PHY604 Lecture 2

August 31, 2023

Review: Memory determines largest number that can be stored

- E.g., 1 byte: | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | $= 1x2^6 + 1x2^5 + 1x2^4 + 1x2^3 + 1x2^2 + 1x2^1 + 1x2^0 = 127_{10}$

  Sign bit; 0 if positive integer
  1 if negative integer

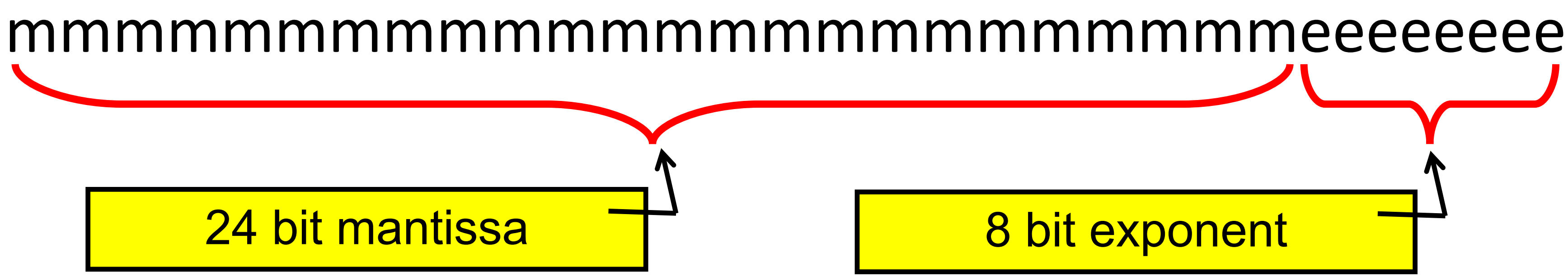
- 2-byte:
  - This can store 2^{15}-1 distinct values: -32,768 to 32,767 (signed)
  - Or it can store 2^{16} values: 0 to 65,535 (unsigned)

- Standard in many languages is 4-bytes
  - This can store 2^{31}-1 distinct values: -2,147,483,648 to 2,147,483,647 (signed)
    - C/C++: int (usually) or int32_t
    - Fortran: integer or integer(4)
  - Or it can store 2^{32} distinct values : 0 to 4,294,967,295 (unsigned)
    - C/C++: uint or uint32_t
    - Fortran (as of 95): unsigned

- For very big integers, 8-byte allows for 2^{64}
  - Fotran: integer(8)
  - C++: long

Review: Storing floating point data

- IEEE 754 mantissa-exponent form:



- Value = mantissa x 2 ^{exponent}
- Single precision:
  - Sign: 1 bit; exponent: 8 bits; significand: 24 bits (23 stored) = 32 bits
  - Range: 2^7-1 in exponent (because of sign) = 2^{127} multiplier ~ 10^{38}
  - Decimal precision: ~6 significant digits
- Double precision:
  - Sign: 1 bit; exponent: 11 bits; significand: 53 bits (52 stored) = 64 bits
  - Range: 2^{10}-1 in exponent = 2^{1023} multiplier ~ 10^{308}
  - Decimal precision: ~15 significant digits

Review: Real/Floating point numbers are more complicated

- Infinite real numbers on the number line need to be represented by a finite number of bits

- Finite memory results in limited **size and precision** of floating point numbers
  - Not all real numbers (even simple ones) can be stored in a finite number of digits in a base-2 representation
  - Example: $1/10 = 0.1_{10} = 0.0001100110011..._2$ does not have a finite representation in base 2 just as $1/3 = 0.333333..._{10}$ has no finite representation in base 10

- This means that even simple floating point numbers are often approximated with some small error
  - This means that floating point arithmetic is not exact! (on all computers and programming languages)

- Errors can compound if not treated carefully!

Review: Epsilon check for comparing floats

- Take two real numbers `a` and `b`
- We take `a==b` if `abs(a-b) < epsilon`

- Have to be very careful with this!!! We should think about:
  - The choice of `epsilon` based on the precision we require/expect for `a` and `b`
  - The choice of `epsilon` based on the magnitude of `a` and `b`
  - What will happen in special cases (`0, NaN, inf`)
  - …

Today’s lecture:

- Roundoff and truncation errors

- Good programming practices:
  - Version control

  - Testing

  - Misc. good practices

OTB: Round-off error example

- Imagine that we can only keep track of 4 significant digits
- Compute $\sqrt{x+1} - \sqrt{x}$
- Take *x* = 1984. Keeping only 4 digits each step of the way:

$$\sqrt{x+1} - \sqrt{x} = 44.55 - 44.54 = 0.01$$

- We've lost a lot of precision
- Instead, consider:

$$\sqrt{x+1} - \sqrt{x} = (\sqrt{x+1} - \sqrt{x}) \left(\frac{\sqrt{x+1} + \sqrt{x}}{\sqrt{x+1} + \sqrt{x}} \right) = \frac{1}{\sqrt{x+1} + \sqrt{x}}$$

