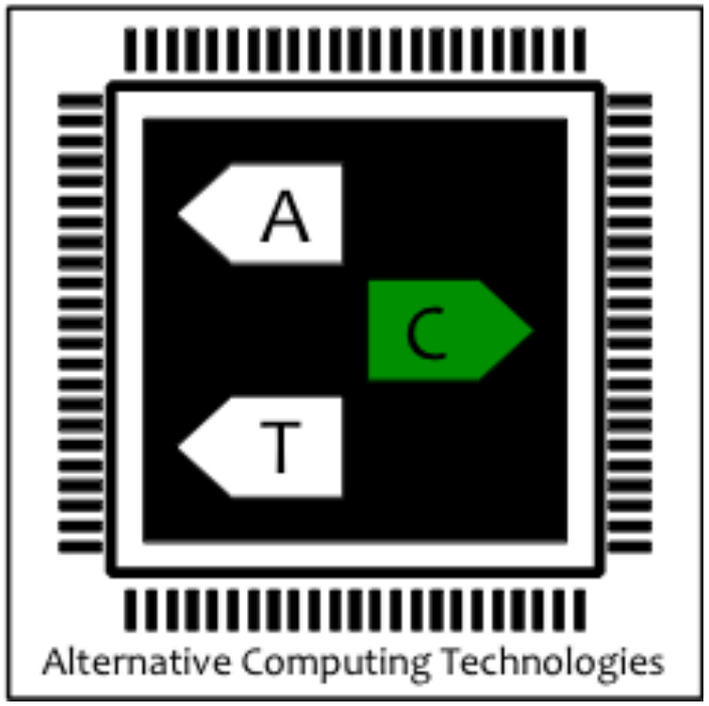
TABLA: A Unified Template-based Framework for Accelerating Statistical Machine Learning

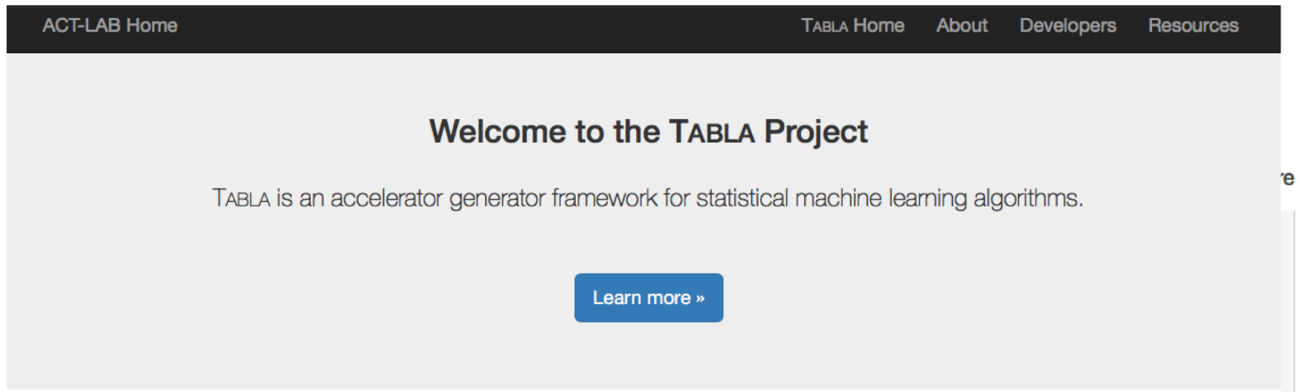


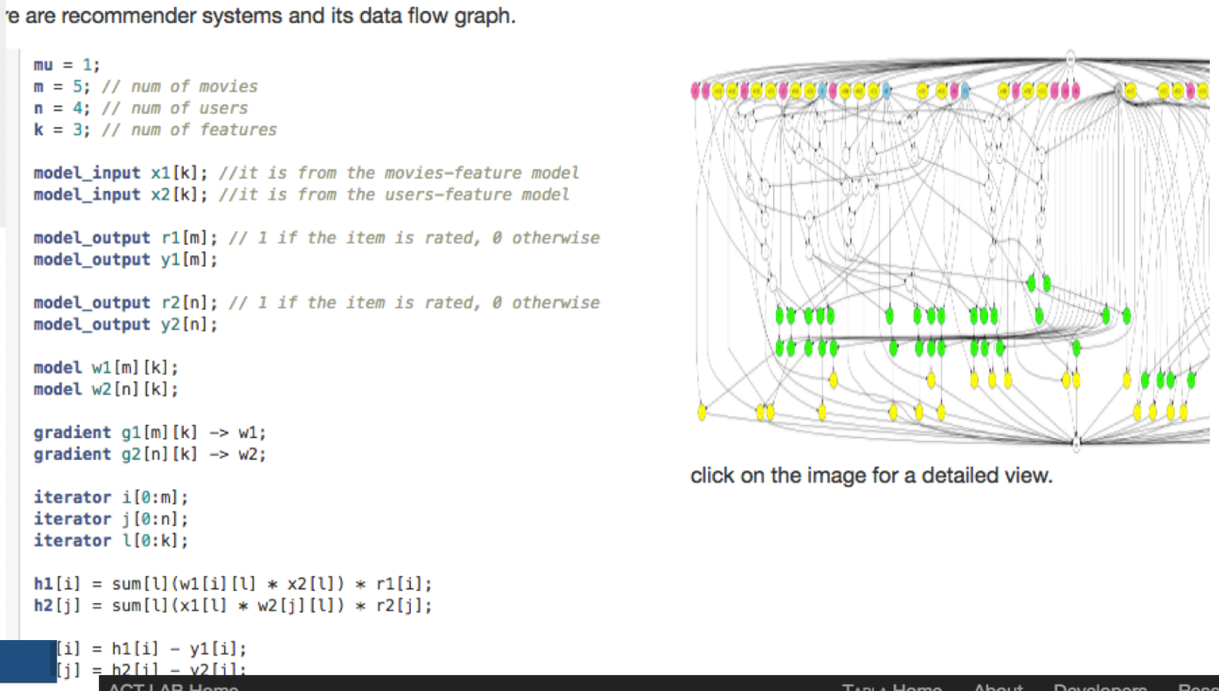
Hadi Esmaeilzadeh

Halicioglu Chair in Computer Architecture

Alternative Computing Technologies (ACT) Lab
University of California, San Diego

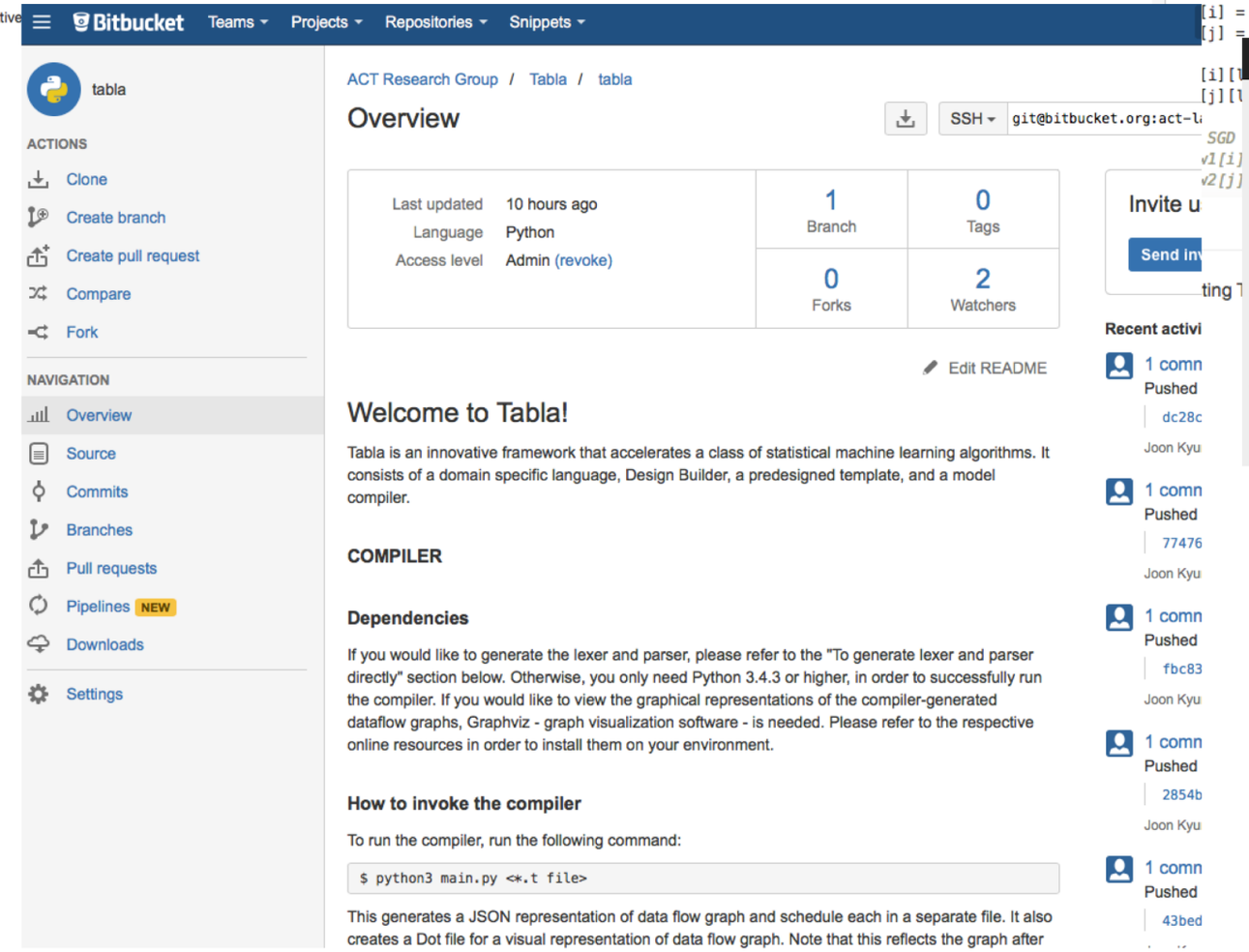
http://act-lab.org/artifacts/tabla/

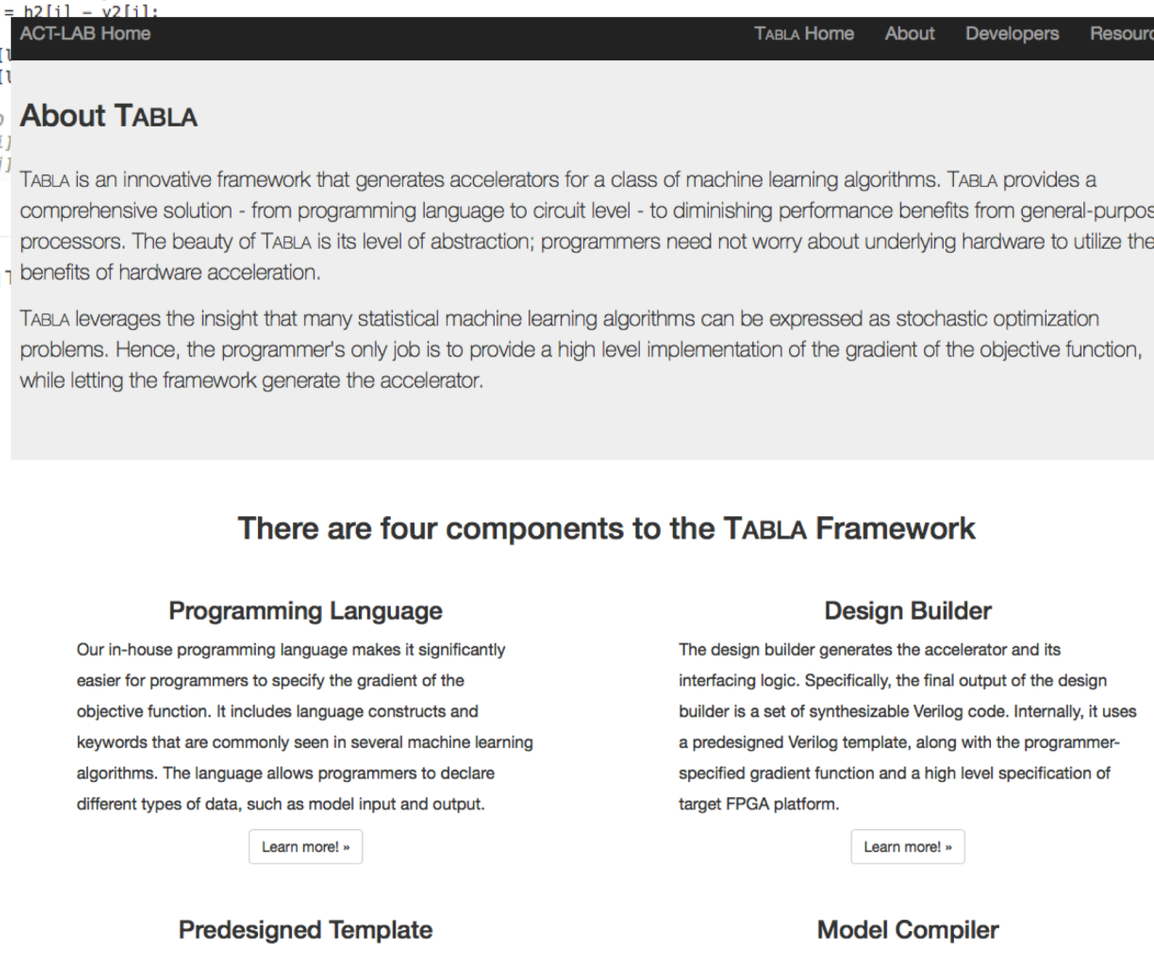




click on the image for a detailed view.

Alternative









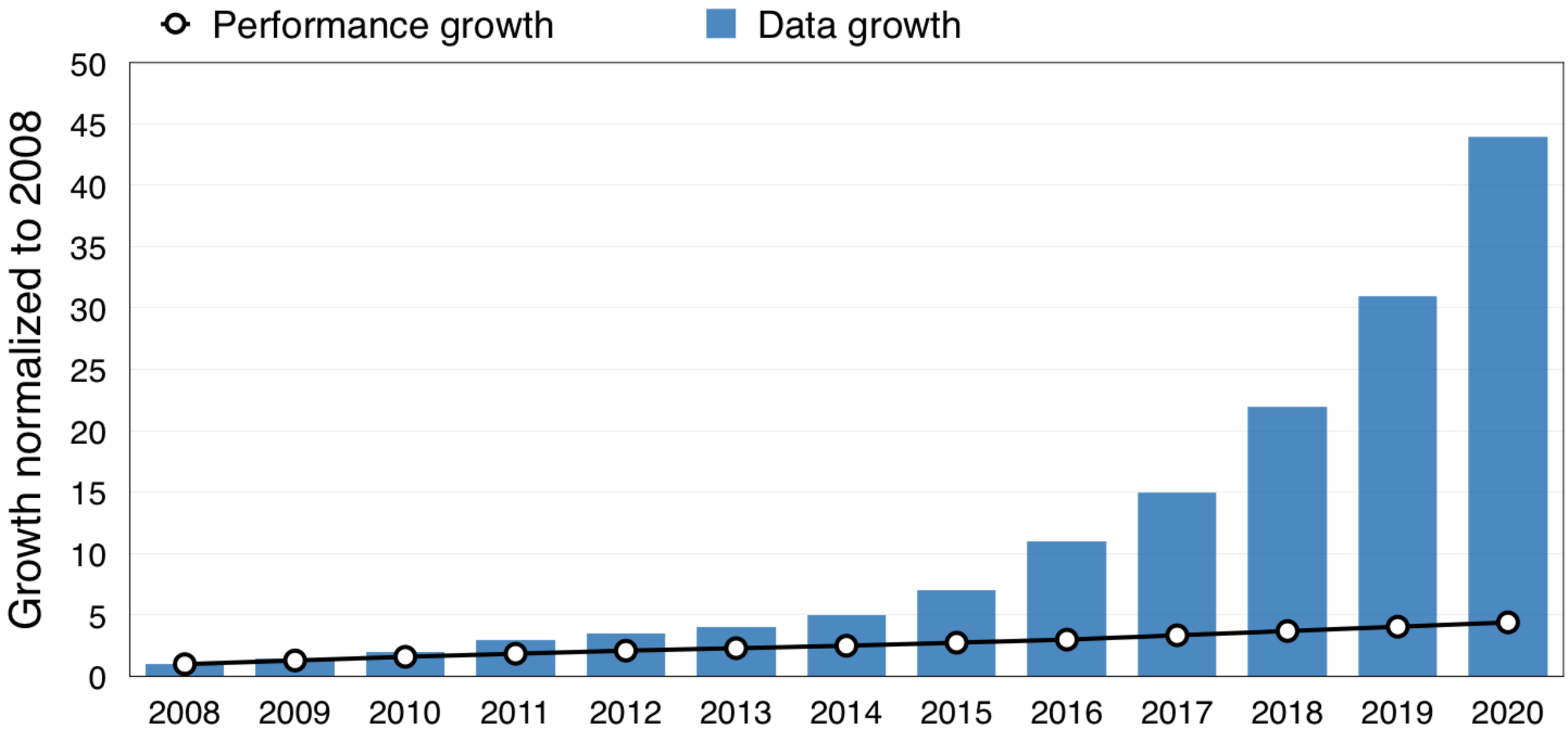

Alumni (MSc): Emmanuel Amaro (UC Berkley, CA), Bradley Thwaites (ARM, Austin, TX), Sindhuja Sethuraman (Apple, Orlando, FL), Anandhavel Nagendrakumar (IM Flash, Lehi, UT)

