LLFI: An Intermediate Code Level Fault Injector For Soft Computing Applications

Anna Thomas and Karthik Pattabiraman
University of British Columbia (UBC)

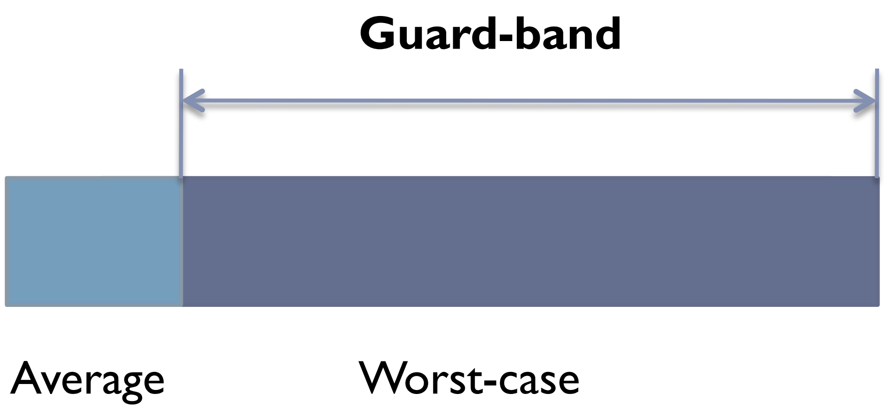
Transient Errors: Traditional "Solutions"

- **Guard-banding**

  Widening gap between average and worst-case due to variations

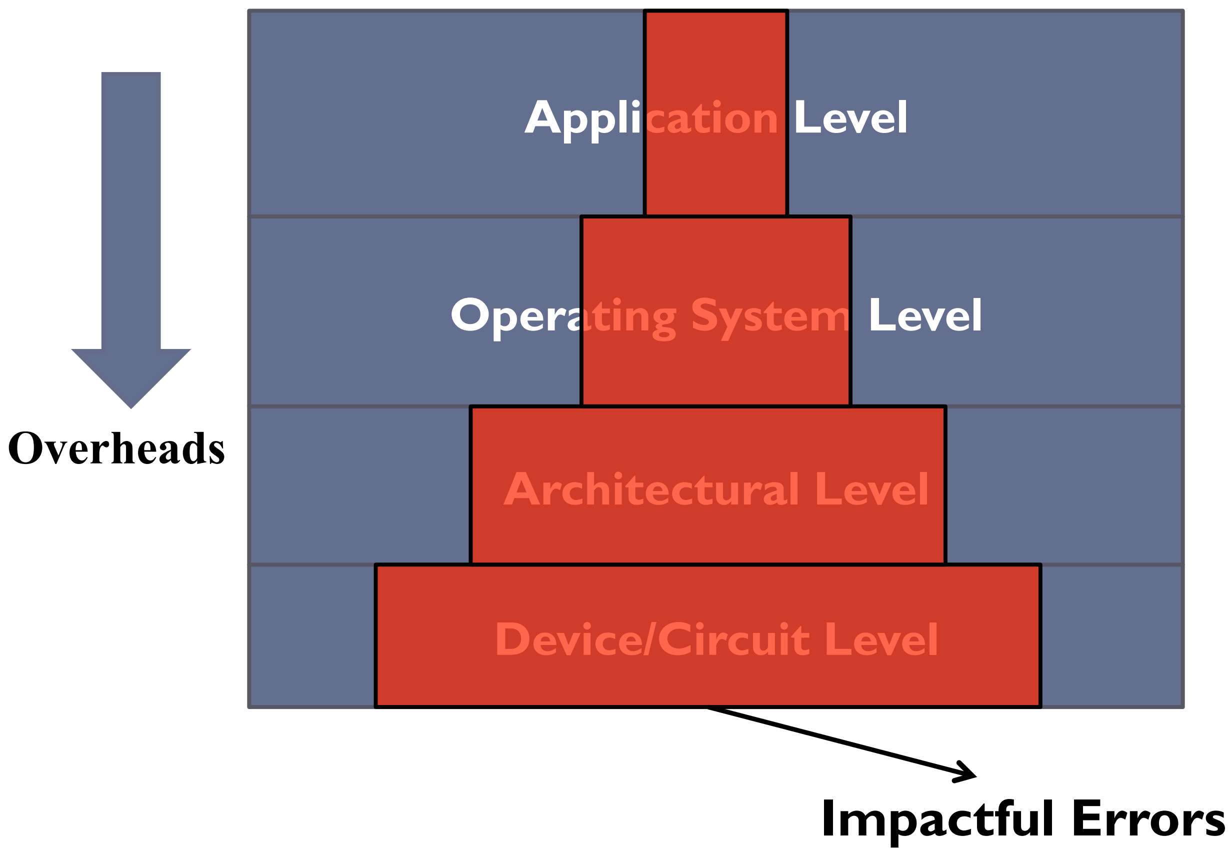
- **Duplication**

  Hardware duplication (DMR) can result in 2X slowdown



High Power and Performance Overheads

Why Software Solutions?



