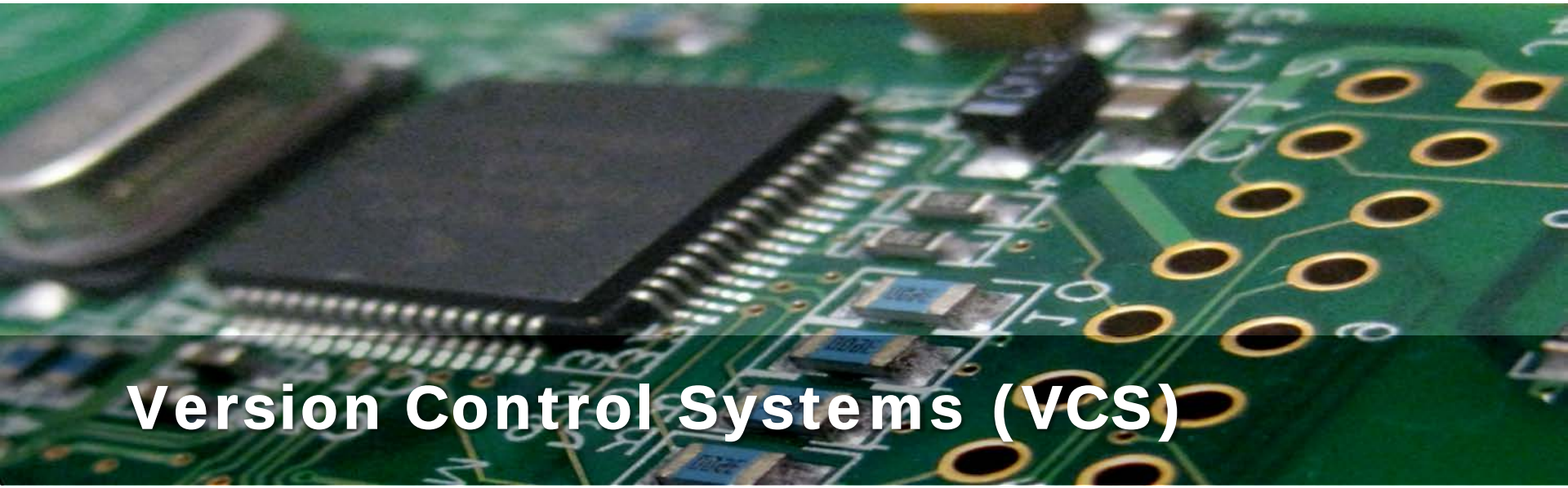
Lucerne University of
Applied Sciences and Arts
HOCHSCHULE
LUZERN
Technik & Architektur



Version Control Systems (VCS)

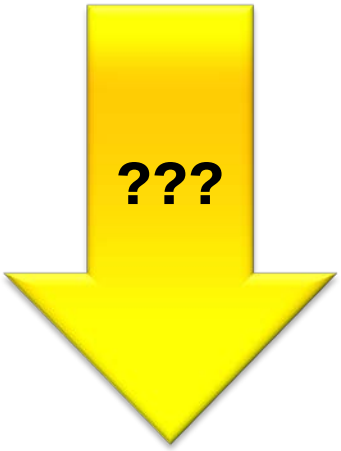
"Good Reading on this topic: http://betterexplained.com/articles/a-visual-guide-to-version-control/" (Credits to this site!)

Prof. Erich Styger
erich.styger@hslu.ch
+41 41 349 33 01

Learning Goals

- Problem: Collaborate, Track and Backup

- Goal
  - Concepts of Version Control Systems
  - Using a version control system
  - Using web interface
  - Using a client
  - Using in Eclipse



- Note:
  - It is about the principles, not the tool
  - You shall be able to apply the principles with VCS

Use Cases?

- Main.c, backup of main.c, main2.c, …
- MyReport1.doc, myReport2.doc, …
- Project.old, project.old2, …
- ➔ ‘Save as’ Method
  - New file without impacting existing one
  - Backup
  - Go back if things break
  - Using ‘version numbers’ or ‘date’ information
- ➔Sharing files
  - Shared folder (everyone saves under a new name), merge
  - Send out per email (have a local copy)

- OK: one-time paper, one person project, …

Do we need something different?

- Things get out of control fast
- Source project with hundreds of files?
- 2, 4, 16, 30 developers?
- Things changing fast, in ‘parallel’ fashion?

