Why Again Logic Synthesis

Giovanni De Micheli

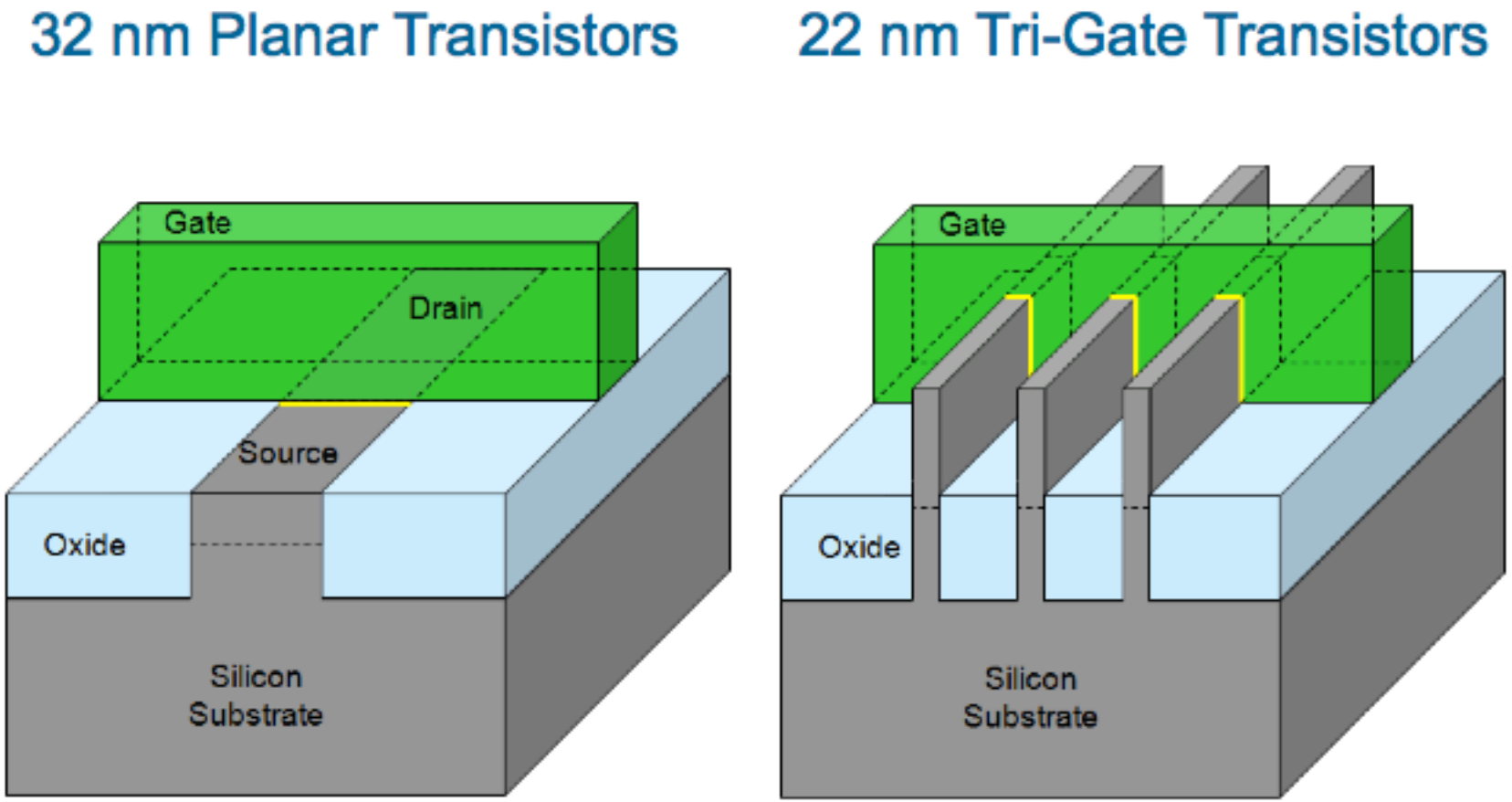
Why again logic synthesis?

- Strong *intellectual value* associated with logic synthesis and optimization
  - Problems are far from being solved
- Current methods and tools grew out of *control* and *random logic* design for CMOS semicustom libraries
  - Still inefficient for computational engines with predominance of arithmetic units
- Emerging *nanotechnologies*
  - New devices are game changers

The emerging nano-technologies

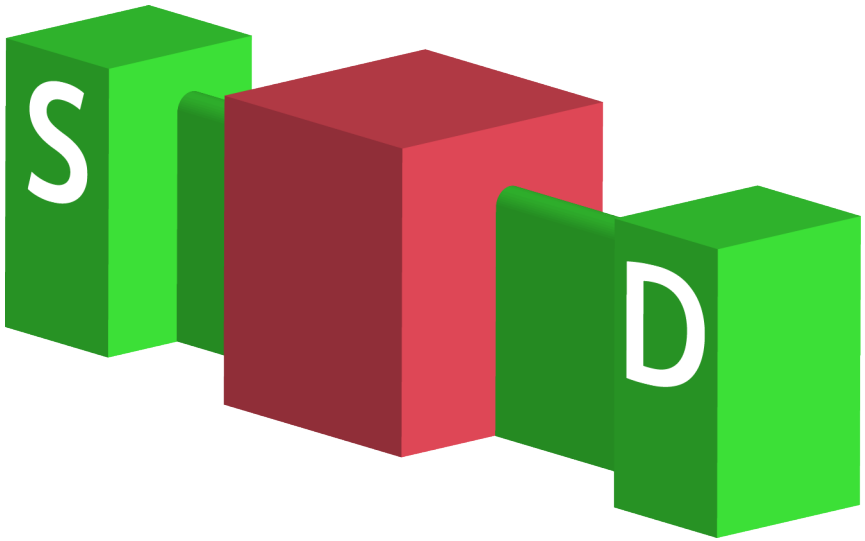
- *Enhanced* silicon CMOS is likely to remain the main manufacturing process in the medium term
  - The 10nm and 7nm technology nodes are on the way
- What are the candidate technologies for the 5nm node and beyond?
  - Silicon Nanowires (SiNW)
  - Tunneling FETs (TFET)
  - Carbon Nanotubes (CNT)
  -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

22 nm Tri-Gate transistors



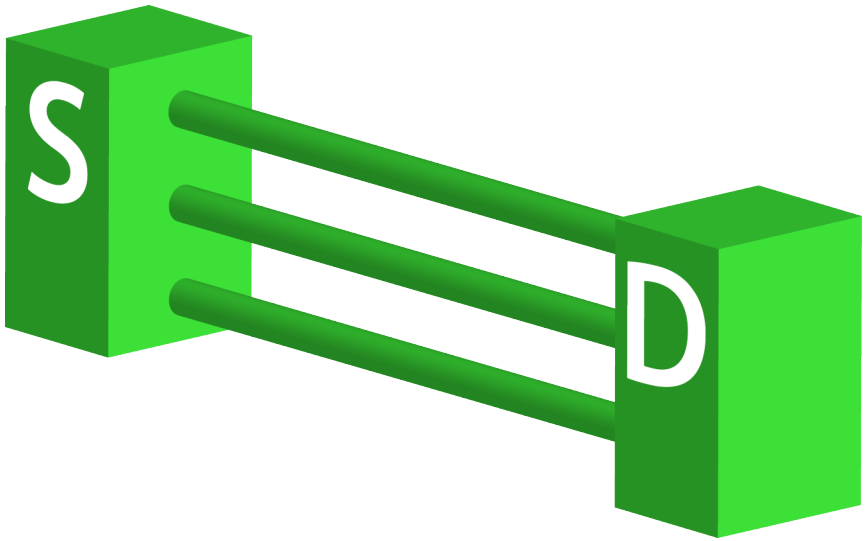
[Courtesy: M. Bohr]

