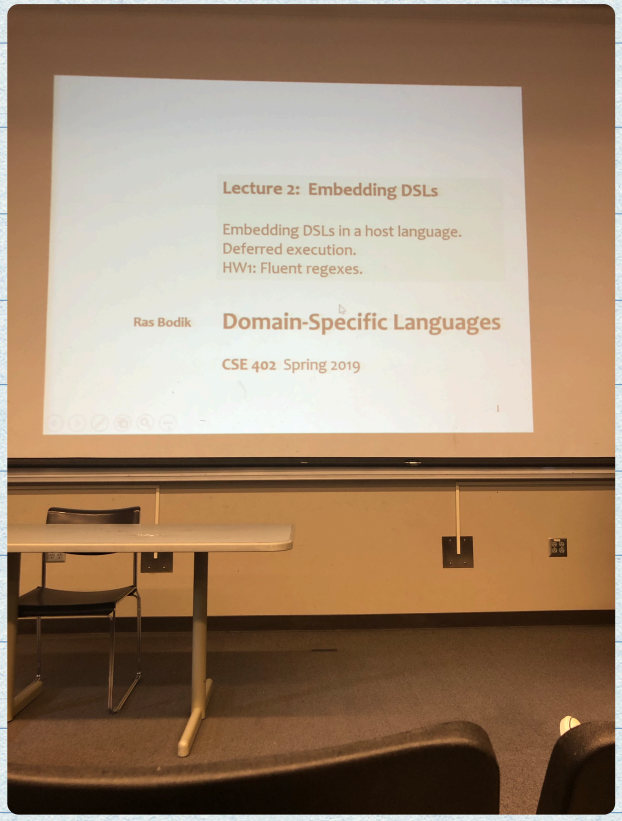
Lecture 2



Embedding DSLs.

Desugaring and Optimization.

External DSL → Parser ⟩⟩ AST$_1$ → Lowering → AST$_2$ → optimizer → AST$_3$ → Interpreter / Compiler

Internal DSL → Embedded ⟩⟩ AST$_1$ → Lowering → AST$_2$ → optimizer → AST$_3$ → Interpreter / Compiler

Why Lowering?

optimization / macro expansion.

E.g. Regex.

AST$_1$ → lowering → AST$_2$

↑ aa+ ↑ aaa*

↑
core language.

→ Optimizer → AST_3

↑

Vector instructions.

Is the Program actually translated to AST?

- Yes: deep embedding

  enables optimization.

- No: shallow embedding

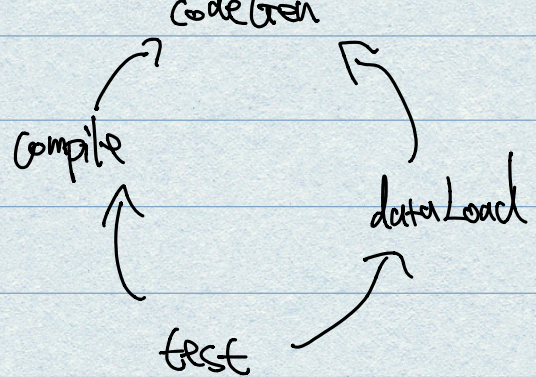
  ⇒ used in HW,

  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)

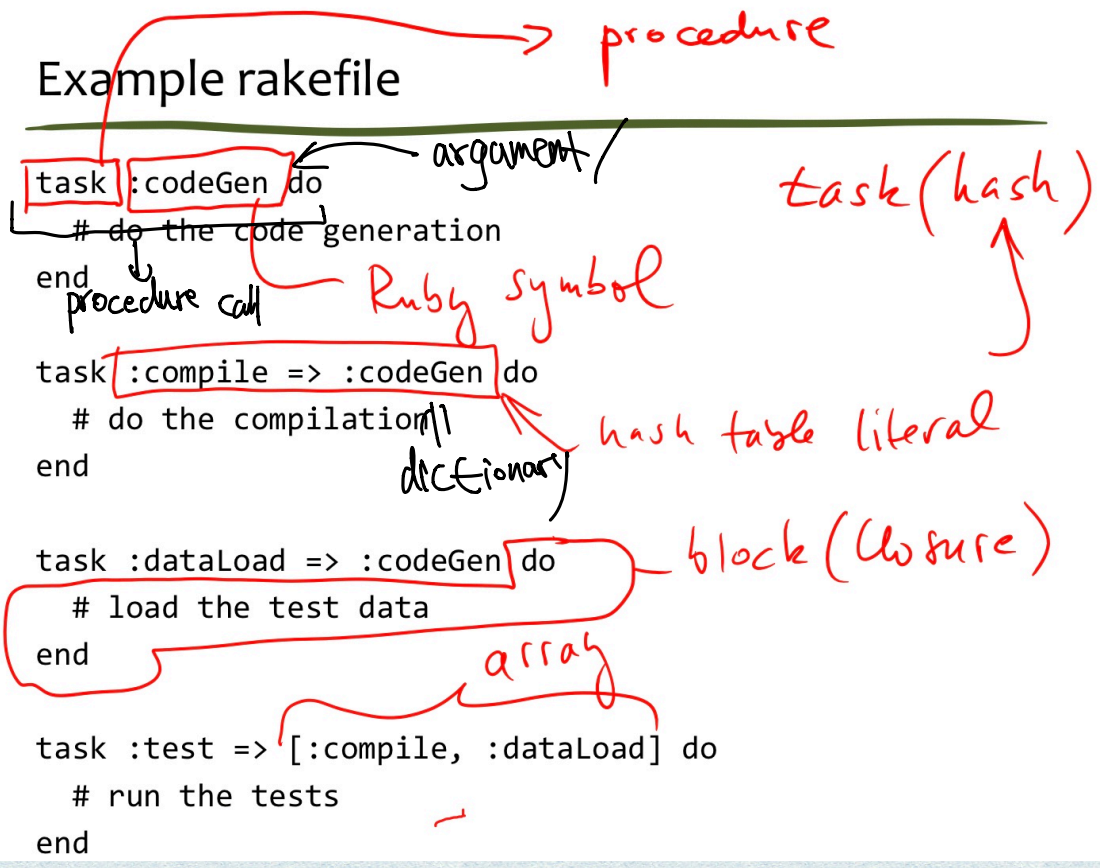
⇒ Case study: rake by Jim Weirich.



Ruby is the host language

⇒ rake files are legal ruby files

Example rakefile



22

⇒ What is produced?

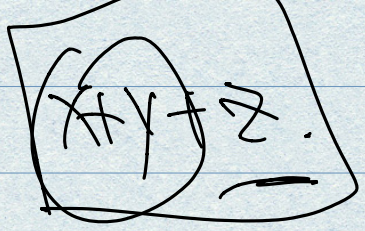
- dependency graph ("AST")

⇒ Who executes the rules?

- runtime in response.

Operator Overloading.

$\lambda x.\lambda y.\ xy$


x+y+z.