Instituto de Computação

Foundations

Leonardo Gresta Paulino Murta

leomurta@ic.uff.br

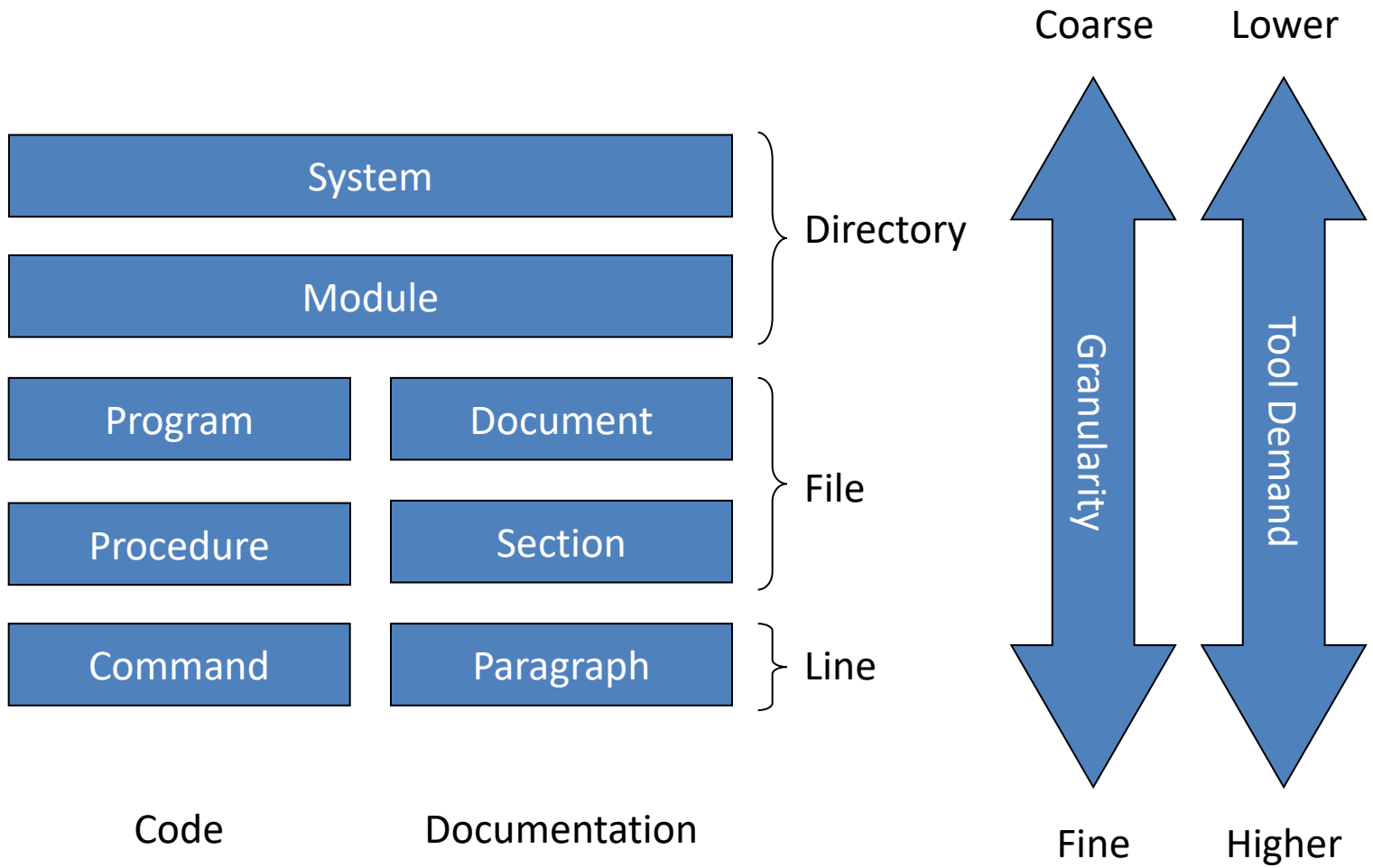
Configuration Item

- Hardware or software aggregation subject to configuration management
- Examples:
  - CM plan
  - Requirement Engineering Process
  - Requirements
  - Models
  - Source-code of component X
  - Etc.

Configuration Item

- The selection of CI should take into considerations basic design principles such as coupling and cohesion
- High coupling introduces complexity to the building process
  - Many dependencies among CI
- Low cohesion introduces complexity to the development process
  - Many developers working over the same CI
- CM benefits from well defined architectures

Configuration Item



Perspective: Code Documentation

Derived Item

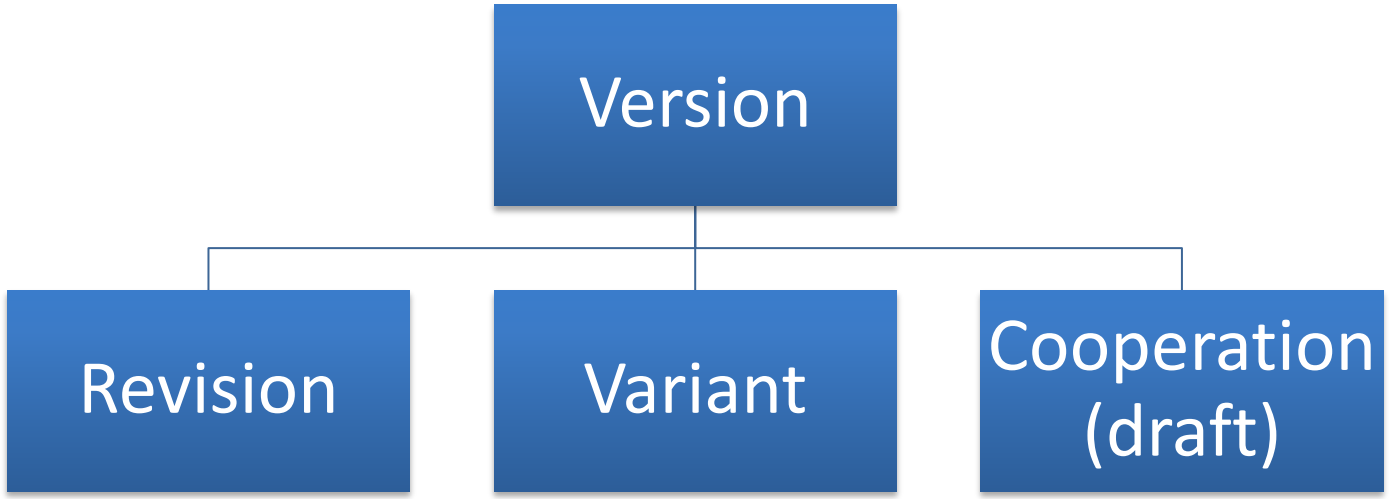
- A CI may be derived from other CI (source items)
- Example:
  - Executable files are derived from the source code
  - DB Schema is derived from class models
  - Etc.
- CM Strategies
  - Version control the derived items
  - Document and version control the derivation process (script, tools, environment, etc.)

Building

- Process that generates derived items from source items considering a target configuration
- Uses automated build scripts to describe the process
- Example:
  - makefile,
  - build.xml,
  - pom.xml
- The build scripts are also subject to CM

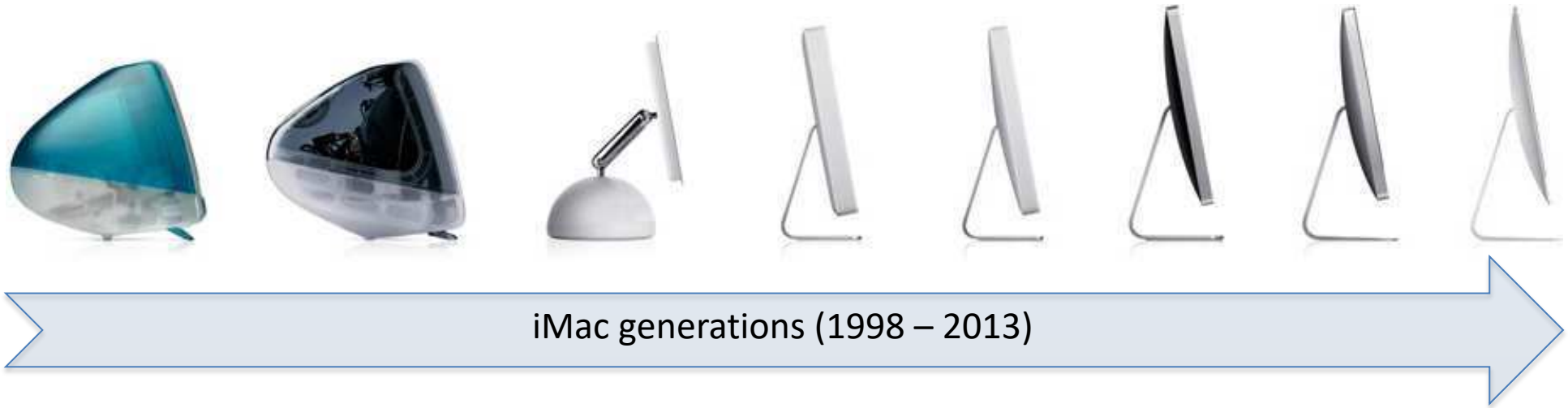
Version

- Different instances of the same CI
- Three types of Versions:

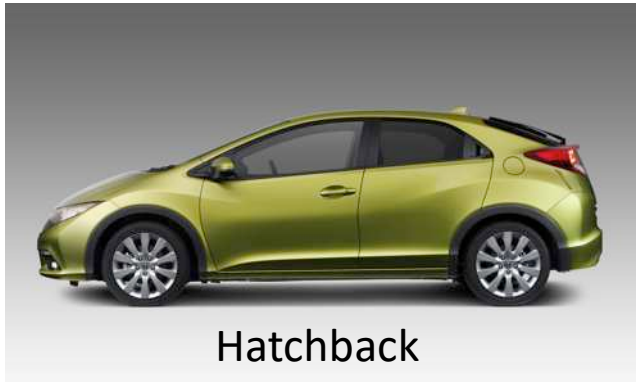


(Conradi and Westfechtel 1998)

