NANYANG TECHNOLOGICAL UNIVERSITY

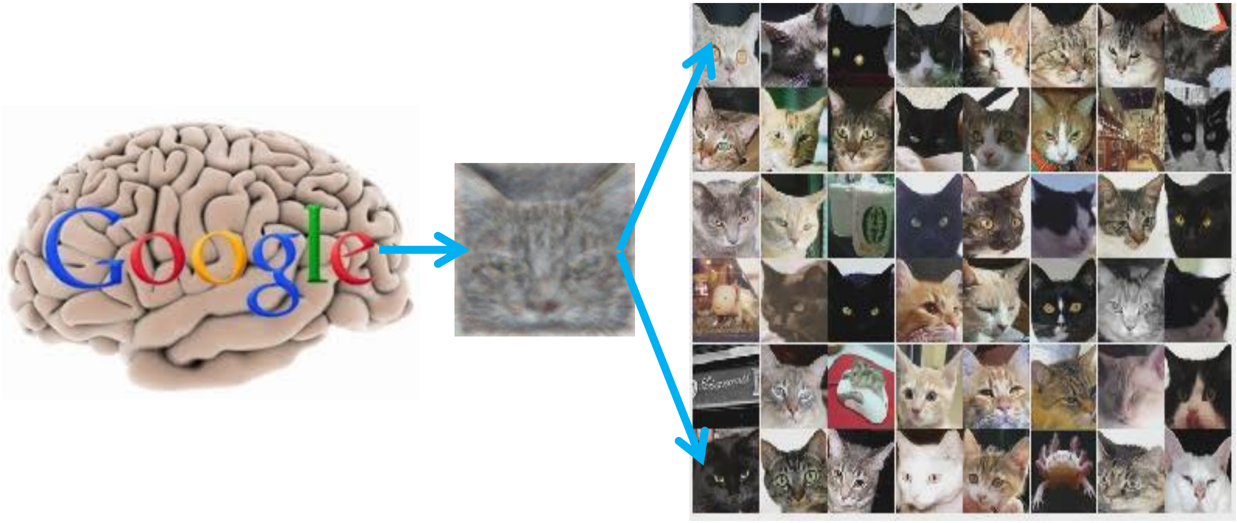
# Energy Efficient In-Memory Machine Learning for Data Intensive Image-Processing by Non-volatile Domain-Wall Memory

Hao Yu[1], Yuhao Wang[1], Shuai Chen[1], Wei Fei[1],
Chuliang Weng[2], Junfeng Zhao[2] and Zhulin Wei[2]
[1]School of Electrical and Electronic Engineering,
Nanyang Technological University, Singapore
[2]Huawei Shannon Laboratory, China
http://www.ntucmosetgp.net

# Machine Learning for Image Recognition

*“We took an artificial neural network and spread the computation across 16,000 of our CPU cores (in our data centers), and trained models with more than 1 billion connections.”* ***-- Google brain team***

## A Cat Neuron

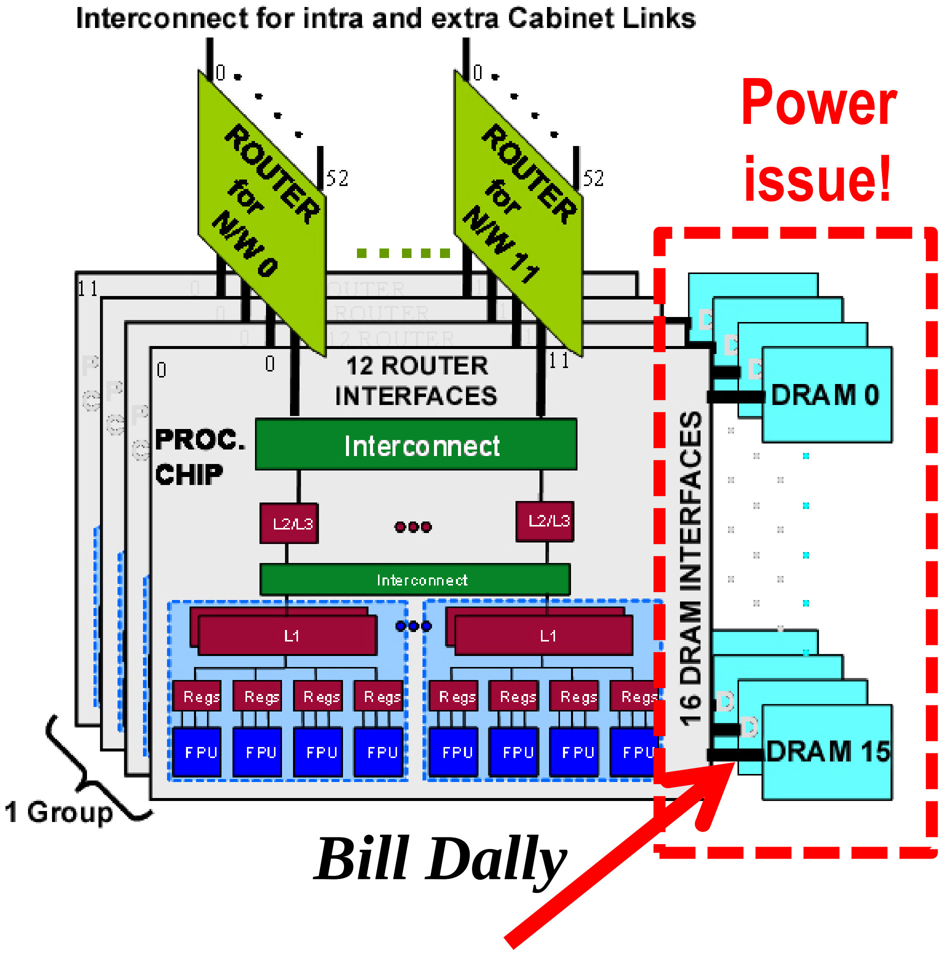


*“One of the neurons in the artificial neural network, trained from still frames from unlabeled YouTube videos, learned to detect cats.”*

# Big-data Center at Exascale

- 1 **Core** = **Microprocessor** (=6 Giga Flops @1.5GHz)
  - 4 FPUs + RegFiles
- 1 **Chip** = *742* Cores (=4.5 Tera Flops/s)
  - 213 MB of L1 I&D + 93 MB of L2
- 1 **Node** = 1 Chip + 16 **DRAMs** (16GB)
- 1 **Group** = 12 Nodes + 12 **Routers** (=54Tera Flops/s)
- 1 **Rack** = 32 Groups (=1.7 Peta Flops/s)
  - 384 nodes / rack
- 1 **Data Center** (=1 Exa Flops/s)
  - 3.6EB of Disk Storage
  - 3.6PB = 0.0036 bytes/flops
  - 583 Racks

Thousand cores in big memory



Power issue!

***Bill Dally***

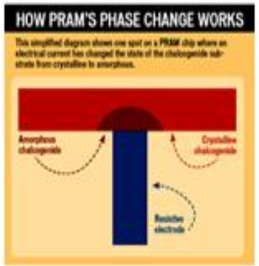
Bandwidth issue!

**100Gbps bandwidth with 68MW power**

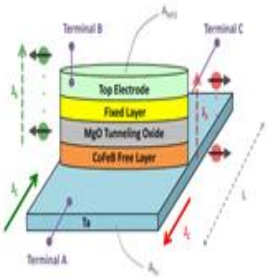
# Nonvolatile Memory Device

1. No-volatile state
2. No leakage power consumption
3. Small overhead between on/off switching
4. Universal memory for logic-in-memory

