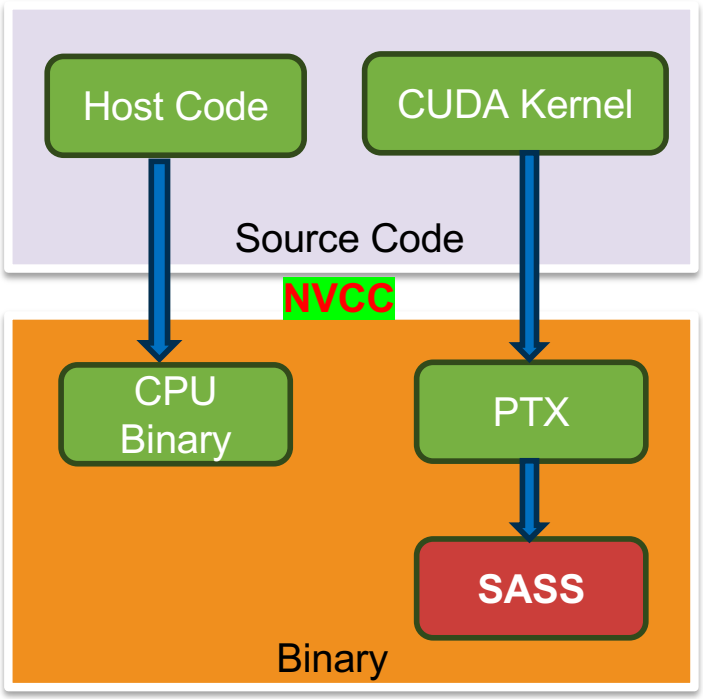
NVLift: Lifting NVIDIA GPU Assembly to LLVM IR for Downstream Security Applications

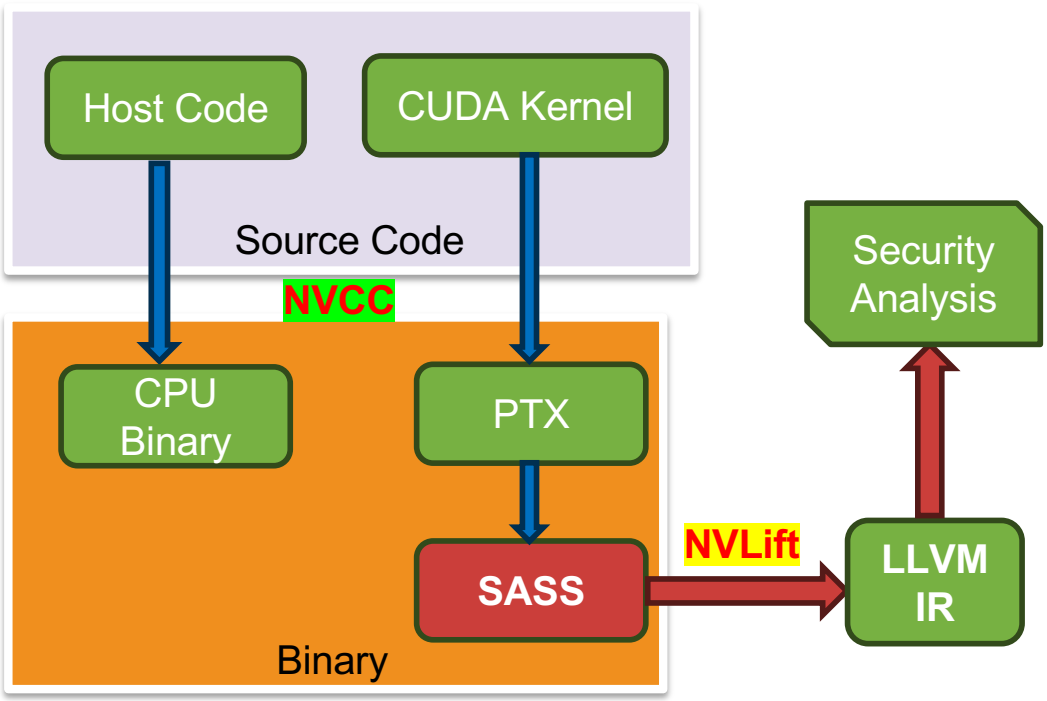
Junpeng Wan, Louis Zheng-Hua Tan, Dave (Jing) Tian

Background



- SASS (Streaming Assembly) under-documented
- Difficult for GPU binary analysis
- NVLift!

