Non-volatile Memory Friendly FPGA Synthesis

Chengmo Yang

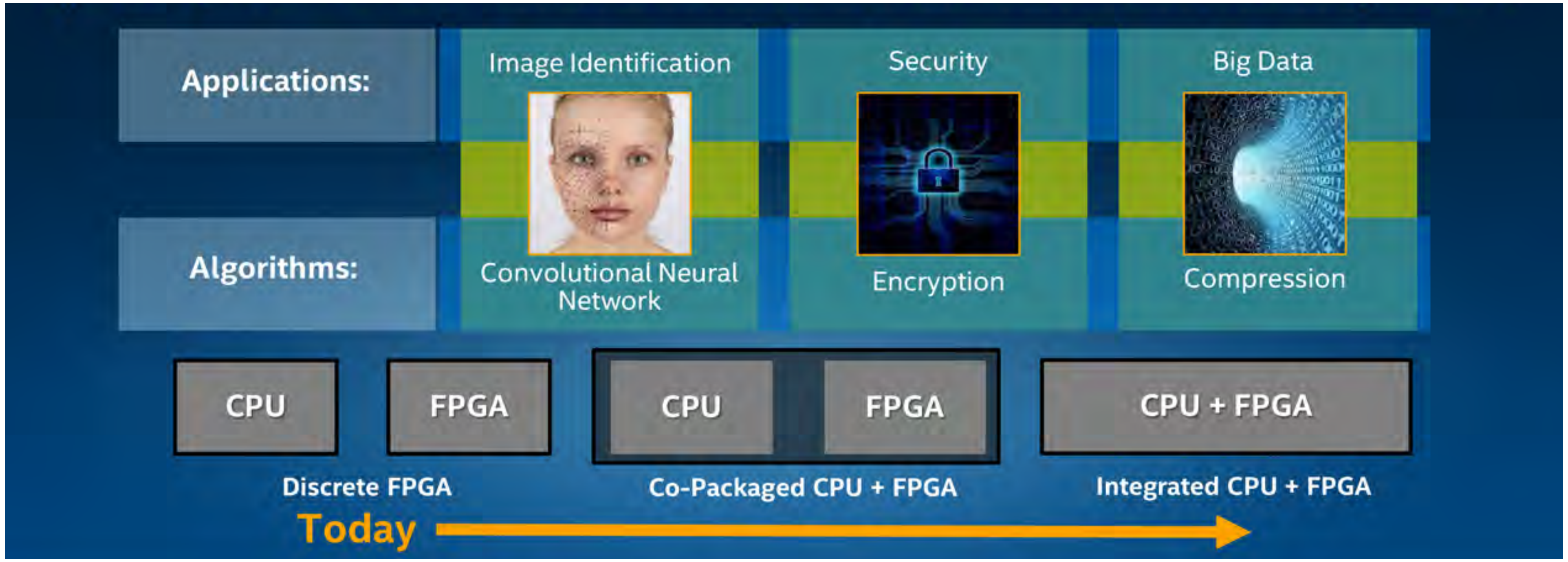
Electrical and Computer Engineering

University of Delaware

Outline

- Introduction
- Rethink about FPGA synthesis
  1. Logic synthesis
  2. Logic/memory co-placement
  3. Logic placement
  4. Routing
- Conclusions

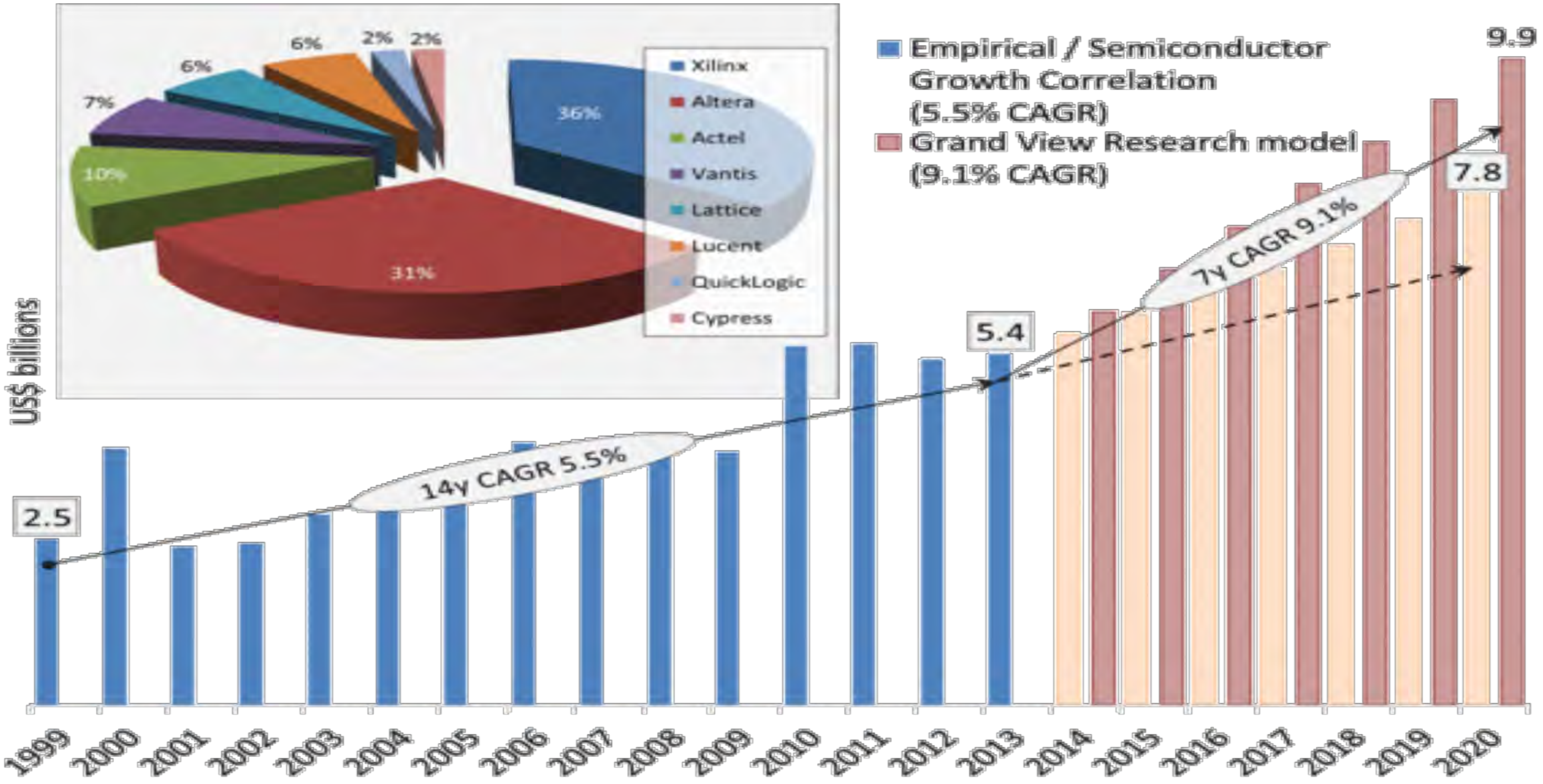
Big data and artificial intelligence



Intel represents a chance to further branch out in the markets.

FPGA could cooperate with microprocessors as accelerator for big data and artificial intelligence applications because of its flexibility and efficiency.

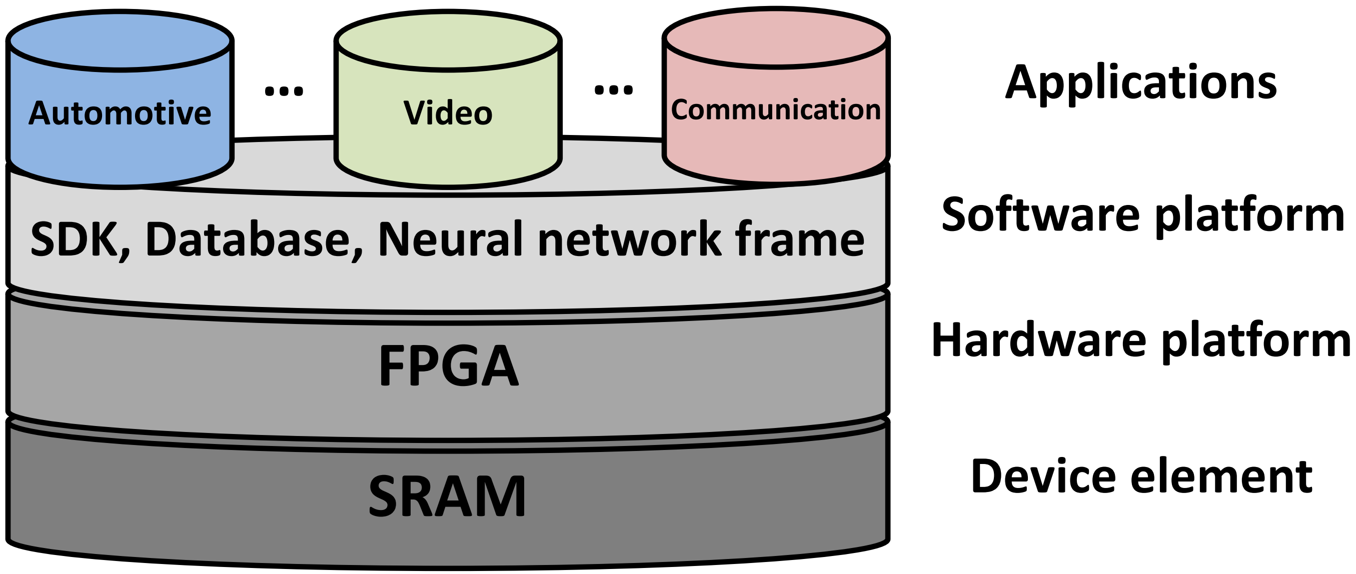
FPGA popularity increases



Sources: Grand View Research, EE Times, TABB Group

System stack

Applications and software grow dramatically.



Automotive … Video … Communication — Applications

SDK, Database, Neural network frame — Software platform

FPGA — Hardware platform

SRAM — Device element

FPGAs face a lot of challenges.

SRAM limitations

Low Scalability

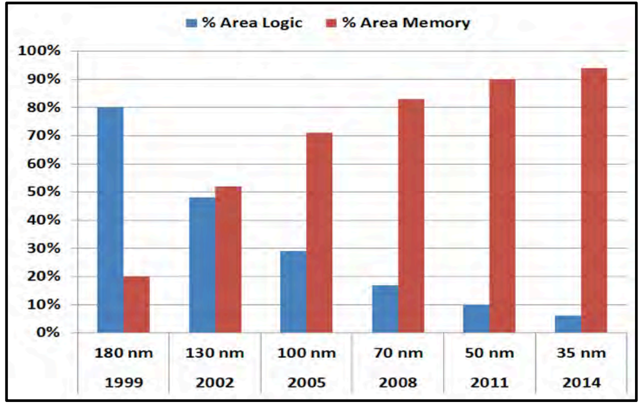
High Leakage Power

Prone to Soft Error

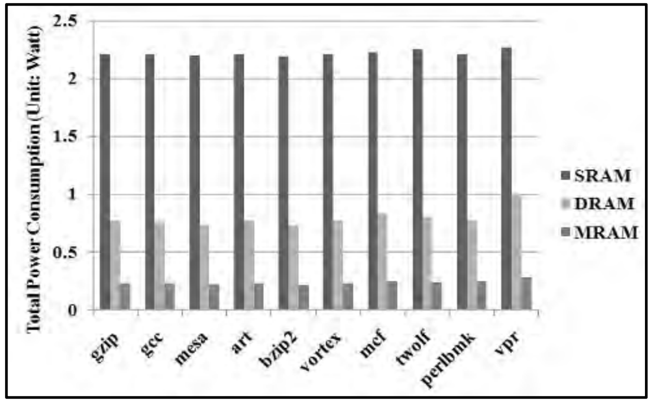
Volatile

Drawbacks of SRAM FPGAs

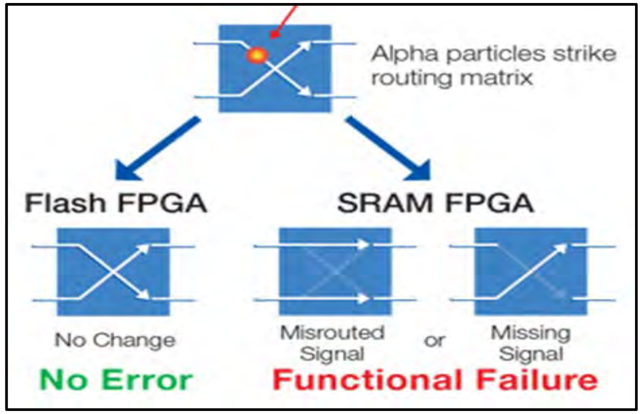
Low Scalability



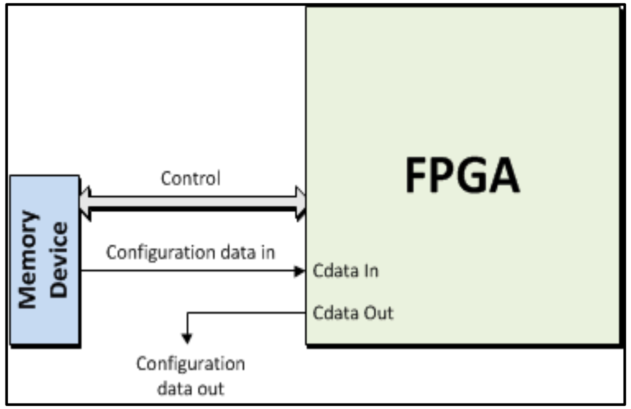
High Leakage Power



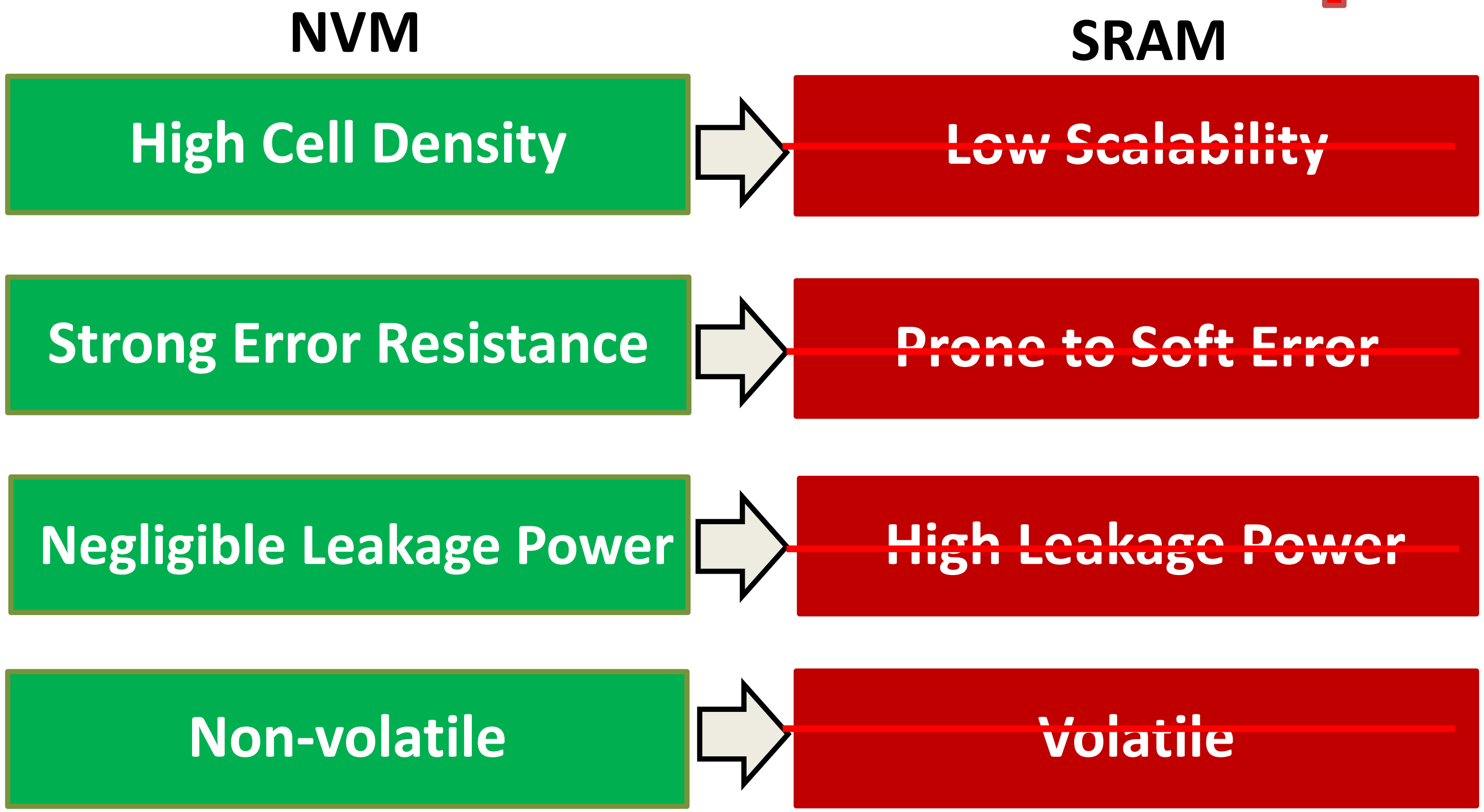
Prone to Soft Error



Volatile



How to tackle these limitations?



NVM	SRAM
High Cell Density	~~Low Scalability~~
Strong Error Resistance	~~Prone to Soft Error~~
Negligible Leakage Power	~~High Leakage Power~~
Non-volatile	~~Volatile~~

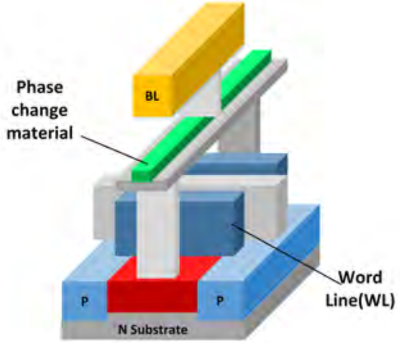
Non volatile Memories (NVMs) can replace SRAM in FPGAs.

Non volatile memories (NVM)

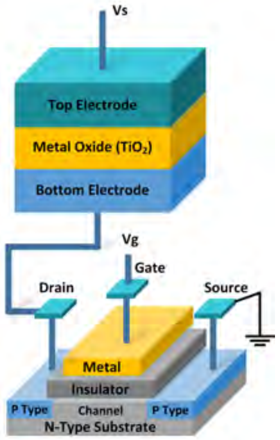
NVM use physical characteristics to represent logic states, so data is still kept when power is off.

- Phase Change Memory (**PCM**)
- Resistive RAM (**RRAM**)
- Spin Transfer Torque Magnetic RAM (**STT-MRAM**)

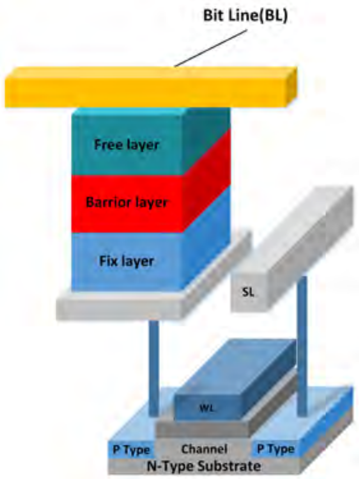
…



PCM



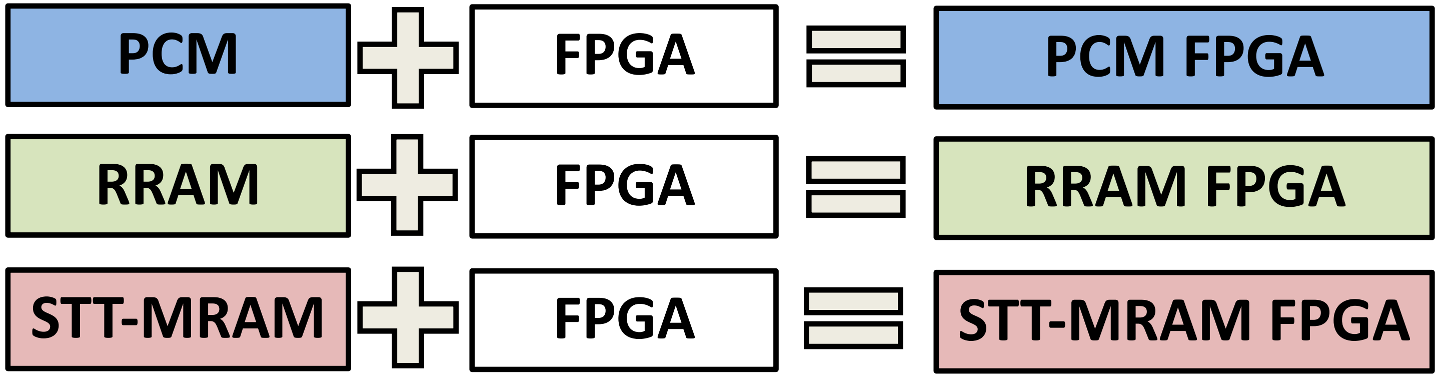
RRAM



STT-MRAM

Various NVM FPGAs

Both academia and industries have made intensive studies on NVM FPGAs.



PCM + FPGA = PCM FPGA

RRAM + FPGA = RRAM FPGA

STT-MRAM + FPGA = STT-MRAM FPGA

Related works

[1] Y. Chen, J. Zhao, and Y. Xie, “3D-NonFAR: Three-dimensional non-volatile FPGA architecture using phase change memory," in ISLPED, 2010, pp. 55-60.

[2] A. D. Chunan Wei and D. Chen, “A scalable and high-density FPGA architecture with multi-level phase change memory,” in Design, Automation and Test in Europe Conference and Exhibition (DATE), 2015, pp. 1365–1370.

[3] Y.-C. Chen, W. Wang, H. Li, and W. Zhang, “Non-volatile 3D stacking RRAM-based FPGA," in FPL, 2012, pp. 367-372.

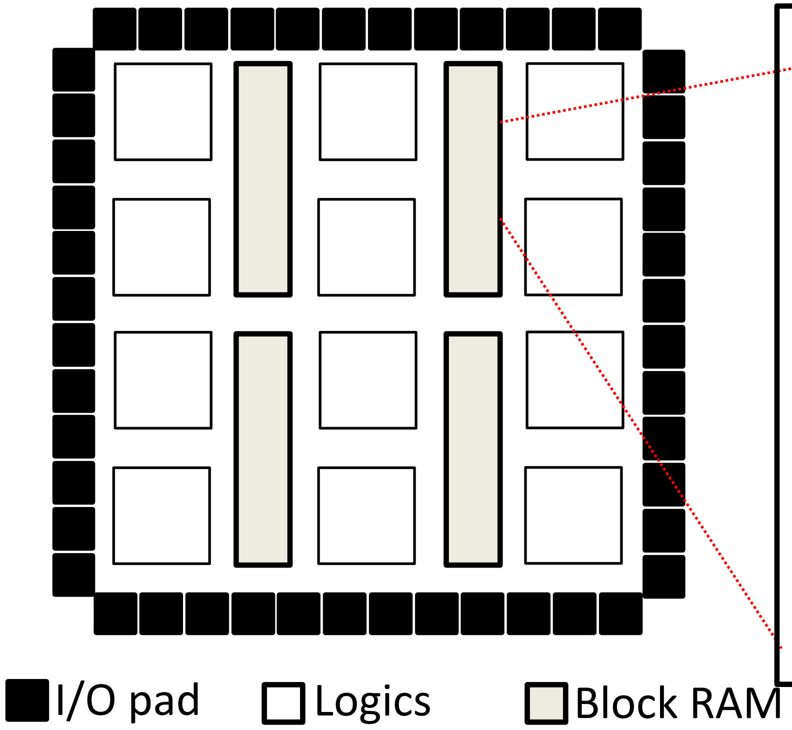
[4] P. Gaillardon, D. Sacchetto, G. Beneventi, M. Ben Jamaa, L. Perniola, F. Clermidy, I. O'Connor, and G. DeMicheli, “Design and architectural assessment of 3-D resistive memory technologies in FPGAs," TNANO, vol. 12, Jan. 2013.

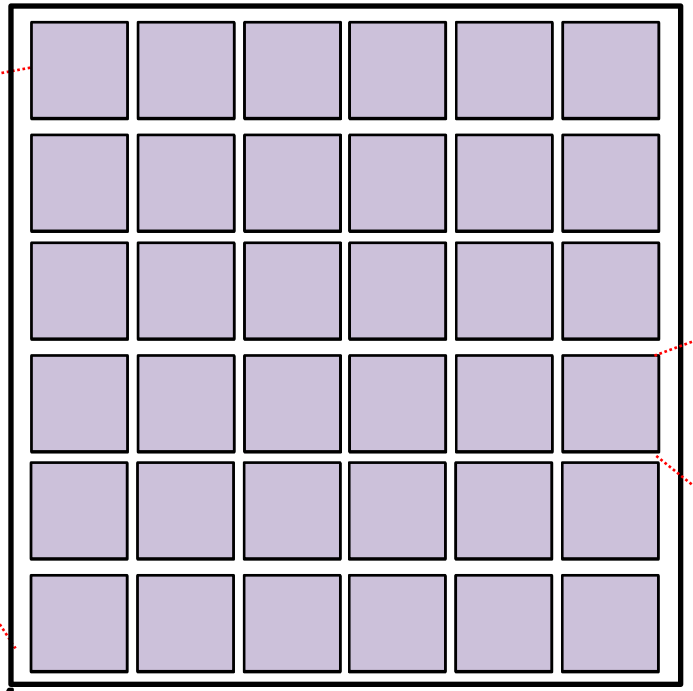
[5] W. Zhao, E. Belhaire, C. Chappert, and P. Mazoyer, “Spin transfer torque (STT)-MRAM{based runtime reconfiguration FPGA circuit," TECS, vol. 9, Oct. 2009.

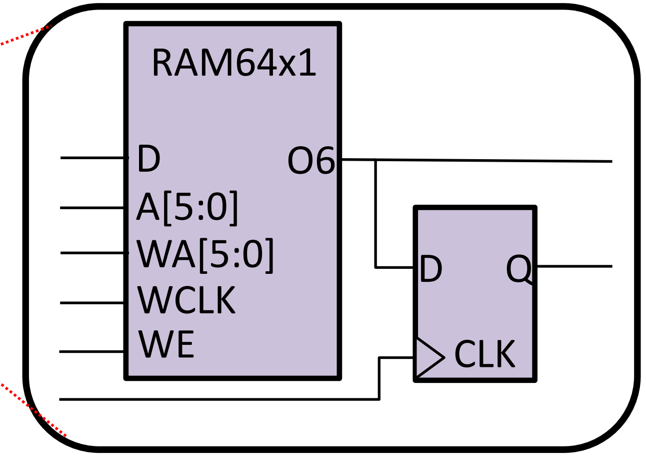
[6] W. Zhao, E. Belhaire, C. Chappert, B. Dieny, and G. Prenat, “TAS-MRAM Based Low-Power High-Speed Runtime Reconfiguration (RTR) FPGA,” ACM Transactions on Reconfigurable Technology and Systems (TRETS), vol. 2, Jun. 2009.

