Towards Principled Error-Efficient Systems

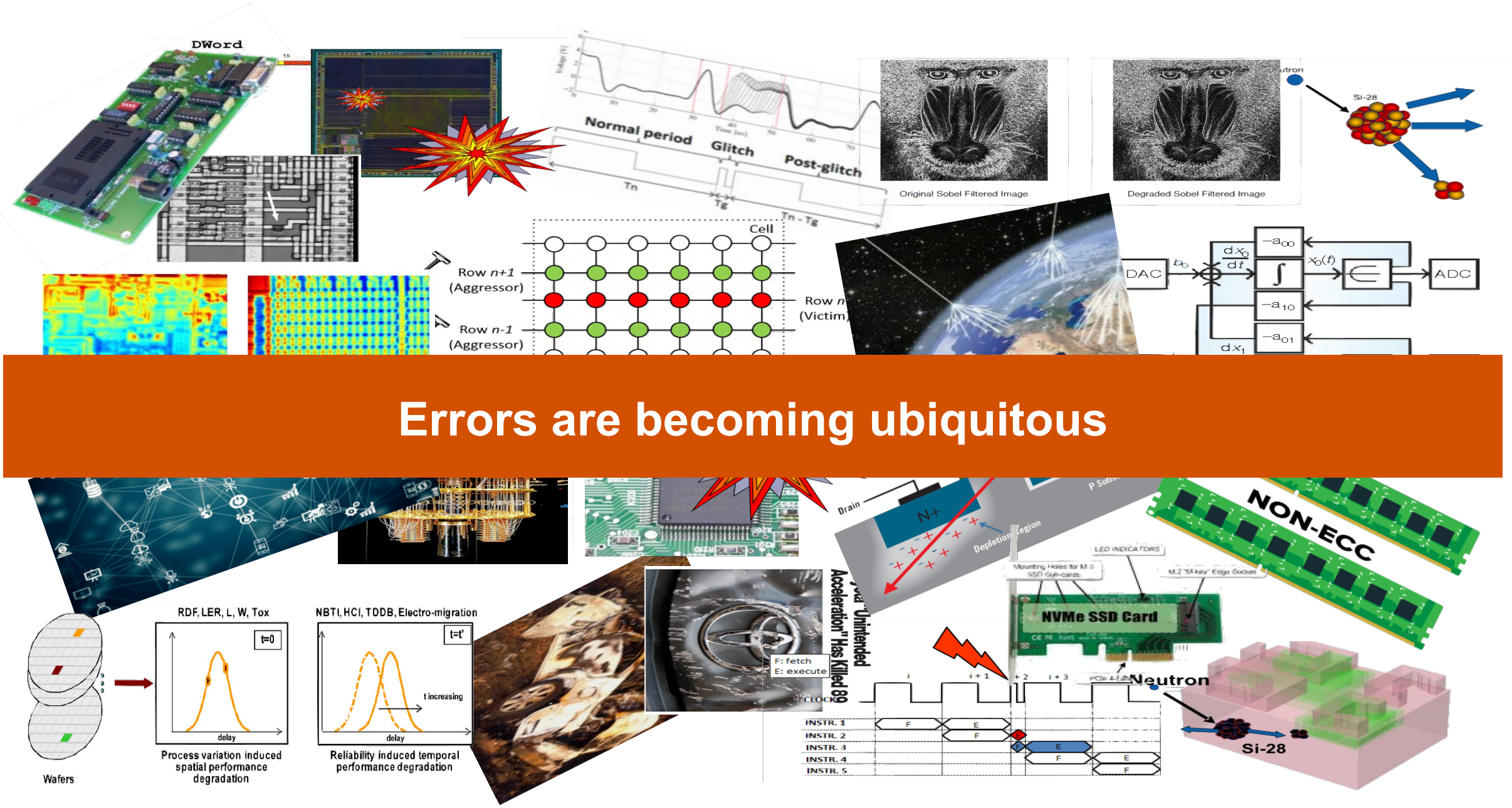
Sarita Adve
University of Illinois at Urbana-Champaign
sadve@illinois.edu

IOLTS 2020 Keynote

Collaborators:

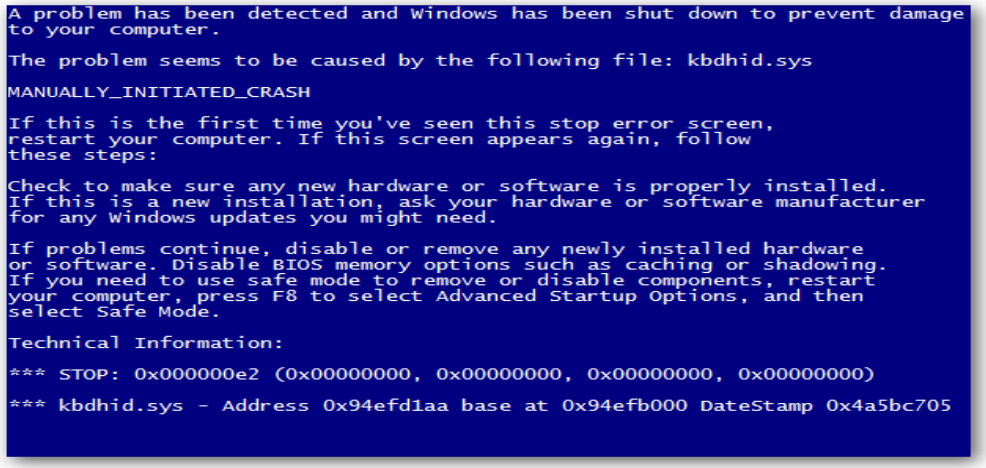
Abdulrahman Mahmoud, Radha Venkatagiri, Vikram Adve, Khalique Ahmed, Christopher Fletcher, Siva Hari, Maria Kotsifakou, Darko Marinov, Sasa Misailovic, Hashim Sharif, Yifan Zhao, and others

This work is supported in part by DARPA, NSF, a Google Faculty Research Award, and by the Applications Driving Architecture (ADA) Research center (JUMP center co-sponsored by SRC and DARPA)



Pictures taken from publicly available academic papers and keynote presentations

Errors  (Must Prevent)



News > Science

Cosmic particles can change elections and cause planes to fall through the sky, scientists warn

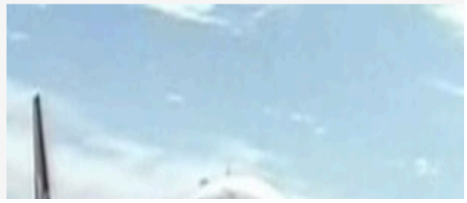
Tiny invisible particles can cause bits of information held by computers to 'flip' with potentially serious ramifications

Ian Johnston Science Correspondent in Boston | @montaukian |
Friday 17 February 2017 15:45 |

Computer error behind Qantas midair drama

Updated 14 Oct 2008, 5:59am

Authorities have blamed a faulty onboard computer system for last week's mid-flight incident on a Qantas flight to Perth.



EE Times

HOME NEWS PERSPECTIVES DESIGNLINES VIDEOS RADIO EDUCATION

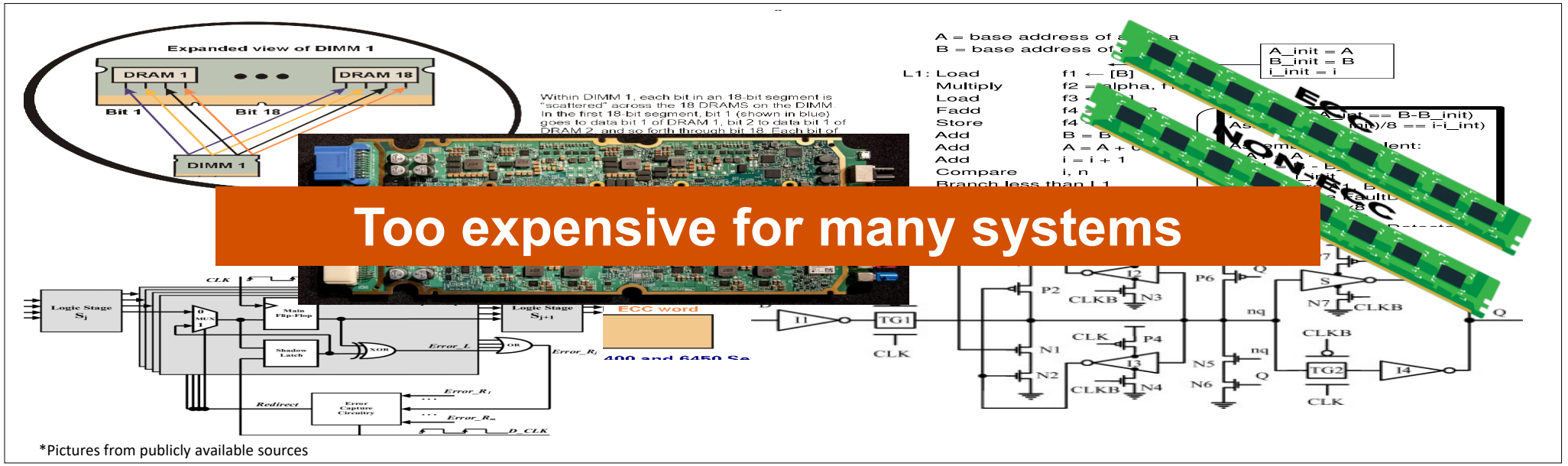
DESIGNLINES | AUTOMOTIVE DESIGNLINE

Toyota Case: Single Bit Flip That Killed

By Junko Yoshida, 10.25.13 108

Errors (Must Prevent)

Designs that aim to prevent all errors



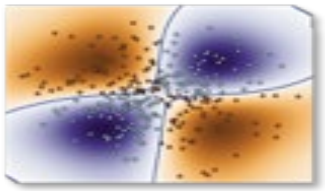
*Pictures from publicly available sources

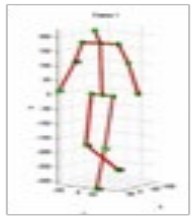
Key Facilitator : ~~Moore's Law + Dennard~~ Scaling

Power, Performance, Area are now limiting factors

Applications Provide Opportunities

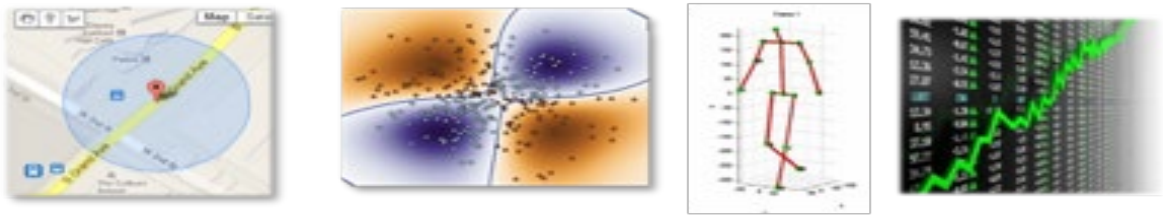
Algorithm / Application





Applications Provide Opportunities

Algorithm / Application



Execution

© MIT News

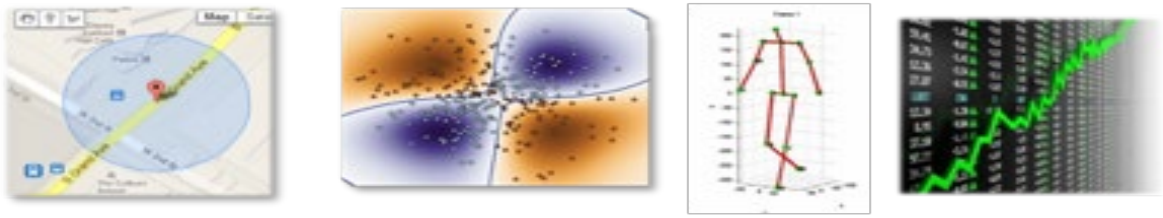
Perfect

Output

Perfect

Applications Provide Opportunities

Algorithm / Application



Execution

© MIT News

~~Perfect~~
Can tolerate errors

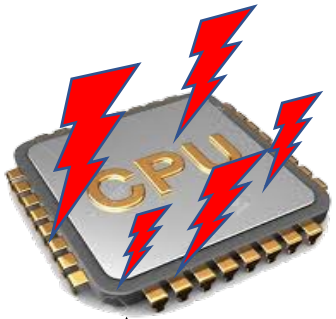
Output

~~Perfect~~
Sufficient Quality

Errors (Tolerable)

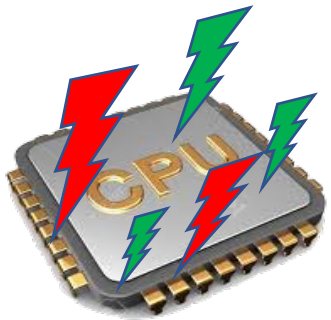
Low-Cost Resiliency

Expensive



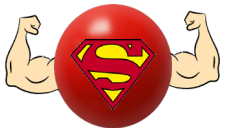
Prevent all

Cheap(er)



Tolerate some

User Acceptable Output

Errors  (Desirable)

Approximate Computing

Expensive

Cheap(er)

User Acceptable Output

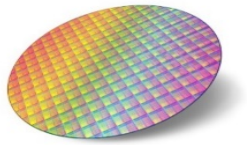
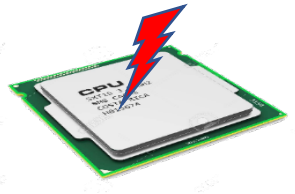
Error-Efficient Systems



Error-Efficient

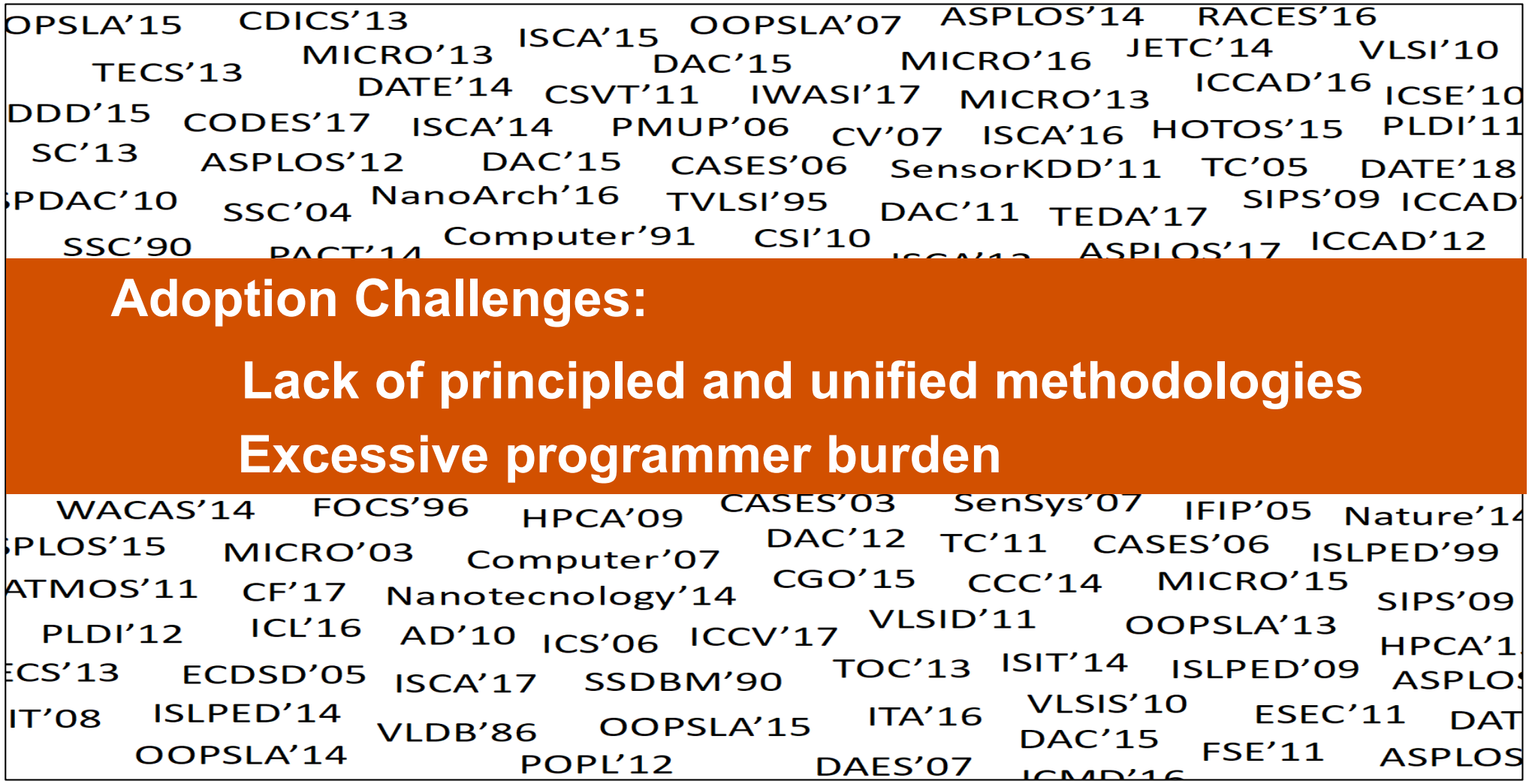
Only prevent as many (HW or SW) errors as absolutely needed (allow others)




Conserve resources across the **system** stack

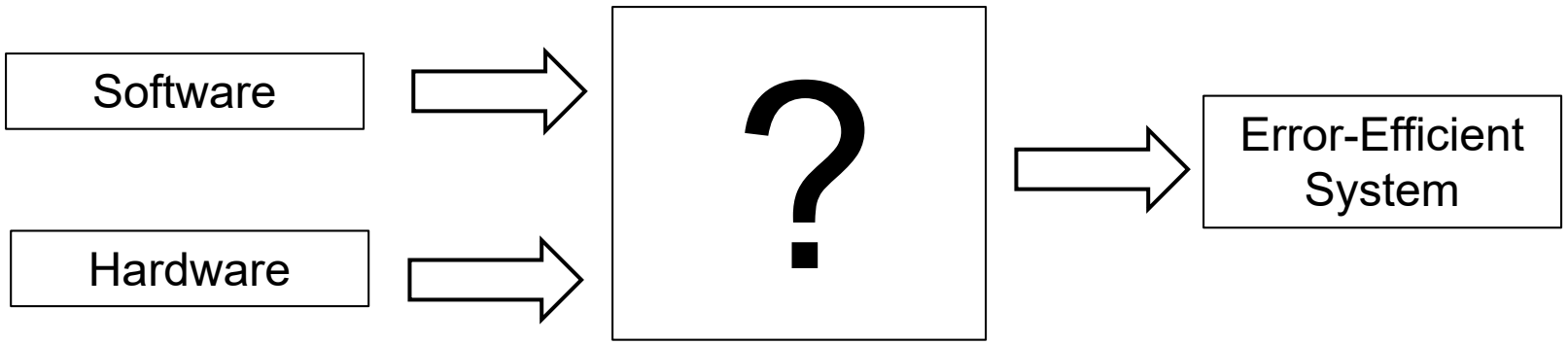
Error-Efficient Systems



Research Vision

1. Enable error-efficiency as a first-class metric for novice and expert users

2. Principled and unified error-efficiency workflow across the system stack



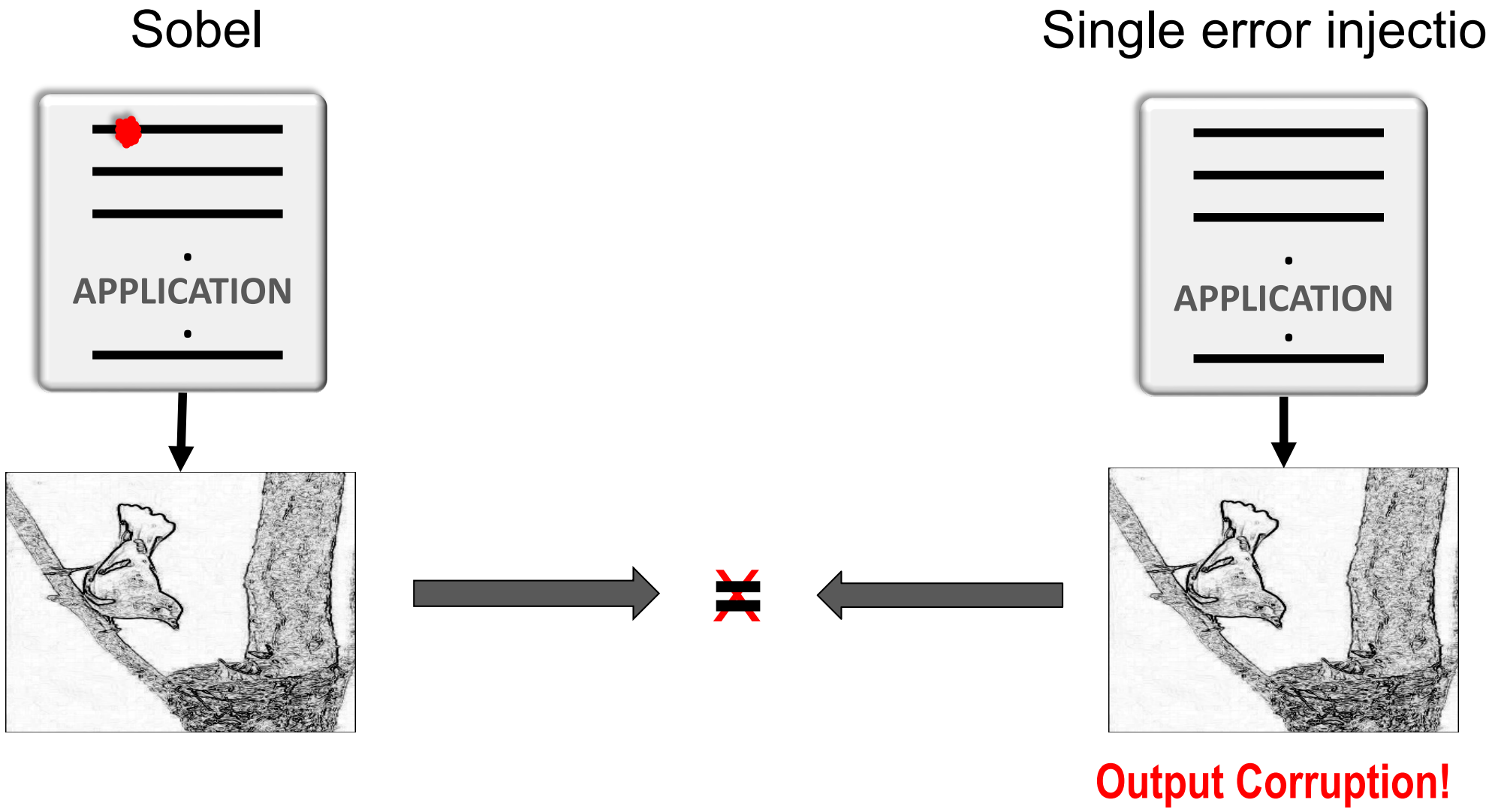
Outline

- Software-centric error analysis and error efficiency: Approxilyzer, Winnow
- Software testing for hardware errors: Minotaur
- Domain-specific error efficiency: HarDNN
- Compiler and runtime for hardware and software error efficiency: ApproxTuner
- Putting it Together: Towards a Discipline for Error-Efficient Systems

Objective of Error Analysis

How do (hardware) errors affect program output?

