# FlowMatrix: GPU-Assisted Information-Flow Analysis through Matrix-Based Representation

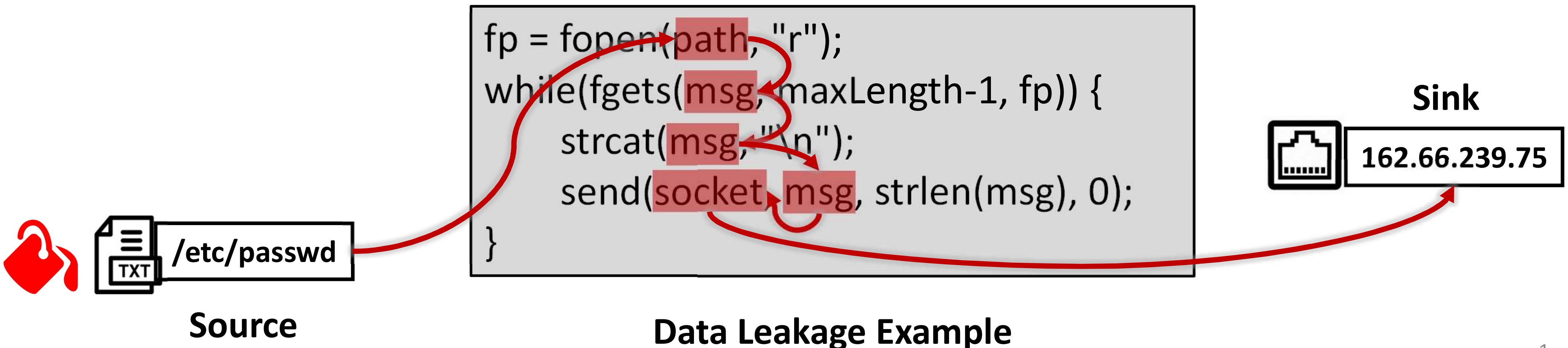
**Kaihang Ji**, Jun Zeng, Yuancheng Jiang, Zhenkai Liang,
Zheng Leong Chua, Prateek Saxena, Abhik Roychoudhury

USENIX Security Symposium, August 2022

NUS
National University of Singapore

# Dynamic Information Flow Tracking (DIFT)

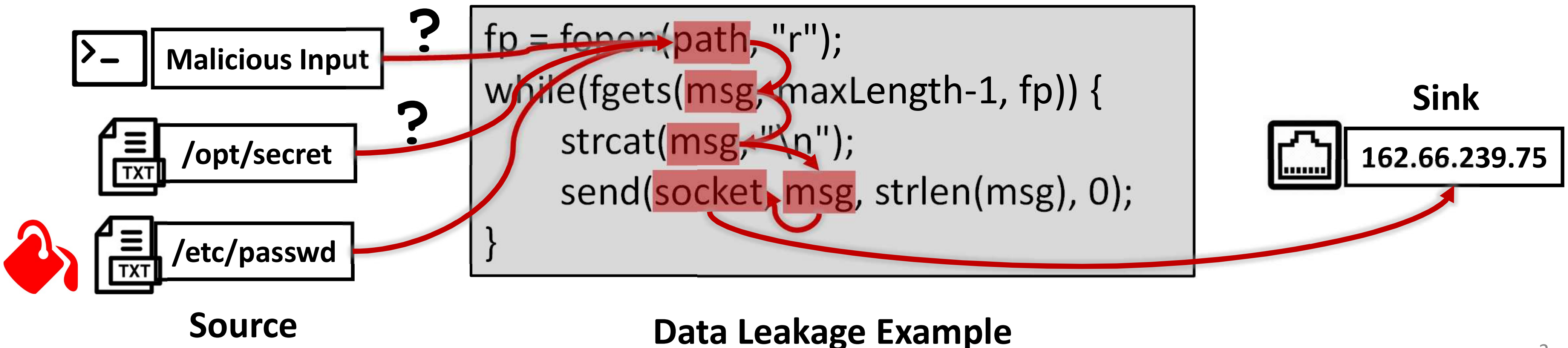
- **DIFT** (aka **Dynamic Taint Analysis**): An important program analysis technique in security
- Track information flows in a program: Taint state transforms between **sources** and **sinks** of interest
- Security applications: Vulnerability analysis, Configuration diagnosis, etc.




