Critical Variable Identifications using Register Vulnerability for Selective Protections

October 19th, 2018

Dukui Song

DEPENDABLE COMPUTING LAB.
DEPT. OF COMPUTER SCIENCE
YONSEI UNIVERSITY

Committee
Kyoungwoo Lee
Bernd Burgstaller
Yosub Han

Agenda

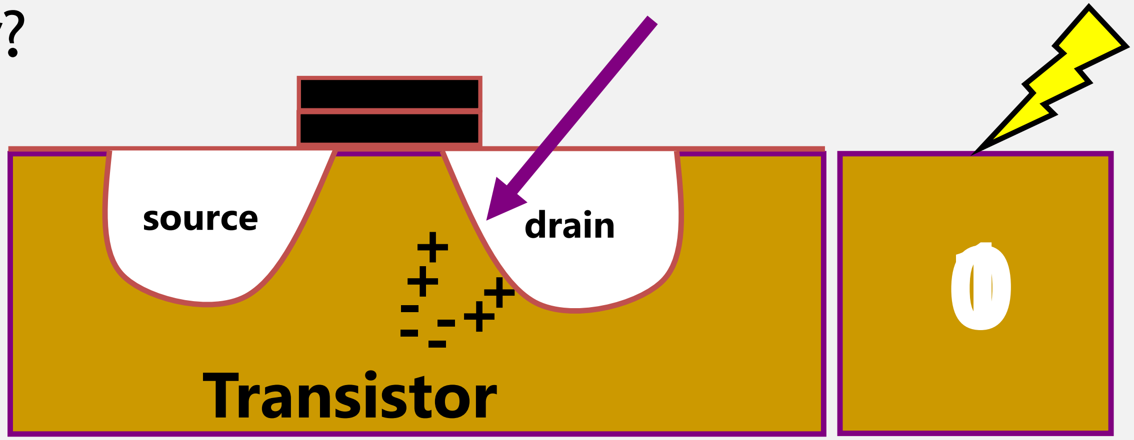
- Motivation
- Related works
- Problem definition
- Method Proposal
- Experiments
- Conclusions

Agenda

- Motivation
- Related works
- Problem definition
- Method Proposal
- Experiments
- Conclusions

What is soft error?

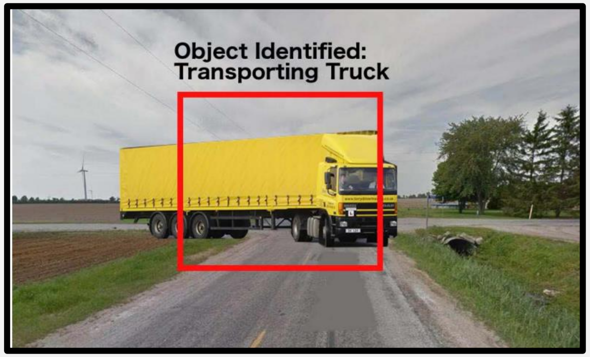
- Soft Error?



  - A phenomenon that the bit of the transistor is temporarily reversed
    - Assume that this transistor contains bit value 0
    - This transistor is attacked by external radiation
    - The external radiation makes some charges
    - The extra charges make the bit value to 1
  - Soft error rate exponentially increase with technology scaling and near-threshold computing

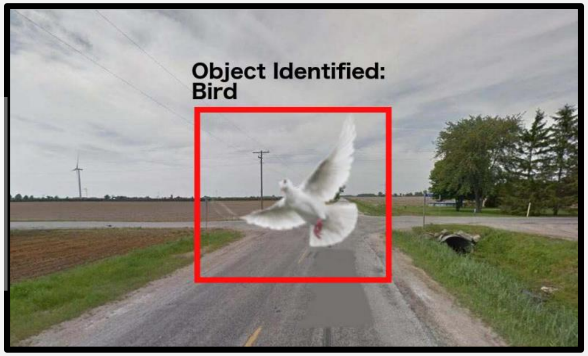
Soft error is important

- Soft error is an increasing concern
  - Soft error is a major threat to system reliability
  - As computer systems are used more and more in industry and life, soft error is becoming important
- If soft error occurs in auto-driving car [2017, Li]
  - Only 1 bit of soft error can lead to misclassification of objects in DNN based vision technique
  - Misclassification can result in the wrong action



action = brake

Soft error makes misclassification (truck → bird)



action = keep driving (collision)

Soft error protection technique is required

- The progress of soft error protection
  - Detect soft error occurrence
  - Execute fault tolerance policy (correction, restart, rollback, etc..)
  - Make the system to operate normally

- Implement soft error protection

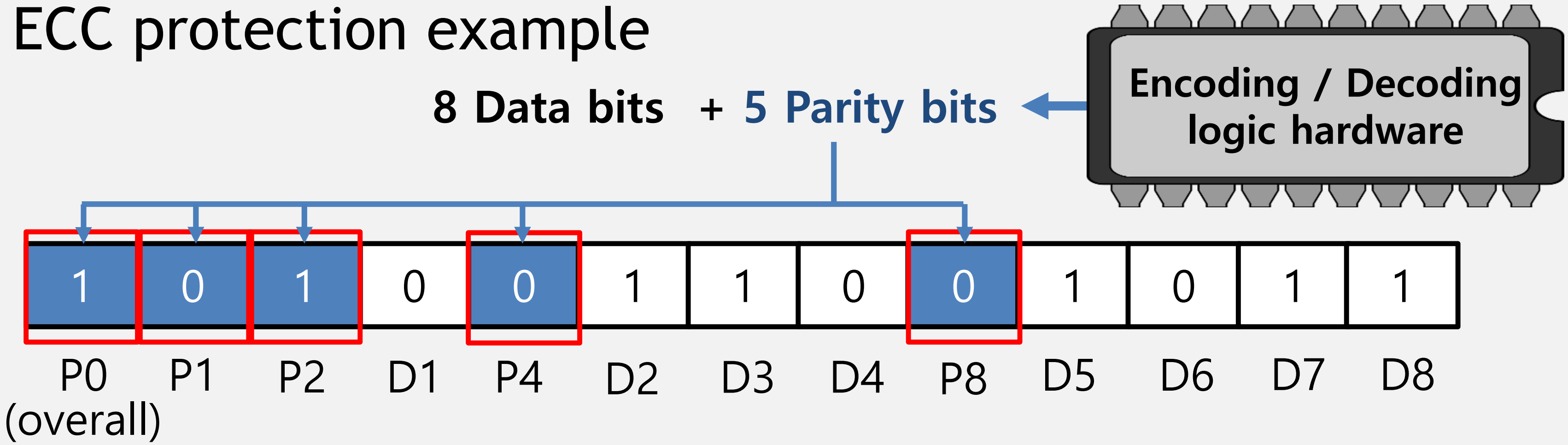
H/W based technique

S/W based technique

Hardware based technique

- Hardware based protection technique
  - Redundant H/W to detect or correct errors
    - Requires additional hardware costs
  - ECC(Error Correction Code) block on L1D (SEC-DED) [2006, chibani]
    - 215% increase runtime than unprotected one
    - 20% additional area occupancy and 300% more power consumption