Soft Computing Applications

- Applications in AI, multimedia processing
- Examples in RMS Workloads [Dubey'07]
- Tolerate many kinds of faults in data and code

Original image (left) versus faulty image from JPEG decoder

Egregious Data Corruptions

- Large or unacceptable deviation in output
- Based on fidelity metric (e.g., PSNR)

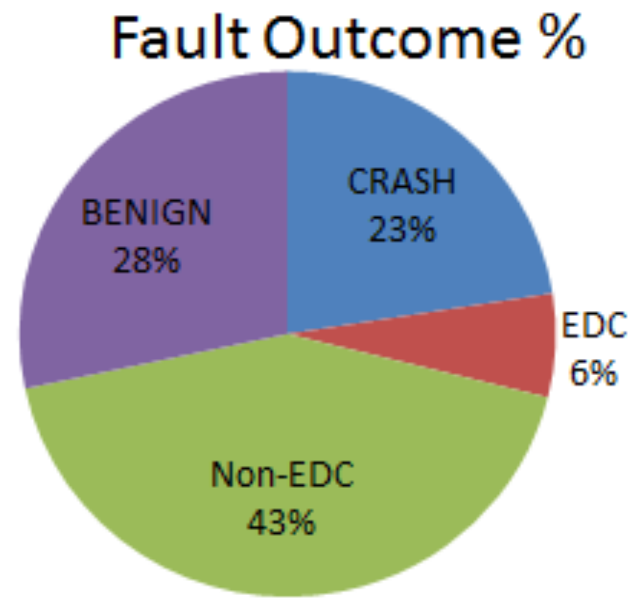
EDC image (PSNR of 11.37) of JPEG vs Non-EDC image (PSNR of 44.79)

Why detect EDC Causing Faults?

- Unacceptable outcome to the end user

- Why not detect all faults?

  92% of faults that do not crash the application result in tolerable outcomes



Goal

- End Goal: Detect EDC causing faults
  - Pre-emptively – to avoid unacceptable outputs

  and

  - Selectively – to avoid wasteful recovery
- Identify Source Level characteristics of EDC causing faults

This Talk

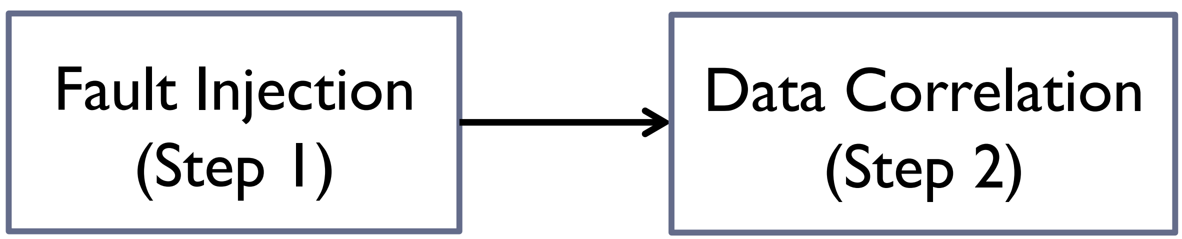
- End Goal: Detect EDC causing faults
  - Pre-emptively – to avoid unacceptable outputs

  and
  - Selectively – to avoid wasteful recovery
- Identify Source Level characteristics of EDC causing faults

