Majority-based Synthesis for Digital Nano-Technologies

Giovanni De Micheli

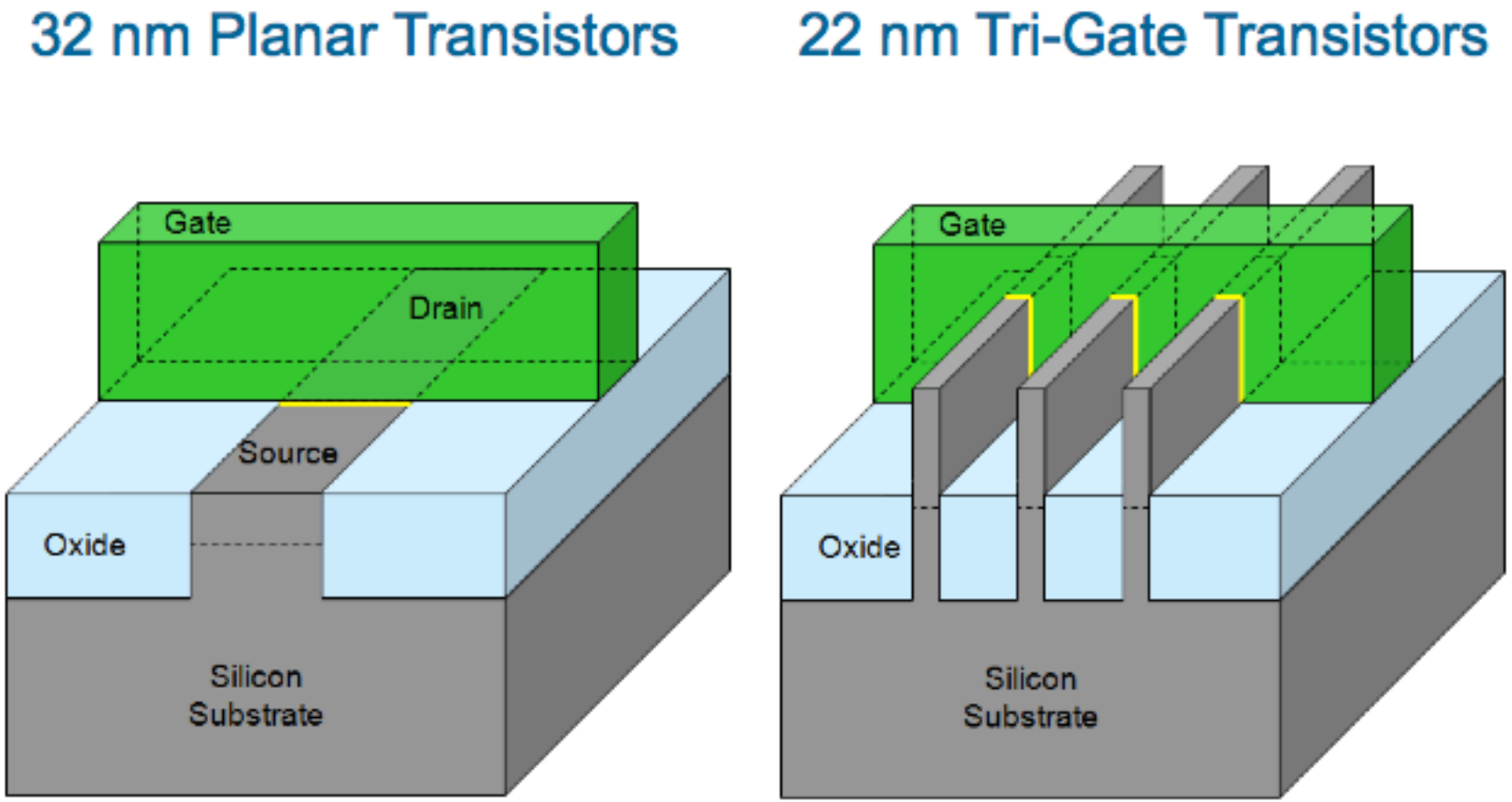
Outline

- **Introduction**
- **Technological innovations and motivation**
  - **Emerging nanotechnologies and devices**
- Design with emerging technologies
  - Physical and logic synthesis
- The majority paradigm in logic synthesis
  - Models, algorithms and tools
- Conclusions

The emerging nano-technologies

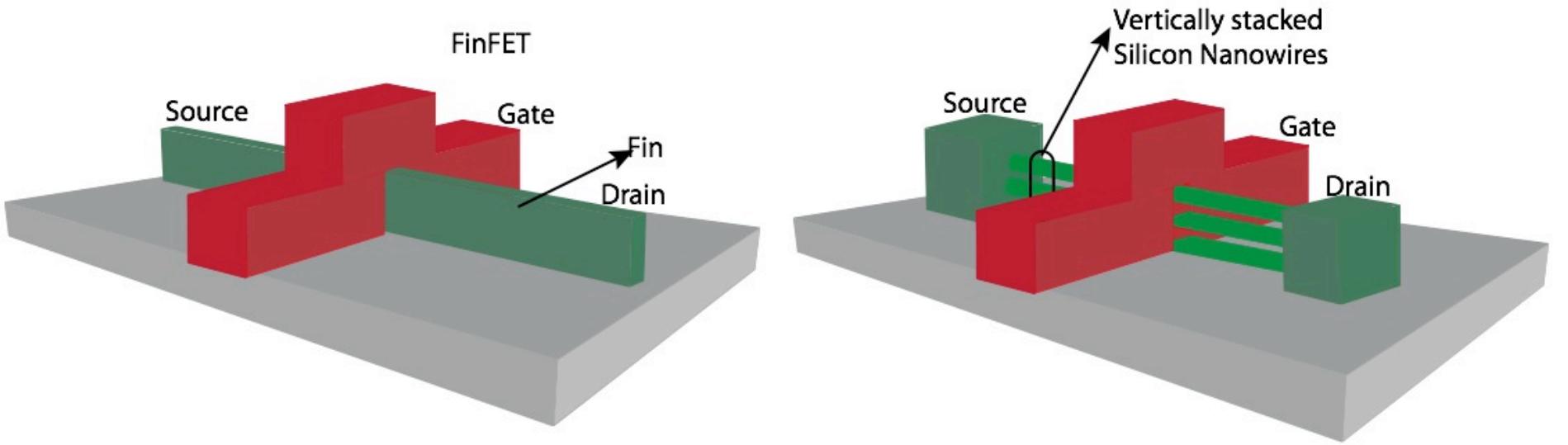
- *Enhanced* silicon CMOS is likely to remain the main manufacturing process in the medium term
  - The 10 and 7nm technology nodes are planned
- What are the candidate technologies for the 5nm node and beyond?
  - Silicon Nanowires (SiNW)
  - Tunneling FETs (TFET)
  - Carbon Nanotubes (CNT)
  - 2D devices (flatronics)
- What are the common denominators from a design standpoint?

22 nm Tri-Gate Transistors

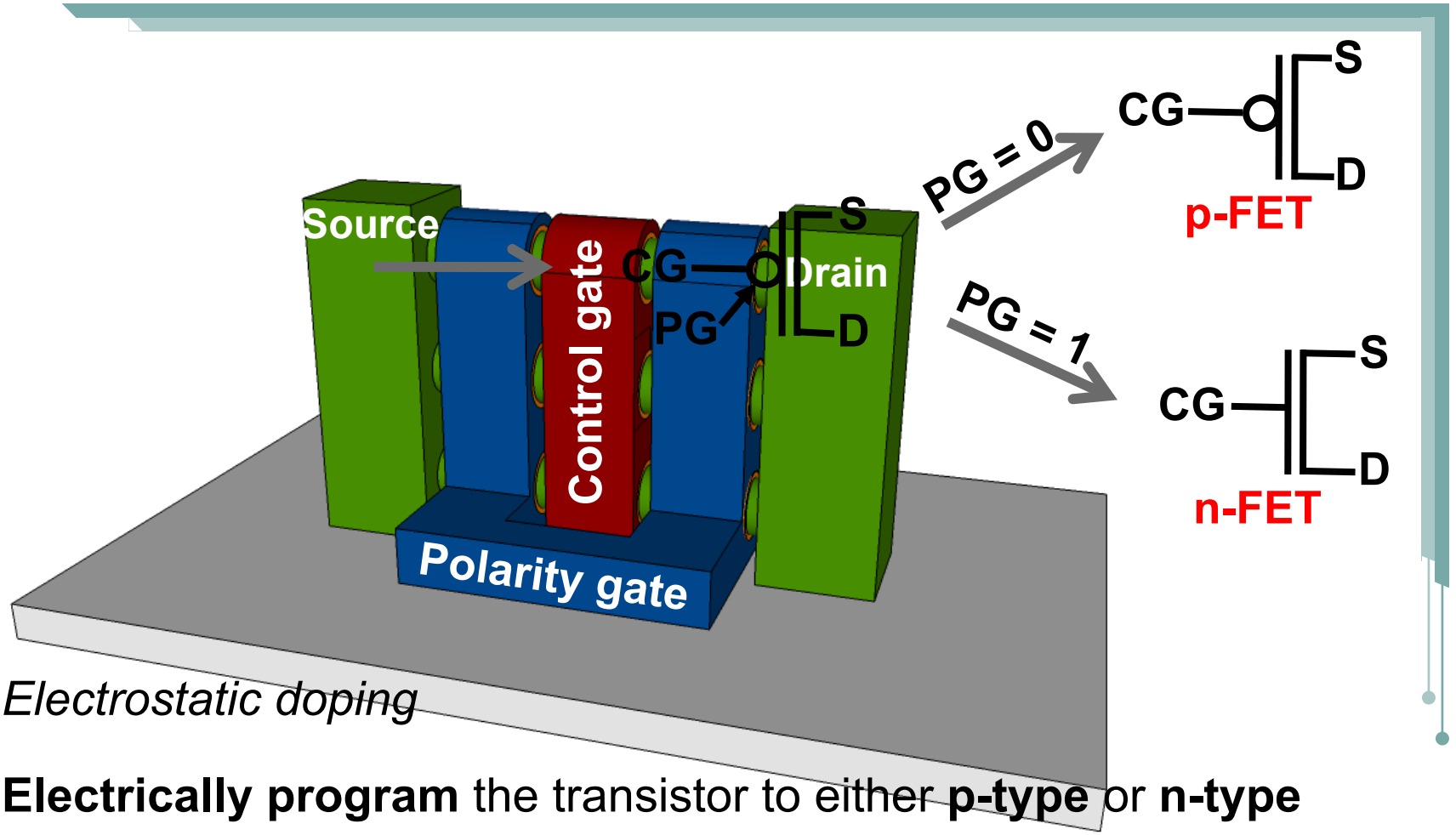


[Courtesy: M. Bohr]

FinFETs versus SiNW FETs



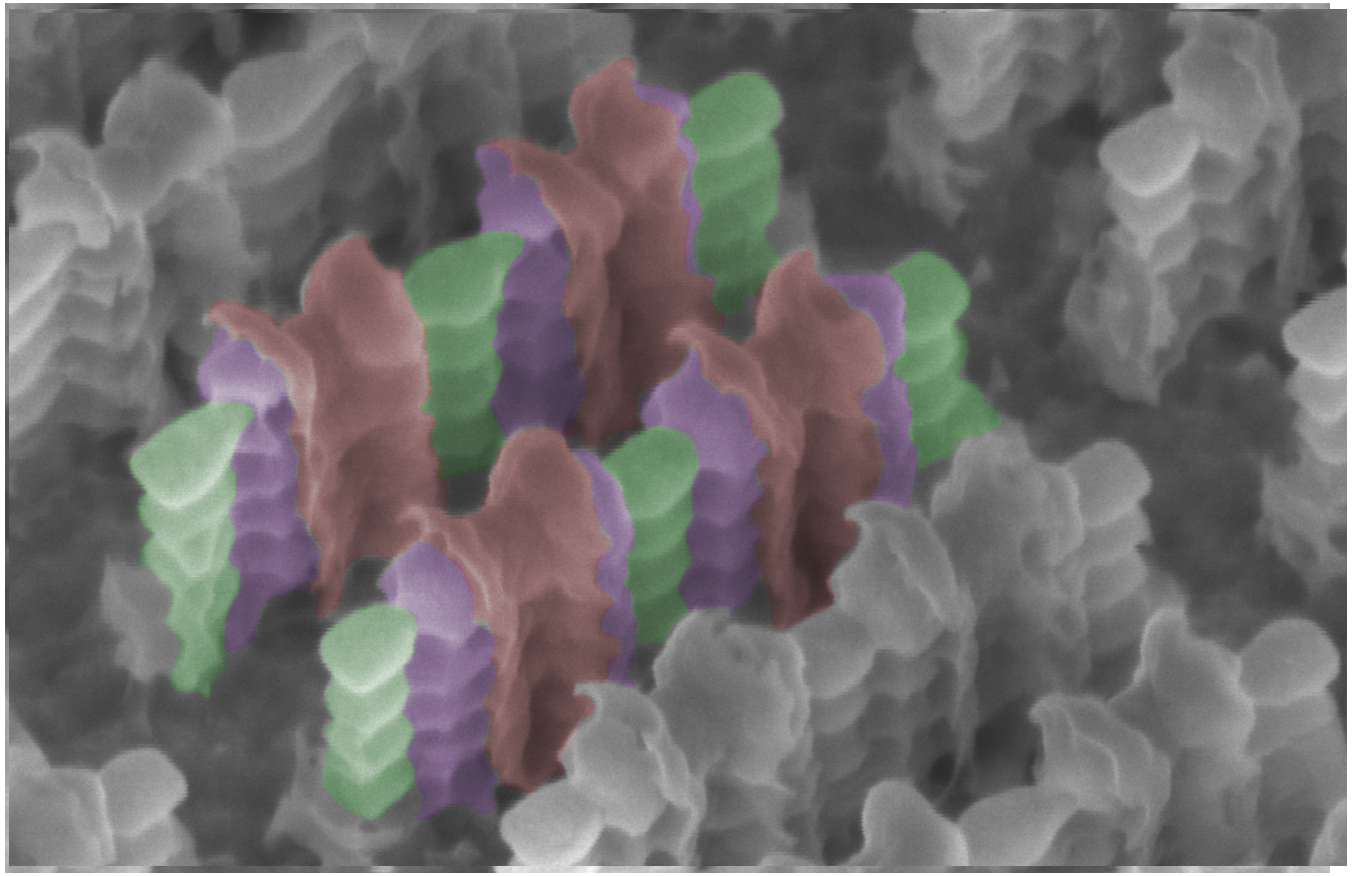
Double gate SiNW FET



- *Electrostatic doping*
- **Electrically program** the transistor to either **p-type** or **n-type**

