# 高性能計算基盤

## – High Performance Computing Platforms–

## ＃6

Analog Computing

*Renyuan Zhang*

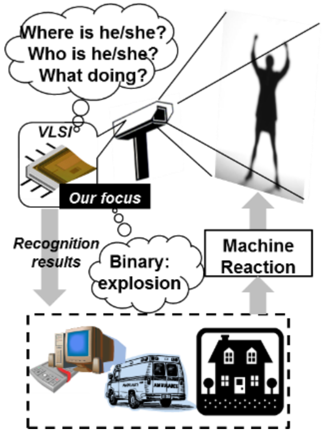
*2020/06/11*

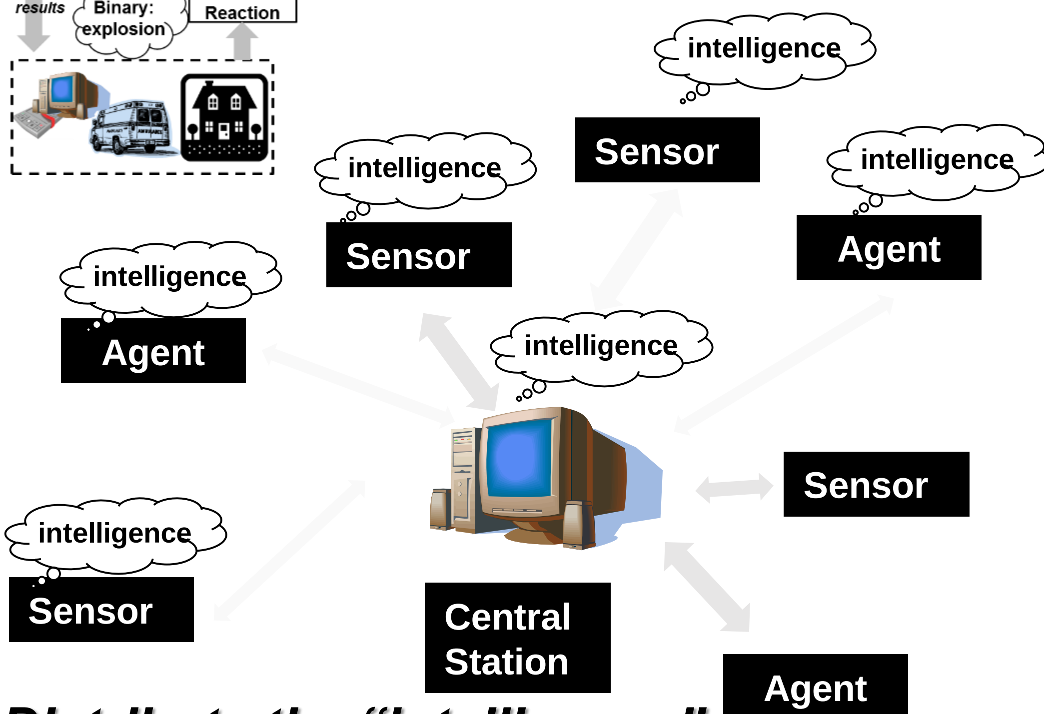
# Efficient & Smart Processors in IoT

Smart chips: VLSI implementations of machine learning (on-chip learning)

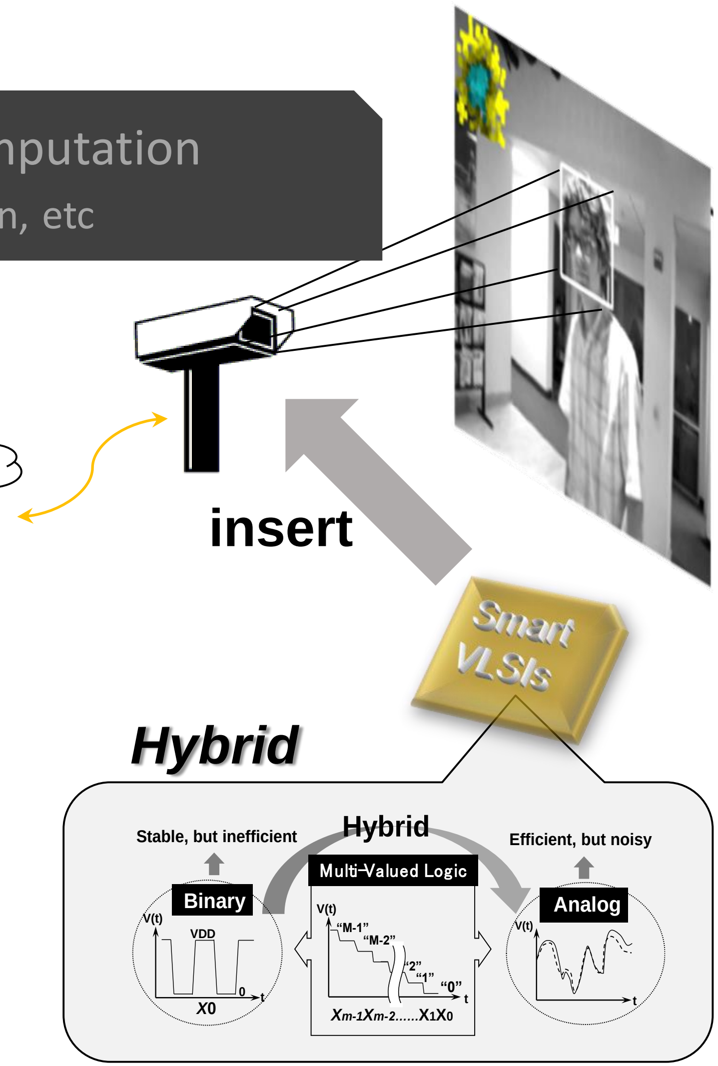
**Challenge: ultra-efficient computation**

Scale, speed, energy, interconnection, etc

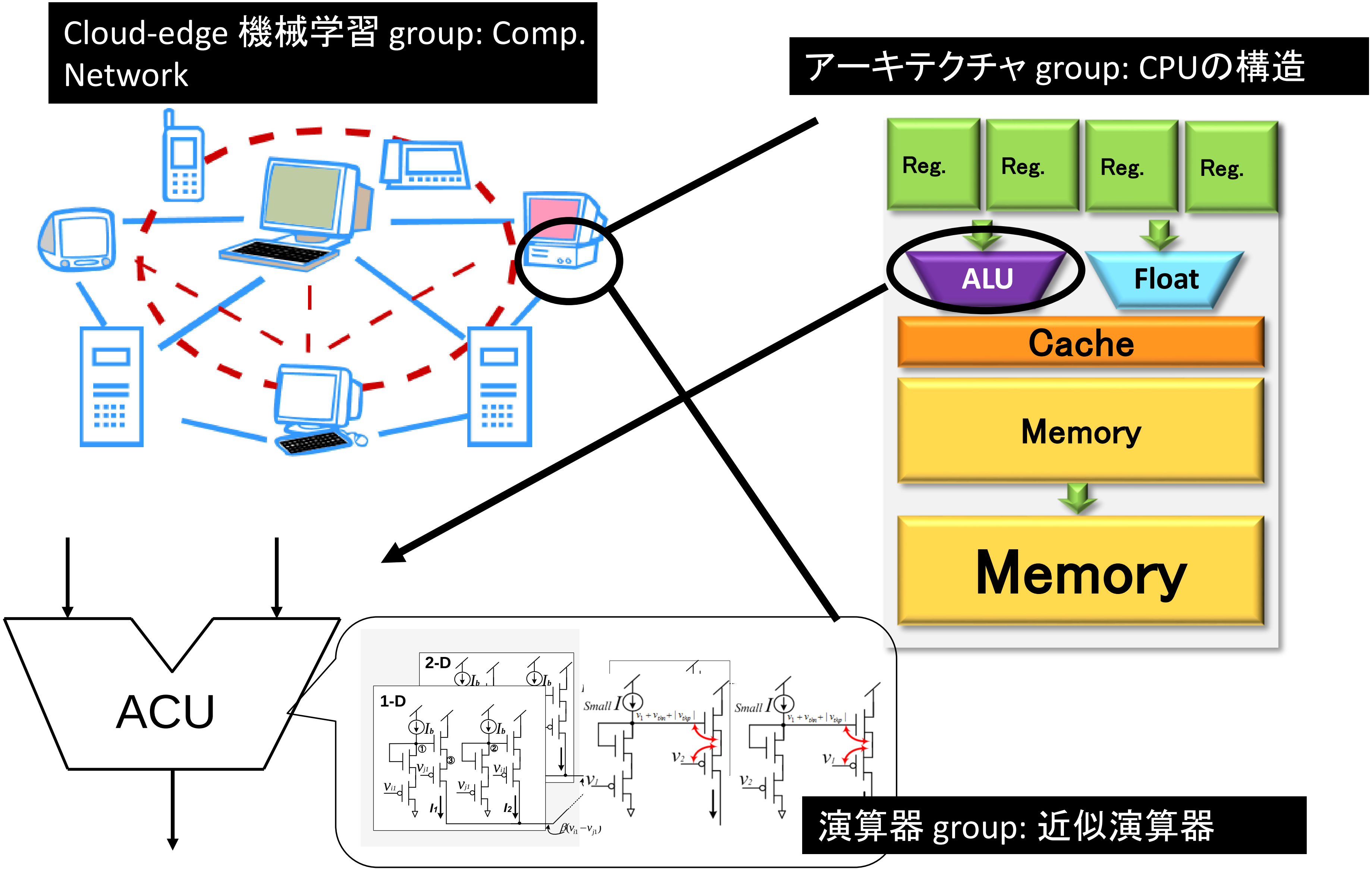


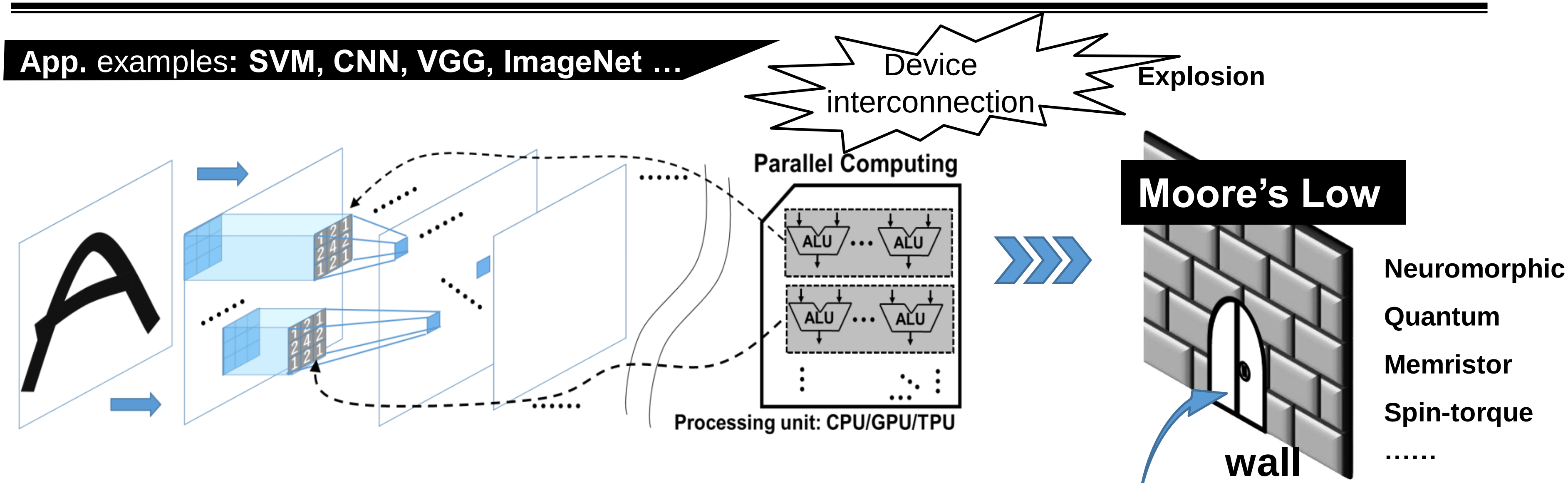

**Distribute the "intelligence"**



# 研究室の全体像


Cloud-edge 機械学習 group: Comp. Network
アーキテクチャ group: CPUの構造
Reg.
Reg.
Reg.
Reg.
ALU
Float
Cache
Memory
Memory
ACU
2-D
1-D
Small I
Small I
演算器 group: 近似演算器

# Accelerating AI algorithms in Post-Moore era




Domain Specified Accelerator (DSA)

Neurongrid @Stanford
Brein @Hokkaido Univ.
CNN kernel array @anywhere

Two directions

General purpose parallel computing

Multi-Core CPU
GPGPU
TPU ……


***Generality***

*Rich trade-off*
*Beyond trade-off*

**Approximate Computing**

- Accuracy ~1/4
- Size ~1/20
- Interconnection ~1/10
- Energy ~1/20

# How about Analog-like?

**Represent ``data" by various ``carriers"**
→ analog, probability, time, oscillation, etc

■ **Pros: Small circuit size** → many functions/area

**Opportunities for low power**

**High speed by Parallelism**

There are a lot of opportunities for:

**(very) low power and (very) high speed**

This is what is needed for current digital system

■ **Cons: Limited accuracy**

**Programmability !!! Expertise**

# Power dispassion

Power of VLSI chips: ~ sub-watt
Lamps: ~ 10-watt
Home-area machines: ~ 100-watt
Motors: ~ 1000-watt
Manufacturing Machines: ~ 10000-watt

Why we are not satisfied by such a tiny power order?

# Power dispassion

Reason 1: unplug devices

portable; body area (wearable); wireless; space exploration; bio-embedded.

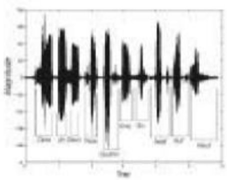
**Cell Phones**


**Digital Cameras**


**Hearing aids**

**Automotive**

**Biomedical**



**Computers**

# Power dispassion

Reason 2: heating!!!

- Speed and reliability: MOS-FETs are extremely temperature dependent devices
- Life-time

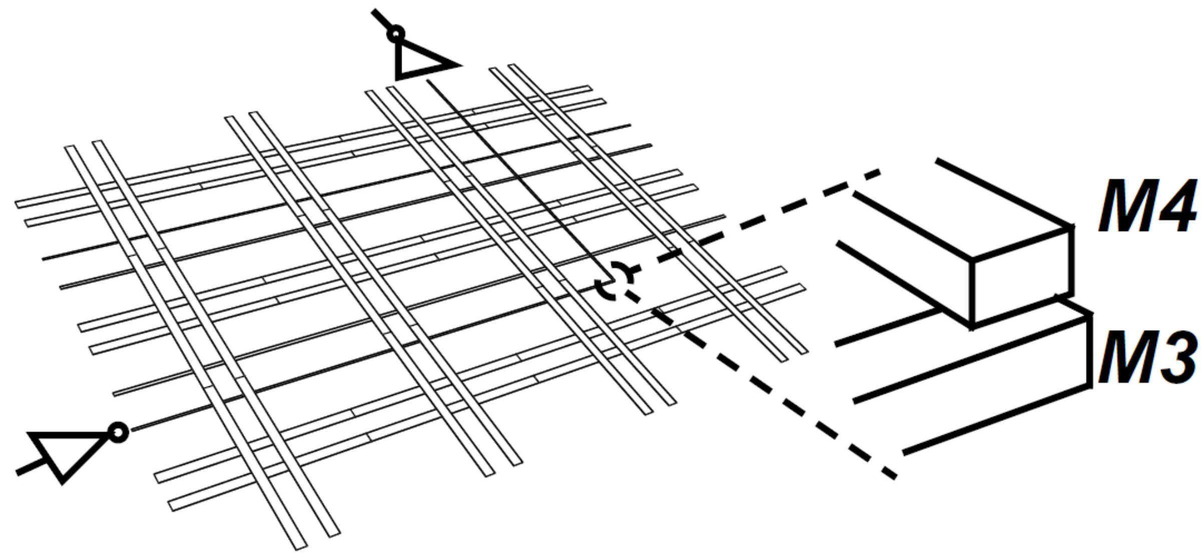
# Chip size

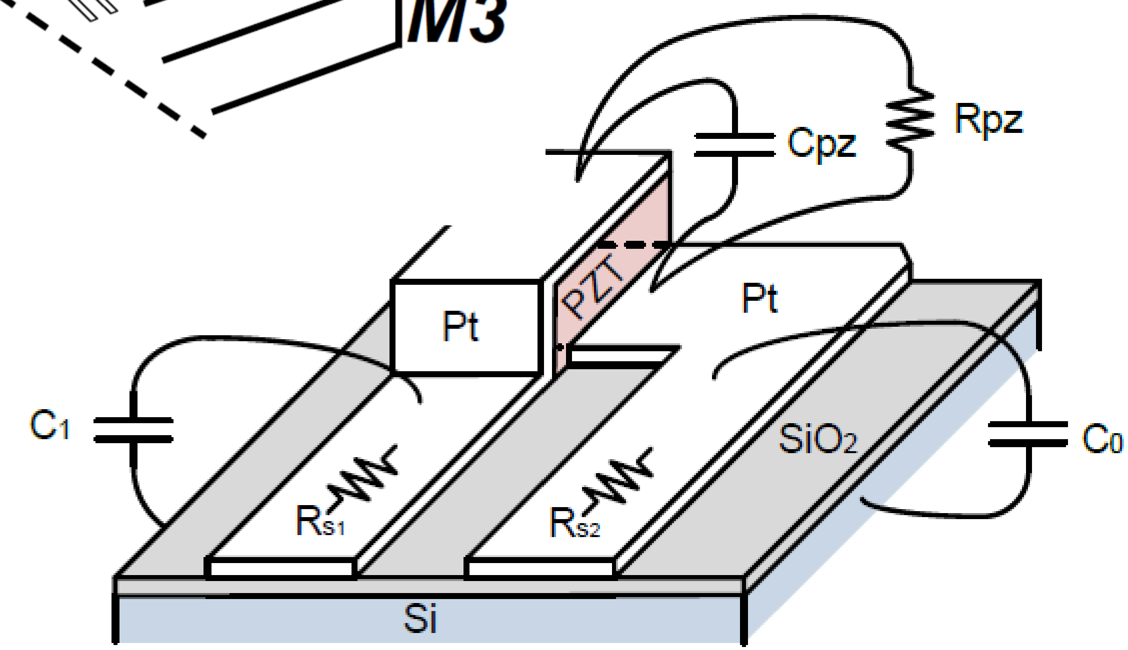
Generally, smaller sizes are always pursued due to:

1. Inner-connection
2. Die yield

# Chip size

1. Inner-connection

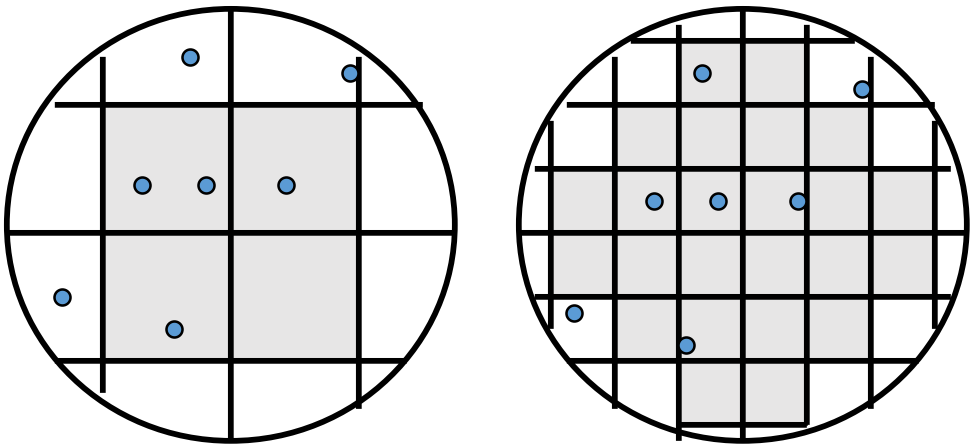


# Chip size

2. Die yield

How many chips are available?



$$\text{die yield} \quad = (1 + (\text{defects per unit area} \times \text{die area})/\alpha)^{-\alpha}$$

# Chip size

- Example
  - wafer size of 12 inches, die size of 2.5 $cm^2$, 1 defects/$cm^2$, $\alpha = 3$ (measure of manufacturing process complexity)
  - 252 dies/wafer (remember, wafers round & dies square)
  - die yield of 16%
  - 252 x 16% = only 40 dies/wafer die yield !

- Die cost is strong function of die area
  - proportional to the third or fourth power of the die area???

