ECE 5775
High-Level Digital Design Automation
Fall 2022

Control Flow Graph

Cornell University

CSL

Announcements

- Lab 2 released

- HW 1 due tomorrow

Agenda

- A typical HLS compilation flow
  - Front-end compilation and intermediate representation
- Basics of control data flow graph
  - Basic blocks
  - Control flow graph
- Dominance relation
  - Finding loops

HLS Adoption for ASIC Design

CHIPS

Why YouTube decided to make its own video chip

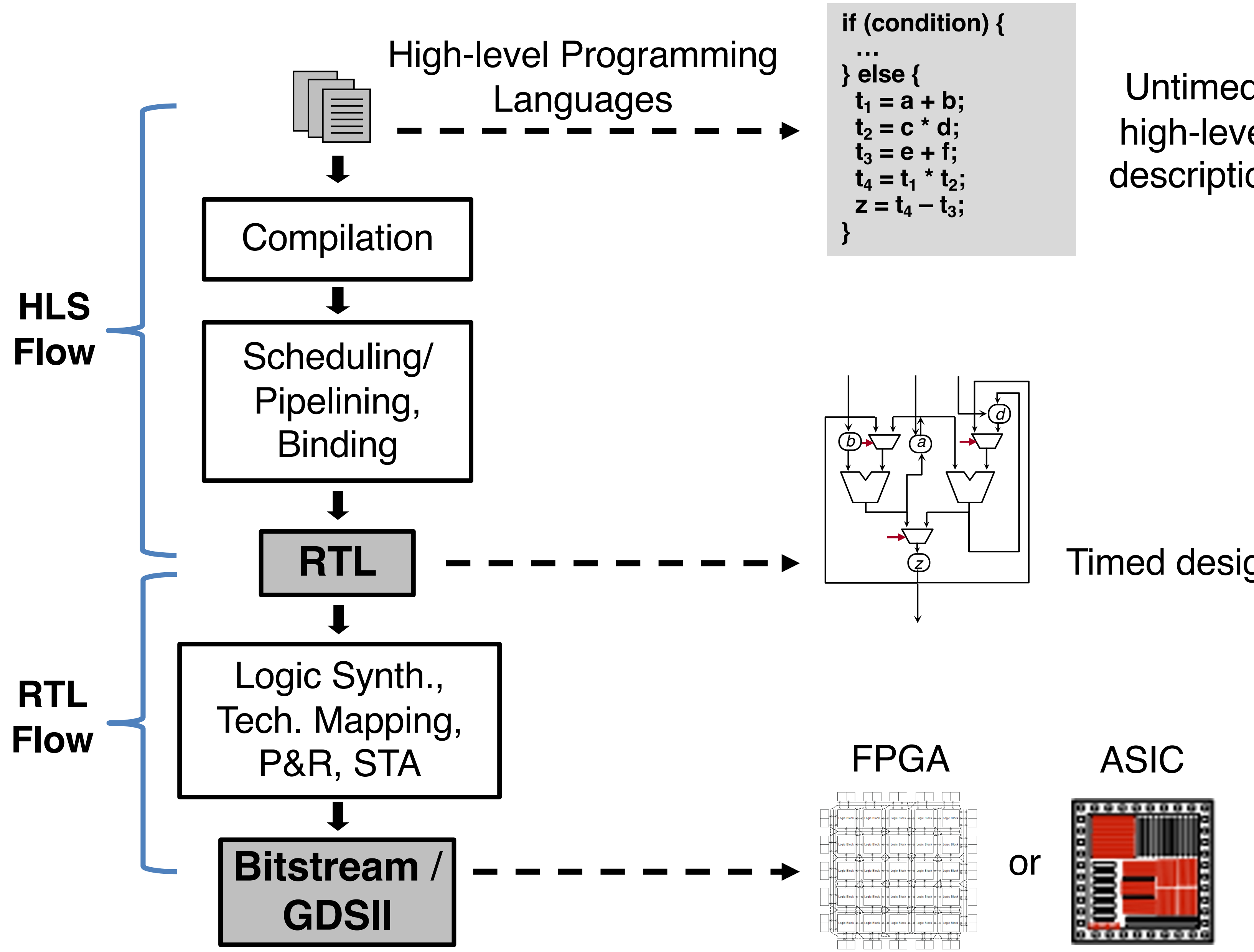
With YouTube designing its own custom chips, the company joins a growing group of big tech companies seeking to offer something unique in the data center. Operating behind the scenes, Argos made YouTube's data centers much more efficient.

YouTube Argos chip, or video-coding unit (VCU)