NVM FPGA memory structure

Both memory and logic can be implemented with uniform NVM block structure.







FPGA chip

NVM

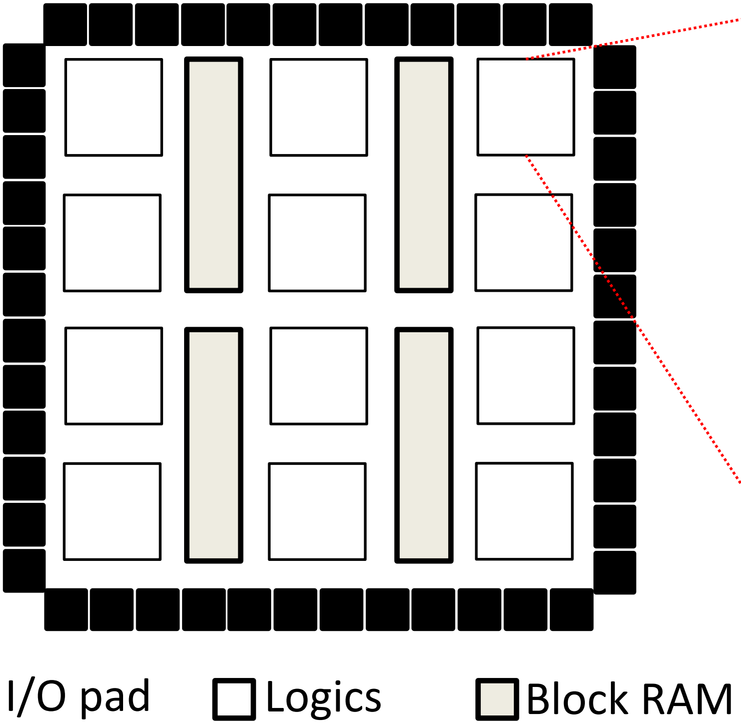
SRAM

Memory array

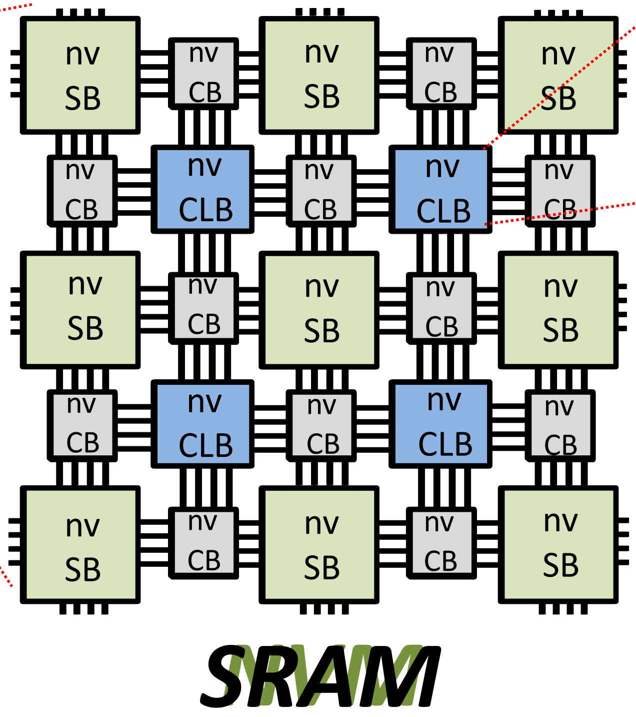
NVM block used as memory

NVM FPGA logic structure

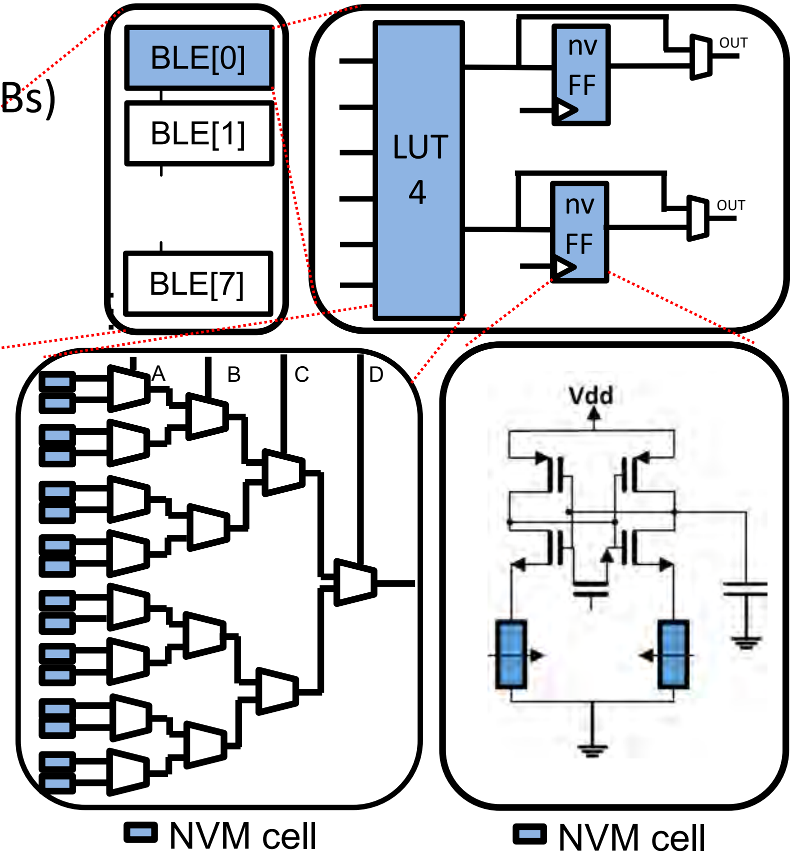
FPGA contains three types of blocks :
Configurable logic blocks (CLBs), Connection blocks(CBs) and Switch blocks (SBs)



FPGA chip



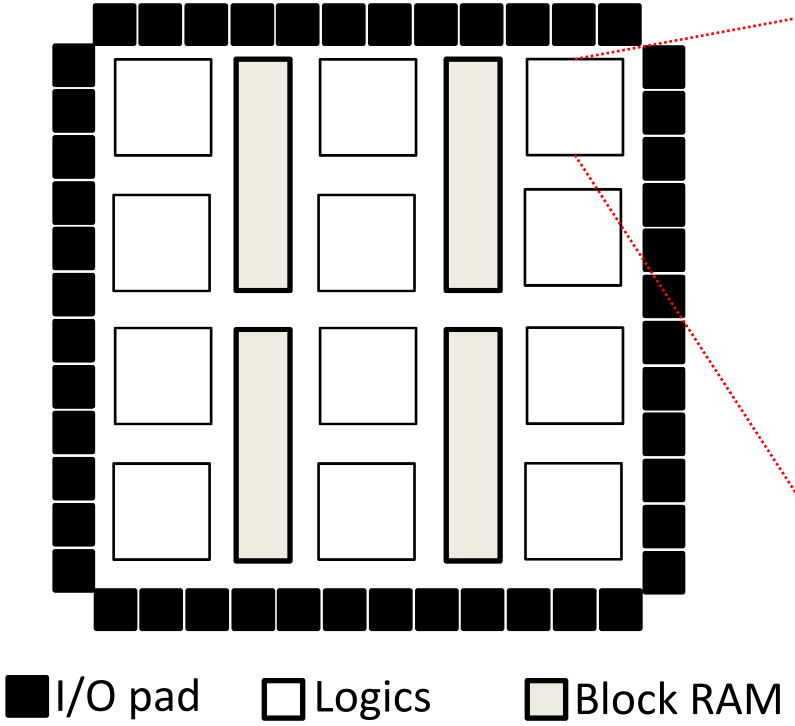
Logic structure



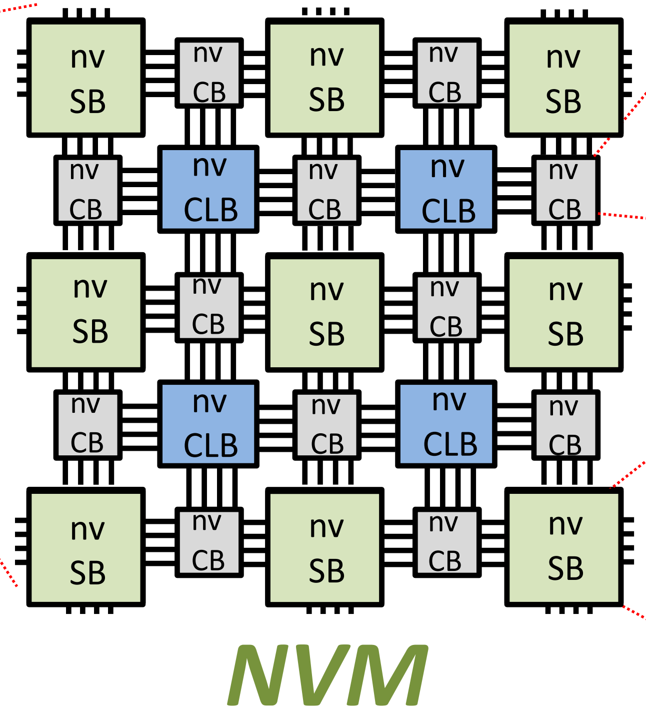
NVM block used as logic

NVM FPGA logic structure

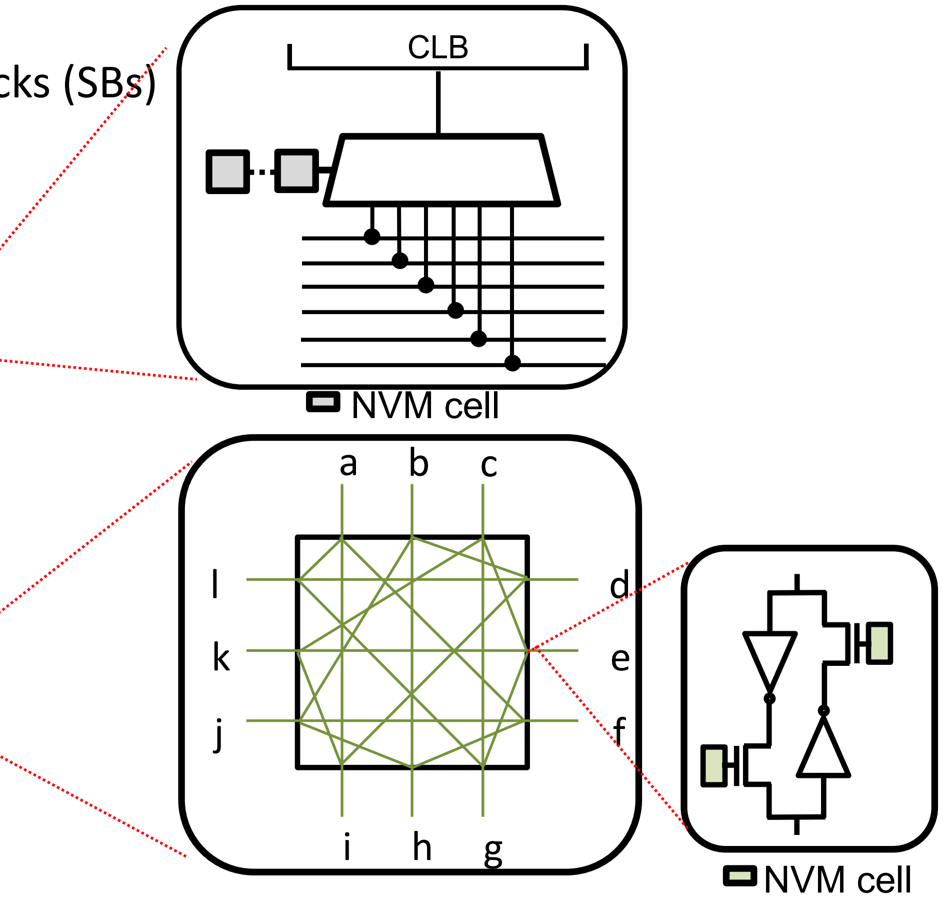
FPGA contains three types of blocks :
Configurable logic blocks (CLBs), Connection blocks(CBs) and Switch blocks (SBs)



FPGA chip



Logic structure



NVM block used as logic

NVM FPGA challenges

Type	Area (F^2)	Read time(ns)	Write time(ns)	Write energy(J)	Write cycles
SRAM	140	0.2	0.2	10^{-17}	10^{16}
PCM	4	12	100	10^{-15}	10^{9}
STT-MRAM	20	35	35	10^{-13}	10^{12}
RRAM	4	40	65	10^{-13}	10^{14}

NVM drawbacks bring two major challenges to FPGAs:

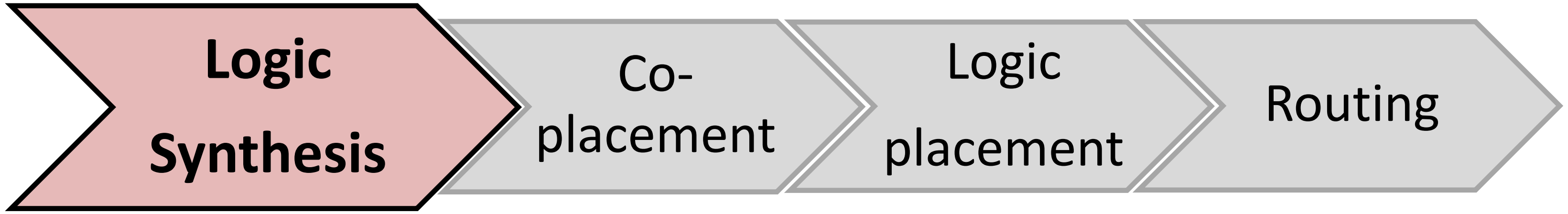
- **Writes** are slow and energy-consuming, harmful to performance and power!
- **Endurance** is short and limits device lifetime!

Where and how to address the challenges?

Where: Focus on NVM elements such as FFs, CLBs and SBs.

How: Tune synthesis flow to be NVM friendly.

FPGA synthesis introduction

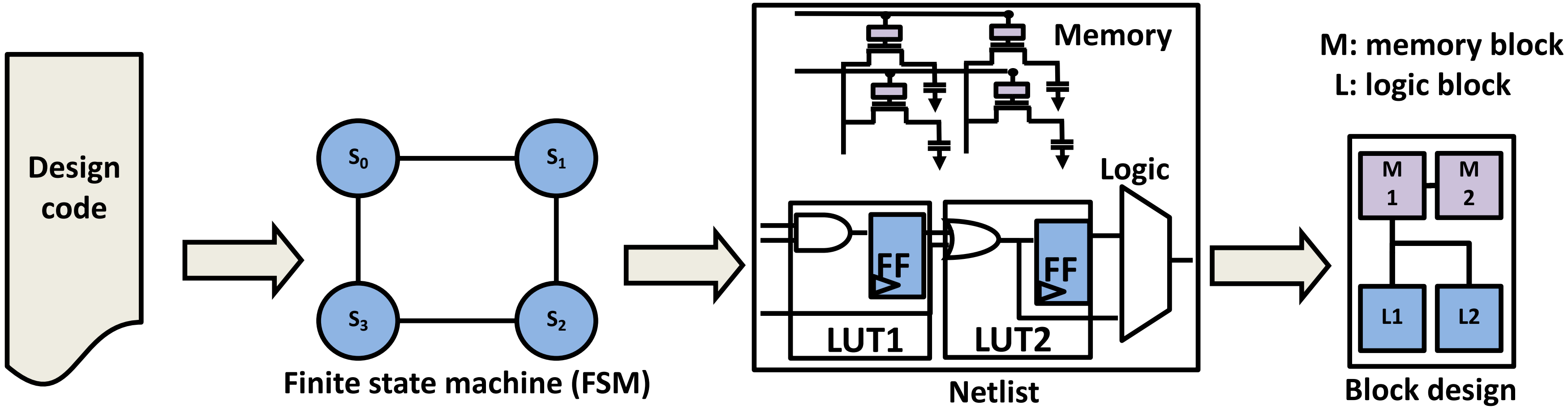


Logic Synthesis

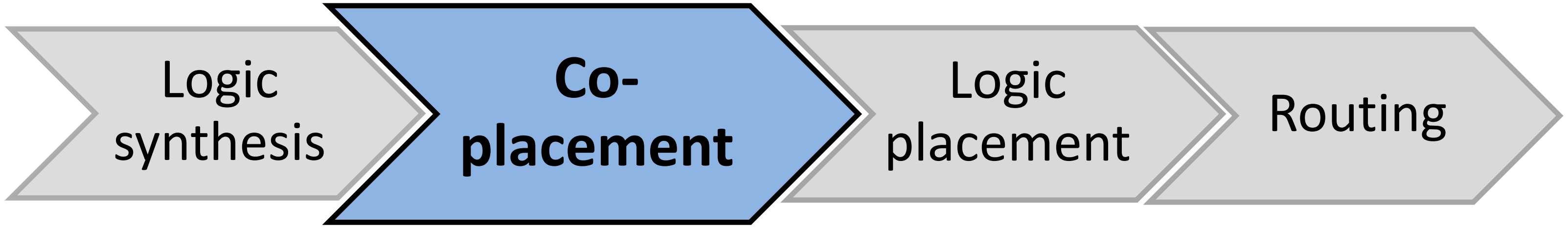
Co-placement

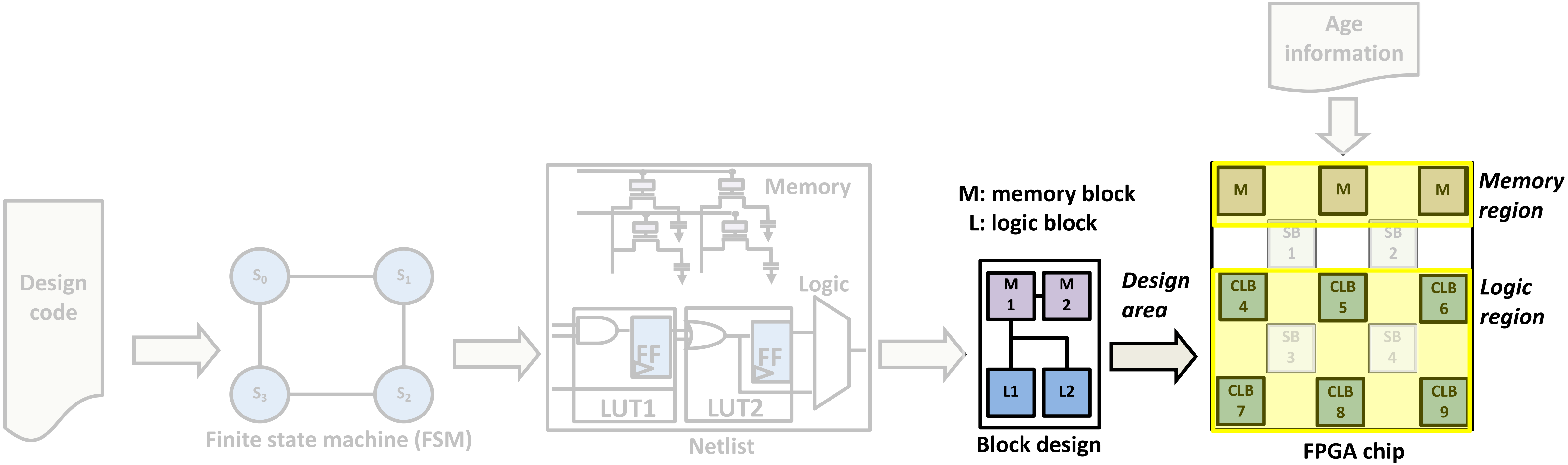
Logic placement

Routing

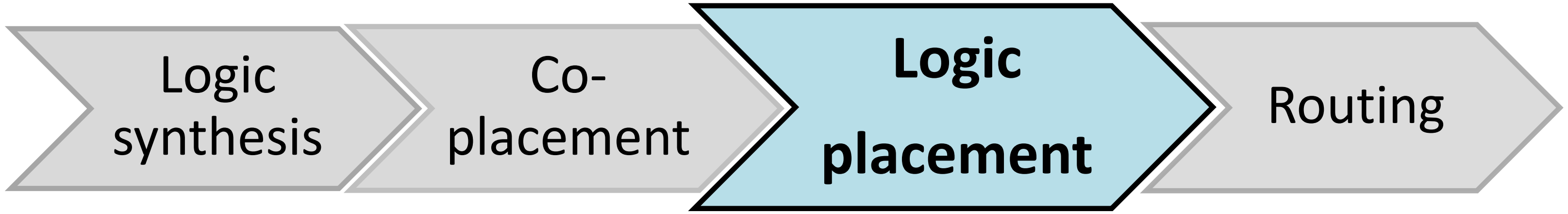


FPGA synthesis introduction





FPGA synthesis introduction

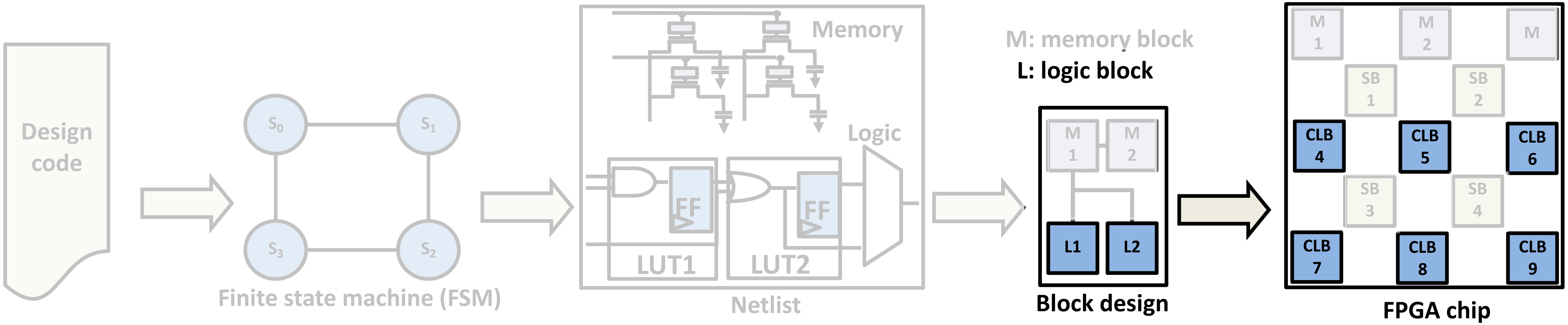


Logic synthesis

Co-placement

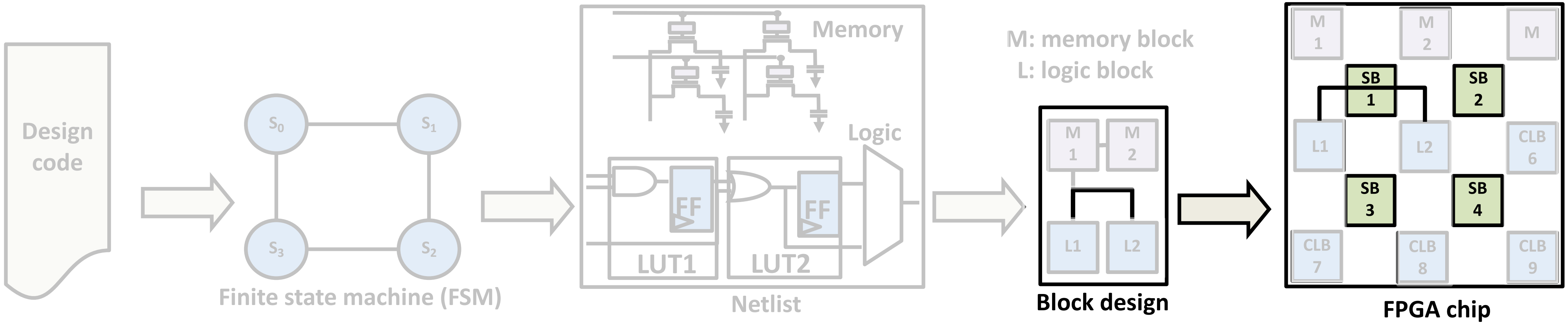
Logic placement

Routing

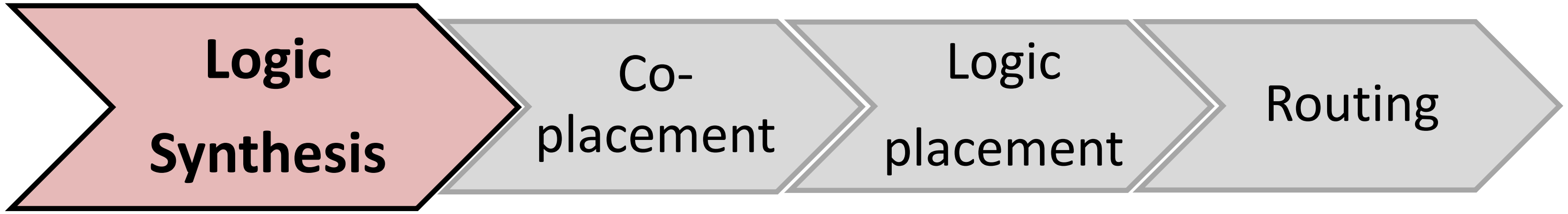


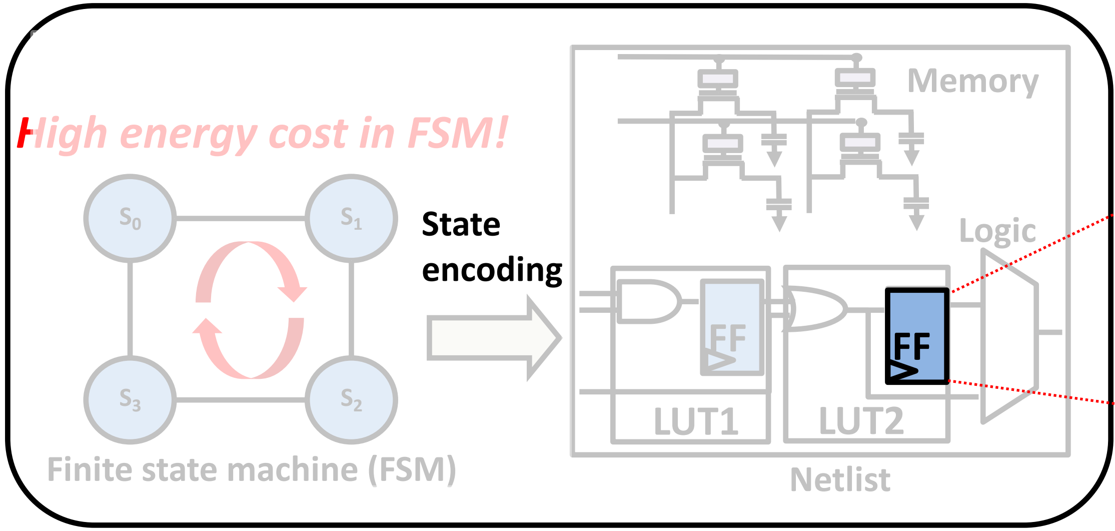
FPGA synthesis introduction

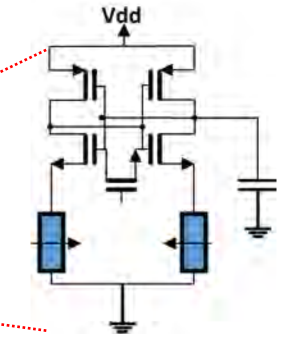
Logic synthesis → Co-placement → Logic placement → **Routing**



Logic synthesis challenge





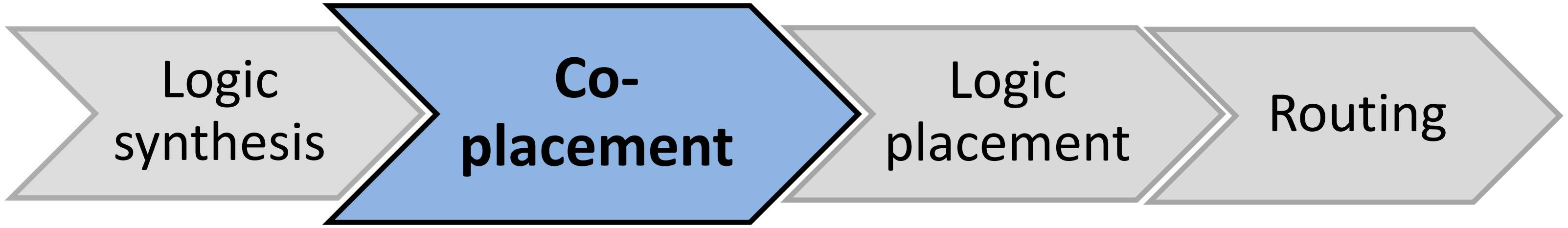


NVM limitations

	Write Energy (J)
SRAM	1.0E-17
STT-MRAM	1.0E-13

Q1: How to tune state encoding to reduce bit-flips in nvFF?

