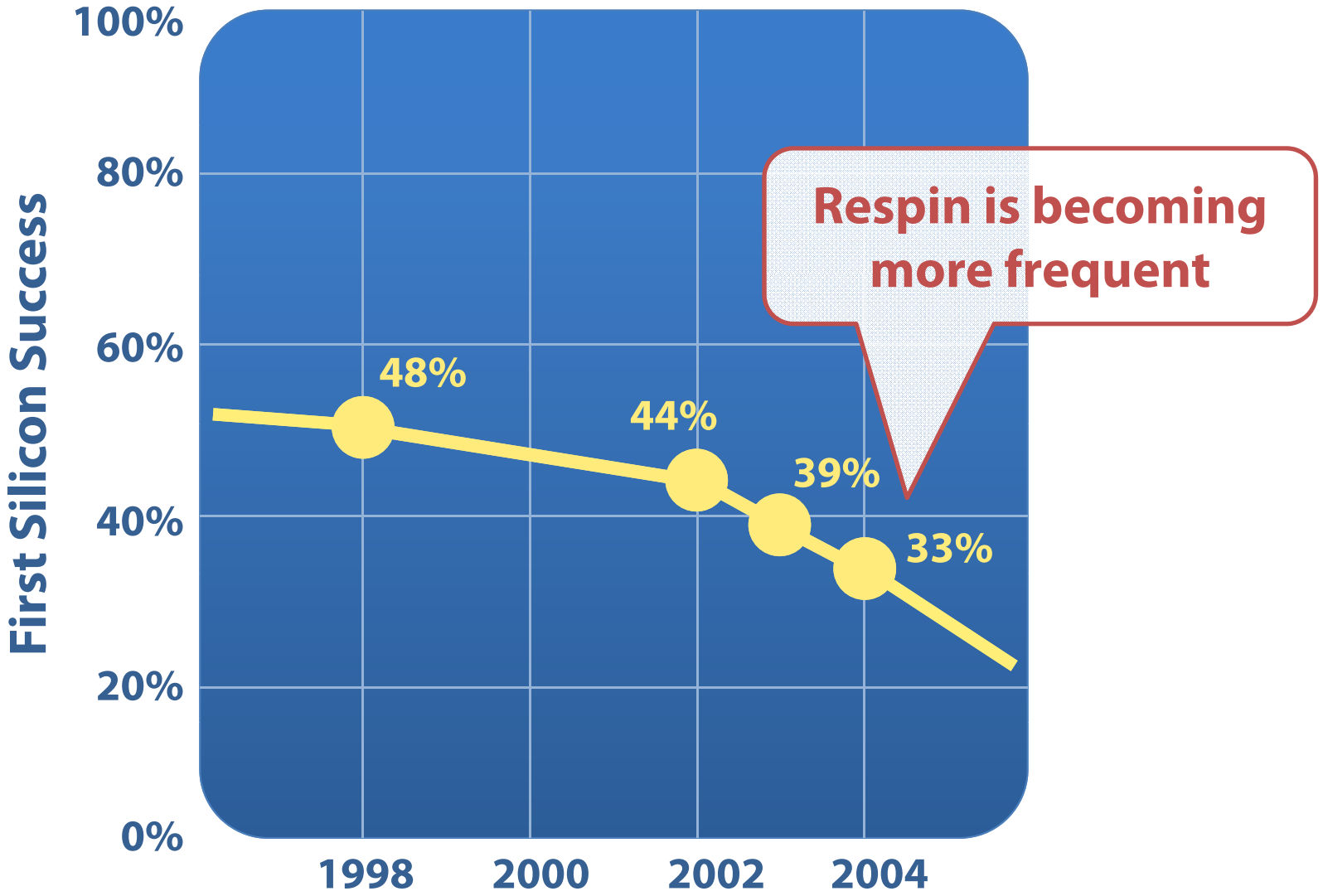
Post-Silicon Patchable Hardware

Masahiro Fujita

VLSI Design and Education Center (VDEC)
The University of Tokyo

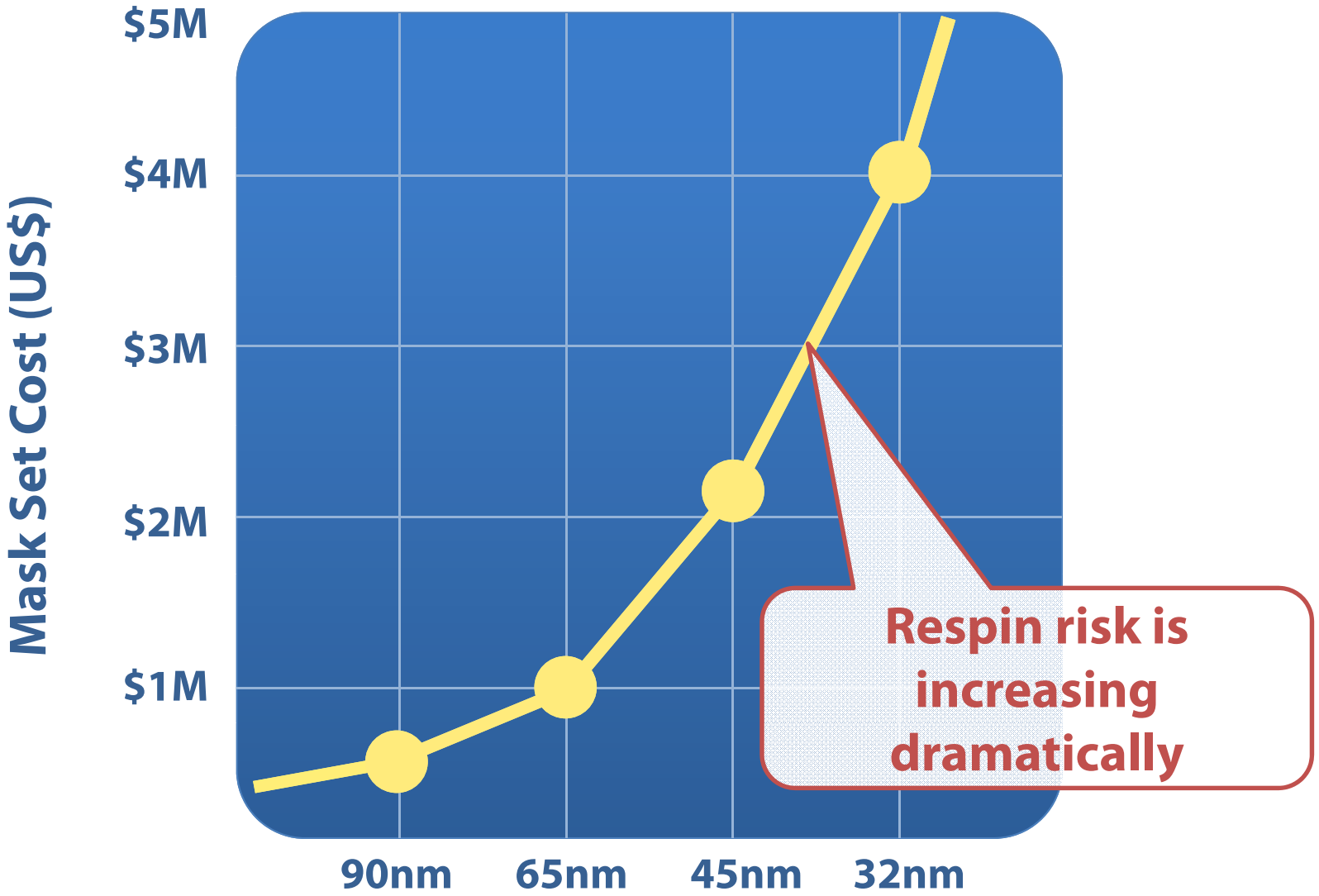
July 22nd, 2011

Respin Statistics (North America)



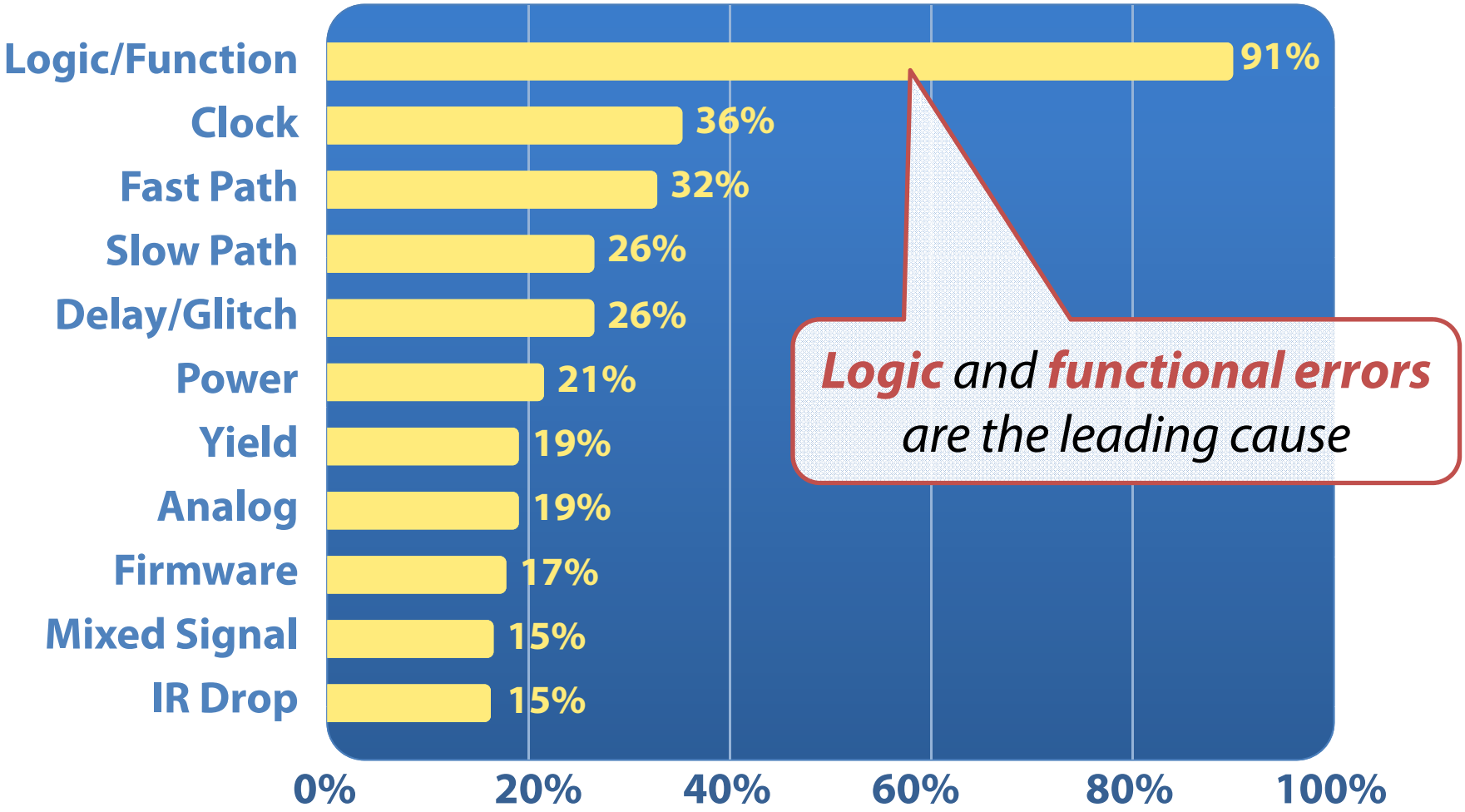
[G. S. Spirakis, DATE 2006]

Manufacturing Cost



[Nikkei Electronics, 2008]

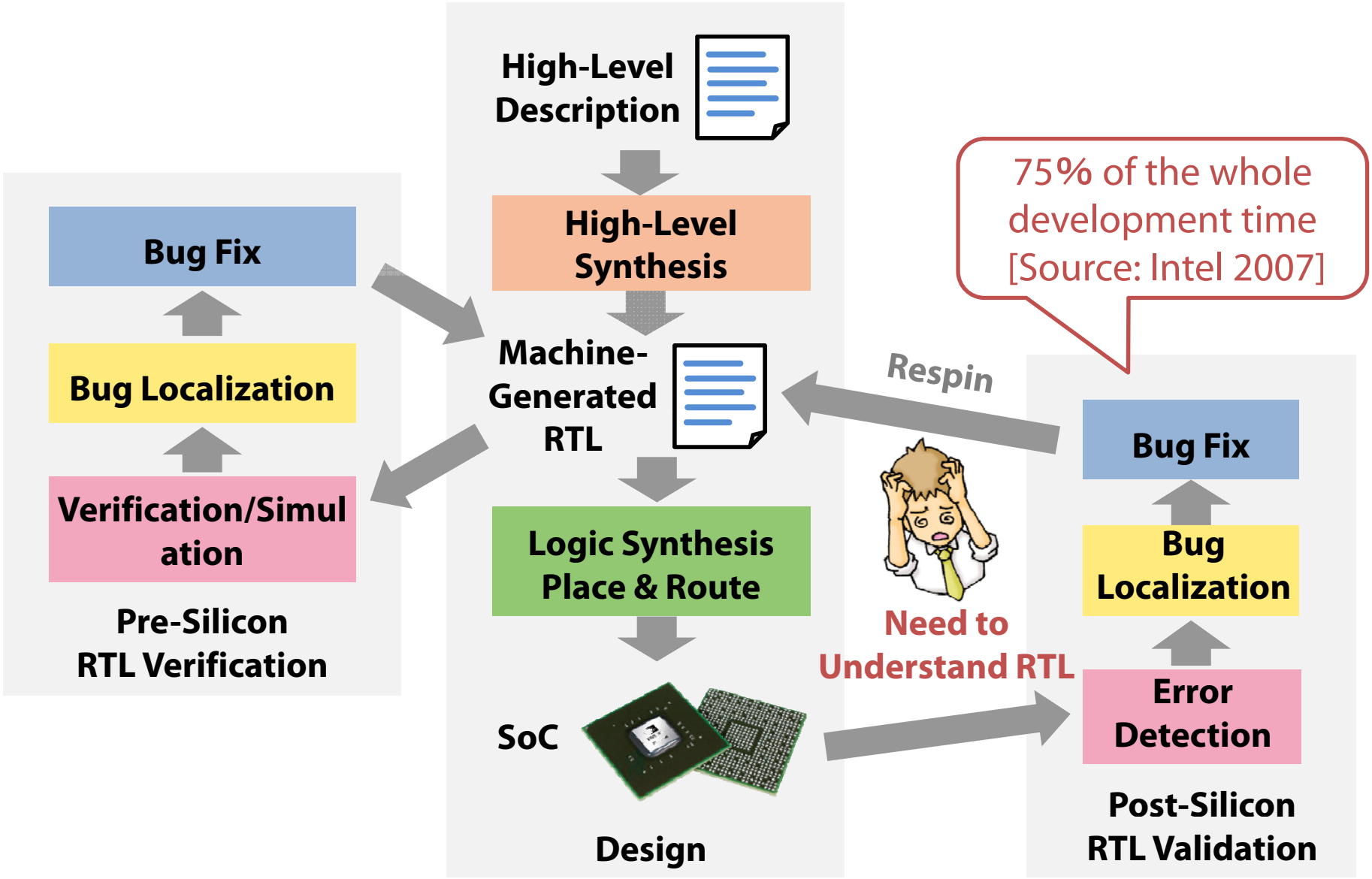
Causes for Respins



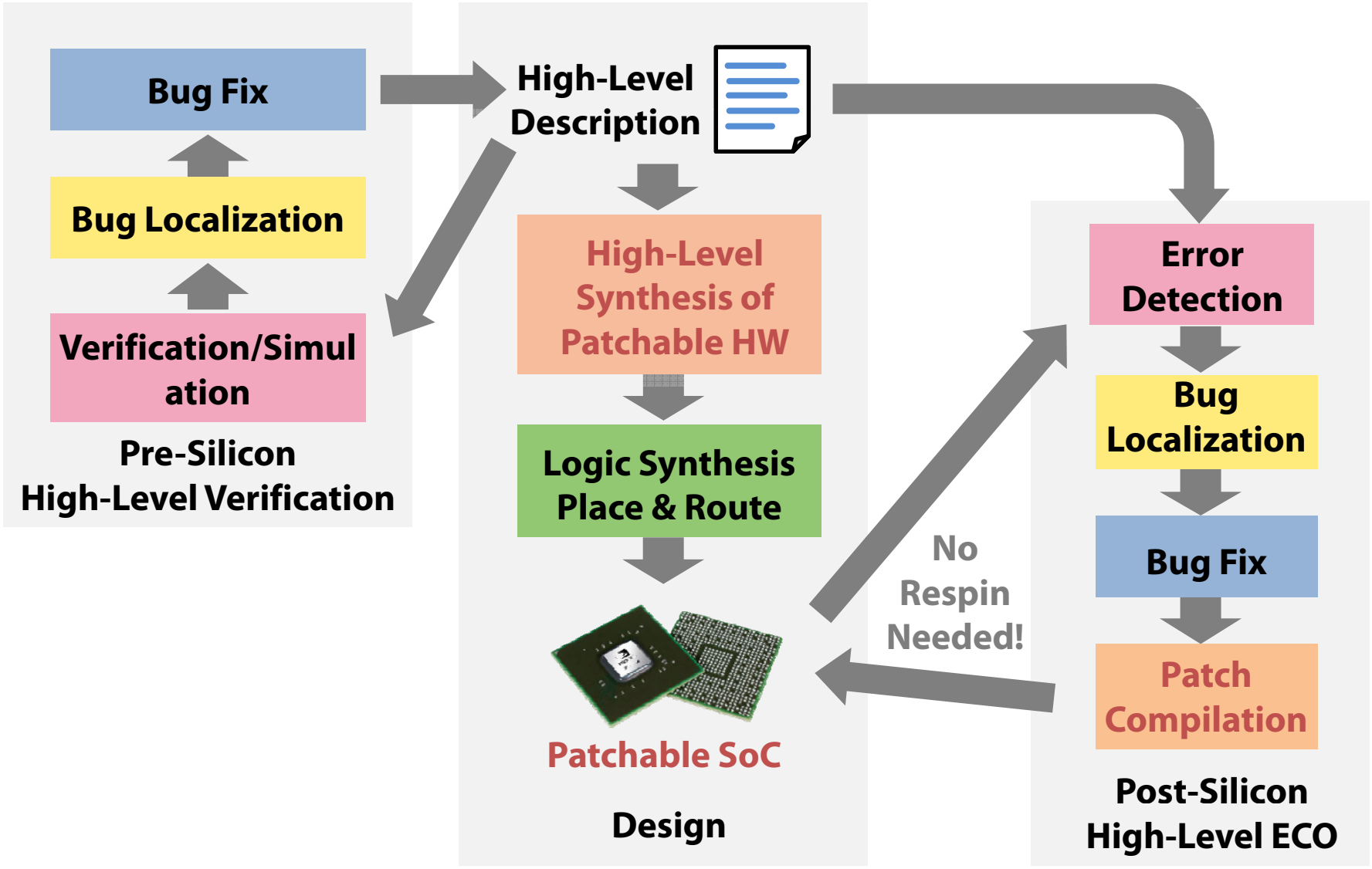
IC/ASIC Designs Having One or More Re-spins by Type of Flaw

