ML for ML: Learning Cost Semantics by Experiment

Ankush Das (CMU)
Jan Hoffmann (CMU)

TACAS 2017

ML for ML: Learning Cost Semantics by Experiment

Machine Learning

Ankush Das (CMU)
Jan Hoffmann (CMU)

TACAS 2017

ML for ML: Learning Cost Semantics by Experiment

Machine Learning

OCaml SML

Ankush Das (CMU)

Jan Hoffmann (CMU)

TACAS 2017

Which one's faster?

```
let rec append l1 l2 =
  match l1 with
  | [] -> l2
  | hd::tl -> hd::(append tl l2);;
```

```
let rec append l1 l2 =
  match l1 with
  | [] -> l2
  | x::[] -> x::l2
  | x::y::[] -> x::y::l2
  | x::y::tl -> x::y::(append tl l2);;
```

What do existing resource analysis tools do?

What do existing resource analysis tools do?

Rely on abstract cost models

loop iterations

function calls

arithmetic operations

comparisons

Abstract cost models?

Abstract cost models?

Program
P (n)

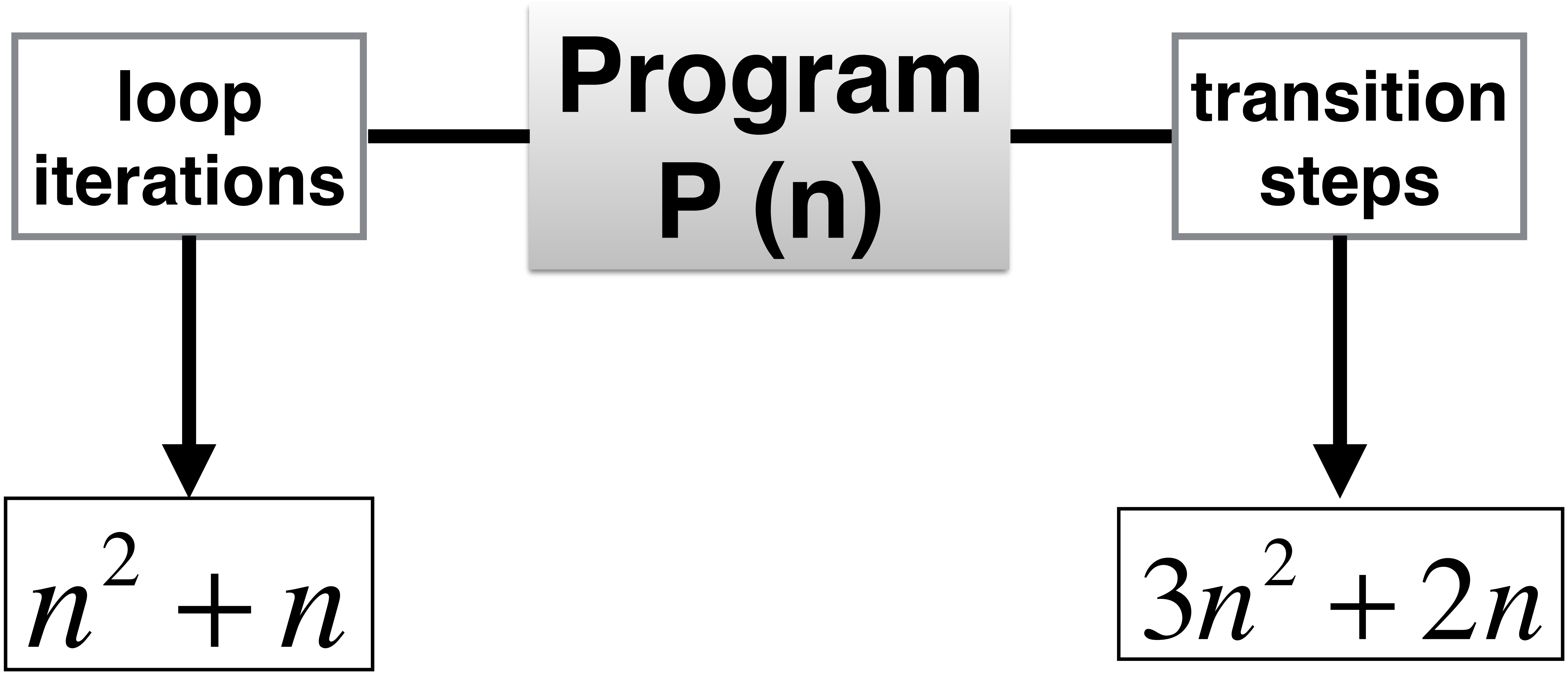
Abstract cost models?

loop iterations

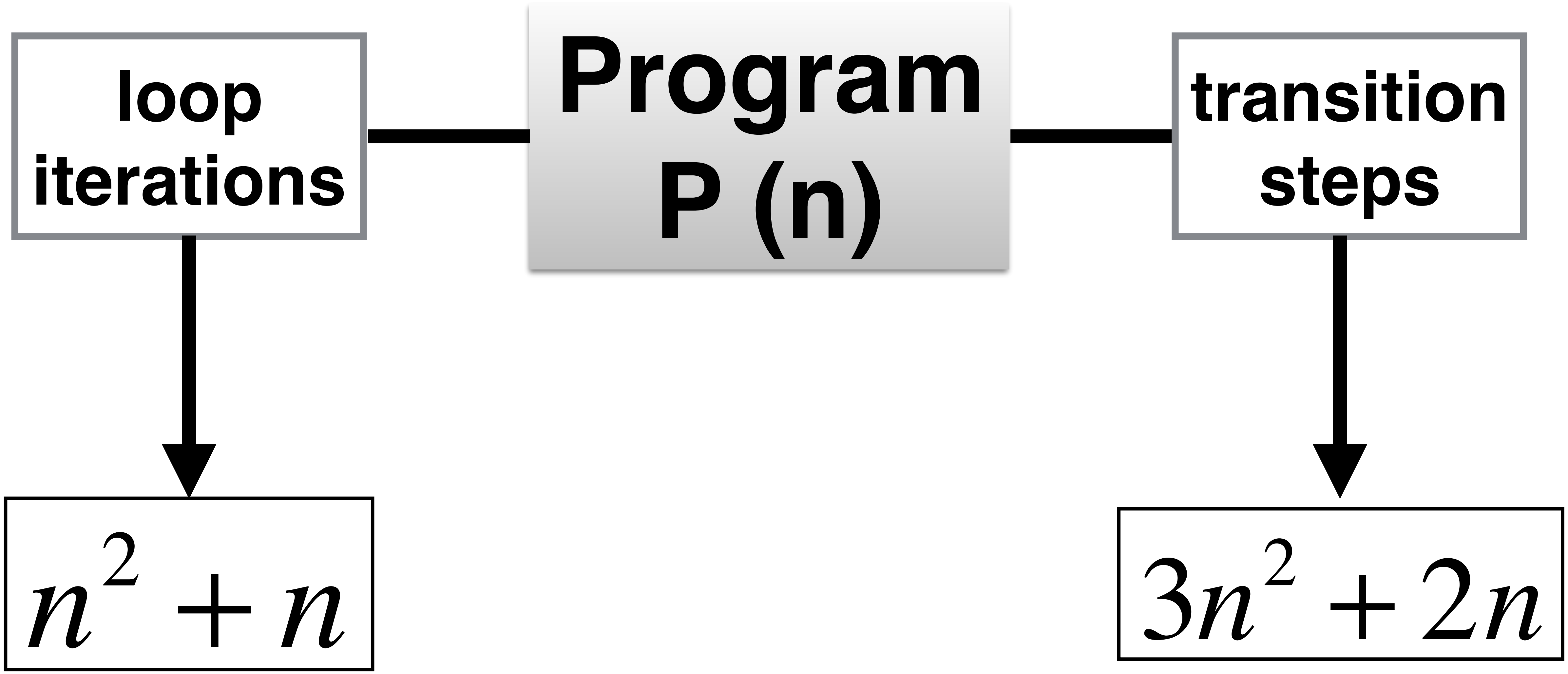
Program P (n)

$$n^2 + n$$

Abstract cost models?