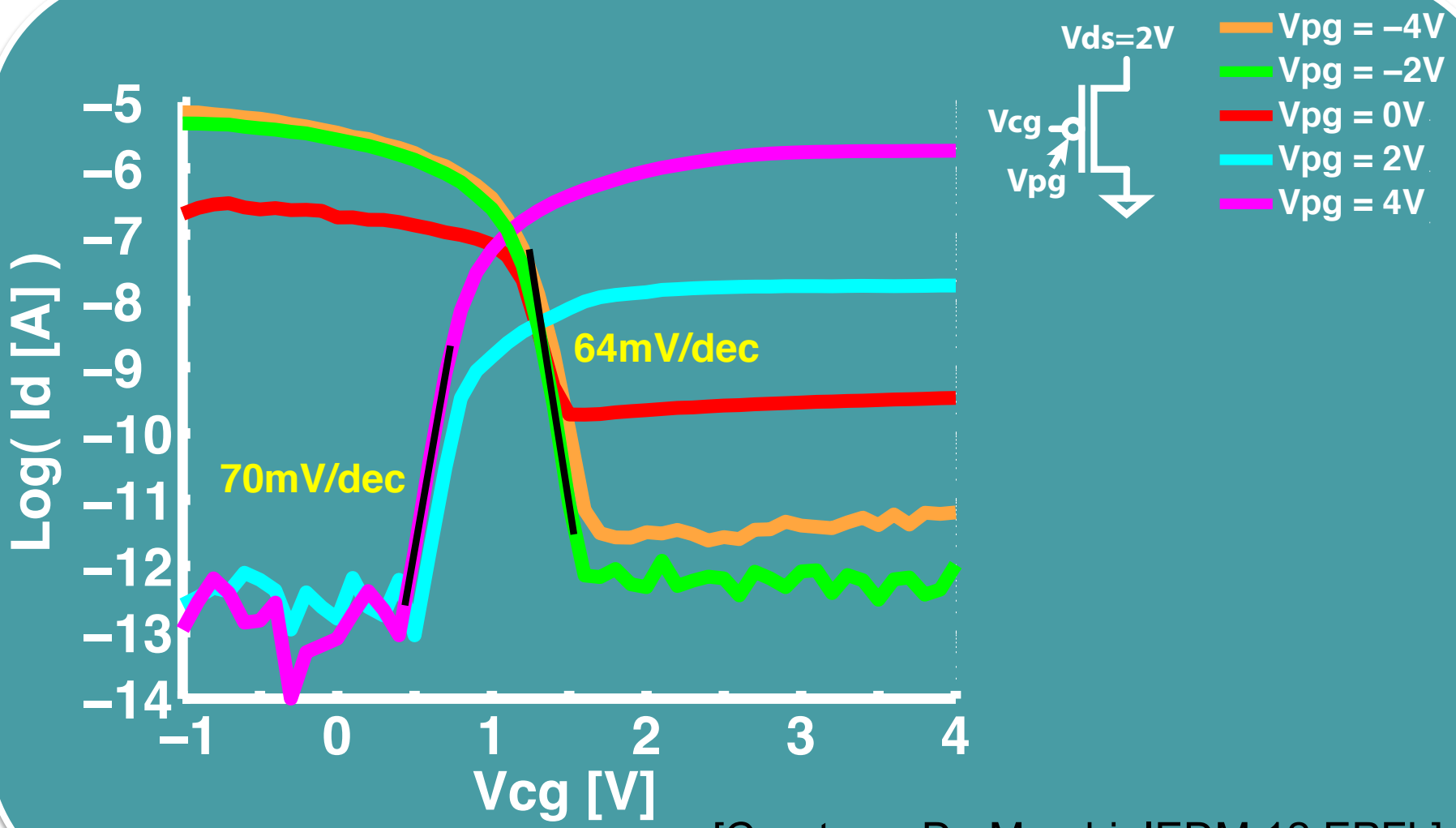
Silicon Nanowire Transistors

- Gate all around transistors
- Double gate to control polarity



[Courtesy: De Marchi, EPFL]

Device I_d/V_{cg}

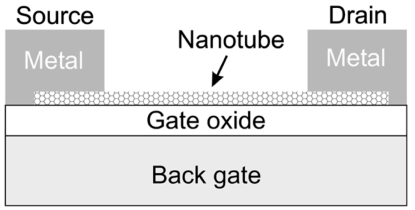


[Courtesy: De Marchi, IEDM 12 EPFL]

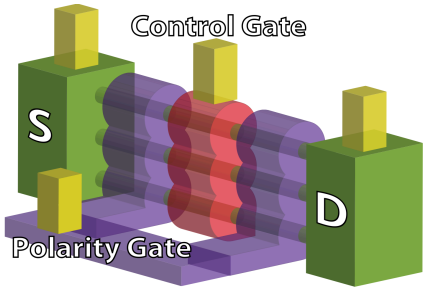
Logic level abstraction

- Three terminal transistors are switches
  - A loaded transistor is an *inverter*
- Controllable-polarity transistors compare two values
  - A loaded transistor is an *exclusive or* (EXOR)
- The intrinsic higher computational expressiveness leads to more efficient data-path design
- The larger number of terminals must be compensated by smart wiring

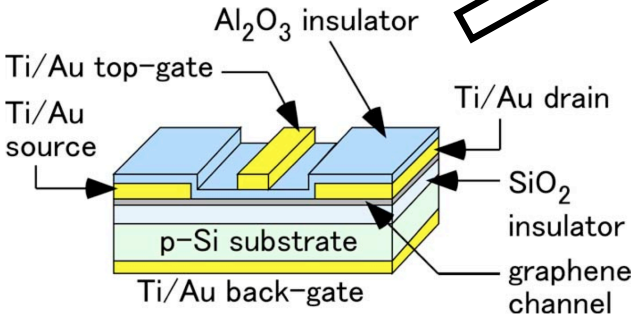
Modeling various emerging nanogates



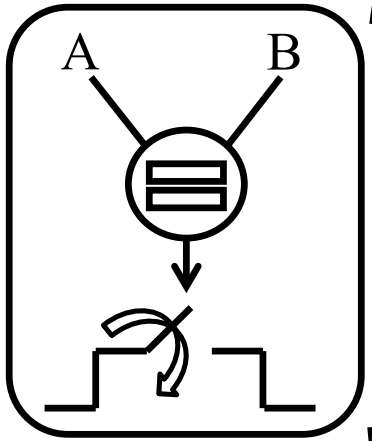
CNFETs

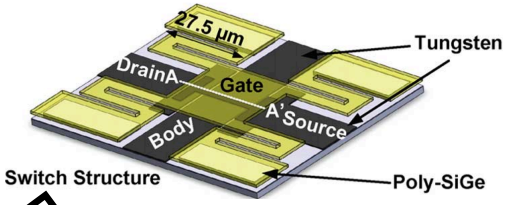


SiNWFETs

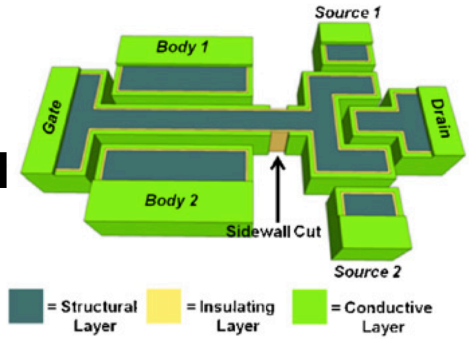


Graphene FETs

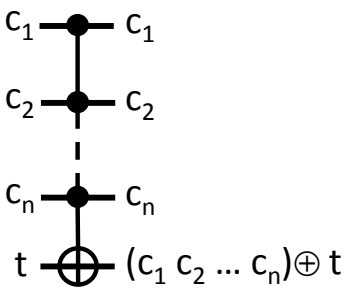




4T Nanorelays



6T Nanorelays



Reversible Logic

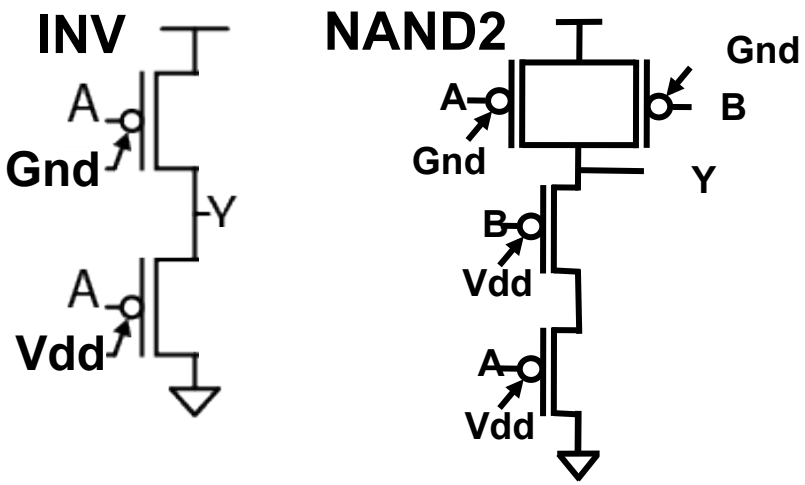
Outline

- Introduction
- Technological innovations and motivation
  - Emerging nanotechnologies and devices
- **Design with emerging technologies**
  - **Physical and logic synthesis**
- The majority paradigm in logic synthesis
  - Models, algorithms and tools
- Conclusions

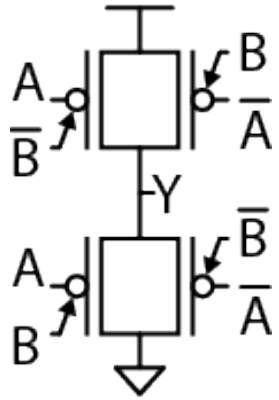
Logic cell design

- CMOS complementary logic is efficient only for negative-unate functions (INV, NAND, NOR…etc)
- Controllable-polarity logic is efficient for all functions
- Best for XOR-dominated circuits (binate functions)