Revisions



Variants



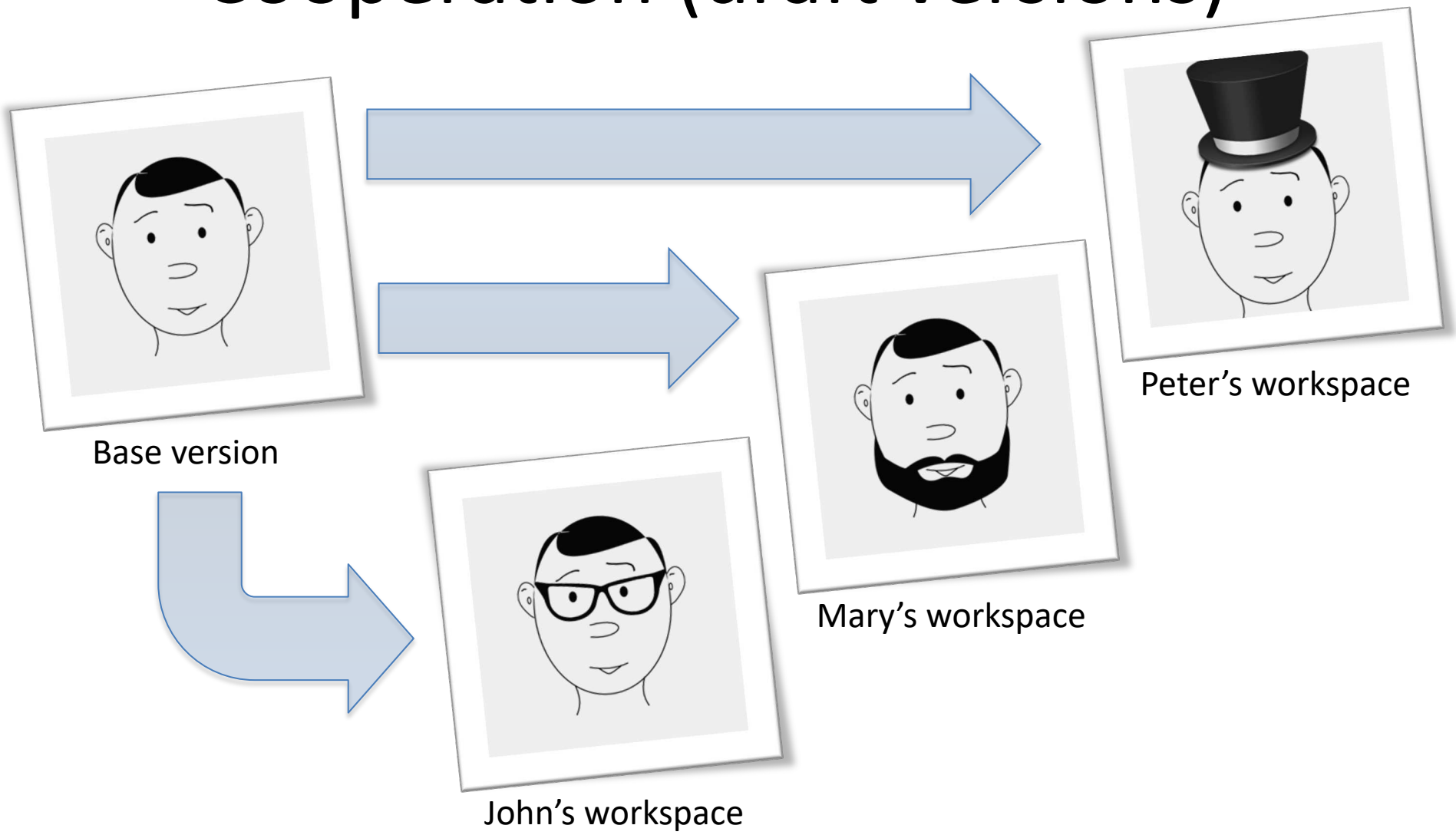
Hatchback

Honda Civic

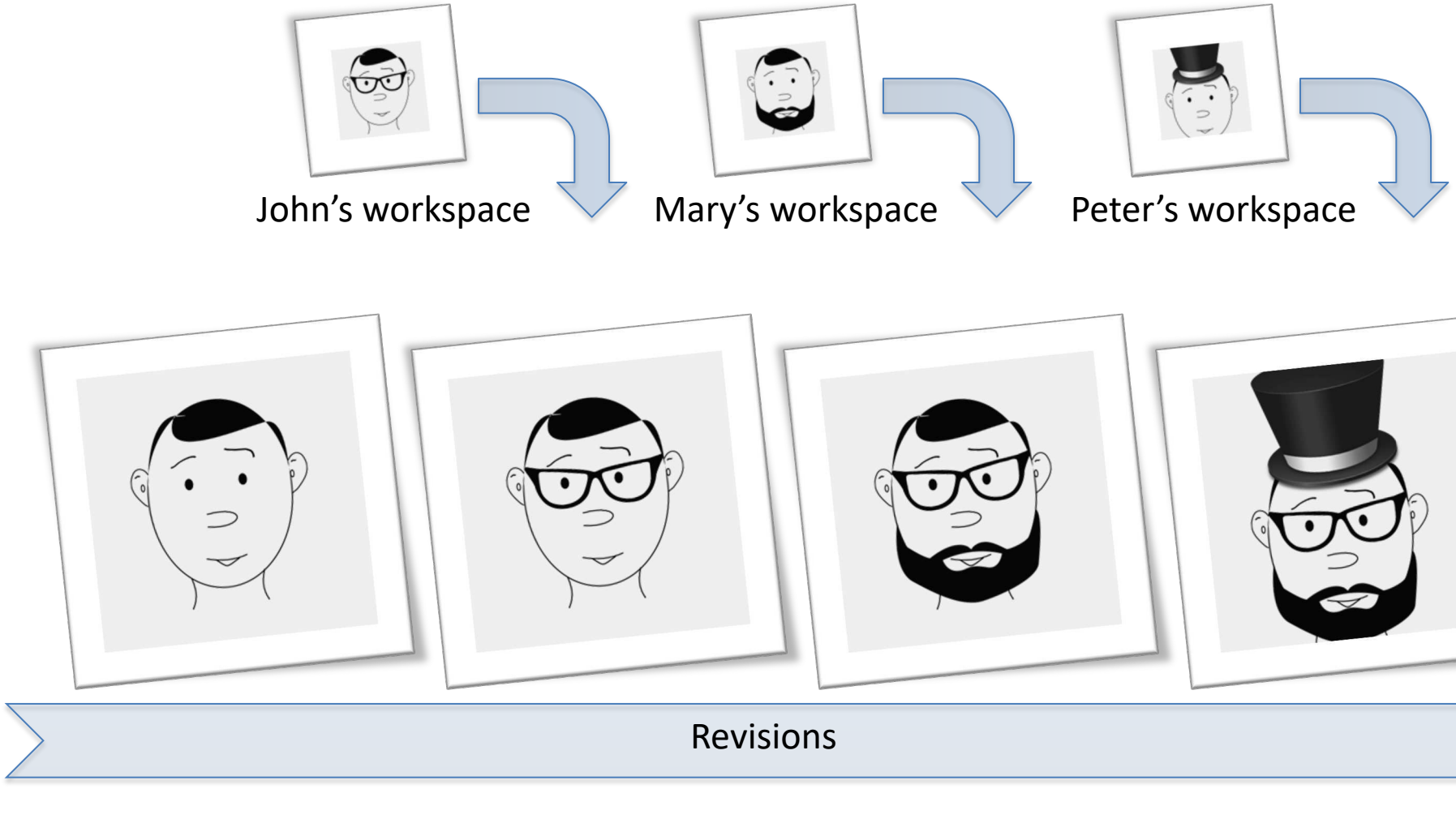
Sedan

Coupe

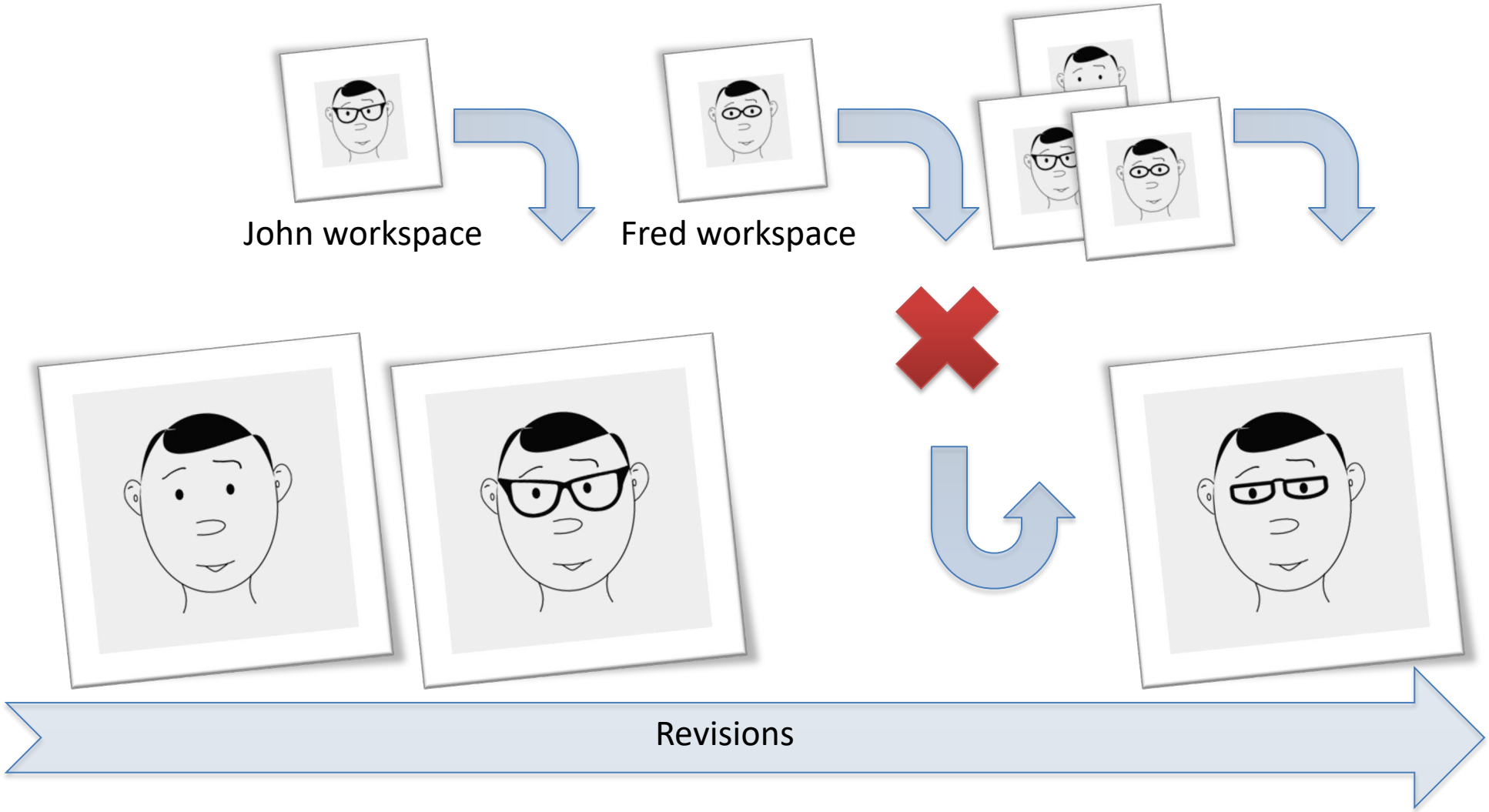
Cooperation (draft versions)



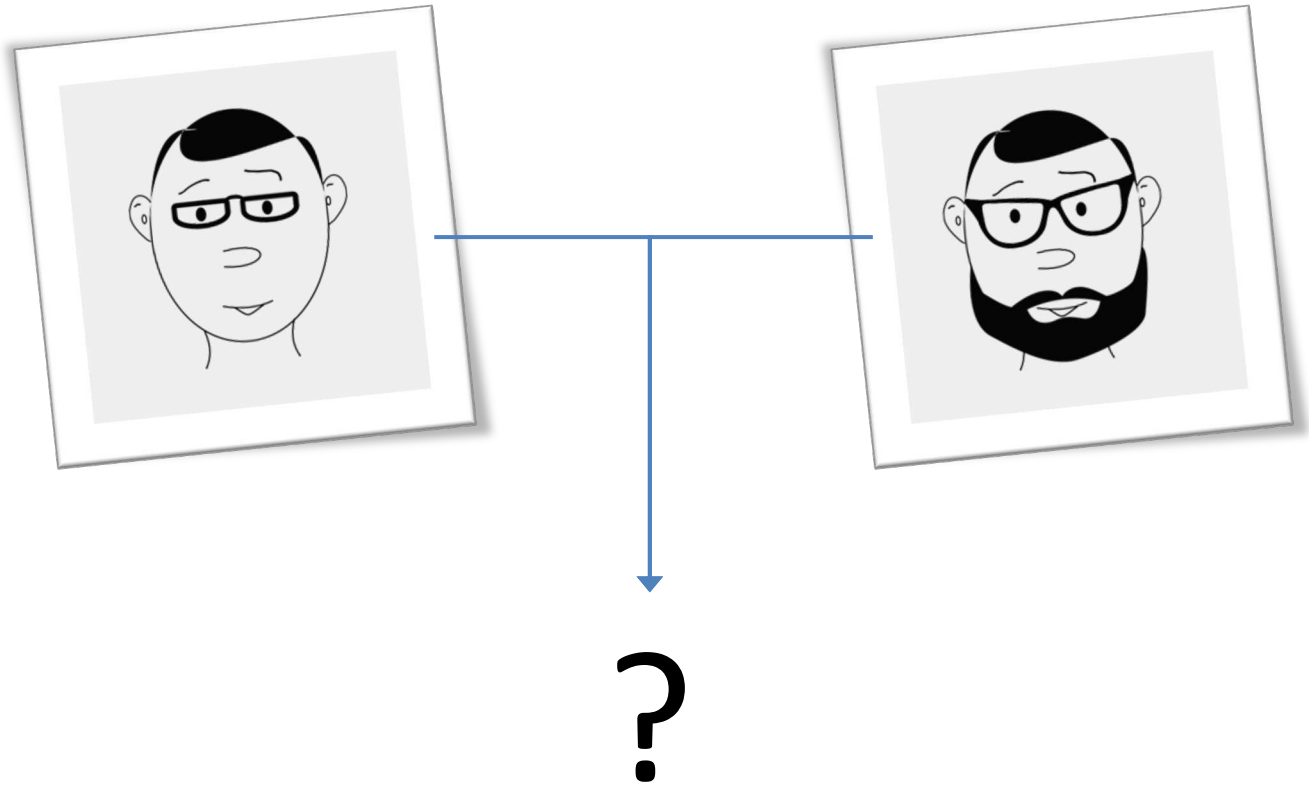
Draft versions may be merged



Conflicts may happen during merge

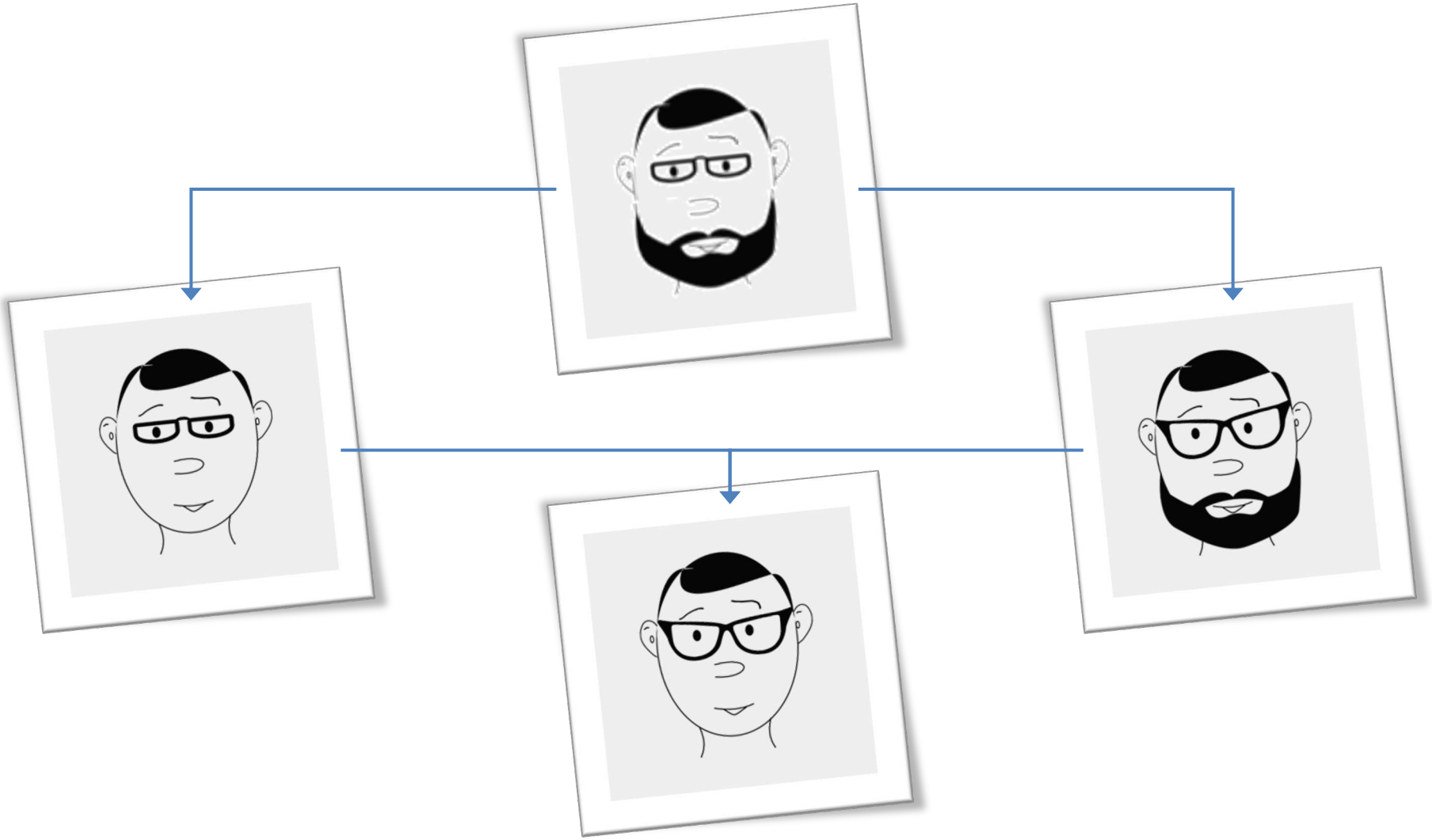


2-way merge

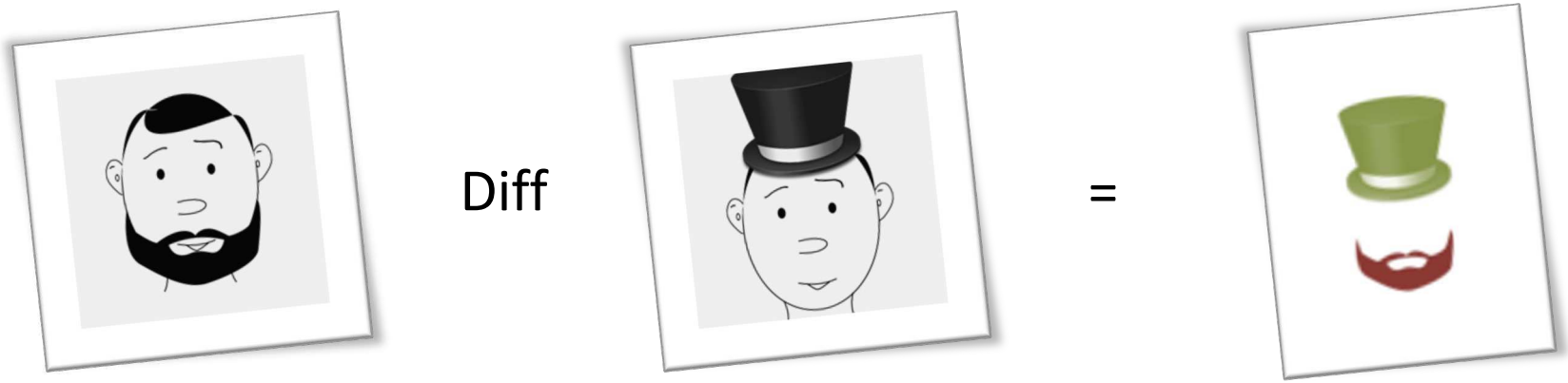


?

