CHANGE MANAGEMENT (CM)*

Overview

Change Management

Version Management (Source Code Control)

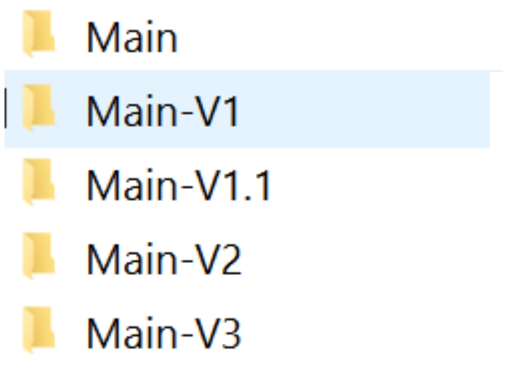
*Some of this material adapted from Ian Sommerville's course notes, Ch. 25, and Wigerd, L. and Seiwald, 1998

WHAT DID YOU DO…

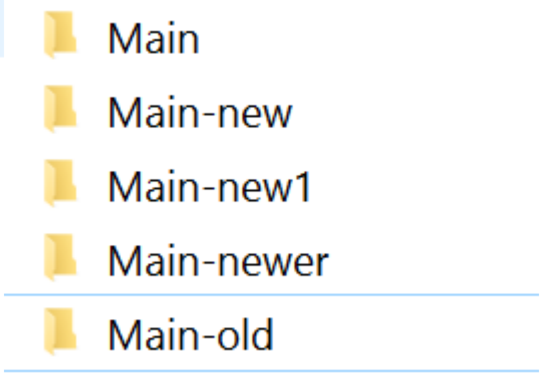
…when you collaborated on your first team paper or project at ASU?

- Gmail? Dropbox? Google site?
- Have you used "track changes"?
- How did you manage experimentation?
- Did you ever just want to "undo" something?

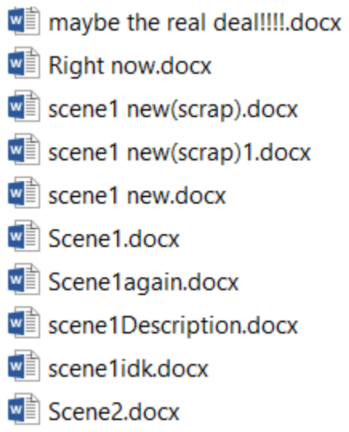
DOES THAT LOOK FAMILIAR?



Main
Main-V1
Main-V1.1
Main-V2
Main-V3



Main
Main-new
Main-new1
Main-newer
Main-old



maybe the real deal!!!!.docx
Right now.docx
scene1 new(scrap).docx
scene1 new(scrap)1.docx
scene1 new.docx
Scene1.docx
Scene1again.docx
scene1Description.docx
scene1idk.docx
Scene2.docx

If it doesen't, good for you!!!

'Local Source Code Management':

- Copy/Pasting folder and renaming
- Complicated if you are not good/consistent/exact with names
- You might change something in the wrong folder
- You cannot track each change

DROPBOX, CLOUD ETC.

- Store on Server
- Access from any device/browser
- Sync automatically
- Share folders/files
- Easy access
- Easy setup
- Version Control
  - Keeps track of changes (to a certain point)
  - Knows who last changed a file

- Pros:
  - Good for sharing pics, files, etc.
  - Easy to use

- Cons:
  - Hard for Software projects
  - Might lead to not working software, because everything is always synchronized

DROPBOX/COUD ETC. FOR SOFTWARE PROJECTS

- Problems with Clouds and SE:
  - Projects often have many developers
  - Work in progress code should not be in cloud
  - Might want to create special release versions
  - Code changes should be commented
  - Code changes should easily be visible

