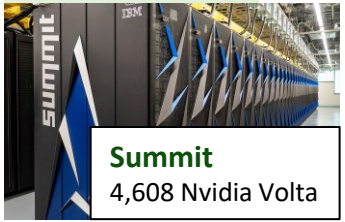
SUGAR: Speeding Up GPGPU Application Resilience Estimation with Input Sizing

Lishan Yang, Bin Nie, Adwait Jog, and Evgenia Smirni

William & Mary

GPUs & Soft Errors

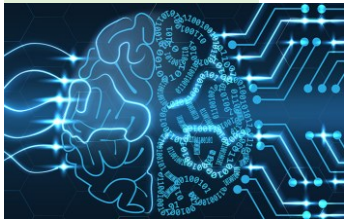
Supercomputing



Summit
4,608 Nvidia Volta

Selene
2,240 Nvidia Ampere

Deep learning



Self-driving cars

http://global.atpinc.com/Memory-insider/what-is-soft-error-detection-sram-emmc

- GPUs are commonly deployed
- GPUs are prone to **soft errors**
  - High-energy radioactive particles (i.e., cosmic rays) cause **bit flips**
  - Commonly observed
  - Impact on long-running applications can be tremendous
    - ***Masked*** output: Correct
    - ***Other*** outputs: Crash, hang, …
    - ***Silent Data Corruption (SDC)*** output: Incorrect
  - SDCs in critical applications can be dangerous

SDC → Misclassification

*Li, Guanpeng, et al. "Understanding error propagation in deep learning neural network (DNN) accelerators and applications." SC 2017.

Reliability Research: Fault Injection

- Inject **single-bit errors** into different locations (**fault sites**) in applications

<kernel_id, thread_id, instruction_id, bit_position>

- Ground truth: ***huge unreachable exhaustive fault sites!***

2DCONV

Input size	Small	Large
Num. of Elements	32×32	2048×2048
Num. of Threads	1024	4.19×10^6
Num. of Fault Sites	1.90×10^6	8.71×10^9
Execution Time (Simulation)	**1 sec**	$10\ min$

Fault site pruning

$$10 \text{ min} \times 440 = 1.2 \text{ h}$$

$$1.2 \text{ h} \times \infty \text{ inputs} = \infty \text{ h}$$

~~*Magic?*~~ **SUGAR** ✓

$$1 \text{ sec} \times 440 = 7.3 \text{ min}$$

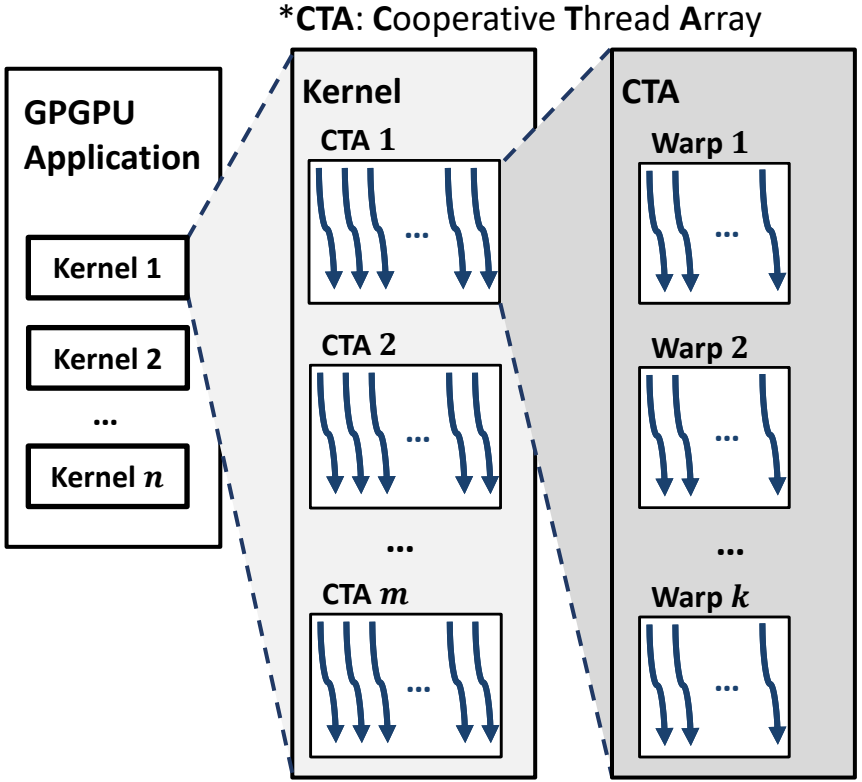
$$7.3 \text{ min} \times \infty \text{ inputs} = 7.3 \text{ min}$$

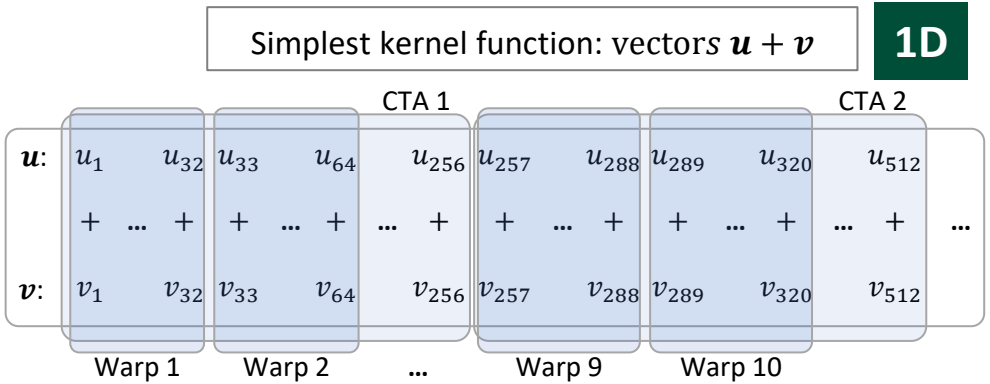
- Fault site selection:
  - **Random sampling** based on statistics: 1K runs, ±3% error
  - *(state-of-the-art)* **Fault site pruning***: 300~2K runs, ~1% error
    - Resilience proxy: Dynamic Instruction (DI) count

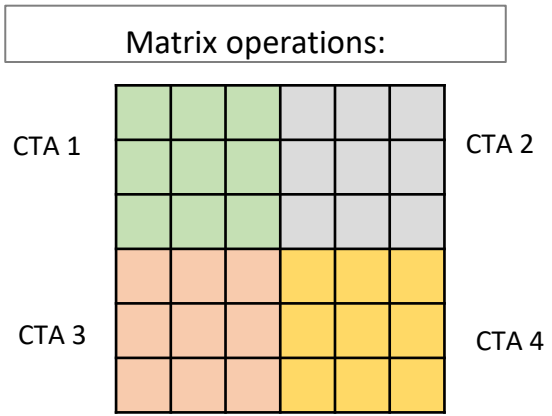
} **Input-dependent**

*B. Nie, et al. "Fault Site Pruning for Practical Reliability Analysis of GPGPU Applications". Micro 2018.

