Bubble Razor

An Architecture-Independent Approach to Timing-Error Detection and Correction

Matthew Fojtik, David Fick, Yejoong Kim, Nathaniel Pinckney, David Harris, David Blaauw, Dennis Sylvester

mfojtik@umich.edu

Electrical Engineering & Computer Science Department

The University of Michigan, Ann Arbor

Outline

- Issues with Prior Razor
- Bubble Razor Algorithm
- Circuitry and Implementation
- Area Overhead Tradeoffs
- Test Chip Results

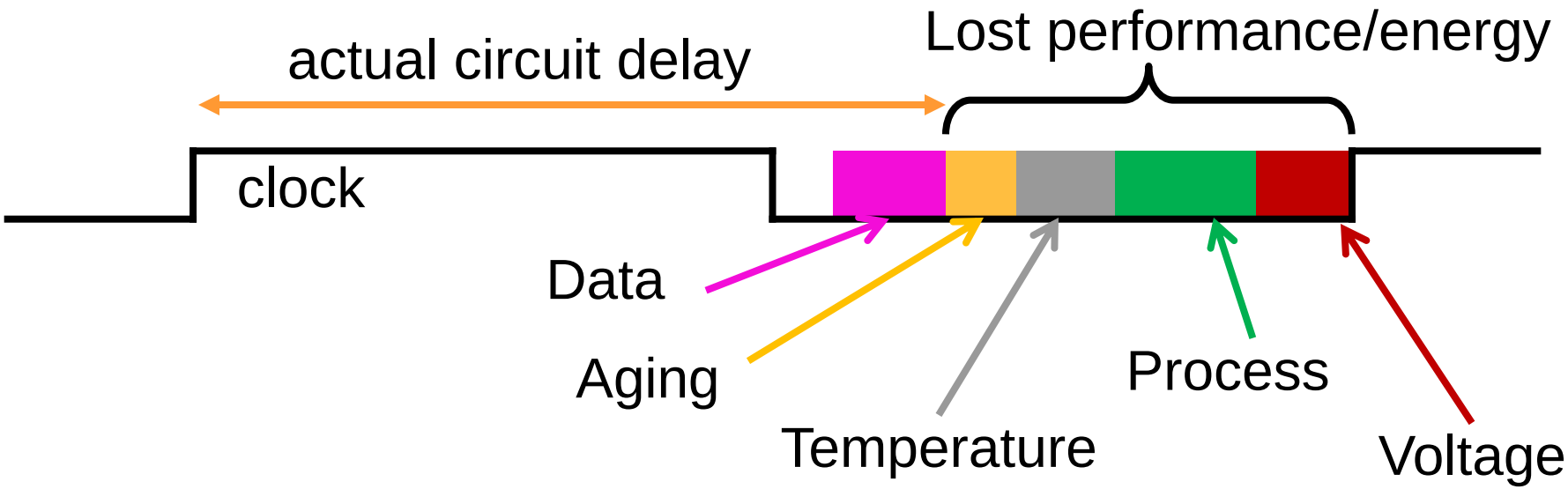
Timing Margins

Margins for uncertainty:

- Process Variation
- Temperature Variation
- Voltage Variation
- Aging Effects

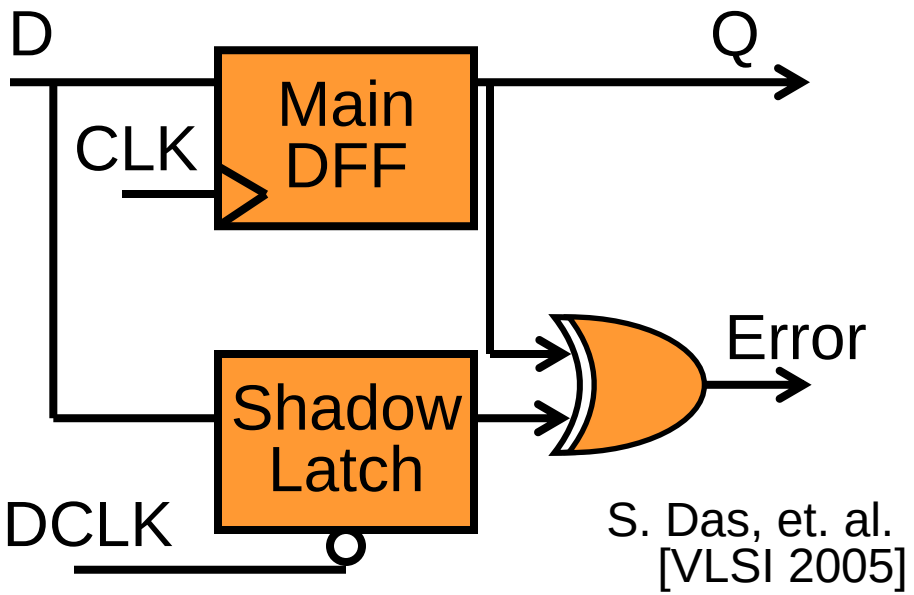
Associated Costs:

- Lost performance
- Lost energy
- Tester time (tradeoff)



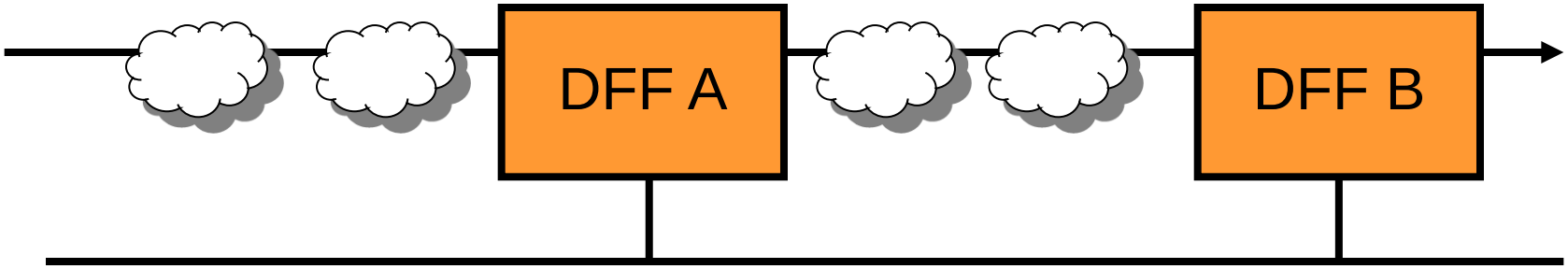
Eliminating Margins

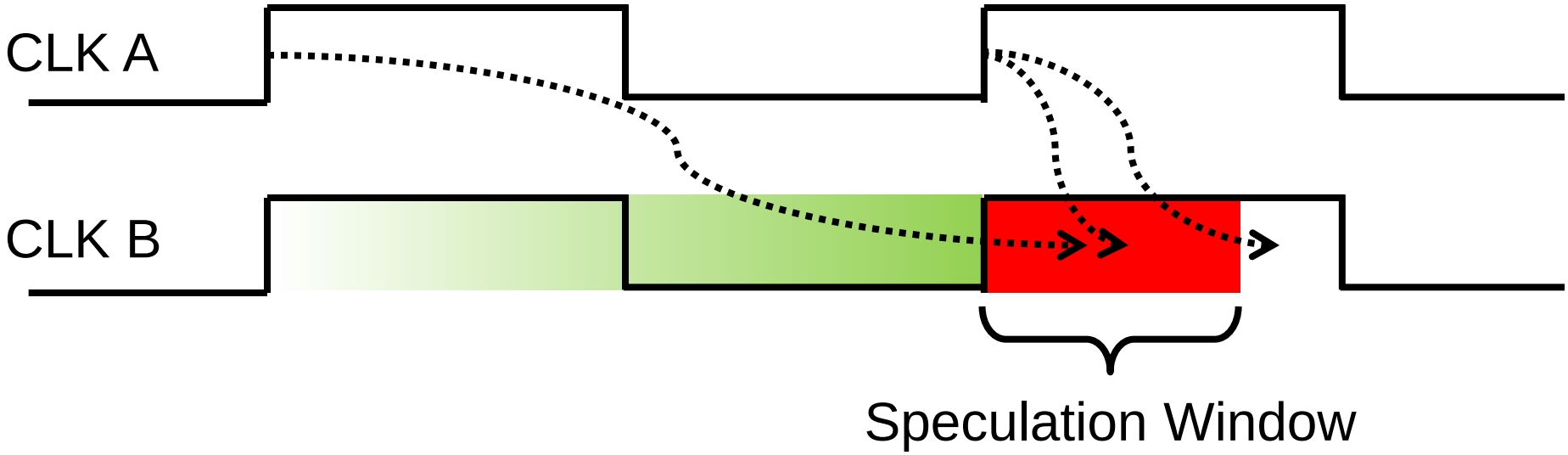
- Always Correct
  - Tables, Canaries
- Detect and Correct
  - Razor Style



Technique	Process		Ambient				Data
	Global	***Local***	***Global***		***Local***		
			Slow	Fast	Slow	Fast	
Table Lookup	X	X					
Table & Sensors	X	X	X				
Canary Circuit	X		X				
Razor Designs	X	X	X	X	X	X	X

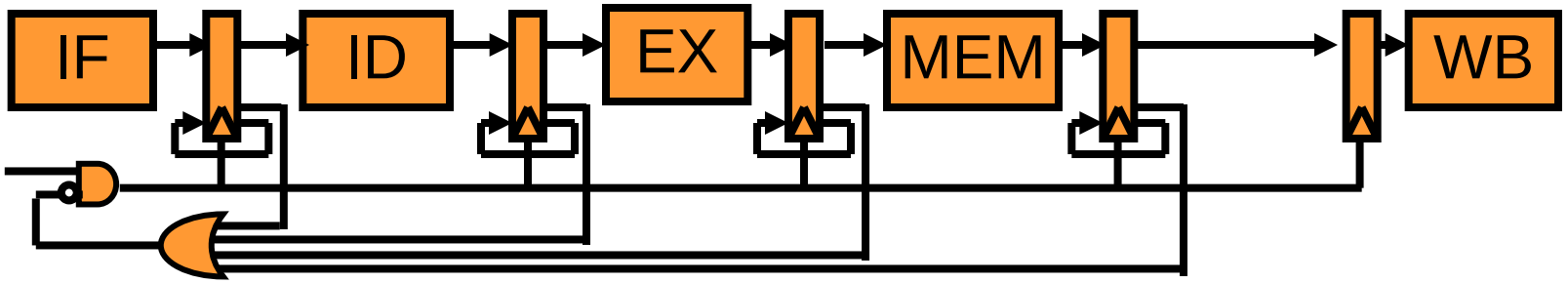
Speculation Window and Hold Time





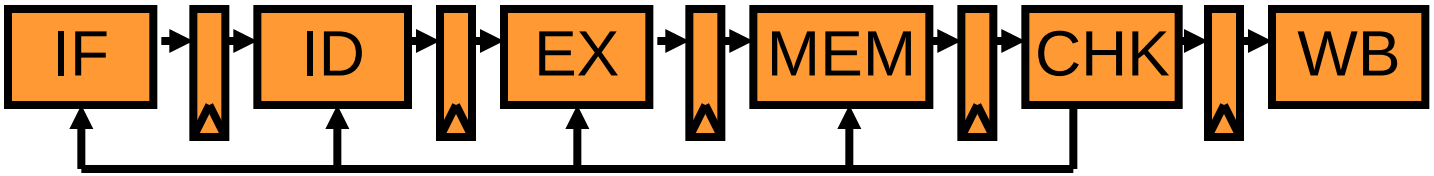
Speculation window linked to minimum delay constraint (hold time)

Architectural Invasiveness



S. Das, et. al. [VLSI 2005]

Razor I Style – All Flops Reload Previous Values



D. Blaauw, et. al. [ISSCC 2008]

K. Bowman, et. al. [ISSCC 2008]

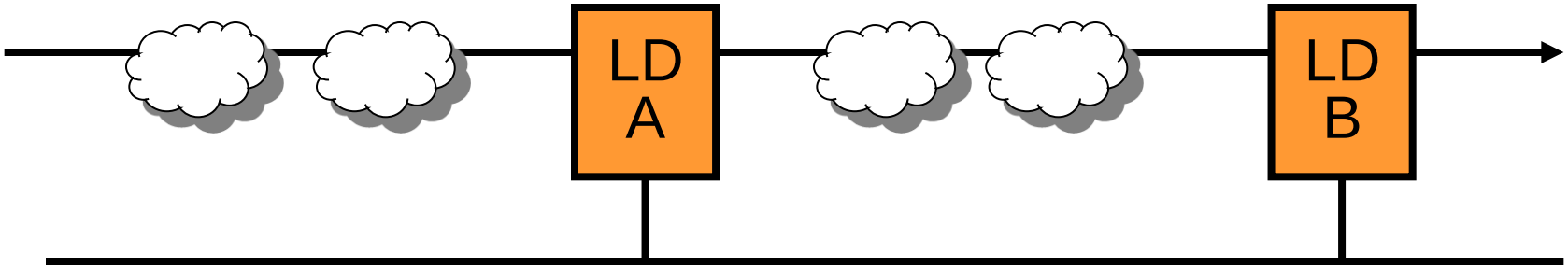
Razor II Style – Check Stage and Architectural Replay

