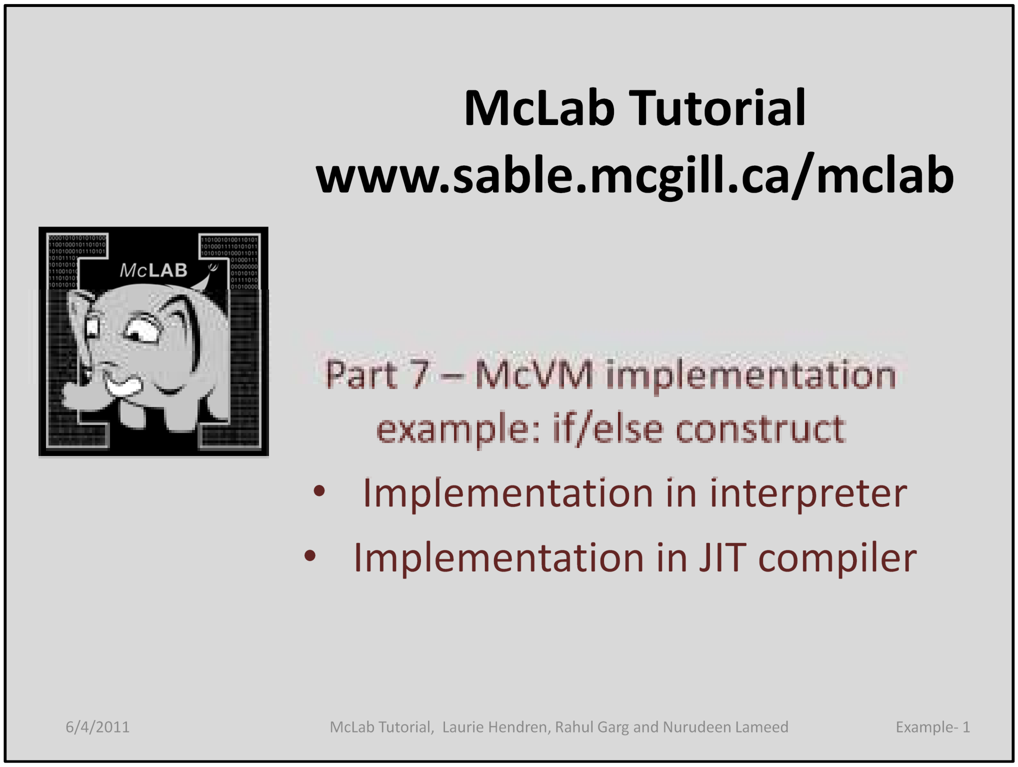
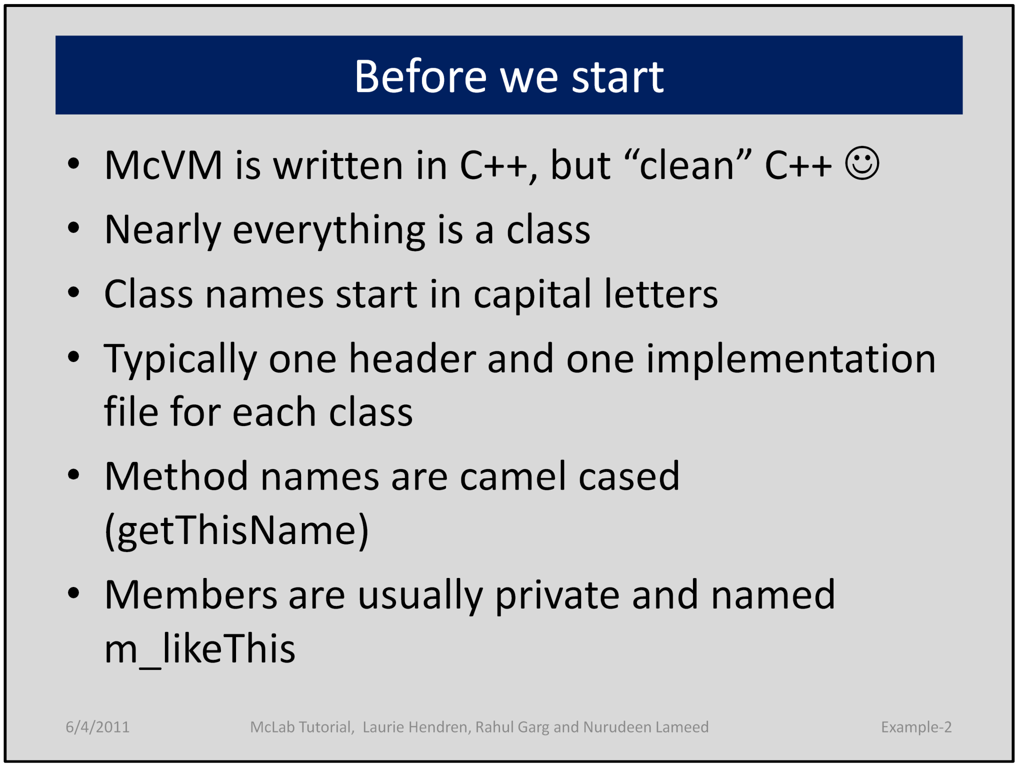


McLab Tutorial
www.sable.mcgill.ca/mclab

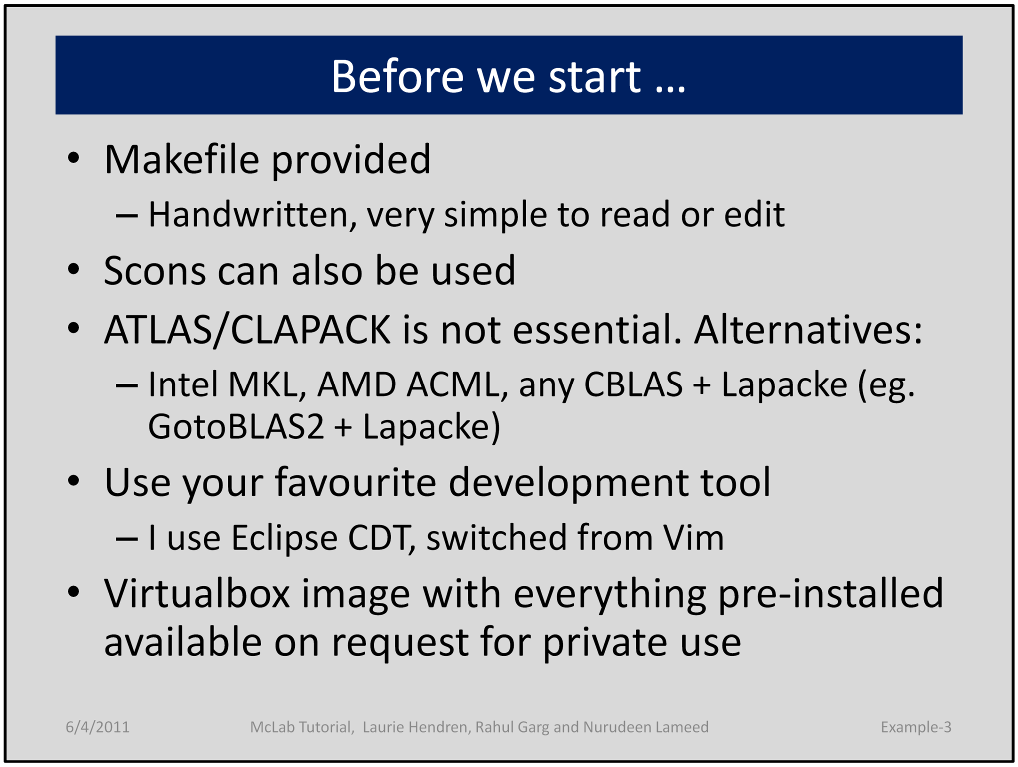
Part 7 – McVM implementation example: if/else construct

