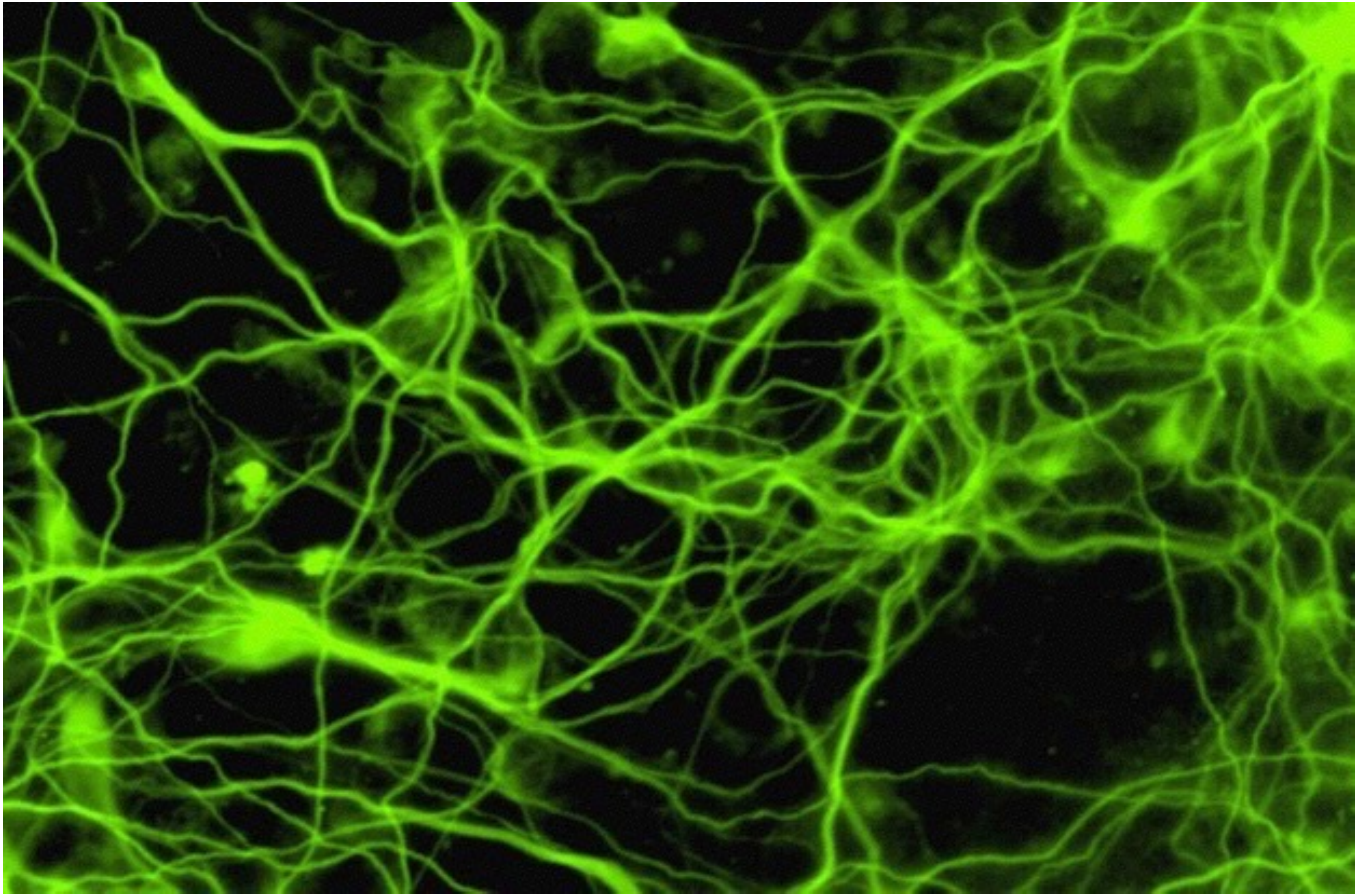
Computer Science for the Physical Sciences

Week 4

Craig Rasmussen (Research Support Services, University of Oregon)

Which one is most like your brain?

