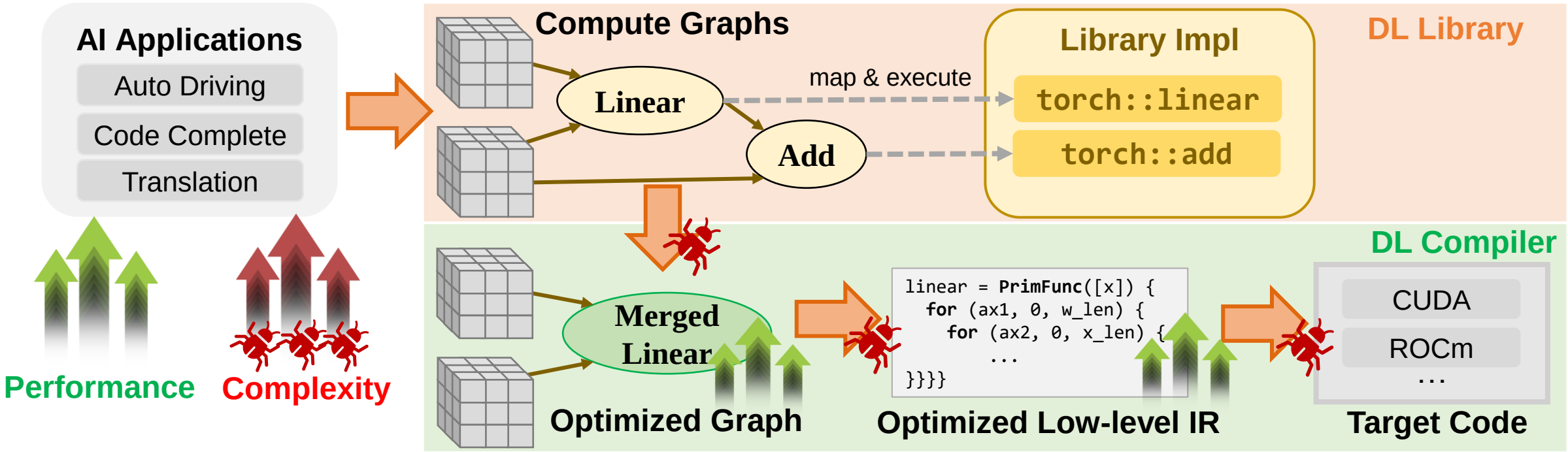
Coverage-Guided *Tensor Compiler Fuzzing* with Joint IR-Pass Mutation

Jiawei Liu, Yuxiang Wei, Sen Yang, Yinlin Deng, Lingming Zhang

jiawei6@illinois.edu

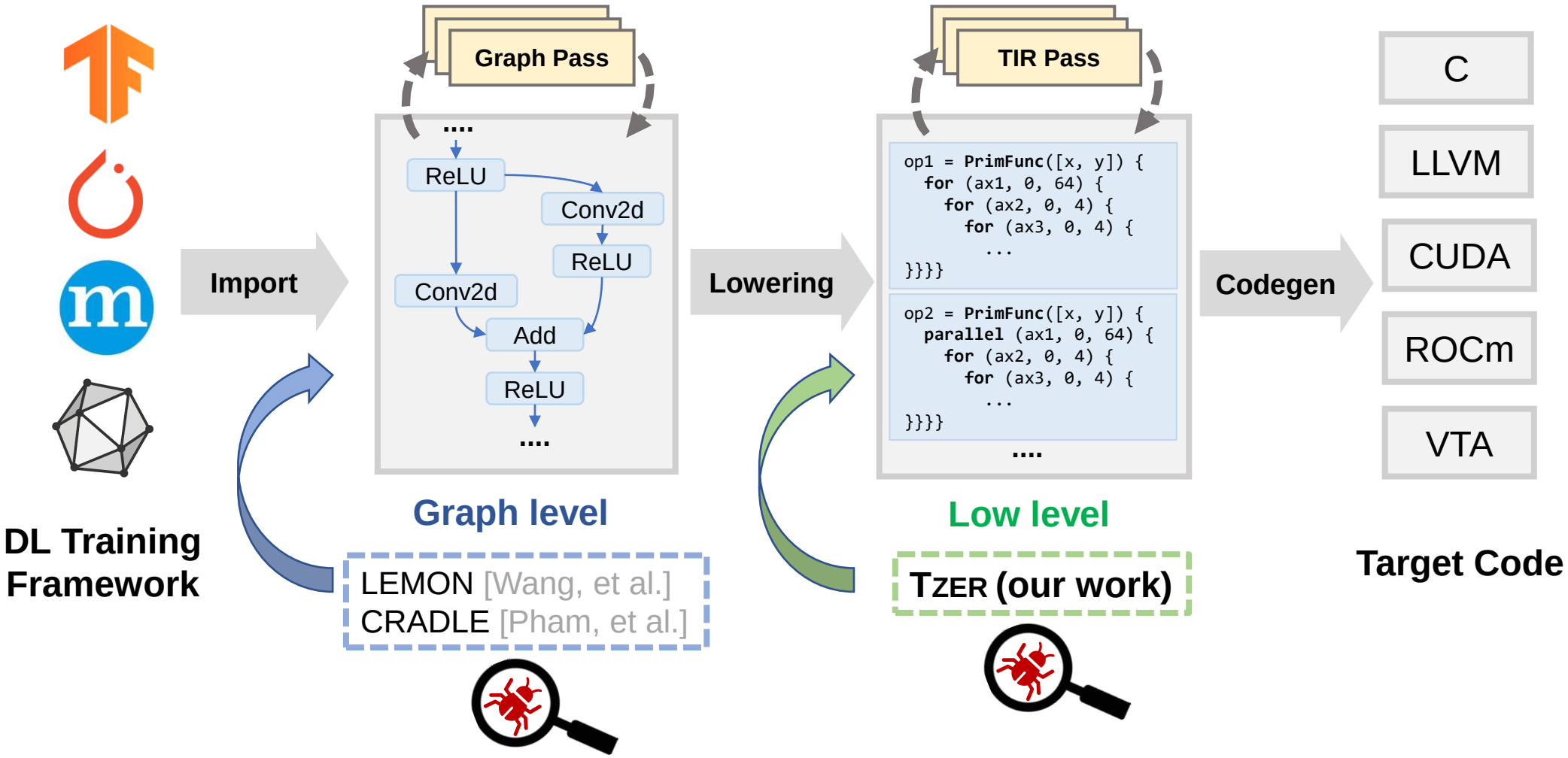
DL Frameworks: from Libraries to Compilers

- **First** gen.: **Libraries**, e.g., PyTorch, TensorFlow.
- **Second** gen.: **Compilers**, e.g., TVM, TensorFlow XLA, PyTorch JIT.

