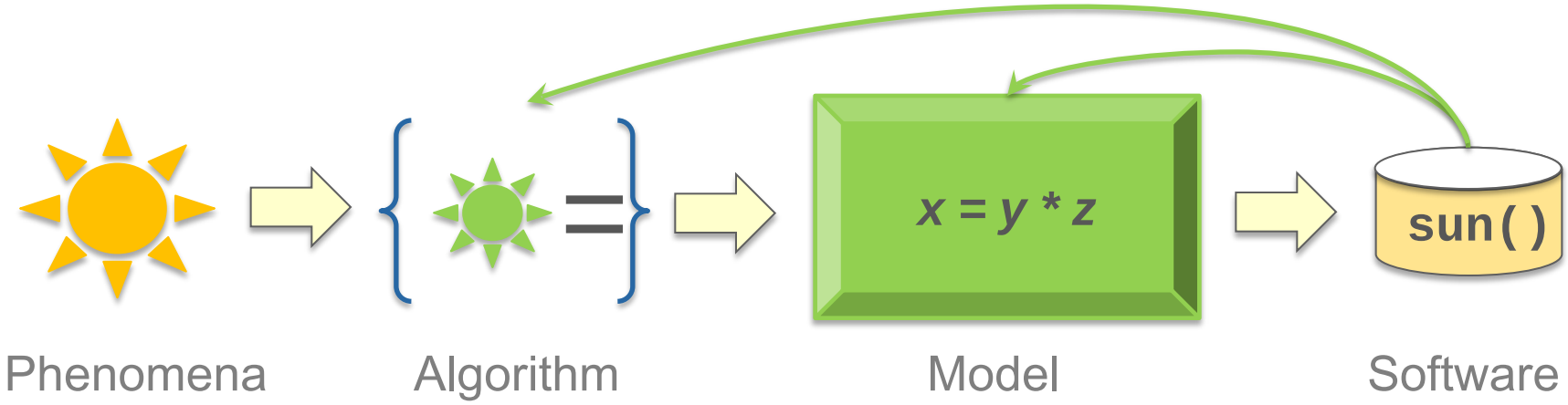
Software Engineering and Scientific Computing

Barbara Paech, Hanna Remmel
Institute of Computer Science
Im Neuenheimer Feld 326
69120 Heidelberg, Germany
http://se.ifi.uni-heidelberg.de
paech@informatik.uni-heidelberg.de

[Dubois 2005]

Scientific Software Engineering (1)

What does it have to do with me and my work?

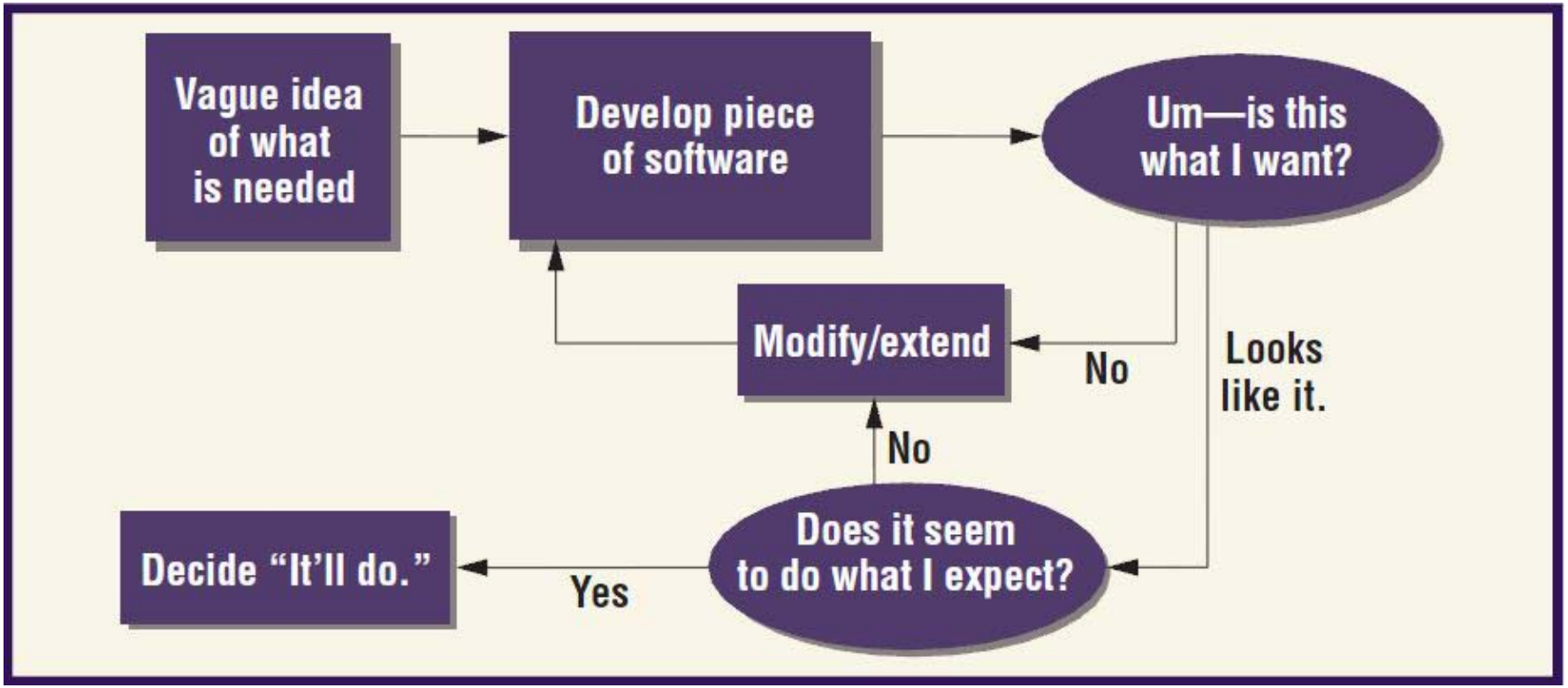


Scientific Software Engineering (2)

- Different from traditional Software Engineering
  - Developed by scientists
    - Alone or in a team, often distributed
    - Mostly the developers are also users
    - Professionals in the application field, not computer science
    - Use software for research: interested in results, not the software development process
  - Often specific hardware needed: High Performance Computing
  - Output not known in advance (missing test oracle)
  - Requirement often non-functional: correctness, performance, portability,…
    - Things like UI not so important

[Segal 2008]

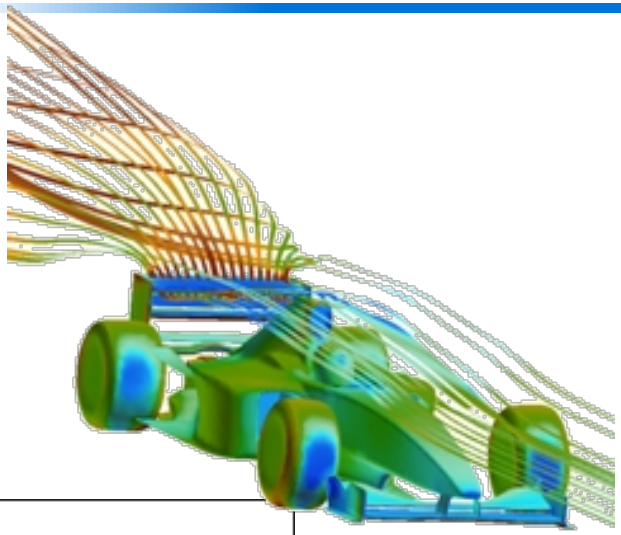
How do scientists develop software?