Spiking neurons and dendritic trees

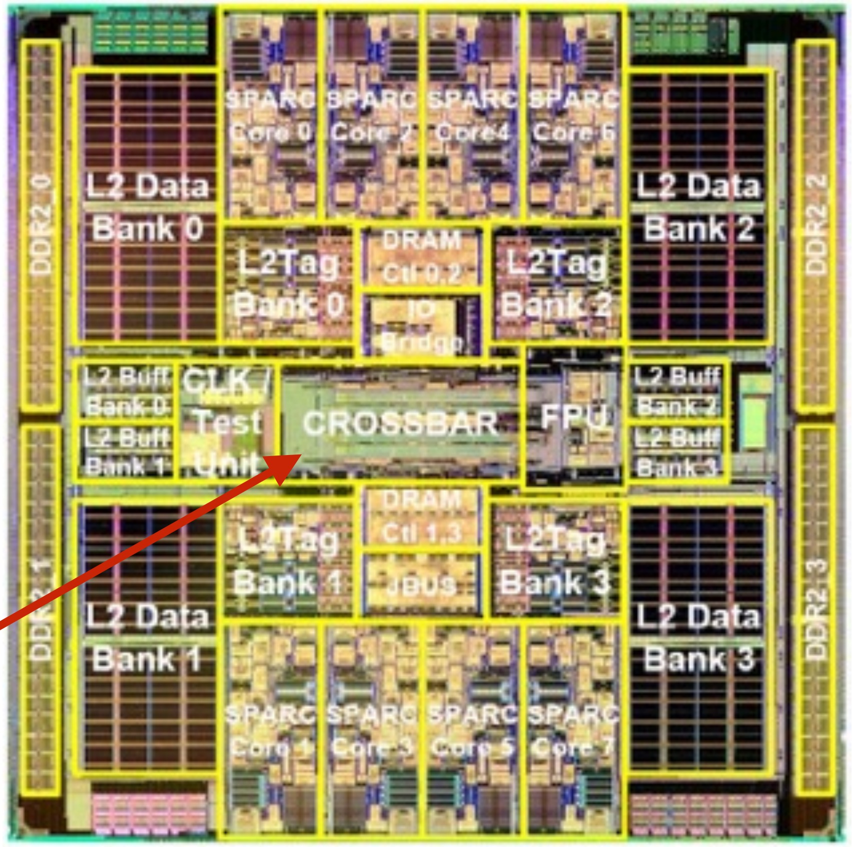


Are computers fast enough to compete?

- 10^9 neurons x 10^3 Hz x 100 operations
  - ~10^{14} op/s
- Roadrunner computer at Los Alamos
  - 16x180 = 2880 hybrid nodes with 4 IBM cell cores
  - each cell core has 8 SPU cores
    - 2880 x 4 x 8 = 92K compute cores
  - each SPU ~200 x 10^9 flops (single precision)
  - 92K x 200 x 10^9 > 10^{15} flops

Are computers big enough to compete?

- Visual cortex
  - 10^9 neurons
  - 10^{13} neurons synaptic connections
- Los Alamos' Roadrunner
  - 92K cores
  - 10K transistors per core
  - 10^9 transistors
  - where are the equivalent connections (memory pathways)?



Do computers have enough memory to compete?

- Assume memory is in the synaptic connections
  - 10^9 neurons x 10^4 connections x 1 byte
  - ~10^{13} bytes
- Roadrunner
  - >10^{13} bytes

