CSE 501
Principles and Applications of Program Analysis

Alvin Cheung

Spring 15

Welcome to CSE 501!

The Cast

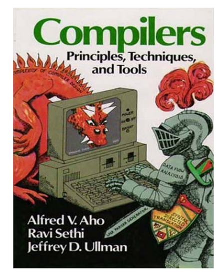
Instructor

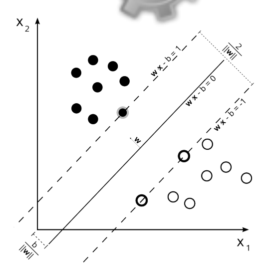
Alvin Cheung
CSE 530

$$\frac{\begin{array}{c}[\![\langle Q, D, \sigma, h\rangle, e]\!] \rightarrow \langle Q', D', \sigma, h'\rangle, (\sigma', e) \\ \text{force}(Q', D', (\sigma', e)) \rightarrow Q'', D'', \text{False} \\ [\![\langle Q'', D'', \sigma, h'\rangle, s_2]\!] \rightarrow \langle Q''', D'', \sigma', h''\rangle\end{array}}{[\![\langle Q, D, \sigma, h\rangle, \text{if}(e) \text{ then } s_1 \text{ else } s_2]\!] \rightarrow \langle Q''', D''', \sigma', h''\rangle}$$

$$\frac{[\![\langle Q, D, \sigma, h\rangle, s]\!] \rightarrow \langle Q', D', \sigma', h'\rangle}{[\![\langle Q, D, \sigma, h\rangle, \text{while(True) do } s]\!] \rightarrow \langle Q', D', \sigma', h'\rangle}$$

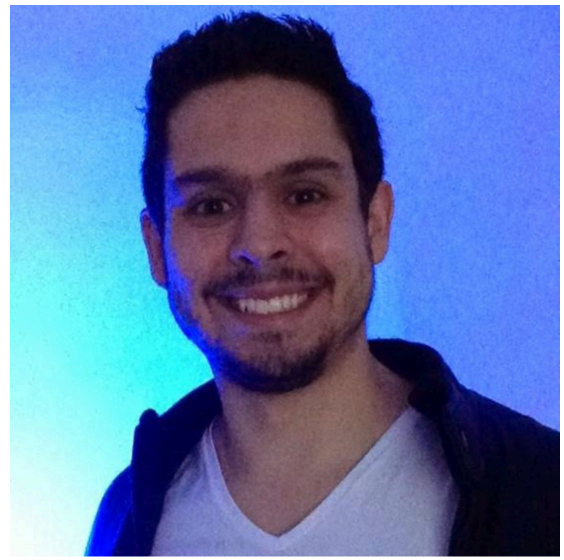
$$\frac{\begin{array}{c}[\![\langle Q, D, \sigma, h\rangle, e]\!] \rightarrow \langle Q', D', \sigma, h'\rangle, (\sigma', e) \\ \text{force}(Q', D', (\sigma', e)) \rightarrow Q'', D'', v \\ \text{update}(D'', v) \rightarrow D''' \\ \forall id \in Q'' \;.\; Q'''[id] = \begin{cases} D'''[Q''[id].s] & \text{if } Q''[id].rs = \emptyset \\ Q''[id].rs & \text{otherwise} \end{cases}\end{array}}{[\![\langle Q, D, \sigma, h\rangle, W(e)]\!] \rightarrow \langle Q''', D''', \sigma, h'\rangle}$$





TA Extraordinaire

Andre Baixo
Office hours: TBD



You!

Course Communication

- Discussion board
  - HW help
  - Find project partners

- Course website:
  `courses.cs.washington.edu/501`

- Email: `cse501-staff@cs.washington.edu`

Course Goals

- What are the techniques used to understand programs?
  - Mix of classical and recent advances
- What can we use these techniques for?
  - Variety of applications across different domains
- How do we build tools that utilize such techniques?

Course Goals

- How to do research?
  - How to choose problems
  - How to devise solutions
  - How to evaluate
  - How to report results

Course Non-Goals

- How to build a compiler from scratch
  - Check out CSE 401
- What are all the compiler optimizations out there?
  - Check out list of references on website
- Cover all research topics in program analysis
  - 35 years of PLDI but we only have 10 weeks!

