Lecture 6 - LANDAUER: Computing with uncertainty

Igor Neri - NiPS Laboratory, University of Perugia
July 18, 2014

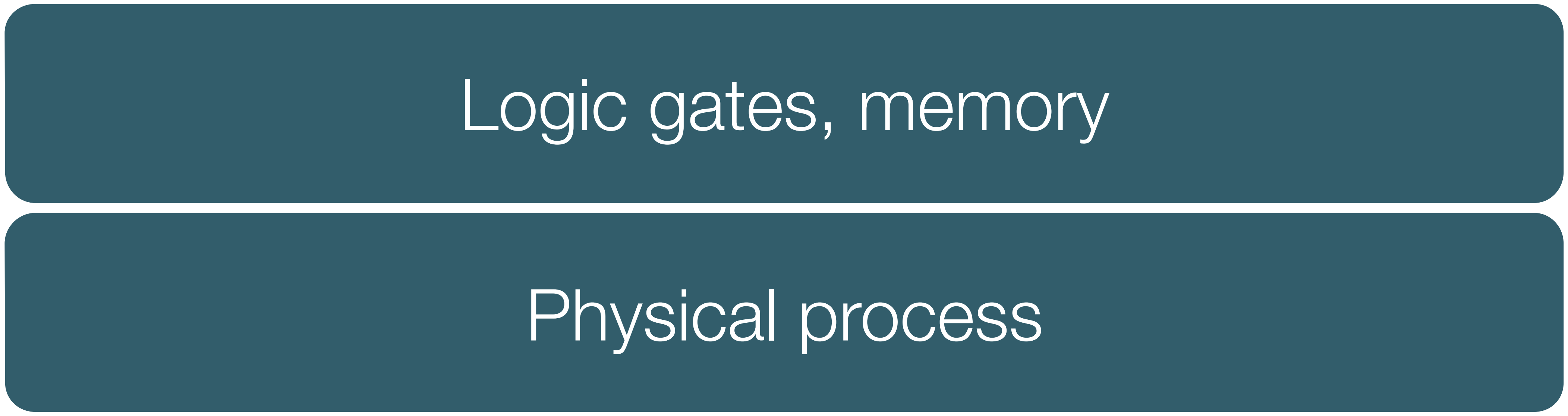
NiPS Summer School 2014
ICT-Energy: Energy management at micro and nanoscales for future ICT

We are used to correct results

User applications, Algorithms

Correct logic operations

Physical process underlying computation are not exact



Additional HW and SW to bridge the gap between reliable and not reliable

User applications, Algorithms

Correct logic operations

Redundancy Error correction

Logic gates, memory

Physical process

Why

Why

- Save time

Why

- Save time
- Save energy

Why

- Save time
- Save energy

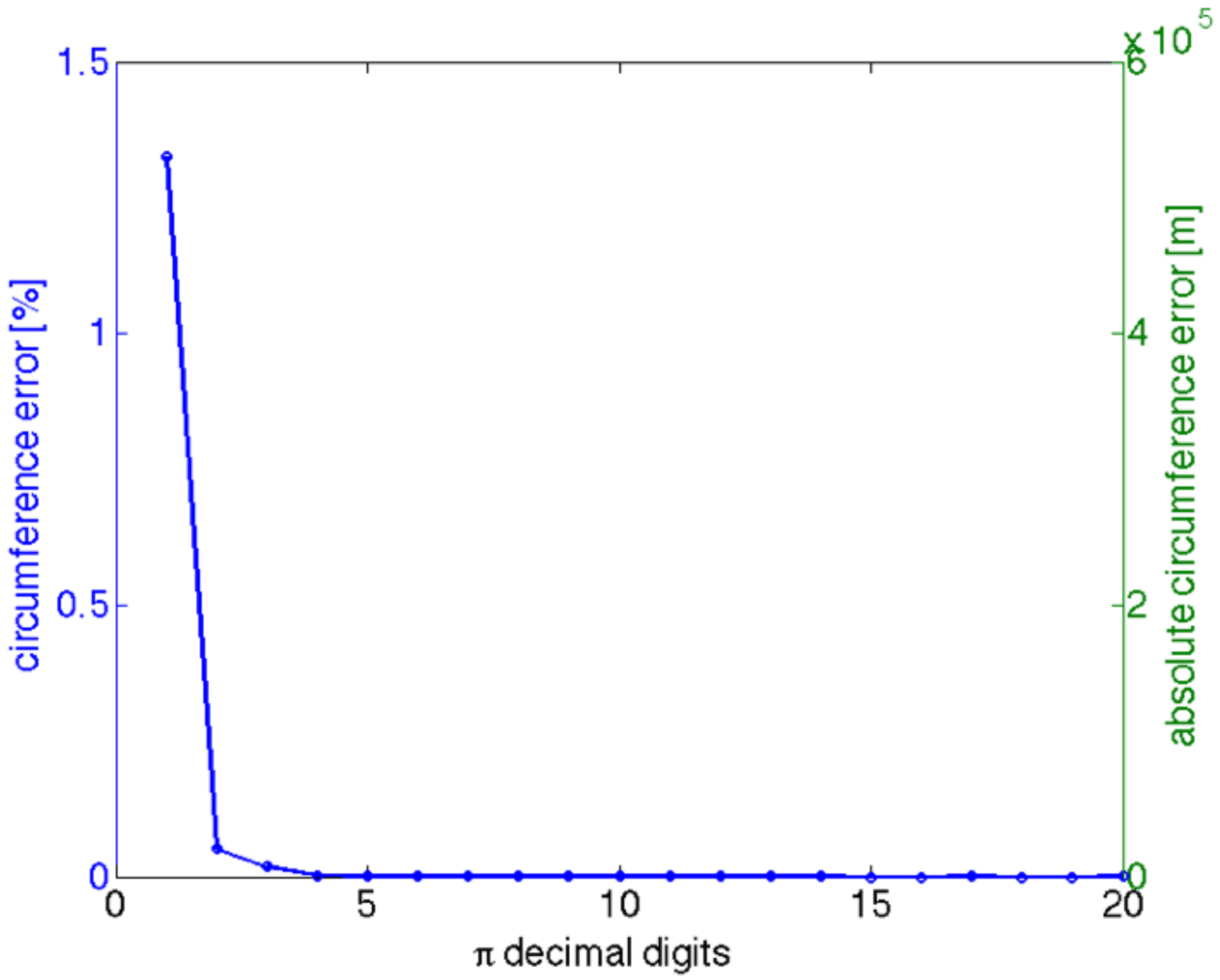
Why

- Save time
- Save energy
- We are already doing it!

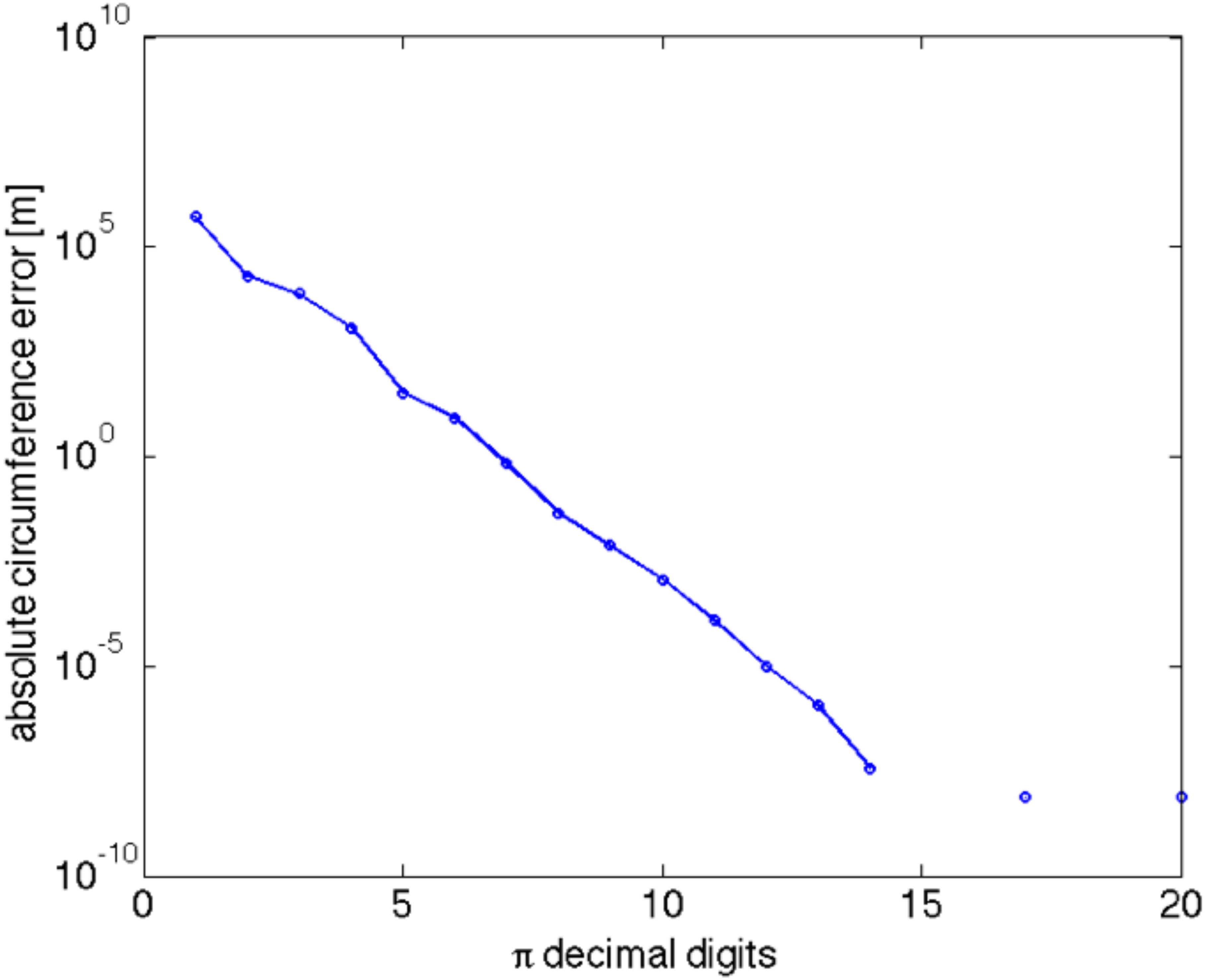
How many digits of π for error lesser then one meter on Earth circumference?

r = 6,371 kilometers

How many digits of π for error lesser then one meter on Earth circumference?



How many digits of π for error lesser then one meter on Earth circumference?

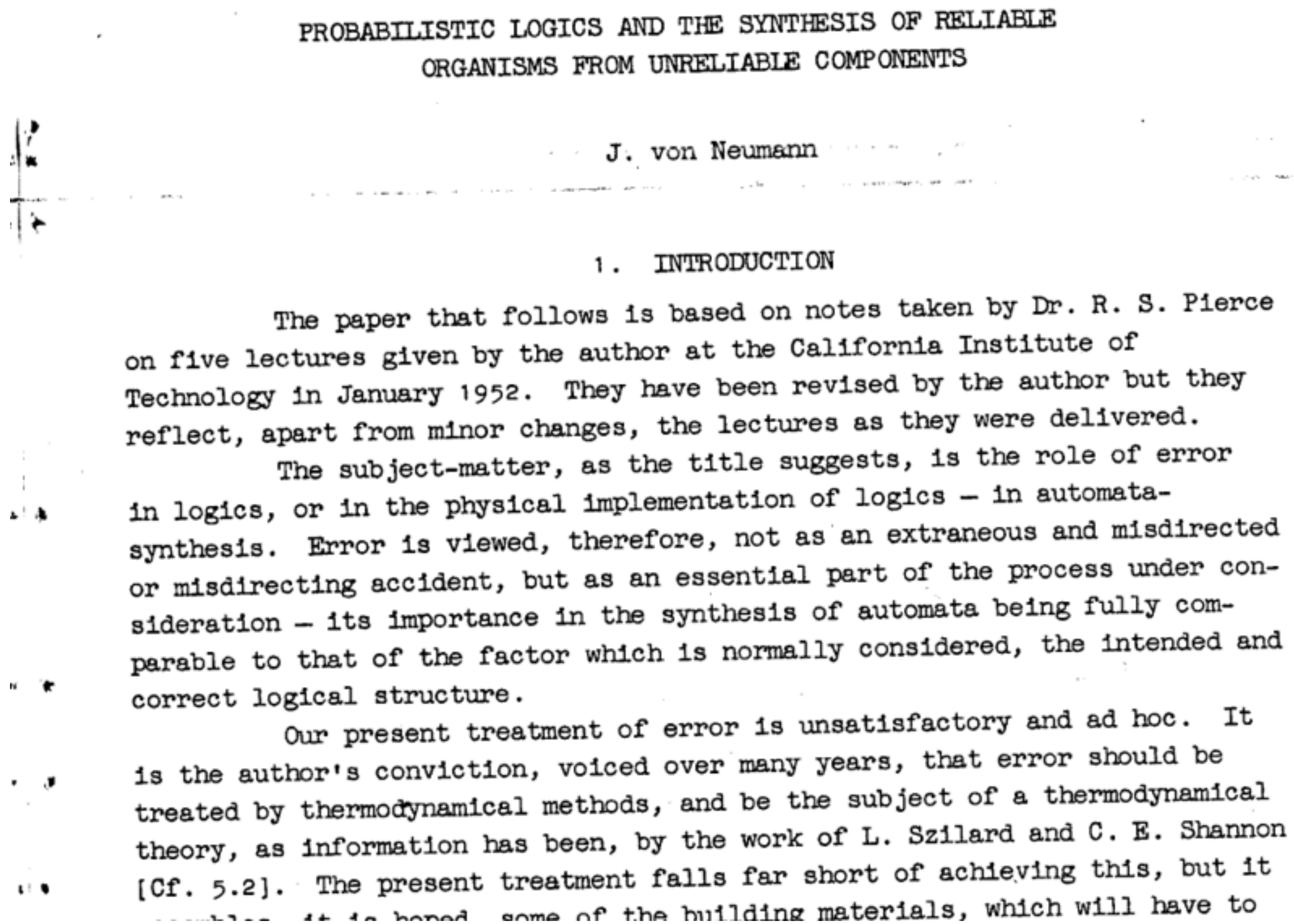


Outline

- History:
  - Probabilistic Logics (Von Neumann) and Stochastic Computing
- Nowadays:
  - Approximate programming
  - Computing with uncertainty

Probabilistic Logics (Von Neumann) and Stochastic Computing

Stochastic computing



PROBABILISTIC LOGICS AND THE SYNTHESIS OF RELIABLE
ORGANISMS FROM UNRELIABLE COMPONENTS

J. von Neumann

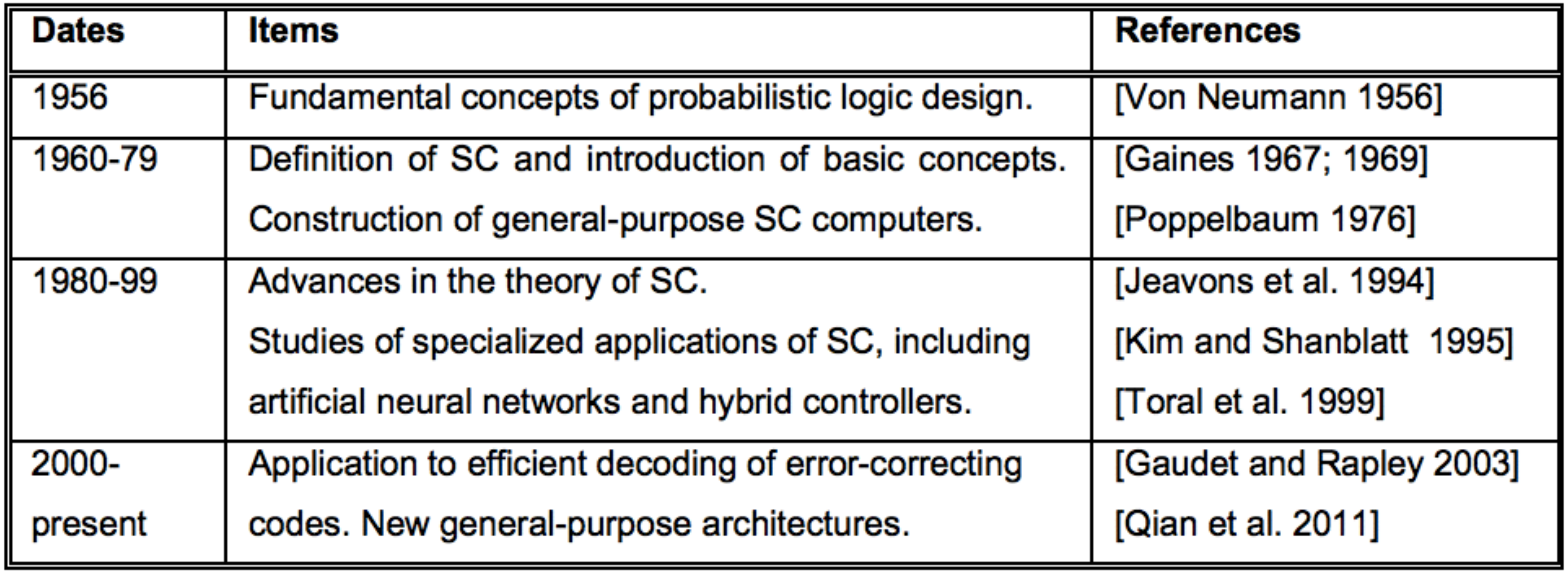
1. INTRODUCTION

The paper that follows is based on notes taken by Dr. R. S. Pierce on five lectures given by the author at the California Institute of Technology in January 1952. They have been revised by the author but they reflect, apart from minor changes, the lectures as they were delivered.

The subject-matter, as the title suggests, is the role of error in logics, or in the physical implementation of logics — in automata-synthesis. Error is viewed, therefore, not as an extraneous and misdirected or misdirecting accident, but as an essential part of the process under consideration — its importance in the synthesis of automata being fully comparable to that of the factor which is normally considered, the intended and correct logical structure.

Our present treatment of error is unsatisfactory and ad hoc. It is the author's conviction, voiced over many years, that error should be treated by thermodynamical methods, and be the subject of a thermodynamical theory, as information has been, by the work of L. Szilard and C. E. Shannon [Cf. 5.2]. The present treatment falls far short of achieving this, but it [illegible] it is hoped, some of the building materials, which will have to

Stochastic computing: timeline



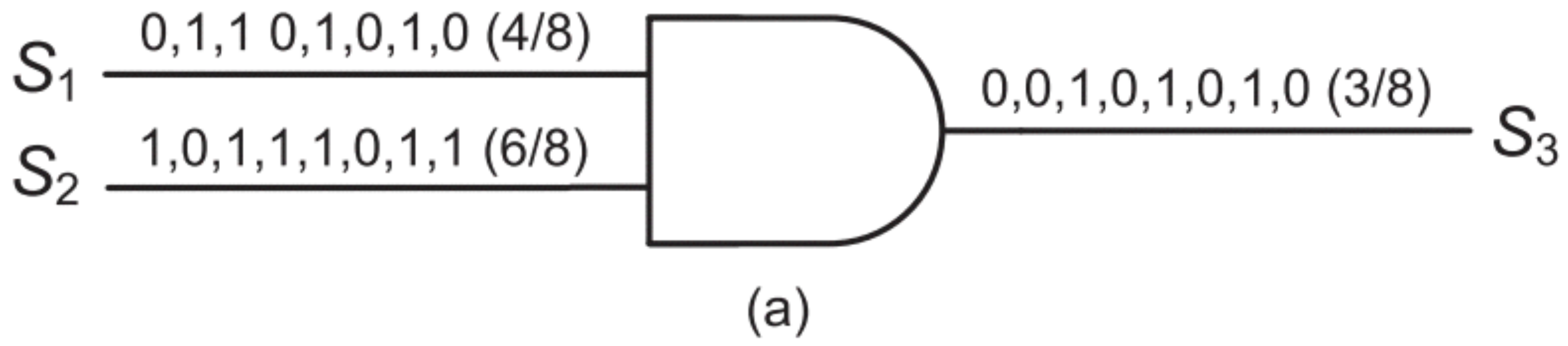
Dates	Items	References
1956	Fundamental concepts of probabilistic logic design.	[Von Neumann 1956]
1960-79	Definition of SC and introduction of basic concepts. Construction of general-purpose SC computers.	[Gaines 1967; 1969] [Poppelbaum 1976]
1980-99	Advances in the theory of SC. Studies of specialized applications of SC, including artificial neural networks and hybrid controllers.	[Jeavons et al. 1994] [Kim and Shanblatt 1995] [Toral et al. 1999]
2000-present	Application to efficient decoding of error-correcting codes. New general-purpose architectures.	[Gaudet and Rapley 2003] [Qian et al. 2011]

Survey of Stochastic Computing - J. Hayes, ACM 2012

Stochastic computing: basic features

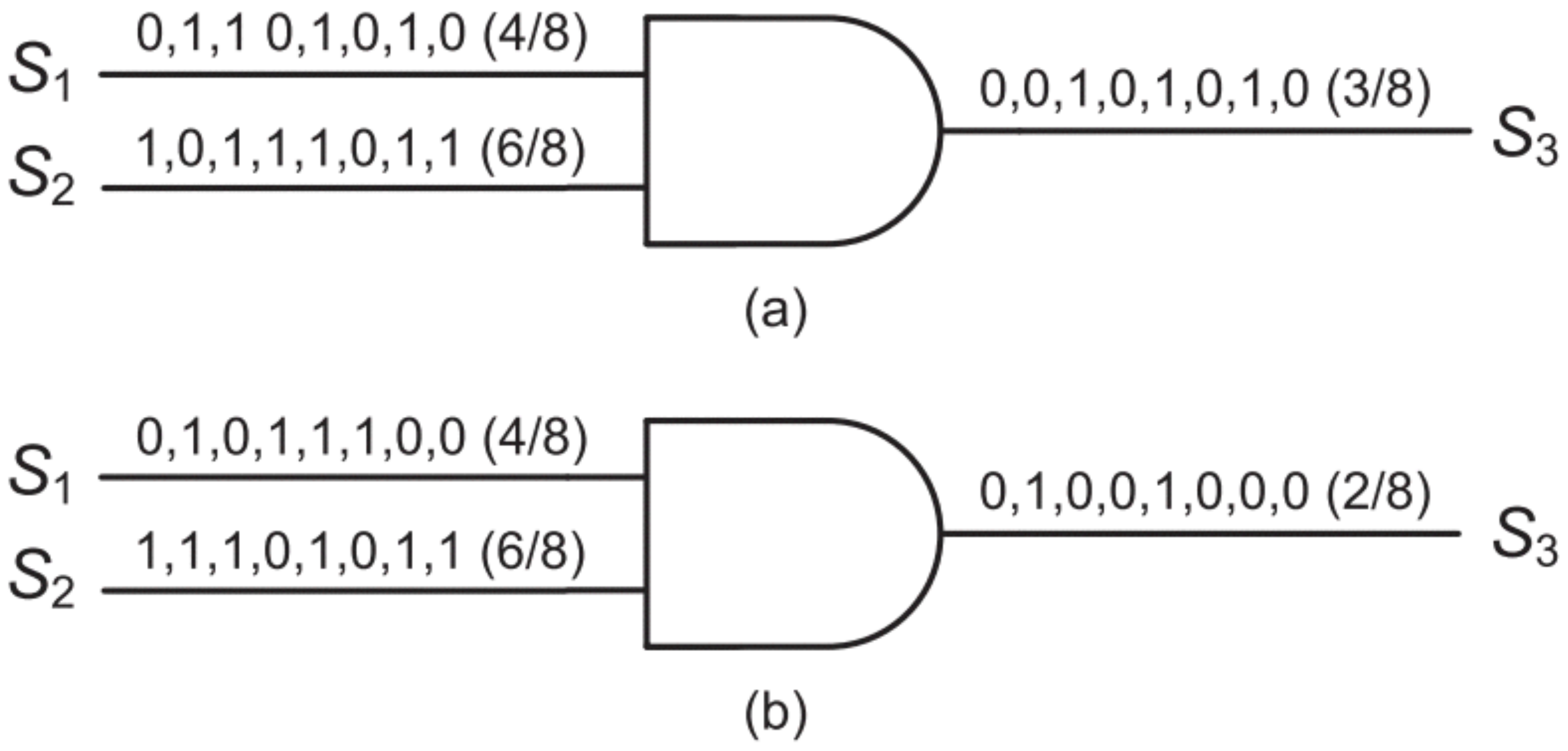
- Numbers are represented by bit-streams
- Numbers can be processed by very simple circuits
- Numbers are interpreted as probabilities under both normal and faulty conditions
    - A bit-stream S containing 25 percent of 1s and 75 percent if 0s denotes the number p = 0.25
        - (1,0,0,0), (0,1,0,0) and (0,1,0,0,0,1,0,0) are all possible representations of 0.25

Stochastic computing: multiplication



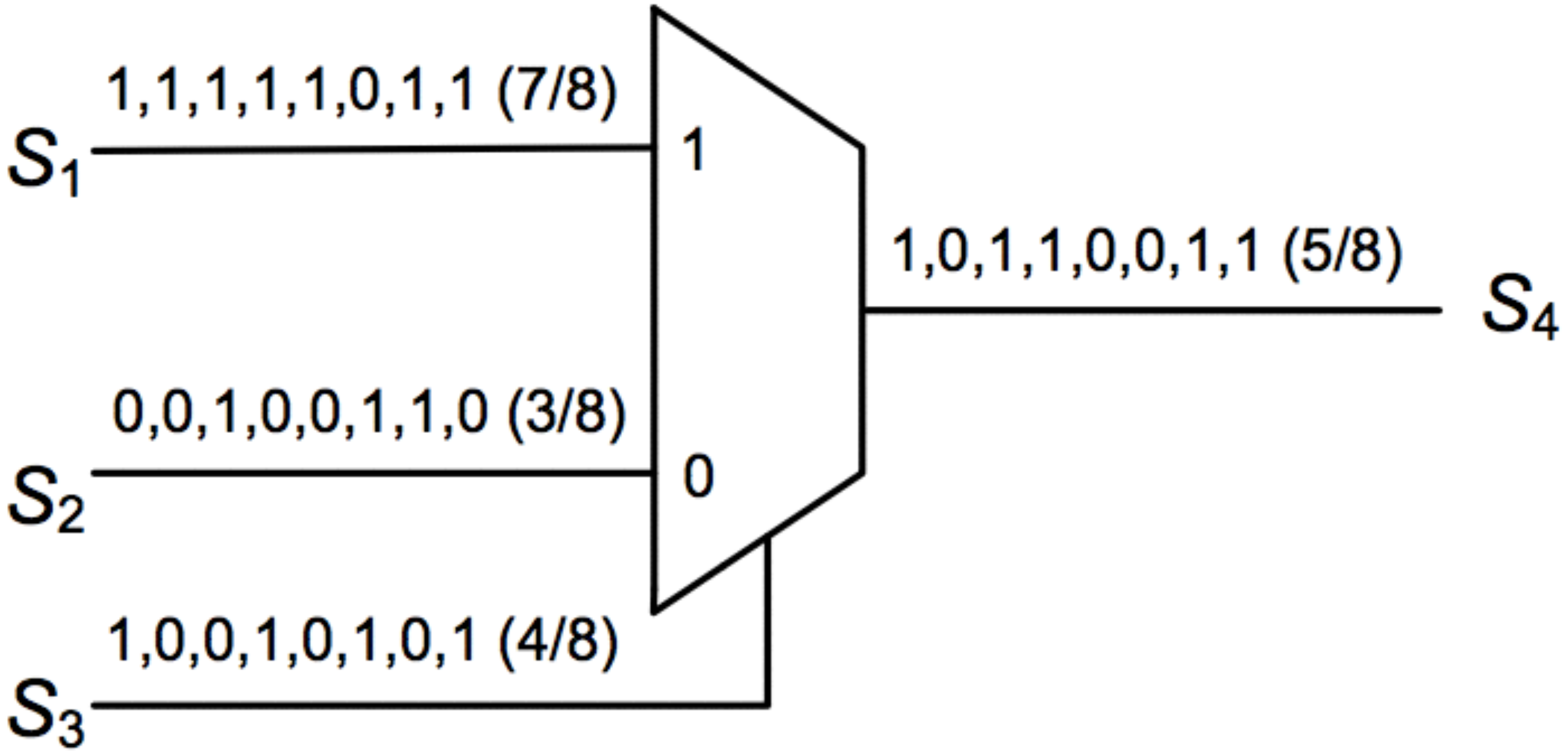
Survey of Stochastic Computing - J. Hayes, ACM 2012

Stochastic computing: multiplication



Survey of Stochastic Computing - J. Hayes, ACM 2012

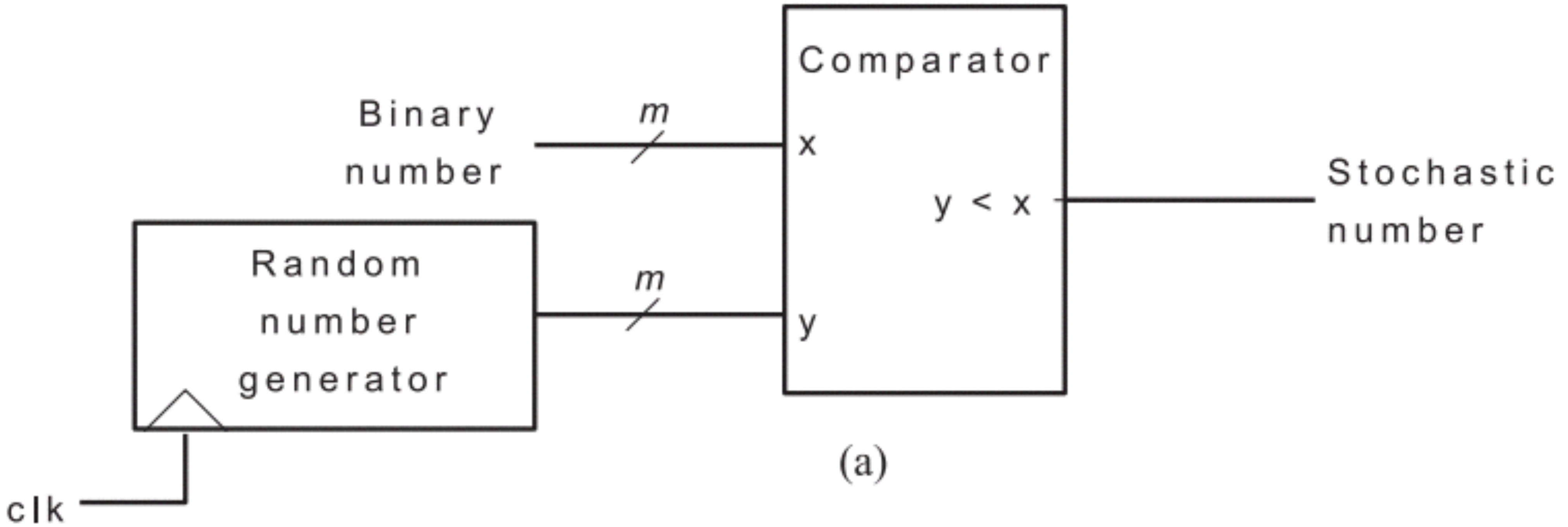
Stochastic computing: sum



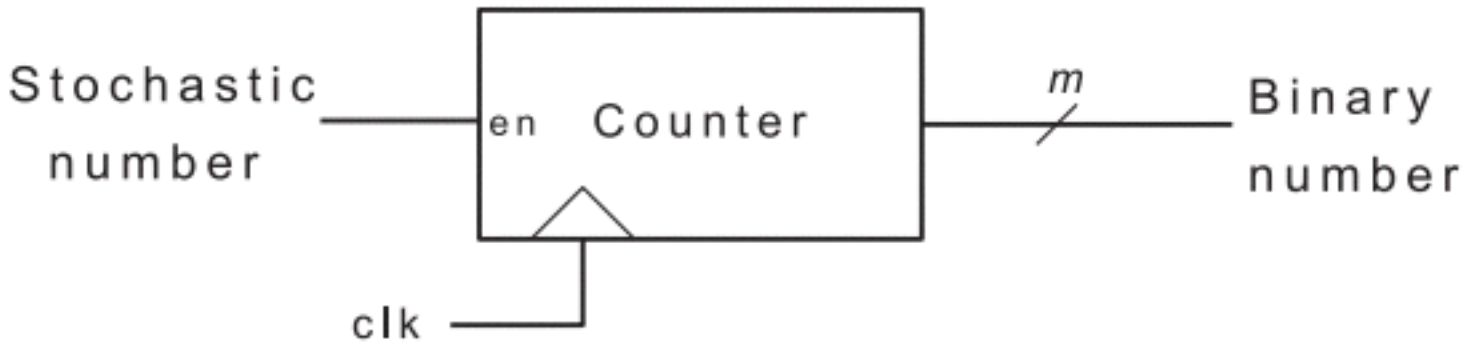
$$p(S_4) = p(S_3)\,p(S_1) + (1 - p(S_3))\,p(S_2) = (p(S_1) + p(S_2))/2$$

Survey of Stochastic Computing - J. Hayes, ACM 2012

Stochastic number conversion



(a)



(b)

Survey of Stochastic Computing - J. Hayes, ACM 2012

Stochastic computing: stochastic numbers

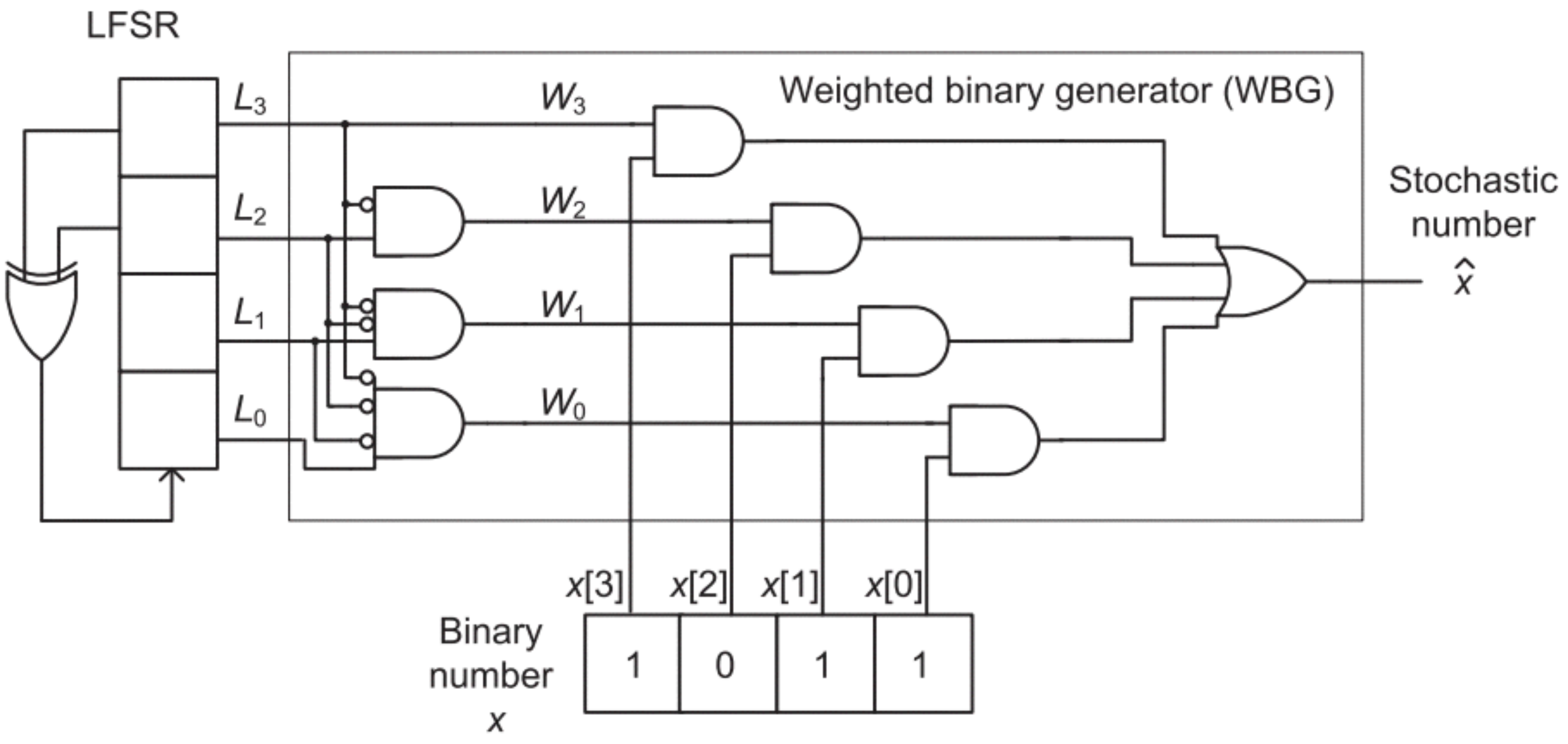
- Fluctuations inherent in random numbers
- Correlations among the stochastic numbers

$$\sum_{i=1}^{n} S_1(i)S_2(i) = \frac{\sum_{i=1}^{n} S_1(i) \times \sum_{i=1}^{n} S_2(i)}{n}$$

