Machine Learning Migration for Efficient Near-Data Processing

Authors: Aline Cordeiro

Sairo Santos

Francis B. Moreira

Paulo C. Santos

Luigi Carro

Marco Zanata Alves

Increase of digital system's usage

- Data transactions
  - Registration, creation, copy, share, download
- For 2020, Gantz et al expected 40 trilion GB of data
  - Doubling every 2 years the digital volume of data until it
- Machine Learning applications to analyze huge amount of data
  - High computational performance
  - Memory capacity

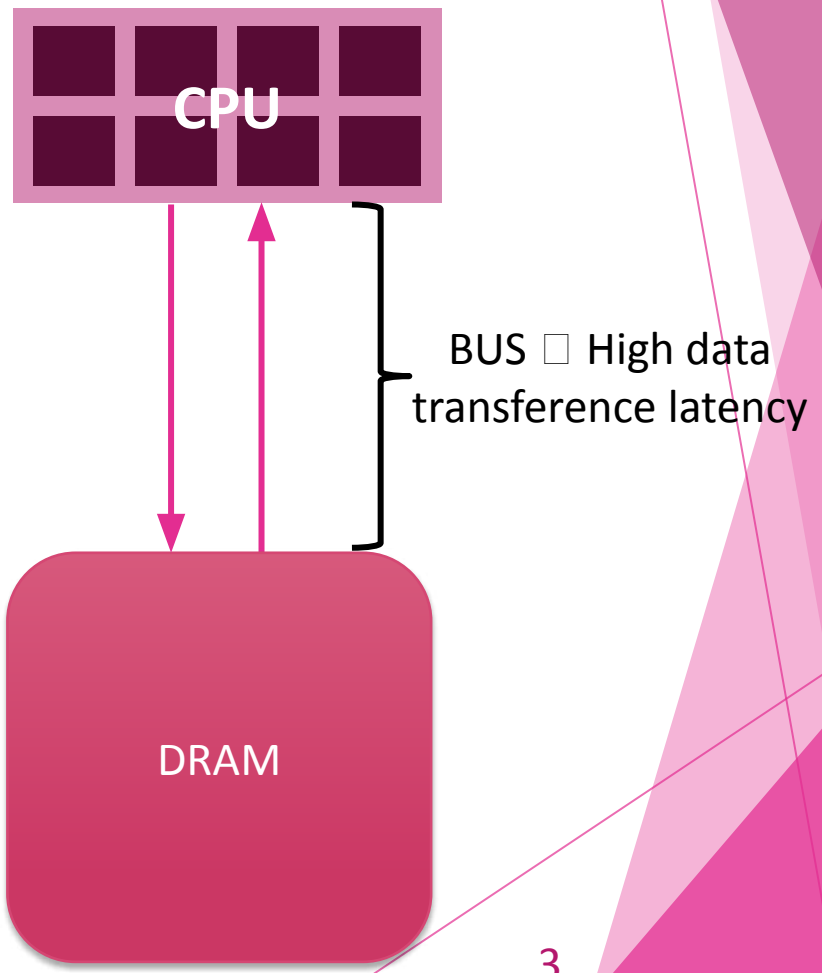
Gantz, John, and David Reinsel. "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east." *IDC iView: IDC Analyze the future* 2007.2012 (2012): 1-16.
Samuel, Arthur L. "Some studies in machine learning using the game of checkers." *IBM Journal of research and development* 3.3 (1959): 210-229.

Memory-Wall

- Not always a x86 Central Process Unit (CPU) can handle this
  - Low computational performance
  - High energy consumption

Von Neumann's bottleneck

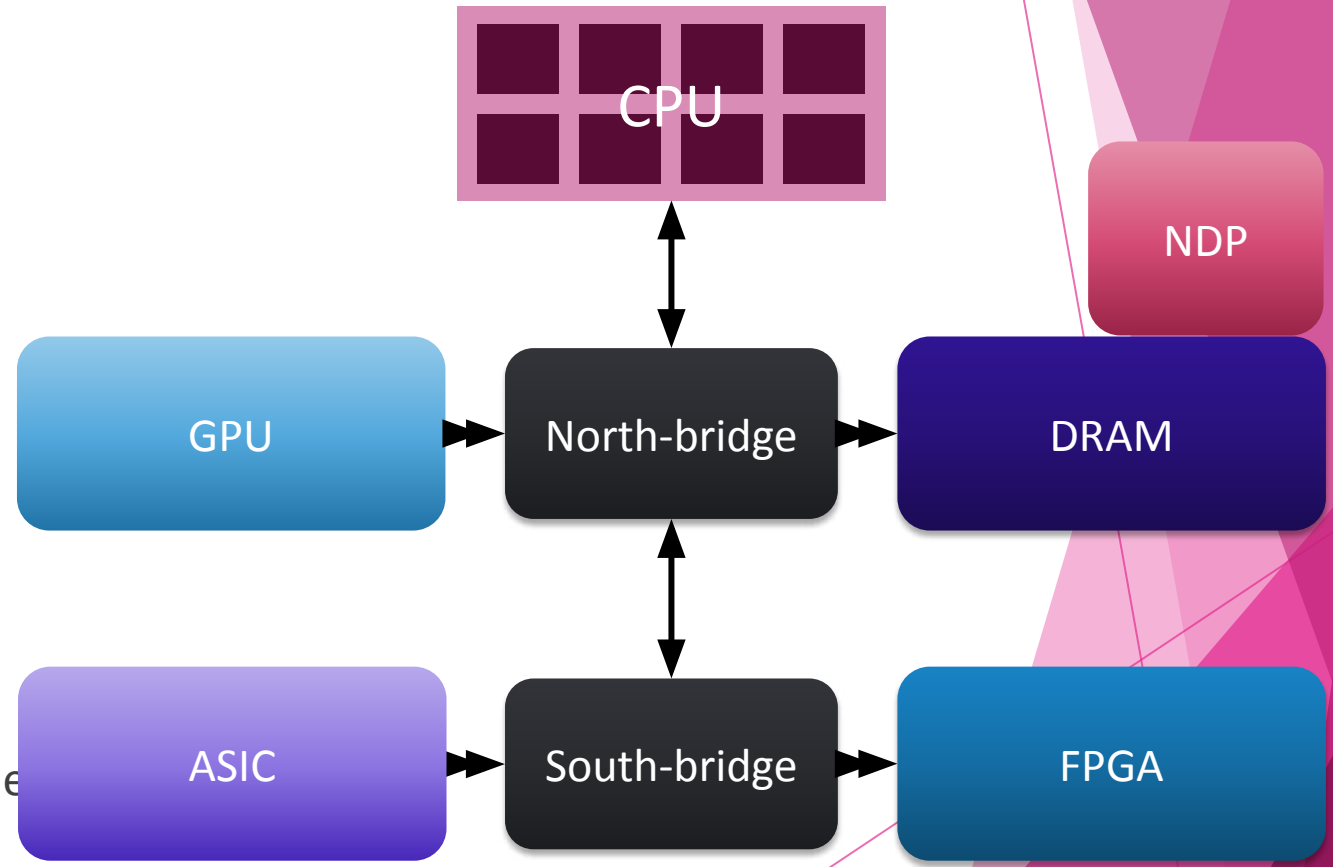
62,7% of the total system energy is spent on data movement.



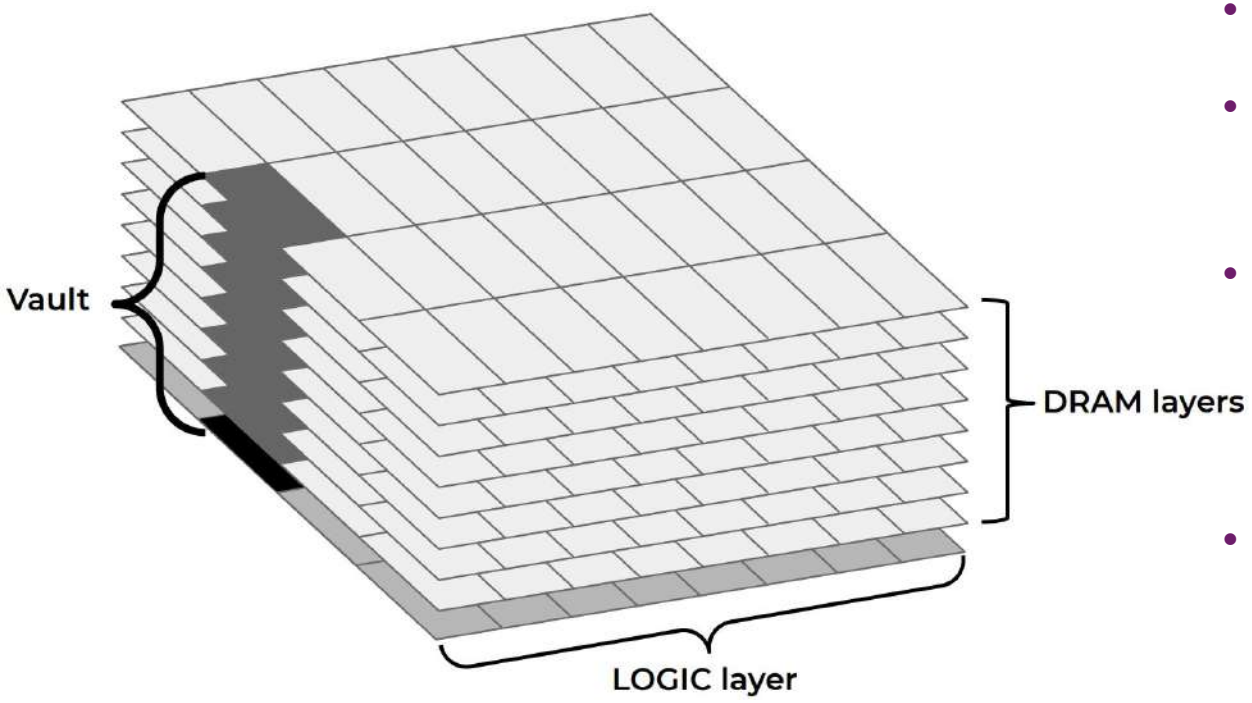
Boroumand, Amirali, et al. "Google workloads for consumer devices: Mitigating data movement bottlenecks." *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 2018.

Accelerators

- Graphic Processing Unit (GPU)
  - rely on the bus
  - High energy consumption
- Application Specific Integrated Circuit (ASIC)
  - Expensive
- Field Programmable Gate Array (FPGA)
  - Reprogrammable circuit
- Near Data Processing (NDP)
  - Processing and memory in the same chip



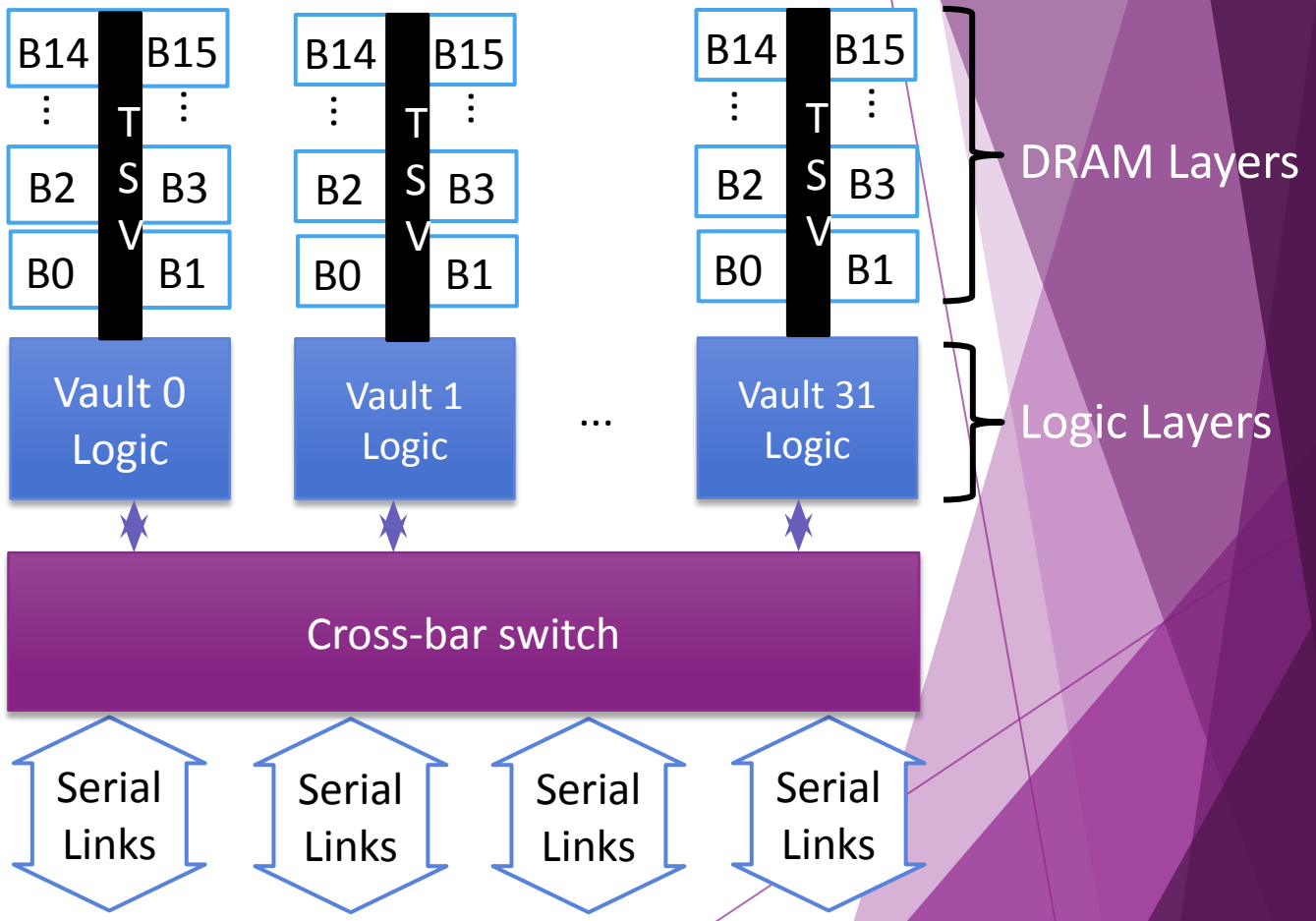
Near-Data Processing



- Emerged in the last few years
- Integration of processing unit and memory device in the same chip
- Compared to CPU:
  - reduce execution time
  - consume low energy
- Up to 8 stacked DRAM layers on a logic layer
  - Atomic and simple functions

Near-Data Processing

- Divided in 32 independent vaults
  - memory controller + up to 16 memory banks
- Logic layer and DRAM banks connected through Through-Silicon Vias (TSVs)
  - Parallel processing
- Hybrid Memory Cube (HMC) and High Bandwidth Memory (HBM2)

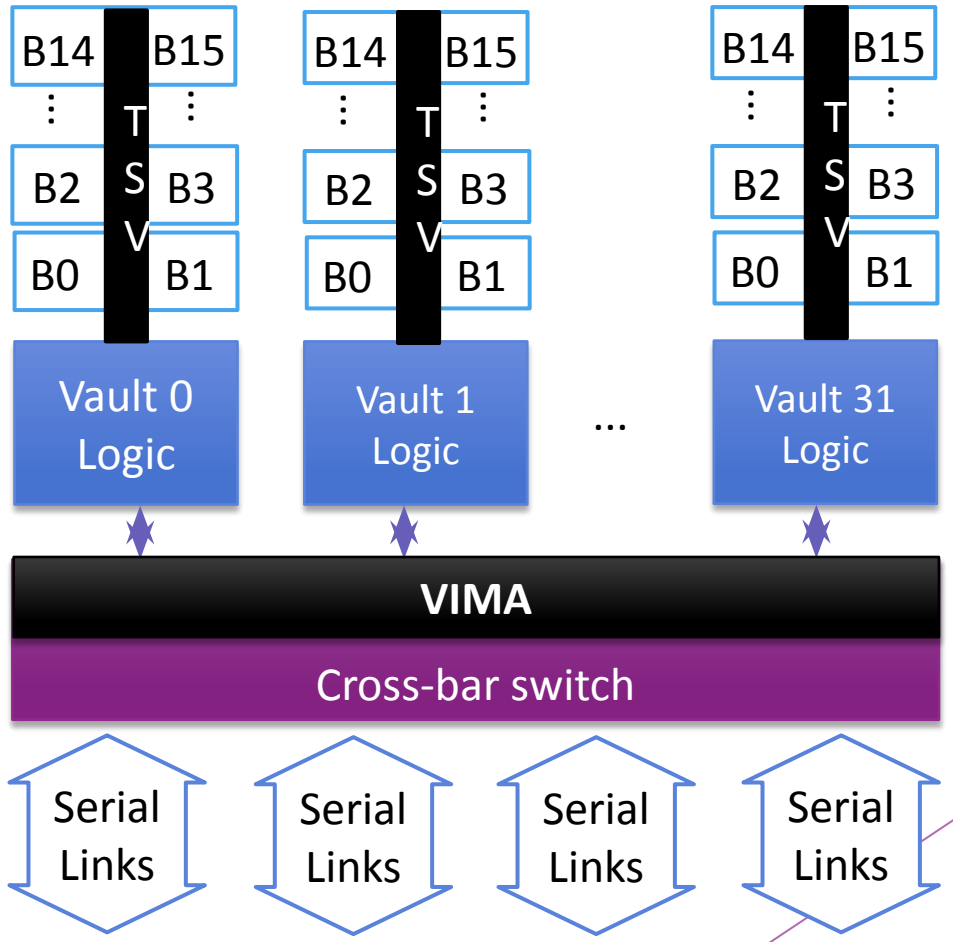


Pawlowski, J. Thomas. "Hybrid memory cube (HMC)." *2011 IEEE Hot chips 23 symposium (HCS)*. IEEE, 2011.

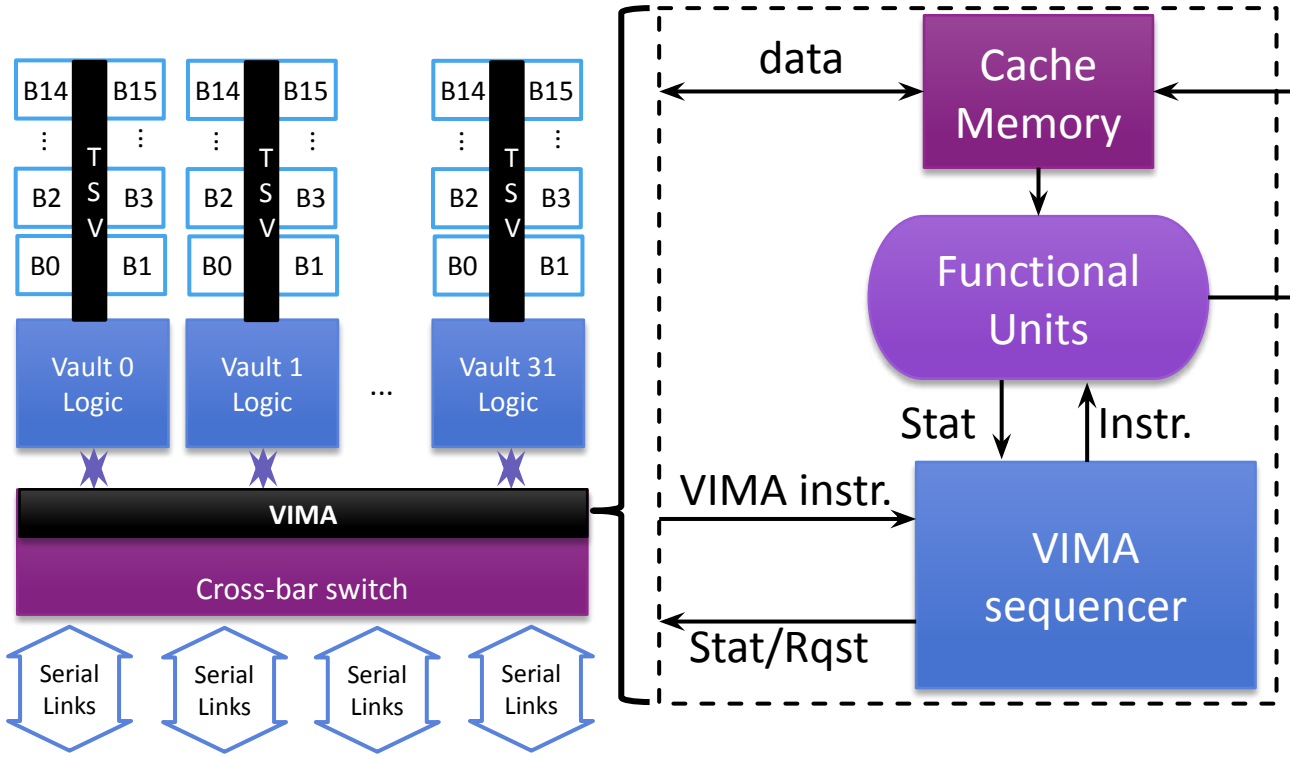
Jun, Hongshin, et al. "Hbm (high bandwidth memory) dram technology and architecture." *2017 IEEE International Memory Workshop (IMW)*. IEEE, 2017.

VIMA: Vector-in-Memory Architecture

- Allows vector operations of 256B and 8KB
- Focuses on algorithms that executes great amount of data while reuse data at certain point
- Combines Reduced Instruction Set Computer (RISC) with vector instructions – NEON Arm

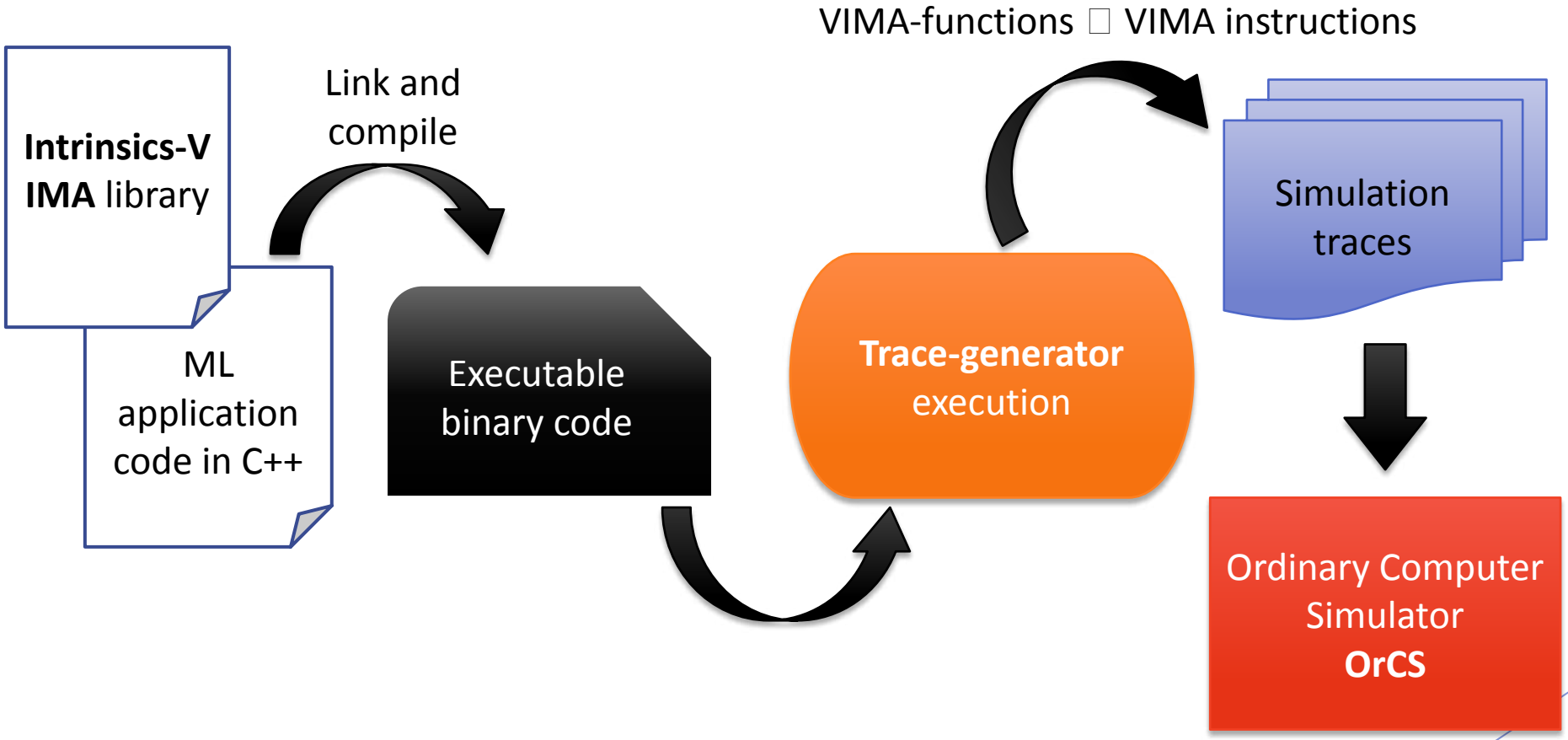


VIMA: Vector-in-Memory Architecture



- Compound of a sequencer, functional units and a cache memory:
  - Data reuse
  - On-chip cache
  - Parallel execution
  - OoO execution

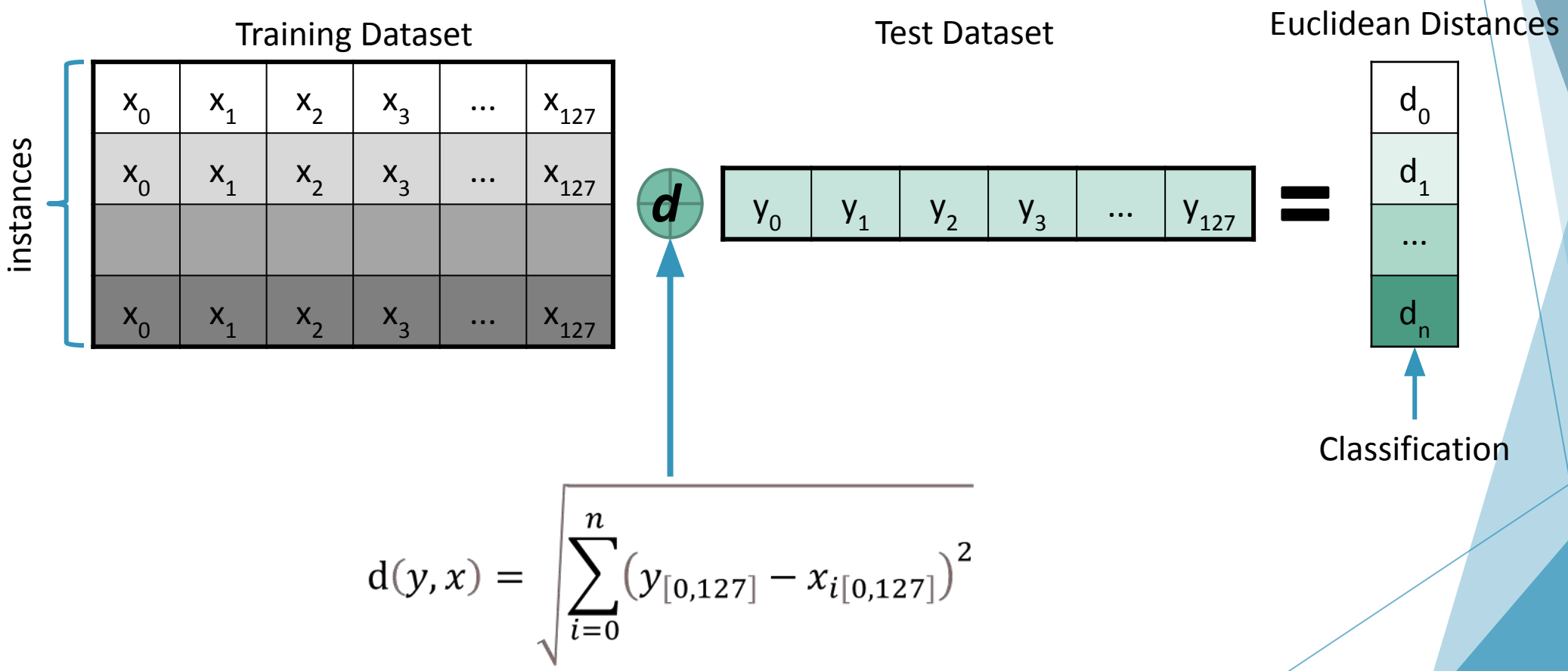
Development steps



Machine Learning Algoritms kernels

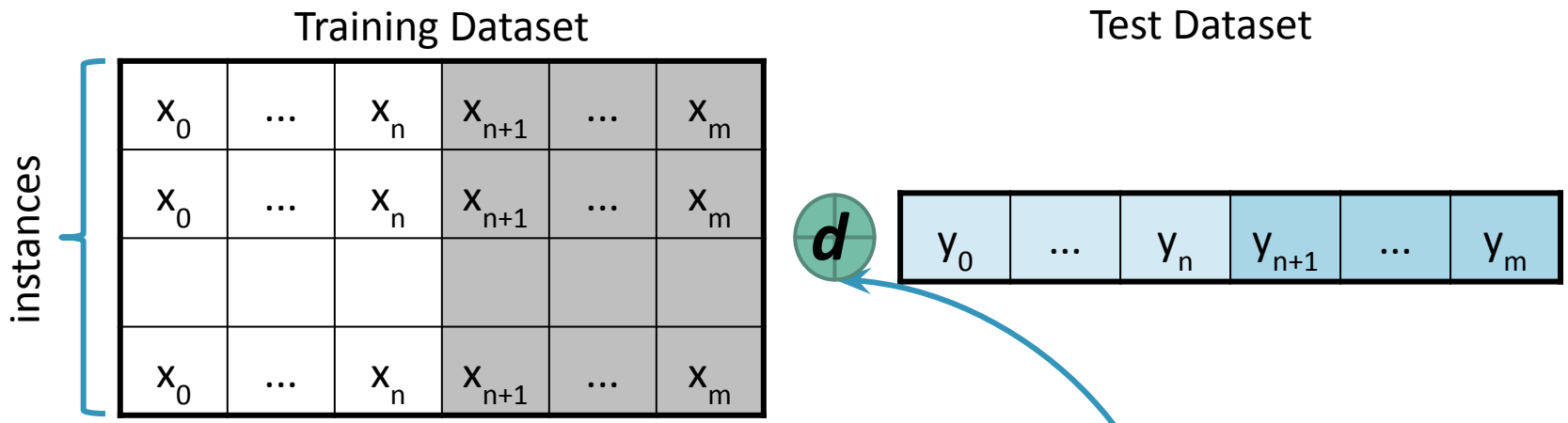
- **K-Nearest Neighbors:** classification by distance
- **MultiLayer Perceptron:** Neural Network
- **Convolution:** image transformation

k-Nearest Neighbors algorithm



$$\mathrm{d}(y,x) = \sqrt{\sum_{i=0}^{n} \left(y_{[0,127]} - x_{i[0,127]}\right)^2}$$

VIMA's implementation

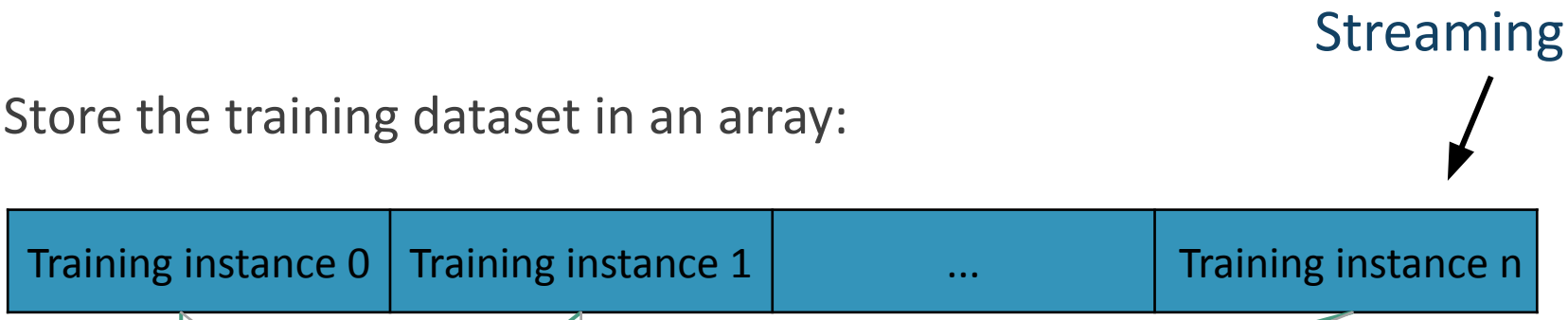


Vector operations between one test instance and the training dataset

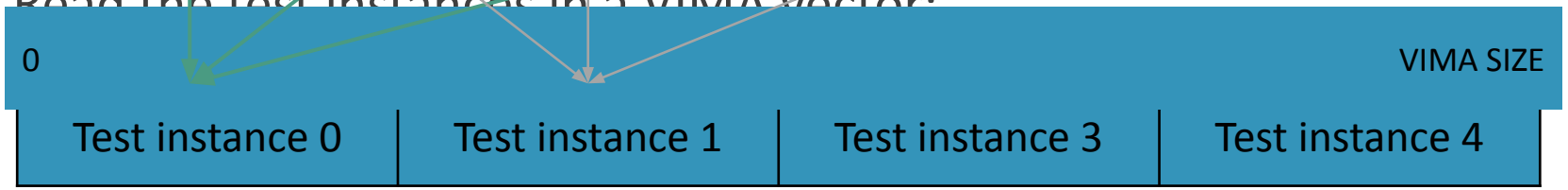
$$d(y,x) = \sqrt{\sum_{i=0}^{n} \left(y_{[0,n]} - x_{i[0,n]}\right)^2}$$

VIMA's implementation

- Store the training dataset in an array:



- Read the test instances in a VIMA vector:

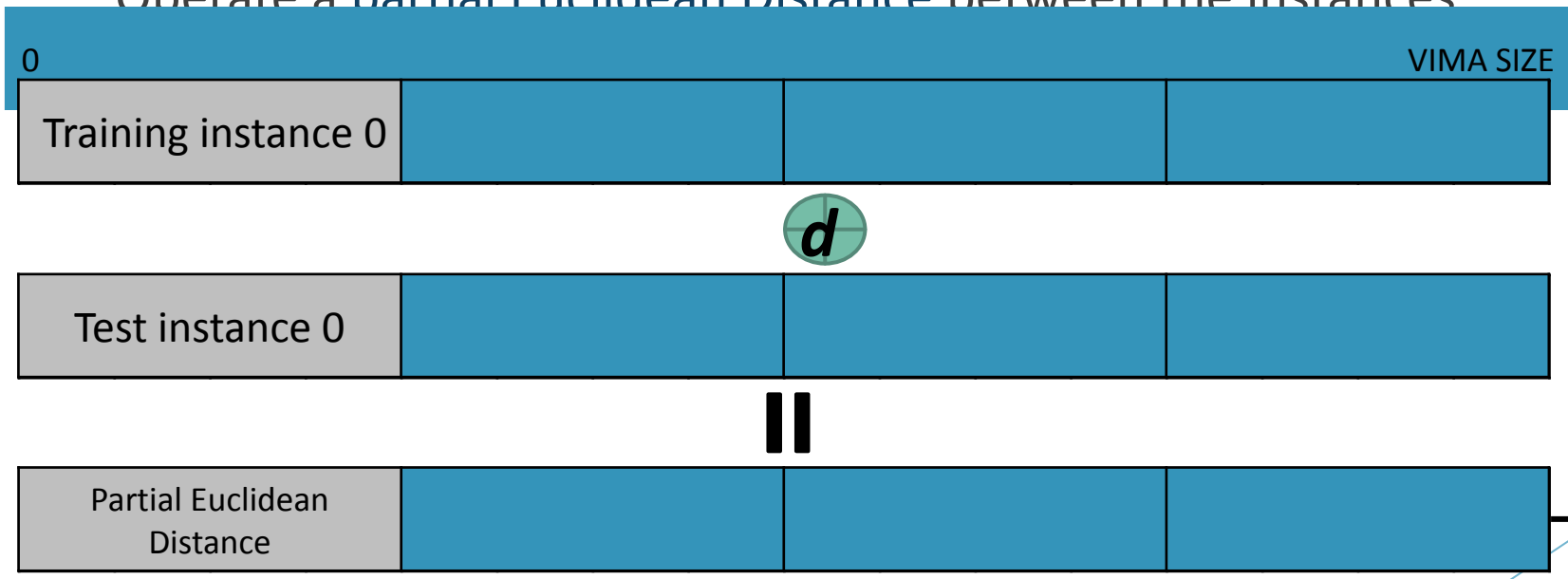


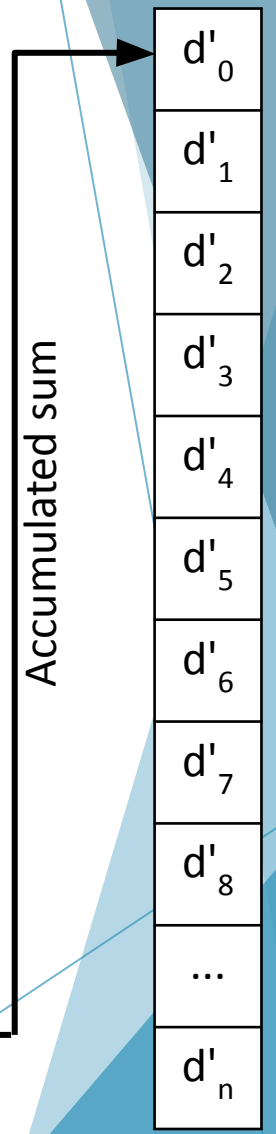
The training and test instances are operated in an inner loop

