C++ Technical Detail on Pointers (Continued from Last Time)

UNIVERSITY OF MINNESOTA
PROFESSOR DANIEL F. KEEFE

Pointers in C++

Consider:

```
int size = 100;
int* ps; // pointer to int
//int *ps, notapointer;
```

Defining ps

```
ps = &size; // ps points to size
```

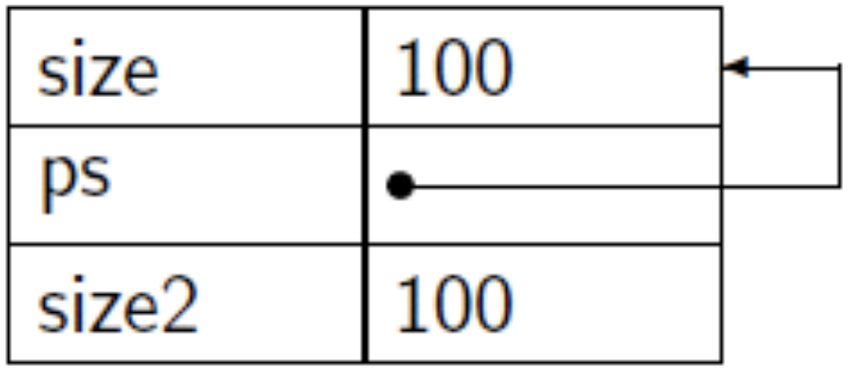
`&size` is the address where size exists in memory

Referencing the data ps points to

```
int size2 = *ps; // size2 == 100
```

Visualizing pointers

- A visual model of memory:



- Memory cells are labelled by names and store values.
- Some values are pointers to other cells.

Exercise

- Draw the model of memory to illustrate the state of memory after the following code fragment has executed.

```
char first = 'A' ;
char second = 'B' ;
char third = 'C' ;
char *p1 = 0 ;
char *p2 = 0 ;
char *p3 = 0 ;
p1 = &first ;
p2 = p1 ;
p3 = &second ;
*p1 = third ;
```

Pointers, evaluation

- The expression `ps` evaluates to the address held in `ps`.
- The expression `*ps` evaluates to the value stored in the address held by `ps`.
- `ps = 0;`
  - makes ps point to “nothing”.
  - No valid data can be stored at address 0.
  - This is “setting it to null”.
  - Can also write: `ps = NULL;`
- It is good programming practice to always initialize pointers to NULL:
  - `int *ps = NULL;`

Pointers, comparison

- If `ps == 0` then evaluating `*ps` has unspecified behavior, in other words “big crash”, “boom”.
- Thus, we also test that pointers are not equal to 0 before accessing the data they may point to.
- `if (ps != NULL) ... *ps ...`
- `if (ps) ... *ps ...`
- These are equivalent - C boolean operators work on integers, 0 = false, anything else = true.

```
int size = 100;
int *ps = 0;
int size2 = 0;
int *ps2 = 0;
ps = &size;
ps2 = &size2;
size2 = *ps;
if (size == size2)
  cout << "size == size2" << endl;
else
  cout << "size != size2" << endl;
if (ps == ps2)
  cout << "ps == ps2" << endl;
else
  cout << "ps != ps2" << endl;
if (*ps == *ps2)
  cout << "*ps == *ps2" << endl;
else
  cout << "*ps != *ps2" << endl;
```

What does this print out?

Arrays in C and C++

Arrays are implemented using pointers.

```
int x[5] = {1,4,9,16,25};
```

This allocates and initializes an array of size 5 and has `x` point to the first element of the array.

Indexing begins at 0, so

```
x[0] == 1, ..., x[4] == 25
```

Arrays and Short-Circuit Evaluation

Example: The following loop finds a value in x greater than 15 if it exists.

```
int i = 0;
int x [5] = {1,4,9,16,25};

while (i < 5 && x[i] <= 15) {
   i++;
}

if (i < 5)
   cout << "found one " << x[i];
else
   cout << "didn't find one";
```

Pointer Arithmetic

- This loop is equivalent but uses **pointer arithmetic**.

```
int i = 0;
int x[5] = {1,4,9,16,25};
int *y = x;
while (i < 5 && *y <= 15) {
   i++;
   y++;  // here we add 1 to the pointer
}
if (i < 5)
   cout << "found one " << *y;
else
   cout << "didn't find one";
```

Pointer Arithmetic

Adding 1 to a pointer, moves it to the next data object after it.