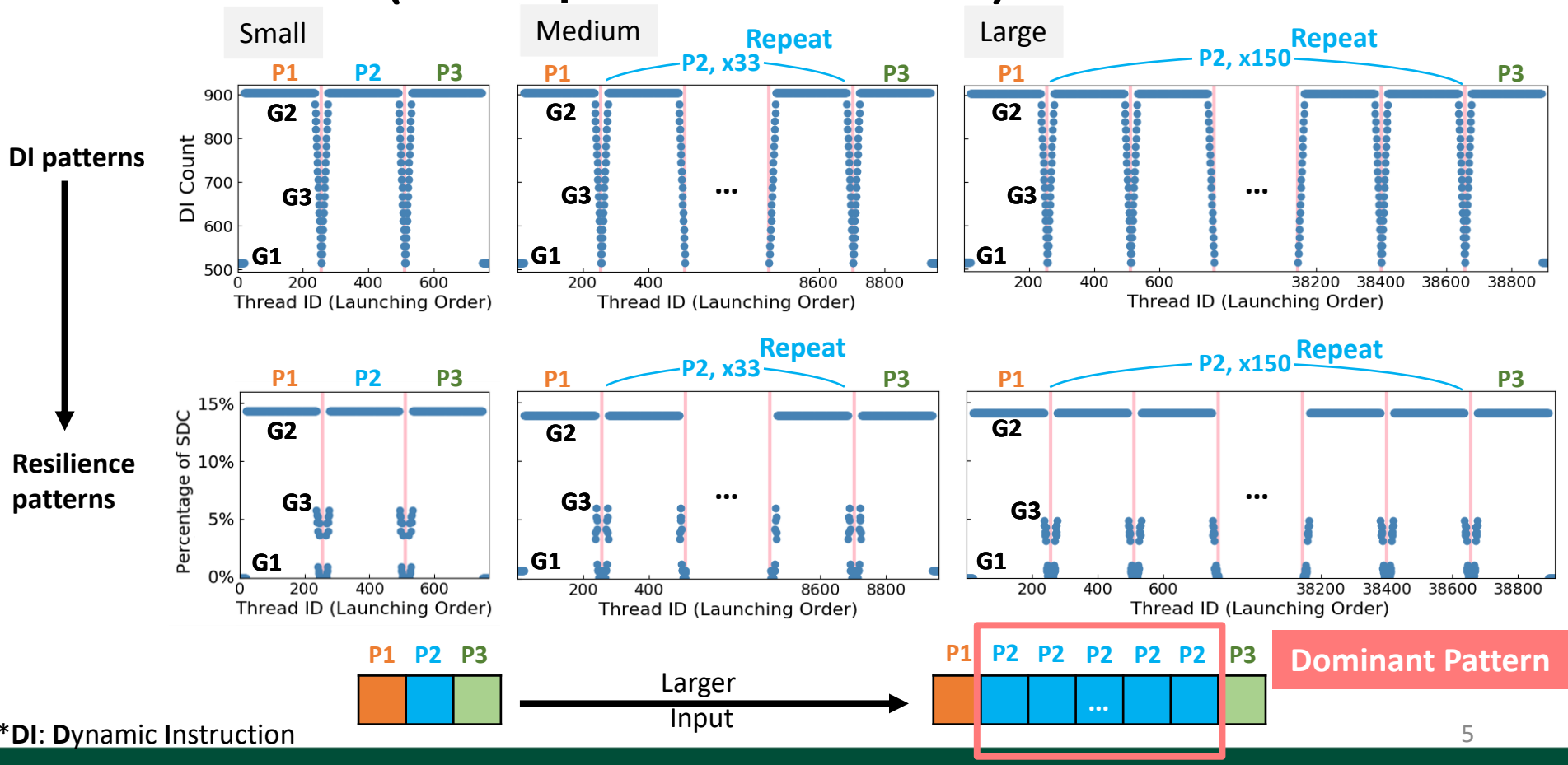
GPGPU Application Parallelization







SUGAR Idea (Example: PathFinder)



*DI: Dynamic Instruction

Why?

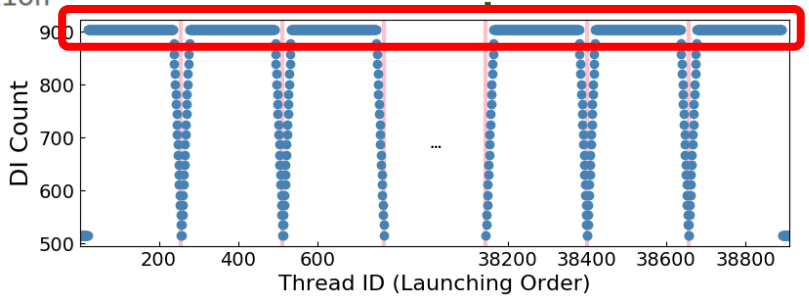
- Code snippet of PathFinder

Calculate Boundary

Trend of repeating patterns

✓Small → Large

```
1  int bx = blockIdx.x;
2  int tx = threadIdx.x;
3  int small_block_cols = BLOCK_SIZE-iteration*HALO*2;  // Data chunk size
4  int blkX = small_block_cols*bx-border;  // Chunk start position
5  int blkXmax = blkX+BLOCK_SIZE-1;  // Chunk end position
6  int xidx = blkX+tx;  // Global thread ID
7  int validXmin = (blkX < 0) ? -blkX : 0;  // Valid ch
8  int validXmax = (blkXmax > cols-1) ? \
9      BLOCK_SIZE-1-(blkXmax-cols+1) : BLOCK_SIZE-1;  /
10 ......
11 bool isValid = IN_RANGE(tx, validXmin, validXmax);
12 ......
13 for (int i=0; i<iteration ; i++){
14   ......
15   if( IN_RANGE(tx, i+1, BLOCK_SIZE-i-2) && isValid){
16     // If valid, take the branch.
17     ..
18     }
19 }
```



