Translation Validation for JIT Compiler in the V8 JavaScript Engine

Seungwan Kwon, Jaeseong Kwon, Wooseok Kang, Juneyoung Lee, Kihong Heo

ICSE 2024

Programming
Systems Laboratory

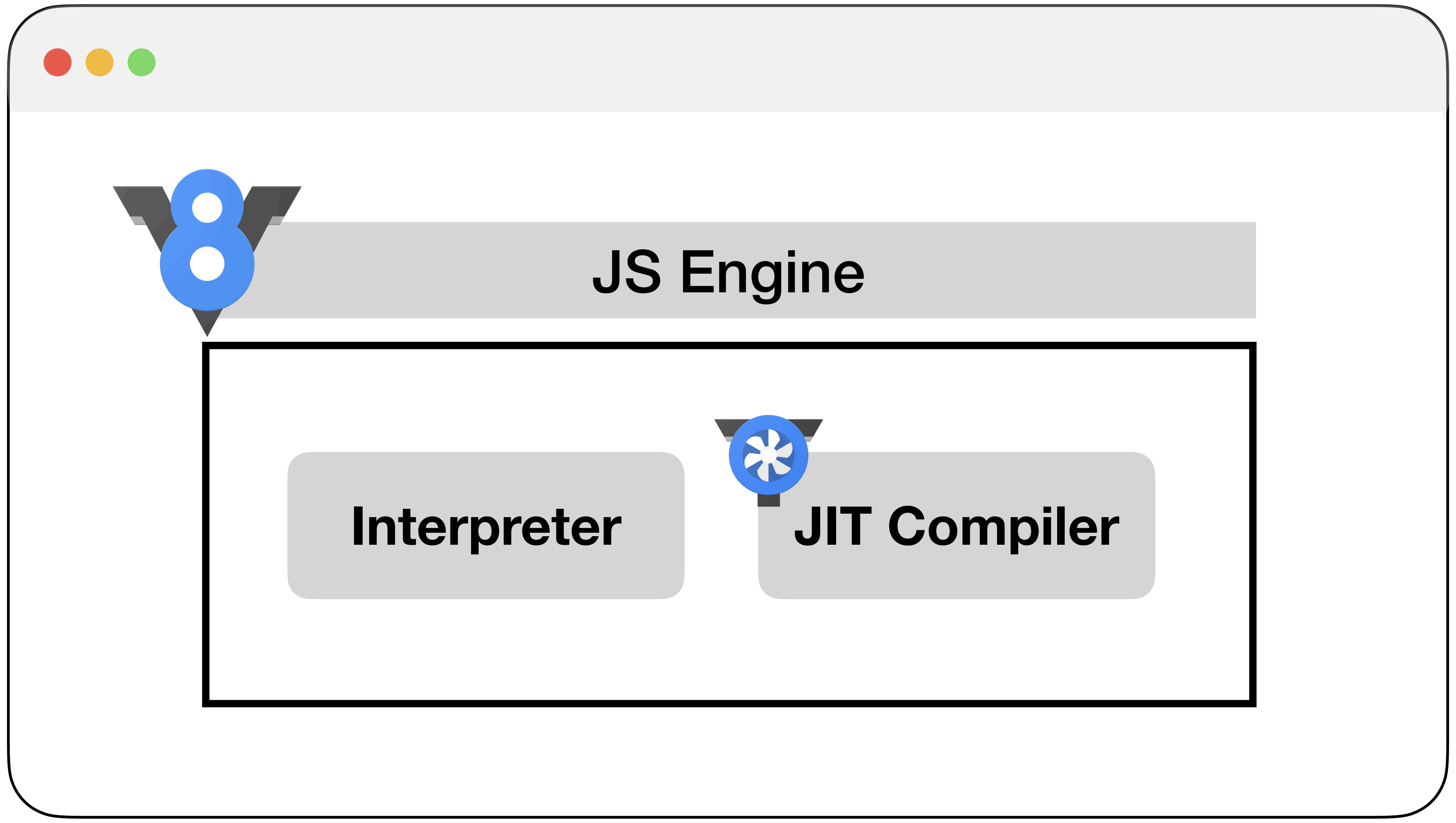
KAIST

aws

V8 JavaScript Engine

JS (JavaScript) Engine made by Google.

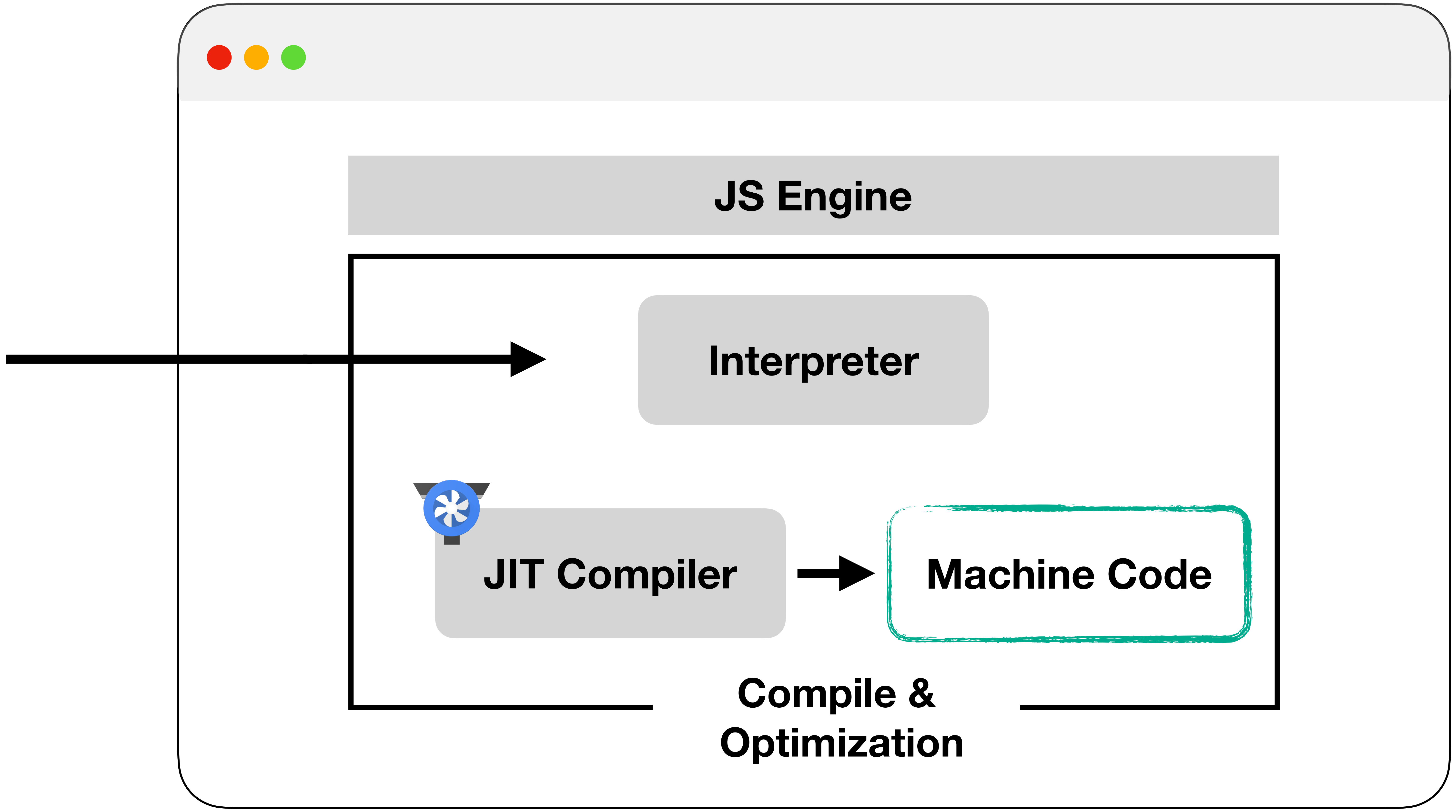
Used by Chrome, Microsoft Edge, Node JS, Amazon Silk, AWS etc.



TurboFan: JIT Compiler of V8

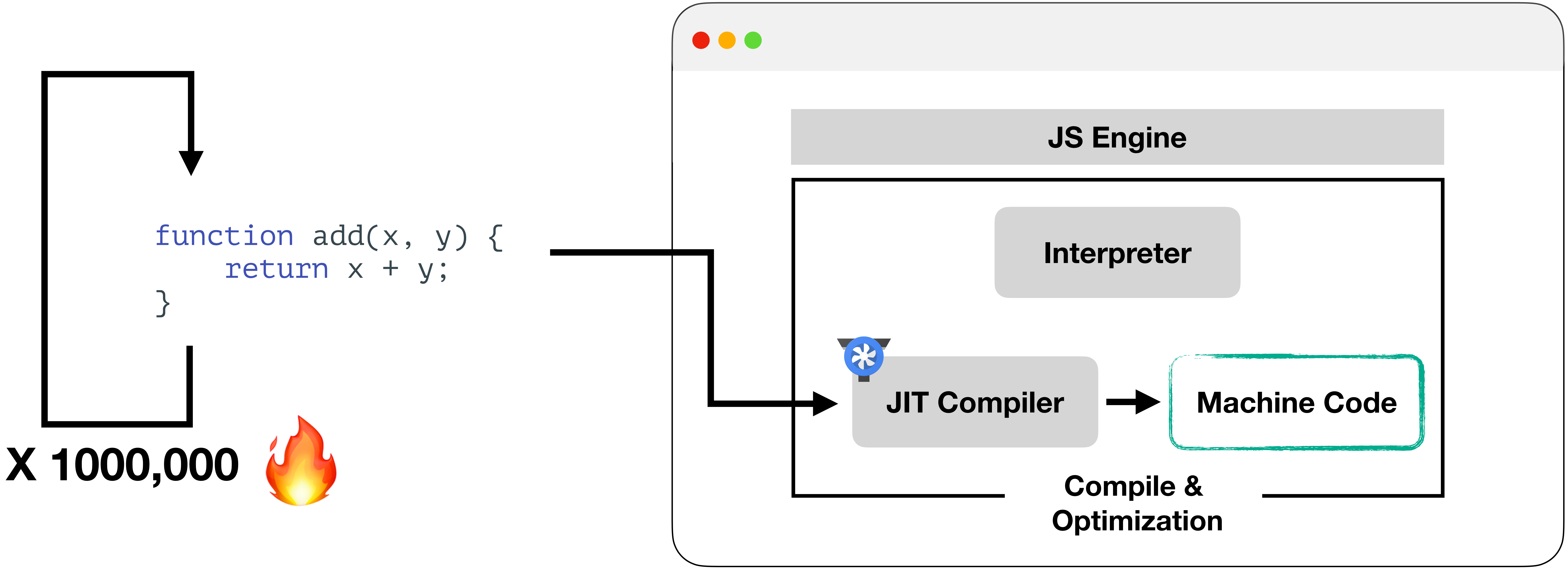
- JIT (Just-In-Time) Compiler for runtime

```
function add(x, y) {
    return x + y;
}
```



TurboFan: JIT Compiler of V8

- JIT (Just-In-Time) Compiler for runtime



```
function add(x, y) {
    return x + y;
}
```

X 1000,000 🔥

JS Engine

Interpreter

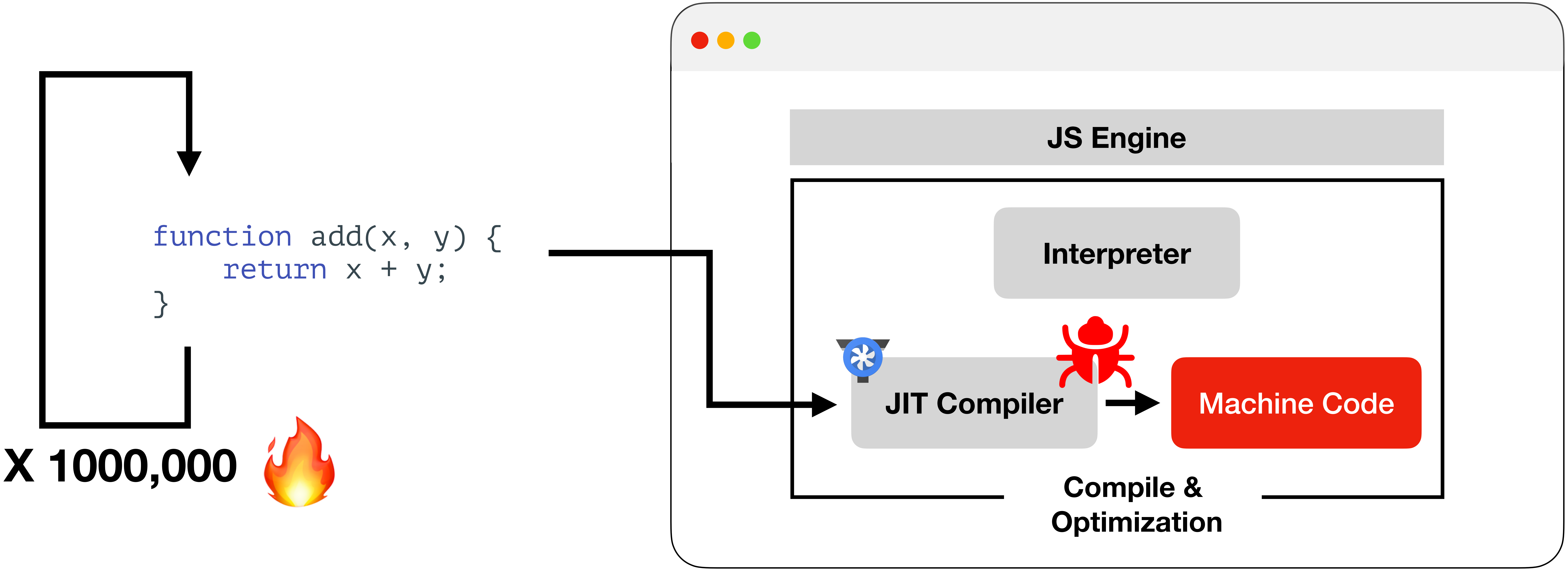
JIT Compiler

Machine Code

Compile & Optimization

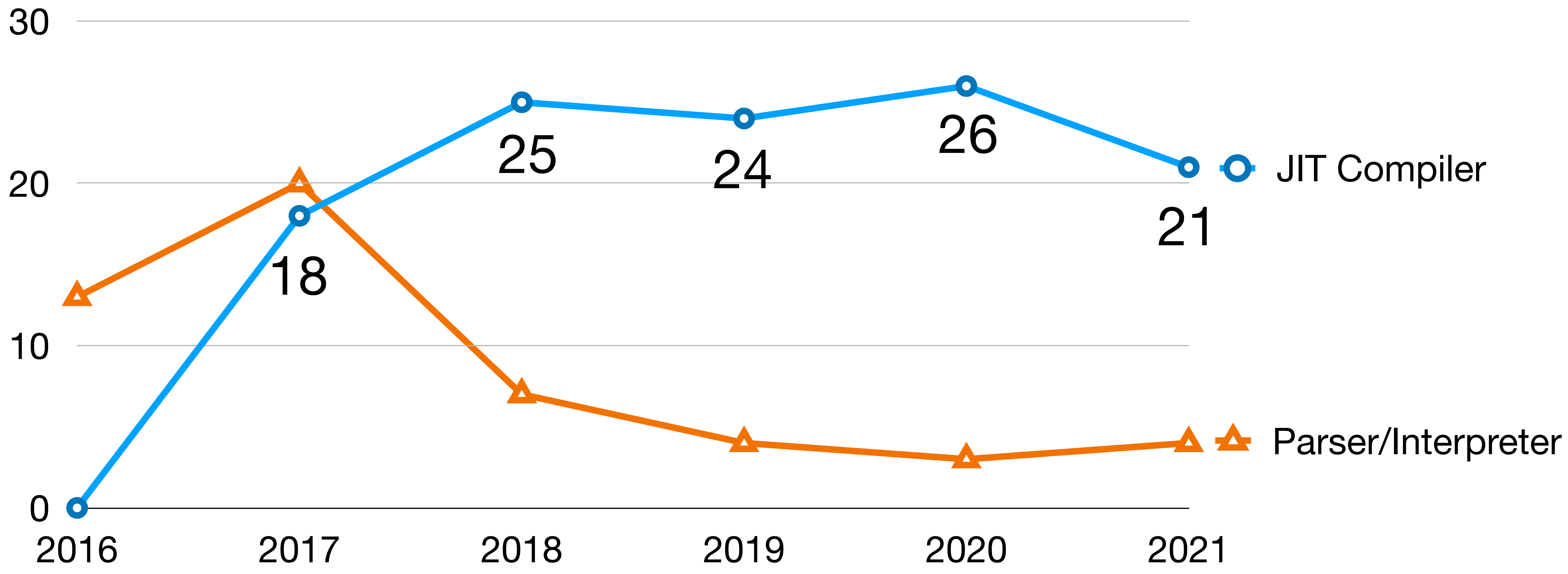
TurboFan: JIT Compiler of V8

- JIT (Just-In-Time) Compiler for runtime



```
function add(x, y) {
    return x + y;
}
```

Motivation: JIT Comiler’s Bugs



The number of bugs discovered by Project Zero in recent years.

J Wang et al. 2023. Oracle-Enhanced Fuzzing for JavaScript Engine JIT Compiler. In USENIX security

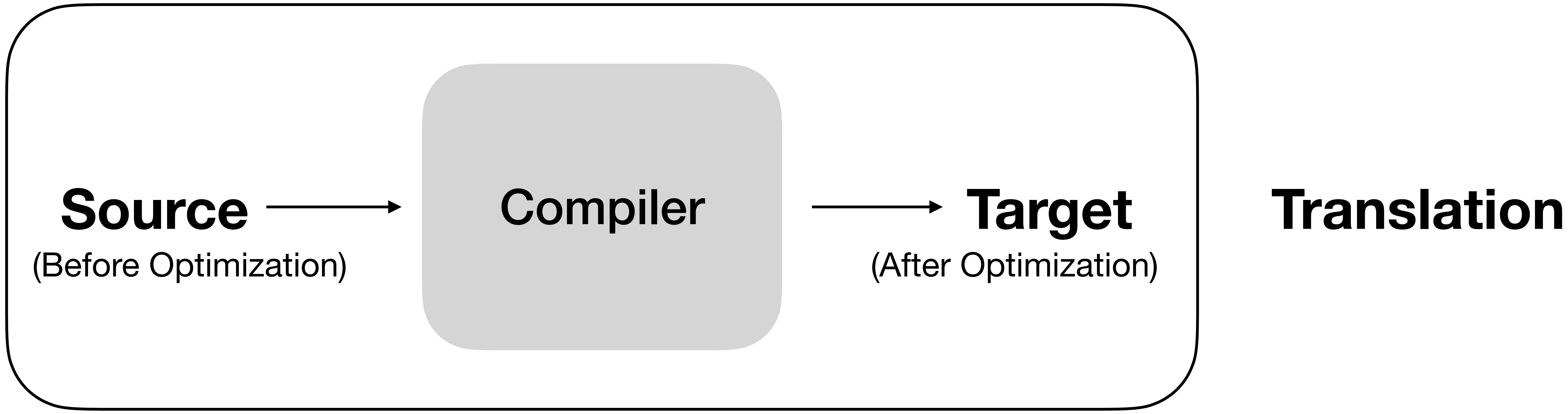
Challenge: TurboFan’s Complexity

- More than **100k** lines of code
- Frequent code changes.
- Complex internal structure

Hard to verify compiler in real-world!