VIMA's implementation

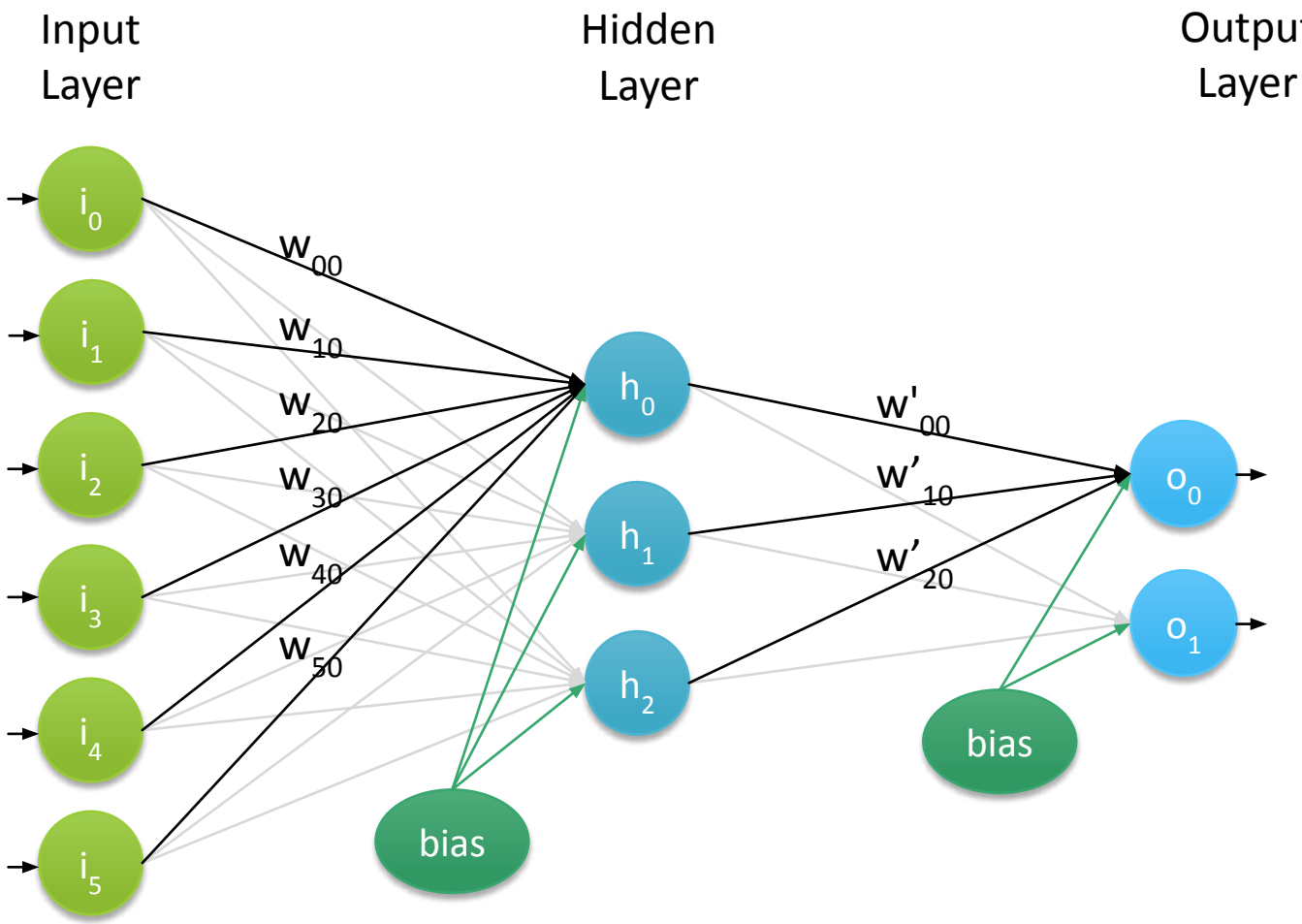
$$d'(y,x) = \sum_{i=0}^{n} \left(y_{[0,127]} - x_{i[0,127]}\right)^2$$

Operate a partial Euclidean Distance between the instances

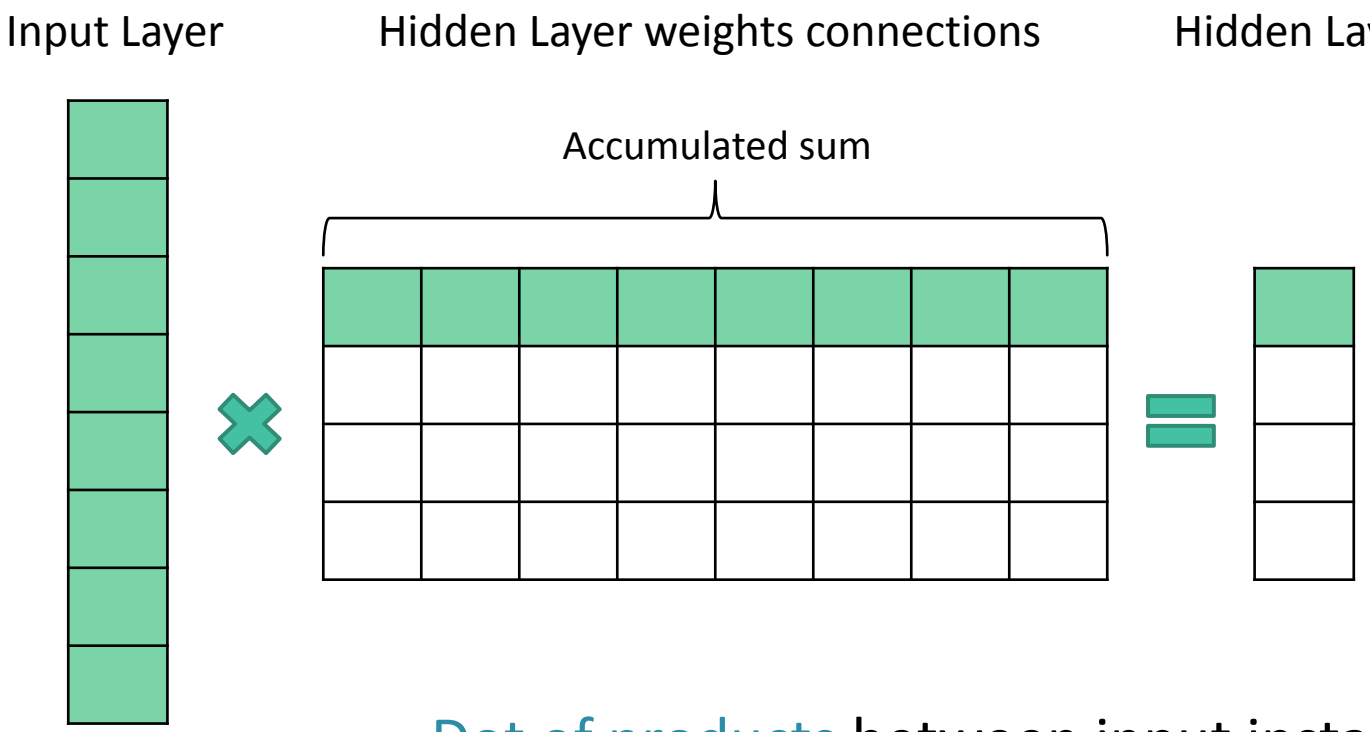




MultiLayer Perceptron algorithm

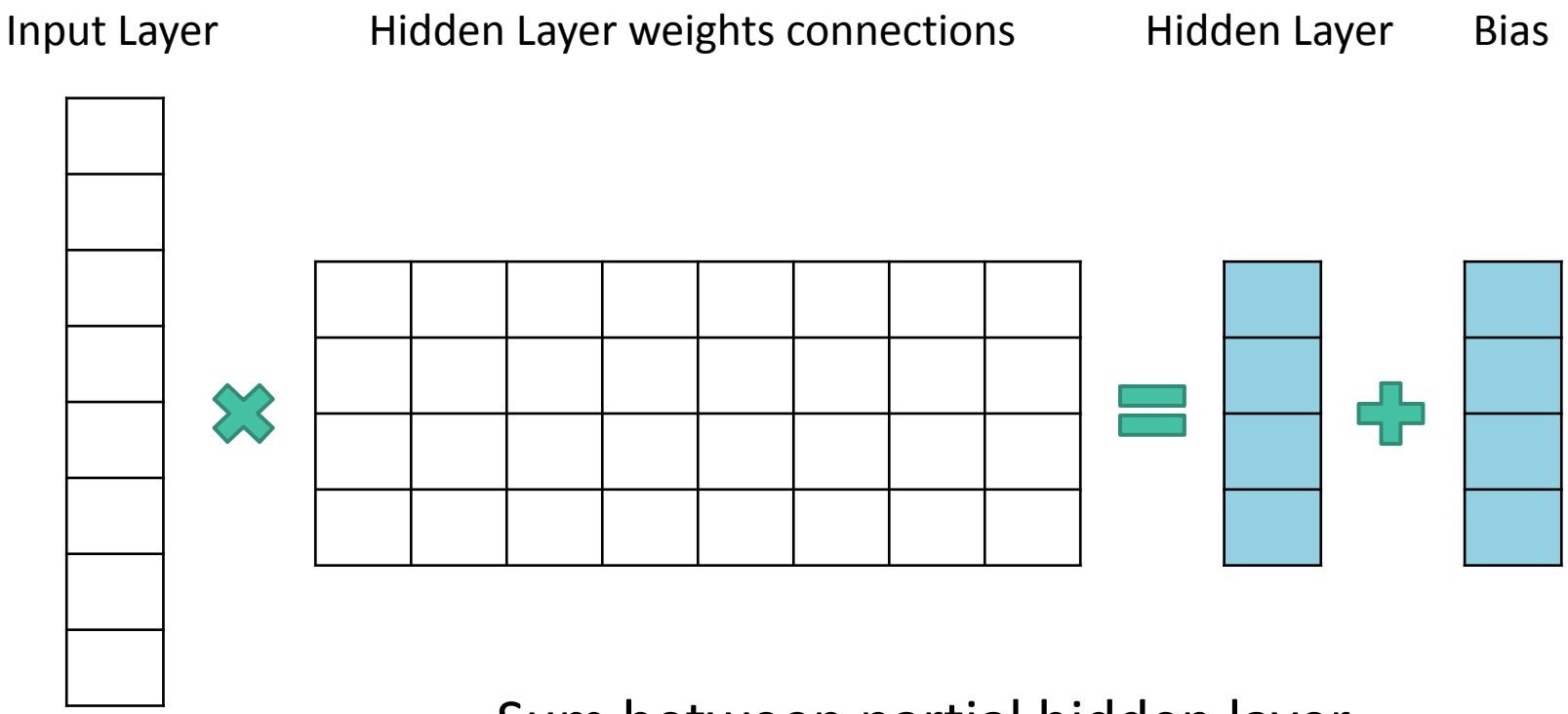


VIMA's implementation



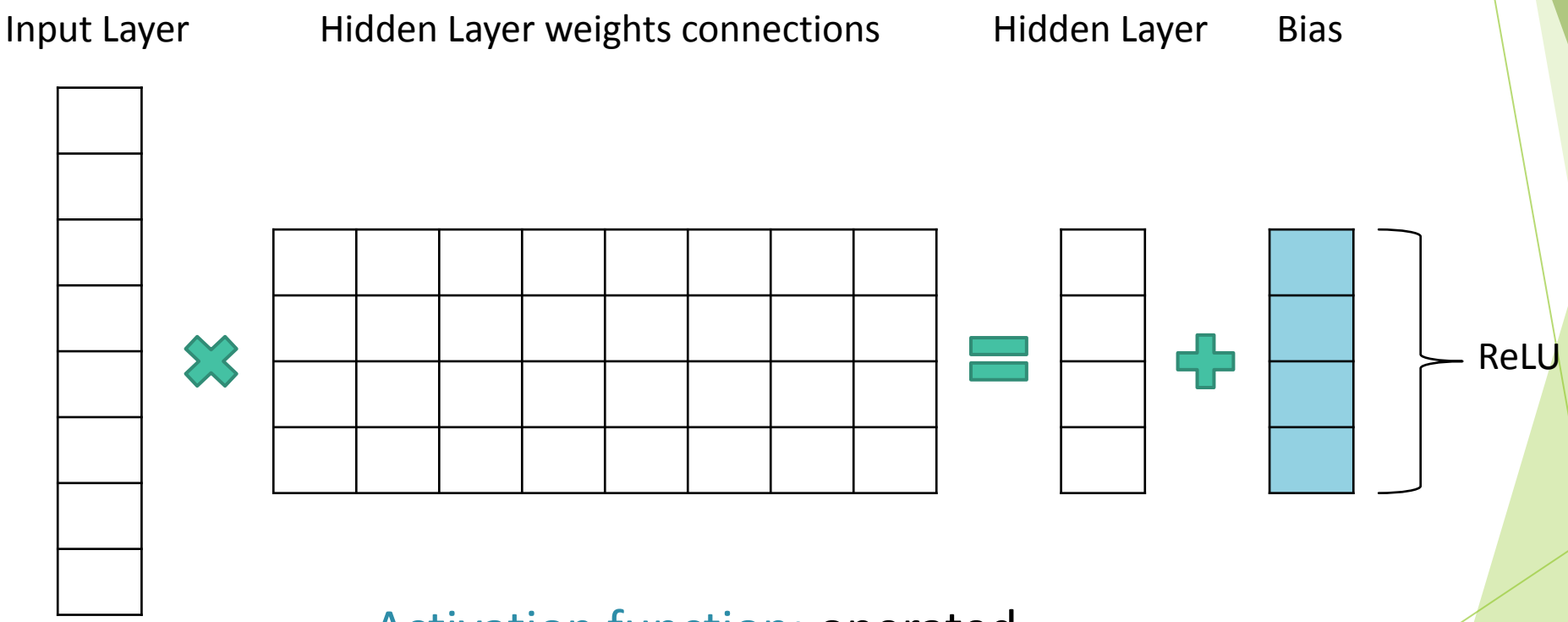
Dot of products between input instance and set of weights

VIMA's implementation



Sum between partial hidden layer activation values and bias

VIMA's implementation



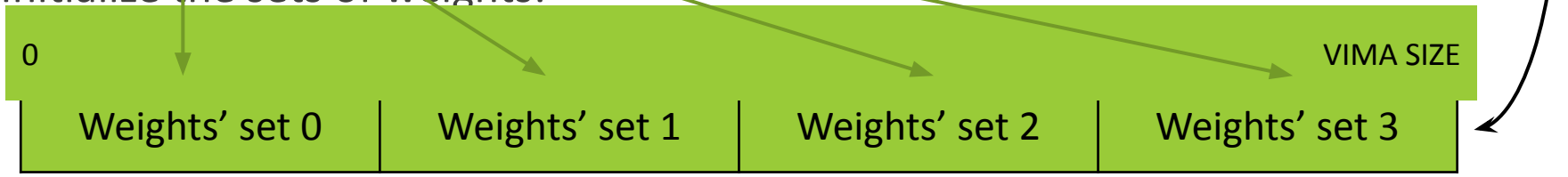
Activation function: operated with max vector function

VIMA's implementation: Inference Only

Streaming like
Just when there are a lot of features

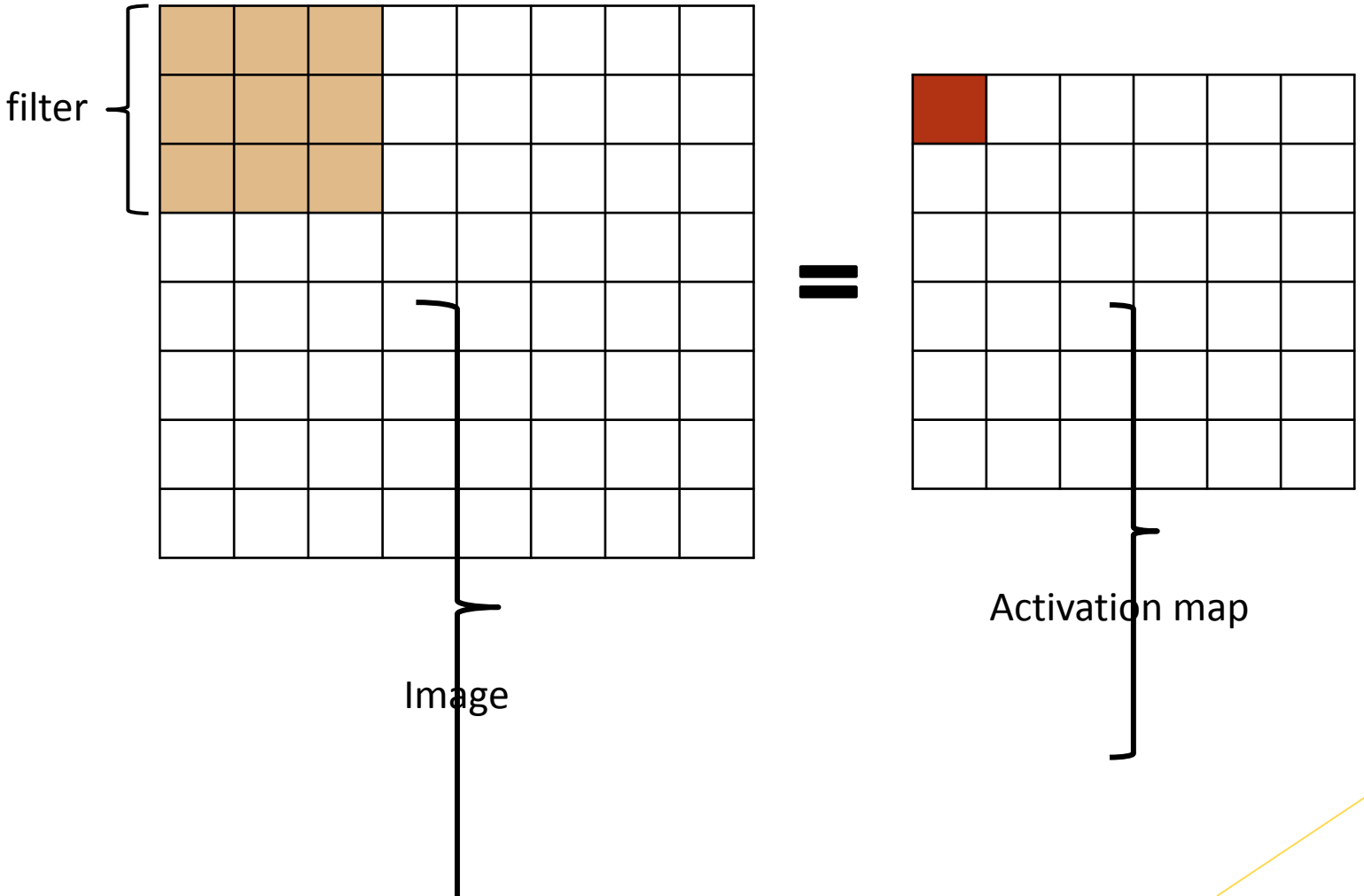
- Read the instances:

- Initialize the sets of weights:

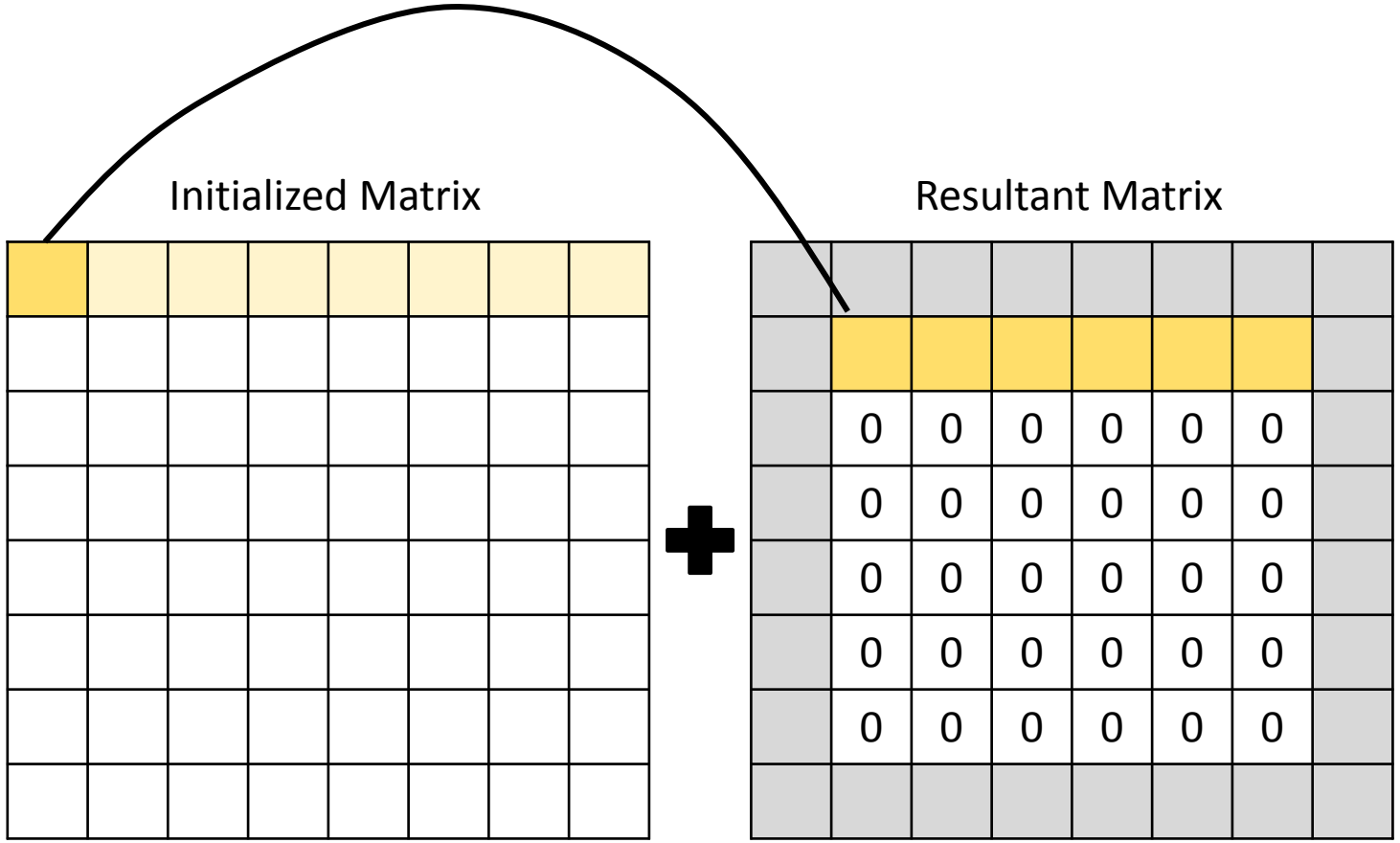


The instances and weights are operated in an inner loop

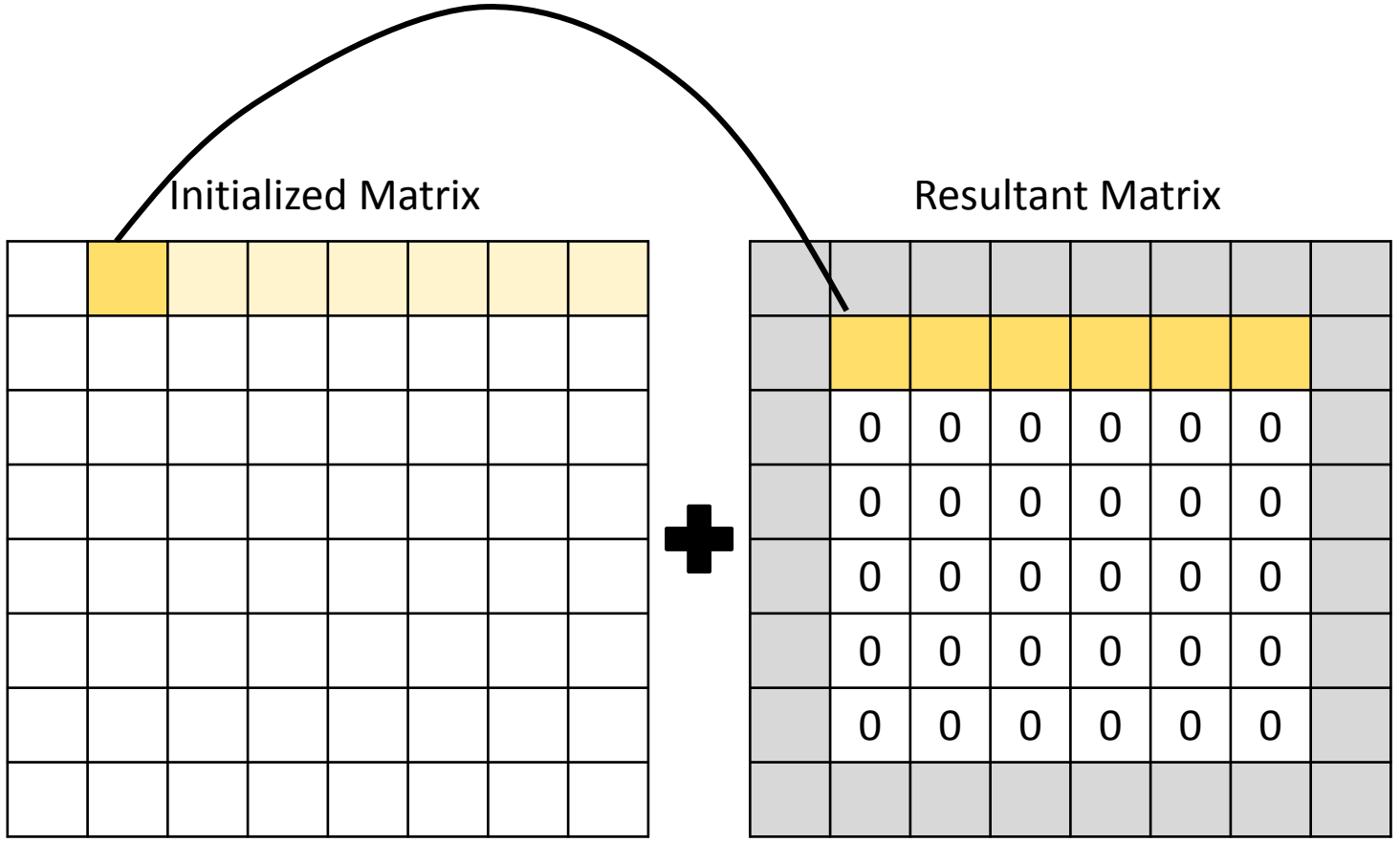
Convolution algorithm



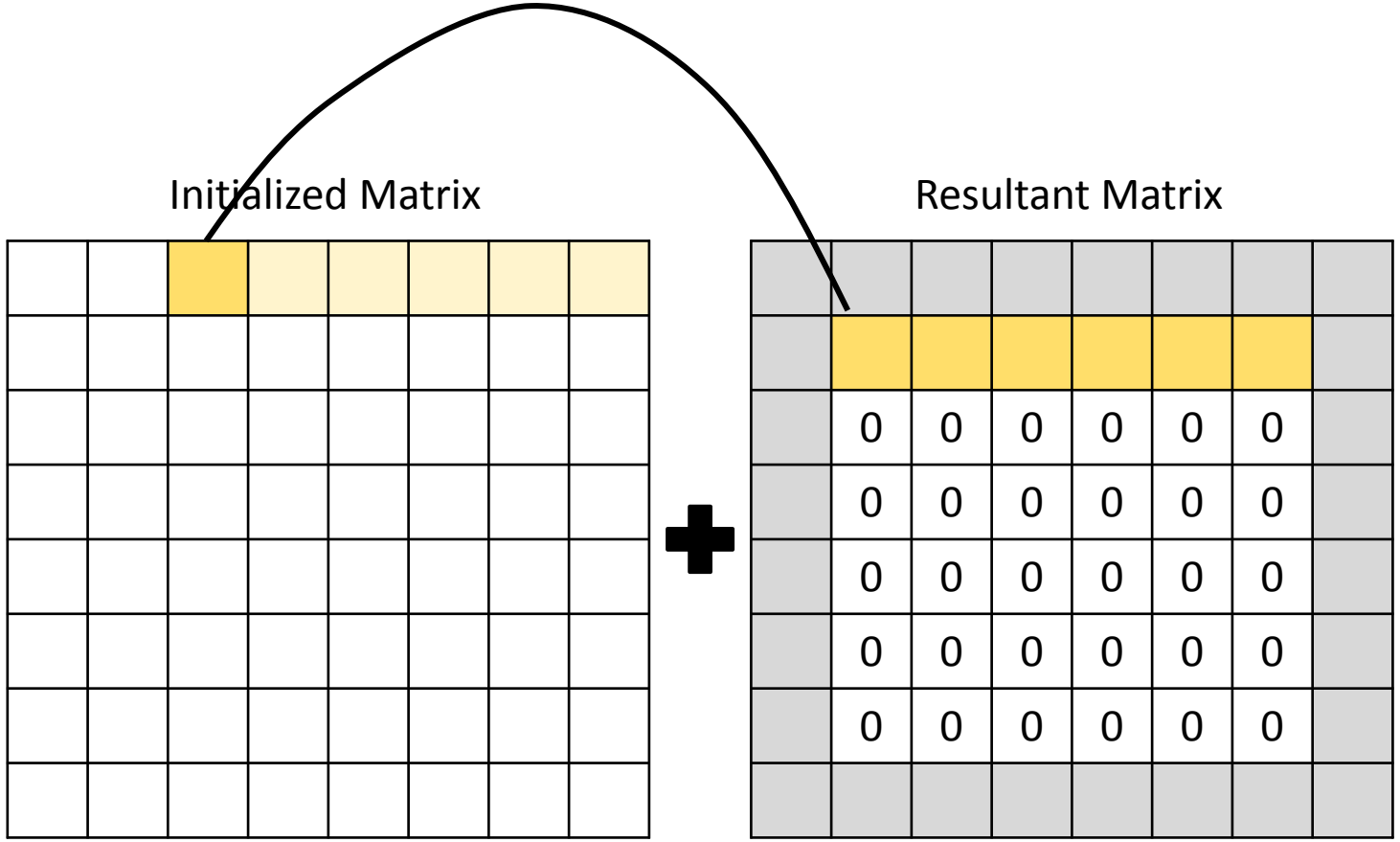
VIMA's implementation



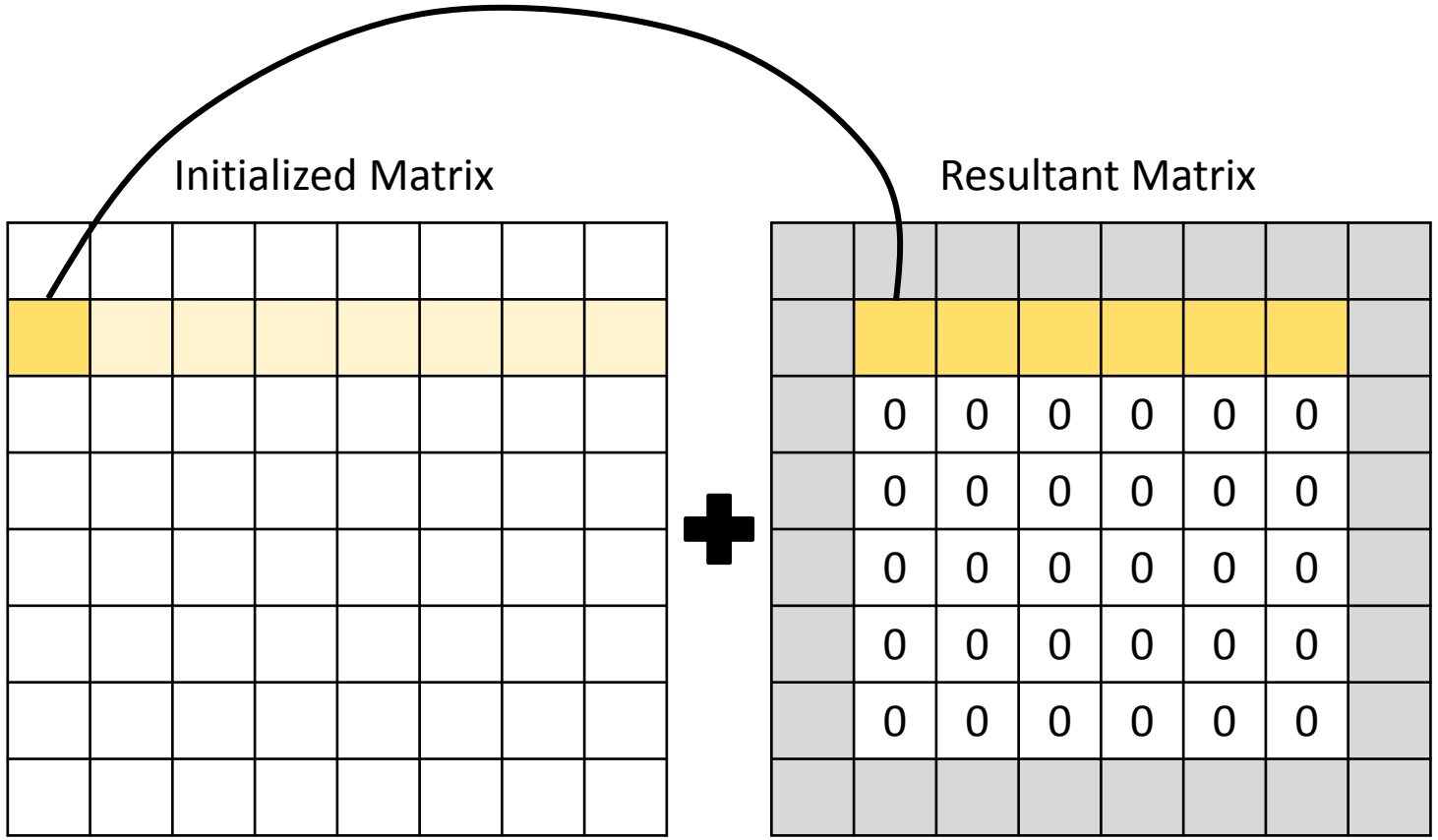
VIMA's implementation



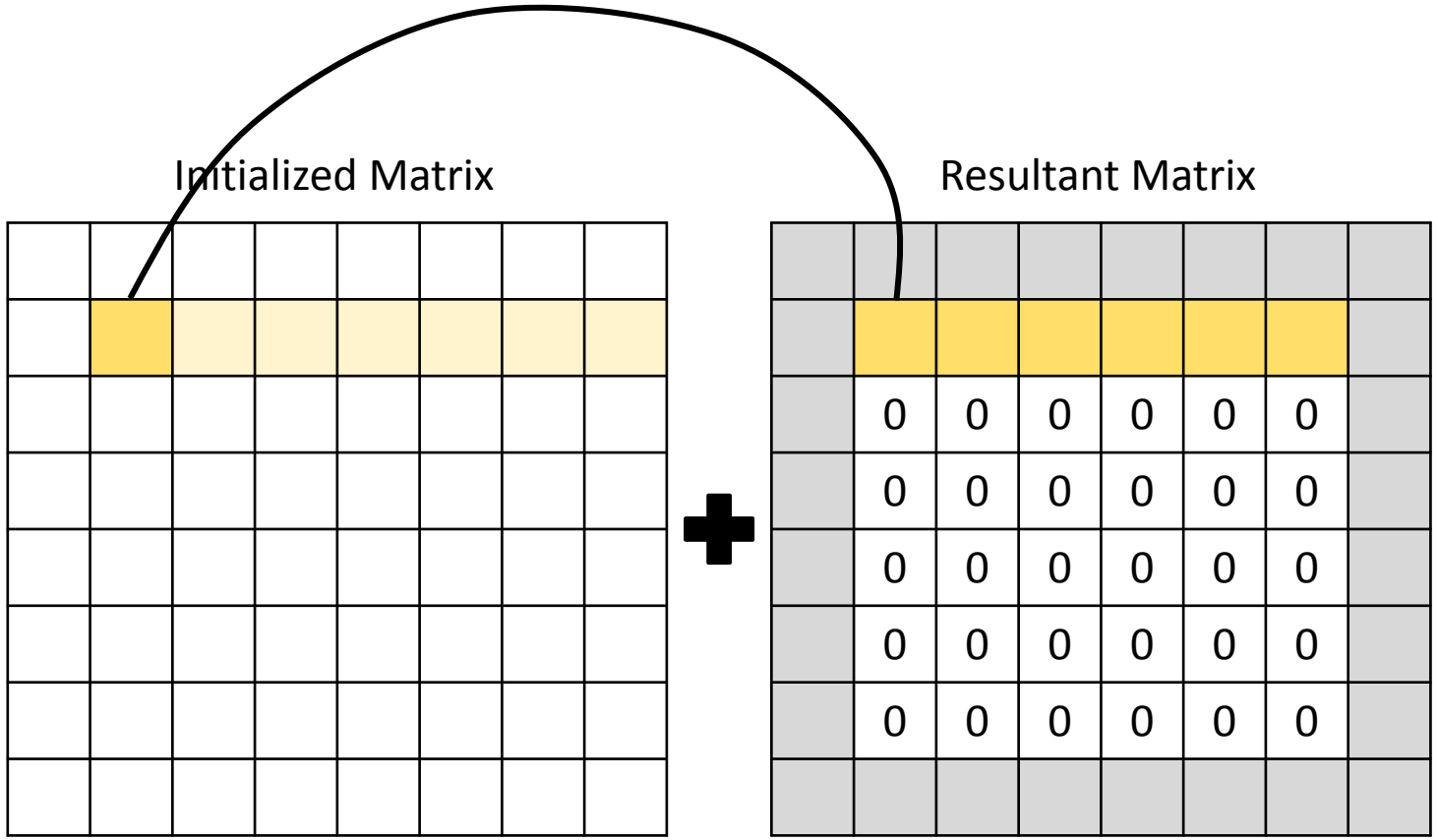
VIMA's implementation



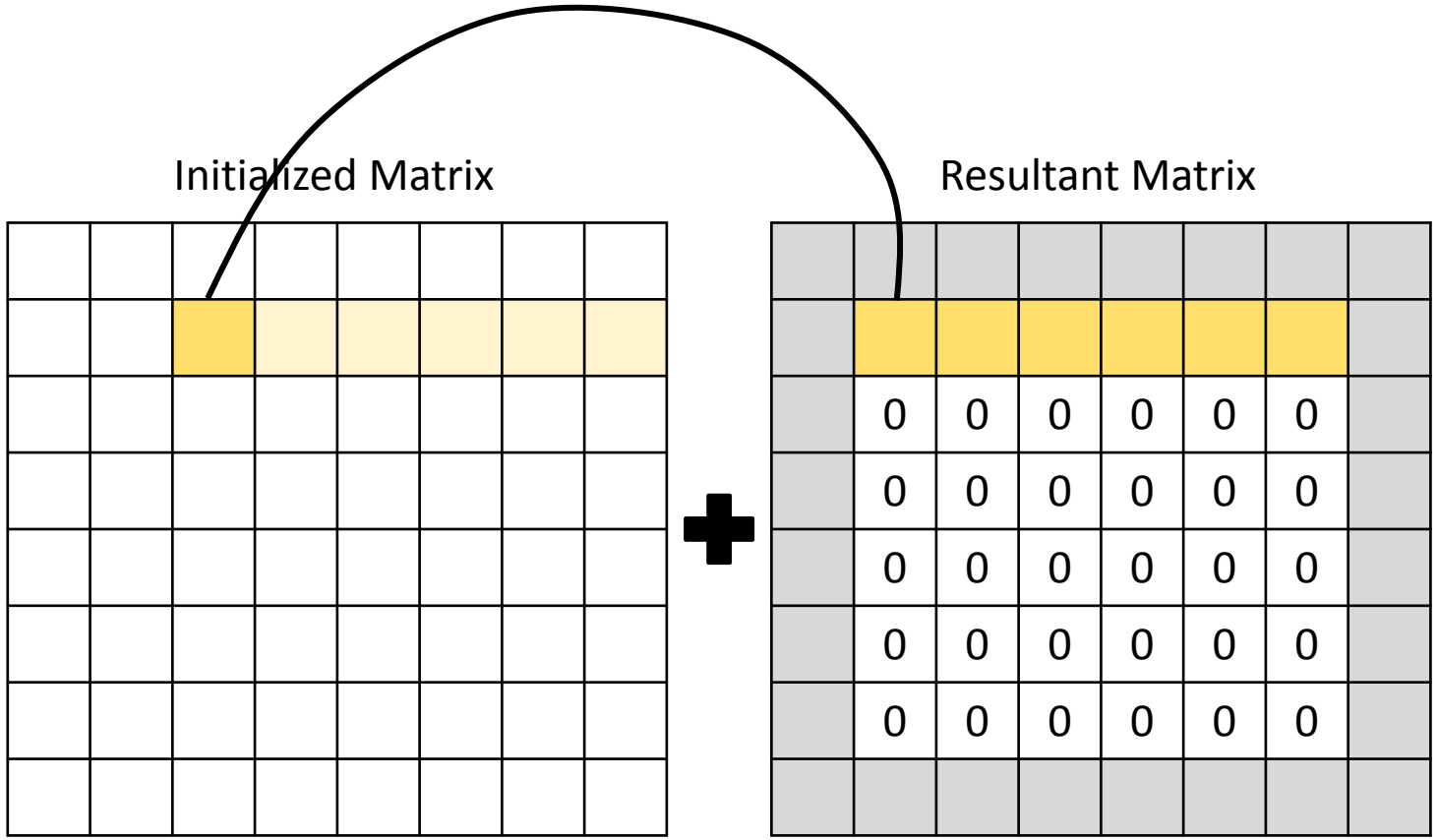
VIMA's implementation



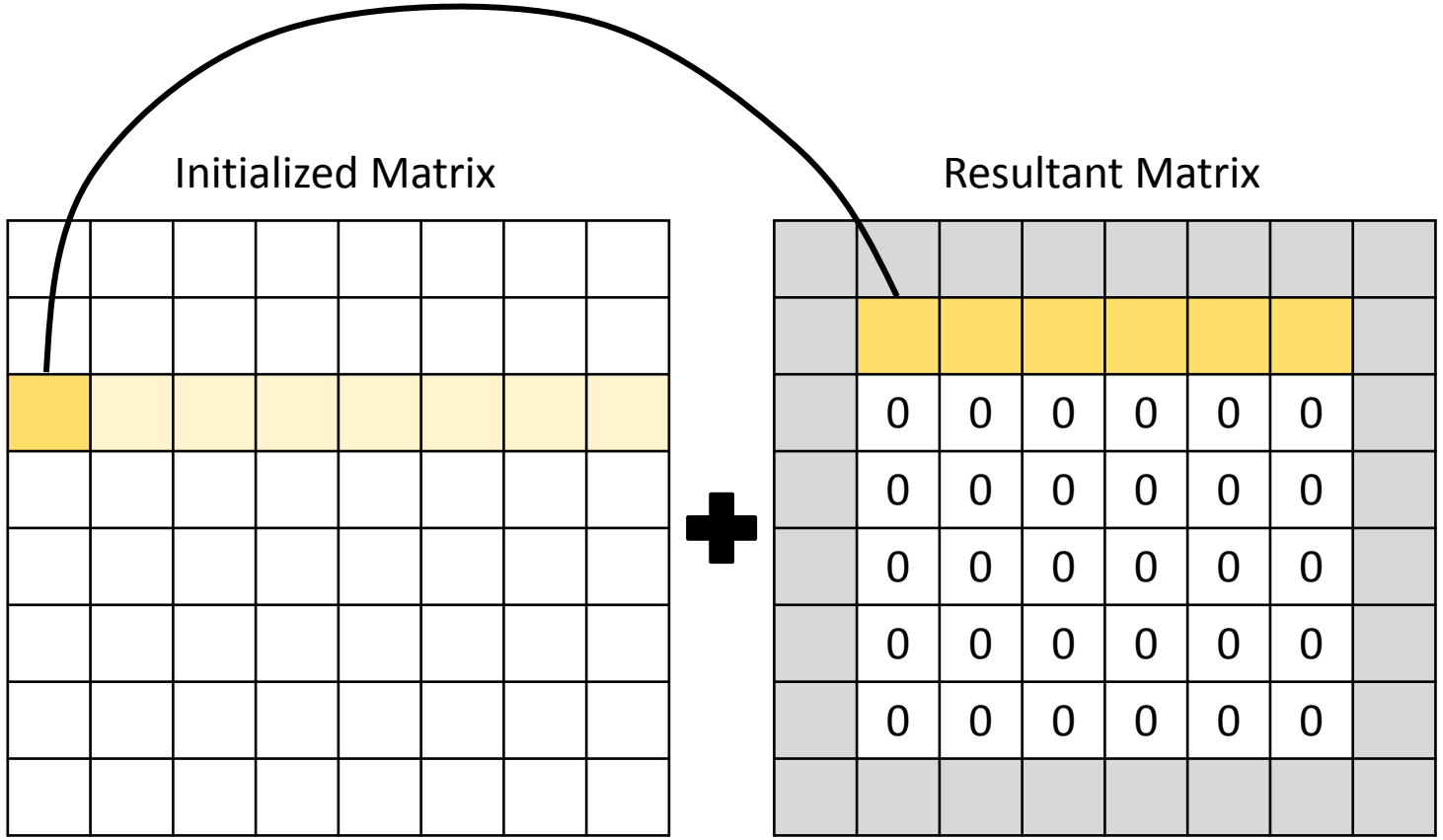
VIMA's implementation



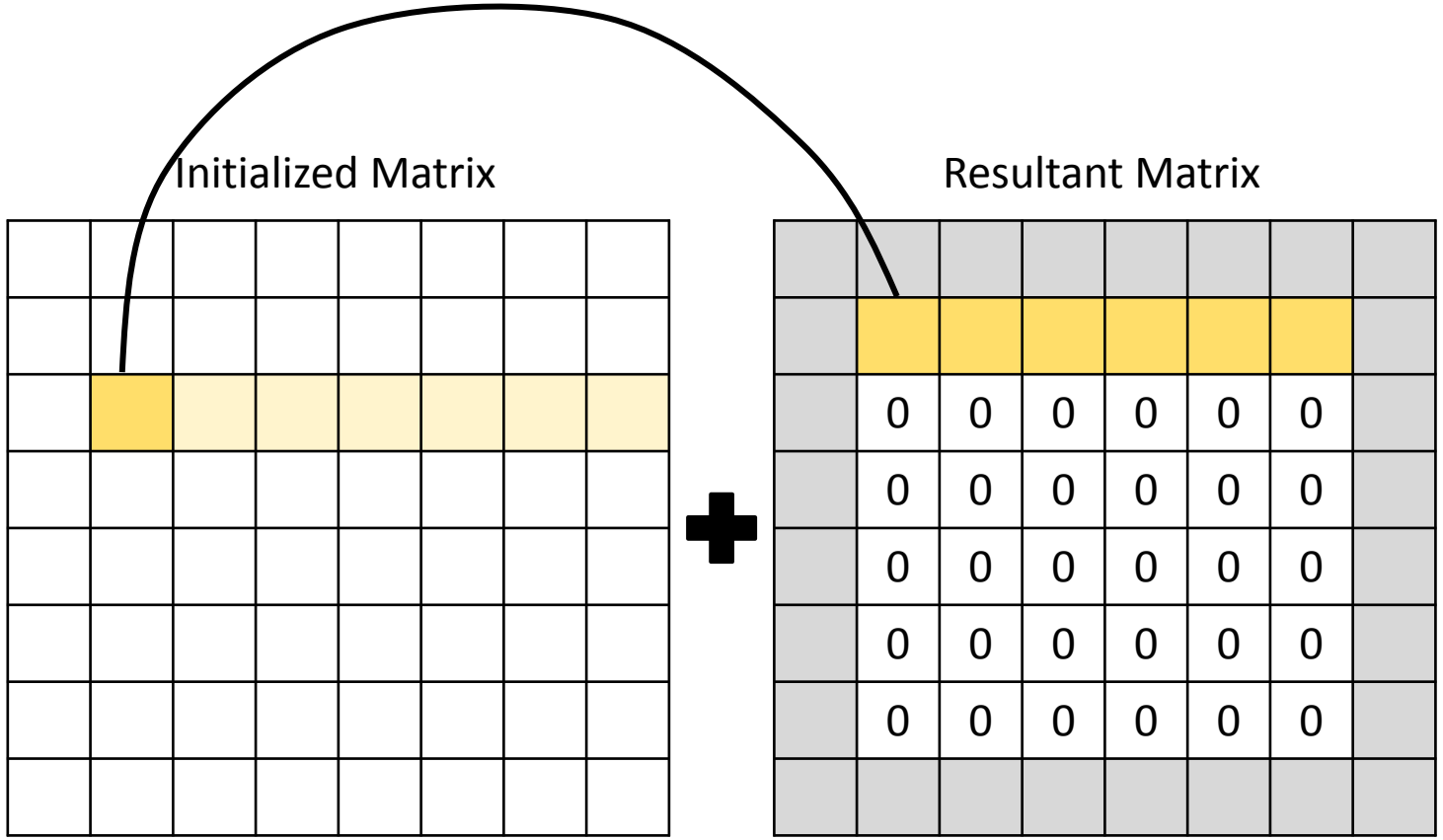
VIMA's implementation



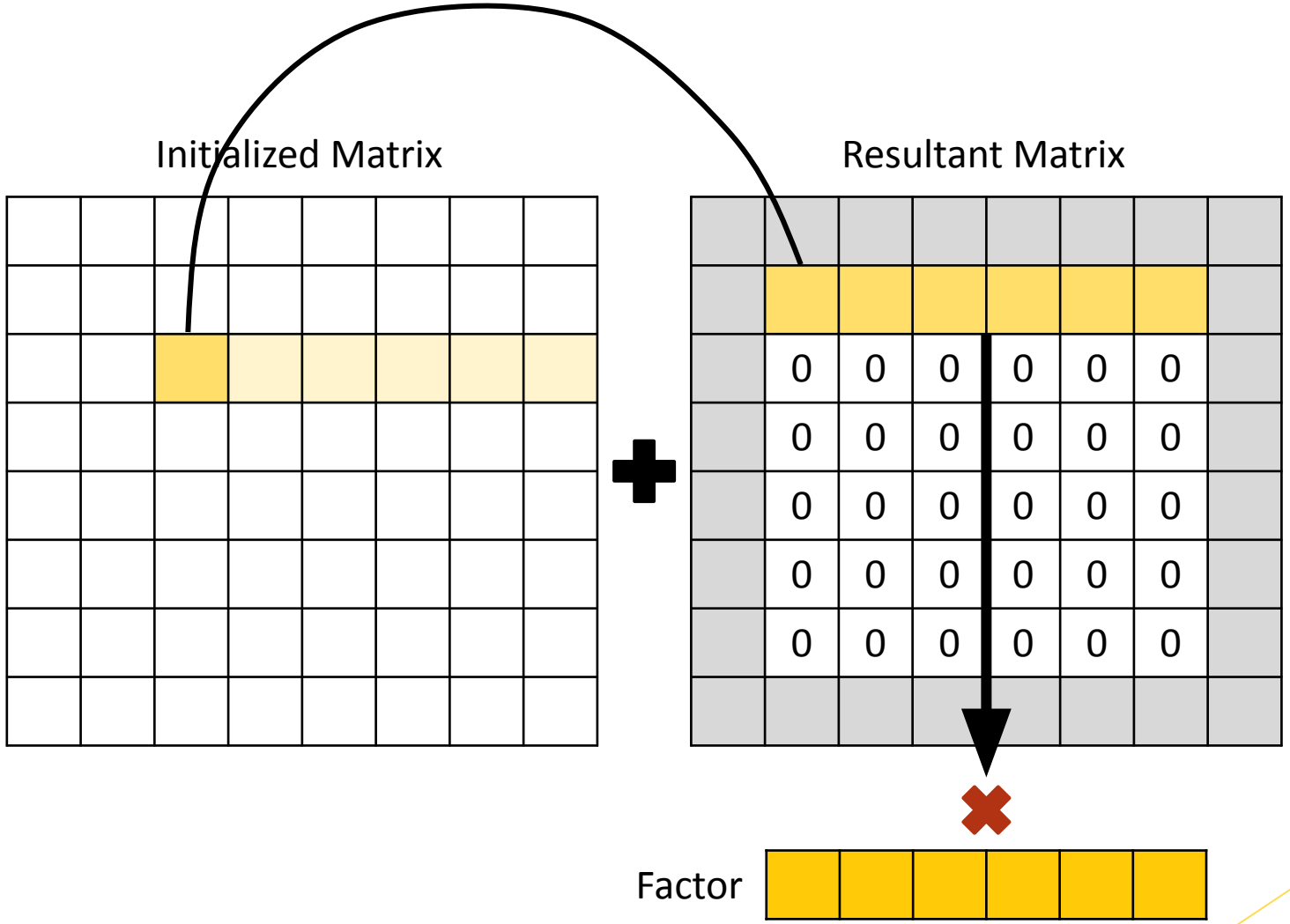
VIMA's implementation



VIMA's implementation



VIMA's implementation



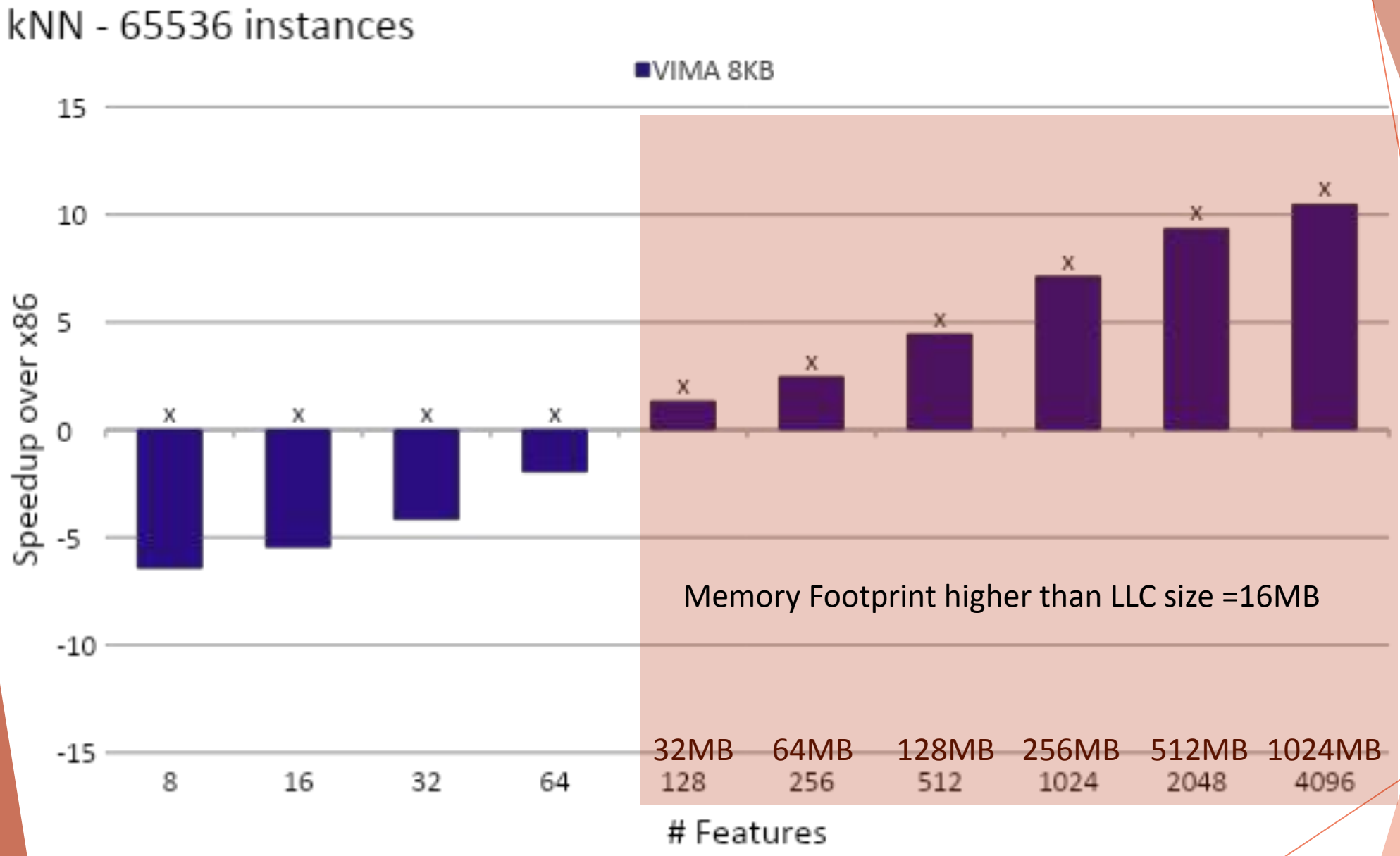
Benchmarks: 133KB □ 512MB

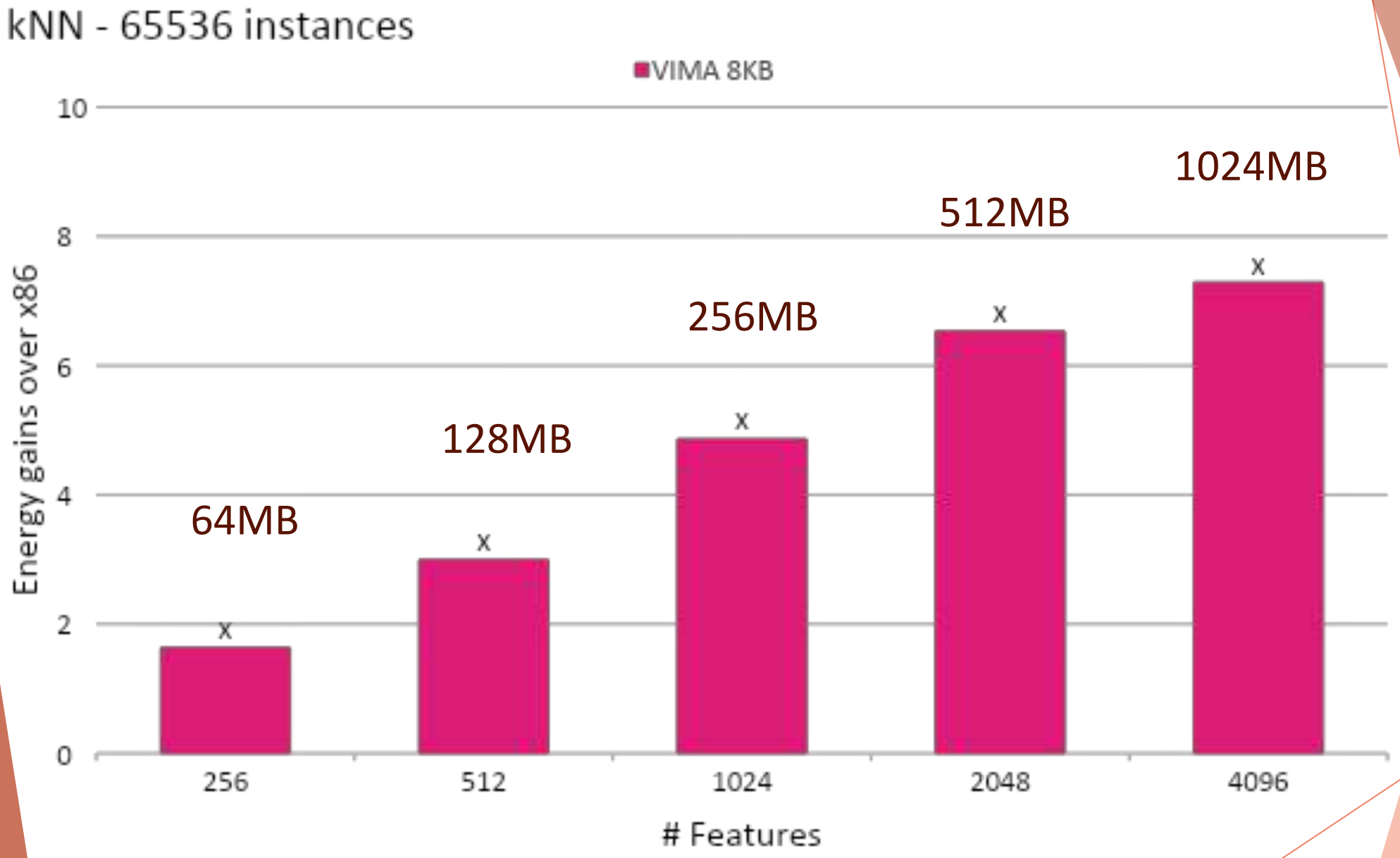
- kNN
  - **# training instances:** 4096, 8192, 16384, 32768, and 65536
  - **# test instances:** 256
  - **# features:** 8, 16, 32, 64, 128, 256, 512, 1024, 2048, and 4096
  - **# neighbors:** 9
- MLP
  - **# instances:** 4096, 8192, 16384, 32768, and 65536
  - **# features:** 8, 16, 32, 64, 128, 256, 512, 1024, 2048, and 4096
- Convolution
  - **Matrix dimensions:** 512x512, 724x724, 1024x1024, 1448x1448, 2048x2048, 2896x2896, 4096x4096, 5792x5792, 8192x8192, and 11648x11648

System configuration

- Intel Sandy Bridge processor 2.0GHz – x86 architecture
- L1 Cache **32KB**
- L2 Cache **256KB**
- LLC Cache **16MB**
- 3D-Stacked Memory as main memory
- VIMA module

VIMA is expected to achieve better performance when the application memory footprint exceeds the LLC size


kNN - 65536 instances
VIMA 8KB
Speedup over x86
15
10
5
0
-5
-10
-15
8
16
32
64
128
256
512
1024
2048
4096
32MB
64MB
128MB
256MB
512MB
1024MB
Features
Memory Footprint higher than LLC size =16MB

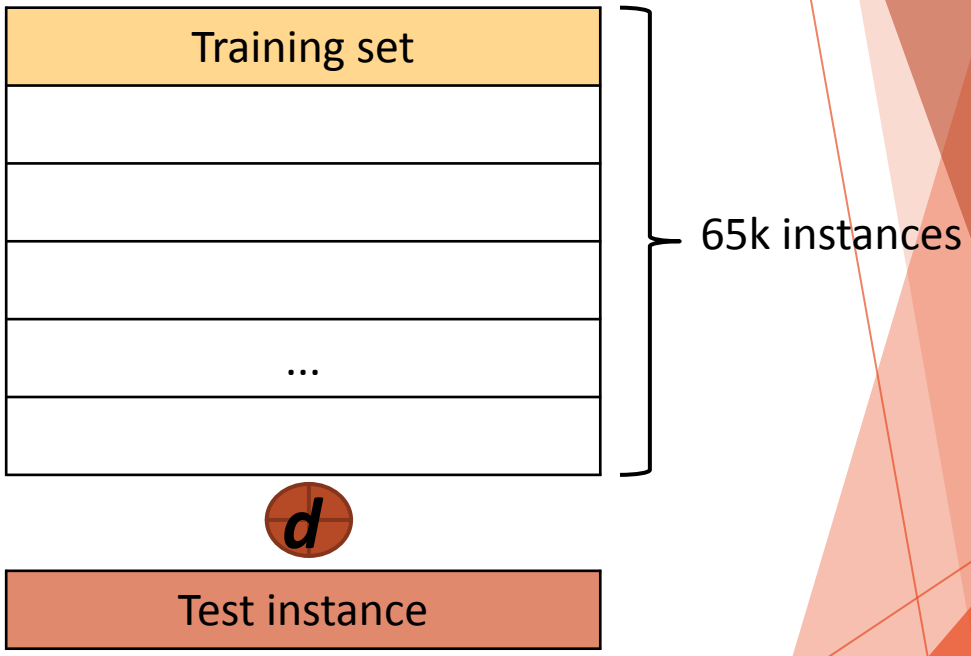


Memory Footprint higher than LLC size = 16MB

Energy from: Host processor + memory hierarchy + VIMA + VIMA's Cache

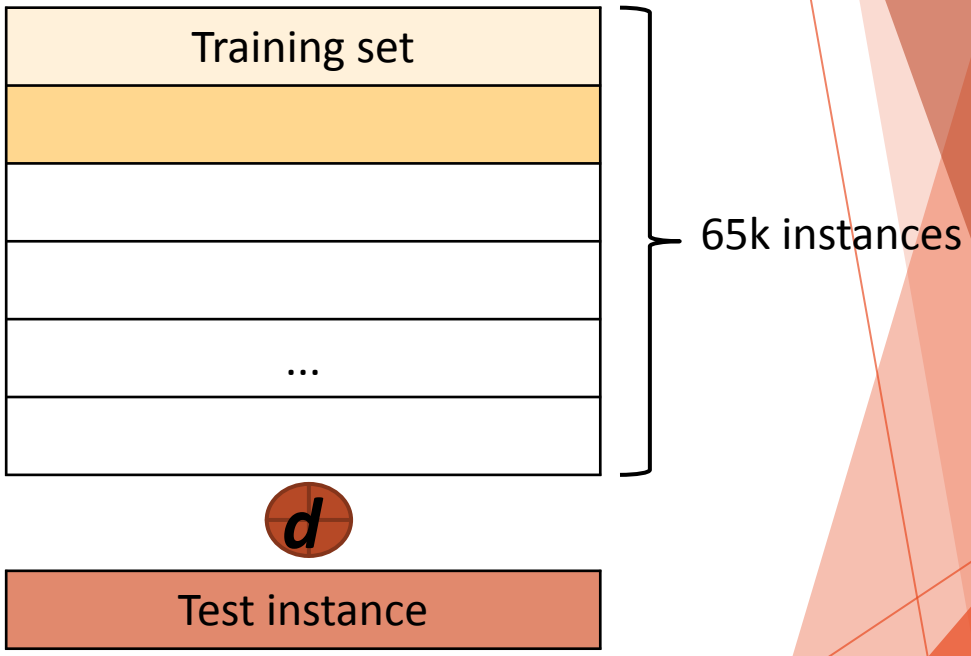
kNN conclusion

- The more instances and features, the higher the memory footprint
  - Quadratic complexity
  - Training dataset with 4096 instances and 1024 features already exceeds cache memory size



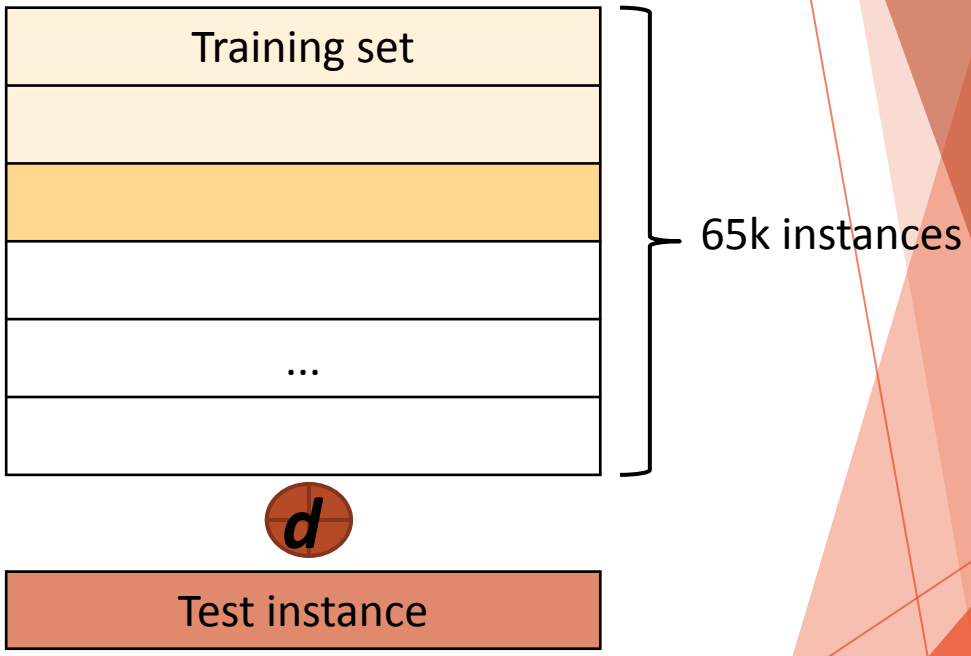
kNN conclusion

- The more instances and features, the higher the memory footprint
  - Quadratic complexity
  - Training dataset with 4096 instances and 1024 features already exceeds cache memory size



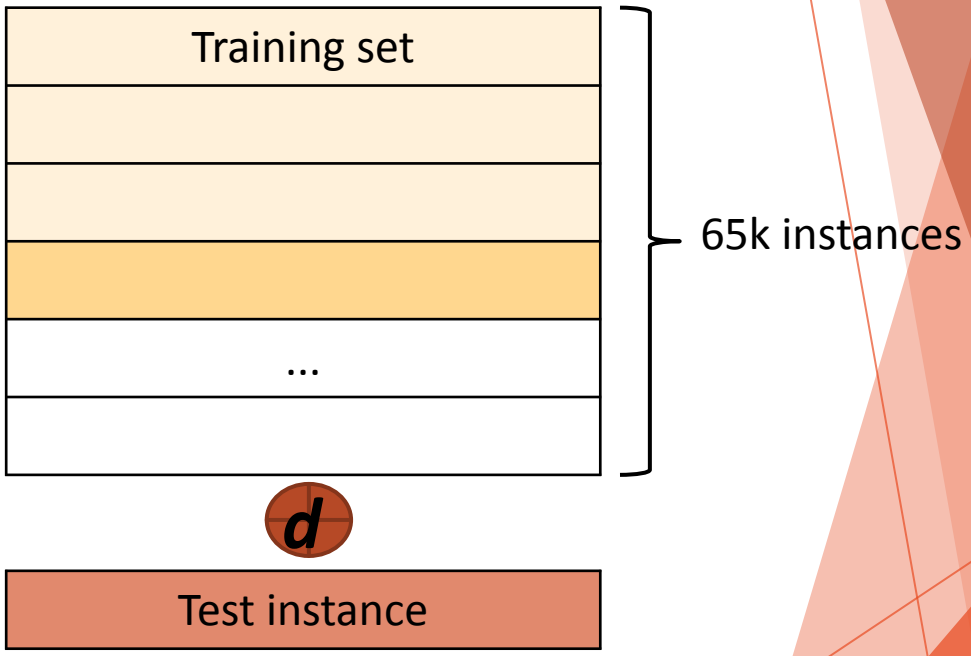
kNN conclusion

- The more instances and features, the higher the memory footprint
  - Quadratic complexity
  - Training dataset with 4096 instances and 1024 features already exceeds cache memory size



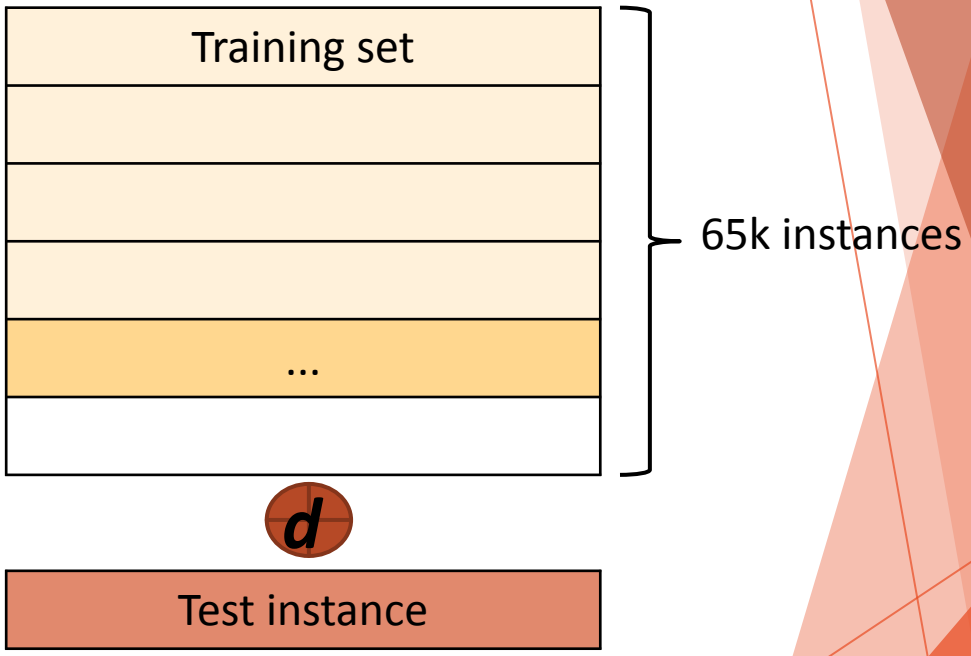
kNN conclusion

- The more instances and features, the higher the memory footprint
  - Quadratic complexity
  - Training dataset with 4096 instances and 1024 features already exceeds cache memory size



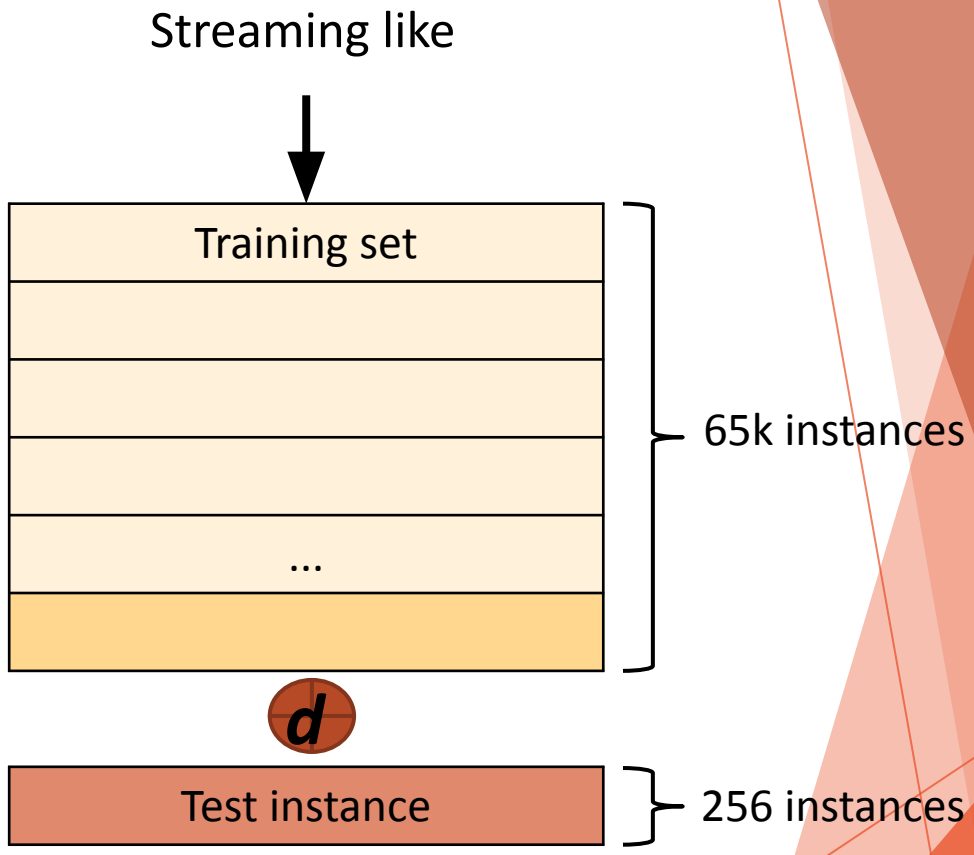
kNN conclusion

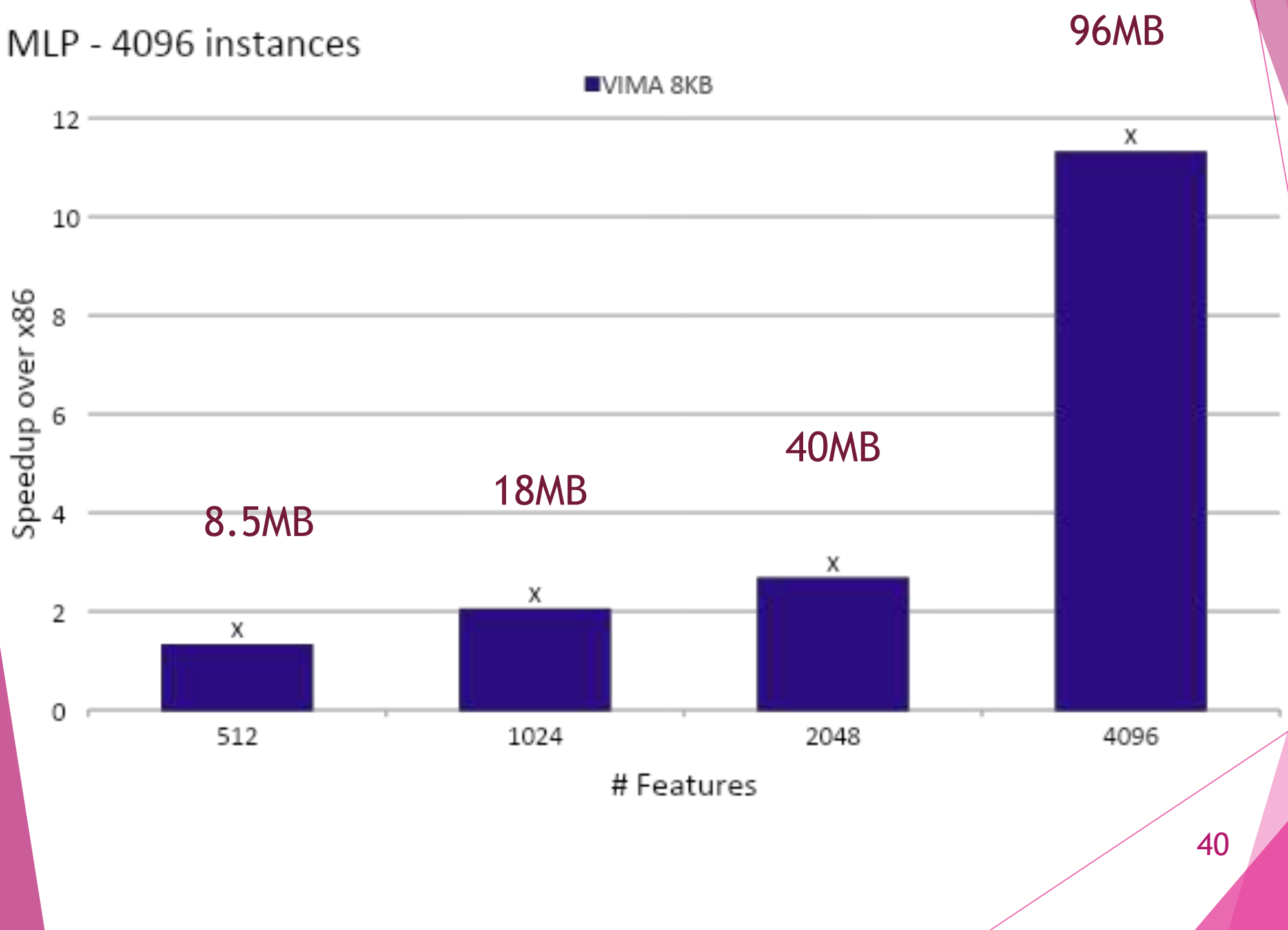
- The more instances and features, the higher the memory footprint
  - Quadratic complexity
  - Training dataset with 4096 instances and 1024 features already exceeds cache memory size