Check Validation

Only valid threads perform computation & touch the data

✓DI Patterns → Resilience Patterns

Input does NOT change branch divergence

DI-insensitive

What if input changes branch divergence? 🤔

- Code snippet of BFS K8

```
1 int tid = blockIdx.x*MAX_THREADS_PER_BLOCK + threadIdx.x;
2 if( tid<no_of_nodes && g_updating_graph_mask[tid])
3 // g_updating_graph_mask is calculated by the previous kernel based on the input.
4 {
5     g_graph_mask[tid]=true;
6     g_graph_visited[tid]=true;
7     *g_over=true;
8     g_updating_graph_mask[tid]=false;
9 }
```

Check Validation

Input data

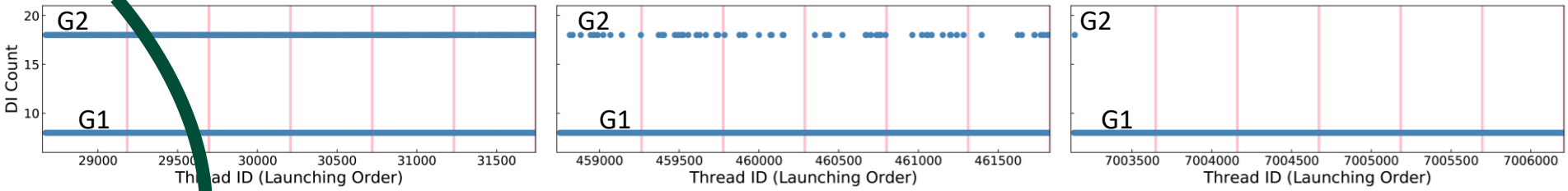
Input changes branch divergence

DI-sensitive

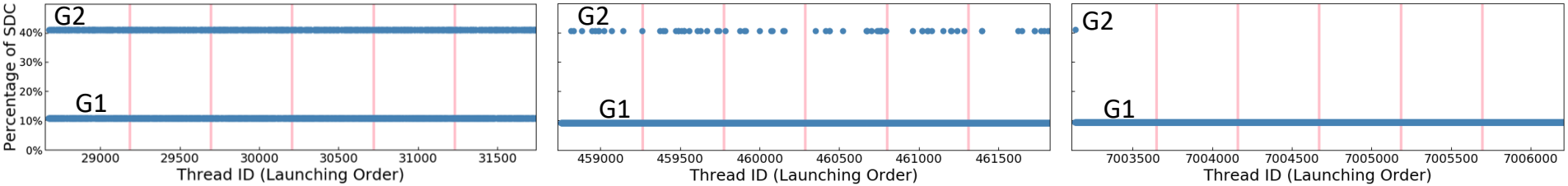
DI-sensitive Patterns (Example: BFS)

Small → (+DI Profiling) → Medium → (+DI Profiling) → Large

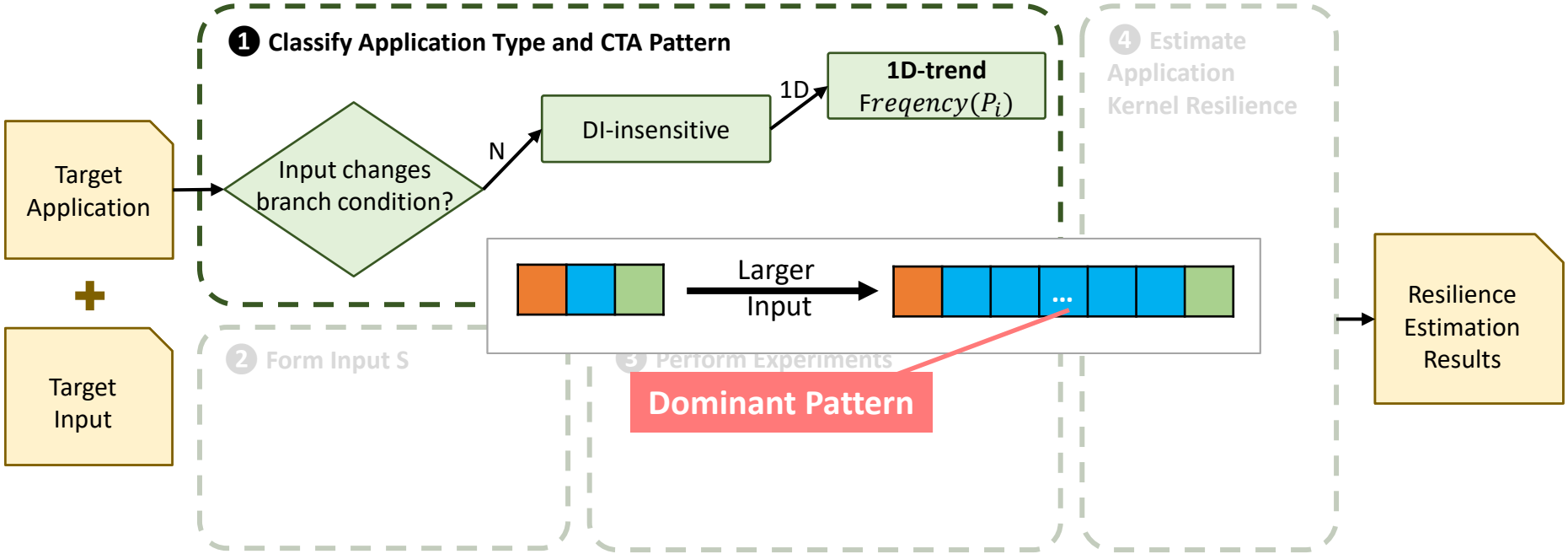
- DI patterns:



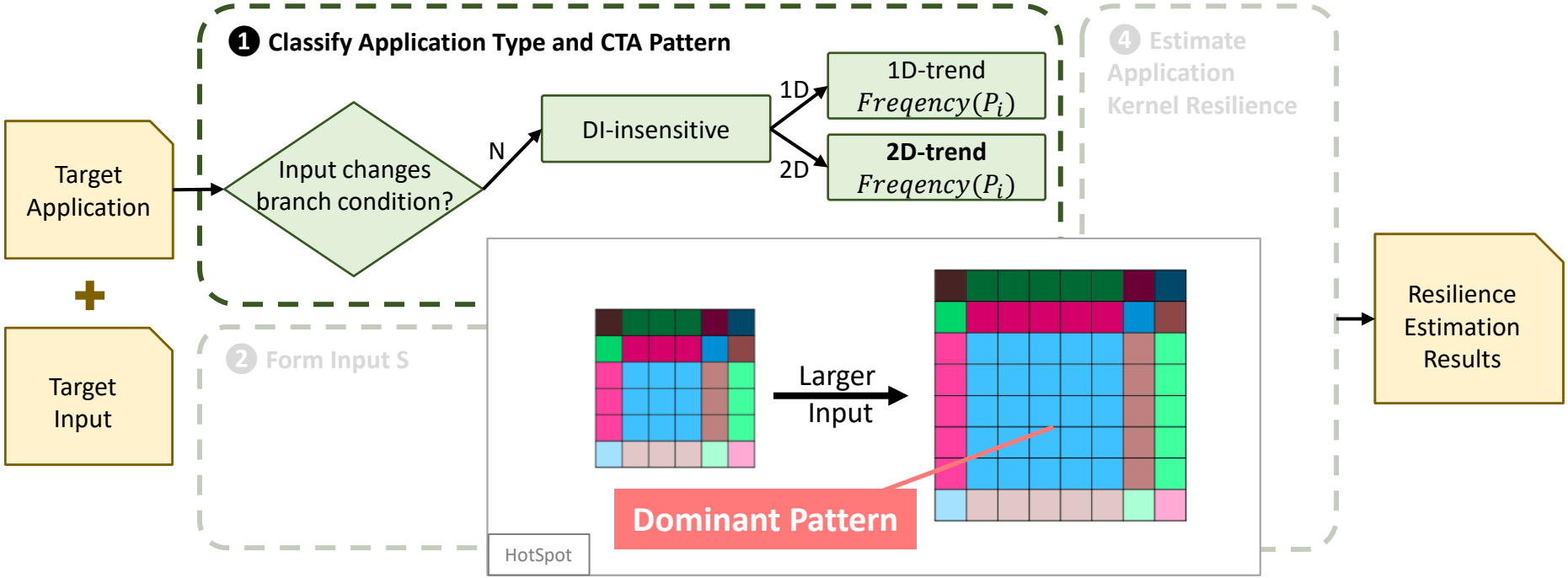
- Resilience patterns:



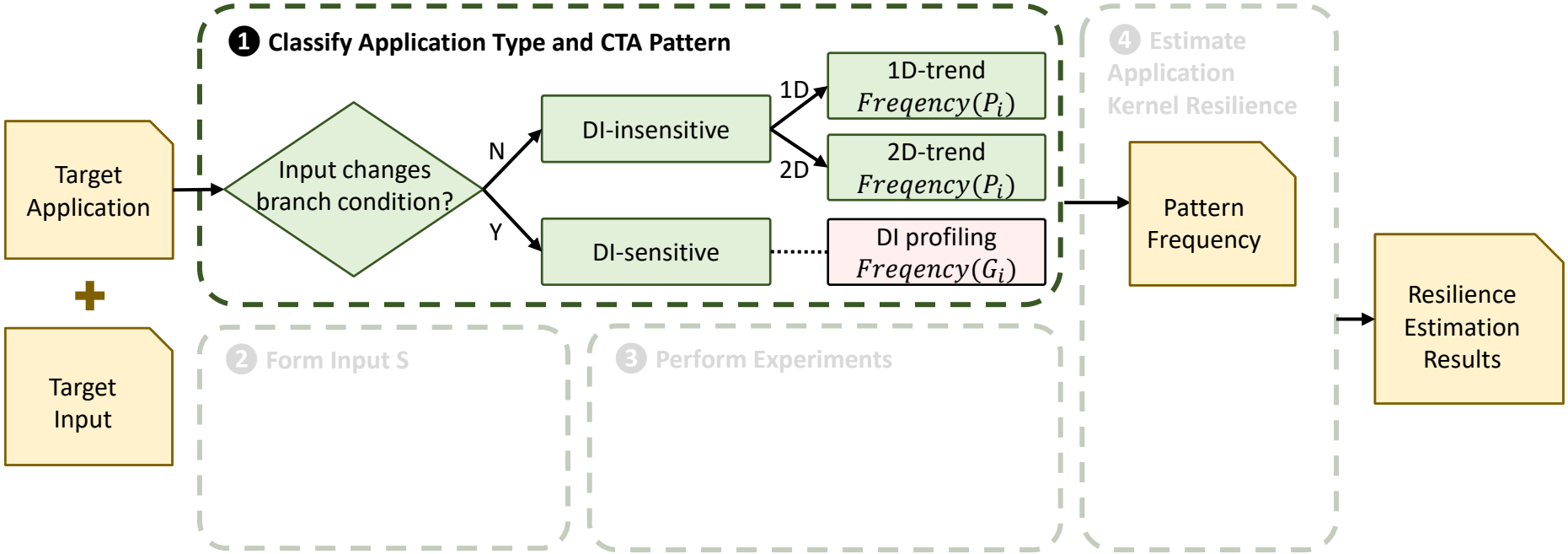
SUGAR Workflow



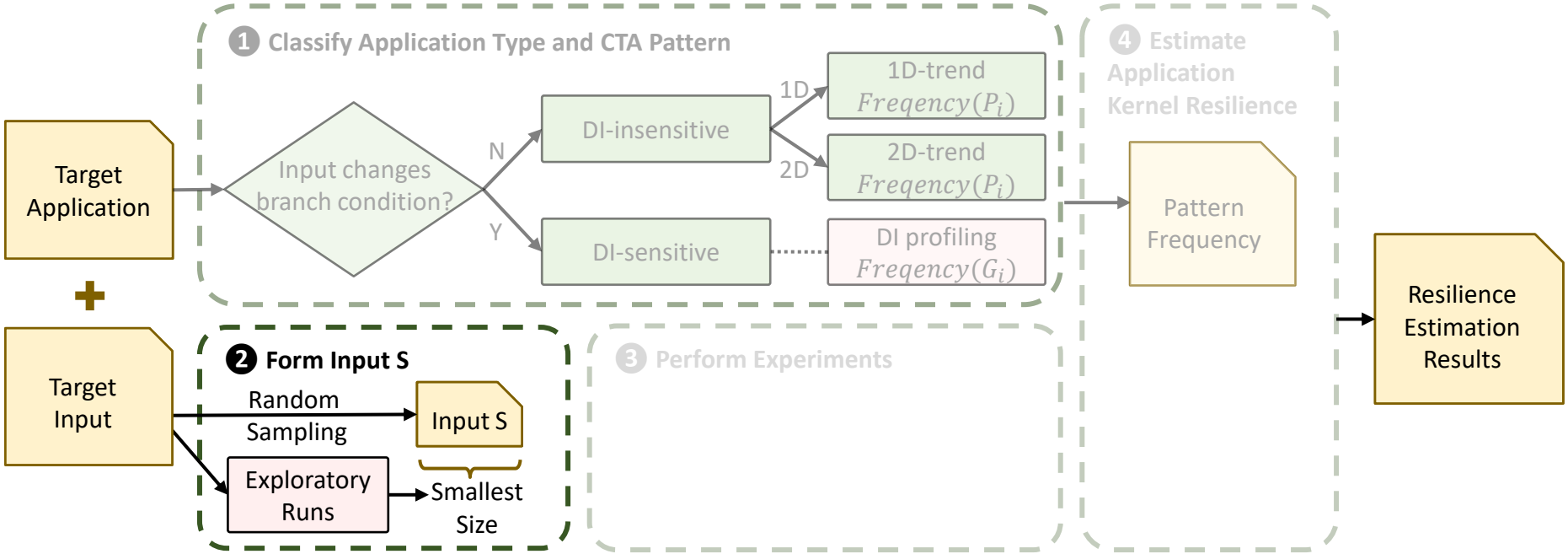
SUGAR Workflow



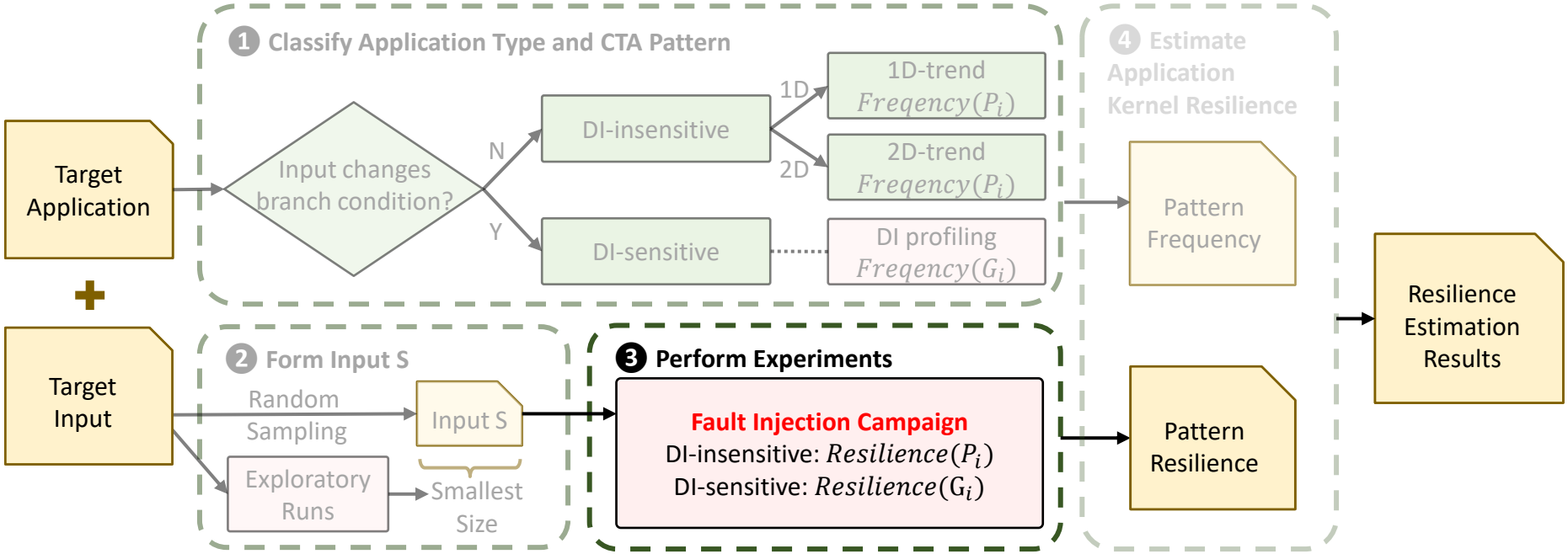
SUGAR Workflow



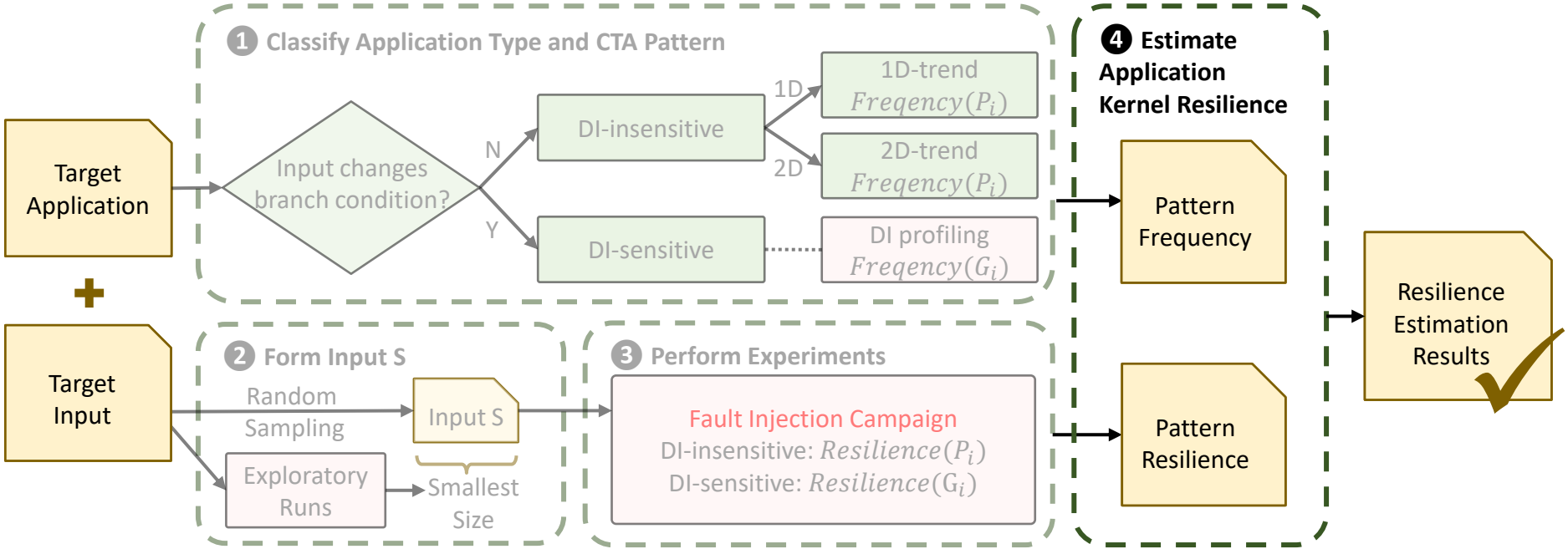
SUGAR Workflow



SUGAR Workflow