New landscape of computing: personalize and targeted experience for users

Connectivity leads to data explosion

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

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

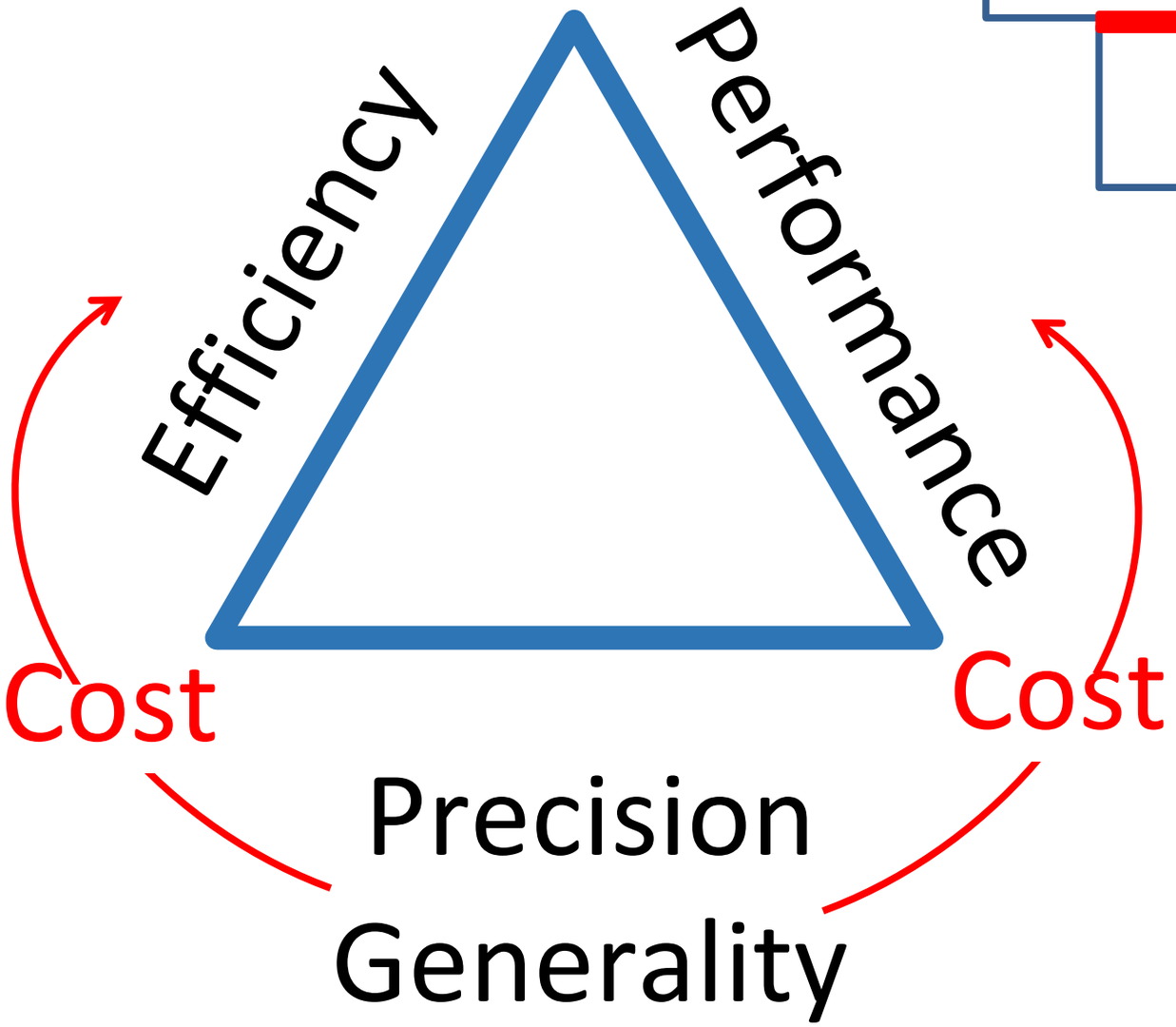
Energy efficiency is a primary concern

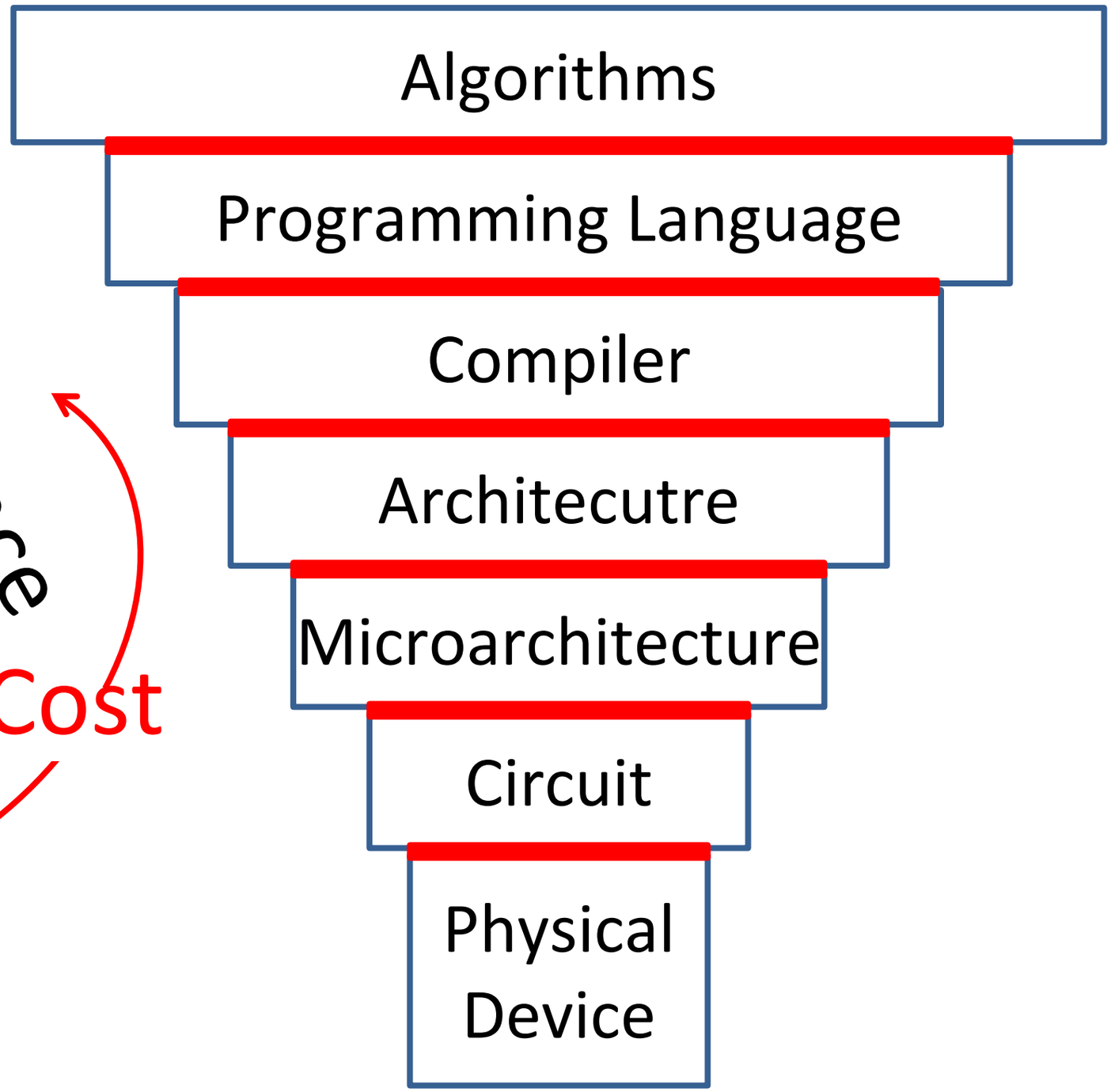
Data Center

Mobile

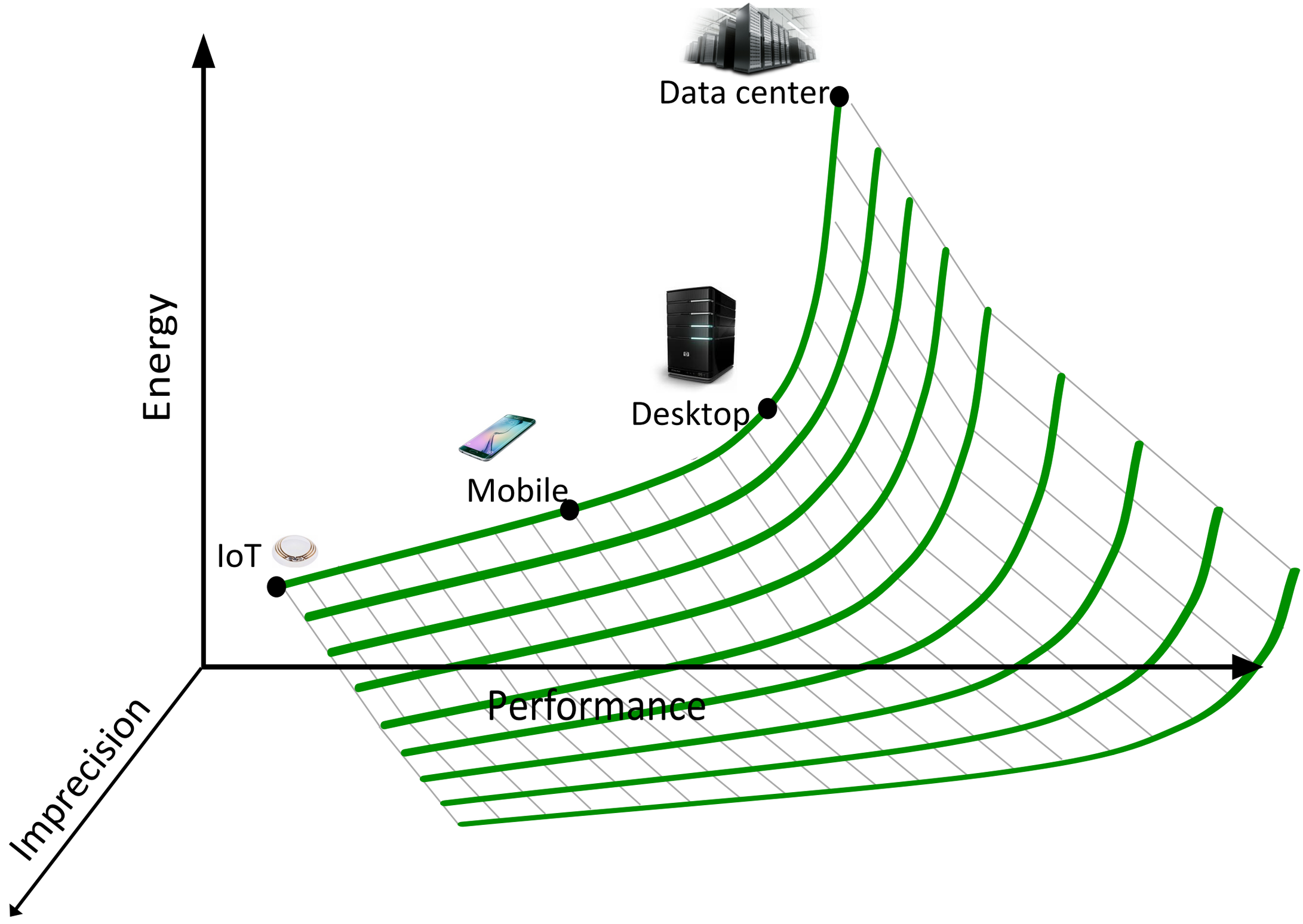
Internet of Things

Rethinking the abstractions

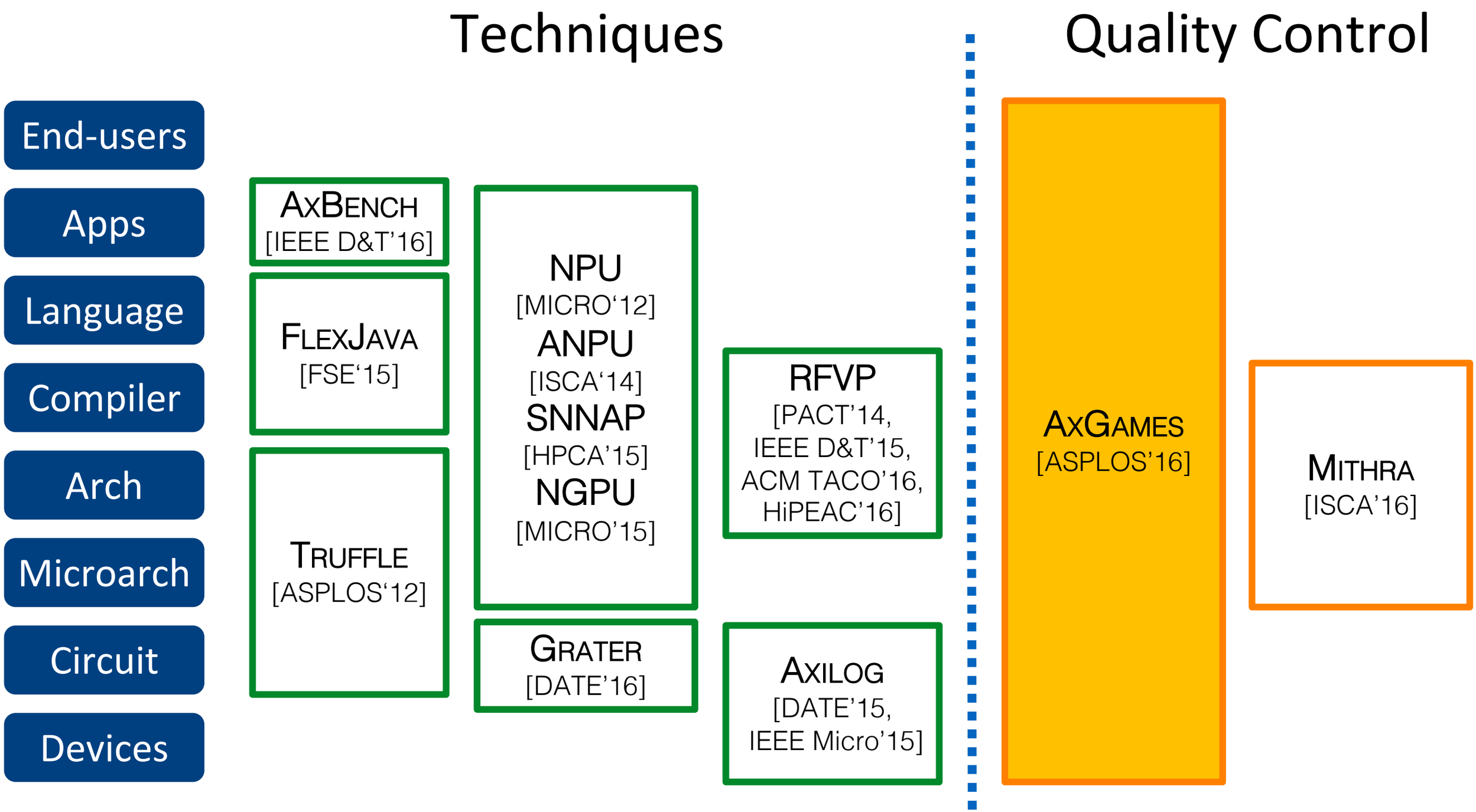




Approximate computing: embracing imprecision



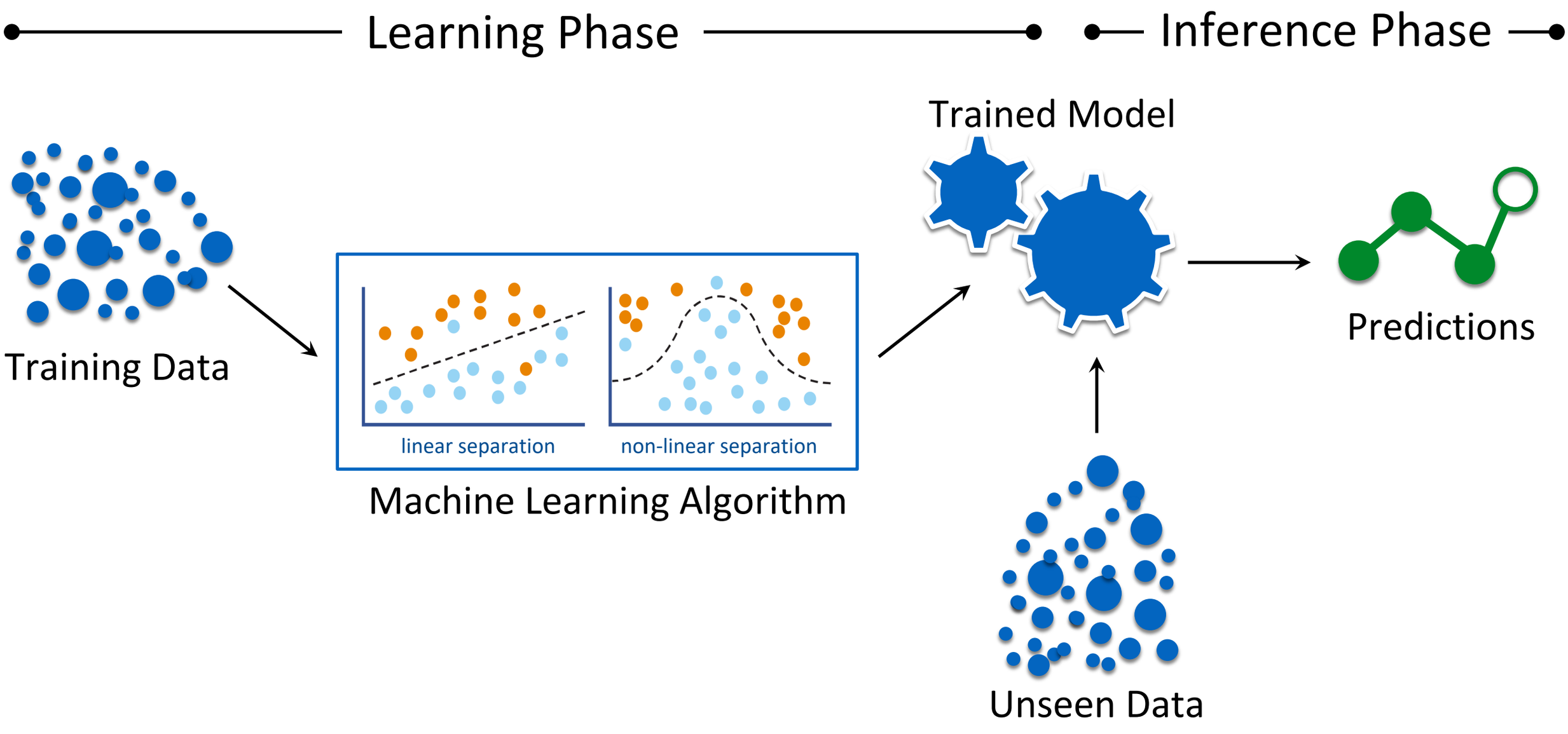
A cross-stack approach to enable approximation



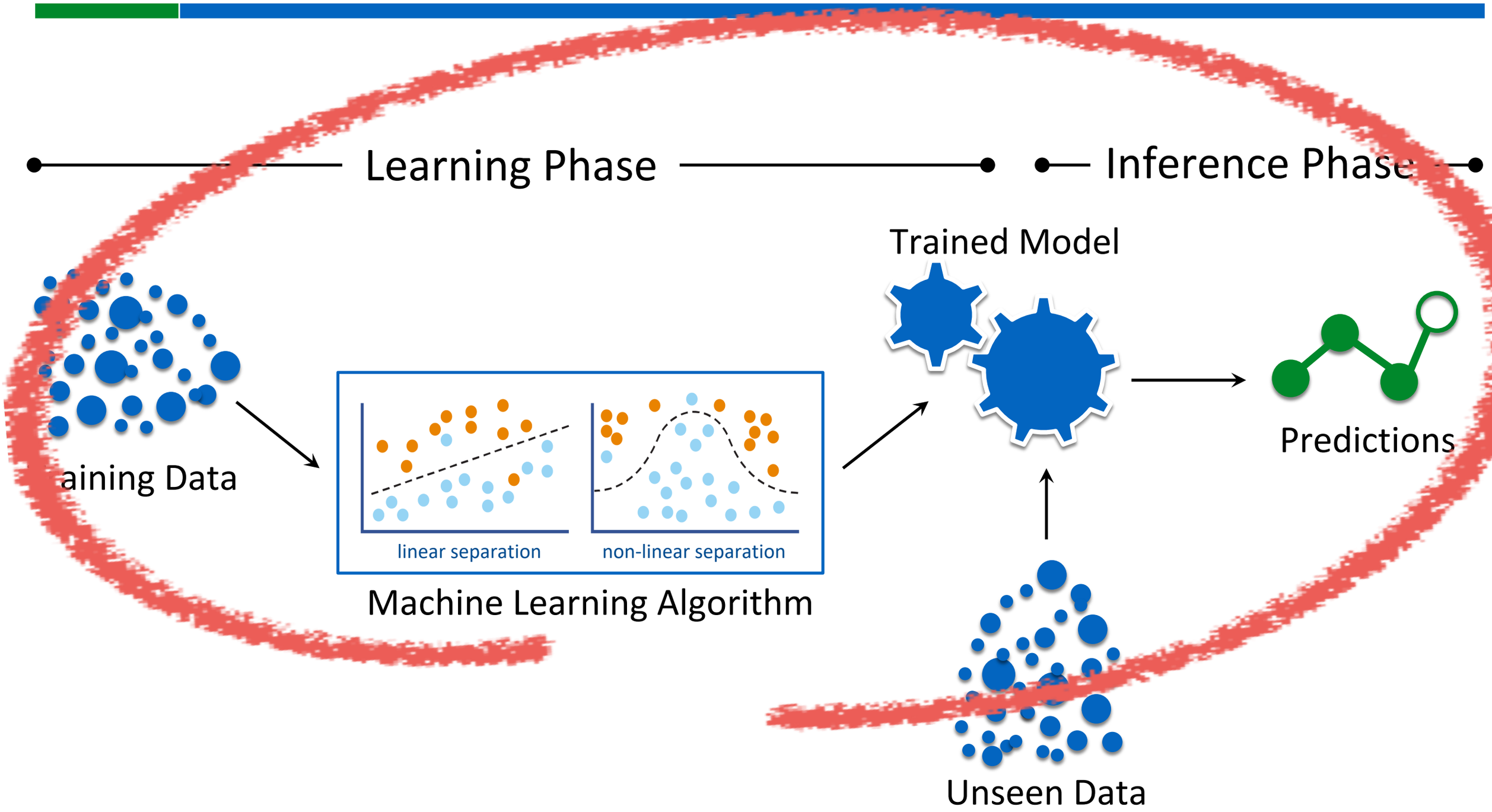
An Algorithmic Approach to Accelerate Learning

