Virtual Machines

CS842:
Virtual Machines for Dynamic Languages

Course

- This is a coding course
- You will modify the implementation of a virtual machine and garbage collector
- +Short presentation of one relevant paper
- https://the.gregor.institute/t/c/

Reqs and grading

- Compilers background a must
- OS background helpful
- `((struct GC_Pool *) (((size_t) &p) & 0xFFFFF000))`
  - 50%: Projects (code)
  - 25%: Presentation
  - 25%: Final (code or paper or…?)

Schedule

- Because the end of this course will be student presentations, depends on how many students…
- Basically: We'll talk JIT's for a while, GC for a while, then presentations
- (It's an 800-level course; it's not THAT well organized!)

Schedule approximation

- January and February: Projects
- March: Presentations, final
- Project 1’s code is posted, will post the text once I’ve explained the relevant material here

Software

- Minimal “learning” JIT provided: SDyn
  - Subset-ish of JavaScript, with all its foibles
- Uses smallish “learning” GC: GGGGC
  - The important, VM-y part is the interaction between the two!

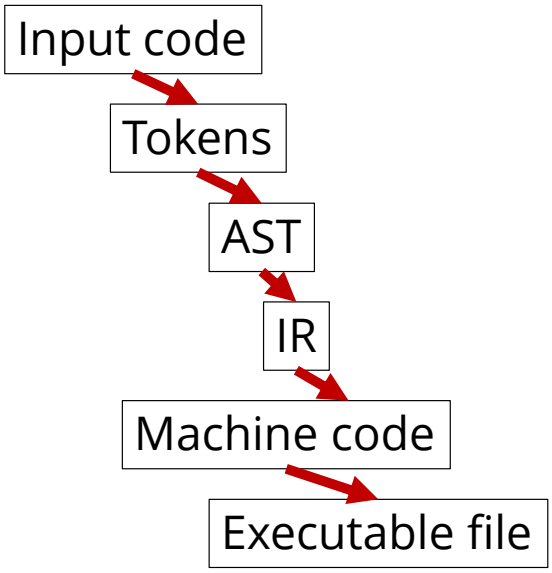
Note

- I previously taught JIT's and GC's as two separate courses
- I'm attempting to mix a bit of GC's into VM's here
- … it might be a bit rough

