CMSC131
Lecture Set 1:
Introduction to Tools

Topics in this set:

1. Tools needed for this course
2. How to get Started

Announcements

1) WaitList -- size has been raised to the capacity of the room (continue to attend and if people drop you will be moved in)

2) Canvas setup – should have access to syllabus and slides etc. Things will be added there on a regular basis.

1) Piazza – invitations sent
    1) for communication
        1) to all individual
        2) to all instructors
        3) to all students
    2) better than email
        1) faster response time
        2) others see the answer so don't have to ask as much

Programming Languages

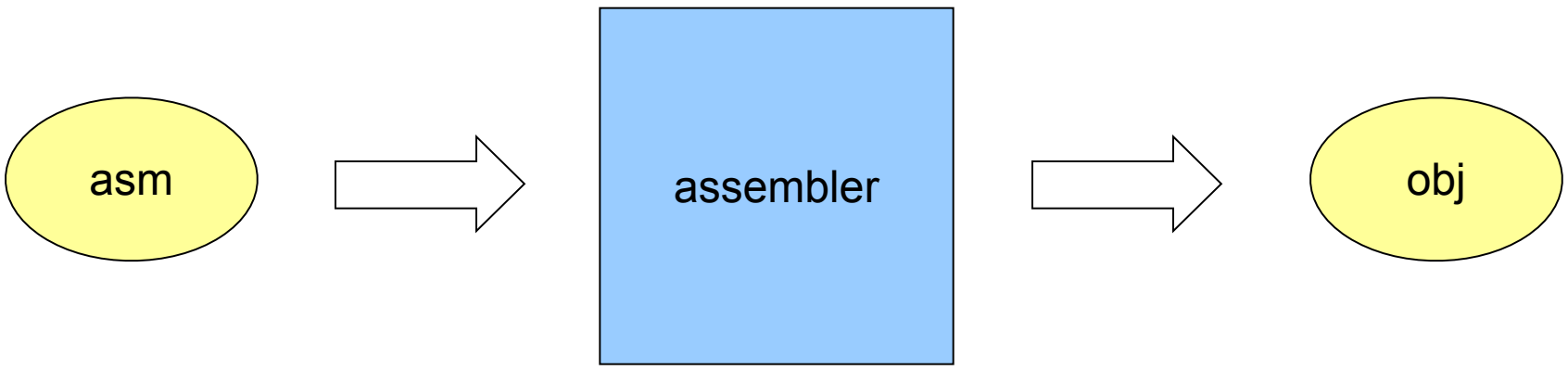
- Used to write programs that run on computers
- Generations of programming languages
  - 1st (1GL): machine code
  - 2nd (2GL): assembly code
  - 3rd (3GL): procedural languages
  - 4th (4GL): application-specific languages
  - 5th (5GL): constraint languages

1st Generation: Machine Code

- Recall: computer data is 0's and 1's.
- In machine code, so are programs!
  - Program: sequence of instructions
  - Machine code: instructions consist of 0's and 1's
- Next slide: example machine code instruction from MIPS (= "Microprocessor without interlocked pipeline stages") architecture
  - Popular in mid-, late 90s
  - Instructions are 4 bytes long