Error Outcome of Single Error



Quantifying Output Quality

Sobel

Single error injection

Quality Metric
(domain specific)

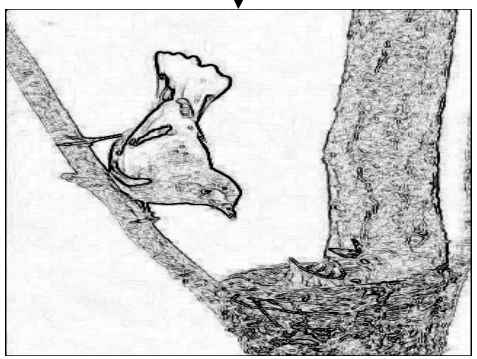
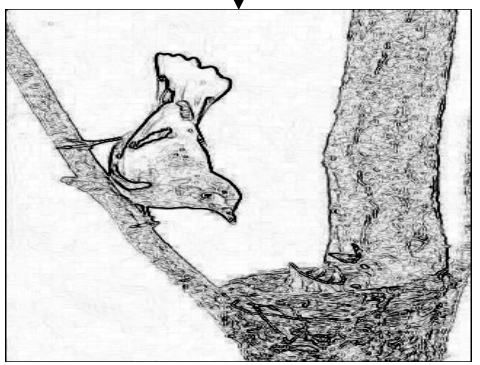


Image difference
(rmse)

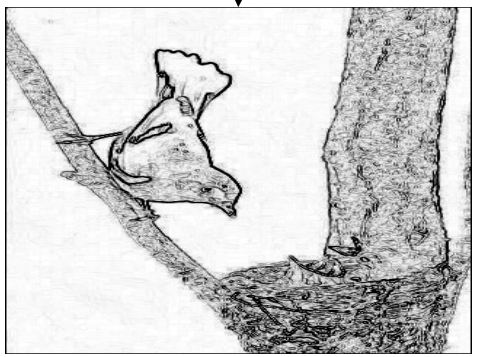
7%
Quality degradation



Output Corruption!

Is Output Quality Acceptable?

Sobel



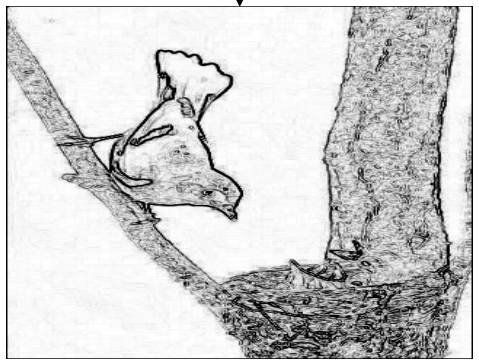
Quality Metric
(domain specific)

Quality Threshold = 10%

Image difference
(rmse)

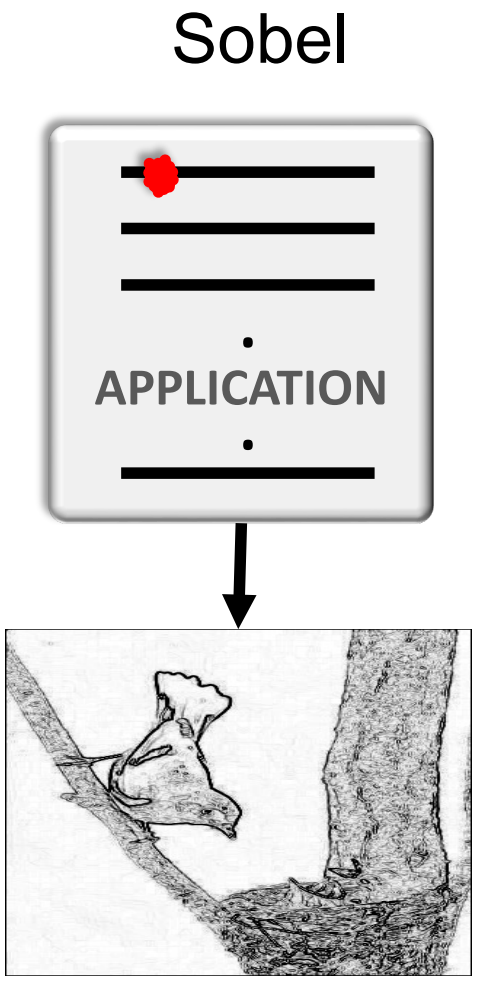
7%
Quality degradation

Single error injection



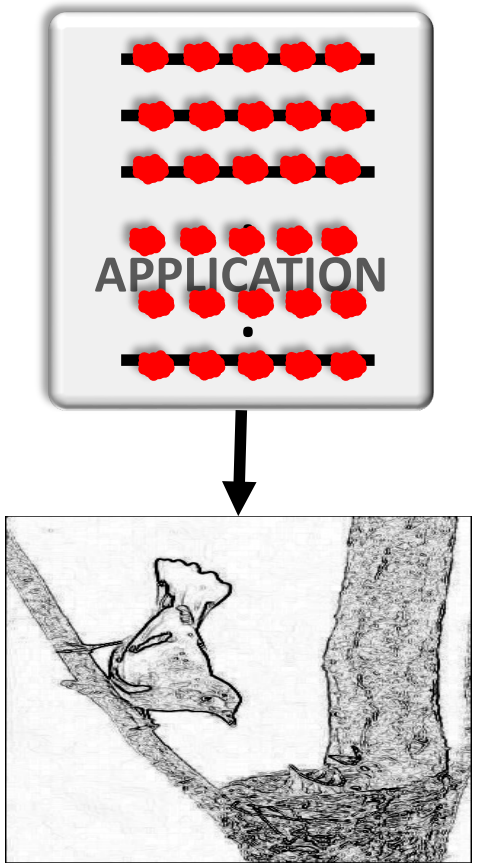
User-Acceptable Output Corruption!

Error Outcome of Single Error



Error Outcome of All Errors

Sobel



Challenges of Automated Error Analysis

- **Accurate** : Precisely calculate output quality
- **Comprehensive** : All errors (for given error model)
- **Automatic** : Minimal programmer burden
- **Cheap** : Many error injections = expensive!
- **General Methodology** : Applications + Error Models

Meeting ALL of the above requirements is hard

Tools Suite for Automated Error Analysis

- **Accurate** : Precisely calculate output quality ✓
- **Comprehensive** : All errors (for given error model) ✓
- **Automatic** : Minimal Programmer Burden ✓
- **Cheap** : Many error injections = expensive! ✓
- **General Methodology** : Applications + Error Models ✓

Tool Suite : **Relyzer, Approxilyzer, gem5-Approxilyzer, Winnow**

ISCA'14, MICRO'16, DSN'19, In Review

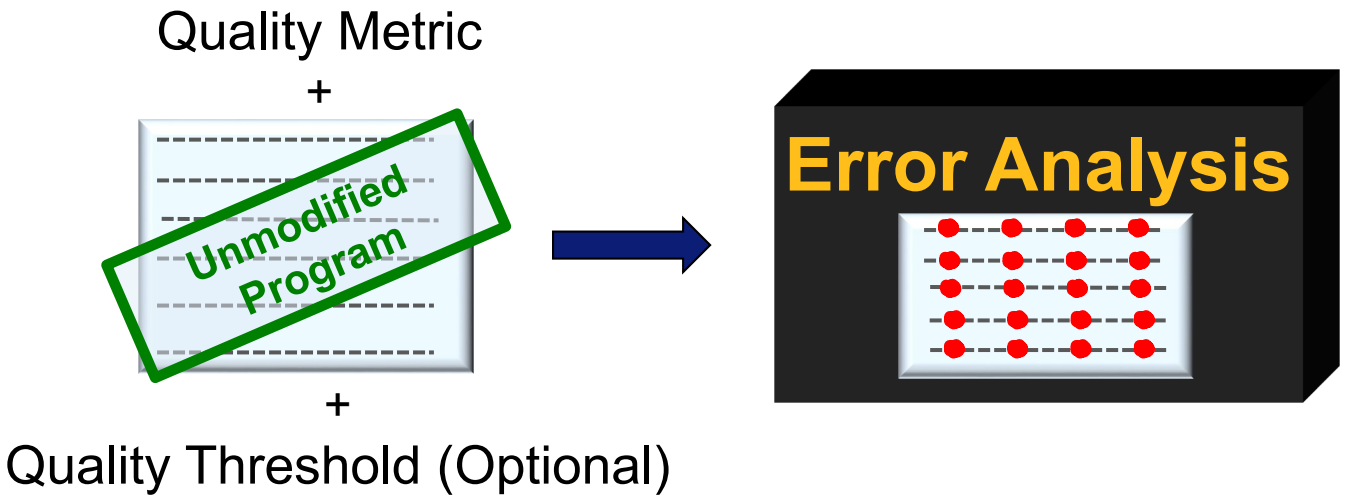
Error Model

- Perturbation in program state (instructions + data)
  - Caused by underlying fault in hardware
- Error Model for Instructions
  - Single bit transient errors in operand registers of dynamic instructions
- Error Model for Data
  - Multi-bit (random 1-bit, 2-bit, 4-bit, 8-bit) transient errors in memory

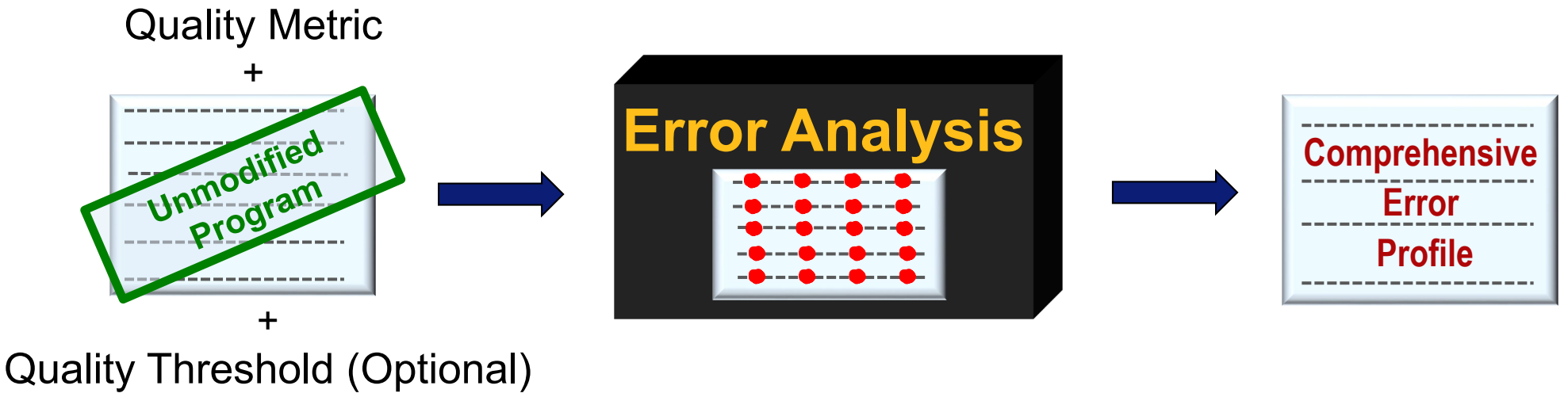
Error Analysis Output :

Application Data Error Profile + Application Instruction Error Profile

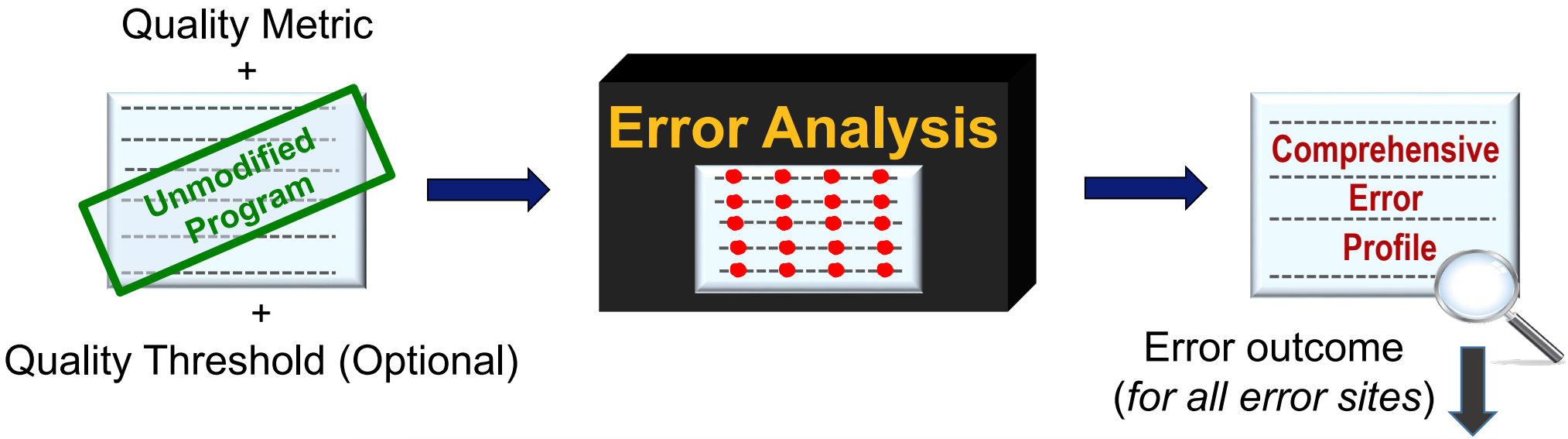
Error Analysis User Interface: Inputs



Error Analysis User Interface: Output

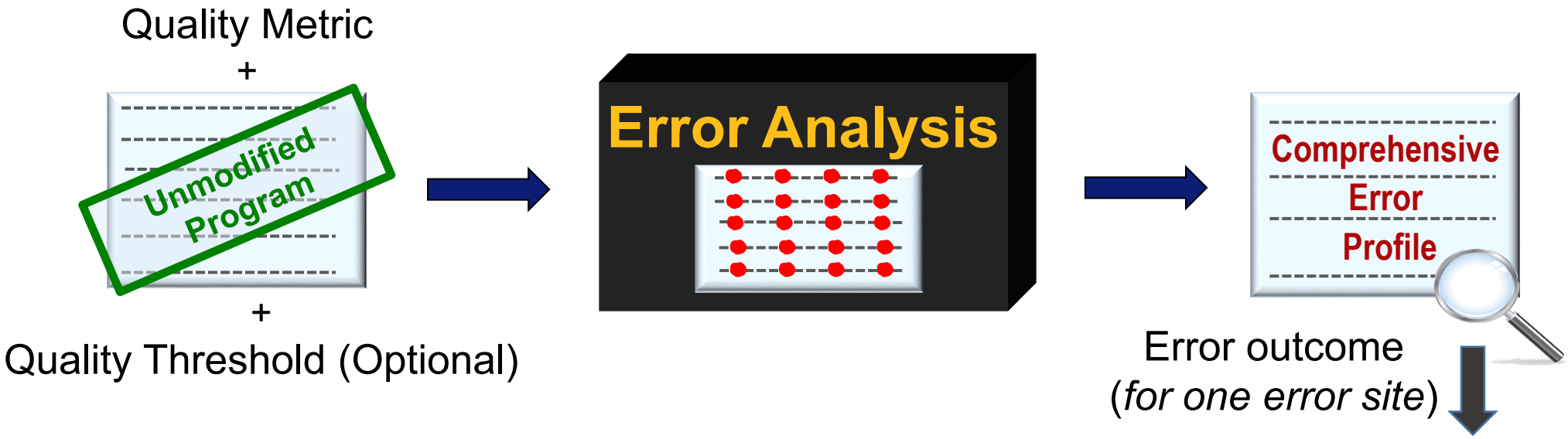


Comprehensive Error Profile




```
0x400995, 594769813038500, r8, 14, Integer, Source :: QD - 0.0218
```

Error Outcome for One Error Site

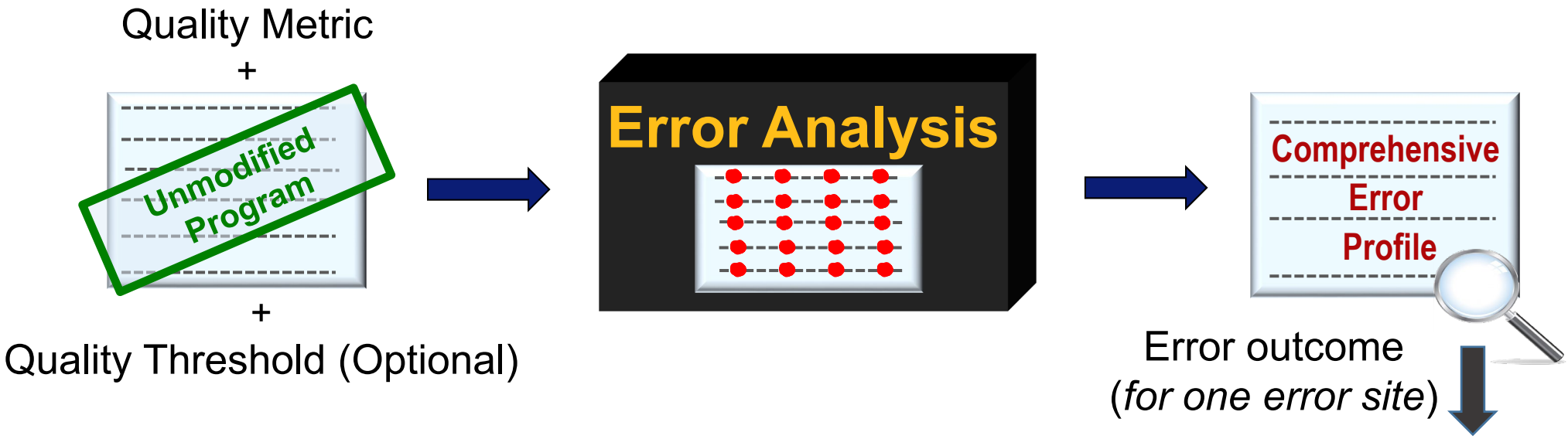



```
0x400995, 594769813038500, r8, 14, Integer, Source :: QD - 0.0218
```

Error Site Description

Error Outcome

Error Outcome for One Error Site



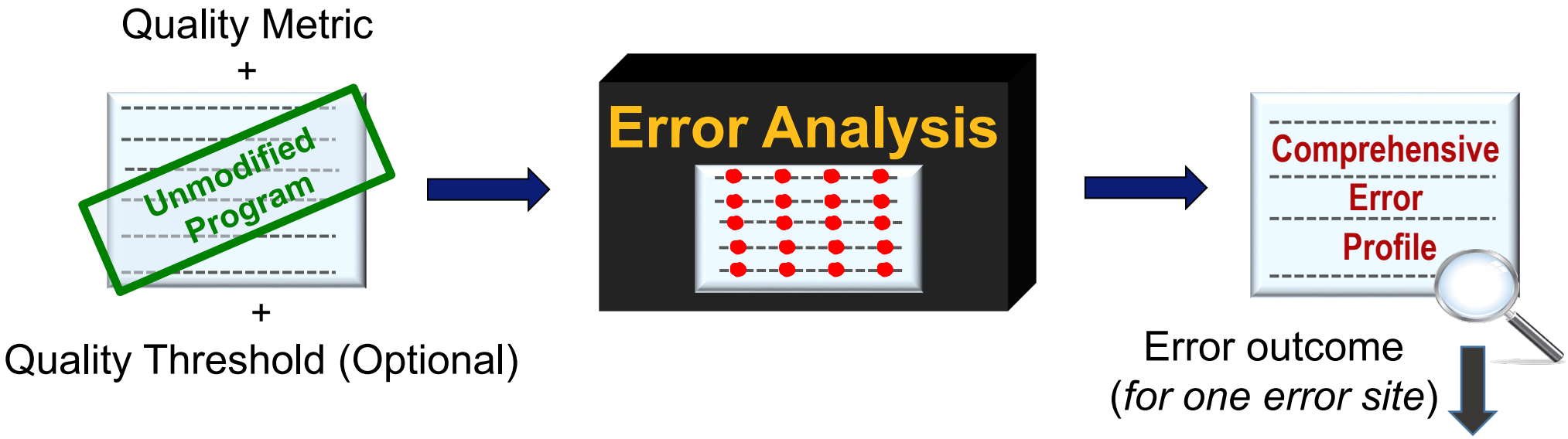
Error Model: Single bit errors in operand registers of dynamic instructions

```
0x400995, 594769813038500, r8, 14, Integer, Source :: QD - 0.0218
```

↑

Error Site Description

Error Outcome for One Error Site

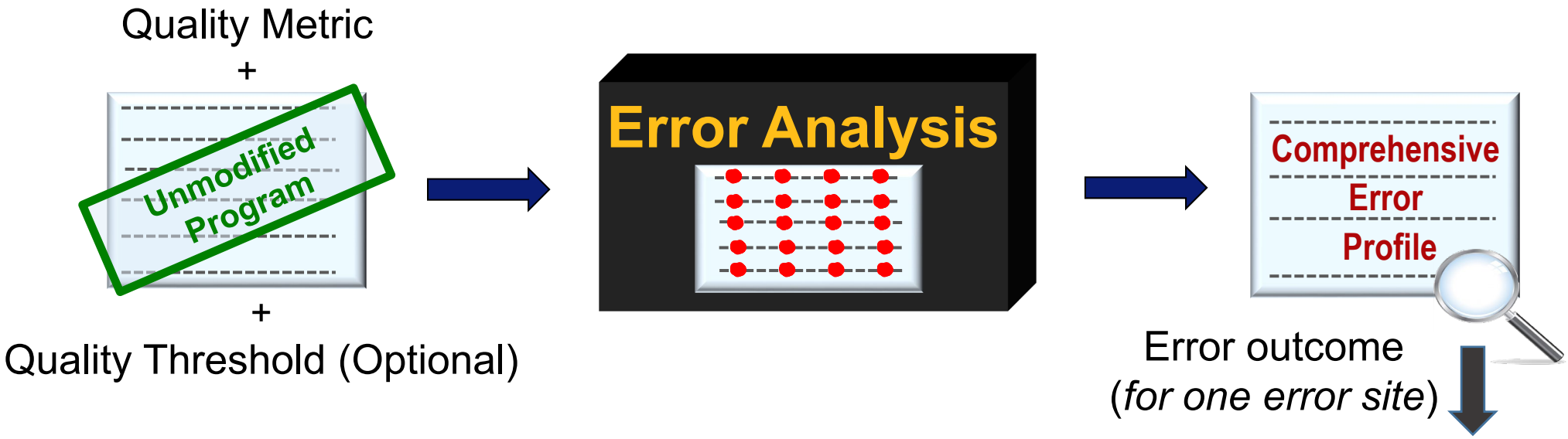


Error Site: Dynamic instruction + Operand Register + Register Bit

```
0x400995, 594769813038500, r8, 14, Integer, Source :: QD - 0.0218
```