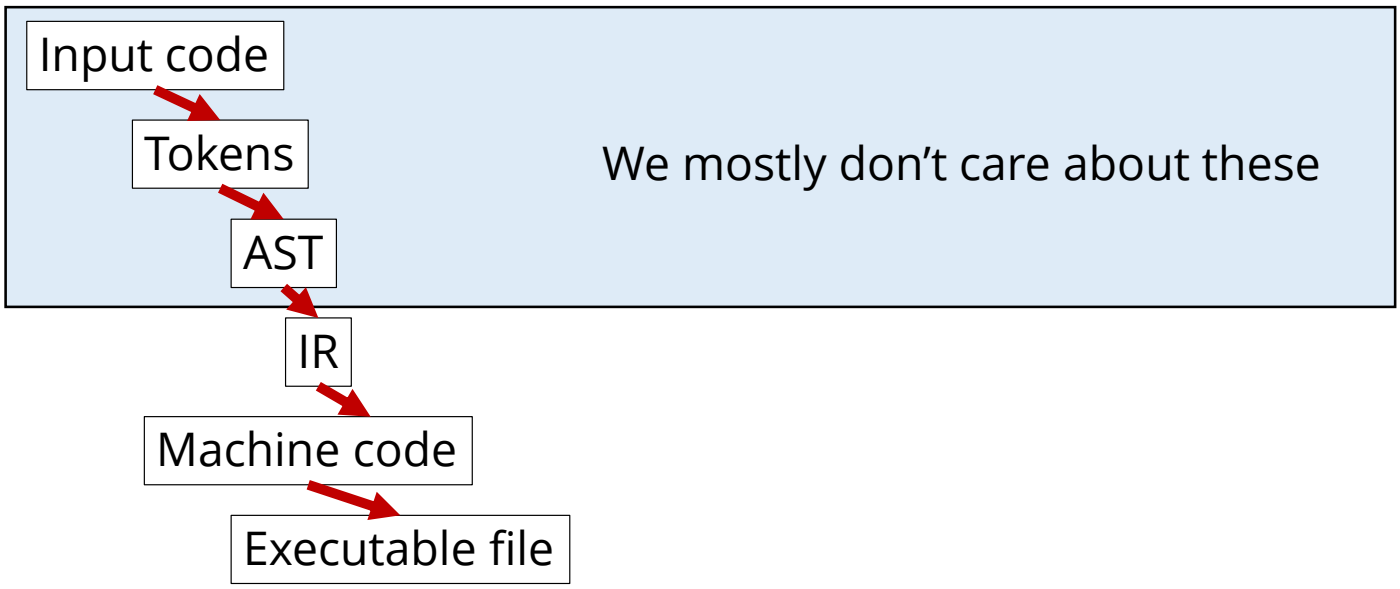
JIT background

Compilation, dynamic languages, types, the JIT concept

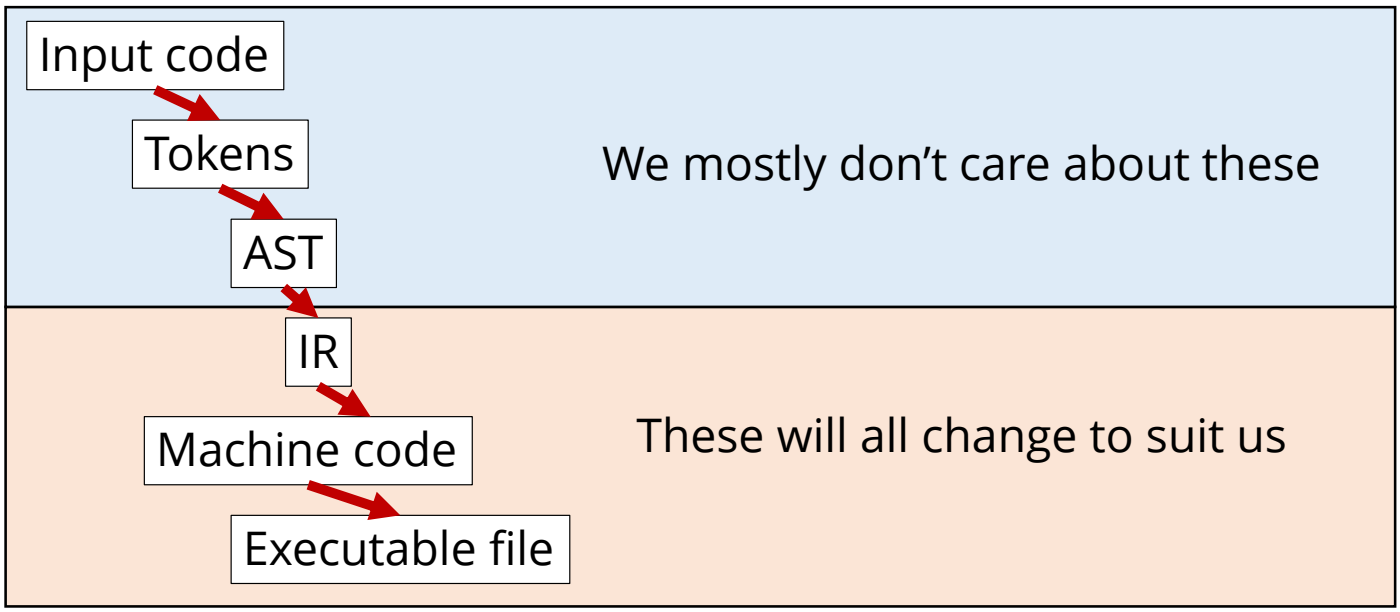
Static compilation



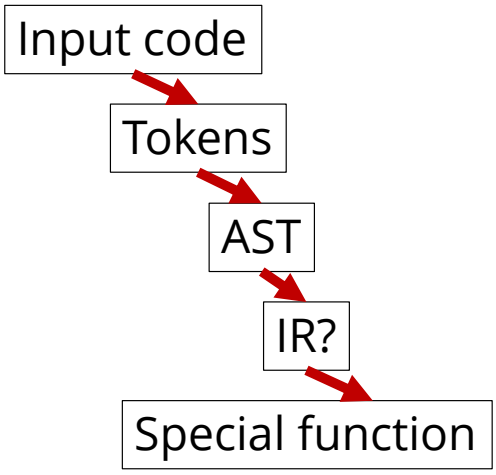
Static compilation



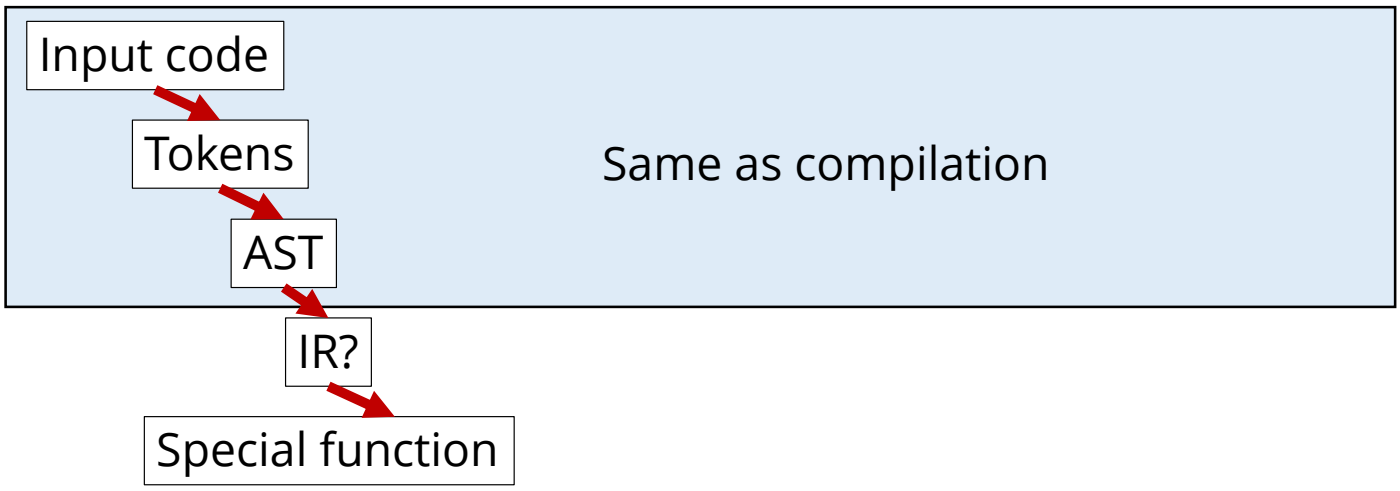
Static compilation



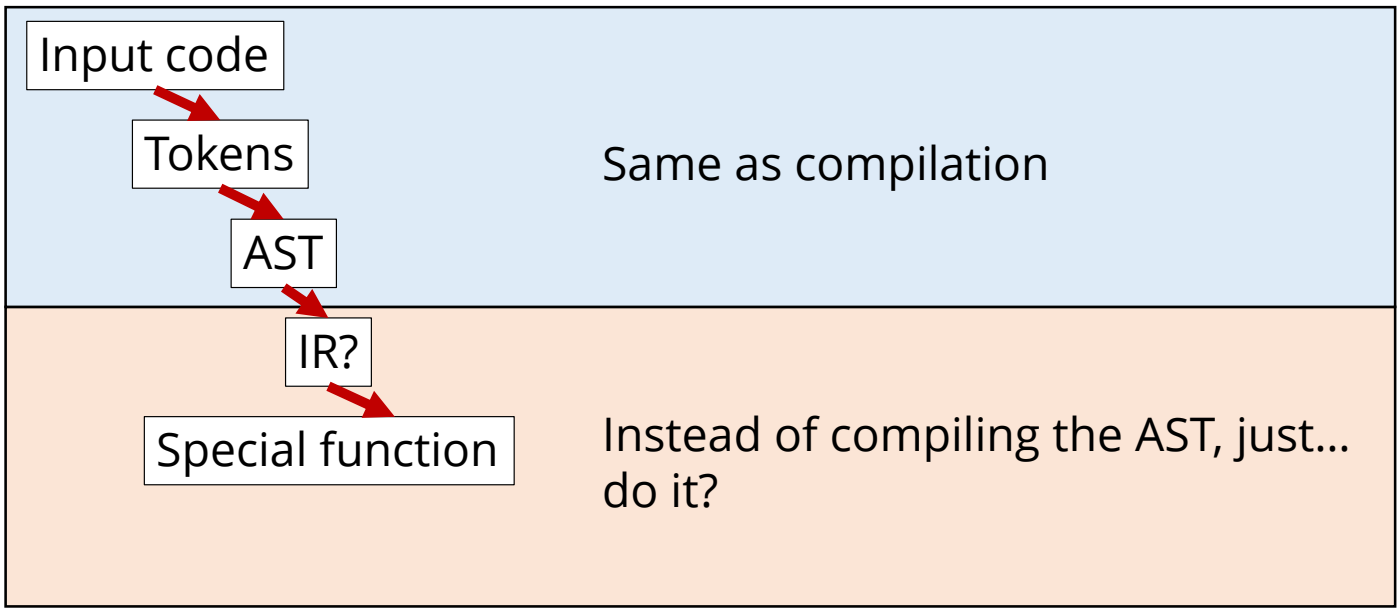
Aside: Interpretation



Aside: Interpretation



Aside: Interpretation



```
for (i = 0; i < ir.length; i++) {
  o = ir[i];
  switch (o.operation) {
    case ADD:
      r = pop();
      l = pop();
      push(add(l, r));
      break;
    ...
  }
}
```

Interpreters

- We won’t talk much about interpreters
- Good interpreters are better than bad JITs!

Interpreters

- We won't talk much about interpreters
- Good interpreters are better than bad JITs!