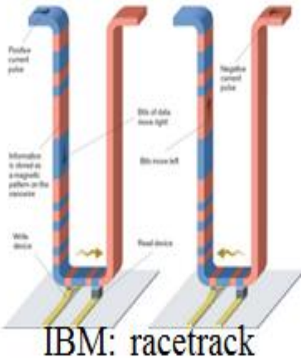
HP:memirstor



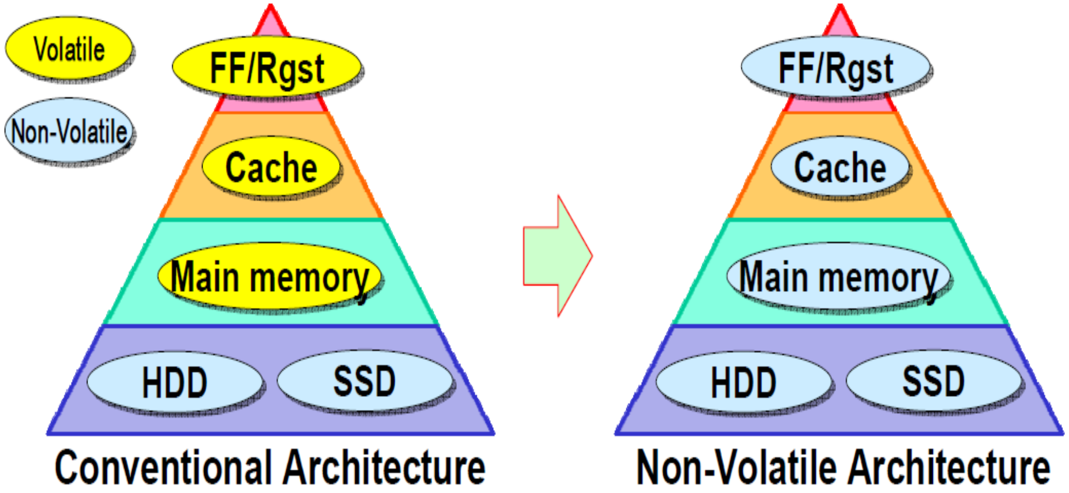
PCM



STT-MTJ



IBM: racetrack

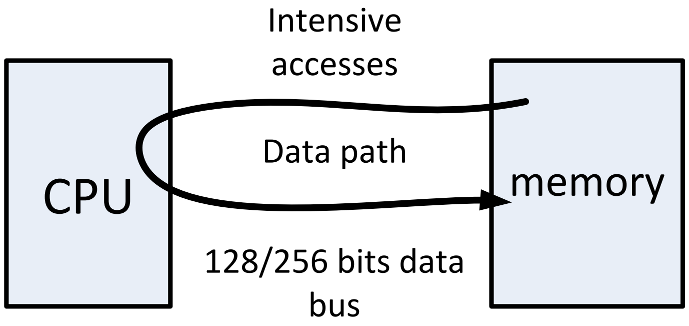


T.Kawahara, Hitachi

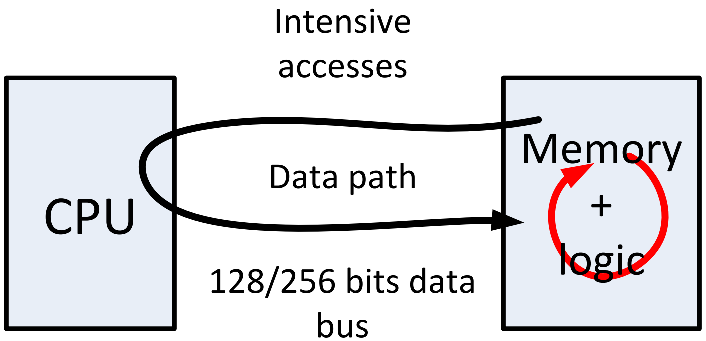
**Power issue**

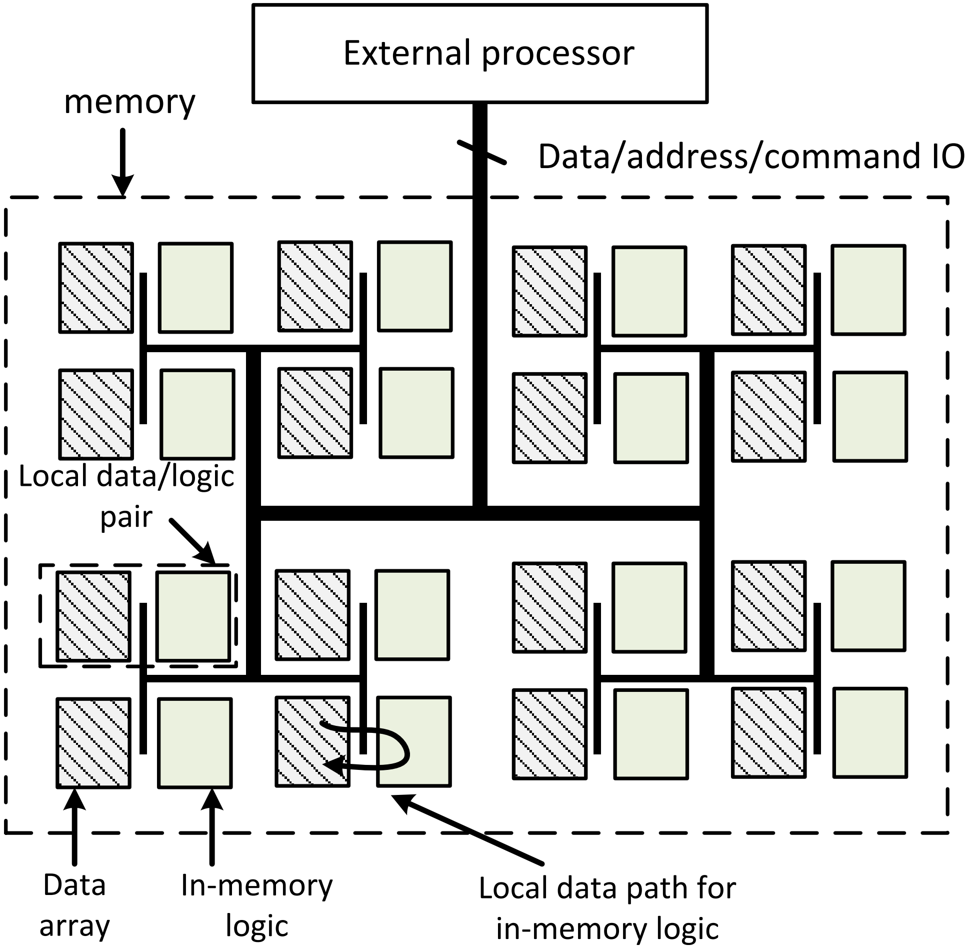
# In-memory Computing Architecture

## Conventional



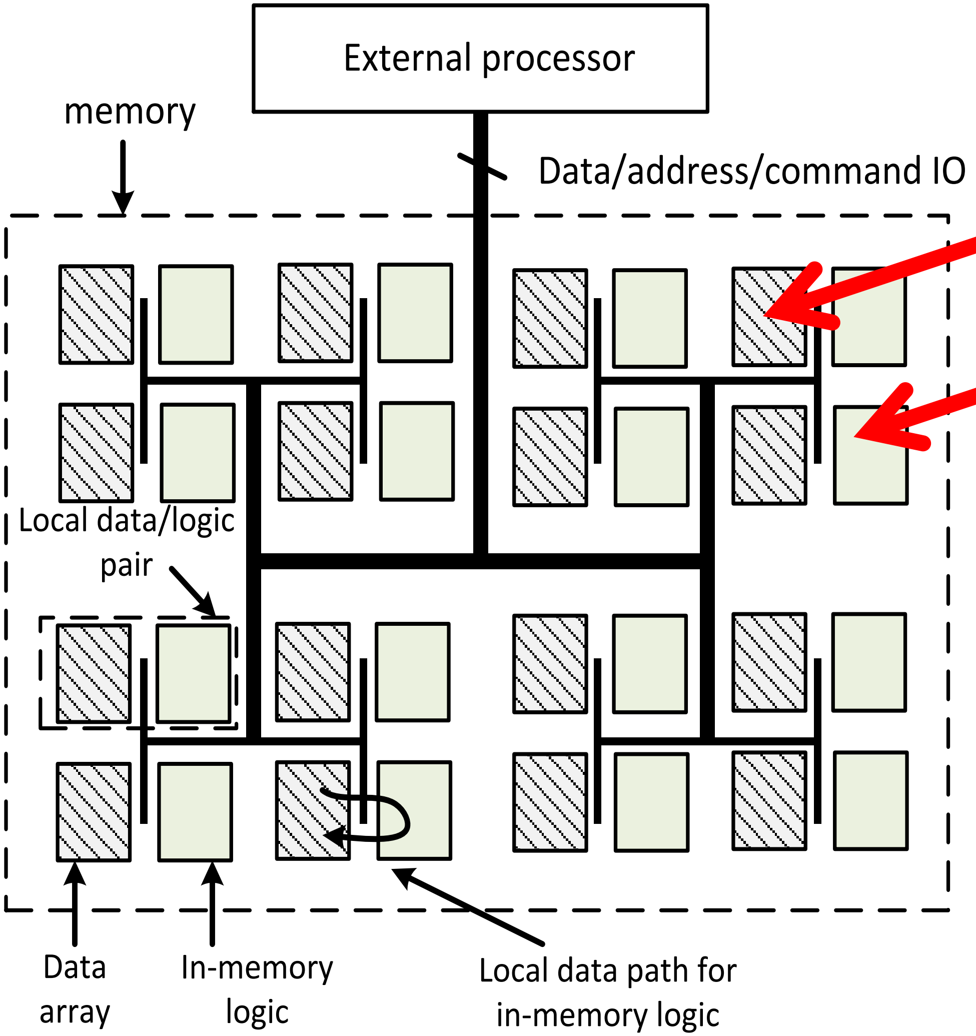
## In-memory architecture





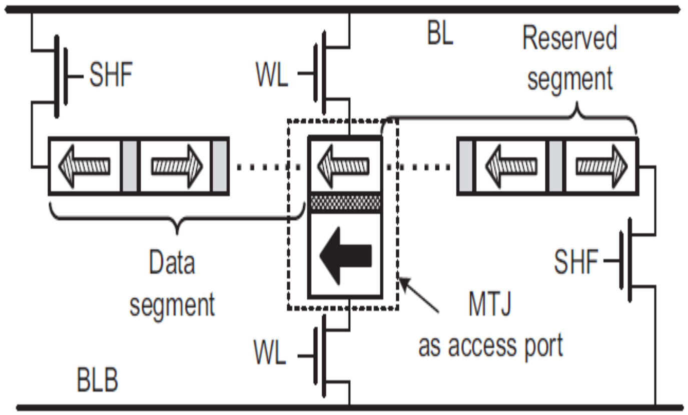
**Bandwidth issue**

# Non-volatile In-memory Computing



Data array: non-volatile

In-memory logic: non-volatile?