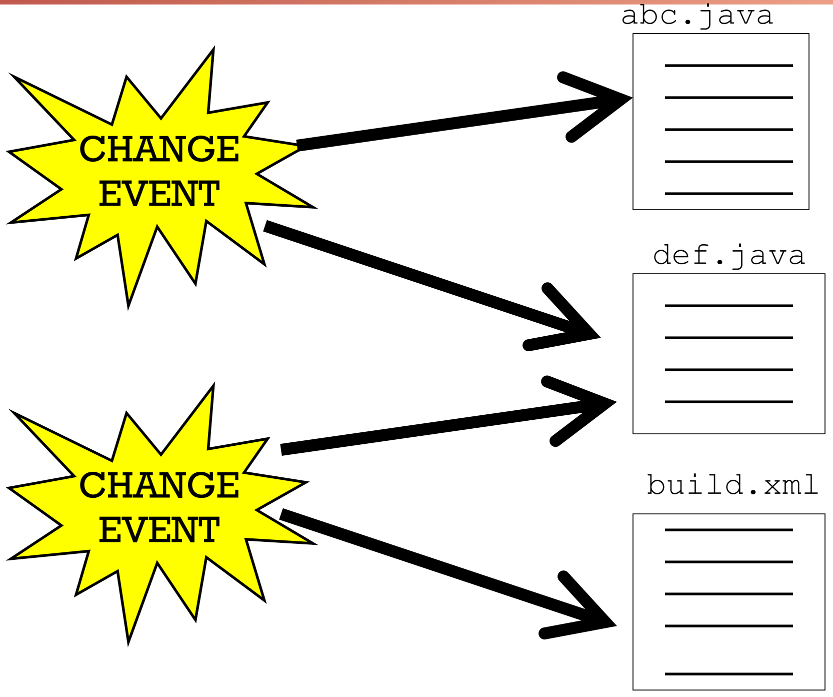
SOFTWARE

- Always changes (especially in Development)
- Often many people work on same software/same files
- Dropbox a good place to do that?

CHANGE MANAGEMENT

CHANGE MANAGEMENT

What happens when change happens?



Change management

Keeping track of *requests for changes* to the software from customers and developers, working out the *costs and impact of changes*, and *deciding changes* to be implemented.

Configuration management (CM)

Keeping track of how software components and artifacts are assembled, including what versions, how they are configured, and associated metadata to inform a release

Version management (Source Code Control)

Keeping track of the *multiple versions of system components* and ensuring that changes made to components by different developers do not interfere with each other.

CHANGE MANAGEMENT

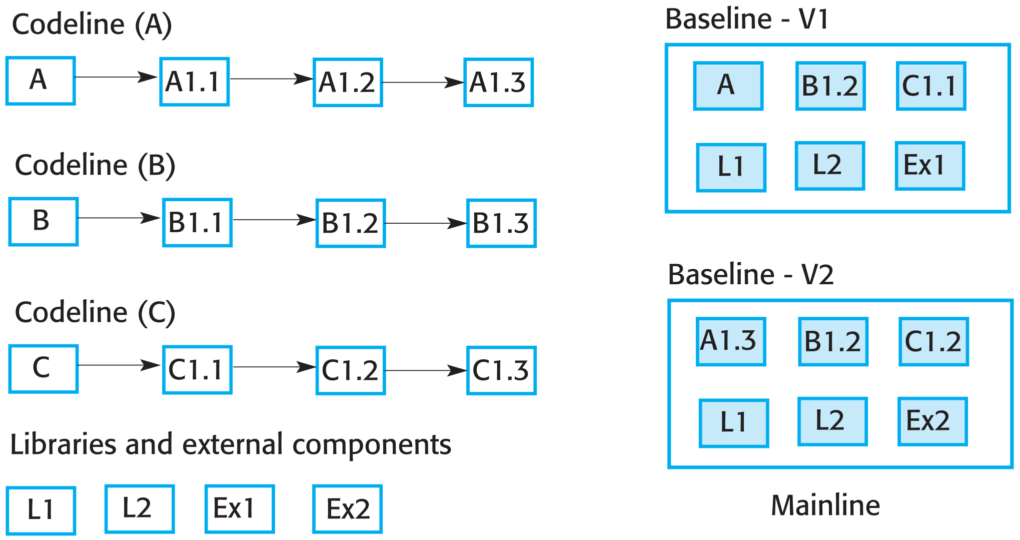
- **Change happens**!
  - Every unit of work requires changing some system artifact
- Many reasons for change:
  - Business opportunity presents itself
  - Incomplete and ambiguous requirements
  - New technology
  - *…and a zillion other reasons*
- Change Management processes identify
  - <u>What</u> system artifacts changed (which new artifact version)
  - <u>Why</u> it needed to be changed (which task caused the artifact change)
  - <u>Who</u> made the change and when it occurred (audit-ability)
- Change Management processes
  - Traditionally requires <u>traceability</u> and a management tool
  - Must inform stakeholders (often there is a <u>CCB – Change Control Board</u>)
  - *Agile says to <u>embrace it</u>*