- "Threaded" interpreters (nothing to do with threads)

Where’s the problem?

- Machine code is great, and we’re generating machine code

Where’s the problem?

- Machine code is great, and we’re generating machine code
- The less information you have, the worse machine code you will make

Add

- C:

  Usually one machine code instruction

- JavaScript:

The addition operator either performs string concatenation or numeric addition.
The production *AdditiveExpression* : *AdditiveExpression* + *MultiplicativeExpression* is evaluated as follows:
1.Let *lref* be the result of evaluating AdditiveExpression.
2.Let *lval* be GetValue(*lref*).
3.Let *rref* be the result of evaluating MultiplicativeExpression.
4.Let *rval* be GetValue(*rref*).
5.Let *lprim* be ToPrimitive(*lval*).
6.Let *rprim* be ToPrimitive(*rval*).
7.If Type(*lprim*) is String or Type(*rprim*) is String, then
    1.Return the String that is the result of concatenating ToString(*lprim*) followed by ToString(*rprim*)
8.Return the result of applying the addition operation to ToNumber(*lprim*) and ToNumber(*rprim*). See the Note below 11.6.3.

Just stupid languages?

Let’s look at this “less information” problem more deeply…

Types

Static — Dynamic

- More known at compile time
- Prevents a class of errors (being “very stupid errors”)
- Can aid compiler???

- More flexible
- Sounds cool on a CV
- …???
- PROFIT

What type is 0?

What type is 0?

- What is a type? Its values? Its API? Its encoding?

What type is 0?

- What is a type? Its values? Its API? Its encoding?
- 0 is an int… right?

What type is 0?

