Two Primary Types of Code Analysis

Static Analysis

- Going through your code without actually executing it
- Manual (“walkthrough”)
- Automatic (using a tool)

Dynamic Analysis

- Done as the code is executing

Static Code Analysis

What automated static analysis tools can do for you

- **Data Type checking**
  - Statically ensure that arithmetic operations are computable
    (prevent adding an integer to a boolean in C++, for example)
  - Guarantee that functions are called with the correct number/type of arguments
  - Ensure that an undefined variable is never read
- **Compiler Optimizations**
  - Software optimizer tools examine the compiled machine code
  - Convert x*2 into x + x
- **Bug Finding and other potential issues**
  - Array length may be less than zero
  - Unused local variable
  - Unnecessary return statement

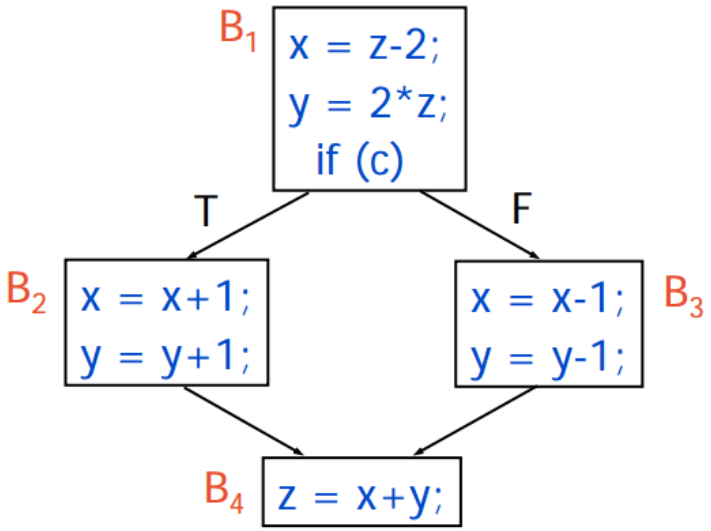
Static Code Analysis

- Control Flow Analysis (simple)
  - Looks at flow between statements
  - Which statements in the program might have affected the value of a variable?

Program

```
x = z-2 ;
y = 2*z;
if (c)  {
    x = x+1;
    y = y+1;
}
else {
    x = x-1;
    y = y-1;
}
z = x+y;
```

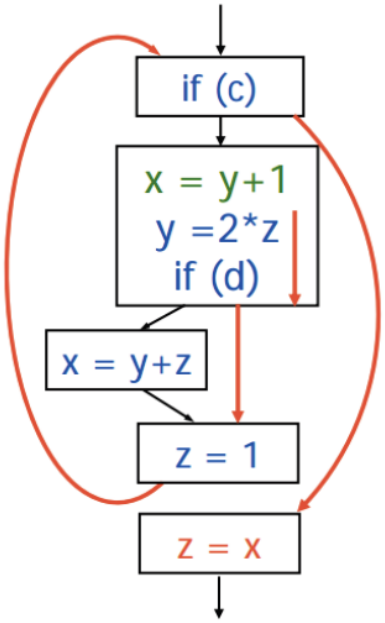
Control Flow Graph



Static Code Analysis

- Data Flow Analysis (more complex)
  - Combines flow with variable definition and use information
- Used to prove and illustrate facts about program
  - Is this variable defined before used?
  - Does this expression need to be recomputed?

The analysis detects that there is an execution that uses the value x = y+1



Static Code Analysis

Some Commercial Static Analysis Tools

- Coverity Code Advisor

  http://www.coverity.com/

- Per Coverity:

  “This tool provides Static Code Analysis allowing you to find critical defects and security weaknesses in code as it is written before they become vulnerabilities, crashes, or maintenance headaches.”

Static Code Analysis

Some Commercial Static Analysis Tools

- KlocWork (from RogueWave software)

http://www.klocwork.com/products-services/klocwork/static-code-analysis

- Per RogueWave:

"Bugs and security vulnerabilities are best found and resolved closest to where they are introduced - at the desktop or through a Continuous Integration build. Finding these sooner and fixing them quickly translates into delivering better products, faster. Klocwork runs while code is created, checking line-by-line, so issues are immediately identified and addressed. In-context resolution, to ensure remediation is done by the right people - those closest to the code."

"Actionable findings that you can fix, instantly."

