



# C/C++ Unix Dev Tutorial

Maslab 2003

# Outline

- ◆ C/C++ Basics
- ◆ C/C++ Specifics
- ◆ C/C++ Debugging
- ◆ Other Tools

# C/C++ Basics

## ◆ Toolflow

- Various files and what they represent
- g++ command line usage

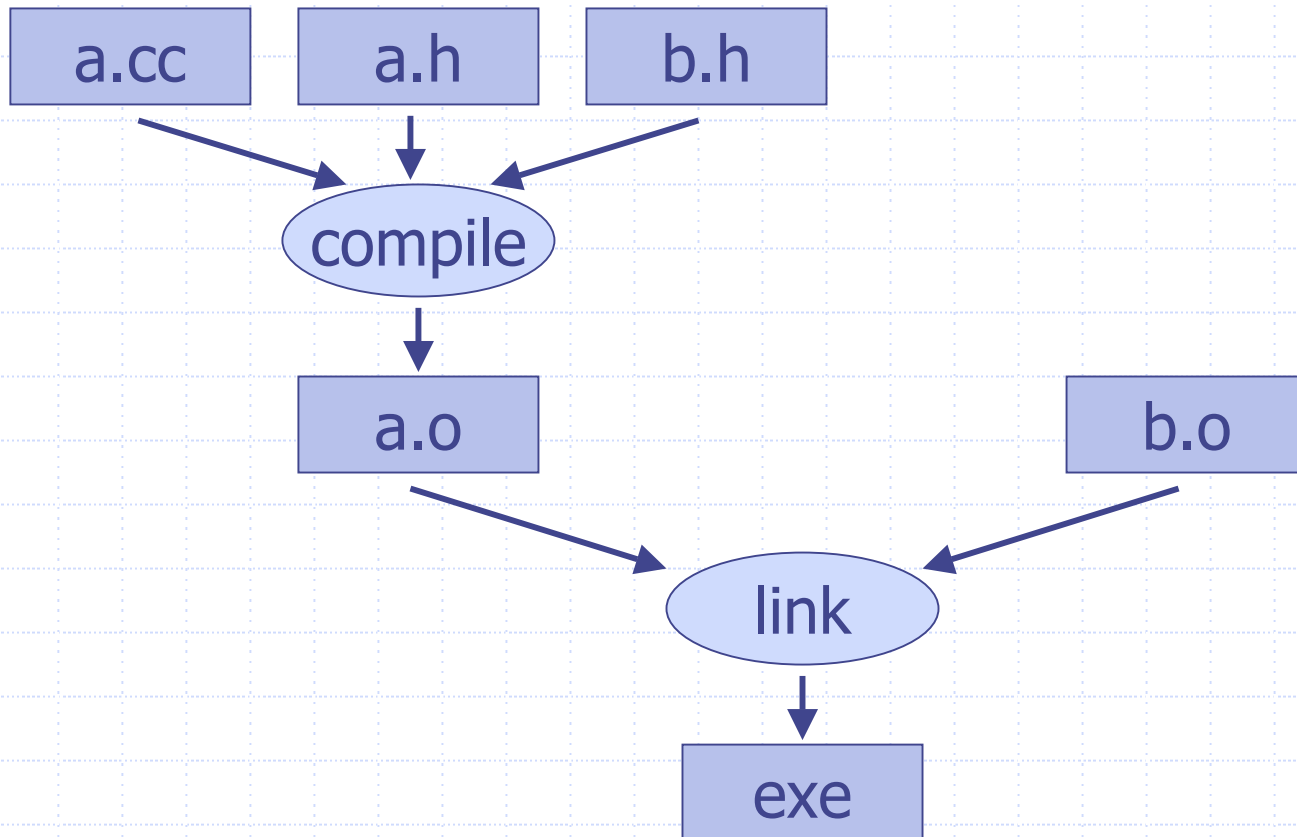
## ◆ Modular Design & Incremental Testing

