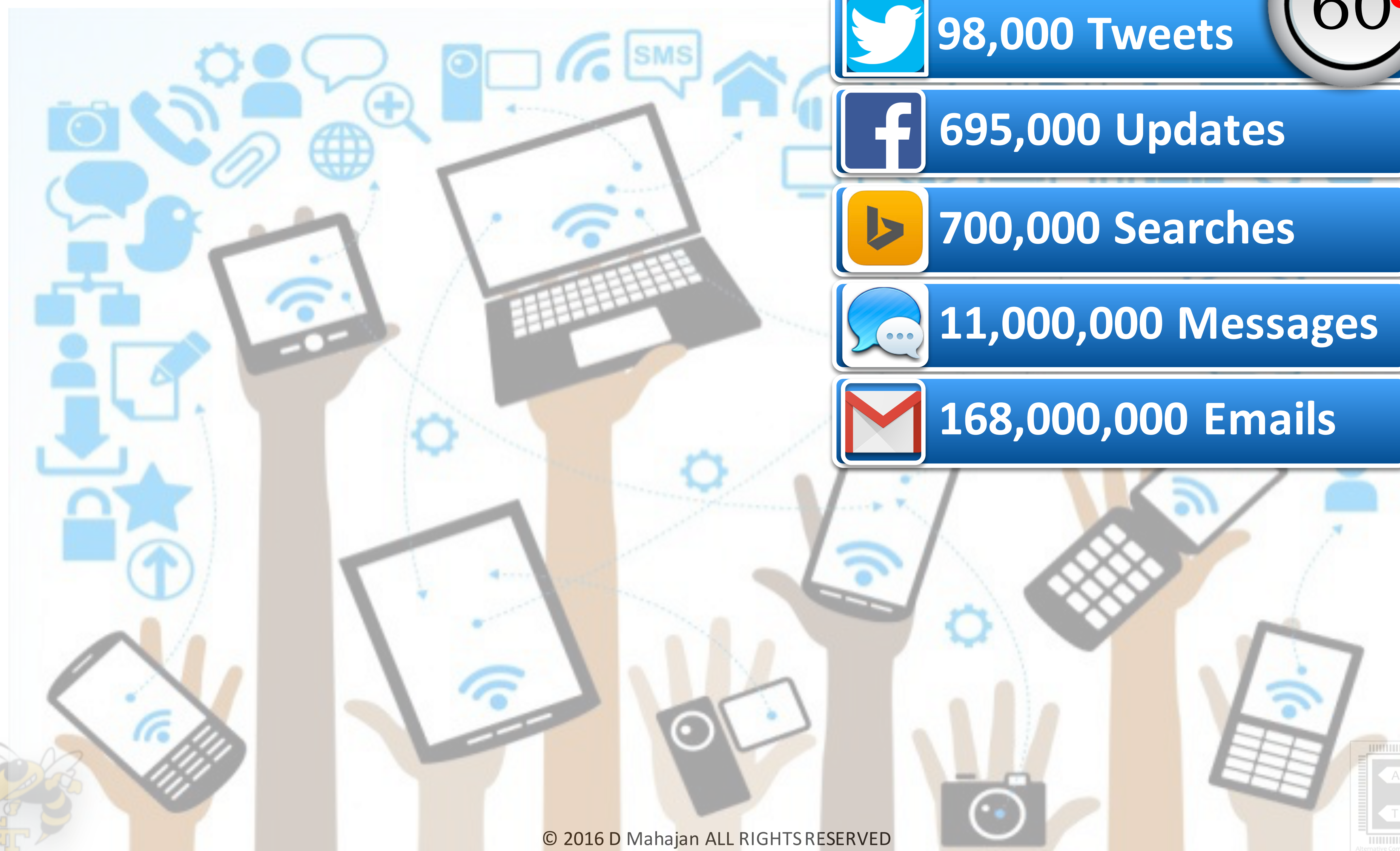
TABLA: A Unified Template-based Framework for Accelerating Statistical Machine Learning

Divya Mahajan
Jongse Park
Emmanuel Amaro
Hardik Sharma
Amir Yazdanbakhsh
Joon Kyung Kim
Hadi Esmaeilzadeh

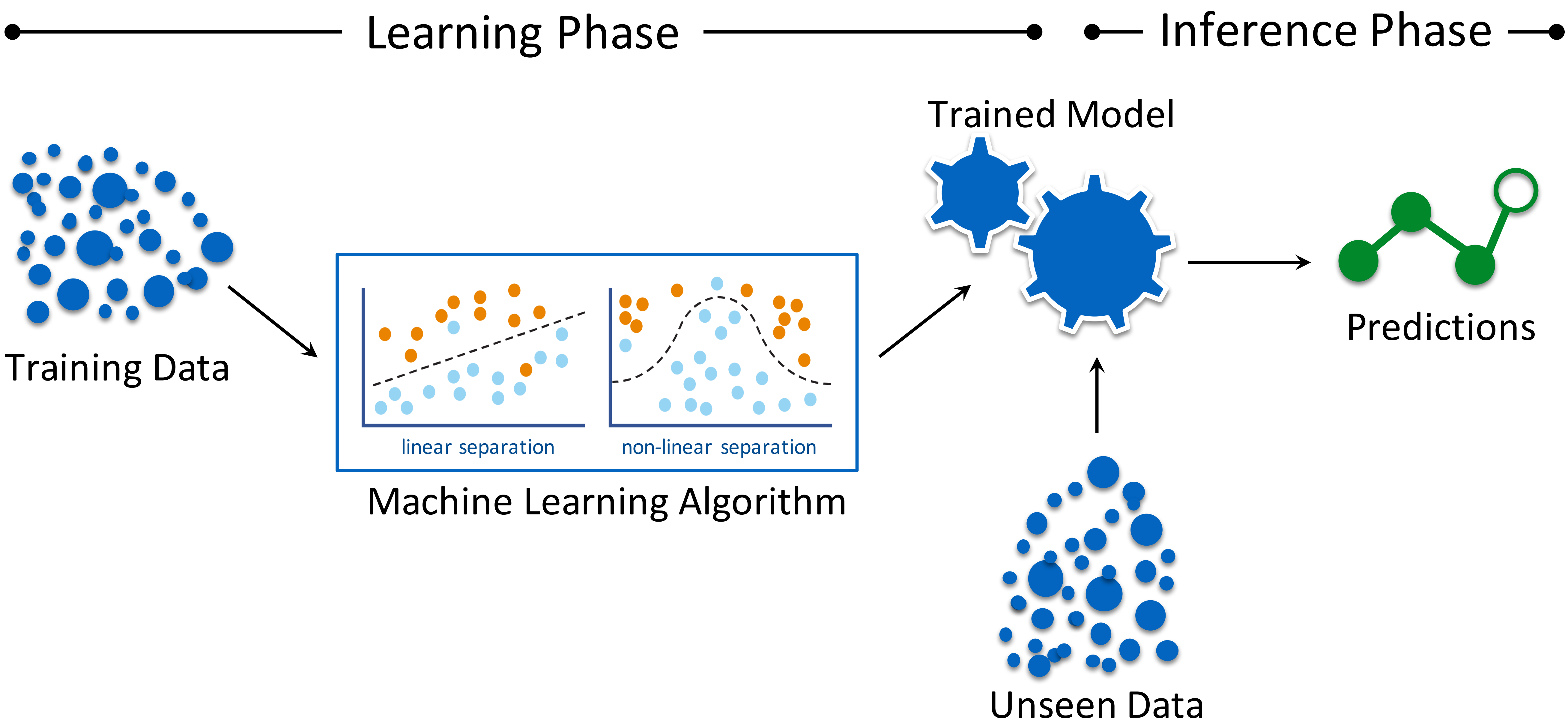
Alternative Computing Technologies (ACT) Lab
Georgia Institute of Technology

Data Explosion

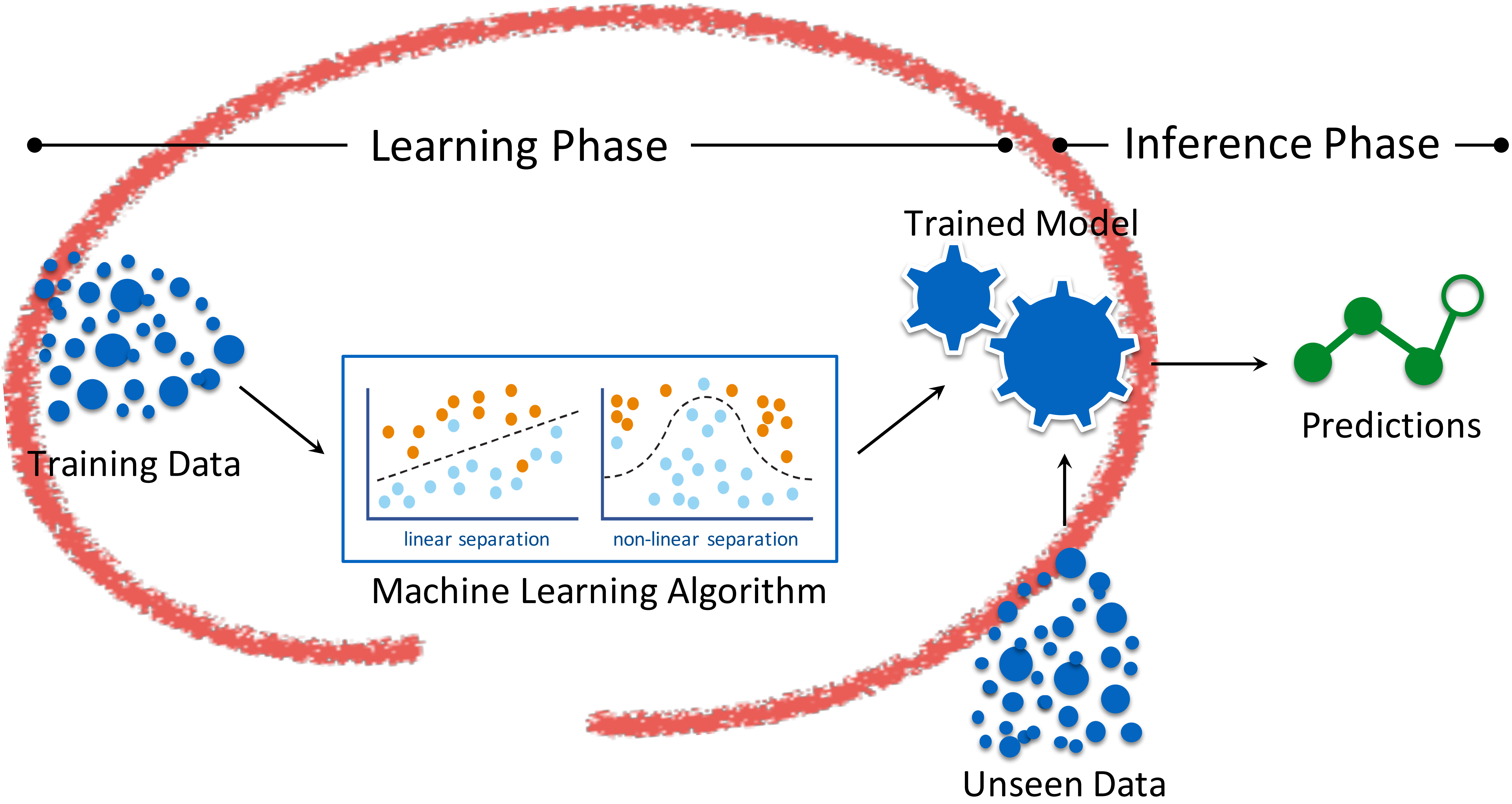


- 98,000 Tweets
- 695,000 Updates
- 700,000 Searches
- 11,000,000 Messages
- 168,000,000 Emails

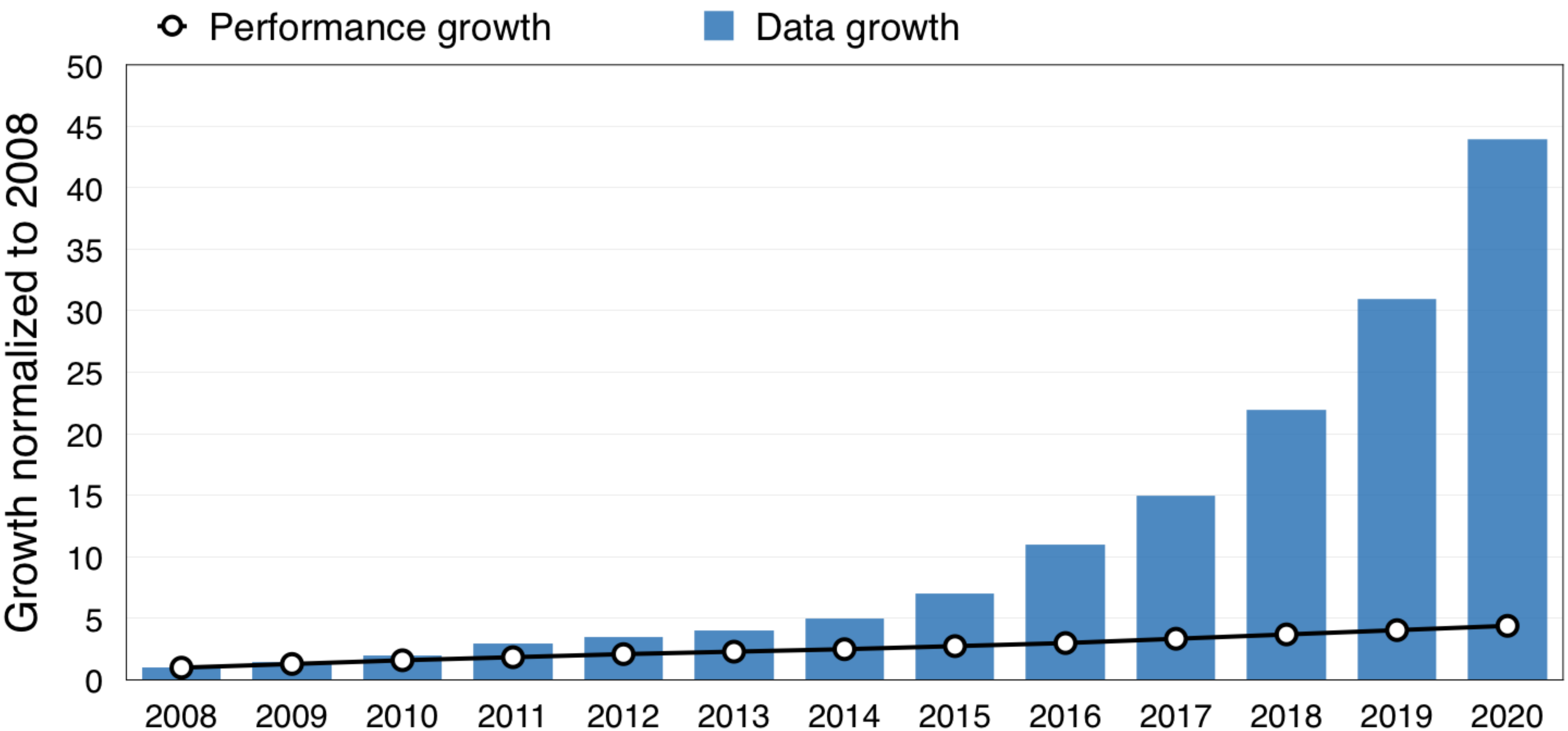
Machines learn to extract insights from data



