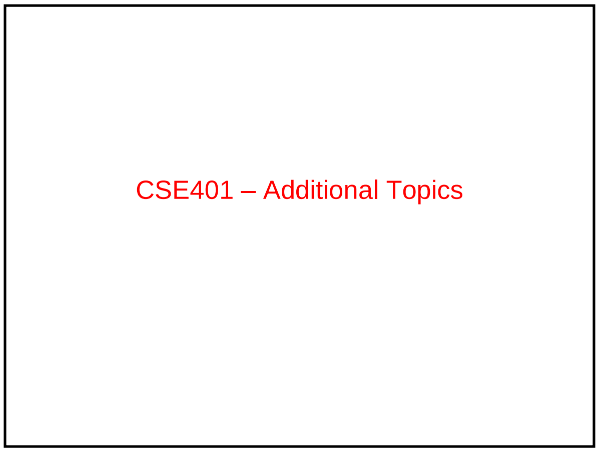
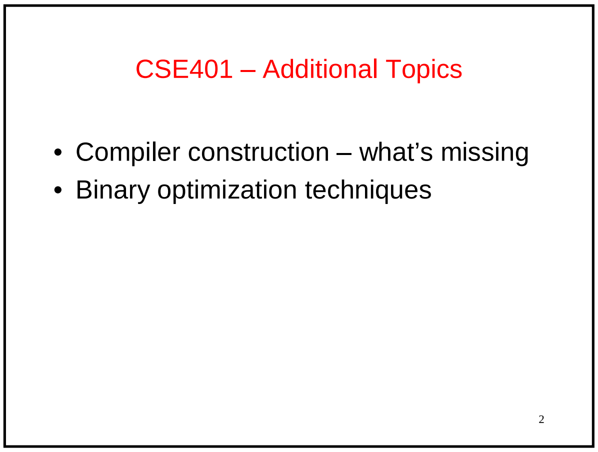
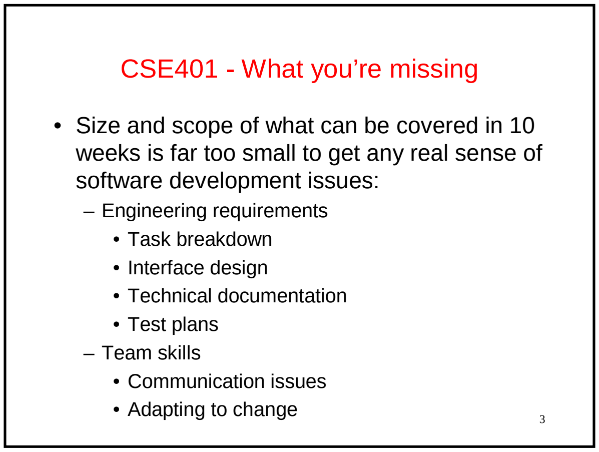


CSE401 – Additional Topics



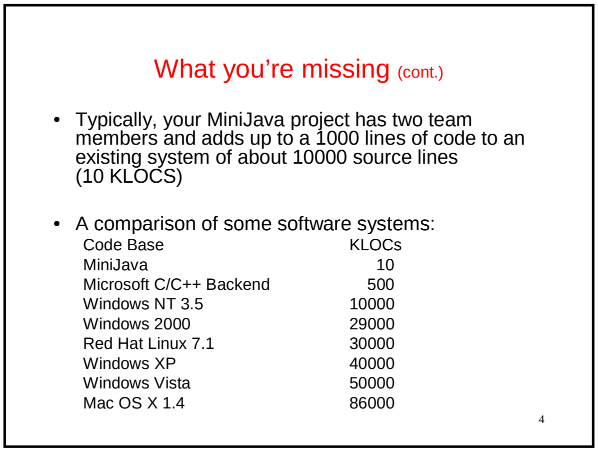
CSE401 – Additional Topics

- Compiler construction – what's missing
- Binary optimization techniques



CSE401 - What you're missing

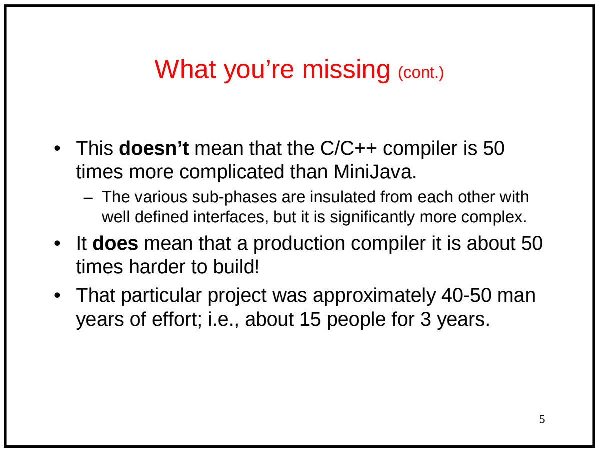
- Size and scope of what can be covered in 10 weeks is far too small to get any real sense of software development issues:
  - Engineering requirements
    - Task breakdown
    - Interface design
    - Technical documentation
    - Test plans
  - Team skills
    - Communication issues
    - Adapting to change



What you're missing (cont.)

