

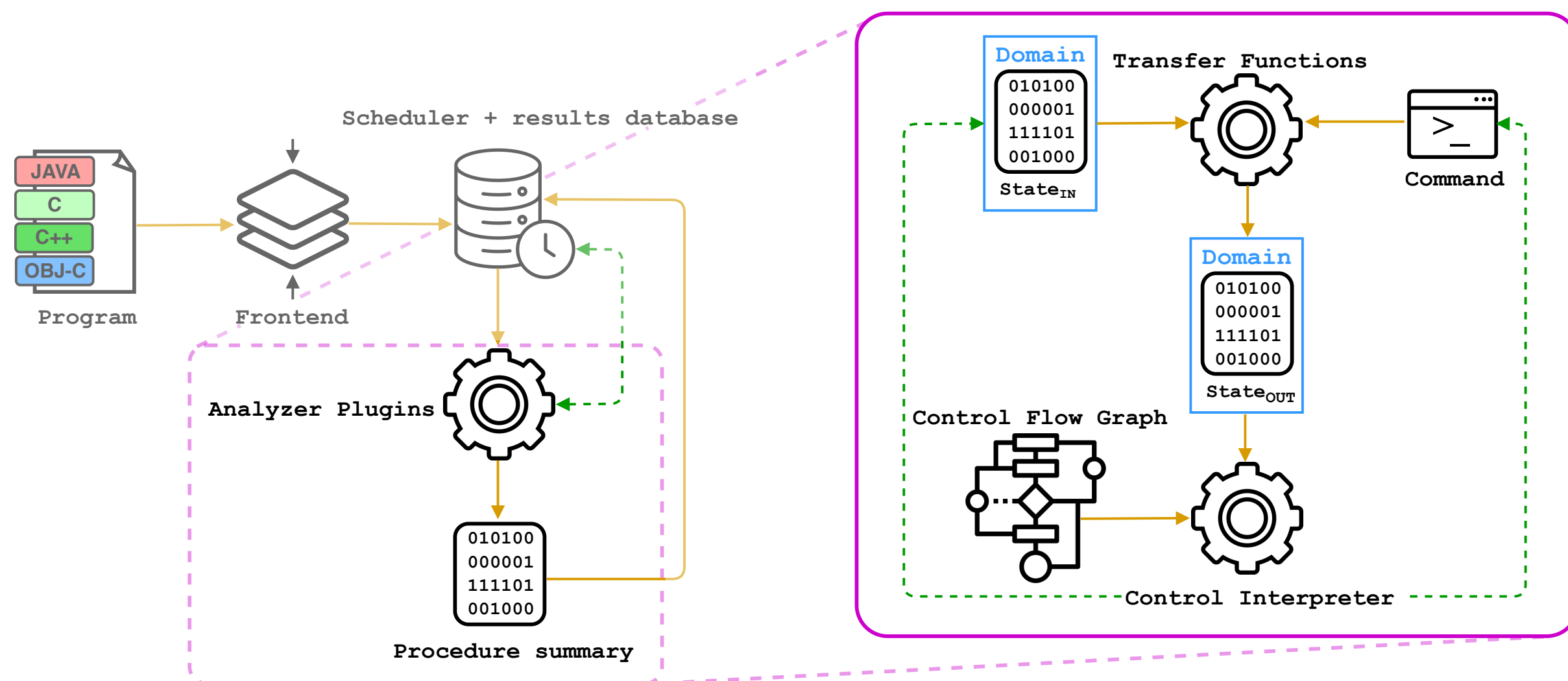
# 59 SCALABLE STATIC ANALYSIS USING FACEBOOK INFER

Dominik Harmim, Vladimír Marcin, Ondřej Pavela

{xharmi00, xmarci10, xpavel34}@stud.fit.vutbr.cz



## FACEBOOK INFER



- Open-source static analysis framework.
- Suite of **modular** static analysers.
  - Checks for, e.g., buffer overflow, thread-safety, null-dereferencing or memory leaks.
- Follows principles of **compositionality**.
  - Highly scalable.
  - Analysis of code changes only.

## LOOPER: Worst-Case Cost Analyser

- Recast of the **Loopus** tool in Infer.
- Supports **amortized complexity** analysis.
- Based on recursive **transition** and **variable** bounds computation.
- Current prototype is **intra-procedural** only.
- Promising results on selected examples in comparison to **Cost Infer** checker.

### Experimental evaluation

	Bound	Looper	Cost
#1	$n$	$2n$	$n^2$
#2	$2n$	$2n$	$5n$
#3	$4n$	$5n$	$\infty$
#4	$*n^2$	$*n^2$	$\infty$
#5	$2n$	$2n$	$12n$
#6	$*n$	$*n$	$\infty$
#7	$2n$	$2n$	$\infty$
#8	$2n$	$2n$	$\infty$

### Bound computation

$TB(\tau_2)$	$\rightarrow \text{Incr}(j) + TB(\tau_0) \times \max(VB(0) + 0, 0)$ $\rightarrow n + 1 \times 0 = n$
$\text{Incr}(j)$	$\rightarrow TB(\tau_1) \times 1 = n \times 1 = n$
$TB(\tau_1)$	$\rightarrow \text{Incr}(i) + TB(\tau_0) \times \max(VB(n) + 0, 0)$ $\rightarrow 0 + 1 \times \max(n + 0, 0) = n$
$VB(n)$	$\rightarrow n$ (formal parameter)

$$\begin{array}{c}
 l_b \\
 \tau_0 \downarrow \begin{array}{l} i' \leq n \\ j' \leq 0 \end{array} \\
 l_1 \longrightarrow l_e \\
 i' \leq i \quad \left( \tau_1 \right) \quad i' \leq i-1 \\
 j' \leq j \quad \left( \tau_3 \right) \quad j' \leq j+1 \\
 l_2 \\
 \tau_2 \cup \quad i' \leq i \\
 \quad \quad j' \leq j-1
 \end{array}$$

## ATOMER: Atomicity Violations Analyser

```

void g(void) {
    f1();
    pthread_mutex_lock(&lock);
    f2();
    f3();
    pthread_mutex_unlock(&lock);
    f4();
}
summary: {(f2 f3)}
    
```

1. phase: Detection of atomic sequences

```

void h(void) {
    f1();
    f2();
    f3();
    f4();
}
    
```

2. phase: Detection of violations of the atomic sequences

- Finds atomicity violations for **sequences of functions**.
- Based on assumption that sequences executed **once atomically** should be executed **always atomically**.
- Adapts the technique of **contracts for concurrency**.
- Targets C/C++ programs that uses **Pthreads** locks.

## L2D2: Low-Level Deadlock Detector

- 11.4 MLOC derived from **debian**.
- 100% deadlock detection rate.
- Roughly 11% FP rate.
- Less than 1% of the time of CPROVER.

### Experimental evaluation

	L2D2	CPROVER
Deadlocks	8	8
False Positives	104	114
No deadlocks	810	292
Failed Cases	80	588
Total	1002	1002