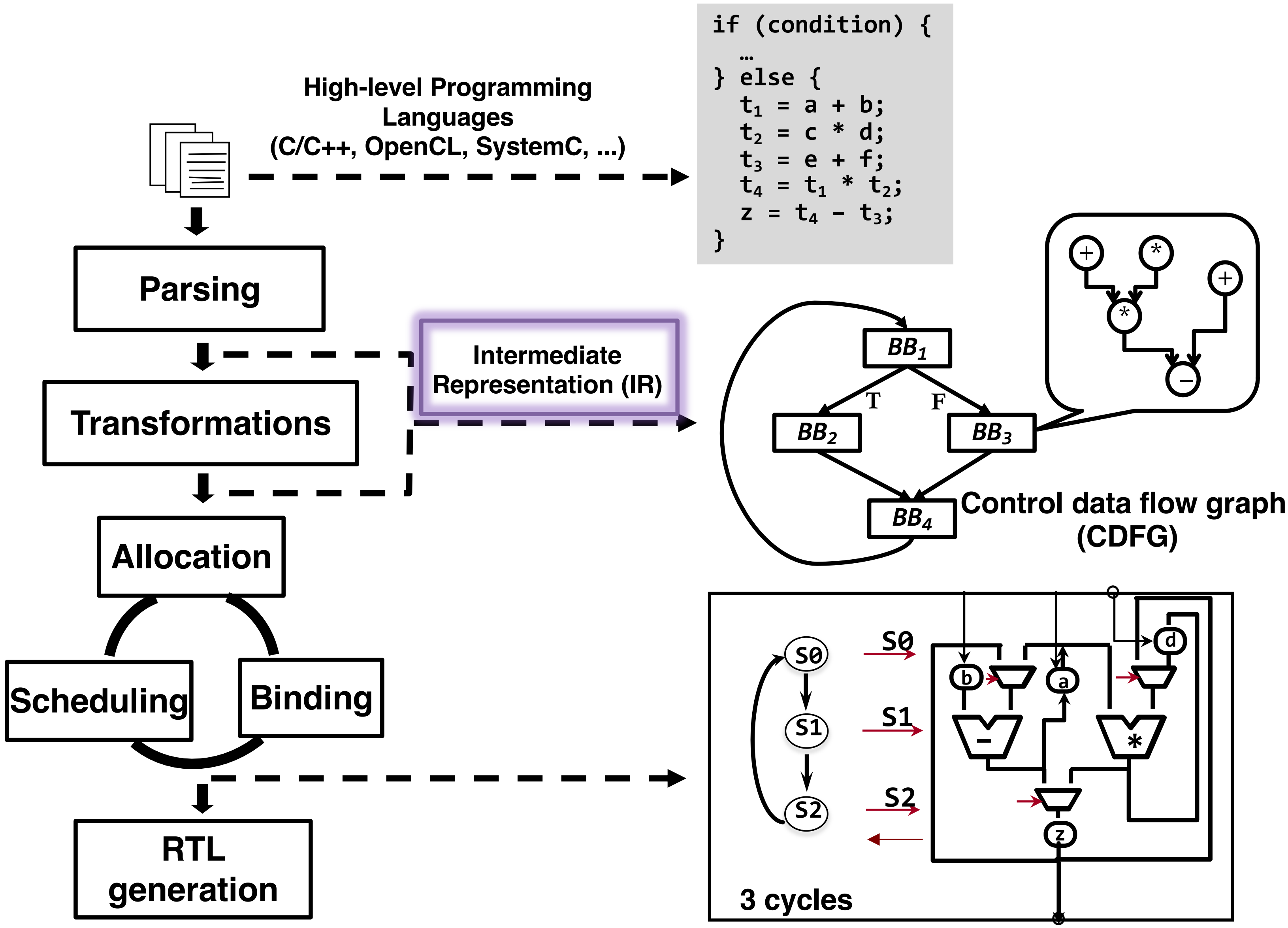
"… **Argos is a piece of hardware defined by software**, which meant that the engineers working on the chip could use what are called high-level synthesis techniques to iterate on the design much more quickly. **Google developed its own version of high-level synthesis software called Taffel that it used to help make the TPUs and the Argos processors** …"

https://www.protocol.com/enterprise/youtube-custom-chips-argos-asics (8/24/2022)

