Approximate Computing Techniques: From Component- to Application-Level

Alberto Bosio

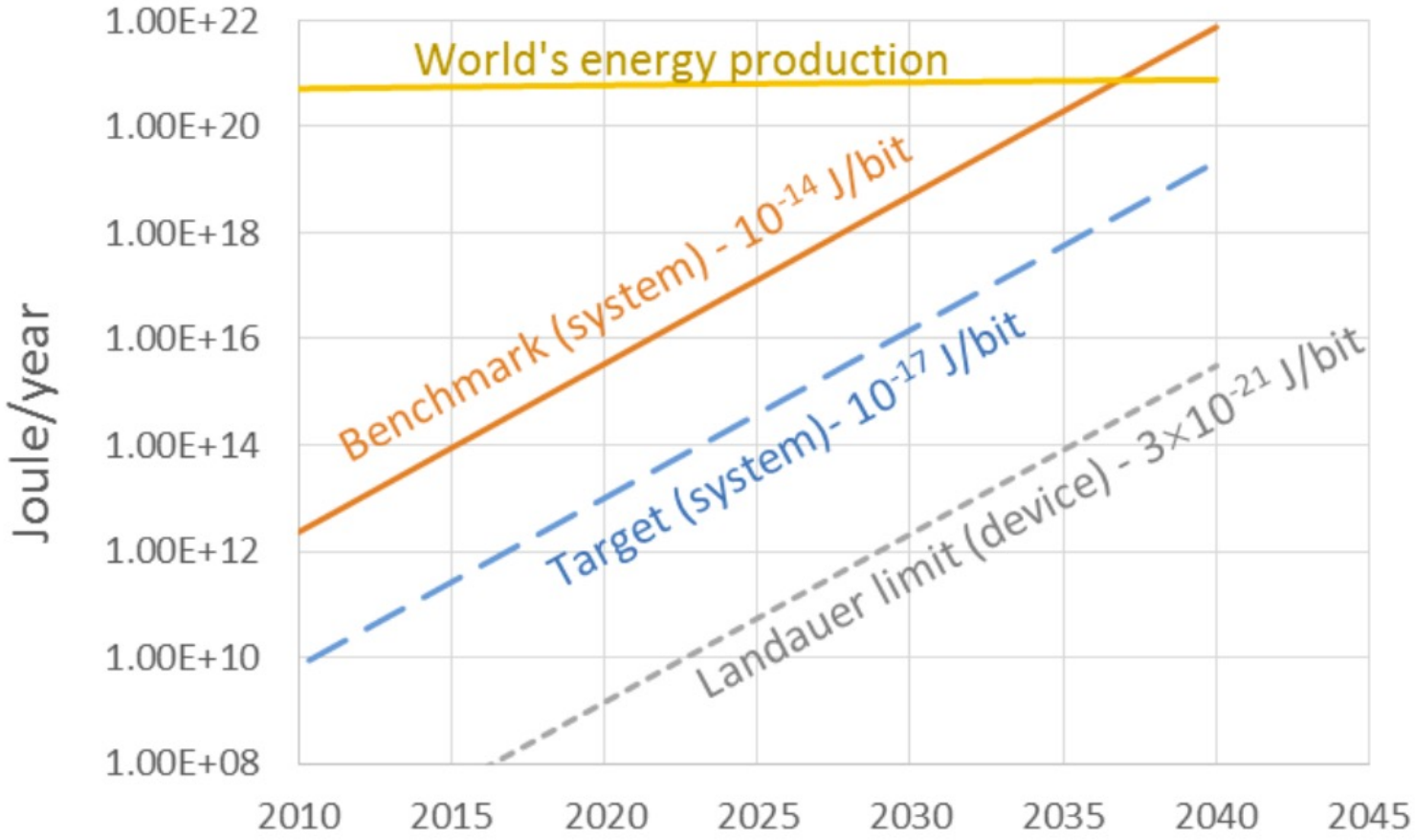
alberto.bosio@ec-lyon.fr

Institut des Nanotechnologies de Lyon
Ecole Centrale de Lyon
France

Agenda

- Introduction
- Approximate Computing
- Techniques and DSE
- Approximate Computing For Safety-Critical Systems
- Conclusions

Context



Source: SIA/SRC
https://www.src.org/newsroom/rebooting-the-it-revolution.pdf, 2015.

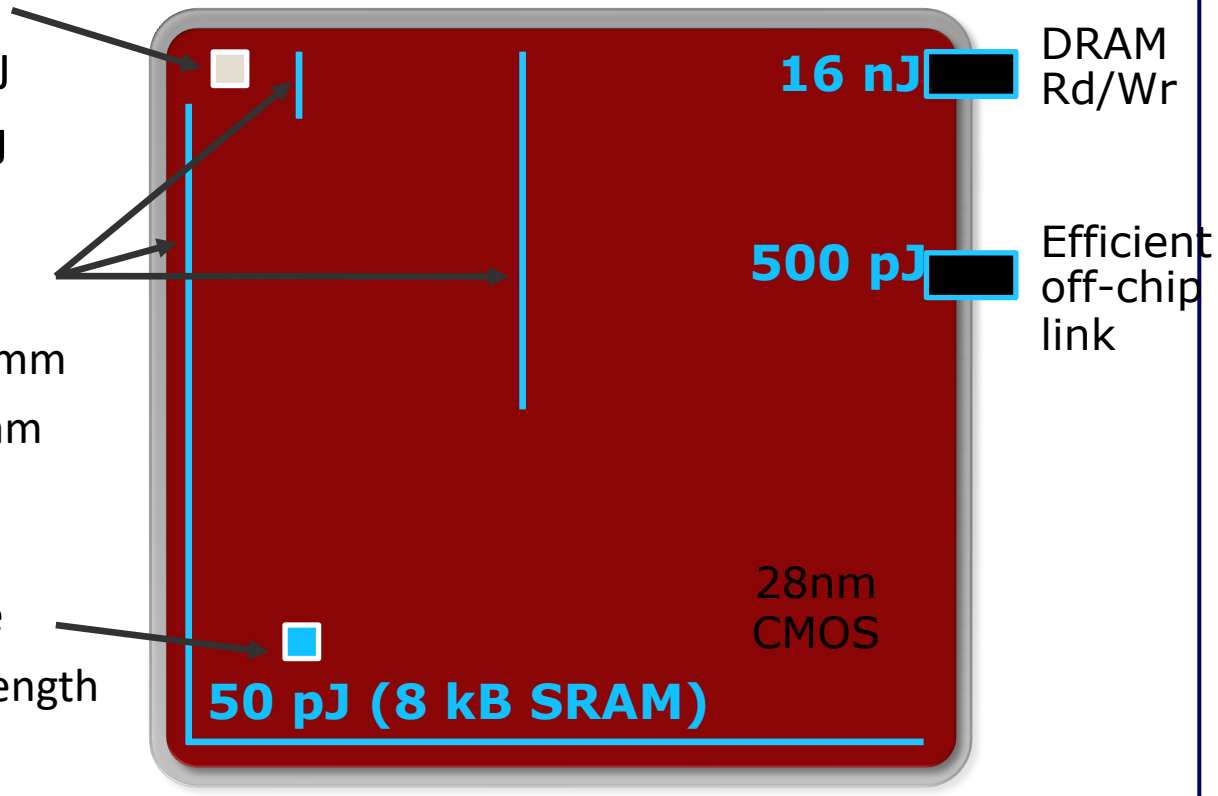
Energy Cost in a Processor/SoC

[Adapted from Dally, IPDPS'11]

- 64-bit FPU: 20pJ/op
- 32-bit addition: 0.05pJ
- 16-bit multiply: 0.25pJ

- Wire energy
  - 32 bits: 40pJ/word/mm
  - 8 bits: 10pJ/word/mm

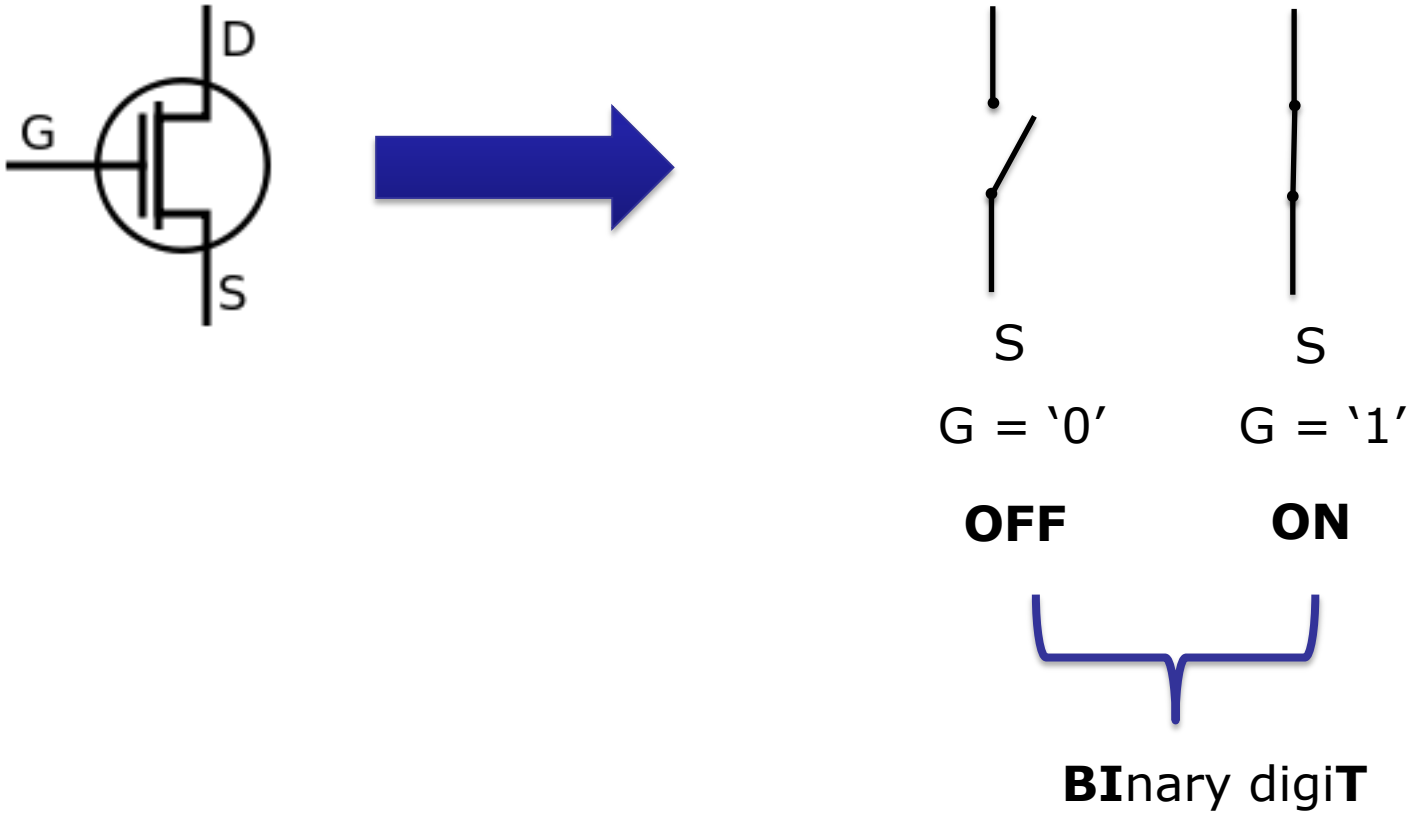
- Memory/Register-File
  - Depends on word-length



Courtesy of O. Sentieys

A closer look at the processor…

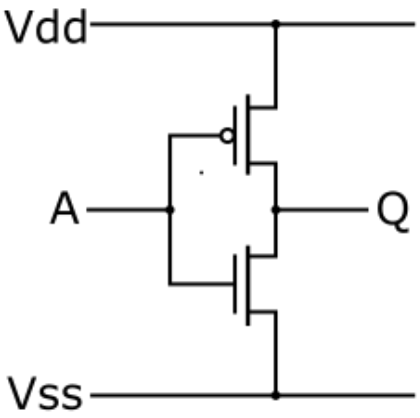
- MOSFET

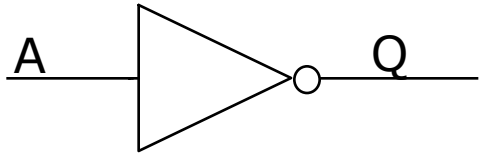


CMOS Technology

- Complementary MOS technology (CMOS)
  - Two p- and n-channel MOSFETs as building blocks

Inverter Gate





Dynamic power consumption

- Power consumption always measured at the power source (V_{DD})
- Energy to charge capacitor C_L (via PMOS):

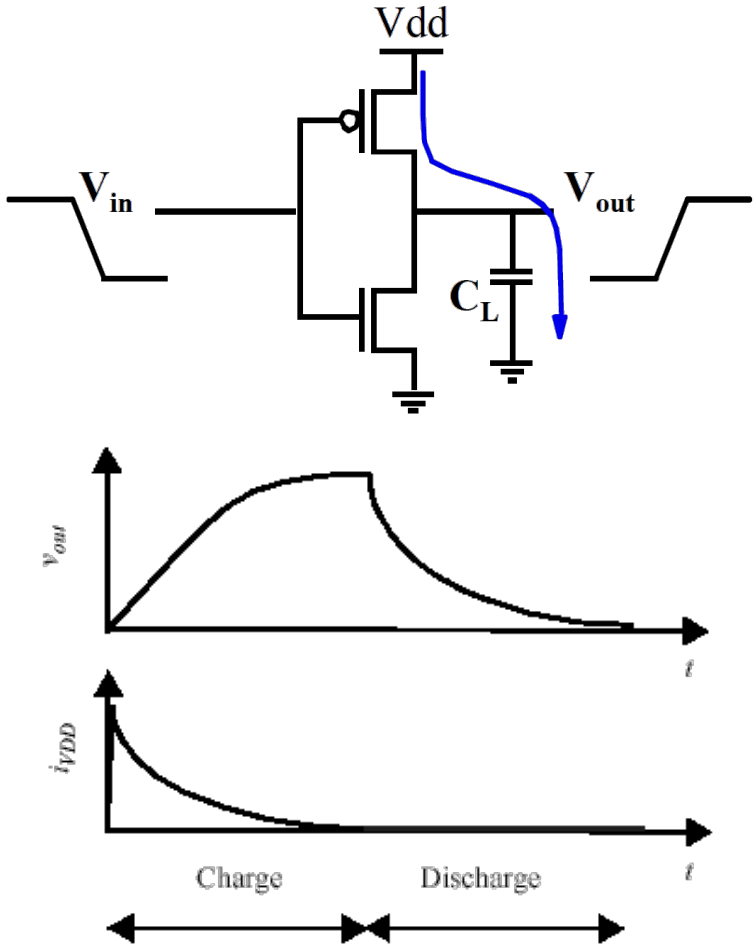
$$\int_0^{\infty} i_{VDD}(t) V_{DD} dt = V_{DD} \int_0^{\infty} C_L \frac{dv_{out}}{dt} dt$$

$$= C_L V_{DD} \int_0^{V_{DD}} dv_{out} = C_L V_{DD}^2$$

- Power consumption:

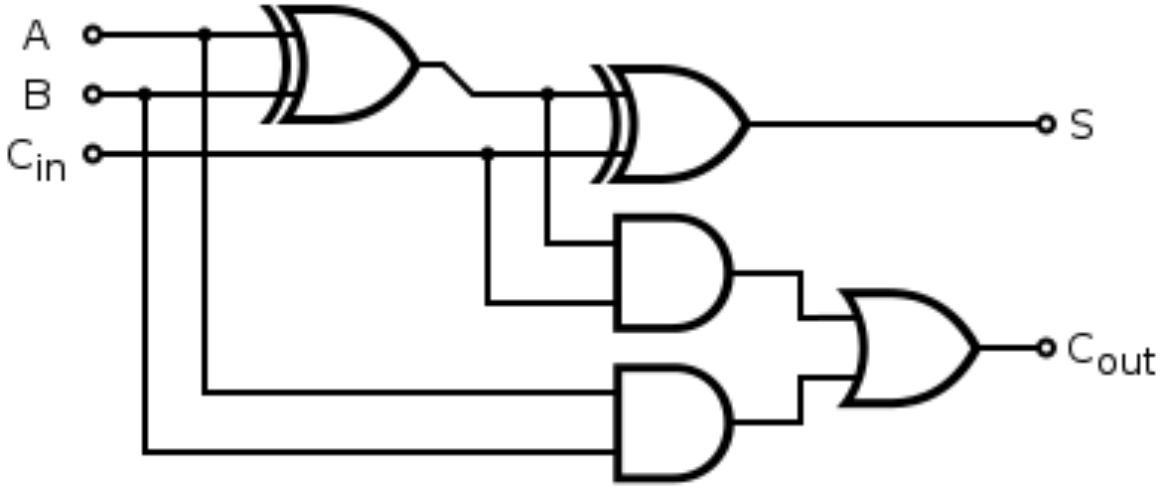
$$P_{dyn} = C_L V_{DD}^2 f_{0\to1}$$

  - Where $f_{0\to1}$ is the number of power-consuming transitions per second

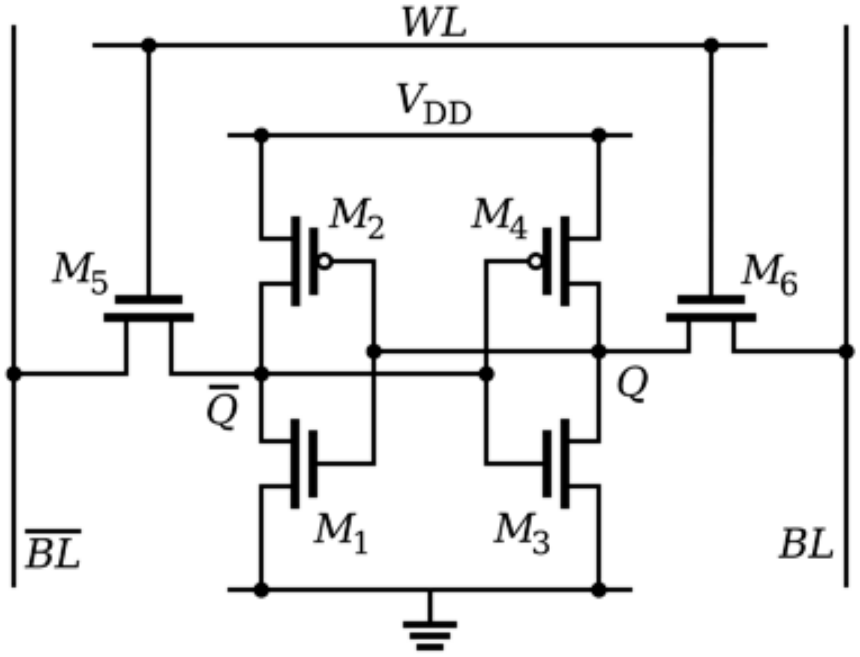


- One gate (0.25μm), 500MHz clock, load capacitance of 15fF, V_{DD}=2.5V and fanout of 4 : 50μW

More Complex Designs



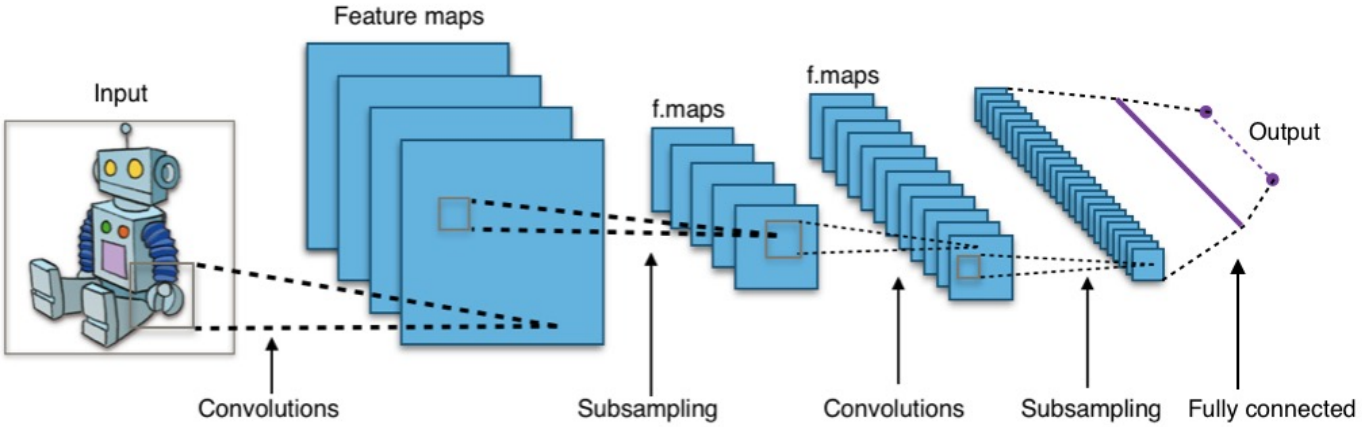
1-bit Adder



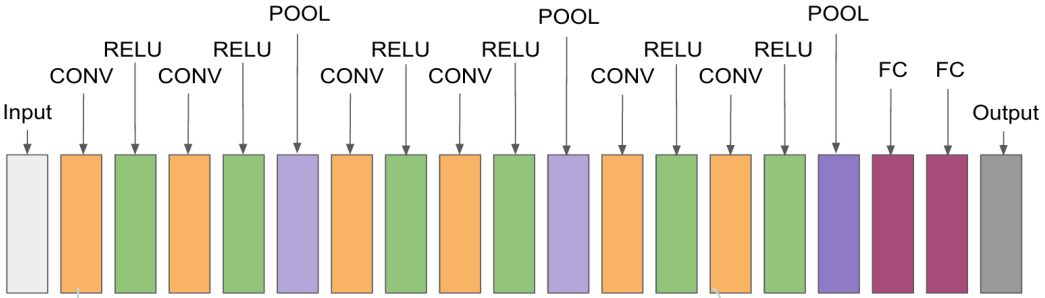
1-bit SRAM cell

Data Intensive Workload (example)