Negative Unate functions	Binate functions



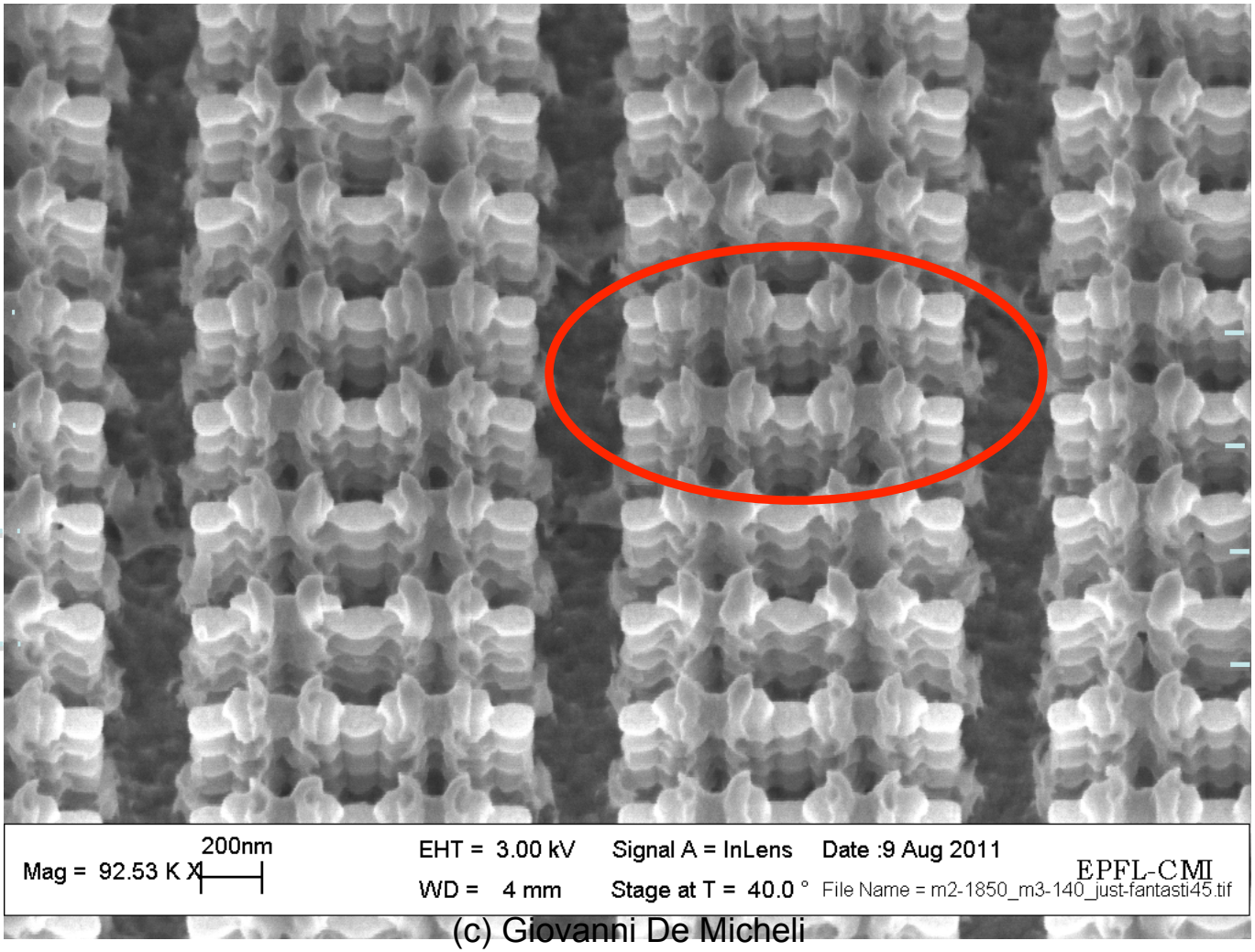
Similar to regular CMOS



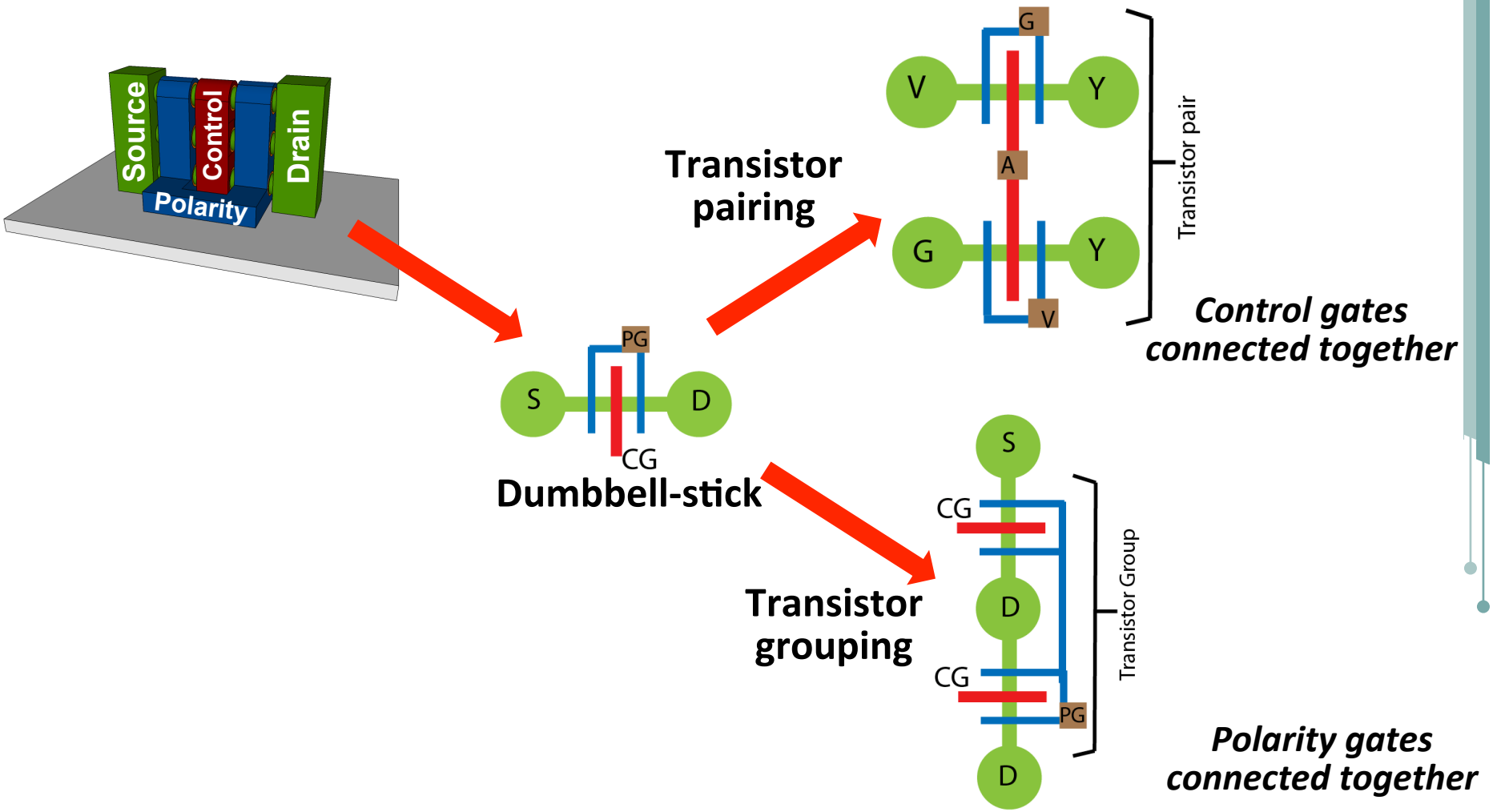
Only 4 transistors when compared to 8 transistors with a regular CMOS

Physical design

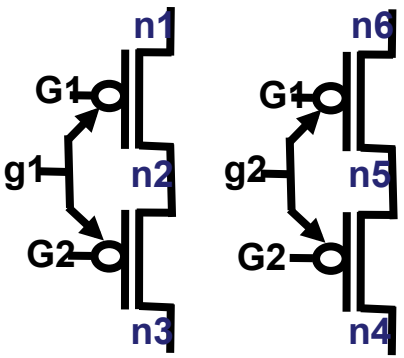
- Sea of Tiles: Homogeneous array of Tiles

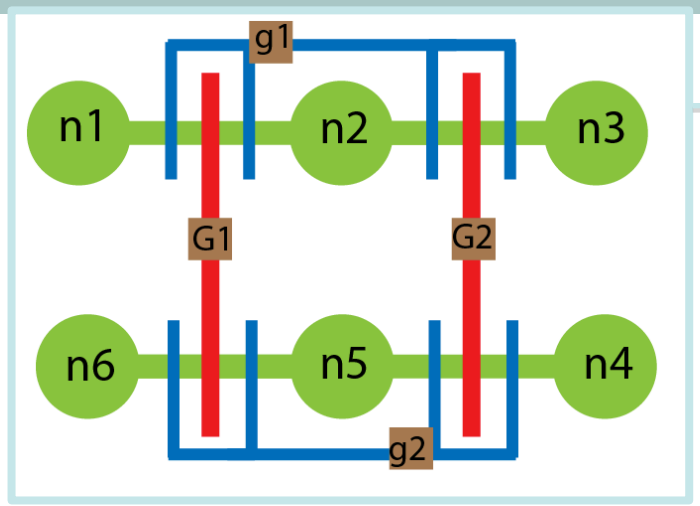


Dumbbell-stick diagrams



Layout abstraction and regularity with *Tiles*

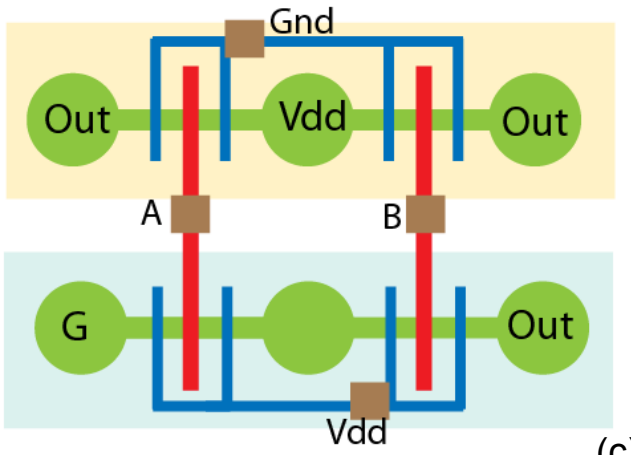




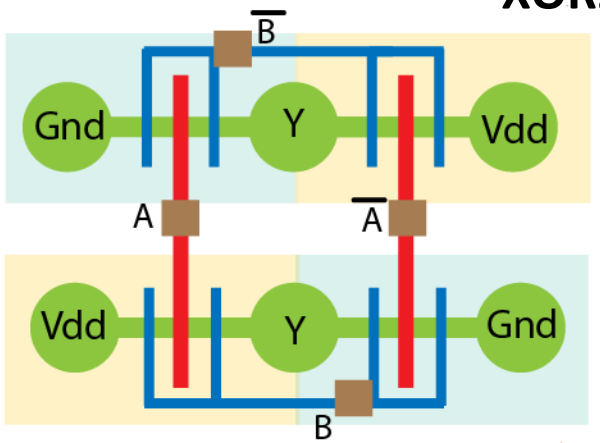
Tile

Two transistor pairs grouped together

NAND2



XOR2

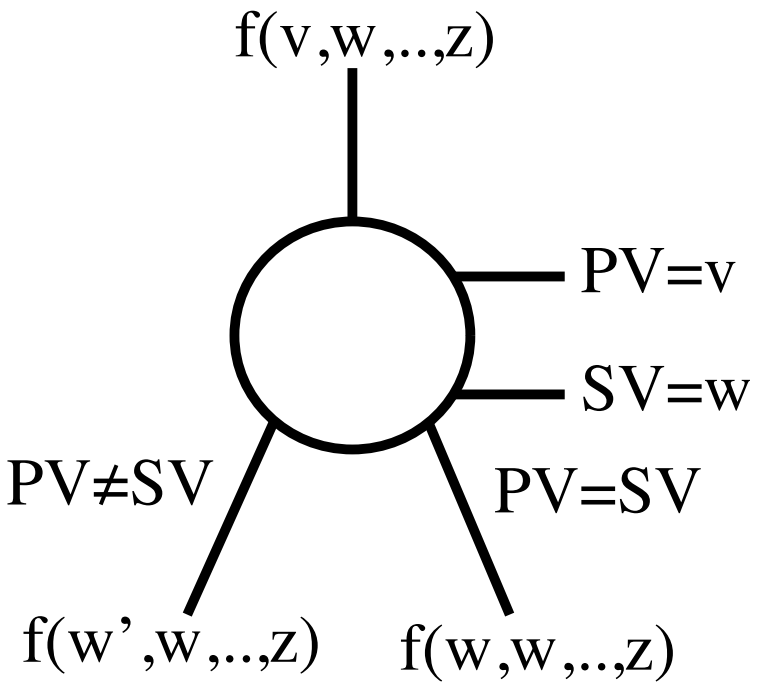


[Courtesy: Bobba, DAC 12]

Biconditional Binary Decision Diagrams

- Native **canonical** data structure for logic design
- *Biconditional* expansion:

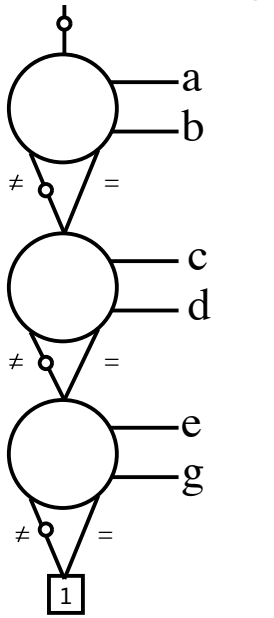
$$f(v,w,..,z) = (v \oplus w) f(w',w,..,z) + (v \overline{\oplus} w) f(w,w,..,z)$$



- Each BBDD node:
  - Has two branching variables
  - Implements the *biconditional* expansion
  - Reduces to Shannon's expansion for single-input functions

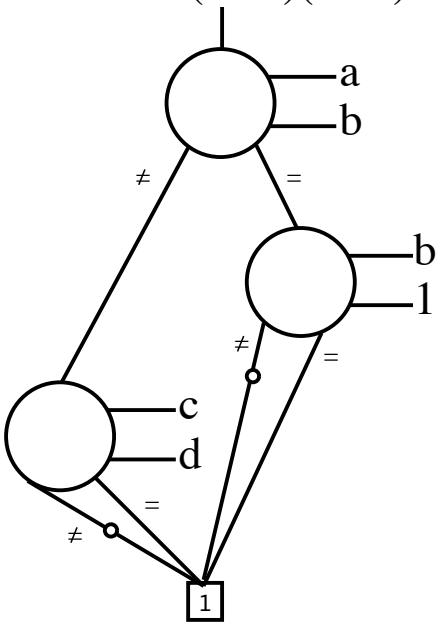
BBDD: Examples

a) f=a⊕b⊕c⊕d⊕e⊕g



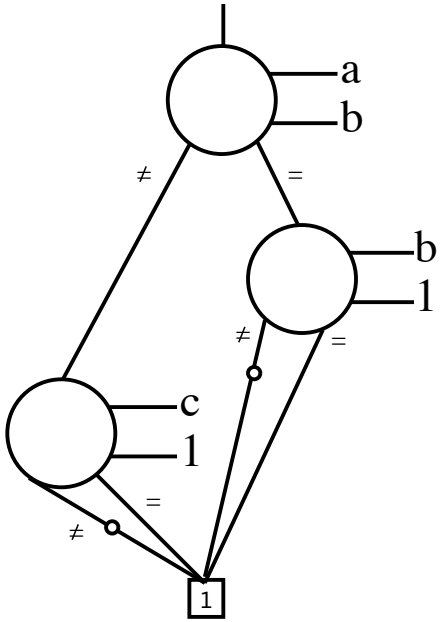
π=(a,b,c,d,e,g)

b) f=ab+(a⊕b)(c⊕̄d)



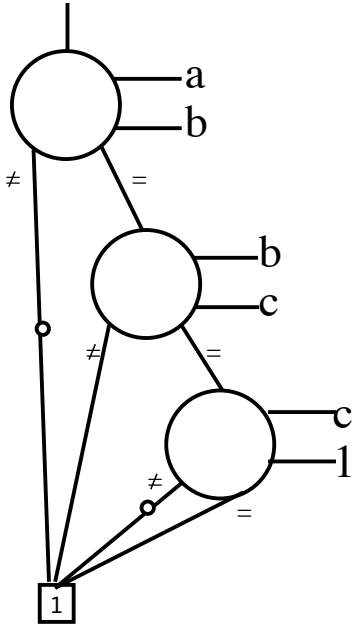
π=(a,b,c,d)

c) f=ab+bc+ac



π=(a,b,c)

d) f=(a⊕̄b)(b+c)