Co-placement challenge

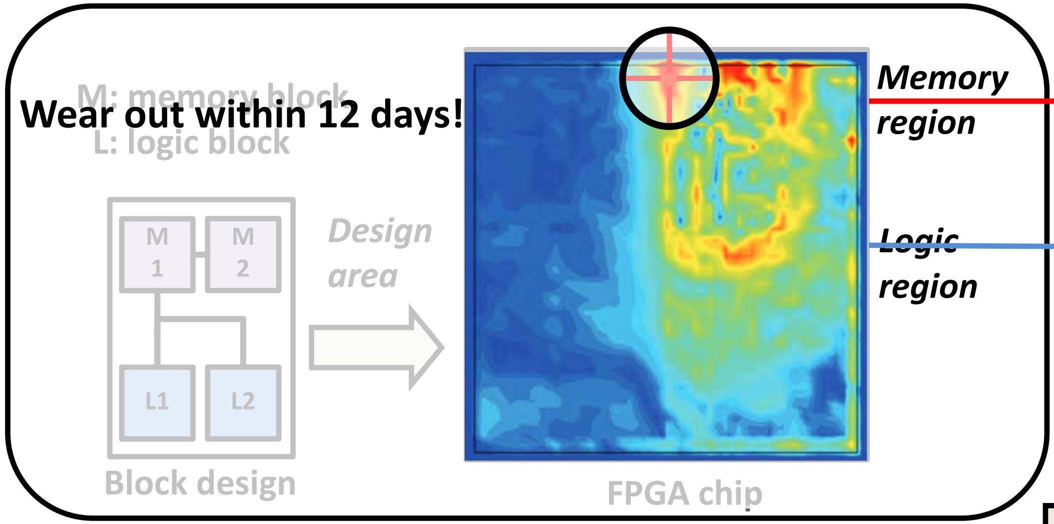


Logic synthesis

Co-placement

Logic placement

Routing



Wear out within 12 days!

Memory updates at 1MHz

Big difference

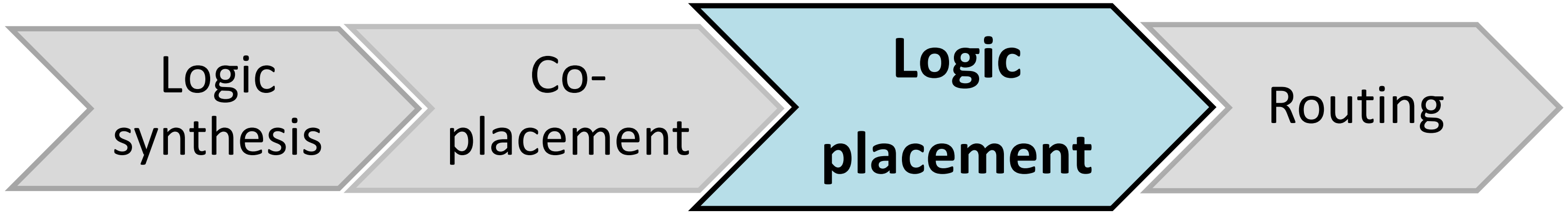
Logic updates once

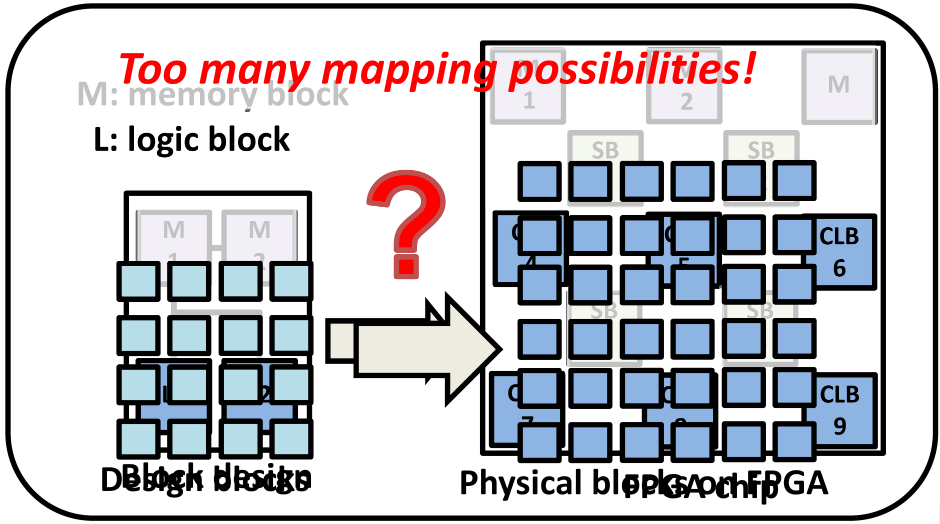
NVM limitations

	Endurance
SRAM	**1.0E16**
STT-MRAM	**1.0E12**

Q2: How to do logic/memory co-placement to globally balance block lifetime?

Logic placement challenge

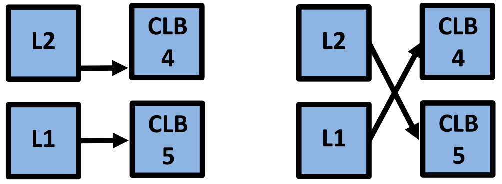

Logic synthesis → Co-placement → **Logic placement** → Routing


Too many mapping possibilities!

M: memory block

L: logic block

+



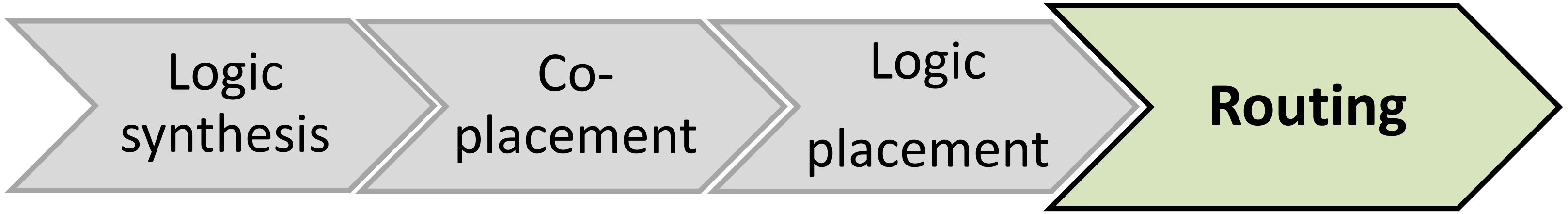
Different mappings cause different number of writes

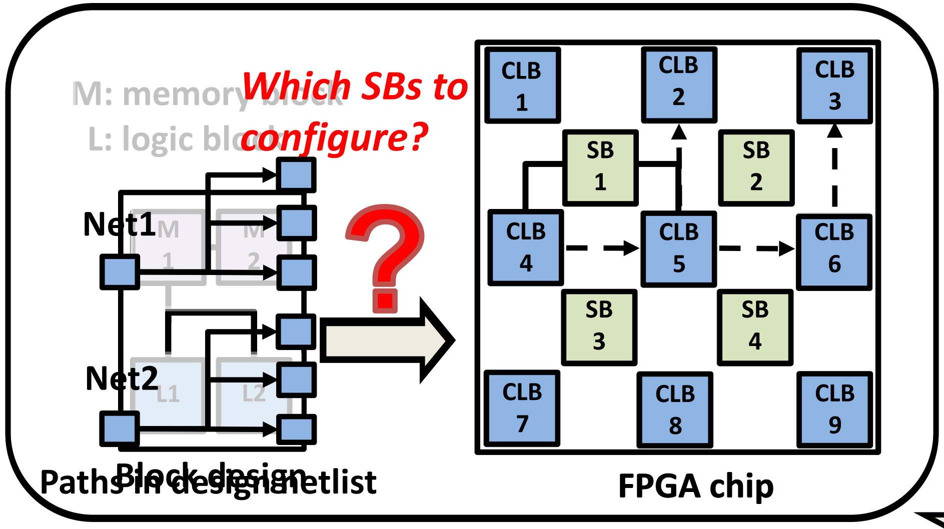
NVM limitations

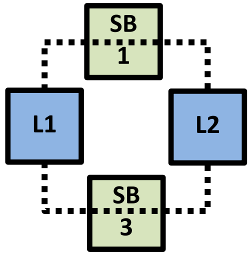
	Endurance	Write time (ns)	Write Energy (J)
SRAM	1.0E16	0.2	1.0E-17
STT-MRAM	1.0E12	35	1.0E-13

Q3: How to explore the huge design space to minimize bit flips in logic placement?

Routing challenge







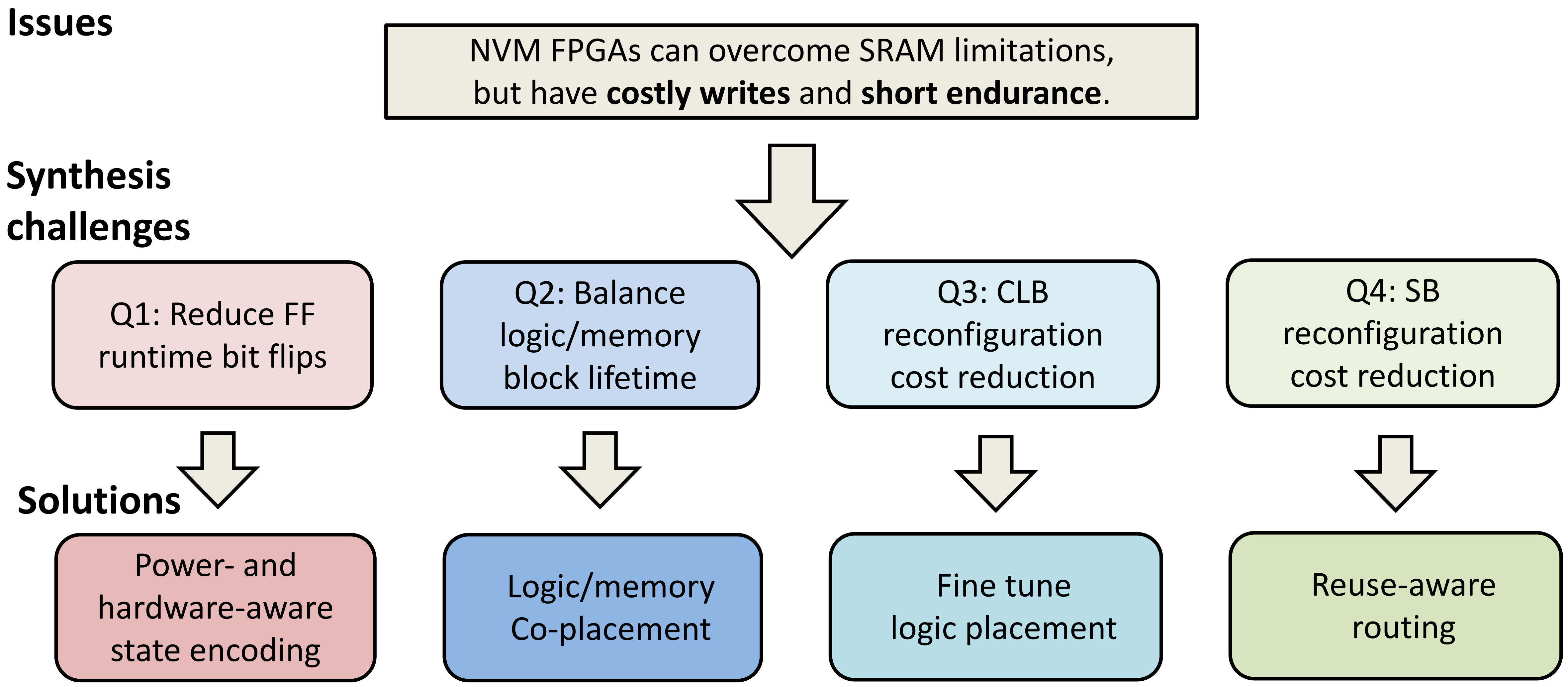
Choose which SB to connect?

NVM limitations

	Endurance	Write time (ns)	Write Energy (J)
SRAM	1.0E16	0.2	1.0E-17
STT-MRAM	1.0E12	35	1.0E-13

Q4: How to reuse paths in routing to minimize SB reconfiguration cost?

Overview



Issues

NVM FPGAs can overcome SRAM limitations, but have **costly writes** and **short endurance**.

Synthesis challenges

Q1: Reduce FF runtime bit flips

Q2: Balance logic/memory block lifetime

Q3: CLB reconfiguration cost reduction

Q4: SB reconfiguration cost reduction

Solutions

Power- and hardware-aware state encoding

Logic/memory Co-placement

Fine tune logic placement

Reuse-aware routing

Our research work

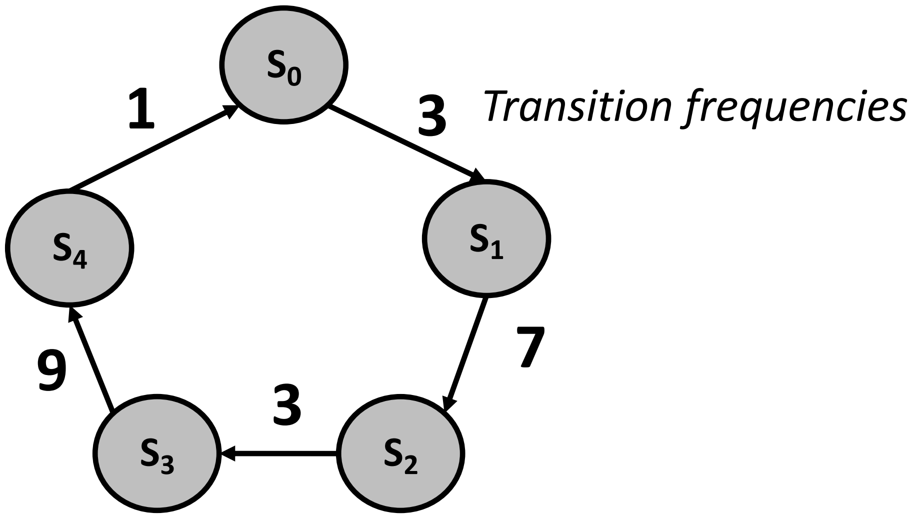
Multi-level Non-volatile FPGA Synthesis to Empower Efficient Self-adaptive System Implementations

Topics	Publications
Logic synthesis	1. Y. Xue, M. McIlvaine and C. Yang, "Power-aware and Cost- efficient State Encoding in Non-volatile memory based FPGAs," in 25th International Conference on Very Large Scale Integration (**VLSI-SoC**), 2017
Logic/memory co-placement	2. Y. Xue, C. Yang and J. Hu, "Age-aware Logic and Memory Co-Placement for RRAM-FPGAs," in 54th Design Automation Conference (**DAC**), 2017
Logic placement	3. Y. Xue, P. Cronin, C. Yang and J. Hu, "Fine-tuning CLB Placement to Speed up Reconfigurations in NVM-based FPGAs," in 25th International Conference on Field-Programmable Logic and Applications (**FPL**), 2015 4. Y. Xue, P. Cronin, C. Yang and J. Hu, "Non-Volatile Memories in FPGAs: Exploiting Logic Similarity to Accelerate Reconfiguration and Increase Programming Cycles," in 23rd International Conference on Very Large Scale Integration (**VLSI-SoC**), 2015
Routing	5. Y. Xue, P. Cronin, C. Yang and J. Hu, "Routing Path Reuse Maximization for Efficient NV-FPGA Reconfiguration," in 21st Asia and South Pacific Design Automation Conference (**ASP-DAC**), 2016 6. Y. Xue and C. Yang, "Path Reuse-aware Routing for Non-volatile Memory based FPGAs," in Integration, the VLSI Journal (**Integration**), Volume 10, Issue 5, 2016

Outline

- Introduction
- **Rethink about FPGA synthesis**
  1. **Logic synthesis**
  2. Logic/memory co-placement
  3. Logic placement
  4. Routing
- Conclusions

State encoding affects nvFF cost and power!

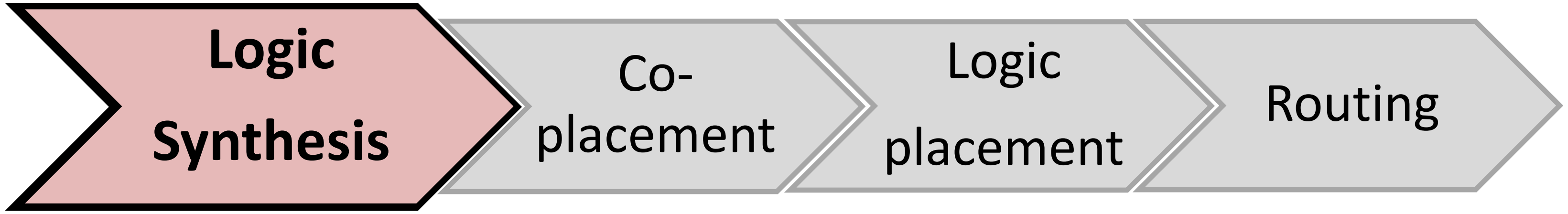


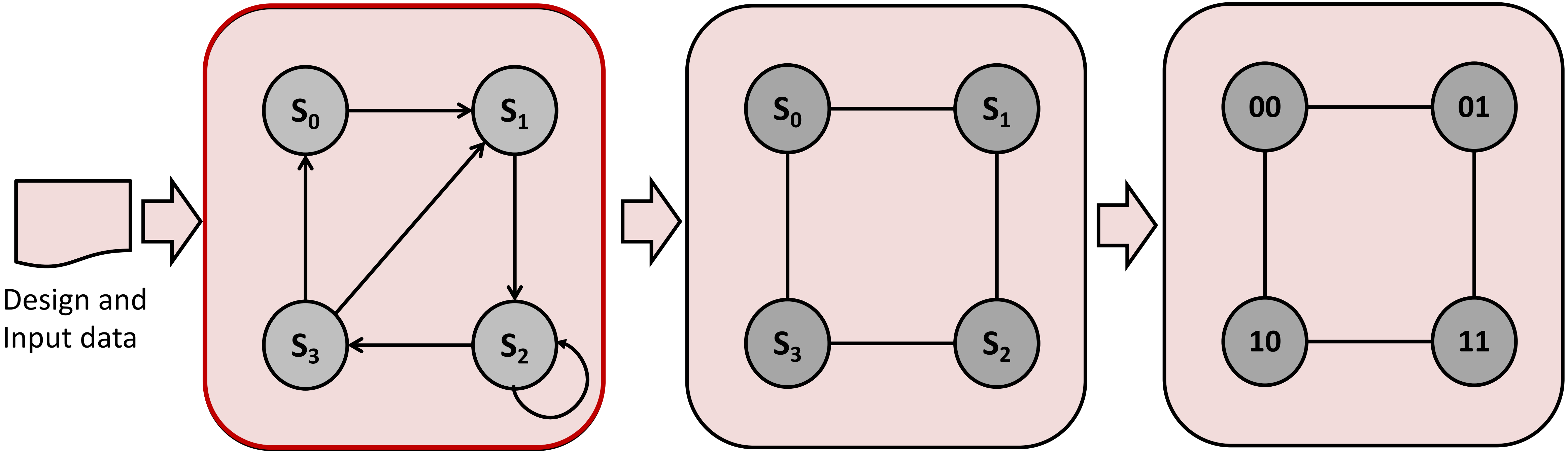
State transition graph example

State	Binary encoding	One-hot encoding
S_0	000	00001
S_1	001	00010
S_2	010	00100
S_3	011	01000
S_4	100	10000
	3 FFs in use 48 bit-flips	5 FFs in use 46 bit-flips

Basic idea: properly encoding for reducing bit flips during state transition.

Logic synthesis





State transition graph construction

Encoding graph conflict removing

Hardware efficient encoding

State transition graph construction

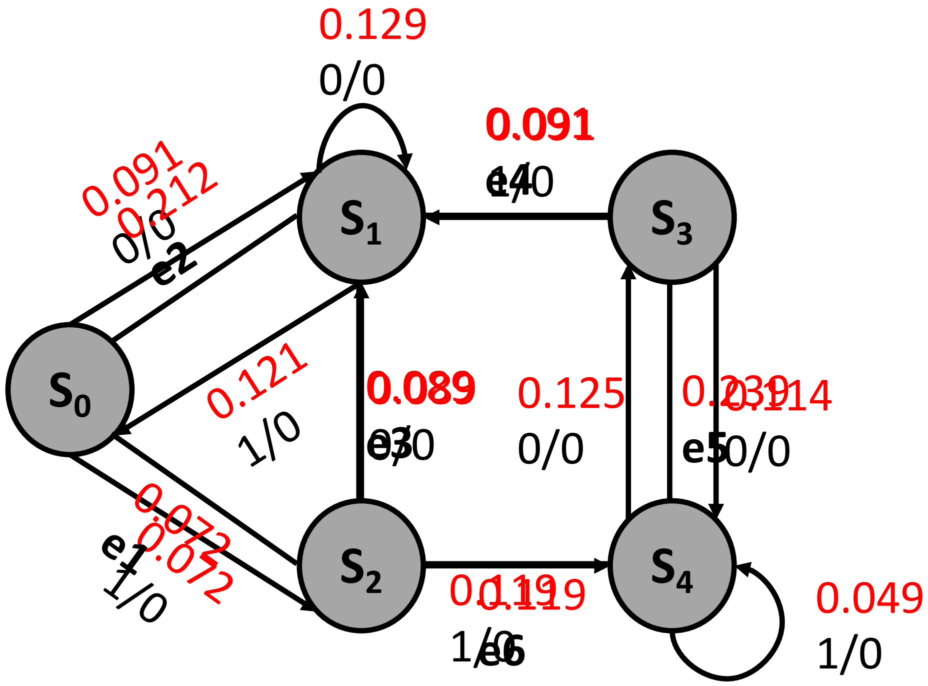
1. Get a directed transition graph G=(S,E,W)

2. Covert to an undirected transition graph G=(S,E,W)

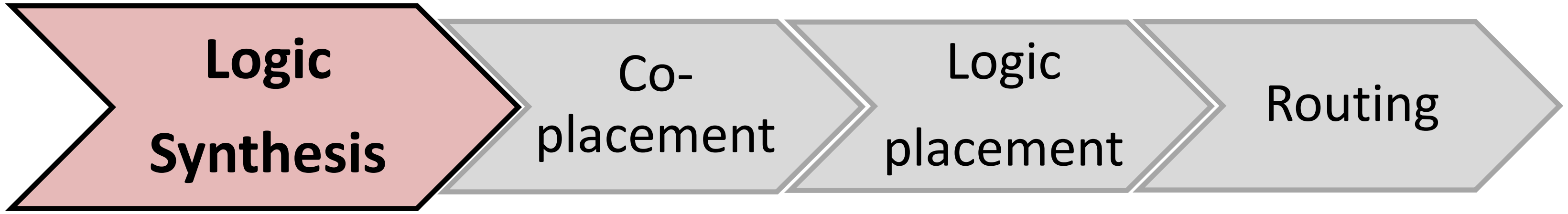
a) Remove self-transition edges

b) Remove direction by merging edges (si→sj) and (sj→si)

S	State set {si,…,sj}
E	Edge set {si↔sj}
W	Weight set {w(si↔sj)}



Logic synthesis

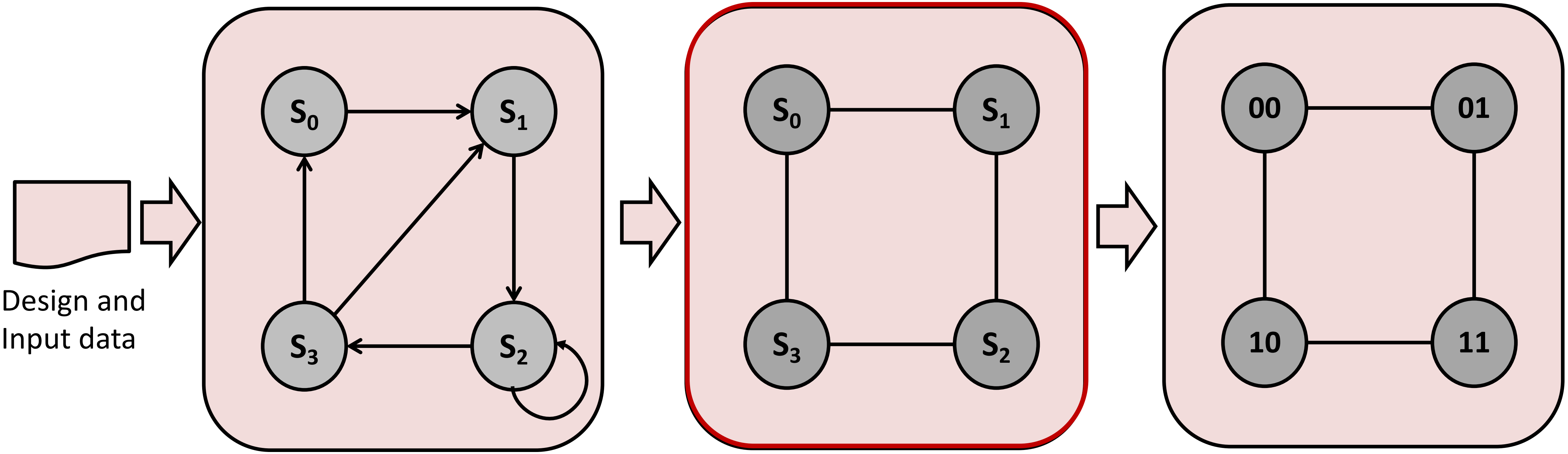


Logic Synthesis

Co-placement

Logic placement

Routing



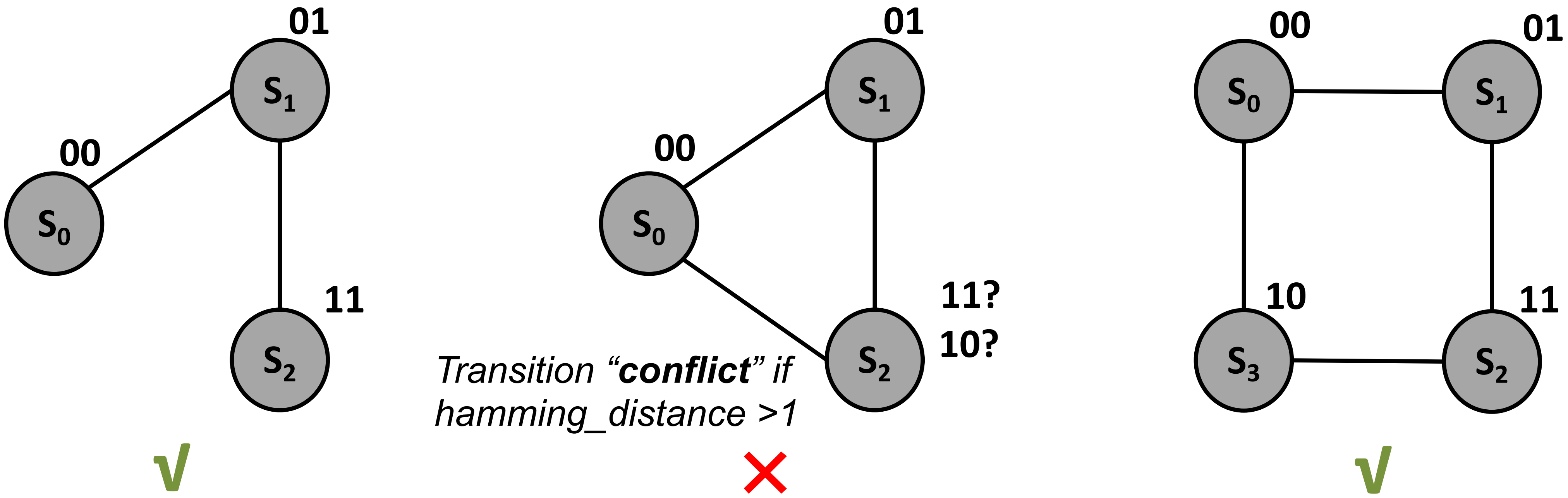
State transition graph construction

Encoding graph conflict removing

Hardware efficient encoding

Encoding graph conflict removing

Ideally, all adjacent states should have a Hamming distance of 1.



