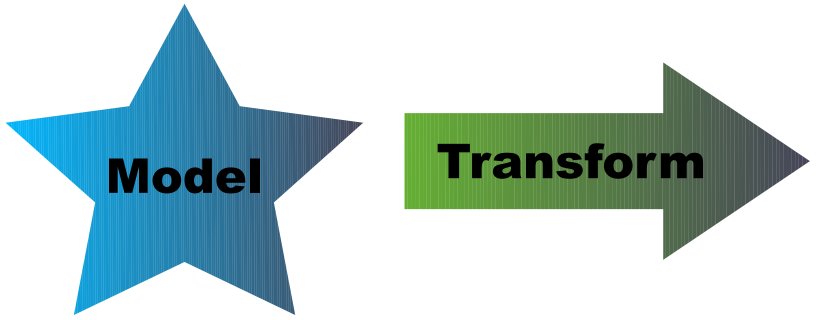
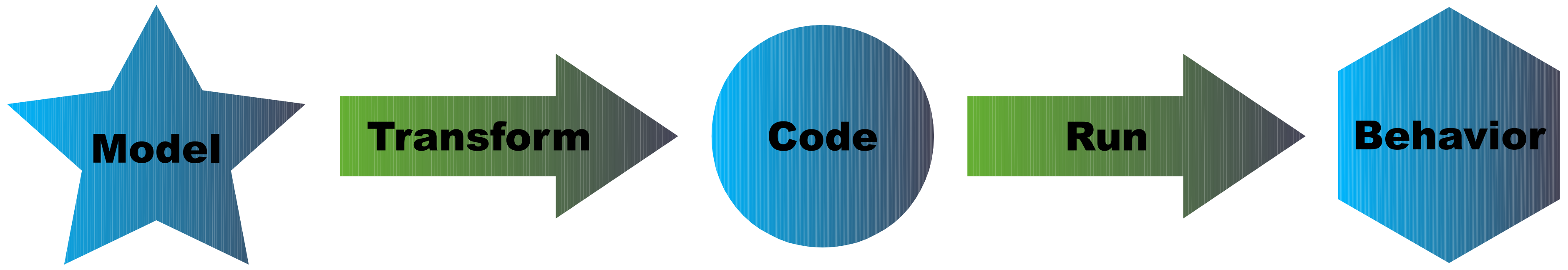
Model Interpretation and Compilation by Partial Evaluation

William R. Cook
University of Texas at Austin
with
Ben Wiedermann, Ali Ibrahim, Ben Deleware,
Sidney Rosairo, Amin Shali, David Kitchin

