SWAPCODES: ERROR CODES FOR HARDWARE-SOFTWARE COOPERATIVE GPU PIPELINE ERROR DETECTION

Michael B. Sullivan, Siva Kumar Sastry Hari,
Brian Zimmer, Timothy Tsai, Stephen W. Keckler

NVIDIA

GPUS FOR HIGH-RELIABILITY APPLICATIONS

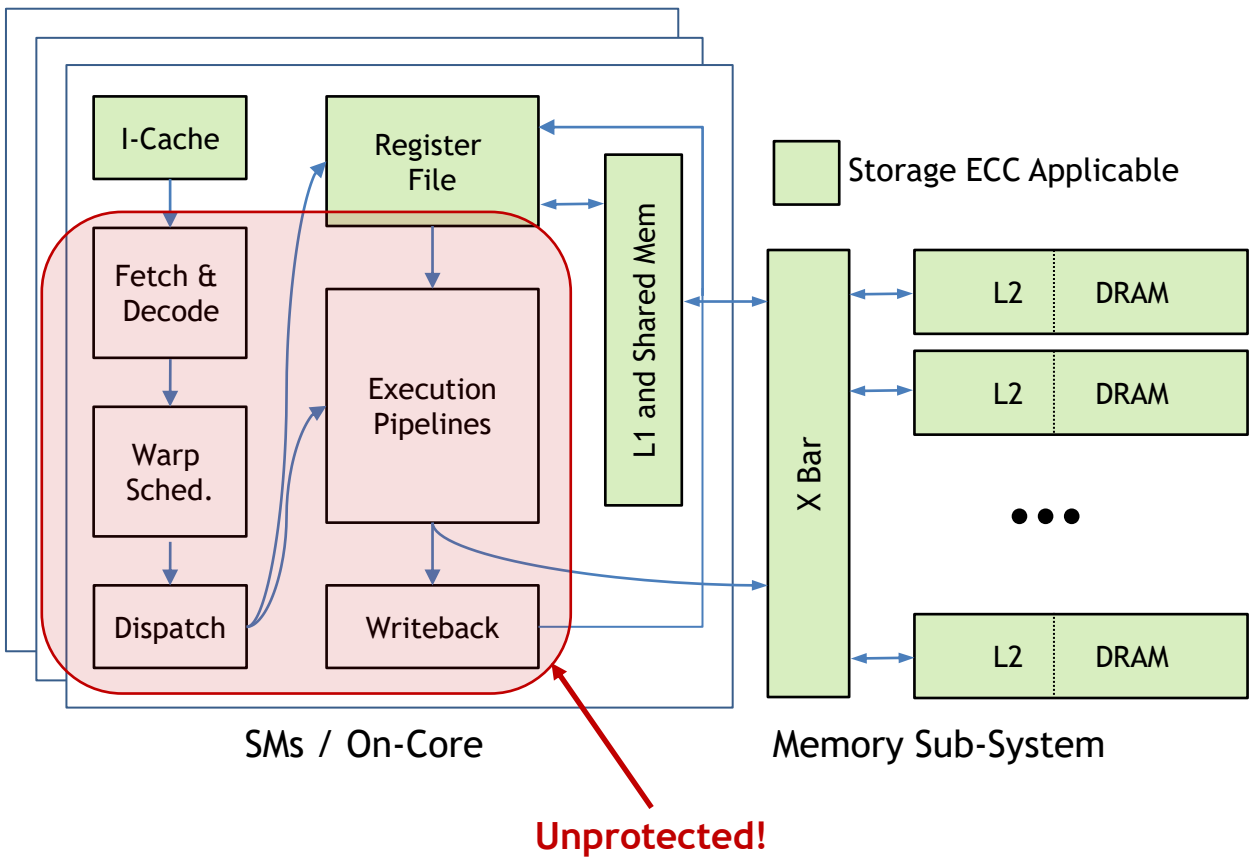
ORNL Summit (World's Most Powerful Supercomputer)
NVIDIA Tesla V100 GPUs

https://www.pcgamer.com/ornls-summit-is-the-most-powerful-supercomputer-in-the-world/

Audi 2018 A8 (World's First Level-3 Autonomous Car)
NVIDIA Tegra K1

https://blogs.nvidia.com/blog/2017/07/11/audi-2018-a8-nvidia-barcelona/

STORAGE ERROR PROTECTION WITH ECC



INSTRUCTION DUPLICATION

(1) Un-Duplicated Code

```
ADD R1, R1, R2
```

(2) Intra-Thread Duplication

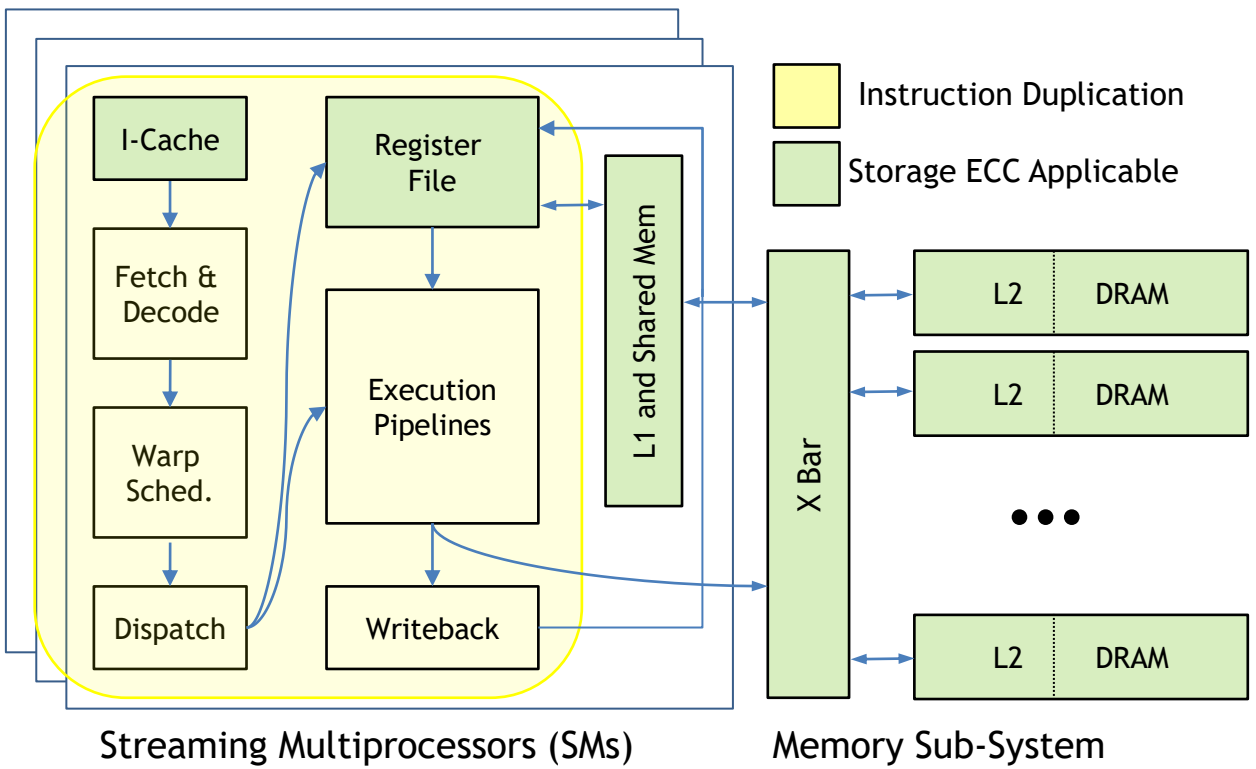
```
    ADD R1, R1, R2 //orig.
    ADD R3, R3, R4 //shad.
    ISETP.EQ P1, R1, R3
@P1 BRA,U `(.L_1) //check
    BPT.TRAP 0x1
.L_1:
```