Accelerator Design Flow with HLS

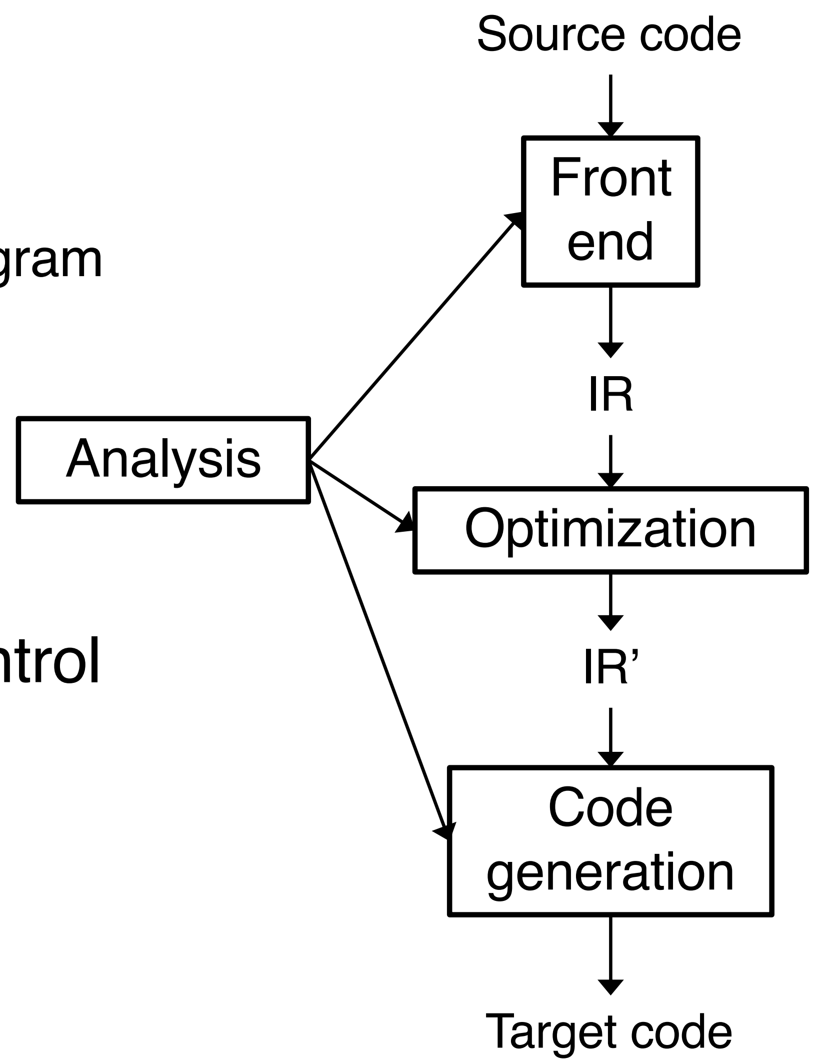


A Typical HLS Flow

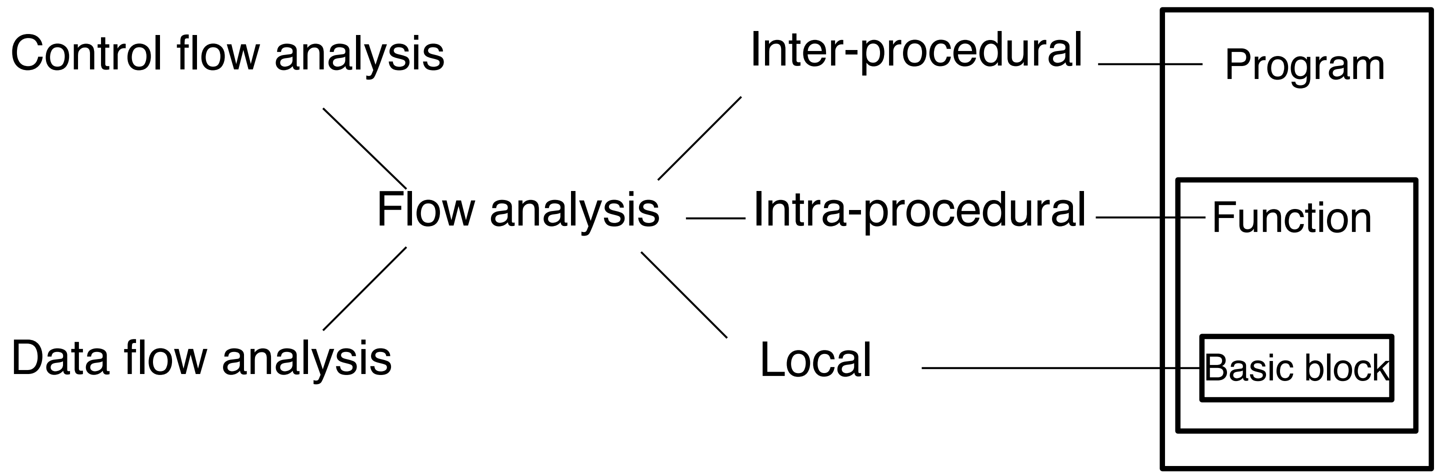


Intermediate Representation (IR)

- Purposes of creating and operating on an IR
  - Encode the behavior of the program
  - Facilitate analysis
  - Facilitate optimization
  - Facilitate retargeting
- The IR we will focus on is control data flow graph (CDFG)



Program Flow Analysis



- Control flow analysis: determine control structure of a program and build control flow graphs (**CFGs**)

- Data flow analysis: determine the flow of data values and build data flow graphs (**DFGs**)

Basic Blocks

- **Basic block**: a sequence of consecutive intermediate language statements in which flow of control can only enter at the beginning and leave at the end
  - Only the last statement of a basic block can be a branch statement and only the first statement of a basic block can be a target of a branch

Partitioning a Program into Basic Blocks

- Each basic block begins with a leader statement

- Identify leader statements (i.e., the first statements of basic blocks) by using the following rules:

  - (i) The **first statement** in the program is a leader

  - (ii) Any statement that is the **target of a branch statement** is a leader (for most intermediate languages these are statements with an associated label)

  - (iii) Any statement that **immediately follows a branch** or **return** statement is a leader

Example: Forming the Basic Blocks

```
(1)  p = 0
(2)  i = 1
(3)  t1 = 4 * i
(4)  t2 = a[t1]
(5)  t3 = 4 * i
(6)  t4 = b[t3]
(7)  t5 = t2 * t4
(8)  t6 = p + t5
(9)  p = t6
(10) t7 = i + 1
(11) i = t7
(12) if i <= 20 goto (3)
(13) j = …
```

Leader statement is:
(1) the first in the program
(2) any that is the target of a branch
(3) any that immediately follows a branch

Basic Blocks:

B1

(1) p = 0
(2) i = 1

B2

(3) t1 = 4 * i
(4) t2 = a[t1]
(5) t3 = 4 * i
(6) t4 = b[t3]
(7) t5 = t2 * t4
(8) t6 = p + t5
(9) p = t6
(10) t7 = i + 1
(11) i = t7
(12) if i <= 20 goto (3)

B3

(13) j = …

Exercise: Basic Blocks and CFG

Partition the given program into basic blocks

```
for (i = 0; i < N; i++)
   if (i > 0) A[i-1] = i;
return;
```

⇨

```
(1)   i = 0
(2)   if (i >= N) goto (7)
(3)   if (i <= 0) goto (5)
(4)   A[i-1] = i
(5)   i++
(6)   goto (2)
(7)   return
```

⇨

Leader statement is:
(1) the first in the program
(2) any that is the target of a branch
(3) any that immediately follows a branch

Control Flow Graph (CFG)

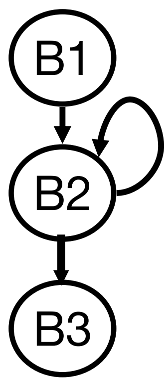
- A **control flow graph** (CFG), or simply a flow graph, is a directed graph in which:
  - (i) the nodes are basic blocks; and
  - (ii) the edges are induced from the possible flow of the program