We assume that objects are stored contiguously.

We can add any integer value, as in `x + 4` above

`x[0]` is the same as `*x`.

Brushwork As A Group Project

3 Things to Know About the Brushwork Project Before Tomorrow's Lab

- Group Handin Requirements (source code + a few design documents)
- Read "Couch Potatoes and Hitchhikers" :)
- Preview Team Policies and Expectations Document

UNIVERSITY OF MINNESOTA
PROFESSOR DANIEL F. KEEFE

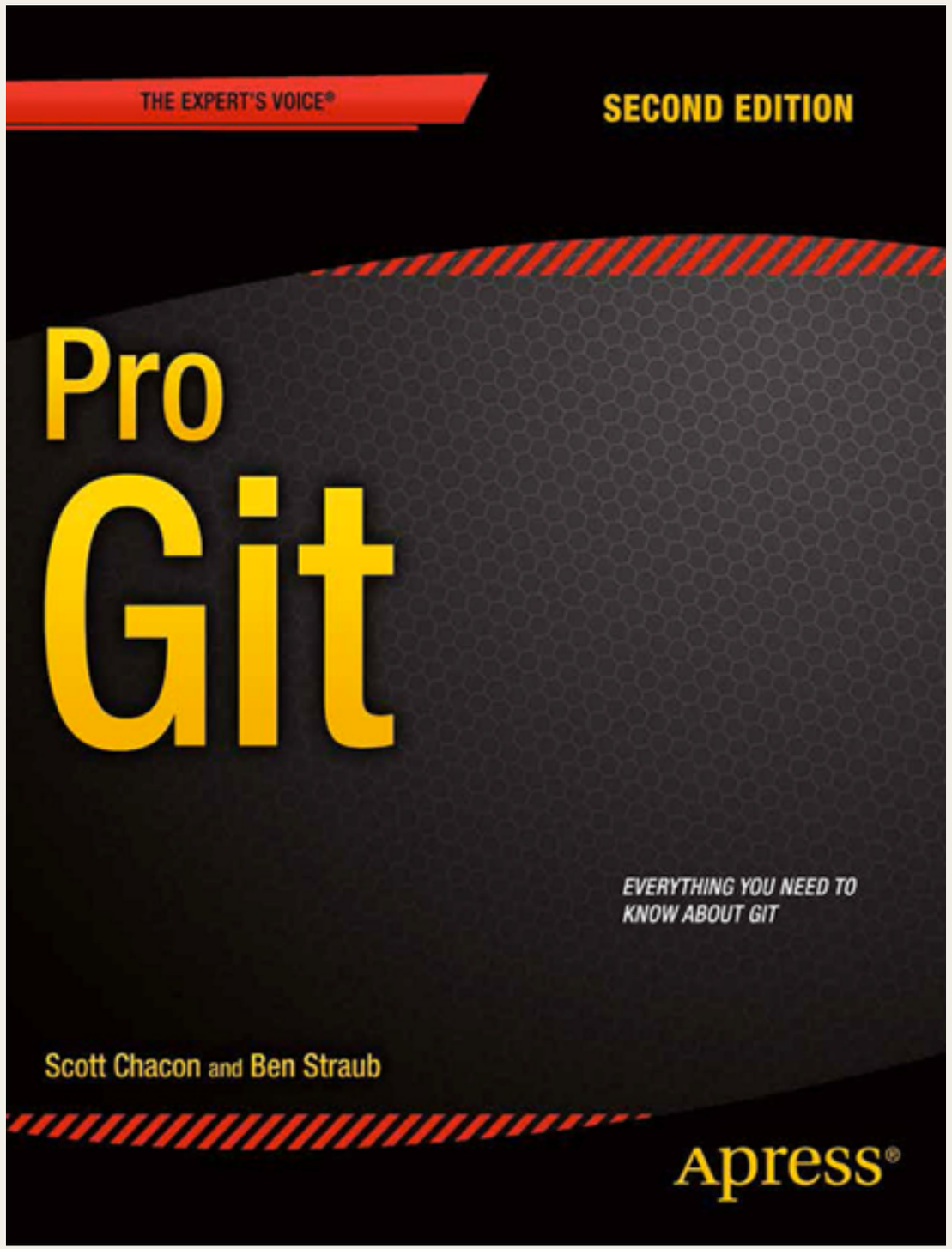
A Brief Intro to Git

UNIVERSITY OF MINNESOTA
PROFESSOR DANIEL F. KEEFE

How do I learn about source control in general and Git specifically?