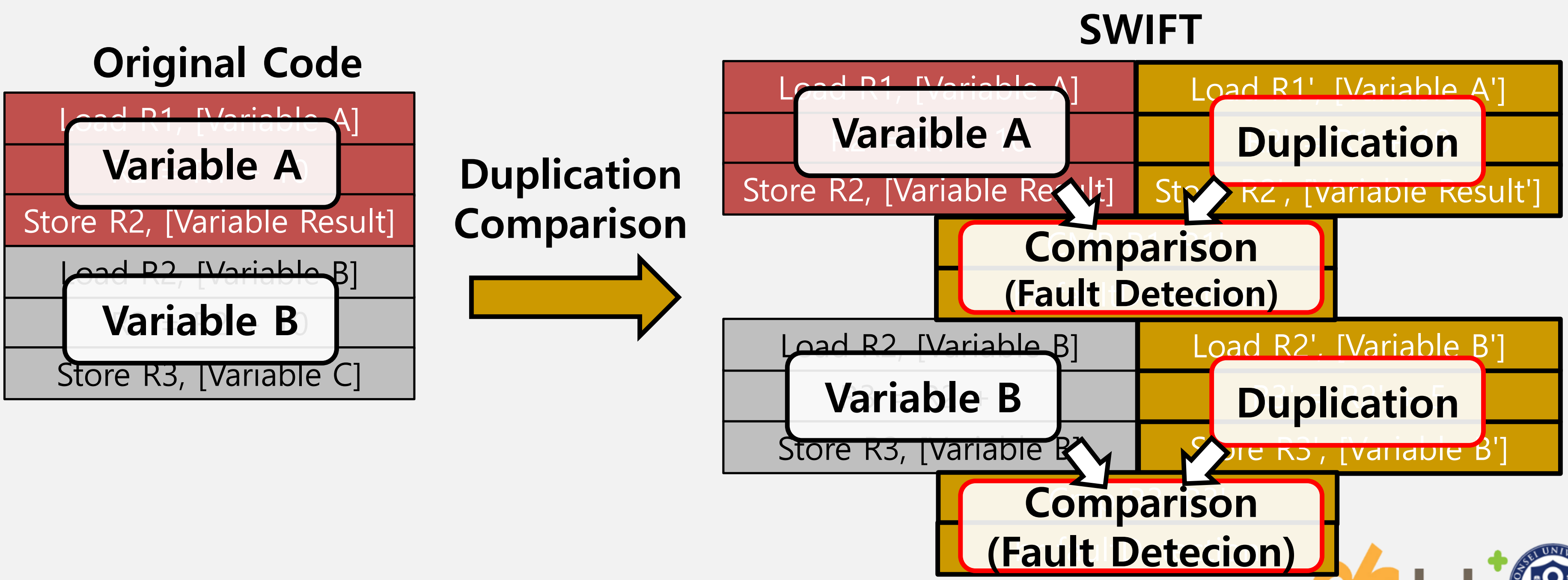
- ECC protection example



1 bit error correction = **P0** will be **wrong, P1,2,4,8** will be **index of error**
2 bit error detection = **P0** will be **correct**

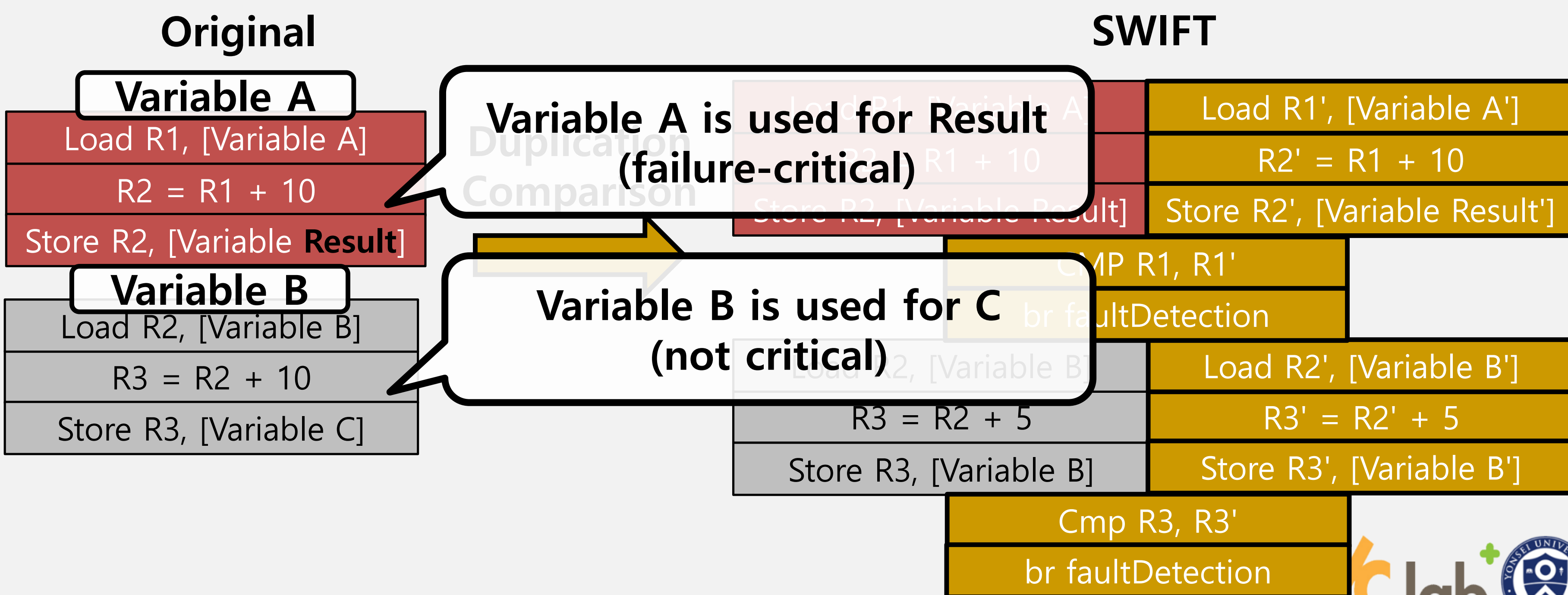
Software based technique

- Software based protection technique
  - No additional hardware costs, flexible to apply and change
  - SWIFT : Insert error detection code on program by duplicating instructions [2005, reis]
    - ◆ 70% of errors detection coverage, 400% increase runtime
- SWIFT (Software Implemented Fault Tolerance) example



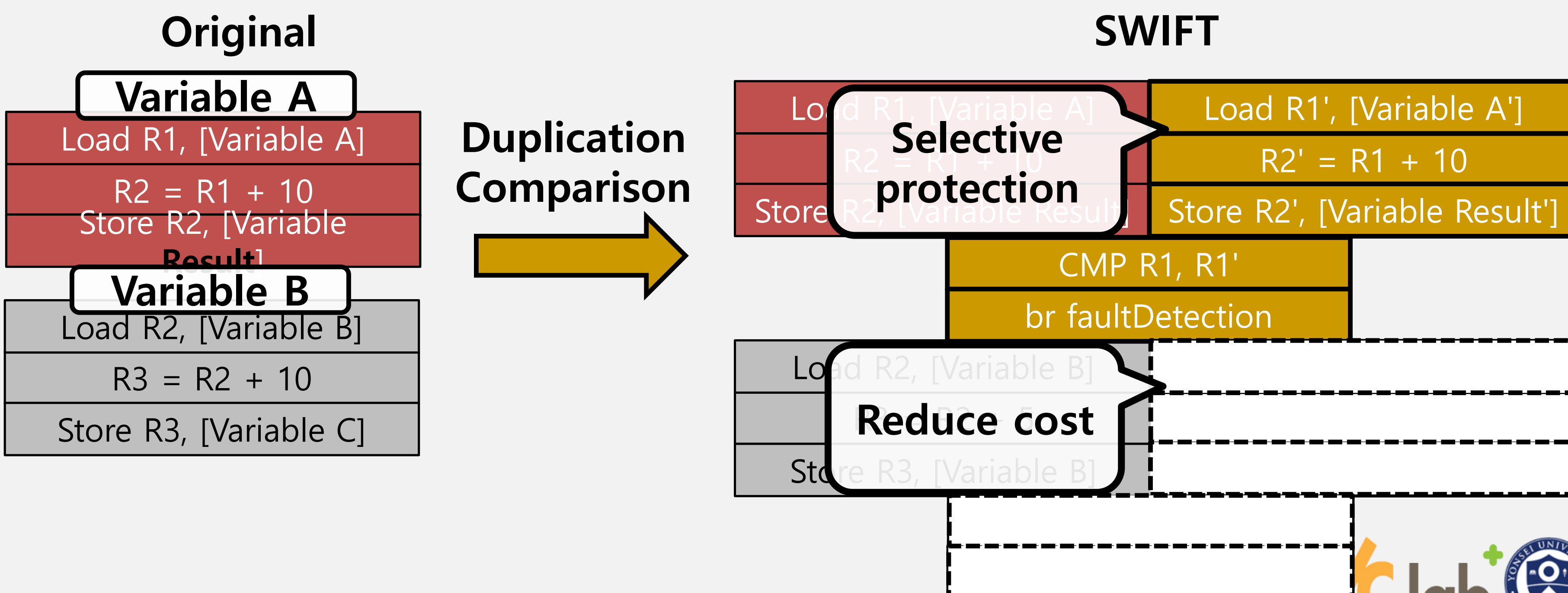
Full protections are highly expensive

- H/W and S/W full protection are highly expensive
  - Runtime overhead : at least 2 times slower
  - May not be suitable for modern computer systems (low-power, IoT)
- Selective protection
  - All variable protection is expensive
  - Only few variables are important, i.e., failure-critical