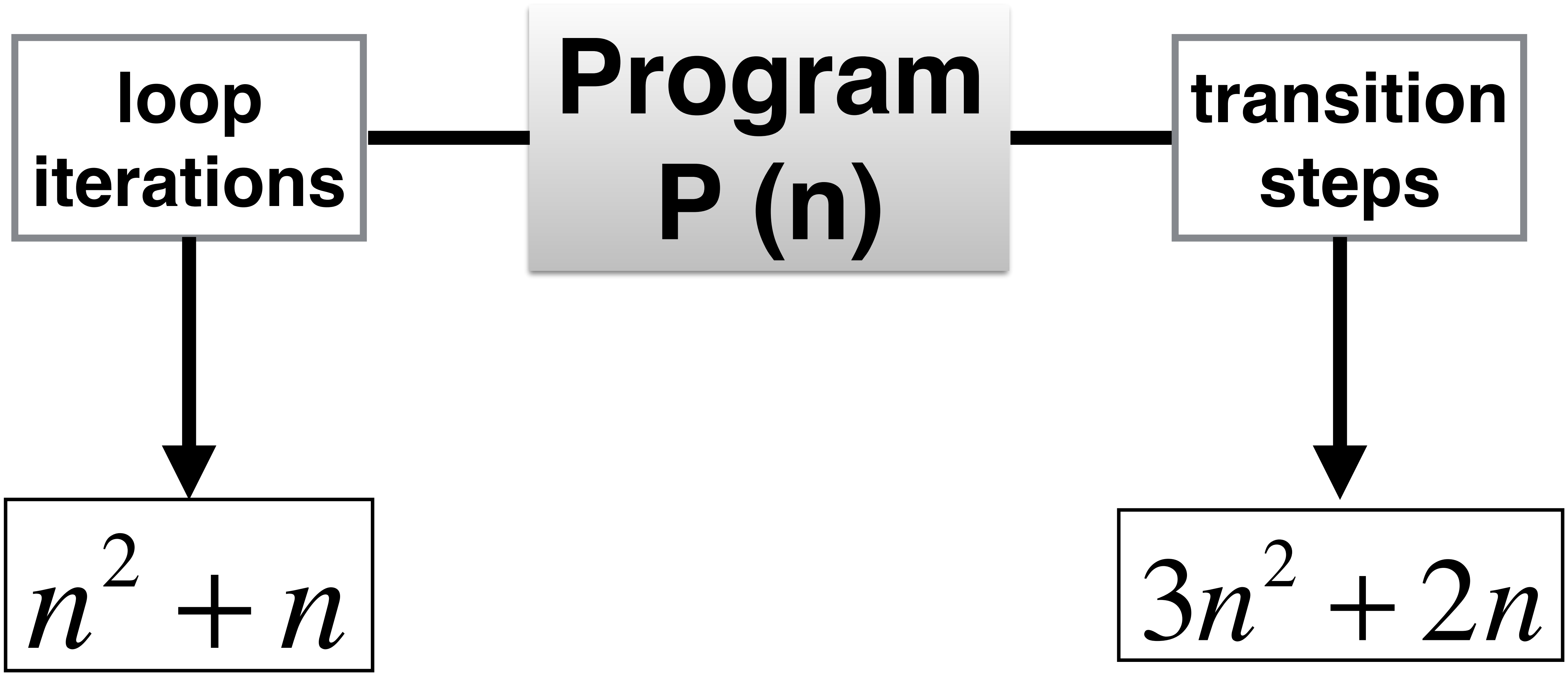


Abstract cost models?



Time on hardware?

Abstract cost models?



Time on hardware?

$$c_1 n^2 + c_2 n + c_3$$

Abstract cost models?

Program P (n)

loop iterations

transition steps

$$n^2 + n$$

$$3n^2 + 2n$$

Cost Semantics can help bridge the gap!

Time on hardware?

$$c_1 n^2 + c_2 n + c_3$$

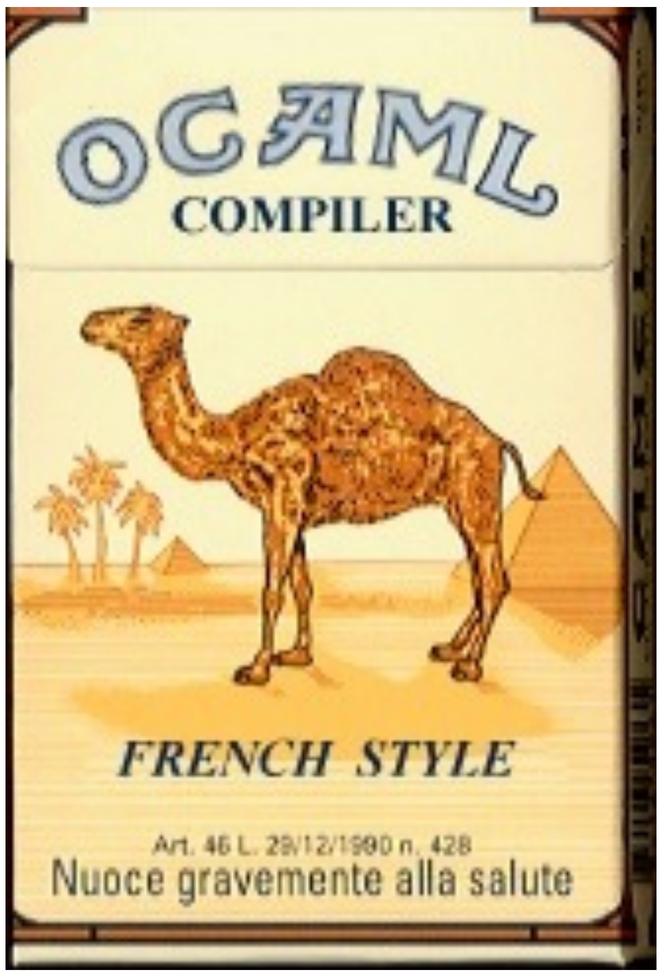
What is Cost Semantics?

A cost semantics specifies the abstract cost of a program that is validated by a provable implementation that transfers the ***abstract cost*** *to a precise* ***concrete cost*** *on a* ***particular platform****.*

— Robert Harper

How do we define a cost semantics for execution time that works well on real hardware?

Challenges



Compiler Optimizations **Garbage Collector**

Contributions

- Cost semantics for execution time
- Learn hardware specific constants
- Model for the garbage collector
- Model some compiler optimizations
- Reasonable error on Intel x86 and ARM
- Fast / slow implementations of same specification

Intuition

Example

```
let rec fact n =
    if (n = 0) then 1 else n * fact (n-1);;

(fact 10);;
```

Example

```
let rec fact n =
    if (n = 0) then 1 else n * fact (n-1);;

(fact 10);;
```

- Startup = 1
- Integer Comparison = 11
- Function Application = 11
- Integer Multiplication = 10
- Integer Subtraction = 10
- Let Rec = 1

Example

```
let rec fact n =
    if (n = 0) then 1 else n * fact (n-1);;

(fact 10);;
```

- Startup = 1
- Integer Comparison = 11
- Function Application = 11
- Integer Multiplication = 10
- Integer Subtraction = 10
- Let Rec = 1

