VISA SYNTHESIS: VARIATION-AWARE INSTRUCTION SET ARCHITECTURE SYNTHESIS

Yuko Hara-Azumi* Takuya Azumi† Nikil D. Dutt‡

*Nara Institute of Science and Technology

†Ritsumeikan University ‡University of California, Irvine

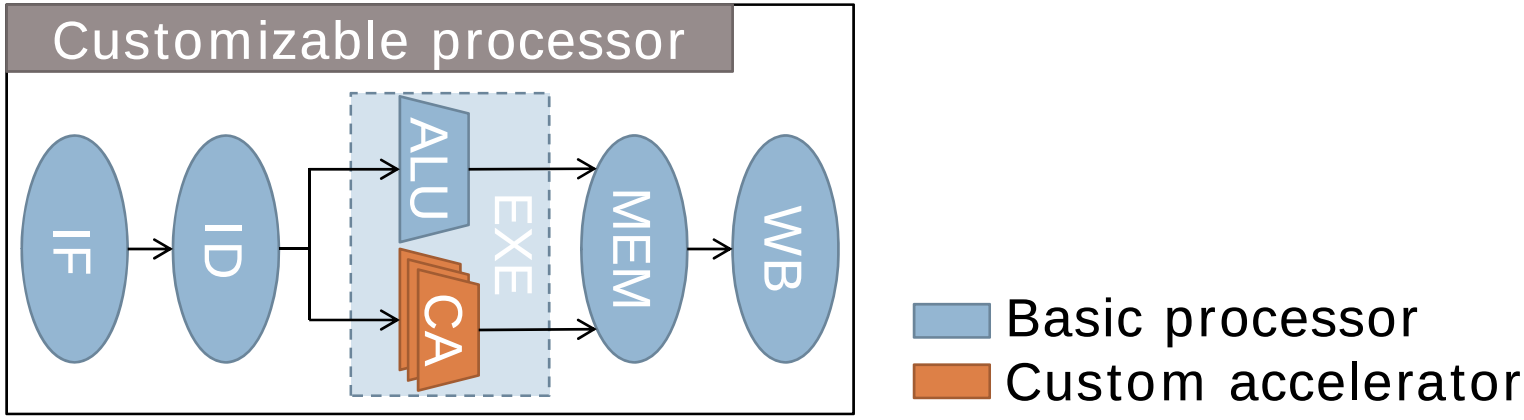
ASP-DAC : Jan. 23rd 2013

Outlines

- Background
  - Previous works
- Variation-aware ISA (VISA) synthesis
  - Razor architecture
  - Our architecture (**HW**-side approach)
  - SSTA-based CI selection (**SW**-side approach)
- Experiments
- Conclusions

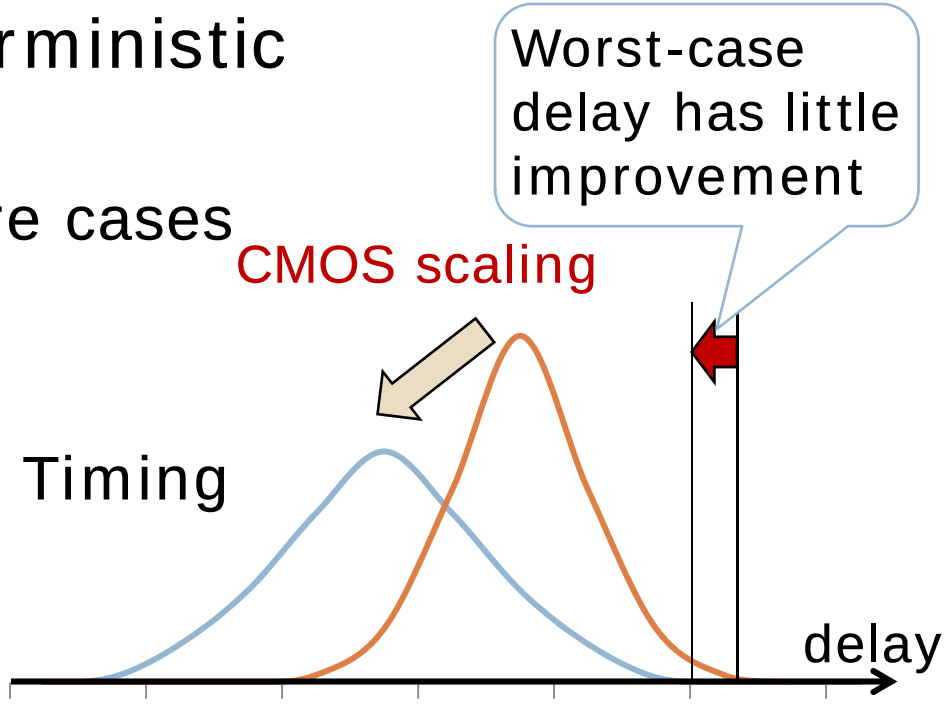
Instruction-set architecture (ISA) synthesis

- Embedded processors are widely used in various applications
  - ISA synthesis: application-specific extension
    - Efficient speedup with less cost (area, power, etc.)
  - Custom instruction (CI) selection
    - Critical computation: CIs → Custom accelerator (CA)
    - The others: basic instructions (BIs) → ALU



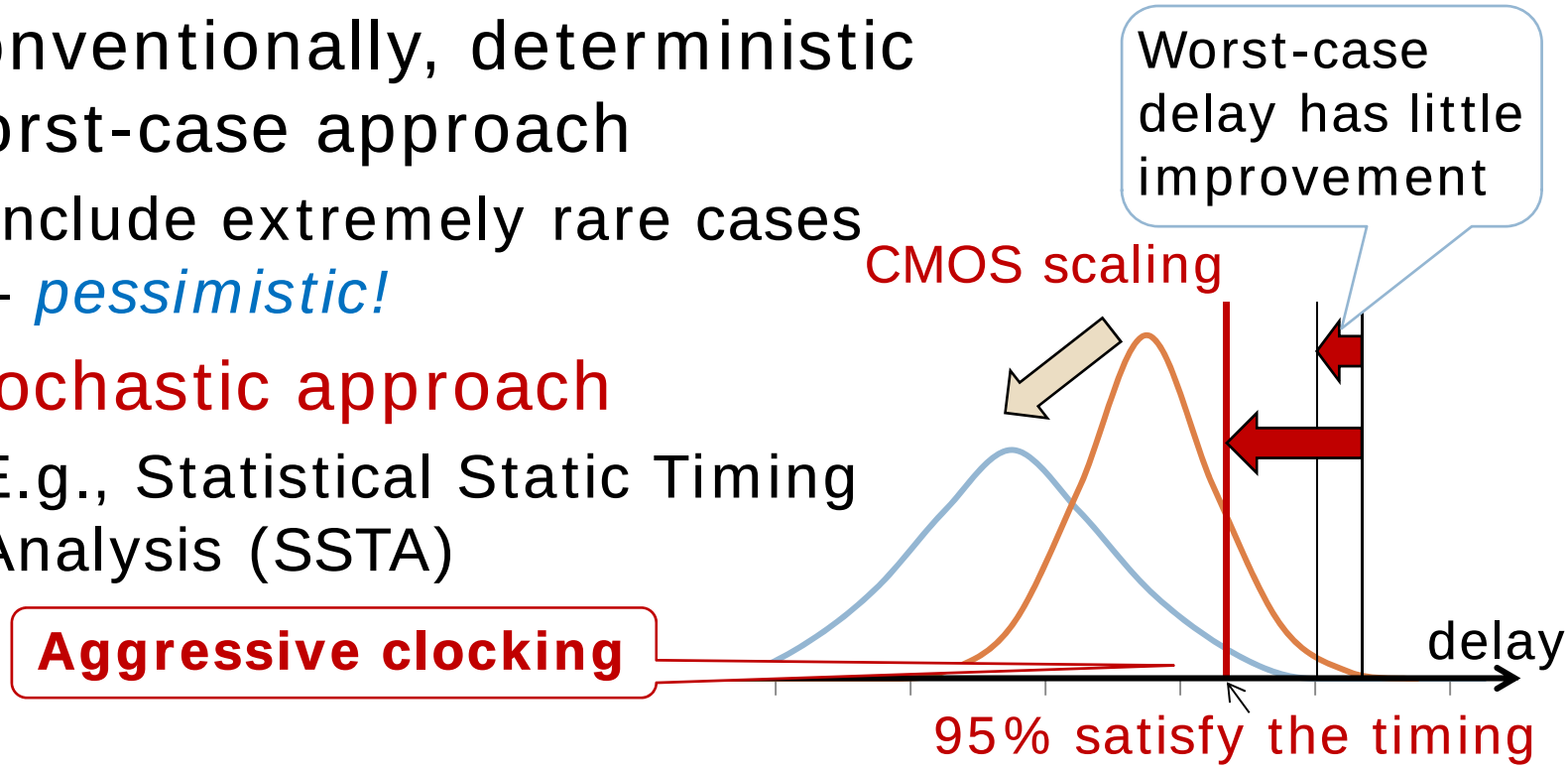
Clock frequency

- A lot of challenging issues of CMOS scaling
  - Cannot expect the frequency scaling
- Process variation
  - Propagation delay varies by environments
  - Conventionally, deterministic worst-case approach
    - Include extremely rare cases – *pessimistic!*
  - Stochastic approach
    - E.g., Statistical Static Timing Analysis (SSTA)