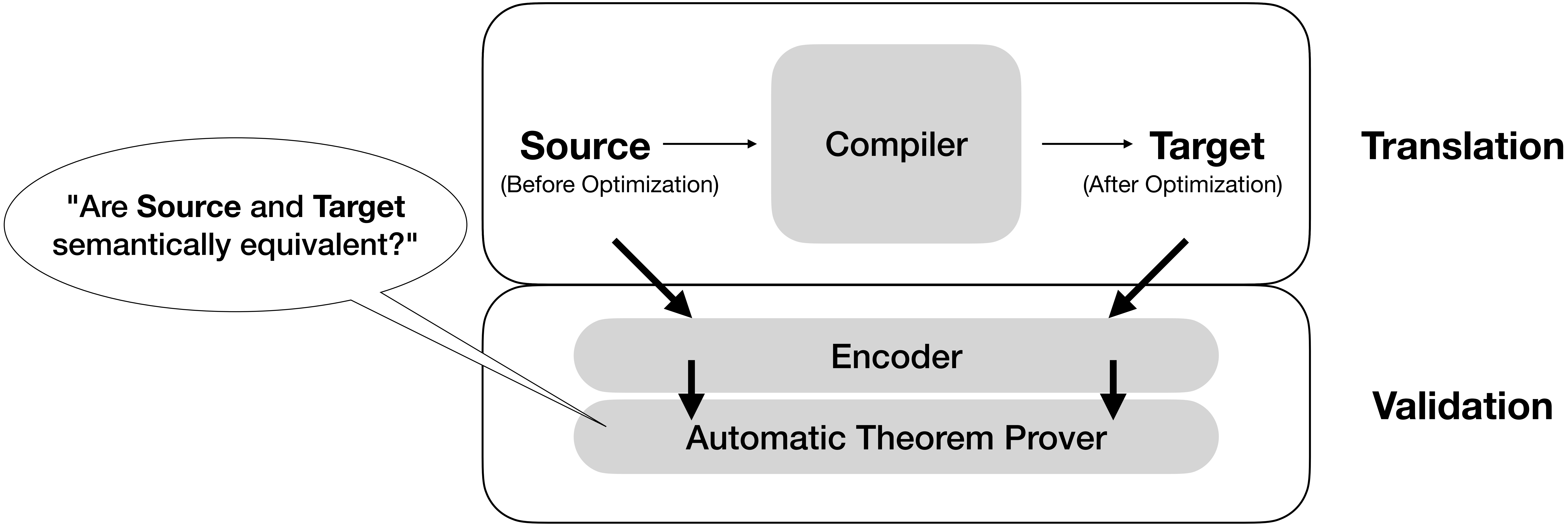
Our Solution: Translation Validation (TV)

- Check if translation preserves the program’s semantics
- Agnostic to compiler's implementation, focusing only on language’s semantics



Our Solution: Translation Validation (TV)

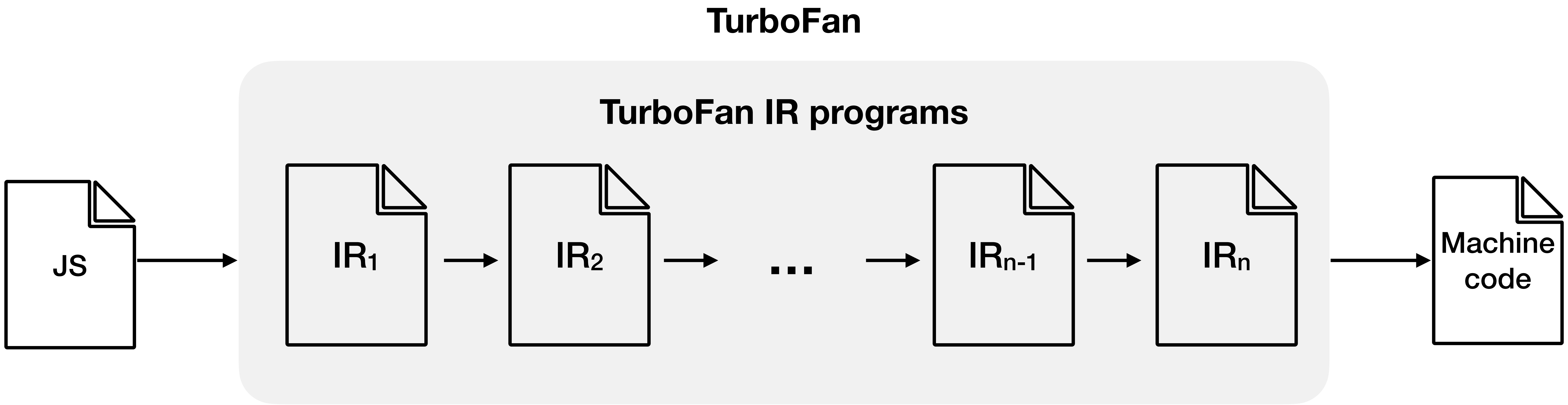
- Check if translation preserves the program's semantics
- Agnostic to compiler's implementation, focusing only on language's semantics



TurboTV: Translation Validator for TurboFan

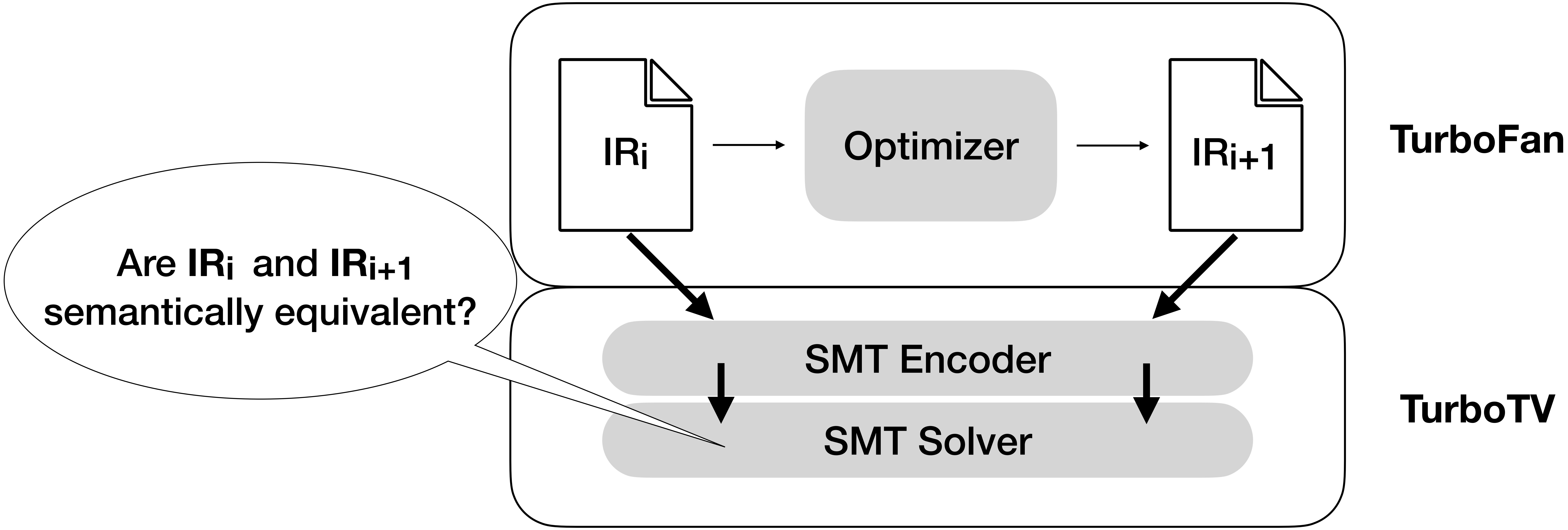
TurboFan's Optimization

- Translate JS into Intermediate Representation (IR); optimize IR
- **Correctness:** Program semantics must be preserved



TurboTV

- Does TurboFan's optimization preserve the semantics?



Our Contributions

Accuracy

Reproduced **8** recent bugs with
0 false positives

Our Contributions

Accuracy

Reproduced **8** recent bugs with **0** false positives

Scalability

Validated 90% of IR within 1 second
& Only 36% overhead as fuzzing oracle

Our Contributions

Accuracy

Reproduced **8** recent bugs with
0 false positives

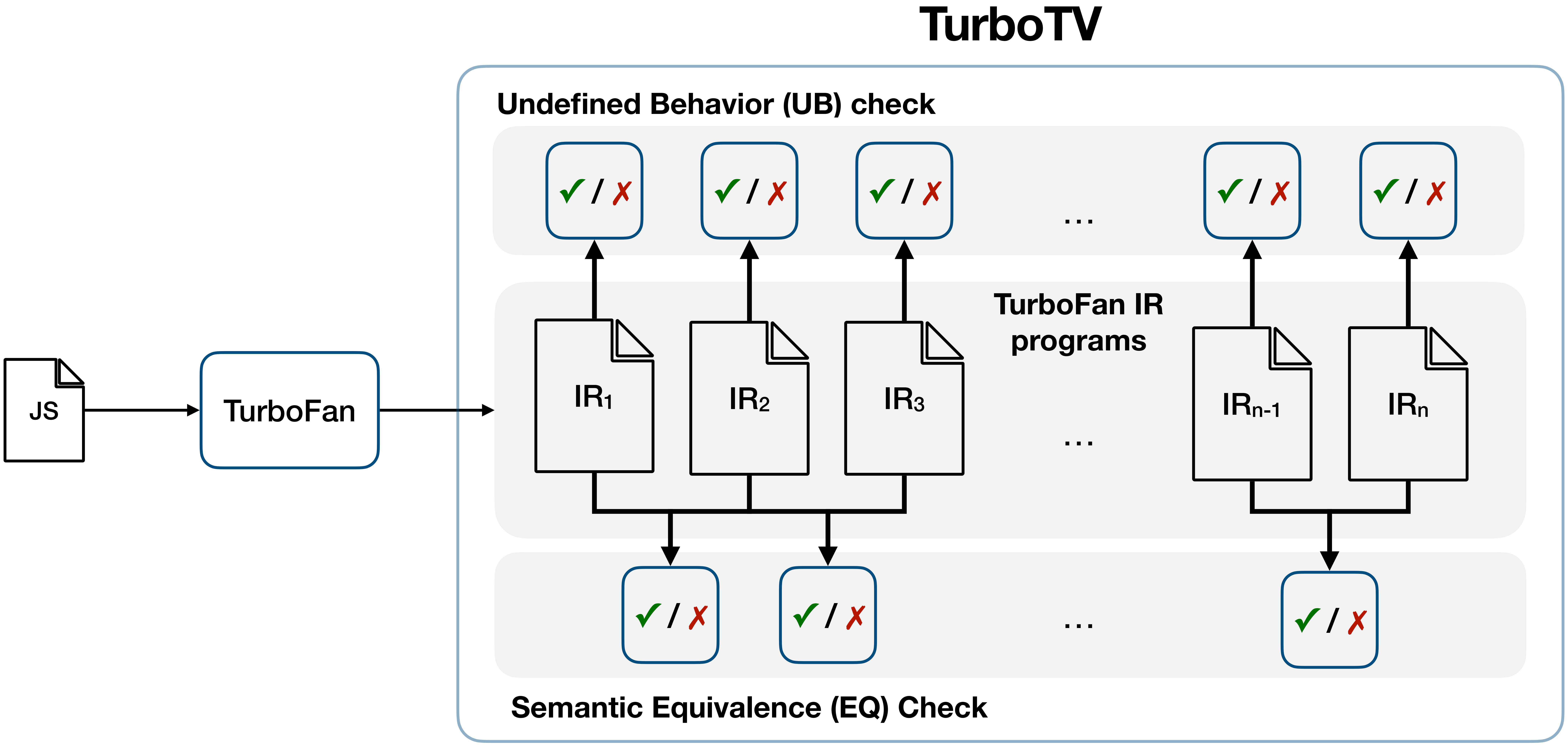
Scalability

Validated 90% of IR within 1 second
& Only 36% overhead as fuzzing oracle

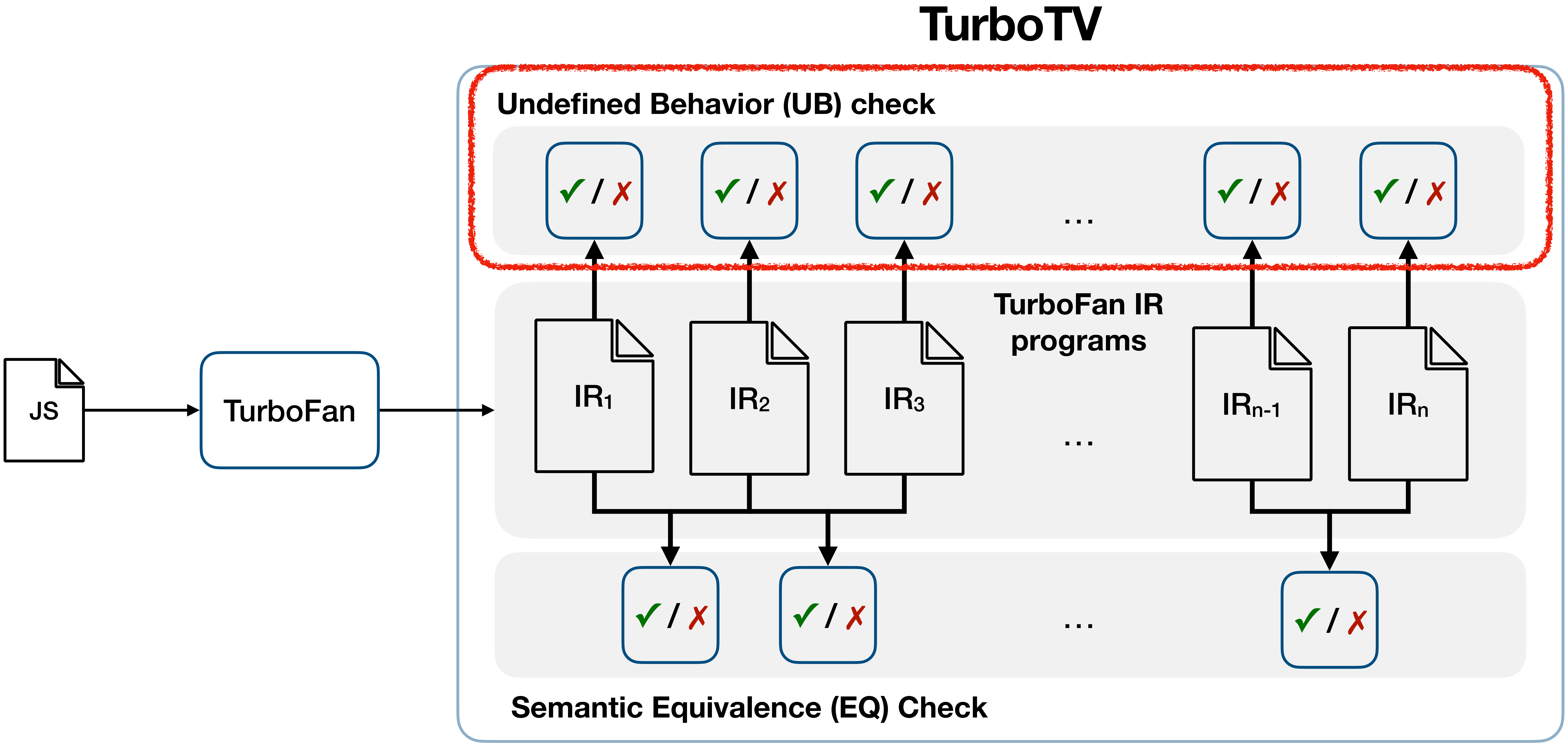
Utilization

Found a new bug in LLVM
in collaboration with another TV

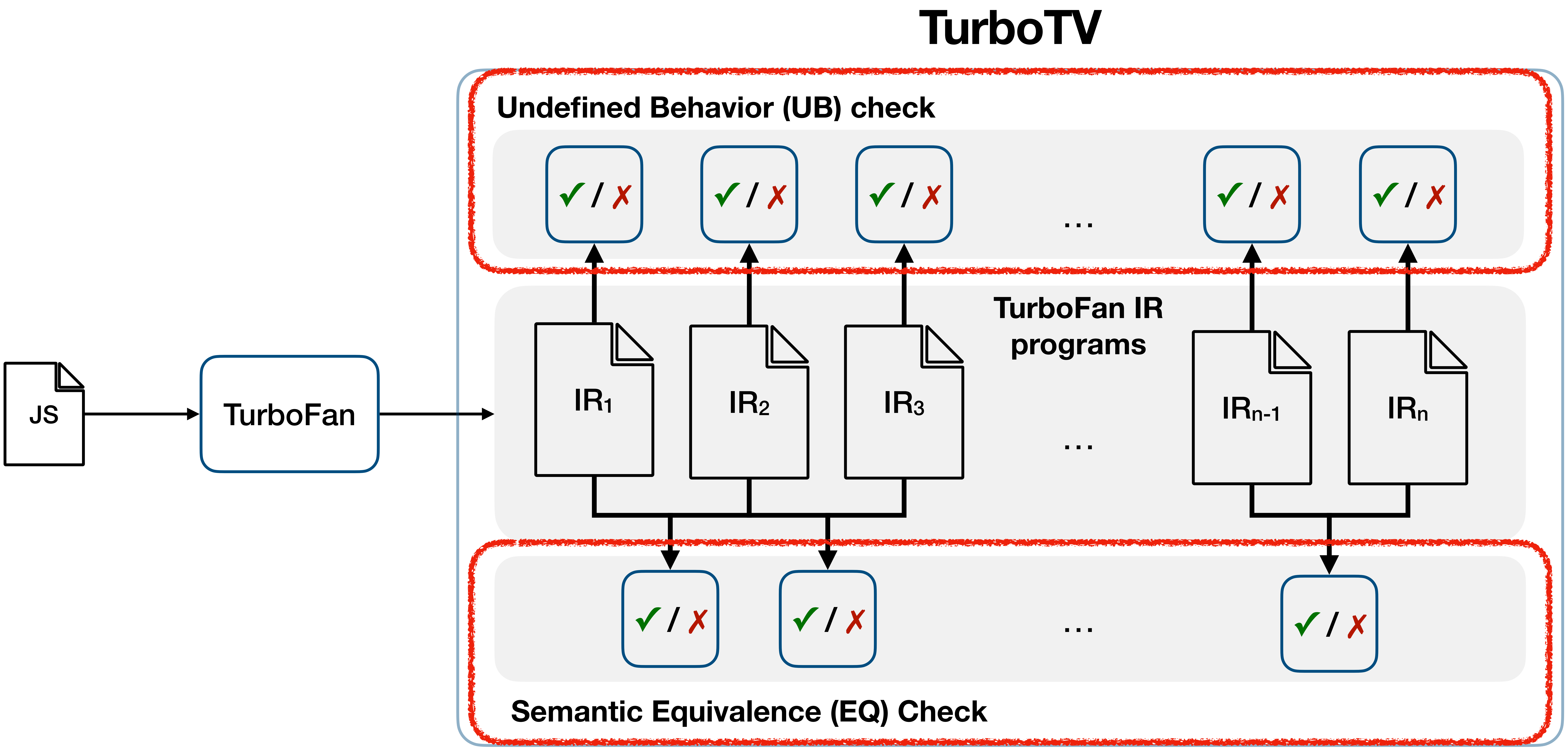
TurboTV: Overall Process



TurboTV: Overall Process

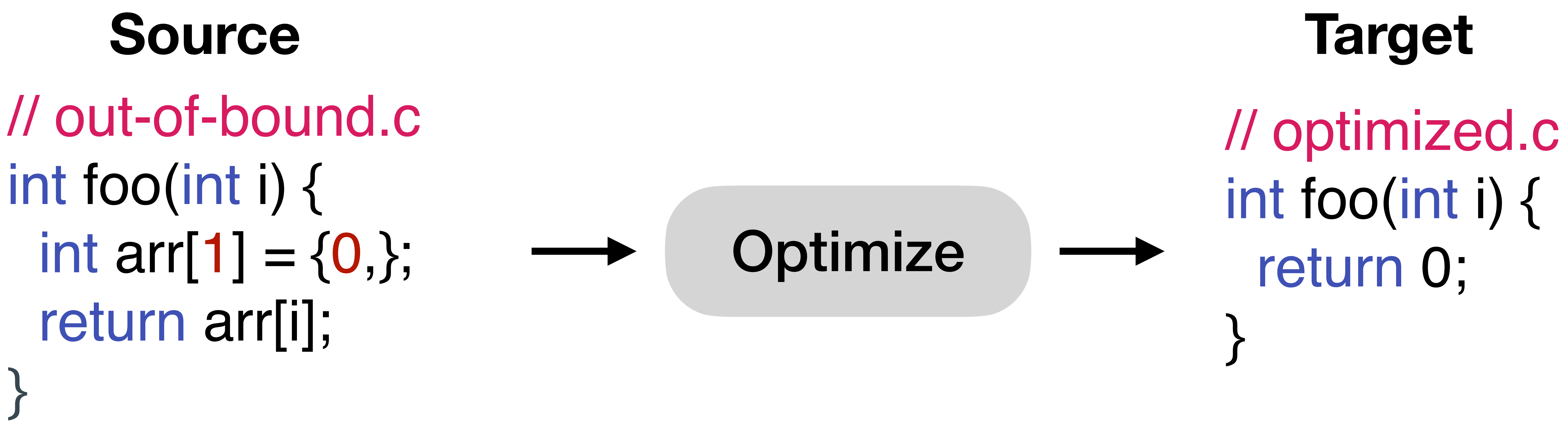


TurboTV: Overall Process


TurboTV
Undefined Behavior (UB) check
✓ / ✗
…
TurboFan IR programs
JS
TurboFan
IR₁
IR₂
IR₃
IRₙ₋₁
IRₙ
Semantic Equivalence (EQ) Check