- Convolutional Neural Networks



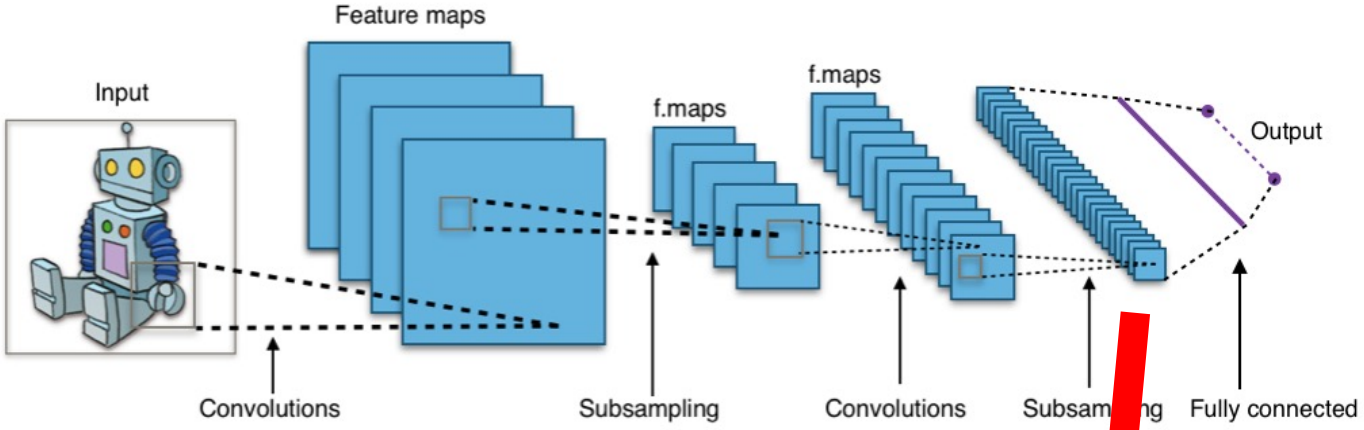
- Layers



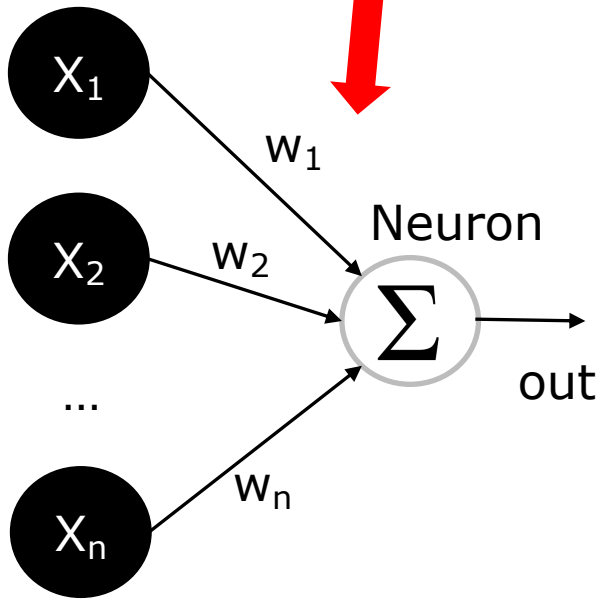

```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
  L:   output_fm[to][row][col] +=
          weights[to][ti][i][j]*
          input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

[Motamedi et al., 2016]

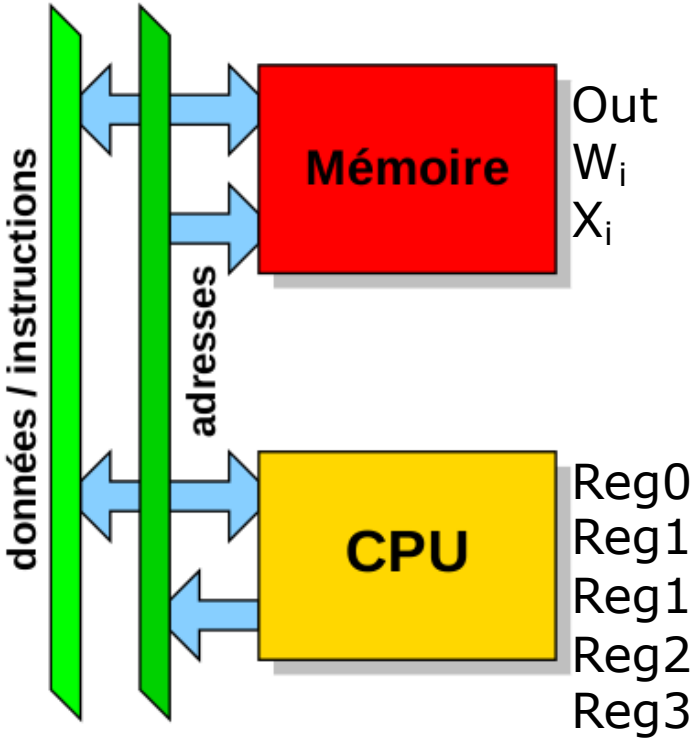
Why CNN are so complex?




```
for (I = 1 to N)
   out += Wi * Xi
```



Computer Architecture



Source: A. Tisserand

```
for (I = 1 to N)
```

```
out += Wi * Xi
```

assembly

```
Load Wi, reg0                    nJ
Load Xi,  reg1                   nJ
Load out, reg2                   nJ
Mul reg3, reg1, reg0             pJ
Add reg2, reg3, reg2             pJ
Store reg2, out                  nJ
```

6 memory accesses for instructions
4 memory accesses for data
2 operations

Let’s approximate…

Intrinsic Application Resiliance

- Ability to produce acceptable outputs despite underlying **computations** being affected by **errors/noise**

[Chi13]

Example: Resilience of ANN

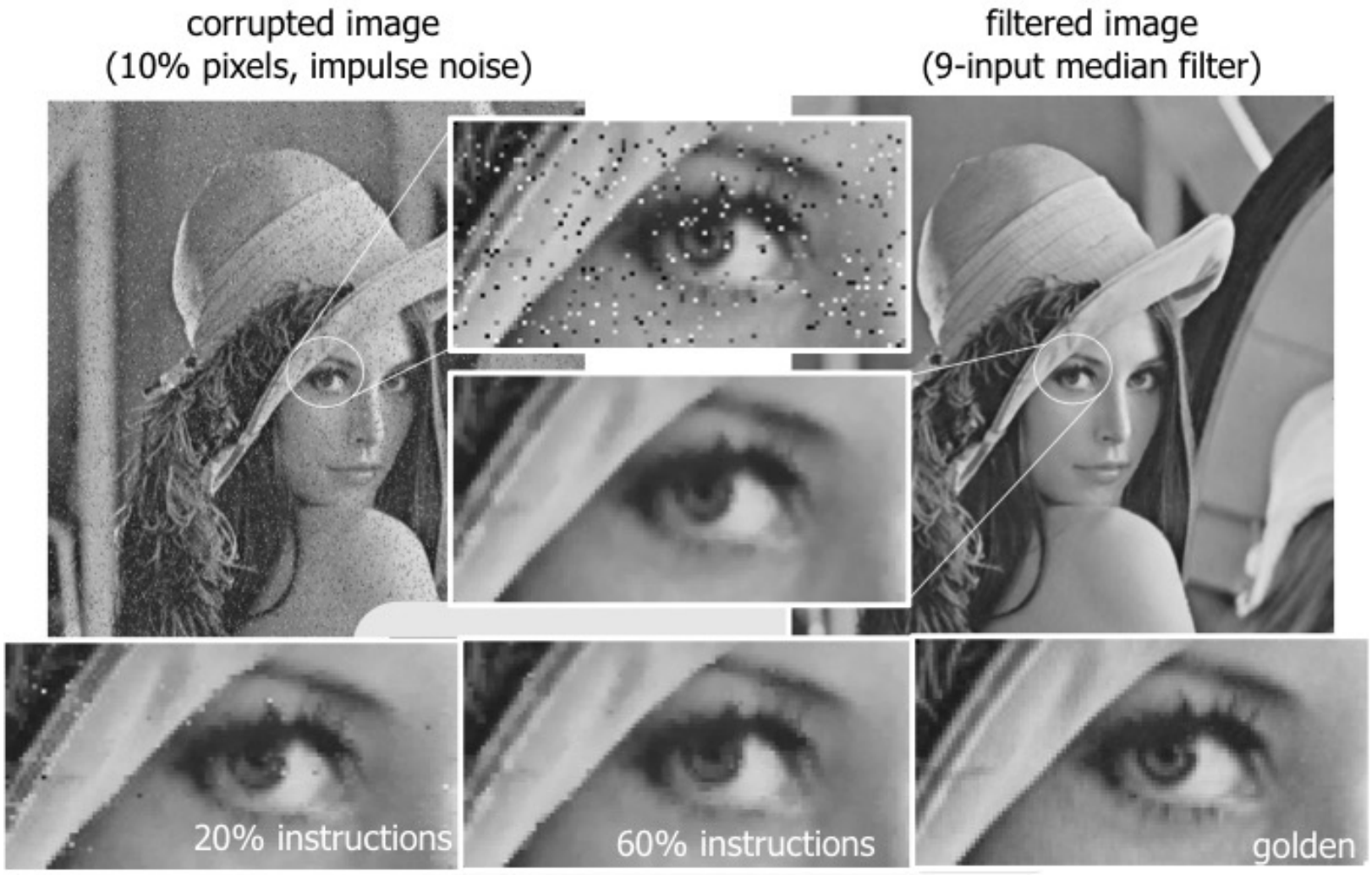
Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoatnt tihng is taht the frist and lsat ltteer be at the rghit pclae. And we spnet hlaf our lfie larennig how to splel wrods. Amzanig, no!

[O. Temam, ISCA10]

- Our biological neurons are tolerant to computing errors and noisy inputs
  - Quantization of parameters and computations provides benefits in throughput, energy, storage

Courtesy of O. Sentieys

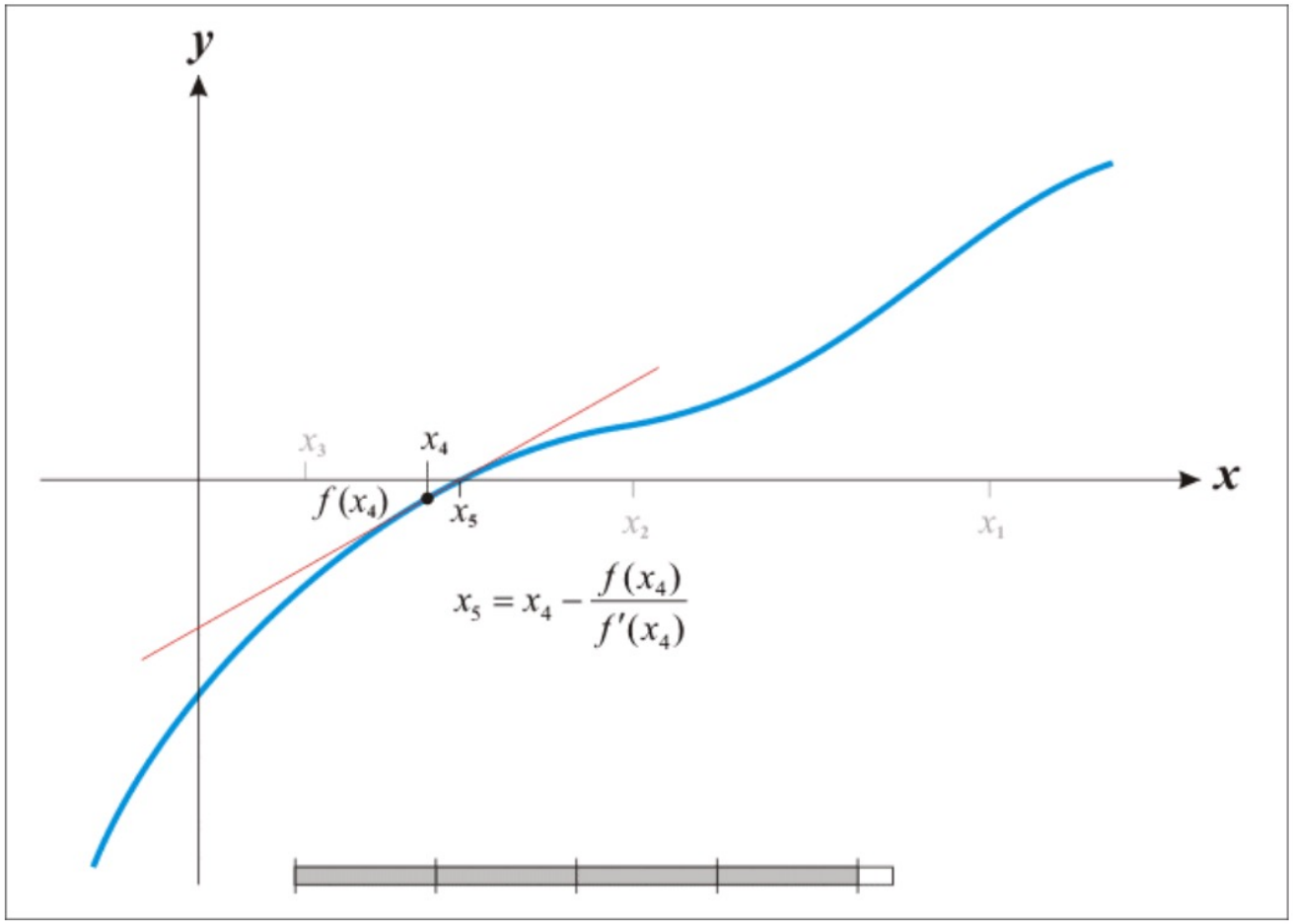
Example: Image Processing



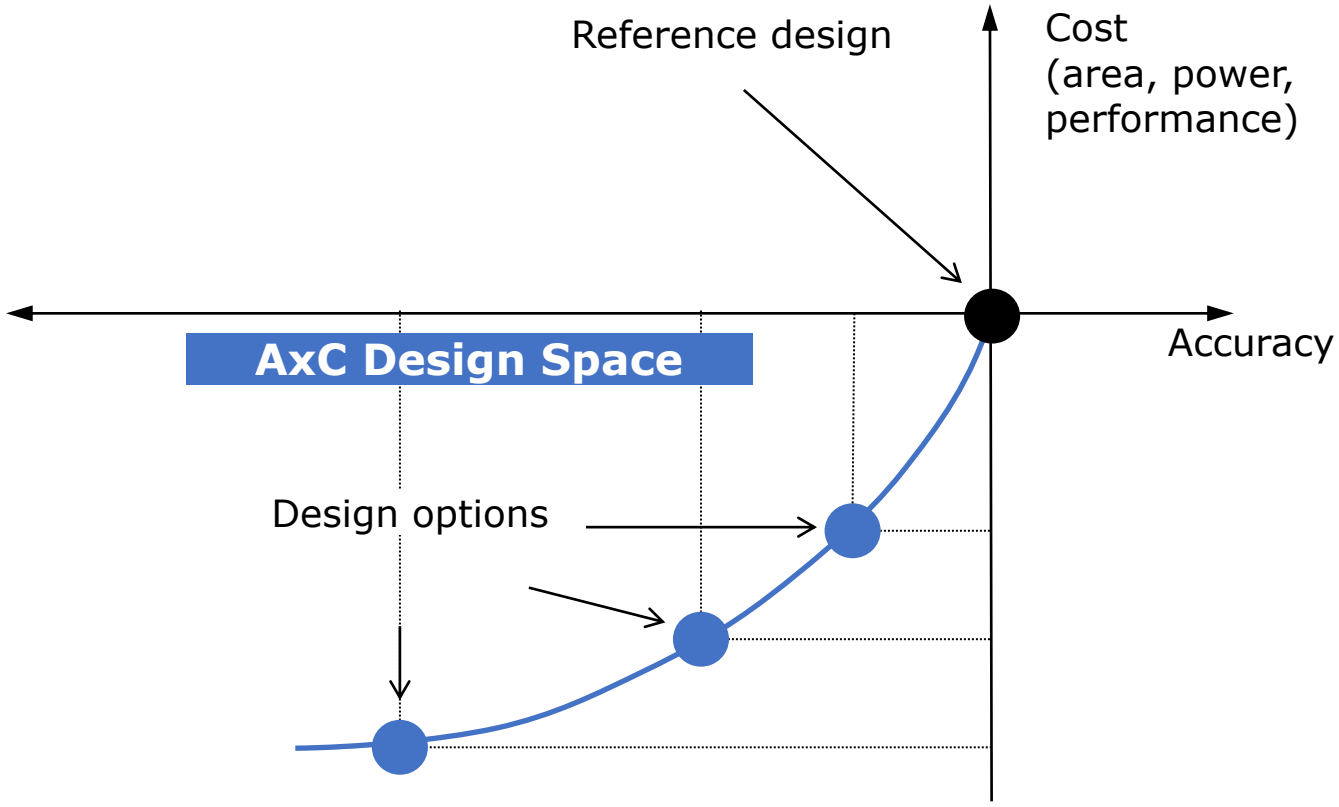
Courtesy of Lukas Sekanina

Example: Newton-Raphson Method

$i = 4$



Design Space Exploration



Approximate Computing

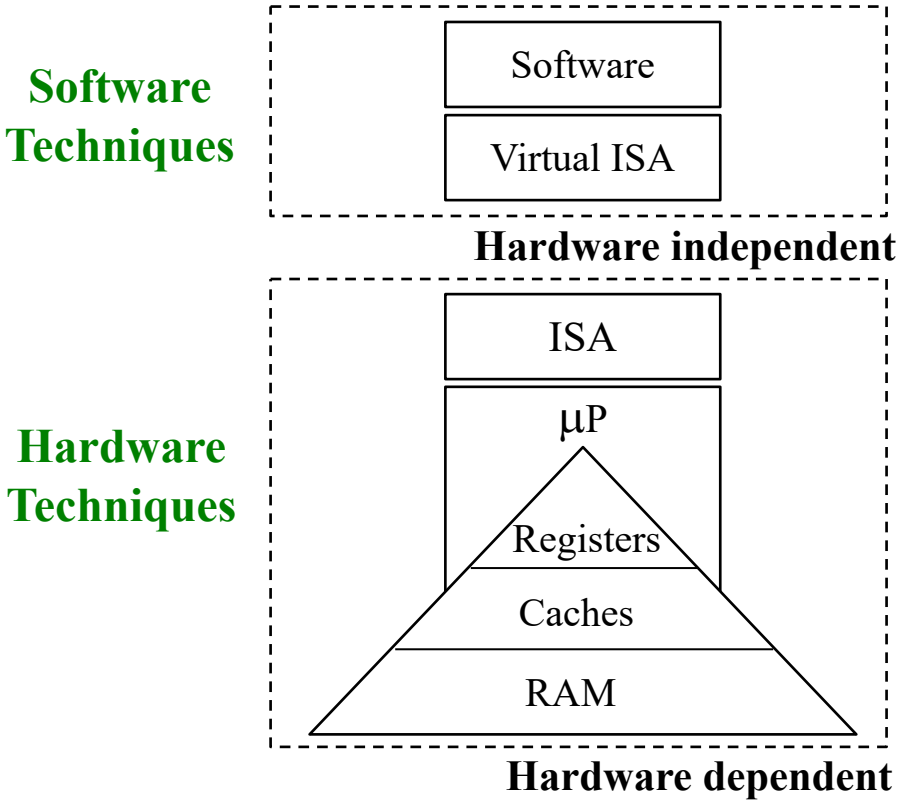
- How can we « approximate » a computing system?

AxC Techniques Taxonomy

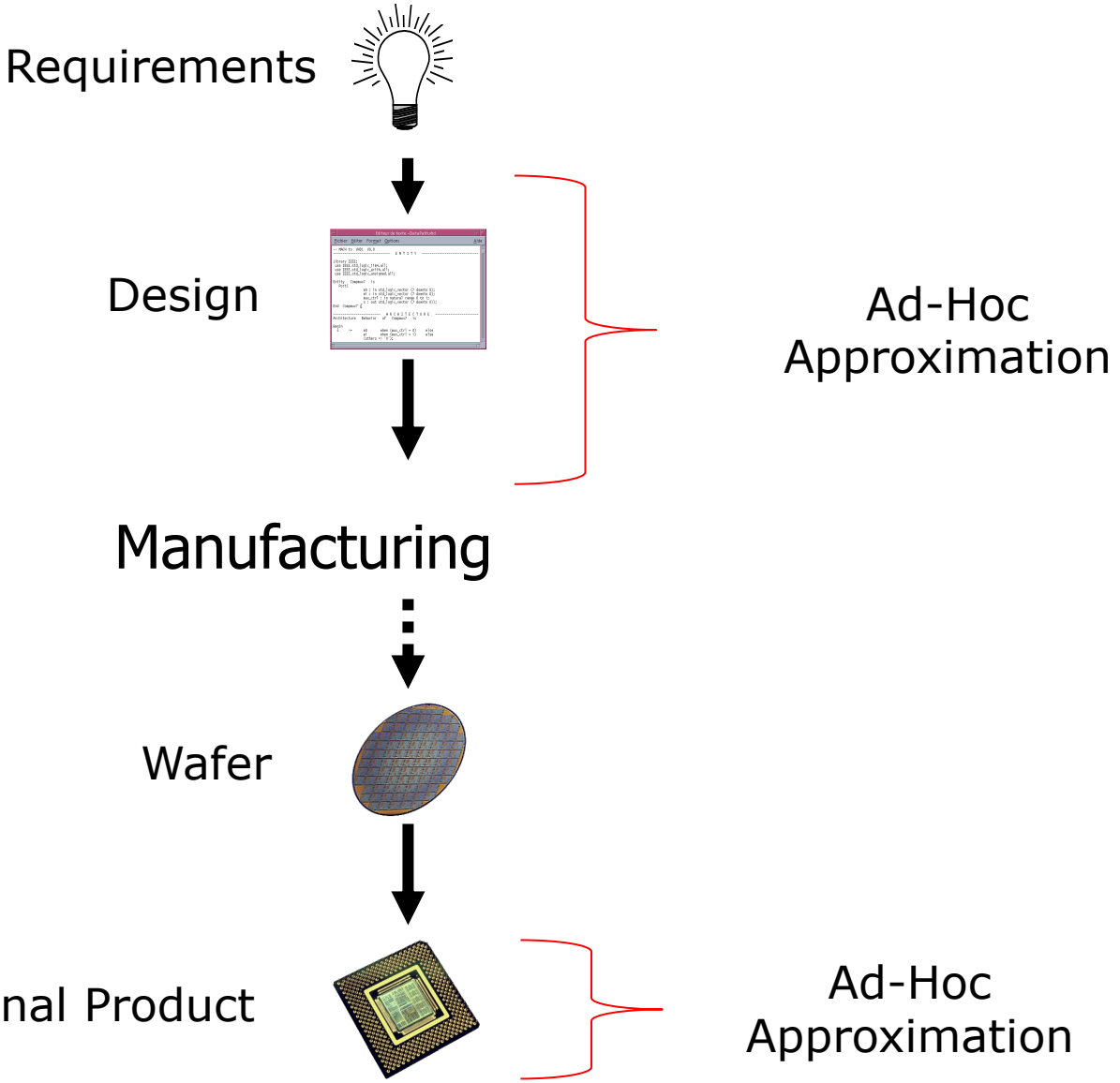
- Design time VS Run time (i.e., static VS dynamic)

AxC Techniques Taxonomy

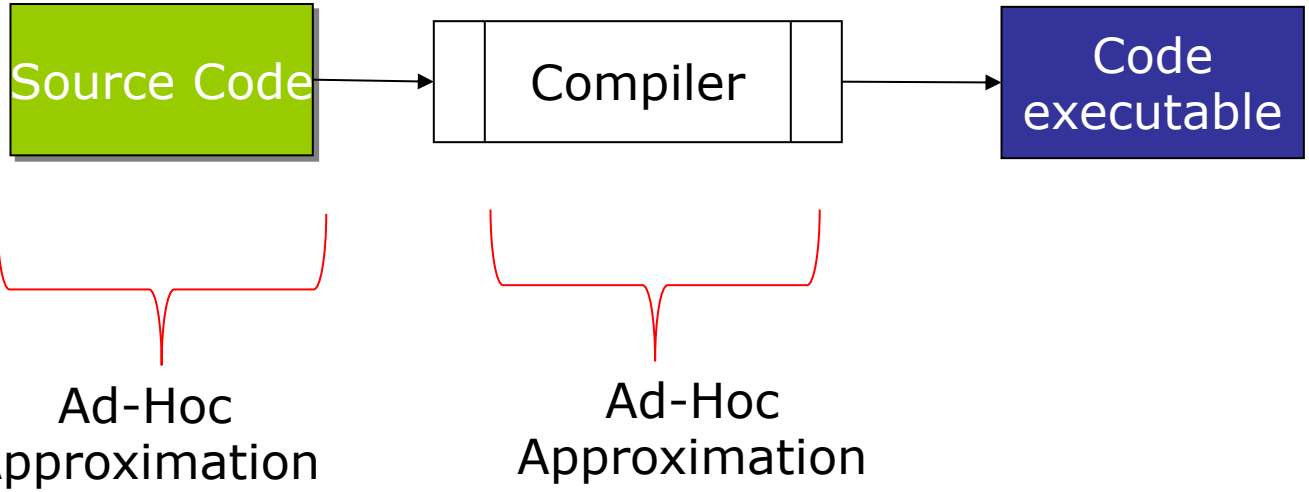
- Design time VS Run time (i.e., static VS dynamic)
- By Abstraction Layer



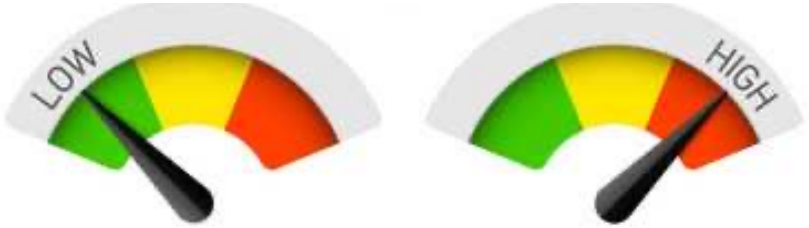
Static (HW)

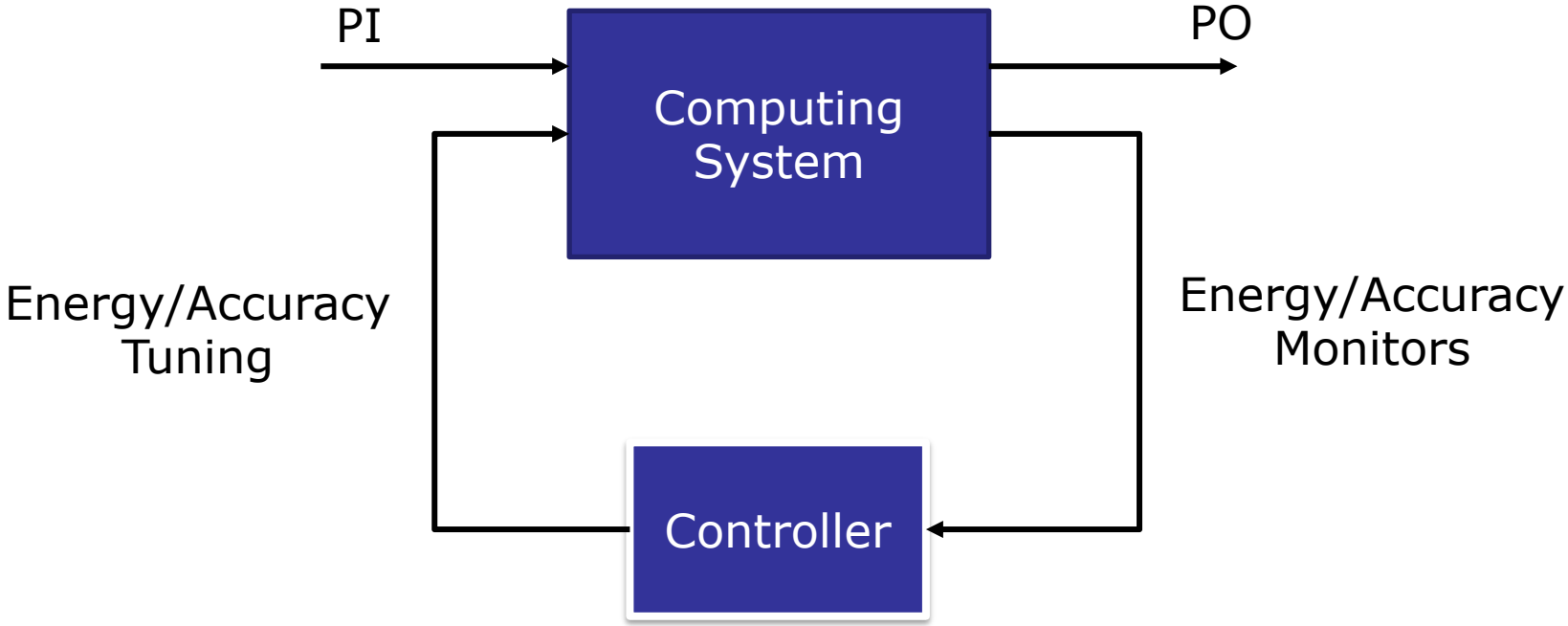

Requirements
Design
Ad-Hoc
Approximation
Manufacturing
Wafer
Final Product
Ad-Hoc
Approximation

Static (SW)


Source Code
Compiler
Code executable
Ad-Hoc Approximation
Ad-Hoc Approximation

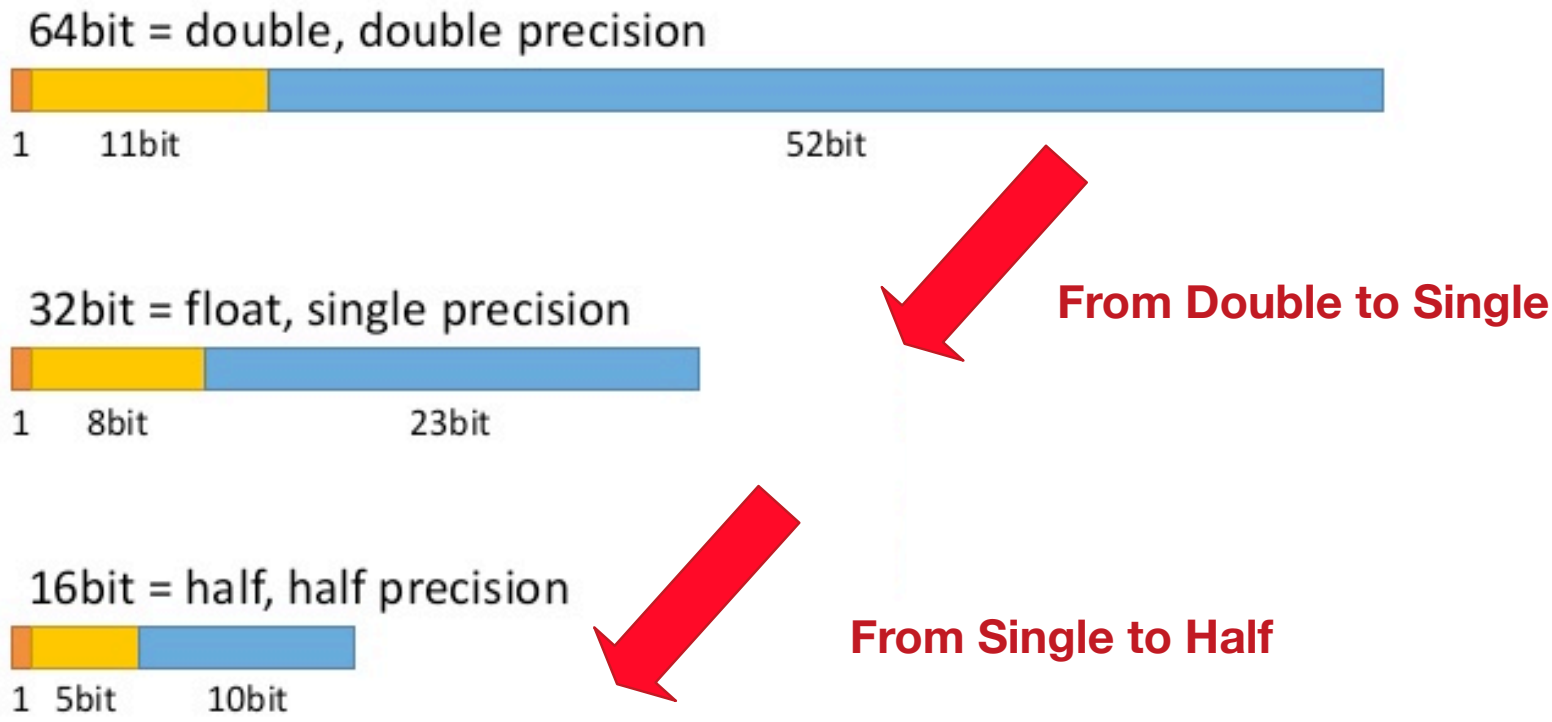
Dynamic


LOW
HIGH


PI
PO
Computing System
Energy/Accuracy Tuning
Energy/Accuracy Monitors
Controller

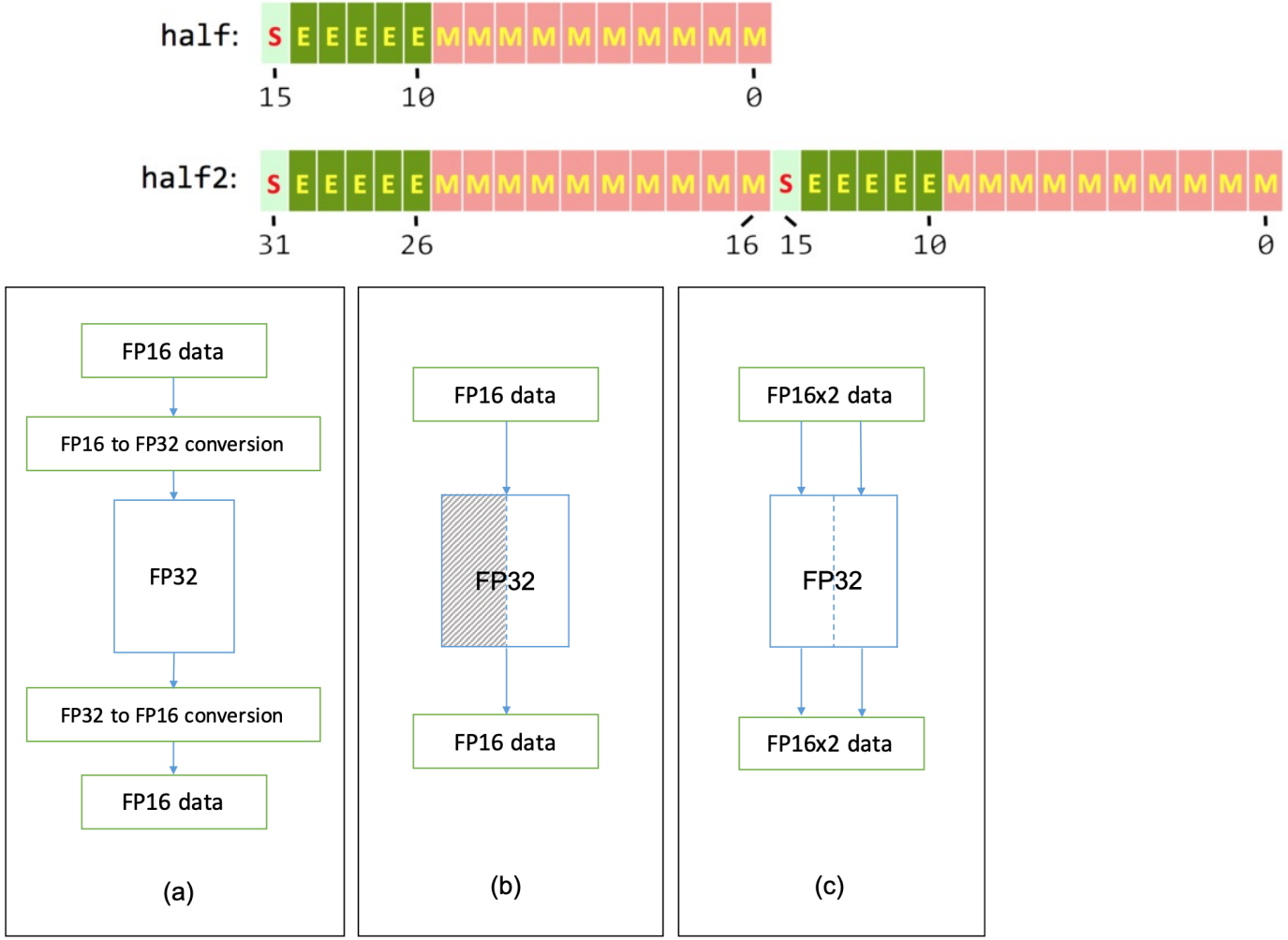
Some Examples

- Precision Reduction
  - SW/HW, static and dynamic



Commercial Example

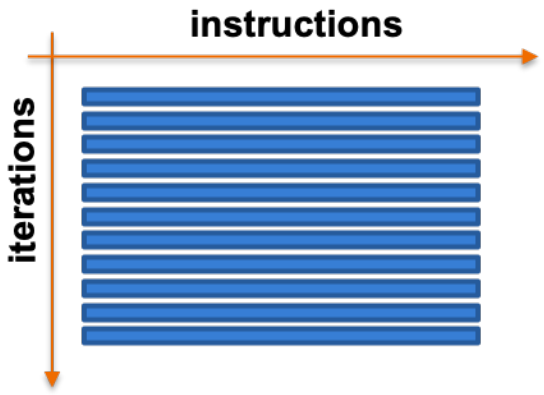
- NVIDIA from the Pascal architecture offered FP16 support.



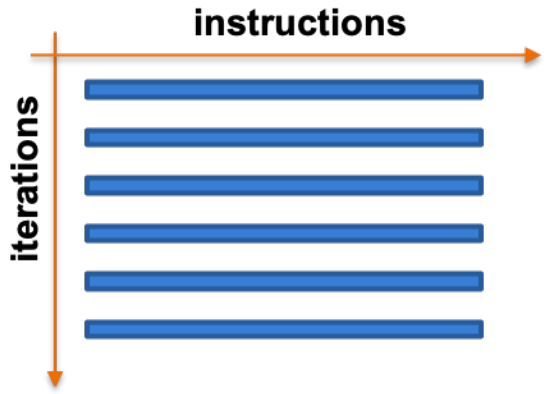
Some Examples

- Loop Perforation

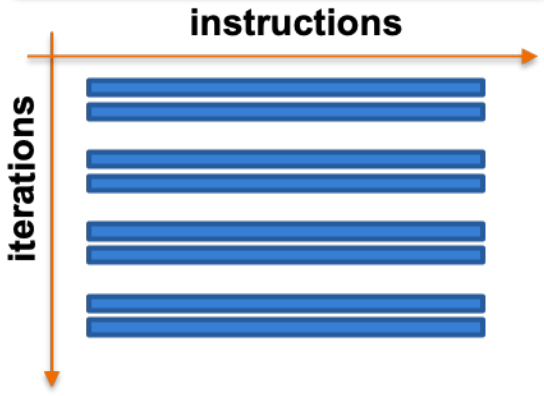
```
for (i = 0; i < b; i++)
{
        ...