Unleashing the Beast

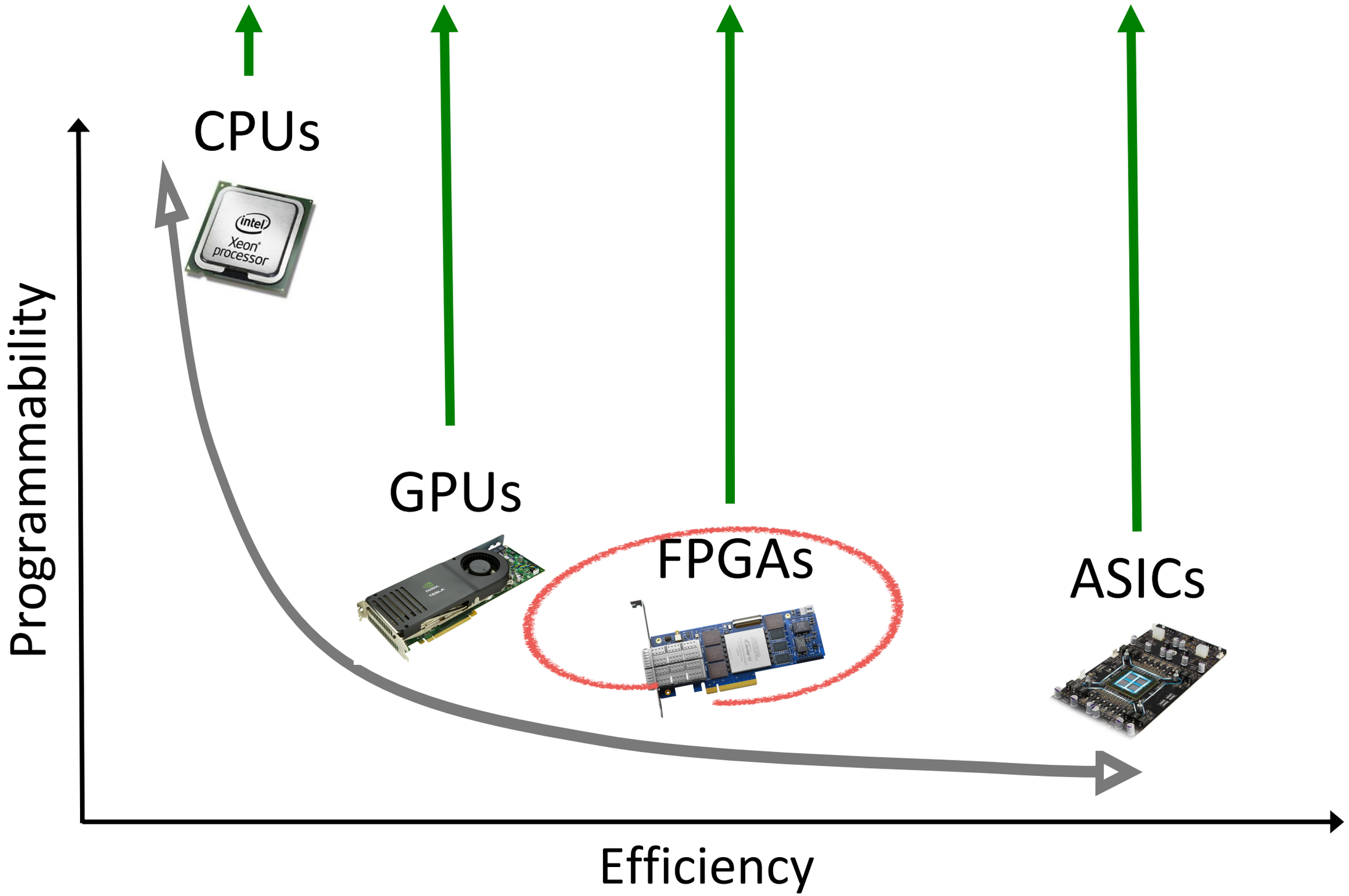
Machines learn to extract insights from data