- It is generally not desirable to use truly random number sources to derive or process stochastic numbers

Stochastic Number Generators (SNGs)

- Linear Feedback Shift Register (LFSR)



Gupta and Kumaresan [1988]

Stochastic computing: accuracy

- With m bits of precision, we can distinguish between 2^m different numbers
- The numbers in the interval [0,1], when represented with eight-bit precision reduce to the following 256-member set: {0/256, 1/256, 2/256, ..., 255/256, 256/256}
- Their stochastic representation requires bit-streams of length 256
- To increase the precision from eight to nine bits requires doubling the bit-stream length to 512

Survey of Stochastic Computing - J. Hayes, ACM 2012

Stochastic computing: applications

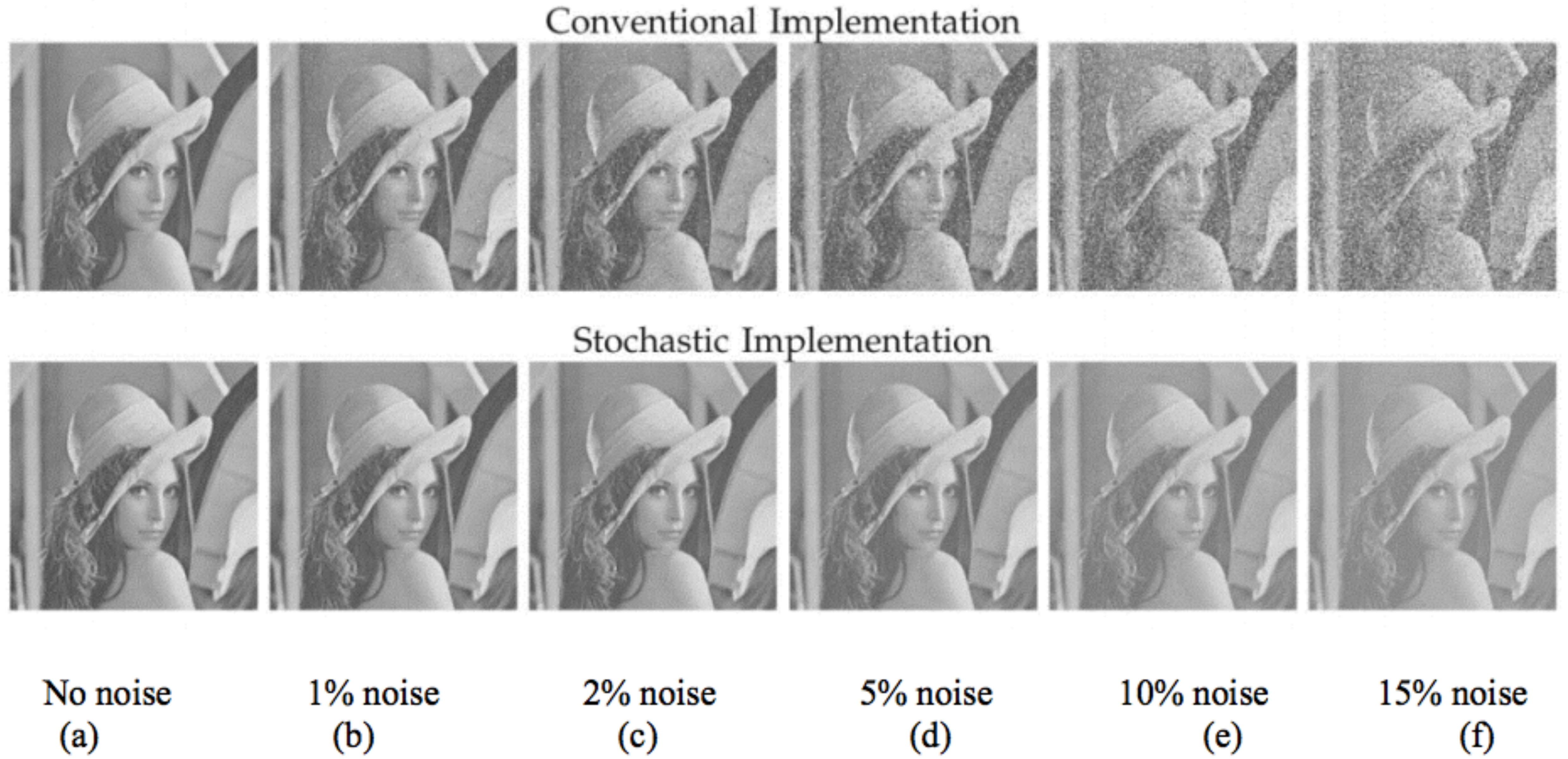


Figure 10. Error-tolerance comparison between conventional and SC implementations of a gamma correction function. © 2011 IEEE. Reprinted, with permission, from [Qian et al. 2011].

Survey of Stochastic Computing - J. Hayes, ACM 2012

Stochastic computing applications: LDPC

- Linear error correcting code
- Method to transmit a message over a noisy channel
- Rate near to the theoretical maximum
- Difficult to implement in practice
- Largely ignored until 1990s
- Used in WiFi and digital video broadcasting

Survey of Stochastic Computing - J. Hayes, ACM 2012

Stochastic computing applications: LDPC

- Nine-bit single-error detecting code

$$x_0 \oplus x_1 \oplus x_2 \oplus \ldots \oplus x_7 \oplus x_8 = 0$$

- x_0, x_1,…,x_7 are data bits
- x_8 is the parity bit

Survey of Stochastic Computing - J. Hayes, ACM 2012

Stochastic computing applications: LDPC

- LDPC code of length 6

$$\begin{aligned} x_0 \oplus x_2 \oplus x_3 &= 0; \\ x_1 \oplus x_2 &= 0; \\ x_0 \oplus x_4 &= 0; \\ x_1 \oplus x_5 &= 0. \end{aligned}$$

Stochastic computing applications: LDPC

- LDPC code of length 6

$$\begin{aligned} x_0 \oplus x_2 \oplus x_3 &= 0; \\ x_1 \oplus x_2 &= 0; \\ x_0 \oplus x_4 &= 0; \\ x_1 \oplus x_5 &= 0. \end{aligned}$$

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = 0,$$

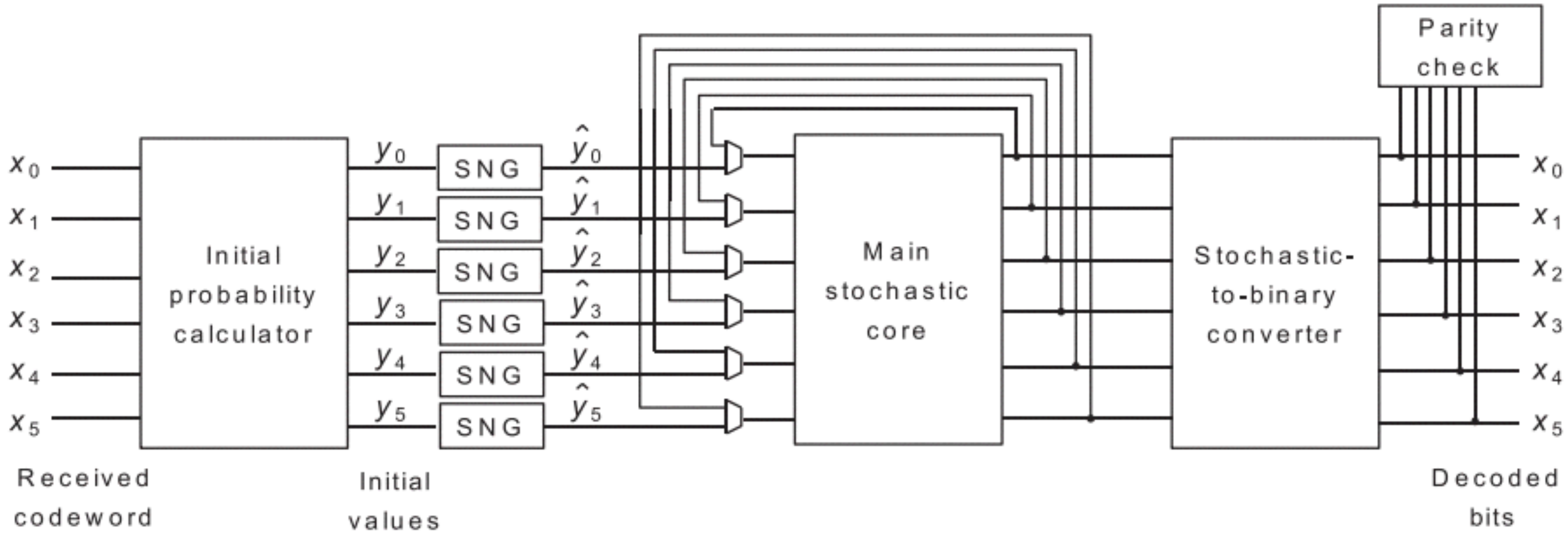
Stochastic computing applications: LDPC

$$\mathbf{H}.\vec{\mathbf{x}} = \begin{bmatrix} b_{0,0} & \cdots & b_{0,n-1} \\ \dots & \ddots & \dots \\ b_{k-1,0} & \cdots & b_{k-1,n-1} \end{bmatrix} \begin{bmatrix} x_0 \\ \vdots \\ x_{n-1} \end{bmatrix} = 0,$$

- A typical parity matrix H is huge (n and k can be many thousands), but it is sparse in that it has only a few 1s in each row and column
- A relatively small subset of all 2^n possible combinations of 0s and 1s on x_0, x_1, ...,x_{n-1} satisfy the code equations
- These are the valid codewords

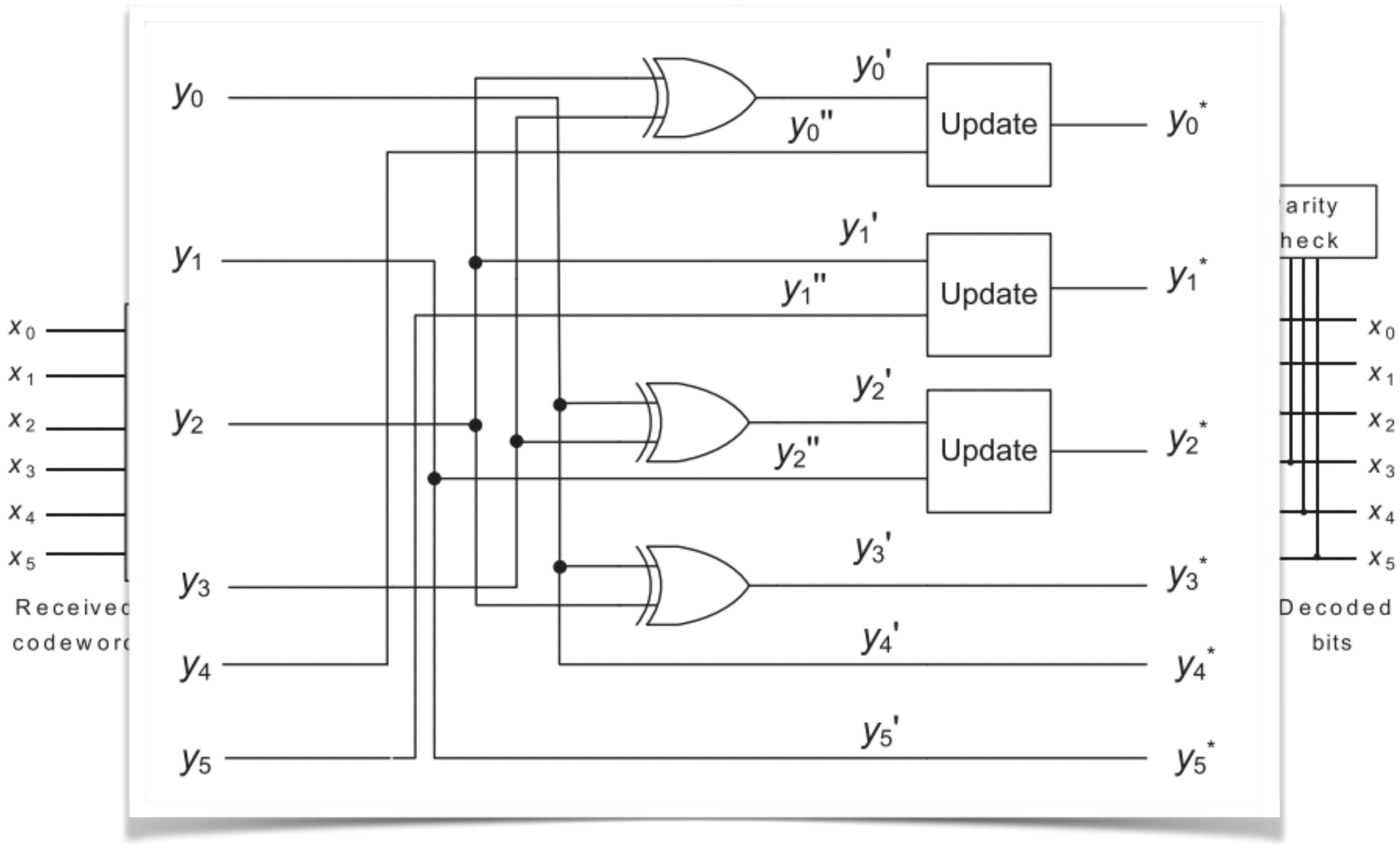
Survey of Stochastic Computing - J. Hayes, ACM 2012

Stochastic computing applications: LDPC



Survey of Stochastic Computing - J. Hayes, ACM 2012

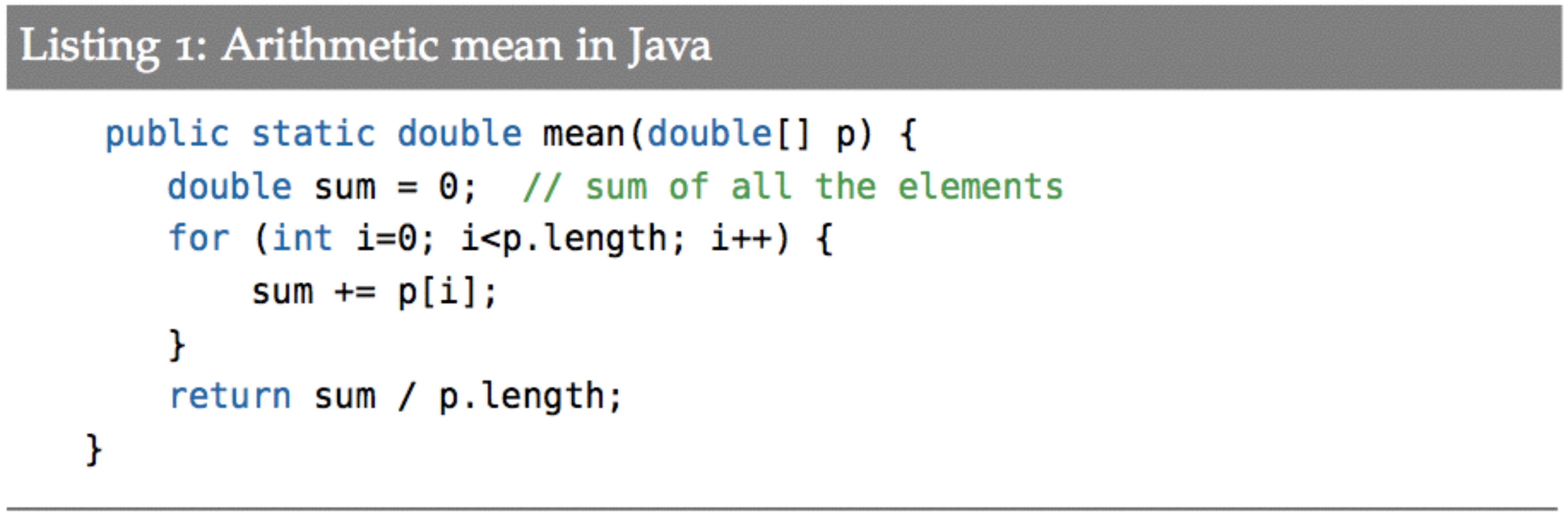
Stochastic computing applications: LDPC



Survey of Stochastic Computing - J. Hayes, ACM 2012

Approximate programming

Approximate programming



Listing 1: Arithmetic mean in Java

```
 public static double mean(double[] p) {
    double sum = 0;  // sum of all the elements
    for (int i=0; i<p.length; i++) {
        sum += p[i];
    }
    return sum / p.length;
}
```

Approximate programming

Listing 1: Arithmetic mean in Java

```java
public static double mean(double[] p) {
   double sum = 0;  // sum of all the elements
   for (int i=0; i<p.length; i++) {
       sum += p[i];
   }
   return sum / p.length;
}
```

Listing 2: Approximate arithmetic mean in Java

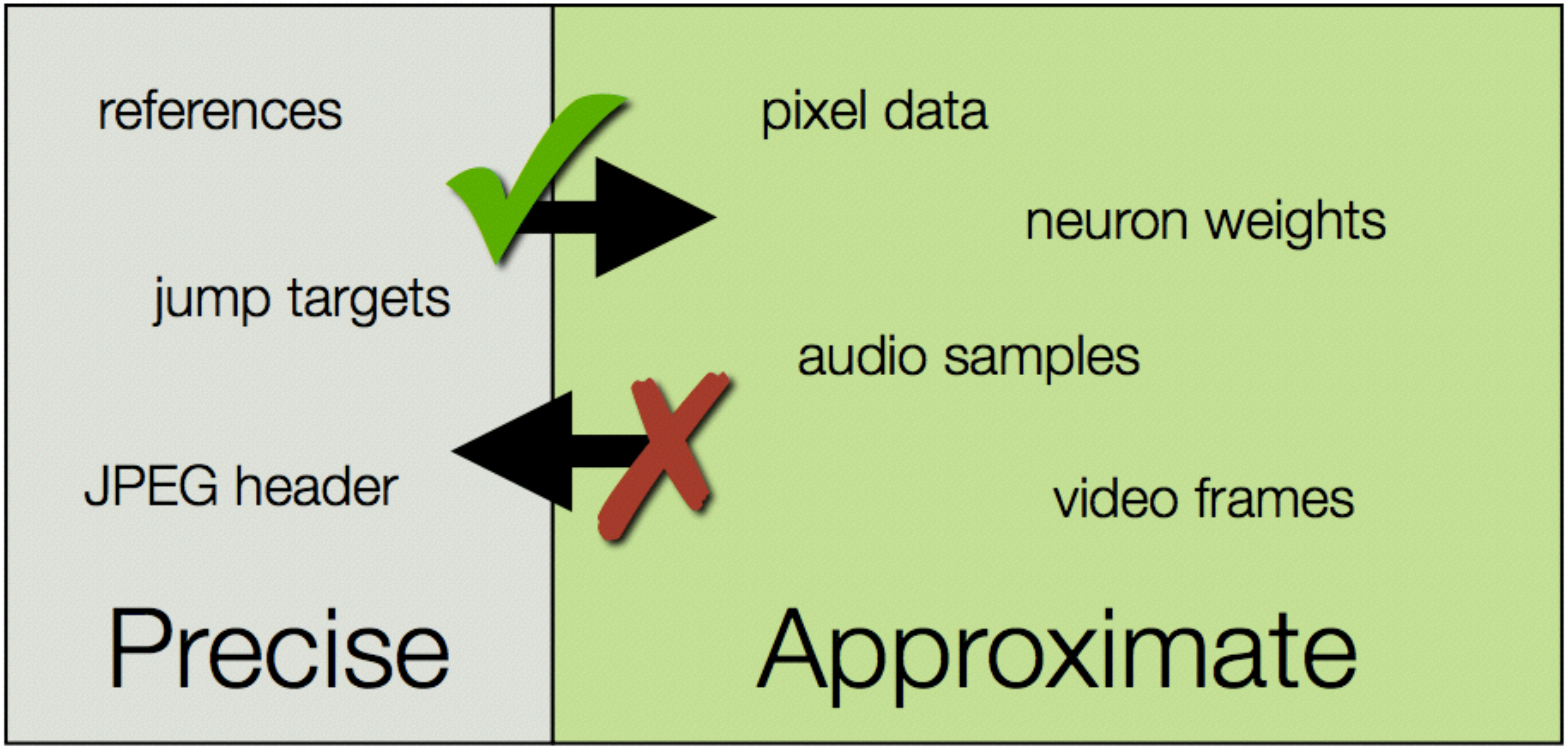
```java
public static double mean(double[] p) {
   double sum = 0;  // sum of all the elements
   for (int i=0; i<p.length; i=+2) {
       sum += p[i];
   }
   return 2 * sum / p.length;
}
```

Approximate programming

```
@Approx float[] nums;
⋮
@Approx float total = 0.0f;
for (@Precise int i = 0;
     i < nums.length;
     ++i)
  total += nums[i];
return total / nums.length;
```

Sampson, Adrian, et al. "EnerJ: Approximate data types for safe and general low-power computation." ACM SIGPLAN Notices. Vol. 46. No. 6. ACM, 2011.
Disciplined Approximate Computing: From Language to Hardware - L. Ceze - 2013

Approximate programming

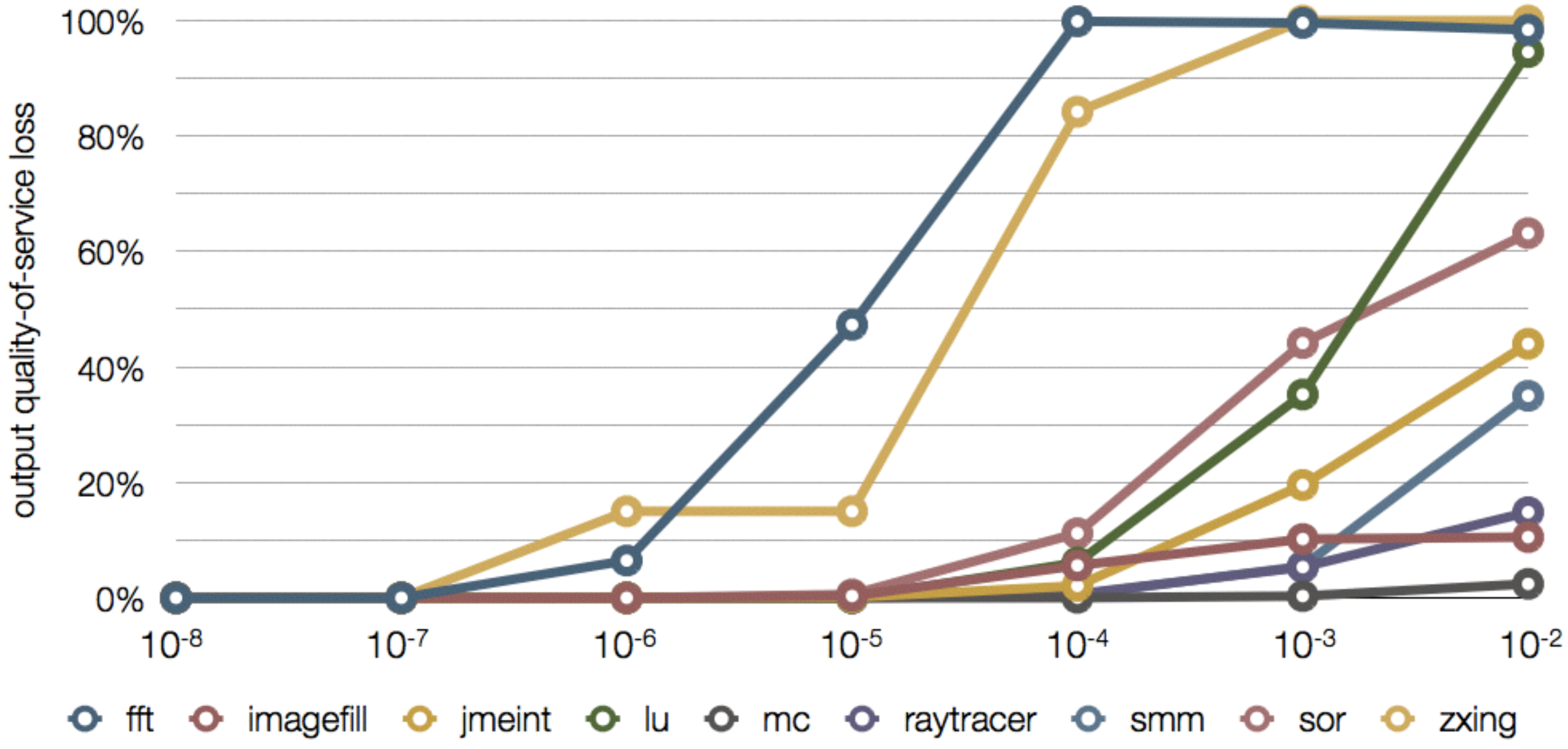


Approximate programming: HW support

- Approximation-aware ISA
- Approximate operations
- Approximate data
- Dual-voltage microarchitecture

Architecture Support for Disciplined Approximate Programming - H. Esmaeilzadeh et al. - ASPLOS'12
Disciplined Approximate Computing: From Language to Hardware - L. Ceze - 2013

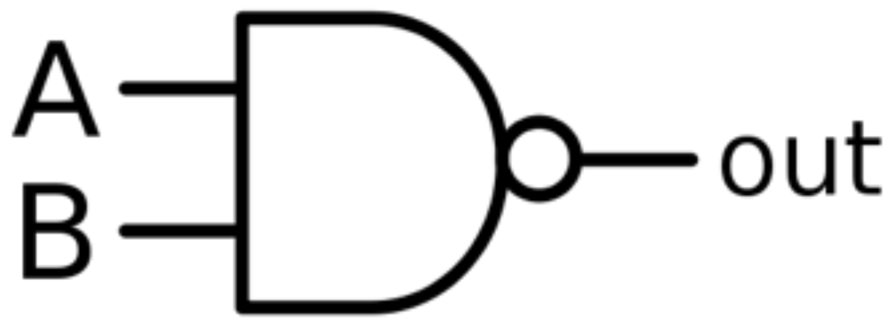
Approximate programming: results

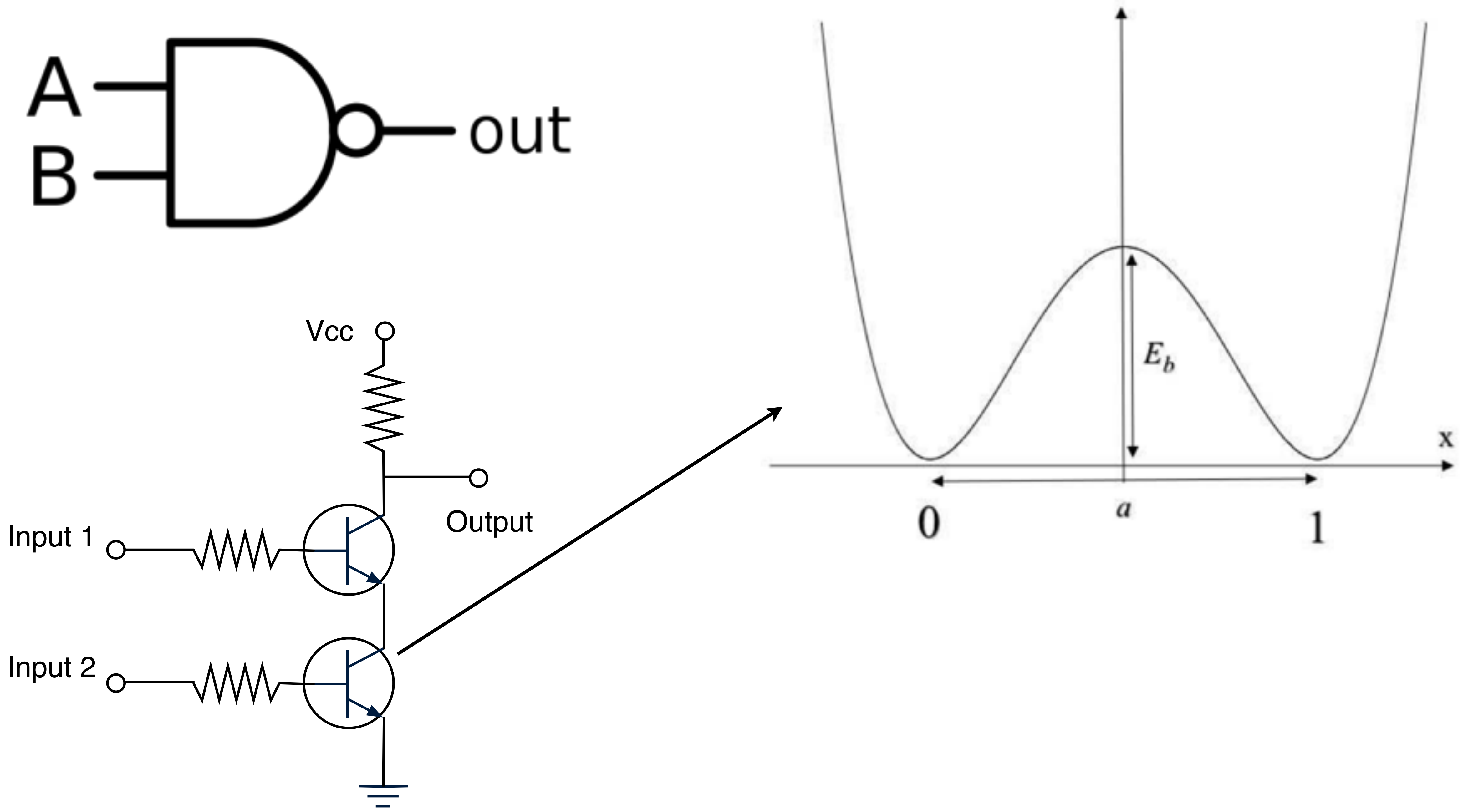


Architecture Support for Disciplined Approximate Programming - H. Esmaeilzadeh et al. - ASPLOS'12

What we did

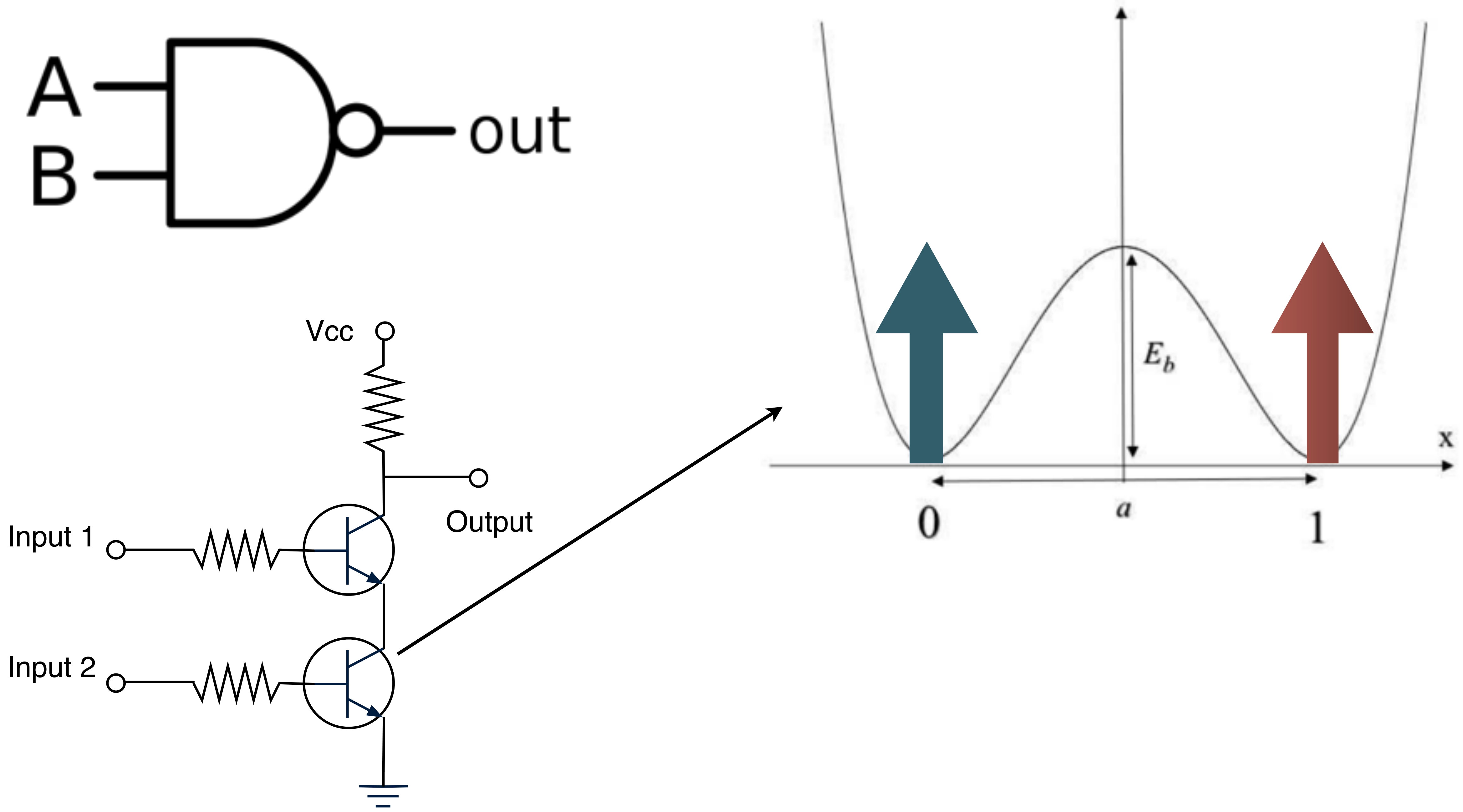
Sub-Landauer gate



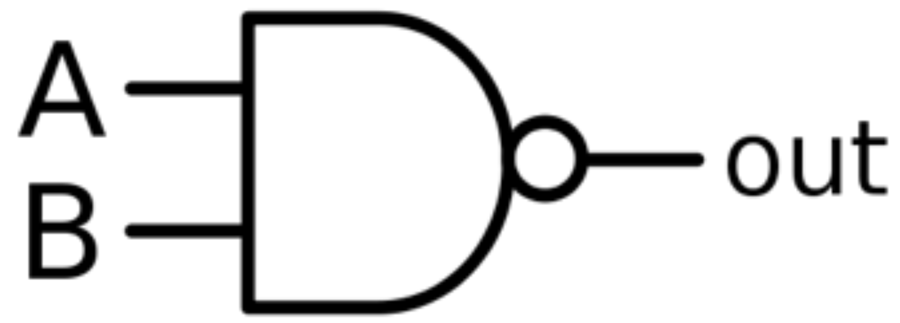


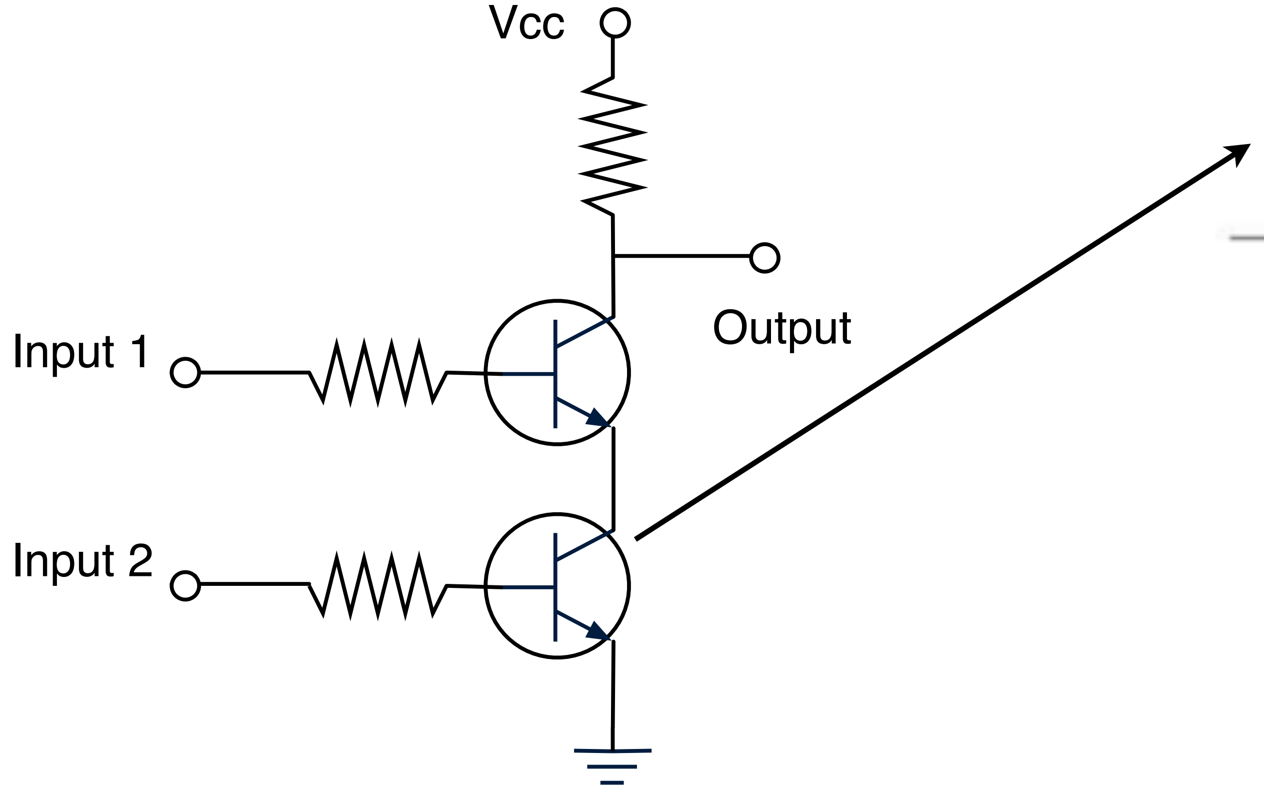
Beating the Landauer's limit by trading energy with uncertainty - L. Gammaitoni - arXiv:1111.2937 [cond-mat.mtrl-sci]