1. Just start using it…
    - Start a project with a few files in it.
    - Add new files when you need to.
    - Commit your changes as a way of saving, backing up, and sharing your work with your team.
    - Figure out the more advanced features later when you need them.

2. Read a book!

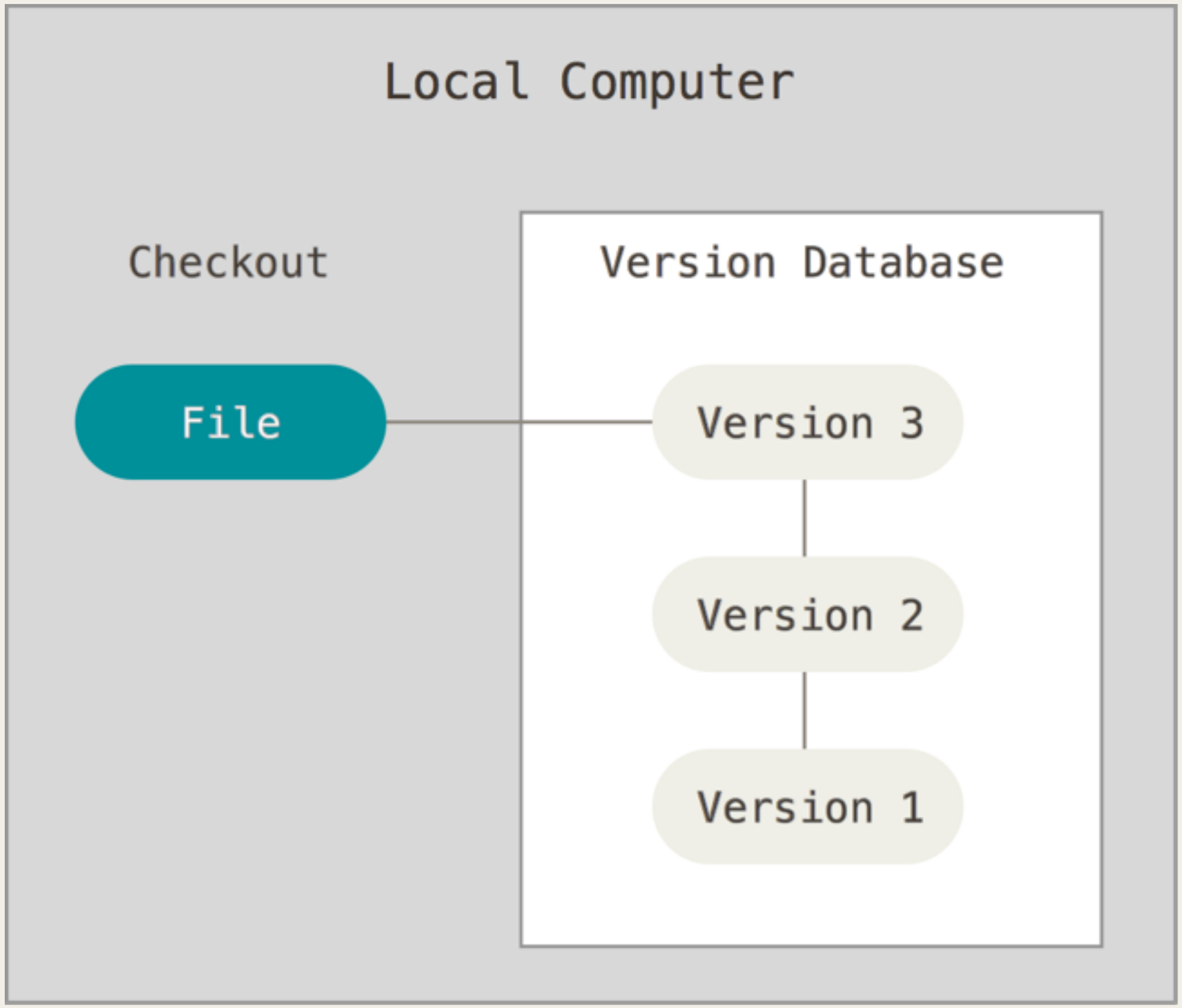


http://git-scm.com/book/en/v2

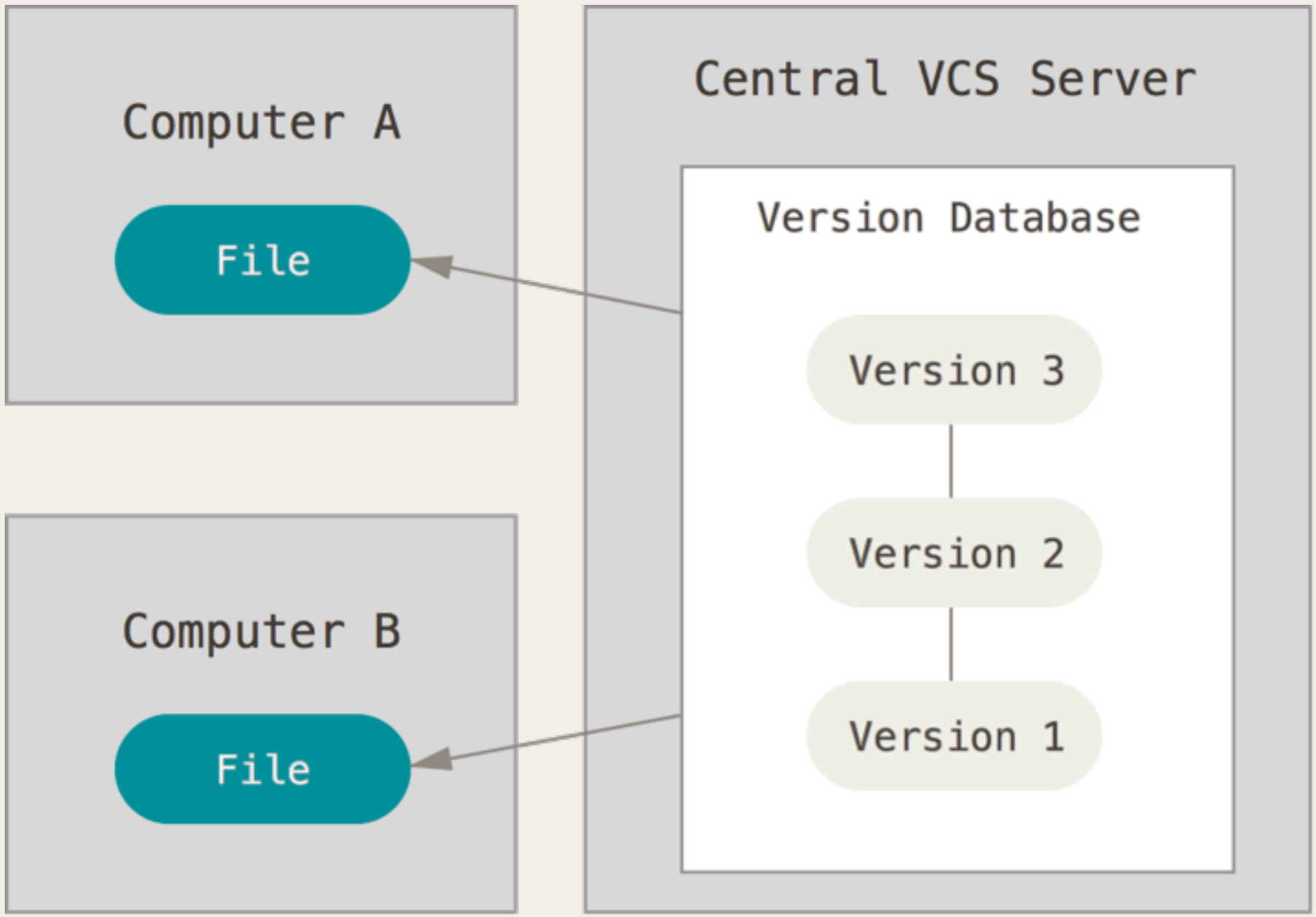
The whole thing is available online.

All the figures in the next slides come from this book.