Class Format

- Two class meetings per week
  - Tuesday and Thursday 11am – 12:20 pm
  - Here!
- Occasional HW help and project feedback sessions

Class Format

- We will discuss 1-2 research papers during each class meeting
  - Please read them beforehand
  - We ask you to write a small commentary before class to share with everyone
  - Be prepared to ask questions!

Grading

- Programming assignments (30%)
  - Get to know available tools out there
  - No late days
- Project (50%)
  - Open-ended: find problems in your research area
  - Work with a partner
  - We will provide you with potential ideas
  - Project milestones, end-of-quarter presentation, final report
- Paper summaries (20%)
  - Submit paper summary 24-hrs before lecture
  - See details on course website

Course Topics

- Dataflow frameworks
- Abstract interpretation
- Domain-specific languages
- Program verification
- Dynamic analysis

Course Topics

- Dataflow frameworks & abstract interpretation
  - Pointer analysis
  - Compiler optimizations
  - Information flow
  - Detecting malware
- Domain-specific languages
  - Parallel programming
  - High-performance computing
  - New hardware

Course Topics

- Program verification
  - Finding program invariants
  - Provably-correct compilers

- Dynamic analysis
  - Program testing
  - Model checking

- Compiler construction

Prerequisites

- Coding

- Data structures
- Mathematical logic

- [Optional] Knowledge about compilers

Now the fun begins…

Why understand programs?

- We all write code!
- It's good to get some understanding about what we are coding
- It's good to develop a *formal framework* for understanding programs
- It's good to have somebody else do this for us, perhaps automatically

List of software bugs

From Wikipedia, the free encyclopedia

Many software bugs are merely annoying or inconvenient but some can have extremely serious consequences – either financially or as a threat to human well-being. The following is a list of notable software bugs with significant consequences:

Space exploration

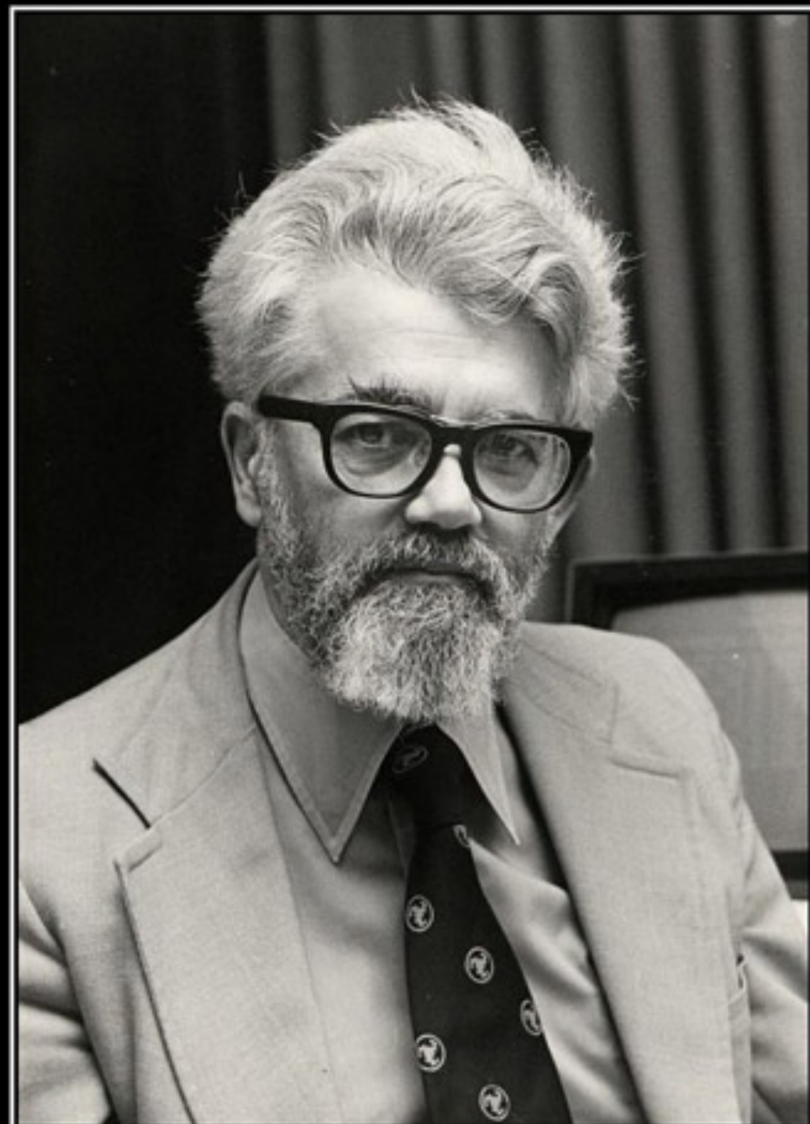
In 1997, the Mars Pathfinder mission was jeopardised by a bug in concurrent software shortly after the rover landed, which had not been found in preflight testing because it only occurred in certain unanticipated heavy-load conditions.[5] The problem, which was identified and corrected from Earth, was due to computer resets caused by priority inversion.[6][7]