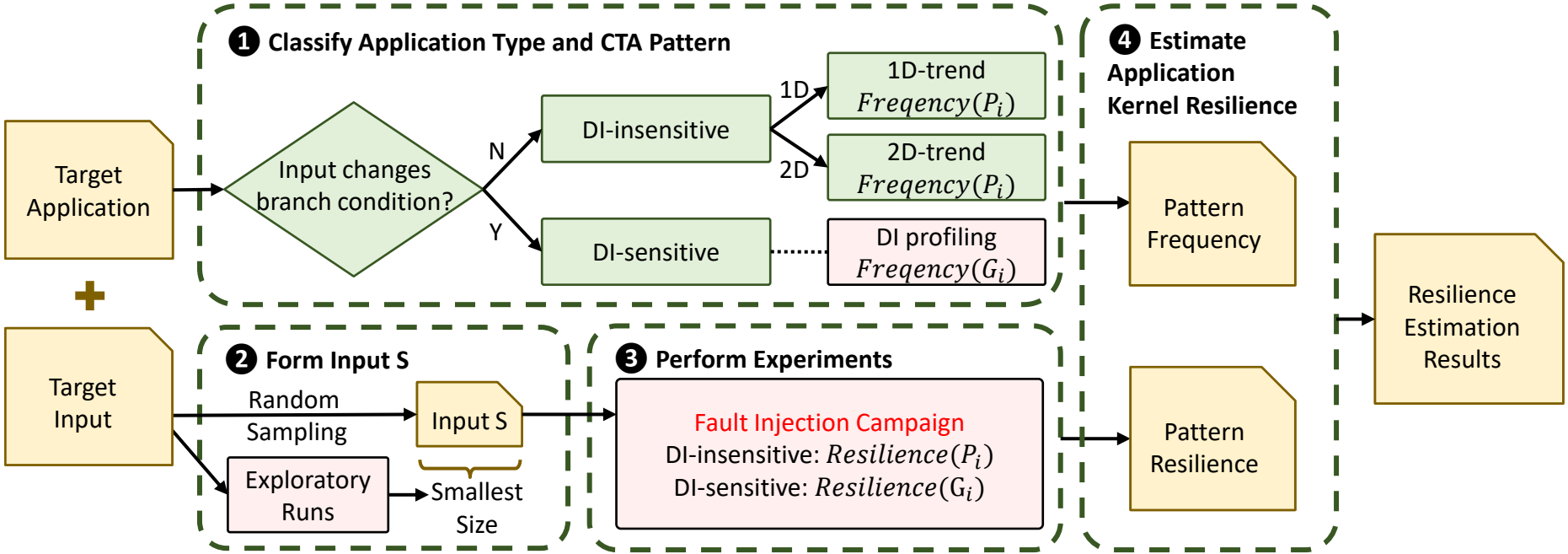


SUGAR Workflow



$$Resilience(Kernel) = \begin{cases} \sum_{i=1}^{n} Resilience(P_i) \times Frequency(P_i) & \text{DI-insensitive} \\ \sum_{i=1}^{n} Resilience(G_i) \times Frequency(G_i) & \text{DI-sensitive} \end{cases}$$