Machines learn to extract insights from data

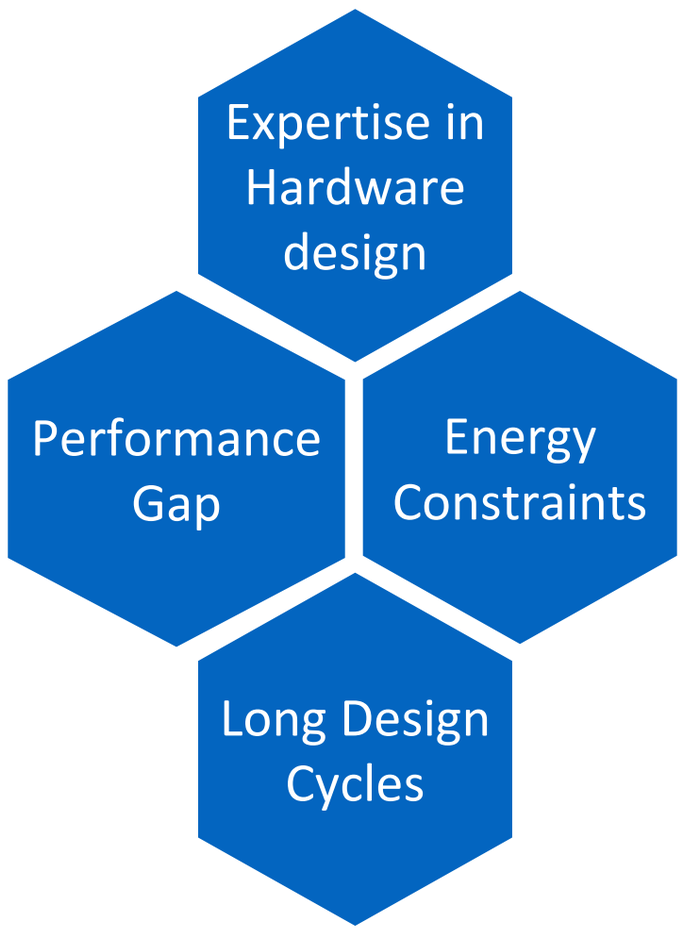


Programmability versus efficiency

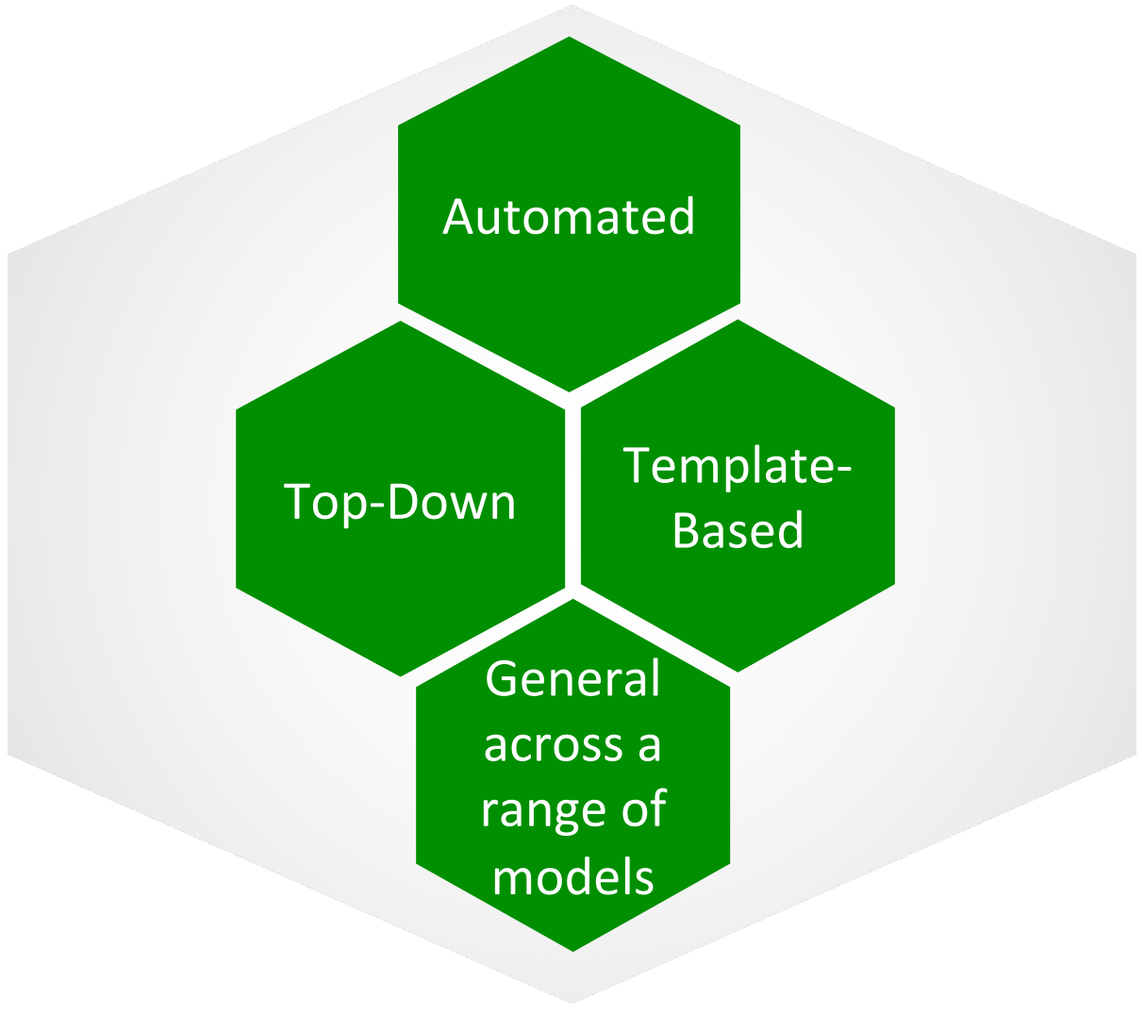


An algorithmic approach towards acceleration at scale

Challenges



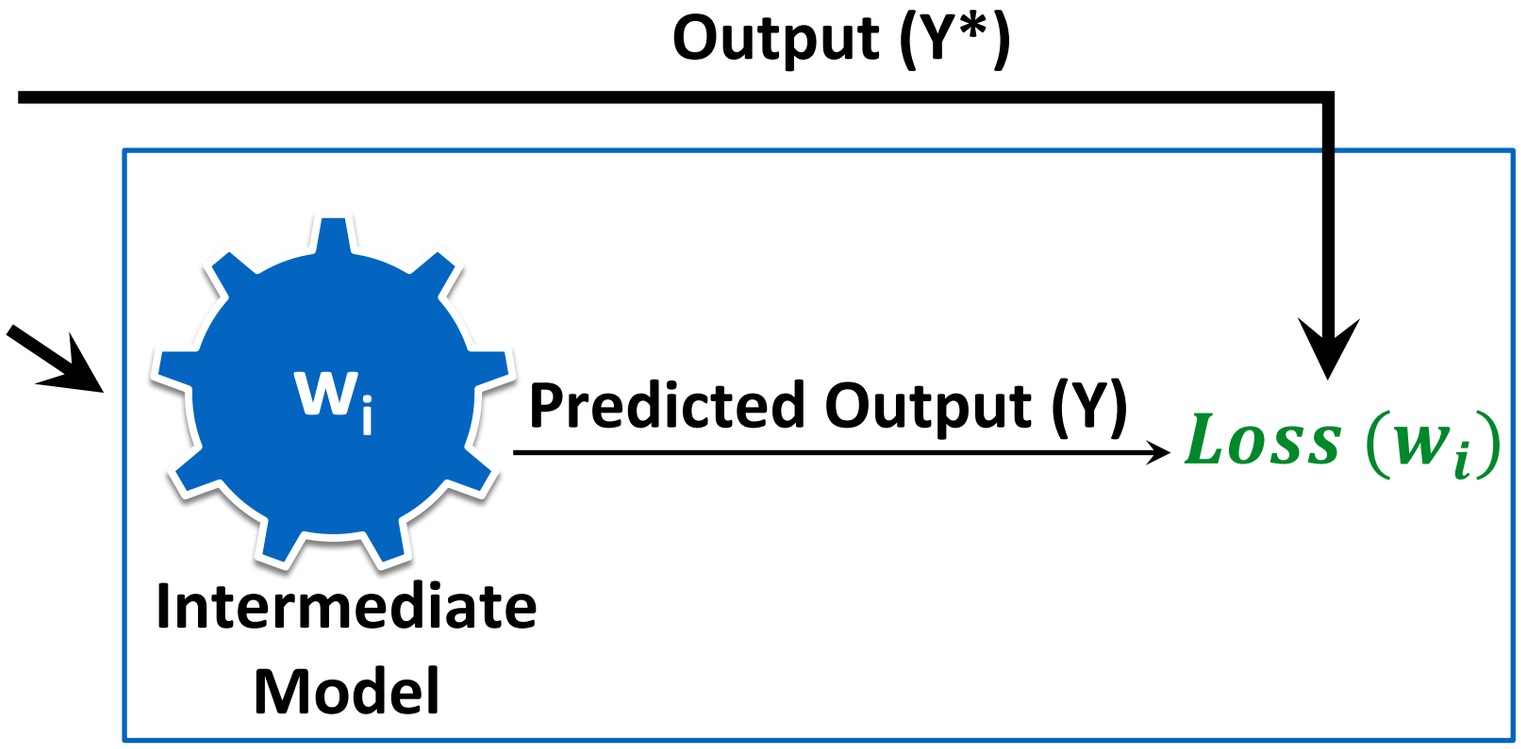
TABLA/MLWEAVER



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



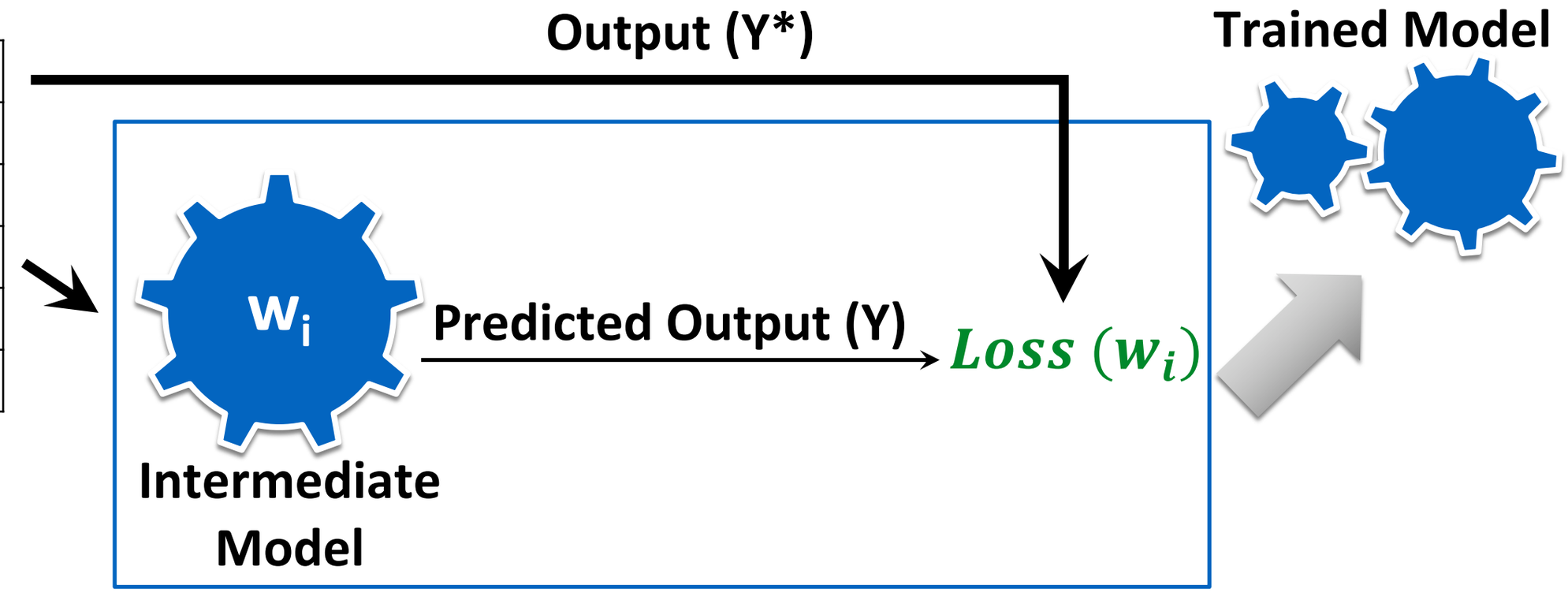
$$Loss(w_i) = \sum_i \|Y - Y^*\|$$

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

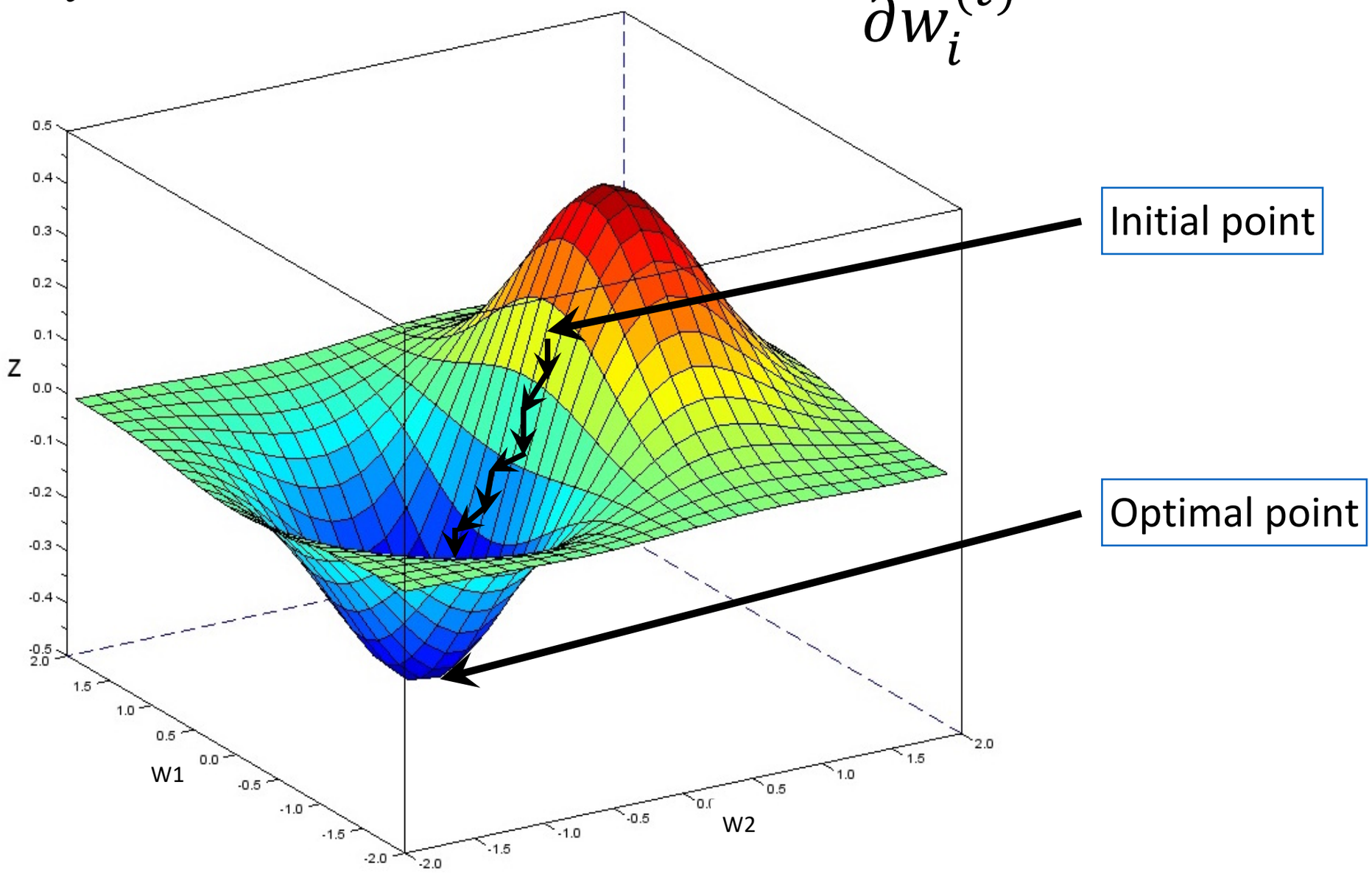


$$find(w_i) \ni \{Loss(w_i) = \sum_i \|Y - Y^*\|\}\ is\ minimized$$

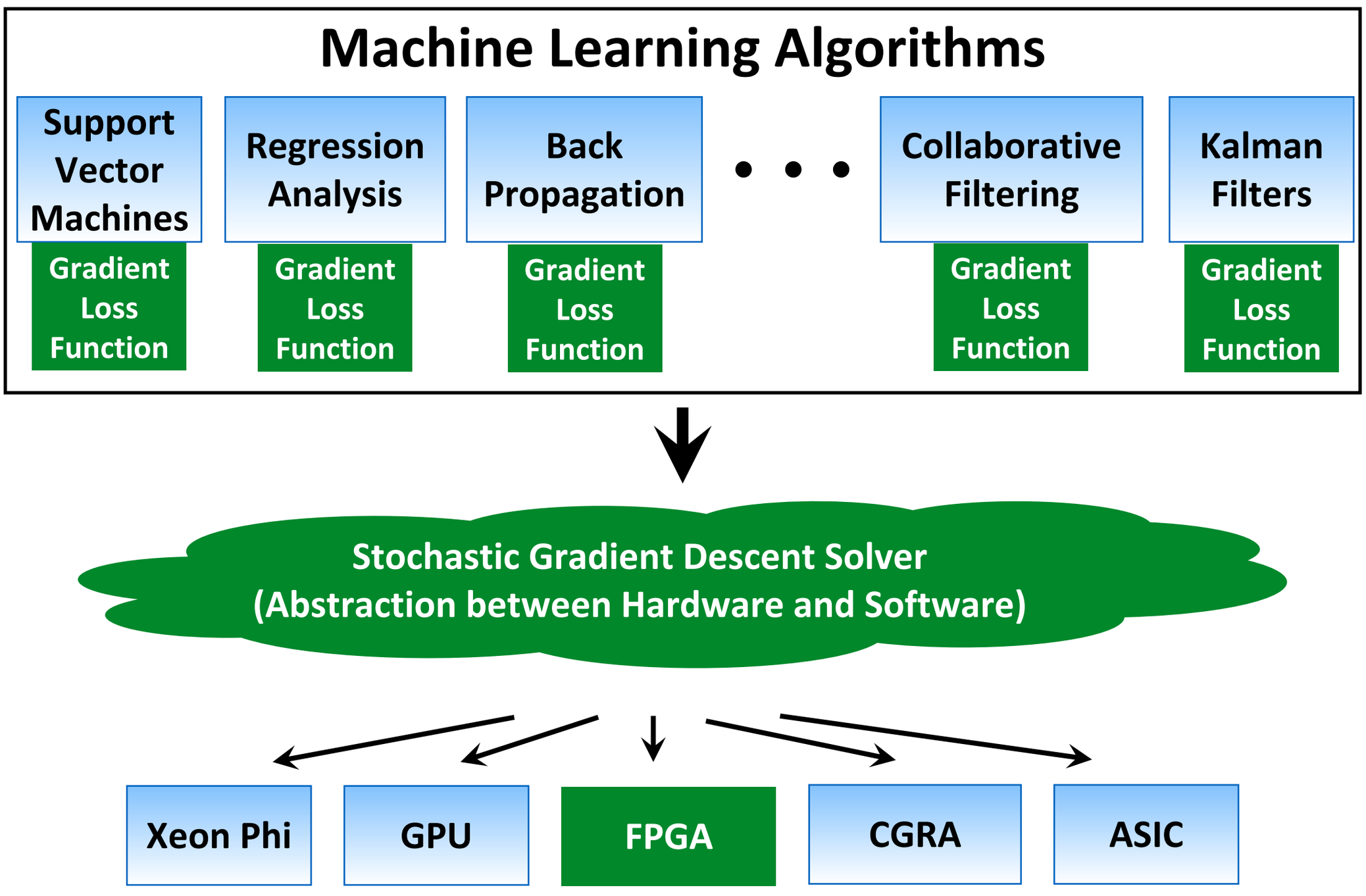
Stochastic gradient descent solver

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

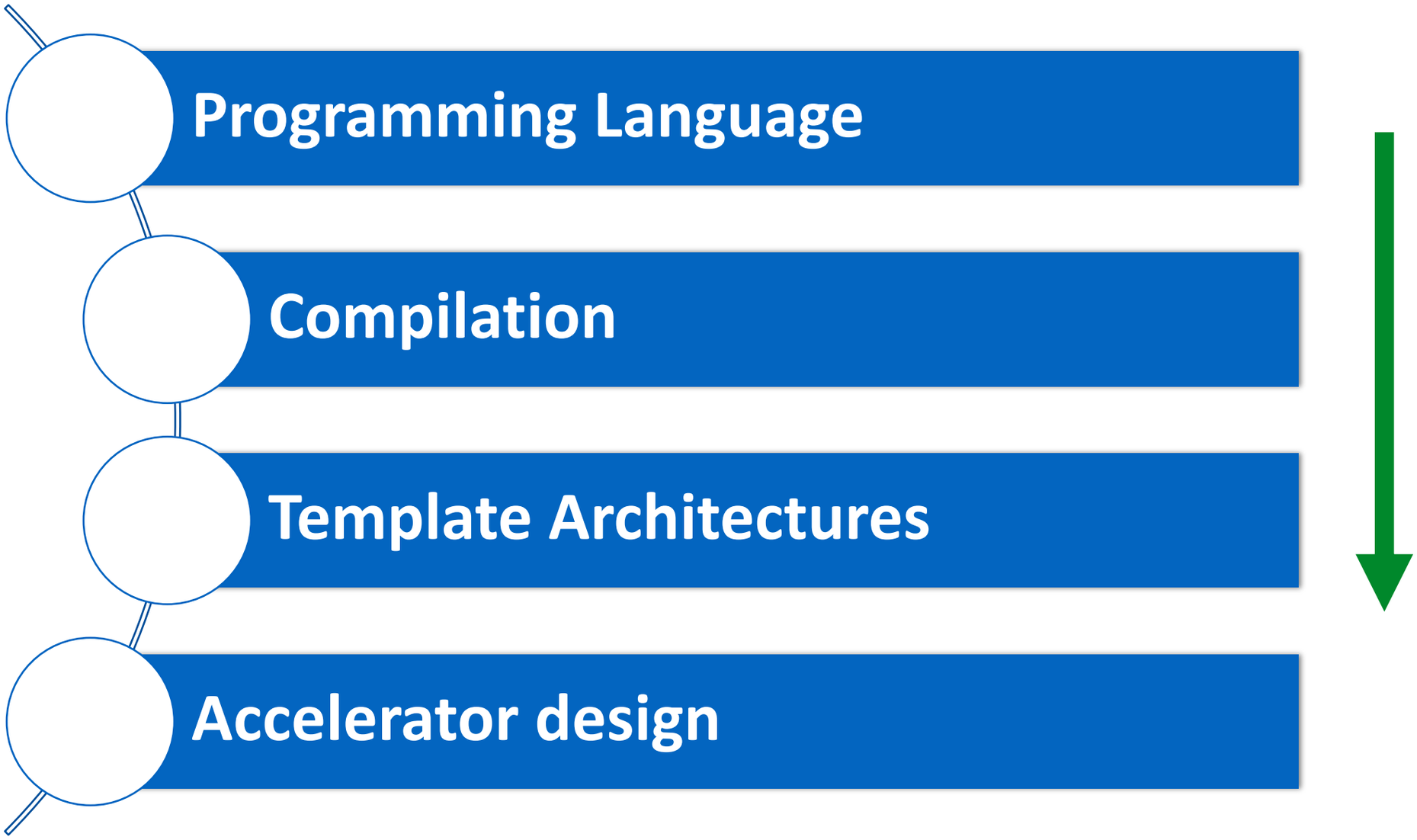
$$w_i^{(t+1)} = w_i^t - u \times \frac{\partial f\left(w_1^{(t)}, w_2^{(t)}, \ldots, w_m^{(t)}\right)}{\partial w_i^{(t)}}$$



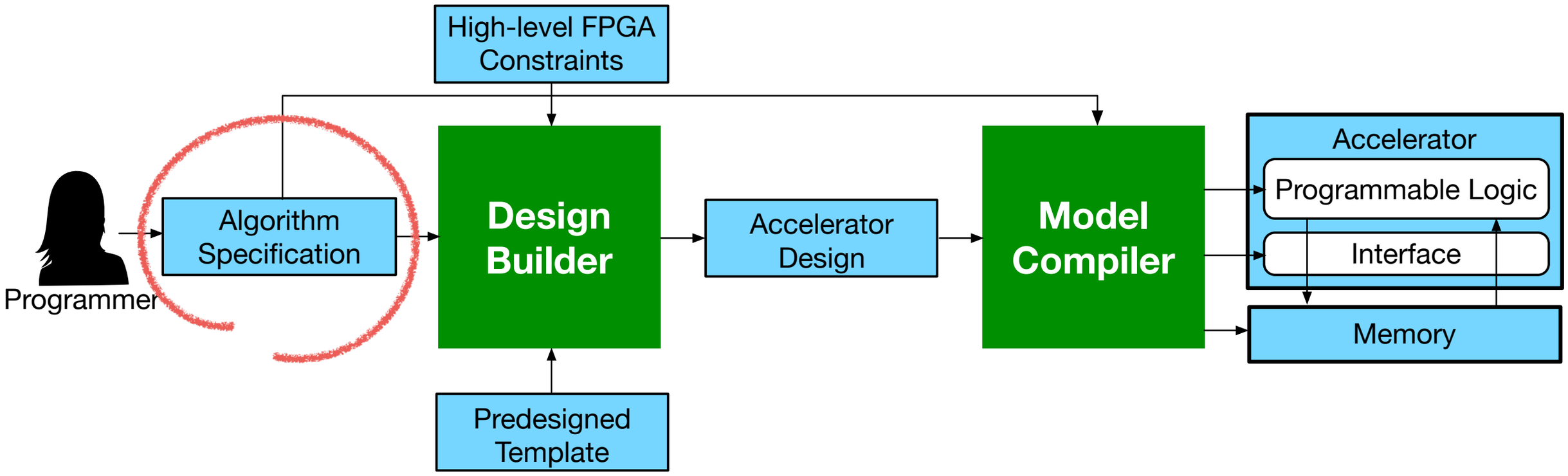
Abstraction between hardware and software



Cross-stack template-based approach



Programming interface



Enables programmer to specify the gradient of loss function

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

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

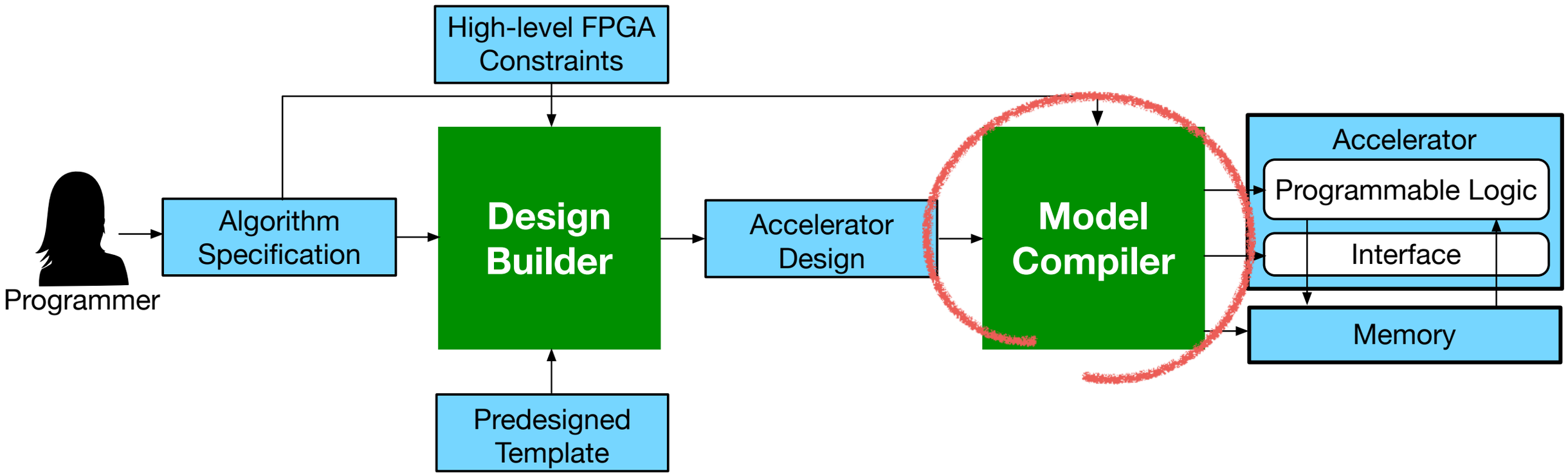
```
m = 53
lambda = 0.1

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

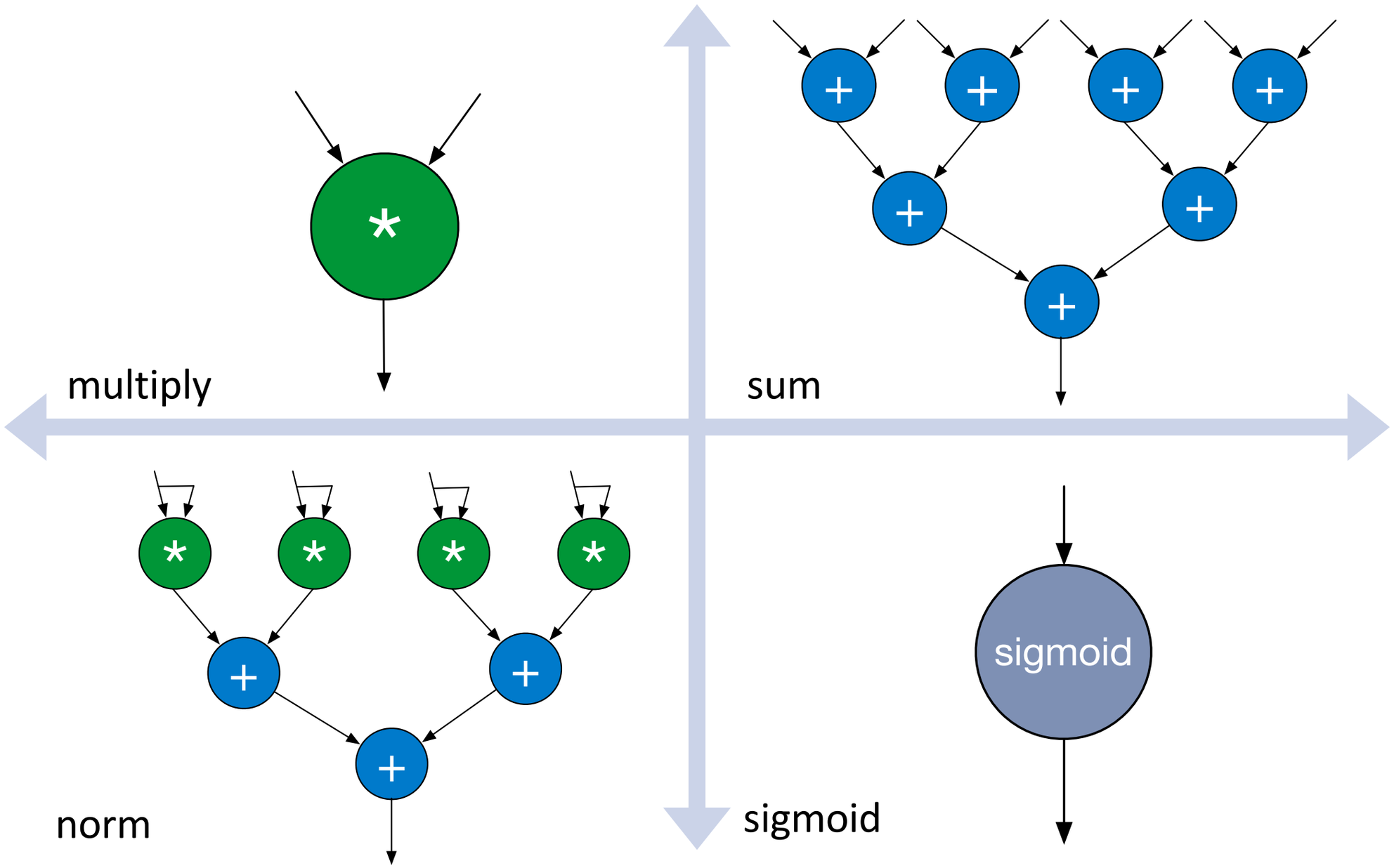


1. Appending the stochastic gradient descent solver
2. Creating the dataflow graph of the entire algorithm
3. Create a schedule for this dataflow graph

1. Appending Stochastic Gradient Descent

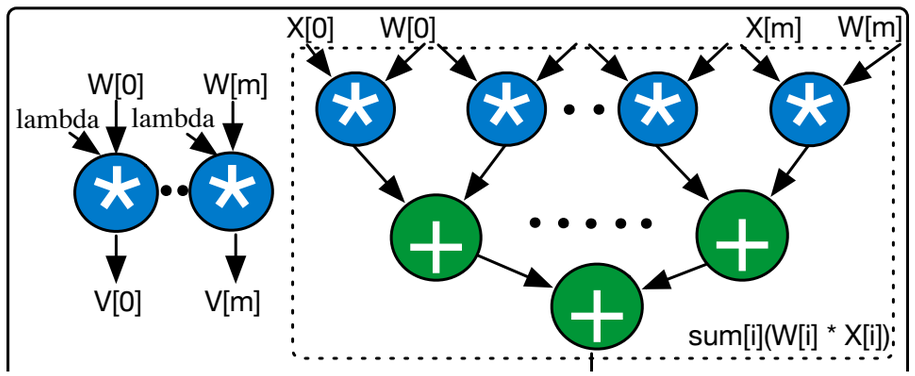
```
m = 53
lambda = 0.1
u = 0.1

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



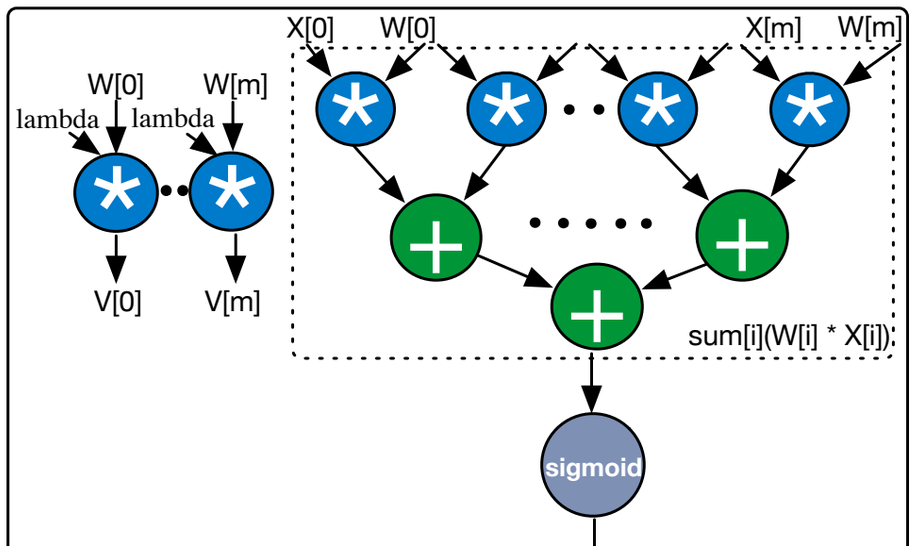
2. Dataflow graph generation

```
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];
```



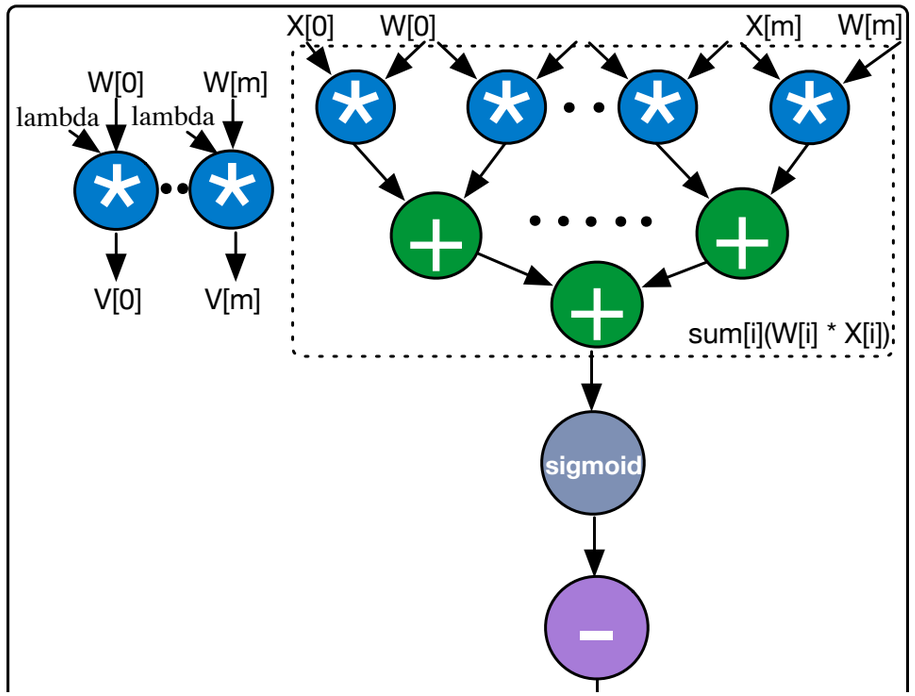
2. Dataflow graph generation

```
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];
```