Overview



1. Understand SASS instructions
2. NVLift: SASS to LLVM IR
3. Case Study: decompilation

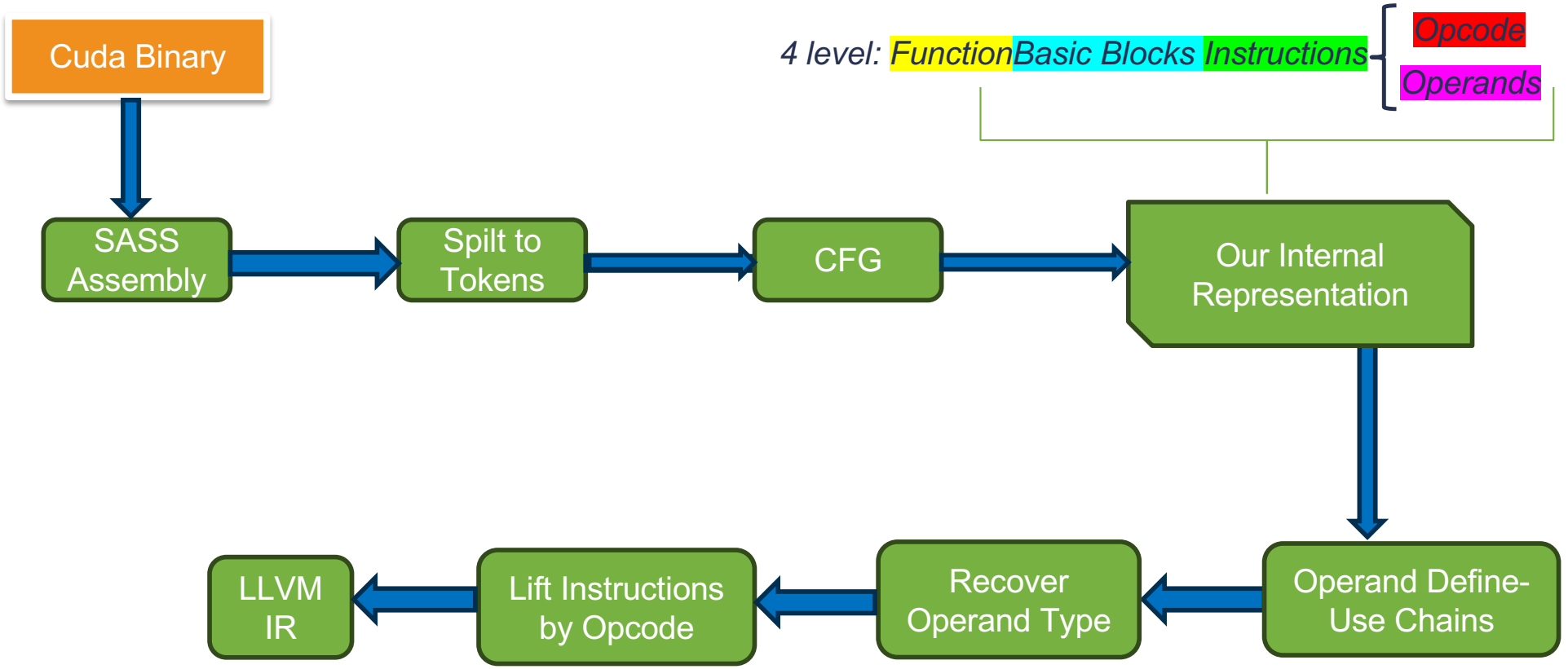
Challenges

- Lack SASS official document
  - Only a sketchy SASS documentation from NVIDIA
- Lack type information in SASS instructions
  - e.g. IMAD R2, R2, R3, R4; MOV R2, R6
- Checking the correctness of lifted LLVM IR