}
```




```
for (i = 0; i < b; i += n)
{
        ...

}
```



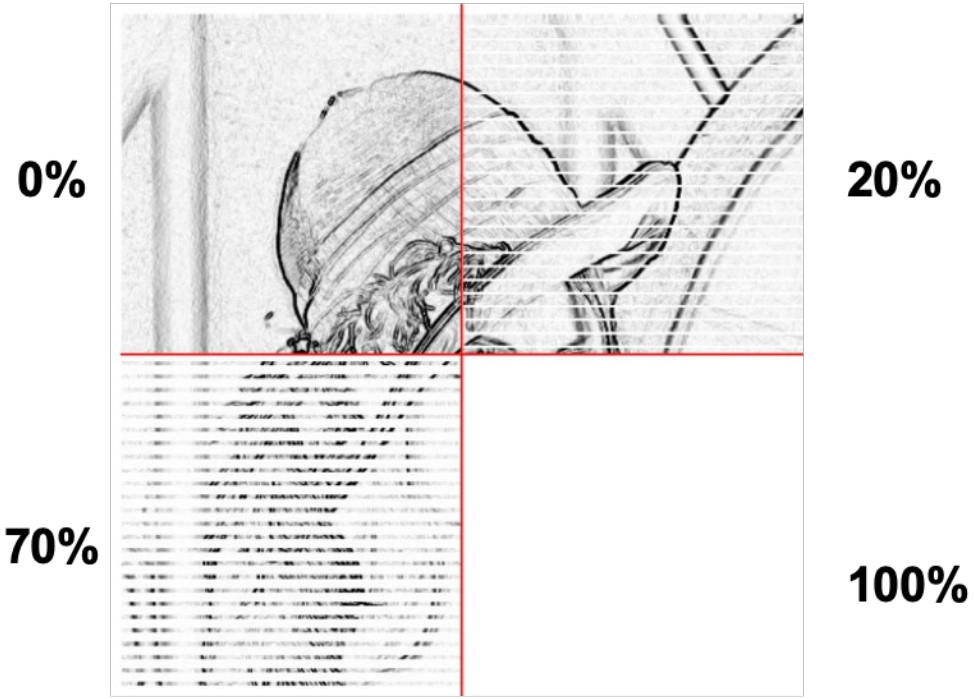

```
int count = 0;
for (i = 0; i < b; i ++)
{
    if (count == skip_factor) {
     count = 0;
    } else {
    ...