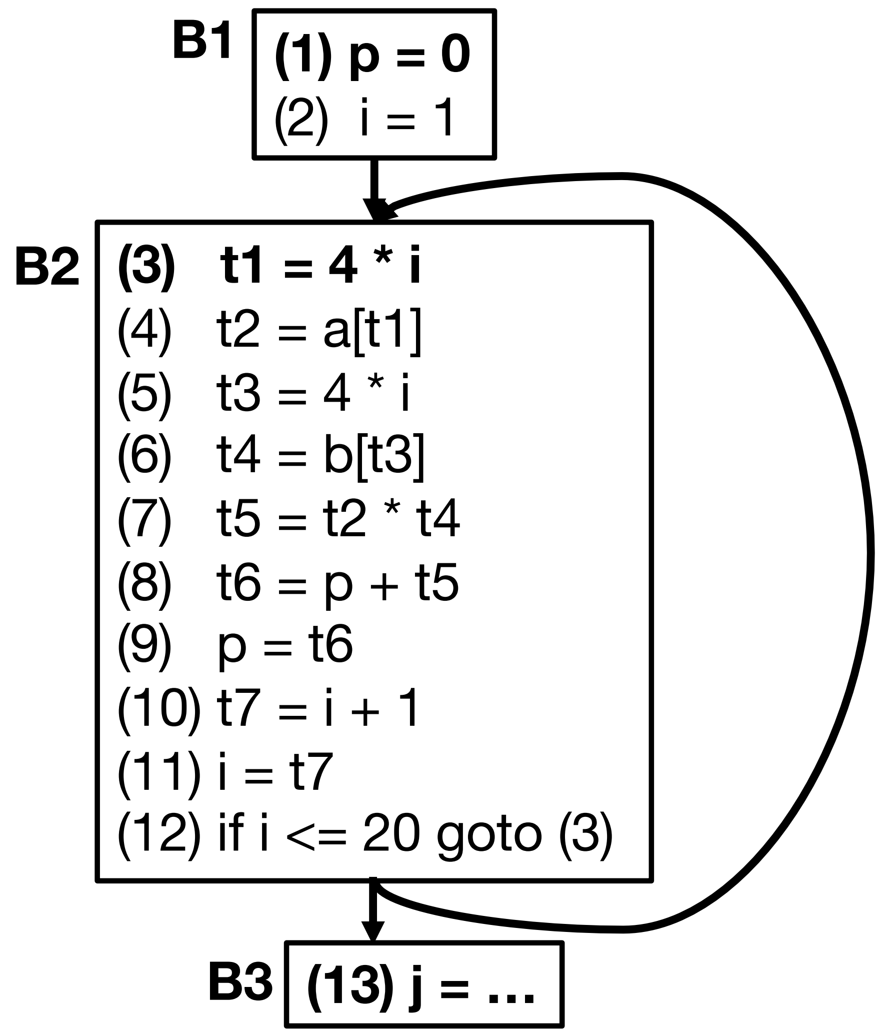
- The basic block whose leader is the first intermediate language statement is called the **entry node**

- In a CFG we assume no information about data values
  - an edge in the CFG means that the program **may** take that path

Example: Control Flow Graph Formation

CFG





Dominators

- A node p in a CFG ***dominates*** a node q if every path from the entry node to *q* goes through p. We say that node p is a ***dominator*** of node q

- The ***dominator set*** of node q**, DOM(**q**),** is formed by all nodes that dominate q
  - Each node dominates itself by definition; thus q $\in$ **DOM(**q**)**

Dominance Relation

- **Definition:** Let $G = (V, E, s)$ denote a CFG, where

  V : set of nodes

  E : set of edges

  s : entry node and

  let $p \in V$, $q \in V$

  - p **dominates** q, written $p \leq q$
    - $p \in DOM(q)$
  - p **properly (strictly) dominates** q, written $p < q$ if $p \leq q$ and $p \neq q$
  - p **immediately** (or directly) **dominates** q,
    written $p <_d q$ if $p < q$ and there is no $t \in V$ such that $p < t < q$
    - $p = IDOM(q)$