CONFIGURATION MANAGEMENT

Configuration Management is a management of software artifacts: (component) assembly and configuration

Codelines define a *trajectory* for [source code] artifacts

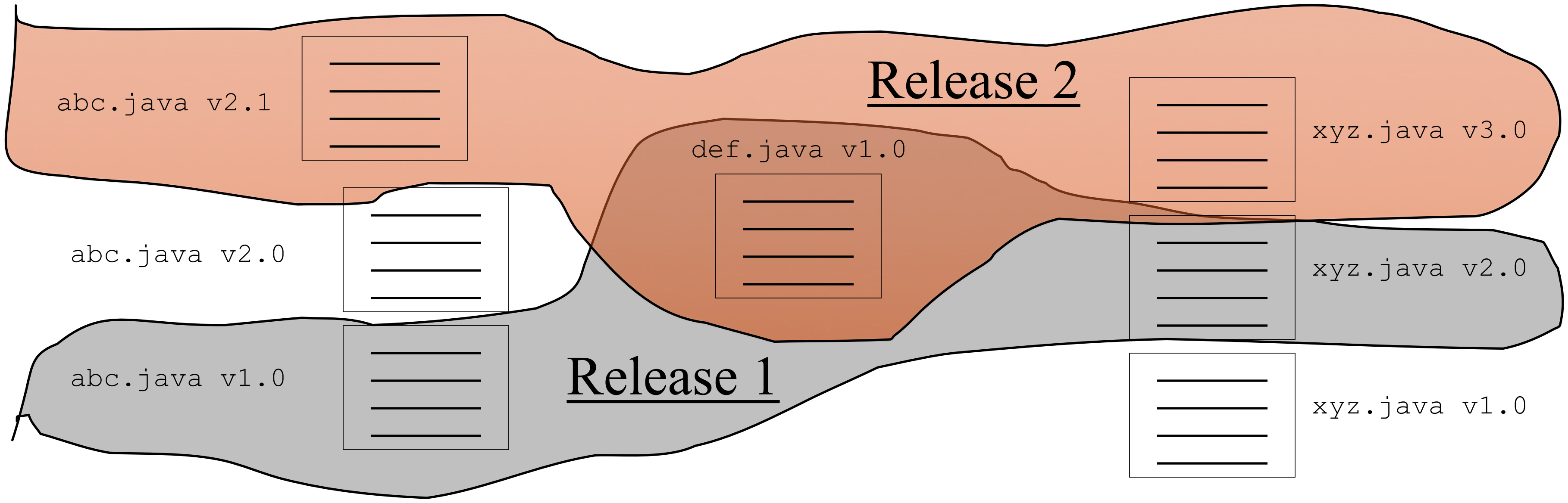
- You have a history
- You have a notion of where it is going
- You have a *set of policies governing participation*



*A **baseline** is a named configuration*

CONFIGURATION MANAGEMENT CONCEPTS

- **Configuration:** An instance of a system composed of specific versions of its artifacts
  - Includes expectations of the target environment(s) & config files!



- **Release:** An instance of a system distributed to users outside of the development team
  - Releases may be targeted for (in)external communities

Version Management

- **Version management (VM)**
  - keeping track of versions of software components or configuration items (CIs) and the systems in which these components are used.
    - involves ensuring that changes made by different developers to these versions do not interfere with each other.
- **VM is what we usually think of as source code control**
- **A source code control (SCC) repository**
  - Often a shared file system of software artifacts
  - Typically supported with client/server tools
  - Often provides some mechanism for assigning jobs to change control on software artifacts
  - Content-Addressable Filesystems

Version Control (SCC)