Execution Time =

$$T_{startup} +$$

$$11 \times T_{\text{int}\,CompEq} +$$

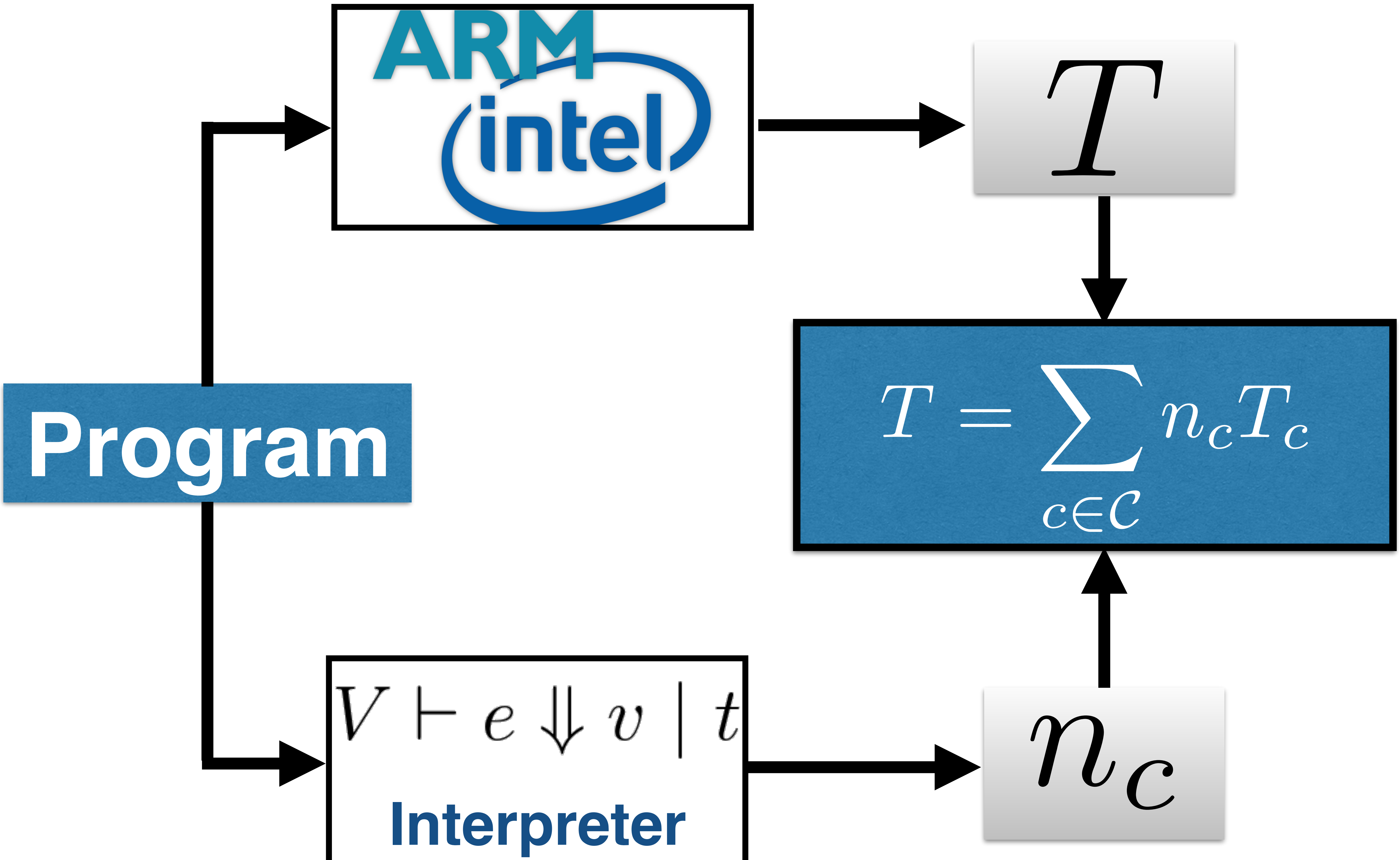
$$11 \times T_{app} +$$

$$10 \times T_{\text{int}\,Mult} +$$

$$10 \times T_{\text{int}\,Sub} +$$

$$1 \times T_{letrec}$$

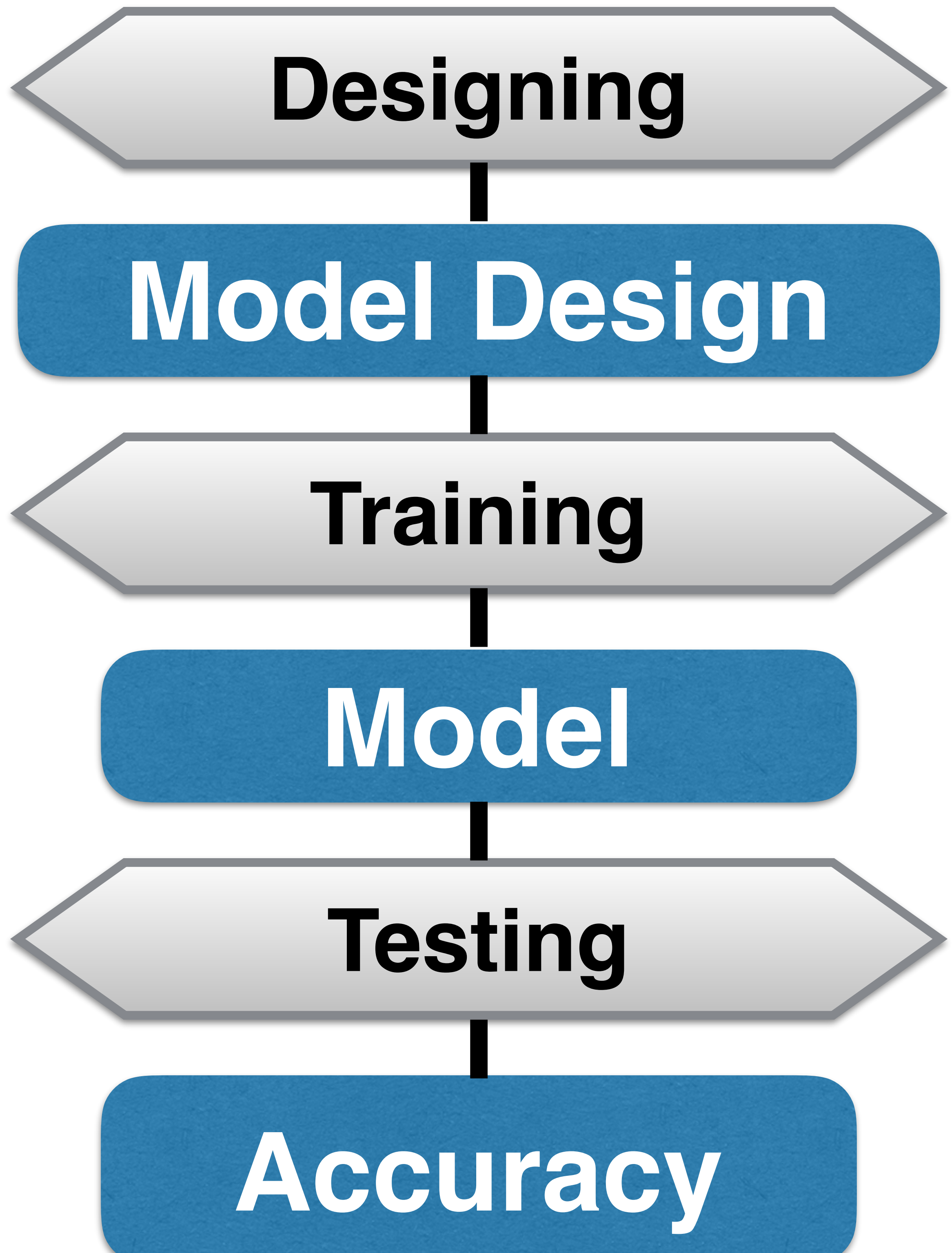