Local Version Control

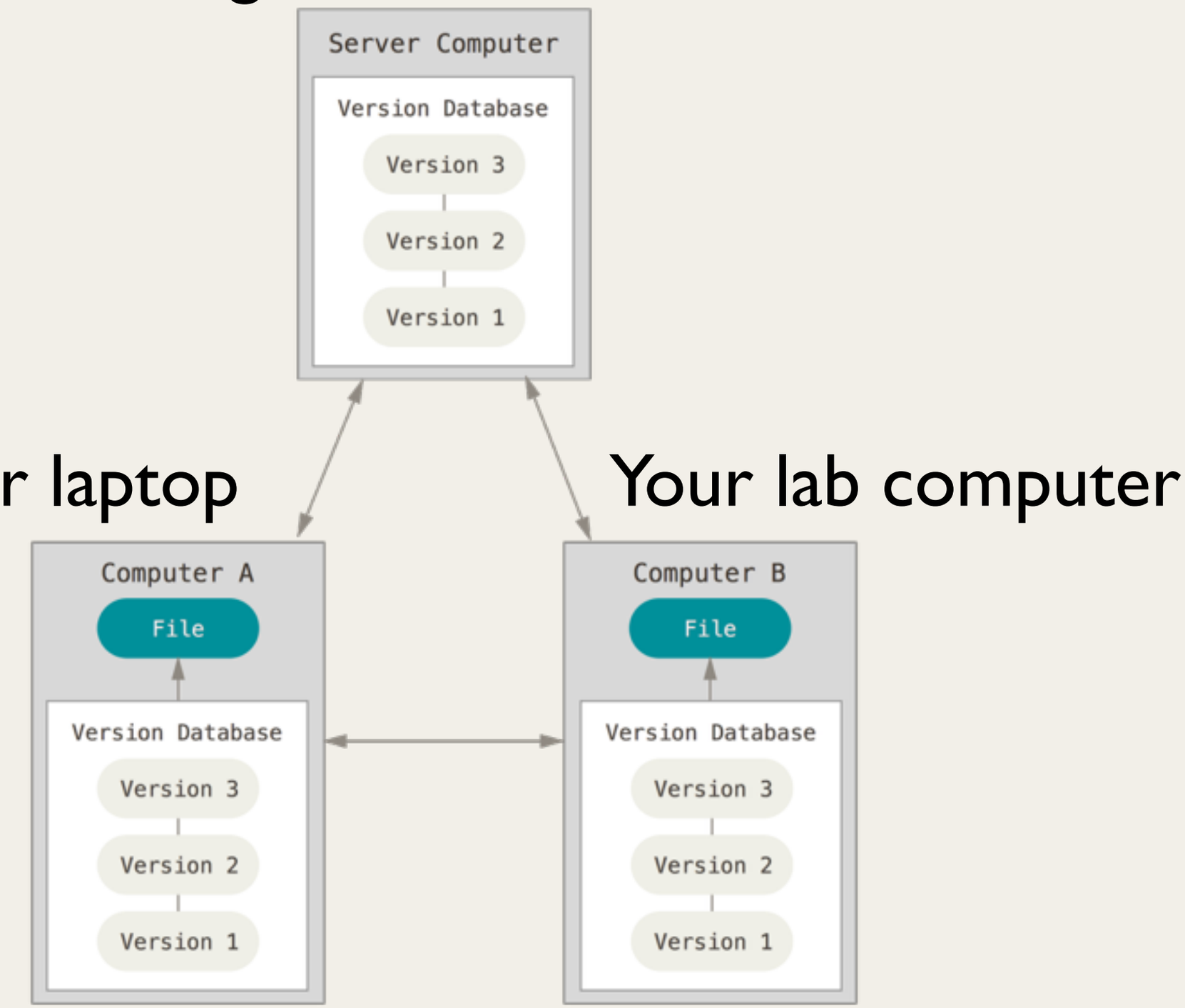


Centralized Version Control



Distributed Version Control (Git uses this)