Clock frequency

- A lot of challenging issues of CMOS scaling
  - Cannot expect the frequency scaling
- Process variation
  - Propagation delay varies by environments
  - Conventionally, deterministic worst-case approach
    - Include extremely rare cases – *pessimistic!*
  - Stochastic approach
    - E.g., Statistical Static Timing Analysis (SSTA)



Previous works (1)

- SSTA-based ISA synthesis works
  - [*Kamal'11*]: CI selection with minimum timing yield degradation
    - **Timing yield**: possibility to complete operations for a given target clock
    - Timing yield degradation: may be intolerable for some applications
  - [*Kamal'12*]: Maximum speedup with no timing yield degradation
    - An extra cycle to CIs with less-than-1.0 timing yield
    - Static approach only: extra cycles even when no timing faults occur – *pessimistic!*
  - Timing violation by BIs are not considered

Previous works (2)

- Variation-tolerable works in other fields
  - Architectures: Razor [*Ernst'03*], etc.
    - Detect and correct timing faults dynamically
  - High-level synthesis: SSTA-based works ([*Cong'09*], etc.)
    - Use Razor-flipflops for aggressive clocking without timing yield degradation
- HW approach only - *very costly!*
  - Comprehensive approach from both HW/SW viewpoints is necessary

Outlines

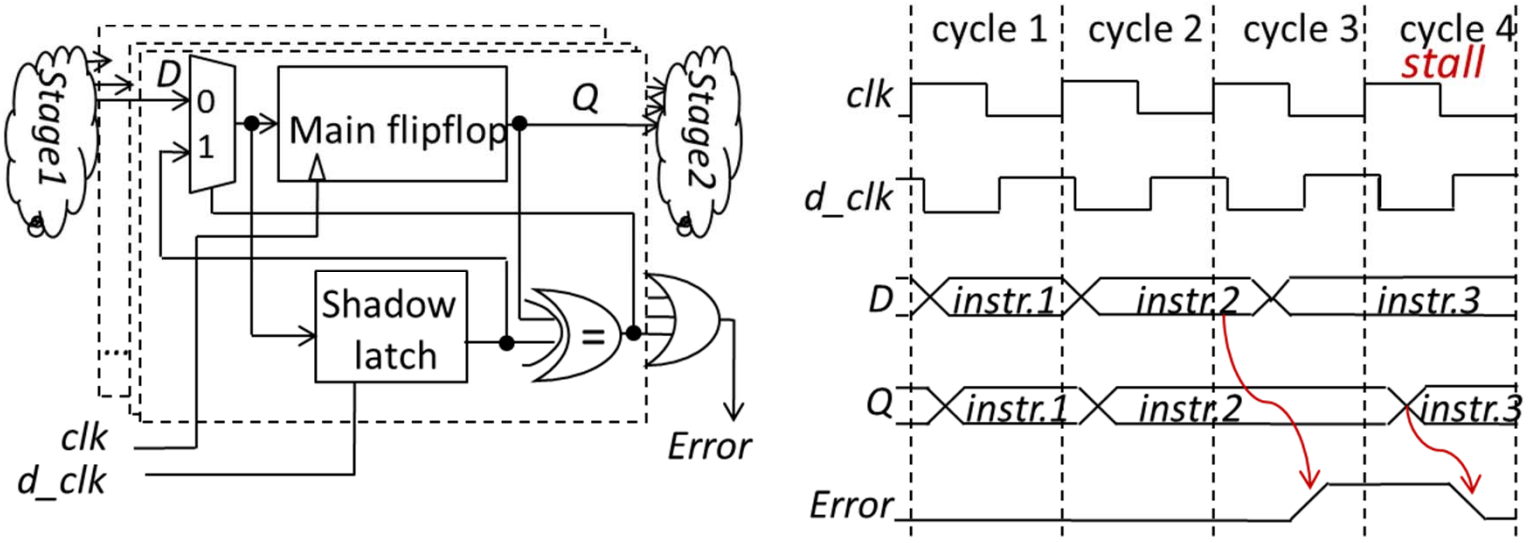
- Background
  - Previous works
- Variation-aware ISA (VISA) synthesis
  - Razor architecture
  - Our architecture (**HW**-side approach)
  - SSTA-based CI selection (**SW**-side approach)
- Experiments
- Conclusions

VISA synthesis

- **VISA**: **V**ariation-aware **ISA** synthesis
  - Performance improvement by making effective use of process variation on both HW and SW
    - **HW**: Dynamic fault detection & correction with minimum performance degradation for aggressive clocking (based on Razor)
    - **SW**: SSTA-based CI selection effectively by exploiting application features
  - Handle timing violation of both BIs & CIs
  - Applicable to any processors which have at least a mechanism of dynamic timing fault detection

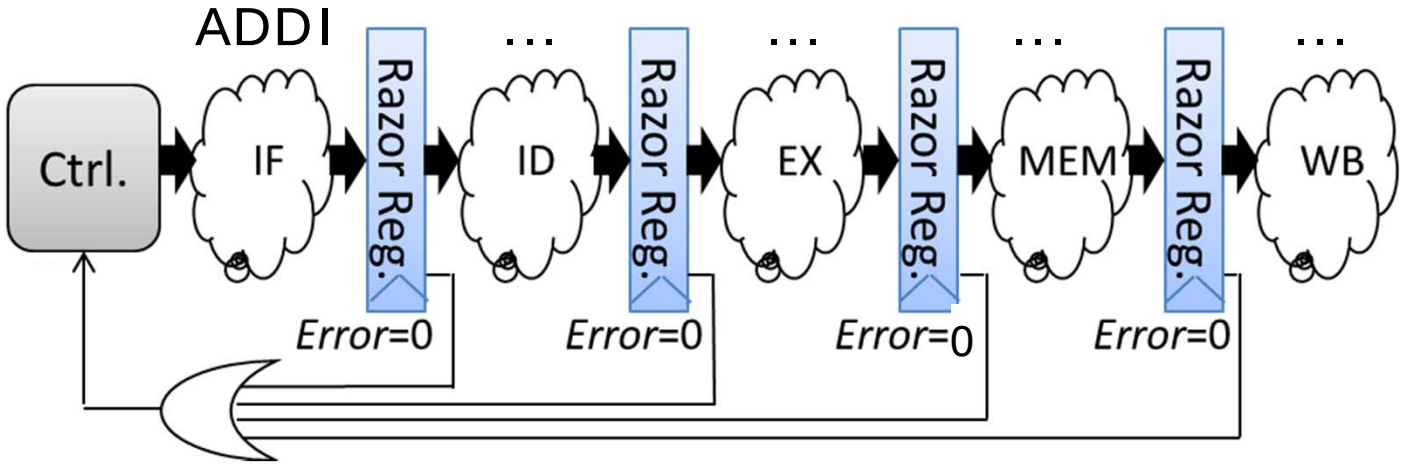
Razor processor architecture (1)

- Razor flipflop: Main flipflop + shadow latch (delayed clock)
  - Fault detection: compares the results
  - Fault correction: simply copies the correct result to the main flipflop during 1-cycle stall



Razor processor architecture (2)

