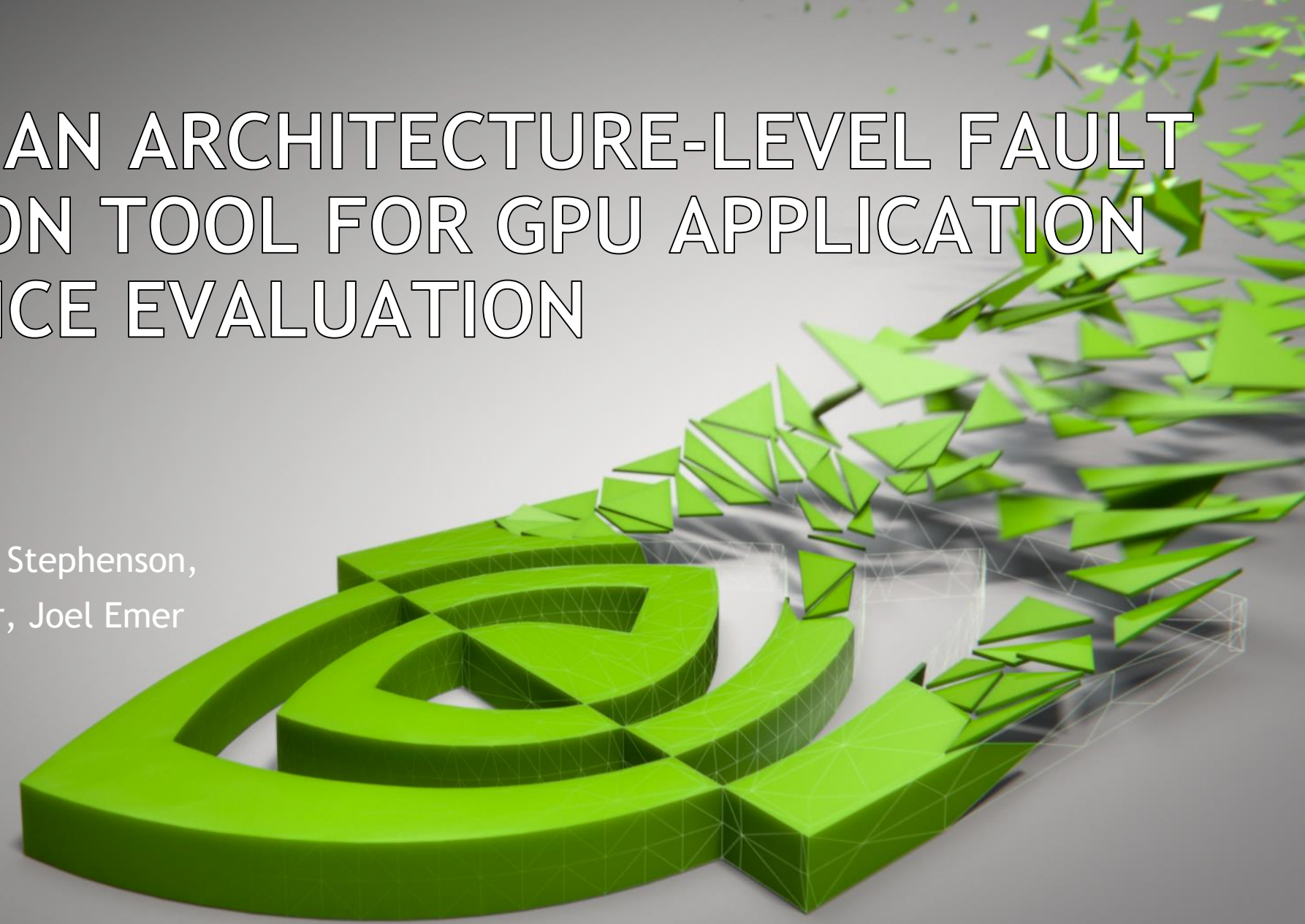
SASSIFI: AN ARCHITECTURE-LEVEL FAULT INJECTION TOOL FOR GPU APPLICATION RESILIENCE EVALUATION

Siva Hari

Timothy Tsai, Mark Stephenson,

Stephen W. Keckler, Joel Emer



MOTIVATION

Automotive and HPC systems need high resilience

Need to evaluate resilience of applications

Silent Data Corruption (SDC), Detected Unrecoverable Error (DUE) probabilities

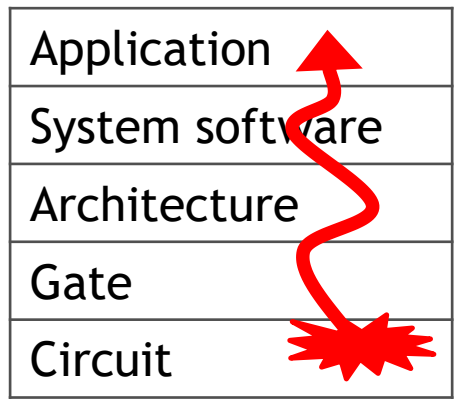
Identify vulnerable program sections – key for developing low-cost mitigation schemes

CHALLENGES

Application-level evaluation can be slow

Application-level resilience evaluation is challenging

- Traditional low-level error injection experiments are slow
- Low visibility into application behavior



Need quicker GPU application resilience evaluation scheme

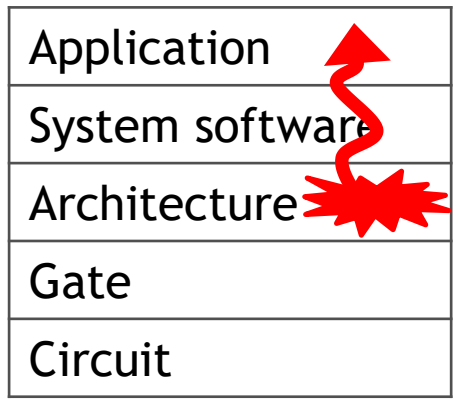
APPROACH

Architecture-level Error Injections

Inject error at architecture level

- Fast and visibility into application

Leverage a low-level assembly-language instrumentation tool (SASSI)