From FinFET to Nanowire FET



FinFET

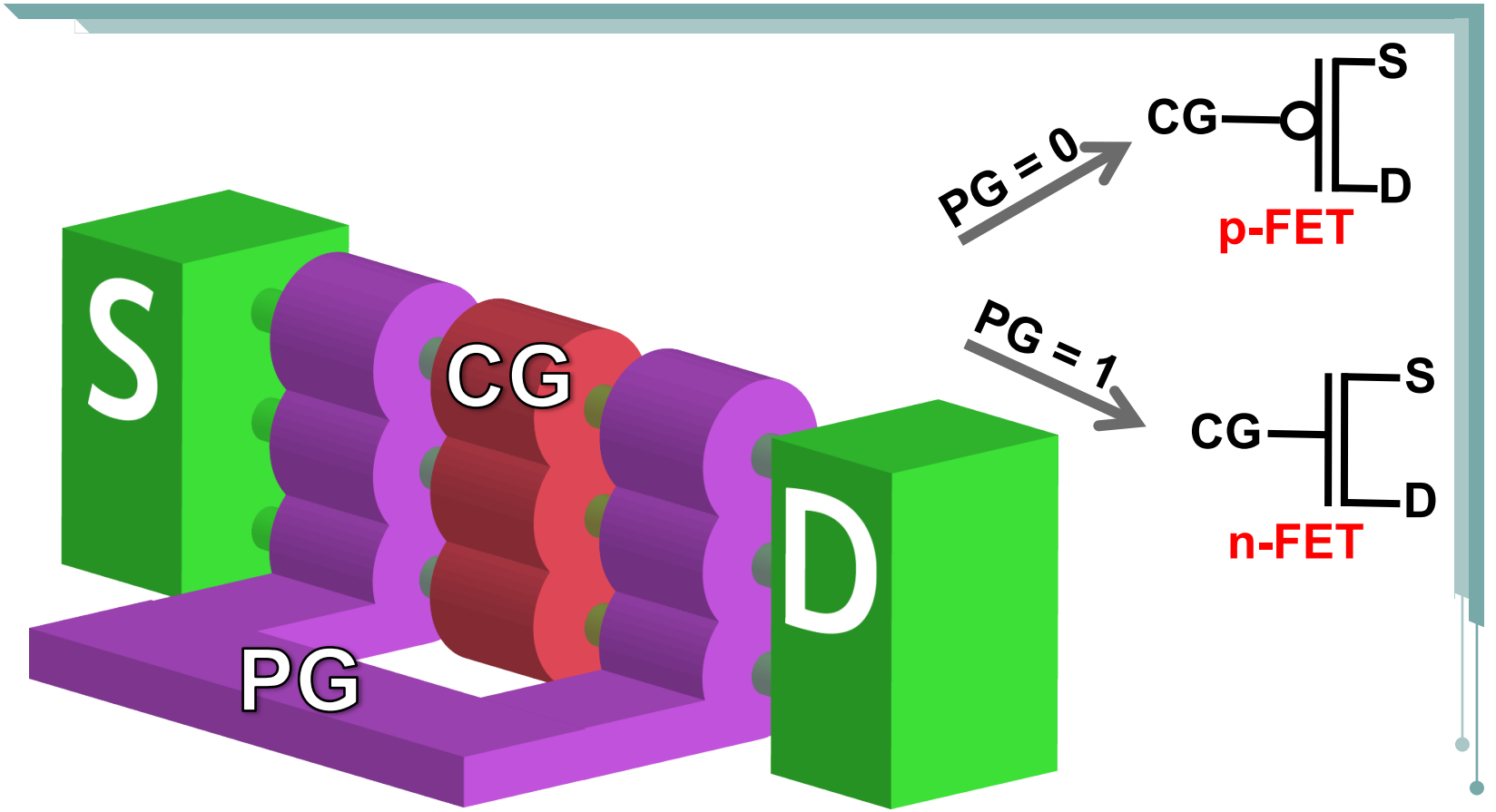
Three-sided gate



NanoWire FET

Gate All Around

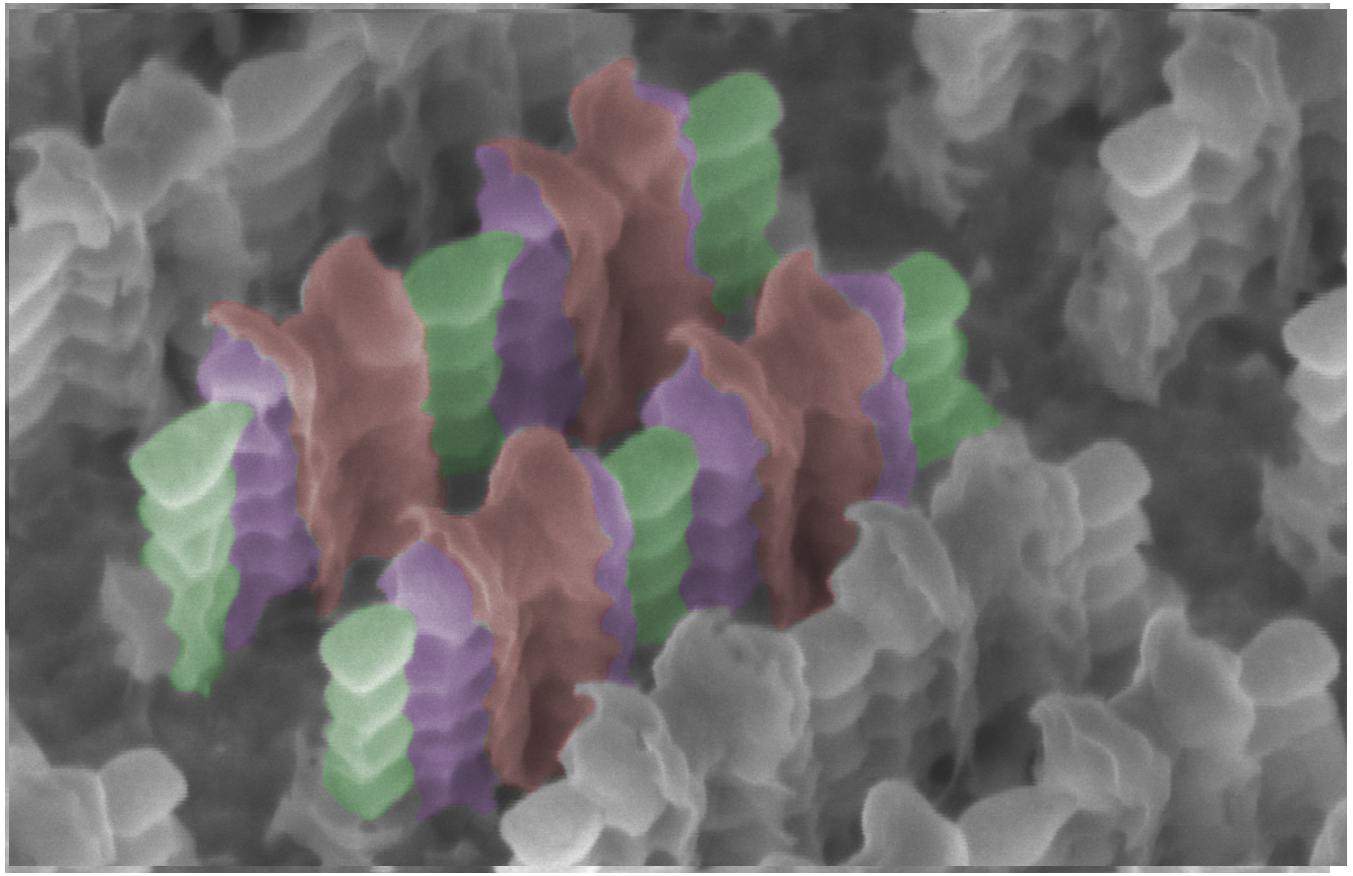
Electrostatic doping



- **Electrically program** the transistor to either **p-type** or **n-type**
- Field-effect control of the Schottky barrier