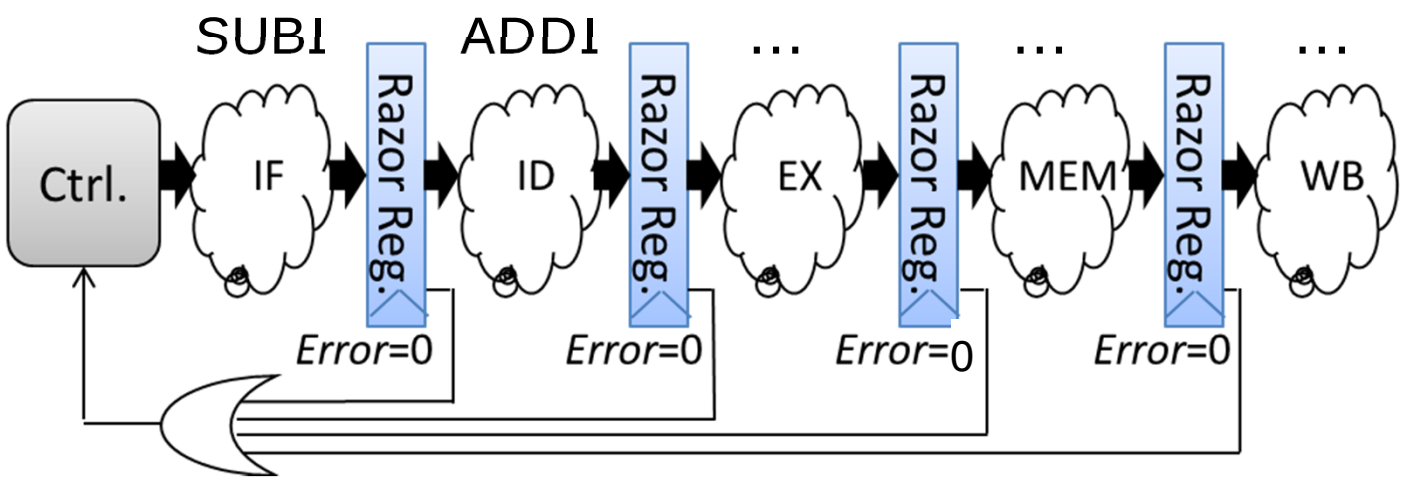

```
...
ADDI R1 R1 7
SUBI R2 R2 1
NEW R5 R1 R3
OR R1 R1 R2
AND R5 R5 R6
ADDI R4 R4 3
...
```



cycle / stage	t
IF	ADDI
ID	...
EX	...
MEM	...
WB	...

Razor processor architecture (2)

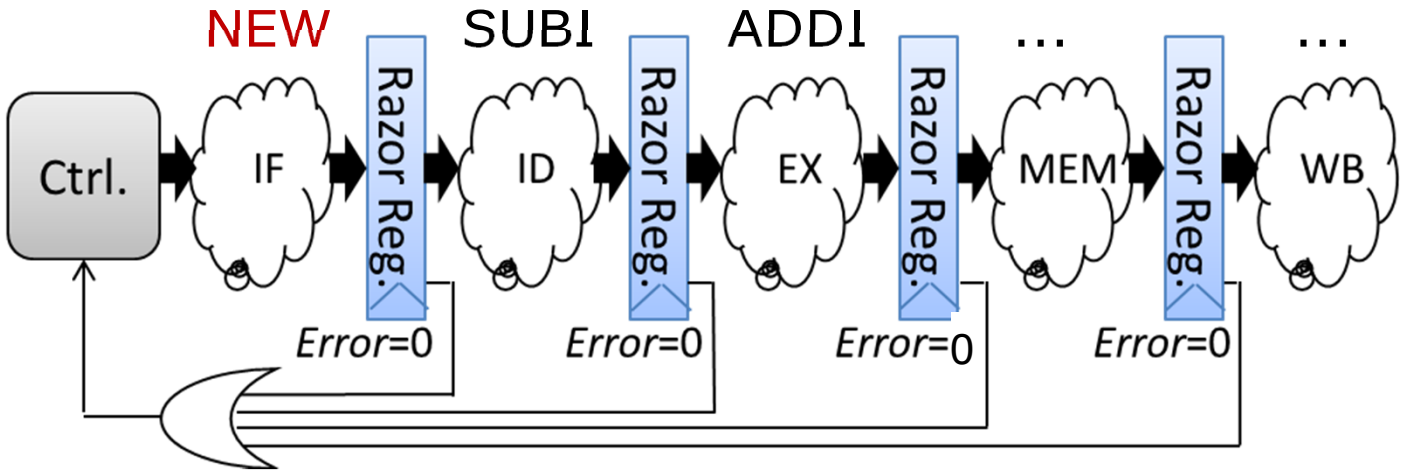

```
...
ADDI R1 R1 7
SUBI R2 R2 1
NEW R5 R1 R3
OR R1 R1 R2
AND R5 R5 R6
ADDI R4 R4 3
...
```



cycle / stage	t	t+1
IF	ADDI	SUBI
ID	...	ADDI
EX	...	...
MEM	...	...
WB	...	...

Razor processor architecture (2)

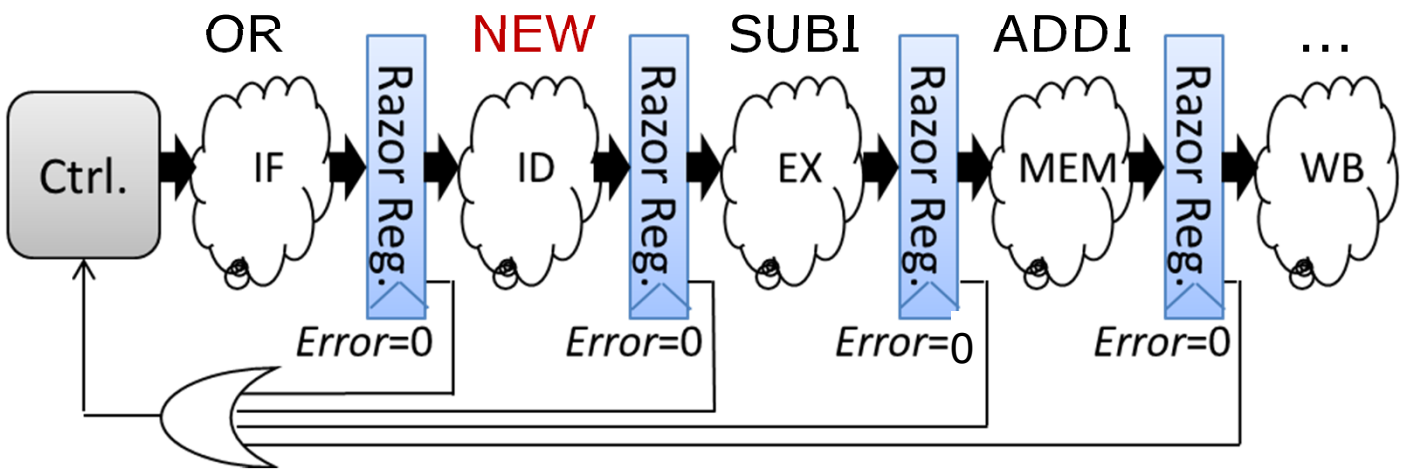

```
...
ADDI R1 R1 7
SUBI R2 R2 1
NEW R5 R1 R3
OR R1 R1 R2
AND R5 R5 R6
ADDI R4 R4 3
...
```



cycle / stage	t	t+1	t+2
IF	ADDI	SUBI	NEW
ID	...	ADDI	SUBI
EX	...	...	ADDI
MEM	...	...	...
WB	...	...	...