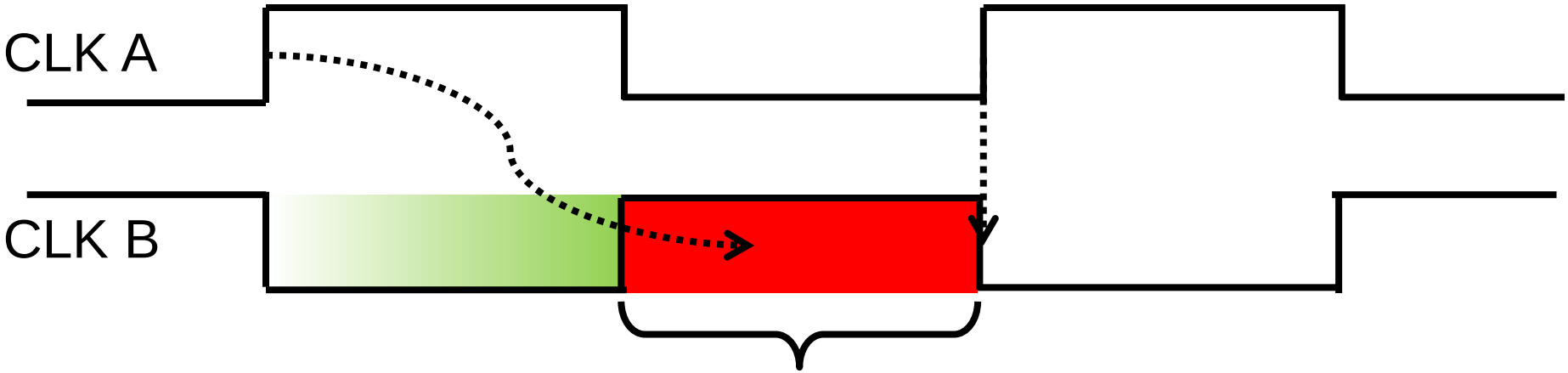
- **Requires Designer Effort**
- **RTL written with Razor in mind**

Fundamentals of Bubble Razor

- Two-Phase Latch Timing
  - Automatically convert Flip-Flop based design
- Time Borrowing as Correction Mechanism
  - Does not modify design architecture
  - Does not require reloading / replaying instructions
- Local Correction (Bubbles)
  - Break requirement of stalling entire chip at once

Two Phase Latch Razor Timing

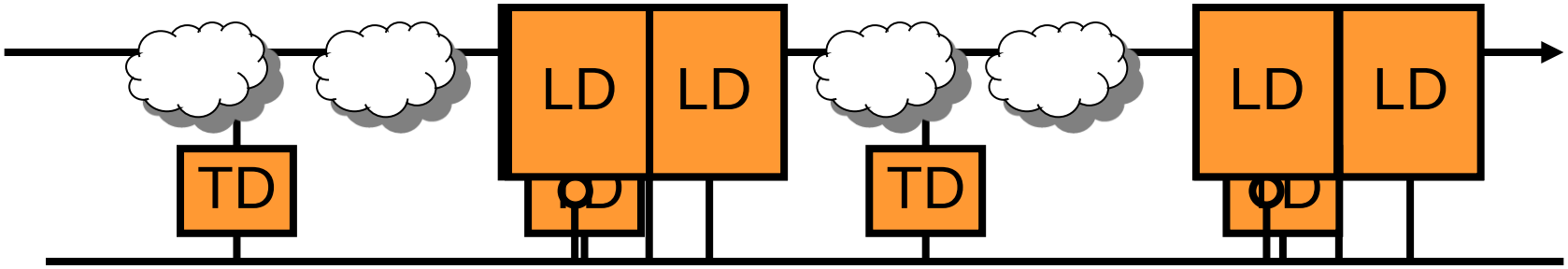




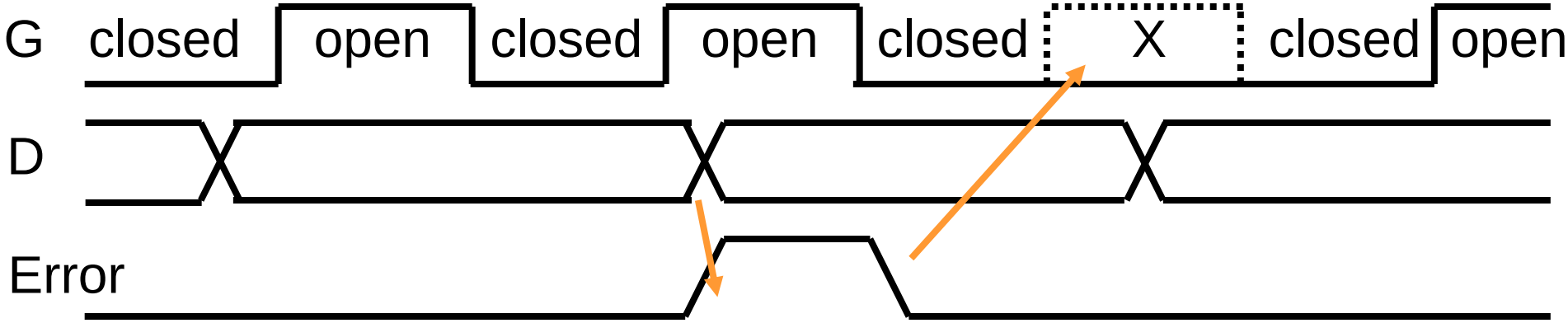
Larger Speculation Window

Minimum delay constraint the same as conventional design

Time Borrowing as Error Correction

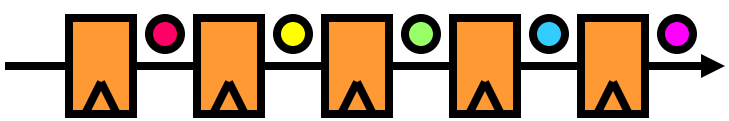


Bubble Razor – Switch to Latches, Borrow Time



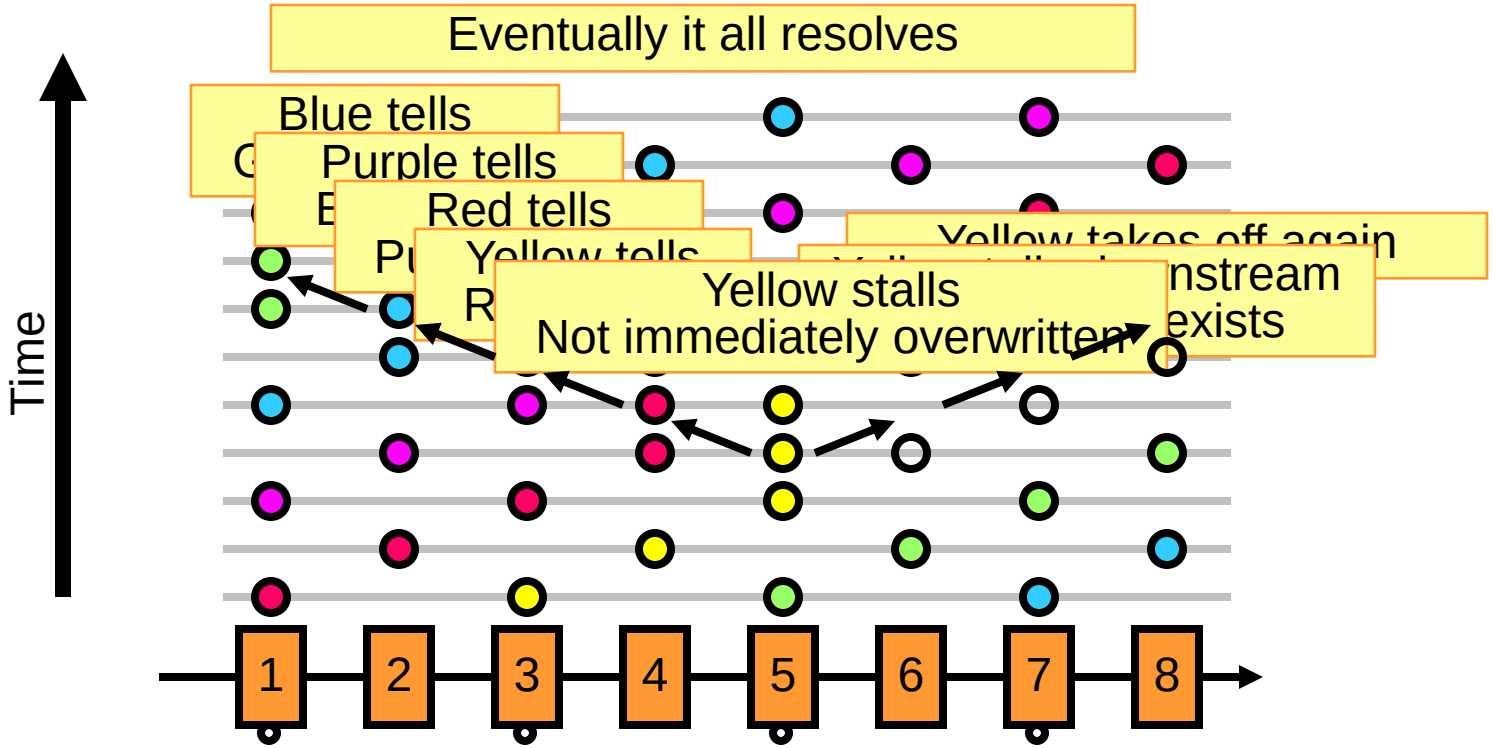
- No Hold Time Issues
- Architecture Agnostic
- Push-button approach
- No metastability on datapath

