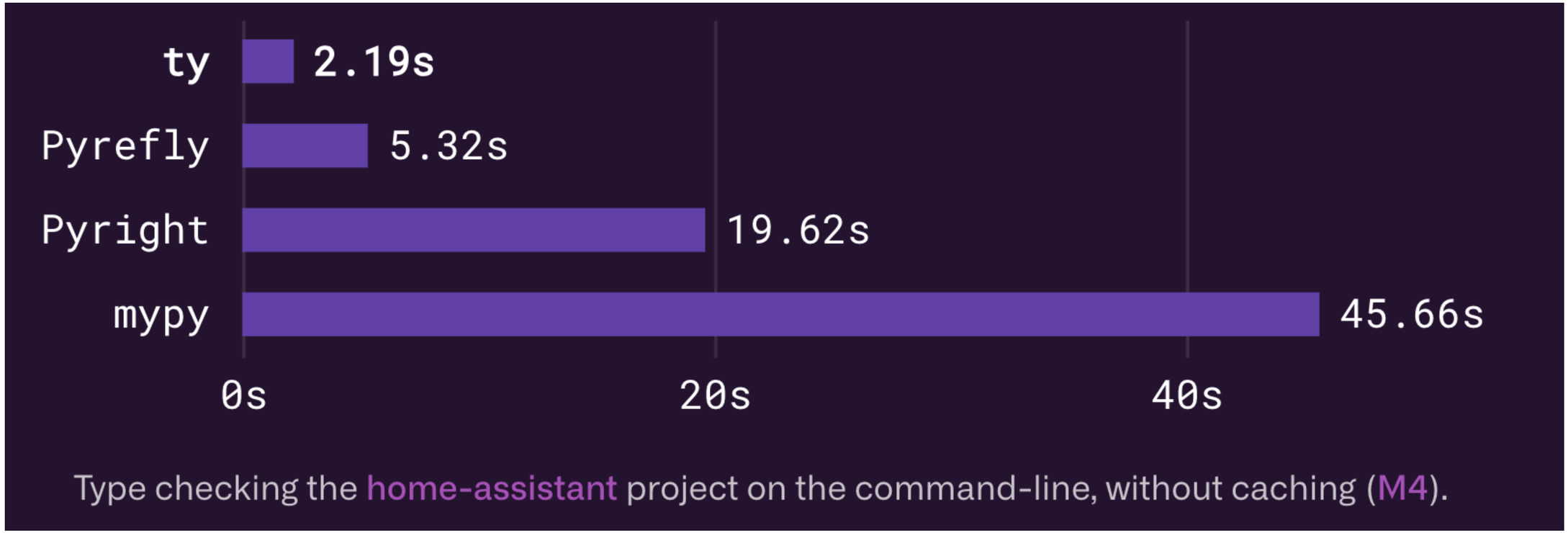
Ty: Adventures of type-checking Python in Rust

FOSDEM 2026

Shaygan Hooshyari

- Type checker for Python from ASTRAL
- Available as CLI, editor LSP, and play.ty.dev
- Focuses on speed and rich diagnostics



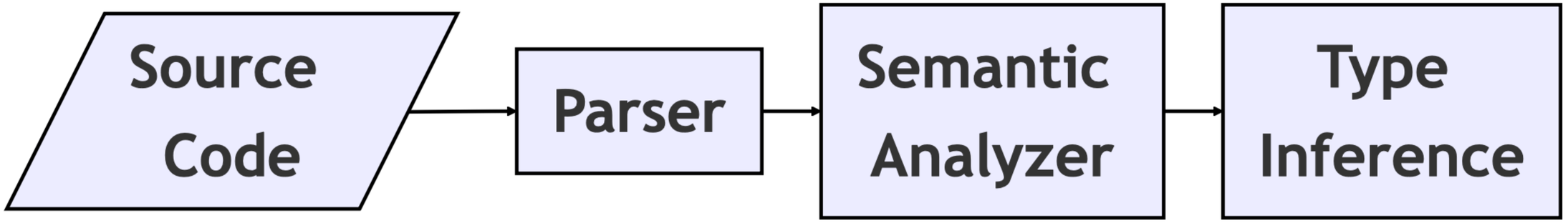
Type checking the home-assistant project on the command-line, without caching (M4).