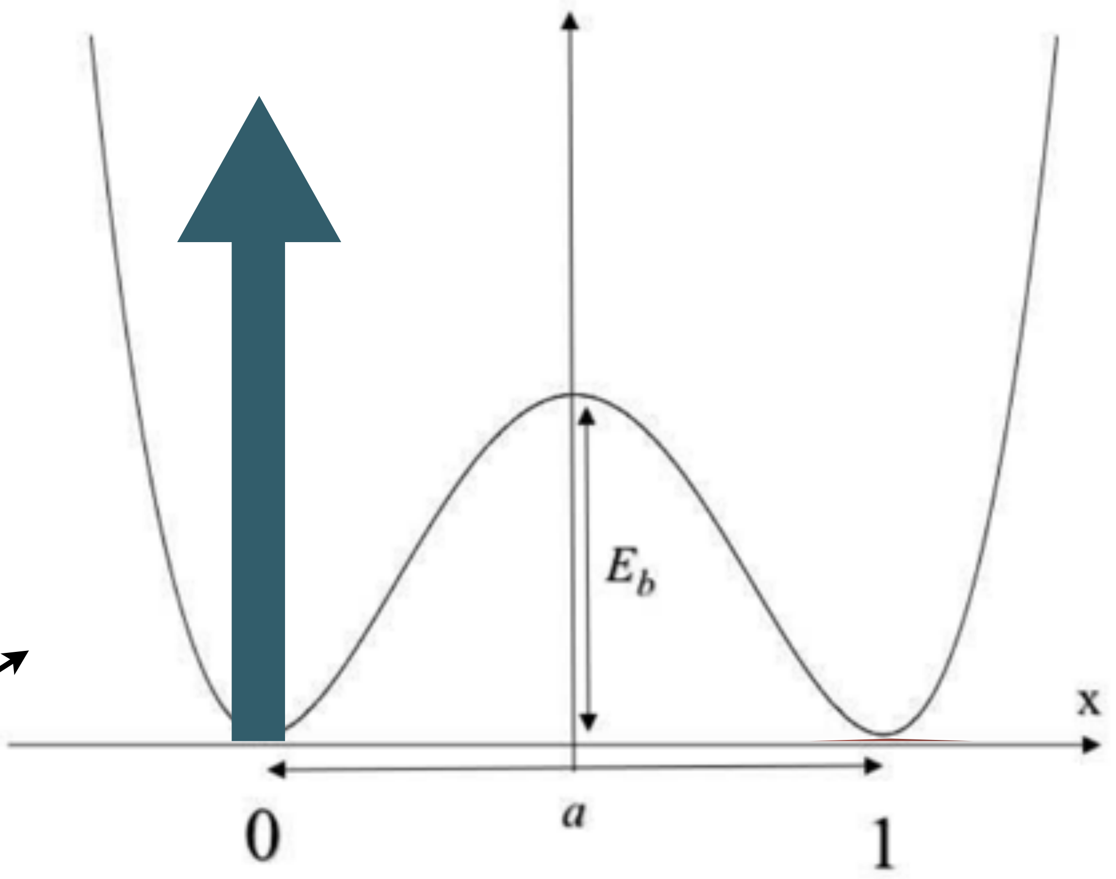
Sub-Landauer gate



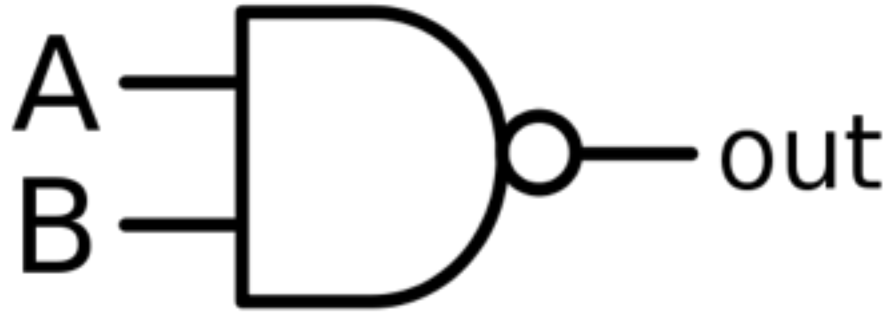
Sub-Landauer gate

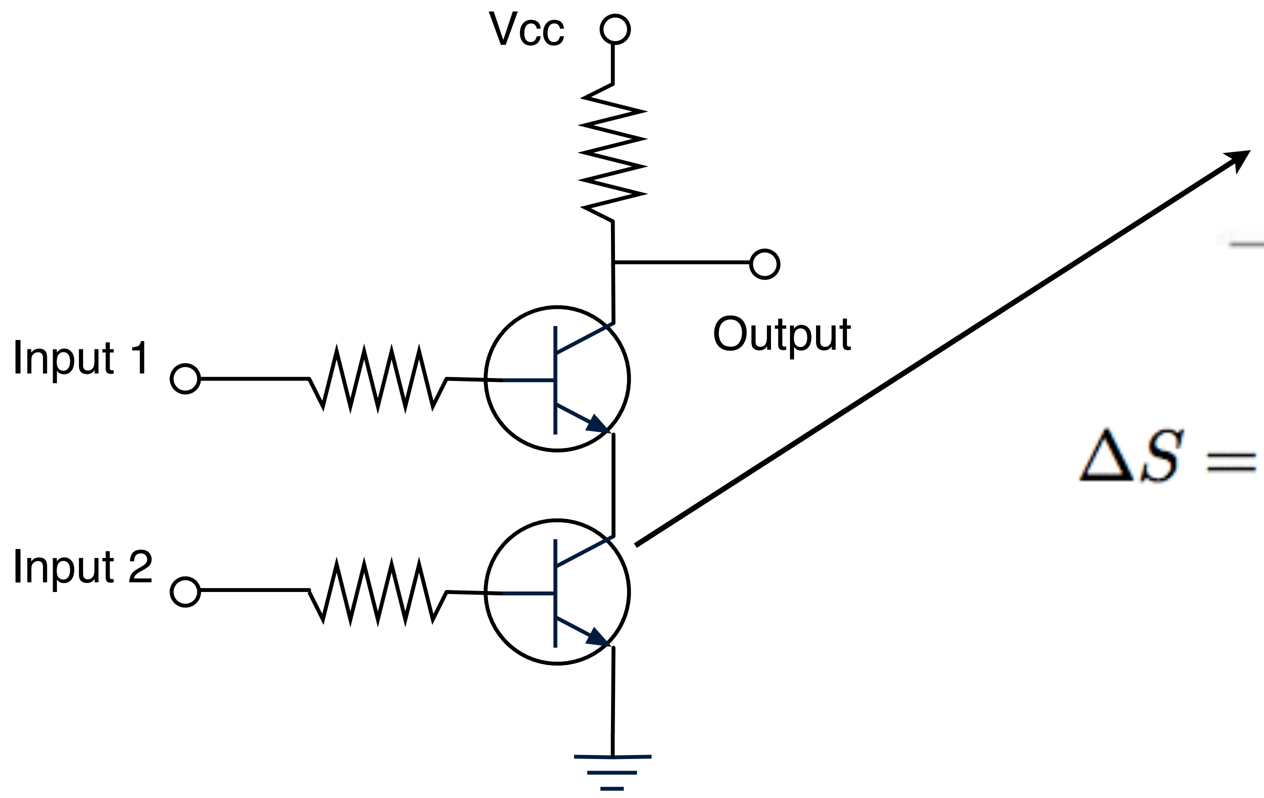


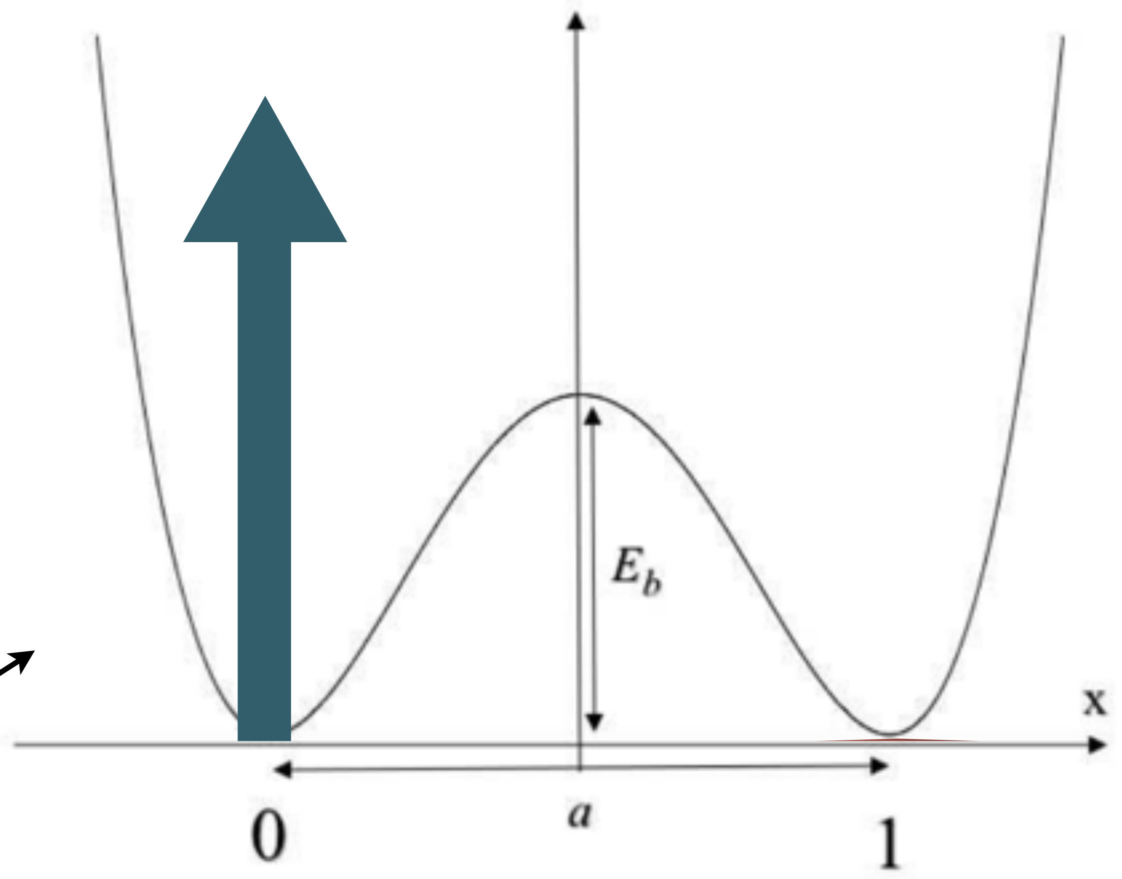




Sub-Landauer gate

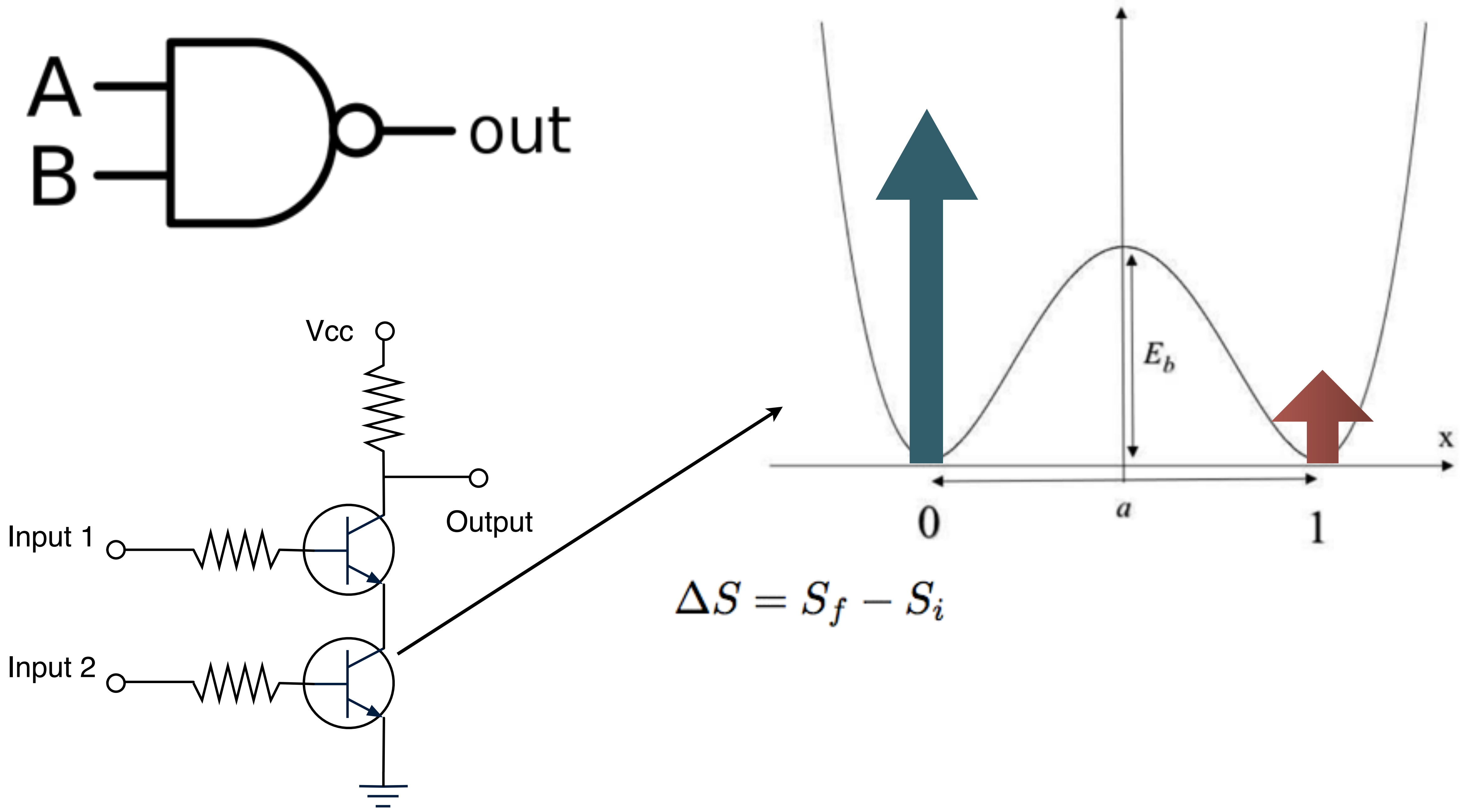






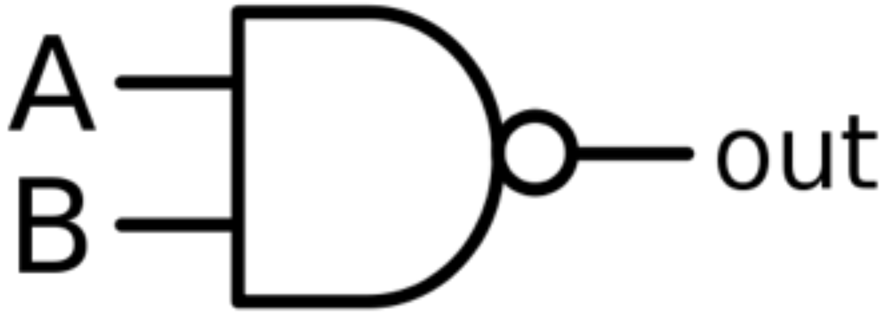
$$\Delta S = S_f - S_i = k_B(\ln(1) - \ln(2)) = -k_B \ln(2)$$

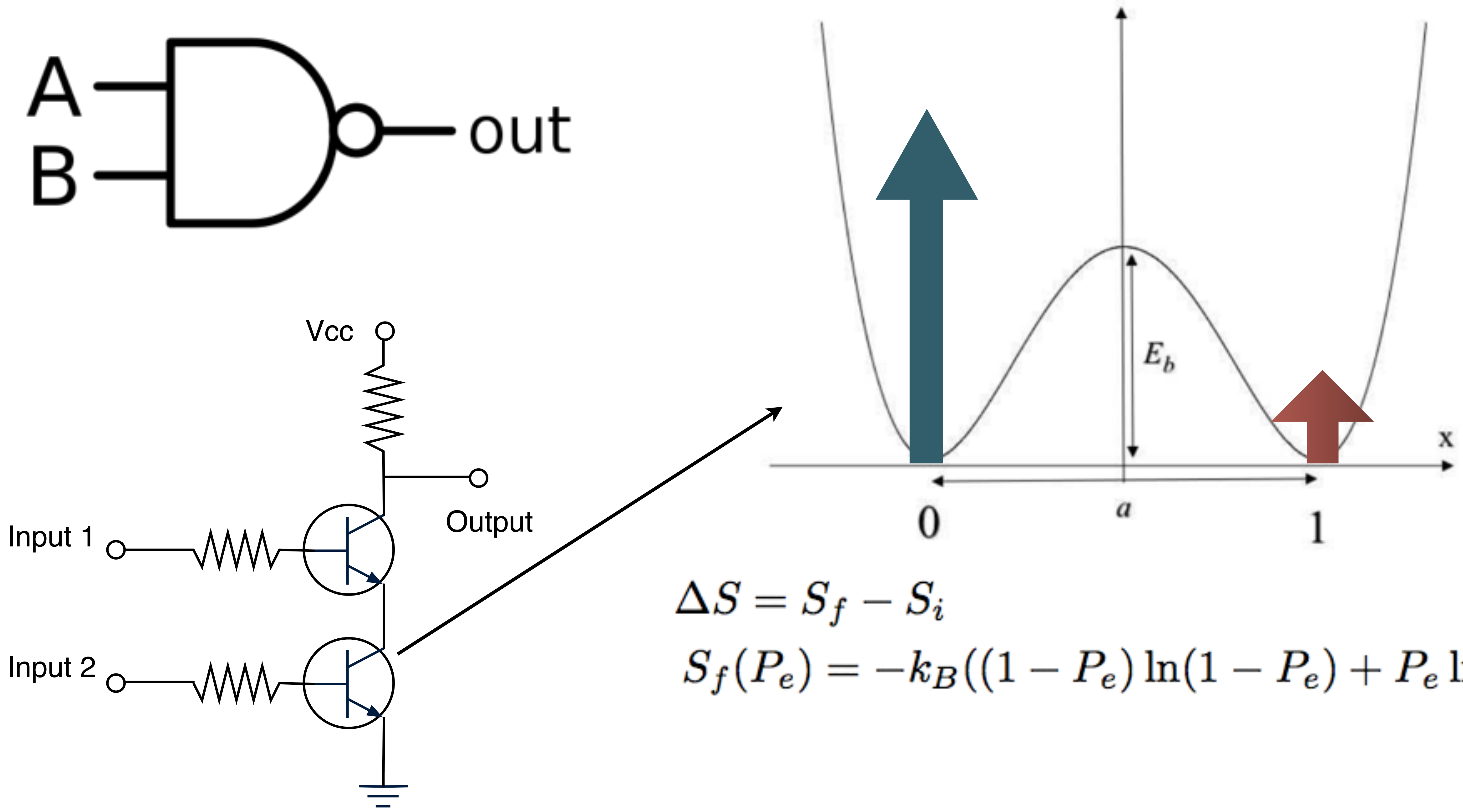
Sub-Landauer gate



$$\Delta S = S_f - S_i$$

Sub-Landauer gate

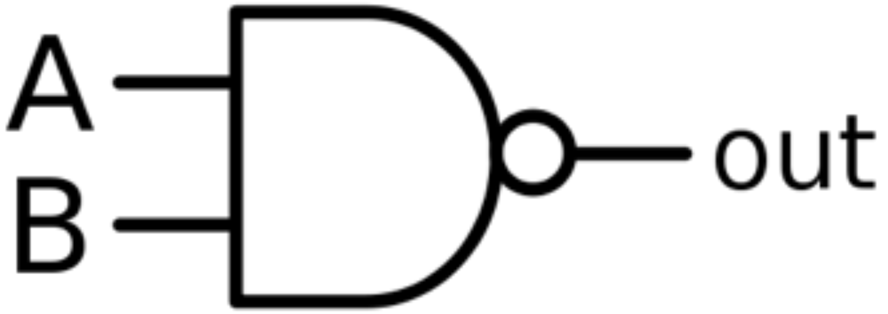


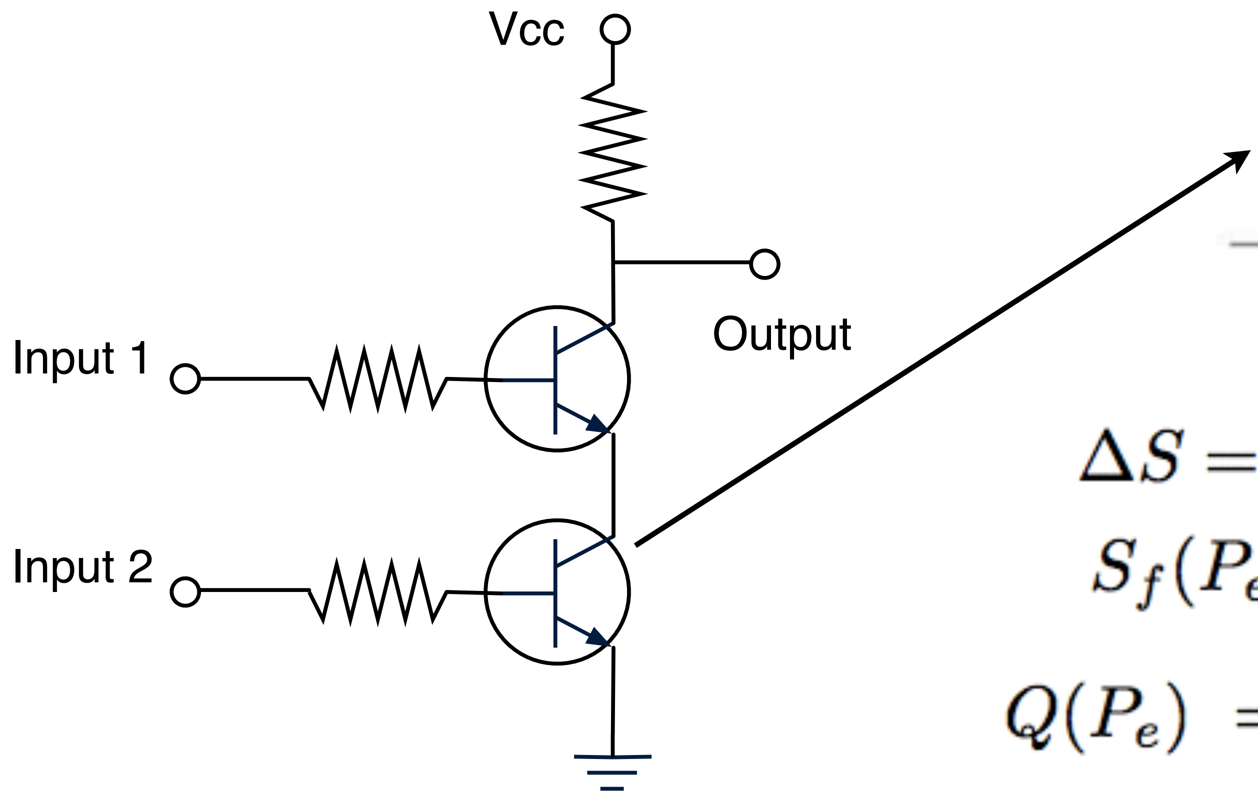


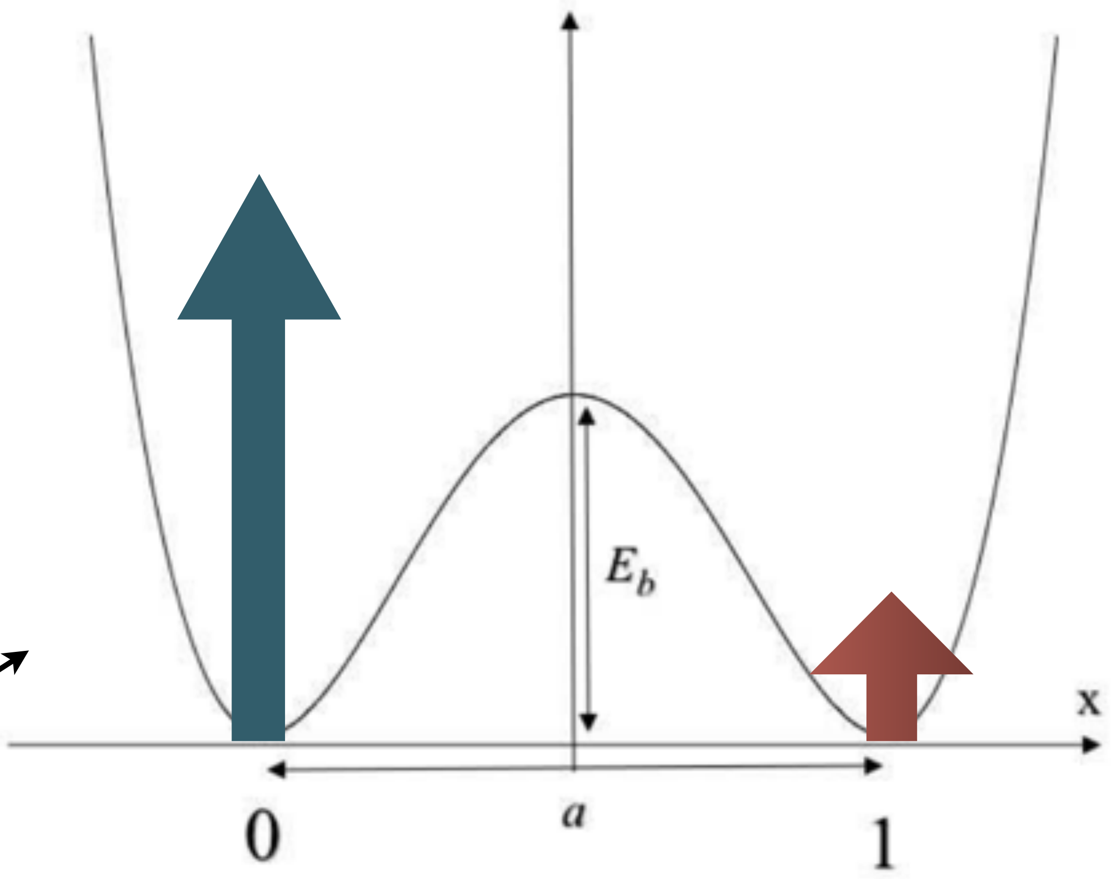
$$\Delta S = S_f - S_i$$

$$S_f(P_e) = -k_B((1 - P_e)\ln(1 - P_e) + P_e \ln(P_e))$$

Sub-Landauer gate







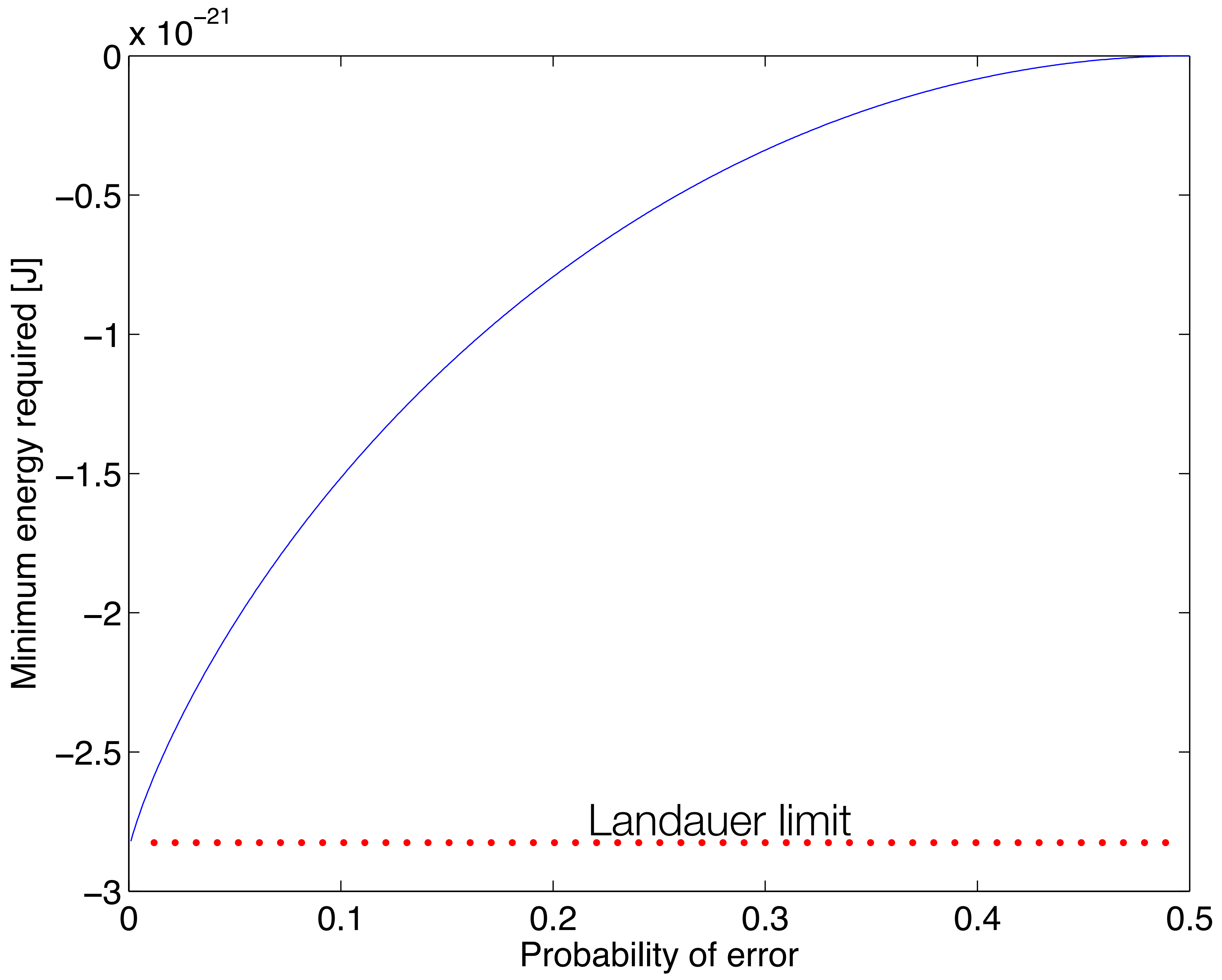
$$\Delta S = S_f - S_i$$

$$S_f(P_e) = -k_B((1-P_e)\ln(1-P_e) + P_e\ln(P_e))$$

$$\begin{aligned} Q(P_e) \;&=\; -k_BT((1-P_e)\ln(1-P_e) + P_e\ln(P_e)) + \\ &\quad -k_BT\ln(2) \end{aligned}$$

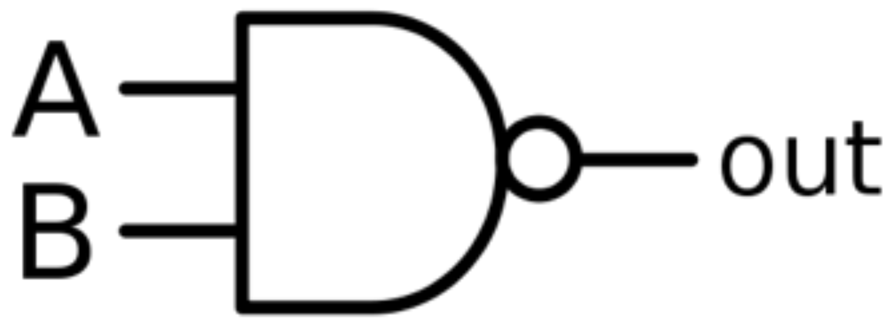
Beating the Landauer's limit by trading energy with uncertainty - L. Gammaitoni - arXiv:1111.2937 [cond-mat.mtrl-sci]

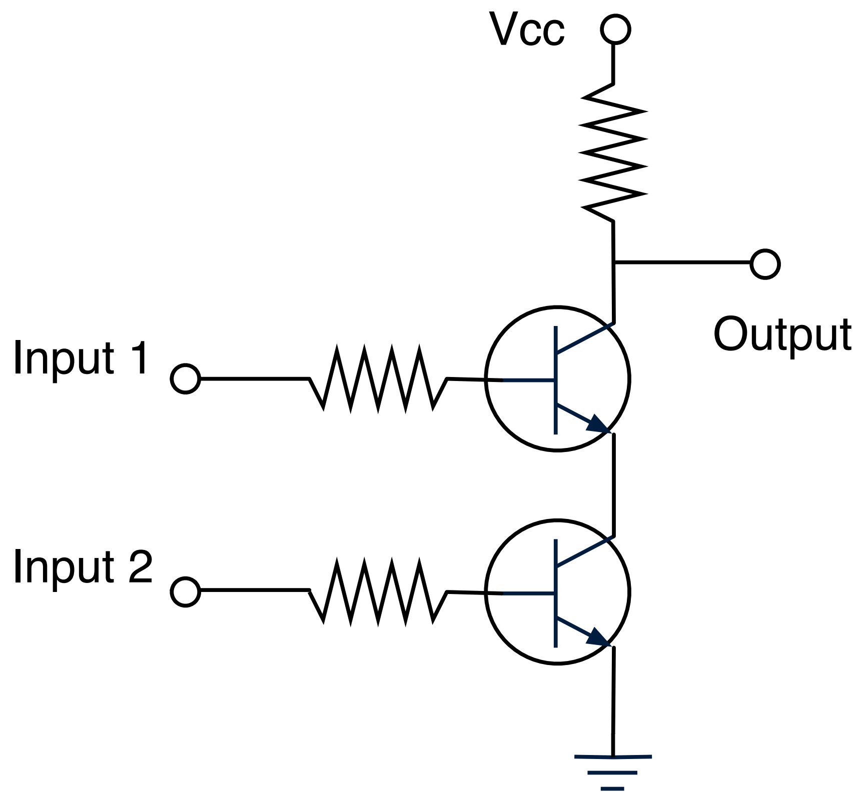
Sub-Landauer gate



Beating the Landauer's limit by trading energy with uncertainty - L. Gammaitoni - arXiv:1111.2937 [cond-mat.mtrl-sci]