- Then

$$\sqrt{1985} - \sqrt{1984} = \frac{1}{\sqrt{1985} + \sqrt{1984}} = \frac{1}{44.55 + 44.54} = 0.01122$$

Roundoff error: Another example

- Consider computing exp(-24) via a truncated Taylor series:

$$e^x \simeq S(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + ... + \frac{x^n}{n!}$$

- Error in the approximation (**i.e., truncation error**) is less than:

$$\frac{|x|^{n+1}}{(n+1)!} \max\{1, e^x\}$$

- But if we compute S(-24) by adding terms until they are less than machine precision (8 byte):
  - S(-24)=3.7814382919759864E-007
  - Exp(-24)=3.7751345442790977E-011
  - Error is larger than the result (much larger than truncation error)!!
  - Looking at terms, we see we are relying on cancellations of terms

How can we make is more accurate? Choose a different algorithm

- Realize that:

$$e^{-24} = (e^{-1})^{24} \Rightarrow S(-24) = S(-1)^{24}$$

- $S(-1)^{24}$ = 3.7751345442791294E-011
- exp(-24) = 3.7751345442790977E-011

Truncation errors are different from roundoff

- Translating continuous mathematical expressions into discrete forms introduces truncation error
- For example: $e^x \simeq S(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + ... + \frac{x^n}{n!}$
- Error: $\frac{|x|^{n+1}}{(n+1)!} \max\{1, e^x\}$
- Or $f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$ vs. $D_h(x) = \frac{f(x+h) - f(x)}{h}$

Floating point arithmetic not associative

- Adding lots of numbers together can compound round-off error
- One solution: sort and add starting with the smallest numbers
- Kahan summation (see reading list)
  - Algorithm for adding sequence of numbers while minimizing roundoff accumulation
  - Keeps a separate variable that accumulates small errors
  - Requires that the compiler obey parenthesis

Floating point arithmetic not associative:

$$(1.0 - 1.0) + 10^{-9} \stackrel{?}{=} 1.0 + (-1.0 + 10^{-9})$$

```
! Purpose:  Test the precision of reals
! Author:    Cyrus Dreyer
! Date:        2/4/2019
program test_prec_reals
  implicit none         ! Turn off implicit typing
  ! Variable dictionary
  real :: factor1     ! Variable for factor 1
  real :: factor2     ! Variable for factor 2
  real :: prec_test_lhs ! Variable for result
  real :: prec_test_rhs ! Variable for result

  factor1 = 1.0           ! Assign a value to factor1
  factor2 = 1.0d-9        ! Assign a value to factor2

  prec_test_lhs = (factor1-factor1) + factor2 ! LHS of inequality on slide
  prec_test_rhs = factor1 + (-factor1 + factor2) ! RHS of inequality on slide

  ! Output
  write(*,'(a20,e20.12e2,a20,e20.12e2)') "Prec_test_lhs:",prec_test_lhs, &
       "Prec_test_rhs:", prec_test_rhs

  stop 0                ! Stop execution of the program
end program test_prec_reals
```

Today’s lecture:

- Roundoff and truncation errors

- Good programming practices:
  - Version control
  - Testing
  - Misc. good practices

Software engineering practices

- Some basic practices that can *greatly* enhance your ability to write maintainable code
  - Version control
  - Build environments
  - Testing procedures
  - Automatic code error checking
  - Profiling
  - Documentation
- There are many tools that will help you write safe code and find bugs as they are introduced. These let you focus more on the science.
- Main goal of this lecture is to just show you what kind of tools are out there and how they can help your workflow

Coding experiences to try and avoid

- *You swear that the code worked perfectly 6 months ago*, but today it doesn't, and you can't figure out what changed
- *Your research group is all working on the same code*, and you need to sync up with everyone's changes, and make sure no one breaks the code
- *Your code always worked fine on machine X*, but now you switch to a new system/architecture, and you code gives errors, crashes, ...
- *Your code ties together lots of code*: legacy code from your advisor's advisor, new stuff you wrote, all tied together by a driver. The code is giving funny behavior sometime—how do you go about debugging such a beast?

Version control

- What is it?
  - A system that records changes to a file or set of files over time so that you can recall specific versions late
- Why is it important?
  - So that if the code stops working, you can go back to specific previous versions to see what changes broke it
  - Allows you to compare changes over time
  - If multiple people are working on a file, see who last modified something that might be causing a problem, who introduced an issue and when, etc.