Advantages:

- Analyze and study SDCs in detail: Magnitude of SDCs and which errors produce SDCs
- Ability to correlate program properties with program vulnerability
  - Key to develop low cost error mitigation schemes
- Ability to quantify application level error derating factors

CONTRIBUTIONS

SASSIFI: Architecture-level GPU fault injection tool

Developed SASSIFI tool

Flexible options to inject many types of errors

Examples: single, multiple bit flips in register values; address vs. value errors

Demonstrated by conducting four types of resilience studies

Released SASSIFI for public usage

GitHub: https://github.com/NVlabs/sassifi

OUTLINE

Background: SASSI

SASSIFI tool

- Error injection methodology
- Use cases: Error models

Results

OVERVIEW OF SASSI

Background

SASSI is a compiler-based instrumentation framework that allows us to inject code before or after specific points in a program

Example: Identify all SASS memory ops and inject code needed to pass op's address to a user-defined function

```
L_8:
    ISCADD R7, R5, R3, 0x
    STS [R7], R2;
    BAR.SYNC 0x0;
    MOV R0, c[0x0][0x28];
    SHF.R R0, R0, 0x1, RZ;
    ISETP.EQ.AND P0, PT, ...
    @P0 BRA `(.L_12);
```

```
L_8:
    ISCADD R7, R5, R3, 0x2;
    IADD R1, R1, -0x4;
    STL [R1], R4;
    IADD R4, R7, 0x0;
    JCAL `(_users_function);
    LDL R4, [R1];
    IADD R1, R1, 0x4;
    STS [R7], R2;
    BAR.SYNC 0x0;
```

1. Create extra stack space
2. Save live registers
3. Pass parameters of interest to user-defined function
4. Call user-defined function
5. Restore live registers
6. Restore stack
7. Execute instrumented instruction

User writes a handler function, _users_function, in CUDA

"Flexible Software Profiling of GPU Architectures," Mark Stephenson, Siva Hari, Yunsup Lee, Eiman Ebrahimi, Daniel Johnson, Dave Nellans, Mike O'Connor, and Steve Keckler, ISCA 2015