3-way merge

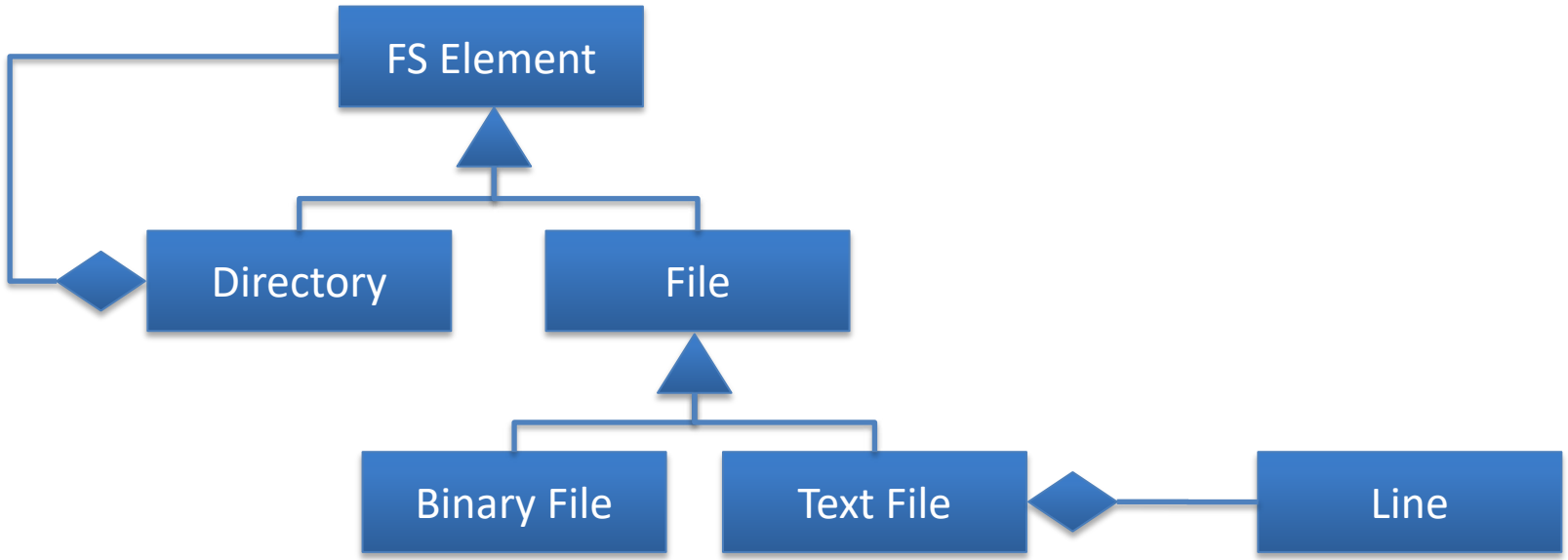


Two other important operations…

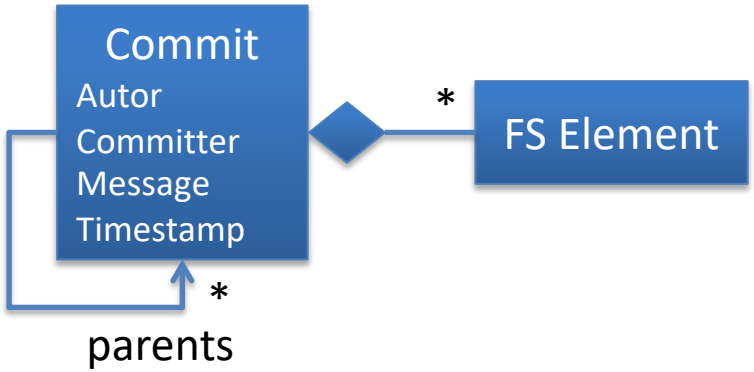


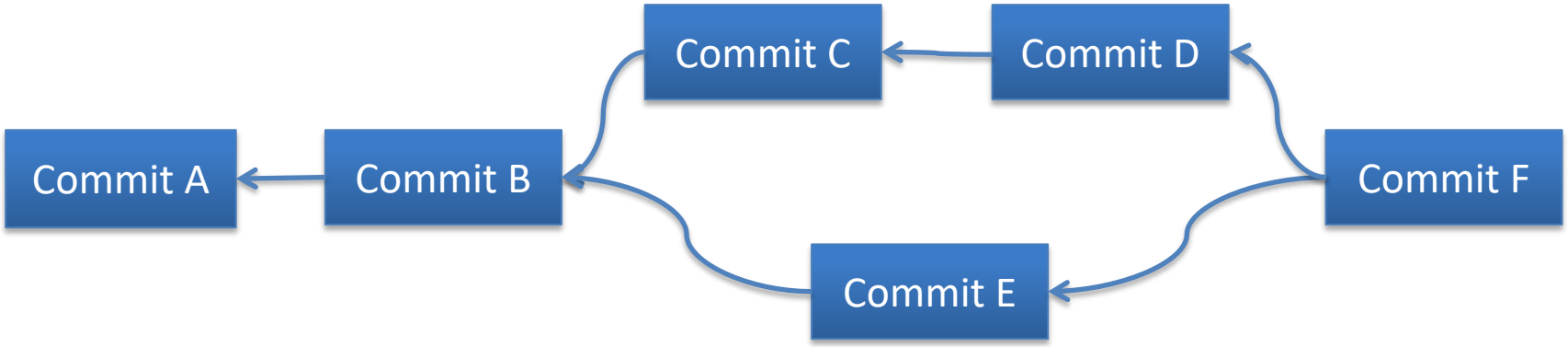
… for storing, transferring, and comprehending versions

What is (usually) subject to versioning?



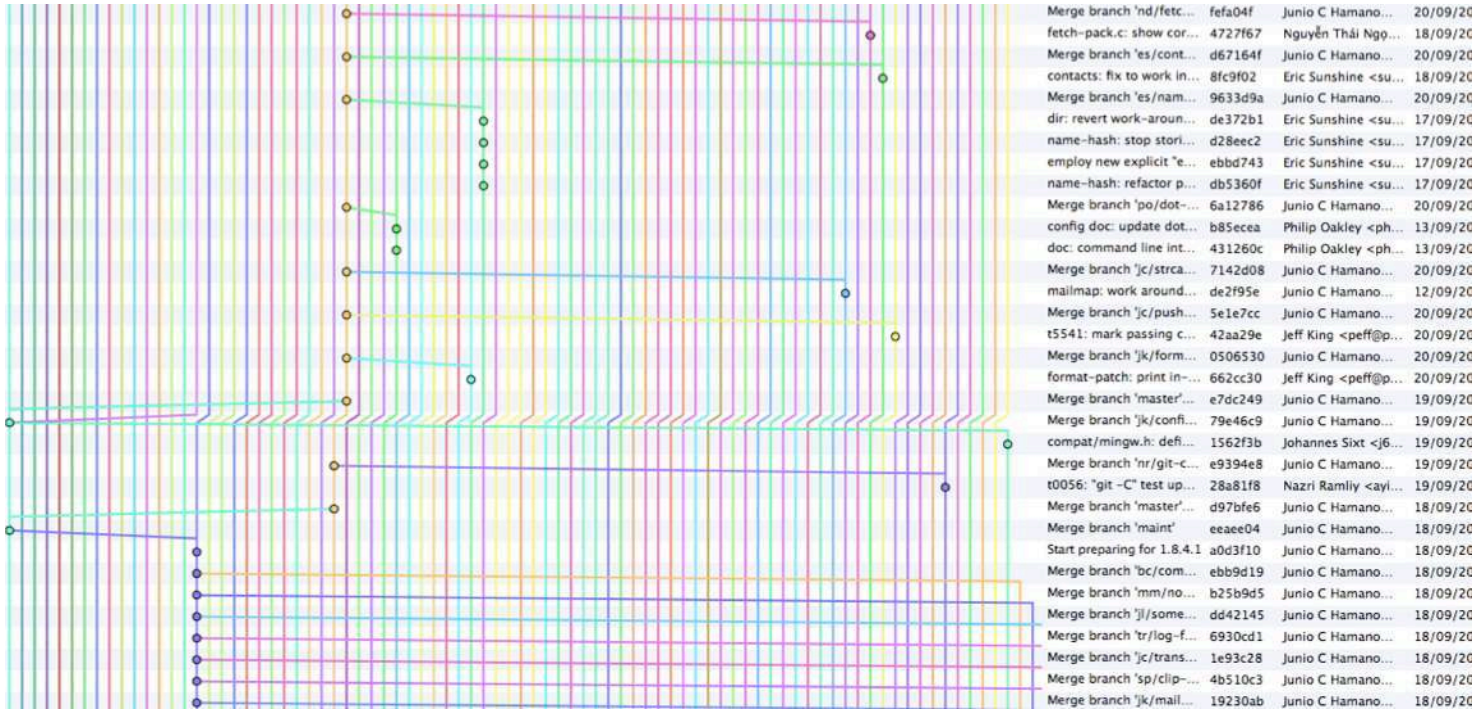
How is it (usually) versioned?





Versions in the wild

- Multitude of revisions and variants altogether (set aside draft versions)

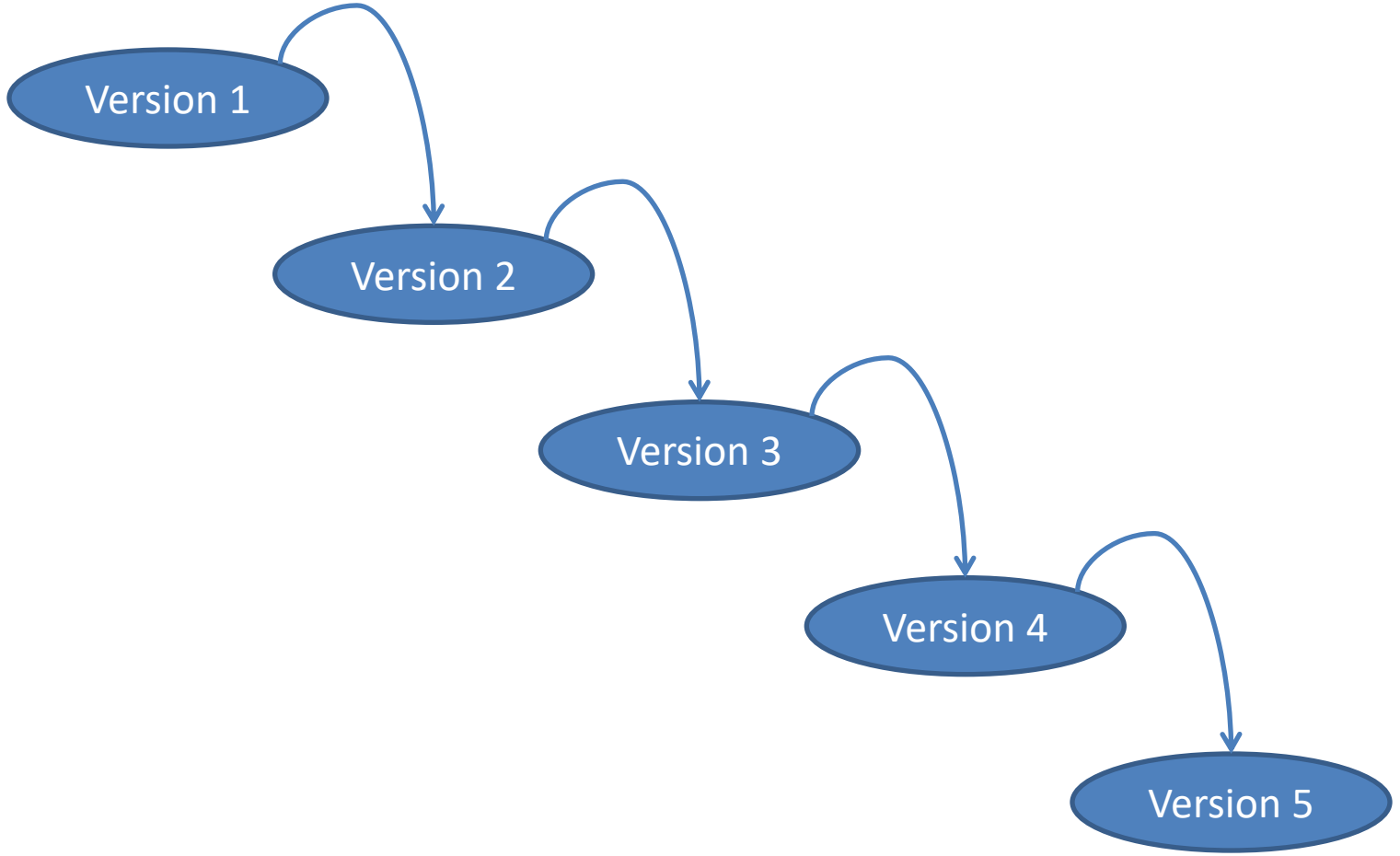


History of Git

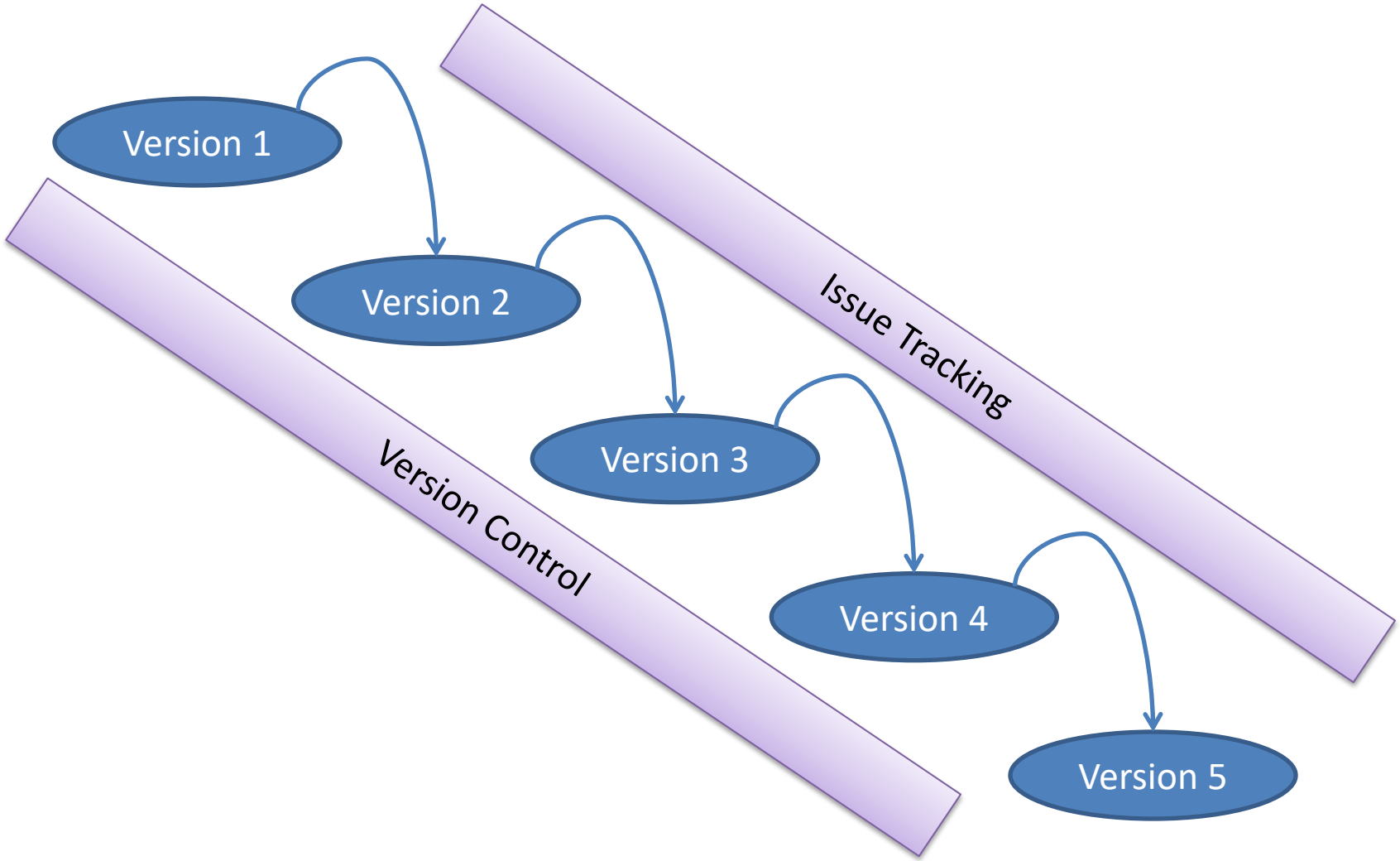
But, what are versions good for?

- Synchronizing teamwork
- Reproducing previous configurations
- Exploring possibilities
- Segregating developers
- Customizing products (SPL)
- Tracing bug introduction (bisect)
- Understanding evolution (MSR)
- Auditing changes (annotate)
- Etc.