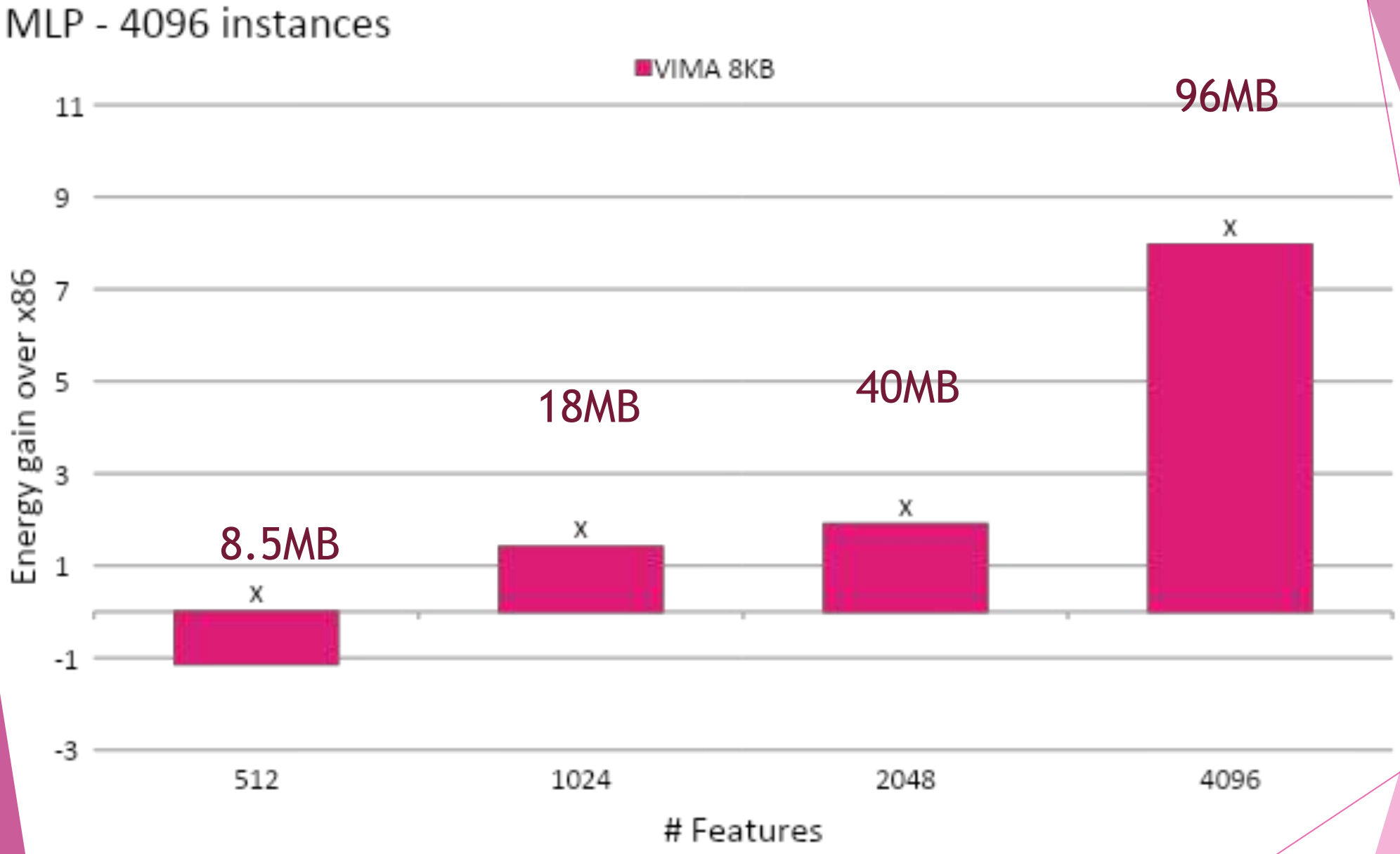
kNN conclusion

- The more instances and features, the higher the memory footprint
  - Quadratic complexity
  - Training dataset with 4096 instances and 1024 features already exceeds cache memory size





Memory Footprint higher than LLC size = 16MB



Memory Footprint higher than LLC size = 16MB

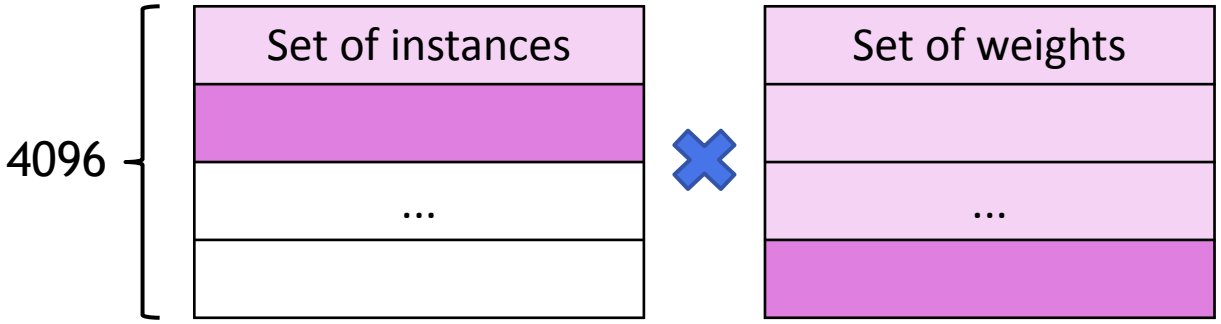
Energy from: Host processor + memory hierarchy + VIMA + VIMA's Cache

MLP conclusion

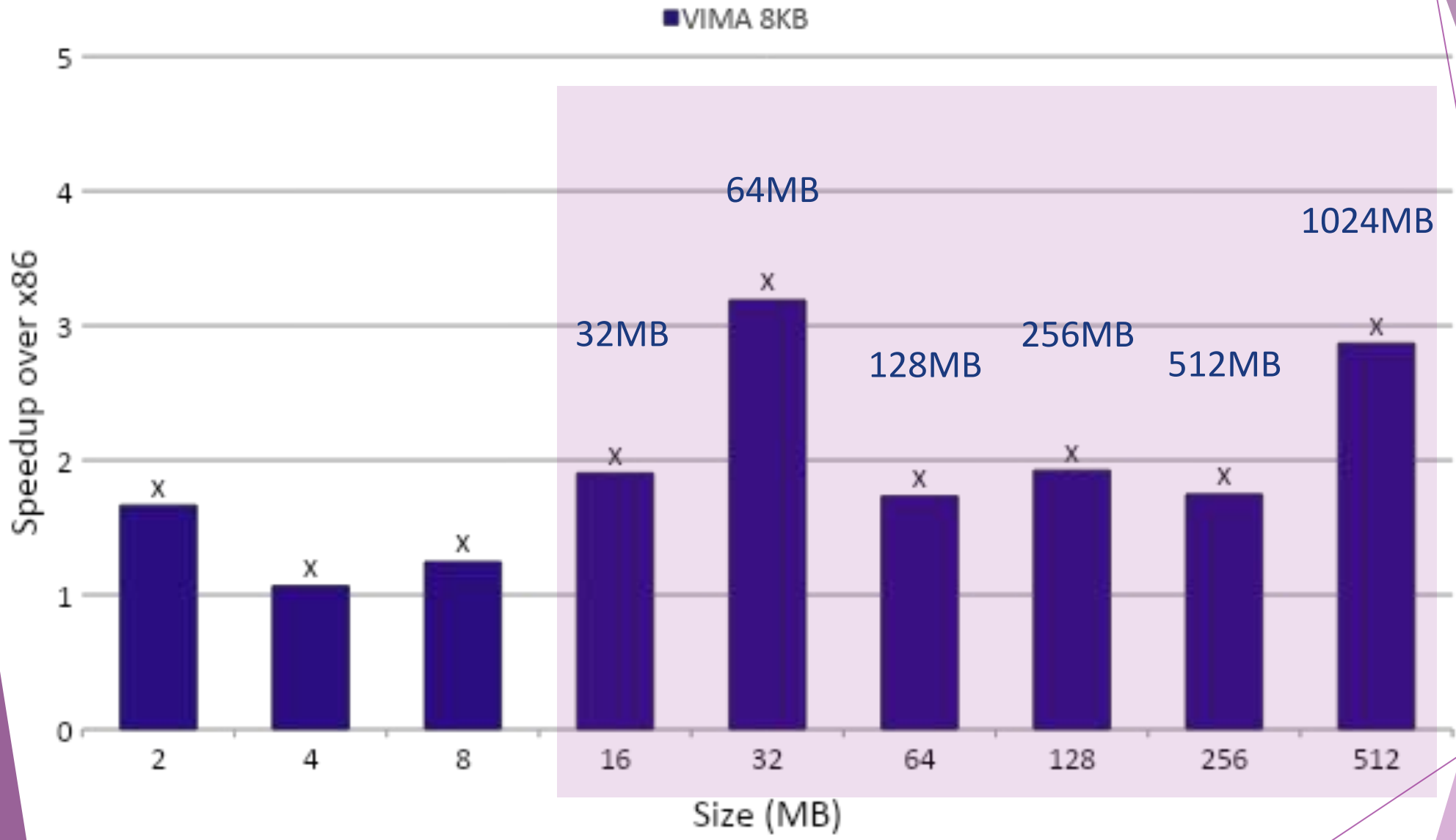
- Just the number of features influences in memory footprint
- Better performance compared to x86 just with greater sizes of features
  - Fair data reuse in cache memory for smaller sizes

The less features, the less weights' sets allows fair usage of cache memory

The more features, the more weights’ sets Behave as streaming

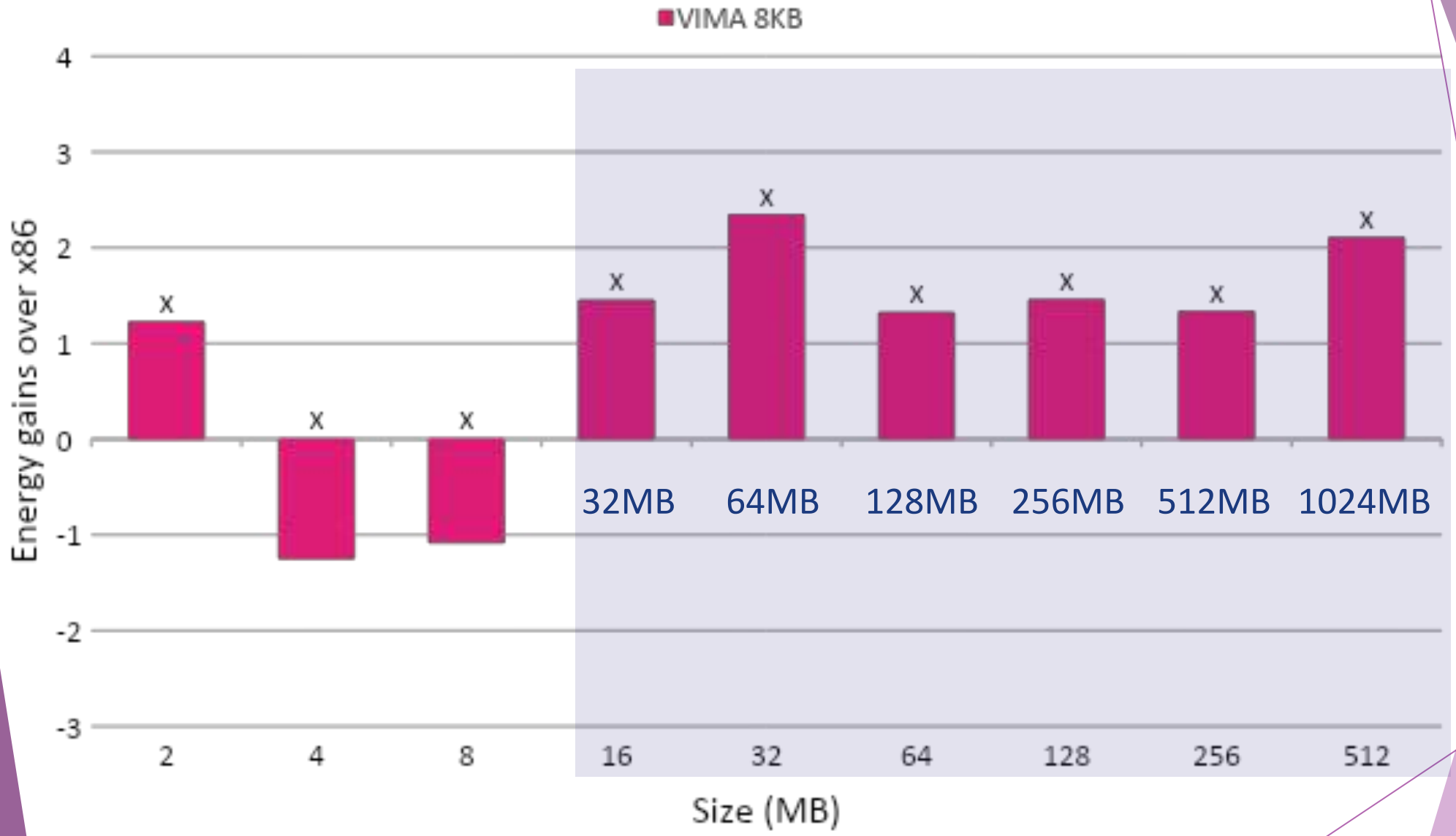


Convolution



Memory Footprint higher than LLC size = 16MB

Convolution

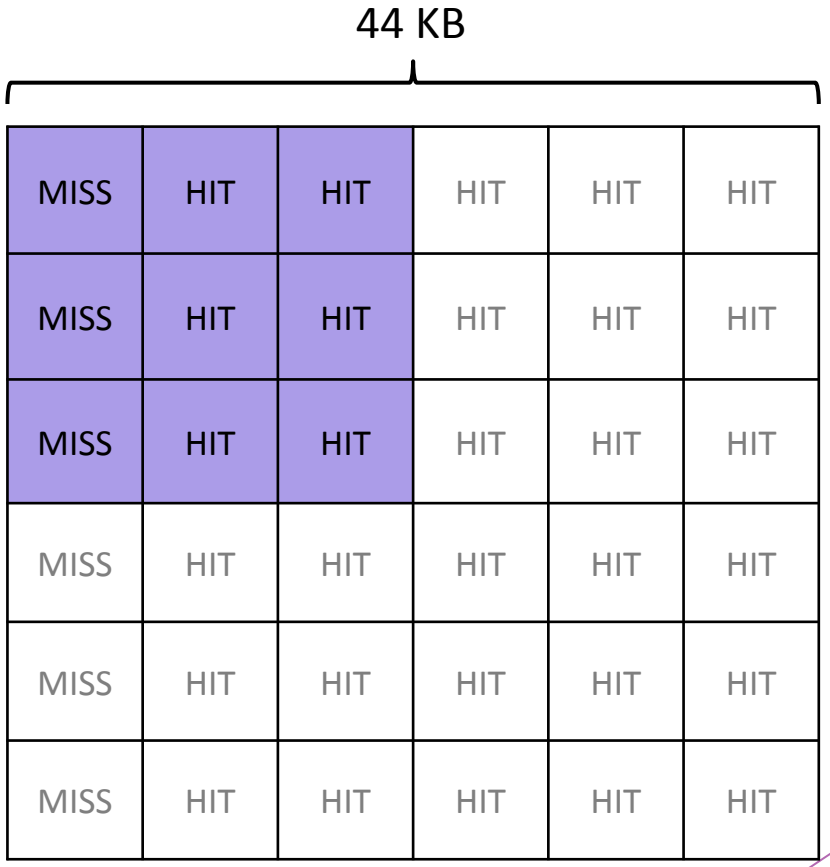


Memory Footprint higher than LLC size = 16MB

Energy from: Host processor + memory hierarchy + VIMA + VIMA's Cache

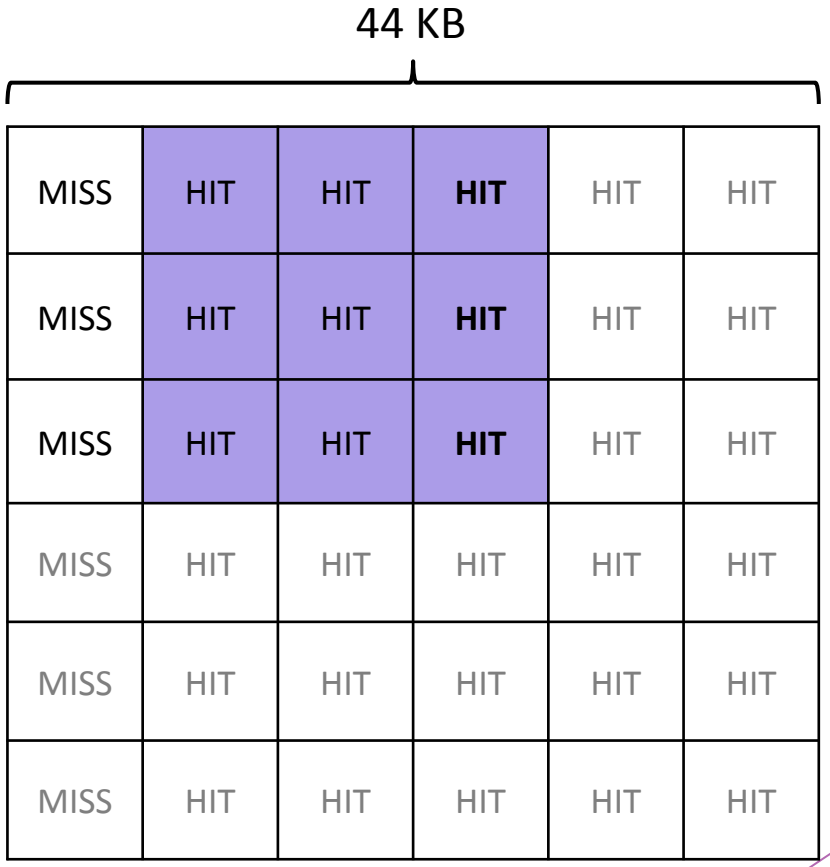
Convolution conclusion

- Slightly gains over x86
  - Fair data reuse in cache memory
- VIMA would achieve better performance if just a matrix line occupies 16MB (Cache memory size)
  - But a line in the greatest matrix occupies 44.2KB



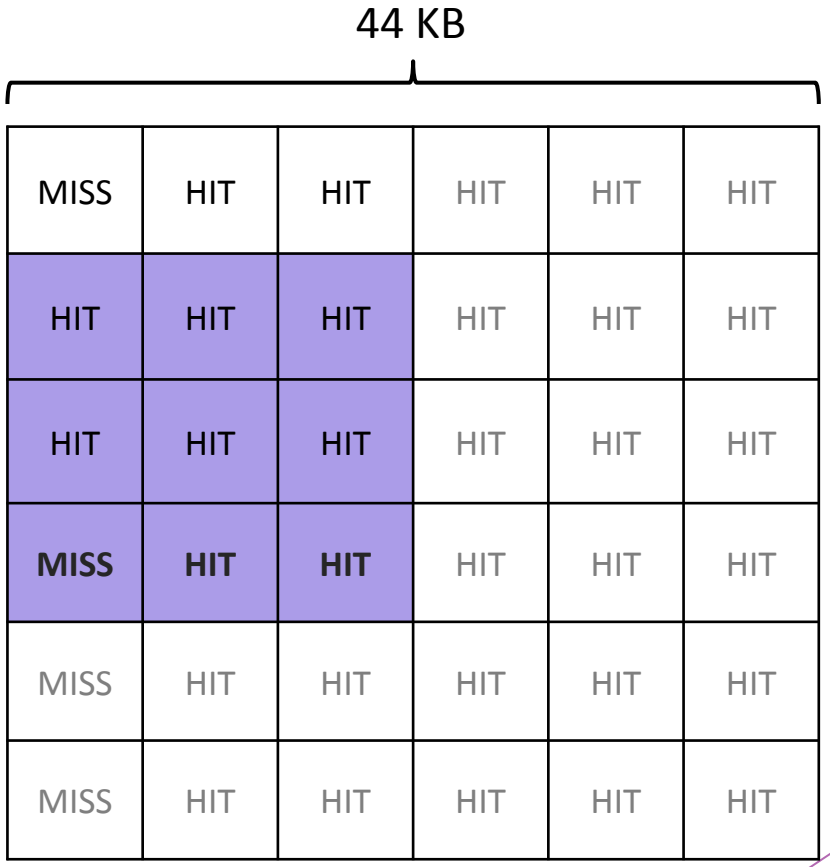
Convolution conclusion

- Slightly gains over x86
  - Fair data reuse in cache memory
- VIMA would achieve better performance if just a matrix line occupies 16MB (Cache memory size)
  - But a line in the greatest matrix occupies 44.2KB



Convolution conclusion

- Slightly gains over x86
  - Fair data reuse in cache memory
- VIMA would achieve better performance if just a matrix line occupies 16MB (Cache memory size)
  - But a line in the greatest matrix occupies 44.2KB



Conclusion

- VIMA is an accelerator
  - It does not achieve high performance for every application
- VIMA achieves higher performance when executing a huge amount of data
  - If data fits in cache memory and makes fair data re-usage, x86 achieves good performance compared to VIMA
- The algorithm’s complexity and data reuse does influence the results

Thank you!

alinesantanacordeiro@gmail.com

Machine Learning Migration for Efficient Near-Data Processing

Authors: Aline Cordeiro

Sairo Santos

Francis B. Moreira

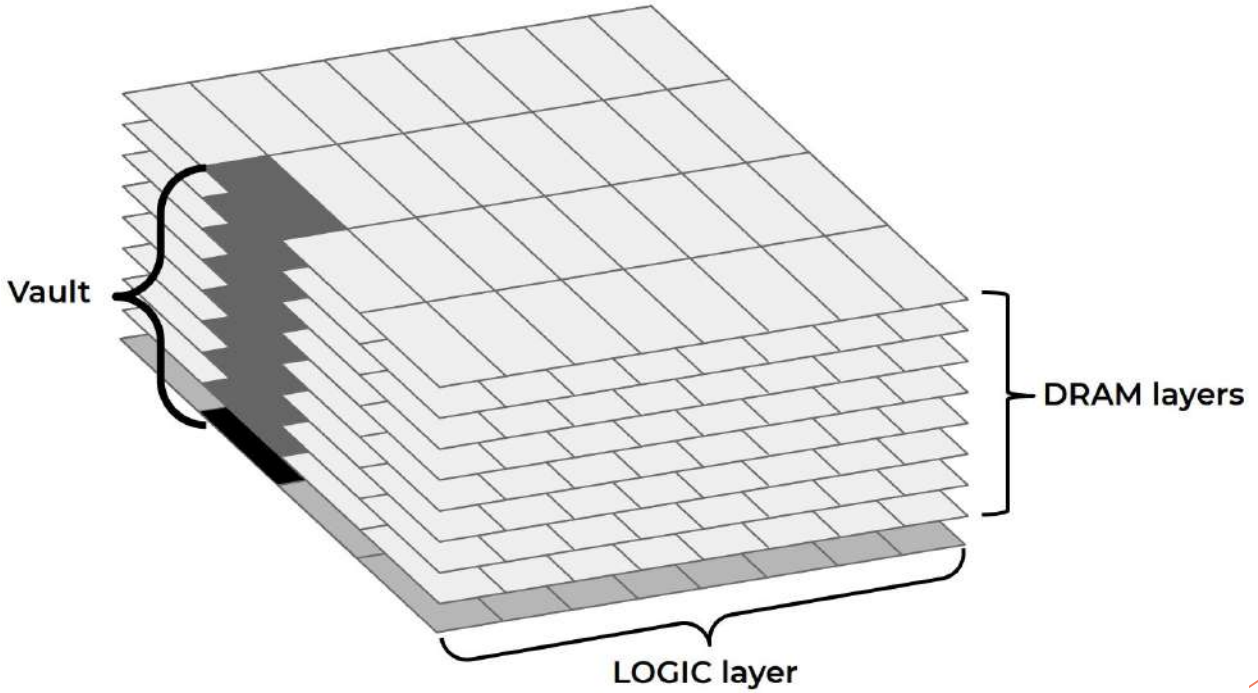
Paulo C. Santos

Luigi Carro

Marco Zanata Alves

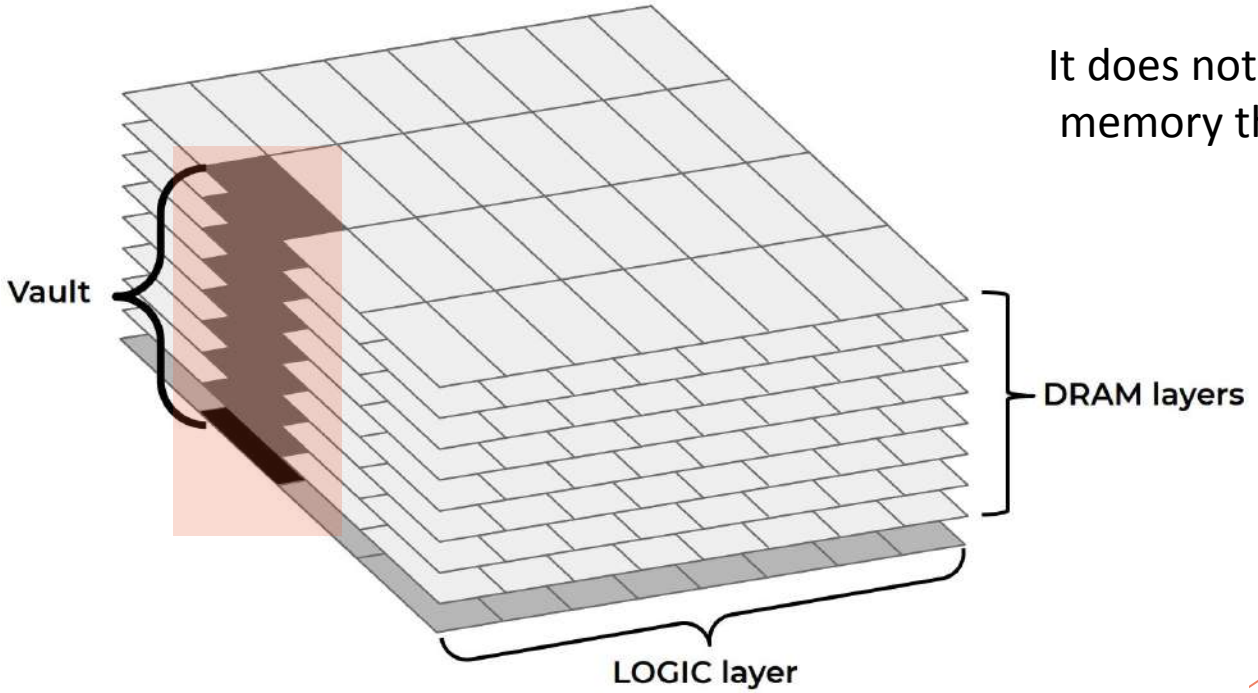
Benchmarks

- All the algorithms were implemented with 256B and 8KB VIMA vectors and AVX 512



Benchmarks

- All the algorithms were implemented with 256B and 8KB VIMA vectors and AVX 512



It does not use all the memory throughput

Estimate of the number of instructions

Considering a matrix of 28x28 and a filter of 3x3 sliding with a stride of 1:

Implementation	# instructions
Scalar	6084
AVX 512	~378
VIMA	78

Estimate of the number of instructions

Considering the inference of one instance of 32 features:

Implementation	# instructions
Scalar	1084
AVX 512	~68
VIMA	13

Estimate of the number of instructions

Considering the Euclidean Distance calculation between one training and test instances with 128 features:

Implementation	# instructions
Scalar	385
AVX 512	~24
VIMA	6

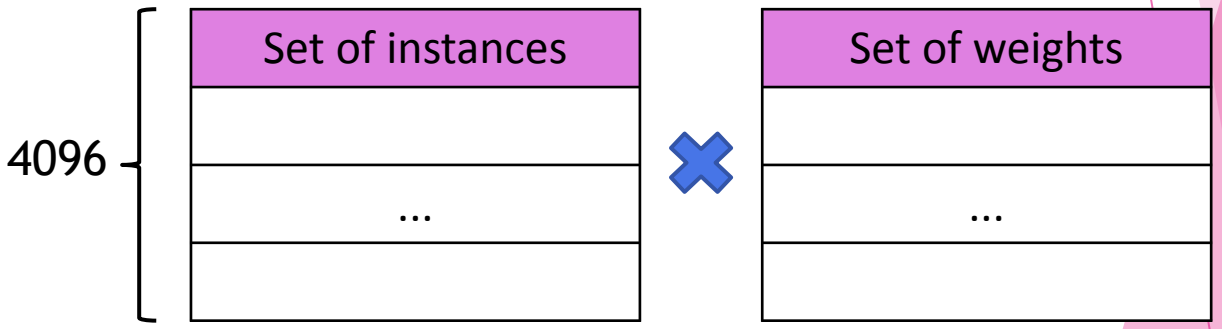
Intrinsics-VIMA

- Based on Intel intrinsics
- A library written in C/C++ language
- Reproduce VIMA Instruction Set Architecture (ISA)
- Vector instructions (256B and 8KB)
- Data representation
  - 32 and 64-bits
  - Integer and floating-point
- Data type standardized

Lomont, Chris. "Introduction to intel advanced vector extensions." *Intel white paper* 23 (2011).

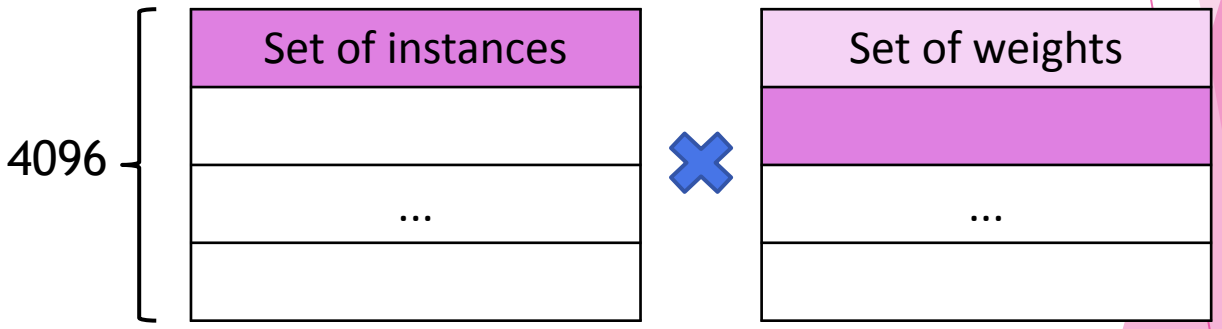
MLP conclusion

- Just the number of features influences in memory footprint
- Better performance compared to x86 just with greater sizes of features
  - Fair data reuse in cache memory for smaller sizes



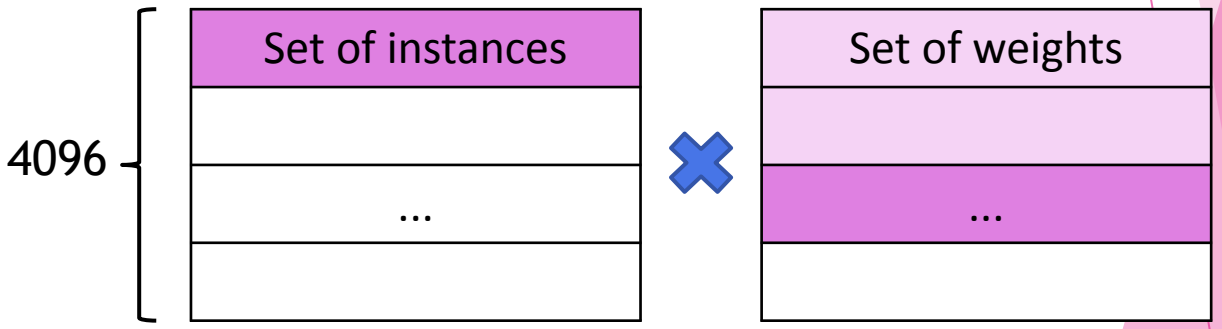
MLP conclusion

- Just the number of features influences in memory footprint
- Better performance compared to x86 just with greater sizes of features
  - Fair data reuse in cache memory for smaller sizes



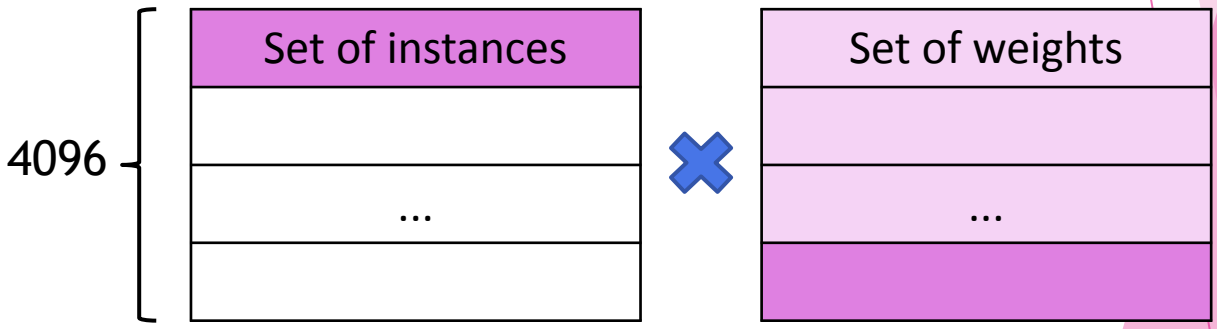
MLP conclusion

- Just the number of features influences in memory footprint
- Better performance compared to x86 just with greater sizes of features
  - Fair data reuse in cache memory for smaller sizes



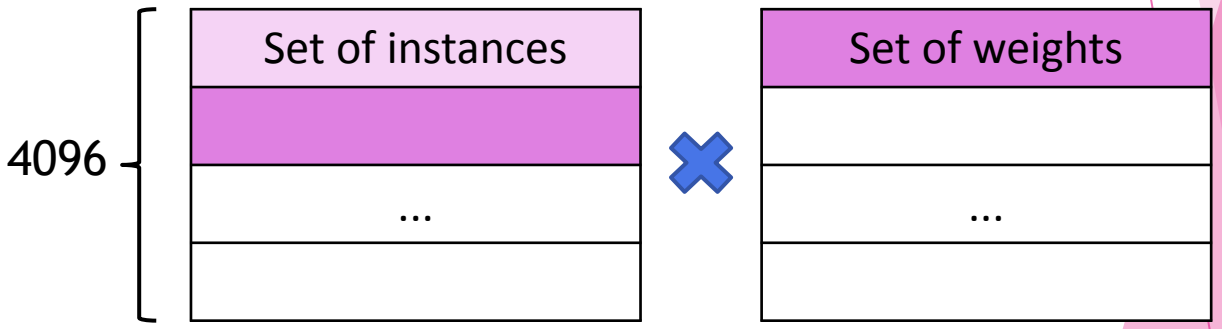
MLP conclusion

- Just the number of features influences in memory footprint
- Better performance compared to x86 just with greater sizes of features
  - Fair data reuse in cache memory for smaller sizes



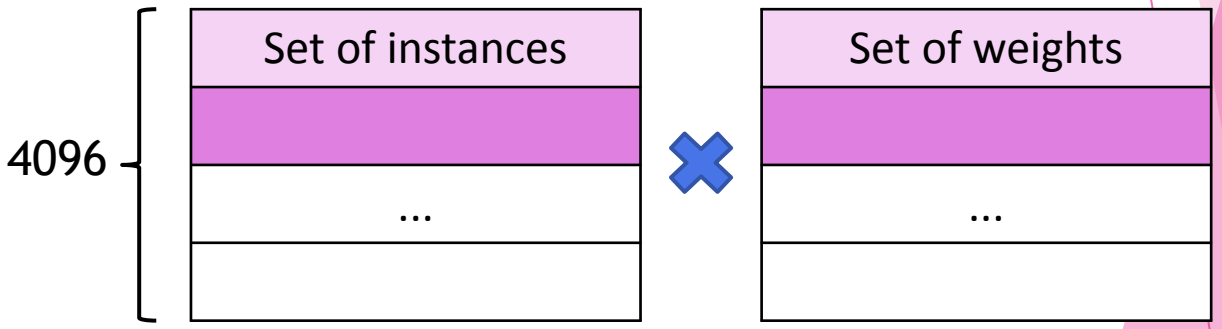
MLP conclusion

- Just the number of features influences in memory footprint
- Better performance compared to x86 just with greater sizes of features
  - Fair data reuse in cache memory for smaller sizes



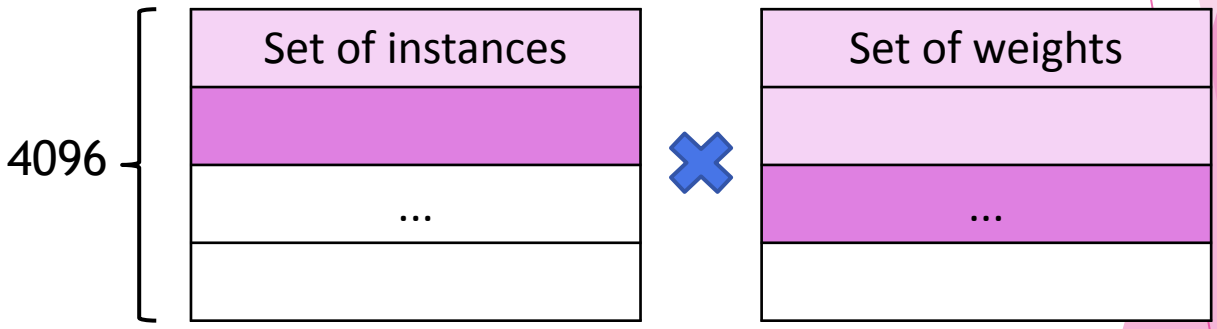
MLP conclusion

- Just the number of features influences in memory footprint
- Better performance compared to x86 just with greater sizes of features
  - Fair data reuse in cache memory for smaller sizes



MLP conclusion

- Just the number of features influences in memory footprint
- Better performance compared to x86 just with greater sizes of features
  - Fair data reuse in cache memory for smaller sizes

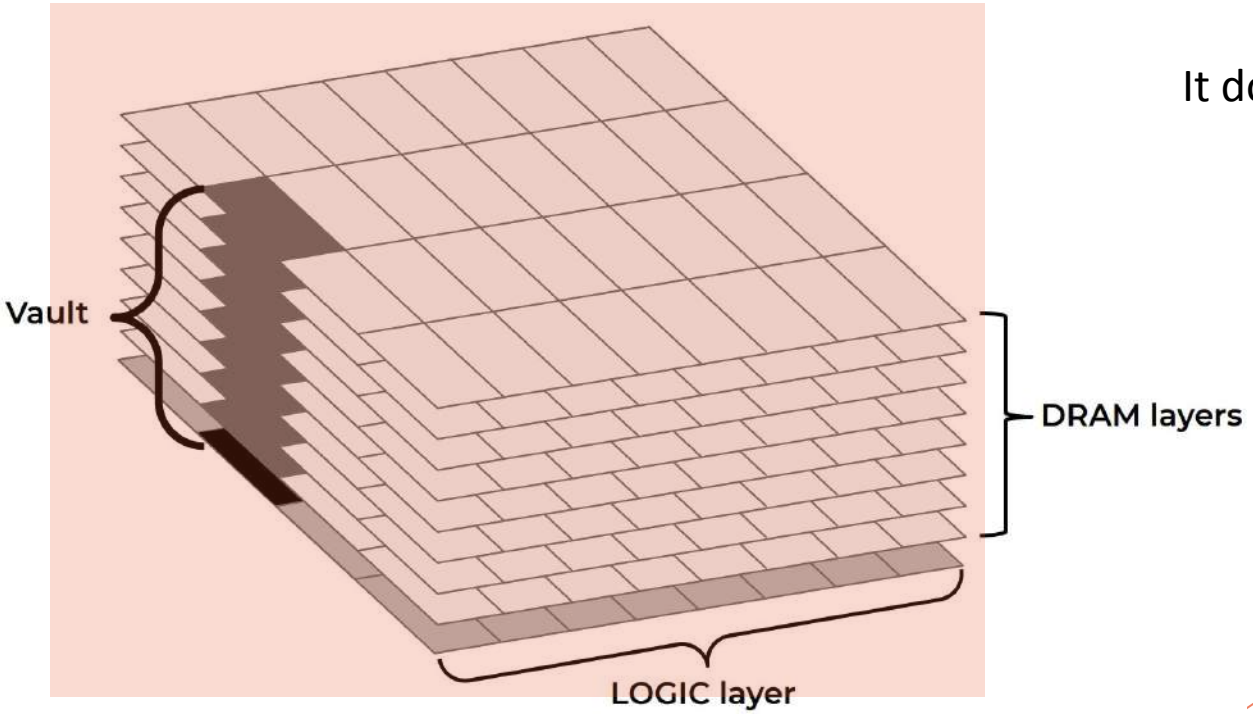


Ordinary Computing Simulator (OrCS)

- Developed in HiPES
- Trace driven simulation
- Cycle accurate
- Based on x86 architecture
- Interprets simulation traces to evaluate application behavior
- Simulates x86 ISA for 32 and 64-bits and VIMA

Benchmarks

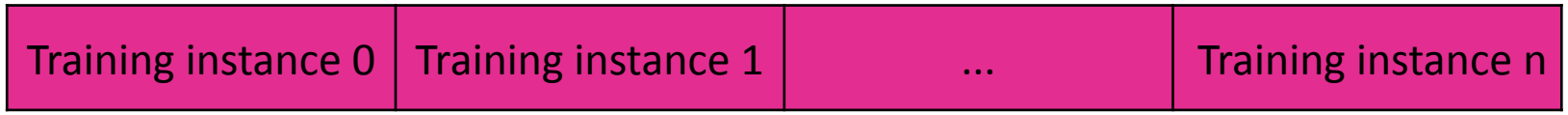
- All the algorithms were implemented with 256B and 8KB VIMA vectors and AVX 512



It does!