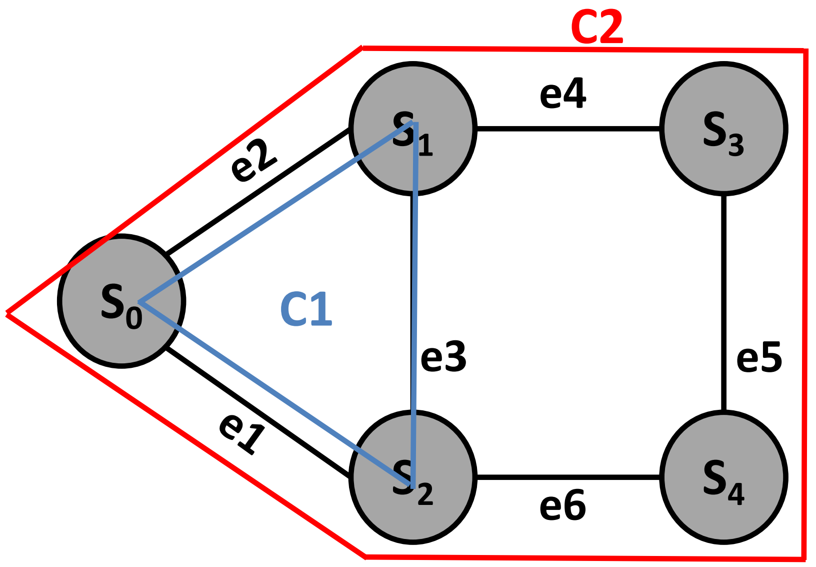
*Transition “**conflict**” if hamming_distance >1*

“conflict” is unavoidable in **odd-node circle**.

Conflict removing algorithm

Step 1: Find all odd-node-circles

Step 2: Remove lowest-cost edges to break found circles



Found circles:

c1={e1,e2,e3}

c2={e1,e2,e4,e5,e6}

Step 1 is based on *Johnson's algorithm* [1].

⇩

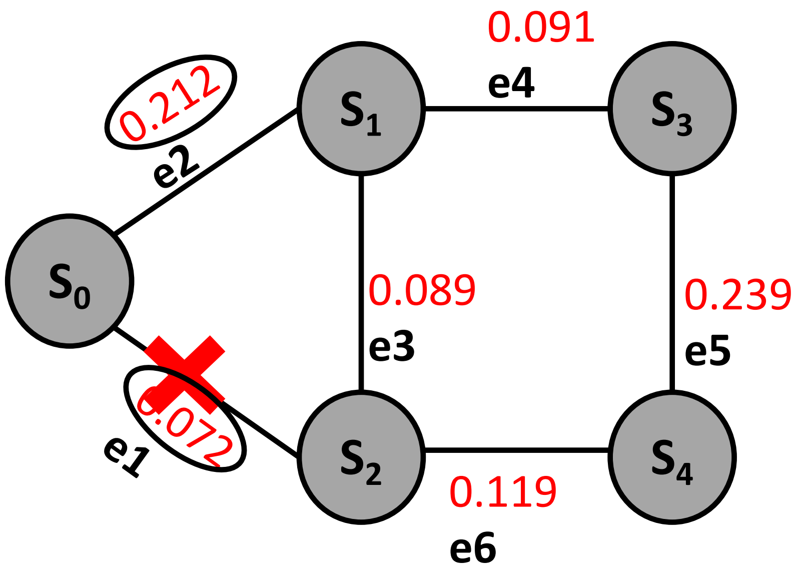
The algorithm fines all elementary circles and only **odd-node circles** are selected.

[1] D. B. Johnson, "Finding all the elementary circuits of a directed graph," in *SIAM*, vol. 4, 1975.

Conflict removing algorithm

Step 1: Find all odd-node-circles

Step 2: Remove lowest-cost edges to break found circles ⇨ ***minimum set cover problem***



Step 2 is based on *Quine-McCluskey algorithm* [2].

⇩

The algorithm offers e1 or e2 to delete.
e1 is selected since it has lower cost.

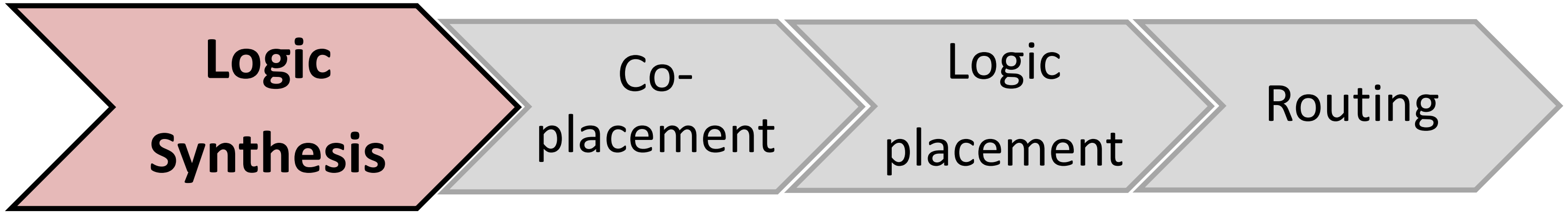
Found circles:

c1={e1,e2,e3}

c2={e1,e2,e4,e5,e6}

[2] A. T. T. Choy, *Digital Logic Design*, 2nd ed. McGraw-Hill, 2011.

Logic synthesis

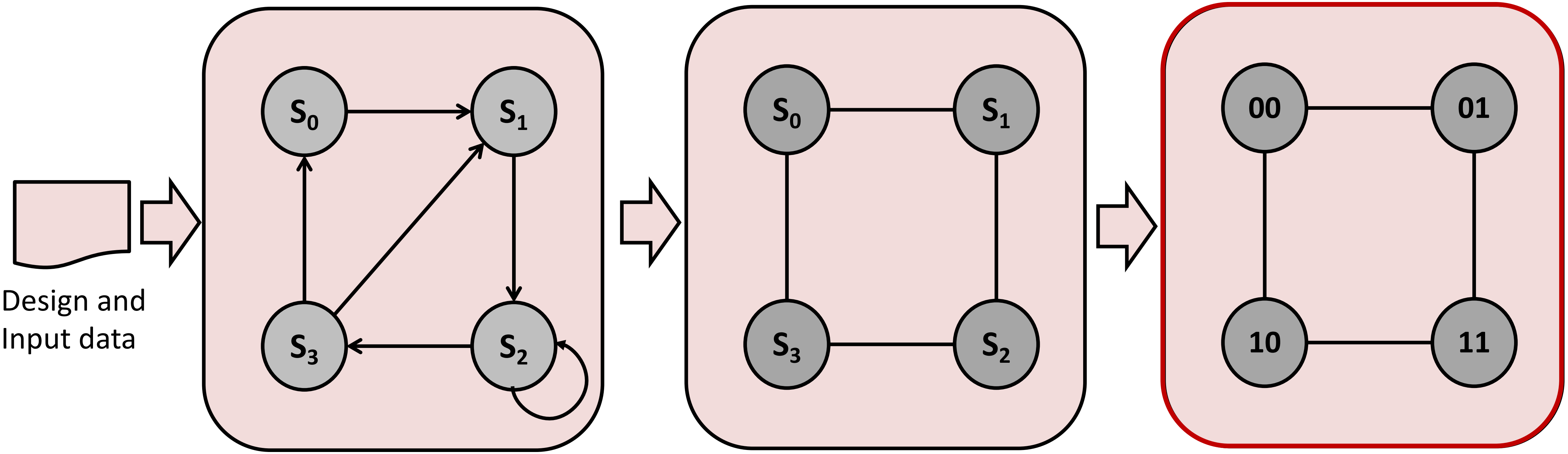


Logic Synthesis

Co-placement

Logic placement

Routing



Design and Input data

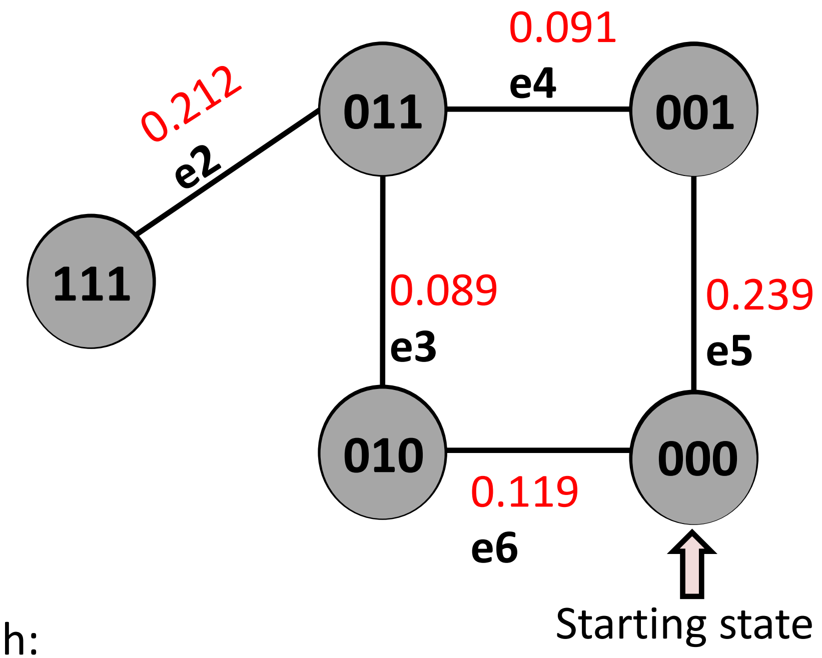
State transition graph construction

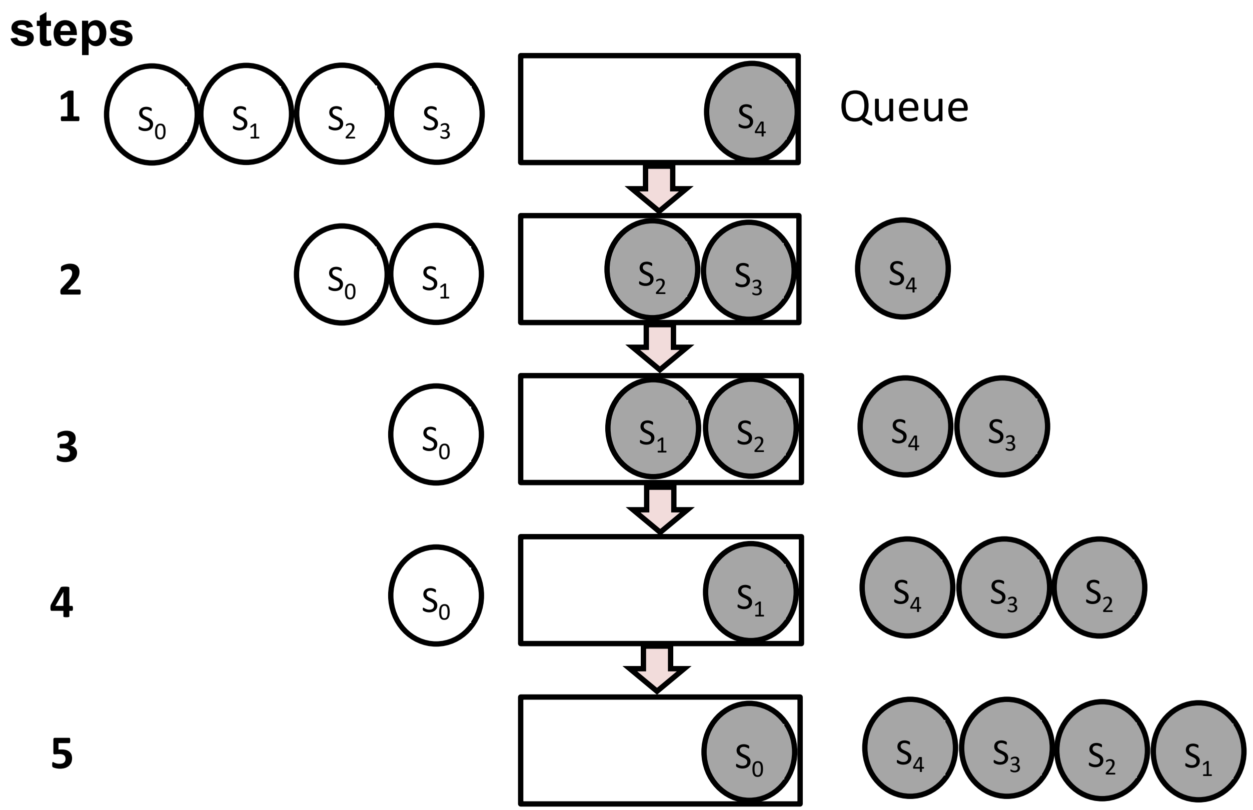
Encoding graph conflict removing

Hardware efficient encoding

Hardware efficient encoding

Starting state is the state with maximum weight summation.
Encoding order is based on Breadth-First Search (BFS).





Code length:

$$L = max(\lceil log_2 N\rceil, \mathrm{max}\_degree)=3$$

N: number of total states
Max_degree: maximum vertex degree of all states

Experiment Setup

Sets	Benchmarks	Number of FFs	Number of bit-flips
ISCAS benchmarks [1]	s386	14	3274
	s820	24	8092
	s832	26	7894
	s1488	43	12632
	s298	220	38674
Real applications [2]	pval	28	7950
	huff	46	14120
	clarissa-str1	131	41482
	clarissa-str2	496	141784
	xval	742	290276

Baseline ⇨

Encoding schemes	Descriptions
One-hot	Standard FPGA encoding scheme
Minimum-Bit-Change (MBC)	Traditional encoding with simple heuristic
Maximum Spanning Tree (MST) [3]	Competitive power-aware encoding
Proposed hardware efficient	Power and hardware aware encoding

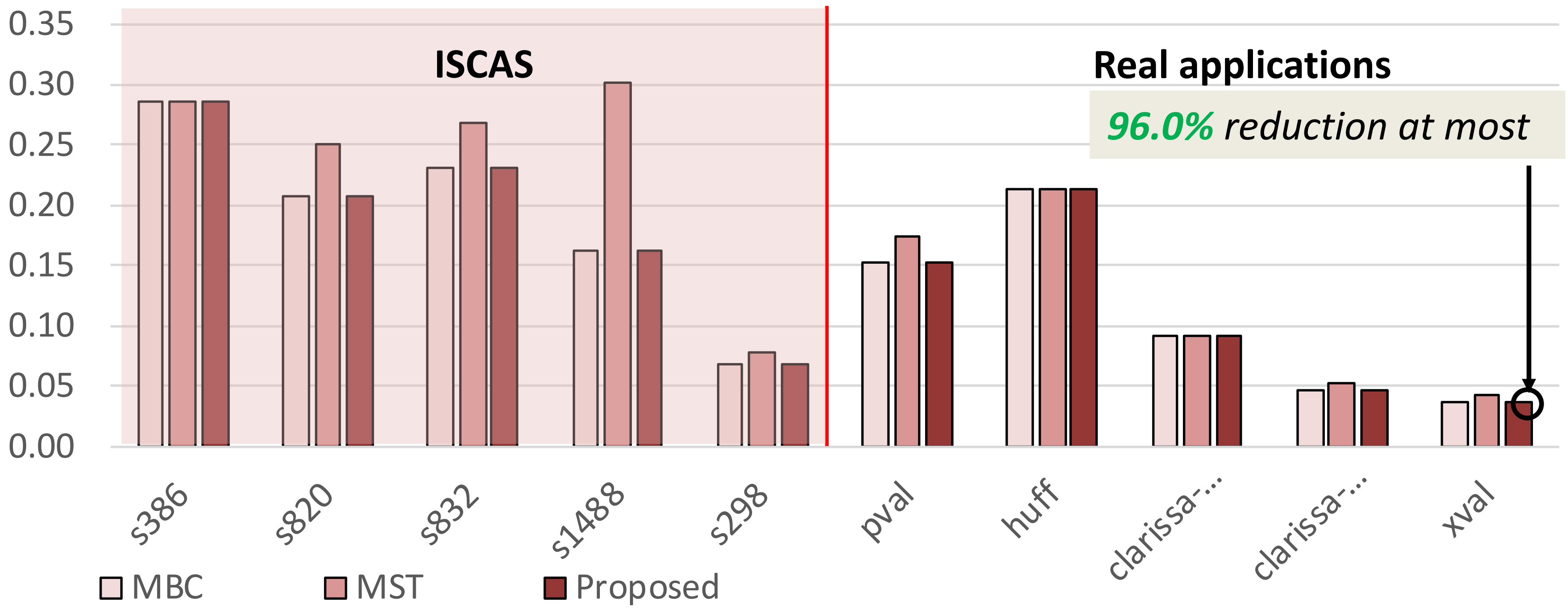
[1] M. Hansen, H. Yalcin, and J. P. Hayes, “Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering,” in *IEEE Design and Test*, vol. 16, no. 3, Jul. 1999, pp. 72-80.

[2] Z. Zhao, B. Wu, and X. Shen, “Challenging the embarrassingly sequential: parallelizing finite state machine-based computations through principled speculation,” in *ASPLOS*, 2014, pp. 543–558.

[3] W. Noth and R. Kolla, “Spanning tree based state encoding for low power dissipation,” in *DATE*, 1999, pp. 1–7.

Normalized flip-flop count

Fewer flip-flops, more hardware efficient



96.0% reduction at most

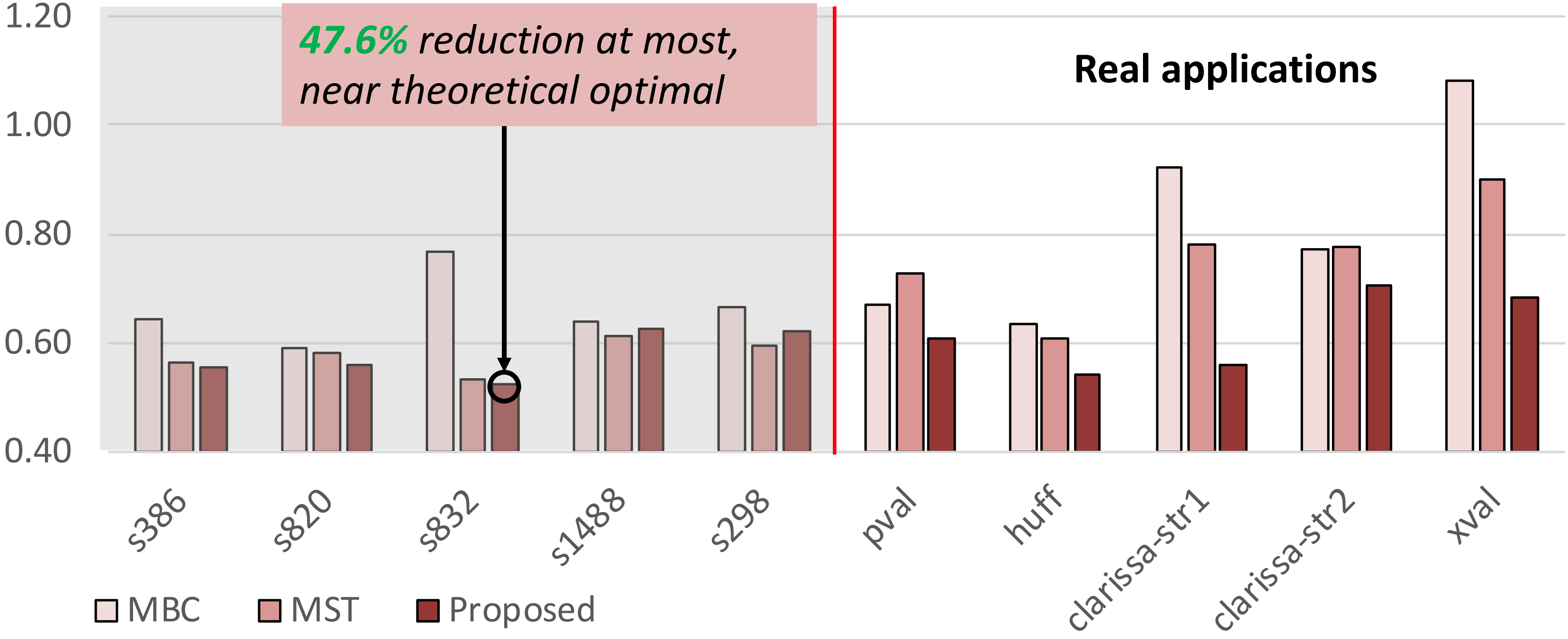
Assume one-hot uses 1 flip-flop.

Minimum-Bit-Change (MBC) Maximum Spanning Tree (MST)

“Proposed” use minimal flip-flops, same as MBC.

Normalized bit flip count

Fewer bit-flops, lower power consumption



47.6% reduction at most, near theoretical optimal

Real applications

*Each state transition causes 2 bit flips in **one-hot**, and requires 1 bit flip at minimum.*

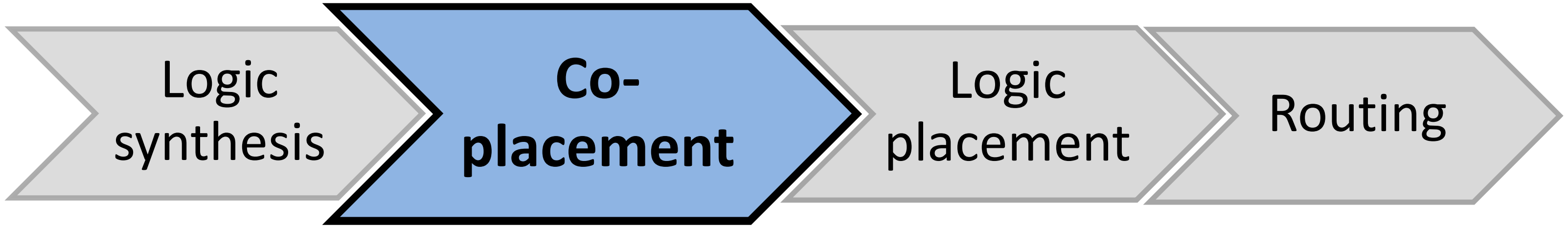
*Therefore theoretical upper bound of reduction is **50%**.*

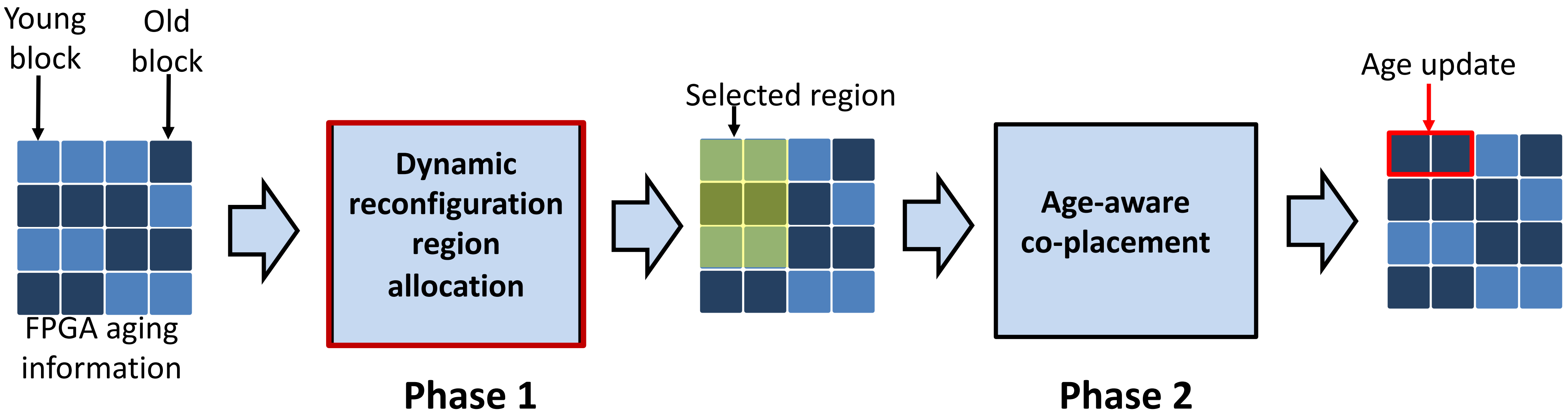
Minimum-Bit-Change (MBC) Maximum Spanning Tree (MST)

Outline

- Introduction
- **Rethink about FPGA synthesis**
  1. Logic synthesis
  2. **Logic/memory co-placement**
  3. Logic placement
  4. Routing
- Conclusions

Co-placement framework



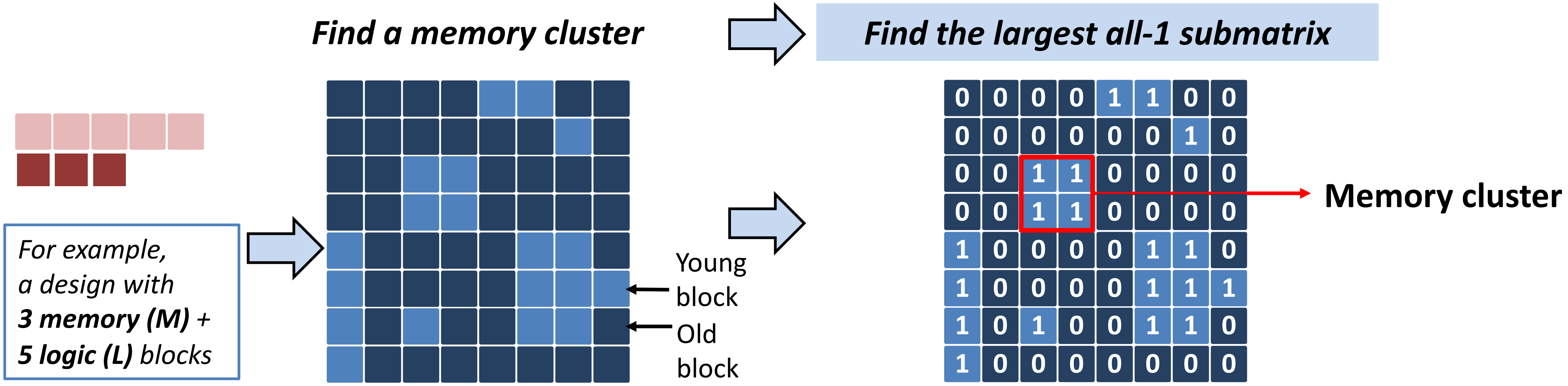


Dynamic reconfiguration region allocation

1. Find a **memory** cluster (young block rectangular region)

Dynamic programming is applied to find an **all-1 submatrix** bigger than memory.