- A simple wall following example

## ◆ Runtime Configuration

- Reducing the need for recompilation

# Toolflow



# Compiling with g++

## ◆ Use g++ to generate object files

- `g++ -c a.cc`

## ◆ Need to specify where to find include files

- Called include directories

- Working directory is implicitly an include directory

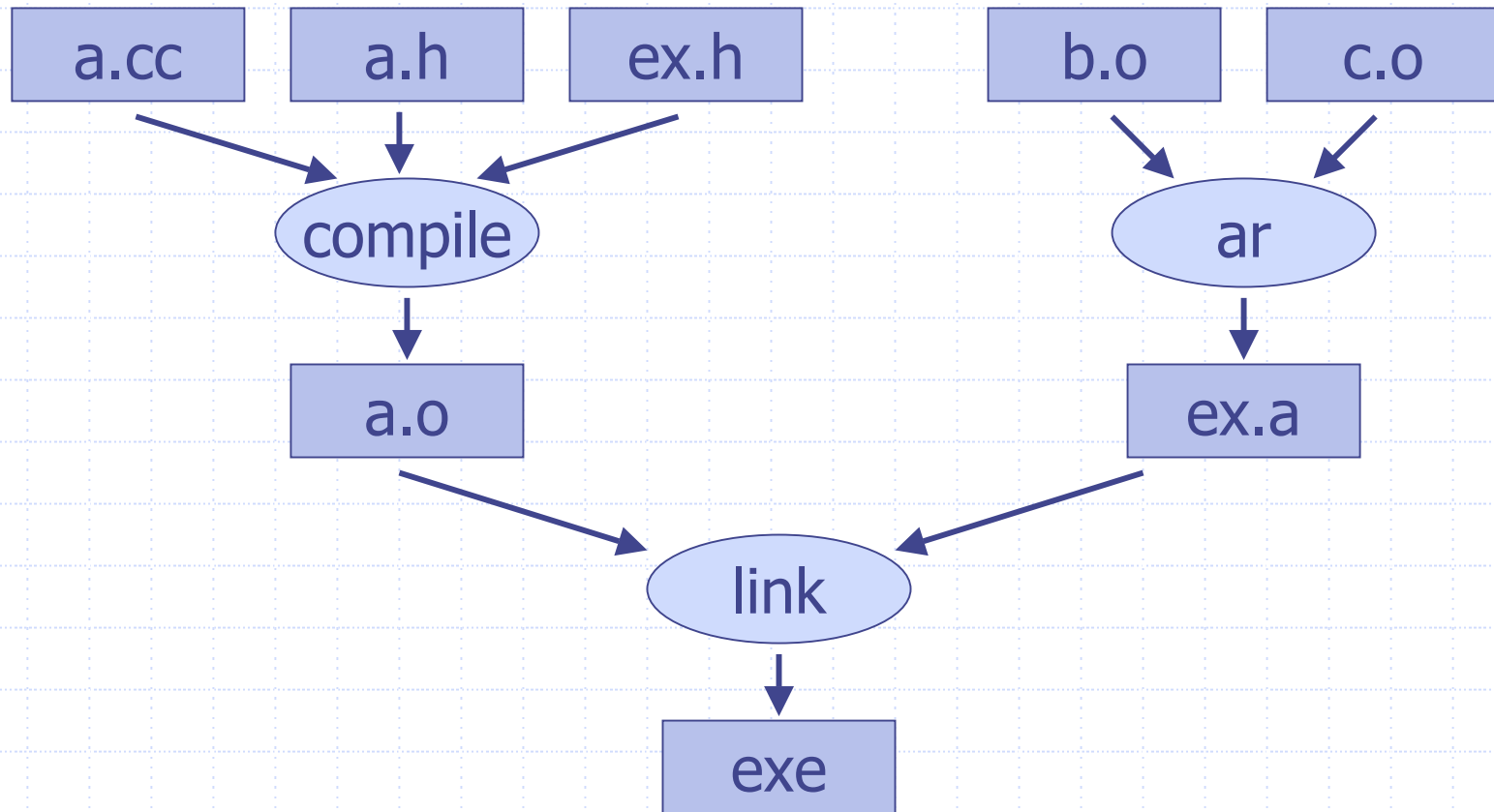
- Use `-I` command line option

- `g++ -I/usr/local/orc/include -c a.cc`

# Linking with g++

- ◆ Use g++ to link object files into an executable
  - `g++ a.o b.o -o test`
- ◆ Link errors indicate that:
  - A function is declared in a header but it is not defined in the corresponding source file
  - An object file has been left off the g++ command line list

# Toolflow with Libraries



# Libraries

- ◆ Libraries group together related object files
- ◆ Maslab will provide three primary libraries
  - Orc library (contains orc api functions)
  - Debug library (contains debug “publish” functions)
  - Vision library (contains vision routines)
- ◆ To use a library:
  - Include the library headers when compiling
  - “link in” the library .a files when linking

# Libraries

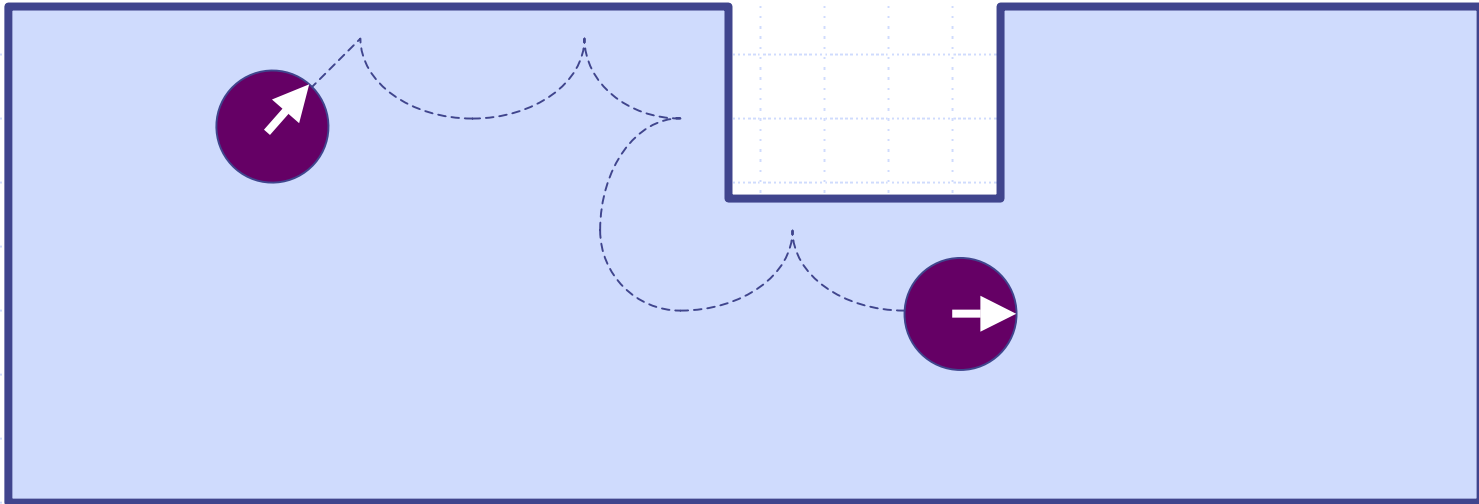
- ◆ Specify library location using `-L` option
- ◆ Specify library name using `-l` option
  - `-lname` refers to the `libname.a` library
- ◆ Example:
  - `g++ a.o -L~/orc/lib -lorc -o test`
- ◆ Wrong order of libraries can cause link errors!