Version Control

- Used in
  - Software development
  - Offices

- Metadata
  - Timestamp

  - Username

  - Comment

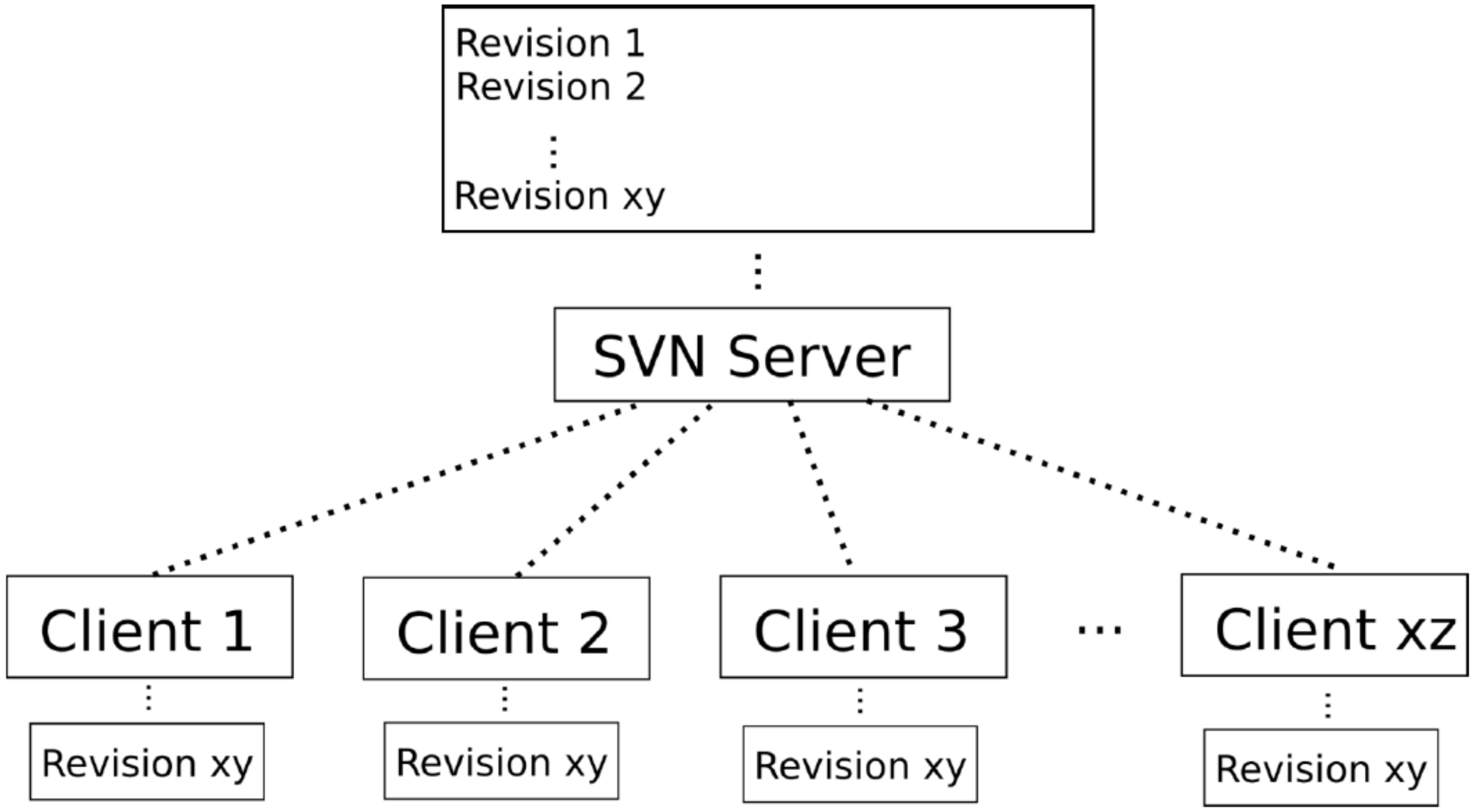
VERSION CONTROL

- Tools
  - SVN, CVS, Git, Darcs….
- Main tasks
  - Tracking of changes
    - Who did what, when, why, how?
  - Backup: Going back to old version
  - History: Archiving and flagging versions
  - Coordination: Many developers
  - Versioning: Distinguishable versions

VERSION CONTROL: ORGANIZATION

- Local (SCCS)
  - Versioning of one file
  - Versioning in file
- Central (CVS, **SVN**)
  - Client/Server System
  - Users have different access
  - Complete Version history on server
- Distributed (Darcs, **Git**)
  - Everybody has local repository
  - Protocol about all changes
  - Merging of different repos is possible