Our Work



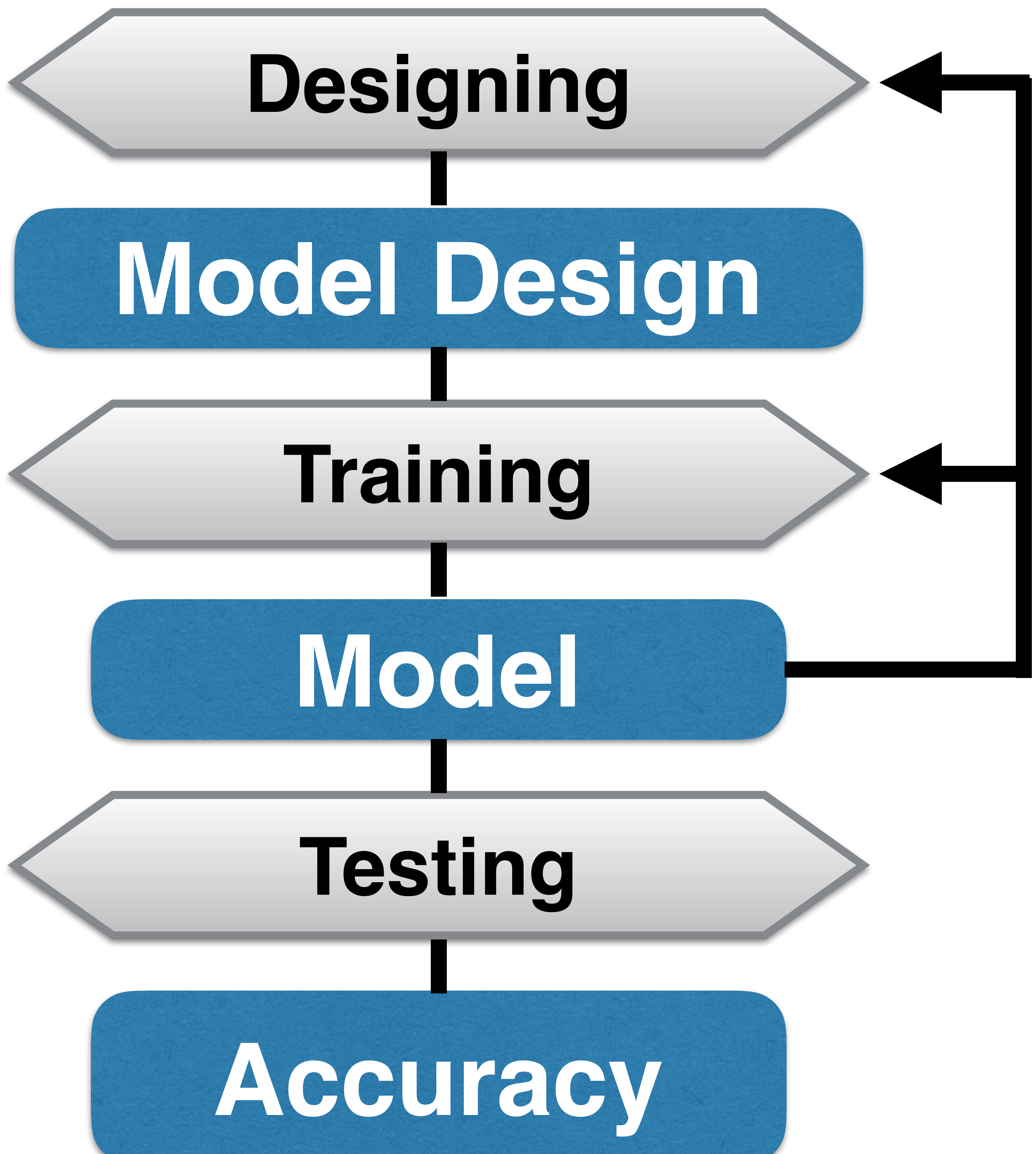
ARM
intel
T
Program
$T = \sum_{c \in C} n_c T_c$
$V \vdash e \Downarrow v \mid t$
Interpreter
n_c

$$T = \sum_{c \in \mathcal{C}} n_c T_c$$

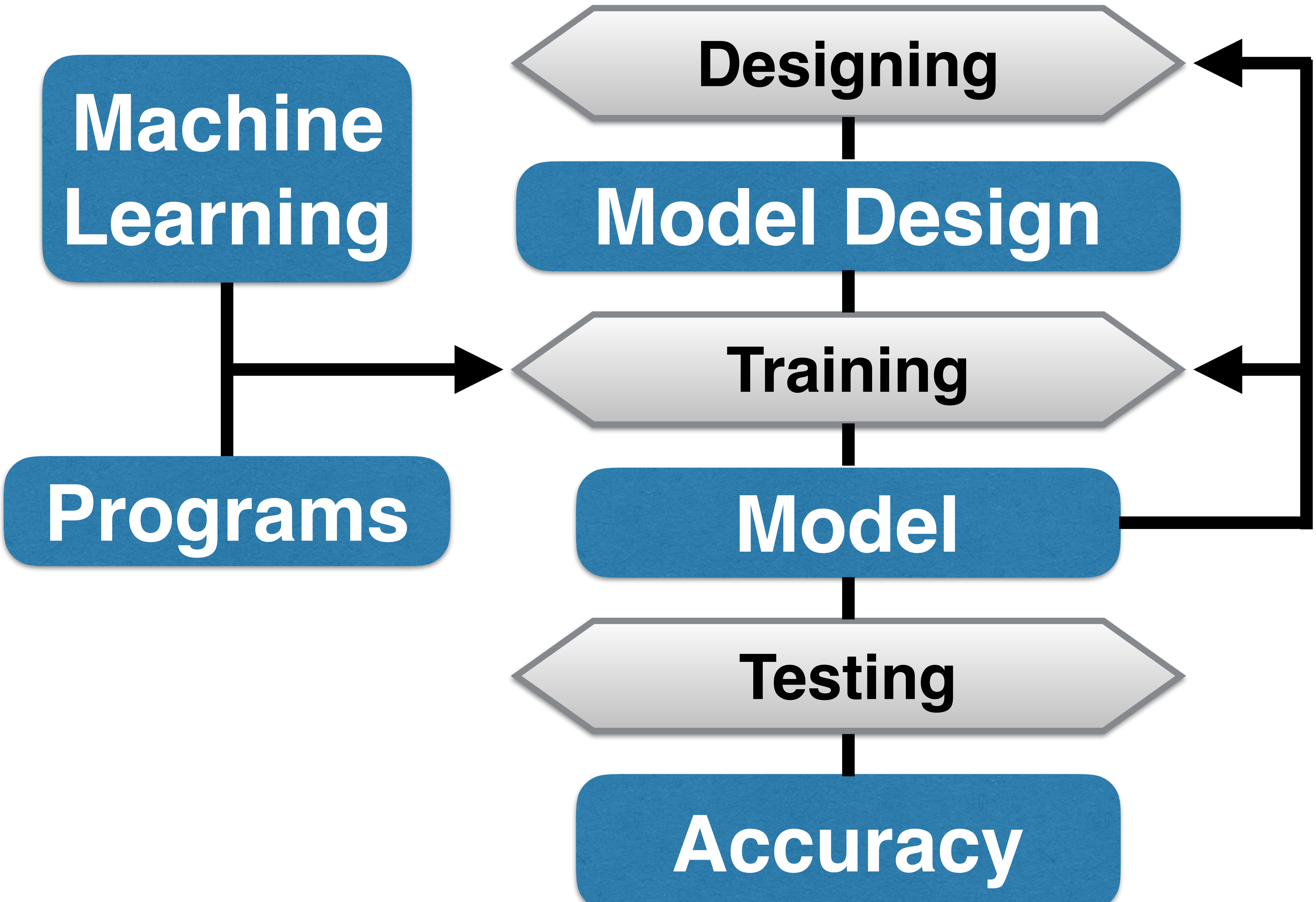
$$T = \sum_{c \in \mathcal{C}} n_c T_c$$