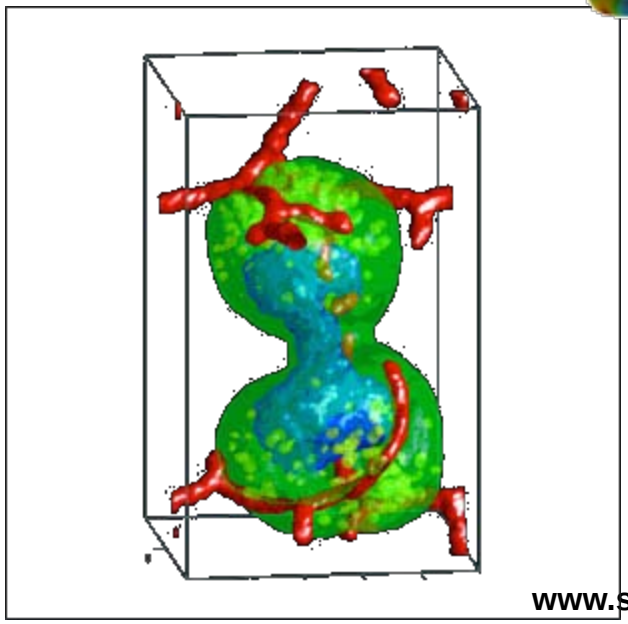


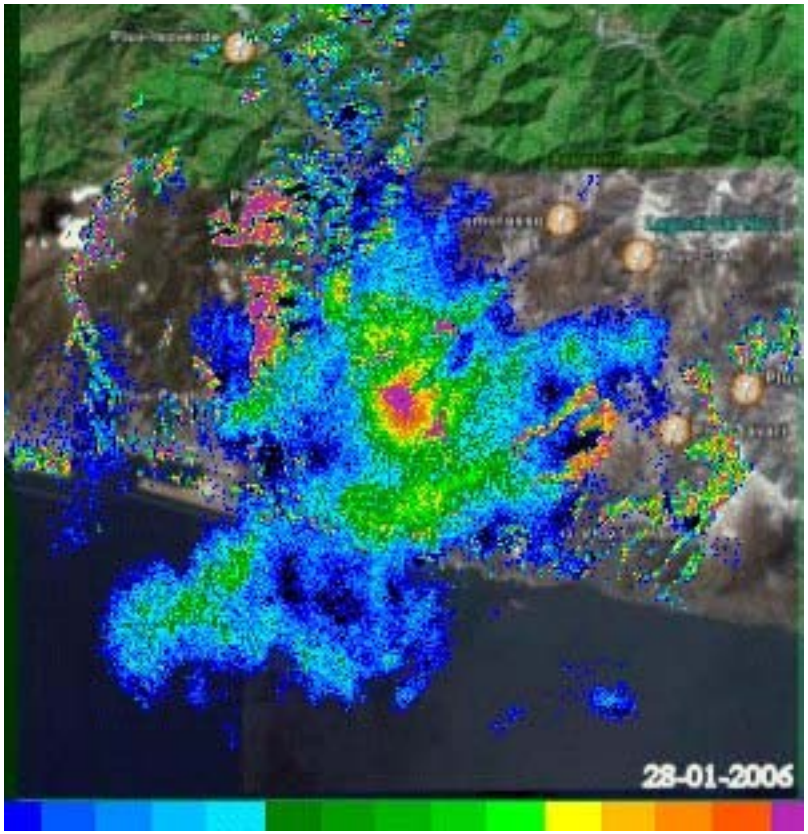
Kinds of scientific software

- High performance computing
  - Complex simulation on parallel computers
- Framework, library
  - Code from which algorithms for a specific problem can be created/adapted
- Scientific workflow
  - Software to automate a process of performing a big experiment or data analysis
  - Describe the structure of the process (workflow)
  - Support the semi-automatic execution (workflow management)
- Small scripts, Data Mining

Simulations using HPC



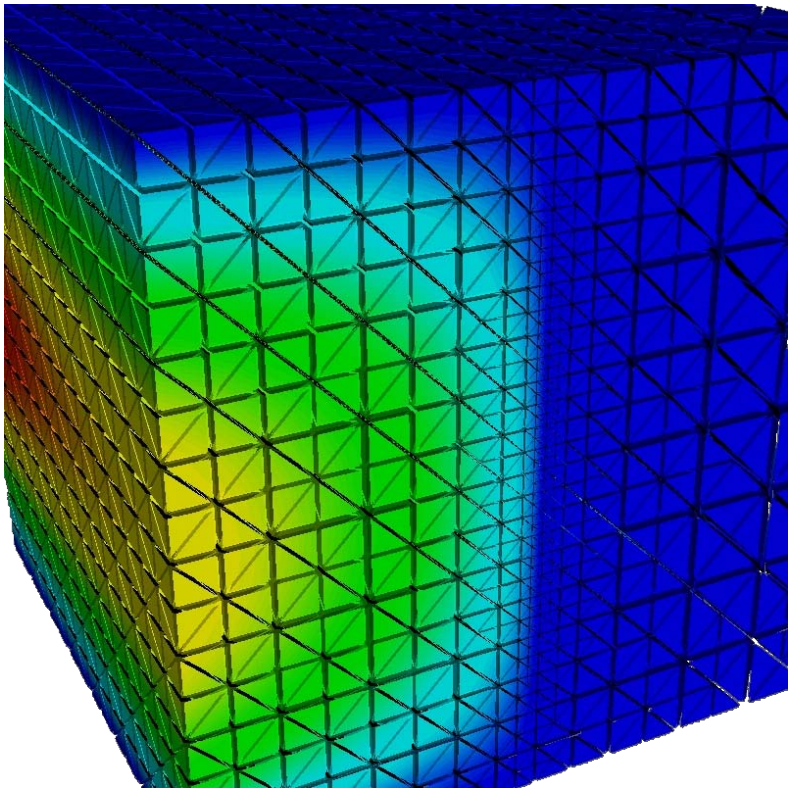




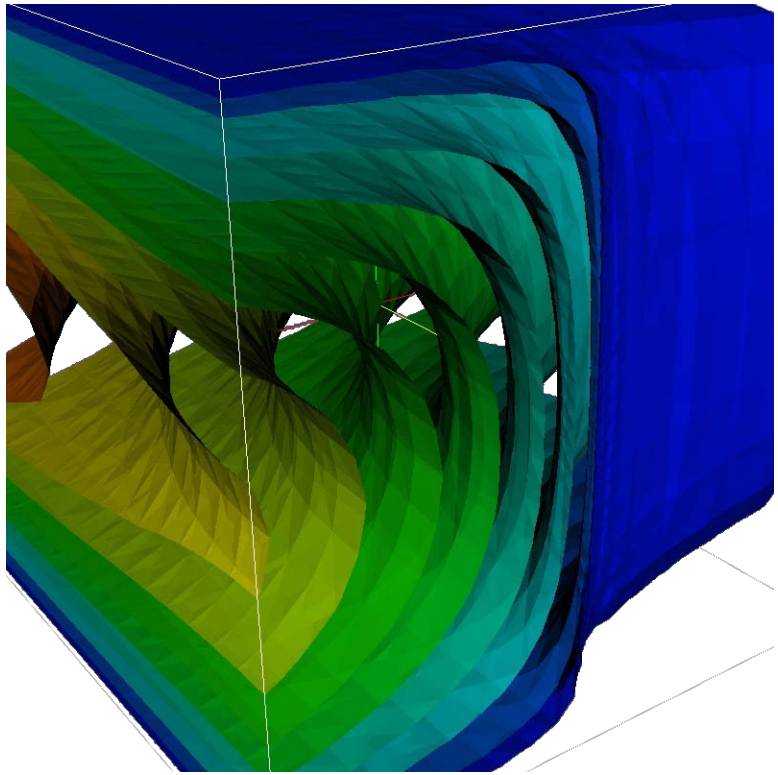
www.scientific-computing.com; http://newsinfo.iu.edu; http://www.ibrsistemi.com