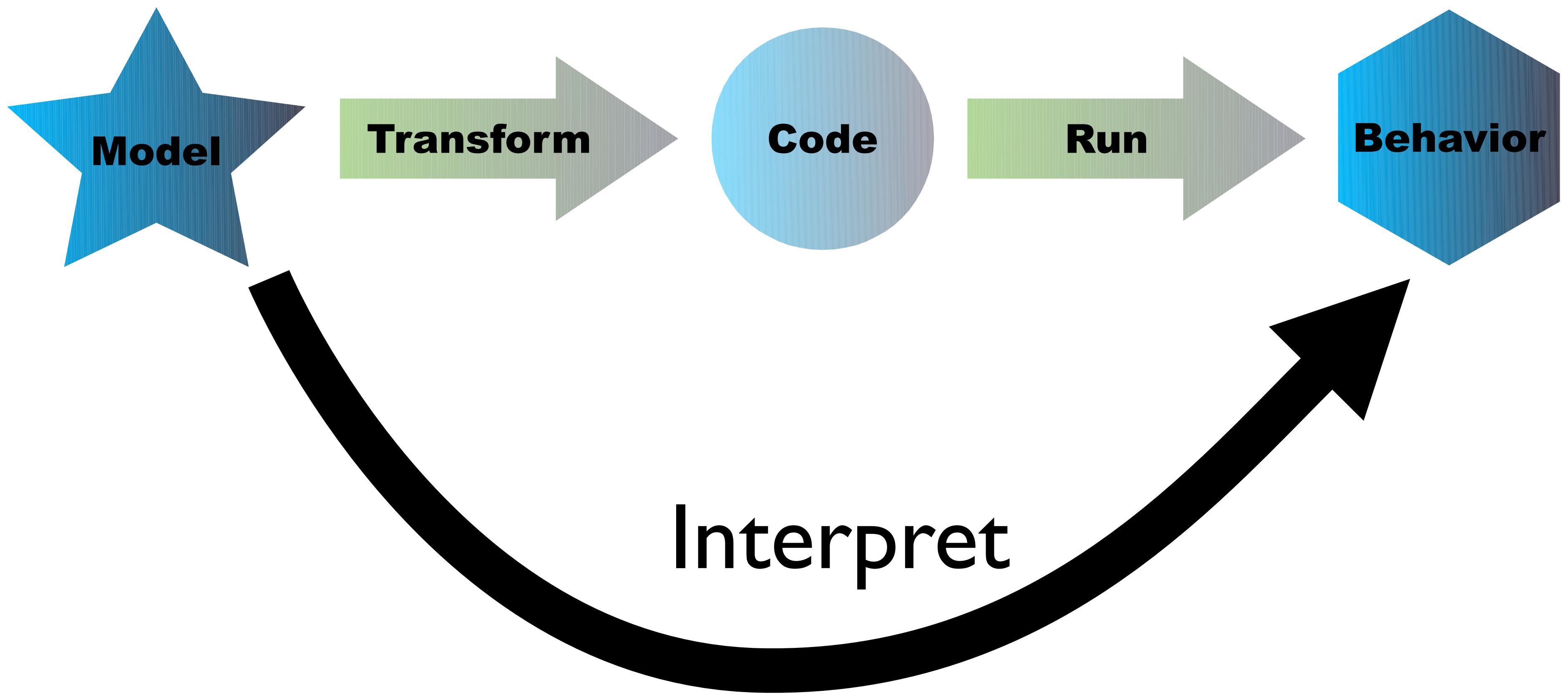
Model Transformation



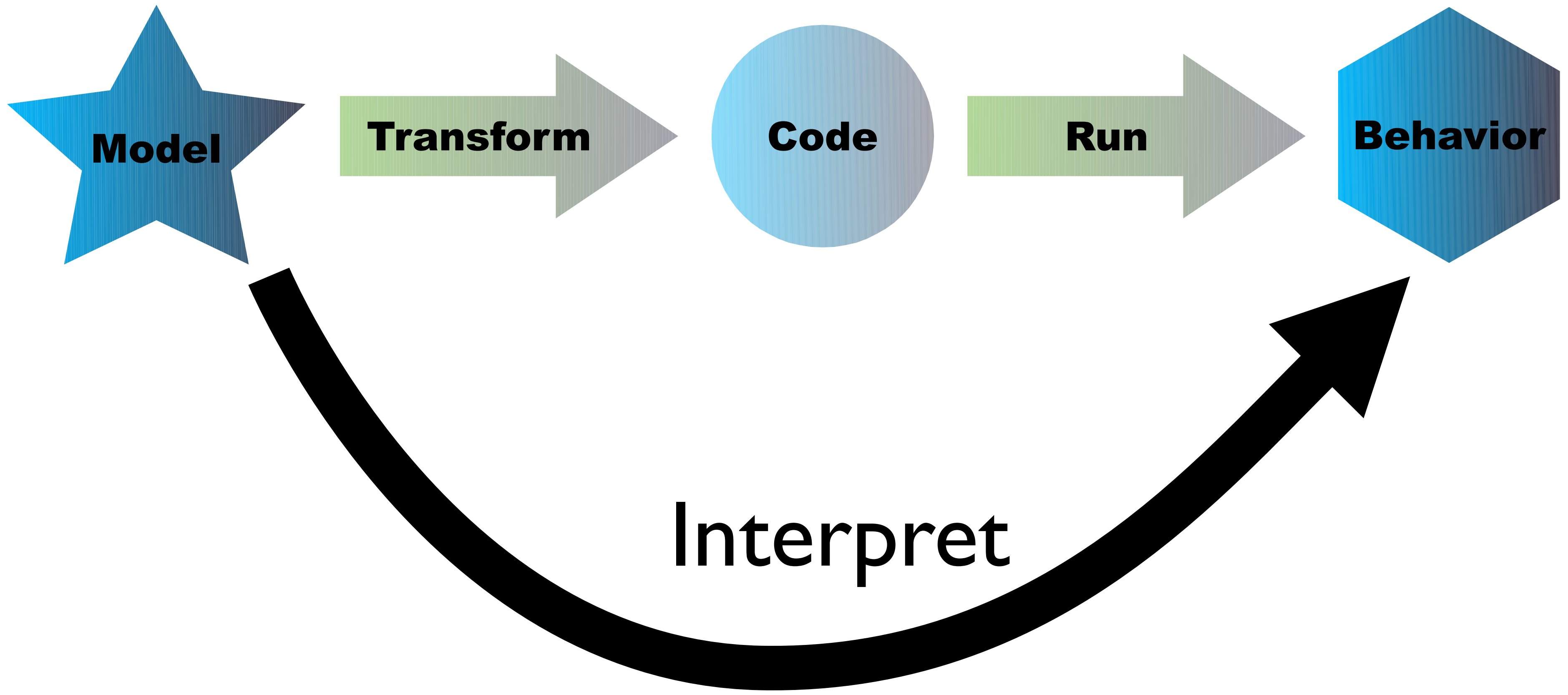