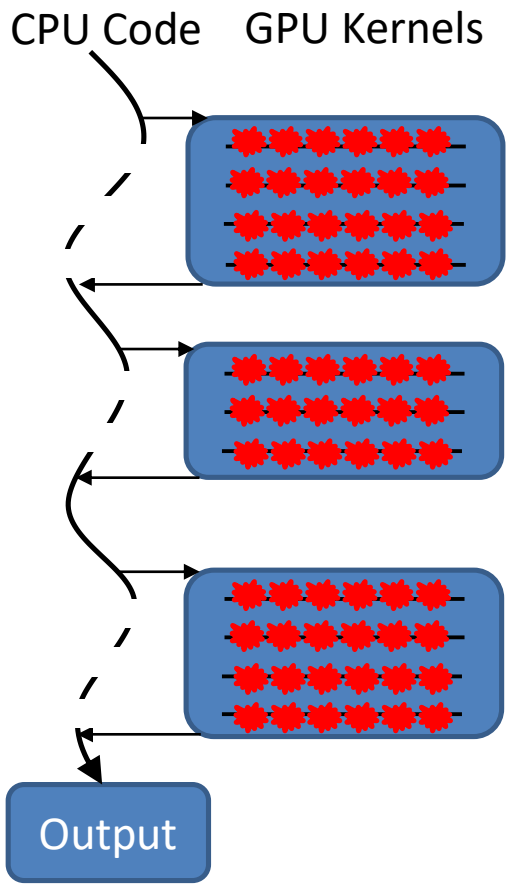
SASSIFI: SASSI BASED FAULT INJECTOR

Leveraged SASSI for error injections

Instrumented kernels for profiling and error injections

SASSIFI METHODOLOGY

Steps

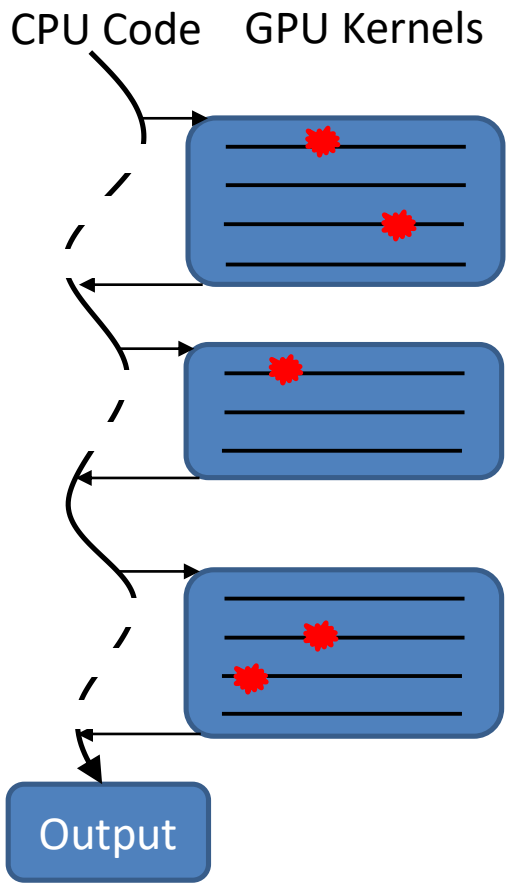


Profile: Identify possible injection sites

Instrumented kernels execute on the GPU

SASSIFI METHODOLOGY

Steps



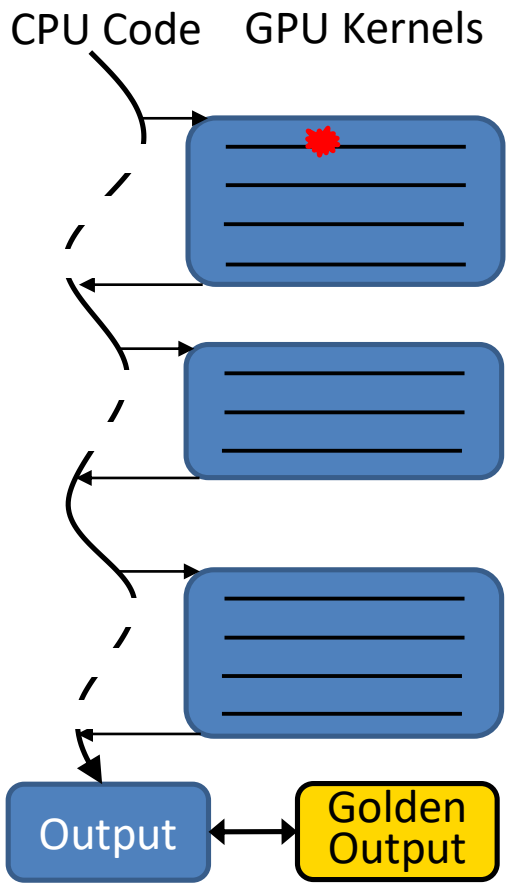
Profile: Identify possible injection sites

Instrumented kernels execute on the GPU

Statistically select injection sites based on the error model

SASSIFI METHODOLOGY

Steps



Profile: Identify possible injection sites

- Instrumented kernels execute on the GPU

Statistically select injection sites based on the error model

Injection runs: inject one error at a time

- Instrument before/after instructions and collect reg/mem info
- Start application, inject error at the selected site
- Continue execution until a crash or the output

OUTCOME CATEGORIES

Categories	Explanation
DUE	Application exits with non-zero exit status Application does not terminate within allocated time (3x fault-free runtime)
Potential DUEs	Kernel exit status is not cudaSuccess Error messages in *stdout*/stderr (e.g., Error: misaligned address)
SDC	Program output file or *stdout* is different
Masked	Application output is same as the error free output without any error symptoms

SASSIFI USE CASES

Many uses of SASSIFI

What is the probability that a particle-strike in the register file produce an SDC?

What is the probability that a bit-flip in the destination register of an executing instruction will result in an SDC?

What instruction types are likely to produce more SDCs when subjected to errors in destination registers?

How do SDC probabilities change when different architecture-level states (addresses vs. values) are subjected to errors?

How do the results change if we inject different bit-flip patterns (single vs. double bit-flips)?

SASSIFI USE CASES

Many uses of SASSIFI

What is the probability that a particle-strike in the register file produce an SDC?

What is the probability that a bit-flip in the destination register of an executing instruct

What ir errors in destina

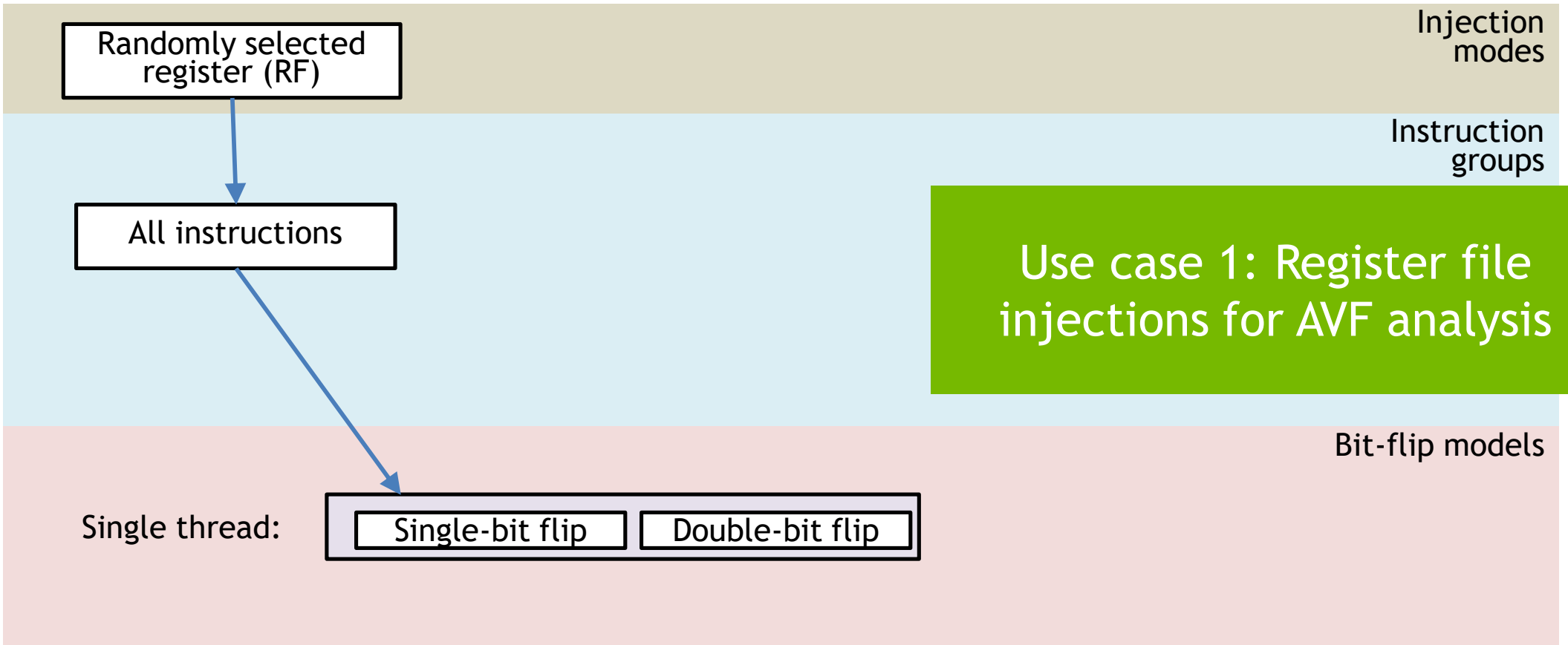
How do (addresses vs. valu

How do the results change if we inject different bit-flip patterns (single vs. double bit-flips)?