[www.dune-project.org]

Frameworks

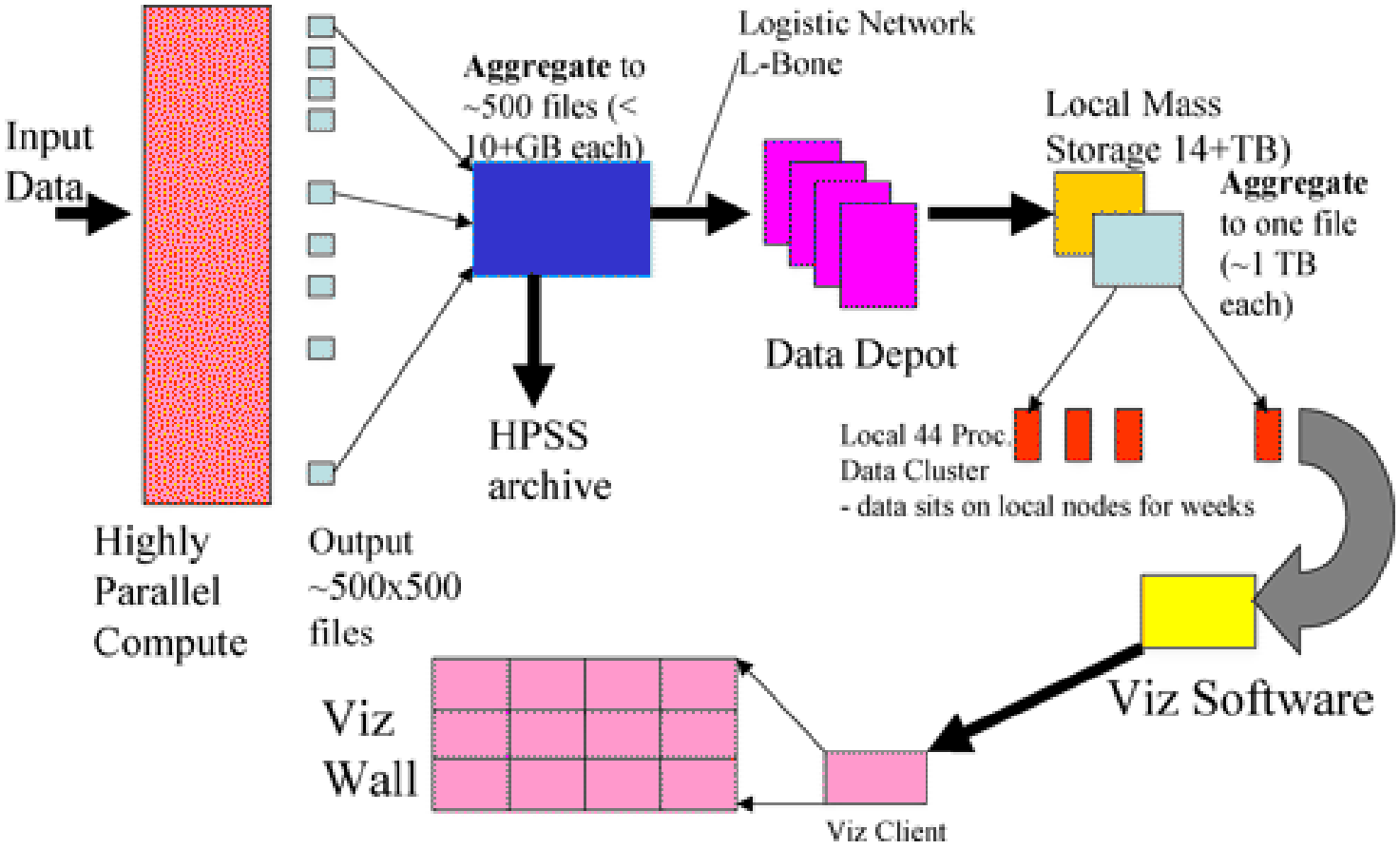


Distributed and Unified Numerics Environment



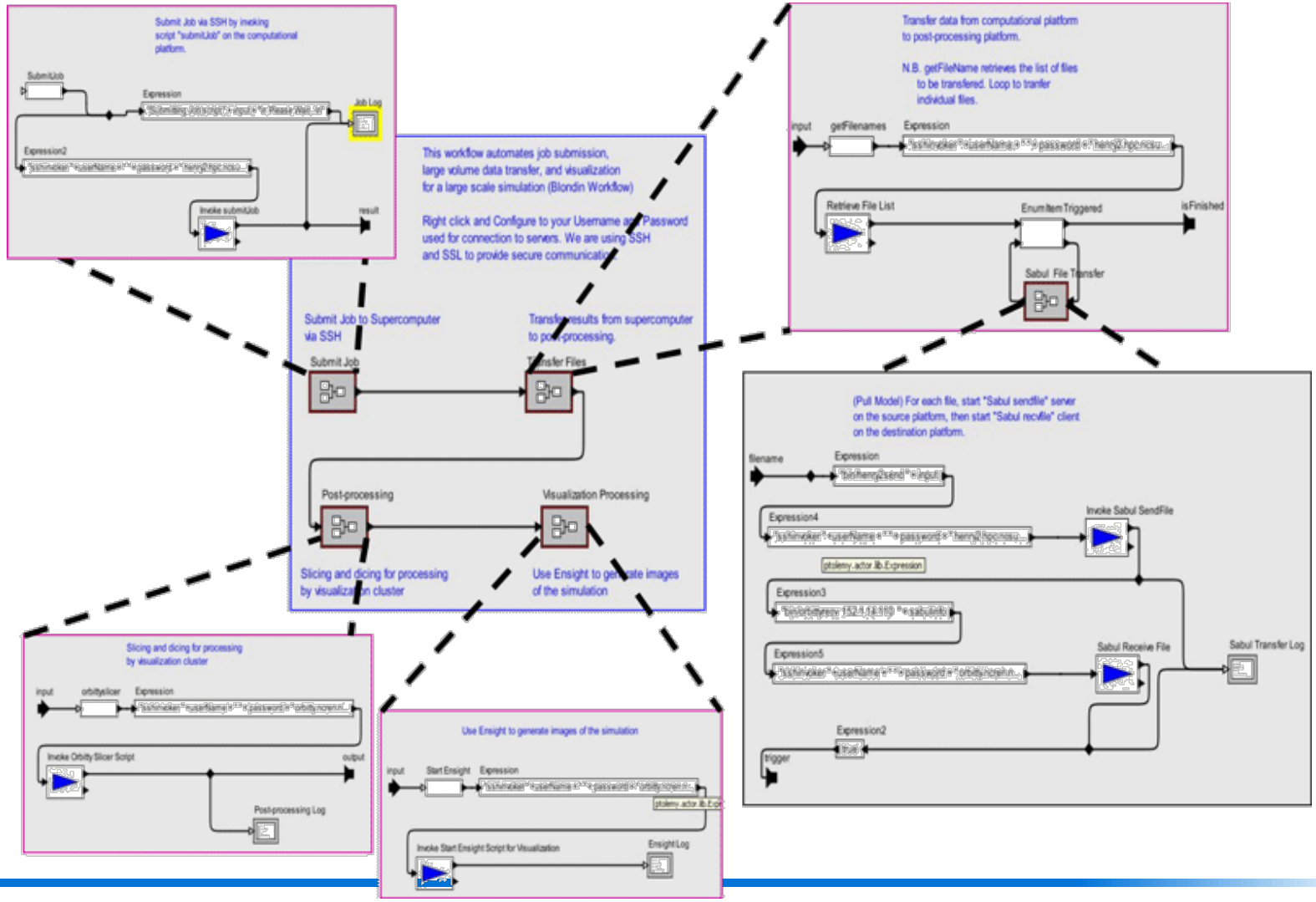
[Shoshani et al. 2007]