2. Specify a design placement (**memory +logic**) area.



Find a memory cluster

Find the largest all-1 submatrix

For example, a design with ***3 memory (M)*** *+* ***5 logic (L)*** *blocks*

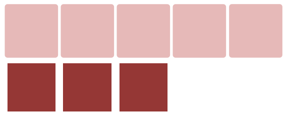
0	0	0	0	1	1	0	0
0	0	0	0	0	0	1	0
0	0	1	1	0	0	0	0
0	0	1	1	0	0	0	0
1	0	0	0	0	1	1	0
1	0	0	0	0	1	1	1
1	0	1	0	0	1	1	0
1	0	0	0	0	0	0	0

Dynamic reconfiguration region allocation

1. Find a **memory** cluster (young block rectangular region)

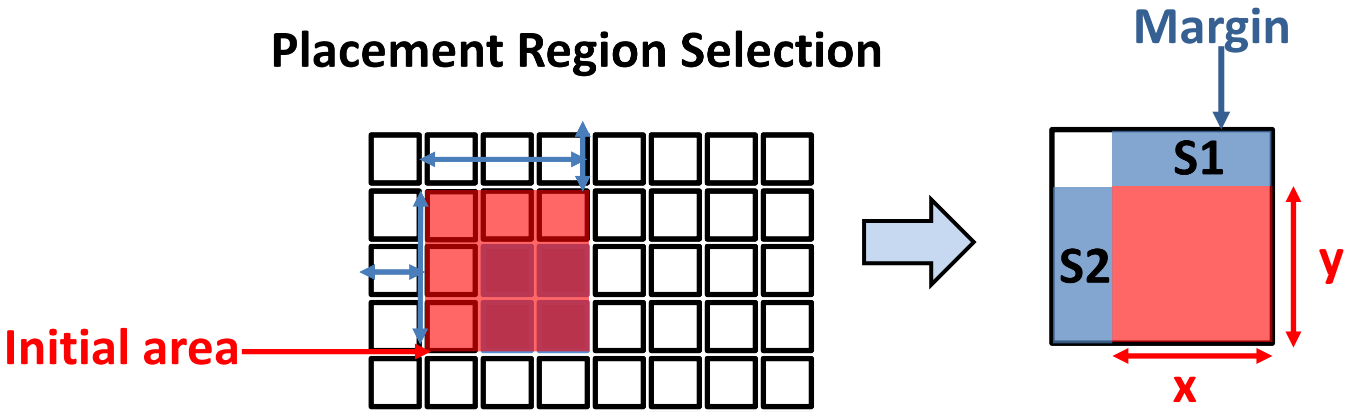
2. Specify a design placement (**memory +logic**) area.

Reshape the area to **minimize** the margin area for performance.



For example,
a design with
***3 memory (M)** +*
***5 logic (L)** blocks*

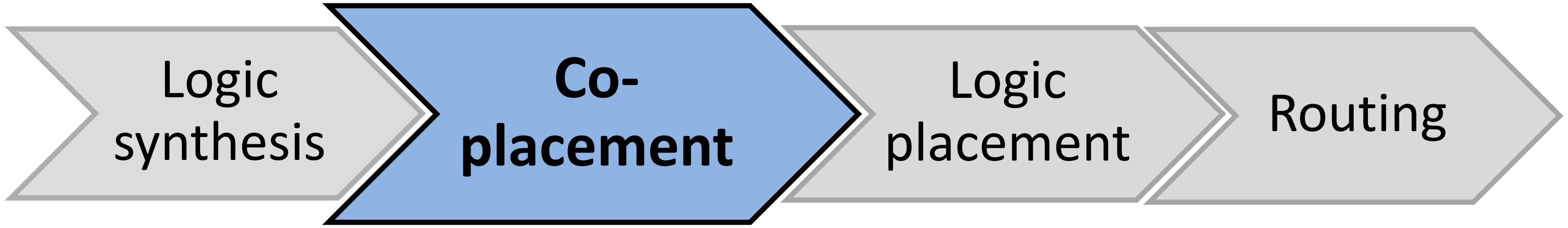
Placement Region Selection



Compute x and y
*to **minimize** S1+S2*

Initialize a **square** area for design including the found cluster (close to corner).

Co-placement framework

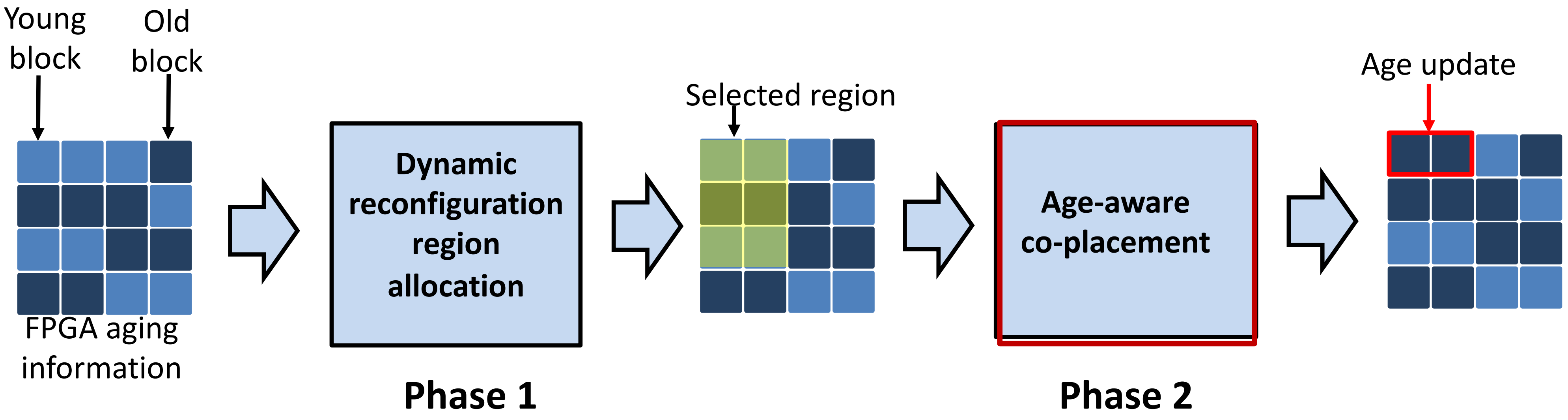


Logic synthesis

Co-placement

Logic placement

Routing



Young block

Old block

FPGA aging information

Dynamic reconfiguration region allocation

Phase 1

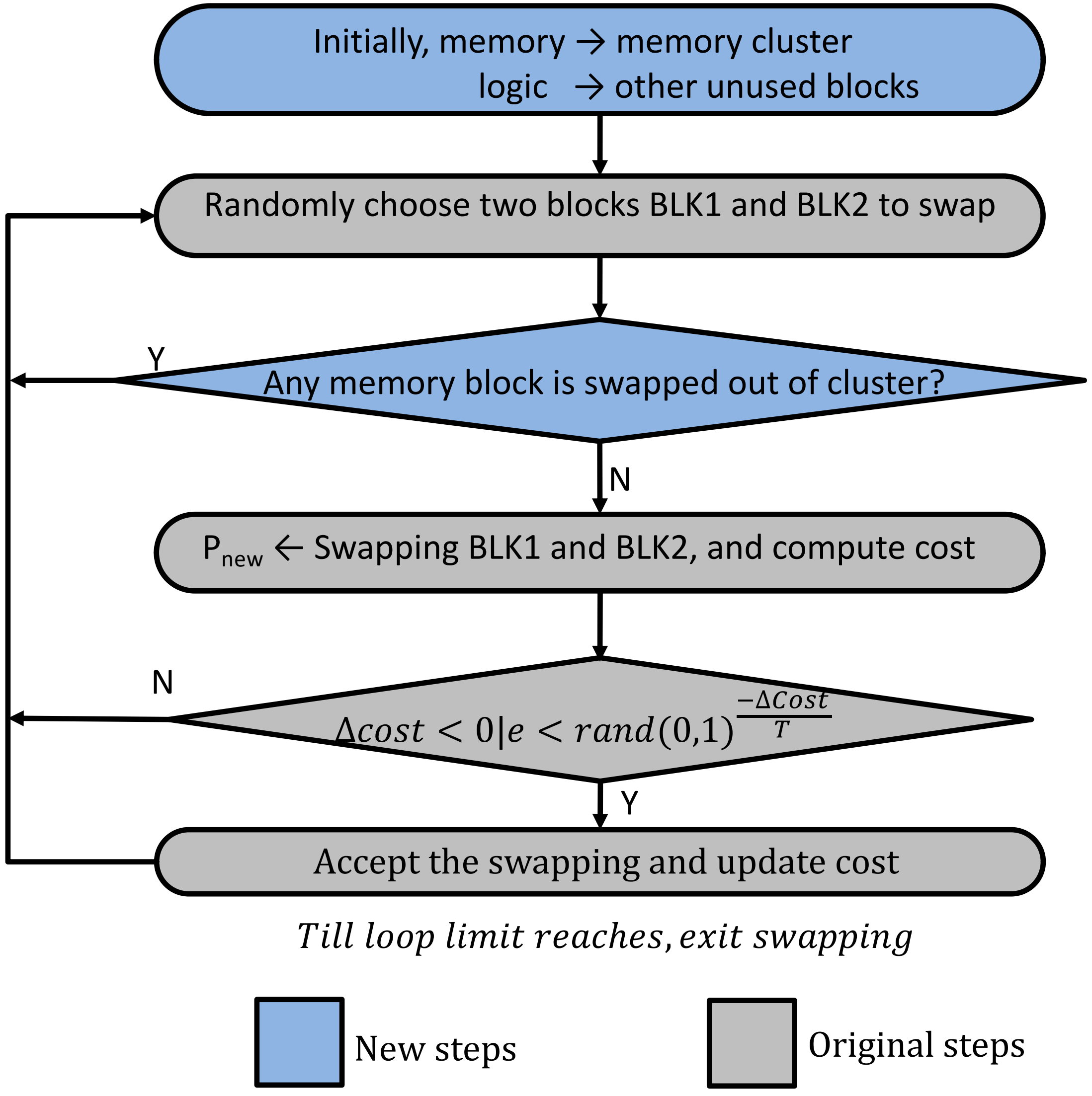
Selected region

Age-aware co-placement

Phase 2

Age update

Age-aware co-placement



Initially, memory → memory cluster
logic → other unused blocks

Randomly choose two blocks BLK1 and BLK2 to swap

Any memory block is swapped out of cluster?

Y

N

P_{new} ← Swapping BLK1 and BLK2, and compute cost

$\Delta cost < 0 | e < rand(0,1)^{\frac{-\Delta Cost}{T}}$

N

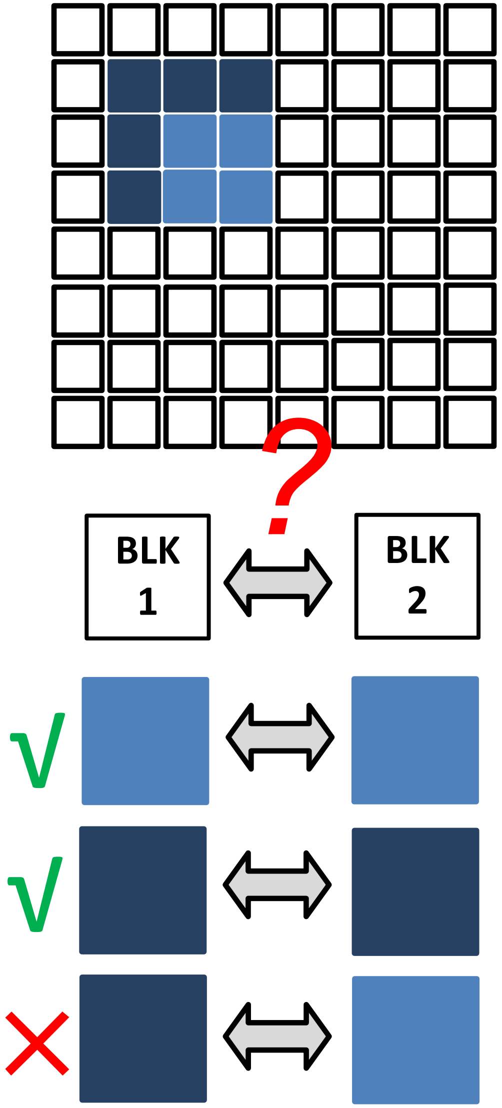
Y

Accept the swapping and update cost

Till loop limit reaches, exit swapping

🟦 New steps

⬜ Original steps



Experimental Setup

- FPGA architecture: Altera Stratix IV
- CAD toolkit: VTR 7.0 [1]

Co-placement methods

Scheme	Baseline (VTR)	Optional region [2] (OR)	Proposed
Region shape flexibility	-	+	++
Region location flexibility	-	+	++
Age aware	-	+	++
Co-placement	-	-	+

10 Titan benchmarks [3]

No	Benchmarks
1	stereo_vision
2	neuron
3	sparcT1_core
4	des90
5	SLAM_spheric
6	dart
7	Bitonic_mesh
8	stap_qrd
9	cholesky_bdti
10	gsm_switch

[1] Jason Luu et al., " VTR 7.0: Next Generation Architecture and CAD System for FPGAs," in *ACM Transactions on Reconfigurable Technology and Systems*, 7(2):1–30, Jun. 2014.

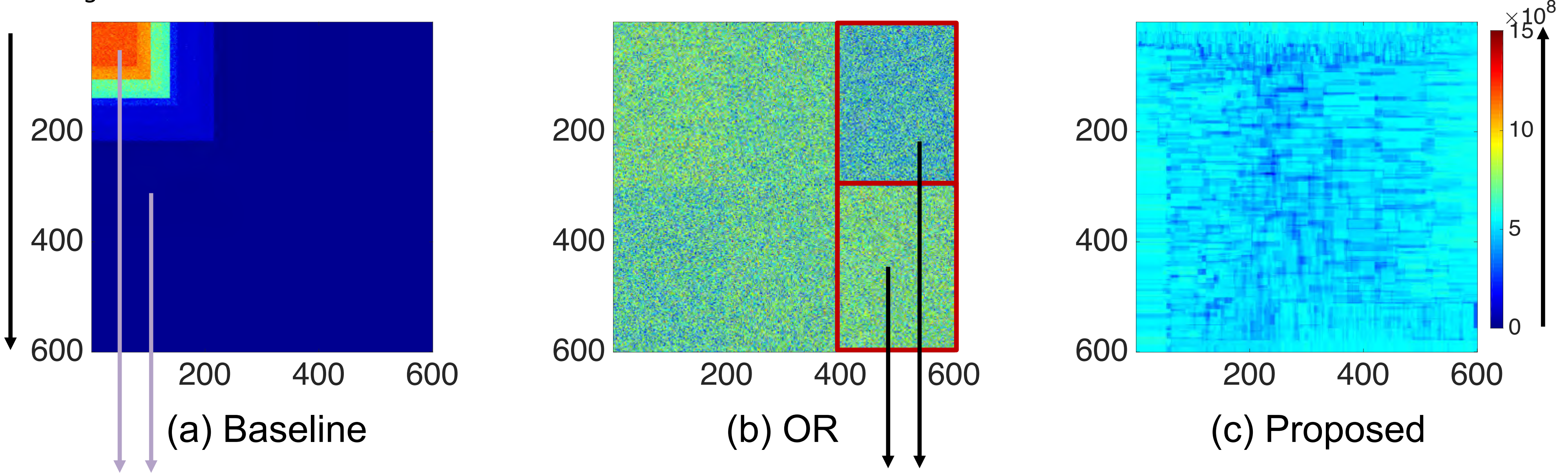
[2] E. Stott, J. S. J. Wong, and P. Y. K. Cheung, "Degradation analysis and mitigation in FPGAs," in *International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2010, pp. 428-433.

[3] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Titan: Enabling large and complex benchmarks in academic CAD," in *Field Programmable Logic and Applications (FPL)*, Sep. 2013, pp. 1-8.

Block write distribution (500 rounds)

The chip is divided into 600×600 grid.

Write counts



(a) Baseline

(b) OR

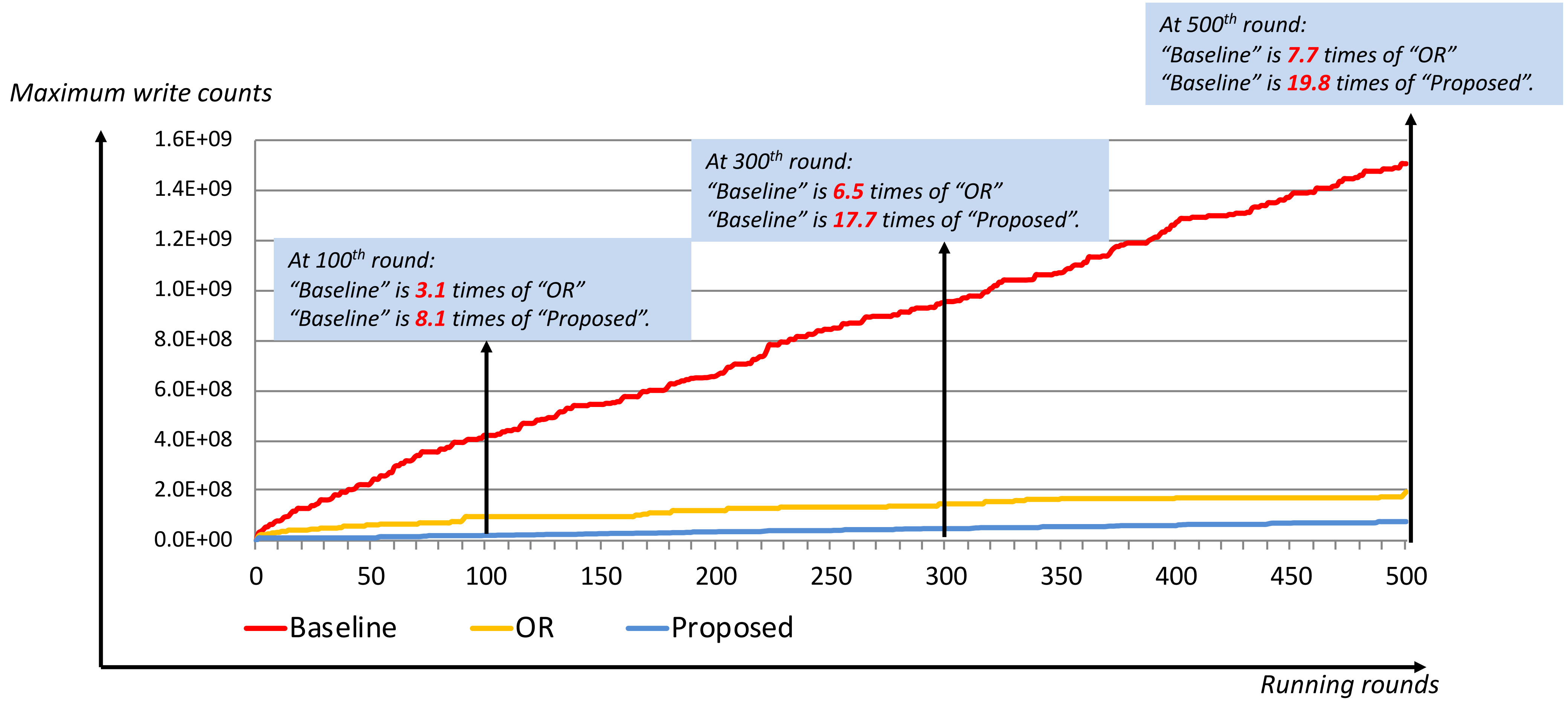
(c) Proposed

Baseline is ***highly skewed*** *in the original VTR.*

OR is more-or-less balanced within each region, but ***unbalanced*** *across regions.*

Proposed is much more balanced than the other two schemes.

Maximum write count curve (500 rounds)



At 100th round:
"Baseline" is ***3.1*** *times of "OR"*
"Baseline" is ***8.1*** *times of "Proposed".*

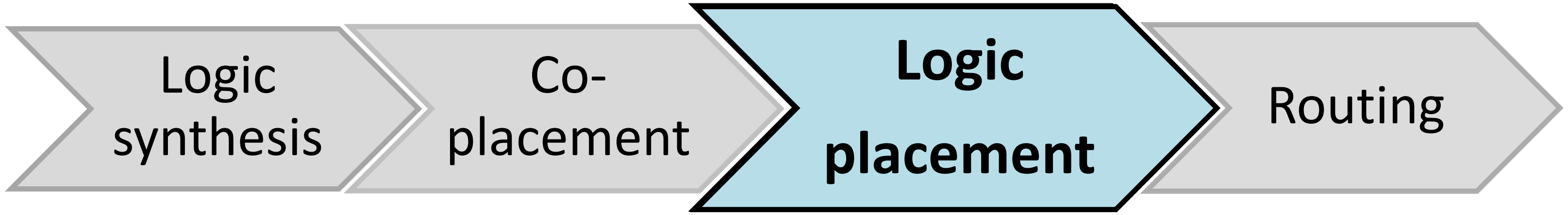
At 300th round:
"Baseline" is ***6.5*** *times of "OR"*
"Baseline" is ***17.7*** *times of "Proposed".*

At 500th round:
"Baseline" is ***7.7*** *times of "OR"*
"Baseline" is ***19.8*** *times of "Proposed".*

Outline

- Introduction
- **Rethink about FPGA synthesis**
  1. Logic synthesis
  2. Logic/memory co-placement
  3. **Logic placement**
  4. Routing
- Conclusions

Logic placement

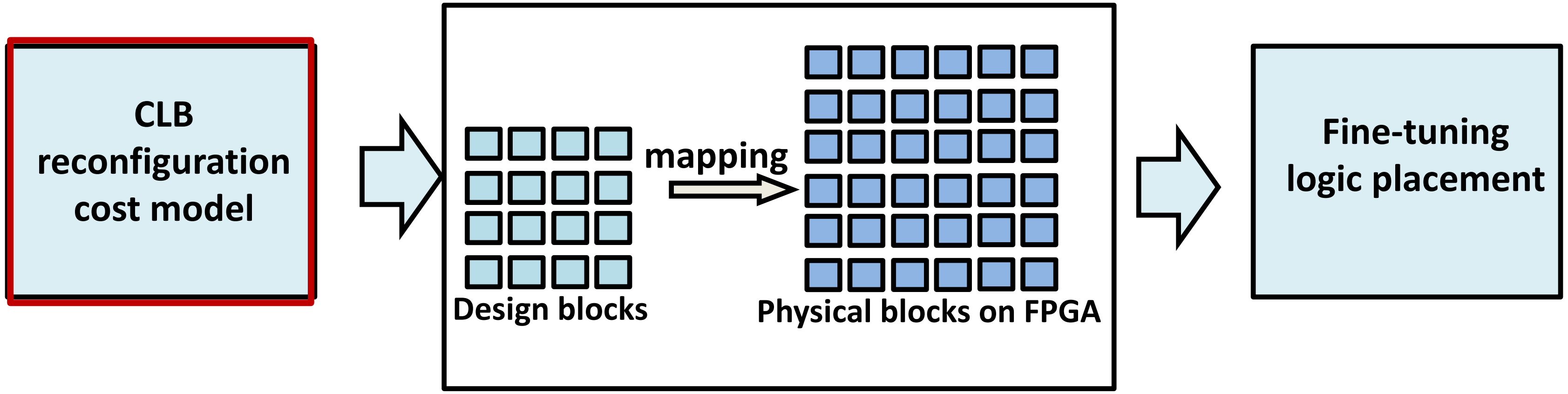


Logic synthesis

Co-placement

Logic placement

Routing



CLB reconfiguration cost model

Design blocks

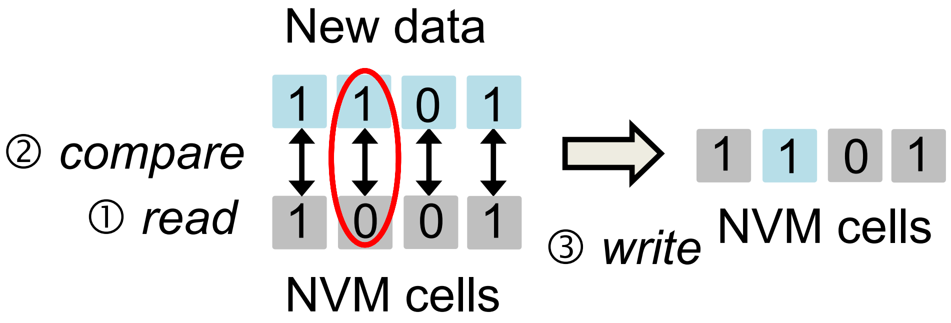
mapping

Physical blocks on FPGA

Fine-tuning logic placement

Write reduction via bit level reuse

Basic idea: **read-before-write** strategy



Type	Read time (ns)	Write time (ns)
SRAM	0.2	0.2
PCM	12	100

Why this works for NVM?