Model Transformation



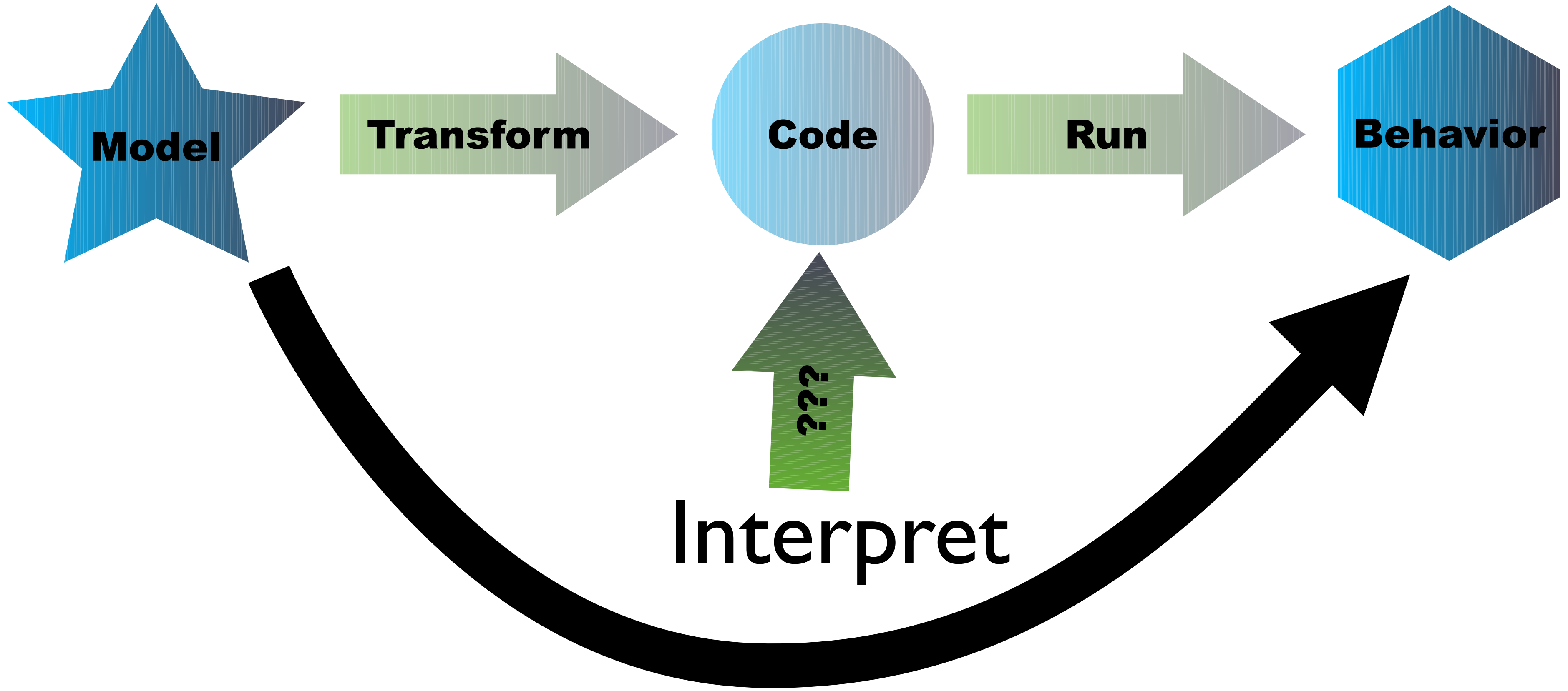
Do we need generate code?