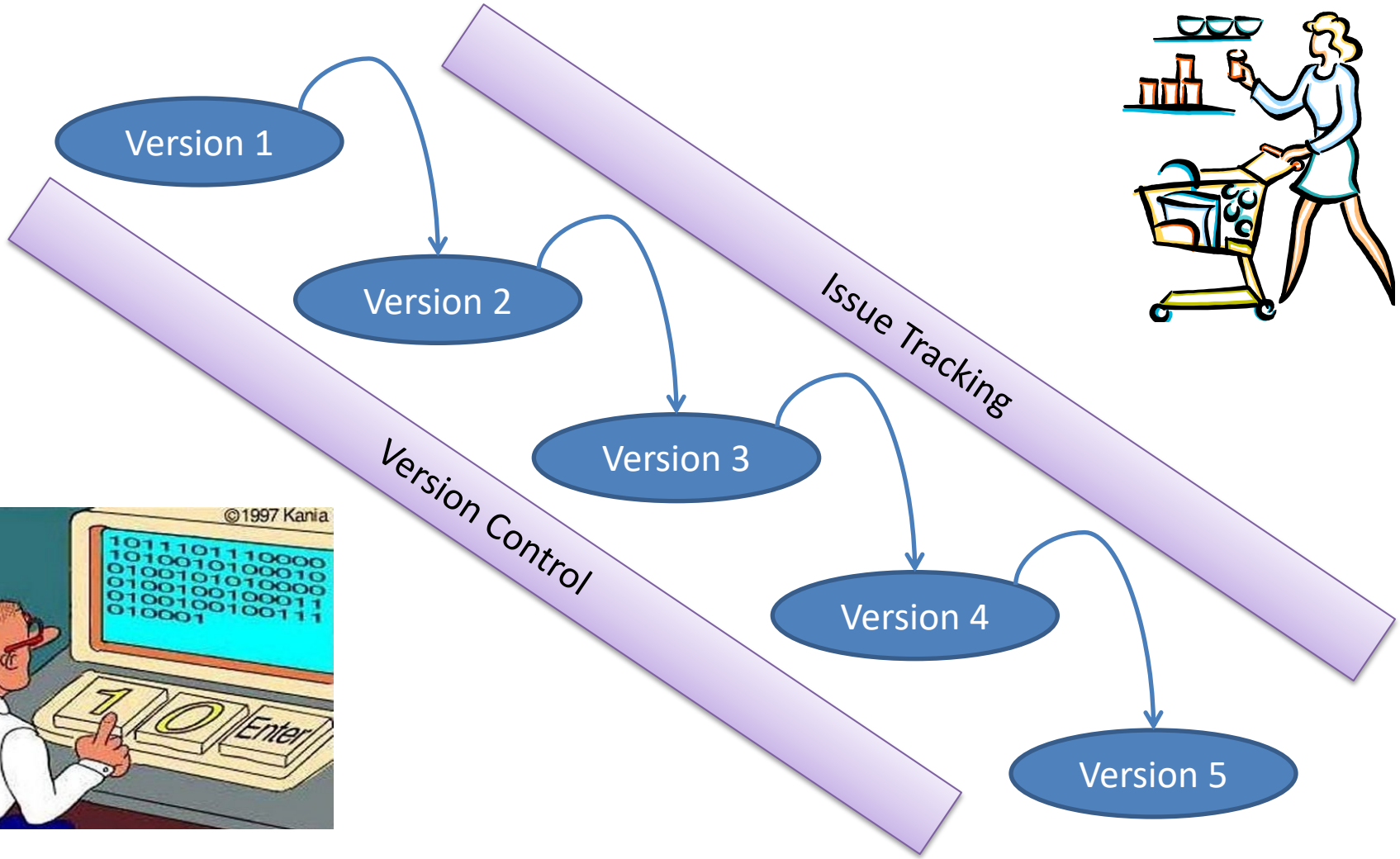
CM System



CM System

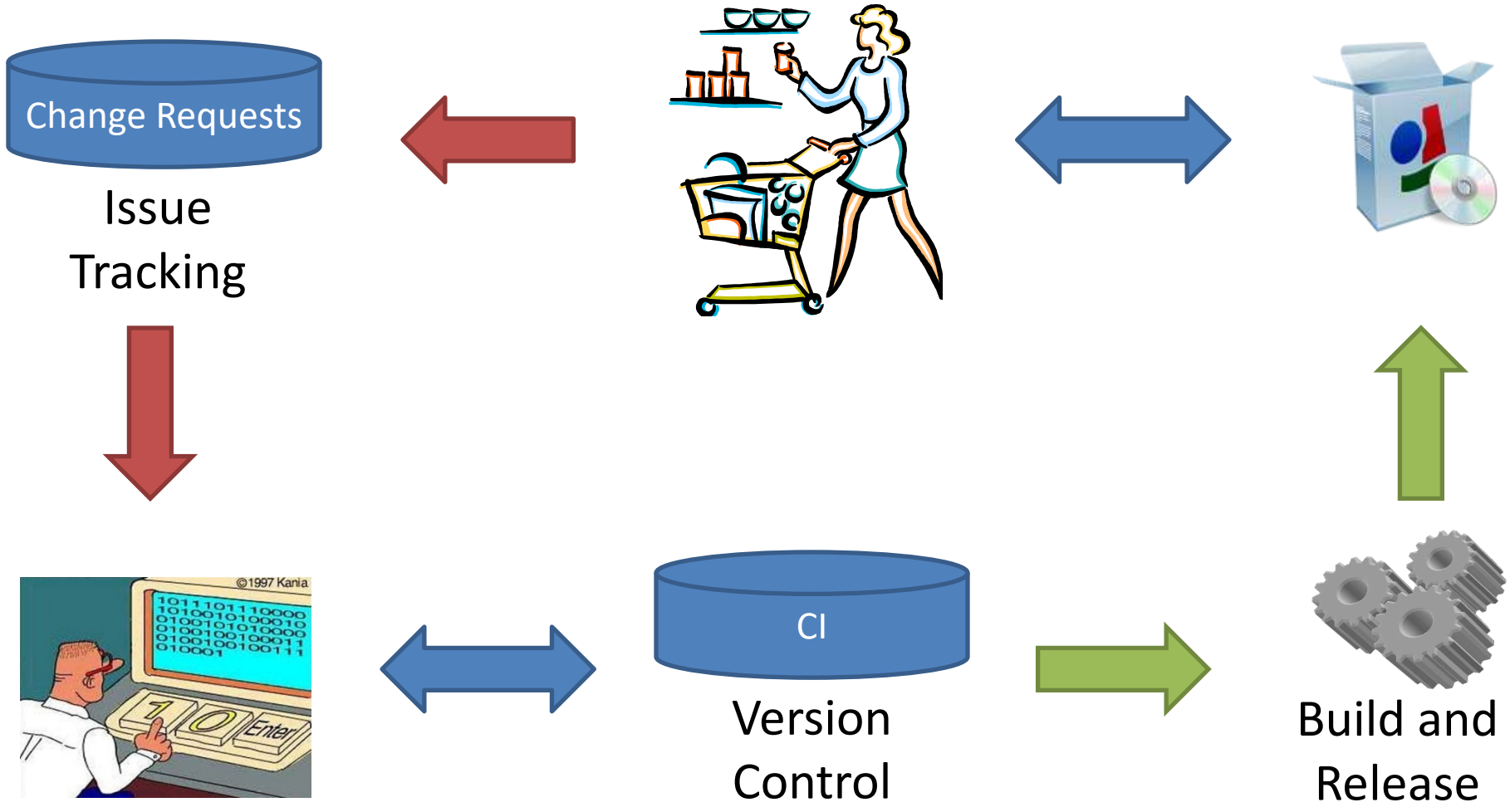

Version 1
Version 2
Version 3
Version 4
Version 5
Issue Tracking
Version Control

CM System

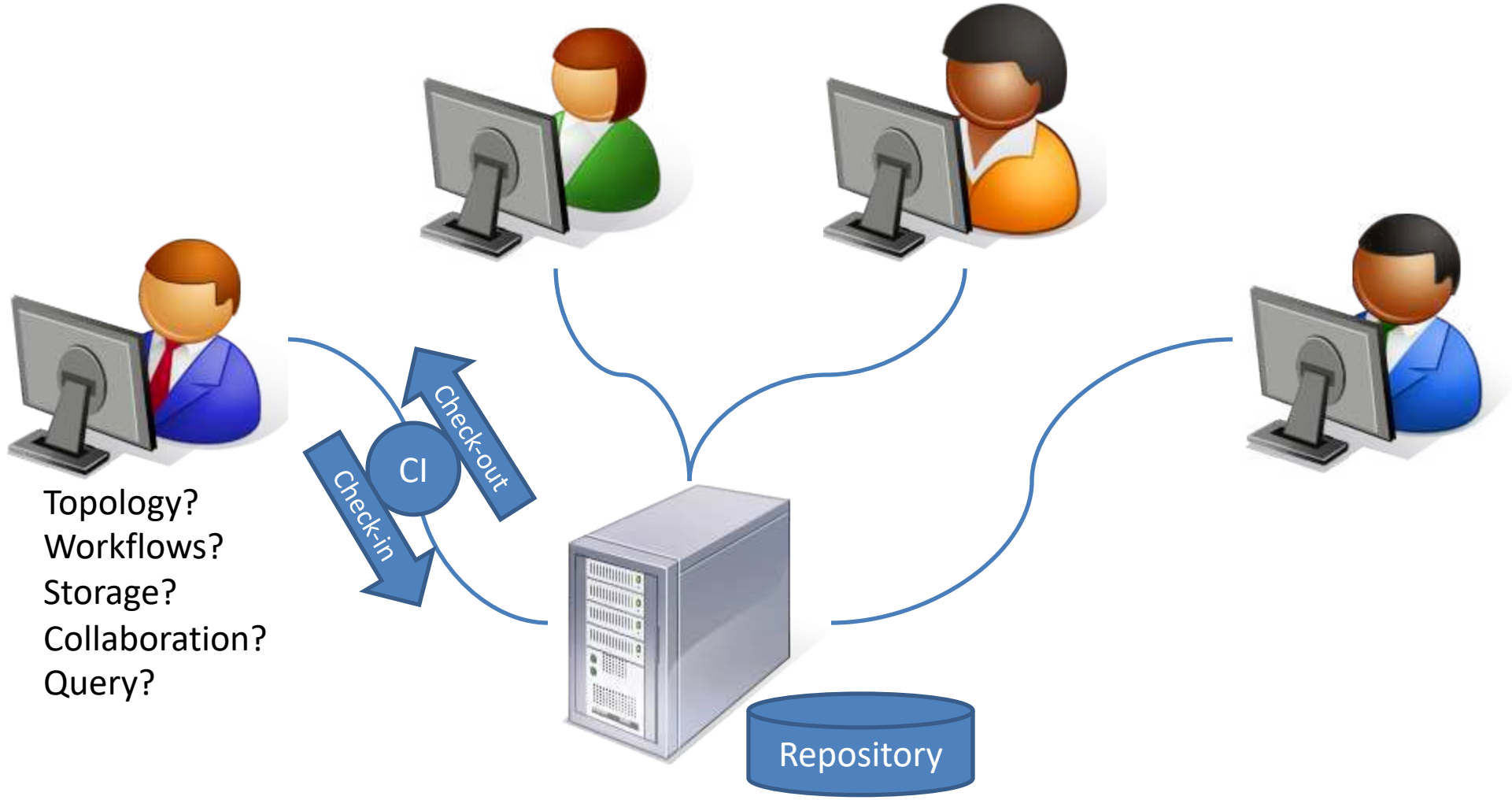

Version 1
Version 2
Version 3
Version 4
Version 5
Issue Tracking
Version Control

©1997 Kania
1
0
Enter

CM System

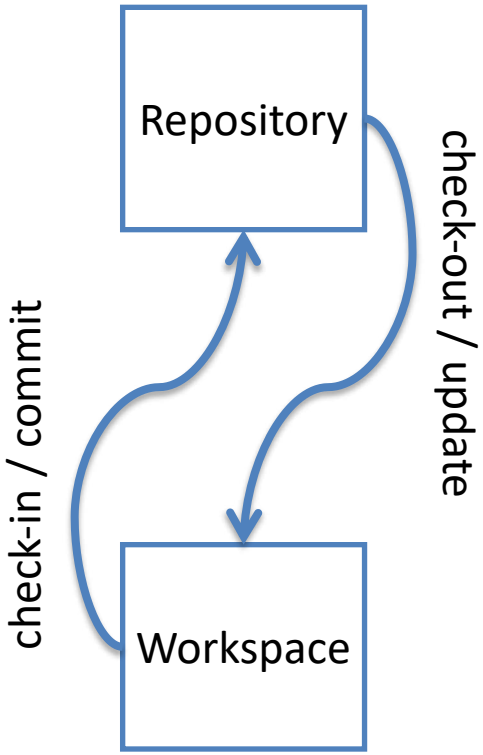

Change Requests
Issue Tracking
©1997 Kania
1
0
Enter
CI
Version Control
Build and Release

Version Control

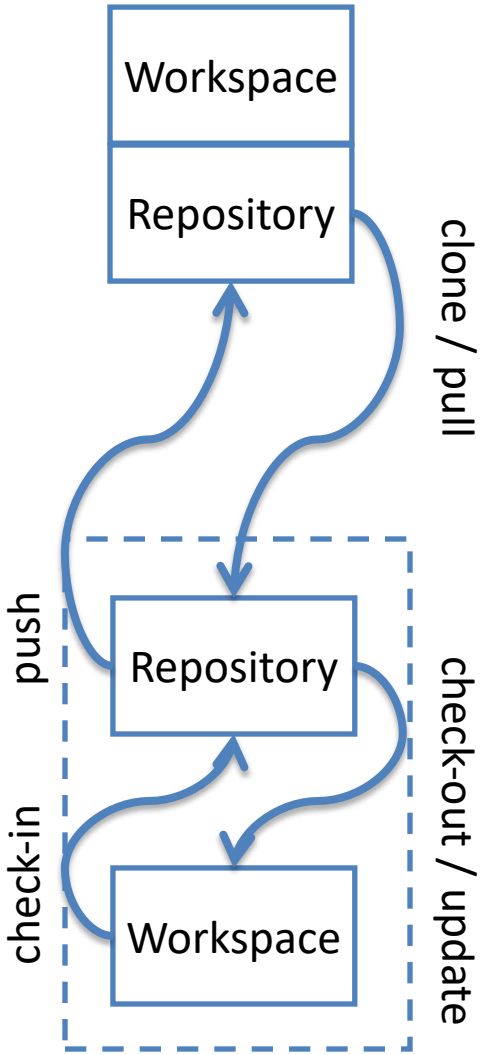


Topology?
Workflows?
Storage?
Collaboration?
Query?

Topology



Centralized

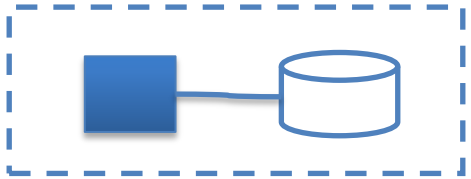


Distributed

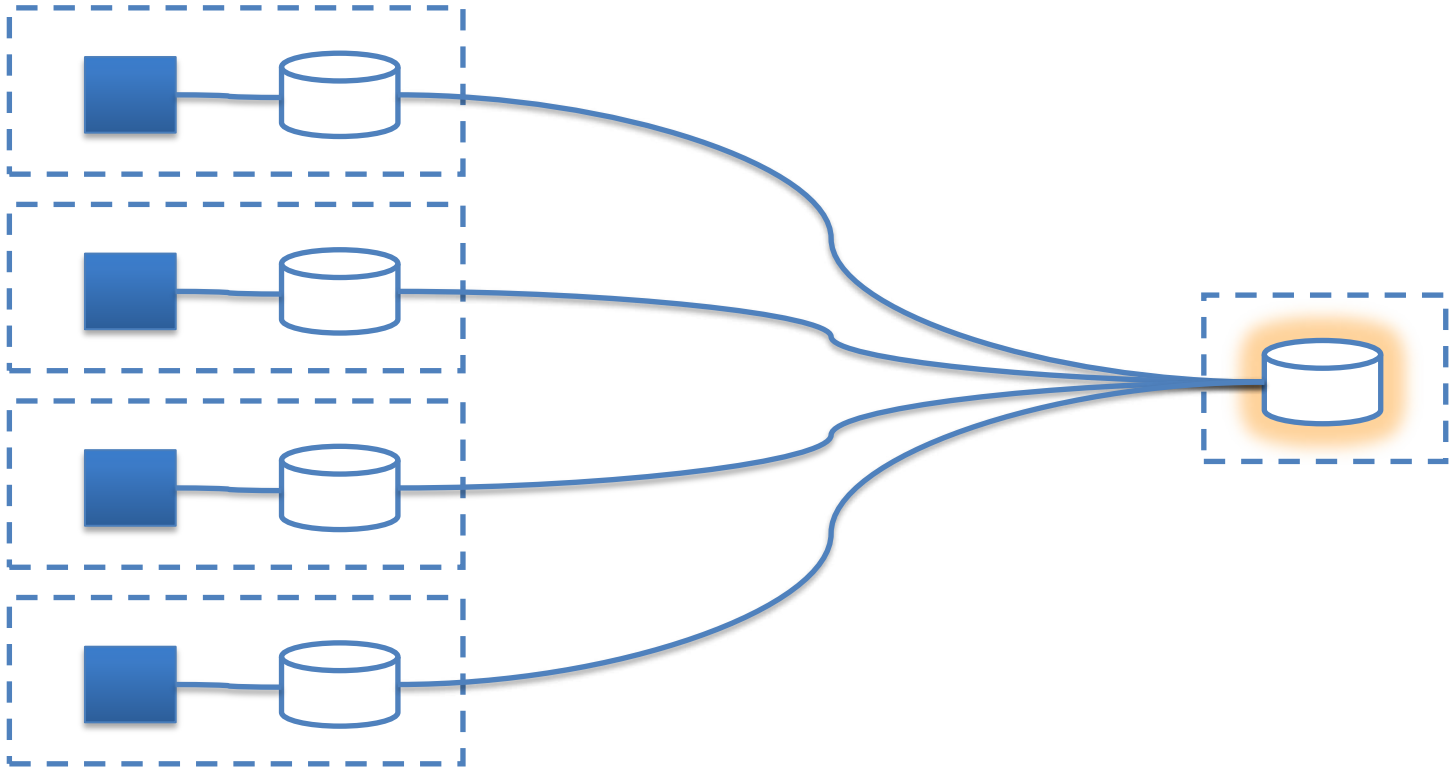
Workflows

- Peer-to-peer system may follow different workflows, according to the project characteristics
  - Individual
  - Client-server
  - Integration manager
  - Dictator/Lieutenants

