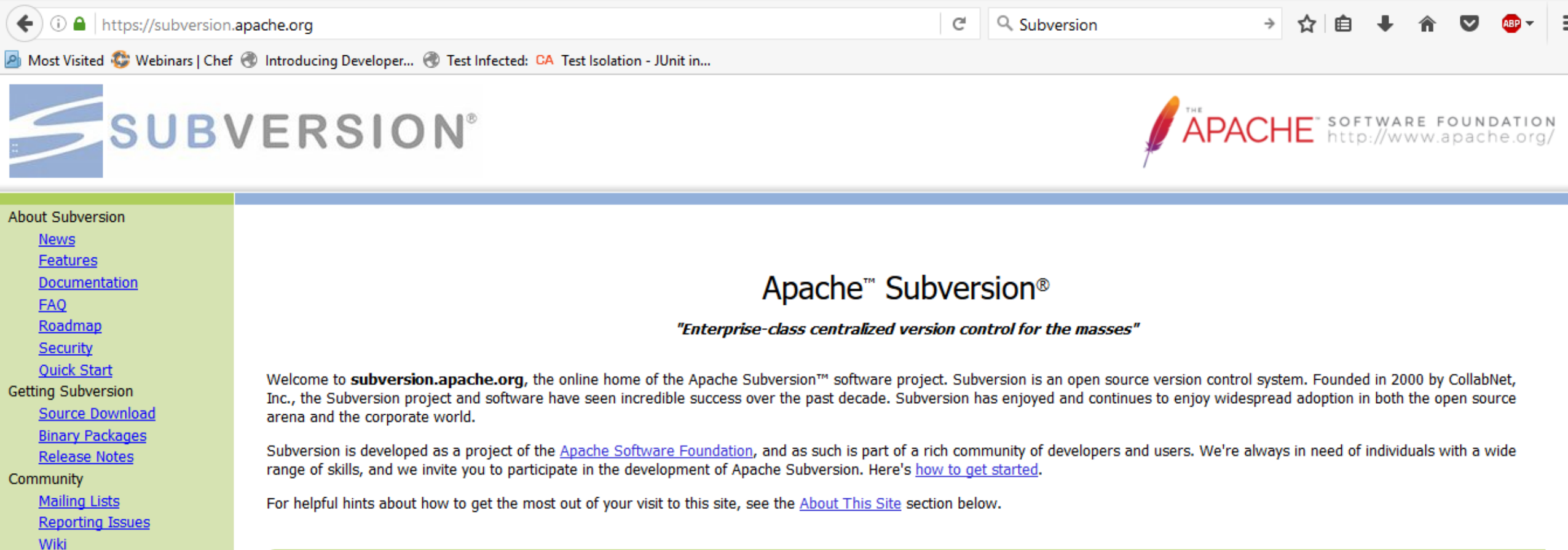
CVS AND THE 3 CM PROBLEMS

- The double maintenance problem
 - CVS uses RCS to provide a repository with a single “master copy” of each file.
- The shared data problem
 - Multiple developers can create local copies of the master copy to work on.
- The simultaneous update problem
 - Multiple developers can edit a file simultaneously!
 - File clobbering is prevented by CVS notifying a developer when an attempt to commit creates “merge conflicts”.
 - CVS creates a “merged” file with all changes.

CVS SUMMARY

- Free, open source, not-so-recent “industry standard”
- Centralized server architecture
- Non-trivial to install server; security issues.
- GUI and CLI clients available for many platforms.
- Provides version control and configuration management but not:
 - Build system
 - Defect tracking
 - Change management
 - Automated testing
- Optimistic locking creates work during merge.

SUBVERSION - SVN



The screenshot shows the Apache Subversion website. The browser's address bar displays 'https://subversion.apache.org'. The page features the Subversion logo on the left and the Apache Software Foundation logo on the right. A left-hand navigation menu lists links for 'About Subversion', 'Getting Subversion', and 'Community'. The main content area includes the title 'Apache™ Subversion®', a tagline, and a welcome message.

https://subversion.apache.org

Subversion

Most Visited Webinars | Chef Introducing Developer... Test Infected: CA Test Isolation - JUnit in...

SUBVERSION®

THE APACHE™ SOFTWARE FOUNDATION
http://www.apache.org/

About Subversion

- [News](#)
- [Features](#)
- [Documentation](#)
- [FAQ](#)
- [Roadmap](#)
- [Security](#)
- [Quick Start](#)

Getting Subversion

- [Source Download](#)
- [Binary Packages](#)
- [Release Notes](#)

Community

- [Mailing Lists](#)
- [Reporting Issues](#)
- [Wiki](#)

Apache™ Subversion®

"Enterprise-class centralized version control for the masses"

Welcome to **subversion.apache.org**, the online home of the Apache Subversion™ software project. Subversion is an open source version control system. Founded in 2000 by CollabNet, Inc., the Subversion project and software have seen incredible success over the past decade. Subversion has enjoyed and continues to enjoy widespread adoption in both the open source arena and the corporate world.

Subversion is developed as a project of the [Apache Software Foundation](#), and as such is part of a rich community of developers and users. We're always in need of individuals with a wide range of skills, and we invite you to participate in the development of Apache Subversion. Here's [how to get started](#).

For helpful hints about how to get the most out of your visit to this site, see the [About This Site](#) section below.

SUBVERSION

- Designed as a "compelling replacement" for CVS.
 - CVS code base old and crufty.
 - File-based versions create problems.
 - The current “industry standard”
- Similar to CVS in many respects (checkout, commit, optimistic locking)
- Some major differences:
 - Back-end repository not files, but a DB.
 - Versions are not file-based, but repository-wide.
 - "Directory" based tags and branches.
 - Arbitrary metadata.
- Downloads, etc: <http://subversion.tigris.org>

SVN VS. CVS IN A NUTSHELL

- Repository:
 - CVS uses flat files; SVN uses BerkeleyDB
- Speed:
 - SVN is faster.
- Tags and Branches:
 - CVS uses metadata; SVN uses folder conventions
- File Types:
 - CVS designed for ASCII, supports binary via 'hacks'; SVN handles all types uniformly.
- Transactional commit:
 - SVN supports; CVS does not.
- Deletion and renaming:
 - CVS support is bogus; SVN support is good.
- Popularity:
 - Most popular for new projects, but Git is gaining!

GIT



🔍 Search entire site...

Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is **easy to learn** and has a **tiny footprint with lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, convenient **staging areas**, and **multiple workflows**.



Learn Git in your browser for free with Try Git.



GIT

- A distributed version control system.
 - multiple redundant repositories
 - branching is a “first class concept”
- Every user has a complete copy of the repository (including history) stored locally.
- No “commit access”. Instead, you decide which users repositories to merge with your own.
- See Linus Torvalds talk on Git on YouTube.
- Mercurial is similar to Git.

GIT VS. SVN

- Advantages of Git:
 - Most operations much faster.
 - Git requires much less storage than SVN.
 - Git supports decentralized development.
 - No need for a “czar”
 - More workflows possible than with SVN
- Advantages of SVN:
 - Centralized repository means you know where the code is.
 - Access control possible.
 - Sometimes a “czar” is helpful for project coordination.

YET MORE CM ALTERNATIVES

- Bazaar (bzt):
 - Decentralized configuration management
 - P2P, not centralized server
 - Individuals publish branches to a web server.
- BitKeeper:
 - Decentralized
 - Not open source!
- ClearCase, Visual Source Safe, etc.
 - Commercial, costly solutions.
 - Include additional services (workflow, issue tracking, etc.)

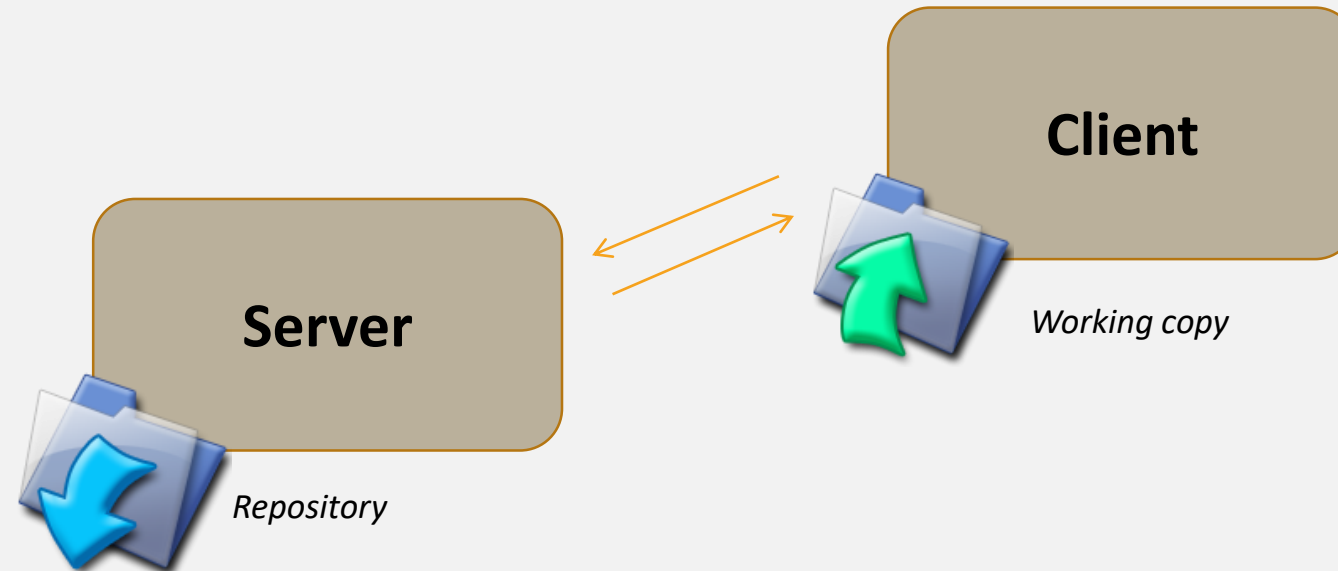
MODULE 02

BASIC VOCABULARY

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.