- Implementation in interpreter
- Implementation in JIT compiler



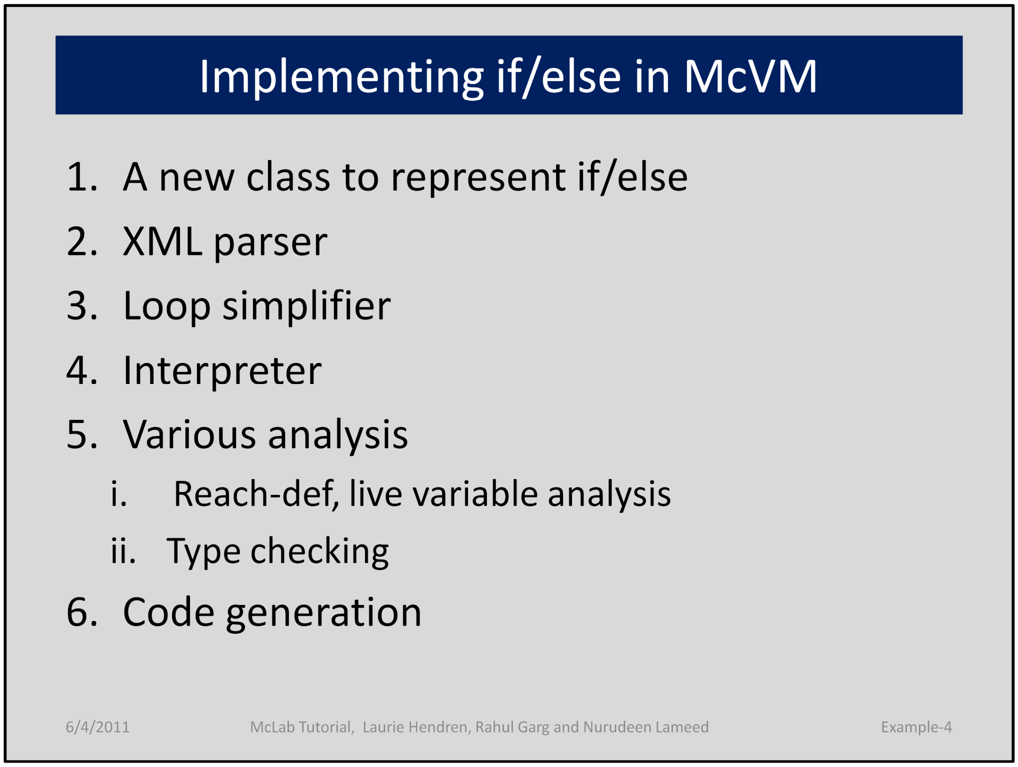
Before we start

- McVM is written in C++, but “clean” C++ ☺
- Nearly everything is a class
- Class names start in capital letters
- Typically one header and one implementation file for each class
- Method names are camel cased (getThisName)
- Members are usually private and named m_likeThis



Before we start …

- Makefile provided
  - Handwritten, very simple to read or edit
- Scons can also be used
- ATLAS/CLAPACK is not essential. Alternatives:
  - Intel MKL, AMD ACML, any CBLAS + Lapacke (eg. GotoBLAS2 + Lapacke)
- Use your favourite development tool
  - I use Eclipse CDT, switched from Vim
- Virtualbox image with everything pre-installed available on request for private use

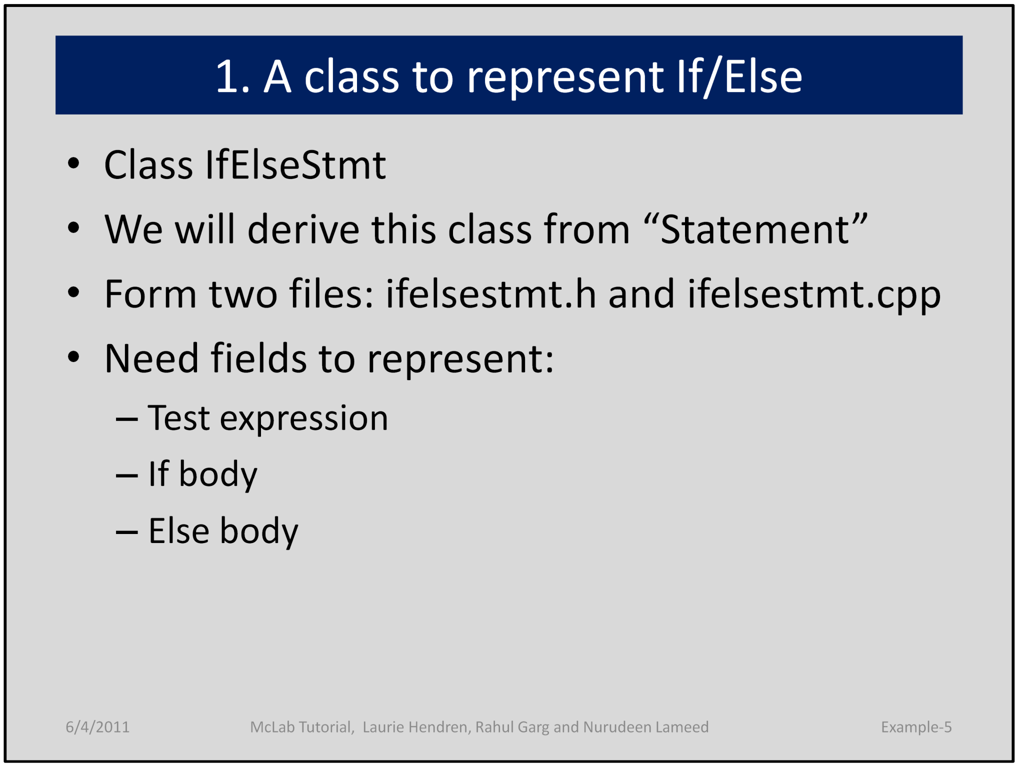


Implementing if/else in McVM

1. A new class to represent if/else
2. XML parser
3. Loop simplifier
4. Interpreter
5. Various analysis
   1. Reach-def, live variable analysis
   2. Type checking
6. Code generation

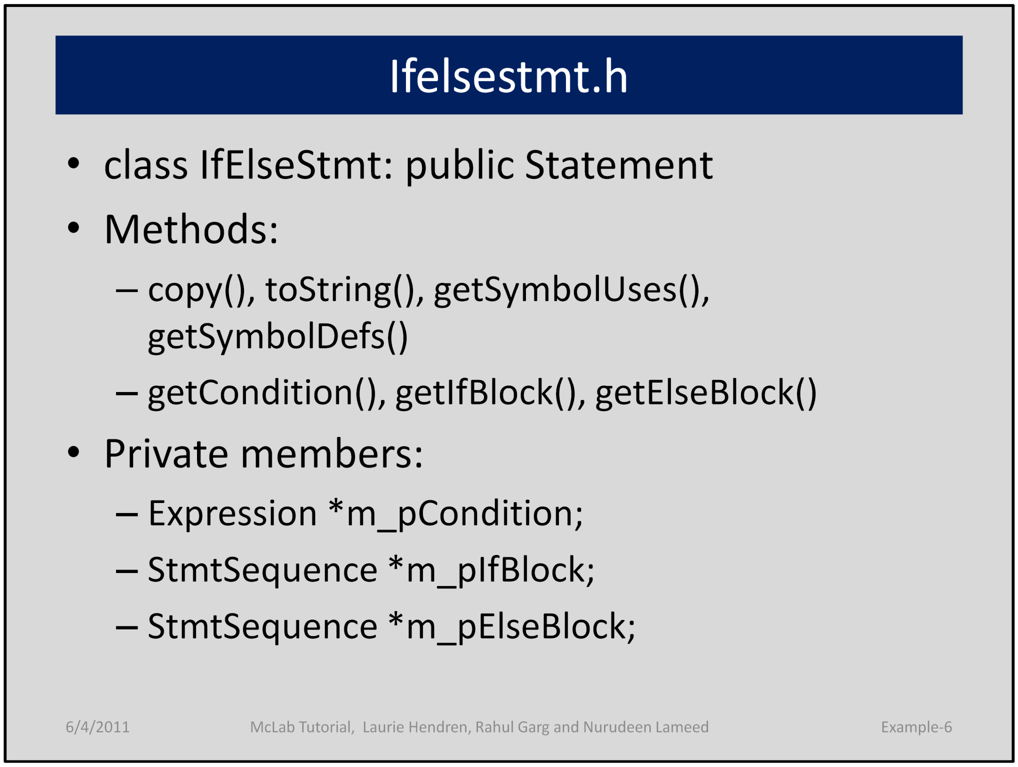
6/4/2011 McLab Tutorial, Laurie Hendren, Rahul Garg and Nurudeen Lameed Example-4

The amount of effort required to implement each step is unequal. Step 5 is the largest most complicated step.