- Need for a Version Control System
  - Kind of file data base
  - System cares about versioning
  - System imposes rules

- And: you need to invest some time to learn it

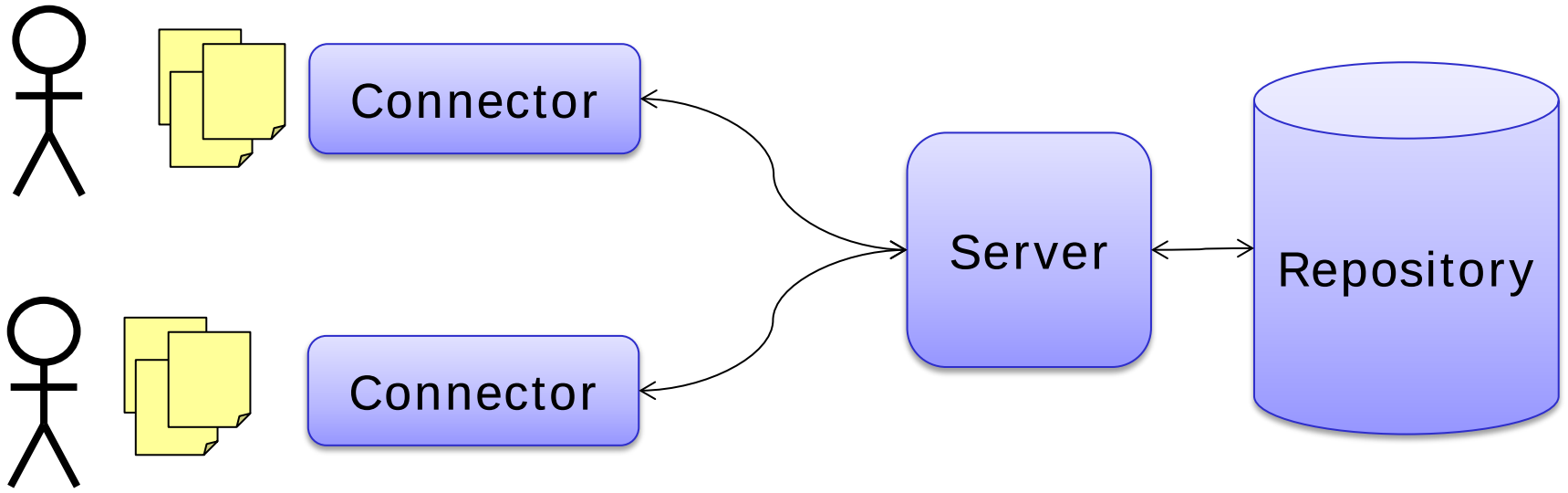
A Good VCS

- Backup and Restore
- Synchronization
- Short-term undo
- Long-term undo
- Track Changes
- Track Ownership
- Sandbox
- Branching
- Merging

➔ Shared folders are quick and simple, but hardly fulfill above!

Typical Version Control System

- Server(s) with data base(s)
- Clients connect to server
  - Locally or remote
  - Single Server or distributed
- ‘Optimistic’ or ‘Pessimistic’ approach



Centralized vs. Distributed

- Centralized
  - Repository server (typically single server)
  - Data is on server
  - Local copy of current snapshot on client
  - Commit/compare when connected
  - Example: CVS (Concurrent Versions System), **SVN** (SubVersion)
- Distributed
  - Repository server (can be multiple)
  - Data is on server and on client
  - Local copy contains full repository history
  - Commit/compare/etc even when not connected
  - Commit(s) to local repo, then sync with server
  - Example: **Git**