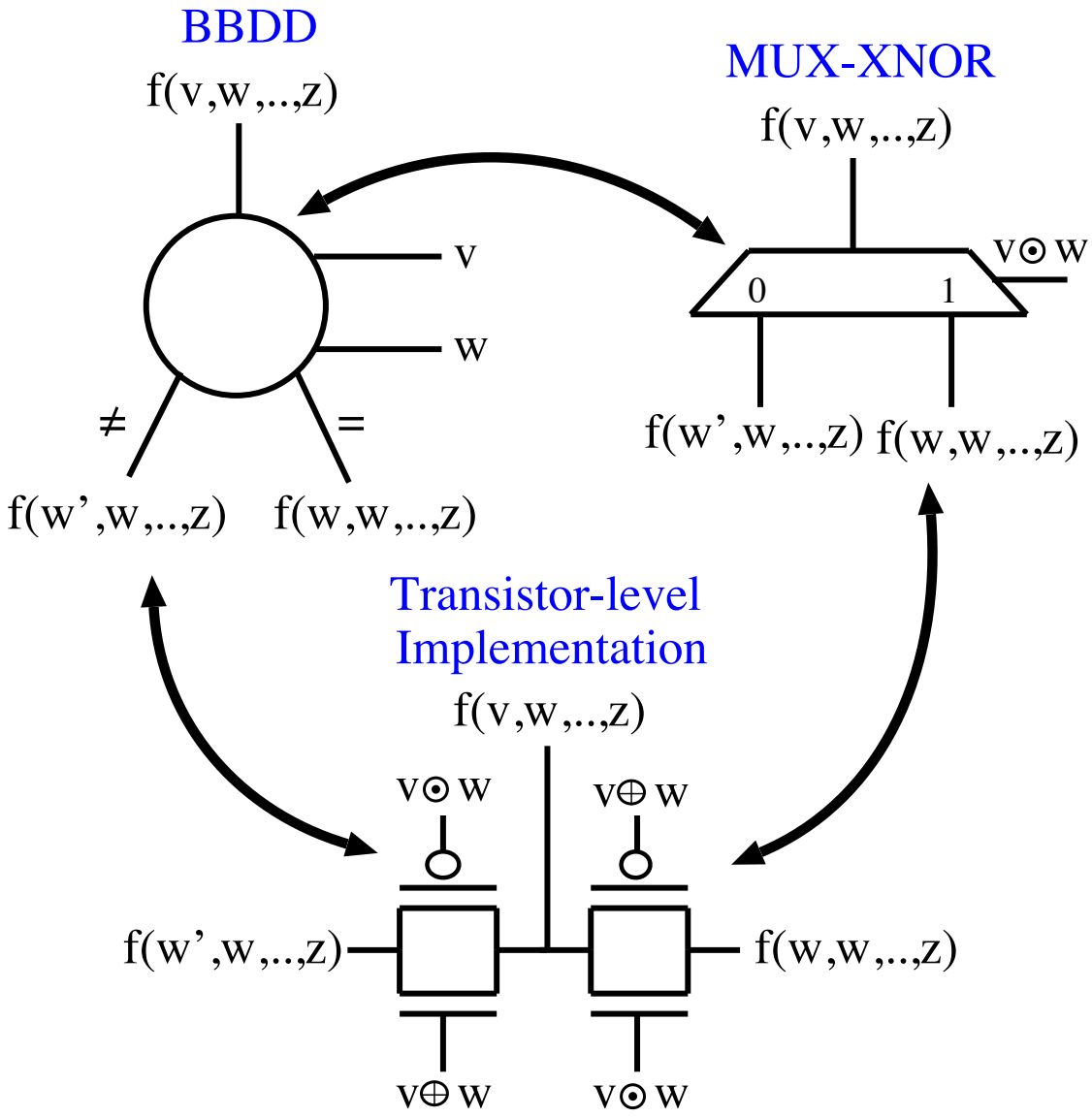
π=(a,b,c)

- The BDD counterparts for these examples have about 50% more nodes!

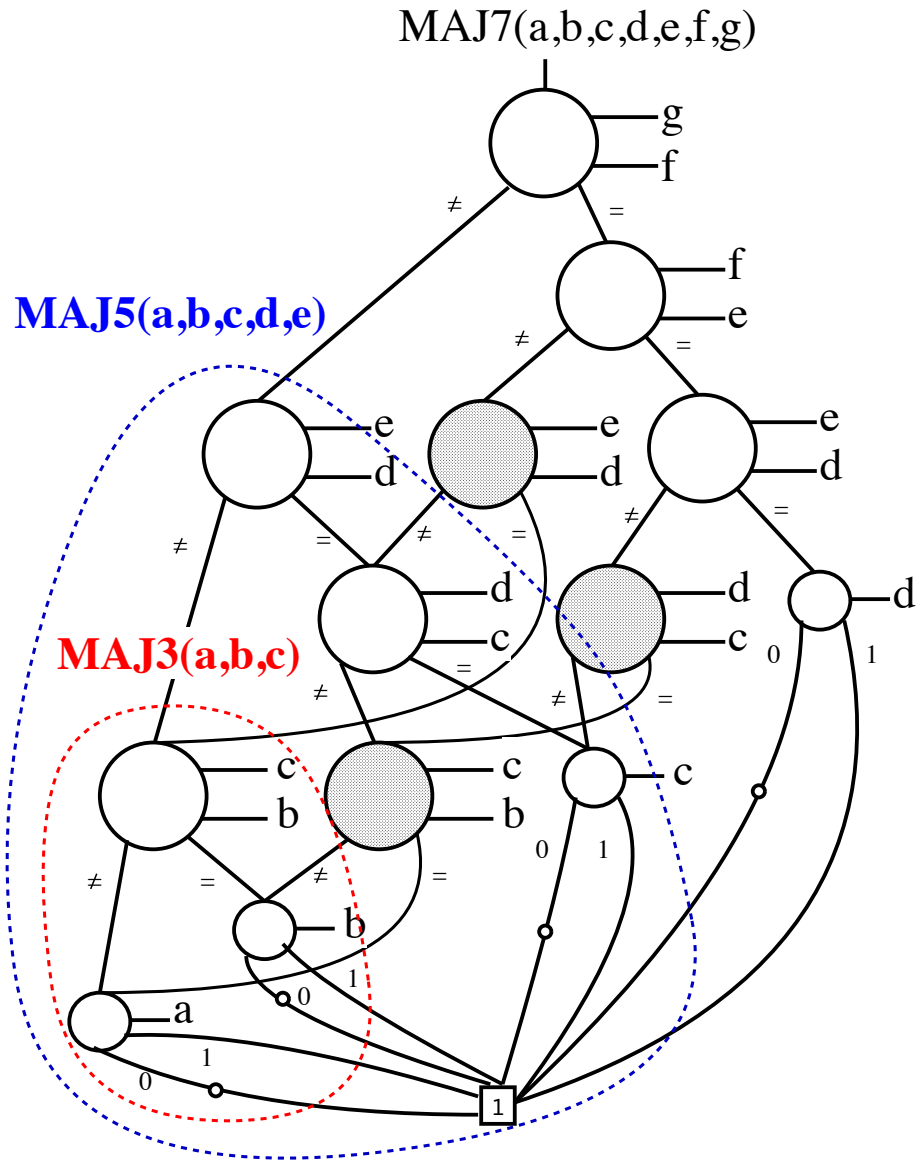
Why BBDDs ?

- BBDDs are the representation of choice for controllable-polarity devices
  - Direct mapping to transistor structures
- BBDDs are very effective for standard CMOS, especially for design of arithmetic circuits
- BBDDs are proven to be more compact for:
  - Adders:
    - BBDD best size: 3n +1
    - BDD best size: 5n +2
  - Majority:
    - BBDD size: $0.25\,(n^2 + 7)$
    - BDD size: $\lceil 0.5n \rceil\,(n - \lceil 0.5n \rceil + 1) + 1$

Efficient Direct Mapping of BBDD Nodes



BBDDs are Compact (Majority Function)



Number of nodes of MAJ(n):

$$0.25(n^2+7)$$

MAJ(3): 4 (including sink)

MAJ(5): 8 (including sink)

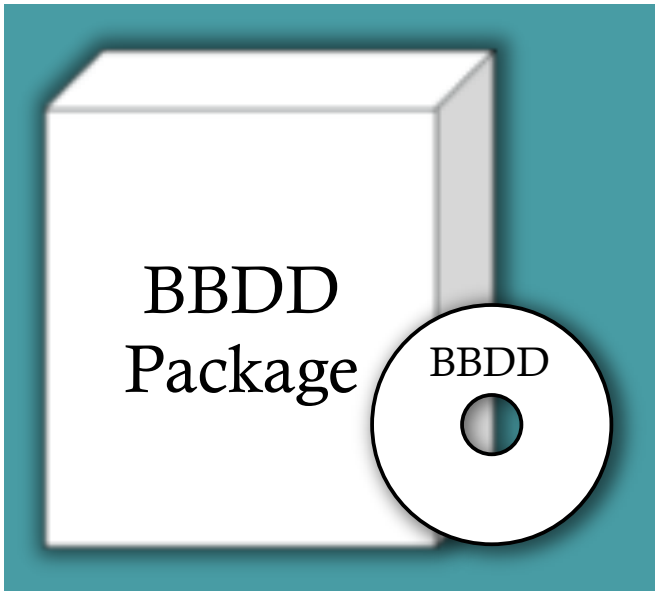
MAJ(7): 14 (including sink)

....

The BBDD optimization tool

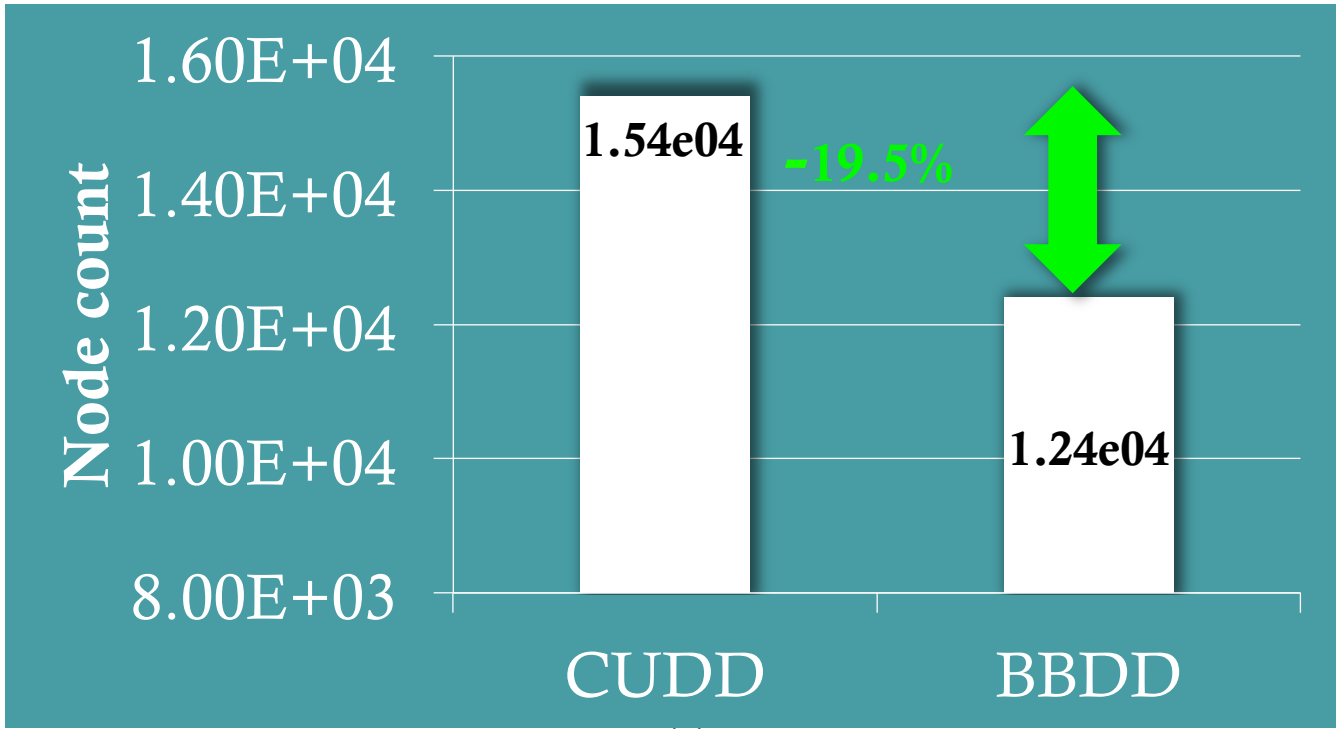
- Unique table to store BBDD nodes
- Recursive formulation of Boolean operations
- Performance-oriented memory management
- Chain variable reordering

http://lsi.epfl.ch/BBDD



Experimental results

- We implemented a BBDD package in C language
  - Comparison with CUDD (BDD)
- Both CUDD and BBDD first build the DDs and then apply sifting (no dynamic reordering)



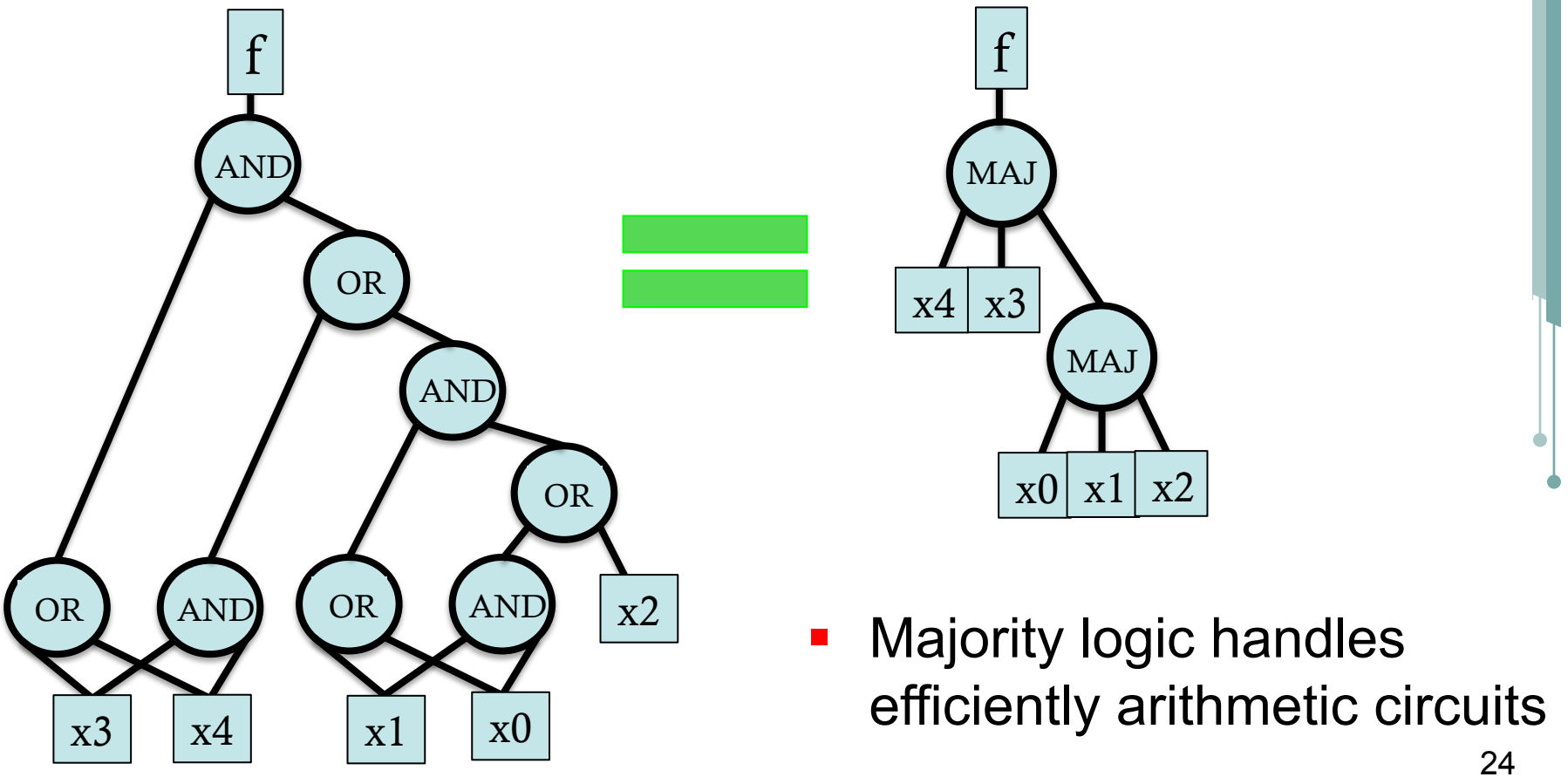
Also 1.63x speedup for arithmetic intensive circuits