# Outline

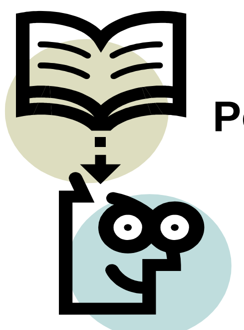
- What/Why/Why_not is analog computing
- Classic textbook type analog computing: OPAMP-based
- Oscillation driving analog
- Machine learning driving analog

# What is analog computing

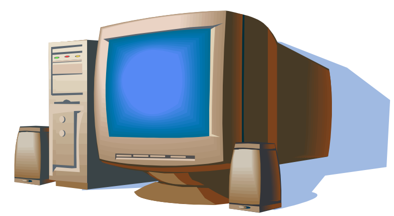
## Error sensitive tasks

$$253 \div 11 = ?$$

$$10010111 + 11011 = ?$$



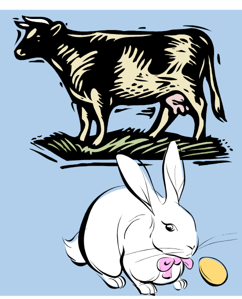
Possible but hard



Faster
Reliably

## Error tolerant tasks

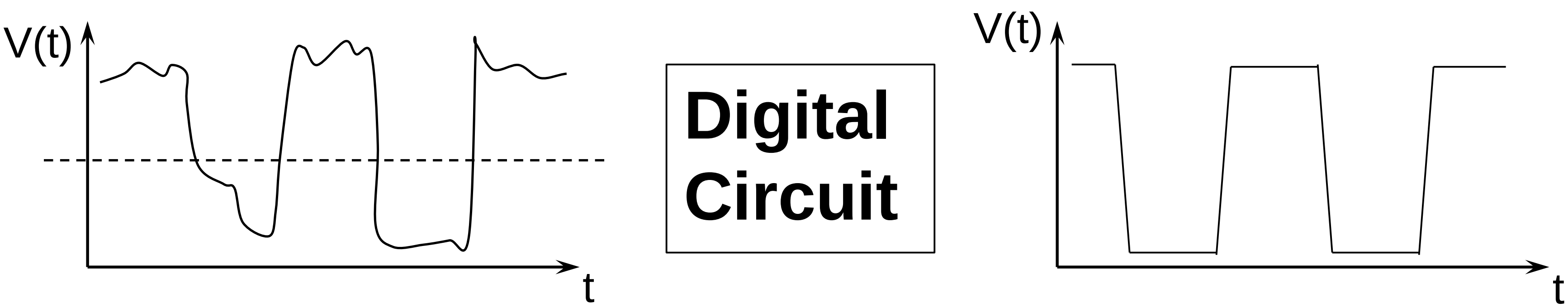
Is he a bad man?



Which is caw?

# Digital V.S. Analog Computation

■**Digital:**

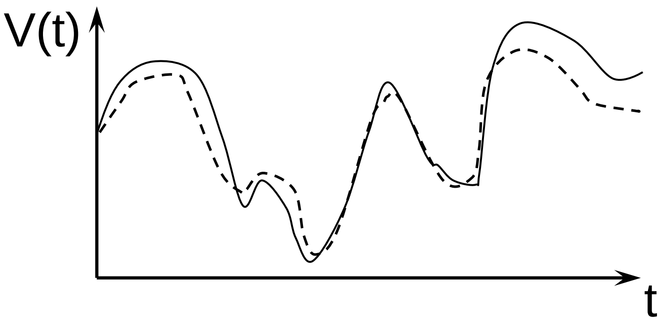



All energies are utilized to <u>eliminate Noise</u>

Strong Non-linearity of digital circuits can eliminate Noise

# Digital V.S. Analog Computation

■ **Analog:**




= noise, distortion → <u>always interfere with signal</u>

V(t) = data

- We cannot eliminate noise
- Linearity of Amplifier is important
- Audio Amplifier: all energy is used to guarantee linearity

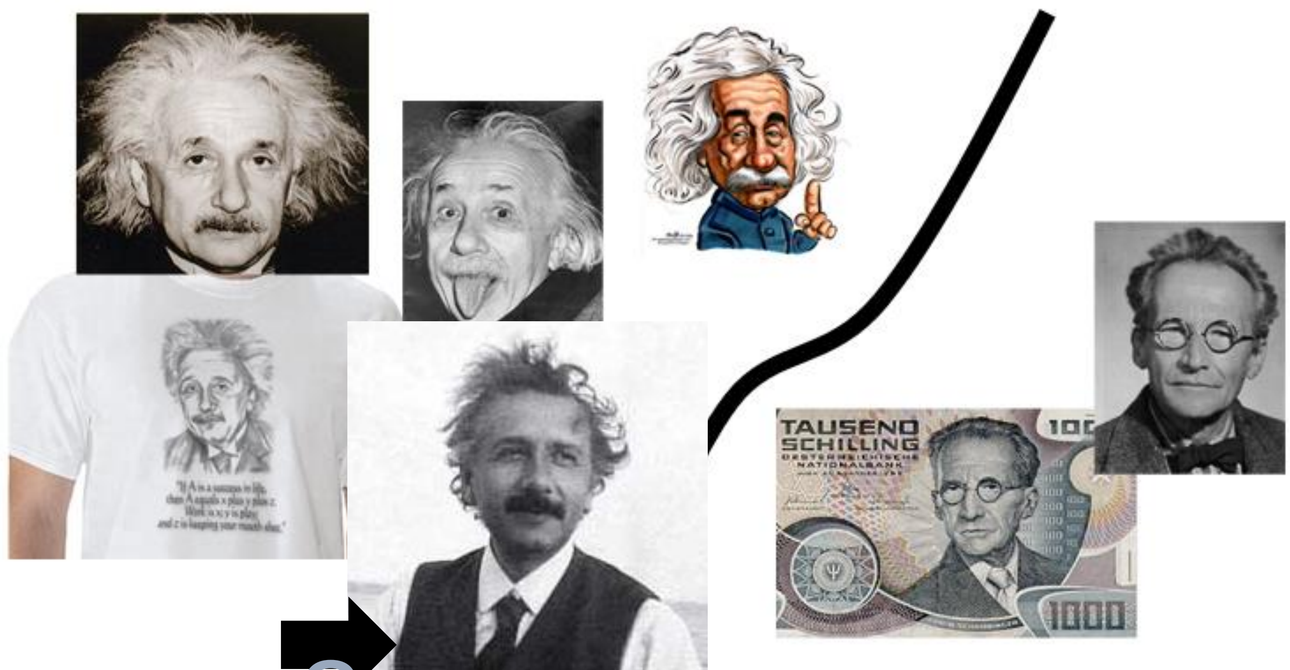
=> There is certain limitation in terms of computation accuracy

# “Brain-like Computing” aided by analog

- ■Algorithms must be “Error tolerant” inherently

(A lot of opportunities for ANALOG computing)

▲ Probl




▲ Deci

voting

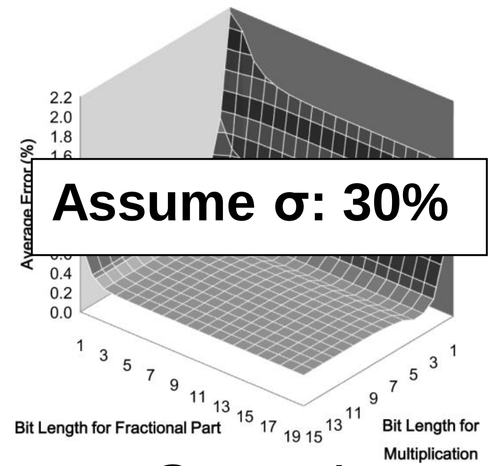
?

# Computational accuracy

■ On algorithm side

Brain-like Computing

→ Classifier (NN, SVM, ……)




*D* = Similarity Meas. (Euclidean, Manhattan, Gaussian……)

■ Circuit/Device inaccuracy

$$D = \exp\Big(\underline{\sum_{i=1}^{n}} d_i\Big)$$

n: dimensions

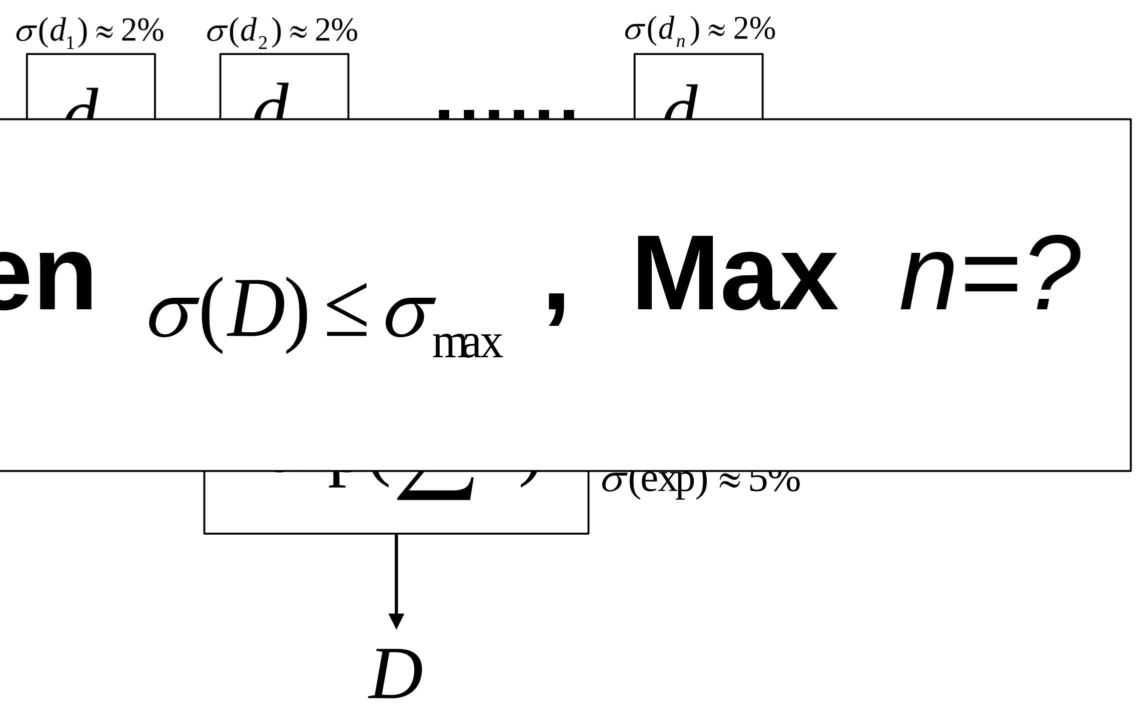
Element similarity-evaluate circuit

$$d_i = \| \mathbf{T}_i - \mathbf{X}_i \|$$

# Number of dimensions

- Algorithms allowable maximum: $\sigma_{\max} = 30\%$

- **Circuit / Device**




**When** $\sigma(D) \leq \sigma_{\max}$ **, Max** $n$=?

# Number of dimensions

- Algorithms allowable maximum: $\sigma_{\max} = 30\%$

- **Circuit / Device**

**Choose analog if you are sure you can sacrifice something in your specific tasks**

$$\sigma(D) = \sqrt{n}\sigma(d_i) + \sigma(\exp)$$

$$\rightarrow \sigma(D) + \Delta(samples) \le 30\%$$

$$\rightarrow \sqrt{n}\cdot 2\% + 5\% + \Delta(samples) \le 30\%$$

$\rightarrow$ $\boxed{\text{n} \rightarrow \text{100, when } \Delta(samples) \approx 5\%}$

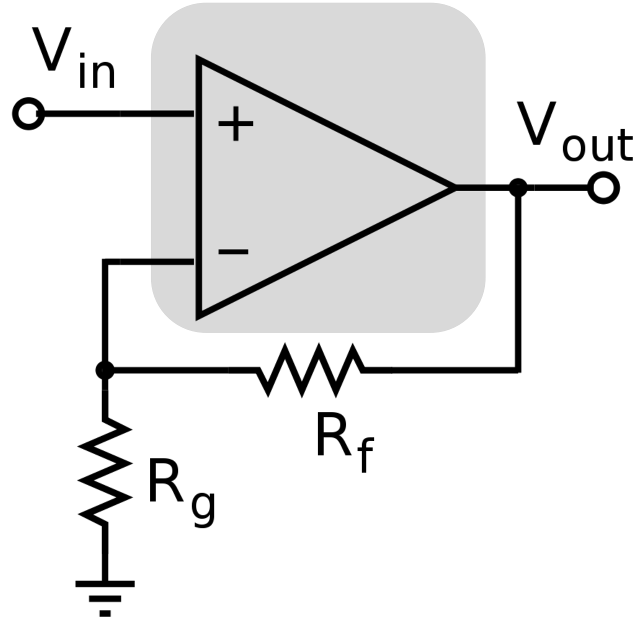
# Outline

- What/Why/Why_not is analog computing
- Classic textbook type analog computing: OPAMP-based
- Oscillation driving analog
- Machine learning driving analog

# **Candidate (1)** OPAMP-based analog computing

**Classic, textbook-like, easy, convenient**

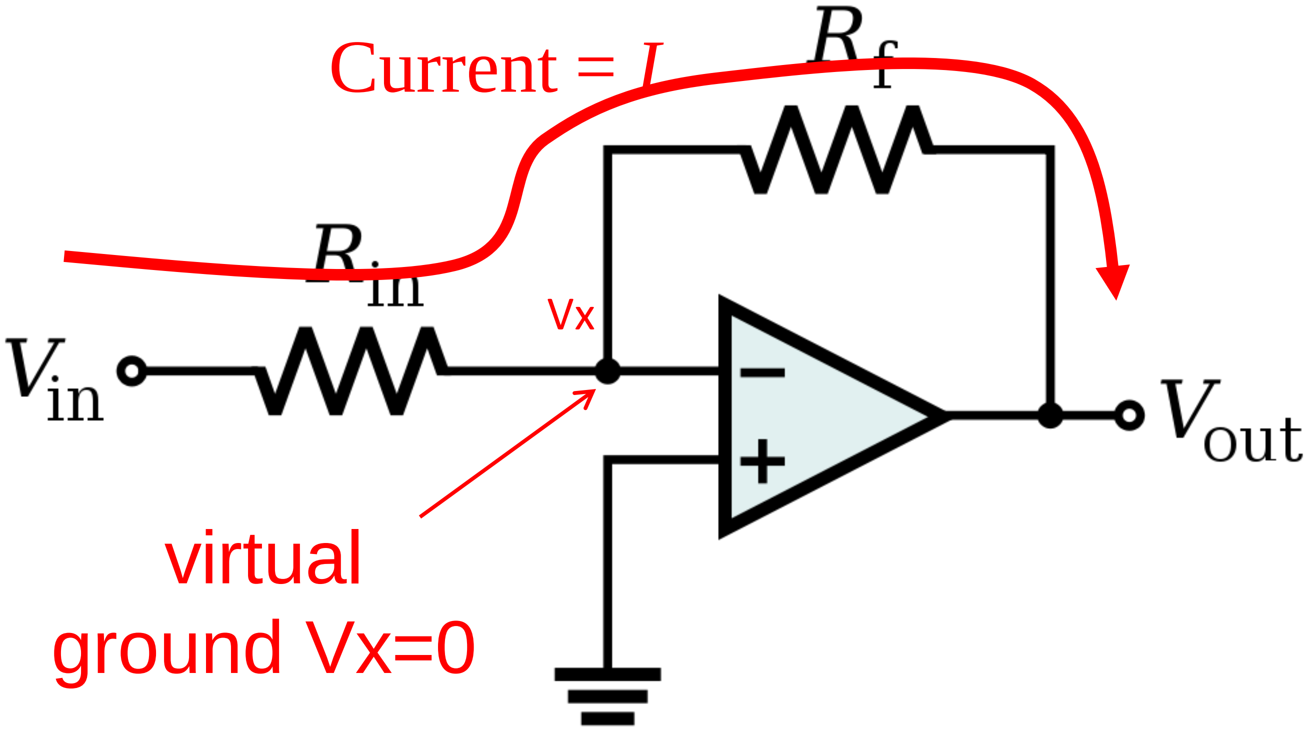
Original idea of OPAMP:
Even tiny difference on +/- will be amplified to infinitely large
Thus, OPerational AMPlifier




To analyze an op-amp feedback circuit:

- Assume no current flows into either input terminal
- Assume no current flows out of the output terminal
- Constrain: $V_+ = V_-$