Algorithm for VIMA

1. Store the training dataset in an array:

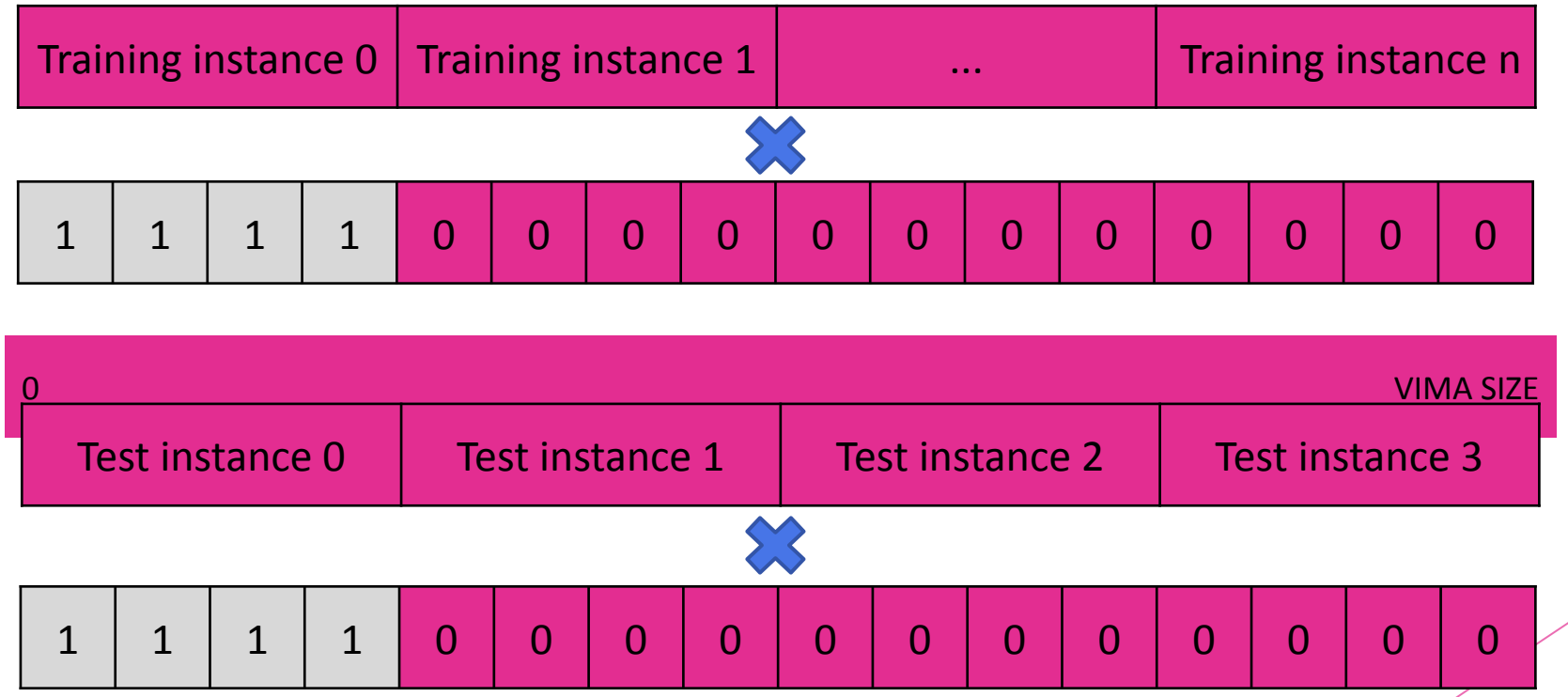


2. Read the test instances in a VIMA vector:

Algorithm for VIMA

For each iteration, a unique mask is applied for both instances
=
BETTER USAGE OF VIMA's CACHE

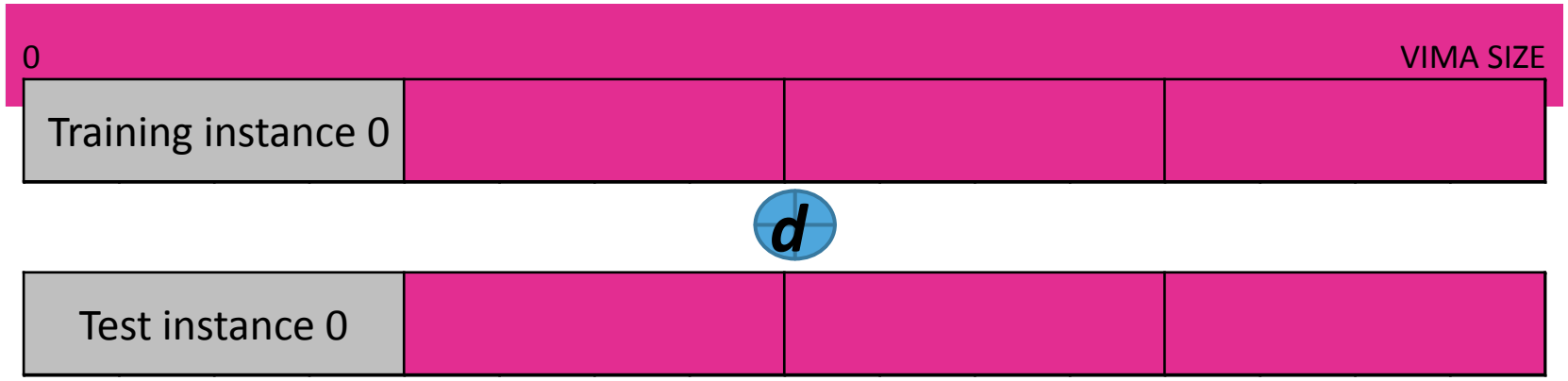
3. Apply the mask for both instances:



Algorithm for VIMA

$$d'(y,x) = \left(y_{[0,127]} - x_{i[0,127]}\right)^2$$

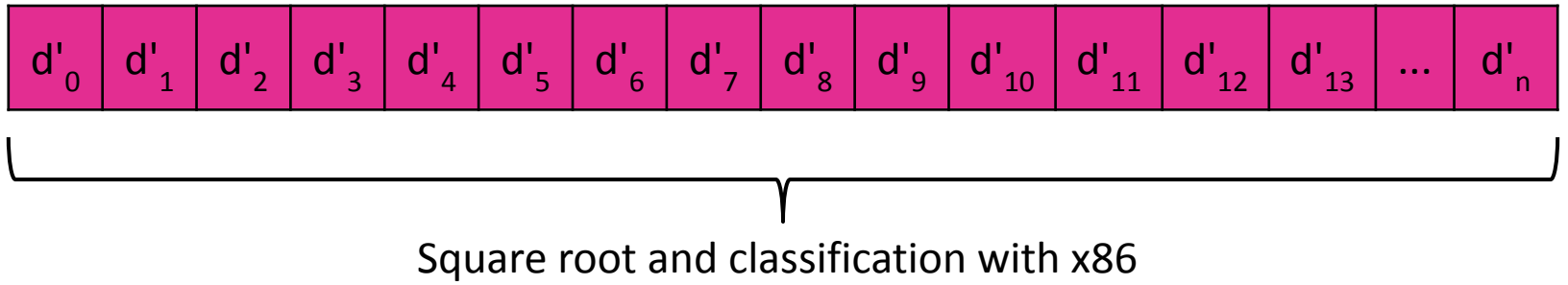
4. Operate a partial Euclidean Distance between the instances



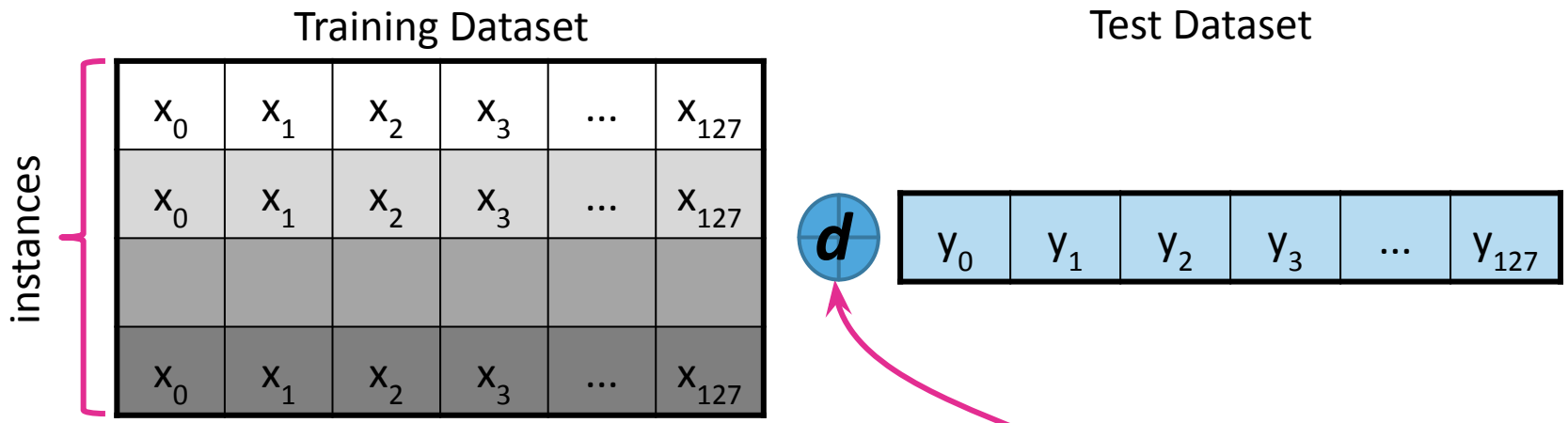
Algorithm for VIMA

$$d'(y,x) = \sqrt{\sum_{i=0}^{n} \left(y_{[0,127]} - x_{i[0,127]}\right)^2}$$

5. Finalize the operation with x86 routines



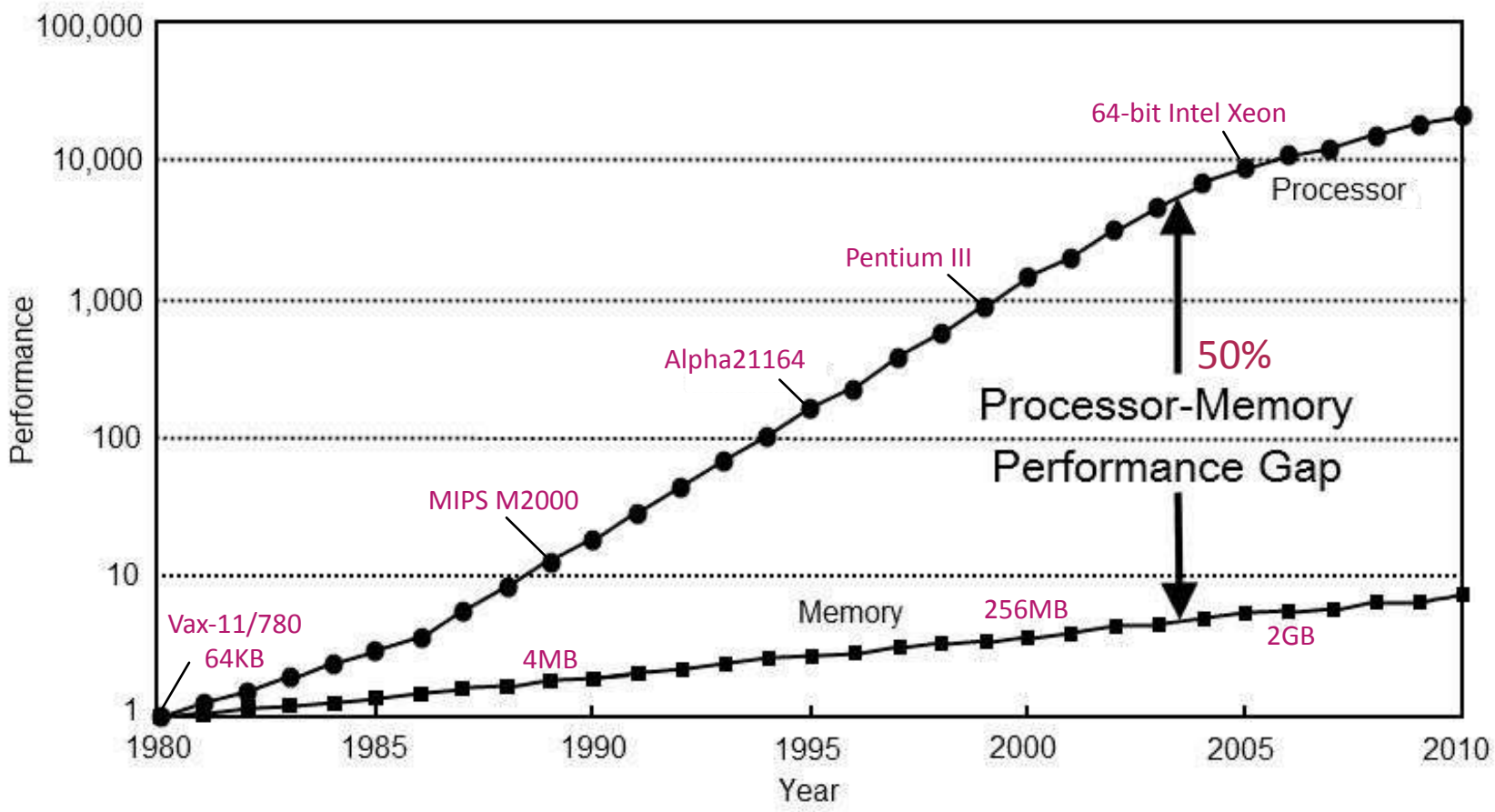
x86's implementation



Scalar operations between one test instance and the training dataset

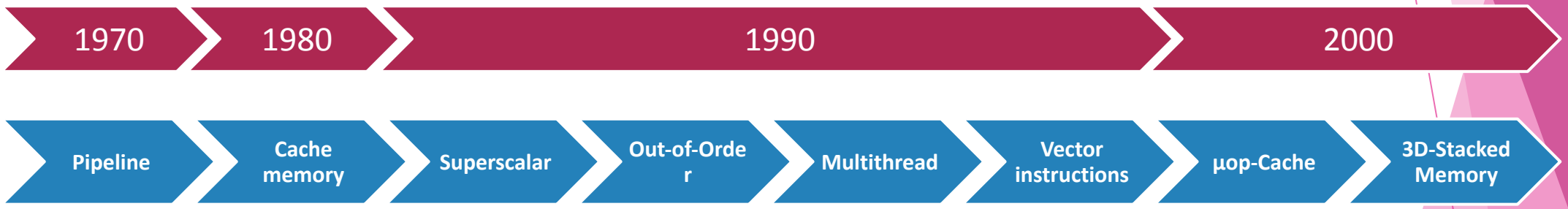
$$d(y,x) = \sqrt{\sum_{i=0}^{n} \left(y_{[0,127]} - x_{i[0,127]}\right)^2}$$

Processor-Memory performance gap: Speed



Efnusheva, Danijela, Ana Cholakoska, and Aristotel Tentov. "A survey of different approaches for overcoming the processor-memory bottleneck." *International Journal of Computer Science and Information Technology* 9.2 (2017): 151-163.

Over the years, different solutions arised



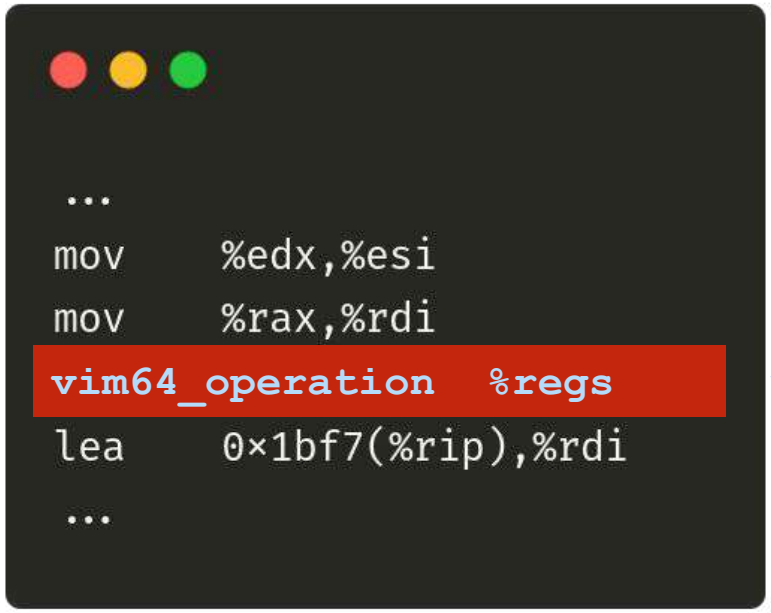
Trace generation

```
...
mov     %edx,%esi
mov     %rax,%rdi
callq   3819 <vim64_routine>
lea     0×1bf7(%rip),%rdi
...
```

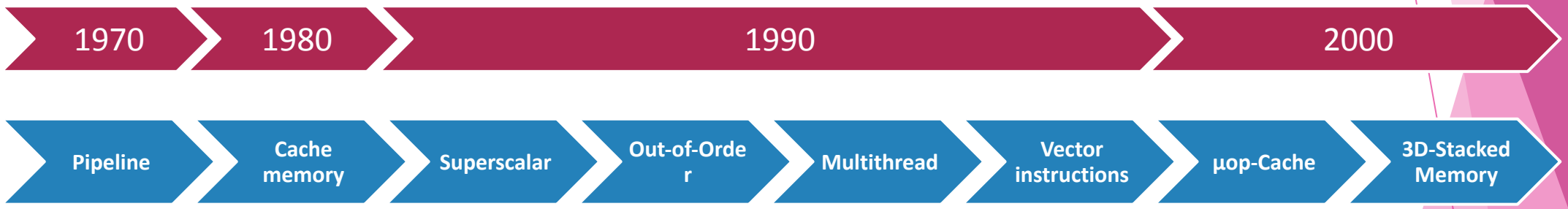
```
<vim64_routine>:
  push   %rbp
  mov    %rsp,%rbp
  mov    %rdi,-0×18(%rbp)
  mov    %esi,-0×1c(%rbp)
  lea    0×0(,%rax,4),%rdx
  add    %rax,%rdx
  mov    -0×1c(%rbp),%eax
  mov    %eax,(%rdx)
  addl   $0×1,-0×4(%rbp)
  pop    %rbp
  retq
```

Trace generation

There is no processor with VIMA instructions so a synthetic VIMA instruction, that is understood by a simulator, is added.



Over the years, different solutions arised

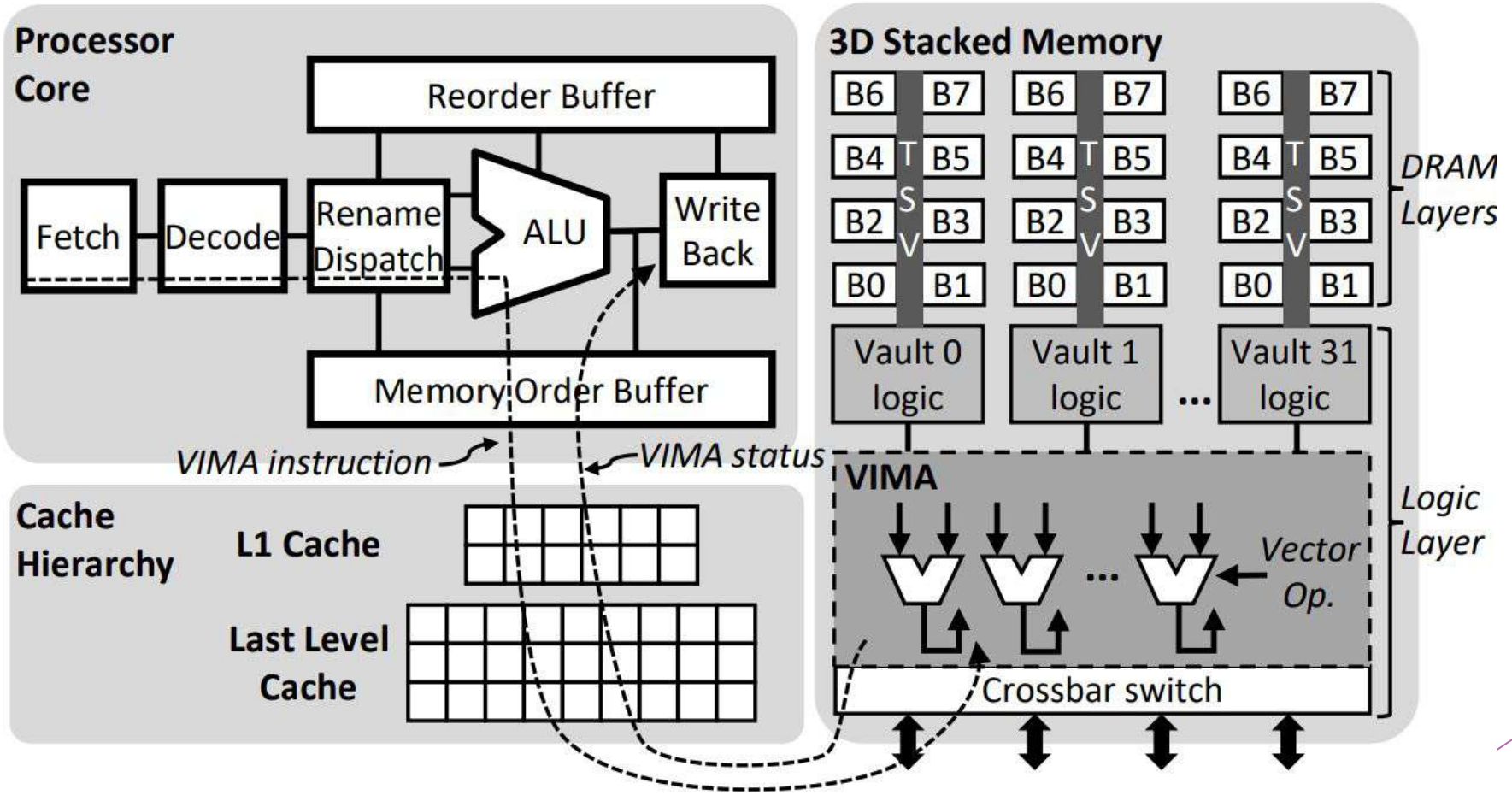


Most of these improvements were implemented on the processor...

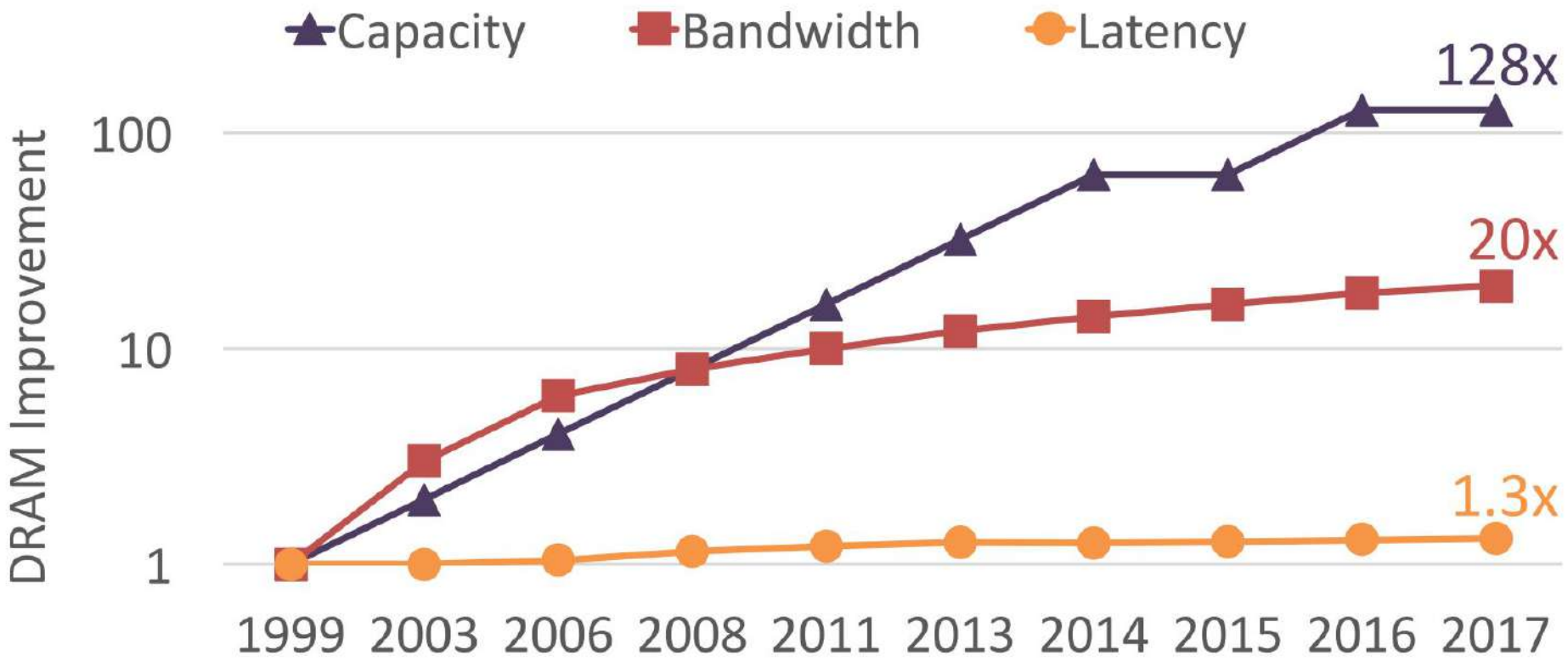
Trace Generator

- Instrumentation tool
- It executes and analyze the binary code at the same time and generate simulation traces

VIMA execution process



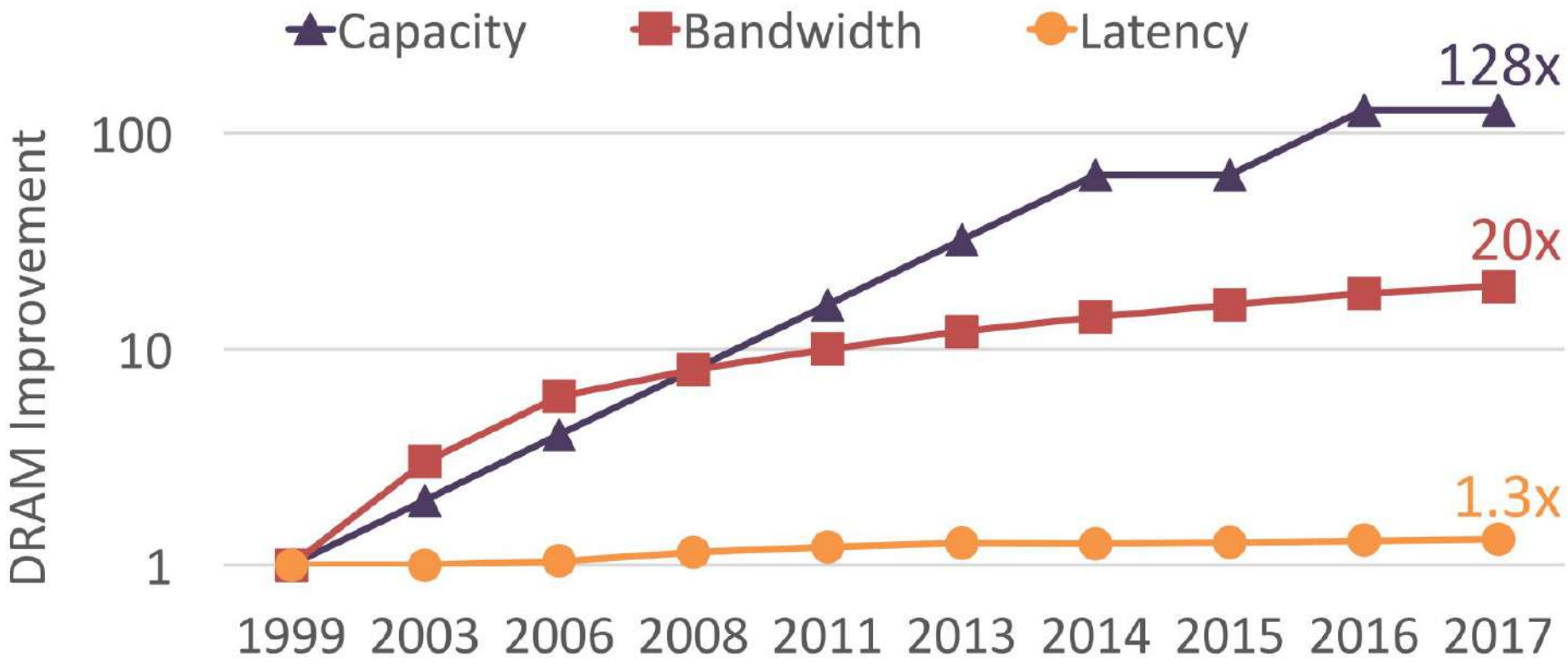
Memory improvement



Chang, Kevin K. "Understanding and improving the latency of DRAM-based memory systems." *arXiv preprint arXiv:1712.08304* (2017).

Memory improvement

But the relative latency remained almost the same

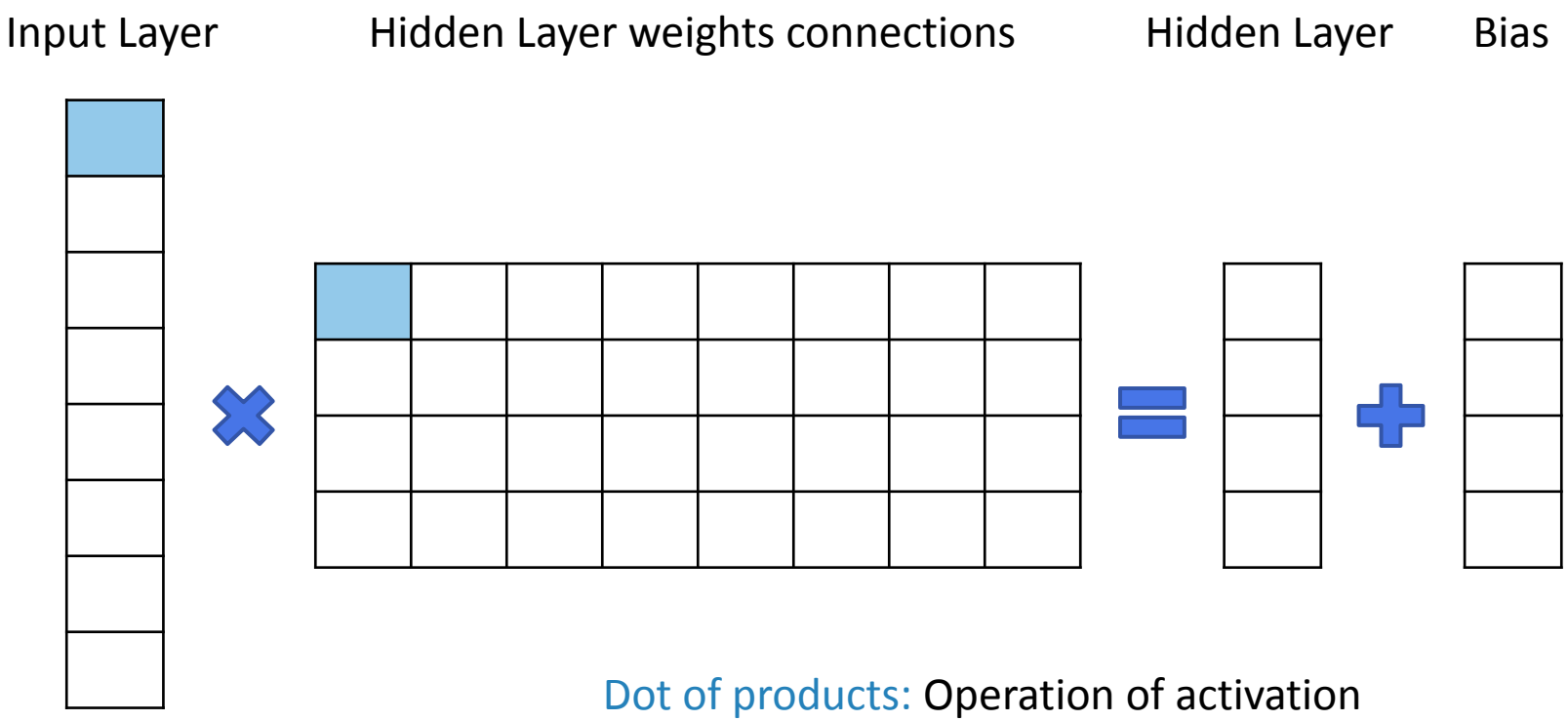


Chang, Kevin K. "Understanding and improving the latency of DRAM-based memory systems." *arXiv preprint arXiv:1712.08304* (2017).