PC + Cycle = Dynamic instruction

Error Outcome for One Error Site

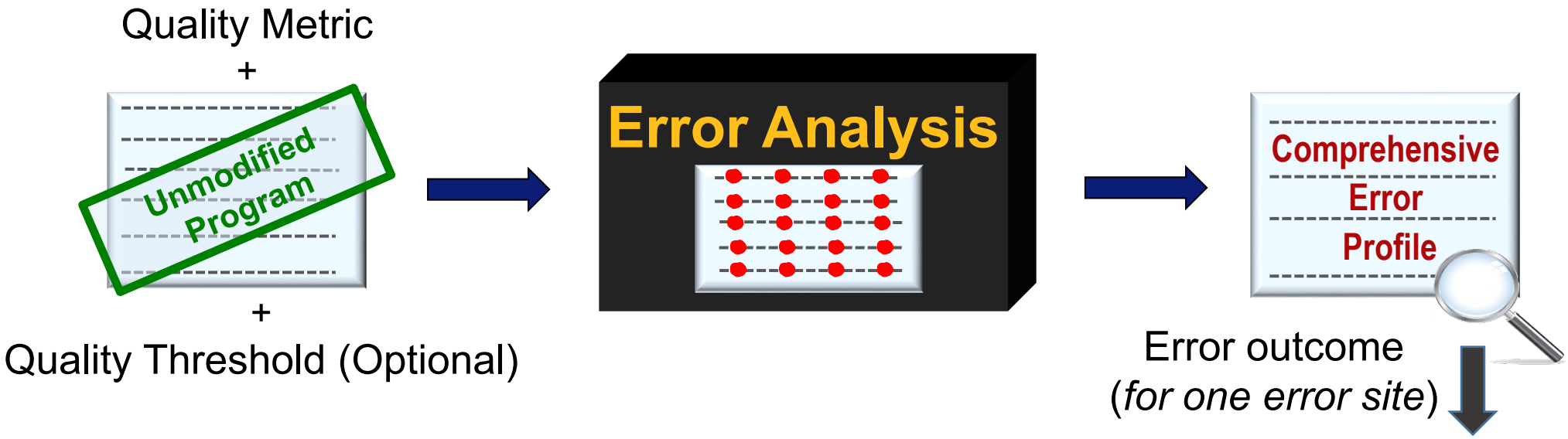


Error Site: Dynamic instruction + Operand Register + Register Bit

```
0x400995, 594769813038500, r8, 14, Integer, Source :: QD - 0.0218
```

Register Name **Register Bit**

Error Outcome for One Error Site

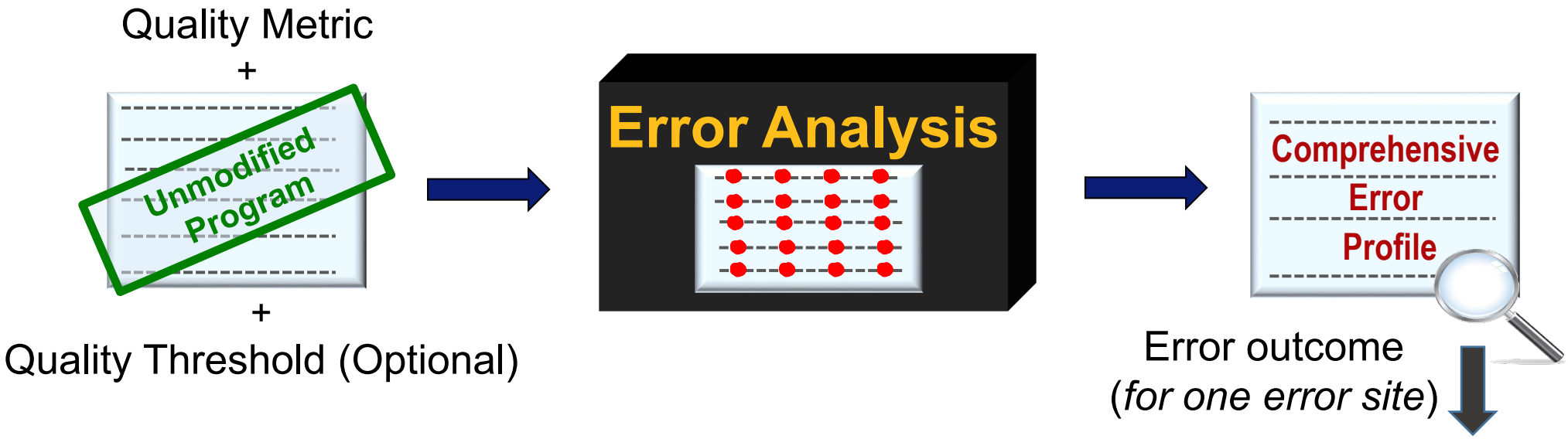


Error Site: Dynamic instruction + Operand Register + Register Bit

```
0x400995, 594769813038500, r8, 14, Integer, Source :: QD - 0.0218
```

Register Type **Operand Type**

Error Outcome for One Error Site

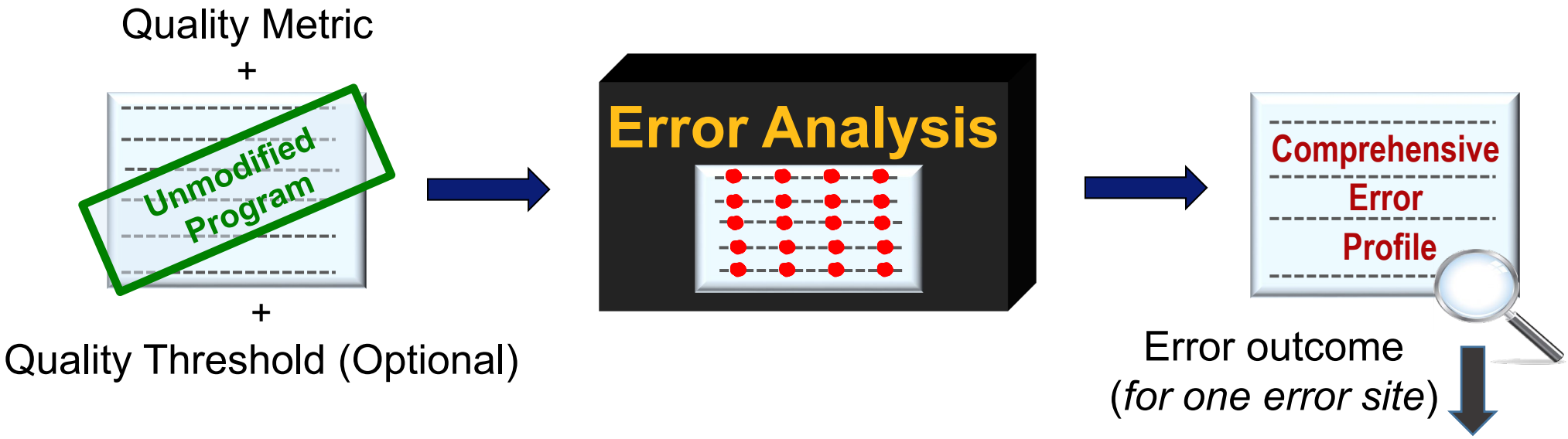


Error Outcome: Impact of an error, at this error site, on program output

```
0x400995, 594769813038500, r8, 14, Integer, Source :: QD - 0.0218
```

Error Outcome

Error Outcome for One Error Site

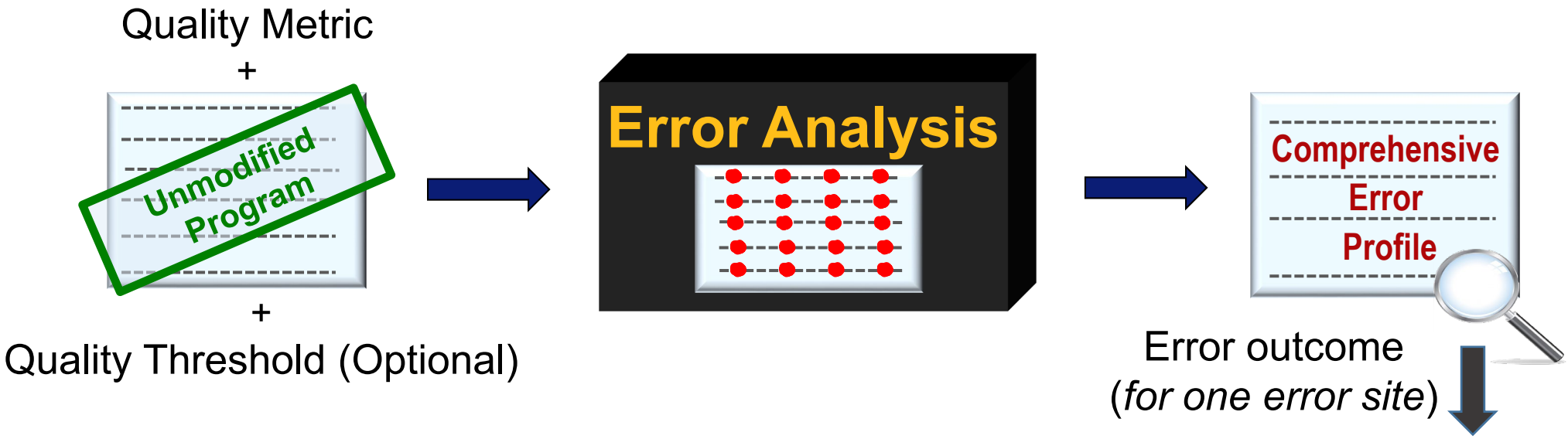


Error Outcome: Impact of an error, at this error site, on program output

```
0x400995, 594769813038500, r8, 14, Integer, Source :: QD - 0.0218
```

Quality Degradation

Error Outcome for One Error Site



Error Model: 1- bit transient error in (data bit stored) in DRAM

Address Bit

`0x40a670, 342769813038500, Read, 0x6a10a, 2, 7 :: QD - 0.0008`

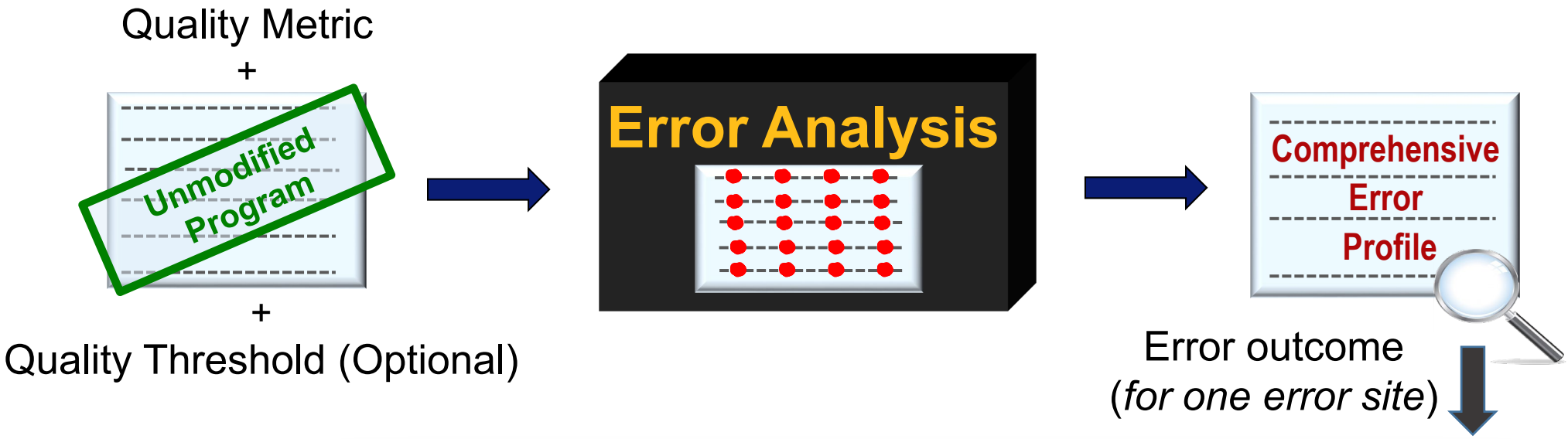
PC + Cycle = Dynamic instruction

Access Type

Byte Offset

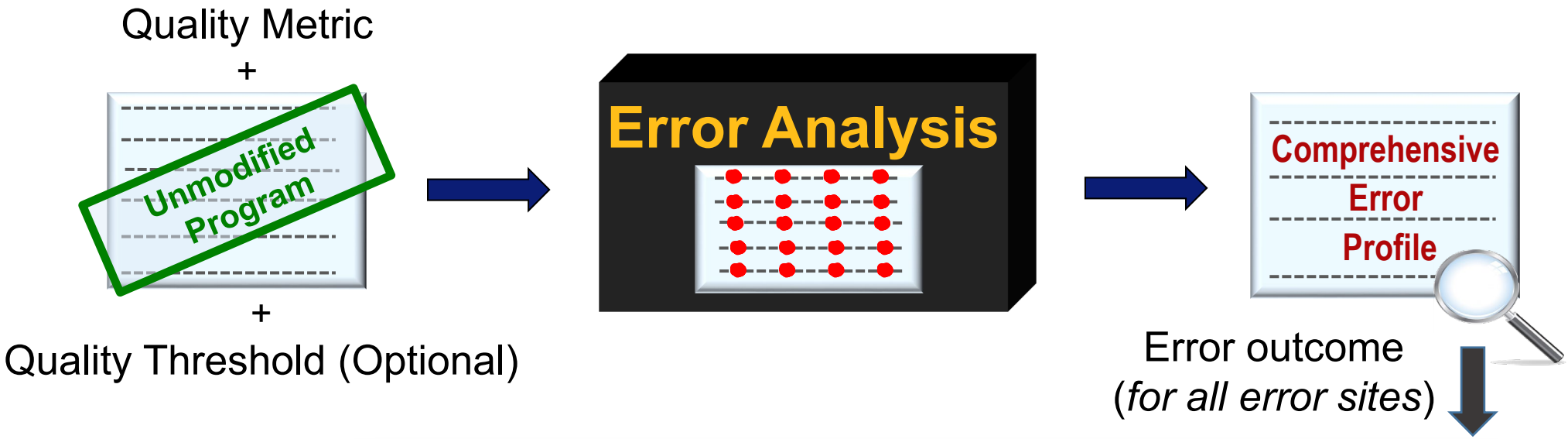
Quality Degradation

Error Outcome for One Error Site




```
0x400995, 594769813038500, r8, 14, Integer, Source :: QD - 0.0218
```

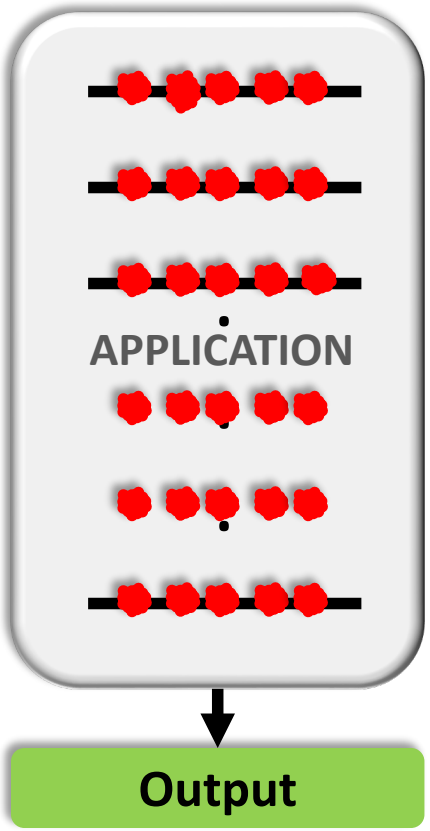
Error Outcome for All Error Sites




```
radha — venktgr2@veena-server:/shared/workspace/venktgr2/GEM5_Approxilyzer/Approxilyz...
...5_Approxilyzer/Approxilyzer/workloads/x86/apps/sobel_bird — ssh -Y venktgr2@veena-server.cs.illinois.edu
4009a7,594766177333000,rsi,58,Integer,Destination::Detected:segfault
400fdd,594780406967500,edx,20,Integer,Destination::Masked
400970,594764472062500,eax,17,Integer,Destination::SDC-Maybe:0.000638
400990,594758560610500,rcx,25,Integer,Source::Detected:segfault
400fd2,594778156118500,r8,29,Integer,Destination::Masked
400995,594769813038500,r8,14,Integer,Source::SDC-Maybe:0.0218
4009e6,594777713444500,xmm0,0,Float,Source::Masked
4009de,594767982584500,xmm3,8,Float,Source::Masked
400fac,594780406956000,rax,28,Integer,Source::Detected:segfault
400a0d,594771221095000,xmm0,56,Float,Source::Masked
4009b6,594774185966500,r8,11,Integer,Source::SDC-Maybe:0.000287
400f83,594778382630500,rdx,45,Integer,Destination::Masked
                                                    379628,1      99%
```

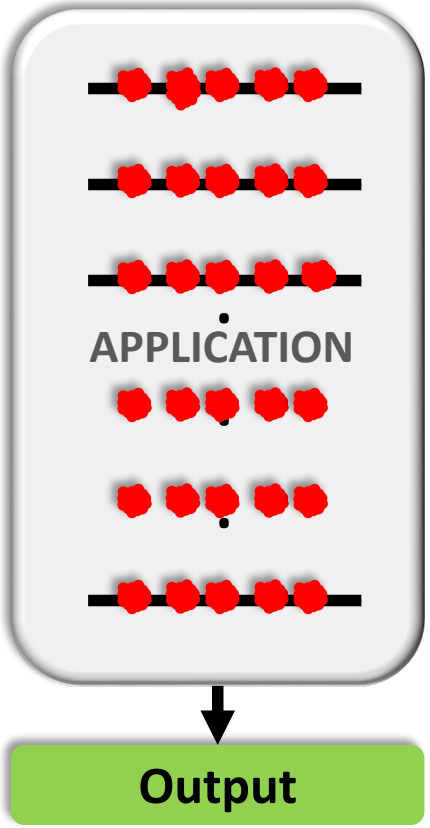
Billions of error sites in average programs → Error injections in all expensive!

Error Pruning Using Equivalence

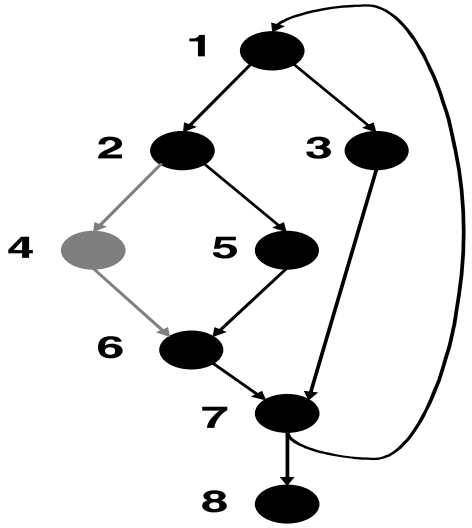


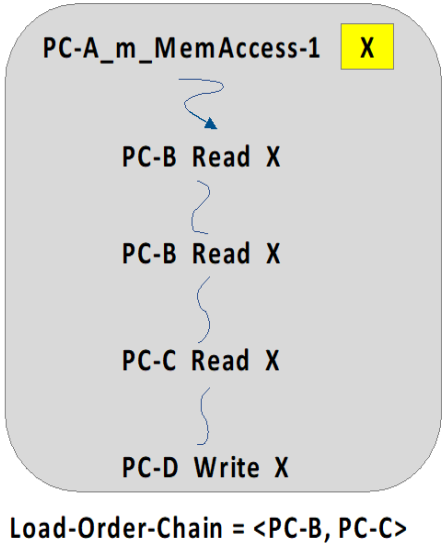
Errors flowing through similar control+data paths produce similar outcomes

Error Pruning Using Equivalence



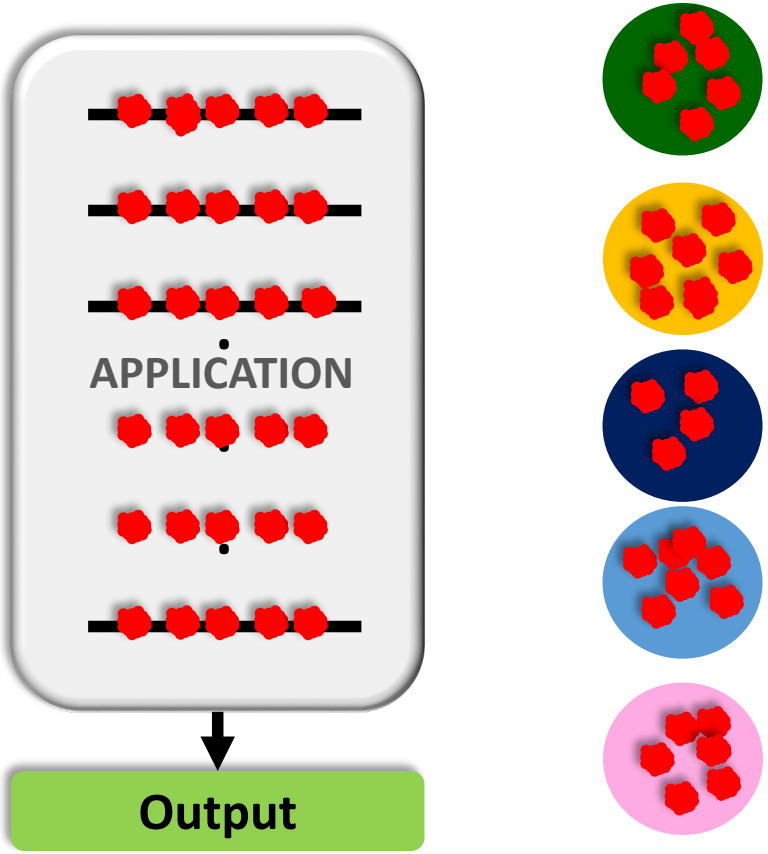
Equivalence Classes (control + data heuristics)



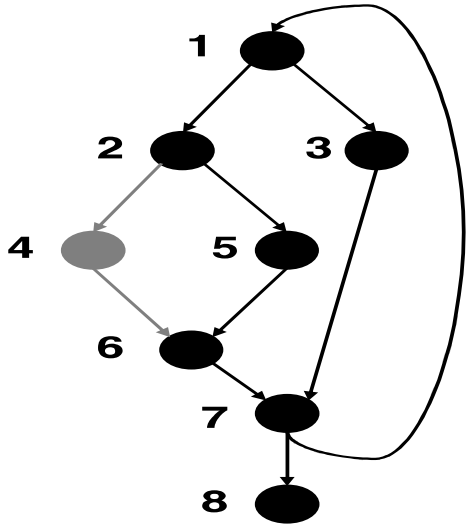


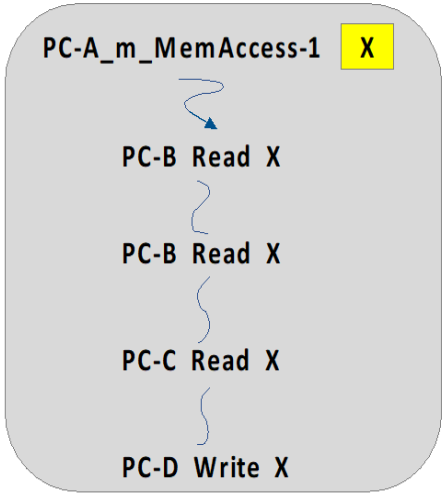
Errors flowing through similar control+data paths produce similar outcomes