Types of version control: Local

- Previous versions (or patch sets) stored elsewhere on local machine

- Can be as simple as copying files into a different folder to store them before making changes
  - Will take up a lot of memory if not done in a smart way

- There are some tools to make this more consistent such as GNU RCS

- Pros: Simplicity
- Cons: Single point of failure

Types of version control: Centralized

- Have a single server that contains all the versioned files, stores history and changes

- User communicates with the server to:
  - Checkout source
  - Commit changes back to the source
  - Request a log (history) of a file from the server
  - Diff your local version with the version on the server

- Has advantages over local version control:
  - Everyone knows what everyone else is doing on a project
  - Administrators have control over who can do what

- Cons: Does not scale well for large projects, single point of failure

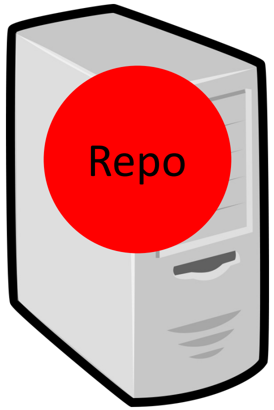
Types of version control: Distributed

- Clients fully mirror (i.e., clone) the repository and its history on their local machine
  - Not just the latest snapshot of the files
  - No single point of failure: if any server dies, any client repository can be copied back to restore it

- Deals well with multiple different groups simultaneously working on a project
  - Easy to “fork”

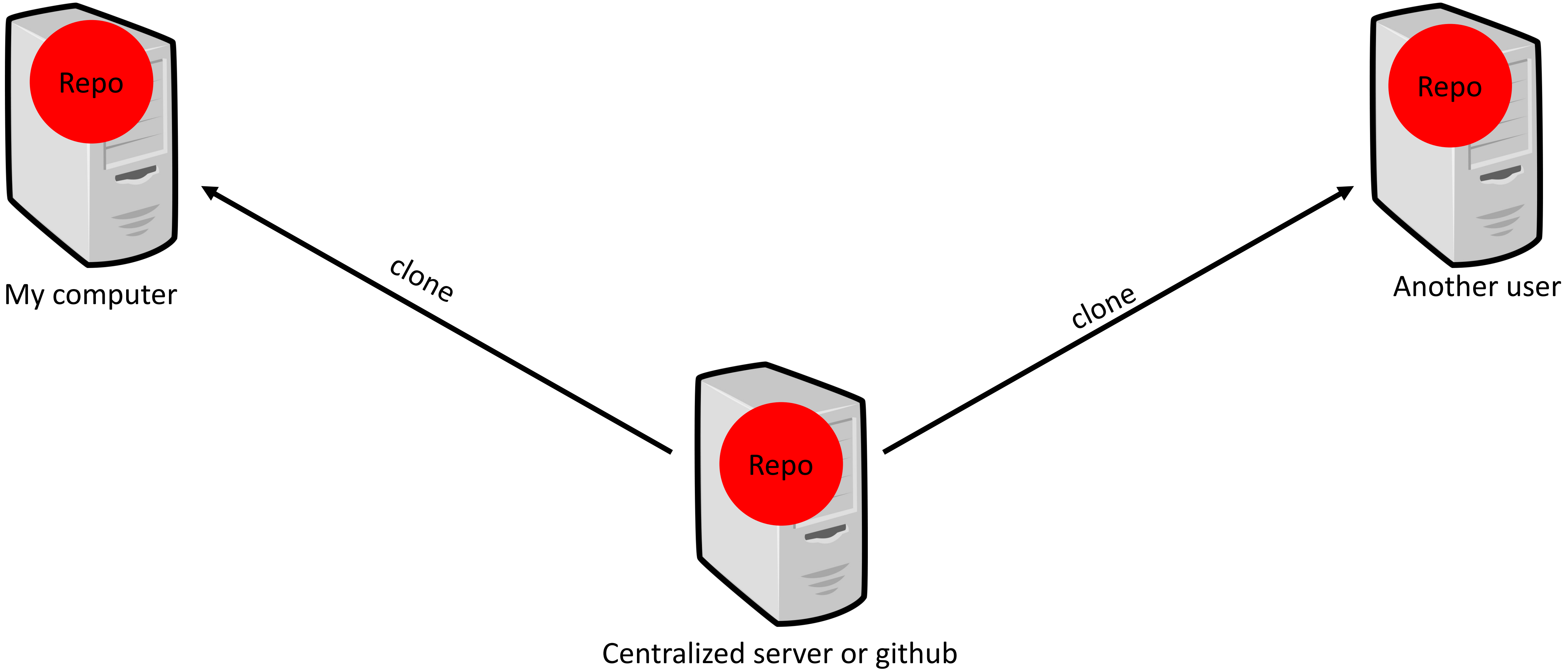
- Common DVCS: **Git**, Mercurial, Bazaar