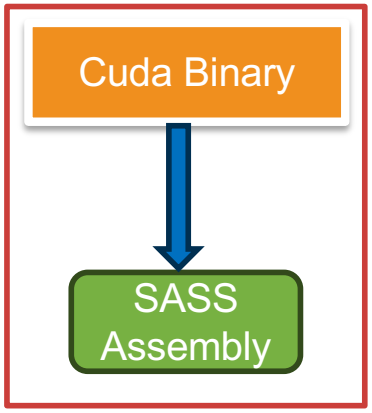
Understanding the SASS Instructions

- Compare Opcode with similar PTX instructions
- Previous reverse engineering work
  - Function/Format/Usage
- Verify instructions behaviors via cuda-gdb
  - e.g. IMAD R2, R2, R3, R4

Design - Overview



Design – Extract SASS

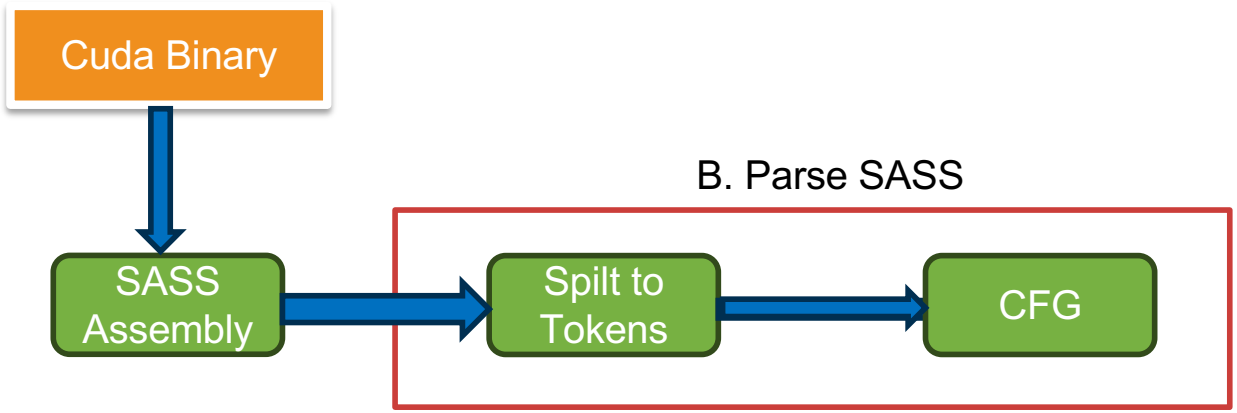


A. Extract SASS

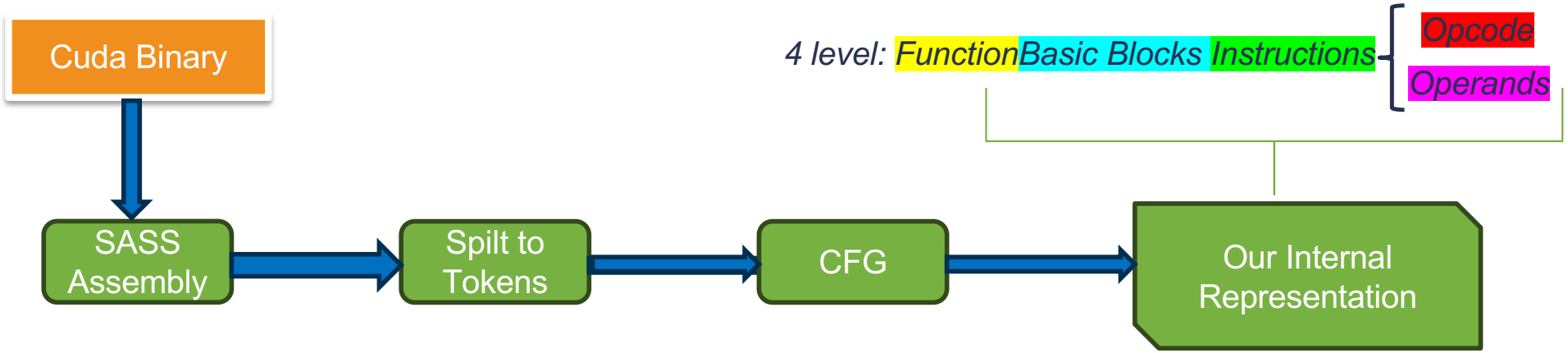
A SASS demo for *RELU* operator in *SM75*

```
        .section        .text._Z4reluPfS_i,"ax",@progbits
        .sectioninfo    @"SHI_REGISTERS=10"
        .align  128
        .global         _Z4reluPfS_i
        .type           _Z4reluPfS_i,@function
        .size           _Z4reluPfS_i,(.L_x_24 - _Z4reluPfS_i)
        .other          _Z4reluPfS_i,@"STO_CUDA_ENTRY STV_DEFAULT"
_Z4reluPfS_i:
.text._Z4reluPfS_i:
        /*0000*/                   MOV R1, c[0x0][0x28] ;
        /*0010*/                   S2R R4, SR_CTAID.X ;
        /*0020*/                   S2R R3, SR_TID.X ;
        /*0030*/                   IMAD R4, R4, c[0x0][0x0], R3 ;
        /*0040*/                   ISETP.GE.AND P0, PT, R4, c[0x0][0x170], PT ;
        /*0050*/               @P0 EXIT ;
        /*0060*/                   MOV R5, 0x4 ;
        /*0070*/                   IMAD.WIDE R2, R4, R5, c[0x0][0x160] ;
        /*0080*/                   LDG.E.SYS R2, [R2] ;
        /*0090*/                   IMAD.WIDE R4, R4, R5, c[0x0][0x168] ;
        /*00a0*/                   FMNMX R7, RZ, R2, !PT ;
        /*00b0*/                   STG.E.SYS [R4], R7 ;
        /*00c0*/                   EXIT ;
.L_x_14:
        /*00d0*/                   BRA `(.L_x_14);
        /*00e0*/                   NOP;
        /*00f0*/                   NOP;
```