Stalling Locally with Bubbles

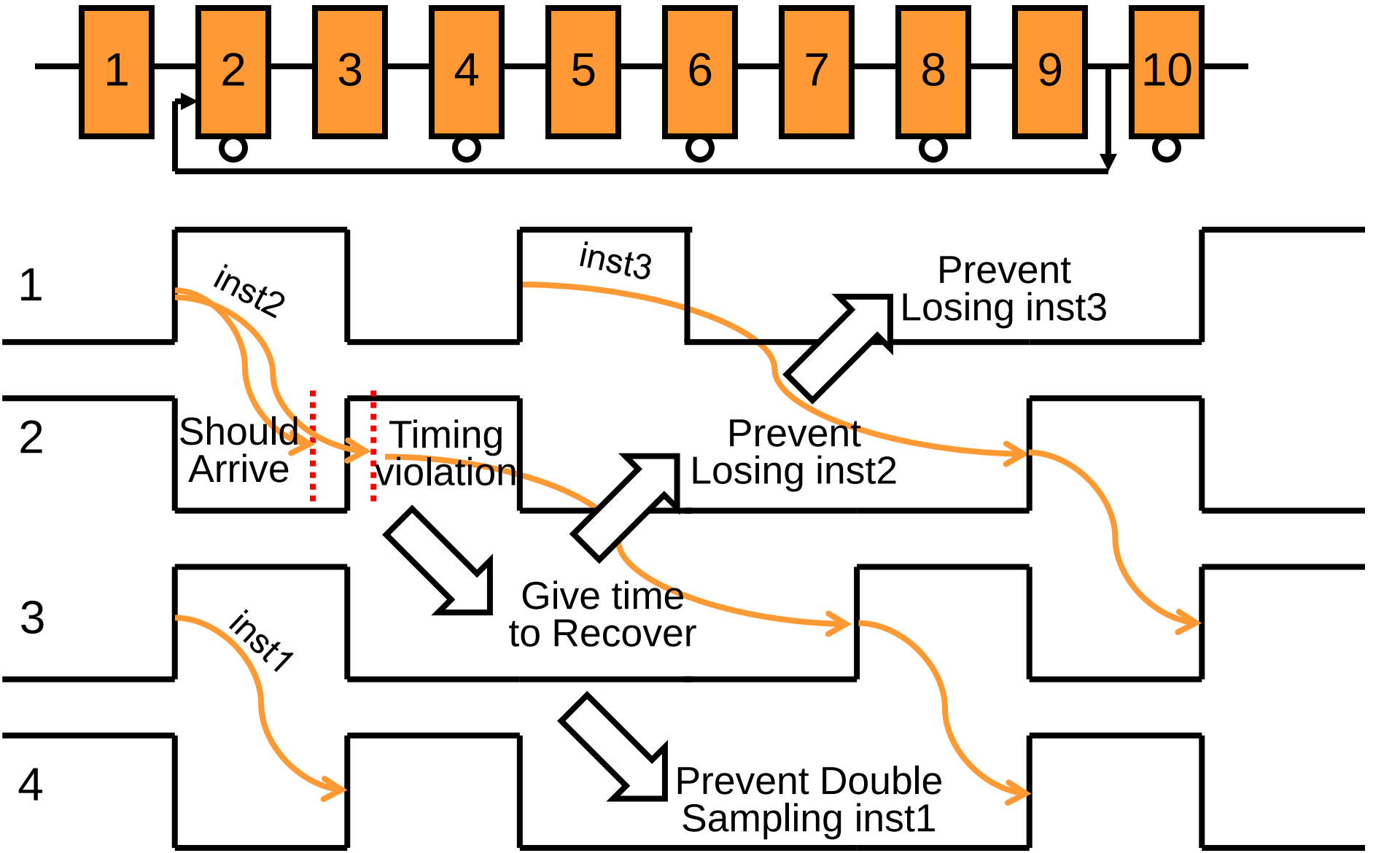


Stalling the Clock Locally

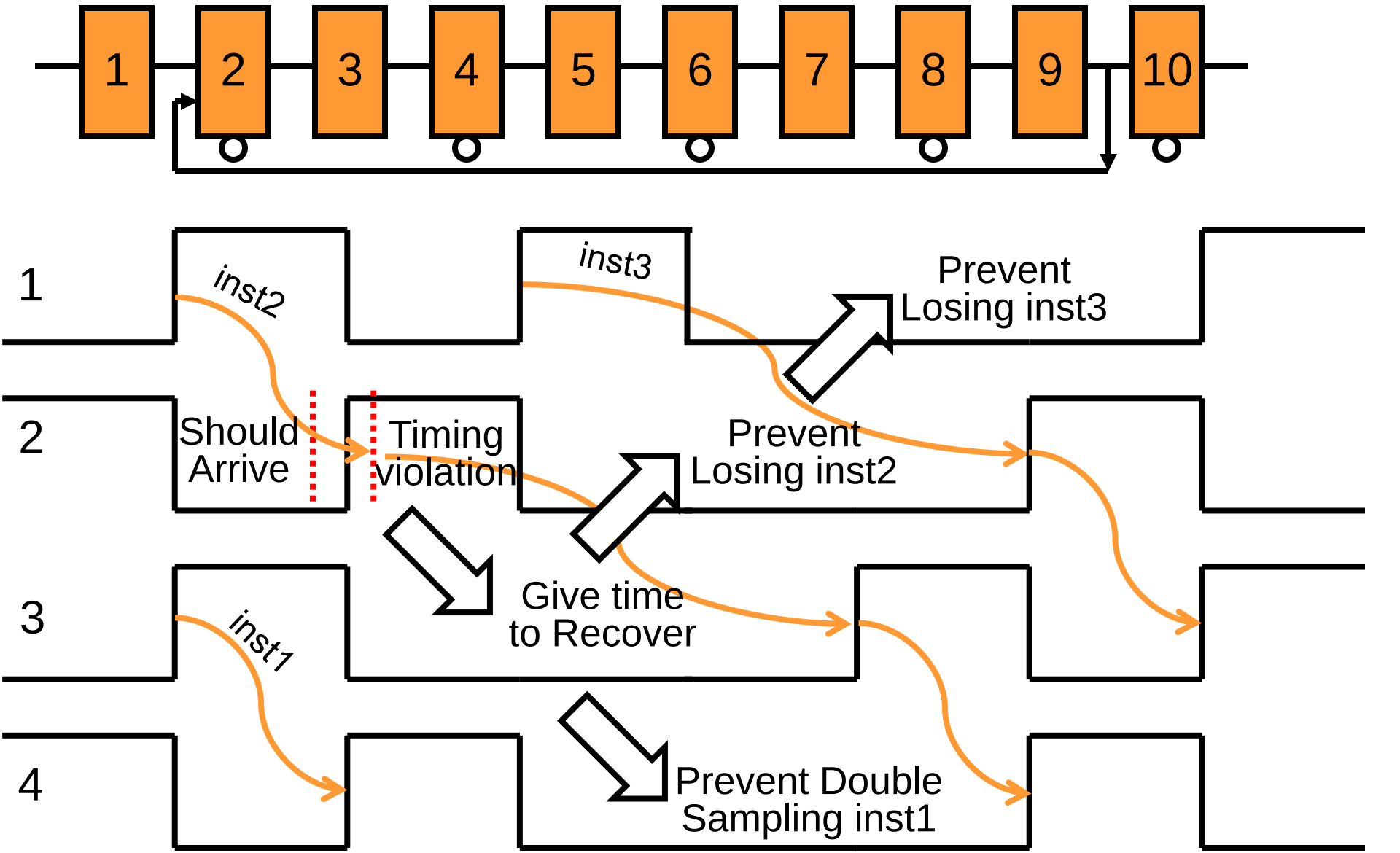
- With flops, all registers hold data
- With latches, half registers hold bubbles
- Every latch stalls exactly once
- Communication only between neighbors