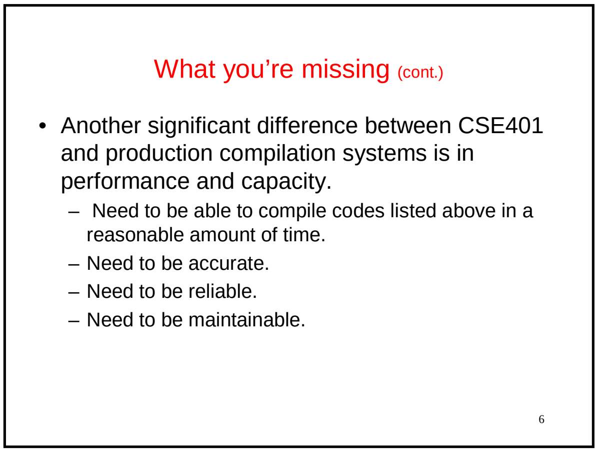
- Typically, your MiniJava project has two team members and adds up to a 1000 lines of code to an existing system of about 10000 source lines (10 KLOCS)
- A comparison of some software systems:

Code Base	KLOCs
MiniJava	10
Microsoft C/C++ Backend	500
Windows NT 3.5	10000
Windows 2000	29000
Red Hat Linux 7.1	30000
Windows XP	40000
Windows Vista	50000
Mac OS X 1.4	86000



What you're missing (cont.)

- This **doesn't** mean that the C/C++ compiler is 50 times more complicated than MiniJava.
  - The various sub-phases are insulated from each other with well defined interfaces, but it is significantly more complex.
- It **does** mean that a production compiler it is about 50 times harder to build!
- That particular project was approximately 40-50 man years of effort; i.e., about 15 people for 3 years.



What you're missing (cont.)

- Another significant difference between CSE401 and production compilation systems is in performance and capacity.
  - Need to be able to compile codes listed above in a reasonable amount of time.
  - Need to be accurate.
  - Need to be reliable.
  - Need to be maintainable.

Binary Optimization

These systems are also known as Post-Link Optimizers.

The basic idea:

- Read in a compiled binary
- Decompile to IR
- Perform a series of optimizations
- Rewrite the binary file back out

Why?

- Modern computer performance is dominated by cost to read/write memory.
- Often, one of the largest users of memory bandwidth is the program itself.
- Analyzing the program at the binary level actually simplifies the process of understanding program control flow, instruction cache use and working-set requirements.

Binary Optimization Process

- Build base version of target binary.
- Instrument base version to add profile data collection code.
- Run instrumented version over selected test cases capturing profile data. This sometimes referred to as ‘training’ runs.
- Run binary optimizer using base version of executable and profile data as input. This will produce an optimized binary.
- Test and ship optimized binary.

Binary Optimizations

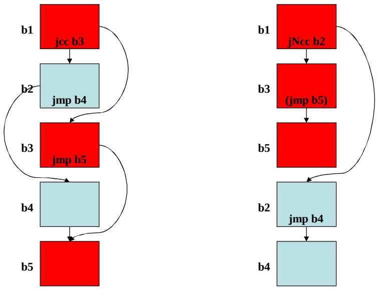
- Many optimizations depend on getting profile data from running the application. This data is then analyzed off-line to:
  - Reorder code to reduce instruction cache paging
  - Reorder code to reduce working set
  - Reorder code to reduce branch penalties
  - Rearrange static data and resource sections for additional paging improvements
  - Procedure inlining

🟥 hot

🟦 cold

Binary Optimizations

code reordering



Binary Rewriting System

- Once you have a system for reading and rewriting binaries there are other cool things you can do:
  - Build instrumentation and tools to collect a variety of program data.
  - Do code coverage testing. Inject additional test code into binaries for error case testing.
  - Modify binaries to take advantage of changes in hardware architecture.