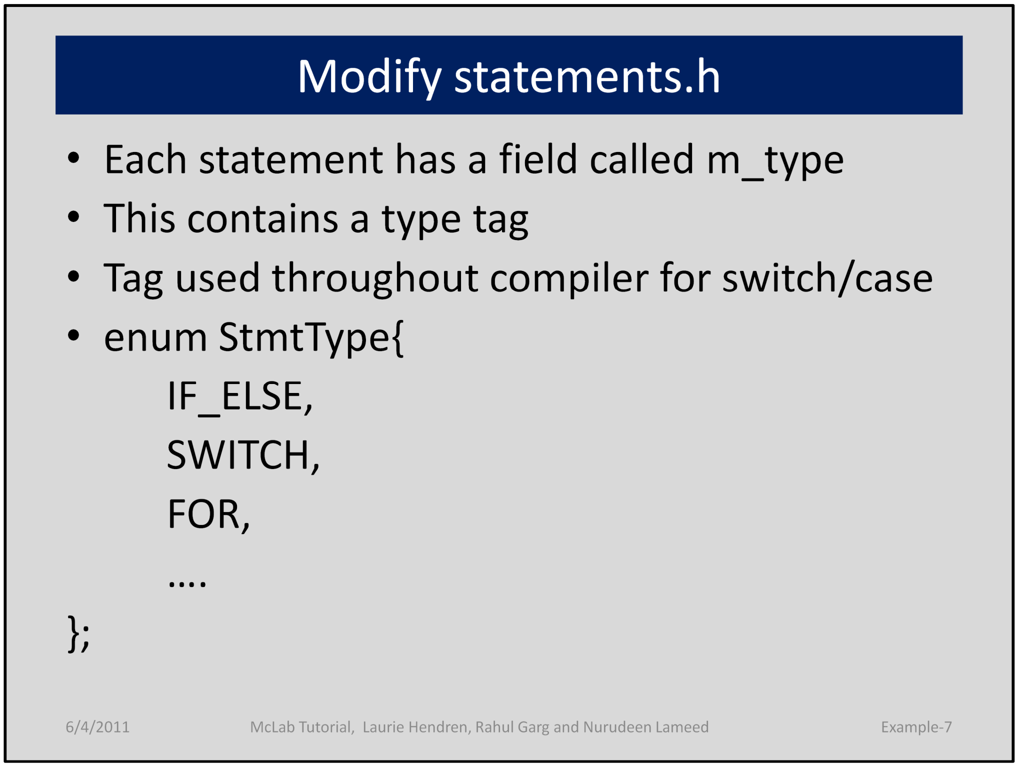
1. A class to represent If/Else

- Class IfElseStmt
- We will derive this class from “Statement”
- Form two files: ifelsestmt.h and ifelsestmt.cpp
- Need fields to represent:
  - Test expression
  - If body
  - Else body



Ifelsestmt.h

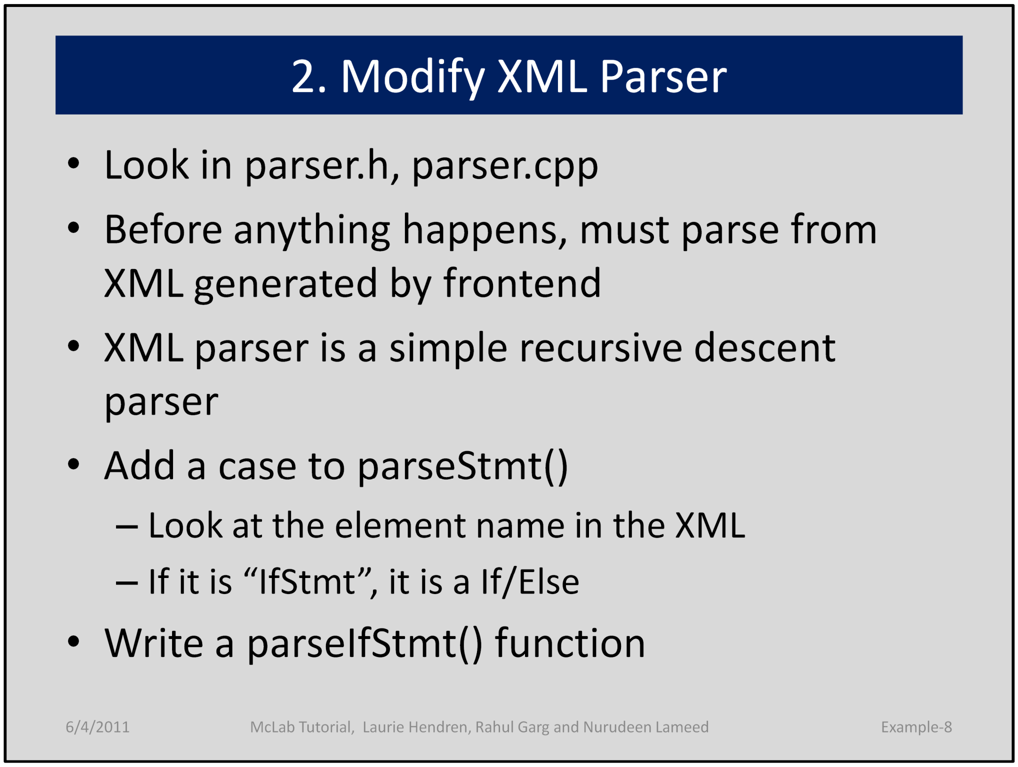
- class IfElseStmt: public Statement
- Methods:
  - copy(), toString(), getSymbolUses(), getSymbolDefs()
  - getCondition(), getIfBlock(), getElseBlock()
- Private members:
  - Expression *m_pCondition;
  - StmtSequence *m_pIfBlock;
  - StmtSequence *m_pElseBlock;



Modify statements.h

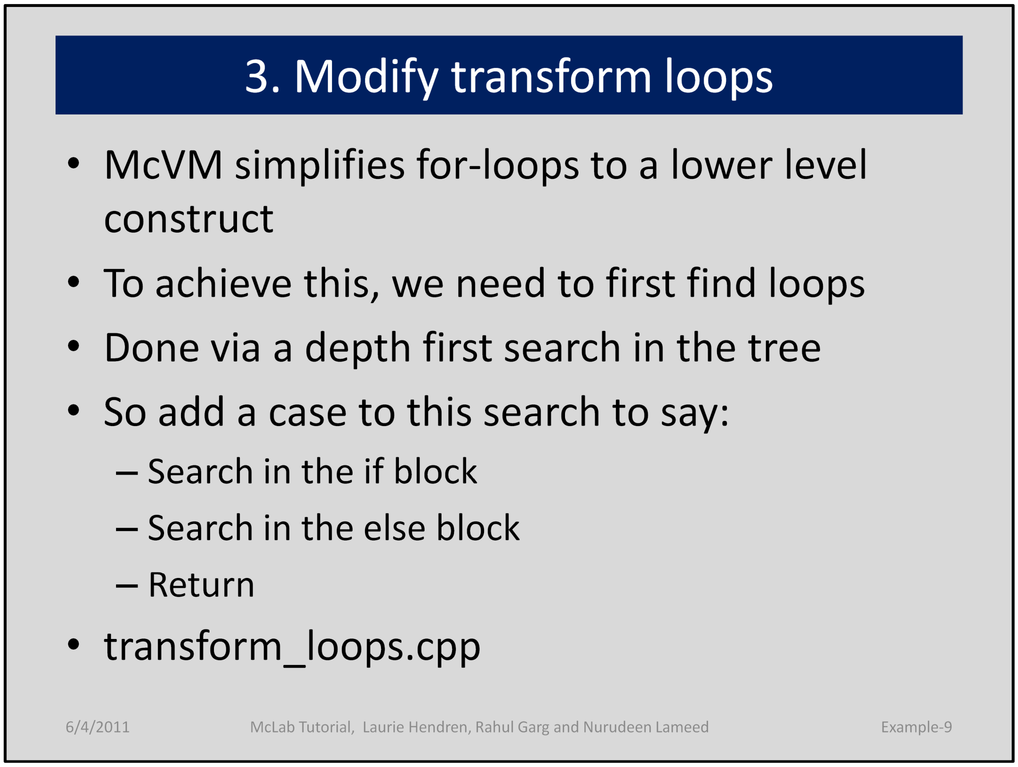
- Each statement has a field called m_type
- This contains a type tag
- Tag used throughout compiler for switch/case
- enum StmtType{

```
        IF_ELSE,
        SWITCH,
        FOR,
        ….
};
```