Outline

- Introduction
- Technological innovations and motivation
  - Emerging nanotechnologies and devices
- Design with emerging technologies
  - Physical and logic synthesis
- **The majority paradigm in logic synthesis**
  - **Models, algorithms and tools**
- Conclusions

Why Majority Logic?

- Majority logic is a powerful generalization of AND/ORs
  - MAJ(a,b,c)=ab+ac+bc. MAJ(a,b,1)=a+b. MAJ(a,b,0)=ab.
- Unlocks optimization opportunities not apparent before



- Majority logic handles efficiently arithmetic circuits

Synthesis Motivation for Majority

MCNC.GENLIB + MIN3

```
module ANDOR (
   x0, x1, x2, x3, x4,
   z0  );
 input  x0, x1, x2, x3, x4;
 output z0;
 wire n6, n7, n8, n9, n10, n11;
   nor2  g0(.a(x4), .b(x3), .O(n6));
   nand2 g1(.a(x4), .b(x3), .O(n7));
   inv1  g2(.a(x2), .O(n8));
   inv1  g3(.a(x1), .O(n9));
   nand2 g4(.a(n9), .b(n8), .O(n10));
   inv1  g5(.a(x2), .O(n11));
   nand2 g6(.a(x1), .b(x0), .O(n12));
   nand2 g7(.a(n12), .b(n11), .O(n13));
   nand2 g8(.a(n13), .b(n10), .O(n14));
   aoi21 g9(.a(n14), .b(n7), .c(n6), .O(z0));
endmodule
```

Area=18 Delay=4.60

```
module MAJ (
   x0, x1, x2, x3, x4,
   z0  );
 input  x0, x1, x2, x3
 output z0;
 wi
   in    (x3), .O(n6));
   inv1  g1(.a(x4), .O(n7));
   min3  g2(.a(x2), .b(x1), .c(x0), .O(n8));
   min3  g3(.a(n8), .b(n7), .c(n6), .O(z0));
endmodule
   nand2 g5(.a(x4), .b(x3), .O(n11));
   nand2 g6(.a(n11), .b(n10), .O(z0));
endmodule
```

Area=12 Delay=3.20

Area ↓ Delay =

Area ↓ Delay ↓

How to Exploit Majority Logic?

We want good and scalable methods for manipulating MAJ

State-of-the-art

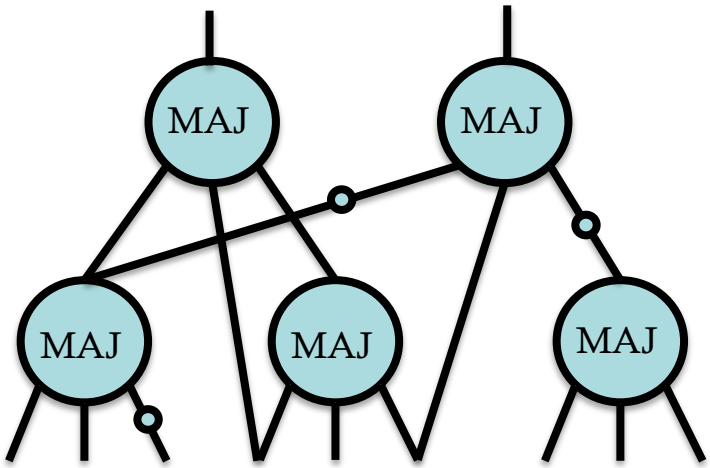
- AND-OR Inverter Graphs (AOIGs)
- Use traditional Boolean algebra axioms and theorems to manipulate & optimize AOIGs

For majority

- Majority Inverter Graphs (MIGs)
- New Boolean algebra to deal natively with majority and inverters

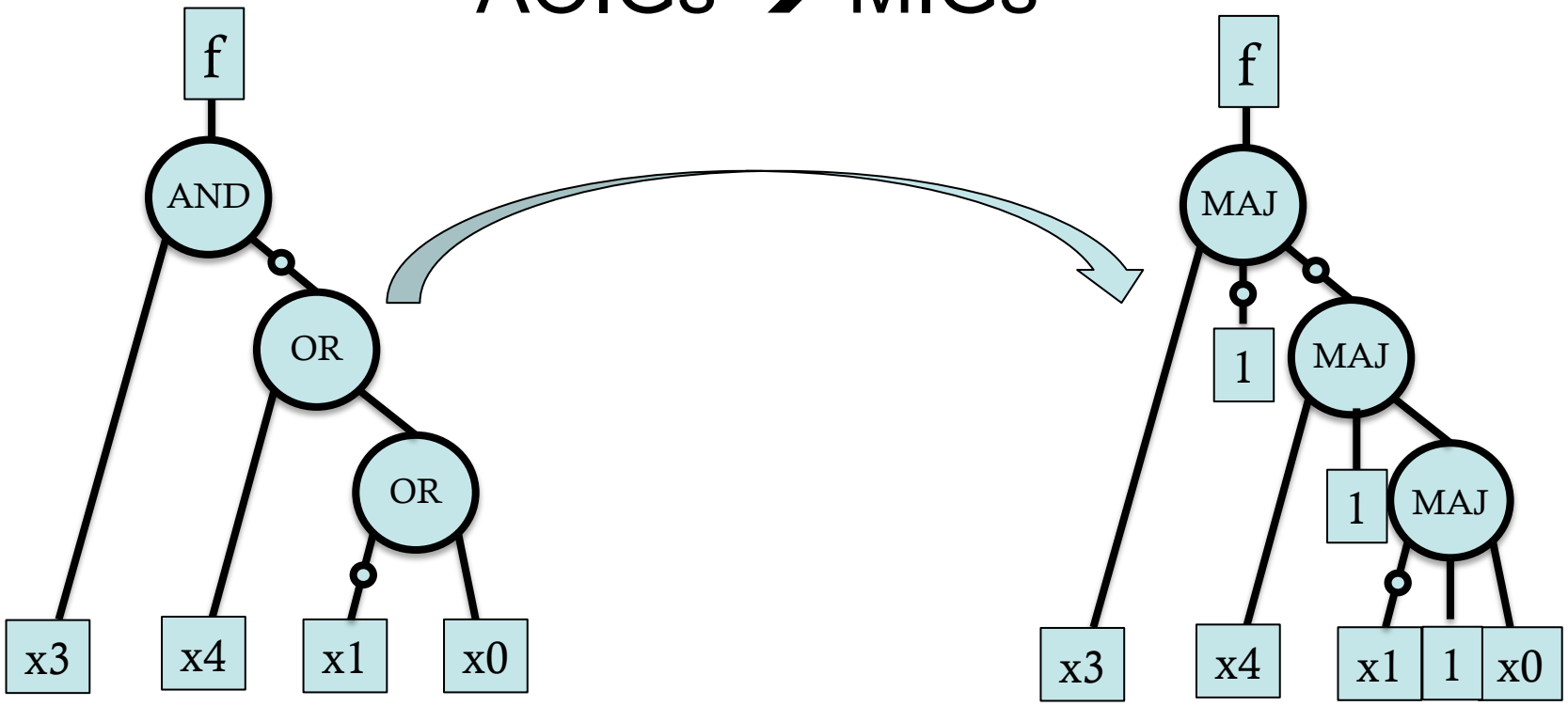
Majority-Inverter Graph

Definition: An MIG is a logic network consisting of 3-input majority nodes and regular/complemented edges



MIG Properties

AOIGs ➔ MIGs



MIGs **include** AOIGs **include** AIGs

Manipulating MIGs: MIG Boolean Algebra

Ω

1- Commutativity: $M(x, y, z) = M(y, x, z) = M(z, y, x)$

2- Majority: $if(x = y),\ M(x, y, z) = x = y$

$if(x = y'),\ M(x, y, z) = z$

3- Associativity: $M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))$

4- Distributivity: $M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)$

5- Inverter Propagation: $M'(x, y, z) = M(x', y', z')$

Theorem: (B,M,',0,1) subject to axioms in Ω is a Boolean algebra

MIG Boolean Algebra

1- **Commutativity**: $M(x, y, z) = M(y, x, z) = M(z, y, x)$

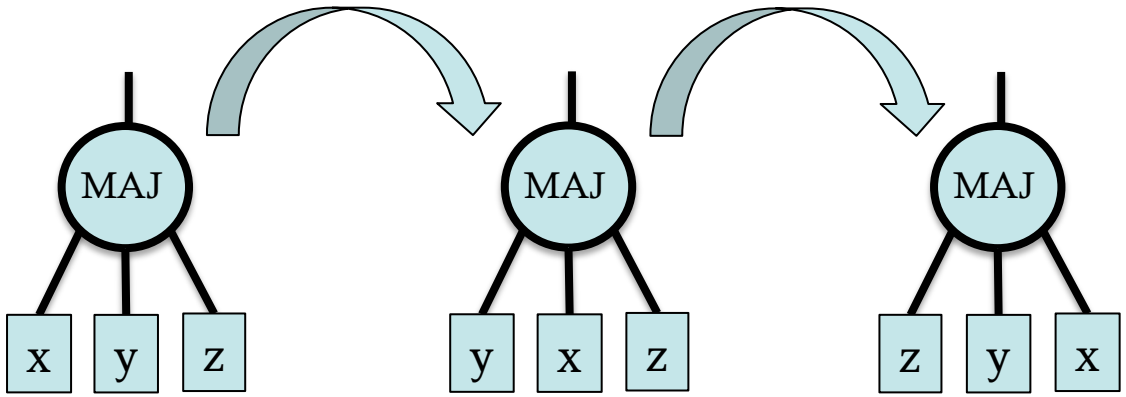
2- Majority: $if(x = y), M(x, y, z) = x = y$

$if(x = y'), M(x, y, z) = z$

3- Associativity: $M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))$

4- Distributivity: $M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)$

5- Inverter Propagation: $M'(x, y, z) = M(x', y', z')$



MIG Boolean Algebra

1- Commutativity: $M(x, y, z) = M(y, x, z) = M(z, y, x)$

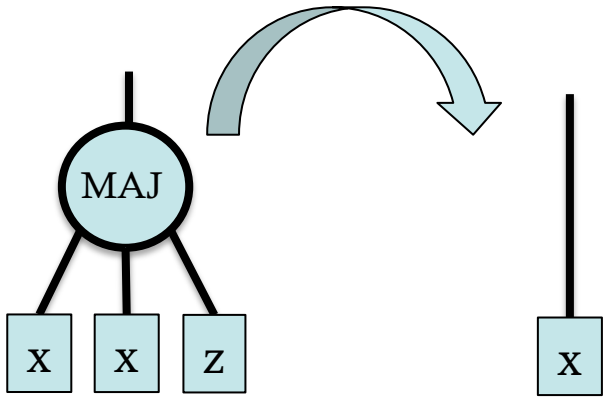
2- **Majority:** $if(x = y),\ M(x, y, z) = x = y$