Approach

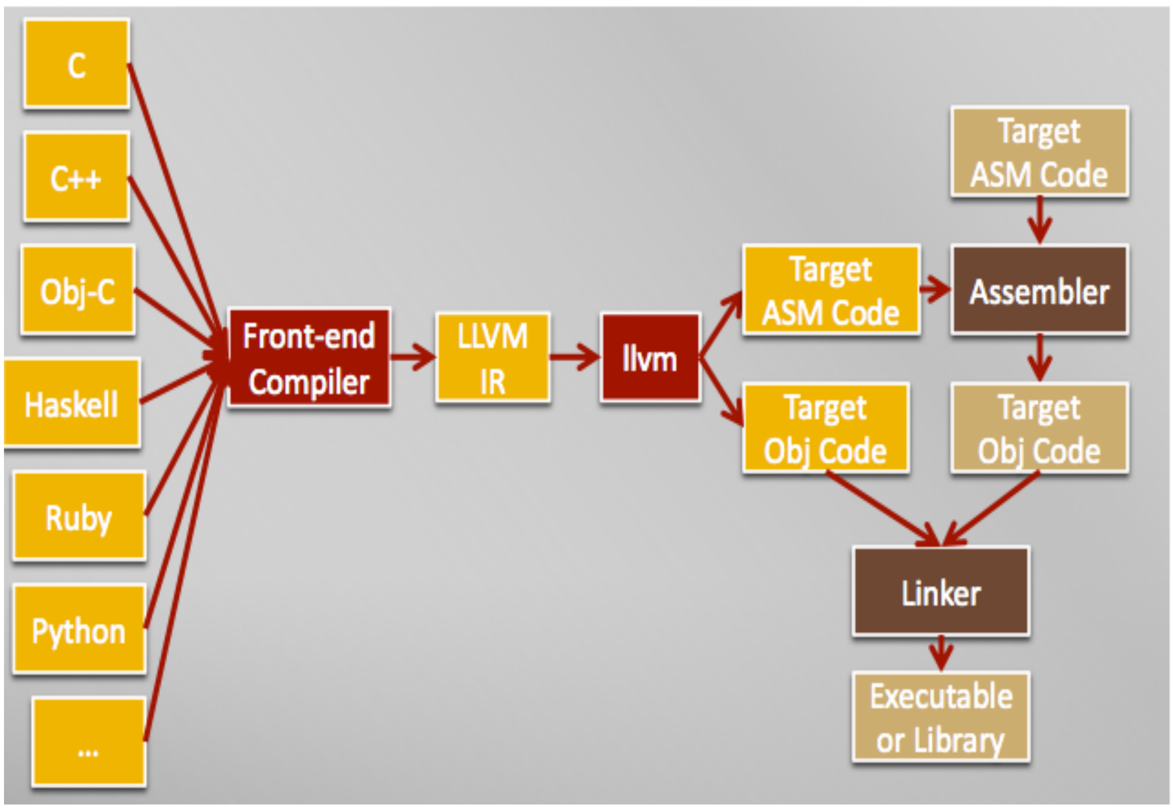
- Step 1: Perform fault injections to separate EDCs from Non-EDCs

- Step 2: Identify correlation between data categories (eg: pointers) and fault outcomes