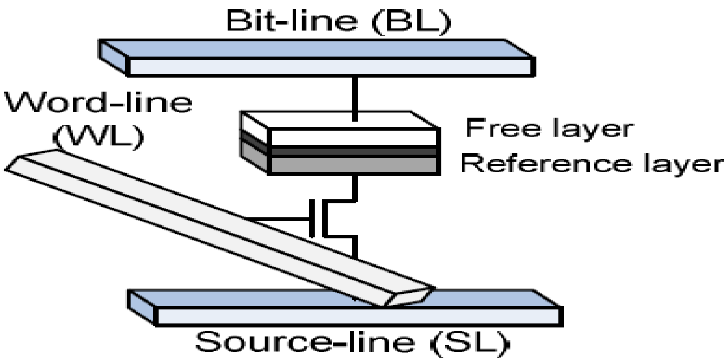


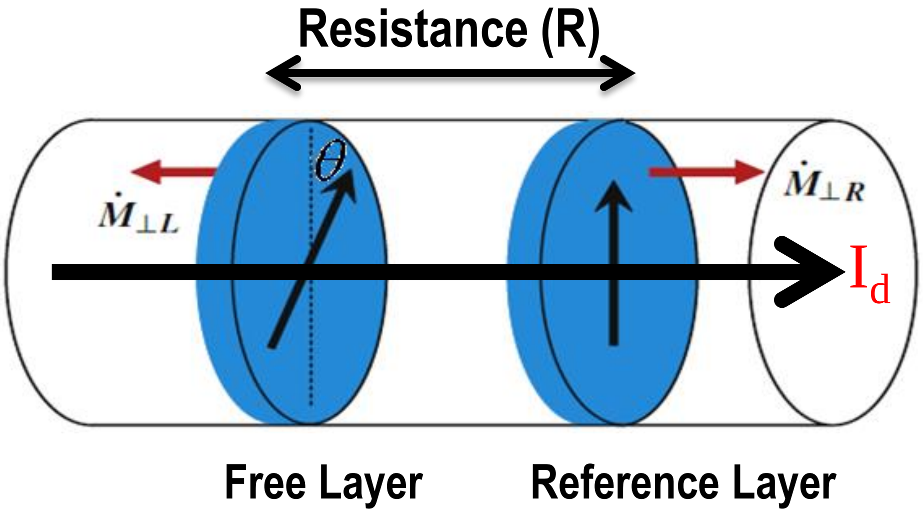
# Outline

- NVM Device Modeling
- NVM In-memory Logic
- NVM In-memory Architecture for Machine Learning

# State of STT-MTJ Devices

- **Macro-scale state of spintroinc device:**
  - Magnetization angle θ(t) between successive magnetic layers
  - State dynamics governed by Landau-Lifshitz-Gilbert equation

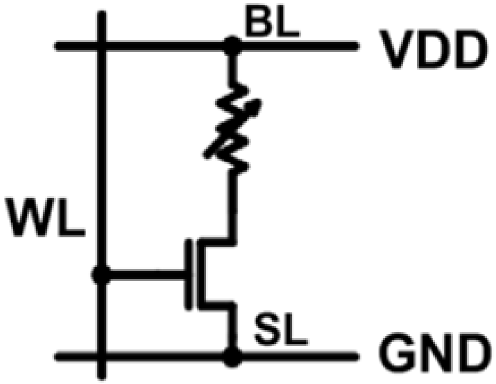


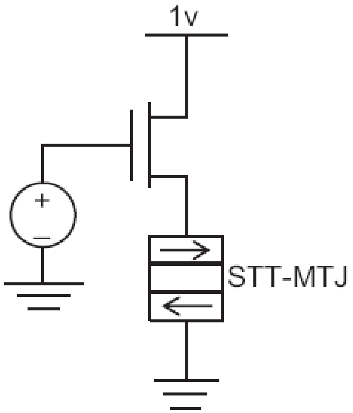


- **State θ(t) in terms of giant magnetization resistance:**

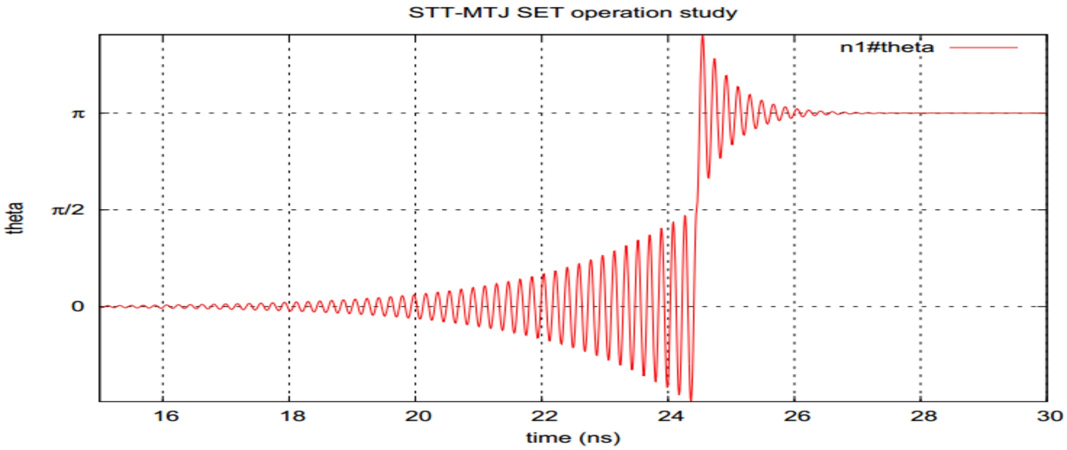
**GMR Equation** $R(\theta)=R(\theta_0)+\Delta R_{GMR}\left(1-\cos\theta(t)\right)/2$

# NVM SPICE for STT-MTJ





```
circuit: STT-MTJ SET operation study
.model nmodel1 sttmtj vcp=0 vcap=0
.model modname nmos level=5
vdd 1 0 1v
vcontrol 2 0 pwl(0 0 4ns 0 5ns 1 15ns 1 16ns 0 20ns 0)
m1 1 2 3 0 modname l=0.5u w=1000u
n1 sttmtj 3 4 nmodel1 theta0=0.001
vasst 4 0 0v
.tran 0.01n 20ns
.end
```



```
plot v(n1#theta)
plot (v(3)-v(4))/i(vasst)
```