SASSIFI can be used to address all these questions

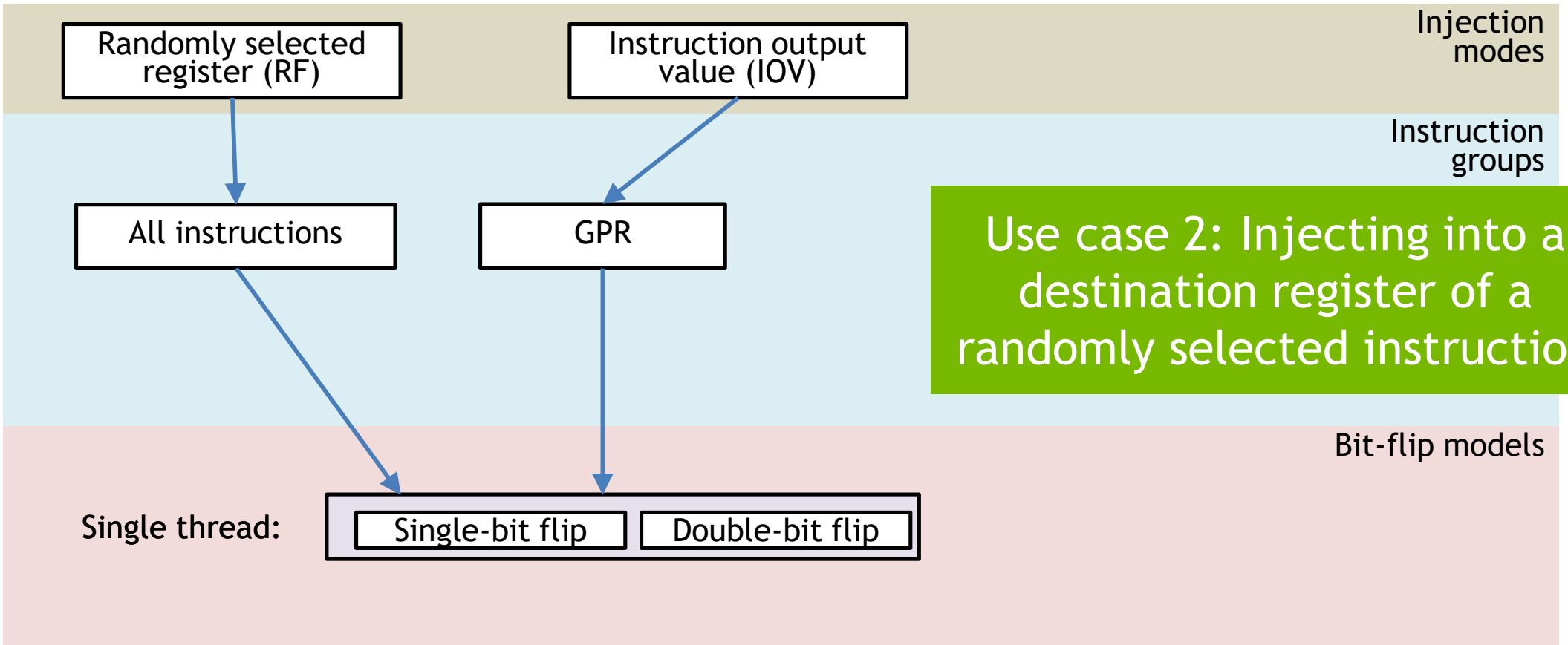
ERROR MODELS

SASSIFI can inject many types of errors



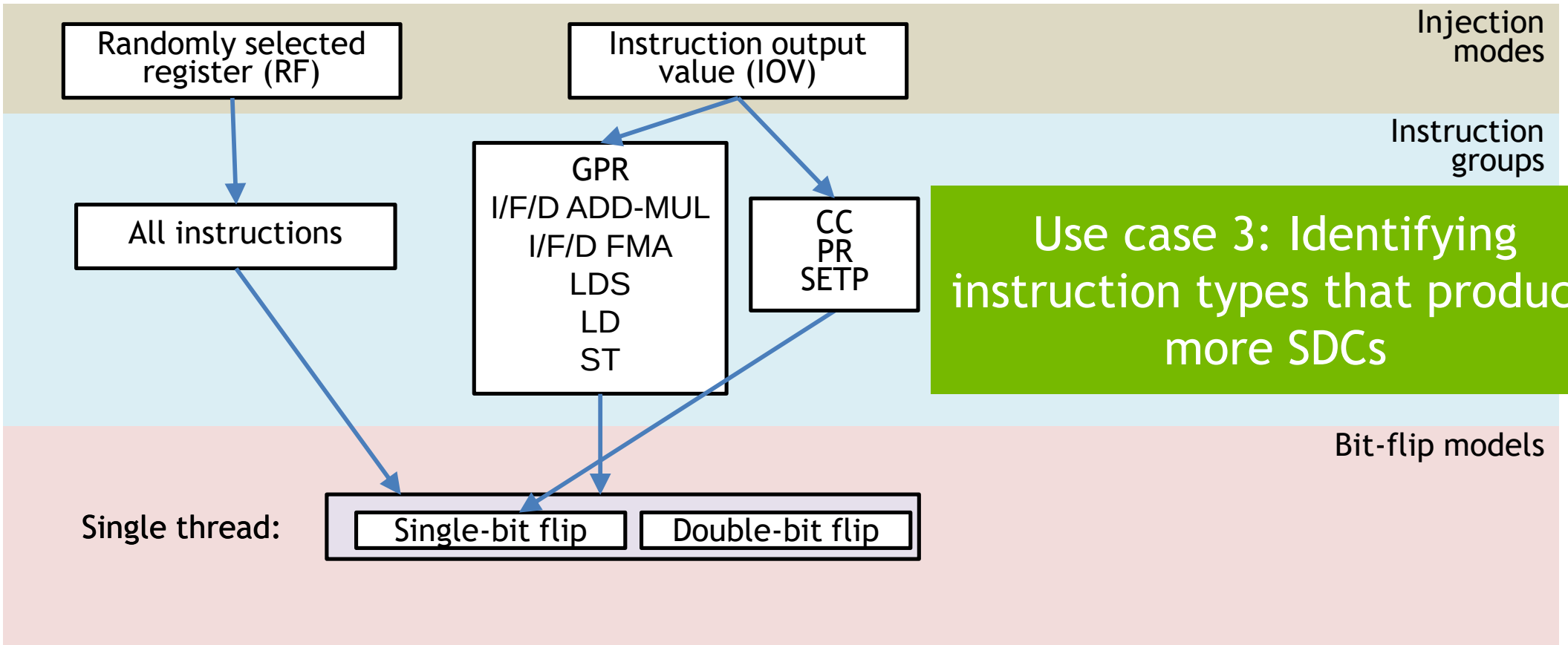
ERROR MODELS

SASSIFI can inject many types of errors



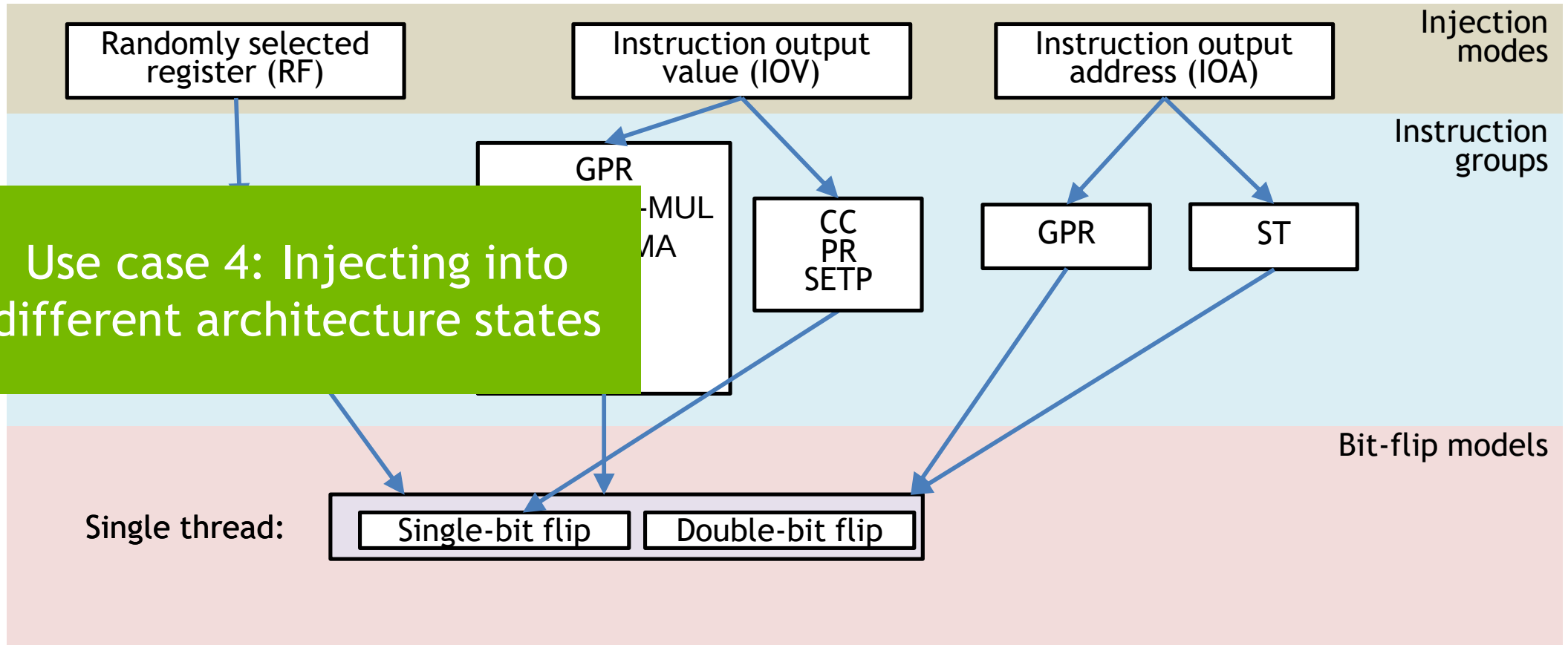
ERROR MODELS

SASSIFI can inject many types of errors



