59 Scalable Static Analysis Using Facebook Infer

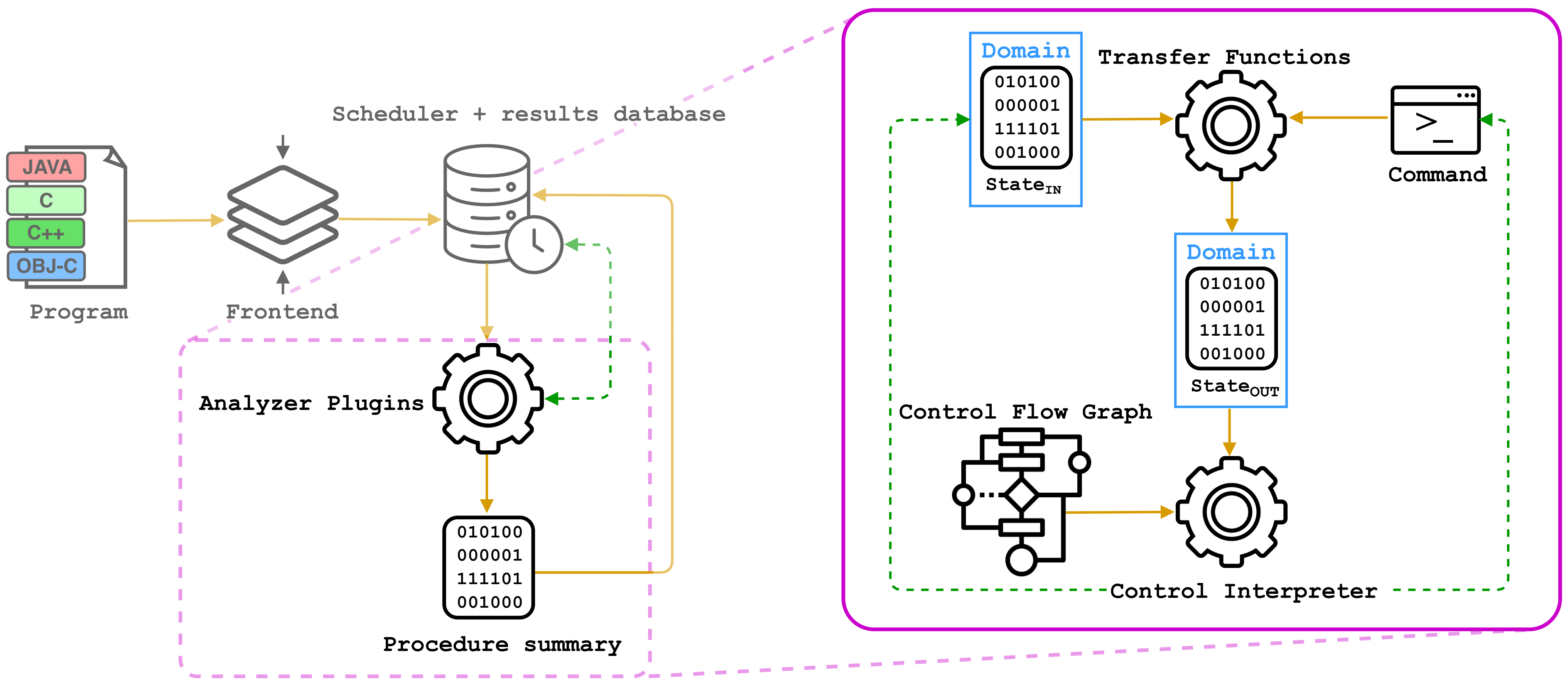
Dominik Harmim, Vladimír Marcin, Ondřej Pavela