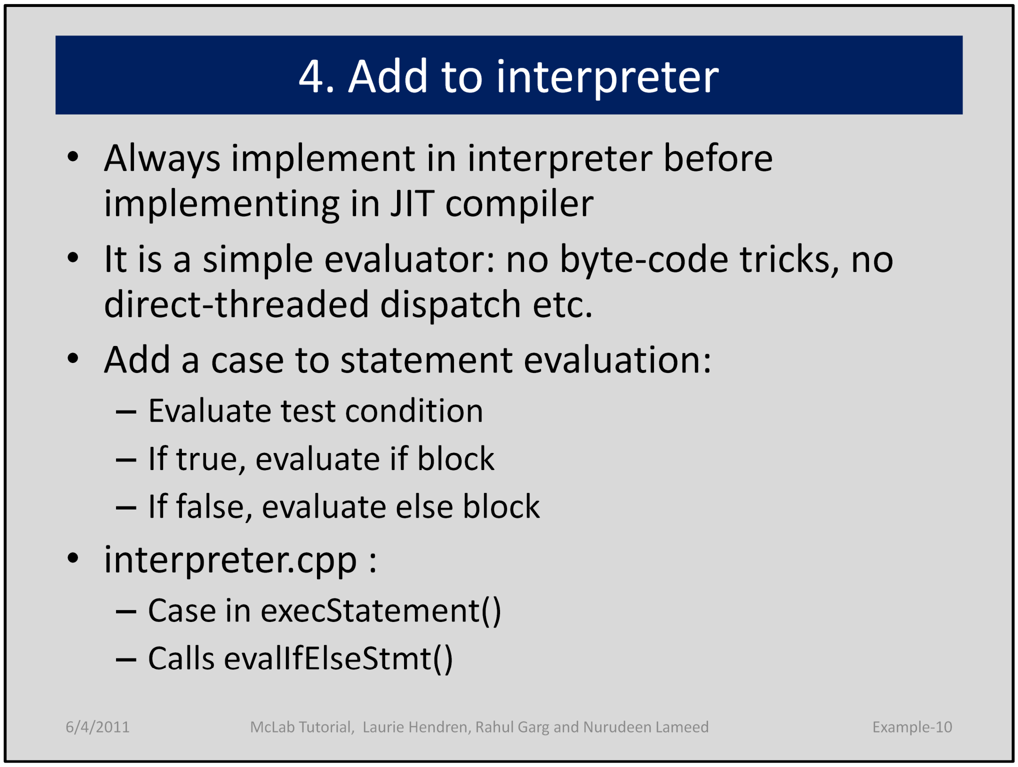
2. Modify XML Parser

- Look in parser.h, parser.cpp
- Before anything happens, must parse from XML generated by frontend
- XML parser is a simple recursive descent parser
- Add a case to parseStmt()
  - Look at the element name in the XML
  - If it is “IfStmt”, it is a If/Else
- Write a parseIfStmt() function



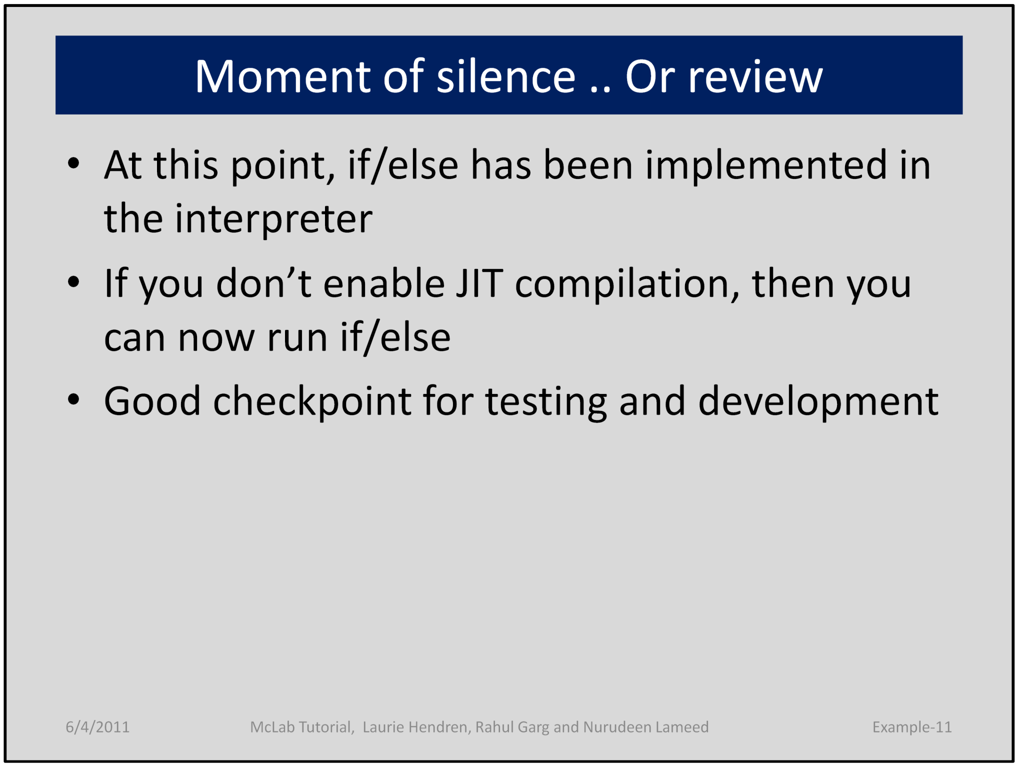
3. Modify transform loops

- McVM simplifies for-loops to a lower level construct
- To achieve this, we need to first find loops
- Done via a depth first search in the tree
- So add a case to this search to say:
  - Search in the if block
  - Search in the else block
  - Return
- transform_loops.cpp