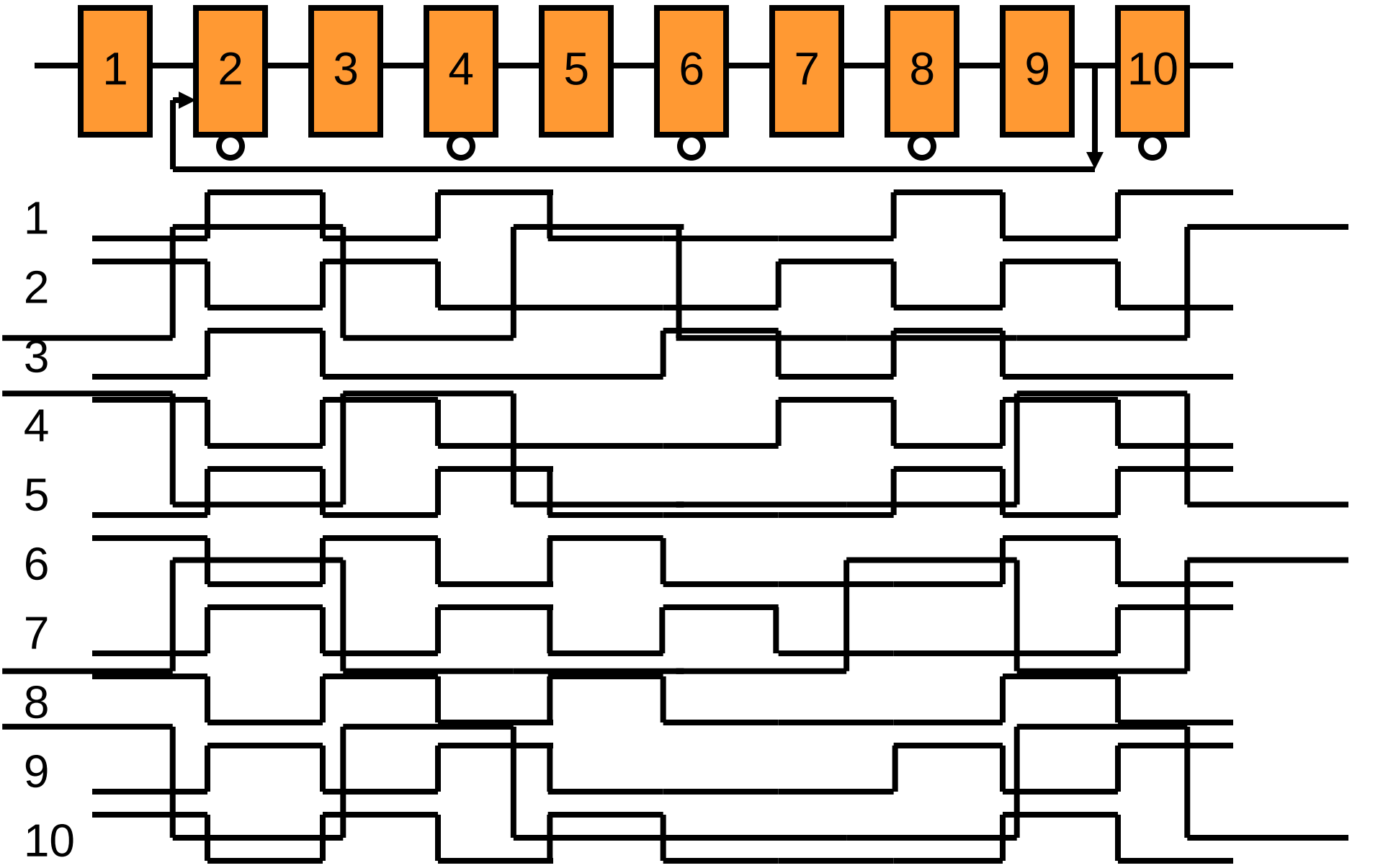
Timing of Clock Waveforms



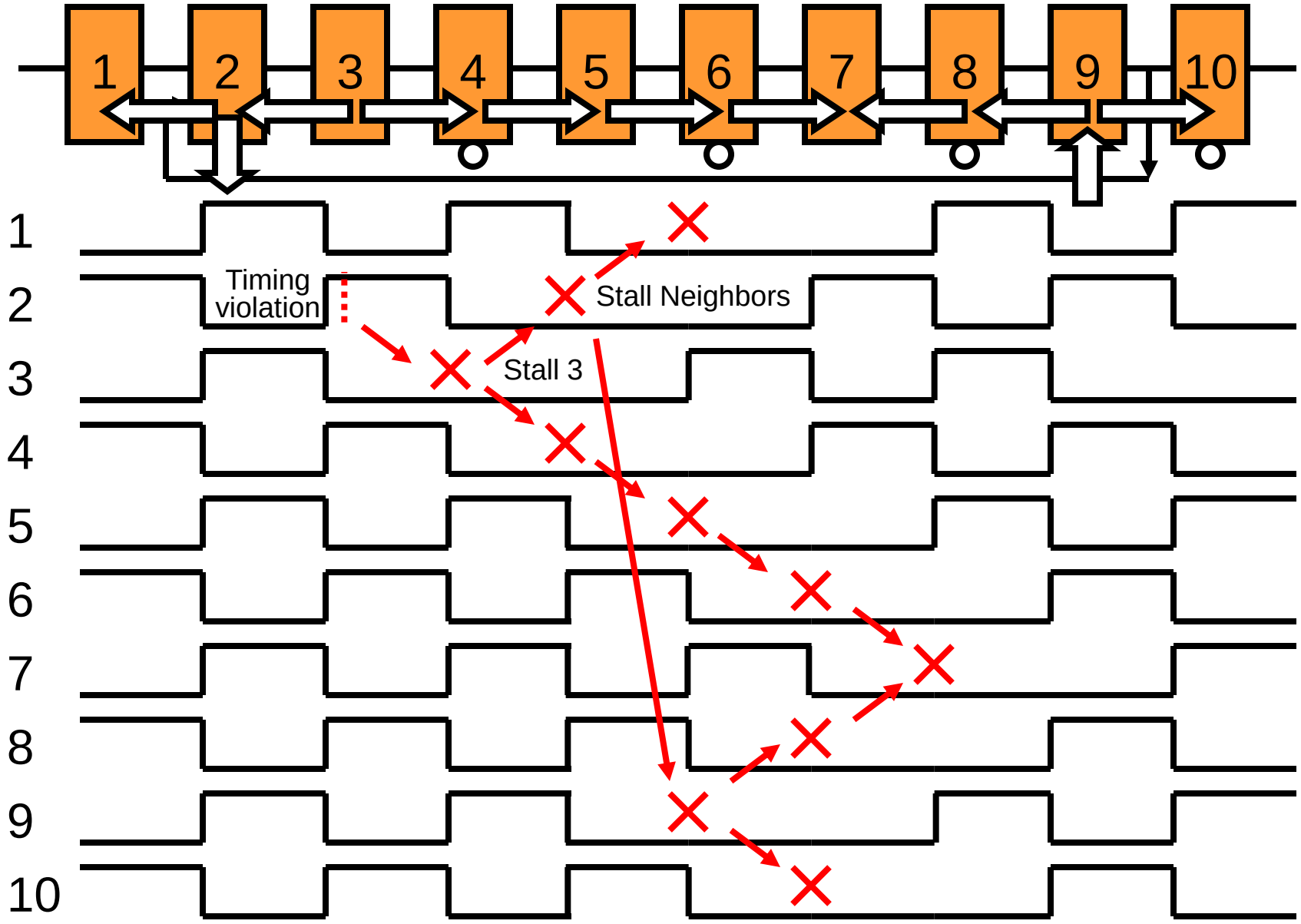
Timing of Clock Waveforms



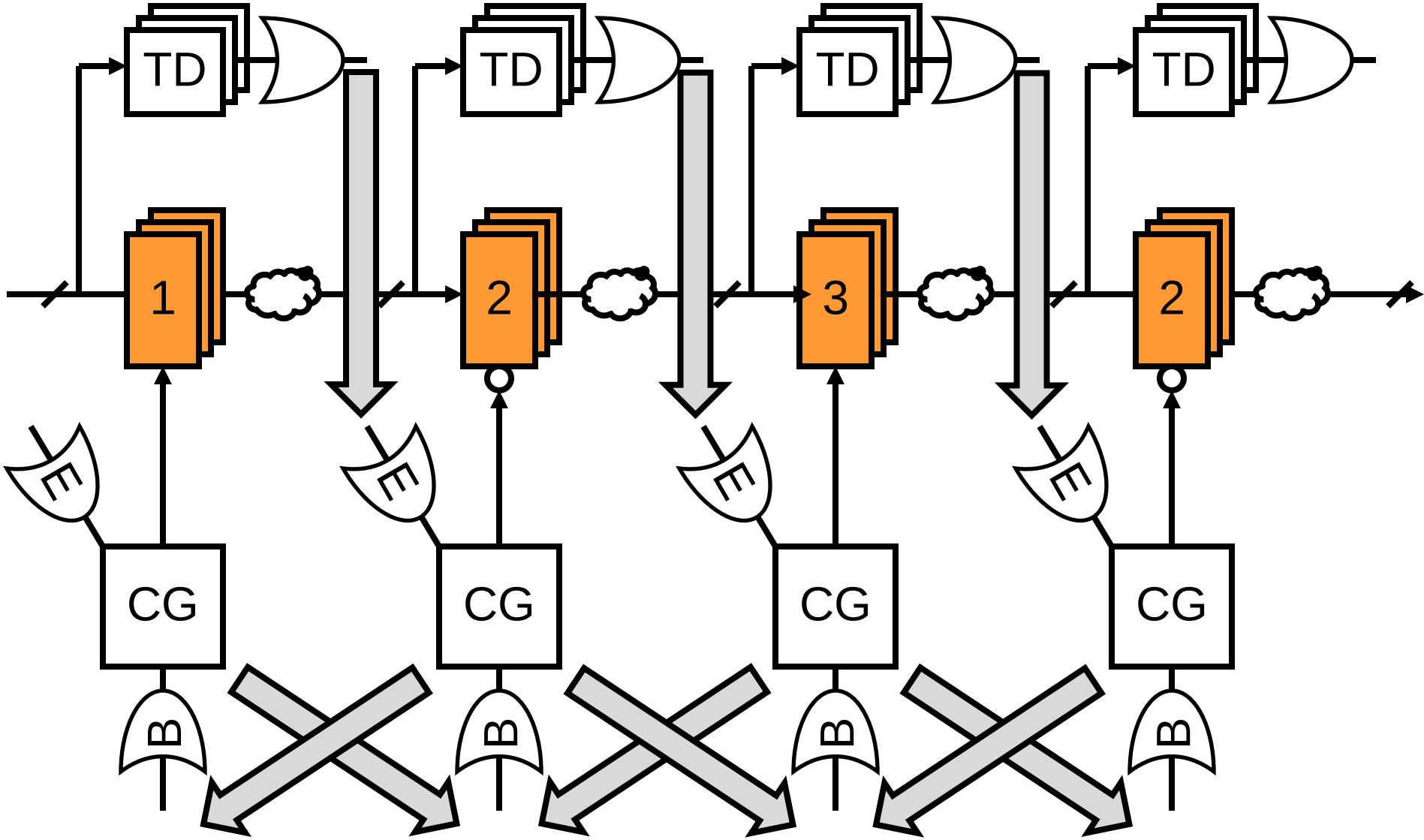
Timing of Clock Waveforms



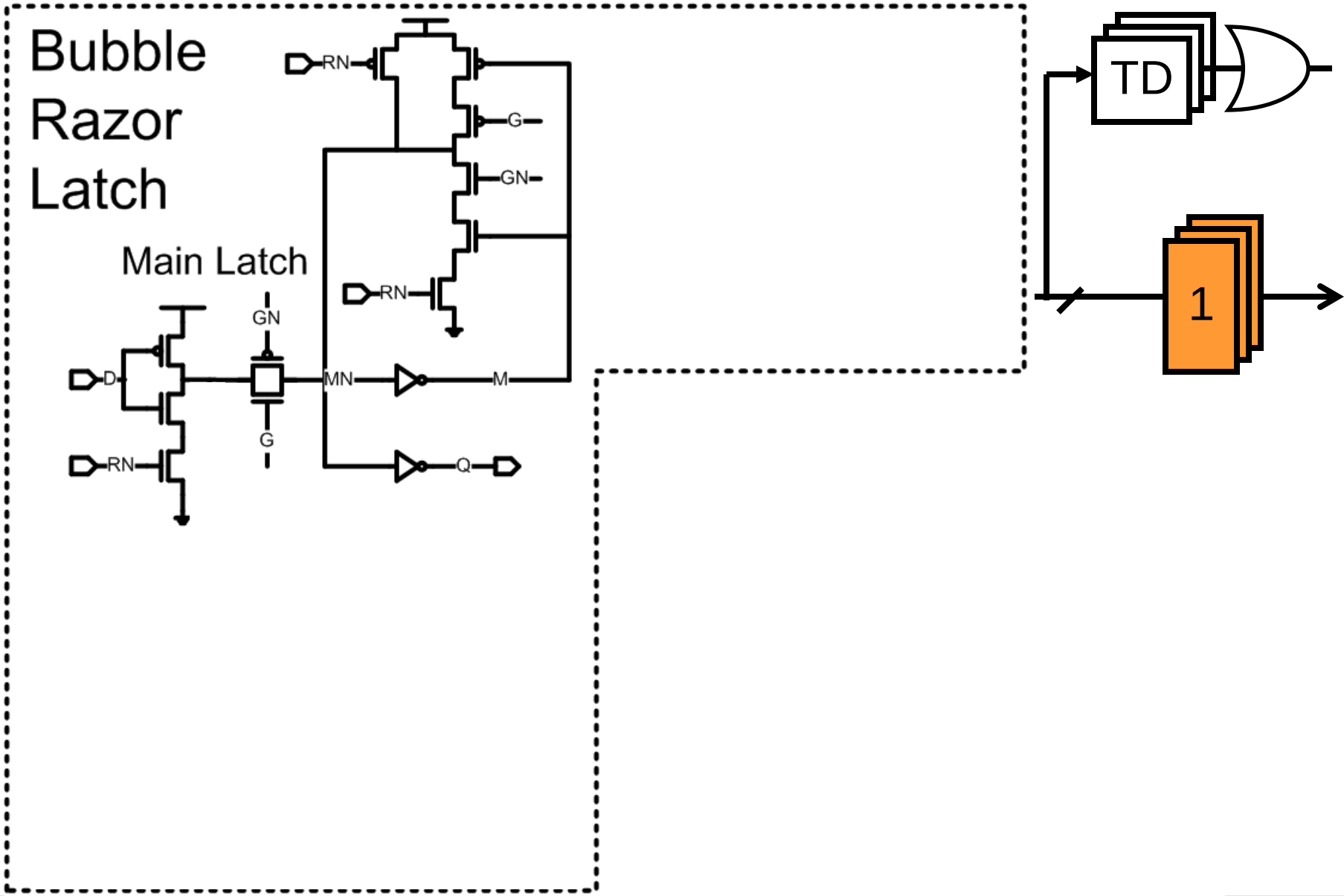
Timing of Clock Waveforms



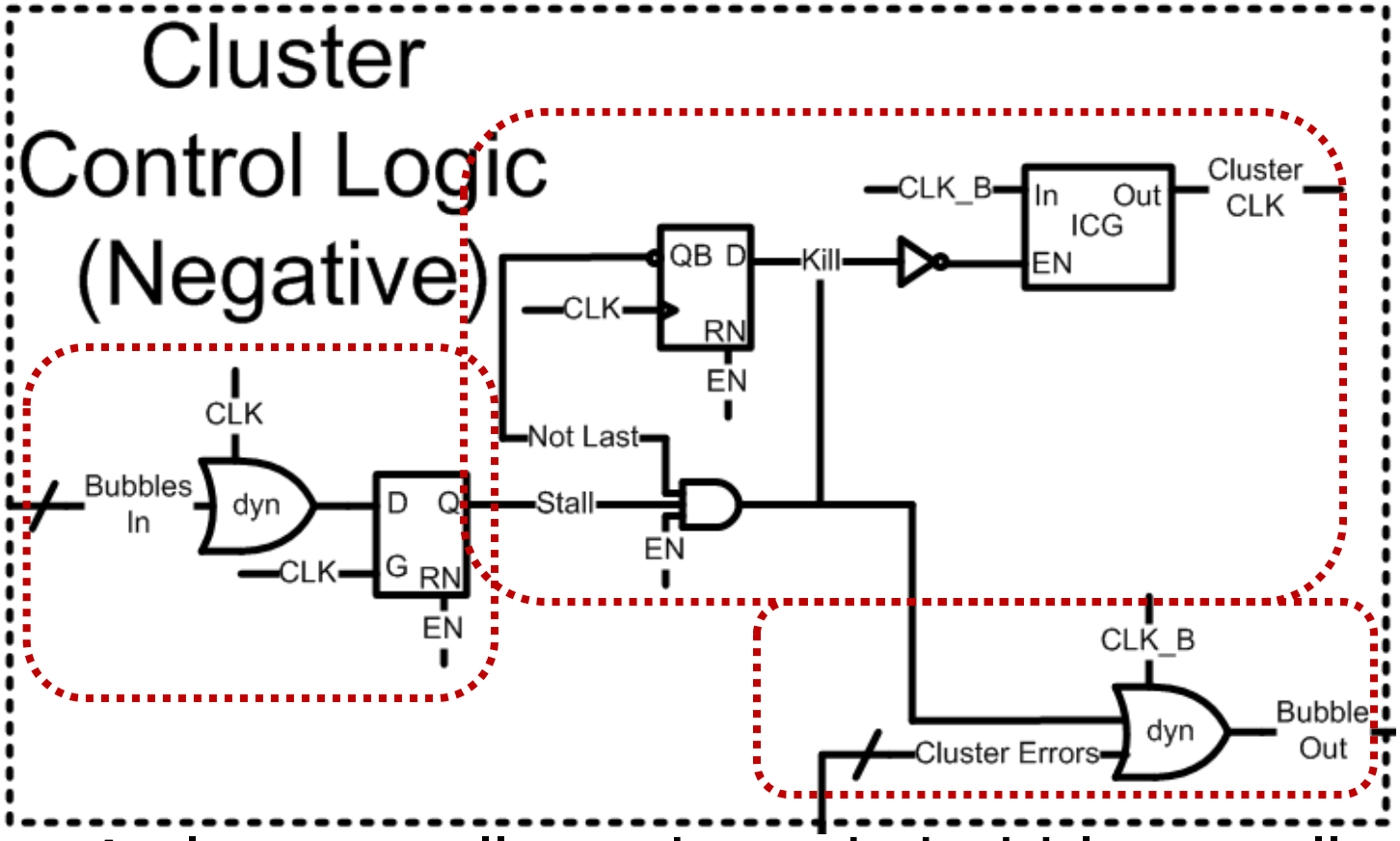
The Required Circuitry

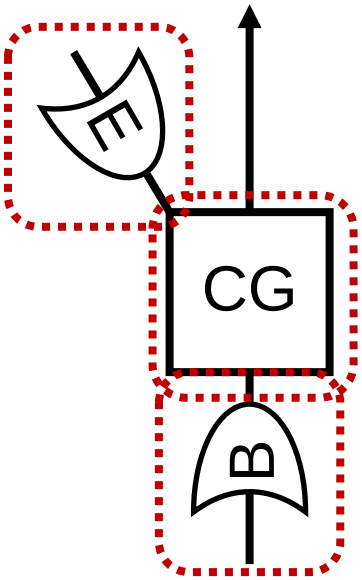


Error Detection And OR Circuitry



Clock Gate Control Logic

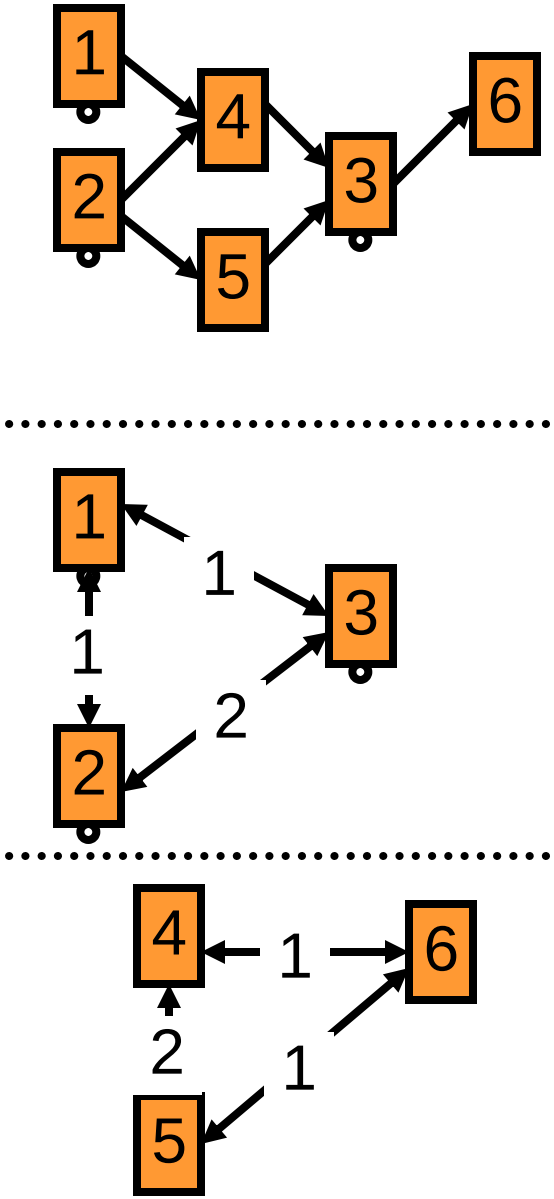




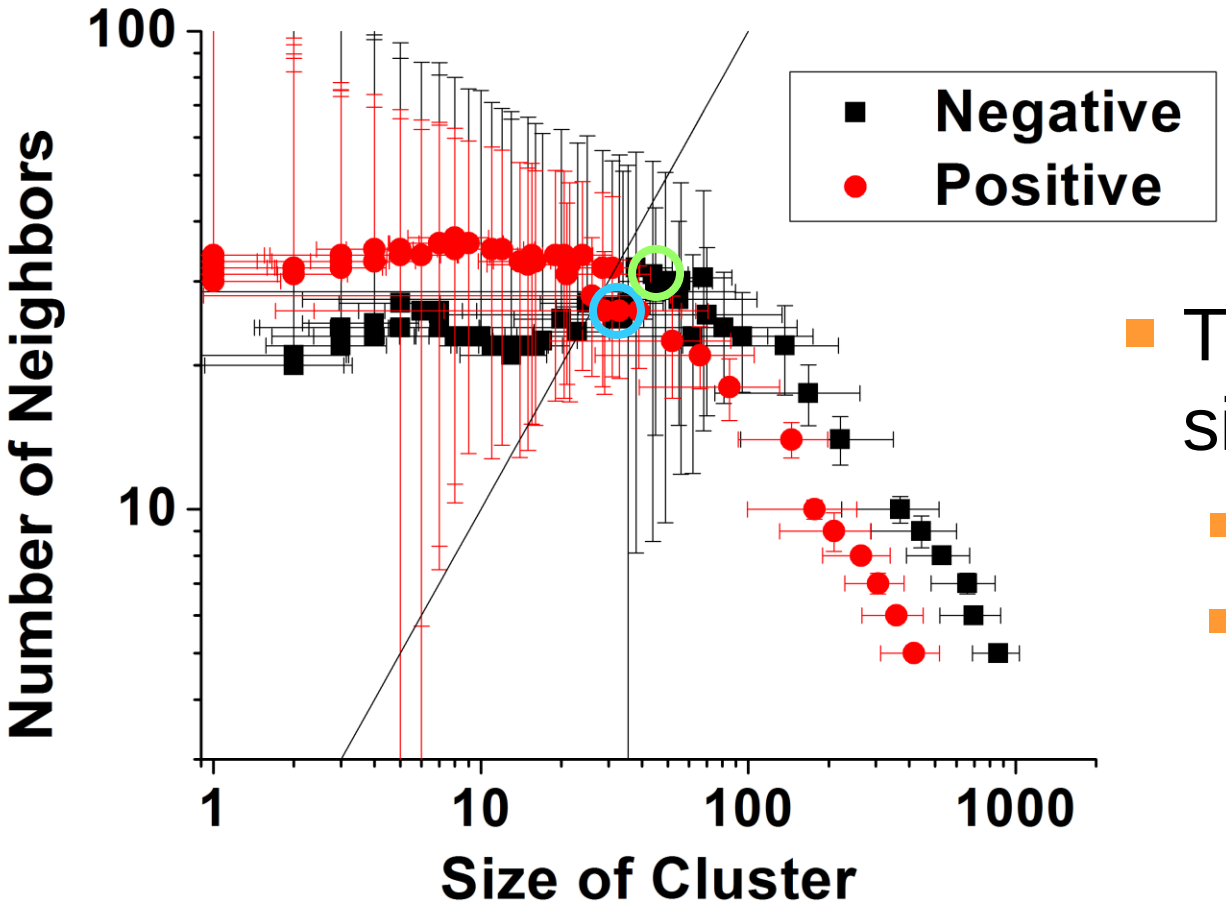
- A cluster stalls and sends bubbles to all neighbors if
  - Told by a neighboring cluster