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

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2019

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

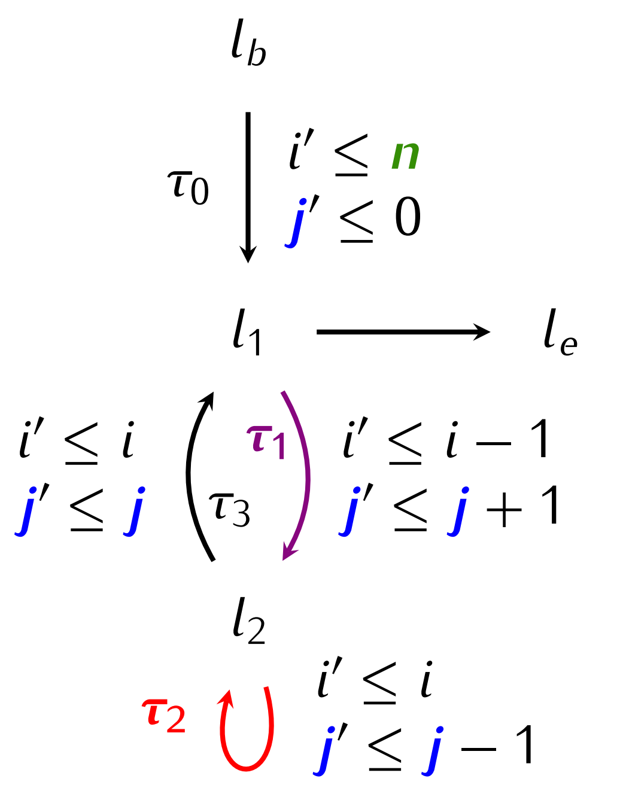
- Recast of the **Loopus** tool in Infer.
- Supports **amortized complexity** analysis.
- Based on recursive **transition** and **variable** bounds computation.
- Current prototype is **intra-procedural** only.
- Promising results on selected examples in comparison to **Cost** Infer checker.

Experimental evaluation

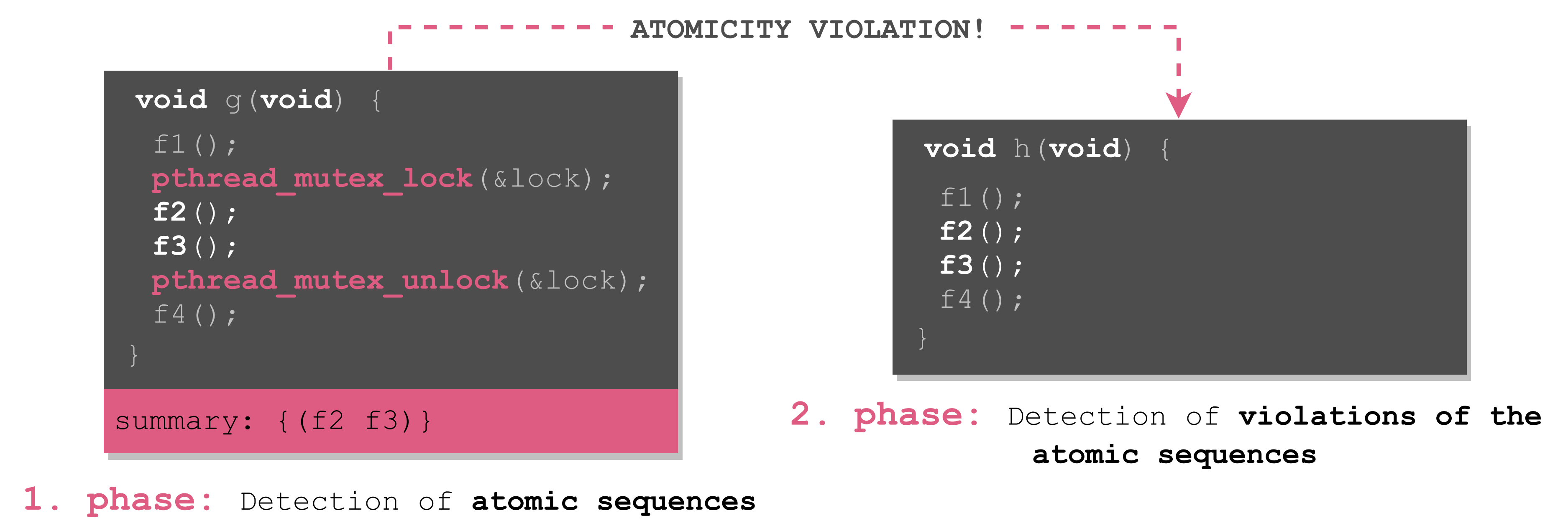
	Bound	Looper	Cost
#1	n	$2n$	n^2
#2	$2n$	$2n$	$5n$
#3	$4n$	$5n$	∞
#4	$^*n^2$	$^*n^2$	∞
#5	$2n$	$2n$	$12n$
#6	$^*n$	$^*n$	∞
#7	$2n$	$2n$	∞
#8	$2n$	$2n$	∞