Medical

- A bug in the code controlling the Therac-25 radiation therapy machine was directly responsible for at least five patient deaths in the 1980s when it administered excessive quantities of X-rays.[13][14][15]
- A Medtronic heart device was found vulnerable to remote attacks in March 2008.[16]

Video gaming

The Corrupted Blood incident was a software bug in World of Warcraft that caused a status ailment, that was supposed to be locally restricted to a certain level of the game, to be set free, affecting all players everywhere in the virtual game world. This caused players to avoid crowded places in-game, just like in a "real world" epidemic, and the bug became the centre of some academic research on the spread of infectious diseases.[33]

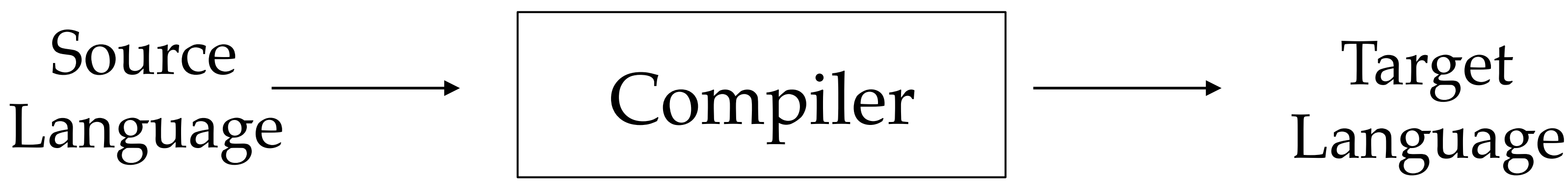


PROGRAMMING

You're Doing It Completely Wrong.