Error Pruning Using Equivalence



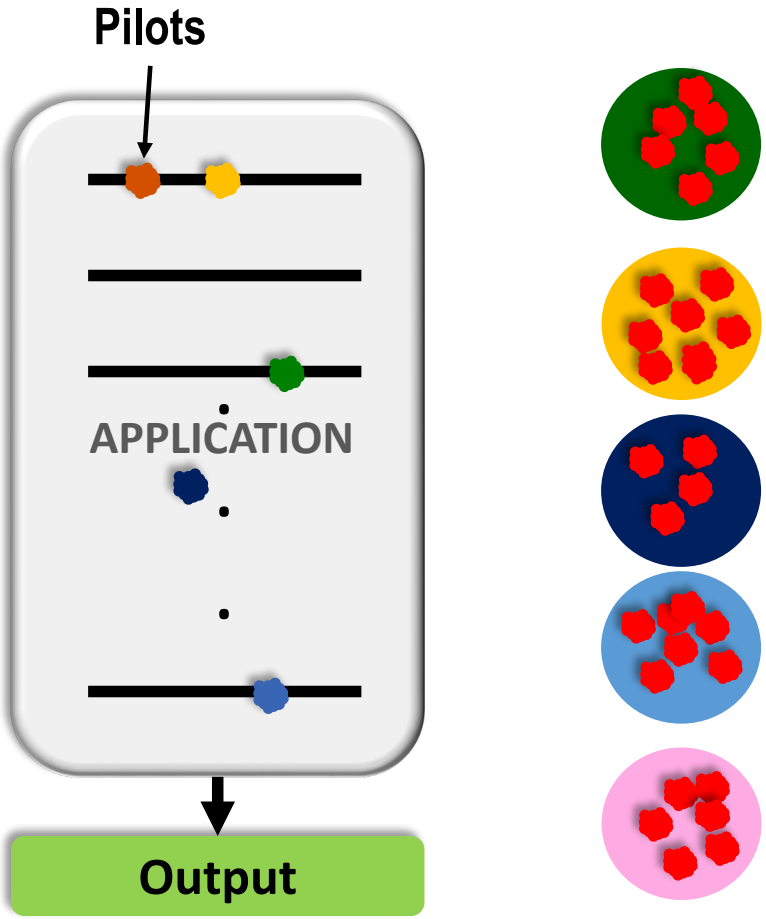
Equivalence Classes (control + data heuristics)





Errors flowing through similar control+data paths produce similar outcomes

Error Pruning Using Equivalence

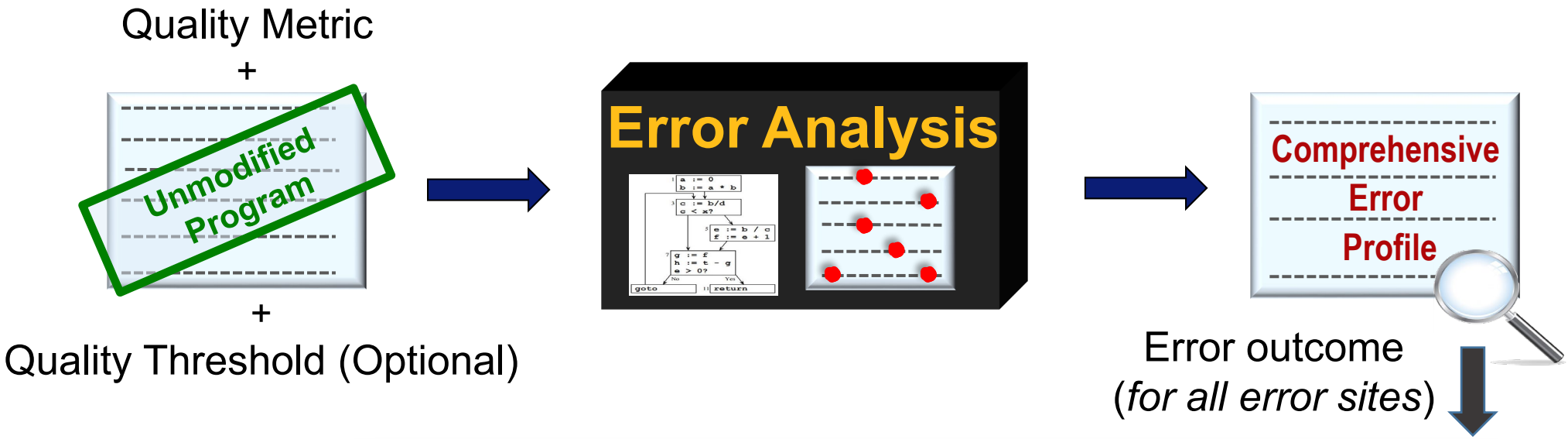


Inject error in Pilot

Pilot outcome = Outcome of all errors in class

Few error injections to predict the outcome of all errors

Comprehensive Error Profile with Few Injections



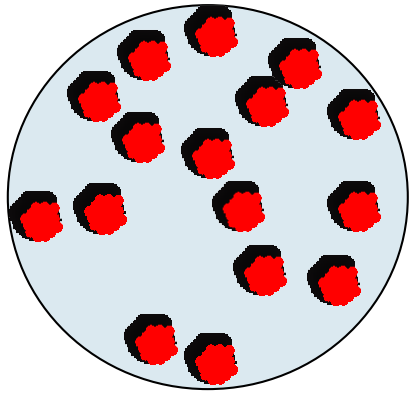

```
radha — venktgr2@veena-server:/shared/workspace/venktgr2/GEM5_Approxilyzer/Approxilyz...
...5_Approxilyzer/Approxilyzer/workloads/x86/apps/sobel_bird — ssh -Y venktgr2@veena-server.cs.illinois.edu
4009a7,594766177333000,rsi,58,Integer,Destination::Detected:segfault
400fdd,594780406967500,edx,20,Integer,Destination::Masked
400970,594764472062500,eax,17,Integer,Destination::SDC-Maybe:0.000638
400990,594758560610500,rcx,25,Integer,Source::Detected:segfault
400fd2,594778156118500,r8,29,Integer,Destination::Masked
400995,594769813038500,r8,14,Integer,Source::SDC-Maybe:0.0218
4009e6,594777713444500,xmm0,0,Float,Source::Masked
4009de,594767982584500,xmm3,8,Float,Source::Masked
400fac,594780406956000,rax,28,Integer,Source::Detected:segfault
400a0d,594771221095000,xmm0,56,Float,Source::Masked
4009b6,594774185966500,r8,11,Integer,Source::SDC-Maybe:0.000287
400f83,594778382630500,rdx,45,Integer,Destination::Masked
                                                    379628,1      99%
```

Up to 5 orders of magnitude reduction is error injections

Validation of Equivalence Heuristics

- Heuristics used to build equivalence classes need validation
  - Does the pilot accurately represent its equivalence class?

Equivalence class (EC)

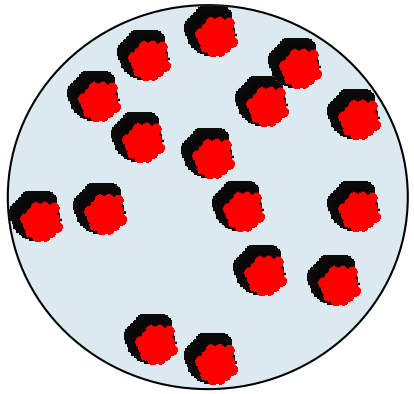


Pilot (Representative error-site from EC)

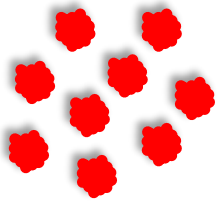
Validation of Equivalence Heuristics

- Heuristics used to build equivalence classes need validation
  - Does the pilot accurately represent its equivalence class?

Equivalence class (EC)



Pilot (Representative error-site from EC)

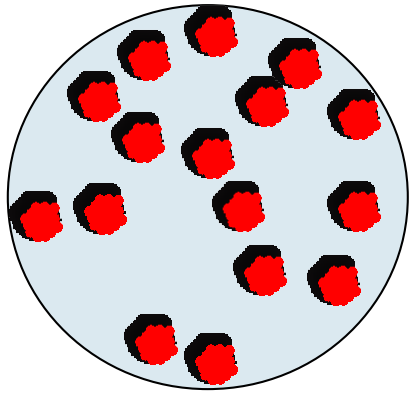


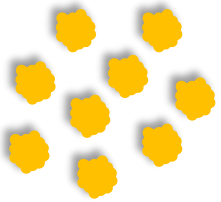
Population

Validation of Equivalence Heuristics

- Heuristics used to build equivalence classes need validation
  - Does the pilot accurately represent its equivalence class?

Equivalence class (EC)





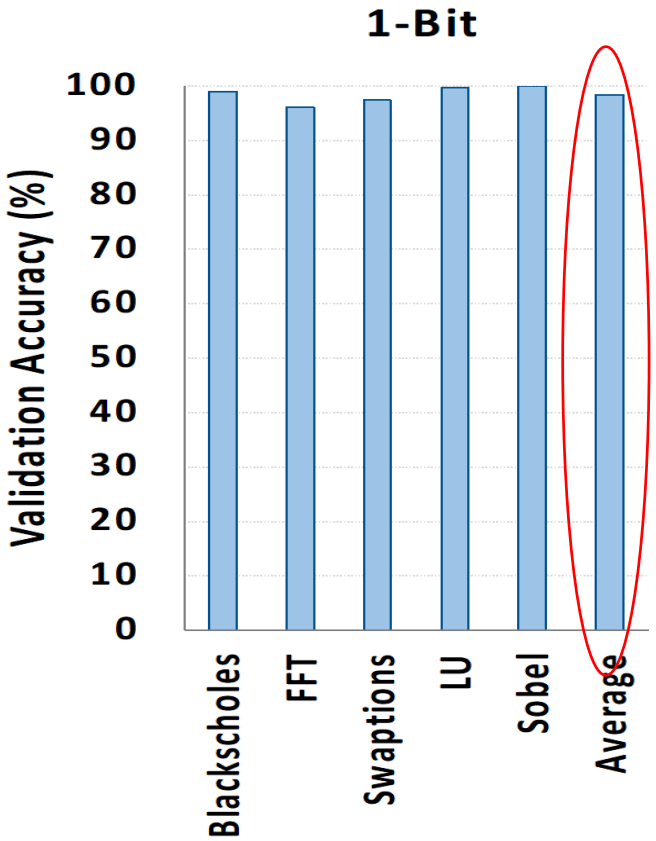
Pilot = Error Outcome E

100% Validation Accuracy

All in Population = Error Outcome E

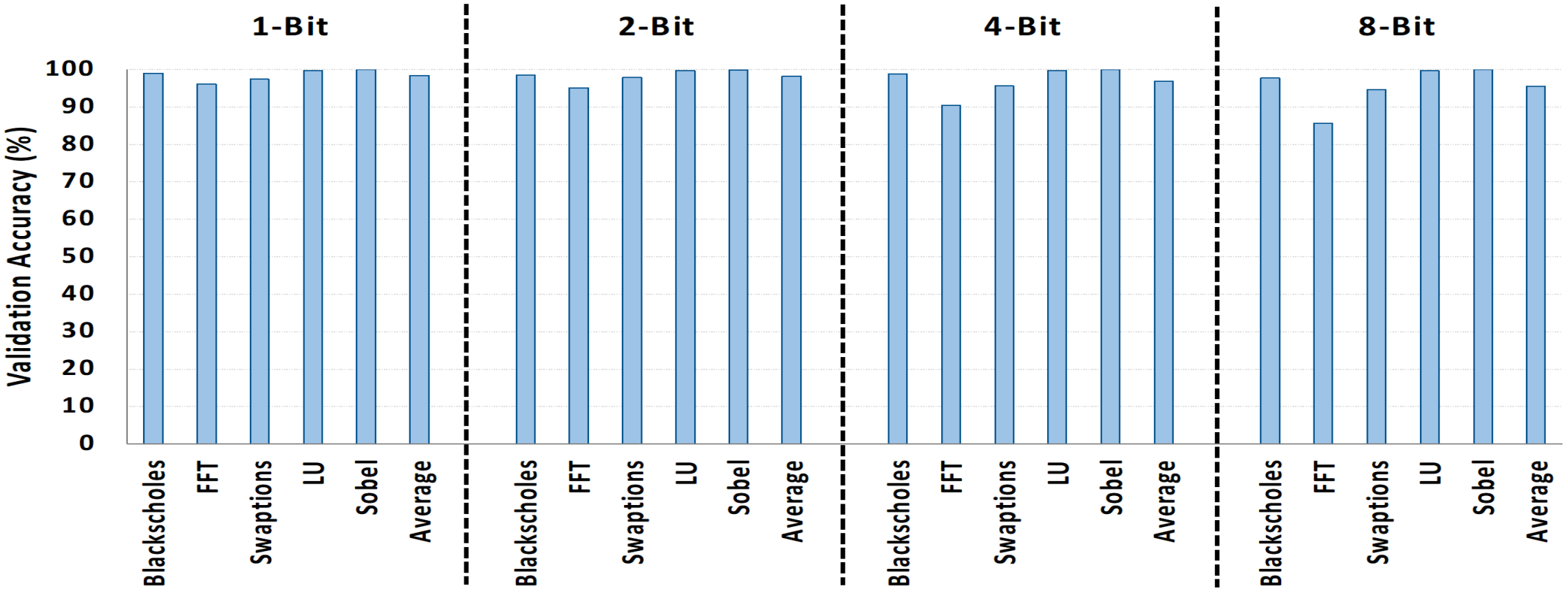
~ 7 million error injections to validate this technique

Validation Accuracy: Data Errors



On average, >97% (up to 99%) validation accuracy

Validation Accuracy: Data Errors



On average, >97% (up to 99%) validation accuracy

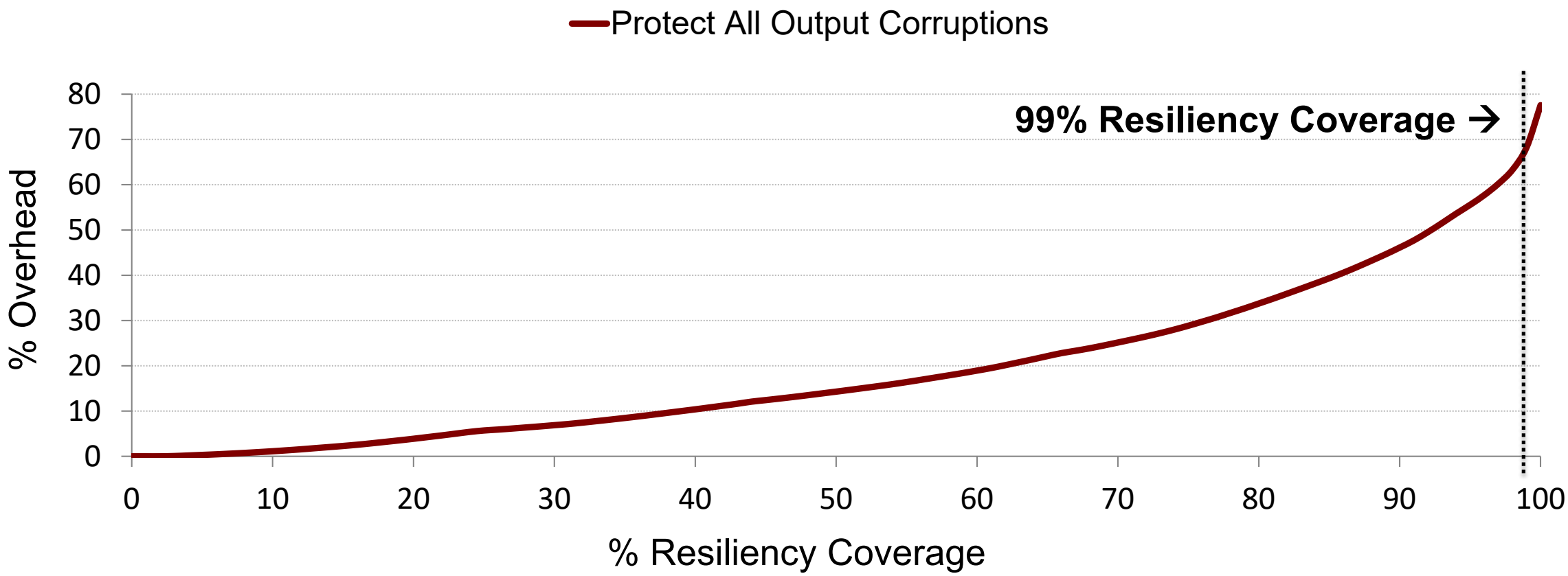
Customized Error Efficiency: Use Case 1

Instruction Error Profile → Customized Ultra Low-Cost Resiliency

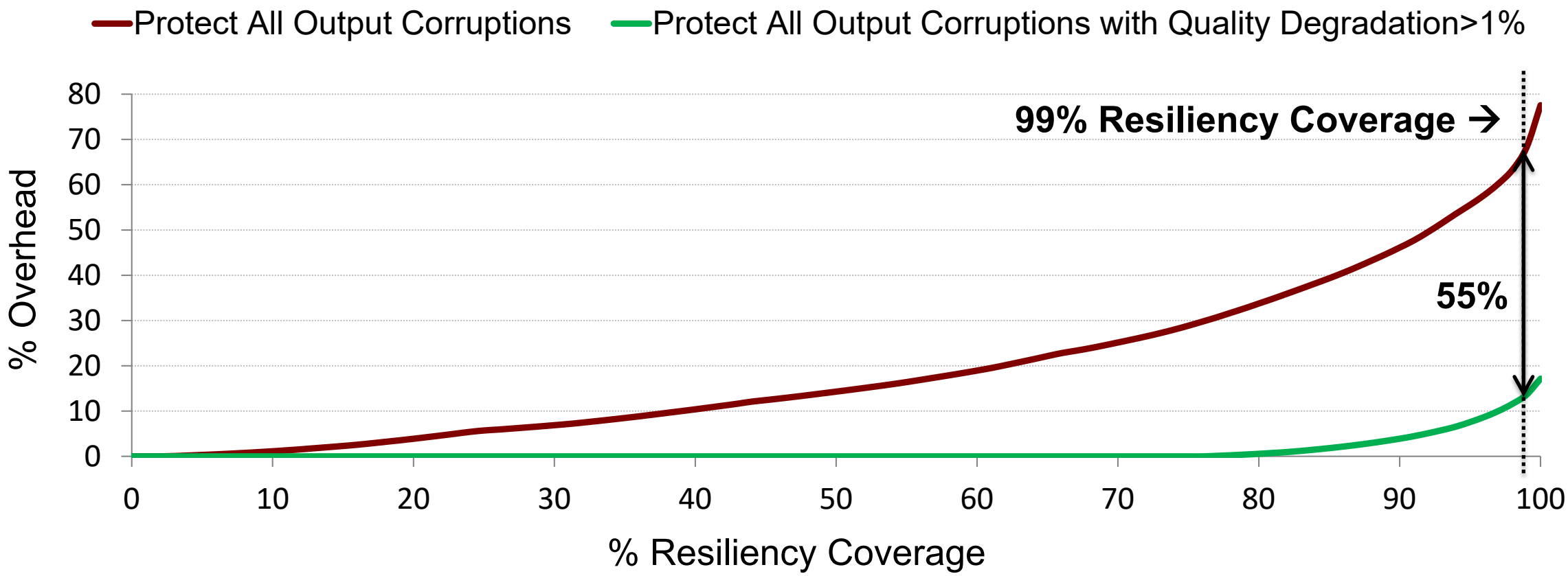
- **Selectively protect instructions**
  - End-to-end output quality is not acceptable to user/application
  - Protection Scheme: Instruction duplication
  - Less instructions protected → Reduced resiliency overhead

- **Optimal (custom) resiliency solution**
  - Quality vs. resiliency coverage vs. overhead

Ultra-Low Cost Resiliency (Water)



Ultra-Low Cost Resiliency (Water)



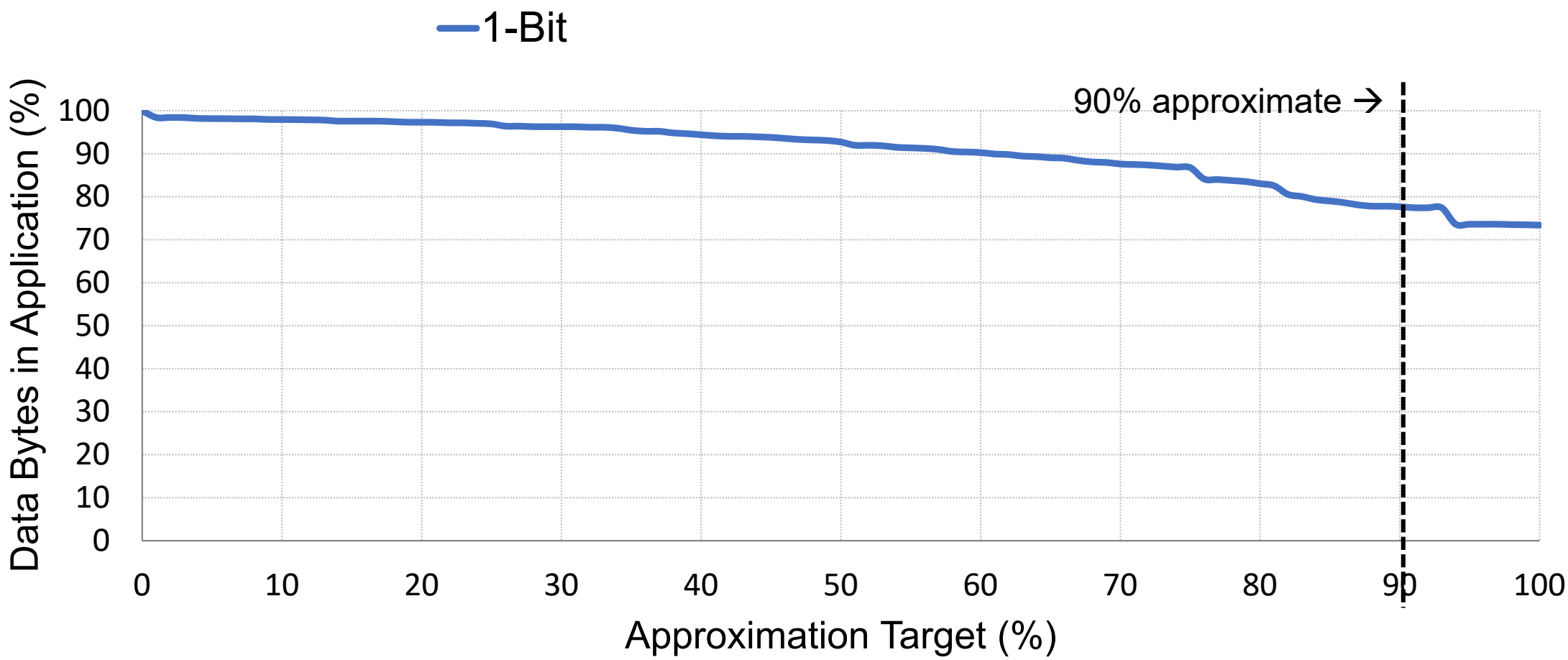
Significant resiliency overhead savings for small loss of quality