Distributed version control

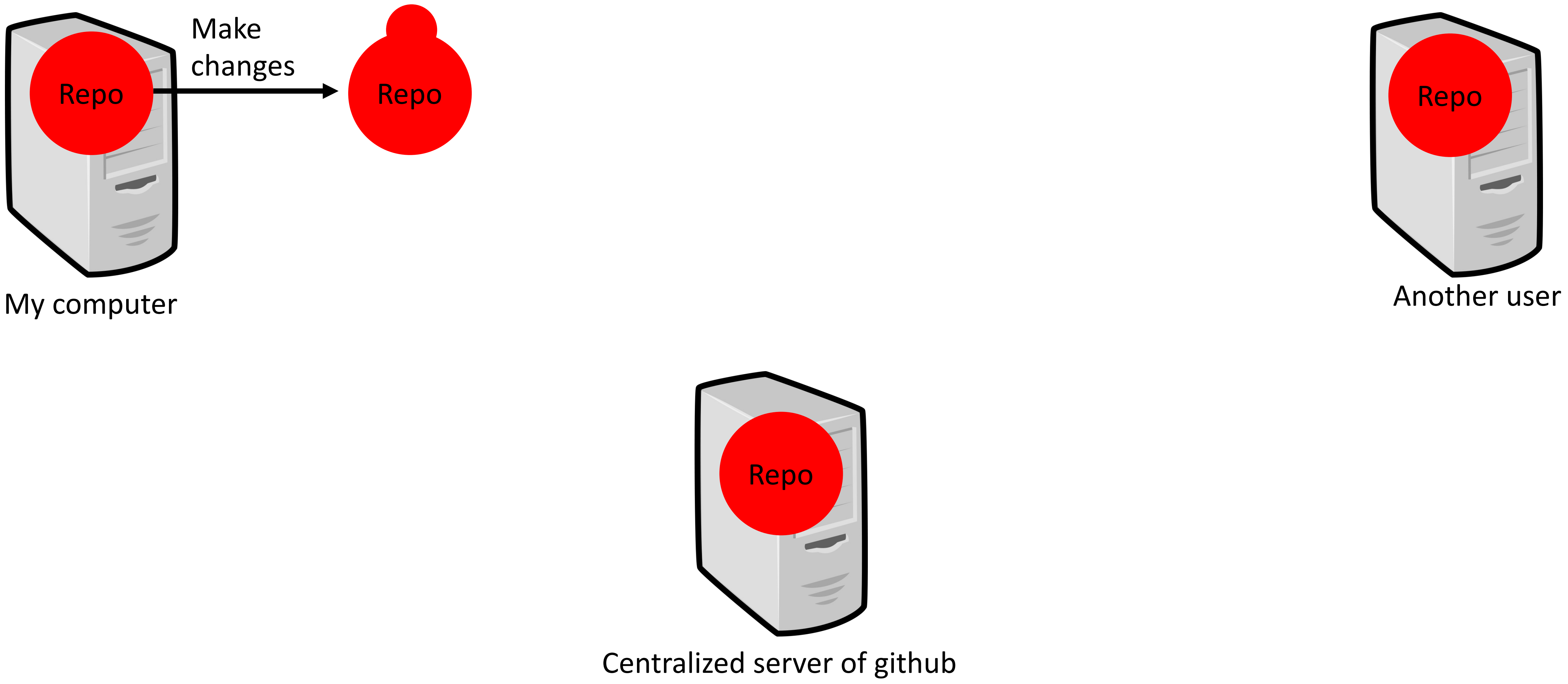


Centralized server or github

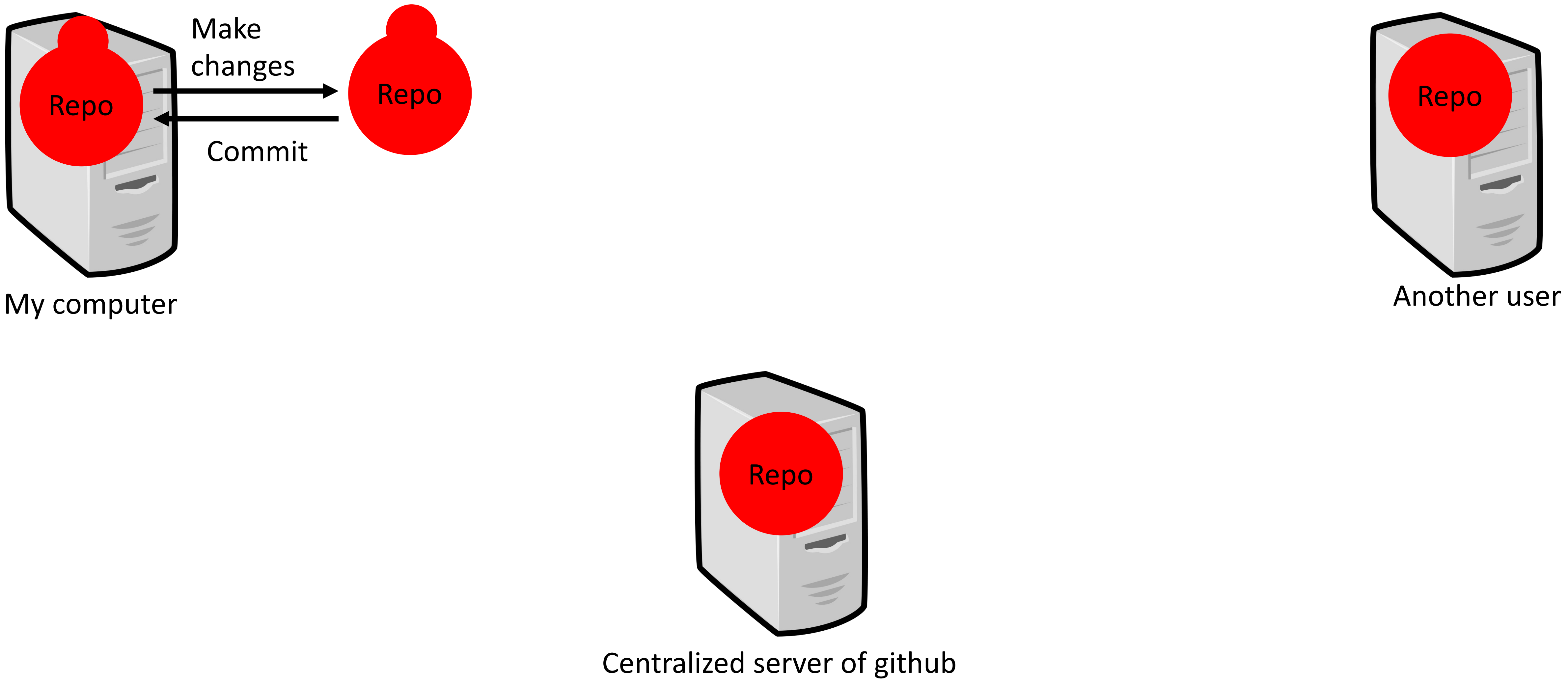
Distributed version control



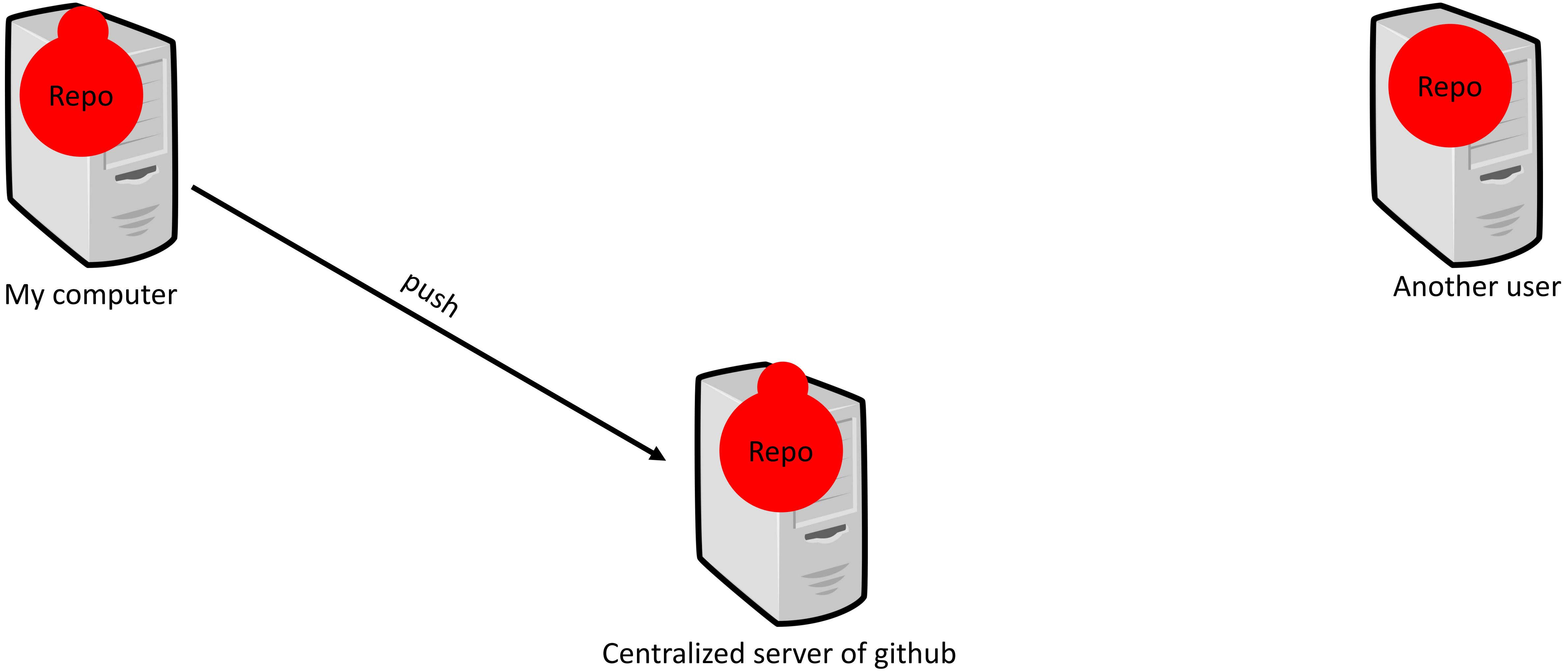
Distributed version control



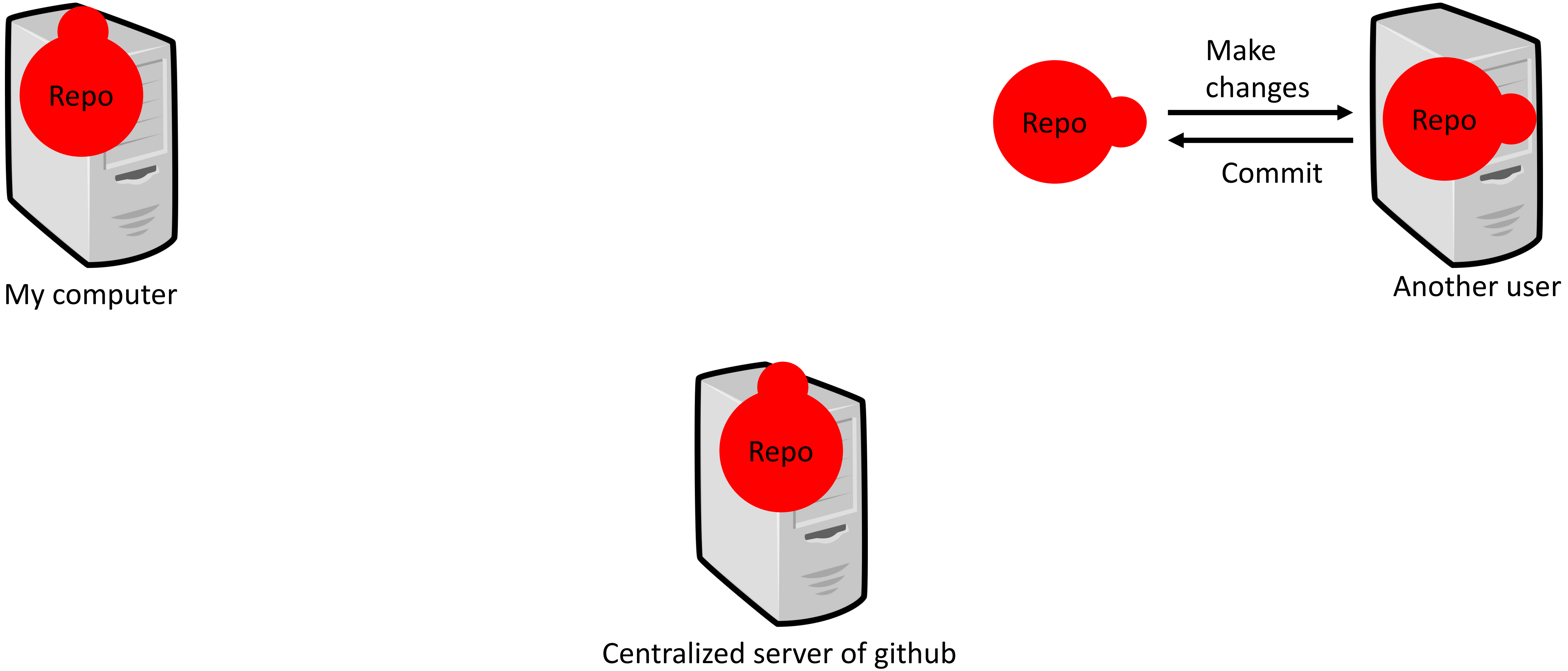
Distributed version control



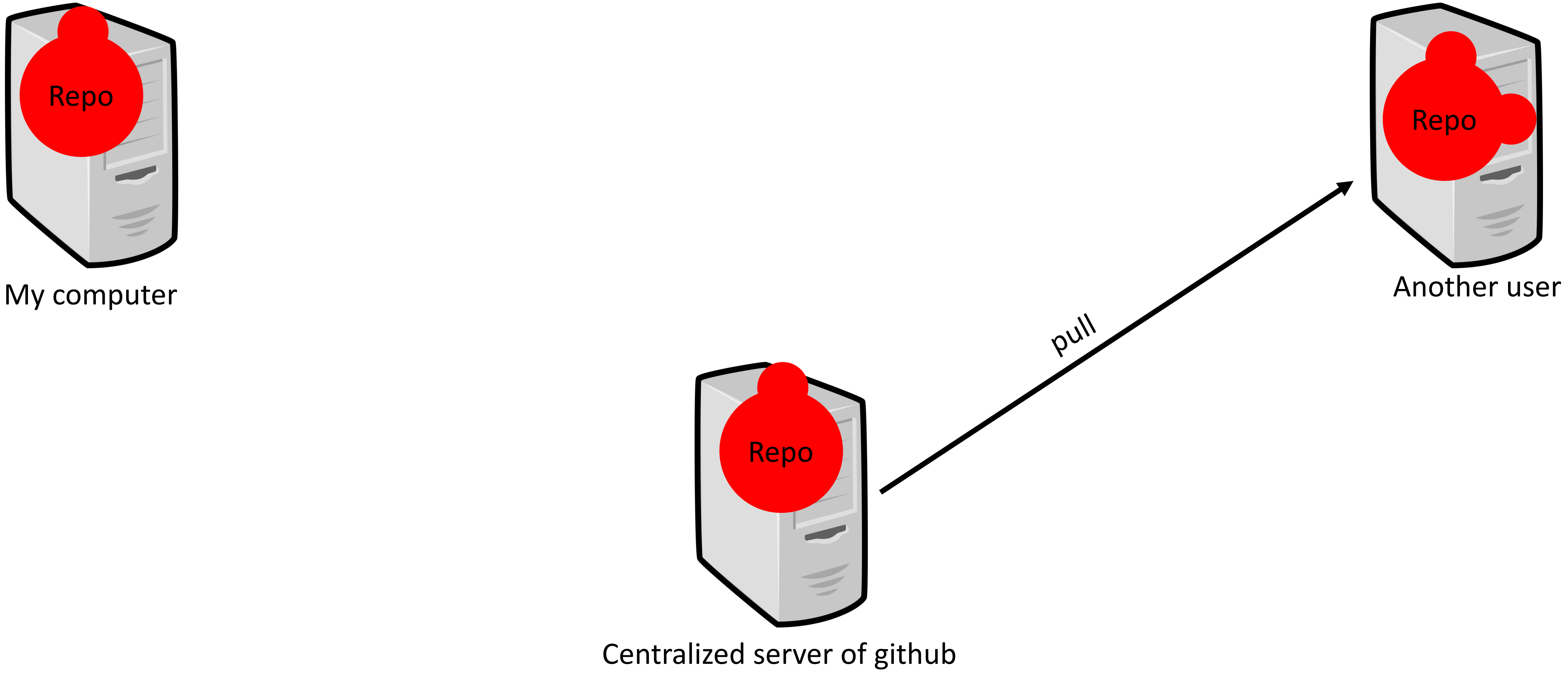
Distributed version control



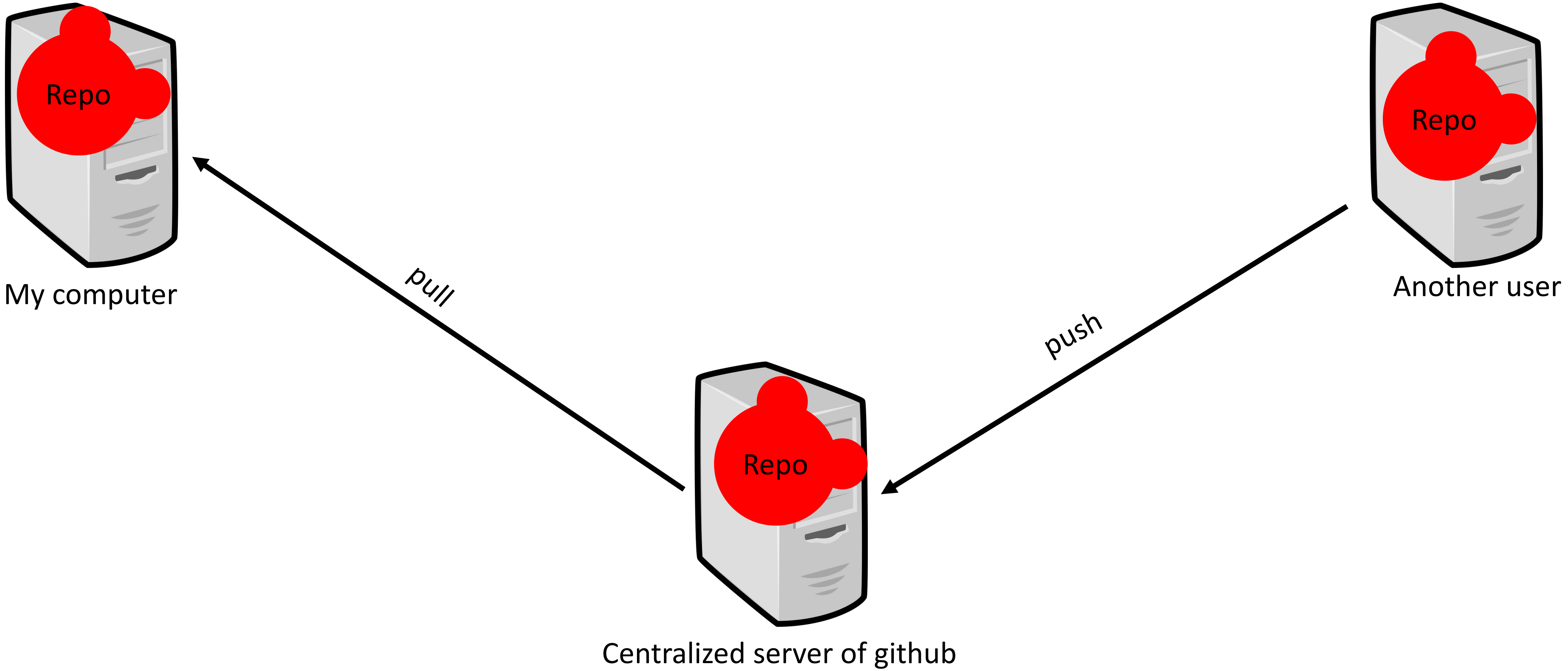