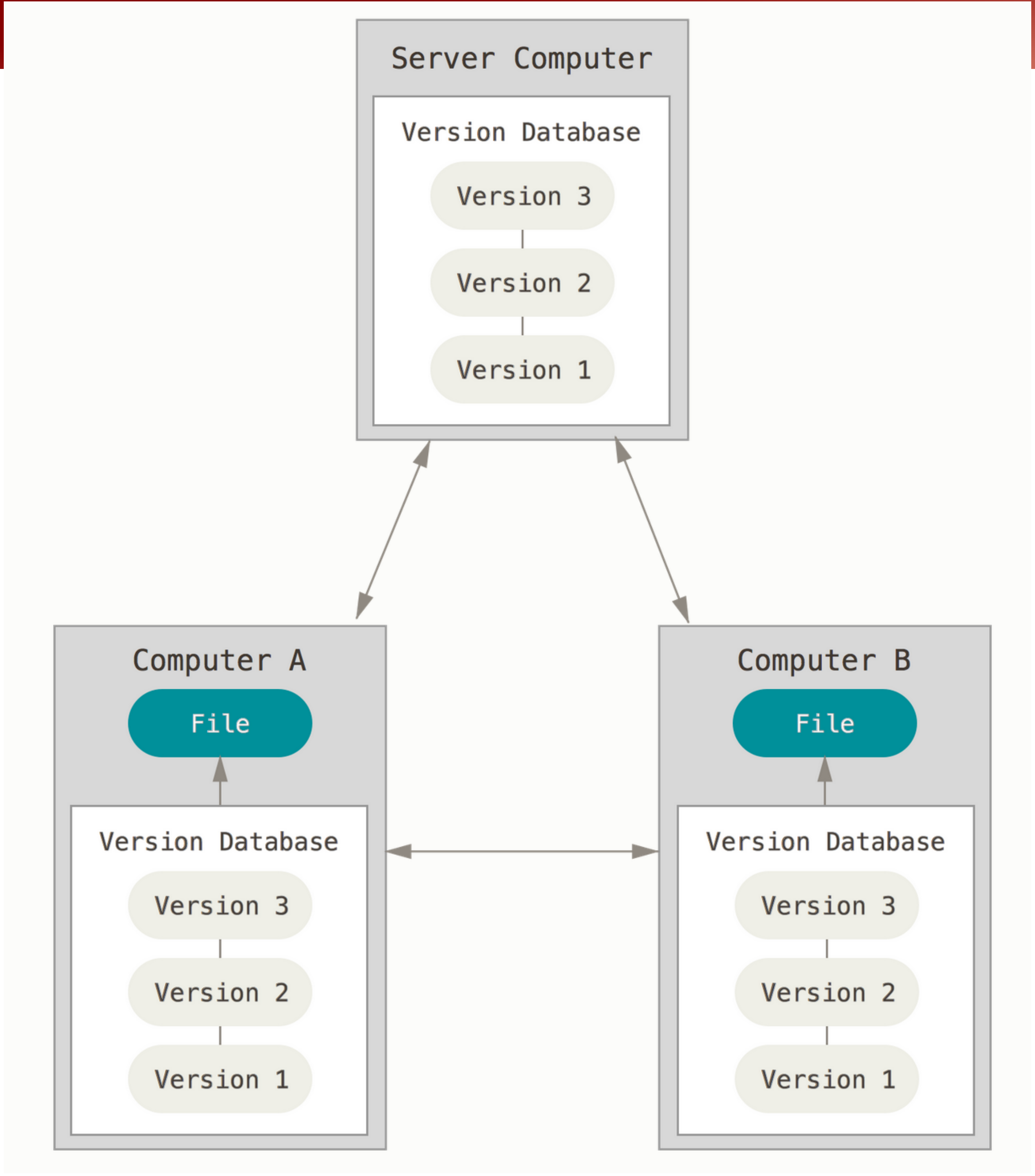
SVN


Revision 1
Revision 2
⋮
Revision xy
SVN Server
Client 1
Client 2
Client 3
...
Client xz
Revision xy
Revision xy
Revision xy
Revision xy

SVN: How does it work?

- Only saves changes to reduce data volume
- Text files: Easy to calculate changes
- Binary files: Often difficult not feasible
    - E.g. images
- Often used for database
- Access through:
    - Client Program
    - PlugIns in Eclipse
    - ….