# Inverting Amplifier Analysis

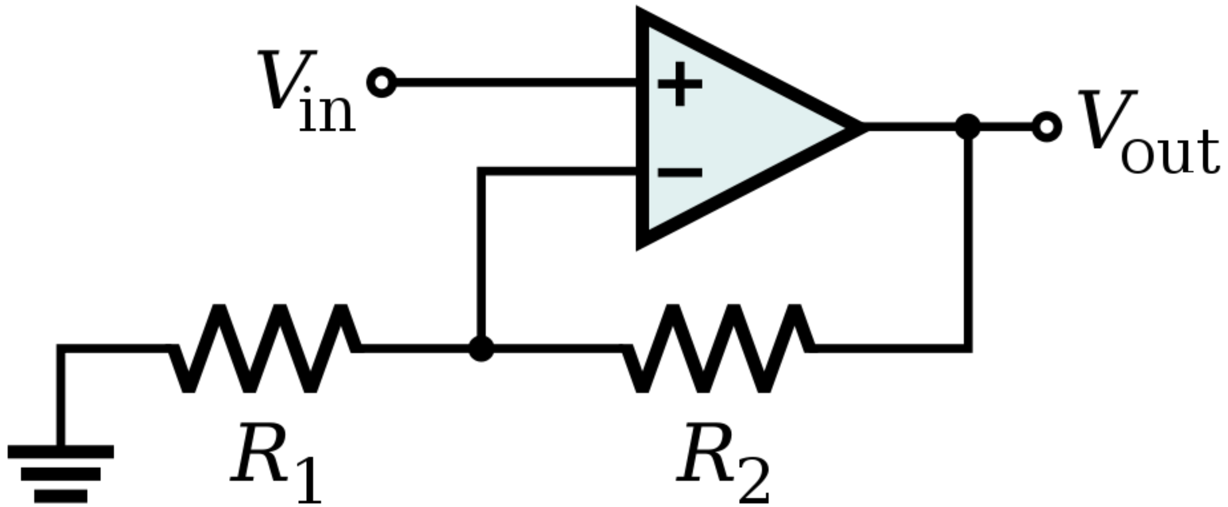



$$I = \frac{V_{in} - V_x}{R_{in}} = \frac{V_x - V_{out}}{R_f}$$

$$V_x = 0$$

$$V_{out} = -\frac{R_f}{R_{in}} V_{in}$$

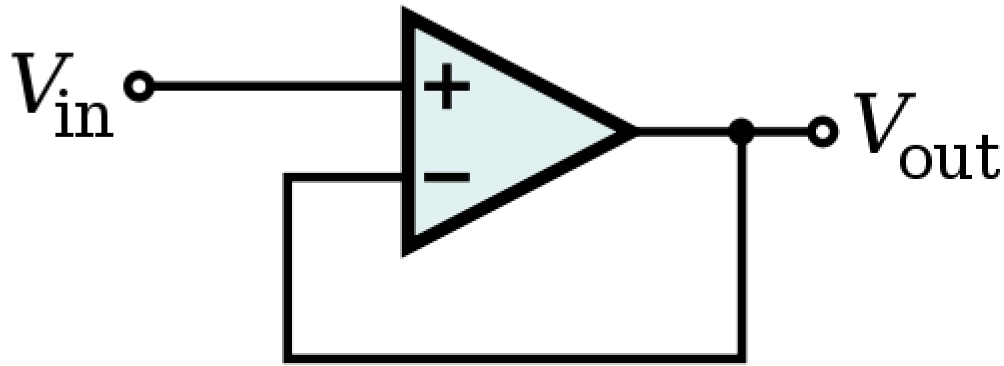
# Non-Inverting Amplifier Analysis



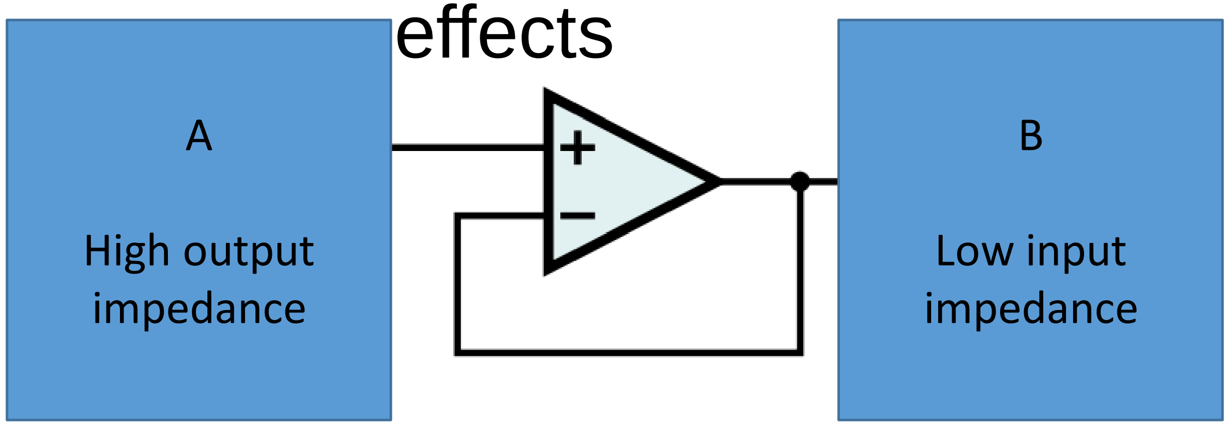

Quiz 1-1
Derivate it
= 3 min.s

$$V_{out} = V_{in}\left(1 + \frac{R_2}{R_1}\right)$$

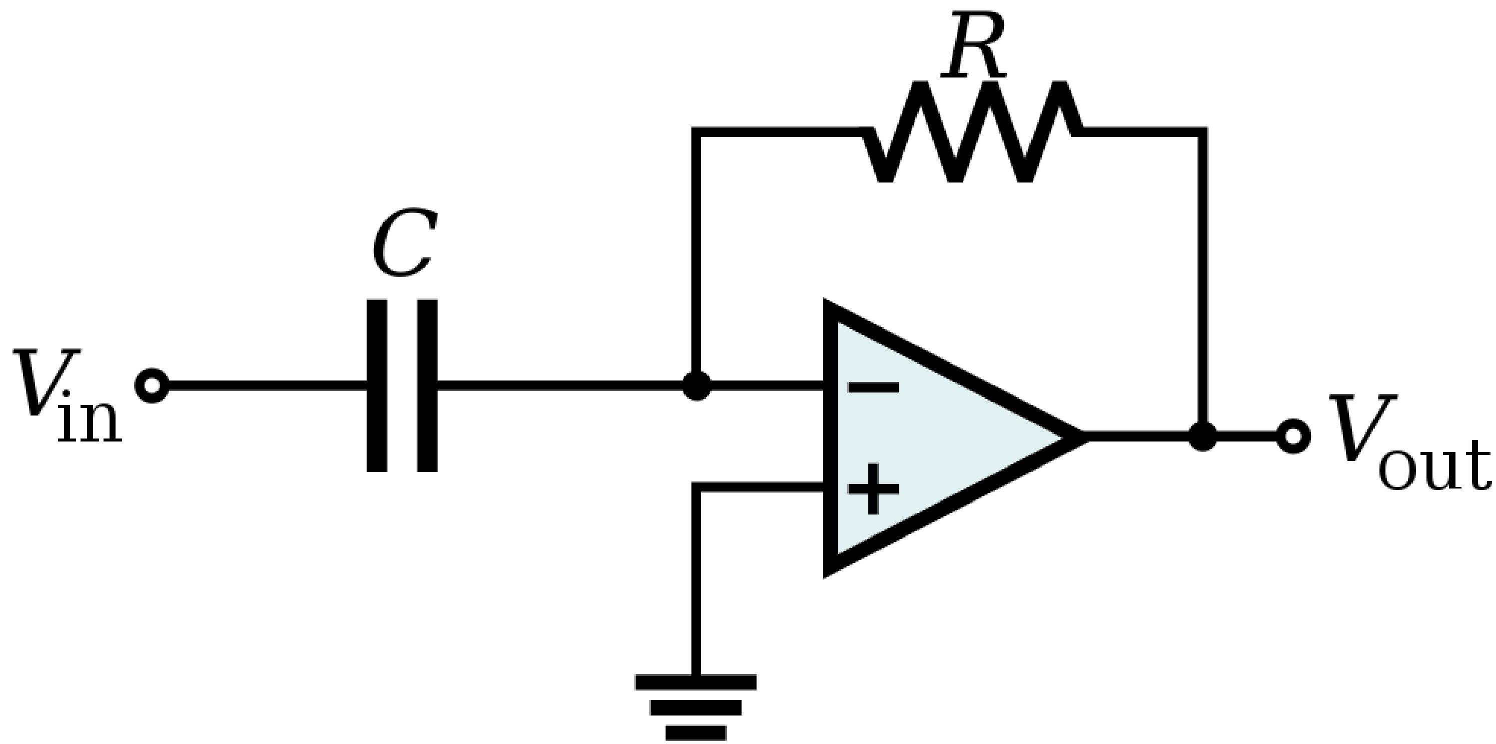
# Op-Amp Buffer




Vout = Vin
Isolates loading effects

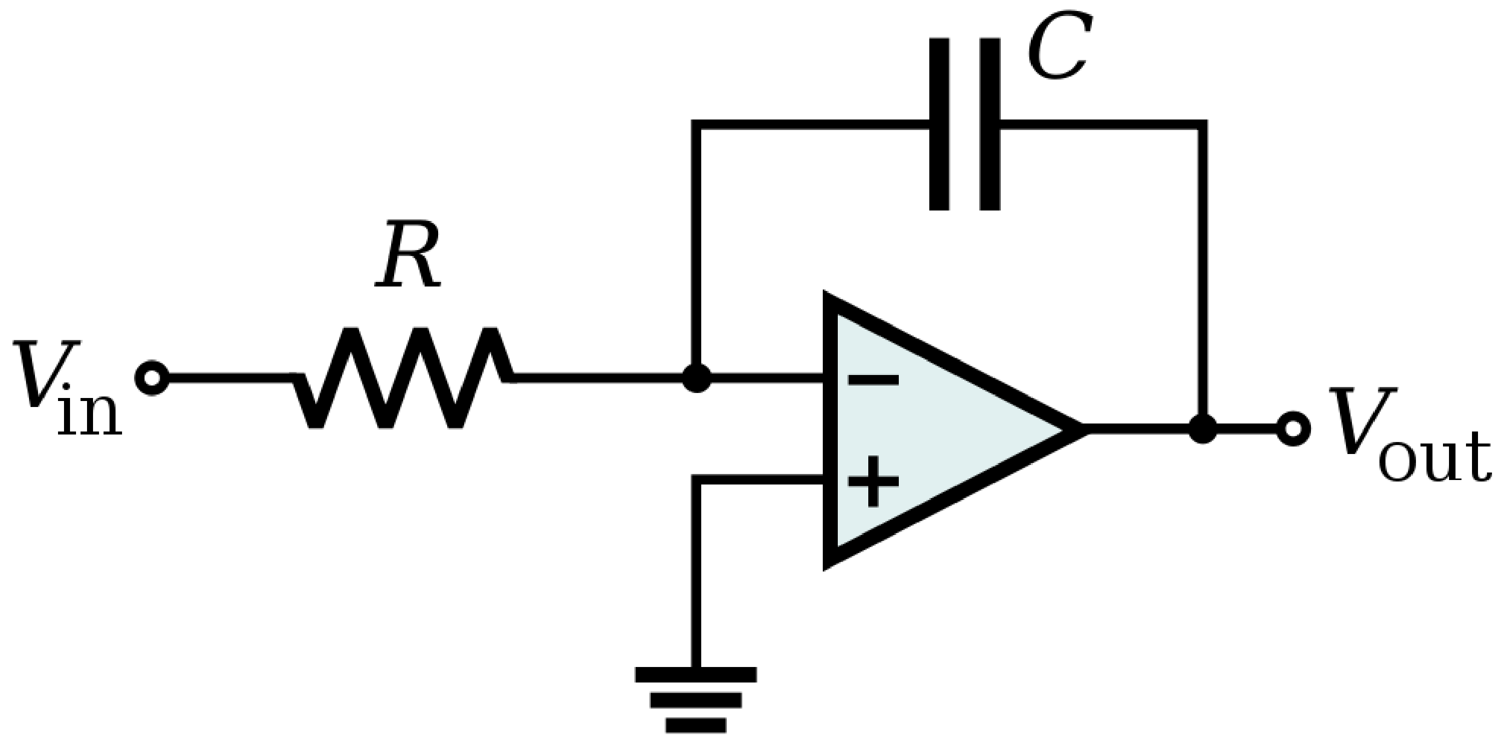


# Op-Amp Differentiator



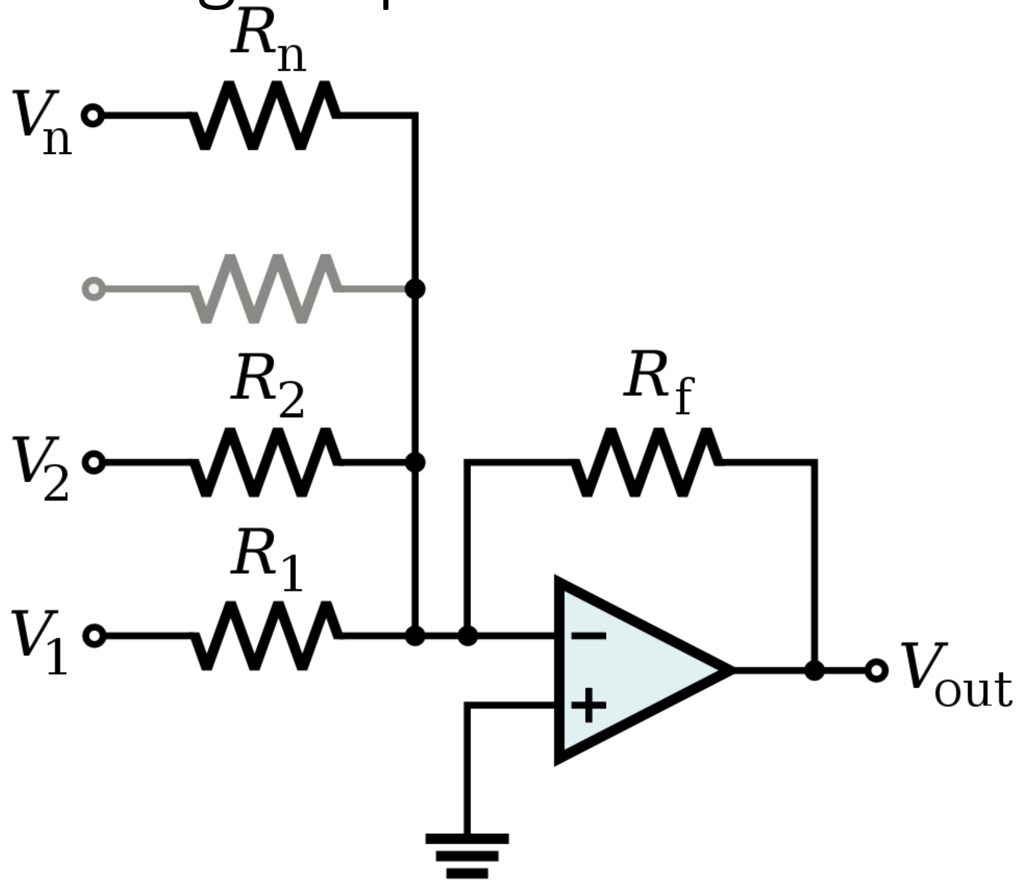

$$V_{\text{out}} = -RC\,\frac{\mathrm{d}V_{\text{in}}}{\mathrm{d}t}$$

# Op-Amp Integrator



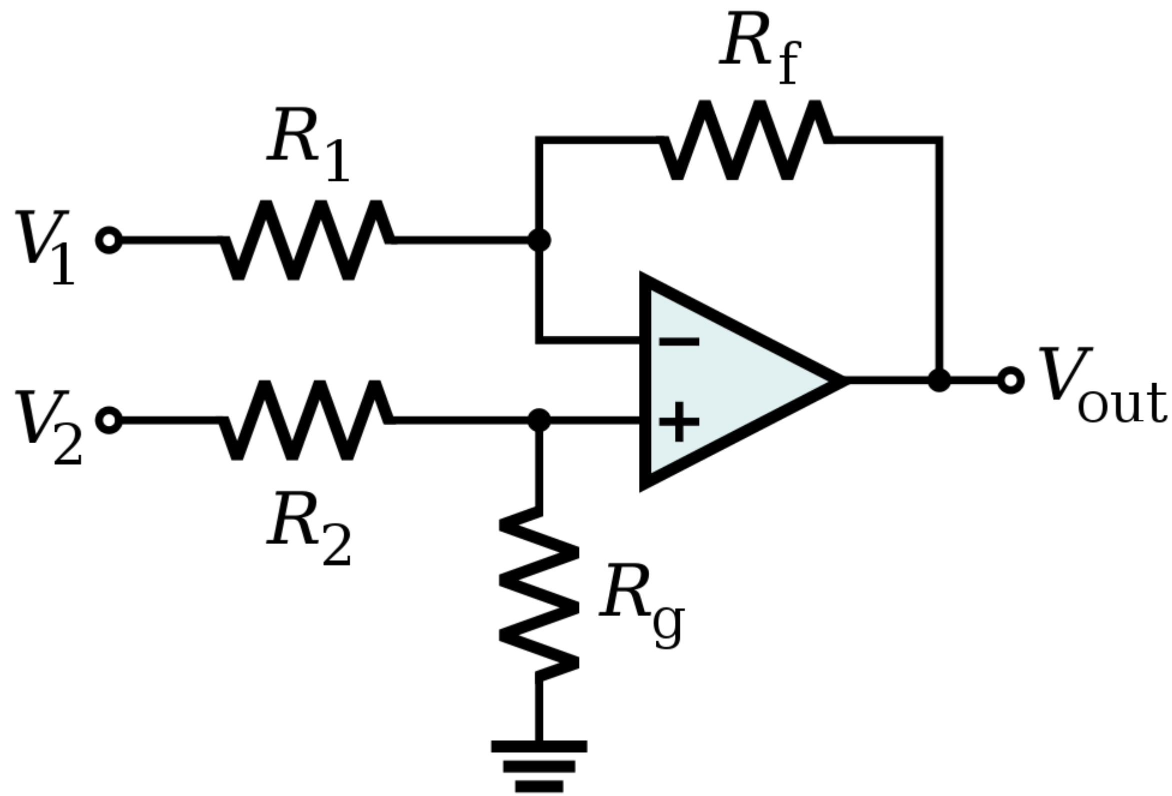

$$V_{\mathrm{out}} = -\int_0^t \frac{V_{\mathrm{in}}}{RC}\,\mathrm{d}t + V_{\mathbf{initial}}$$

# Op-Amp Summing Amplifier




$$V_{\text{out}} = -R_{\text{f}} \left( \frac{V_1}{R_1} + \frac{V_2}{R_2} + \cdots + \frac{V_n}{R_n} \right)$$

# Op-Amp Differential Amplifier




$$V_{out} = \frac{(R_f + R_1)\,R_g}{(R_g + R_2)\,R_1} V_2 - \frac{R_f}{R_1} V_1$$

If $R_1 = R_2$ and $R_f = R_g$: $$V_{out} = \frac{R_f}{R_1}(V_2 - V_1)$$

# Problem of this

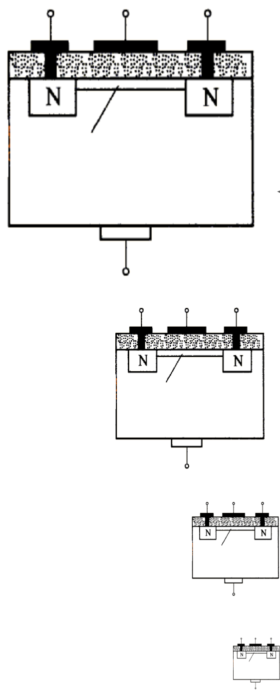
- Please remember, in analog circuits
  1. Noise
  2. Mismatch (what you get is NEVER what you expect)
  3. Static power
  4. You can almost NEVER store info.
  5. Not programmable

# Outline

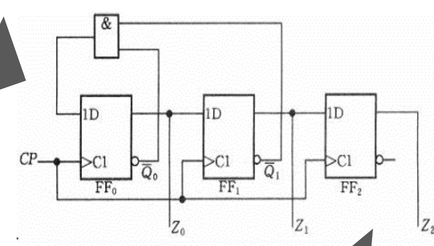
- What/Why/Why_not is analog computing
- Classic textbook type analog computing: OPAMP-based
- Oscillation driving analog
- Machine learning driving analog