Example: Dominance Relation

- **Dominator sets:**

  DOM(1) = {1}

  DOM(2) = {1, 2}

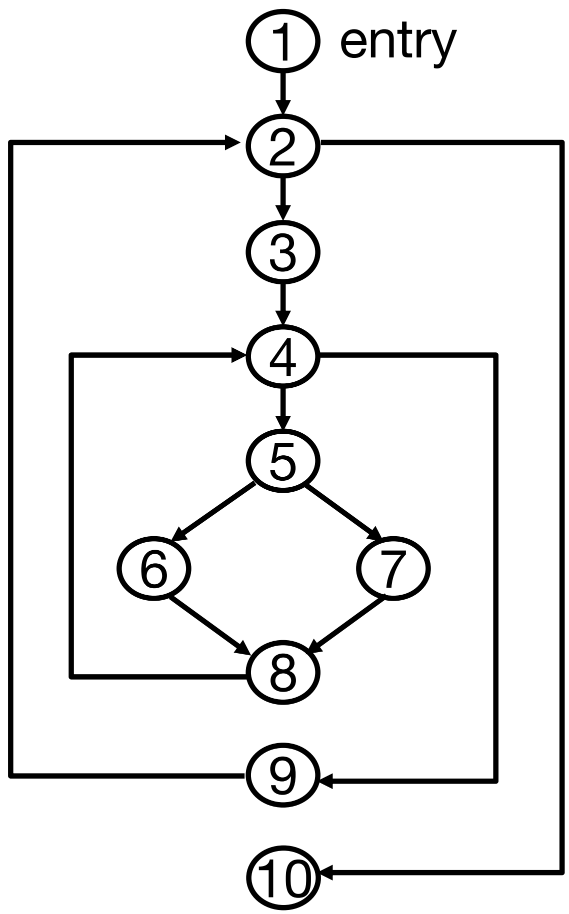
  DOM(3) = {1, 2, 3}

  DOM(10) = {1, 2, 10}

- **Immediate domination:**

  $1 <_d 2$, $2 <_d 3$, …

  IDOM(2) = 1, IDOM(3) = 2 …

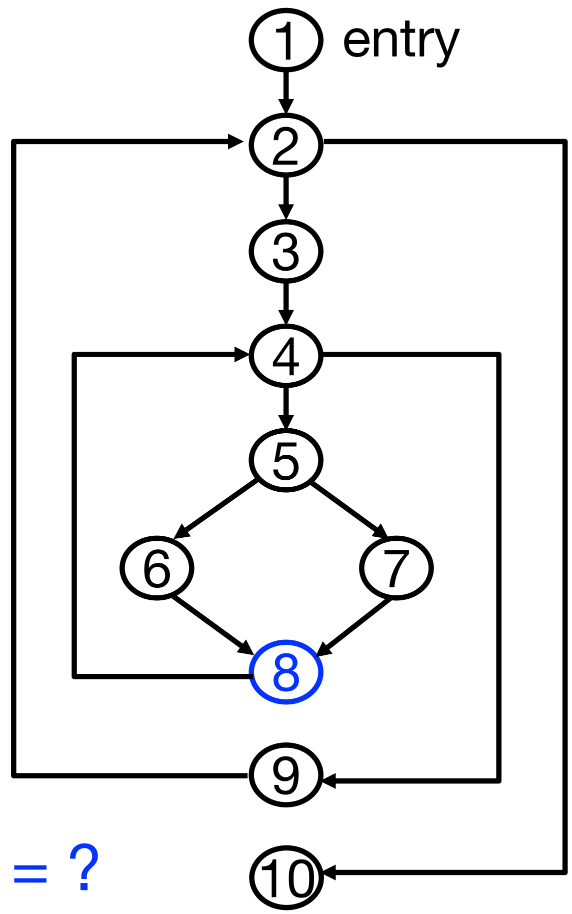


Dominance Question

Assume that node *P* is an immediate dominator of node *Q*

Question: Is *P* necessarily an immediate predecessor of *Q* in the CFG?

Answer: **NO**



IDOM(8) = ?

Identifying Loops

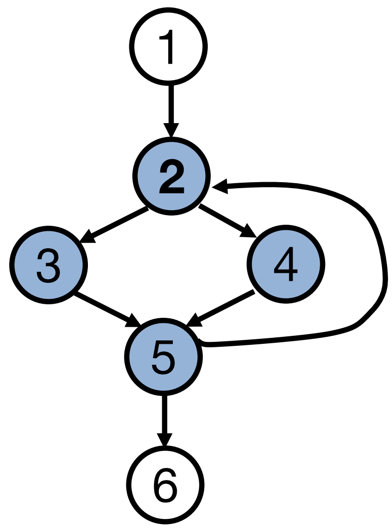
- **Motivation**: Programs spend most of the execution time in loops, therefore there is a larger payoff for optimizations that exploit loop structure

- **Goal**: Identify loops in a CFG, not sensitive to syntax of the input language
  - Create a uniform treatment for program loops written using different syntactical constructs (e.g., while, for, goto)

- **Approach:** Use a general approach based on analyzing graph-theoretical properties of the CFG

Loop Definition

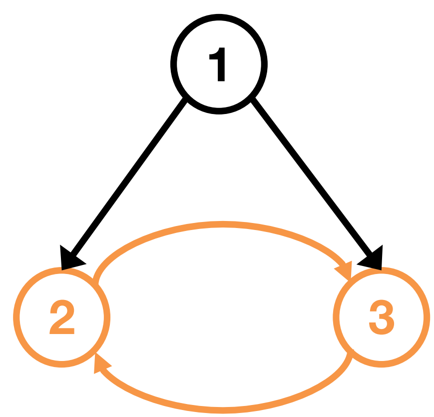
- Definition of a (natural) loop
  - A **strongly connected component** (SCC) of the CFG, with a single-entry point called the **header** which dominates all nodes in the SCC



- All nodes in blue form a loop, which is an SCC
- Node 2 is the header

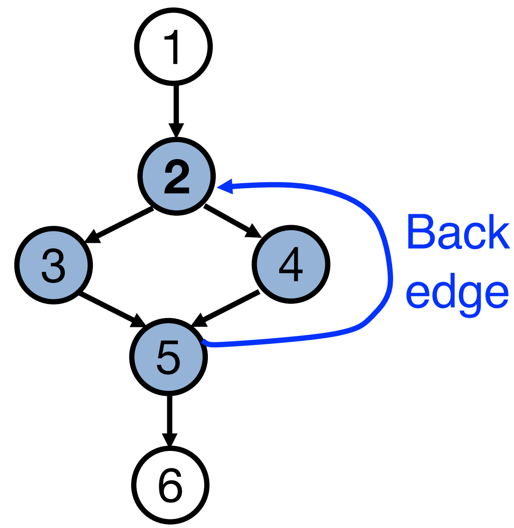
Is it a Loop?

- Question: In the CFG below, nodes 2 and 3 form an SCC; but **do they form a loop**?