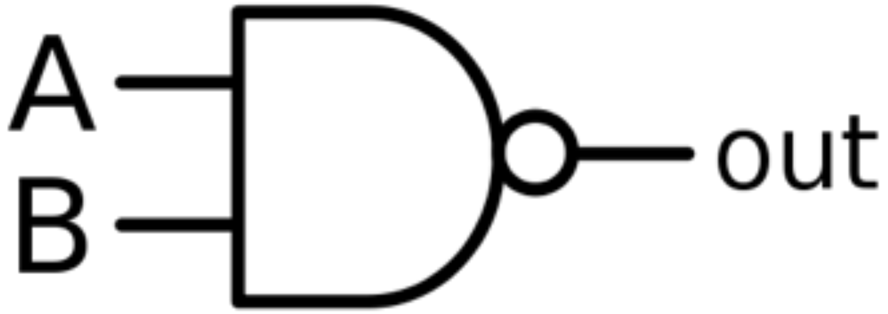
Sub-Landauer gate

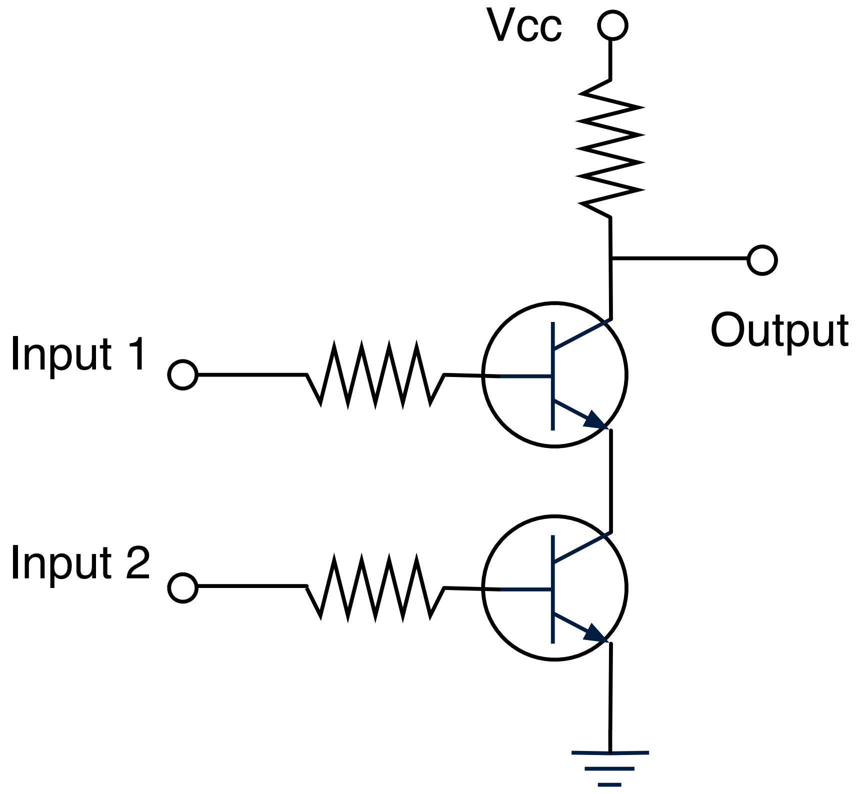


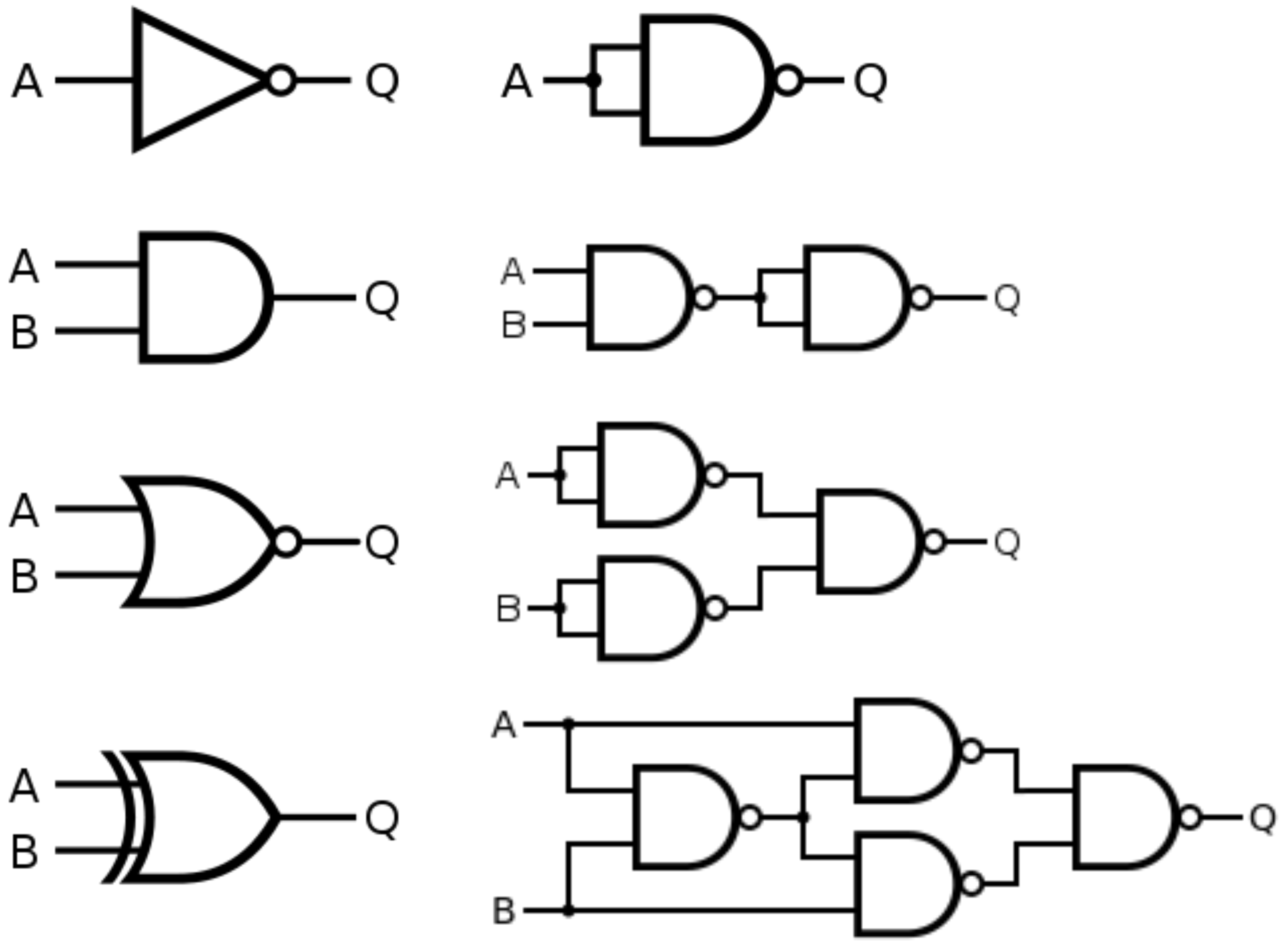


A	B	Q	Q error prob.
0	0	1	e^2
0	1	1	$2(e-e^2)$
1	0	1	$2(e-e^2)$
1	1	0	$2e-e^2$

Sub-Landauer gate



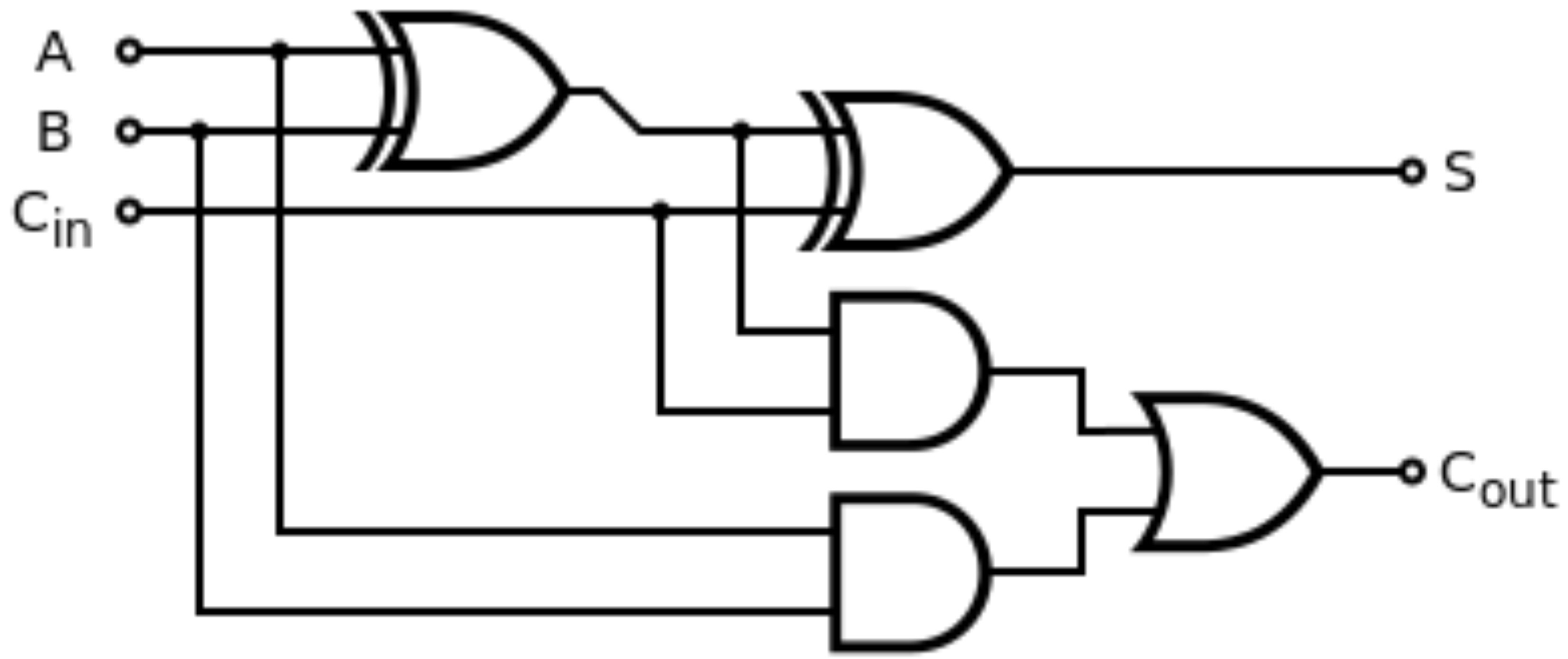




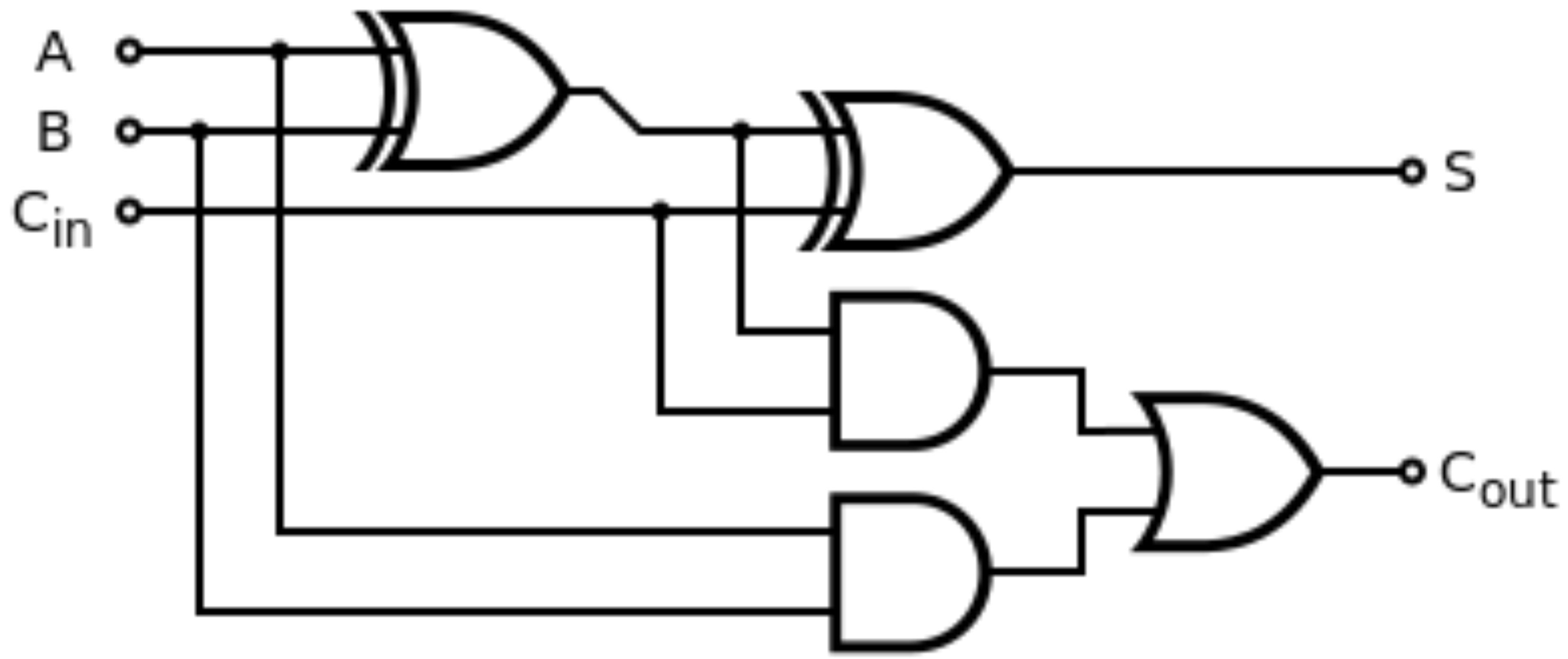
ALU simulator

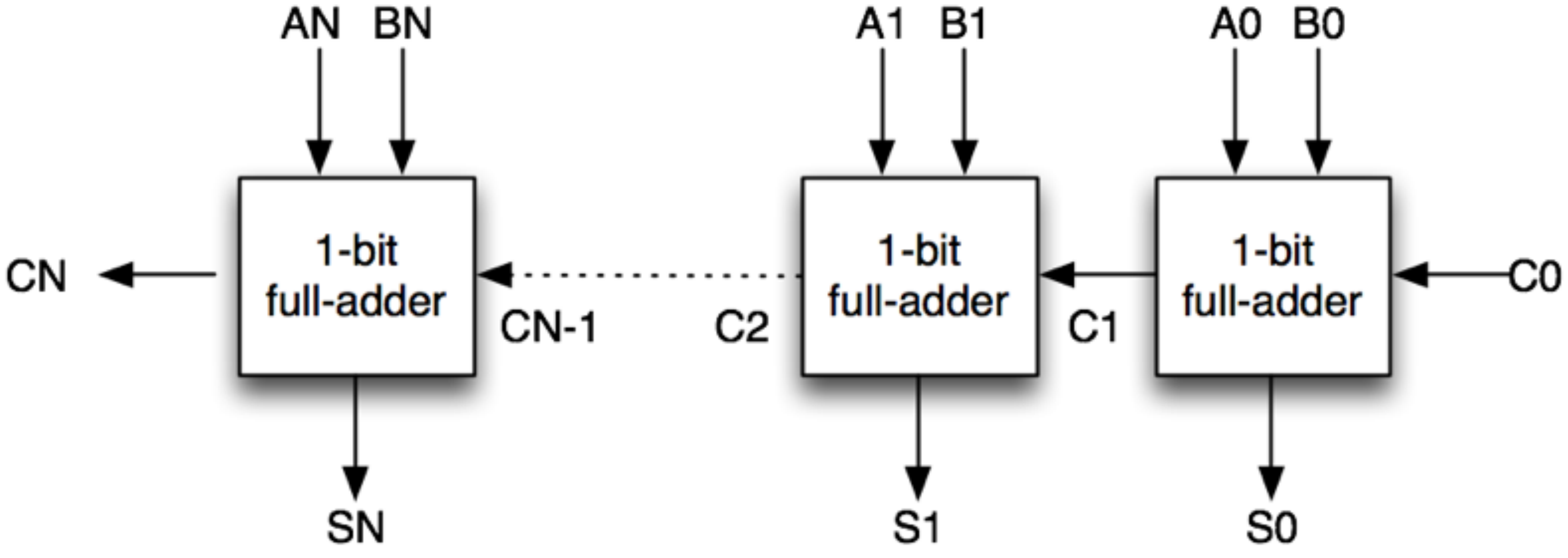
- ALU simulator developed in Java
- Based on NAND gates
- Implemented functional units:
    - Adder (ripple-carry)
    - Comparator (based on SN54/74LS85)

Full Adder

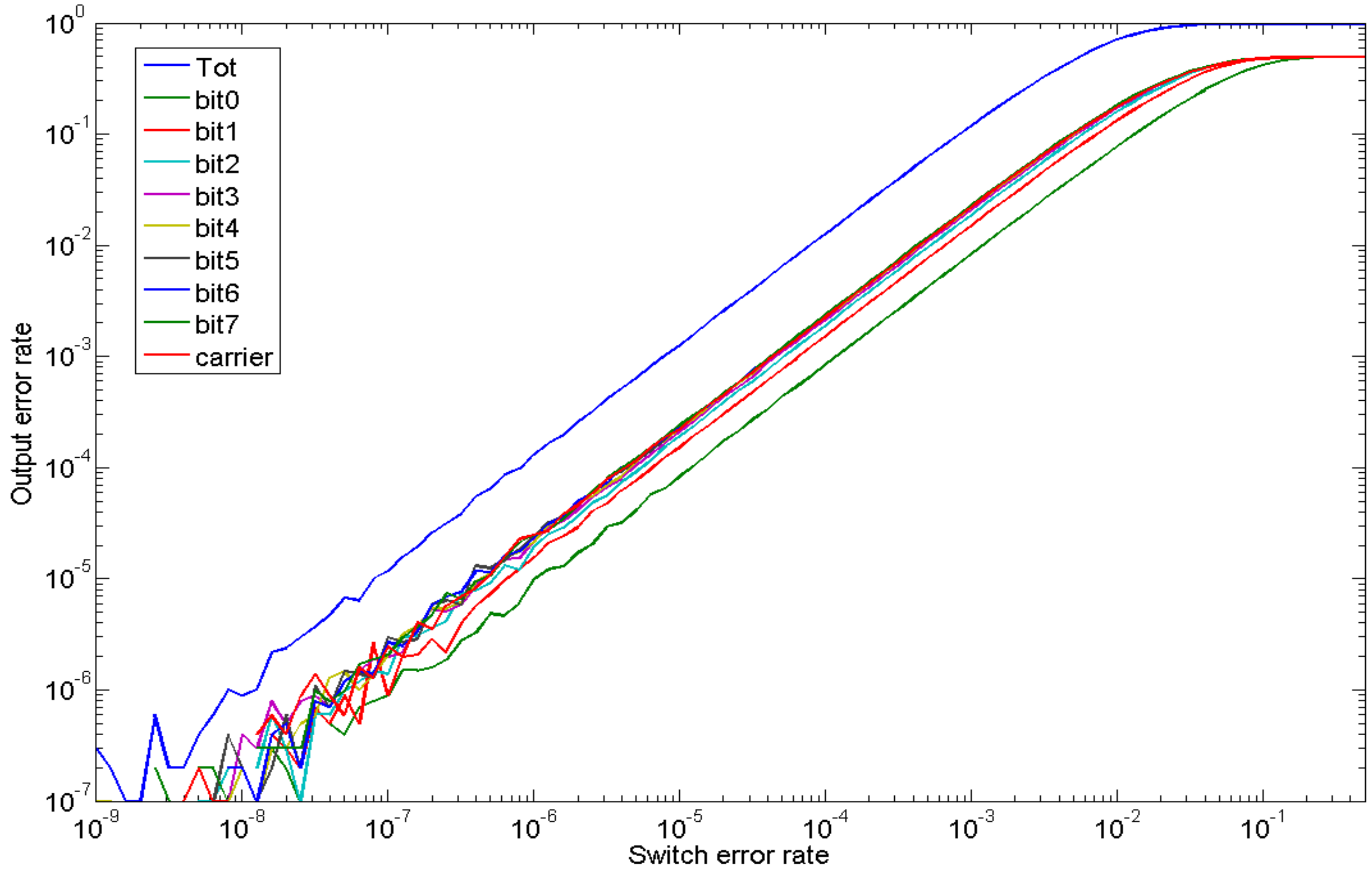


Full Adder


A
B
C_{in}
S
C_{out}


AN
BN
A1
B1
A0
B0
1-bit full-adder
1-bit full-adder
1-bit full-adder
CN
CN-1
C2
C1
C0
SN
S1
S0

Full Adder



Tot
bit0
bit1
bit2
bit3
bit4
bit5
bit6
bit7
carrier
Output error rate
Switch error rate
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-9}
10^{-8}
10^{-7}
10^{-6}
10^{-5}
10^{-4}
10^{-3}
10^{-2}
10^{-1}

Sorting: definition

Process of taking a list of objects with a linear ordering

$(a_1, a_2, \cdots, a_{n-1}, a_n)$

and output a permutation of the list

$(a_{k1}, a_{k2}, \cdots, a_{kn-1}, a_{kn})$

such that

$a_{k1} \leqslant a_{k2} \leqslant \cdots \leqslant a_{kn-1} \leqslant a_{kn}$

Usually we're interested in sorting a list of “records” in order by some field, however, without loss of generality, this is equal to sorting values

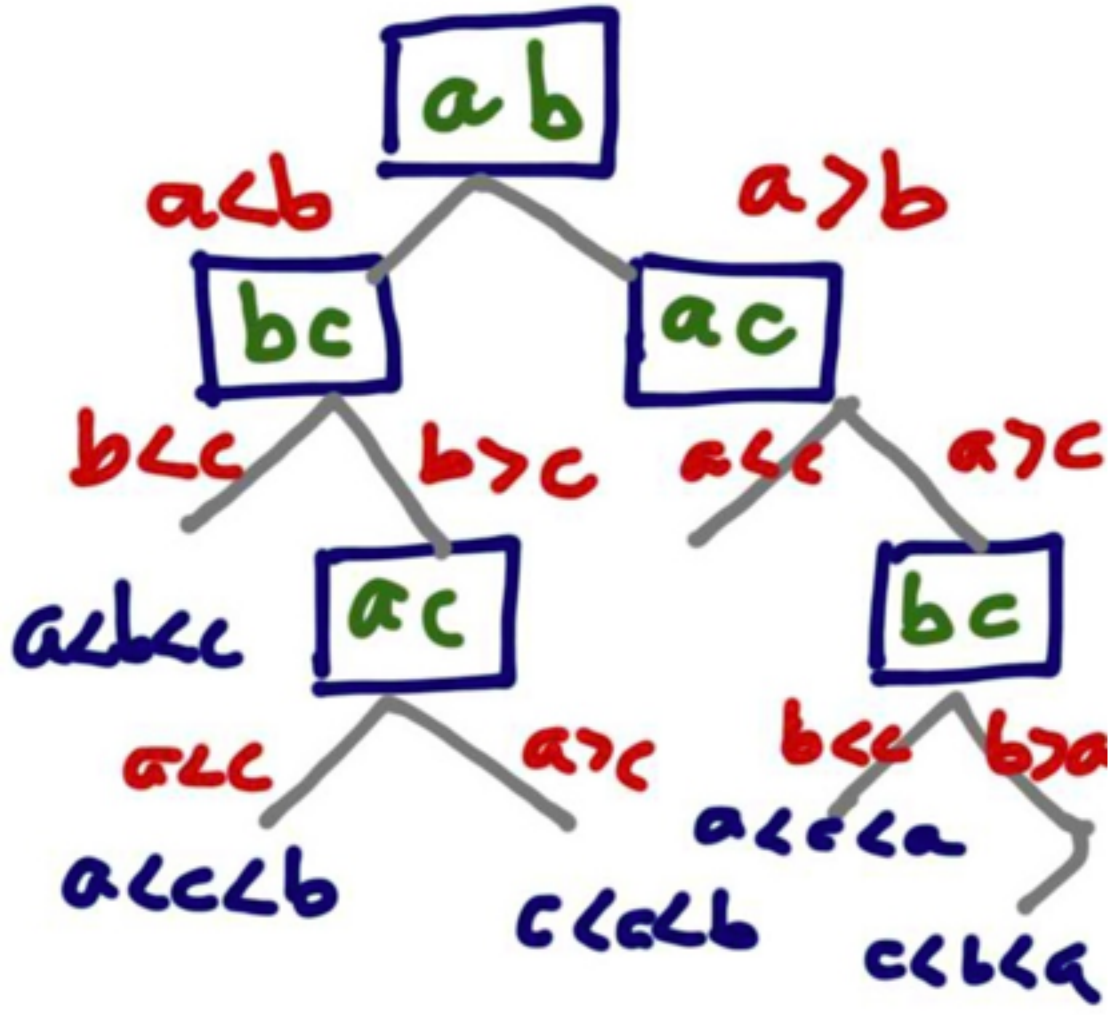
Sorting: limits

- In general no sort algorithms, based on comparison, can have a run time better than *n log(n)*

# leafs: n! permutation

tree height: *log(n!) ~ n log(n)*

***n log(n)* comparison**



- However under some circumstance there are algorithms that run in linear time

Sorting algorithms performances evaluation

	Time			
Name	**Best**	**Average**	**Worst**	**Memory**
Selection	n	n	n	1
Quick	$n\log n$	$n\log n$	n	$\log n$
Merge	$n\log n$	$n\log n$	$n\log n$	n
Insertion	n	n	n	1
Bubble	n	n	n	1

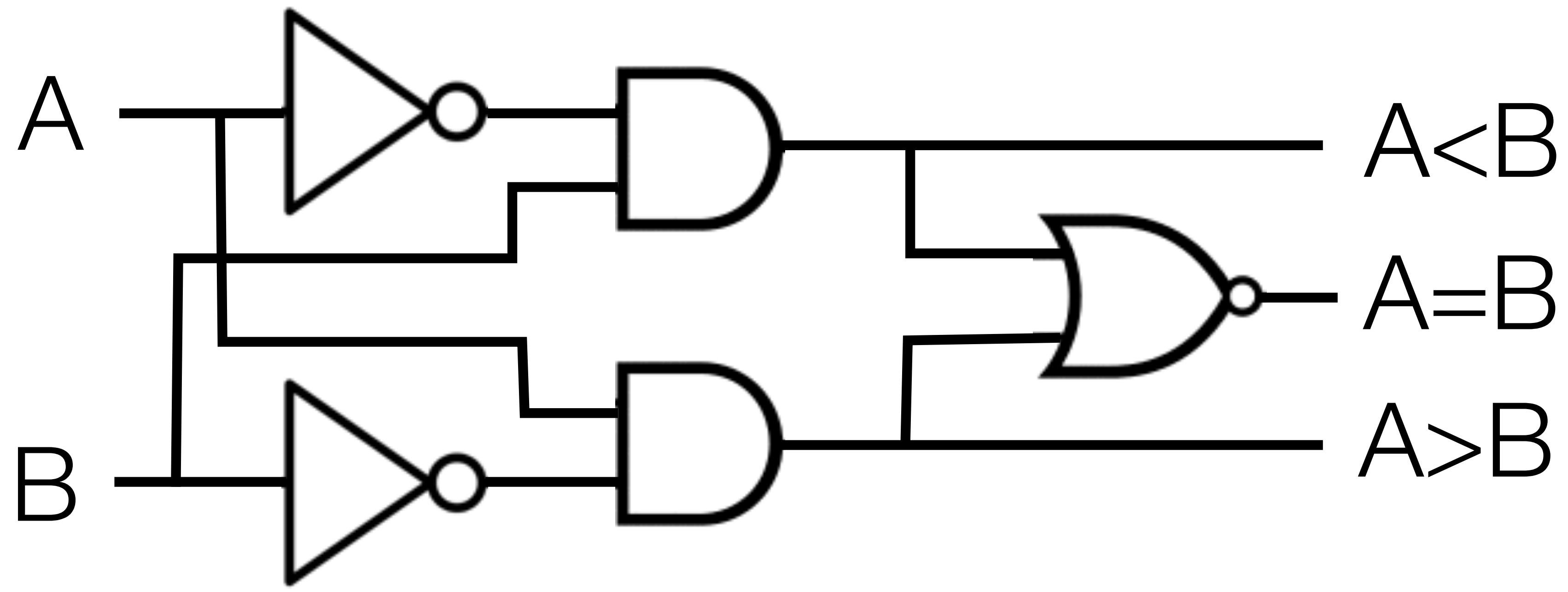
Relevant time operations:

- # of Comparisons
- # of read/write operations

Digital Comparator

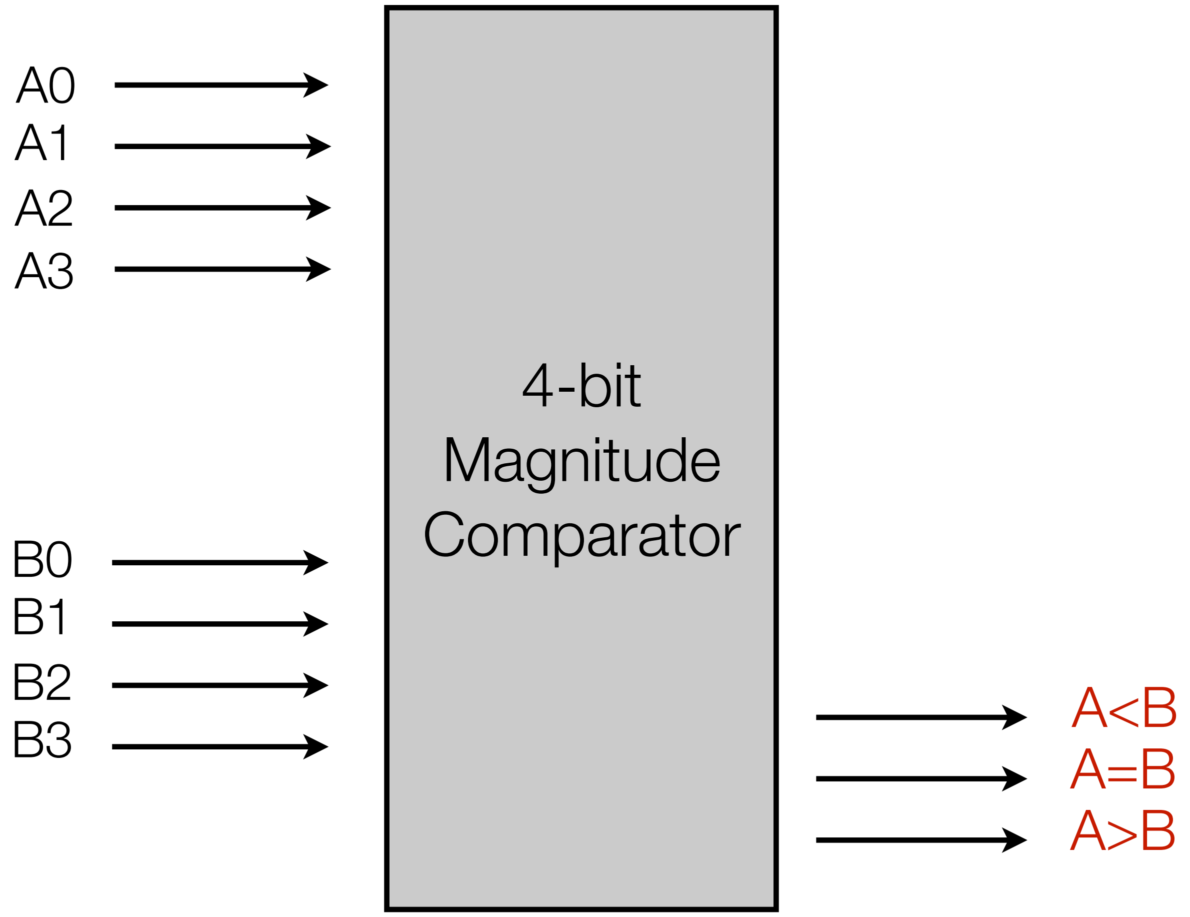
- The purpose of a Digital Comparator is to compare a set of variables or unknown numbers, for example A (A1, A2, A3, ... An, etc) against that of a constant or unknown value such as B (B1, B2, B3, ... Bn, etc) and produce an output condition or flag depending upon the result of the comparison.
  - **Identity Comparator**: digital comparator with only one output terminal for when **A = B**
  - **Magnitude Comparator**: digital comparator that has three output terminals, one each for equality, **A = B** greater than, **A > B** and less than **A < B**

