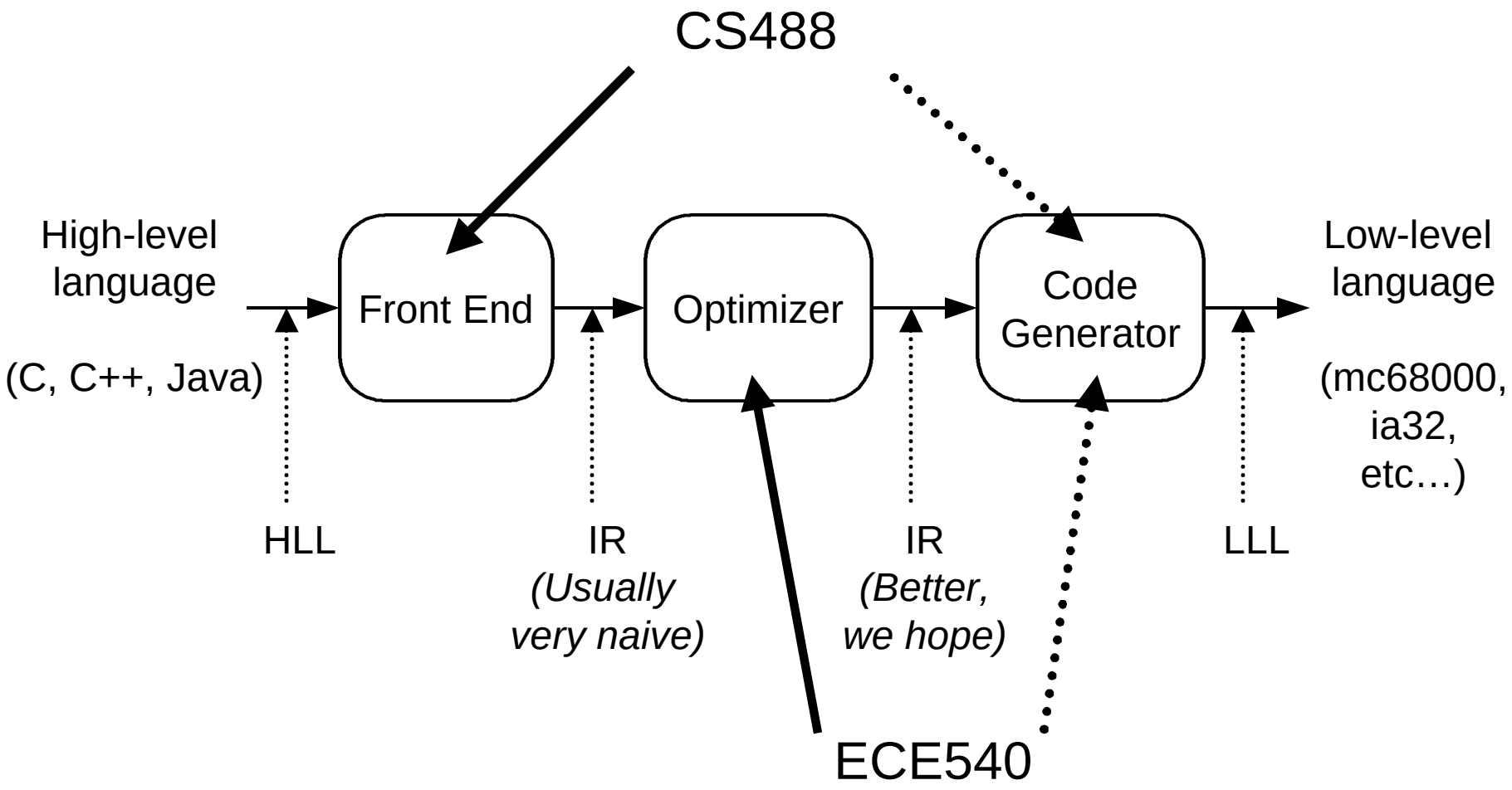
ECE1724F

Compiler Primer

http://www.eecg.toronto.edu/~voss/ece1724f

Sept. 18, 2002

What's in an optimizing compiler?



What are compiler optimizations?