- What is a type? Its values? Its API? Its encoding?
- 0 is an int… right?
- int / int → int

What type is 0?

- What is a type? Its values? Its API? Its encoding?
- 0 is an int… right?
- int / int → int
- We need to check for this *dynamically*

What type is 0?

- OK, then an int is any integer *other* than 0!

What type is 0?

- OK, then an int is any integer *other* than 0!
- int – int → int

What type is 0?

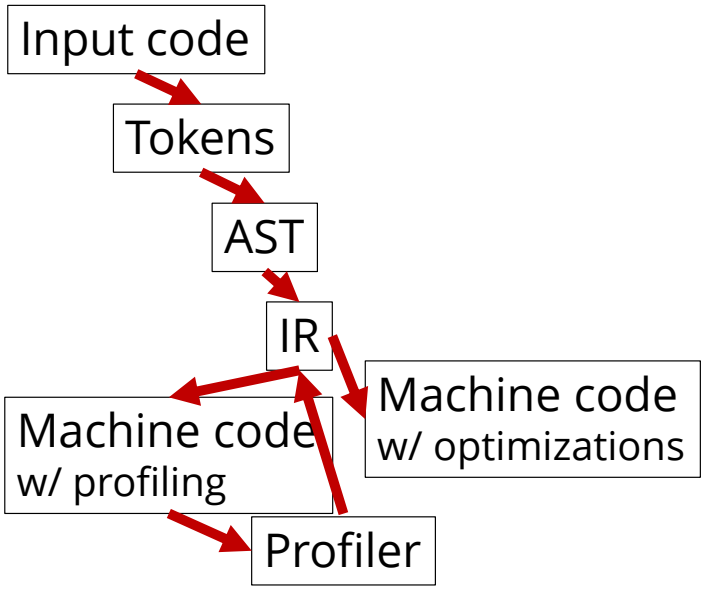
- OK, then an int is any integer *other* than 0!
- int – int → int
- By the time we've resolved this trivial problem, our type system is Turing-complete, and nobody can write a program

To do better we need to know more than the type!