Data Leakage Example

# Dynamic Information Flow Tracking (DIFT)

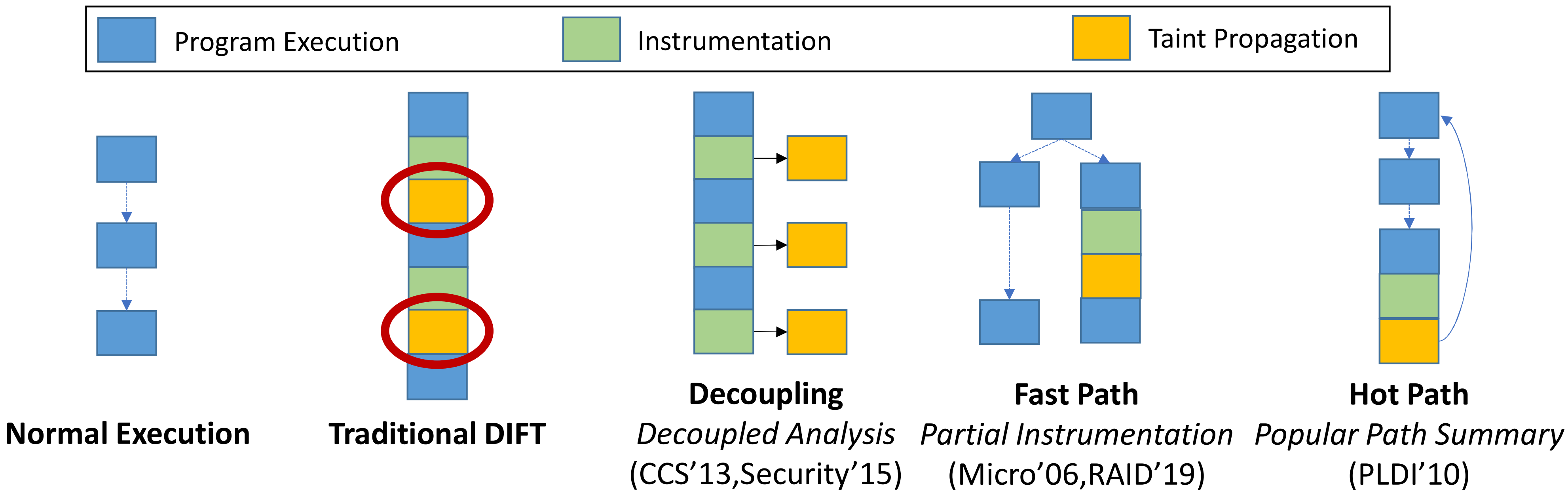
- **Challenge**: Users often need to check multiple information flows
  - Calls for efficient **DIFT Query**: Rapidly DIFT with different given sources and sinks
- DIFT is **expensive**: 4~8 times performance overhead
  - One way to support DIFT query: Heavy computing support (OSDI'16)
  - Another way is to speed up DIFT itself…




Data Leakage Example

# Problem of DIFT

- Existing work of accelerating DIFT




- Lack of speeding up propagation operation itself.

# Complexity of DIFT Operation Rules

- **Taint propagation logics** in current DIFT mechanisms are
  - Implemented in high-level programming languages with *if* and *loops*
  - Unnecessarily complex
  - Challenging to be computationally speeded-up

```
void taint_parallel_compute(shad,
    dest, opcode, ...)
{
 if (opcode == llvm::Instruction::
    Or) {
  cb_mask_out.cb_mask =
   (cb_mask_1.zero_mask &
    cb_mask_2.cb_mask) |
   (cb_mask_2.zero_mask &
    cb_mask_1.cb_mask);
 }
 write_cb_masks(shad, dest,
    cb_mask_out, ...);
 ...
}
```

(a) Panda

```
void r2r_binary_opl(dst, src, ...)
     {
 thread_ctx->vcpu.gpr[dst] |=
     thread_ctx->vcpu.gpr[src];
}
void ins_inspect(INS ins) {
 ...
 switch (ins_indx) {
  case XED_ICLASS_OR:
   INS_InsertCall(
    r2r_binary_opl,
    REG32_INDX(reg_dst),
    REG32_INDX(reg_src), ...);
 }
}
```

(b) Libdft

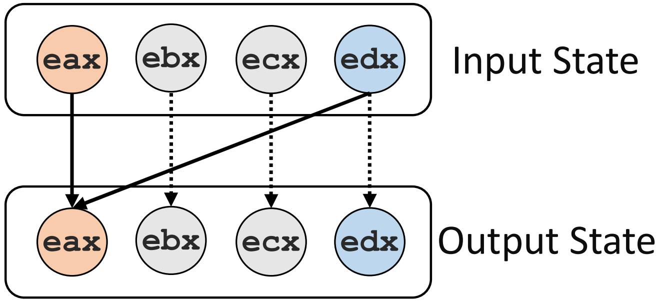
```
int gen_taintcheck_insn(...) {
 switch(opc) {
  case INDEX_op_or_i32:
   /* t0 = arg1 || arg2 */
   tcg_gen_or_i32(t0,arg1,arg2);
   /* t2 = (t0 != 0)     */
   tcg_gen_movi_i32(t_zero,0);
   tcg_gen_setcond_i32(TCG_COND_NE
     ,t2,t_zero,t0);
   /* result = ~t2       */
   tcg_gen_neg_i32(result,t2);
   break;
   ...
 }
}
```

(c) Decaf

Different implementations of taint propagation rule of **or** instruction in **Panda**, **libdft** and **DECAF**.