# Oscillation-based Computing

*Plain analog circuits are not the hero of computational VLSIs.*

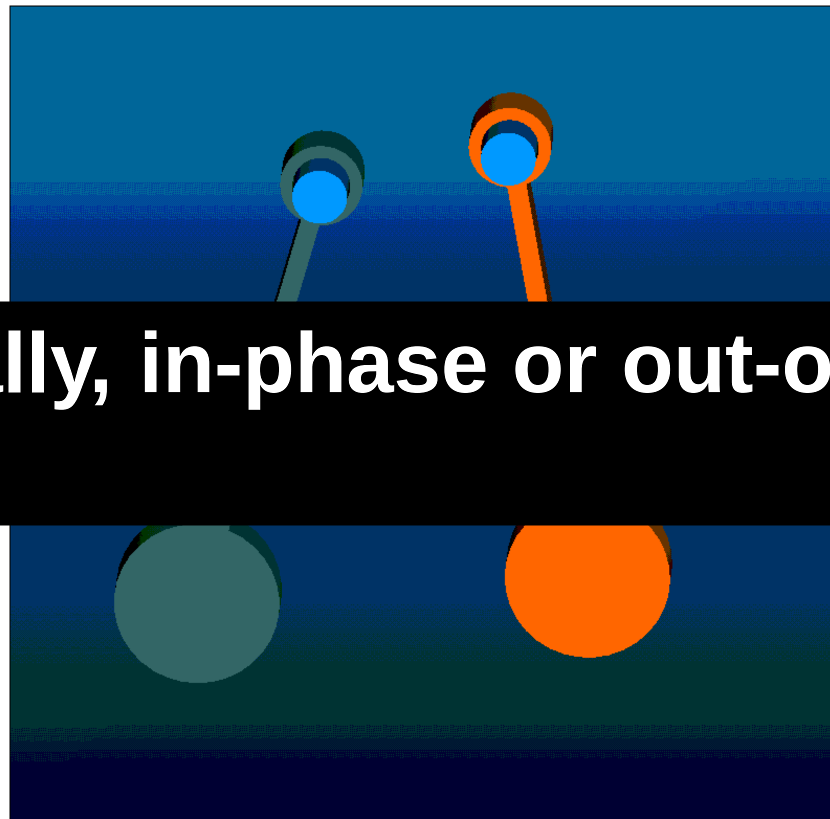



**What's the next?**




# Oscillation-based Computing

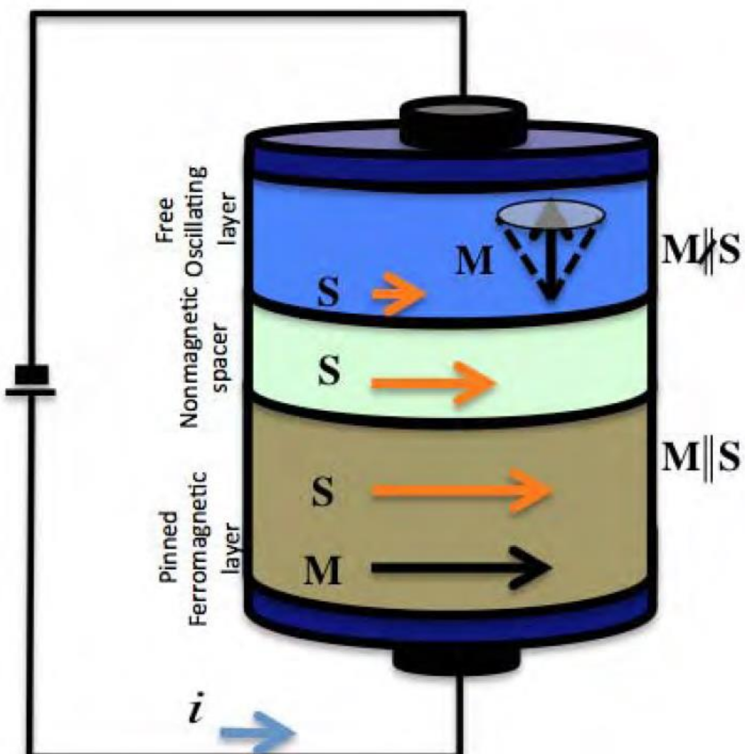
*Synchronization:*

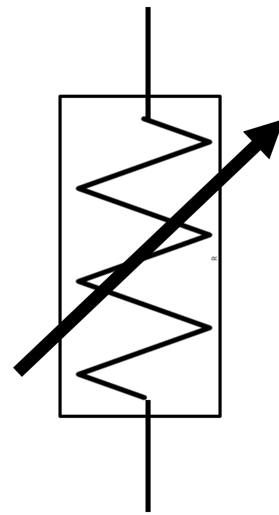


**Finally, in-phase or out-of-phase**

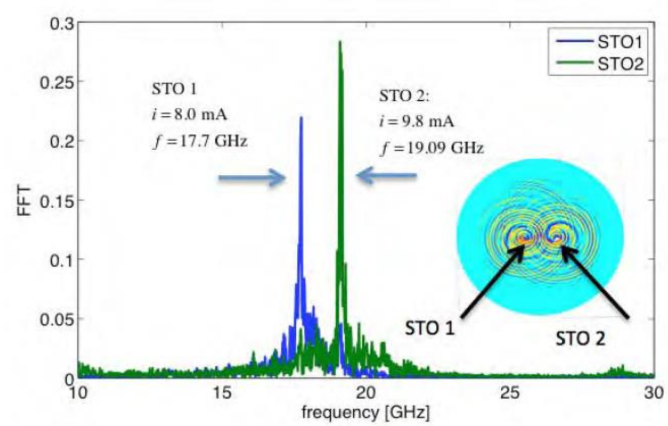
# Oscillation-based Computing

*Spin Torque Oscillator (STO)*



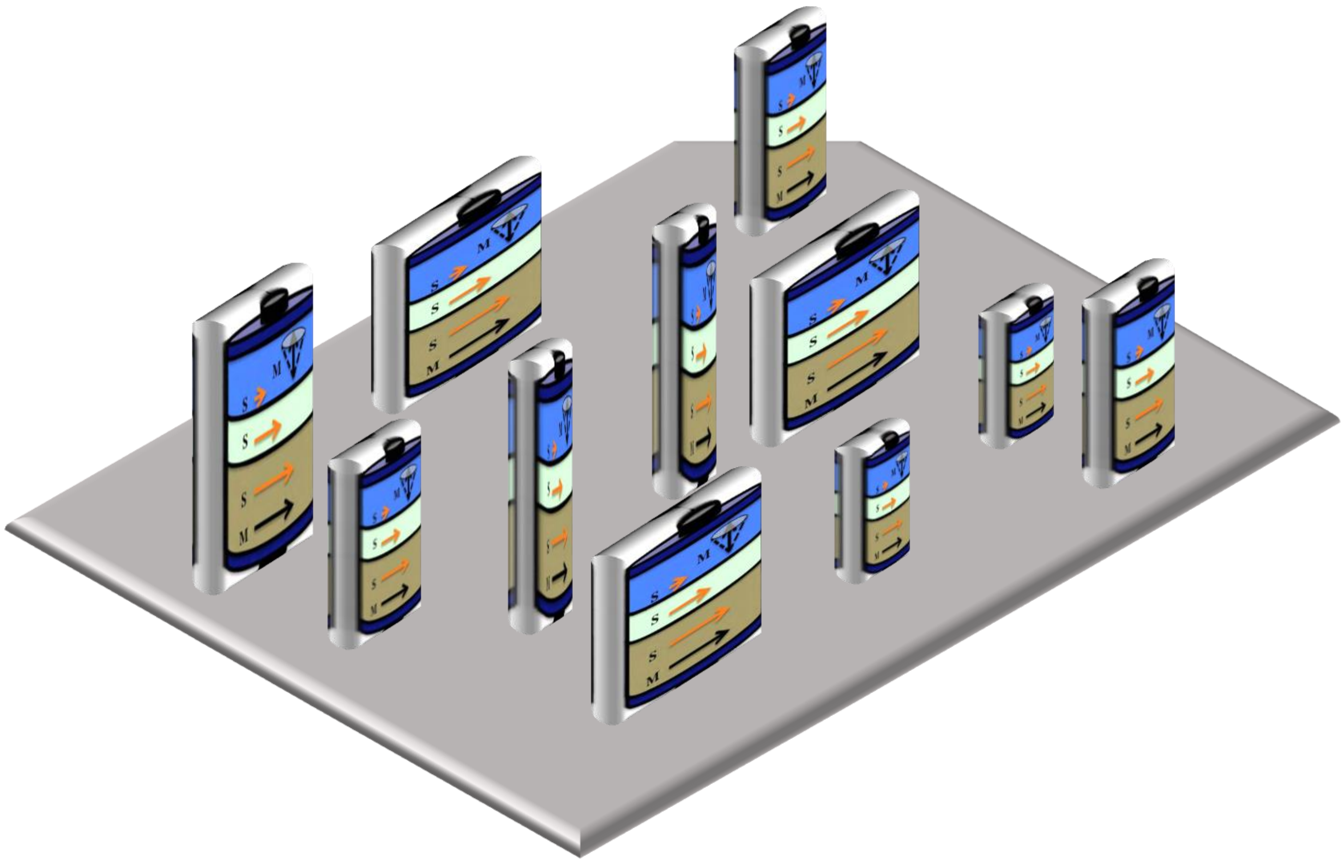



$R=R0(1+\sin(\omega t))$



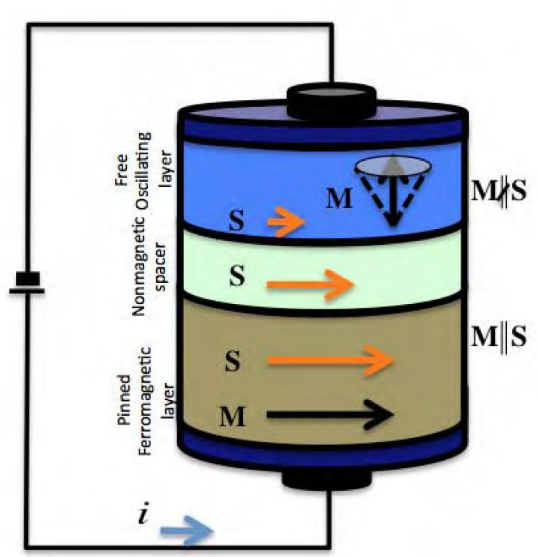
# Oscillation-based Computing

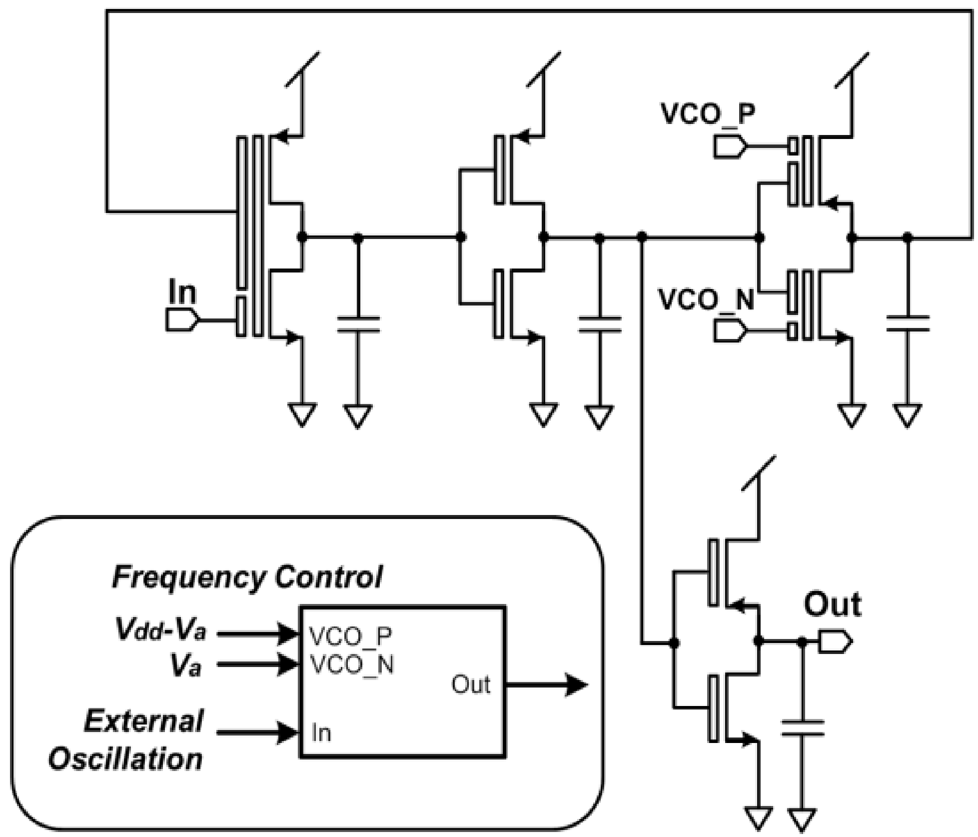
*What happens when we interfere millions ~*



# Oscillation-based Computing

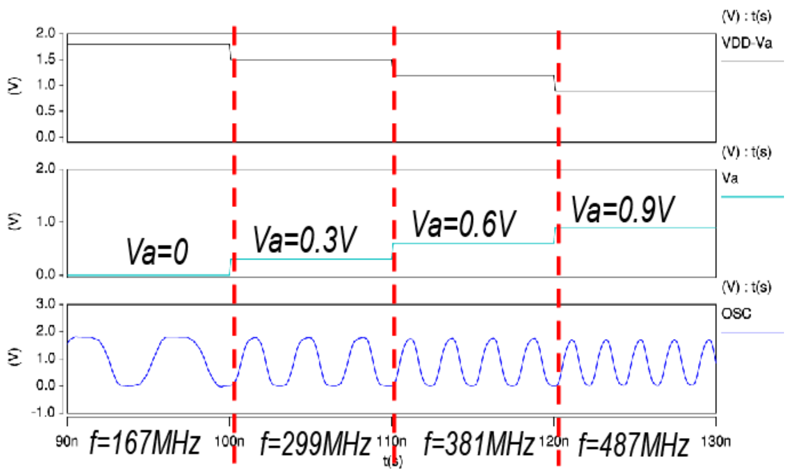
*Emulate the behavior of STO*

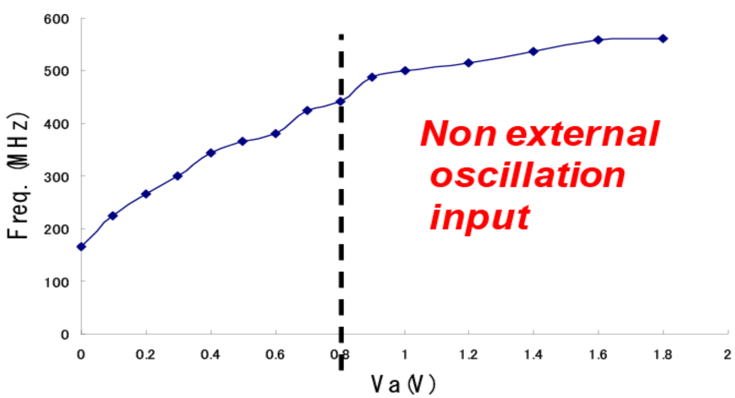


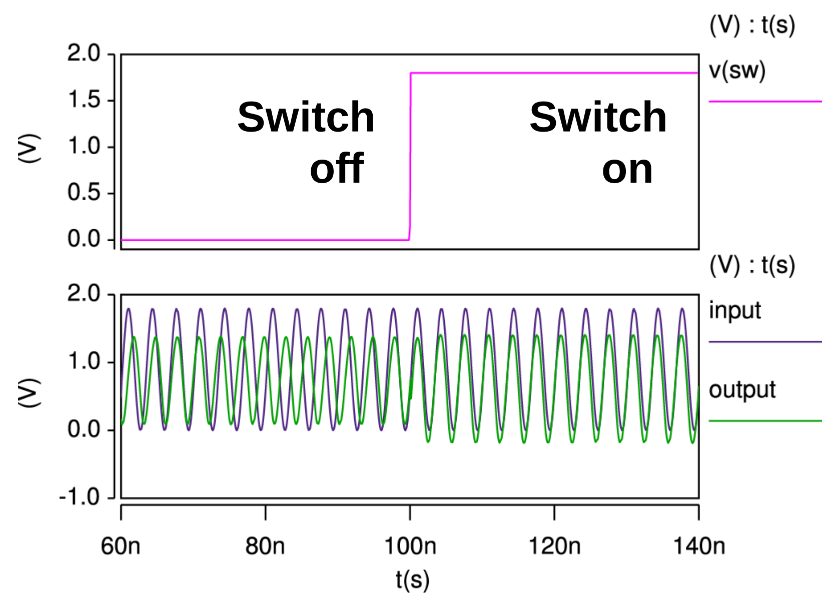



# Oscillation-based Computing

*Emulate the behavior of STO*
*spin torque oscillator (STO)*



# Oscillation-based Computing

*How to use oscillation for recognition problems?*

**Phase Keying Scheme**

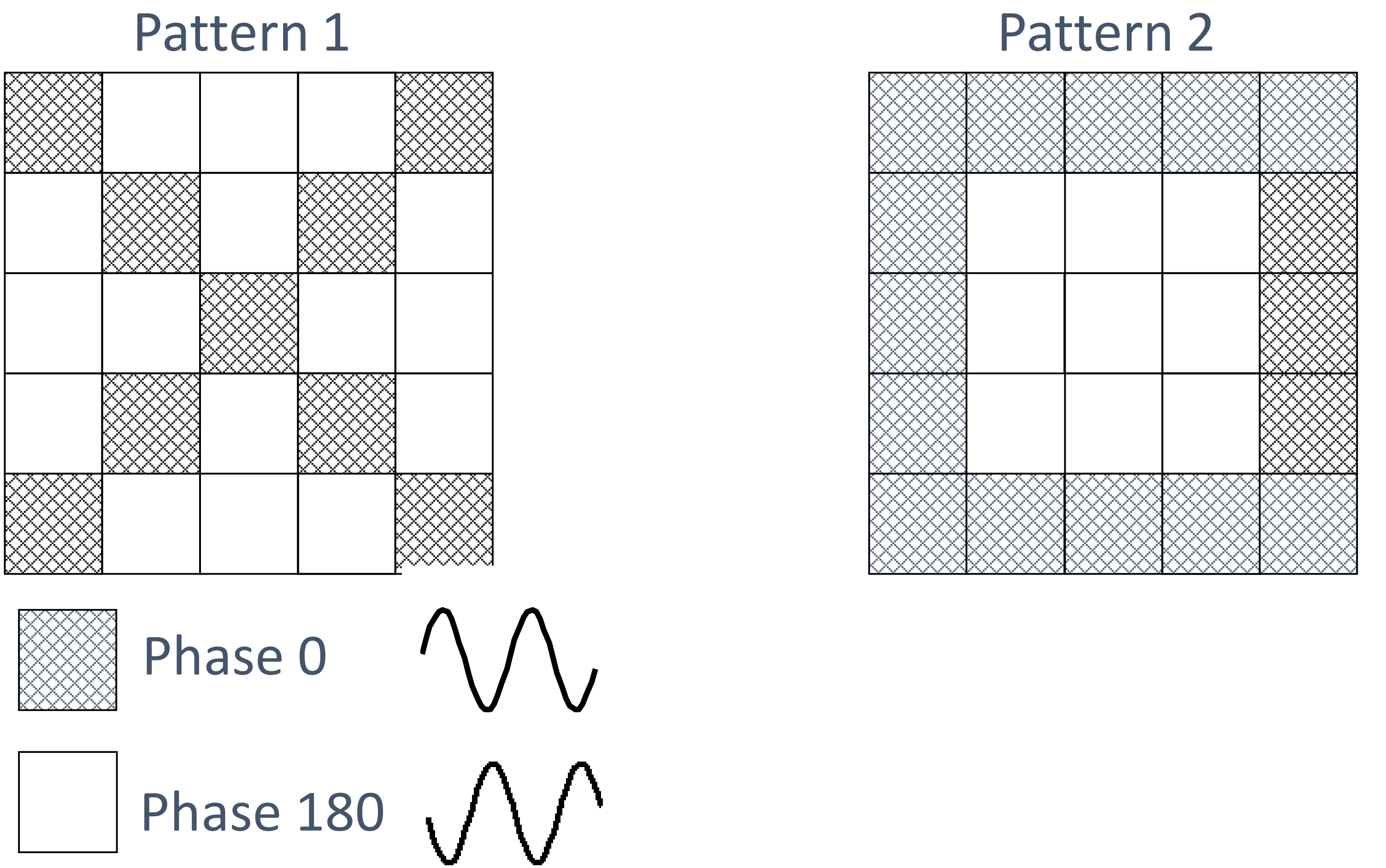
*phases are used to represent information*

**Frequency Keying Scheme**

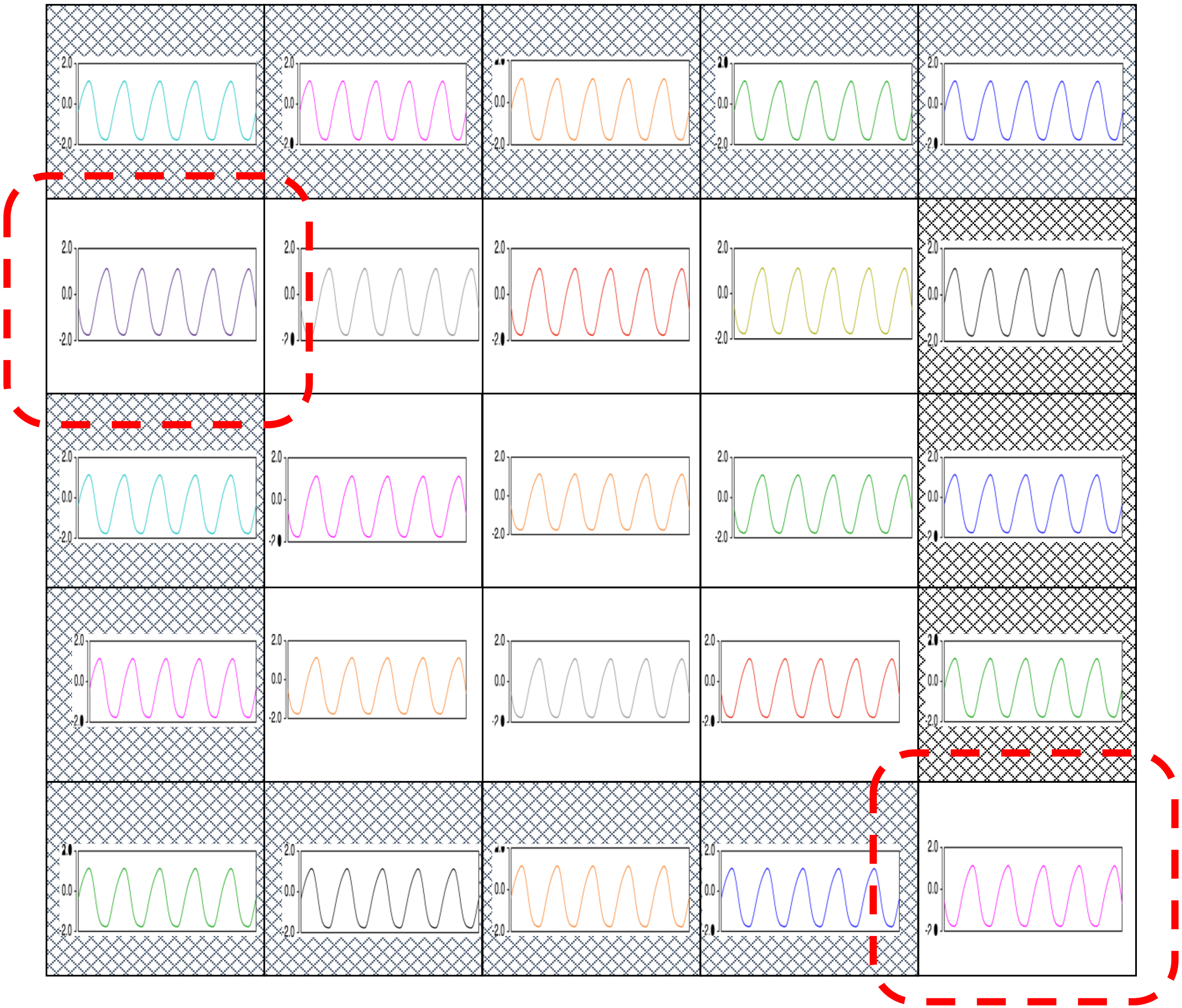
*frequencies are used to represent information*

# Phase Keying Scheme

## To represent a “pattern”

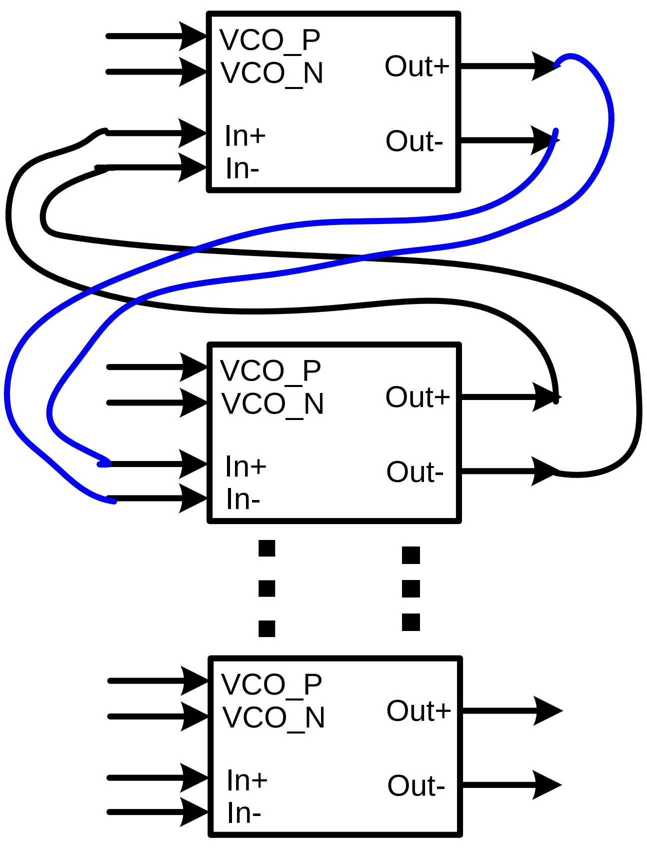


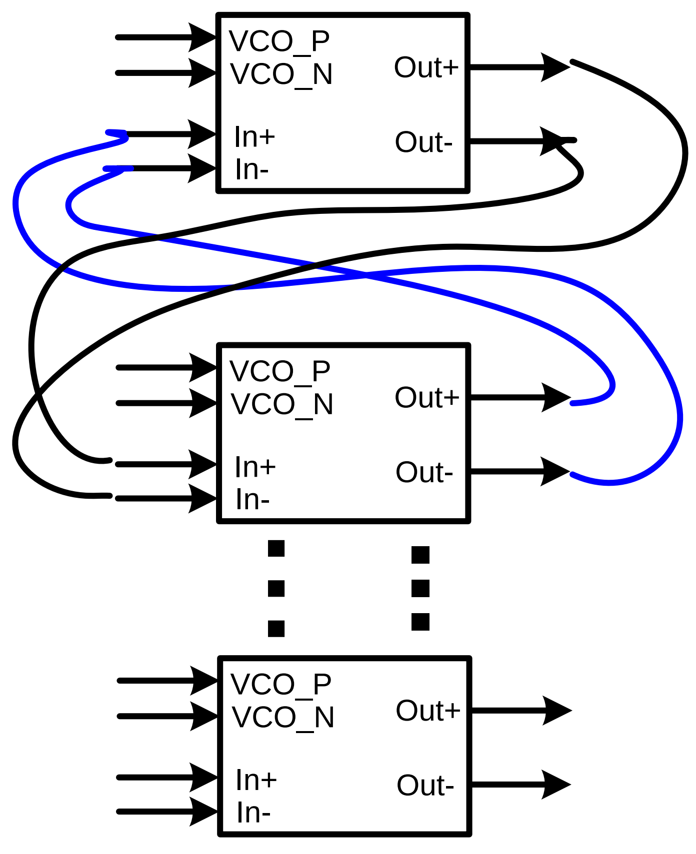
# Phase Keying Scheme



# Phase Keying Scheme

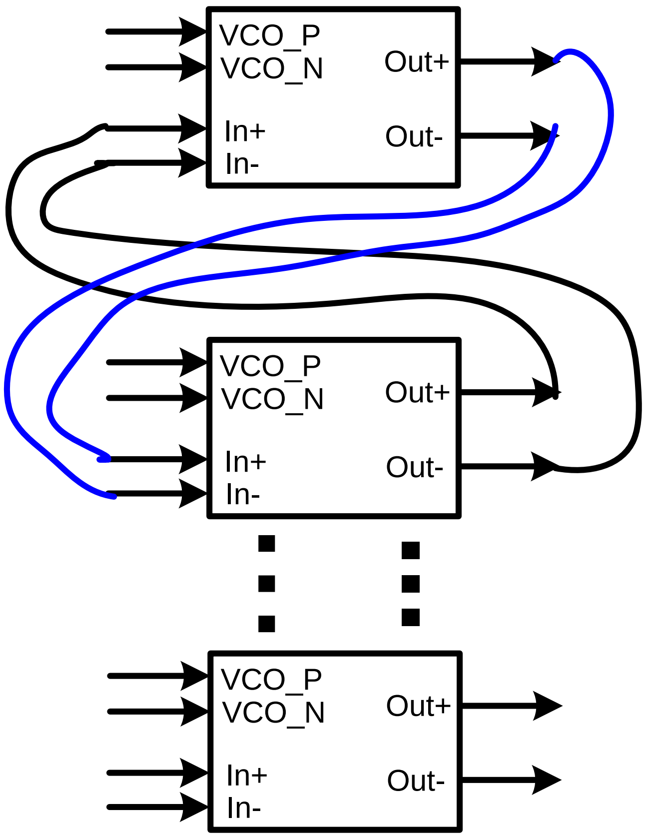
## To memorize a “sample”