ERROR MODELS

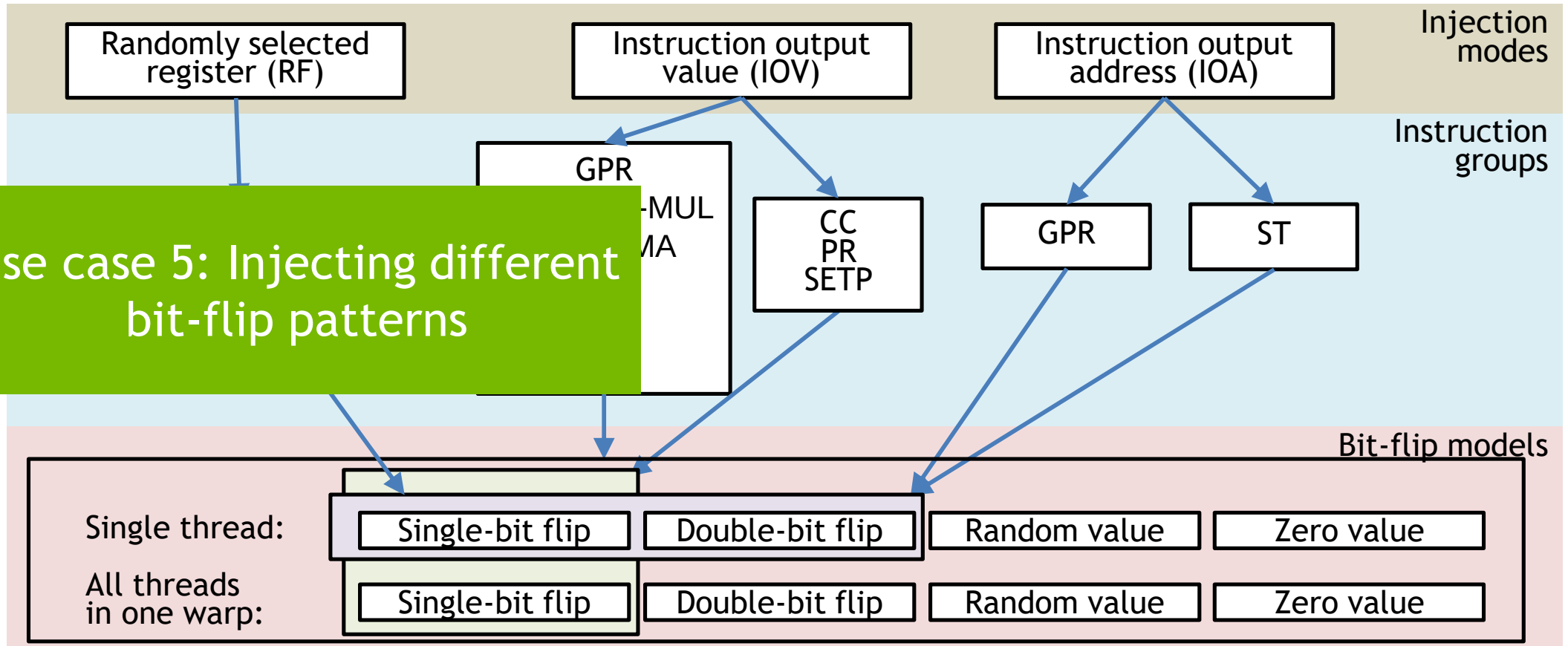
SASSIFI can inject many types of errors



Use case 4: Injecting into different architecture states

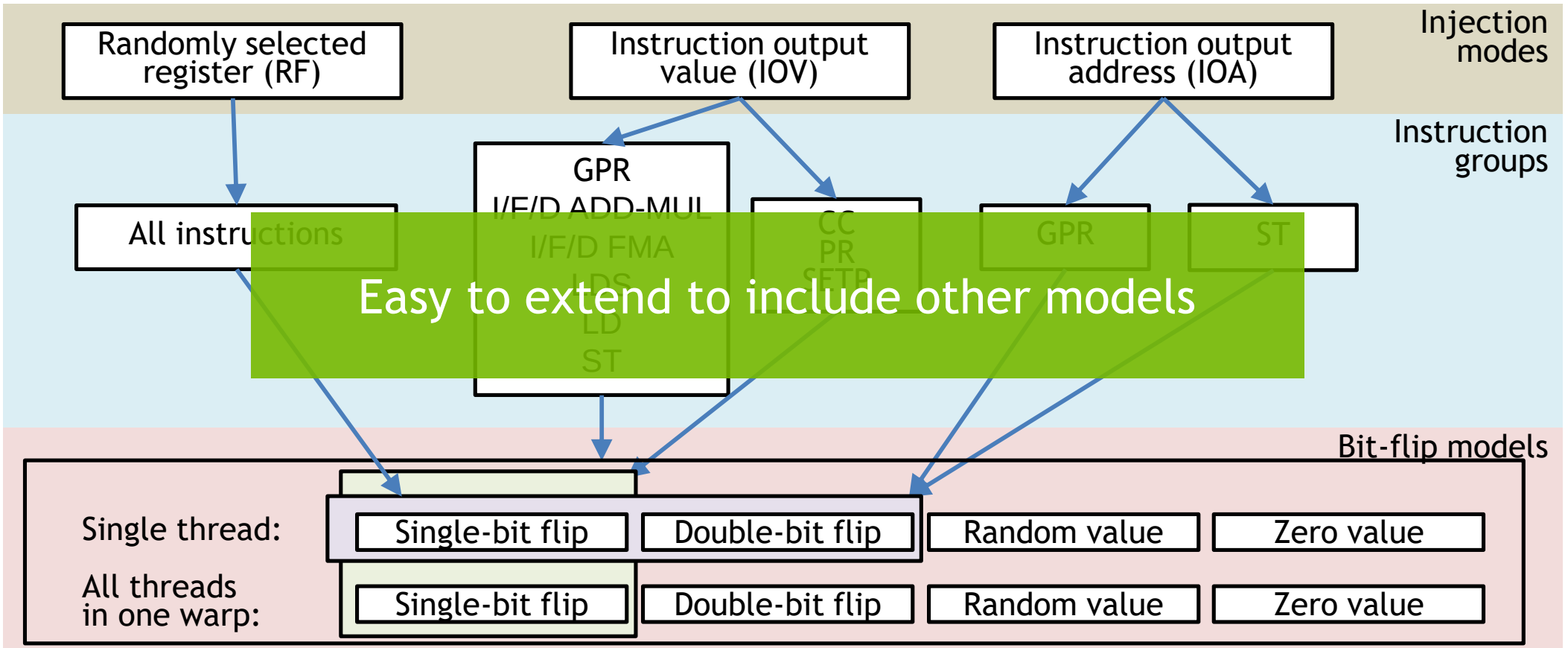
ERROR MODELS

SASSIFI can inject many types of errors

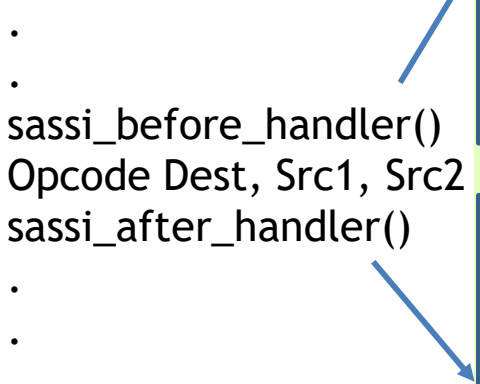


ERROR MODELS