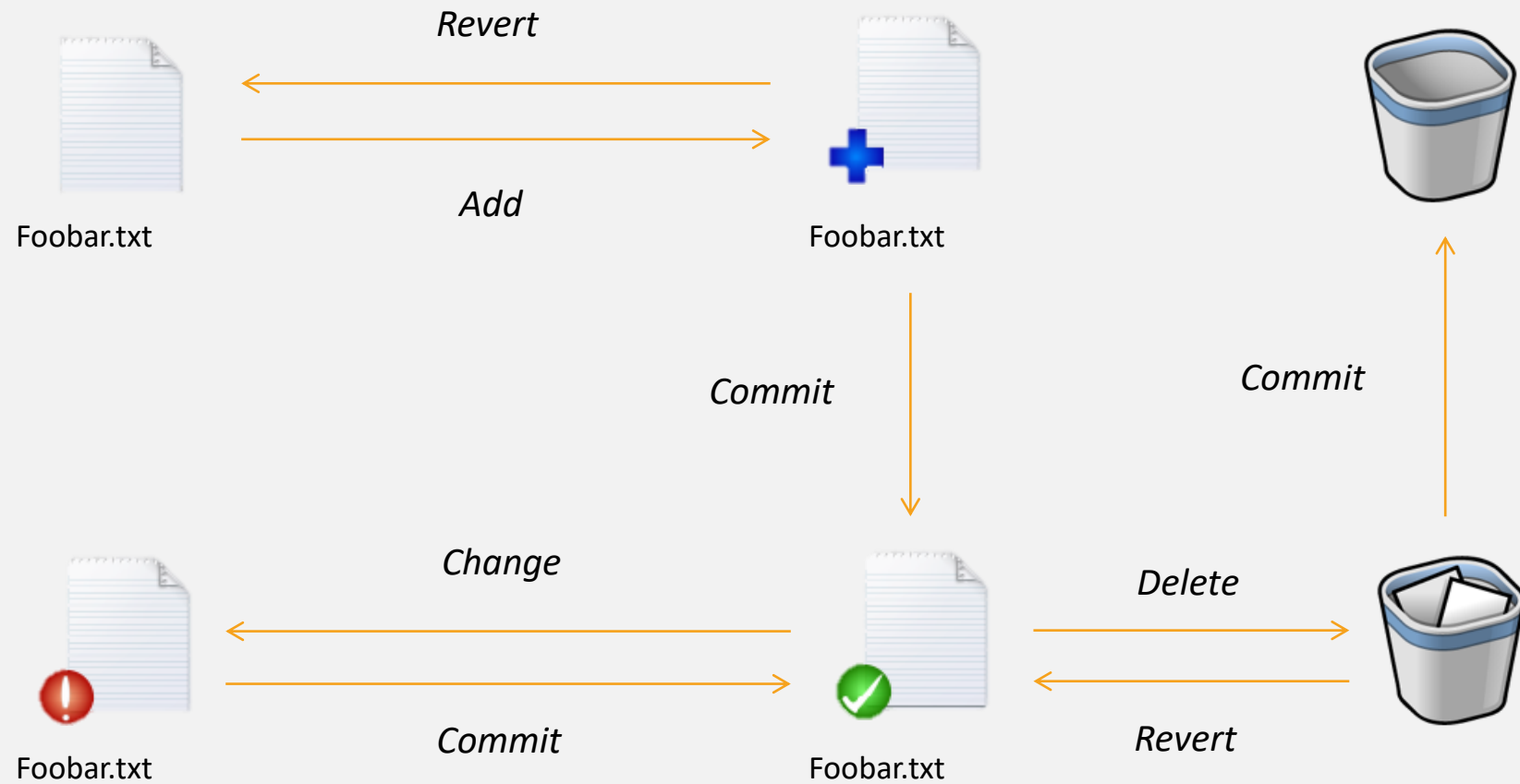
ENVIRONMENT



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

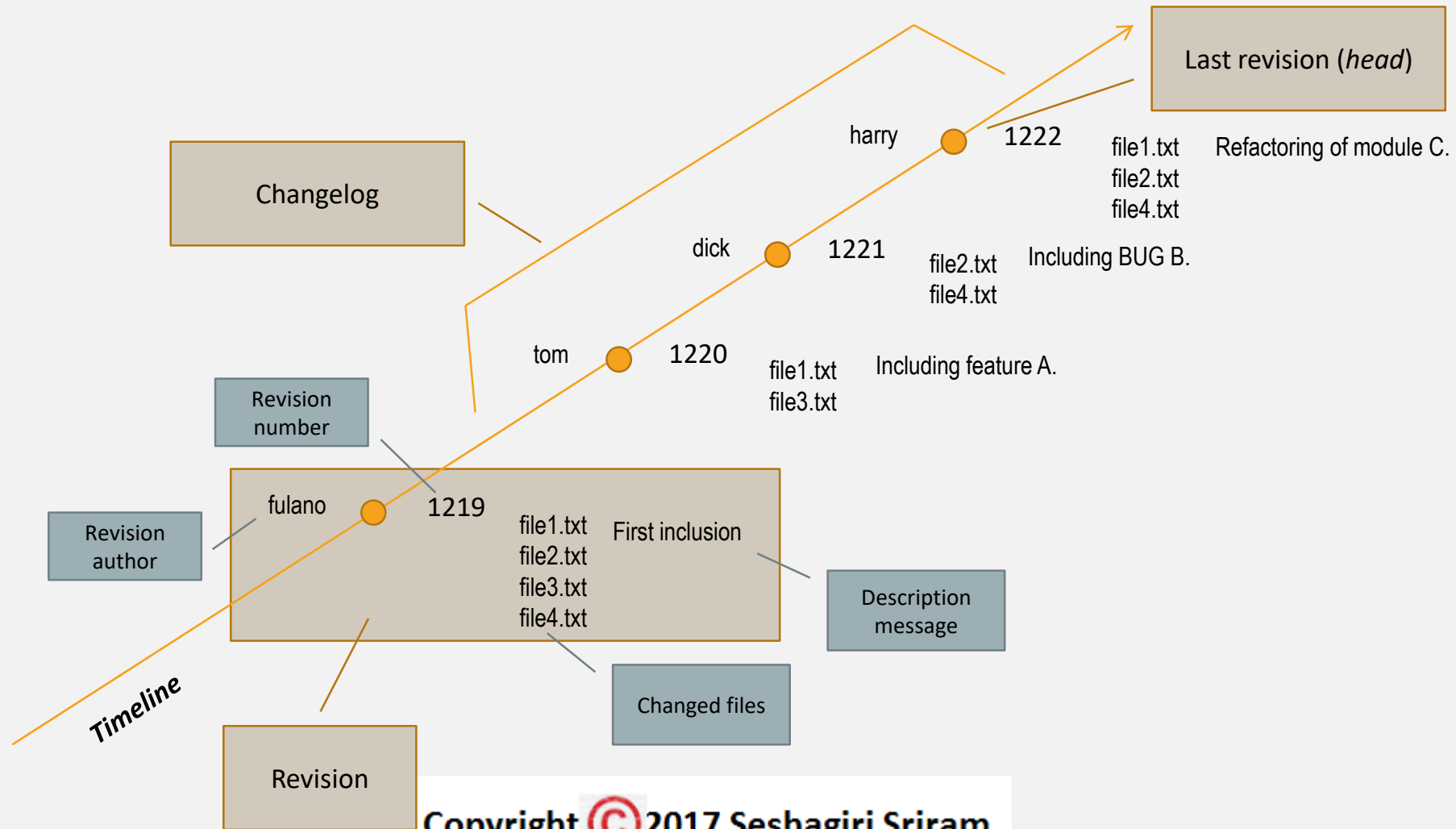
BASIC OPERATIONS



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

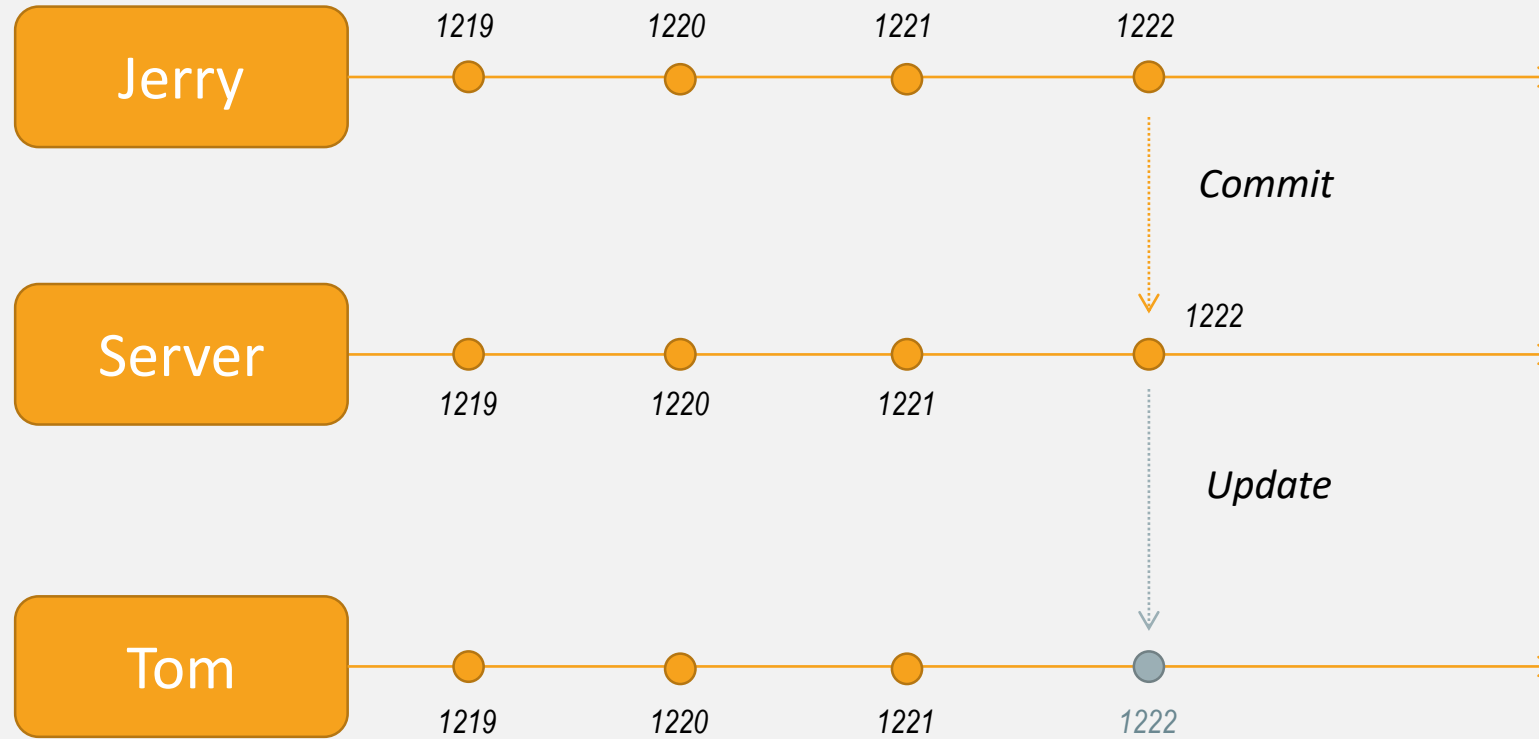
BASE ARTIFACTS



SYNCHRONIZATION

- Update
 - Synchronize changes from the repository to the local copy

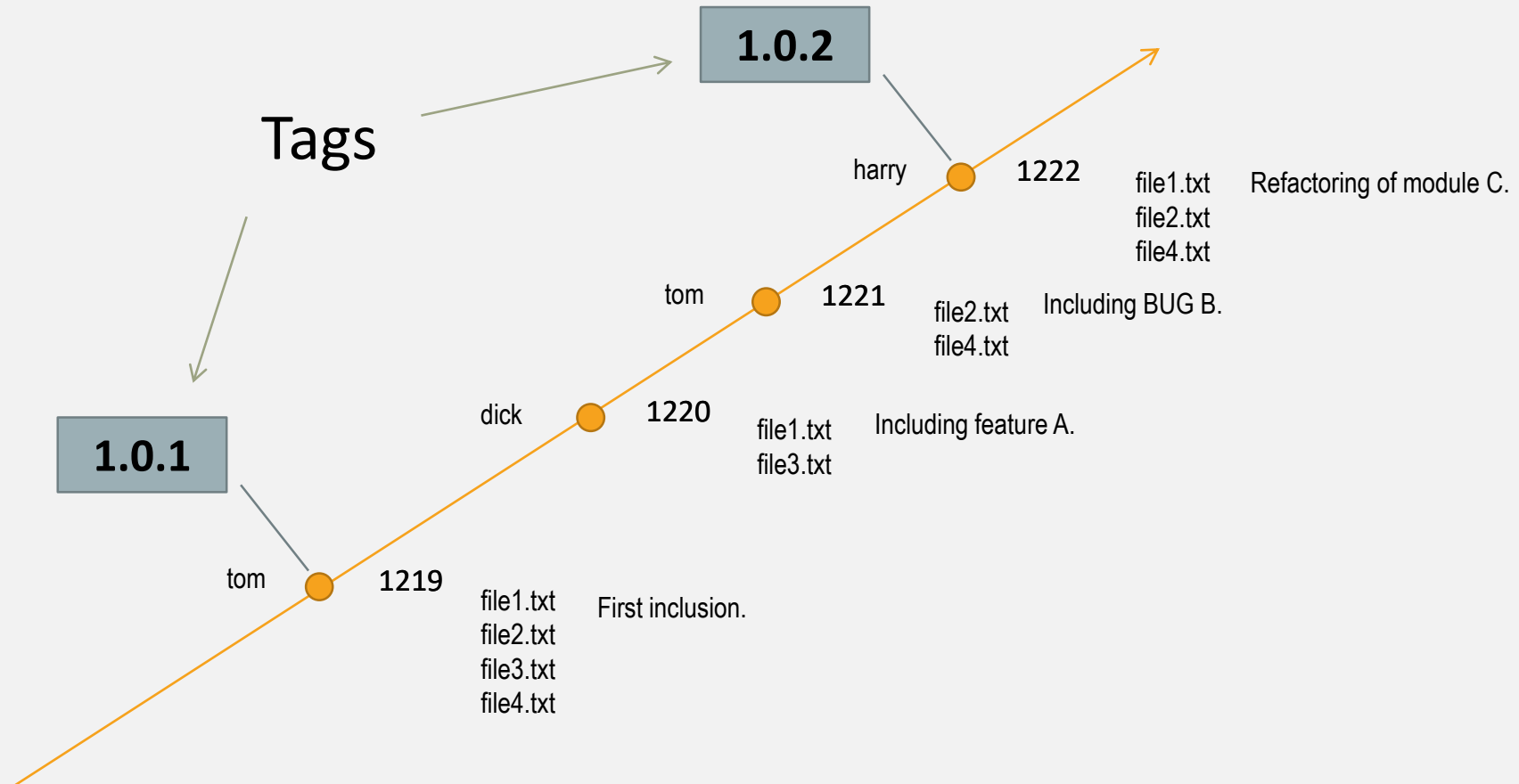
SYNCHRONIZATION



LABELS

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

LABELS



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

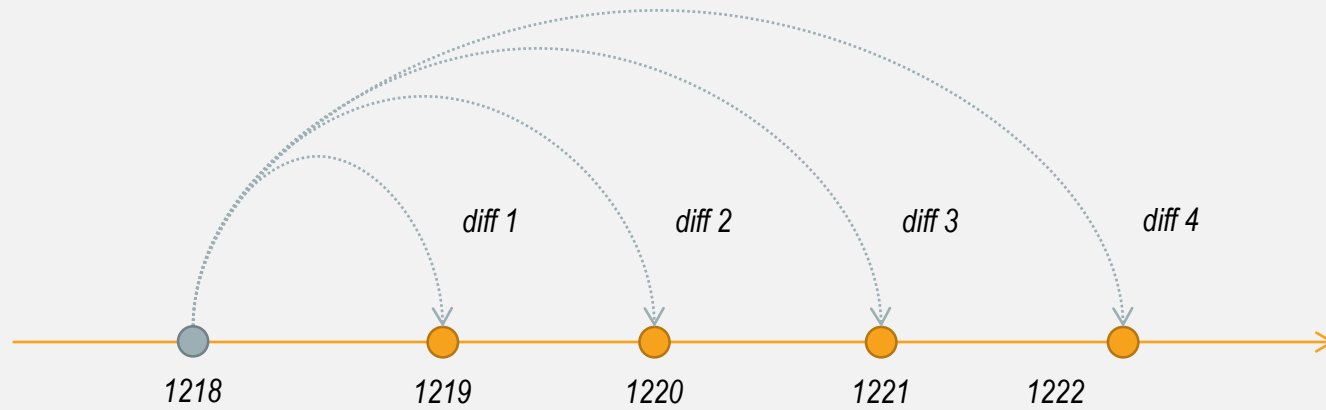
The command `diff -c original new` produces the following output:

```
*** /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



MODULE 03

BRANCHING

PARALLEL WORLDS

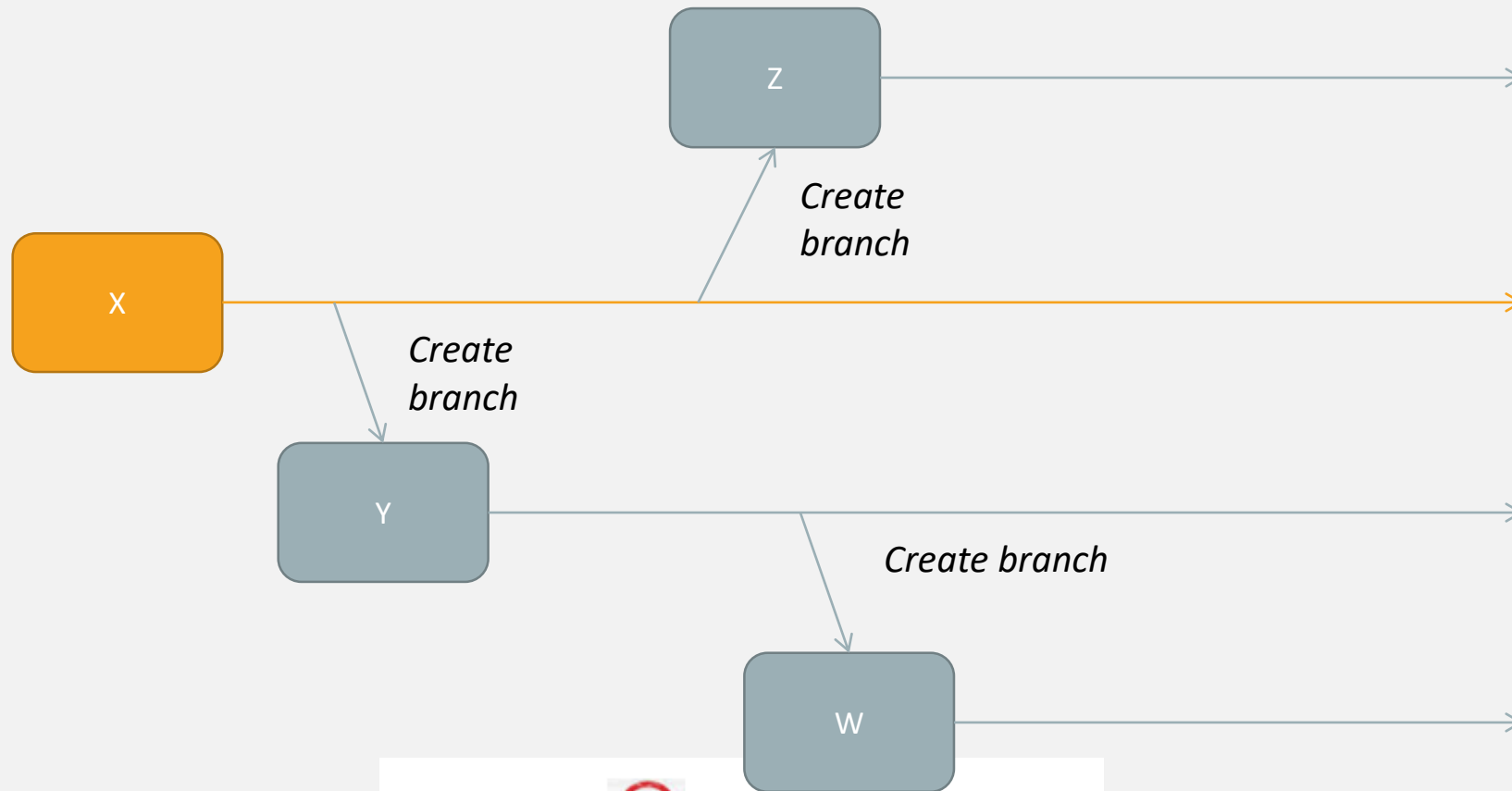
Perhaps the most accessible way to think of branches is as parallel universes. They're places where, for whatever reason, history didn't go quite the same way as it did in your