# Other Command Line Options

- ◆ Already mentioned
  - -I, -L, -l, -c, -o
- ◆ Other options
  - -ggdb: compile with debug information
  - -Wall: enables all warnings
  - -ansi: remove non-standard gnu extensions
  - -D,-U: Define or undefine preprocessor name
- ◆ More information at [g++.gnu.org](http://g++.gnu.org)

# Modular Design & Incremental Testing

- ◆ Wall following example
- ◆ Robot needs to:
  - Detect obstacles
  - Rotate
  - Move in an arc



# Monolithic Design

- ◆ One file, one main function, one while loop

```
float irTable[20] = { 1.0, 2.2 , 2.7, ... }
int main() {
    orc_initialize();
    while (true) {
        int sensorVal = orc_analog(0);
        float distance = itTable[ <calcIndex> ];
        if ( distance < 10 ) {
            orc_motorPWM(0,0);
            orc_motorPWM(1,0);
            orc_motorPWM(0,100);
            <sleep>
            orc_motorPWM(0,100);
            orc_motorPWM(1,150);
        }
    }
}
```

# Monolithic Design

- ◆ Eventually keep adding to this until it is completely unmaintainable
- ◆ Difficult to test
  - Is sensor conversion correct? Does rotation work?
- ◆ Difficult to modify
  - Change distance threshold?
- ◆ Difficult to understand
  - What do the PWM commands do?
- ◆ Cut and paste is a really bad form of reuse
  - Reuse rotation code or sensor table lookup code

# Modular Design

- ◆ Multiple functions to do common tasks and abstract away implementation details
  - `readFrontSensor()`
  - `robotTurnLeft(degrees)`
  - `robotStop()`
  - `robotMoveInArc()`
- ◆ Group related functions into the separate files
  - `sensor.h` and `sensor.cc`
  - `robotControl.h` and `robotControl.cc`
- ◆ Incrementally test each component with its own test harness (ie its own test executable)
  - `sensor.t.cc` and `robotControl.t.cc`

# Modular Design

- ◆ Use `#ifndef` guards around headers
  - Prevents including the same header twice
  - Use a unique name

```
#ifndef TEST_H
#define TEST_H
... header code here ...
#endif
```

- ◆ Include corresponding header first in `.cc`
  - So `test.cc` should include `test.h` first
  - Makes sure not accidentally relying on other headers which are not included in the `.h`
- ◆ `.h/.cc` components should be self sufficient (do not rely on externally declared values in other files)

# Runtime Configuration

- ◆ Runtime configuration can drastically reduce debug time by eliminating the need to constantly recompile
- ◆ Command line options
  - `followWalls -right -threshold 20`
- ◆ Configuration file
  - Holding sensor lookup tables

# C/C++ Specifics

See Ed Faulkner's Slides



# Standard Template Library

# STL

- ◆ A collection of **extremely** useful classes
- ◆ Examples
  - `#include <string>`
  - `#include <vector>`
  - `#include <list>`
  - `#include <map>`
- ◆ More information at
  - <http://www.sgi.com/tech/stl>

# STL Examples

## ◆ Vectors

- Use them just like arrays but it takes care of dynamic memory allocation

```
vector vecInt(vecLength);  
for ( int i = 0; i < vecLength; i++ )  
    cout << vecLength[i] << endl;
```

## ◆ Lists

- Linked lists of objects

```
list lst;  
for ( int i = 0; i < vecLength; i++ )  
    lst.push_back( getSensorValue() );
```

# STL Examples

## ◆ Strings

- Useful string stuff without strlen, strcmp

```
string mystring = "hello";
string newstr = mystring + "world";
for ( int i = 0; i < newstr.length(); i++ )
    cout << newstr[i] << endl;
if ( newstr == "hello world" )
    cout << "yay" << endl;
```

# C/C++ Debugging

## ◆ printf/cout method

- Works better if you wrap debug statements in either `#ifdefs` or `if/else` (with a global var)
- Possibly even have multiple debug levels
- The more debugging information the better