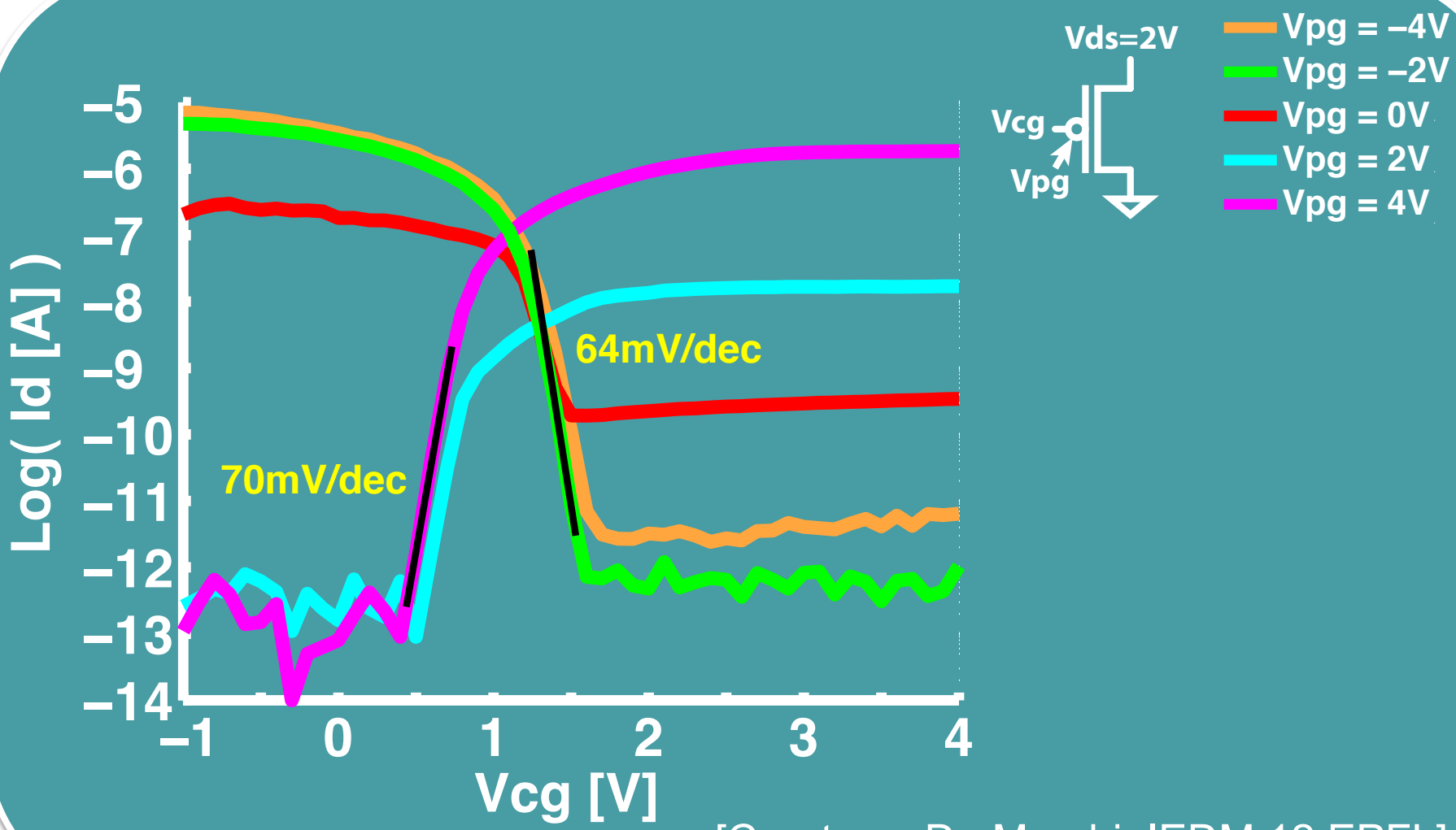
Silicon Nanowire Transistors

- Gate all around transistors
- Double gate to control polarity



[Courtesy: De Marchi, EPFL]

Device I_d/V_{cg}



[Courtesy: De Marchi, IEDM 12 EPFL]