MultiLayer Perceptron

x86 x VIMA

x86's implementation



Dot of products: Operation of activation values of the hidden layer

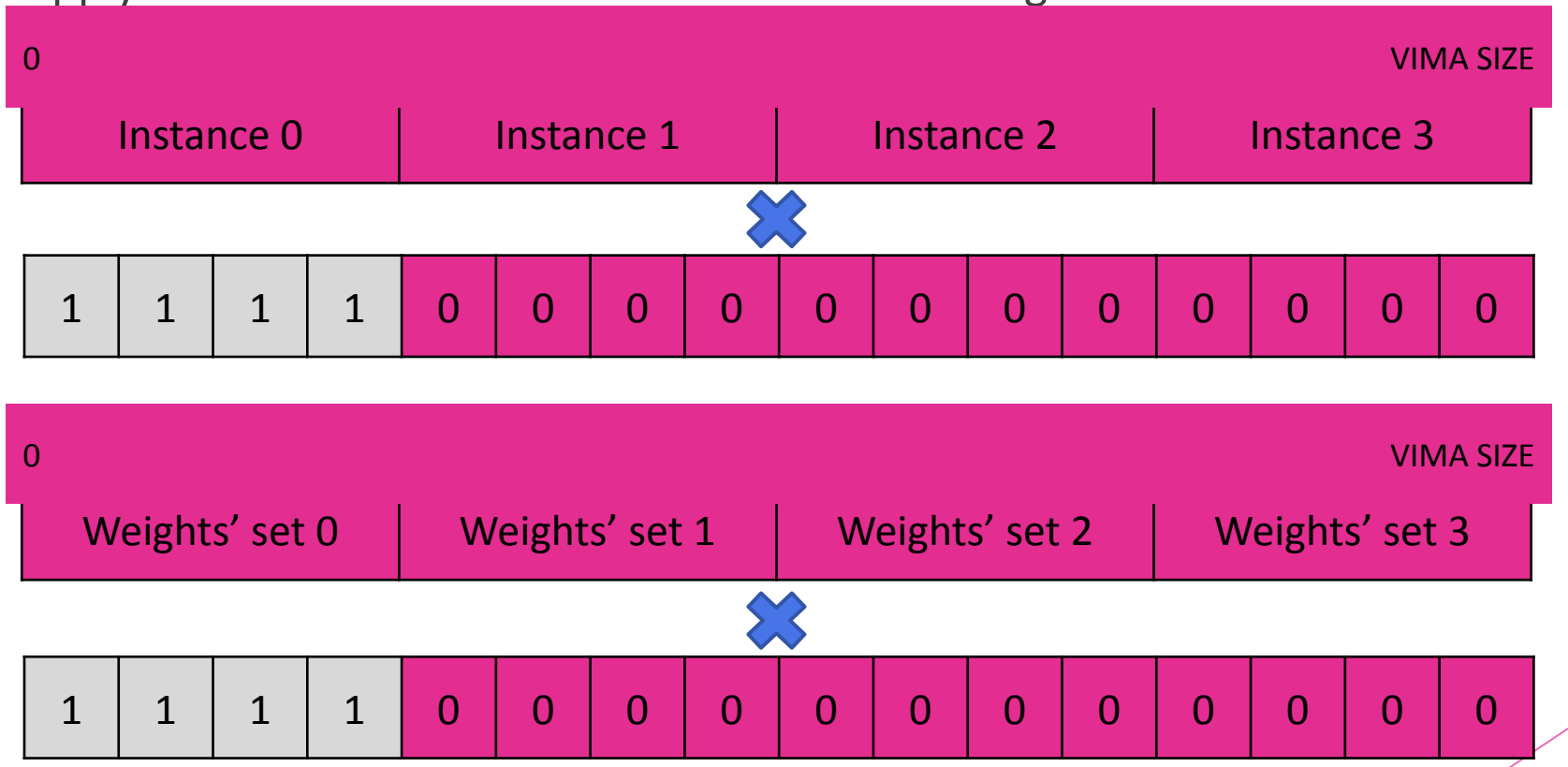
Algorithm for VIMA: Inference Only

1. Read the instances:

2. Initialize the sets of weights:

Algorithm for VIMA: Inference Only

3. Apply the mask for both instances and sets of weights:



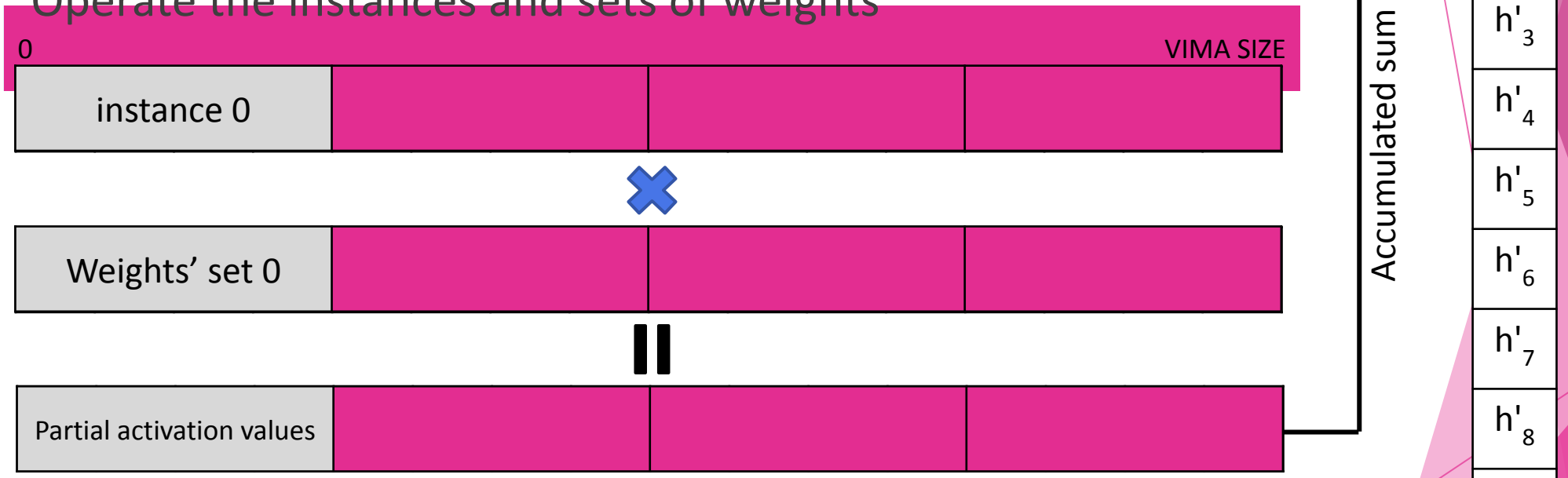
For each iteration, a unique mask is applied for both instance and weights

=

BETTER USAGE OF VIMA's CACHE

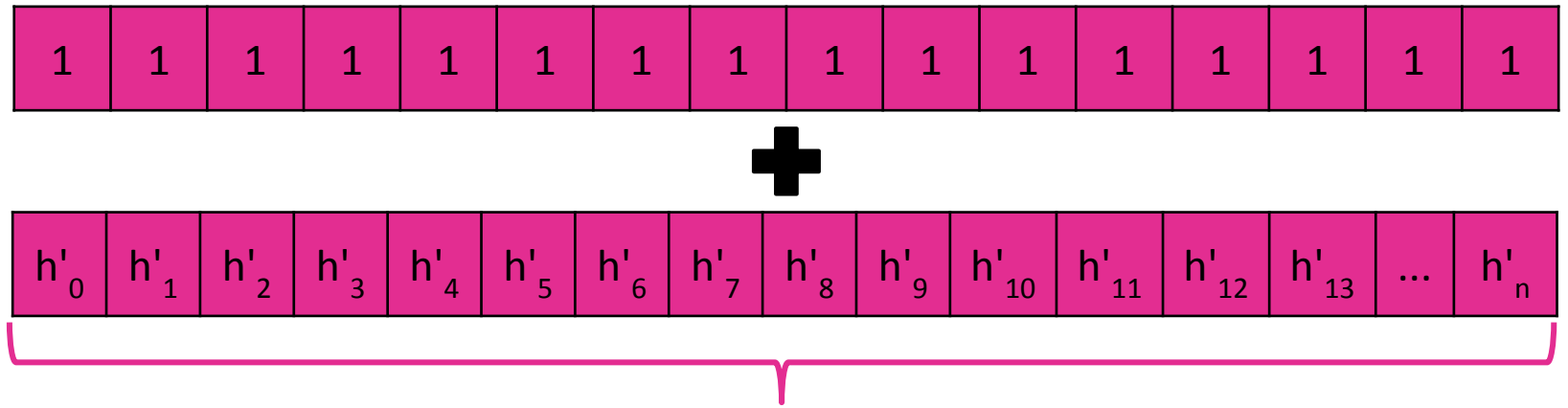
Algorithm for VIMA: Inference Only

4. Operate the instances and sets of weights



Algorithm for VIMA: Inference Only

5. Hidden layer partial activation values added to the bias



Algorithm for VIMA: Inference Only

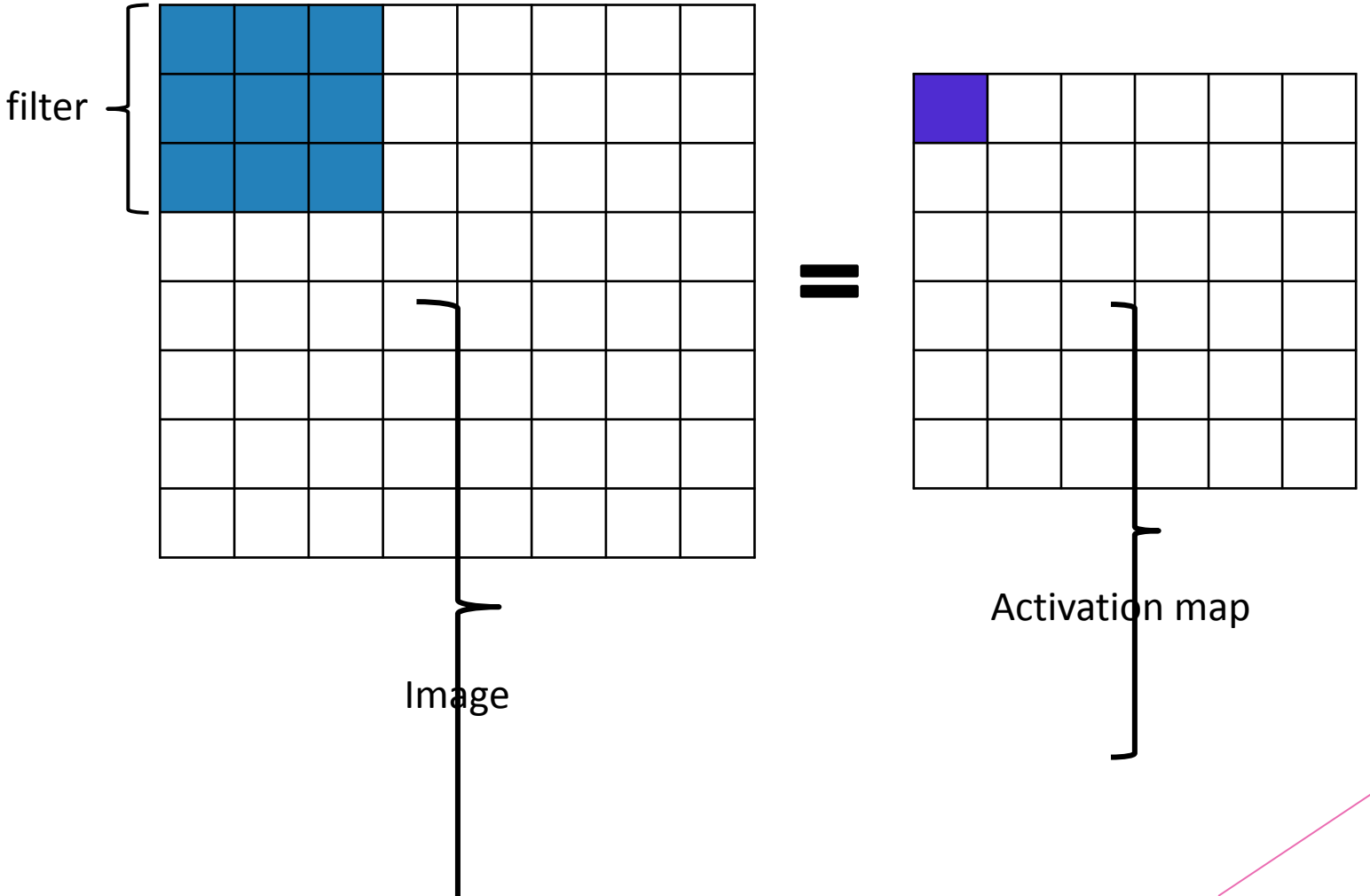
6. The same process is executed to calculate the output layer activation values, the differences are:
   - The mask size;
   - The activation function applied at the final of the operation

7. With the activation values of the output layer, each instance is classified with x86 instructions

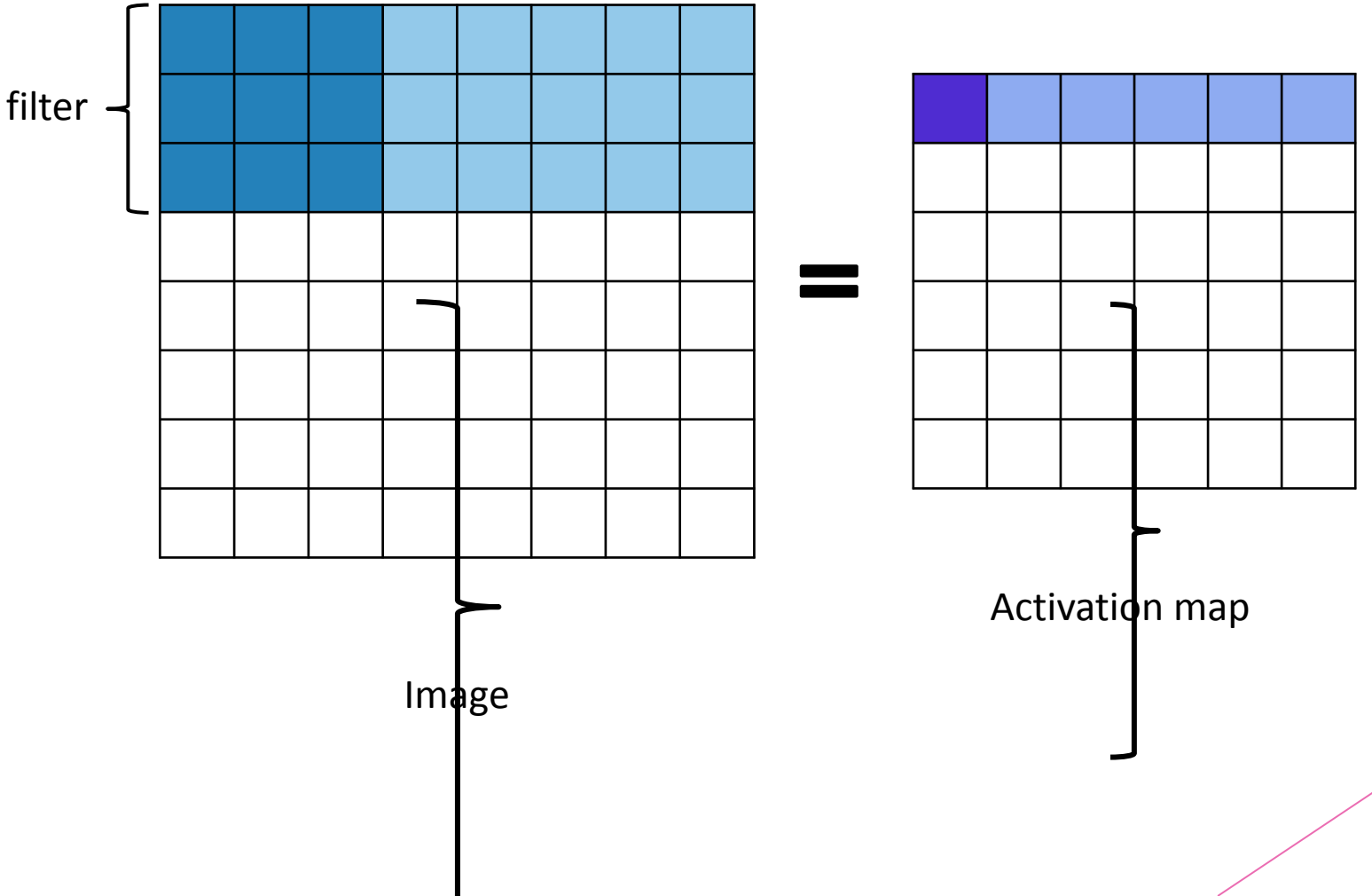
Convolution

x86 x VIMA

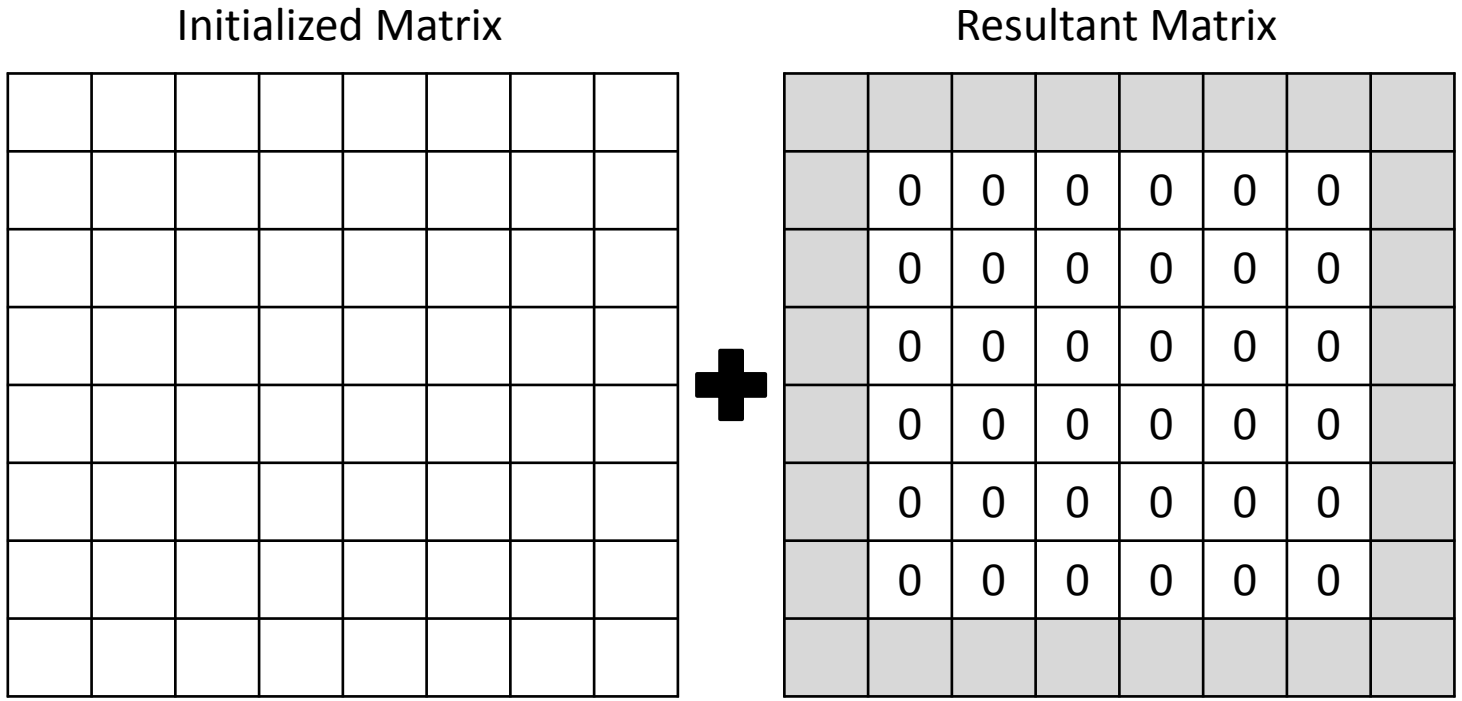
x86’s implementation



VIMA's implementation



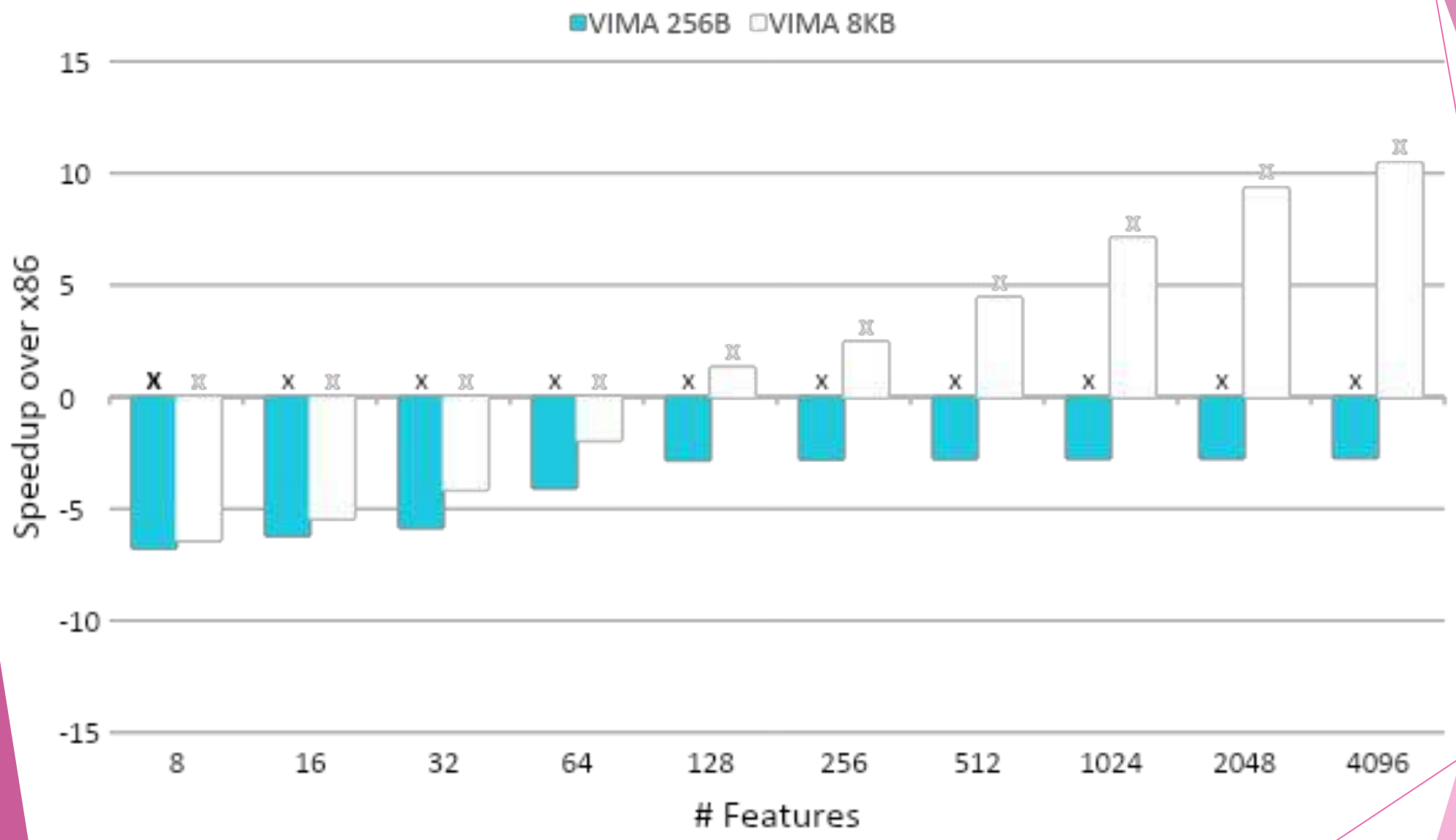
Algorithm for VIMA

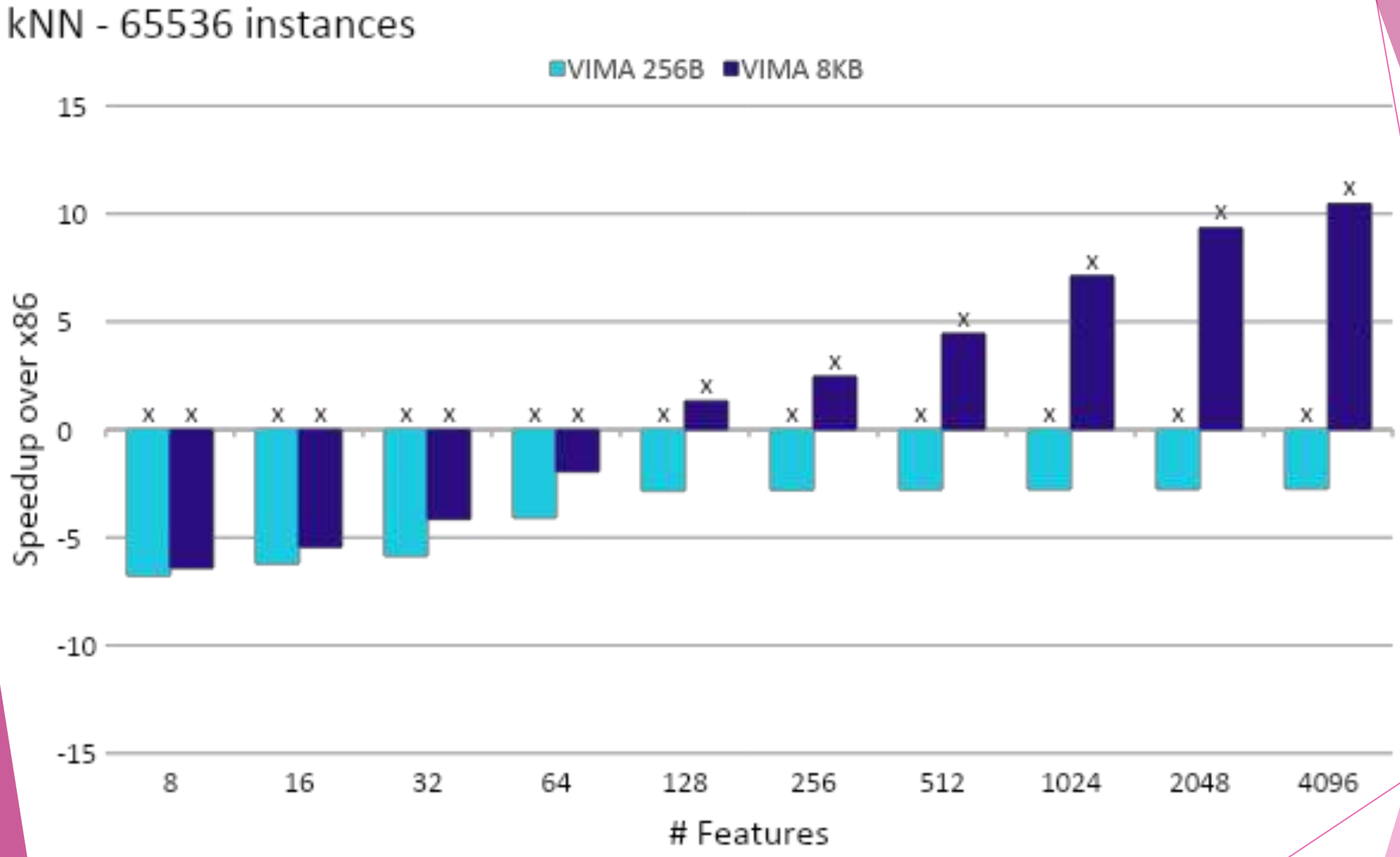


Benchmarks

Results

Speedup and Energy consumption


kNN - 65536 instances
VIMA 256B
VIMA 8KB
Speedup over x86
15
10
5
0
-5
-10
-15
8
16
32
64
128
256
512
1024
2048
4096
Features


kNN - 65536 instances
VIMA 256B
VIMA 8KB
Speedup over x86
15
10
5
0
-5
-10
-15
8
16
32
64
128
256
512
1024
2048
4096
Features

Conclusion

And contributions

GPU comparison

- Considering papers that evaluate the same applications used here on GPUs
  - GPUs can be 30x faster than VIMA
  - But they consume 66x more energy!

A Dawwd, Shefa, and Noor M AL Layla. "Training Acceleration of Multi-Layer Perceptron using Multicore CPU and GPU under MATLAB Environment." *AL-Rafdain Engineering Journal (AREJ)* 23.3 (2015): 136-148.

Skryjomski, Przemysław, Bartosz Krawczyk, and Alberto Cano. "Speeding up k-nearest neighbors classifier for large-scale multi-label learning on GPUs." *Neurocomputing* 354 (2019): 10-19.

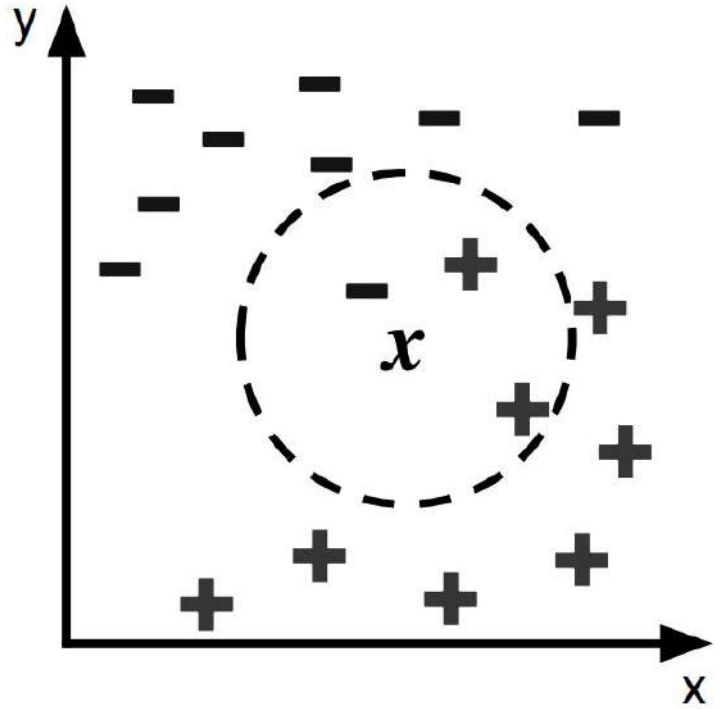
Siklosi, Balint, Istvan Z. Reguly, and Gihan R. Mudalige. "Heterogeneous cpu-gpu execution of stencil applications." *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*. IEEE, 2018.

Contributions

- Simulator: Refactored cache memory hierarchy
- Simulator: Implemented a first version of configuration files
- Participation in VIMA's idealization and implementation
- Participation in Trace-Generator implementation
- Intrinsics-VIMA development
- Portability of ML applications
- Participation in writing of 2 papers
  - One rejected by DATE
  - One submitted in PDP and waiting

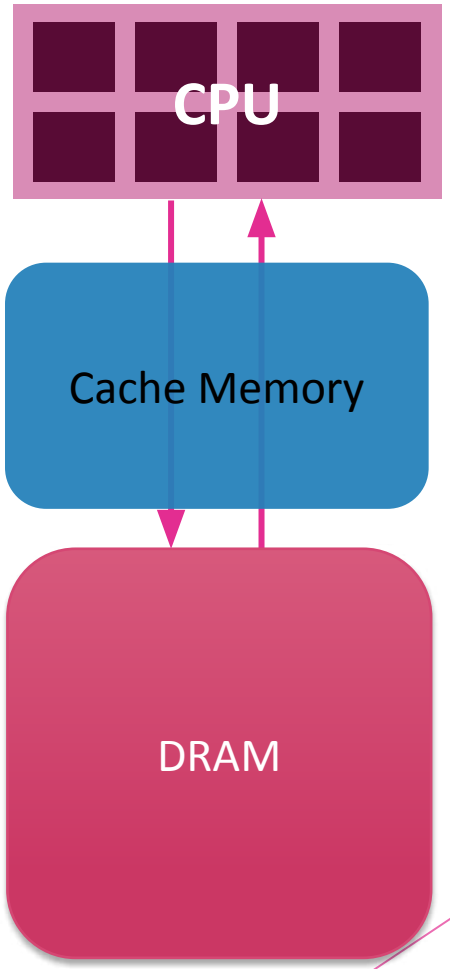
k-Nearest Neighbors

x86 x VIMA



Machine Learning applications in x86 CPU

- Memory and computationally expensive algorithms
  - Multiple memory requests
  - High data transference rate
- Massive amount of executed data
  - High energy consumption
  - High execution latency



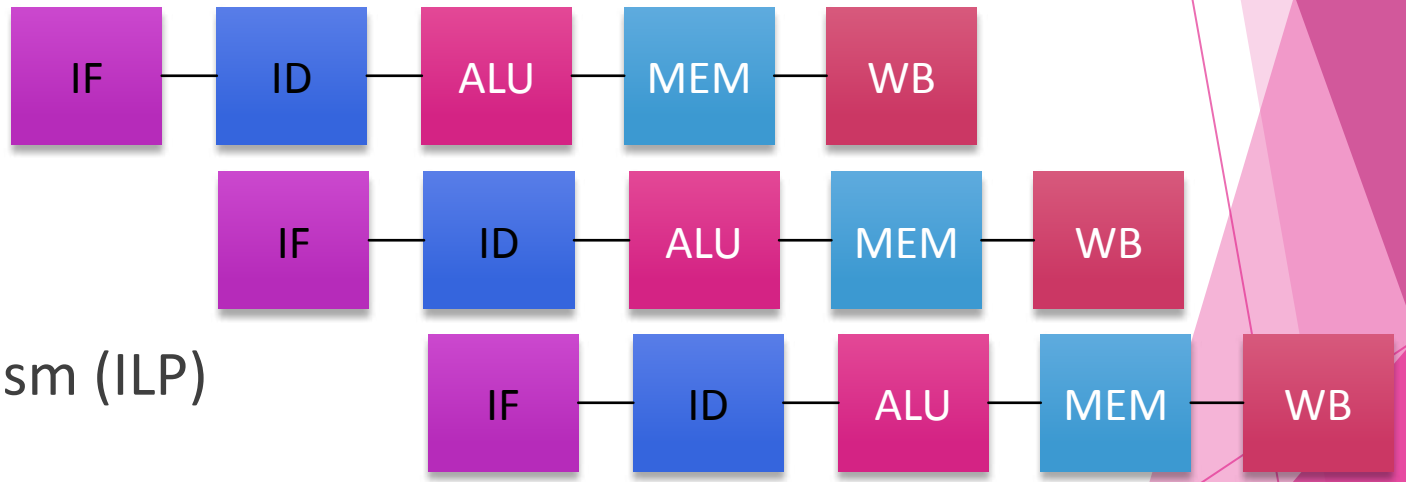
What Machine Learning is?

- Allow a computer with no prior knowledge to make correct decisions:
  - Progressive learning
  - Adapting itself to environment changes
  - Generalize learning

- Complex algorithms:
  - High computational performance
  - Memory capacity
  - Analyze a huge amount of data

Samuel, Arthur L. "Some studies in machine learning using the game of checkers." *IBM Journal of research and development* 3.3 (1959): 210-229.

CPU optimizations: Pipeline



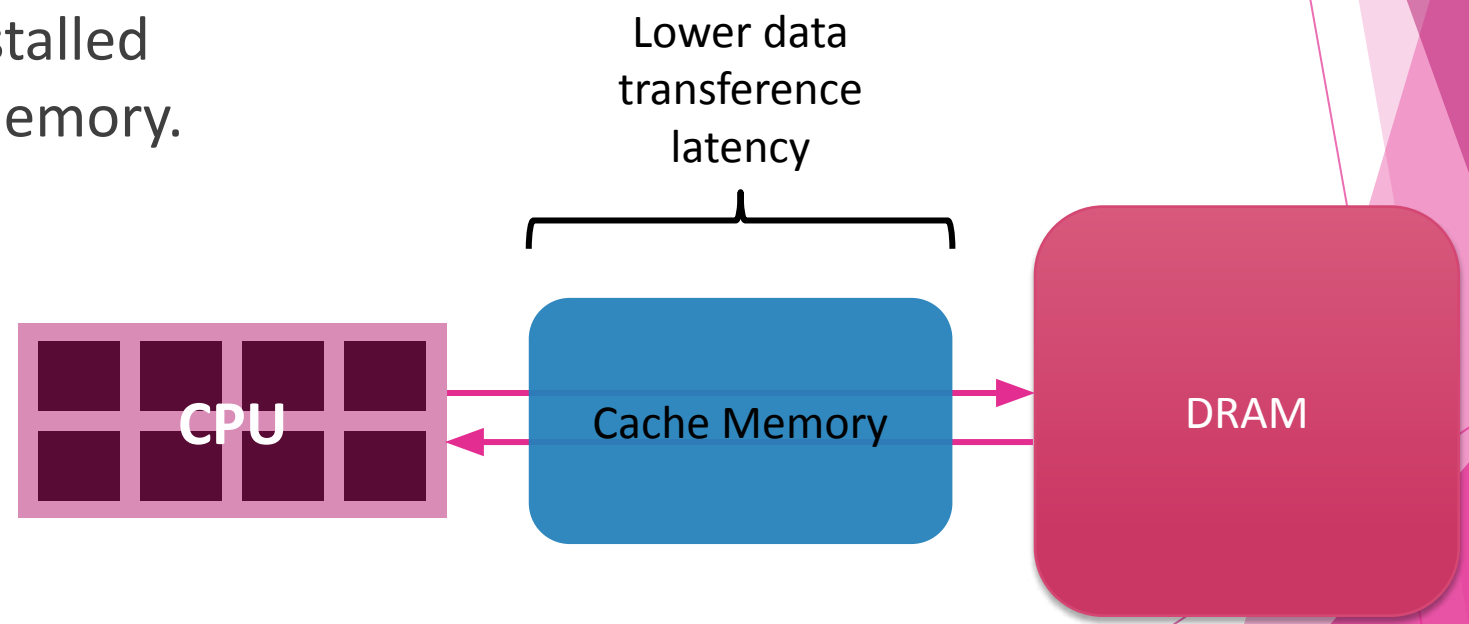
- 1 instruction/cycle
- Instruction-Level Parallelism (ILP)

Shen, John Paul, and Mikko H. Lipasti. *Modern processor design: fundamentals of superscalar processors*. Waveland Press, 2013.

Data transference latency optimization: Cache Memory

1970 | 1980 | 1990 | 2000

- A Cache memory installed between CPU and memory.
- Level latencies:
  - L1 □ 2 ns
  - L2 □ 10 ns
  - LLC □ 22 ns
  - RAM □ 100 ns

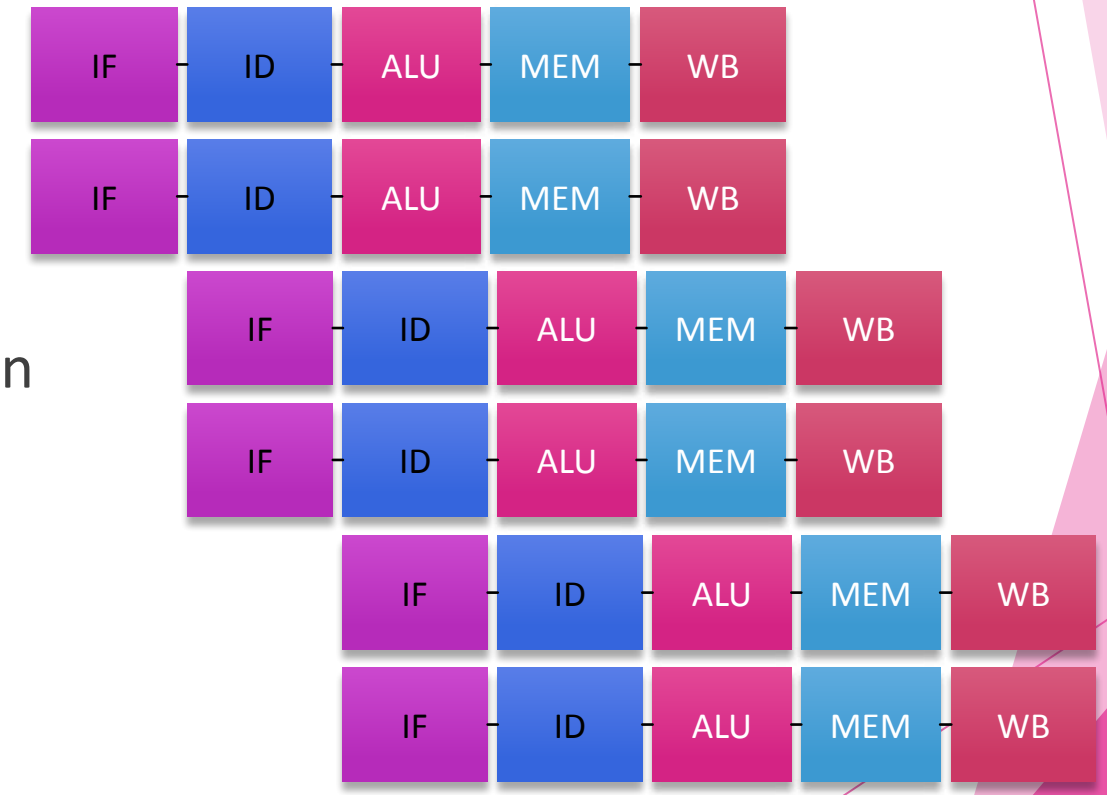


Smith, Alan Jay. "Cache memories." *ACM Computing Surveys (CSUR)* 14.3 (1982): 473-530.

CPU optimizations: Superscalar

1970 | 1980 | 1990 | 2000

- >1 instruction/cycle
- Only one functional unit (FU) in the pipeline

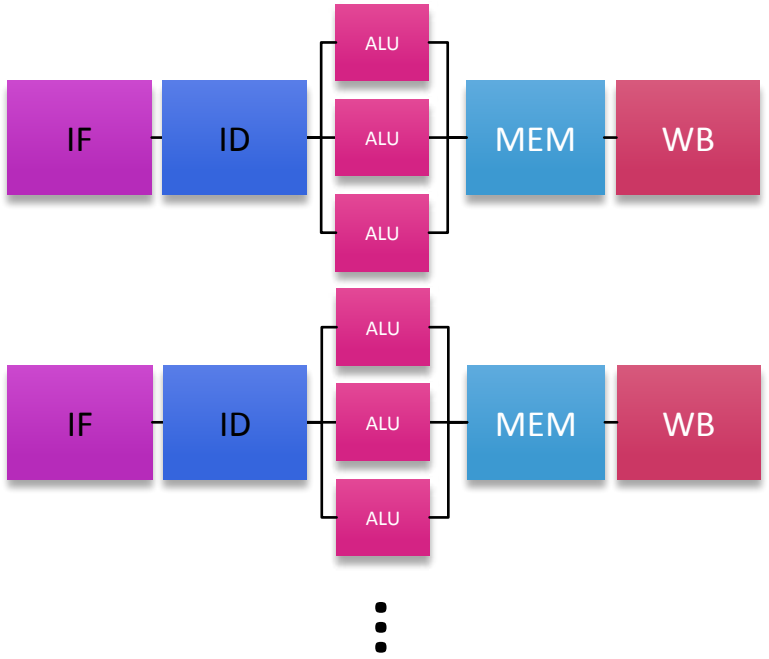


Shen, John Paul, and Mikko H. Lipasti. *Modern processor design: fundamentals of superscalar processors*. Waveland Press, 2013.

CPU optimizations: Superscalar Out-of-Order

1970 | 1980 | 1990 | 2000

- >1 instruction/cycle
- More than one FU in the pipeline



Shen, John Paul, and Mikko H. Lipasti. *Modern processor design: fundamentals of superscalar processors*. Waveland Press, 2013.

CPU optimizations: Multithreading and Multiprocessing

- Multiple concurrent threads and multiple cores
  - Shared memory
  - High memory throughput
  - High energy consumption

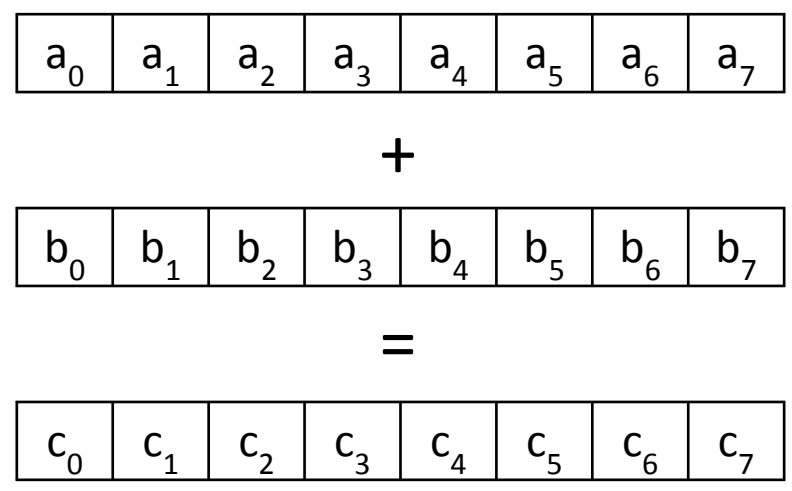
Shen, John Paul, and Mikko H. Lipasti. *Modern processor design: fundamentals of superscalar processors*. Waveland Press, 2013.