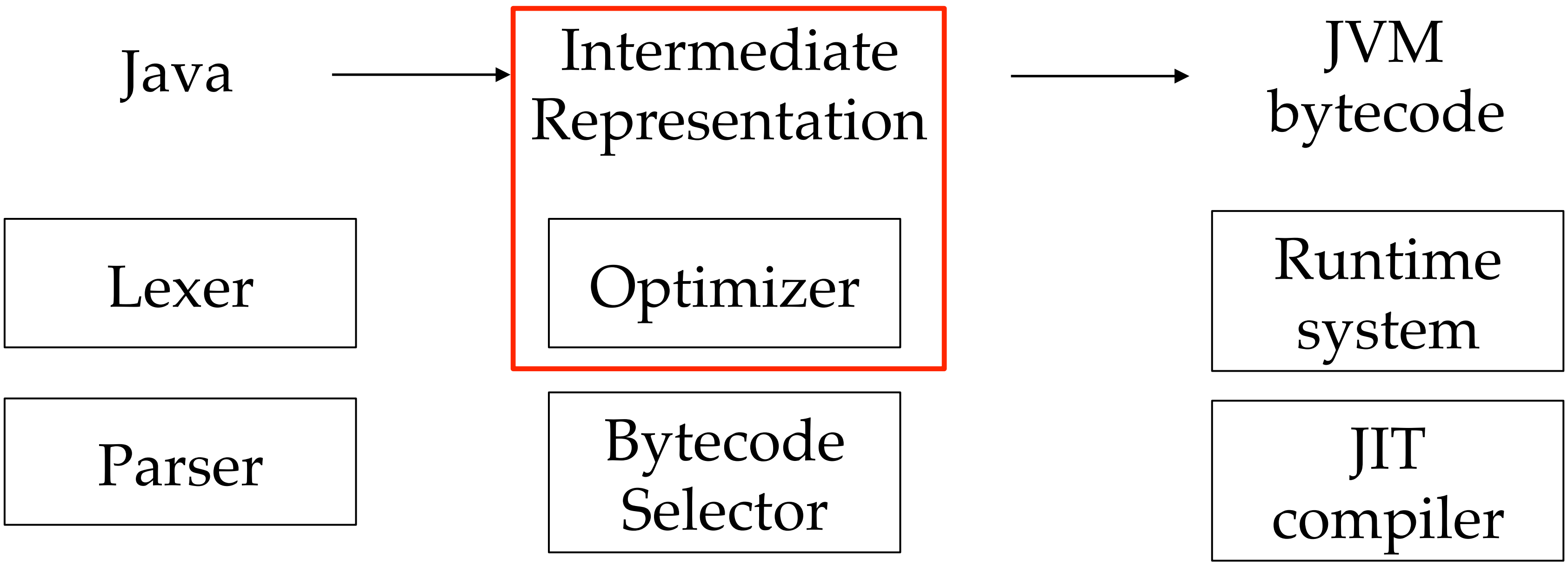
A Classical Example: Compilers

A 50,000 ft view:



A Classical Example: Compilers

A 10,000 ft view:



[See CSE 401 for details]

Optimizations

- Dead code elimination
- Partial redundancy elimination
- Function inlining
- Strength reduction
- Loop transformations
  - Hoisting
  - Unrolling
  - Vectorizing
- Constant propagation

Dataflow Analysis!!

Intermediate Representation

Optimizer

Beyond compilers

- Program correctness
- Security breaches
- Have programs write themselves

Program representation

```
int pow (int a, int n) {
  int p = 1;
  for (int i = 0; i < n; ++i)
    p *= a;
  return p;
}
```

Program representation

```
int pow (int a, int n) {
  int p = 1;
  for (int i = 0; i < n;
       ++i)
    p *= a;
  return p;
}
```

```
p = 1
```

```
i = 0
```

```
i < n
```

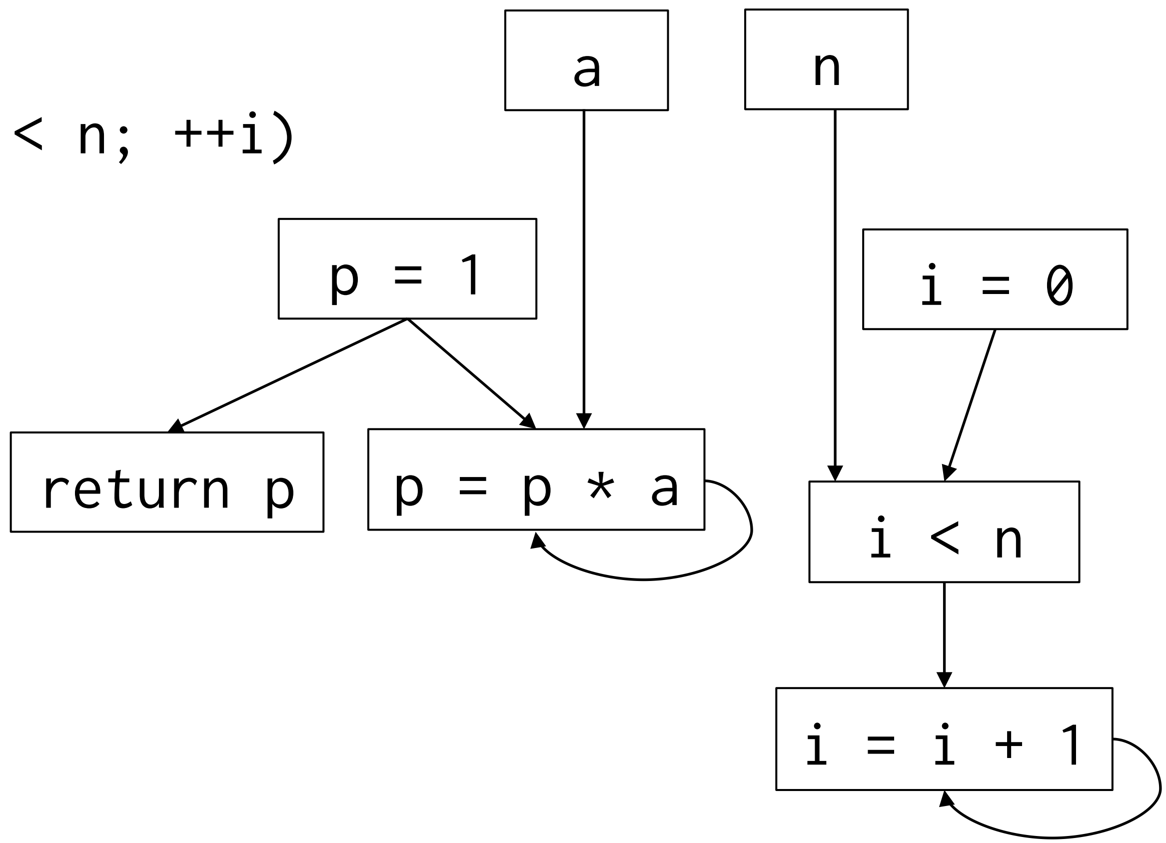
```
i = i + 1
```

```
p = p * a
```

```
return p
```