[Collett International Research 2005]

Conventional SoC Design Flow

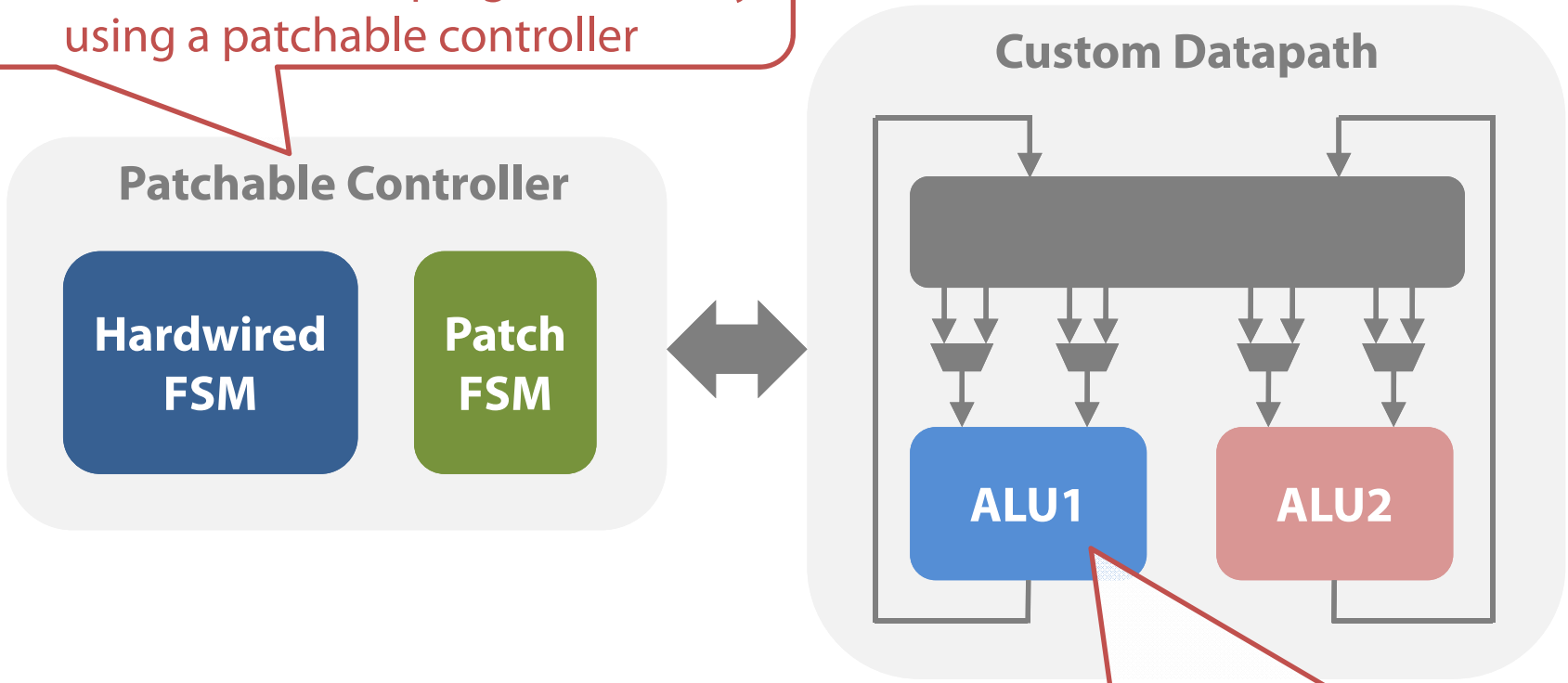


Proposed Patchable SoC Design Flow



Proposed Patchable Hardware

Efficeum
offers behavioral-level programmability
using a patchable controller

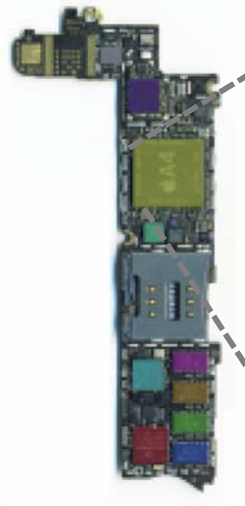


Partially-Programmable Circuit (PPC)
offers logic-level programmability
using a mixed gate/LUT circuit

Efficeum:

An Energy-Efficient Patchable Accelerator For Post-Silicon Engineering Changes

Energy Efficiency vs. Programmability



Energy Efficiency of 90nm OFDM

Fixed-function HW: 200GOPS/W

Embedded Proc.: 4GOPS/W **50X!**

Laptop Proc.: 0.05GOPS/W **4,000X!**

>100GOPS High Performance

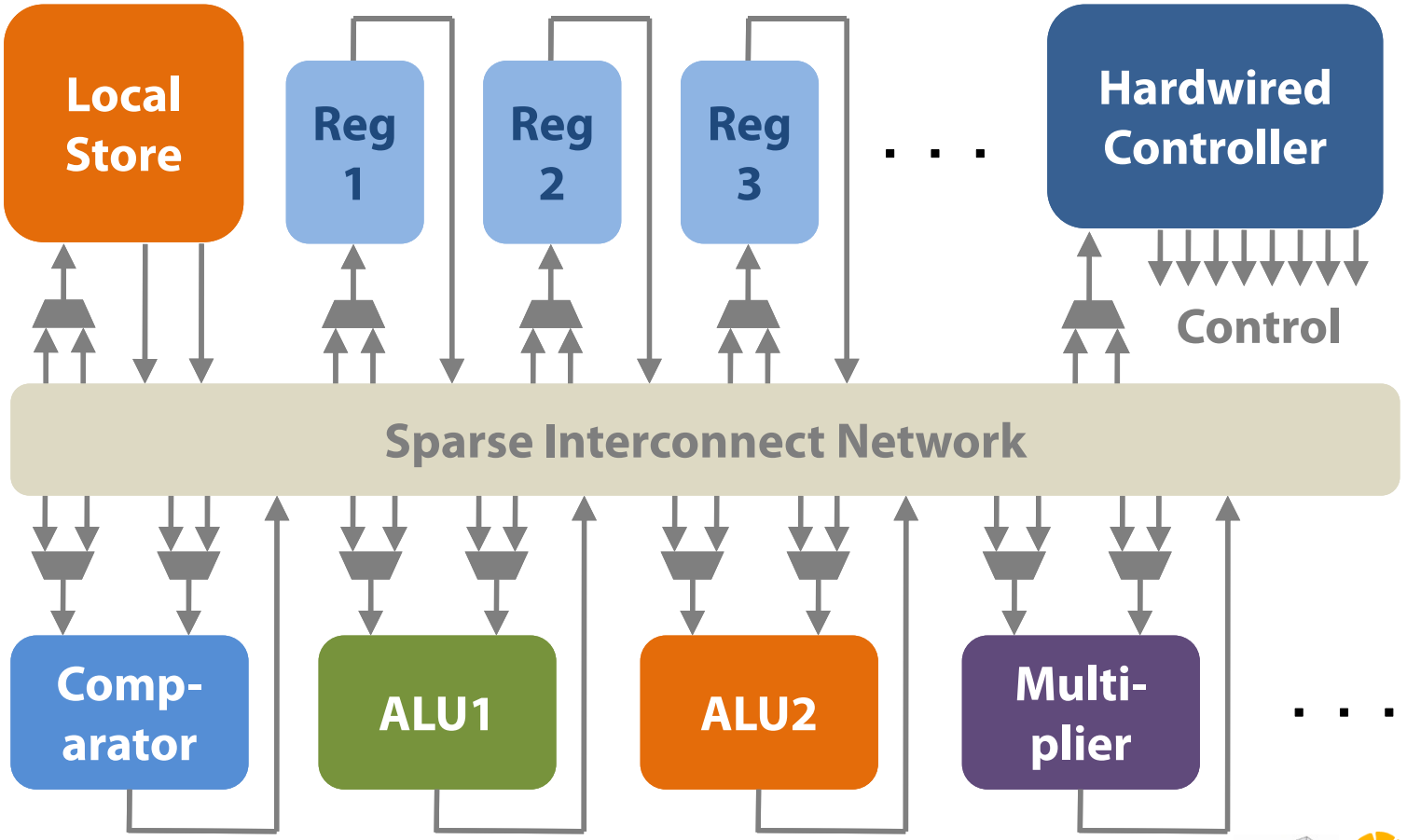
～1W Power/Thermal Constraints

Energy efficiency (in [GOPS/W] or [J/op])

- How much computation can be done in a given energy
- Slowing down the chip reduces power but not efficiency

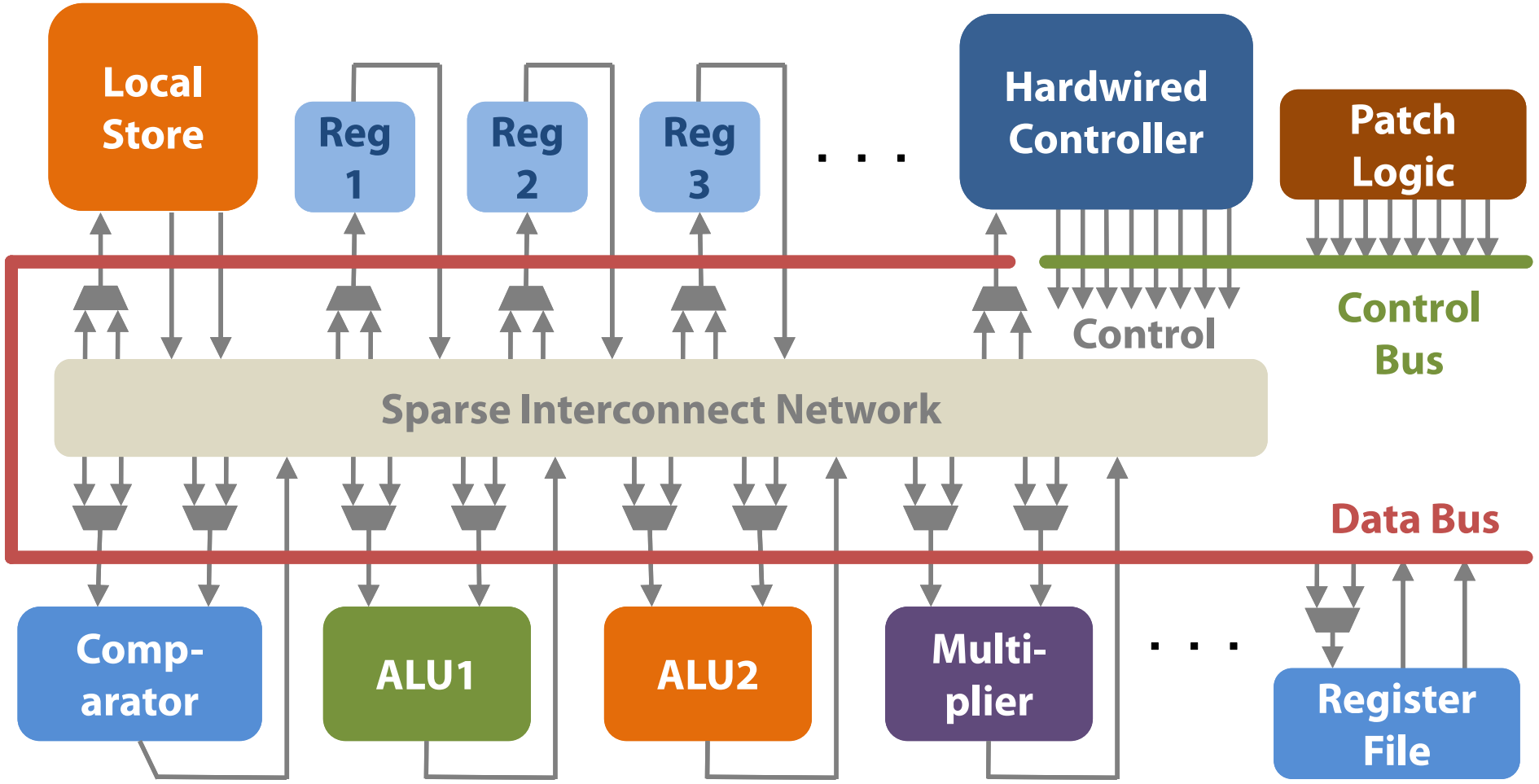
Fixed-Function Accelerator

- Achieves high energy efficiency by customization:
  - Hardwired controller → **No reprogrammability**
  - Highly-customized datapath → **Low flexibility**

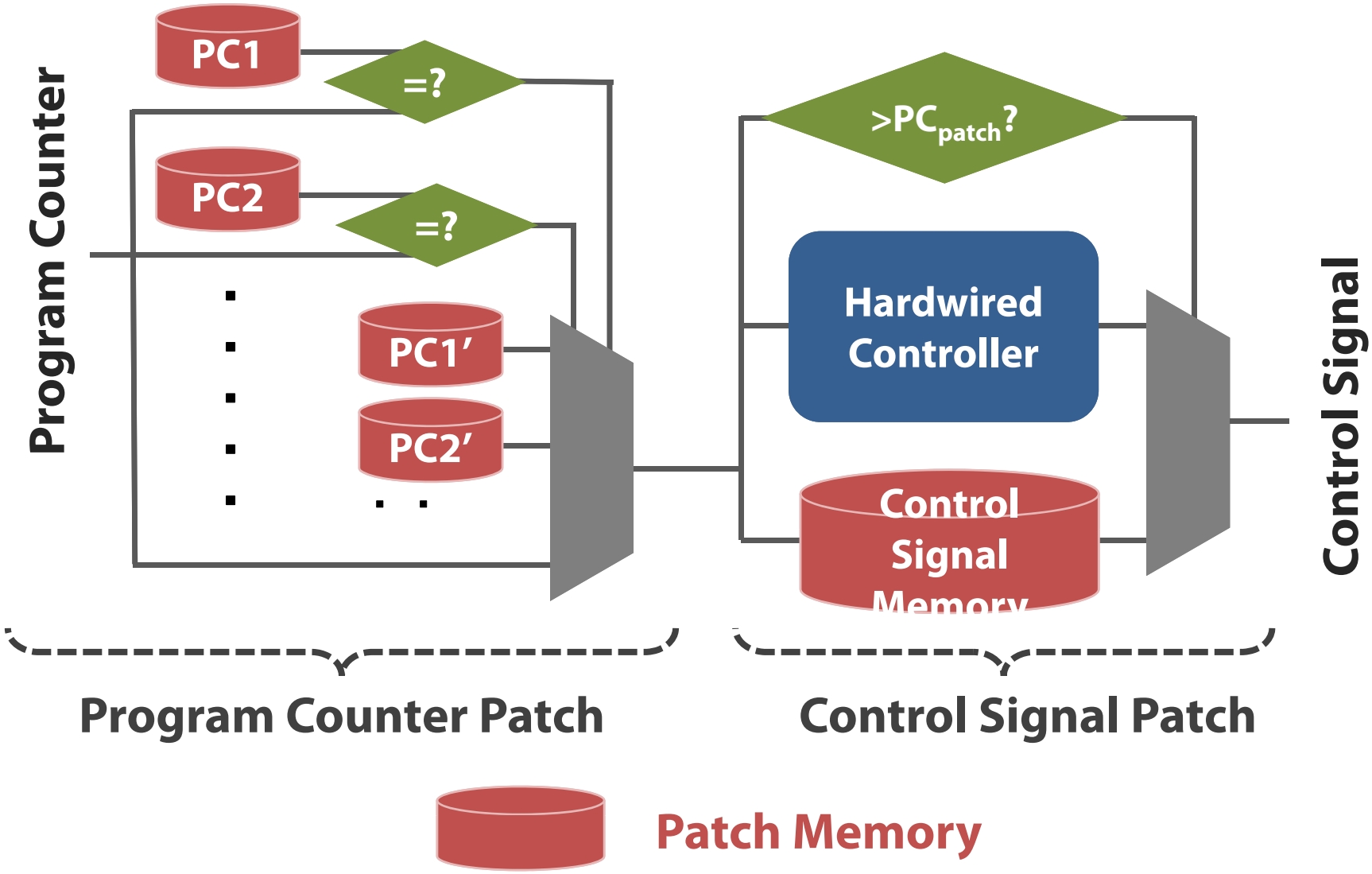


Proposed Patchable Accelerator

- Behavioral reprogrammability by control patching
- Increased flexibility by adding register file via data bus