Explicit checking code before any:

- Loads/Stores
- Branches
- Non-deterministic Instructions (e.g. Clock Read)

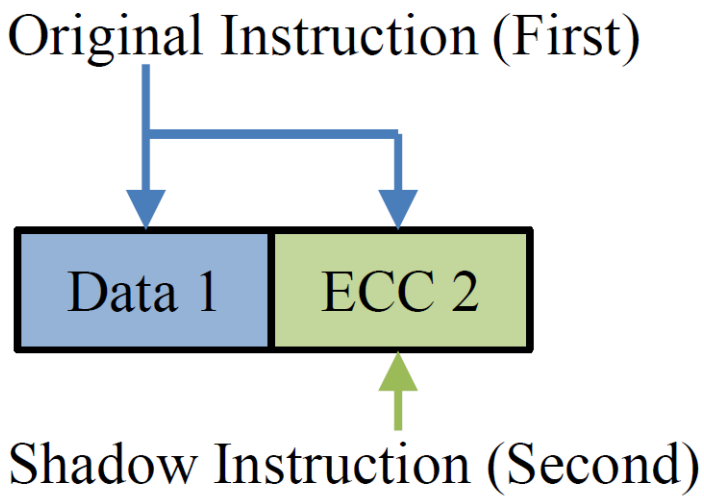
INSTRUCTION DUPLICATION FOR PIPELINE ERROR DETECTION

+ Reliability!
~10x SDC reduction



2x RF storage
> 2x instruction bloat
→ 49% average slowdown ☹

SWAPCODES

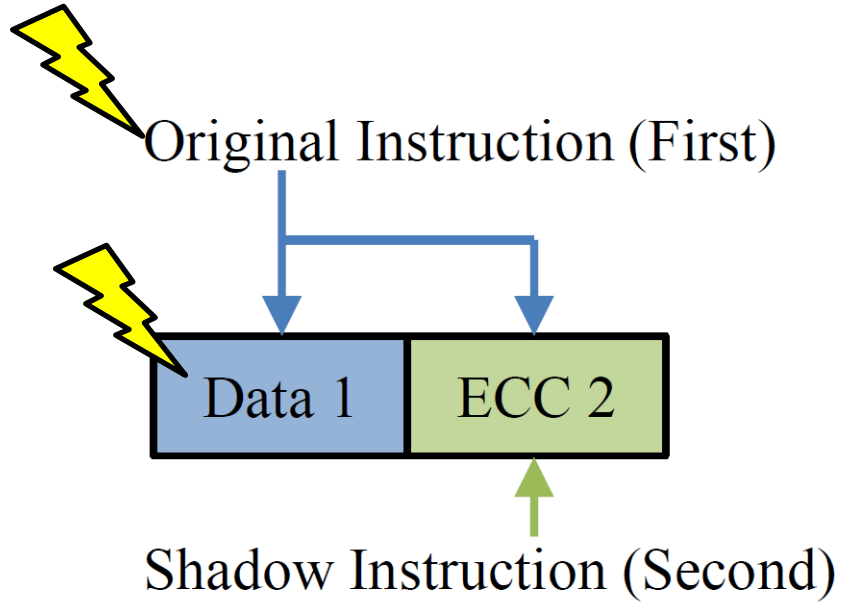


Register file error detection functions as normal but can now detect pipeline errors!

1x RF storage
< 2x instruction bloat
→ less slowdown ☺

SWAPCODES

Example: Error in Original Instruction!



Error in original instruction: corrupted data bits
Error in shadow instruction: corrupted check-bits

SWAPCODES DESIGN PRINCIPLES

Three Key Principles Simplify Implementation

1. No New Per-Thread State
   - In a massively-threaded GPU, per-thread state is expensive!
   - No redundant registers or new microarchitectural buffers.
2. No New Hardware Error Checkers
   - No new self-checking checkers
3. Full Error Containment
   - Errors are detected immediately and cannot “leak”

SWAPCODES CHANGES

Hardware-Software Cooperative with Modest Changes

TABLE: The Swap-ECC hardware and software changes.

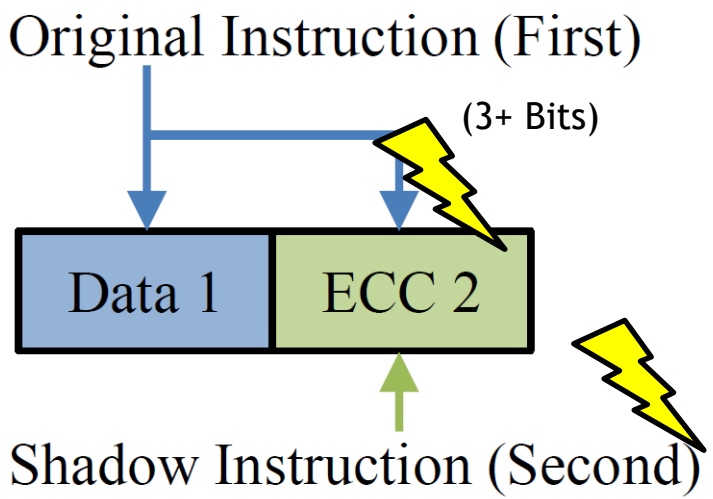
Structure/Program	Swap-ECC Changes	
Backend Compiler	Add an intra-thread duplication pass.	Software
Backend Compiler	Swap-ECC-aware scheduling.	Software
ISA Meta-Data	Add a 1b data write enable.	Hardware
Register File	Add a data write enable and muxes for move propagation.	Hardware
Error Reporting (Storage Correction)	Augmented error reporting to separate storage from pipeline errors.	Hardware

Overheads evaluated with synthesis estimates and shown to be small.

STORAGE CORRECTION

Challenge: Just Correcting All Single-Bit Errors is Not Safe!