Customized Error Efficiency: Use Case 2

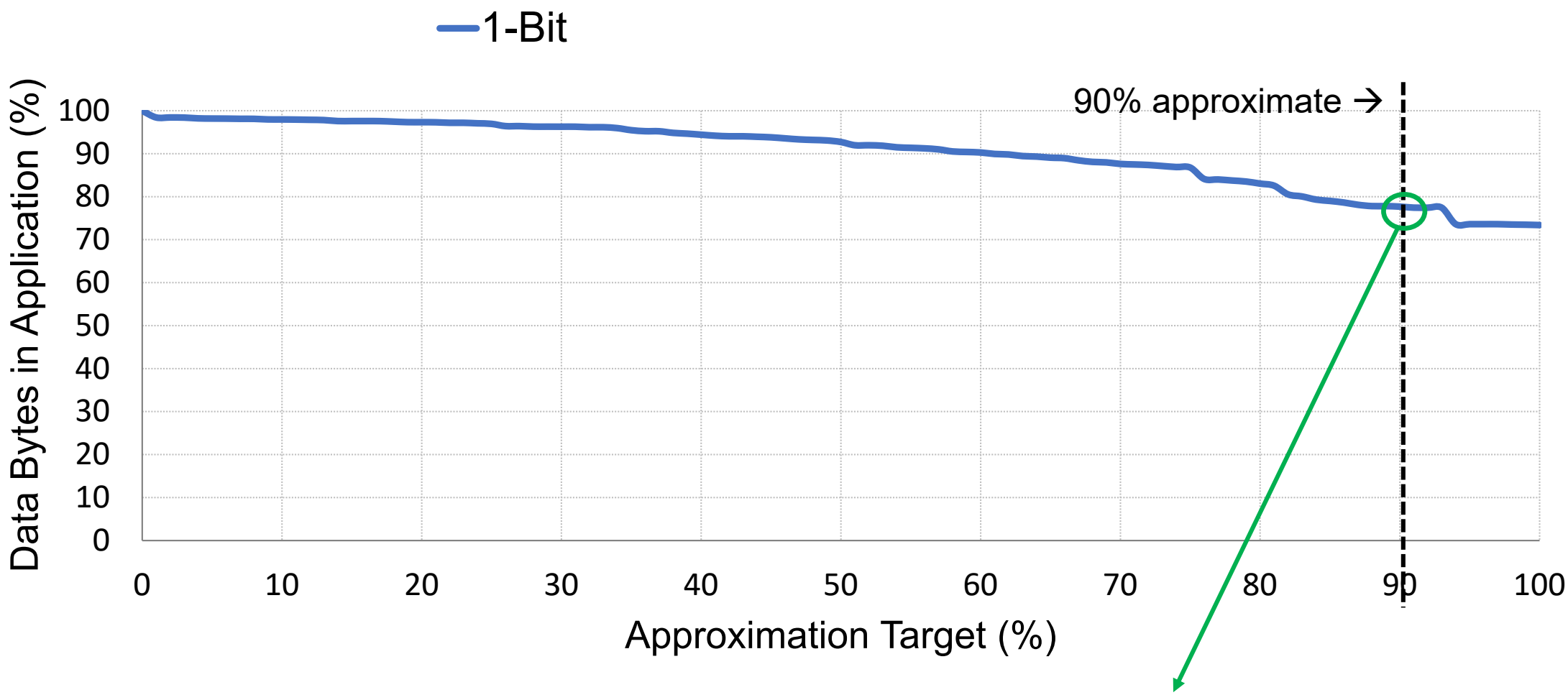
Data Error Profile → Approximate Computing

Identify first-order approximable data in a program

Customized Approximate Computing (FFT)

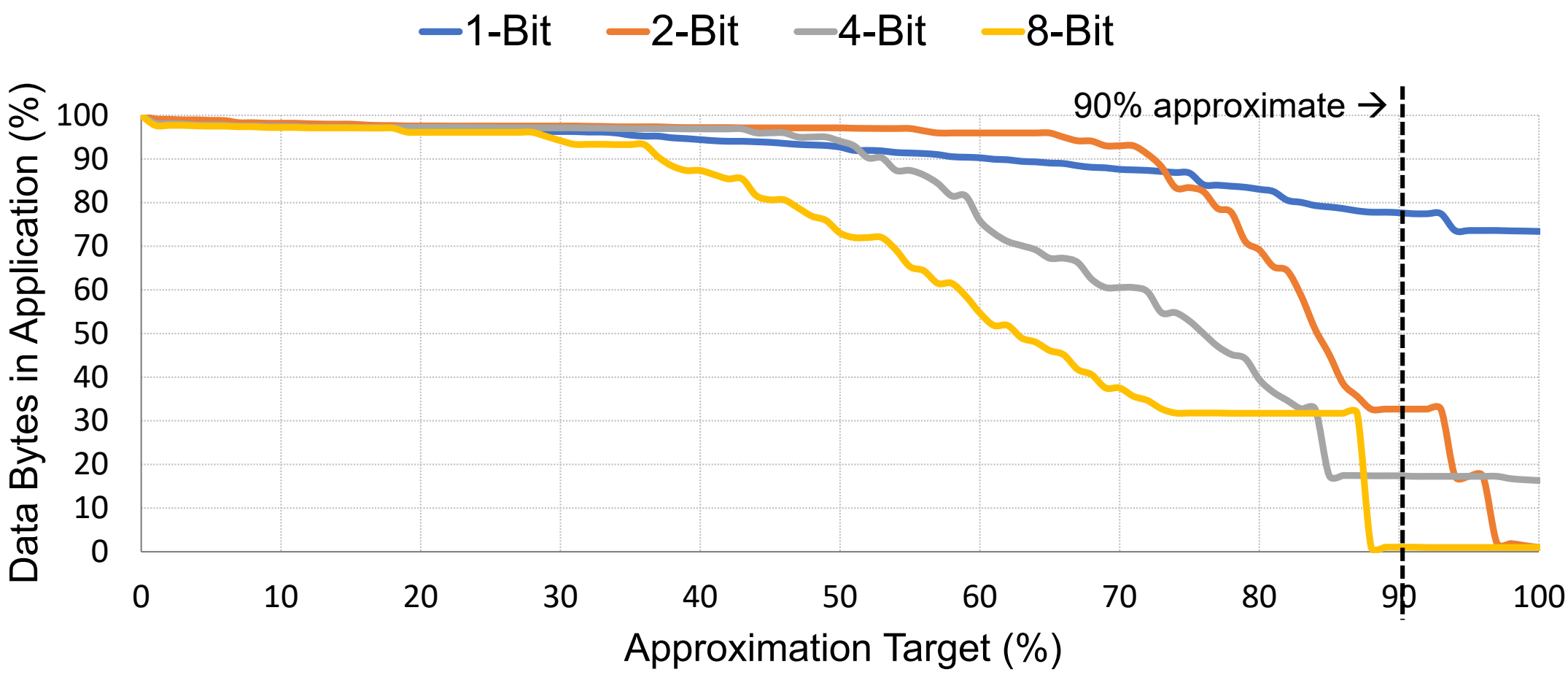


Customized Approximate Computing (FFT)

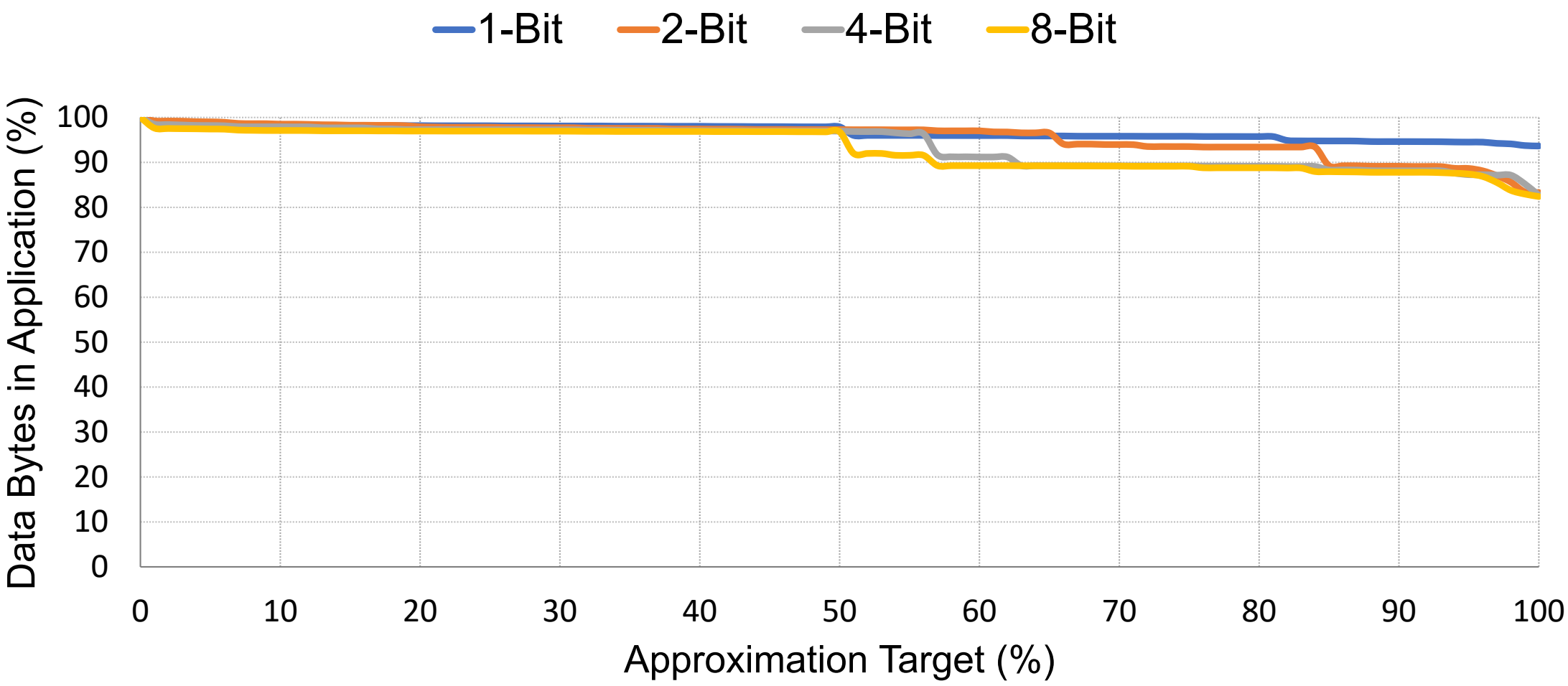


77% of data bytes are approximable 90% of the time when corrupted with a single-bit error

Customized Approximate Computing (FFT)



Customized Approximate Computing (Swaptions)



Mapping Data to Approximate Memory

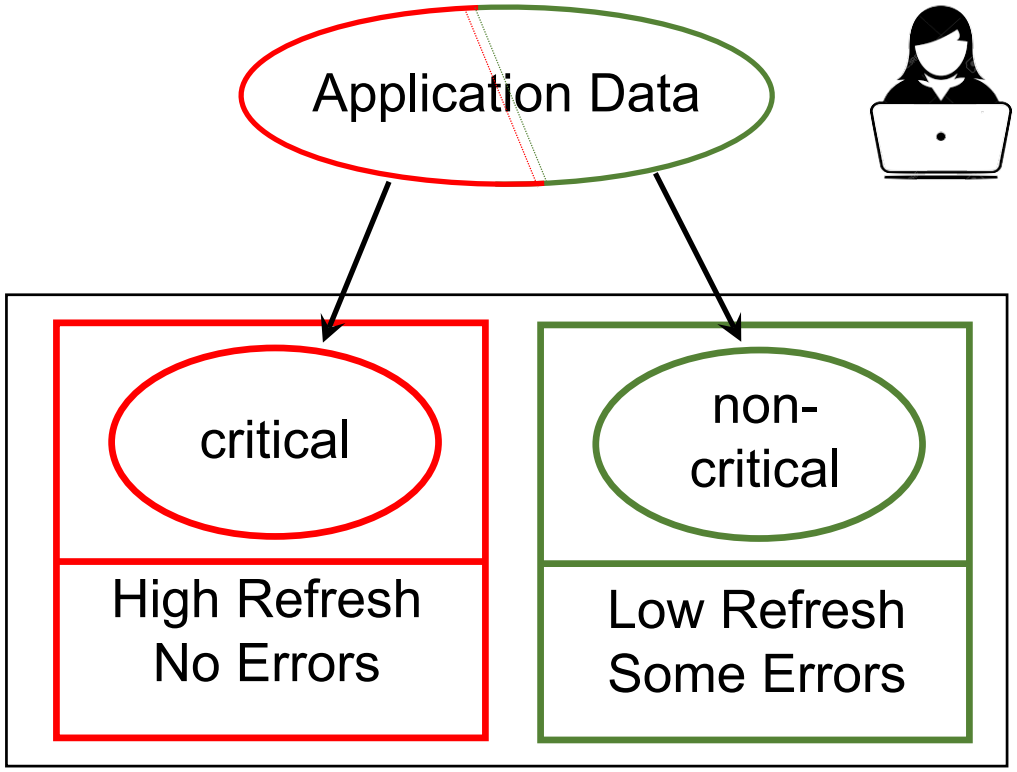
Approximate Memory Technique → Lower DRAM refresh rate to save power*

*Flikker [ASPLOS'11]

Mapping Data to Approximate Memory

Approximate Memory Technique → Lower DRAM refresh rate to save power*

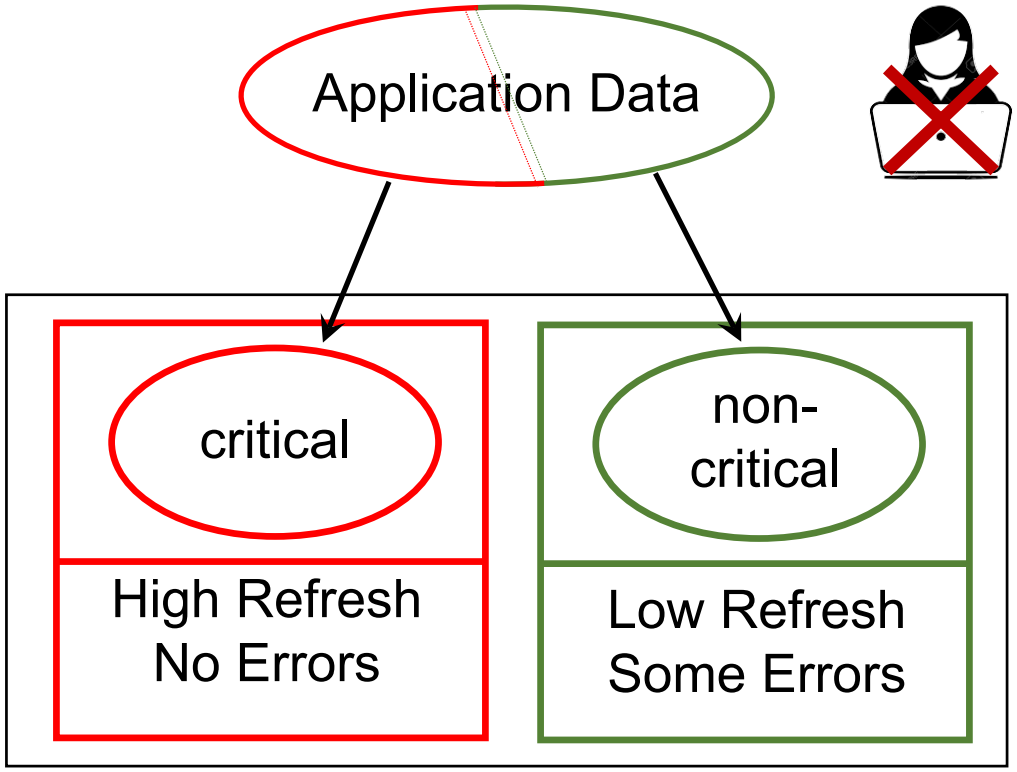
*Flikker [ASPLOS'11]



Mapping Data to Approximate Memory

Approximate Memory Technique → Lower DRAM refresh rate to save power*

*Flikker [ASPLOS'11]



Automatic identification of Critical data

Swaptions

Quality Threshold = $0.001

Mapping Accuracy = 99.9%

Power Savings = 23%

Outline

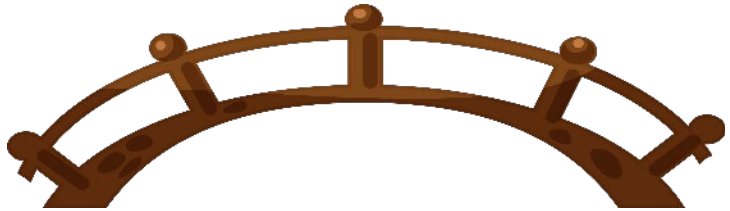
- Software-centric error analysis and error efficiency: Approxilyzer, Winnow
- Software testing for hardware errors: Minotaur
- Domain-specific error efficiency: HarDNN
- Compiler and runtime for hardware and software error efficiency: ApproxTuner
- Putting it Together: Towards a Discipline for Error-Efficient Systems

Minotaur: Key Idea

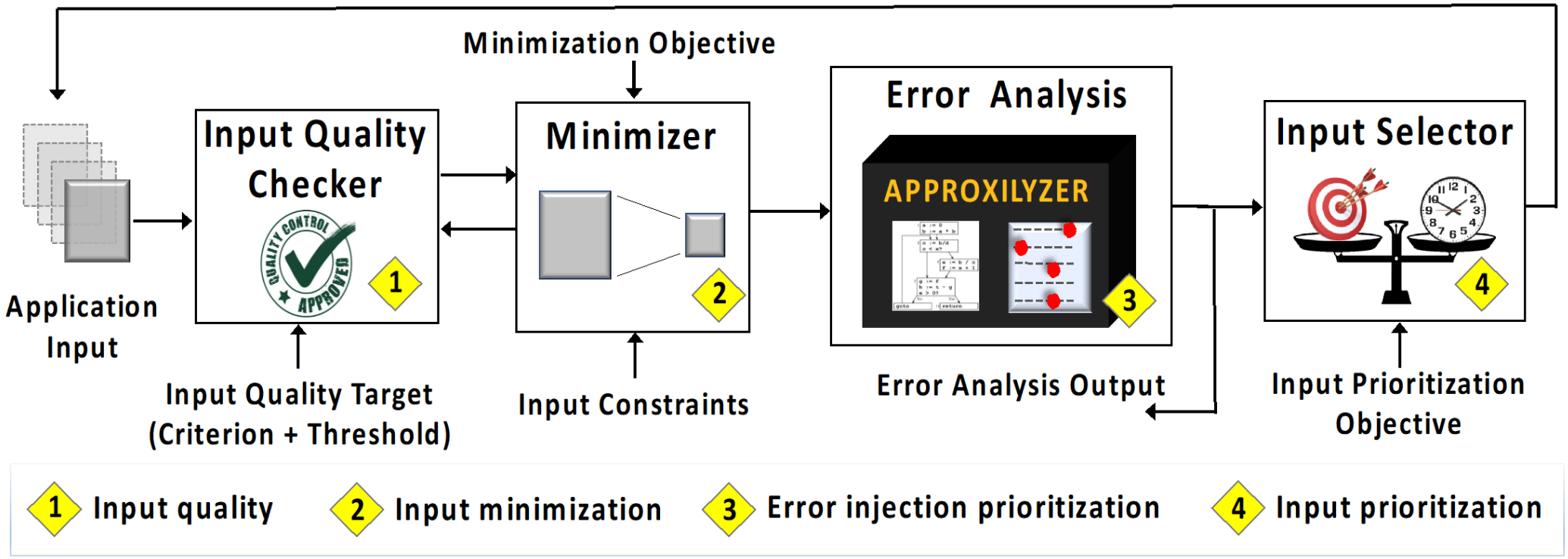
Analyzing software for…

…hardware errors $\approx$ …software bugs

Leverage software testing techniques to improve hardware error analysis

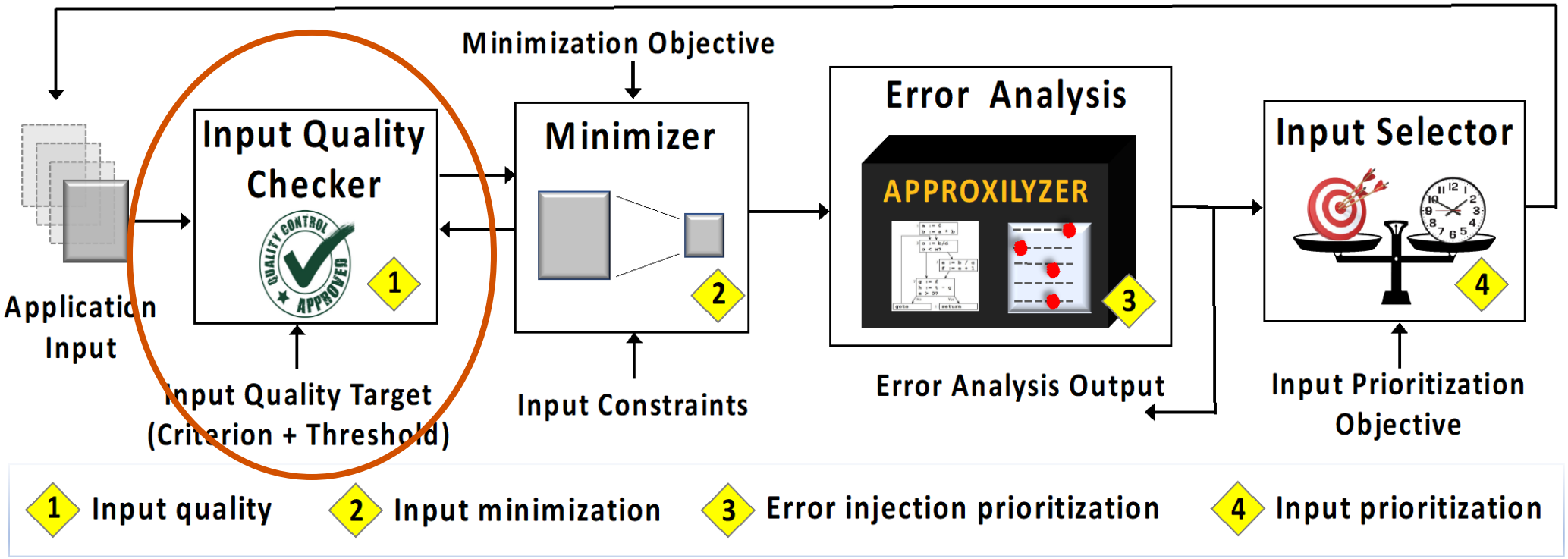
Software Testing  Hardware Error Analysis