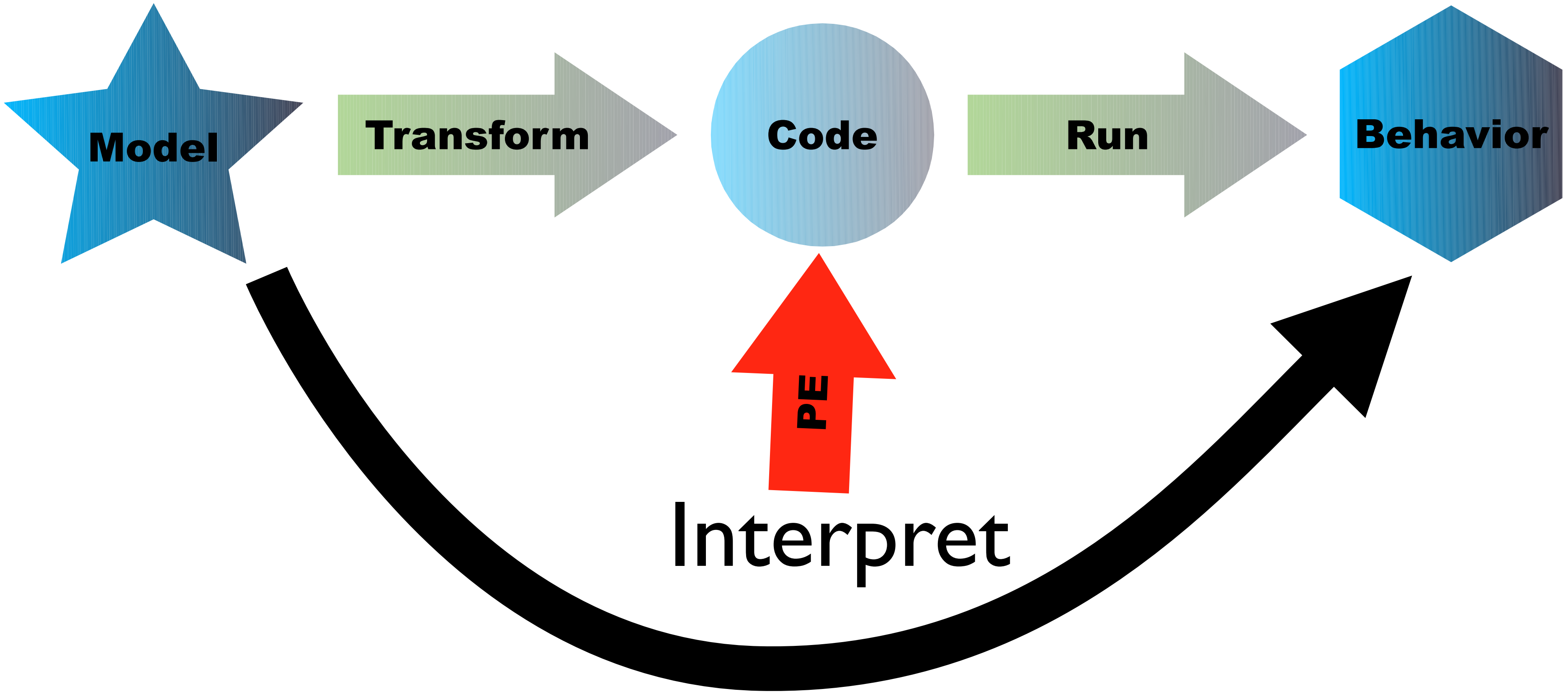
Interpreters are Slower



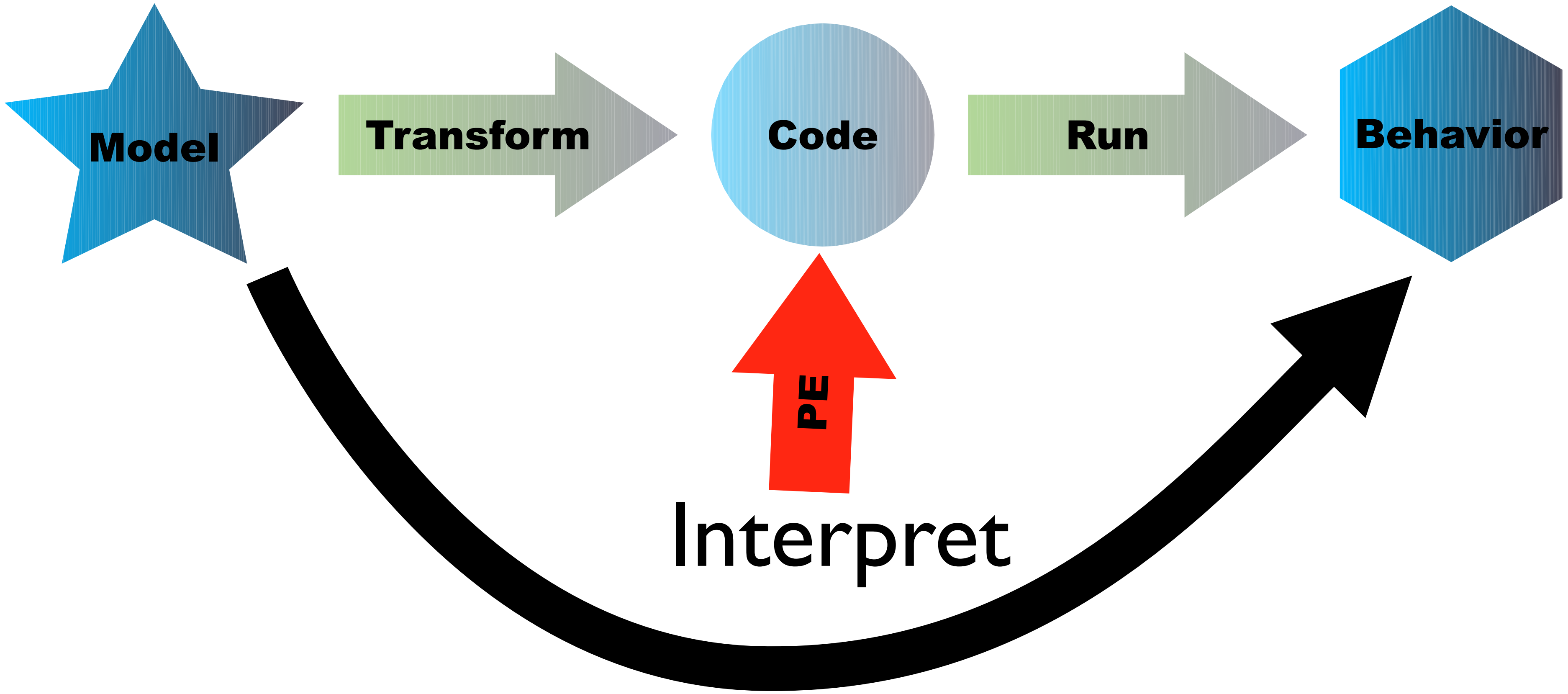
Get the code from the interpreter?