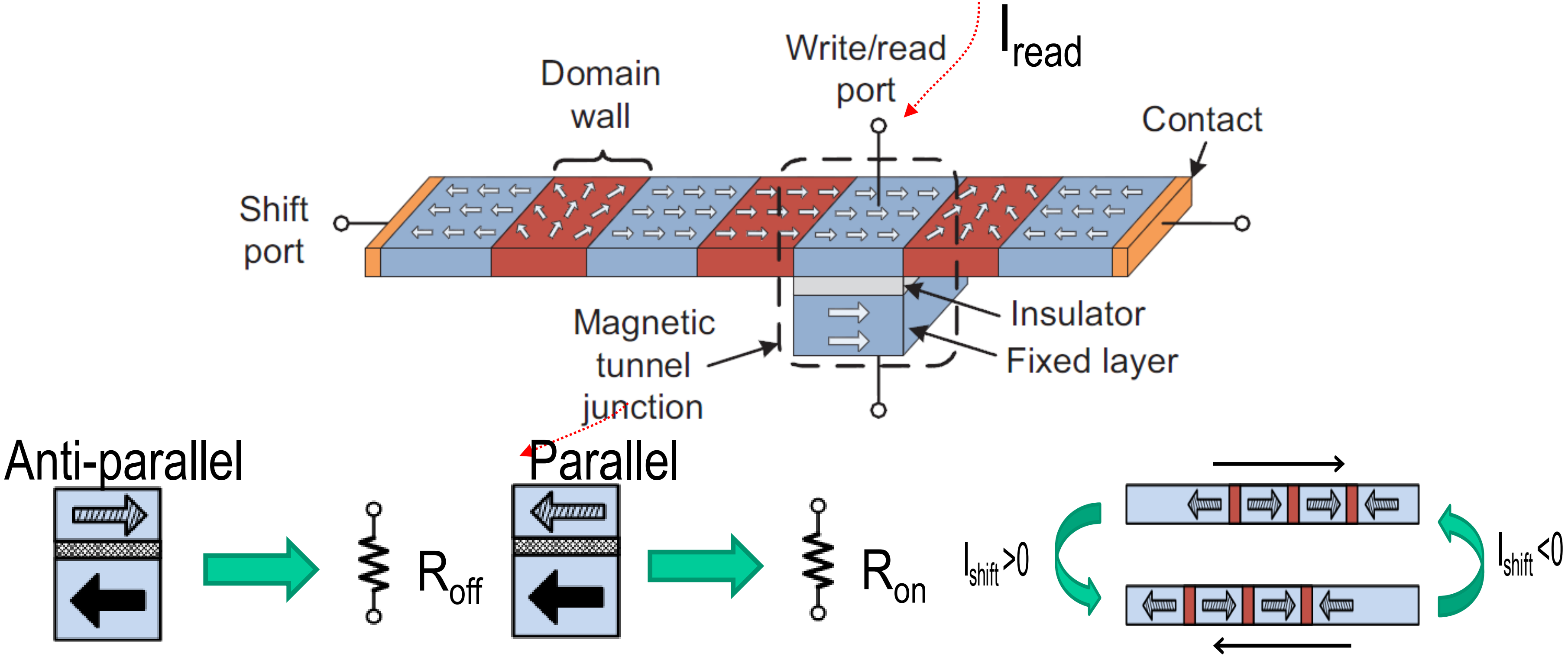
http://www.nvmspice.org

| Array size | Behavioral Macromodel (s) | Physical model in NVM-SPICE (s) | Speedup ratio |
|---|---|---|---|
| 8*8 | 2.522 | 0.257 | 10x |
| 16*16 | 98.131 | 1.87 | 52x |
| 32*32 | 1119.99 | 11.533 | 97x |
| 64*64 | 22188.8 | 189 | 117x |

# From STT-MTJ to Domain-wall Nanowire

**Shifter, Write, Read operation:**

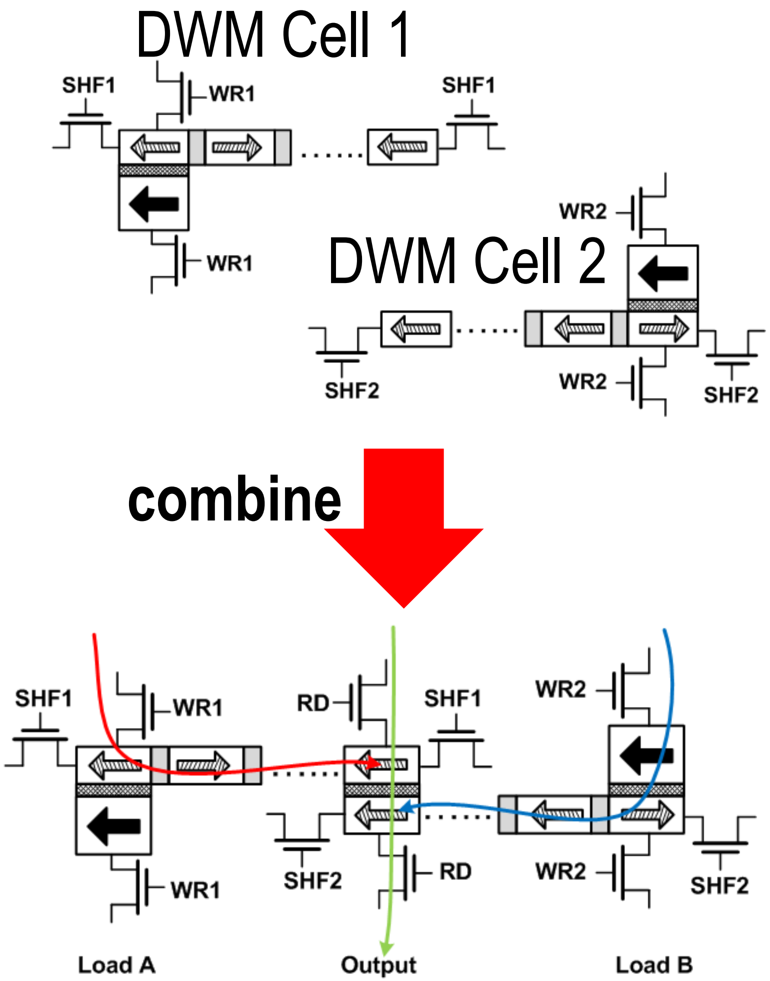
1. Apply shift current to select domain
2. Apply write/read current through write/read port
3. The state can be read out by detecting the MTJ resistance



# Outline

- NVM Device Modeling
- NVM In-memory Logic
- NVM In-memory Architecture for Machine Learning

# Domain-wall based XOR Logic



Domain-wall nanowire based XOR logic (DWL)

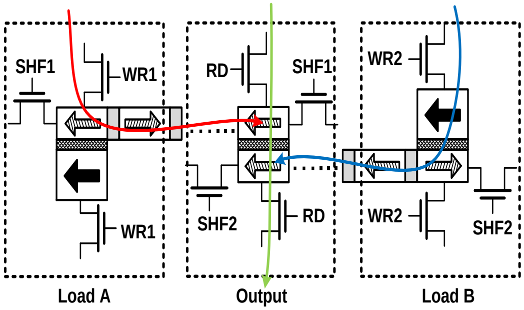
- •XOR gate is most complicated (16 transistors each gate) among all logic gates
- •XOR is highly used for big-data applications such as comparison and addition
- • Power optimized XOR gate by DWL

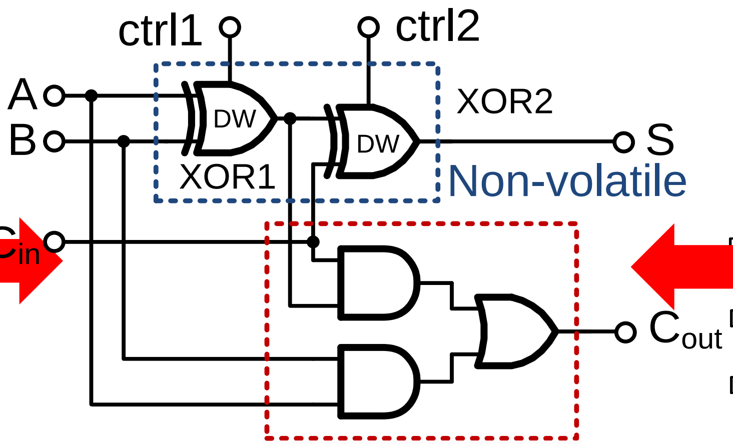
**Two domain-wall nanowire devices to build one XOR gate:**

- Write A to left nanowire
- Shift A to constructed port
- Write B to right nanowire
- Shift B to constructed port
- Read resistance of constructed port

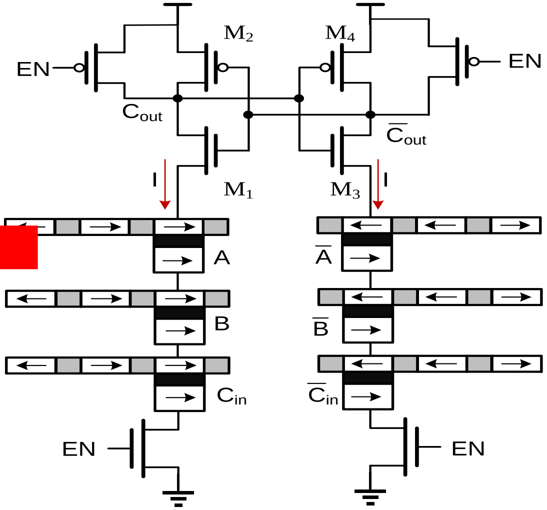
# Domain-wall based Full-adder and Multiplier

SUM





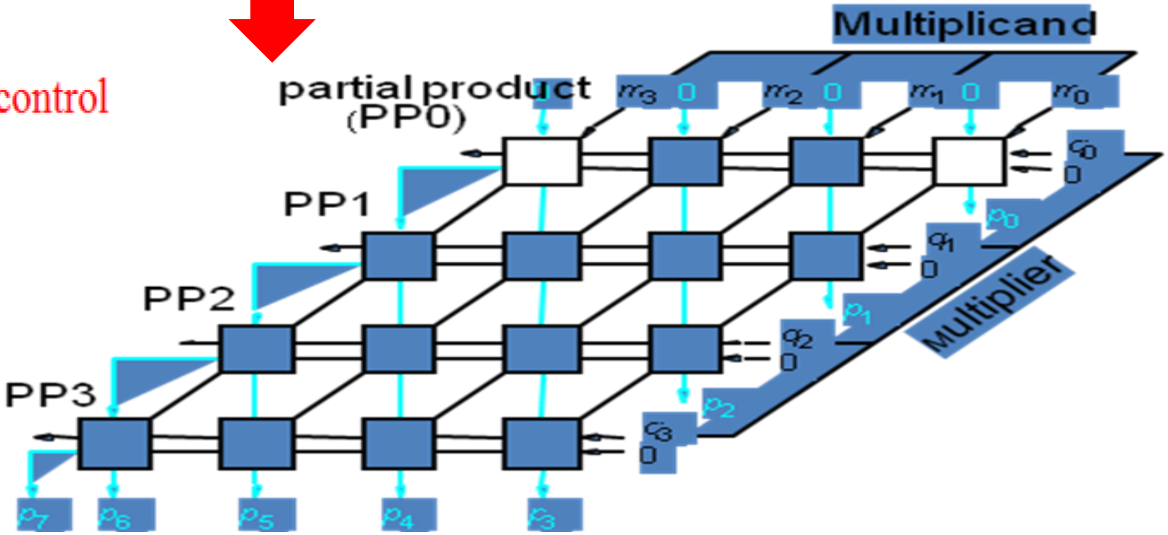
CARRY[1]