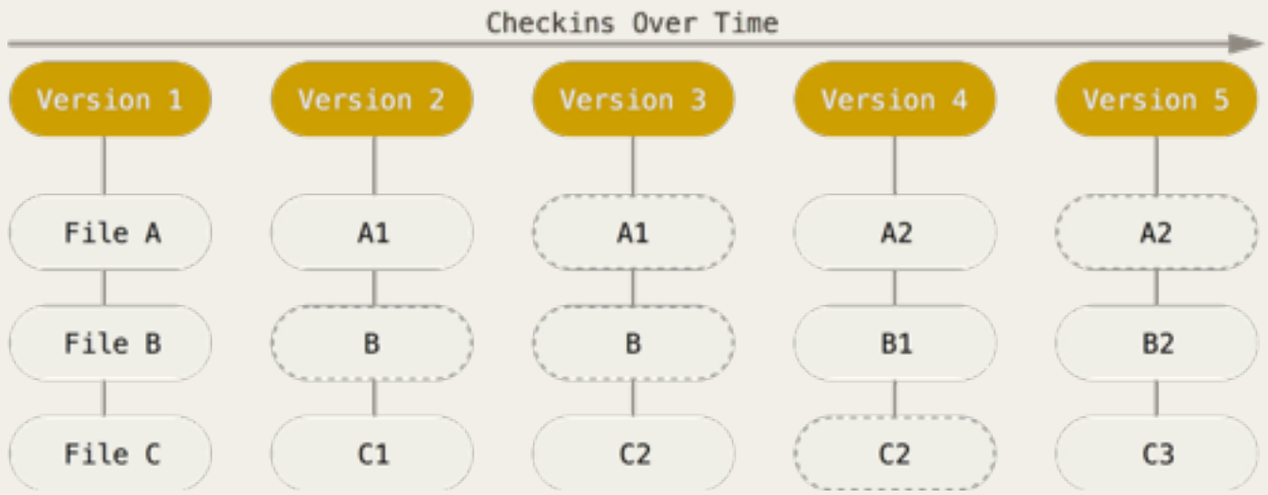
github.umn.edu



Git stores snapshots of a Whole Project

If you read more about version control systems you might see them described this way — tracking changes to each file. If you know exactly how each file changes over time, then it's possible to recreate the state of a whole project as of some previous date, but this isn't always easy because the focus is on individual files not the project as a whole.

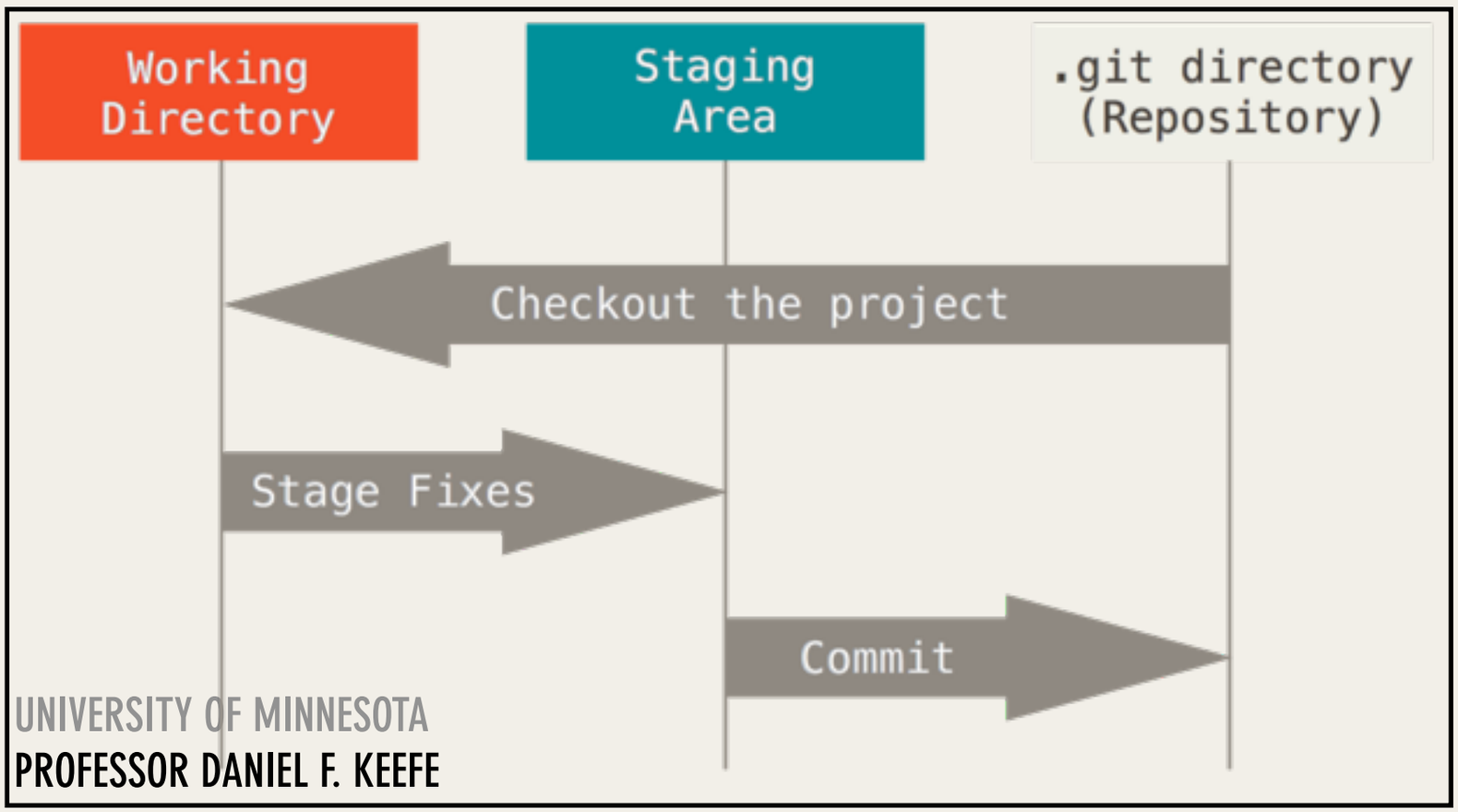
But, git is different (and I think more intuitive). With git we think about tracking changes to an entire project and we can refer to different versions of the whole project very easily.