SASSIFI can inject many types of errors



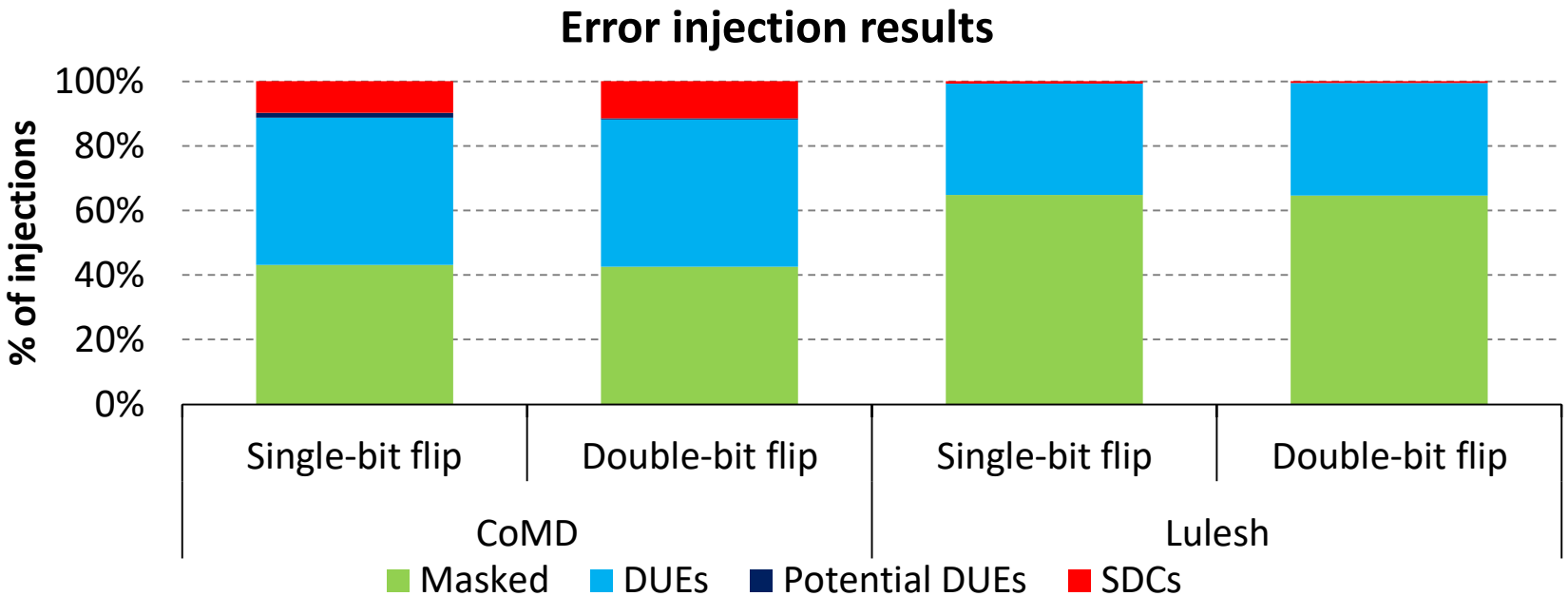
IMPLEMENTING DIFFERENT ERROR MODES



RF mode	IOV mode	IOA mode
At the selected instruction, inject error if the selected register is a source. If not, monitor subsequent instructions and inject when found as a source.	Empty handler	Record register/memory content at the selected instruction count
Empty handler	Inject error at the selected instruction count according to the selected bit-flip model	At the injection instruction • Read values from correct address and write them to the corrupted address • Revert content at the correct address with the recorded values