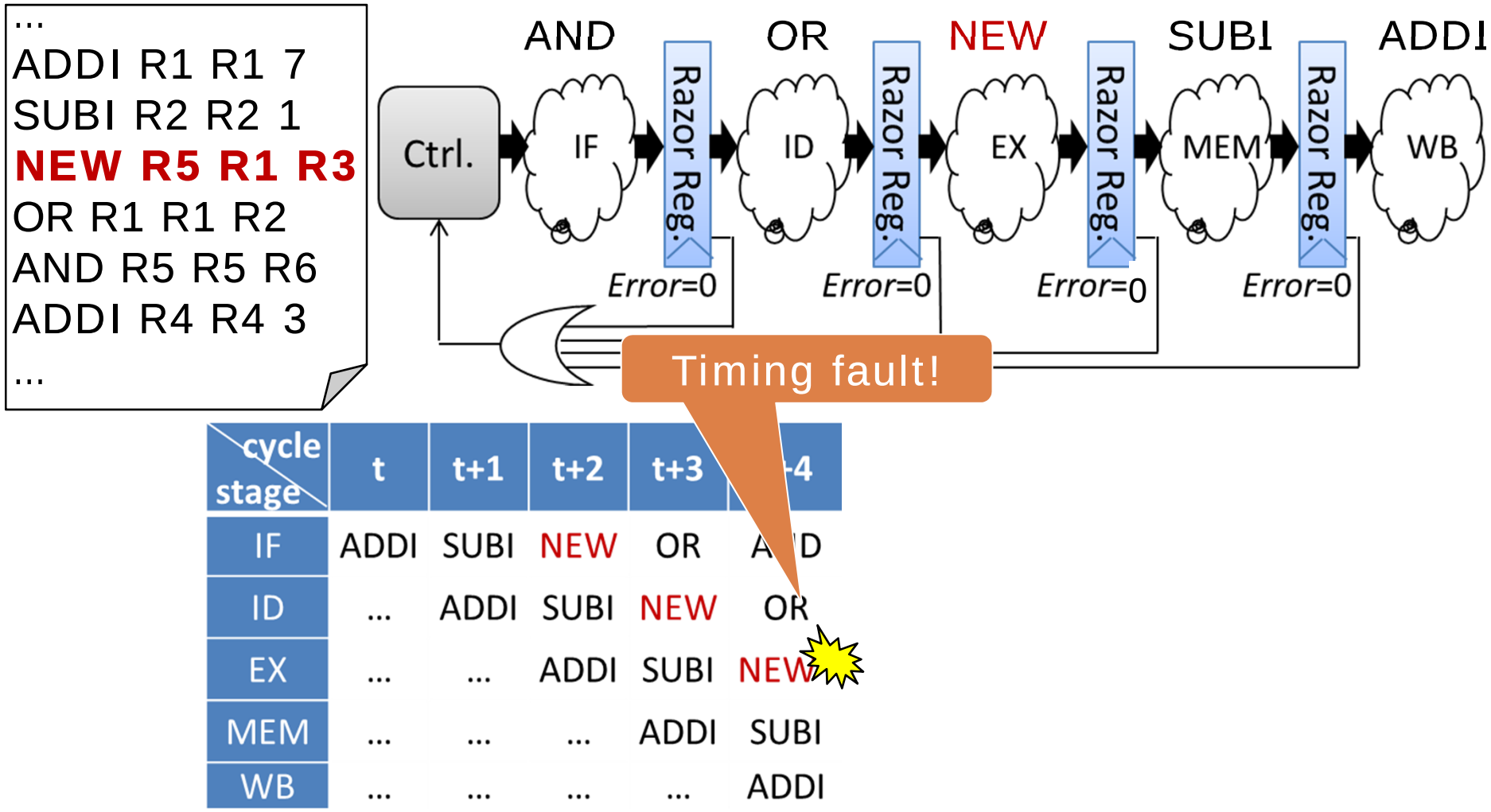
Razor processor architecture (2)


```
...
ADDI R1 R1 7
SUBI R2 R2 1
NEW R5 R1 R3
OR R1 R1 R2
AND R5 R5 R6
ADDI R4 R4 3
...
```



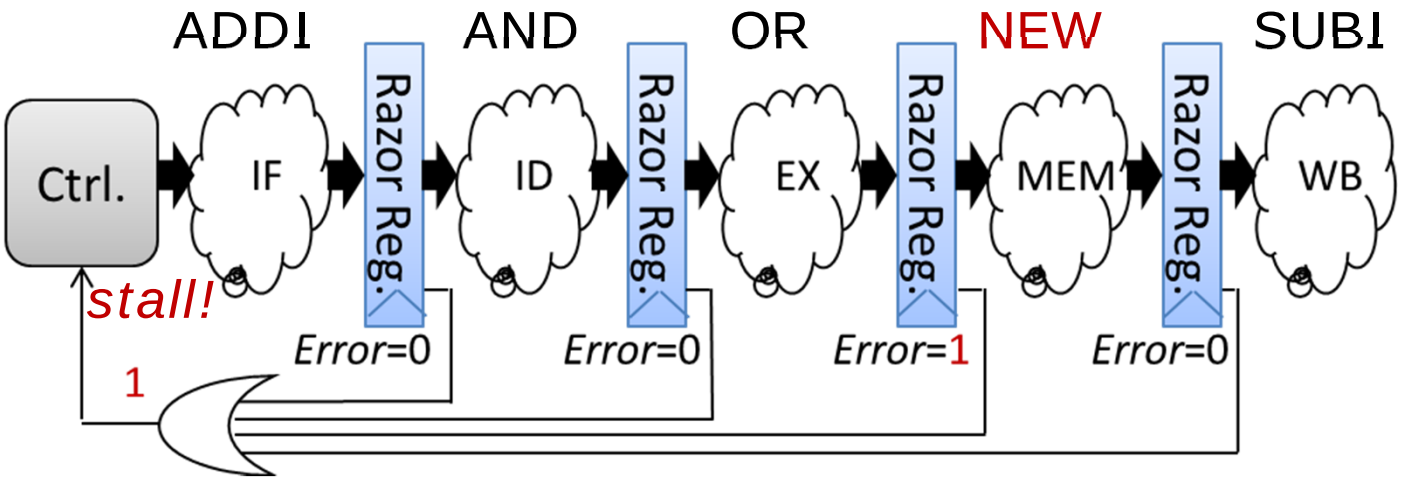
cycle / stage	t	t+1	t+2	t+3
IF	ADDI	SUBI	NEW	OR
ID	...	ADDI	SUBI	NEW
EX	...	...	ADDI	SUBI
MEM	...	...	...	ADDI
WB	...	...	...	...

Razor processor architecture (2)



Razor processor architecture (2)


```
...
ADDI R1 R1 7
SUBI R2 R2 1
NEW R5 R1 R3
OR R1 R1 R2
AND R5 R5 R6
ADDI R4 R4 3
...
```

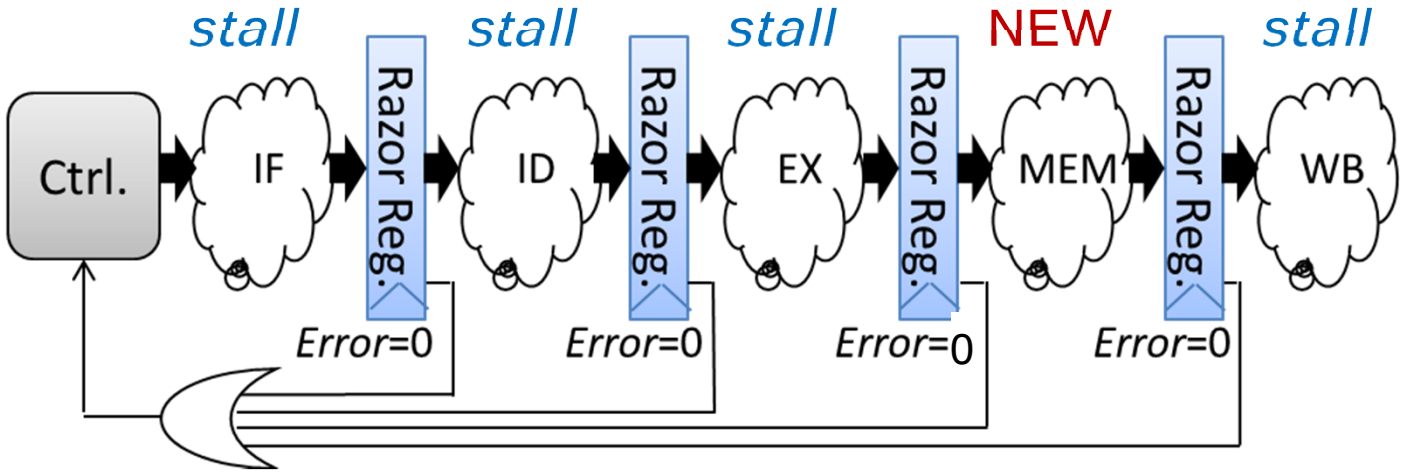


cycle / stage	t	t+1	t+2	t+3	t+4	t+5
IF	ADDI	SUBI	NEW	OR	AND	ADDI
ID	...	ADDI	SUBI	NEW	OR	AND
EX	...	...	ADDI	SUBI	NEW*	OR
MEM	...	...	...	ADDI	SUBI	NEW*
WB	...	...	...	...	ADDI	SUBI