     count ++;
    }
}
```



Execute one iteration after n Skip one iteration after n

Some Examples

- Loop Perforation
  - Sobel filter for edge detection
  - Different perforation rates [Vassiliadis et al., ACMSIGPLANS, 2015.]



Some Examples

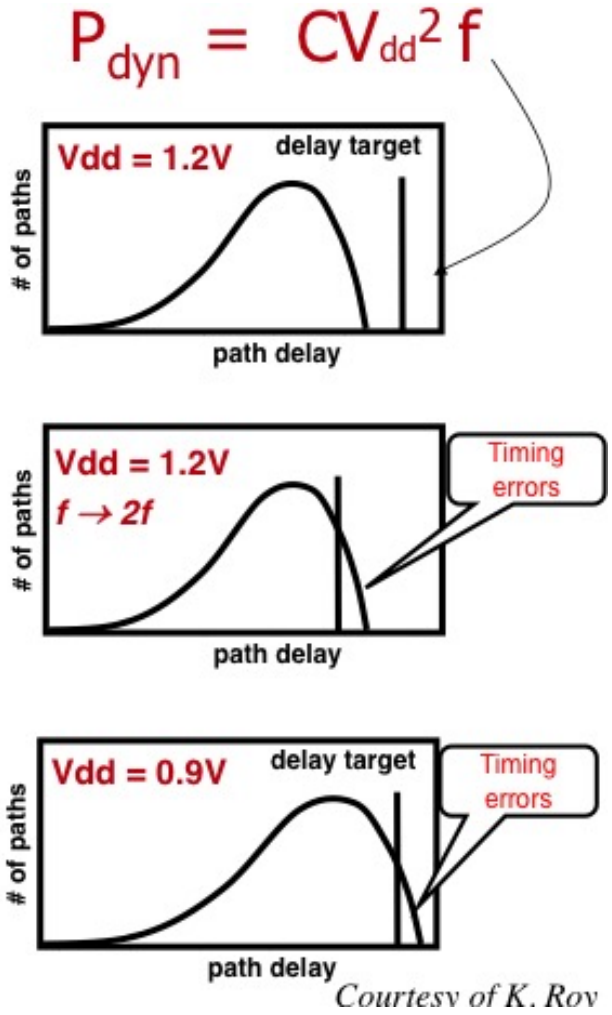
- Over-Scaling Hardware

- Power reduction tricks
  - Assume: Accurate design D_1 at frequency f_1
  - D_1 is approximated to D_2 at freq. f_2 ($f_2 > f_1$)
  - But, D_2 is run at f_1 with lower Vdd => power saving

- Design techniques
  - over-clocking
  - voltage over-scaling

↑speed

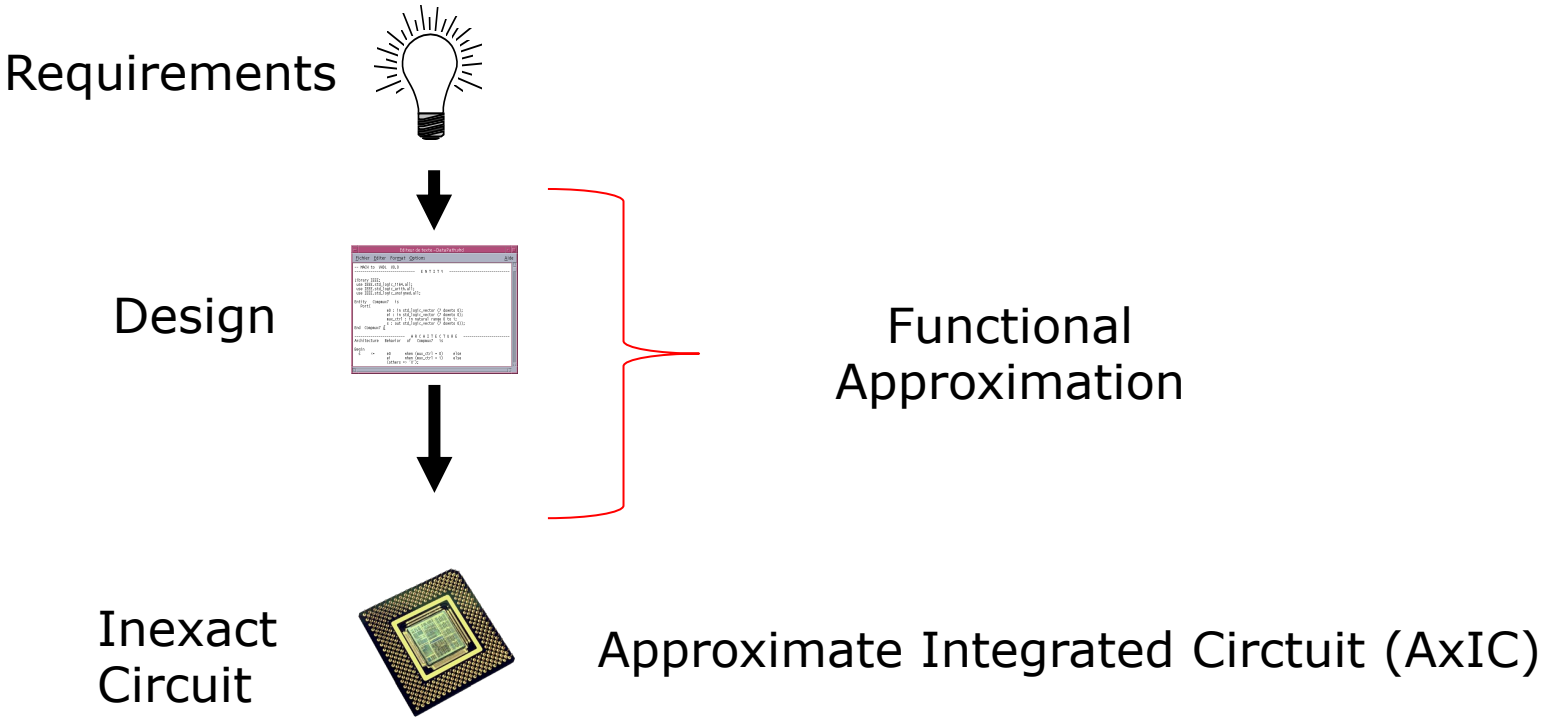
↓power



Courtesy of K. Roy

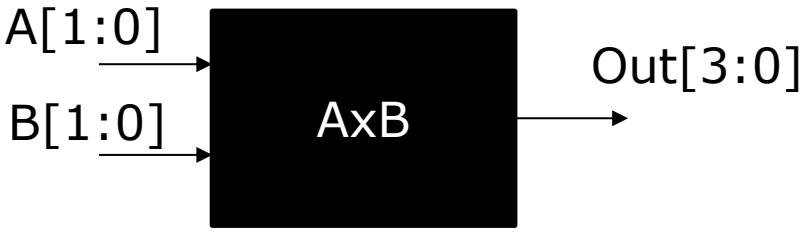
Some Examples

- Functional Approximation:
  - Modify the circuit netlist



Some Examples

- 2-bit Multiplier [Kul11]



Spec

A	B	Out
0	0	0
0	1	0
0	2	0
0	3	0
1	0	0
1	1	1
1	2	2
1	3	3
2	0	0
2	1	2
2	2	4
2	3	6
3	0	0
3	1	3
3	2	6
3	3	9

Some Examples

- Relax initial specifications

A	B	Out
0	0	0
0	1	0
0	2	0
0	3	0
1	0	0
1	1	1
1	2	2
1	3	3
2	0	0
2	1	2
2	2	4
2	3	6
3	0	0
3	1	3
3	2	6
3	3	9

A	B	Out
00	00	0000
00	01	0000
00	10	0000
00	11	0000
01	00	0000
01	01	0001
01	10	0010
01	11	0011
10	00	0000
10	01	0010
10	10	0100
10	11	0110
11	00	0000
11	01	0011
11	10	0110
11	11	1001

Relax??

Some Examples

- Relax initial specifications

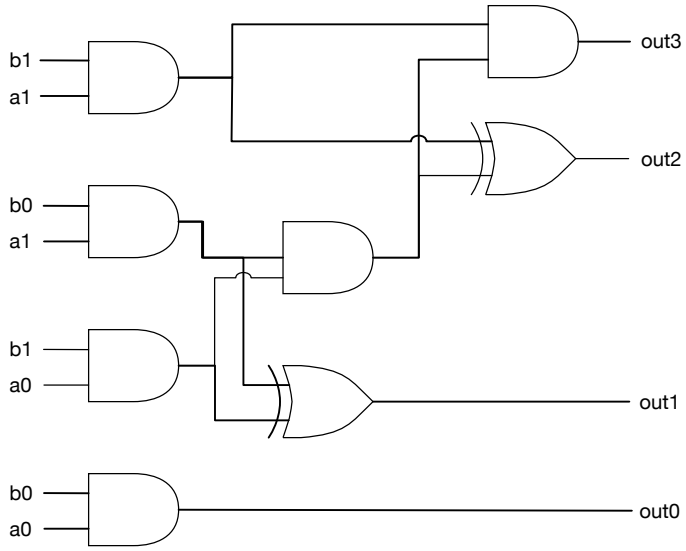
A	B	Out
0	0	0
0	1	0
0	2	0
0	3	0
1	0	0
1	1	1
1	2	2
1	3	3
2	0	0
2	1	2
2	2	4
2	3	6
3	0	0
3	1	3
3	2	6
3	3	9

A	B	Out
00	00	0000
00	01	0000
00	10	0000
00	11	0000
01	00	0000
01	01	0001
01	10	0010
01	11	0011
10	00	0000
10	01	0010
10	10	0100
10	11	0110
11	00	0000
11	01	0011
11	10	0110
11	11	1001

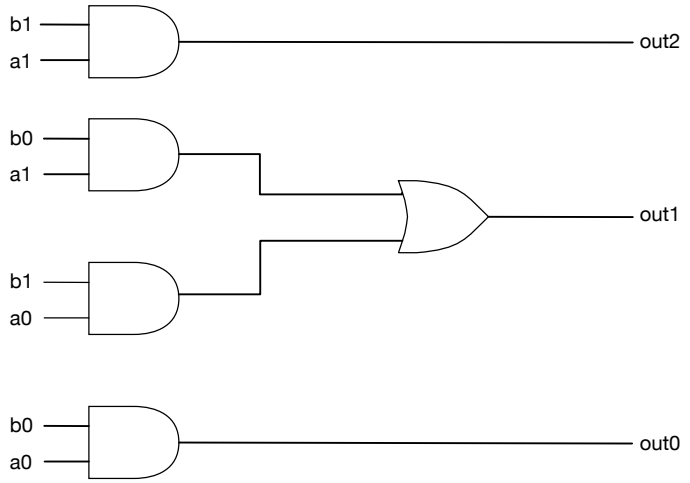
A	B	Out
00	00	0000
00	01	0000
00	10	0000
00	11	0000
01	00	0000
01	01	0001
01	10	0010
01	11	0011
10	00	0000
10	01	0010
10	10	0100
10	11	0110
11	00	0000
11	01	0011
11	10	0110
11	11	**0111**

Some Examples

- Approximate 2-bit multiplier: almost 50% area reduction; shorter delay. [Kul11]



Precise



Approximated

Metrics

- How can we evaluate the impact of a given Approximation Technique?
- It depends on the abstraction layer:
  - Component-layer: arithmetic citcuits, data precision, memory, …
  - System-layer: user related metrics such as SSI for multimedia applications.

Metrics

- How can we evaluate the impact of a given Approximation Technique?
- It depends on the abstraction layer:
  - Component-layer: arithmetic citcuits, data precision, memory, …
  - System-layer: user related metrics such as SSI for multimedia applications.

Metrics (some examples)

- Error magnitude (worst case error):

$$WCE = \max_{\forall i} \left| O_{approx}^{(i)} - O_{prec}^{(i)} \right|$$

- Average error magnitude:

$$e_{avg} = \frac{e_{tot}}{n} = \sum_{\forall i} max \frac{\left| O_{approx}^{(i)} - O_{prec}^{(i)} \right|}{n}$$

- Error probability (error rate):

$$e_{prob} = \frac{\sum_{\forall i} \left| O_{approx}^{(i)} \neq O_{prec}^{(i)} \right|}{n}$$

Metrics (2-bit multiplier example)

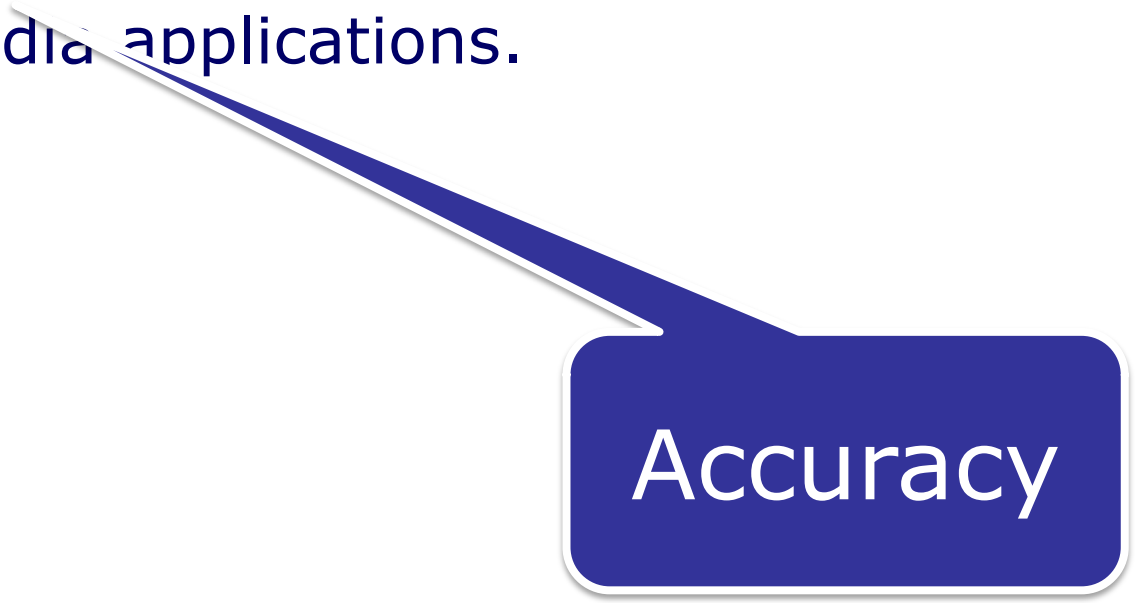
Component-level: precision of the component

A x B	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	4	6
3	0	3	6	**7**

- WCE: 2 (7 instead of 9 for 3x3);
- Average error magnitude: 2/16;
- Error Probability: 1/16;

Metrics

- How can we evaluate the impact of a given Approximation Technique?
- It depends on the abstraction layer:
  - Component-layer: arithmetic citcuits, data precision, memory, …
  - System-layer: user related metrics such as SSI for multimedia applications.



Metrics (2-bit multiplier example)

System-level: Accuracy of the result

Precise

AxC

- Application: JPEG encoding
  - Power Reduction 41.5%
  - SNR : 20.365dB;

[Kul11]

Some real applications

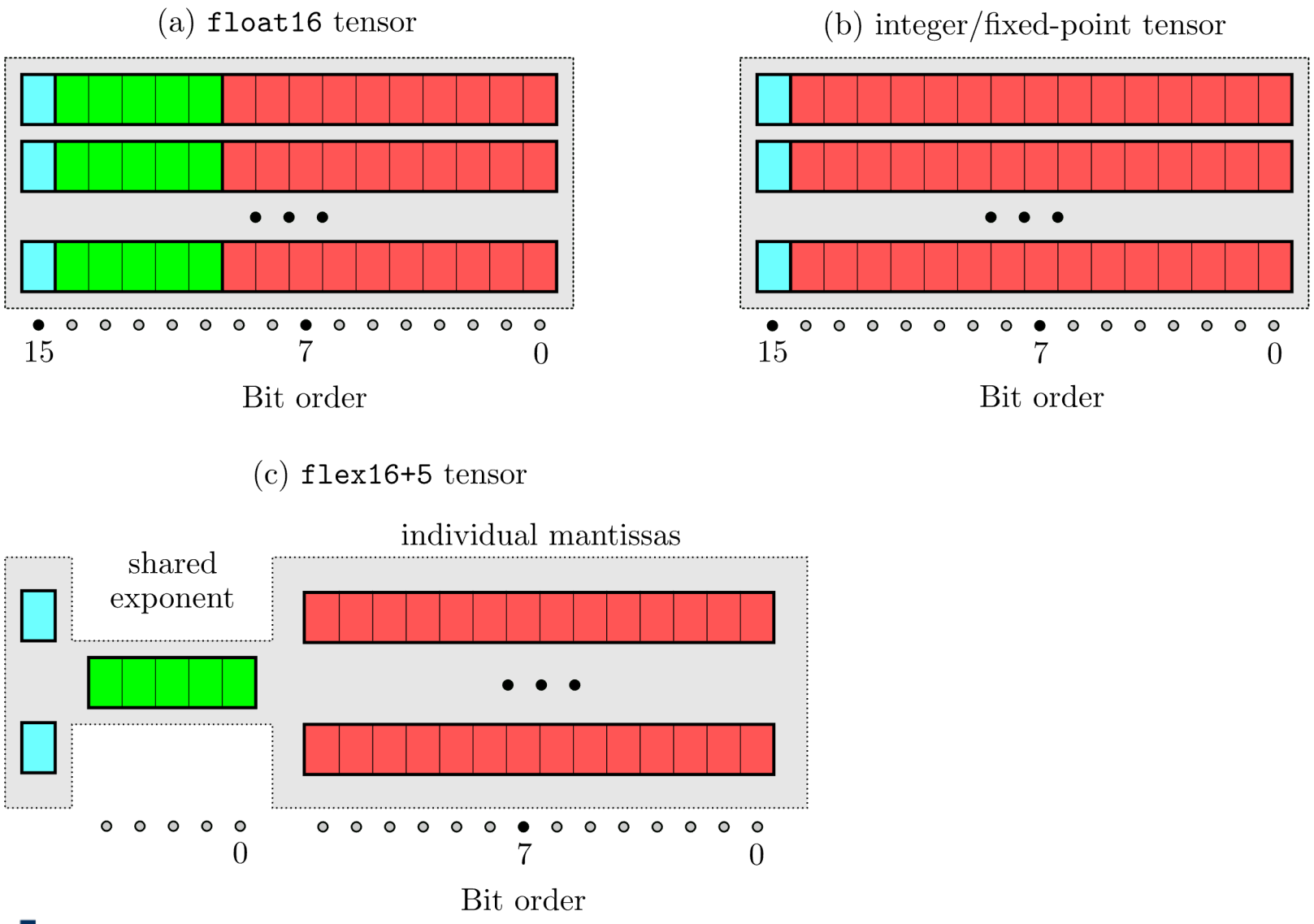
Precision Reduction for CNN

- **Quantization***: change the data type and bit-width
  - +Reduced memory footprint
  - +Simpler arithmetic circuit
- Challenges:
  - Which data type?
  - Which part(s) of the CNN to quantize;
  - Homogeneity/heterogeneity of the data type
  - When to quantize (during or after training)

*arXiv:1602.02830 , arXiv:1605.04711 , arXiv:1712.05877

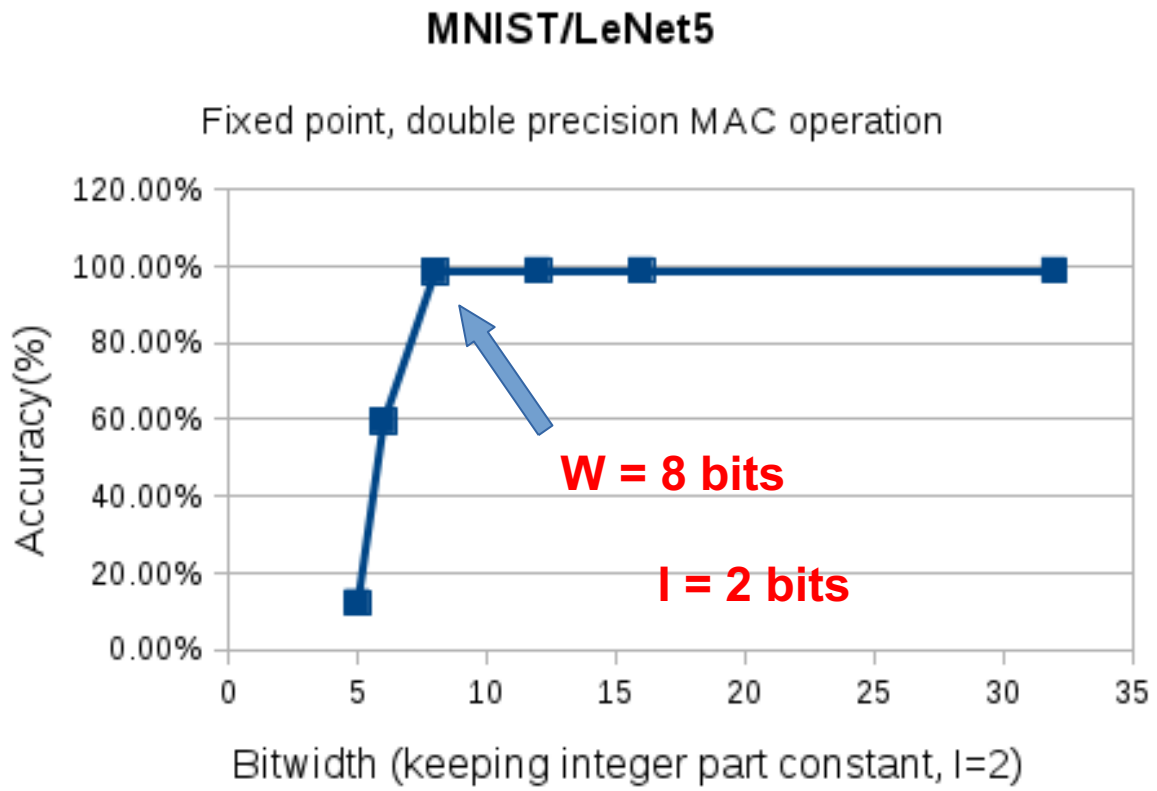
10.1007/978-3-319-46493-0_32

Data Type Representation



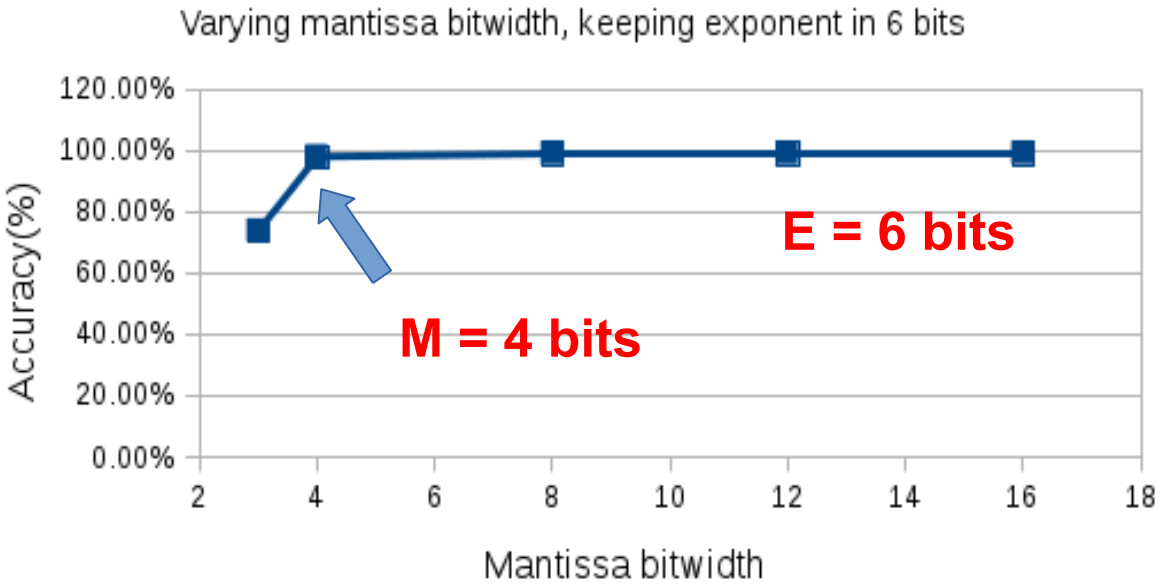
Approximate CNNs: Accuracy

- 10k images, MNIST/LeNet-5
- Fixed-Point Arithmetic



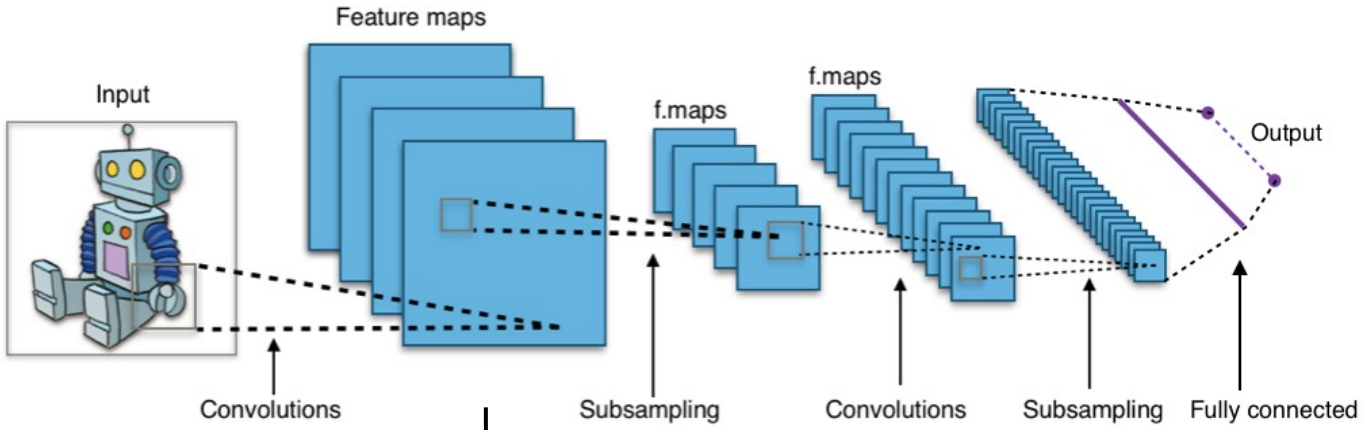
Approximate CNNs: Accuracy

- 10k images, MNIST/ LeNet-5
- Custom float



- 10-bit FlP keeps accuracy near reference
- Better results would be achieved with longer training and fine tuning and also smarter word-length opt.

Weight Sharing



5x5 Convolutional Kernel

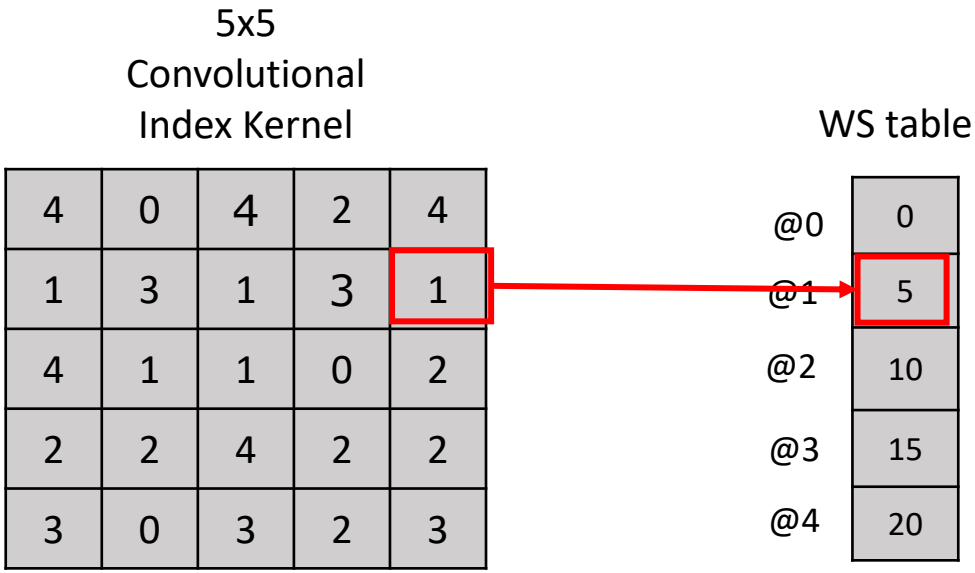
K-means Clustering

- 8 bits x W
  - 200 bits

17	3	18	11	19
5	14	9	13	7
19	5	7	0	12
10	12	20	8	10
14	1	16	10	14

17	3	18	11	19
5	14	9	13	7
19	5	7	0	12
10	12	20	8	10
14	1	16	10	14

Weight Sharing

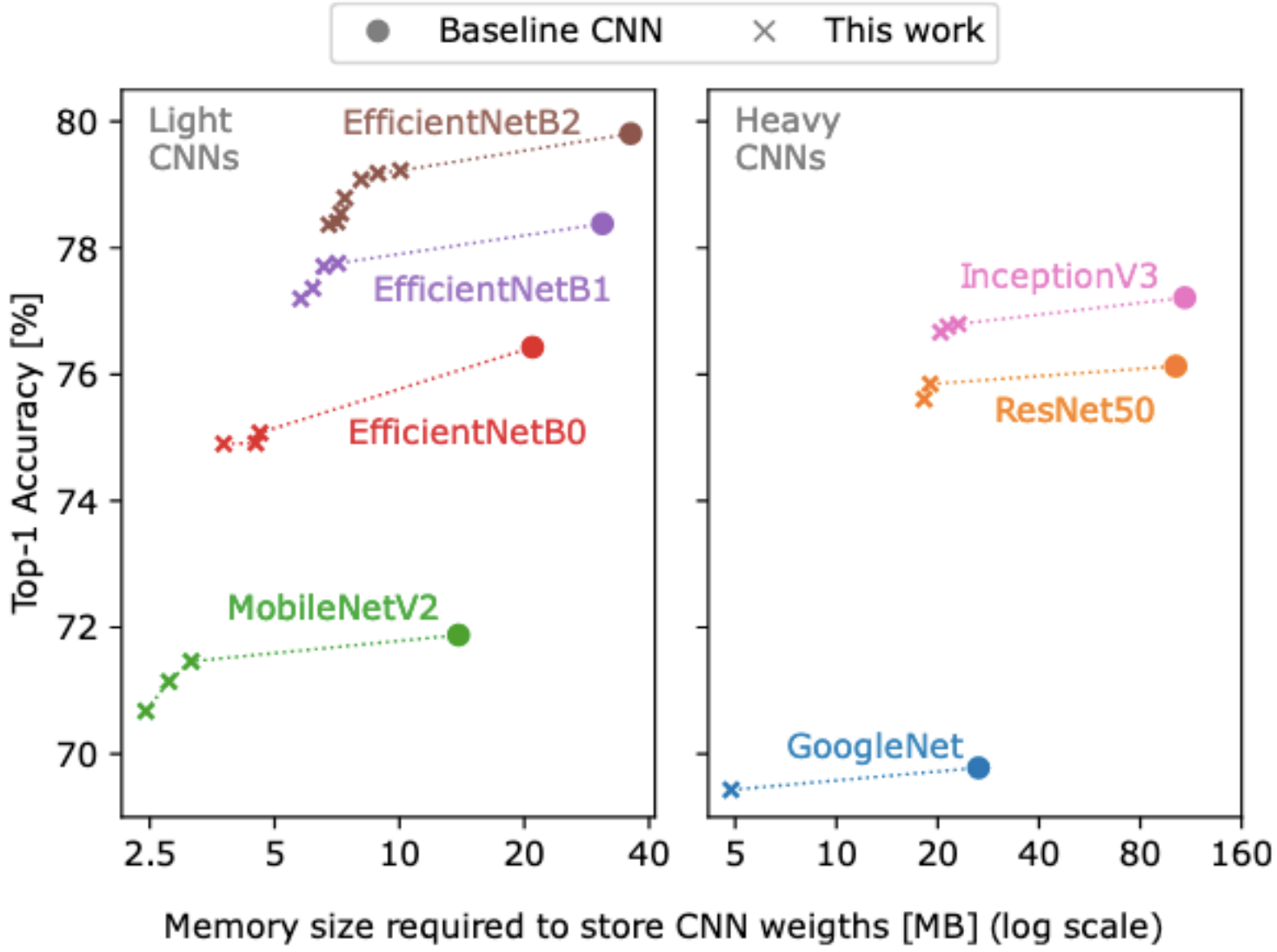


- 3 bits x W
  - 75 + 40 = 115 bits (instead of 200)
- ~42% bits reduction

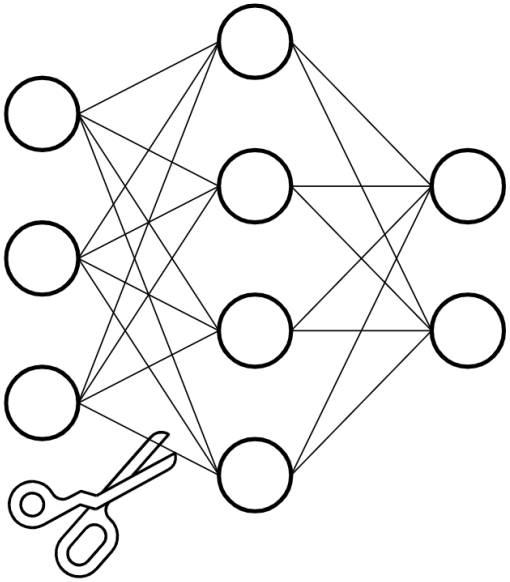
10.23919/DATE48585.2020.9116350

Results

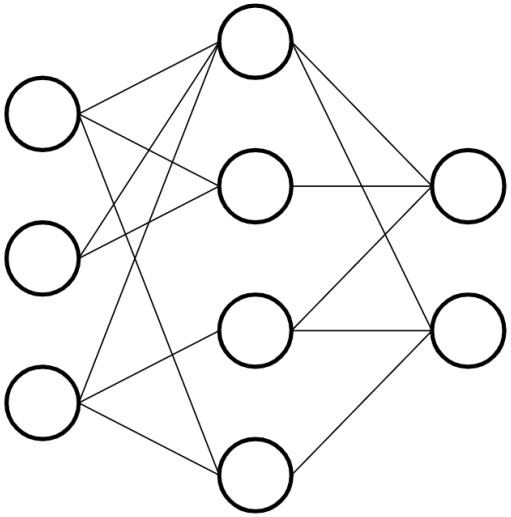
E. Dupuis, et al. AS-DAC 2022



Pruning: loop perforation for CNN



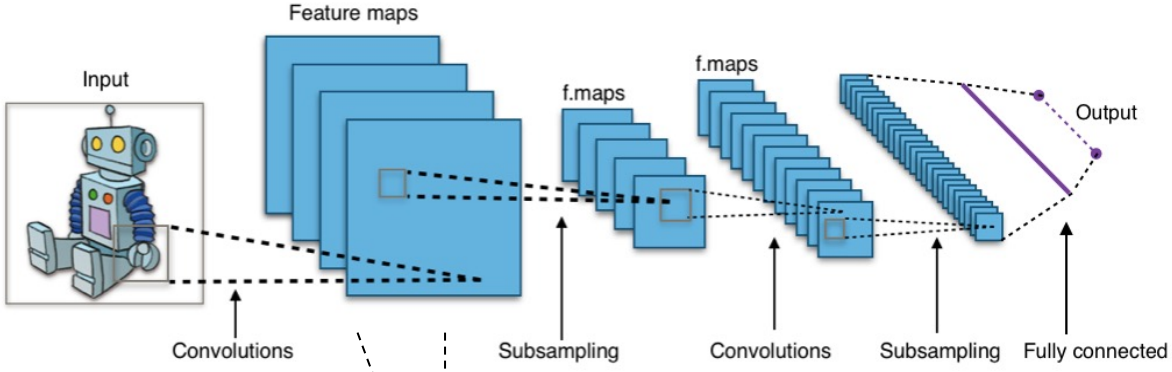
Before pruning

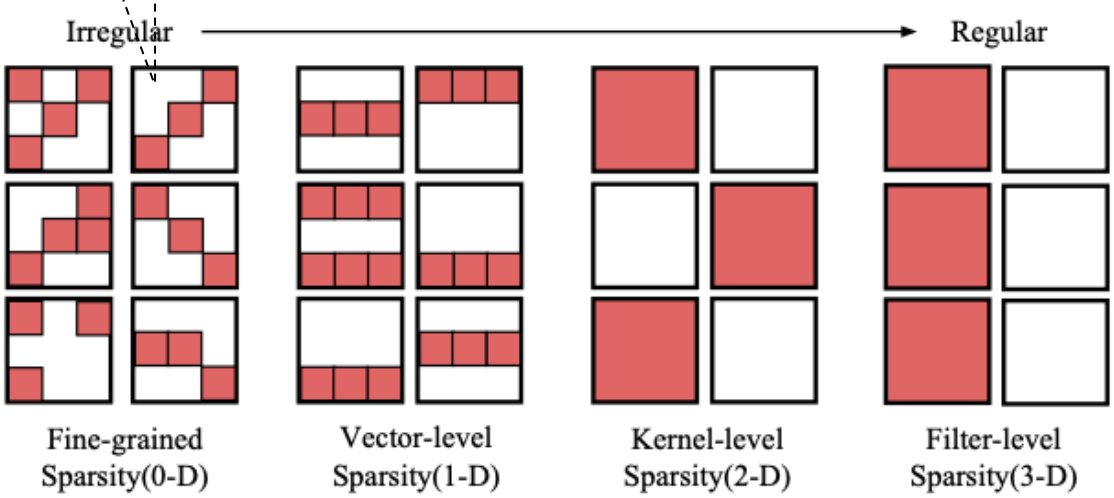


After pruning

Source: https://medium.com/tensorflow/tensorflow-model-optimization-toolkit-pruning-api-42cac9157a6a

Pruning: loop perforation for CNN