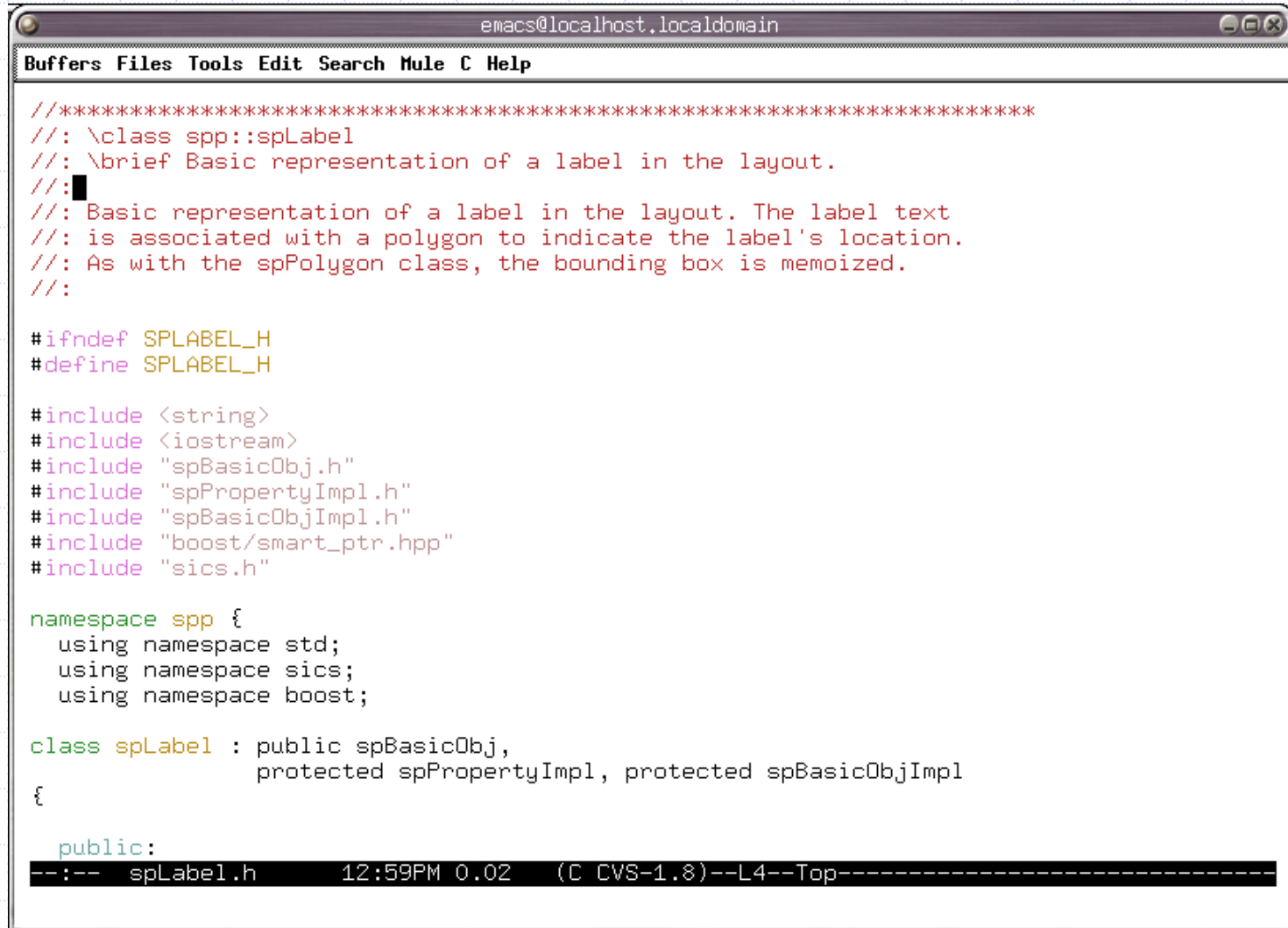
## ◆ Emacs/GDB

- Emacs is an excellent tool for writing code
- Gnu debugger integrates nicely with emacs

# Emacs/GDB

- ◆ Emacs includes color formatting for C/C++
- ◆ Some common commands
  - Type "emacs &" to start emacs in a new window
  - C-x C-f to load in file
  - C-x C-s to save a file
  - C-k cut from the cursor to the end of the line
  - C-y paste last cut at the cursor position

# Emacs/GDB



```
emacs@localhost.localdomain
Buffers Files Tools Edit Search Mule C Help

//*****
//: \class spp::spLabel
//: \brief Basic representation of a label in the layout.
//:
//: Basic representation of a label in the layout. The label text
//: is associated with a polygon to indicate the label's location.
//: As with the spPolygon class, the bounding box is memoized.
//:

#ifndef SPLABEL_H
#define SPLABEL_H

#include <string>
#include <iostream>
#include "spBasicObj.h"
#include "spPropertyImpl.h"
#include "spBasicObjImpl.h"
#include "boost/smart_ptr.hpp"
#include "sics.h"

namespace spp {
    using namespace std;
    using namespace sics;
    using namespace boost;

    class spLabel : public spBasicObj,
                   protected spPropertyImpl, protected spBasicObjImpl
    {

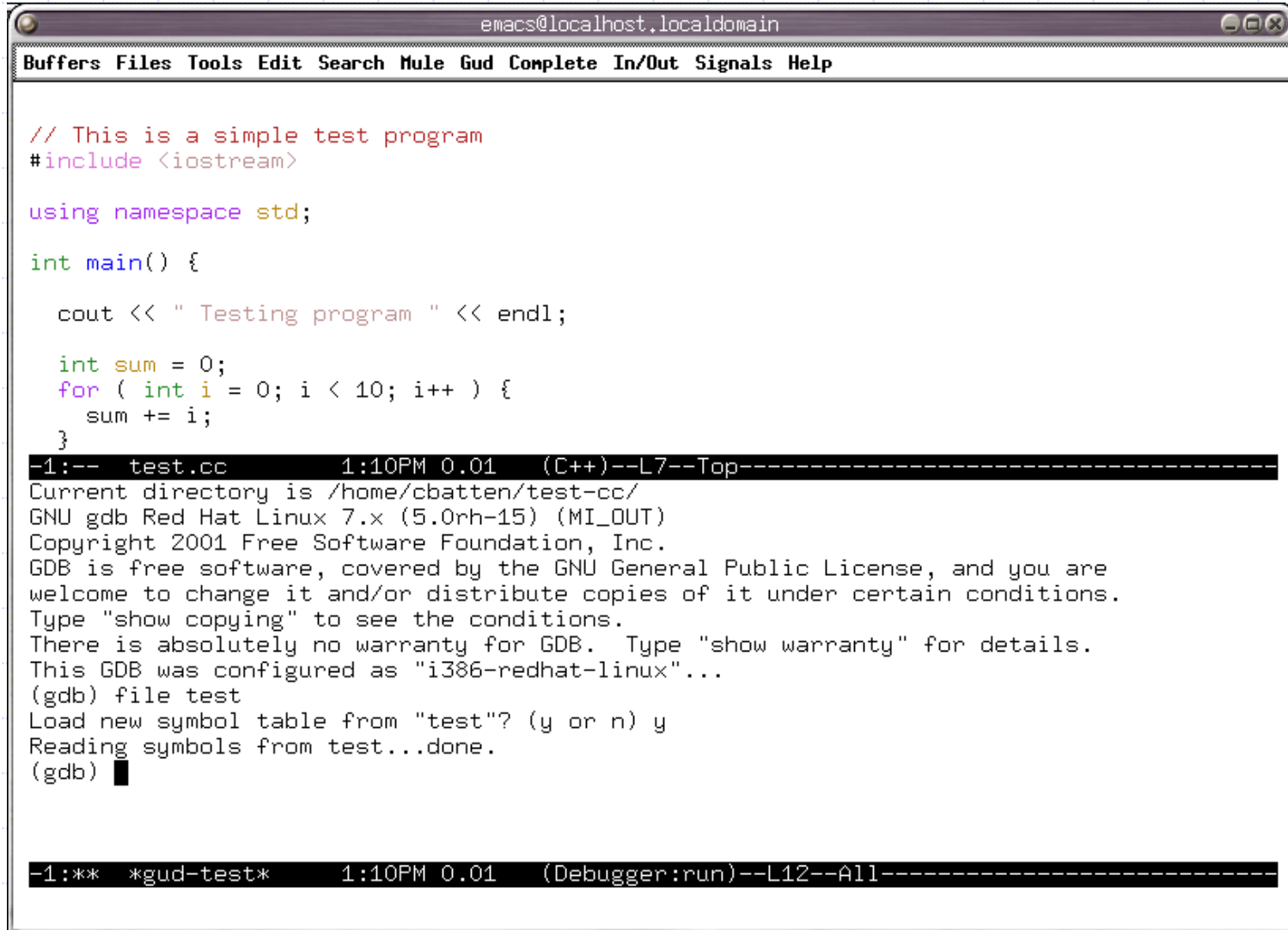
    public:

--:-- spLabel.h 12:59PM 0.02 (C CVS-1.8)--L4--Top-----
```