Traditional TV for Languages with UB

- **Undefined Behavior (UB)** is a major reason of increased complexity (e.g., C/C++)



Source

```
// out-of-bound.c
int foo(int i) {
  int arr[1] = {0,};
  return arr[i];
}
```

Target

```
// optimized.c
int foo(int i) {
  return 0;
}
```

A. Pnueli et al. 1998. Translation Validation. In TACAS

Traditional TV for Languages with UB

- **Undefined Behavior (UB)** is a major reason of increased complexity (e.g., C/C++)

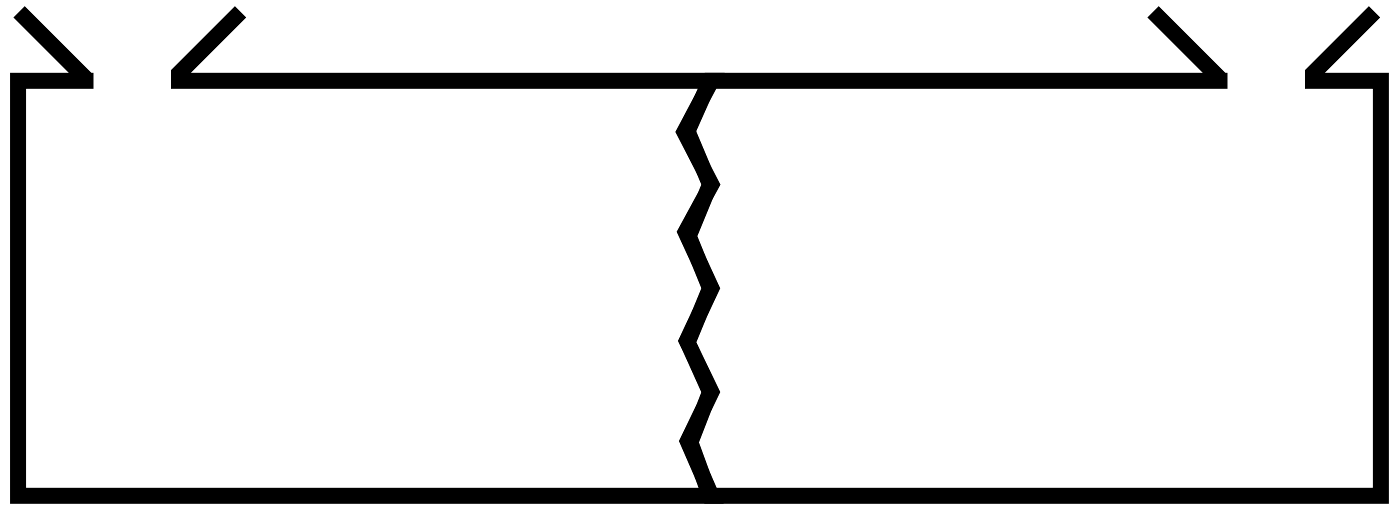
Source

```
// out-of-bound.c
int foo(int i) {
  int arr[1] = {0,};
  return arr[i];
}
```

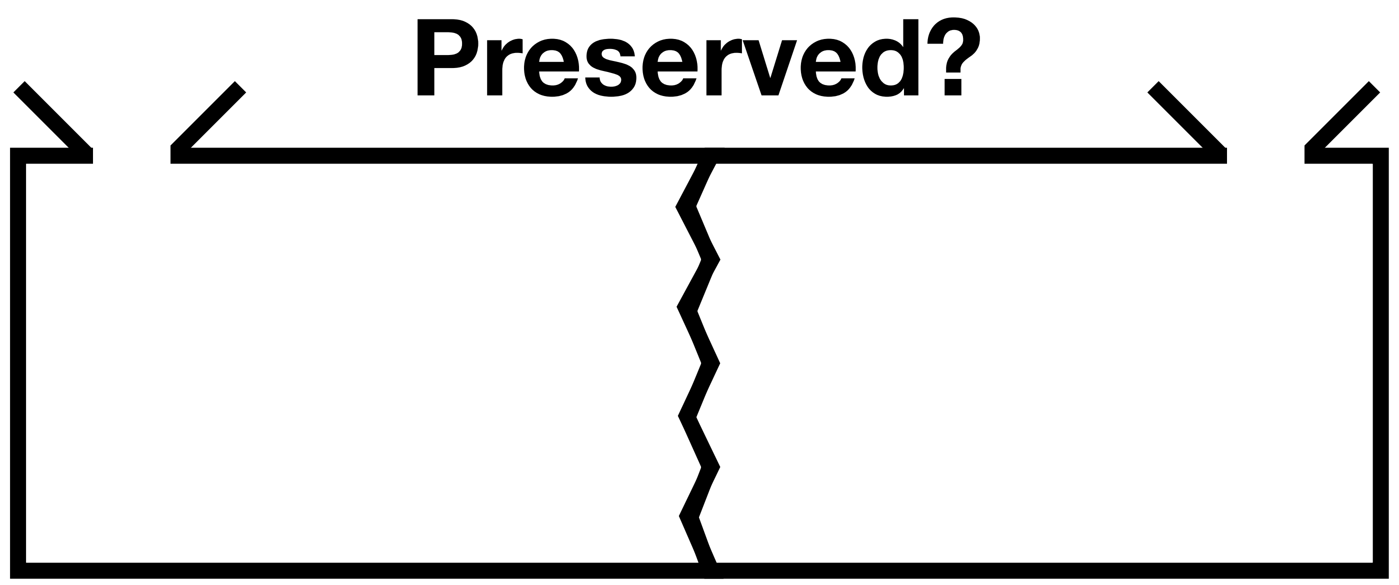
→ Optimize →

Target

```
// optimized.c
int foo(int i) {
  return 0;
}
```



Source’s Semantic



Target’s Semantic

A. Pnueli et al. 1998. Translation Validation. In TACAS

Traditional TV for Languages with UB

- **Undefined Behavior (UB)** is a major reason of increased complexity (e.g., C/C++)

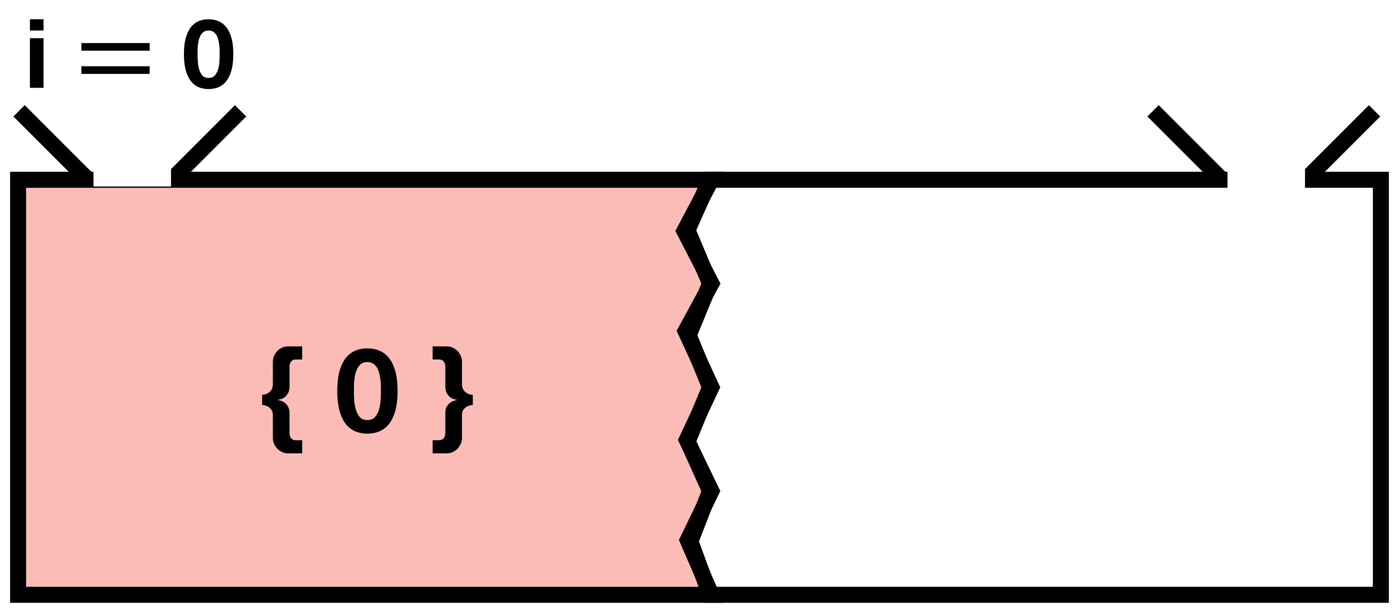
Source

```
// out-of-bound.c
int foo(int i) {
  int arr[1] = {0,};
  return arr[i];
}
```

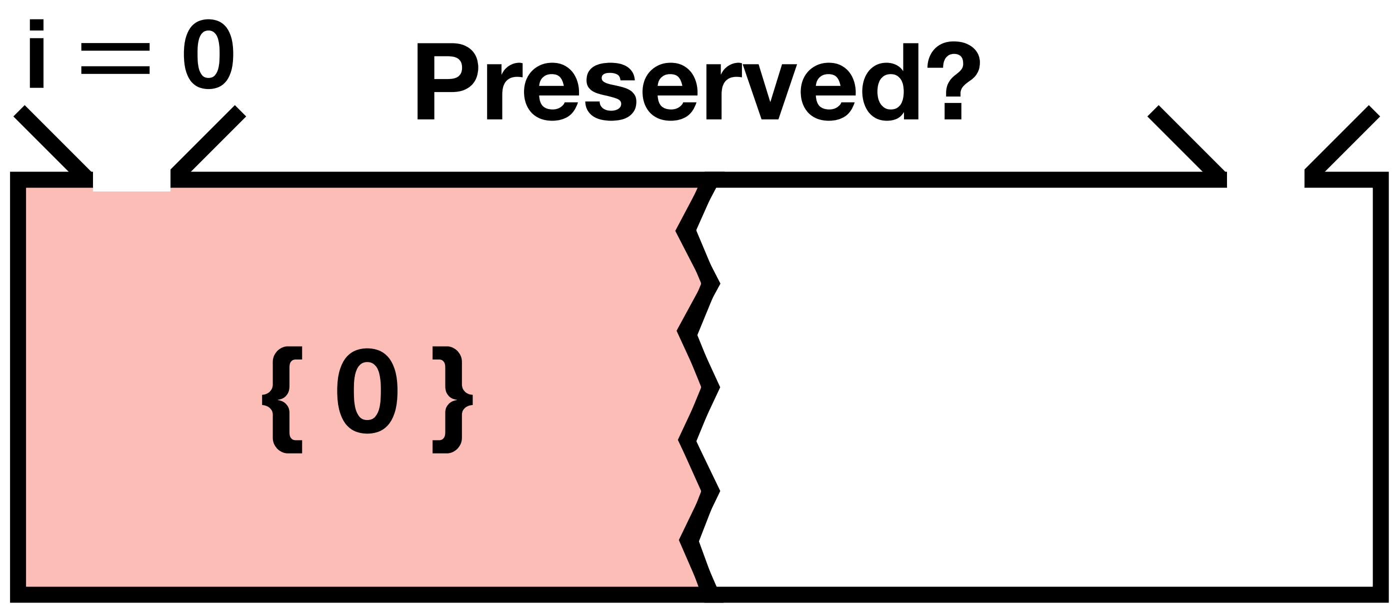
→ Optimize →

Target

```
// optimized.c
int foo(int i) {
  return 0;
}
```



Source's Semantic



Target's Semantic

A. Pnueli et al. 1998. Translation Validation. In TACAS

Traditional TV for Languages with UB

- **Undefined Behavior (UB)** is a major reason of increased complexity (e.g., C/C++)

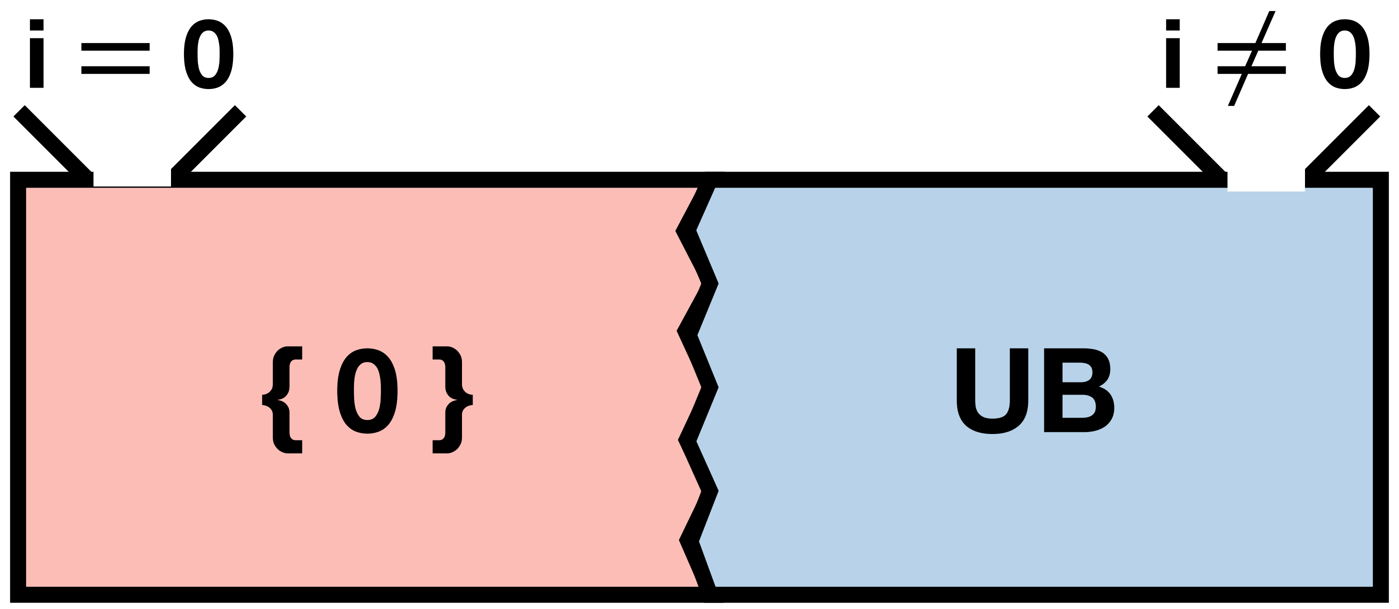
Source

```
// out-of-bound.c
int foo(int i) {
  int arr[1] = {0,};
  return arr[i];
}
```

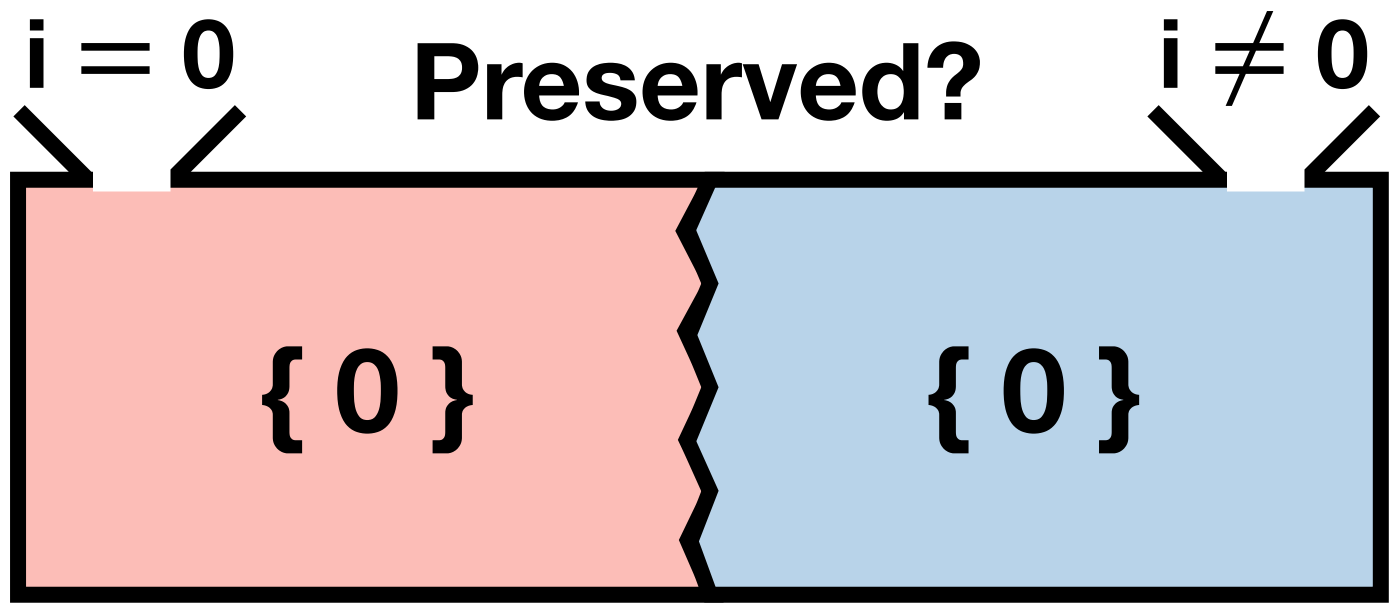
→ Optimize →

Target

```
// optimized.c
int foo(int i) {
  return 0;
}
```



Source's Semantic



😓

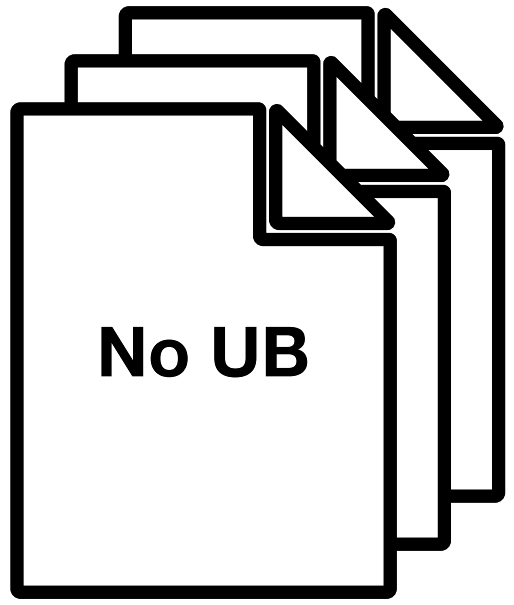
Target's Semantic

A. Pnueli et al. 1998. Translation Validation. In TACAS

TurboFan IR's UB

- If TurboFan is correct

According to JS specification*, UB does not exist in JS programs.