For PCM

Without RBW → 4 writes → 400ns

With RBW → 4 reads+ 1 write → 148ns

Great Improvement!

For SRAM

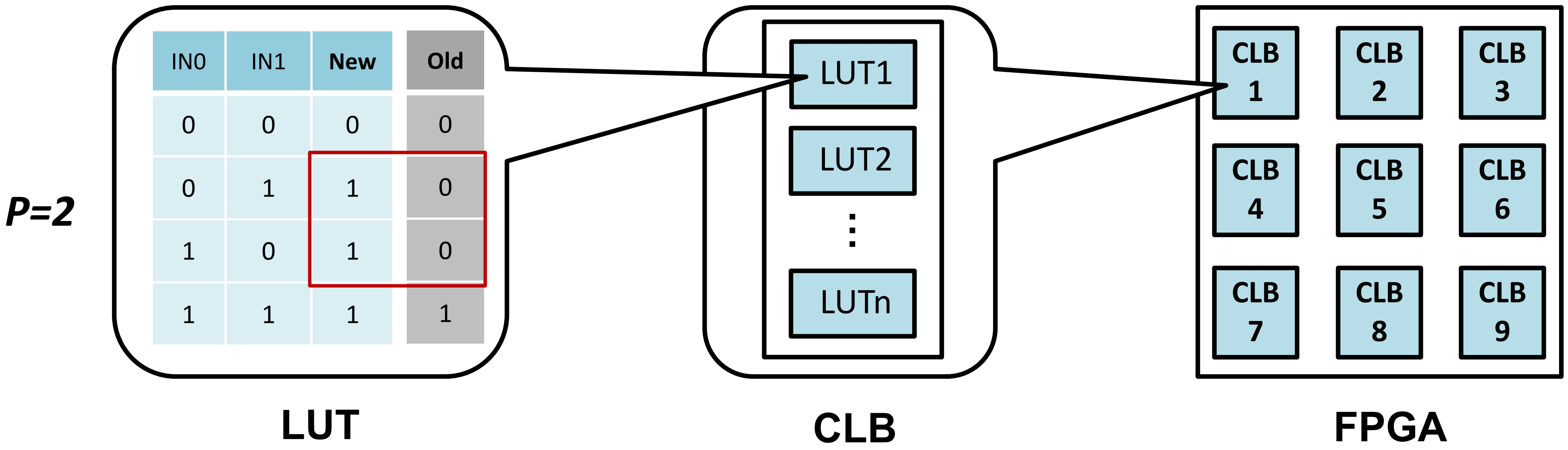
Without RBW → 4 writes → 0.8ns

With RBW → 4 reads+ 1 write → 1ns

No benefit!

Fine-tuning logic placement cost model

Suppose LUTi has ***P*** bit flips

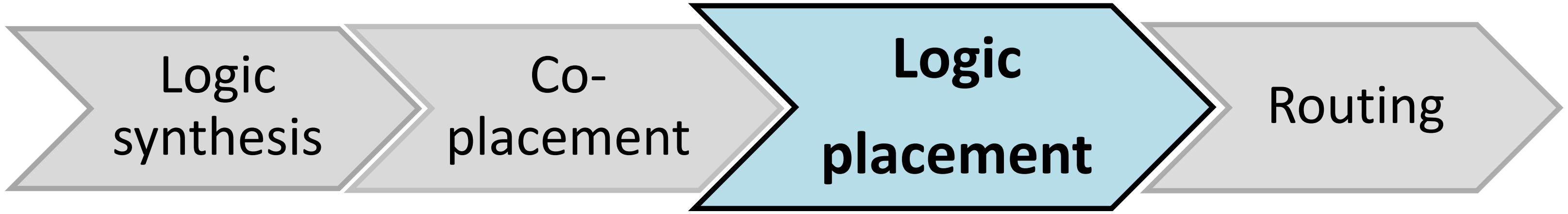


IN0	IN1	New	Old
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

$$RR_{LUT} = P \Rightarrow RR_{CLB} = \sum_{LUT_i \in CLB} RR_{LUT_i} \Rightarrow Cost_{reconfig} = \sum_{i}^{N_{CLB}} RR_{CLB_i}$$

CLB reconfiguration cost is defined by three-level costs.

Logic placement

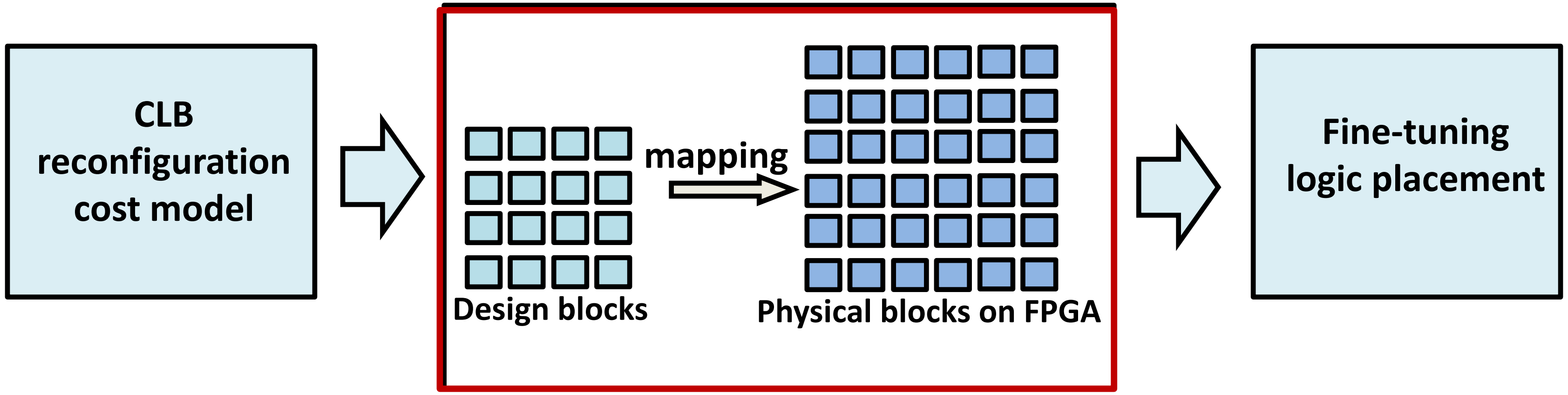


Logic synthesis

Co-placement

Logic placement

Routing



CLB reconfiguration cost model

Design blocks

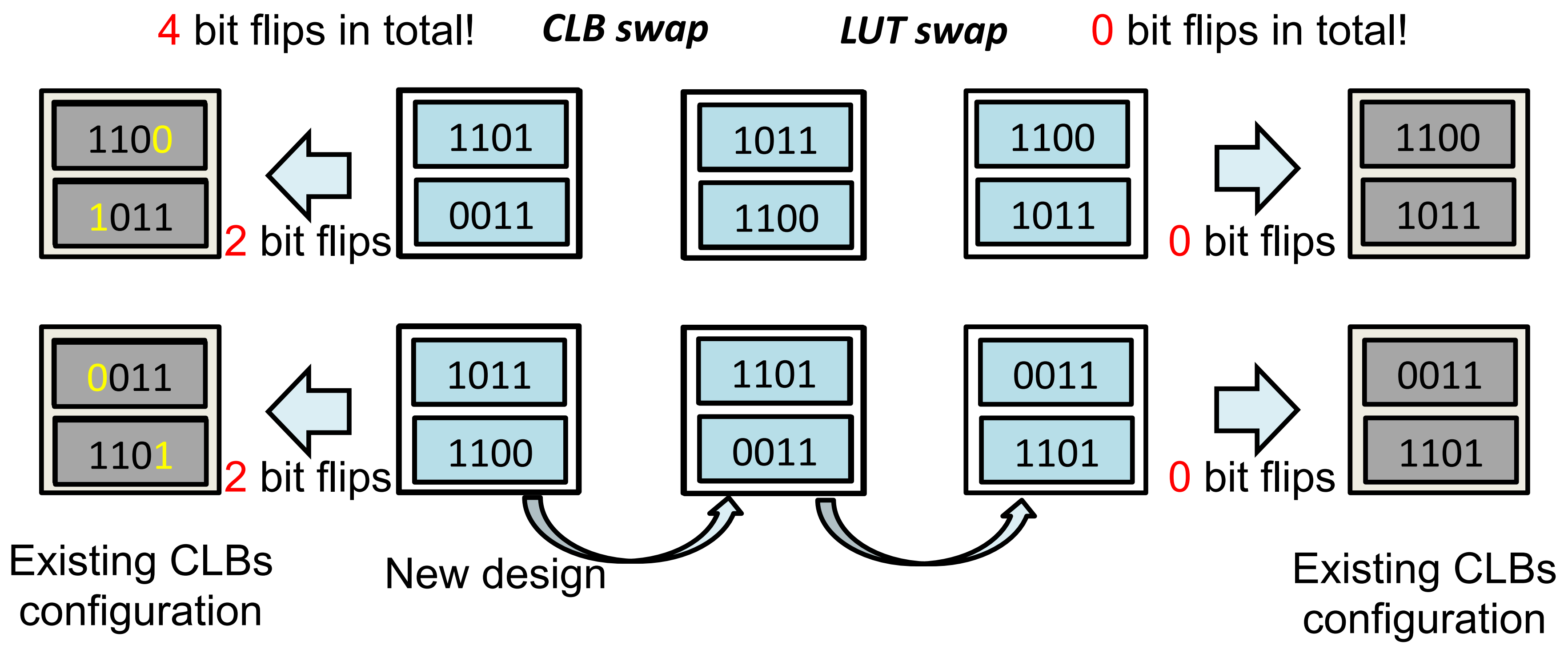
mapping

Physical blocks on FPGA

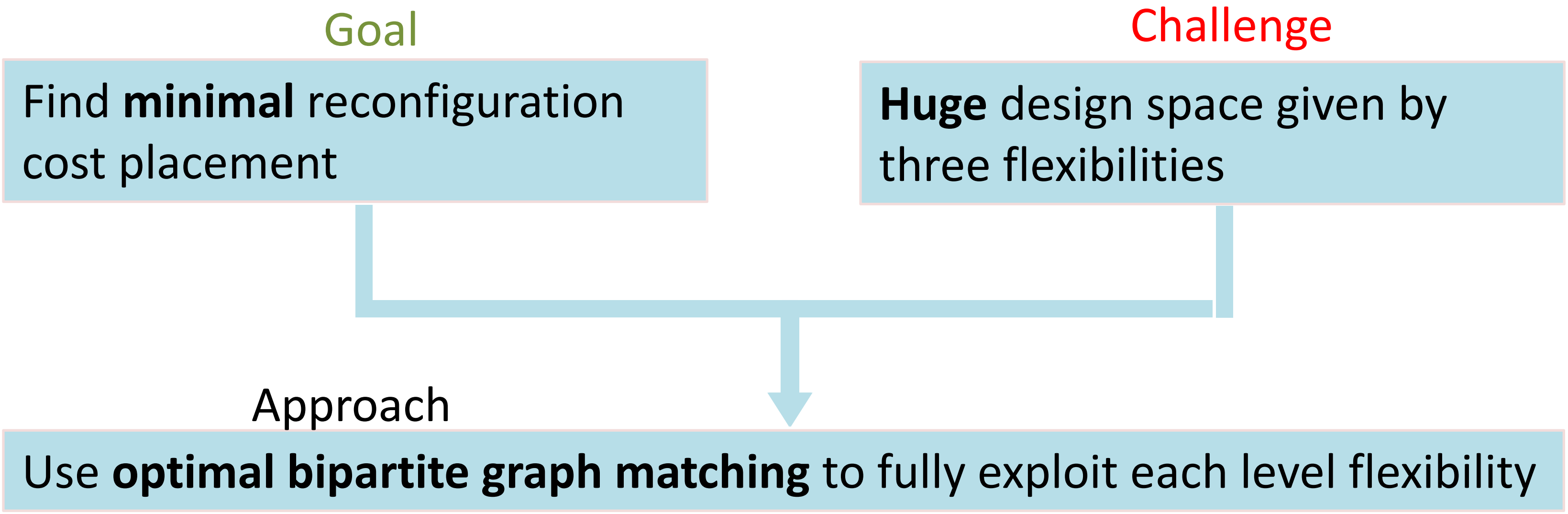
Fine-tuning logic placement

Two-level flexibility

Different **LUT** and **CLB** mappings will result in different CLB cost.



Huge design space to explore!



Goal

Find **minimal** reconfiguration cost placement

Challenge

Huge design space given by three flexibilities

Approach

Use **optimal bipartite graph matching** to fully exploit each level flexibility

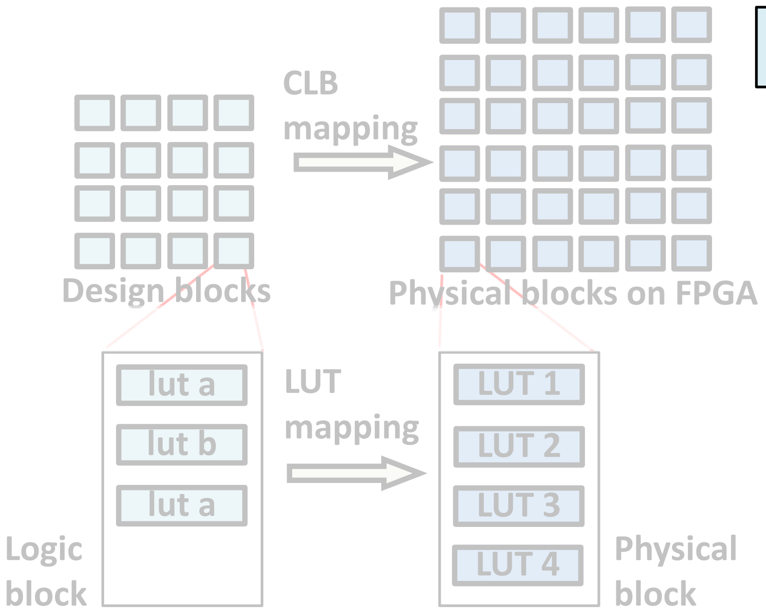
flexibility = $\frac{n!}{(n-m)!} \times \frac{x!}{(x-y)!}$

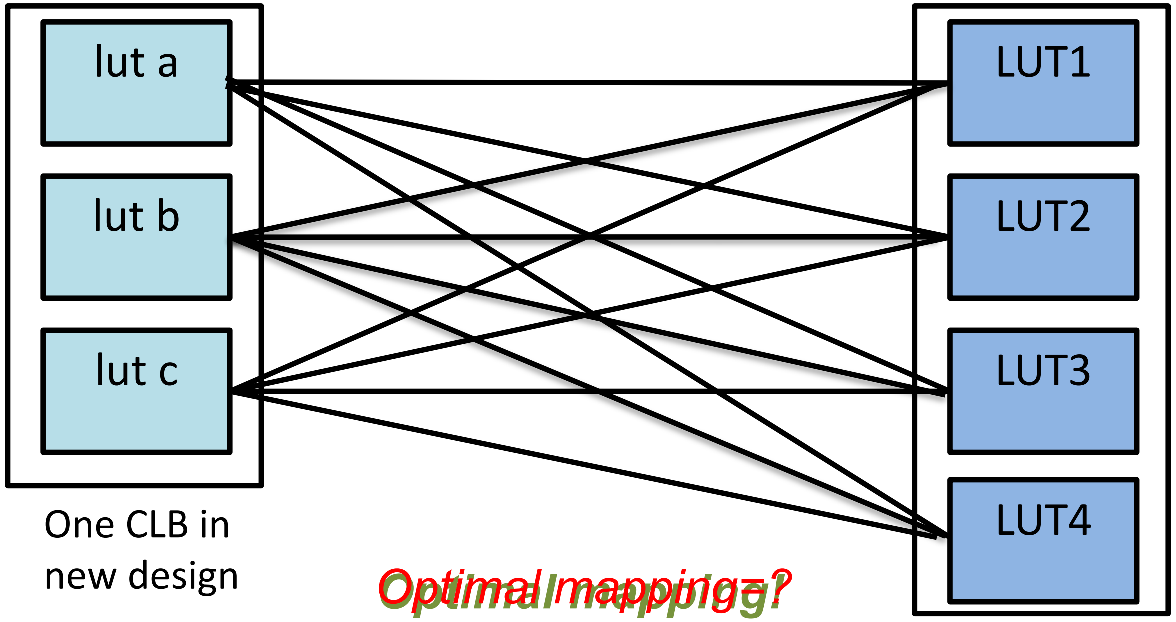
n *total LUTs* ***m*** *LUTs in use*	***x*** *total CLBs* ***y*** *CLBs in use*

- ✓ Use **matrix** to represent all possible cost values at each level
- ✓ Use **Kuhn-Munkres (KM)** algorithm to solve the optimal matching problem

Proposed design space exploration approach

KM is also used for CLB mappings.





Use LUT level as an example

Kuhn-Munkres (KM) algorithm [1]

- ❖ Problem : **Optimal weighted** bipartite mapping
- ❖ Input : Bipartite graph weights (matrix)
- ❖ Output : Optimal one-to-one mapping
- ❖ Complexity: O(n^3)

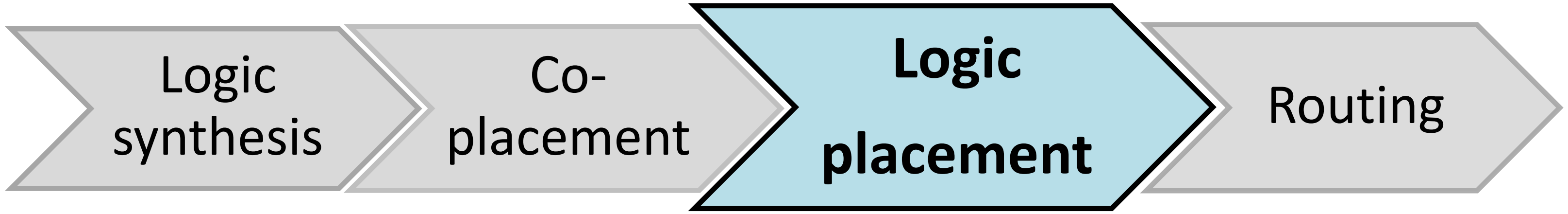
Optimal mapping=?

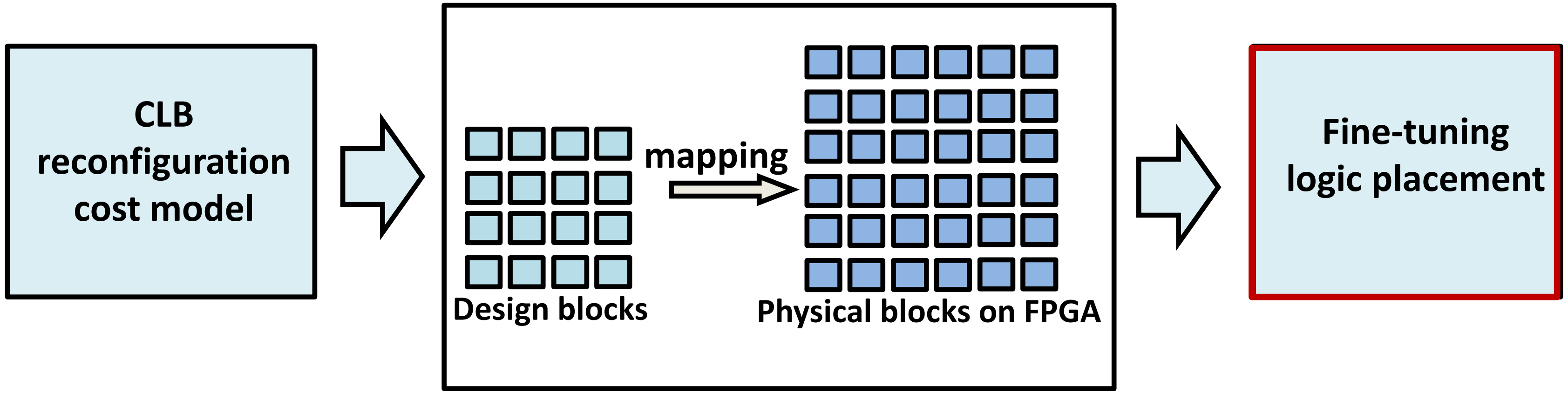
Minimal cost!

$$\text{LUT cost=}\left.\overbrace{\begin{matrix} \boxed{1} & 4 & 4 & 5 \\ 3 & 2 & \boxed{1} & 4 \\ 3 & \boxed{1} & 2 & 3 \end{matrix}}^{\textit{LUT 1-4}}\right\}\textit{lut a-c}$$

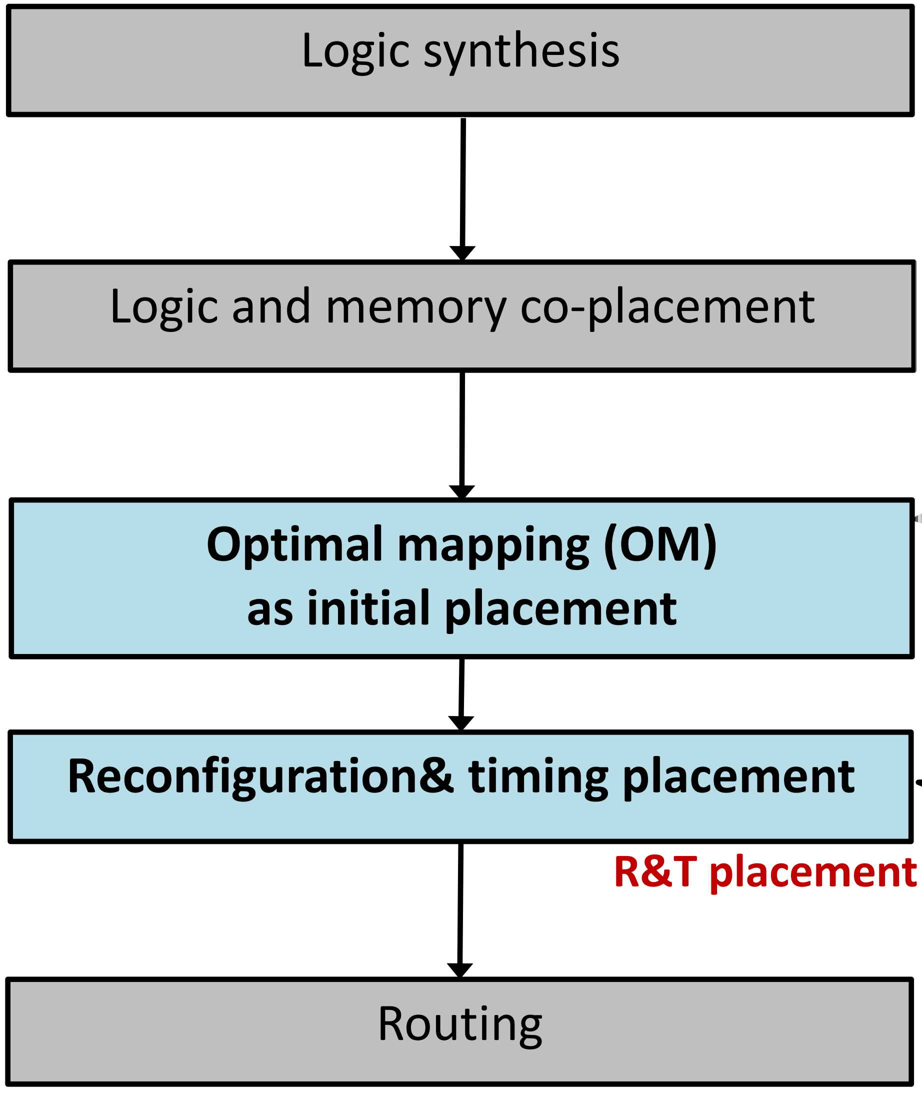
[1] H. Kuhn, “The hungarian method for the assignment problem,” in *Naval Research Logistics Quarterly*, 1955, pp. 29–47.

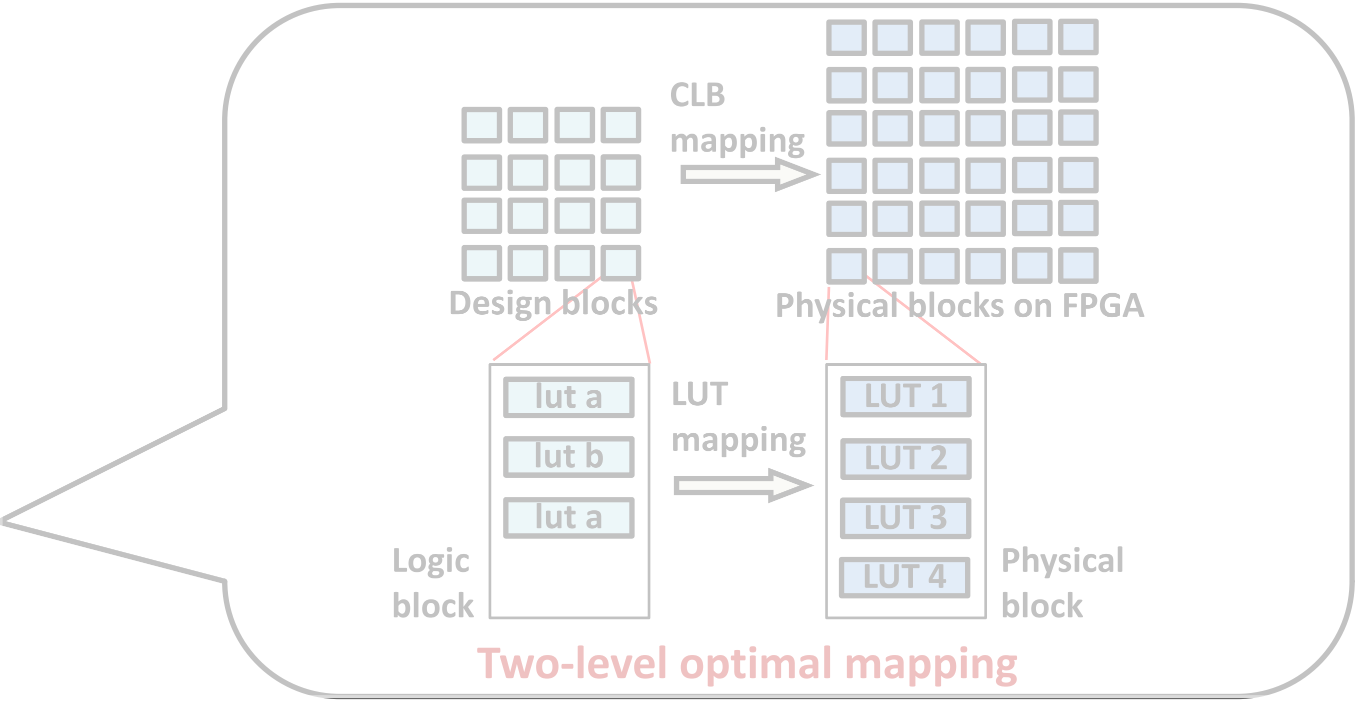
Logic placement



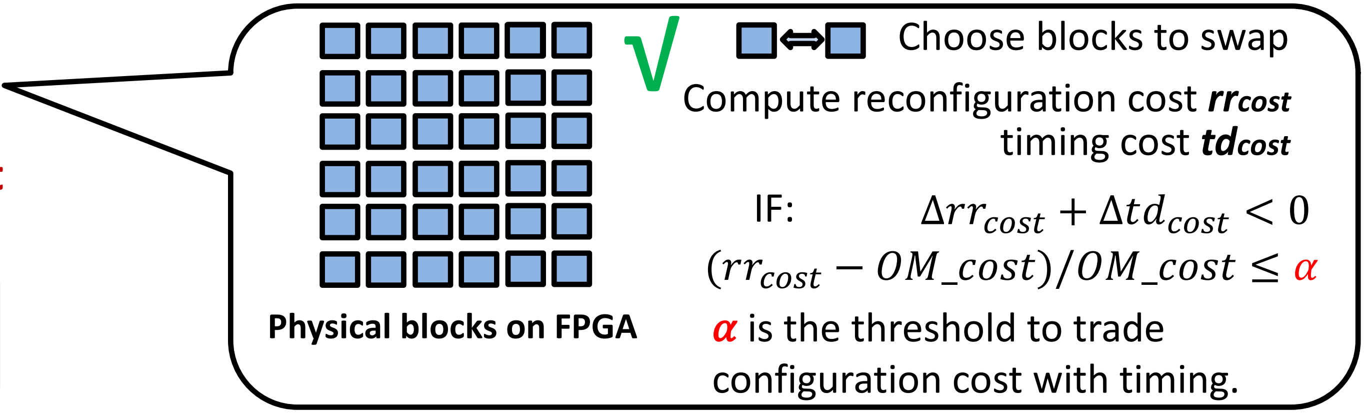


Proposed fine-tuning logic placement





Two-level optimal mapping



Choose blocks to swap

Compute reconfiguration cost ***rrcost***

timing cost ***tdcost***

IF: $\Delta rr_{cost} + \Delta td_{cost} < 0$

$(rr_{cost} - OM\_cost)/OM\_cost \leq \alpha$

α is the threshold to trade configuration cost with timing.