# Emacs/GDB

- ◆ To use GDB inside of emacs
  - C-x 2 (splits emacs window into two)
  - M-x gdb (M is meta character – alt key)
  - Will ask for gdb command line – type in exec
  - Use “file execname” in gdb window

# Emacs/GDB



The screenshot shows an Emacs window titled 'emacs@localhost.localdomain'. The menu bar includes 'Buffers Files Tools Edit Search Mule Gud Complete In/Out Signals Help'. The main text area contains C++ code for a simple test program. Below the code, a GDB prompt shows the user has loaded the file 'test.cc'. The GDB output includes the current directory, GNU gdb version, copyright information, and a confirmation to load the symbol table from 'test'.

```
emacs@localhost.localdomain
Buffers Files Tools Edit Search Mule Gud Complete In/Out Signals Help

// This is a simple test program
#include <iostream>

using namespace std;

int main() {

    cout << " Testing program " << endl;

    int sum = 0;
    for ( int i = 0; i < 10; i++ ) {
        sum += i;
    }
}

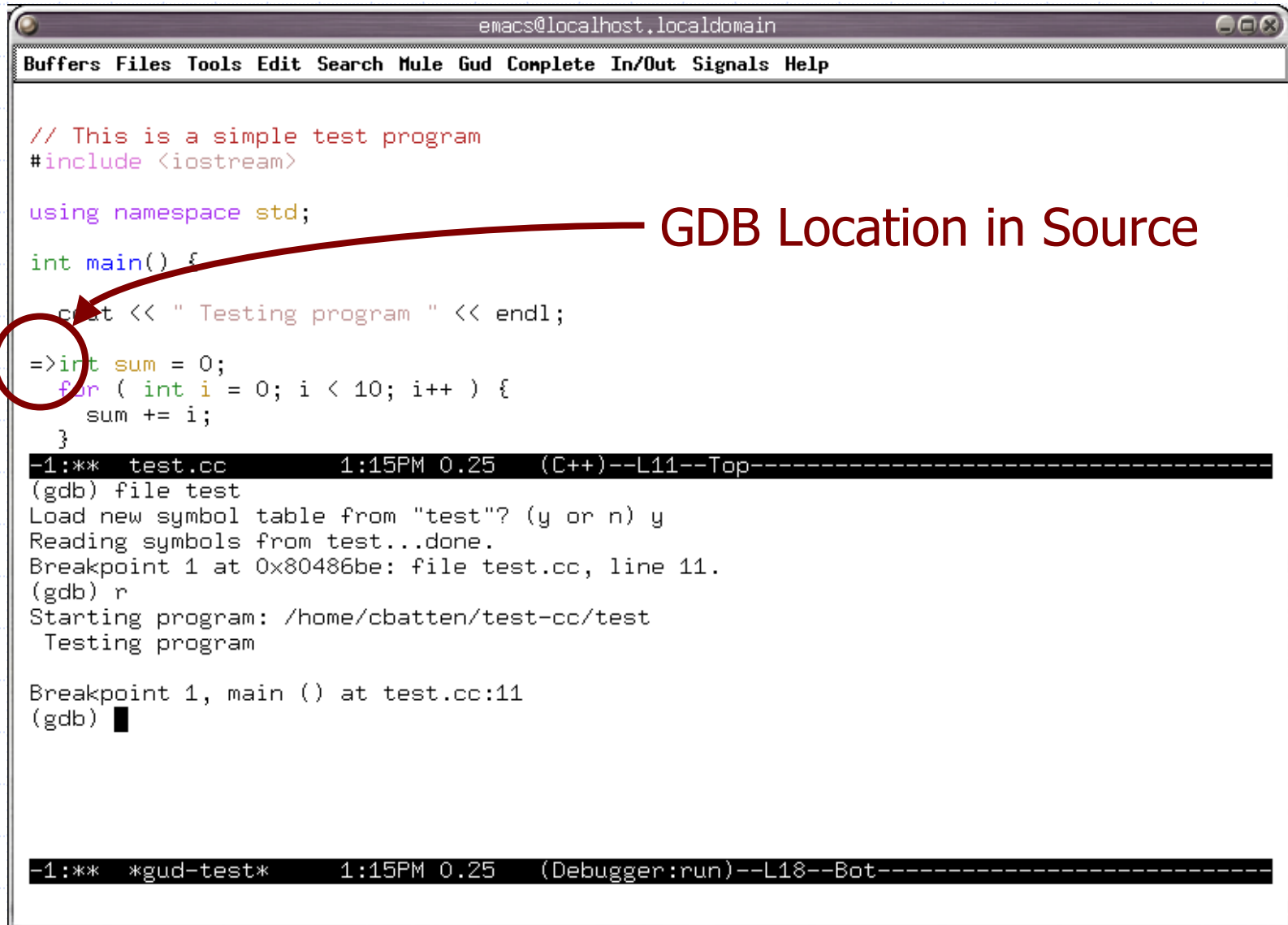
-1:-- test.cc      1:10PM 0.01  (C++)--L7--Top-----
Current directory is /home/cbatten/test-cc/
GNU gdb Red Hat Linux 7.x (5.0rh-15) (MI_OUT)
Copyright 2001 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(gdb) file test
Load new symbol table from "test"? (y or n) y
Reading symbols from test...done.
(gdb) █

-1:** *gud-test*   1:10PM 0.01  (Debugger:run)--L12--All-----
```

# Emacs/GDB

- ◆ Use C-x space in source win to set breakpoint
- ◆ GDB Common Commands
  - r run until breakpoint
  - s step to next line
  - n step to next line (and into function calls)
  - bt back trace (very useful for segfaults)
- ◆ With threads, gdb will stop all threads when it reaches a breakpoint in any one thread