CPU optimization: Data vectorization

1970 | 1980 | 1990 | 2000

Scalar execution { a + b = c

Vector execution {



- Intel Intrinsics:
  - MMX, SSE, AVX, AVX2, AVX-512
  - Code optimization
  - executes up to 16 operands in an instruction

Lomont, Chris. "Introduction to intel advanced vector extensions." *Intel white paper* 23 (2011).

µop-Cache

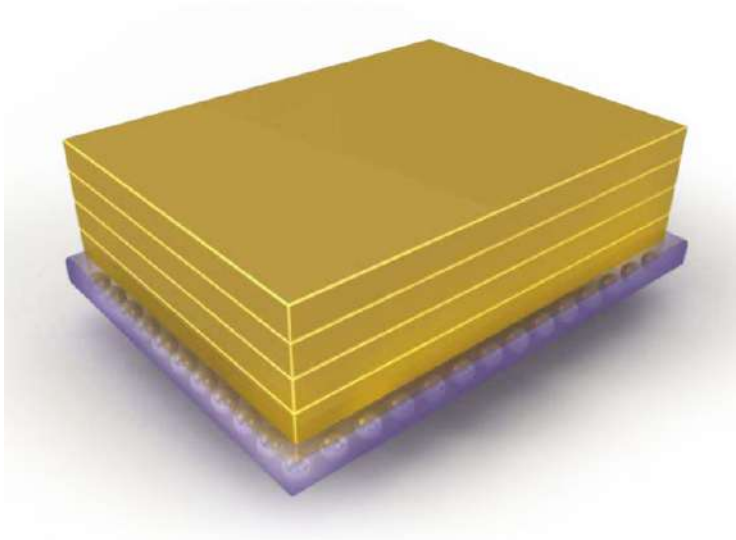
- Specialized cache
- Stores small sequences of micro-operations of decoded instructions
- Avoids fetching data from memory and decode it again

Solomon, Baruch, et al. "Micro-operation cache: A power aware frontend for variable instruction length isa." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 11.5 (2003): 801-811.

3D-Stacked Memory

1970 → 1980 → 1990 → 2000

- Integrates processing in the same chip as memory
- Avoids data transference
- Reduce execution time
- Reduce energy consumption



Pawlowski, J. Thomas. "Hybrid memory cube (HMC)." *2011 IEEE Hot chips 23 symposium (HCS)*. IEEE, 2011.

Jun, Hongshin, et al. "Hbm (high bandwidth memory) dram technology and architecture." *2017 IEEE International Memory Workshop (IMW)*. IEEE, 2017.

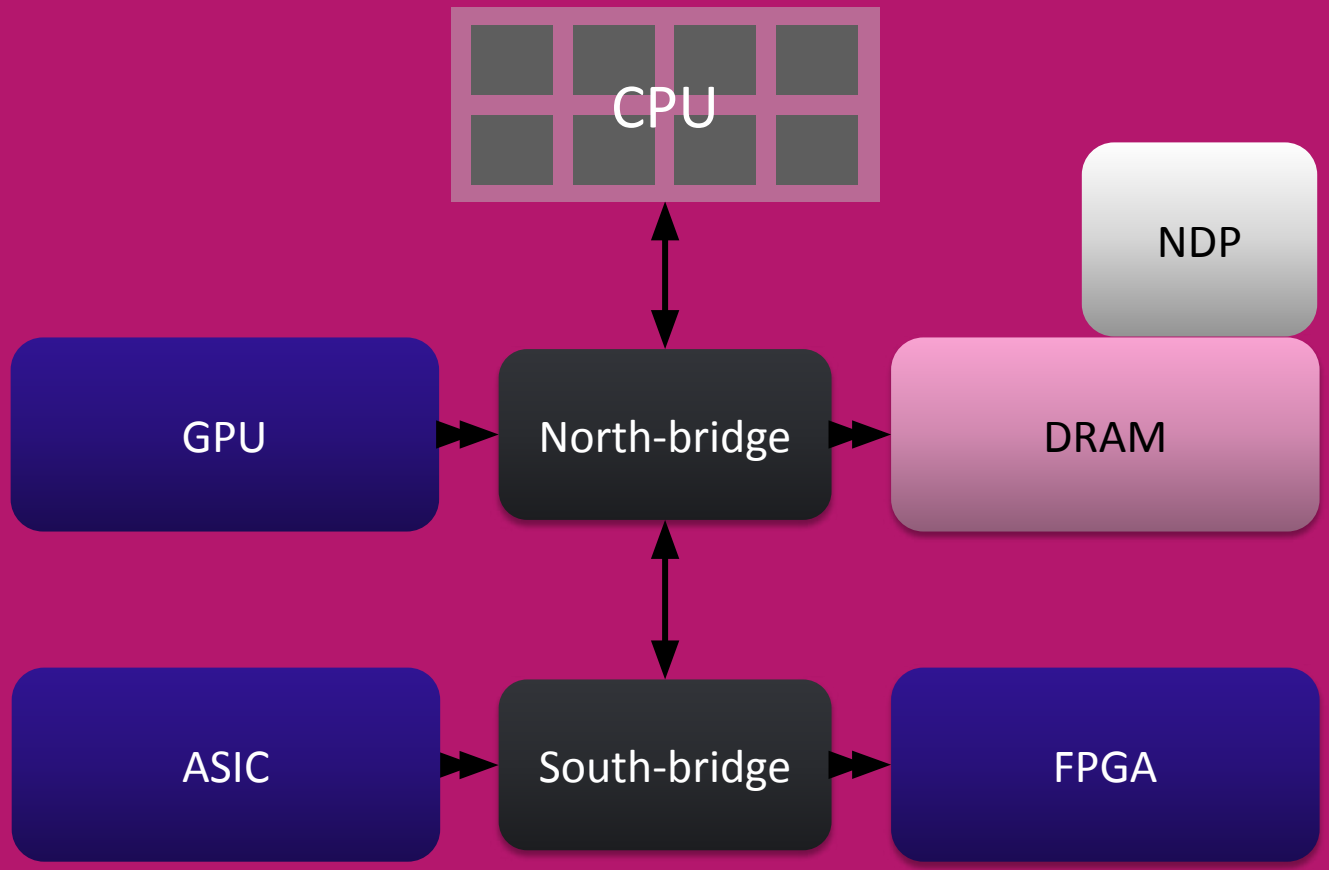
Since 1970 researchers studied advances in processor to mitigate data transfer latency:

- Pipelining
- Cache levels
- Superscalar processing
- Out-of-order execution
- Vectorization
- Multithreading and Multiprocessing
- µop Cache
- 3D-Stacked Memory

But the relative latency remains almost the same!

Research Question

Is it possible to achieve high performance with a Near-Data Processing accelerator for Machine Learning applications?


CPU
NDP
GPU
North-bridge
DRAM
ASIC
South-bridge
FPGA

Challenges

1. Methodology
   - Intrinsics-VIMA library
   - Trace-Generator
   - Simulator
2. Migration of Machine Learning applications

Challenge 1

Methodology

Intrinsics-VIMA

```
#include “../vima.hpp”

int main (int argc, char *argv[]) {
    __v32f *v_A, *v_B, *v_C, s_oper;

    v_A = (__v32f*)malloc(sizeof(__v32f) * VM64I);
    v_B = (__v32f*)malloc(sizeof(__v32f) * VM64I);
    v_C = (__v32f*)malloc(sizeof(__v32f) * VM64I);

    ...

    _vim64_fmuls(v_A, v_B, v_C);
    _vim64_fcums(v_C, s_oper);
}
```

Intrinsics-VIMA

```
#include “../vima.hpp”

int main (int argc, char *argv[]) {
    __v32f *v_A, *v_B, *v_C, s_oper;

    v_A = (__v32f*)malloc(sizeof(__v32f) * VM64I);
    v_B = (__v32f*)malloc(sizeof(__v32f) * VM64I);
    v_C = (__v32f*)malloc(sizeof(__v32f) * VM64I);

    ...

    _vim64_fmuls(v_A, v_B, v_C);
    _vim64_fcums(v_C, s_oper);
}
```

Intrinsics-VIMA

```
#include “../vima.hpp”

int main (int argc, char *argv[]) {
    __v32f *v_A, *v_B, *v_C, s_oper;

    v_A = (__v32f*)malloc(sizeof(__v32f) * VM64I);
    v_B = (__v32f*)malloc(sizeof(__v32f) * VM64I);
    v_C = (__v32f*)malloc(sizeof(__v32f) * VM64I);

    ...

    _vim64_fmuls(v_A, v_B, v_C);
    _vim64_fcums(v_C, s_oper);
}
```

Intrinsics-VIMA

```
#include “../vima.hpp”

int main (int argc, char *argv[]) {
    __v32f *v_A, *v_B, *v_C, s_oper;

    v_A = (__v32f*)malloc(sizeof(__v32f) * VM64I);
    v_B = (__v32f*)malloc(sizeof(__v32f) * VM64I);
    v_C = (__v32f*)malloc(sizeof(__v32f) * VM64I);

    ...

    _vim64_fmuls(v_A, v_B, v_C);
    _vim64_fcums(v_C, s_oper);
}
```

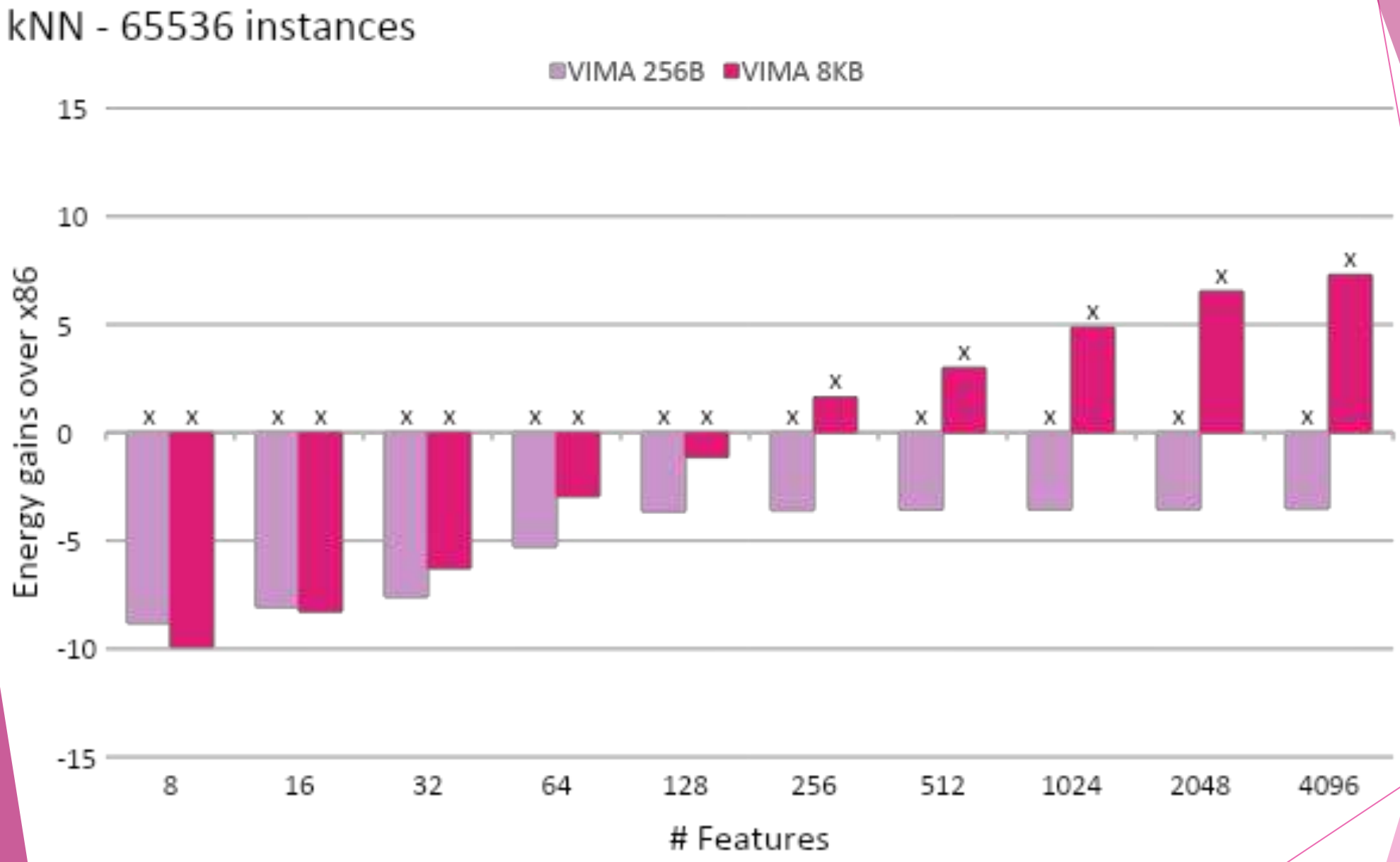
Intrinsics-VIMA

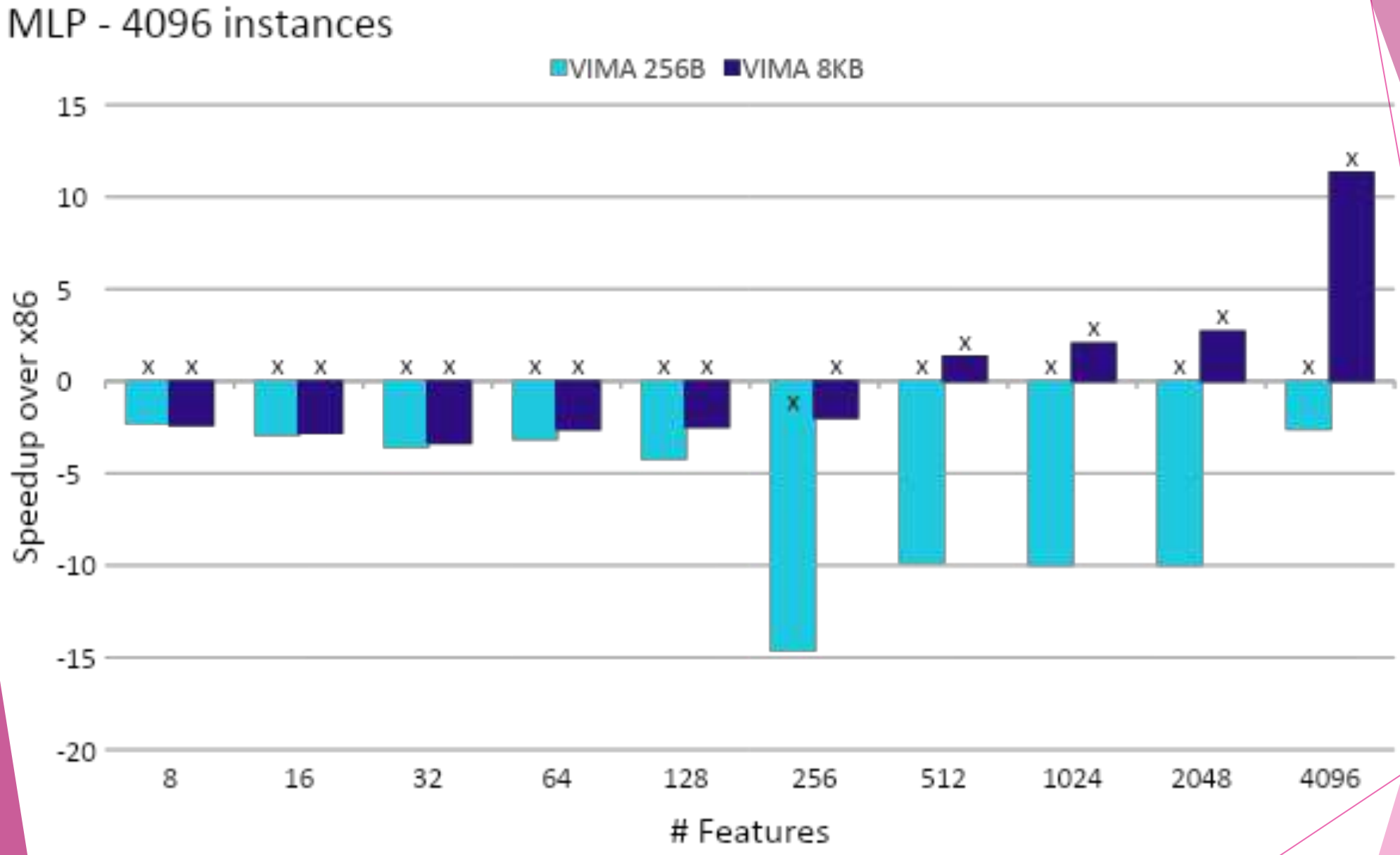
```
#include “../vima.hpp”

int main (int argc, char *argv[]) {
    __v32f *v_A, *v_B, *v_C, s_oper;

    v_A = (__v32f*)malloc(sizeof(__v32f) * VM64I);
    v_B = (__v32f*)malloc(sizeof(__v32f) * VM64I);
    v_C = (__v32f*)malloc(sizeof(__v32f) * VM64I);

    ...

    _vim64_fmulS(v_A, v_B, v_C);
    _vim64_fcumS(v_C, s_oper);
}
```

Challenge 2

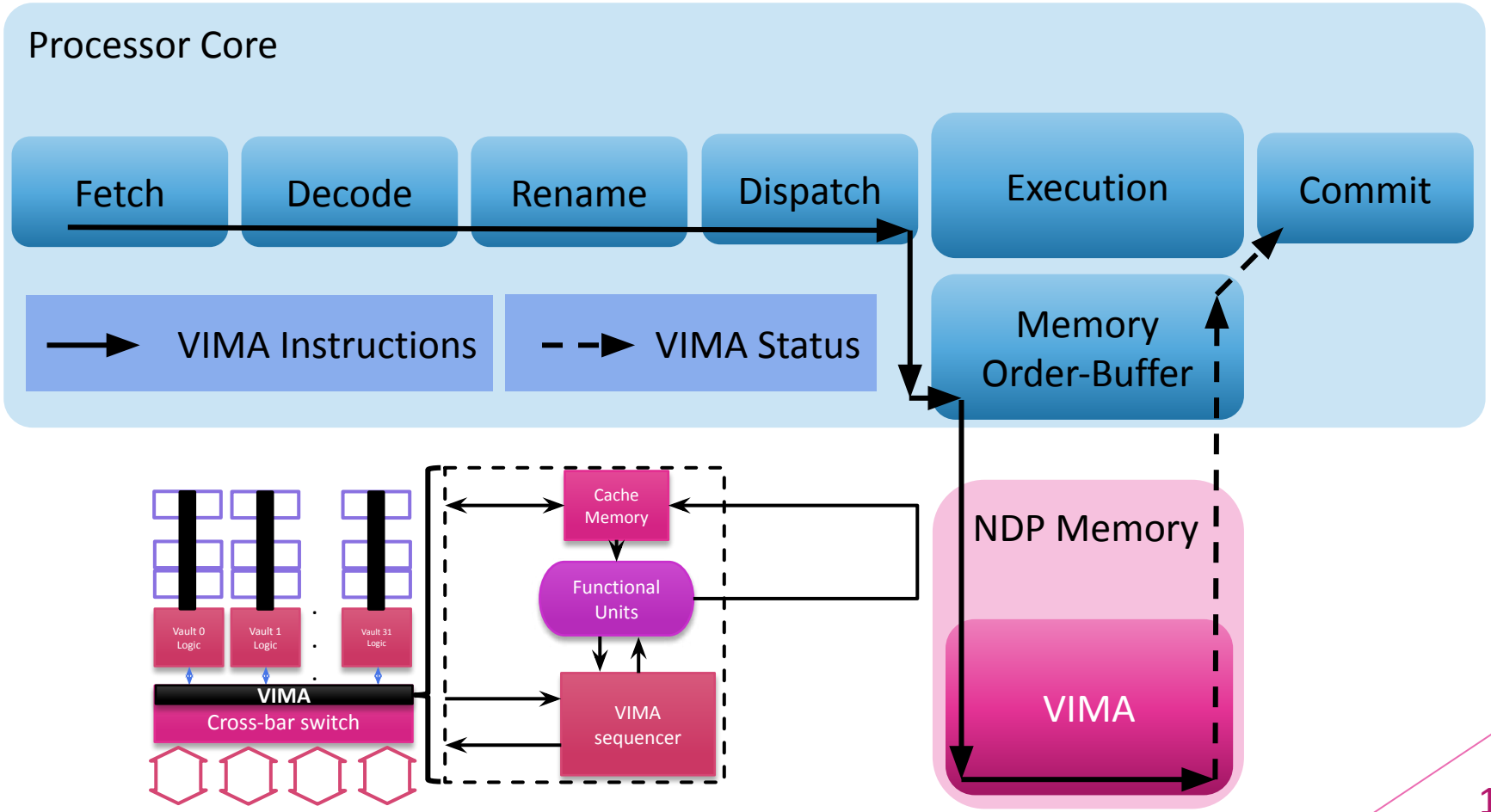
Port the algorithms to VIMA

GPU comparison


kNN - 65536 instances
VIMA 256B
VIMA 8KB
Energy gains over x86
15
10
5
0
-5
-10
-15
8
16
32
64
128
256
512
1024
2048
4096
Features


MLP - 4096 instances
VIMA 256B
VIMA 8KB
Speedup over x86
15
10
5
0
-5
-10
-15
-20
8
16
32
64
128
256
512
1024
2048
4096
Features

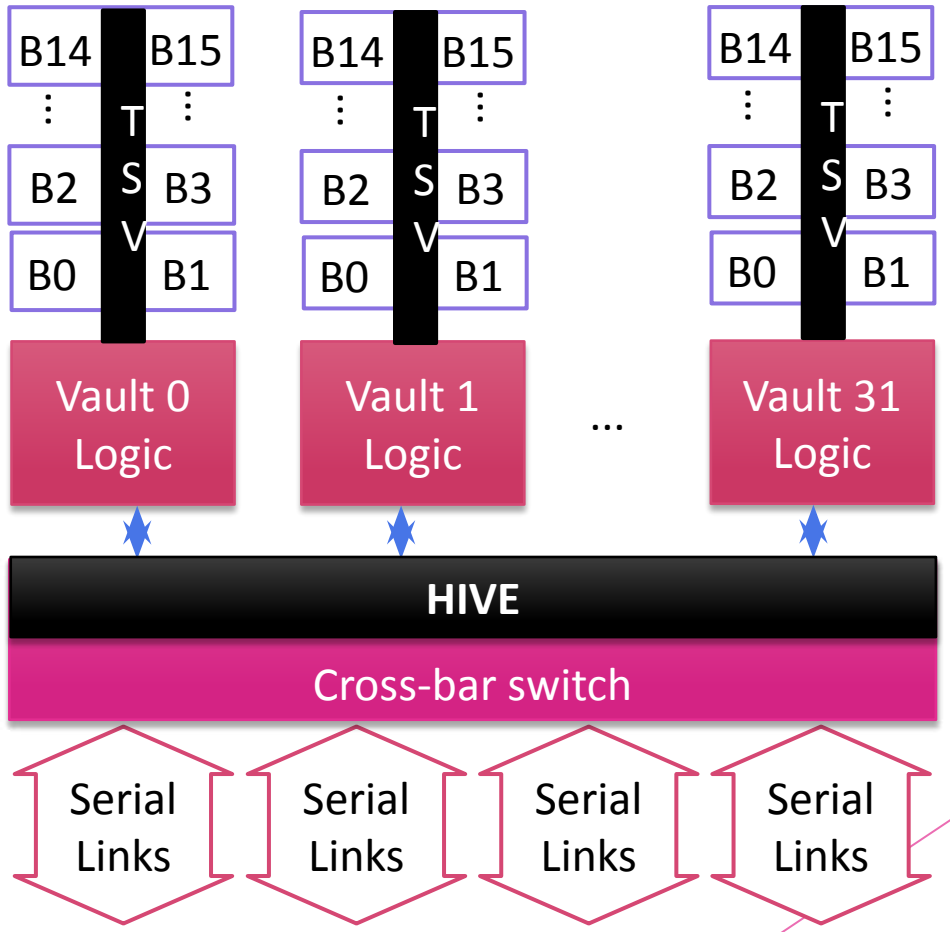
VIMA execution process



```
for (i = dim_size; i + dim_size + VSIZE < v_size; i+=VSIZE){
    _vim2K_fadds(&B[i], &A[i-dim_size-1], &B[i]); // 1º line
    _vim2K_fadds(&B[i], &A[i-dim_size], &B[i]);
    _vim2K_fadds(&B[i], &A[i-dim_size+1], &B[i]);

    _vim2K_fadds(&B[i], &A[i-1], &B[i]); // 2º line
    _vim2K_fadds(&B[i], &A[i], &B[i]);
    _vim2K_fadds(&B[i], &A[i+1], &B[i]);

    _vim2K_fadds(&B[i], &A[i+dim_size-1], &B[i]); // 3º line
    _vim2K_fadds(&B[i], &A[i+dim_size], &B[i]);
    _vim2K_fadds(&B[i], &A[i+dim_size+1], &B[i]);

    _vim2K_fmuls(&B[i], &mul[i], &B[i]);
}
```

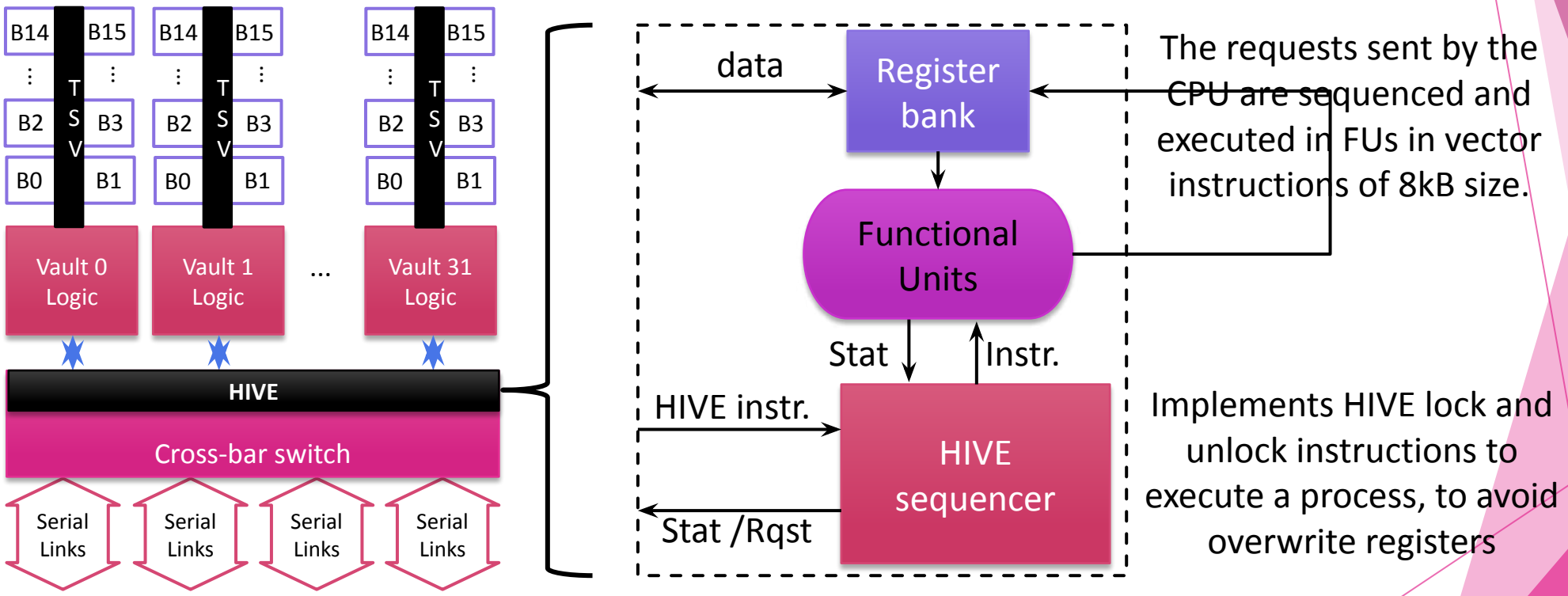
HIVE: Integrating vectorization and NDP

- HMC Instruction Vector Extensions
- A proposal to allow executing vector instructions inside a NDP module
- HIVE is a module attached in logic layer
- Ideal to streaming algorithms



Compared to HMC+SSE, HIVE+HMC gains up to 17.3x for Vector sum, 5.4x for Stencil and 4.1x for Matrix Multiplication. (Alves et al., 2016)

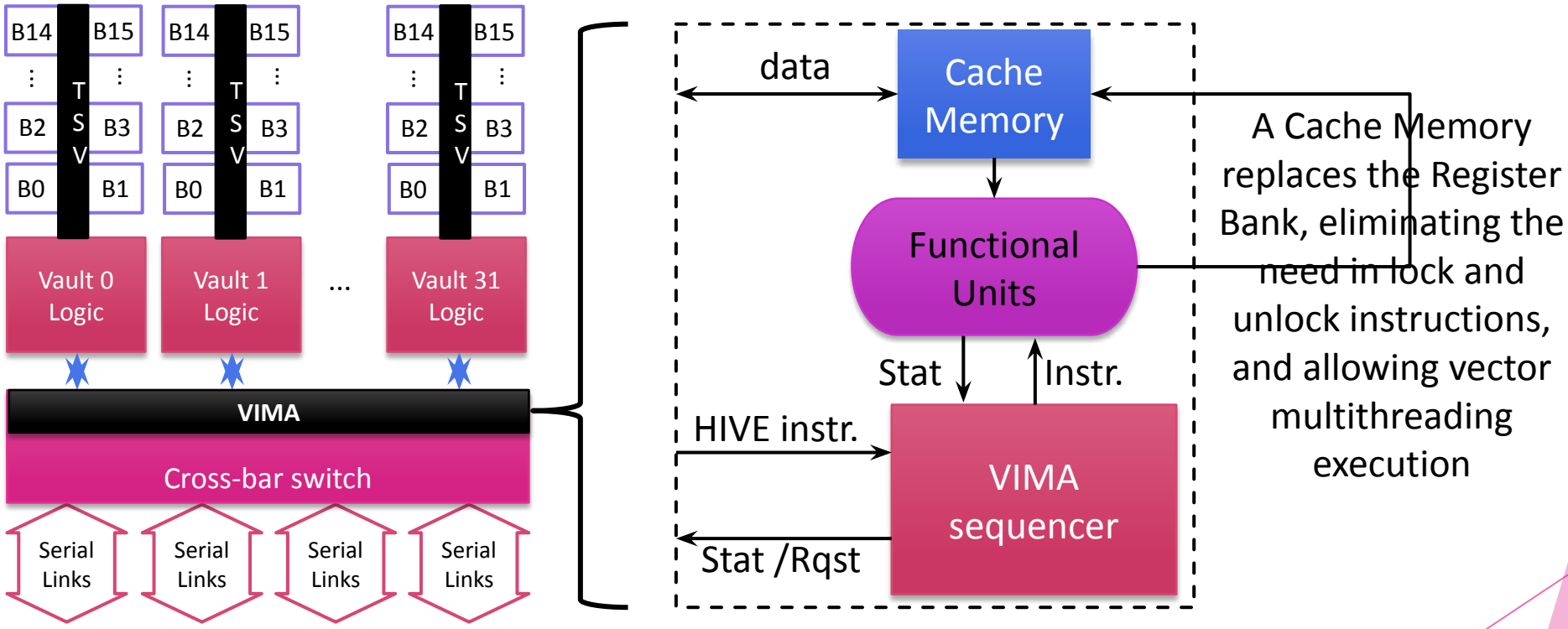
HIVE: HMC Instruction Vector Extensions



The requests sent by the CPU are sequenced and executed in FUs in vector instructions of 8kB size.

Implements HIVE lock and unlock instructions to execute a process, to avoid overwrite registers

VIMA: Vector-in-Memory Architecture



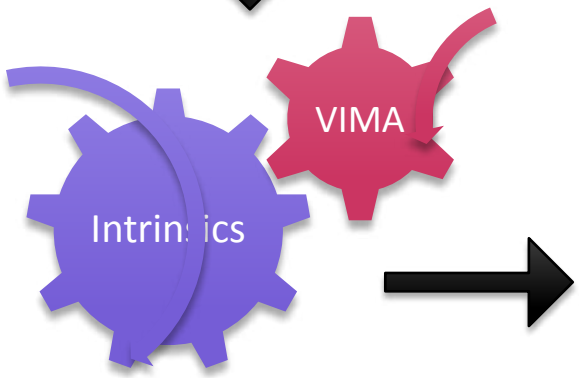
A Cache Memory replaces the Register Bank, eliminating the need in lock and unlock instructions, and allowing vector multithreading execution

Evaluation steps

```
while (x < 100){
    ...
    ...
    ...
} do
```

kNN
MLP
Convolução

Intrinsics-VIMA: a library that emulates VIMA and allows programmers to write high level code, compile, execute and debug it

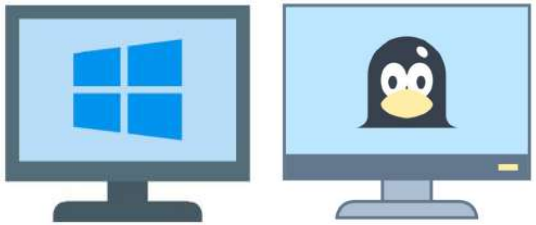


PIM code

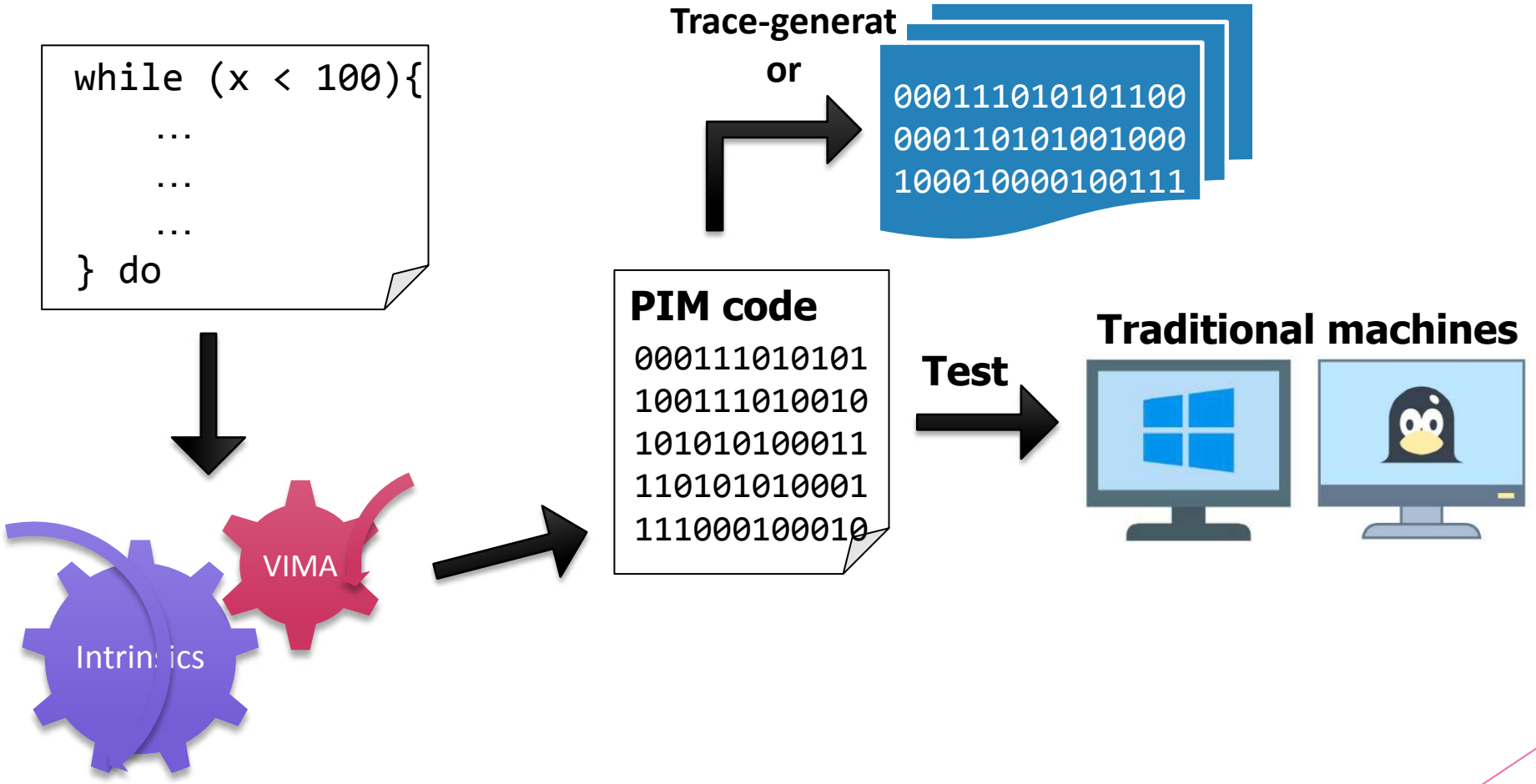
```
000111010101
100111010010
101010100011
110101010001
111000100010
```

Test

Traditional machines

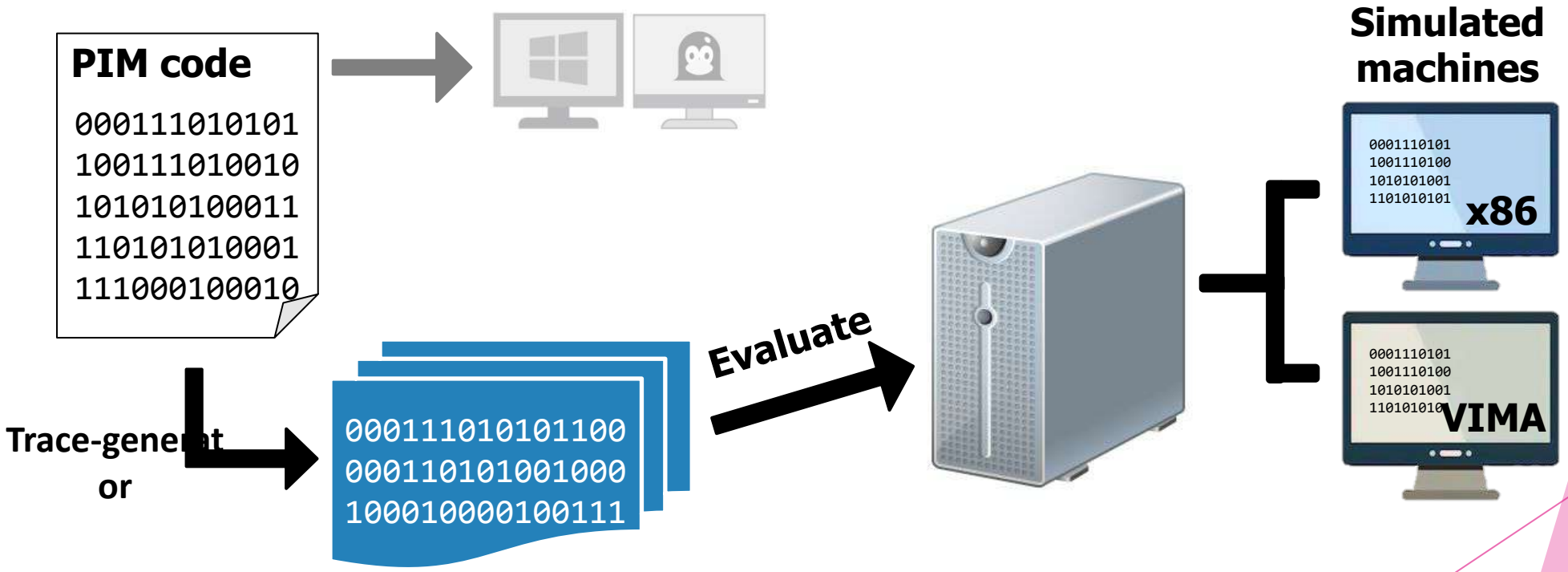


Evaluation steps



Evaluation steps

Simulation: Use Ordinary Computer Simulator (OrCS) to evaluate simulation traces



ML algorithms to port to VIMA

- **K-Nearest Neighbors:** Searches in Euclidean Space the k nearest neighbors to classify an instance
- **MultiLayer Perceptron:** Neural Network
- **Convolution:** image transformation