1-bit Magnitude Comparator

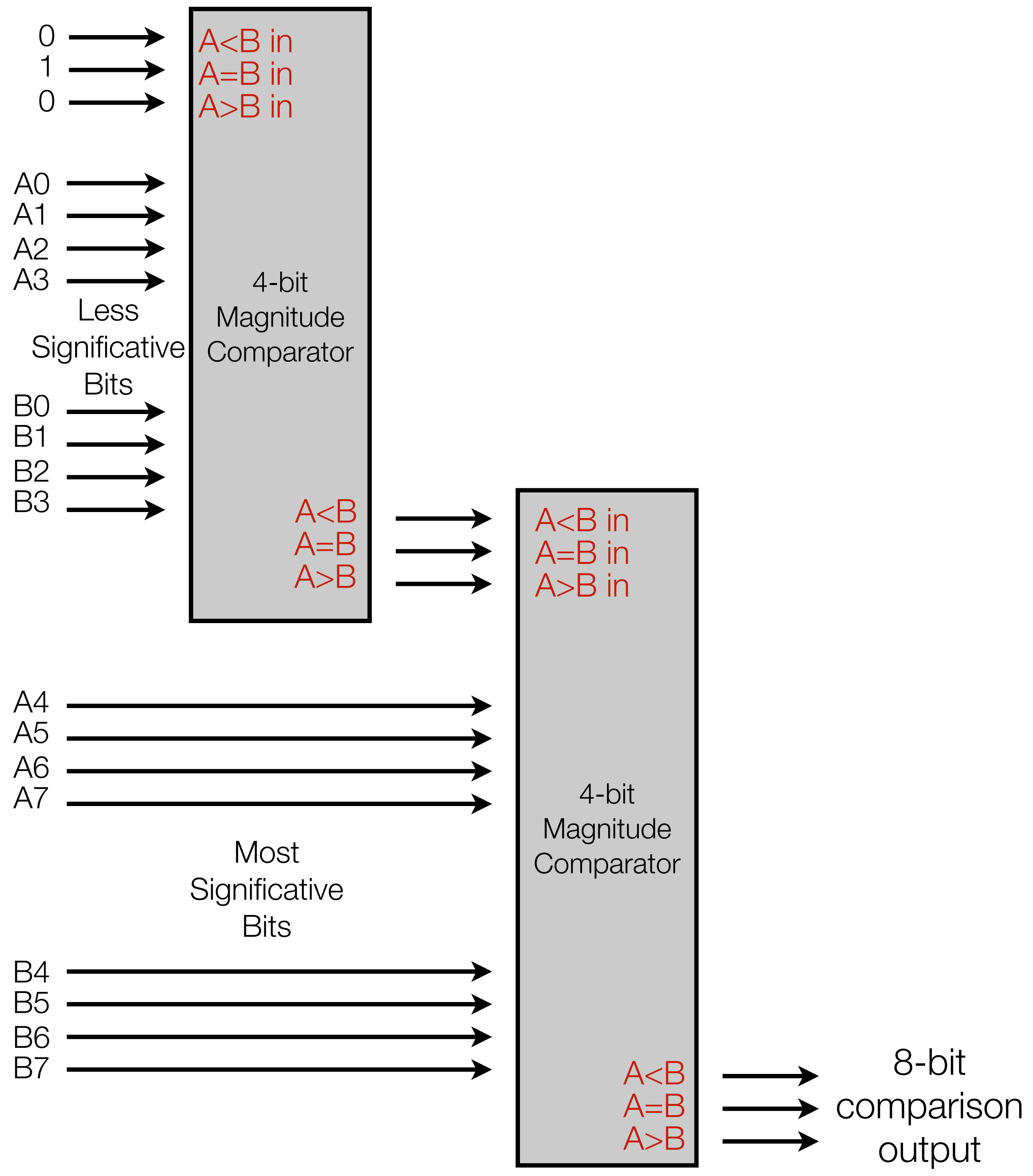


A	B	A<B	A=B	A>B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

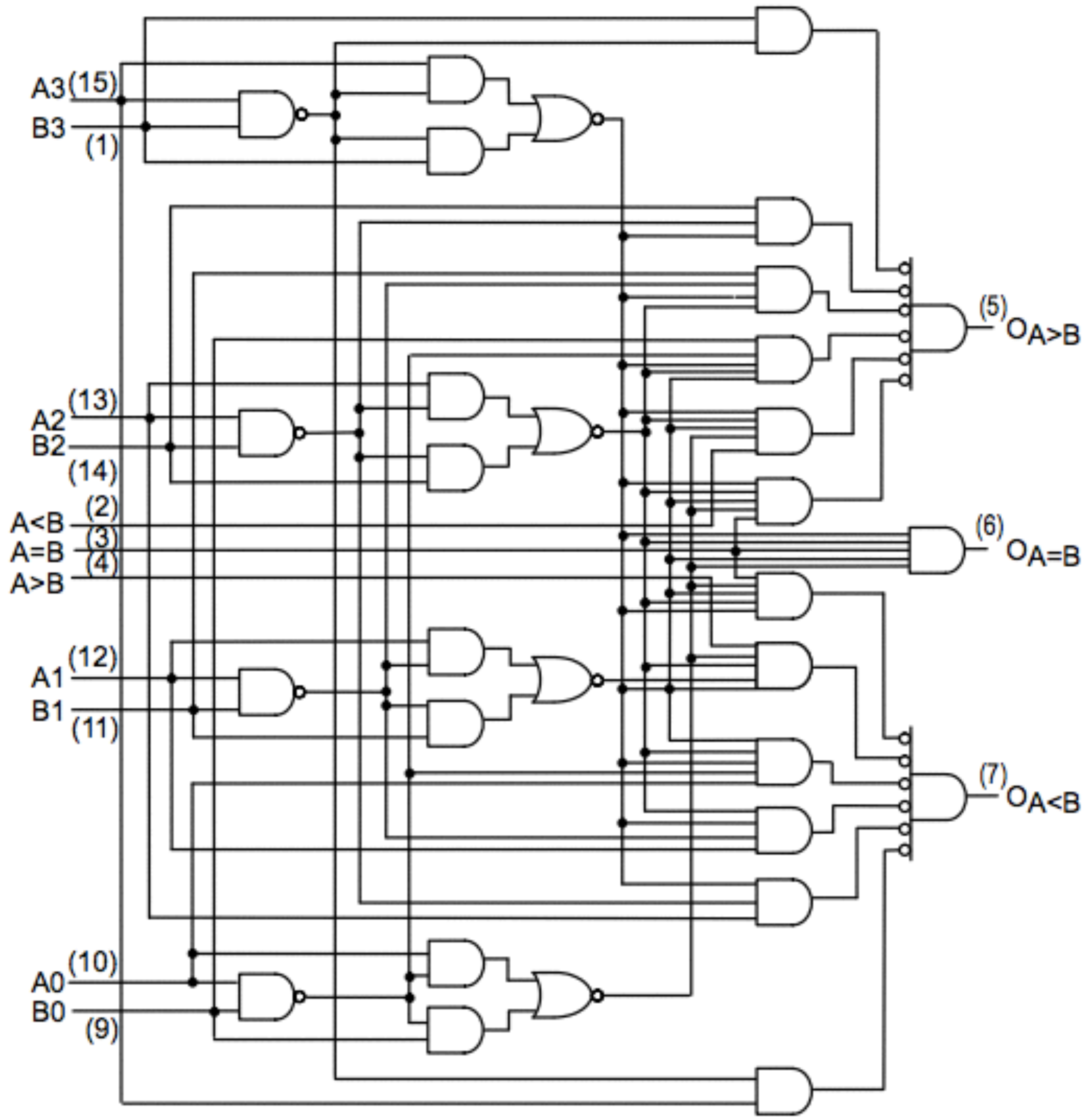
4-bits Magnitude Comparator



8-bits Magnitude Comparator



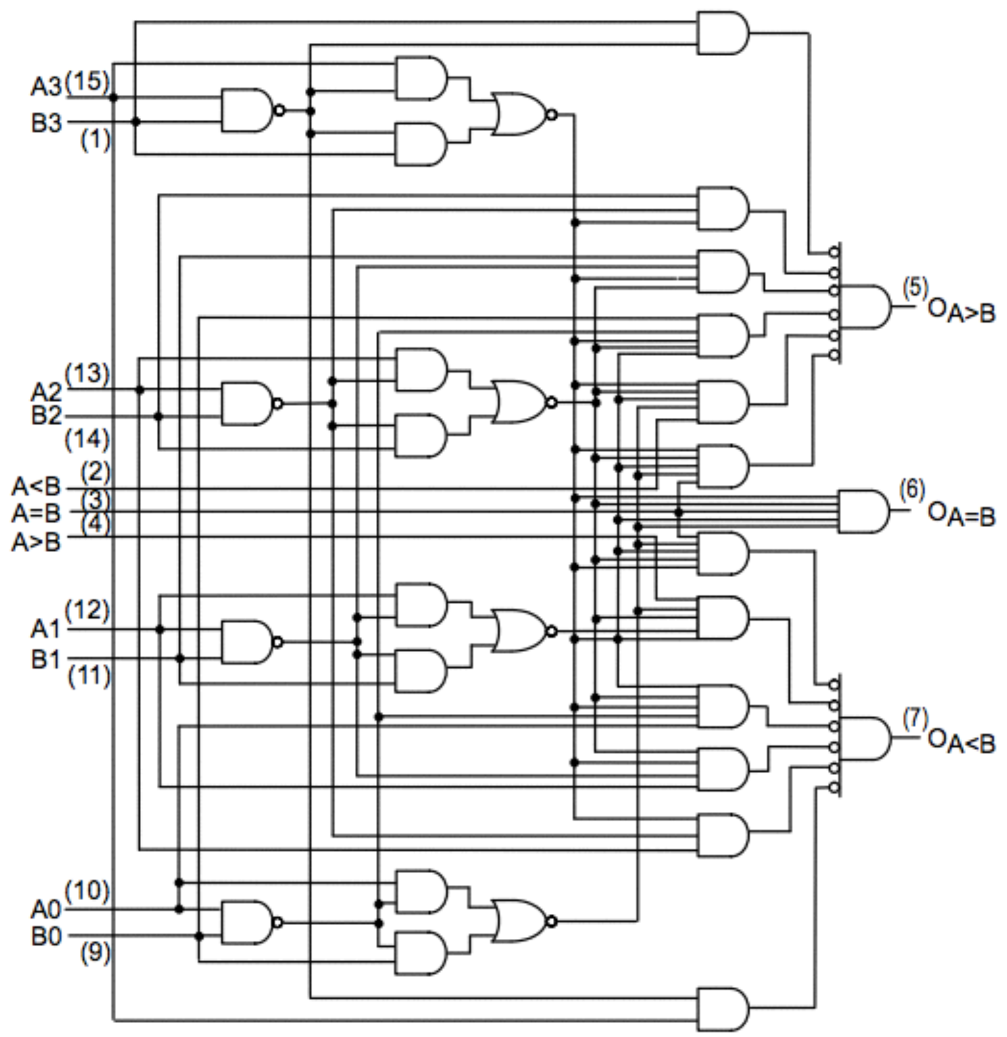
4-bits Magnitude Comparator - SN54/74LS85



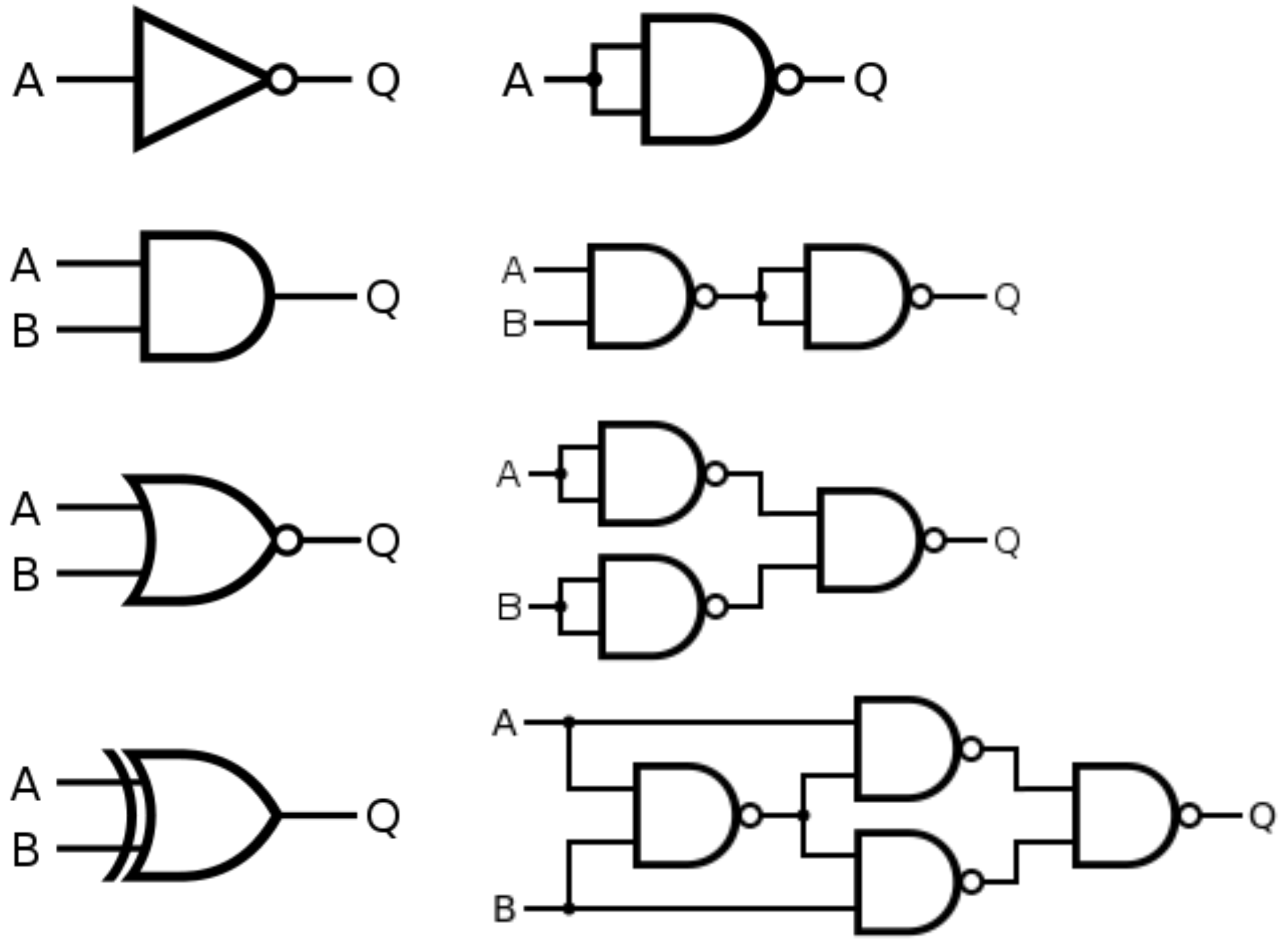
SN54/74LS85 data-sheet

16-bits Magnitude Comparator with NAND gates

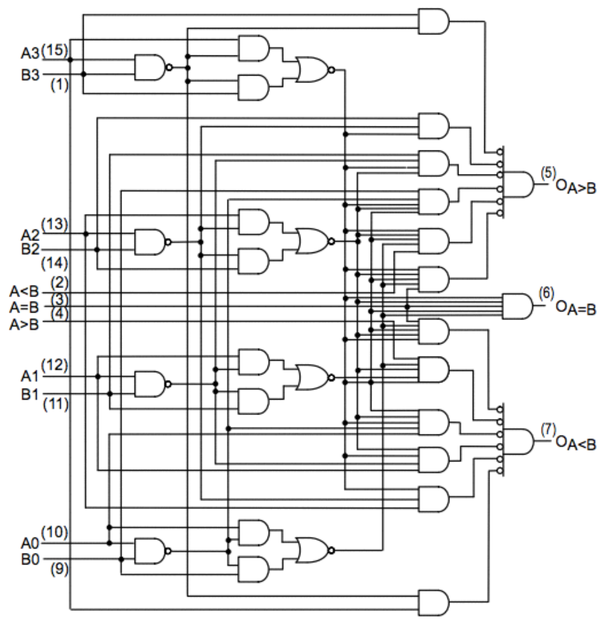
4x



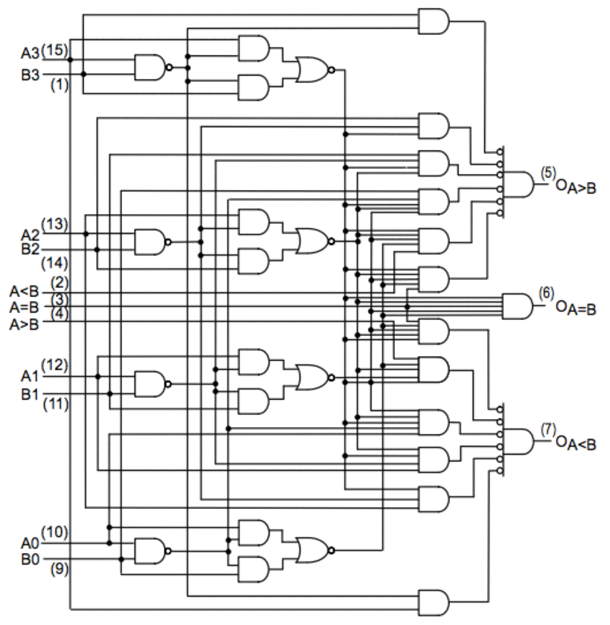
Traditional gates replaced with NANDs



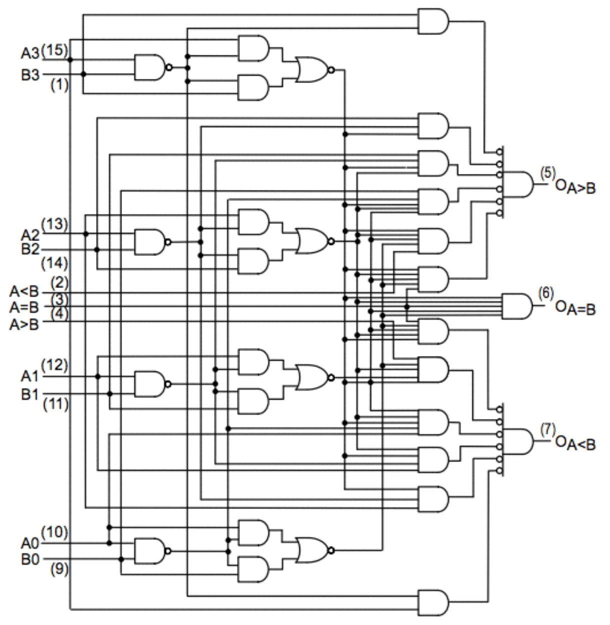
Control parameters



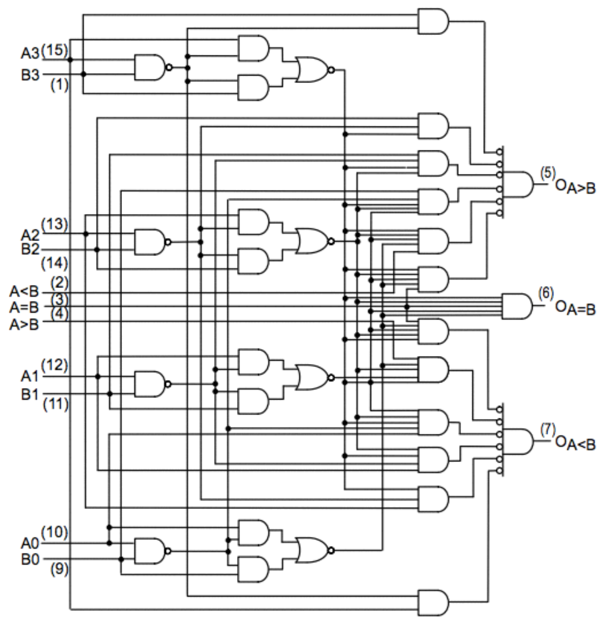
Switch error rate
bits 0-3



Switch error rate
bits 4-7



Switch error rate
bits 8-11

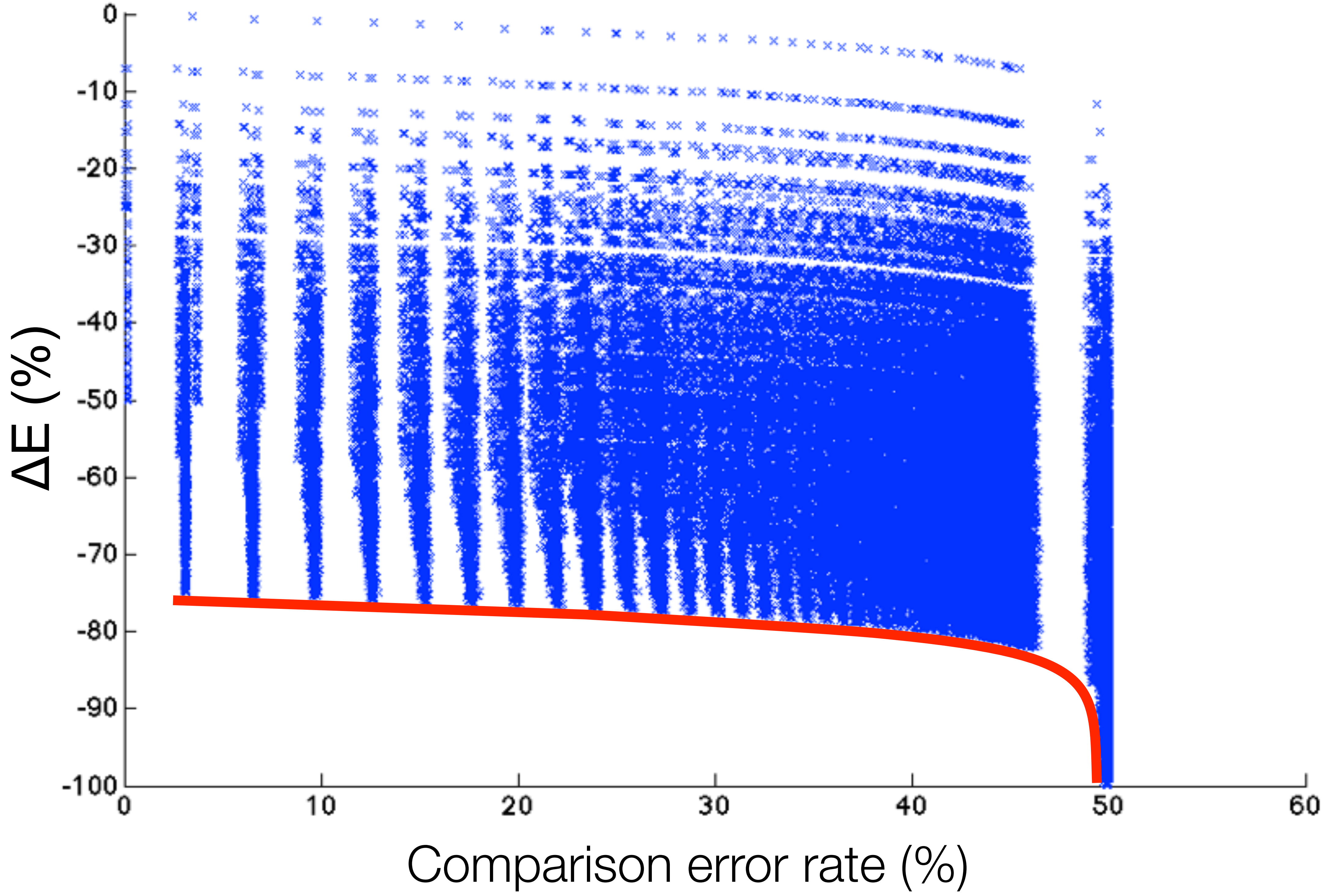


Switch error rate
bits 12-15

Monte Carlo simulations

- Varying the switch error rate of each comparator to compute:
    - Comparison error probability given switch error probability of each comparator
    - Computed minimum energy required by each comparison given switch error probability for each comparator in respect to the Landauer limit
- Computed noise level after sorting procedure for different sorting algorithms varying comparison error rate

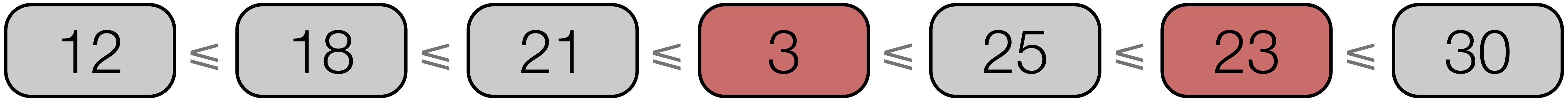
Comparison error rate vs energy saving



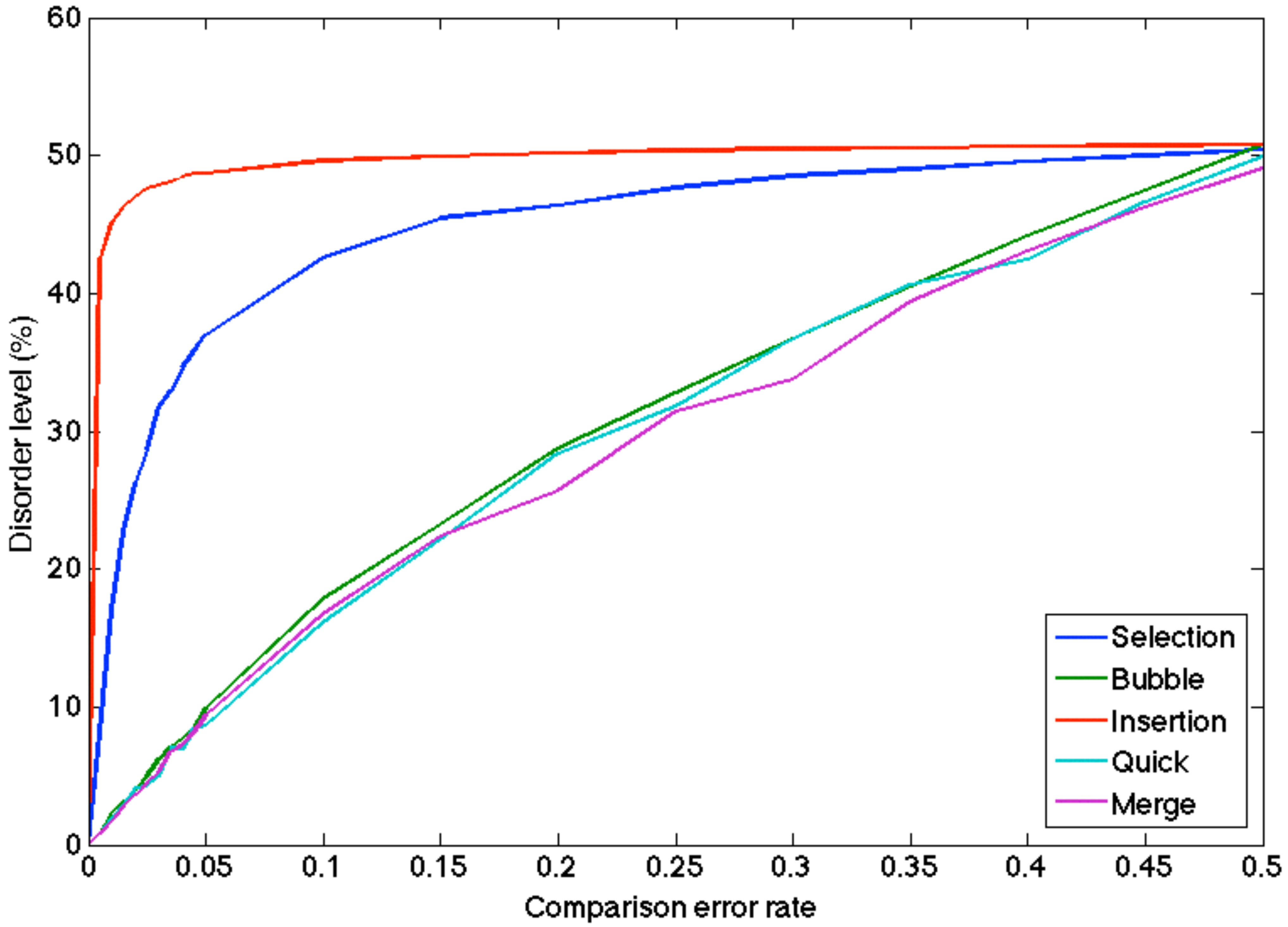
Sorting with uncertainty: Trading accuracy with energy saving, I. Neri et al., NANOENERGY 2013

Sorting performances metrics: output noise level

- Metrics that returns the noise in the output sequence: it is a measure of how the sequence is far from being ordered
- For each element on the output array check how many pairs of elements are in wrong order in respect to the perfectly ordered sequence
- noise level: *#pairs in wrong order* over *#total number of pairs*

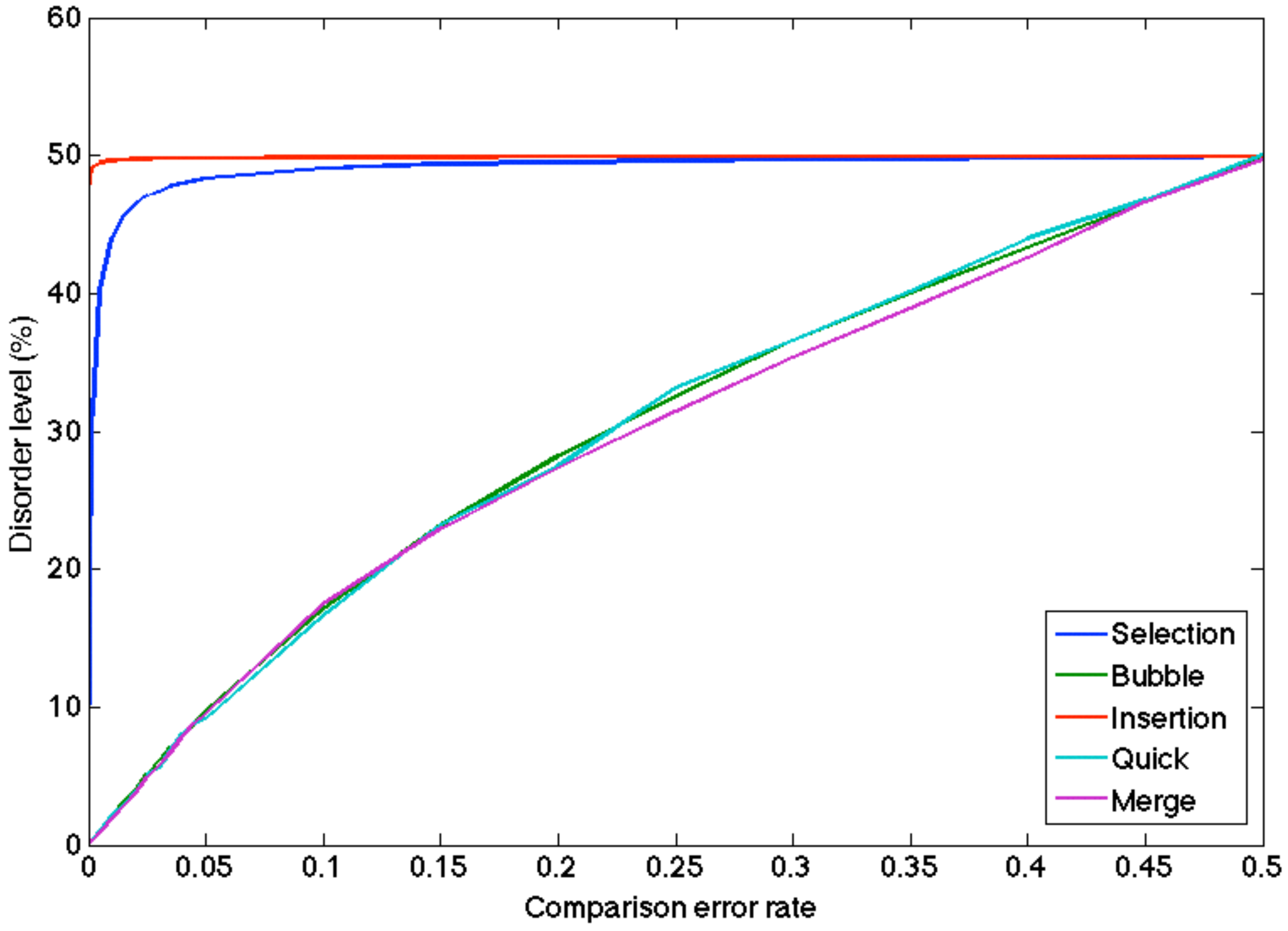


Results: input sequence #500 - uniform 0-2^{16}



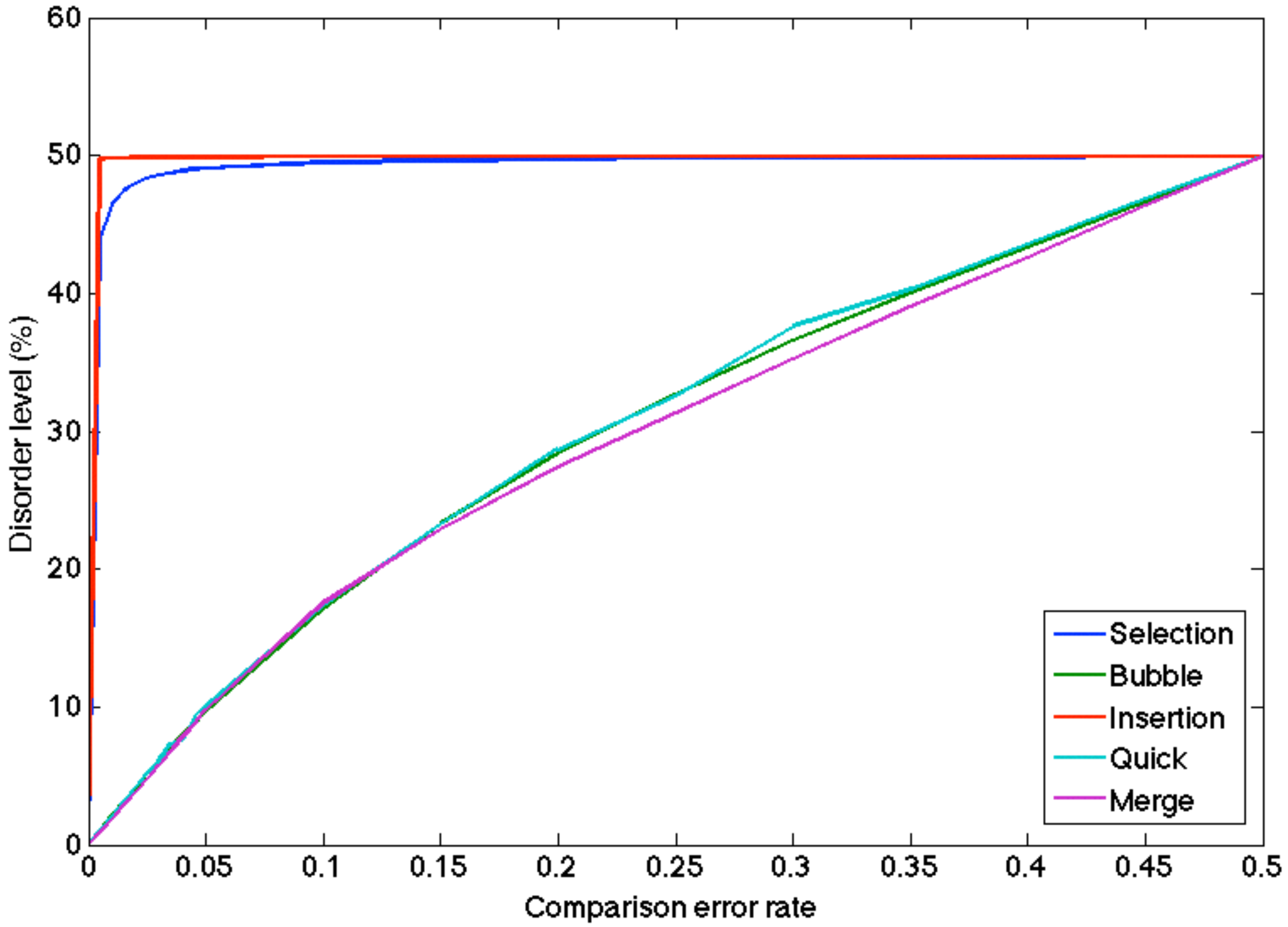
Sorting with uncertainty: Trading accuracy with energy saving, I. Neri et al., NANOENERGY 2013

Results: input sequence #10000 - uniform 0-2^{16}



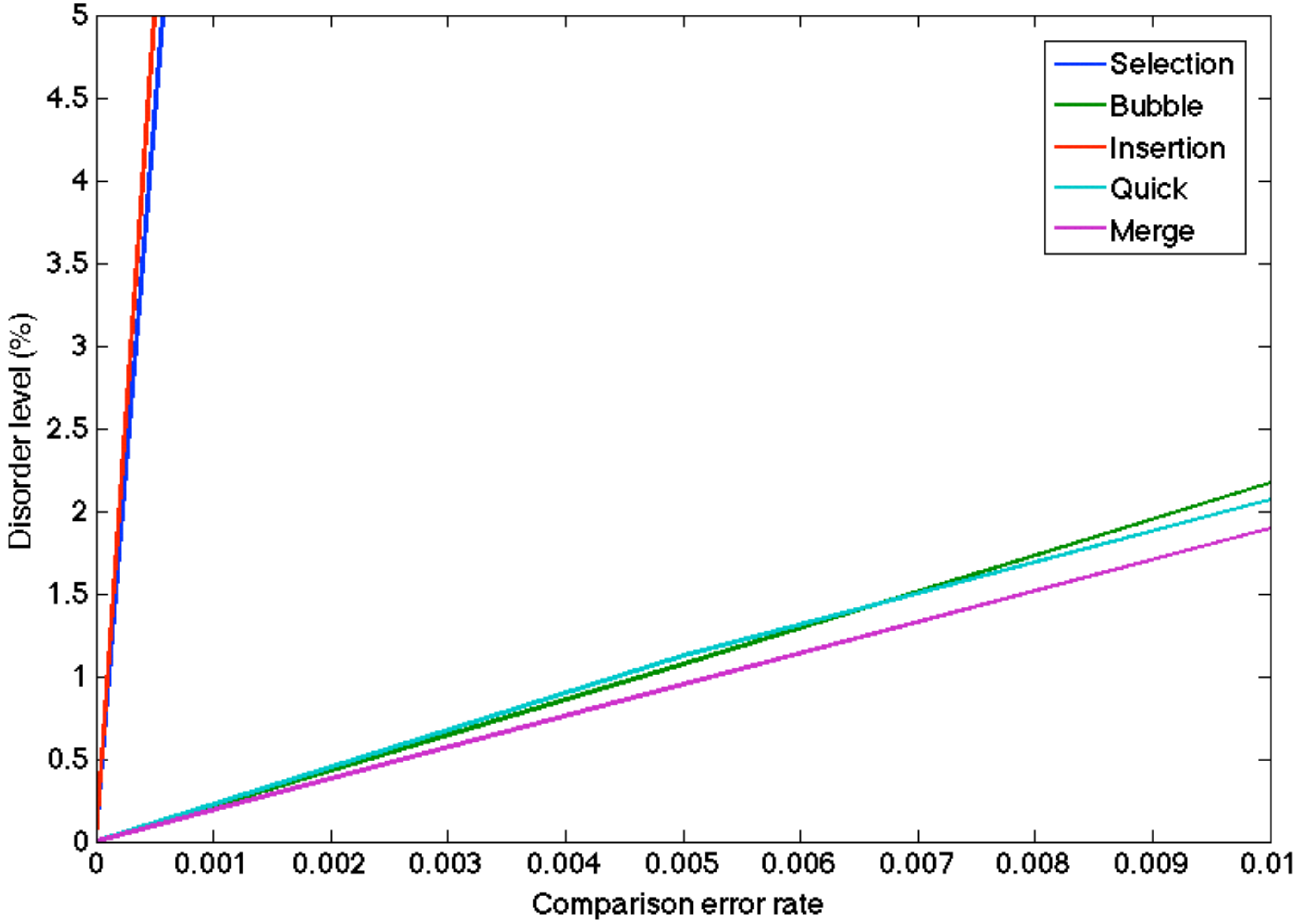
Sorting with uncertainty: Trading accuracy with energy saving, I. Neri et al., NANOENERGY 2013