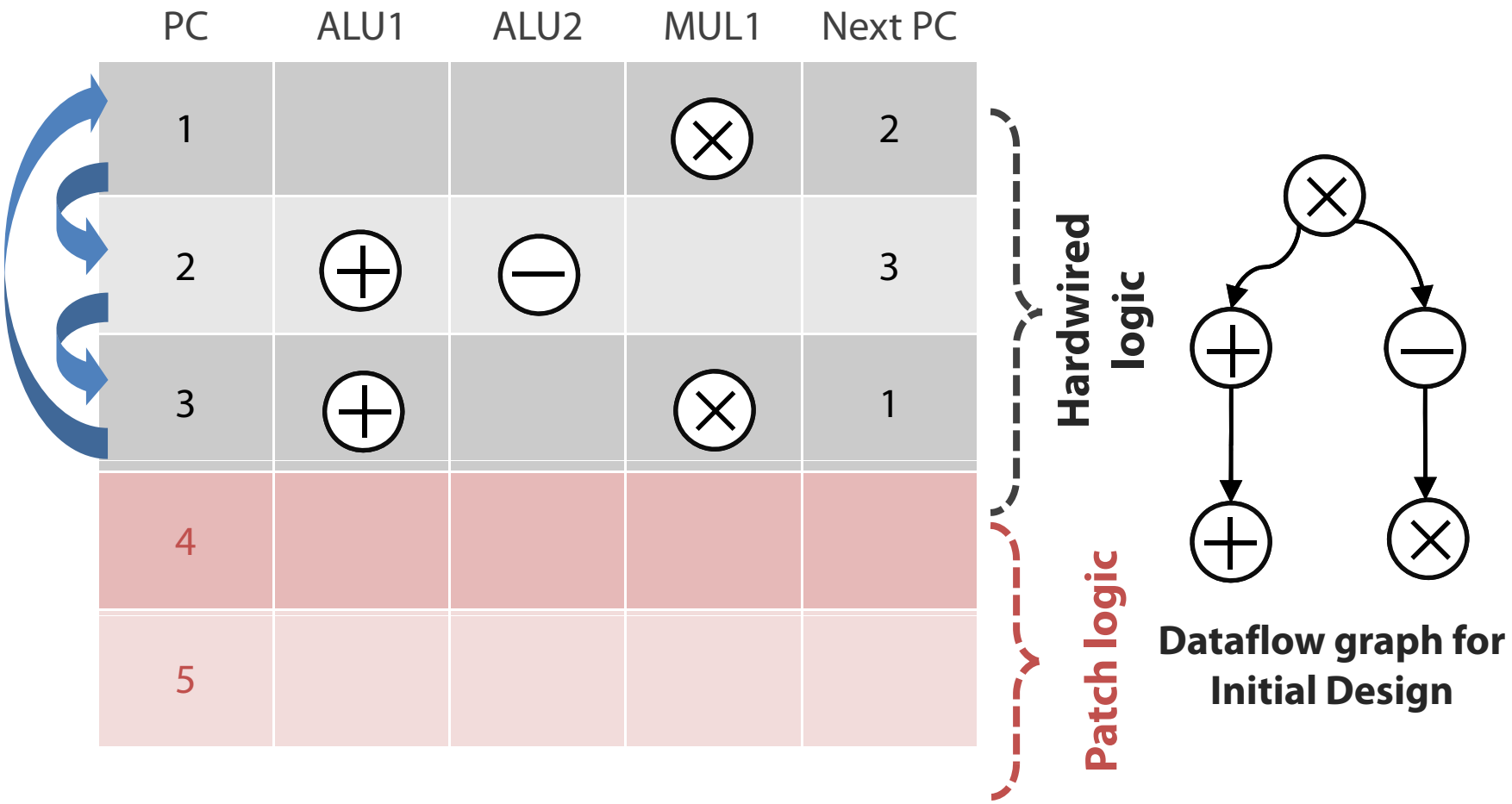


Patch Logic


Program Counter
PC1
=?
PC2
=?
PC1'
PC2'
>PC_patch?
Hardwired Controller
Control Signal Memory
Control Signal
Program Counter Patch
Control Signal Patch
Patch Memory

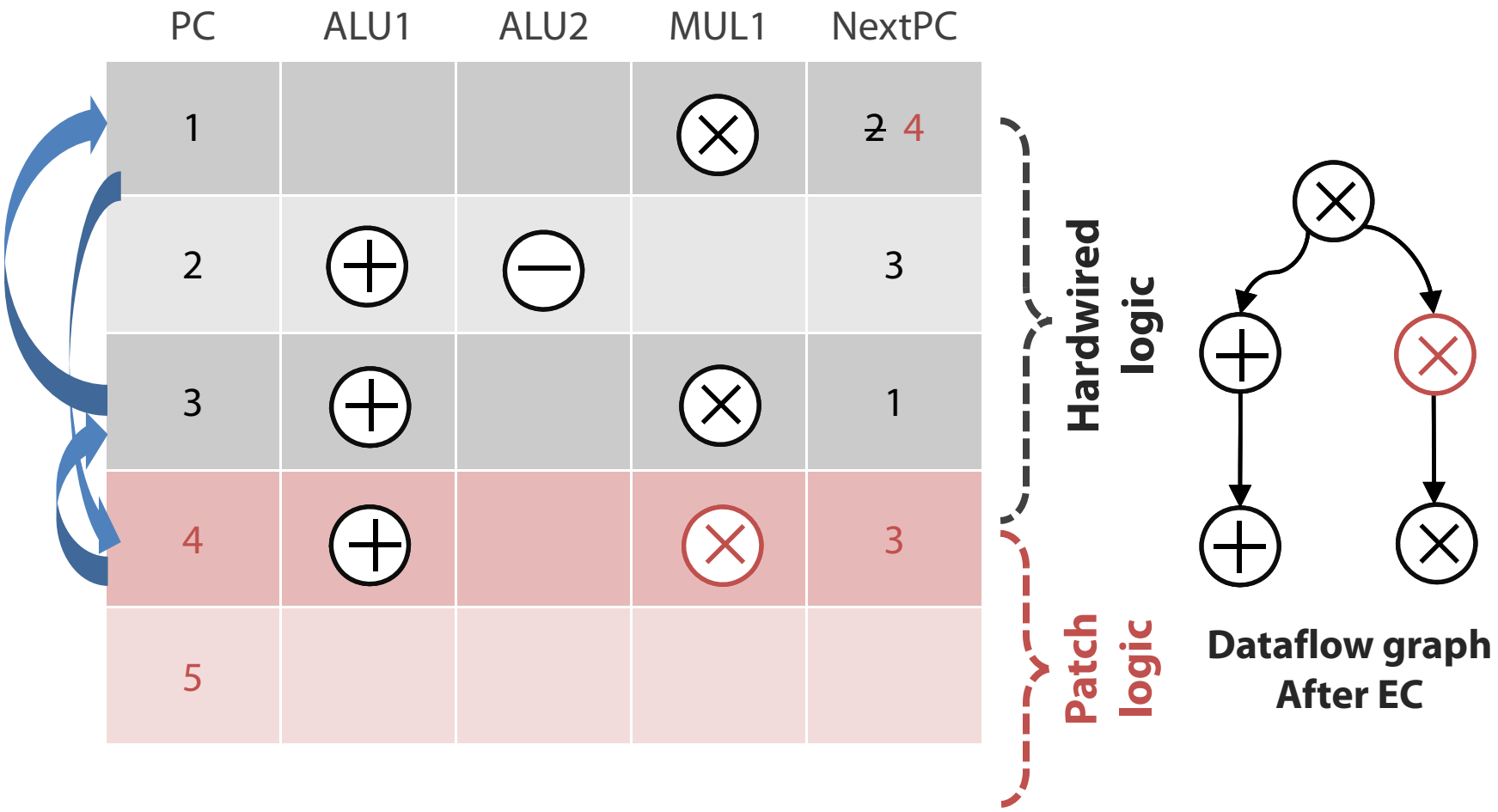
Patching Example (1/2)

Scheduling Result of Initial Design

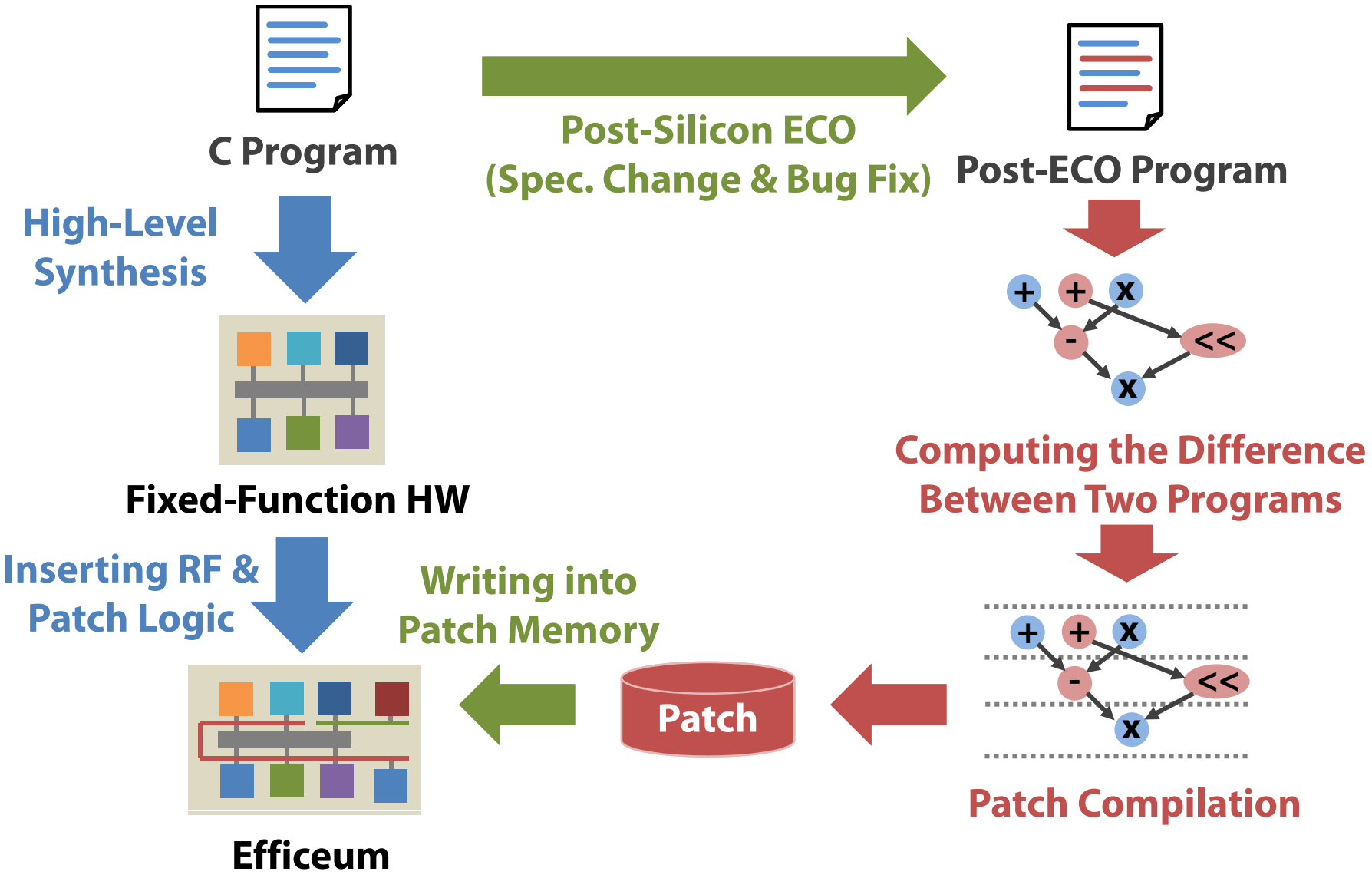


Patching Example (2/2)

Scheduling Result After Engineering Change



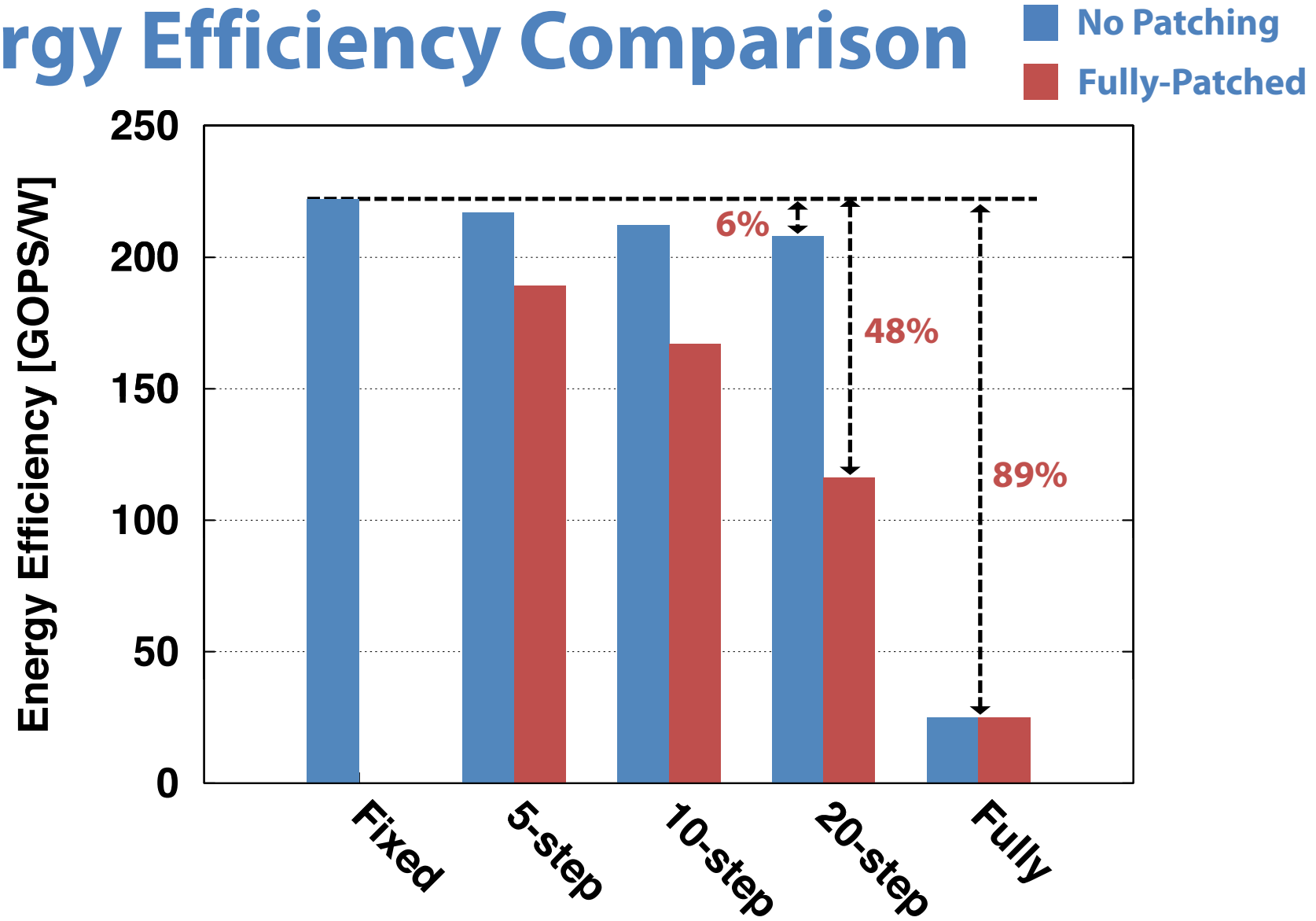
Patching-Based Post-Silicon ECO Flow



Experimental Setup

- Example: 8x8 IDCT
- Technology: FreePDK 45nm
- Logic Synthesis: Synopsys Design Compiler Ultra
  - High effort options with gated clock optimization
- P&R: Cadence SoC Encounter
- Simulation: Synopsys VCS
- Power/timing analysis: Synopsys PrimeTime PX
  - Simulation results are used for power calculation
- Energy efficiencies (GOPS/W) are compared