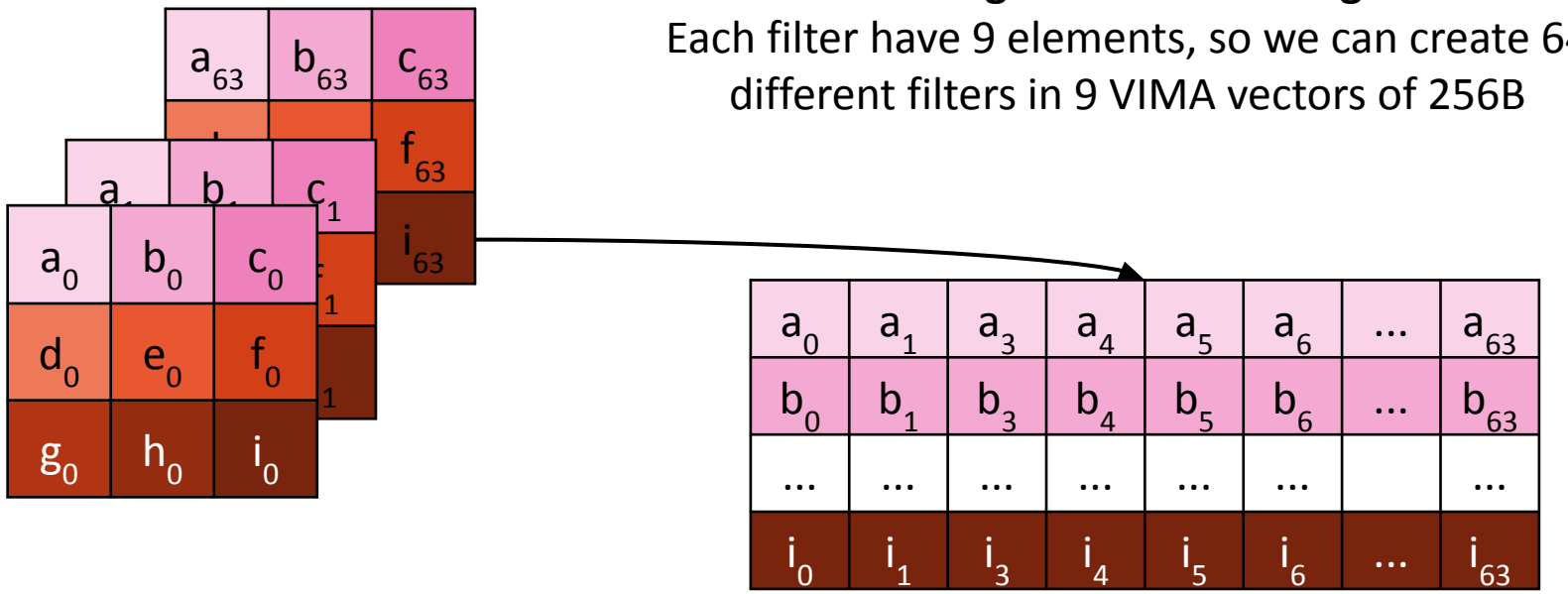
Why simulating kNN, MLP and convolution?

kNN has a quadratic complexity and low data reuse in conventional architectures for higher amount of data

MLP has a linear complexity and low data reuse in conventional architectures for higher amount of data

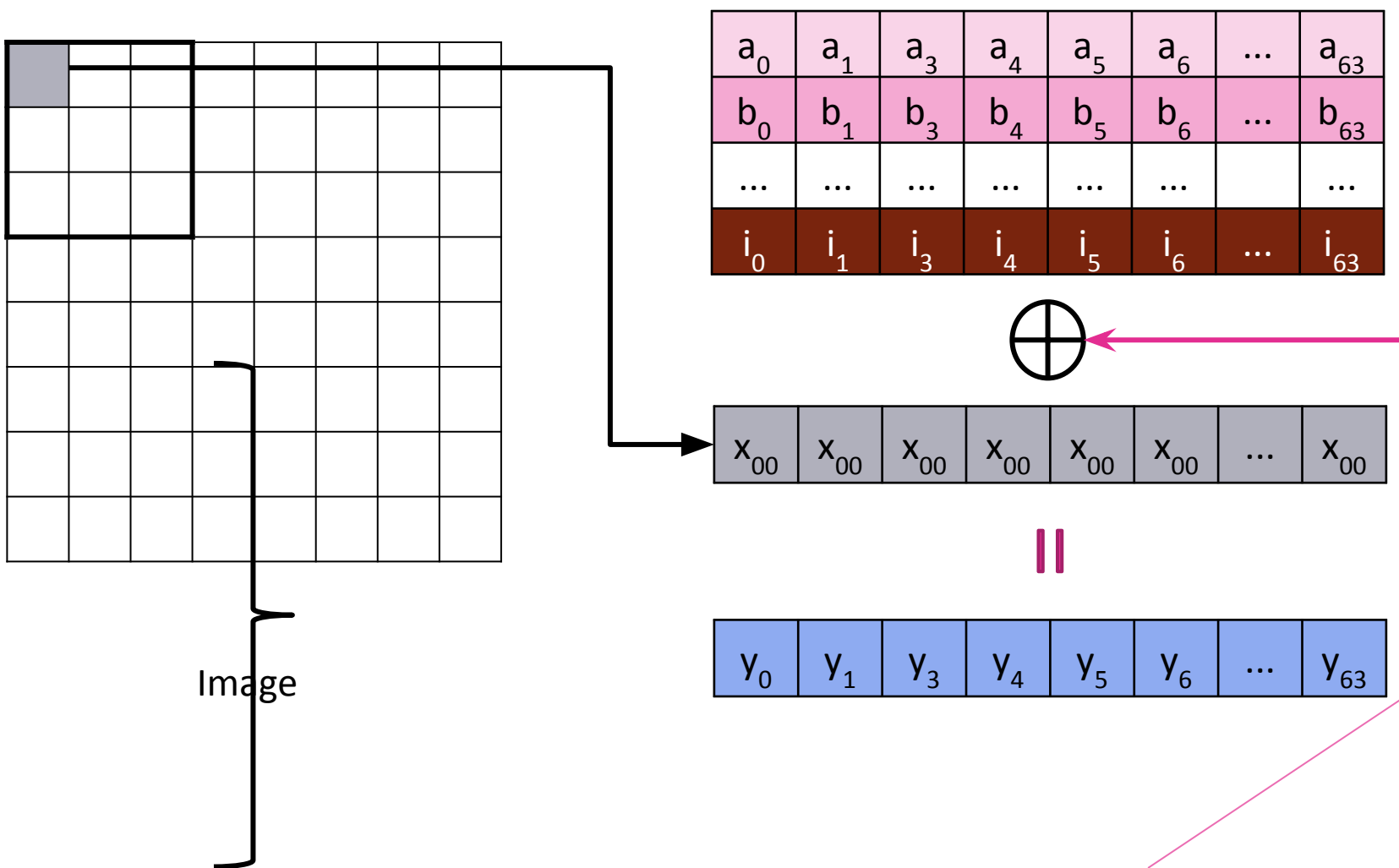
Convolution has a linear complexity and fair data reuse in conventional architectures

Convolutional Layer:
How it can be implemented in VIMA?



We can change the filters configuration:
Each filter have 9 elements, so we can create 64 different filters in 9 VIMA vectors of 256B

Convolutional Layer: How it can be implemented in VIMA?



Broadcasting each image pixel to a VIMA vector and operating with the filters...

The 64 activation maps will be formed by accumulation

Convolutional Layer: Comparison

Pros

- In VIMA the image is iterated only once compared to the 64 iterations on CPU
- It executes Dot Product 64 times less

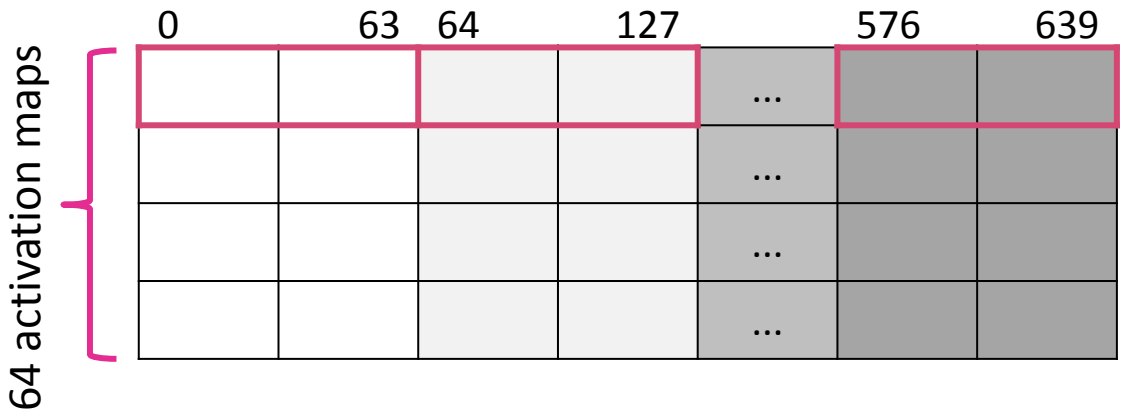
Cons

- The activation maps storage is sparse, resulting in low computational performance

Activation Layer:
How it can be implemented in VIMA?

In the CPU the instructions execute scalar values

Apply non-linearity to the activation maps, so it zeroes every negative value



Pooling Layer:
Why it can not be implemented in VIMA?

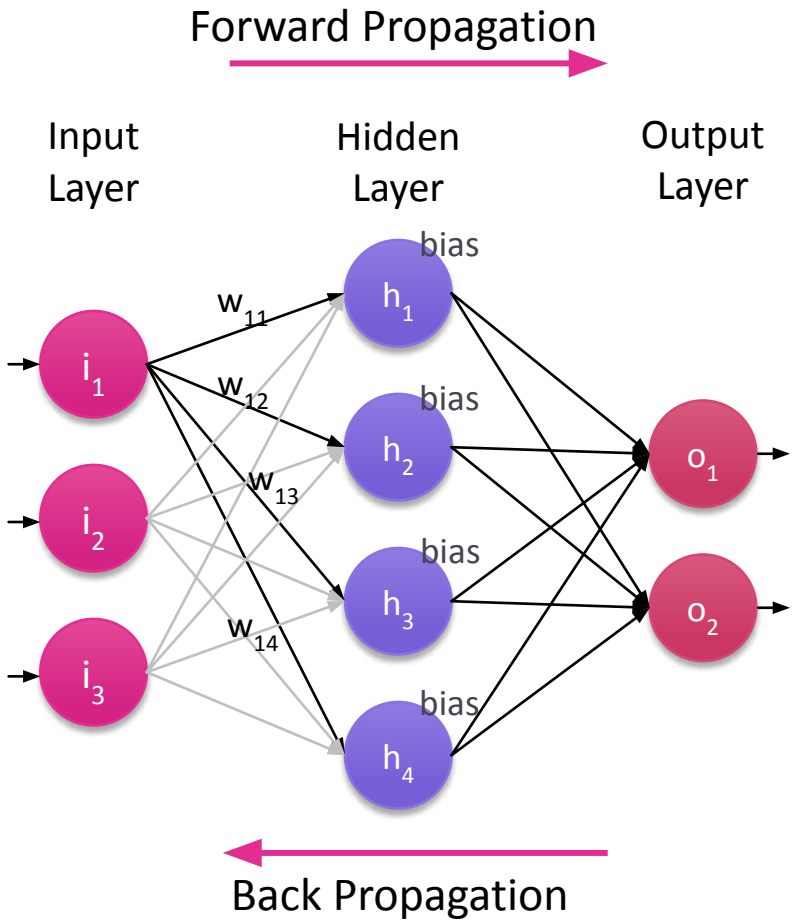
- Applies MaxPooling of 2x2 window
- Data can be reconfigured in vectors to apply Map Reduce operation
  - However, it is not a Reduced Instruction Set Computer (RISC) instruction, emented

8	0	22	3
5	15	9	6
7	14	7	14
2	1	32	5

15	22
14	32

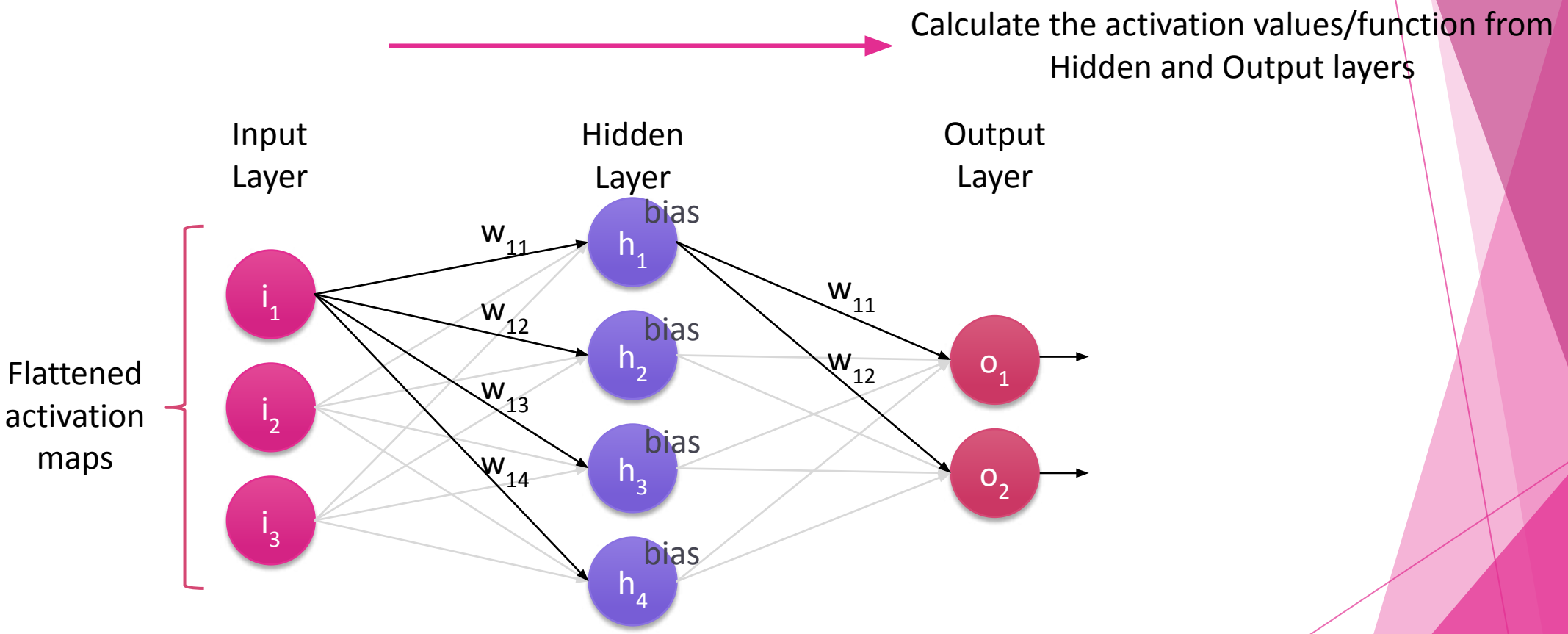
Reduces the dimension while keeps important information linked to the activation map, and controll overfitting and

Fully Connected Layer

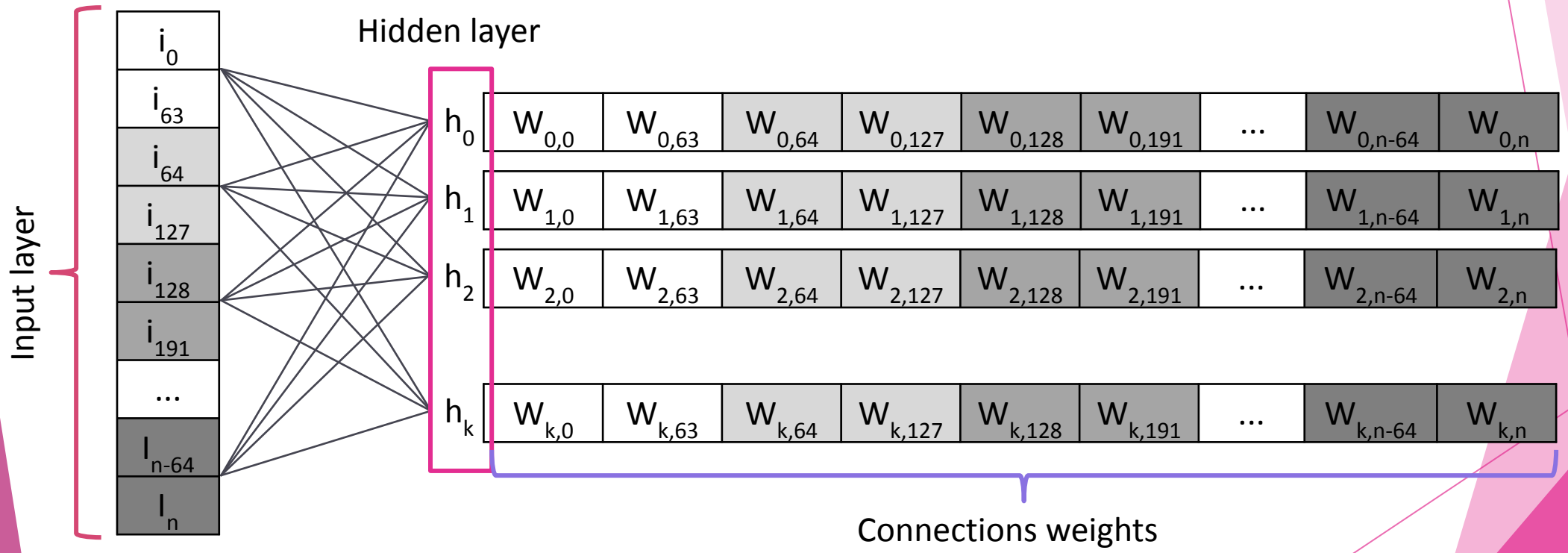


- Forward Propagation:
  - Calculates the neurons activation values
  - Error calculation
- Back Propagation:
  - Updates connection weights first and second derivatives

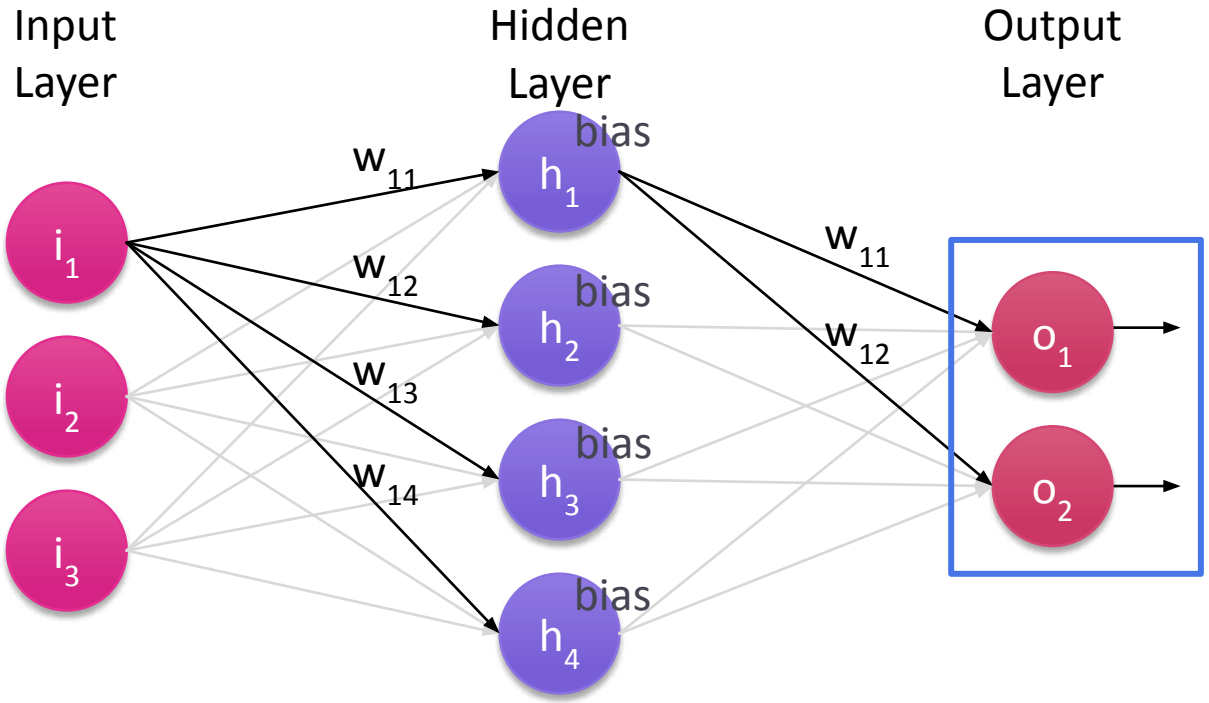
Forward Propagation



Forward Propagation:
How it can be implemented in VIMA?



Error calculation

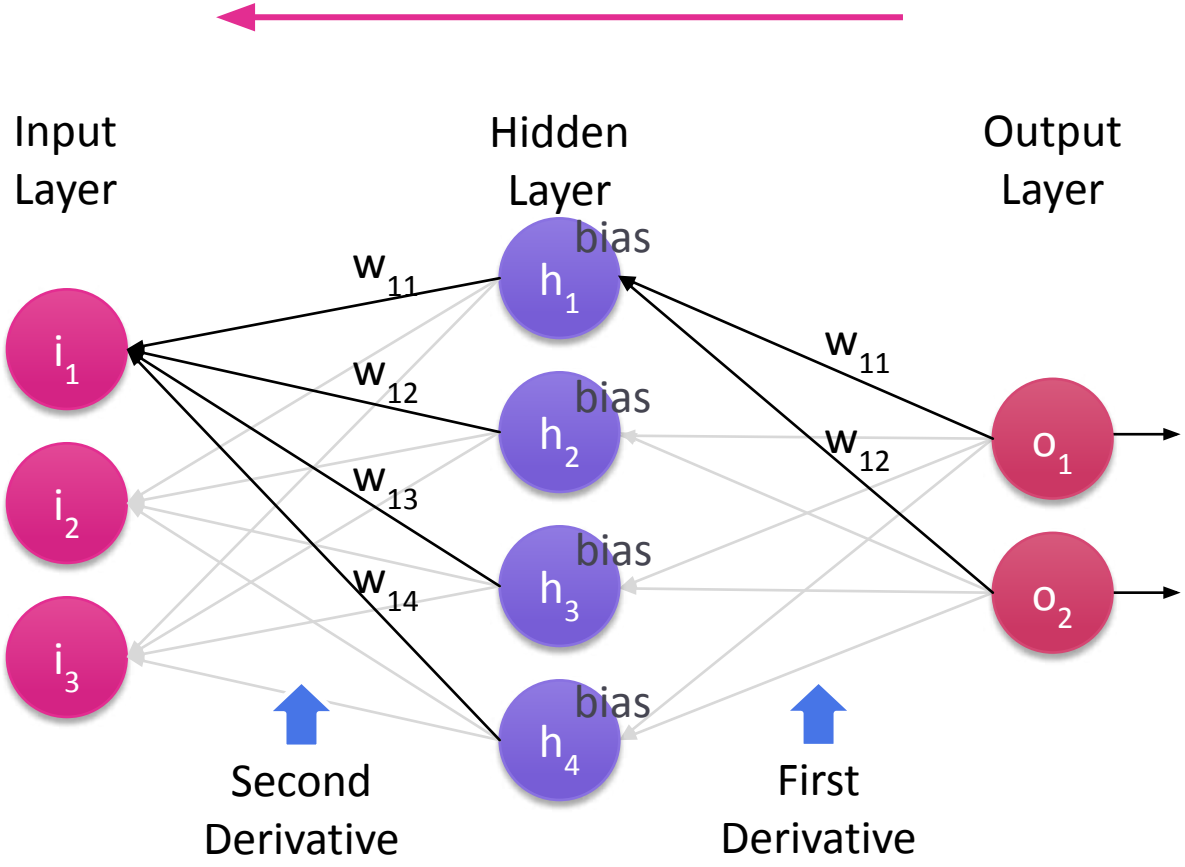


Error calculation: Uses Mean Squared Error (MSE) to evaluate the obtained value compared to the expect one.

We are using MNIST, a dataset of handwritten digits, and our output layer has only 10 neurons to allocate in a 256B VIMA vector

Back Propagation

Except for the output layer, the rest of the Back Propagation can be vectorized with VIMA vectors



Update weights using the MSE result.

Main contributions

1. Intrinsics-VIMA: a library that allows programmers to write code in high level language and evaluate it
2. Tracer-generator: a tool to port code to VIMA in easy way
3. VIMA in OrCS: the implementation of VIMA in OrCS to evaluate simulation traces
4. Migration of ML code to VIMA: an implementation specific to VIMA

Schedule

	2019				2020	
Activities	AGO	SEP	OCT	NOV	DEC	JAN	FEV
Trace-generator	●						
Dynamic cache		●					
VIMA in OrCS		●	●				
Adjustments in kNN and CNN				●			
Algorithms tests				●			
SImulate traces with VIMA in OrCS					●		
Analyze the results					●		
Write and submit a paper						●	
Write dissertation and presentation						●	●

Related Work

Paper	General-Purpose	Vector execution	PIM	Simple FU	Easy to simulate
Cadambi et al. (2010)	●	●	●		
Xu et al. (2015)	●		●		●
Gao et al. (2015)	●		●		●
Ahn et al. (2016)	●		●		
Li et al. (2017)	●	●		●	●
Oliveira et al. (2017b)		●	●	●	●
Gao et al. (2017)		●	●		
Thottethodi et al. (2018)	●	●	●		●
Azarkhish et al. (2018)		●	●		●
VIMA	●	●	●	●	●

Related Work

Paper	General-Purpose	Vector execution	PIM	Simple FU	Easy to simulate
Gao et al. (2018)				●	
Schuiki et al. (2018)			●		●
Liu et al. (2018)			●		
Deng et al. (2018)				●	●
Ganguly et al. (2018)			●		●
Sim et al. (2018)				●	
Min et al. (2019)			●		●
de Lima et al. (2019)		●	●		●
Deng et al. (2019)		●		●	●
VIMA	●	●	●	●	●

References

- Alves, M. A., Diener, M., Santos, P. C., and Carro, L. (2016). Large vector extensions inside the hmc. In 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), pages 1249–1254. IEEE.
- Boroumand A., Ghose S., Kim Y., Ausavarungnirun R., Shiu E., Thakur R., Kim D., Kuusela A., Knies A., Ranganathan P., and Mutlu O. (2018). Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks. In Proceedings of the23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pages 316-331, ACM.
- Cadambi, S., Majumdar, A., Becchi, M., Chakradhar, S., and Graf, H. P. (2010). A programmable parallel accelerator for learning and classification. In Proceedings of the 19th international conference on Parallel architectures and compilation techniques, pages 273–284. ACM

References

- Thottethodi, M., Vijaykumar, T., et al. (2018). Millipede: Die-stacked memory optimizations for big data machine learning analytics. In 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pages 160–171. IEEE.
- Li, S., Niu, D., Malladi, K. T., Zheng, H., Brennan, B., and Xie, Y. (2017). Drisa: A dram-based reconfigurable in-situ accelerator. In Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, pages 288–301. ACM.
- Ahn, J., Hong, S., Yoo, S., Mutlu, O., and Choi, K. (2016). A scalable processing-in-memory accelerator for parallel graph processing. ACM SIGARCH Computer Architecture News, 43(3):105–117.

References

- Gao, M., Pu, J., Yang, X., Horowitz, M., and Kozyrakis, C. (2017). Tetris: Scalable and efficient neural network acceleration with 3d memory. ACM SIGOPS Operating Systems Review, 51(2):751–764.
- Gao, D., Shen, T., and Zhuo, C. (2018). A design framework for processing-in-memory accelerator. In 2018 ACM/IEEE International Workshop on System Level Interconnect Prediction (SLIP), pages 1–6. IEEE
- Schuiki, F., Schaffner, M., Gürkaynak, F. K., and Benini, L. (2018). A scalable near-memory architecture for training deep neural networks on large in-memory datasets. arXiv preprint arXiv:1803.04783.
- Xu, L., Zhang, D. P., and Jayasena, N. (2015). Scaling deep learning on multiple in-memory processors. In Proceedings of the 3rd Workshop on Near-Data Processing.

References

- Gao, M., Ayers, G., and Kozyrakis, C. (2015). Practical near-data processing for in-memory analytics frameworks. In Parallel Architecture and Compilation (PACT), 2015 International Conference on, pages 113–124. IEEE.
- Oliveira, G. F., Santos, P. C., Alves, M. A., and Carro, L. (2017b). Nim: An hmc-based machine for neuron computation. In International Symposium on Applied Reconfigurable Computing, pages 28–35. Springer.
- Azarkhish, E., Rossi, D., Loi, I., and Benini, L. (2018). Neurostream: Scalable and energy efficient deep learning with smart memory cubes. IEEE transactions on Parallel & Distributed Systems, pages 1–1.
- Liu, J., Zhao, H., Ogleari, M. A., Li, D., and Zhao, J. (2018). Processing-in-memory for energy-efficient neural network training: A heterogeneous approach. In 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 655–668. IEEE.

References

- Min, C., Mao, J., Li, H., and Chen, Y. (2019). Neuralhmc: an efficient hmc-based accelerator for deep neural networks. In Proceedings of the 24th Asia and South Pacific Design Automation Conference, pages 394–399. ACM.
- Deng, Q., Jiang, L., Zhang, Y., Zhang, M., and Yang, J. (2018). Dracc: a dram based accelerator for accurate cnn inference. In Proceedings of the 55th Annual Design Automation Conference, page 168. ACM.
- Ganguly, A., Singh, V., Muralidhar, R., and Fujita, M. (2018). Memory-system requirements for convolutional neural networks. In Proceedings of the International Symposium on Memory Systems, pages 291–197. ACM.
- Sim, J., Seol, H., and Kim, L.-S. (2018). Nid: processing binary convolutional neural network in commodity dram. In 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pages 1–8. IEEE.

References

- de Lima, J. P. C., Santos, P. C., de Moura, R. F., Alves, M. A., Beck, A. C., and Carro, L. (2019). Exploiting reconfigurable vector processing for energy-efficient computation in 3d-stacked memories. In International Symposium on Applied Reconfigurable Computing, pages 262–276. Springer.
- Deng, Q., Zhang, Y., Zhang, M., and Yang, J. (2019). Lacc: Exploiting lookup table-based fast and accurate vector multiplication in dram-based cnn accelerator. In Proceedings of the 56th Annual Design Automation Conference, page 128. ACM.

Porting Machine Learning Algorithms to Vector-In-Memory Architecture

Student: Aline Santana Cordeiro

Advisor: Prof. Marco Zanata Alves

Master Qualification – Federal University of Paraná

Thank you!

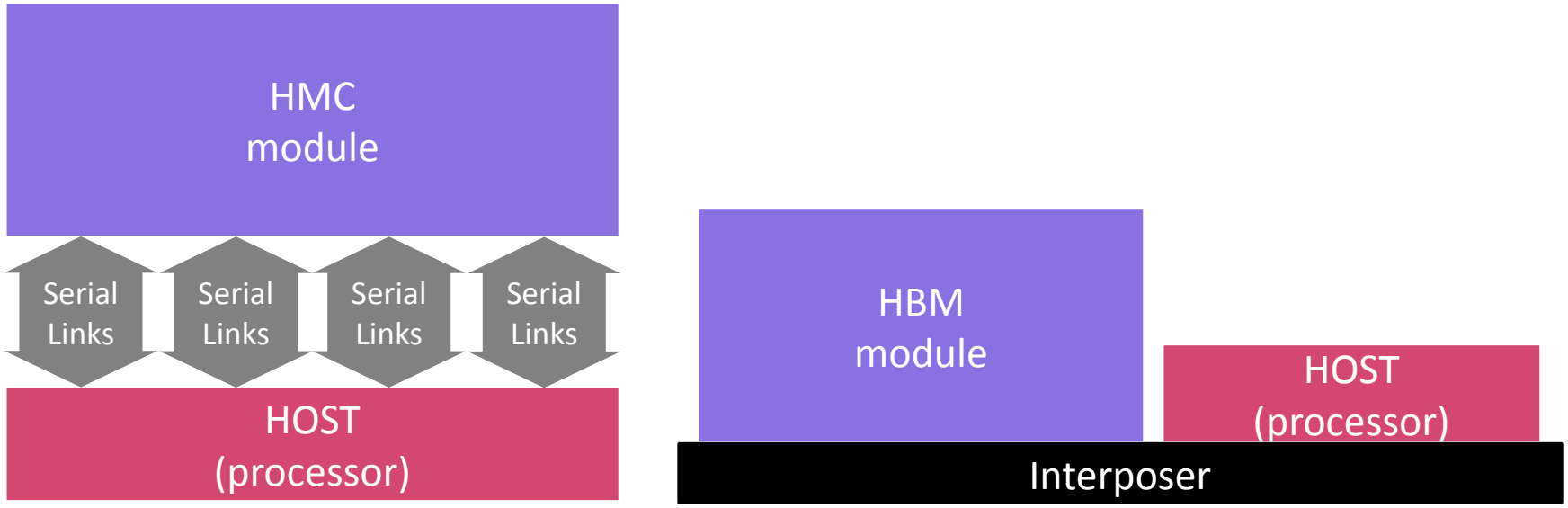
Memory bandwidth and latency

Architecture	Bandwidth	Latency (cycles)		
		L1 Cache	L2 Cache	Global Memory
CPU – Intel i7 7th	79 GB/s	5	14	70 + 50ns
GPU – GTX 980	112GB/s	-	-	368
GPU – Volta V100	900 GB/s	28	193	400
HBM2	256 GB/s	-	-	16

- Volkov, V. (2016). Understanding latency hiding on gpus. Doctoral dissertation, UC Berkeley.
- https://www.renesas.com/us/en/doc/products/memory/r10ds0281ej0001_memory.pdf
- Jia, Z., Maggioni, M., Staiger, B., & Scarpazza, D. P. (2018). Dissecting the nvidia volta gpu architecture via microbenchmarking. *arXiv preprint arXiv:1804.06826*.

Processing-in-Memory

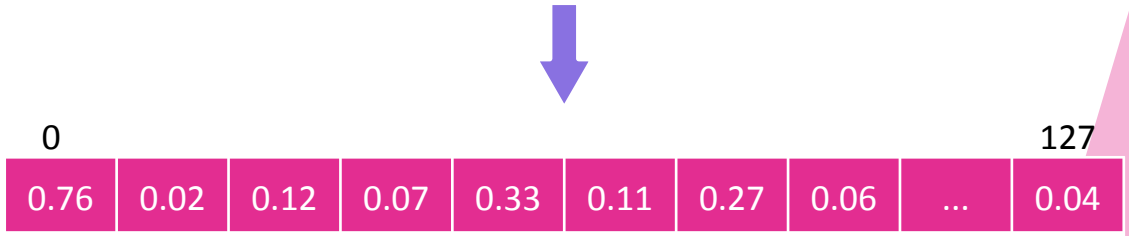
- The main difference between HBM and HMC is the way how it communicates with host processor:



Internet Movie Database (IMDb)

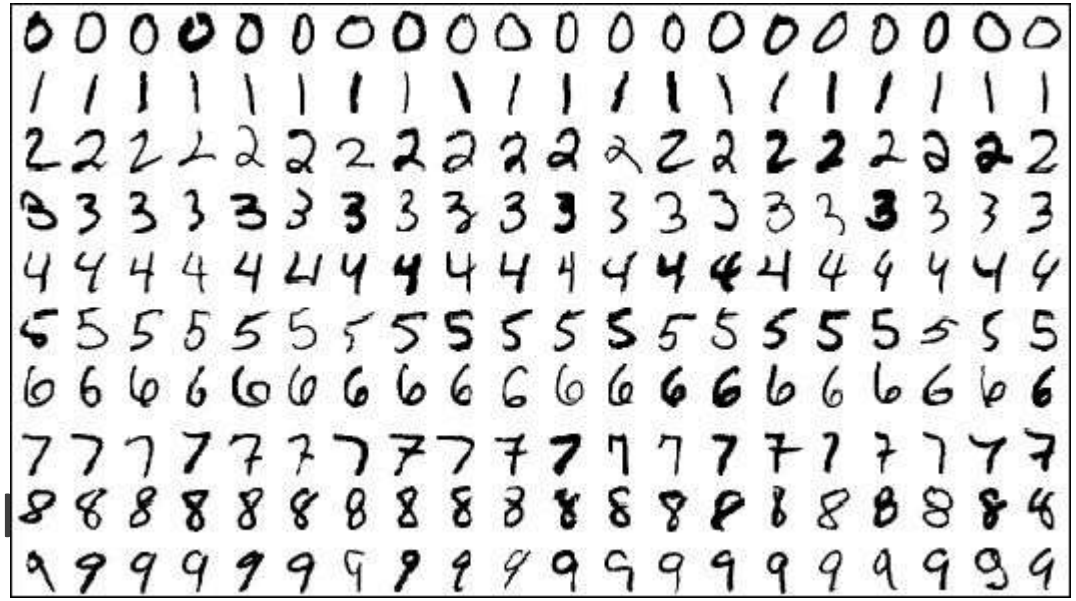
- A benchmark for sentiment analysis containing movie reviews
- 100,000 movie reviews:
  - Each review is a labeled instance
  - Labels: *pos*, *neg*, *unsup*
- Word2vec transforms reviews into feature vectors:
  - Vectors with 128 features

a pure reality bytes film. Fragile, beautiful and amazing first film of the director. Represented Spain on the Berlinale 2002. Some people has compared the grammar of the film with Almodovar's films...Well, that shouldn't be a problem... || 1



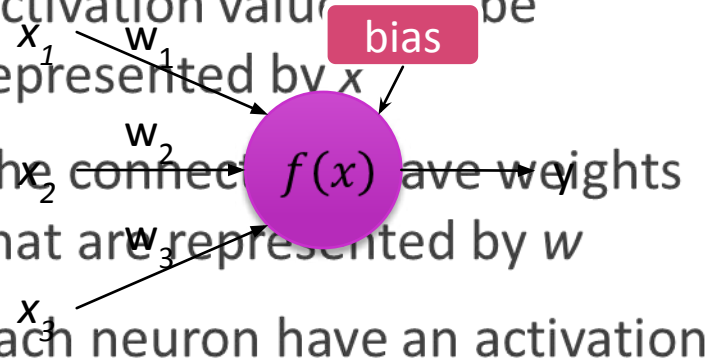
MNIST Database

- Handwritten digits:
  - 60,000 trainning images
  - 10,000 test images
- Images of 28x28 in grayscal



Fully Connected Layer: Neuron

-
- Perceptron/Neuron
- Activation values can be represented by x
- The connections have weights that are represented by w
- Each neuron have an activation value x and an activation function $f(x)$



Training and test steps