Results: input sequence #20000 - uniform 0-2^{16}



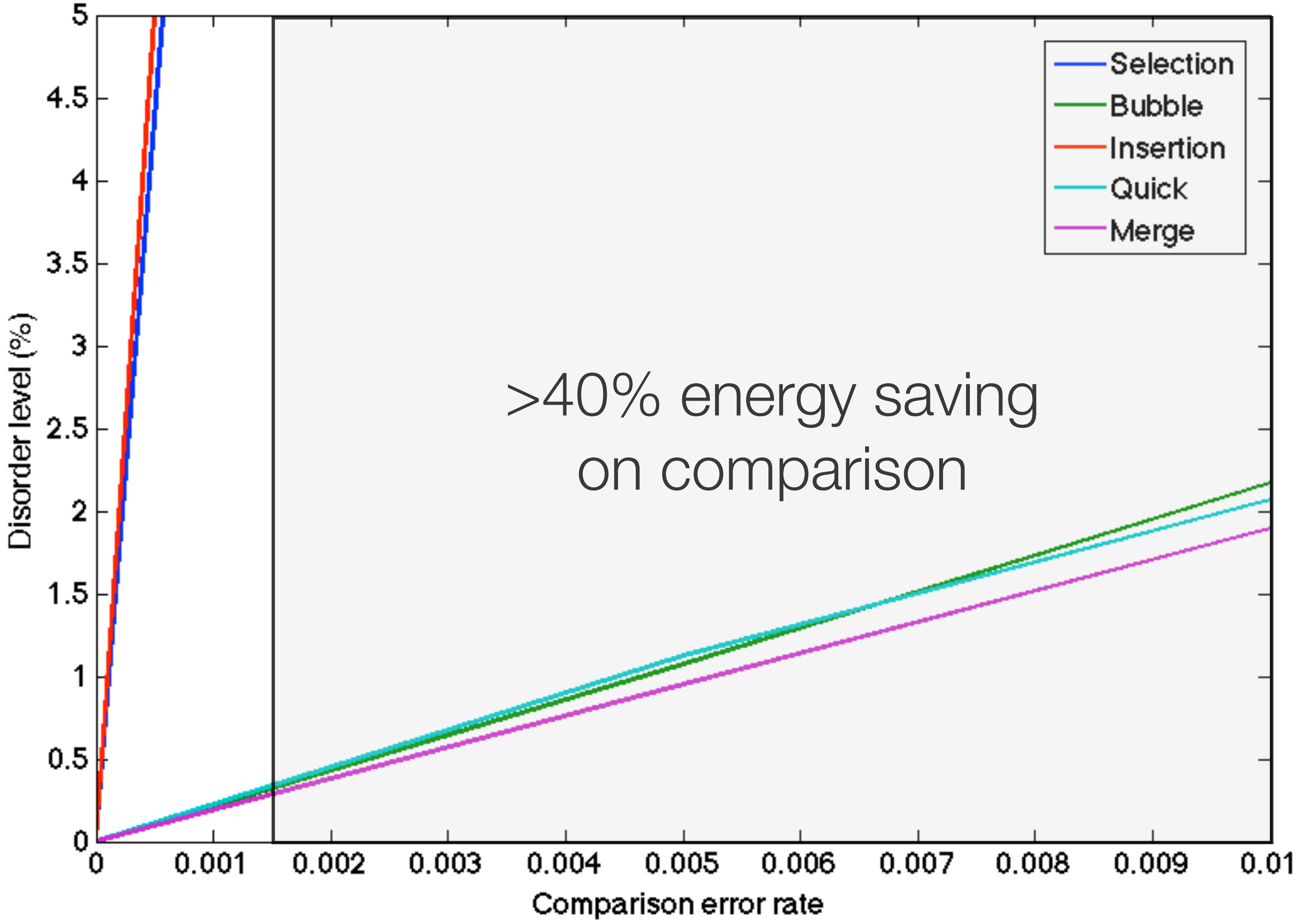
Sorting with uncertainty: Trading accuracy with energy saving, I. Neri et al., NANOENERGY 2013

Results: input sequence #20000 - uniform 0-2^{16}



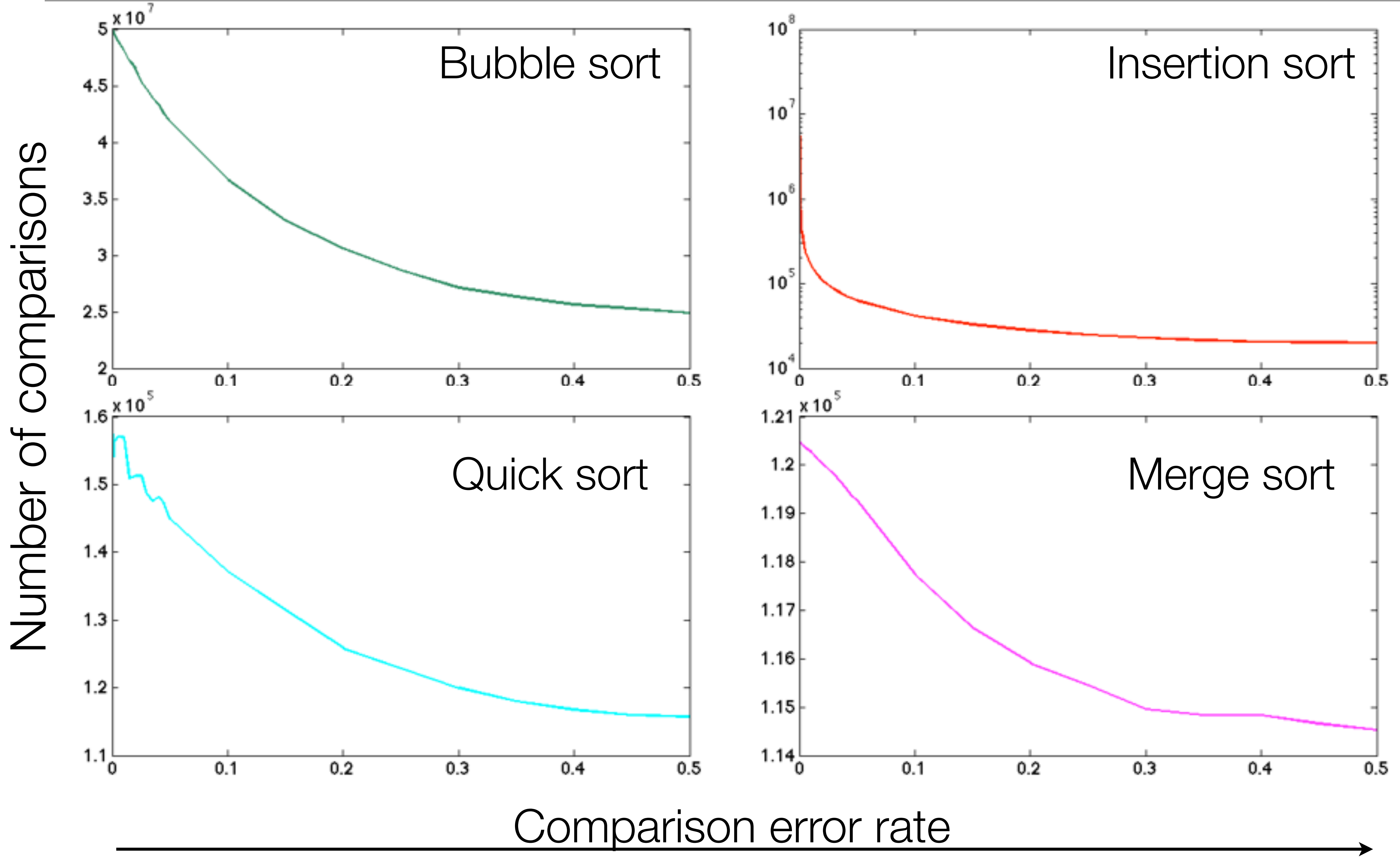
Sorting with uncertainty: Trading accuracy with energy saving, I. Neri et al., NANOENERGY 2013

Results: input sequence #20000 - uniform 0-2^{16}

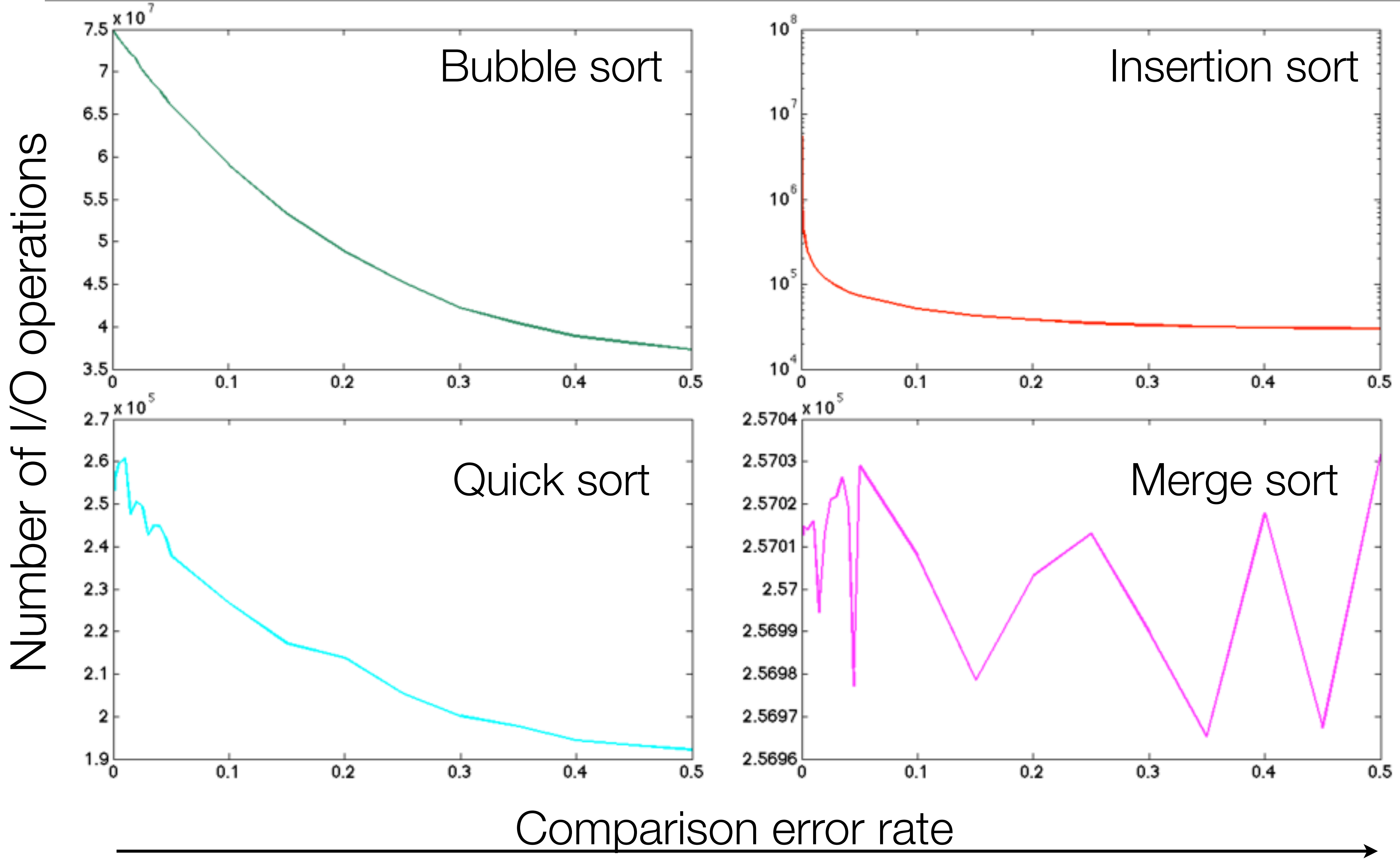


Sorting with uncertainty: Trading accuracy with energy saving, I. Neri et al., NANOENERGY 2013

Results: seq. #10000 - number of comparisons



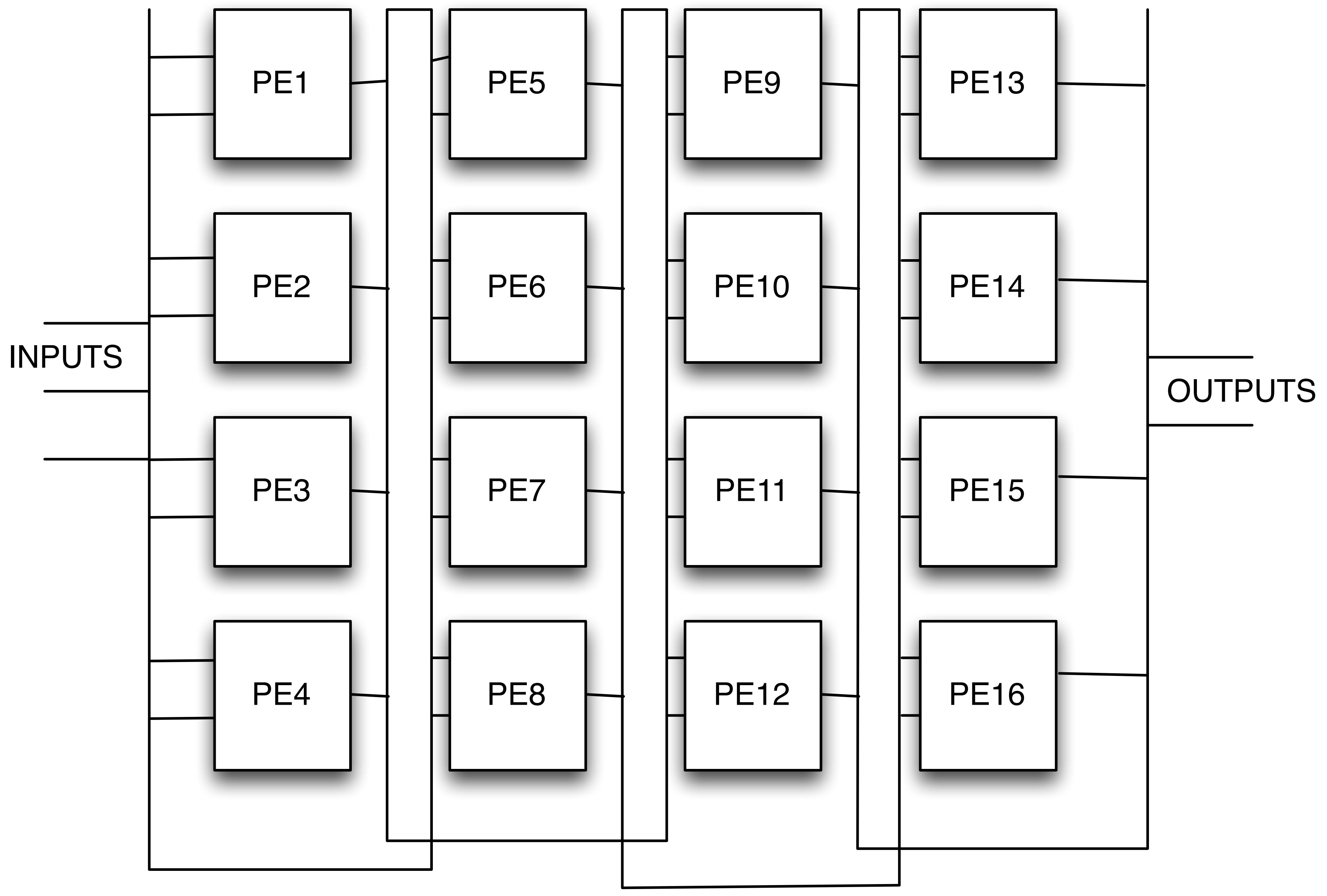
Results: seq. #10000 - number of I/O operations



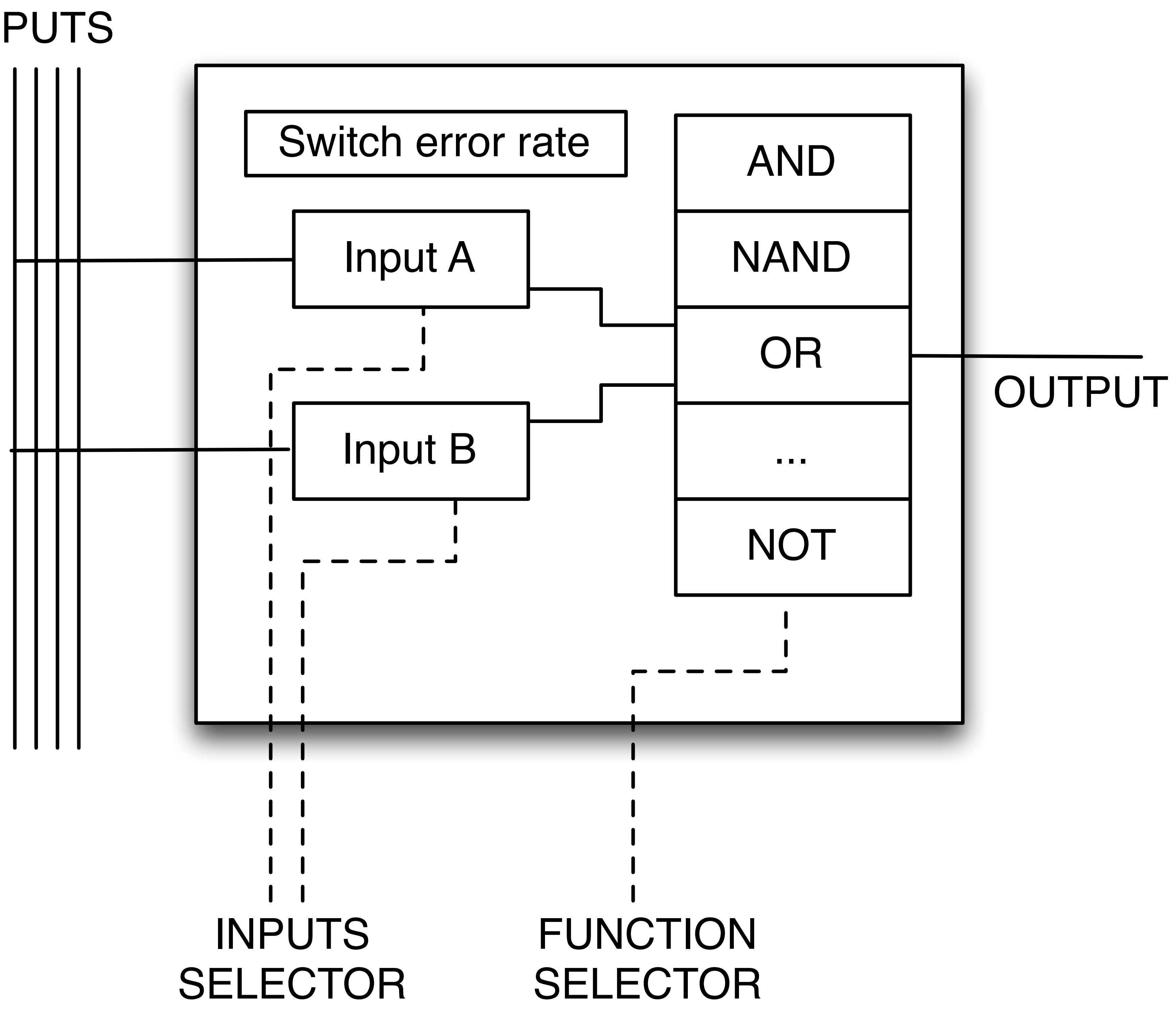
Energy/Error ratio optimisation through Genetic Algorithms

- Starting from truth table evolve logic network
- Initial population randomly generated
- Each generation random selected individuals breed and mutate
- Fitness evaluated on:
  - maximising the number of correct outputs given the truth table
  - minimising the number of logic gates used
  - minimising the energy consumption

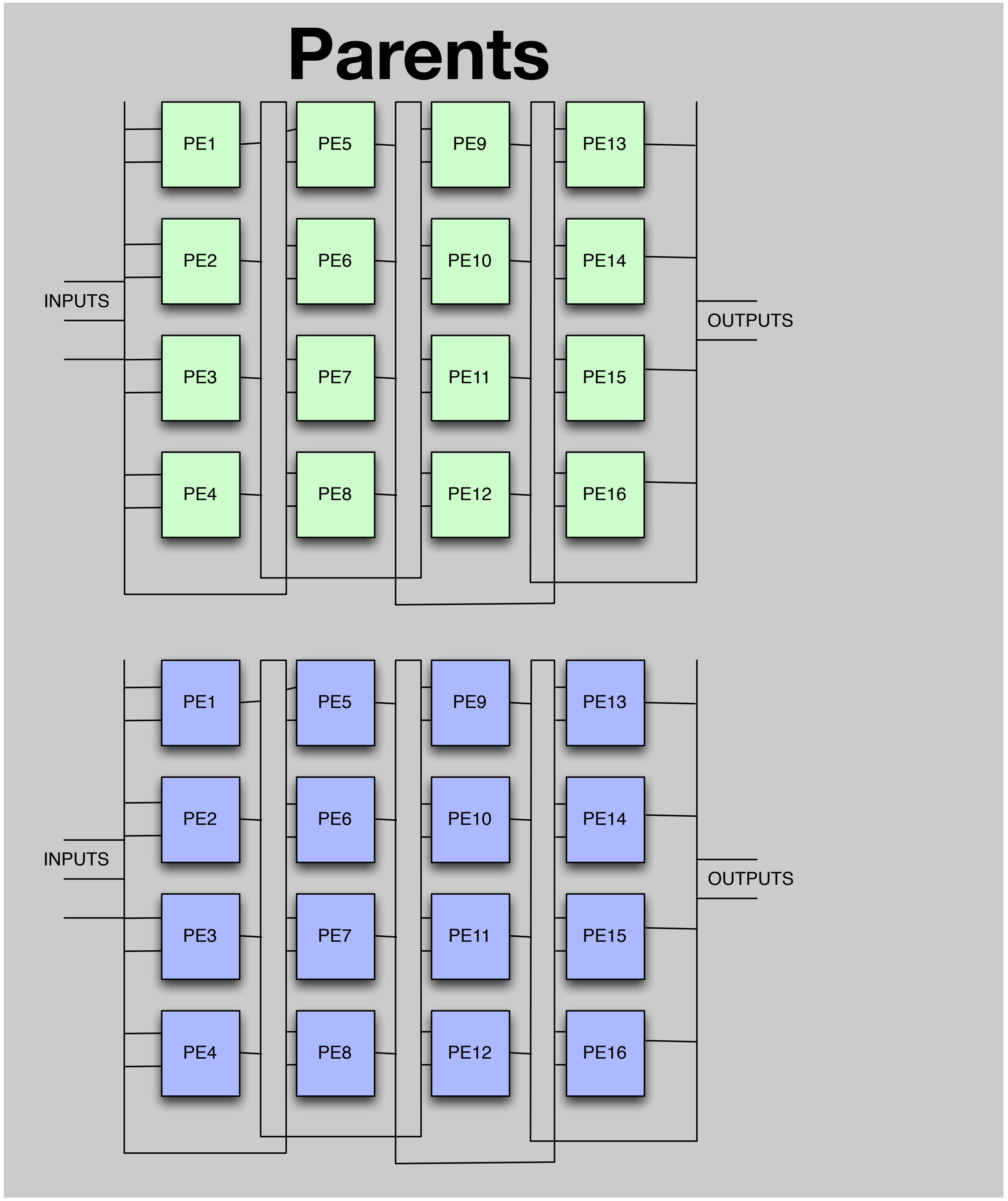
Individual


PE1
PE5
PE9
PE13
PE2
PE6
PE10
PE14
INPUTS
OUTPUTS
PE3
PE7
PE11
PE15
PE4
PE8
PE12
PE16

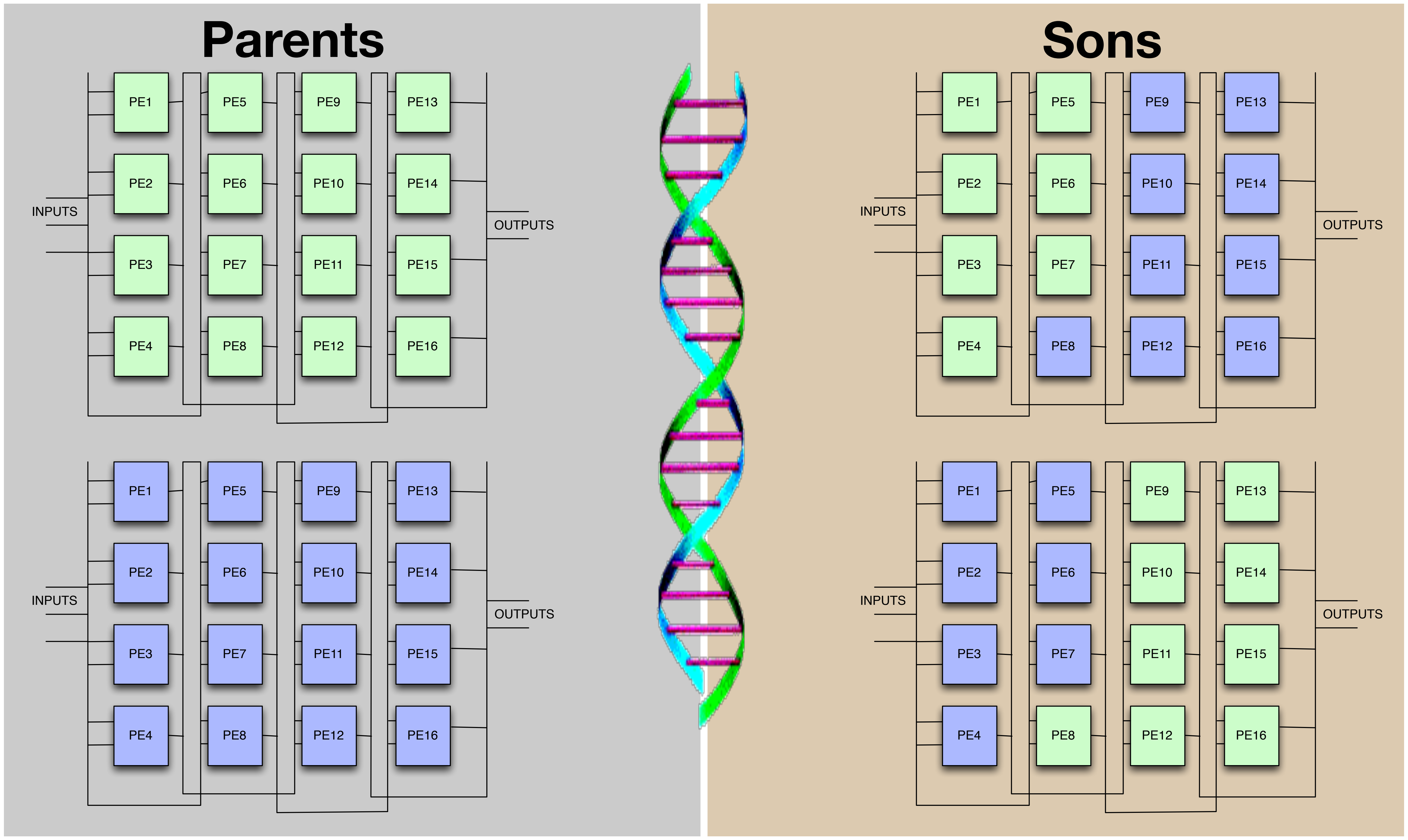
Individual: Processing Element



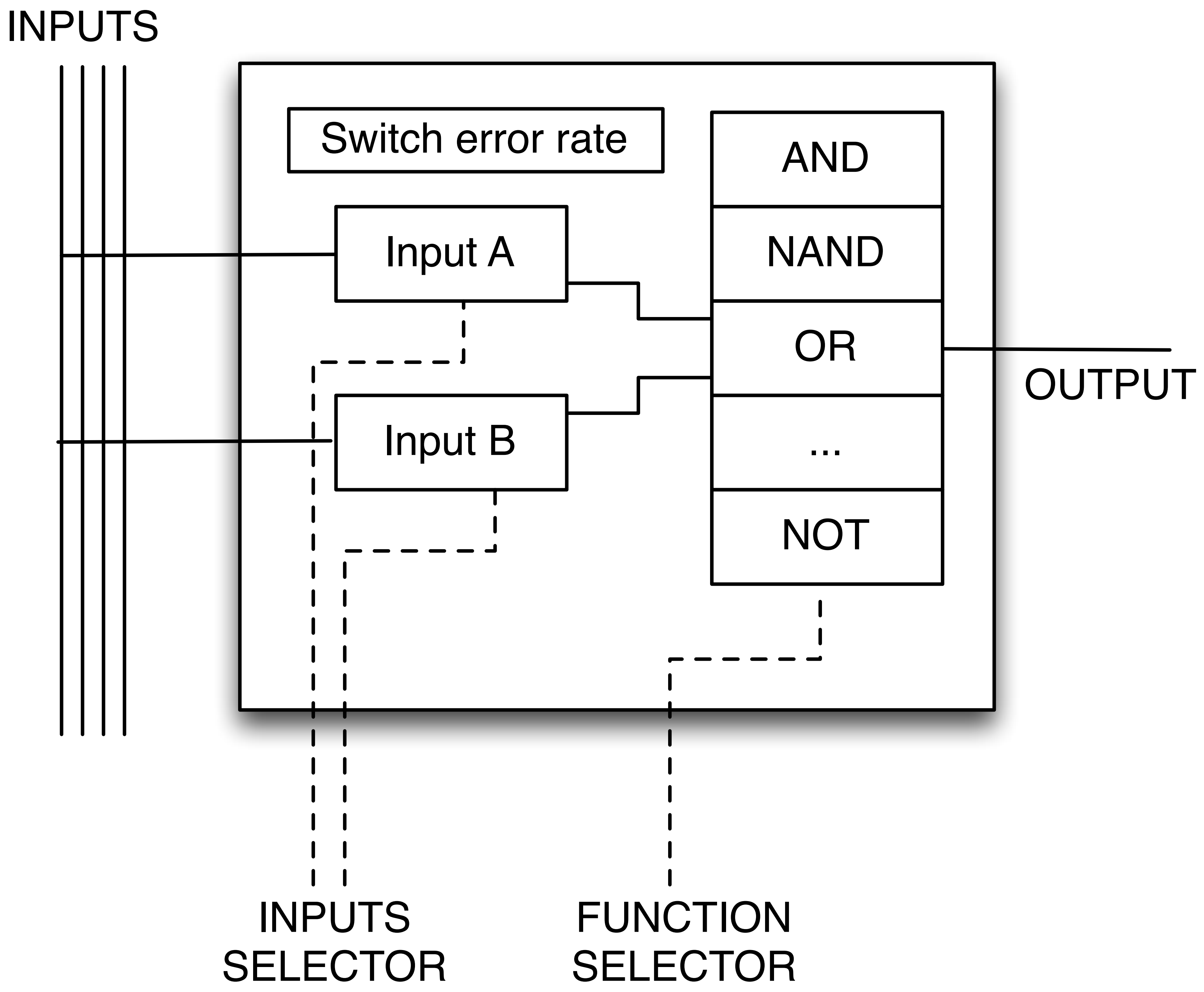
Crossover


Parents
INPUTS
PE1
PE5
PE9
PE13
PE2
PE6
PE10
PE14
PE3
PE7
PE11
PE15
PE4
PE8
PE12
PE16
OUTPUTS
INPUTS
PE1
PE5
PE9
PE13
PE2
PE6
PE10
PE14
PE3
PE7
PE11
PE15
PE4
PE8
PE12
PE16
OUTPUTS