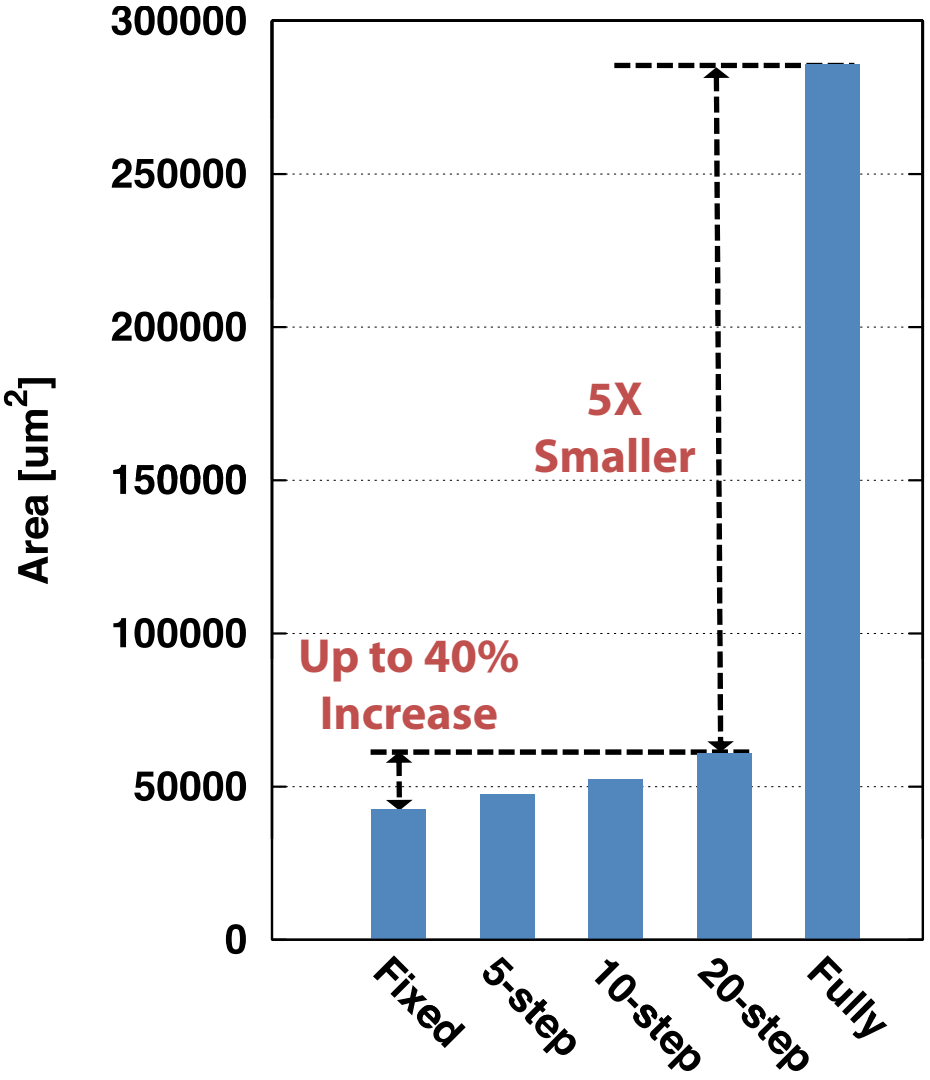
Energy Efficiency Comparison

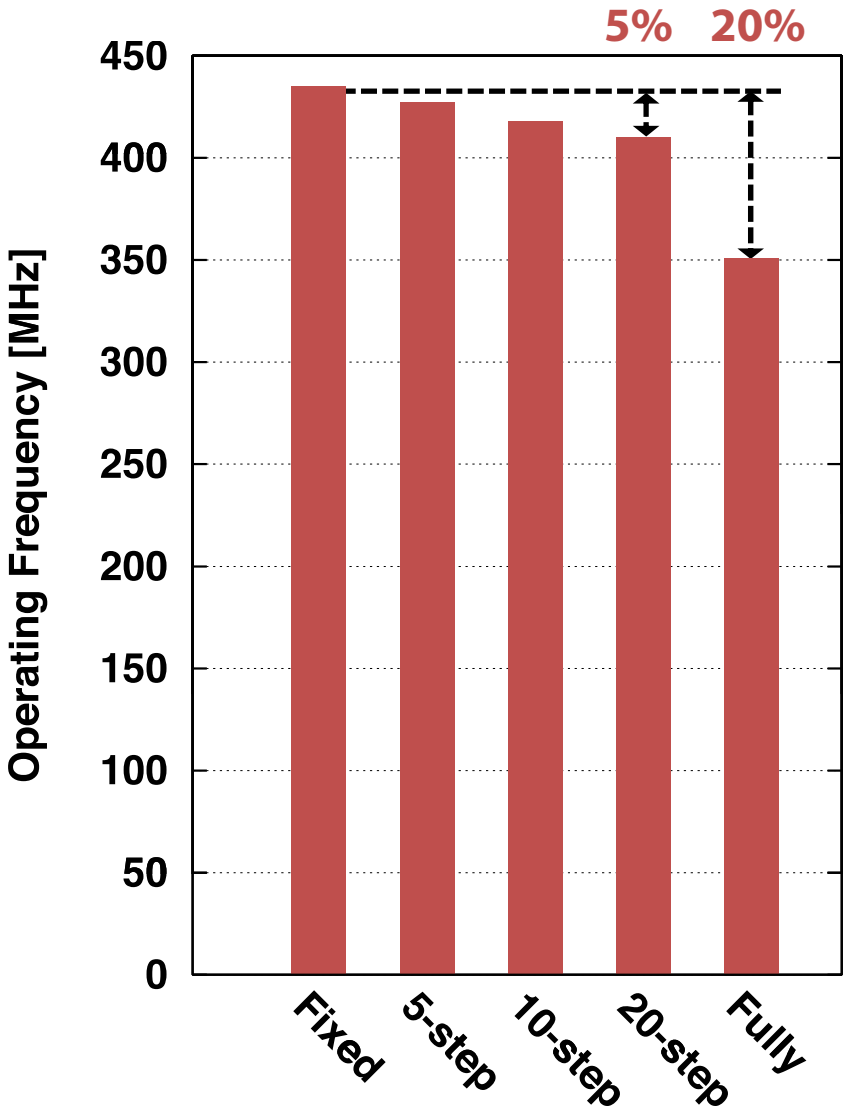


8x8 IDCT (FreePDK 45nm technology)

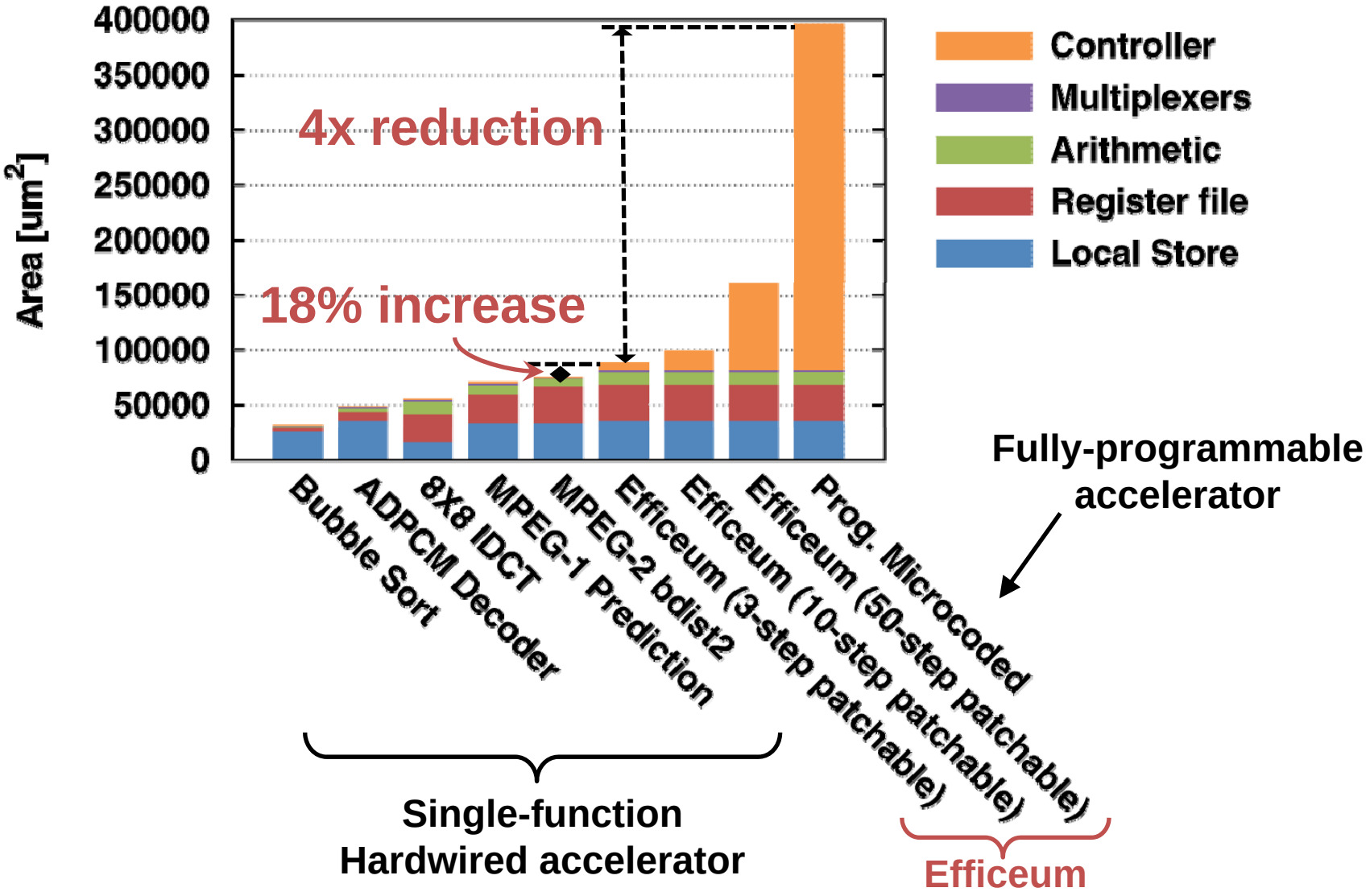
Offers a tradeoff between efficiency and programmability

Area & Performance Comparison



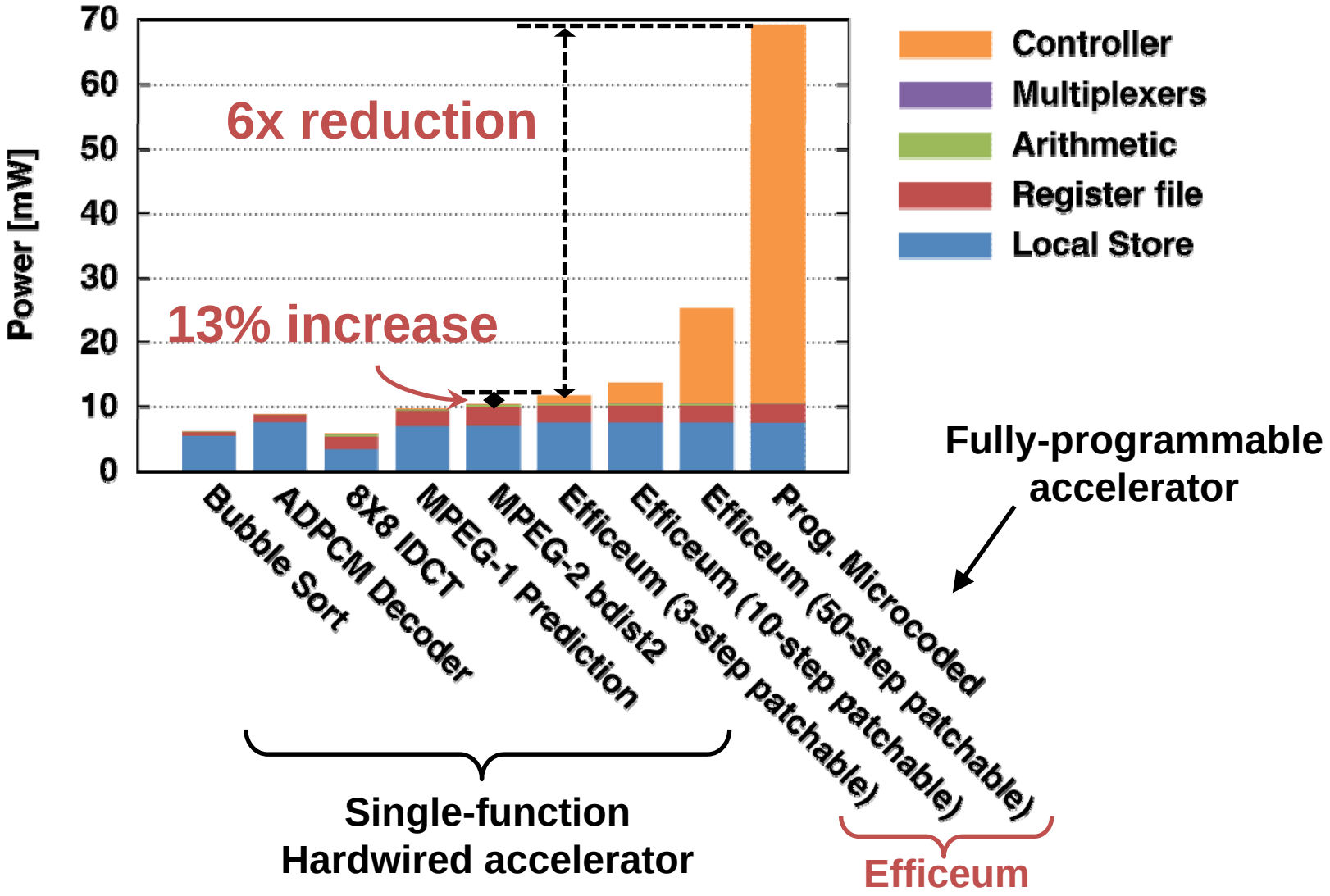


Area Comparison



(Technology: FreePDK 45nm (NCSU/Nangate), Operating Frequency: 200MHz)

Power Comparison

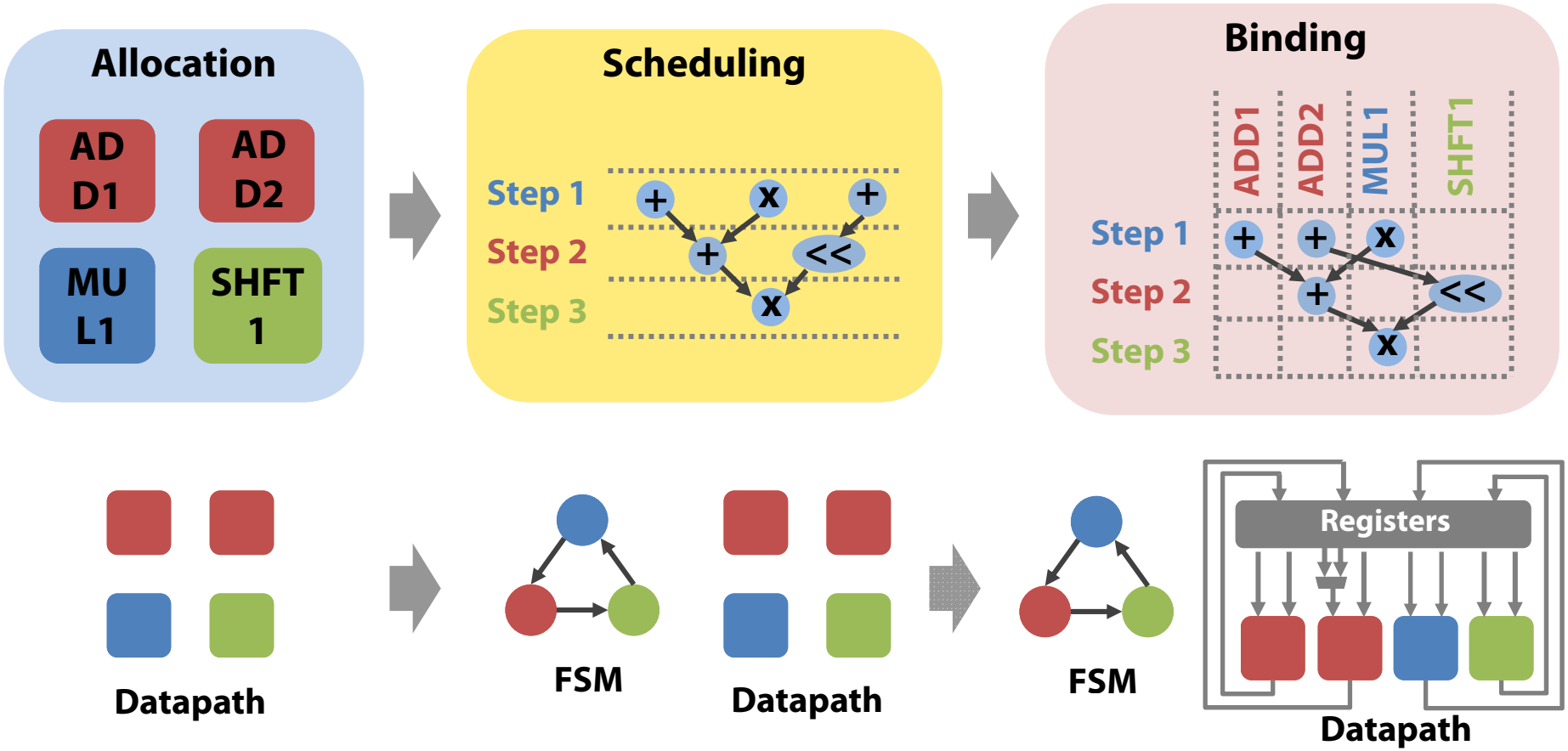


(Technology: FreePDK 45nm (NCSU/Nangate), Operating Frequency: 200MHz)