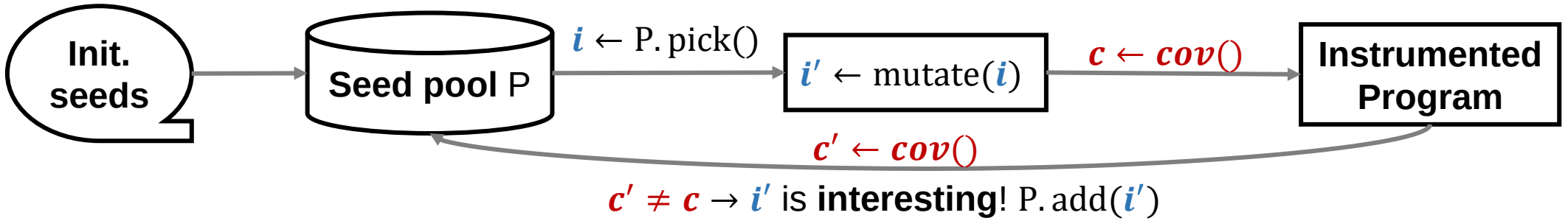


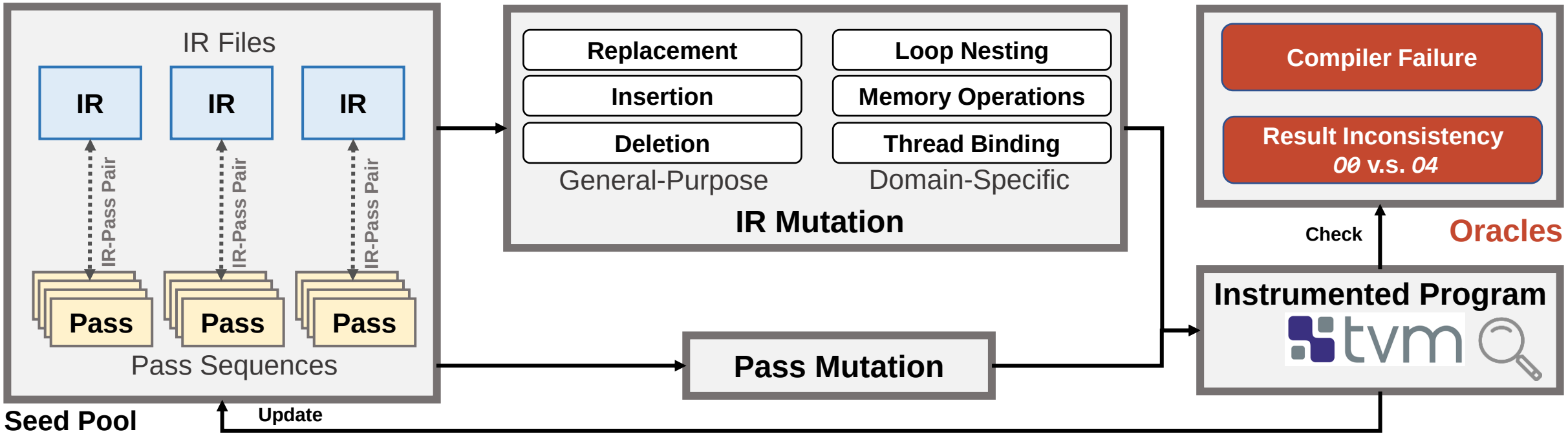
Goal: automatically detect bugs in tensor compilers.