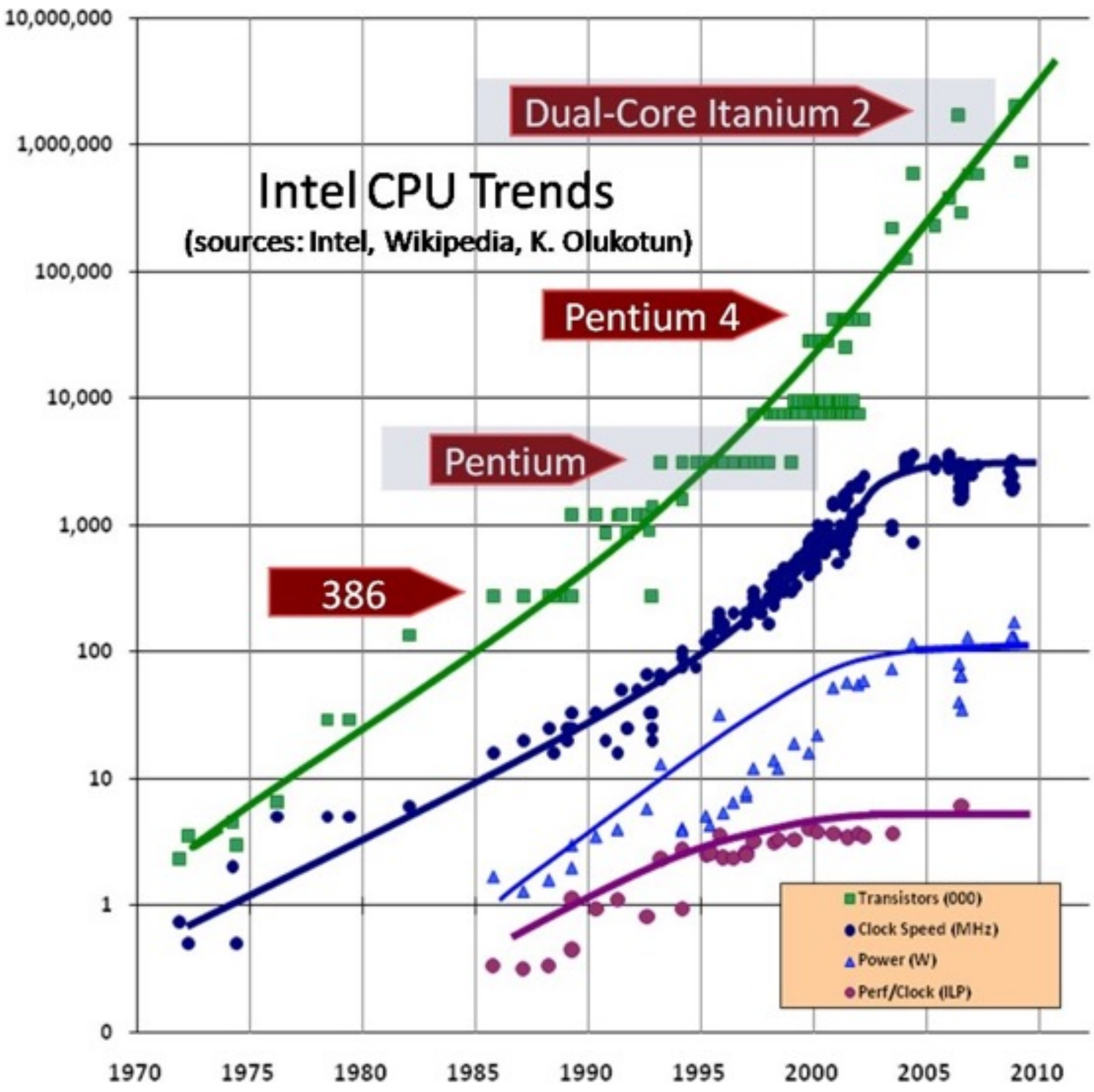
Do computers have enough bandwidth to compete?

- Assume the bandwidth is in the synaptic connections
  - 10^{13} connections x 1000 Hz
  - 10^{16} bytes/sec
- Roadrunner
  - 11520 Cell processors x 20 Gbytes/sec
  - 10^{14} bytes/sec

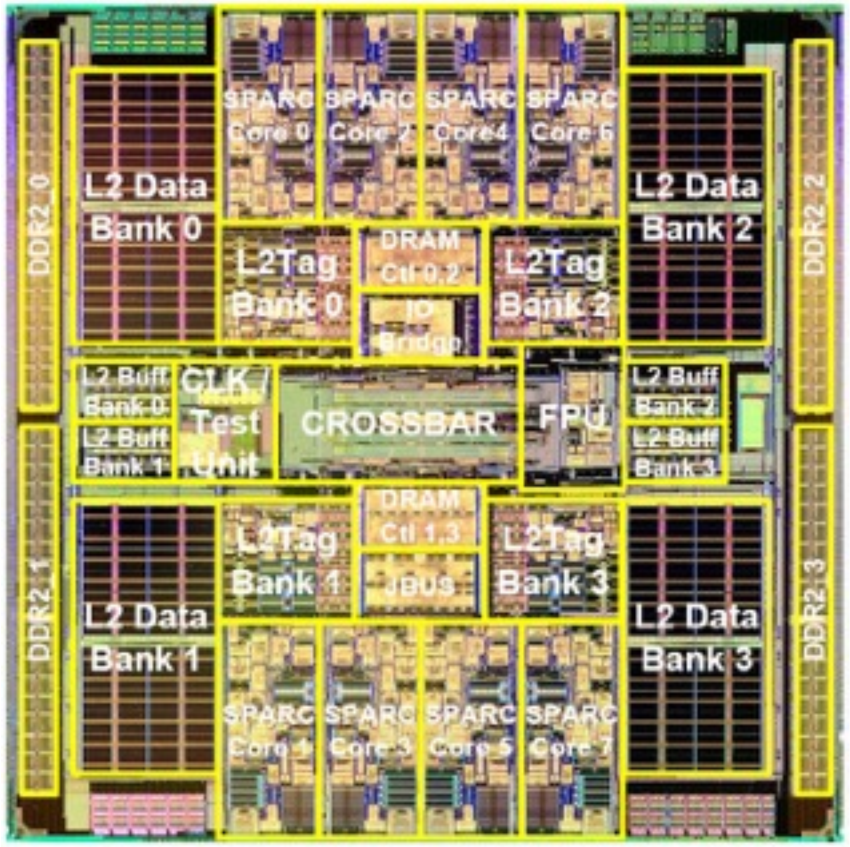
So why can’t thinking machines think?