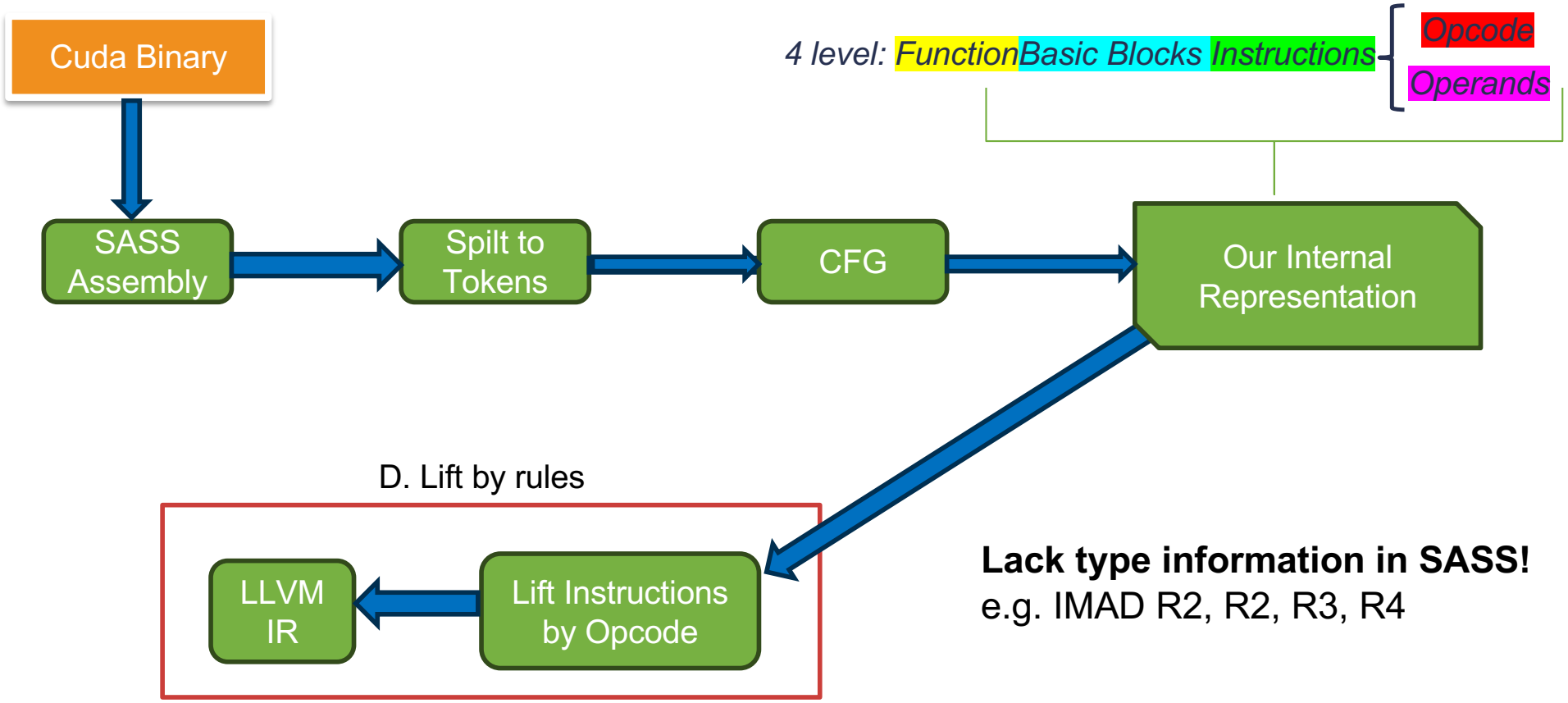
Design– Parse SASS



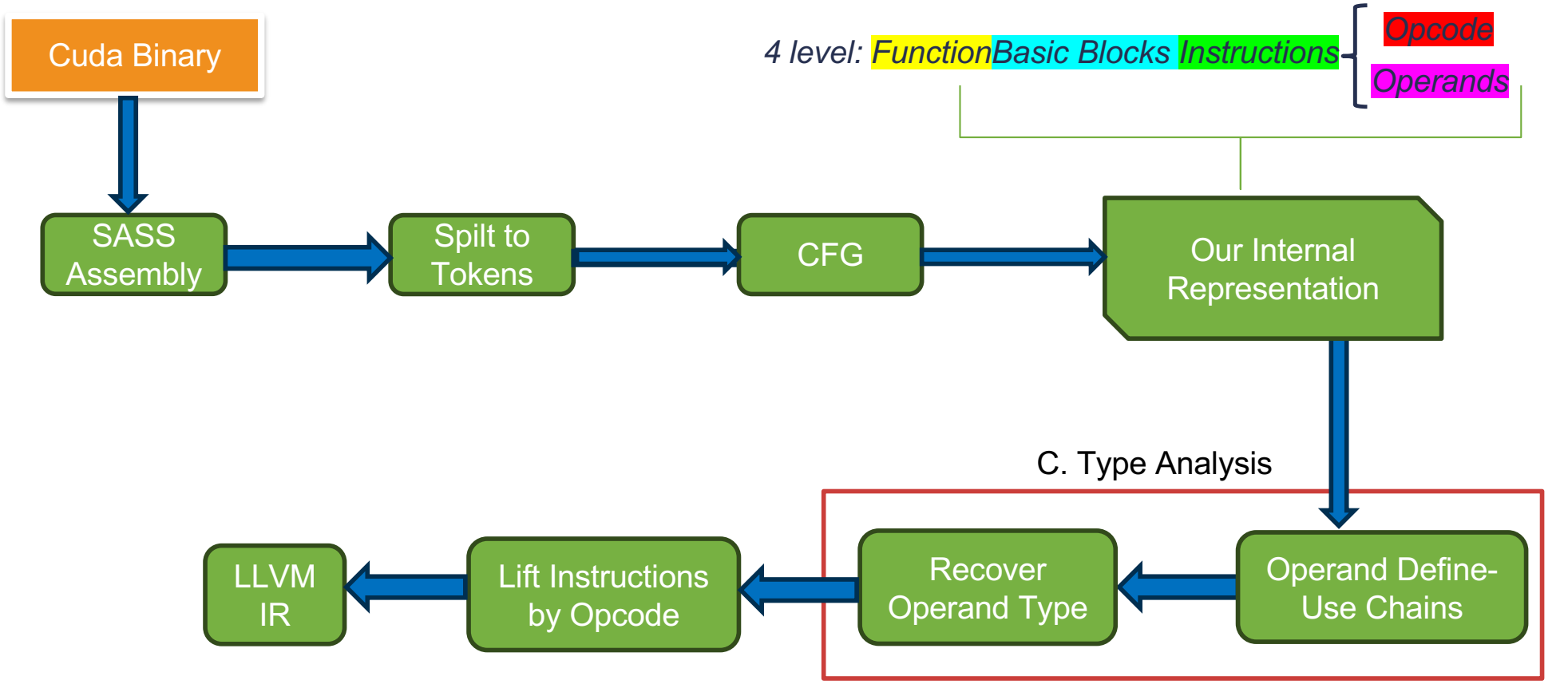
Design– Parse SASS



Design – Lifting by Rules



Design – Type Analysis



Type Analysis

- Infer types from opcode
  - IMAD R4, R4,R5, R3 & FADD R7, R2, R7

Type Analysis

- Infer types from opcode
- Infer types from context
  - e.g., MOV R6, R2