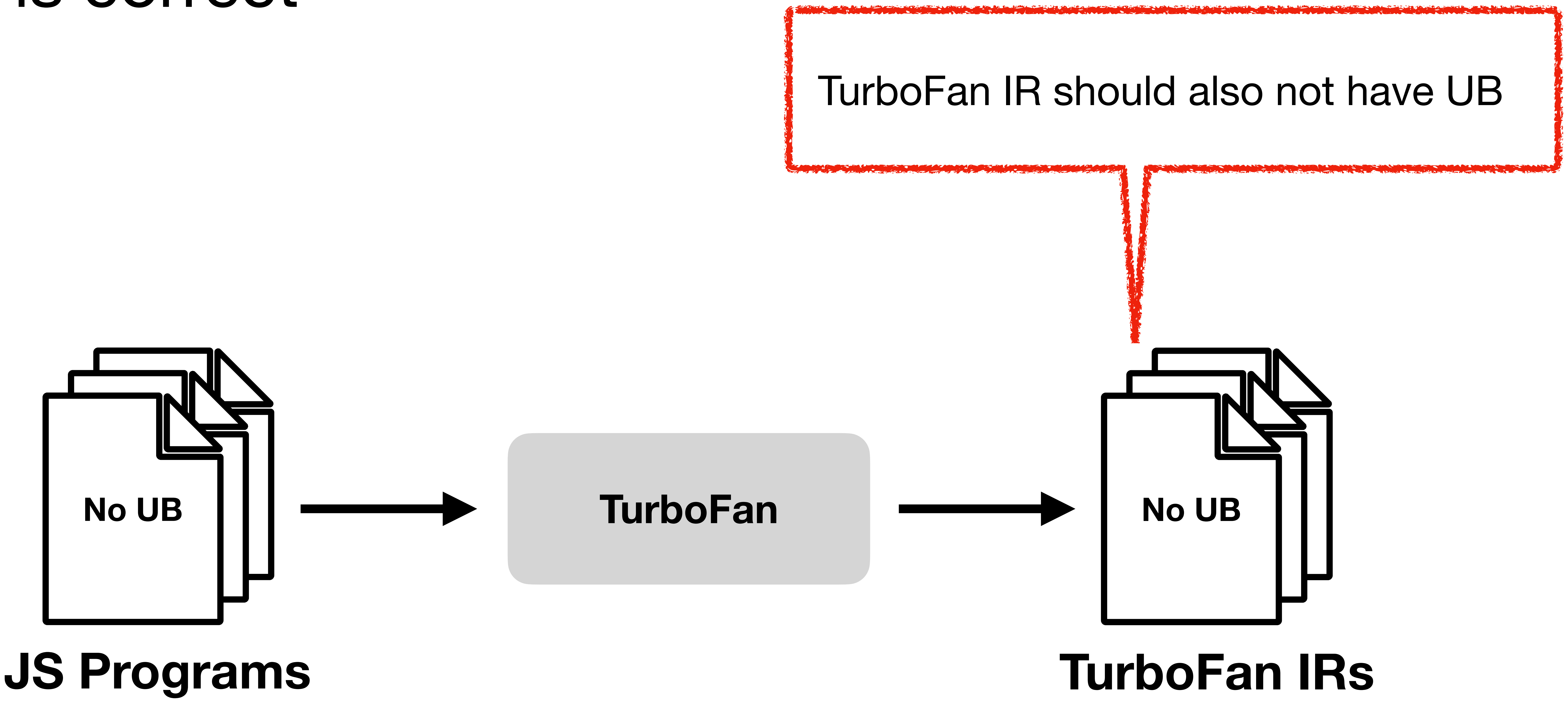


JS Programs

*https://ecma-international.org/publications-and-standards/standards/ecma-262/

TurboFan IR’s UB

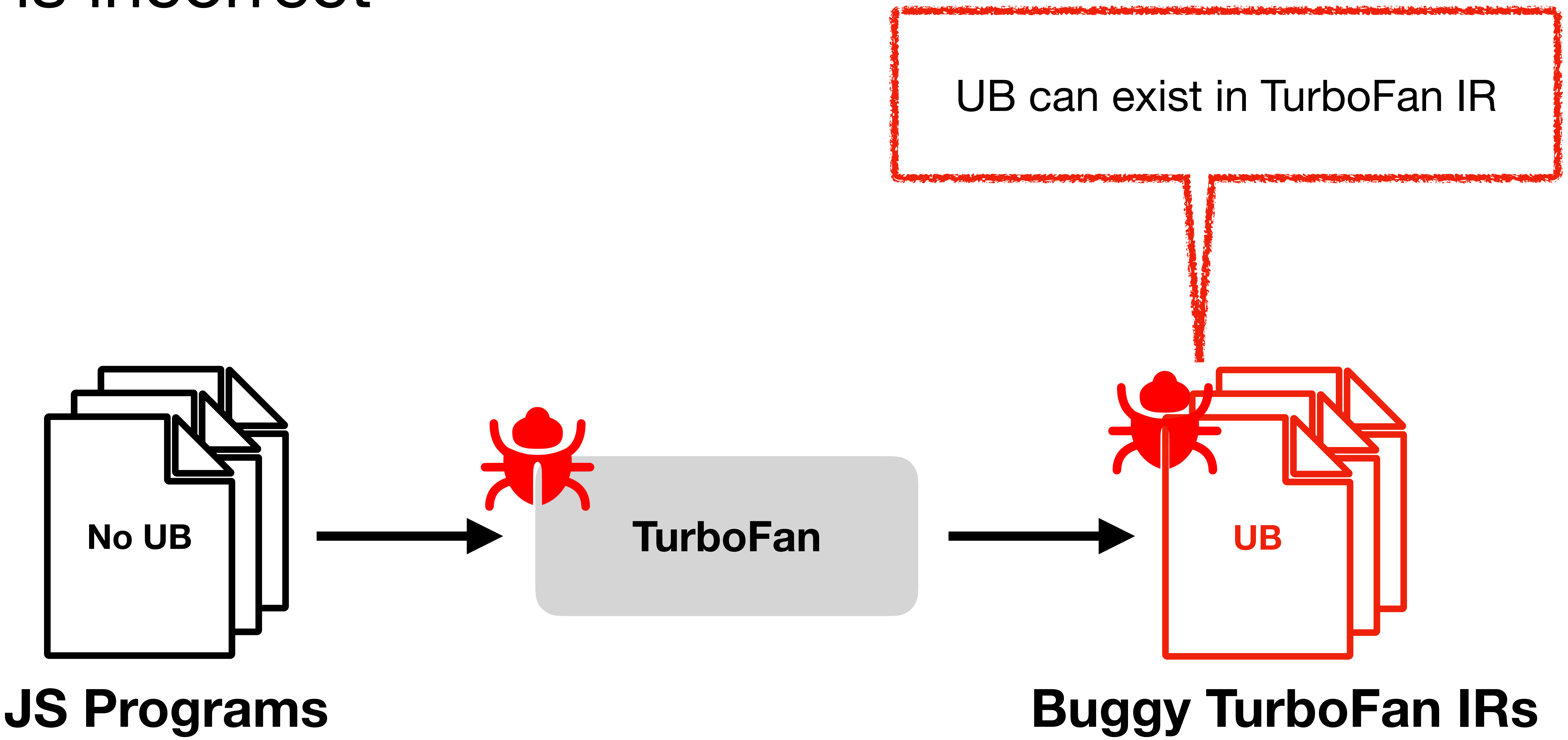
- If TurboFan is correct



*https://ecma-international.org/publications-and-standards/standards/ecma-262/

TurboFan IR’s UB

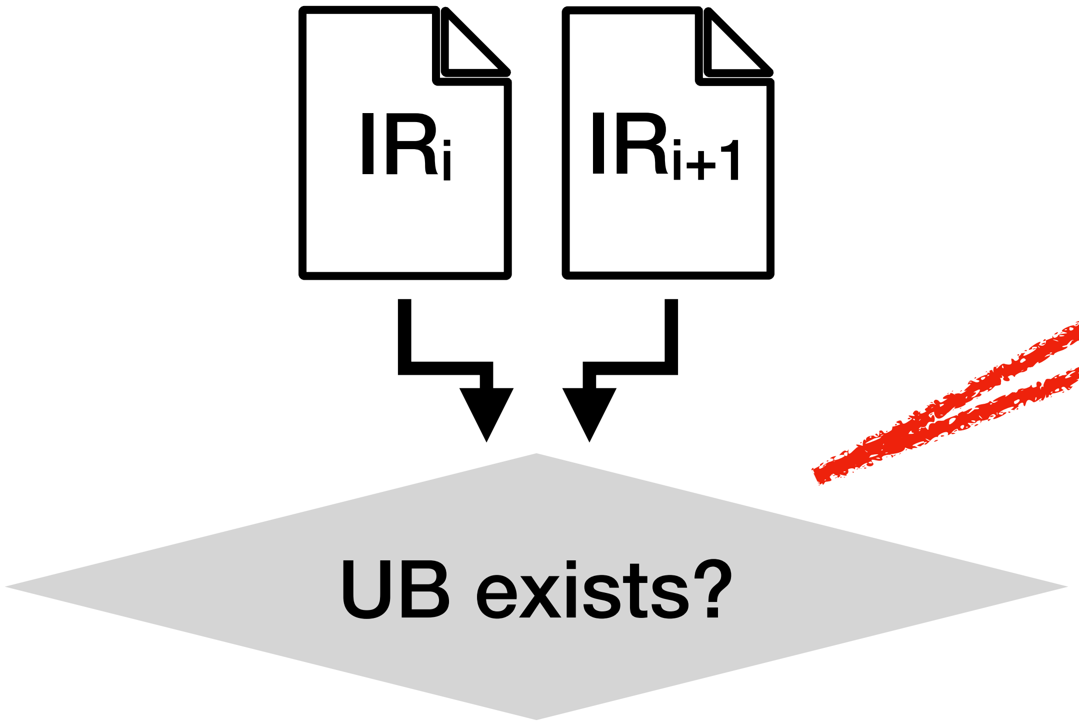
- If TurboFan is incorrect



*https://ecma-international.org/publications-and-standards/standards/ecma-262/

Two-Phase TV

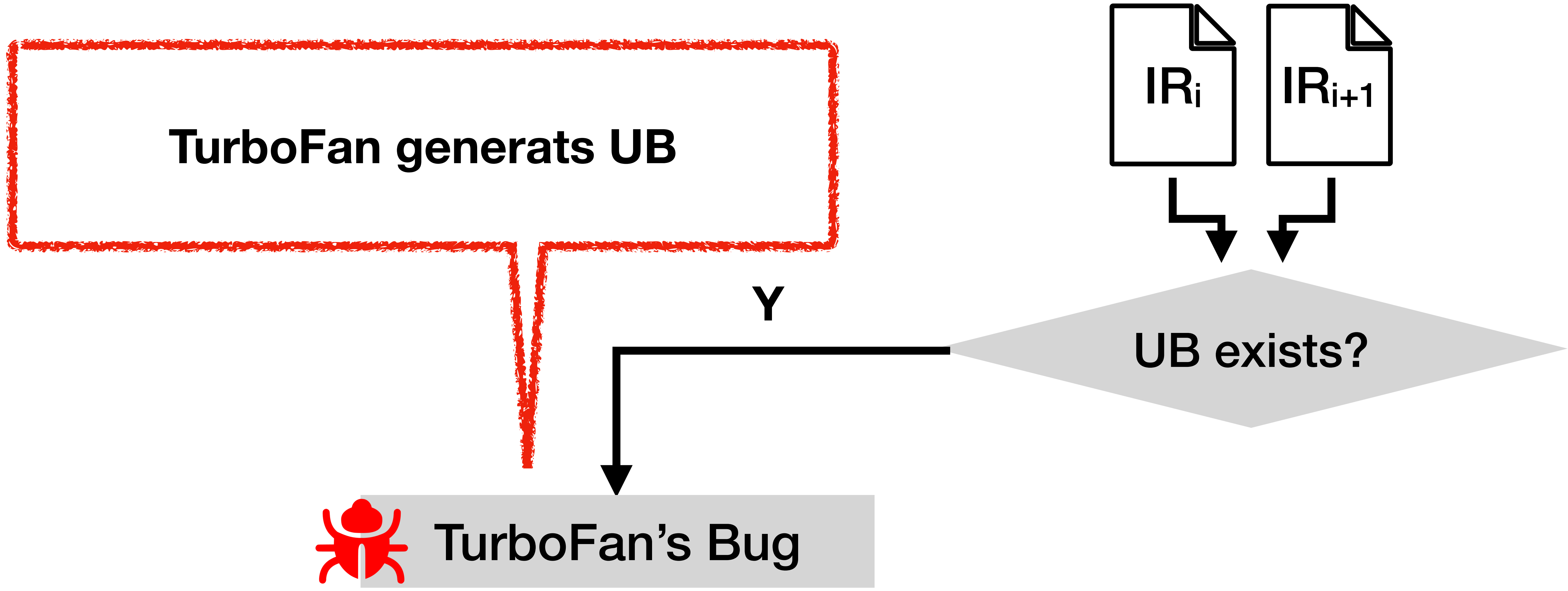
- Splitting TV process into two phases



Does each IR have a UB?