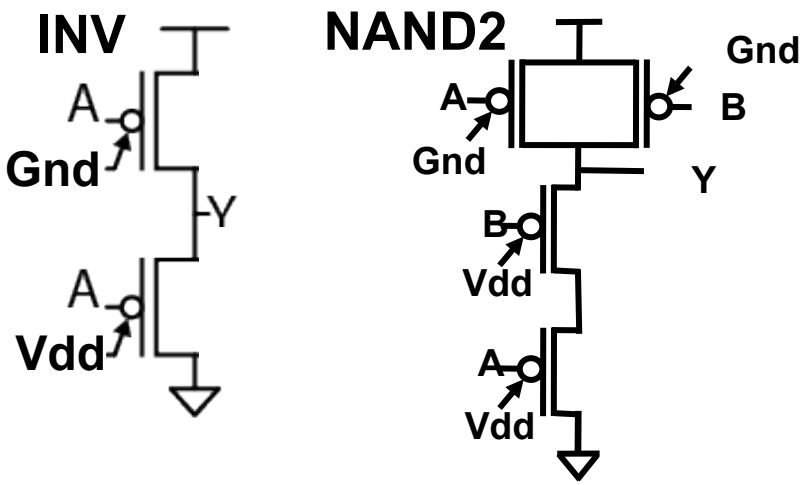
Logic level abstraction

- 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

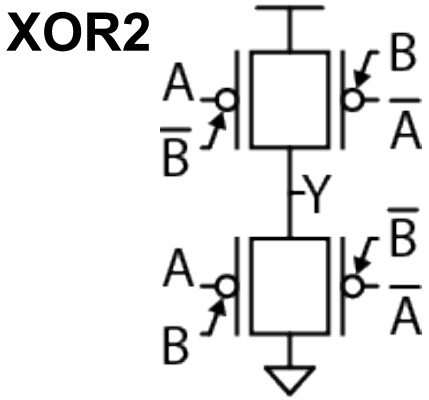
Logic cell design

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

Negative Unate functions	Binate functions

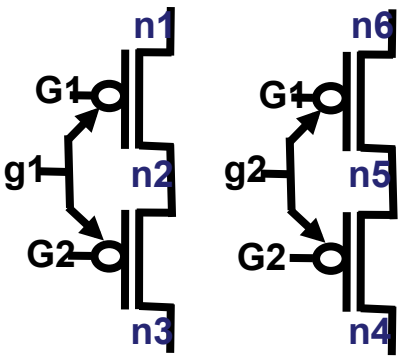


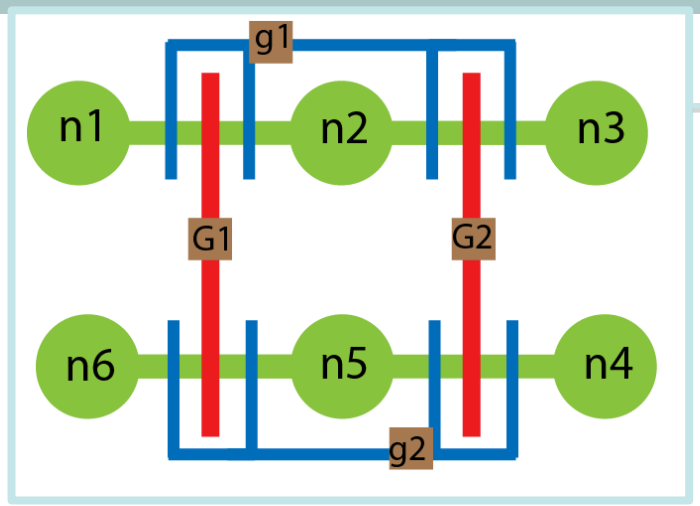
Similar to regular CMOS



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

Modular physical cell design

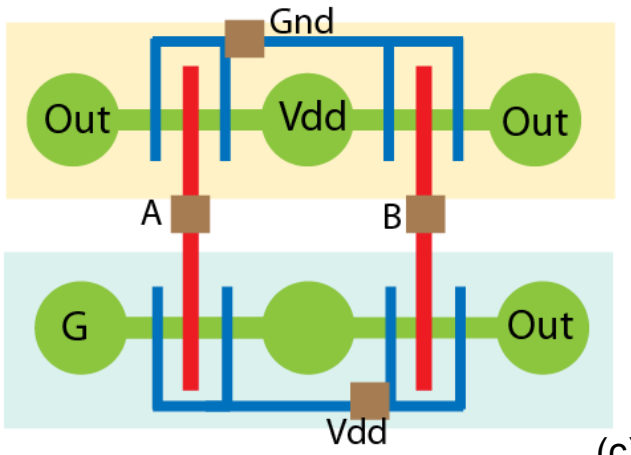




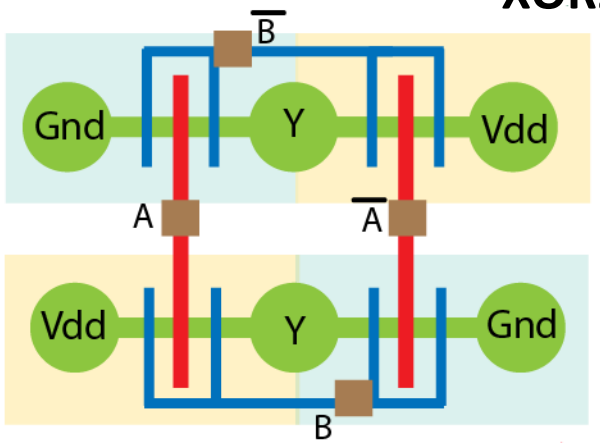
Tile

Two transistor pairs grouped together

NAND2

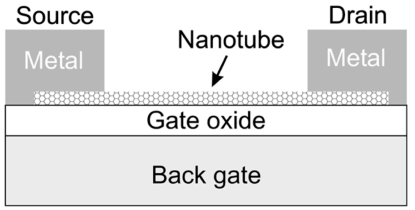


XOR2

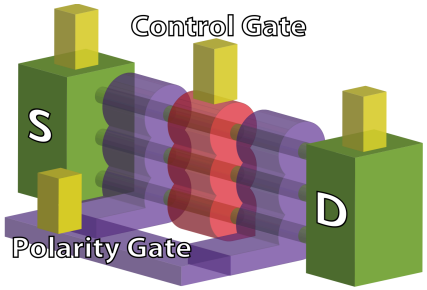


[Courtesy: Bobba, DAC 12]