FULL ADDER/SHIFT MULTIPLIER

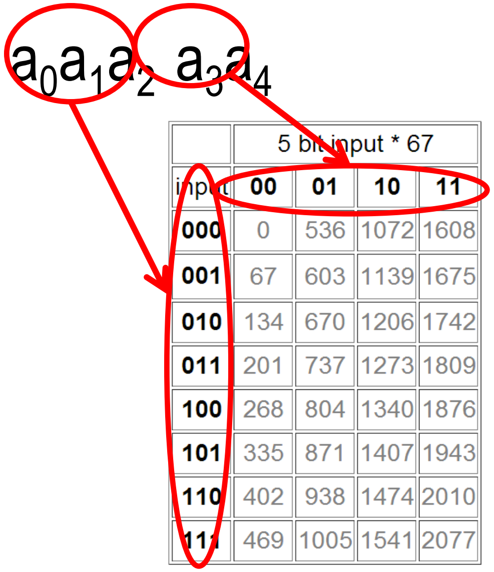
1000
* 1001 → Shift control
1000 → DW-ADDER
1000 → DW-Shift
1001000

1. Horizontal ops by shift
2. Vertical ops by adder



[1] H.-P. Trinh et.al. Electronics Letters, 2013

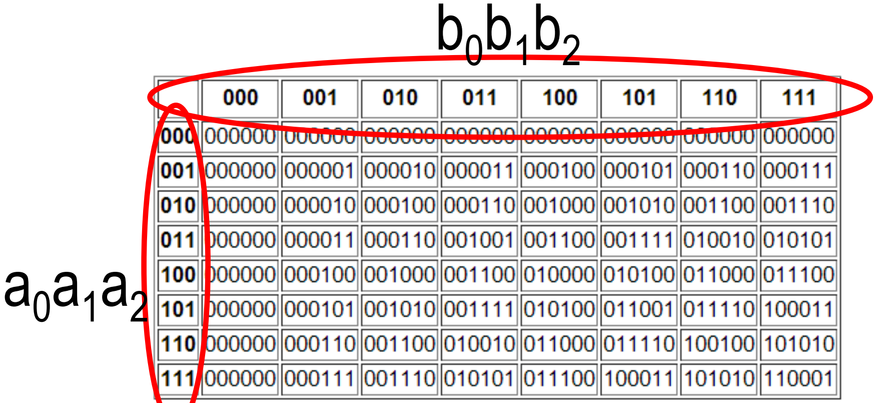
# Domain-wall based LUT Logic



$a_0a_1a_2$ $a_3a_4$

| | 5 bit input * 67 | | | |
|---|---|---|---|---|
| **input** | **00** | **01** | **10** | **11** |
| **000** | 0 | 536 | 1072 | 1608 |
| **001** | 67 | 603 | 1139 | 1675 |
| **010** | 134 | 670 | 1206 | 1742 |
| **011** | 201 | 737 | 1273 | 1809 |
| **100** | 268 | 804 | 1340 | 1876 |
| **101** | 335 | 871 | 1407 | 1943 |
| **110** | 402 | 938 | 1474 | 2010 |
| **111** | 469 | 1005 | 1541 | 2077 |

Single 5-bit input multiplication with constant



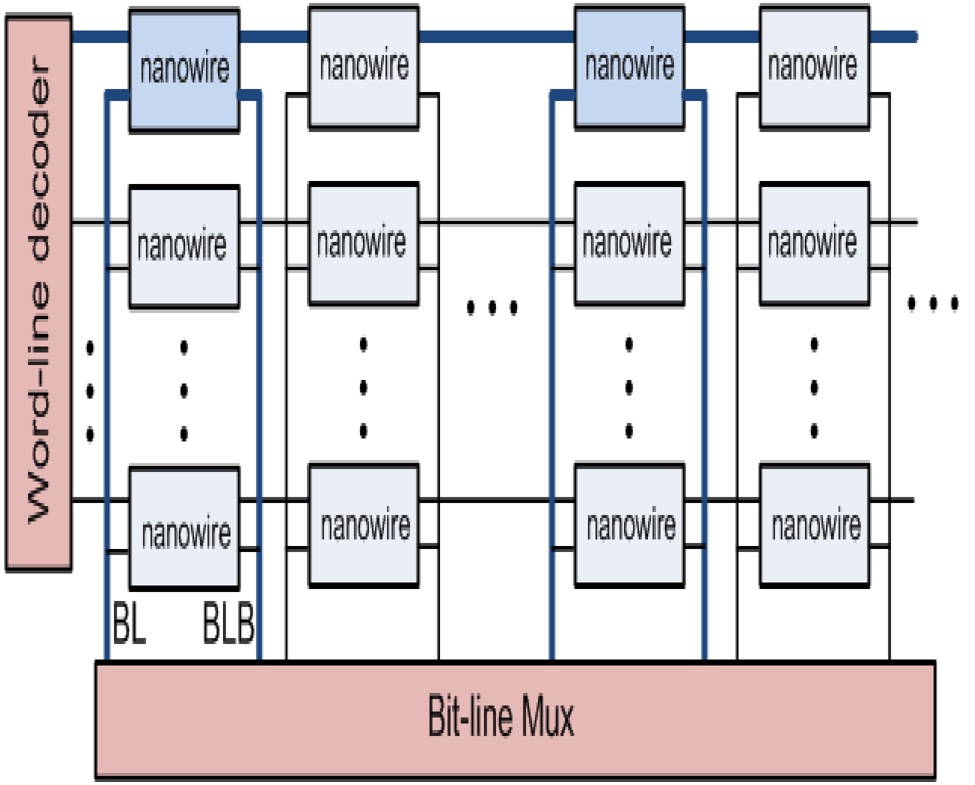
$b_0b_1b_2$

$a_0a_1a_2$

| | **000** | **001** | **010** | **011** | **100** | **101** | **110** | **111** |
|---|---|---|---|---|---|---|---|---|
| **000** | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 |
| **001** | 000000 | 000001 | 000010 | 000011 | 000100 | 000101 | 000110 | 000111 |
| **010** | 000000 | 000010 | 000100 | 000110 | 001000 | 001010 | 001100 | 001110 |
| **011** | 000000 | 000011 | 000110 | 001001 | 001100 | 001111 | 010010 | 010101 |
| **100** | 000000 | 000100 | 001000 | 001100 | 010000 | 010100 | 011000 | 011100 |
| **101** | 000000 | 000101 | 001010 | 001111 | 010100 | 011001 | 011110 | 100011 |
| **110** | 000000 | 000110 | 001100 | 010010 | 011000 | 011110 | 100100 | 101010 |
| **111** | 000000 | 000111 | 001110 | 010101 | 011100 | 100011 | 101010 | 110001 |

Two 3-bit  operands multiplication

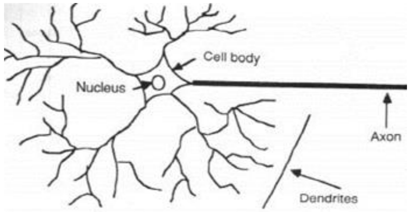
- Any logic function y=f(x) can be mapped to look-up table (LUT) with specified inputs
- DWM for LUT word-line and bit-line decoders take the input and find the target nanowire cell that stores results