A Typical Workflow of Tensor Compiler



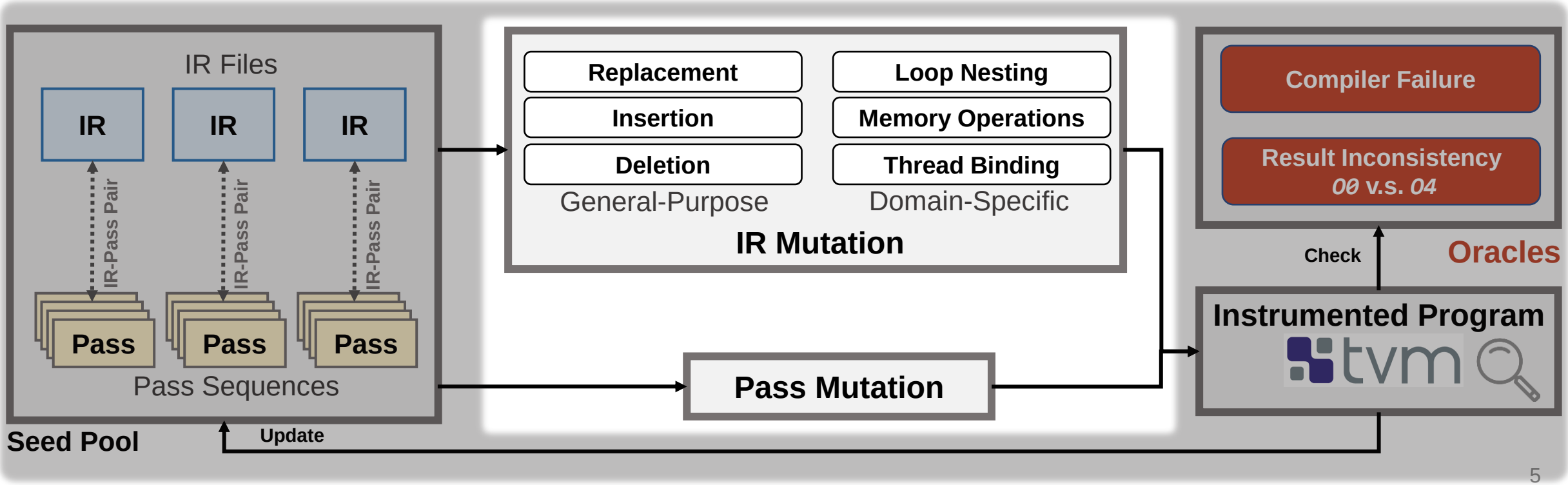
Fuzzing Loop in TZER





Generating New Test-Cases with Mutation

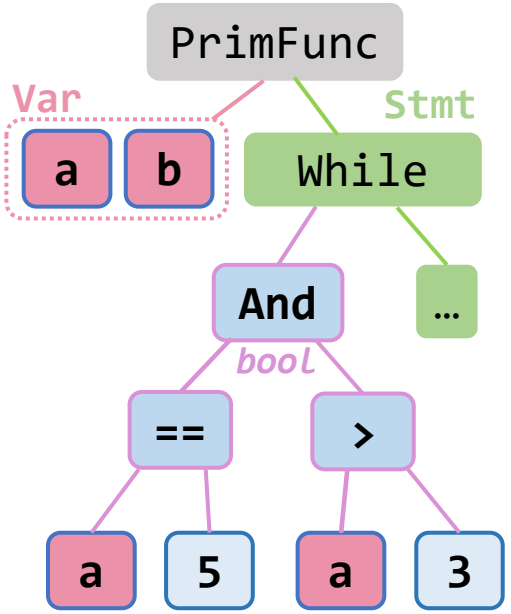
- **IR Mutation**: practice the compilation of various IR structures.
- **Pass Mutation**: practice the compatibility of pass orders.



General IR Mutation

```
PrimFunc(a, b) {
  while (a == 5 && a > 3) {
    …
  }
}
```

- Select target AST node to mutate.
- Perform insertion/deletion/replacement with constraints.
  - e.g., accessible variables, type requirements, etc.

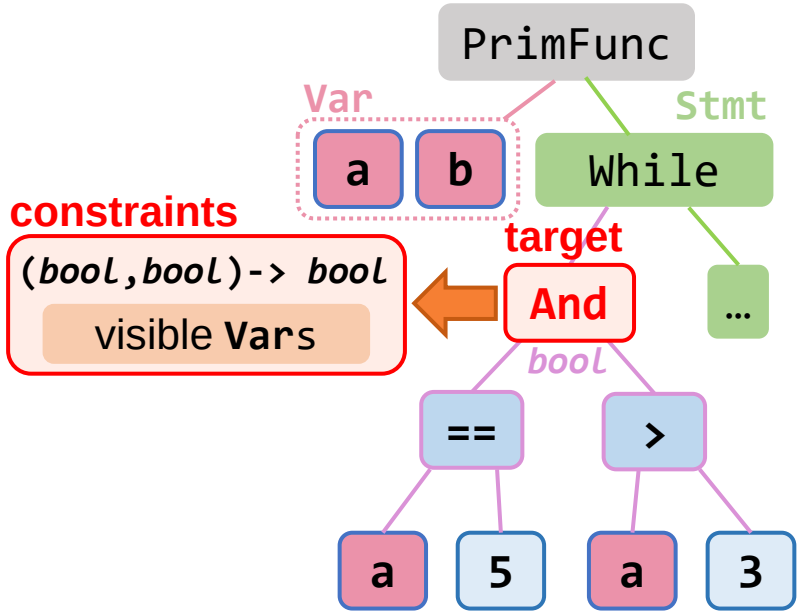


Original IR

General IR Mutation

```
PrimFunc(a, b) {
  while (a == 5 && a > 3) {
    …
  }
}
```

- Select **target AST node** to mutate.
- Perform insertion/deletion/replacement with **constraints**.
  - e.g., accessible variables, type requirements, etc.

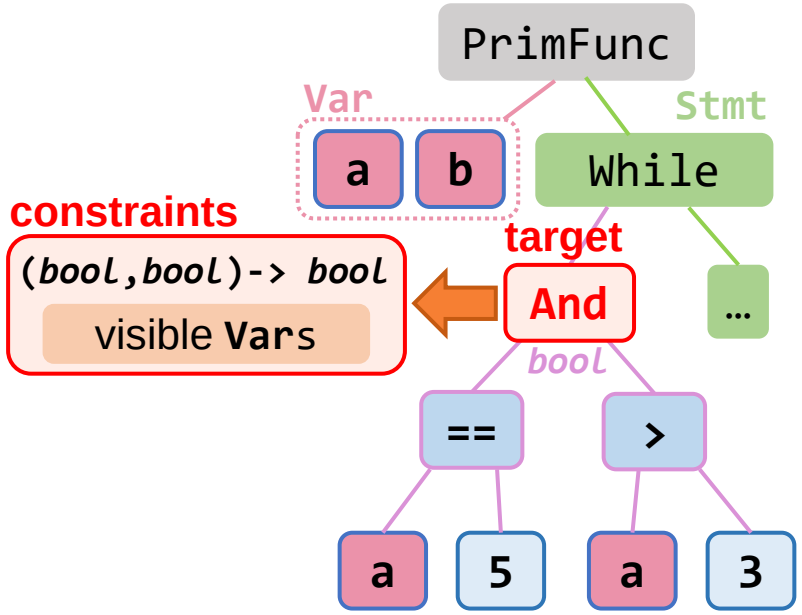


Original IR

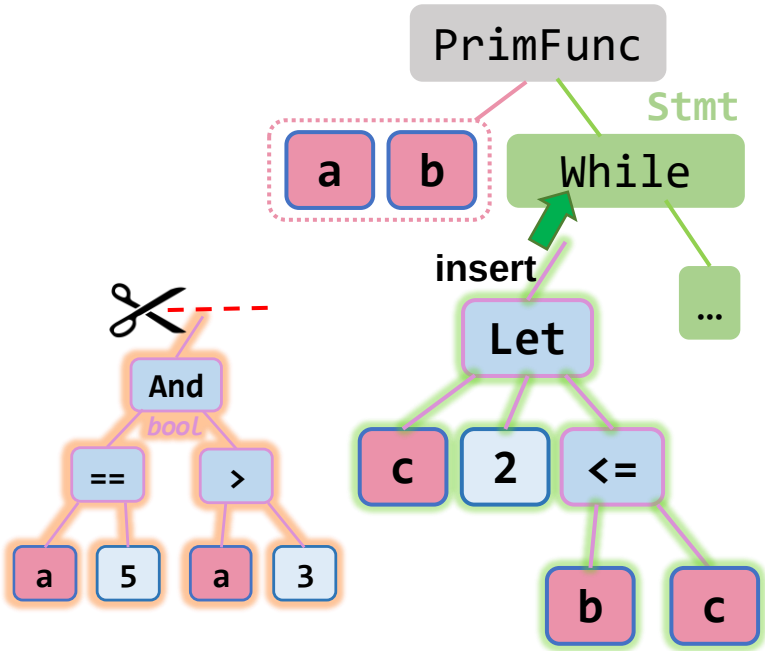
General IR Mutation

```
PrimFunc(a, b) {
- while (a == 5 && a > 3) {
+ while (c = 2; b <= c  ) {
    …
  }
}
```

- Select **target AST node** to mutate.
- Perform **insertion**/deletion/replacement with **constraints**.
  - e.g., accessible variables, type requirements, etc.