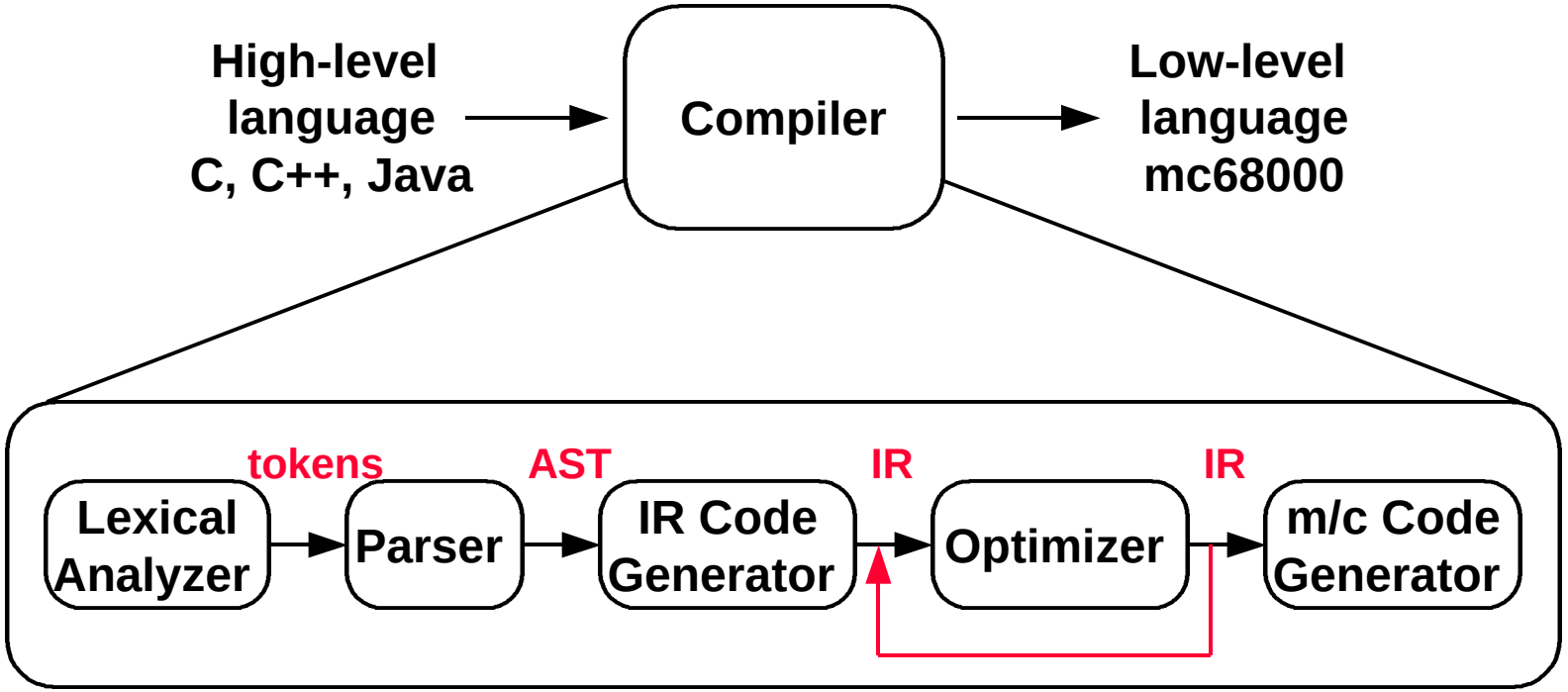
Optimization: the transformation of a program P into a program P', that has the same input/output behavior, but is somehow "better".

- "better" means:
  - faster
  - or smaller
  - or uses less power
  - or whatever you care about
- P' is not optimal, may even be worse than P

An optimizations must:

- Preserve correctness
  - the speed of an incorrect program is irrelevant
- On average improve performance
  - P’ is not optimal, but it should usually be better
- Be “worth the effort”
  - 1 person-year of work, 2x increase in compilation time, a 0.1% improvement in speed?
  - Find the bottlenecks
  - 90/10 rule: 90% of the gain for 10% of the work

Compiler Phases (Passes)



IR: Intermediate Representation

Control Flow Analysis

Purpose of Control Flow Analysis

- Determine the control structure of a program
  - determine possible control flow paths
  - find basic blocks and loops
- Intraprocedural: within a procedure
- Interprocedural: across procedures
  - Whole program
  - Maybe just within the same file

cc –c file1.c
cc –c file2.c
cc –o myprogram file1.o file2.o -l mylib

All about Control flow analysis:

- Finding basic blocks
- Creating a control flow graph
- Finding dominators
  - dominators, proper dominators, direct dominators
- Finding post-dominators
- Finding loops

Basic Blocks

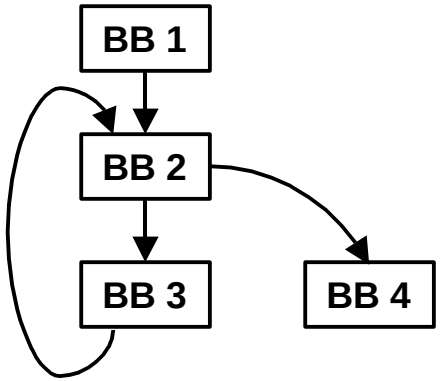
- A Basic Block (BB) is a maximal section of “straight-line” code which can only be entered via the first instruction and can only be existed via the last instruction.

