SOURCE CODE MANAGEMENT

- JIDNYASA CORPORATE TRAINING CENTRE
- Prashant Sir (SRE DevOps Engineer)

WHAT IS SOURCE CODE MANAGEMENT?

- Source control refers to tracking and managing changes to code.
- This ensures that developers are always working on the right version of source code.
- Every development team needs a good way to manage changes and version code in their codebases.
- That's why they use source control tools.
- Source control management (SCM) refers to tools that help you keep track of your code with a complete history of changes.

SCM

- Source code management tool (SCM) tracks changes to a source code repository.
- SCM also maintains a history of changes.
- This is used to resolve conflicts when merging updates from multiple developers.

WHY SOURCE CONTROL IS IMPORTANT

- When multiple developers are working within a shared codebase it is a common occurrence to make edits to a shared piece of code.
- Separate developers may be working on a seemingly isolated feature, however this feature may use a shared code module.
- Therefore developer 1 working on Feature 1 could make some edits and find out later that Developer 2 working on Feature 2 has conflicting edits.
- Before the adoption of SCM this was a nightmare scenario.
- Developers would edit text files directly and move them around to remote locations using FTP or other protocols. Developer 1 would make edits and Developer 2 would unknowingly save over Developer 1's work and wipe out the changes.

THE BENEFITS OF SOURCE CODE MANAGEMENT

- Collaborative code development a more user friendly experience.
- Historical record can then be used to ‘undo’ changes to the codebase.
- A clean and maintained SCM history log can be used interchangeably as release notes. This offers insight and transparency into the progress of a project that can be shared with end users or non-development teams.
- SCM will reduce a team’s communication overhead and increase release velocity.
- With SCM developers can work independently on separate branches of feature development, eventually merging them together.
- The most IMP benefit Is Version control.

VERSION CONTROL

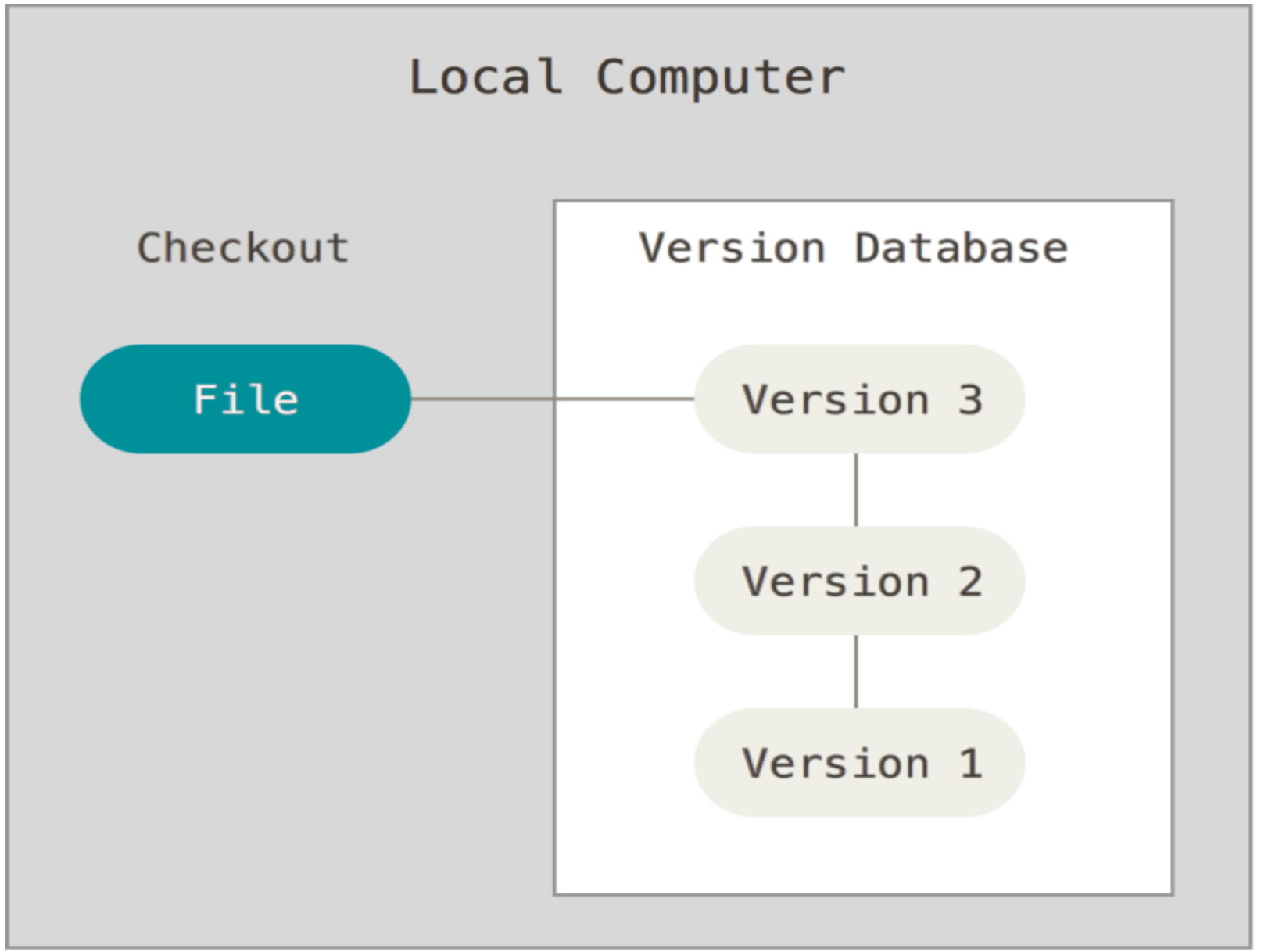
- Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.
- It allows you to revert selected files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more.
- Using a VCS also generally means that if you screw things up or lose files, you can easily recover.
- Here version control is also called as source control its one and the same thing. SCM or VCM is one and the same.