```
S)

granularity
:, :, :, :]
:, k, :, :]
:, k, r, :]
nts[c, k, r, s]
```

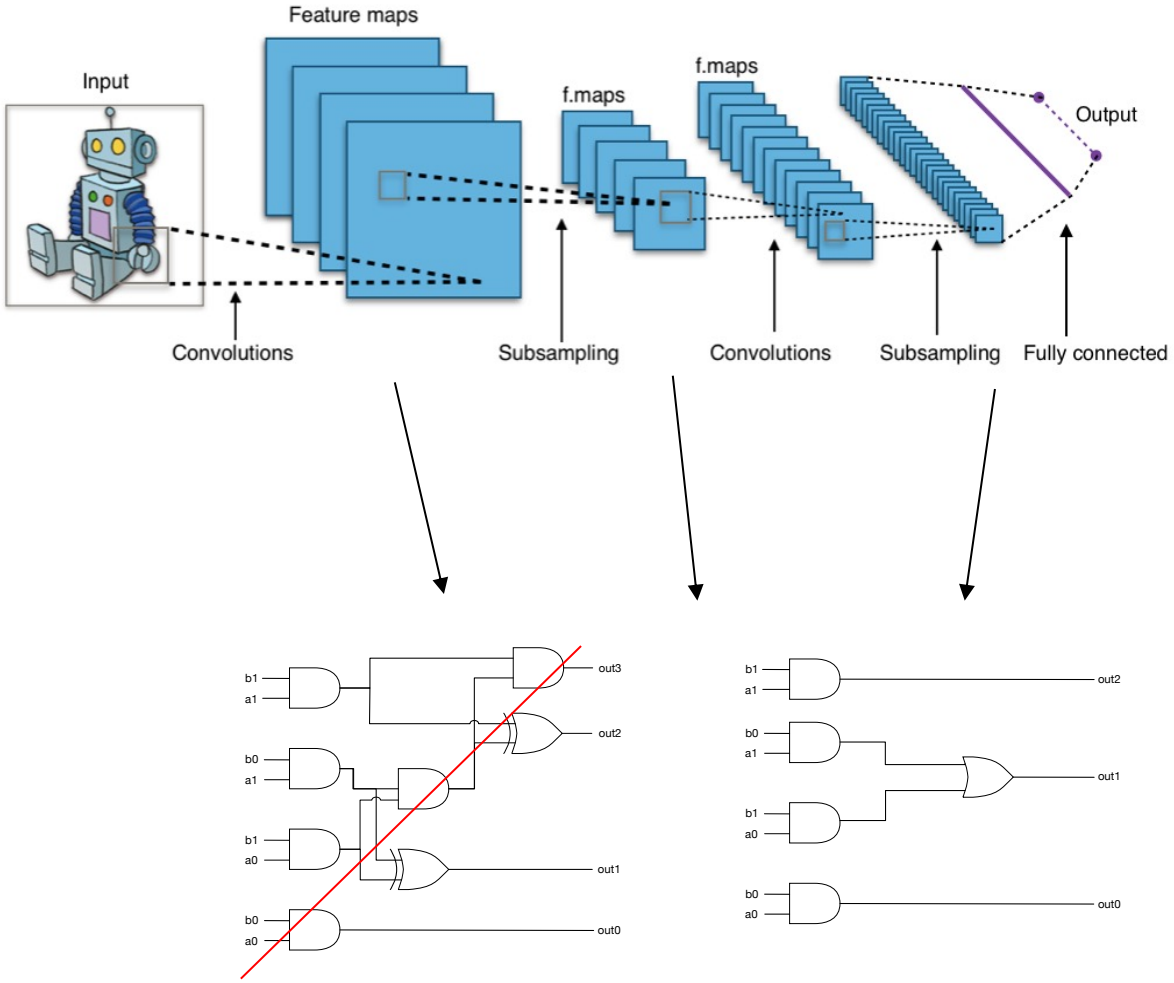
arXiv:1705.08922

Results

Model	Density	Granularity	Top-5
AlexNet	24.8%	Kernel Pruning (2-D)	79.20%
		Vector Pruning (1-D)	79.94%
		Fine-grained Pruning (0-D)	**80.41%**
VGG-16	23.5%	Kernel Pruning (2-D)	89.70%
		Vector Pruning (1-D)	90.48%
		Fine-grained Pruning (0-D)	**90.56%**
GoogLeNet	38.4%	Kernel Pruning (2-D)	88.83%
		Vector Pruning (1-D)	89.11%
		Fine-grained Pruning (0-D)	**89.40%**
ResNet-50	40.0%	Kernel Pruning (2-D)	92.07%
		Vector Pruning (1-D)	92.26%
		Fine-grained Pruning (0-D)	**92.34%**
DenseNet-121	30.1%	Kernel Pruning (2-D)	91.56%
		Vector Pruning (1-D)	91.89%
		Fine-grained Pruning (0-D)	**92.21%**

arXiv:1705.08922

Functional Approximation



10.1109/TVLSI.2019.2940943

Experiments

- Inexact Multipliers
  - EvoApprox8b Library
    - http://www.fit.vutbr.cz/research/groups/ehw/ approxlib/

Multiplier	Area (μm^2)	Power (μW)	Delay (nS)	PDP (fJ)
Exact	290.98	176.90	0.92	162.74
mul8-350	92.86	43.37	0.62	26.97
mul8-439	95.96	44.03	0.62	27.40
mul8-120	97.92	44.30	0.65	28.62
mul8-183	109.67	46.45	0.62	28.92
mul8-134	111.14	46.69	0.62	29.07

Results

Multiplier	ER	Accuracy (%)	
		MNIST	SVHN
Exact	0	97.69	86.93
mul8-350	99.0	97.70	**87.00**
mul8-439	97.8	97.71	86.96
mul8-120	98.5	97.70	**87.00**
mul8-183	97.2	97.70	86.98
mul8-134	93.9	**97.72**	86.95

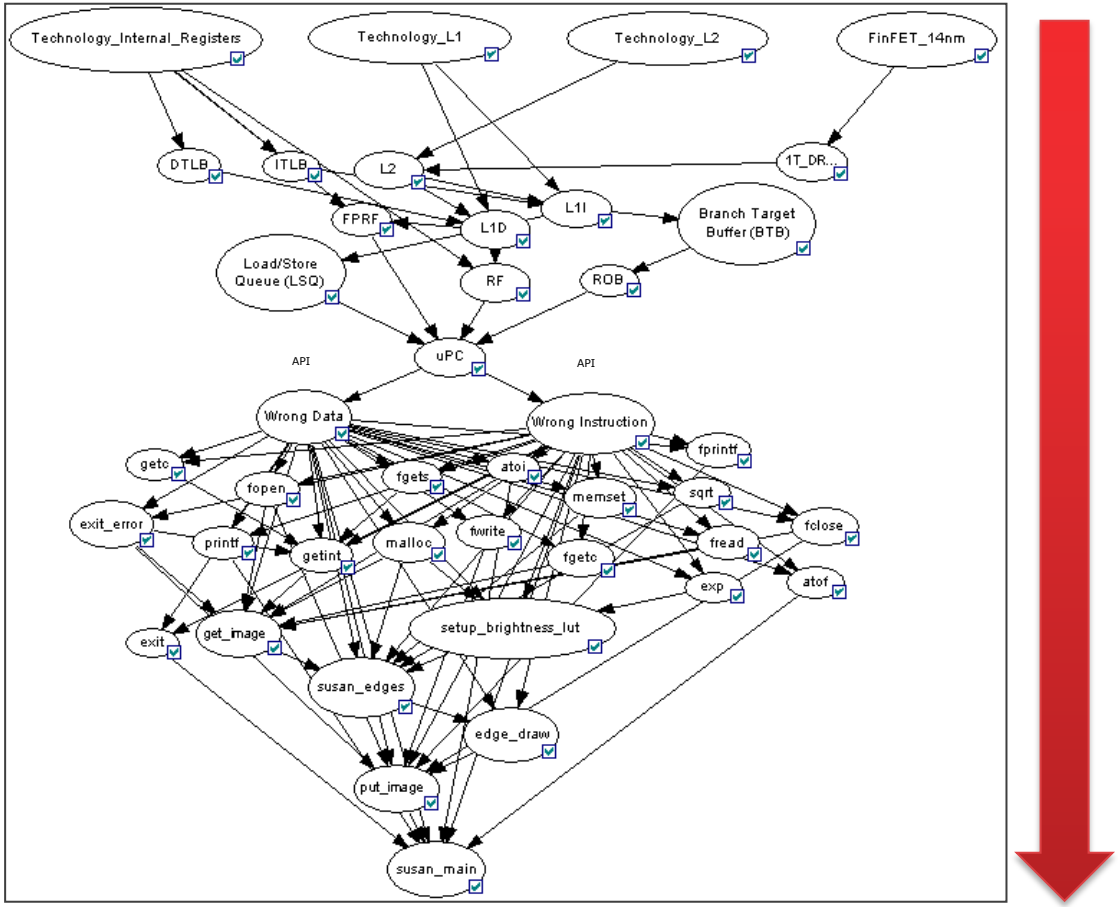
Results

Multiplier used in the neuron	Energy(fJ)	Area(μm^2)
Exact	944.08	956.02
mul8-350	269.48	367.52
mul8-439	438.29	475.72
mul8-120	553.68	599.10
mul8-183	631.76	561.40
mul8-134	801.73	583.92

- Up to 71.45% more energy-efficient
- Up to 61.55% smaller

Design Approaches

- **Bottom-Up:** From a given precision compute the accuracy



Design Flow

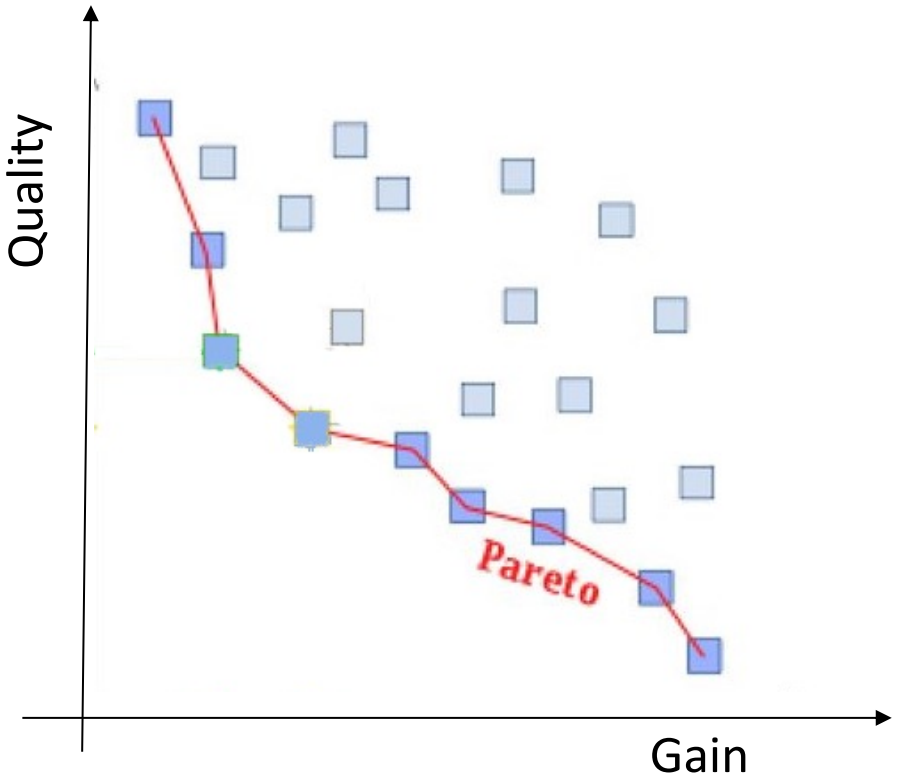
User's Requirements

- What are the inputs?
  - Error Metrics: quantify the error;
  - Error Threshold: what is acceptable;
  - Gain: what you want to optimize (e.g., area, performance, power).
- How to introduce the approximation?
  - Source-code: modify the source code and then perform a classical synthesis/compile:
    - Gate-Level: modify the netlist;
    - HDL/C code: RTL/HLS synthesis;
    - C code: compiler;

Exploring the approximation space

'Moving' through the approximation space

- Controlling the approximation parameters;
- Observing the effect of the changes.



State-of-the-Art

- RTL/C Code: Existing approaches exploit source code annotations

Original Code