universe. From that point forward, that universe can be slightly different – or it can be radically and utterly transformed. Like the Marvel comic book series “What If?”, branching lets you answer some interesting and possibly even dangerous "what if" questions with your software development.



PARALLEL WORLDS

- Parallel universes offer unlimited possibilities.
- They also allow you to stay safely ensconced in the particular universe of your choice, completely isolated from any events in other alternate universes.
- Branch is like a parallel world.
- Changes done in one branch doesn't have effect over the another.
- Although branching offers the seductive appeal of multiple possibility with very little risk, it also brings along something far less desirable: **increase of complexity**.

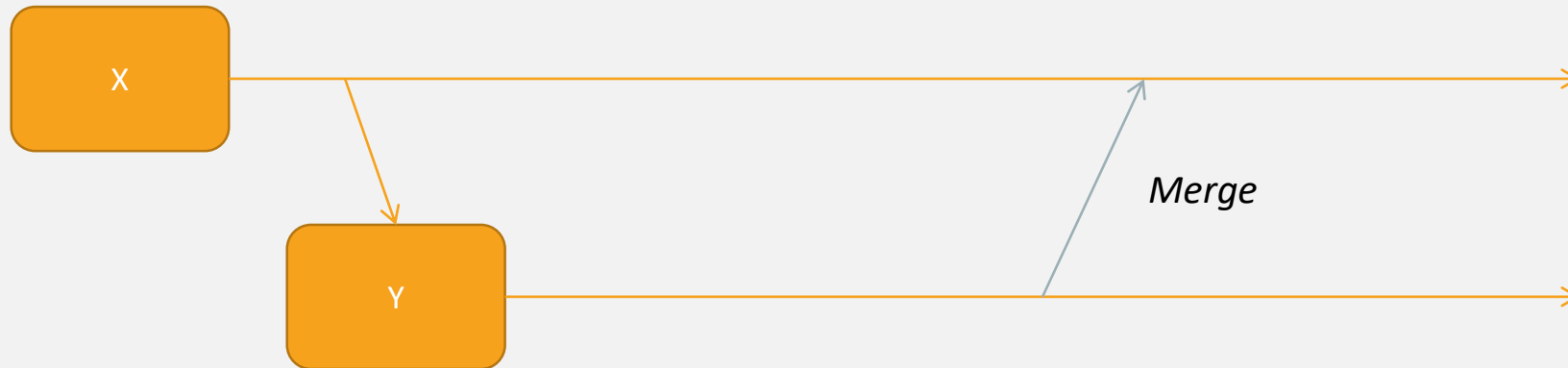
BRANCHES



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.

Question:

Is it possible to solve these conflicts automatically?