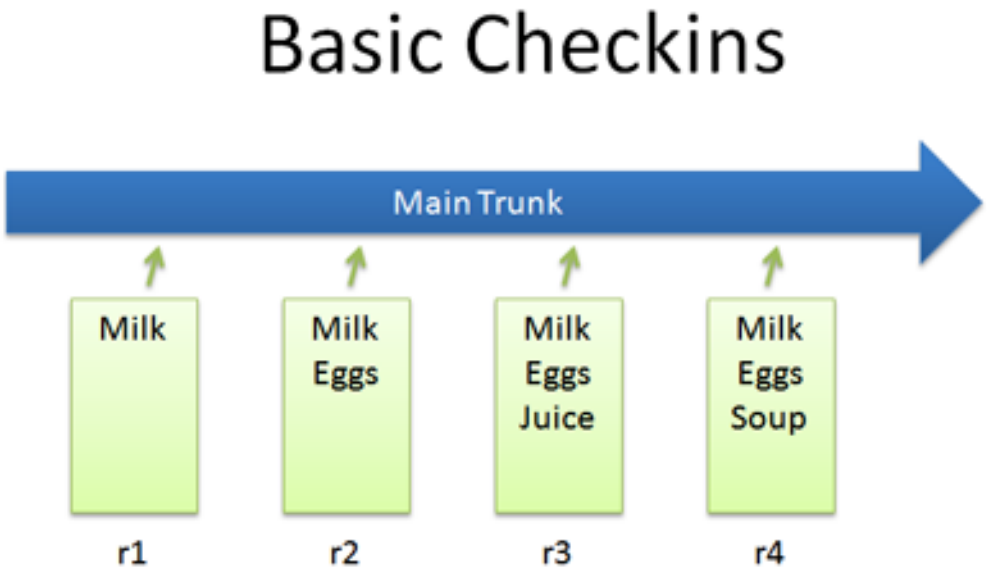
Language

- **Setup**
  - Repository
  - Server
  - Working Set/Copy
  - Trunk/Main(line)/Master/Head
- **Basic Actions**
  - Add
  - Revision
  - Head
  - Check-out
  - Update, Sync, Pull
  - Check-in / Commit / Push
  - Check-in message
  - Revert
- **Advanced Actions**
  - Branch
  - Diff/Delta
  - Merge
  - Conflict
  - Resolve

A Day in Joe's Live

1. Joe **adds** main.c to the **repository**.
2. Bill **checks** it **out**, makes a change and **commits** it with a **commit message**.
3. Two hours later, Joe **updates** his local working set and sees the latest **revision** of main.c which contains the change.
4. Joe can browse the **change log** or **diff** to see what Bill had changed the day before.

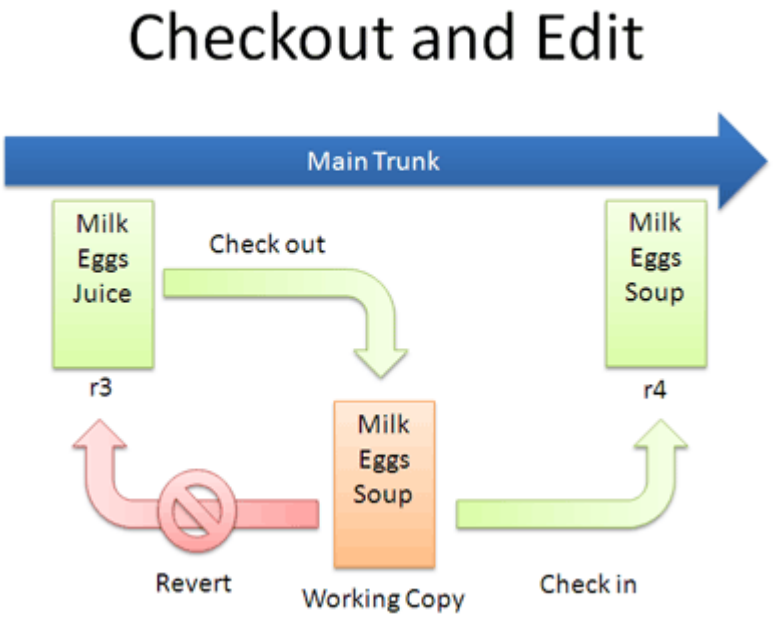
Check-In/Commit



Source: betterexplained.com

- Each commit creates a revision
- Modification over time

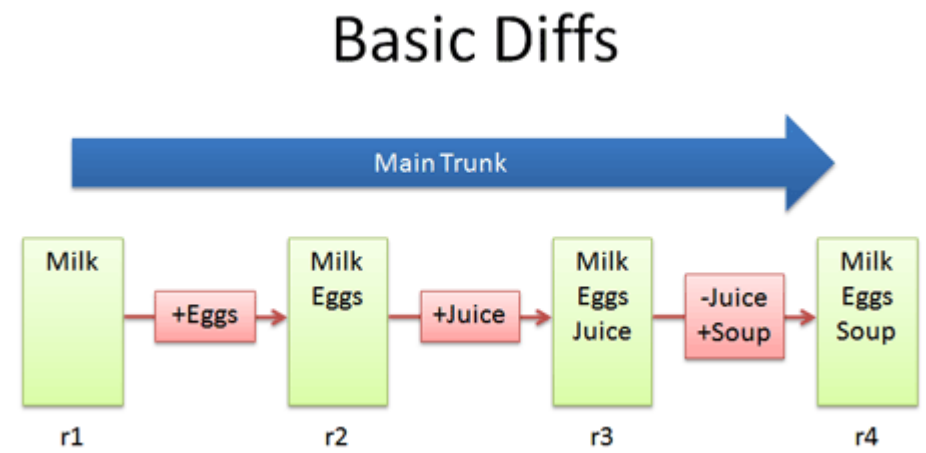
Check-Out and Check-In



Source: betterexplained.com

- Check-Out ➔ Edit working Copy ➔ Check-In
- Revert to throw away local changes ➔ get back version from server