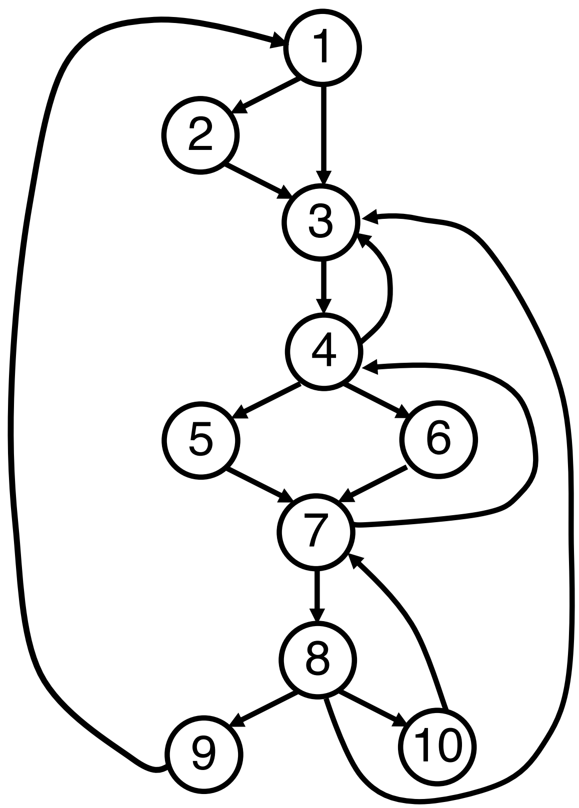


Finding Loops

- Loop identification algorithm
  - Find an edge B→H where H dominates B
    - This edge is called a **back edge**
  - Find all nodes that
    (1) are dominated by H, and
    (2) can reach B via nodes dominated by H
  - Add these nodes to the loop
    - H and B are naturally included

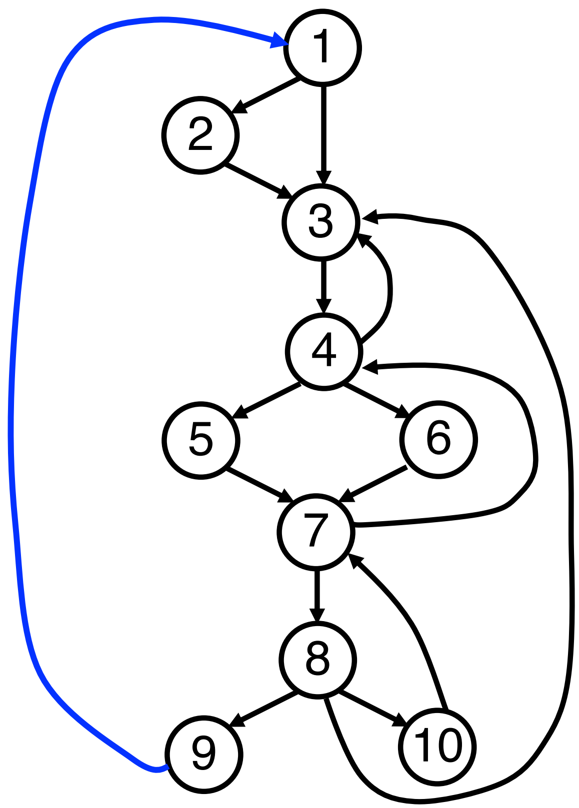


Finding Loops



Find all back edges in this graph and the natural loop associated with each back edge

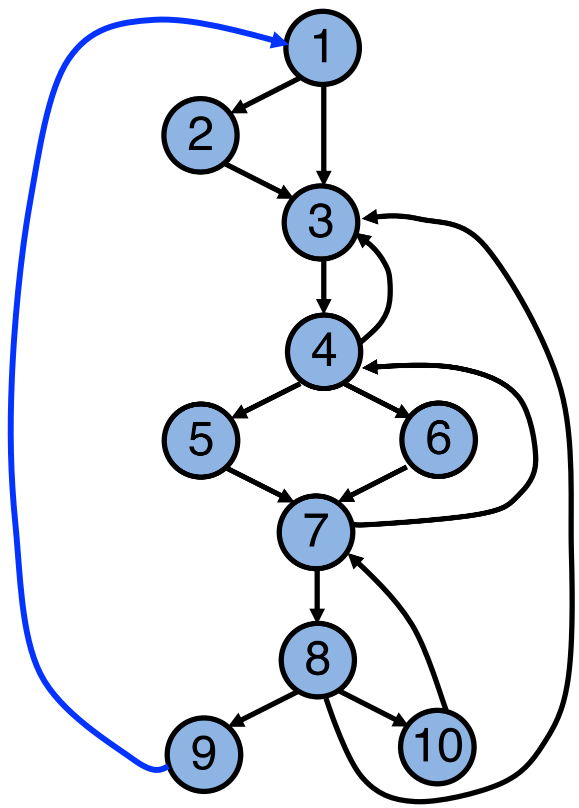
Finding Loops (1)



Find all back edges in this graph and the natural loop associated with each back edge

(9,1)

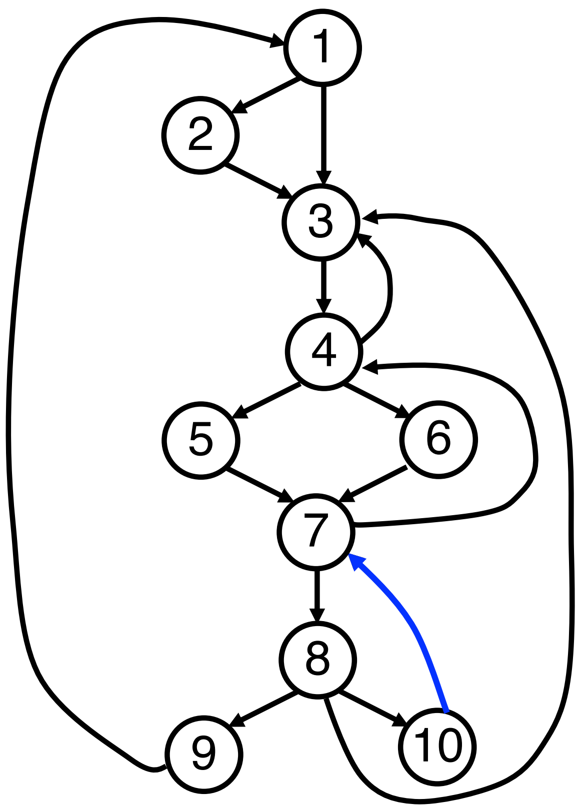
Finding Loops (1)