# Insights

- DIFT propagation logic is data dependency (TaintInduce NDSS'19)
- Example: DIFT operations for x86 instructions




DIFT operations for instruction

`OR eax, edx`

$$
\begin{aligned}
eax_{out} &= 1 * eax_{in} + 1 * edx_{in} \\
ebx_{out} &= 1 * ebx_{in} \\
ecx_{out} &= 1 * ecx_{in} \\
edx_{out} &= 1 * edx_{in}
\end{aligned}
$$

Dependencies in Boolean space

$$
\begin{aligned}
eax_{out} &= 1 * eax_{in} + 0 * ebx_{in} + 0 * ecx_{in} + 1 * edx_{in} \\
ebx_{out} &= 0 * eax_{in} + 1 * ebx_{in} + 0 * ecx_{in} + 0 * edx_{in} \\
ecx_{out} &= 0 * eax_{in} + 0 * ebx_{in} + 1 * ecx_{in} + 0 * edx_{in} \\
edx_{out} &= 0 * eax_{in} + 0 * ebx_{in} + 0 * ecx_{in} + 1 * edx_{in}
\end{aligned}
$$

Dependencies in a verbose form

# Insights

- DIFT propagation logic is data dependency (TaintInduce NDSS'19)
- Example: DIFT operations for x86 instructions




DIFT operations for instruction

`OR eax, edx`

$$eax_{out} = 1 * eax_{in} + 1 * edx_{in}$$
$$ebx_{out} = 1 * ebx_{in}$$
$$ecx_{out} = 1 * ecx_{in}$$
$$edx_{out} = 1 * edx_{in}$$

Dependencies in Boolean space

$$eax_{out} = 1 * eax_{in} + 0 * ebx_{in} + 0 * ecx_{in} + 1 * edx_{in}$$
$$ebx_{out} = 0 * eax_{in} + 1 * ebx_{in} + 0 * ecx_{in} + 0 * edx_{in}$$
$$ecx_{out} = 0 * eax_{in} + 0 * ebx_{in} + 1 * ecx_{in} + 0 * edx_{in}$$
$$edx_{out} = 0 * eax_{in} + 0 * ebx_{in} + 0 * ecx_{in} + 1 * edx_{in}$$

Dependencies in a verbose form

# Insights

We identify the **linearity** in DIFT:

- The DIFT operation between input states and output states is a linear relationship.

A system of linear equations:

$$f: S_{in} \rightarrow S_{out}$$

$$\begin{aligned}
eax_{out} &= 1 * eax_{in} + 0 * ebx_{in} + 0 * ecx_{in} + 1 * edx_{in} \\
ebx_{out} &= 0 * eax_{in} + 1 * ebx_{in} + 0 * ecx_{in} + 0 * edx_{in} \\
ecx_{out} &= 0 * eax_{in} + 0 * ebx_{in} + 1 * ecx_{in} + 0 * edx_{in} \\
edx_{out} &= 0 * eax_{in} + 0 * ebx_{in} + 0 * ecx_{in} + 1 * edx_{in}
\end{aligned}$$

Dependencies in a verbose form

# DIFT Operations as Matrix Transformations

FlowMatrix: a new matrix-based representation of DIFT propagation rule

$$\begin{aligned} eax_{out} &= 1 * eax_{in} + 0 * ebx_{in} + 0 * ecx_{in} + 1 * edx_{in} \\ ebx_{out} &= 0 * eax_{in} + 1 * ebx_{in} + 0 * ecx_{in} + 0 * edx_{in} \\ ecx_{out} &= 0 * eax_{in} + 0 * ebx_{in} + 1 * ecx_{in} + 0 * edx_{in} \\ edx_{out} &= 0 * eax_{in} + 0 * ebx_{in} + 0 * ecx_{in} + 1 * edx_{in} \end{aligned}$$

$$\begin{pmatrix} eax_{out} \\ ebx_{out} \\ ecx_{out} \\ edx_{out} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} eax_{in} \\ ebx_{in} \\ ecx_{in} \\ edx_{in} \end{pmatrix}$$