DISTRIBUTED VERSION CONTROL

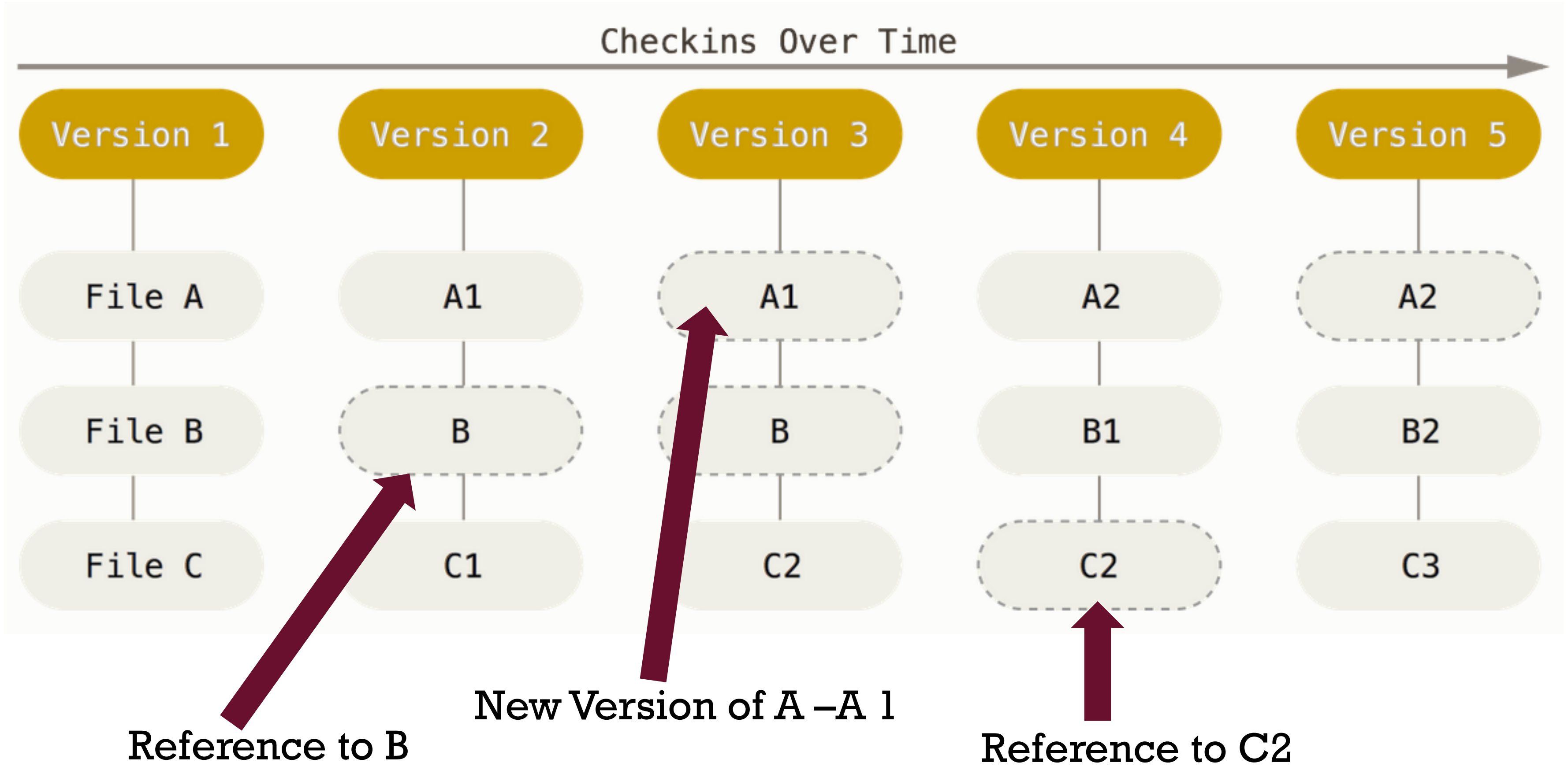


- **Git**
- Mercurial
- Bazaar
- Darcs

GIT

- Can have several local repos
- Collaboration with different groups of people
- Simultaneously work on same project
- Several types of workflows
- Saves snapshots of filesystem
- Every time you commit/push a picture of how your files look is taken
  - A reference to that snapshot is stored
  - If file has not changed it is not stored again (reference will point to already stored file)

GIT CHECKINS OVER TIME



NEARLY EVERYTHING IS LOCAL

- Your entire history is stored locally
- You do not need a network connection to commit a change
- You can browse through your history locally

MORE ABOUT GIT

- Integrity
  - Everything is check-summed
  - You cannot lose any information
- Generally only adds data
  - Can always undo things
  - It does not erase data
- You will never loose your data (unless used wrong)

SUMMARY

- Version Control is important especially in SE projects
- Change always happens during Development
- Traceability helps in development
- Git is one Distributed Version Control system often used in SE