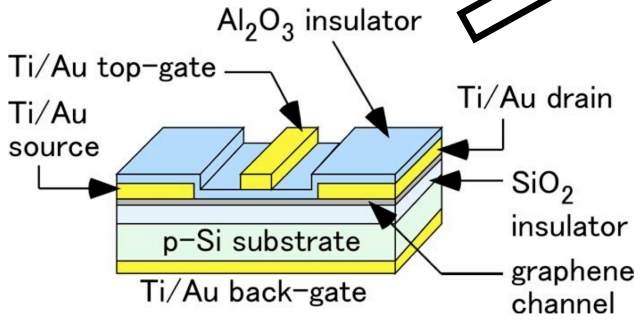
Modeling various emerging nanogates



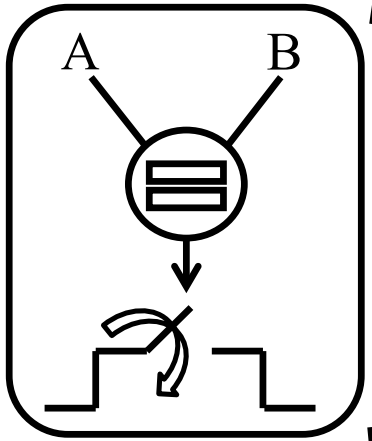
CNFETs

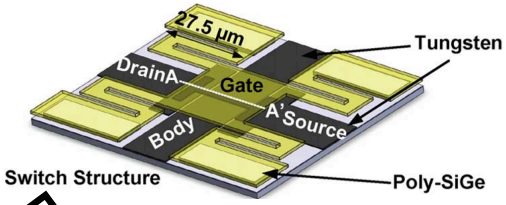


SiNWFETs

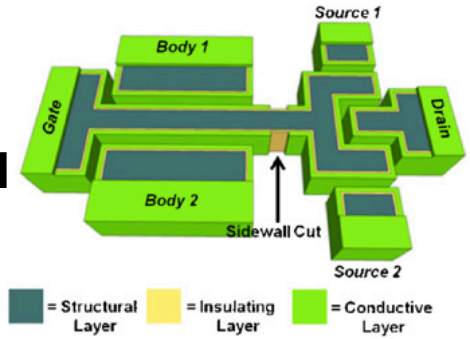


Graphene FETs

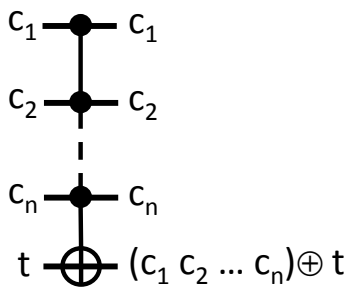




4T Nanorelays



6T Nanorelays

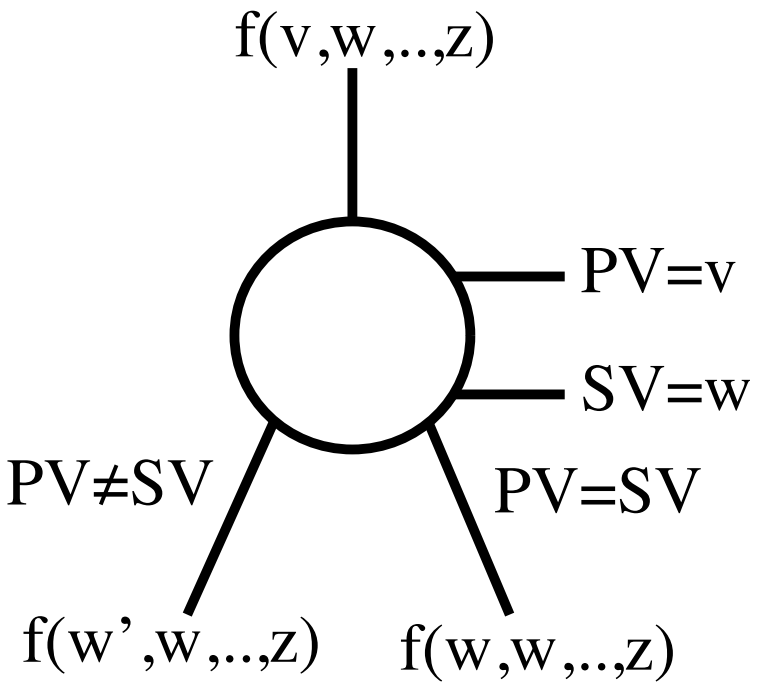


Reversible Logic

Biconditional Binary Decision Diagrams

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

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

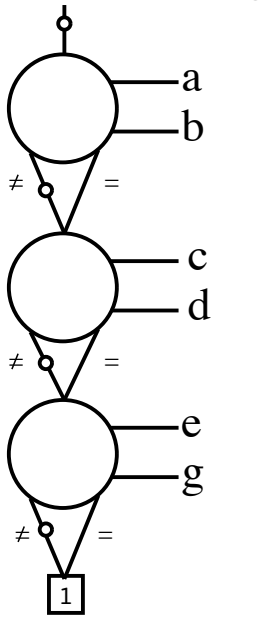


- Each BBDD node:
  - Has two branching variables
  - Implements the *biconditional* expansion
  - Reduces to Shannon's expansion for single-input functions

[Courtesy: Amaru', JETCAS 14]

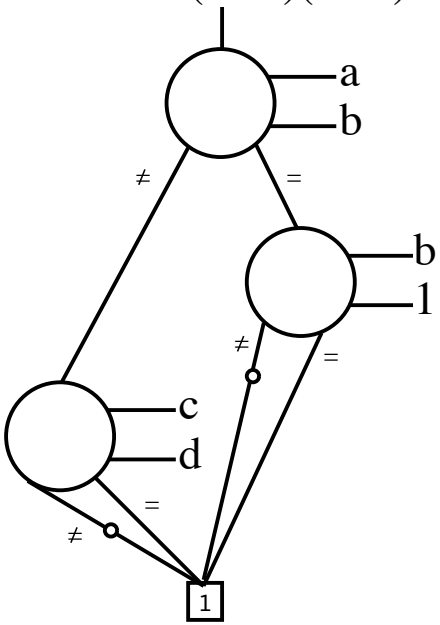
BBDD: Examples

a) f=a⊕b⊕c⊕d⊕e⊕g



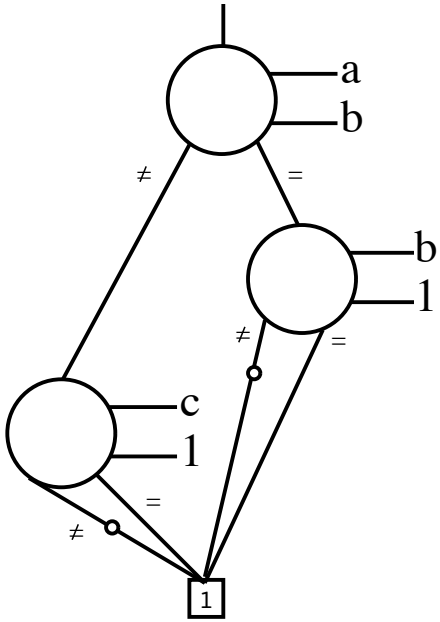
π=(a,b,c,d,e,g)

b) f=ab+(a⊕b)(c⊕̄d)



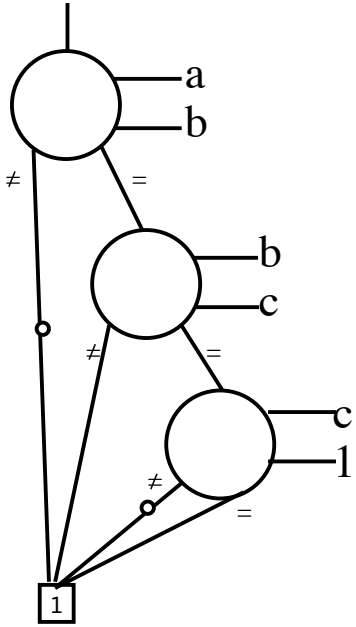
π=(a,b,c,d)

c) f=ab+bc+ac



π=(a,b,c)

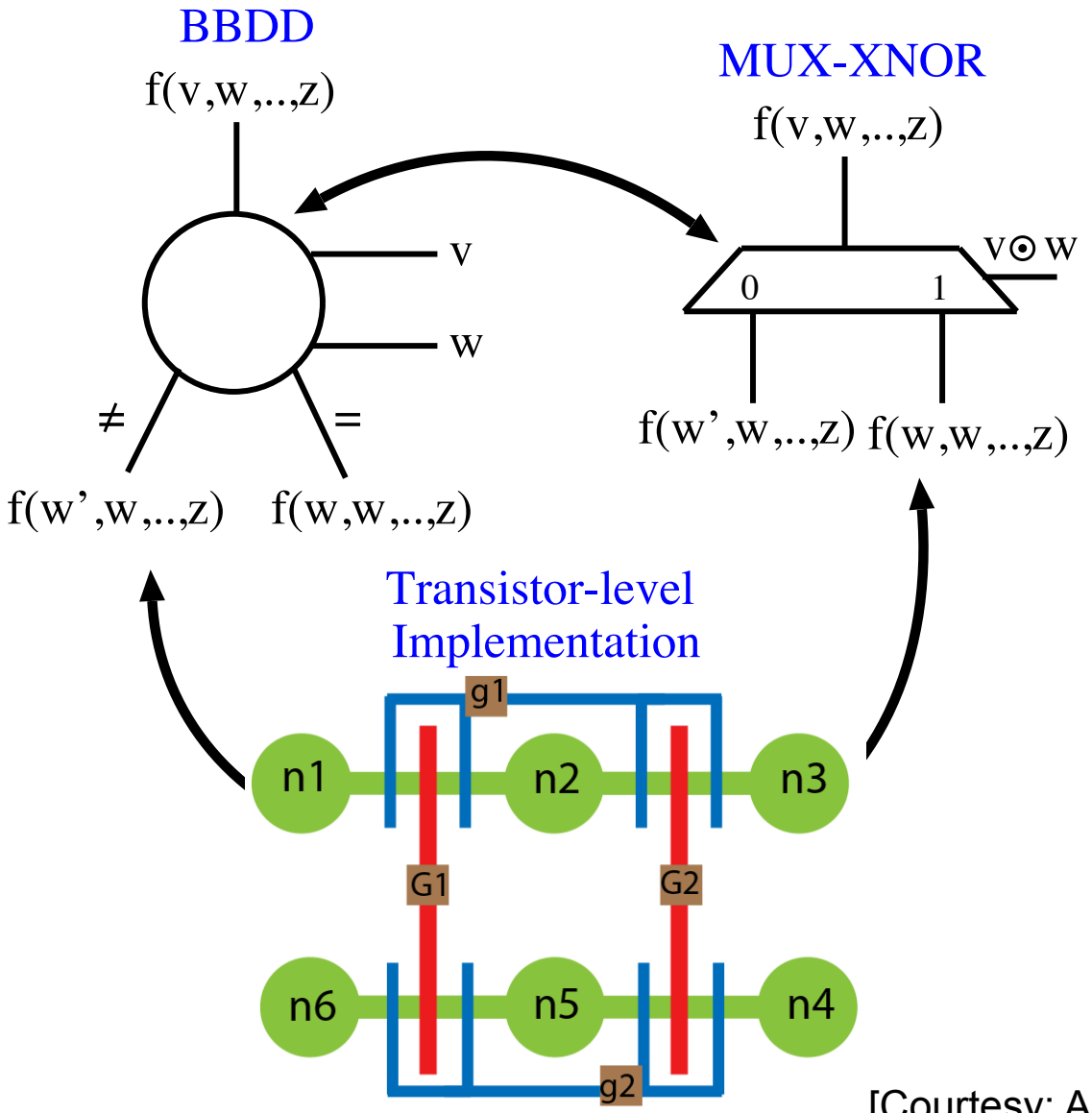
d) f=(a⊕̄b)(b+c)