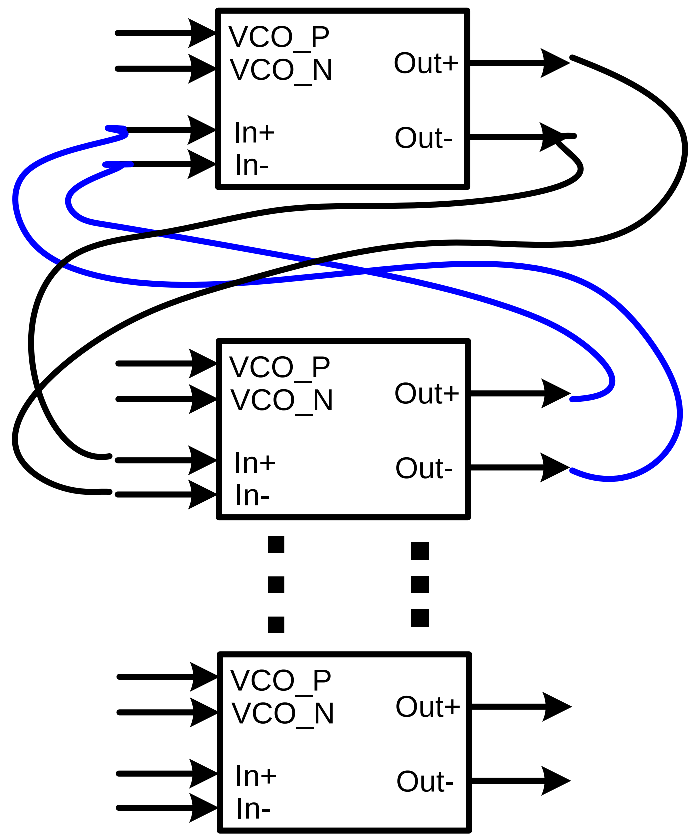


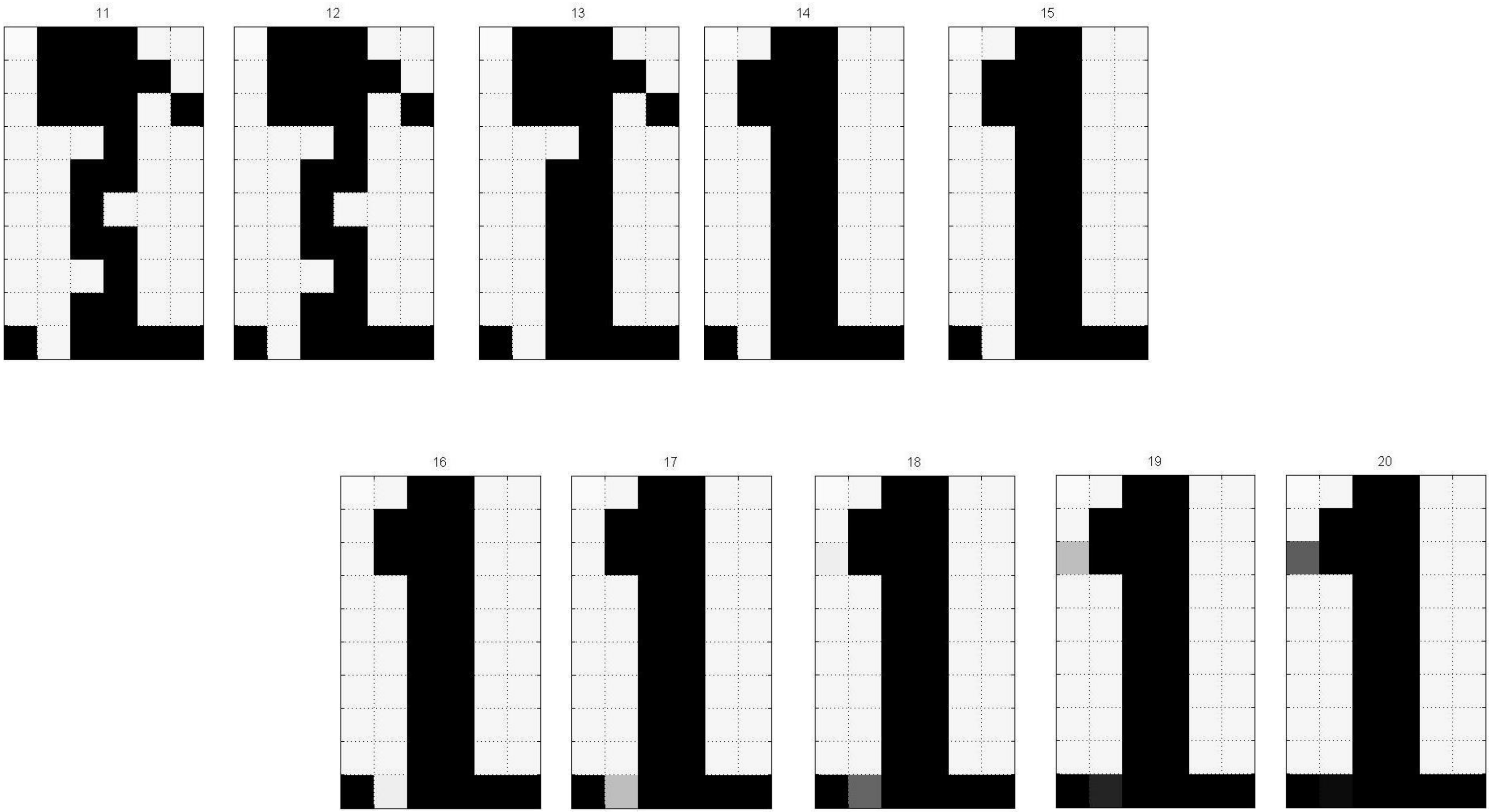
# Phase Keying Scheme

## To memorize a “sample”



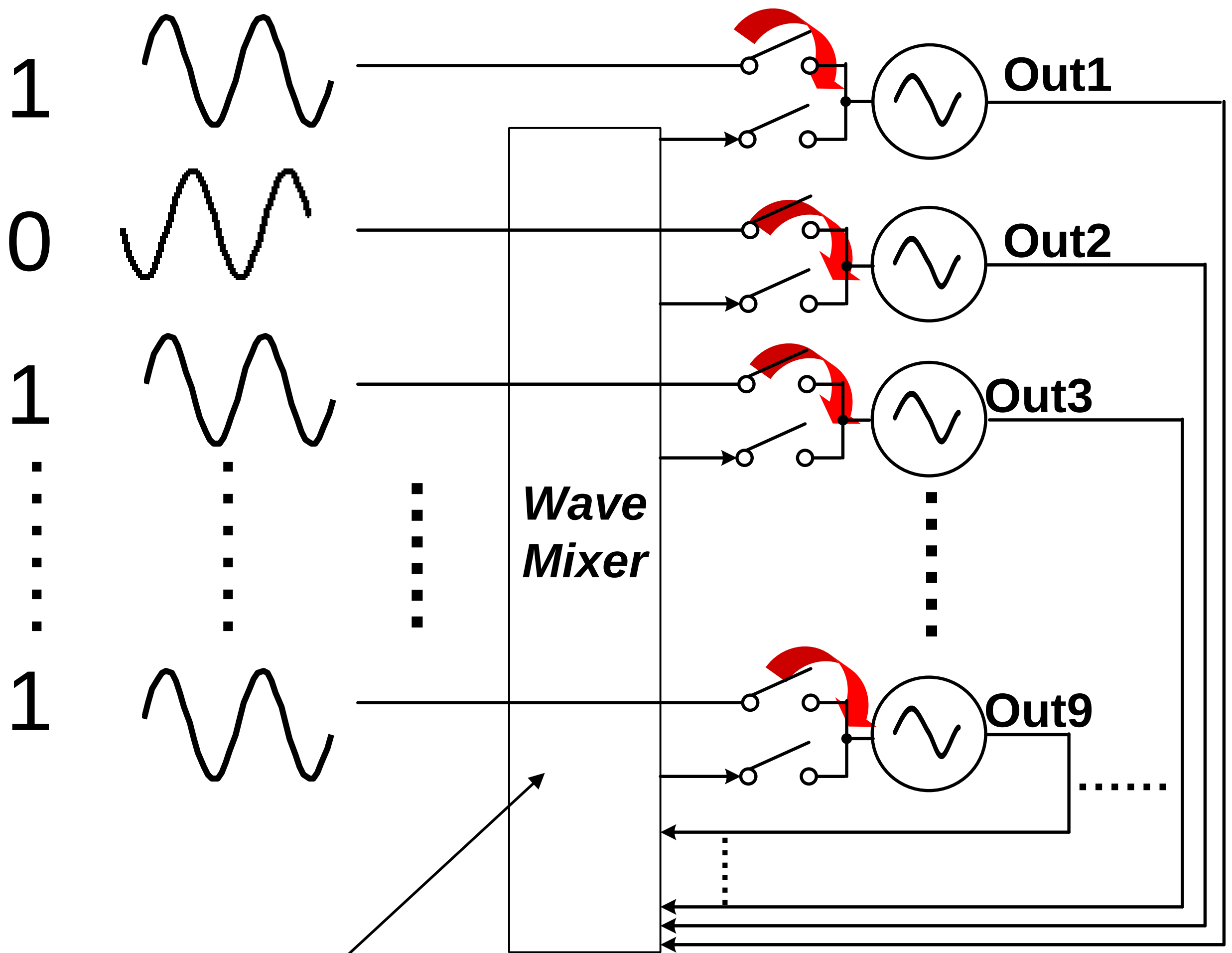



# Phase Keying Scheme



Network

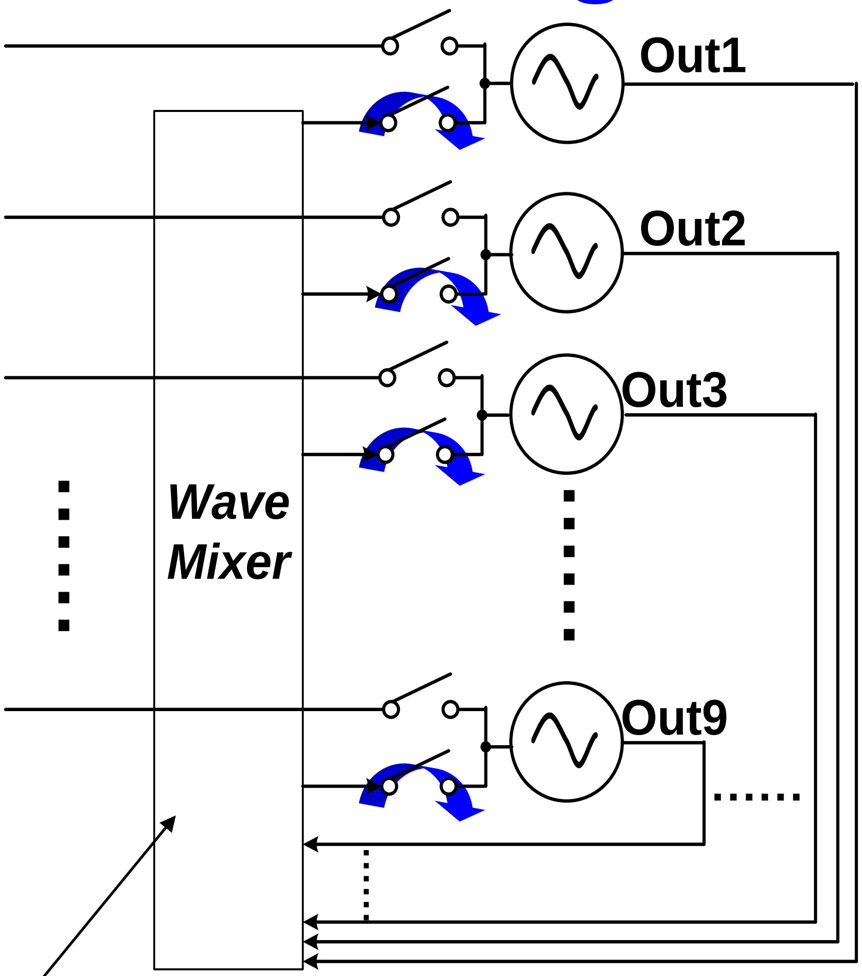
# Step 1: Read in input




*Template memorized here. (two templates, 25-bit for each one)*

# Network

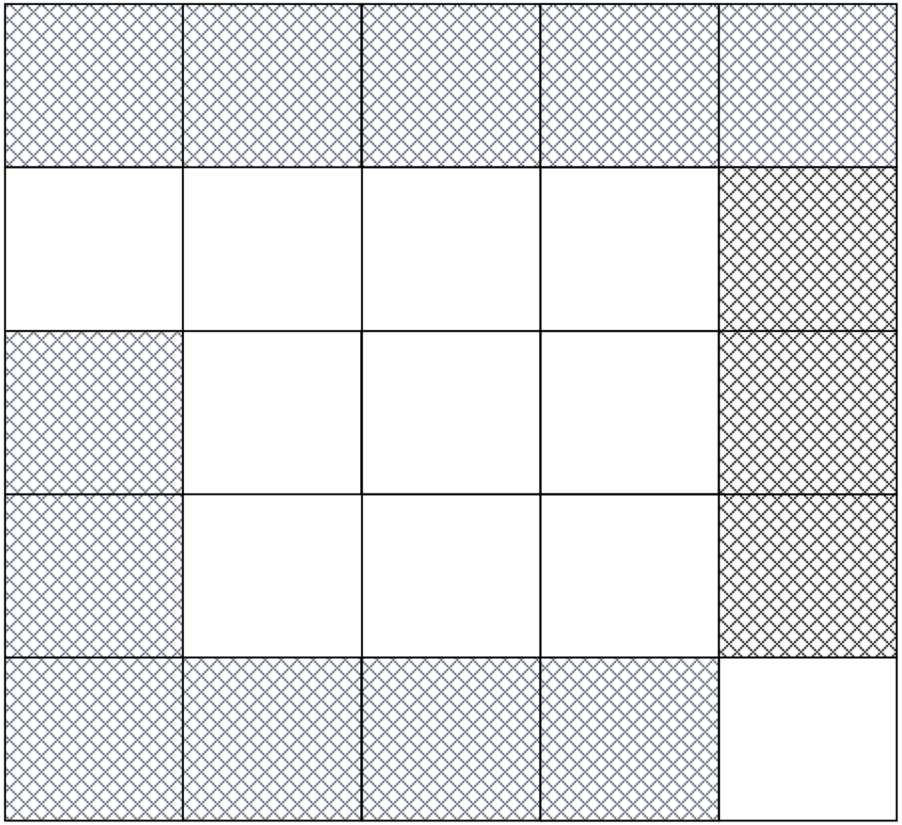
## Step 2: Wave mixing



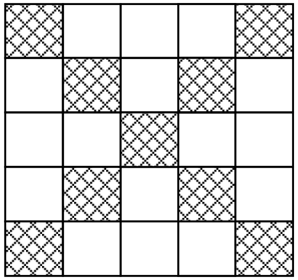

*Template memorized here. (two templates, 25-bit for each one)*

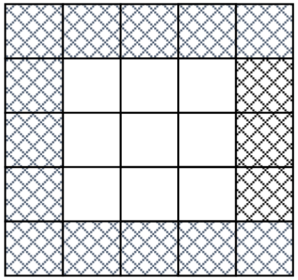
# Test

Input

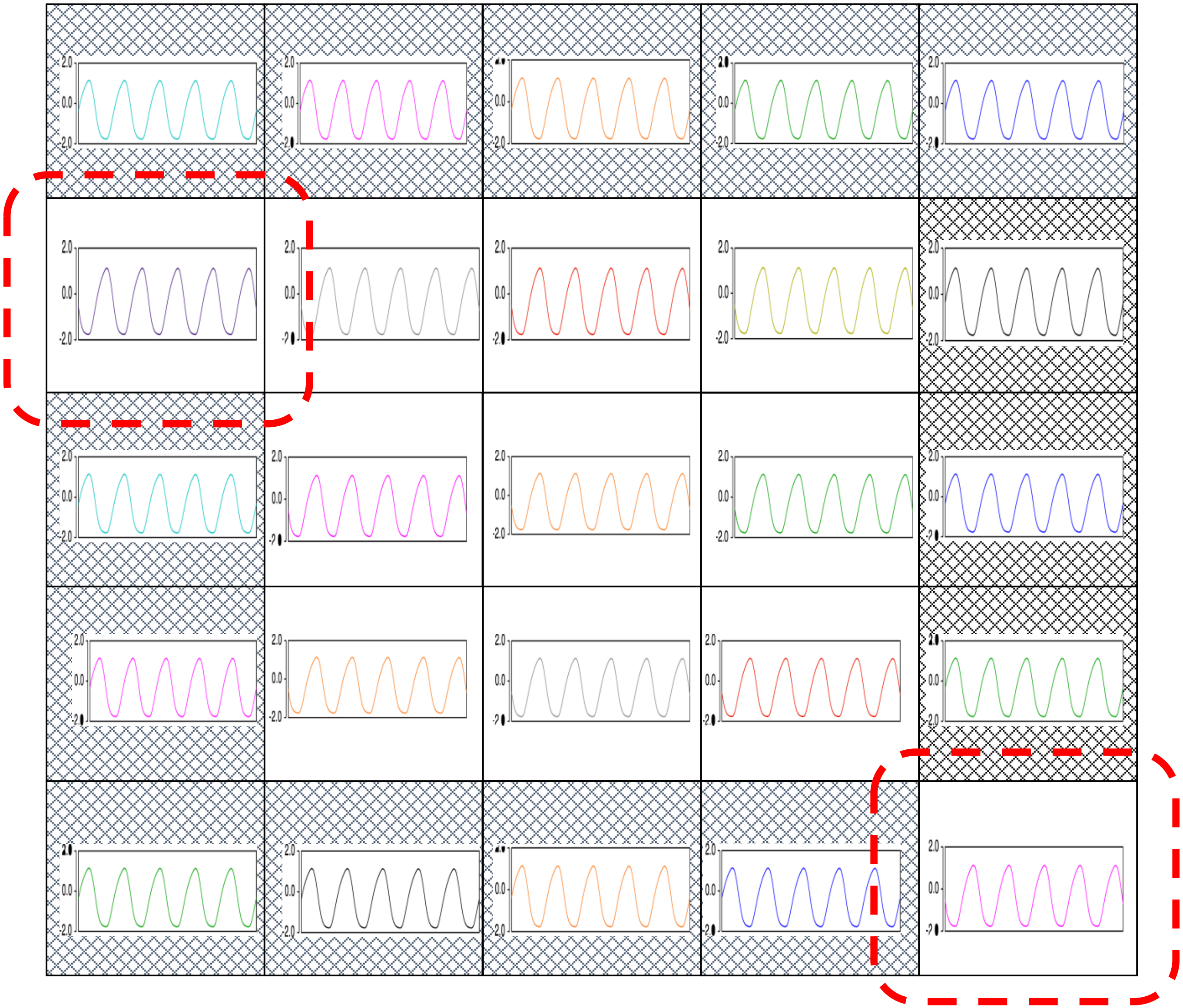


Templates

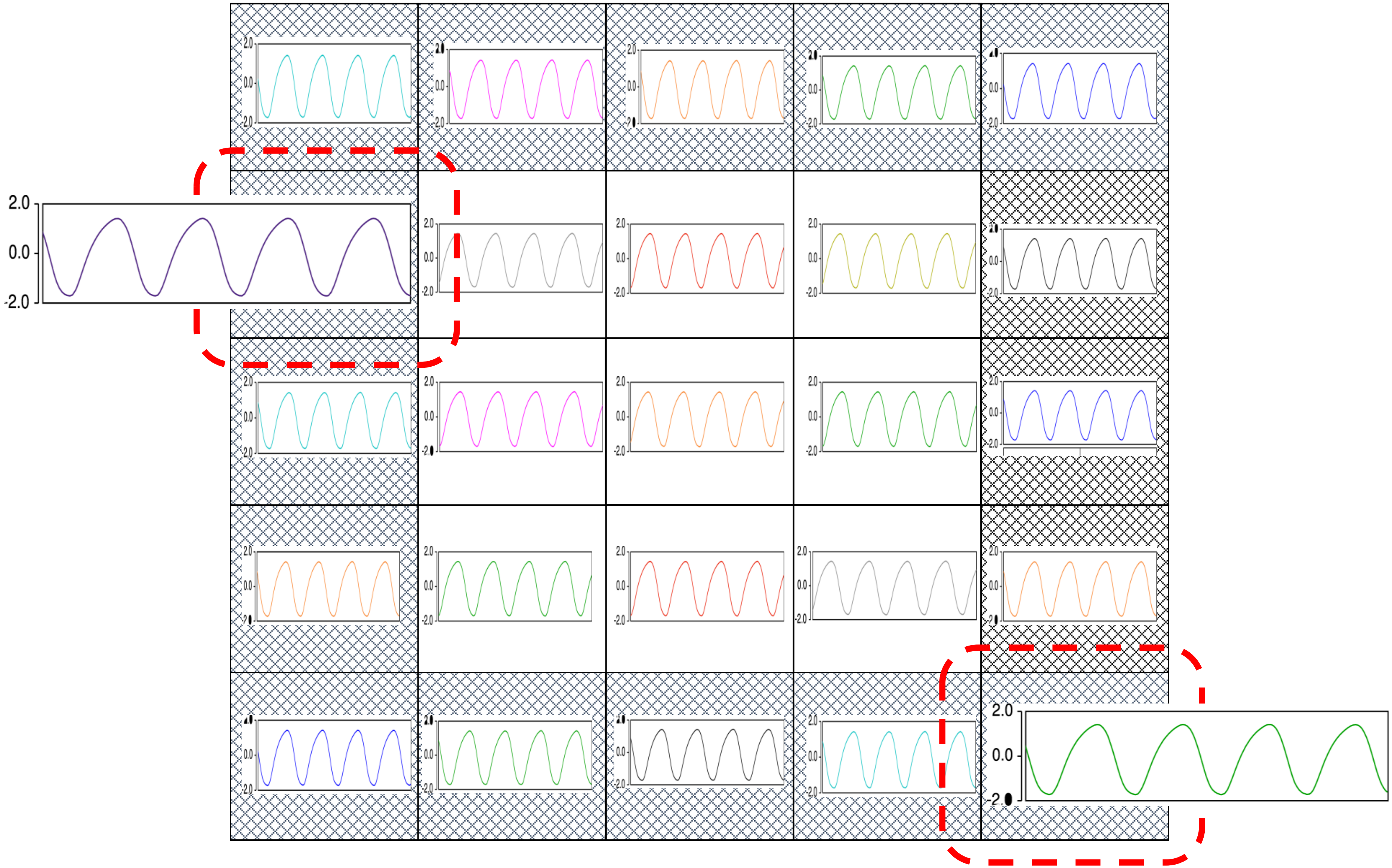




# Initial (read in the input)

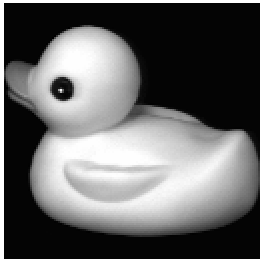


# Result (phase shifted by the mixer)



# Frequency Keying Scheme

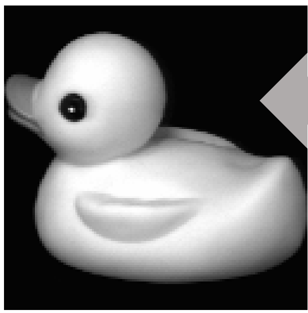
To represent a “pattern”, by frequency



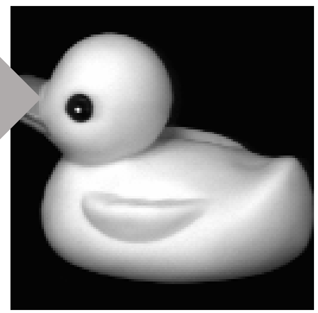
| | | |
|---|---|---|
| *129MHz* | *9MHz* | *12MHz* |
| *37MHz* | *229MHz* | ⋮ |
| *98MHz* | | |
| …… | | …… |

# Frequency Keying Scheme

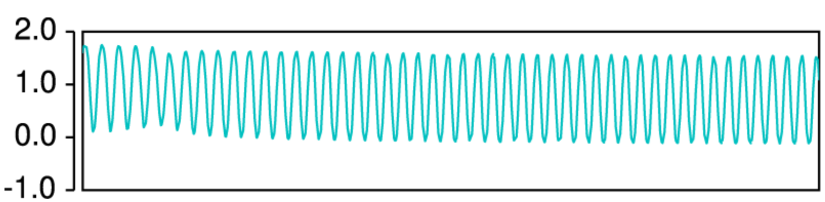
To match a “pattern”, by frequency

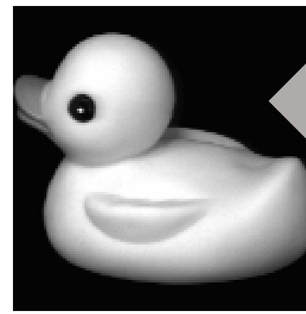


minus

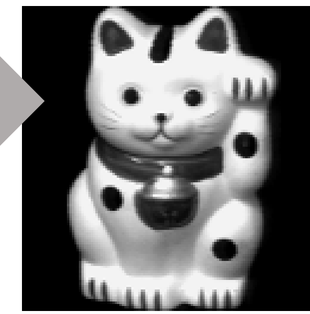


All mixed

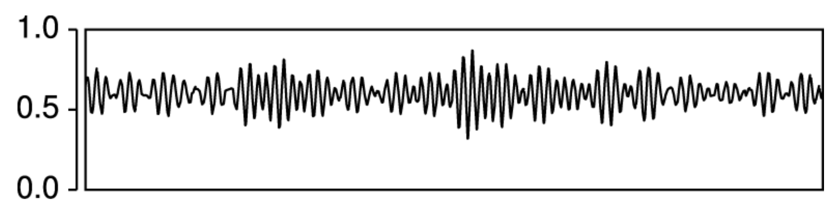


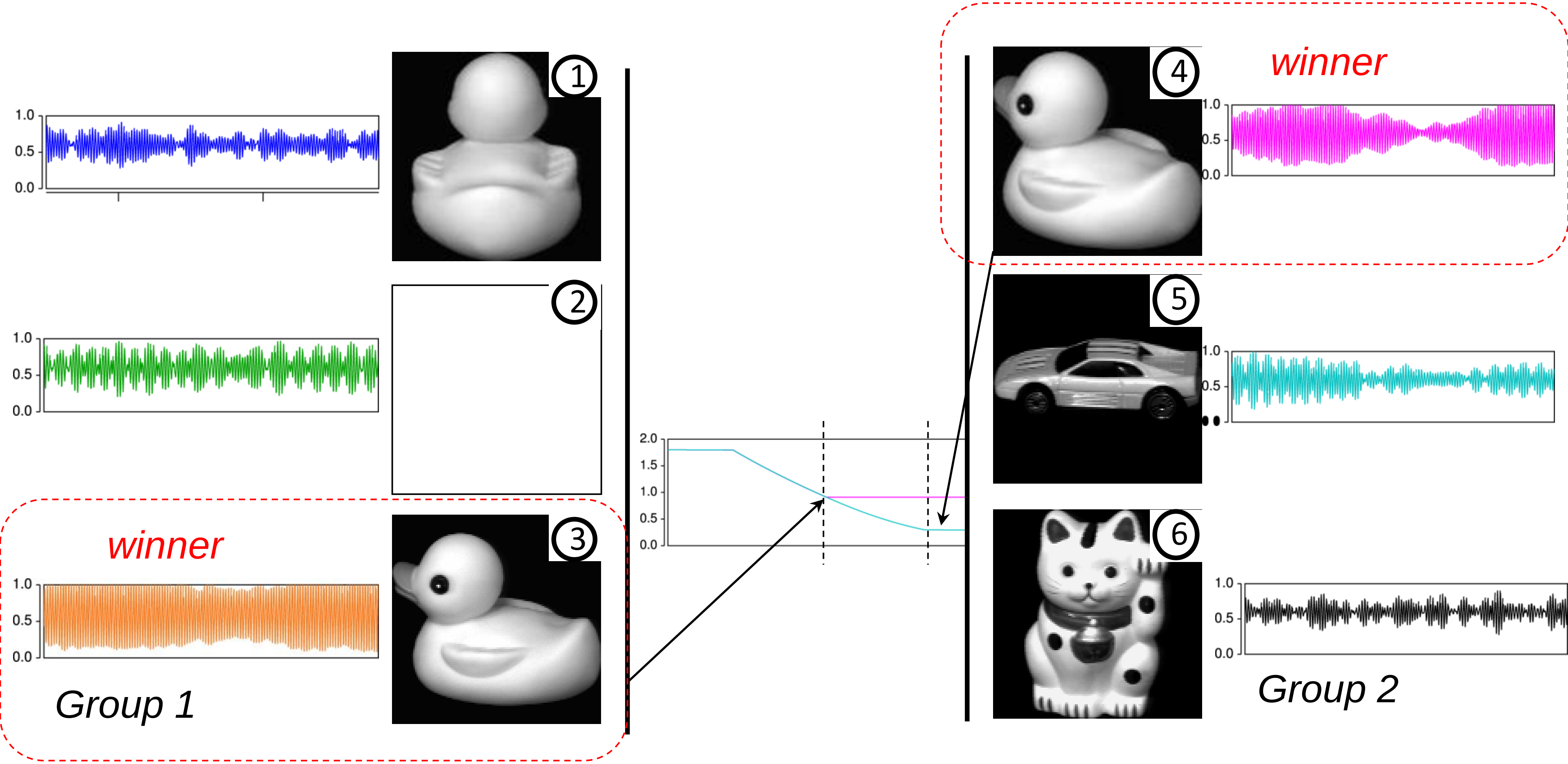
minus



All mixed



# Result

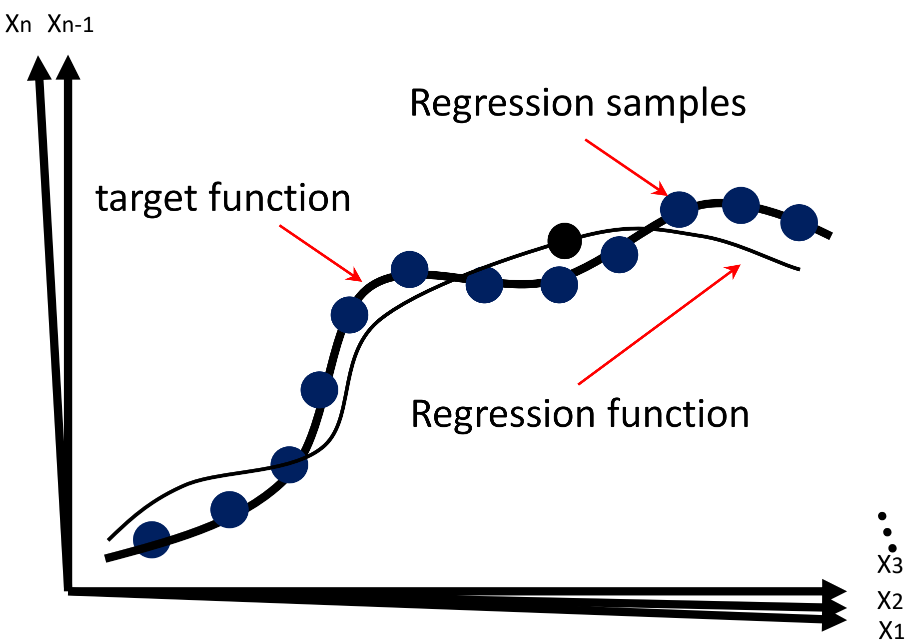


# Outline

- What/Why/Why_not is analog computing