Yes!! Partial Evaluation of an interpreter creates compiled code



Yes!! Partial Evaluation of an interpreter creates compiled code



Its a long story…

Partial Evaluation (by hand)

Example: Power function

pow(n, x) = if (n==0) then 1 else x*pow(n-1, x)

What if you know n?

pow(3, x) = x*pow(2, x)

if (3==0) then 1
else x*pow(3-1, x)

This depends on pow(2, x)

pow(2, x) = x*pow(1, x)

pow(1, x) = x*pow(0, x)

pow(0, x) = 1

Partial Evaluation

Example: Power function

pow(n, x) = if (n==0) then 1 else x*pow(n-1, x)

Lets call this final function "pow3":

pow3(x) = x*x*x

Partial evaluation

- Eliminates computations that depend on known inputs
- Result is "residual program"
- Doesn't always work:
  - pow(n, 19) = if (n==0) then 1 else 19*pow(n-1, 19)
- Useful when raising many numbers to 3rd power

Automatic Partial Evaluation

Example: Power function

pow(n, x) = if (n==0) then 1 else x*pow(n-1, x)

Can we compute residual code automatically?

peval(pow, 3) $\rightarrow$ fun(x) x*x*x $\equiv$ pow3

Partial evaluation function: peval

Inputs:

- Source code of a function
- Value of the first argument

Output:

- Residual code from partially evaluating

Automatic Partial Evaluation

More formally, for any function f and value v

peval(f, v) = g

- "peval" is traditionally called "mix"

such that for any value x

g(x) = f(v, x)

One implementation is "currying"

peval(f, v) = λx. f(v, x)

a true partial evaluator returns residual code, not a curried function

Interpreters

Command line for python interpreter

```
python  notify.pl  in.txt  >  out.txt
```

An interpreter is a function of two arguments

python(“notify.pl”, “in.txt”) ➔ output

Just like the pow function

pow(3, 19) ➔ 6,895

Partial Evaluation of Interpreters

What if the program is known but input is not?

python(“notify.pl”, ?)

Useful because we often run the same python program many times on different inputs

Apply automatic partial evaluation

peval(python, “notify.pl”) → g

where

g(“in.txt”) = python(“notify.pl”, “in.txt”)

What is “g”?

Partial Evaluation of Interpreters

What if the program is known but input is not?

python("notify.pl", ?)

Useful because we often run the same python program many times on different inputs

Apply automatic partial evaluation

peval(python, "notify.pl") → g

where

g("in.txt") = python("notify.pl", "in.txt")

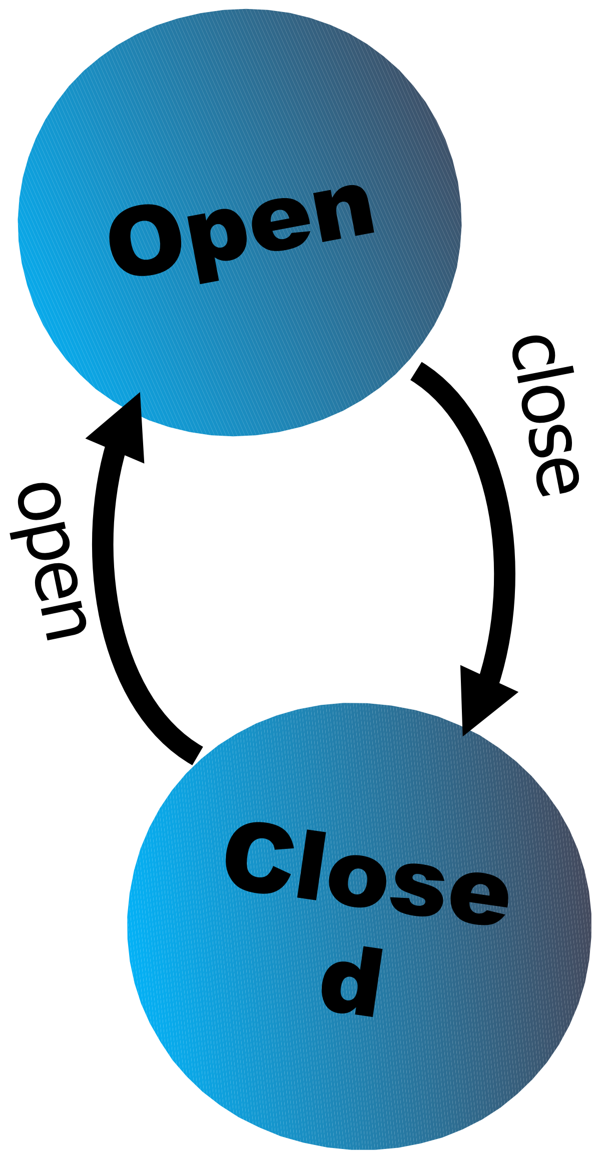
What is "g"?