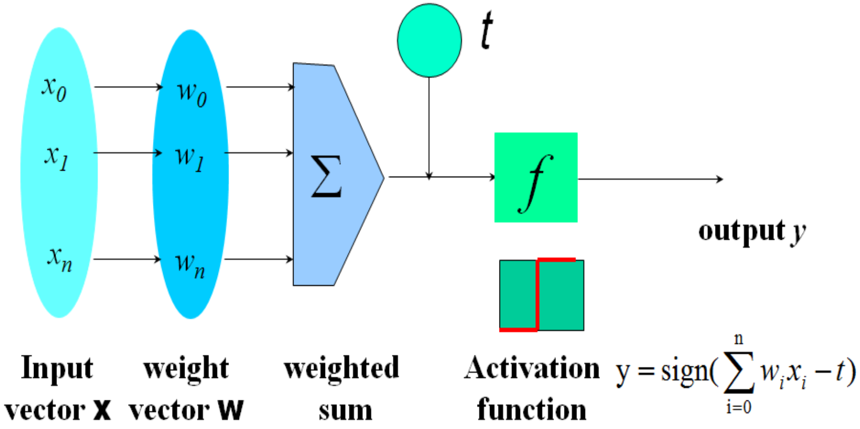


# Outline

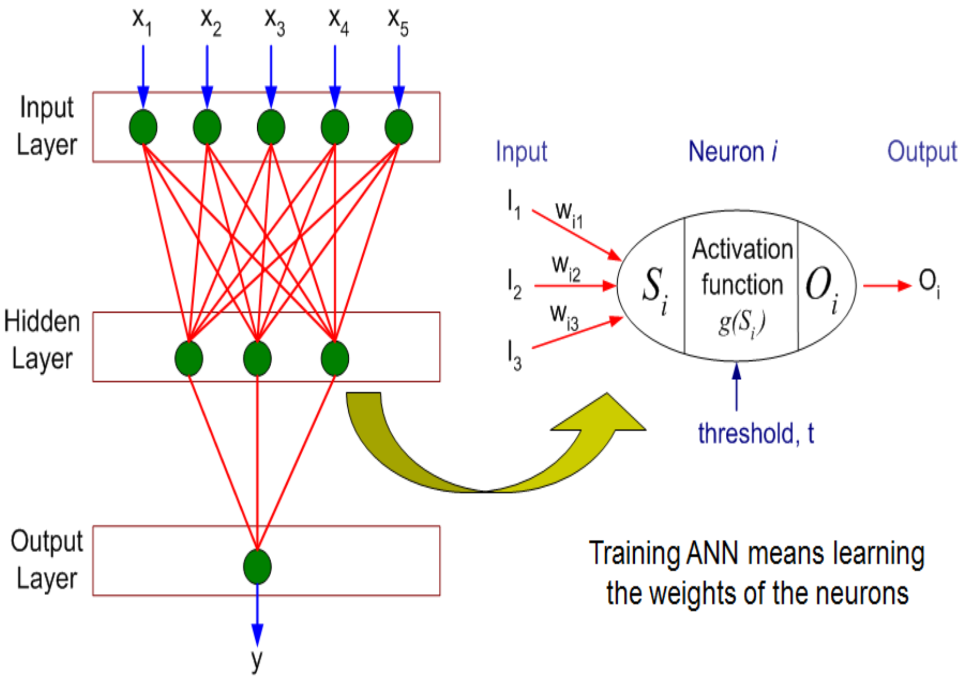
- NVM Device Modeling
- NVM In-memory Logic
- NVM In-memory Architecture for Machine Learning

# Neuron and Neuron Network





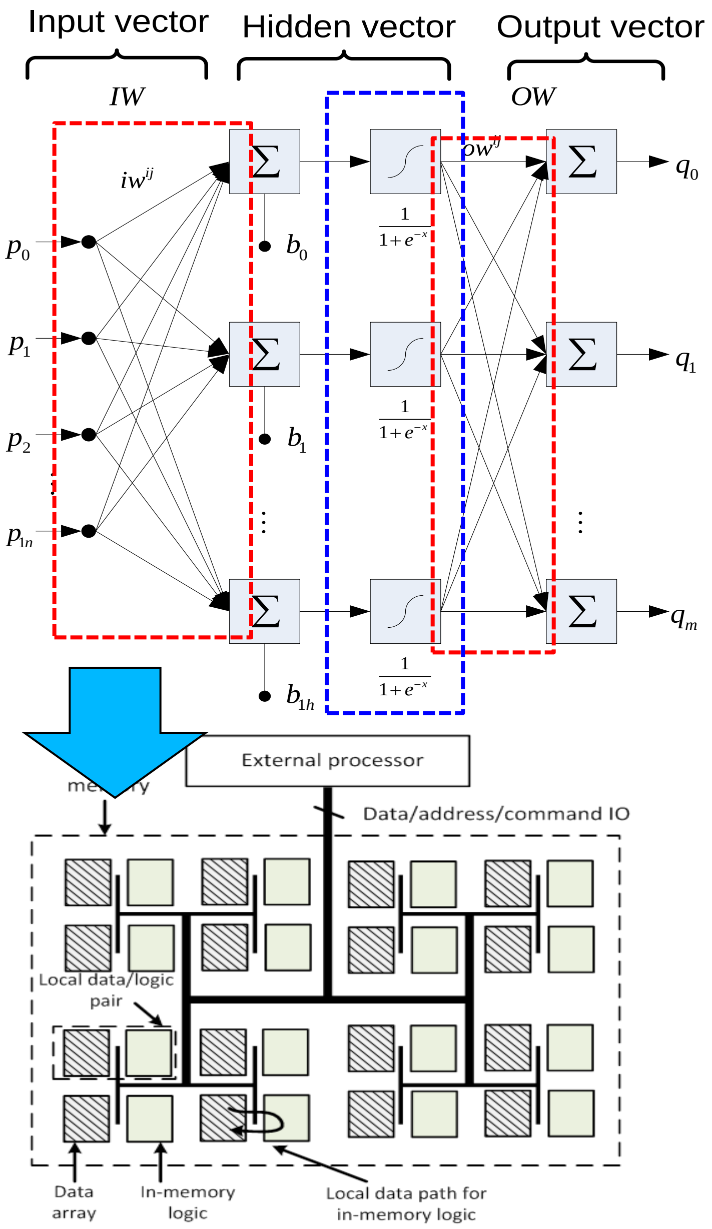
$y = \text{sign}(\sum_{i=0}^{n} w_i x_i - t)$



Training ANN means learning the weights of the neurons

- **Neuron model**
  - An assembly of interconnected nodes and weighted links
  - Output node sums up each of its input value according to weights of its links
  - Compare output node against some threshold *t*
- **Neuron network**
  - A set of neurons with forwarded connection from inputs to outputs
  - Hidden layer weights are obtained from off-line training and updated from on-line learning

# Non-volatile In-Memory ELM-SR



- **Extreme learning machine**
  - Single hidden layer feed-forward neural networks
  - Tuning-free without expensive iterative training of parameters
- **ELM based image super-resolution (ELM-SR)**
  - Enhance resolution in image recognition for recognition
- **How to map ELM-SR to non-volatile in-memory architecture?**

# Extreme Learning Machine based Super-resolution

**ELM-SR flow:**

**a) Input (offline memory) : feature vector *P* extracted from images**

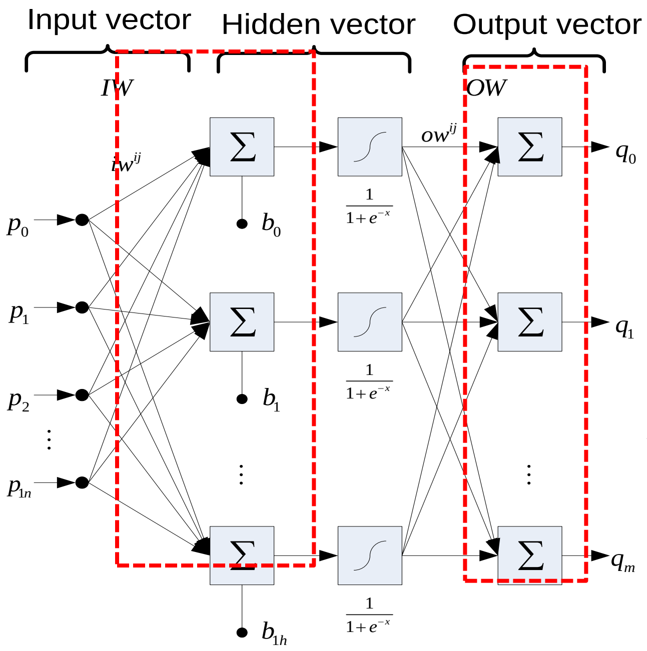
**b) Training (offline memory) → obtain output weight vector *ow***