  - Did not stall in the previous cycle
- Equivalent to sending bubbles to “other” neighbors

Clustering with hMETIS

- Widely used Hypergraph partitioning program, hMETIS
- Clusters must only contain members with the same phase
  - Create two graphs, and partition independently
- Connected in hMETIS graph, if transitively connected in circuit
  - Edge Weight = number of latches that form transitive connection



Clustering Results

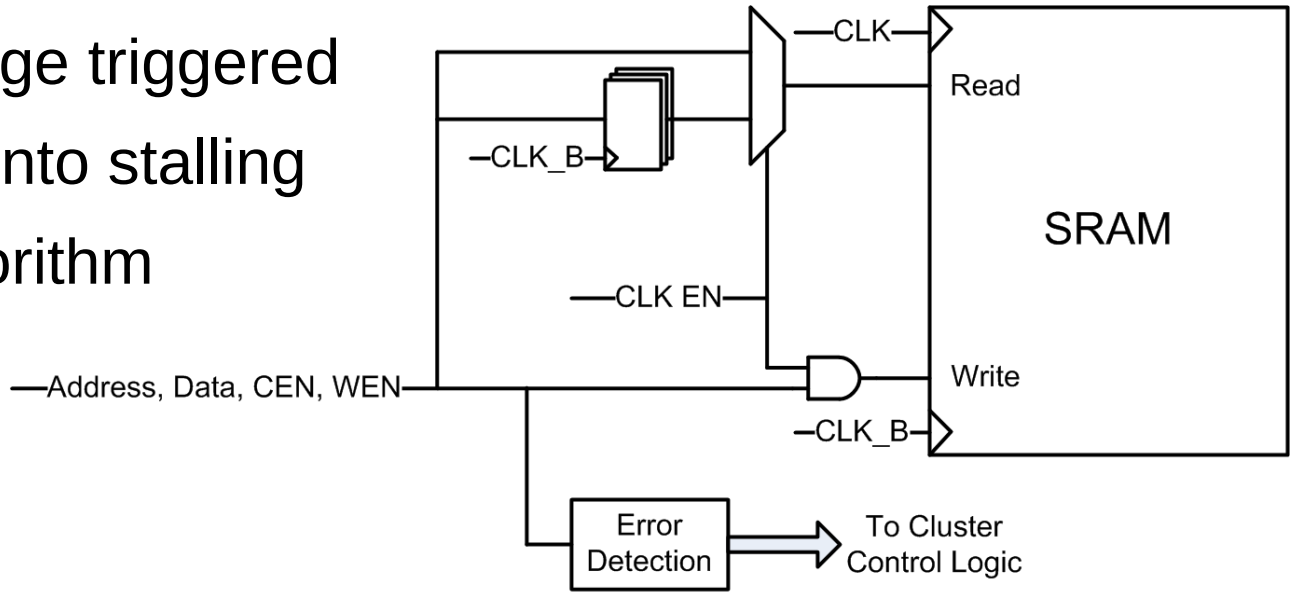


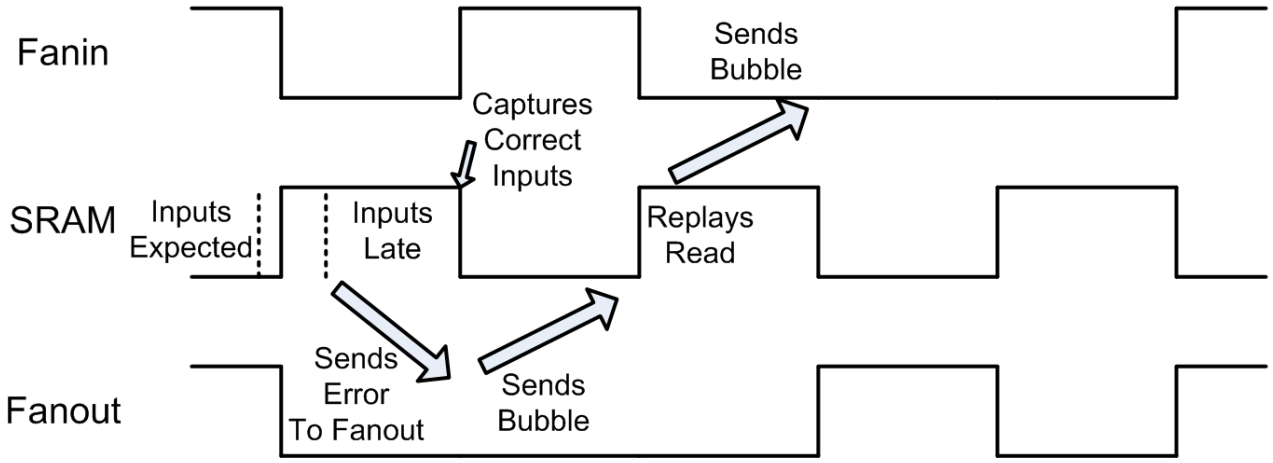
- Tradeoff between sizes of OR gates
  - Combining errors
  - Combining bubbles