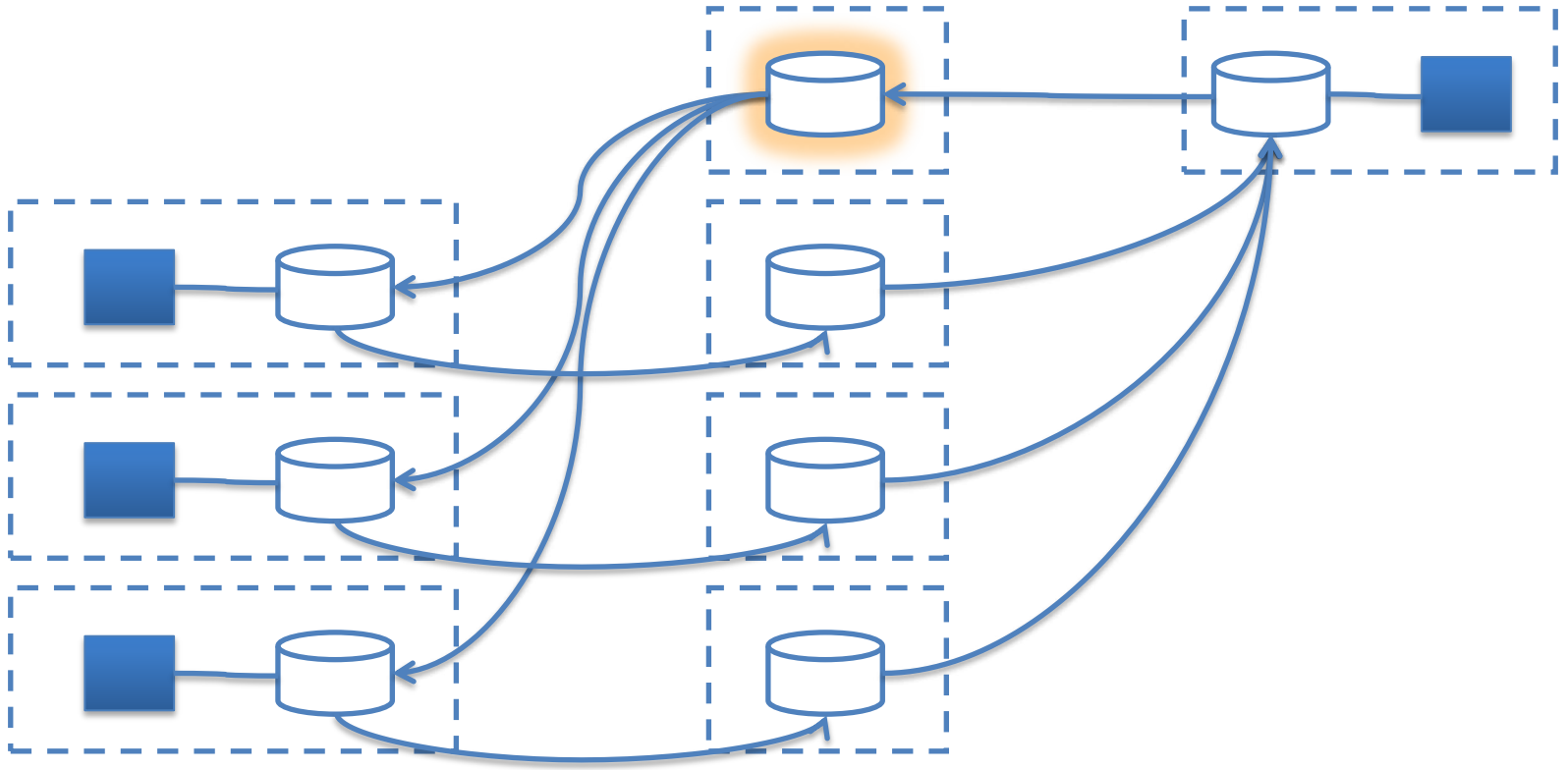
Individual



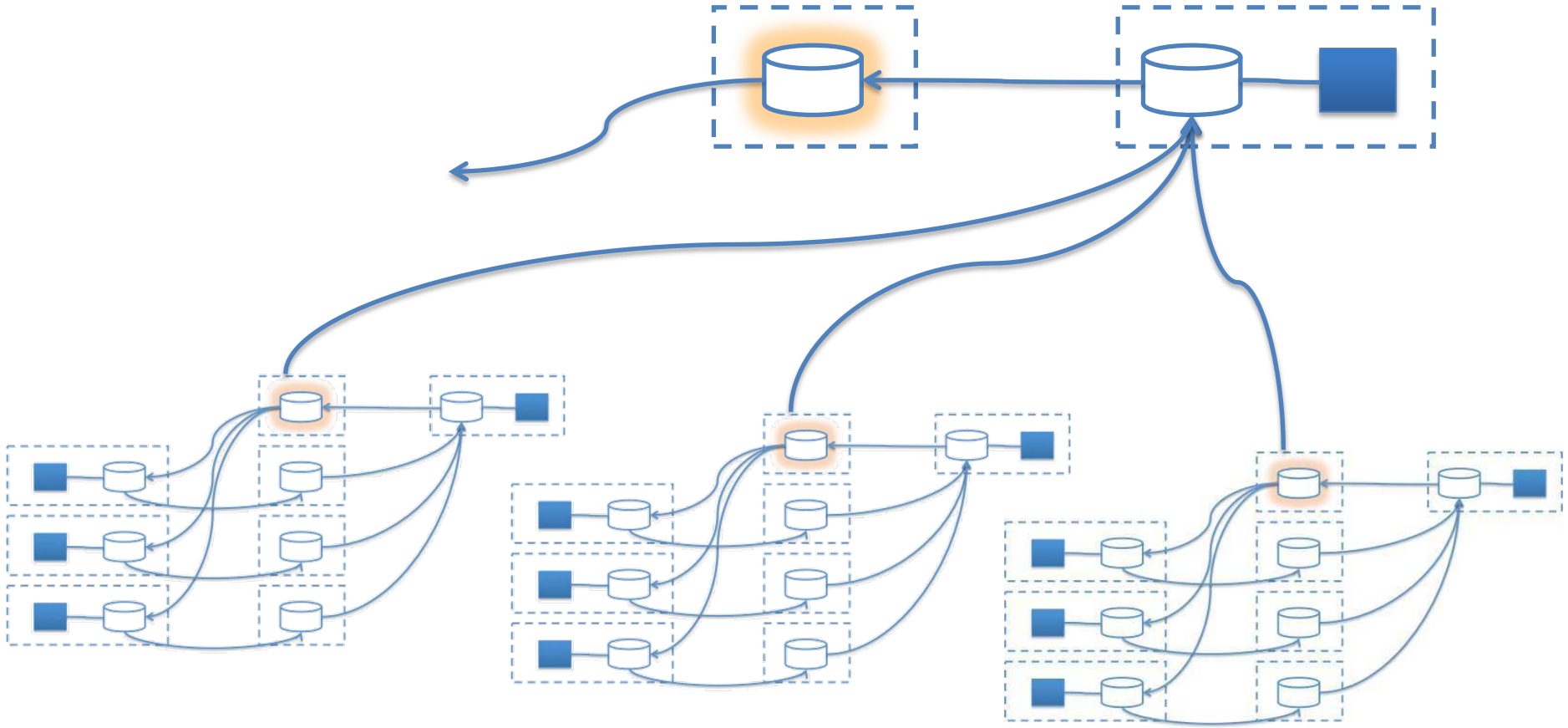
Client-server



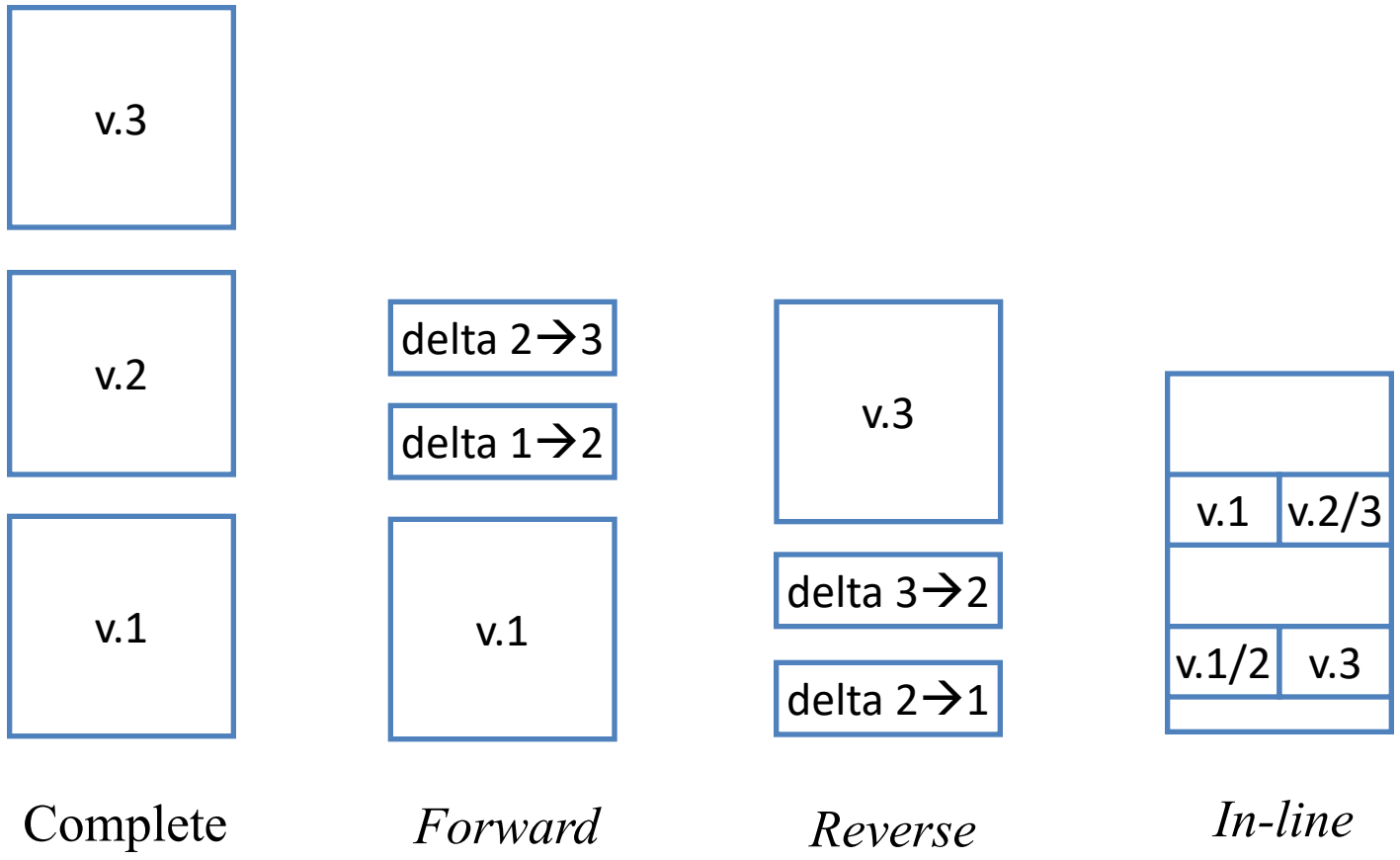
Integration manager (fork + pull request)



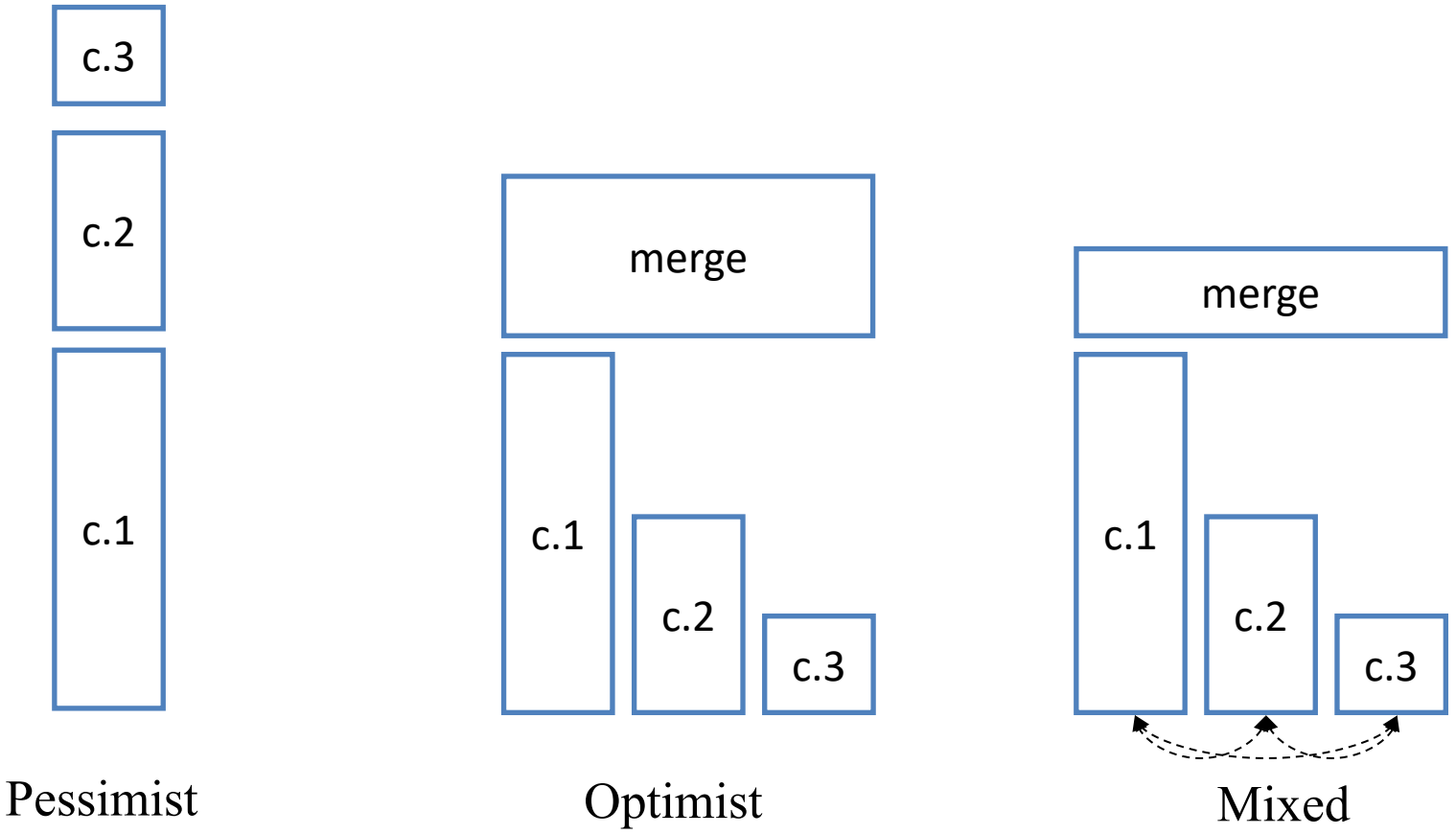
Dictator/Lieutenants (cascade pull requests)