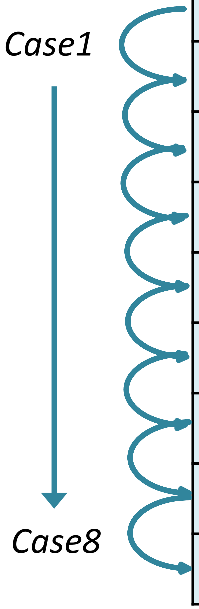
Experimental Setup

- FPGA architecture: Altera Stratix IV
- CAD toolkit: VTR 7.0 [1]

Placement methods

Schemes	Flexibilities exploited		Timing optimization
	LUT swap	CLB swap	
Baseline	-	-	+
Optimal mapping (OM)	+	+	-
R&T1 (α=0.25)	+	+	+
R&T2 (α=0.50)	+	+	+
R&T3 (α=0.75)	+	+	+

9 Microelectronics Center of North Carolina (MCNC) benchmarks [2]



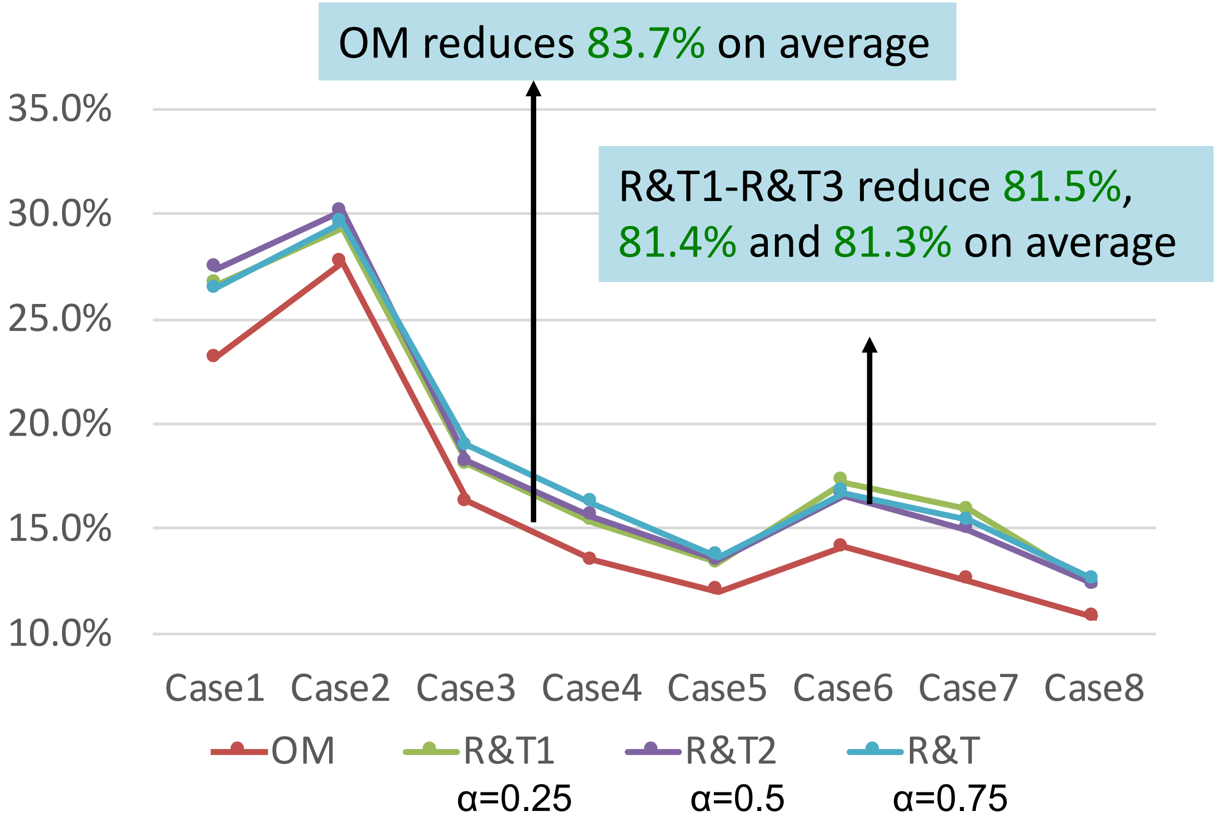
No	Benchmark	LUT#	CLB#
1	tseng	1046	105
2	ex5p	1064	107
3	diffeq	1494	150
4	alu4	1522	153
5	seq	1750	175
6	s298	1930	194
7	elliptic	3602	361
8	spla	3690	369
9	ex1010	4598	460

[1] Jason Luu et al., " VTR 7.0: Next Generation Architecture and CAD System for FPGAs," in *ACM Transactions on Reconfigurable Technology and Systems*, 7(2):1–30, Jun. 2014.

[2] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *International Symposium on Circuits and Systems (ISCAS)*, 1989, pp. 1929–1934.

Normalized reconfiguration bit flips

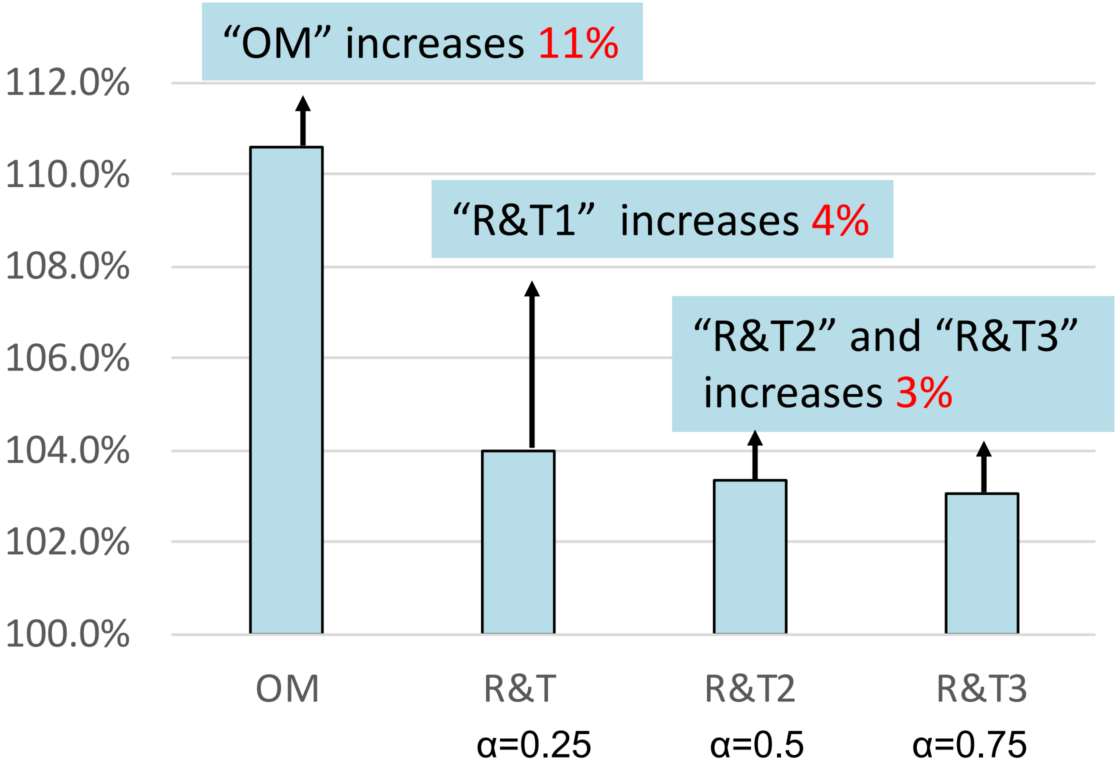
The smaller the better


OM reduces 83.7% on average

R&T1-R&T3 reduce 81.5%, 81.4% and 81.3% on average

Normalized critical path

The smaller the better


"OM" increases 11%

"R&T1" increases 4%

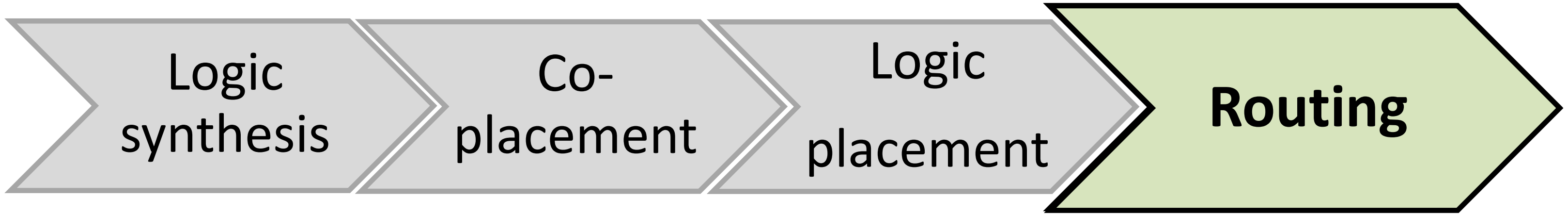
"R&T2" and "R&T3" increases 3%

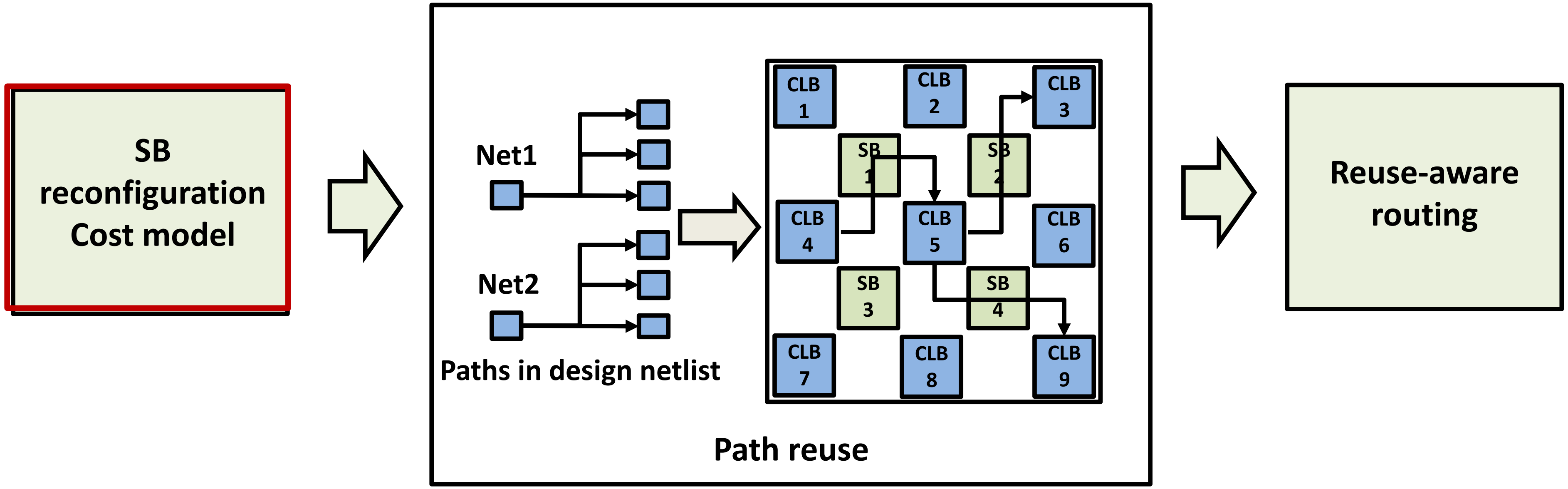
*Overall **α= 0.5** is the best among these configurations.*

Outline

- Introduction
- **Rethink about FPGA synthesis**
  1. Logic synthesis
  2. Logic/memory co-placement
  3. Logic placement
  4. **Routing**
- Conclusions

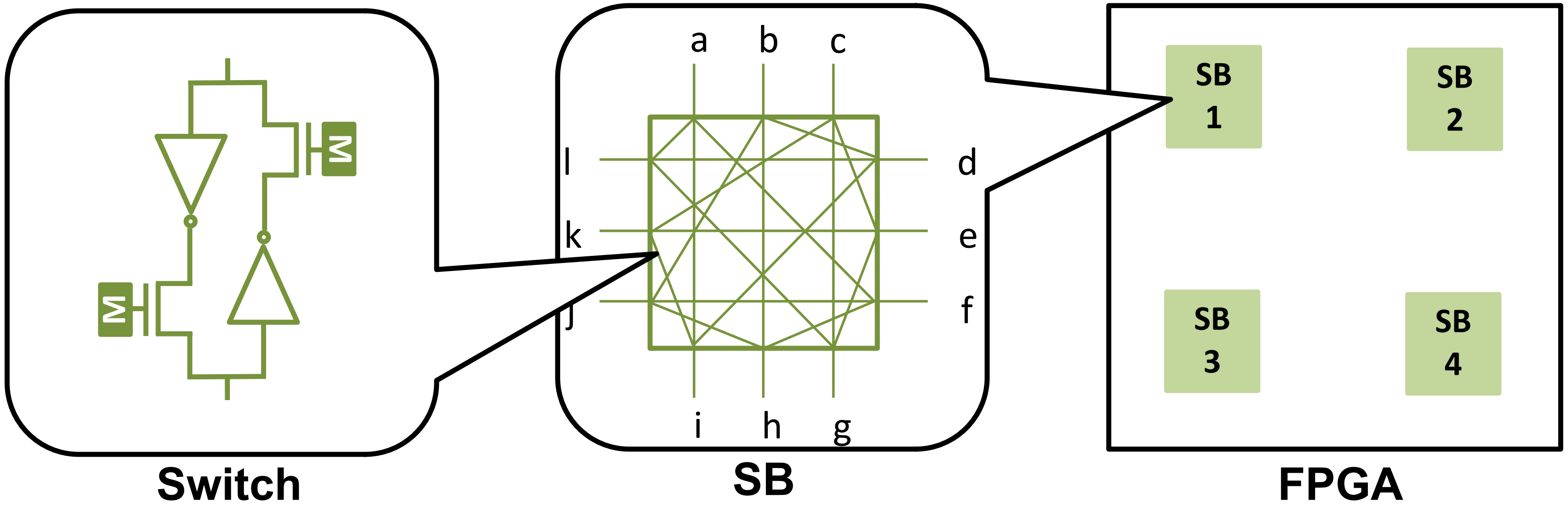
Routing


Logic synthesis
Co-placement
Logic placement
Routing


SB reconfiguration Cost model
Net1
Net2
Paths in design netlist
CLB 1
CLB 2
CLB 3
SB 1
SB 2
CLB 4
CLB 5
CLB 6
SB 3
SB 4
CLB 7
CLB 8
CLB 9
Path reuse
Reuse-aware routing

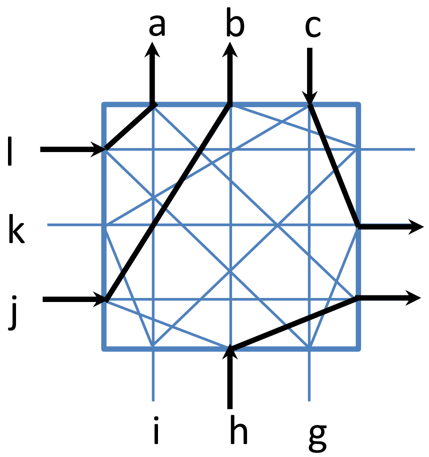
Bit-level SB reconfiguration cost

Bidirectional connection.
Each time only one direction
can be connected.



SB reconfiguration cost is defined by two level costs.

Bit-level SB reconfiguration cost



Existing configuration

switch	On/ off
l→a	1
j→b	1
c→e	1
h→f	1
k→i	0

$N_{old}=4$

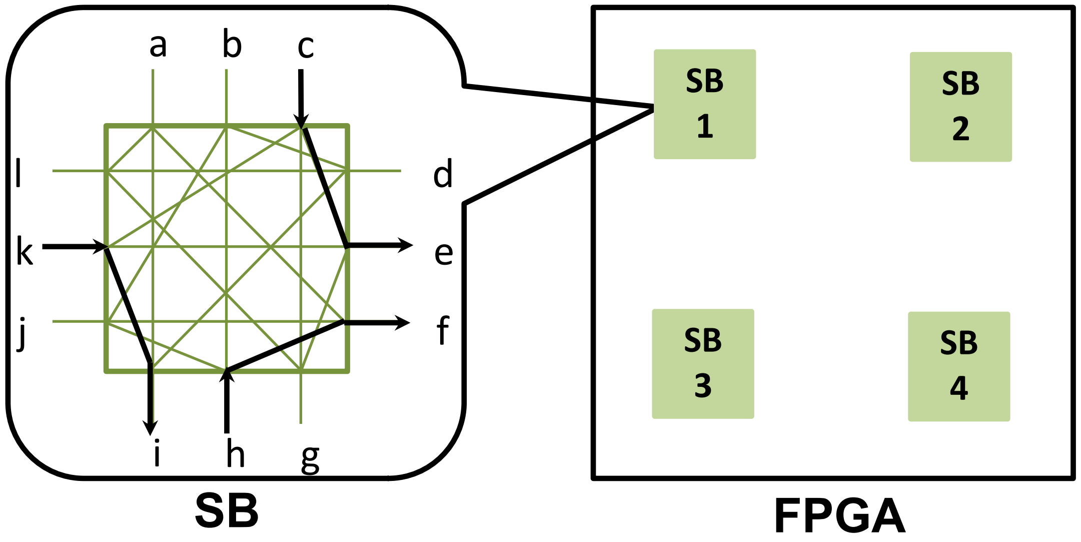
3 changed switches

⇔

Reuse switch: **R=2**

switch	On/ off
l→a	0
j→b	0
c→e	1
h→f	1
k→i	1

$N_{new}=3$



New configuration

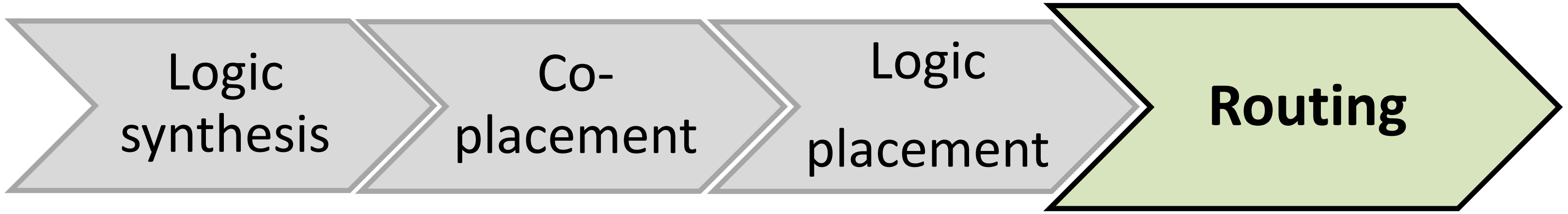
SB reconfiguration cost is defined by two level costs.

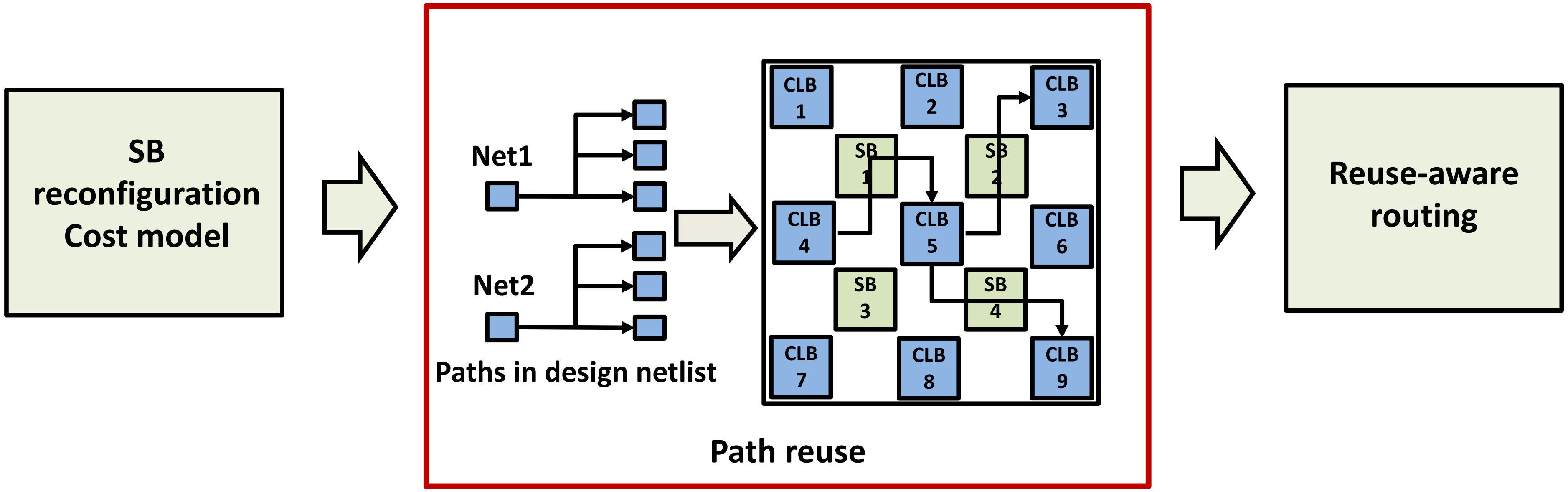
One SB reconfiguration: $\boxed{RR_{SB}=N_{old}+N_{new}-2R}$

Total SB reconfiguration:

$$\boxed{Cost_{reconfig}=\sum_{i=1}^{NUM_{SB}} RR_{SB_i}}$$

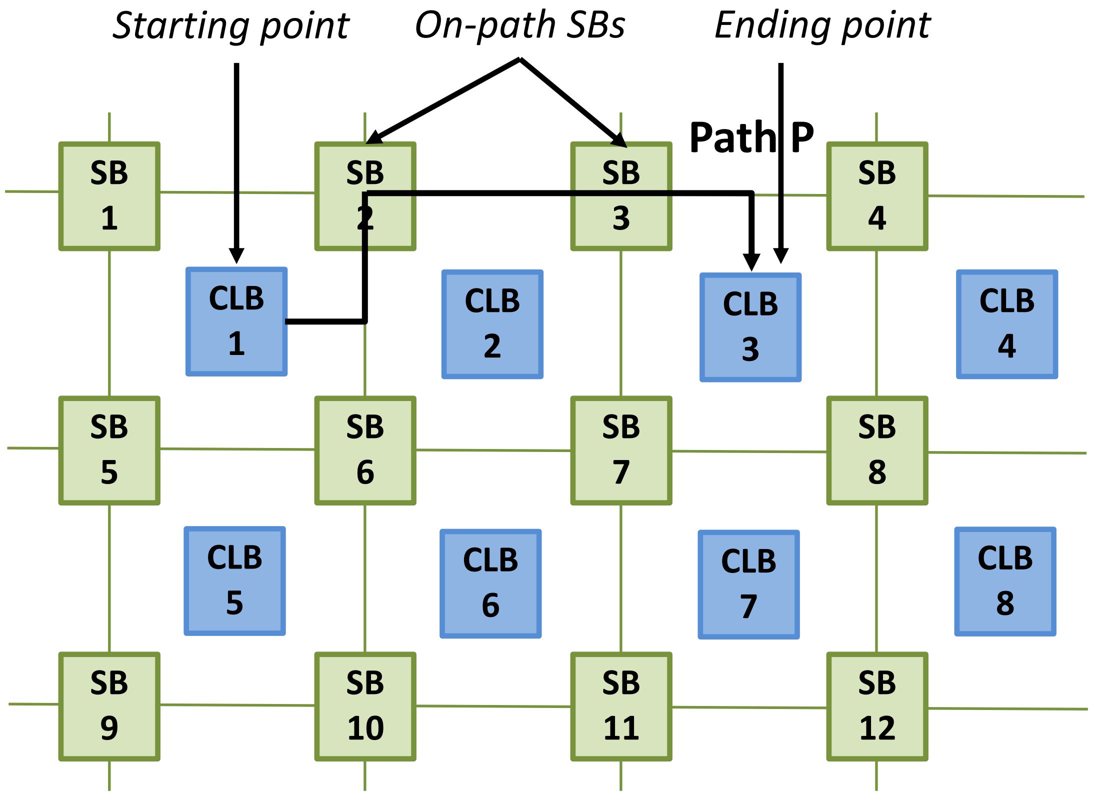
Routing


Logic synthesis
Co-placement
Logic placement
Routing


SB reconfiguration Cost model
Net1
Net2
Paths in design netlist
CLB 1
CLB 2
CLB 3
SB 1
SB 2
CLB 4
CLB 5
CLB 6
SB 3
SB 4
CLB 7
CLB 8
CLB 9
Path reuse
Reuse-aware routing

Path definition and characterization

Characterization: Path P={(CLB$_i$,CLB$_j$),(SB$_{first}$...SB$_{last}$)}.



FPGA

If path P can be reused,

⇩

Switches in SB2 and SB3 can be reused.