Minotaur



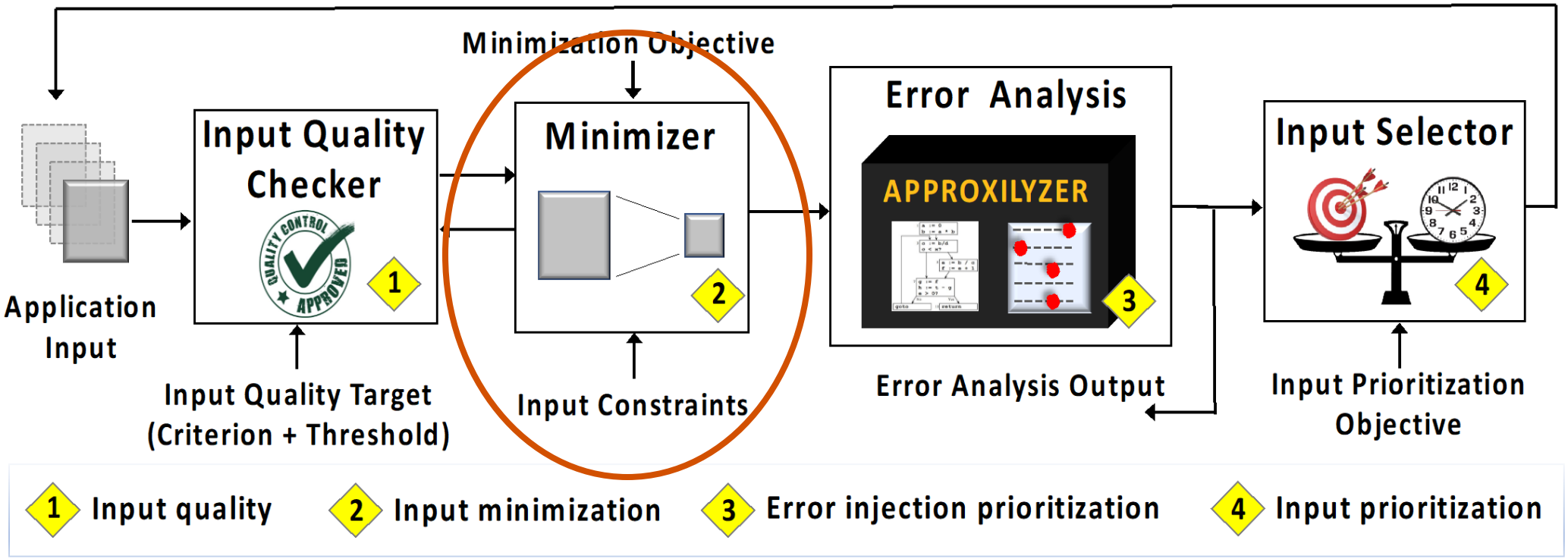
Adapts four software testing techniques to hardware error analysis

Minotaur



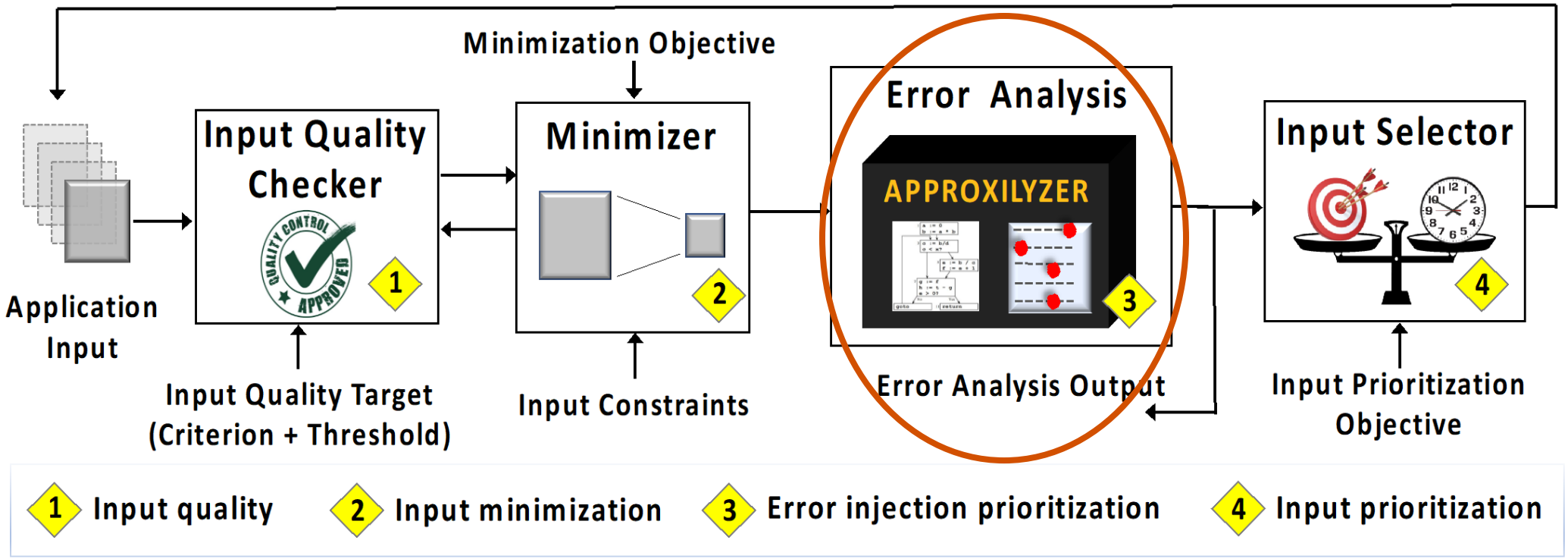
Input Quality for Error Analysis → PC coverage

Minotaur



High quality (fast) minimized inputs from (slow) standard inputs

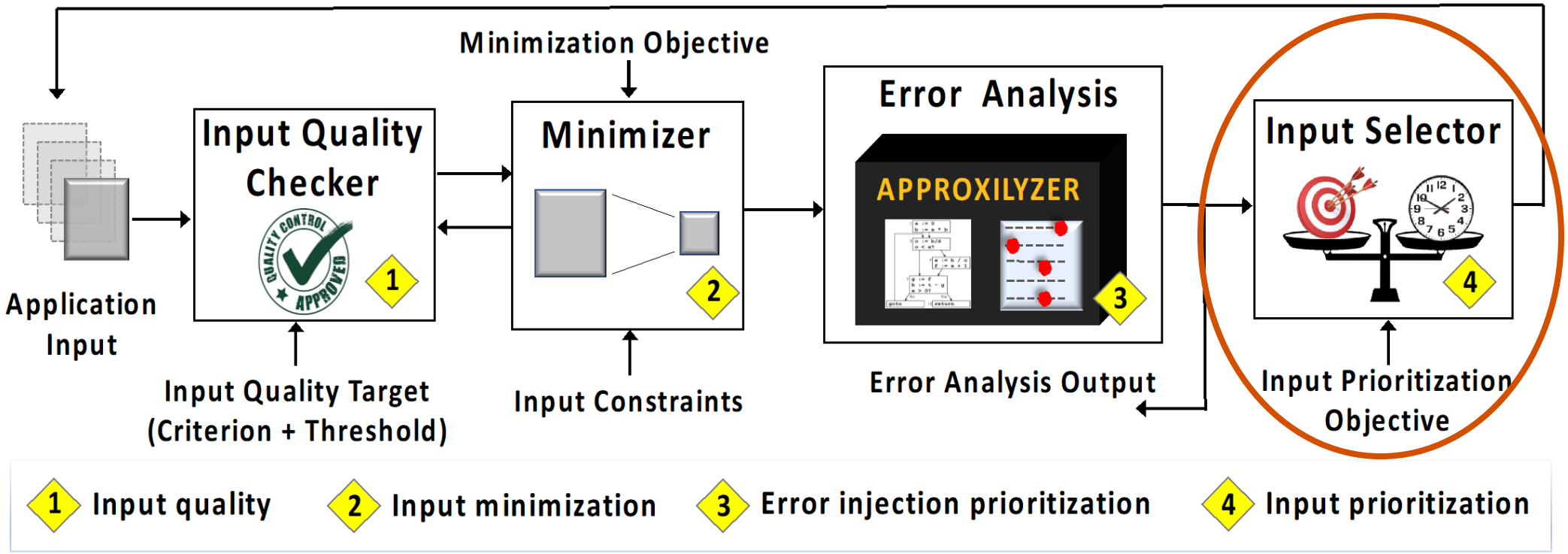
Minotaur



Prioritize analyzing specific program locations based on analysis objectives

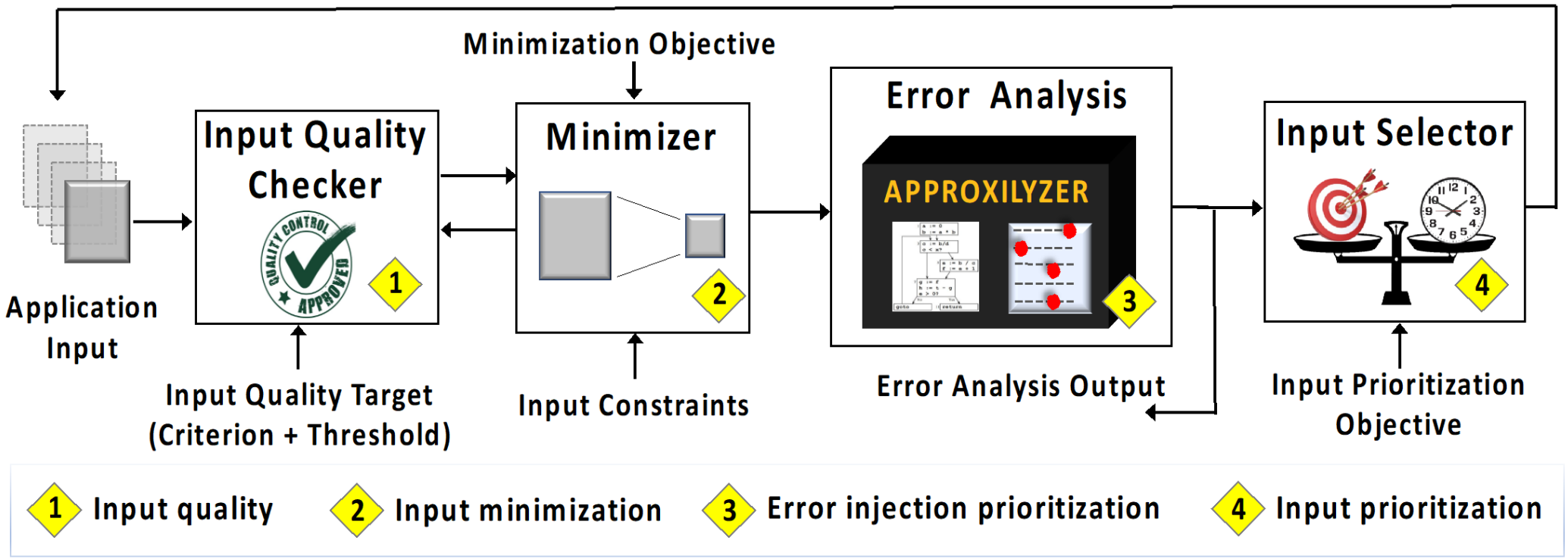
Terminate analysis (early) when objective is met

Minotaur



Prioritize analysis over fast, (potentially) inaccurate inputs first

Minotaur



4X average speedup in error analysis

10x average speedup (upto 39x) for analysis targeting low-cost resiliency

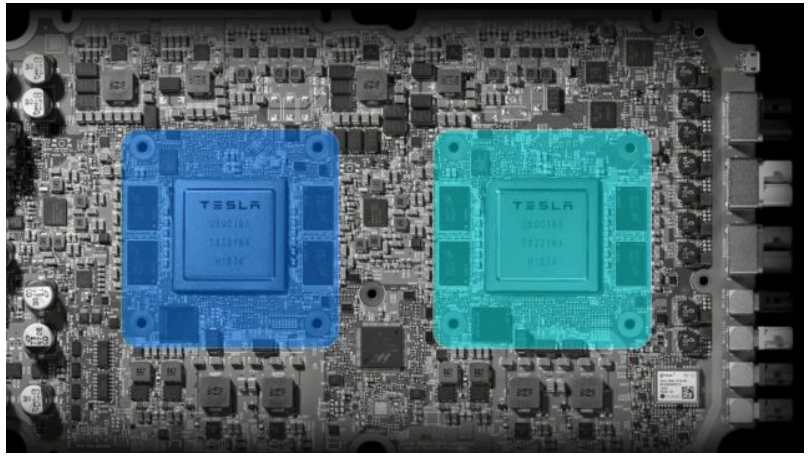
18x average speedup (up to 55x) for analysis targeting approximate computing

Outline

- Software-centric error analysis and error efficiency: Approxilyzer, Winnow
- Software testing for hardware errors: Minotaur
- Domain-specific error efficiency: HarDNN
- Compiler and runtime for hardware and software error efficiency: ApproxTuner
- Putting it Together: Towards a Discipline for Error-Efficient Systems

Deep Neural Networks (DNNs)

- Deep Neural Networks (DNNs) used in many application domains
  - Entertainment/personal devices to safety-critical autonomous cars
  - DNN software accuracy is < 100%: ResNet50 on ImageNet is ~76% accurate
  - But must execute "reliably" in the face of hardware errors
- Traditional reliability solution:



https://www.extremetech.com/extreme/290029-tesla-well-have-full-self-driving-by-2020-robo-taxis-too

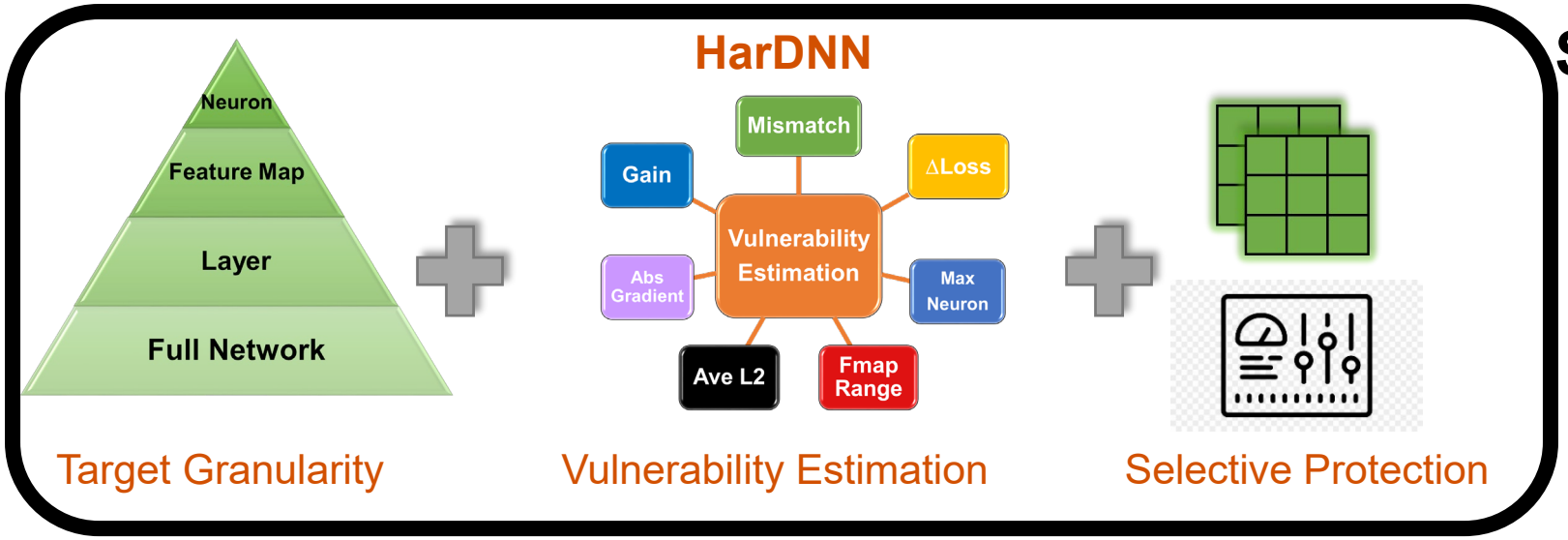
Tesla's Full Self-Driving Chip (FSD), 2019

~2X Overhead in area, power

- Can we use domain knowledge to reduce overheads of DNN resilience?

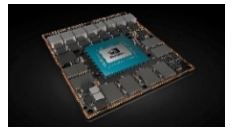
HarDNN: Approach

- Software-directed approach for hardening CNNs for inference



SW | HW

GPU



Embedded

? Future Accelerator

Identify DNN component granularity for analysis

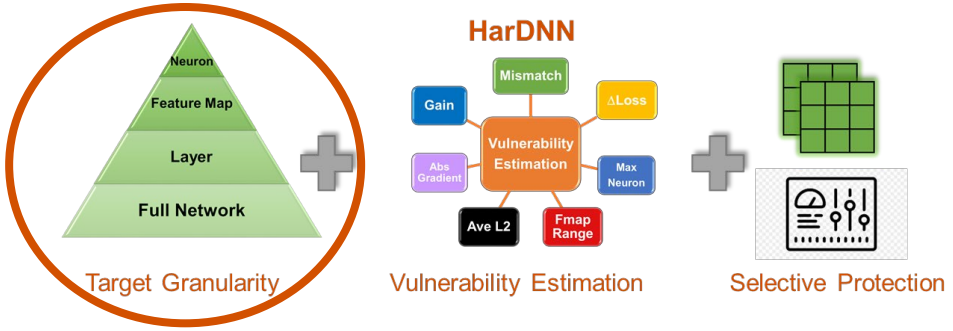
Efficiently estimate DNN component vulnerability

Selectively protect to meet coverage and overhead targets

High, tunable resiliency with low overhead

SARA'20, arXiv'20, DSML'20

HarDNN Challenges



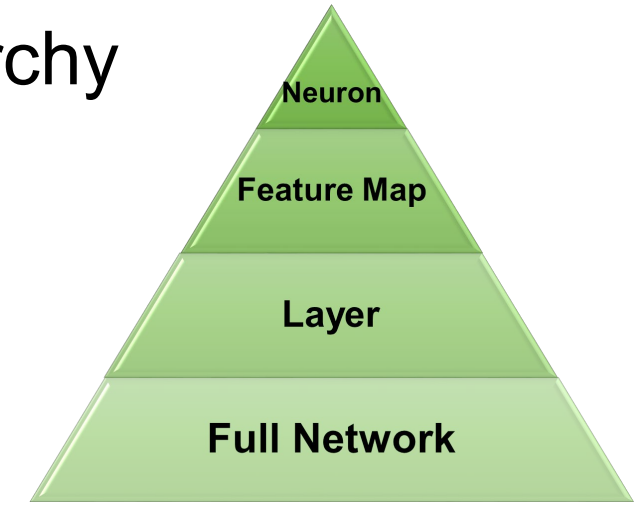
- *What granularity* components to protect?
  - Challenge: Identify granularity for selective protection

- *Which* components to protect?
  - Challenge: Accurately estimate vulnerability of each component

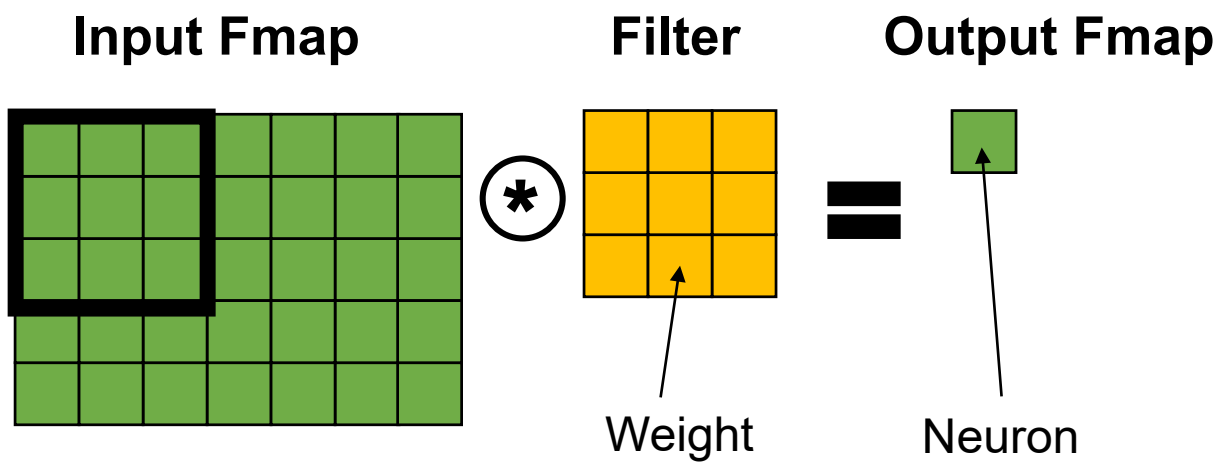
- *How* to protect?
  - Challenge: Low-cost protection mechanism

What Granularity Components to Target?

CNN computation hierarchy



Key computation: Convolution on feature maps

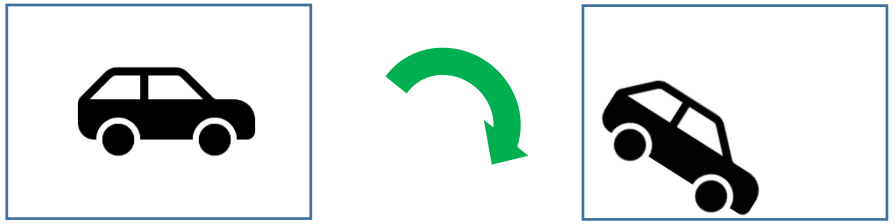


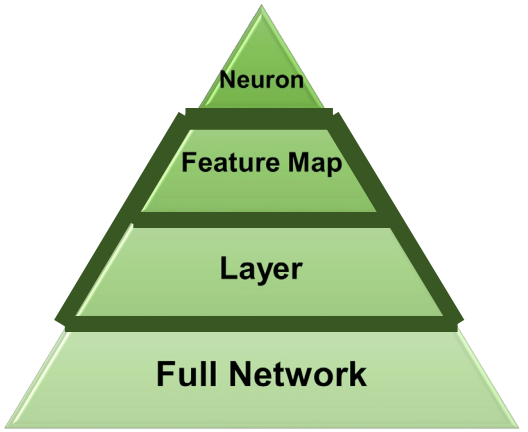
What Granularity Components to Target?