Diffs / Delta



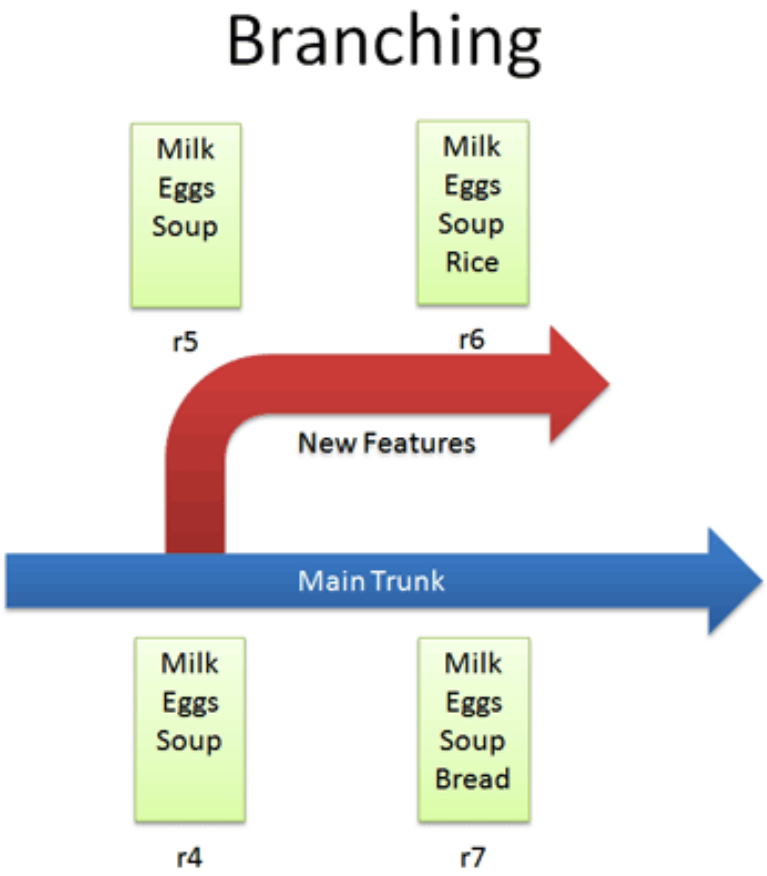
Source: betterexplained.com

- Making diff between (arbitrary) versions
- Most VCS just store the delta (save space)

Branching

- Sandboxing
- Adding large features
- Development over a longer time
- Isolating from main trunk development

- Usually: to be **merged** back to main trunk/line

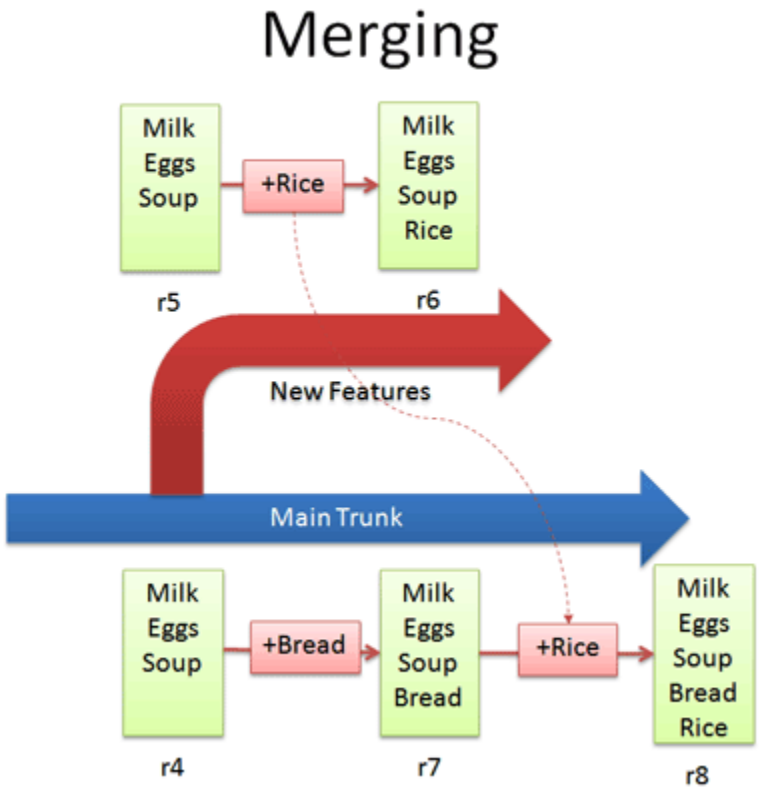


Source: betterexplained.com

Merging

- Developing on a branch needs discipline
- Merging needs to be carefully planned

- ➔ write/keep change/commit logs to assist you in merging!



