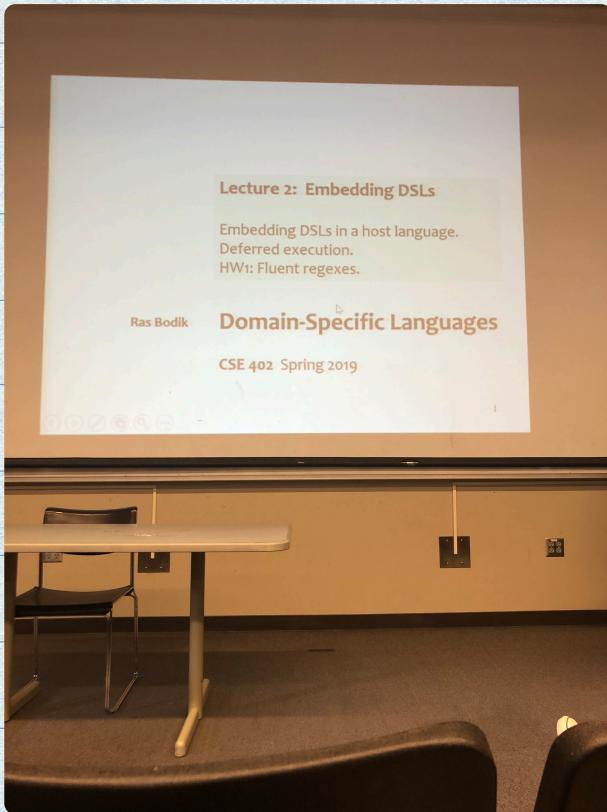


Lecture 2



Embedding DSLs.

Desugaring and Optimisation.

External DSL

Parser

→ Embedded

Internal DSL

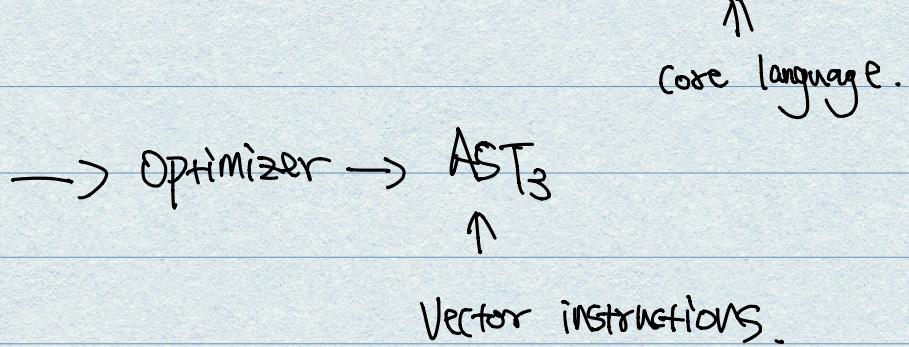
Why Lowering?

Optimization / Macro expansion.

```
graph TD; AST1[AST] --> Optimizer[optimizer]; Optimizer --> AST2[AST]; AST2 --> Interpreter[Interpreter]; AST2 --> Compiler[Compiler];
```

The diagram illustrates the flow of an Abstract Syntax Tree (AST) through an optimizer. The process starts with an initial AST, which is then processed by the optimizer to produce a second AST. This second AST can be directed to either an Interpreter or a Compiler, representing two different execution paths.

E.g. Regex .



Is the Program actually translated to AST?

- Yes : deep embedding
enables optimization.

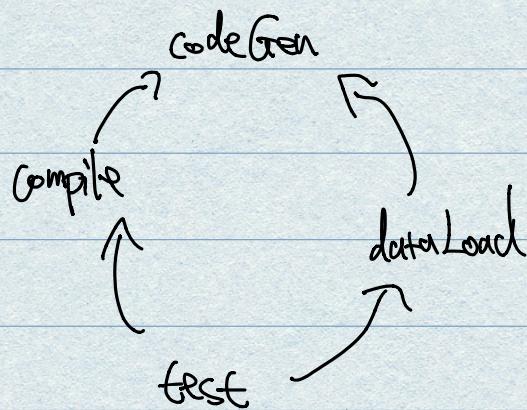
- No : shadow embedding,
⇒ used in HW,

Do not necessarily build an AST

3 methods for embedding a DSL into host language.

- rule-based syntax
- Operator overloading
- fluent syntax (aka call chaining)

⇒ Case study : raku by Jim Weirich.



Ruby is the host language
⇒ rake files are legal ruby files

Example rakefile

```
task :codeGen do
  # do the code generation
end
task :compile => :codeGen do
  # do the compilation
end
task :dataLoad => :codeGen do
  # load the test data
end
task :test => [:compile, :dataLoad] do
  # run the tests
end
```

procedure

argument /

task (hash)

Ruby symbol

hash table literal

block (closure)

array

22

⇒ What is produced?

- dependency graph ("AST")

⇒ Who executes the rules?

- runtime in response.

Operator Overloading.

$x \cdot y \cdot xy$

