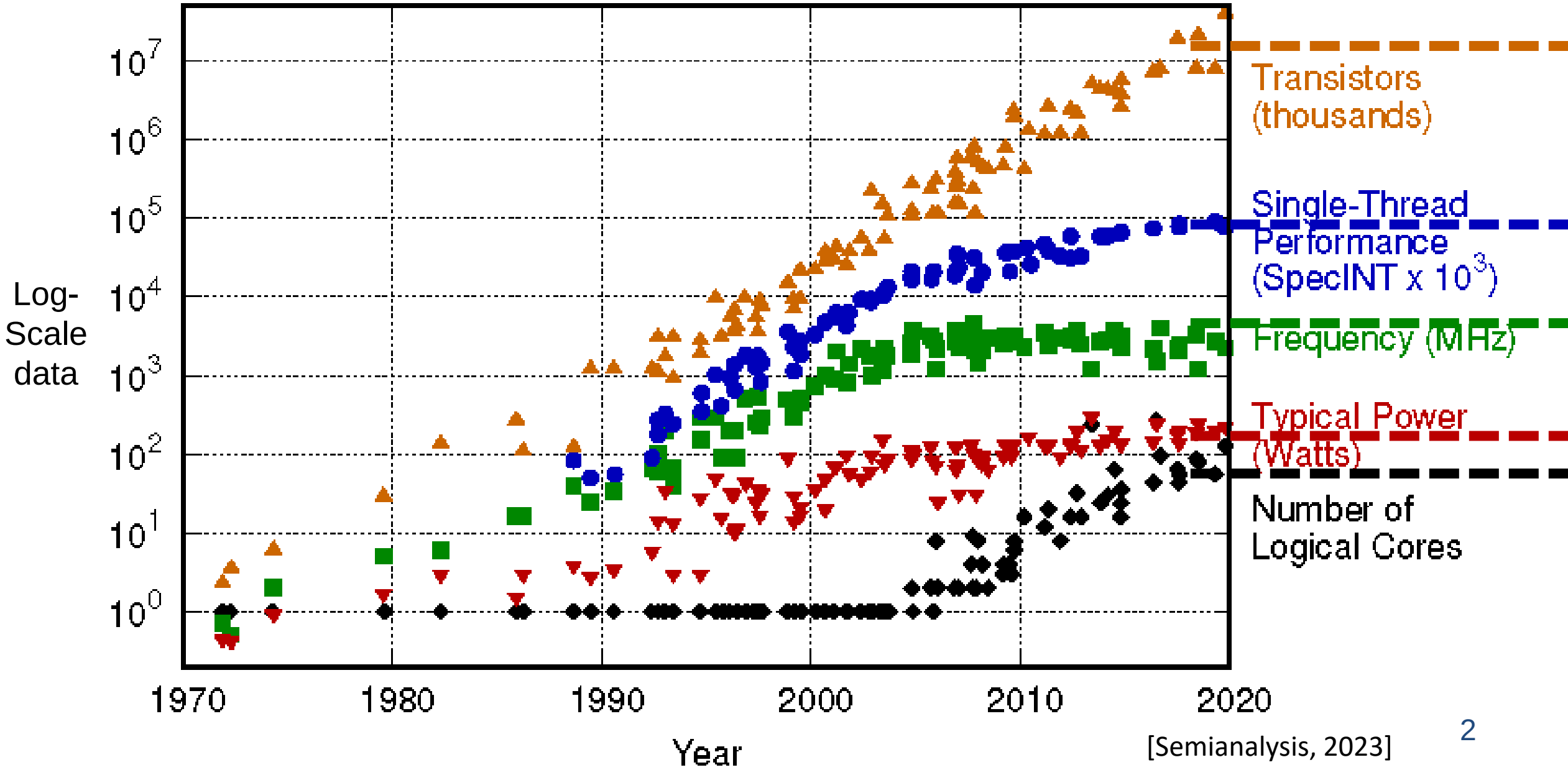
Logic Synthesis for Approximate Computing Circuits

Chang Meng, Postdoc

November 19, 2024

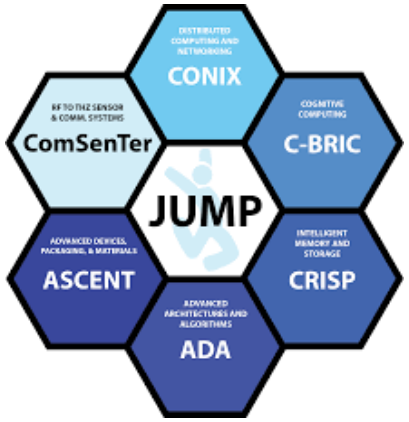
Moore's Law



[Semianalysis, 2023]

Exploration in Post-Moore Era

US DARPA Joint University Microelectronics Program (JUMP)



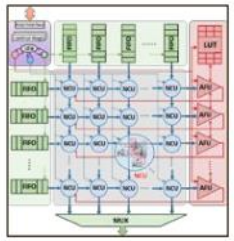
Approximate Computing

Architecture

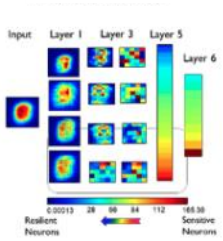
Device

Multicores/GPUs

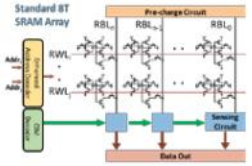
Accelerators



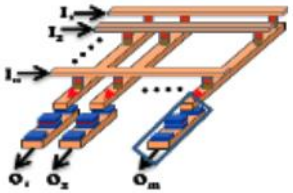
Computing Diagrams



In-memory computing



Post-CMOS Devices



Outline

- Background on Approximate Computing
- Logic Synthesis for Approximate Circuits
- Applications of Approximate Circuits
- Conclusion

Outline

- Background on Approximate Computing
- Logic Synthesis for Approximate Circuits
- Applications of Approximate Circuits
- Conclusion

Our World Is Approxmate

Our World Is Approximate

Error Tolerance E.g. 1: No Golden Answer

Please introduce sustainable computing in one sentence.

ChatGPT

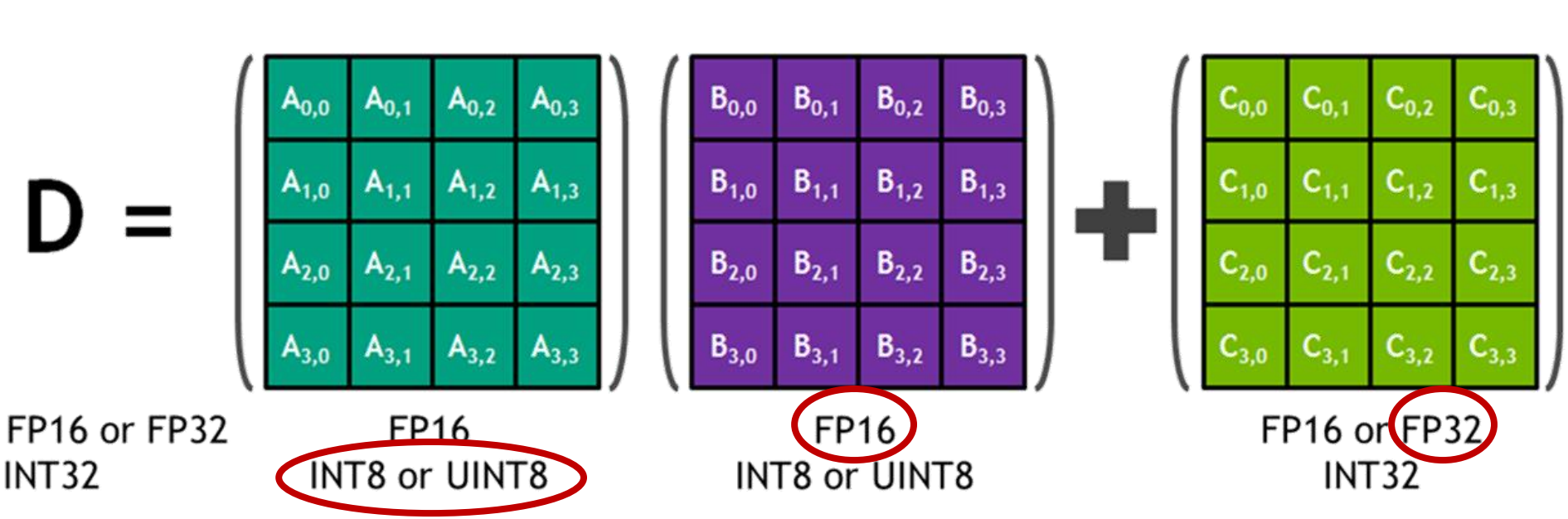
Sustainable computing focuses on designing and implementing computing systems that minimize energy consumption and environmental impact while maintaining performance and efficiency.

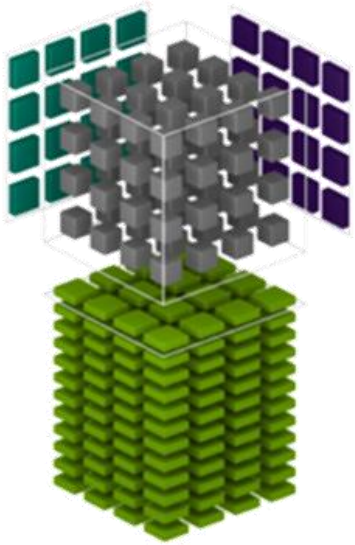
ANSWER 1

Sustainable computing involves designing, using, and disposing of computer systems and related technology in ways that minimize environmental impact, reduce energy consumption, and promote long-term ecological balance.