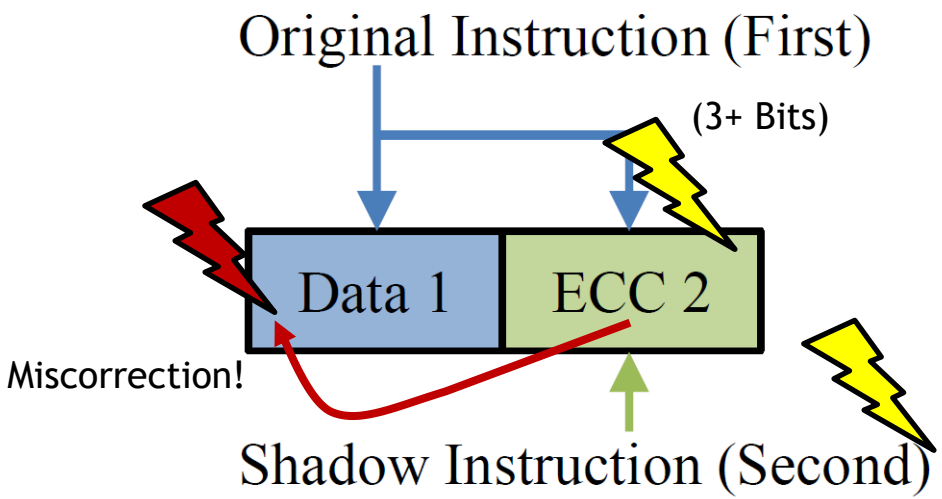
Problem: Error in Shadow (Second) Instruction:



STORAGE CORRECTION

Challenge: Just Correcting All Single-Bit Errors is Not Safe!

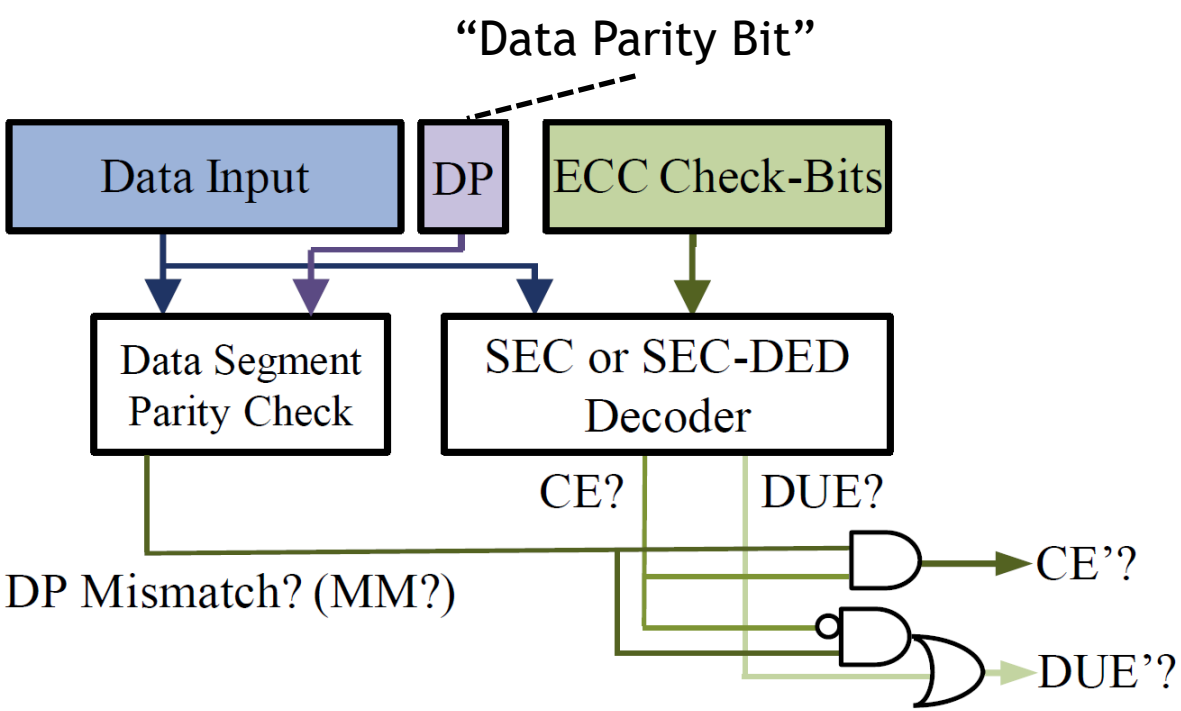
Problem: Error in Shadow (Second) Instruction:



→ **Compute error correction is unsafe.**

Preserving Storage Correction

Allowing Storage Error Correction, Promoting ALL Compute Errors to DUEs

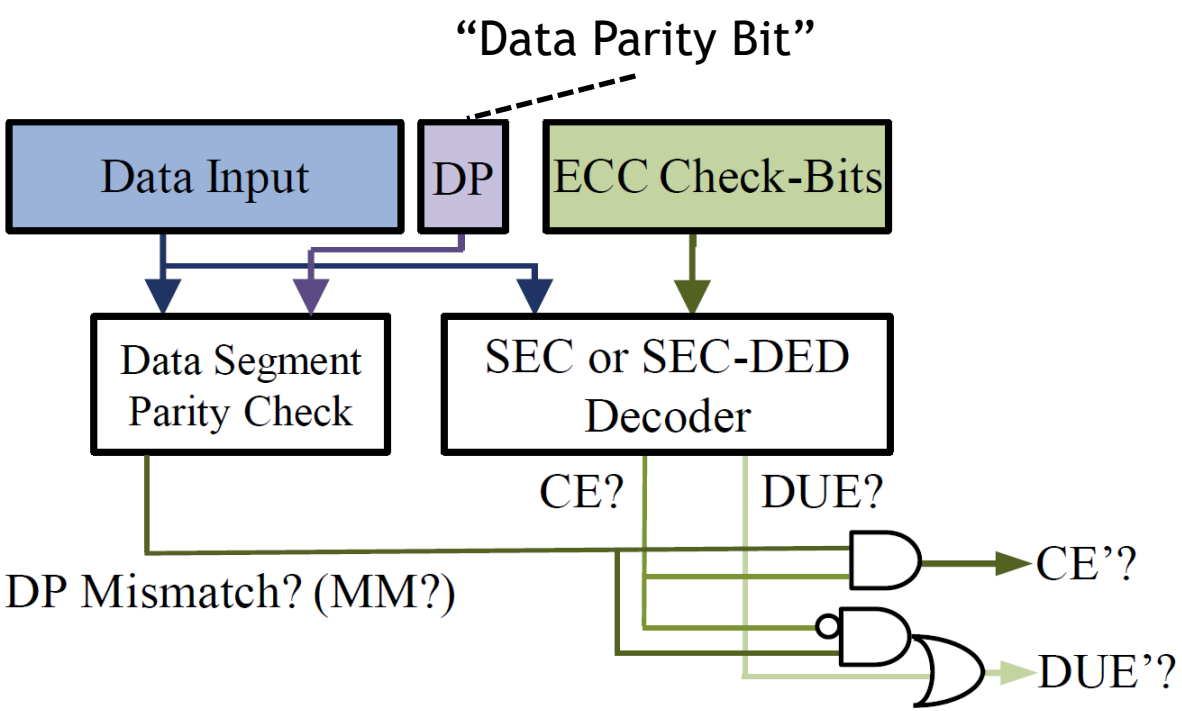


Parity bit *JUST* for the data.