- Big enough
- Fast enough
- Have enough memory
- But lack bandwidth
  - using memory to simulate synaptic connections
  - must time share access to memory
- So what, we’ll just run 100-1000 times slower
- And Voila, we’ll have a brain!
- But we don’t know the circuit (but can be learned with STDP)

But LANL Roadrunner required 2.35 MW.
So why doesn't your just brain melt?



Ivan Sutherland: “The tyranny of the clock” Should time be quantized?



- A chip is big
- A chip is clocked at 2-3 GHz
- Takes several clock cycles to traverse the chip
- Therefore modern chips have many clocks (10K?)
- A transistor does something each clock cycle
    - mostly nothing but waste energy
- Neurons only fire when necessary (mostly true)
- Ivan Sutherland examining computer circuits that are asynchronous
    - only active when necessary

Computer Science Minor: *Last week*

- Required courses (24 credits)
  - Introduction to Computer Science I-II-III
  - Elements of Discrete Mathematics I-II
  - Introduction to Data Structures Lists and Maps
- Upper-division courses (8 credits)
  - Computer Architecture
  - Introduction to Algorithms Complexity
  - C/C++ and Unix Python and Shell
  - Operating Systems
  - Automata Theory
  - Software Methodology I-II
    Revision Control and Make Files
  - Introduction to Compilers
  - Computational Science
  - Bioinformatics
  - Data Mining
  - Introduction to Artificial Intelligence
  - Machine Learning