The shadow latch detects the timing fault and issues the Error signal

Razor processor architecture (2)


```
...
ADDI R1 R1 7
SUBI R2 R2 1
NEW R5 R1 R3
OR R1 R1 R2
AND R5 R5 R6
ADDI R4 R4 3
...
```

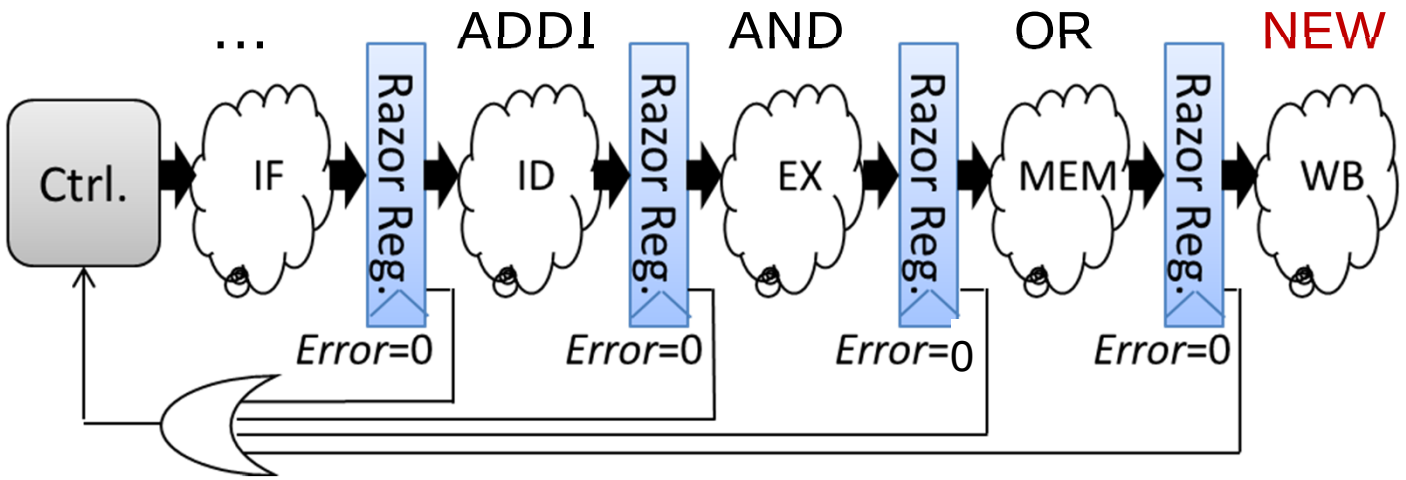


cycle / stage	t	t+1	t+2	t+3	t+4	t+5	t+6
IF	ADDI	SUBI	NEW	OR	AND	ADDI	*stall*
ID	...	ADDI	SUBI	NEW	OR	AND	*stall*
EX	...	...	ADDI	SUBI	NEW*	OR	*stall*
MEM	...	...	...	ADDI	SUBI	NEW*	NEW
WB	...	...	...	...	ADDI	SUBI	*stall*

The correct result of MULT is provided to the MEM stage

Razor processor architecture (2)

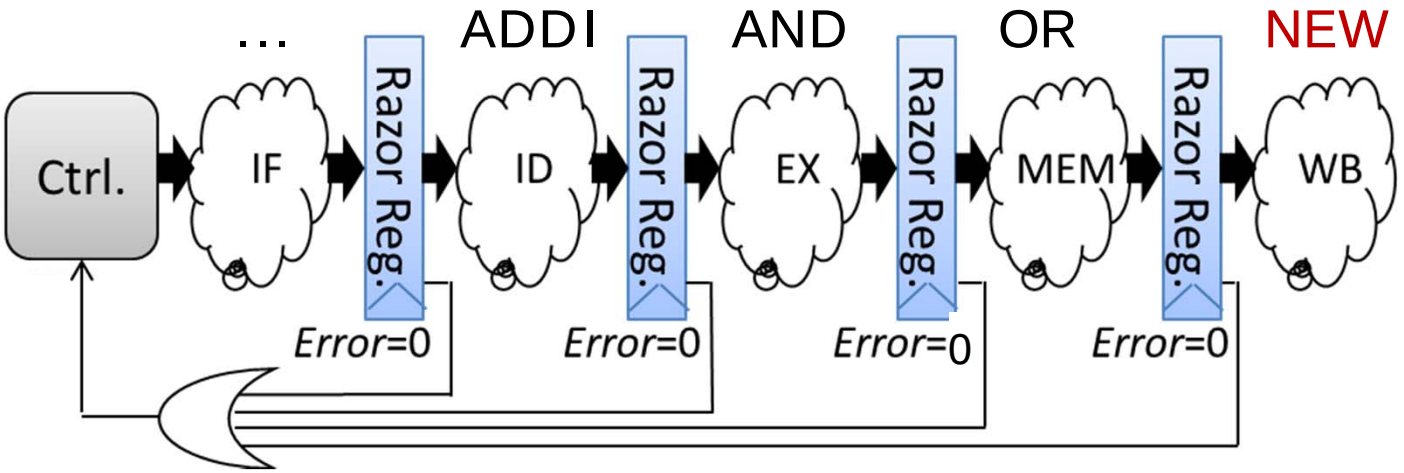

```
...
ADDI R1 R1 7
SUBI R2 R2 1
NEW R5 R1 R3
OR R1 R1 R2
AND R5 R5 R6
ADDI R4 R4 3
...
```



cycle / stage	t	t+1	t+2	t+3	t+4	t+5	t+6	t+7
IF	ADDI	SUBI	NEW	OR	AND	ADDI	*stall*	...
ID	...	ADDI	SUBI	NEW	OR	AND	*stall*	ADDI
EX	...	...	ADDI	SUBI	NEW*	OR	*stall*	AND
MEM	...	...	...	ADDI	SUBI	NEW*	NEW	OR
WB	...	...	...	...	ADDI	SUBI	*stall*	NEW

Razor processor architecture (2)


```
...
ADDI R1 R1 7
SUBI R2 R2 1
NEW R5 R1 R3
OR R1 R1 R2
AND R5 R5 R6
ADDI R4 R4 3
...
```



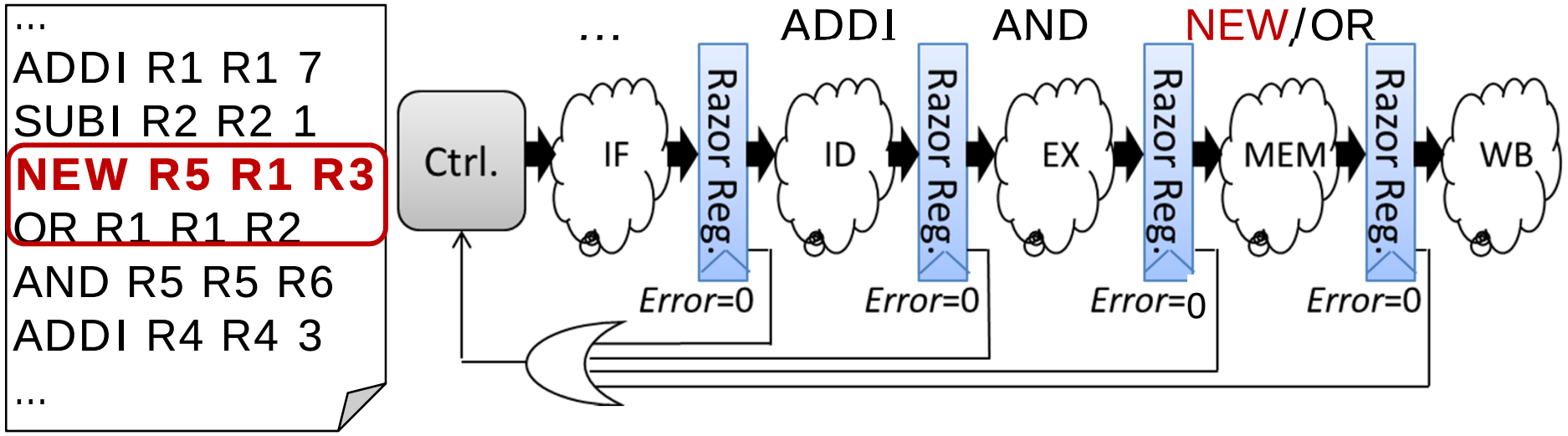
	t+3	t+4	t+5	t+6	t+7
		AND	ADDI	*stall*	...
		OR	AND	*stall*	ADD
		NEW*	OR	*stall*	AND
MEM	ADDI	SUBI	NEW*	NEW	OR
WB	...	ADDI	SUBI	*stall*	NEW

OR has no dependency from & share no arithmetic unit with **NEW**

NEW can be executed *with no stall*

Actually this stall is **not** necessary!