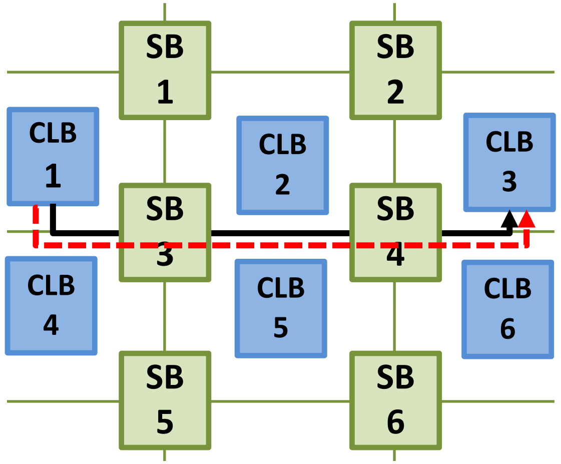
⇩

SB reuse is equivalent to path reuse!

Two types of reusable paths

Full Reuse

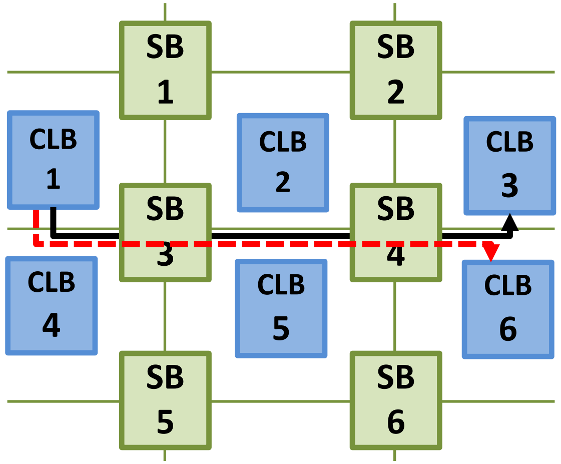
***All SBs** on old path can be reused.*



*Starting and ending points are the **same**.*

Partial Reuse

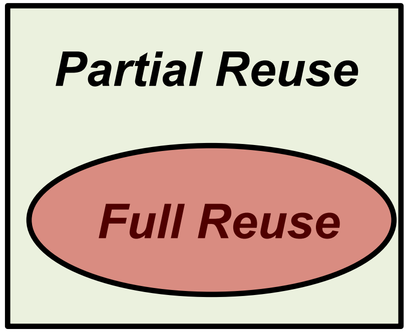
***Some SBs** on old path can be reused.*



*Starting/ending point is **not the same**.*

- - -→ ***New path***

→ ***Old path***

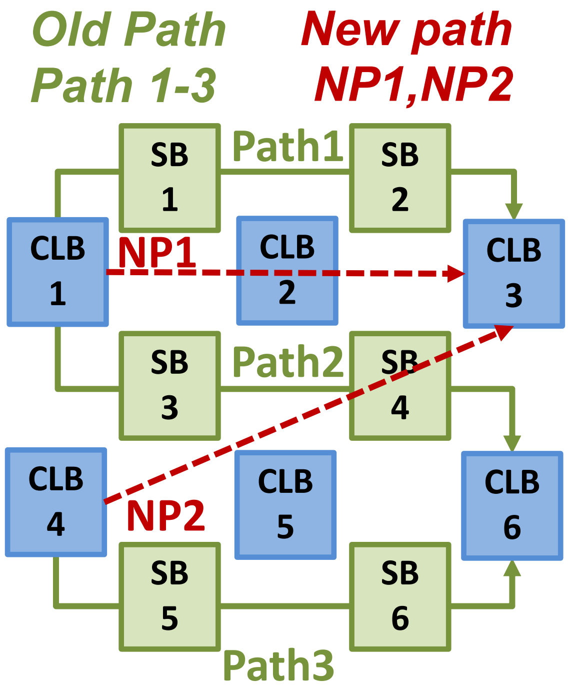


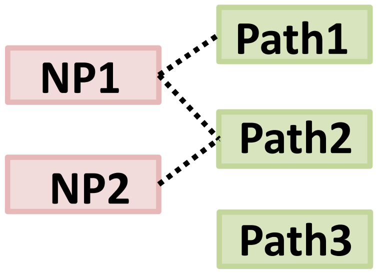
Partial Reuse

Full Reuse

Full reuse is a subset of partial reuse.

Reusable path recognition





1. Find all possible **partial Reuse**

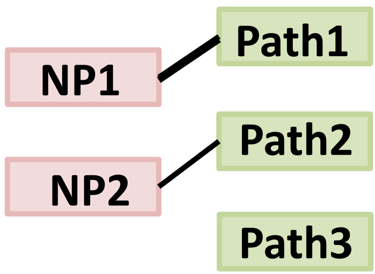
Construct path mapping

$$\begin{array}{c|ccc} & \multicolumn{3}{c}{\text{Path 1-3}} \\ \text{NP 1-2} & \boxed{1} & 1 & 0 \\ & 0 & \boxed{1} & 0 \end{array}$$

2. **Maximize** partial reuse

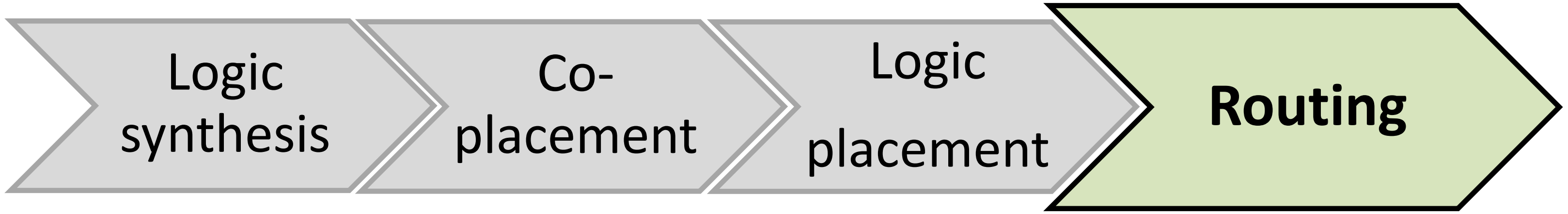
Use **KM algorithm** for reuse maximization

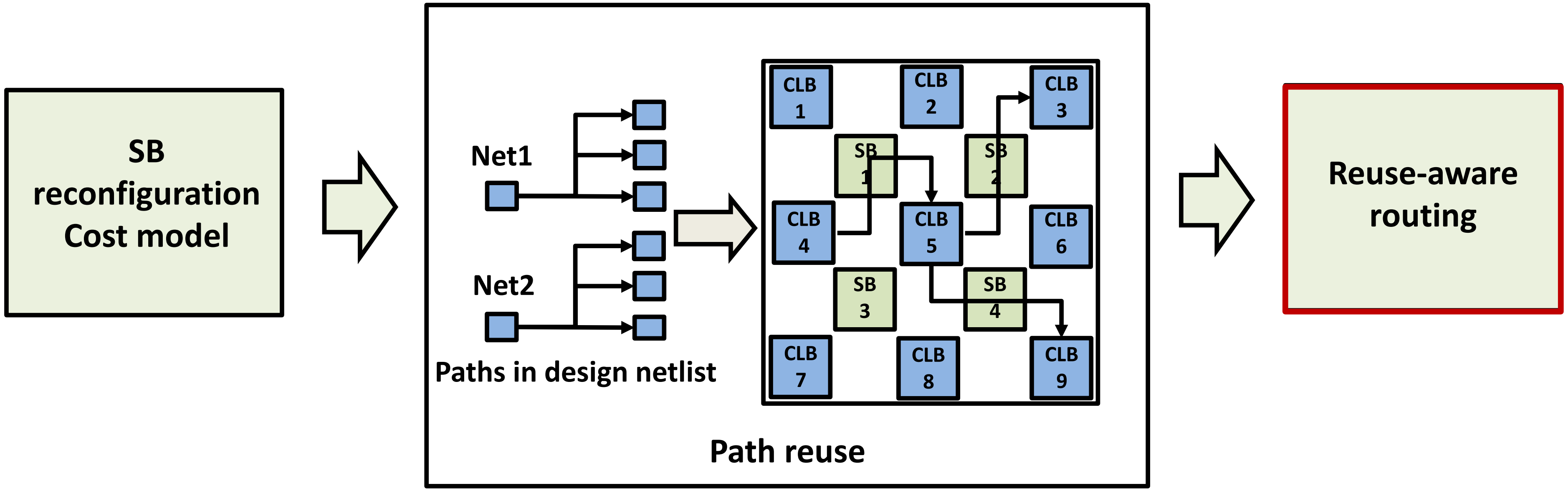
full reuse



3. Upgrade to **full reuse** as much as possible

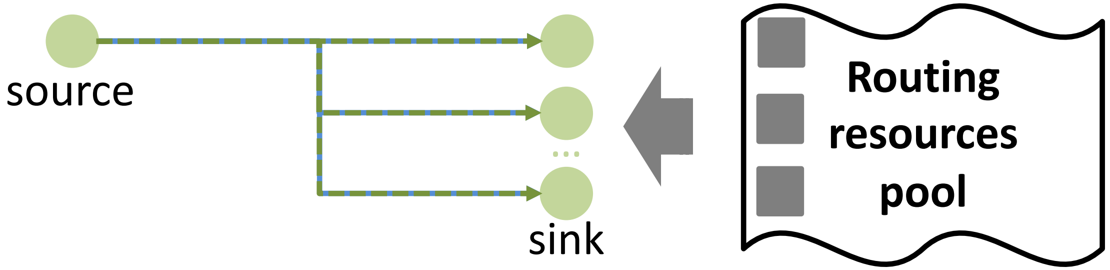
Routing


Logic synthesis
Co-placement
Logic placement
Routing


SB reconfiguration Cost model
Net1
Net2
Paths in design netlist
CLB 1
CLB 2
CLB 3
SB 1
SB 2
CLB 4
CLB 5
CLB 6
SB 3
SB 4
CLB 7
CLB 8
CLB 9
Path reuse
Reuse-aware routing

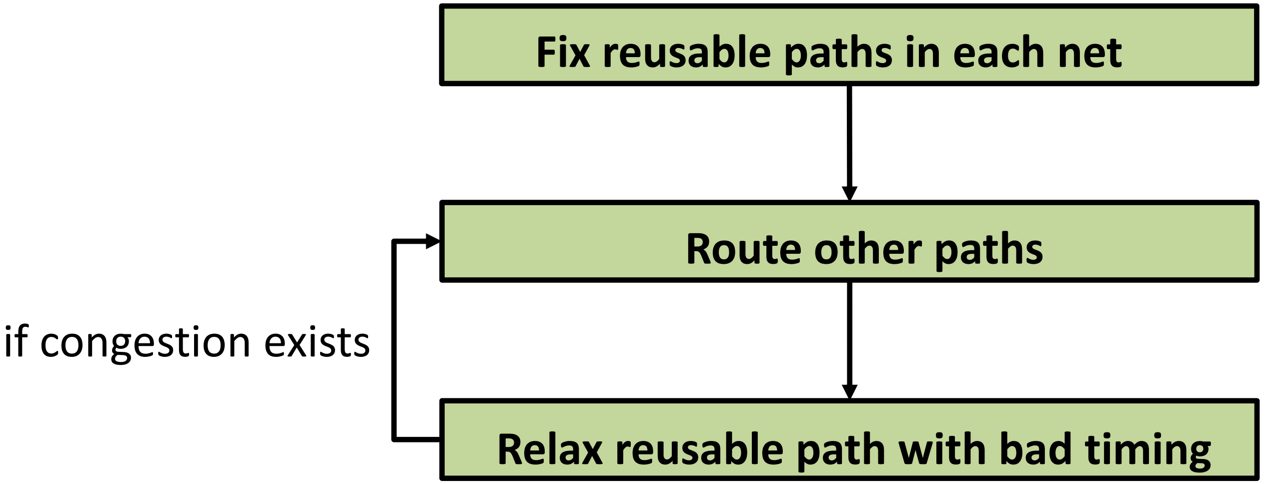
Reuse-aware routing algorithm

Basic routing: Select routing resources to finish connection in each net.



Proposed routing: reuse reusable paths and route other paths

Three-stage Algorithm


Fix reusable paths in each net
Route other paths
Relax reusable path with bad timing
if congestion exists

ε: threshold of relaxation
ε=1% means relax paths whose delay is < 1% shorter than **critical path**.

Experimental Setup

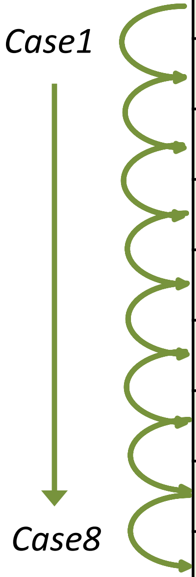
- ❖ FPGA architecture: Altera Stratix IV
- ❖ CAD toolkit: VTR 7.0 [1]

Routing methods

Schemes	Reusable path identification	Reuse maximization	Reuse-aware routing
Baseline	-	-	-
DIR	+	-	+
Proposed	+	+	+

DIR: Proposed scheme without reuse maximization

9 Microelectronics Center of North Carolina (MCNC) benchmarks [2]

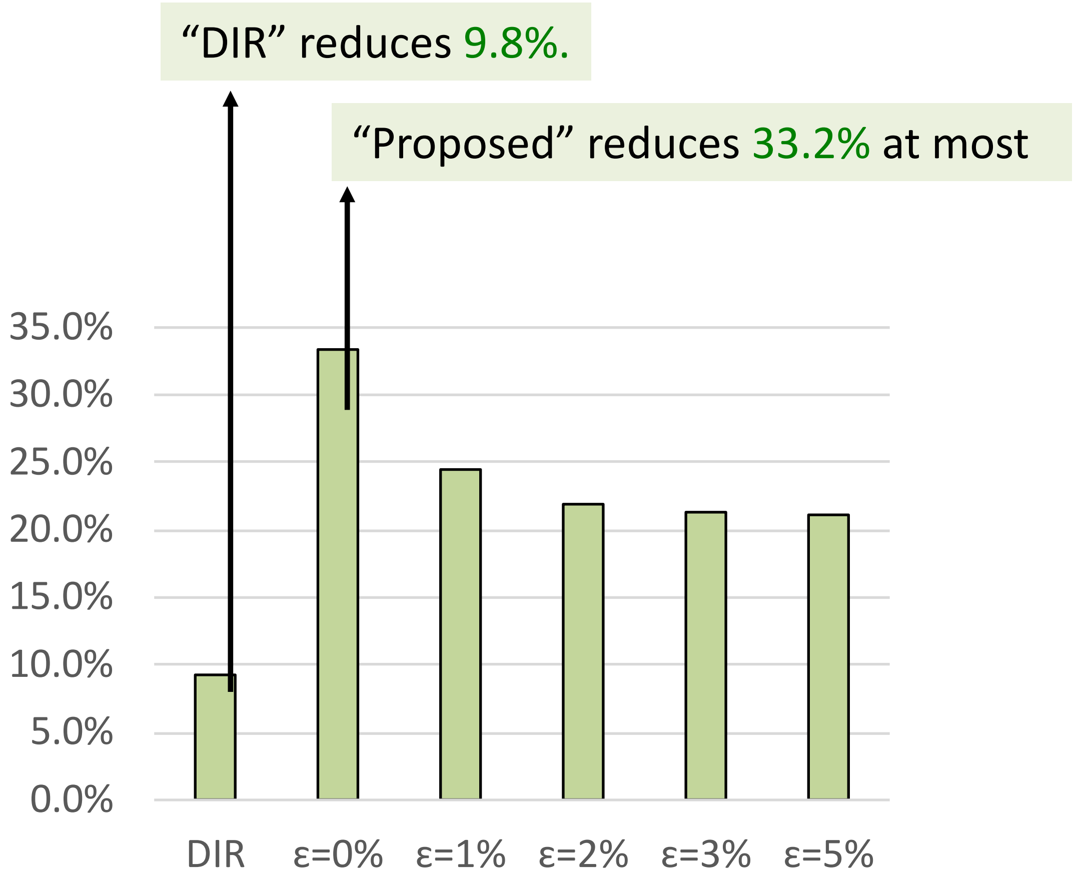


No	Benchmark	LUT#	CLB#
1	tseng	1046	105
2	ex5p	1064	107
3	diffeq	1494	150
4	alu4	1522	153
5	seq	1750	175
6	s298	1930	194
7	elliptic	3602	361
8	spla	3690	369
9	ex1010	4598	460

[1] Jason Luu et al., " VTR 7.0: Next Generation Architecture and CAD System for FPGAs," in *ACM Transactions on Reconfigurable Technology and Systems*, 7(2):1–30, Jun. 2014.

[2] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *International Symposium on Circuits and Systems (ISCAS)*, 1989, pp. 1929–1934.

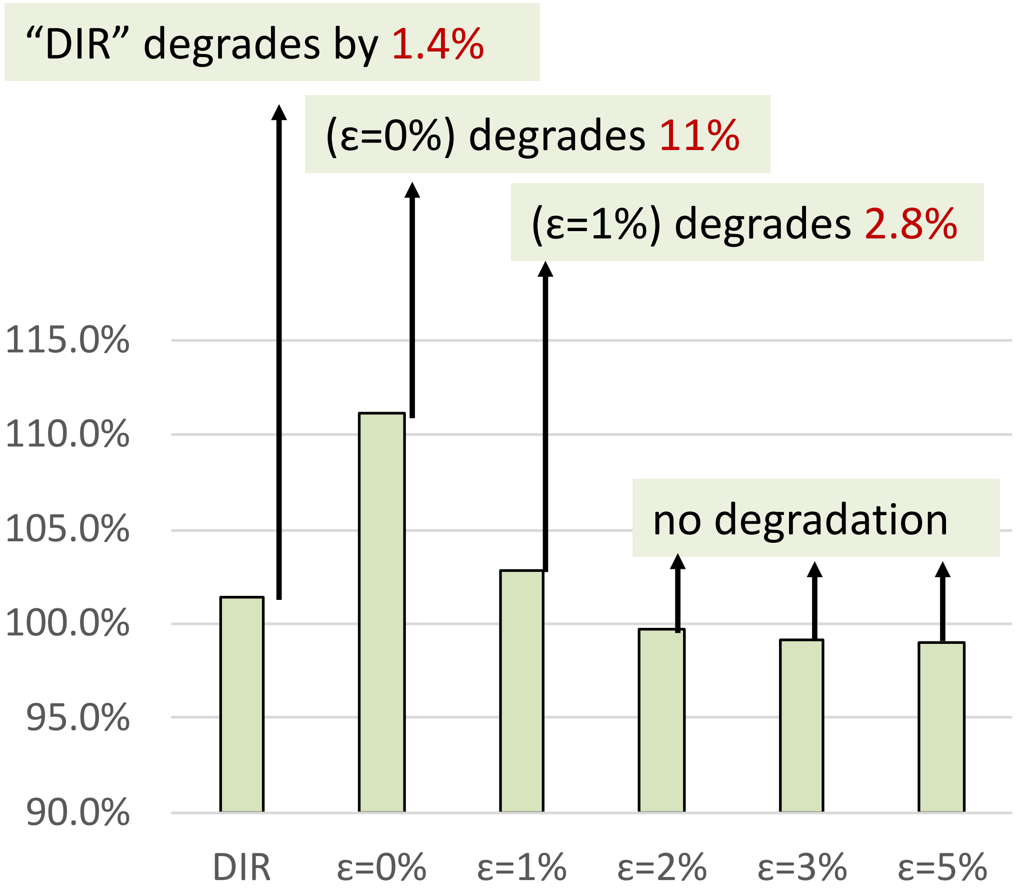
SB reconfiguration cost reduction



“DIR” reduces 9.8%.

“Proposed” reduces 33.2% at most

Normalized critical path delay



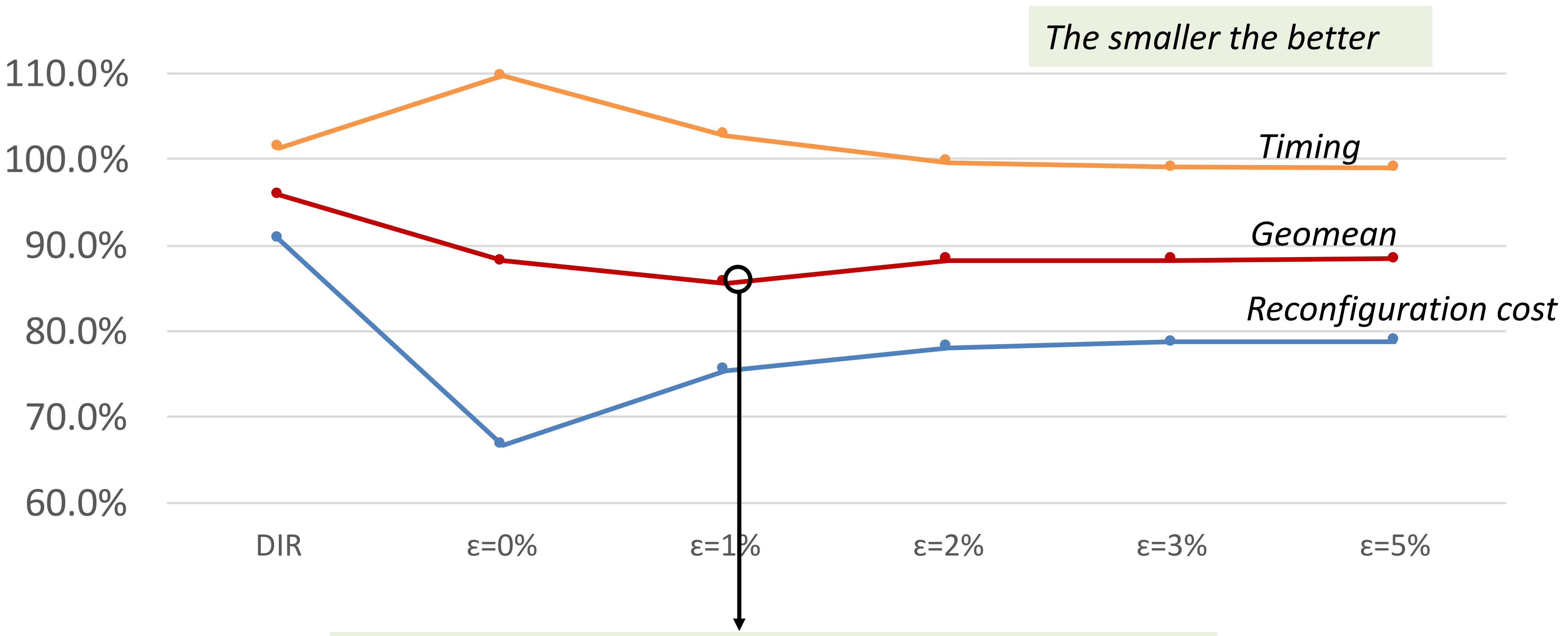
“DIR” degrades by 1.4%

(ε=0%) degrades 11%

(ε=1%) degrades 2.8%

no degradation

Select the best ε



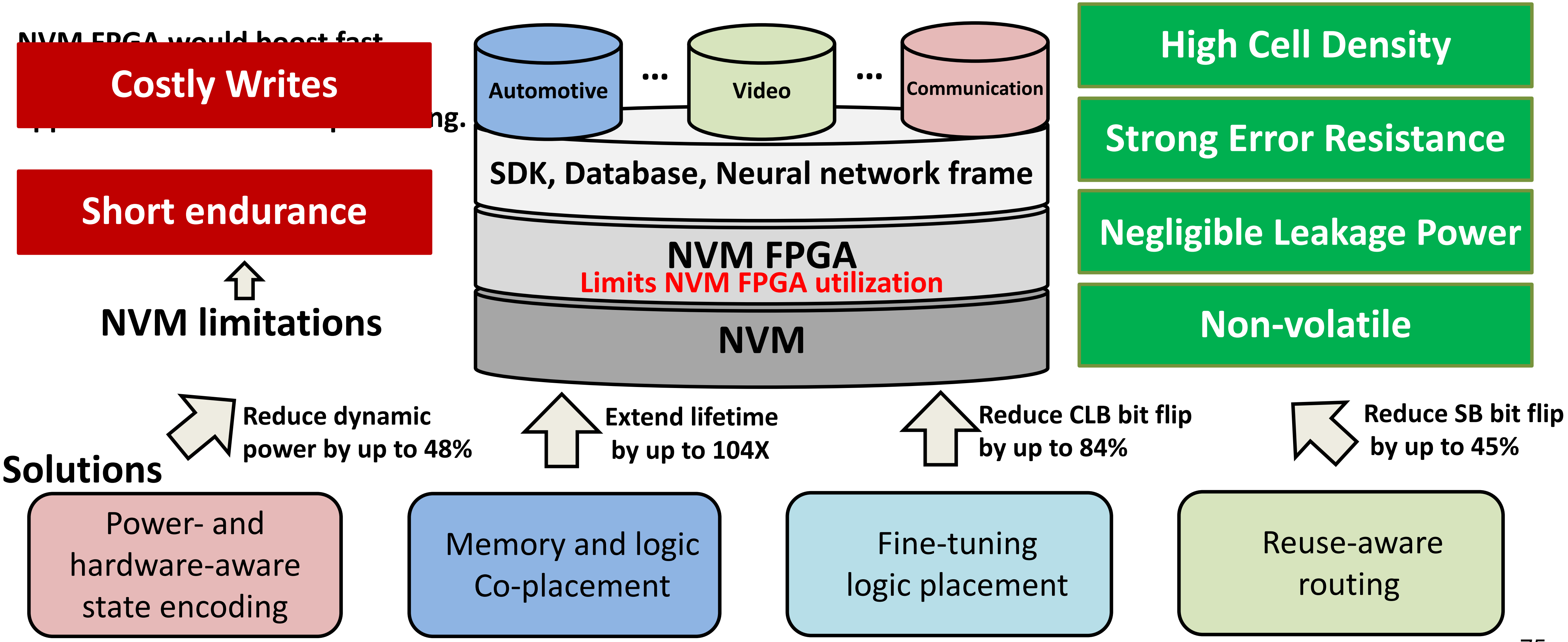
The **geometric means** show that **ε=1%** is the best.

Outline

- Introduction
- Rethink about FPGA synthesis
  1. Logic synthesis
  2. Logic/memory co-placement
  3. Logic placement
  4. Routing
- **Conclusions**

NVM friendly synthesis flow

NVM friendly synthesis flow would increase the popularity and practicality of NVM FPGAs.



NVM FPGA would boost fast

ng.

Costly Writes

Short endurance

NVM limitations

Automotive

…

Video

…

Communication

SDK, Database, Neural network frame

NVM FPGA

Limits NVM FPGA utilization

NVM

High Cell Density

Strong Error Resistance

Negligible Leakage Power

Non-volatile

Reduce dynamic power by up to 48%

Extend lifetime by up to 104X

Reduce CLB bit flip by up to 84%

Reduce SB bit flip by up to 45%

Solutions

Power- and hardware-aware state encoding

Memory and logic Co-placement

Fine-tuning logic placement

Reuse-aware routing

THE END

Questions are welcome.

chengmo@udel.edu