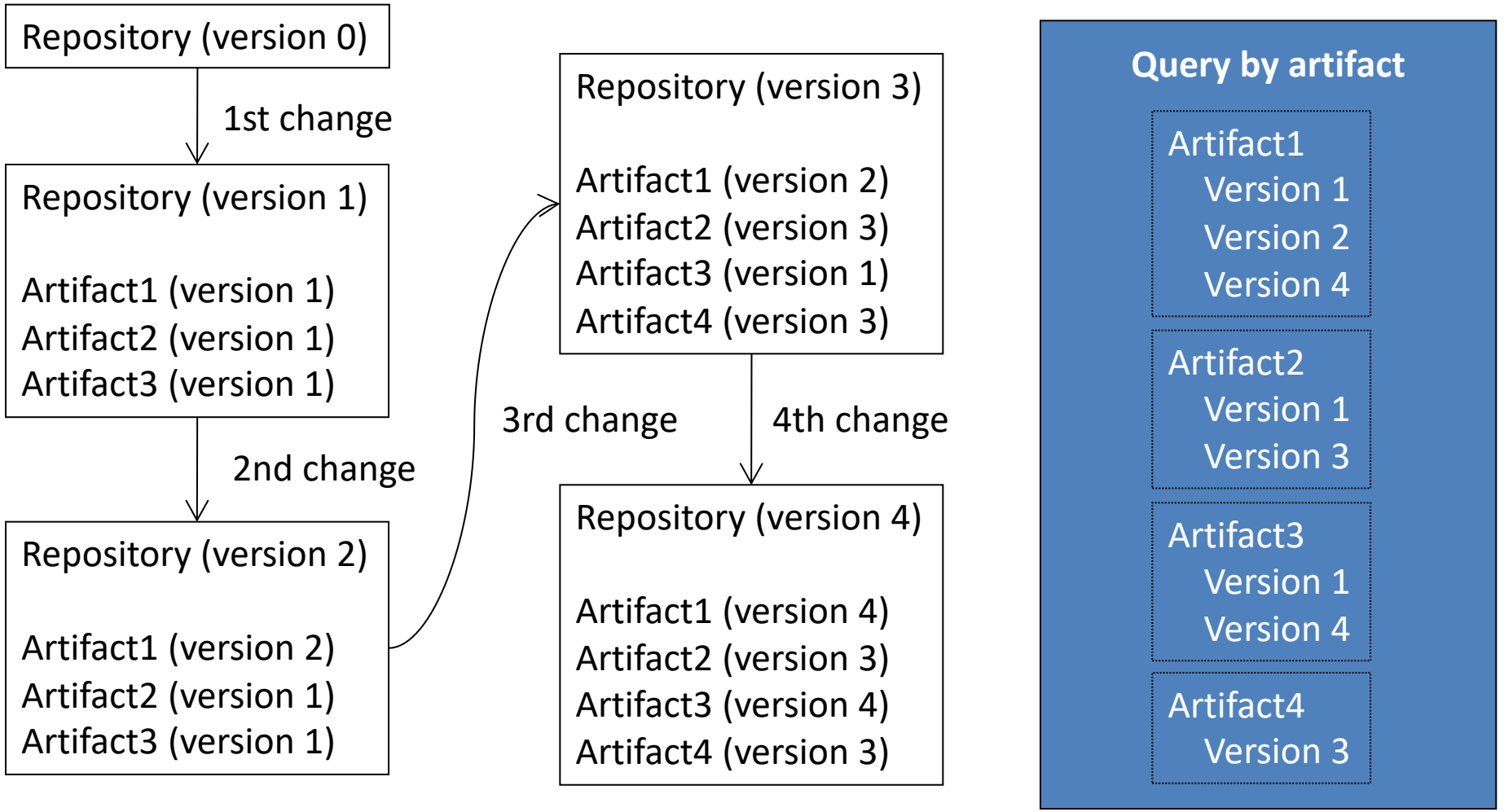
Storage


v.3
v.2
v.1
delta 2→3
delta 1→2
v.1
v.3
delta 3→2
delta 2→1
v.1
v.2/3
v.1/2
v.3
Complete
Forward
Reverse
In-line

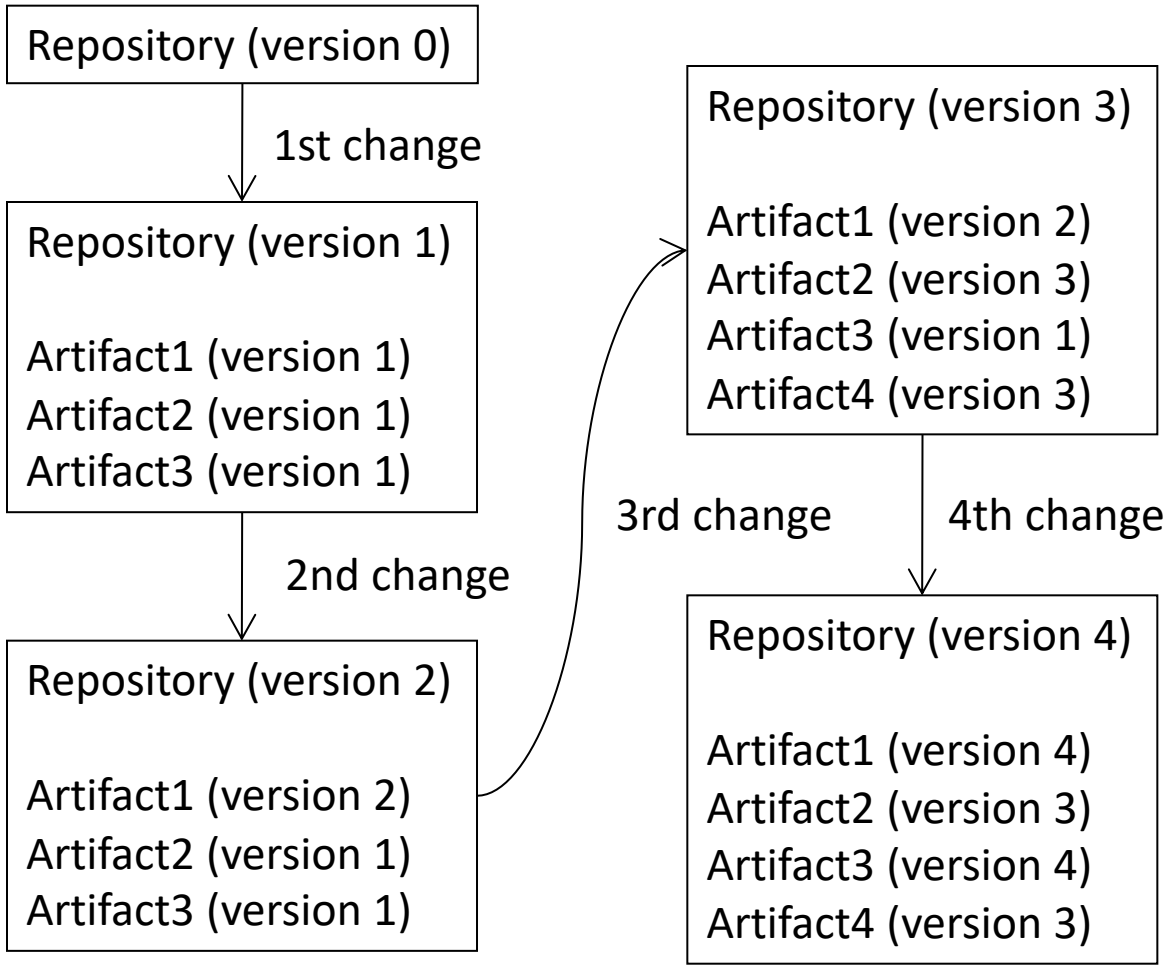
Collaboration



Query


Repository (version 0)
1st change
Repository (version 1)
Artifact1 (version 1)
Artifact2 (version 1)
Artifact3 (version 1)
2nd change
Repository (version 2)
Artifact1 (version 2)
Artifact2 (version 1)
Artifact3 (version 1)
3rd change
Repository (version 3)
Artifact1 (version 2)
Artifact2 (version 3)
Artifact3 (version 1)
Artifact4 (version 3)
4th change
Repository (version 4)
Artifact1 (version 4)
Artifact2 (version 3)
Artifact3 (version 4)
Artifact4 (version 3)
Query by artifact
Artifact1
Version 1
Version 2
Version 4
Artifact2
Version 1
Version 3
Artifact3
Version 1
Version 4
Artifact4
Version 3

Query



Query by change

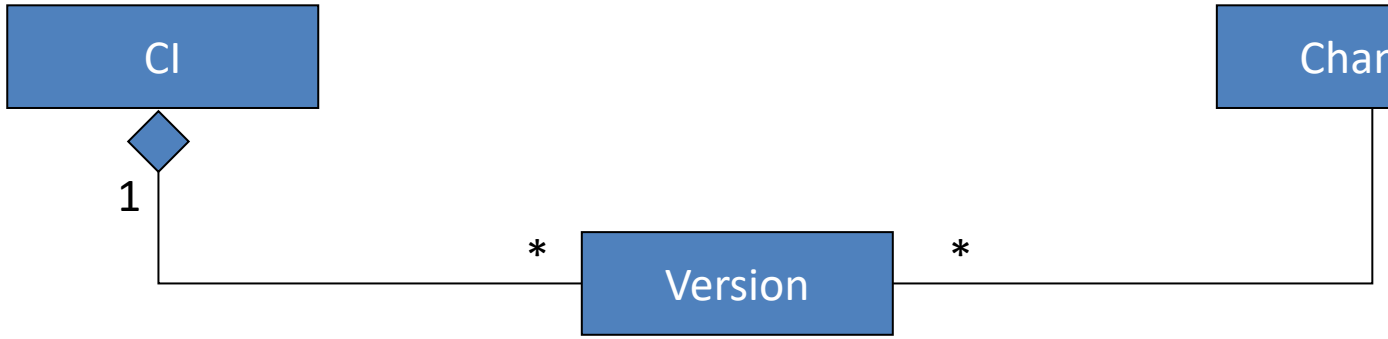
1st change
- Artifact1 added
- Artifact2 added
- Artifact3 added

2nd change
- Artifact1 changed

3rd change
- Artifact2 changed
- Artifact4 added

4th change
- Artifact1 changed
- Artifact3 changed

Query



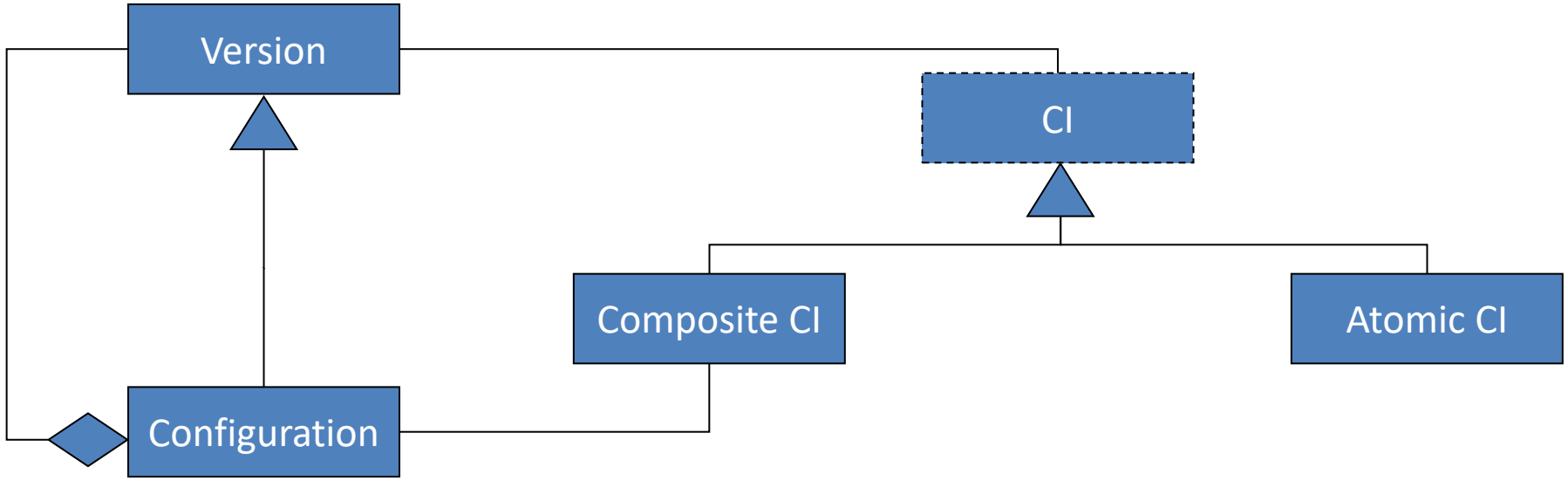
Artifact1
- Version 1
- Version 2
- Version 4

Change 4
- Artifact1
- Artifact3

Configuration

- A set of CI versions where there is one and only one version per CI
- A configuration can be seen as a CI composed by other CI
- Examples
  - System configuration
  - Process configuration
  - Module X configuration
  - Requirements configuration
  - Source-code configuration

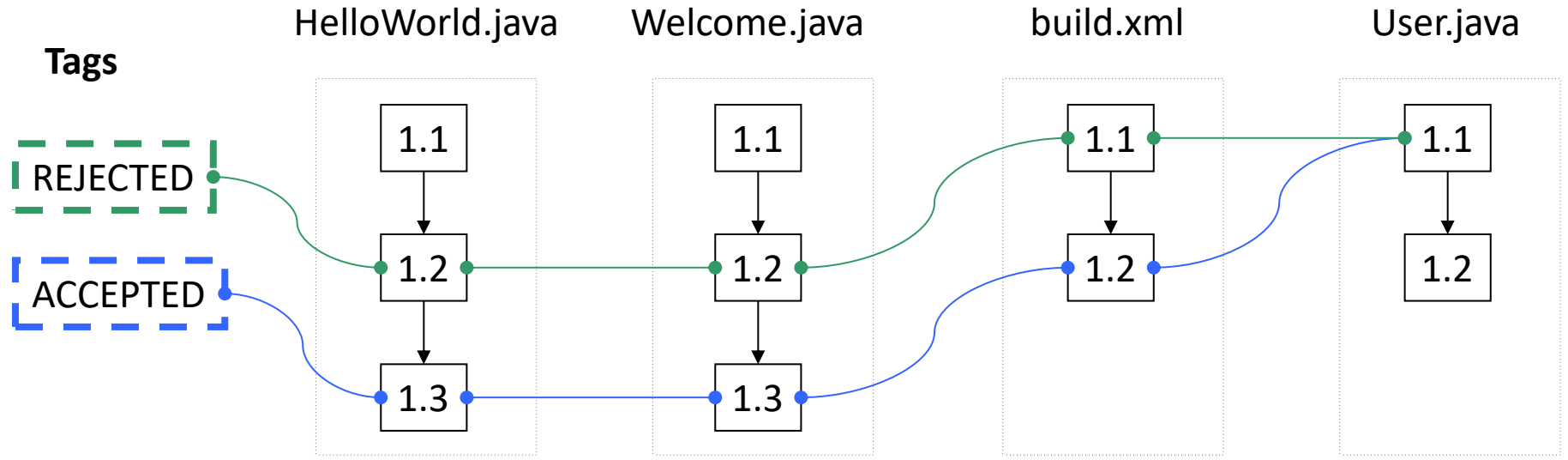
Configuration vs. version



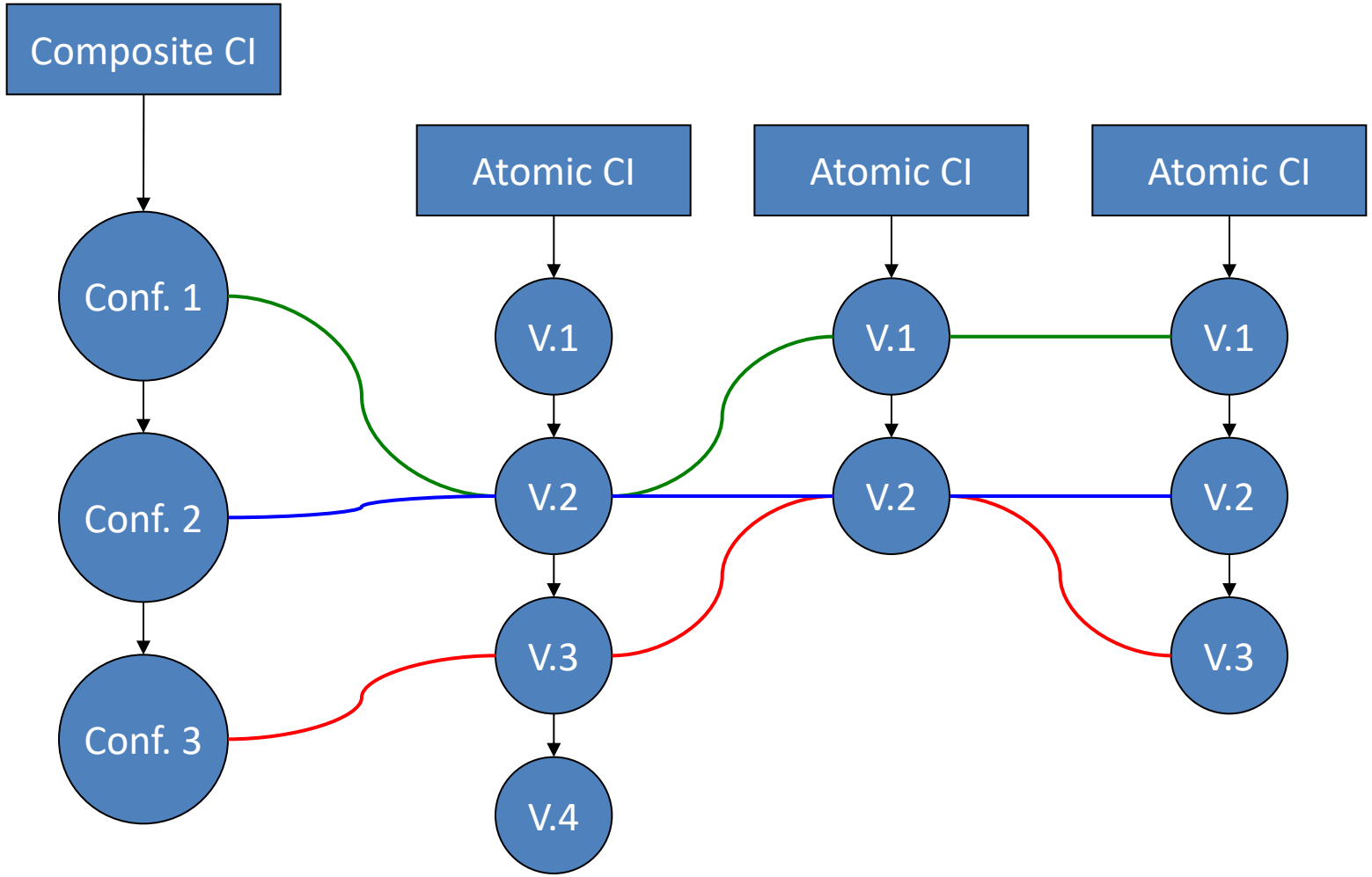
- Generically speaking
  - The system S is composed by CI X, Y, and Z
- Concretely speaking
  - The configuration 5 of system S is composed by version 2 of CI X, version 4 of CI Y, and version 6 of CI Z

Tag

- VCS usually register multiple configurations, but just few are of interest to the user
- Tags allow naming such configurations
- Names can be user to indicate versions, quality levels, etc.