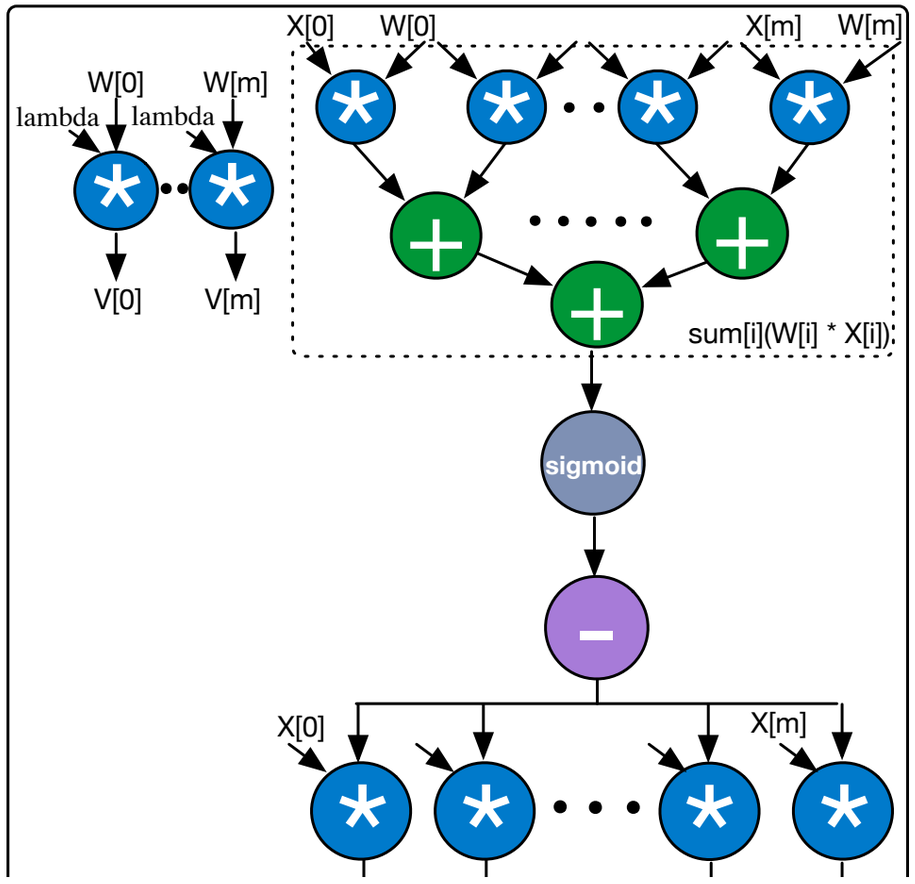
2. Dataflow graph generation

```
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];
```



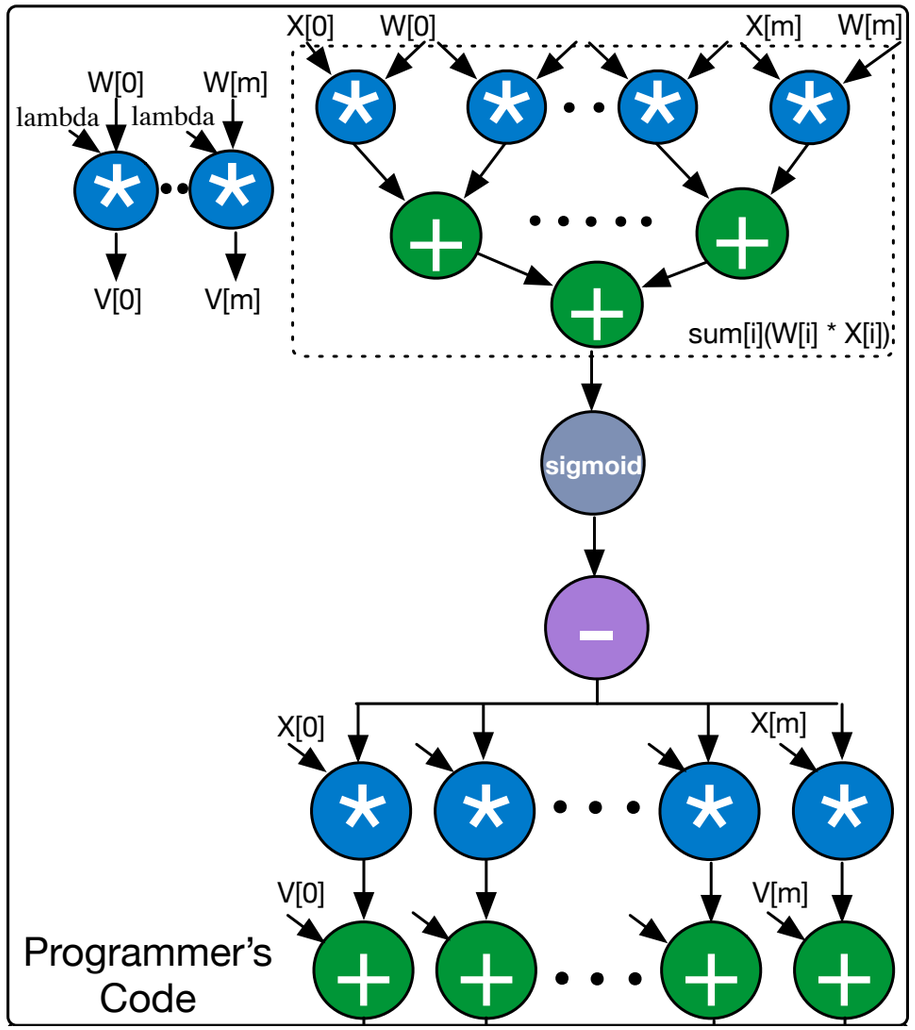
2. Dataflow graph generation

```
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];
```



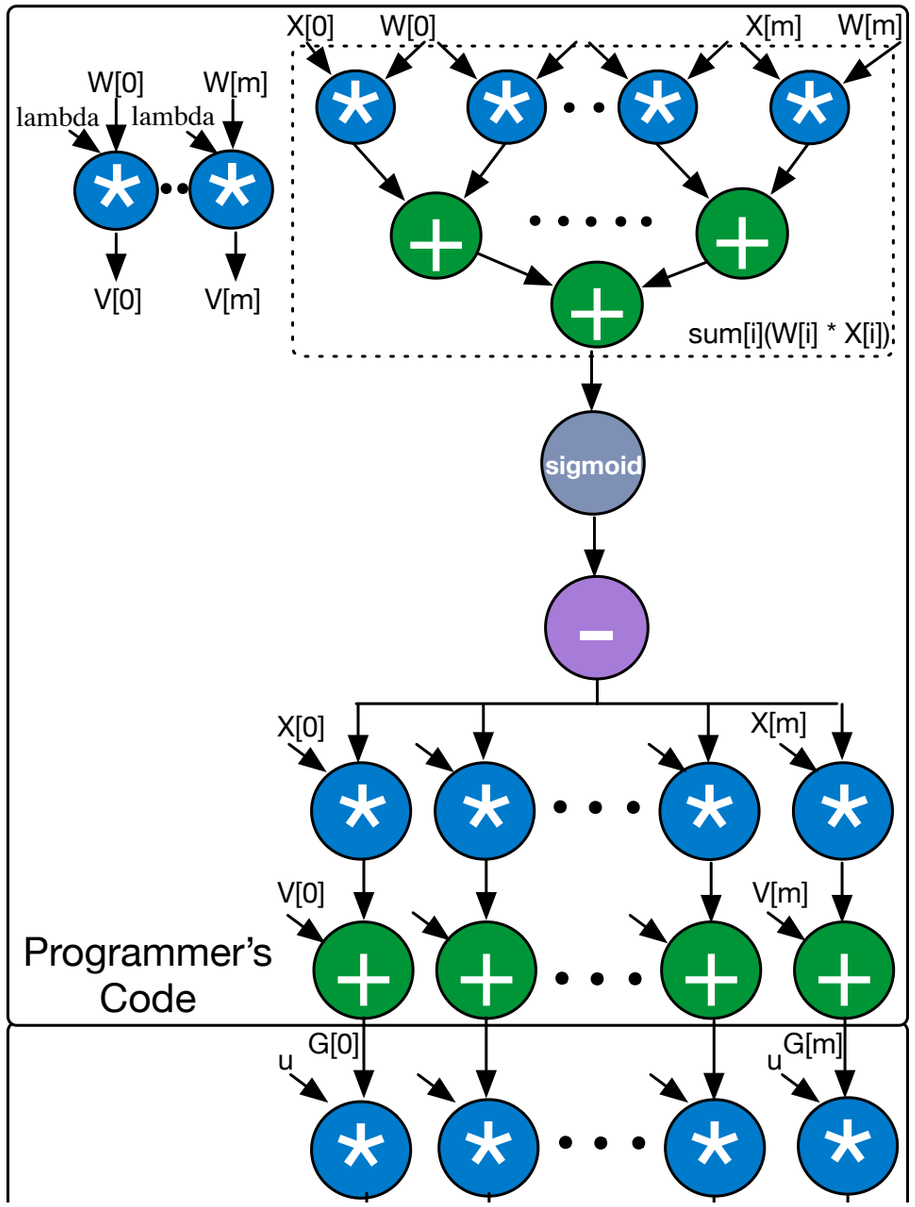
2. Dataflow graph generation

```
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];
```



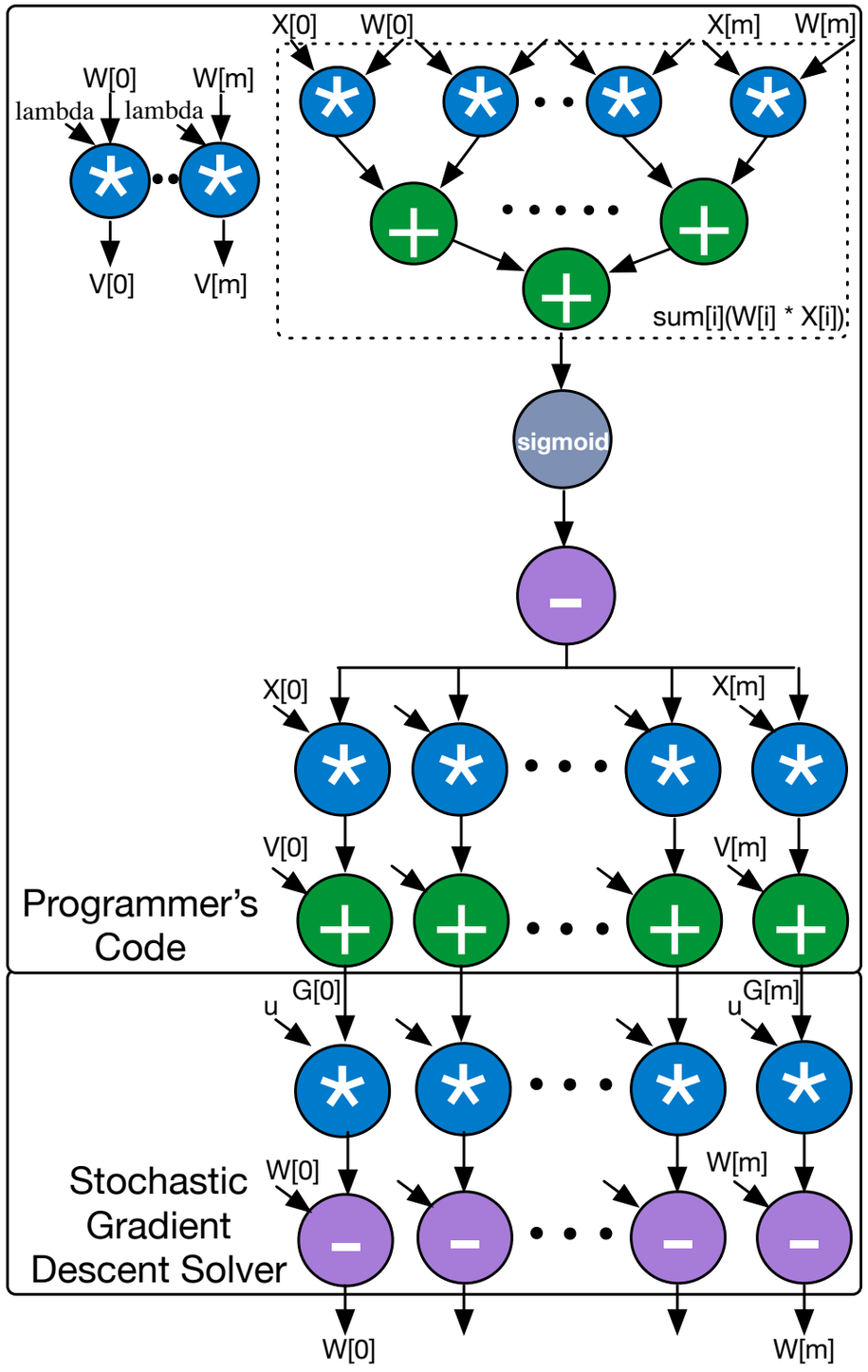
2. Dataflow graph generation

```
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];
```