SUGAR Workflow

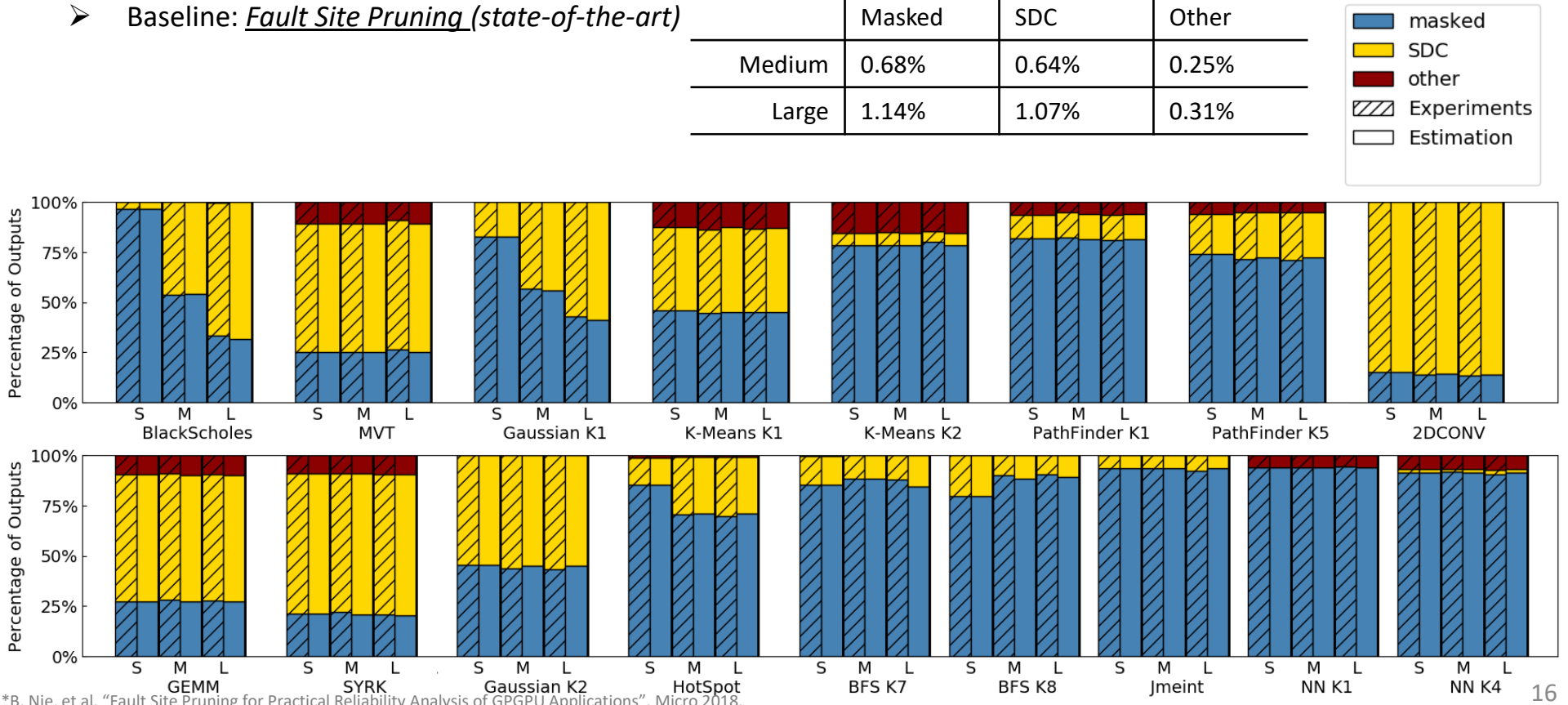


Evaluation: Accuracy

- Baseline: *Fault Site Pruning (state-of-the-art)*

Average Difference

	Masked	SDC	Other
Medium	0.68%	0.64%	0.25%
Large	1.14%	1.07%	0.31%



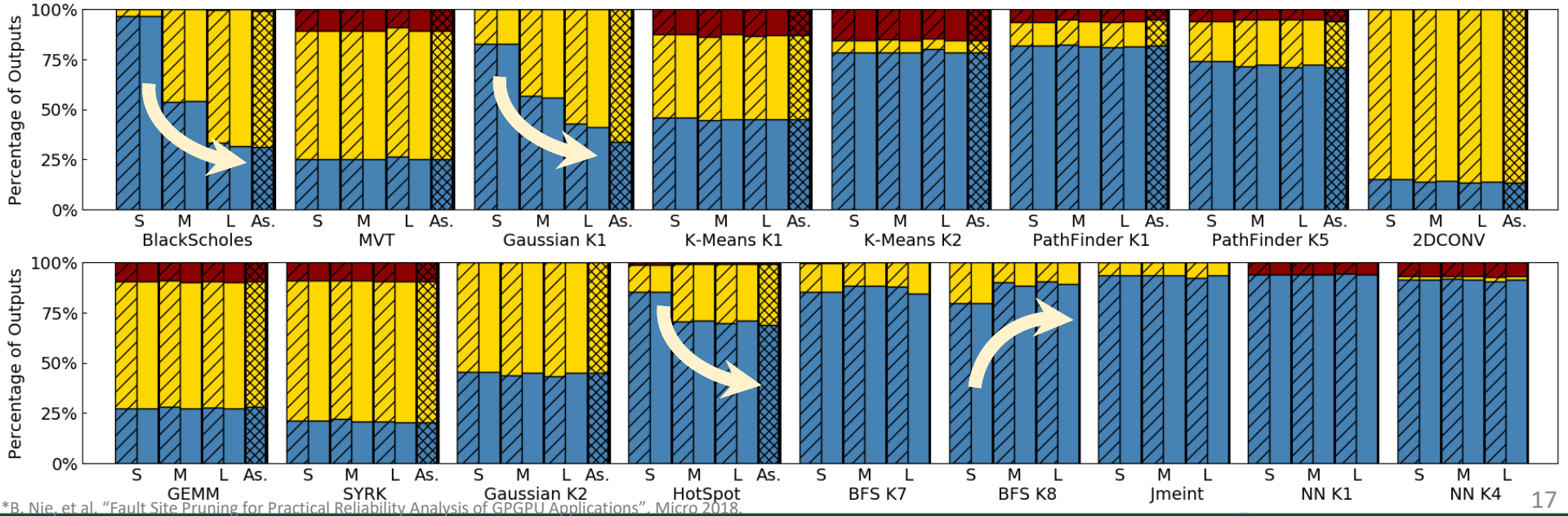
*B. Nie, et al. "Fault Site Pruning for Practical Reliability Analysis of GPGPU Applications", Micro 2018.