Source: betterexplained.com

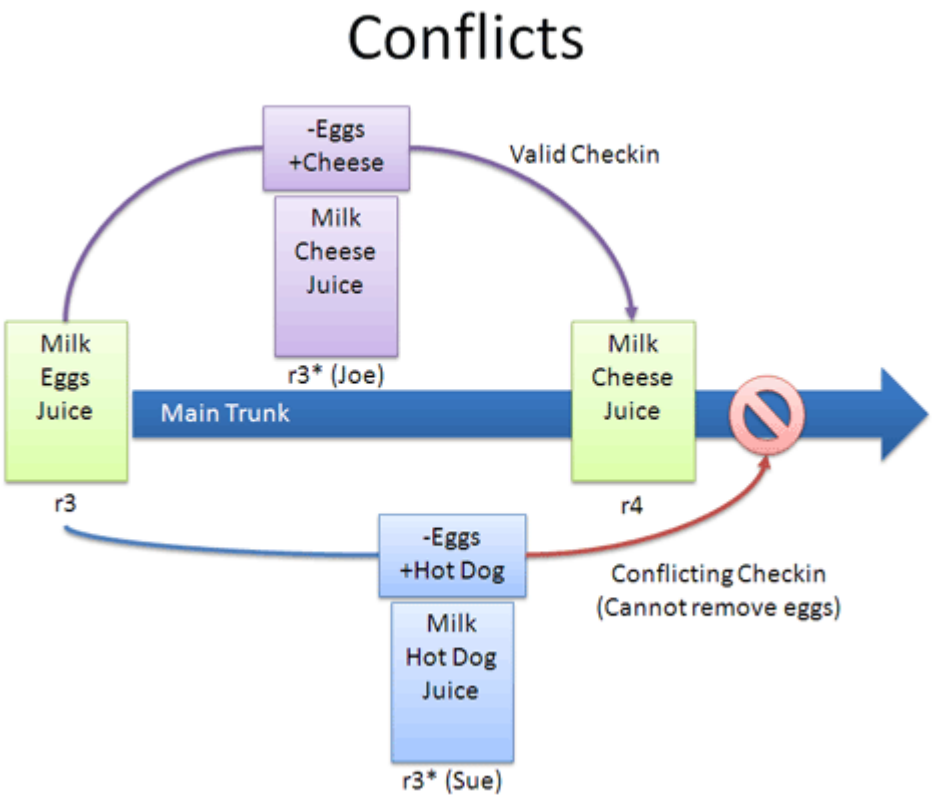
Conflicts

- VCS's can automatically merge non-conflicting changes

- 'same line' changes
  - Race change

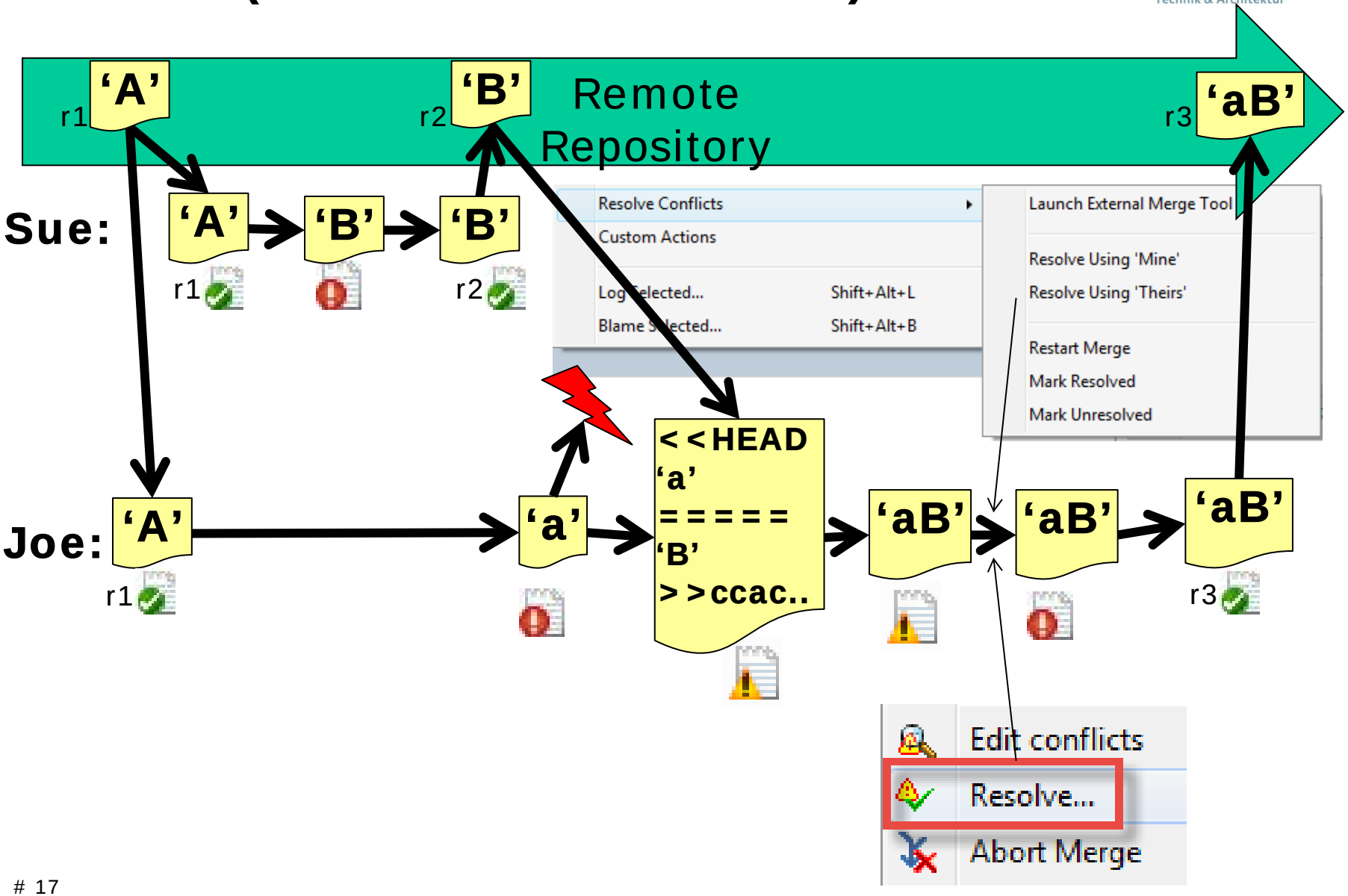
- VCS reports **conflict**

- Need to **resolve**