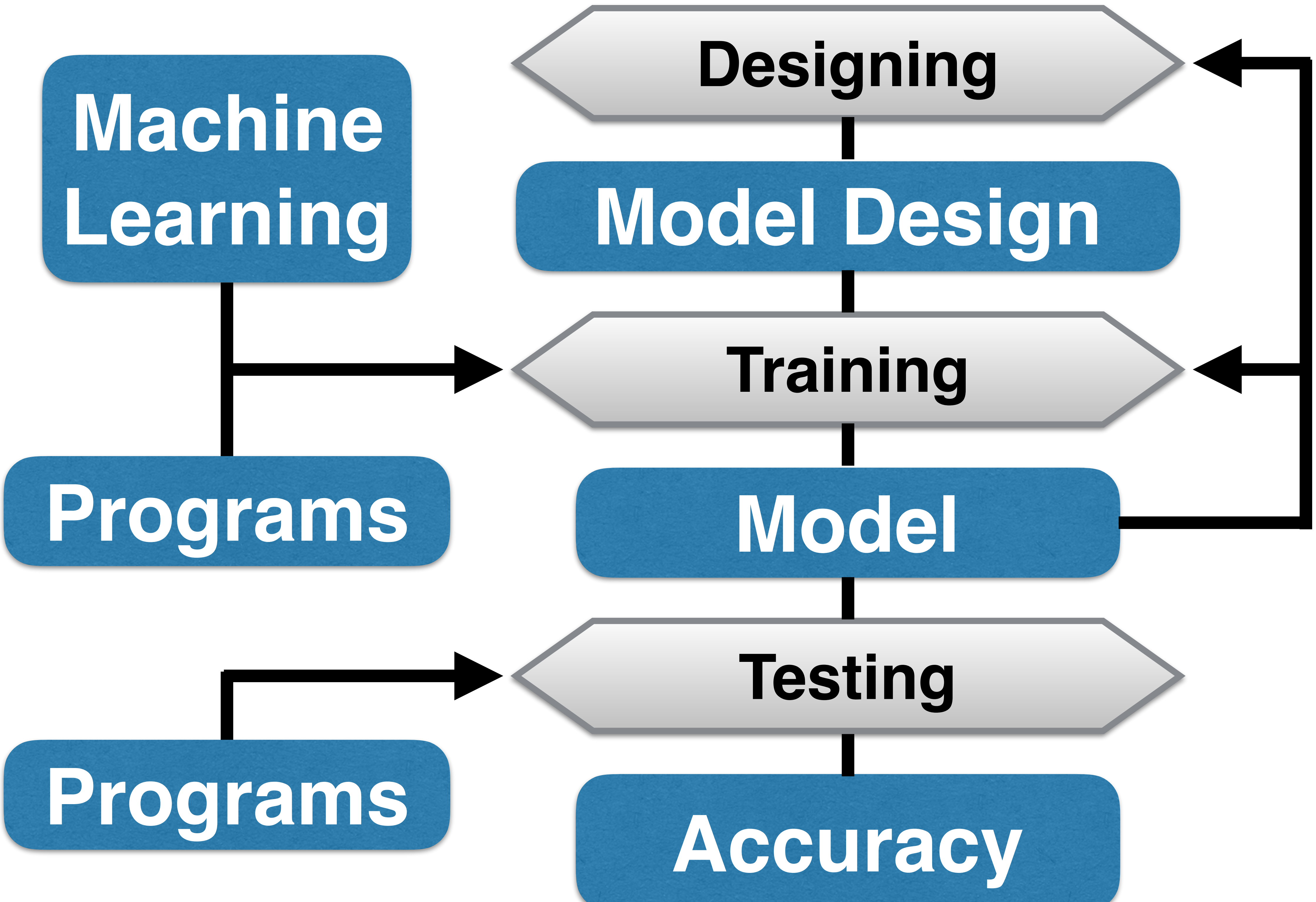
$$T = \sum_{c \in \mathcal{C}} n_c T_c$$



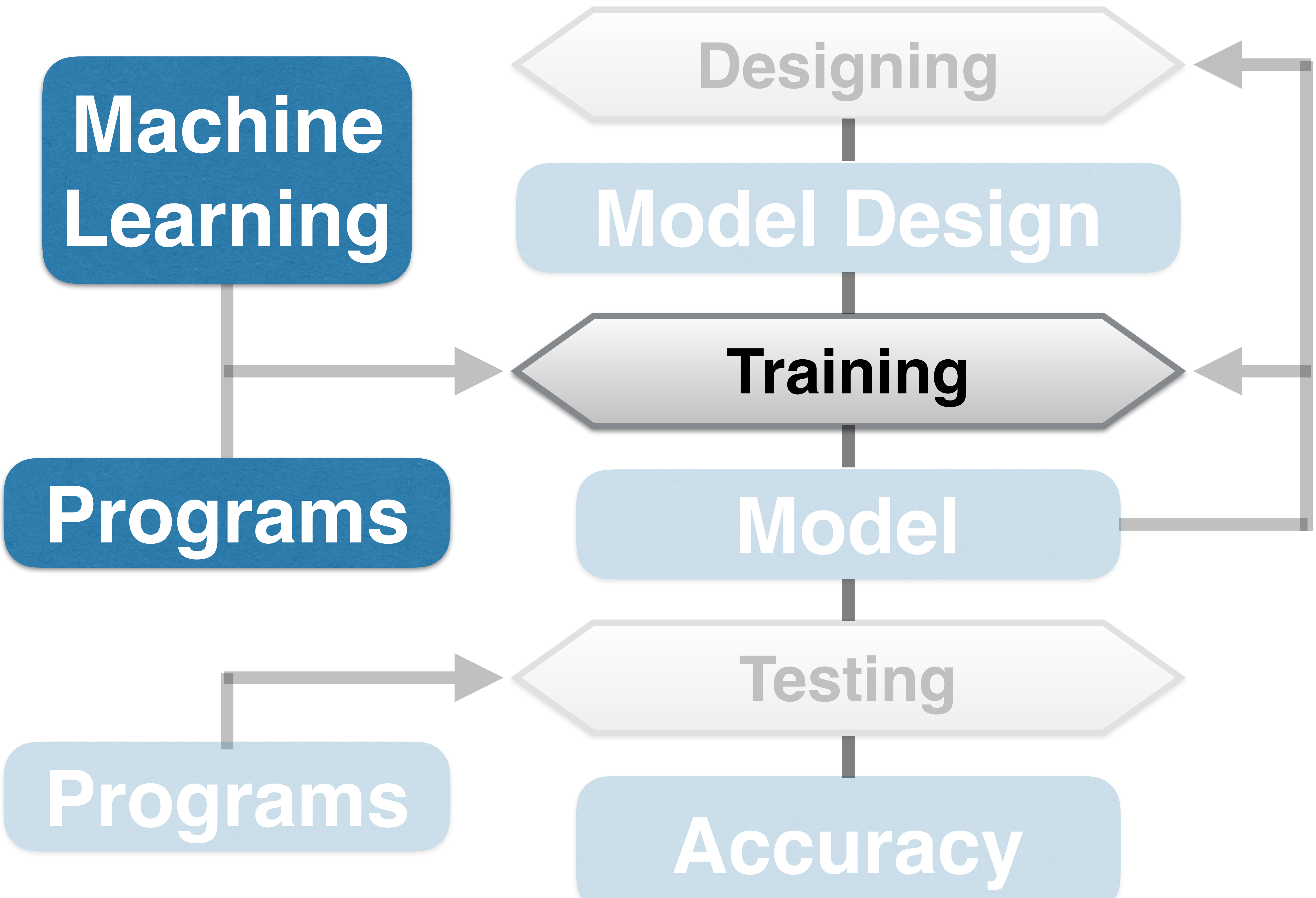
$$T = \sum_{c \in \mathcal{C}} n_c T_c$$



$$T = \sum_{c \in \mathcal{C}} n_c T_c$$



$$T = \sum_{c \in \mathcal{C}} n_c T_c$$



Training Programs

- One program per construct
- Execute with different values of `x` and `y`

```
let rec fadd x y =
  if (x = 0) then 0
  else y + y + y + fadd (x-1) y;;
```

Training Programs

- One program per construct
- Execute with different values of `x` and `y`

```
let rec fadd x y =
  if (x = 0) then 0
  else y + y + y + fadd (x-1) y;;
```

$$V \vdash e \Downarrow v \mid t$$

Interpreter

Linear Regression

ARM intel

$V \vdash e \Downarrow v \mid t$

Interpreter

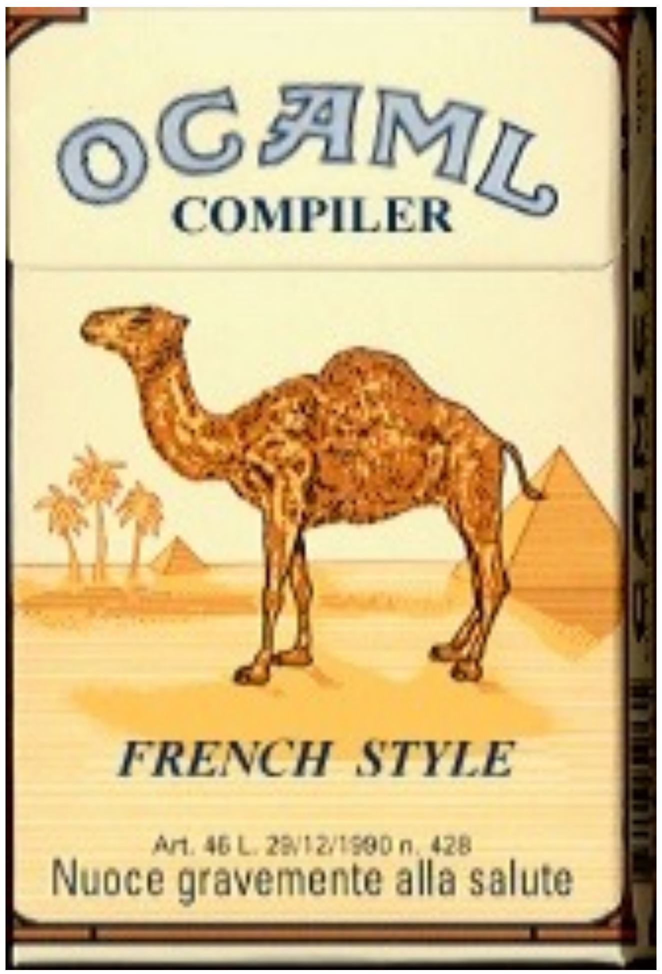
$$T = \sum_{c \in \mathcal{C}} n_c T_c$$