Scientific workflow (1)



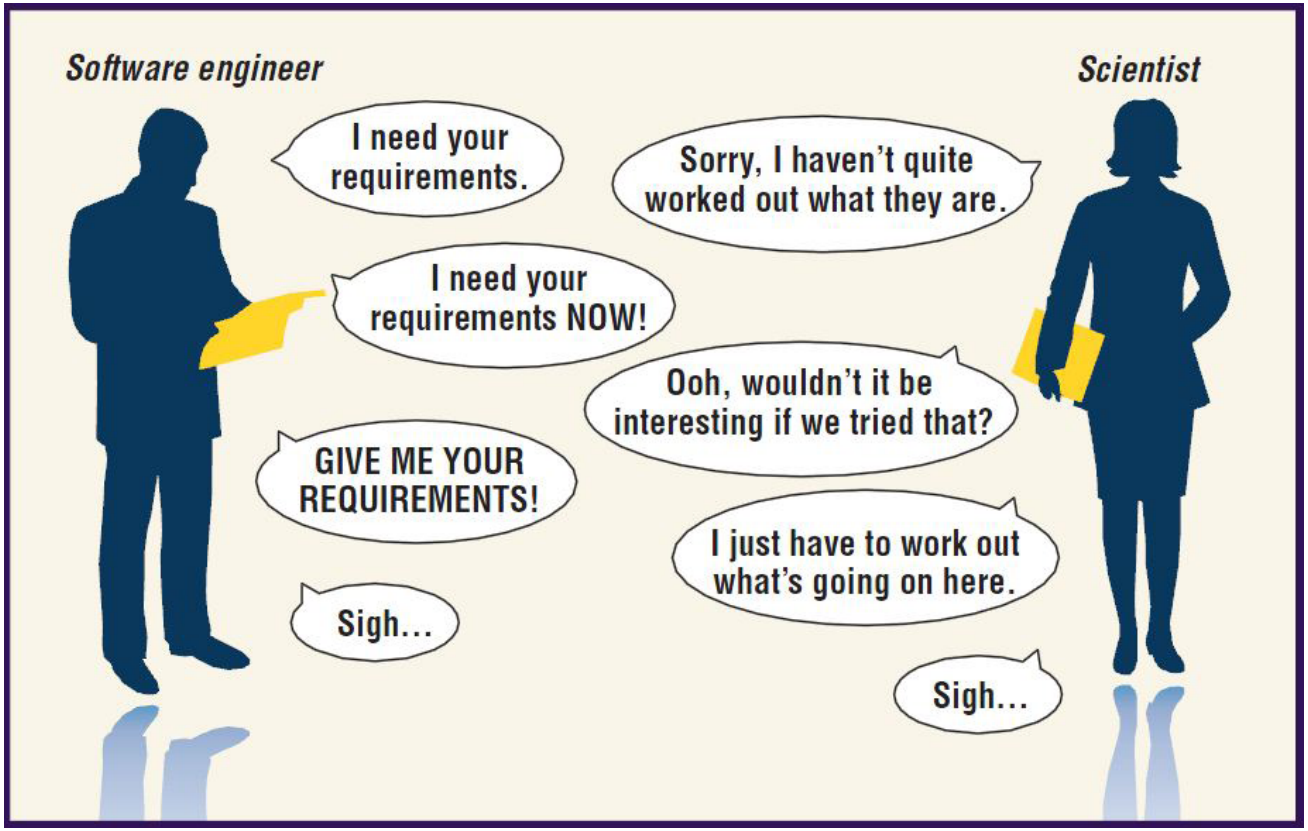
[Shoshani et al. 2007]

Scientific workflow (2)



[Segal, Morris 2008]

Scientific Software Engineering can help you to develop better scientific software!



Programming in a small team

What is Ron doing?

Project management
Issue Tracking

I want to explain my ideas to Hermione

Modeling
Knowledge Management

I want to change Ginnys code

Version management,
Build management

I want to check Harrys changes

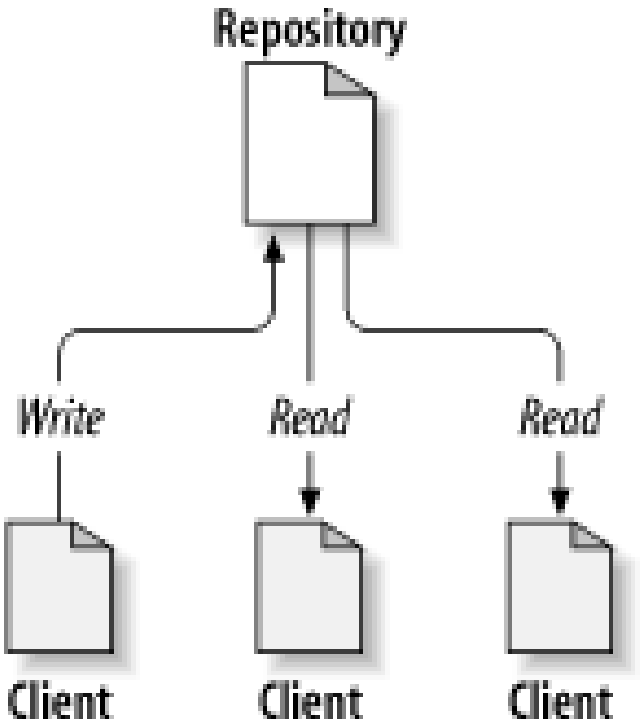
Quality assurance
Testing

Version Management

- What do i need a Version Management for?
  - Keeping track of different versions of the software
  - Collaboration with other developers
  - A safe copy of the software
  - Possibilty to revert changes in many files at the same time
  - …

[http://svnbook.red-bean.com/en/1.5/svn-book.html]