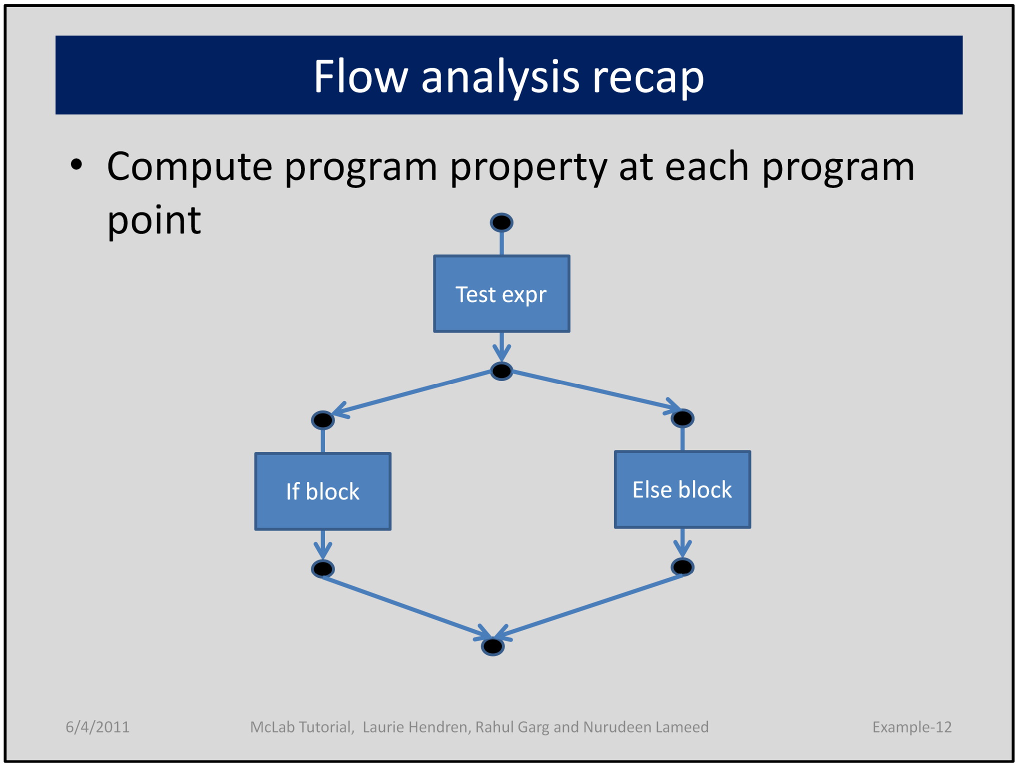
4. Add to interpreter

- Always implement in interpreter before implementing in JIT compiler
- It is a simple evaluator: no byte-code tricks, no direct-threaded dispatch etc.
- Add a case to statement evaluation:
  - Evaluate test condition
  - If true, evaluate if block
  - If false, evaluate else block
- interpreter.cpp :
  - Case in execStatement()
  - Calls evalIfElseStmt()



Moment of silence .. Or review

- At this point, if/else has been implemented in the interpreter
- If you don’t enable JIT compilation, then you can now run if/else
- Good checkpoint for testing and development



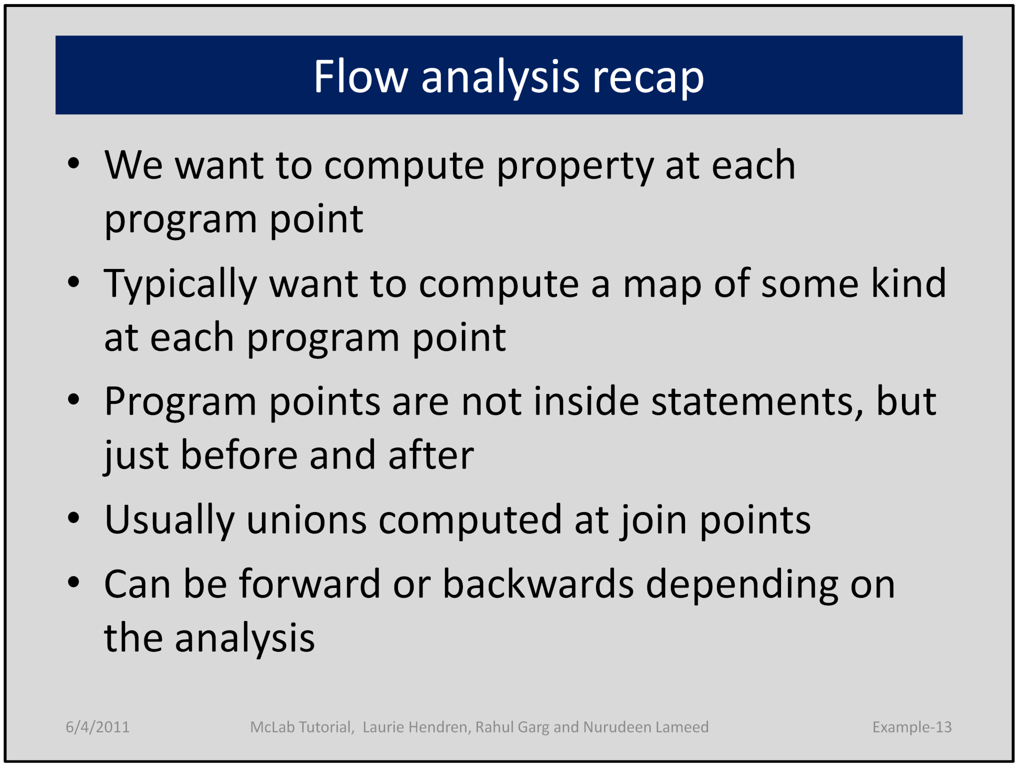
Flow analysis recap

- Compute program property at each program point

Test expr

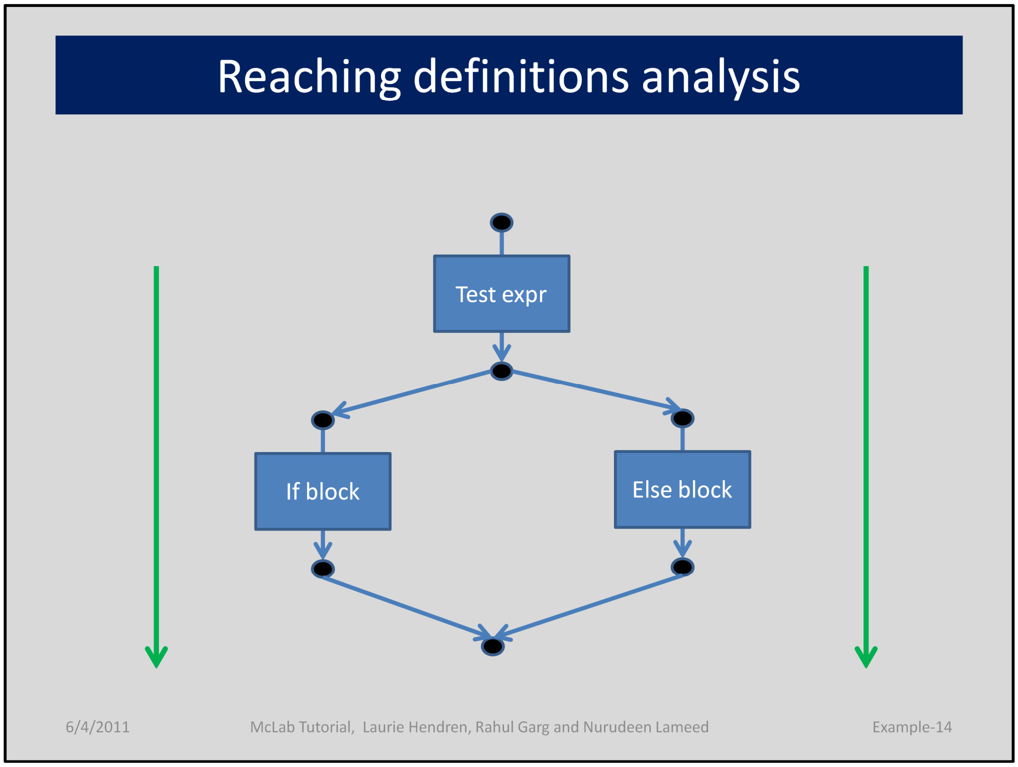
If block

Else block



Flow analysis recap

- We want to compute property at each program point
- Typically want to compute a map of some kind at each program point
- Program points are not inside statements, but just before and after
- Usually unions computed at join points
- Can be forward or backwards depending on the analysis