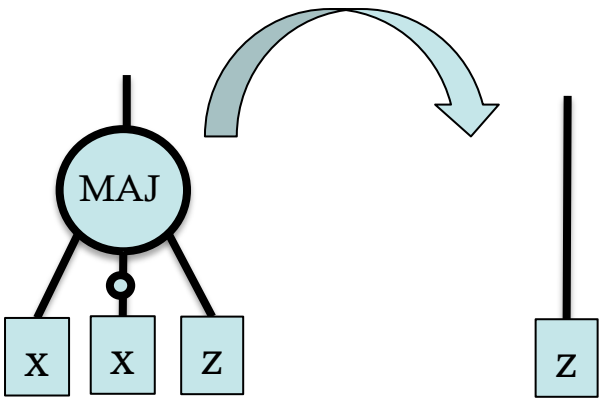
$if(x = y'),\ M(x, y, z) = z$

3- Associativity: $M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))$

4- Distributivity: $M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)$

5- Inverter Propagation: $M'(x, y, z) = M(x', y', z')$





MIG Boolean Algebra

1- Commutativity: $M(x, y, z) = M(y, x, z) = M(z, y, x)$

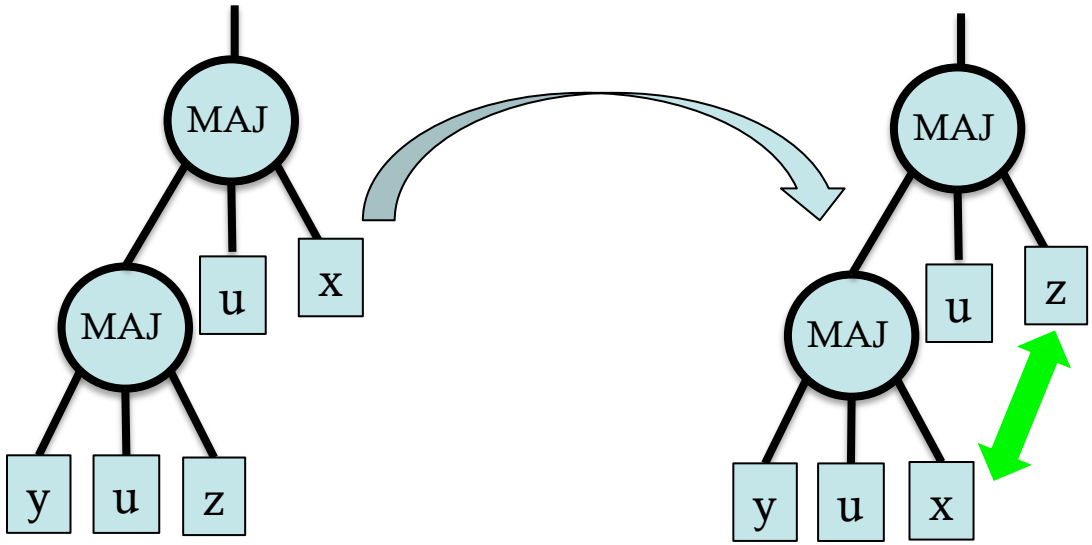
2- Majority: $if(x = y), M(x, y, z) = x = y$

$if(x = y'), M(x, y, z) = z$

3- **Associativity:** $M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))$

4- Distributivity: $M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)$

5- Inverter Propagation: $M'(x, y, z) = M(x', y', z')$



MIG Boolean Algebra

1- Commutativity: $M(x, y, z) = M(y, x, z) = M(z, y, x)$

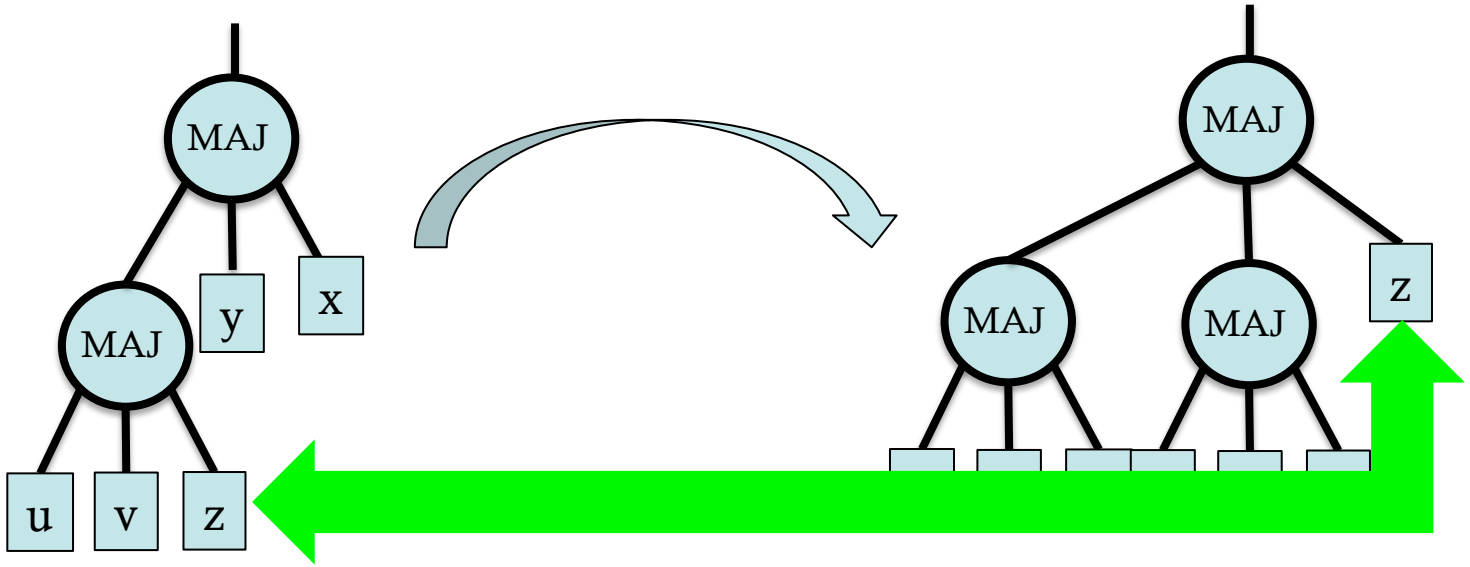
2- Majority: $if(x = y), M(x, y, z) = x = y$

$if(x = y'), M(x, y, z) = z$

3- Associativity: $M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))$

4- **Distributivity:** $M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)$

5- Inverter Propagation: $M'(x, y, z) = M(x', y', z')$



MIG Boolean Algebra

1- Commutativity: $M(x, y, z) = M(y, x, z) = M(z, y, x)$

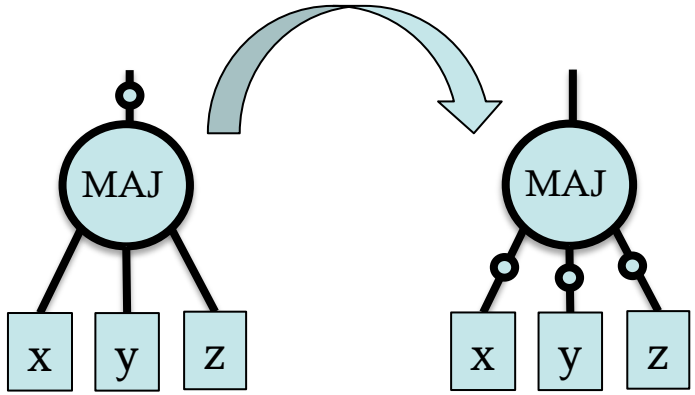
2- Majority: $if(x = y), M(x, y, z) = x = y$

$if(x = y'), M(x, y, z) = z$

3- Associativity: $M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))$

4- Distributivity: $M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)$

5- **Inverter Propagation:** $M'(x, y, z) = M(x', y', z')$



Properties

- The Boolean algebra with axioms Ω is:
  - Sound:
    - If a formula is derivable from Ω, then it is valid
  - Complete:
    - Each valid formula is derivable from Ω
- Any MIG configuration is reachable from any other equivalent MIG configuration

Enhancing Ω

- Powerful macro-transformations: Ψ
- Serve as shortcut to longer sequences in Ω
- Define: $z_{x/y}$ as replace x by y in all appearances in z

Ψ

1- Relevance: $M(x, y, z) = M(x, y, z_{x/y'})$

2- Complementary Associativity:

$M(x, u, M(y, u', z)) = M(x, u, M(y, x, z))$

3- Substitution:

$M(x, y, z) =$

$M(v, M(v', M_{v/u}(x, y, z), u), M(v', M_{v/u'}(x, y, z), u'))$

Optimizing MIGs

Ω

1- Commutativity: $M(x, y, z) = M(y, x, z) = M(z, y, x)$

2- Majority: $if(x = y),\ M(x, y, z) = x = y$

$if(x = y'),\ M(x, y, z) = z$

3- Associativity: $M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))$

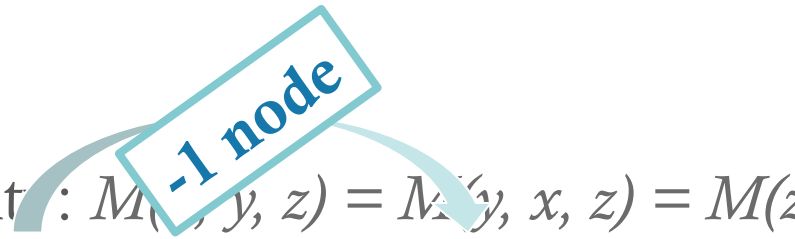
4- Distributivity: $M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)$

5- Inverter Propagation: $M'(x, y, z) = M(x', y', z')$

- By using Ω and Ψ we optimize an MIG
- What we really care about?
- Area ➔ MIG size
- Delay ➔ MIG depth
- Power ➔ MIG switching activity

MIG Size Optimization

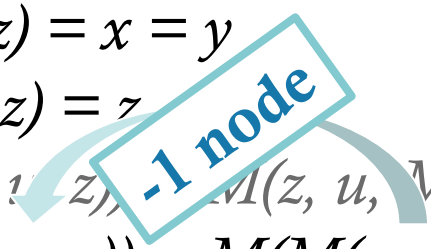
- How to reduce the number of nodes in an MIG?
- Let's see what comes handy from Ω:



1- Commutativity: $M(x, y, z) = M(y, x, z) = M(z, y, x)$

2- Majority: $if(x = y),\ M(x, y, z) = x = y$

$if(x = y'),\ M(x, y, z) = z$



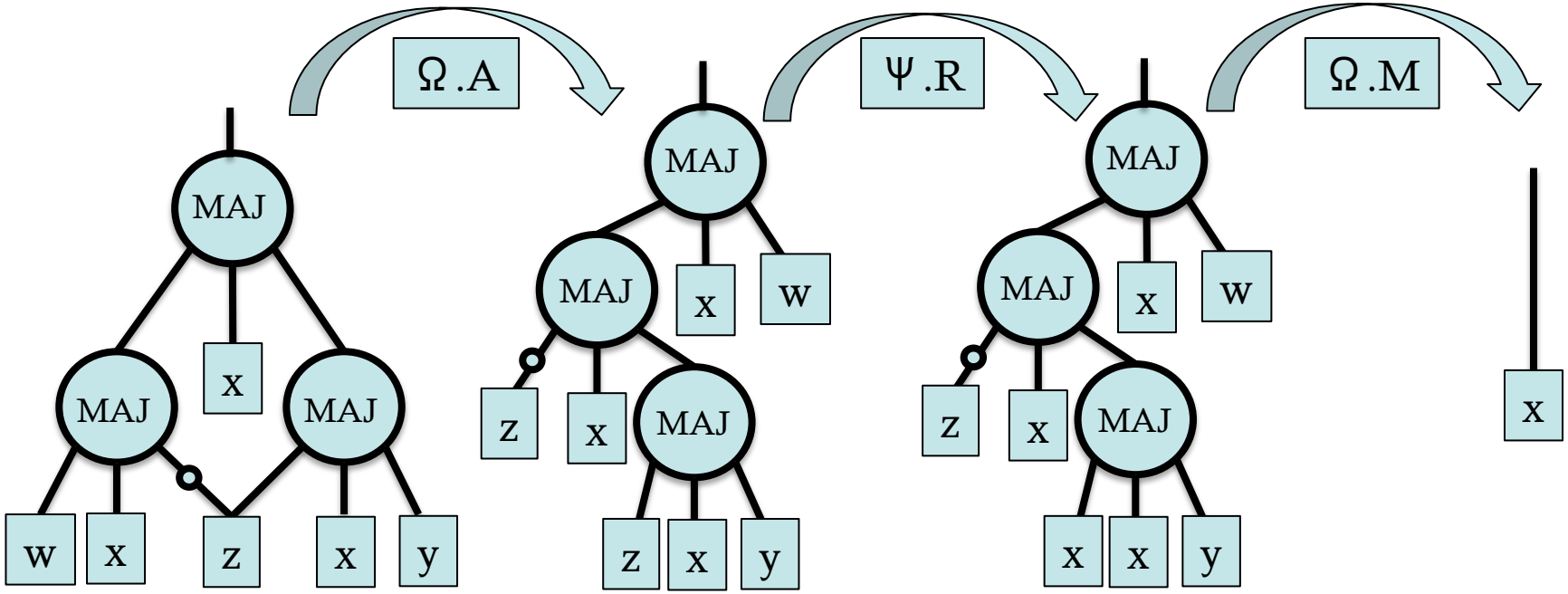
3- Associativity: $M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))$

4- Distributivity: $M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)$

5- Inverter Propagation: $M'(x, y, z) = M(x', y', z')$

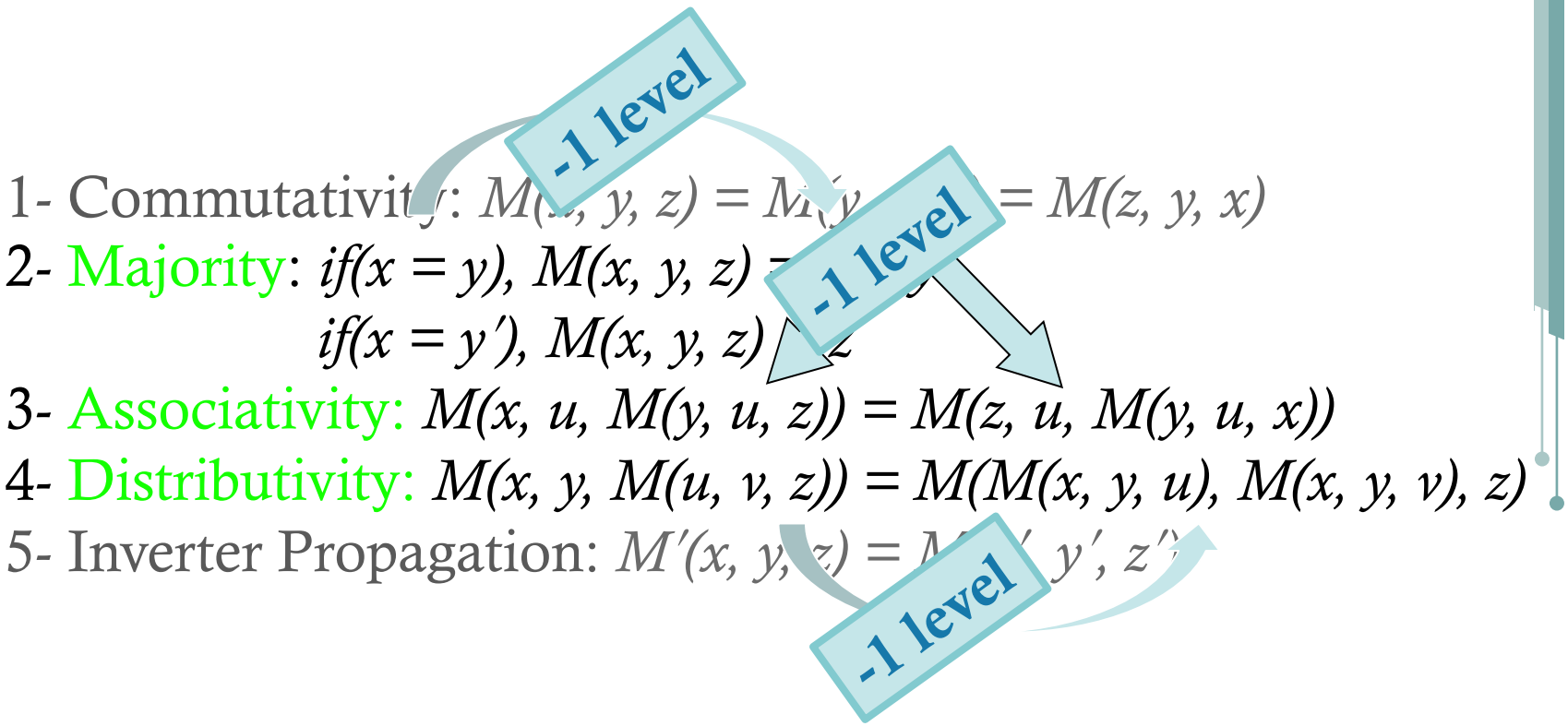
MIG Size Optimization

- How to enable majority and distributivity laws for node reduction?
- Other rules from Ω and Ψ to reshape the MIG
- Reshape rationale: move closer similar/equivalent variables/ nets