ANSWER 2

Error Tolerance 2: No Need Exact Numerical Values

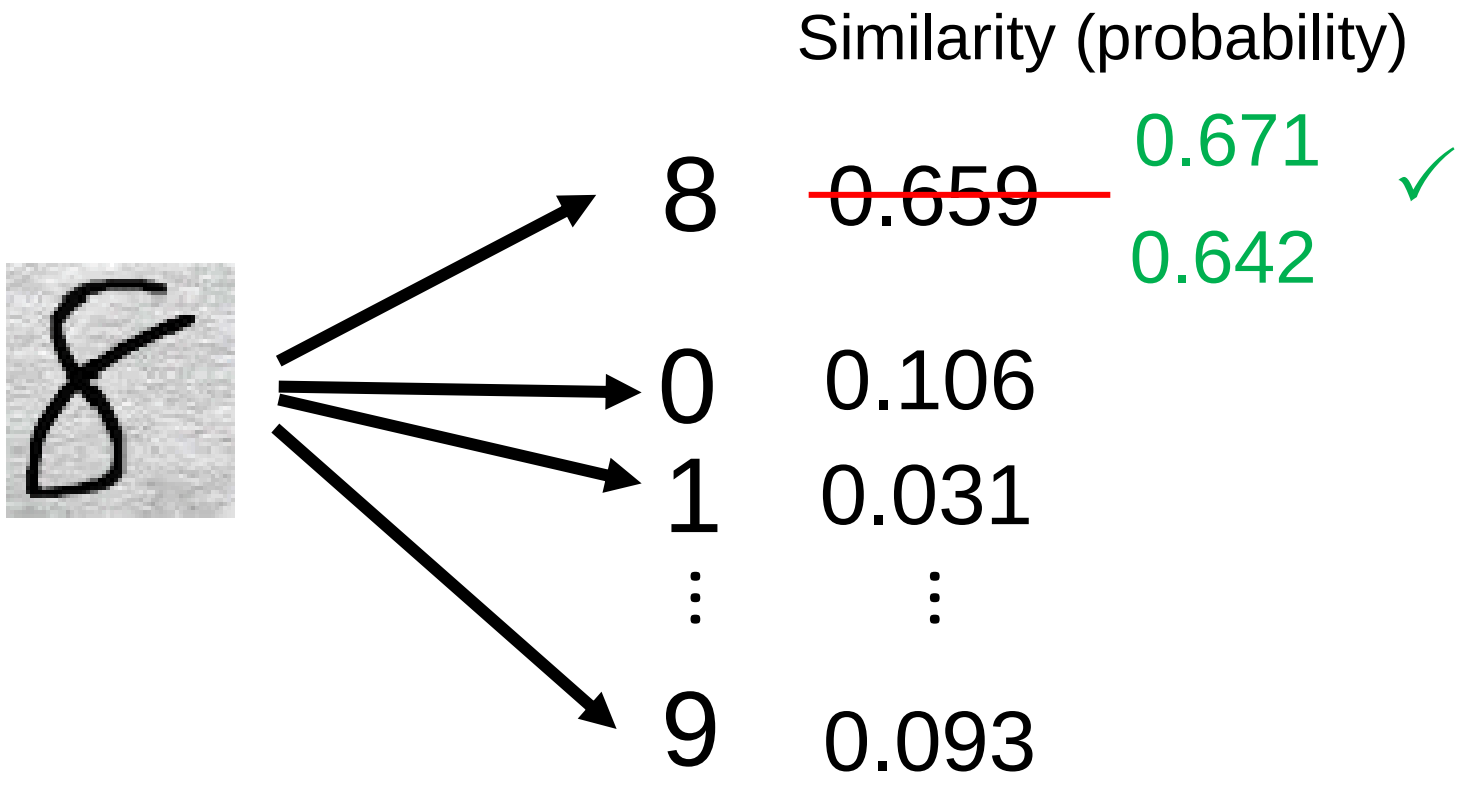




Mixed precision architecture in NVIDIA GPU

No Need Exact Numerical Values

- Deep neural network classifier



Error Tolerance 3: Limitation of Human Perception

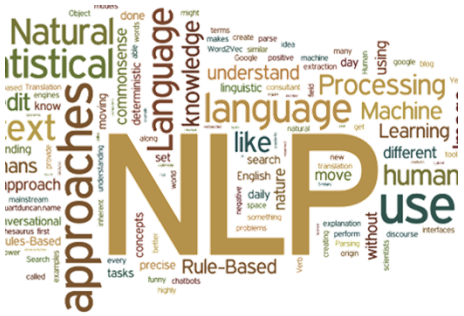
Picture without noise

Picture with noise

Noise doesn't affect recognition of the sportsman [Han+, 13]

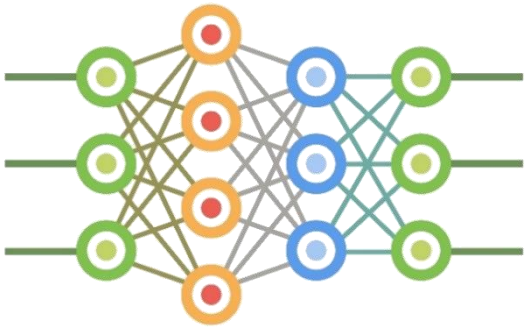
Approximate Computing

- Deliberately introduce **small errors** ➔ reduce **power consumption**
- **Error-tolerant** apps: system-level functions not affected by small errors



Language processing

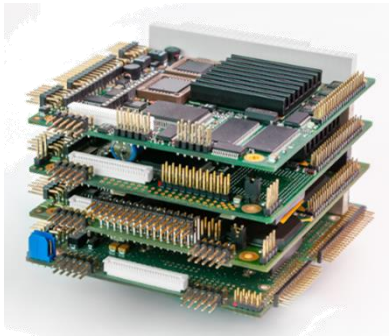

Image processing


Machine learning

- Reduce power for


AI accelerator


IoT system


Embedded device

Approximate Computing

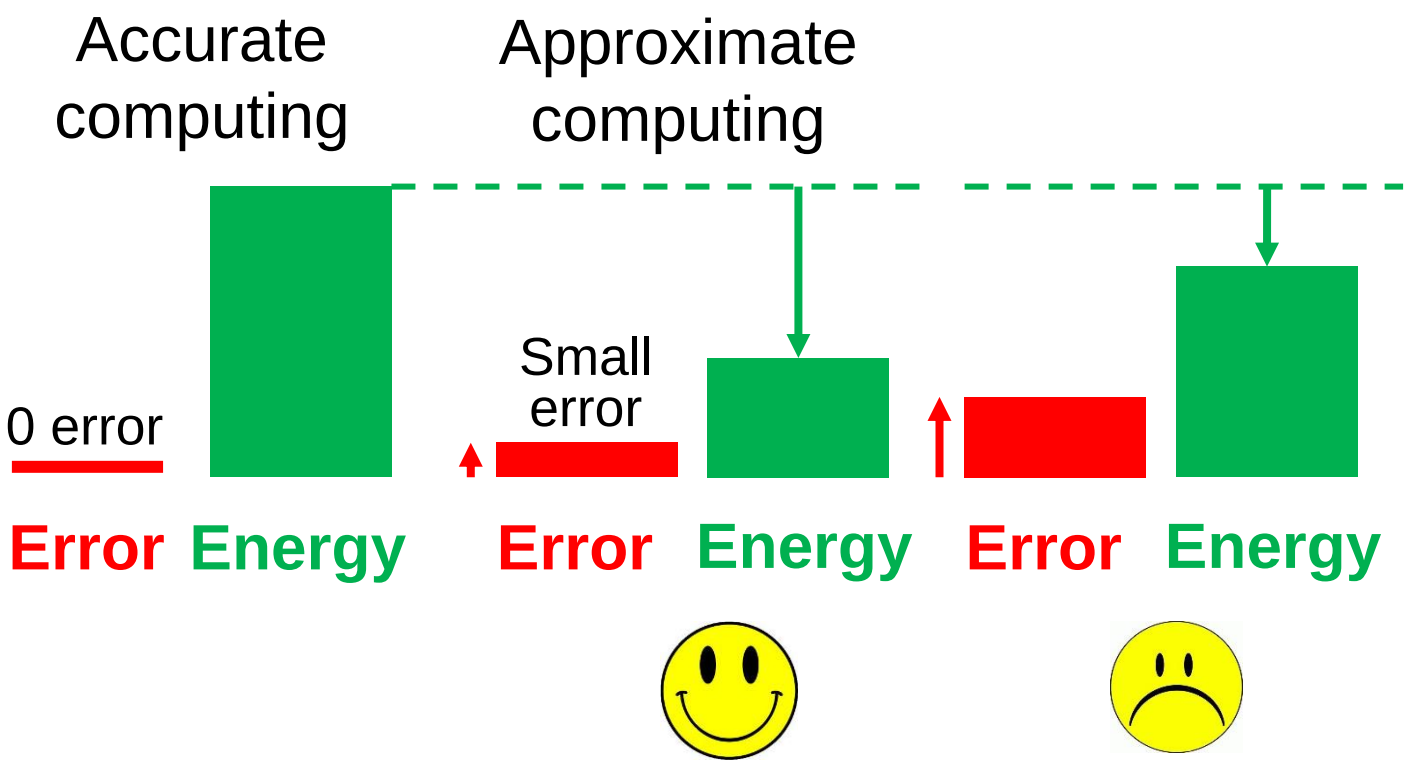
- Sacrifice accuracy for performance and energy improvement

Algorithm

Compiler

Architecture

Circuit



Approximate Circuit

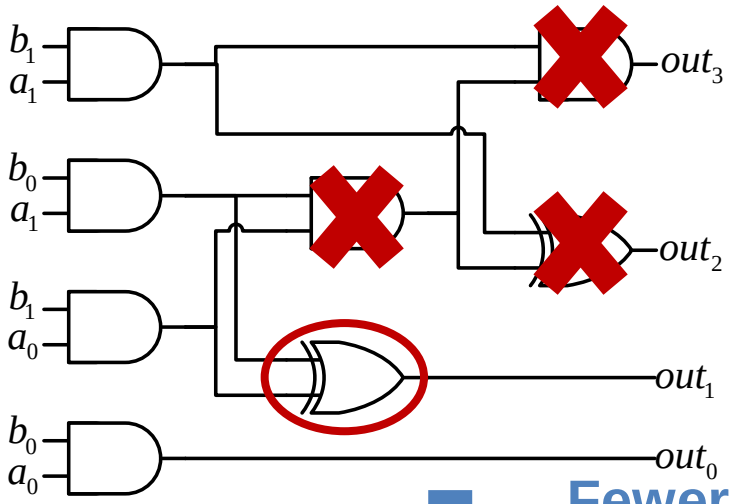
- Trade off **accuracy** ➔ reduce **power, delay & area**
- Karnaugh map for a 2x2 multiplier

a_1a_0 \ b_1b_0	00	01	11	10
00	0000	0000	0000	0000
01	0000	0001	0011	0010
11	0000	0011	1001	0110
10	0000	0010	0110	0100

0111

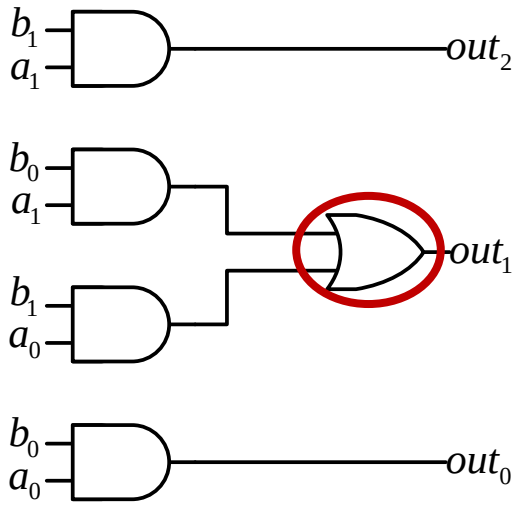
Error rate =1/16

[Kulkarni+, 11]



Exact circuit

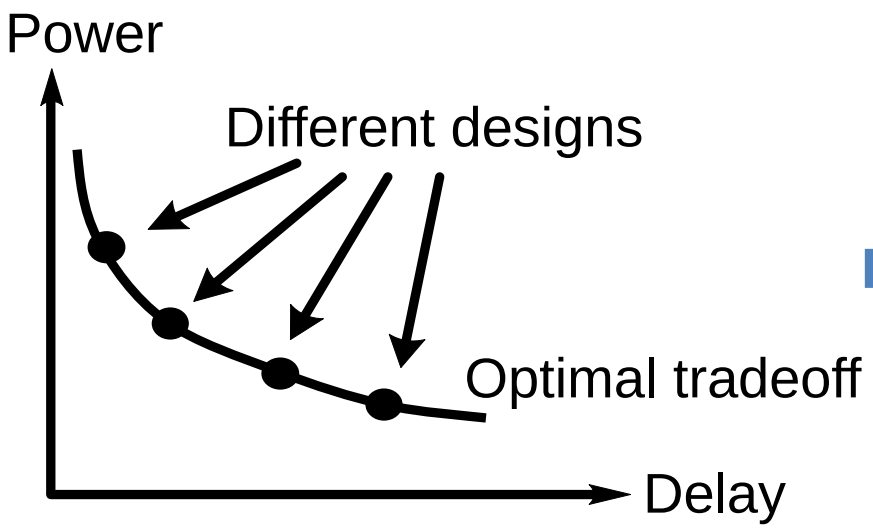
Fewer & simpler gates ➔ smaller area & lower power



Approximate circuit

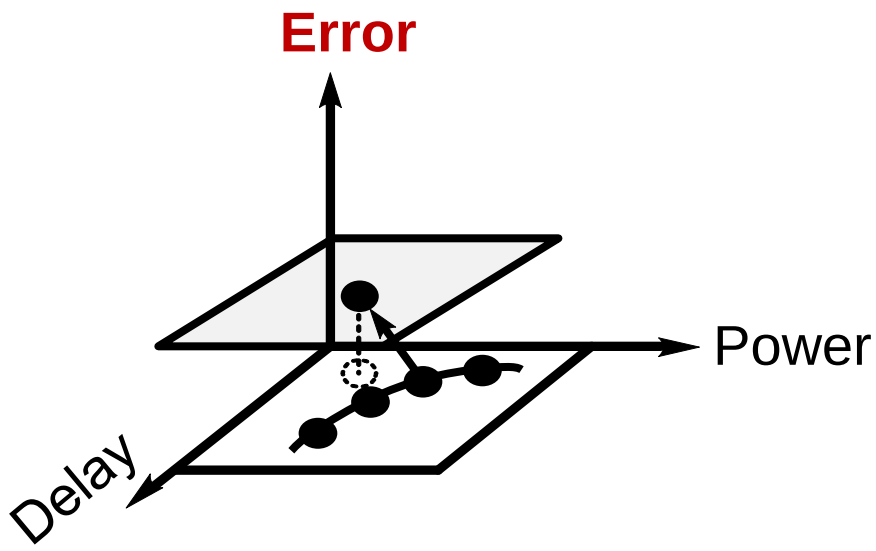
New Design Space with Approx. Computing!