Machines learn to extract insights from data



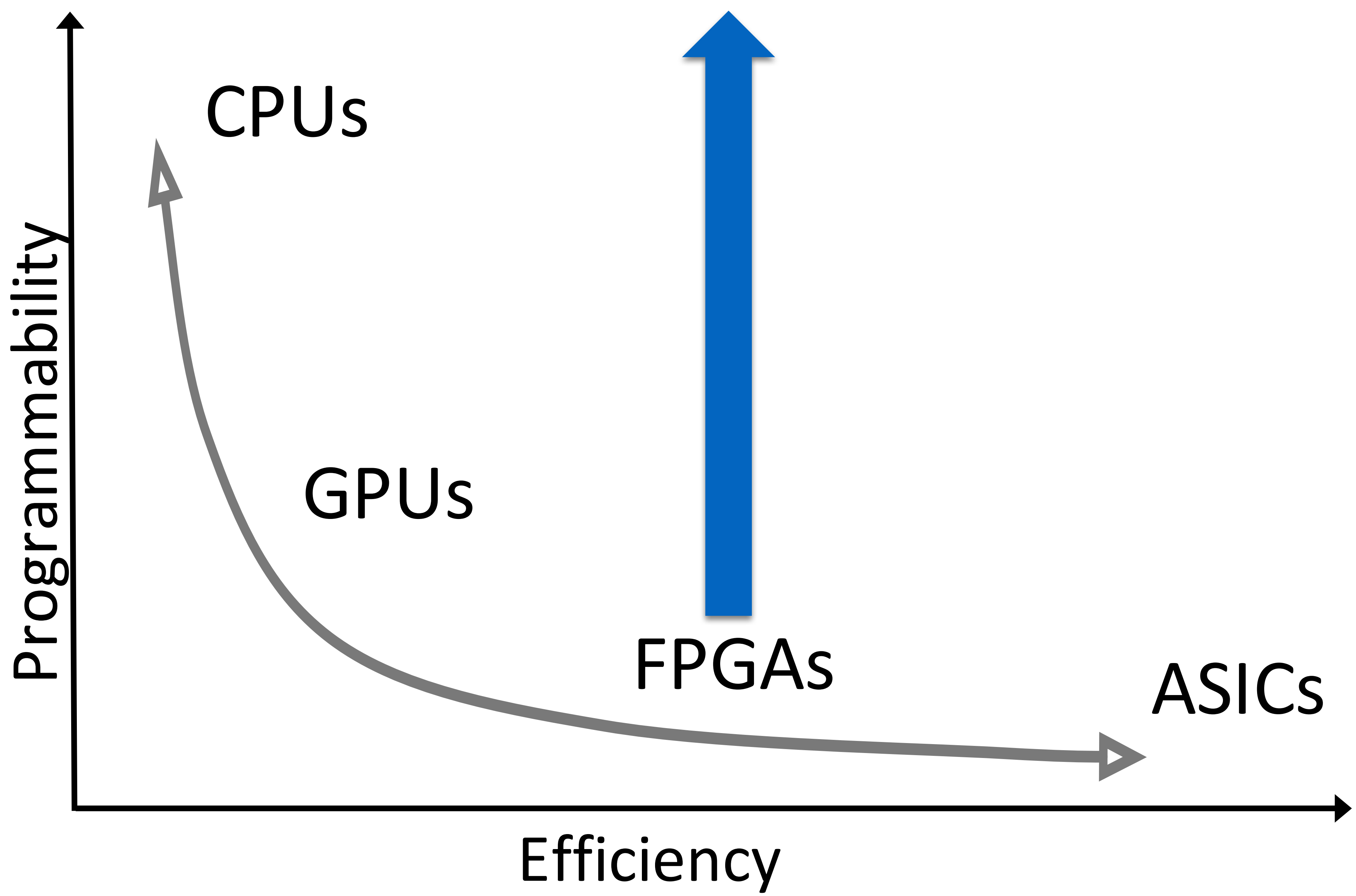
Growing gap between data and compute



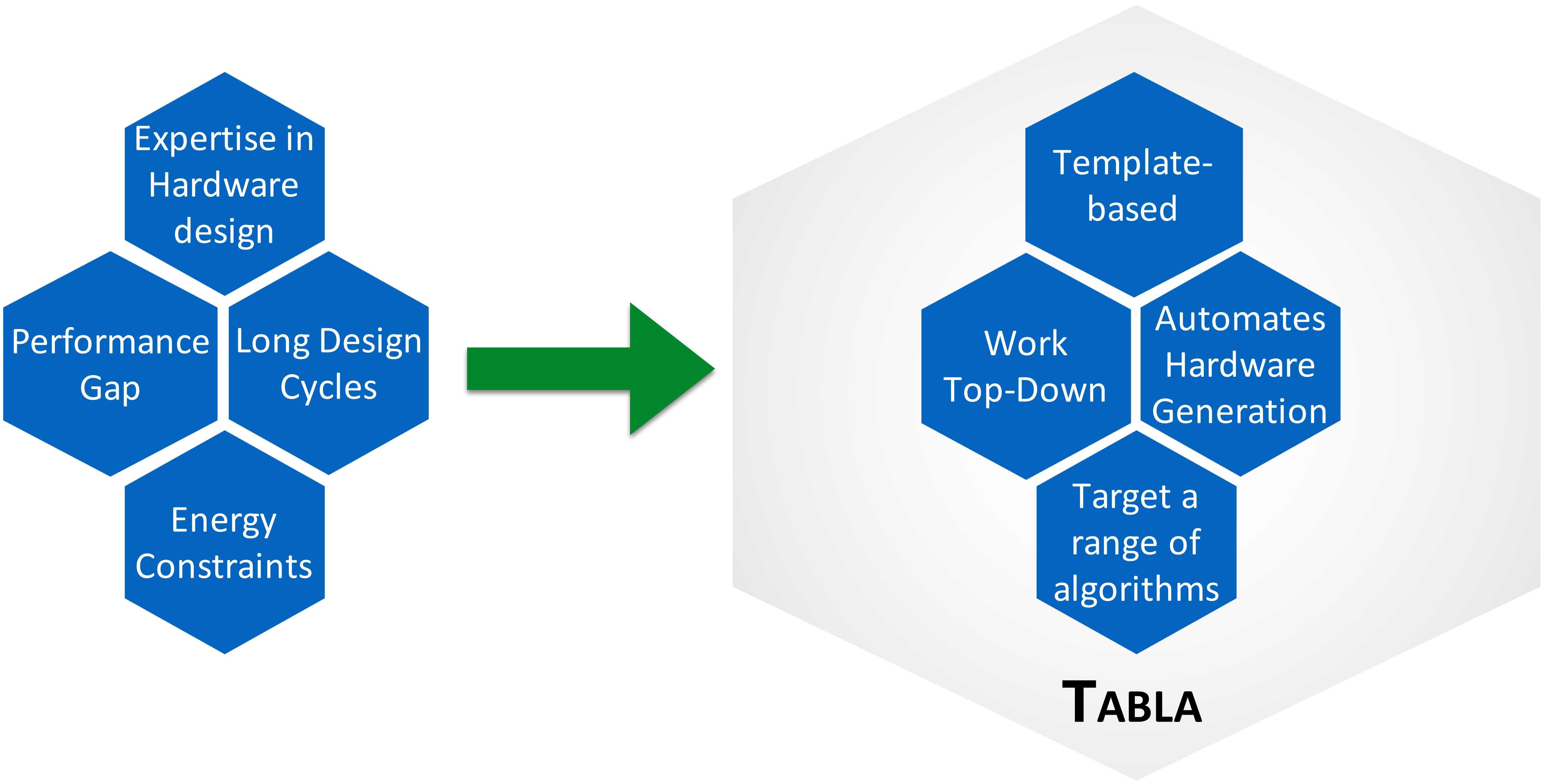
Data growth trends: IDC's Digital Universe Study, December 2012

Performance growth trends: Esmaeilzadeh et al, "Dark Silicon and the End of Multicore Scaling," ISCA 2011

Programmability versus efficiency



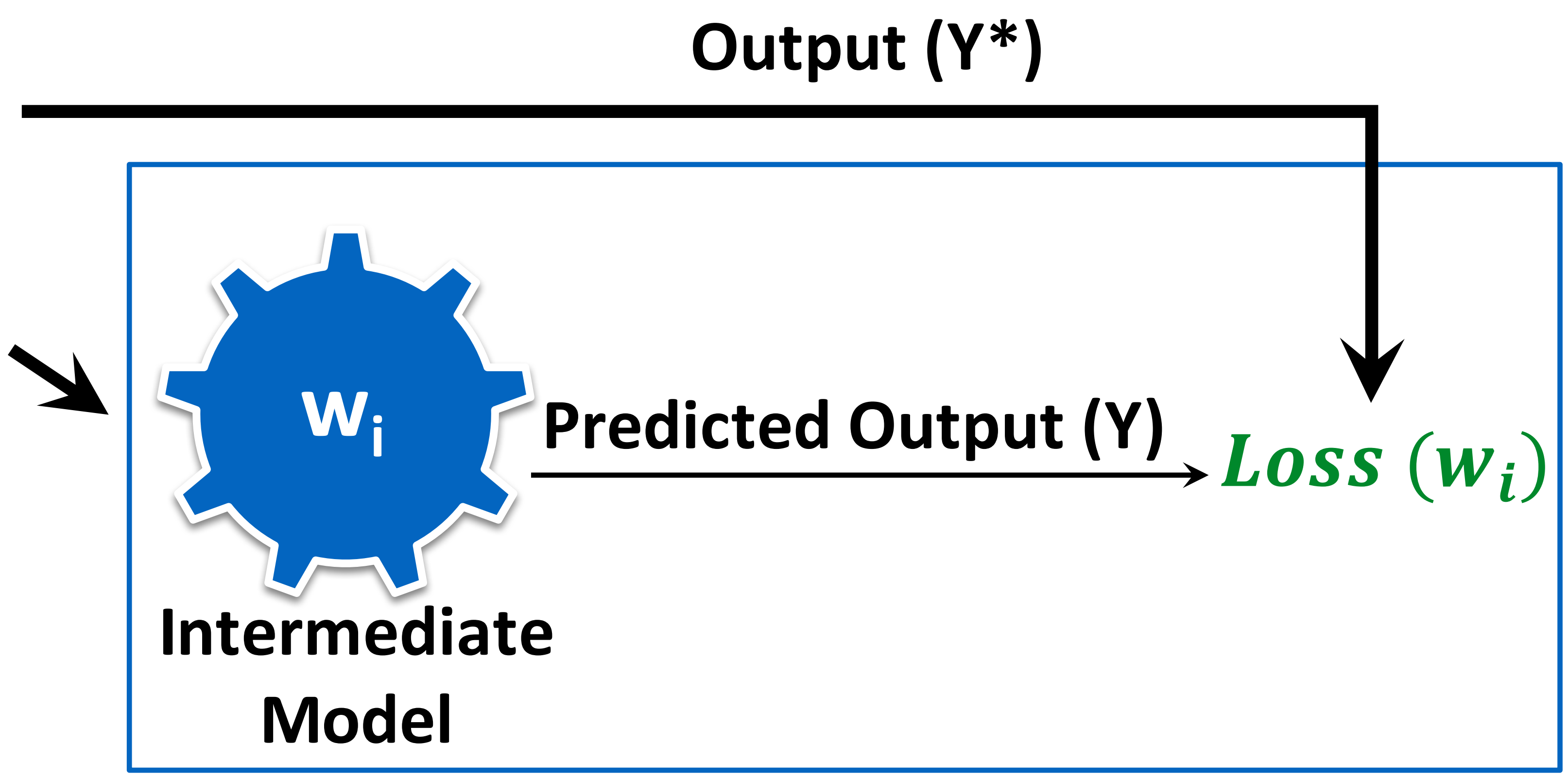
An algorithmic approach towards acceleration