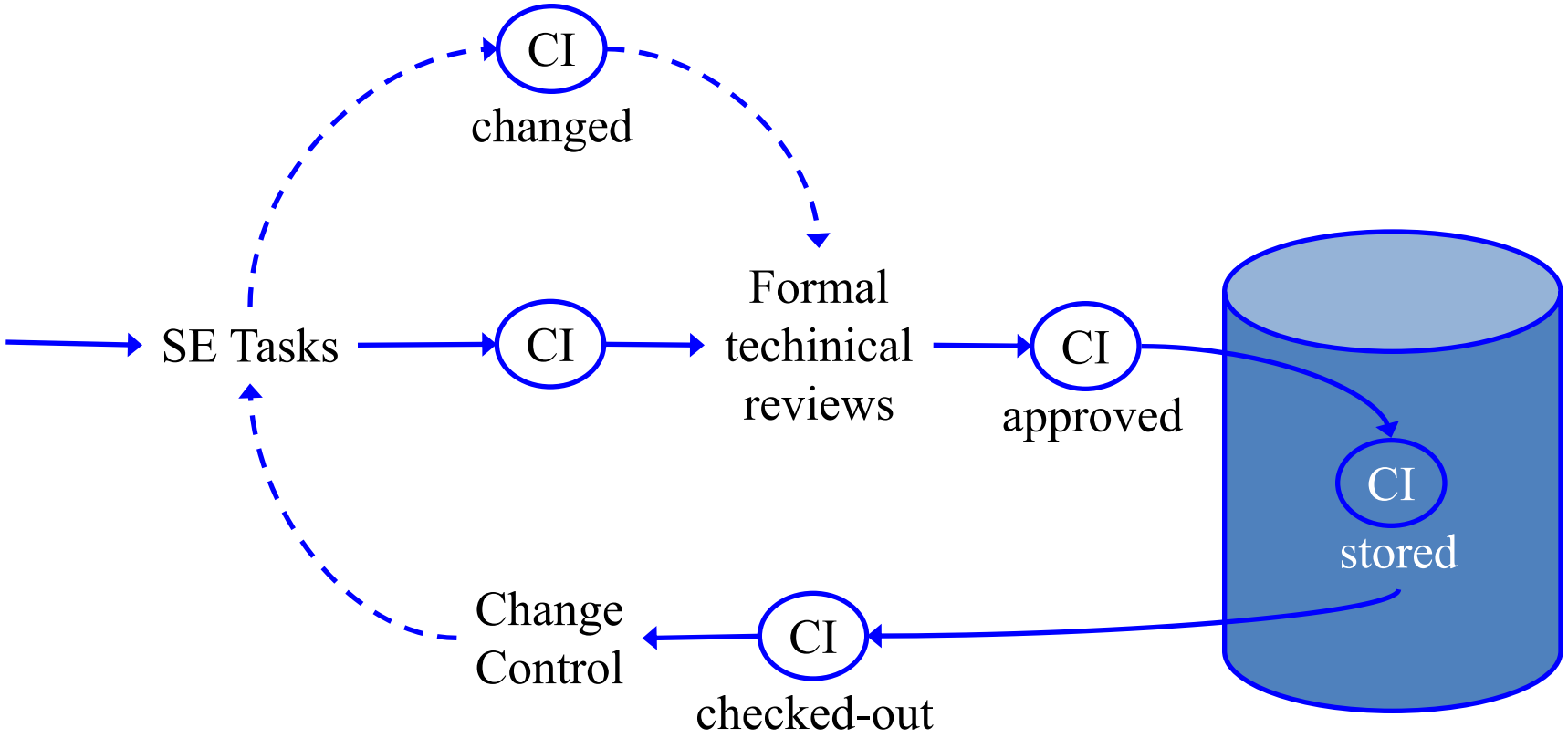
Configuration vs. version



Baseline

- "A specification or product that has been **formally reviewed and agreed upon**, that thereafter **serves as the basis for further development**, and that can be **changed only through formal change control procedures**" (IEEE 610.12, 1990).
- Baselines are created at the end of each development phase: analysis (functional), design (allocated), and coding (product)
- When is the correct moment for creating baselines?
  - Control vs. Bureaucracy

Baseline



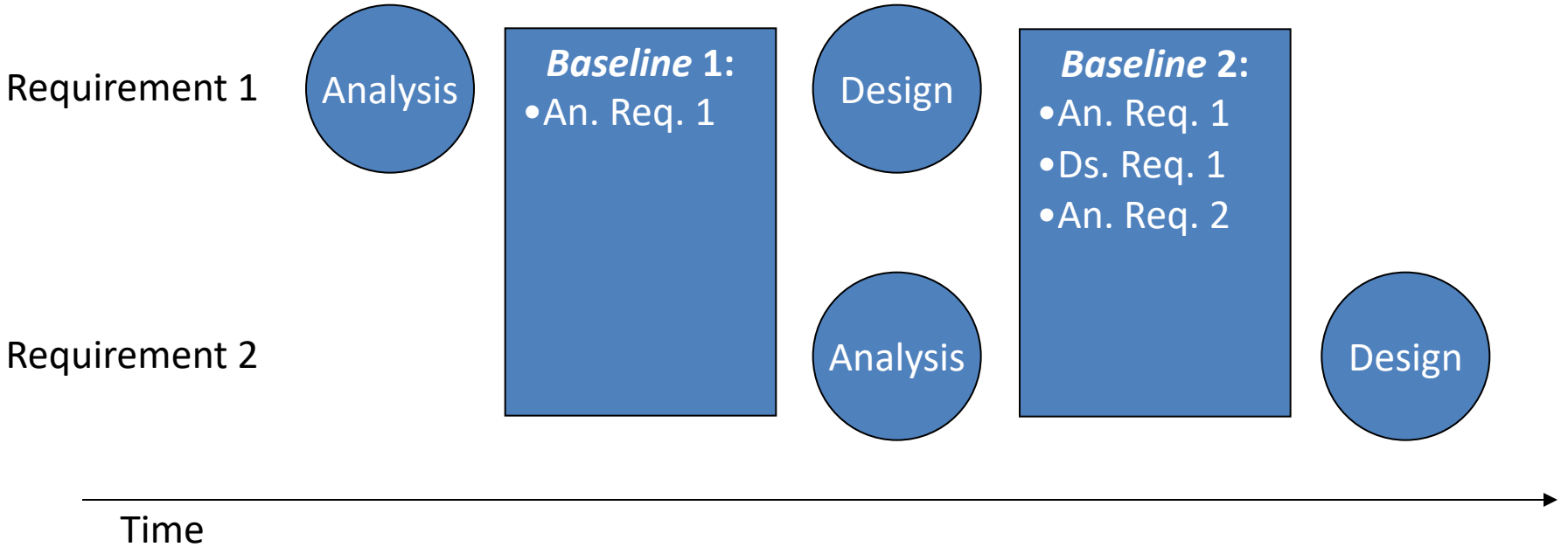
[Pressman, 1997] Baseline update process

Baseline (levels of control)

Pre *baseline*:
- Informal
- Without request
- Without evaluation
- Without verification
- Agile
- Ad-hoc

Post *baseline*:
- Formal
- With request
- With evaluation
- With verification
- Bureaucratic
- Planned

Baseline (levels of control)



Req.
1
2

Analysis	Design
Inform.	-
-	-

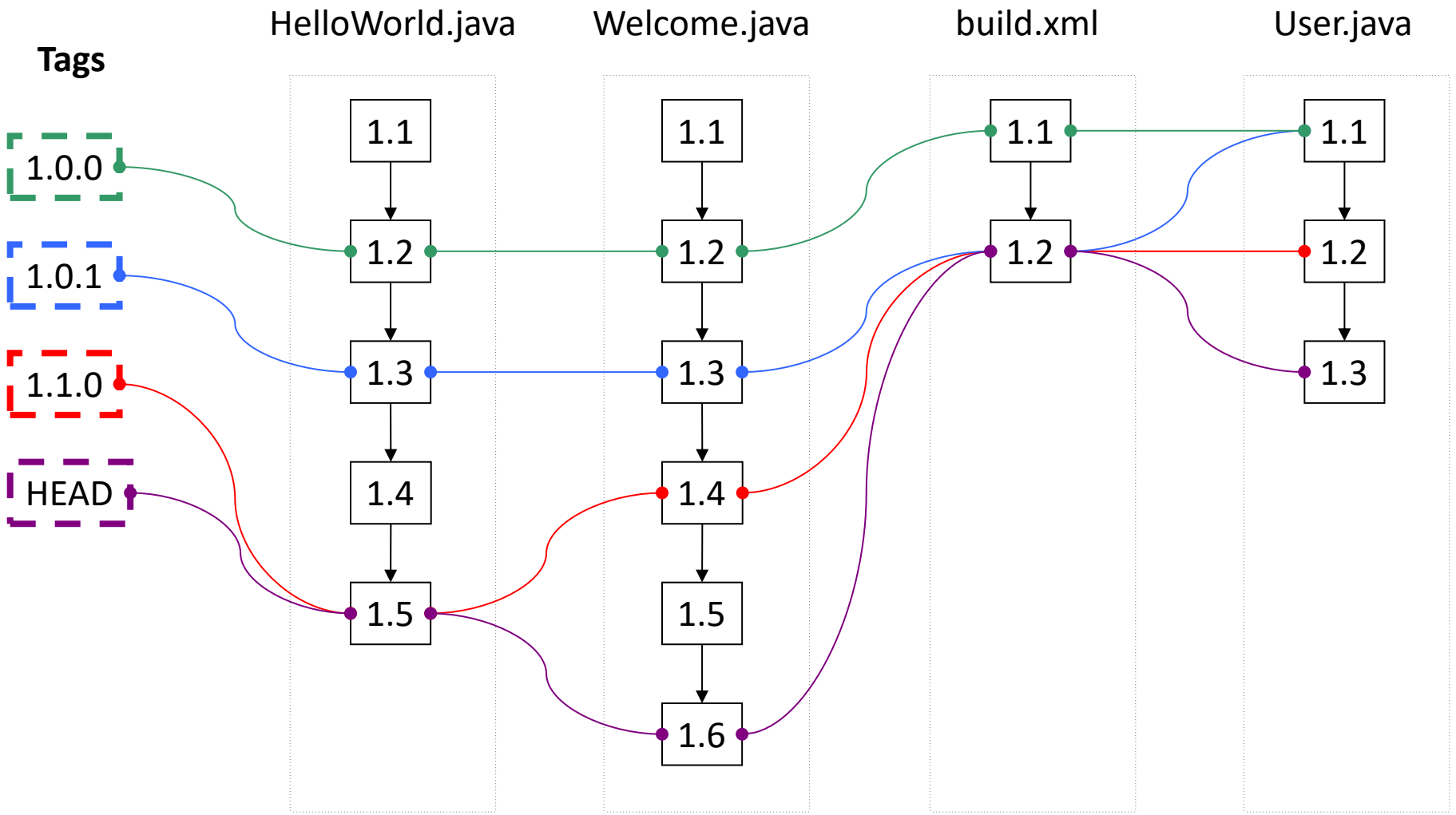
Analysis	Design
Formal	Inform.
Inform.	-

Analysis	Design
Formal	Formal
Formal	Inform.

Release

- Noun: Version provided for a specific purpose
- Verb: Formal notification and distribution of a version (usually baseline)
- All release are versions, but not all versions are released
- Sometimes, releases may be developed in parallel due to time to market
- Examples
  - Test release
  - Staging release
  - Product release

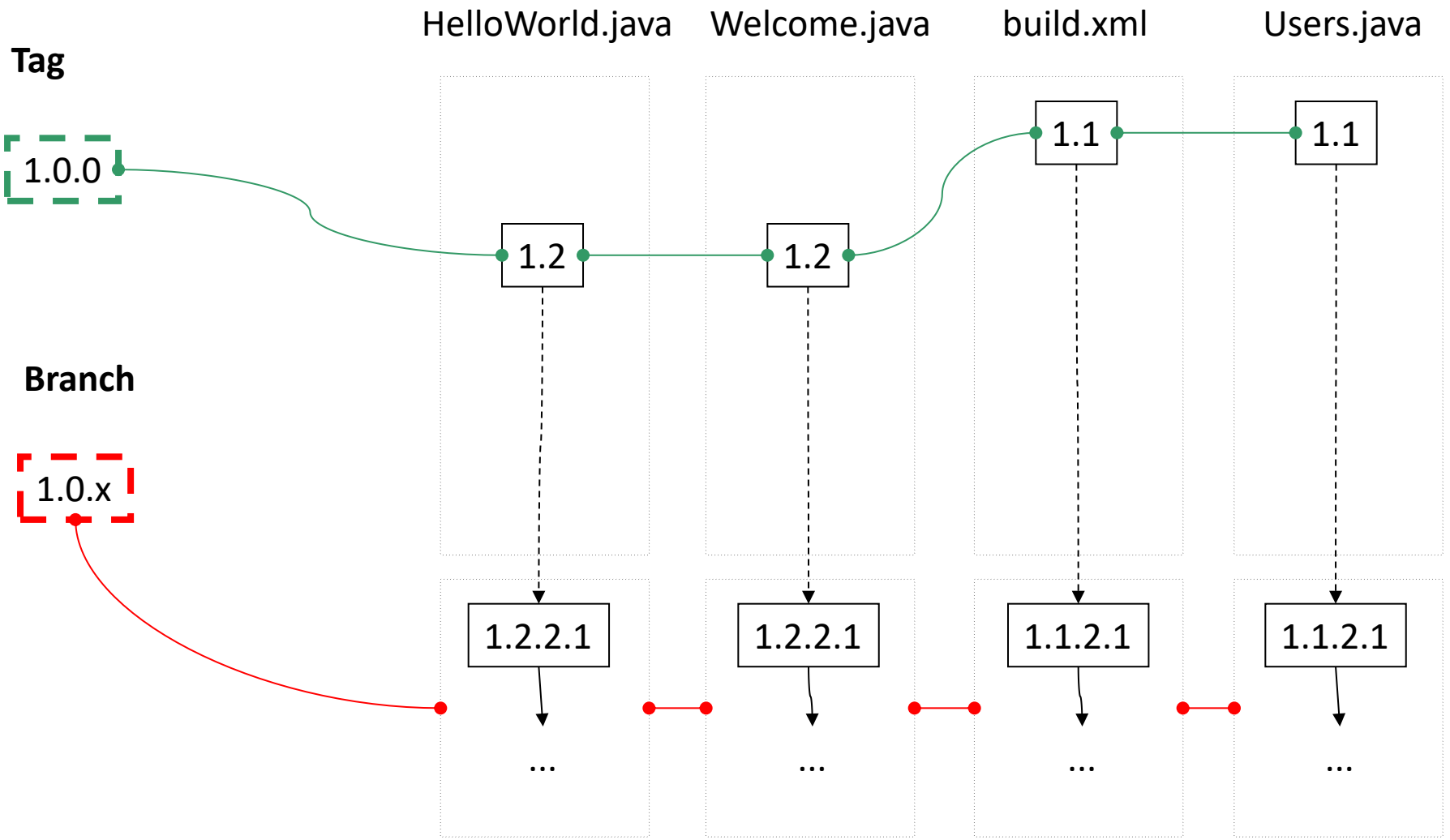
Tags naming releases



Branches

- Versions that deviate from the main development line
- Allow isolation to the development process
  - Usually reintegrated to the main development line
  - The reintegration sometimes is a difficult process
- Workspaces (CVCS) can be seen as a temporary branch
- Clones (DVCS) can be seen as repository forks that may lead to branches if parallel development occurs