# Emacs/GDB



```
emacs@localhost.localdomain
Buffers Files Tools Edit Search Mule Gud Complete In/Out Signals Help

// This is a simple test program
#include <iostream>

using namespace std;

int main() {
    cout << " Testing program " << endl;
    =>int sum = 0;
    for ( int i = 0; i < 10; i++ ) {
        sum += i;
    }
}

-1:** test.cc 1:15PM 0.25 (C++)--L11--Top-----
(gdb) file test
Load new symbol table from "test"? (y or n) y
Reading symbols from test...done.
Breakpoint 1 at 0x80486be: file test.cc, line 11.
(gdb) r
Starting program: /home/cbatten/test-cc/test
Testing program

Breakpoint 1, main () at test.cc:11
(gdb) █

-1:** *gud-test* 1:15PM 0.25 (Debugger:run)--L18--Bot-----
```

GDB Location in Source

# Debug Client

See Ed Faulkner's Slides

# Other Tools

## ◆ Make

- Controls the generation of object files and executables from source files

## ◆ Concurrent Versions System (CVS)

- Allows multiple people to efficiently work on the same source code and keep track of their changes

# GNU Make

◆ Typing “make” runs the file named Makefile in the current directory

◆ The Makefile is a list of rules of the form

```
target ... : dependencies ...  
<tab>command  
<tab>command ...
```

- **Target** – a file to be generated or an “action label”
- **Dependencies** – Files or targets on which this target depends
- **Command** – A command for regenerating the target

◆ Essentially, make will determine whether or not it needs to regenerate each target by seeing if one or more of the target’s dependencies have changed since the last build

# Make

- ◆ Can use variables
  - Specify with `varname=value` on its own line
  - Use with `$(varname)`
- ◆ Text manipulation functions
  - `$(patsubst pattern,replacement,text)`
  - `$(patsubst %.o,%.cc,<list of objfiles>)`

# Make – An Example

```
CC=g++                                # Compiler to use
FLAGS=-g                               # Compile flags
LIB_DIR=../liborc                      # orc-related libraries
INC_DIR=../liborc                     # path to header files
LIBS=-lm -lpthread -lorc             # Library files

HELLOWORLD_SRCS=helloworld.cc exfile1.cc exfile2.cc
HELLOWORLD_HDRS=exfile1.h exfile2.h
HELLOWORLD_OBJS=$(patsubst %.cc,%.o,$(HELLOWORLD_SRCS))

all : helloworld

$(HELLOWORLD_OBJS) : %.o : %.cc $(HELLOWORLD_HDRS)
    $(CC) $(FLAGS) -c $*.cc -o $@

helloworld: $(HELLOWORLD_OBJS) $(HELLOWORLD_HDRS)
    $(CC) -o helloworld $(HELLOWORLD_OBJS) $(LIBS)

clean:
    rm -f *.o helloworld
```

# Make – A Backup Target

## ◆ Backup to nfs mounted team directory

```
datestr=`date +%Y-%m-%d_%H-%M`
```

```
backup :
```

```
    tar -czf test-$(datestr).tgz .
```

```
    cp test-$(datestr).tgz /nfsmnt/teamdir
```

## ◆ Backup to athena account

```
datestr=`date +%Y-%m-%d_%H-%M`
```

```
backup :
```

```
    tar -czf test-$(datestr).tgz .
```

```
    scp test-$(datestr).tgz athena.dialup.mit.edu:~/.
```

```
    rm test-$(datestr).tgz
```

# Make

- ◆ Make is very powerful
  - Many built-in text transformation functions
  - Implicit rules based on file extension
  - Control commands
- ◆ More information
  - <http://www.gnu.org/manual/make>

# CVS

## ◆ Concurrent Versions System

- Creates a **central repository** where all source code is stored as well as information on who changed what and when
- Users **checkout** a copy of the source code, edit it, and then **commit** their modified version
- Users can see what has changed to help track down bugs and allows multiple users to work on the same source code at the same time

## ◆ CVS commands are executed using

- `cvs <commandname>`
- `"cvs help"` will print out valid commands

# CVS – Setting up

## ◆ First set CVSROOT environment variable

- `CVSROOT=~ /cvsroot; export CVSROOT`

## ◆ Now run "cvs init"

- Will create the cvs root directory and other files needed by cvs

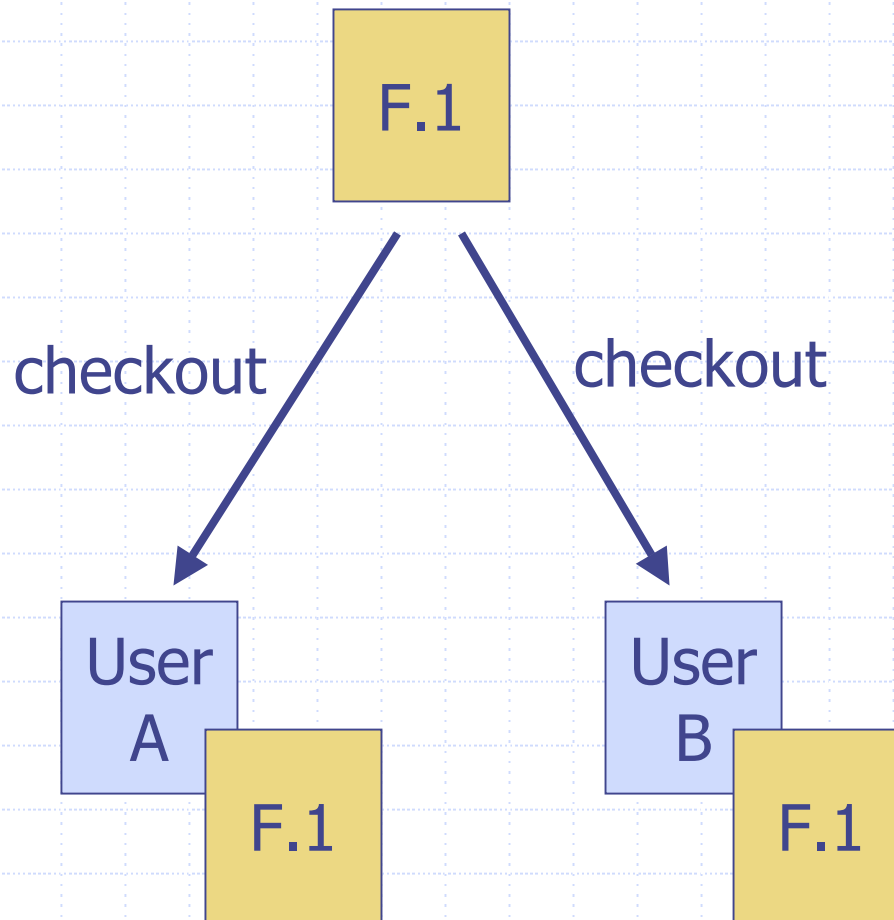
## ◆ To put your project under cvs

- Usually useful to code a bit without cvs
- `cd` into project directory
- `cvs import -m "Msg" pname head start`