TYPES OF VERSION CONTROL SYSTEM (VCS)

- Local VCS
- Centralized VCS
- Distributed VCS

LOCAL VCS

- Many people's version-control method of choice is to copy files into another directory (perhaps a time-stamped directory, if they're clever).
- This approach is very common because it is so simple, but it is also incredibly error prone. It is easy to forget which directory you're in and accidentally write to the wrong file or copy over files you don't mean to.
- To deal with this issue, programmers long ago developed local VCSs that had a simple database that kept all the changes to files under revision control.
- One of the most popular VCS tools was a system called RCS (Revision control system).(1982)

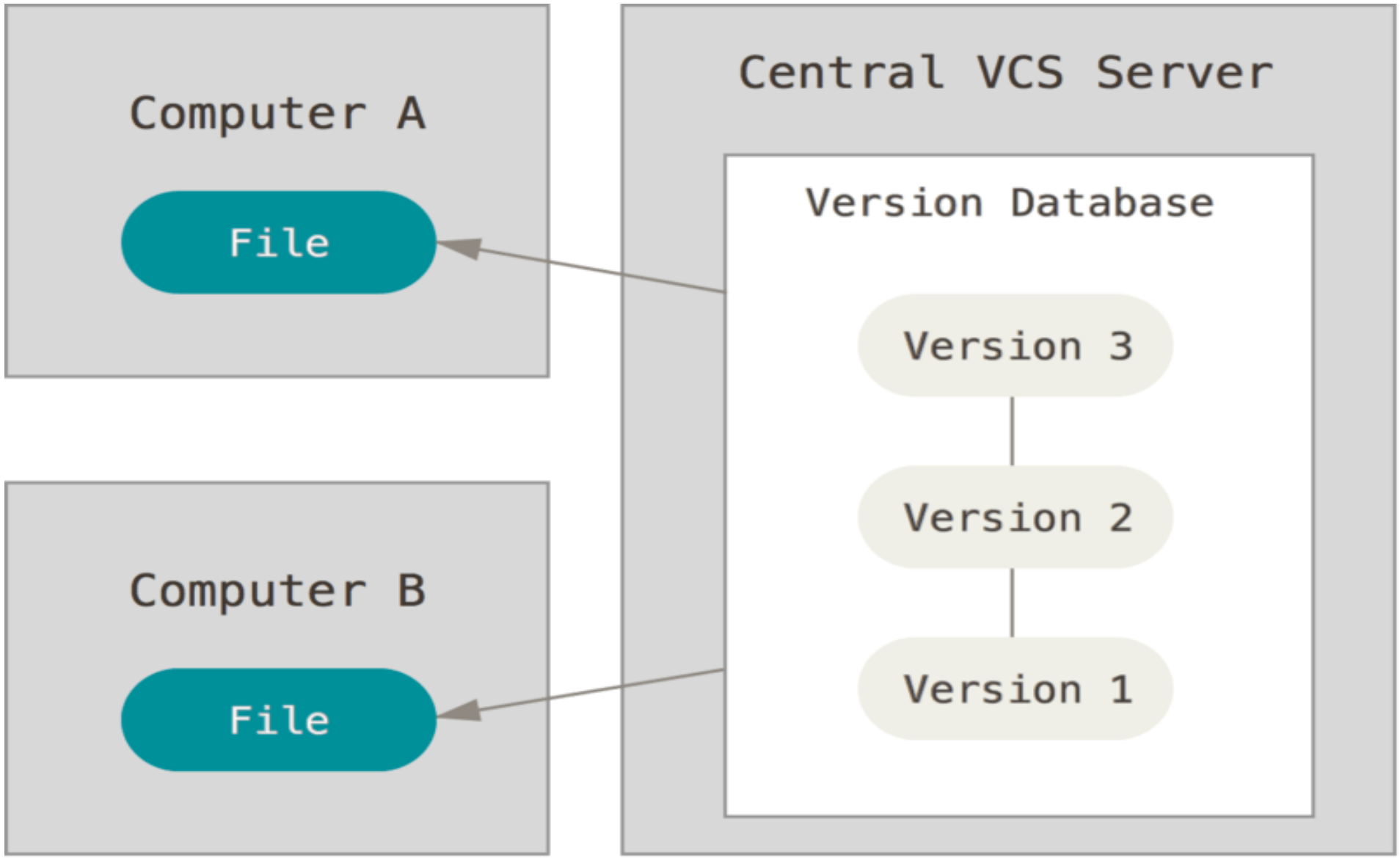

Local Computer
Checkout
File
Version Database
Version 3
Version 2
Version 1

CENTRALIZED VCS

- The next major issue that people encounter is that they need to collaborate with developers on other systems.
- To deal with this problem, Centralized Version Control Systems (CVCSs) were developed.
- These systems (such as CVS, Subversion, and Perforce) have a single server that contains all the versioned files, and a number of clients that check out files from that central place.
- For many years, this has been the standard for version control.