```
block_i;
for(i = 0; i < n; i++){
    body
}
block_{i+1};
```

Annotated Code

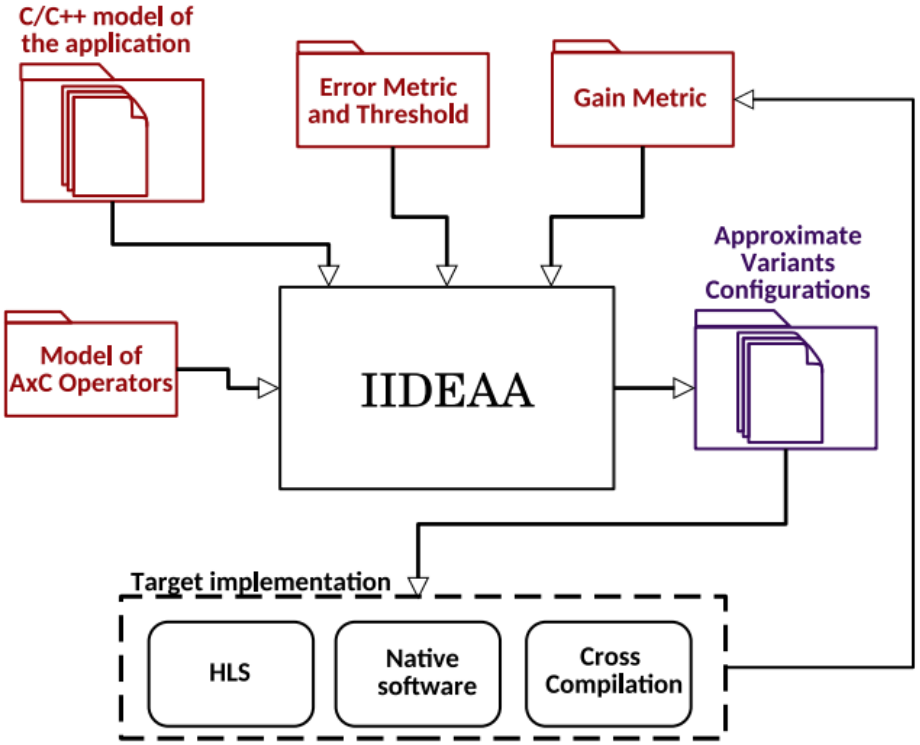
```
block_i;
#pragma approximate
for(i = 0; i < n; i ++){
    body
}
#pragma end approximate
block_{i+1};
```

- It requires a huge knowledge of the source code
- You must know the impact of your approximation..
- … and its gain

[Rub13], [Sam15], [Wys15], [Nep14]

IIDEAA

- IIDEAA Is a Design Exploration tool for Approximate Algorithms
  - https://ieeexplore.ieee.org/abstract/document/9449861



Goal: automatically finding the approximation parameters to obtain the best variants in terms of trade-off between error and gain

Chimera Example (loop perforation)

Original code:

```
1 int main (void) {
2   for (int i = 0; i < N; i++) {
3       for (int j = 0; j < M; j++) {
4           body;
5       }
6   }
7 }
```

Clang-chimera is capable to automatically mutate the code by applying an approximation operator: e.g., Loop perforation

Mutated code:

```
1 int main (void) {
2   for (int i = 0; i < N; i+=stride1) {
3       for (int j = 0; j < M; j+=stride2) {
4           body;
5       }
6   }
7 }
```

The next step is to explore the effects entailed by different values of *stride1* and *stride2*, in terms of Error and Gain.

IIDEAA explores the approximation variants through **Bellerophon**

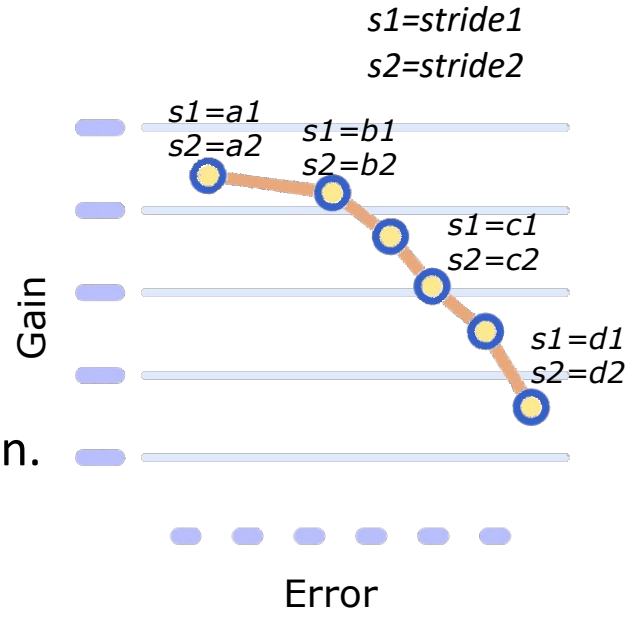
How does it work?

Bellerophon automatically performs a design space exploration (DSE).

Goal: finding the best values for approximate parameters in terms of Error and Gain.

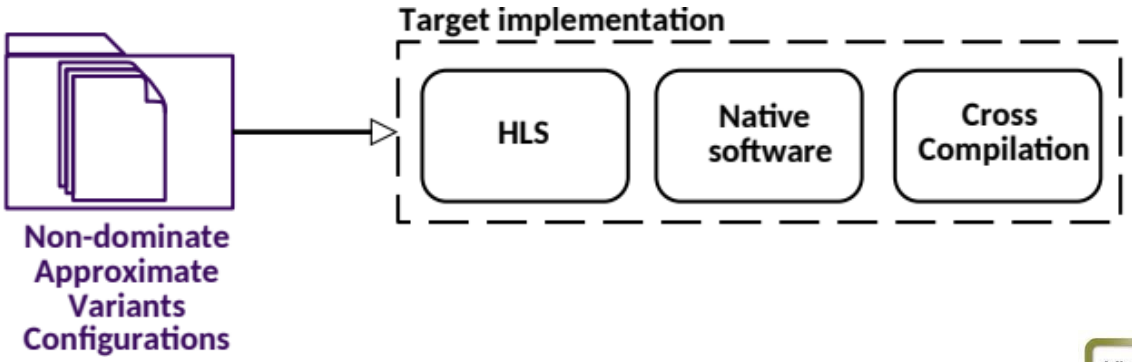
Example: Loop perforation

- *stride1* and *stride2* are the approximate parameters;
- the gain is the number of skipped loops
- the error is measured according to the application.

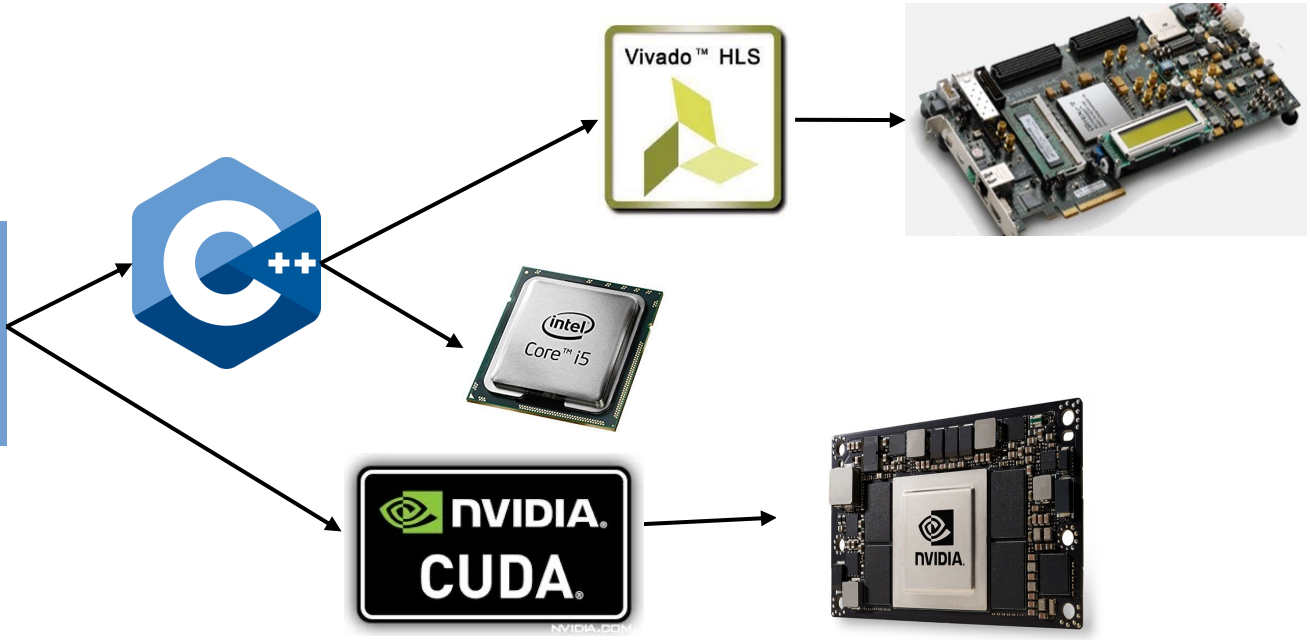


To measure both gain and error, IIDEAA executes the approximate variants.
The DSE is performed by means of a **genetic algorithm**

Target implementation



Individual	Stride1	Stride2	Error	Gain
0	3	5	5	8
1	2	9	7	11
.	.	.	.	.
.	.	.	.	.
k-1	5	2	3	7



Results

JPEG software implementation

Setup:

Discrete Cosine Transform (DCT) approximation

Approximation parameters: loop perforation

Quality fitness: structural similarity index measure (SSIM

Gain fitness: % of skipped loop

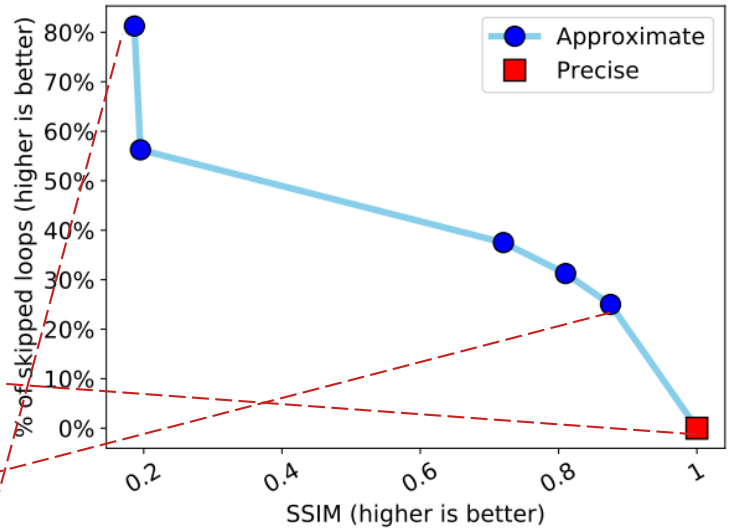
Target: CPU

DSE time: ~ few hours

(a) Input image (b) Precise output

(c) Approximate variant; SSIM = 0.87, skipped loops = 25%

(d) Approximate variant; SSIM = 0.18, skipped loops = 81.25%



Results

Sobel filter software implementation

Setup:

Approximation parameters:

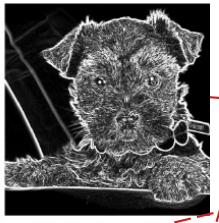
loop perforation

Quality fitness: structural similarity index measure (SSIM)

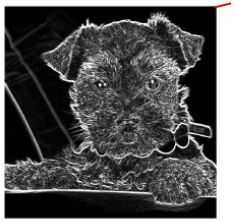
Gain fitness: % of skipped loops

Target: CPU

(a) Input image

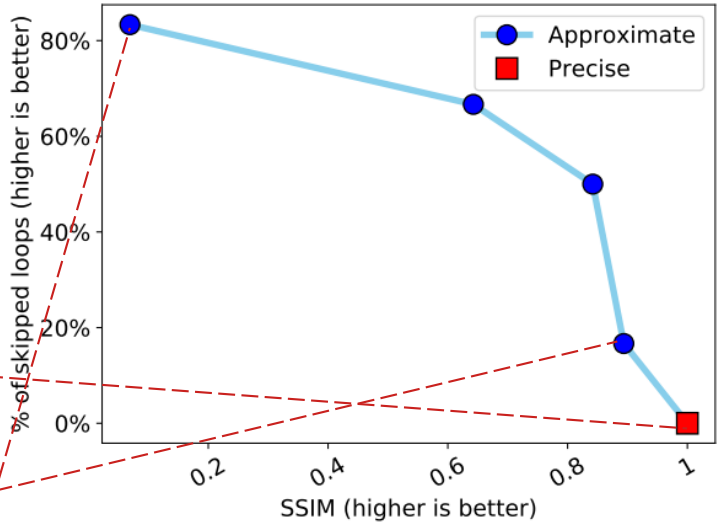


(b) Precise output



(c) Approximate variant; SSIM = 0.89, skipped loops = 16.66%

(d) Approximate variant; SSIM = 0.06, skipped loops = 83.33%



DSE time: ~ few minutes

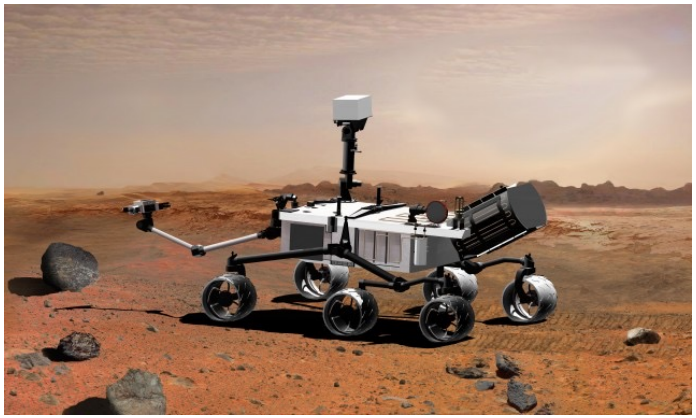
CAD tools

- Design Exploration Tools
  - From Precise C/C++ to Approximate C/C++
    - https://sampa.cs.washington.edu/research/approximation/enerj.html
    - http://wpage.unina.it/mario.barbareschi/iideaa/
  - HLS: From Precise C/C++ to HDL
    - https://link.springer.com/chapter/10.1007/978-3-319-99322-5_10
- Approximate Logic Synthesis (ALS)
  - **[DATE'10] Minterm Complement Technique**
  - **[LASCAS'15]** Pruning Technique
  - **[ICCAD'13] GALS:** ALS under EM and ER constraints
  - **[ICCAD'14] MALS:** ML-ALS under EM and ER constraints
  - **[DAC'12] SALSA**: ALS as Traditional Logic Synthesis
  - **[ICCAD'14] ASLAN:** ALS for Sequential Circuits
  - **[DATE'17] Evoapprox8b:** ALS using genetic algorithms

Agenda

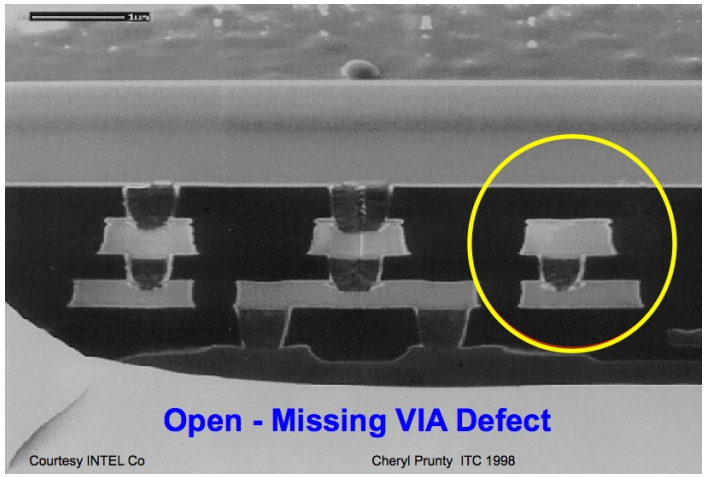
- Introduction
- Approximate Computing
- Techniques and DSE
- Approximate Computing For Safety-Critical Systems
- Conclusions

Safety-Critical Systems

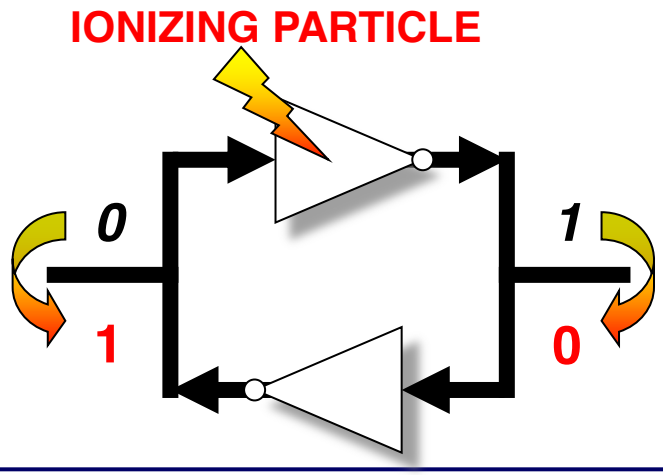


What are the consequences?

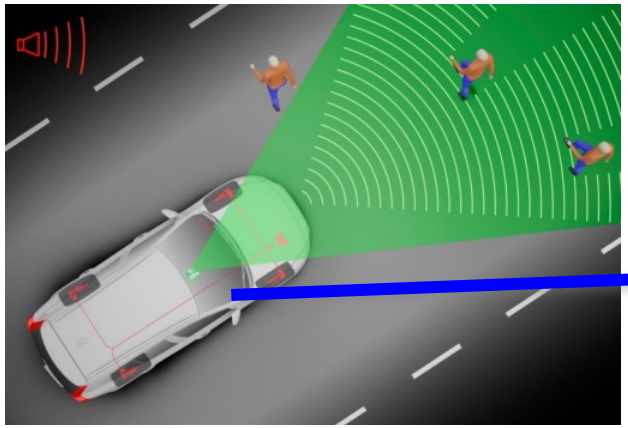
- Defects in the HW
  - Manufacturing Defects

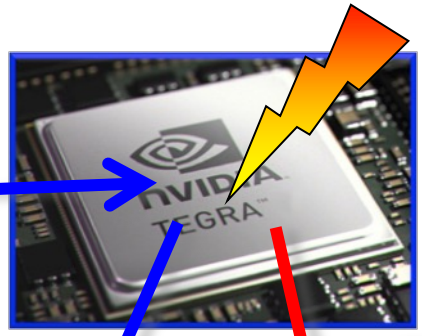


- Ageing
- Harsh Environment:
  - Neutron radiations from cosmic rays, alpha particles from packaging materials and environmental/design variations are common causes of **perturbations**

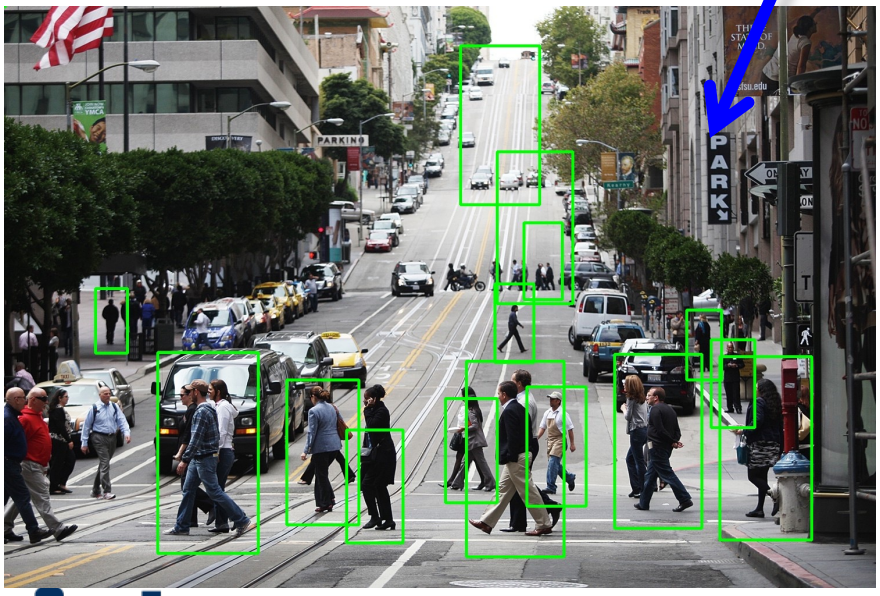


Error Classification



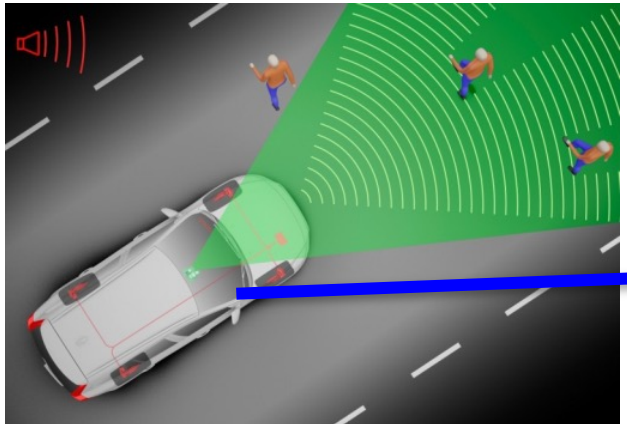


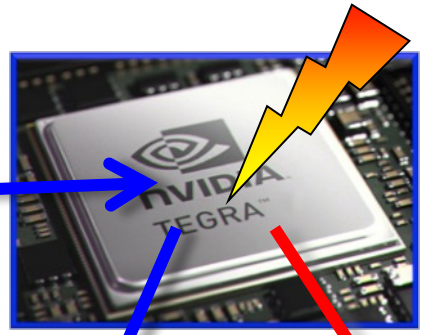
Observed failures under neutron beam



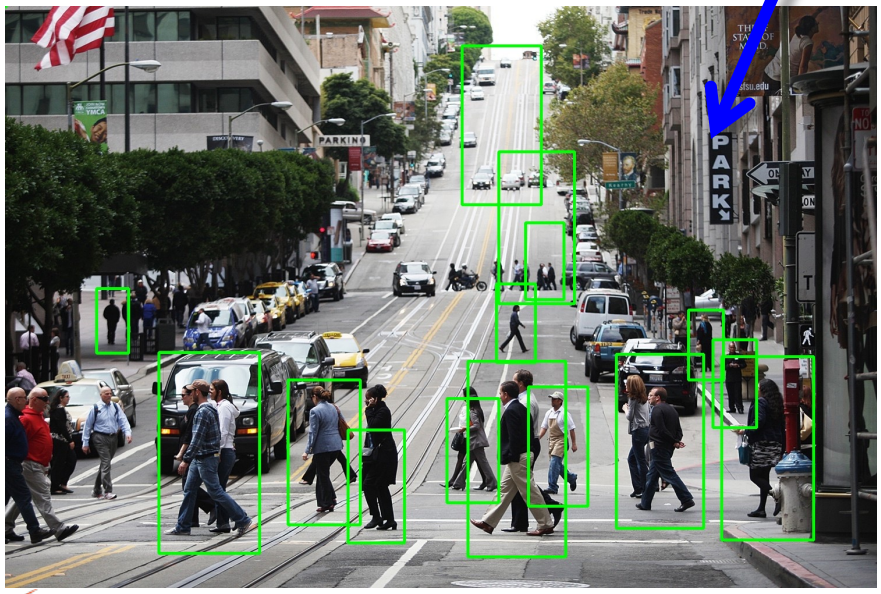
P. Rech et al. [VTS'18]

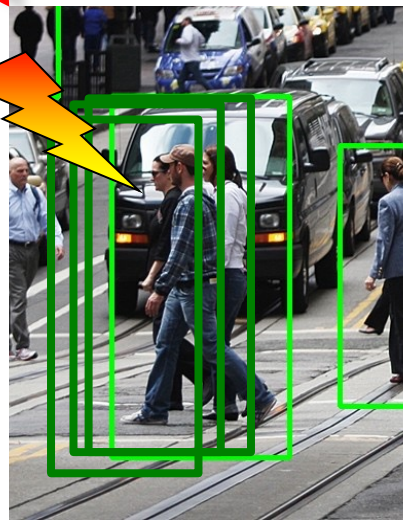
Error Classification





Tolerable errors

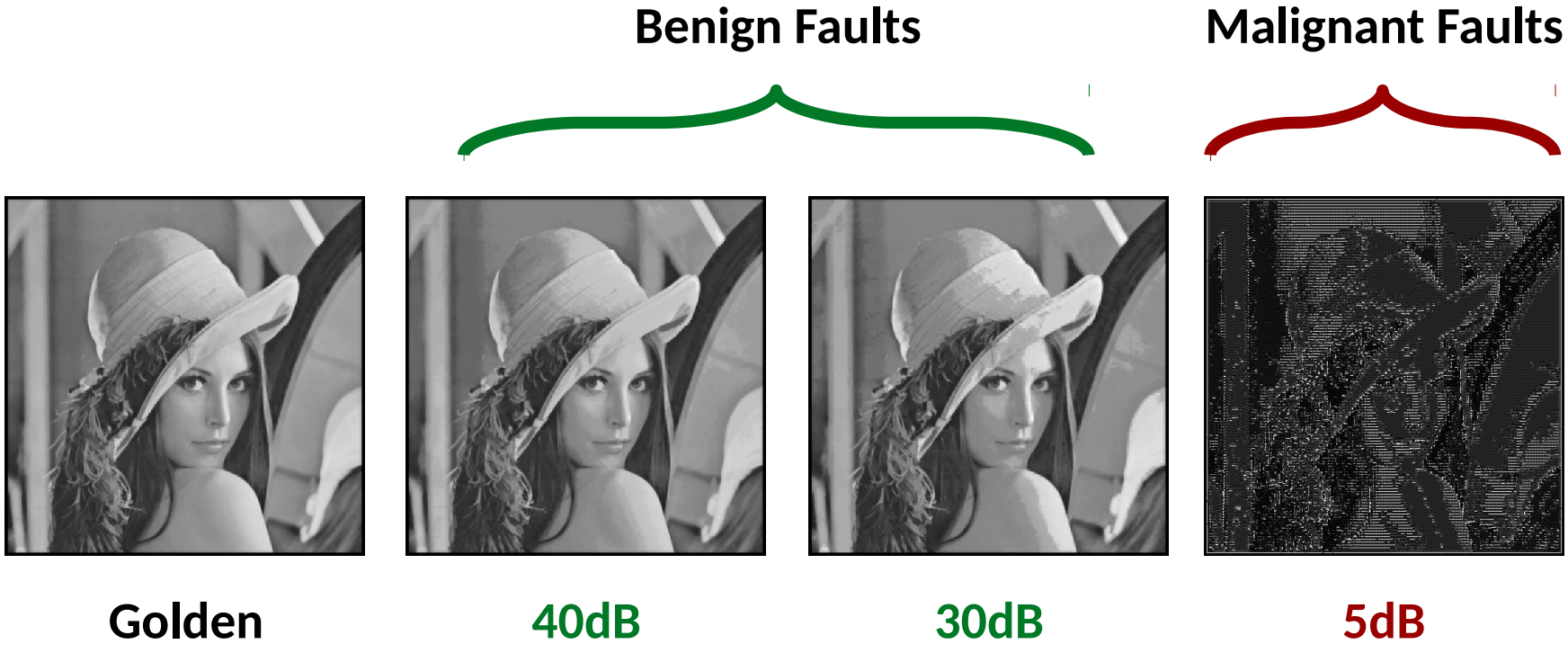




P. Rech et al. [VTS'18]

Error Classification

- Observed failures can be very different!

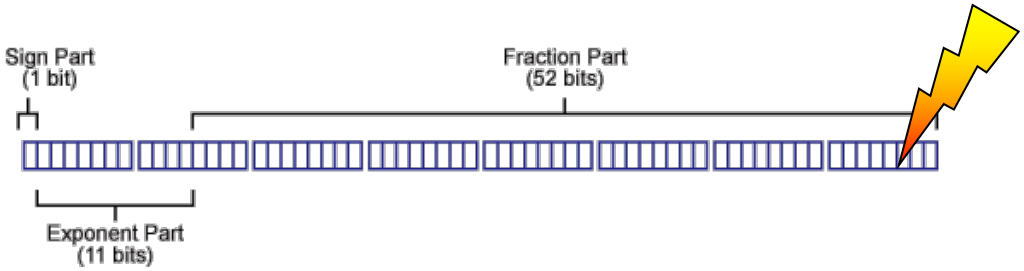


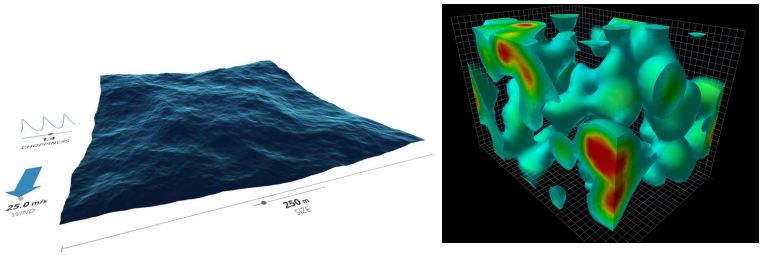
M. Benabdenbi et al. [JETTA'18]

Error Classification

- Observed failures can be very different!

error can be in the float intrinsic variance





Values in a given range are accepted as correct in physical simulations

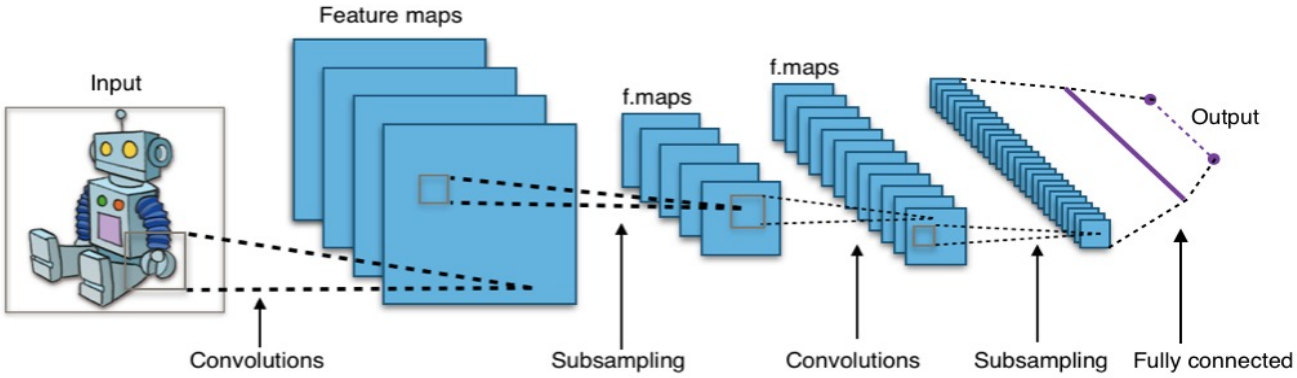
P. Rech et al. [VTS'18]

Reliability VS Approximation

- An **approximate** application is more or less reliable w.r.t. the **precise** application?
  - Issue?

Reliability VS Approximation