- Opportunities: Large design space
- Challenges: how to find **low-power** designs **efficiently?**



2D design space

Traditional computing



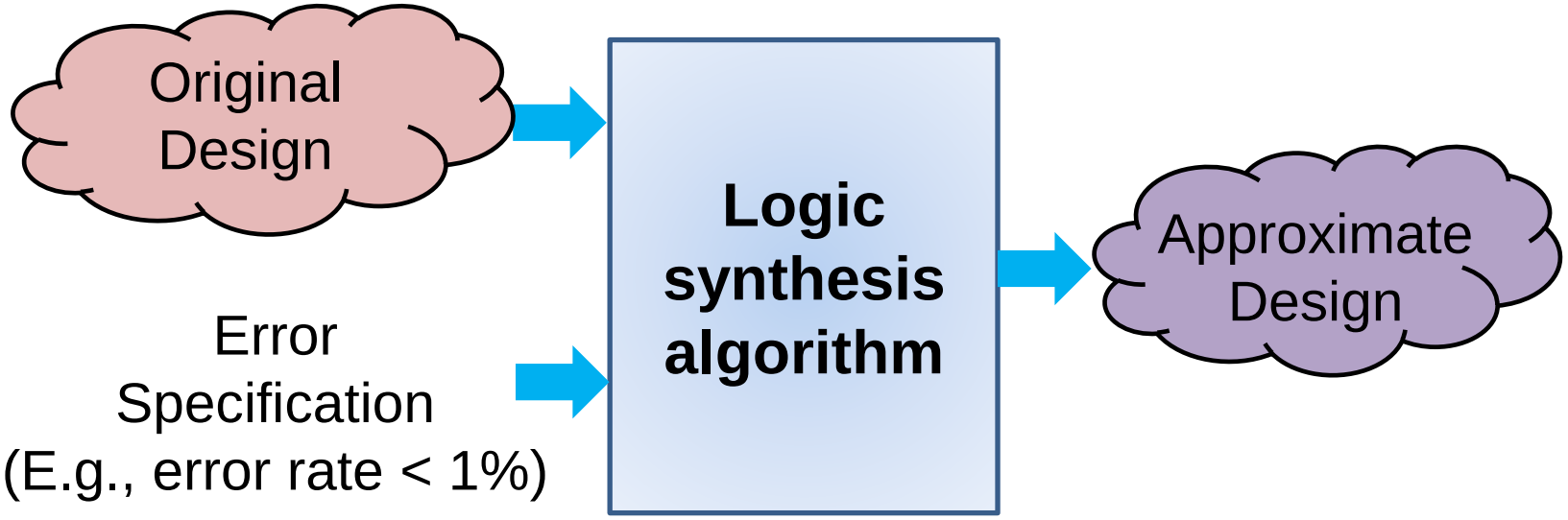
3D design space

Approximate computing

Approximate Logic Synthesis

A1A0 \ B1B0	00	01	11	10
00	000	000	000	000
01	000	001	011	010
11	000	011	1001	110
10	000	010	110	100

What's the optimal way to introduce error?



Error Metrics

Error rate (ER): $\Pr(\vec{f}(\vec{X}) \neq \vec{f}'(\vec{X}))$

- $\vec{f}$ is circuit function, $\vec{X}$ is circuit input

- **Error distance (ED)**: Applicable to arithmetic circuits
  - **Maximal ED (MaxED)**: $\max_{\vec{X}} |\vec{f}(\vec{X}) - \vec{f}'(\vec{X})|$
  - **Mean ED (MED)**: $E[|\vec{f}(\vec{X}) - \vec{f}'(\vec{X})|]$

A1A0 \ B1B0	00	01	11	10
00	0000	0000	0000	0000
01	0000	0001	0011	0010
11	0000	0011	1001	1110
10	1000	1010	1110	1100

0011 ER = 2/16
MaxED = 2
0111 MED = 3/16

Error Metrics

- **Normalized mean error distance (NMED)**

$$\frac{E\left[\left|\vec{f}(\vec{X}) - \vec{f}'(\vec{X})\right|\right]}{2^O - 1}$$

  - O is the number of output bits

- **Mean relative error distance (MRED)**

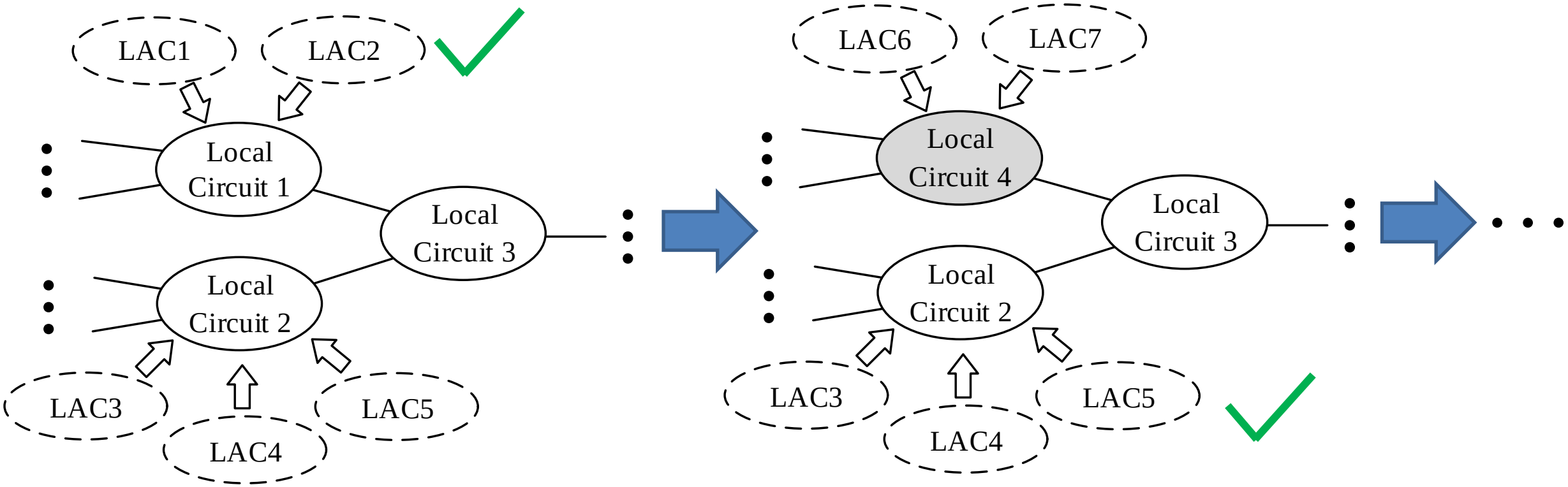
$$E\left[\frac{\left|\vec{f}(\vec{X}) - \vec{f}'(\vec{X})\right|}{\vec{f}(\vec{X})}\right]$$

Outline

- Background on Approximate Computing
- Logic Synthesis for Approximate Circuits
- Applications of Approximate Circuits
- Conclusion

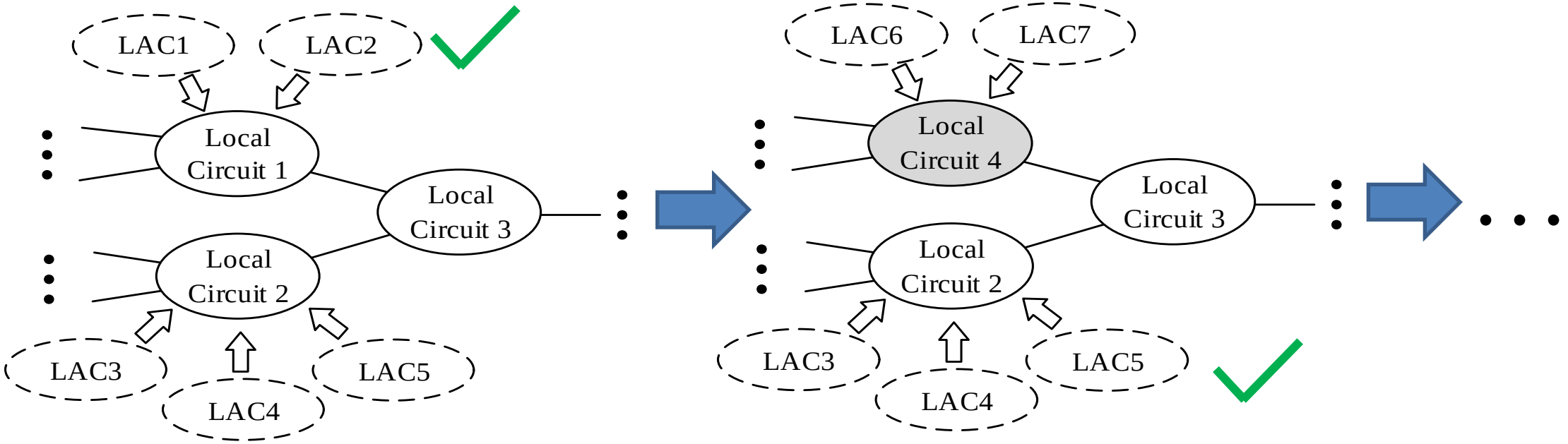
Iterative Framework

- Local approximate changes (LACs) + Iterative improvement



Iterative Framework: Key Problems

- Which types of LACs?
- For all the LACs, which of them should be considered?
- From those considered, which of them should be finally selected?