Original IR

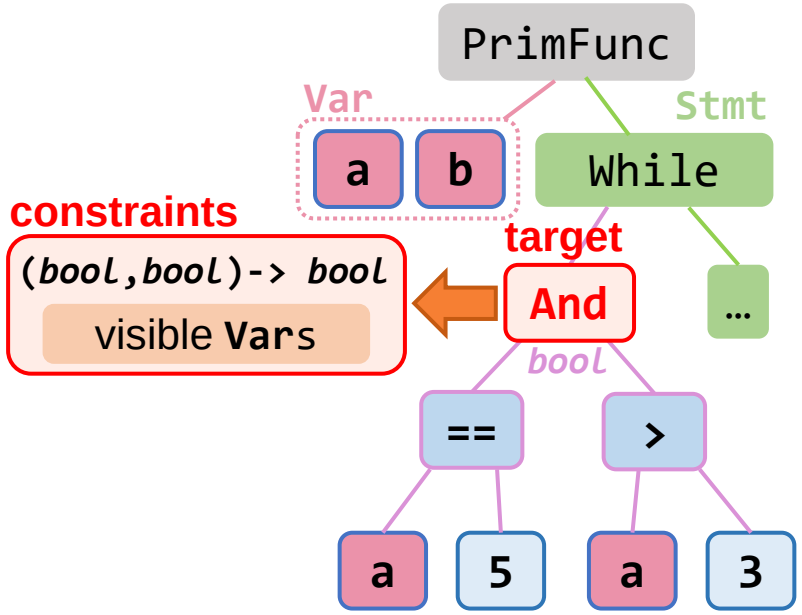


Mutated IR (*Insertion*)

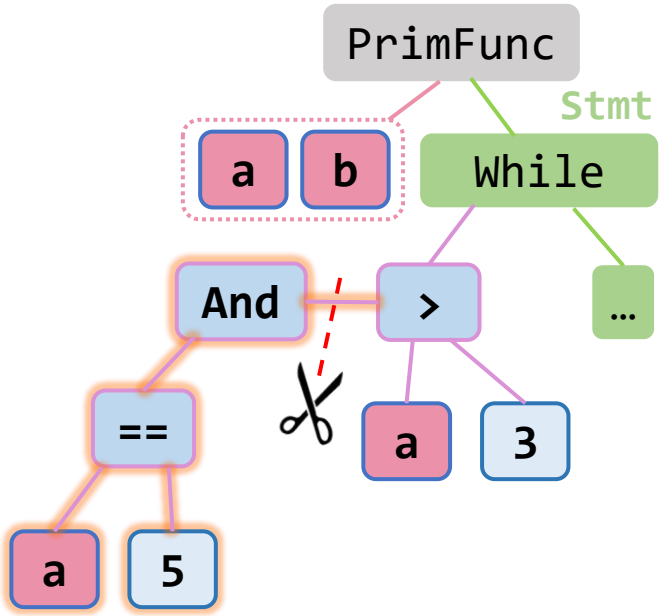
General IR Mutation

```
PrimFunc(a, b) {
- while (a == 5 && a > 3) {
+ while (a > 3) {
    …
  }
}
```

- Select **target AST node** to mutate.
- Perform insertion/**deletion**/replacement with **constraints**.
  - e.g., accessible variables, type requirements, etc.



Original IR

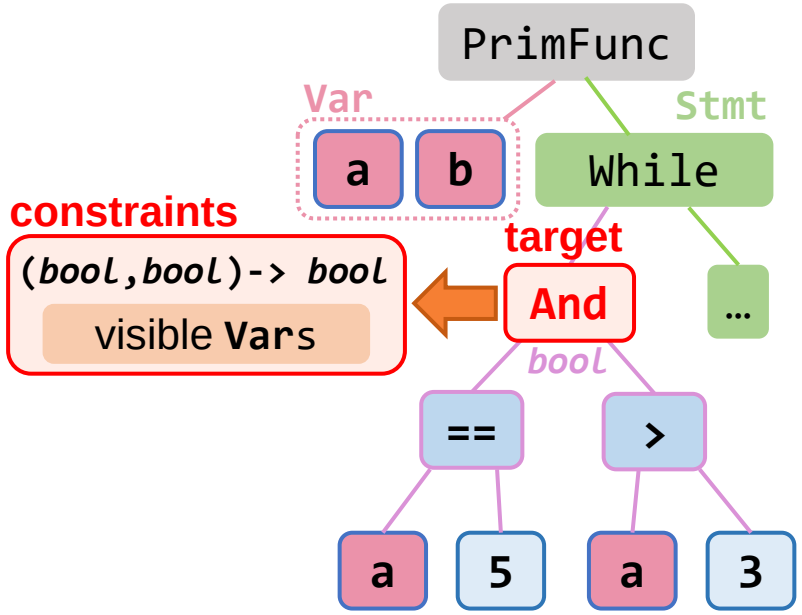


Mutated IR (*Deletion*)