MIG Depth Optimization

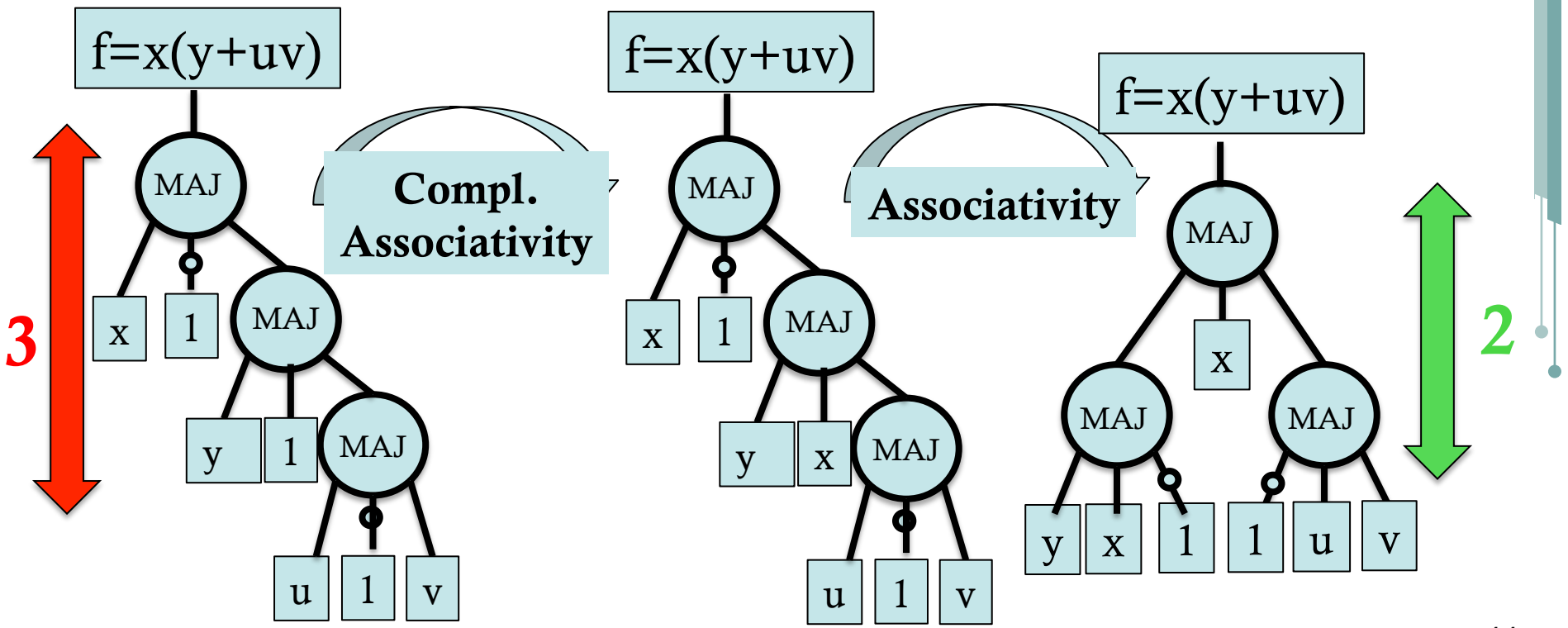
- How to reduce the depth of an MIG?
- Let's see what comes handy from Ω:



1- Commutativity: $M(x, y, z) = M(y, x, z) = M(z, y, x)$
2- Majority: $if(x = y), M(x, y, z) = x$
$if(x = y'), M(x, y, z) = z$
3- Associativity: $M(x, u, M(y, u, z)) = M(z, u, M(y, u, x))$
4- Distributivity: $M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)$
5- Inverter Propagation: $M'(x, y, z) = M(x', y', z')$

MIG Depth Optimization

- Rationale: move critical variables closer to the outputs via associativity, distributivity and majority rules
- Reshaping the MIG with other Ω rules

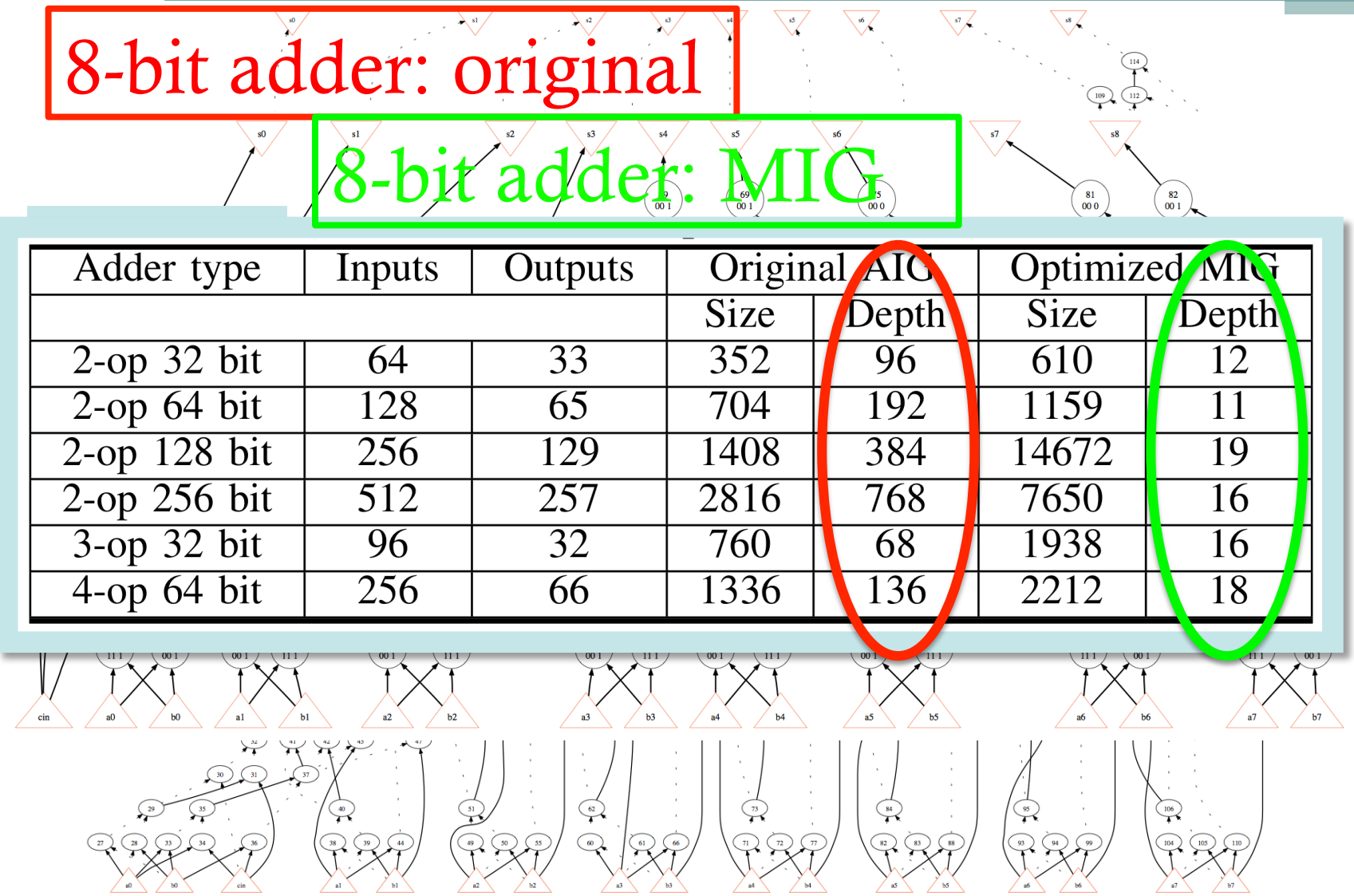


MIG Depth Optimization: Adders

8-bit adder: original

8-bit adder: MIG

Adder type	Inputs	Outputs	Original AIG		Optimized MIG	
			Size	Depth	Size	Depth
2-op 32 bit	64	33	352	96	610	12
2-op 64 bit	128	65	704	192	1159	11
2-op 128 bit	256	129	1408	384	14672	19
2-op 256 bit	512	257	2816	768	7650	16
3-op 32 bit	96	32	760	68	1938	16
4-op 64 bit	256	66	1336	136	2212	18



MIG Activity Optimization

- How to reduce the switching activity of an MIG?
- We want to make the switching probability of nodes close to 0
- Solution: substitute variables with p~0.5 with other having p~0 or p~1
- How to make this? Let's see what comes handy from Ψ:

if $|p(y) - 0.5| > |p(x) - 0.5|$

Ψ

1- Relevance: $M(x, y, z) = M(x, y, z_{x/y'})$

if $|p(x) - 0.5| > |p(u) - 0.5|$

2- Comp

$M(x, u, M(y, u', z)) = M(x, u, M(y, x, z))$

3- Subst

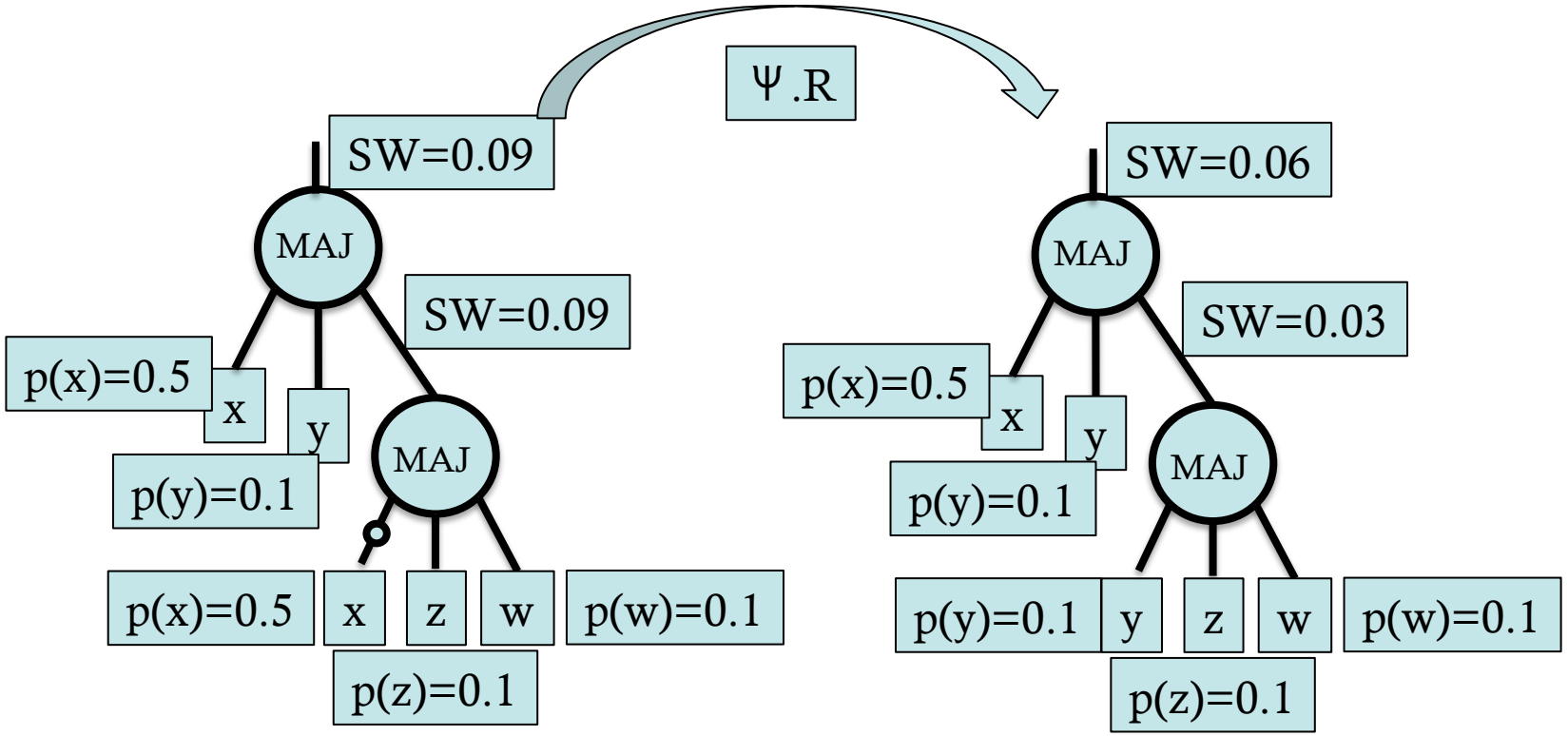
if $|p(u) - 0.5| > |p(v) - 0.5|$
+ extra nodes overhead