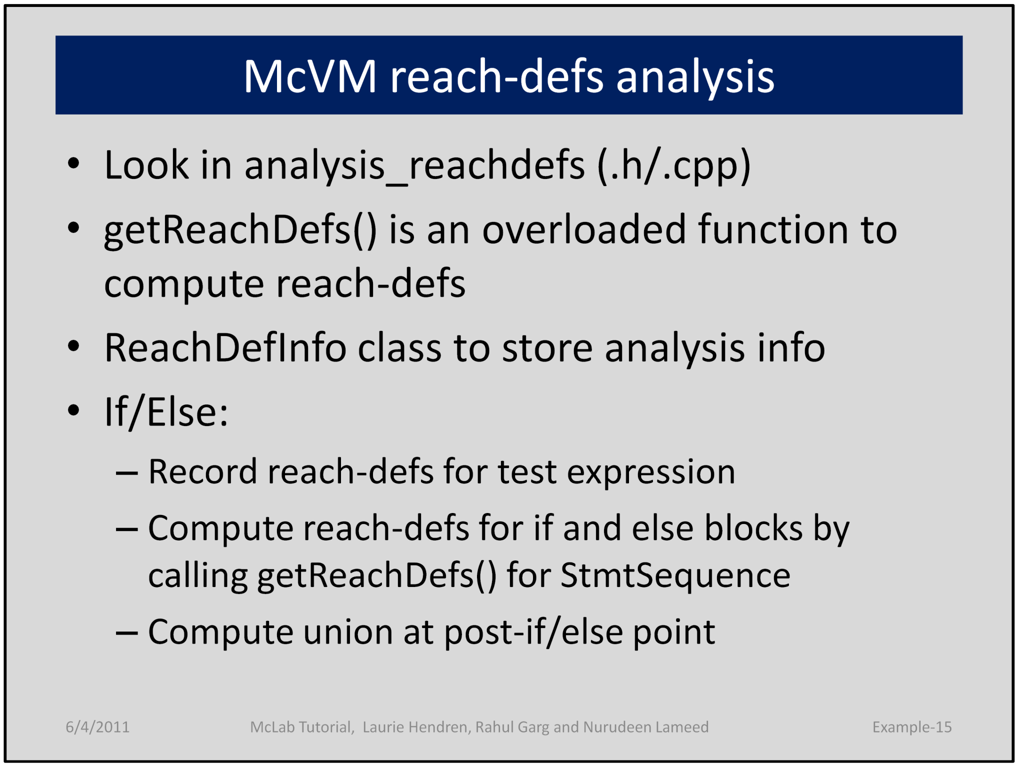
Reaching definitions analysis

Test expr

If block

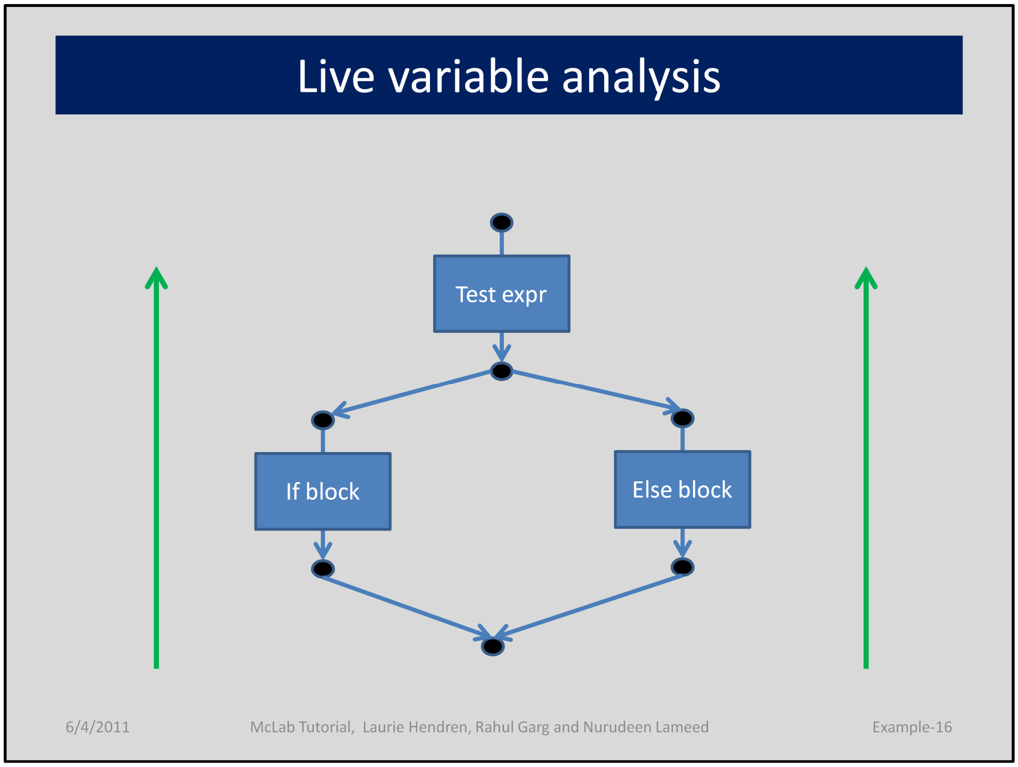
Else block

6/4/2011 McLab Tutorial, Laurie Hendren, Rahul Garg and Nurudeen Lameed Example-14

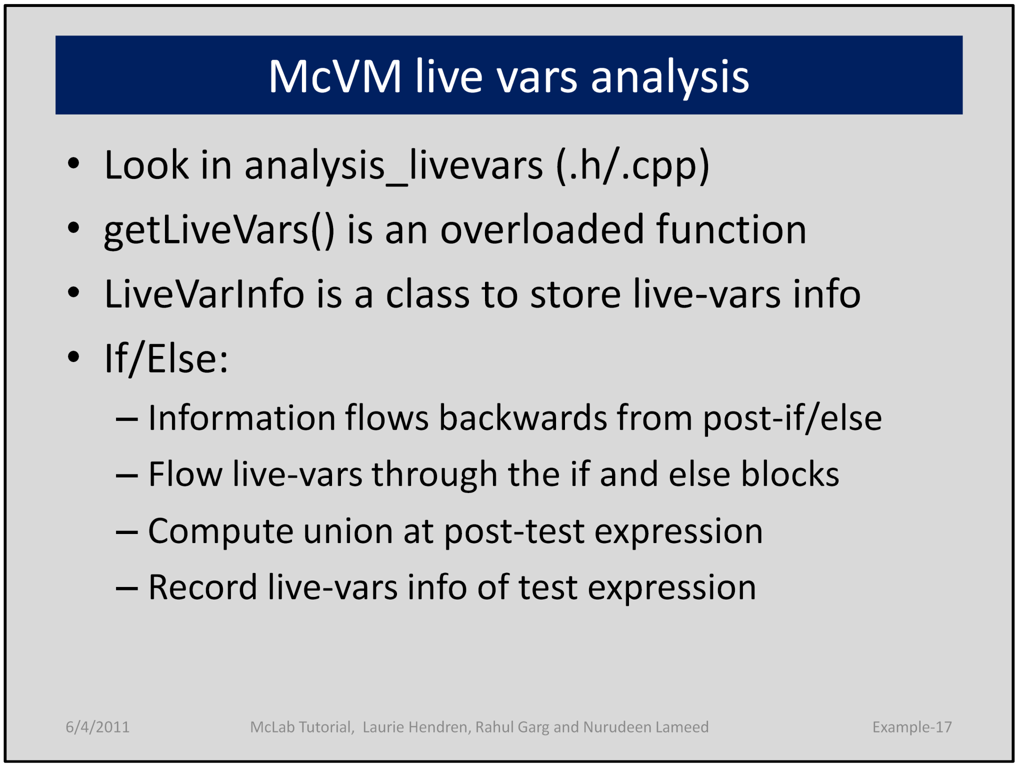


McVM reach-defs analysis

- Look in analysis_reachdefs (.h/.cpp)
- getReachDefs() is an overloaded function to compute reach-defs
- ReachDefInfo class to store analysis info
- If/Else:
  - Record reach-defs for test expression
  - Compute reach-defs for if and else blocks by calling getReachDefs() for StmtSequence
  - Compute union at post-if/else point