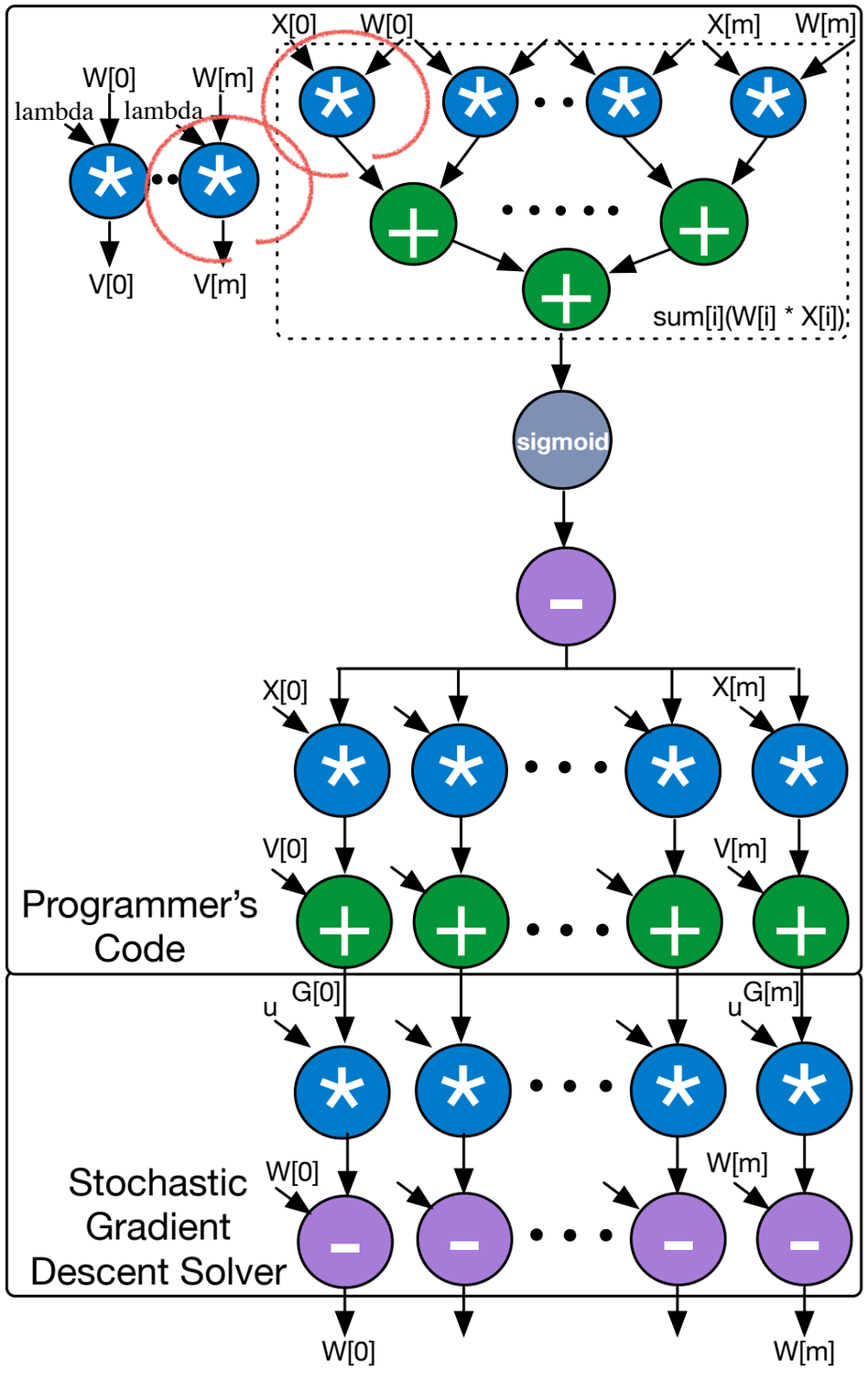


2. Dataflow graph generation

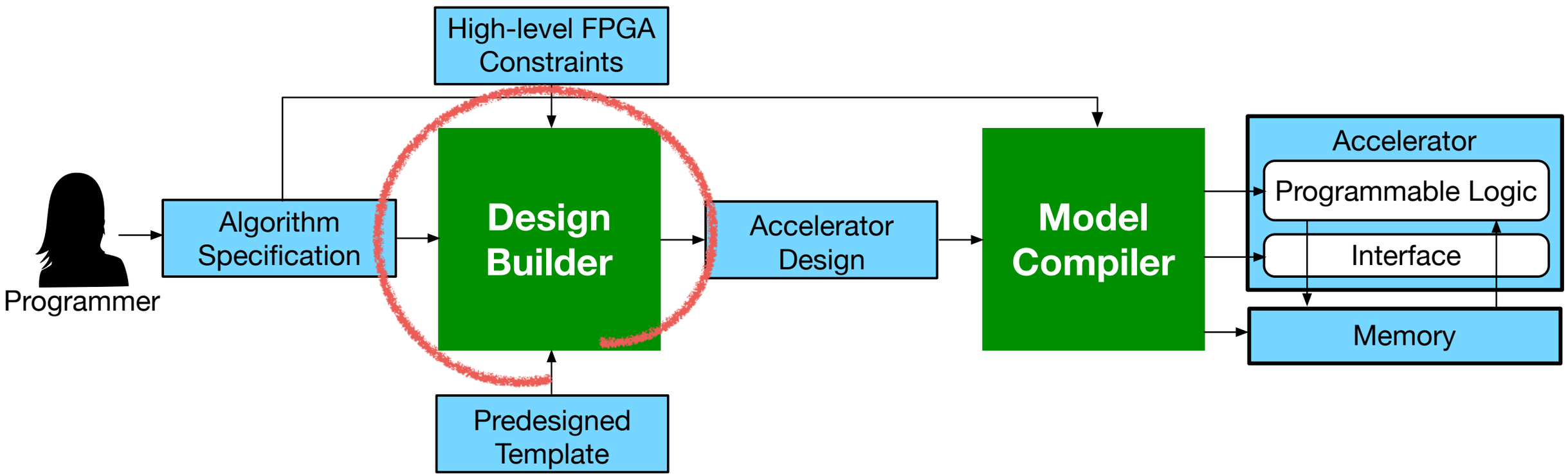
```
m = 53
lambda = 0.1
u = 0.1

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

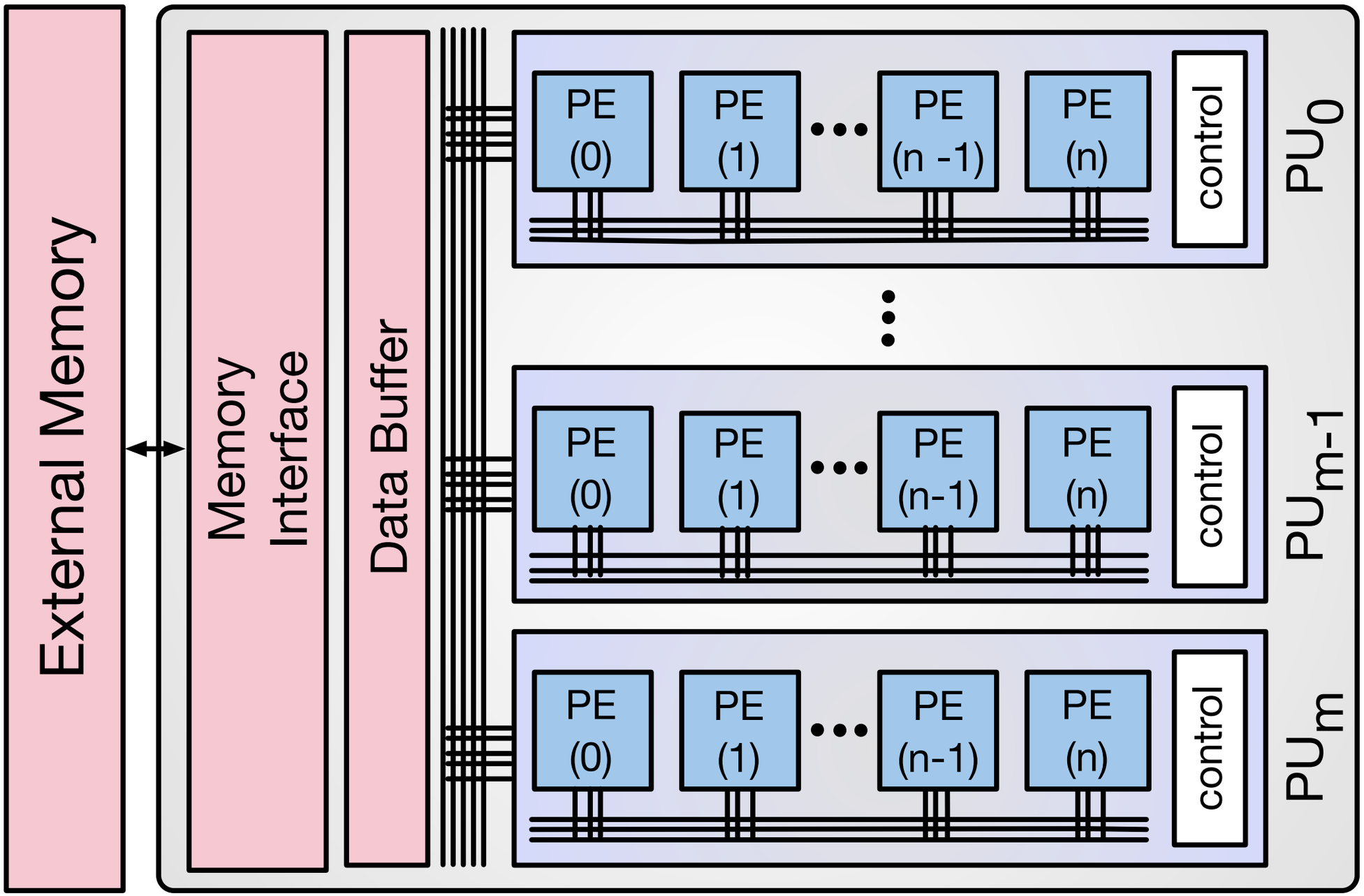


Design Builder

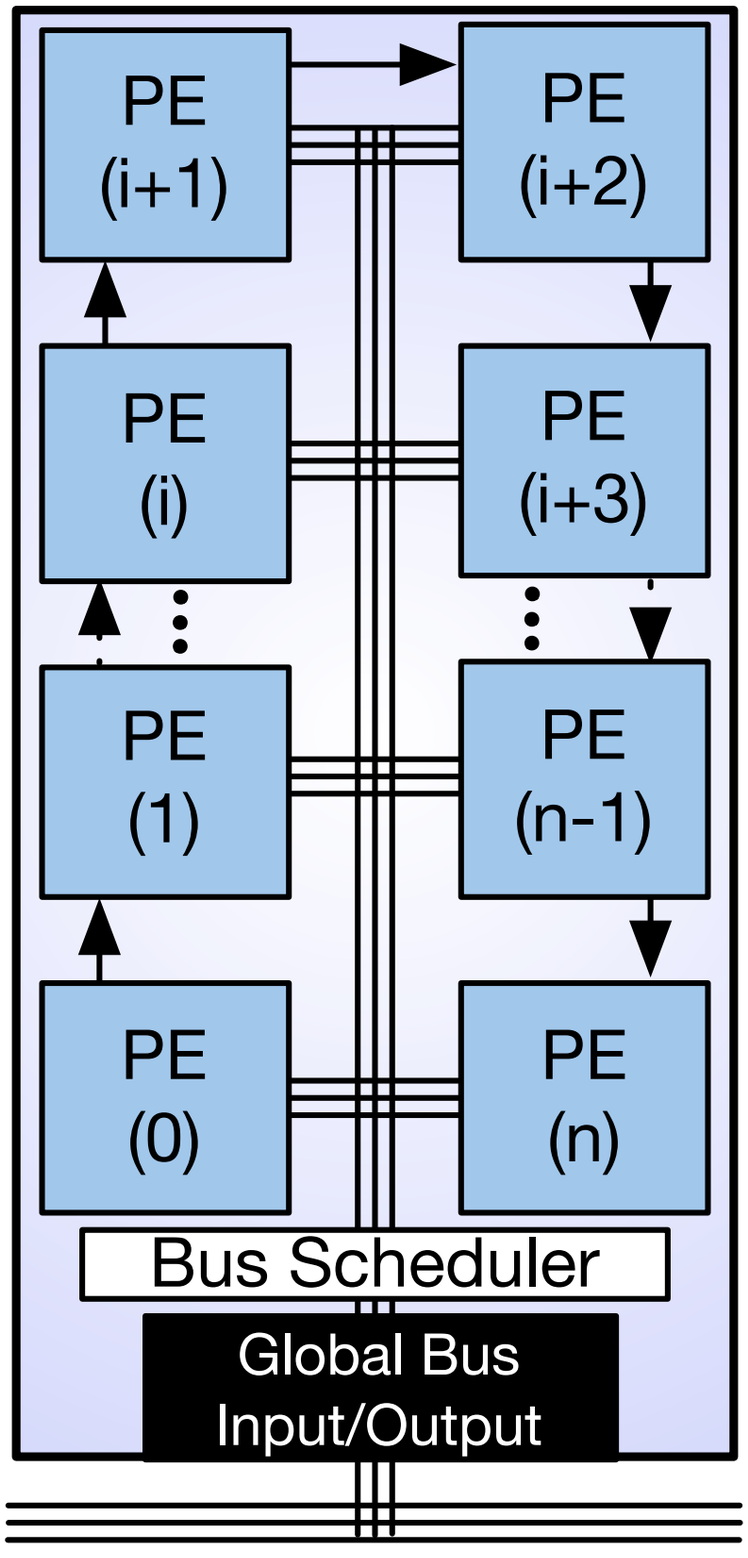


Together with the model compiler generates the final accelerator design

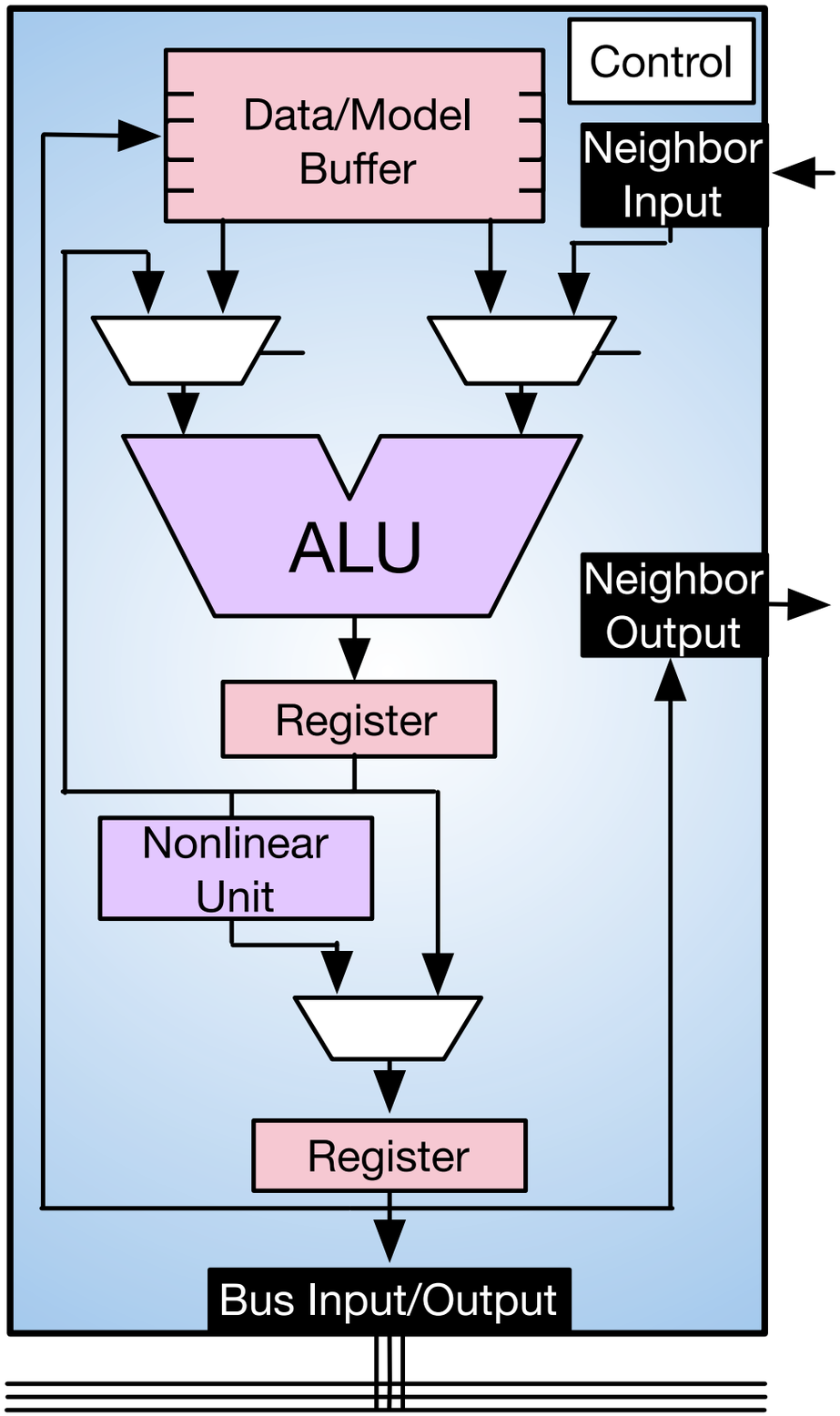
Hierarchical template design of accelerator



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

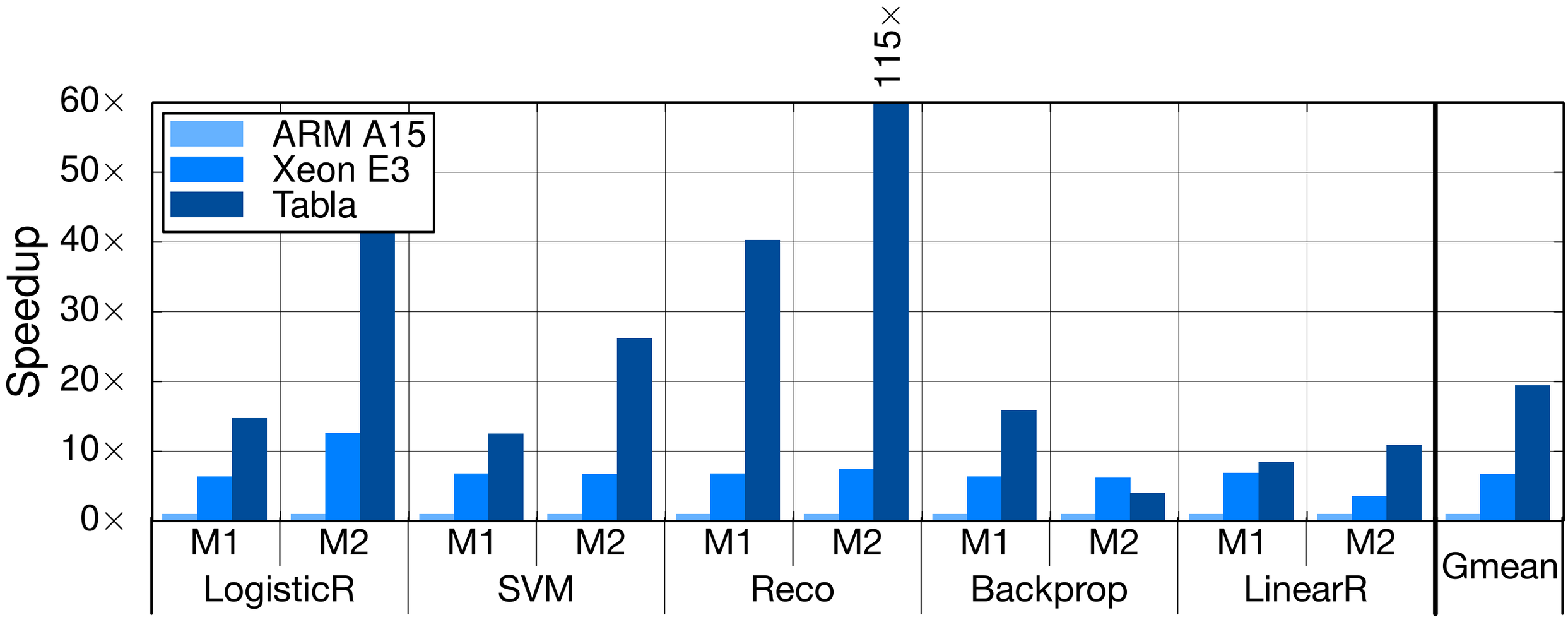


TABLA generated accelerators provide 19× speedup over ARM and 2.9× speedup over Xeon

Static scheduling alleviates the traditional Von-Neumann overheads

Speedup in comparison to GPUs

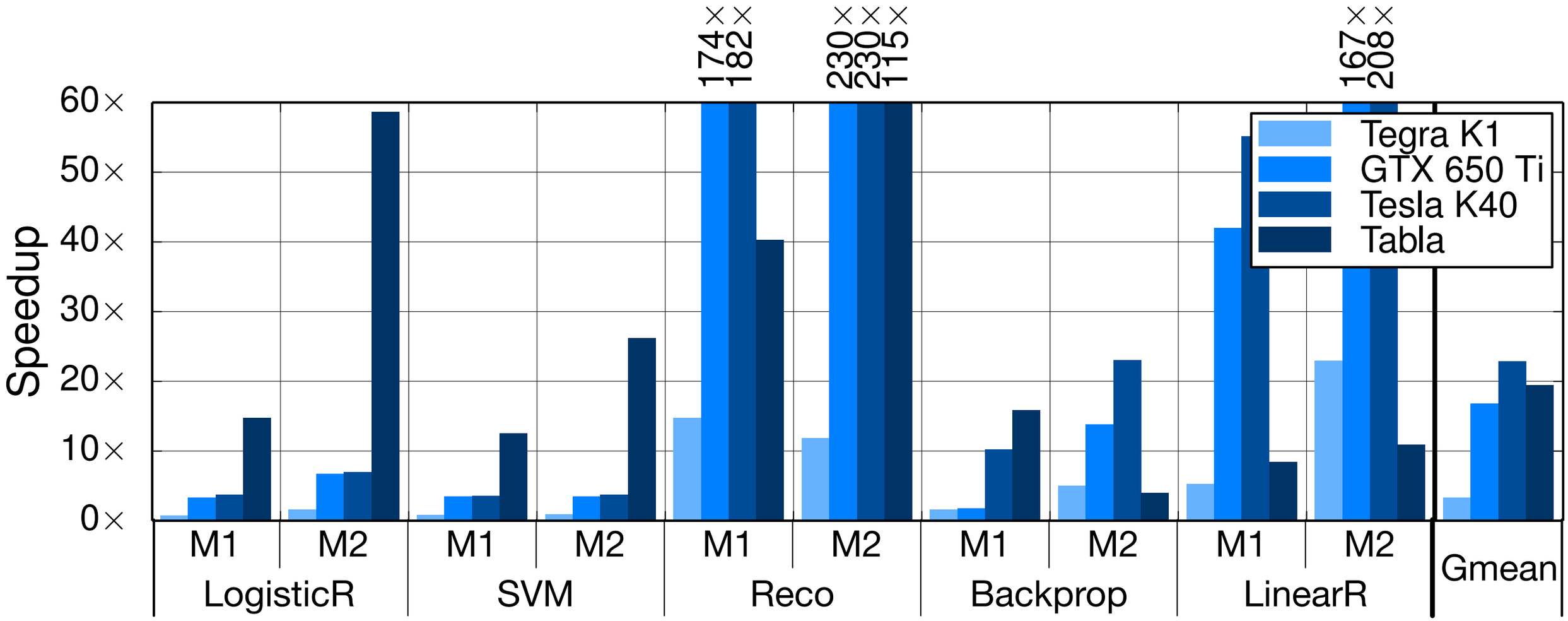


TABLA generated accelerators provide 5.9× speedup over Tegra K1, 1.16x over GTX 650 Ti and 1.18x slowdown over Tesla K40

Performance-per-Watt in comparison to GPUs

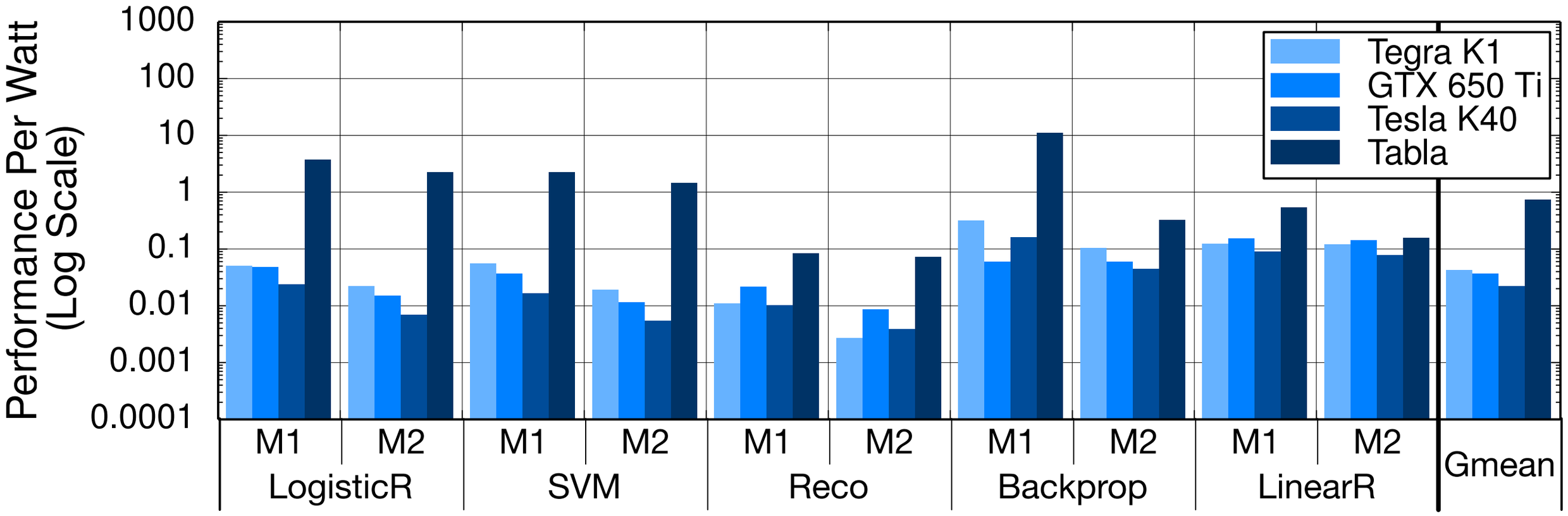


TABLA generated accelerators provide 17× performance-per-Watt over Tegra K1, 20× over GTX 650 Ti and 33× over Tesla K40

Specialized template designs extract fine-grained parallelism while consuming less power

Ongoing and future directions

Scale out Acceleration
[MICRO 2017]

Integration with Database Management Systems
[VLDB 2018]

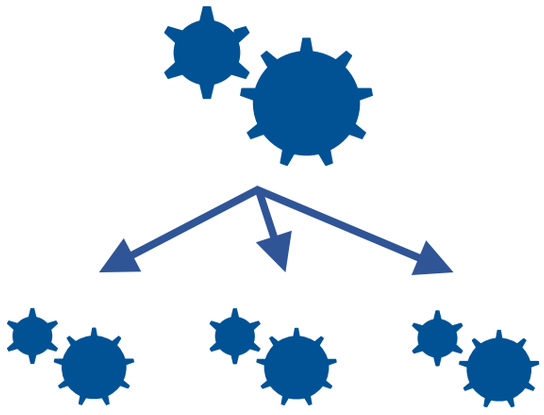
Approximating Data and Accelerating Communication
[MICRO 208]

Accelerated Motion Planning and Control for Robotics
[ISCA 2018]

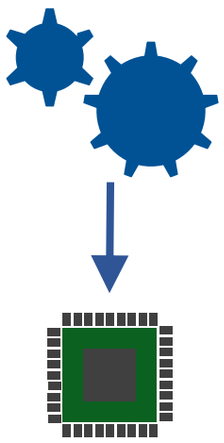
Scale-Out Acceleration for Machine Learning

Challenges

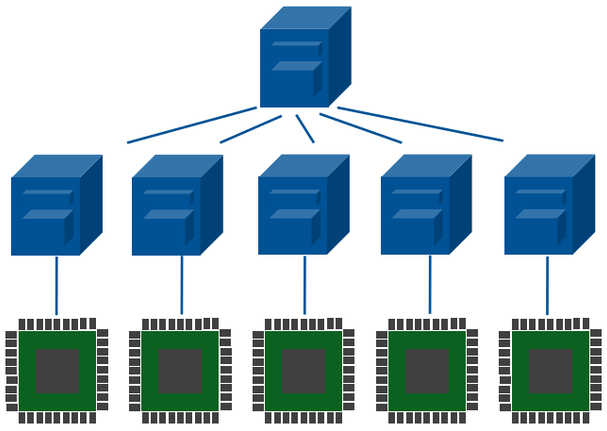
1. How to distribute ML training?



2. How to design customizable accelerators?



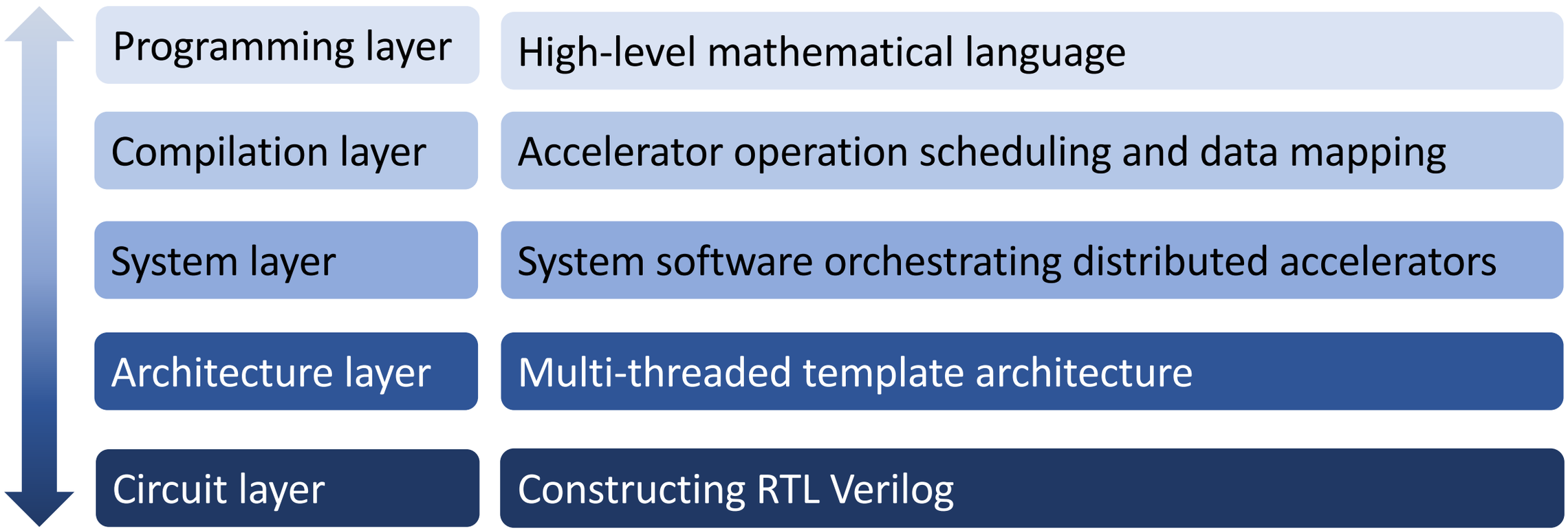
3. How to reduce the overhead of distributed coordination?



We need a full stack

CoSMIC

Computing Stack for ML Acceleration In the Cloud

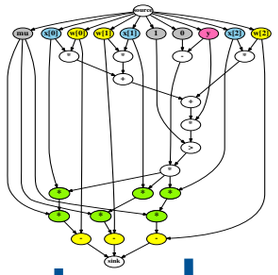


CoSMIC workflow

Model Specification

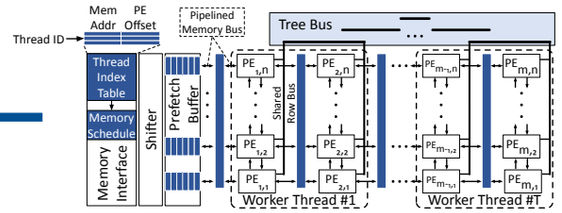
1 Translator

Dataflow Graph

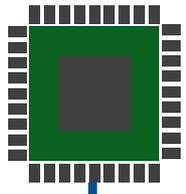


2 Planner

Multi-threaded Template Architecture



Accelerator Architecture



3 Compiler

Operation Schedule/Map

4 Constructor

Circuit Constraints