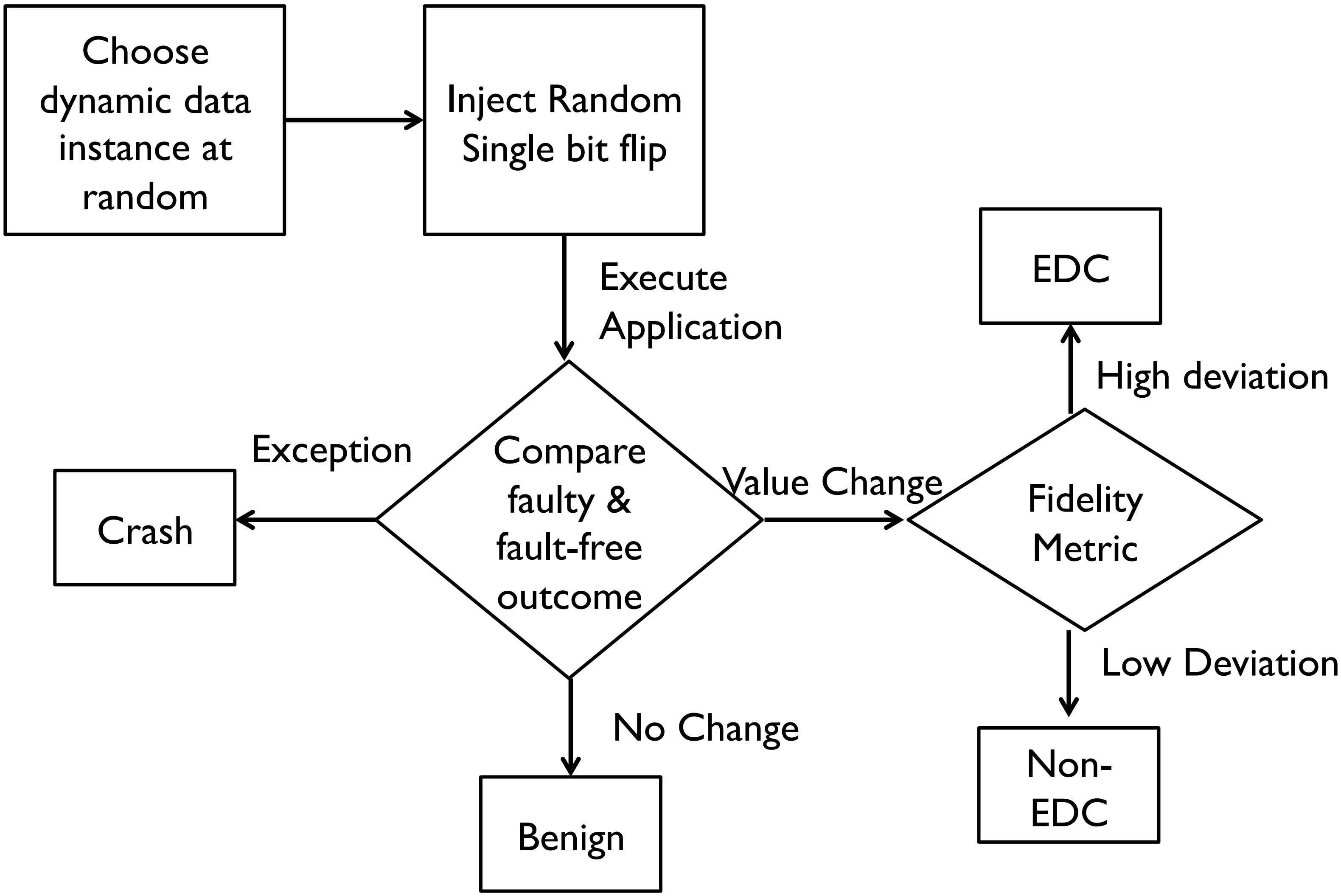
Step 1: LLVM Fault Injector LLFI

Fault Injector at LLVM compiler's intermediate code level (widely used compiler framework) [Lattner'05]



- Easy source code mapping
- Program analysis and transformation support
- Robust Lowering

LLFI Framework



Step 2: Data Categorization

- Trace backward slice of Control and Pointer Data

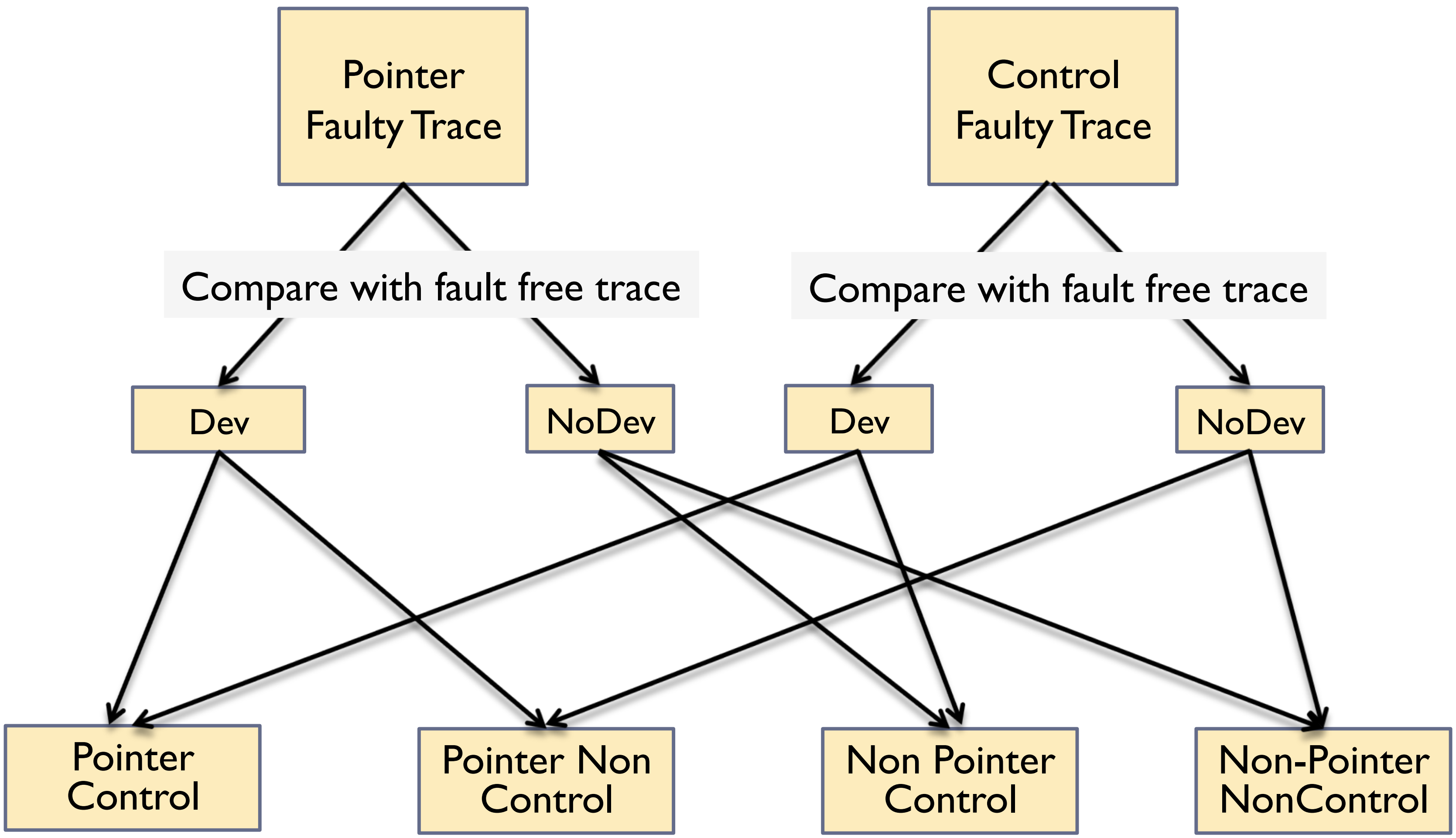
```
int main() {
    int fact, i, n;
    n = atoi (argv[1]);
    fact = 1;

   for( i = 1 ; i <= n; i++ )
      fact = fact * i;

    print fact;
}
```