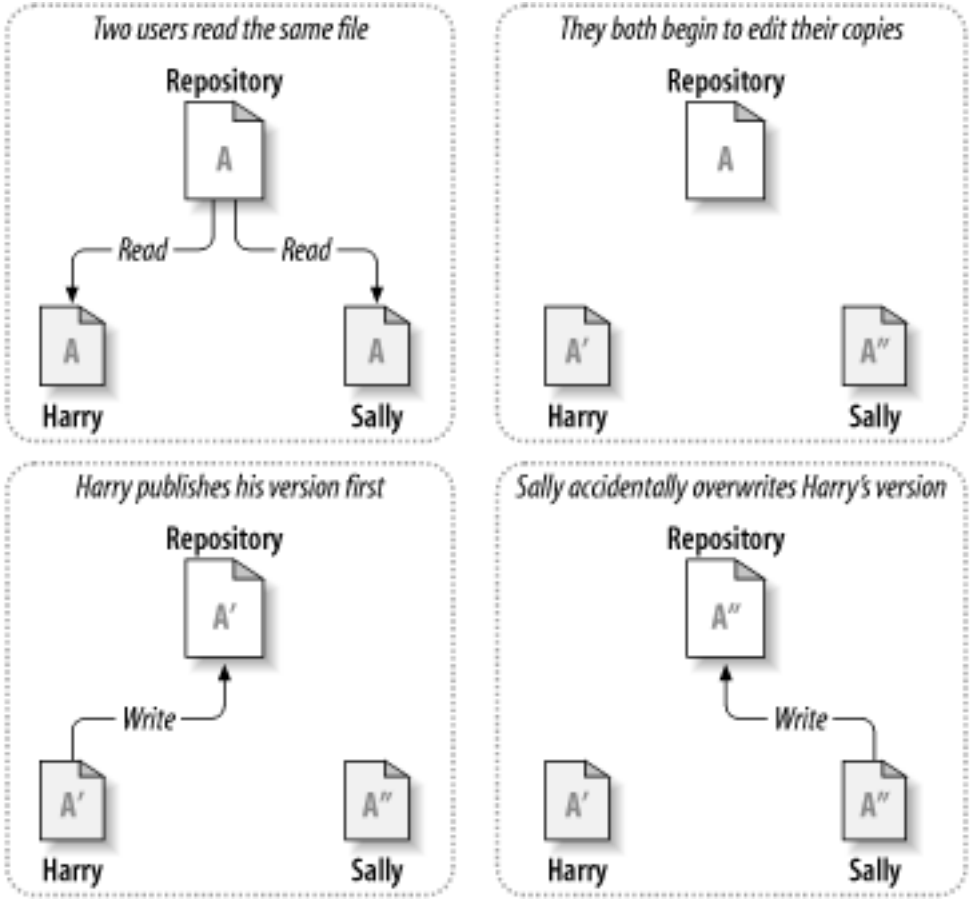
Version Management



[http://svnbook.red-bean.com/en/1.5/svn-book.html]

Problem #1: Collaboration

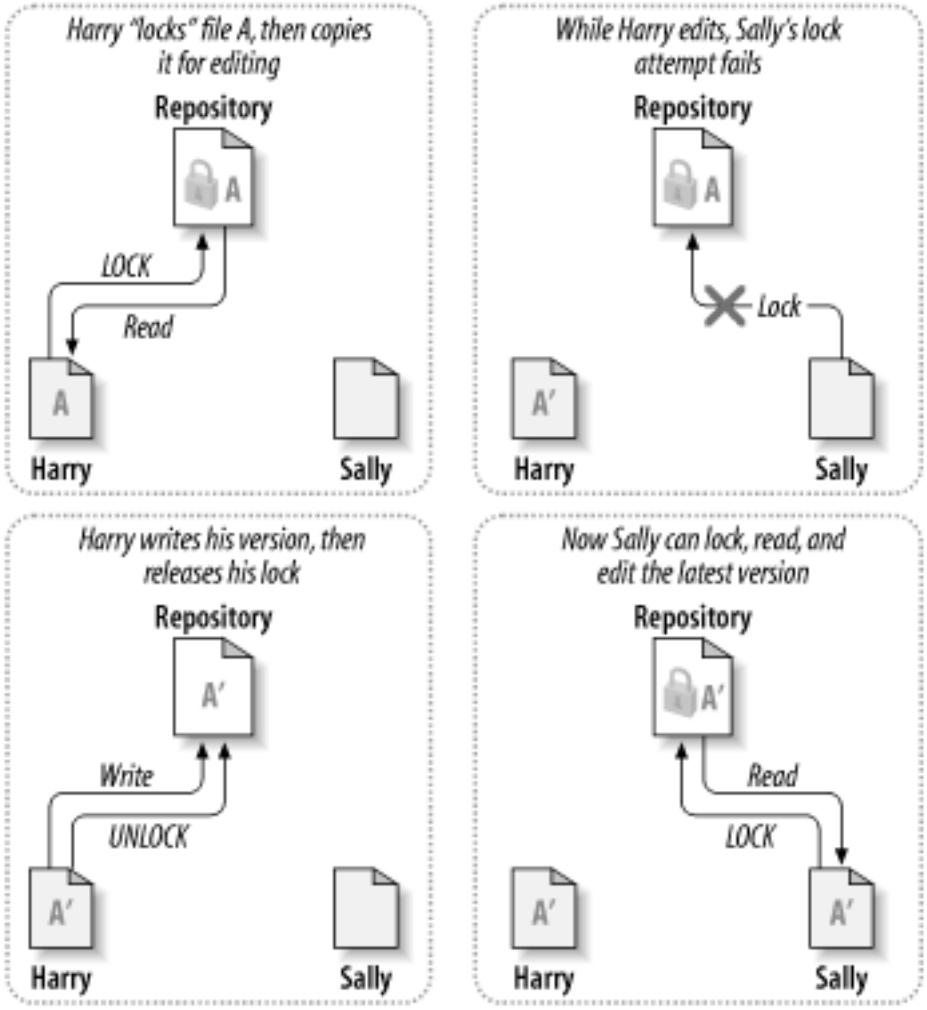
- What if two or more people want to edit the same file at the same time?



[http://svnbook.red-bean.com/en/1.5/svn-book.html]