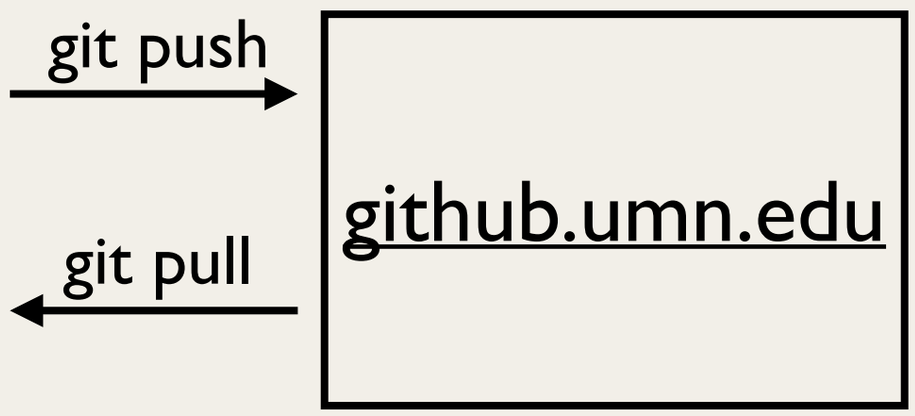


Most Important Slide: The 3 States of Git

- Each file will be in one of these three states:
  - **committed:** data safely stored in your local database
  - **modified:** you have changed the file but have not yet committed it to your database
  - **staged:** you have marked a modified file in its current version to go into your next commit snapshot.