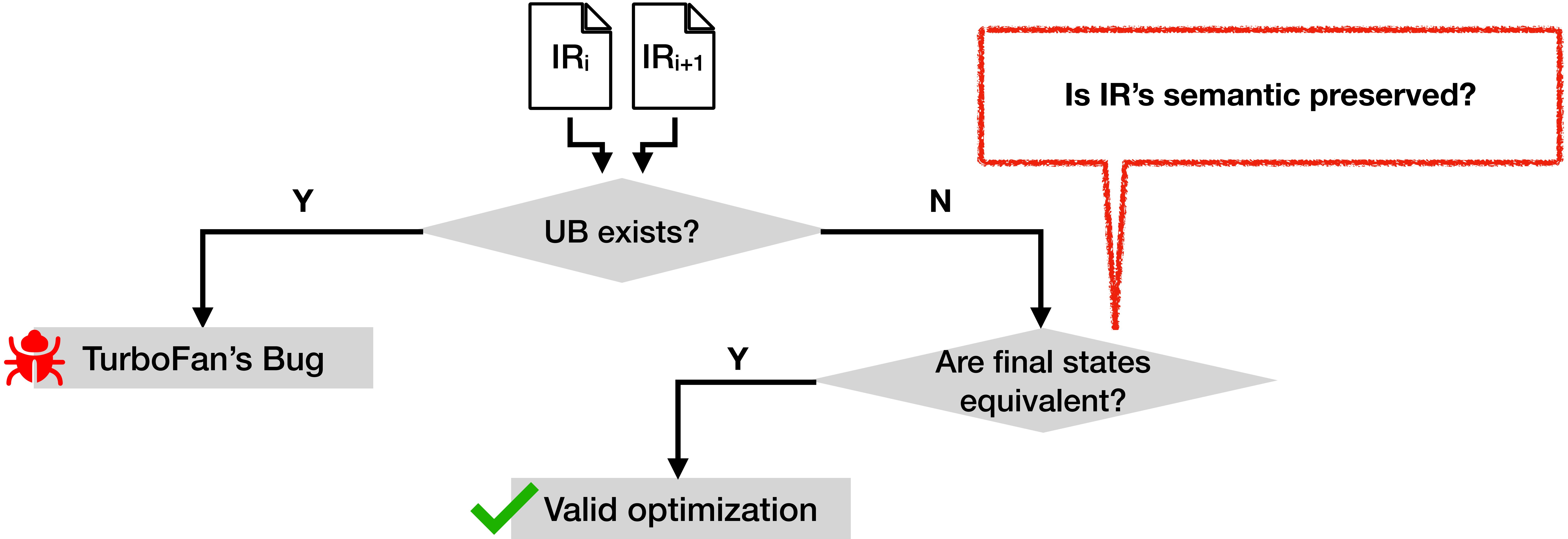
Two-Phase TV

- Splitting TV process into two phases



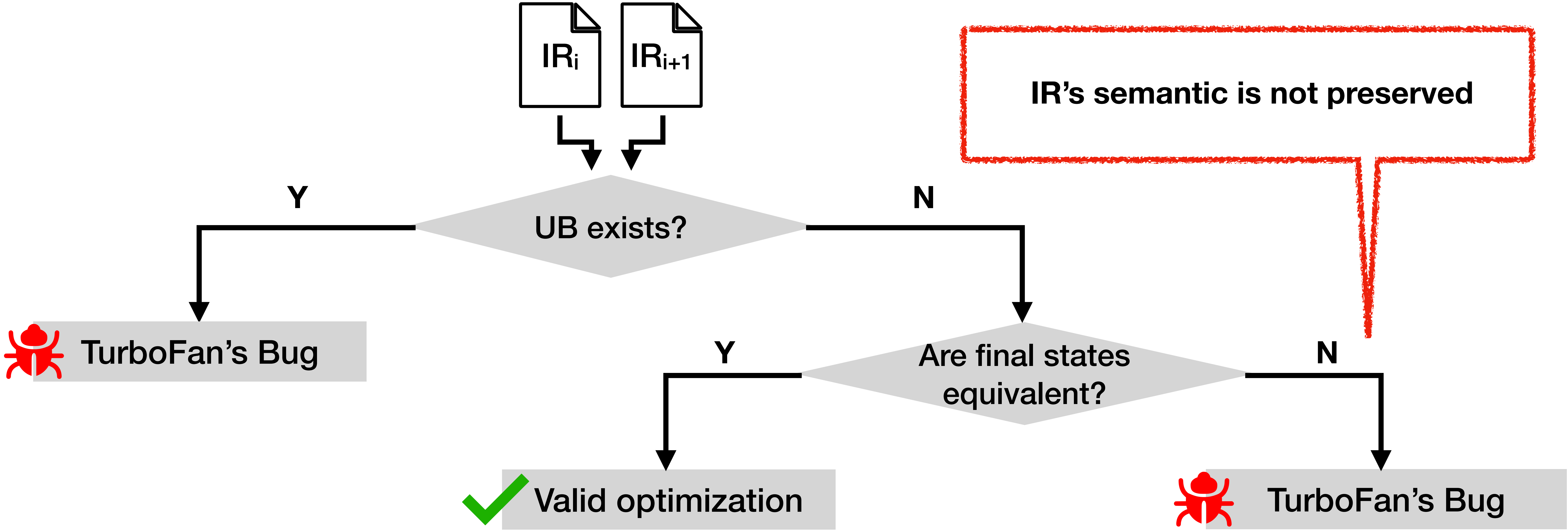
Two-Phase TV

- Splitting TV process into two phases



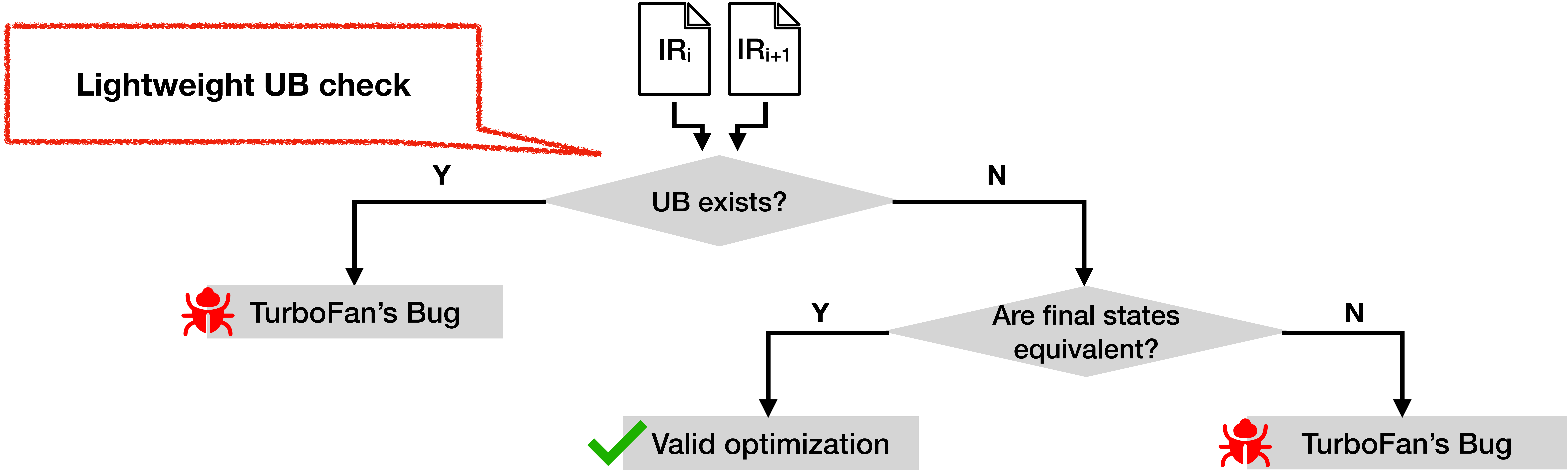
Two-Phase TV

- Splitting TV process into two phases



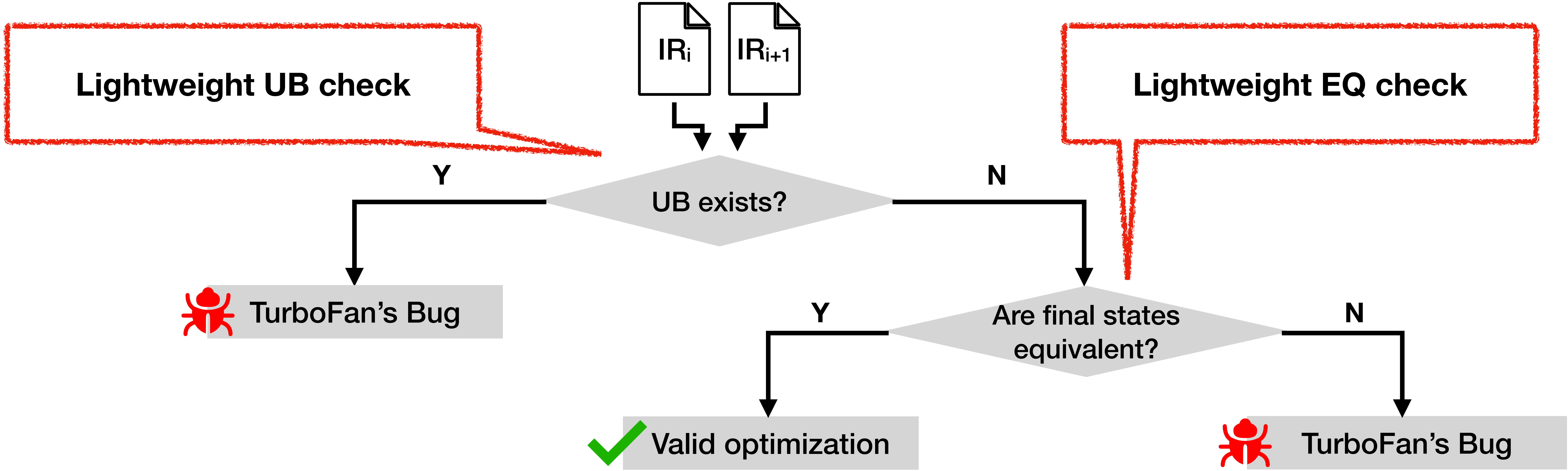
Two-Phase TV

- Splitting TV process into two phases



Two-Phase TV

- Splitting TV process into two phases



Evaluation

TurboTV's Accuracy

IR: The total number of IR programs
UB Checks: The number of UB check
EQ Checks: The number of EQ check
Timeout: 5 minutes

- Evaluate accuracy with 8 recently reported TurboFan optimization bugs

Bug ID	IR	UB Checks	EQ Checks	FP	Timeout	Result
1126249	20	20	19	0	0	✓
1195650	13	13	12	0	8	✓
1404607	33	33	32	0	0	✓
1198705	32	32	31	0	3	✓
1199345	13	13	12	0	0	✓
1200490	30	30	29	0	0	✓
1234764	19	19	18	0	5	✓
1234770	12	12	11	0	5	✓
1323114	11	11	10	0	0	✓

TurboTV’s Accuracy

IR: The total number of IR programs
UB Checks: The number of UB check
EQ Checks: The number of EQ check
Timeout: 5 minutes

- Evaluate accuracy with 8 recently reported TurboFan optimization bugs

Bug ID	IR	UB Checks	EQ Checks	FP	Timeout	Result
1126249	20	20	19	0	0	✓
1195650	13	13	12	0	8	✓
1404607	33	33	32	0	0	✓
1198705	32	32	31	0	3	✓
1199345	13	13	12	0	0	✓
1200490	30	30	29	0	0	✓
1234764	19	19	18	0	5	✓
1234770	12	12	11	0	5	✓
1323114	11	11	10	0	0	✓

IR: The total number of IR programs
UB Checks: The number of UB check
EQ Checks: The number of EQ check
Timeout: 5 minutes

TurboTV's Accuracy

- Evaluate accuracy with 8 recently reported TurboFan optimization bugs

Bug ID	IR	UB Checks	EQ Checks	FP	Timeout	Result
1126249	20	20	19	0	0	✓
1195650	13	13	12	0	8	✓
1404607	33	33	32	0	0	✓
1198705	32	32	31	0	3	✓
1199345	13	13	12	0	0	✓
1200490	30	30	29	0	0	✓
1234764	19	19	18	0	5	✓
1234770	12	12	11	0	5	✓
1323114	11	11	10	0	0	✓

UB Checker

EQ Checker

TurboTV’s Scalability

UnitTests: The number of TurboFan unit tests
Corpus: The number of large-scale JS files
IR: The total number of IR programs
Timeout: 5 minutes
Avg Time: Average validation time excluding timeouts

- Measuring scalability for TurboFan unit tests & large-scale JS files

Benchmarks	JS	IR	UB Checks	EQ Checks	FP	Timeout	Avg Time
UnitTests	576	4,387	4,387	4,386	41	328	2.61s
Corpus	196K	13,870	13,870	13,869	0	298	0.71s

TurboTV's Scalability

UnitTests: The number of TurboFan unit tests
Corpus: The number of large-scale JS files
IR: The total number of IR programs
Timeout: 5 minutes
Avg Time: Average validation time excluding timeouts

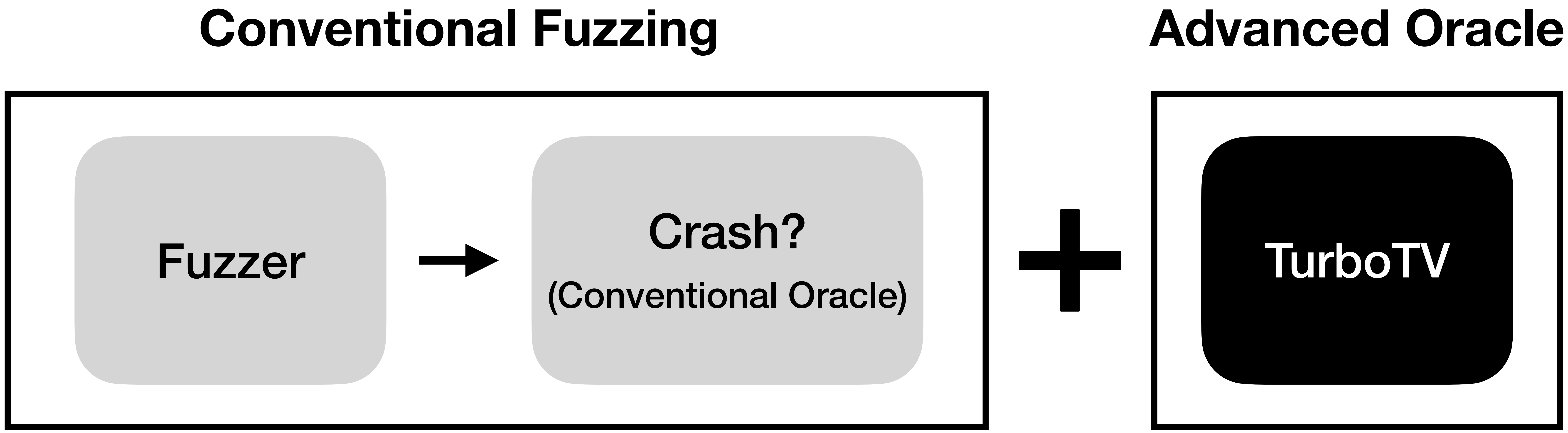
- Measuring scalability for TurboFan unit tests & large-scale JS files

Only 1% False Positives!

Benchmarks	**JS**	**IR**	**UB Checks**	**EQ Checks**	**FP**	**Timeout**	**Avg Time**
UnitTests	576	4,387	4,387	4,386	41	328	2.61s
Corpus	196K	13,870	13,870	13,869	0	298	0.71s

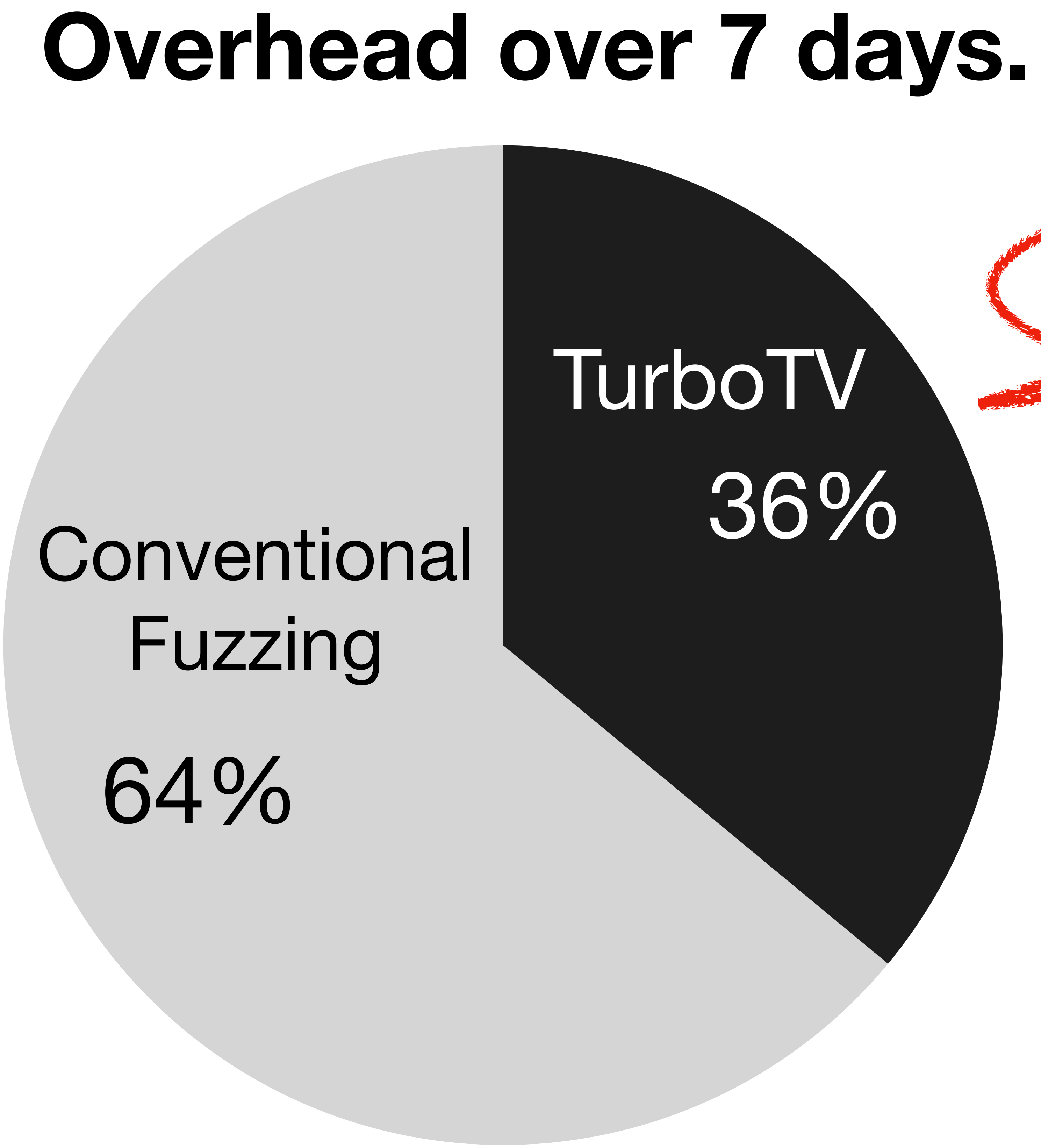
TurboTV as a Fuzzing Oracle

- TurboTV + Fuzzing: Detecting latent miscompilation bugs



TurboTV as a Fuzzing Oracle

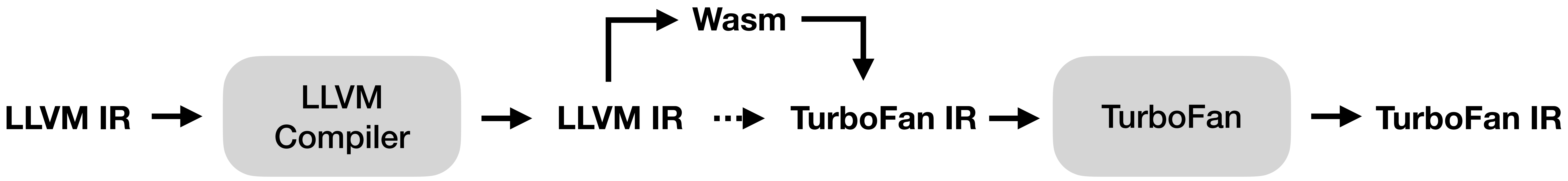
- TurboTV + Fuzzing: Detecting latent miscompilation bugs



Only 36% Overhead

Cross-language TV

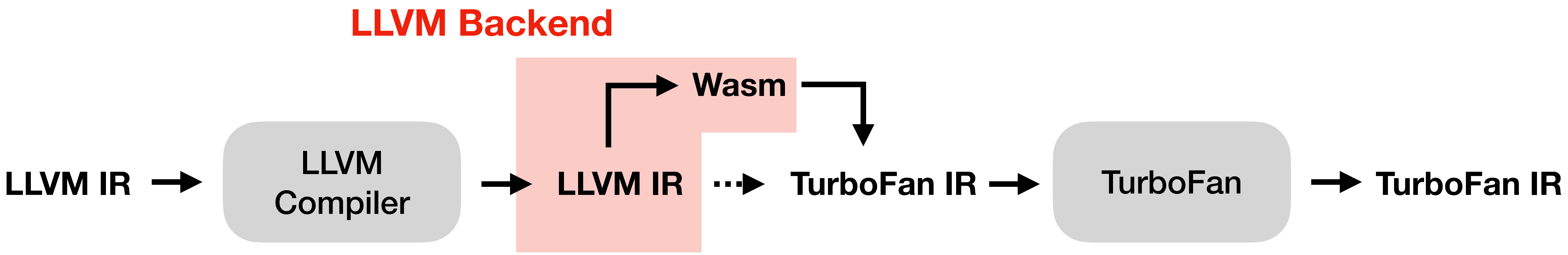
- TV across two compilers



*N.P.Lopes Et al . 2021. Alive2: Bounded Translation Validation for LLVM. PLDI 2021

Cross-language TV

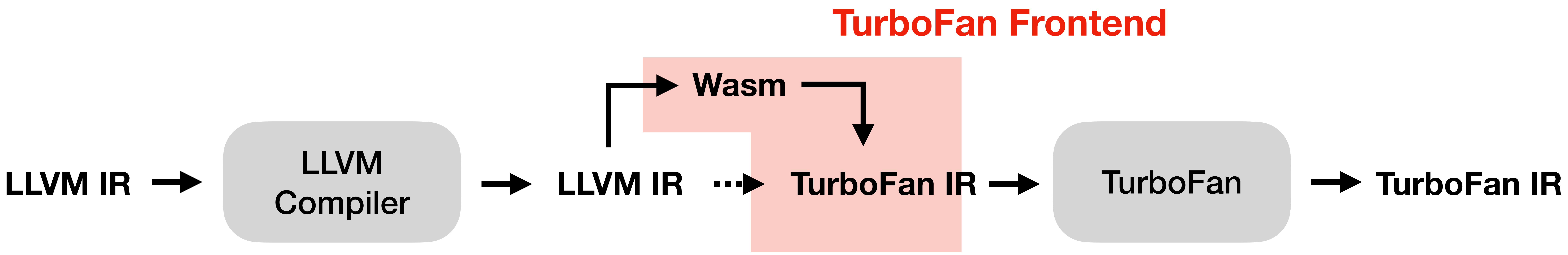
- TV across two compilers



*N.P.Lopes Et al . 2021. Alive2: Bounded Translation Validation for LLVM. PLDI 2021

Cross-language TV

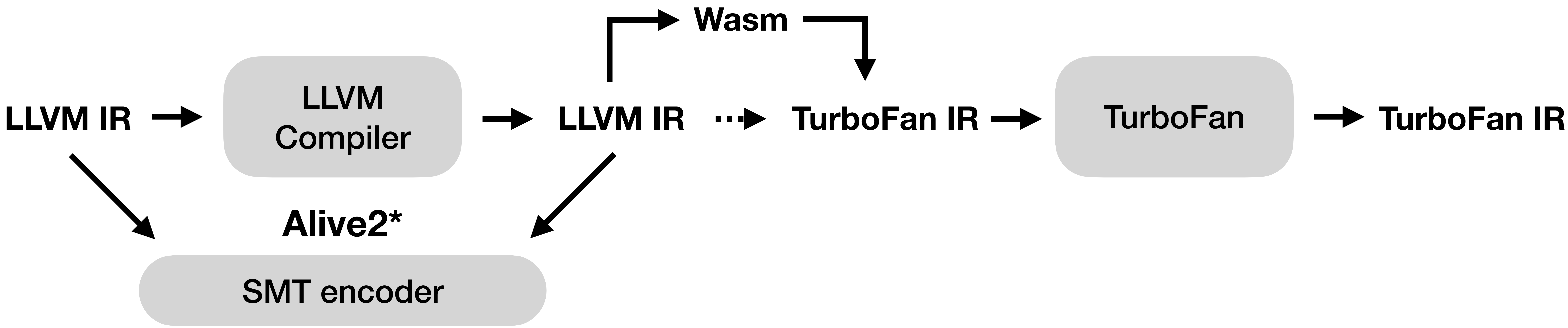
- TV across two compilers



*N.P.Lopes Et al . 2021. Alive2: Bounded Translation Validation for LLVM. PLDI 2021