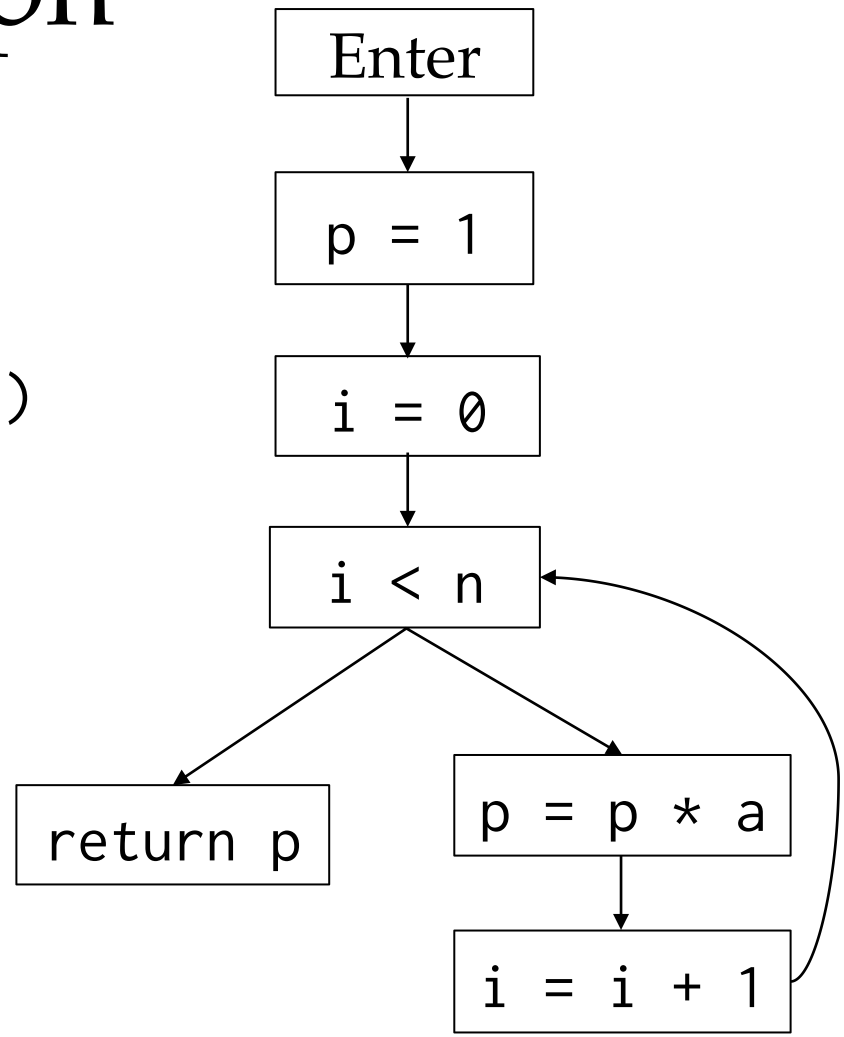
Data-flow graph

```
int pow (int a, int n) {
  int p = 1;
  for (int i = 0; i < n; ++i)
    p *= a;
  return p;
}
```