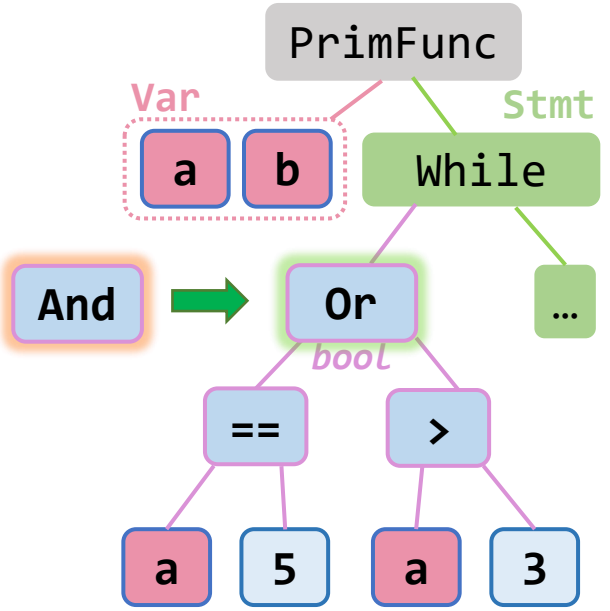
General IR Mutation

```
PrimFunc(a, b) {
- while (a == 5 && a > 3) {
+ while (a == 5 || a > 3) {
    …
  }
}
```

- Select **target AST node** to mutate.
- Perform insertion/deletion/**replacement** with **constraints**.
  - e.g., accessible variables, type requirements, etc.



Original IR



Mutated IR (*Node Replacement*)

Domain-specific IR Mutation

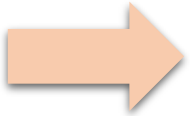
- Tensor program optimization is highly related to
  - **Loops**: add outer loops;
  - **Memory**: mutate memory load/store;
  - **Threads**: mutate threading patterns.

Domain-specific IR Mutation

- Tensor program optimization is highly related to
  - **Loops**: insert loops;
  - **Memory**: mutate memory load/store;
  - **Threads**: mutate threading patterns.

```
PrimFunc(...) {
  buf[0] = 1
}
```

Loop Nesting



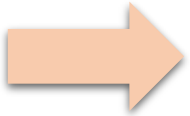
```
PrimFunc(...) {
  // attr ... unroll_max_step = 2
  unrolled (i, 0, 16) {
    unrolled (j, 0, 16) {
      buf[0] = 1
    }
  }
}
```

Domain-specific IR Mutation

- Tensor program optimization is highly related to
  - **Loops**: add outer loops;
  - **Memory**: mutate memory load/store;
  - **Threads**: mutate threading patterns.

```
PrimFunc(...) {
  // attr ... unroll_max_step = 2
  unrolled (i, 0, 16) {
    unrolled (j, 0, 16) {
      buf[0] = 1
    }
  }
}
```

Memory Operation



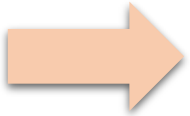
```
PrimFunc(...) {
  // attr ... unroll_max_step = 2
  unrolled (i, 0, 16) {
    unrolled (j, 0, 16) {
      buf[i * 16 + j] = buf[i + j * 16]
    }
  }
}
```

Domain-specific IR Mutation

- Tensor program optimization is highly related to
  - **Loops**: add outer loops;
  - **Memory**: mutate memory load/store;
  - **Threads**: mutate threading patterns.

```
PrimFunc(...) {
  // attr ... unroll_max_step = 2
  unrolled (i, 0, 16) {
    unrolled (j, 0, 16) {
      buf[i * 16 + j] = buf[i + j * 16]
    }
  }
}
```

Thread Binding



```
PrimFunc([]) {
  // attr ... virtual_thread = 2
  launch_thread (t, 0, 2) {
    // attr ... unroll_max_step = 2
    unrolled (i, 0, 16) {
      unrolled (j, 0, 16) {
        buf[i * 16 + j] = buf[i + j * 16]
      }
    }
  }
}
```

How to mutate <IR, PassSeq> jointly?

Coverage is sensitive to mutation frequency of **IR** & **PassSeq**.

$\langle IR_0, PassSeq_0 \rangle$

$\langle IR_0, PassSeq_1 \rangle$

$\langle IR_0, PassSeq_2 \rangle$

$\langle IR_0, PassSeq_3 \rangle$

$\langle IR_1, PassSeq_0 \rangle$

$\langle IR_2, PassSeq_0 \rangle$

$\langle IR_3, PassSeq_0 \rangle$

Always mutate PassSeqs?

- ❖ Less coverage-efficient than IR mutation.
- ❖ Lower compile rate by specific pairs (bug duplicates).

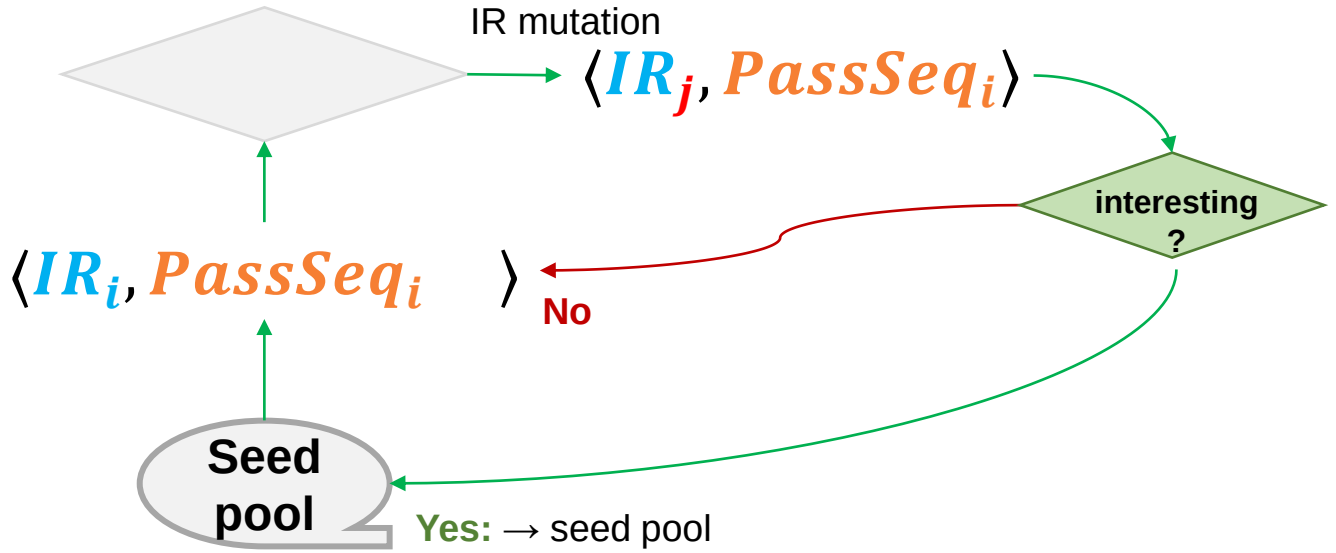
Always mutate IRs?