Classification

- Based on which LACs to consider and which to select
  - Consider a deterministic subset + select one
  - Consider a deterministic subset + select multiple
  - Consider a random subset + select one

Outline

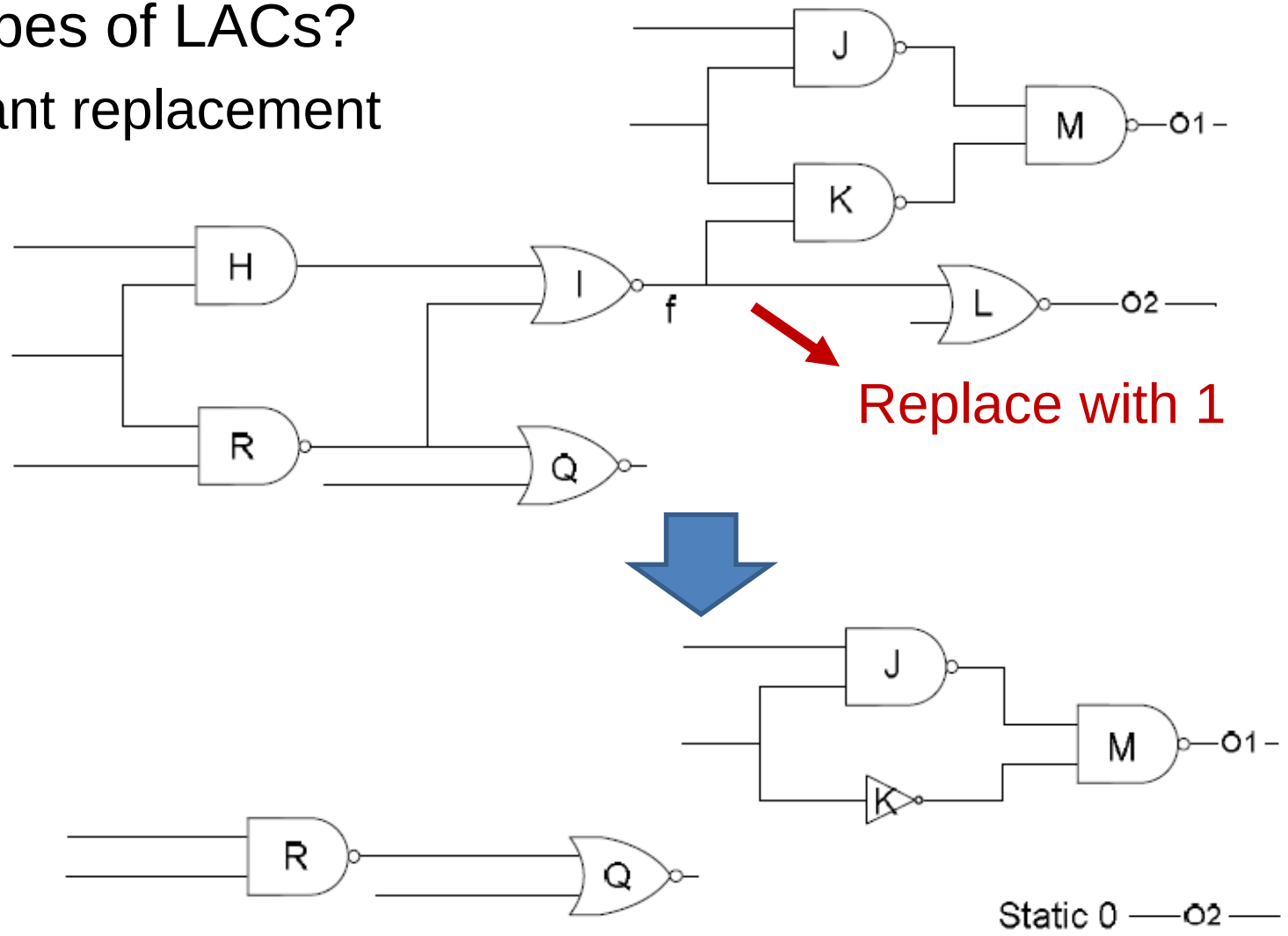
- Based on which LACs to consider and which to select
  - Consider a deterministic subset + select one
    - Constant replacement-based LAC
    - Signal substitution-based LAC
    - Resubstitution-based LAC
  - Consider a deterministic subset + select multiple
  - Consider a random subset + select one

Outline

- Based on which LACs to consider and which to select
  - Consider a deterministic subset + select one
    - Constant replacement-based LAC
    - Signal substitution-based LAC
    - Resubstitution-based LAC
  - Consider a deterministic subset + select multiple
  - Consider a random subset + select one

Constant Replacement

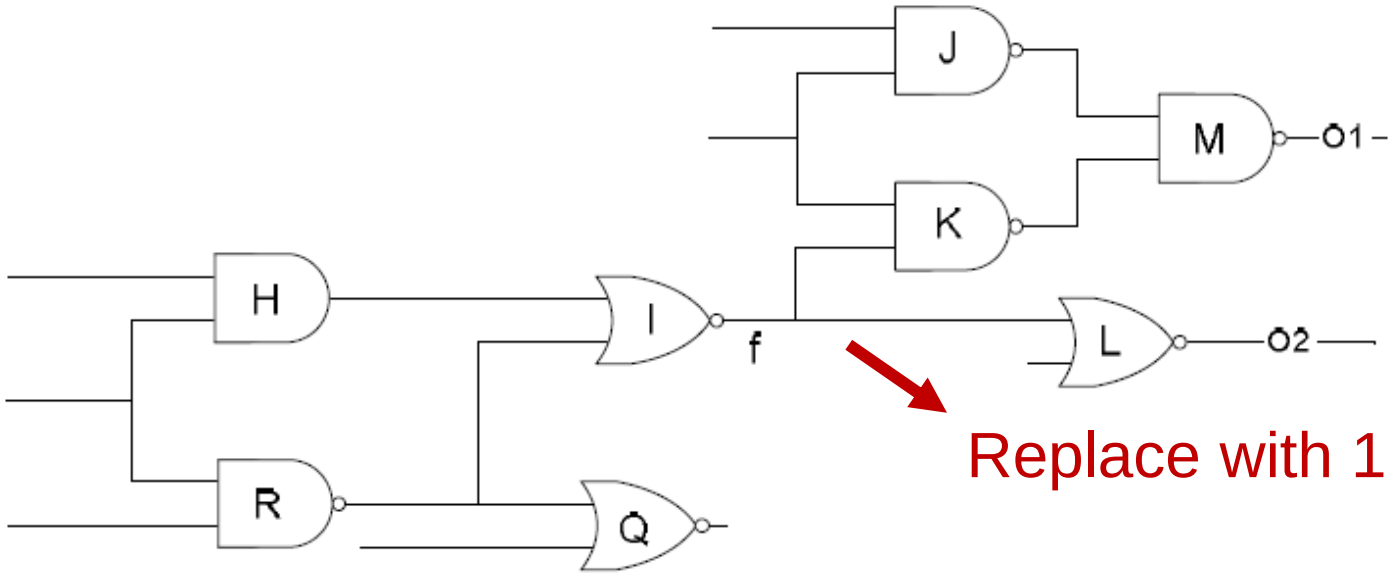
- Which types of LACs?
  - Constant replacement



Shin & Gupta, "A new circuit simplification method for error tolerant applications," *DATE*'11

Constant Replacement

- For all the LACs, which of them should be considered?
  - Consider all the wires, and for each wire, consider both choices (0/1)
- From those considered, which of them should be finally selected?
  - Select the one with the largest $AreaReduction/(ER * MaxED)$



Outline

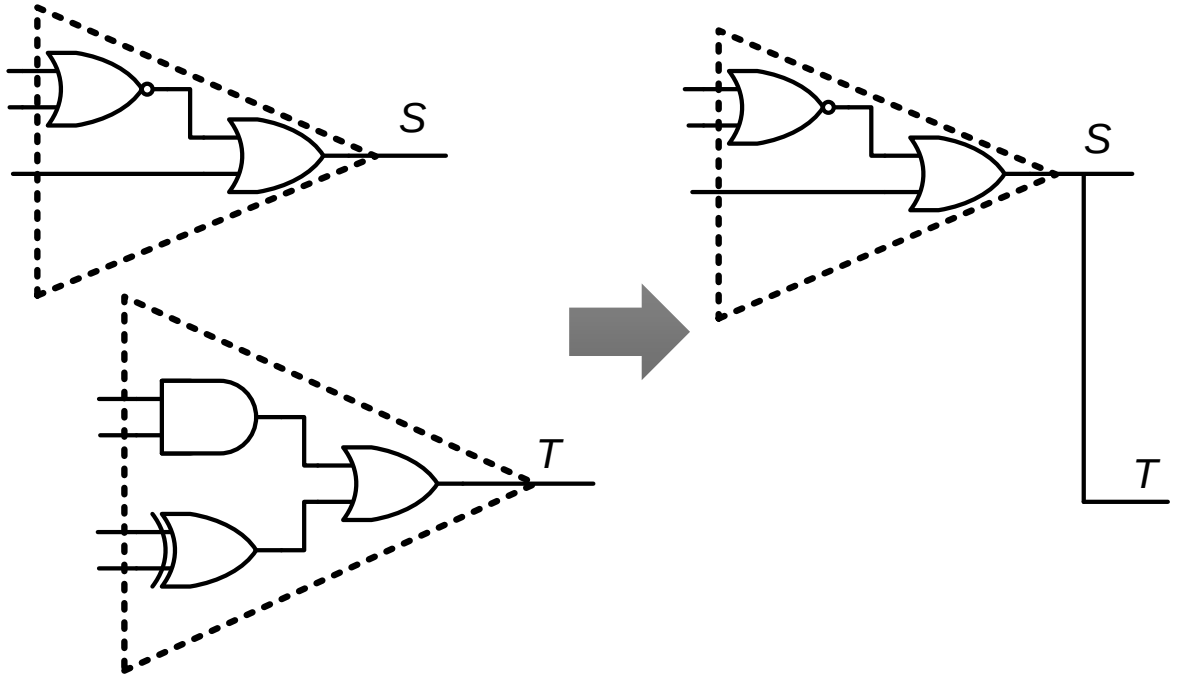
- Based on which LACs to consider and which to select
  - Consider a deterministic subset + select one
    - Constant replacement-based LAC
    - Signal substitution-based LAC
    - Resubstitution-based LAC
  - Consider a deterministic subset + select multiple
  - Consider a random subset + select one

SASIMI: Signal Substitution

- Which types of LACs?
  - Signal substitution
  - Basic idea: find two signals not identical but very close

S: 0010001101

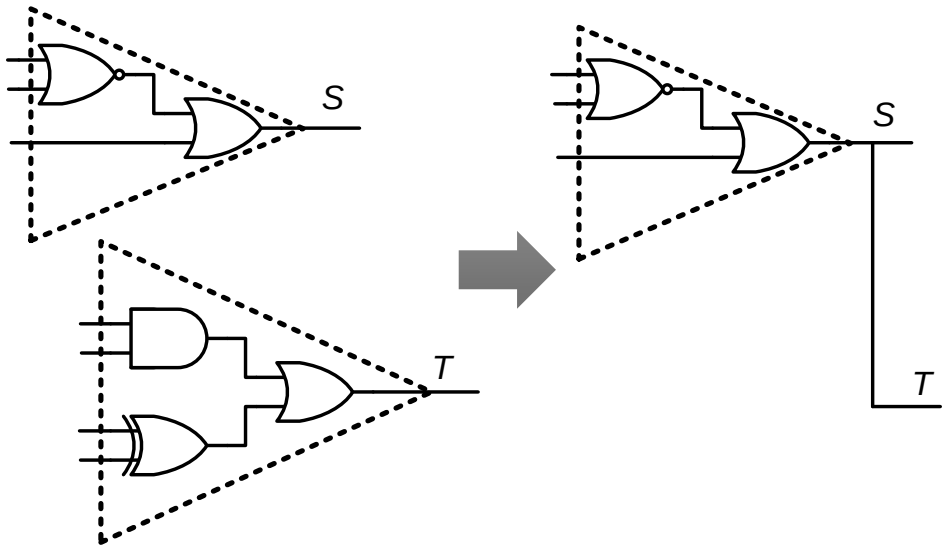
T: 0011001101



Venkataramani *et al.*, "Substitute-and-Simplify: A Unified Design Paradigm for Approximate and Quality Configurable Circuits," *DATE*'13