Control Data

Trace this value

Control Faulty Trace	Control Fault Free Trace
Pointer Faulty Trace	Pointer Fault Free Trace

Step 2: Data Categorization



Data Categorization: MPEG Decoder

```
void conv422to444 (char *src, char *dst,
int offset, int width) {
    if(dst < src + offset)
        return;
    for(int i=0; i < width; i++) {
        im1 = (i < 1) ? 0 : i – 1
            …
        dst[im1] = Clip [(21*src[im1])>>8];
    }
    …
}
```

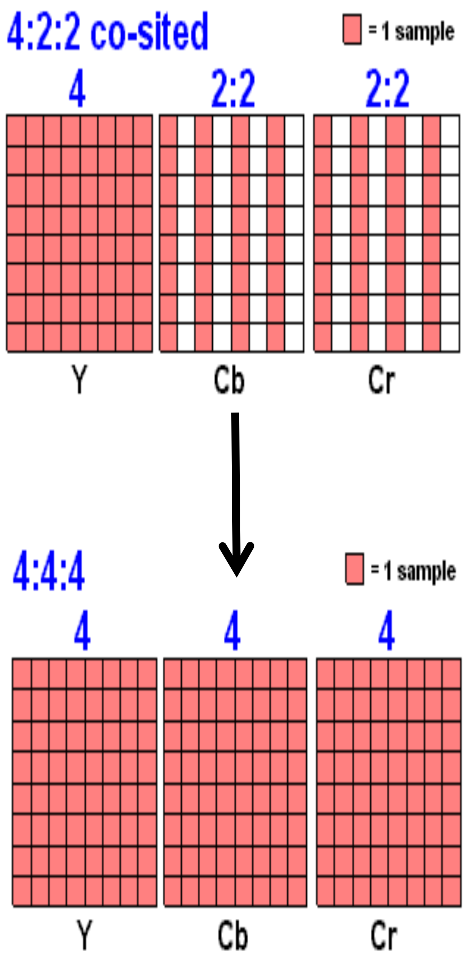


Image Source: Computer Desktop Encyclopedia c 2004, The Computer Language Co. Inc.

Data Categorization: Control Pointer

```
void conv422to444 (char *src, char *dst,  int
offset, int width) {
    if(dst < src + offset)
        return;
    for(int i=0; i < width; i++) {
        im1 = (i < 1) ? 0 : i – 1
        …
        dst[im1] = Clip [(21*src[im1])>>8];
    }
    …
}
```

Control Pointer

- Fault in low bit of src or dst

Data Categorization:Control Non-Pointer

```
void conv422to444 (char *src, char *dst,  int offset, int width) {
    if(dst < src + offset)
        return;
    for(int i=0; i < width; i++) {
        im1 = (i < 1) ? 0 : i – 1
            …
        dst[im1] = Clip [(21*src[im1])>>8];
    }
    …
}
```

Control Non-Pointer

- Fault in i
- Branch Flip

Data Categorization:Pointer Non-Control

```
void conv422to444 (char *src, char *dst,  int
offset, int width) {
    if(dst < src + offset)
        return;
    for(int i=0; i < width; i++) {
        im1 = (i < 1) ? 0 : i – 1
            …
        dst[im1] = Clip [(21*src[im1])>>8];
    }
    …
}
```

Pointer Non-Control

- Fault in index of Clip

Experimental Setup

- Six Benchmarks from MediaBench Suite
  - Video, Image and Speech Decoders
  - Fidelity Metric: PSNR, Segmental SNR

- Performed fault injections using LLFI
  - 1000 fault injections, 1 fault per run (2.2% at 90% CI)
  - All injected instructions are executed