- ❖ Limited chances for practicing pass orders.

Joint IR-Pass Mutation

Solutions:

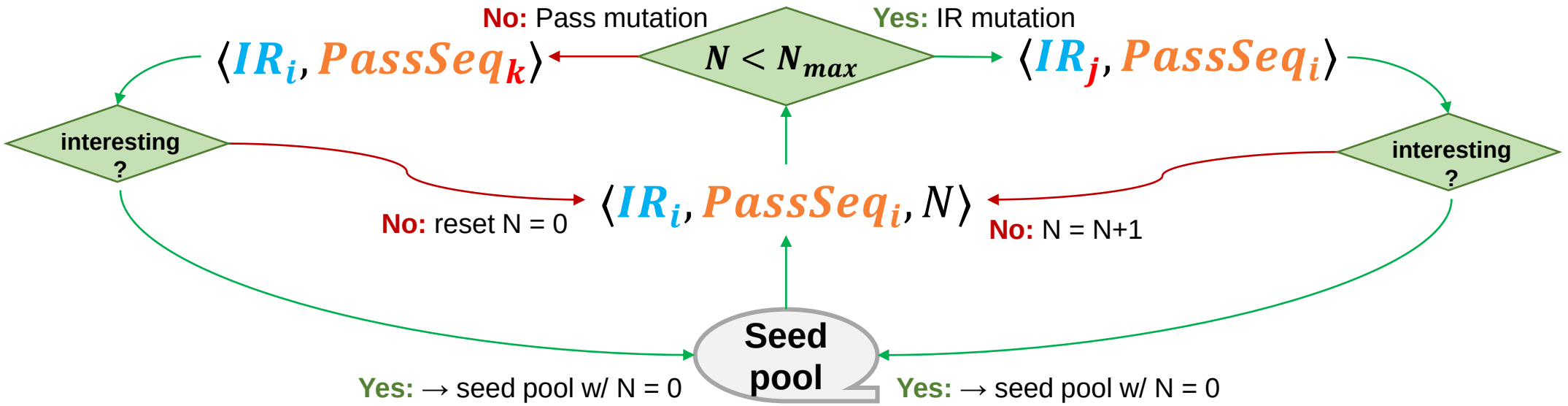
- **Prefer IR mutation with interleaving factor N_{max}**
- Avoid uninteresting $\langle IR, PassSeq \rangle$ (no new cov./uncompilable)



Joint IR-Pass Mutation

Solutions:

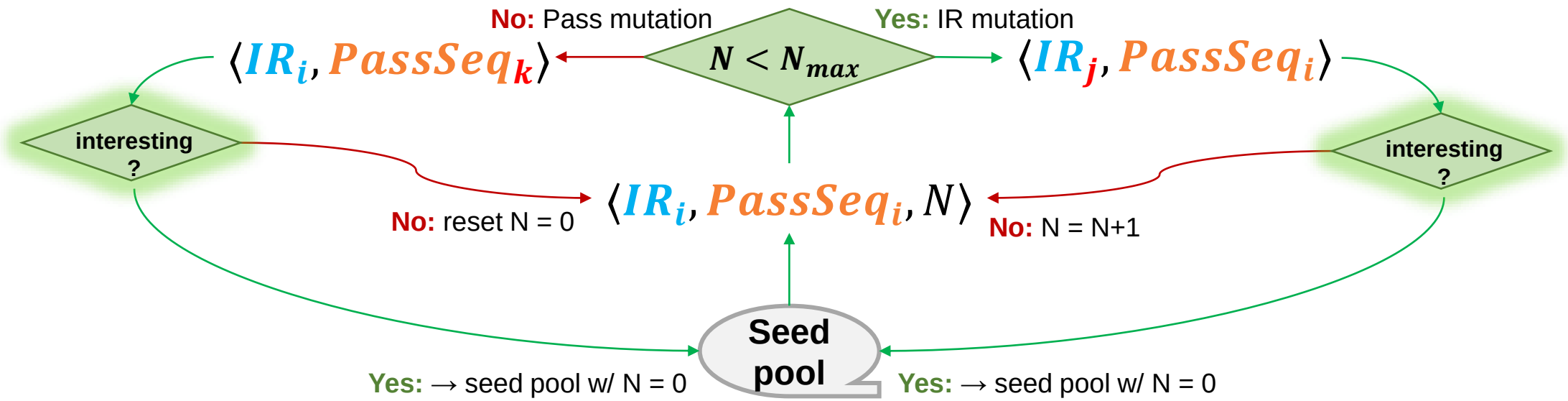
- **Prefer IR mutation with interleaving factor N_{max}**
- Avoid uninteresting $\langle IR, PassSeq \rangle$ (no new cov./uncompilable)



Joint IR-Pass Mutation

Solutions:

- Prefer IR mutation with interleaving factor N_{max}
- **Avoid uninteresting $\langle IR, PassSeq \rangle$ (no new cov./uncompilable)**



Bug Finding in TVM

49 **reported;** 37 **confirmed;** 25 **fixed**

Among all (37) confirmed bugs:

- **LEMON: 3** (8%) — Fuzzing at **Graph** Level
- **TVMFuzz: 6** (16%) — Fuzzing at **TIR** Level
- **LibFuzzer: 3** (8%) — Fuzzing at **Binary** Level
- **TZER** without **pass mutation: 17** (46%)