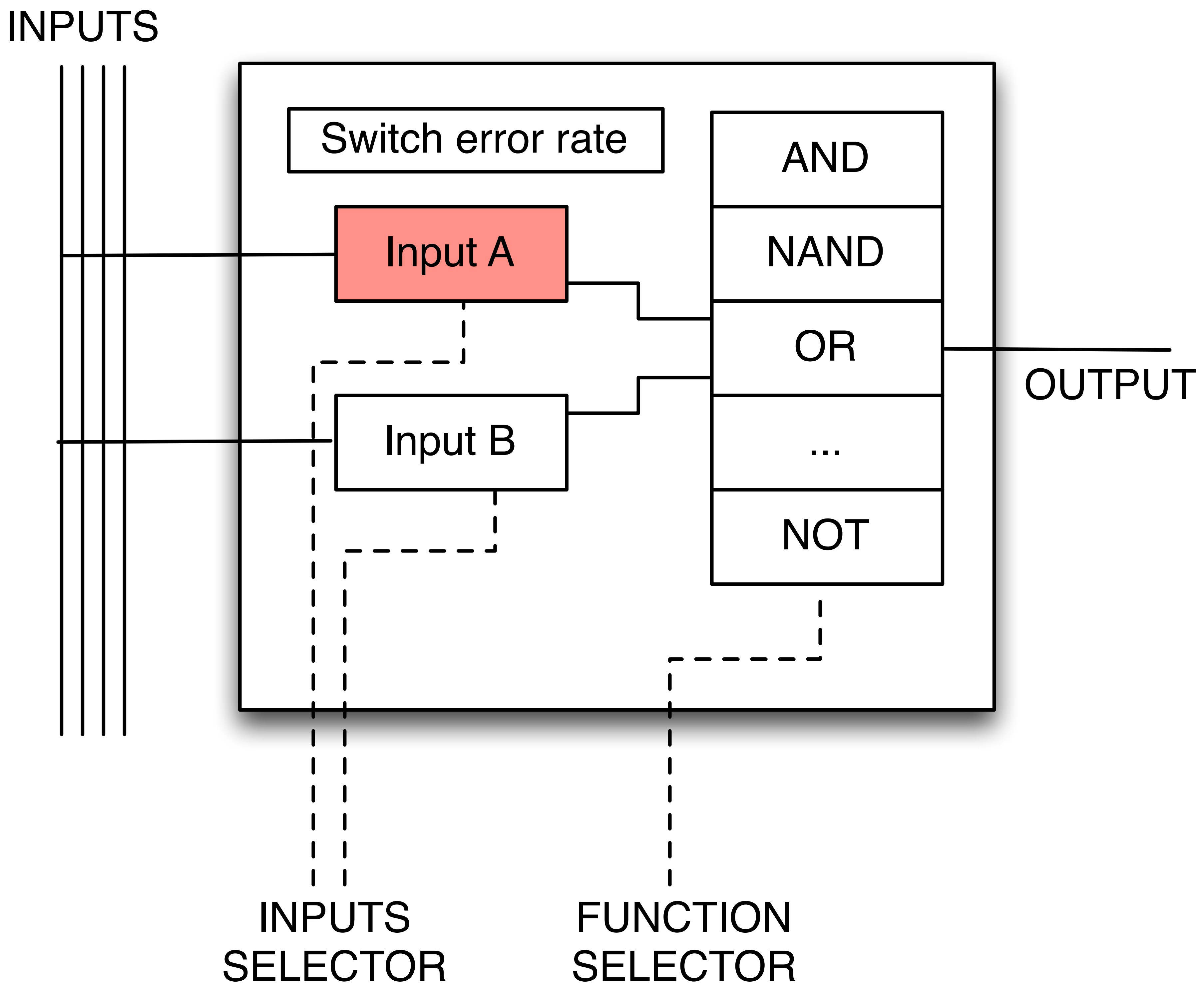
Crossover


Parents
Sons
PE1
PE2
PE3
PE4
PE5
PE6
PE7
PE8
PE9
PE10
PE11
PE12
PE13
PE14
PE15
PE16
INPUTS
OUTPUTS

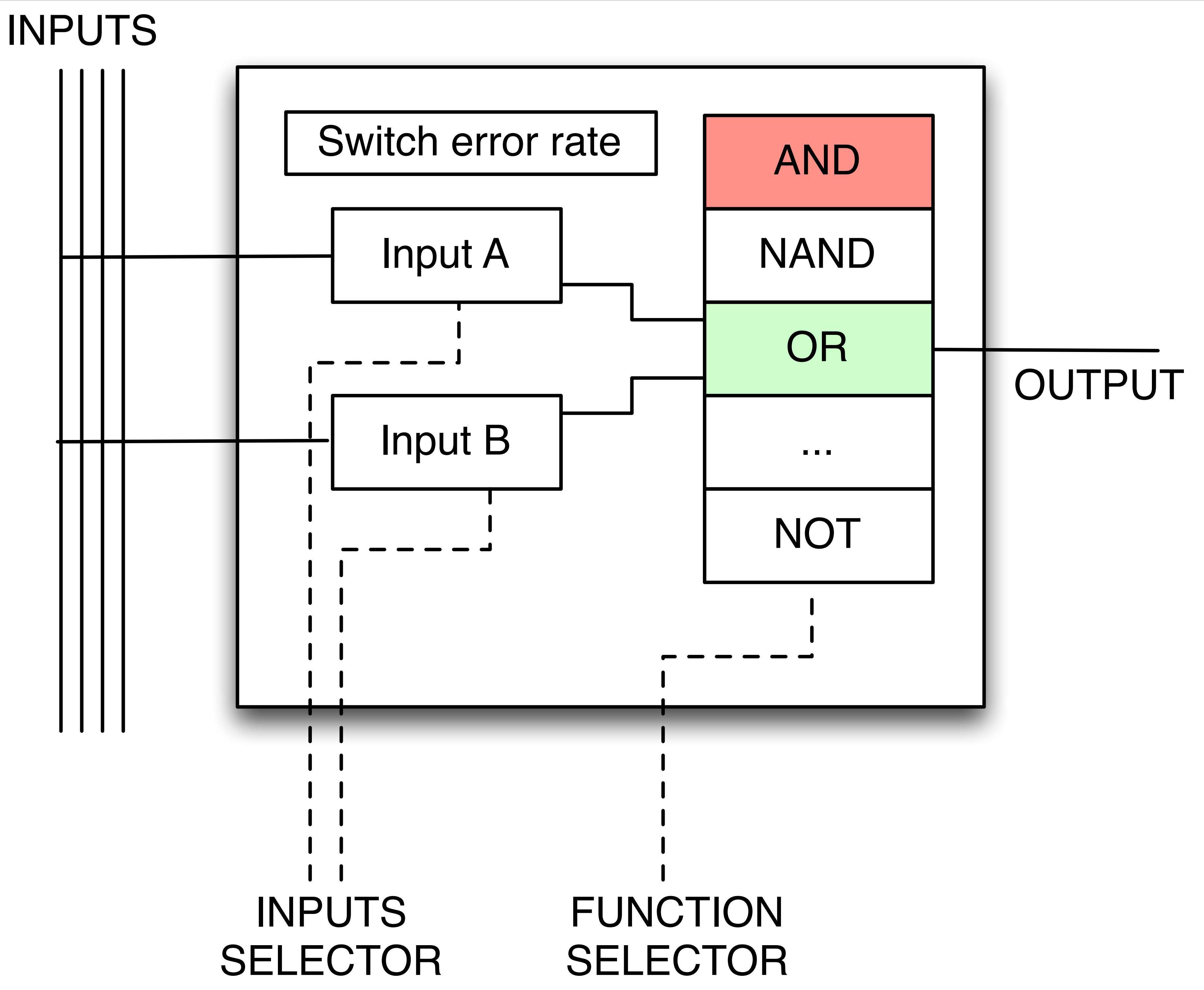
Mutation


INPUTS
Switch error rate
Input A
Input B
AND
NAND
OR
...
NOT
OUTPUT
INPUTS SELECTOR
FUNCTION SELECTOR

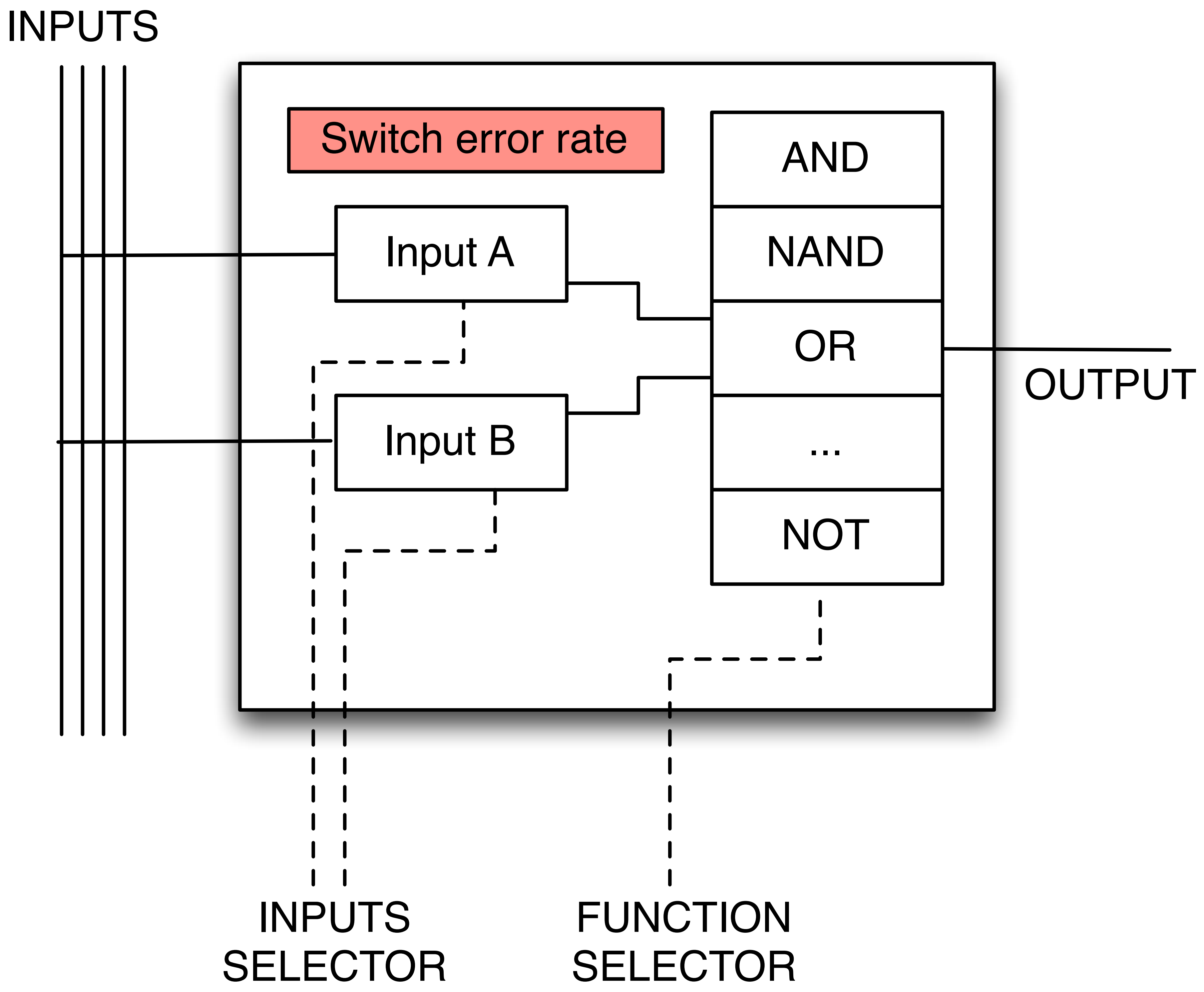
Mutation


INPUTS
Switch error rate
Input A
Input B
AND
NAND
OR
...
NOT
OUTPUT
INPUTS SELECTOR
FUNCTION SELECTOR

Mutation

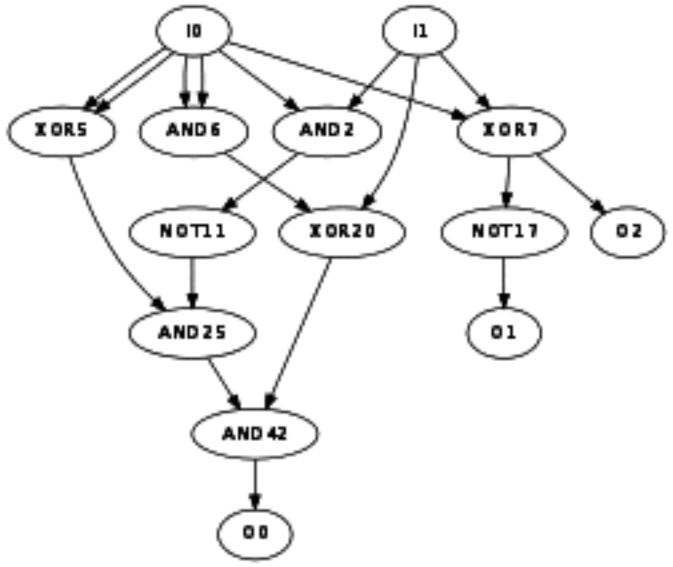
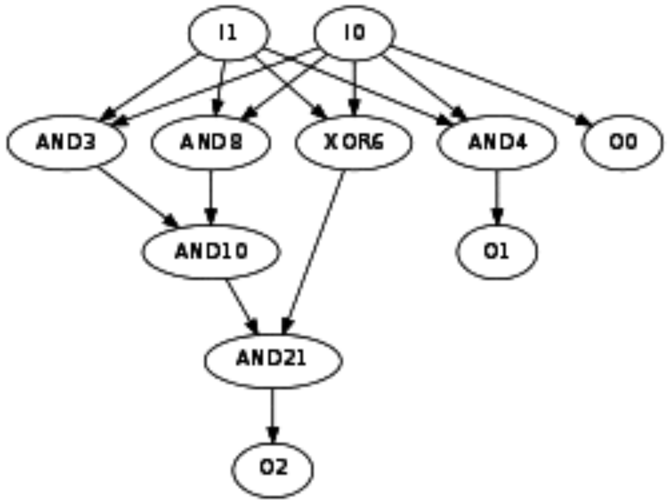
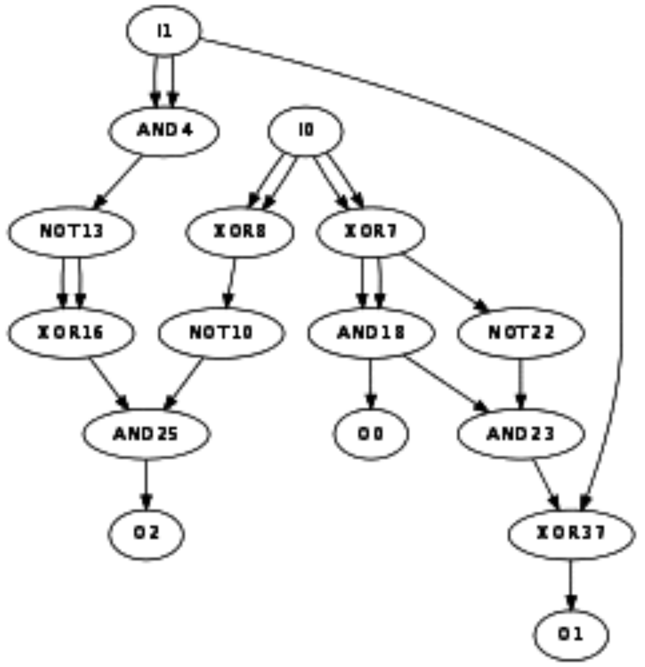
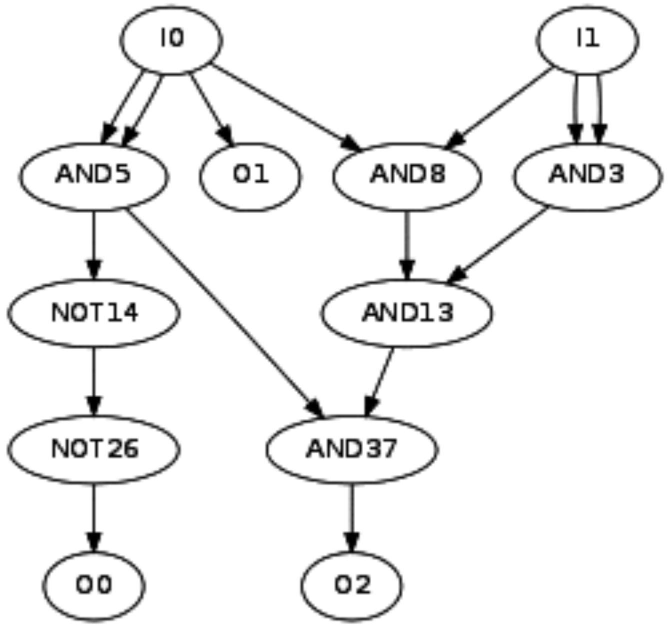
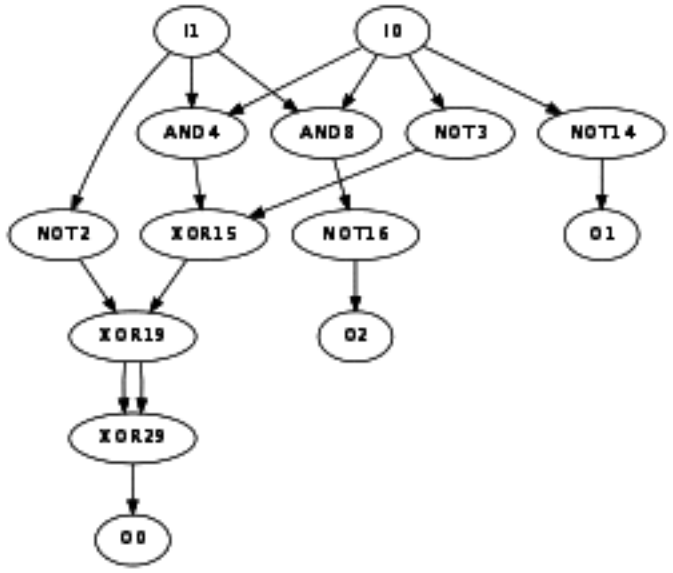
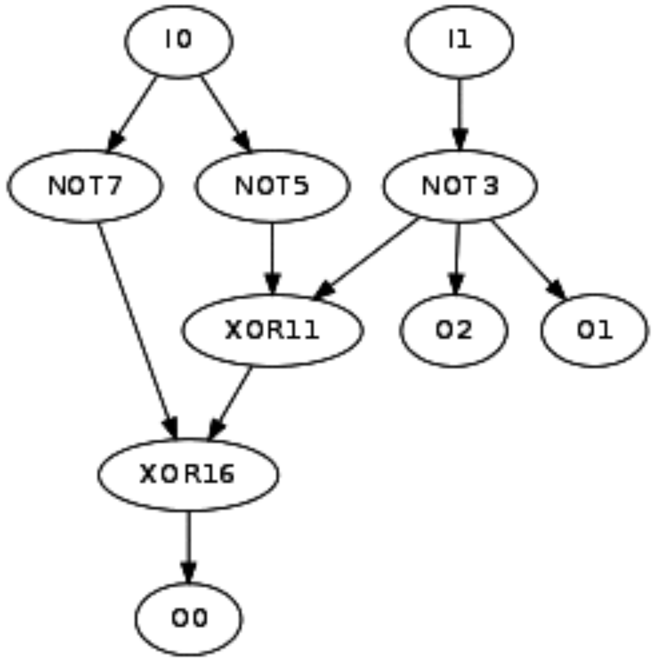
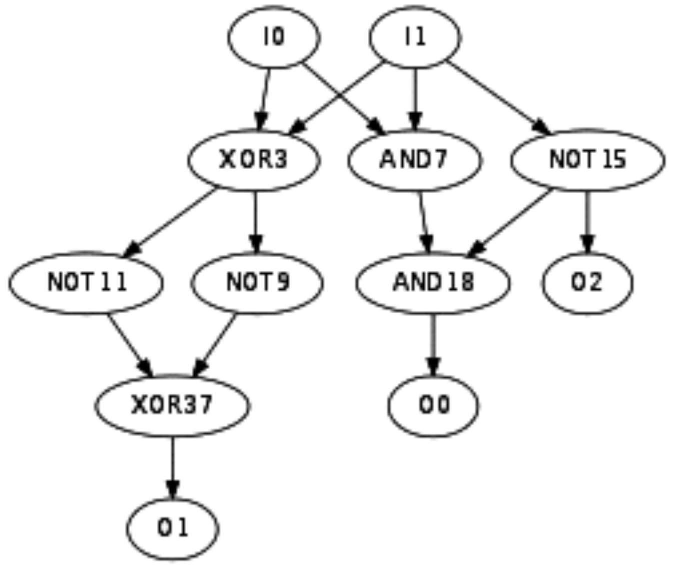
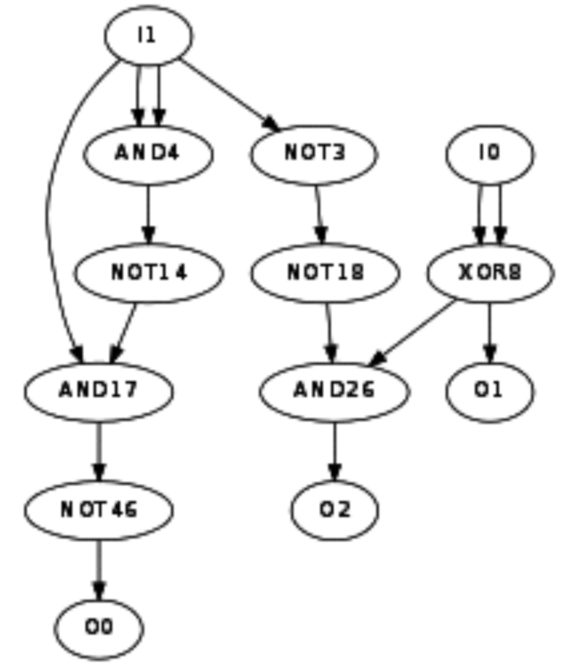
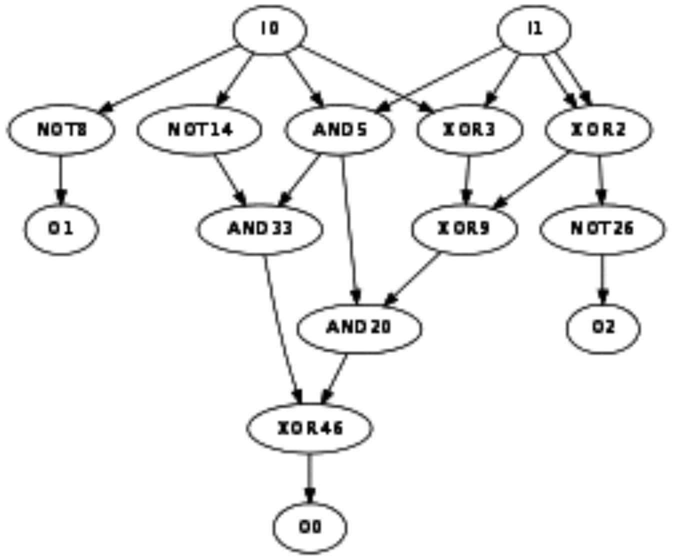
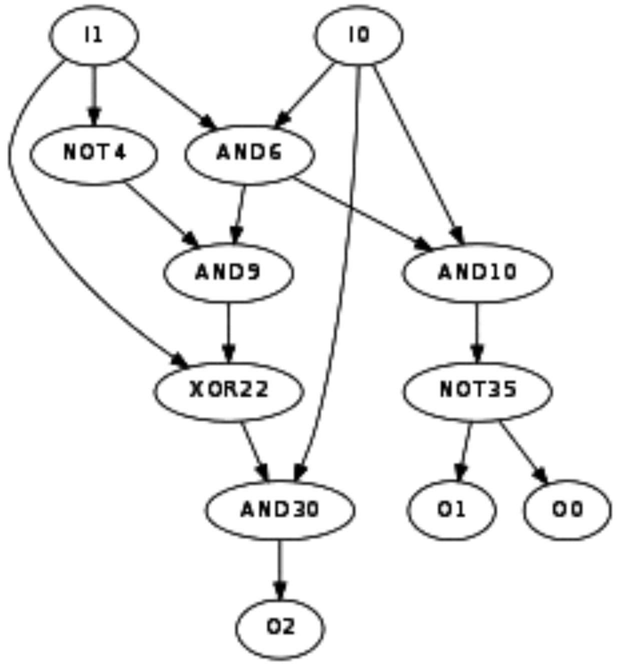
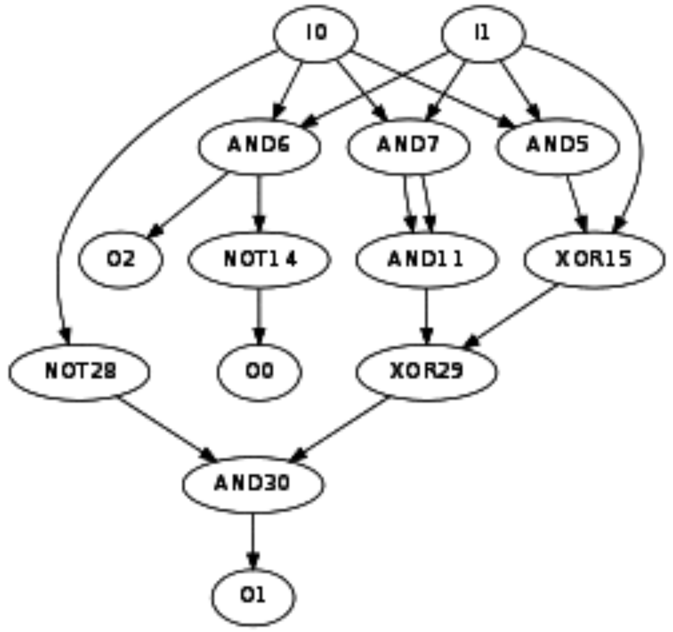
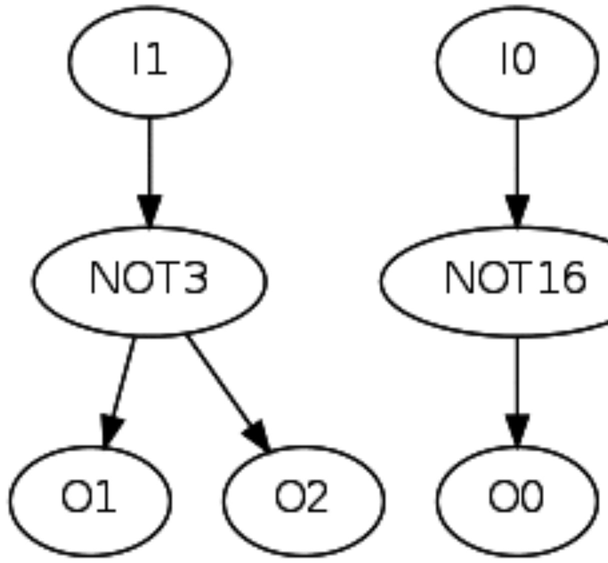

INPUTS
Switch error rate
Input A
Input B
AND
NAND
OR
...
NOT
OUTPUT
INPUTS SELECTOR
FUNCTION SELECTOR

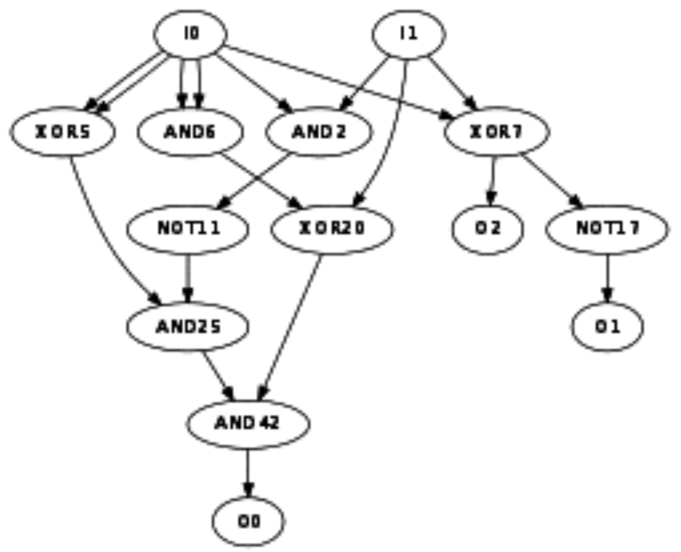
Mutation

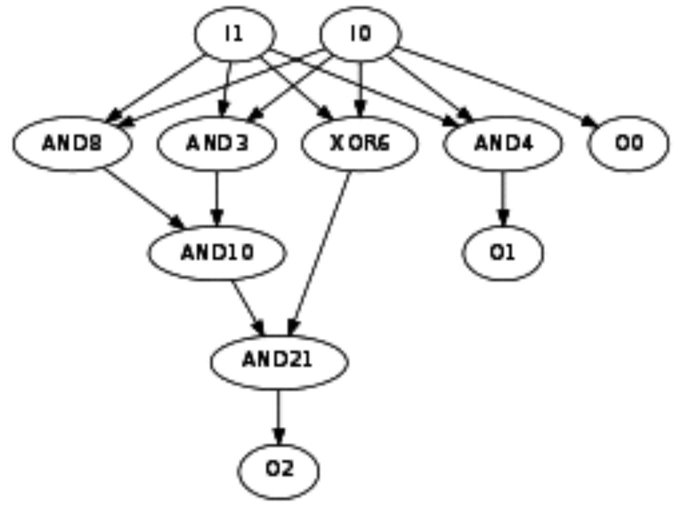

INPUTS
Switch error rate
AND
NAND
OR
...
NOT
Input A
Input B
OUTPUT
INPUTS SELECTOR
FUNCTION SELECTOR

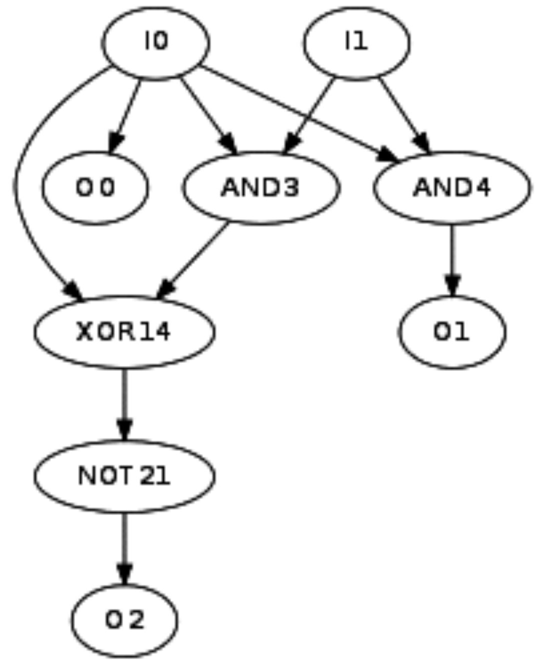
Unreliable logic circuit simulator: genetic evolution

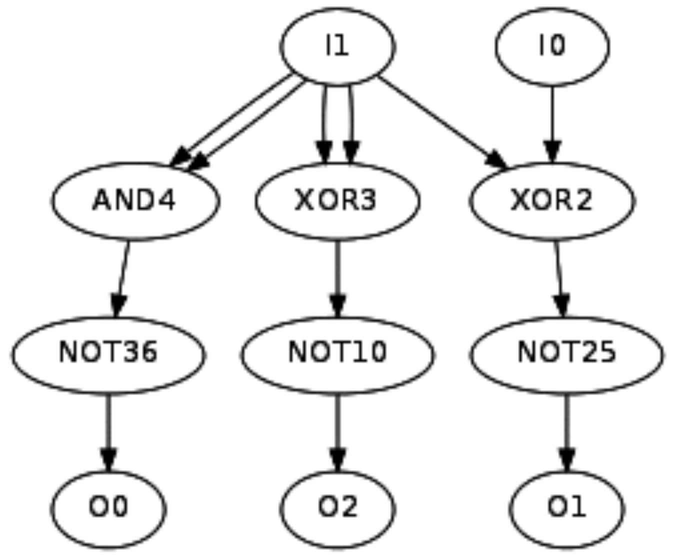
Inputs		Outputs		
A	B	A>B	A=B	A<B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