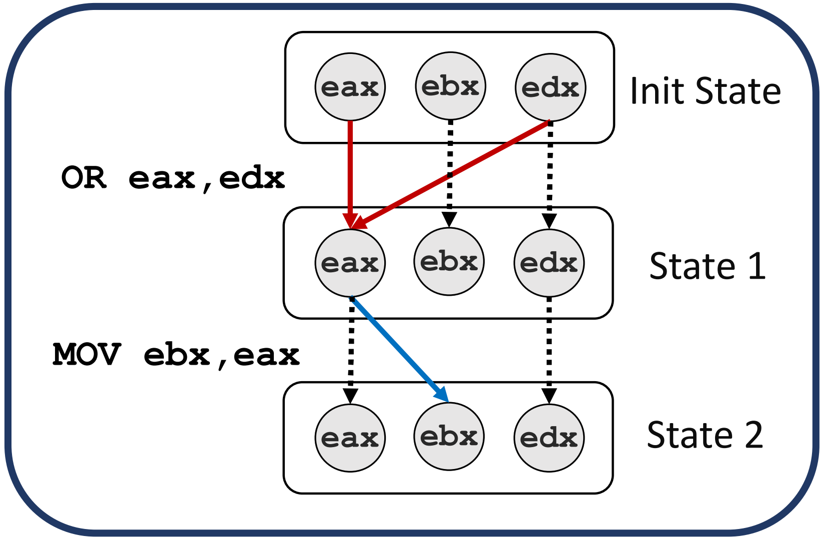
The coefficient matrix, the dependencies between $S_{in}$ and $S_{out}$

# Propagation Summary as Matrix Multiplication

- Example: DIFT propagation of two x86 instructions
- Summarizing two DIFT propagation rules is to multiply two FlowMatrices:

$$M_{sum} = M_2 \times M_1$$

- FlowMatrix operations: matrix-matrix multiplication, etc.



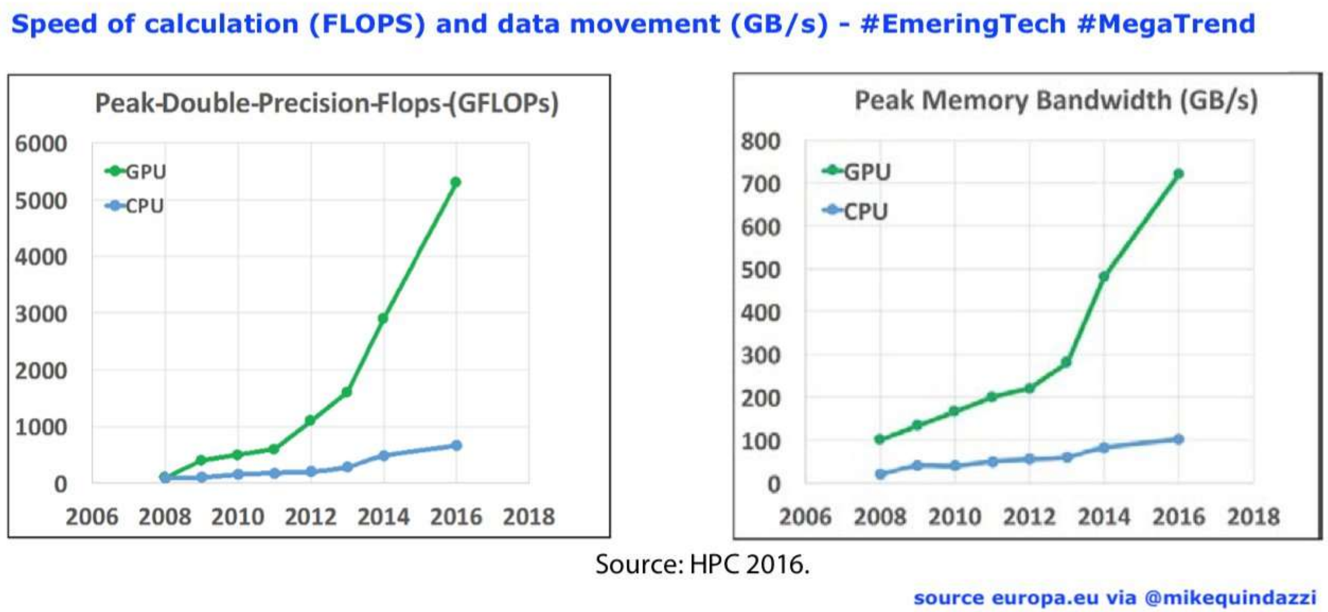

$$\begin{pmatrix} eax_{out} \\ ebx_{out} \\ edx_{out} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} eax_{in} \\ ebx_{in} \\ edx_{in} \end{pmatrix}$$