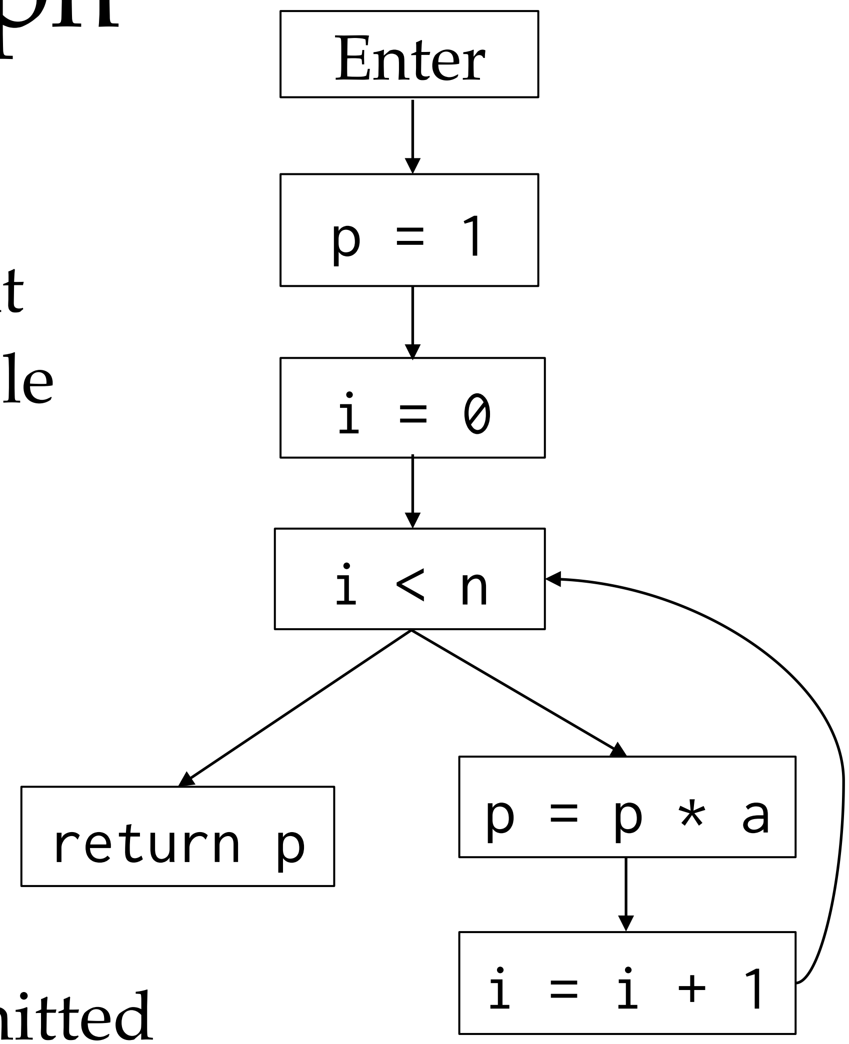
Control-flow graph

```
int pow (int a, int n) {
  int p = 1;
  for (int i = 0; i < n; ++i)
    p *= a;
  return p;
}
```