- 100 negative clusters
- 70 positive clusters

Two Port Memory Boundary Approach

Must fit edge triggered memory into stalling algorithm

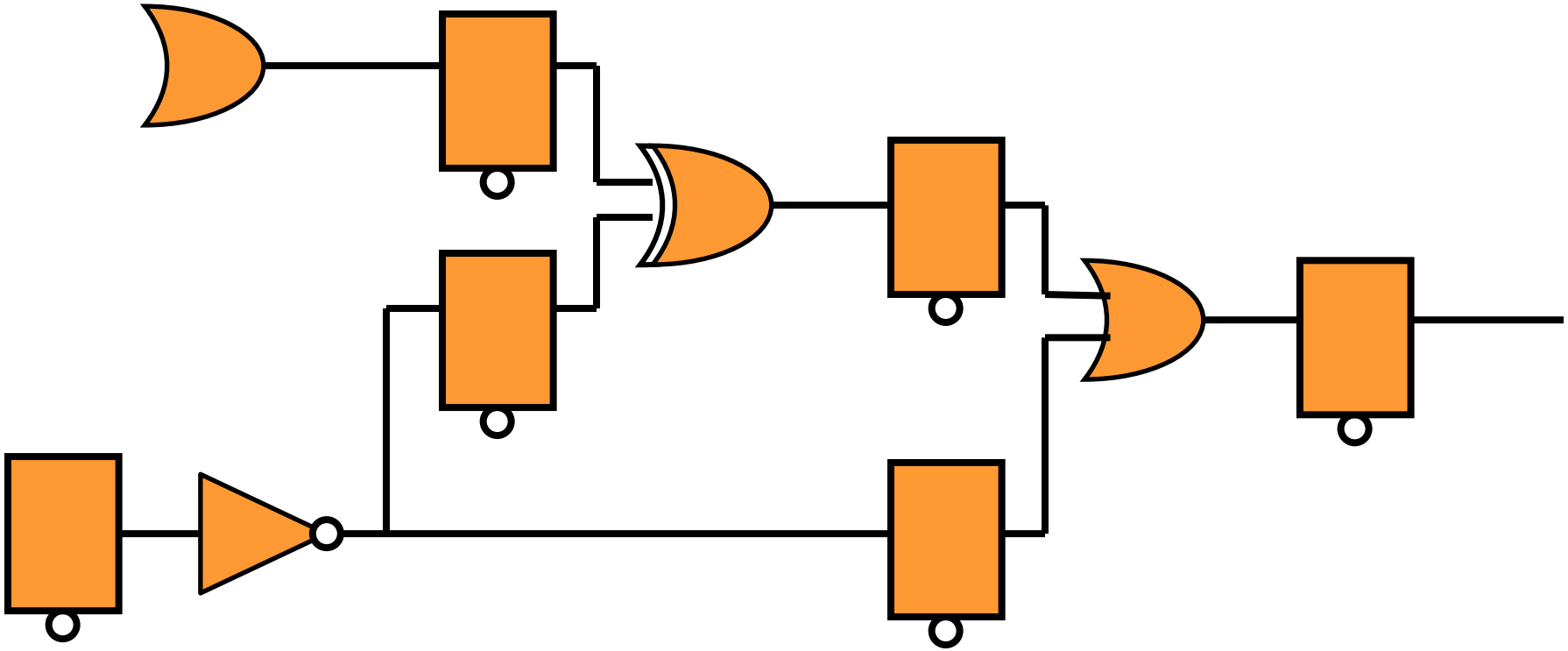




“Managing” the Synthesis/APR Tools

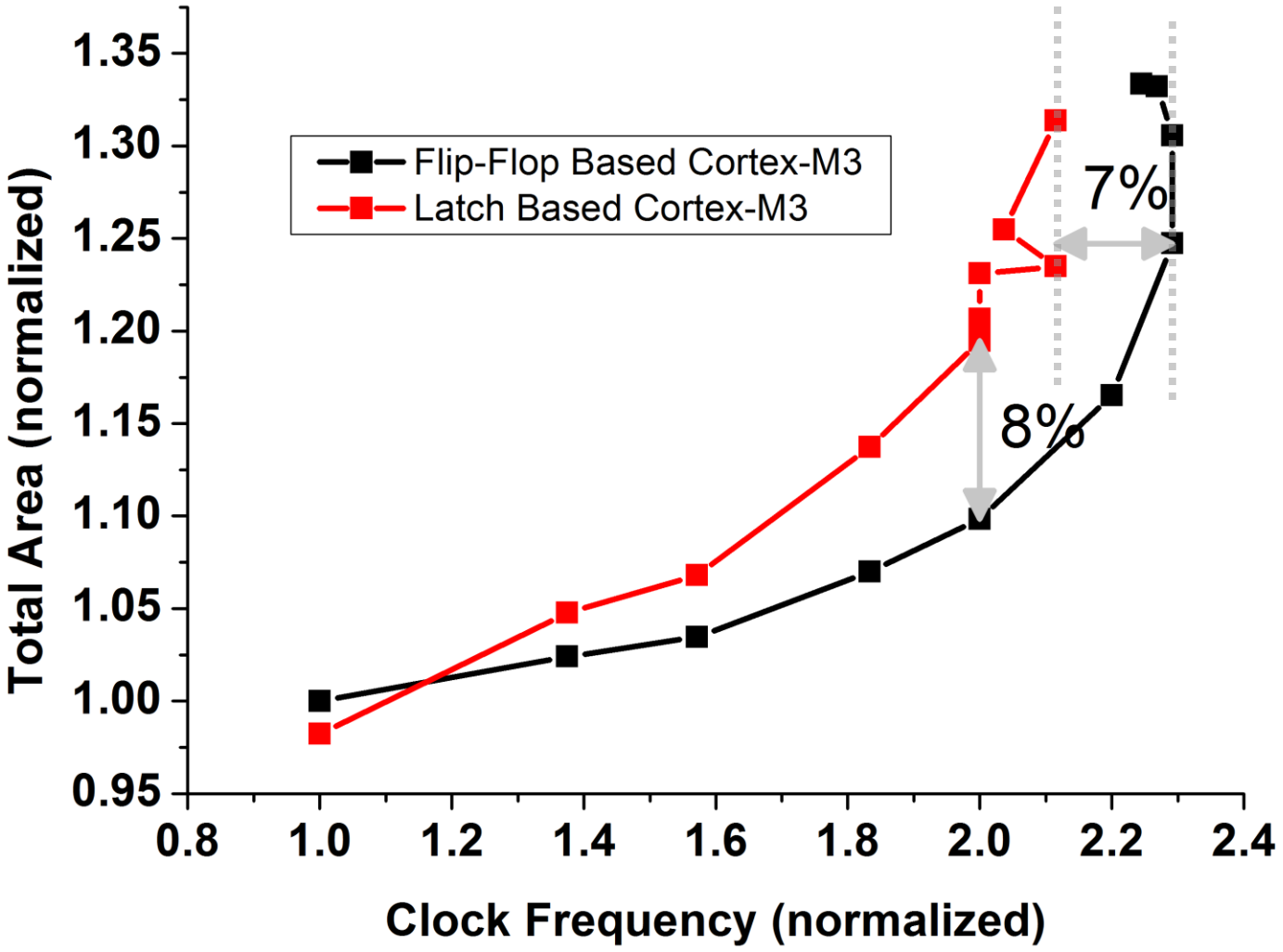
- Want balanced pipelines, no time borrowing
  - Model razor latches as flip flops
- Dynamic OR always followed by latch
  - Model dynamic OR as static
  - Model latch as flip flop (captures when latch closes)
- Use regular ICG cells
  - Can use conventional clock tree synthesis
- Final design appears to be relatively “normal”
  - Flip-flop based design with clock gating
  - Everything is timing constrained
- “Razorization” process is entirely automated
  - Synthesis and netlist transformation scripts

Retiming And Number of Latches



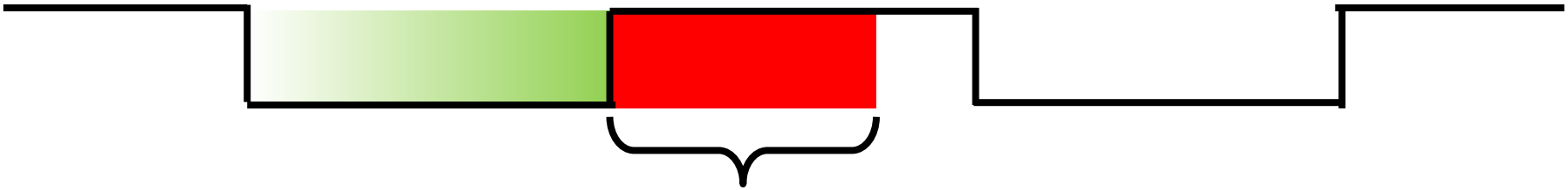
- Retiming can increase the number of latches
- Results in area overhead

Area Overhead of Latch Transformation

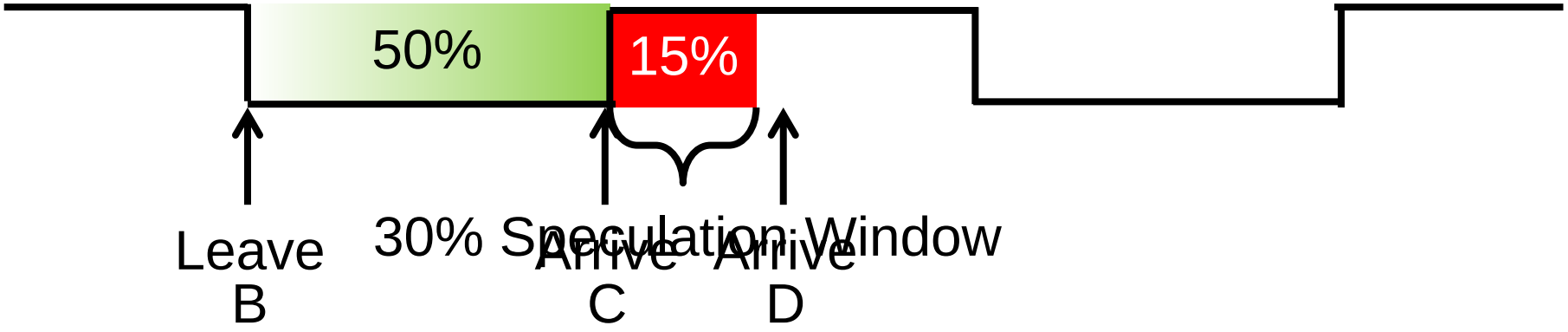


Speculation Window Size

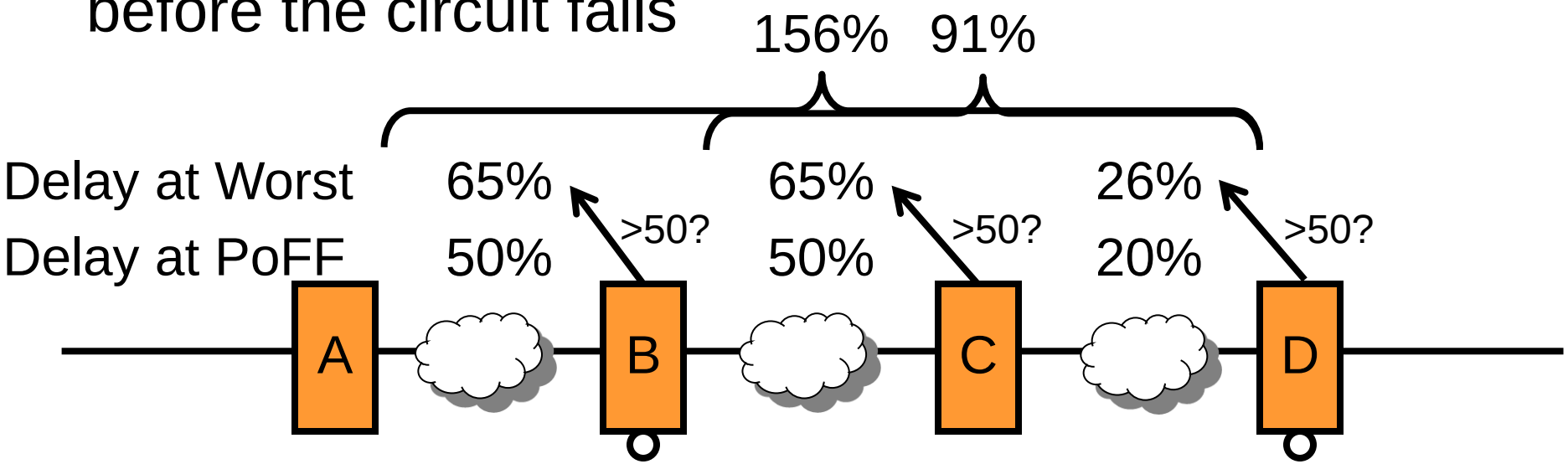
- Full Clock Phase (100%) Minus Delay of Error Propagation Circuits
  - Maximum allowed by technique
- Number / Location of Latches with Error Checking
  - Maximum slowdown that does not result in unchecked error