- Classic textbook type analog computing: OPAMP-based
- Oscillation driving analog
- Machine learning driving analog

# **ML driving:** Programmable Analog Calculation Unit (ACU)

Complex functions: two expensive to exactly calculate → retrieve them by regression




Step 1: sampling
take samples from target function

$$(x_1, x_2, \dots x_{n-1}, x_n) = (0,0,\dots,0,0) \qquad f = 0.1$$
$$(x_1, x_2, \dots x_{n-1}, x_n) = (0.1,2,\dots,1.6,3) \qquad f = 2.1$$
$$\vdots \qquad \vdots$$
$$(x_1, x_2, \dots x_{n-1}, x_n) = (1,3,\dots,0.8,0) \qquad f = 0.9$$

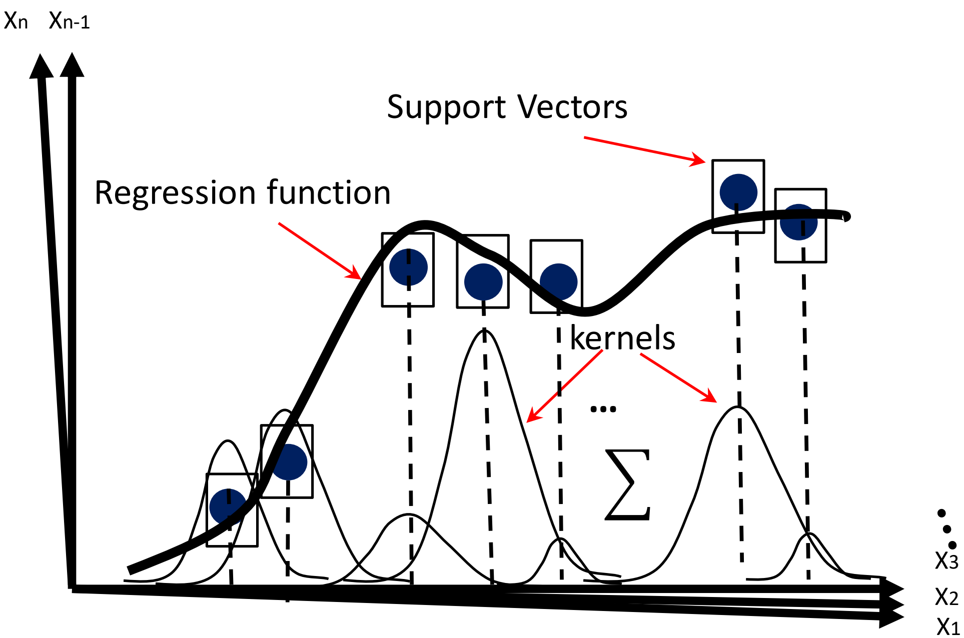
Step 2: learning
Construct the regression network
by using the samples

Step 3: use
receive any new variable, predict result

It is special scheme of regression:
Function is known

# **ML driving:** Programmable Analog Calculation Unit (ACU)

SVR (with Gaussian kernel) is high performance regression algorithms




Function is retrieved by combination of kernels around SVs

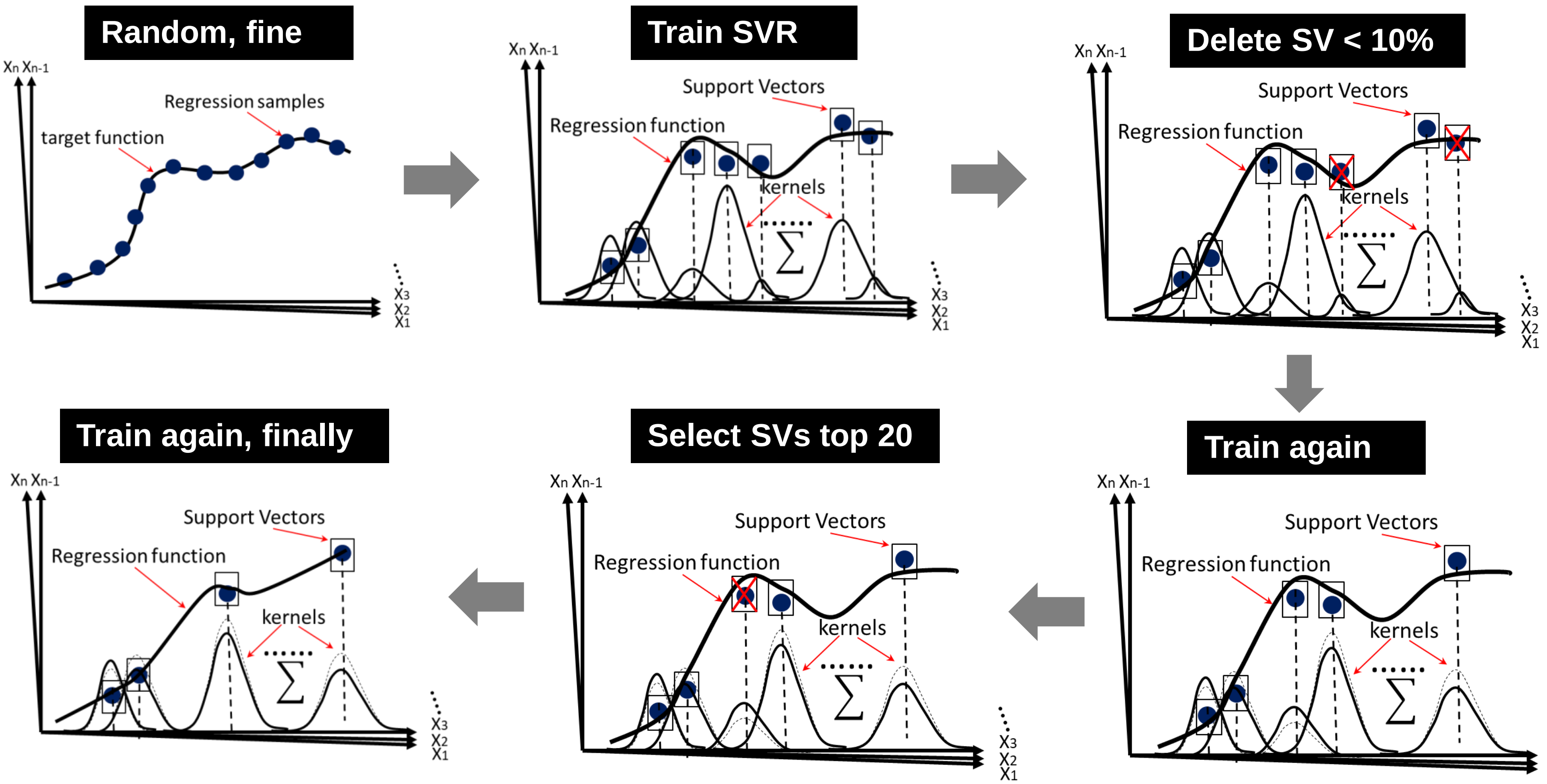
$$f(\mathbb{X}) = \sum \underbrace{(\alpha_i^* - \alpha_i)}_{\text{Weight}} \, e^{-\gamma(\underbrace{\mathbb{X}}_{\text{New variable}} - \underbrace{\mathbb{SV}_i}_{\text{SVs}})^2}$$

Only key samples remains, called "Support Vectors" (SVs)

We just need to know:
which ones are SVs
Corresponding $(\alpha_i^* - \alpha_i)$

# **ML driving:** Programmable Analog Calculation Unit (ACU)

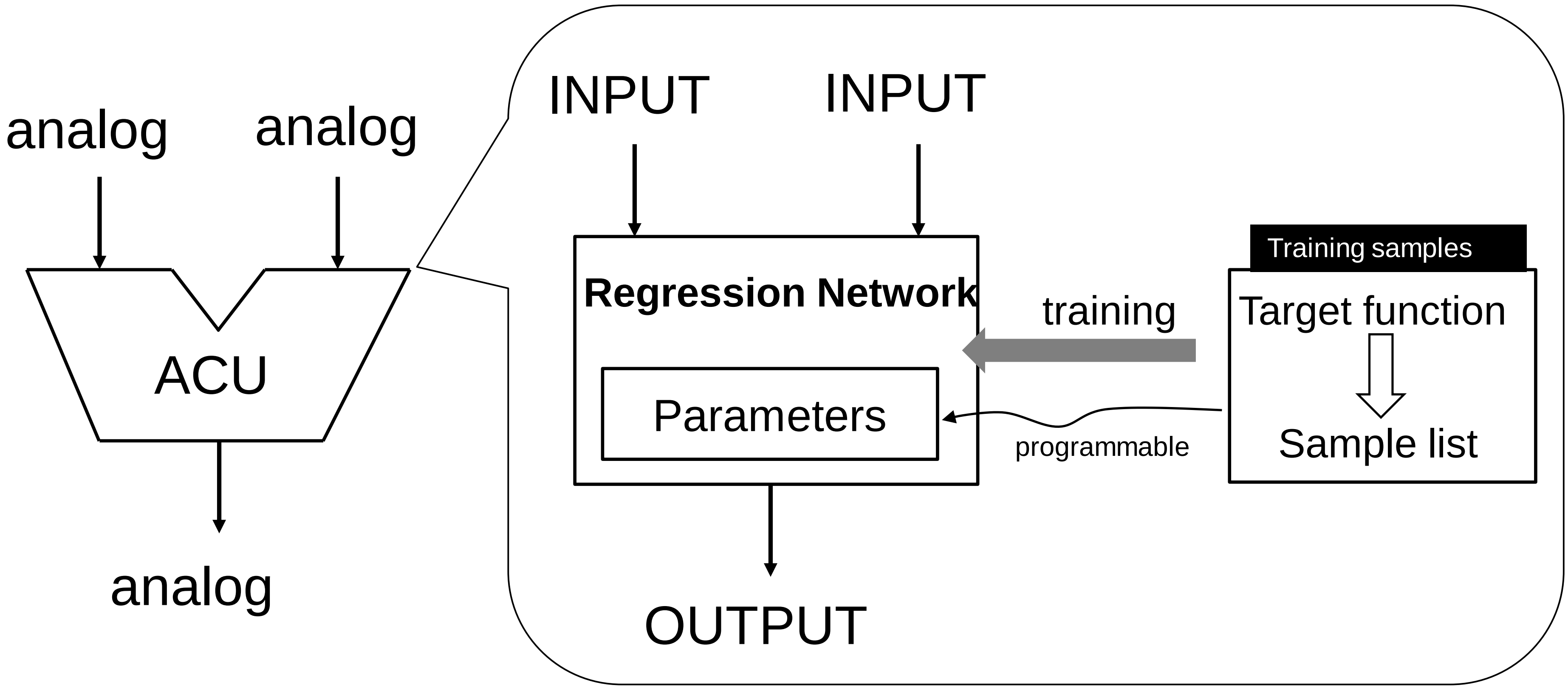
Purpose: reduce SVs to a small and constant number → friendly to HW implementation