Static Code Analysis

Some Commercial Static Analysis Tools

- GrammaTech CodeSonar

https://www.grammatech.com/products/codesonar

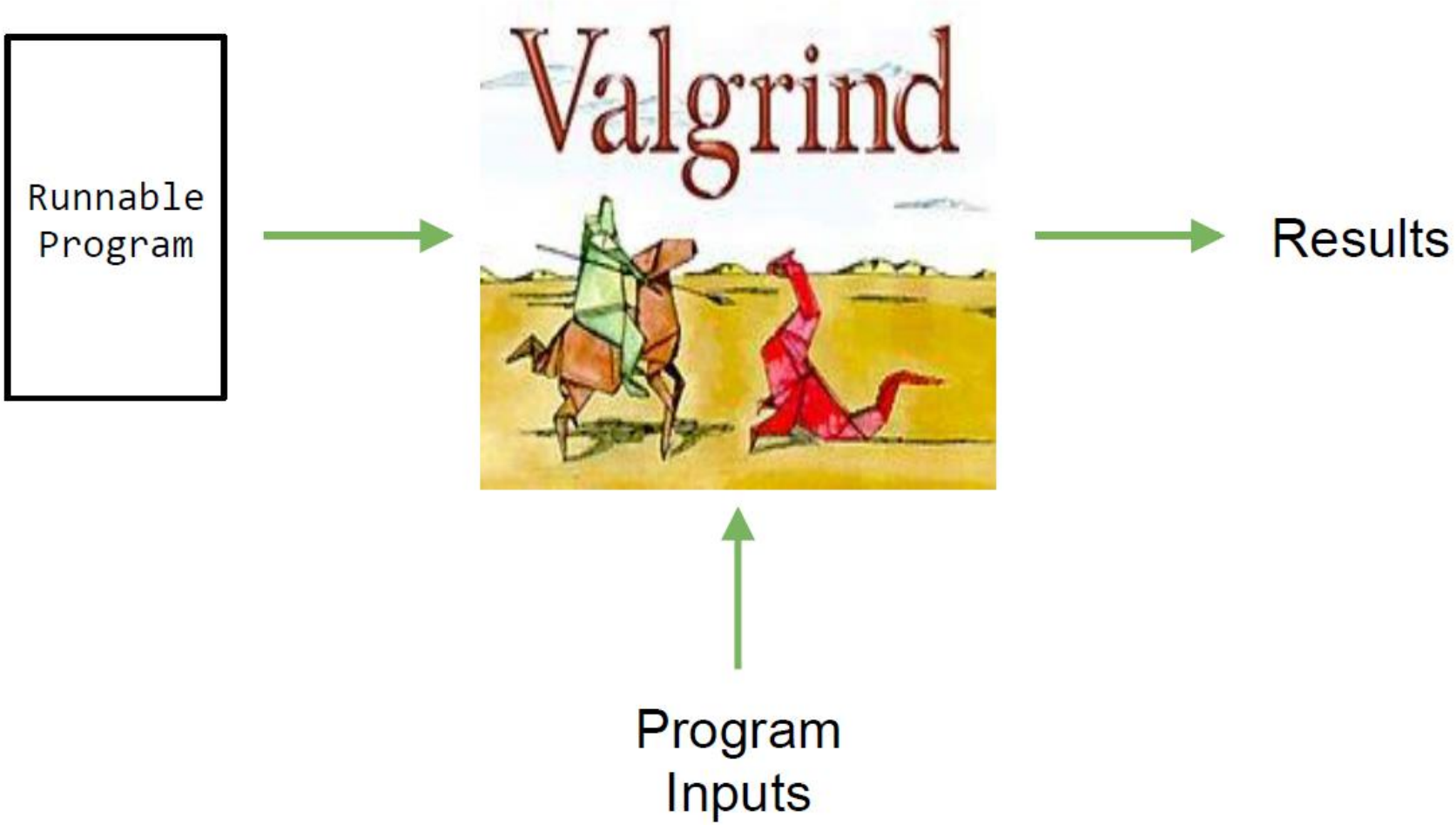
- Per GrammaTech:

“CodeSonar, GrammaTech's flagship static analysis tool, identifies bugs that can result in system crashes, unexpected behavior, and security breaches.”

“CodeSonar has been proven to provide the deepest static analysis, finding more critical defects than other static analysis tools on the market.”

Static Analysis

Dynamic Analysis



Dynamic Code Analysis

Static Analysis	Dynamic Analysis
Processes Source Code	Processes Executable Code
Detects Potential Issues	Detects Actual Issues
Analyses All Code	Analyses Only Executed Code Path
Independent of Program Inputs	Dependent on Program Inputs
Limited to Provable Issues	Limited to Runtime Issues

Dynamic Code Analysis

- Debuggers
  - GDB, DDD, IDB, LLDB, MS Visual Studio Debugger
- Memory Analyzers
  - Valgrind (Memcheck + Cachegrind + Massif)
  - mtrace, Purify, Deleaker
- Thread/Concurrency Analyzers
  - Valgrind (Helgrind + DRD)
  - jstack, Racer
- Simulators/Emulators
  - Valgrind (Chachegrind)
  - Dinero, Android Emulator
- Profilers
  - Valgrind + Kcachegrind (Callgrind + Cachegrind + Massif)
  - gprof, Pin, Windows Performance Toolkit

Dynamic Code Analysis

Debuggers

- Run the program through the debugger
- Allows you to
  - Intercept an abort and work backwards
  - Step through code line by line
  - See the value of a variable
  - Set a break point

You did GDB in lab.

Dynamic Code Analysis

Memory Analyzer

- Run the program through the tool checking memory usage
- Allows you to
  - Pinpoint a memory leak (where your program is attempting to access or grab memory beyond its boundaries.)
  - Find an example of a memory leak

(You did Valgrind Memcheck in lab.)

Dynamic Code Analysis

Profiler

- Run the program through the tool to examine where in the code it is spending the most time
- Allows you to
  - See if data and/or instruction reads are using cache or not (define what is cache and why it is important. L1 versus L2.)
  - See time spent during function calls
    - Perhaps a call is performing poorly and you can tune it

(You did Valgrind Kcachegrind in lab.)