all on your computer

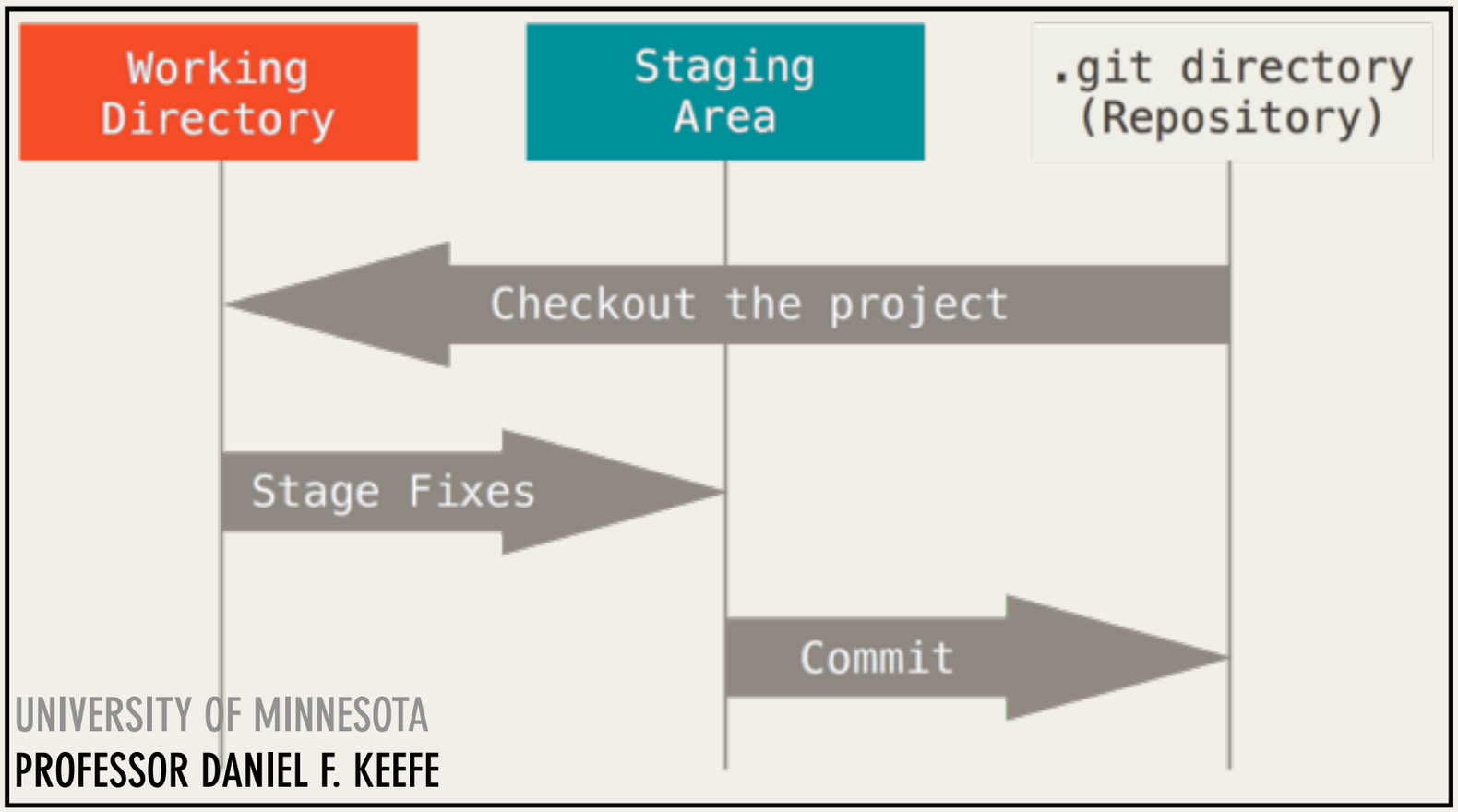


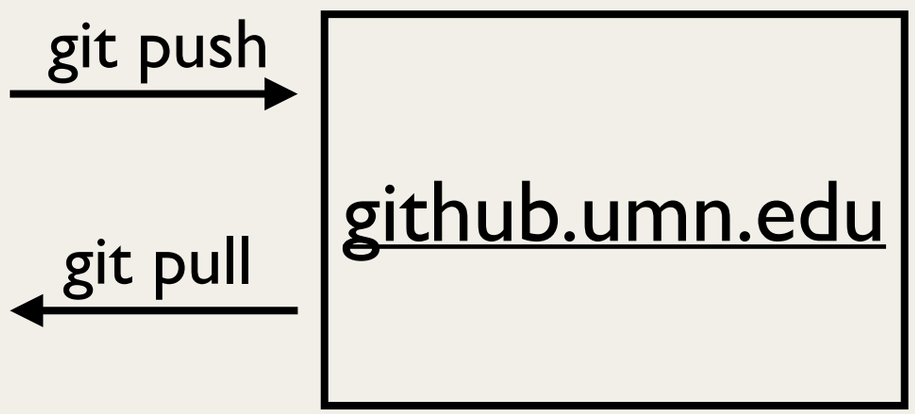


Typical Workflow

1. You do a pull to bring in any recent changes made by your team members.
2. You modify files in your working directory.
3. You stage the files, adding snapshots of them to your staging area.
4. You do a commit, which takes the files as they are in the staging area and stores the snapshot permanently in your .git directory.
5. You do a push, which sends these the committed changes up to the server so that other programmers on your project team can access them by running git pull on their machines.

all on your computer





Starting to Use Git

- You can initialize a blank project with Git.
- Or, what is most fun is to clone some other project:
  - Clone the TA's support code and start building on it in your own project.
  - Clone some cool project you find on github.com.
  - Use a command like:

```
git clone https://github.com/libgit2/libgit2
```

Tracking New Files

- `git add README`

Tracking New Files

```
$ git add README
```

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README
```

Staging Modified Files

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   CONTRIBUTING.md
```

```
$ git add CONTRIBUTING.md
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README
    modified:   CONTRIBUTING.md
```

Committing Staged Files

```
$ git commit -m "Story 182: Fix benchmarks for speed"
[master 463dc4f] Story 182: Fix benchmarks for speed
 2 files changed, 2 insertions(+)
 create mode 100644 README
```

Pushing the Changes You Committed Out to the World

```
$ git push
```

Pulling in Changes Others Have Committed

```
$ git pull
```

- In order for these to work, you have to have one or more “Remote” repositories linked to yours.
- This link is setup automatically when you clone, but you can also add and remove remote links later.