Retrieve 2-operand calculations by 20 SVs

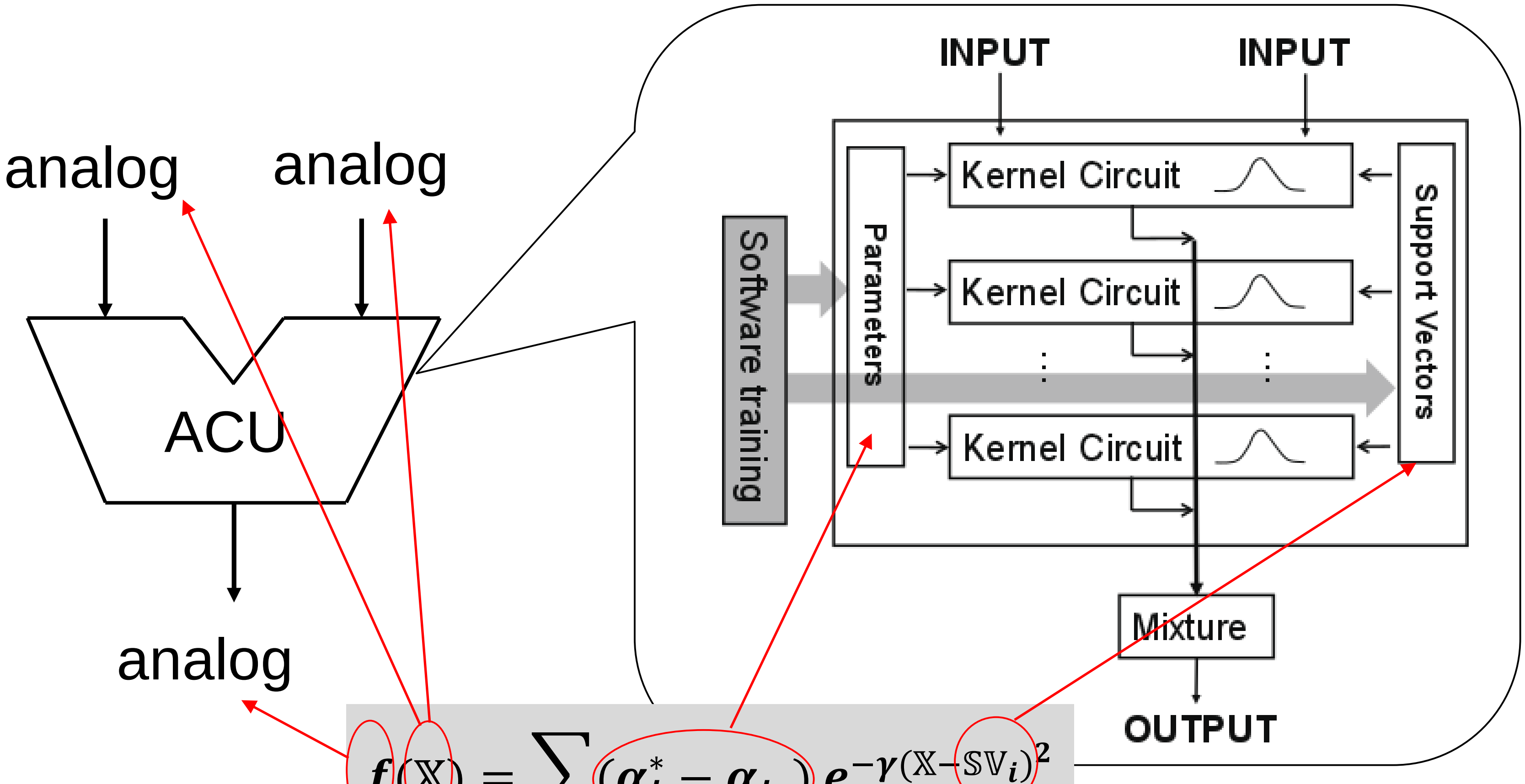
# **ML driving:** Programmable Analog Calculation Unit (ACU)

Analog Calculation Unit (ACU): Calculate analog functions in real-time by regression



# **ML driving:** Programmable Analog Calculation Unit (ACU)

Analog Calculation Unit (ACU): Calculate analog functions in real-time by regression

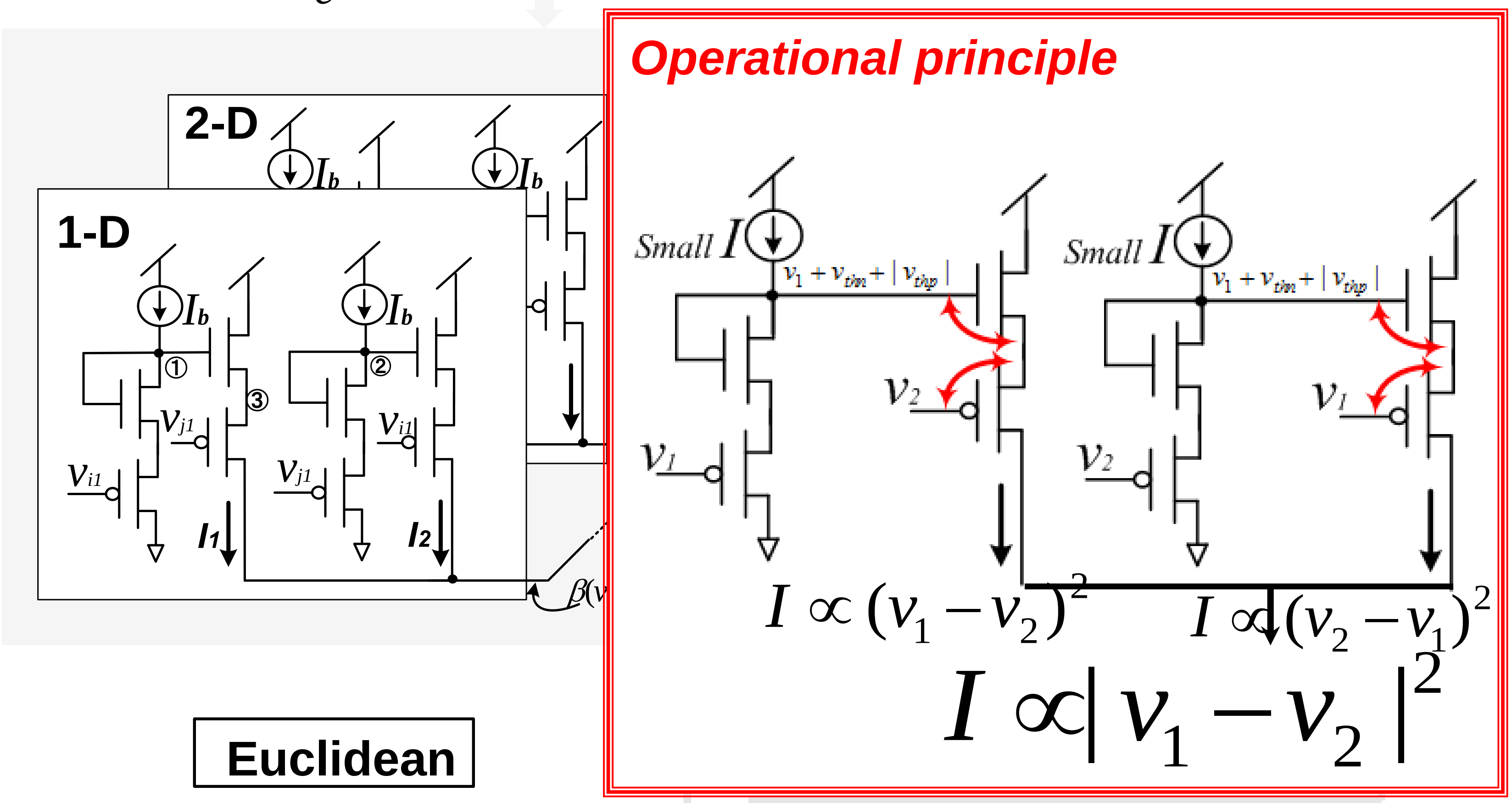



$$f(\mathbb{X}) = \sum (\alpha_i^* - \alpha_i)\, e^{-\gamma(\mathbb{X} - \mathbb{SV}_i)^2}$$

# **ML driving:** Programmable Analog Calculation Unit (ACU)

How to realize the “Gaussian Kernel” in silicon?

$$I_c \cdot e^{-\frac{(v_{i1}-v_{j1})^2+(v_{i2}-v_{j2})^2}{\sigma}}$$


$$I \propto (v_1 - v_2)^2 \qquad I \propto (v_2 - v_1)^2$$

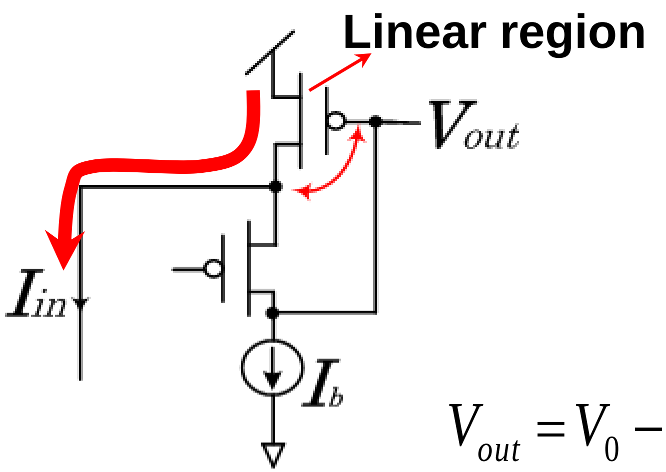
$$I \propto |v_1 - v_2|^2$$

# **ML driving:** Programmable Analog Calculation Unit (ACU)

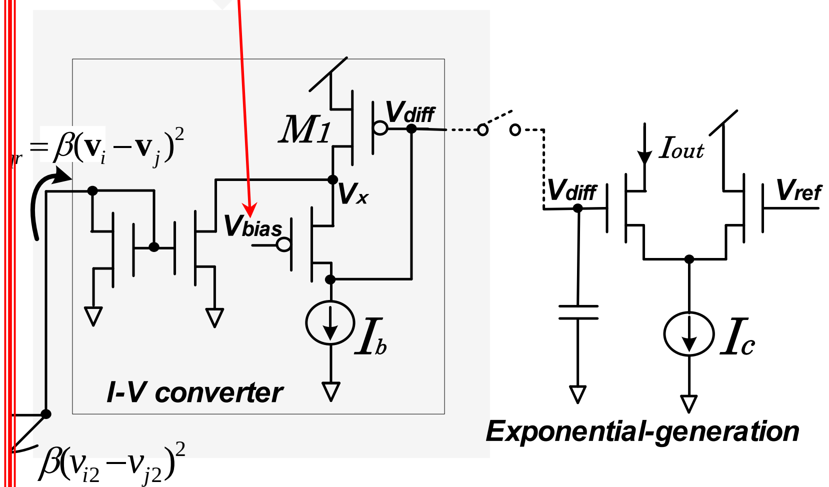
How to realize the “Gaussian Kernel” in silicon?

$$I_{out} = e^{-\frac{(v_{i1}-v_{j1})^2+(v_{i2}-v_{j2})^2}{\sigma}}$$


## *Operational principle*




$$V_{out} = V_0 - \frac{I_{in}}{\sigma}$$




**Euclidean**

**Exp.**

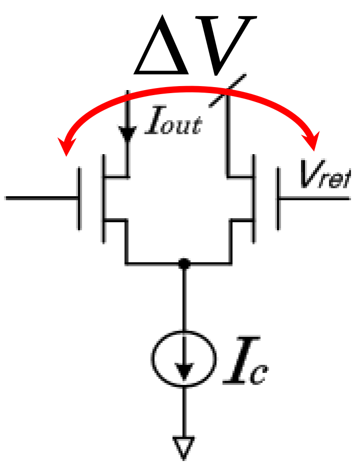
# **ML driving:** Programmable Analog Calculation Unit (ACU)

How to realize the “Gaussian Kernel” in silicon?

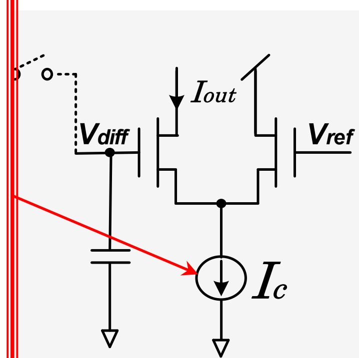
$$I_c \cdot e^{-\frac{(v_{i1}-v_{j1})^2+(v_{i2}-v_{j2})^2}{\sigma}}$$


***Operational principle***

$$I_{out} \approx \frac{I_c}{2} \cdot e^{\frac{q}{kT}\Delta V}$$




**Unbalance of differential pair**

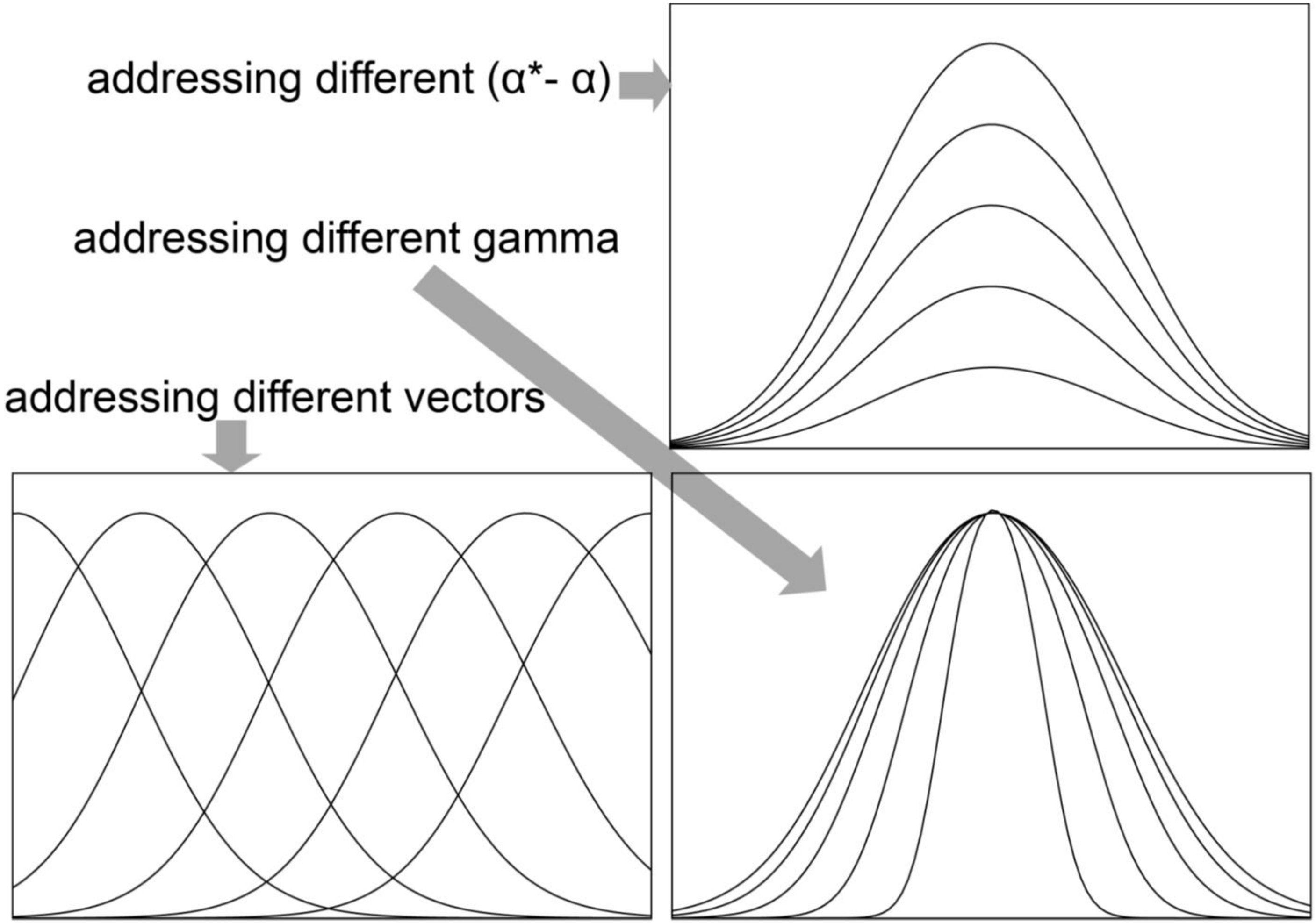



**Euclidean**

**Exp.**

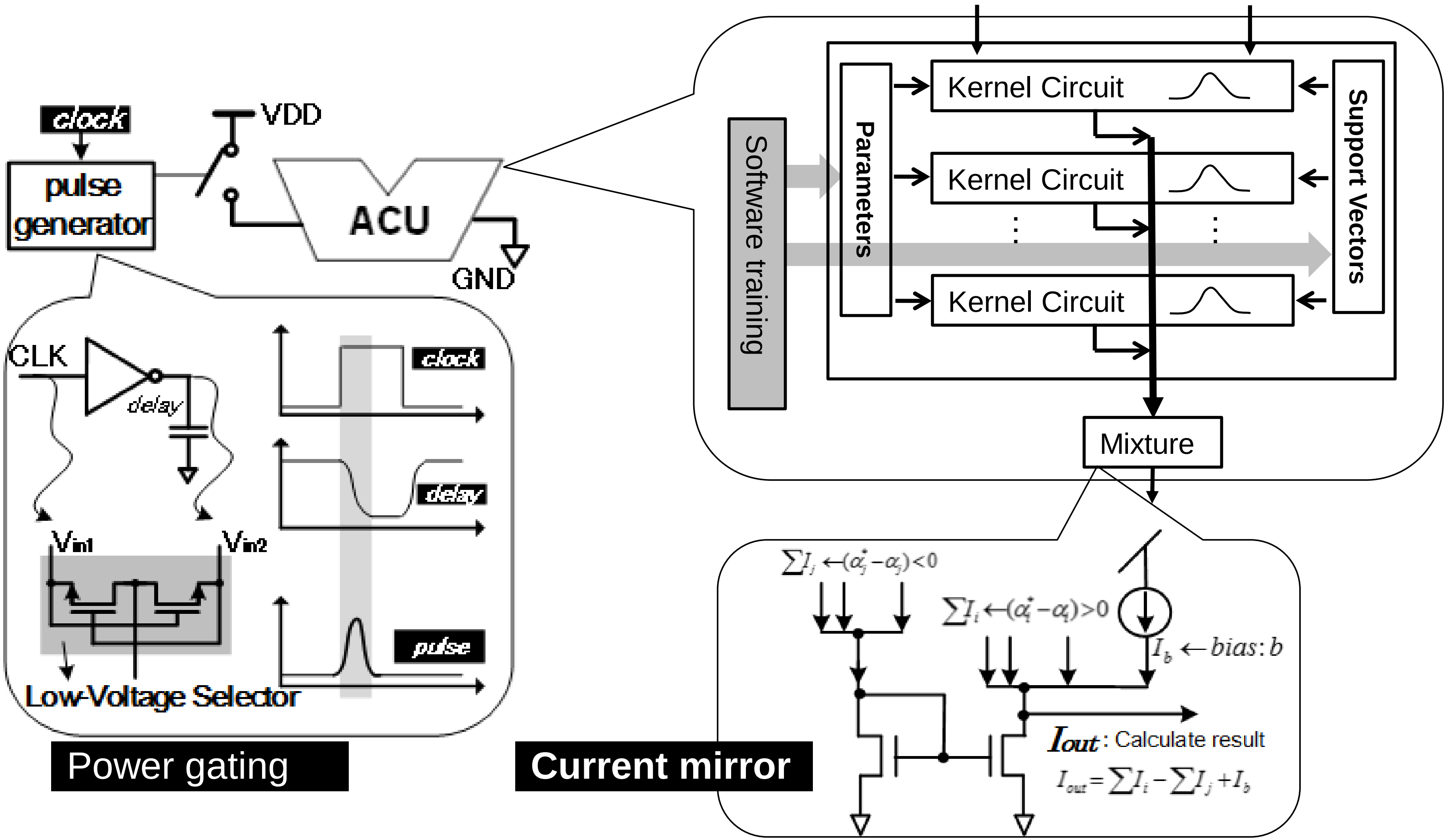
# **ML driving:** Programmable Analog Calculation Unit (ACU)