- Generated by the original instruction, *not* swapped.

Preserving Storage Correction

Allowing Storage Error Correction, Promoting ALL Compute Errors to DUEs



Parity bit *JUST* for the data.

- Generated by the original instruction, *not* swapped.

Semantics:

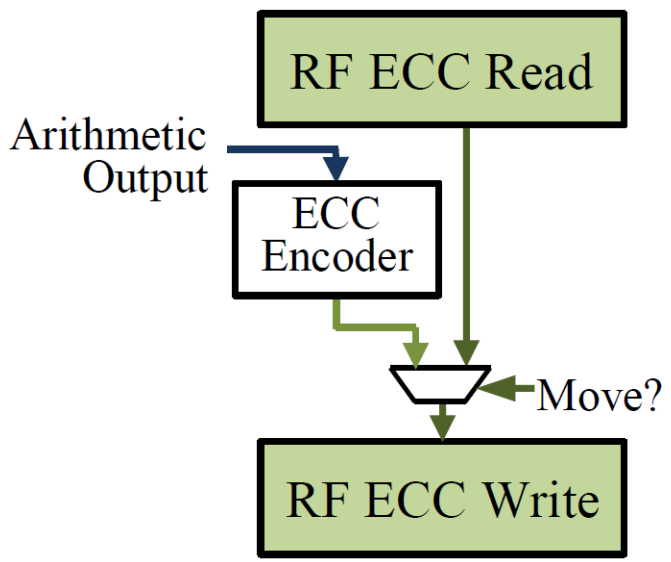
- Data parity bit mismatch: the single-bit error is a **storage error** and it is **allowed**.
- Data parity bit does not mismatch: the single-bit error is a **compute error** and it is flagged as a **DUE**.

Results: Single-Bit Correction, Double-Bit Detection (Storage Errors)
Triple-Bit Detection (Compute Errors)

OPTIMIZATION: MOVE PROPAGATION

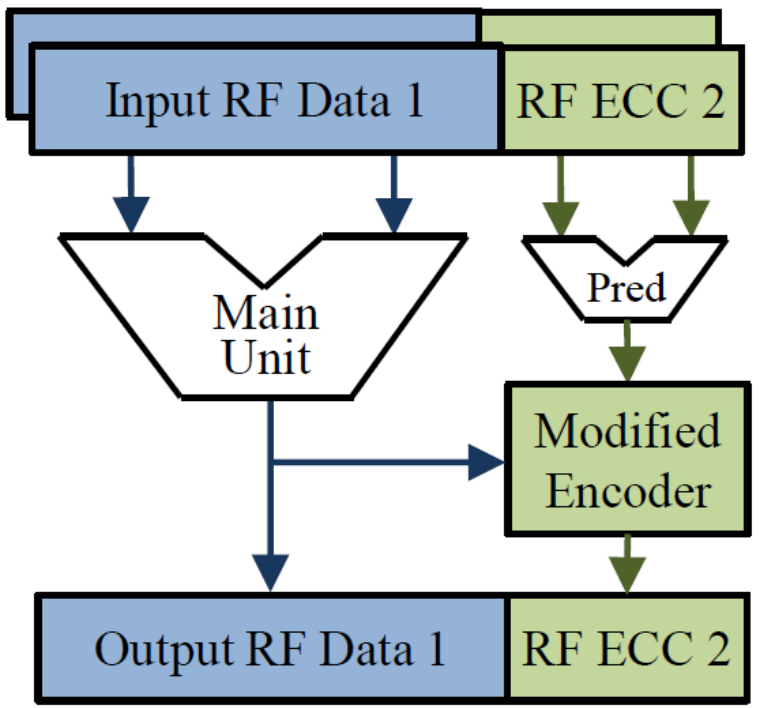
Avoiding the Need to Duplicate MOV Instructions

So long as the ECC check-bits are propagated, then MOV instructions do not need to be duplicated.



OPTIMIZATION: SWAP-PREDICT

Avoiding the Need to Duplicate Other Instructions (e.g. ±, ×)

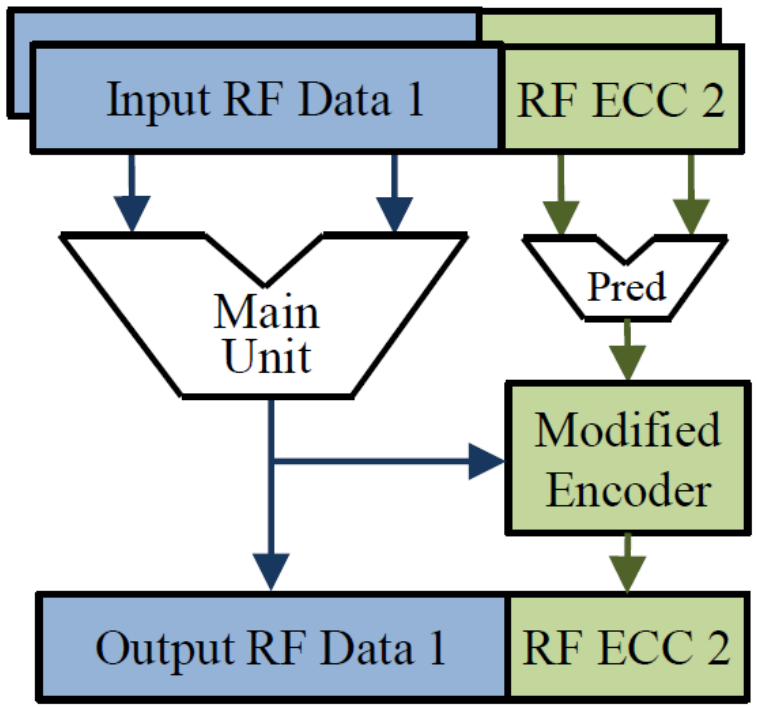


Operation and ECC-specific check-bit prediction can avoid the need to duplicate selected other operations.

Note: “Predict” as in *Parity Prediction*. Not speculative!

SWAP-PREDICT (CONTINUED)

Avoiding the Need to Duplicate Other Instructions (e.g. ±, ×)



Difference from Concurrent Checking:
1. Opportunistic! For selected instructions.
2. No new checkers! Reusing RF decoder.