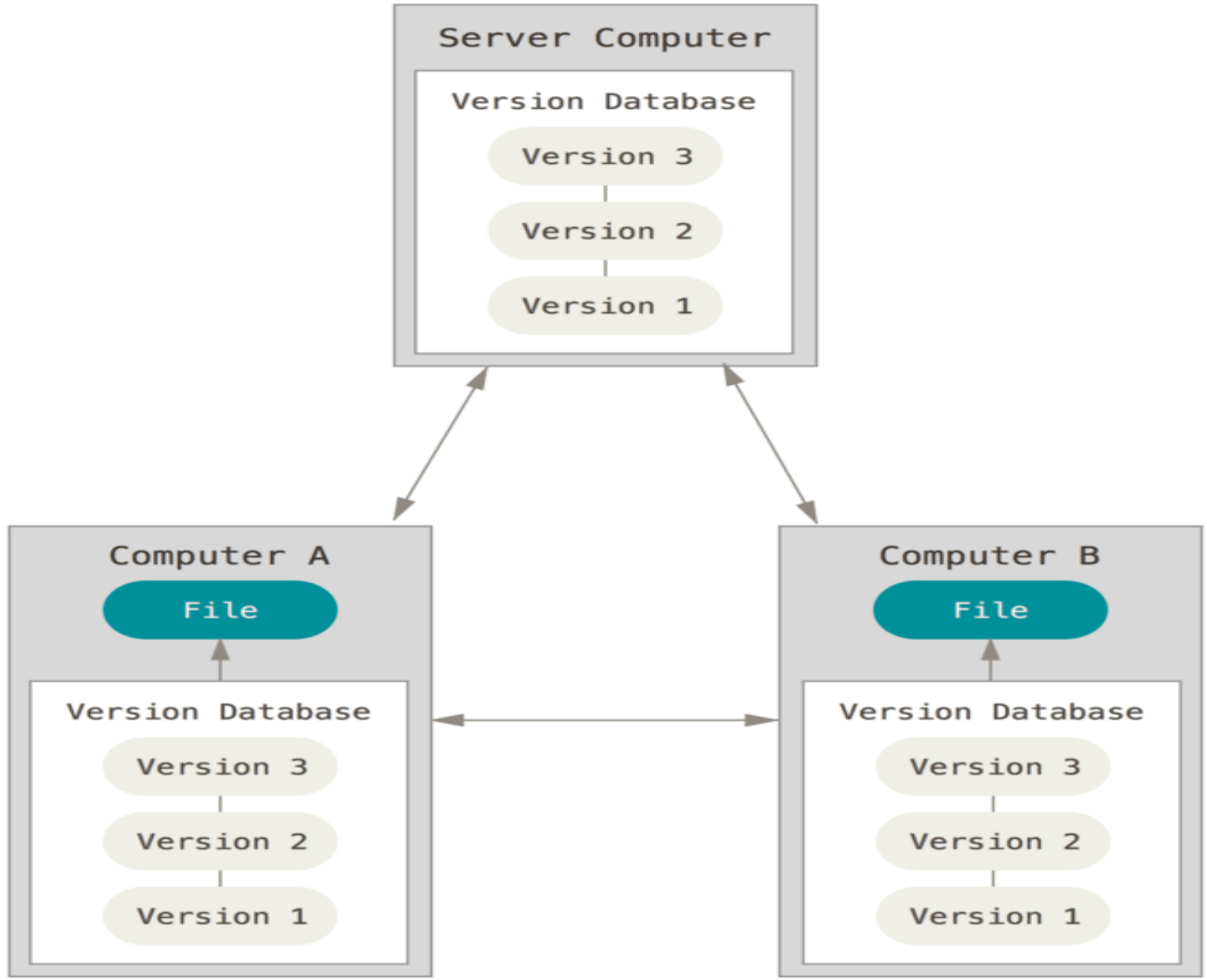
CVCS

- This setup offers many advantages, especially over local VCSs. For example, everyone knows to a certain degree what everyone else on the project is doing.
- However, this setup also has some serious downsides. The most obvious is the single point of failure that the centralized server represents.
- If that server goes down for an hour, then during that hour nobody can collaborate at all or save versioned changes to anything they're working on.
- If the hard disk the central database is on becomes corrupted, and proper backups haven't been kept, you lose absolutely everything — the entire history of the project except whatever single snapshots people happen to have on their local machines.


Computer A
File
Computer B
File
Central VCS Server
Version Database
Version 3
Version 2
Version 1

DISTRIBUTED VERSION CONTROL SYSTEM

- This is where Distributed Version Control Systems (DVCSs) step in. In a DVCS clients don't just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history.
- Thus, if any server dies, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to restore it.
- Every clone is really a full backup of all the data.
- Furthermore, many of these systems deal pretty well with having several remote repositories they can work with, so you can collaborate with different groups of people in different ways simultaneously within the same project.


Server Computer
Version Database
Version 3
Version 2
Version 1
Computer A
File
Version Database
Version 3
Version 2
Version 1
Computer B
File
Version Database
Version 3
Version 2
Version 1

ONE OF THE MOST POPULAR DVCS IS

GIT