π=(a,b,c)

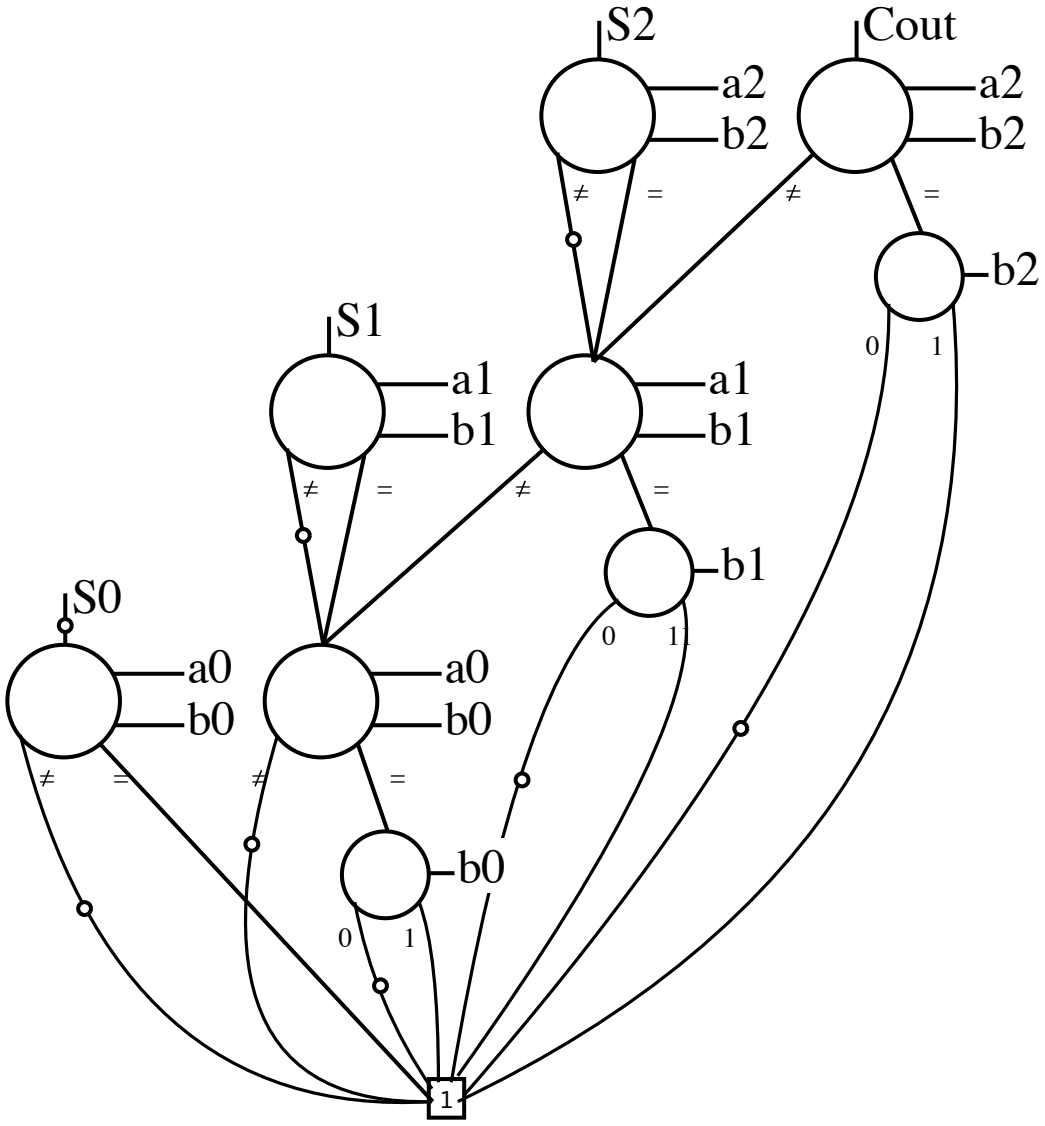
- The BDD counterparts for these examples have about 50% more nodes!

Efficient direct mapping of BBDD nodes



[Courtesy: Amaru', DATE 13]