Learn

Minimize

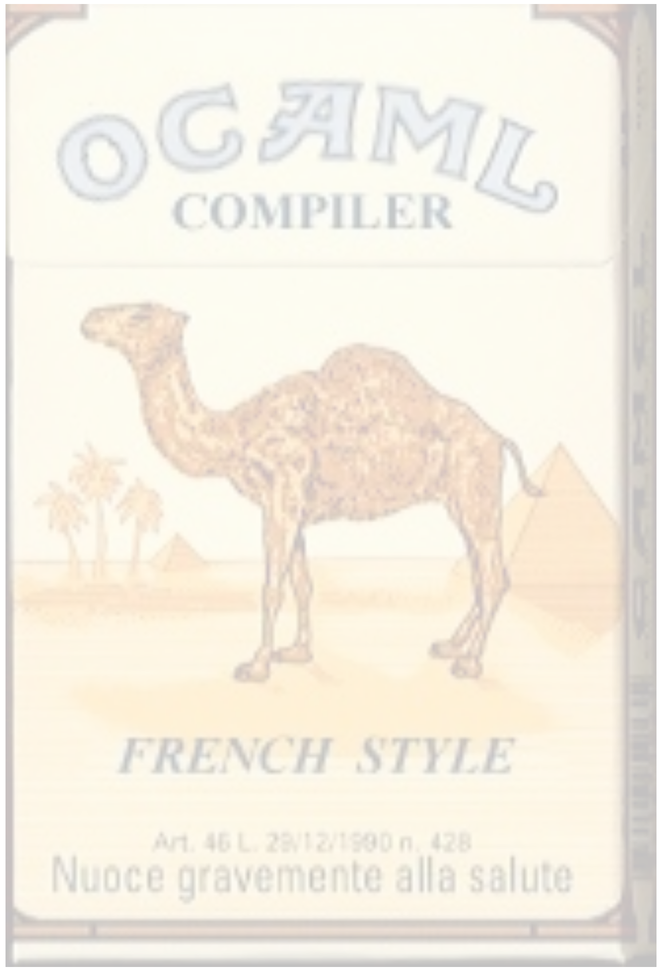
$$\left(T - \sum_{c \in \mathcal{C}} n_c T_c \right)^2$$

What about the challenges?



Compiler Optimizations **Garbage Collector**

What about the challenges?



Compiler Optimizations

Garbage Collector

Simplified GC model

- Only model the minor heap
- Each GC cycle starts with the full heap and ends with the empty heap
- All GC cycles take the same time
- Number of cycles is heap allocations divided by minor heap size

GC model

GC time $\longrightarrow \left\lfloor \frac{M}{H_0} \right\rfloor T_{gc}$

M → **Memory allocations of program**

T_{gc} → **Time for 1 minor GC cycle**

H_0 → **Size of Minor Heap**

Heap Allocations

- **Learn time without GC**
- **Use same interpreter for number of allocations**

$$V \vdash e \Downarrow v \mid t$$

Interpreter

$$(T, M) = \left(\sum_{c \in \mathcal{C}} n_c T_c, \sum_{c \in \mathcal{C}} n_c M_c \right)$$

Time with GC

$$T = \sum_{c \in \mathcal{C}} n_c T_c + \left\lfloor \frac{\sum_{c \in \mathcal{C}} n_c M_c}{H_0} \right\rfloor T_{gc}$$

Time with GC

$$T = \sum_{c \in \mathcal{C}} n_c T_c + \left\lfloor \frac{\sum_{c \in \mathcal{C}} n_c M_c}{H_0} \right\rfloor T_{gc}$$

without GC

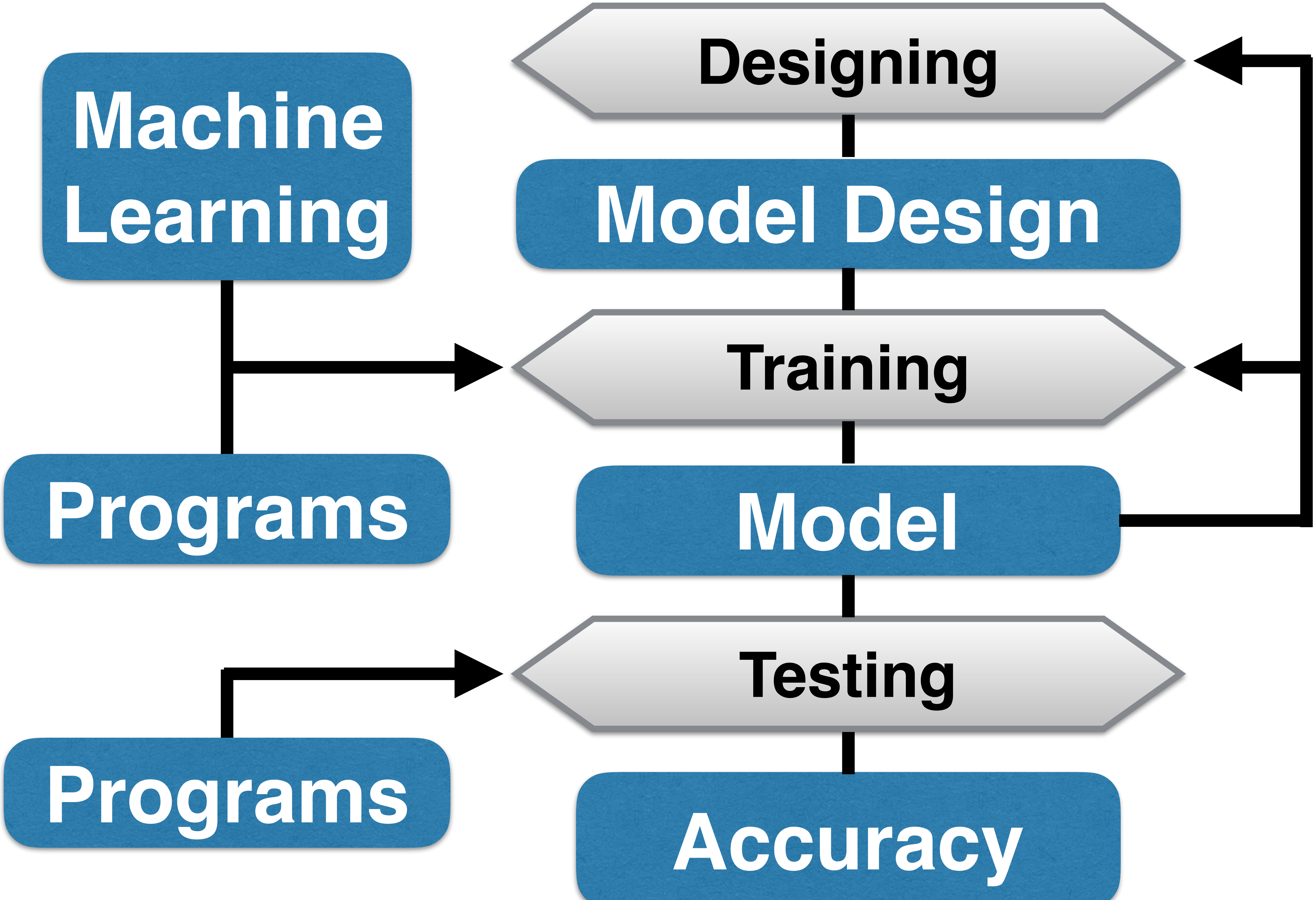
Time with GC

$$T = \sum_{c \in \mathcal{C}} n_c T_c + \left\lfloor \frac{\sum_{c \in \mathcal{C}} n_c M_c}{H_0} \right\rfloor T_{gc}$$