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?

A note before continuing:

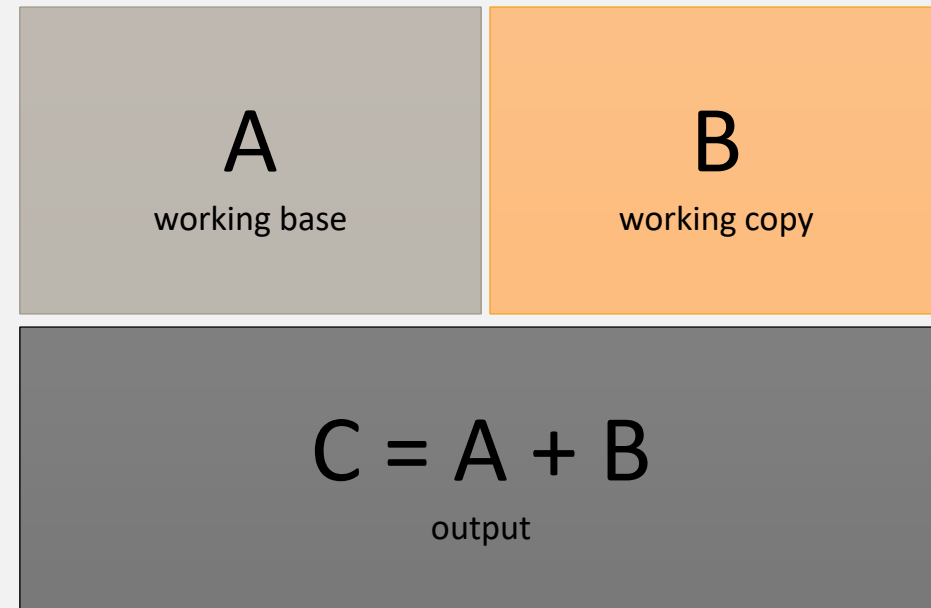
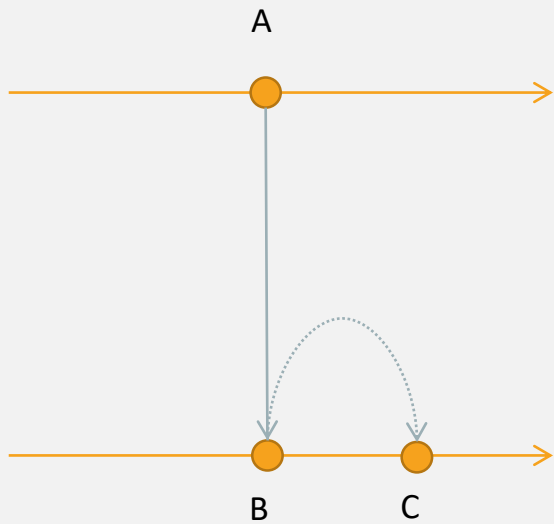
It is always better to avoid unnecessary conflicts.

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

Copyright © 2017 Seshagiri Sriram

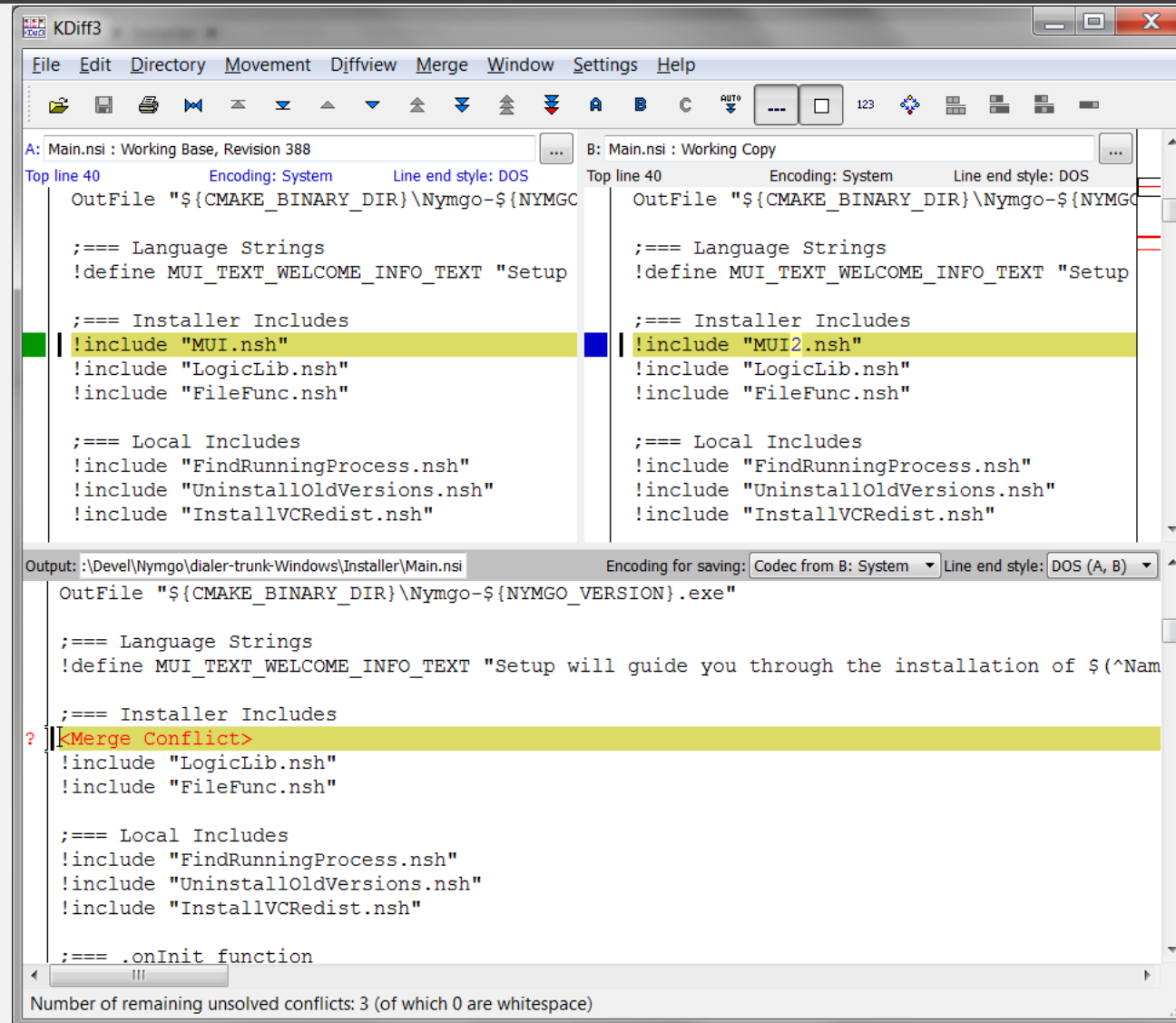
HOW TO SOLVE CONFLICTS?

Two way merge



TOOLS: KDIFF3

• Z



The screenshot shows the KDiff3 application window with three panes. The top pane shows the file menu and toolbar. The middle pane is split into two columns: 'A: Main.nsi : Working Base, Revision 388' on the left and 'B: Main.nsi : Working Copy' on the right. Both panes show the same code snippet, which includes language strings, installer includes, and local includes. The bottom pane shows the merged output, which includes a merge conflict marker and the code from both versions. The status bar at the bottom indicates 'Number of remaining unsolved conflicts: 3 (of which 0 are whitespace)'.