Case Study: Residue codes for GPUs, with a novel MAC prediction circuits.

PERFORMANCE EVALUATION

Implemented in the Compiler, Run on Silicon

We implement and run each approach on a P100.

Instruction Duplication: Fully functional prototype (verified using error injection and neutron beam testing).

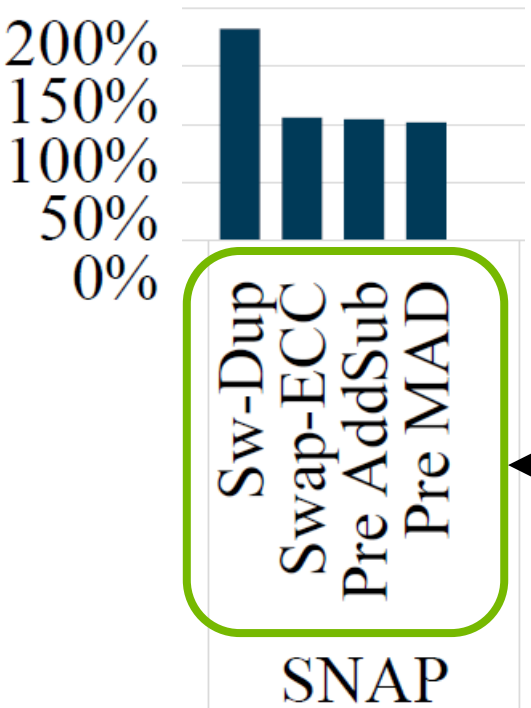
SwapCodes: Software changes only.

- Performance should be representative so long as the modest hardware changes do not alter the system clock period. (Highly likely.)

Benchmarks: Rodinia 2.3, the DOE miniapp “SNAP”, and MatMul from the CUDA SDK.

PERFORMANCE

Clustering Results into 4 Classes!

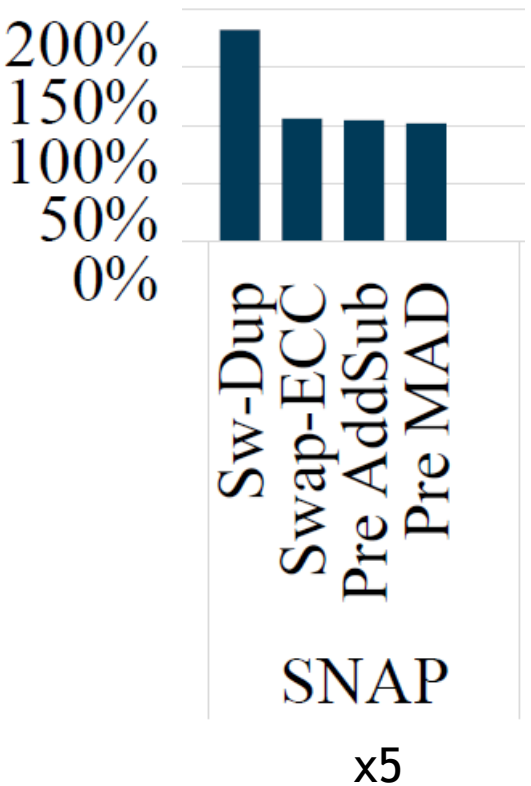


Legend:

SW-Dup: Instruction Duplication Baseline
Swap-ECC: SwapCodes with Move Propagation
Pre-AddSub: Swap-Predict for ±
Pre-MAD: Swap-Predict for ±, ×

PERFORMANCE

Clustering Results into 4 Classes!



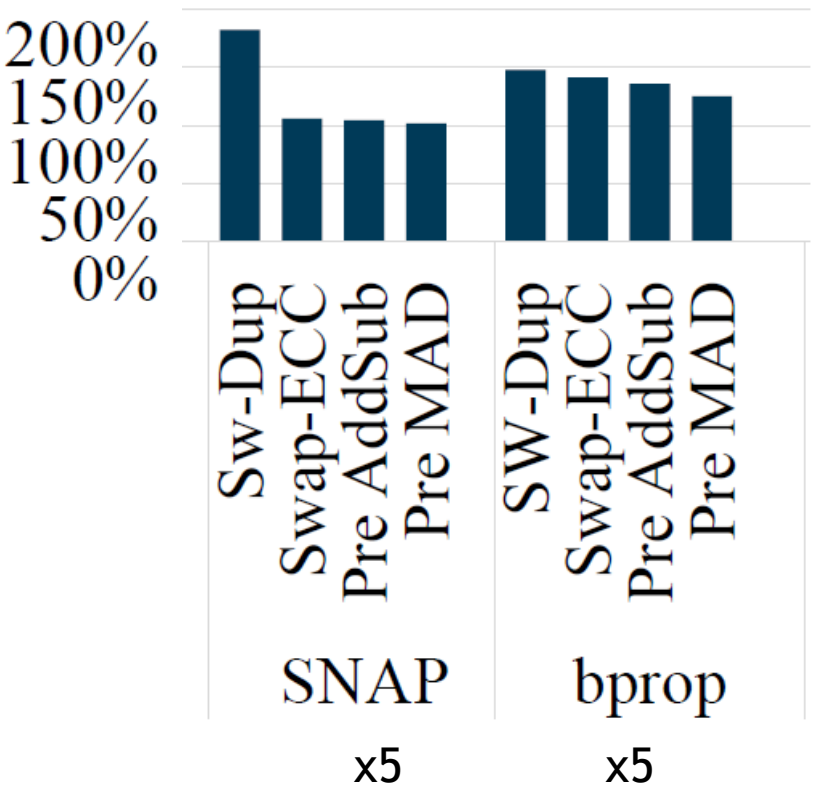
❶ Swap-ECC Responders:

High overhead with SW-Dup, Swap-ECC brings this to ~0.

Little further benefit from Swap-Predict.