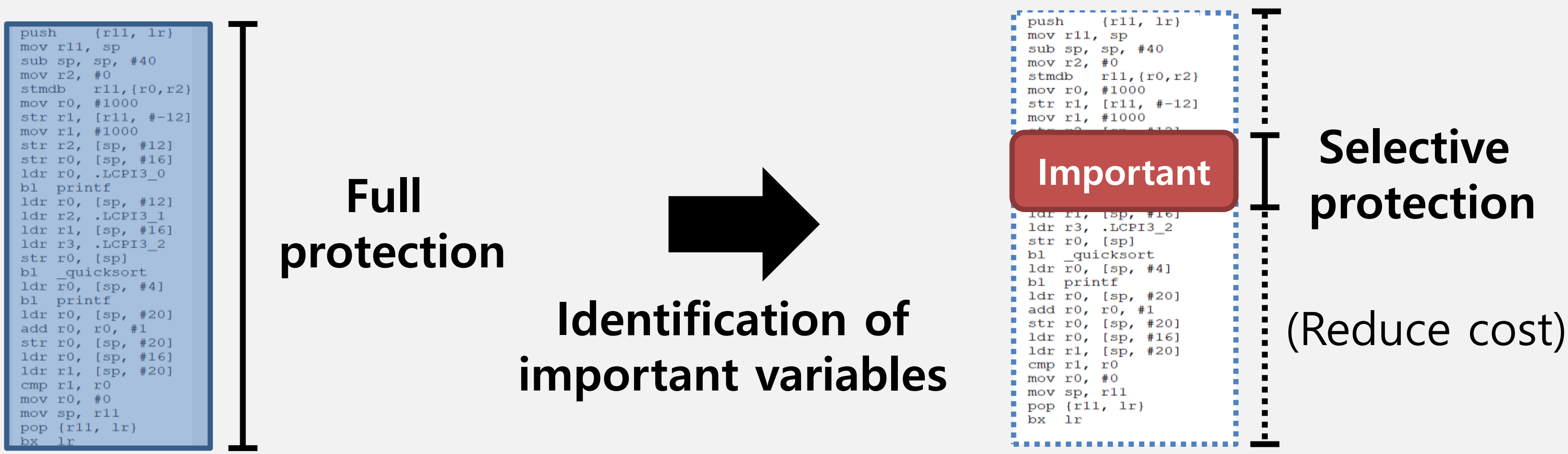
Selective protection : a cost effective way

- H/W and S/W full protection are highly expensive
  - Runtime overhead : at least 2 times slower
  - May not be suitable for modern computer systems (low-power, IoT)
- Selective protection
  - All variable protection is expensive
  - Only few variables are important, i.e., failure-critical



Selective protection : a cost effective

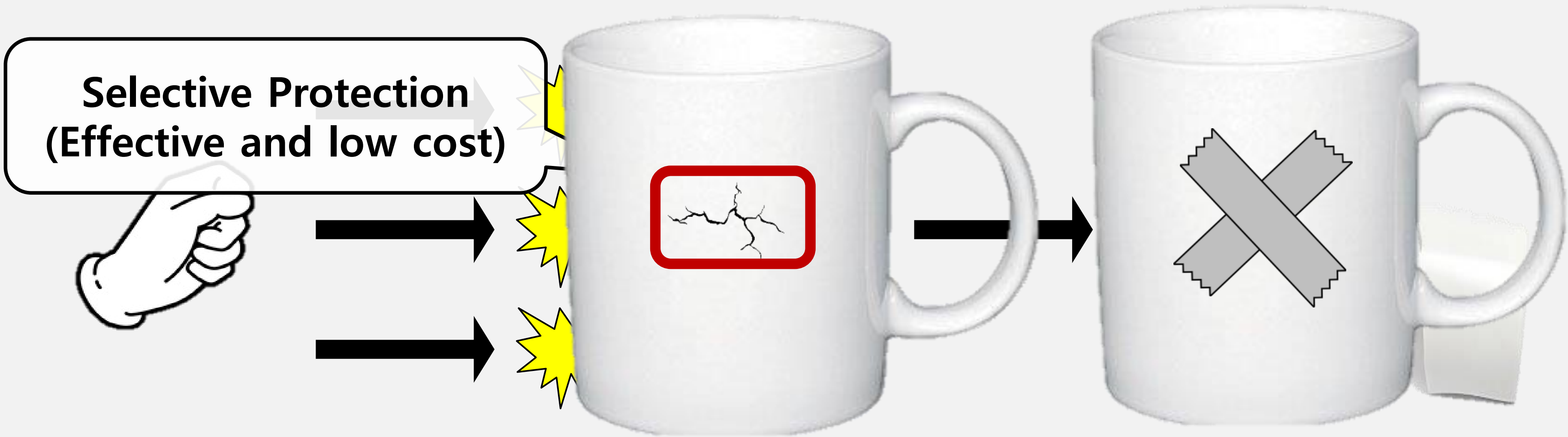
- H/W and S/W technique are highly expensive
  - Runtime overhead : at least 2 times slower
  - May not be suitable for modern computer systems (low-power, IoT)
- Selective protection
  - All variable protection is expensive
  - Just some of them are important, i.e., failure-critical



Selective protections on important variables can be cost effective

Selective protection : a cost effective

- Selective protection example
  - Mug cup with an invisible crack
  - The cup will be broken if knock the cracked part
  - Attaching the tapes to the only crack can prevent broken



We need to figure out where to be protected

Agenda

- Motivation
- Related works
  - Method for finding where to protect in H/W
  - Method for finding where to protect in S/W (Critical variables)
- Problem definition
- Method Proposal
- Experiments
- Conclusions

H/W Vulnerability Measurement

- AVF(Architectural Vulnerability Factor) [2003, Mukherjee]
  - Vulnerability : possibility that a fault in that particular structure will result in an error

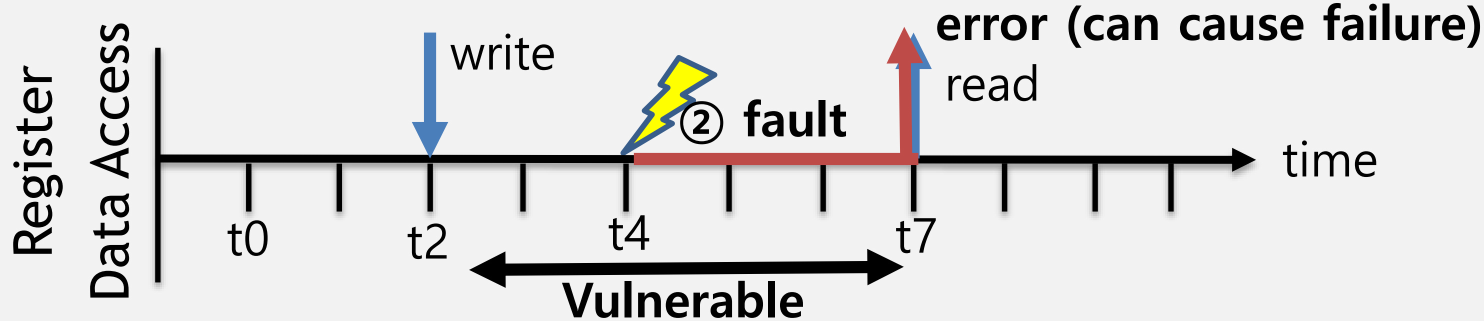
Fault is overwritten, not vulnerable

  - Before reading after writing (t2 ~ t7) is vulnerable
  - Weakness : accuracy (instruction unit), scalability (limited H/W)

- gemV toolset [2016, Tanikella]
  - Improve accuracy and scalability
  - Accuracy : CPU-cycle unit measurement (gem5 based)
  - Scalability : supports various hardware and components
  - Validation of vulnerability measurement by fault injection

H/W Vulnerability Measurement

- AVF(Architectural Vulnerability Factor) [2003, Mukherjee]
  - Vulnerability : possibility that a fault in that particular structure will result in an error

