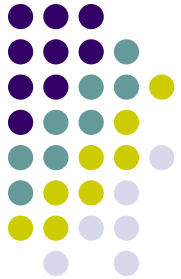
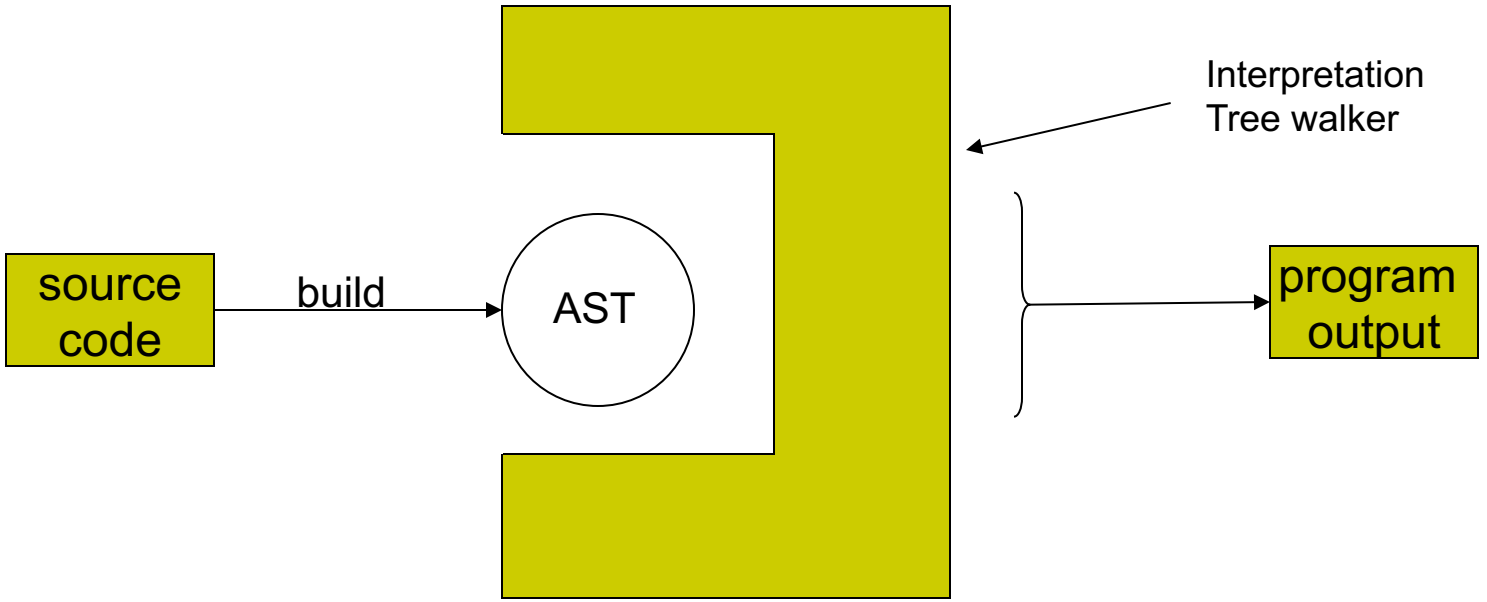
Language Implementation Review



- A typical programming language can be implemented in one of two ways
  - *Interpreter* - interpreters execute the instructions of the source language directly and produce the desired ouput
  - *Compiler* - compilers translate the source program into another programming language which in turn needs to be interpreted to produce results.
    - If the target language is interpreted by a hardware interpreter then we refer to it as *machine language*.

Interpreters

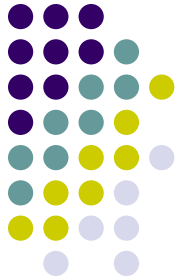
- Interpreters are usually constructed using tree walkers
- We have seen two interpreters: the bytecode interpreter and the Cuppa1 interpreter.



The bytecode Interpreter

- “AST” was a list of instructions
- Tree walker simply simulated the instructions using a ‘symbol’ table and a ‘label’ table.
- See: csc402-In005.pdf