SASIMI: Signal Substitution

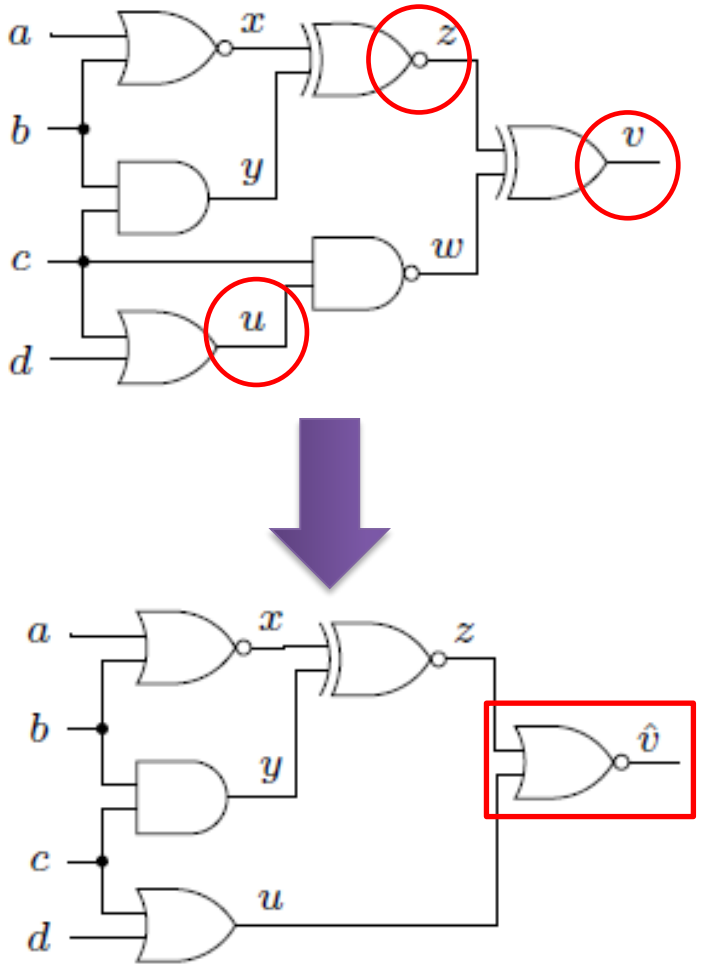
- For all the LACs, which of them should be considered?
  - Consider all the signal pairs $(SubSignal, TargetSignal)$ and $(\overline{SubSignal}, TargetSignal)$
- From those considered, which of them should be finally selected?
  - Select the one with the largest $(\alpha \cdot AreaReduction + (1-\alpha) DelayReduction)/(P_{diff}(1-P_{diff}))$
  - $P_{diff} = \Pr(SubSignal \neq TargetSignal)$



Outline

- Based on which LACs to consider and which to select
  - Consider a deterministic subset + select one
    - Constant replacement-based LAC
    - Signal substitution-based LAC
    - Resubstitution-based LAC
  - Consider a deterministic subset + select multiple
  - Consider a random subset + select one

ALSRAC: Resubstitution



Can we resubstitute v as a function of u and z? NO!

- $abcd = 0001$ ➔ $uz = 10$, $v = 0$
- $abcd = 0010$ ➔ $uz = 10$, $v = 1$

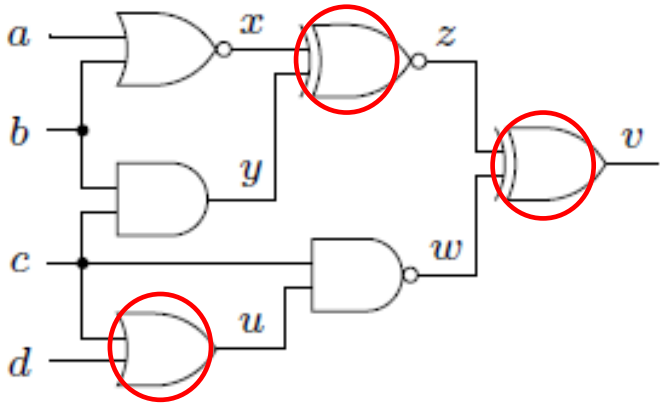
But, we can approximately resubstitute v as a function of z and u!

Meng, Qian & Mishchenko, "ALSRAC: approximate logic synthesis by resubstitution with approximate care set," DAC'20
Code: https://github.com/SJTU-ECTL/ALSRAC

ALSRAC Methodology

- Step 1: Use random logic simulation to generate approximate care set

abcd	x	y	u	z	w	v
0000	1	0	0	0	1	1
0001	1	0	1	0	1	1
0010	1	0	1	0	0	0
0011	1	0	1	0	0	0
0100	0	0	0	1	1	0
0101	0	0	1	1	1	0
0110	0	1	1	0	0	0
0111	0	1	1	0	0	0
1000	0	0	0	1	1	0
1001	0	0	1	1	1	0
1010	0	0	1	1	0	1
1011	0	0	1	1	0	1
1100	0	0	0	1	1	0
1101	0	0	1	1	1	0
1110	0	1	1	0	0	0
1111	0	1	1	0	0	0

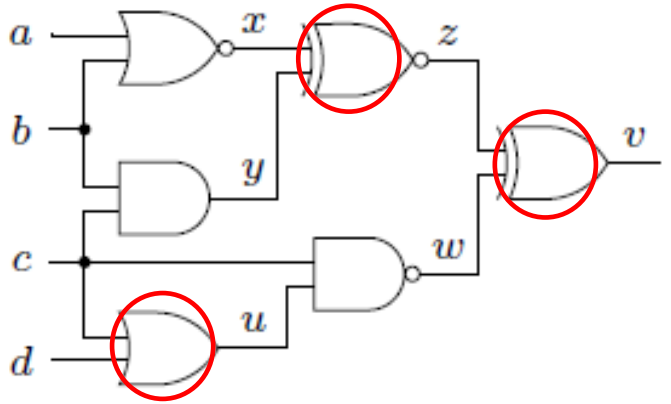


- Randomly select 5 input patterns (**in red**) and simulate the circuit
- Only **care** the patterns appeared in simulation
- Other patterns are treated as **don’t-cares**

ALSRAC Methodology

- Step 2: Build truth table and check feasibility

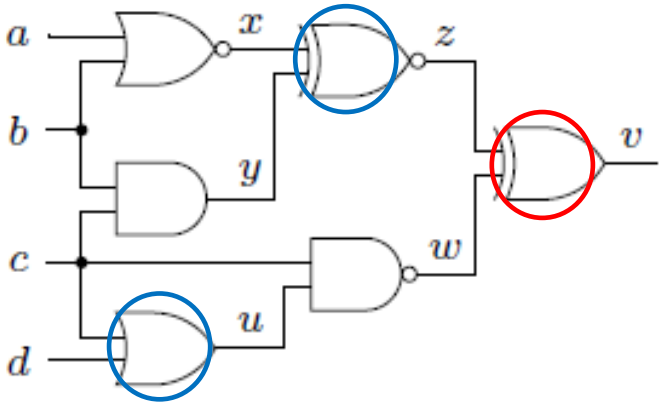
abcd	x	y	u	z	w	v
0000	1	0	0	0	1	1
0001	1	0	1	0	1	1
0010	1	0	1	0	0	0
0011	1	0	1	0	0	0
0100	0	0	0	1	1	0
0101	0	0	1	1	1	0
0110	0	1	1	0	0	0
0111	0	1	1	0	0	0
1000	0	0	0	1	1	0
1001	0	0	1	1	1	0
1010	0	0	1	1	0	1
1011	0	0	1	1	0	1
1100	0	0	0	1	1	0
1101	0	0	1	1	1	0
1110	0	1	1	0	0	0
1111	0	1	1	0	0	0



uz	00	01	10	11
$\hat{v}$	1	0	0	–

- No conflicts in the truth table (each uz pattern corresponds to only one value of $\hat{v}$)
- Truth table of approximate function: $\hat{v} = \bar{u} + \bar{z}$

Which LACs to Consider and Which to Select?



- Which LACs to consider?
  - u and z are called divisors of v
  - LACs: $(v, DivisorSet)$, e.g., $(v, \{u, z\})$
  - For each node v, consider the following divisors
    - Remove a fanin of v, i.e., $\{z\}, \{w\}$
    - Replace a fanin of v by another node in v's TFI cone, i.e.,
      $\{z,a\},\{z,b\},\{z,c\},\{z,d\},\{z,x\},\{z,y\},\{z,u\},$
      $\{w,a\},\{w,b\},\{w,c\},\{w,d\},\{w,x\},\{w,y\},\{w,u\}$
- Which LAC(s) to select?
  - Select the one with the smallest error

Outline

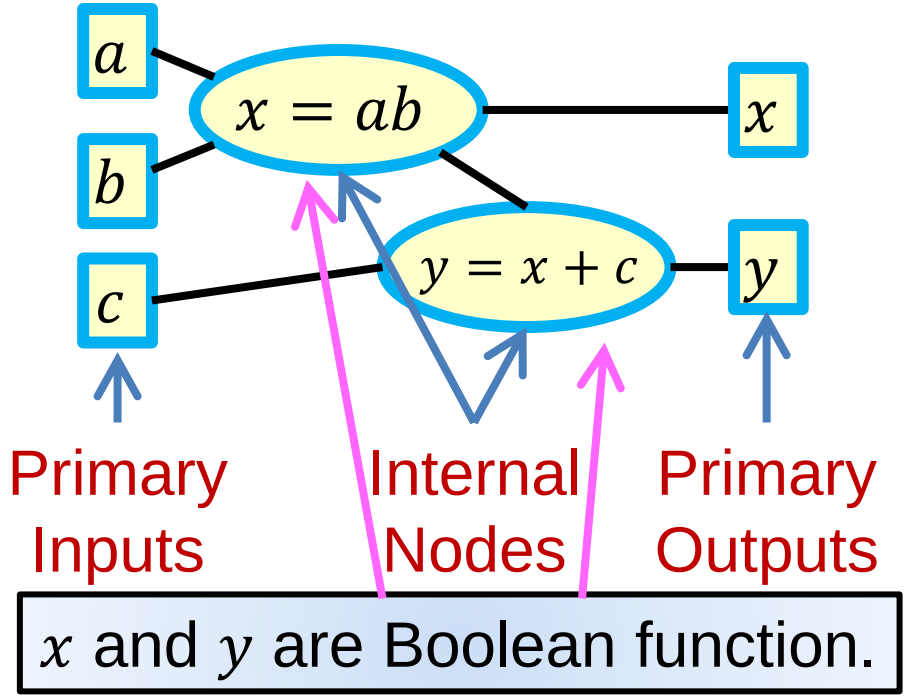
- Based on which LACs to consider and which to select
  - Consider a deterministic subset + select one
  - Consider a deterministic subset + select multiple
    - Approximate node simplification
    - Delay-driven ALS
  - Consider a random subset + select one

Outline

- Based on which LACs to consider and which to select
  - Consider a deterministic subset + select one
  - Consider a deterministic subset + select multiple
    - Approximate node simplification
    - Delay-driven ALS
  - Consider a random subset + select one

Approximate Node Simplification

- Work on Boolean logic network model
  - A direct acyclic graph. Each node is a Boolean function
  - Boolean function could be in either sum-of-product (SOP) form or factored form
  - What to optimize? Total literal count

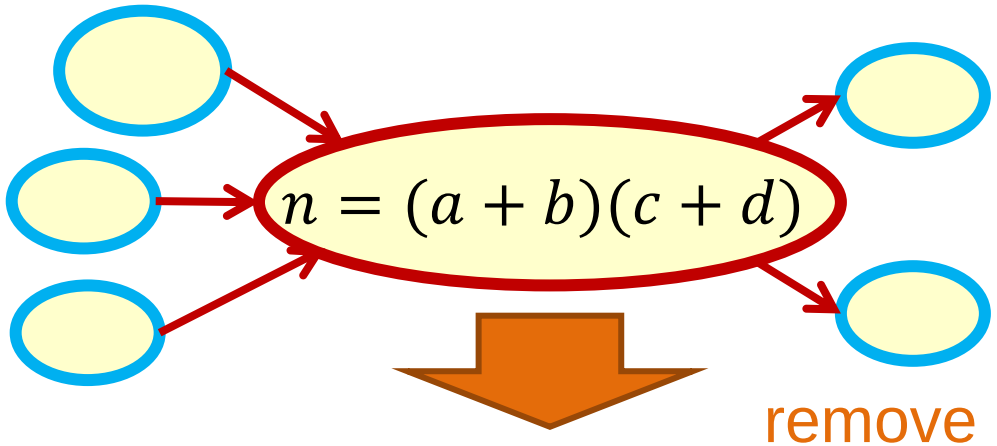


#Literals = 5

Wu & Qian, "An efficient method for multi-level approximate logic synthesis under error rate constraint," *DAC'16*