- Full network
  - Rerun inference in its entirety
- Layer
  - Estimate vulnerability of layer, duplicate vulnerable layers by running ltwice
- Feature Map
  - Estimate vulnerability of feature map, duplicate vulnerable fmaps by duplicating filters
- Neuron
  - Estimate vulnerability of neuron, duplicate vulnerable neurons
- Instruction

Feature Map (Fmap) Granularity

- Robustness to translational effects of inputs



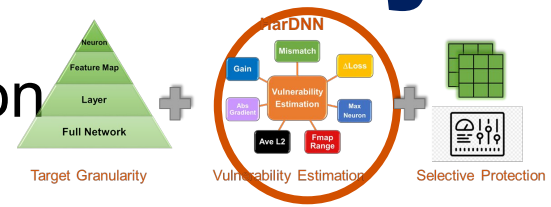


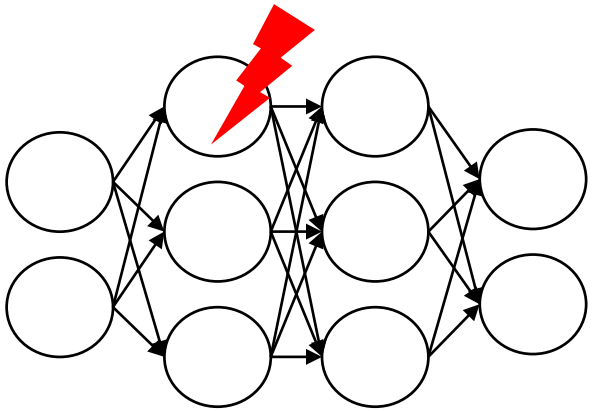
- Granularity “sweet spot”
  - Fine-grained + composable to layers

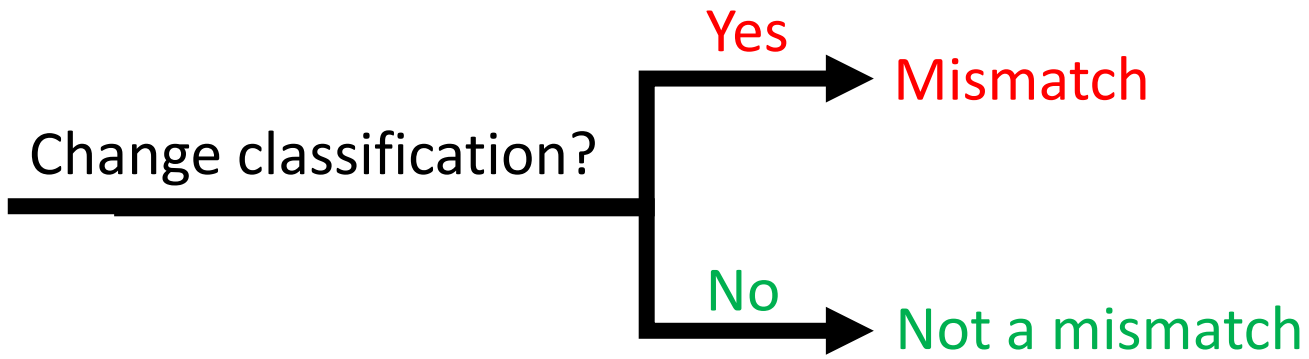
Network-Dataset	Conv Layers	Fmaps
AlexNet-ImageNet	5	1,152
VGG19-ImageNet	16	5,504
SqueezeNet-ImageNet	26	3,944
ShuffleNet-ImageNet	56	8,090
GoogleNet-ImageNet	57	7,280
MobileNet-ImageNet	52	17,056
ResNet50-ImageNet	53	26,560

How to Estimate Feature Map Vulnerability

- P_{prop} = Probability an error in fmap causes a Top-1 misclassification
- Use statistical injection for neurons within feature map







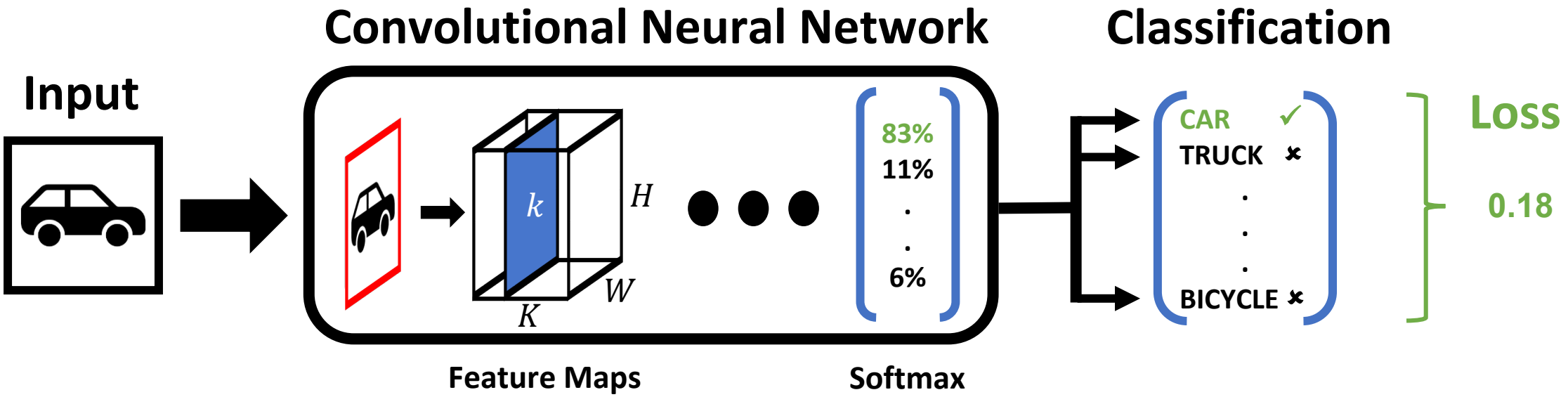
P_{prop} = # Yes / (Total error injections)

- BUT mismatches are relatively rare, takes too many injections to converge
- Insight: Replace binary view of error propagation with continuous view
- Cross-entropy loss: Used to train DNNs to determine/enhance goodness of network

Loss: Continuous Metric for Error Propagation

Insight: Replace binary view of propagation with continuous view

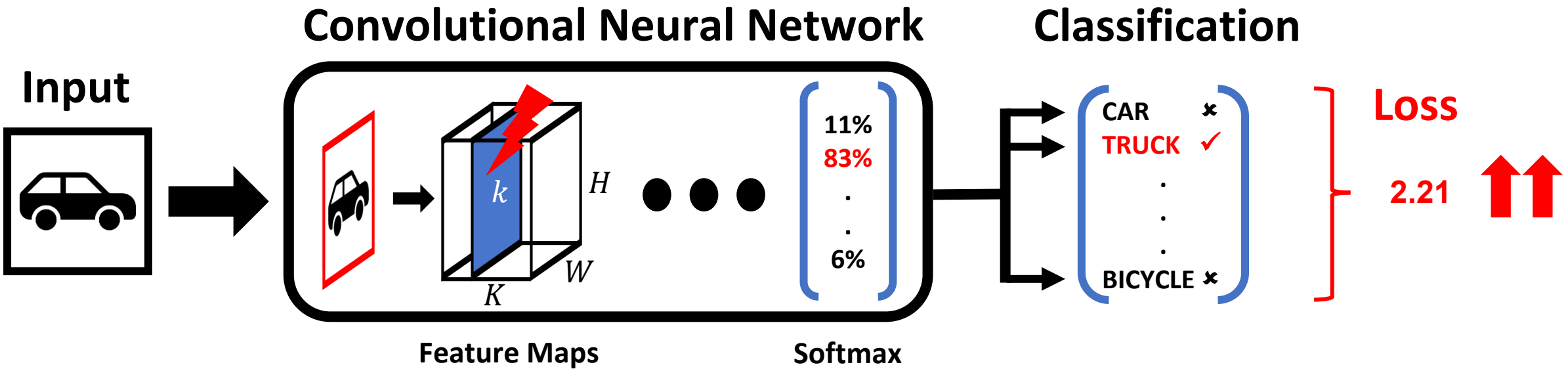
Use cross-entropy loss



Loss: Continuous Metric for Error Propagation

Insight: Replace binary view of propagation with continuous view

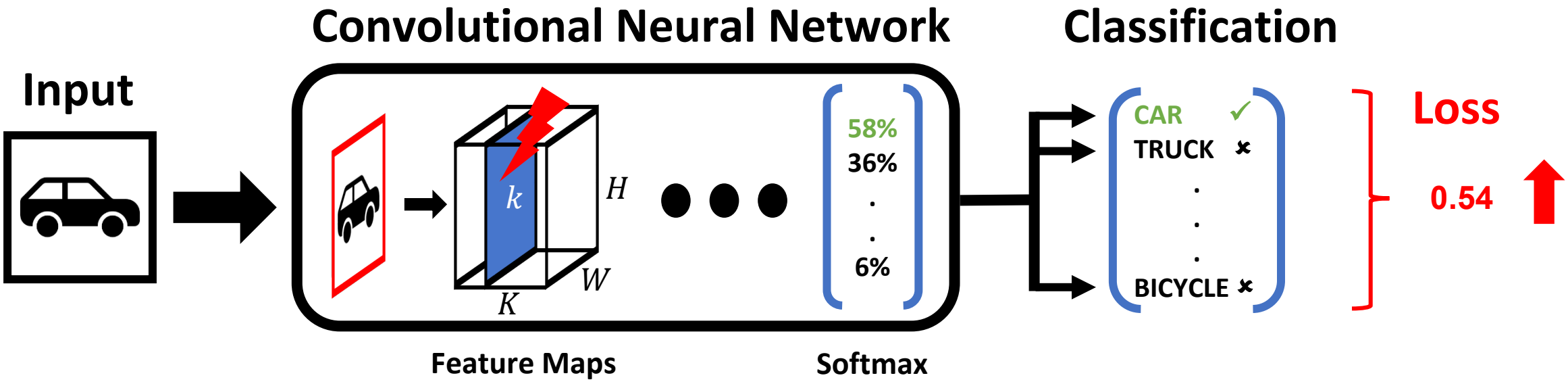
Use cross-entropy loss



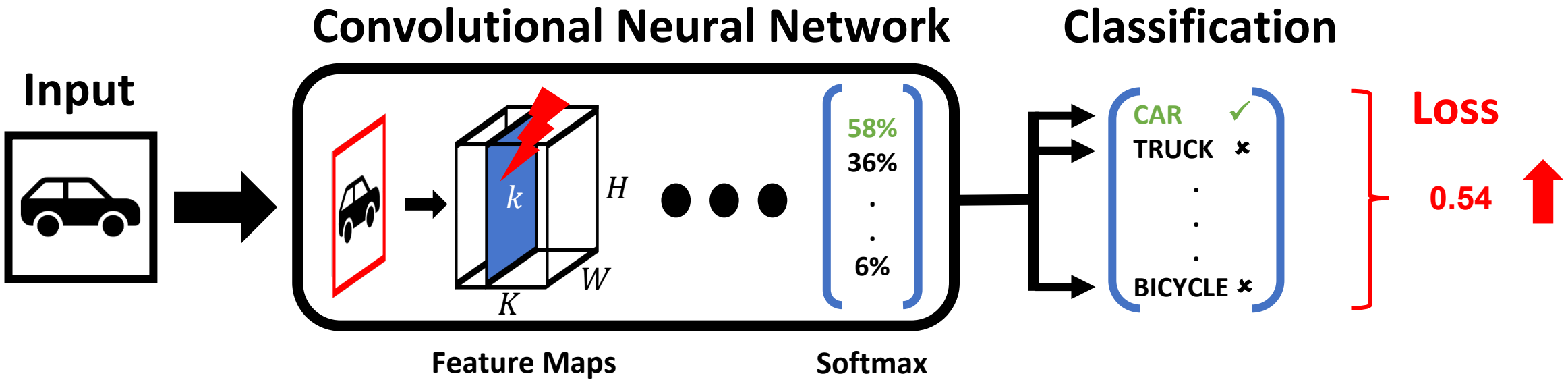
Loss: Continuous Metric for Error Propagation

Insight: Replace binary view of propagation with continuous view

Use cross-entropy loss



Loss: Continuous Metric for Error Propagation

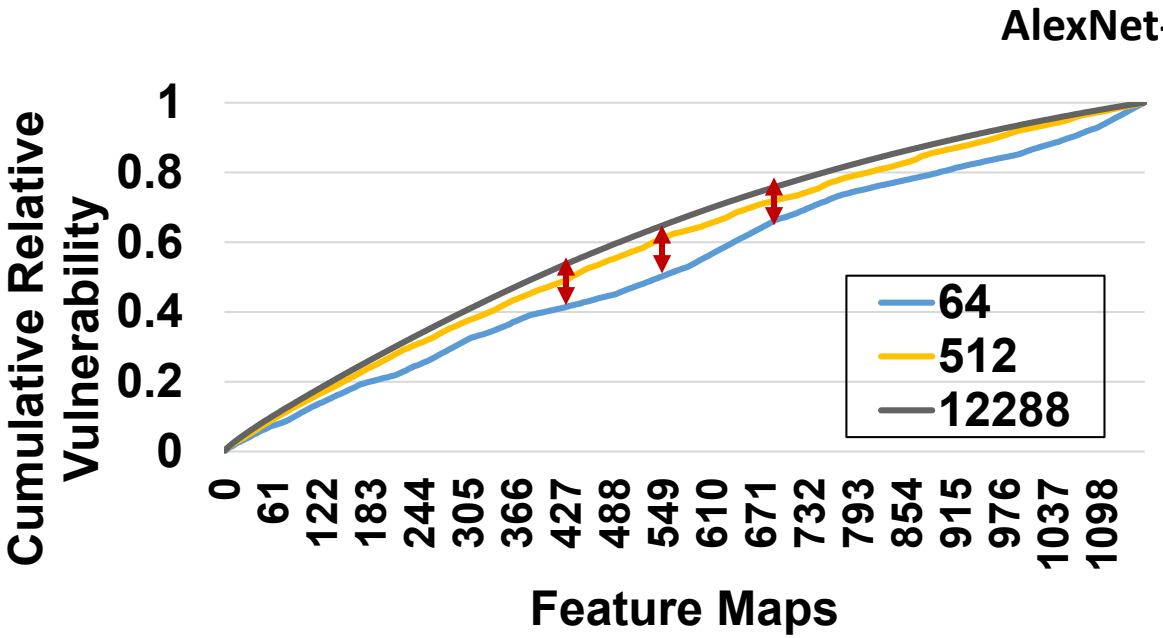


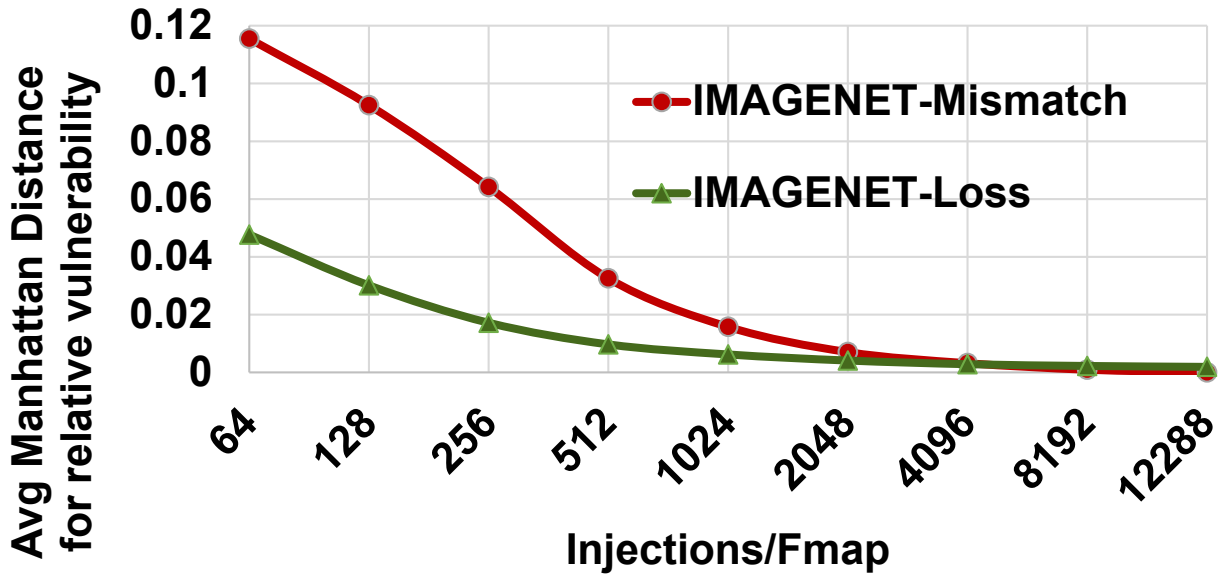
Our metric: average delta cross entropy loss: $\Delta L_{Fmap} = \frac{\sum_i^N |L_{golden} - L_i|}{N}$

P_{prop} for Fmap = $\Delta L_{Fmap} / \sum_i^N \Delta L_{Fmap}$

Mismatch vs. Loss: Which Converges Faster?

- How many injections per feature map? Sweep from 64 to 12,288
  - Use Manhattan distance from 12,288 injections to quantify “similarity” of vulnerability estimates

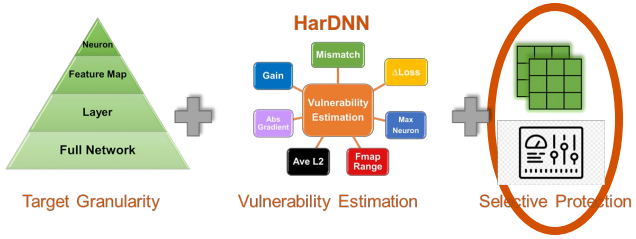




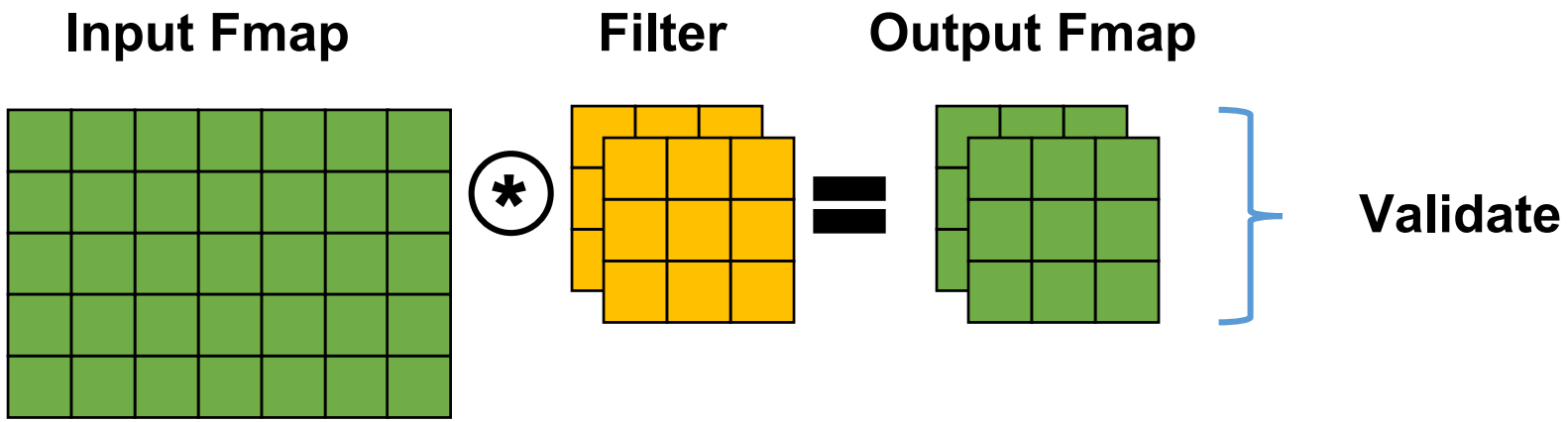
Mismatch and Loss vulnerability estimates converge with increasing injections

Loss converges faster

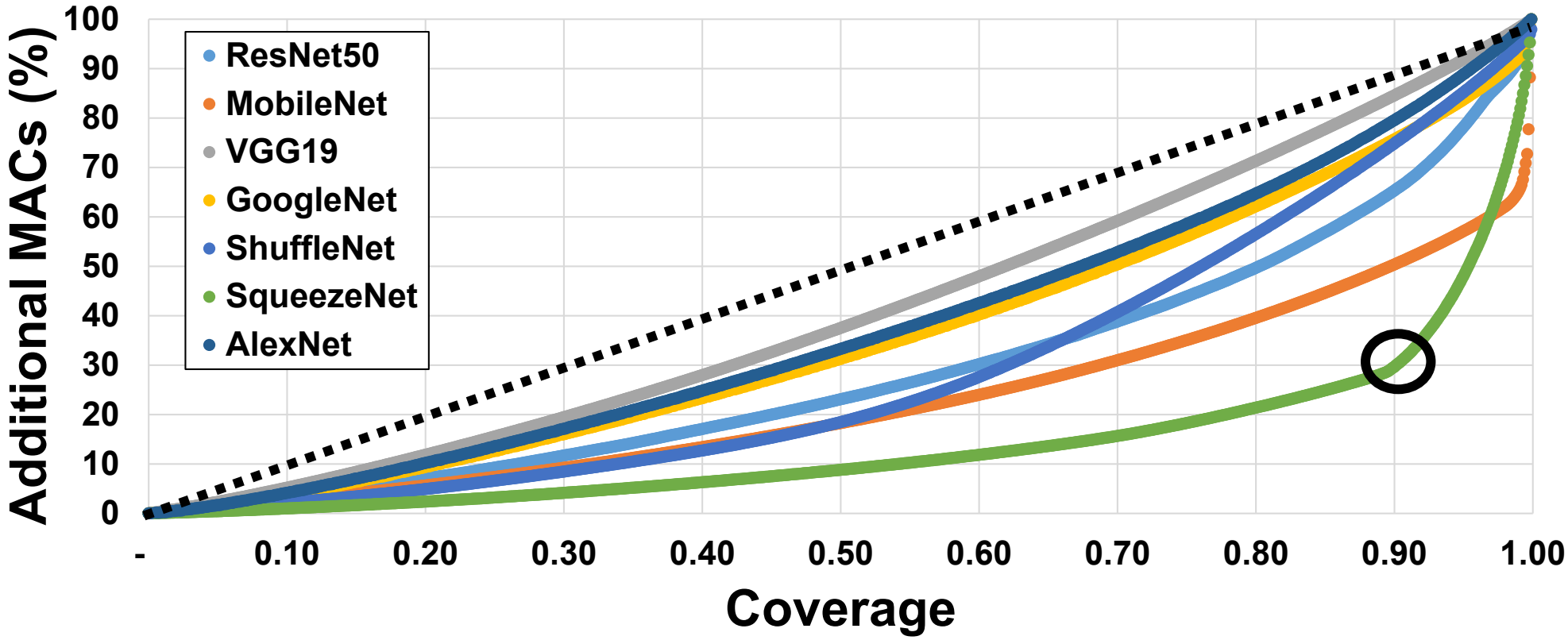
How to Protect?