$$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Summarized Matrix

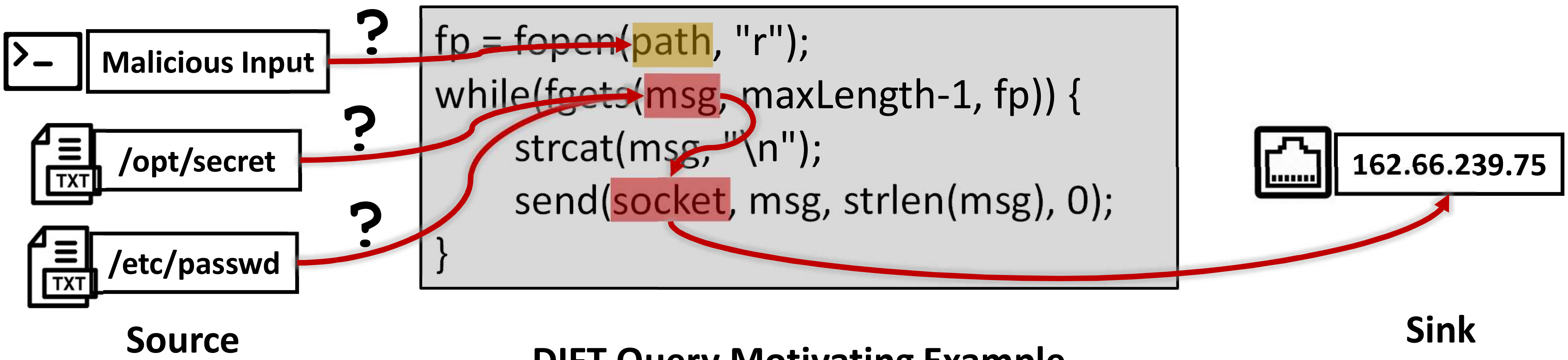
# GPU-assisted DIFT Operations

- GPUs are suitable for highly parallel applications such as matrix and vector computations.
- FlowMatrix operations are accelerated by GPUs!



# GPU-assisted FlowMatrix-based DIFT Query

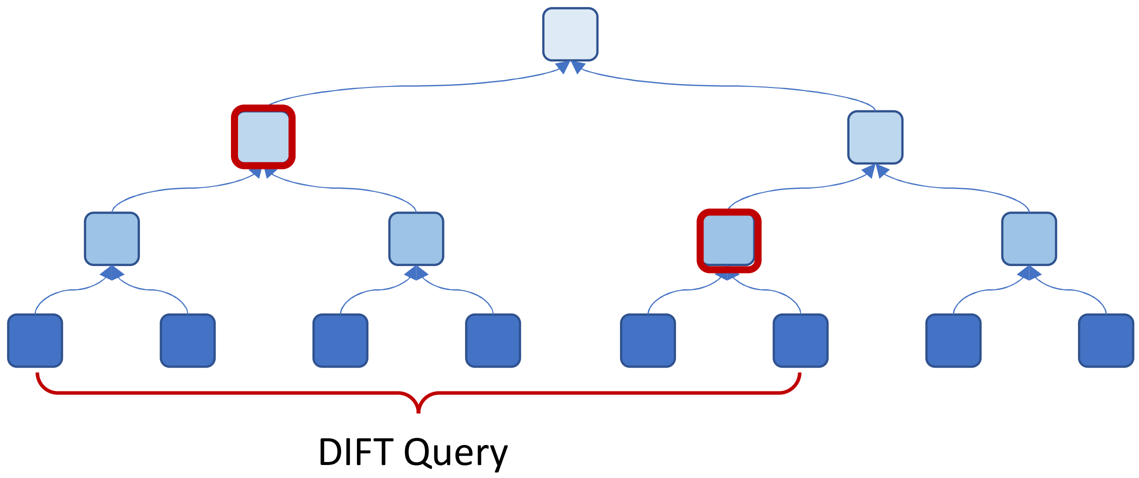
- How can GPUs and FlowMatrix support efficient DIFT queries?
  - Answer a query by propagating each instruction sequentially? ☹ Query too slow
  - Prepare queries by pre-computing every possible query? ☹ Too much to prepare
- **Goal**: Reasonable pre-processing cost and rapid query response




DIFT Query Motivating Example

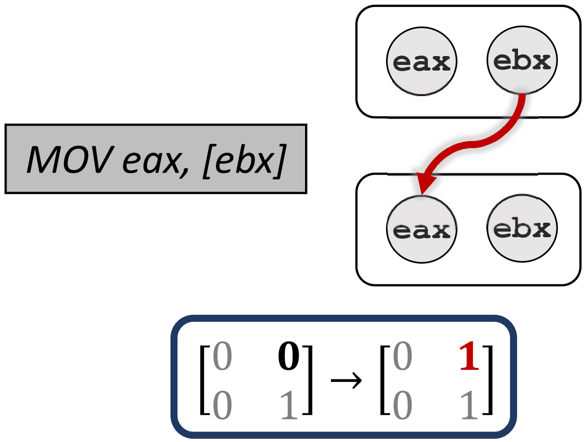
# Trace-based Repeated DIFT Query

- Offline DIFT query on instruction execution traces
- (Segment-tree-like) Query Tree
  - Leaf nodes: FlowMatrix for a single instruction
  - Non-leaf nodes: Summarized FlowMatrices of two child nodes
- Pre-processing (Tree Construction): Linear time complexity
- Query: Logarithmic time complexity