- Identified Correlation between faults in pointer/ control data and EDC outcomes

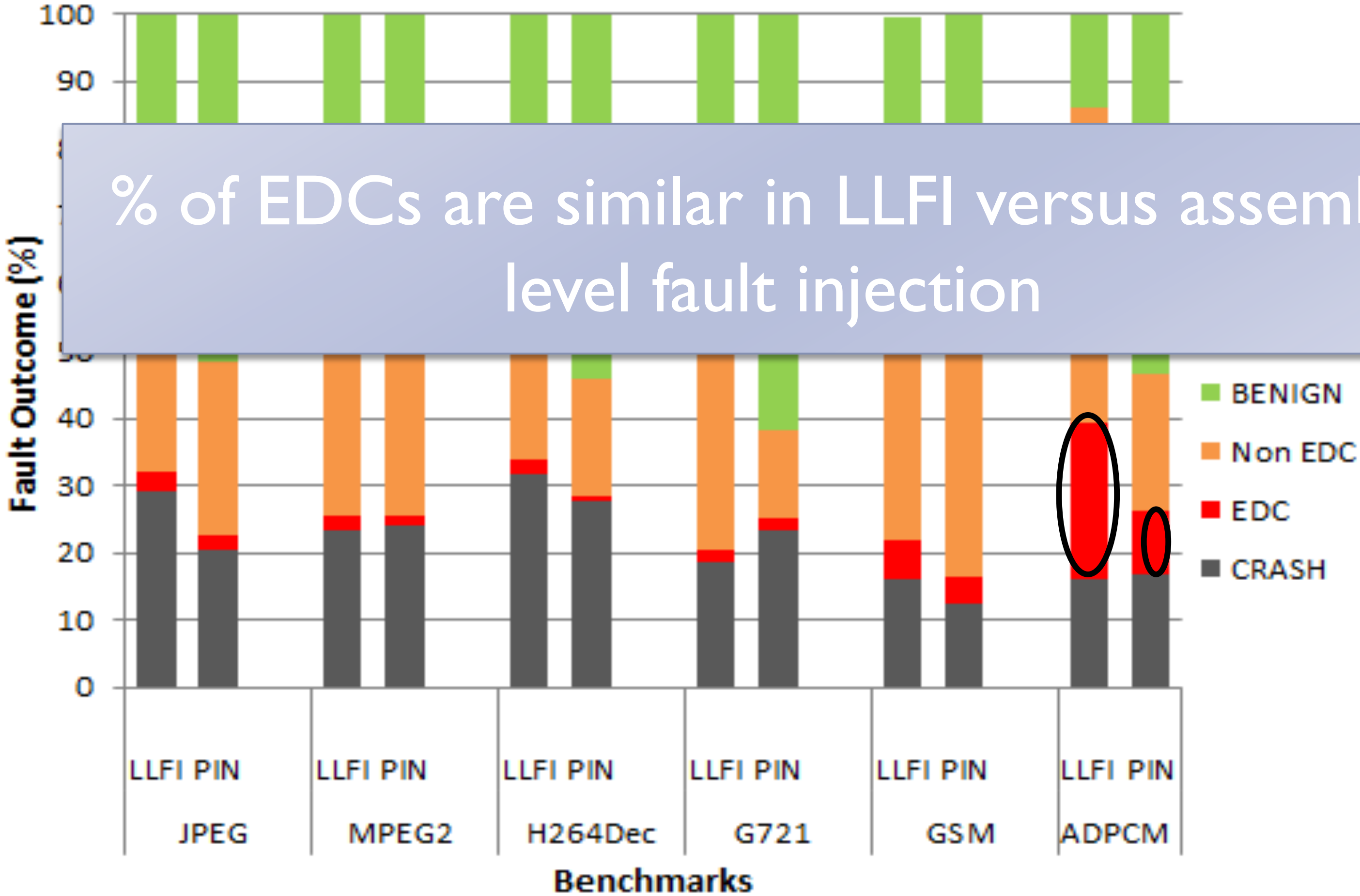
LLFI Accuracy

- Differences introduced by translation from Intermediate Representation to assembly

- Quantified difference between LLFI versus Assembly fault injector built using PIN
  - PIN: binary rewriting tool from Intel