RTL Verilog for Accelerator

System Specification

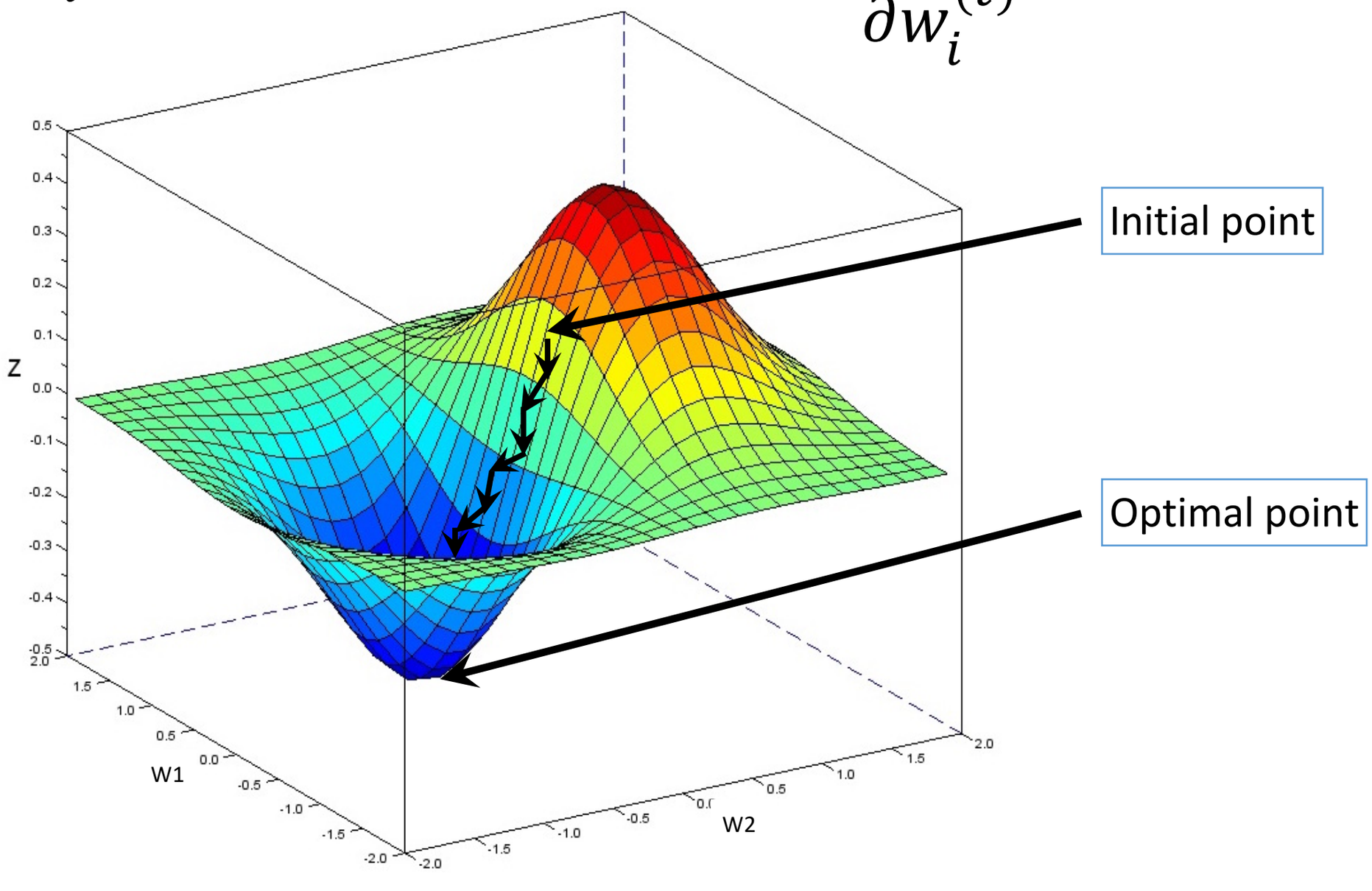
5 System Director

System Software

Stochastic gradient descent solver

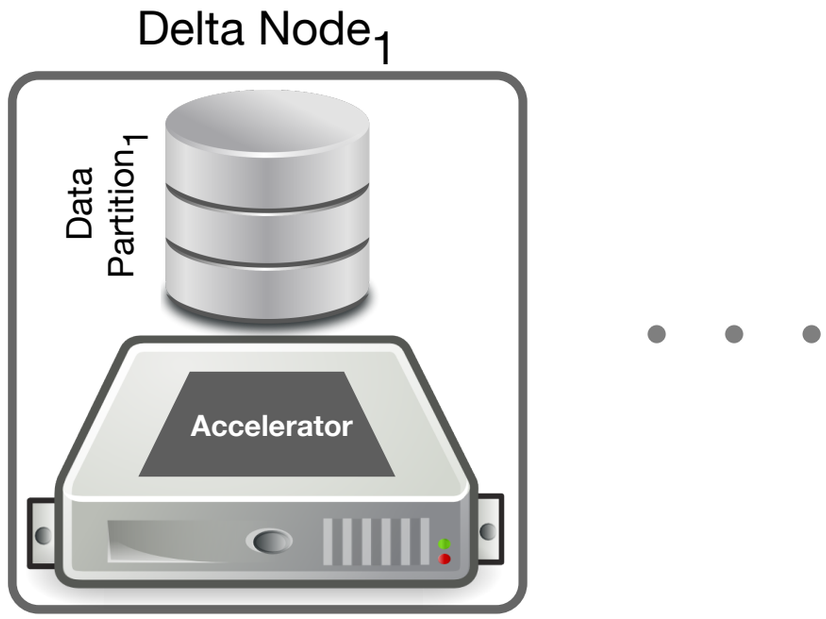
$$Loss\ function\ (w_i) = f\left(w_1^{(t)}, w_2^{(t)}, \ldots, w_m^{(t)}\right)$$

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

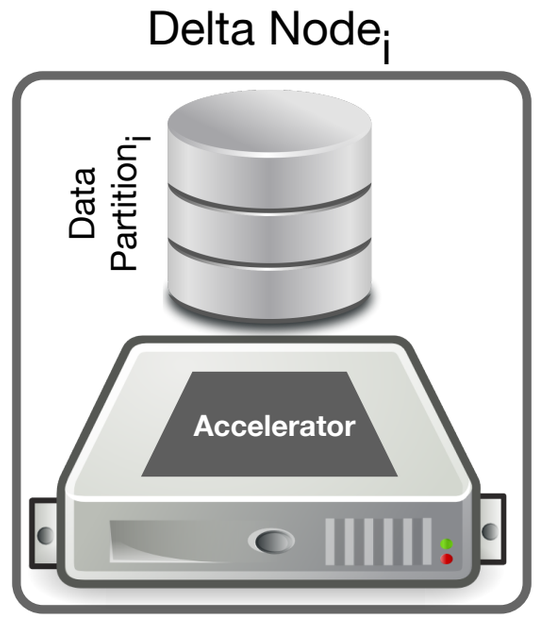


Leverage linearity of differentiation for distributed learning

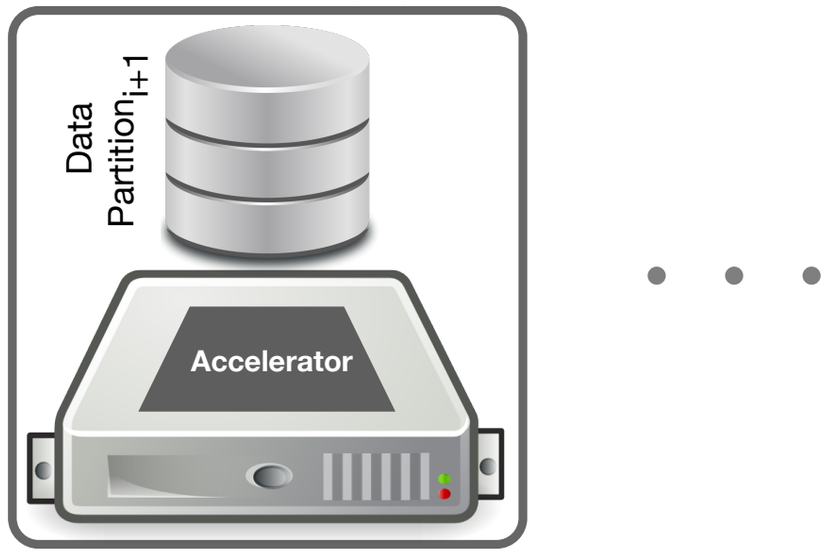
$$\Delta W_1^{(t+1)} = \frac{\delta f}{\delta W^{(t)}}$$

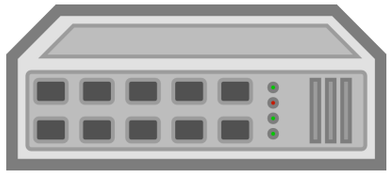


$$\Delta W_i^{(t+1)} = \frac{\delta f}{\delta W^{(t)}}$$

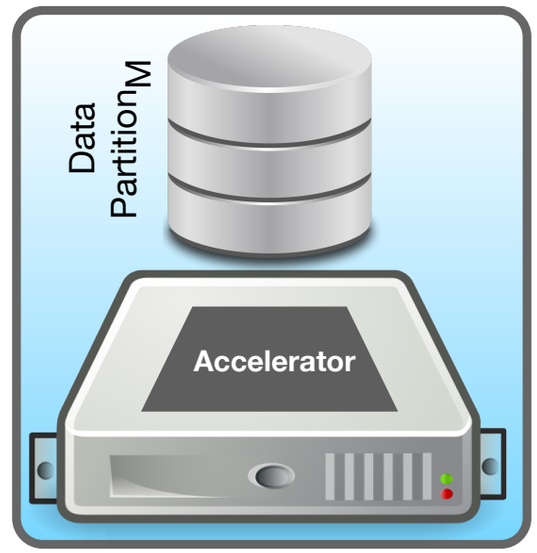


$$\Delta W_{i+1}^{(t+1)} = \frac{\delta f}{\delta W^{(t)}}$$





$$\Delta W_M^{(t+1)} = \frac{\delta f}{\delta W^{(t)}}$$

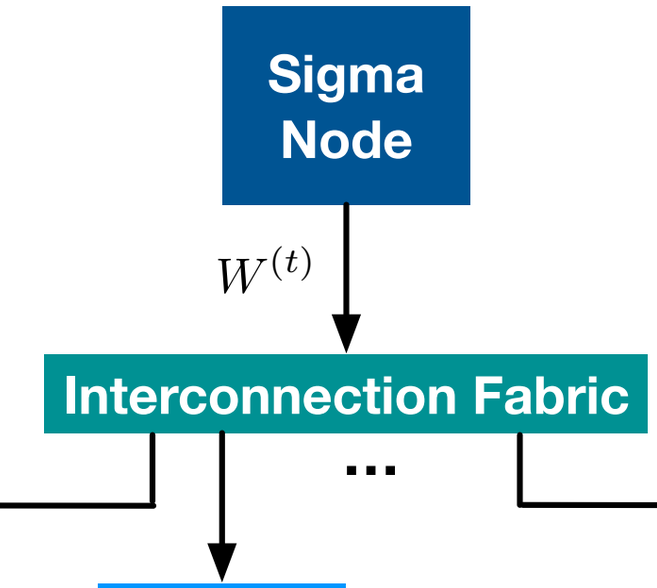


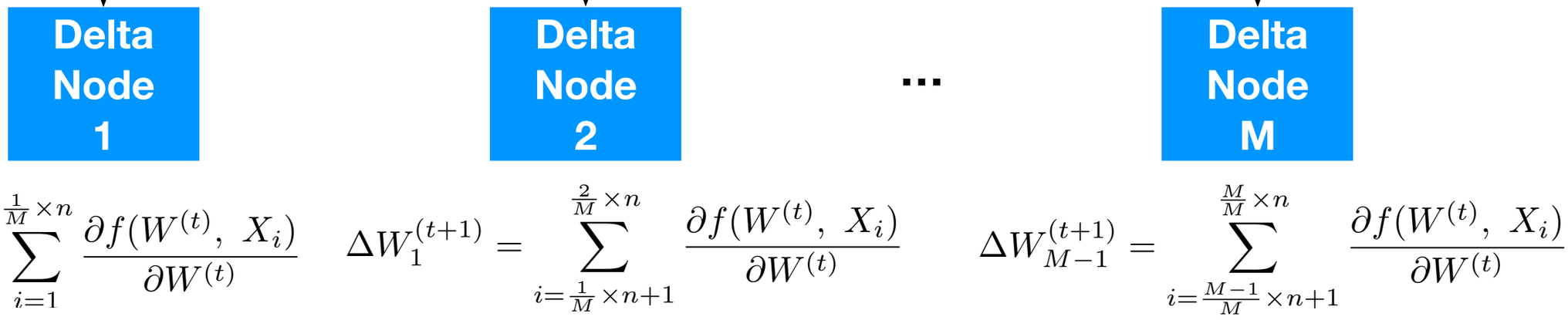
$$\Delta W^{(t+1)} = \frac{\sum_i \Delta W_i^{(t+1)}}{M}$$

Execution model (Phase I)

1 Communicate Current Model

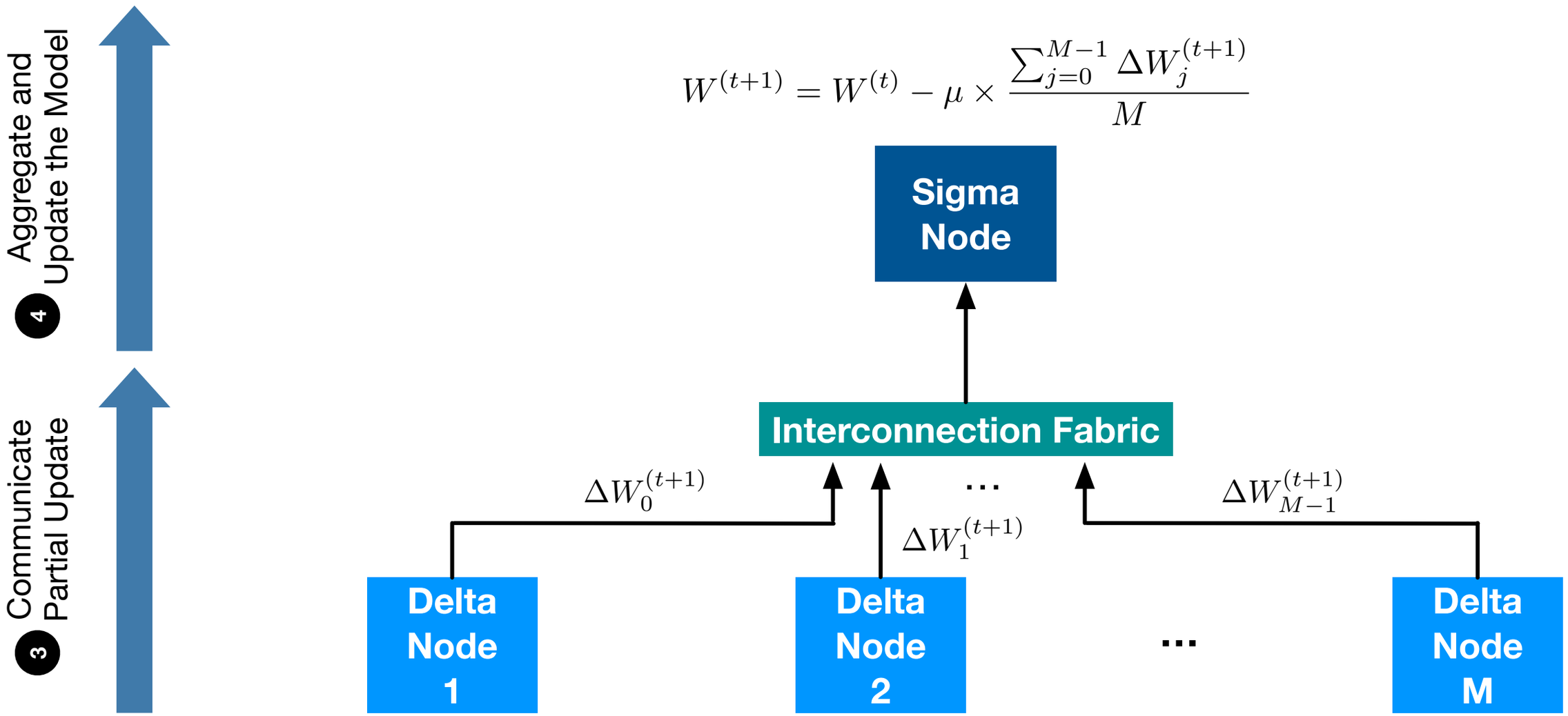
2 Compute Partial Updates



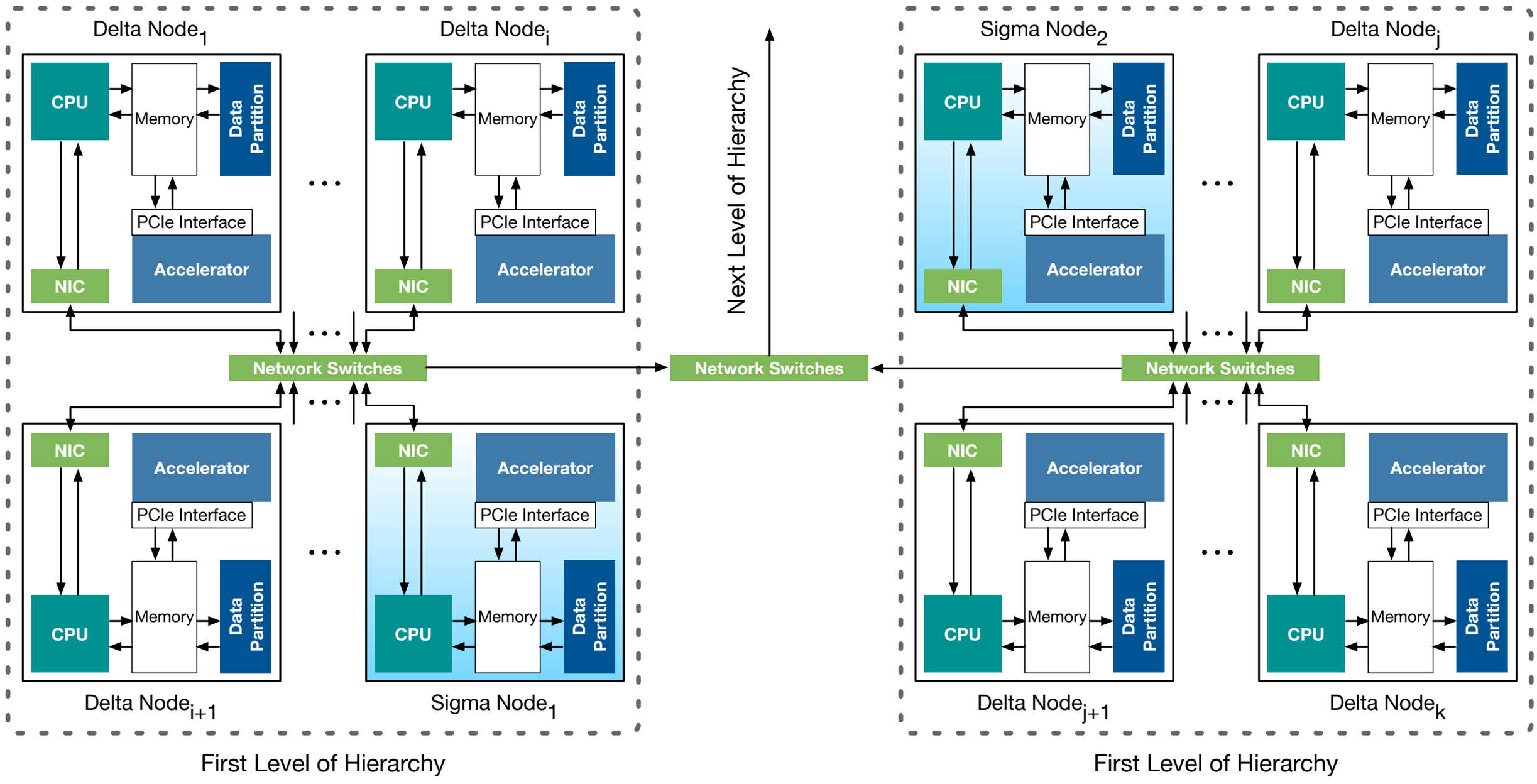


$$\Delta W_0^{(t+1)} = \sum_{i=1}^{\frac{1}{M}\times n} \frac{\partial f(W^{(t)},\ X_i)}{\partial W^{(t)}} \qquad \Delta W_1^{(t+1)} = \sum_{i=\frac{1}{M}\times n+1}^{\frac{2}{M}\times n} \frac{\partial f(W^{(t)},\ X_i)}{\partial W^{(t)}} \qquad \Delta W_{M-1}^{(t+1)} = \sum_{i=\frac{M-1}{M}\times n+1}^{\frac{M}{M}\times n} \frac{\partial f(W^{(t)},\ X_i)}{\partial W^{(t)}}$$

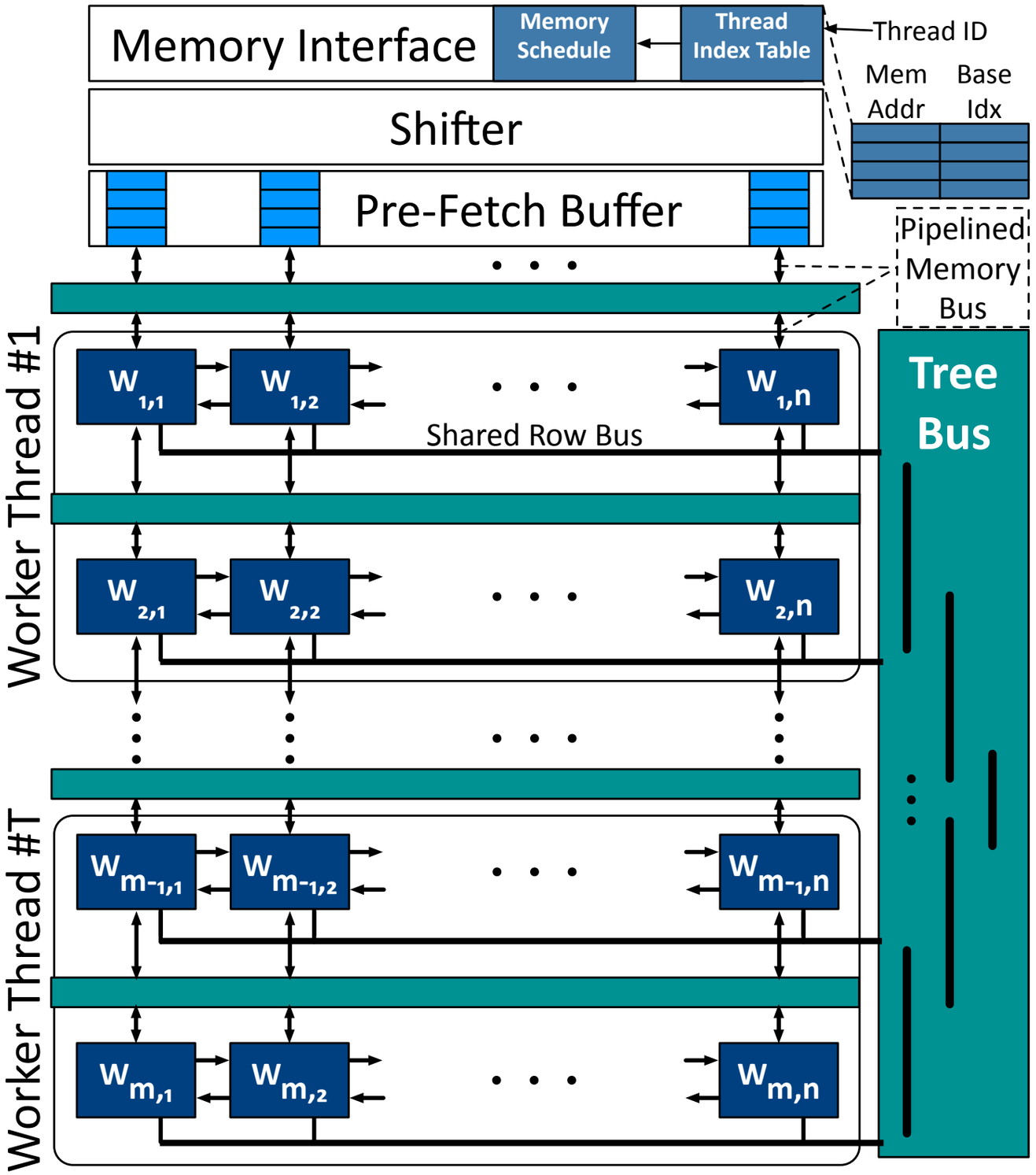
Execution model (Phase II)



Hierarchical design based-on commodity systems and network



Multi-threaded template architecture



Benchmarks for distributed learning

Name	Description	Algorithm	Model Size (KB)	Training Data Size (GB)	Lines of Code
mnist	Handwritten digit recognition	Backpropagation	2,432 KB	2.9 GB	55
acoustic	Speech recognition modeling		1,527 KB	5.6 GB	55
stock	Stock price prediction	Linear Regression	31 KB	14.7 GB	23
texture	Image texture recognition		64 KB	17.9 GB	23
tumor	Tumor classification	Logistic Regression	8 KB	10.4 GB	22
cancer1	Prostate cancer diagnosis		24 KB	13.5 GB	22
movielens	Movielens recommender system	Collaborative Filtering	1,176 KB	0.6 GB	42