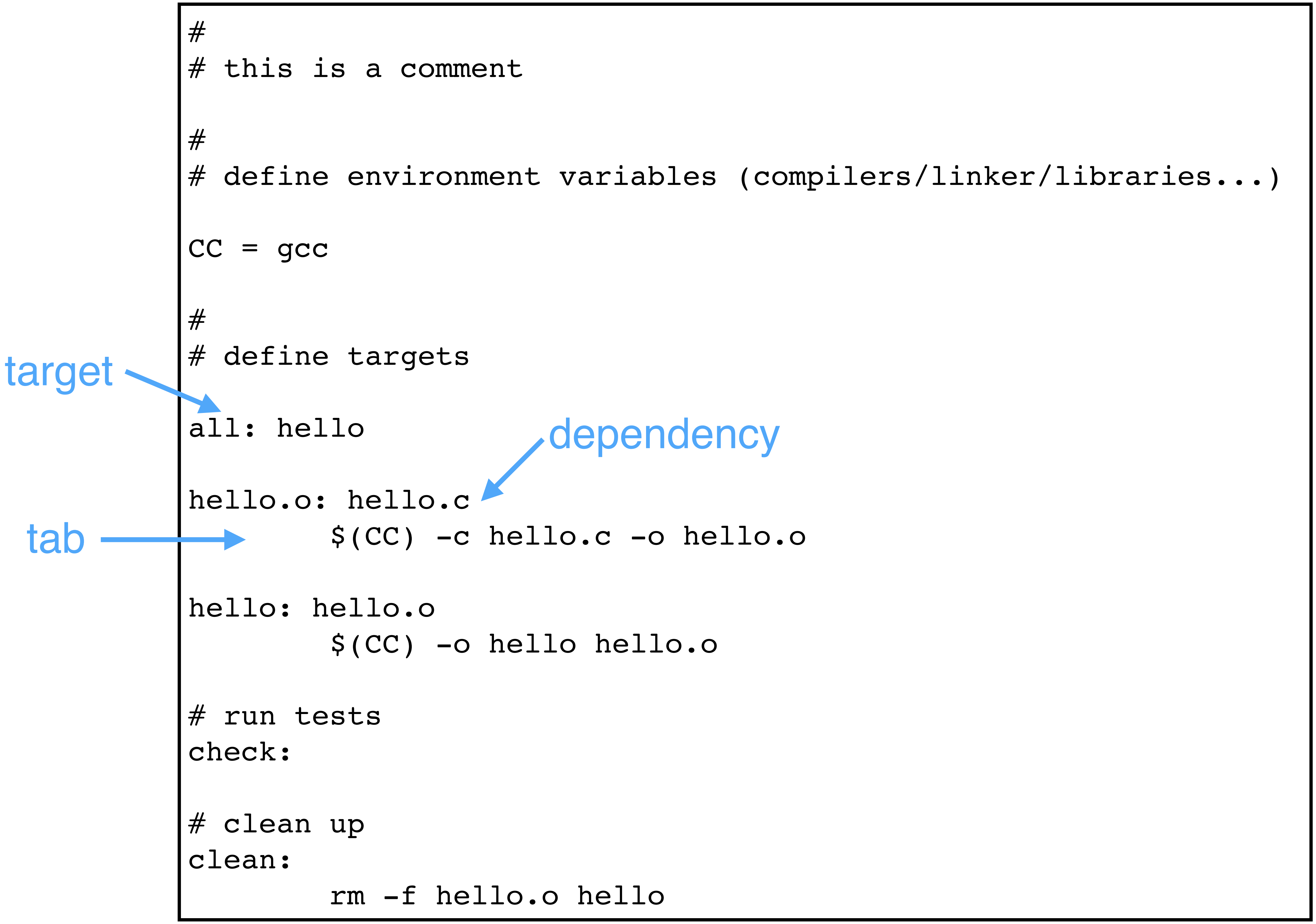
Computer Science Minor: *This week*

- Required courses (24 credits)
  - Introduction to Computer Science I-II-III
  - Elements of Discrete Mathematics I-II
  - Introduction to Data Structures
- Upper-division courses (8 credits)
  - Computer Architecture
  - Introduction to Algorithms
  - C/C++ and Unix Functions and pipes
  - Operating Systems
  - Automata Theory
  - Software Methodology I-II
    Revision Control and Make Files
  - Introduction to Compilers
  - Computational Science
  - Bioinformatics
  - Data Mining
  - Introduction to Artificial Intelligence
  - Machine Learning

Computer Science Minor: *Make files*

- Required courses (24 credits)
  - Introduction to Computer Science I-II-III
  - Elements of Discrete Mathematics I-II
  - Introduction to Data Structures
- Upper-division courses (8 credits)
  - Computer Architecture
  - Introduction to Algorithms
  - C/C++ and Unix Functions and pipes
  - Operating Systems
  - Automata Theory
  - Software Methodology I-II
    Revision Control and Make Files
  - Introduction to Compilers
  - Computational Science
  - Bioinformatics
  - Data Mining
  - Introduction to Artificial Intelligence
  - Machine Learning

A *makefile* maintains groups of programs based on dependencies being satisfied



```
#
# this is a comment

#
# define environment variables (compilers/linker/libraries...)

CC = gcc

#
# define targets

all: hello

hello.o: hello.c
        $(CC) -c hello.c -o hello.o

hello: hello.o
        $(CC) -o hello hello.o

# run tests
check:

# clean up
clean:
        rm -f hello.o hello
```

Computer Science Minor: Functions, classes, and Unix pipes

- Required courses (24 credits)
  - Introduction to Computer Science I-II-III
  - Elements of Discrete Mathematics I-II
  - Introduction to Data Structures

- Upper-division courses (8 credits)
  - Computer Architecture
  - Introduction to Algorithms
  - C/C++ and Unix Functions and pipes
  - Operating Systems
  - Automata Theory
  - Software Methodology I-II
    Revision Control and Make Files
  - Introduction to Compilers
  - Computational Science
  - Bioinformatics
  - Data Mining
  - Introduction to Artificial Intelligence
  - Machine Learning

Classes

- A class encapsulates functions and state variables
- Class Foo
  - int x, y, z;   // state variables
  - void f1();   // function
- A class is a template (recipe) for creating objects
- A program can have pointers to many live objects at once
- Each object contains state
- In parallel programming state is evil!
  - who modified x and when?

Functions

- A function takes input and produces output
- Functions are composable
  - *f3(f2(f1(x)))*
- What happens to the state variables?
  - *f2()* consumes the output of *f1()*
- Going stateless is good!