Incremental High-Level Synthesis and Patch Compilation For High-Level ECO

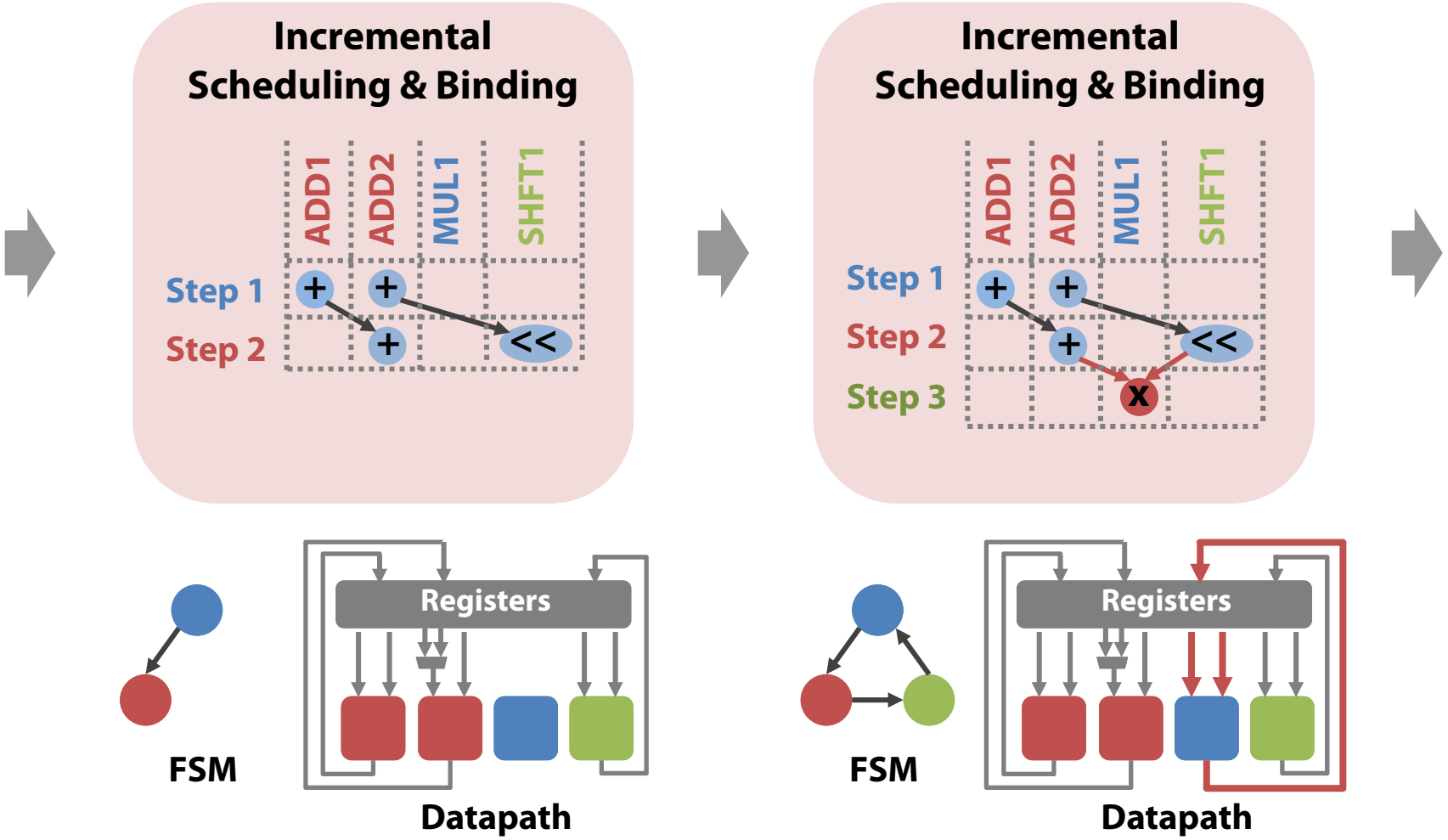
Conventional High-Level Synthesis

- Several phases are applied separately
  - This prevents incremental synthesis



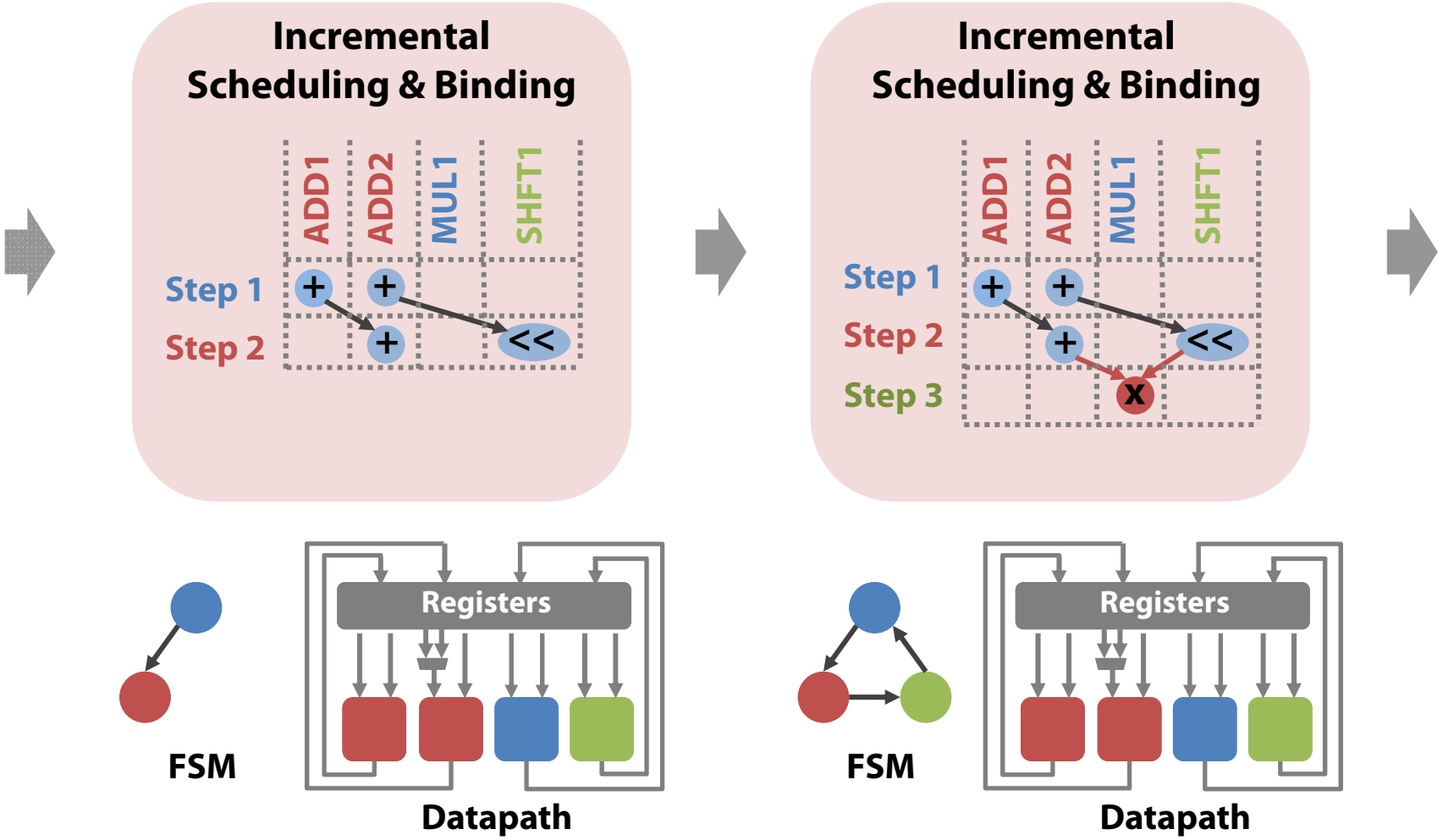
Incremental High-Level Synthesis

- Each operation is scheduled and bound incrementally, and the hardware is enhanced accordingly



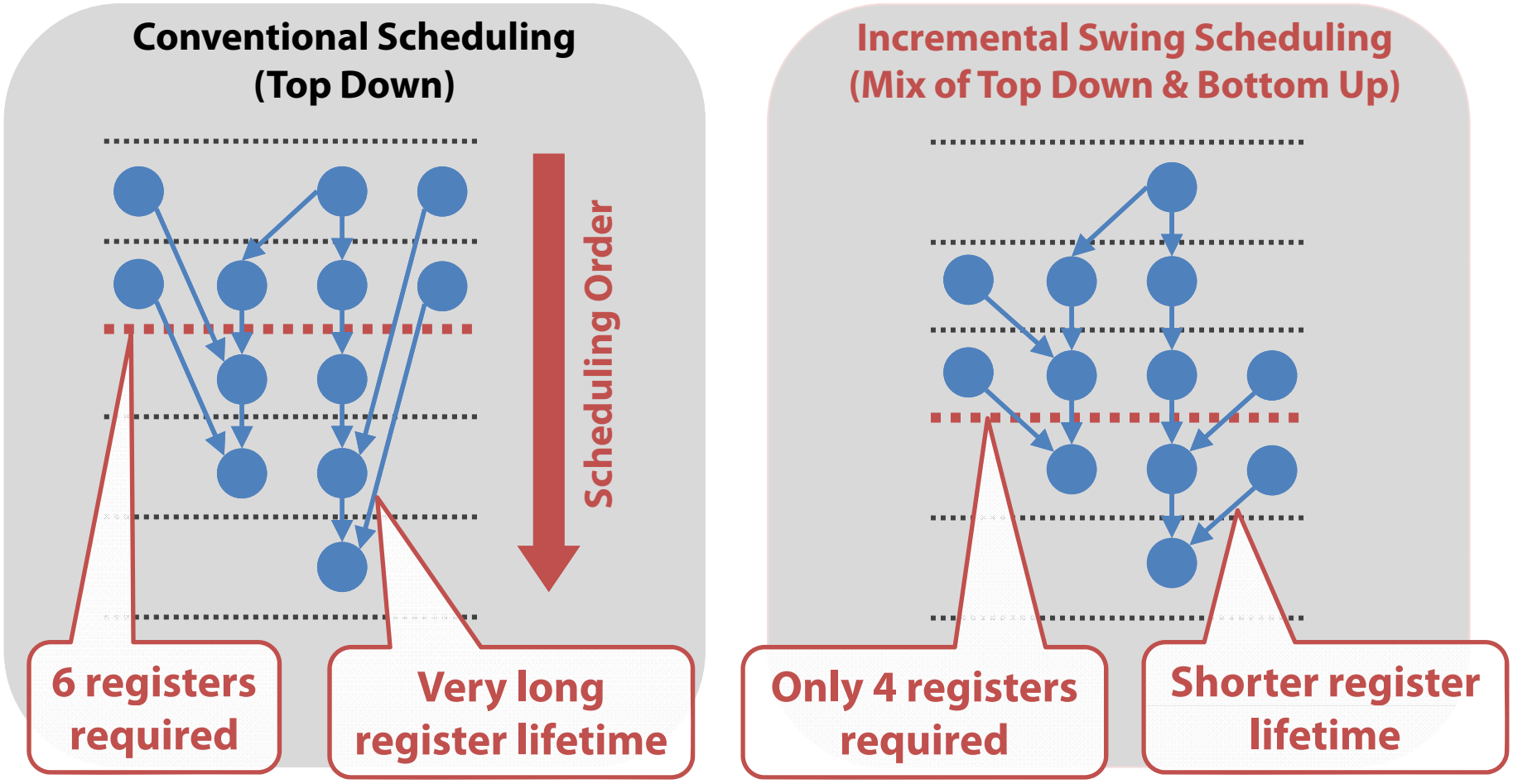
Patch Compilation

- Same as incremental synthesis, except the datapath enhancement is not allowed (only FSM is enhanced)



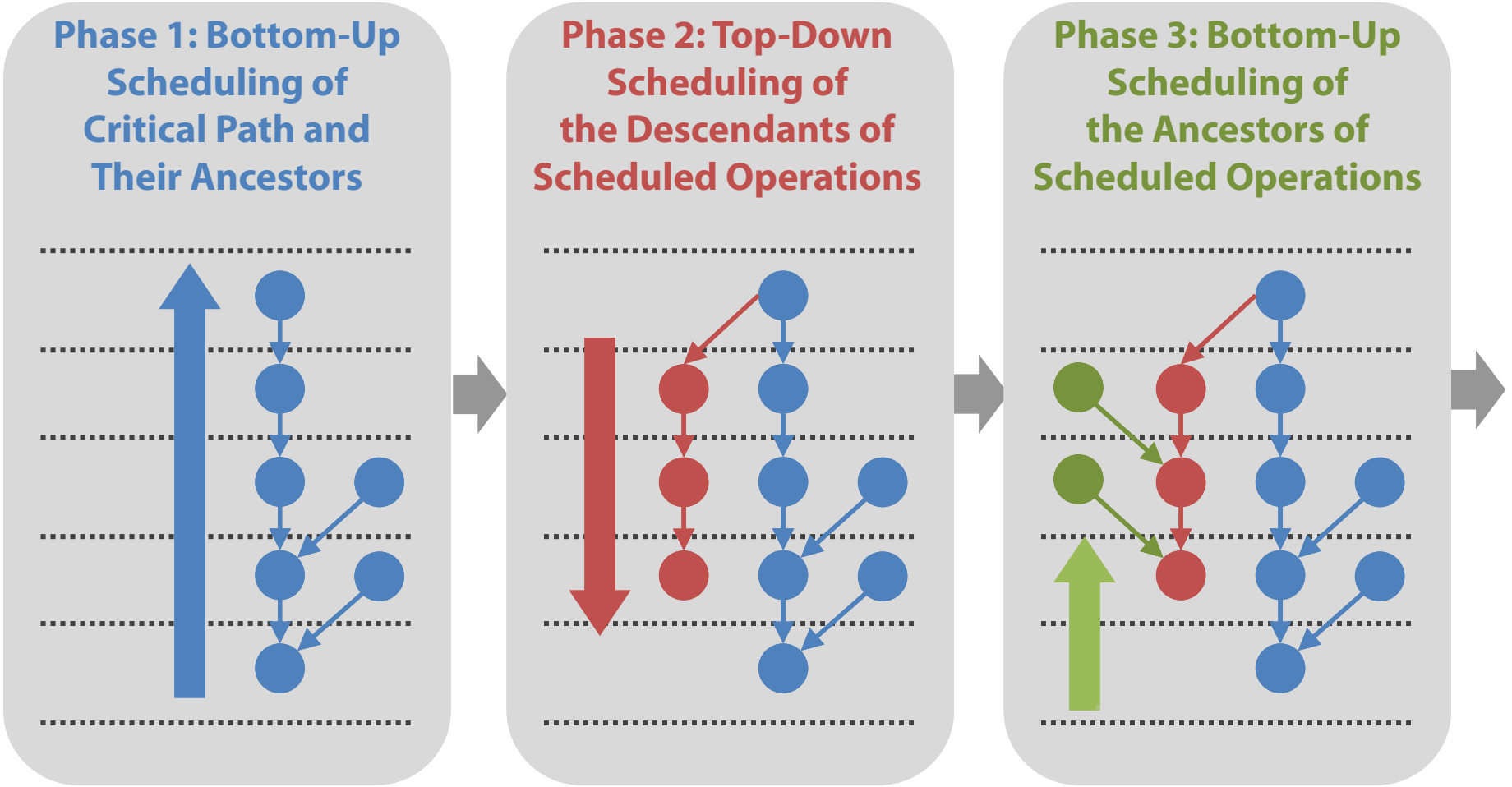
Incremental Scheduling Procedure

- Extension of Swing Modulo Scheduling for VLIW compilers [Llosa et al., PACT '96]