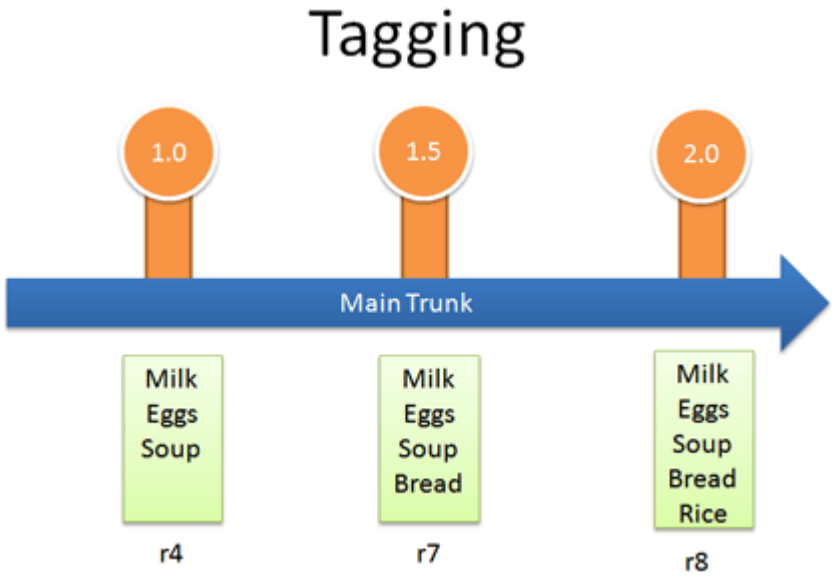


Source: betterexplained.com

Conflicts (TortoiseGit / SourceTree)



Tagging



Source: betterexplained.com

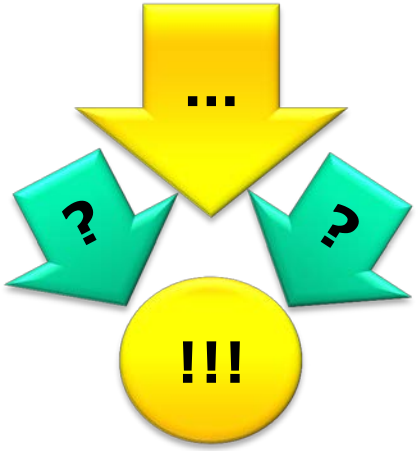
- Apply a label
- ‘V1.0’, ‘Release’, ‘beta’, …
- Check-out all files with a given label