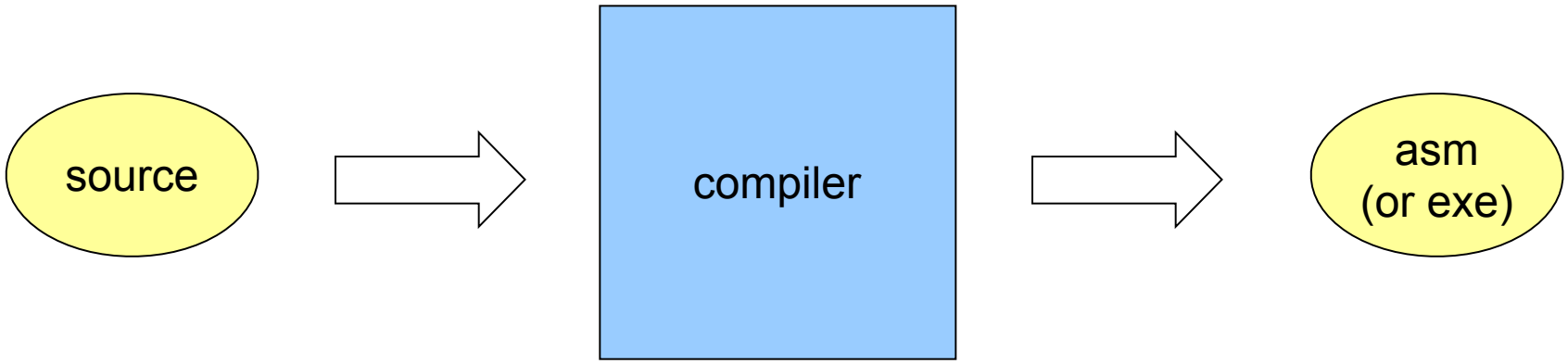
2nd Generation: Assembly

- Problem with 1GLs: Who can remember those opcodes, addresses, etc. as 0's, 1's?
- Solution (1950s): *assembly language*
  - *mnemonics* = descriptive character strings for opcodes
  - Let programmers give descriptive names to addresses
- Needs an **Assembler** to translate



3rd Generation: Procedural Languages

- Problems with 2GLs
  - *Platform dependency*
    - Different kinds (*architectures*) of computers use different instruction formats
    - E.g. x86, Pentium, 68K, MIPS, SPARC, etc.
    - 1GL / 2GL programs written for one kind of machine will not work on another
  - *Low level*: programs difficult to understand



Interpreters

- Another way to execute 3GL programs
  - Interpreters take source code as input
  - Interpreters execute source directly
  - Much slower than compiled programs

- *Debuggers* are based on interpreters
  - Debuggers support step-by-step execution of source code
  - Internal behavior of program can be closely inspected

Object Oriented Terminology

- Original Procedural Languages
  - have procedures that can be reused ("verb" centric)
- Object Oriented Languages
  - centered on the objects ("noun" centric)
- object
  - principal entities that are manipulated by the program (nouns)
- class
  - a "blueprint" that defines the structure for one or more objects
- method
  - java term for a "function", a "procedure" or a "subroutine"
  - this is the code that does something (verbs)
- main method
  - a special method that defines where program execution begins
- *statements*
  - individual instructions

Tools for Writing Programs

- The old days
  - Text editor: used to create files of source code
  - Compiler: generate executables from source
  - Debugger: trace programs to locate errors

- Today: IDE = "integrated development environment"
  - Text editor / compiler / debugger rolled in one
  - Examples: Eclipse, Visual Studio, NetBeans, etc.

Basics of Eclipse

- http://www.cs.umd.edu/eclipse/EclipseTutorial/
- Eclipse is used to:
  - Create
  - Edit
  - Compile
  - Run
  - Debug

programs (for this class, Java programs).