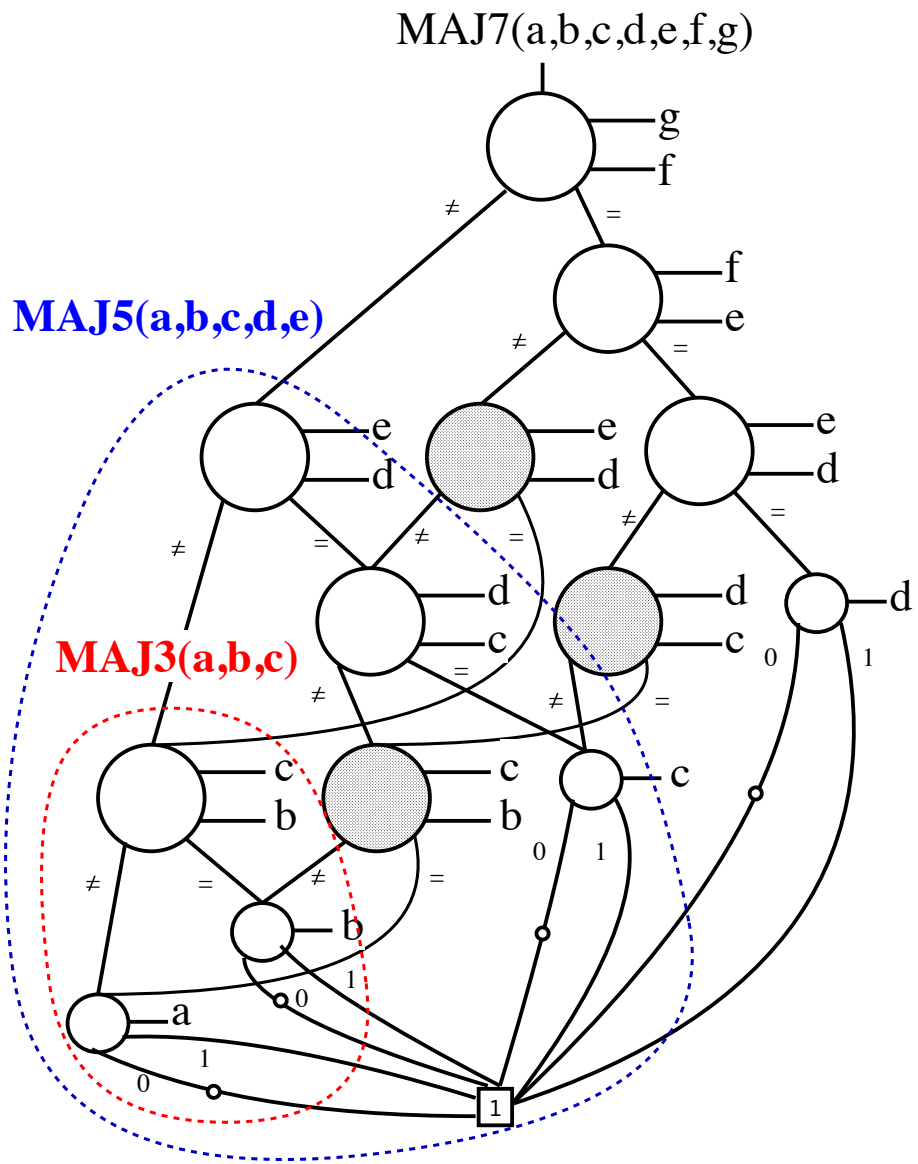
Compact BBDD representations



3-bit adder

- n-bit adder size:
  - 3n+1 nodes
- BDD counterpart:
  - 5n+2 nodes

Compact BBDD representations



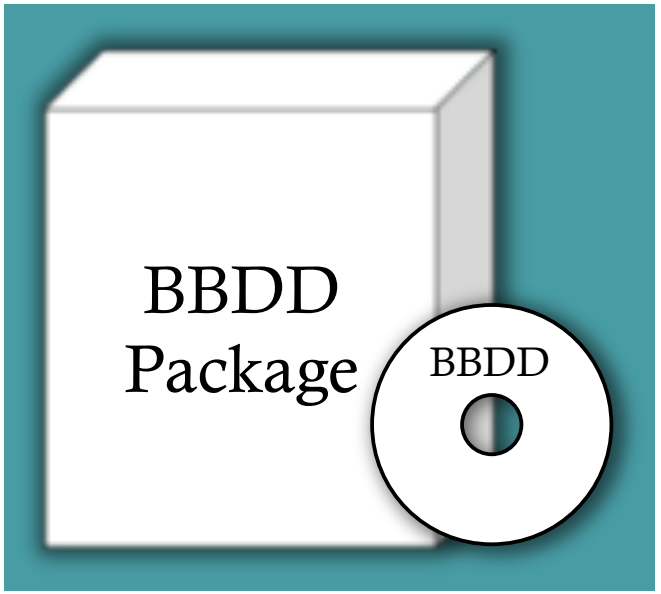
7-bit majority

- n-bit majority size:
  - 0.25 (n^2 + 7) nodes
- BDD counterpart:
  - $\lceil 0.5n \rceil (n - \lceil 0.5n \rceil + 1) + 1$ nodes

The BBDD optimization tool

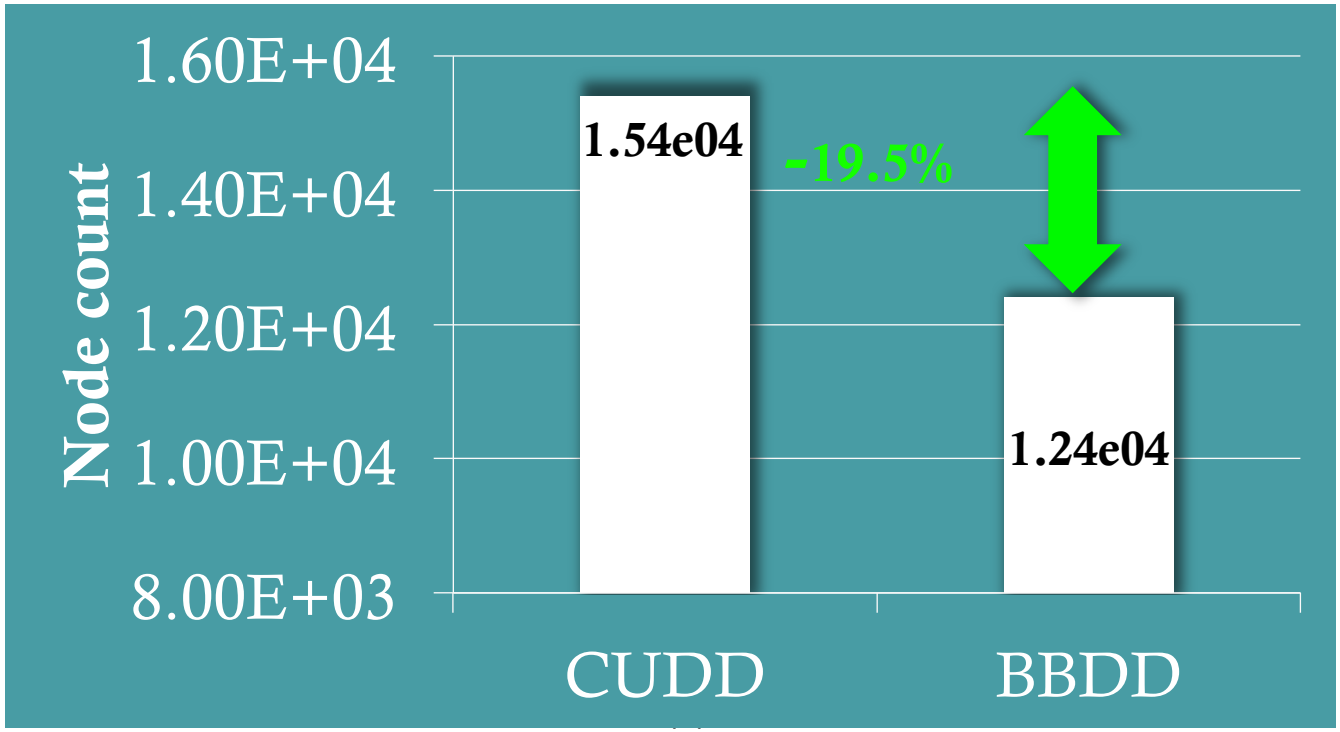
- Recursive formulation of Boolean operations
- Unique table to store BBDD nodes
- Performance-oriented memory management
- Chain variable reordering

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 diagrams and then apply sifting