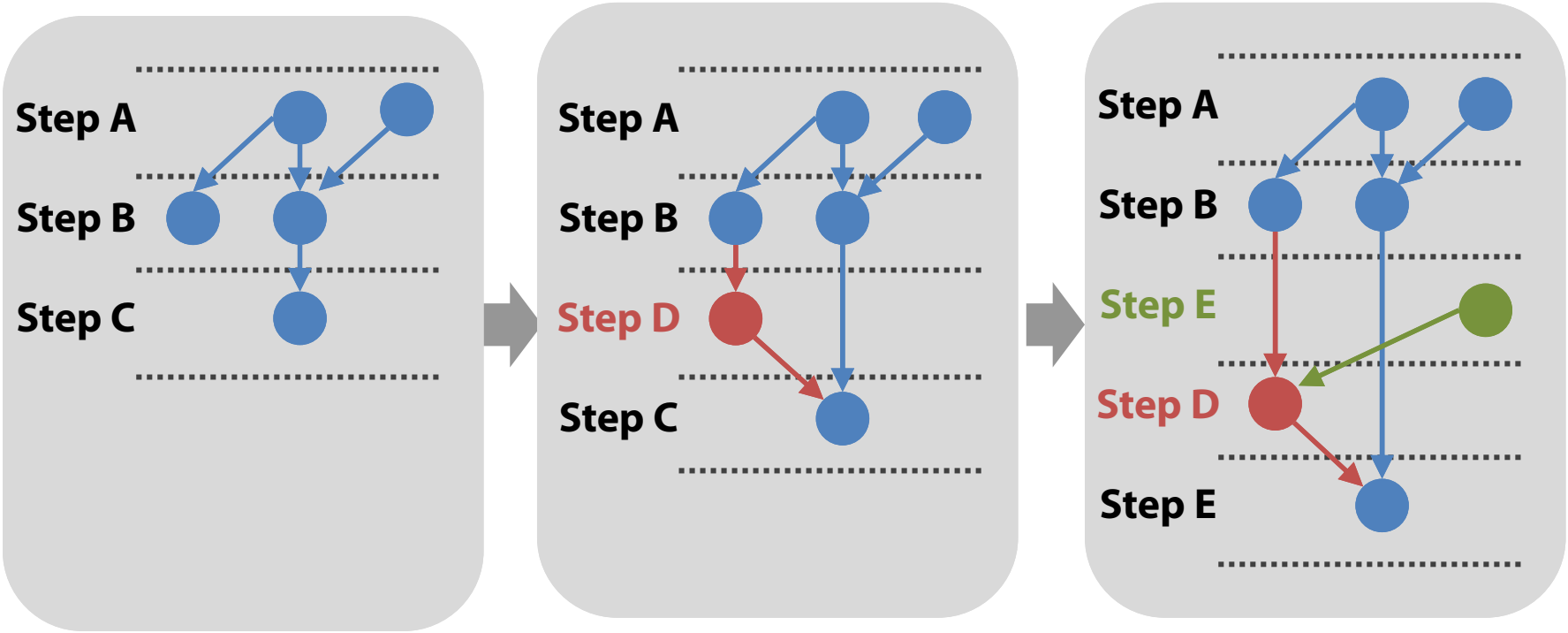
Swing Scheduling Procedure

- Mix of top-down and bottom-up scheduling



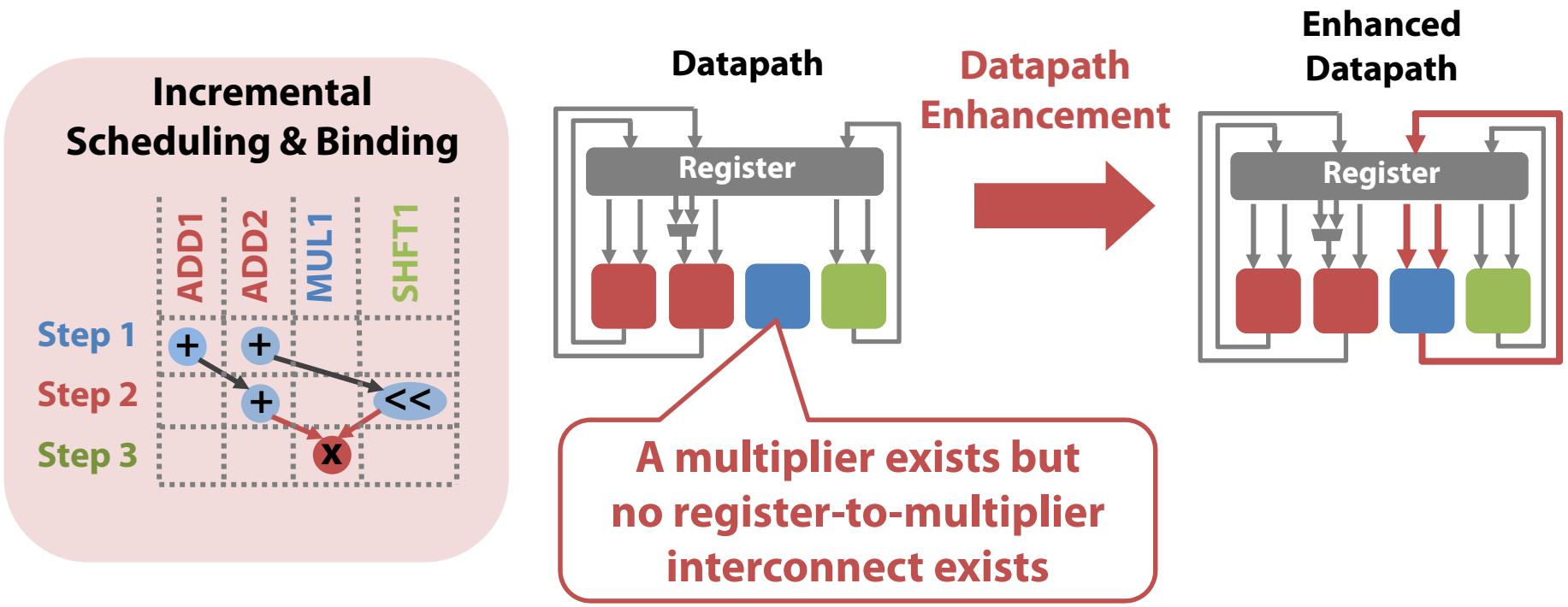
Incremental Step Insertion

- A novel technique enabling incremental swing scheduling
  - During swing scheduling, a new control step is inserted between the scheduled steps when needed



Incremental Binding Procedure

- For each operation, all possible combinations of (resource, registers) are examined
  - If no such binding is found, new interconnects between resource and registers are introduced

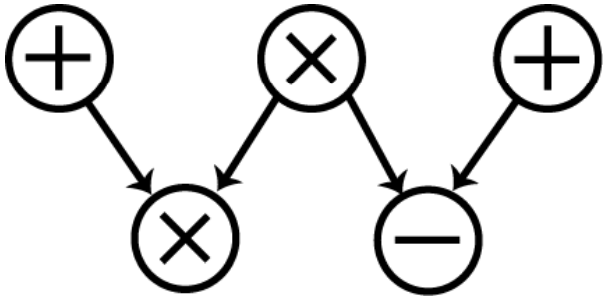


Experimental Setup

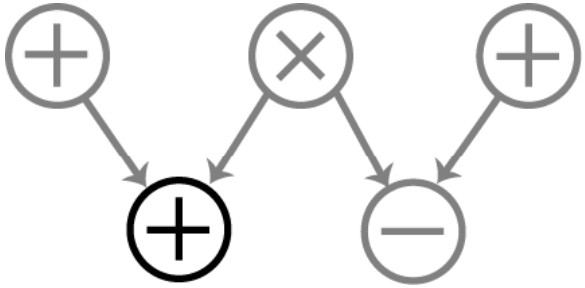
- The proposed method has been implemented in our high-level synthesis framework Cyneum
  - Incremental high-level synthesis
  - Patch compiler for Efficieum
- Example: 5 benchmark designs
  - C programs of about ~100 lines
  - Functions from IDCT, ADPCM, MPEG
  - Post-ECO examples are generated by random graph perturbation (next slide)
- Evaluated the quality of the method through the patch size and compilation time

Generating ECO Examples

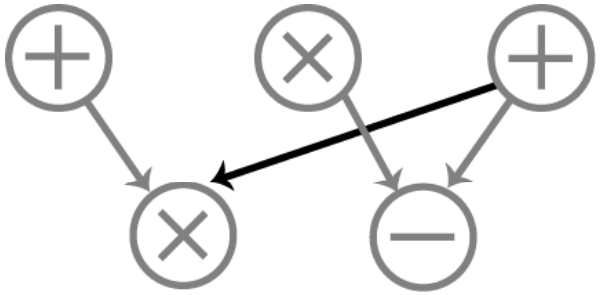
- The following graph perturbations are randomly applied to CDFG



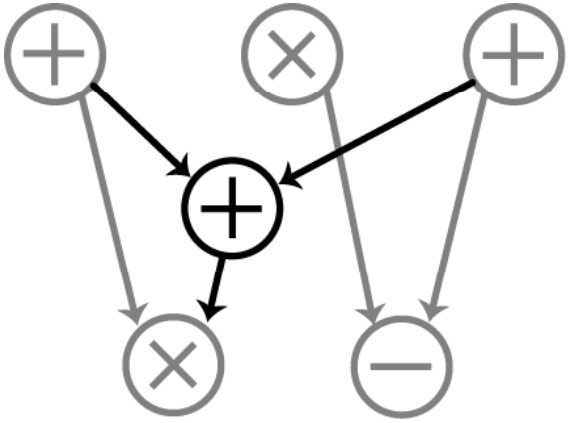
Original CDFG



Perturbation 1: Opcode Change



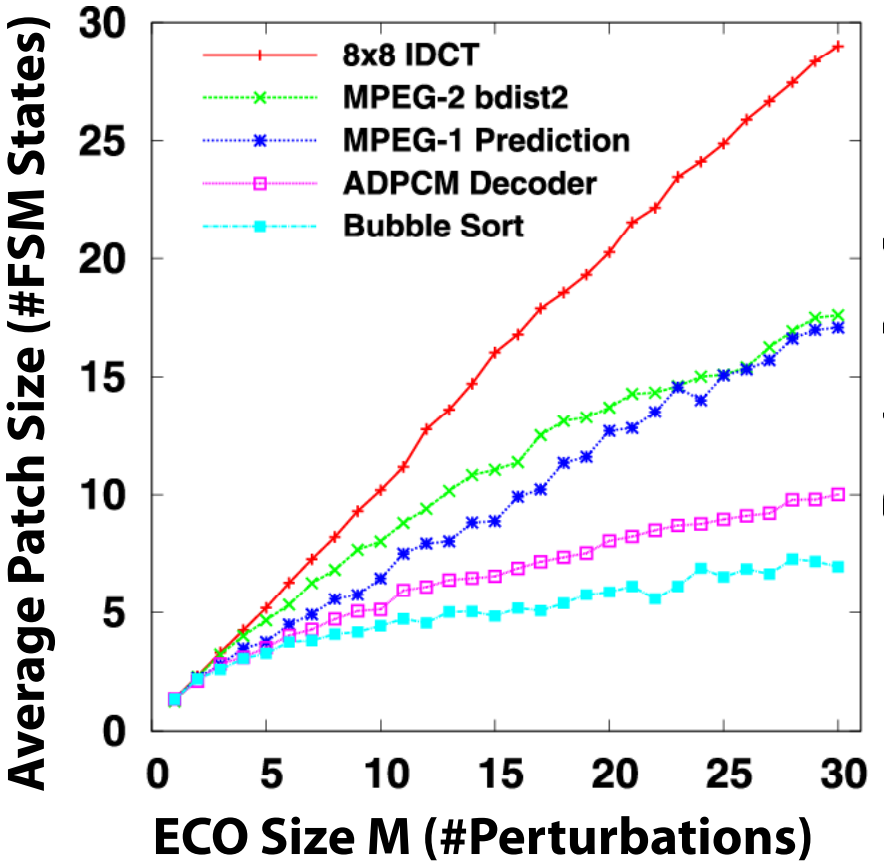
Perturbation 2: Operand Change

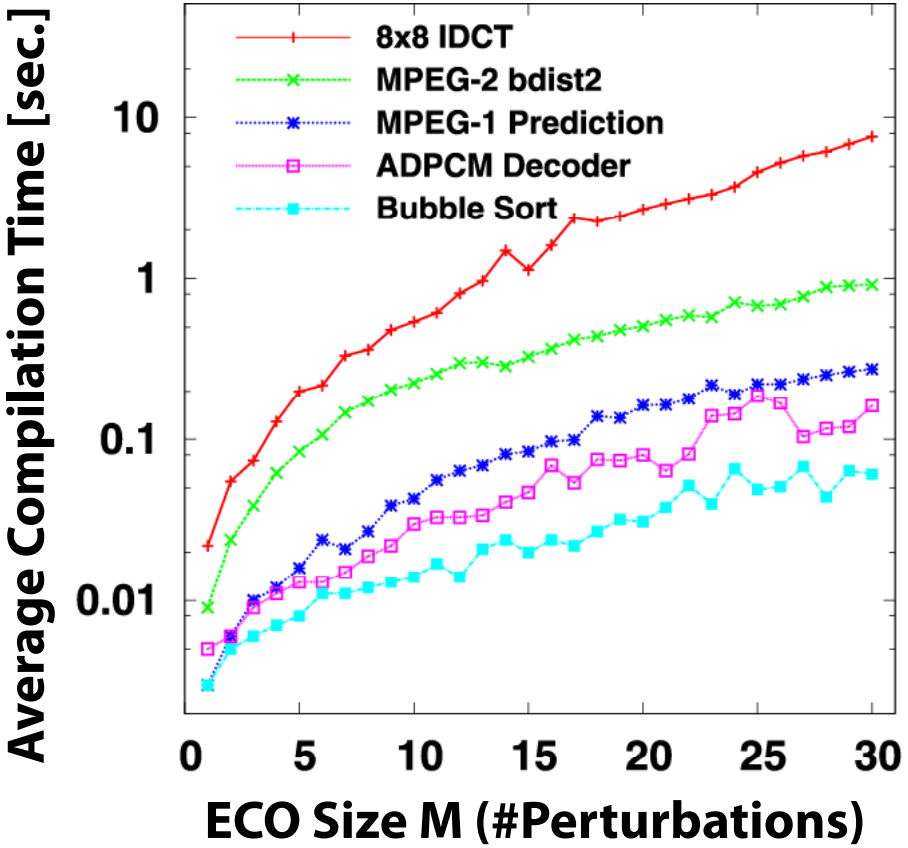


Perturbation 3: Introducing A New Operation

Evaluation of Patch Compiler

- For each benchmark, random CDFG perturbation is applied M times. For each M, 100 different post-ECO designs are generated and then patches are compiled.





PPC:
Increasing Yield Using Partially-Programmable Circuits

A collaborative work with
Prof. Shigeru Yamashita (Ritsumeikan Univ.)

Design For Yield

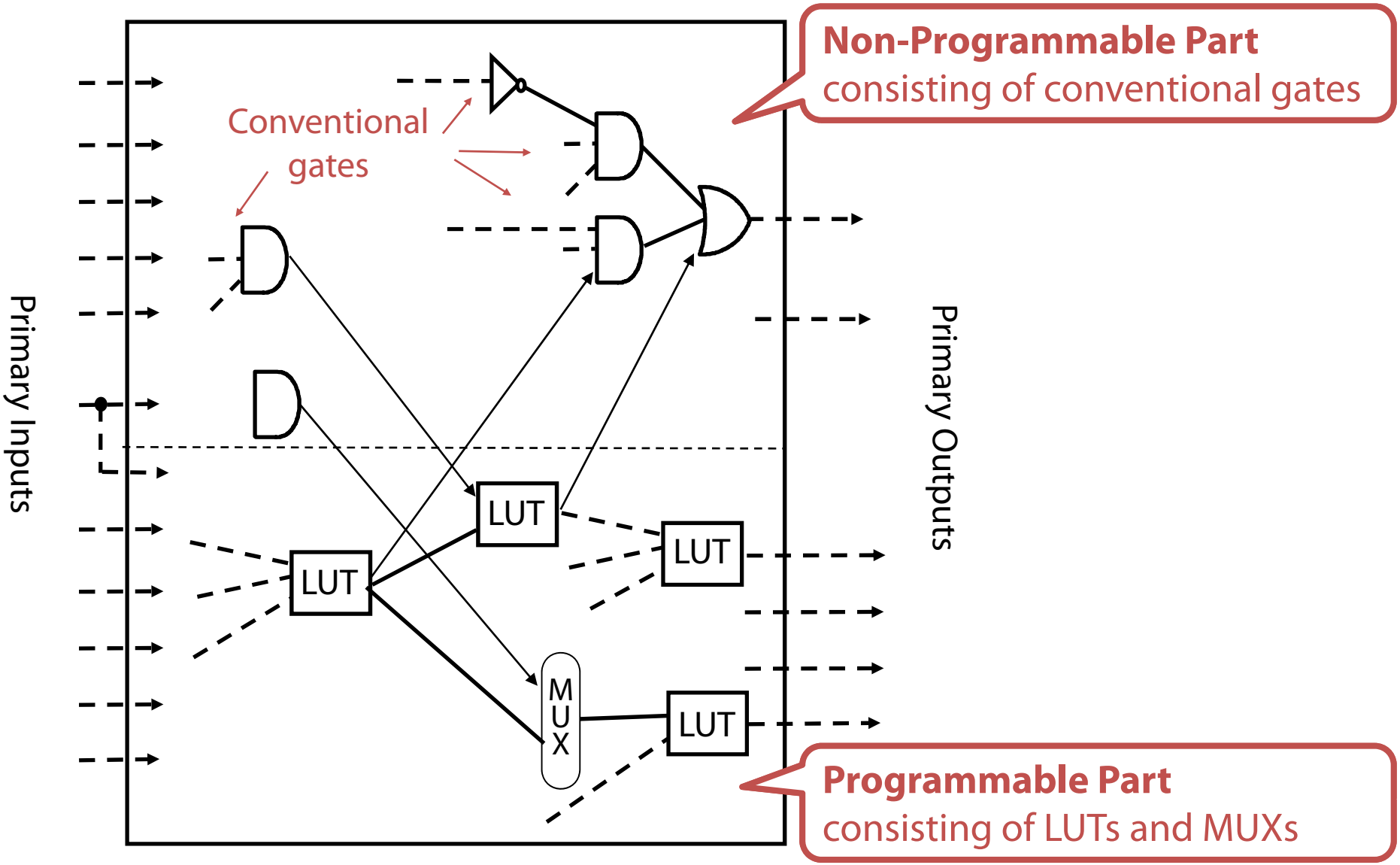
- At physical level, there are many techniques available for yield/defect tolerance enhancement
- At logic level, the following techniques can be applied for each module
  - TMR: Voting
  - DMR: If one module is defective, the other can be used
  - Reconfigurable devices: synthesize not to use defective parts