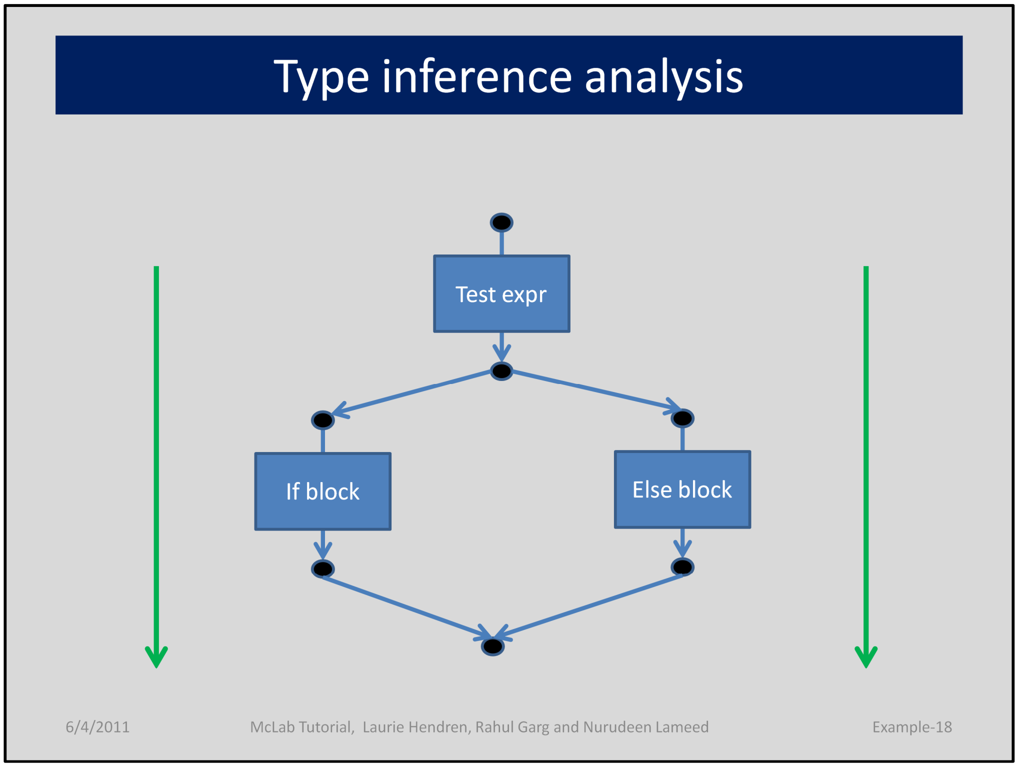


Live variable analysis

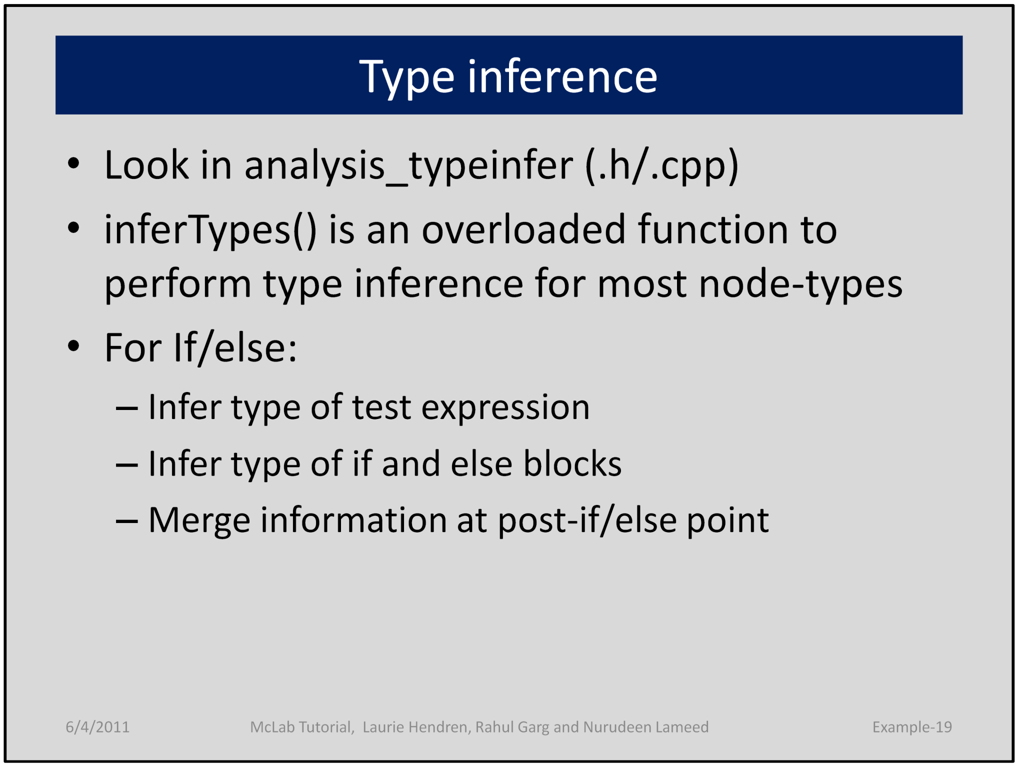


McVM live vars analysis

- Look in analysis_livevars (.h/.cpp)
- getLiveVars() is an overloaded function
- LiveVarInfo is a class to store live-vars info
- If/Else:
  - Information flows backwards from post-if/else
  - Flow live-vars through the if and else blocks
  - Compute union at post-test expression
  - Record live-vars info of test expression