  - Operand Define-Use Chains on the CFG

Type Analysis

- Infer types from opcode
- Infer types from context
- Infer 64-bit types
  - R0-R255 are 32-bit registers
  - 2 registers assembly one 64-bit register
  - E.g., "IMAD.WIDE R2, R2, R3, R4" R2→ R3||R2

NVLift Example: IMAD Instruction

```
; IMAD R3, R3, c[0x0][0x4], R4
```

R3=R3*c[0x0][0x4] + R4

```
%".35" = load i32, ptr %"R3"
%"nvvm_blockdim_y" = call i32 @"llvm.nvvm.read.ptx.sreg.ntid.y"()
%".36" = load i32, ptr %"R4"
%"mul" = mul i32 %".35", %"nvvm_blockdim_y"
%"add" = add i32 %"mul", %".36"
store i32 %"add", ptr %"R3"
```

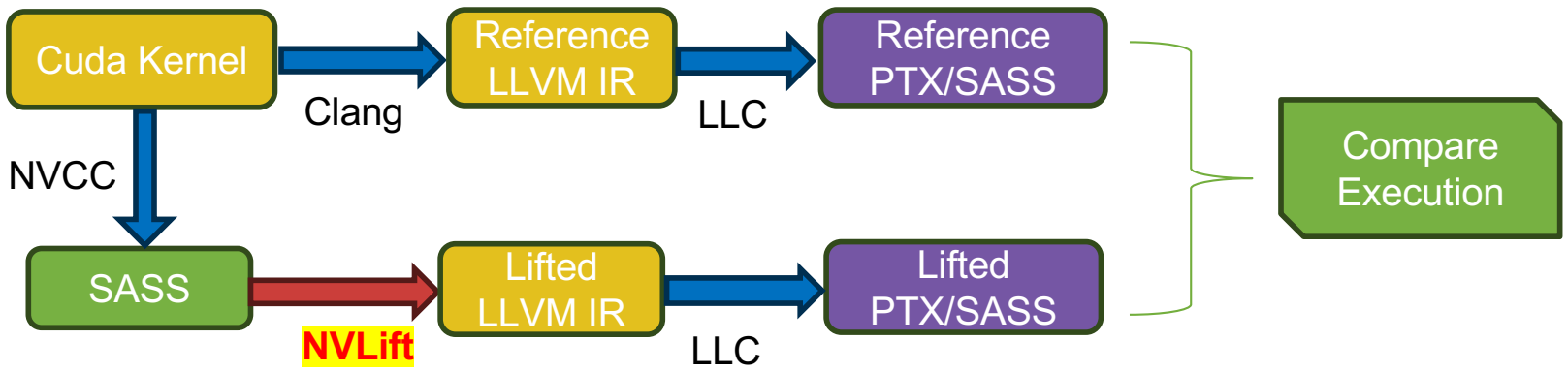
Load R3, c[0x0][0x4], R4 to temporary LLVM variables