S1	**read L**	**BB 1**
S2	**n = 0**	
S3	**k = 0**	
S4	**m = 1**	
S5	**k = k + m**	**BB 2**
S6	**c = k > L**	
S7	**if (c) goto S11**	
S8	**n = n +1**	**BB 3**
S9	**m = m +2**	
S10	**goto S5**	
S11	**write n**	**BB 4**

Control Flow Graphs

- The Control Flow Graph (CFG) of a program is a directed graph G=(N, E) whose nodes N represent the basic blocks in the program and whose edges E represent transfers of control between basic blocks.

```
S1    read L          }
S2    n = 0           }  BB 1
S3    k = 0           }
S4    m = 1           }
S5    k = k + m       }
S6    c = k > L       }  BB 2
S7    if (c) goto S11 }
S8    n = n +1        }
S9    m = m +2        }  BB 3
S10   goto S5         }
S11   write n         }  BB 4
```



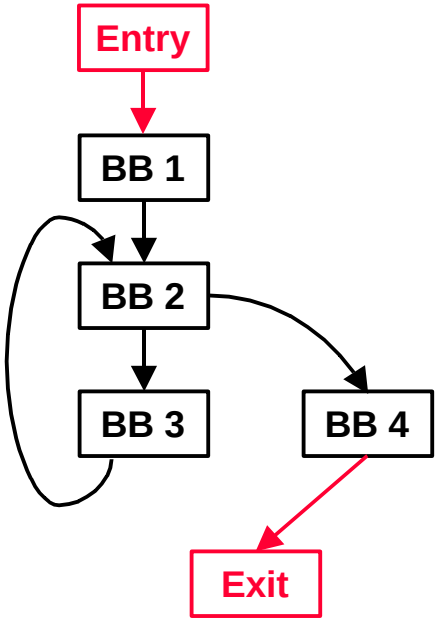
Control Flow Graphs (continued)

- Given G = (N, E) and a basic block b $\in$ N.
- The successors of b, denoted by succ(b), is the set of basic blocks that can be reached from b by traversing one edge:

$$succ(b) = \{n \in N \mid (b,n) \in E\}$$

- The predecessors of b, denoted by pred(b), is the set of basic blocks that can reach b by traversing one edge:

$$pred(b) = \{m \in N \mid (m,b) \in E\}$$

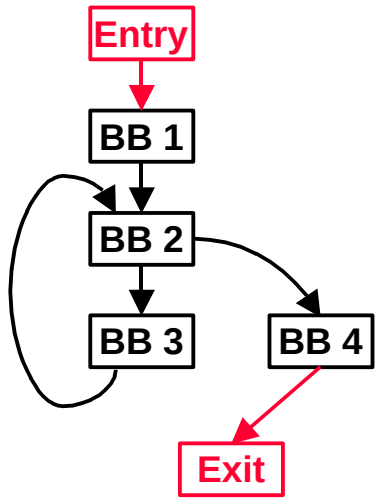


- An entry node in G is one which has no predecessors.
- An exit node in G is one which has no successors.
- It is common and convenient to add an entry node Entry and an exit node Exit to the CFG.

Dominators

- Let G=(N, E) denote a CFG. Let n, n' $\in$ N.
- n is said to dominate n', denoted n *dom* n', iff every path from Entry to n' contains n.

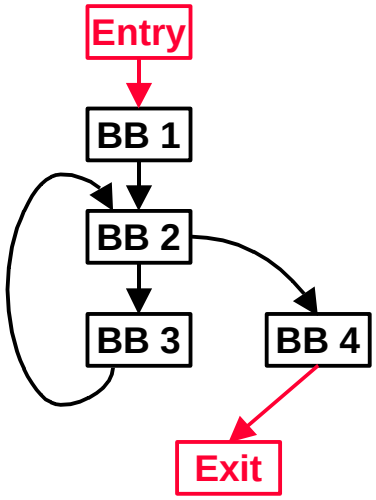
1 *dom* 1 ; 1 *dom* 2 ; 1 *dom* 3 ; 1 *dom* 4 ;
2 *dom* 2 ; 2 *dom* 3 ; 2 *dom* 4 ; 3 *dom* 3 ;
4 *dom* 4.