Compiled version of "notify.pl"!

Example Modeling Language/Model

State Machine Model

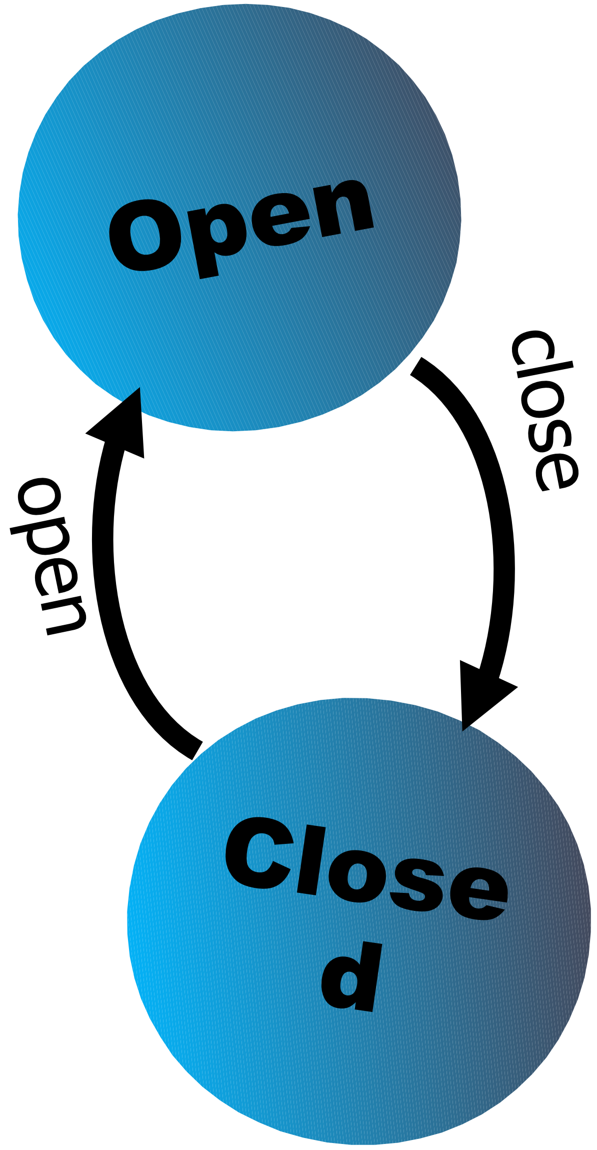
```
class State {
  String label;
  Transition[] trans;
}

class Transition {
  String event;
  State to;
}
```



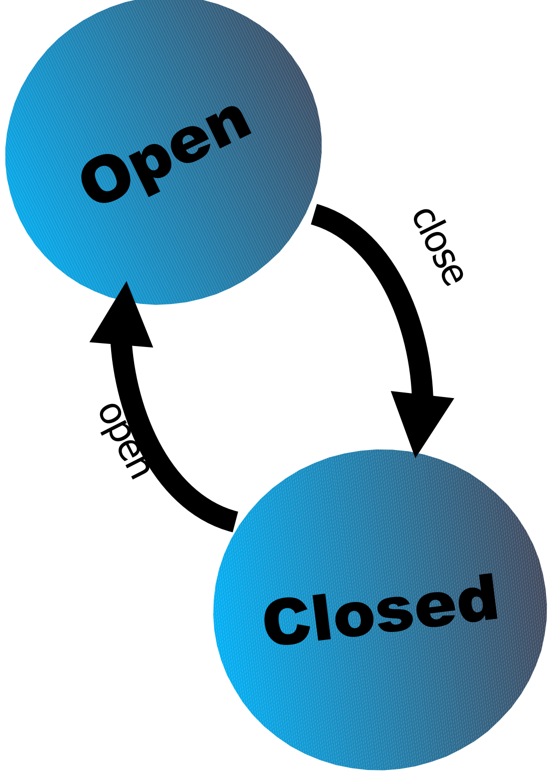
Example Model Interpreter

Interpreter

```
int run(State current) {
  print(current.label);
  String input = in.readLine();
  for (Trans t : current.trans)
    if (t.event == input)
     return run(t.to);
  return run(current);
}
```