Multiply and Add in LLVM IR

Store the result to the R3 register

Differential Testing Pipeline

- Check the semantic correctness of the lifted LLVM IR



Implementation / Evaluation

- Works for Turing (SM75)
- Supports 47 core SASS instructions
  - ~90% occurrence coverage
- Lifted SASS of 11 CUDA Kernels

Opcode	Modifiers
EXIT	–
NOP	–
BRA	–
S2R	–
MOV	–
UMOV	–
LDG	–
LDS	–
LDL	–
ULDC	–
STG	.E
STS	.E
STL	.E
IMAD	.WIDE, .MOV, .IADD, .SHL, .HI, .X, .U32, .S32
IABS	–
ISETP	.AND, .OR, .EQ, .NE, .LT, .LE, .GT, .GE, .FTZ
UISETP	.AND, .OR, .EQ, .NE, .LT, .LE, .GT, .GE, .FTZ
FSETP	.AND, .OR, .EQ, .NE, .LT, .LE, .GT, .GE, .FTZ
IADD	.FTZ
FADD	.FTZ
IADD3	.X
UIADD3	.X
IMNMX	–
FMNMX	–
FFMA	–
LEA	–
LOP3	–
ULOP3	–
PLOP3	–
I2I	–
I2F	–
F2F	–
F2I	.FTZ, .NTZ, .U32, .S32, .TRUNC, .FLOOR, .CEIL
MUFU	–
BMOV	–
BSSY	–
BSYNC	–
SEL	–
FSEL	–
USEL	–
SHF	.L, .R, .W, .C, .U32, .S32, .U64, .S64, .HI
USHF	.L, .R, .W, .C, .U32, .S32, .U64, .S64, .HI
CALL	.REL, .NOINC
RET	–
IMUL	–
FMUL	–
BAR	.SYNC

Case Study: Decompilation

- Lifted LLVM IR benefits future analysis.
- We leverage RetDec, an LLVM-based decompiler.
- 7 of 11 CUDA kernel binaries are successfully decomplied.

```
__global__ void relu(float *input, float *output, int size) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < size)
        output[idx] = fmaxf(0.0f, input[idx]);
}
```

```
void _Z4reluPfS_i(float32_t * a1, float32_t * a2, int32_t a3) {
    uint32_t v1 = llvm_nvvm_read_ptx_sreg_ctaid_x() * llvm_nvvm_read_ptx_sreg_ntid
    if (v1 < a3 || false) {
        int64_t v2 = (int64_t)v1 * (int64_t)4 + (int64_t)a1;
        int32_t v3 = v2;
        float32_t v4 = *(float32_t *)(char *)(0x100000000 * (int64_t)(int32_t)((v2
        *(float32_t *)&v3 = v4;
        int64_t v5 = (int64_t)v1 * (int64_t)4 + (int64_t)a2;
        float32_t v6 = *(float32_t *)&v3;
        int32_t v7;
        *(float32_t *)&v7 = 0 ? 0.0f < v6 ? 0.0f : v6 : 0.0f > v6 ? 0.0f : v6;
        float32_t v8 = *(float32_t *)&v7;
        *(float32_t *)(char *)(0x100000000 * (int64_t)(int32_t)((v5 & -0x100000000
    }
}
```

Conclusion

NVLift, an end-to-end lifter from SASS to LLVM IR.

A differential testing–based method to verify the semantic correctness

The value for downstream analysis

- CUDA kernel decompilation as a case study

Thank you!

Code: https://github.com/purseclab/Sass-LLVM-Lifter

Email: wan155@purdue.edu