Also 1.63x speedup for arithmetic intensive circuits

Case study: arithmetic restructuring

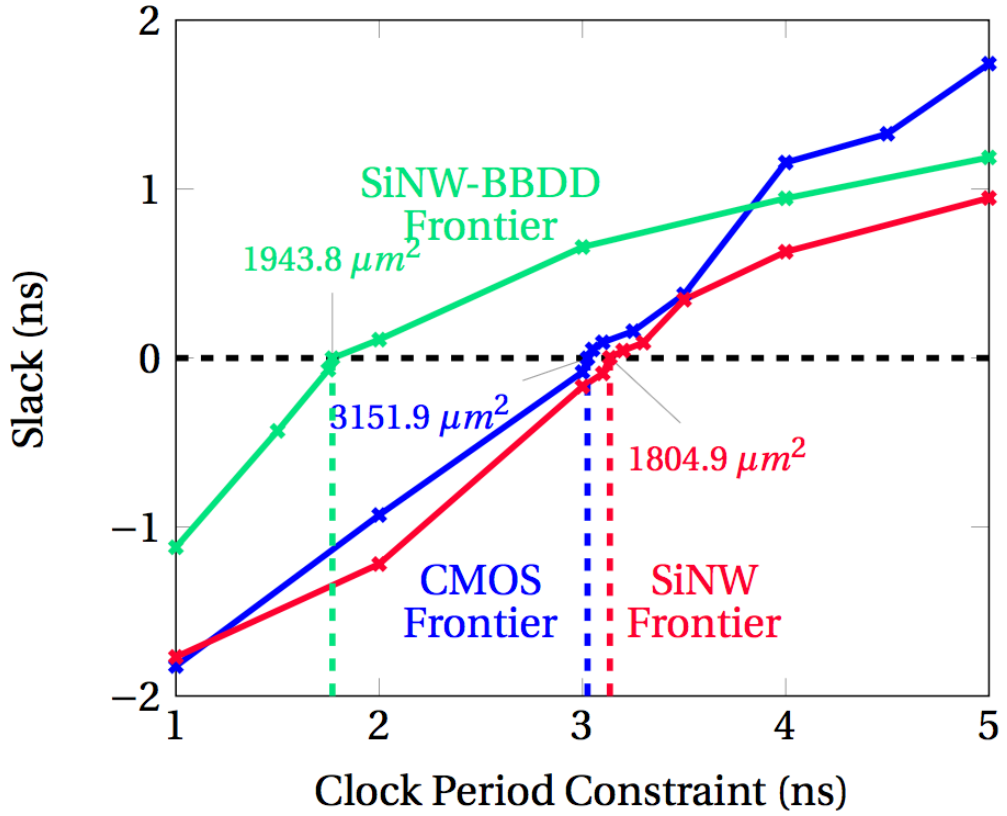
- Use BBDD to restructure arithmetic circuits prior to synthesis
- Front-end to a commercial synthesis tool
- Real-life telecommunication design: *Iterative Product Code Decoder*

Optimization via BBDD-rewriting								
Logic Circuits	Type	I/O		BBDD-rewriting		Original		Gain
		Inputs	Outputs	Nodes	Levels	Nodes	Levels	
adder08_bit.vhd	Comb.	16	9	16	8	78	19	✓
bit_comparator.vhd	Comb.	5	3	3	1	8	3	✓
comparator_7bits.vhd	Comb.	14	3	21	7	58	14	✓
fulladder.vhd	Comb.	3	2	2	1	9	4	✓
ext_val.vhd	Comb.	16	8	674	16	173	29	✗
twos_c_8bit.vhd	Comb.	8	8	20	8	29	8	✓
ser2par8bit.vhd	Seq.	11	64	-	-	-	-	-
product_code.vhd	Top	10	4	-	-	-	-	-

Synthesis in 22-nm CMOS Technology – Clock Period Constraint: 0.6 ns (1.66 GHz)								
				BBDD + Synthesis Tool		Synthesis Tool		
		Inputs	Outputs	Area (μm^2)	Slack (ns)	Area (μm^2)	Slack (ns)	Constraint met
product_code.vhd	Top	10	4	1291.03	0.00	1177.26	-0.12	✓

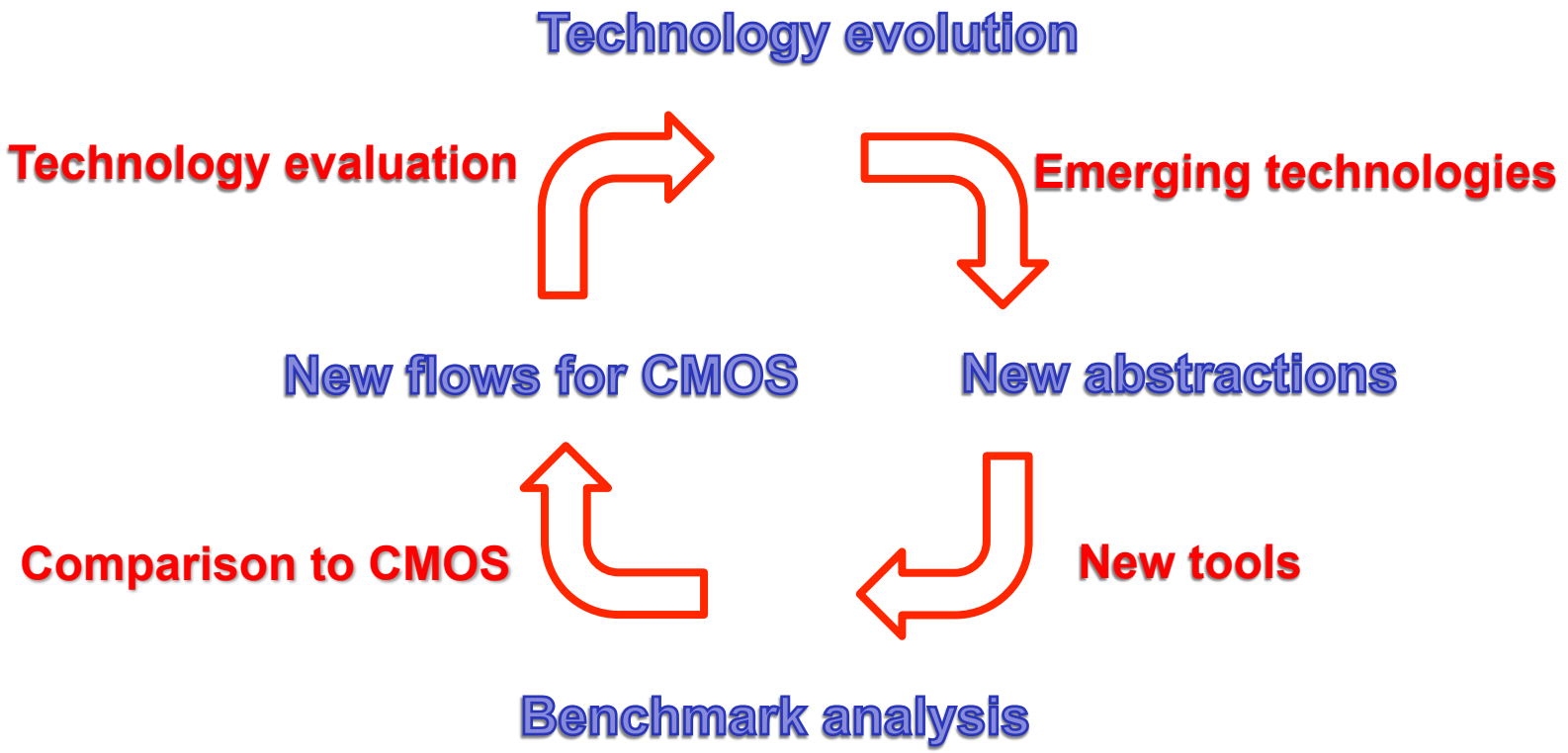
Nanotechnology design

- *Iterative Product Code Decoder*
- Analysis after Physical Design:
  - 22 nm FINFET
  - 22-nm DG-SiNWFE



[Courtesy: Amaru', IEEE Proceedings 15]

Logic synthesis for design and assessment



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

Thank you