Control-flow graph

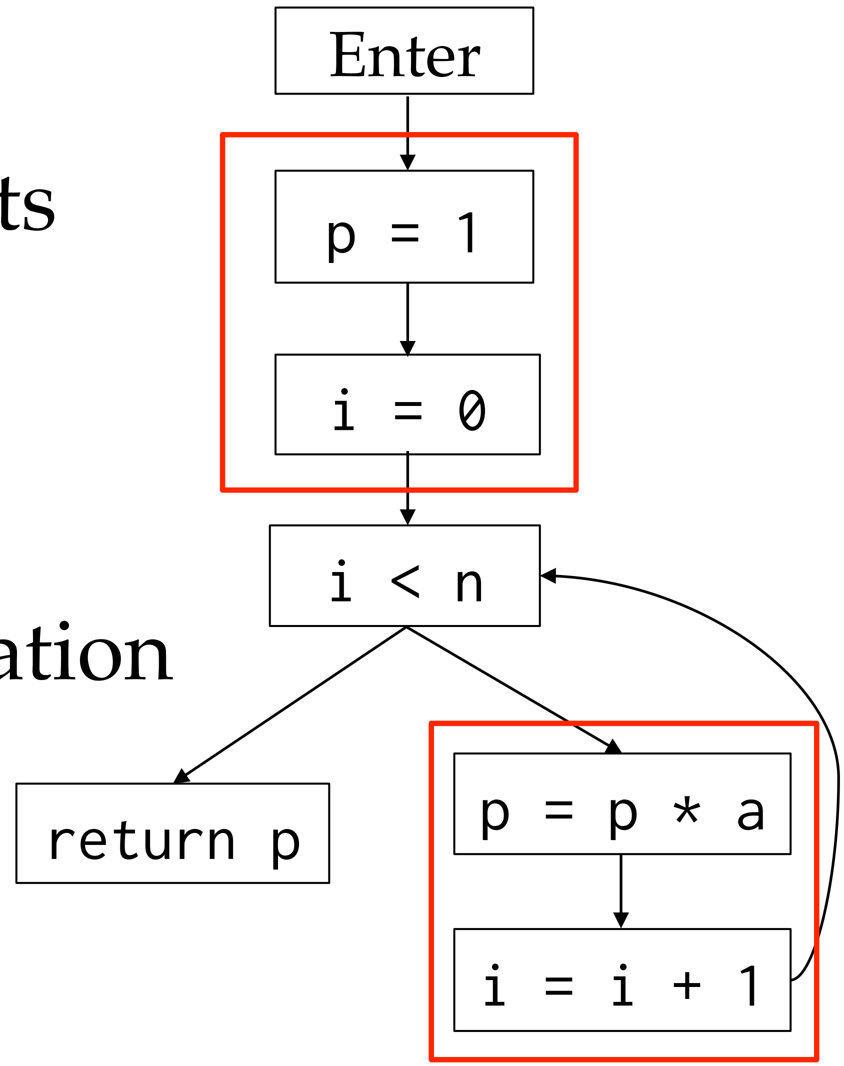
- Directed graph
  - Each node is a statement
  - Edges represents possible flow of control
- Statements
  - Assignments
  - Branches
  - Enter / return
  - Declarations usually omitted



Basic blocks

- Sequence of statements with only one entry and exit point

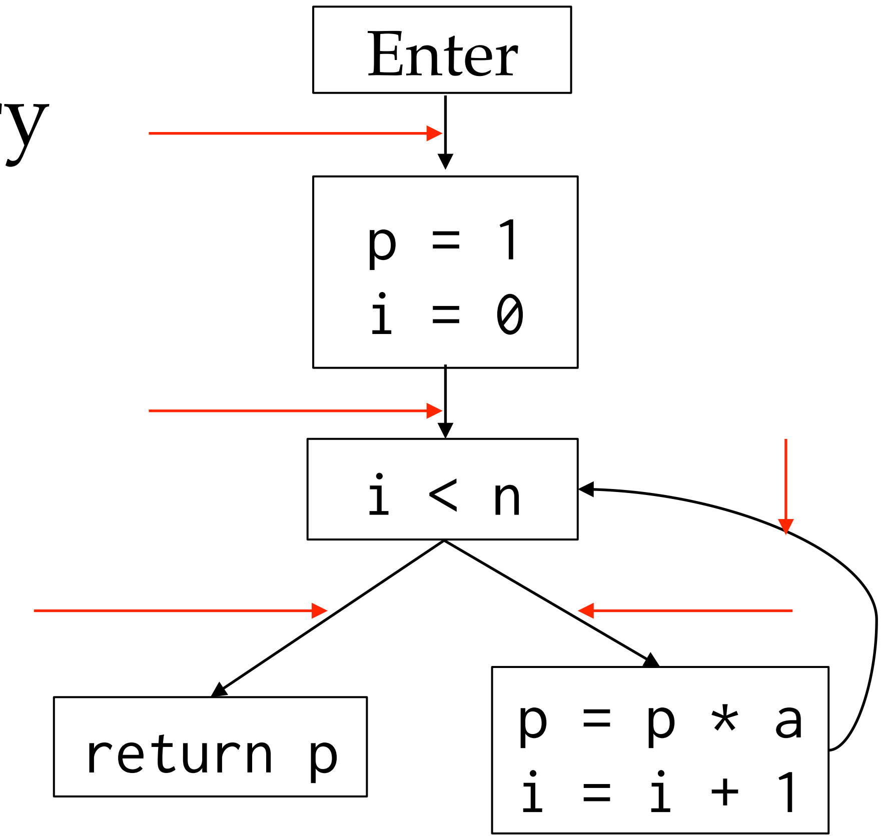
- Condensed representation of statements