Understanding machine learning

Training Data

Input (X)	Output (Y*)
25,1,76,0	1
6,43,9,93	6
:	:
23,56,2,0	12
12,0,9,0	0



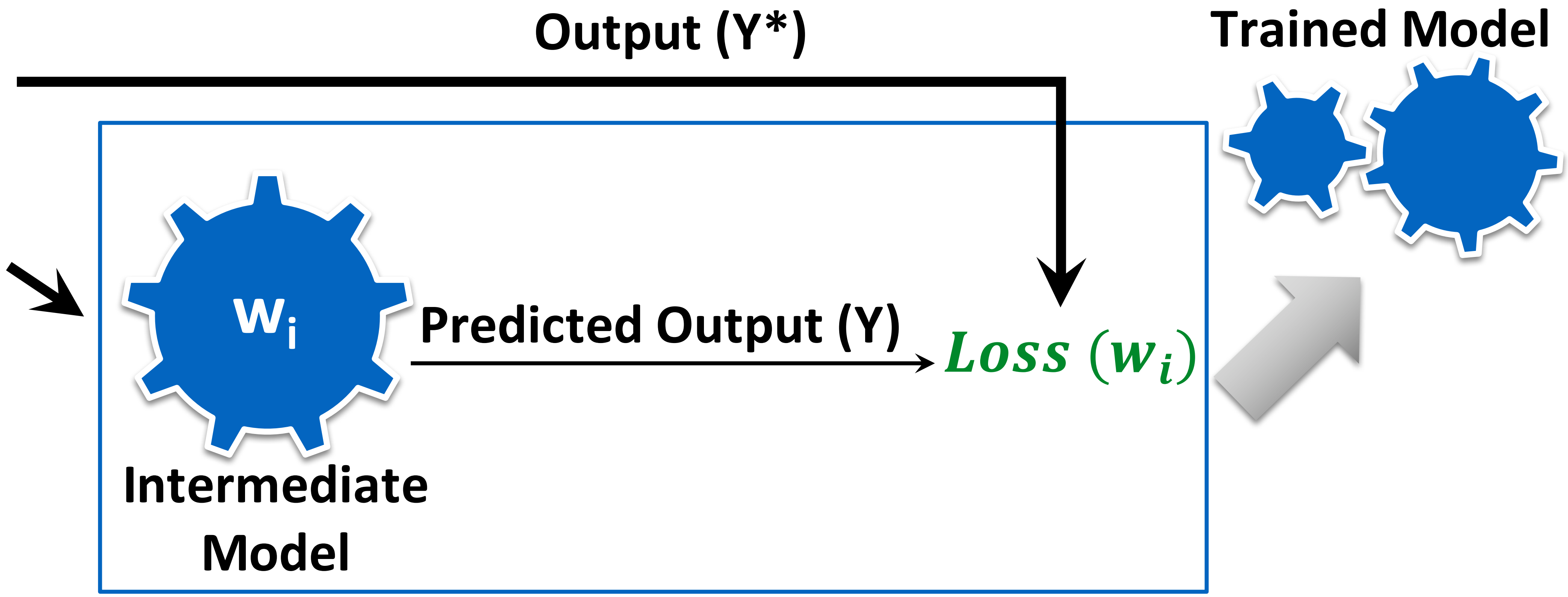
$$Loss(w_i) = \sum_i \|Y - Y^*\|$$

Algorithmic commonalities

Learning is solving an iterative optimization problem!

Training Data

Input (X)	Output (Y*)
25,1,76,0	1
6,43,9,93	6
:	:
23,56,2,0	12
12,0,9,0	0

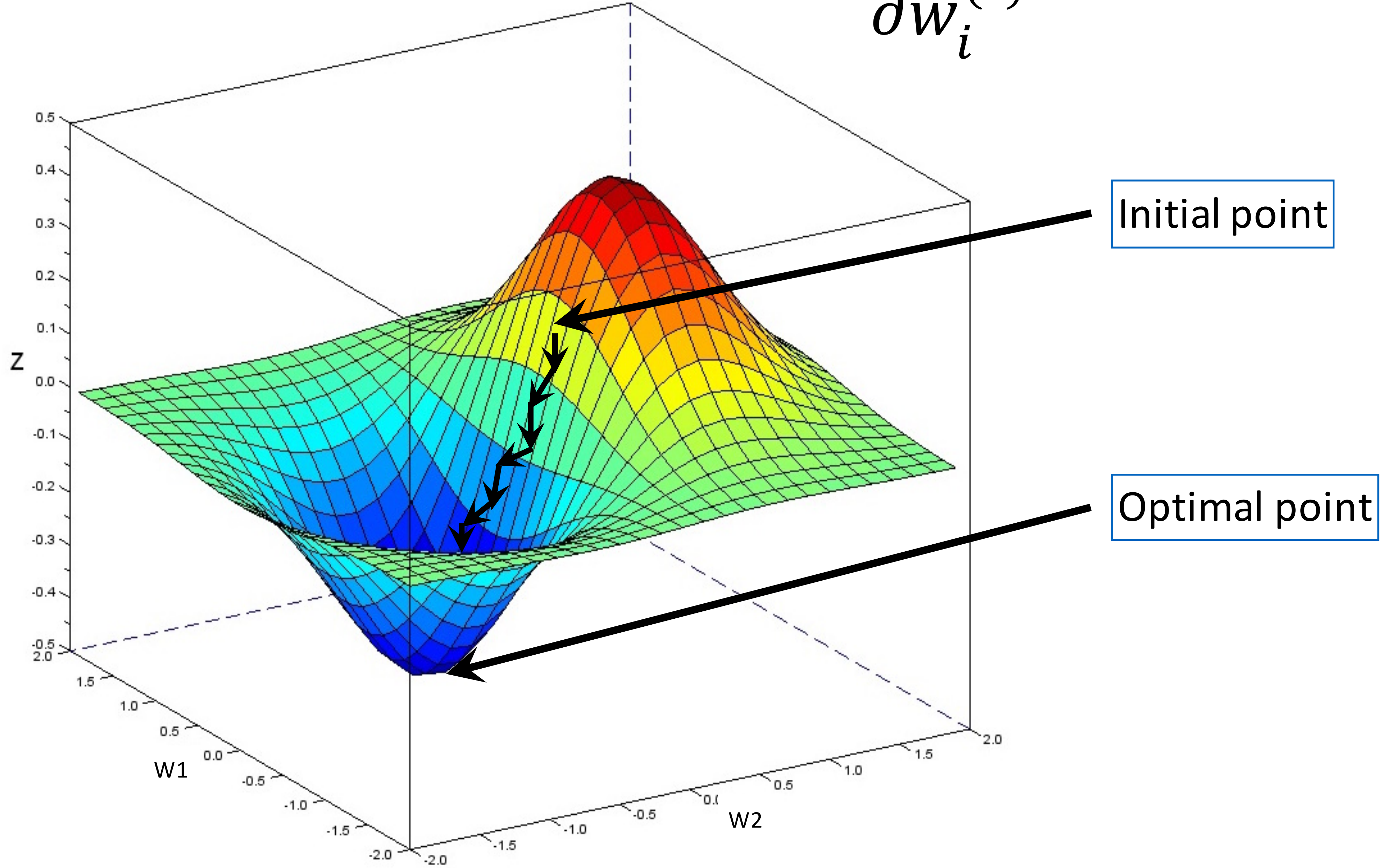


$$find(w_i) \ni \{Loss(w_i) = \sum_i \|Y - Y^*\|\}\ \textit{is minimized}$$

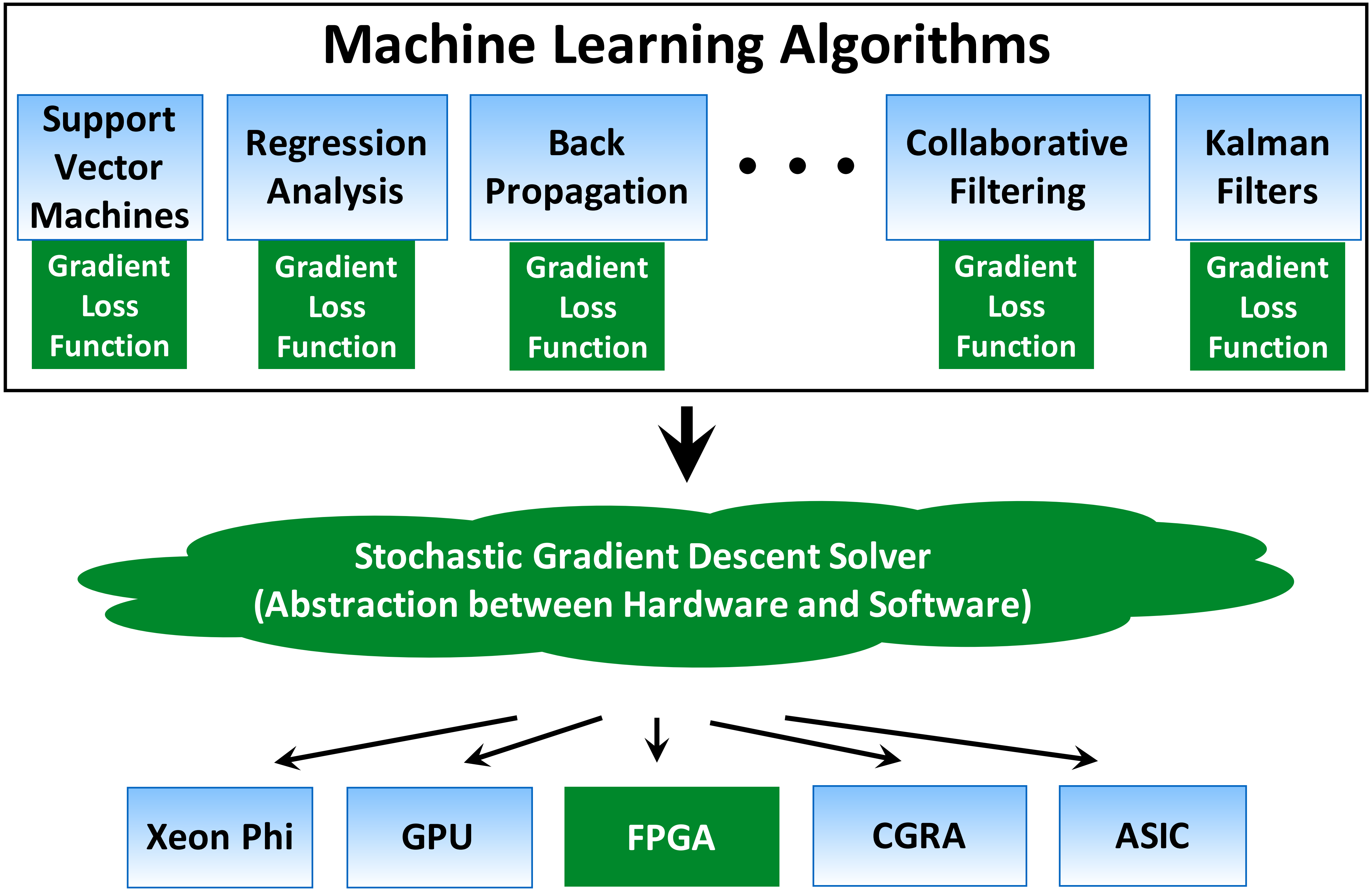
Stochastic gradient descent solver

$$Loss\ function\ (w_i)\ = f\left(w_1^{(t)}, w_2^{(t)}, \ldots.., w_m^{(t)}\right)$$

$$w_i^{(t+1)} = w_i^t - u \times \frac{\partial f\left(w_1^{(t)}, w_2^{(t)}, \ldots.., w_m^{(t)}\right)}{\partial w_i^{(t)}}$$



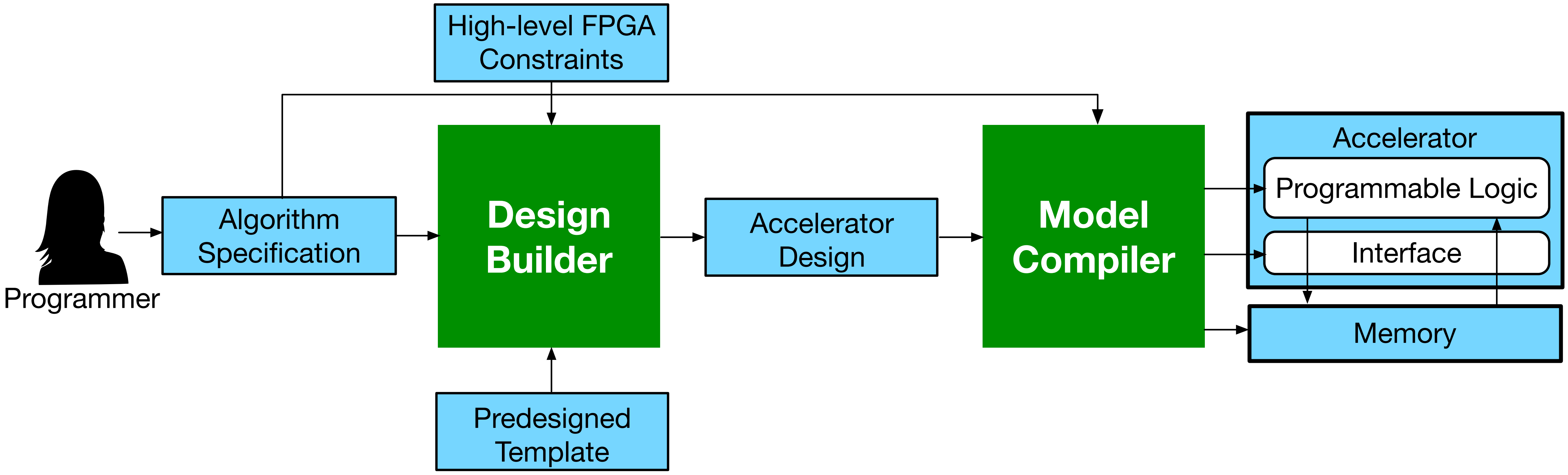
Abstraction between hardware and software