- Objective: Duplicate computations (MACs) of vulnerable feature maps
- Duplication Strategy: Filter Duplication
  - Software directed approach: portable across different HW backends
  - Duplicates the corresponding *filter* to recompute *output fmap*
  - Validate computations off the critical path



Overhead vs. Coverage



Overhead (MACs) sub-linear to coverage

SqueezeNet: 10X reduction in errors for 30% additional computation

Next step: combination with other granularities, prune injection space

Outline

- Software-centric error analysis and error efficiency: Approxilyzer, Winnow
- Software testing for hardware errors: Minotaur
- Domain-specific error efficiency: HarDNN
- Compiler and runtime for hardware and software error efficiency: ApproxTuner
- Putting it Together: Towards a Discipline for Error-Efficient Systems

ApproxTuner: Hardware + Software Approx

- Unified compiler+runtime framework for software and hardware approximations

- Goal:
  For each operation in the application
  – select hardware and/or software approximation with
  – acceptable end-to-end accuracy and maximum speedup (minimum energy)

- Currently for applications with tensor operations; e.g., DNNs

- Example approximations studied
  – Software: Perforated convolutions, filter sampling, reduction sampling
  – Hardware: lower precision, PROMISE analog accelerator [ISCA18]

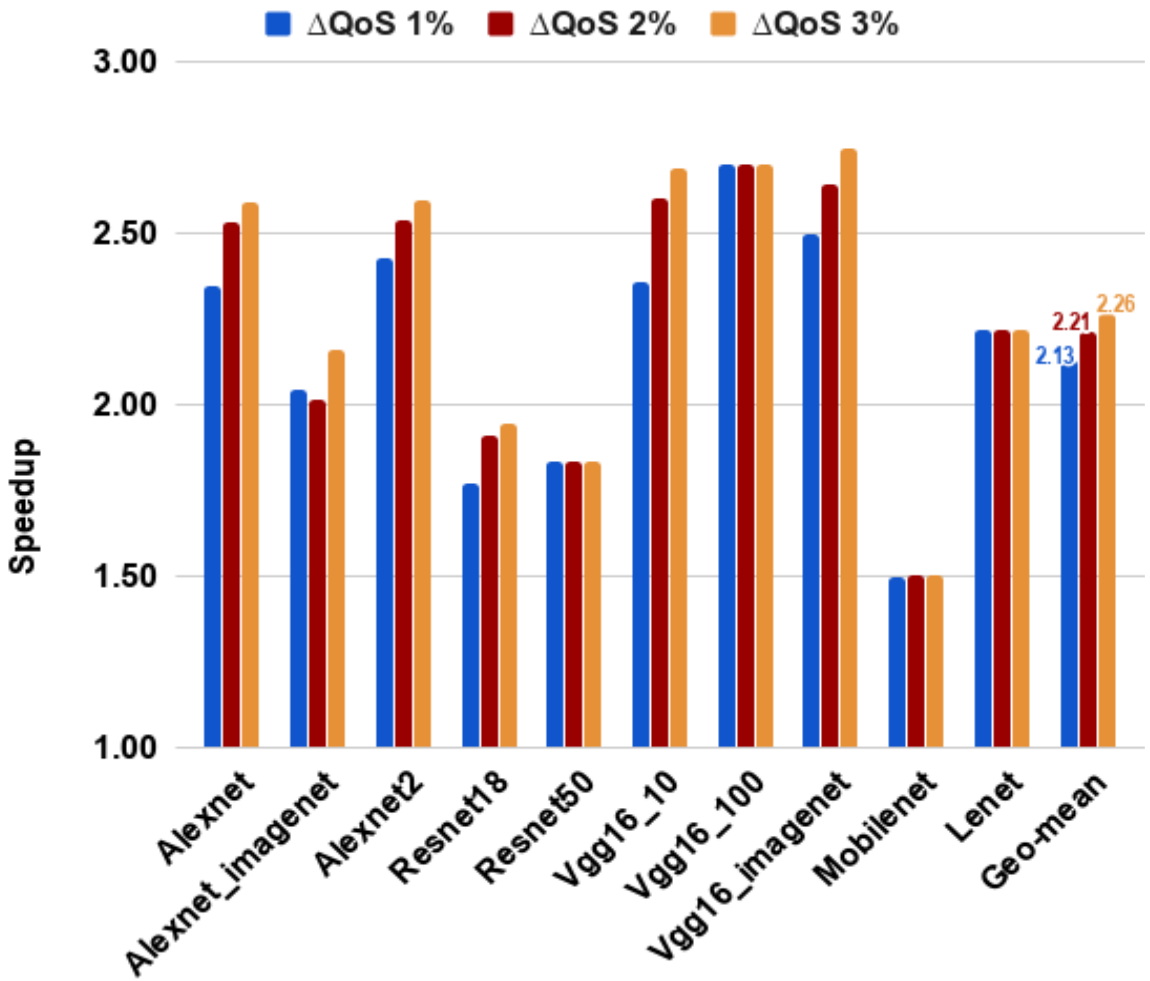
OOPSLA'19, in review

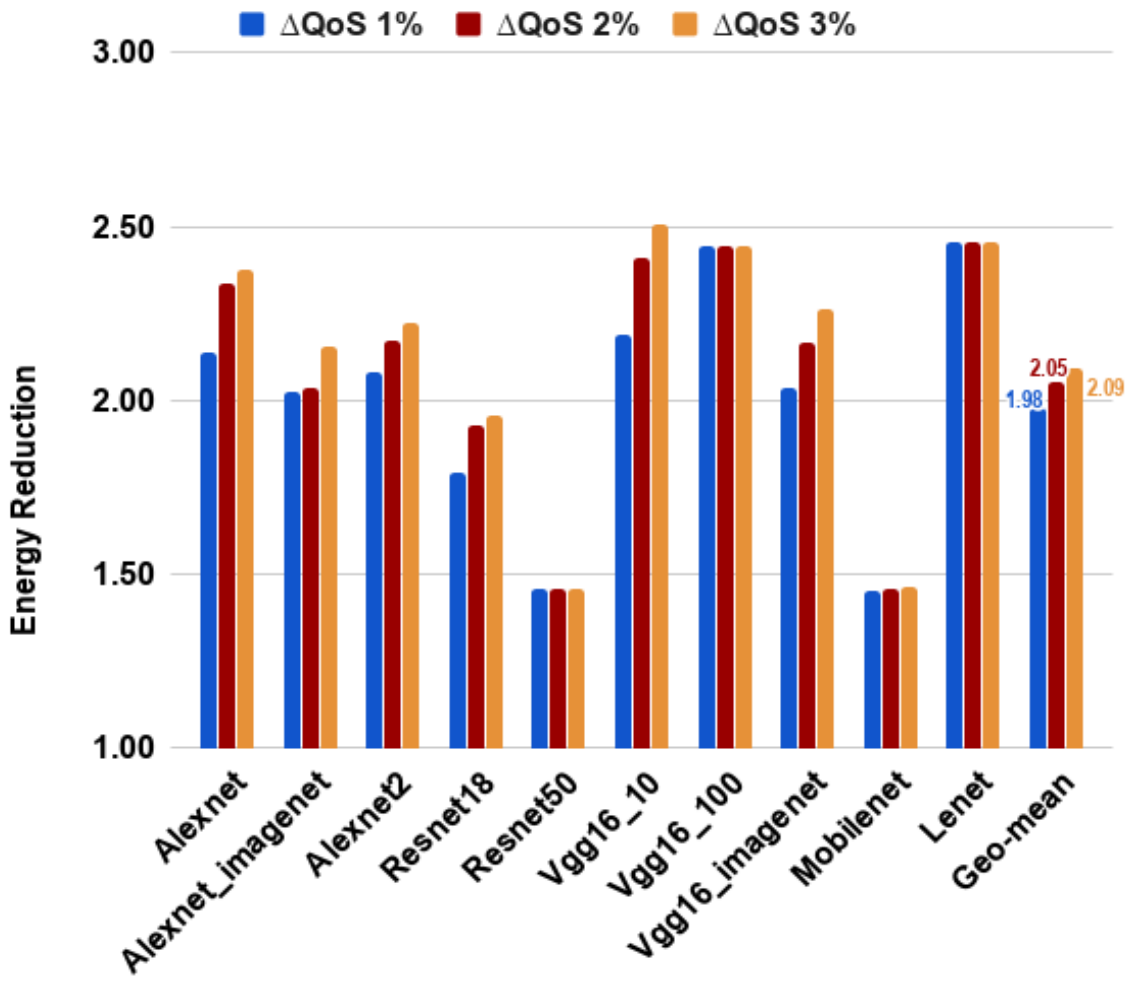
ApproxTuner Innovations

- Combines multiple software and hardware approximations
- Uses predictive models to compose accuracy impact of multiple approximations
- 3-phase approximation tuning
  - Development-time preserves hardware portability via ApproxHPVM IR
  - Install-time allows hardware-specific approximations
  - Run-time allows dynamic approximation tuning
- Federated Tuning for efficiency at install-time
  - Install-time tuning is expensive under resource constraints

GPU Speedup and Energy Reduction

Approximations: Sampling, Perforation, FP16

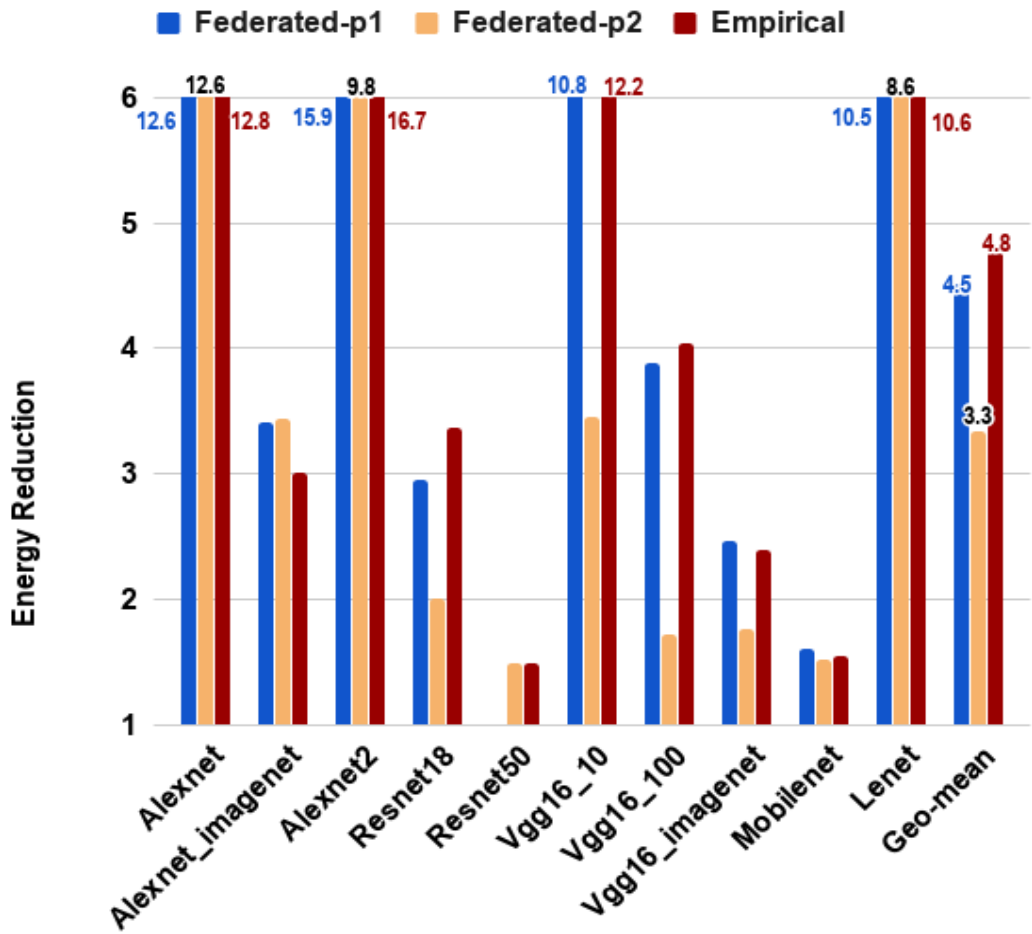




2.1x mean speedup and 2x mean energy reduction with 1% QoS loss

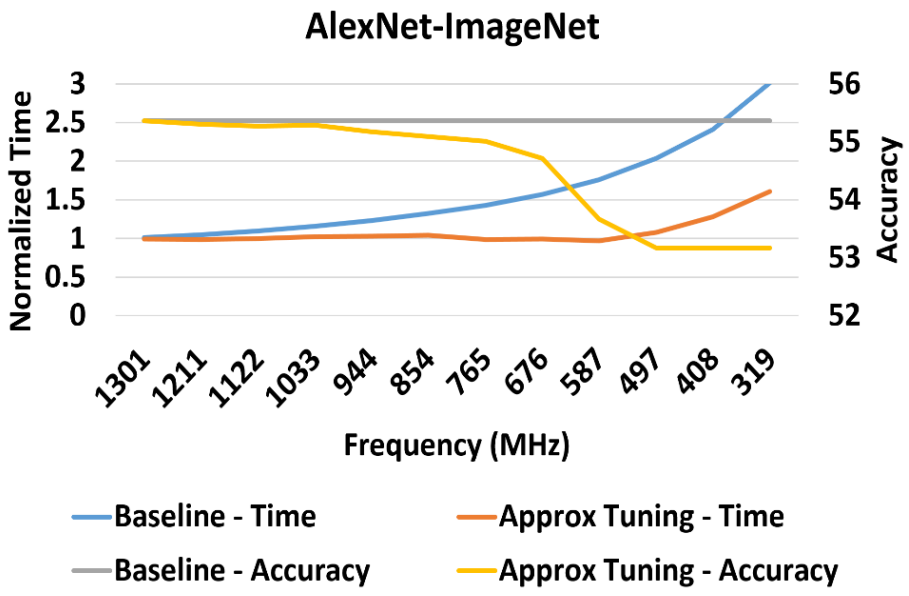
Federated vs Empirical: Energy Reduction

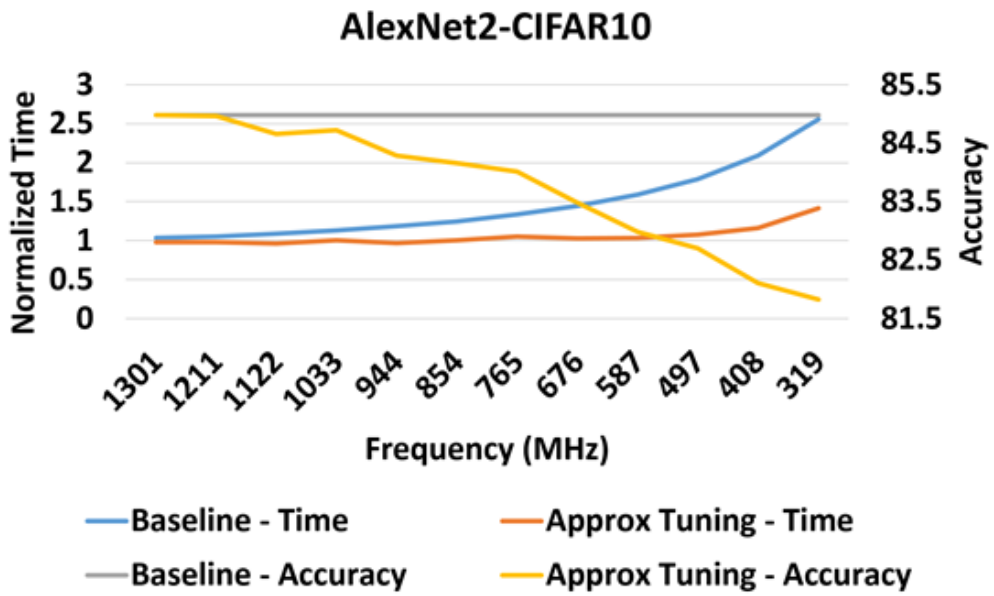
Approximations: PROMISE accelerator, Sampling, Perforation, FP16



Federated-p1 gives 4.5x energy reduction, comparable to empirical tuning

Runtime Approximation Tuning



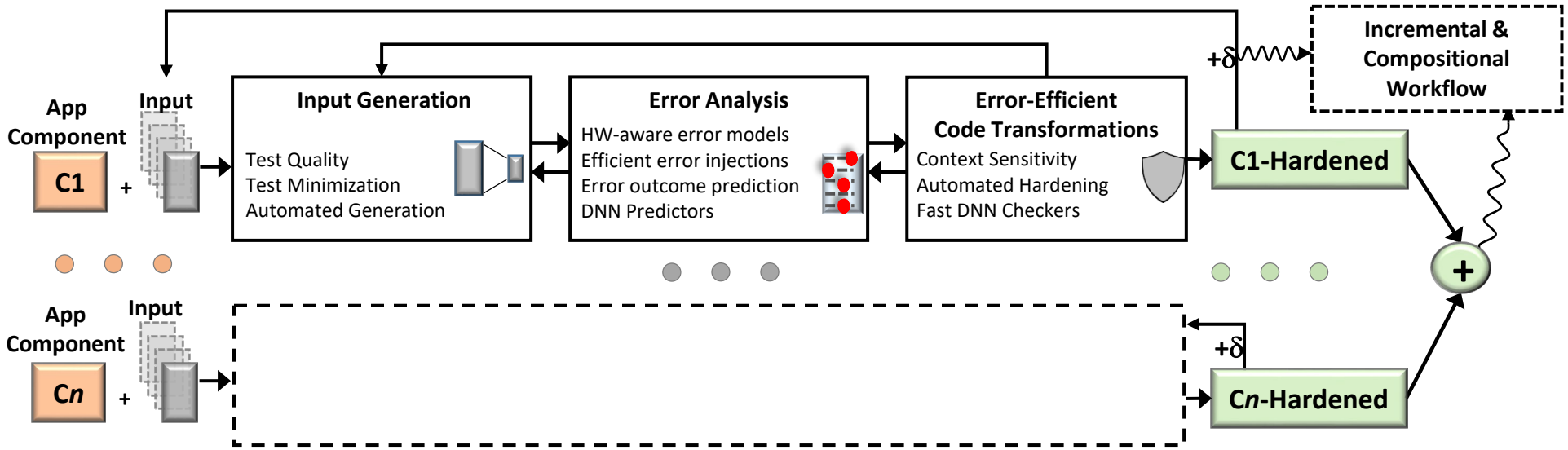


Runtime tuning helps maintain responsiveness in face of frequency changes

Outline

- Software-centric error analysis and error efficiency: Approxilyzer, Winnow
- Software testing for hardware errors: Minotaur
- Domain-specific error efficiency: HarDNN
- Compiler and runtime for hardware and software error efficiency: ApproxTuner
- Putting it Together: Towards a Discipline for Error-Efficient Systems

Towards a Discipline for Error-Efficient Systems

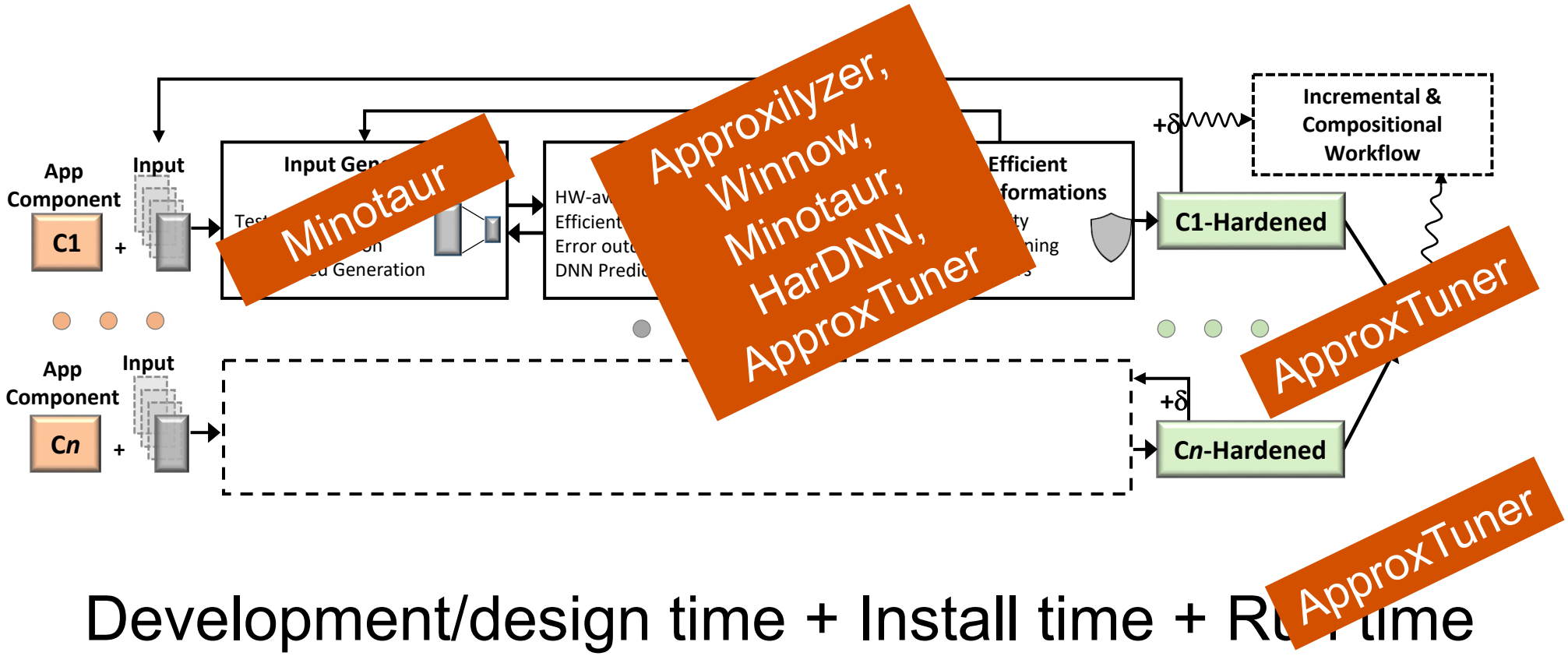


Development/design time + Install time + Run time

End of Moore's law and Dennard scaling motivate error efficient systems

- Integrate hardware errors in software engineering workflow
- Integrate hardware and software error optimization for error efficient system workflows

Towards a Discipline for Error-Efficient Systems



Development/design time + Install time + Run time

End of Moore's law and Dennard scaling motivate error efficient systems

- Integrate hardware errors in software engineering workflow
- Integrate hardware and software error optimization for error efficient system workflows