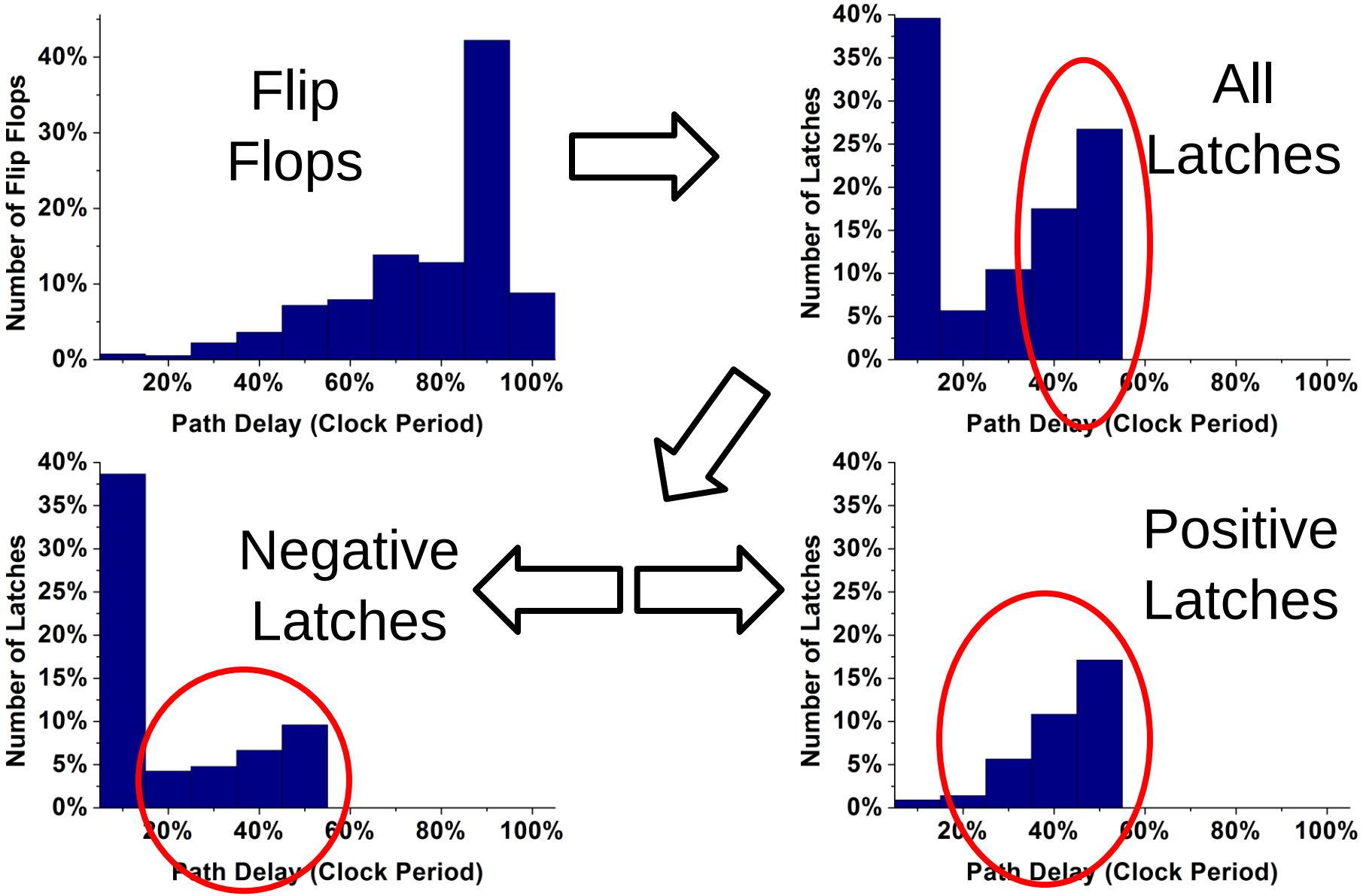
Where Error Checking is Needed



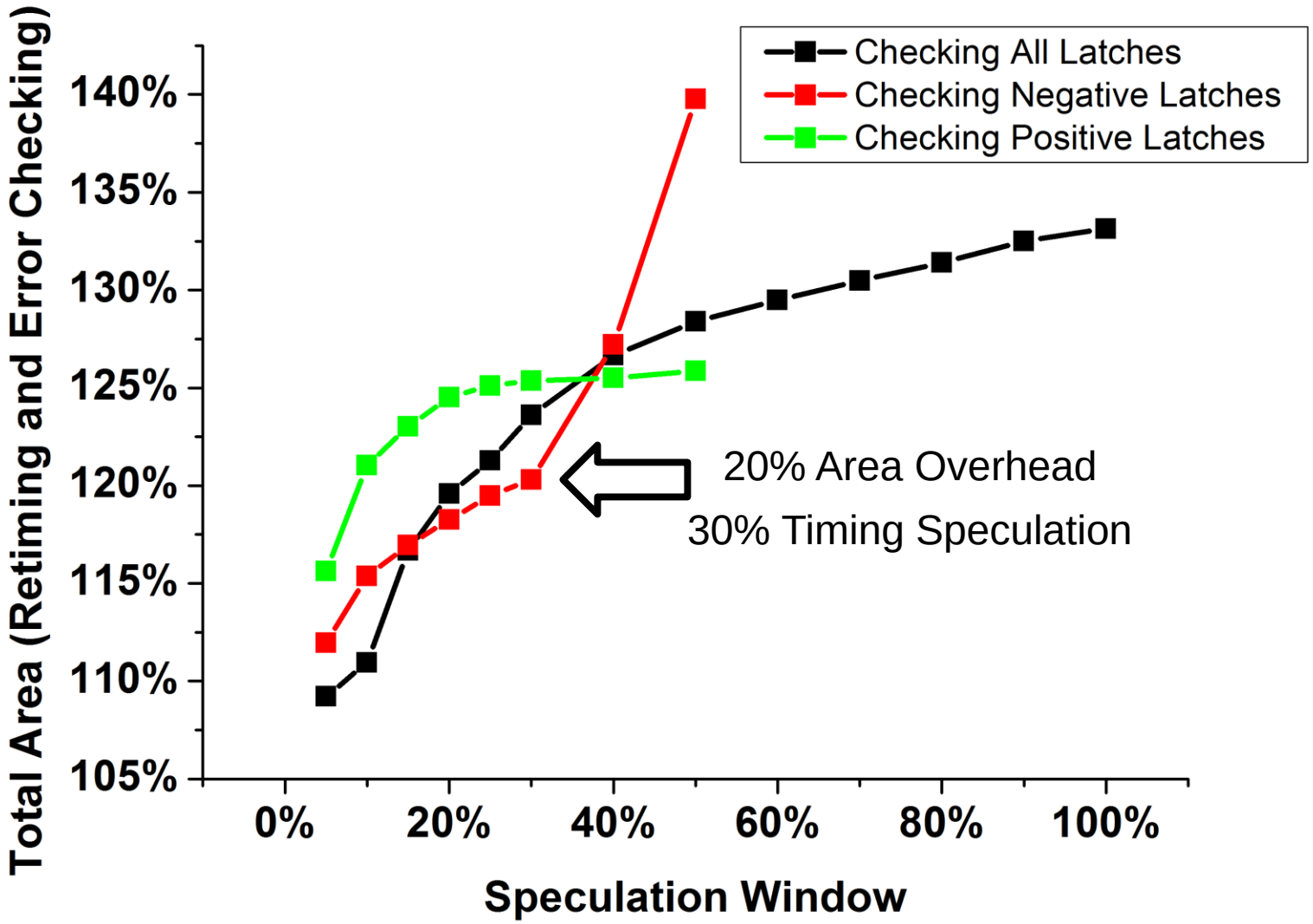
- If circuit delay suddenly becomes 130% of its nominal value, all timing errors will be detected before the circuit fails



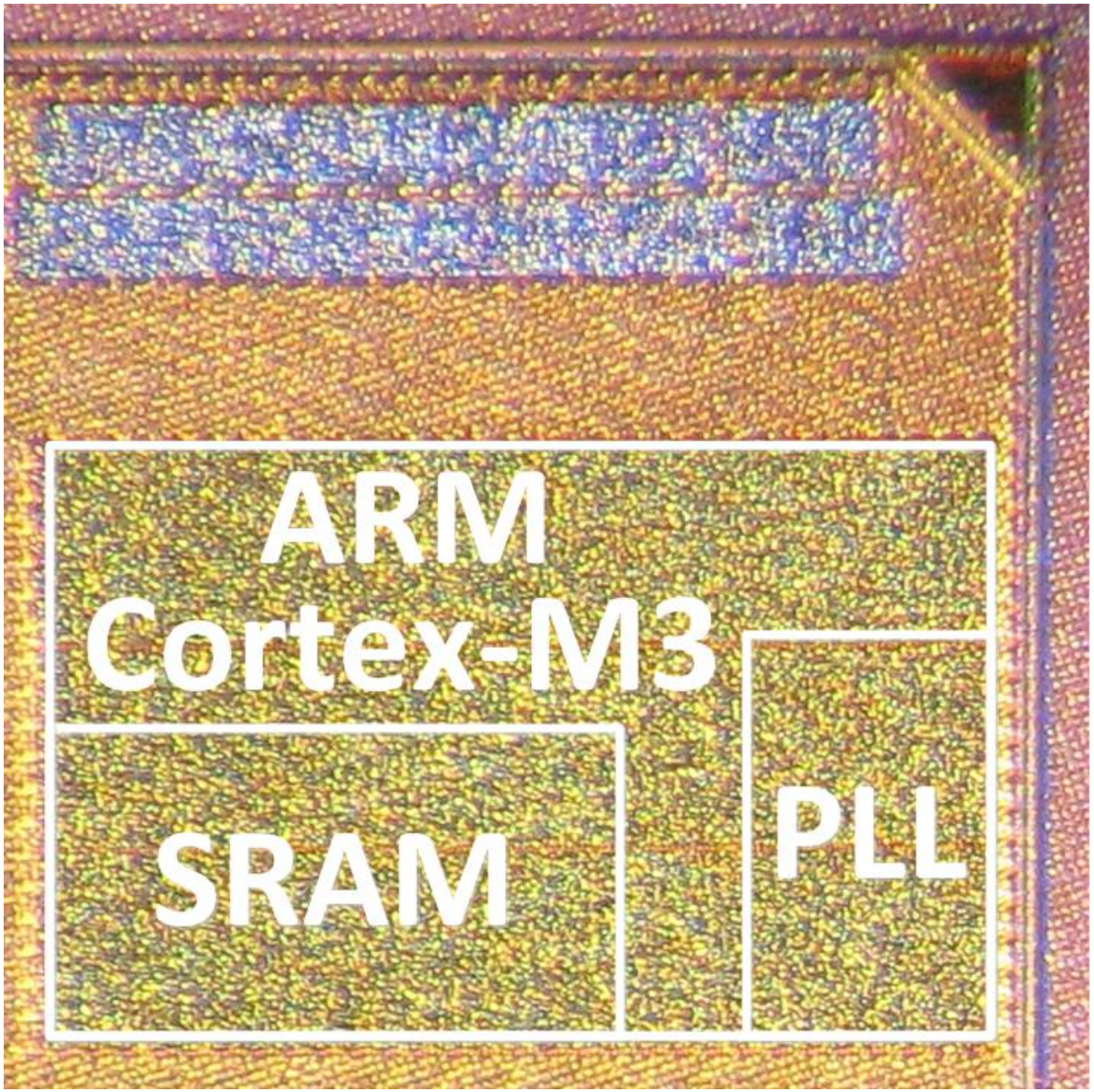
Path Distribution for Cortex-M3



Area Increase from Error Checking



Implementation on ARM Cortex-M3

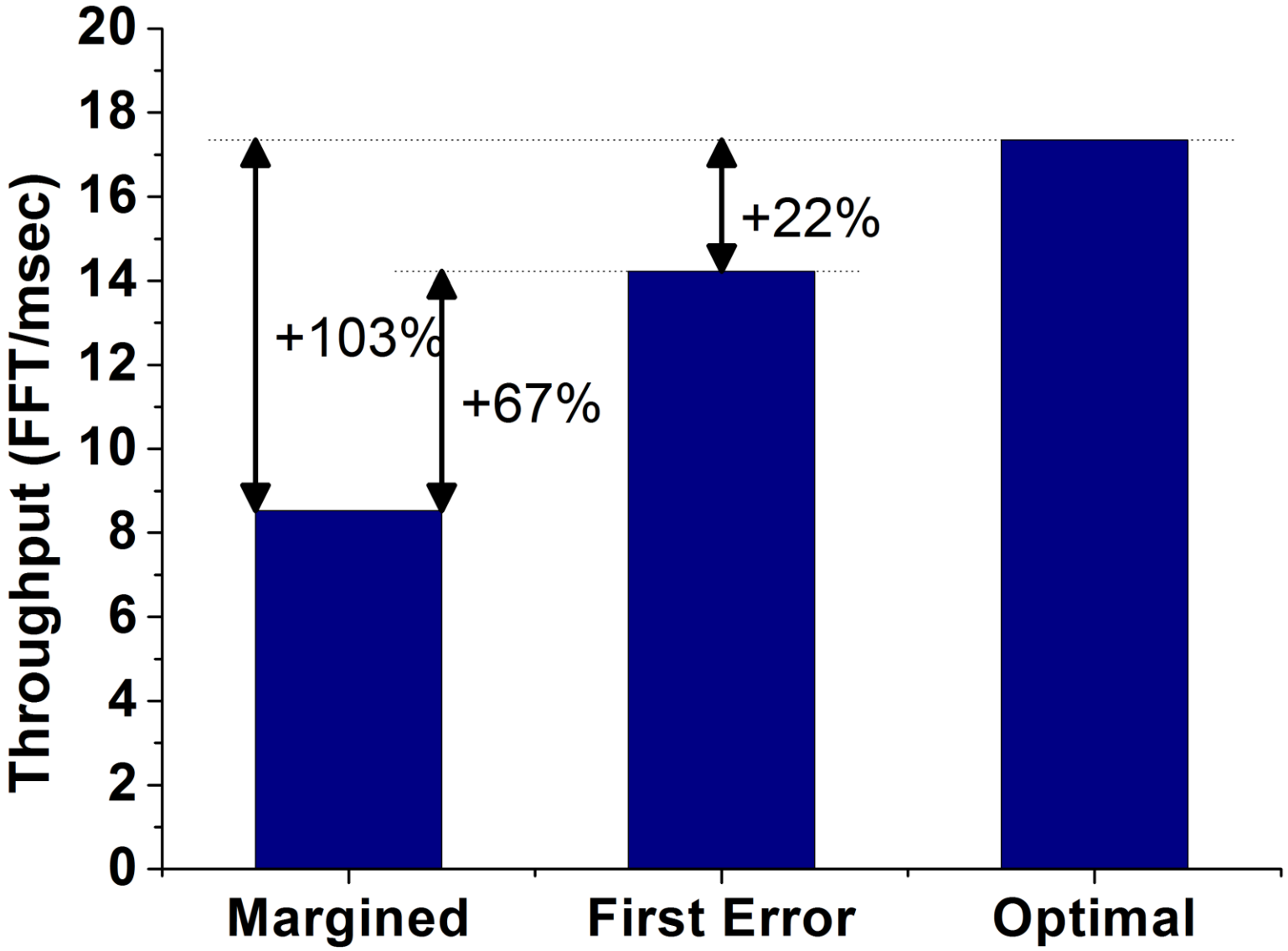


Processor Core	ARM Cortex-M3
Process Technology	IBM 45nm SOI12S0
Nominal VDD	1.0 V
SRAM Size	16 kB
Latches	7159
Positive Clusters	70
Negative Clusters	100
Speculation Window	55%