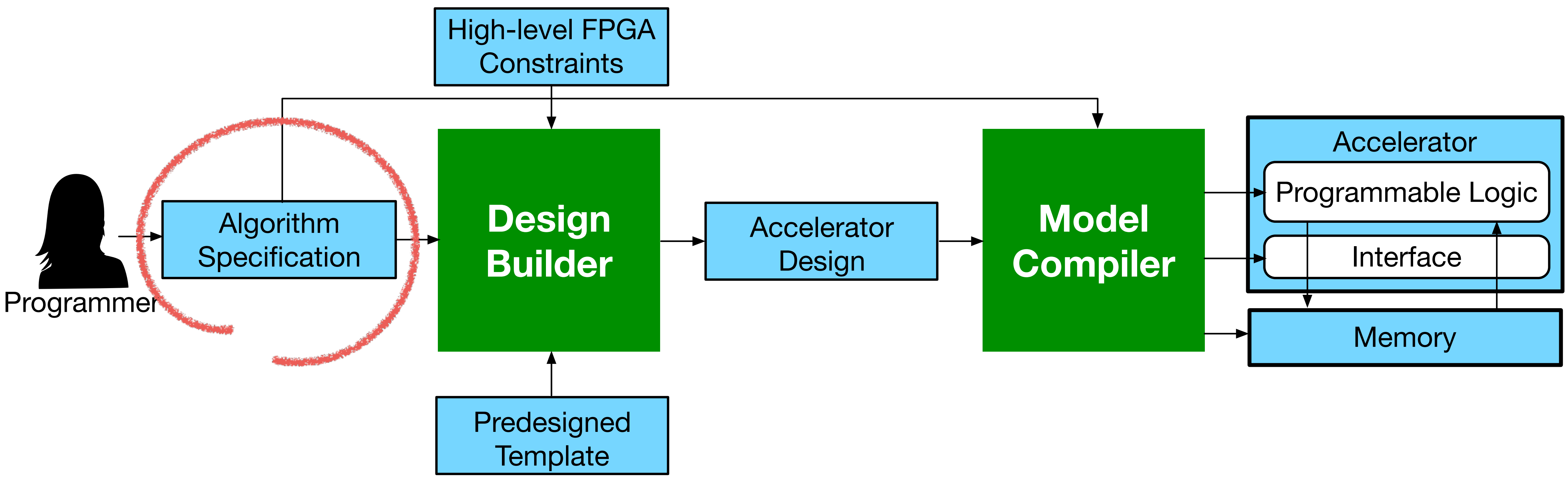
Cross-stack template-based approach

- Programming Language
- Compilation
- Template Architectures
- Accelerator design

TABLA workflow

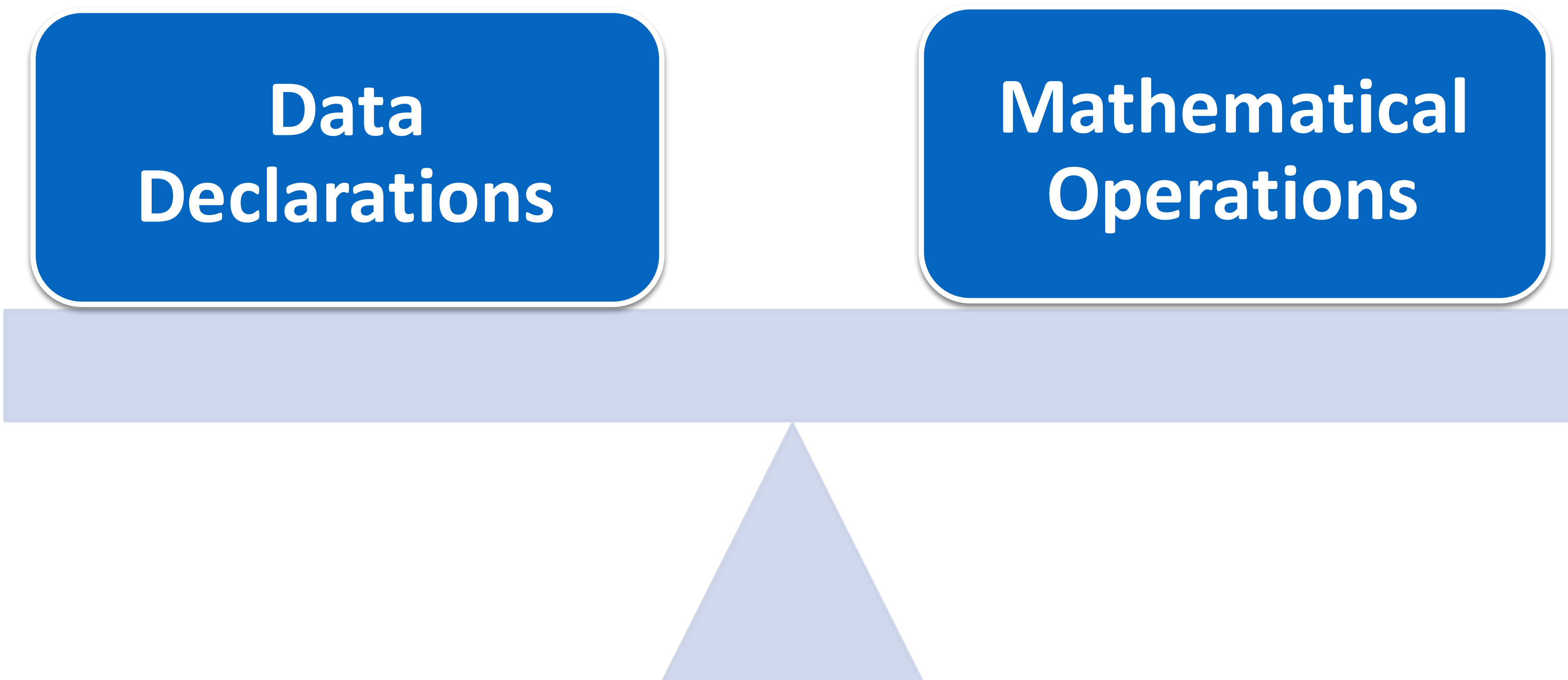


Programming interface



Enables programmer to specify the gradient of loss function

Programming interface



Example: Logistic Regression

$$Loss\ Function\ Gradient(W^T) = \left\{\left(\left(sigmoid\left(\sum_i W^T X_i\right)\right) - Y\right) \cdot Xi\right\} + lambda \cdot W^T$$

```
m = 53
lambda = 0.1

model_input   X[m];
model_output  Y';
model         W[m];
gradient      G[m];

iterator i[0:m - 1];

S = sum [i](X[i] * W[i]);

Y = sigmoid(S);
E = Y - Y';
G[i] = X[i] * E;
V[i] = lambda * W[i];
G[i] = G[i] + V[i];
```

Example: Logistic Regression

$$Loss\ Function\ Gradient(W^T) = \left\{\left(\left(sigmoid\left(\sum_i W^T X_i\right)\right) - Y\right) \cdot Xi\right\} + lambda \cdot W^T$$

```
m = 53
lambda = 0.1

model_input   X[m];
model_output  Y';
model         W[m];
gradient      G[m];

iterator i[0:m - 1];

S = sum [i](X[i] * W[i]);

Y = sigmoid(S);
E = Y - Y';
G[i] = X[i] * E;
V[i] = lambda * W[i];
G[i] = G[i] + V[i];
```

Example: Logistic Regression

$$Loss\ Function\ Gradient(W^T) = \left\{ \left(\left(sigmoid\left(\sum_i W^T X_i \right) \right) - Y \right) \cdot Xi \right\} + lambda\ \cdot W^T$$

```
m = 53
lambda = 0.1

model_input   X[m];
model_output  Y';
model         W[m];
gradient      G[m];

iterator i[0:m - 1];

S = sum [i](X[i] * W[i]);

Y = sigmoid(S);
E = Y - Y';
G[i] = X[i] * E;
V[i] = lambda * W[i];
G[i] = G[i] + V[i];
```

Example: Logistic Regression

$$Loss\ Function\ Gradient(W^T) = \left\{\left(\left(sigmoid\left(\sum_i W^T X_i\right)\right) - Y\right) \cdot Xi\right\} + lambda \cdot W^T$$

```
m = 53
lambda = 0.1

model_input   X[m];
model_output  Y';
model         W[m];
gradient      G[m];

iterator i[0:m - 1];

S = sum [i](X[i] * W[i]);

Y = sigmoid(S);
E = Y - Y';
G[i] = X[i] * E;
V[i] = lambda * W[i];
G[i] = G[i] + V[i];
```

Example: Logistic Regression

$$Loss\ Function\ Gradient(W^T) = \left\{\left(\left(sigmoid\left(\sum_i W^T X_i\right)\right) - Y\right) \cdot Xi\right\} + lambda \cdot W^T$$

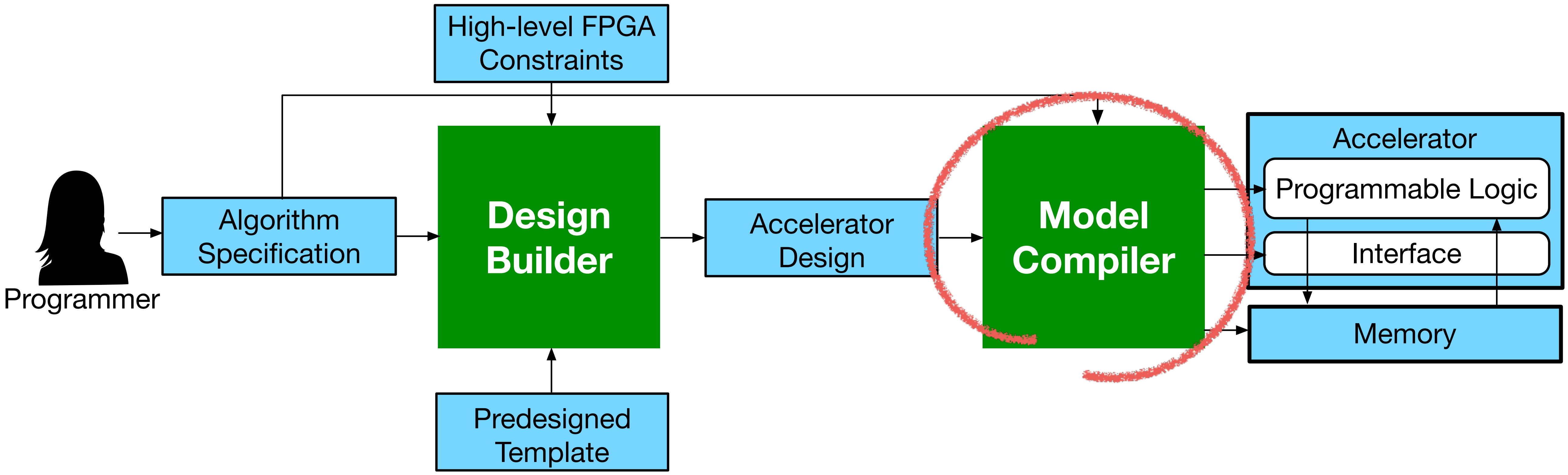
```
m = 53
lambda = 0.1

model_input   X[m];
model_output  Y';
model         W[m];
gradient      G[m];

iterator i[0:m - 1];

S = sum [i](X[i] * W[i]);

Y = sigmoid(S);
E = Y - Y';
G[i] = X[i] * E;
V[i] = lambda * W[i];
G[i] = G[i] + V[i];
```

Example: Logistic Regression

$$Loss\ Function\ Gradient(W^T) = \left\{\left(\left(sigmoid\left(\sum_i W^T X_i\right)\right) - Y\right) \cdot Xi\right\} + lambda \cdot W^T$$

```
m = 53
lambda = 0.1

model_input   X[m];
model_output  Y';
model         W[m];
gradient      G[m];

iterator i[0:m - 1];

S = sum [i](X[i] * W[i]);

Y = sigmoid(S);
E = Y - Y';
G[i] = X[i] * E;
V[i] = lambda * W[i];
G[i] = G[i] + V[i];
```