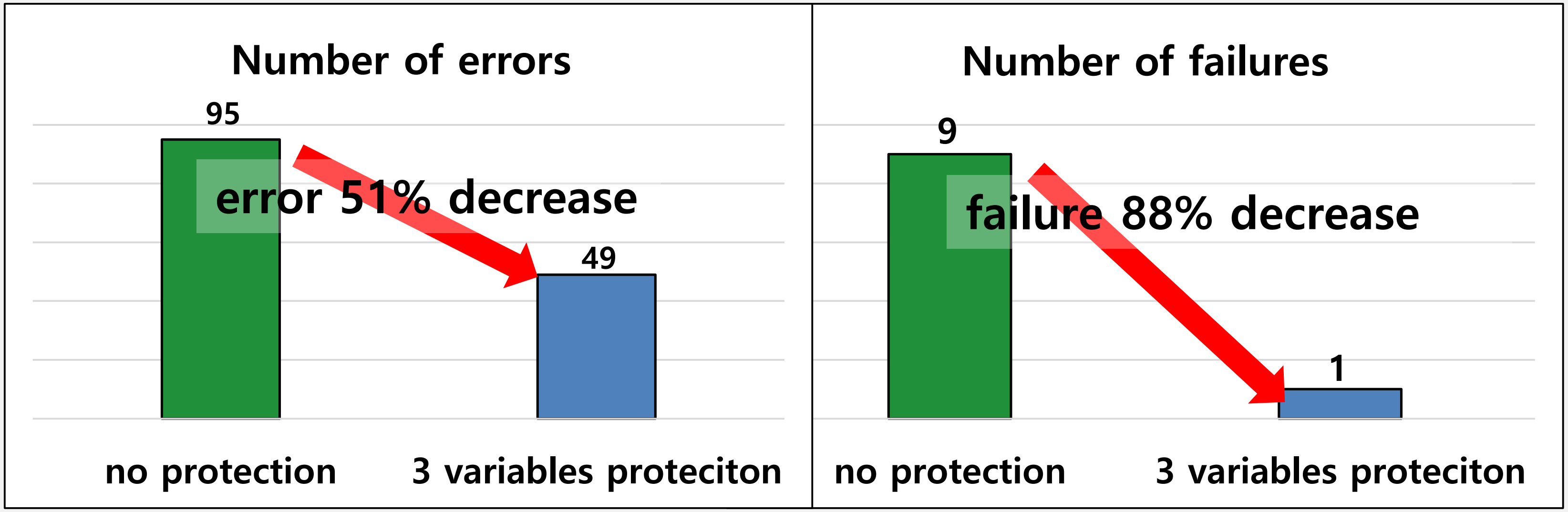


  - Before reading after writing (t2 ~ t7) is vulnerable
  - Weakness : accuracy (instruction unit), scalability (limited H/W)

- gemV toolset [2016, Tanikella]
  - Improve accuracy and scalability
  - Accuracy : CPU-cycle unit measurement (gem5 based)
  - Scalability : supports various hardware and components
  - Validation of vulnerability measurement by fault injection

S/W Vulnerability Measurement

- Identification of Critical Variables using an FPGA-based Fault Injection Framework [Riefert, 2013]
  - **The Critical variable** : a variable that significantly affect on program **execution and calculation results** (frequently used)
  - In fault injection, critical variable will be highly injected
  - Experiment : protecting 3 variables (Runtime 18% increase)



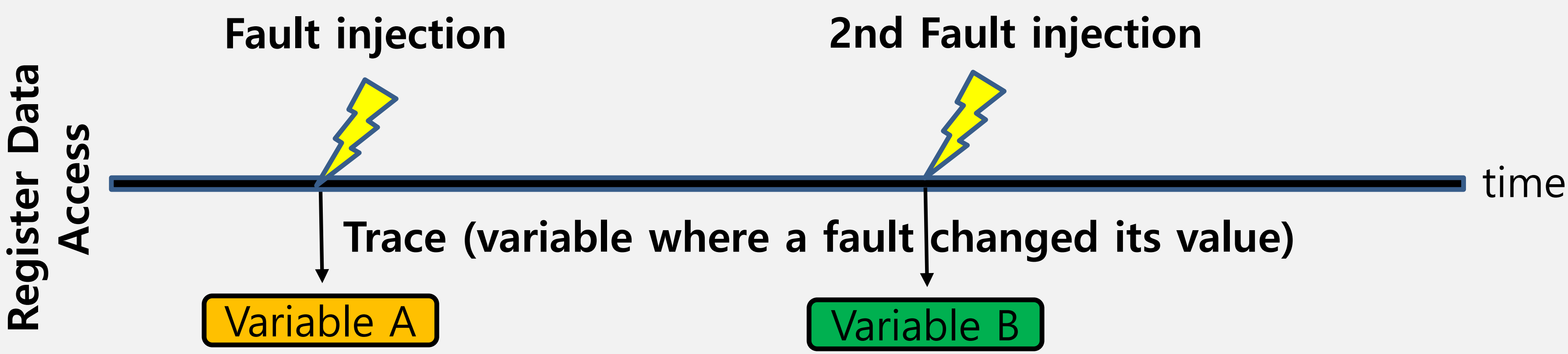
  - However, Fault injection it takes large of time to run fault injection campaigns (at least 7,000 program run)

Agenda

- Motivation
- Related works
- Problem definition
  - Finding critical variable(fault injection) takes a lot of time
- Method Proposal
- Experiments
- Conclusions

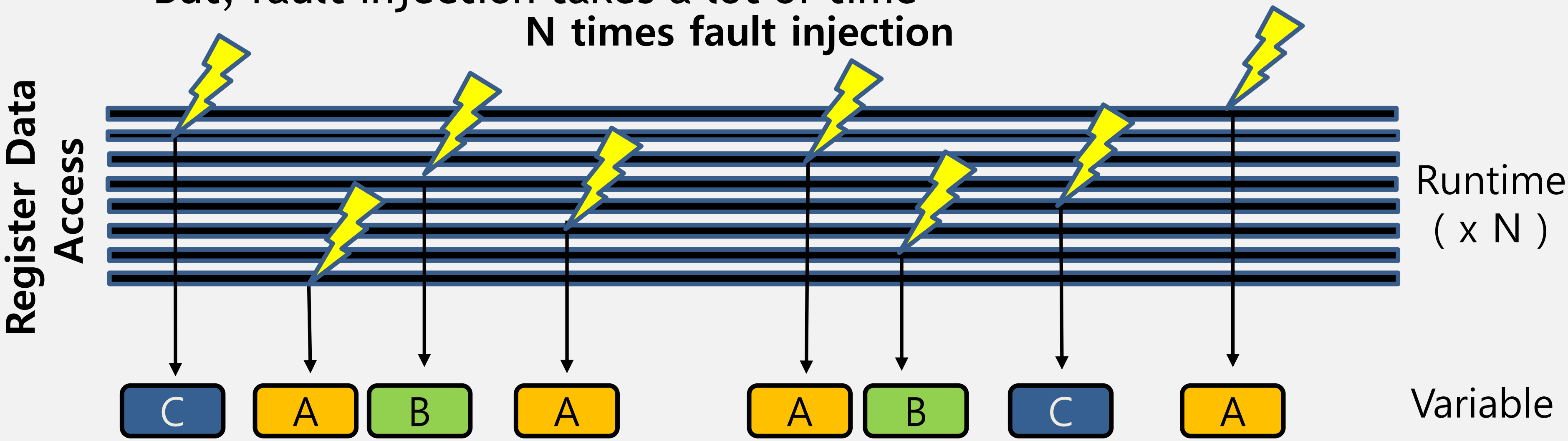
Fault Injection takes a lot of time

- Fault Injection method for finding critical variables
  - Fault injection is a good technique for finding critical variables
  - A large number of faults will be injected into the critical variables
  - Protecting critical variable, effective protection method at low cost
  - But, fault injection takes a lot of time



Fault Injection takes a lot of time

- Fault Injection method for finding critical variables
  - Fault injection is a good technique for finding critical variables
  - A large number of faults will be injected into the critical variables
  - Protecting critical variable, effective protection method at low cost
  - But, fault injection takes a lot of time