**c) Randomly generated input weight *iw* bias *b* matrices (offline memory): parameters tuning free**

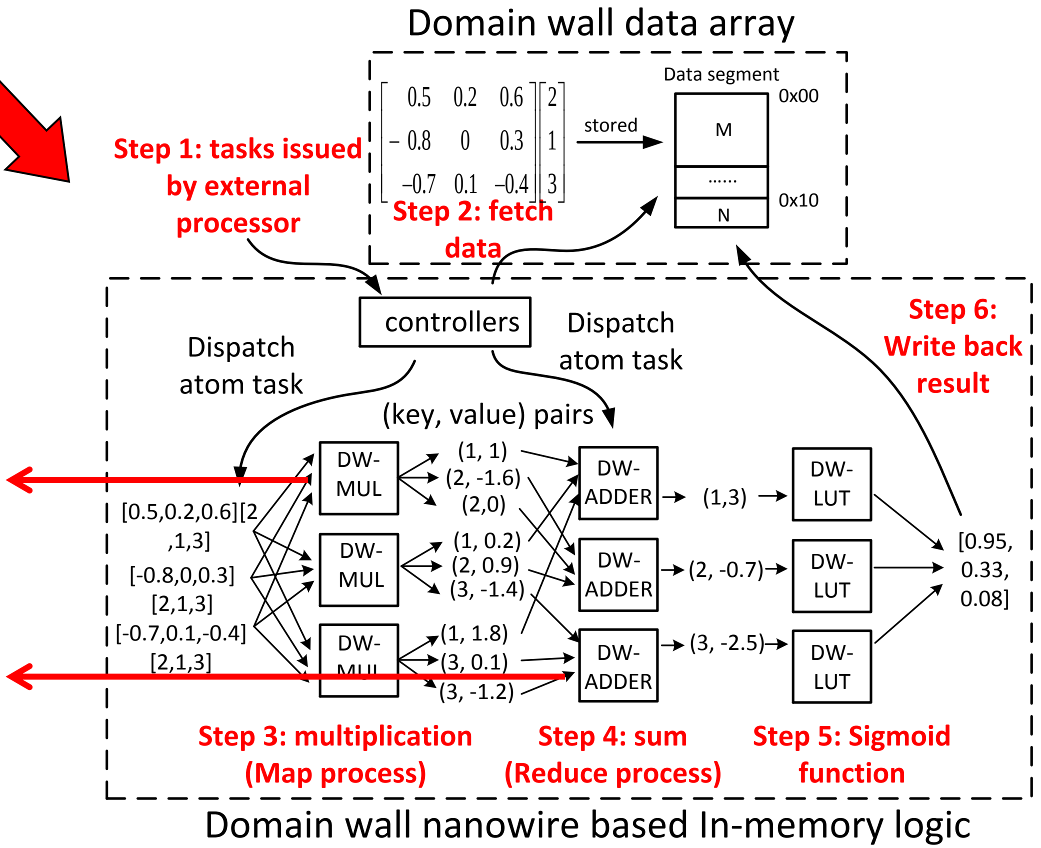
**d) Testing (online logic)**

1. input vectors times input weight vector $P*iw$
2. *sigmoid* function $s = sigmoid(P*iw+b)$
3. multiplication by output weight matrix $s*ow$

# ELM-SR Operation Mapping: Weighted Sum



1. **Weighted sum (inner product):**
   (a) DW-ADDER and DW-MULTIPLIER
   (b) MapReduce parallel computing



Domain wall data array

$$\begin{bmatrix} 0.5 & 0.2 & 0.6 \\ -0.8 & 0 & 0.3 \\ -0.7 & 0.1 & -0.4 \end{bmatrix}\begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}$$

stored

Data segment

0x00

M

......

N

0x10

Step 1: tasks issued by external processor

Step 2: fetch data

controllers

Dispatch atom task

Dispatch atom task

(key, value) pairs

[0.5,0.2,0.6][2,1,3]

[-0.8,0,0.3][2,1,3]

[-0.7,0.1,-0.4][2,1,3]

DW-MUL

DW-MUL

DW-MUL

(1, 1)
(2, -1.6)
(2,0)

(1, 0.2)
(2, 0.9)
(3, -1.4)

(1, 1.8)
(3, 0.1)
(3, -1.2)

DW-ADDER

DW-ADDER

DW-ADDER

(1,3)

(2, -0.7)

(3, -2.5)

DW-LUT

DW-LUT

DW-LUT

[0.95, 0.33, 0.08]

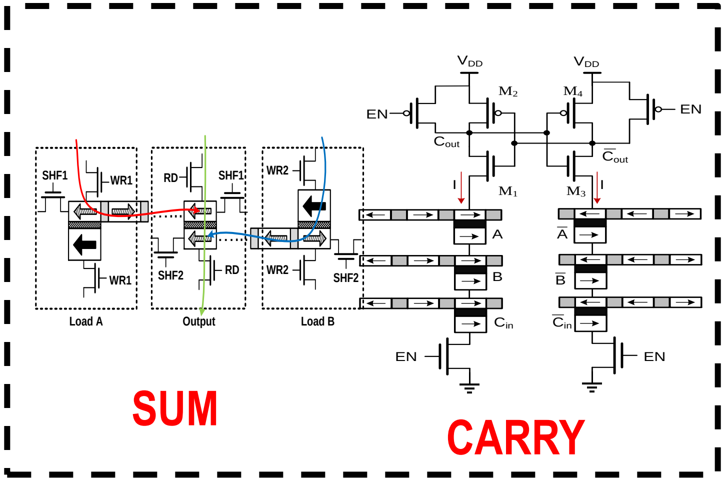
Step 6: Write back result

Step 3: multiplication (Map process)

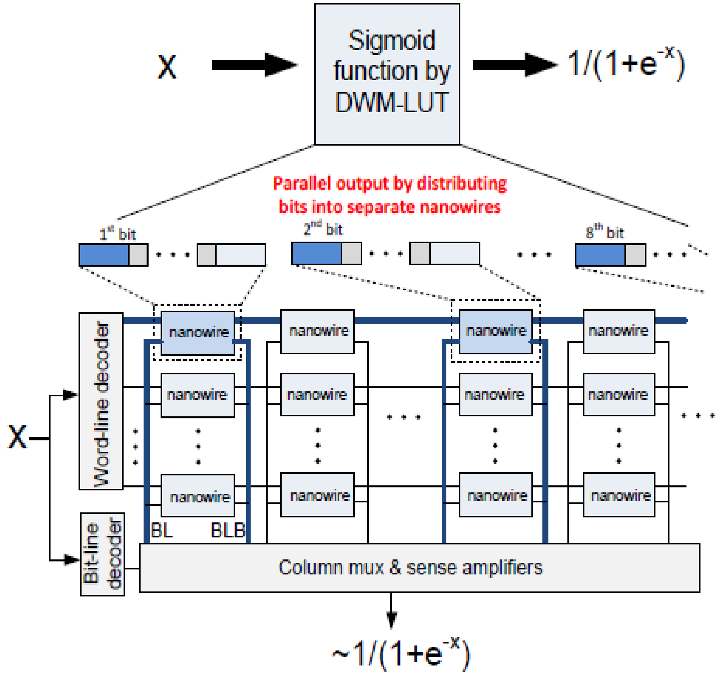
Step 4: sum (Reduce process)

Step 5: Sigmoid function

Domain wall nanowire based In-memory logic

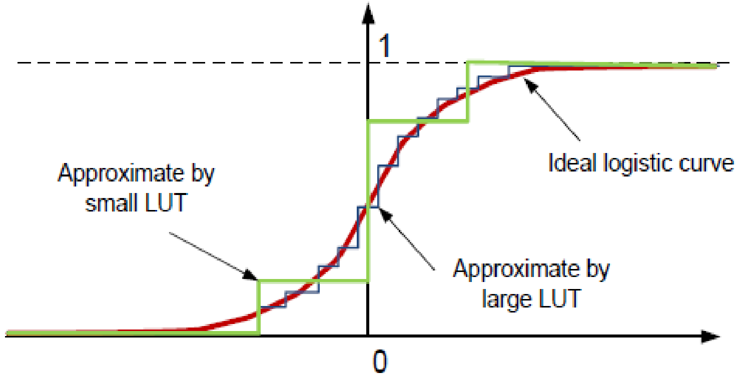


# ELM-SR Operation Mapping: Sigmoid



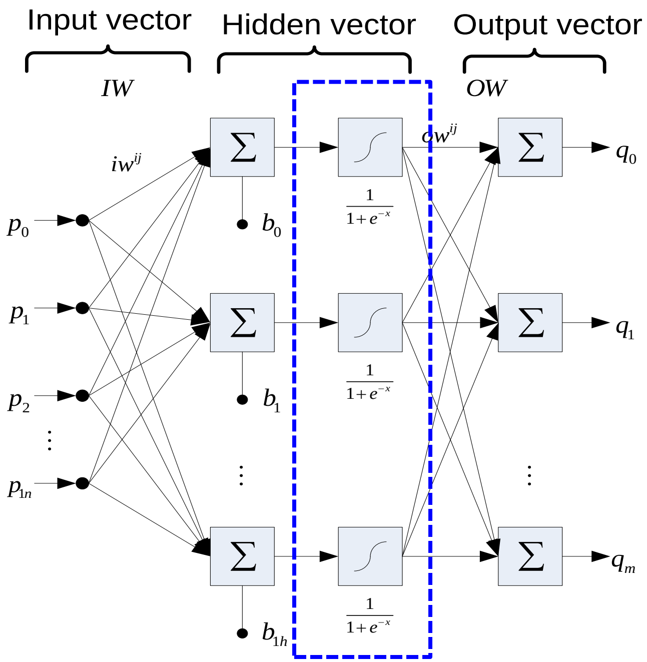
X
Sigmoid function by DWM-LUT
1/(1+e^-x)
Parallel output by distributing bits into separate nanowires
1st bit
2nd bit
8th bit
Word-line decoder
X
nanowire
BL
BLB
Bit-line decoder
Column mux & sense amplifiers
~1/(1+e^-x)

(a)