Model Compiler

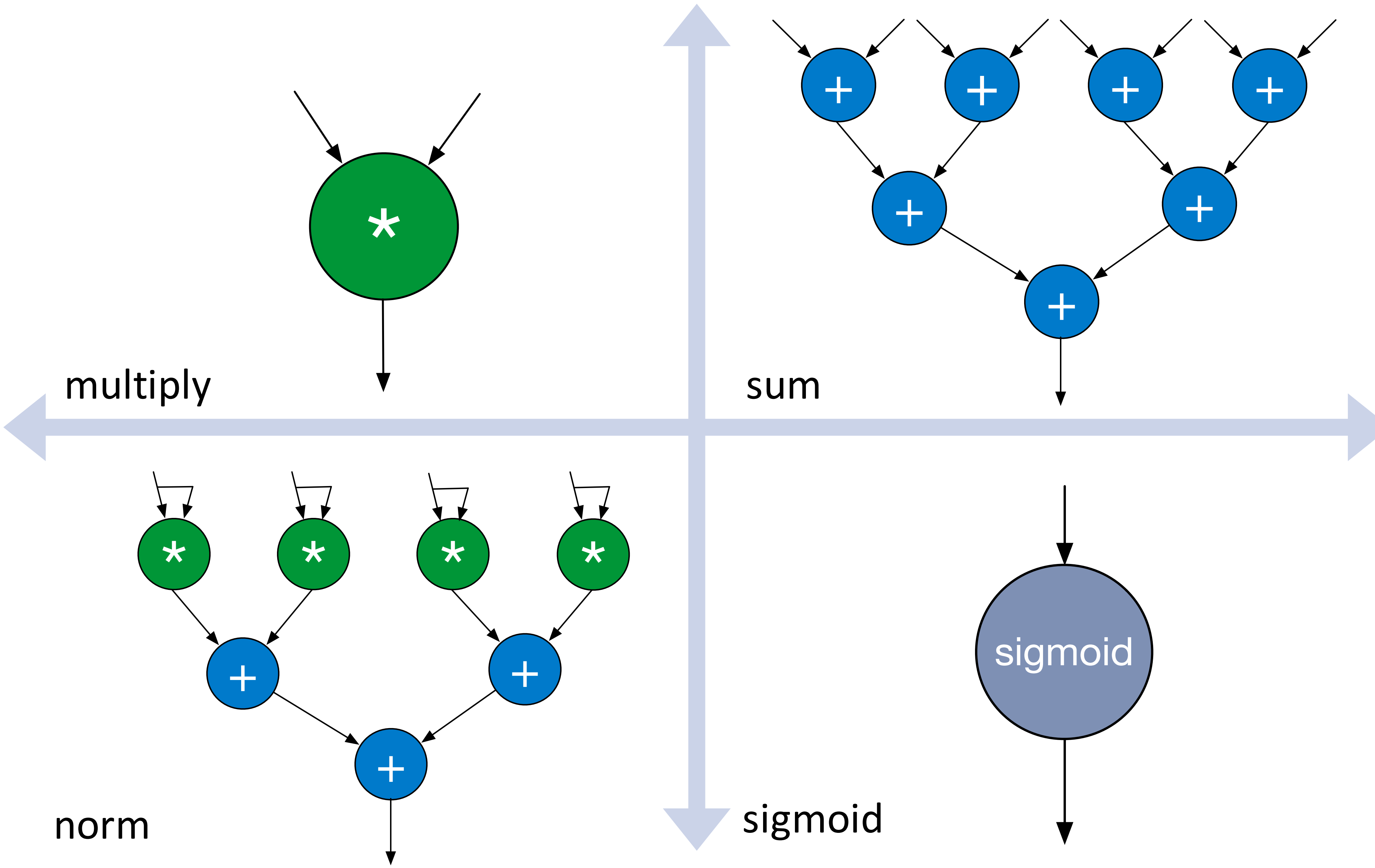


1. Appending the stochastic gradient descent solver
2. Creating the dataflow graph of the entire algorithm
3. Create a schedule for this dataflow graph

1. Appending Stochastic Gradient Descent

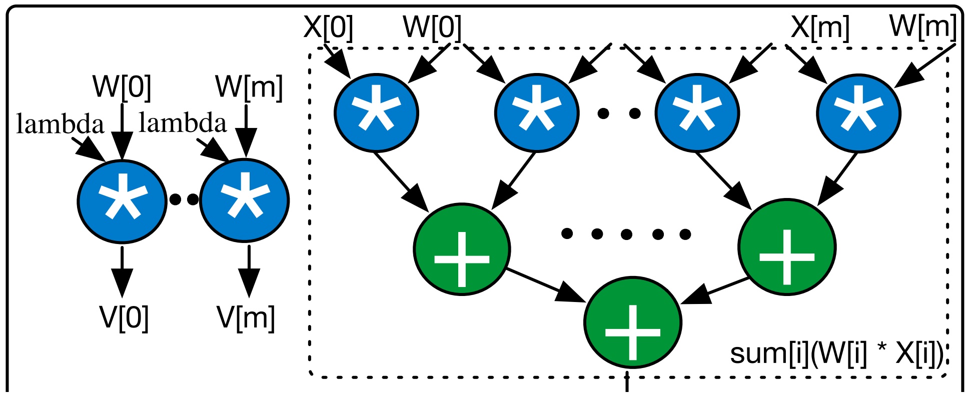
```
m = 53
lambda = 0.1
u = 0.1

model_input   X[m];
model_output  Y';
model         W[m];
gradient      G[m];

iterator i[0:m - 1];

S = sum [i](X[i] * W[i]);

Y = sigmoid(S);
E = Y - Y';
G[i] = X[i] * E;
V[i] = lambda * W[i];
G[i] = G[i] + V[i];

G[i] = u * G[i];
W[i] = W[i]-G[i];
```

2. Dataflow graph generation



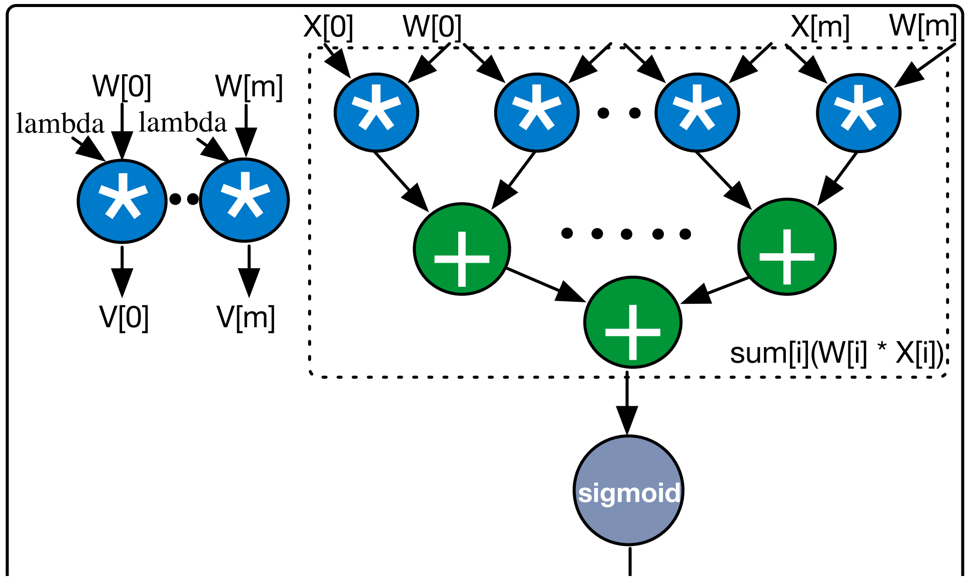
2. Dataflow graph generation

```
m = 53
lambda = 0.1
u = 0.1

model_input   X[m];
model_output  Y';
model         W[m];
gradient      G[m];

iterator i[0:m - 1];

S = sum [i](X[i] * W[i]);

Y = sigmoid(S);
E = Y - Y';
G[i] = X[i] * E;
V[i] = lambda * W[i];
G[i] = G[i] + V[i];

G[i] = u * G[i];
W[i] = W[i]-G[i];
```



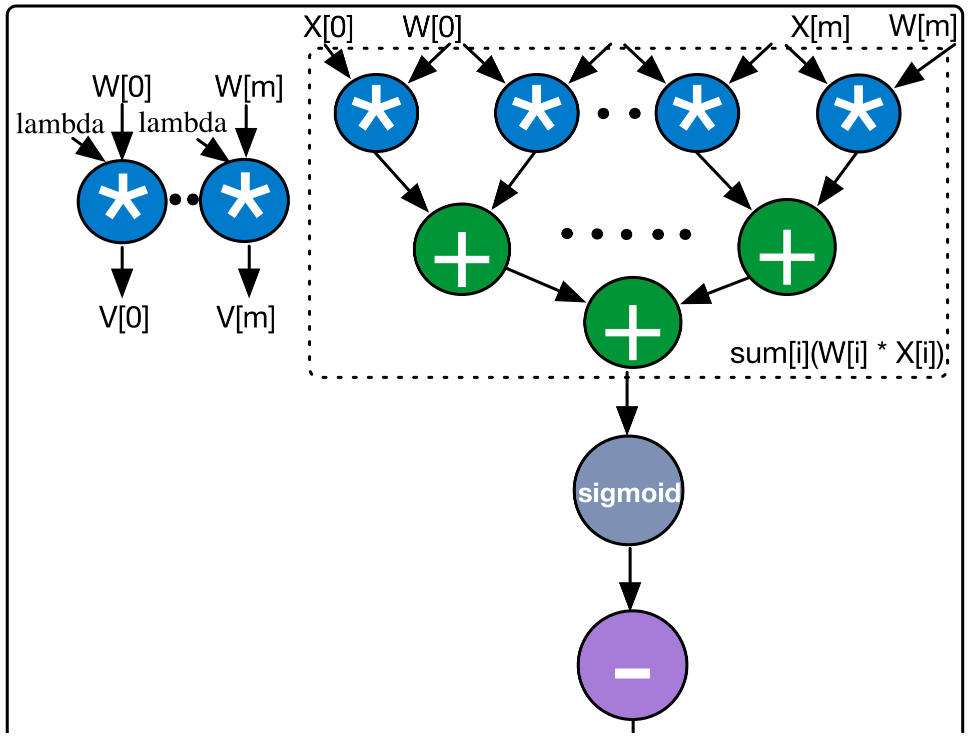
2. Dataflow graph generation

```
m = 53
lambda = 0.1
u = 0.1

model_input   X[m];
model_output  Y';
model         W[m];
gradient      G[m];

iterator i[0:m - 1];

S = sum [i](X[i] * W[i]);

Y = sigmoid(S);
E = Y - Y';
G[i] = X[i] * E;
V[i] = lambda * W[i];
G[i] = G[i] + V[i];

G[i] = u * G[i];
W[i] = W[i]-G[i];
```



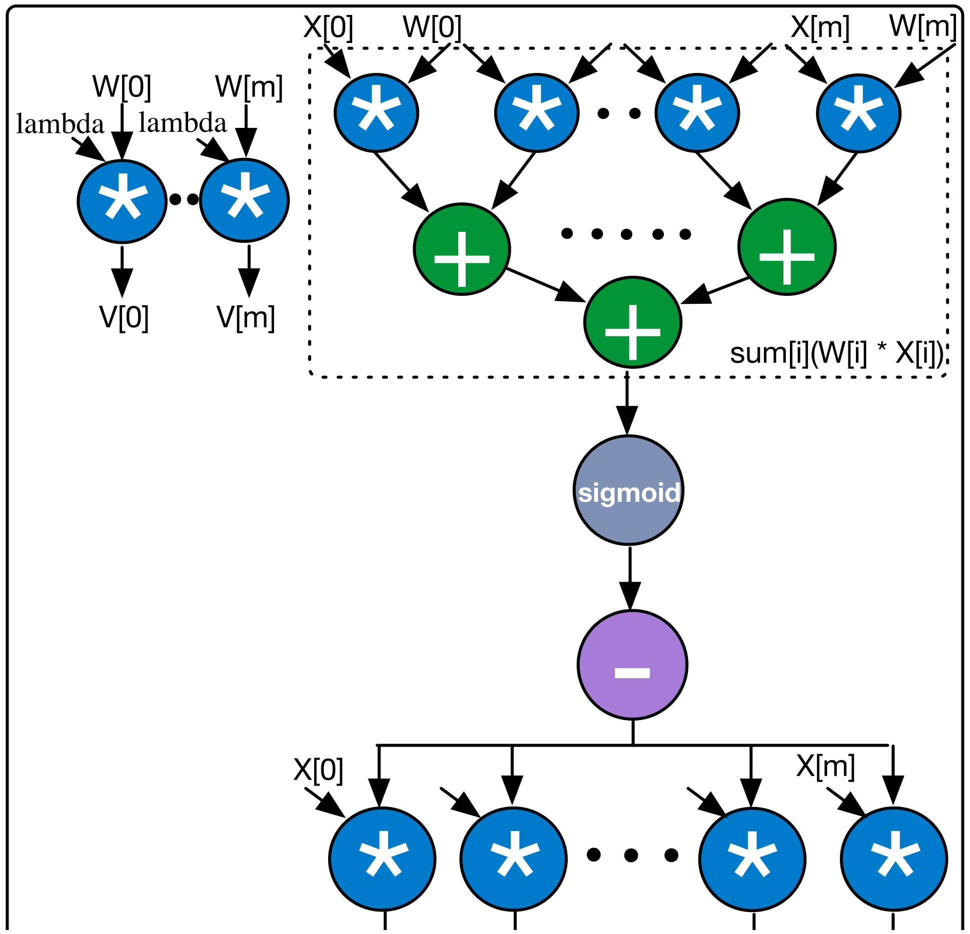
2. Dataflow graph generation

```
m = 53
lambda = 0.1
u = 0.1

model_input    X[m];
model_output   Y';
model          W[m];
gradient       G[m];

iterator  i[0:m - 1];

S = sum [i](X[i] * W[i]);

Y = sigmoid(S);
E = Y - Y';
G[i] = X[i] * E;
V[i] = lambda * W[i];
G[i] = G[i] + V[i];

G[i] = u * G[i];
W[i] = W[i]-G[i];
```