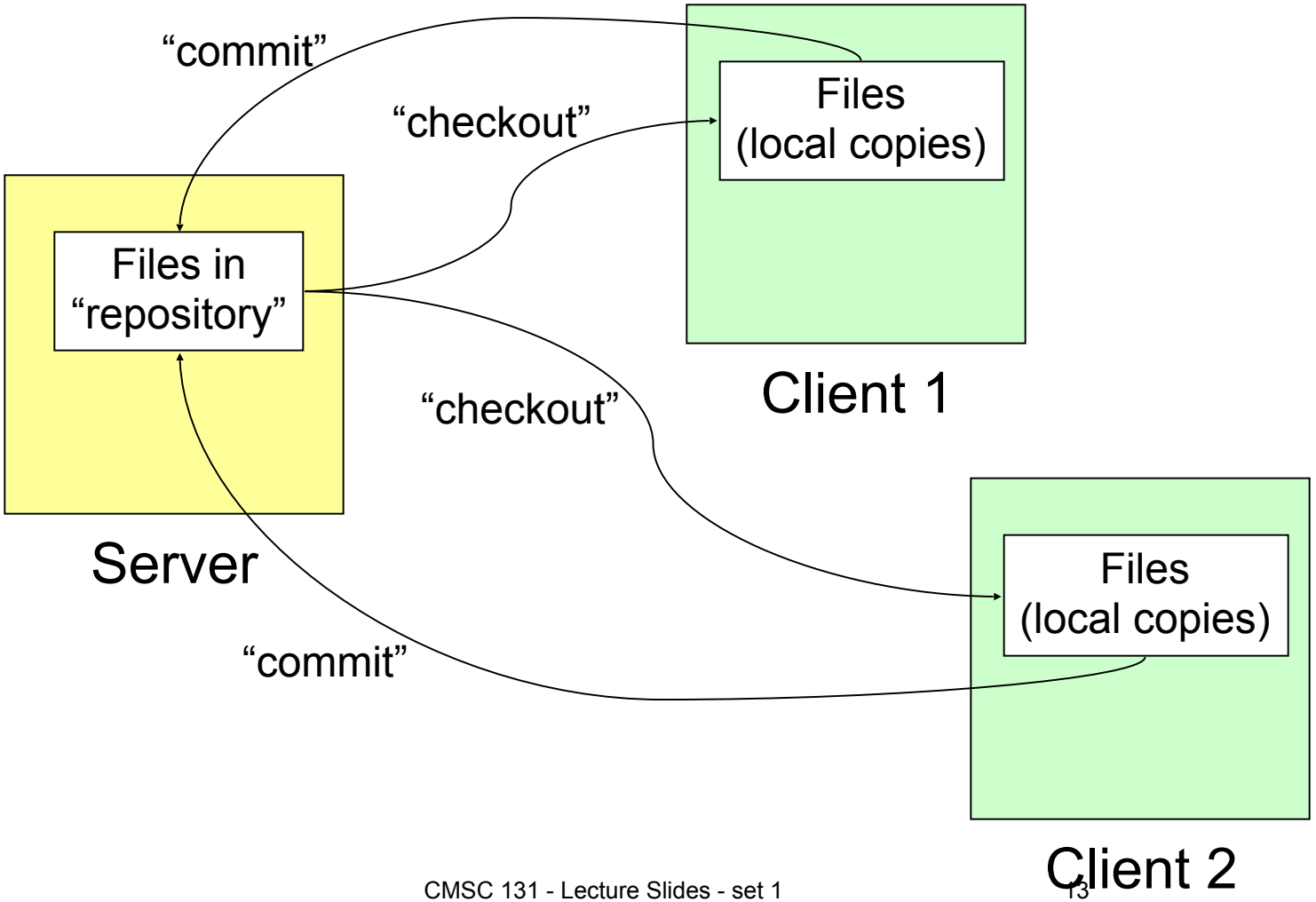
Basics of Eclipse-speak

- *Project*: collection of related source files
  - To create a program in Eclipse:
    - Create a new project
    - Create files in the project
- *Perspective*: framework for viewing and/or manipulating programs
  - Important perspectives in this class:
    - *Java*: for creating, running programs
    - *Debug:* for tracing, removing errors in programs
    - *CVS repository*: for interacting with assignment-submission system
- *Workspace: Where your files are stored locally*
- *Buffer: Window where editing takes place*

Class Projects with CVS

- You will use Eclipse for Java programming in this course
- How will you:
  - obtain (check-out) files that are supplied to you
  - save (commit) the files for later work
  - turn in (submit) when you are finished
- *CVS (= Concurrent Versions System)*
  - Tool for project-file management
  - Maintains versions, etc.
  - Allows different sites to work on same project