Other Benchmarks: pathfinder, srad_v2, kmeans, needle

PERFORMANCE



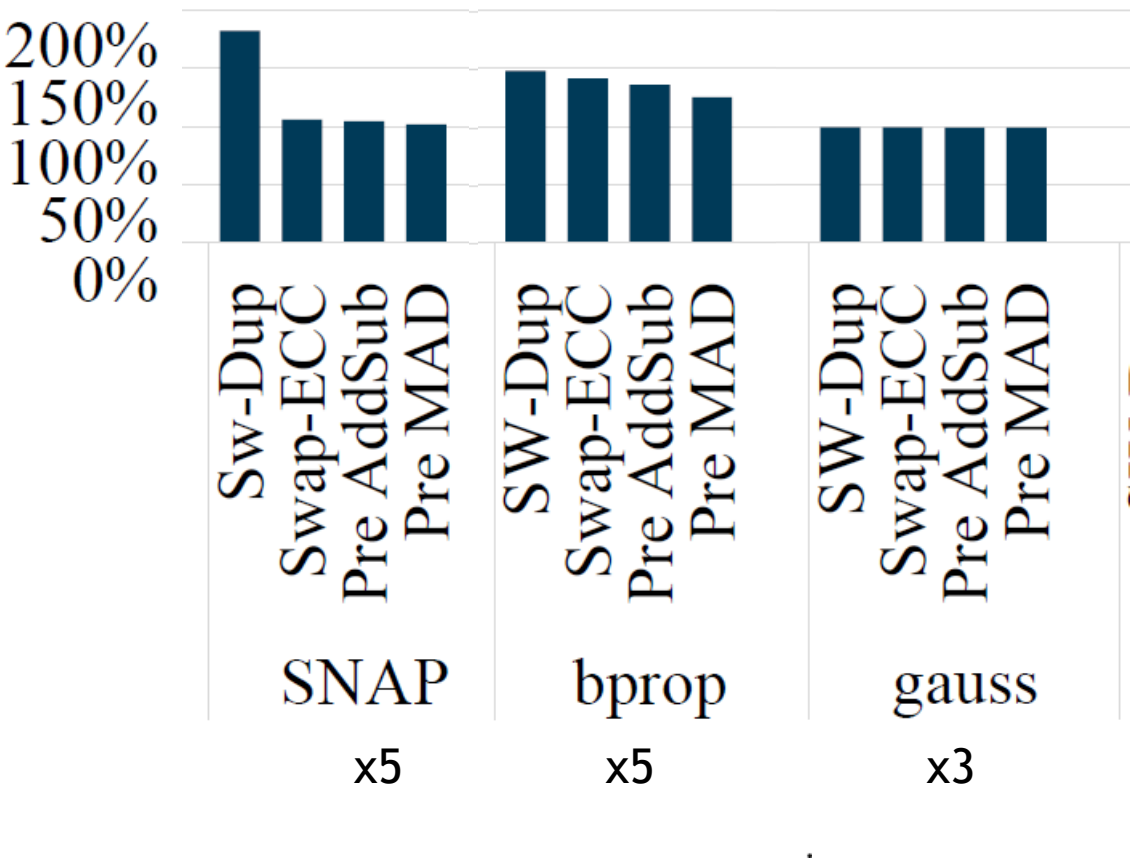
❷ Swap-Predict Responders:

SwapCodes helps.

Benefit progressively more with prediction.

Other Benchmarks: b+tree, lud, hotspot, heartwall

PERFORMANCE

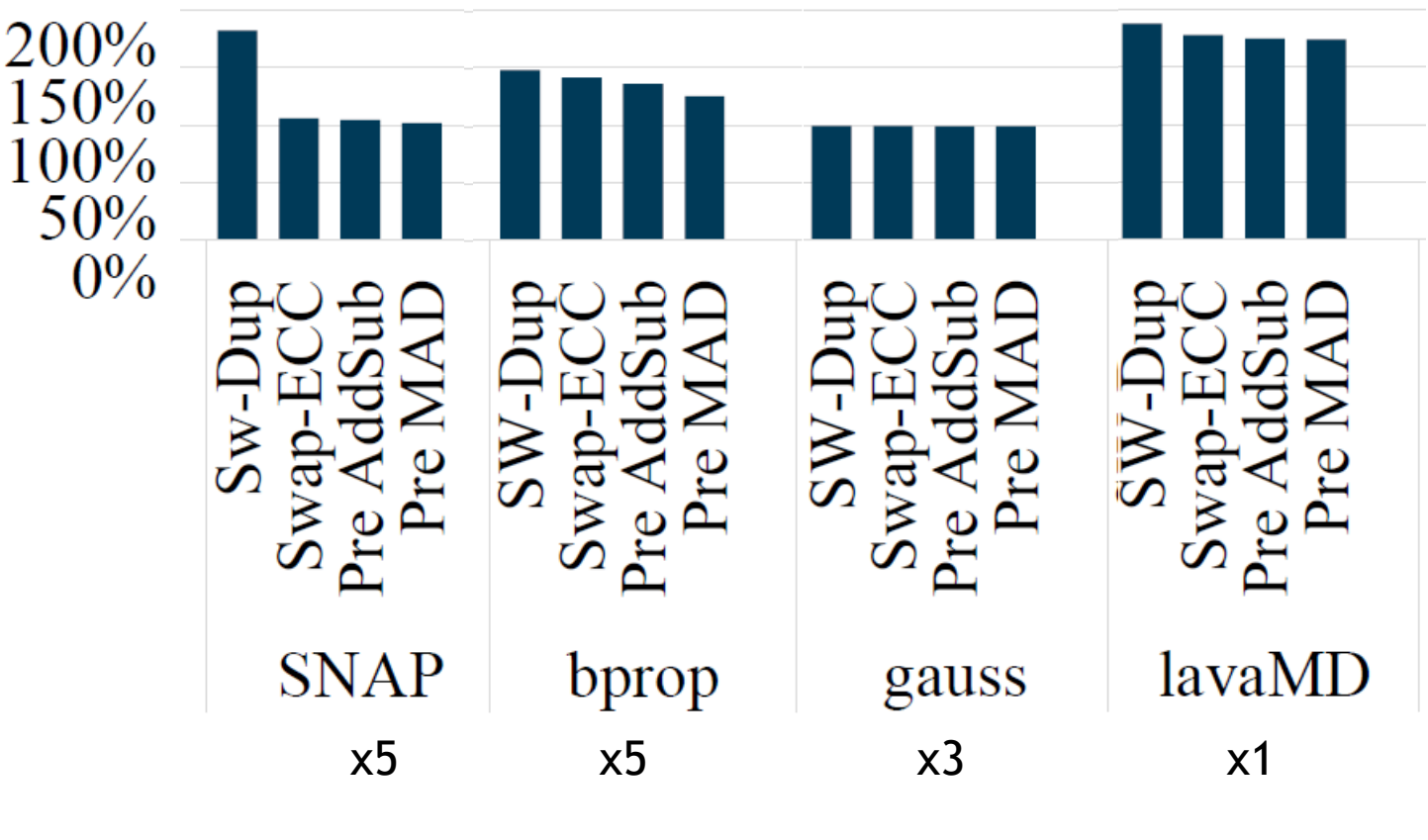


❸ Already Fine:

Even SW-Dup has ~0 overhead.