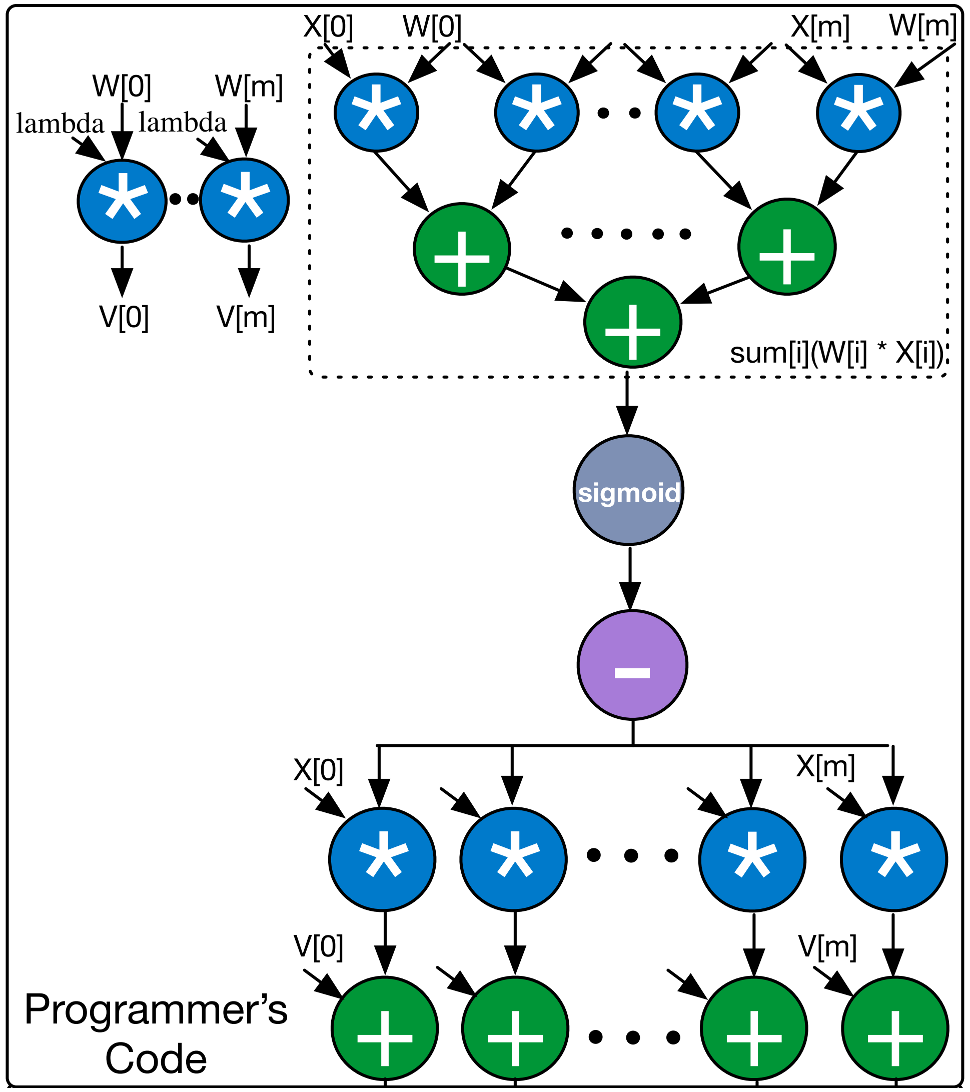
2. Dataflow graph generation

```
m = 53
lambda = 0.1
u = 0.1

model_input   X[m];
model_output  Y';
model         W[m];
gradient      G[m];

iterator i[0:m - 1];

S = sum [i](X[i] * W[i]);

Y = sigmoid(S);
E = Y - Y';
G[i] = X[i] * E;
V[i] = lambda * W[i];
G[i] = G[i] + V[i];

G[i] = u * G[i];
W[i] = W[i]-G[i];
```



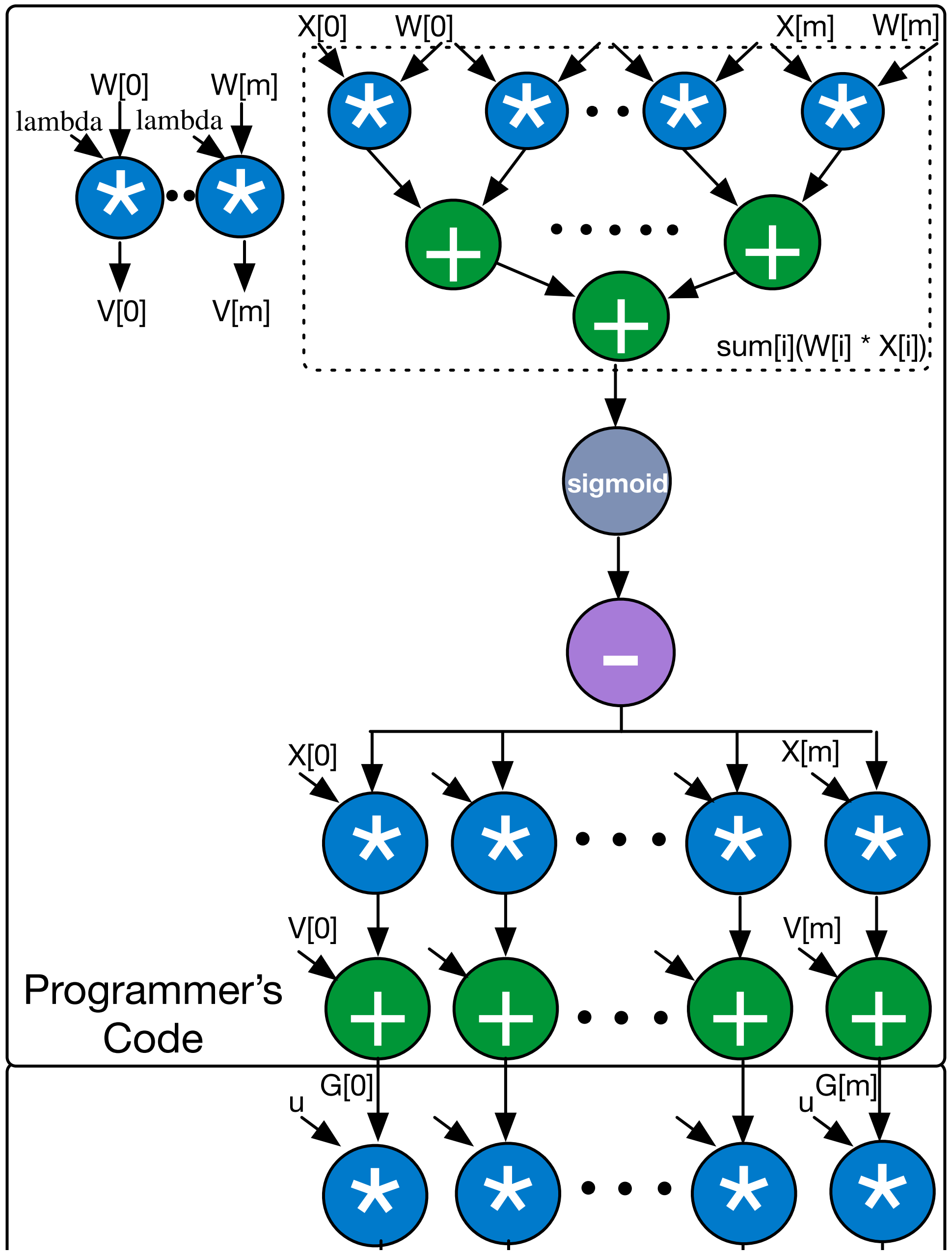
2. Dataflow graph generation

```
m = 53
lambda = 0.1
u = 0.1

model_input   X[m];
model_output  Y';
model         W[m];
gradient      G[m];

iterator i[0:m - 1];

S = sum [i](X[i] * W[i]);

Y = sigmoid(S);
E = Y - Y';
G[i] = X[i] * E;
V[i] = lambda * W[i];
G[i] = G[i] + V[i];

G[i] = u * G[i];
W[i] = W[i]-G[i];
```



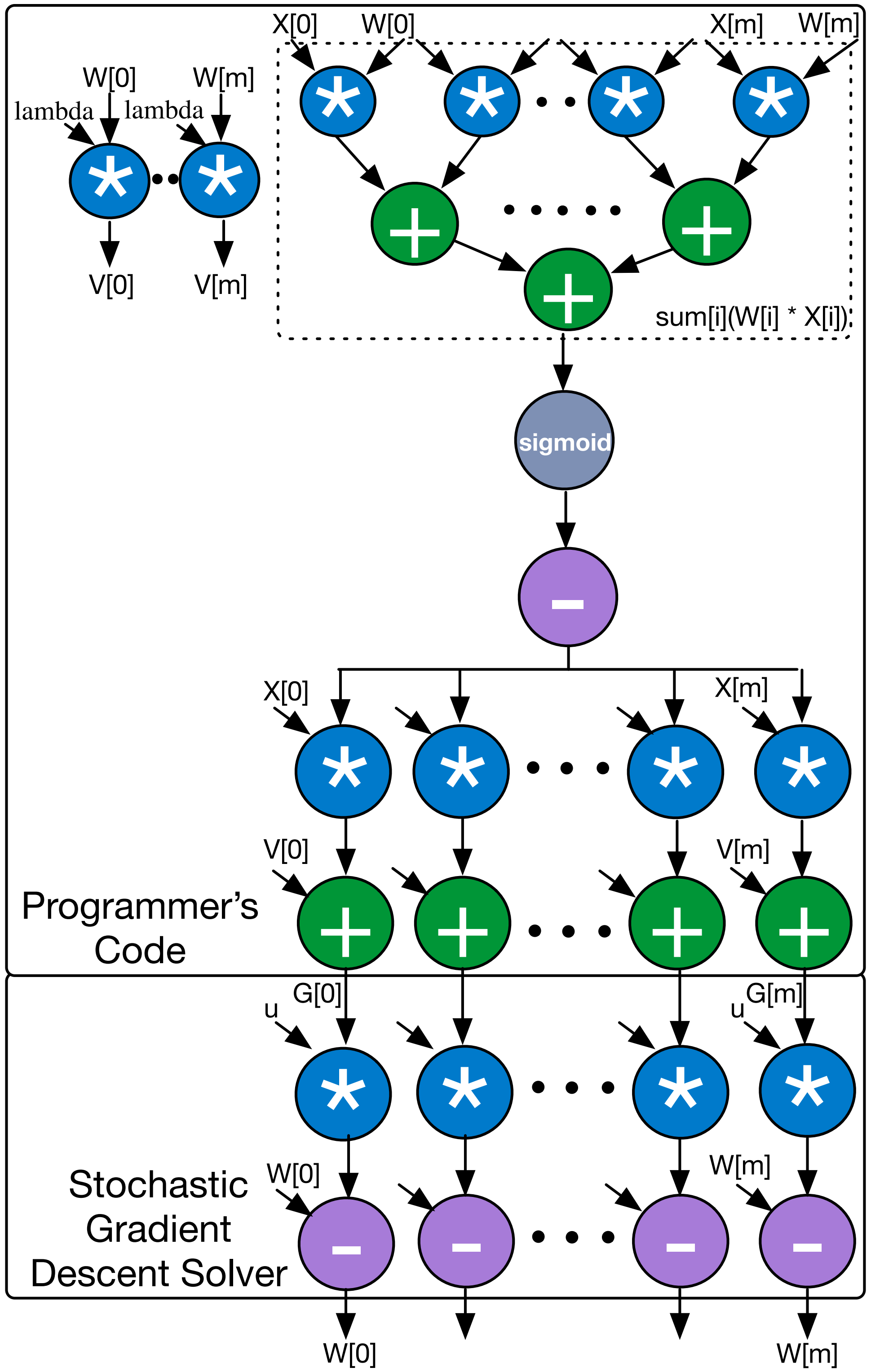
2. Dataflow graph generation

```
m = 53
lambda = 0.1
u = 0.1

model_input   X[m];
model_output  Y';
model         W[m];
gradient      G[m];

iterator i[0:m - 1];

S = sum [i](X[i] * W[i]);

Y = sigmoid(S);
E = Y - Y';
G[i] = X[i] * E;
V[i] = lambda * W[i];
G[i] = G[i] + V[i];

G[i] = u * G[i];
W[i] = W[i]-G[i];
```