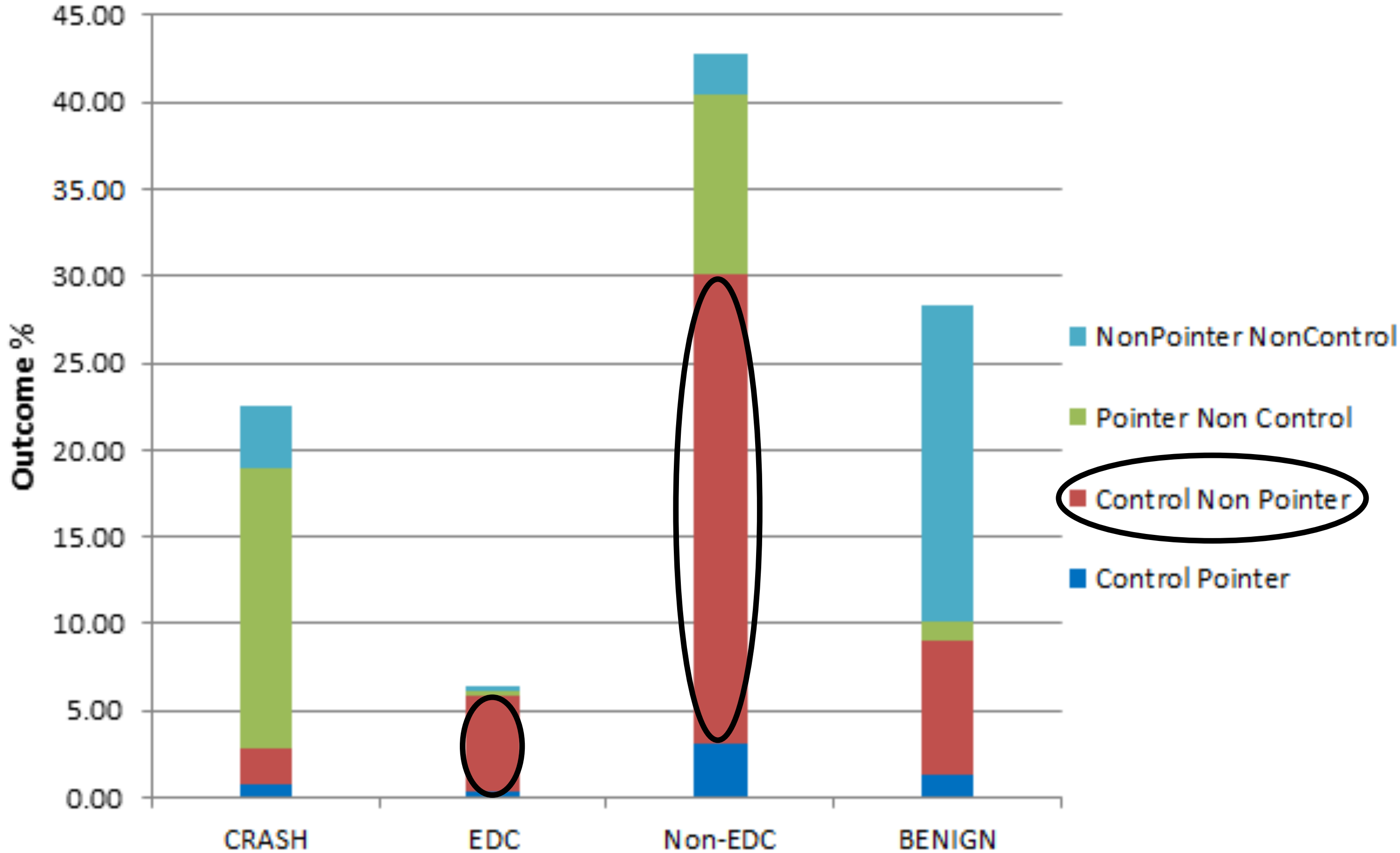
LLFI Accuracy

Same number of fault injections and fidelity threshold values



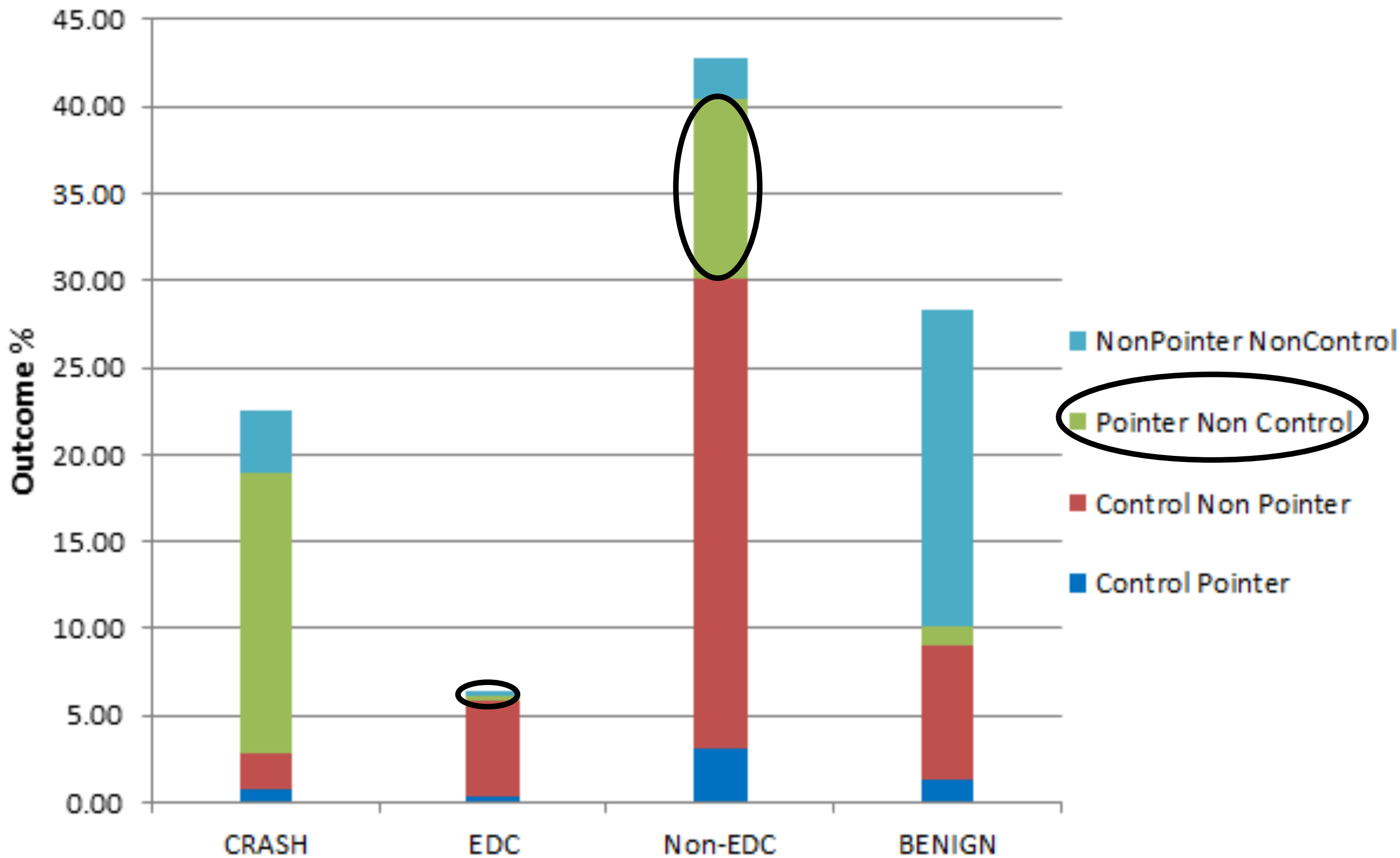
% of EDCs are similar in LLFI versus assembly level fault injection

Data Categorization of Fault Outcomes



High correlation between Control Non-Pointer and EDC/Non-EDC

Data Categorization of Fault Outcomes



Pointer Non Control: Faults in low Order bits caused EDC/ Non-EDC

Conclusion

LLFI: Intermediate Code Level Fault Injector

- Identify source level characteristics of EDC faults
- Validated accuracy of LLFI versus assembly level injection
- Correlation between EDC faults and data categories

Current Work (To Appear in DSN'13)

- Identified heuristics based on data correlation

LLFI: https://github.com/DependableSystemsLab/LLFI

Contact: annat@ece.ubc.ca

Qualitative Difference

- 0.4% of total injected faults affect IP register and cause EDCs

- All faults affecting SP register cause Crashes

- Test instructions in branch conditions affect RFLAGS register – high number of benign outcomes

Factorial IR

```
define i32 @main(i32 %argc, i8** %argv) nounwind {
entry:
  %"alloca point" = bitcast i32 0 to i32
  %0 = getelementptr inbounds i8** %argv, i64 1
  %1 = load i8** %0, align 1
  %2 = call i32 (...)* @atoi(i8* %1) nounwind
  br label %bb1

bb:                                                   ; preds = %bb1
  %3 = mul nsw i32 %fact.0, %i.0
  %4 = add nsw i32 %i.0, 1
  br label %bb1

bb1:                                                  ; preds = %bb, %entry
  %i.0 = phi i32 [ 1, %entry ], [ %4, %bb ]
  %fact.0 = phi i32 [ 1, %entry ], [ %3, %bb ]
  %5 = icmp sle i32 %i.0, %2
  br i1 %5, label %bb, label %bb2

bb2:                                                  ; preds = %bb1
  %6 = call i32 (i8*, ...)* @printf(i8* noalias getelementptr
       inbounds ([4 x i8]* @.str, i64 0, i64 0), i32 %fact.0) nounwind
  br label %return

return:                                               ; preds = %bb2
  ret i32 undef
}
```