Unix pipes

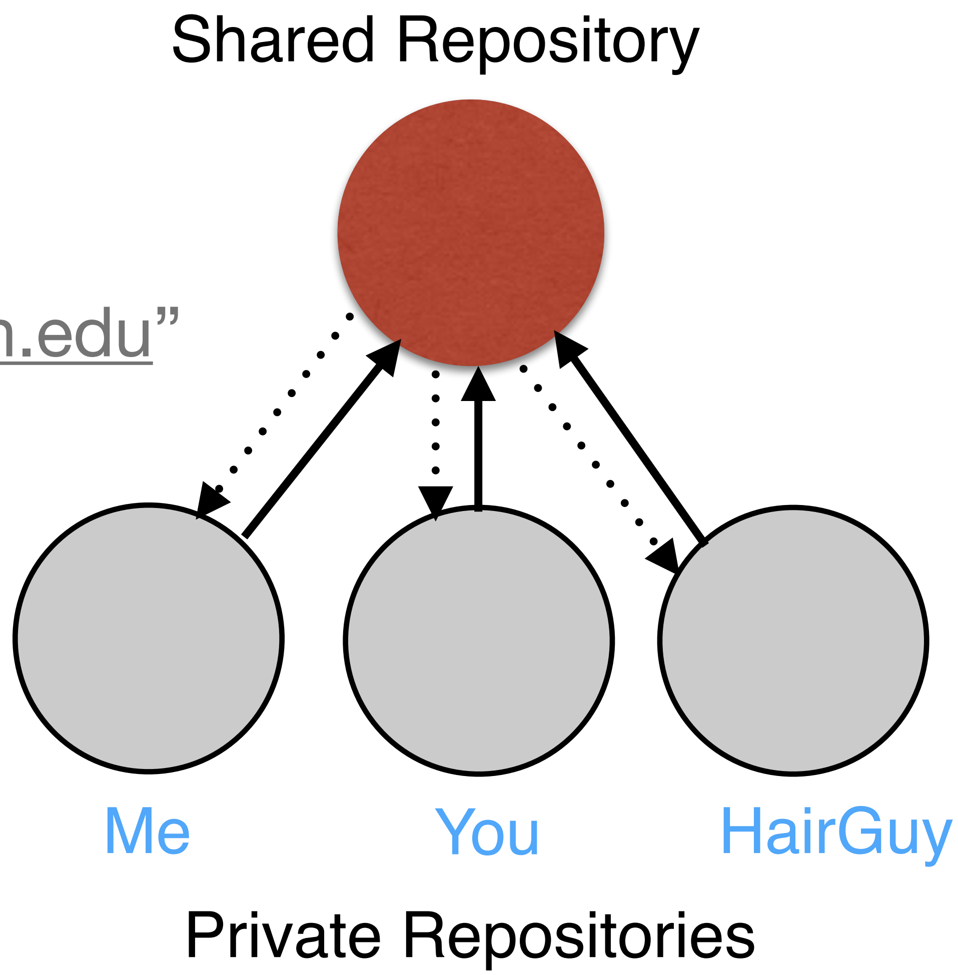
- A unix shell program takes input and produces output
  - standard input (file)
  - standard output (file)
- Unix shell programs are composable with pipes
  - program1 | program2 | program3
  - the output of program1 is said to be “piped” to the input of program2
- What happens to the state variables (files)?
  - program2 consumes the output of program1

Computer Science Minor: Revision control

- Required courses (24 credits)
  - Introduction to Computer Science I-II-III
  - Elements of Discrete Mathematics I-II
  - Introduction to Data Structures
- Upper-division courses (8 credits)
  - Computer Architecture
  - Introduction to Algorithms
  - C/C++ and Unix Functions and pipes
  - Operating Systems
  - Automata Theory
  - Software Methodology I-II
    Revision Control and Make Files
  - Introduction to Compilers
  - Computational Science
  - Bioinformatics
  - Data Mining
  - Introduction to Artificial Intelligence
  - Machine Learning