Partial Evaluation



```
int runOpen() {
  print("Open");
  String input = in.readLine();
  if ("close" == input)
    return runClosed();
  return runOpen();
}
int runClosed() {
  print("Closed");
  String input = in.readLine()
  if ("open" == input)
    return runOpen();
  return runClosed();
}
```

```
int run(State current) {
  print(current.label);
  String input = in.readLine();
  for (Trans t : current.trans)
    if (t.event == input)
     return run(t.to);
  return run(current);
}
```

First Futamura Projection (1971)

Partial evaluation
of an interpreter
with respect to a program
is a
compiled version
of the program

That is, result is a version of the interpreter specialized to run just that one program

Futamura Projections I

Interpreter

python("notify.pl", "in.txt") $\rightarrow$ o

First Futamura projection

peval(python, "notify.pl") $\rightarrow$ g   where g("in.txt") = o

Futamura Projections (pattern)

Interpreter

python("notify.pl", "in.txt") ➔ o

First Futamura projection

peval(python, "notify.pl") ➔ g where g("in.txt") = o

g is compiled version of notify.pl

Futamura Projections II

Interpreter

python("notify.pl", "in.txt") ➔ o

First Futamura projection

peval(python, "notify.pl") ➔ g   where g("in.txt") = o

g is compiled version of notify.pl

Second Futamura projection

peval(peval, python) ➔ c   where c("notify.pl") = g

Futamura Projections II

Interpreter

python(“notify.pl”, “in.txt”) ➔ o

First Futamura projection

peval(python, “notify.pl”) ➔ g where g(“in.txt”) = o

g is compiled version of notify.pl

Second Futamura projection

peval(peval, python) ➔ c where c(“notify.pl”) = g

c is a python compiler

Futamura Projections III

Interpreter

python("notify.pl", "in.txt") ➔ o

First Futamura projection

peval(python, "notify.pl") ➔ g where g("in.txt") = o

g is compiled version of notify.pl

Second Futamura projection

peval(peval, python) ➔ c where c("notify.pl") = g

c is a python compiler

Third Futamura projection

peval(peval, peval) ➔ z where z(python) = c

Futamura Projections III

Interpreter

python("notify.pl", "in.txt") ➔ o

First Futamura projection

peval(python, "notify.pl") ➔ g   where g("in.txt") = o

g is compiled version of notify.pl

Second Futamura projection

peval(peval, python) ➔ c   where c("notify.pl") = g

c is a python compiler

Third Futamura projection

peval(peval, peval) ➔ z   where z(python) = c

z is a compiler compiler!

We only need First Projection

Interpreter

python("notify.pl", "in.txt") ➔ o

First Futamura projection

peval(python, "notify.pl") ➔ g where g("in.txt") = o

g is compiled version of notify.pl

~~Second Futamura projection~~

~~peval(peval, python) ➔ c where c("notify.pl") = g~~

~~c is a python compiler~~

~~Third Futamura projection~~

~~peval(peval, peval) ➔ z where z(python) = c~~

~~z is a compiler compiler!~~

Avoid Need for Self-Applicable peval

Interpreter

python("notify.pl", "in.txt") ➜ o

First Futamura projection

peval(python, "notify.pl") ➜ g where g("in.txt") = o

g is compiled version of notify.pl

Second Futamura projection

$\boxed{\text{peval(peval, python)}}$ c where c("notify.pl") = g

c is a python compiler

Third Futamura projection

$\boxed{\text{peval(peval, peval)}}$ z where z(python) = c

z is a compiler compiler!

Futamura in Practice

Interpreters have “good” behavior

- Control flow depends on program first, then input
  - just like pow(n, x): control flow depends on n

Can’t make good compilers via 2nd/3rd Futamura

- Trying to make a C compiler via Futamura will fail
- Was that the right goal?
  - Be careful what you pick as challenge problem

Hypothesis:

- First Futamura projection will work well enough for model interpreters
  - solves real problem, simple partial evaluator

Plan

Goal

Compile model languages
by partial evaluation of model interpreters

Technique

Model interpreters written in Java

e.g. ModelTalk, WebDSL, others?

Partial Evaluator for Java

Residual (compiled code) is also in Java

So…

How do we write a partial evaluator for Java?