Other Benchmarks: mummer, bfs, MatMul (CUDA SDK)

PERFORMANCE



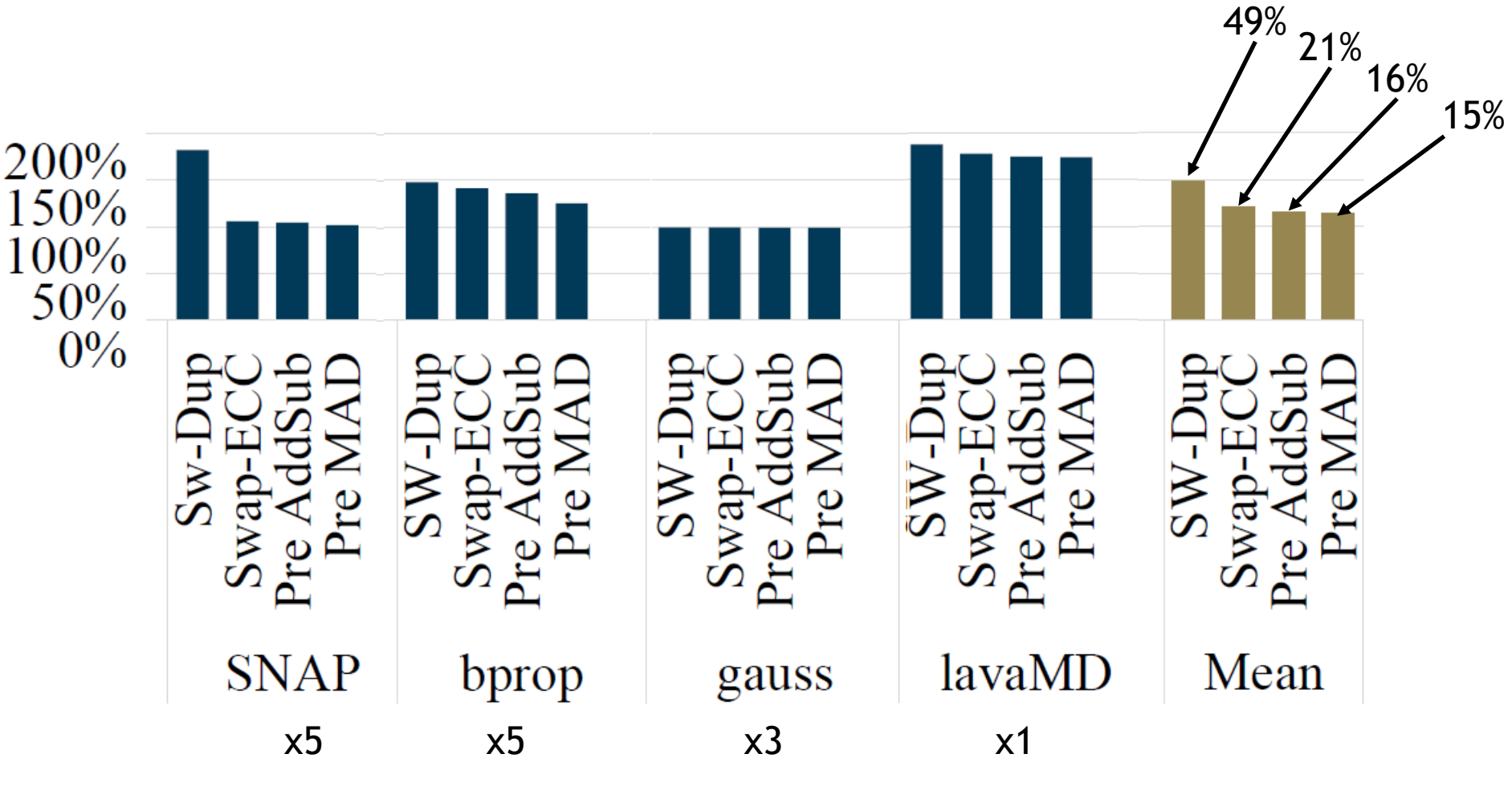
❹ Non-Responders:

Floating-point compute bound.

Modest improvements.

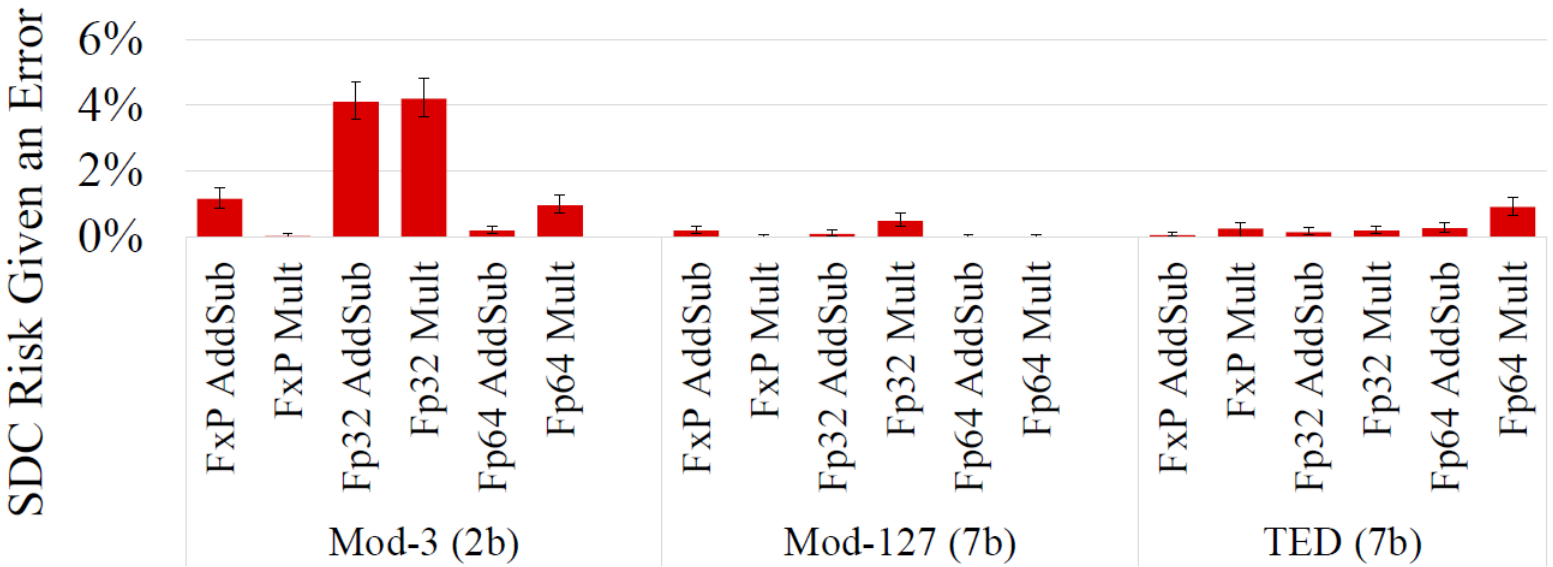
Other Benchmarks:
None

PERFORMANCE



RESILIENCE

Using Gate-Level Error Injection Into (Example) Compute Pipelines



>**98.8%** pipeline errors with a SEC-DED (TED) code.

>**99.3%** with an equal-redundancy residue code.