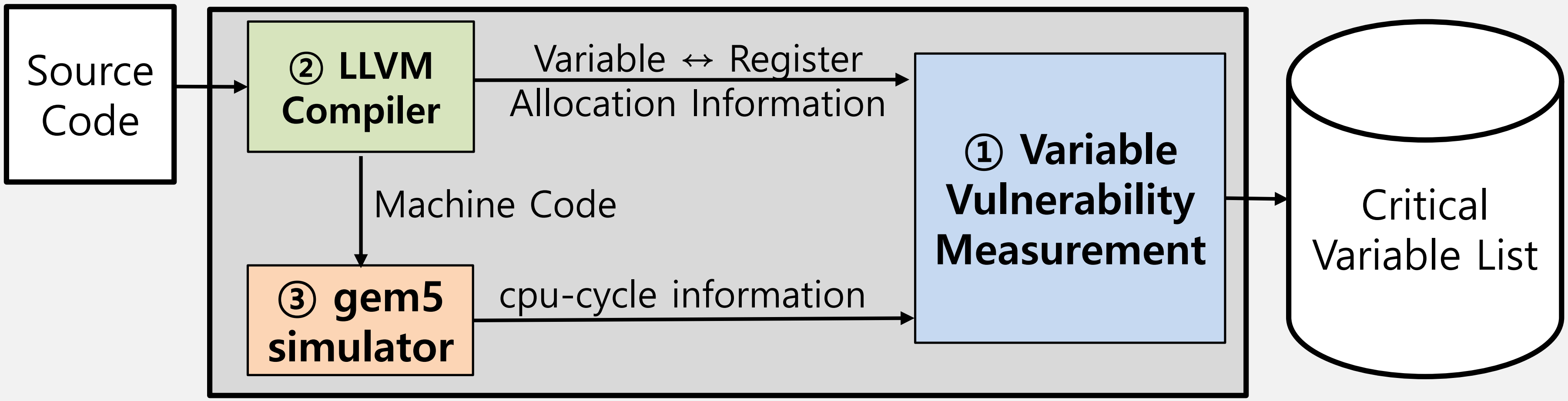
Count of faults = Variable A : 4 Variable B : 3 Variable C : 2

Agenda

- Motivation
- Related works
- Problem definition
- Method Proposal
  - Since fault injection take lots of time, we develop an alternative way to find out the critical variable
  - The framework of variable vulnerability measurement with LLVM compiler and gem5 simulator
- Experiments
- Conclusions