```
File Edit Directory Movement Diffview Merge Window Settings Help
[Icons] 123 [Icons]

A: Main.nsi : Working Base, Revision 388
Top line 40 Encoding: System Line end style: DOS
OutFile "${CMAKE_BINARY_DIR}\Nymgo-${NYMGO_VERSION}.exe"

;=== Language Strings
!define MUI_TEXT_WELCOME_INFO_TEXT "Setup"

;=== Installer Includes
!include "MUI.nsh"
!include "LogicLib.nsh"
!include "FileFunc.nsh"

;=== Local Includes
!include "FindRunningProcess.nsh"
!include "UninstallOldVersions.nsh"
!include "InstallVCRedist.nsh"

B: Main.nsi : Working Copy
Top line 40 Encoding: System Line end style: DOS
OutFile "${CMAKE_BINARY_DIR}\Nymgo-${NYMGO_VERSION}.exe"

;=== Language Strings
!define MUI_TEXT_WELCOME_INFO_TEXT "Setup"

;=== Installer Includes
!include "MUI2.nsh"
!include "LogicLib.nsh"
!include "FileFunc.nsh"

;=== Local Includes
!include "FindRunningProcess.nsh"
!include "UninstallOldVersions.nsh"
!include "InstallVCRedist.nsh"

Output: ..\Devel\Nymgo\diabler-trunk-Windows\Installer\Main.nsi
Encoding for saving: Codec from B: System Line end style: DOS (A, B)
OutFile "${CMAKE_BINARY_DIR}\Nymgo-${NYMGO_VERSION}.exe"

;=== Language Strings
!define MUI_TEXT_WELCOME_INFO_TEXT "Setup will guide you through the installation of $('Nam

;=== Installer Includes
? !<Merge Conflict>
!include "LogicLib.nsh"
!include "FileFunc.nsh"

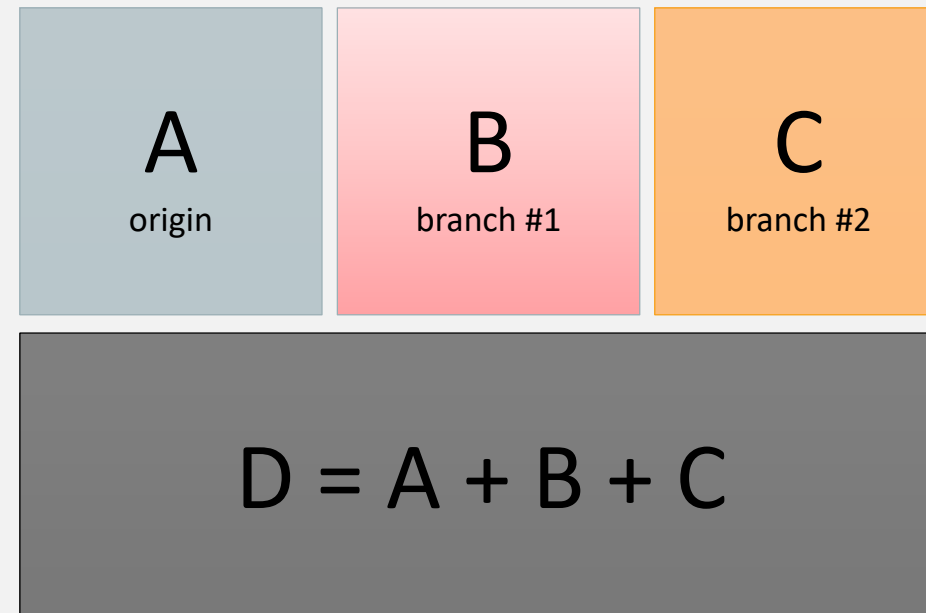
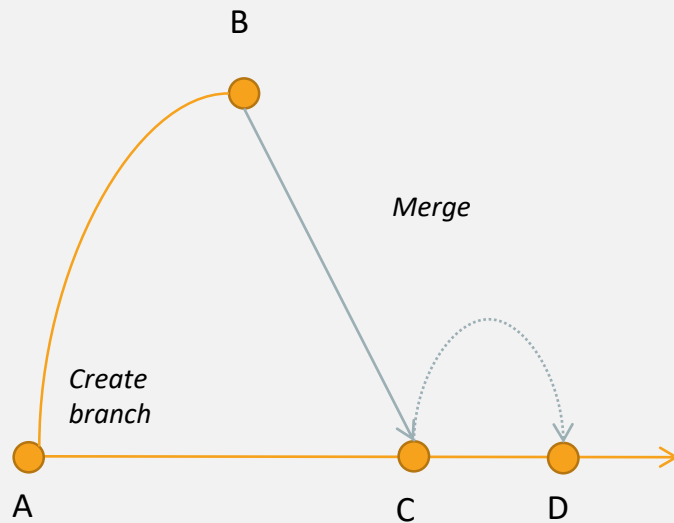
;=== Local Includes
!include "FindRunningProcess.nsh"
!include "UninstallOldVersions.nsh"
!include "InstallVCRedist.nsh"

;=== .onInit function
!include "FindRunningProcess.nsh"