CONCLUSION

SwapCodes: Hardware-Software Acceleration for Instruction Duplication

Reduces the overhead of intra-thread instruction duplication by **>50%** on average, despite requiring:

- No new per-thread state
- No new hardware error checkers

With a standard SEC-DED protected register file, detects **>98.8%** of pipeline errors.

Swap-Predict can further optimize overheads to achieve just a **16%** performance overhead with check-bit prediction for addition and subtraction.

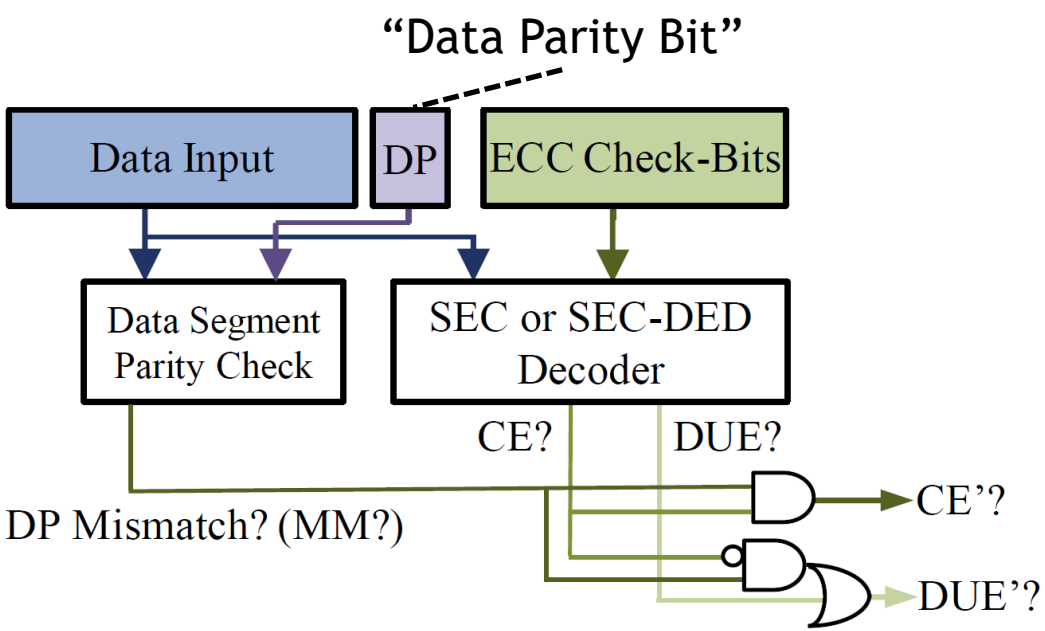
BACKUP

PIPELINE ERROR DETECTION OPTIONS

	High Level Duplication	Thread Duplication	Instruction Duplication	Concurrent Checking	SwapCodes
Transparent	No	No	Yes	Yes	**Yes**
H/W Changes	None	None	None	Many	**Few**
Perf. Overhead	High	High	Medium—High	None—Low	**Low**
Major Issue	Nondeteminism & Perf.	Threads & Perf.	Performance	Scope & Complexity	**None?**

Preserving Storage Correction

Two Flavors, Each with SEC-DED Protection Against Storage Errors



We describe two algorithms:

❶ SEC-DED-DP:

Adds a data parity bit to the SEC-DED decoder.

+ No constraints
– Requires a bit of redundancy

❷ SEC-DP:

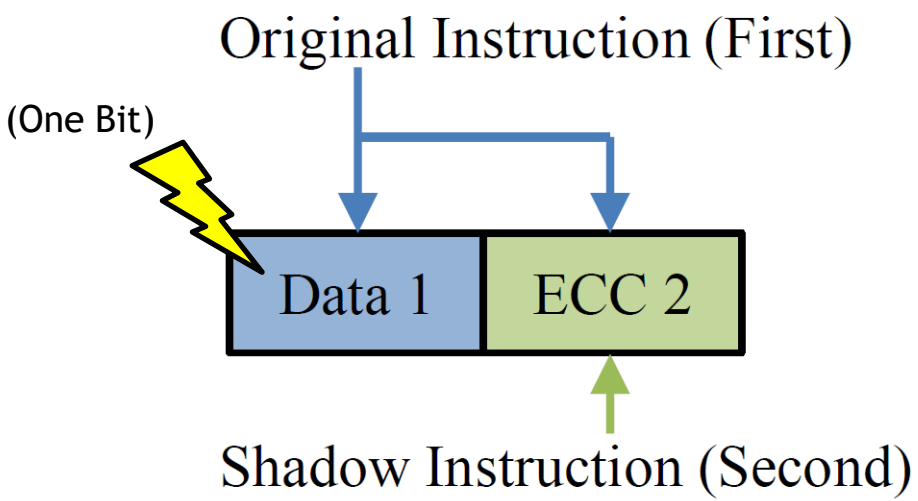
Adds a data parity bit to the SEC decoder, and uses selective bit placement to achieve DED.

+ No redundancy
– Constrains the RF bit placement

STORAGE CORRECTION

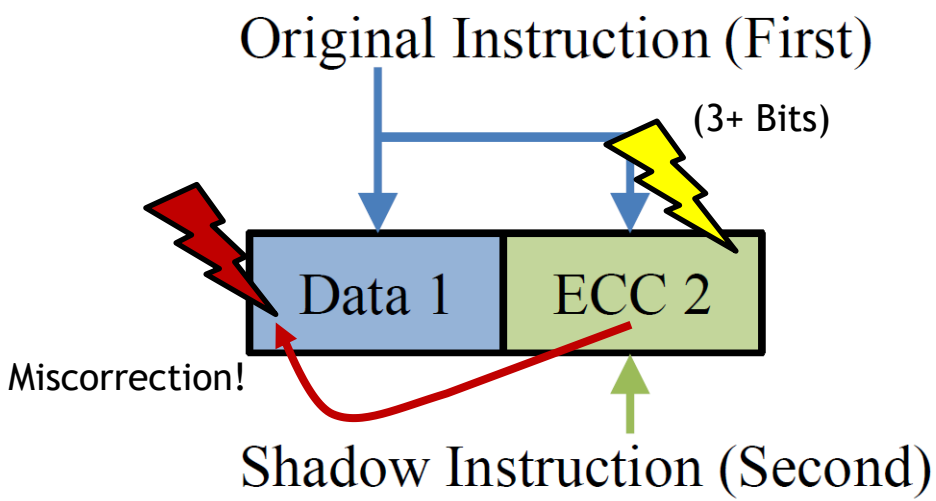
Challenge: Just Correcting All (Apparent) Single-Bit Errors is Not Safe!

❶ Error in Original (First) Instruction:



→ No problem, the compute error is corrected.

❷ Error in Shadow (Second) Instruction:



→ Correction of compute errors is unsafe.