CVS Worldview



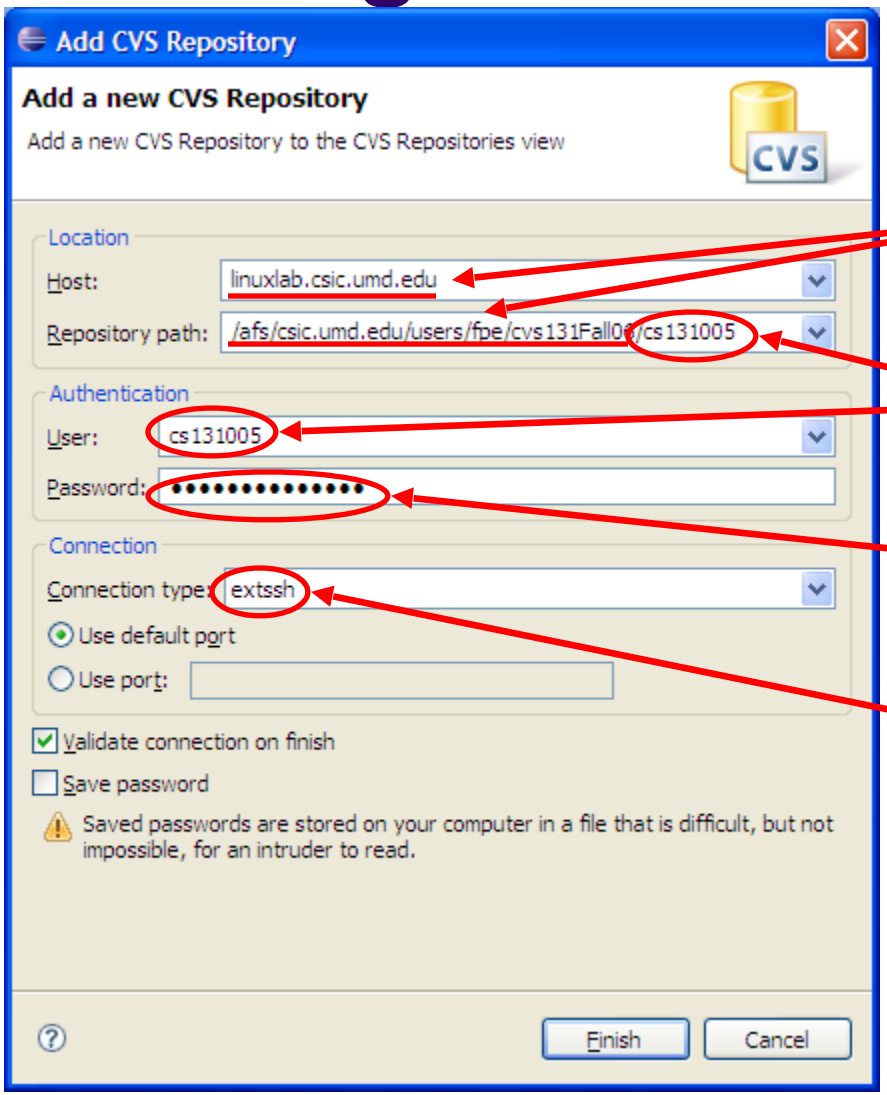
CVS in More Detail

- CVS server maintains current versions of files in project (= “repository”)
- To access files from another machine (“client”), repository files must be “checked out”
- Changes to files on client may be “committed” to server, with changed files becoming new version
- (Once a repository is checked out by a client, subsequent versions may be accessed via “update”)

How CMSC Project Submission Works

- Repository created for each student linuxlab account
- You check out repository to start work on project
- When you “save” changes in Eclipse, “commit” automatically invoked by plug-ins
- You “submit” when finished using Eclipse (UMD plug-in handles relevant CVS commands)

EXAMPLE – only an example
Adding a CVS Repository



Common to everyone but details change on instructor & semester

Your linuxlab username

Your linuxlab password

Don't forget to set this!

Working on Project

- You do not have this project showing in the Java perspective.
- You go to the CVS perspective and check it out.
- When you switch back to “Java” perspective, your project is now there!
- Make sure you are in the Java perspective to edit
- When you save in “Java” perspective, changes are automatically committed to CVS repository.

Submitting the Project

- Edit the file
- Make sure it runs correctly
- Submit the project for grading
- Go to submit.cs.umd.edu to see test results
  - Public tests
  - Private (Secret) tests
  - Release tests
    - give limited feedback (first two failed tests give more)
    - costs you “tokens” – usually 3 to start with
    - spent tokens regenerate in 24 hours