Problem #1: Collaboration

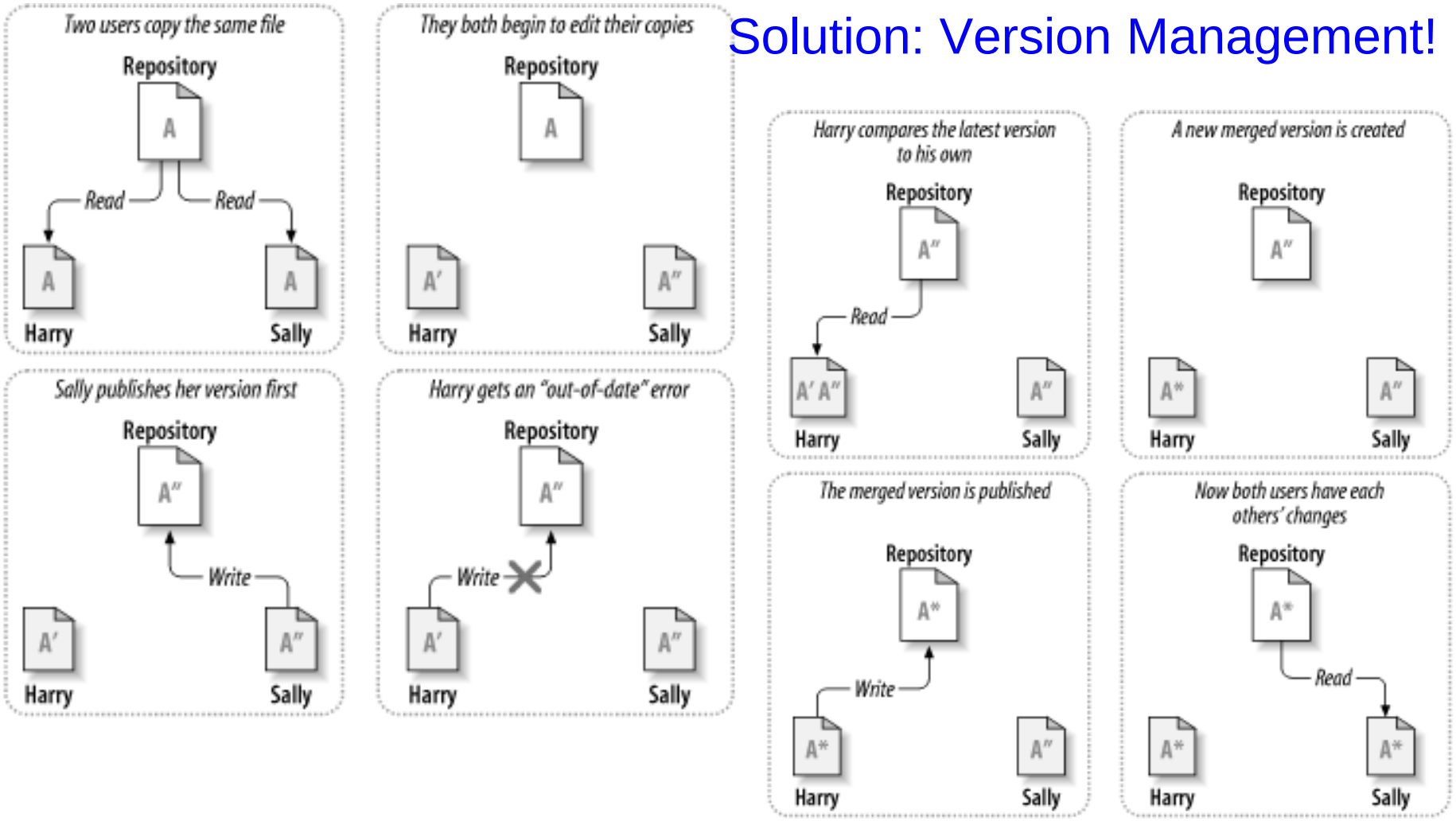
- Option 1: make them take turns
  - But then only one person can be working at any time
  - And how do you enforce the rule?
- Option 2: patch up differences afterwards
  - Requires a lot of re-working
  - Stuff always gets lost



[http://svnbook.red-bean.com/en/1.5/svn-book.html]

Problem #1: Collaboration

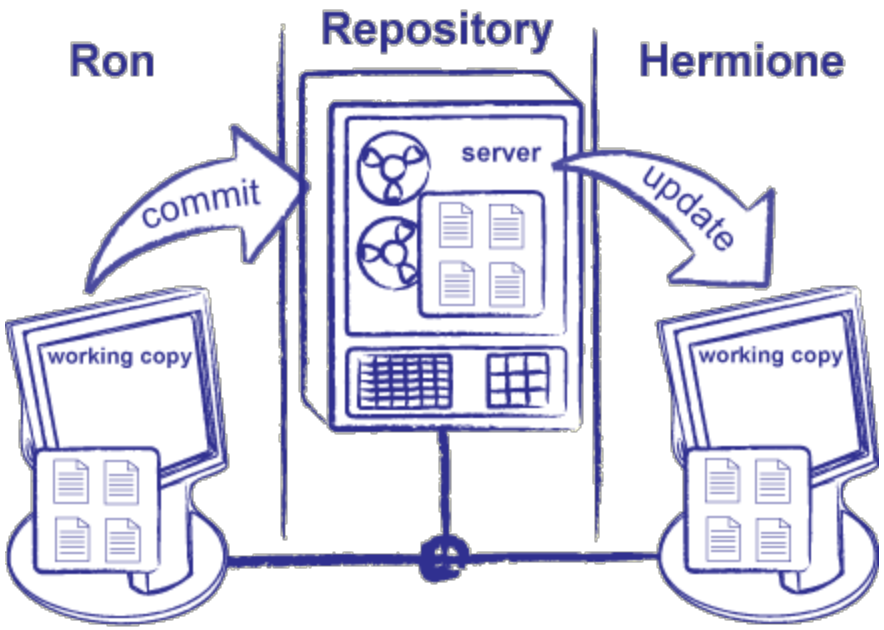
Solution: Version Management!



[http://software-carpentry.org/]

Solution: Version management

- The right solution is to use a version control system
- Keep the master copy of the file in a central repository
- Each author edits a working copy. When they're ready to share their changes, they commit them to the repository
- Other people can then do an update to get those changes



When working alone

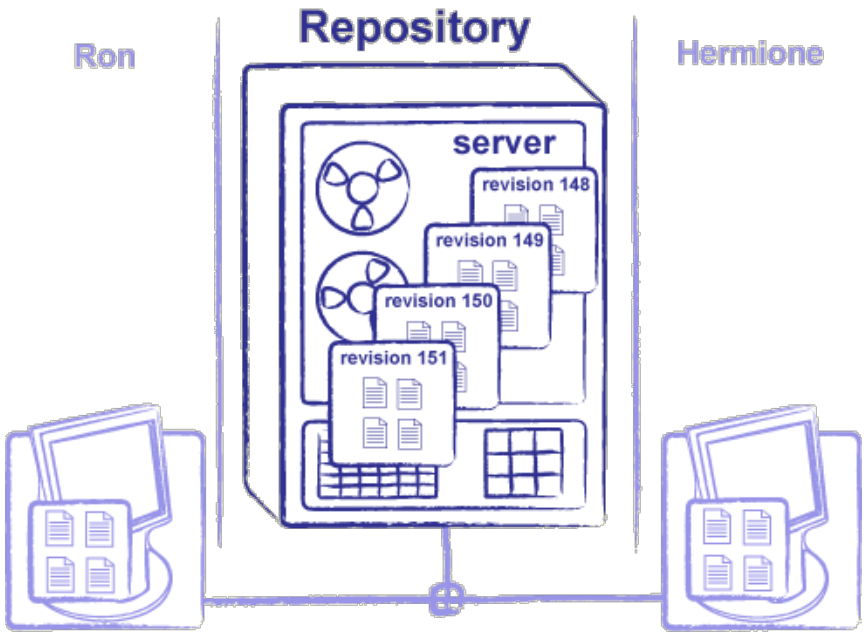
- This is also a good way for one person to manage files on multiple machines
  - Keep one working copy on your personal laptop, the lab machine, and the departmental server
  - No more mailing yourself files, or carrying around a USB drive (and forgetting to copy things onto it)
- This by itself is reason enough to use version control even when you are the only author

Problem #2: Undoing Changes

- Often want to undo changes to a file
  - Start work, realize it's the wrong approach, want to get back to starting point
  - Like "undo" in an editor...
  - ...but keep the whole history of every file, forever
- Also want to be able to see who changed what, when
  - The best way to find out how something works is often to ask the person who wrote it