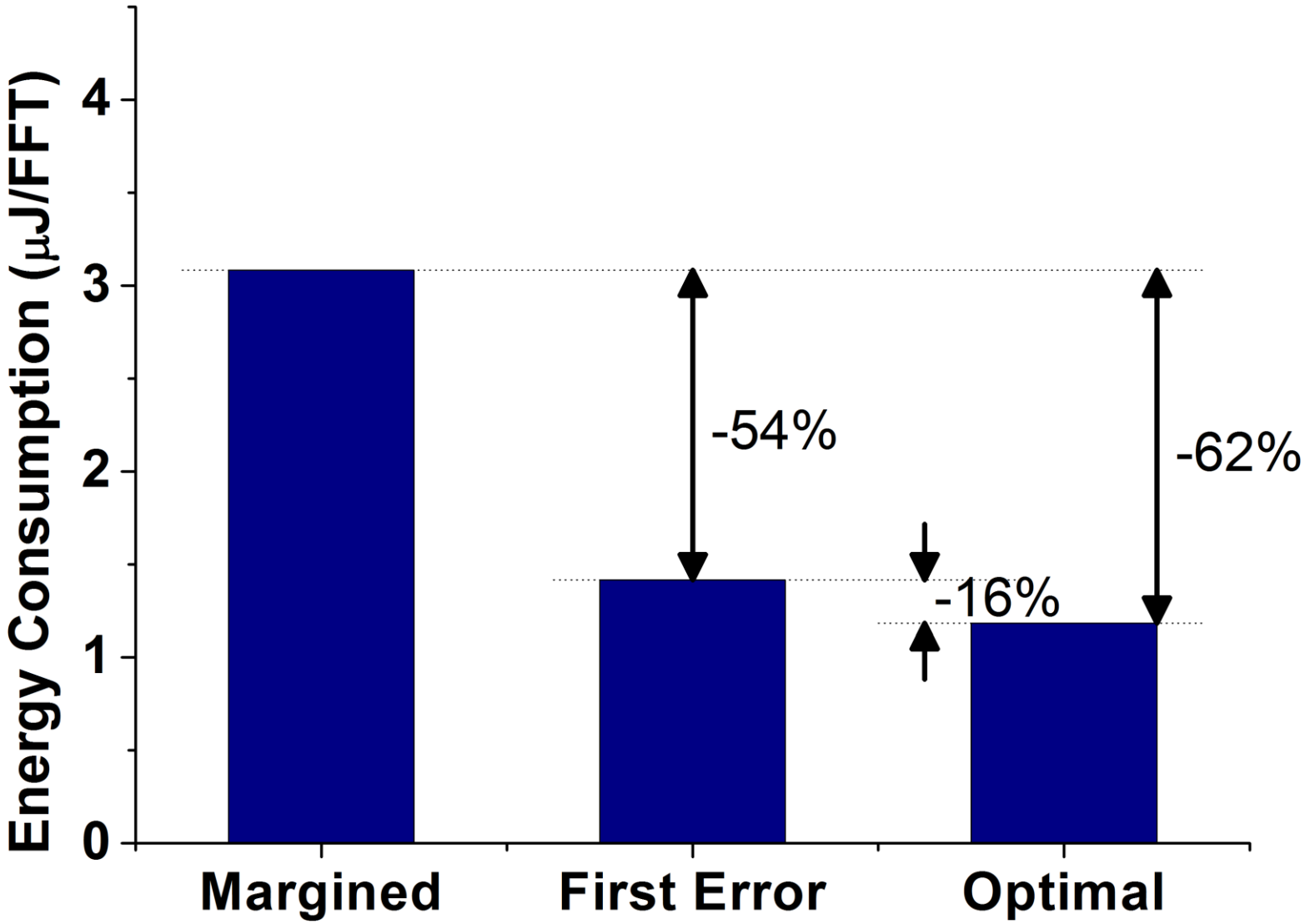
Characterizing Throughput / Energy

- Operating Point Set for Worst Case Operation
  - 85°C
  - 10% Supply Droop
  - 2σ Process
  - 5% Safety Margin
- 200 MHz at 1.0 V

Gains from Bubble Razor

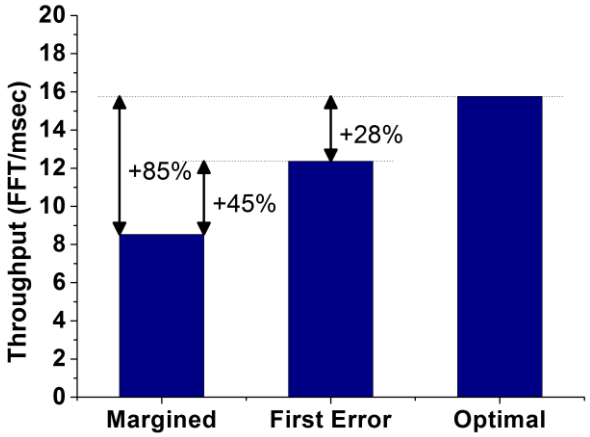


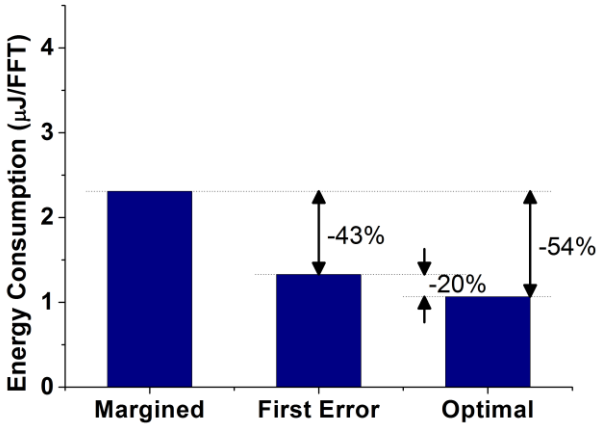
Gains from Bubble Razor



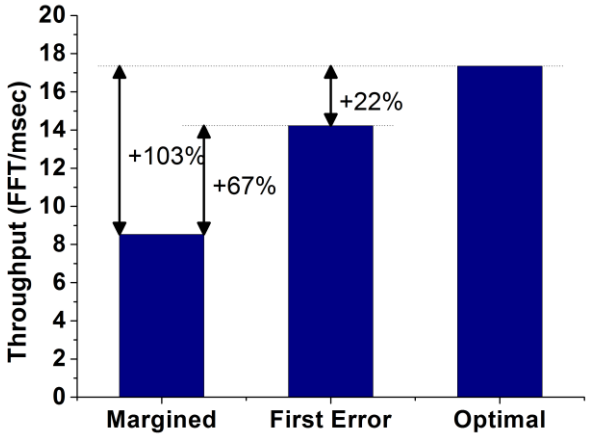
Bubble Razor Results

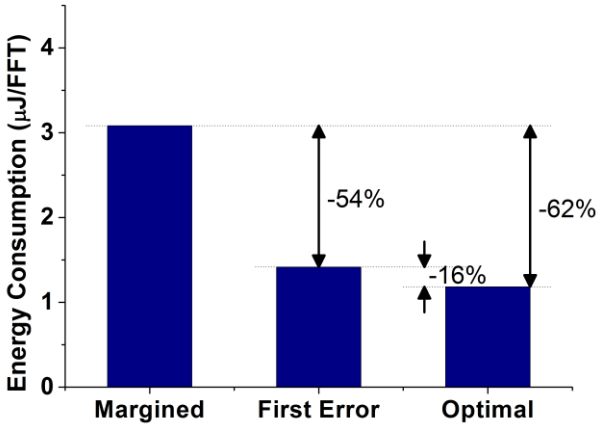
Slow



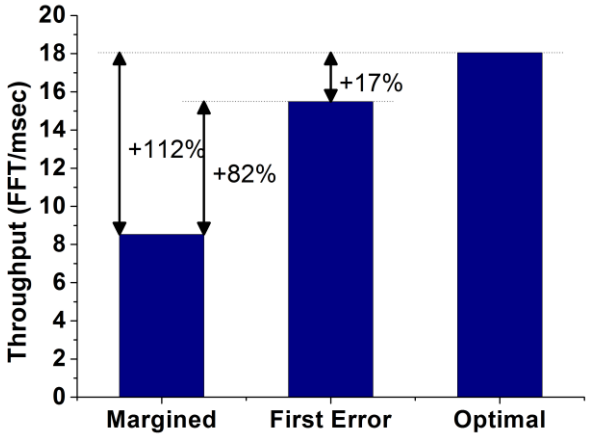


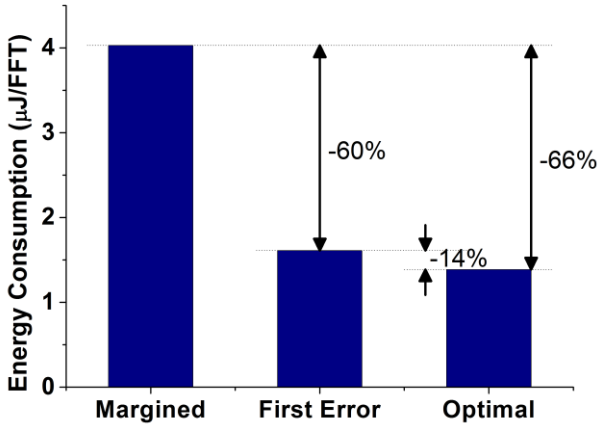
Average



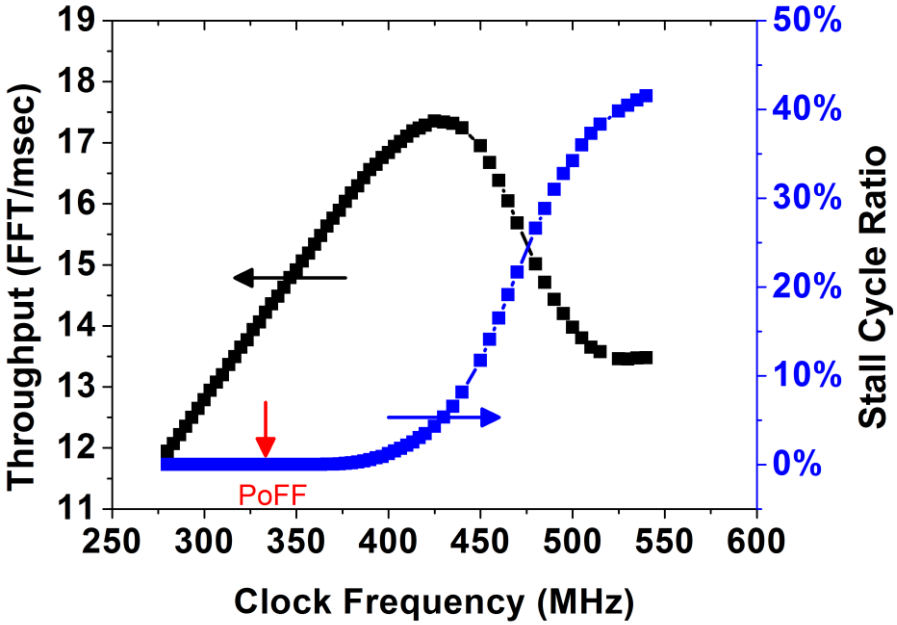


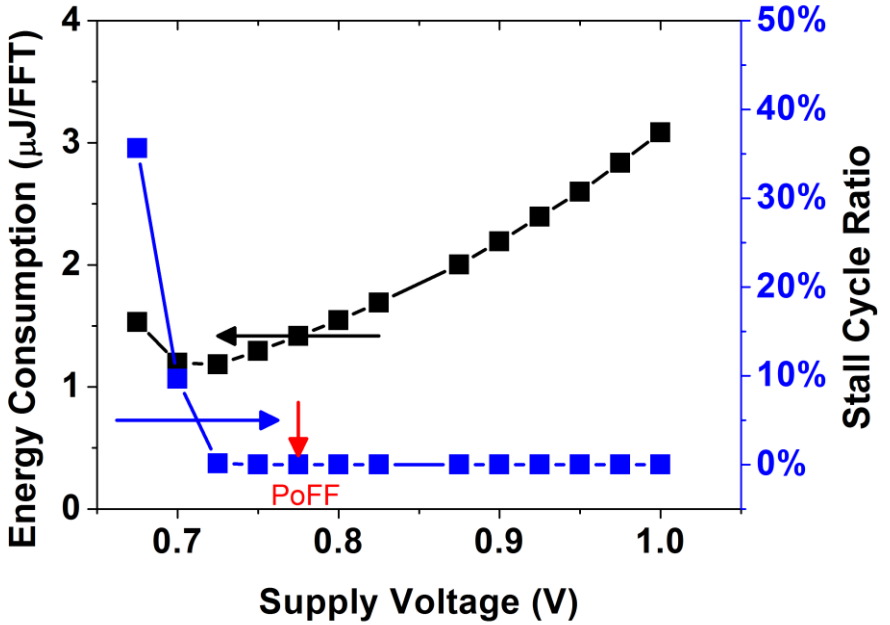
Fast





Bubble Razor Results





Worst Case	200 MHz	8.5 FFT/ms
First Failure	333 MHz	14.2 FFT/ms
Optimum	425 MHz	17.3 FFT/ms

Worst Case	1.0 V	3.08 µJ/FFT
First Failure	0.775 V	1.42 µJ/FFT
Optimum	0.725 V	1.18 µJ/FFT

Conclusion

- First Razor style implementation on a complete, commercial processor (ARM Cortex-M3).

- Proposed two-phase latch based Razor technique
- Novel local replay algorithm
- Demonstrated automated nature of technique
- Successfully implemented and fabricated in 45nm

- 60% energy efficiency or 100% throughput increase over worst case margining