RESULTS: USE CASE 1

Register File AVF

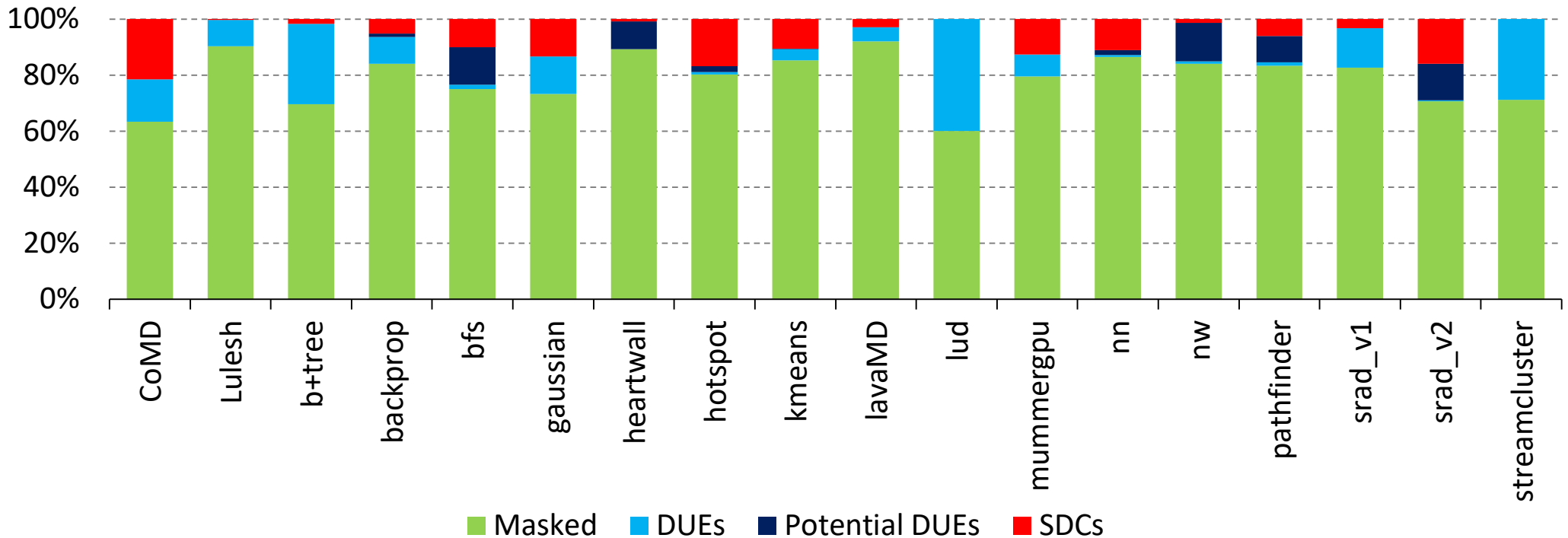


SDC AVF = SDC probability from injections in occupied registers * RF occupancy

0.075 and 0.07 for CoMD and Lulesh, respectively, for single-bit flips

RESULTS: USE CASE 2

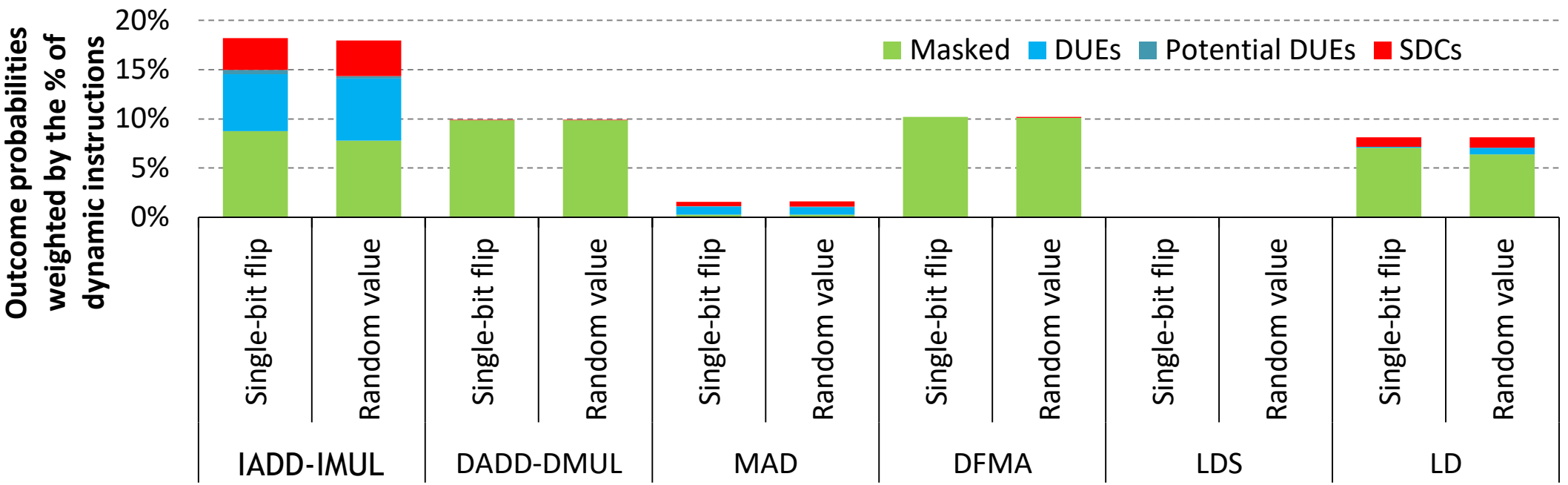
Injecting into a destination register of a randomly selected instruction



Program level manifestations of errors that propagate to instruction outputs are application dependent

RESULTS: USE CASE 3

Identifying instruction types that produce more SDCs



Double instructions are less susceptible than integer instructions for CoMD

Results for the remaining use cases can be found in the paper

SLOWDOWNS

Modest application-level slowdowns

1.02x to 166x slowdowns at the application-level (depends on host vs. GPU runtime)

Kernel-level slowdowns were higher, ranging from 5.2x to 488x

Orders of magnitude faster than lower level simulators

Performance in Million Warp-Instructions Per Second (MWIPS)

		CoMD	Lulesh	Rodinia (Geomean)
SASSIFI	RF Mode	94	85	57
	IOV Mode	81	81	57
	IOA Mode	55	46	64
	GPGPU Sim	<0.1		

CONCLUSIONS

SASSIFI tool for GPU application resilience evaluations

Developed an error injection based tool for GPU application resilience evaluation

- Fast in-silicon error injections
- Flexible to inject many types of errors
- Demonstrated by conducting various types of resilience studies

Released the code for public usage

- GitHub: https://github.com/NVlabs/sassifi