Overview of our proposed technique

- Variable vulnerability method for finding critical variables

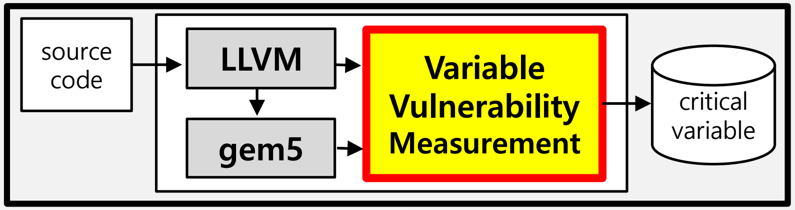


① Measures the vulnerability of variables that can cause an error

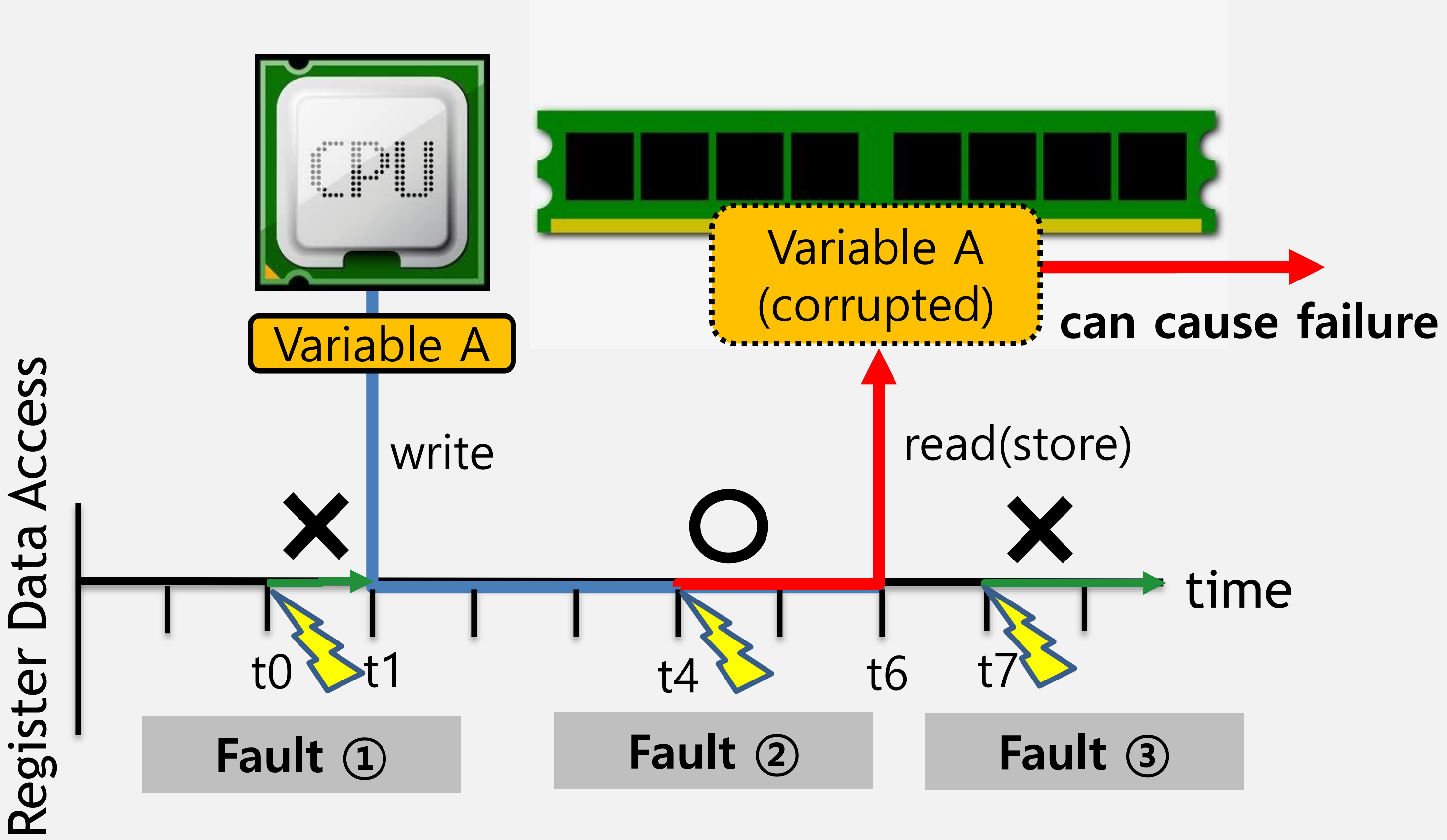
② LLVM maps variables and register

③ gem5 calculates actual CPU-cycle for vulnerability

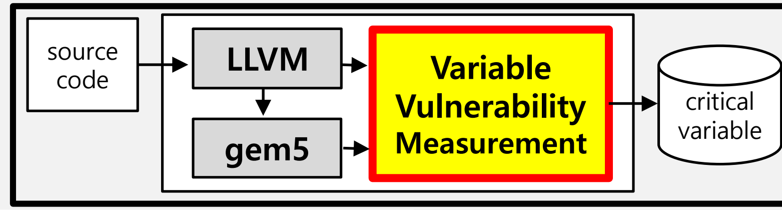
① Varaible Vulnerability Measurement



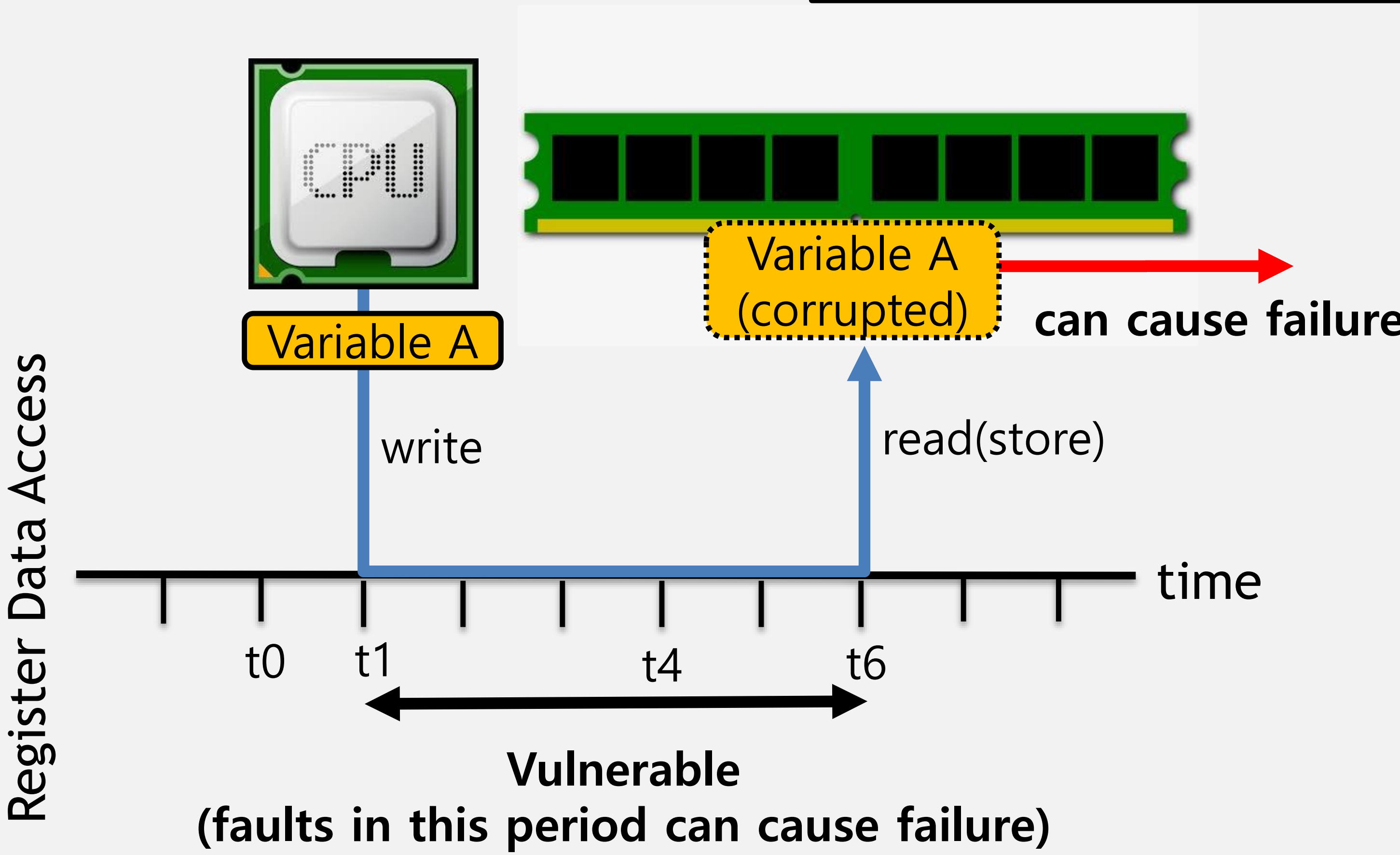
- Fault and Vulnerability



① Varaible Vulnerability Measurement



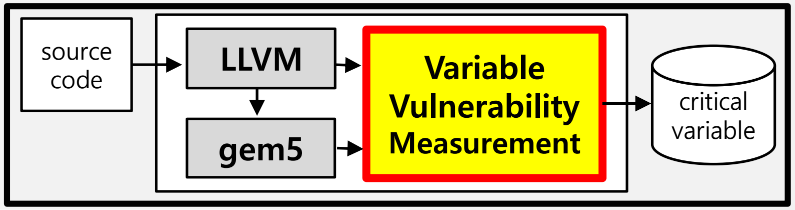
- Vulnerable Period

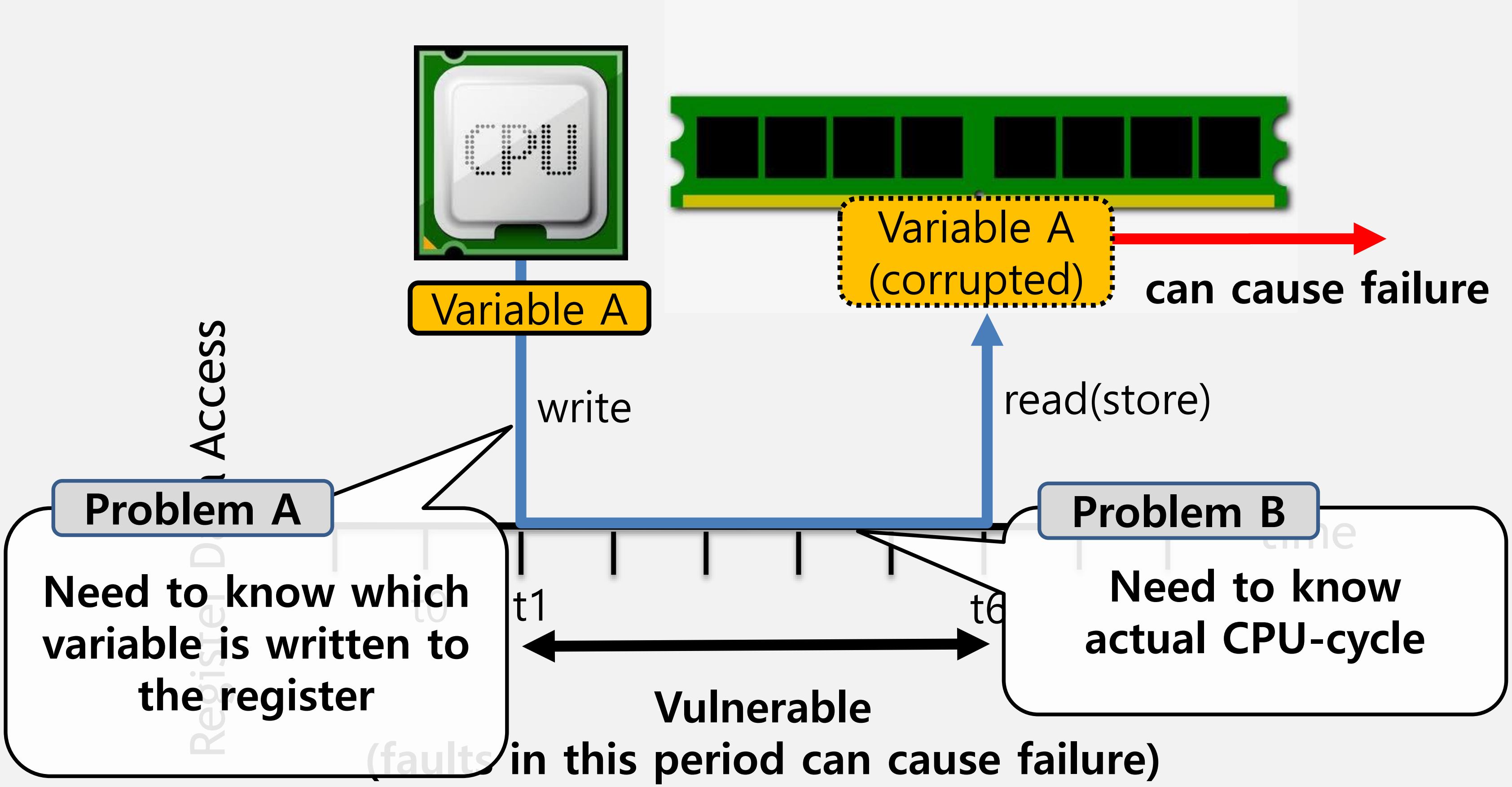


Variable A' Vulnerability = t6 - t1 = 5 time units

① Varaible Vulnerability Measurement

- Requirement for measurement

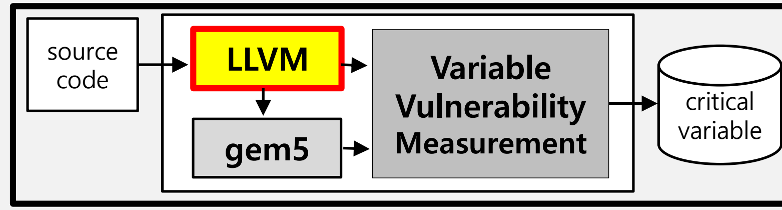




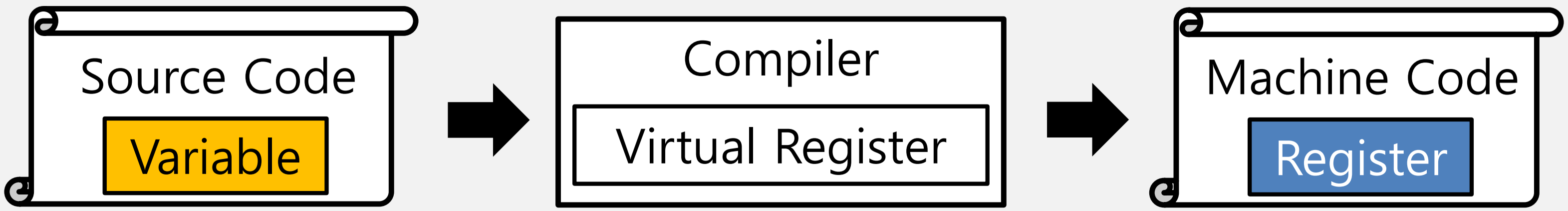
Variable A' Vulnerability = t6 - t1 = 5 time units