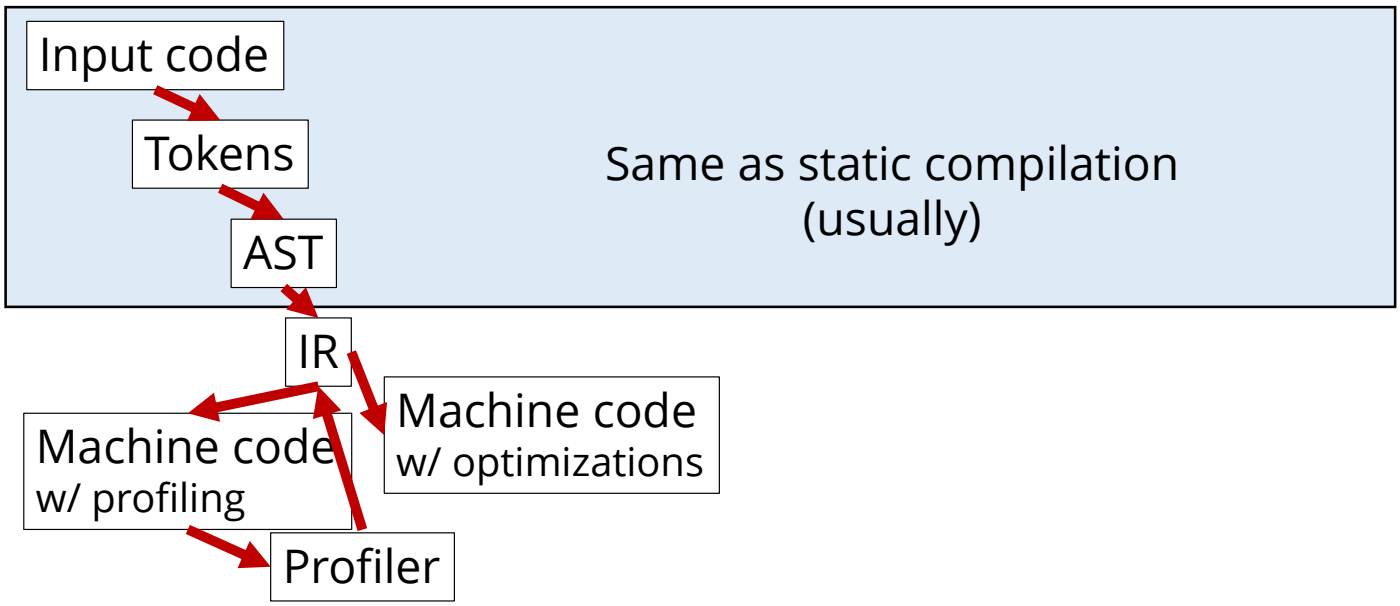
Aside: Data and code

- Data isn’t the only problem
- Inlining is an important optimization
- Knowing when to inline is AI-complete

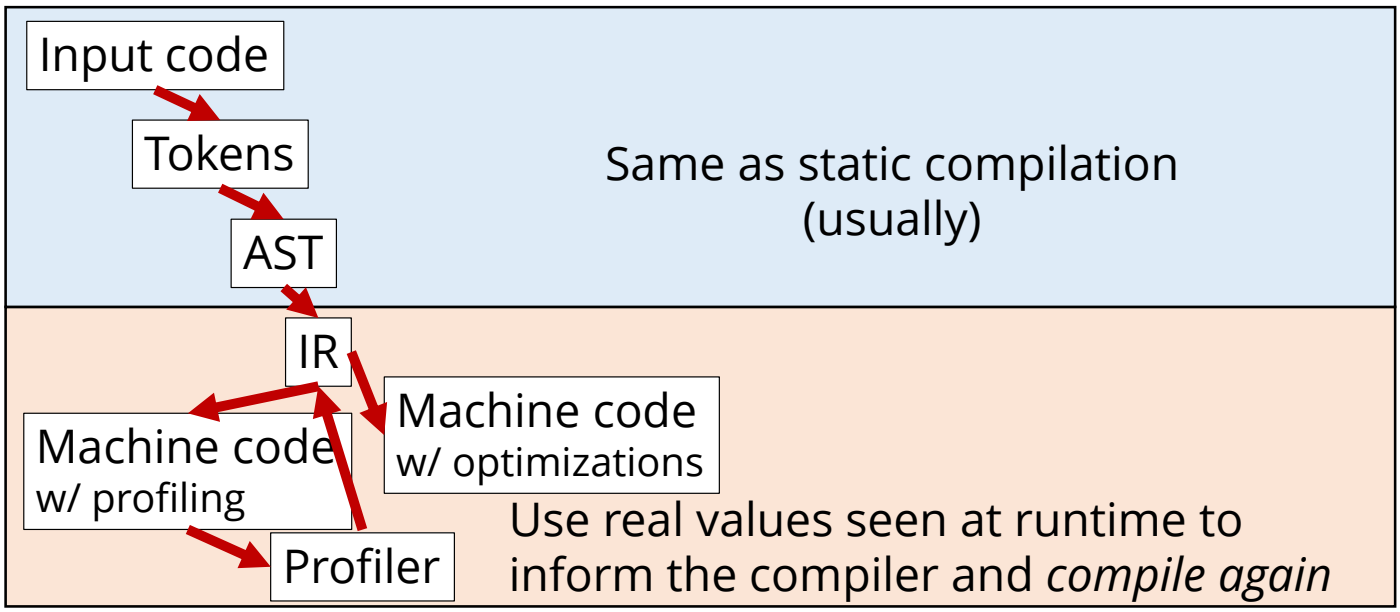
Just-in-Time compilation



Just-in-Time compilation



Just-in-Time compilation



Baby’s first JIT (demonstration)

Demo

- In that demo we didn’t make use of profiling

Demo

- In that demo we didn’t make use of profiling
- We had only one type, with a perfect API

SDyn