Find all back edges in this graph and the natural loop associated with each back edge

(9,1)	Entire graph

Finding Loops (2)



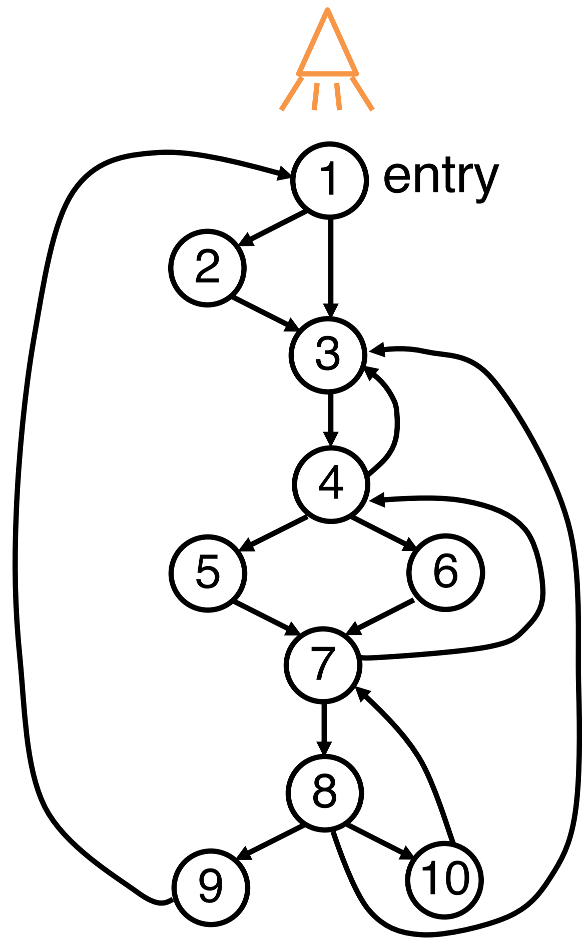
Find all back edges in this graph and the natural loop associated with each back edge

(9,1) Entire graph
(10,7)

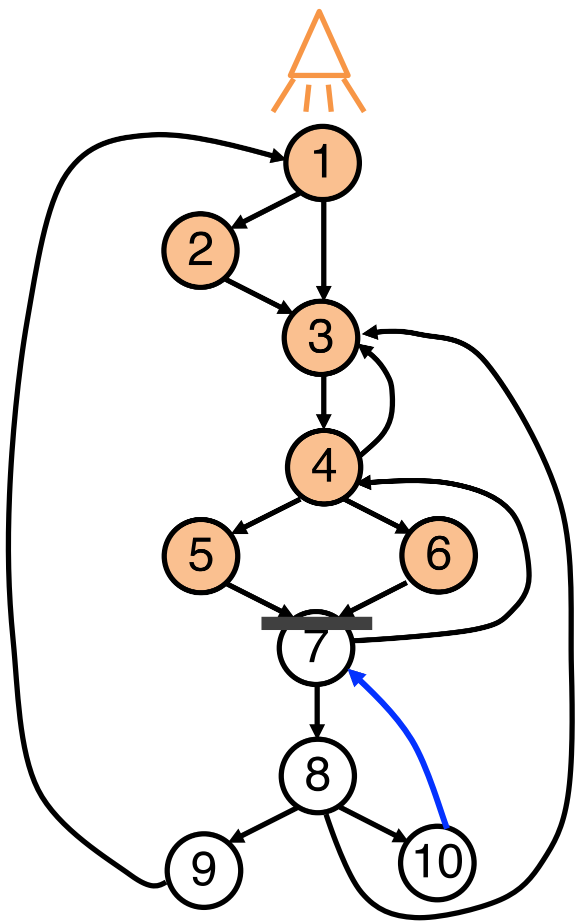
Intuition of Dominance Relation

Imagine a source of light
at the entry node, and that
the edges are optical fibers

To find which nodes are
dominated by a given node,
place an opaque barrier at that
node and observe which nodes
become dark



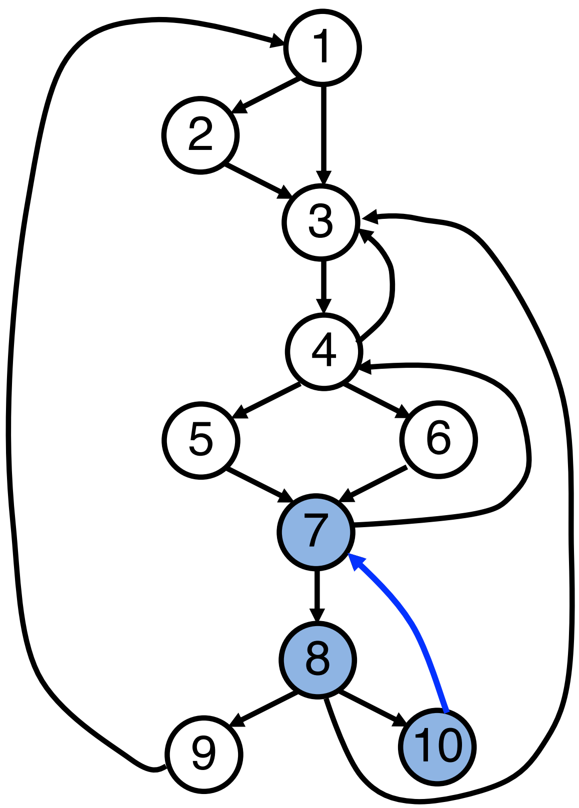
Finding Loops (2)



Find all back edges in this graph and the natural loop associated with each back edge

(9,1)	Entire graph
(10,7)	