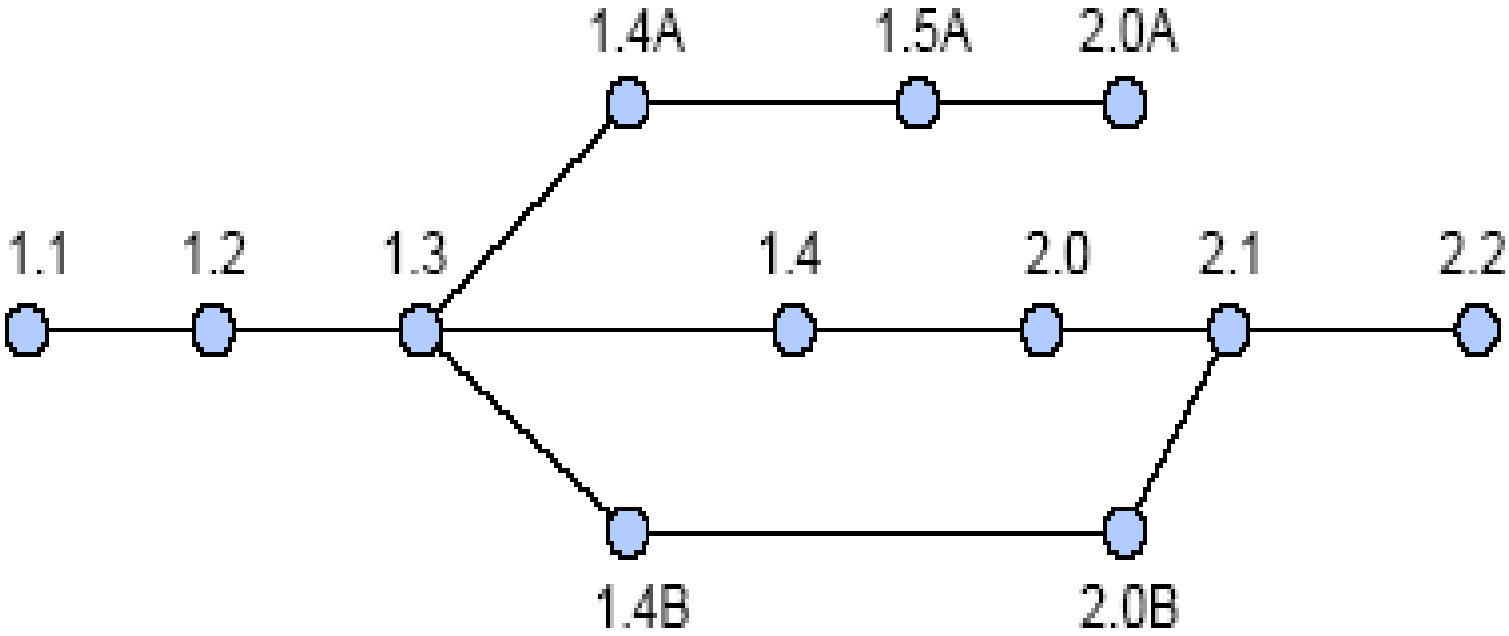
Solution: Version Control (again)

- Have the version control system keep old revisions of files
  - And have it record who made the change, and when
- Authors can then roll back to a particular revision or time
- (again) This by itself is reason enough to use version control even when you are the only author



Versions

Branching possible!



Work Distribution

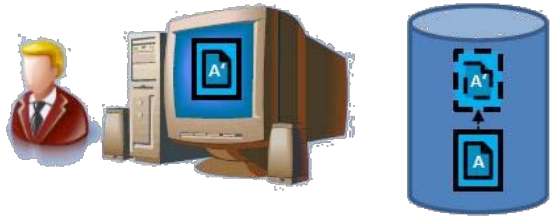
- Define and check access rights

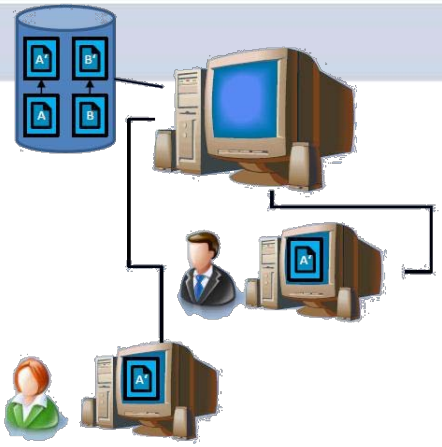
- Define and check parallel access
  - **Element based**: a developer can access a certain element whenever s/he wants
  - **Role based**: a developer can access a certain element whenever s/he performs a certain task (role)

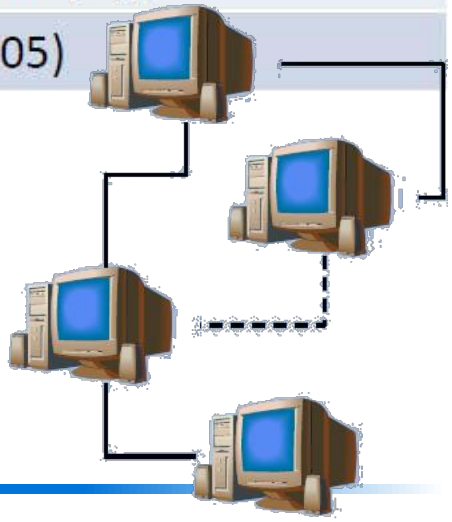
[Wiesner 2010]

Different solutions

Local Version Management	Central Version Management	Distributed Version Management
SCCS (1972)	Concurrent Version System (CVS , 1986)	MS Visual SourceSafe (komerziell, 1994)
SourceSafe (=> MS SourceSafe)	Subversion (SVN, 2004)	BitKeeper (kommerziell)
		Monotone
		Mercurial (hg)
		Git (2005)

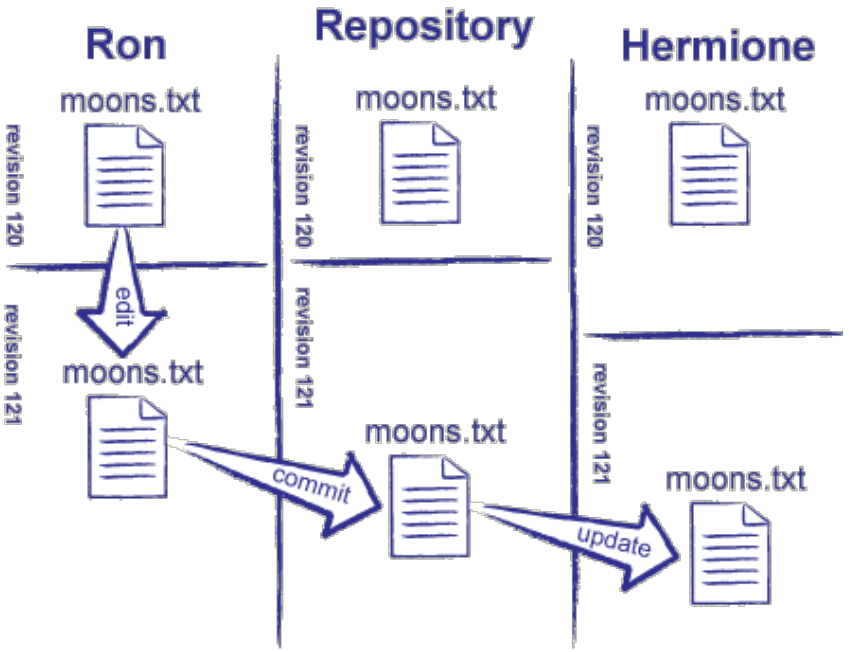






Basic Use (Subversion examples)