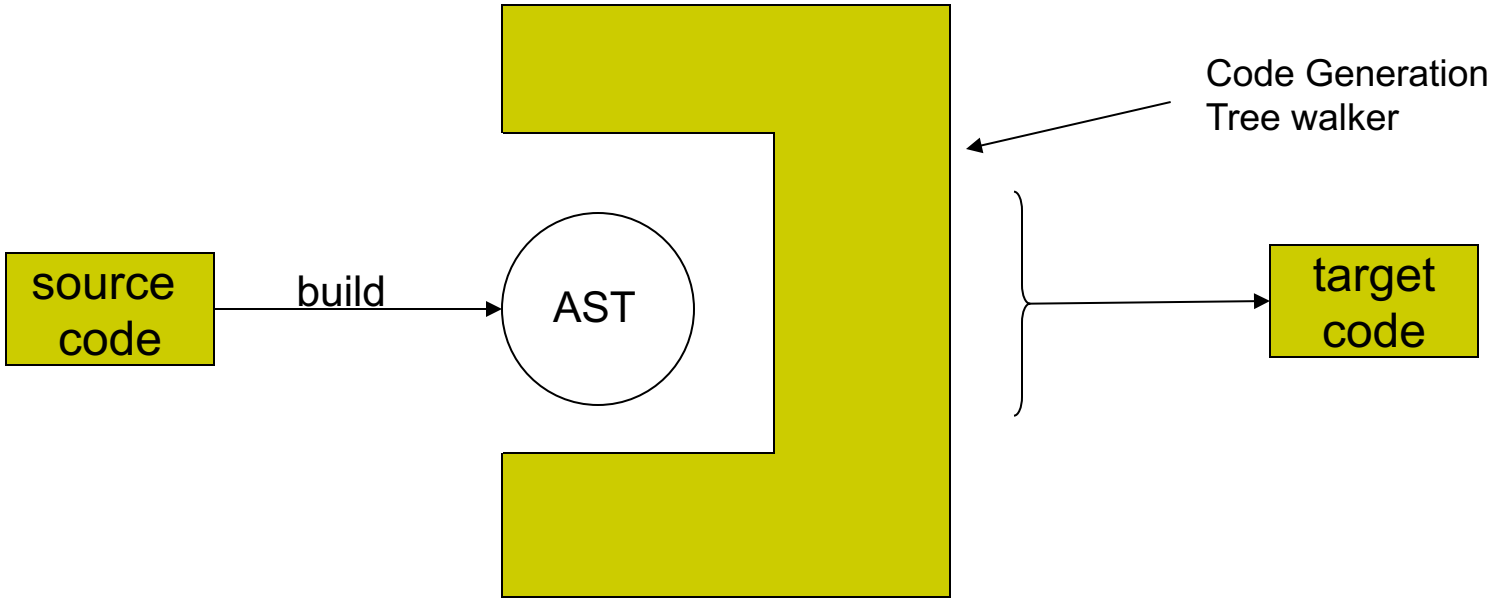
Cuppa1 Interpreter



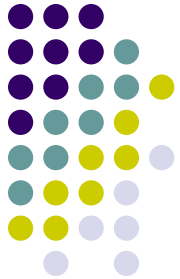
- AST was an abstract representation derived from the parse tree
- We used tree walker to interpret the ast which made use of a ‘symbol’ table
- See: csc402-In006.pdf

Compilers

- Compilers are usually constructed using tree walkers
- We have seen one compiler: the cuppa1 to bytecode compiler



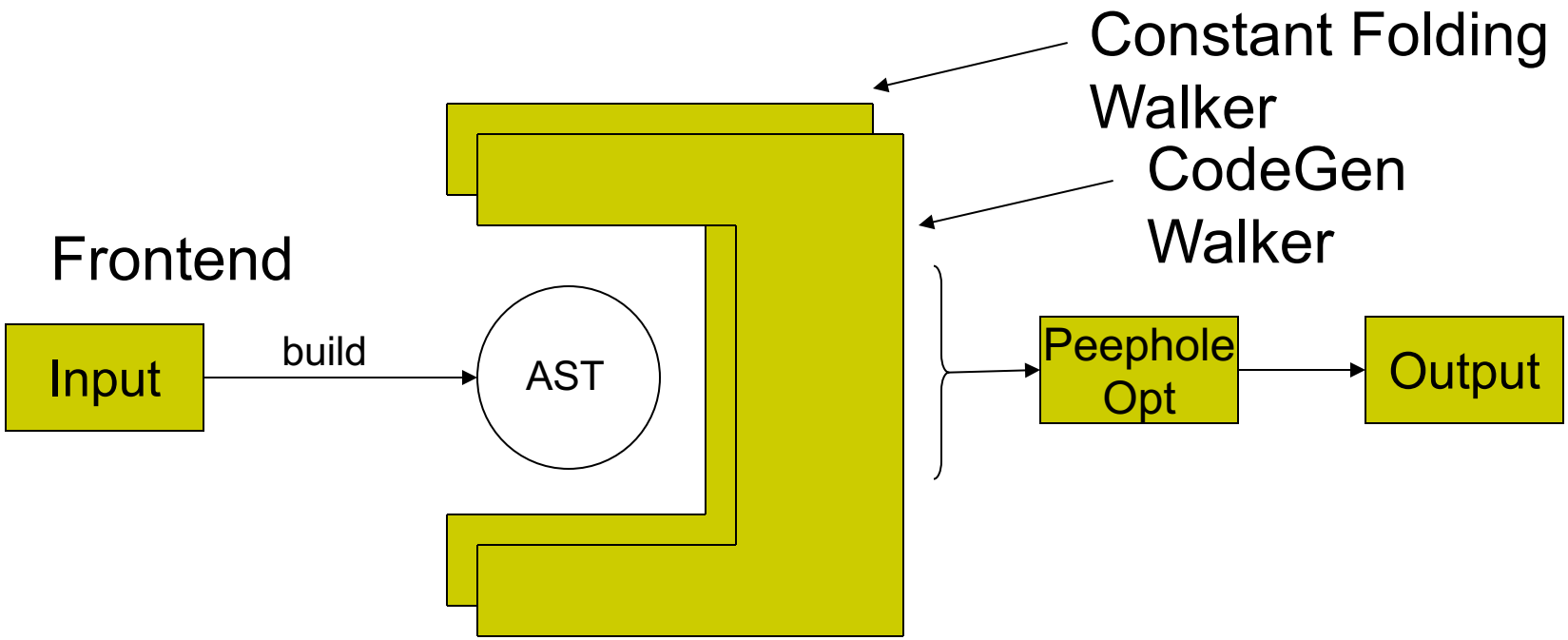
Cuppa1 to Bytecode Compiler



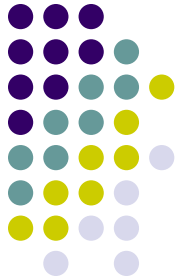
- AST derived from parse tree
- Tree walker for code generation
- See: csc402-ln007.pdf

Optimizing Compilers

- Optimizing compilers have additional phases and modules that allow them to produce more efficient target code.



Optimizing Compilers



- Our optimizing compiler for Cuppa1 had two optimization phases:
  - A constant folding tree rewriter
  - A peephole optimizer
- See: csc402-ln008.pdf