Post-Dominators

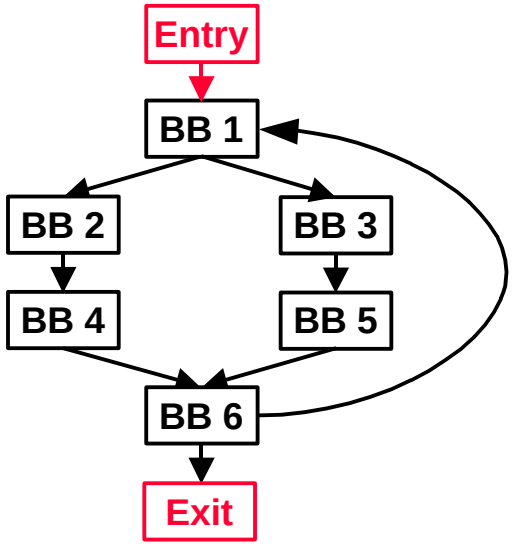
- Let G=(N, E) denote a CFG. Let n, n' ∈ N. Then:
- n is said to post-dominate n', denoted n *pdom* n', iff every path from n' to Exit contains n.

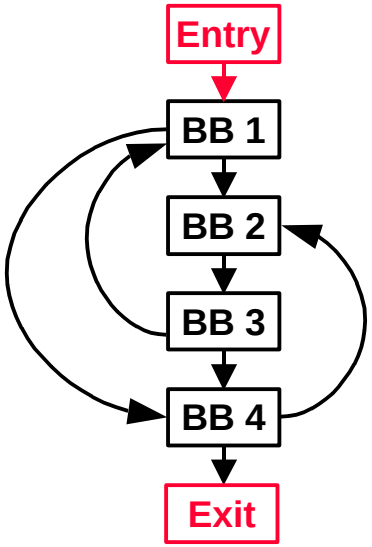
1 *pdom* 1 ; 2 *pdom* 1 ; 4 *pdom* 1 ; 2 *pdom* 2 ;
4 *pdom* 2 ; 3 *pdom* 3; 2 *pdom* 3 ; 4 *pdom 3;*
4 *pdom* 4.

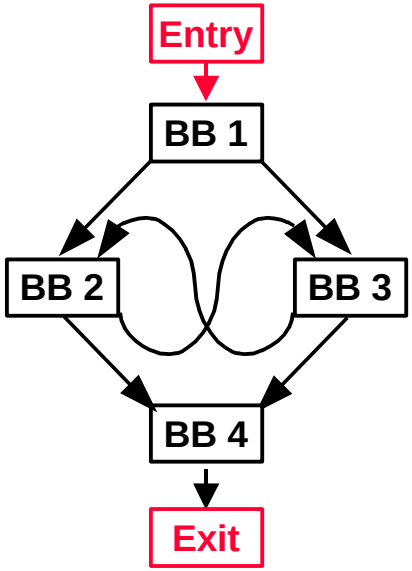


Loops

- Goal: find loops in CFG irrespective of input syntax
  - DO, while, for, goto, etc.
- Intuitively, a loop is the set of nodes in a CFG that form a cycle.
- However, not every cycle is a “loop”.



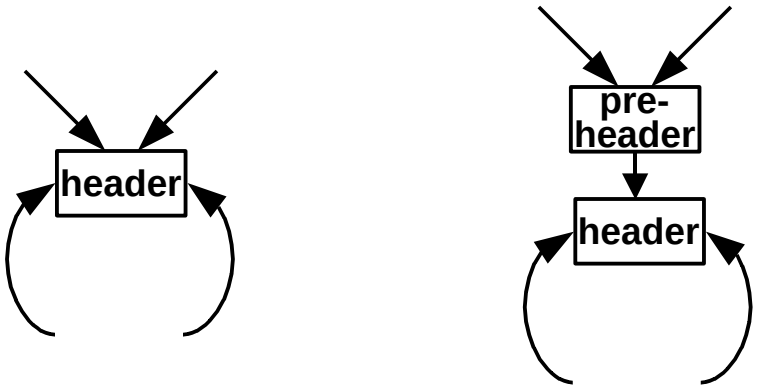




- A “natural” loop has a single entry node h $\in$ N and a tail node t $\in$ N, such that (t,h) $\in$ E; loop can be entered only through h; the loop contains h and all nodes that can reach t without going through h.

Loop Pre-Headers

- Several optimizations require that code be moved “before the header”.
- It is convenient to create a new block called the pre-header.
- The pre-header has only the header as successor.
- All edges that formerly entered the header instead enter the pre-header, with the exception of edges from inside the loop.



- See that you have a store of an expression to a temporary followed by an assignment to a variable. If the expression