$M(x, y, z$

$M(v, M(v', M_{v/u}(x, y, z), u), M(v', M_{v/u'}(x, y, z), u'))$

MIG Activity Optimization

- How to enable switching activity reduction?
- Rationale: same as size and activity but oriented at reducing the switching probability

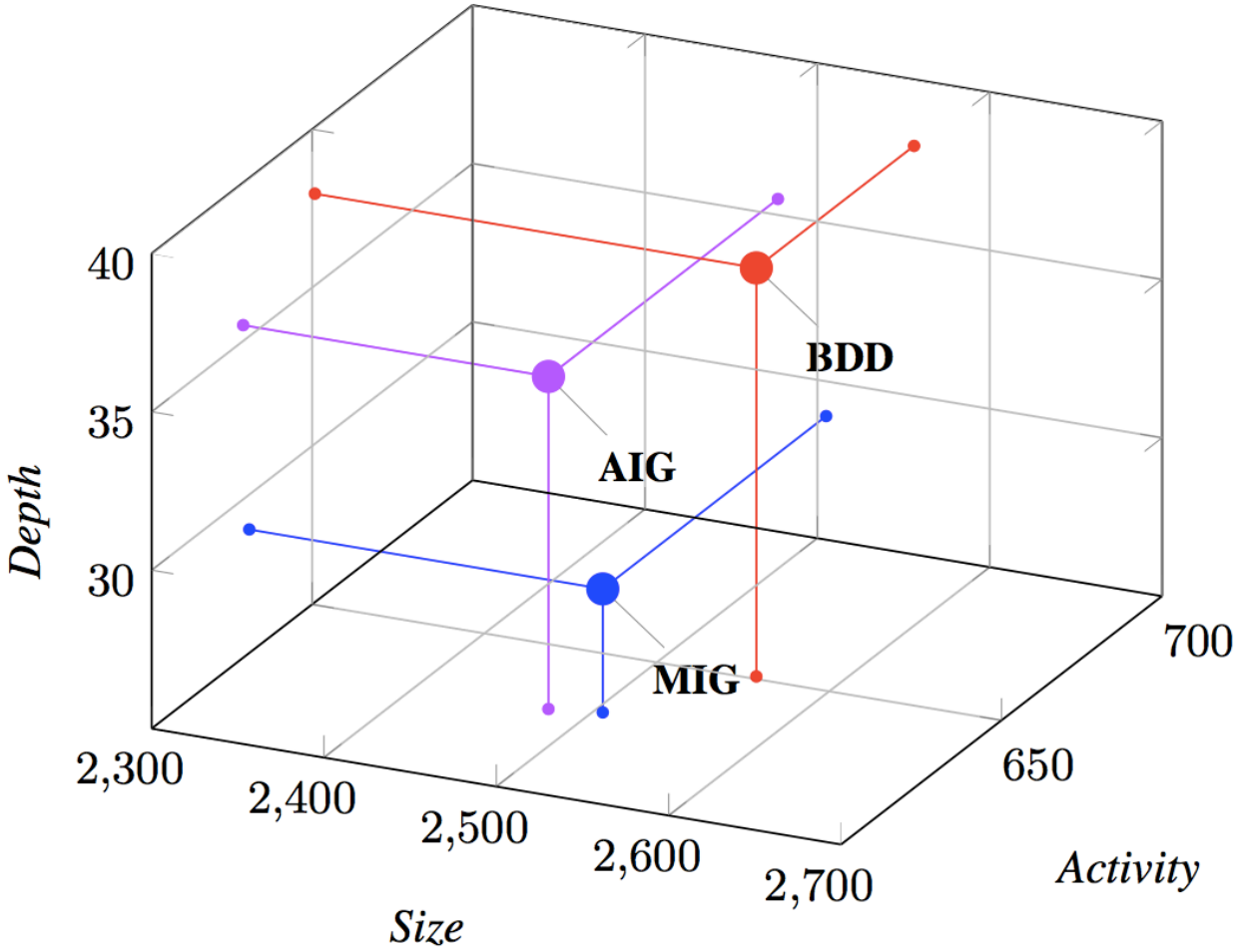


Majority-based synthesis: MIGthy

- **MIGhty:** a logic manipulation package for MIG
  - *MIGhty* reads and writes Verilog
  - Different optimization strategies (depth/area/activity)
  - Hybrid optimization: depth-oriented interlaced with area/power recovery phases
- MCNC, IWLS'05, arithmetic HDL benchmarks
  - Comparison with ABC, BDS and commercial synthesis tool
  - First set of experiments: pure logic optimization
  - Second set of experiments: complete design flow (logic optimization + technology mapping + physical design)

Experimental Results: MCNC circuits

Logic Synthesis: MIGthy



MIGs & AIGs better than BDDs

MIGs size & activity ~ AIGs

MIGs depth -20% w.r.t AIGs

CMOS Design Results

Advanced 22nm CMOS MIG as front-end to LS & PD

Behavioral

```
module div32 (a, b, quotient_uns, quotient_tc, remainder_uns,
 remainder_tc);
 parameter width = 32;
 input [width-1 :
 output [width-1 :
 output signed [wi                                    er_tc;
 // operators for                                     mainder
 assign quotient_u
 assign quotient_tc = $signed(a) / $signed(b);
 assign remainder_uns = a % b;
 assign remainder_tc = $signed(a) % $signed(b);
endmodule
```

Area: 0.21 mm²

Delay: 11.22 ns

GC: 37k

All circuits underwent formal verification with success

Well-established 90nm CMOS MIG as front-end to LS & PD

MIG

```
assign w0 = (w39266 & w27196) | (w39266 & w42866) | (w27196 & w42866);
assign w1 = w44100 & w4464;
assign w2 = ~w19966 & w27946;
assign w3 = ~w22044 & ~w30809;
assign w4 = w42722 & ~b[26];
assign w5 = (~w30032 & w19016) | (~w30032 & w47192) | (w19016 & w47192);
assign w6 = ~w26074 & w5179;
assign w7 = (w42822 &                                    w16107);
assign w8 = ~w35174 &
assign w9 = ~w45553 &
assign w10 = ~w14416
assign w11 = (~w16181                                    18 & w16835);
assign w12 = (w27179                                     w4052);
assign w13 = (w2836 &                                    204);
assign w14 = ~w2529 &
assign w15 = (w43756                                     w36731);
assign w16 = ~w44399
assign w17 = ~w22912
assign w18 = ~w30302
assign w19 = (~w2377                                     & w50584);
assign w20 = w26809 &
assign w21 = (~w43756                                    w3953);
assign w22 = ~w36188 & w1158
assign w23 = ~w33838 & ~w16188;
assign w24 = ~w38787 & w19348;
assign w25 = w42096 & w20651;
assign w26 = (w45543 & w39924) | (w45543 & w25446) | (w39924 & w25446);
assign w27 = ~w34177 & w38267;
assign w28 = ~w10074 & ~w29118;
assign w29 = w25495 & w32055;
assign w30 = (w48190 & w13546) | (w48190 & w19290) | (w13546 & w19290);
```

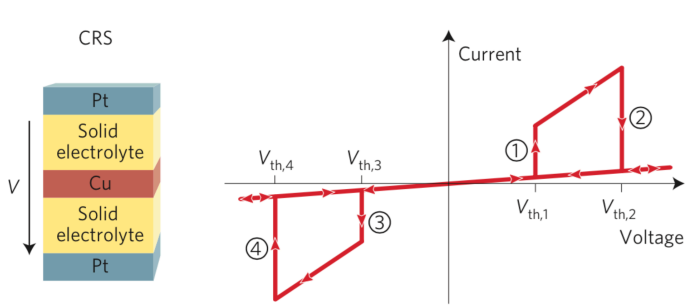
Area: 0.18 mm²

Delay: 10.10 ns

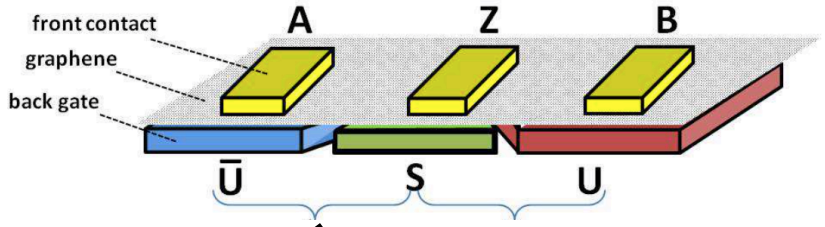
GC: 24k

Both circuits underwent formal verification with success

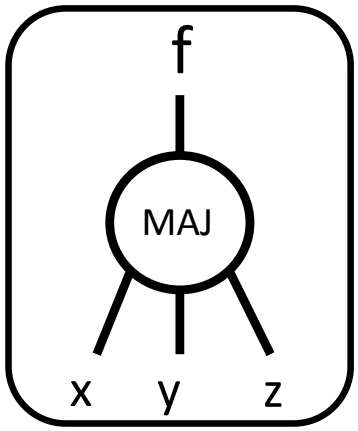
Modeling various emerging nanogates

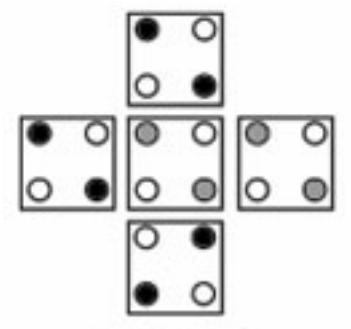


RRAM



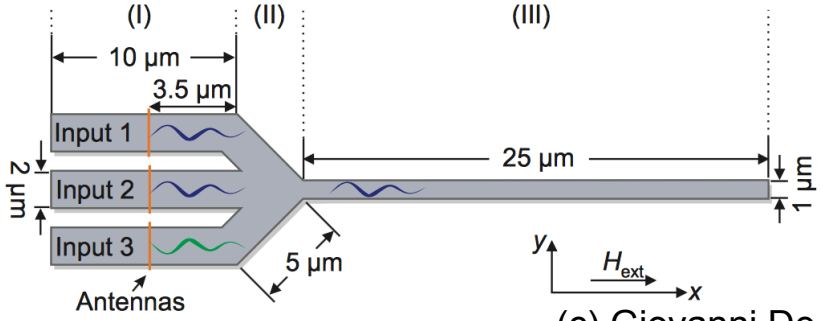
Graphene RG

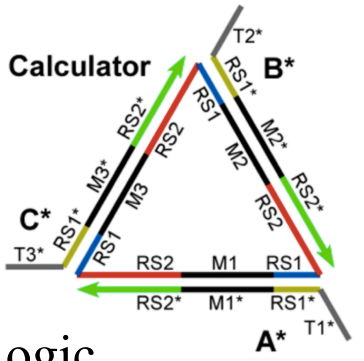




QCA

Spin-Wave Device





DNA Logic

(c) Giovanni De Micheli

Nanotechnology Design

Spin Wave Device

feature size 24 nm

DG-SiNWFET 22 nm

Divisor 32-bit IP

Behavioral

```
module div32 (a, b, quotient_uns, quotient_tc, remainder_uns,
remainder_tc);
parameter width = 32;
input [width-1
output [width-1
output signed [w
// operators for
assign quotient_
assign quotient_
assign remainder_uns = a % b;
assign remainder_tc = $signed(a) % $signed(b);
endmodule
```

Area: 4863 um²

Delay: 5.32 ns

GC: 35k

MIG

```
assign w0 = (w39266 & w27196) | (w39266 & w42866) | (w27196 & w42866);
assign w1 = w44100 & w4464;
assign w2 = ~w19966 & w27946;
assign w3 = ~w22044 & ~w30809;
assign w4 = w42722 & ~b[26];
assign w5 = (~w30032 & w19016) | (~w30032 & w47192) | (w19016 & w47192);
assign w6 = ~w26074 & w5179;
assign w7 = (w42822 & ~w46131) | (w42822 & w16107) | (~w46131 & w16107);
assign w8 = ~w3517
assign w9 = ~w4555
assign w10 = ~w144
assign w11 = (~w16
assign w12 = (w271
assign w13 = (w283
assign w14 = ~w252
assign w15 = (w437
assign w16 = ~w443
assign w17 = ~w229
assign w18 = ~w303
assign w19 = (~w23
assign w20 = w2680
assign w21 = (~w43
assign w22 = ~w36188 & w11587;
assign w23 = ~w33838 & ~w16188;
assign w24 = ~w38787 & w19348;
assign w25 = w42096 & w20651;
assign w26 = (w45543 & w39924) | (w45543 & w25446) | (w39924 & w25446);
assign w27 = ~w34177 & w38267;
assign w28 = ~w10074 & ~w29118;
assign w29 = w25495 & w32055;
assign w30 = (w48190 & w13546) | (w48190 & w19290) | (w13546 & w19290);
```

Area: 6115 um²

Delay: 3.55 ns

GC: 43k

Benc							
bi							
my							
c							
c							
a							
c							
mi							
s3							
mise							
Average	212/176	90.02	9.07	0.53	89.60	11.02	0.53

MIGs Summary

- Majority-Inverter Graphs with their Boolean algebra push further the capabilities of contemporary logic synthesis
- Improvements at the design level for general benchmarks but also for highly-optimized units (div32)
- Promising results for CMOS (22nm and 90nm nodes) and even better results for DG-SiNWFET nanotechnology with enhanced device functionality
- MIGs unveil efficient design opportunities unseen by state-of-art synthesis techniques

Outline

- Introduction
- Technological innovations and motivation
  - Emerging nanotechnologies and devices
- Design with emerging technologies
  - Physical and logic synthesis
- The majority paradigm in logic synthesis
  - Models, algorithms and tools
- **Conclusions**

Conclusions

- Emerging nano-technologies with enhanced-functionality devices increase computational density
- New design, synthesis and verification methods stem from new abstractions of logic devices
- Current logic synthesis is based on specific heuristics: new models with stronger properties lead us to better methods and tools for both CMOS and emerging devices

Never stop exploring!

Thank you