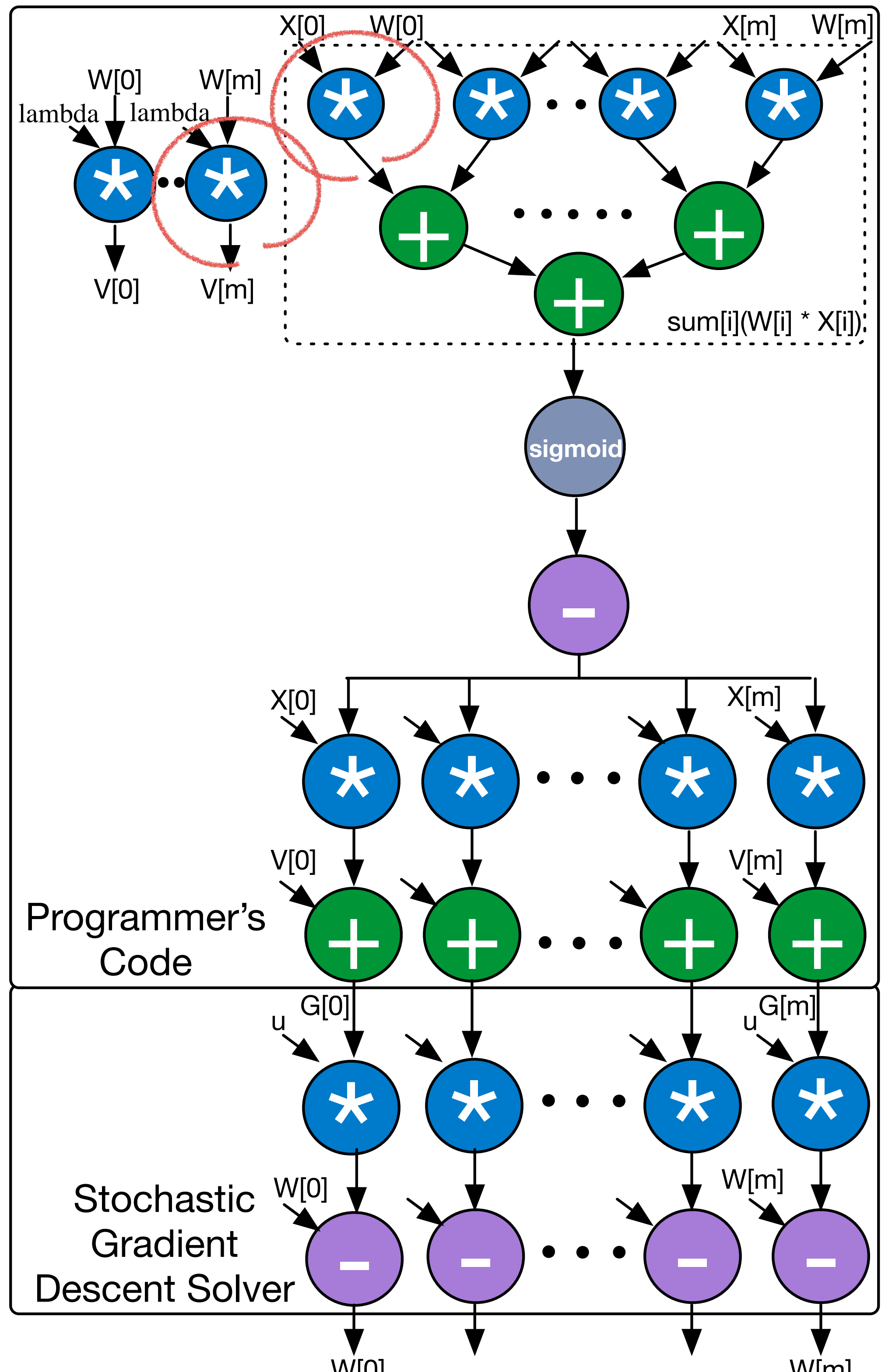


2. Dataflow graph generation

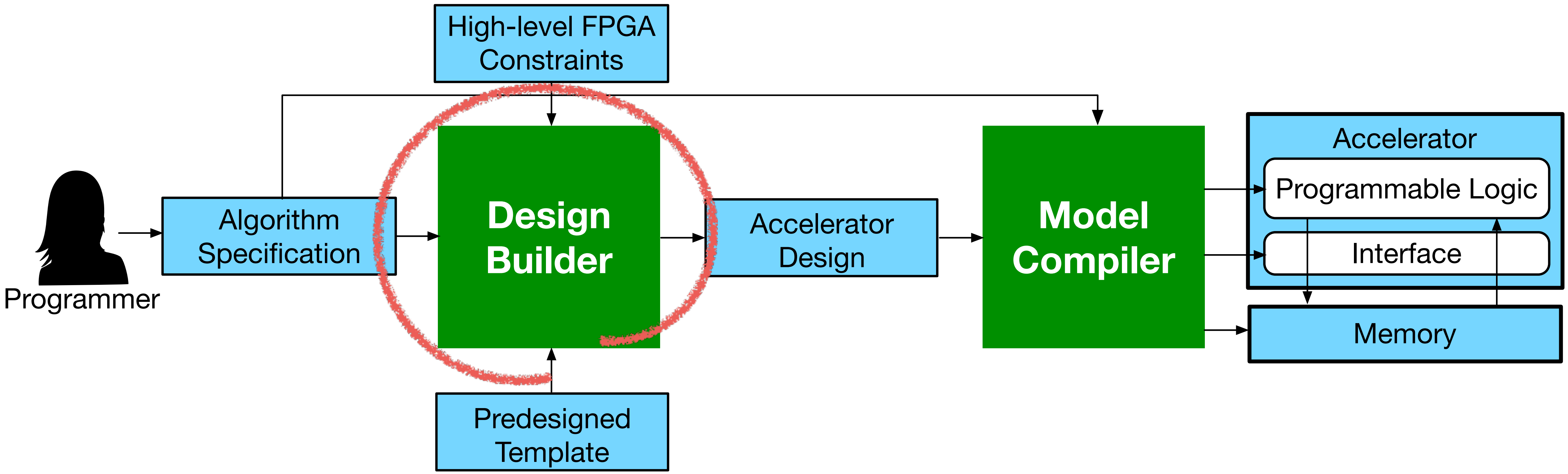
```
m = 53
lambda = 0.1
u = 0.1

model_input   X[m];
model_output  Y';
model         W[m];
gradient      G[m];

iterator i[0:m - 1];

S = sum [i](X[i] * W[i]);

Y = sigmoid(S);
E = Y - Y';
G[i] = X[i] * E;
V[i] = lambda * W[i];
G[i] = G[i] + V[i];

G[i] = u * G[i];
W[i] = W[i]-G[i];
```