Sample: Violation of IR immutability

- The IR module in TVM is copied-on-write.
- The **`ToBasicBlockNormalForm`** pass could write the input IR in place even if it is not uniquely owned.
- This is because the use of **`T* operator->() const`** which actually returns a (non-const) mutable.

```
- IRModule ToBasicBlockNormalForm(const IRModule& mod) {
+ IRModule ToBasicBlockNormalForm(const IRModule& mod_) {
+     auto mod = IRModule(mod_->functions, mod_->type_definitions, ...); // Deep copy.
```

tvm/8778: [BUG] ToBasicBlockNormalForm immutability

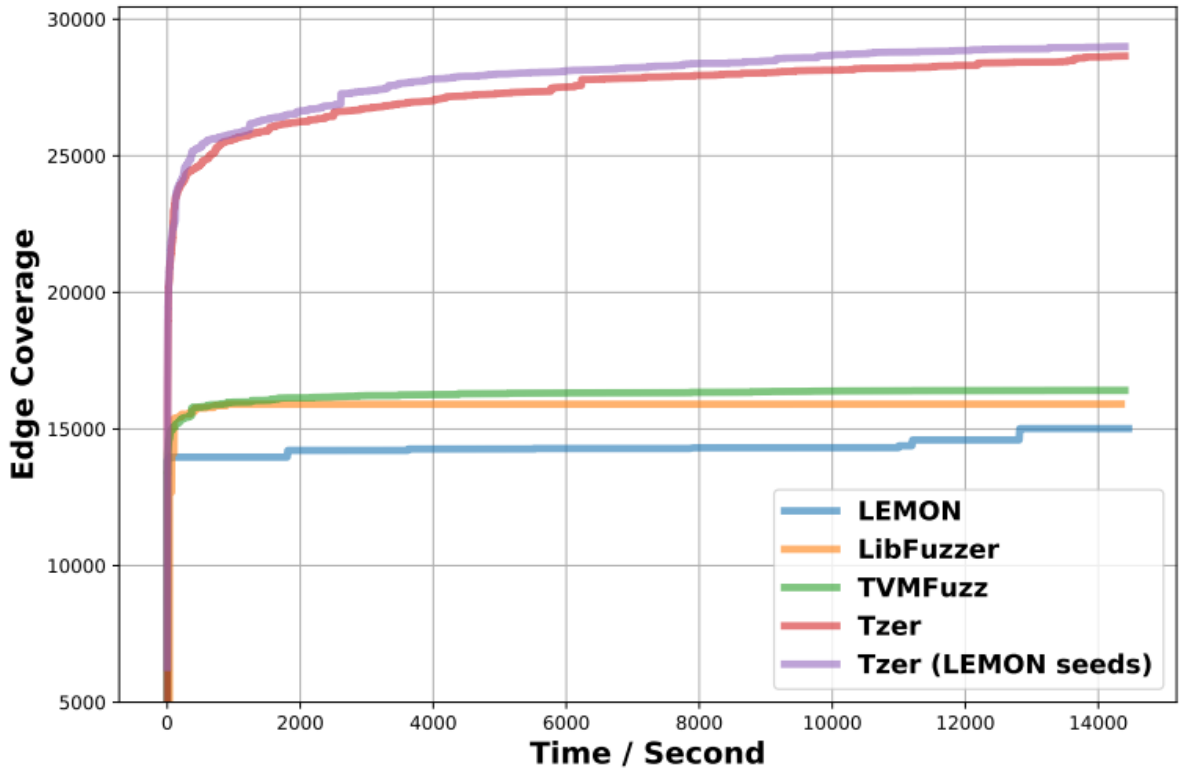
Sample: Out of Bound Read

- The **`IRConvertSSA`** pass can crash for out-of-bound read when any of the elements in **`scope_`** is an empty vector.

```
-     if (scope_.count(v)) {
+     if (scope_.count(v) && !scope_[v].empty()})
          return Load(op->dtype, scope_[v].back(), op->index, op->predicate);
```