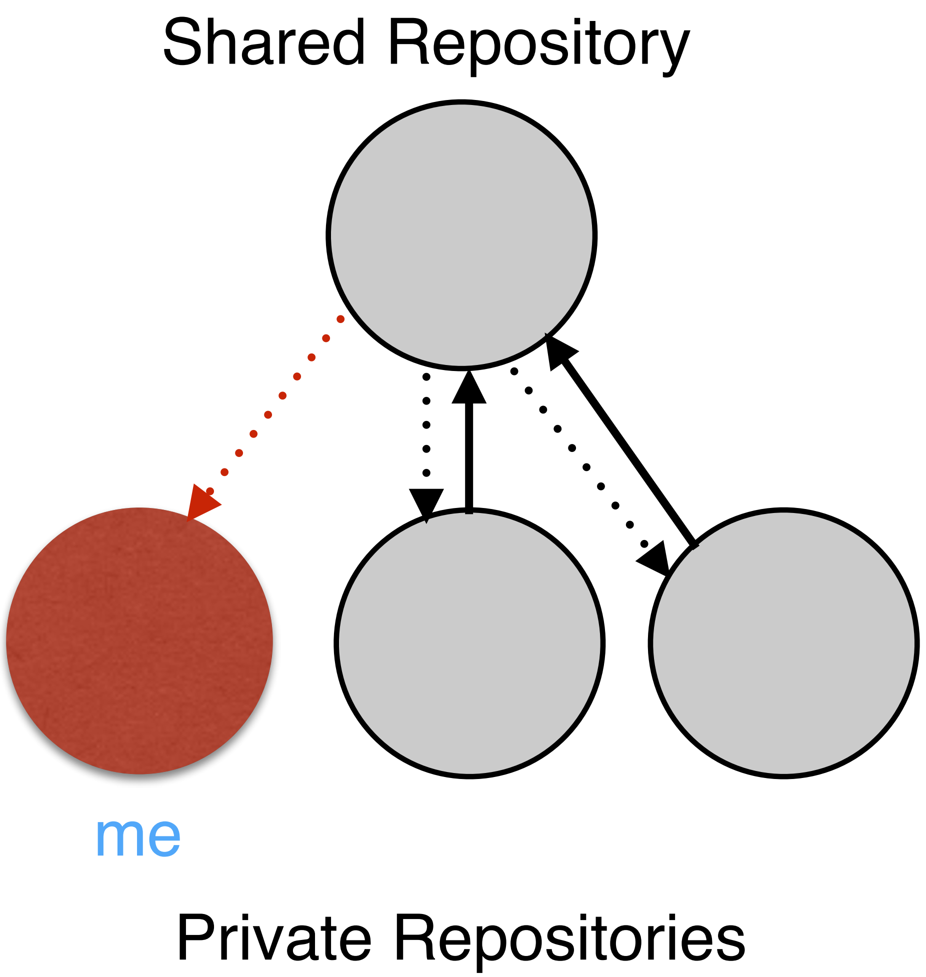
Git is now the standard version control system

- Configuring Git
  - git config —global user.name “Your Name”
  - git config —global user.email “user@uoregon.edu”
  - git config —global core.editor emacs
- Creating a new shared repository
  - mkdir repo
  - cd repo
  - git init —bare



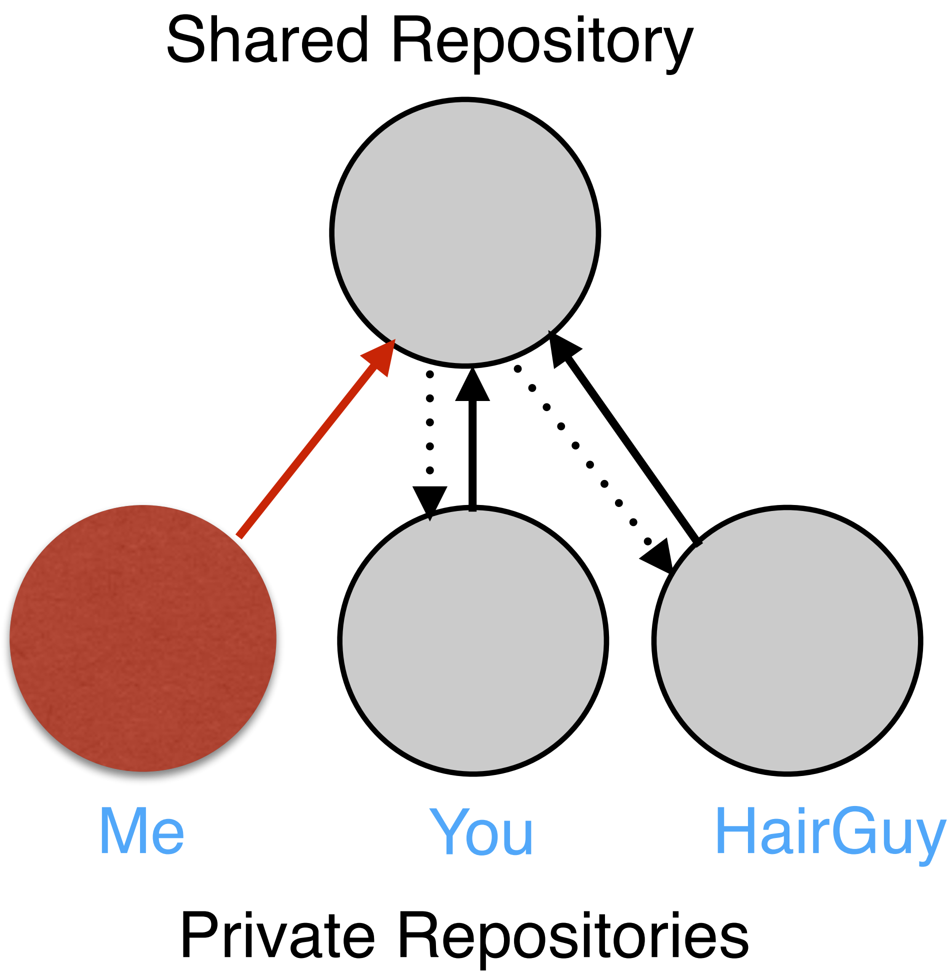
Creating a local copy of an existing repository and adding files