Approximate Node Simplification

- Work on factored-form expression of a node
  - Simplify it by removing some literals
  - An approximation; can cause error
  - Call the result approximate simplified expression (ASE)

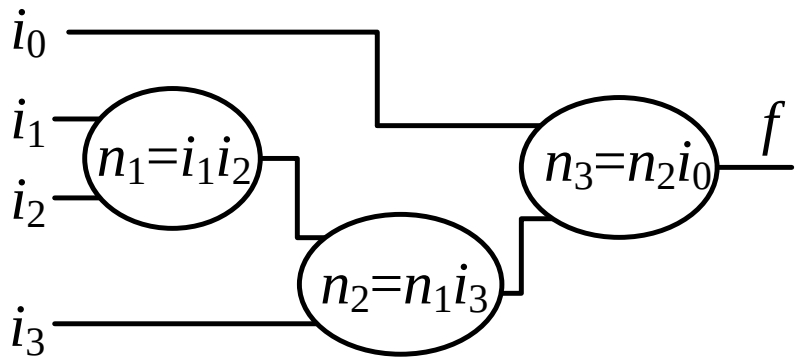


- Each node has multiple ASEs
- Consider all the nodes and all their ASEs

remove 1 literal

$n = a(c + d)$ $n = b(c + d)$ $n = (a + b)c$ $n = (a + b)d$

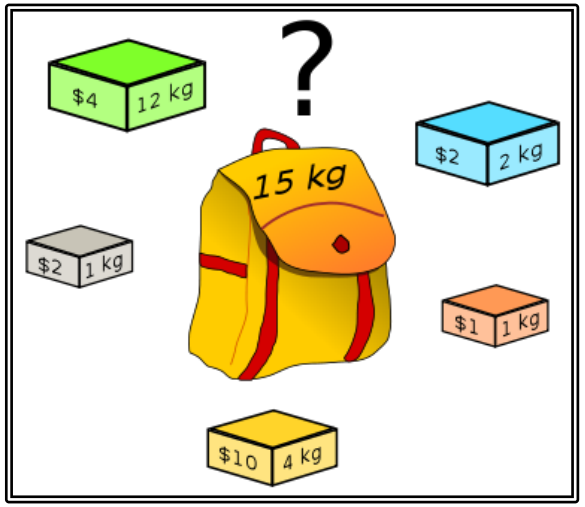
Selection Problem: Multiple Selection



- Questions:
  1. Which set of nodes should we choose to make change?
  2. For these chosen nodes, which of their ASEs should we pick?
- Proposed solution: model this as a 0/1 multi-state knapsack problem

Mapping to 0/1 Multi-state Knapsack Problem

- A node → an candidate item
- An ASE of a node → a state of a item
- Error rate of an ASE → weight
- Number of saved literals → value
- Error rate margin → capacity of knapsack



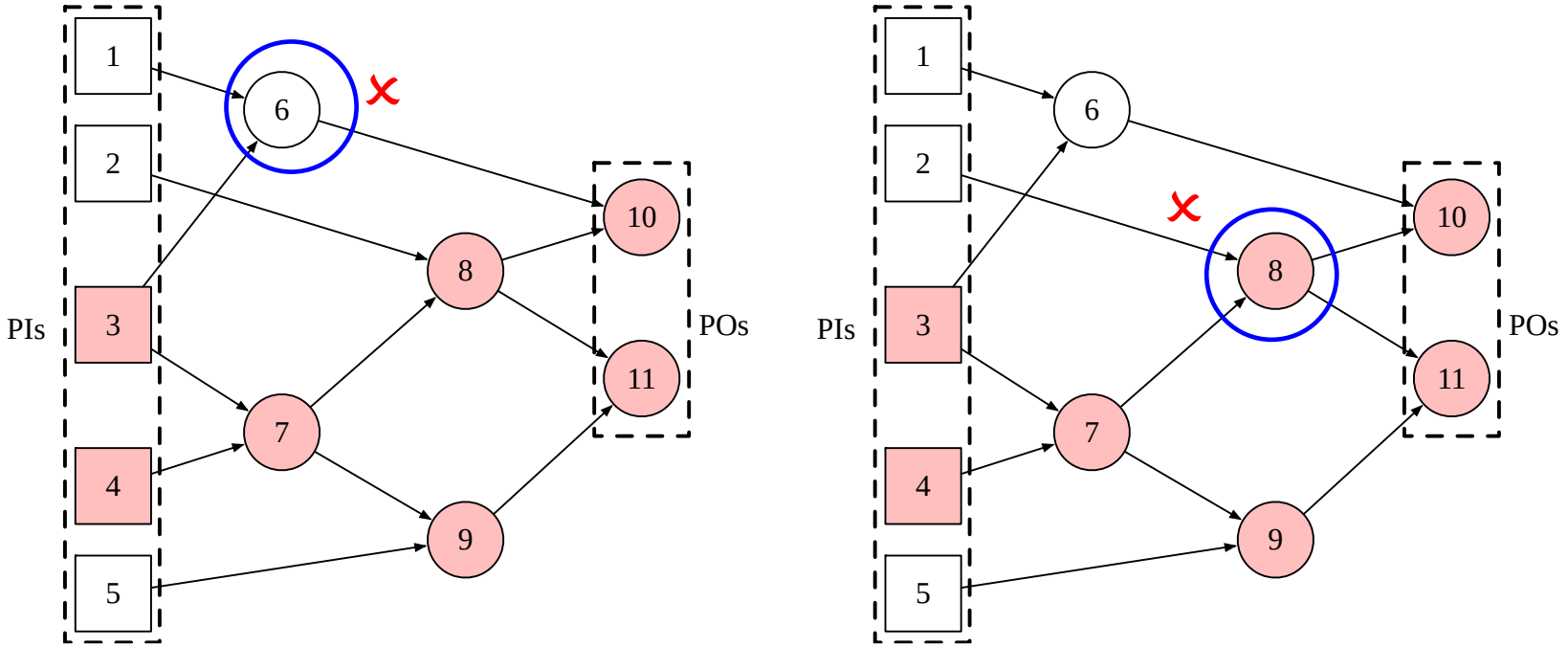
- Multi-state knapsack problem can be solved by extending the classical dynamic programming solution to basic 0/1 knapsack problem
- Flow: after each round of multi-selection, evaluate actual error rate, update error rate margin and do another round until margin used up

Outline

- Based on which LACs to consider and which to select
  - Consider a deterministic subset + select one
  - Consider a deterministic subset + select multiple
    - Approximate node simplification
    - Delay-driven ALS
  - Consider a random subset + select one

Delay Optimization

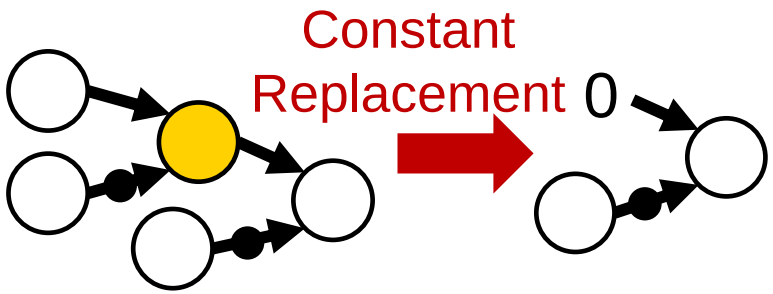
- Area-Driven ALS Is Not Good in Reducing Delay
  - Applying local approximate changes (LACs) on non-critical gates
  - Applying LAC on a single critical gate is not effective, since there exist multiple critical paths

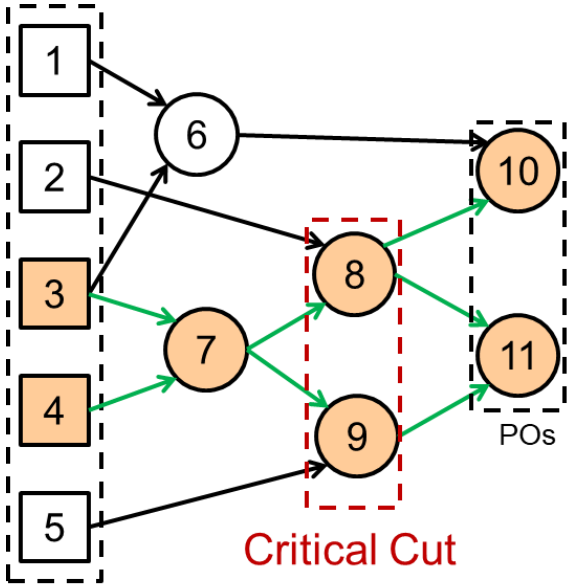


Zhou *et al.*, "DALS: delay-driven approximate logic synthesis," *ICCAD'18*

Delay Optimization: Basic Idea

- Apply depth-reduction local approximate changes (LACs) to **critical cut** of the critical graph