② Mapping Register ↔ Variables

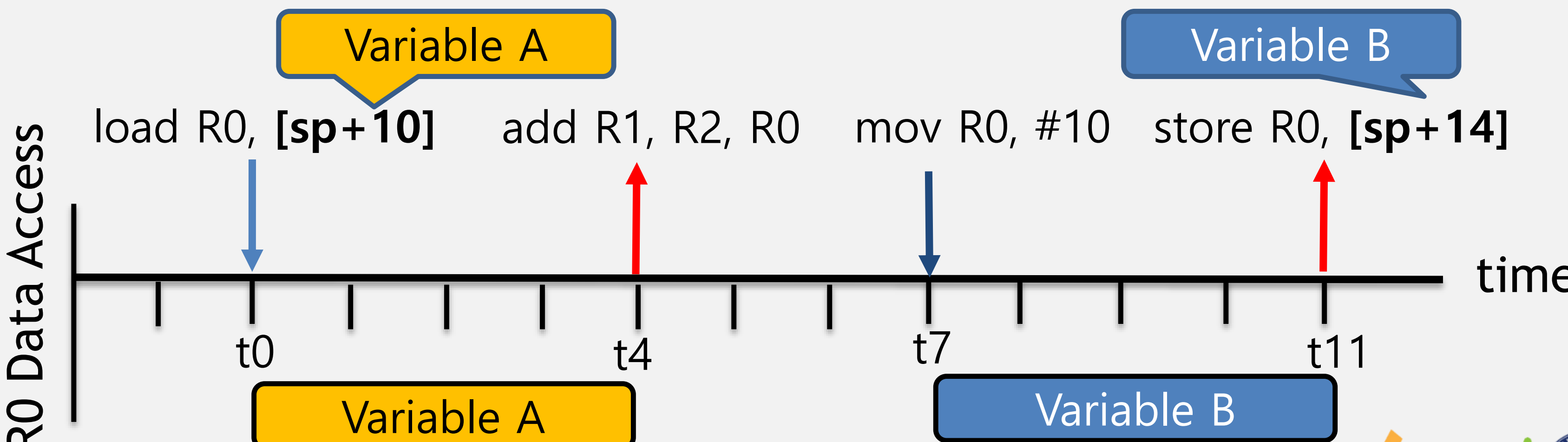
☞ **Answer to problem A**
(Which variable is written to the register)



- Modified LLVM Compiler
  - Variables are assigned to registers during compile

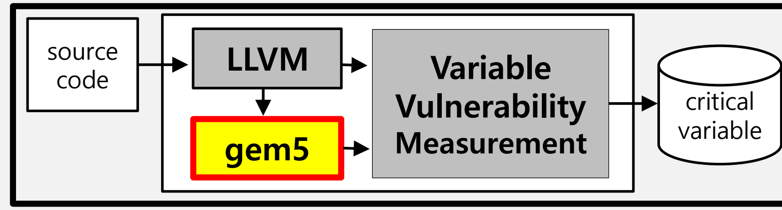


  - Modify compiler to output variable↔register allocation information
  - Machine code with variable name

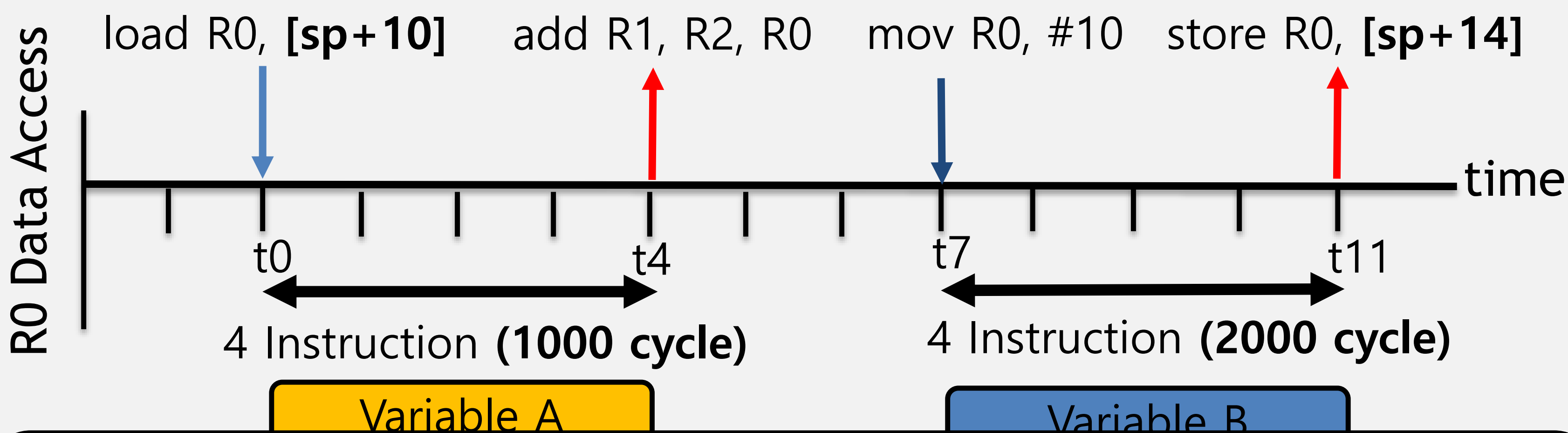


③ Actual vulnerable period cpu-cycle

☞ **Answer to problem B**
(need to know actual CPU-cycle)



- gem5 calculate CPU-cycle
  - Although the number of instructions is the same, the actual vulnerable time may be different



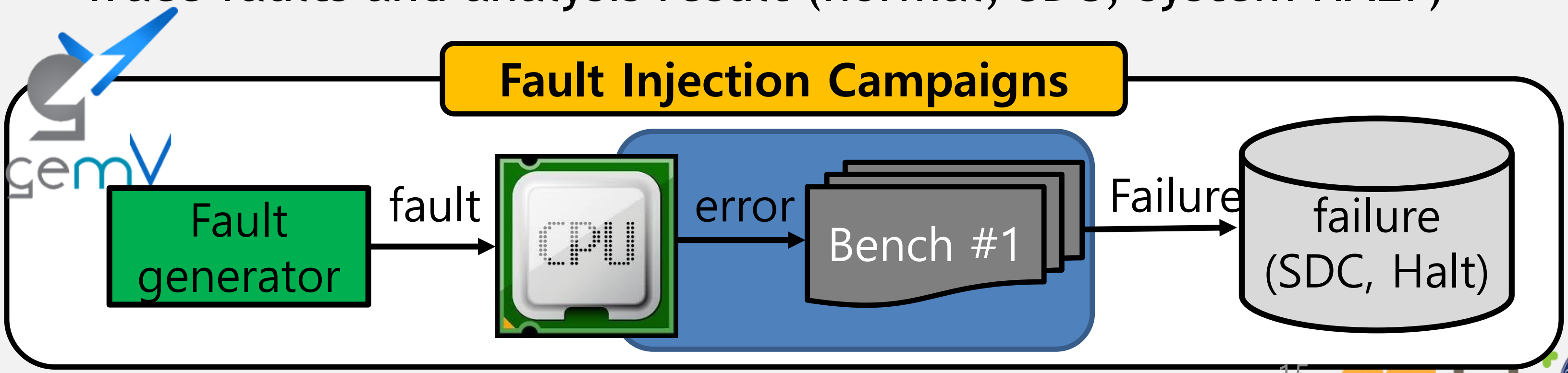
With our framework(vulnerability measurement with LLVM and gem5),
Now we can calculate variable vulnerability clearly