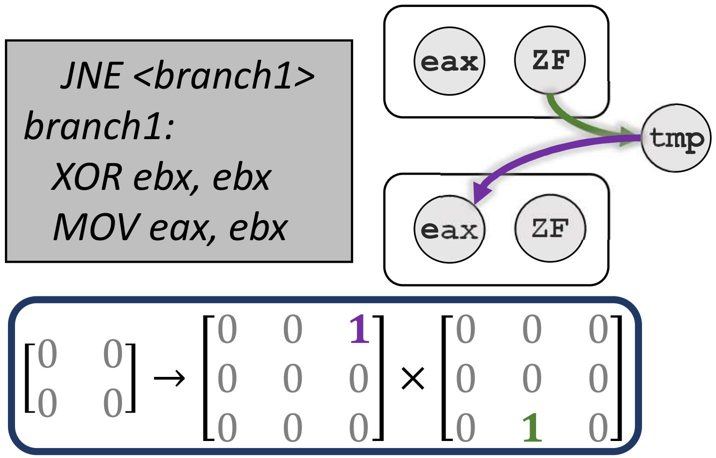


# Under/Over-tracking in DIFT queries

- Improper tracking policy may lead under/over-tracking
  - E.g., Common under-tracking cases: dependencies between pointers and values, between condition and in-branch variables
- How to mutate tracking policy with FlowMatrix?
  - Directly patch DIFT rule matrix
  - Add a temporary variable to bridge information flows




Rule Patching



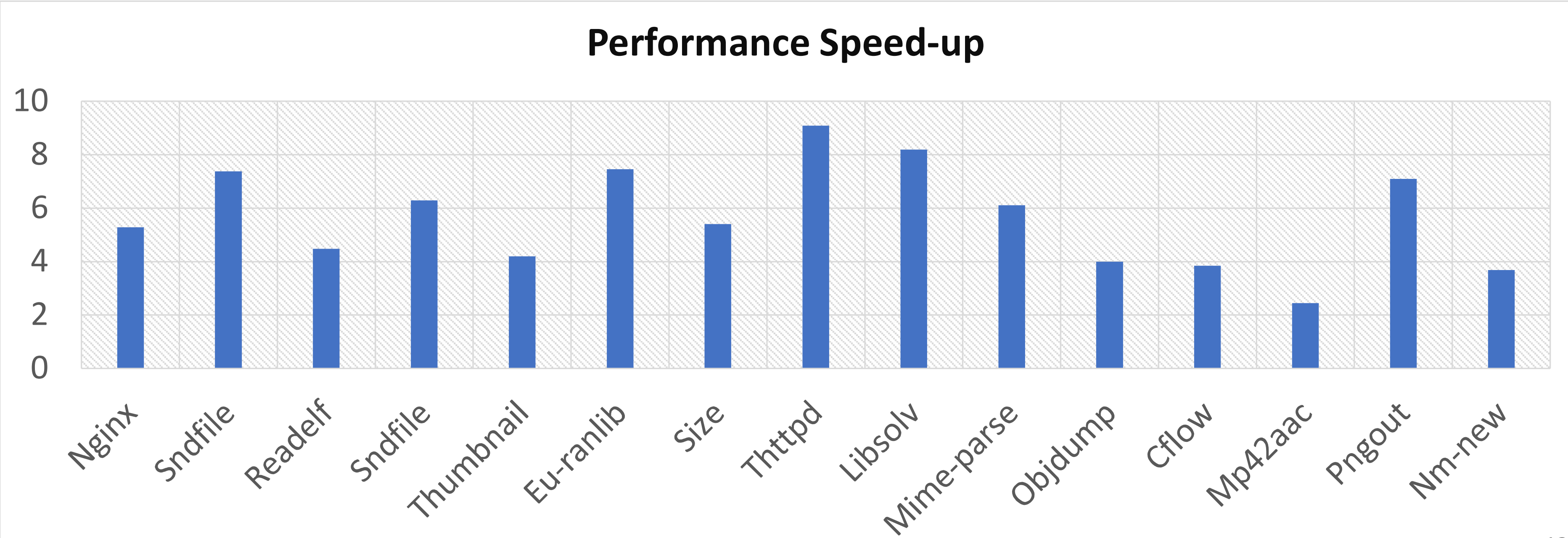

Temporary Variable Bridging

# Evaluation

- Evaluation Aspects
  - Performance
    - How much improvement is achieved by GPU assistance?
    - How fast is FlowMatrix-based DIFT query?
  - Throughput
    - What is the throughput of FlowMatrix-based DIFT queries?
  - Comparison
    - How does FlowMatrix-based DIFT query compare with existing taint tools and DIFT query systems?
- Date Set
  - 15 CVEs and 7 common applications