Summary

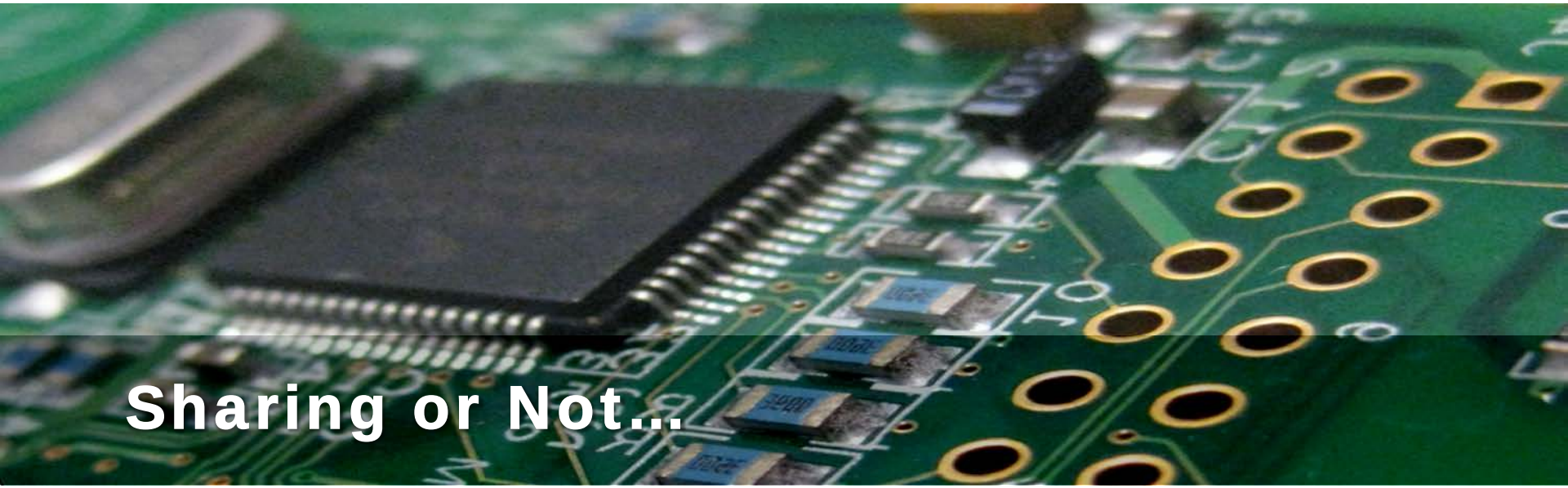
- **Use Version Control**
  - Pretty much for everything you want to maintain and you are changing

- **Take it slow and easy**
  - Advanced features are not needed in the first place (branching, merging, ...)
  - Check-out, check-in, update, merge is critical

- **Keep Learning**
  - Every tool is a little bit different
  - Know the concepts!
  - VCS does **not** replace communication!



Lucerne University of
Applied Sciences and Arts
HOCHSCHULE
LUZERN
Technik & Architektur



Sharing or Not...

"... that's the question?"

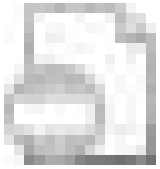
Prof. Erich Styger
erich.styger@hslu.ch
+41 41 349 33 01

What to share with a VCS

Lucerne University of Applied Sciences and Arts
HOCHSCHULE LUZERN
Technik & Architektur

- **Share:** everything needed to build the project
  - **.project** and **.cproject** (project files and build options)
  - sources files **(*.c, *.h, etc**) and folders
  - **Project_Settings** folder and files (linker files, startup files)
  - ***.launch** (contains launch/debugger settings)
  - Processor Expert specific
    - **ProcessorExpert.pe** (contains component settings)
      - ➔ Special rules/consideration needed!!!
    - **Static_Code**
    - **main.c**, **Events.c** and **Events.h**
      - ➔ **Processor Experts adds content!**
      - ➔ **Shared between user and PE!**