Agenda

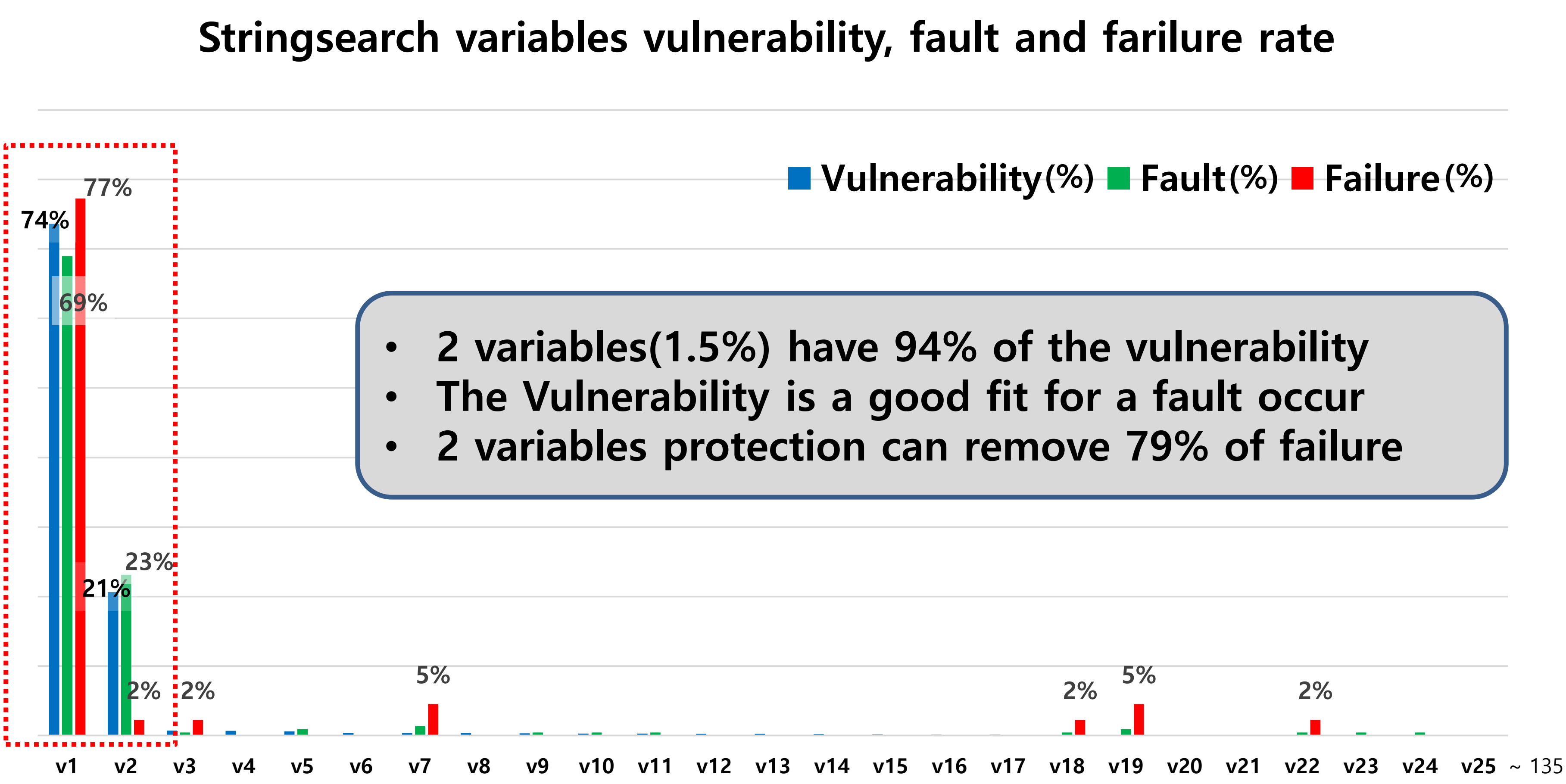
- Motivation
- Related works
- Problem definition
- Method Proposal
- Experiments
  - Validate our vulnerability based framework with fault injection campaigns
- Conclusions

Experiments

- Validation
  - Are variables with high vulnerability be more fault injected?
  - Can assume variable with a high vulnerability is a critical variable?
- Fault injection experiment setup
  - Benchmark : 6 programs (MiBench version 1.0)
  - 4,000 times fault injection for each benchmark
  - gemV toolset is used for fault injection experiment
  - Trace faults and analysis result (normal, SDC, system HALT)

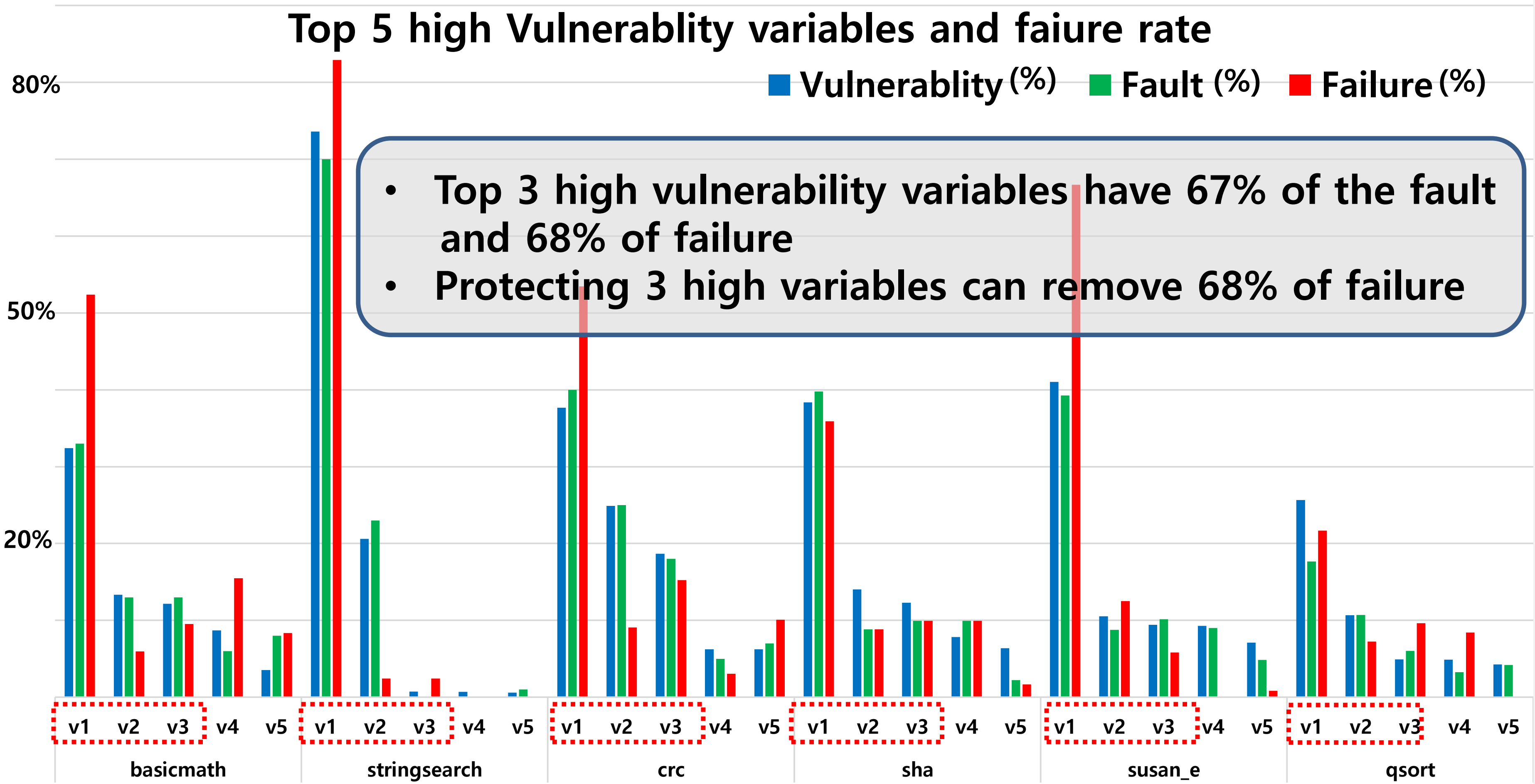


Result : Vulnerablity method fit a fault occur



* Variables are sorted in descending order of vulnerability.

Result : Find critical variable with vulnerablity



- Top 3 high vulnerability variables have 67% of the fault and 68% of failure
- Protecting 3 high variables can remove 68% of failure

* Variables are sorted in descending order of vulnerability.

Agenda

- Motivation
- Related works
- Problem definition
- Method Proposal
- Experiments
- Conclusions

Conclusion

- Conclusion
  - Soft error is an important concern
  - H/W and S/W full protection is expensive
  - Need to identify the critical variables for selective protections
    - The higher the variable vulnerability, more faults are injected
  - We propose a framework for critical variable identifications with vulnerability measurement
    - Modeling vulnerability of variable
    - Early estimation of critical variables (without fault injection)
  - In the experiment, only protecting top 3 vulnerable variables, 68% failures can be removed (3 variable are critical variable)
  - Provide protection priority for selective technique

- Future work
  - There is a difference between variable vulnerability and failure (Masking effect : not all faults cause failure)
  - Research to minimize the difference caused by masking effect