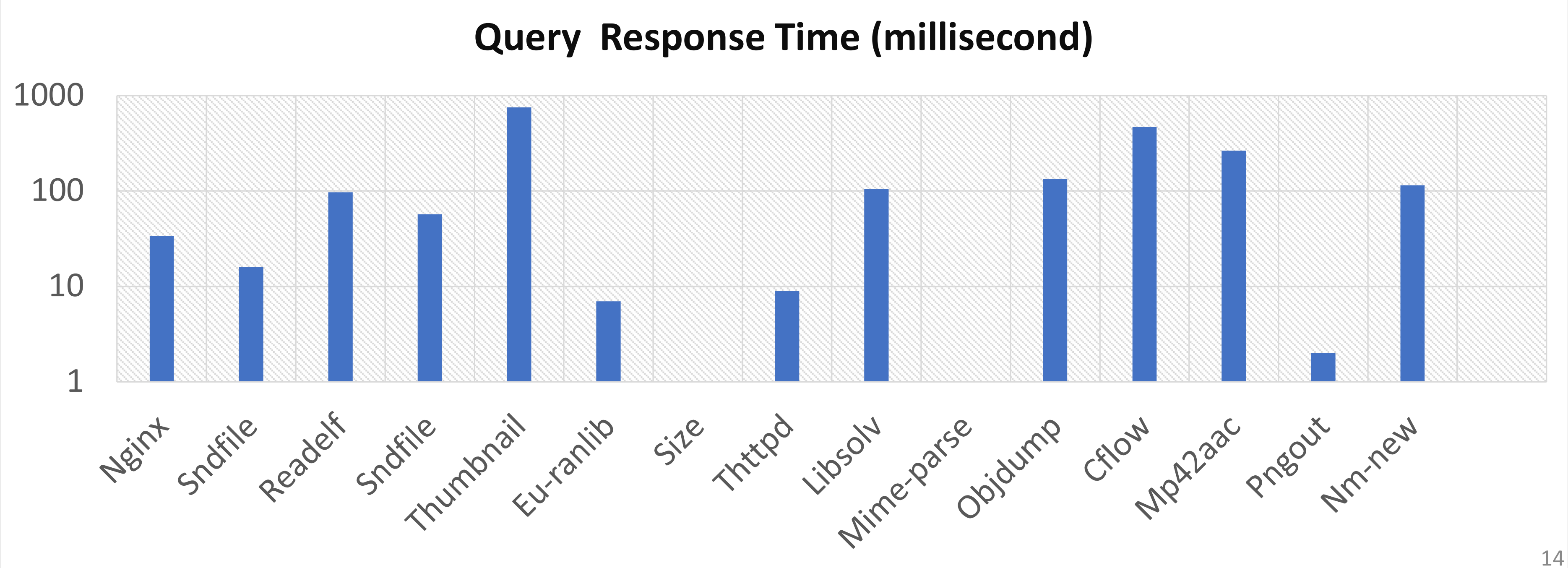
# Evaluation - Performance

- Question: How much improvement achieved by GPU assistance?
- Answer:
  - Our prototype outperforms CPU-based DIFT tool over **5** times in performance on average.



# Evaluation - Performance

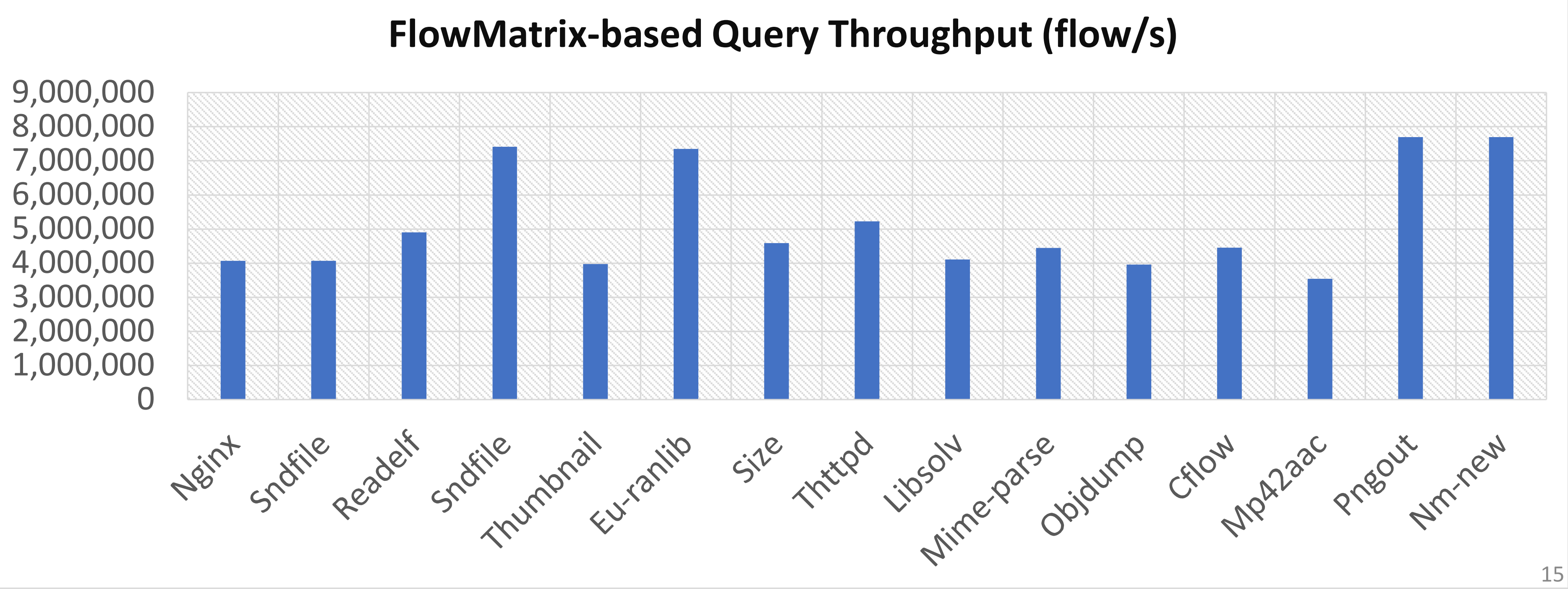
- Question: How fast is FlowMatrix-based DIFT query?
- Answer:
  - Most DIFT query requests can be answered in less than **0.5 sec**.