- Clone a repository
  - git clone /usr/local/repos/repo me
  - git clone URL
- Add a file
  - cd me
  - touch README
  - git add README
  - git commit -m”Initial version.” README



Sharing changes to a file

- Edit the file then compare changes
  - emacs README
  - git diff README
- Discovery
  - git status
- Commit changes to local repository
  - git commit -m”a message” README
- Push changes to shared repository
  - git push



Shared Repository

Me

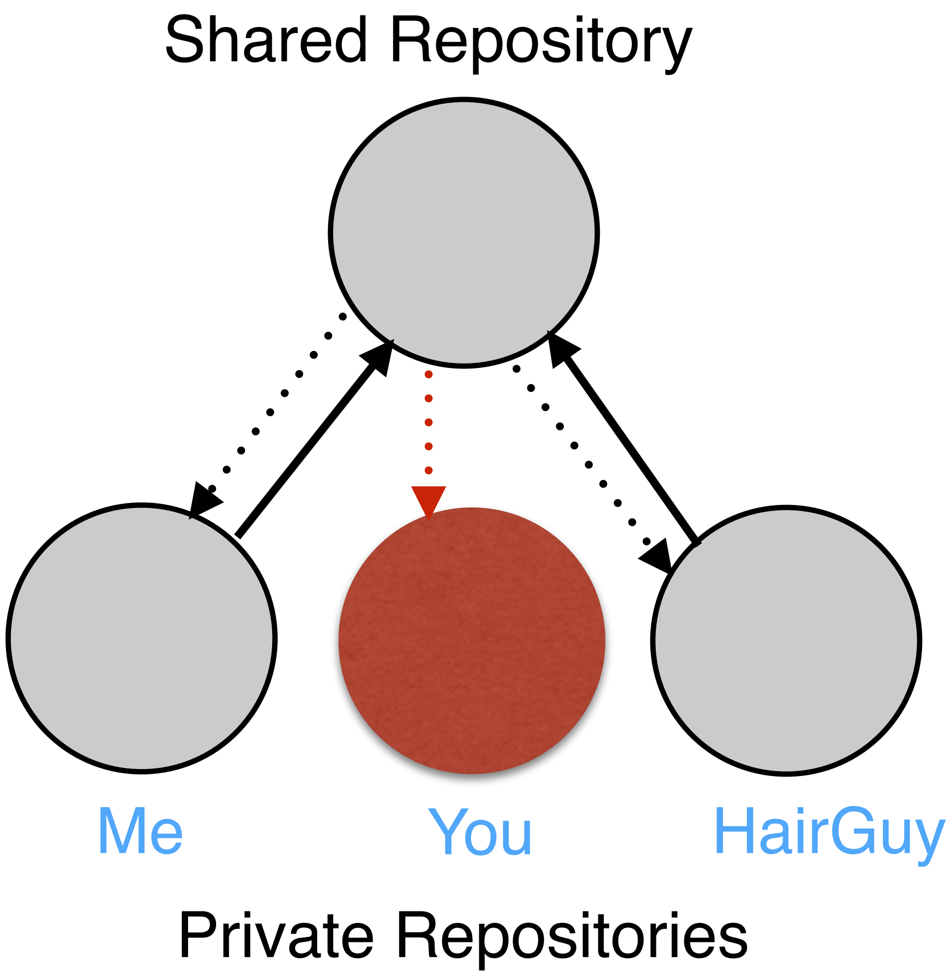
You

HairGuy

Private Repositories

Retrieving changes that a teammate has made

- Fetch the latest from the shared repository
  - git fetch
- Merge with local private repository
  - git merge origin/master
- If you have merge conflicts you must fix them
  - emacs README
  - git add README
  - git commit -m”Liked my changes better”


Shared Repository
Me
You
HairGuy
Private Repositories