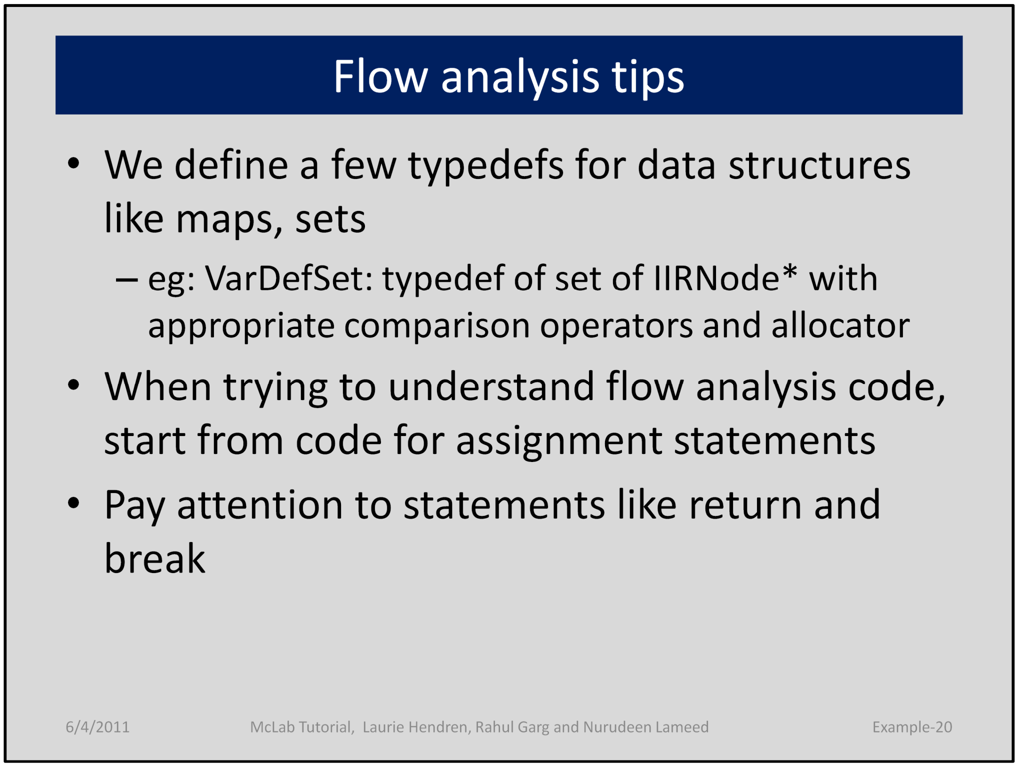


Type inference analysis



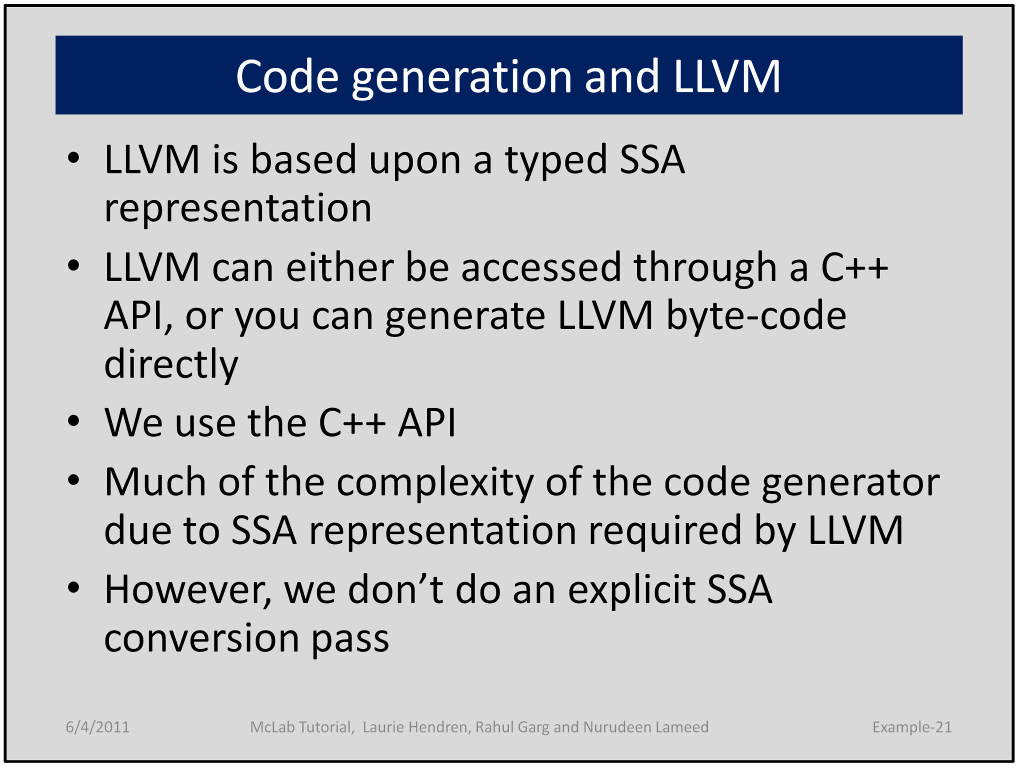
Type inference

- Look in analysis_typeinfer (.h/.cpp)
- inferTypes() is an overloaded function to perform type inference for most node-types
- For If/else:
  - Infer type of test expression
  - Infer type of if and else blocks
  - Merge information at post-if/else point



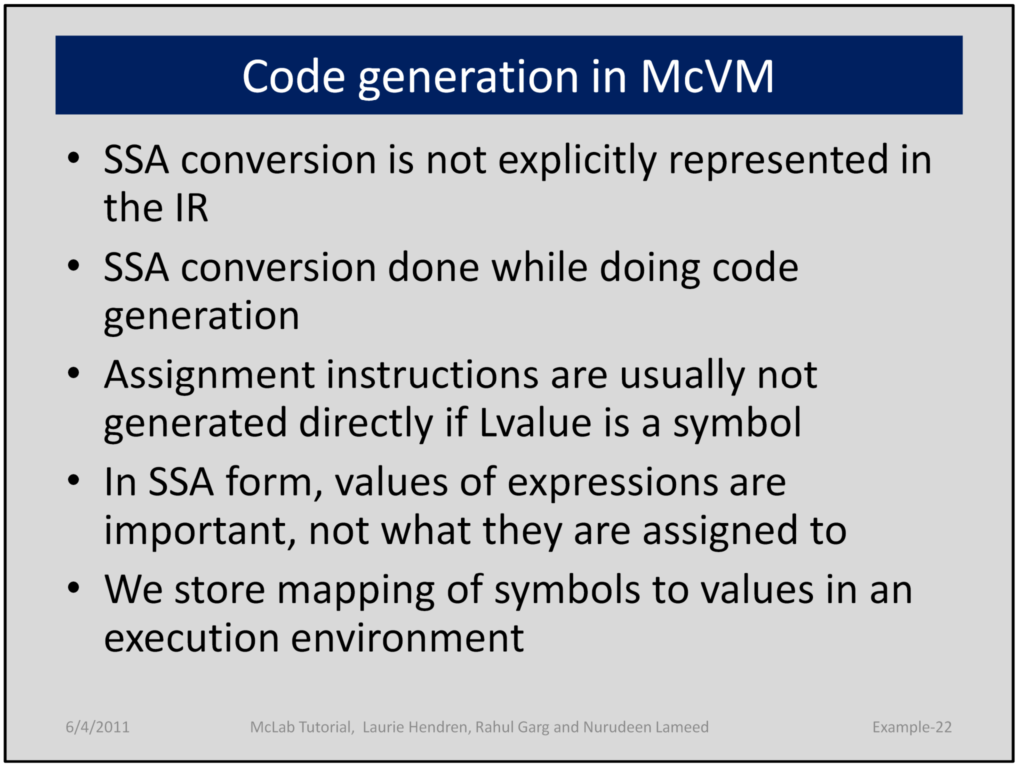
Flow analysis tips

- We define a few typedefs for data structures like maps, sets
  - eg: VarDefSet: typedef of set of IIRNode* with appropriate comparison operators and allocator
- When trying to understand flow analysis code, start from code for assignment statements
- Pay attention to statements like return and break



Code generation and LLVM

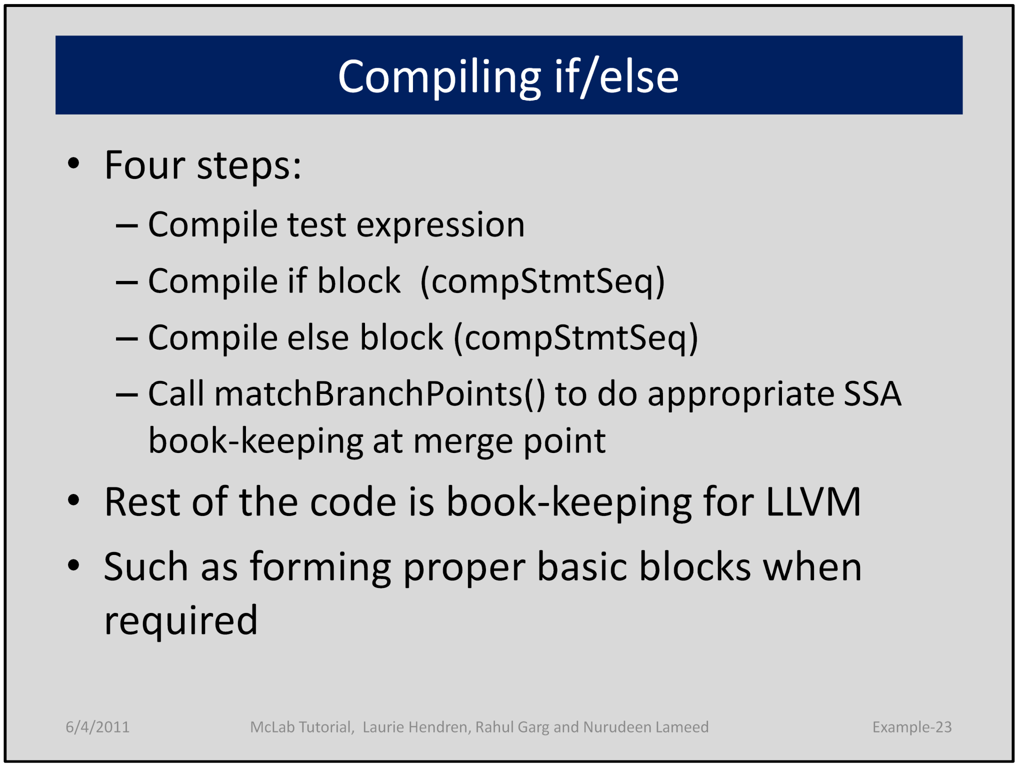
- LLVM is based upon a typed SSA representation
- LLVM can either be accessed through a C++ API, or you can generate LLVM byte-code directly
- We use the C++ API
- Much of the complexity of the code generator due to SSA representation required by LLVM
- However, we don’t do an explicit SSA conversion pass



Code generation in McVM

- SSA conversion is not explicitly represented in the IR
- SSA conversion done while doing code generation
- Assignment instructions are usually not generated directly if Lvalue is a symbol
- In SSA form, values of expressions are important, not what they are assigned to
- We store mapping of symbols to values in an execution environment

A complete explaination is probably out of the scope of the tutorial. You are referred to compAssignStmt() in jitcompiler.cpp for details.



Compiling if/else

- Four steps:
  - Compile test expression
  - Compile if block (compStmtSeq)
  - Compile else block (compStmtSeq)
  - Call matchBranchPoints() to do appropriate SSA book-keeping at merge point
- Rest of the code is book-keeping for LLVM
- Such as forming proper basic blocks when required