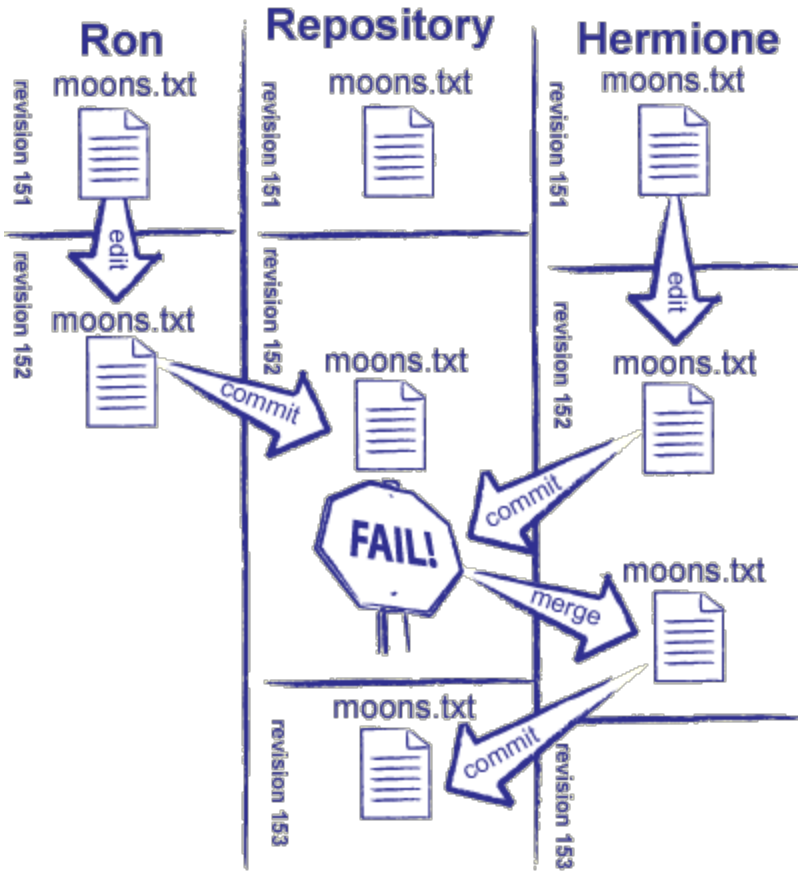
- Ron and Hermione each have a working copy of the solarsystem project repository
- Ron wants to add some information about Jupiter's moons
  - Runs `svn update` to synchronize his working copy with the repository
  - Goes into the jupiter directory and edits moons.txt
- Ron then:
  - Runs `svn commit` to save his changes in the repository
    - Repository is now at revision 121
- That afternoon, Hermione runs `svn update` on her working copy
  - Repository sends her Ron's changes



[http://software-carpentry.org/]

Resolving Conflicts

- Back to the problem of conflicting edits (or, more simply, conflicts)
- Option 1: only allow one person to have a writeable copy at any time
  - This is called pessimistic concurrency
  - Used in Microsoft Visual SourceSafe
- Option 2: let people edit, and resolve conflicts afterward by merging files
  - Called optimistic concurrency
  - "It's easier to get forgiveness than permission"
  - Most modern systems (including [Subversion]) do this



[http://software-carpentry.org/]

Starvation

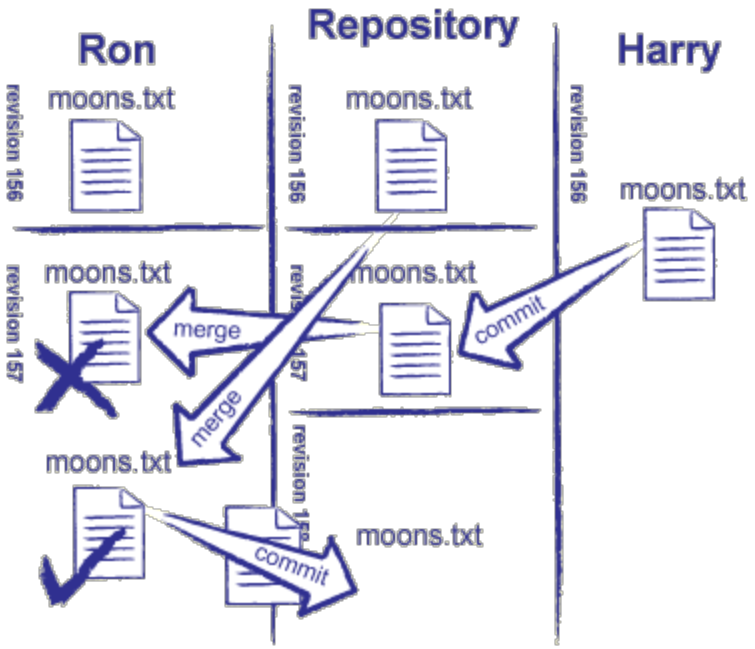
- But what happens if Ginny commits another set of changes while Hermione is resolving?
  - And then Harry commits yet another set?
- Starvation: Hermione never gets a turn because someone else always gets there first
- This is a management problem, not a technical one
  - Break the file(s) up into smaller pieces
  - Give people clearer responsibilities
  - The version control system is trying to tell you that people on your team are working at cross purposes
  - If you are doing things right, you will probably never (or rarely) encounter this

Reverting

- After doing some more work, Ron notices he's on the wrong path
- `svn diff` shows him which files he has changed, and what those changes are
- He hasn't committed anything yet, so he uses `svn revert` to discard his work
  - I.e., throw away any differences between his working copy and the master as it was when he started
  - Synchronizes with where he was, *not* with any changes other people have made since then (the base revision, not latest revision in the repository)
- If you find yourself reverting repeatedly, you should probably go and do something else for a while...

Rolling Back

- Now Ron decides that he doesn't like the changes Harry just made to moons.txt
  - Wants to do the equivalent of "undo"
- `svn log` shows recent history
  - Current revision is 157
  - He wants to revert to revision 156
- `svn merge -r 157:156` moons.txt will do the trick
  - The argument to the -r flag specifies the revisions involved
  - Merging allows him to keep some of Harry's changes if he wants to
  - Revision 157 is still in the repository
  - Remember, this affects *Ron's* local copy, he still needs to commit this undo if he wishes to commit these changes



[http://software-carpentry.org/]

Summary

- Version control is one of the things that distinguishes professionals from amateurs
  - And successful projects from failures
- Everything that a human being had to create should be under version control
- You'll see the benefits almost immediately
- You will want to put all your work (even solo work) under version control once you experience the benefits

Literature

- Software carpentry (http://software-carpentry.org)
- Version Control with Subversion **(**http://svnbook.red-bean.com/en/1.5/svn-book.html)
- N. Ford: Produktiv Programmieren, O Reilly, 2008
- B. Collins-Sussman, B. W. Fitzpatrick, C. M. Pilato: Versionskontrolle mit Subversion, 2008 (http://svnbook.red-bean.com/nightly/de/svn-book.html)