Cross-language TV

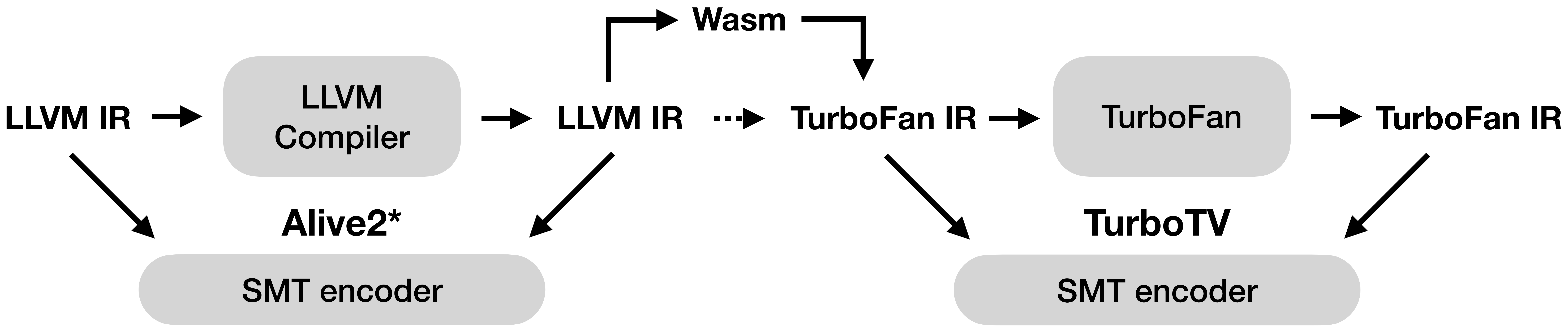
- TV across two compilers



*N.P.Lopes Et al . 2021. Alive2: Bounded Translation Validation for LLVM. PLDI 2021

Cross-language TV

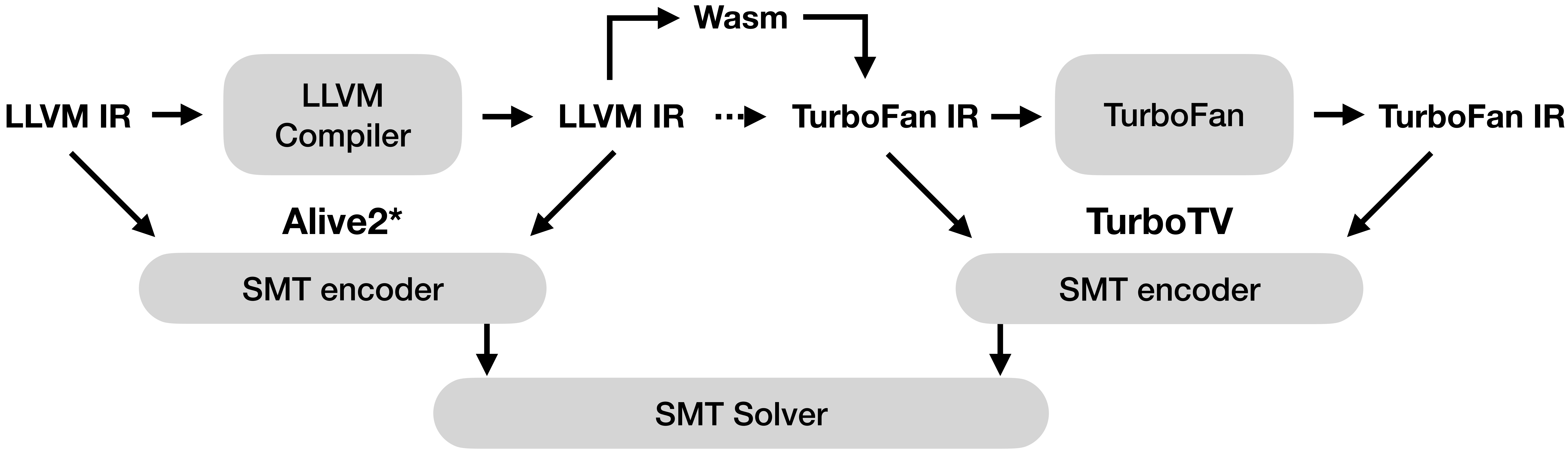
- TV across two compilers



*N.P.Lopes Et al . 2021. Alive2: Bounded Translation Validation for LLVM. PLDI 2021

Cross-language TV

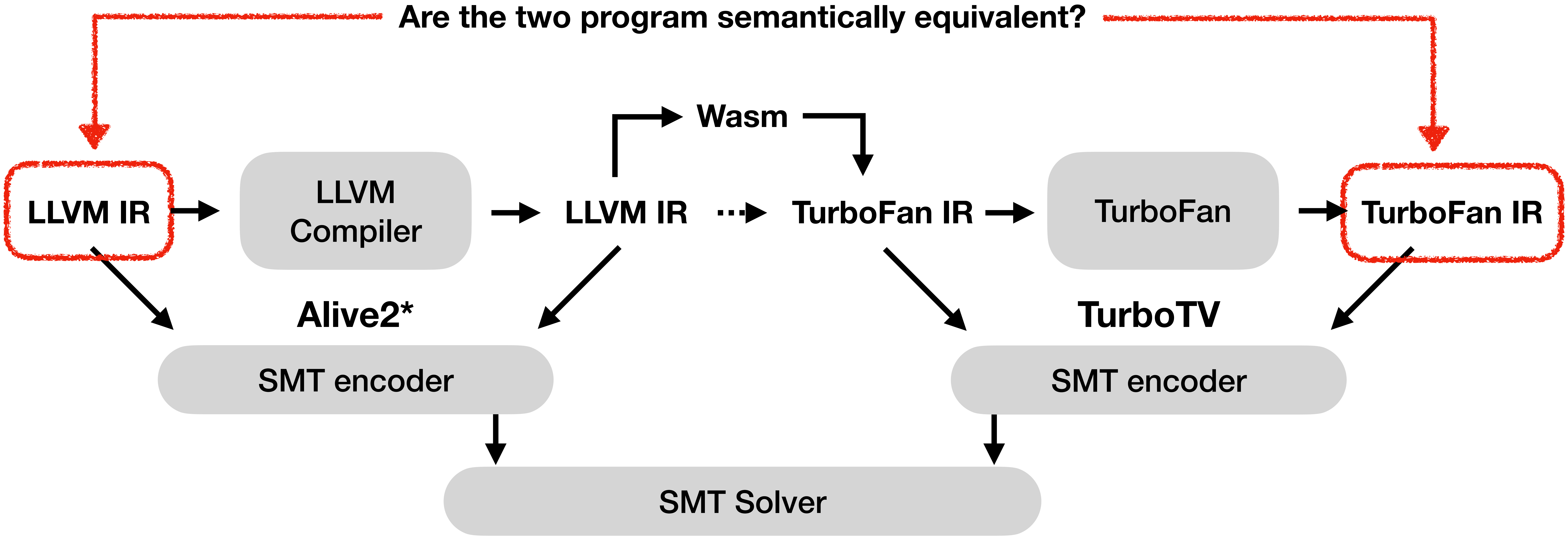
- TV across two compilers



*N.P.Lopes Et al . 2021. Alive2: Bounded Translation Validation for LLVM. PLDI 2021

Cross-language TV

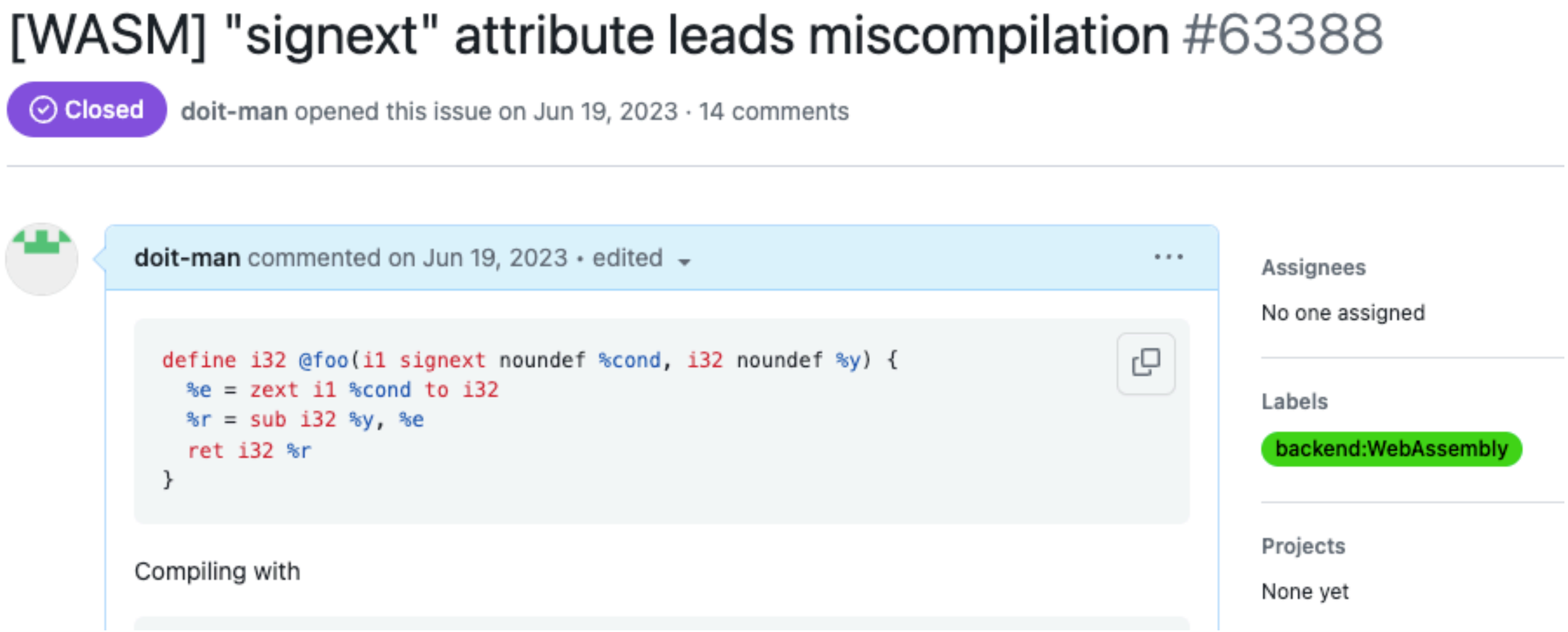
- TV across two compilers



*N.P.Lopes Et al . 2021. Alive2: Bounded Translation Validation for LLVM. PLDI 2021

Cross-language TV

- Bug discovered in LLVM's Wasm backend, reported, and patched



[WASM] "signext" attribute leads miscompilation #63388

Closed doit-man opened this issue on Jun 19, 2023 · 14 comments

doit-man commented on Jun 19, 2023 • edited

```
define i32 @foo(i1 signext noundef %cond, i32 noundef %y) {
  %e = zext i1 %cond to i32
  %r = sub i32 %y, %e
  ret i32 %r
}
```

Compiling with

Assignees

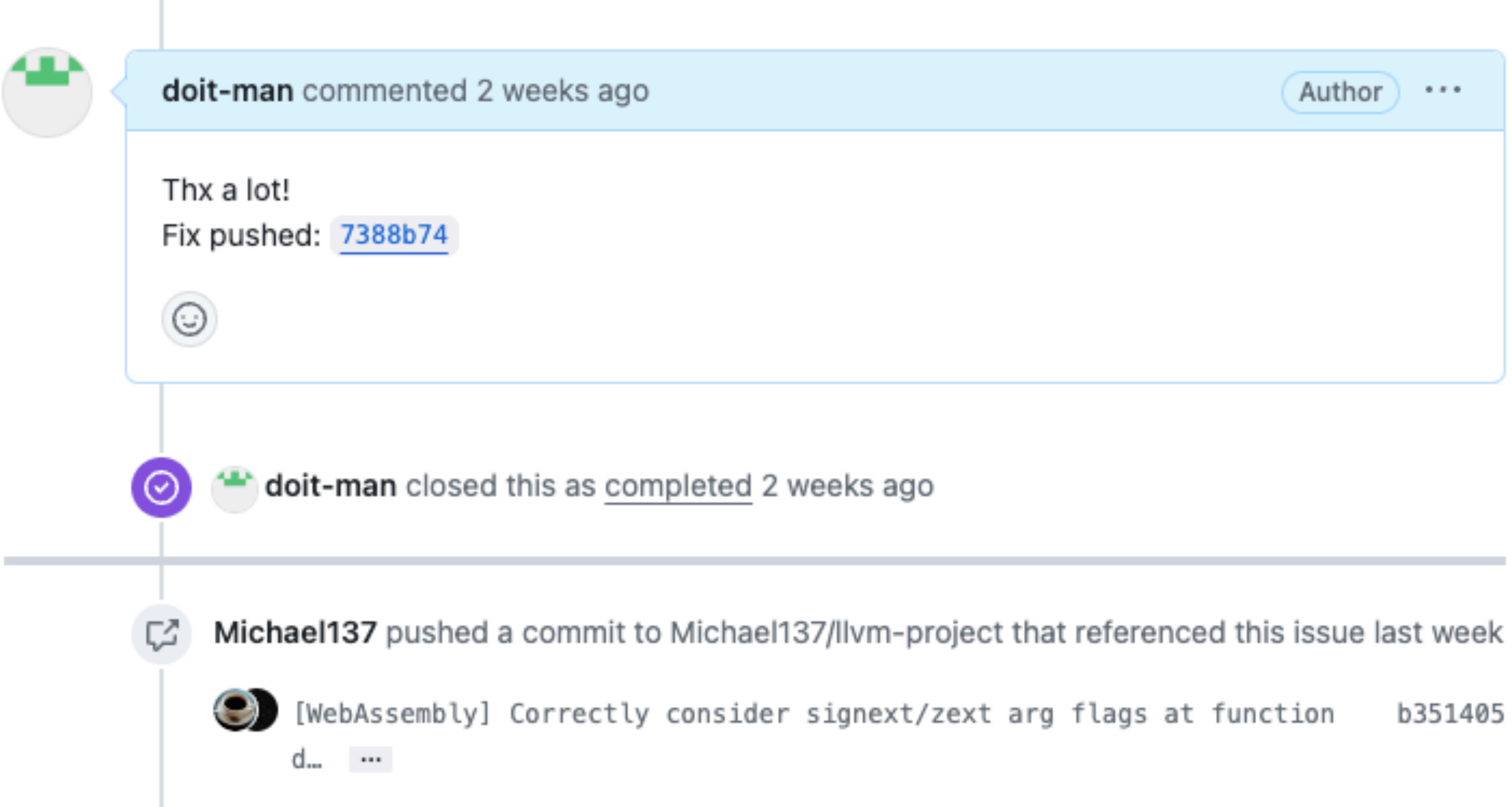
No one assigned

Labels

backend:WebAssembly

Projects

None yet



doit-man commented 2 weeks ago Author

Thx a lot!
Fix pushed: 7388b74

doit-man closed this as completed 2 weeks ago

Michael137 pushed a commit to Michael137/llvm-project that referenced this issue last week

`[WebAssembly] Correctly consider signext/zext arg flags at function d…` b351405

https://github.com/llvm/llvm-project/issues/63388

More details are in the paper

Detailed Examples

TurboTV Details

More Evaluation

Future Works

Conclusion

Our Webpage

- Translation validator for TurboFan
  - Validate optimizations agnostic of TurboFan’s internals

Conclusion

Our Webpage

- Translation validator for TurboFan
  - Validate optimizations agnostic of TurboFan's internals
- High performance & Low cost through **two-phase TV**
  - Applicable as fuzzing oracle

Conclusion

Our Webpage

- Translation validator for TurboFan
  - Validate optimizations agnostic of TurboFan's internals
- High performance & Low cost through **two-phase TV**
  - Applicable as fuzzing oracle
- **Cross-language TV**: Combined two translation validators
  - Discovered a new bug in LLVM Wasm backend

Conclusion

Our Webpage

- Translation validator for TurboFan
  - Validate optimizations agnostic of TurboFan’s internals
- High performance & Low cost through **two-phase TV**
  - Applicable as fuzzing oracle
- **Cross-language TV**: Combined two translation validators
  - Discovered a new bug in LLVM Wasm backend
- Our Webpage: https://prosys.kaist.ac.kr/turbo-tv

Appendix

UB Criteria

- **Since TurboFan IR has no standards, UBs are selected based on the following criteria:**
    - Gather representative cases from past TurboFan security bugs.
    - Cross-check with LLVM's UB classification.
    - Experiment with our UB checker's effectiveness in detecting TurboFan's incorrect behavior.

The UB checked by TurboTV

- **Out-of-bound (Undefined) Memory** Access
- reachable to **Unreachable**
    - **Unreachable:** Inserted at points TurboFan's analysis marks as unreachable
- When operators with **incorrect types** exist

Future Works

- **Current: For single Loop-free functions (intraprocedural)**
- **Future Works**:
  - Interprocedural: Encode the meaning of each function and check correct function invocation.
  - Functions with Loop : Loop invariant synthesis, loop unrolling

TurboTV's False Positives

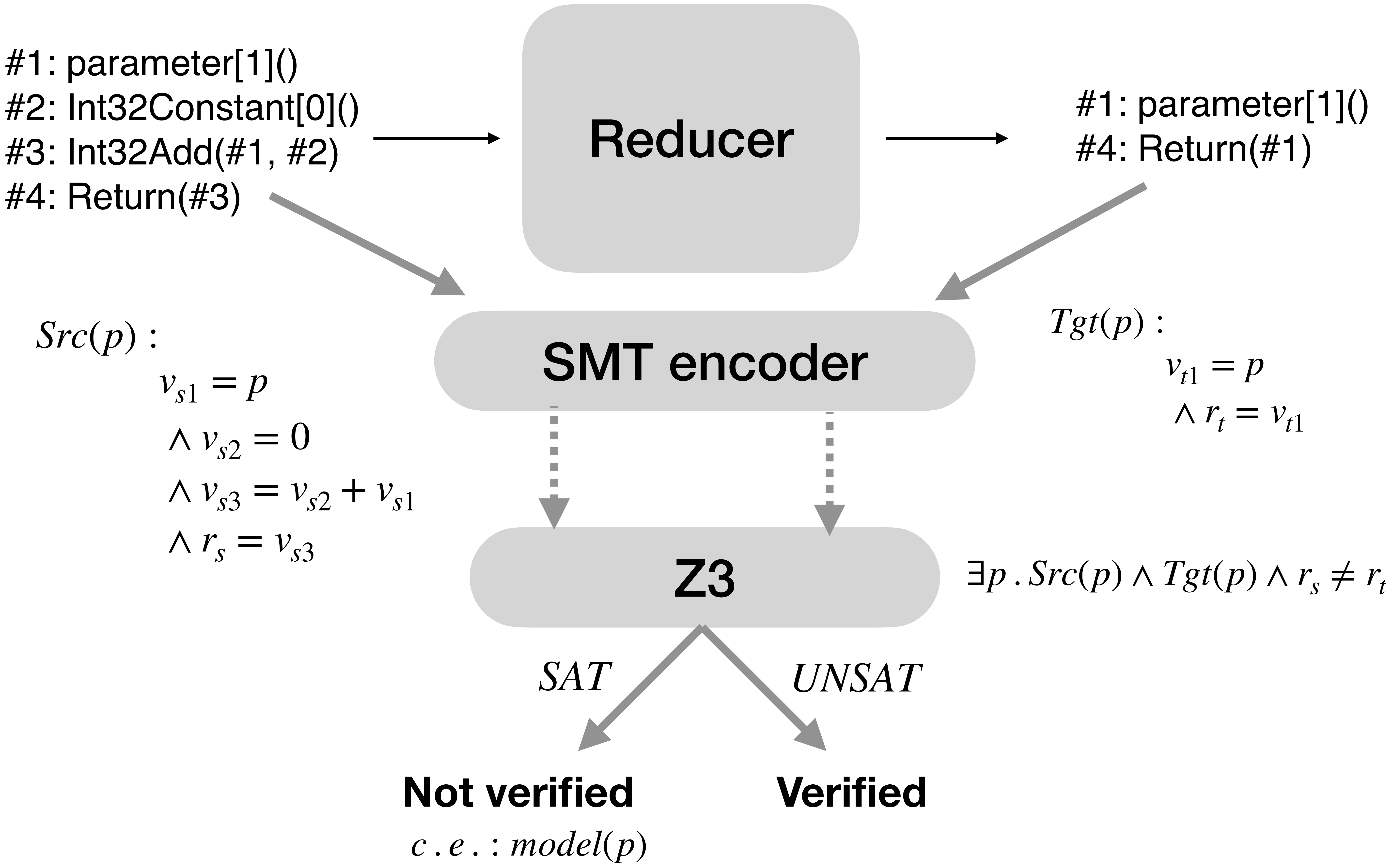
- **Interprocedural Optimizations**
  - TurboTV falsely report after interprocedural optimizations (Our scope is intraprocedural optimizations)