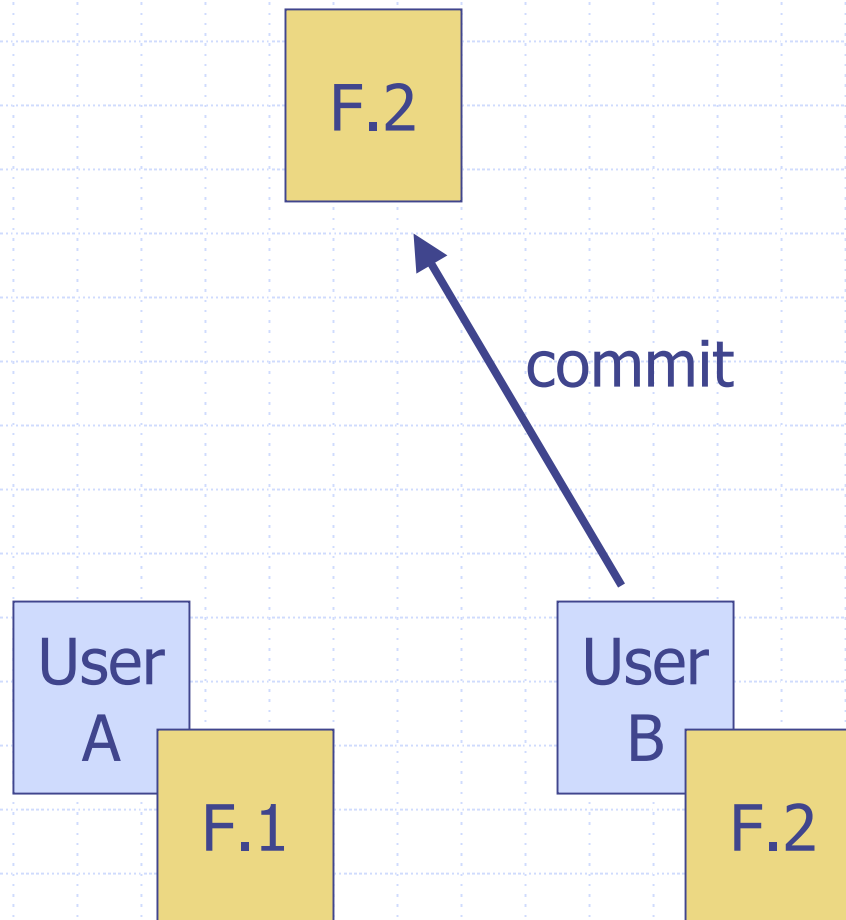
# CVS - Basics

- ◆ Common commands
  - cvs checkout pname
  - cvs update pname [filelist]
  - cvs commit [-m "log message"] [filelist]
  - cvs add [-m "log message"] [filelist]
  - cvs diff
- ◆ Set the \$CVSEEDITOR environment variable to change which editor is used for log messages