netflix	Netflix recommender system		2,854 KB	2.0 GB	42
face	Human face detection	Support Vector Machine	7 KB	15.9 GB	27
cancer2	Cancer diagnosis		28 KB	20.0 GB	27

List of results

1. Performance comparison with Spark

2. Breakdown of performance between computation and system coordination

3. Performance comparison of FPGA-CoSMIC with Programmable ASIC and GPUs

4. Power Efficiency comparison between FPGA, Programmable ASICs and GPUs

5. Sensitivity to mini-batch size, compute units, memory bandwidth

6. Design space exploration for multithreading

7. Comparison of template architecture with prior accelerator designs

List of results

1. Performance comparison with Spark

2. Breakdown of performance between computation and system coordination

3. Performance comparison of FPGA-CoSMIC with Programmable ASIC and GPUs

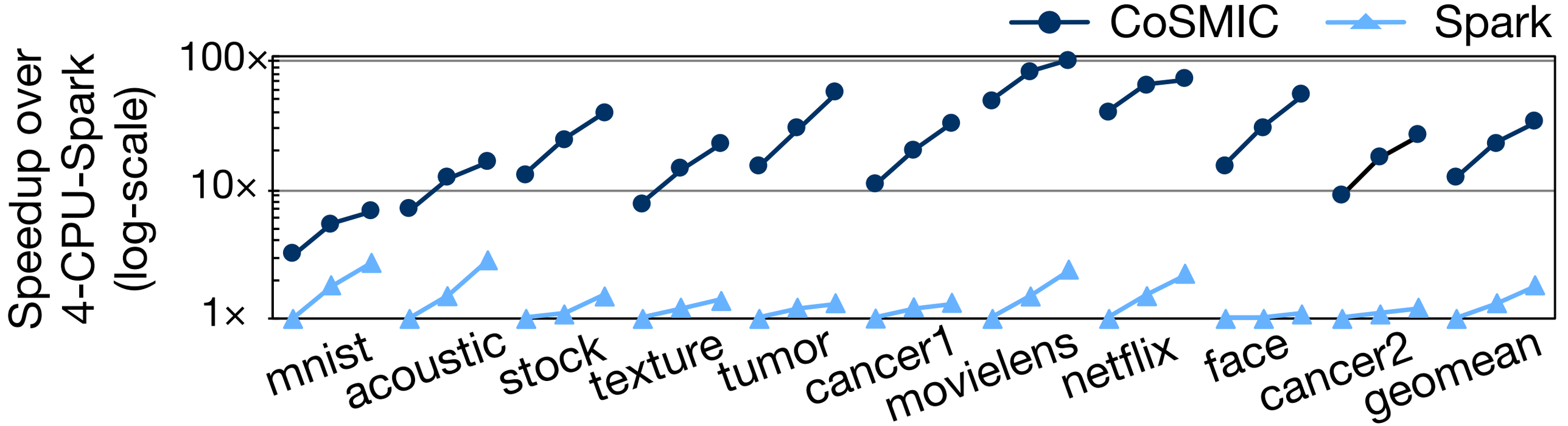
4. Power Efficiency comparison between FPGA, Programmable ASICs and GPUs

5. Sensitivity to mini-batch size, compute units, memory bandwidth

6. Design space exploration for multithreading

7. Comparison of template architecture with prior accelerator designs

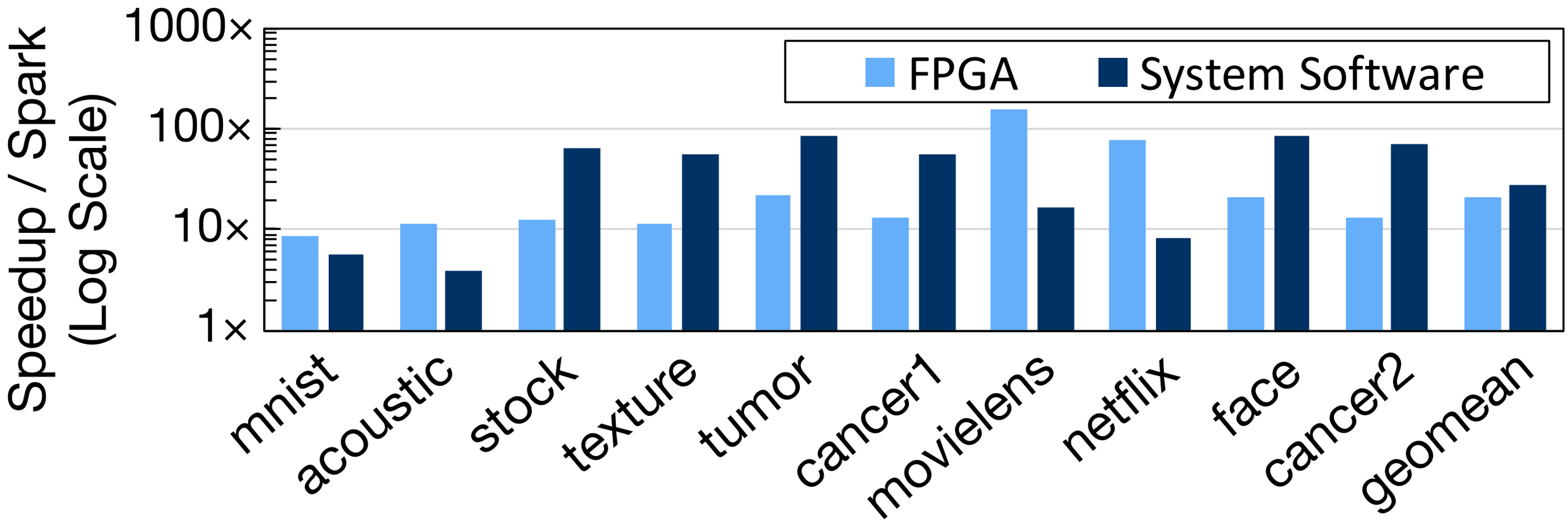
Speedup in comparison with Spark



16-node CoSMIC with UltraScale+ FPGAs offer 18.8× speedup over 16-node Spark with Xeon E3 Skylake CPUs

Scaling from 4 to 16 nodes with CoSMIC yields 2.7× improvement while Spark offers 1.8×.

Speedup breakdown between computation and system coordination

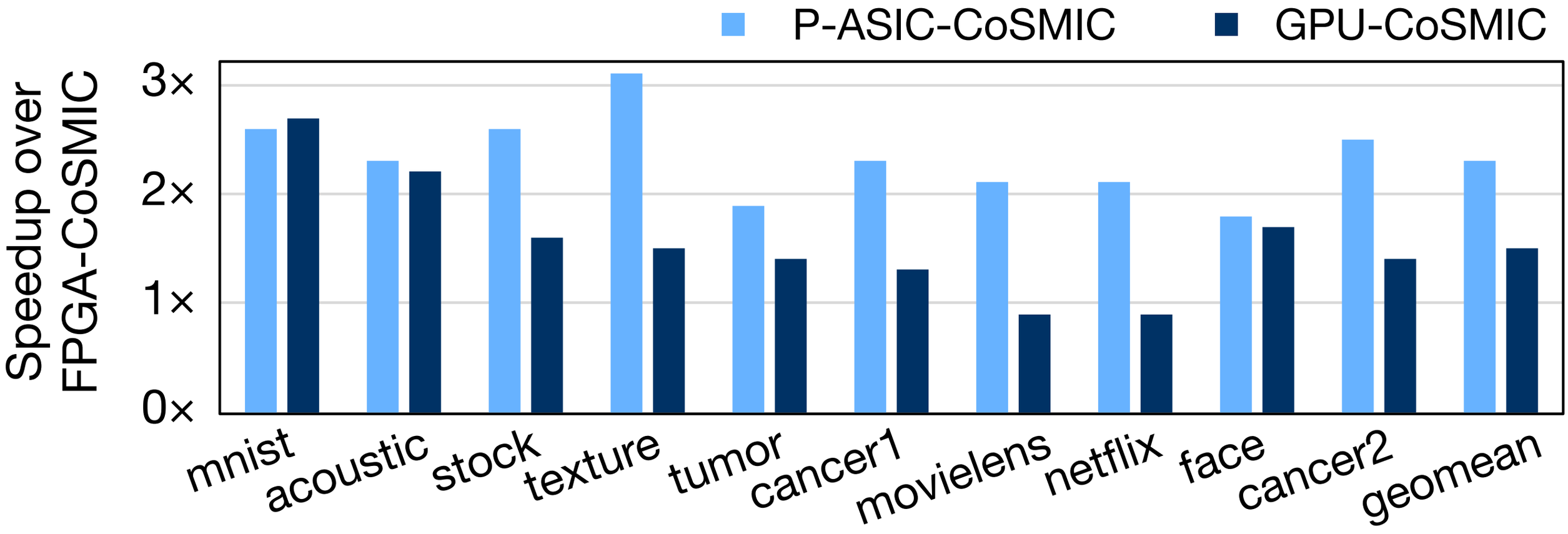


CoSMIC system software makes system coordination (34%) 28.4× faster

CoSMIC FPGA hardware makes computation (66%) 20.7× faster

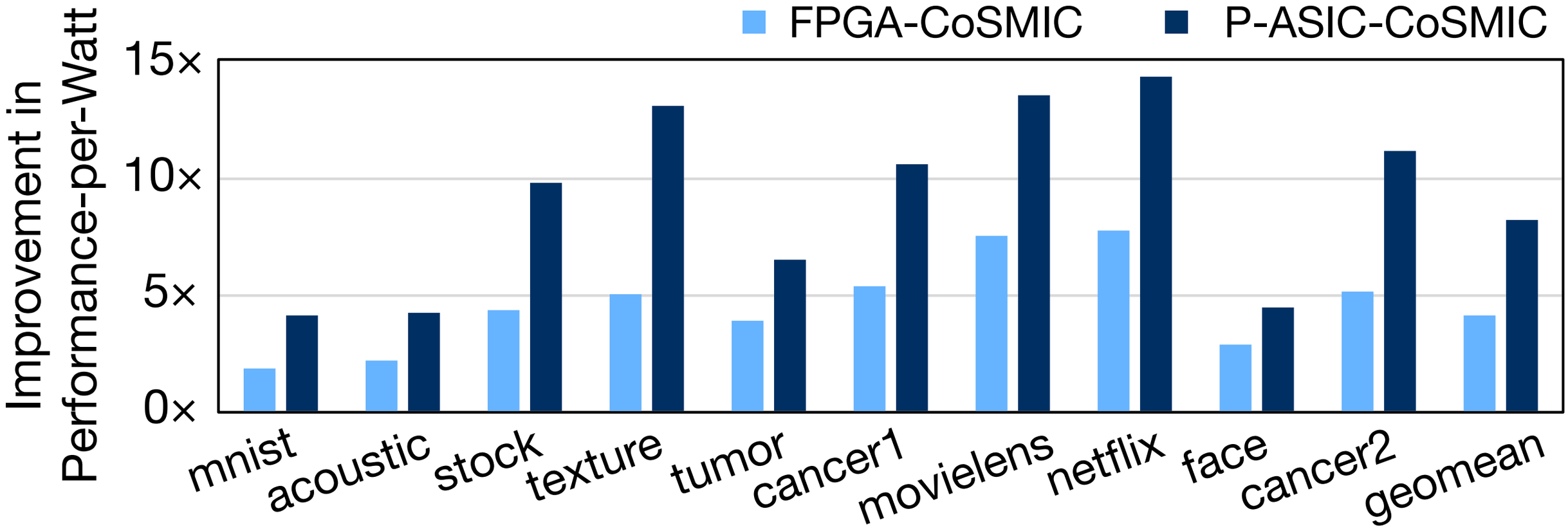
The overall speedup is 22.8× speedup

Hardware is not enough, we need a novel system stack



P-ASIC, and GPU provide 2.3× and 1.5× extra speedup over FPGA CoSMIC, which is 22.8× faster than Spark

Performance-per-Watt comparison



With CoSMIC FPGAs and P-ASICs provide 4.2× and 8.2× higher Performance-per-Watt than GPUs, respectively

Ongoing and future directions

Scale out Acceleration
[MICRO 2017]

Integration with Database Management Systems
[VLDB 2018]

Approximating Data and Accelerating Communication
[MICRO 208]

Accelerated Motion Planning and Control for Robotics
[ISCA 2018]

Thank you