- **Uninterpreted Functions (UF)**
  - We assume that all function calls do not update the memory

Translation Validation

Example

```
function foo(x) {
    return x+0
}
```



```
#1: parameter[1]()
#2: Int32Constant[0]()
#3: Int32Add(#1, #2)
#4: Return(#3)
```

Reducer

```
#1: parameter[1]()
#4: Return(#1)
```

SMT encoder

$Src(p):$

$$v_{s1} = p \wedge v_{s2} = 0 \wedge v_{s3} = v_{s2} + v_{s1} \wedge r_s = v_{s3}$$

$Tgt(p):$

$$v_{t1} = p \wedge r_t = v_{t1}$$

Z3

$$\exists p \,.\, Src(p) \wedge Tgt(p) \wedge r_s \neq r_t$$

SAT → **Not verified** $c.e.: model(p)$

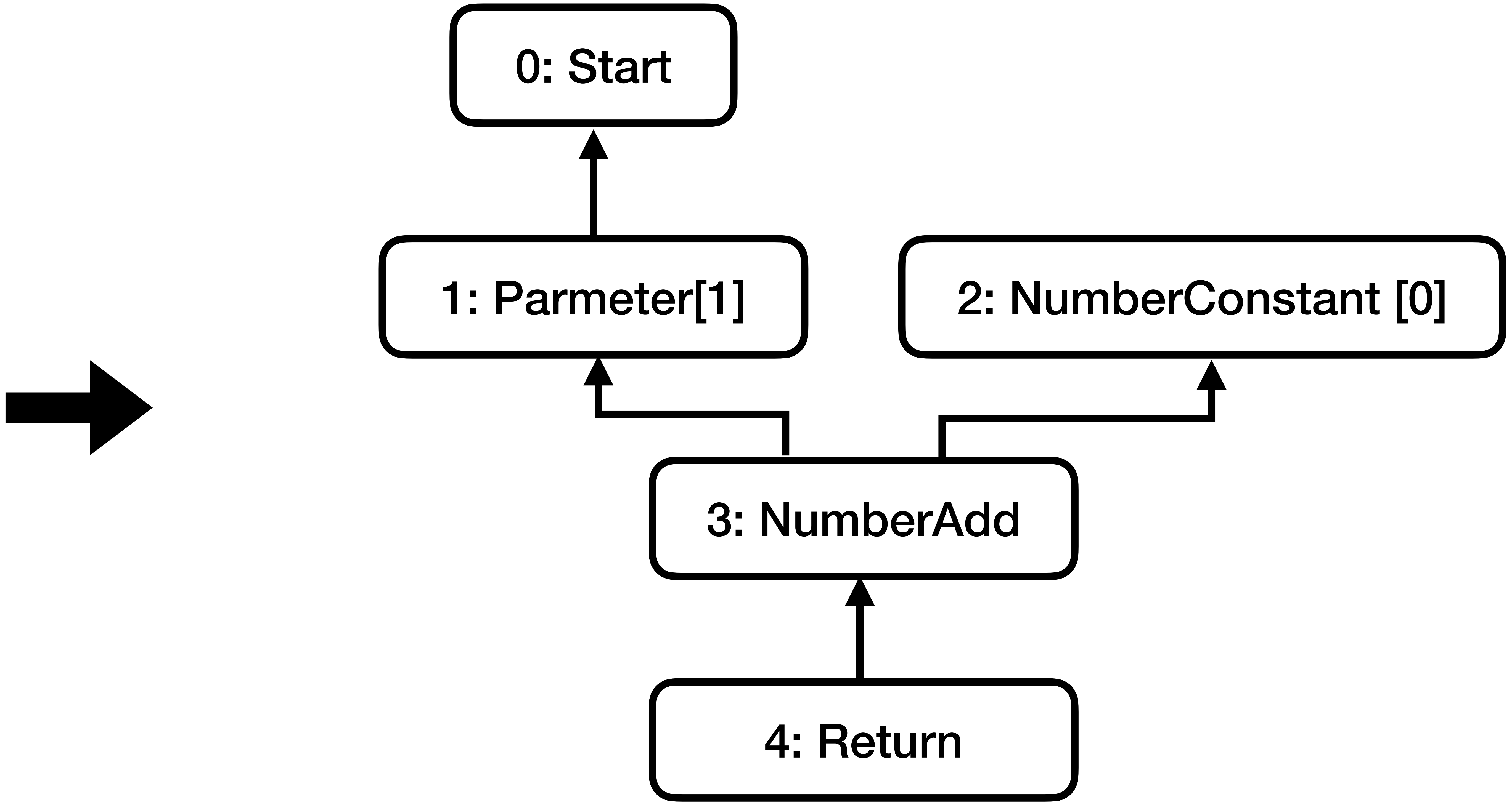
$UNSAT$ → **Verified**

TurboFan Intermediate Representation (IR)

Functions is a directed graph of nodes and edges

$G_S = \langle Node, \rightarrow_S \rangle$

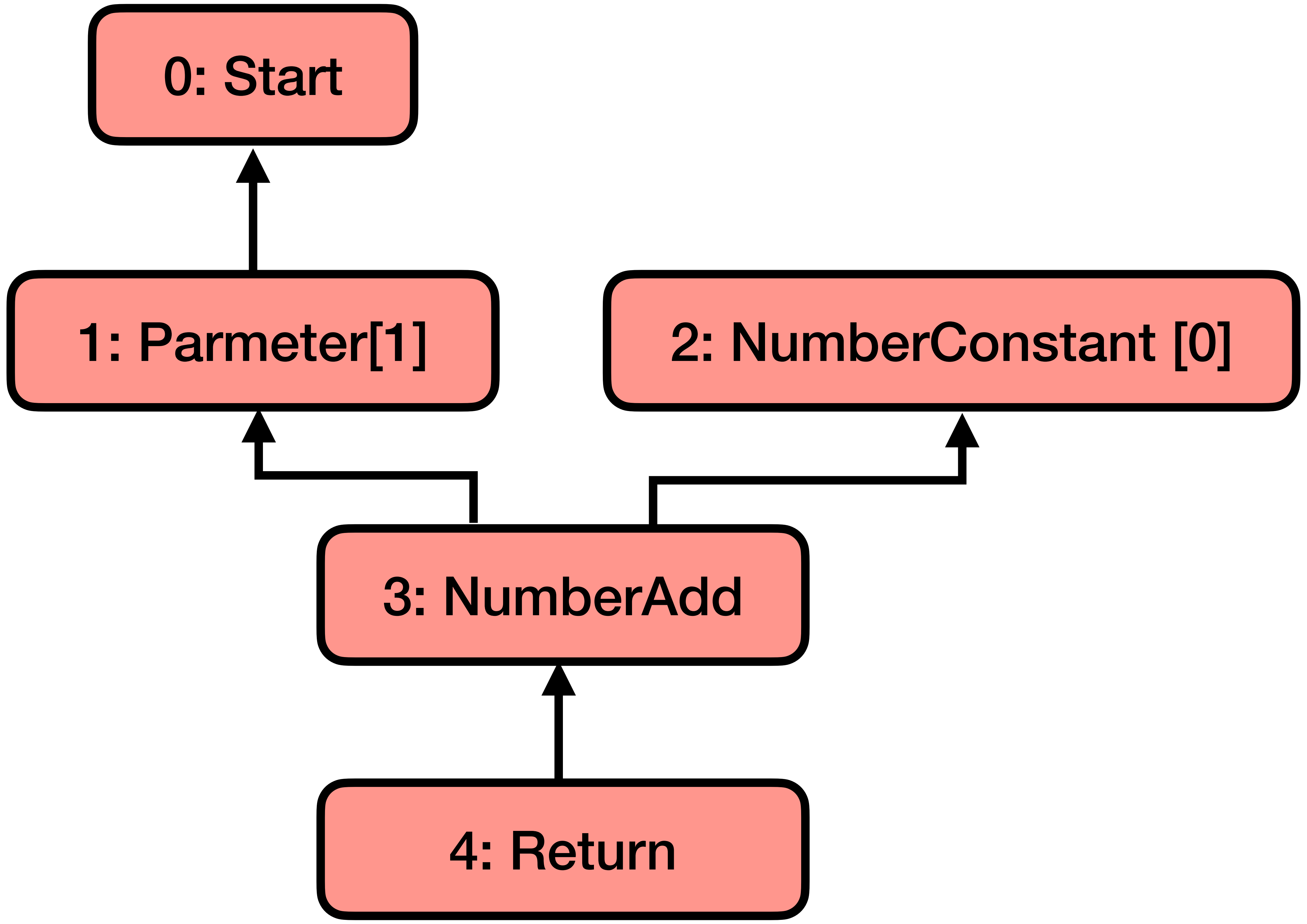
```
function foo(x) {
    return x+0
}
```



TurboFan Intermediate Representation (IR)

Node: opcode, value

```
function foo(x) {
    return x+0
}
```



0: Start

1: Parmeter[1]

2: NumberConstant [0]

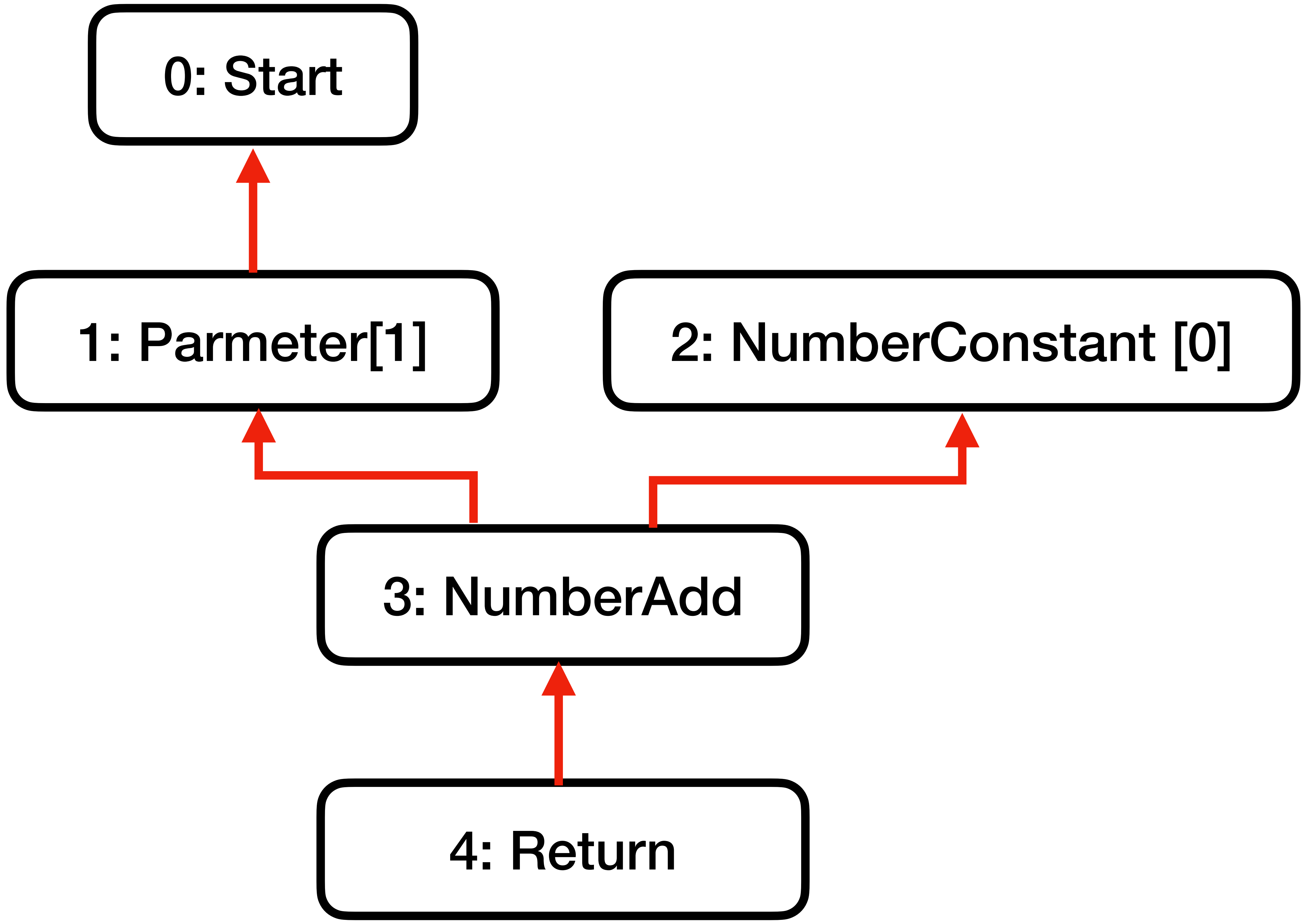
3: NumberAdd

4: Return

TurboFan Intermediate Representation (IR)

Edge: flow of information

```
function foo(x) {
    return x+0
}
```



0: Start

1: Parmeter[1]

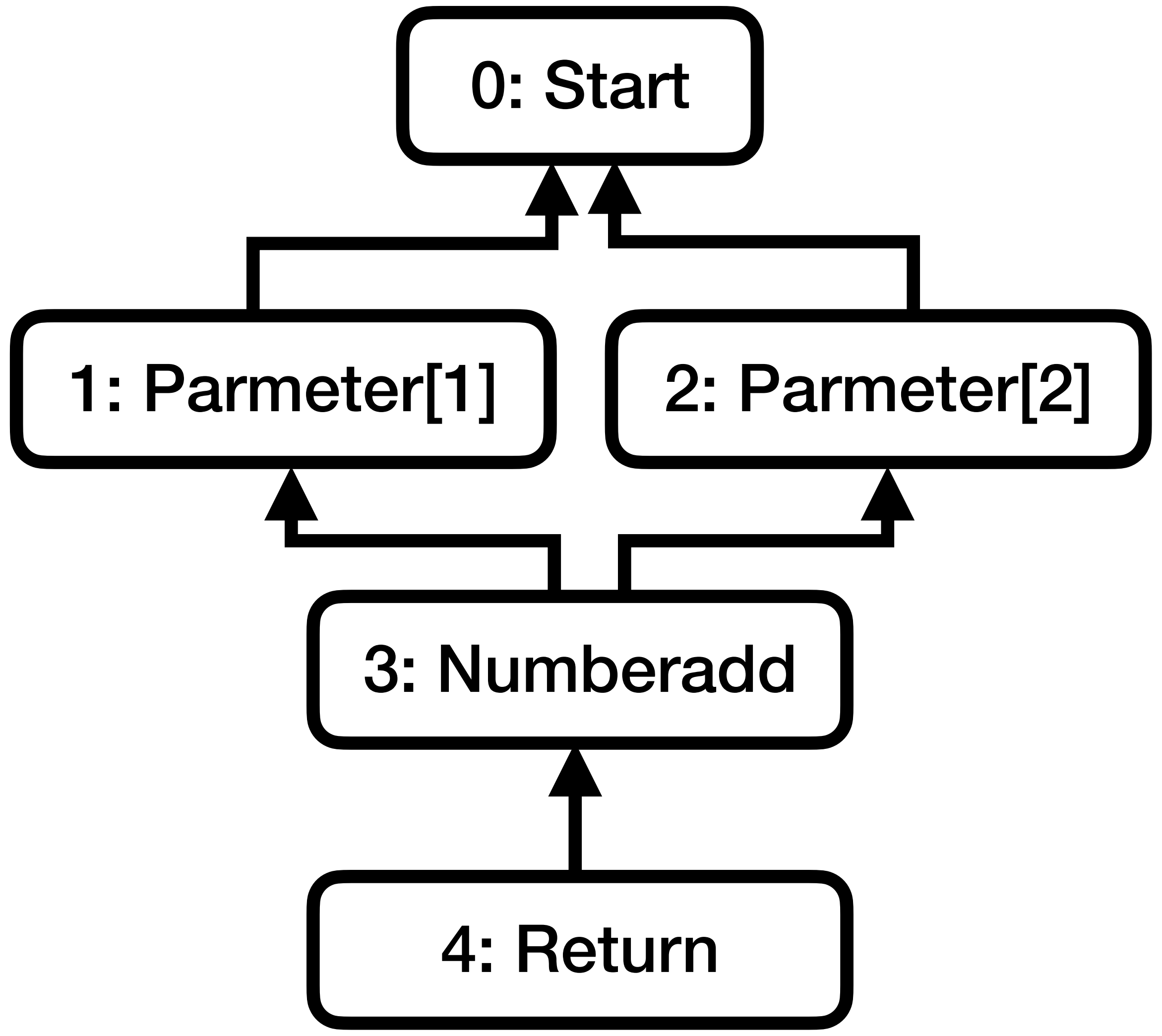
2: NumberConstant [0]