Too much overhead in area and performance

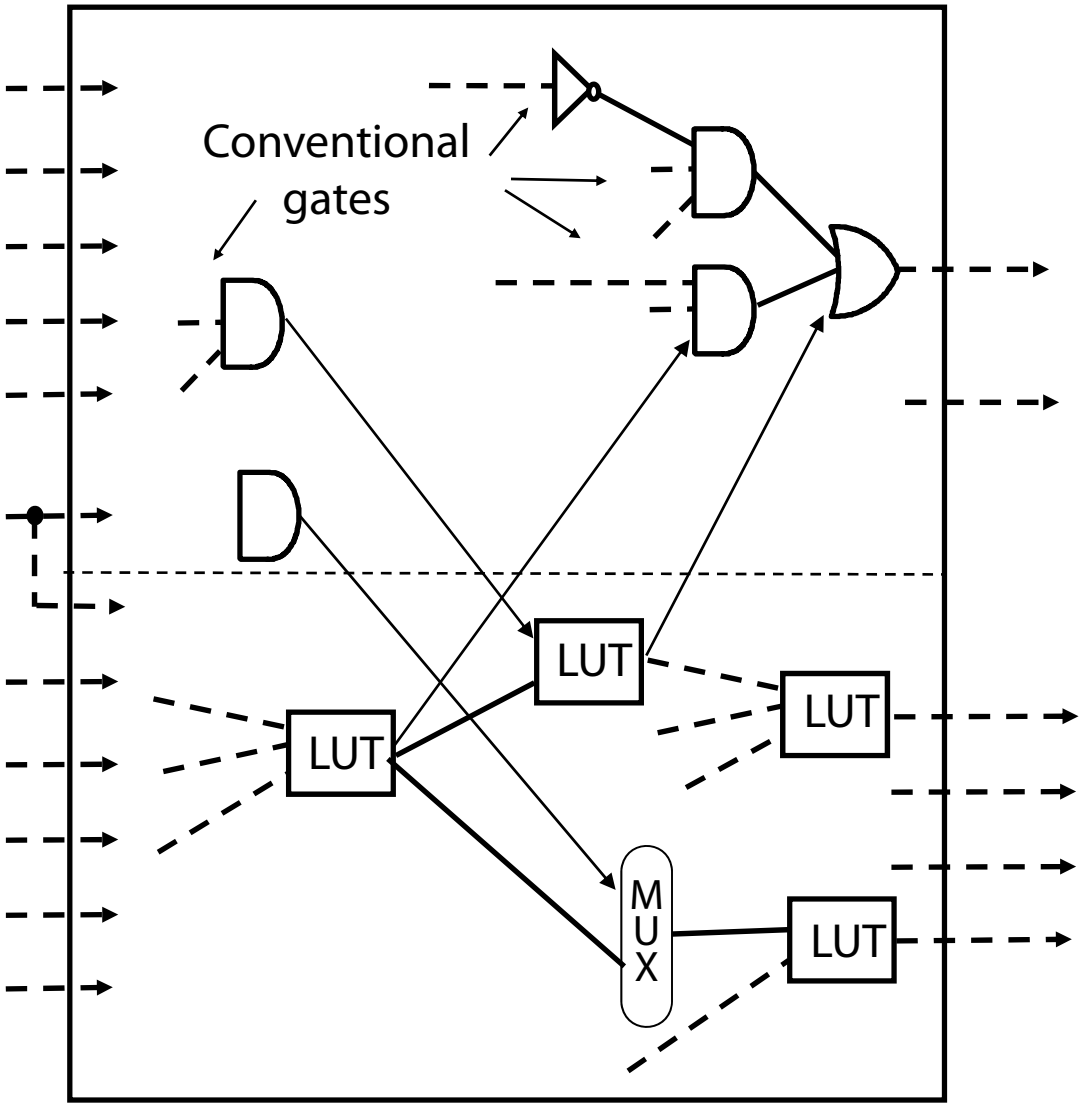
Objective of This Work

- Enhance the defect tolerance by using a Partially-Programmable Circuit (PPC)
  - PPC is a hybrid LUT/gate circuit
  - To correct a single defect, full programmability such as FPGAs is unnecessary
  - A defective wire can be made redundant by reprogramming LUTs in PPC
- Propose a design methodology
  - Synthesis of PPC
    - Where to put LUTs
  - How to reprogram LUTs for defective wires

PPC (Partially-Programmable Circuit)



Defect Correction in PPC

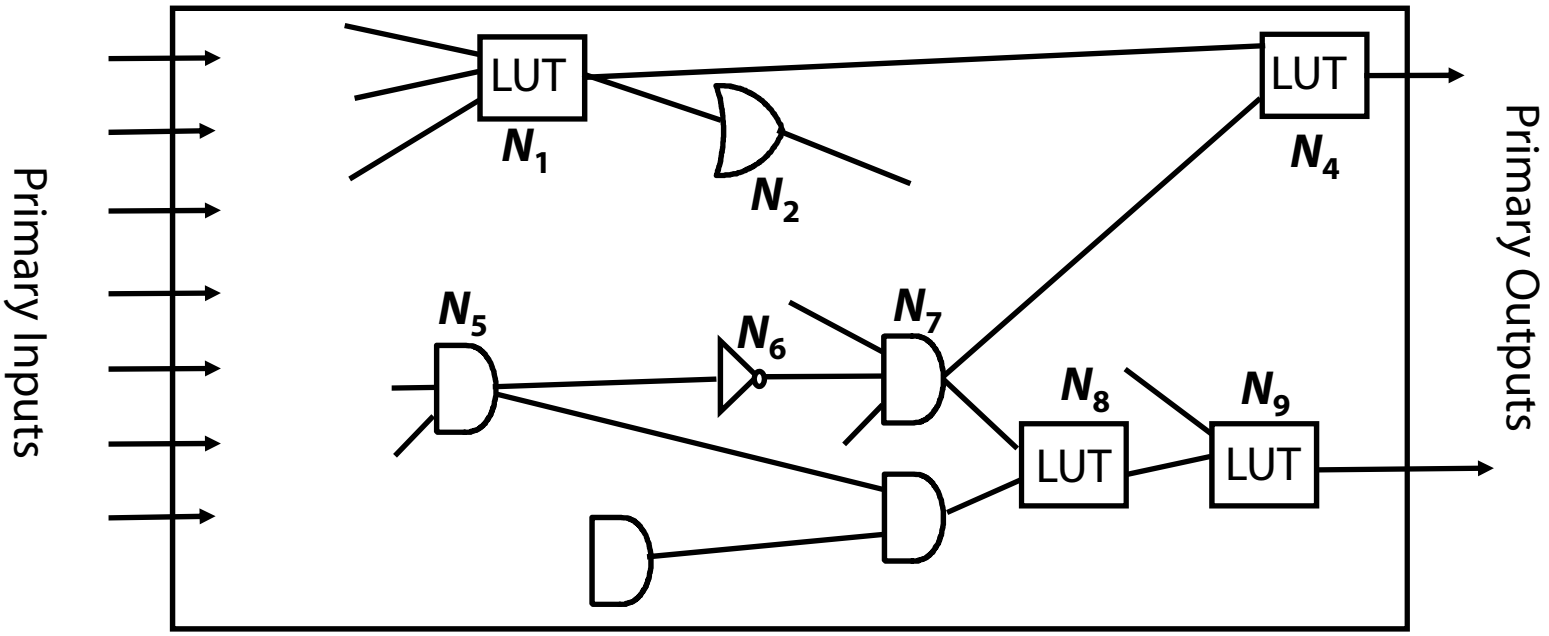


Find out that
a wire c_i is defective

By reprogramming LUTs, the
wire c_i becomes redundant

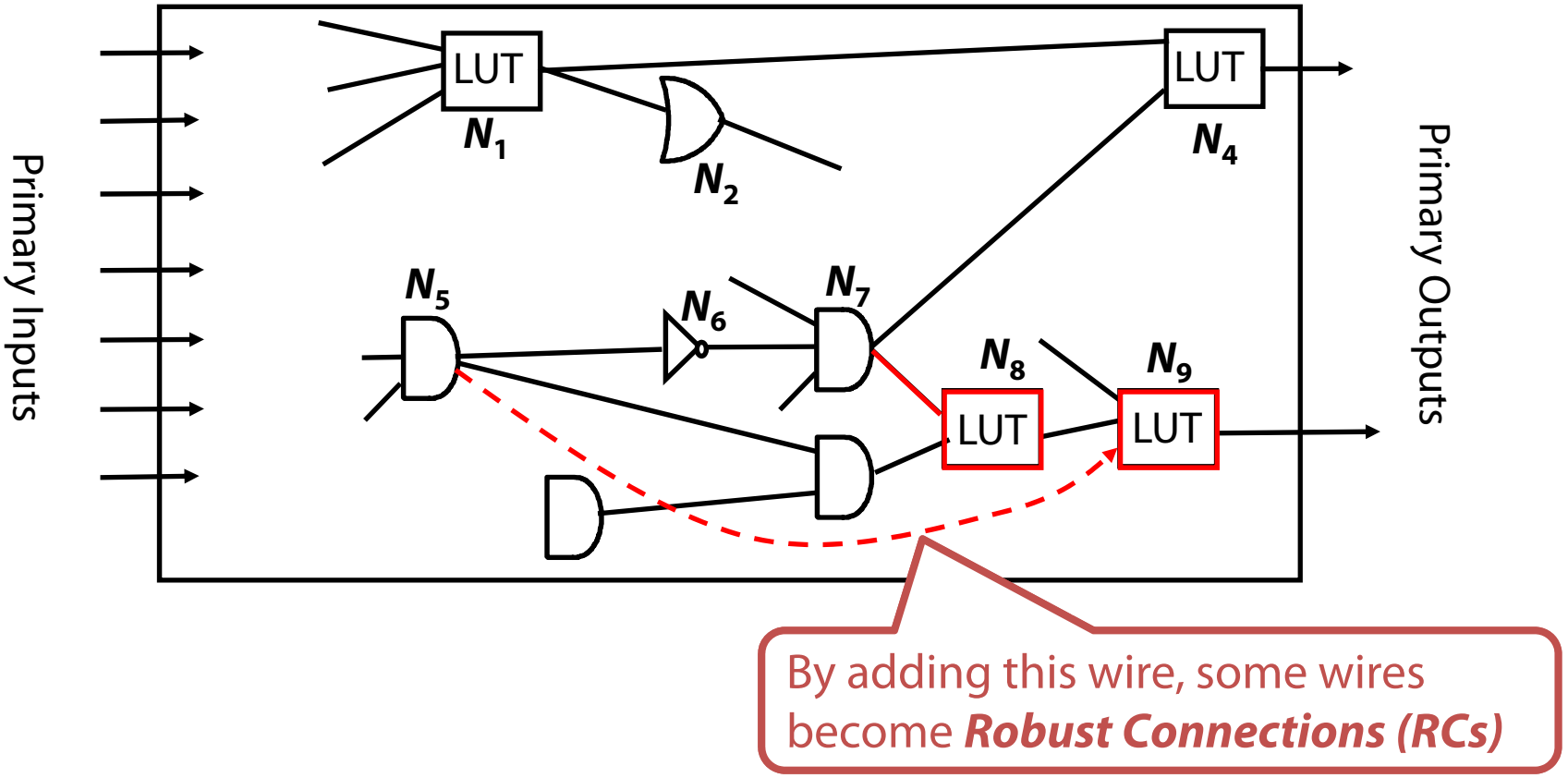
Call c_i as ***Robust Connection (RC)***

PPC Example: Initial Circuit



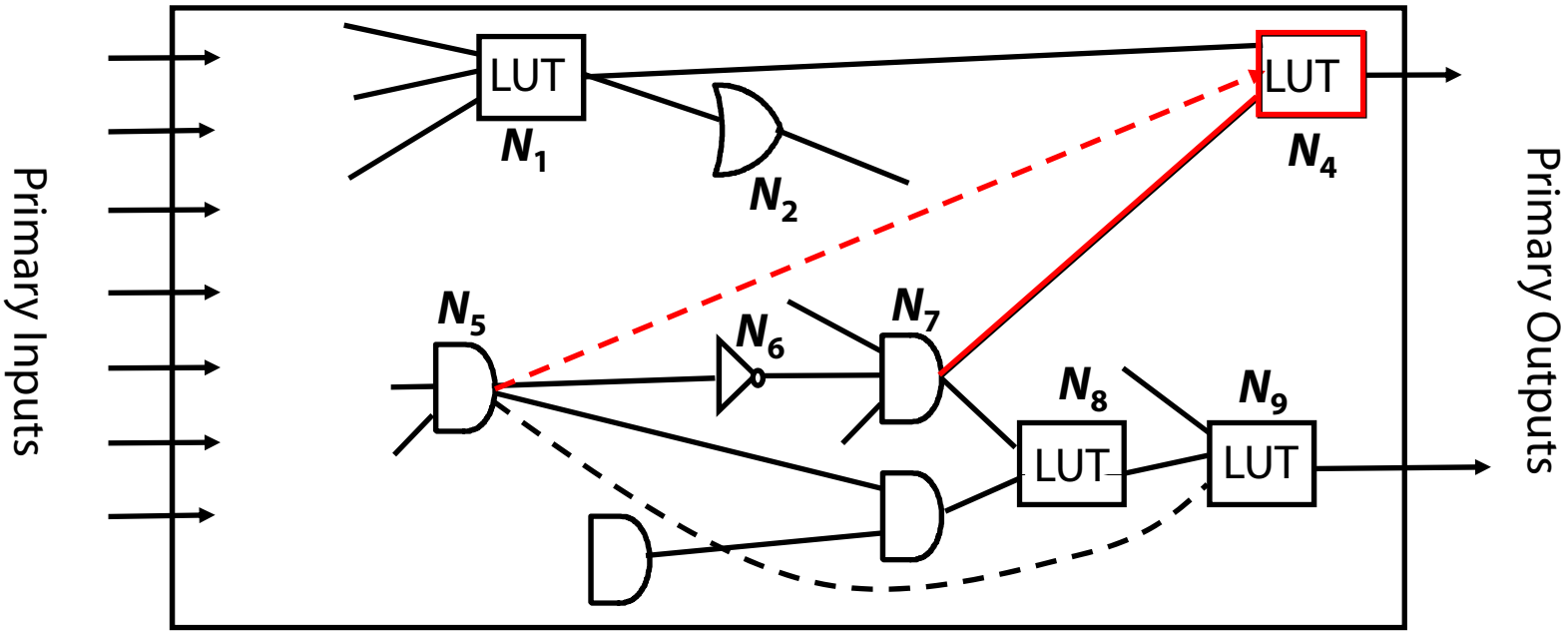
- LUTs are used partially in the circuit
- There is ***no redundancy*** now