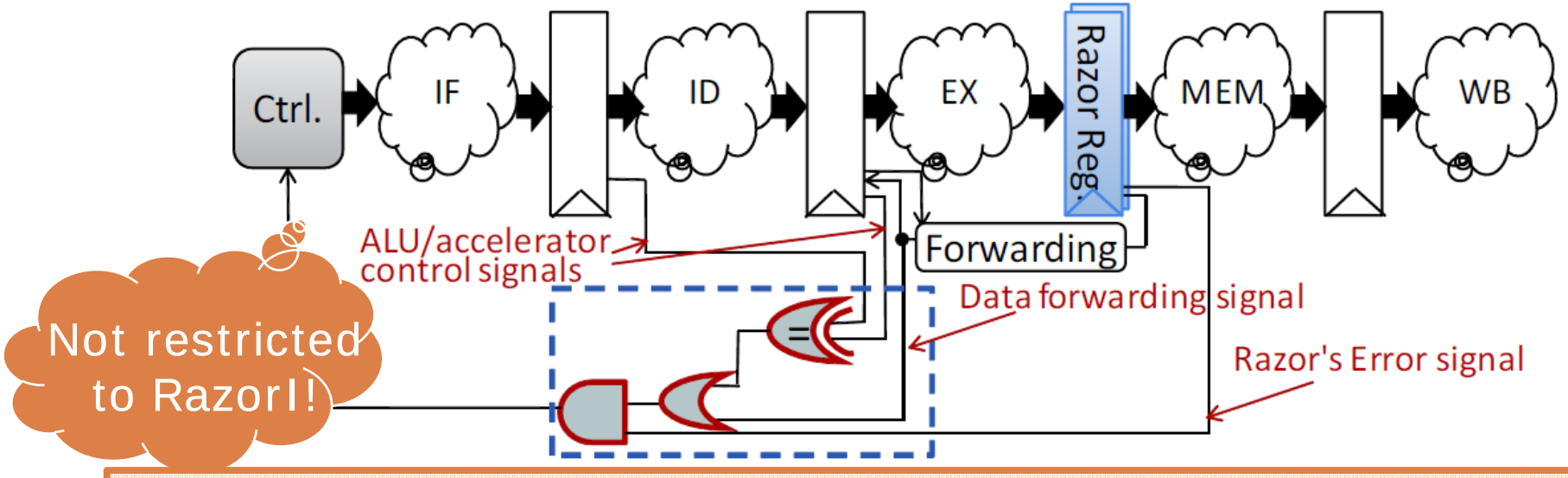
Razor processor architecture (2)



cycle / stage	t	t+1	t+2	t+3	t+4	t+5	t+6	t+7
IF	ADDI	SUBI	NEW	OR	AND	ADDI	...	...
ID	...	ADDI	SUBI	NEW	OR	AND	ADDI	...
EX	...	...	ADDI	SUBI	NEW*	OR	AND	ADDI
MEM	...	...	...	ADDI	SUBI	NEW*	NEW/OR	AND
WB	...	...	...	...	ADDI	SUBI	-	NEW/OR

Our architecture model

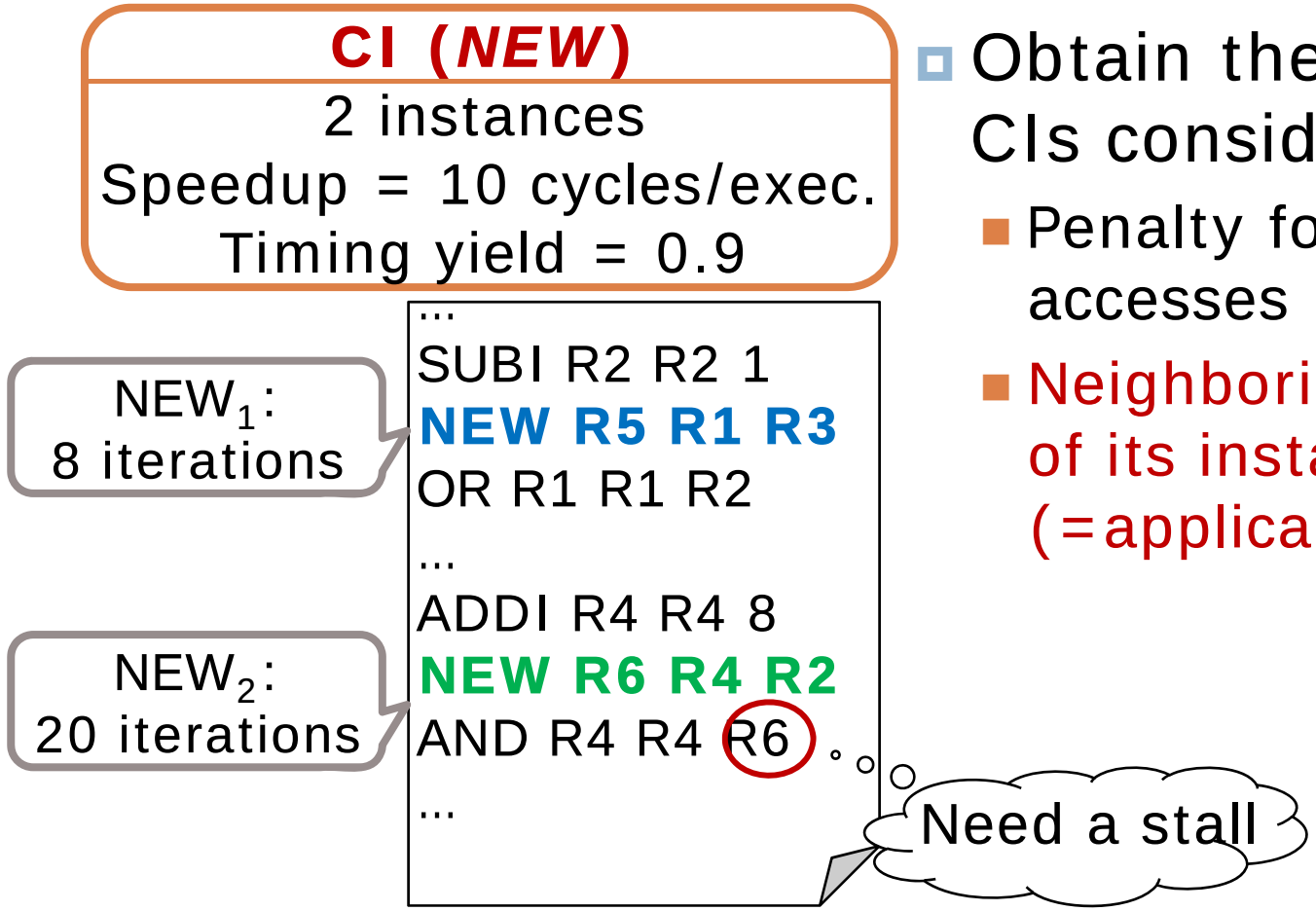
- Removable stalls:
  If instructions in the EX and ID stages have
  1. No temporal dependency (data dependency)
     - Judge from forwarding signals
  2. No physical dependency (resource sharing in EX)
     - Judge from control signals to ALU/custom accelerators



Extension is only for the EX-MEM register

CI selection

- SSTA + application features
  - Speedup effects of individual instances of CIs



- Obtain the speedup of CIs considering
  - Penalty for register file accesses
  - Neighboring instructions of its instances (=application features)

CI selection

- SSTA + application features
  - Speedup effects of individual instances of CIs

CI (*NEW*)
2 instances
Speedup = 10 cycles/exec.
Timing yield = 0.9

NEW_1: 8 iterations

NEW_2: 20 iterations

```
...
SUBI R2 R2 1
NEW R5 R1 R3
OR R1 R1 R2
...
ADDI R4 R4 8
NEW R6 R4 R2
AND R4 R4 R6
...
```