3. Operation scheduling

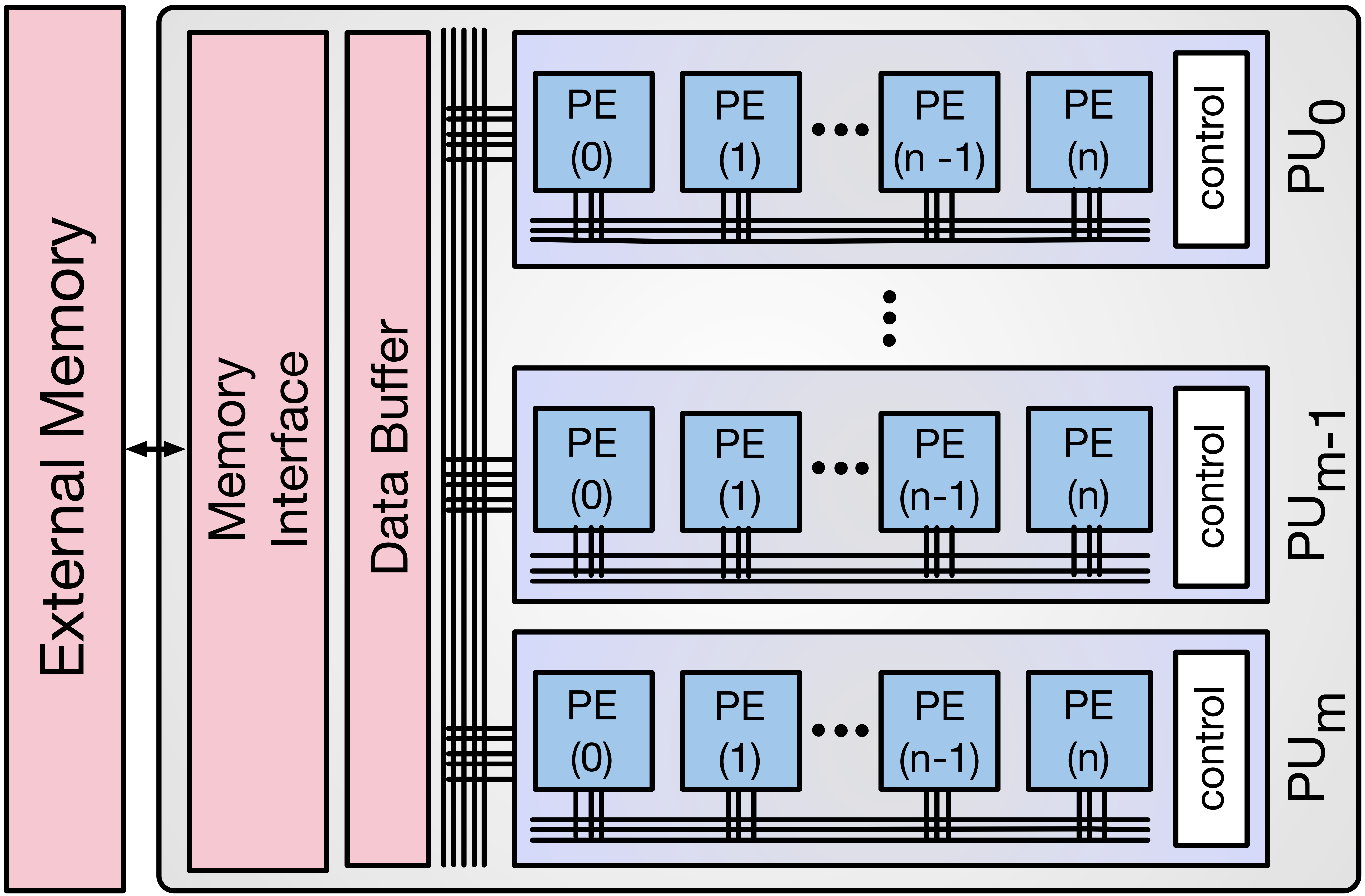


Design Builder

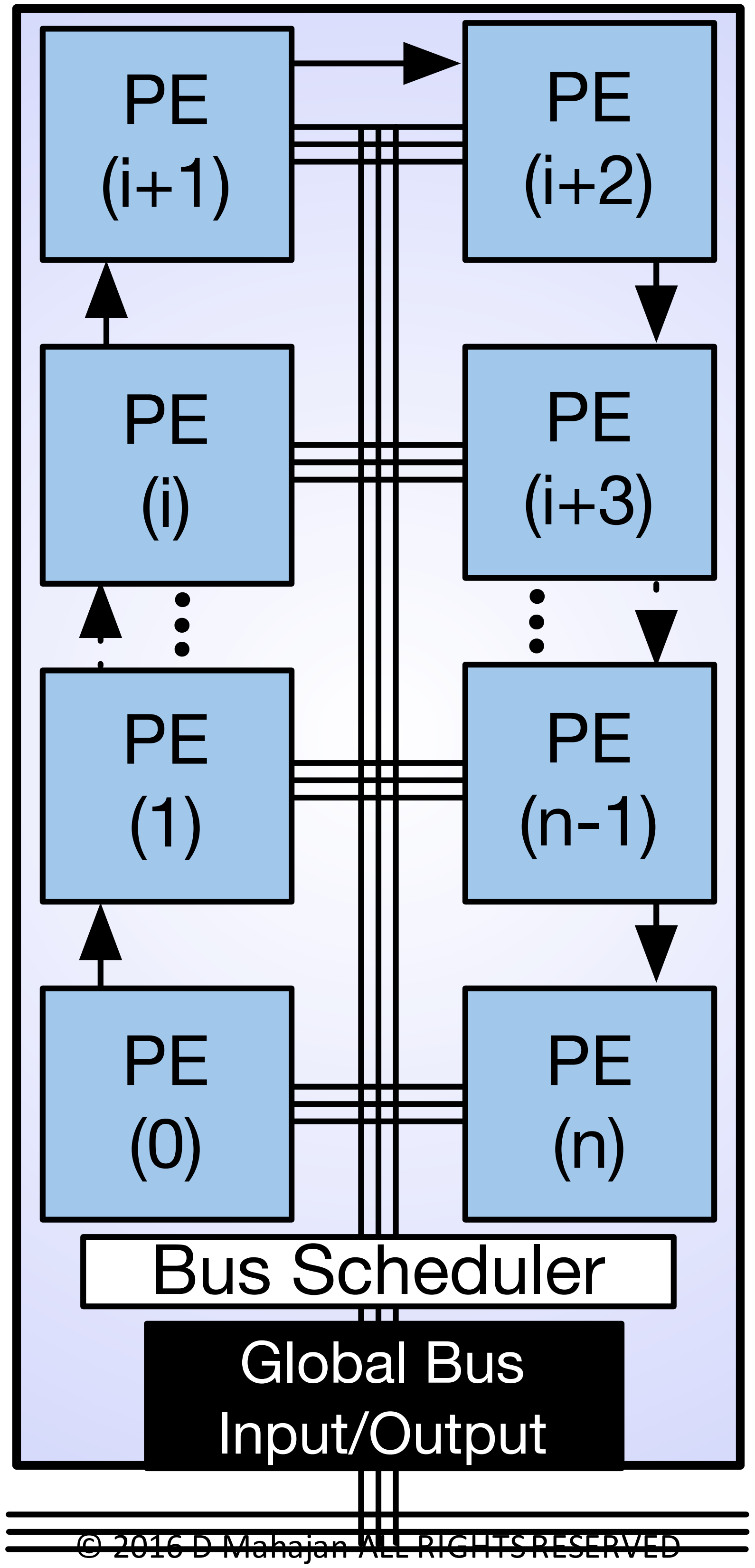


Together with the model compiler generates the final accelerator design

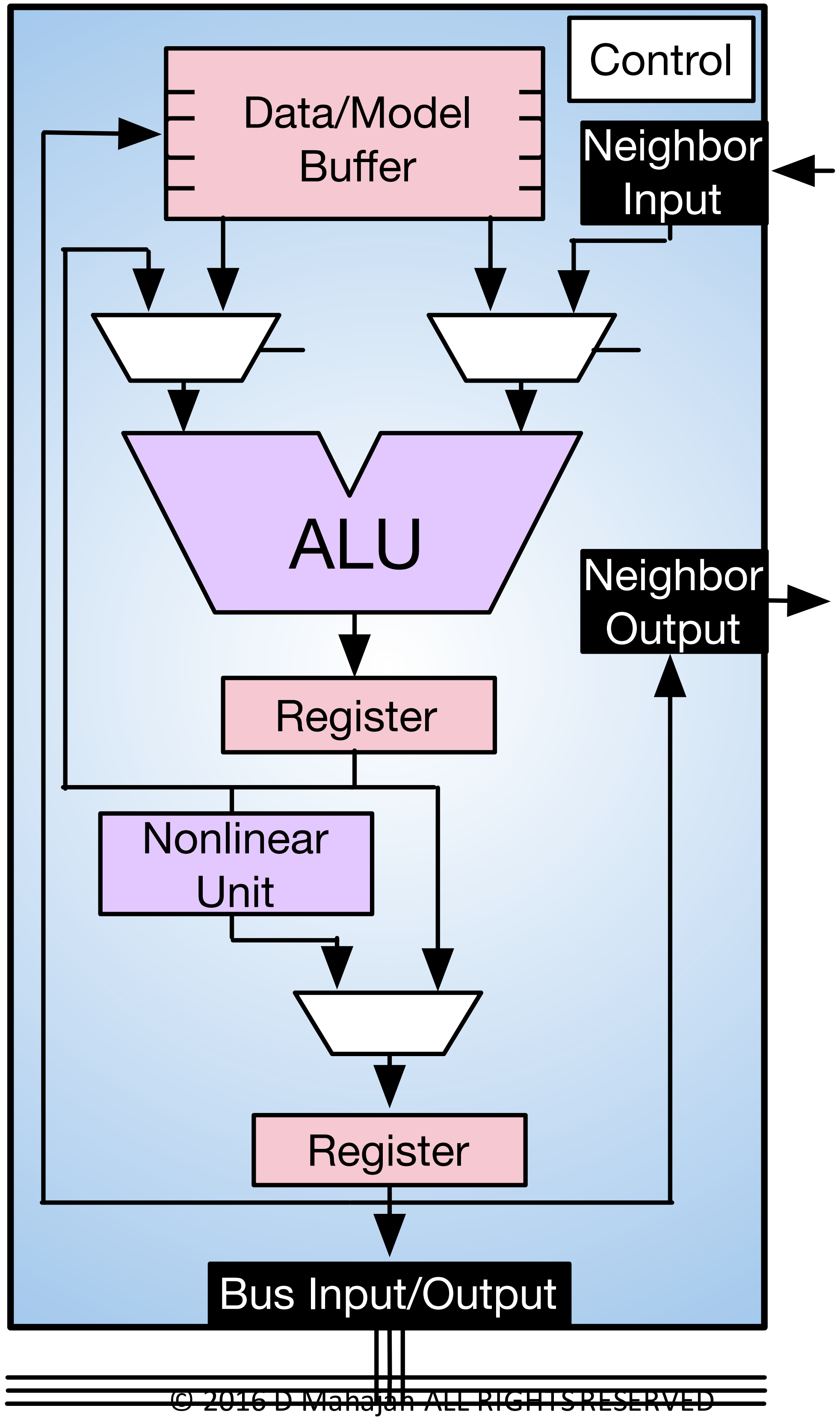
Hierarchical template design of accelerator



Template design of Processing Unit



Template design of Processing Engine



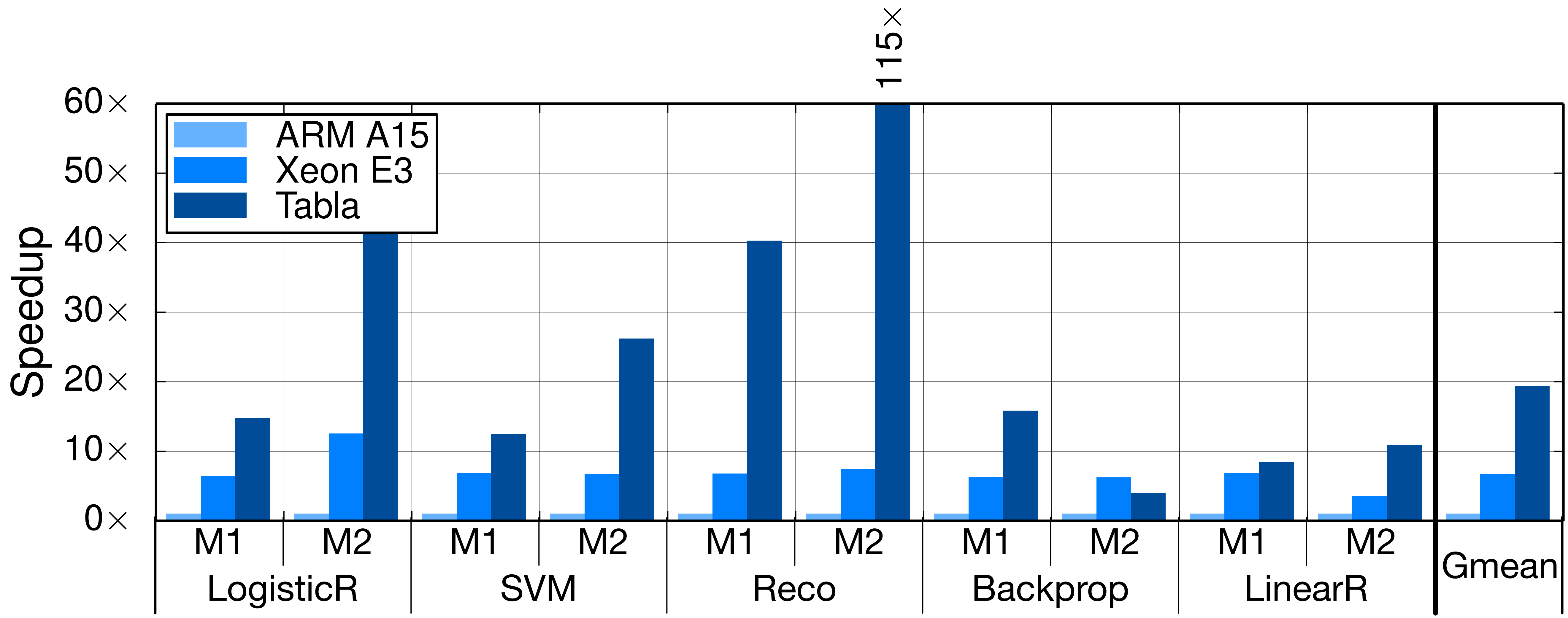
Learning tasks and their topologies

		Model Topology	# Lines
Regression Model	**LogisticR** Logistic Regression	M1: 54 M2: 200	20
Classification	**SVM** Support Vector Machines	M1: 54 M2: 200	23
Collaborative Filtering	**Reco** Recommender Systems	M1: 1700 x 1000 M2: 6000 x 4000	31
Multilayer Perceptron	**BackProp** Back Propagation	M1: 10 -> 9 -> 1 M2: 256 -> 128 -> 256	48
Regression Analysis	**LinearR** Linear Regression	M1: 55 M2: 784	20

Evaluation platforms

FPGA	Xilinx Zynq 7000 ZC702 TDP: 2W $129		
CPU	ARM Cortex 15 TDP: 5W $191	Intel Xeon E3-1276 V3 TDP: 84W $339	
GPU	Tegra K1 GPU TDP: 10 W $191	GeForce GTX 650 Ti TDP: 110 $150	Tesla K40 TDP: 210 W $5499

Speedup in comparison to CPUs



- TABLA generated accelerators provide 19× speedup over ARM and 2.9× speedup over Xeon
- Static scheduling alleviates the traditional Von-Neumann overheads

Speedup in comparison to GPUs

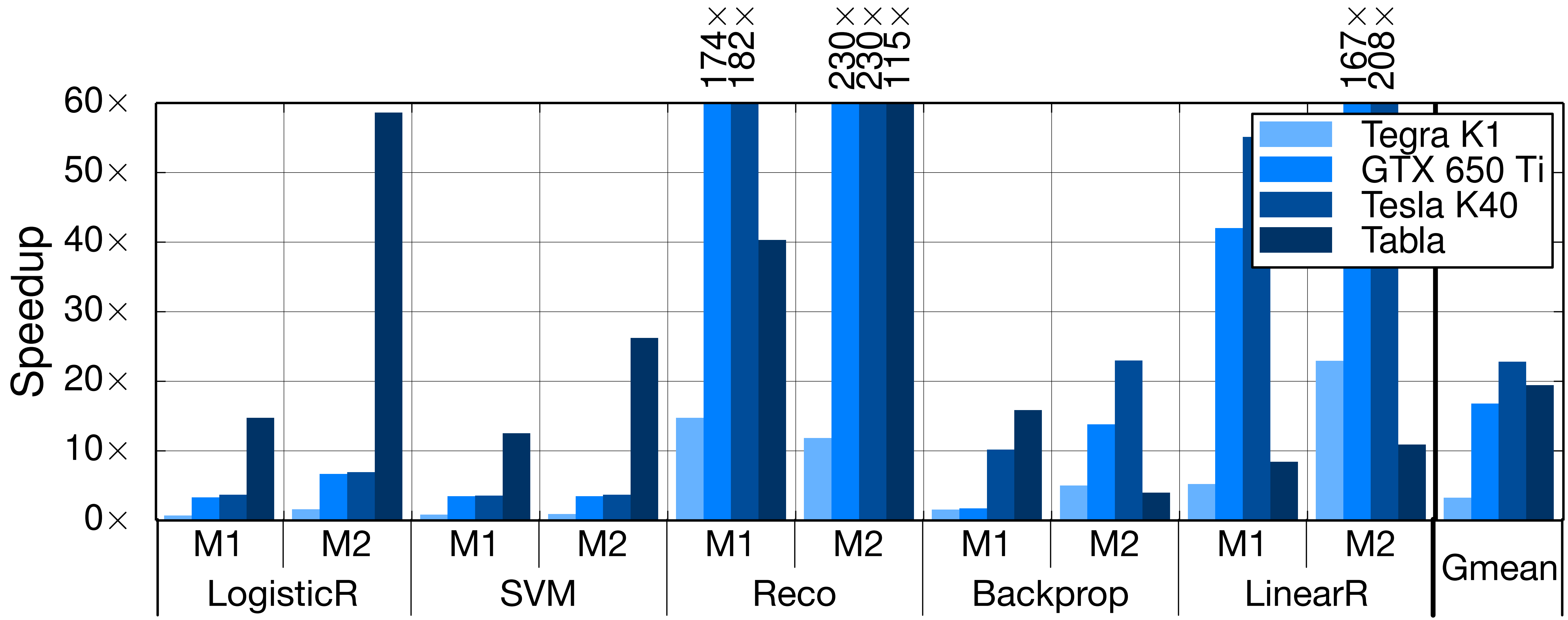
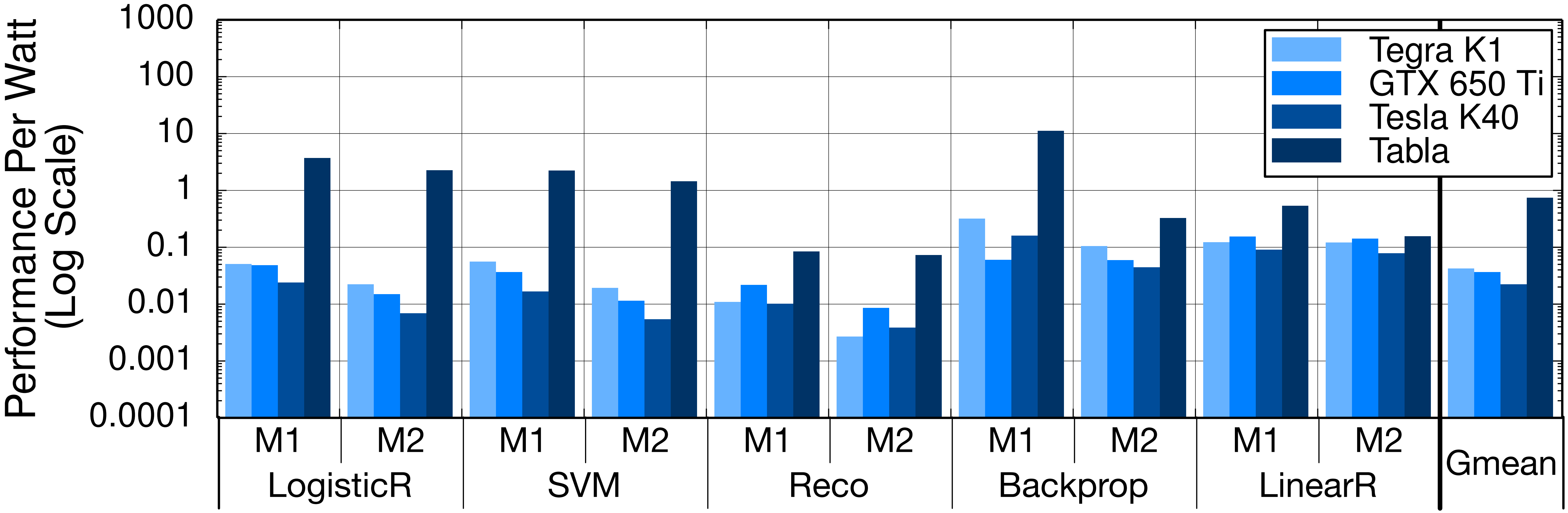


TABLA generated accelerators provide 5.9× speedup over Tegra K1, 1.16x over GTX 650 Ti and 1.18x slowdown over Tesla K40

Performance-per-Watt in comparison to GPUs



- TABLA generated accelerators provide 17× performance-per-Watt over Tegra K1, 20× over GTX 650 Ti and 33× over Tesla K40
- Specialized template designs extract fine-grained parallelism while consuming less power

Conclusions

- Exploit algorithmic commonalities to accelerate machine learning
- Create a template-based framework
- Abstract details of hardware from programmer
- Enable FPGAs to be accelerator of choice for machine learning
- http://act-lab.org/artifacts/tabla

Thank you