# Evaluation - Throughput

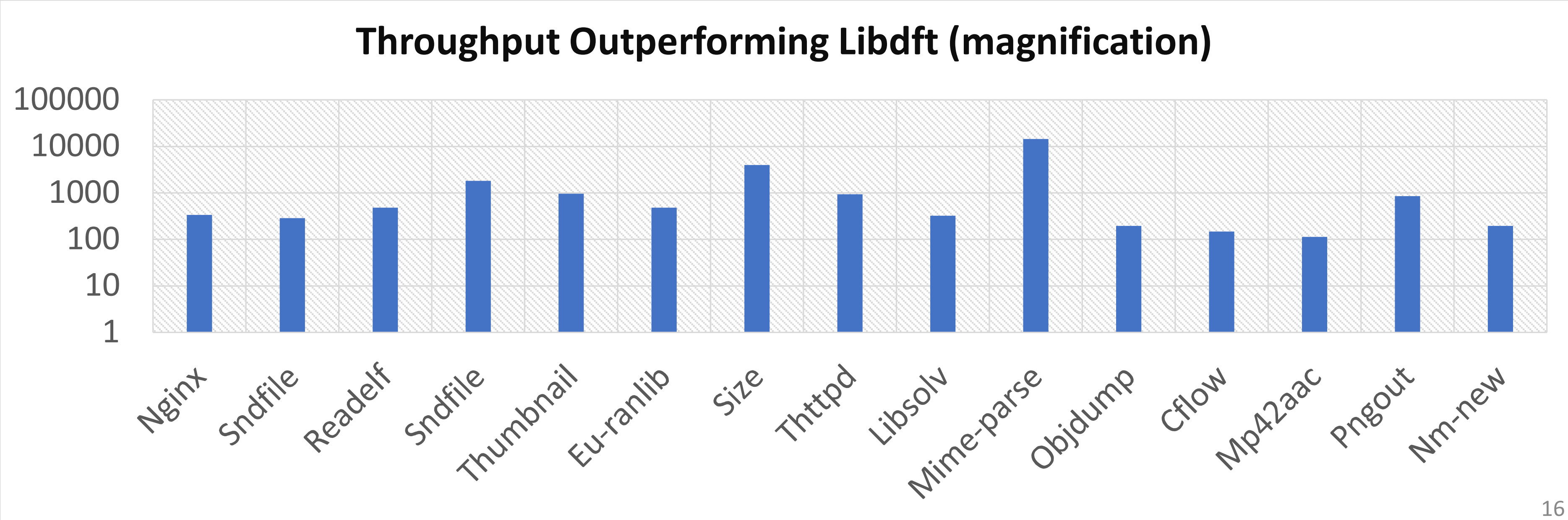
- Question: What is the throughput of the DIFT query operations?

- Answer:
  - Over **5,000,000** dataflows per second on average



# Evaluation - Comparison

- Question: Is FlowMatrix comparable to existing taint engines and DIFT query systems?
- Answer:
  - Three orders of magnitude larger than LibDFT
  - Comparable with JetStream (achieved by 128 CPU cores)



# Summary

- **FlowMatrix: a Matrix-based DIFT Representation**
  - We recognize linearity of dynamic information flow operations
  - We propose a matrix-based representation for DIFT operations
- **GPU-assisted DIFT**
  - FlowMatrix enables GPU as co-processors for efficient DIFT operations
- **DIFT Query**
  - We design an efficient DIFT query with high throughput

# Thanks!

# Q&A

kaihang@comp.nus.edu.sg

Code Available at https://github.com/mimicji/FlowMatrix