HSPICE simulation results of Gaussian kernel circuit



# **ML driving:** Programmable Analog Calculation Unit (ACU)

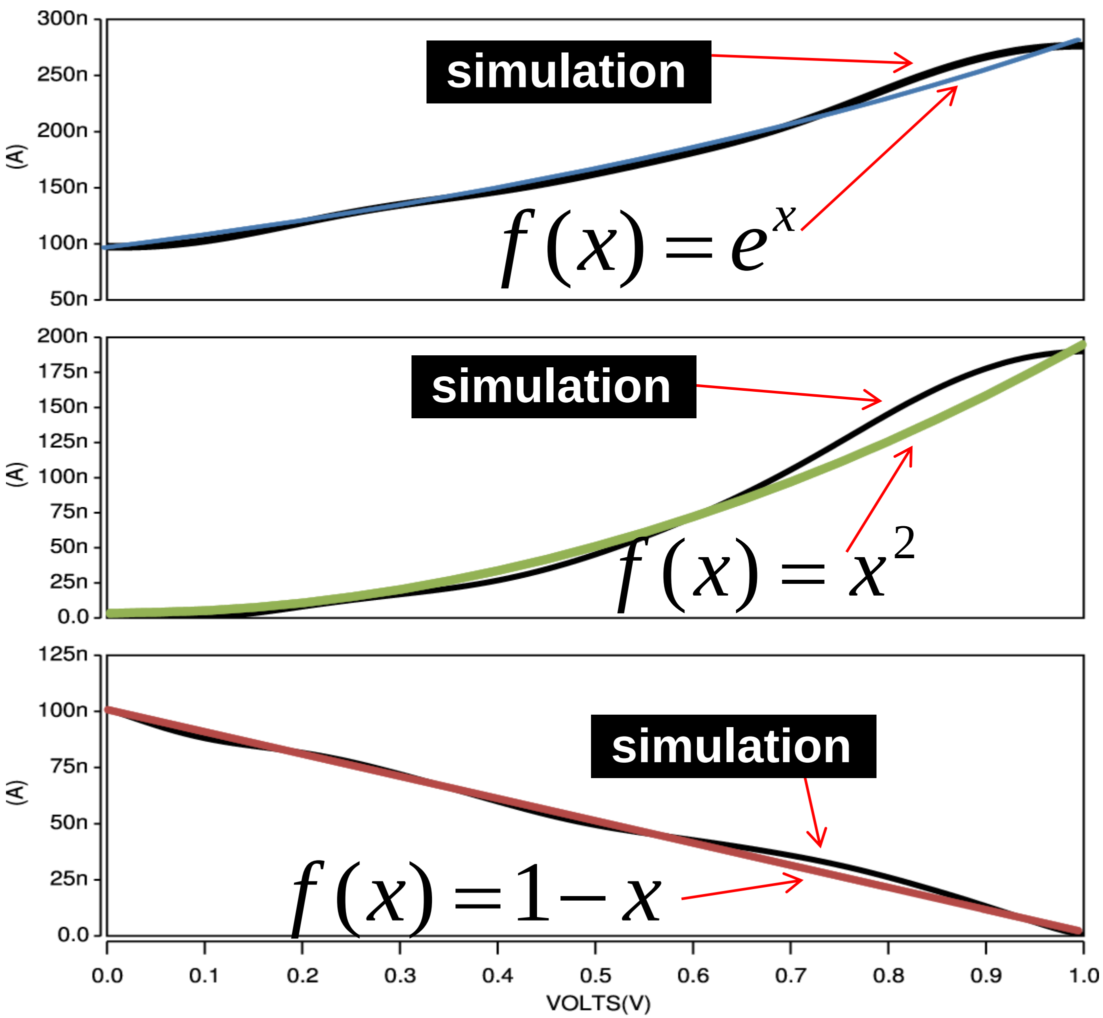
Several minor circuits to implement entire ACU

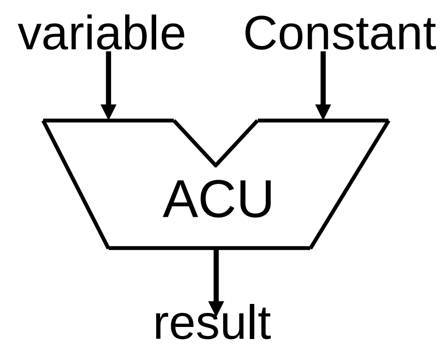


## **ML driving:** Programmable Analog Calculation Unit (ACU)

# Circuit simulation results of ACU

1-D examples (single operand calculation)




Average error

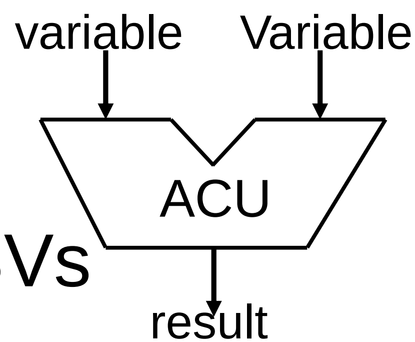
| | Theoretical regression | circuits |
|---|---|---|
| $f(x) = e^x$ | 0.03% | 2% |
| $f(x) = x^2$ | 0.06% | 3.5% |
| $f(x) = 1 - x$ | 0.0002% | 0.9% |

10 SVs

# **ML driving:** Programmable Analog Calculation Unit (ACU)

## Circuit simulation results of ACU

2-D examples (2-operand calculation) $f(x_1, x_2) = \sqrt{x_1{}^2 - x_2{}^2}$




20 SVs

Average error

| | Original SVR | Our |
|---|---|---|
| # of SVs | 98 | 20 |
| Error theoretical | 0.03% | 0.5% |
| Error circuit | 4.1% | 4.3% |

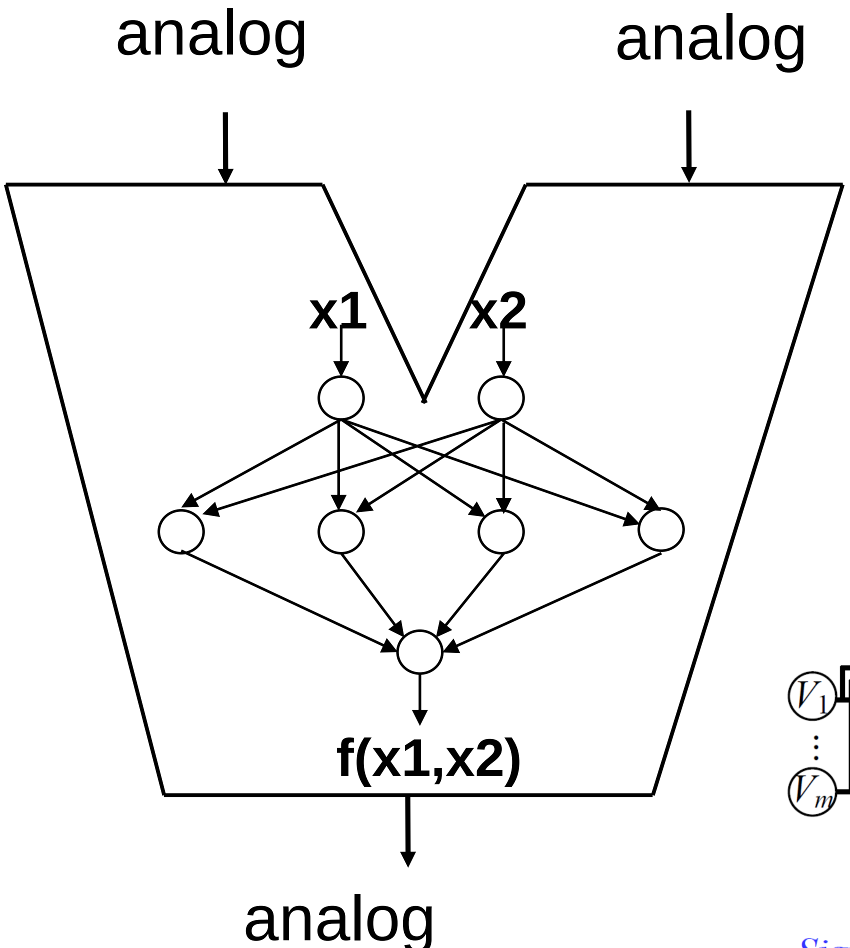
Comparisons on hardware resource

| | 4-bit ALU | 4-bit FPGA | MVL-FPGA | NN-trans. | This work |
|---|---|---|---|---|---|
| Radix | Binary | Binary | Hex | Binary | Analog |
| Bits/Error | 4 | 4 | 1-hex(=4-bit) | Error~44.8% | Error~7% |
| Function | Simple | Arbitrary | Arbitrary | Arbitrary | Arbitrary |
| Operands | 2 | 2 | 2 | 1 | **9** |
| # of Tr.s | >10000 | 12288 | 5808 | >3000+CPU | **5000** |
| Speed | Multi-cycle | Real-time | Real-time | Multi-cycle | Real-time |

# **Simpler algorithms?** NN-regression in silicon

**Idea:**

Similar to previous ACU, implement regression algorithms by HW. This time, neural-network in analog.

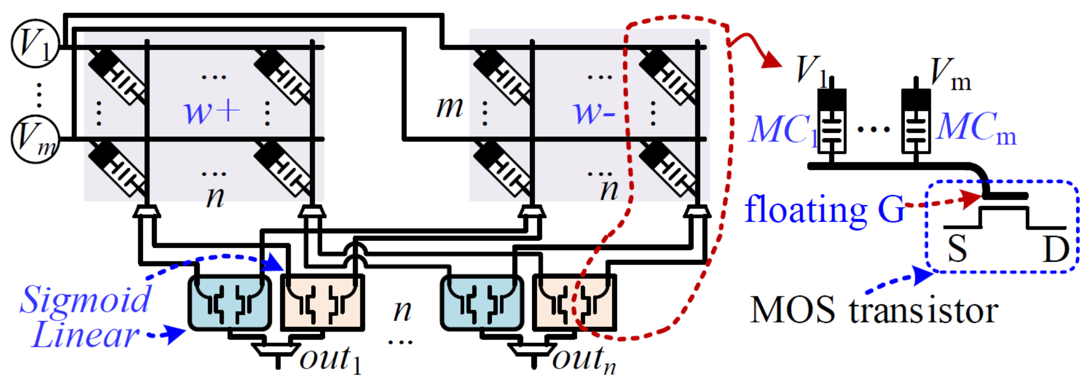



1. Circuit should be simple;
2. ``weight” should be programmable;
3. No OPAMPs

➔ MemCapacitor + Neuron MOS

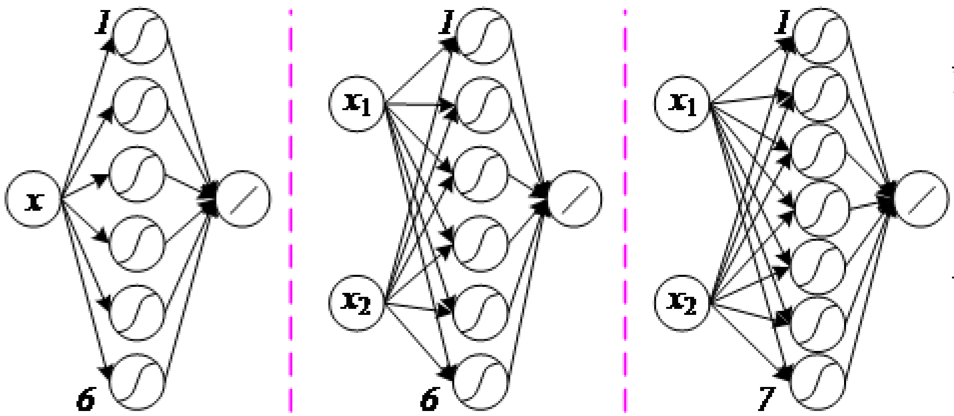
➔ 1 Synapse = 1 Cap.; 1 Neuron = 1 MOS

$$\emptyset_F = \frac{C_1V_1 + C_2V_2 + \cdots + C_nV_n}{C_{total}}$$



# **Simpler algorithms?** NN-regression in silicon

## Current progress




$f(x) = \sin(x)$ | $f(x_1, x_2) = x_1 x_2$ | $f(x_1, x_2) = x_1 \wedge x_2$

Error = 2% | Error = 4% | Error = 3%

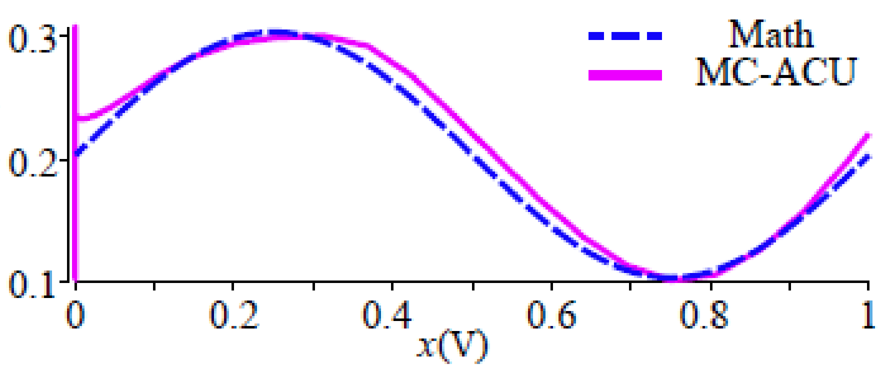
[*] S. Eldridge, et al. "Neural network-based accelerators for transcendental function approximation", GLSVLSI, 2014.

| Energy(pJ) | *floating* in 45nm* | *fixed* in 45nm* | MC-ACU in 180nm |
|---|---|---|---|
| *sin* | 311 | 8.3 | 6.16 |
| *exp* | 453 | 24.81 | 8.03 |
| *pow* | 1292 | 101.67 | 4.17 |
| *norm* | - | - | 50.5 |
| Gmean | 566.7 | 27.6 | 10.1 |

Baseline (fixed point)
Error ~ 1% for
Operand only one

| calc. | *floating* in 45nm * | *fixed* in 45nm * | MC-ACU in 180nm | | |
|---|---|---|---|---|---|
| device | # of Tr.s | # of Tr.s | # of Tr.s | #memcapacitor | total |
| *sin* | >50,000 | >1100 | 276 | 28 | 304 |
| *exp* | >80,000 | >3000 | 276 | 28 | 304 |
| *pow* | >200,000 | >3000 | 276 | 28 | 304 |
| *norm* | - | - | 417 | 44 | 461 |

Example: retrieve sin(x)




Conclusion:
MemCap. + Neuron-MOS = efficient

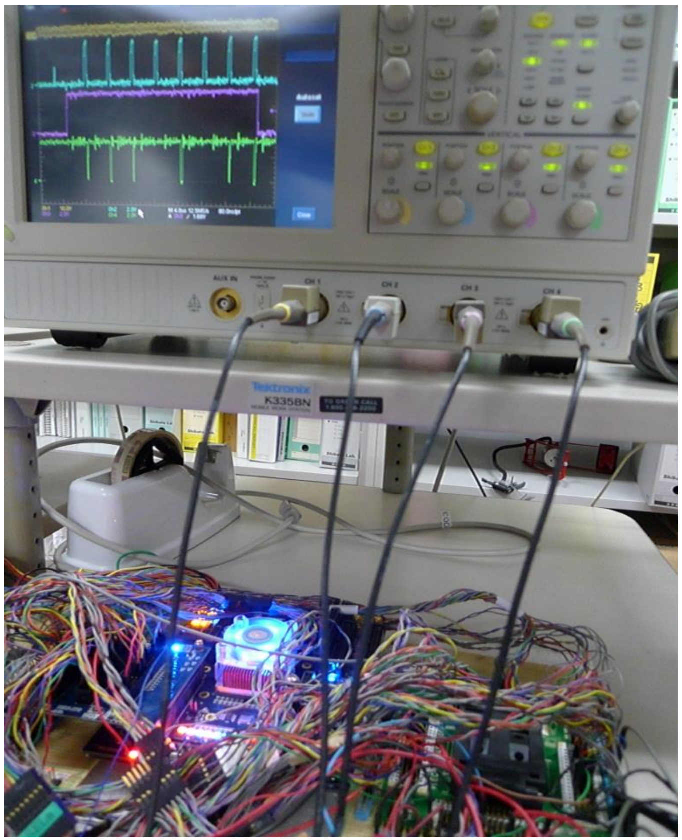
Prob.:
How to re-set MemCap. in physics?

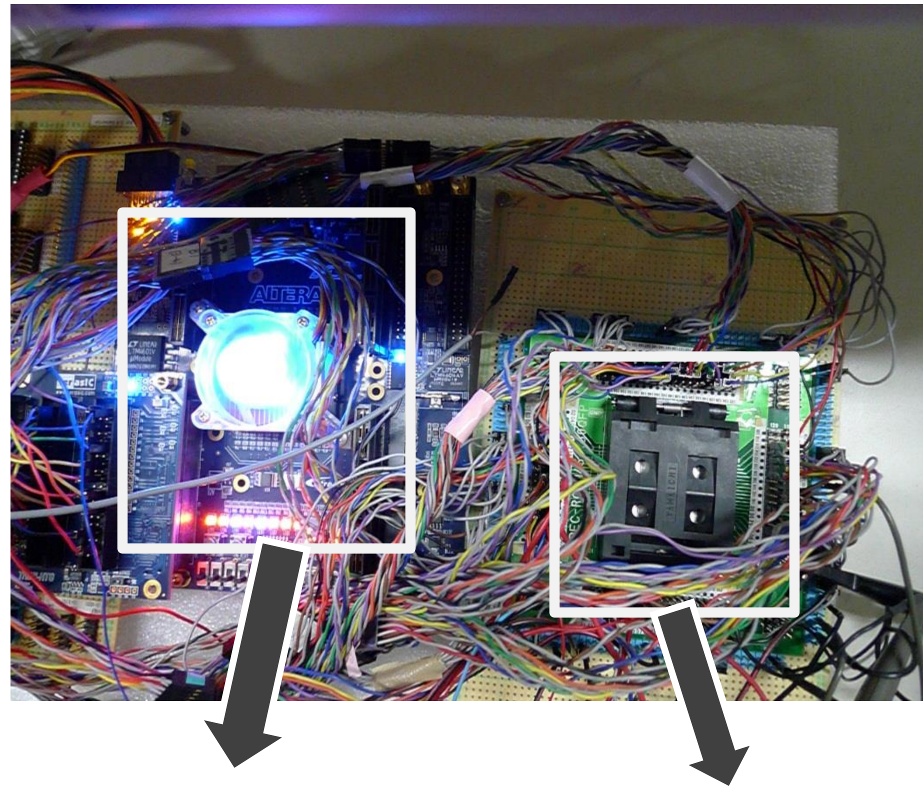
# Comparison of three analog computing

| | Merit | Demerit |
|---|---|---|
| OPAMP based<br>(textbook-like) | Easily understandable;<br>Rich IP; | Static power;<br>Not programmable;<br>R/C/L hungry;<br>Noise/Varitions |
| Physics Computing<br>(Oscillation etc., same story for Quantum) | Interconnection free;<br>Potential of super small/lower_power/fast…; | Func. Limited (specified?);<br>Expertise hungry;<br>Expensive;<br>Noise/Variations |
| Programmable ACU<br>(powered by ML?) | Programmable;<br>Expertise free; | Noise/Variations;<br>Static power; |

# Application example

SVM in tracking





**FPGA: Tracking Framework**

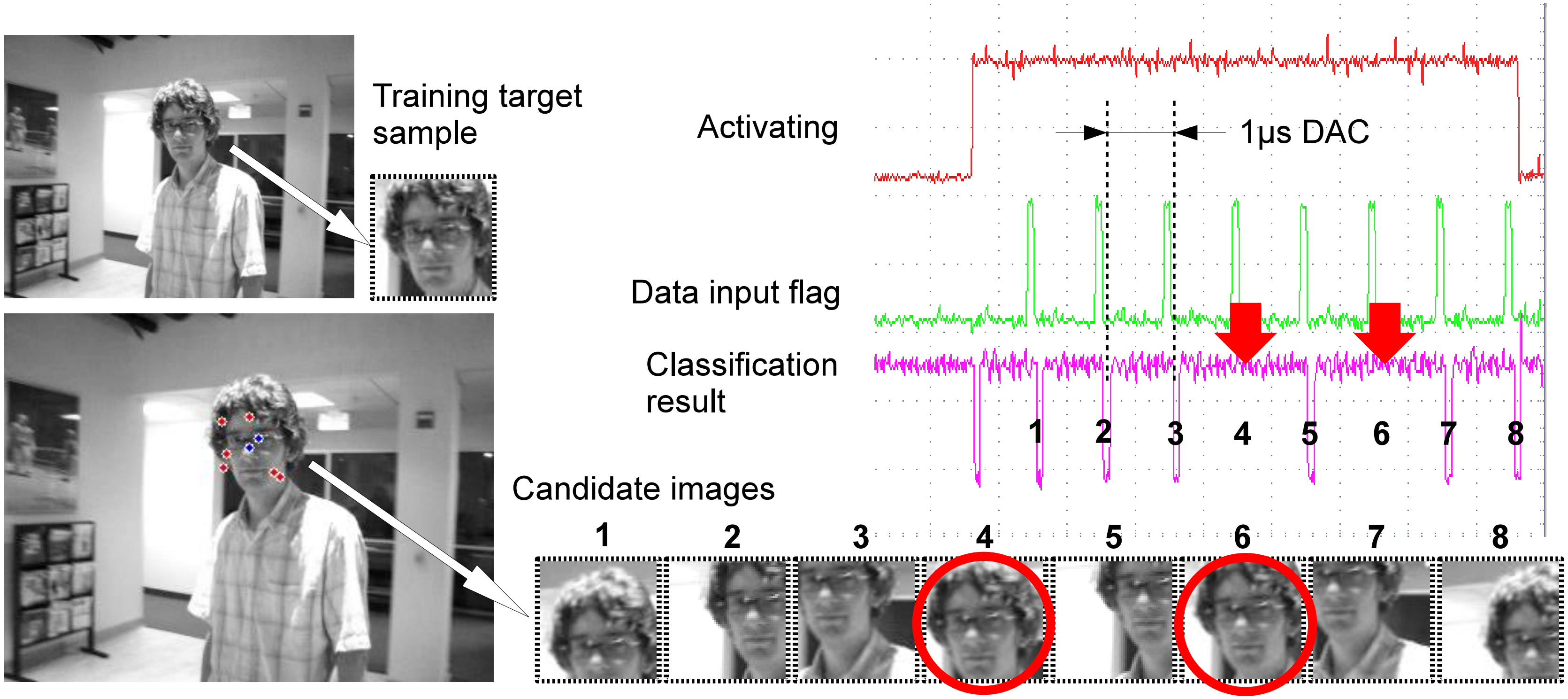
**Analog SVM chip**

*Analog SVM chip was employed*

# Application example

SVM in tracking

*Measurement*



# End


Thank you very much.