- Main flow

Obtain critical graph

Find a critical cut and a set of LACs with minimum error impact

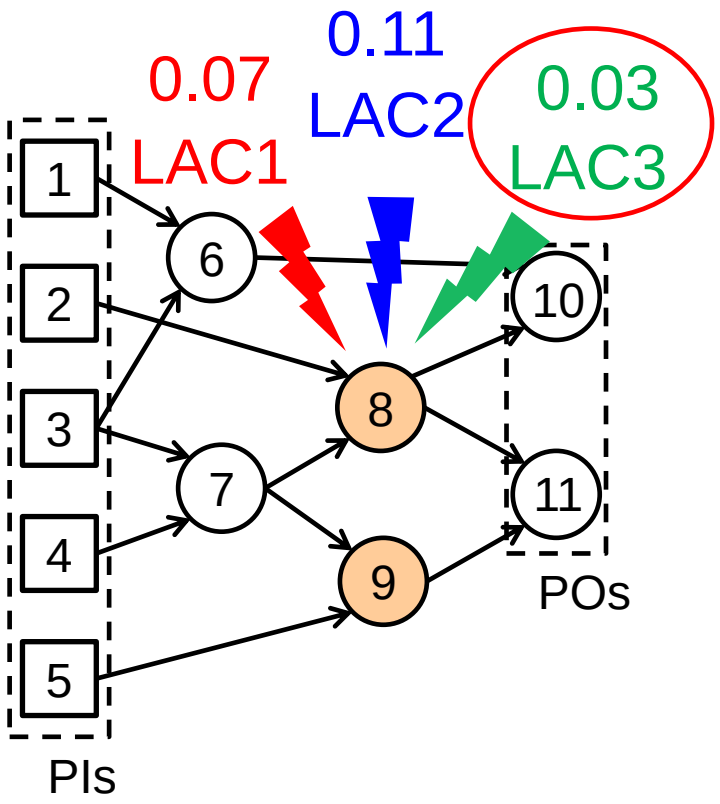
Until error bound is reached

Apply the best choice (cut + LACs) to get new approximate circuit

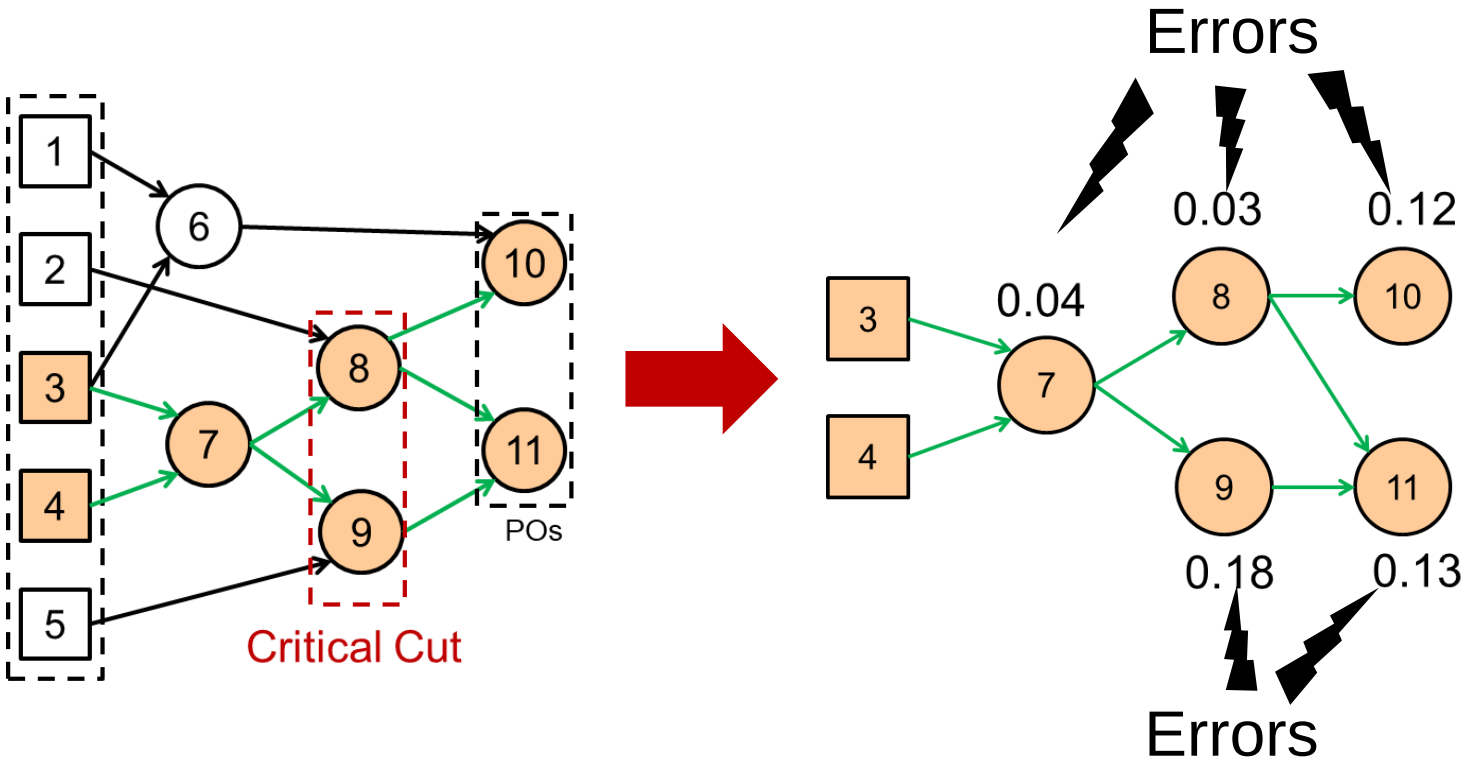
?

Delay Optimization: Finding Optimized LACs

- Focus on the min-error LAC for each node



- Which cut to select?
  - Solve a network flow problem



Outline

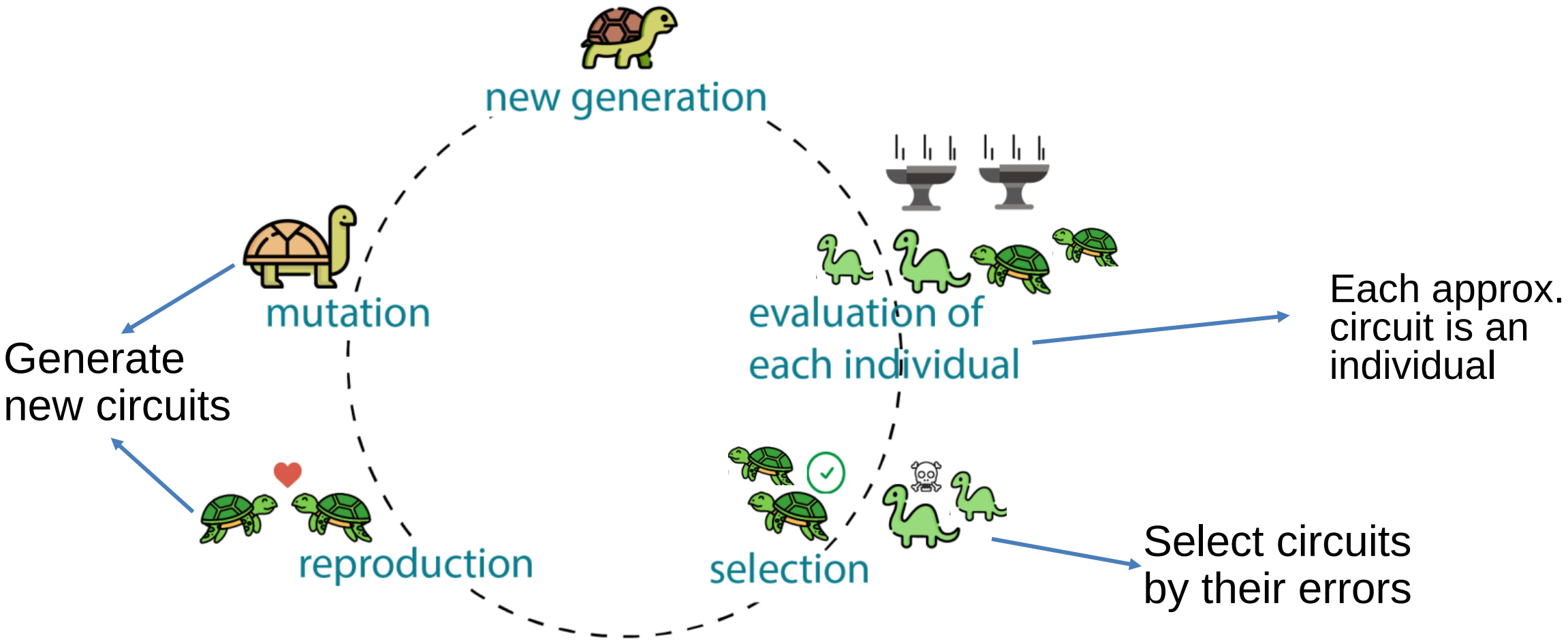
- Based on which LACs to consider and which to select
  - Consider a deterministic subset + select one
  - Consider a deterministic subset + select multiple
  - Consider a random subset + select one
    - Evolutionary algorithm-based method
    - Random choice and probabilistic acceptance-based method

Outline

- Based on which LACs to consider and which to select
  - Consider a deterministic subset + select one
  - Consider a deterministic subset + select multiple
  - Consider a random subset + select one
    - Evolutionary algorithm-based method
    - Random choice and probabilistic acceptance-based method

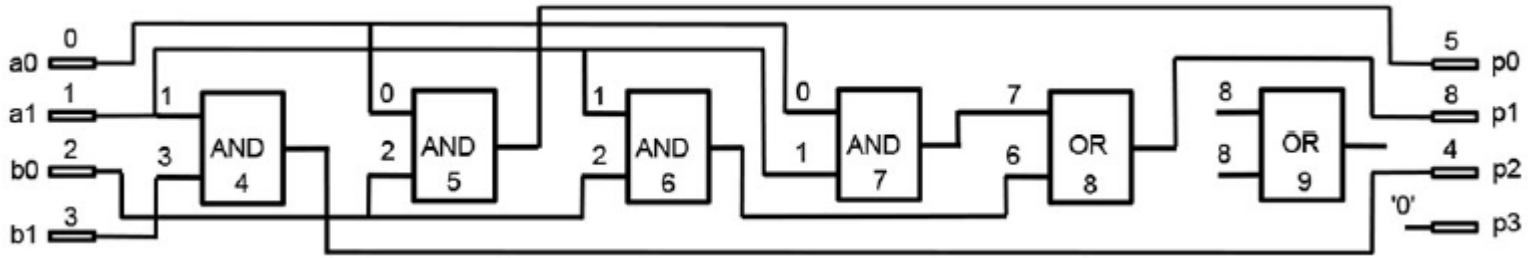
Evolutionary Algorithm-based Method

- Genetic algorithm: mimic the process of natural selection



Evolutionary Algorithm-based Method

- Represent circuit (individual) as a chromosome
  - Assume circuit has n_i primary inputs and n_o primary outputs
  - A candidate circuit is modeled as a gate array of n_c internal nodes
  - Primary inputs and nodes are labeled as $0,1,\ldots,n_i-1,n_i,n_i+1,\ldots,n_i+n_c-1$
  - Encode each node as $(fanin1ID, fanin2ID, gateTypeID)$
  - Last part contains n_o values specifying the outputs



AND: 0
OR: 1

Chromosome: (1, 3, 0); (0, 2, 0); (1, 2, 0); (0, 1, 0); (7, 6, 1); (8, 8, 1); [5, 8, 4, '0']

Vasicek & Sekanina, "Evolutionary approach to approximate digital circuits design," *Trans. Evo. Compt.'15*

Evolutionary Algorithm-based Method

- Step 1: create initial population of size $(1 + \lambda)$
- Step 2: calculate MED for each candidate circuit
- Step 3: select the candidate circuit with the lowest MED as the parent
- Step 4: applying point mutation to generate λ offspring individuals from the parent
- Step 5: go to Step 2 unless termination condition is satisfied

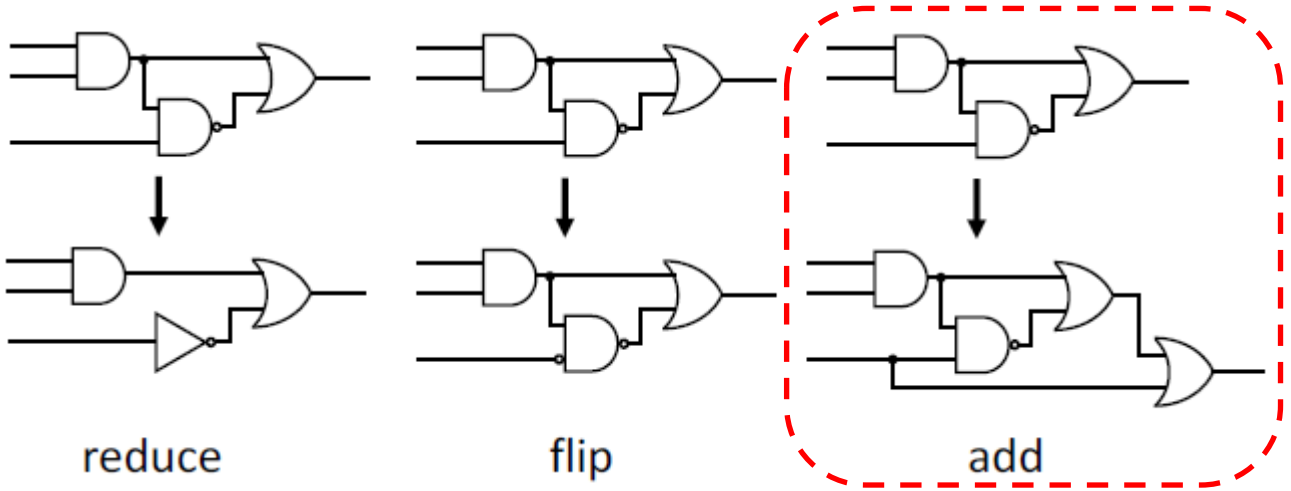
Chromosome: $(1, 3, 0); (0, 2, 0); (1, 2, 0); (0, 1, 0); (7, 6, 1); (8, 8, 1); [5, 8, 4, \text{‘0’}]$