Branch example

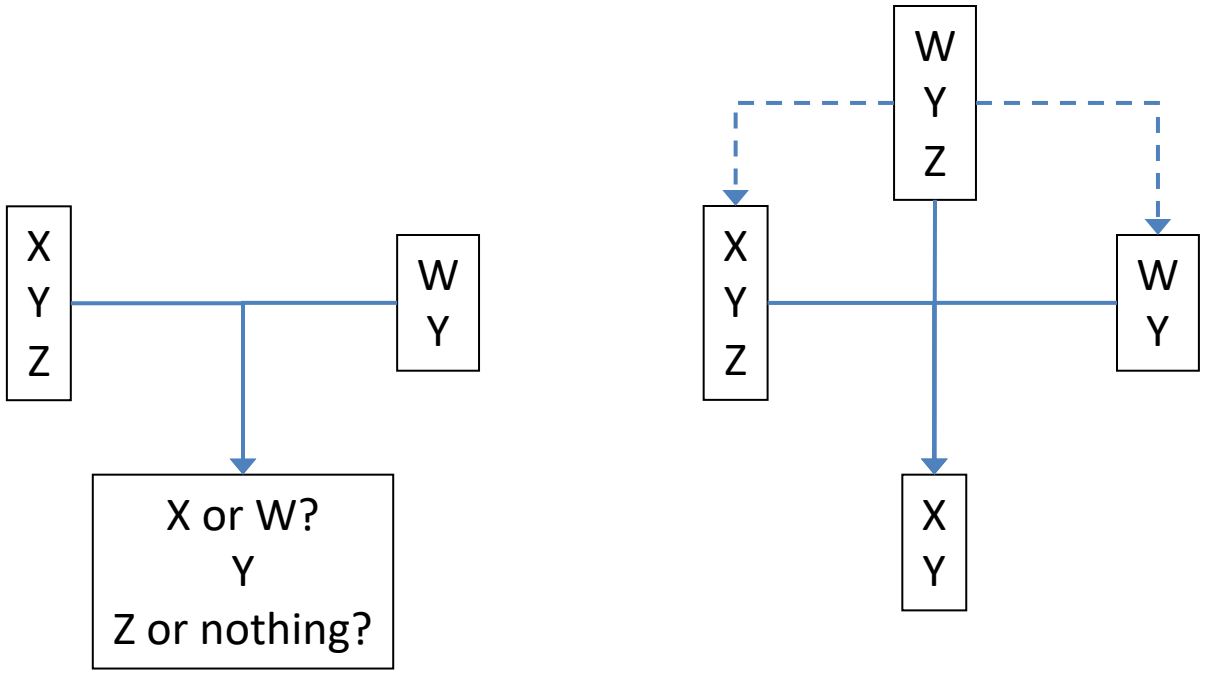


Merge

- Help on reintegrating
  - Workspaces
  - Branches
- It is necessary even when pessimistic concurrency control (lock) is in place, due to branches
- Automatic algorithms categories
  - Generic (work with all programming languages)
  - Language Specific (take into consideration the syntax and semantics of the programming language)

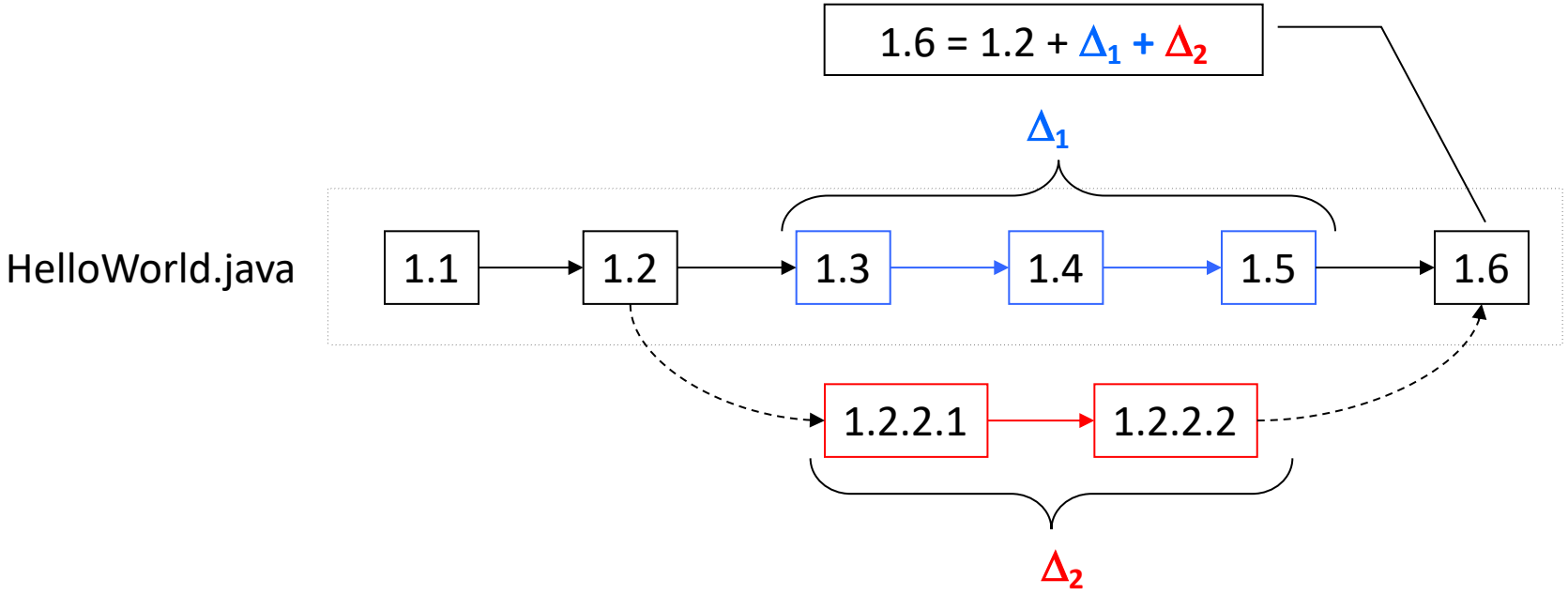
Merge

- Types of generic merge algorithms
  - 2-way merge
  - 3-way merge



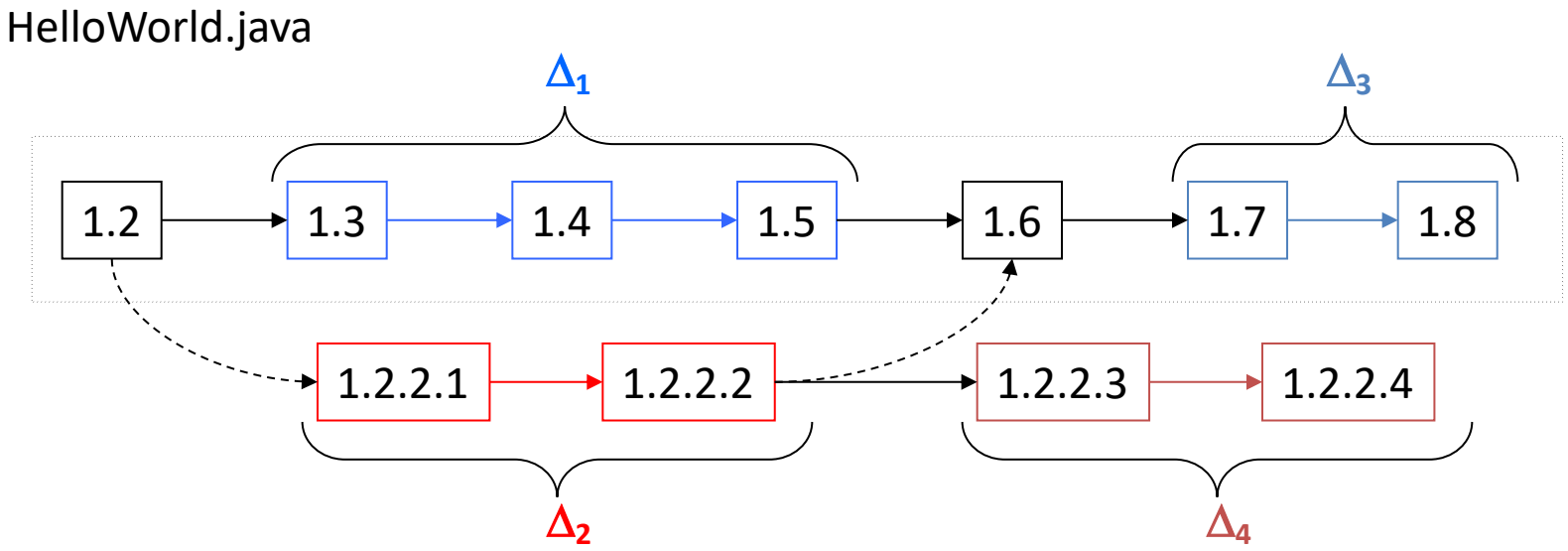
Merge

- Merge occurs for each CI in the branch
- All changes since the common ancestor are taken into consideration



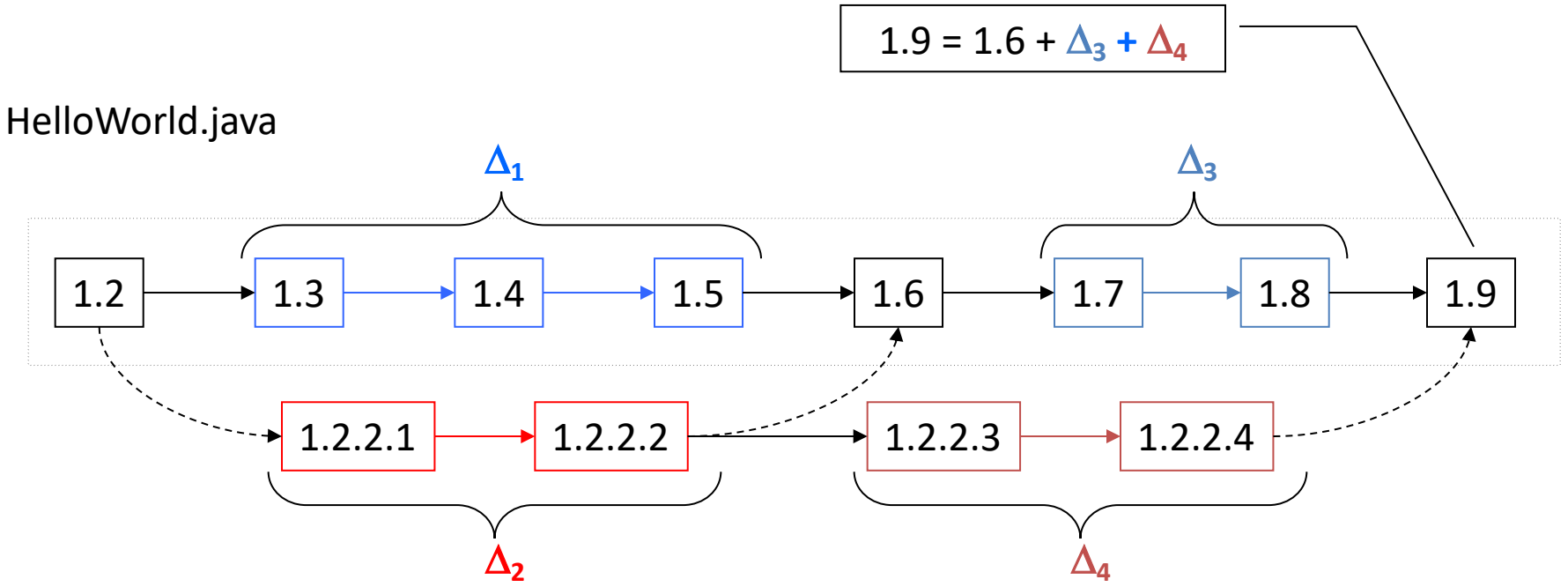
Merge

- Merge can be done incrementally



Merge

- Merge can be done incrementally



Conflicts

- Situations where it is not possible to perform automatic merge
- Types
  - Physical (line of a file)
  - Syntactic (elements of the file grammar)
  - Semantic (dependencies among elements)
- Current tool support focus on the physical level!
- Examples of physical conflicts
  - Parallel change in the same line
  - Parallel change and deletion of the same line
  - Parallel additions of lines in the same file region

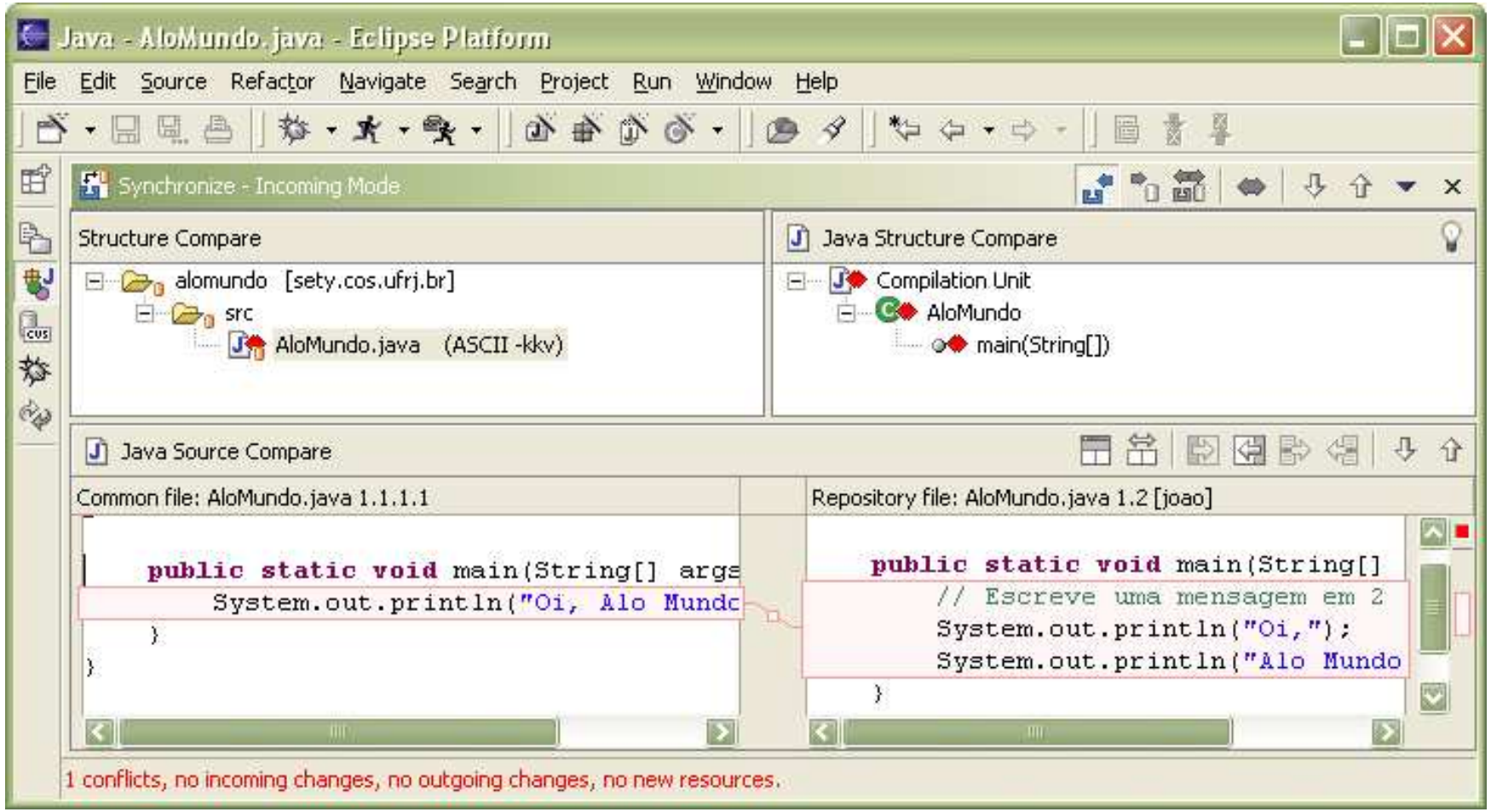
Raw conflict demarcation

~/Meus documentos/sandbox/alomundo/src

```
public class AloMundo {

        public static void main(String[] args) {
<<<<<<< AloMundo.java
                System.out.println("Oi, Alo Mundo!!! Tchau.");
=======
                // Escreve uma mensagem em 2 linhas
                System.out.println("Oi,");
                System.out.println("Alo Mundo!!! Tchau.");
>>>>>>> 1.2
        }
}
~
~
~
~
~
~
~
~
~
~
~
~
"AloMundo.java" [dos format] 12 lines, 287 characters
```

IDE conflict resolution



References

- Leon A., “Software Configuration Management Handbook”, Artech House, 1st ed., 2004.
- Chacon S., “Pro Git”, 2nd ed., 2014.

Instituto de Computação

Foundations

Leonardo Gresta Paulino Murta

leomurta@ic.uff.br