Language Levels

Two levels of language

- M: Language being interpreted
  - e.g. Python or a Modeling language
- L: Language in which interpreter is written
  - e.g. Java, C, etc

Components

- Partial evaluator for L
- Model interpreter I for M written in L
- A model R written in M
- Residual code peval(I, R) is also in L

Simple Evaluator E

x : Variable v : Value

e = x | v | **`if`** e **`then`** e **`else`** e | e+e | `f(`e, …, e`)`

ρ environment maps *all* variables to values

$E[v]\rho = v$

$E[x]\rho = \rho(x)$

E[**`if`** e_1 **`then`** e_2 **`else`** e_3]ρ = **if** E[e_1]ρ **then** E[e_2]ρ **else** E[e_3]ρ

$E[e_1+e_2]\rho = E[e_1]\rho + E[e_2]\rho$

E[`f(`e_1, …, e_n`)`]ρ = E[e]ρ'

lookup function definition: `f(`x_1, ..., x_n`)=`e

$\rho' = \{ x_1 = E[e_1]\rho, \ldots, x_n = E[e_n]\rho \}$

From Full Evaluation to Partial Evaluation

The type of eval

E : Expression → Environment → Value

Environment = Variable → Value

FreeVars(e) $\subseteq$ Domain(v)

All variables are bound

What about a partial evaluator?

Environment gives values to some variables

P : Expression → Environment → Expression

Result might not be a complete value

P[x+y] {x=3, y=2} ➔ 5

P[x+y] {x=3} ➔ [3+y]

Online Partial Evaluator P

x : Variable v : Value

e = x | v | **`if`** e **`then`** e **`else`** e | e+e | `f(`e, …, e`)`

ρ environment maps ***some*** variables to values

P[v]ρ = v

P[x]ρ = **if** $x \in \mathrm{dom}(\rho)$ **then** ρ(x) **else** [x]

returns code [x] if the variable is not defined

Online Partial Evaluator P

x : Variable v : Value

e = x | v | **`if`** e **`then`** e **`else`** e | e+e | `f`(e, …, e)

ρ environment maps ***some*** variables to values

P[v]ρ = v

P[x]ρ = **if** $x \in$ dom(ρ) **then** ρ(x) **else** [x]

P[**`if`** e_1 **`then`** e_2 **`else`** e_3]ρ =

  case P[e_1]ρ **of**

    $v \rightarrow$ **if** v **then** P[e_2]ρ **else** P[e_3]ρ

  [e] $\rightarrow$ [**`if`** e **`then`** P[e_2]ρ **`else`** P[e_3]ρ]

if its a boolean v, then pick branch.
else create a new if statement

Online Partial Evaluator P

$P[v]\rho = v$

$P[x]\rho = \textbf{if } x \in \mathrm{dom}(\rho) \textbf{ then } \rho(x) \textbf{ else } [x]$

$P[\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3]\rho =$

```
  case P[e₁]ρ of
     v → if v then P[e₂]ρ else P[e₃]ρ
  [e] → [if e then P[e₂]ρ else P[e₃]ρ]
```

$P[e_1+e_2]\rho =$

$v_1 + v_2 \quad \textbf{if } v_i = P[e_i]\rho$

$[e'_1+e'_2] \quad \textbf{if } e'_i = P[e_i]\rho$

apply operator if arguments are both are values
otherwise generate new expression

Function Calls

$\mathsf{P}[\mathtt{f}(e_1, \ldots, e_n)]\rho = [\mathtt{f}_{v_1,\ldots,v_j}(e'_{d_1}, \ldots, e'_{d_k})]$

1. lookup function definition: $\mathtt{f}(x_1, \ldots, x_n)\mathtt{=}e$
2. Partially evaluate the arguments

   $e'_i = \mathsf{P}[e_i]$
3. partition arguments into static and dynamic

   $\{ s_1, \ldots, s_j \} \cup \{ d_1, \ldots, d_k \} = \{ 1, \ldots, \mathrm{n} \}$

   $\vee\ \{ e'_{s_1}, \ldots, e'_{s_j} \} = \{ v_1, \ldots, v_j \}$
4. create environment with static variables

   $\rho' = \{ x_{s_1} = v_1, \ldots, x_{s_j} = v_j \}$
5. create new function specialized by statics

   $\mathtt{f}_{v_1,\ldots,v_j}(x_{d_1}, \ldots, x_{d_k})\mathtt{=}\ \mathsf{E}[e]\rho'$
6. Residual code is call with dynamic arguments

   $[\mathtt{f}_{v_1,\ldots,v_j}(e'_{d_1}, \ldots, e'_{d_k})]$

Java Partial Evaluation Concerns

Mutable state

- Supported!
- A mutable object is either static or dynamic stage
  - All mutations happen in correct order with stage
    (But static stage happens before dynamic stage)

Reflection becomes static

```
String name = "getSize";
Method m = o.getClass().getMethod(name);
m.invoke(o);
```

converts to:

```
o.getSize();
```