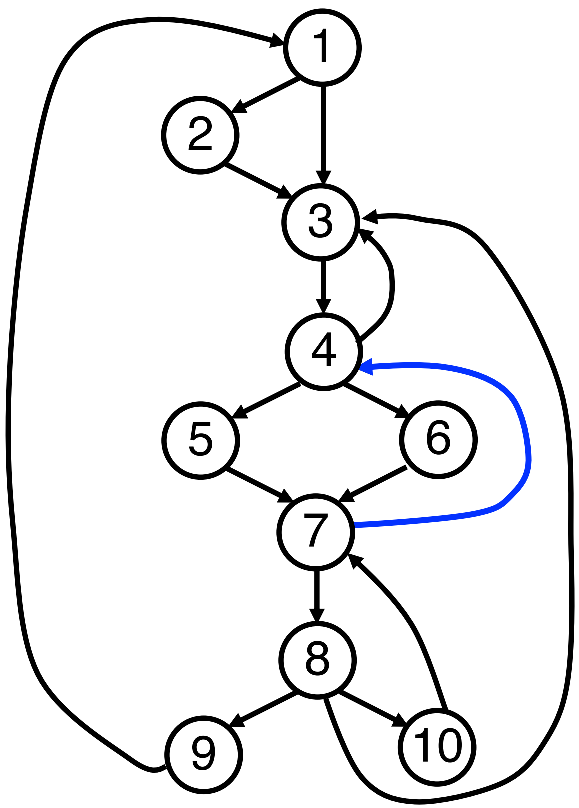
Finding Loops (2)



Find all back edges in this graph and the natural loop associated with each back edge

(9,1)	Entire graph
(10,7)	{7,8,10}

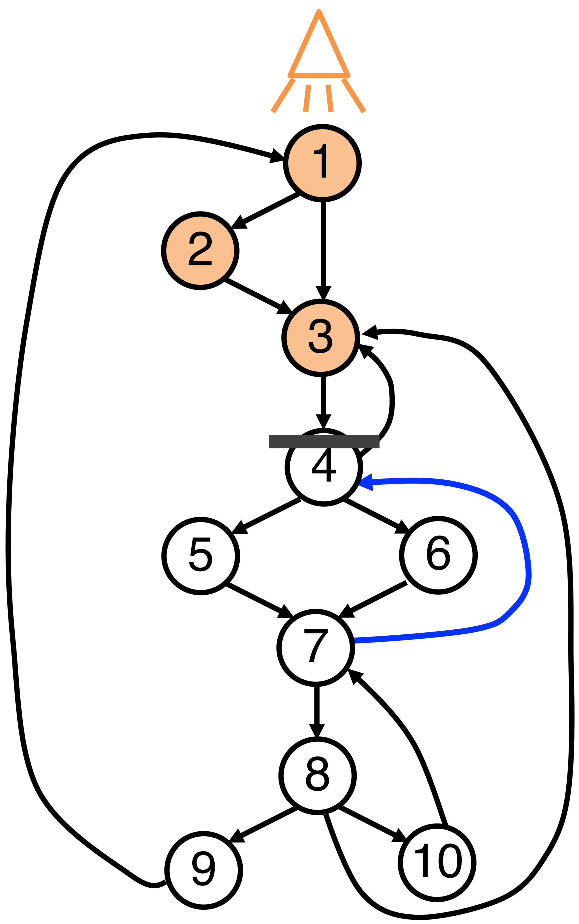
Finding Loops (3)



Find all back edges in this graph and the natural loop associated with each back edge

(9,1)	Entire graph
(10,7)	{7,8,10}
(7,4)	

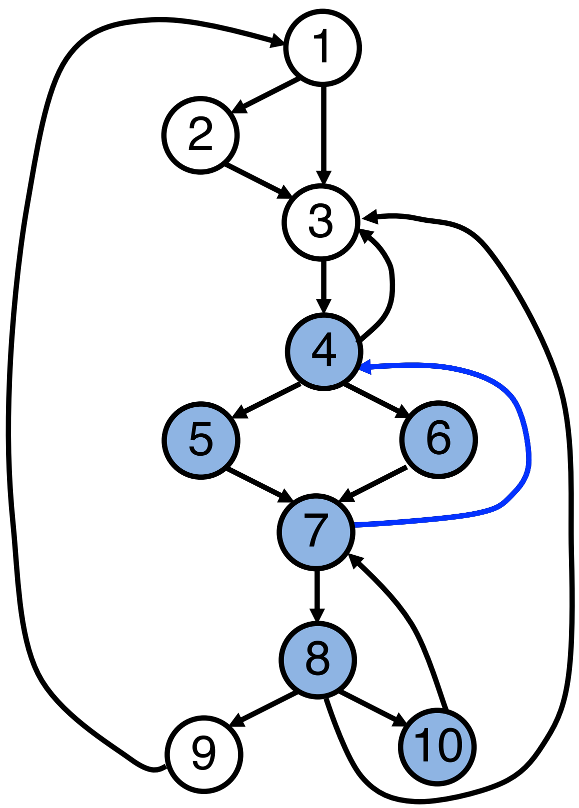
Finding Loops (3)



Find all back edges in this graph and the natural loop associated with each back edge

(9,1)	Entire graph
(10,7)	{7,8,10}
(7,4)	

Finding Loops (3)



Find all back edges in this graph and the natural loop associated with each back edge

(9,1)	Entire graph
(10,7)	{7,8,10}
(7,4)	{4,5,6,7,8,10}

Summary

- Basic Blocks
  - Group of statements that execute atomically
- Control Flow Graphs
  - Model the control dependences between basic blocks
- Dominance relations
  - Shows control dependences between BBs
  - Used to determine natural loops

Next Lecture

- Static single assignment (SSA)

Acknowledgements

- These slides contain/adapt materials developed by
  - Forrest Brewer (UCSB)
  - Ryan Kastner (UCSD)
  - Prof. José Amaral (Alberta)