Need a stall

Benefit by NEW
= average speedup by NEW_1
+ average speedup by NEW_2
= {0.9x10 + 0.1x(10-0)}x8
+ {0.9x10 + 0.1x(10-**1**)}x20
=80 + 198
=278

Penalty (stall)

CI selection: constraint

- Only CIs which always finish by the setup time of the shadow latch are selectable
  - Const.: $y_i(c_iT+d) = 1.0$
    - $y_i(t)$: timing yield of instruction i at time t
    - c_i: minimum latency of instruction i in the EX stage
    - T: target clock period
    - d: delayed setup time of the shadow latch
  - *All BIs must hold* the Const.
  - CIs are pruned by the Const.
    - CIs are also pruned by an *area constraint*

Outlines

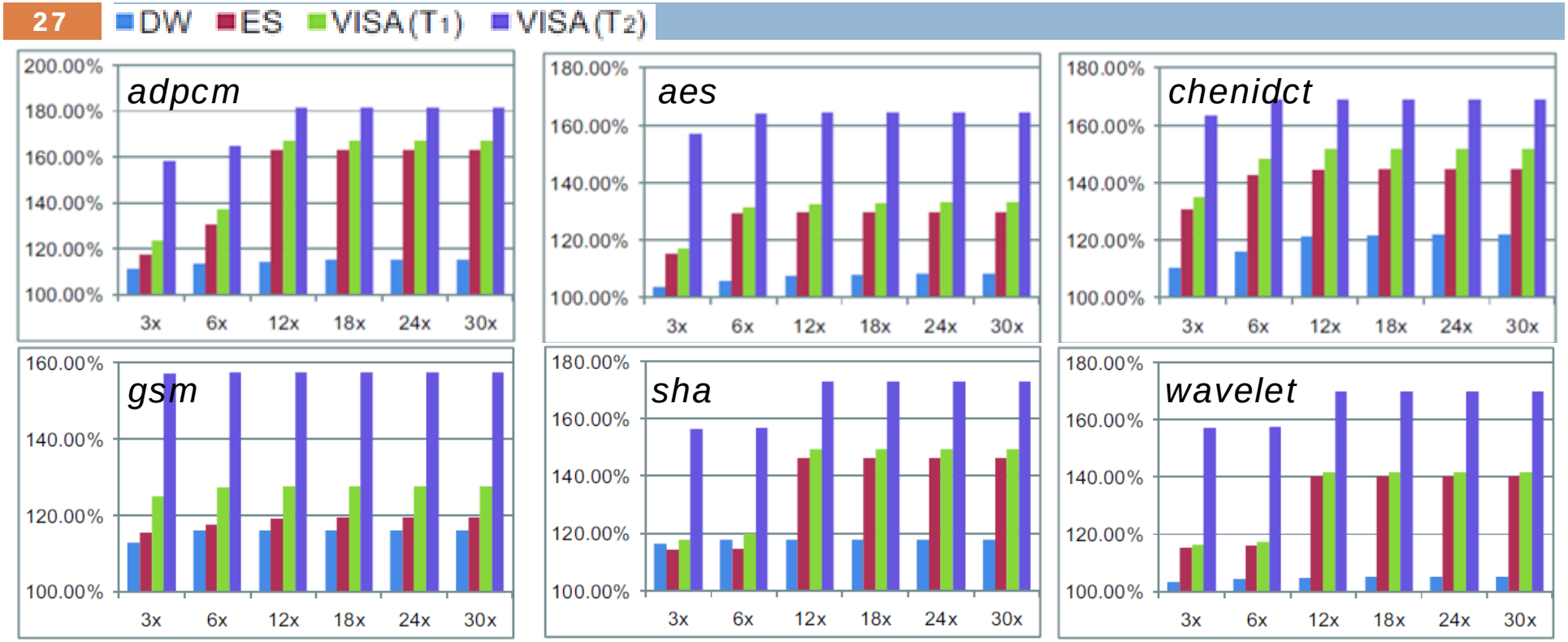
- Background
  - Previous works
- Variation-aware ISA (VISA) synthesis
  - Razor architecture
  - Our architecture (**HW**-side approach)
  - SSTA-based CI selection (**SW**-side approach)
- Experiments
- Conclusions

Experimental setup

- Benchmarks: *adpcm*, *aes*, *chenidct*, *gsm*, and *sha*, and *wavelet*
- Target device: 90nm technology
  1. T_0 : $y_i(c_i T_0) = 1.0$ for all BIs and CIs
  2. T_1 : $y_i(c_i T_1) = 1.0$ for all BIs
  3. T_2 : $y_i(c_i T_2) < 1.0$ & $y_i(c_i T_2 + d) = 1.0$ for some BIs
- Simulator: SimpleScalar
  - MIPS (PISA)
- Comparative methods
  1. A deterministic worst-case method (**DW**): only for T_0
  2. An existing SSTA-based method (**ES**) [*Kamal'12*]: an extra cycle (stall) is always given to CIs with less-than-1.0 timing yield – only for T_1
  3. Our proposed method (**VISA**): compensation by both HW and SW – for T_1 and T_2

All performed greedily

Experimental results: speedup

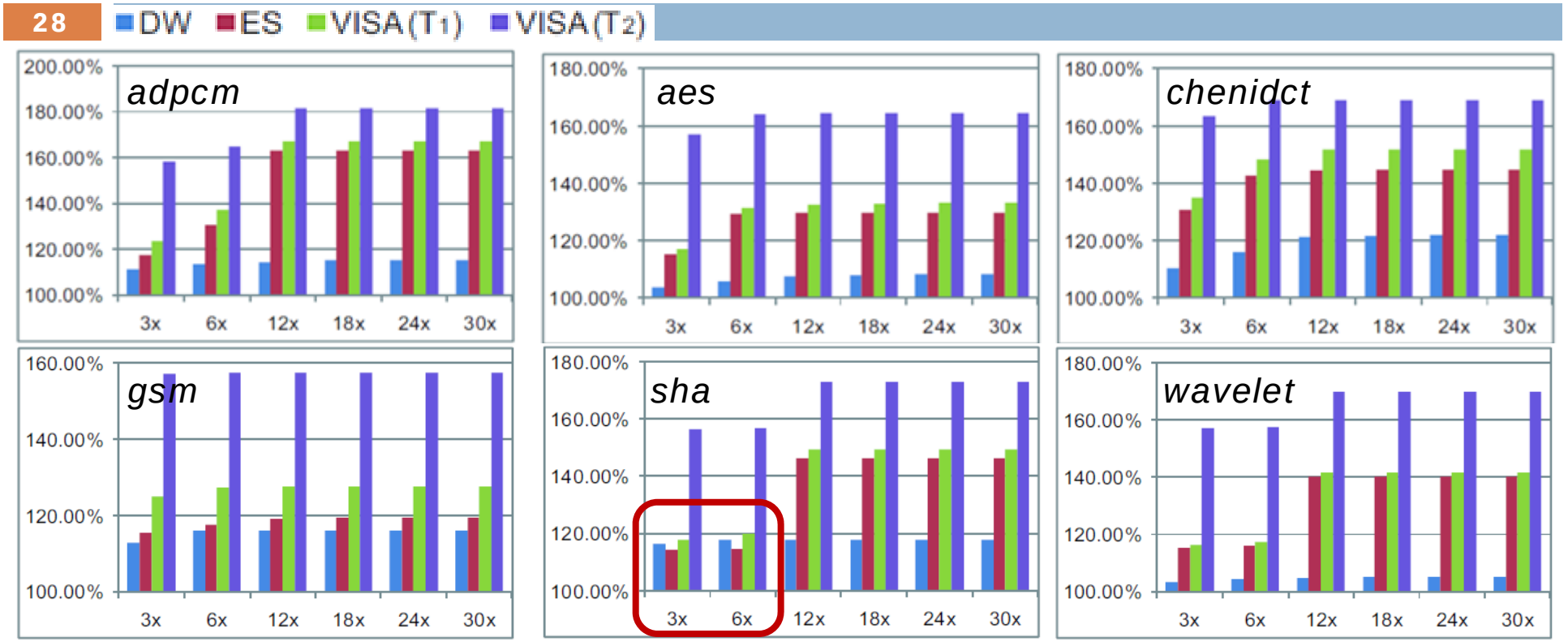


x-axis: Area introduced for CIs (#x ALU's area)

y-axis: Execution time improvement

- DM
  - Not very large speedup
  - Similarly with DM, deterministic approaches quickly face the clock wall