without GC

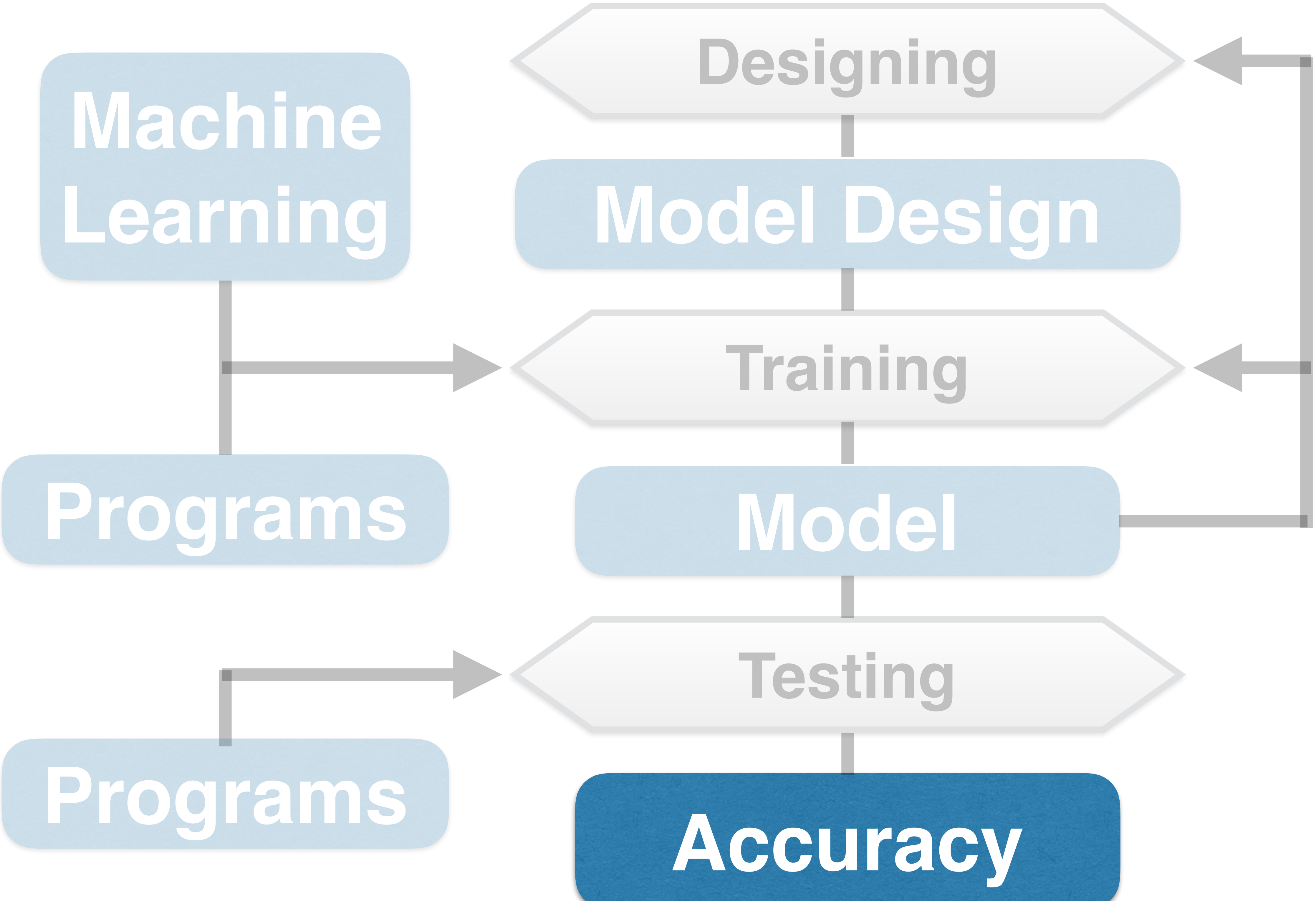
GC time

Cost Model

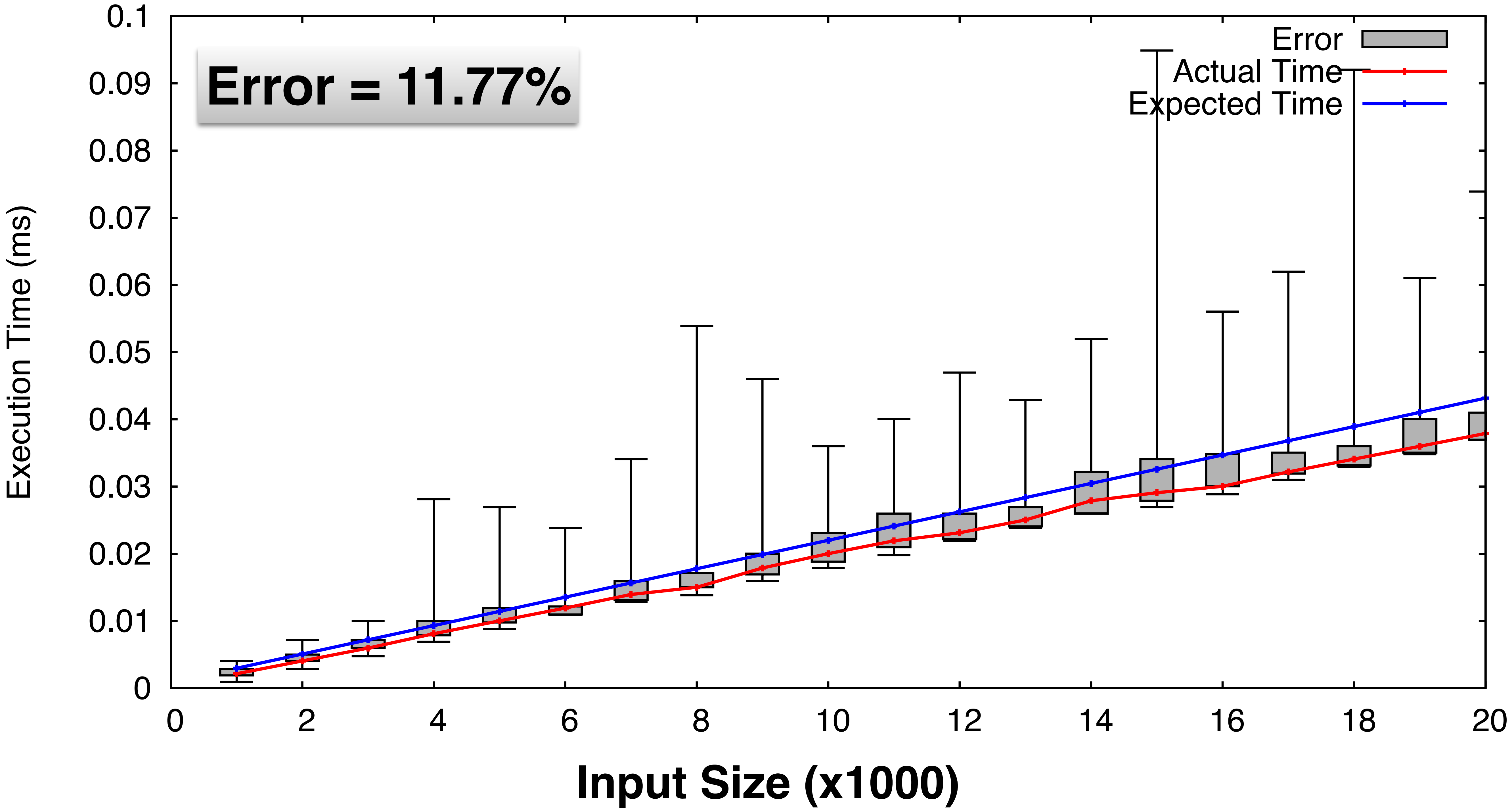


Machine Learning
Designing
Model Design
Training
Programs
Model
Testing
Programs
Accuracy