Number of remaining unsolved conflicts: 3 (of which 0 are whitespace)
```

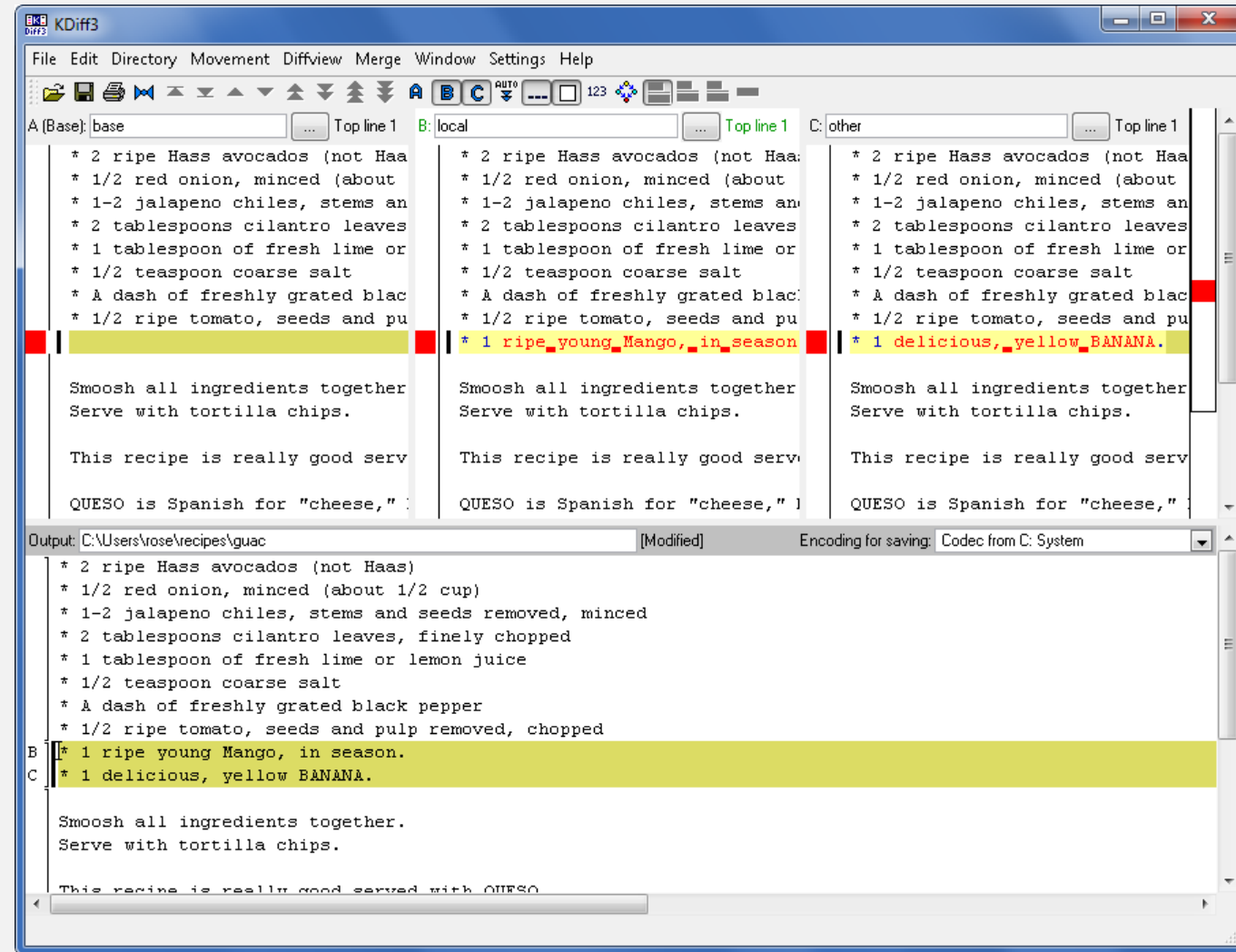
HOW TO SOLVE CONFLICTS?

Three way merge



TOOLS: KDIFF3

- Z



CHOOSE ANY ONE..

Guiffy SureMerge **Meld** *Ultracompare*
DiffMerge *Notepad++*
Apple Filemerge *Araxis* **tkmerge**
TFS diffmerge *xxdiff*

...

CHOOSE ANY ONE

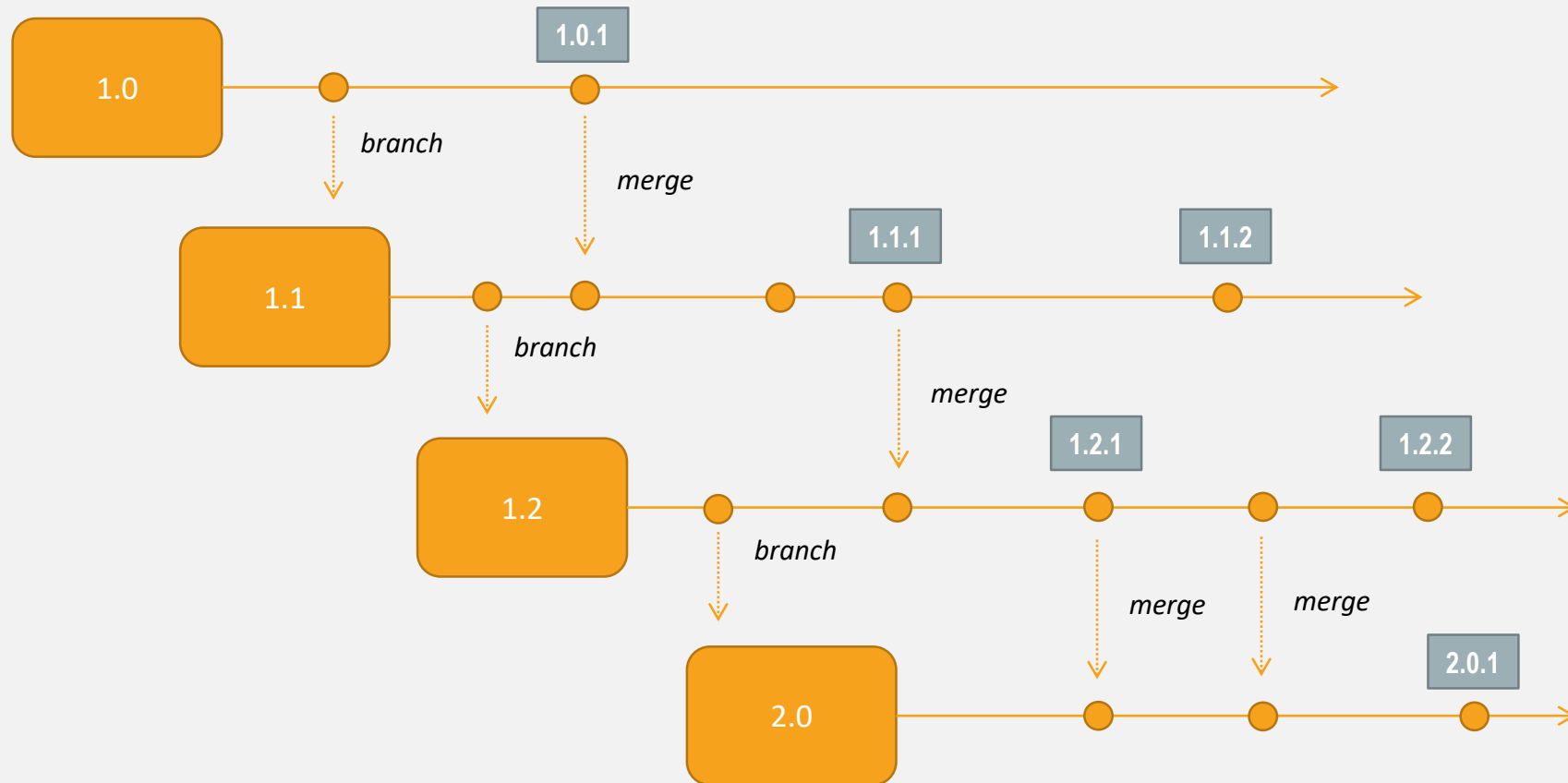
Which one should
I use?

*Choose one that best
fits your team!*

MODULE 04

BRANCHING PATTERNS AND ANTI-PATTERNS

MAINLINE



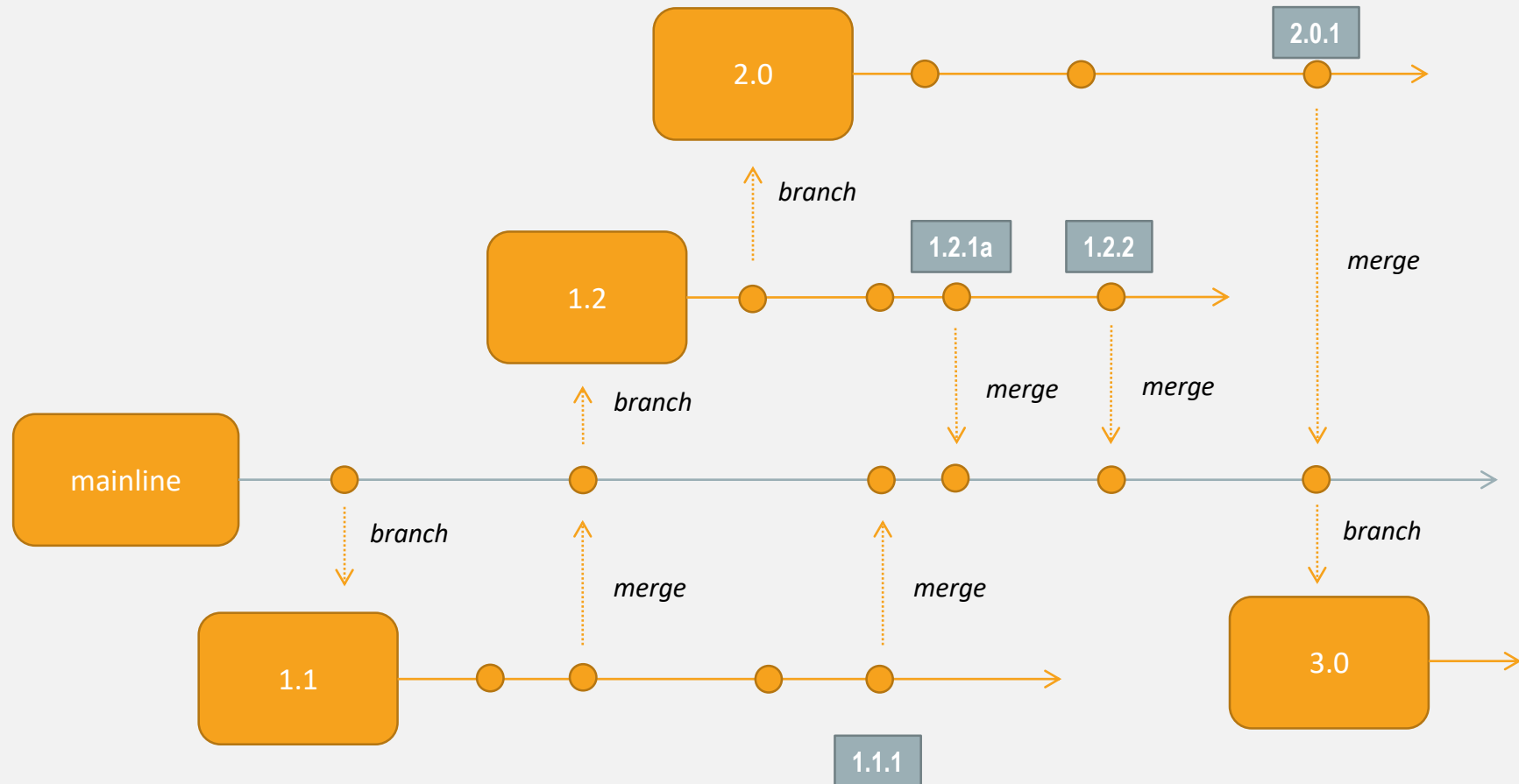
Bring a fix from branch n to branch m require $m-n$ merges (*linear complexity with the number of branches*).

MAINLINE

Learned lesson:

Perform merges the early and frequently as possible.

MAINLINE / VARIANT

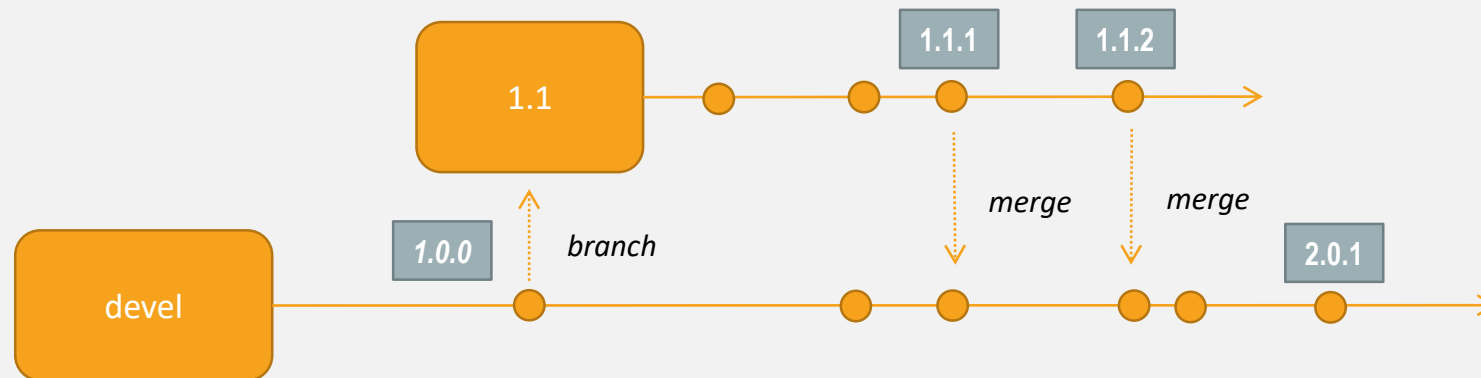


Scenario:

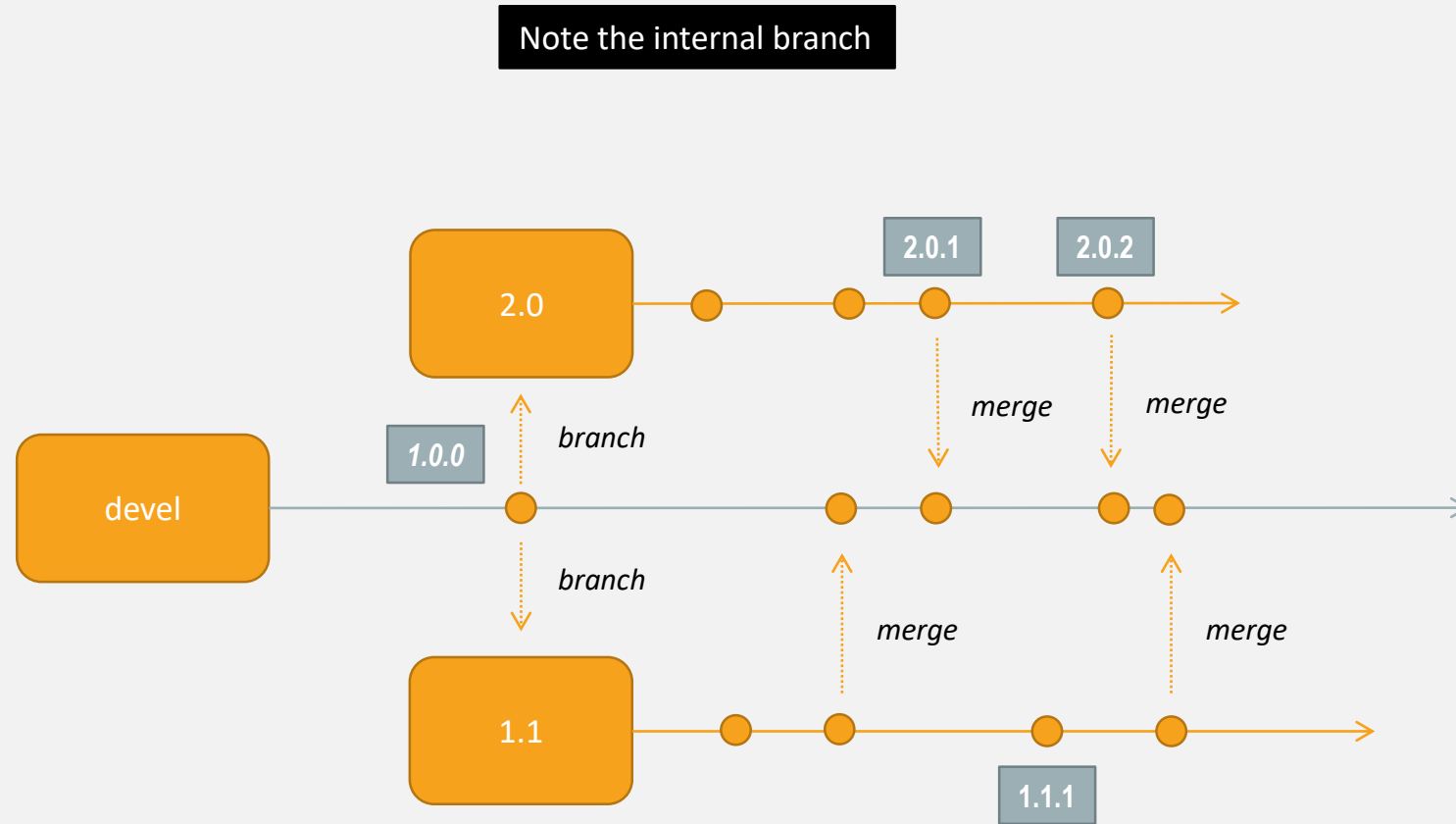
You just developed a stable version of the software and you need to create a new version with new features and still provide small fixes for the last stable version.

PARALLEL MAINTENANCE / DEVELOPMENT LINES

Most used option



PARALLEL MAINTENANCE / DEVELOPMENT LINES

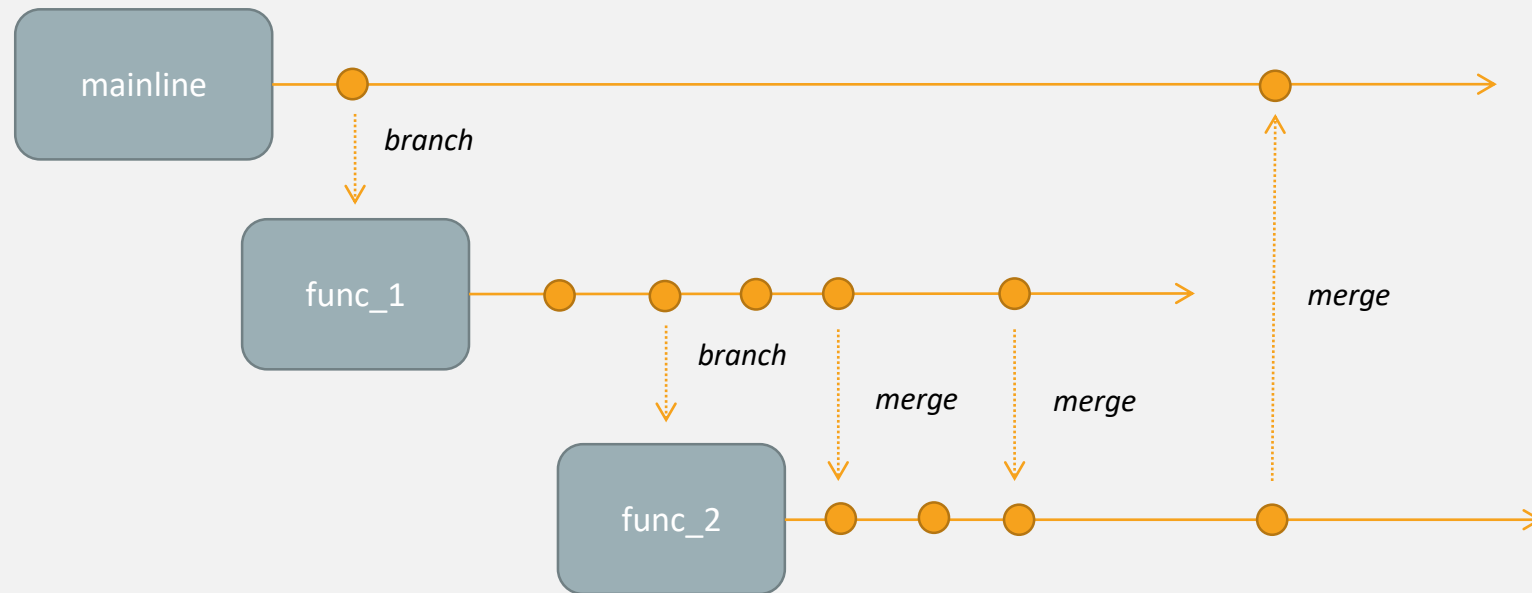


Scenario:

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

(laughs)

OVERLAPPING RELEASE LINES