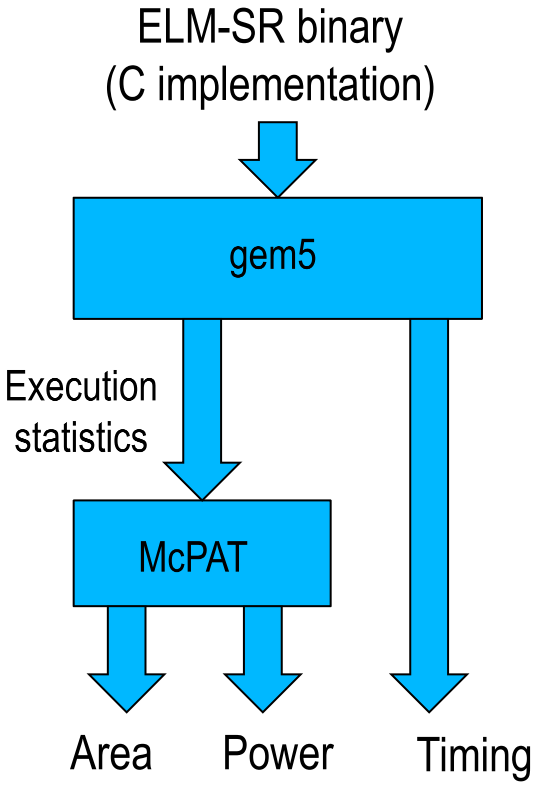
(b)

2. **Sigmoid function:**
   (a) DW-LUT (x meaningful around -8 to 8, y ranges from -1 to 1, efficient by LUT)
   (b) Tradeoff between LUT size and accuracy

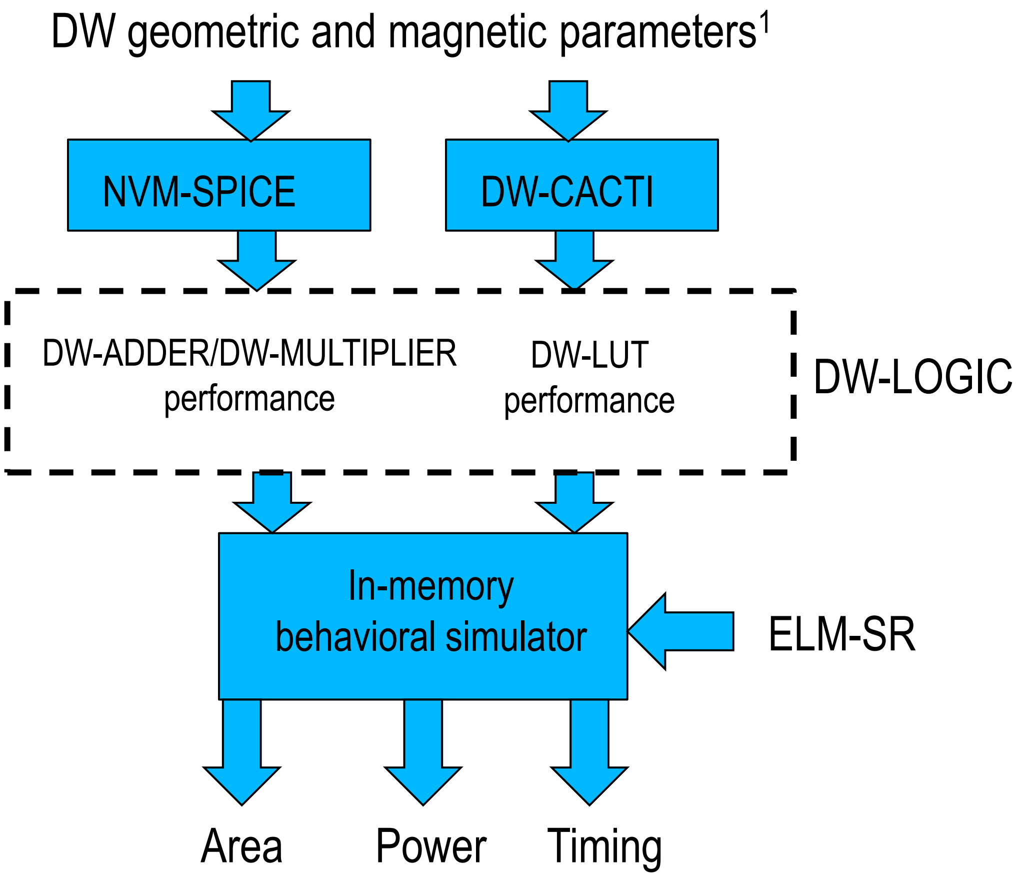


# Experimental Settings and Methodology

## Conventional general purpose processor platform



## Proposed in-memory domain-wall based neural network platform

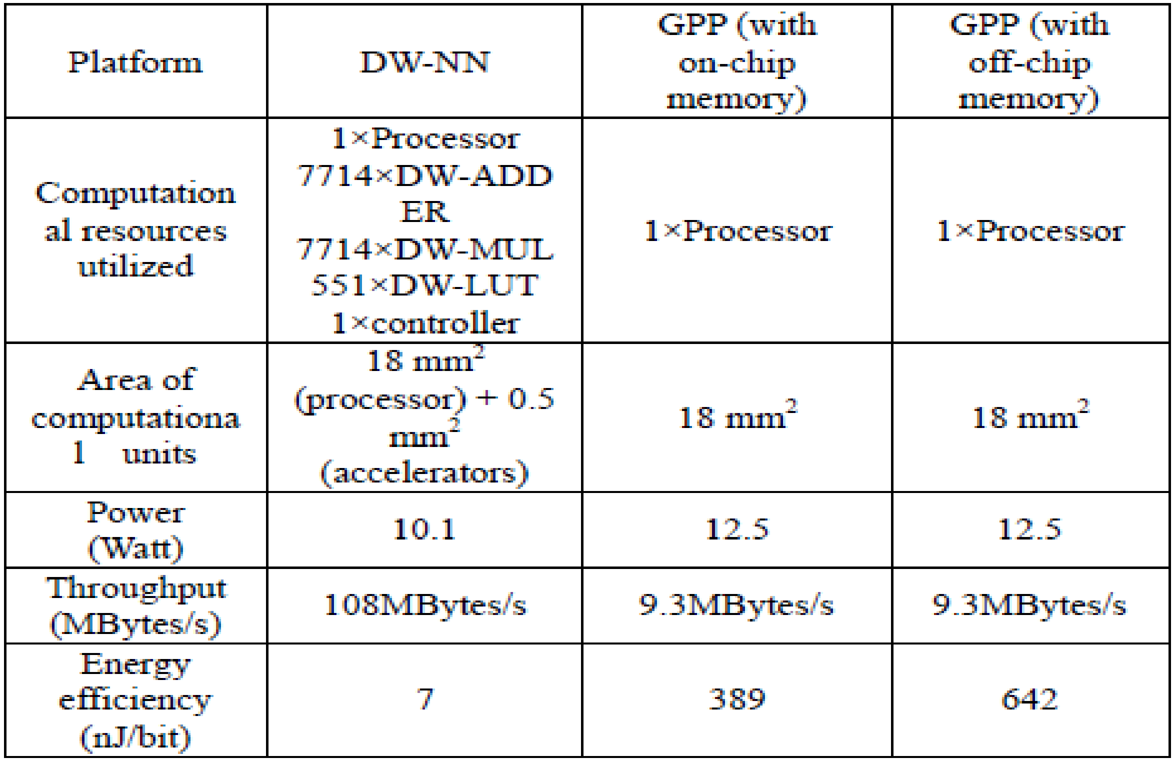

DW geometric and magnetic parameters[1]
NVM-SPICE
DW-CACTI
DW-ADDER/DW-MULTIPLIER performance
DW-LUT performance
DW-LOGIC
In-memory behavioral simulator
ELM-SR
Area
Power
Timing

[1] Technology node of 32nm is assumed with width of 32nm, length of 64nm per domain, and thickness of 2.2nm for one domain-wall nanowire; the $R_{off}$ is set at 2600Ω, the $R_{on}$ at 1000Ω, the writing current at $100\mu A$, and the current density at $6 \times 10^8 A/cm^2$ for shift-operation.

# Preliminary Results and Conclusions

Machine learning  for super-resolution imaging

*Comparisons with conventional architecture*



| Platform | DW-NN | GPP (with on-chip memory) | GPP (with off-chip memory) |
|---|---|---|---|
| Computation al resources utilized | 1×Processor<br>7714×DW-ADD ER<br>7714×DW-MUL<br>551×DW-LUT<br>1×controller | 1×Processor | 1×Processor |
| Area of computationa l   units | 18 mm$^2$ (processor) + 0.5 mm$^2$ (accelerators) | 18 mm$^2$ | 18 mm$^2$ |
| Power (Watt) | 10.1 | 12.5 | 12.5 |
| Throughput (MBytes/s) | 108MBytes/s | 9.3MBytes/s | 9.3MBytes/s |
| Energy efficiency (nJ/bit) | 7 | 389 | 642 |

1. All operations involved in machine learning on neural network can be mapped to a logic-in-memory architecture by non-volatile domain-wall nanowire.
2. I/O traffic in proposed DW-NN is greatly alleviated with an energy efficiency improvement by 92x and throughput improvement by 11.6x compared to the conventional image processing system by general purpose processor.

# Thank you!

Please send comments to haoyu@ntu.edu.sg

http://www.ntucmosetgp.net

NANYANG TECHNOLOGICAL UNIVERSITY