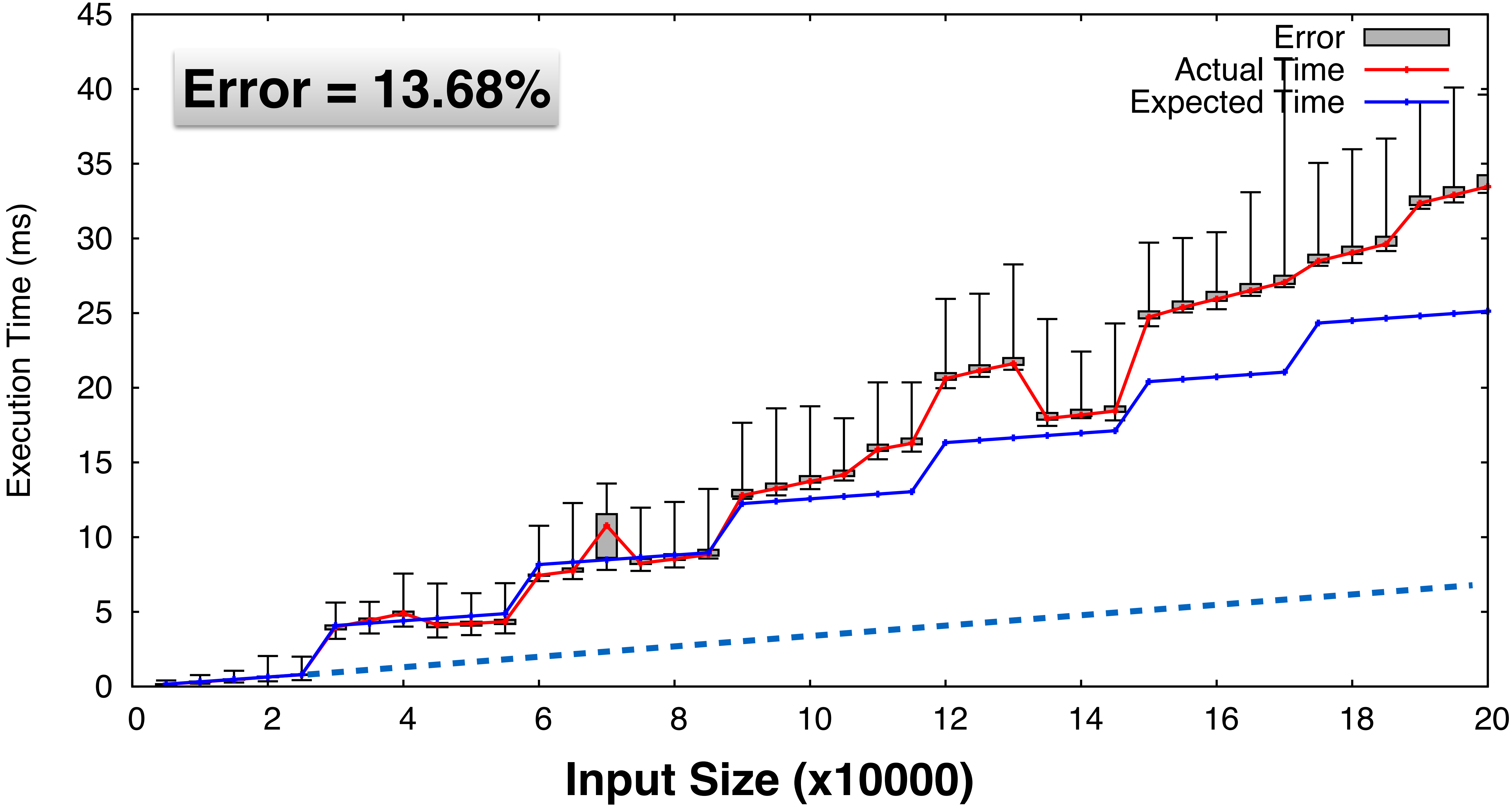
Cost Model


Designing
Machine Learning
Model Design
Training
Programs
Model
Testing
Programs
Accuracy

Factorial


Error = 11.77%
Error
Actual Time
Expected Time
Execution Time (ms)
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
0
2
4
6
8
10
12
14
16
18
20
Input Size (x1000)

Append



Error = 13.68%
Error
Actual Time
Expected Time
Execution Time (ms)
Input Size (x10000)
0
5
10
15
20
25
30
35
40
45
0
2
4
6
8
10
12
14
16
18
20

Applications

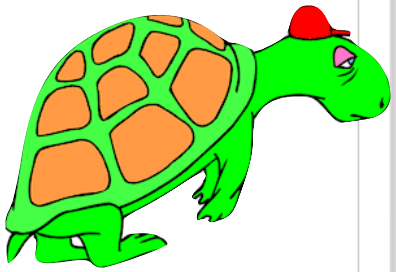
Which one's faster?

```
let rec append l1 l2 =
  match l1 with
  | [] -> l2
  | hd::tl -> hd::(append tl l2);;
```

```
let rec append l1 l2 =
  match l1 with
  | [] -> l2
  | x::[] -> x::l2
  | x::y::[] -> x::y::l2
  | x::y::tl -> x::y::(append tl l2);;
```

Which one's faster?

```
let rec append l1 l2 =
  match l1 with
  | [] -> l2
  | hd::tl -> hd::(append tl l2);;
```



```
let rec append l1 l2 =
  match l1 with
  | [] -> l2
  | x::[] -> x::l2
  | x::y::[] -> x::y::l2
  | x::y::tl -> x::y::(append tl l2);;
```

Symbolic Bounds (RAML)

Program	Time Bound (ns)
append	$0.45 + 11.28M + \left\lfloor \frac{24M}{2097448} \right\rfloor \times 3125429.15$
map	$0.60 + 13.16M + \left\lfloor \frac{24M}{2097448} \right\rfloor \times 3125429.15$
insertion sort	$0.45 + 6.06M + 5.83M^2 + \left\lfloor \frac{12M + 12M^2}{2097448} \right\rfloor \times 3125429.15$

Conclusion

- Cost model for execution time and heap allocations
- Learned hardware specific constants
- Added a simple model for the garbage collector
- Roughly 20% on Intel x86 and ARM
- Fast and slow implementations of specification
- Execution time prediction (symbolic bounds)

Conclusion

- Cost model for execution time and heap allocations
- Learned hardware specific constants
- Added a simple model for the garbage collector
- Roughly 20% on Intel x86 and ARM
- Fast and slow implementations of specification
- Execution time prediction (symbolic bounds)

It works!

Learned Cost for Time without GC (x86)

Base : 832.6918

App : 1.5056
App (tail) : 0.1562
Let Const : 2.8280
Let Func : 1.3127
Let Rec : 3.7381
Closure : 2.9210

Pattern Match : 0.2231
Tuple Head : 5.8929
Tuple Elem : 1.7177
Tuple Match : 0.2370

Not op : 0.4242
And op : 0.1843
Or op : 0.1838

Int Add : 0.2972
Int Sub : 0.2781
Int Mult : 1.2992
Int Mod : 19.2316
Int Div : 19.0119
Int Uminus : 0.4196
Int Eq : 0.3826
Int (<) : 0.3818
Int (<=) : 0.3815
Int (>) : 0.3750
Int (>=) : 0.3819

Float Add : 2.1020
Float Sub : 2.1166
Float Mult : 1.7370
Float Div : 8.5757
Float Uminus : 1.2322
Float Eq : 0.5826
Float (<) : 0.6191
Float (<=) : 0.6258
Float (>) : 0.5855
Float (>=) : 0.6295

Heap Consumption (x86)

Base : 96.03

App : 0.00
App (tail) : 0.00
Let Const : 0.00
Let Func : 0.00
Let Rec : 0.00
Pattern Match : 0.00
Tuple Match : 0.00

Fun Def : 24.00
Closure : 7.99
Cons : 24.00

Int Add : 0.00
Int Sub : 0.00
Int Mult : 0.00
Int Mod : 0.00
Int Div : 0.00
Int Uminus : 0.00
Int Eq : 0.00
Int (<) : 0.00
Int (<=) : 0.00
Int (>) : 0.00
Int (>=) : 0.00

Not op : 0.00
And op : 0.00
Or op : 0.00

Tail Call

- Function call on the outermost
- Optimized to “jump” instruction in assembly

```
let f x = 1 + x;;

let g n =
  if (n = 0) then 0 else f n;;

let h n =
  if (n = 0) then 0 else 1 + f n;;
```

Tuples

```
let x = (1, 2, 3)
```

```
let x = 1::(2::[])
```

Tuple Head = 1
Tuple Elem = 3

Tuple Head = 2
Tuple Elem = 4