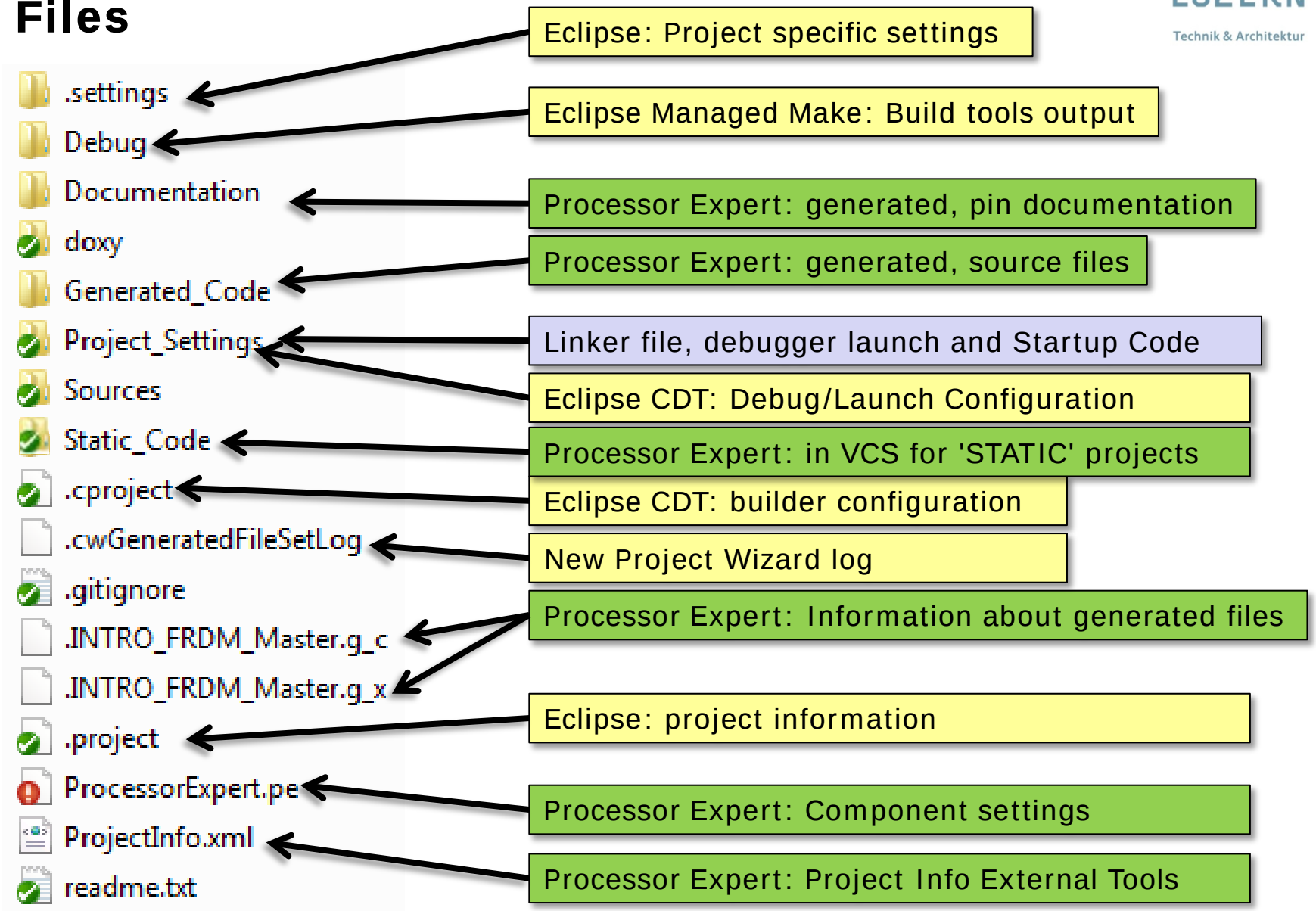
What NOT to share with a VCS



Lucerne University of Applied Sciences and Arts
HOCHSCHULE LUZERN
Technik & Architektur

- **Do not share: generated or derived resources**
  - Derived Folders/Files (resource attribute!)
  - generated documentation/log files
  - Build output files (named as the build target, e.g. '**Debug**')
  - Processor Expert (as generated)
    - **Documentation**
    - **Generated_Code**
  - .ProcessorExpert.g_c and .ProcessorExpert.g_x contain information about the generated files. As we do not share the generated files, they are not shared
  - *Discussion point: sharing the generated Processor Expert Sources?????*

Files


.settings
Debug
Documentation
doxy
Generated_Code
Project_Settings
Sources
Static_Code
.cproject
.cwGeneratedFileSetLog
.gitignore
.INTRO_FRDM_Master.g_c
.INTRO_FRDM_Master.g_x
.project
ProcessorExpert.pe
ProjectInfo.xml
readme.txt
Eclipse: Project specific settings
Eclipse Managed Make: Build tools output
Processor Expert: generated, pin documentation
Processor Expert: generated, source files
Linker file, debugger launch and Startup Code
Eclipse CDT: Debug/Launch Configuration
Processor Expert: in VCS for 'STATIC' projects
Eclipse CDT: builder configuration
New Project Wizard log
Processor Expert: Information about generated files
Eclipse: project information
Processor Expert: Component settings
Processor Expert: Project Info External Tools

Summary

- Only share what is needed to build the project

- Do **NOT** share derived files
  - Object files
  - Generated files

- In case of tool/XML/binary files
  - Special consideration needed!

- ALWAYS check what you commit!