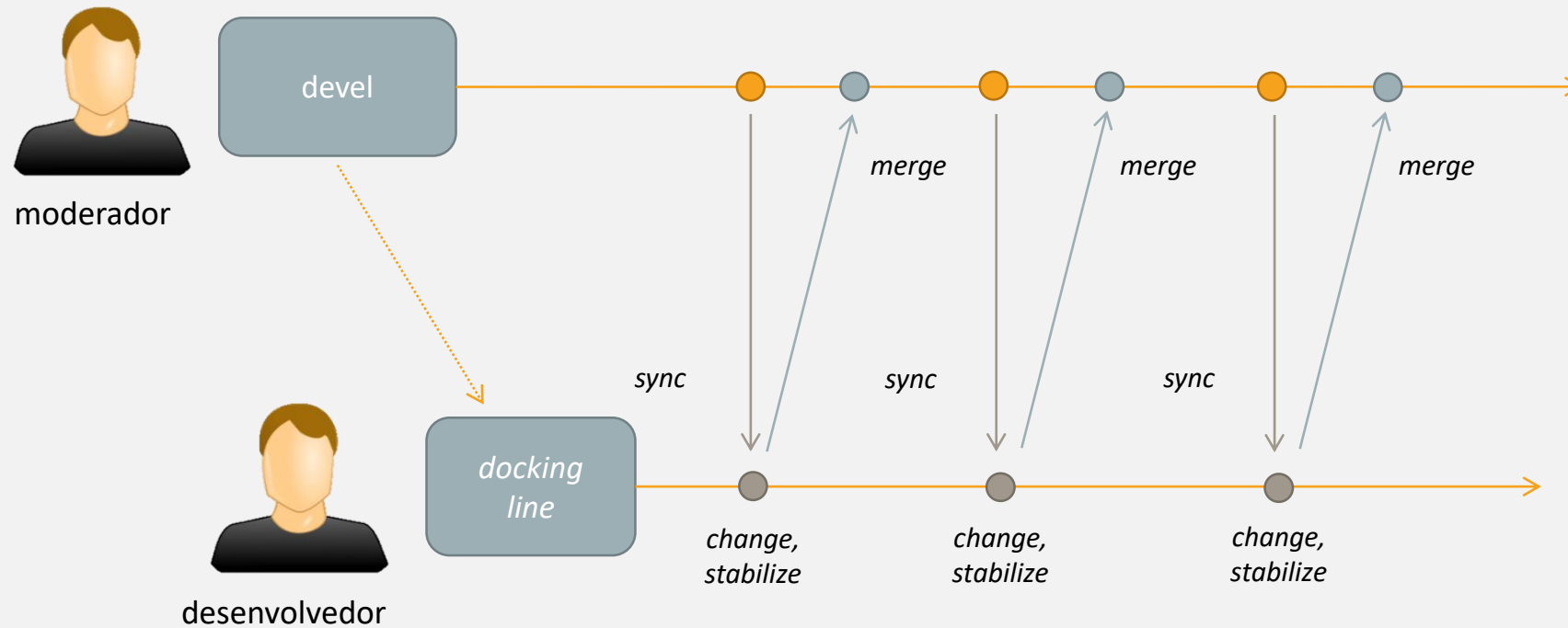


DOCKING LINE

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

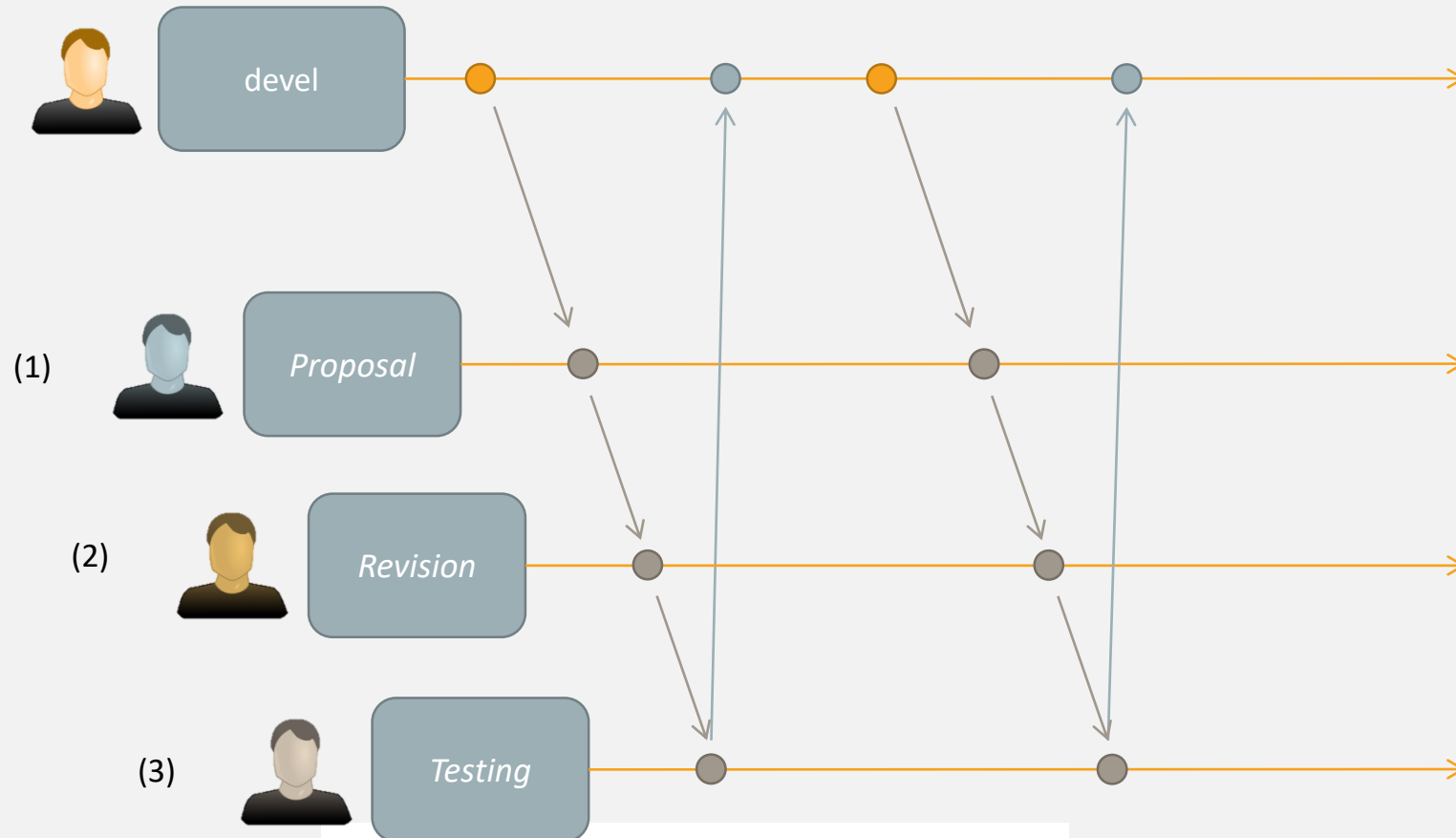


STAGED INTEGRATION LINES

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.

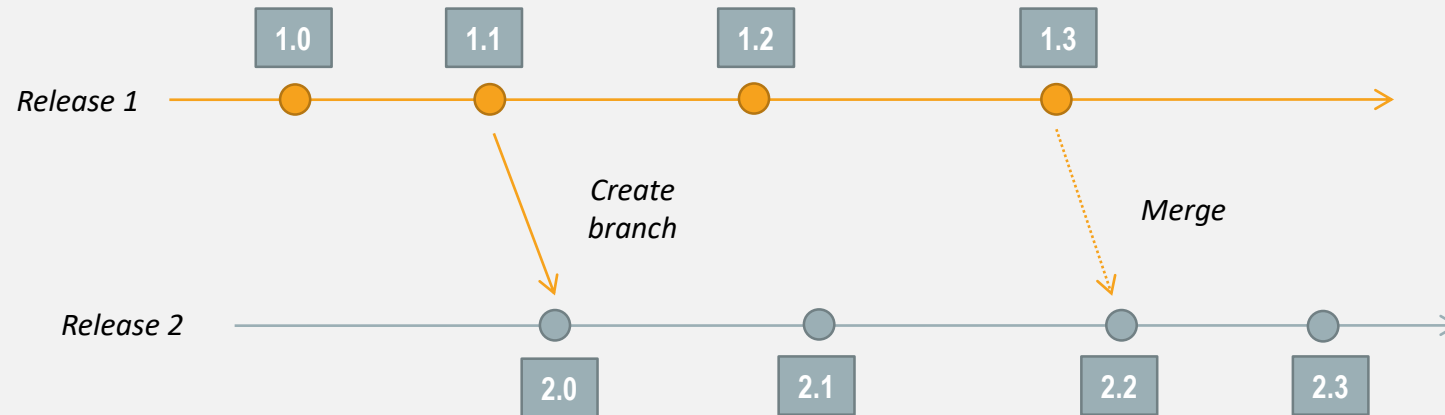
STAGED INTEGRATION LINES



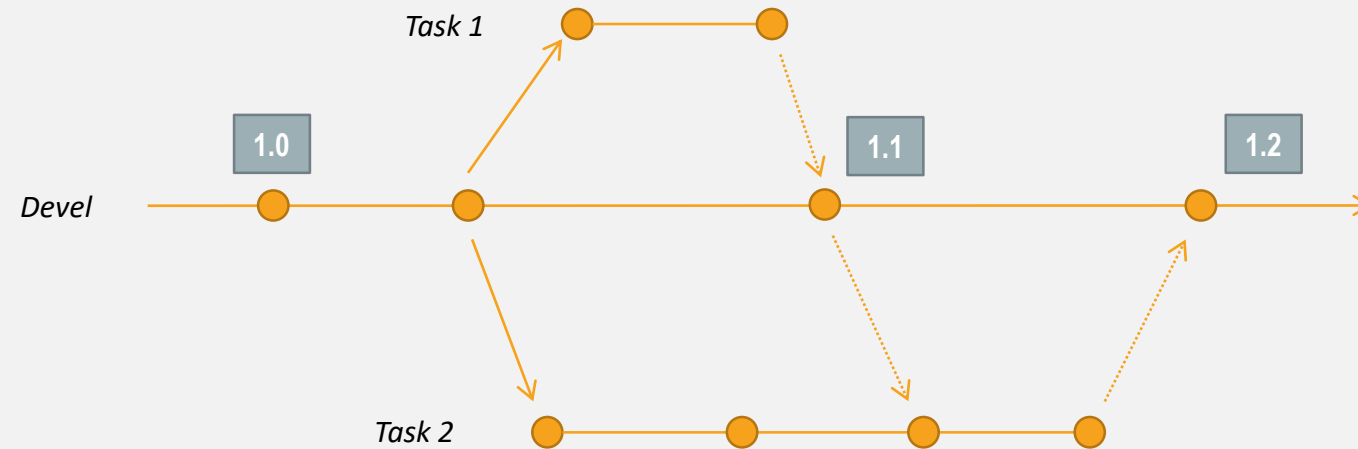
WHEN TO CREATE A BRANCH?

- Some most common patterns.

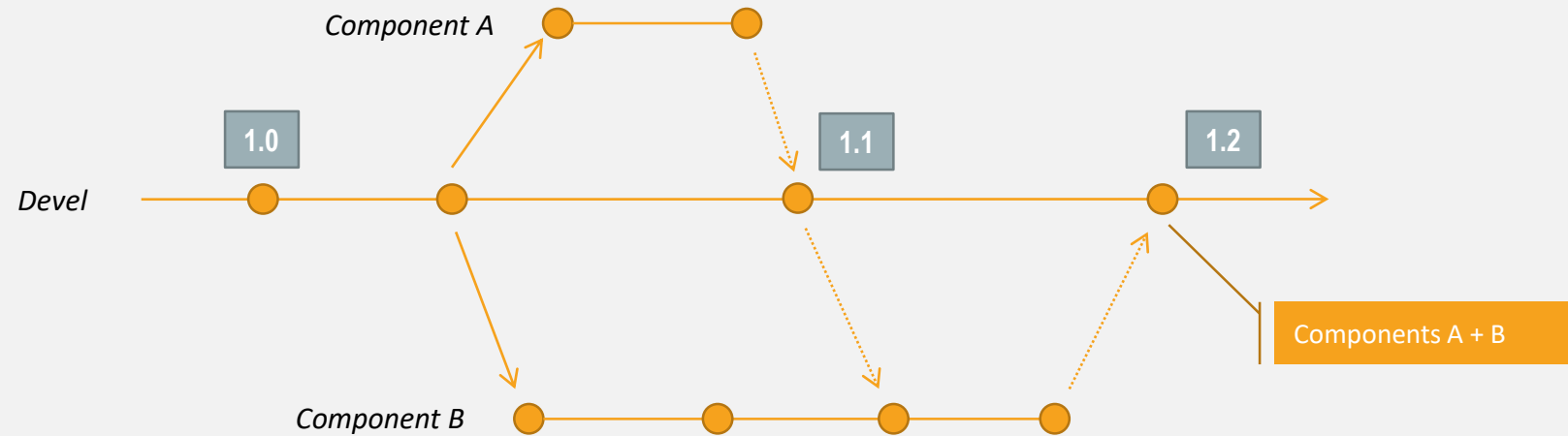
BRANCH PER DELIVERY



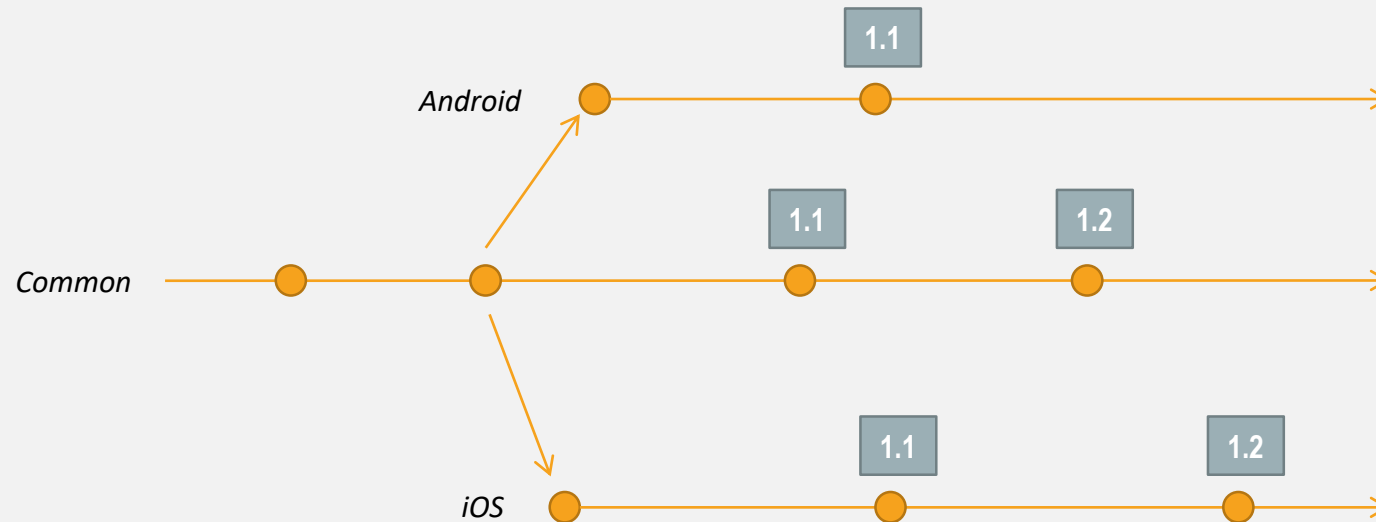
BRANCH PER TASK



BRANCH PER COMPONENT



BRANCH PER TECHNOLOGY



ANTI-PATTERNS

- What can happen if you don't manage branching?

PROBLEMS

| Anti-Pattern | Description |
|---------------------|--|
| Merge Paranoia | avoiding merging at all cost, usually because of a fear of the consequences. |
| Merge Mania | spending too much time merging software assets instead of developing them. |
| Big-Bang Merge | deferring branch merging to the end of the development effort and attempting to merge all branches simultaneously. |
| Nerver-Ending Merge | continuous merging activity because there is always more to merge. |
| Wrong-Way Merge | merging a software asset version with an earlier version. |
| Branch Mania | creating many branches for no apparent reason. |
| Cascading Branches | branching but never merging back to the main line. |

PROBLEMS

| Anti-Pattern | Description |
|---------------------|--|
| Mysterious Branches | branching for no apparent reason. |
| Temporary Branches | branching for changing reasons, so the branch becomes a permanent temporary workspace. |
| Volatile Branches | <p>branching with unstable software assets shared by other branches or merged into another branch.</p> <p>Note Branches are volatile most of the time while they exist as independent branches. That is the point of having them. The difference is that you should not share or merge branches while they are in an unstable state.</p> |
| Development Freeze | stopping all development activities while branching, merging, and building new base lines. |
| Berlin Wall | using branches to divide the development team members, instead of dividing the work they are performing. |

A grayscale image of a hand holding a globe. The map of India is highlighted with a darker, textured overlay. The globe is positioned on the left side of the slide, partially obscured by a large black rectangle.

Questions and Answers.



Thanks for listening!!!

Seshagiri Sriram

Email: SeshagiriSriram@gmail.com

Copyright © 2017 Seshagiri Sriram