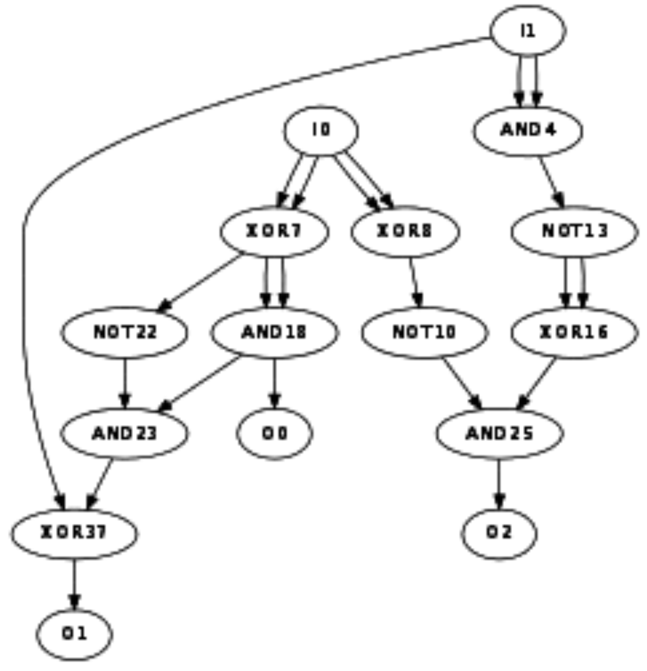

I0
I1
XOR5
AND6
AND2
XOR7
NOT11
XOR20
NOT17
O2
AND25
O1
AND42
O0

I1
I0
AND3
AND8
XOR6
AND4
O0
AND10
O1
AND21
O2

I1
AND4
I0
NOT13
XOR8
XOR7
XOR16
NOT10
AND18
NOT22
AND25
O0
AND23
O2
XOR37
O1

I0
I1
AND5
O1
AND8
AND3
NOT14
AND13
NOT26
AND37
O0
O2

I1
I0
AND4
AND8
NOT3
NOT14
NOT2
XOR15
NOT16
O1
XOR19
O2
XOR29
O0

I0
I1
NOT7
NOT5
NOT3
XOR11
O2
O1
XOR16
O0

I0
I1
XOR3
AND7
NOT15
NOT11
NOT9
AND18
O2
XOR37
O0
O1

I1
AND4
NOT3
I0
NOT14
NOT18
XOR8
AND17
AND26
O1
NOT46
O2
O0

I0
I1
NOT8
NOT14
AND5
XOR3
XOR2
O1
AND33
XOR9
NOT26
AND20
O2
XOR46
O0

I1
I0
NOT4
AND6
AND9
AND10
XOR22
NOT35
AND30
O1
O0
O2

I0
I1
AND6
AND7
AND5
O2
NOT14
AND11
XOR15
NOT28
O0
XOR29
AND30
O1

I1
I0
NOT3
NOT16
O1
O2
O0

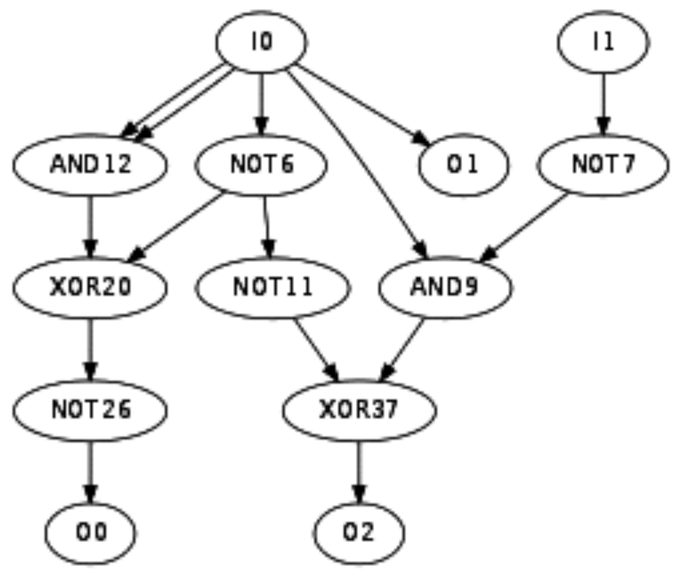

I0
I1
XOR5
AND6
AND2
XOR7
NOT11
XOR20
O2
NOT17
AND25
O1
AND42
O0

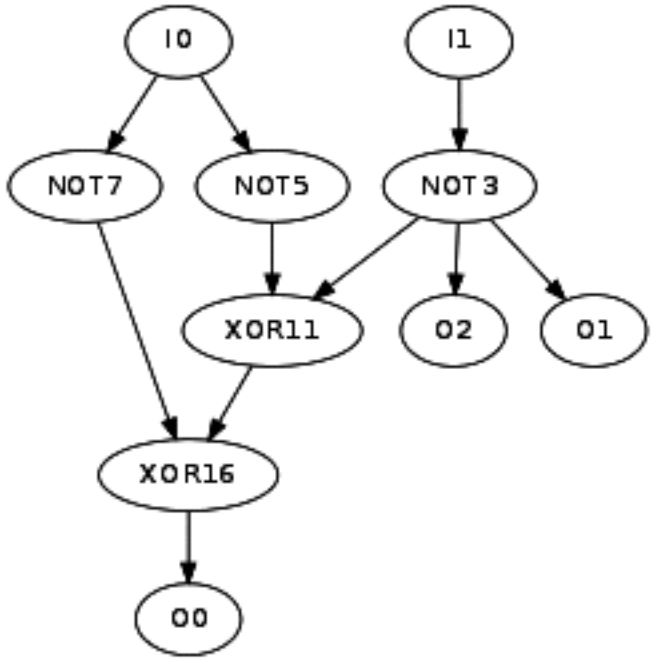

I1
I0
AND8
AND3
XOR6
AND4
O0
AND10
O1
AND21
O2

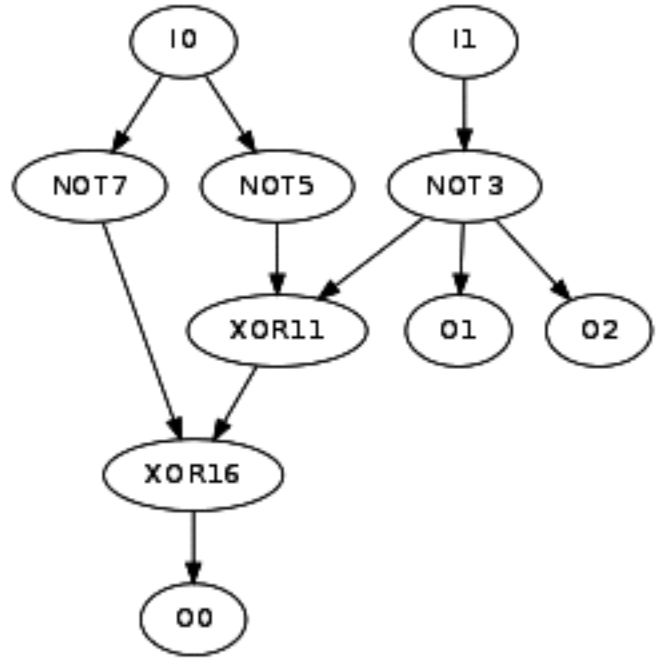

I0
I1
O0
AND3
AND4
XOR14
O1
NOT21
O2

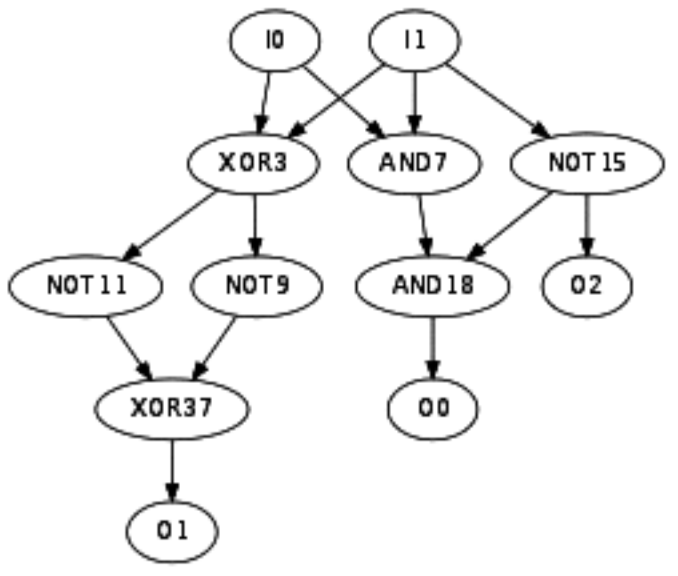

I1
I0
AND4
XOR3
XOR2
NOT36
NOT10
NOT25
O0
O2
O1

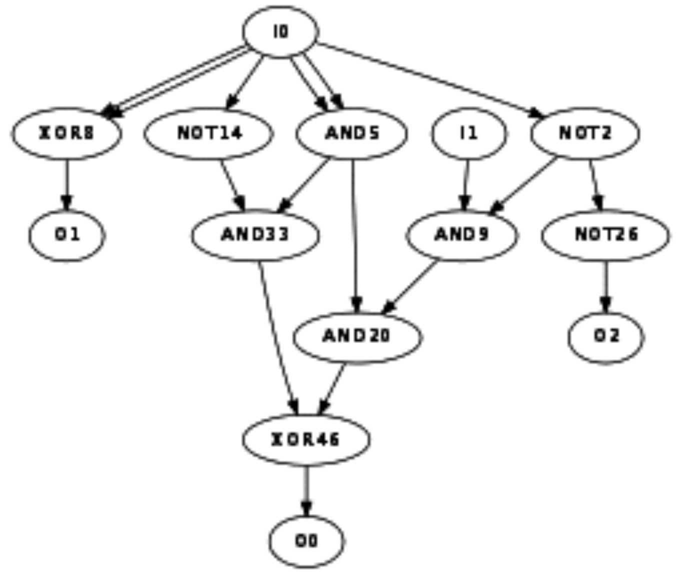

I1
I0
AND4
XOR7
XOR8
NOT13
NOT22
AND18
NOT10
XOR16
AND23
O0
AND25
XOR37
O2
O1

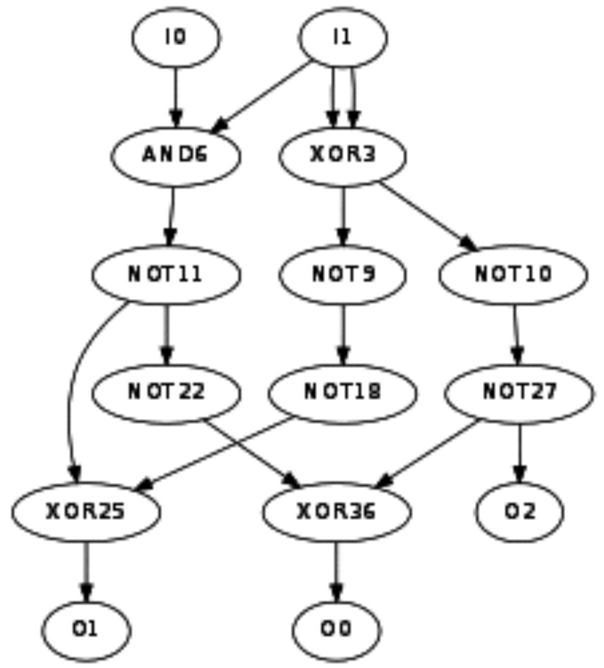

I0
I1
AND12
NOT6
O1
NOT7
XOR20
NOT11
AND9
NOT26
XOR37
O0
O2

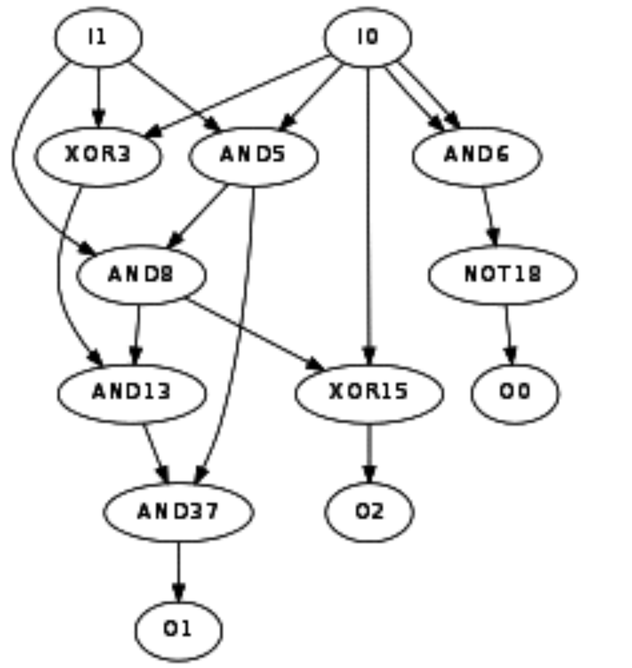

I0
I1
NOT7
NOT5
NOT3
XOR11
O2
O1
XOR16
O0

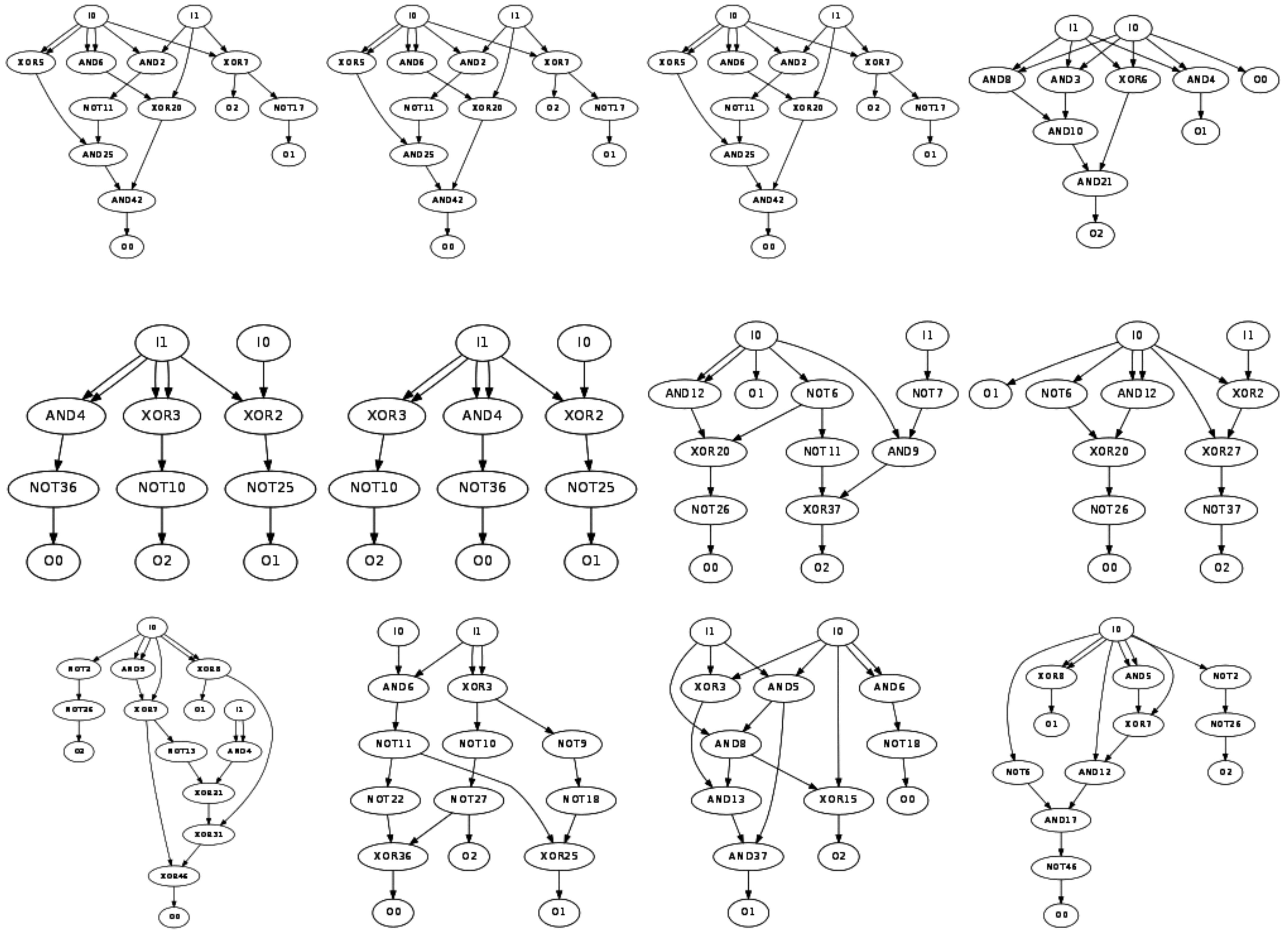

I0
I1
NOT7
NOT5
NOT3
XOR11
O1
O2
XOR16
O0


I0
I1
XOR3
AND7
NOT15
NOT11
NOT9
AND18
O2
XOR37
O0
O1


I0
XOR8
NOT14
AND5
I1
NOT2
O1
AND33
AND9
NOT26
AND20
O2
XOR46
O0

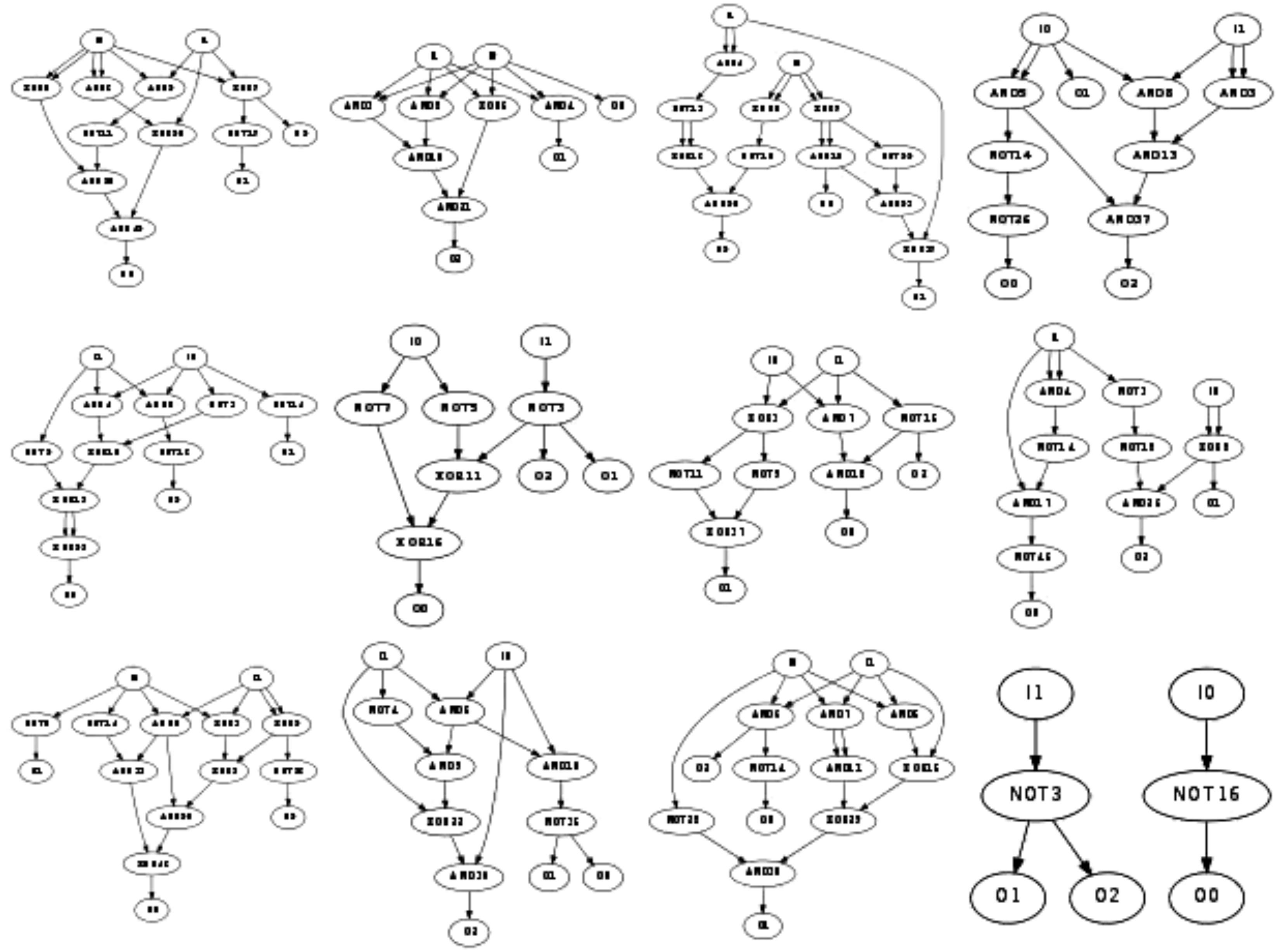

I0
I1
AND6
XOR3
NOT11
NOT9
NOT10
NOT22
NOT18
NOT27
XOR25
XOR36
O2
O1
O0


I1
I0
XOR3
AND5
AND6
AND8
NOT18
AND13
XOR15
O0
AND37
O2
O1

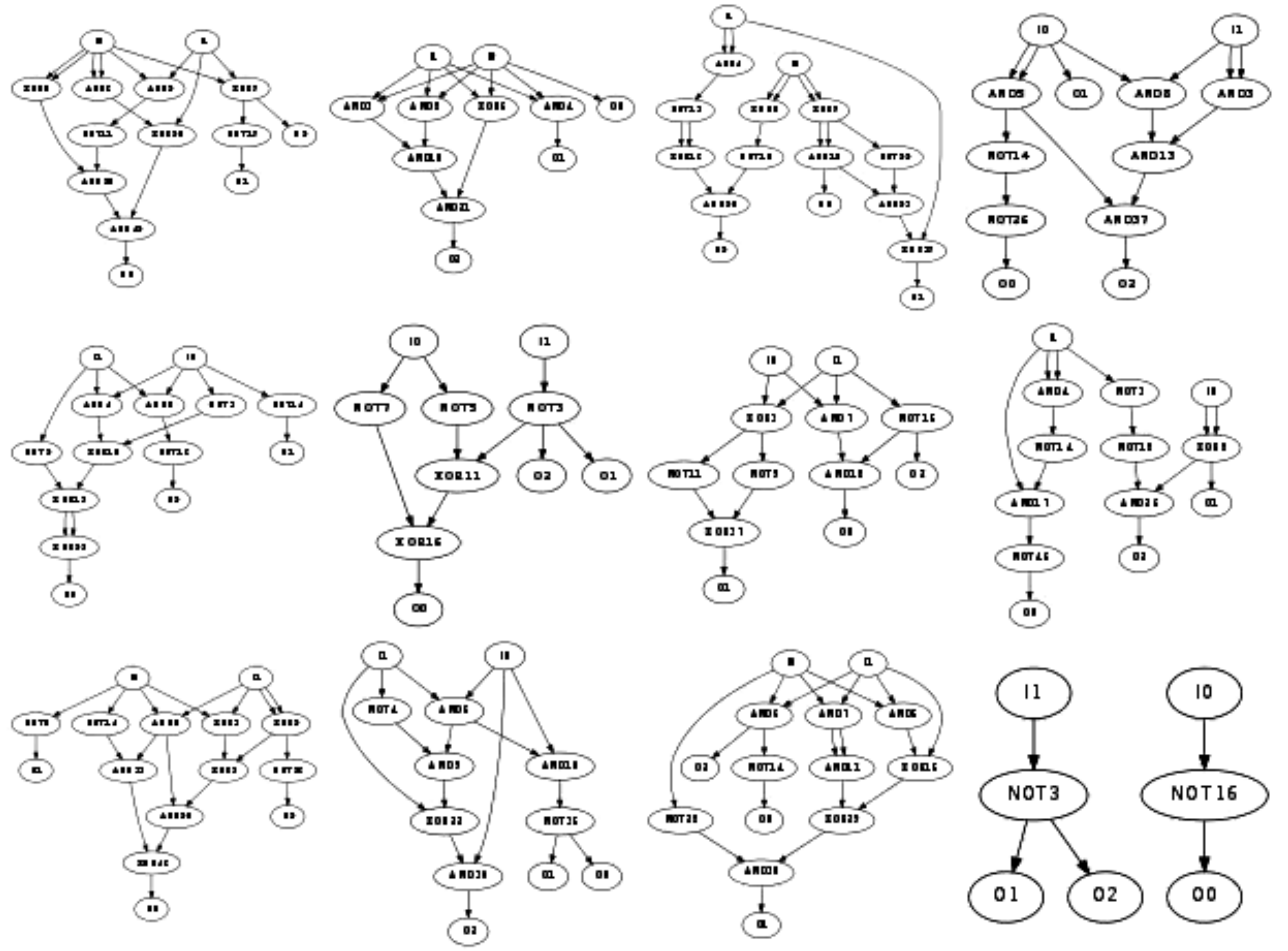


I0
I1
XOR5
AND6
AND2
XOR7
NOT11
XOR20
O2
NOT17
AND25
O1
AND42
O0
I0
I1
XOR5
AND6
AND2
XOR7
NOT11
XOR20
O2
NOT17
AND25
O1
AND42
O0
I0
I1
XOR5
AND6
AND2
XOR7
NOT11
XOR20
O2
NOT17
AND25
O1
AND42
O0
I1
I0
AND8
AND3
XOR6
AND4
O0
AND10
O1
AND21
O2
I1
I0
AND4
XOR3
XOR2
NOT36
NOT10
NOT25
O0
O2
O1
I1
I0
XOR3
AND4
XOR2
NOT10
NOT36
NOT25
O2
O0
O1
I0
I1
AND12
O1
NOT6
NOT7
XOR20
NOT11
AND9
NOT26
XOR37
O0
O2
I0
I1
O1
NOT6
AND12
XOR2
XOR20
XOR27
NOT26
NOT37
O0
O2
I0
NOT2
AND5
XOR8
NOT26
XOR7
O1
I1
O2
NOT13
AND4
XOR21
XOR31
XOR46
O0
I0
I1
AND6
XOR3
NOT11
NOT10
NOT9
NOT22
NOT27
NOT18
XOR36
O2
XOR25
O0
O1
I1
I0
XOR3
AND5
AND6
AND8
NOT18
AND13
XOR15
O0
AND37
O2
O1
I0
XOR8
AND5
NOT2
O1
XOR7
NOT26
NOT6
AND12
O2
AND17
NOT46
O0

Unreliable logic circuit simulator: genetic evolution

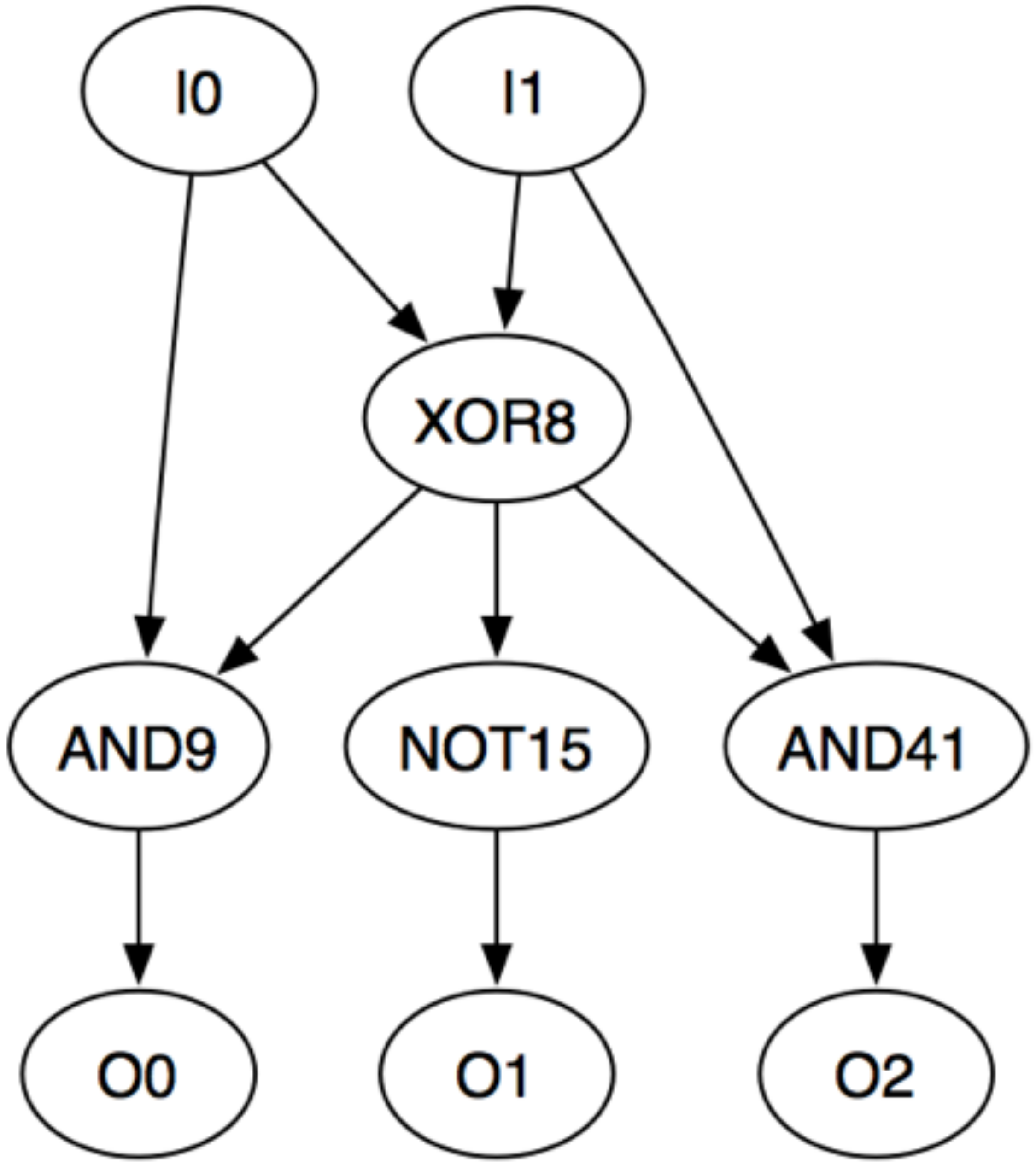


Unreliable logic circuit simulator: genetic evolution

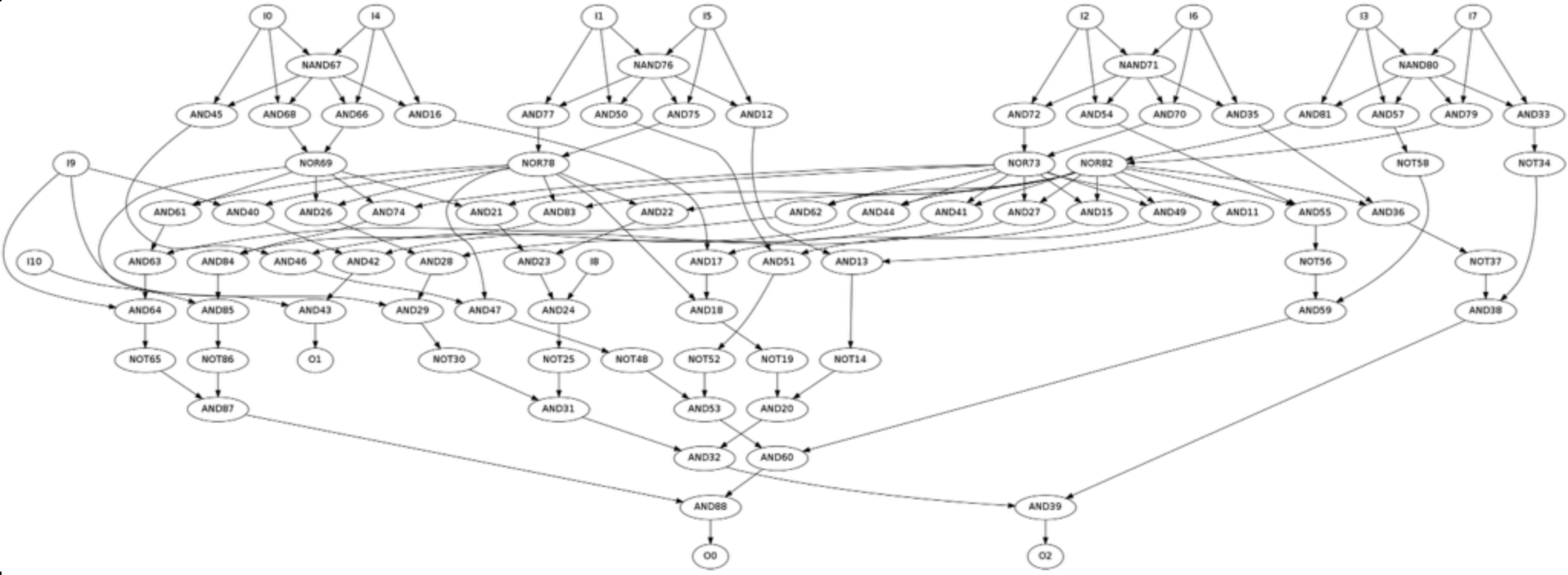


Unreliable logic circuit simulator: genetic evolution

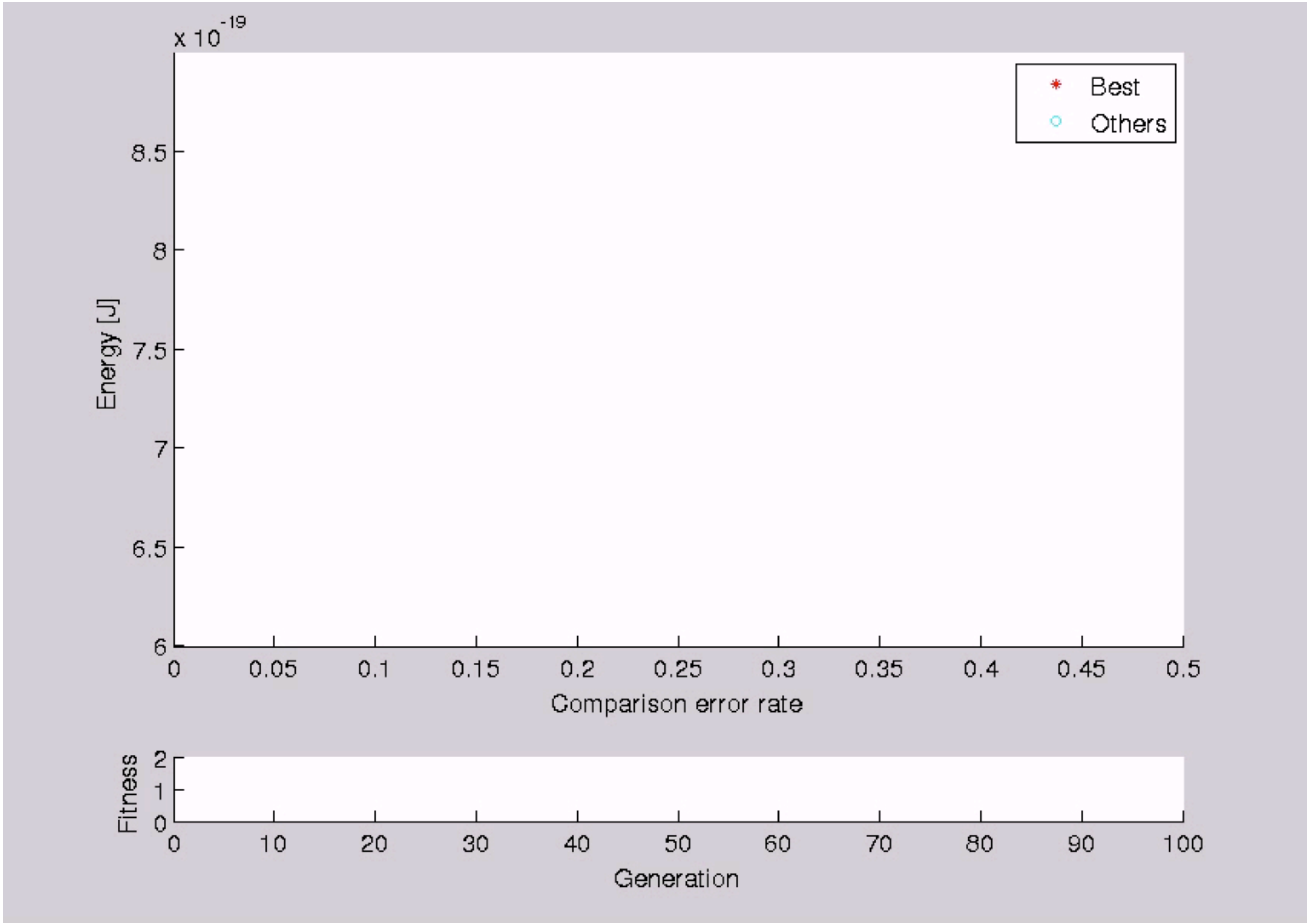
Inputs		Outputs		
A	B	A>B	A=B	A<B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0



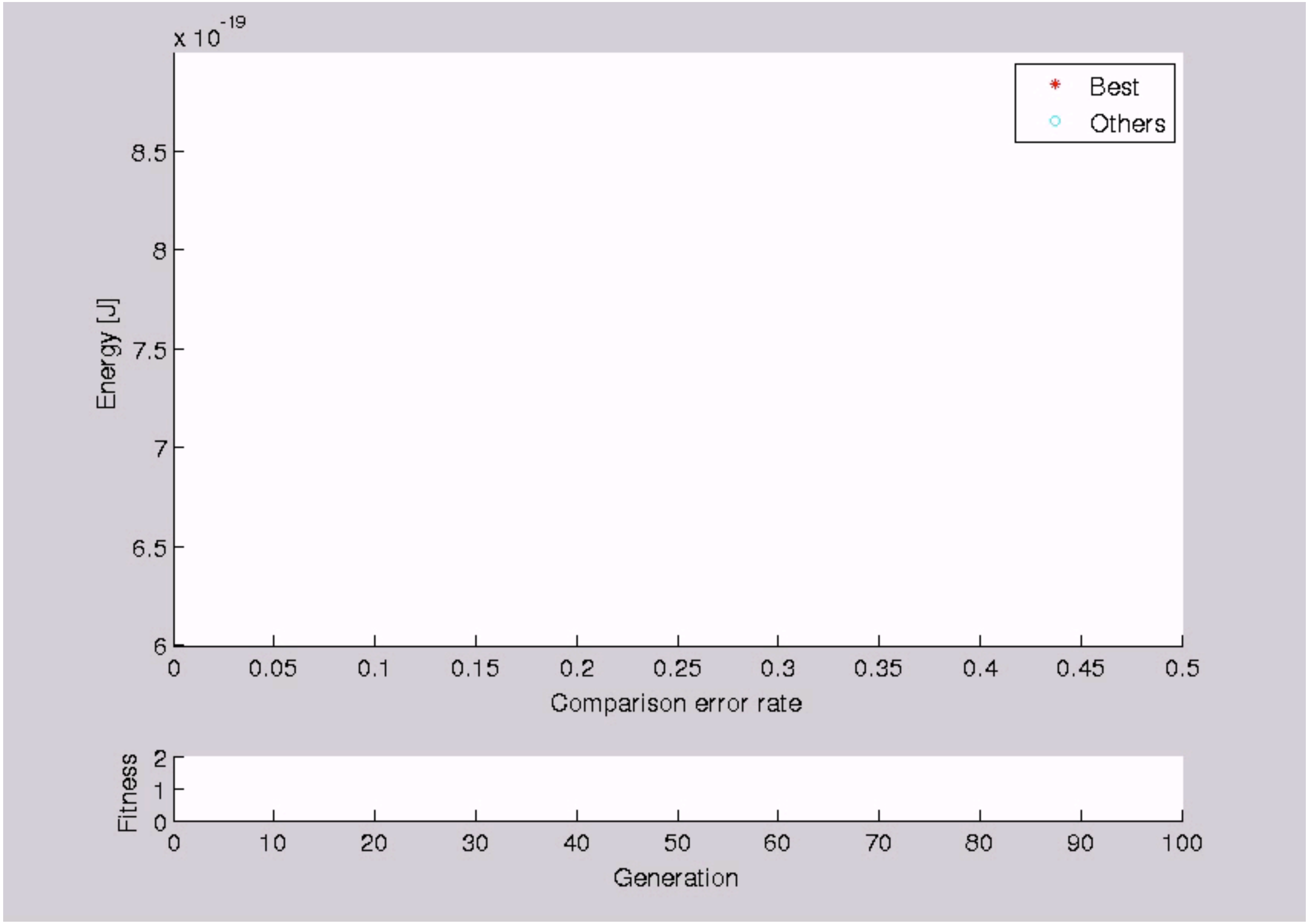
Unreliable logic circuit simulator: genetic evolution



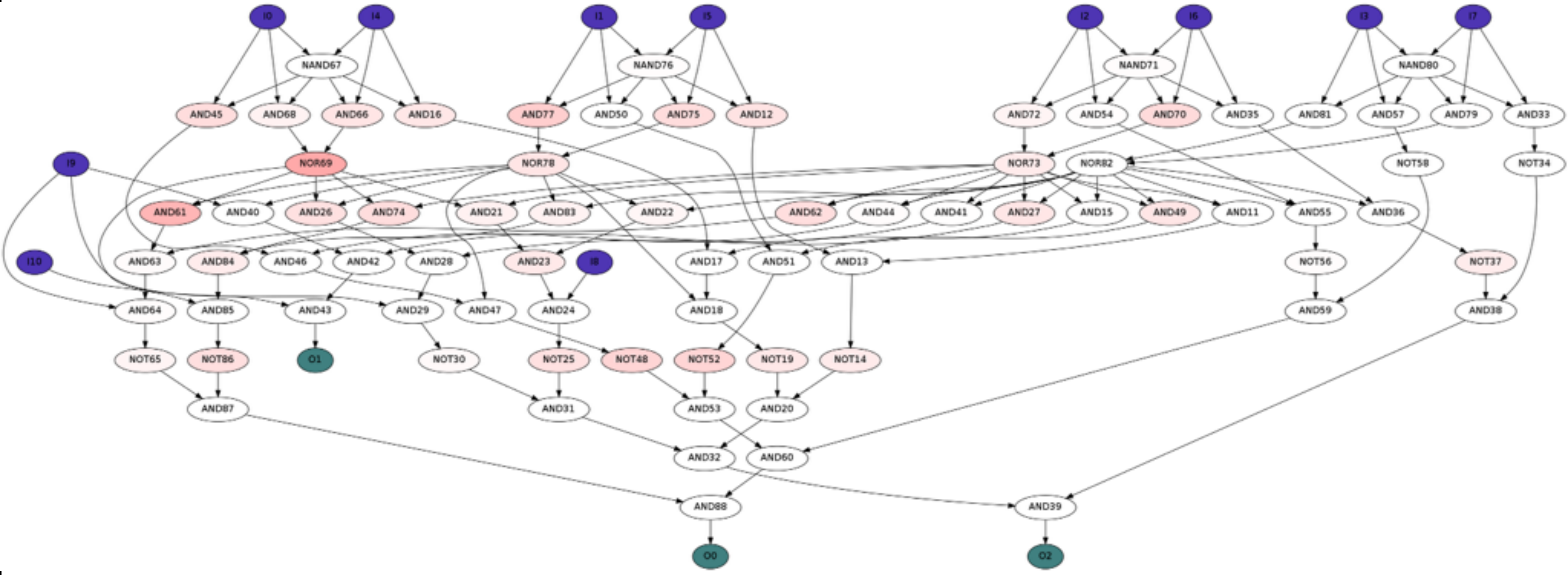
Unreliable logic circuit simulator: genetic evolution



Unreliable logic circuit simulator: genetic evolution



Unreliable logic circuit simulator: genetic evolution



Thank you for your attention!

igor.neri@nipslab.org

NiPS Laboratory
Noise in Physical Systems

LANDAUER