Evaluation: Accuracy

- Baseline: *Fault Site Pruning (state-of-the-art)*
- Asymptote resilience estimation
- Resilience trend:

Average Difference

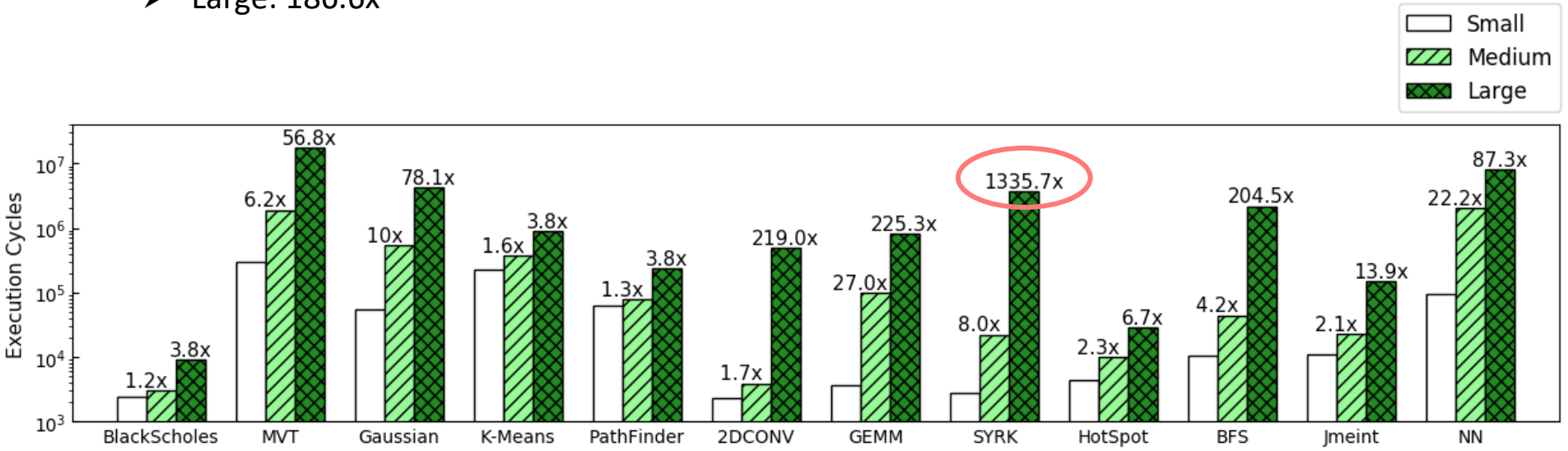
	Masked	SDC	Other
Medium	0.68%	0.64%	0.25%
Large	1.14%	1.07%	0.31%
All	0.89%	0.83%	0.28%



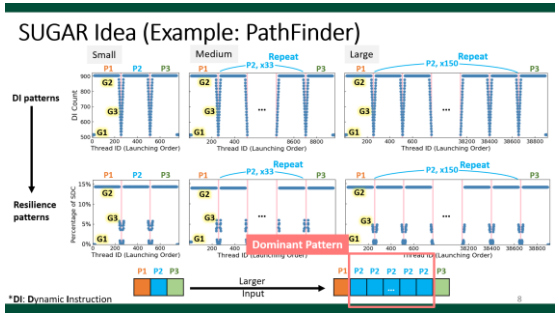
*B. Nie, et al. "Fault Site Pruning for Practical Reliability Analysis of GPGPU Applications", Micro 2018.

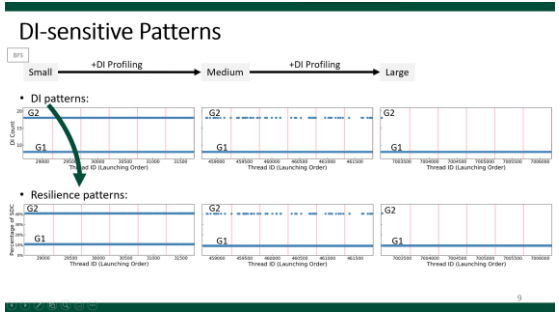
Evaluation: Speedup

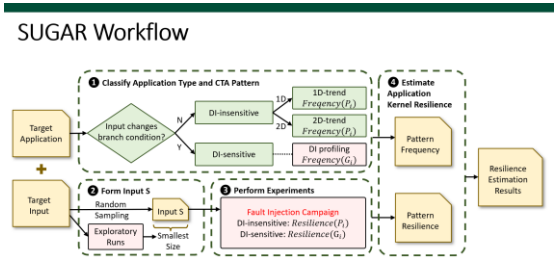
- Average speedup:
  - Medium: 7.3x
  - Large: 186.6x



SUGAR Summary



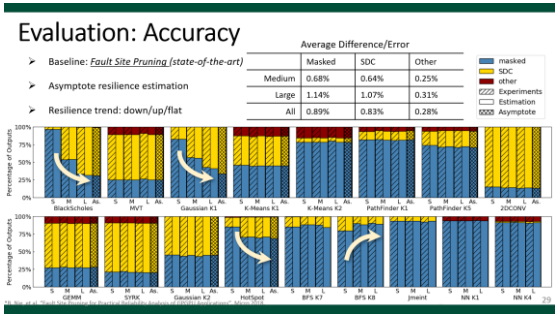


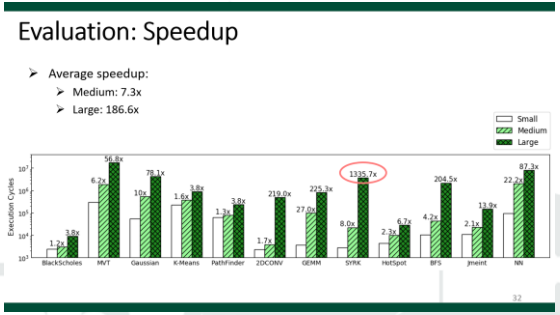


Magic? ~~Magic?~~ **SUGAR** ✓

$1\ \text{sec} \times 440 = 7.3\ \text{min}$

$7.3\ \text{min} \times \infty\ \text{inputs} = 7.3\ \text{min}$





✓**DI Patterns ⟶ Resilience Patterns**

✓**Small ⟶ Large**

✓**Accurate**

✓**Fast**

WILLIAM & MARY
CHARTERED 1693

Thank you :)

SUGAR: Speeding Up GPGPU Application Resilience Estimation with Input Sizing

Lishan Yang, Bin Nie, Adwait Jog, and Evgenia Smirni
William & Mary

This work is supported by NSF grant CCF-1717532