PPC Example: Redundancy Addition 1

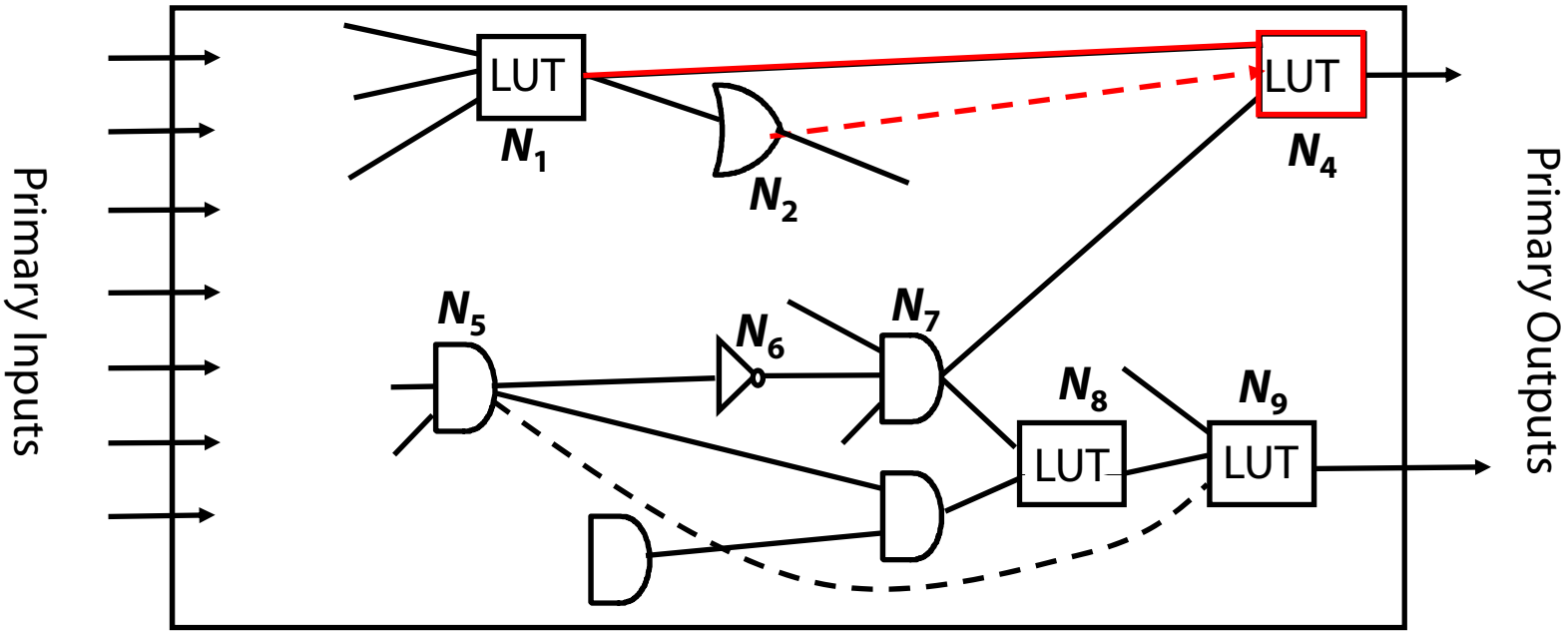


RC: Wires which can become redundant by reprogramming LUTs
NRC: Wires which are not RCs

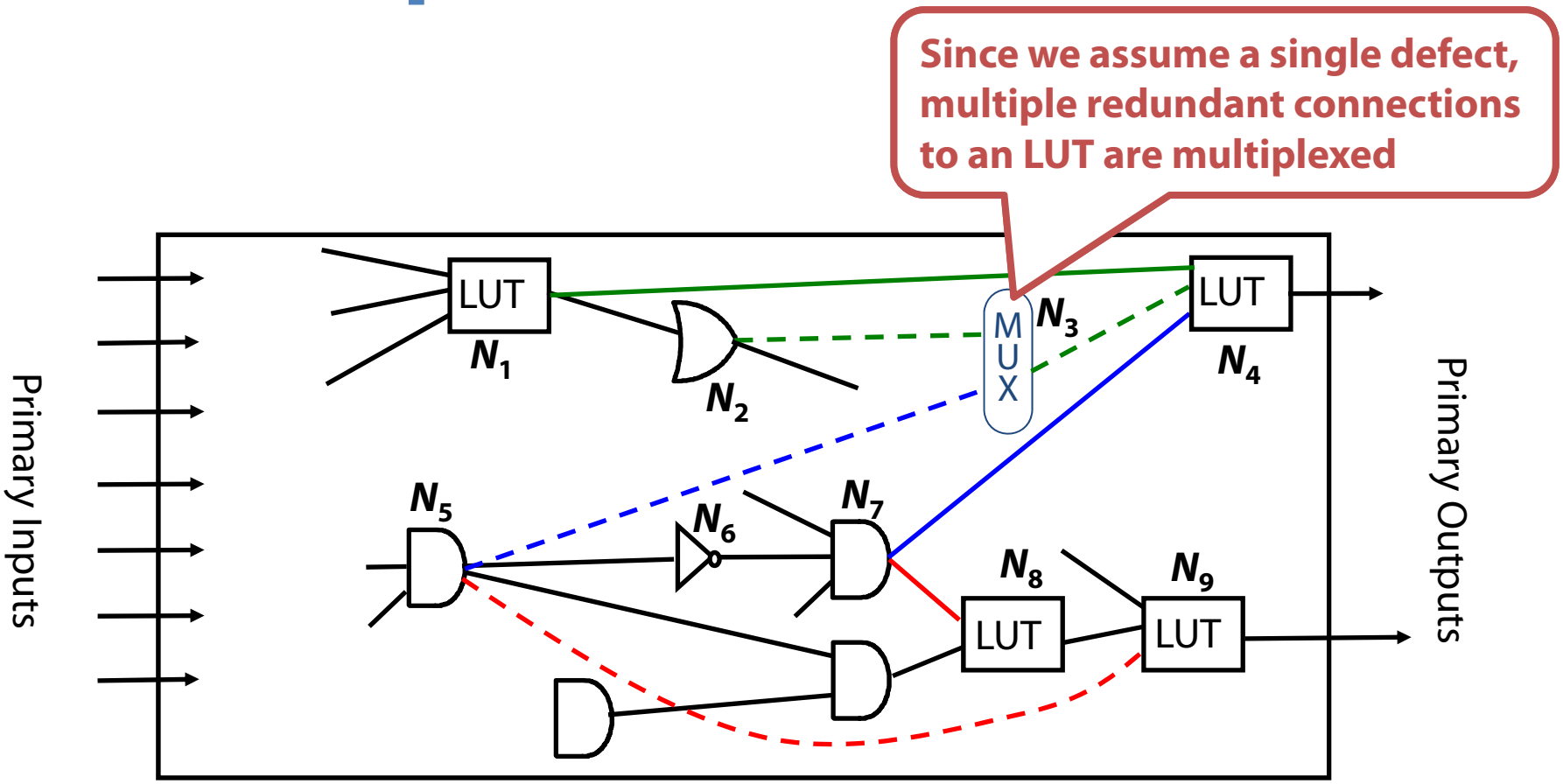
PPC Example: Redundancy Addition 2



PPC Example: Redundancy Addition 3



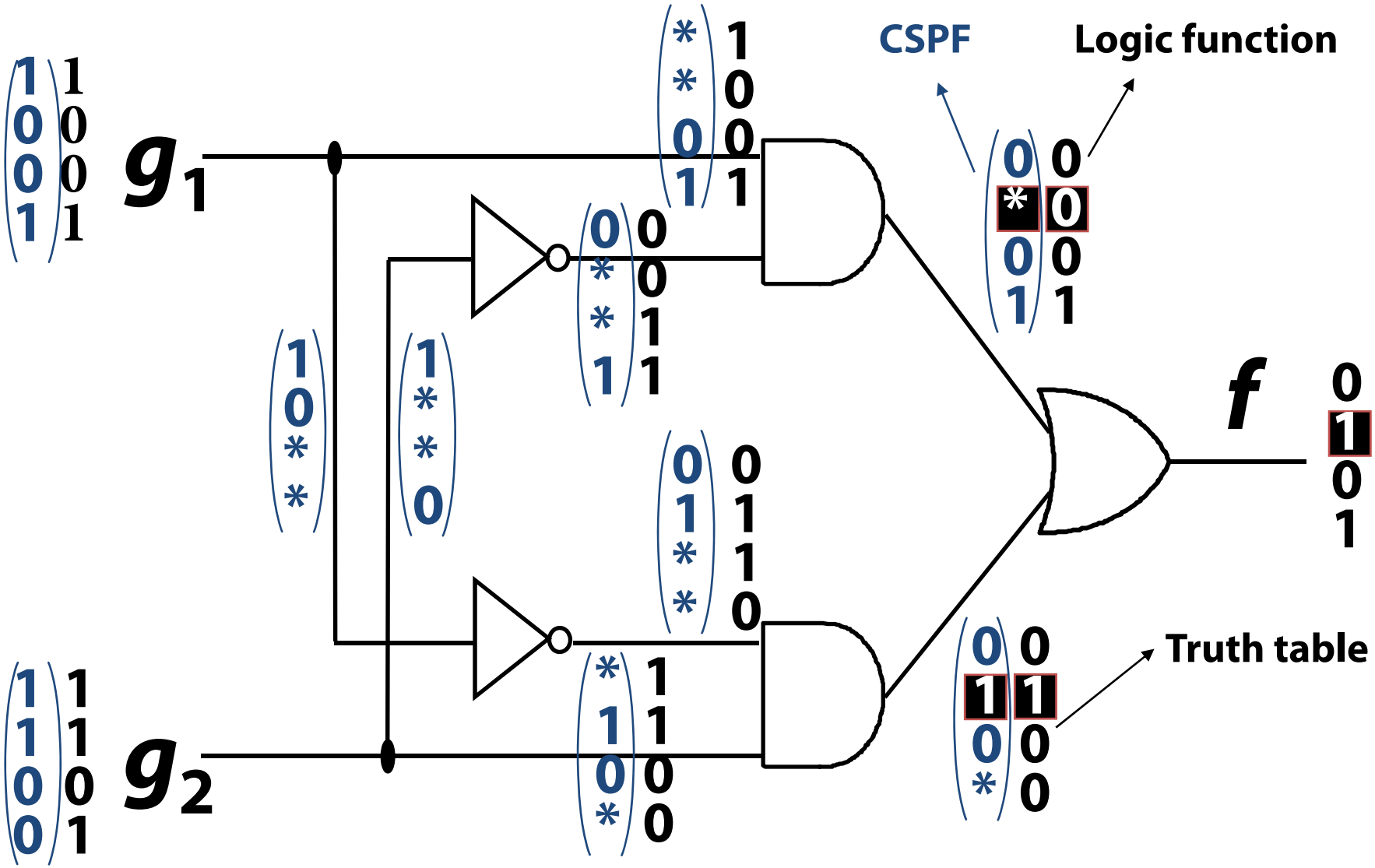
PPC Example: Final Circuit



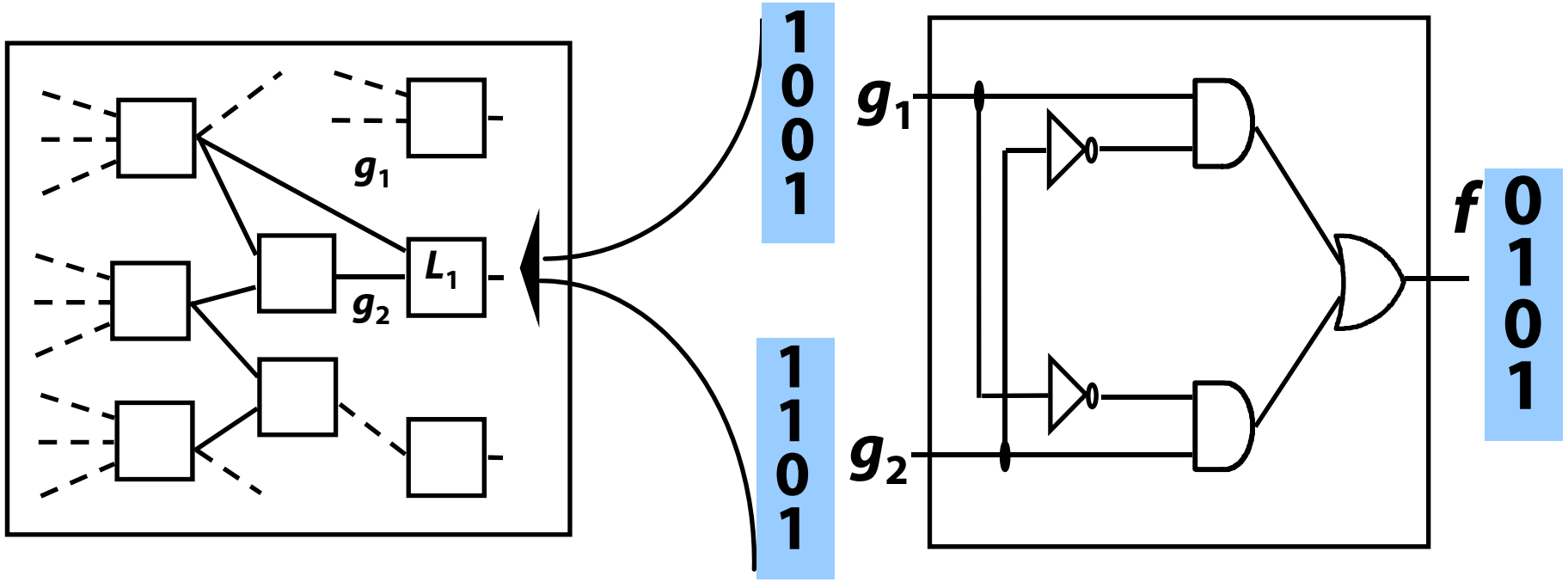
Colored wires: Robust Connection (RC)

Black wires: Non-Robust Connection (NRC)

CSPF: Flexibility of Logic Circuits



SPFD: Flexibility of LUT Circuits



g_1 's flexibility by SPFDs

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

Proposed Synthesis Method

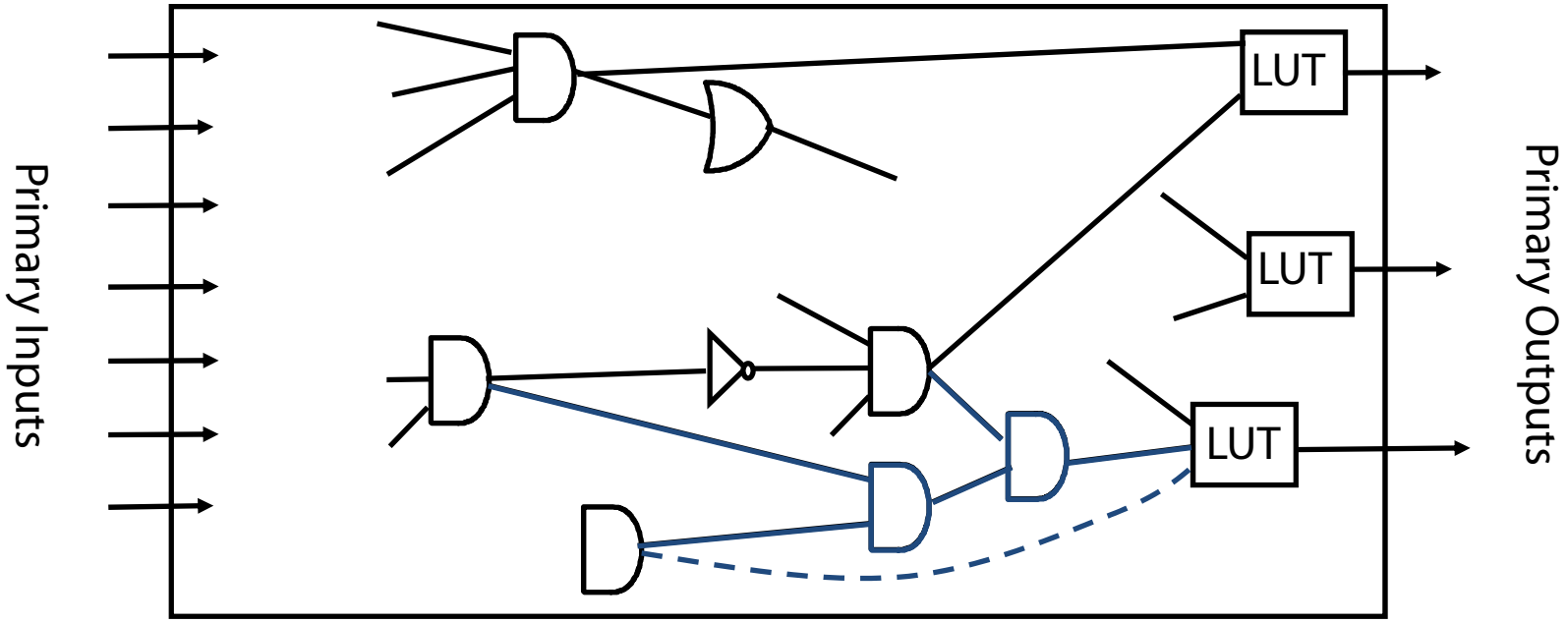
- Basic procedure
  1. Perform a LUT mapping
  2. Determine LUTs to keep
     - Needs a better heuristic
       - Reconvergence points, non-critical nodes
  3. Perform a technology re-mapping
  4. Adding redundant wires to LUTs
     - How to find good wires?
     - Currently, wires are identified exhaustively

Preliminary Experiments

1. Mapping to K-input LUTs (K=3,4,5)

2. Re-mapping with keeping LUTs at the outputs

3. For each LUT, if connecting a wire to the LUTs make another wire RC, then it is selected. Terminate if the number of LUT inputs is 6.



Experimental Results

#Connections which are robust to stuck-at-0/1

Circuit	Stuck-at-0			Stuck-at-1		
	Robust		Non-Robust	Robust		Non-Robust
	Original	Added		Original	Added	
alu2	586	13	25	582	20	29
alu4	1093	28	92	1070	25	115
apex6	591	86	106	589	98	108
rot	421	161	218	411	172	228
too_large	472	15	256	453	9	275
vda	1251	153	221	1318	213	154
C880	185	32	144	212	57	117
C1355	219	225	143	390	176	182
C1908	626	37	66	599	36	93
C2670	710	59	176	704	66	182
C3540	1500	63	210	1462	52	248