Experimental results: speedup



x-axis: Area introduced for CIs (#x ALU's area)

y-axis: Execution time improvement

- ES
  - Larger speedup than DM, but still pessimistic in that ES *always* gives a stall for CIs with less-than-1.0 timing yield
  - For *sha* with 3x and 6x, even less speedup than DM

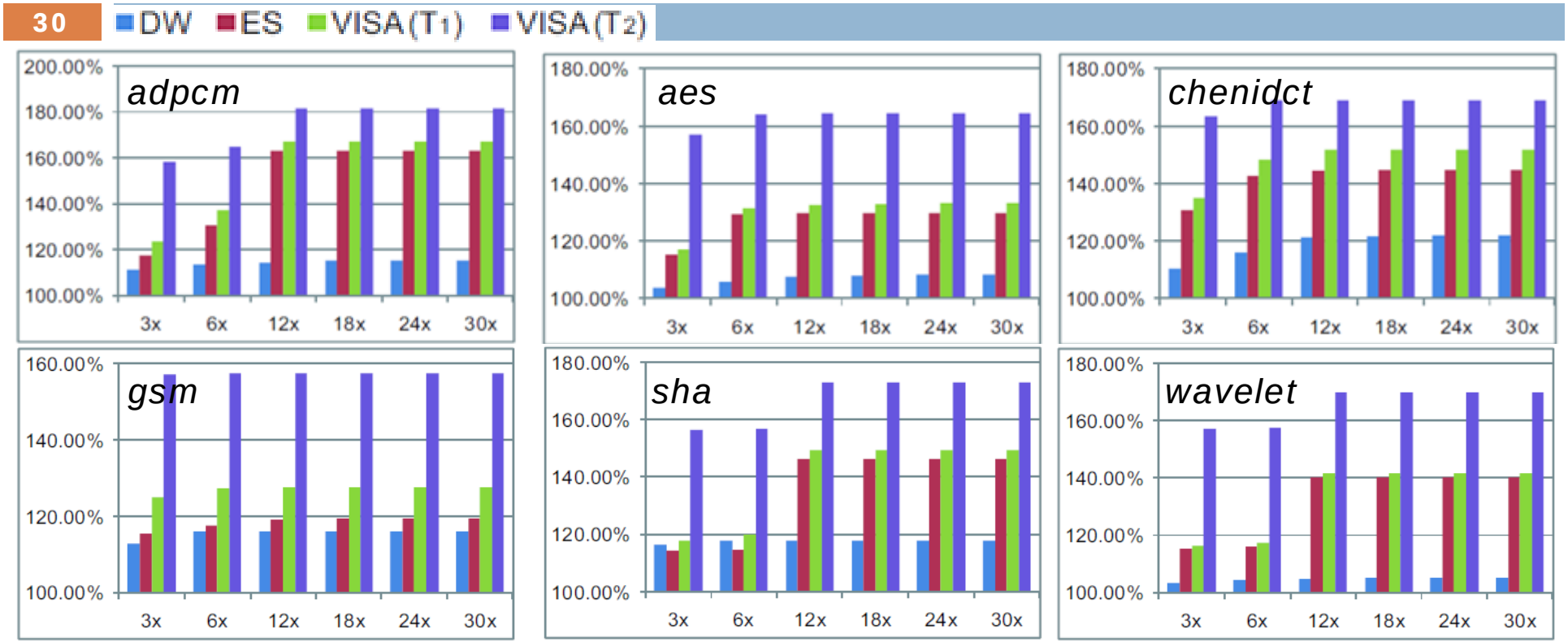
Experimental results: stalls

No. of instr. with a stall / No. of instr. with less-than-1.0 timing yield

Method	*adpcm*	*aes*	*chenidct*	*gsm*	*sha*	*wavelet*
ES (T_1)	2/2	2/2	18/18	7/7	1/1	3/3
VISA(T_1)	0/70	0/112	6/78	4/95	0/17	1/22
VISA(T_2)	497/875	618/1155	417/633	547/907	390/689	366/655

- ES
  - 100% of CIs with less-than-1.0 timing yield *always* take a stall for T_1
- VISA
  - Up to 8% of such CIs *may* take a stall for T_1 → Effectively remove stalls
  - Can select more effective CIs by considering application features

Experimental results: speedup

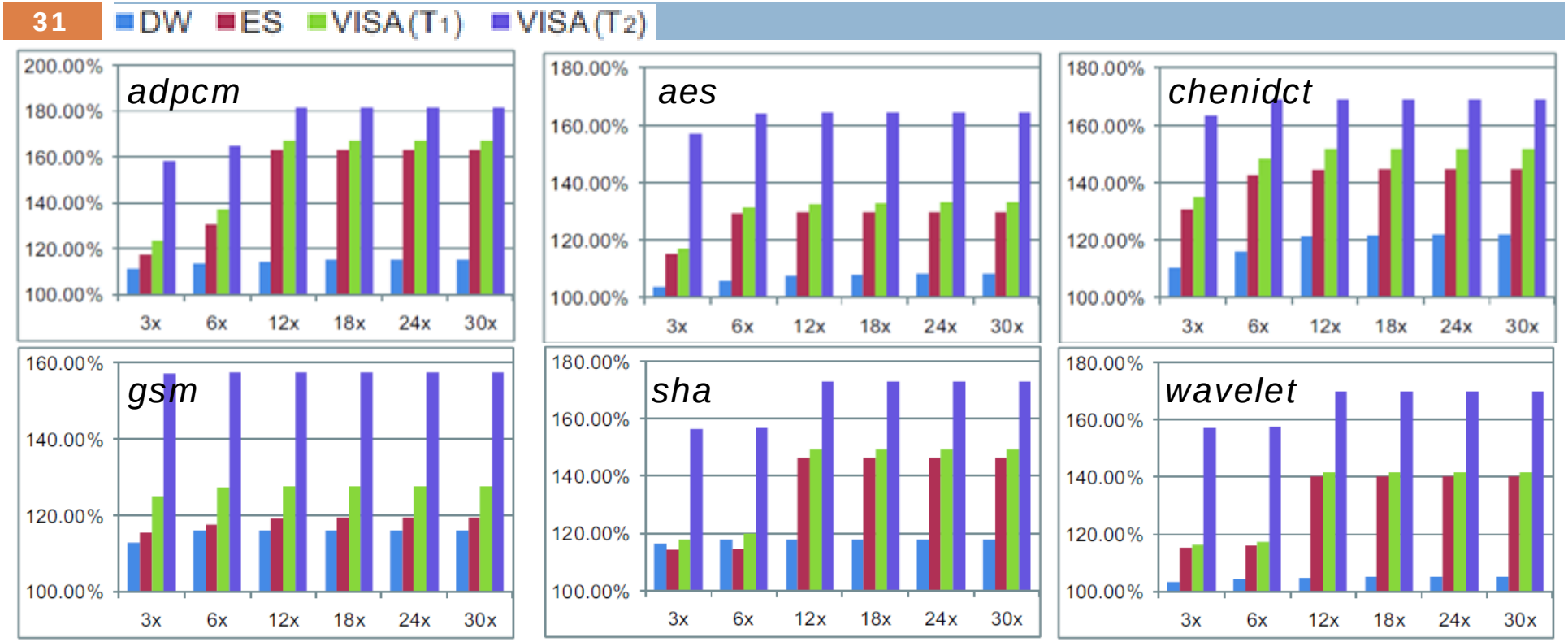


x-axis: Area introduced for CIs (#x ALU's area)

y-axis: Execution time improvement

- VISA
  - More speedup than ES for T_1
    - Stall removal & effective CI selection
  - Outperform DM and ES by up to 61.3% and 13.0%, respectively

Experimental results: speedup



x-axis: Area introduced for CIs (#x ALU's area)

y-axis: Execution time improvement

- VISA
  - More effective for more aggressive clocking
  - Outperform DM and ES by up to 78.0% and 49.4%, respectively, for T_2

Conclusion

- **VISA**: novel **V**ariation-aware **ISA** synthesis
  - Make effective use of process variation comprehensively from both HW and SW
    - HW: Dynamic fault detection & correction with minimum performance degradation
    - SW: SSTA-based CI selection considering application features
  - Substantially improves performance compared with existing methods
    - More effective for more aggressive clocking
    - Up to *78.0%* and *49.4%* performance improvement over two existing approaches