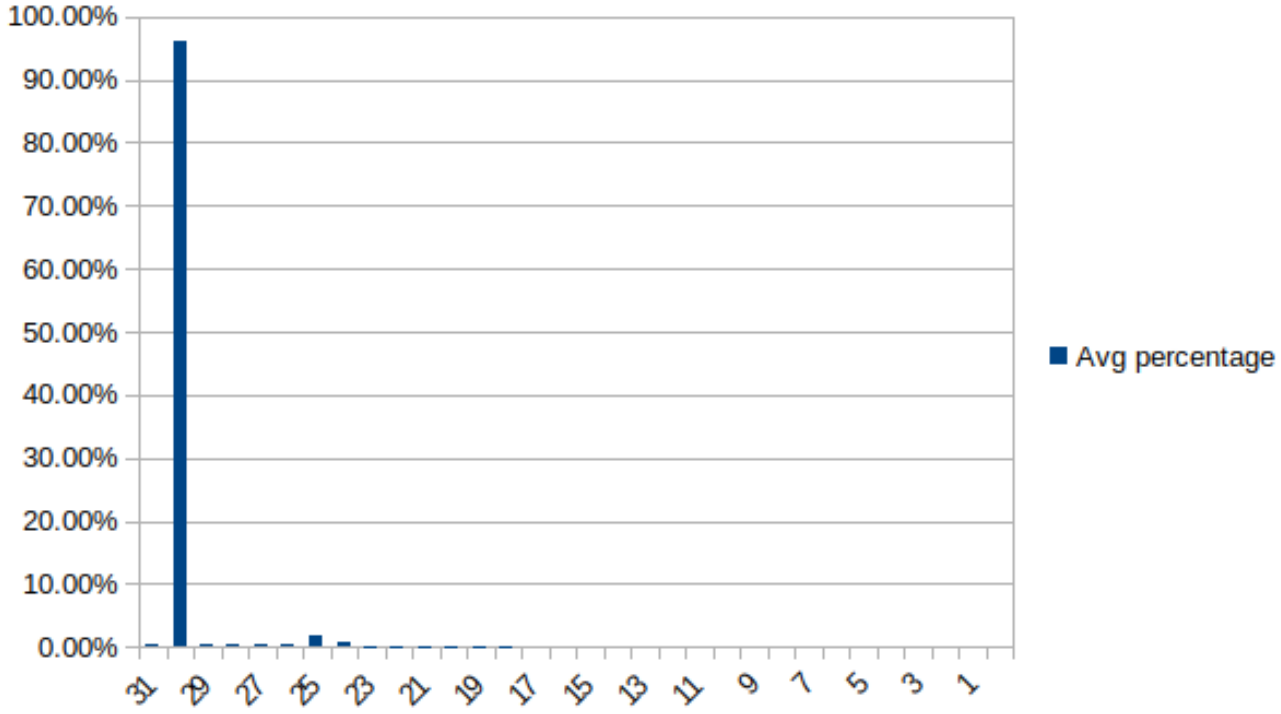
- Example: Lenet



- Precise version:
  - Weights stored as 32 bit floating point data types.

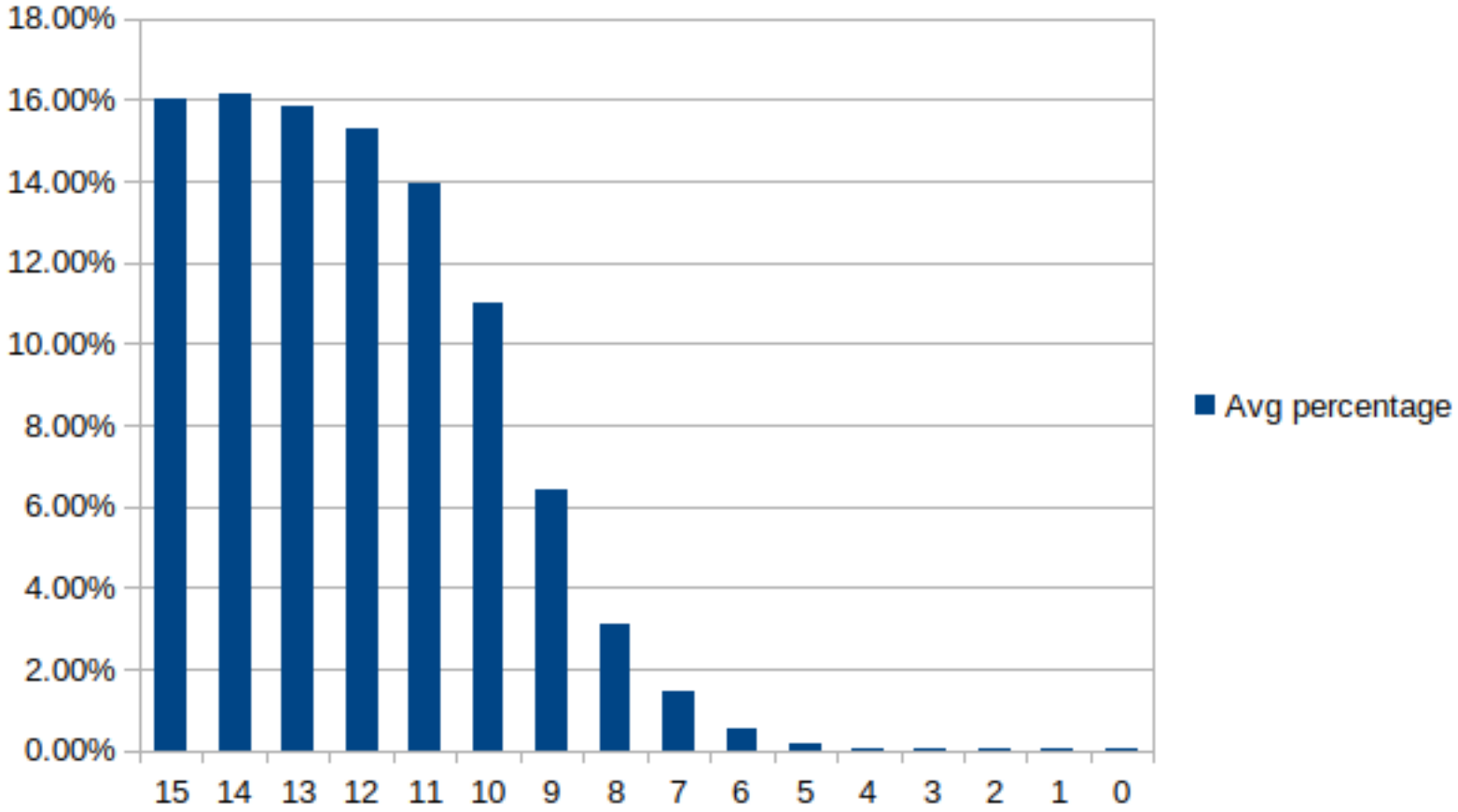
Reliability VS Approximation

- Fault Injection on the weights [DSD'20]



Reliability VS Approximation

- Approximate Lenet
  - From 32 to 16 bit fixed point data type (9,7)
  - Fault Injection

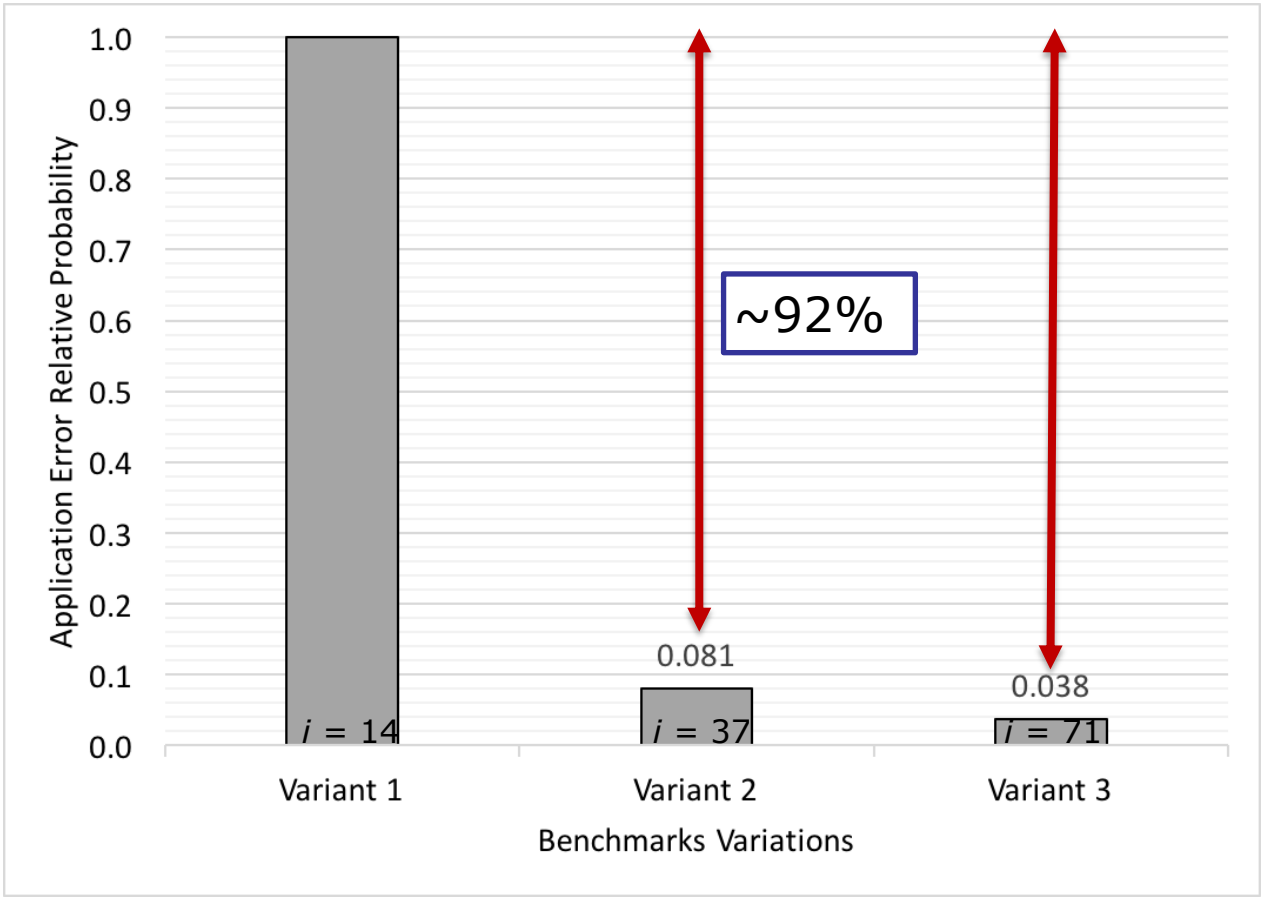


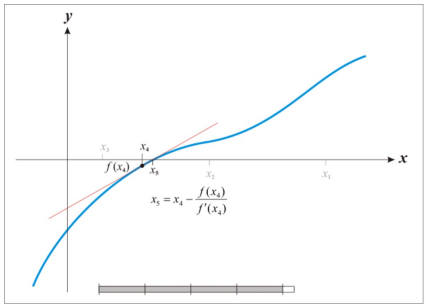
Reliability VS Approximation

- Some data considering 1 MB of memory
  - Precise:
    - 512 KB of weights in 32 bit floating points
      - 131K weights -> 1 critical bit per weigth -> 131Kbit/8Mbit => 16 * 10^{-3} -> **1.6 * 10^{-2}**
  - Approximate:
    - 256 KB of weights in 16 bit fixed points:
      - 6 critical bits per weigth -> 1536 Kbit / 8 Mbit -> 192 * 10-3 -> **1.92 * 10^{-1}**

Approximate Lenet is more energy efficient but **less reliable**

Another Example: Newton-Raphson

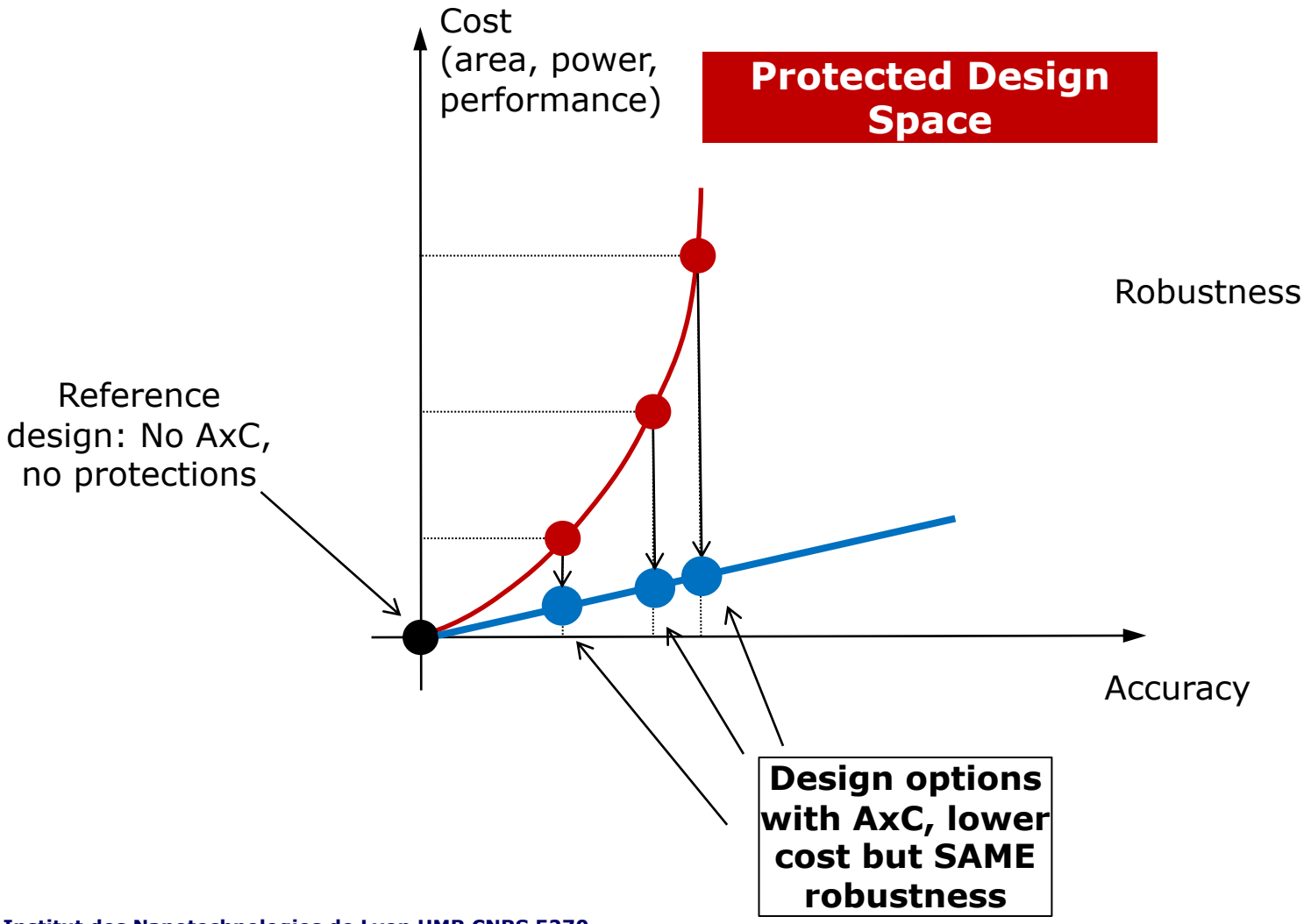




[M&R 2019]

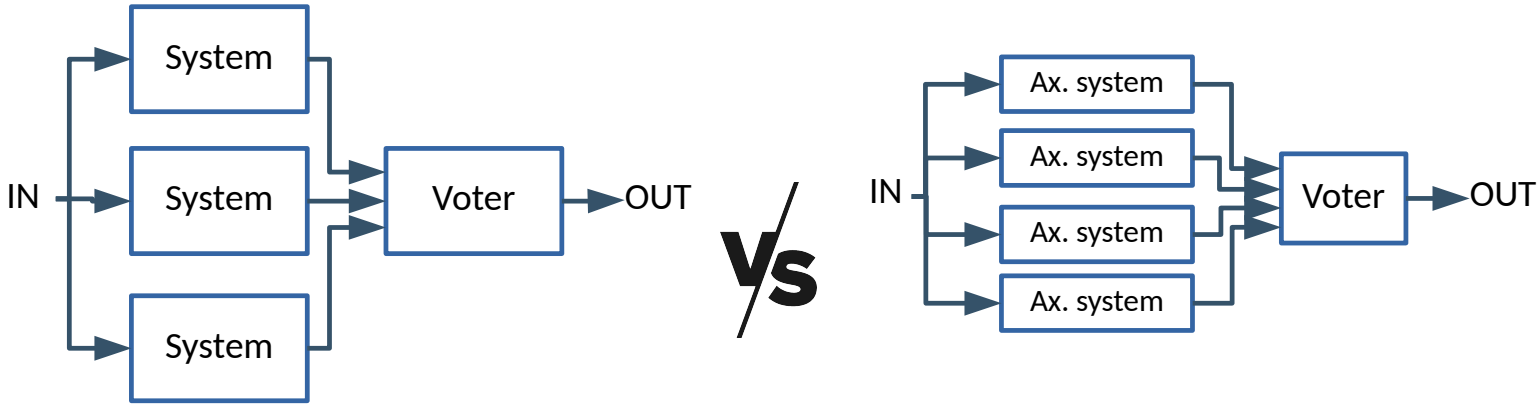
Reliability: Approximate Fault Tolerance

- Opportunity



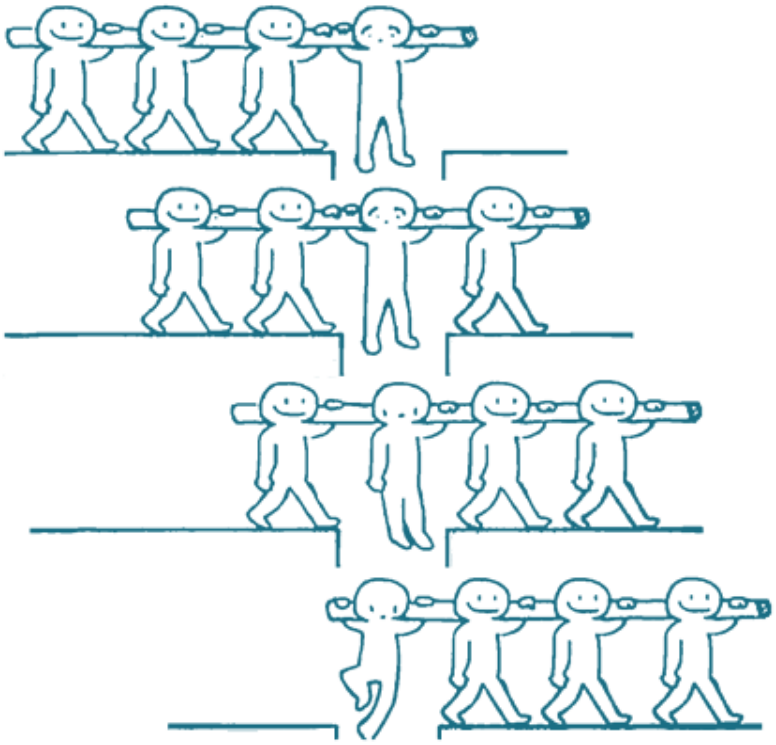
Quadruple Approximate Modular Redundancy

- QAMR guarantees the same reliability level as a full TMR [ETS'20][PESW'20]
  - Underlying insight: **a good approximation achieves more gains than it reduces the system accuracy**

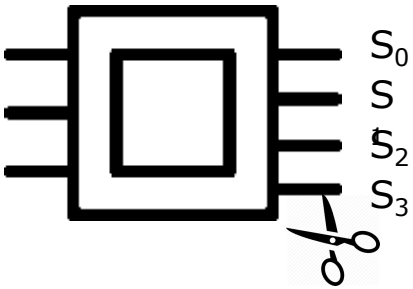


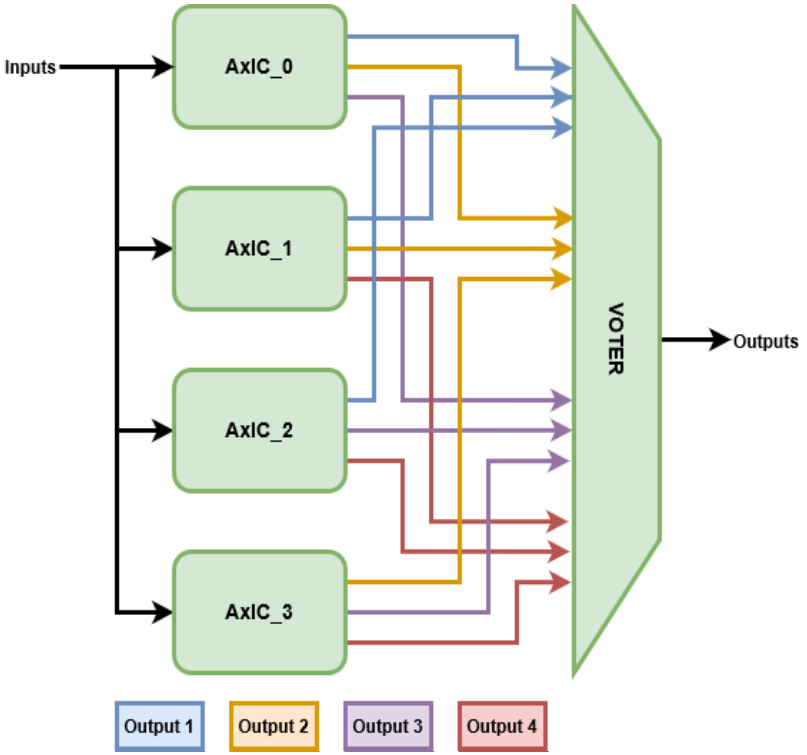
QAMR: How does it works?

- When one module delivers an approximate (i.e., wrong) response, the others must deliver the precise (i.e., correct) one



Output-cutting-based QAMR





The voter is the same as the TMR!

For each output, we still have three replicas

B. Deveautour, M. Traiola, A. Virazel, P Girard, "QAMR: an Approximation-Based Fully Reliable TMR Alternative for Area Overhead Reduction" - IEEE European Test Symposium (ETS) 2020

How to implement QAMR

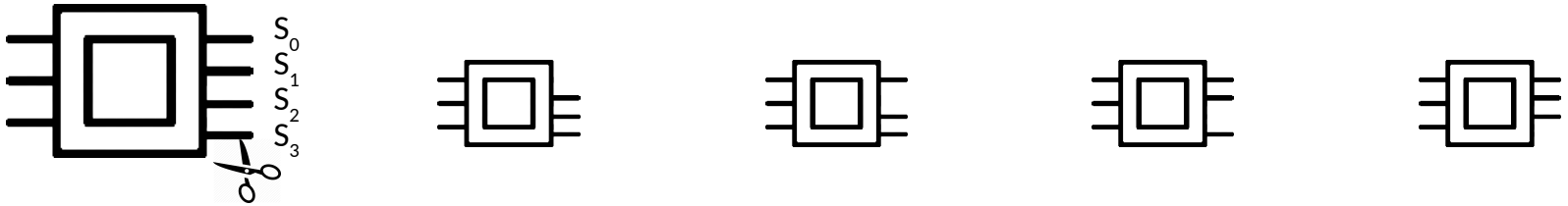
A	B	C	O_0	O_1	O_2	O_3
0	0	0	1	0	1	1
0	0	1	0	0	0	1
0	1	0	0	1	0	0
0	1	1	1	1	1	1
1	0	0	0	0	0	1
1	0	1	0	1	1	1
1	1	0	1	0	1	0
1	1	1	1	0	1	0

A	B	C	O_1	O_2	O_3
0	0	0	0	1	1
0	0	1	0	0	1
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	0	0	1
1	0	1	1	1	1
1	1	0	0	1	0
1	1	1	0	1	0

A	B	C	O_0	O_2	O_3
0	0	0	1	1	1
0	0	1	0	0	1
0	1	0	0	0	0
0	1	1	1	1	1
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	1	1	0
1	1	1	1	1	0

A	B	C	O_0	O_1	O_3
0	0	0	1	0	1
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	1	1	1
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	1	0	0

A	B	C	O_0	O_1	O_2
0	0	0	1	0	1
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	0	1



For a circuit having a set of outputs S, the subset of outputs S_i will be removed from the AxIC$_i$.

Constraints:

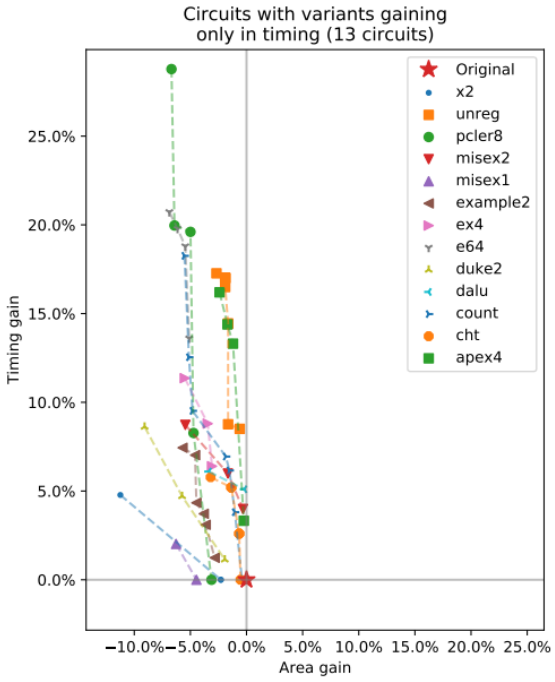
$$S_i \cap S_j = \emptyset, \ \forall i, j \in [0,3], i \neq j$$

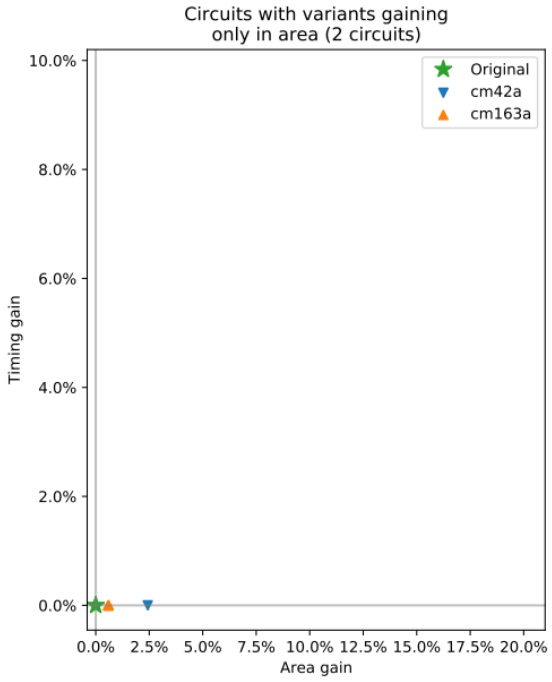
$$\bigcup_{i=0}^{3} S_i = S$$

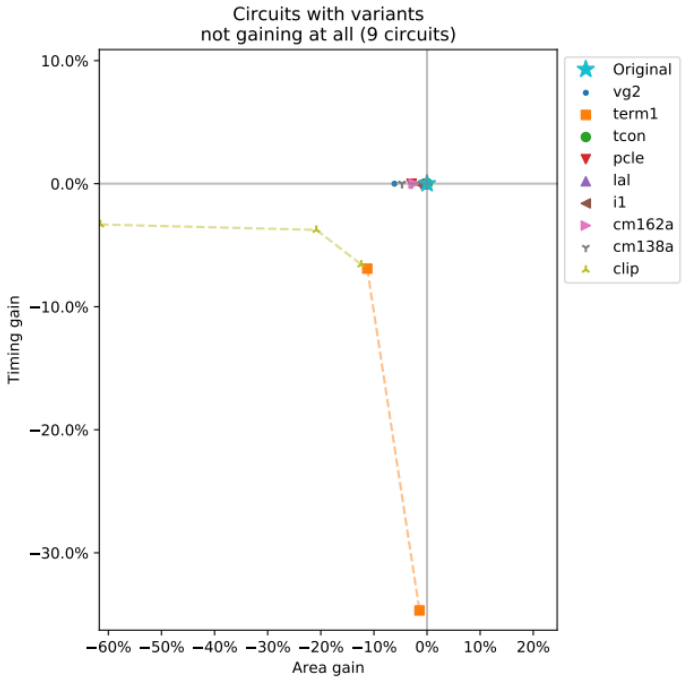
The voter is the same as the TMR!

For each output, we still have three replicas

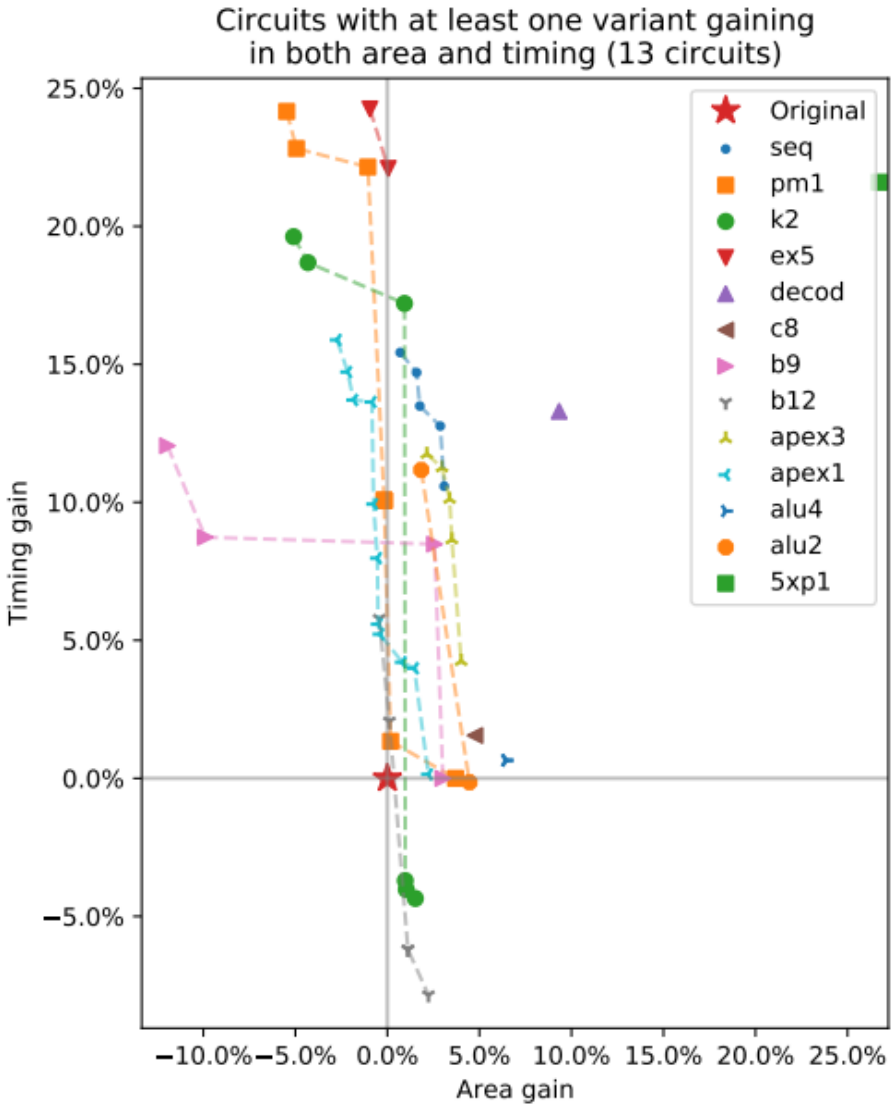
Results


Circuits with variants gaining only in timing (13 circuits)
Timing gain
Area gain
Original
x2
unreg
pcler8
misex2
misex1
example2
ex4
e64
duke2
dalu
count
cht
apex4


Circuits with variants gaining only in area (2 circuits)
Timing gain
Area gain
Original
cm42a
cm163a


Circuits with variants not gaining at all (9 circuits)
Timing gain
Area gain
Original
vg2
term1
tcon
pcle
lal
i1
cm162a
cm138a
clip

Results


Circuits with at least one variant gaining
in both area and timing (13 circuits)
Timing gain
Area gain
25.0%
20.0%
15.0%
10.0%
5.0%
0.0%
−5.0%
−10.0%
−5.0%
0.0%
5.0%
10.0%
15.0%
20.0%
25.0%
Original
seq
pm1
k2
ex5
decod
c8
b9
b12
apex3
apex1
alu4
alu2
5xp1

Conclusions

- Approximate computing is (still) a hot topic!
- We are investigating on the impact of hardware faults on the quality of applications outputs.
  - Reliability
- Need of automated tools and Frameworks
  - Complex applications: DSE feasible?
- Fault Classification is still key issue
  - Time consuming
  - Linked to application/workload

Acknowledgments

- Marcello Traiola, Etienne Dupuis, Ian O'Connor (INL, FR)
- Bastien Deveautour, Patrick Girard, Arnaud Virazel, David Novo (LIRMM, FR)
- Salvatore Barone, Mario Barbareschi (UNINA, IT)
- Annachiara Ruospo, Ernesto Sanchez (Polito, IT)
- Gennaro S. Rodrigues, Fernanda Lima Kastensmidt (UFRGS, BR)
- Jorge Echavarria, Ju¨rgen Teich (FAU, DE)