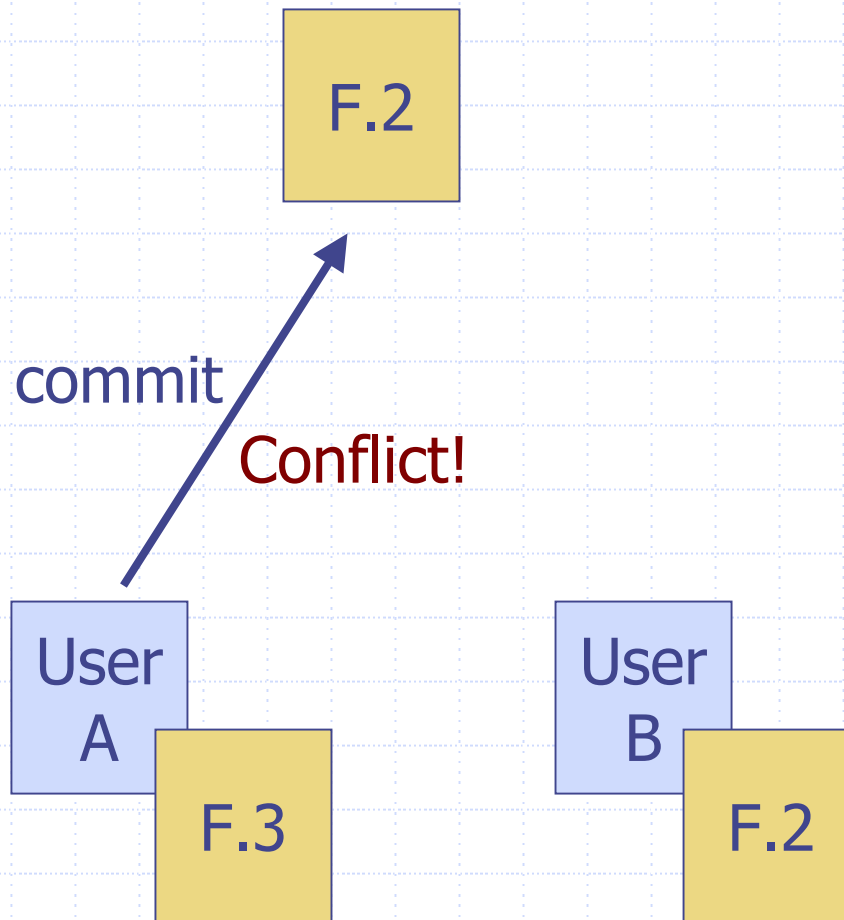
# CVS – Multiple Users



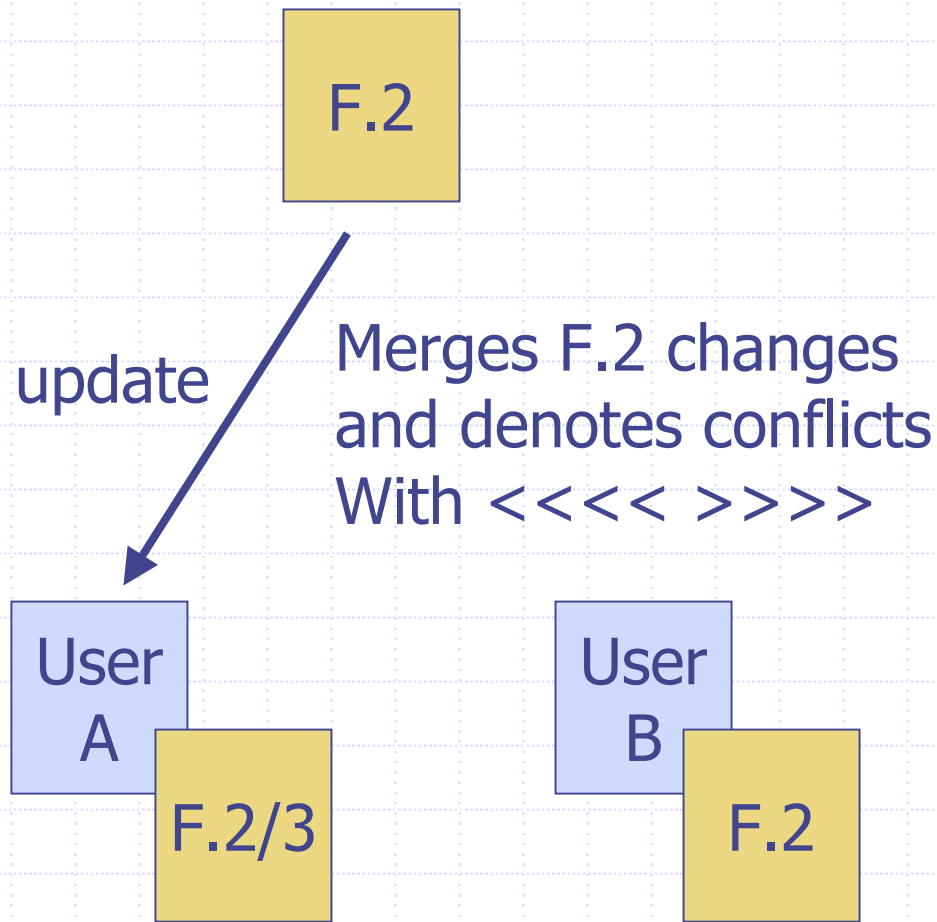
# CVS – Multiple Users



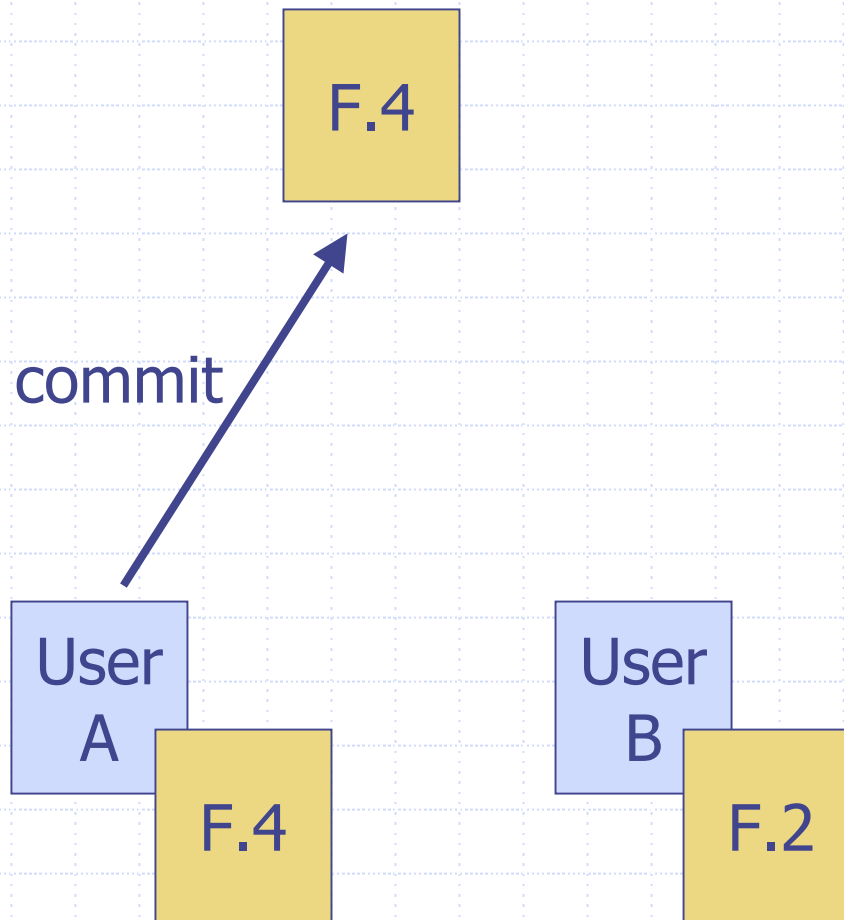
# CVS – Multiple Users



# CVS – Multiple Users



# CVS – Multiple Users



# CVS

- ◆ Conflicts are rare because developers are working on different parts of the project
- ◆ Rule of thumb: always update before commit
- ◆ Informative log messages can be very helpful tracking down bugs

# CVS for Backup

- ◆ Keep cvs repository in nfs mounted team dir
- ◆ On athena need to access cvs through ssh
  - See <http://www.jfipa.org/publications/CVSGuide/>
  - Basically just set these two env variables
    - ◆ Export CVSROOT=":ext:uname@athena...:/cvsrootdir"
    - ◆ Export CVS\_RSH="ssh"
  - Will need to entire password for all cvs commands