Outline

- Based on which LACs to consider and which to select
  - Consider a deterministic subset + select one
  - Consider a deterministic subset + select multiple
  - Consider a random subset + select one
    - Evolutionary algorithm-based method
    - Random choice and probabilistic acceptance-based method

Random Choice and Probabilistic Acceptance

- LACs



increase area

+ **Traditional logic synthesis**

- Which LACs to consider? Randomly choose one
- Should it be applied? (i.e., selection criterion)
  - Calculate quality metric: $Q = \alpha \cdot Area_m + \beta \cdot ErrorMetric$
  - Accept a move stochastically
    - If the move improves the quality, accept the move
    - Otherwise, accept with probability $e^{-\gamma(Q_{new}/Q_{old})}$

Liu & Zhang, "Statistically certified approximate logic synthesis," *ICCAD'17*

Outline

- Background on Approximate Computing
- Logic Synthesis for Approximate Circuits
- Applications of Approximate Circuits
  - Image Processing
  - Deep Neural Networks
- Conclusion

Outline

- Background on Approximate Computing
- Logic Synthesis for Approximate Circuits
- Applications of Approximate Circuits
  - Image Processing
  - Deep Neural Networks
- Conclusion

Edge Detection Application

- Sobel edge detection $O_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \circledast I_x, O_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \circledast I_y$

- Approximate adder generated by approximate logic synthesis [Meng+, DAC'20]

Input image

Output by 16-bit accurate adder

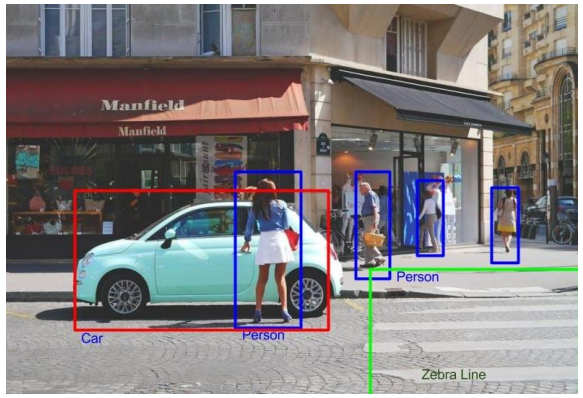
Output by synthesized approximate adder

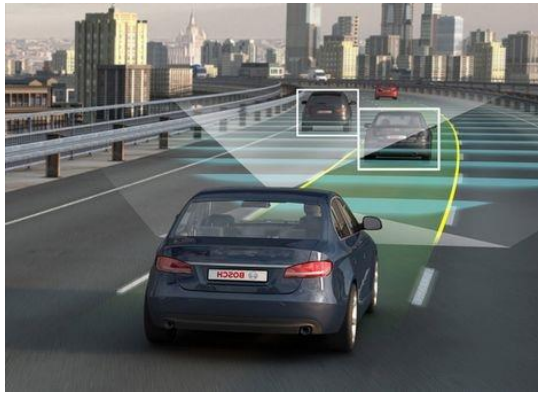
Area (um^2)	82.73	56.9
Delay (ns)	5	2.8
Area x Delay	413	159 (2.6X)

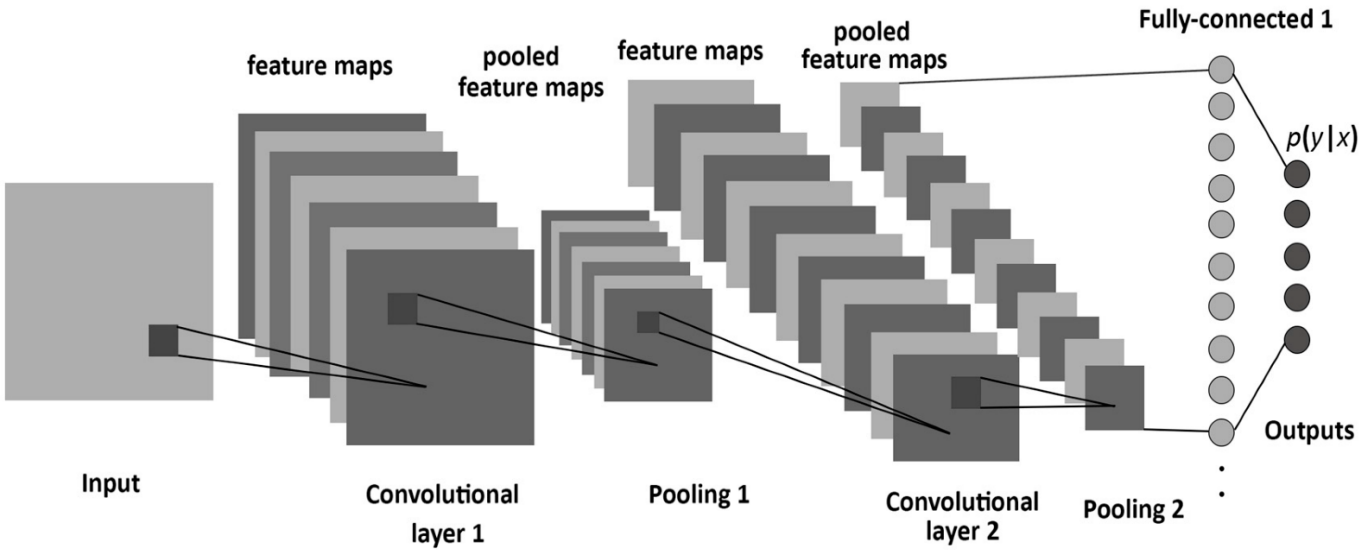
Outline

- Background on Approximate Computing
- Logic Synthesis for Approximate Circuits
- Applications of Approximate Circuits
  - Image Processing
  - Deep Neural Networks
- Conclusion

Deep Neural Networks (DNNs)

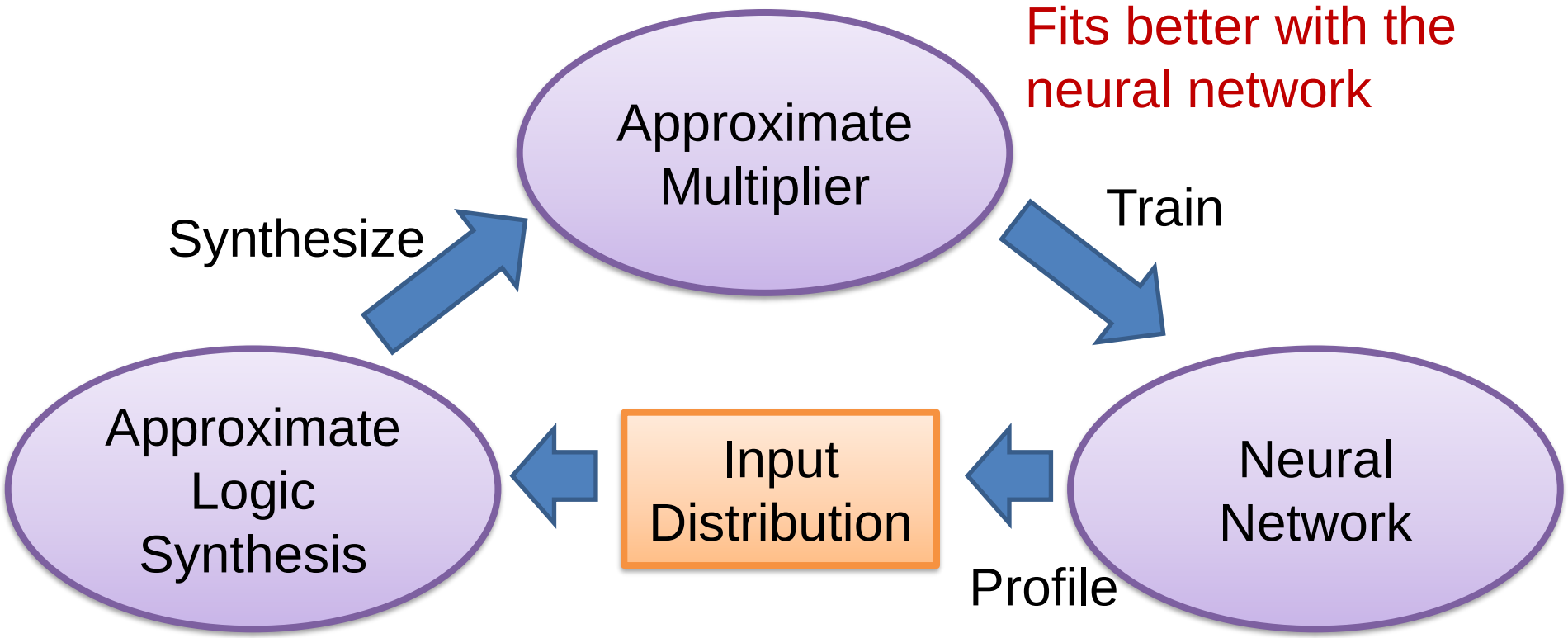






- Many arithmetic operations

Low-Power DNN Accelerator by Approximate Logic Synthesis



[Qian+, ISCAS'21]

Experimental Results

- LeNet-5 + MNIST dataset
- Synthesize approximate multipliers [Meng+, DAC'20]
- Loop has 5 iterations

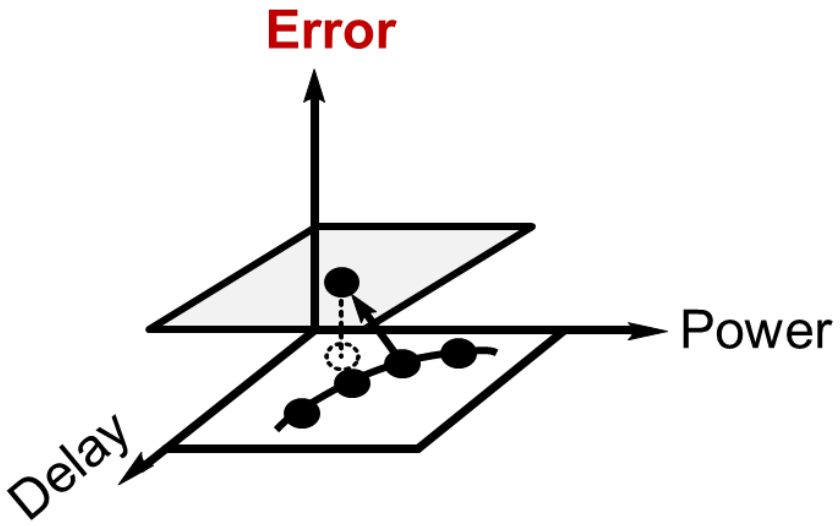
Circuit type	error bound	area	delay	accuracy
Accurate 8-bit multiplier	-	1326	27.1	99.00%
Approximate multiplier 1	0.001	966	27	98.86%
Approximate multiplier 2	0.003	786	26	98.74%
Approximate multiplier 3	0.006	202	18.1	98.67%
Approximate multiplier 4	0.012	73	10.7	98.44%
Approximate multiplier 5	0.024	56	6.7	97.86%
2-bit rounded multiplier	-	72	7.7	97.52%

Final design (Area 4.2% of the accurate multiplier) →

Outline

- Background on Approximate Computing
- Logic Synthesis for Approximate Circuits
- Applications of Approximate Circuits
- Conclusion

Conclusion



- Approximate computing
  - Targeting at error-tolerant application
  - Trading accuracy for area/delay/power
- Logic synthesis for approximate circuits
  - Consider a deterministic subset of LACs + select one
  - Consider a deterministic subset of LACs + select multiple
  - Consider a random subset of LACs + select one
- Applications of approximate circuits
  - Image Processing
  - Deep Neural Networks

Thank you