Distributed version control



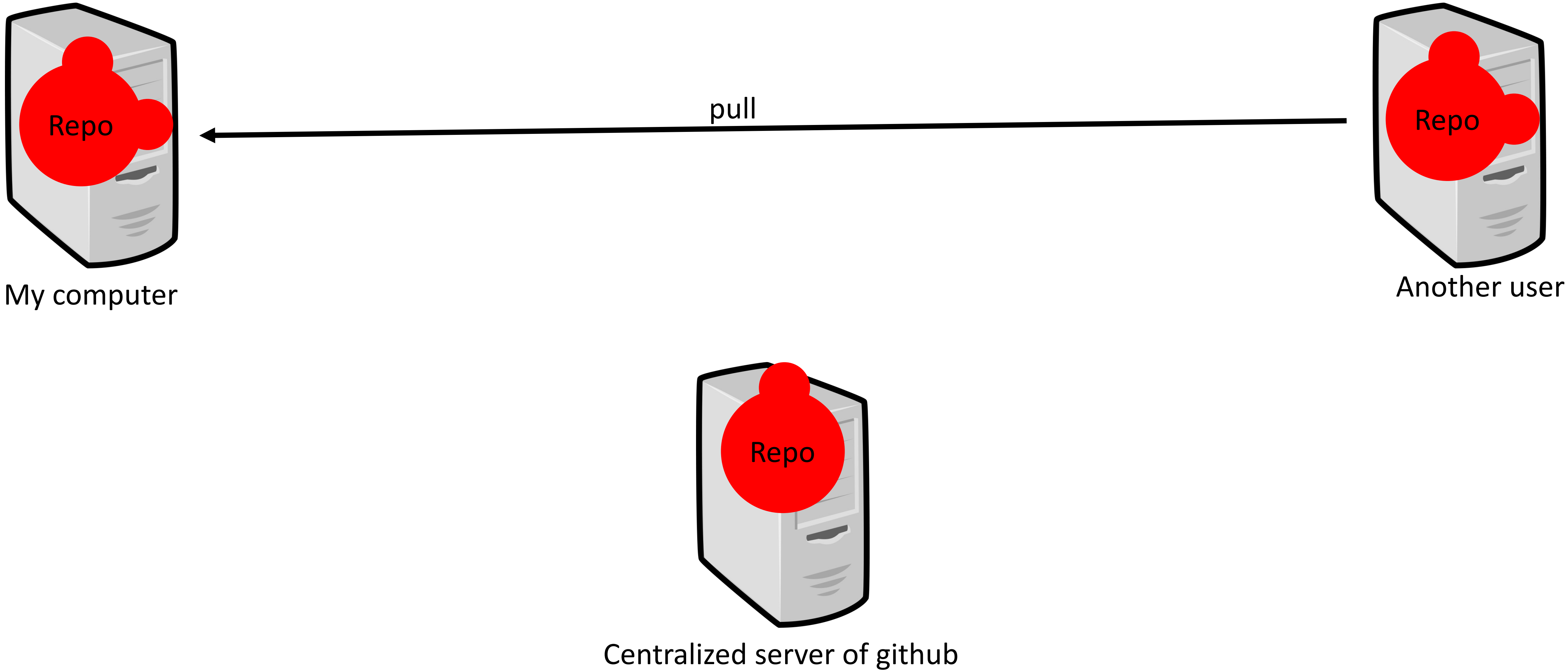
Distributed version control



Distributed version control



Distributed version control



Comments about Git

- Note that with git, every change generates a new “hash” that identifies the entire collection of source.
  - You cannot update just a single sub-directory—it's all or nothing.
- Branches in a repo allow you to work on changes in a separate are from the main source.
  - You can perfect them, then merge back to the main branch, and then push back to the remote.
  - Overall, very light weight!!
- LOTS of resources on the web (see readings)
  - Best way to learn is to practice.
  - There is more than one way to do most things

Example: “Local” version control with Git

- You can use Git to do local version control on your computer:

- **`git init`** to create a new git repository
- **`git add`** to add file contents to the index
- **`git commit`** to record changes to the repository
- **`git log`** to show previous commits
- …

Branching with git

- One of the killer apps of git is lightweight "branching"
  - Creates a different line of development which can be merged back into the main one
  - Does not require making multiple copies of source code, etc.

- Allows you to work in different directions and later merge together as you wish

- Git will help if there are conflicting changes

After class tasks

- No office hours today
- If you do not already have one, make an account on github: https://github.com/
- Readings:
  - What every computer scientist should know about floating-point arithmetic
  - Wikipedia page on the Floating Point
  - Wikipedia page on the Kahan Summation Algorithm

  - Pro Git online book