3: NumberAdd

4: Return

TurboTV - Example

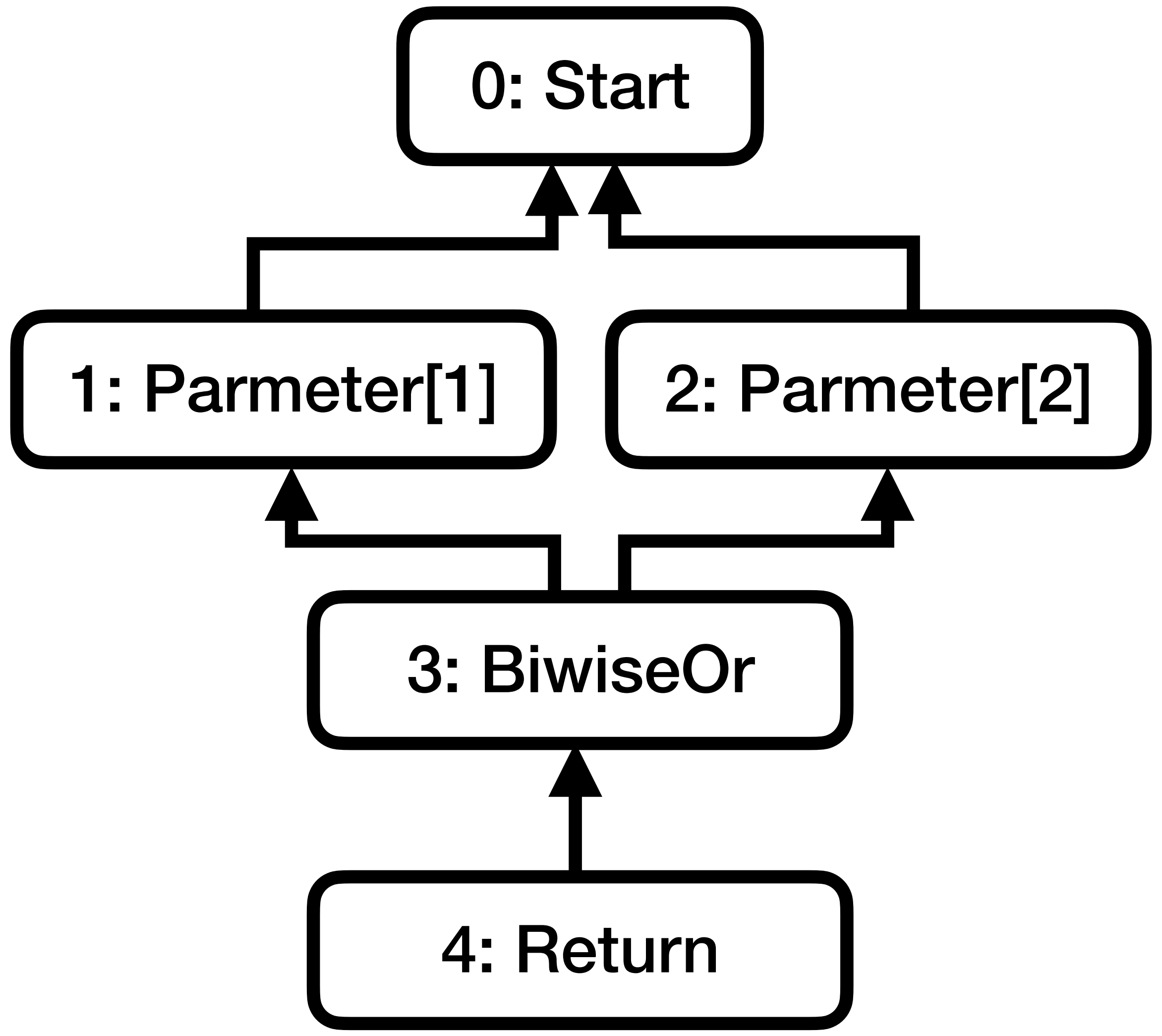
TurboFan's optimization



Source TurboFan IR

$$X + Y \rightarrow X \mid Y$$

Wrong Optimization



Target TurboFan IR

Reducer: TurboFan's optimizer

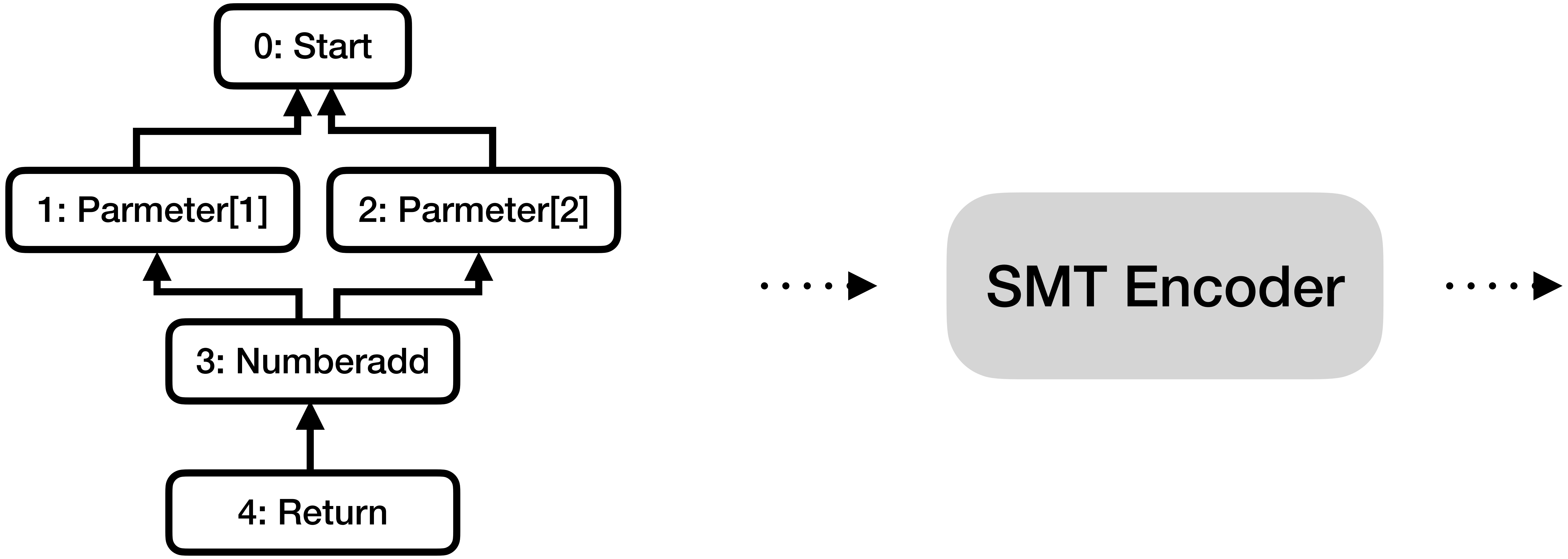
p_i : **Program parameter**

v_{si} : **Program values**

r_s : **Program return value**

TurboTV - Example

Program Encoding



Source TurboFan IR

$$Src(p_1, p_2) :$$

$$\begin{aligned} & v_{s1} = p_1 \\ & \wedge v_{s2} = p_2 \\ & \wedge v_{s3} = v_{s2} + v_{s1} \\ & \wedge r_s = v_{s3} \end{aligned}$$

Encoded Source program

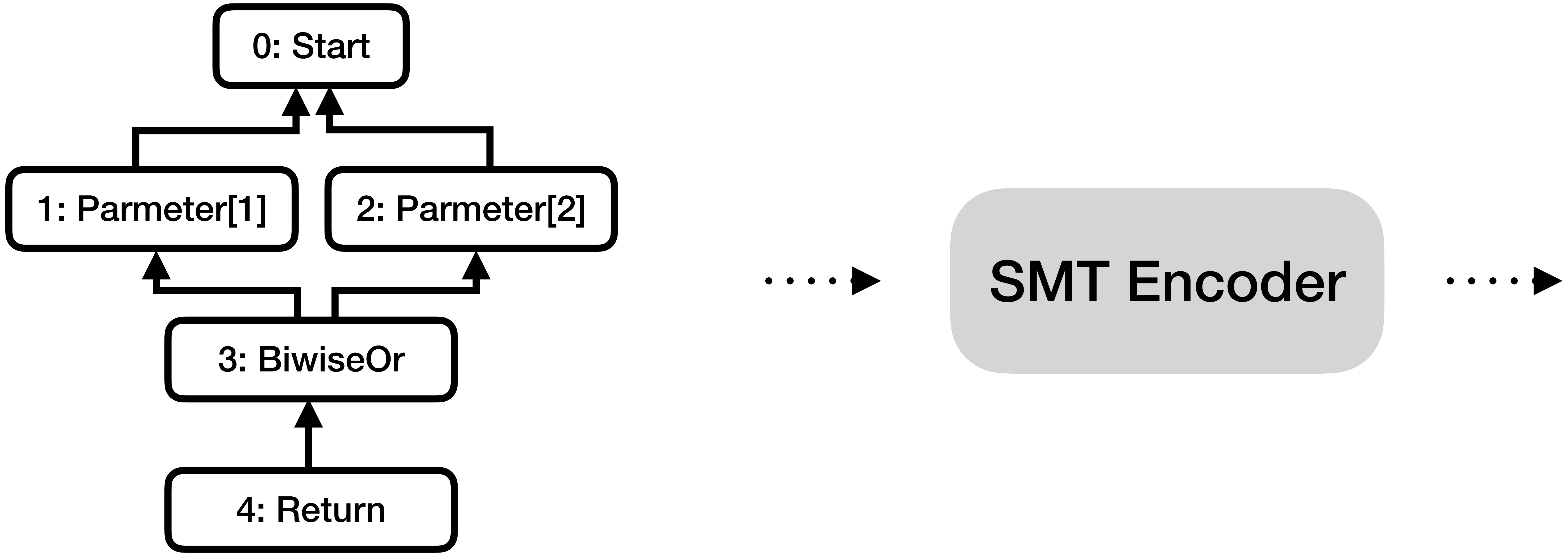
p_i : **Program parameter**

v_{ti} : **Program values**

r_t : **Program return value**

TurboTV - Example

Program Encoding



Target TurboFan IR

$$Tgt(p_1, p_2) :$$

$$\begin{aligned} & v_{t1} = p_1 \\ & \wedge v_{t2} = p_2 \\ & \wedge v_{t3} = v_{t1} \mid v_{t2} \\ & \wedge r_t = v_{t3} \end{aligned}$$

Encoded target program

TurboTV - Example

Program Equivalence Check

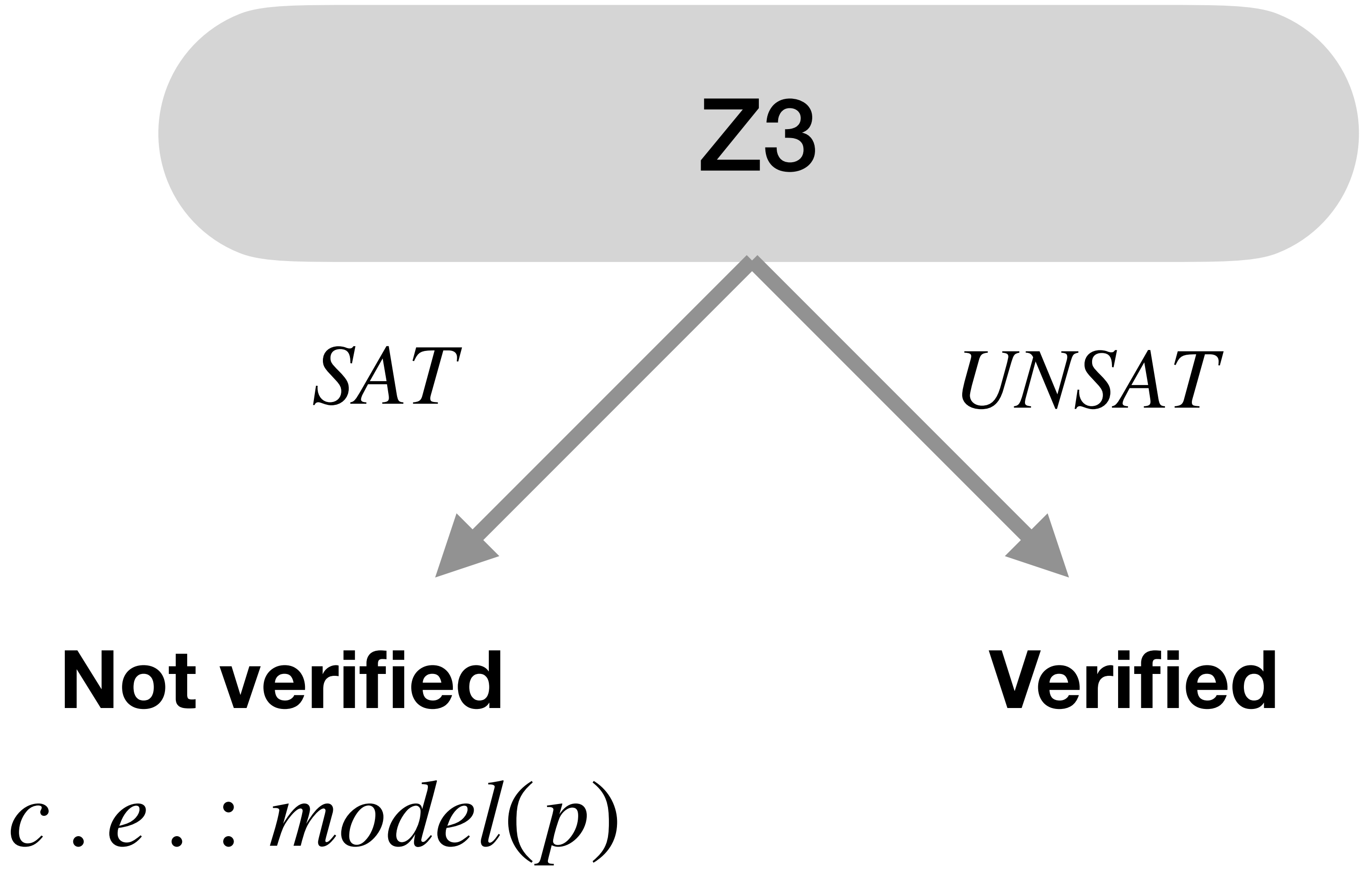
$$
\begin{aligned}
Src(p_1, p_2) :& \\
& v_{s1} = p_1 \\
& \wedge v_{s2} = p_2 \\
& \wedge v_{s3} = v_{s2} + v_{s1} \\
& \wedge r_s = v_{s3}
\end{aligned}
$$

Encoded source program

$$
\begin{aligned}
Tgt(p_1, p_2) :& \\
& v_{t1} = p_1 \\
& \wedge v_{t2} = p_2 \\
& \wedge v_{t3} = v_{t1} \mid v_{t2} \\
& \wedge r_t = v_{t3}
\end{aligned}
$$

Encoded target program

$$\exists p_1 \exists p_2 . Src(p_1, p_2) \wedge Tgt(p_1, p_2) \wedge r_s \neq r_t$$



Z3

SAT

$UNSAT$

Not verified

$c.e.: model(p)$

Verified

TurboTV - Example

Program Equivalence Check

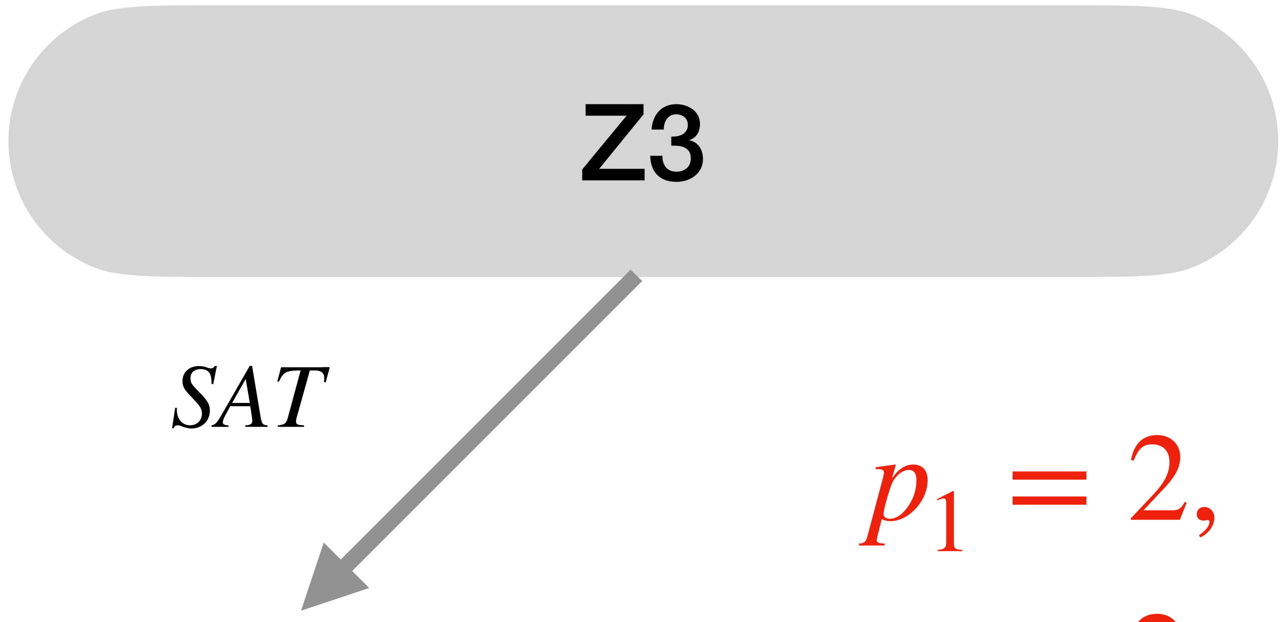
$$
\begin{aligned}
Src(p_1, p_2) :& \\
& v_{s1} = p_1 \\
& \wedge v_{s2} = p_2 \\
& \wedge v_{s3} = v_{s2} + v_{s1} \\
& \wedge r_s = v_{s3}
\end{aligned}
$$

Encoded source program

$$
\begin{aligned}
Tgt(p_1, p_2) :& \\
& v_{t1} = p_1 \\
& \wedge v_{t2} = p_2 \\
& \wedge v_{t3} = v_{t1} \mid v_{t2} \\
& \wedge r_t = v_{t3}
\end{aligned}
$$

Encoded target program

$$\exists p_1 \exists p_2 . Src(p_1, p_2) \wedge Tgt(p_1, p_2) \wedge r_s \neq r_t$$



Not verified

$c.e.: model(p)$

$p_1 = 2,$

$p_2 = 2,$

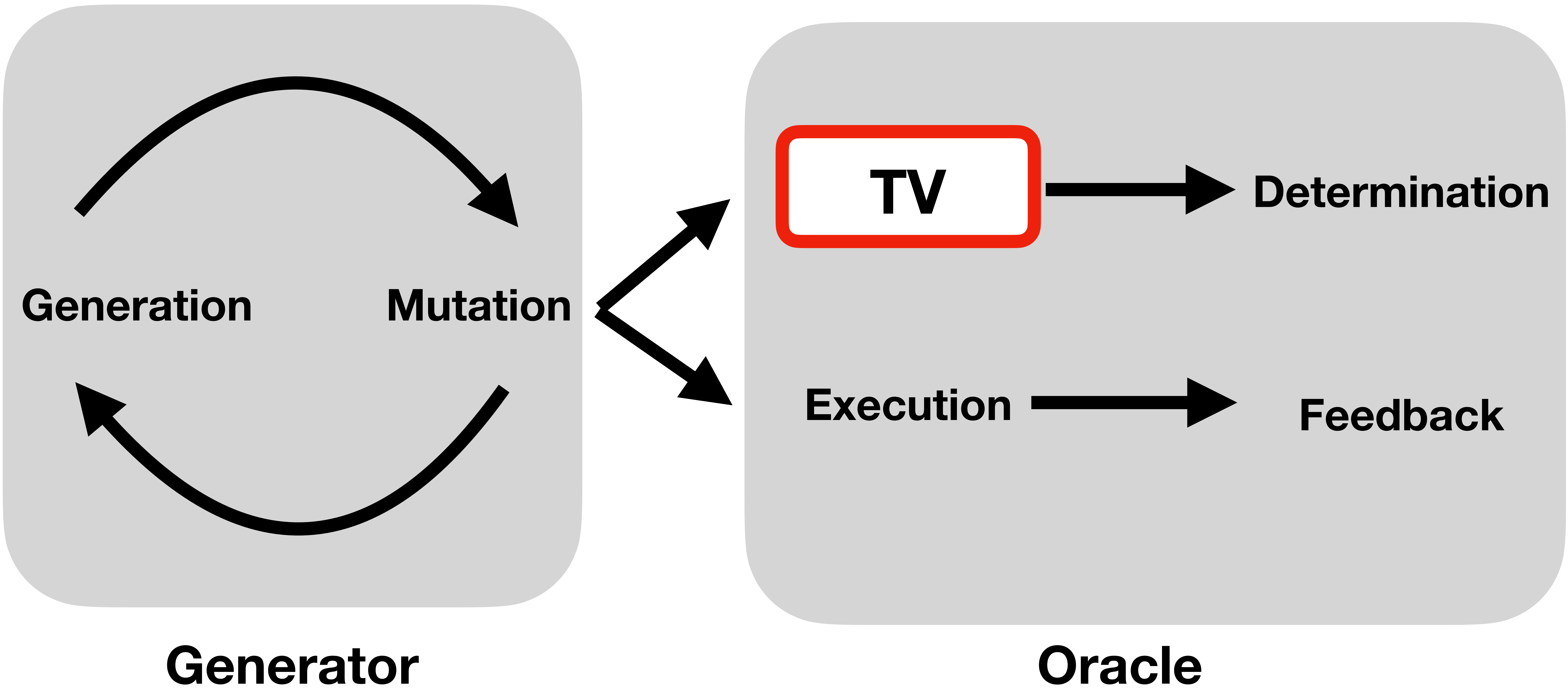
$r_s = 4,$

$r_t = 2,$

$\ldots$

Generative Translation Validation

Utilize translation validator as a fuzzing bug oracle


Generation
Mutation
Generator
TV
Determination
Execution
Feedback
Oracle