tvm/8930: [Bugfix] Add check to avoid calling back() on an empty container

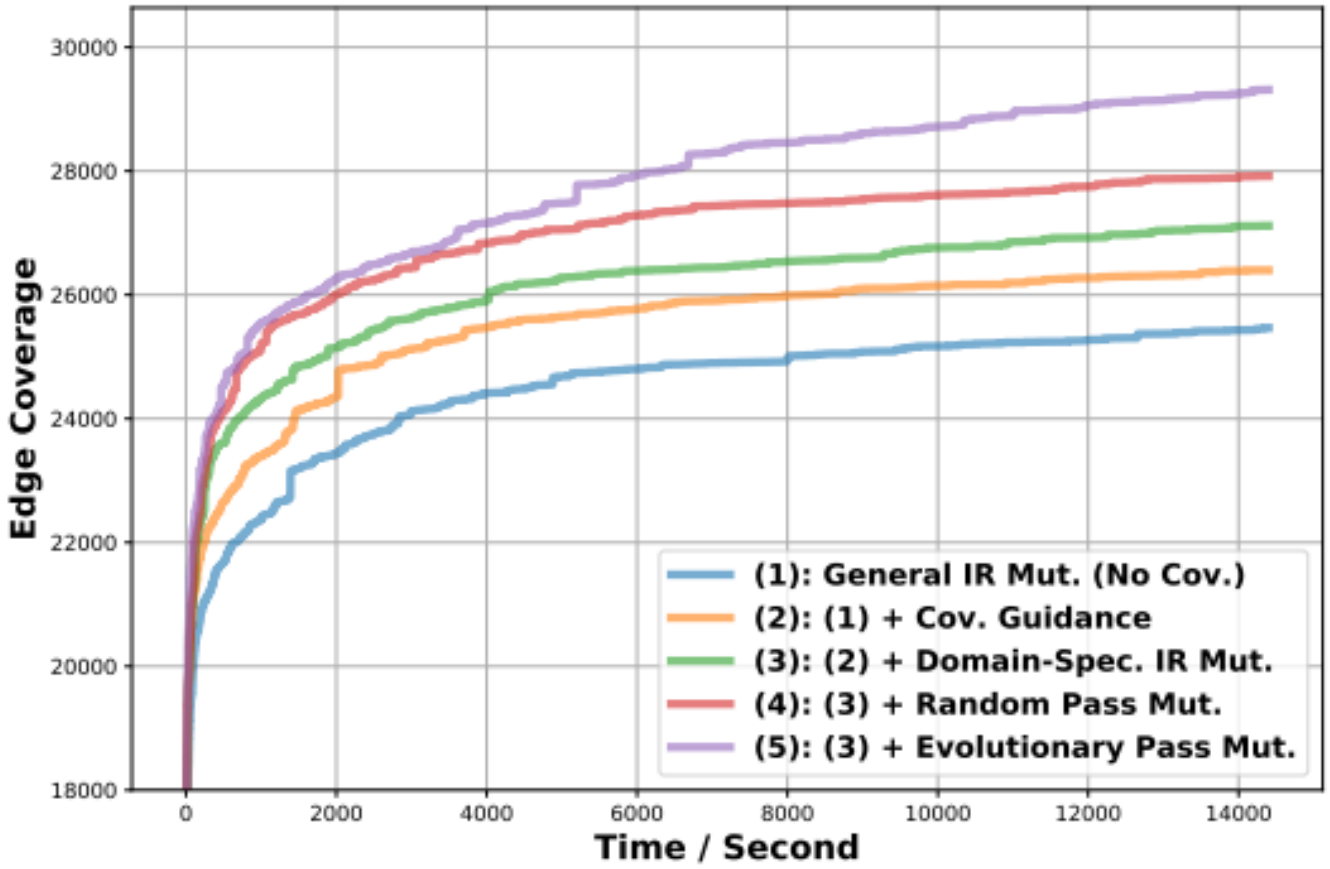
CFG Edge Coverage over 4 Hours



1.75x higher than **TVMFuzz**

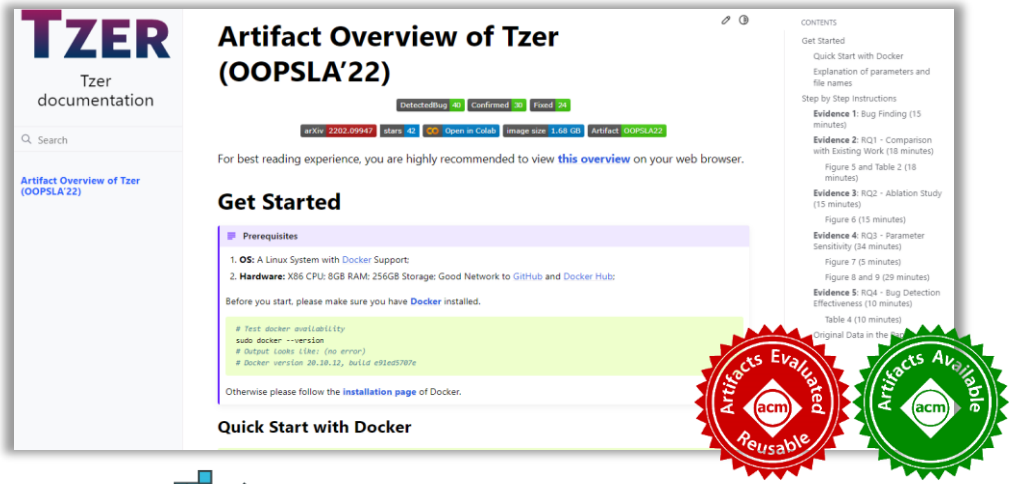
Coverage: Ablation Study

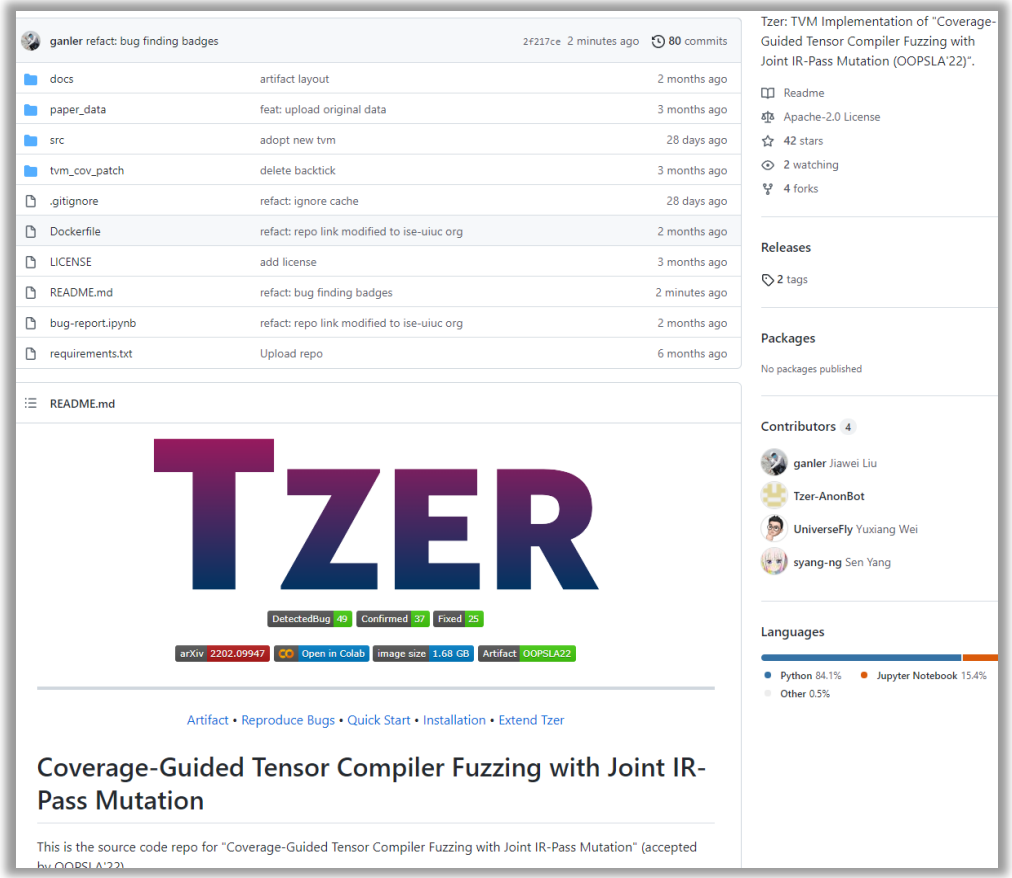
- ✓ **Coverage Guidance.**
- ✓ **Domain-spec. IR Mut.**
- ✓ **Pass Mutation.**
- ✓ **Evolutionary Fuzzing.**



Summary

- **TZER: a tensor compiler fuzzer**
  - General & Domain-spec. IR Mutation
  - Pass Mutation
  - Coverage-guided Evolutionary Fuzzing
- **Found 49 bugs (37 confirmed)**





docker Image: tzerbot/oopsla

GitHub ise-uiuc/tzer