Program point

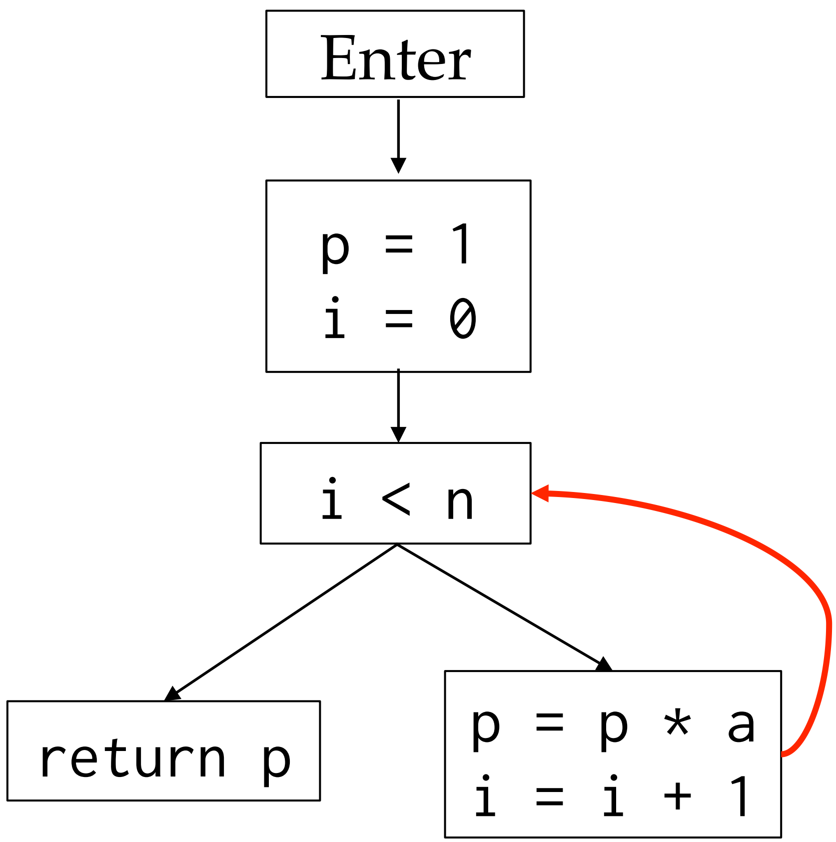
- Every statement entry and exit

- Program behavior at each program point

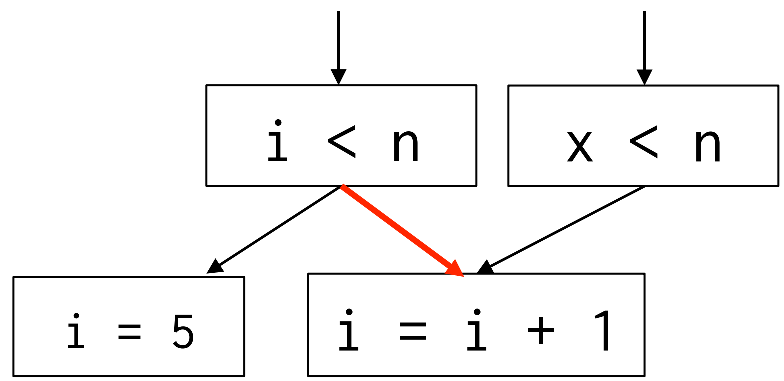


Special edges

- Back edge
  - Points to a block that has been traversed



- Critical edge
  - Edge that is neither the only edge leaving source nor entering target



Summary

- We will study techniques to understand code
- Not (just) a compiler class!
- Many connections to programming languages, systems, security, architecture etc
- [Programming systems quals for grad students]

- Next time: dataflow!