Bound computation

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



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**

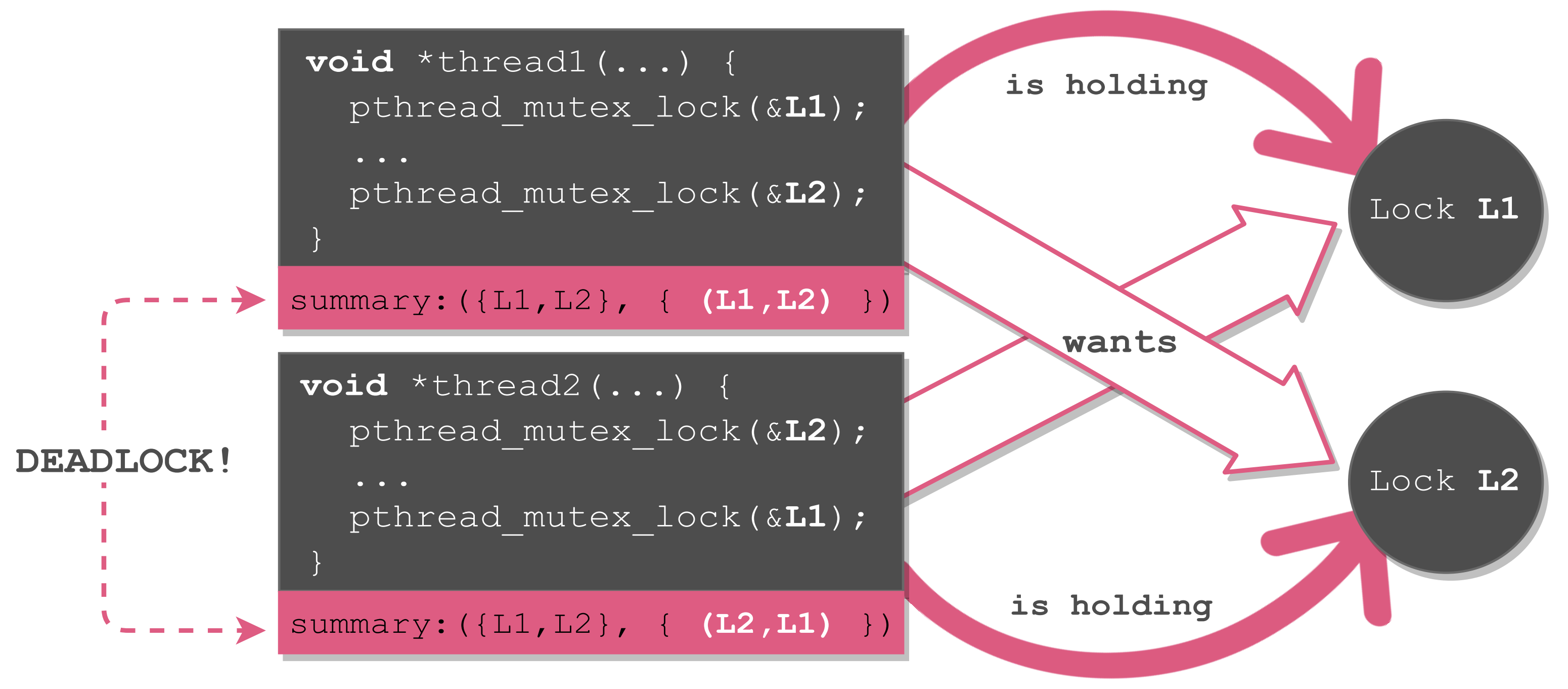
- Finds atomicity violations for **sequences of functions**.
- Based on assumption that sequences executed **once atomically** should be executed **always atomically**.
- Adapts the technique of **contracts for concurrency**.
- Targets **C/C++** programs that uses **Pthreads** locks.

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



```
void *thread1(...) {
  pthread_mutex_lock(&L1);
  ...
  pthread_mutex_lock(&L2);
}
summary:({L1,L2}, { (L1,L2) })
```

```
void *thread2(...) {
  pthread_mutex_lock(&L2);
  ...
  pthread_mutex_lock(&L1);
}
summary:({L1,L2}, { (L2,L1) })
```

The work is supported by the H2020 ECSEL project **AQUAS**: Aggregated Quality Assurance for Systems and **VeriFIT** group (BUT FIT).