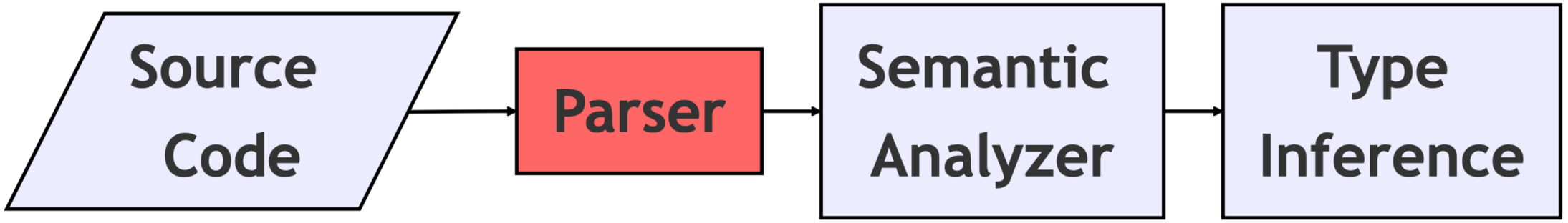
```
def square(x: int):
    return x * x

square("10")
```

```
4| square("10")
 |        ^^^^ Expected `int`, found `Literal["10"]`
 |
info: Function defined here
 --> example.py:1:5
  |
1| def square(x: int) -> int:
 |     ^^^^^^ Parameter declared here
```

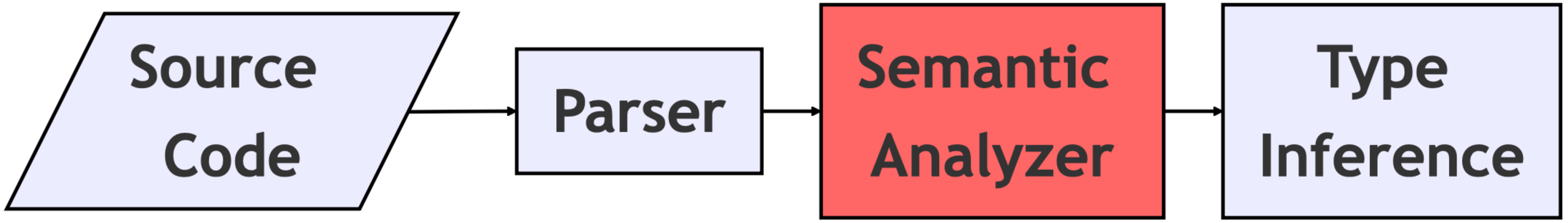
https://play.ty.dev/70b371db-9910-4e1b-8b44-636faf8745b9


Source Code
Parser
Semantic Analyzer
Type Inference

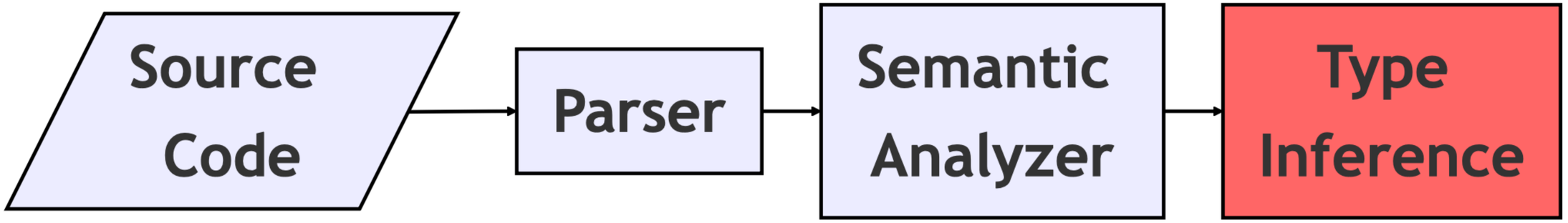



```
x: int = 1
```

```
AnnotatedAssign {
    target: "x",
    annotation: "int",
    value: 1,
}
```




```
Definition {
  kind: AnnotatedAssignment {
    target: "x",
    annotation: "int",
    value: Some(1),
  }
  ...
}
```




```
Inference {
   expressions: {
      int =>
ClassLiteral(int),
      1 => IntLiteral(1),
      x => IntLiteral(1),
   }
}
```

Semantic index

```
class A:
    def foo(self, x: int | None):
        ...
```

- Scopes: class A, method foo

```
IndexVec<FileScopeId, Scope>
```

```
def foo(x: int | None):
    if x is not None:
        b = x # int
```

- Definitions

  `FxHashMap<NodeKey, Definitions>`

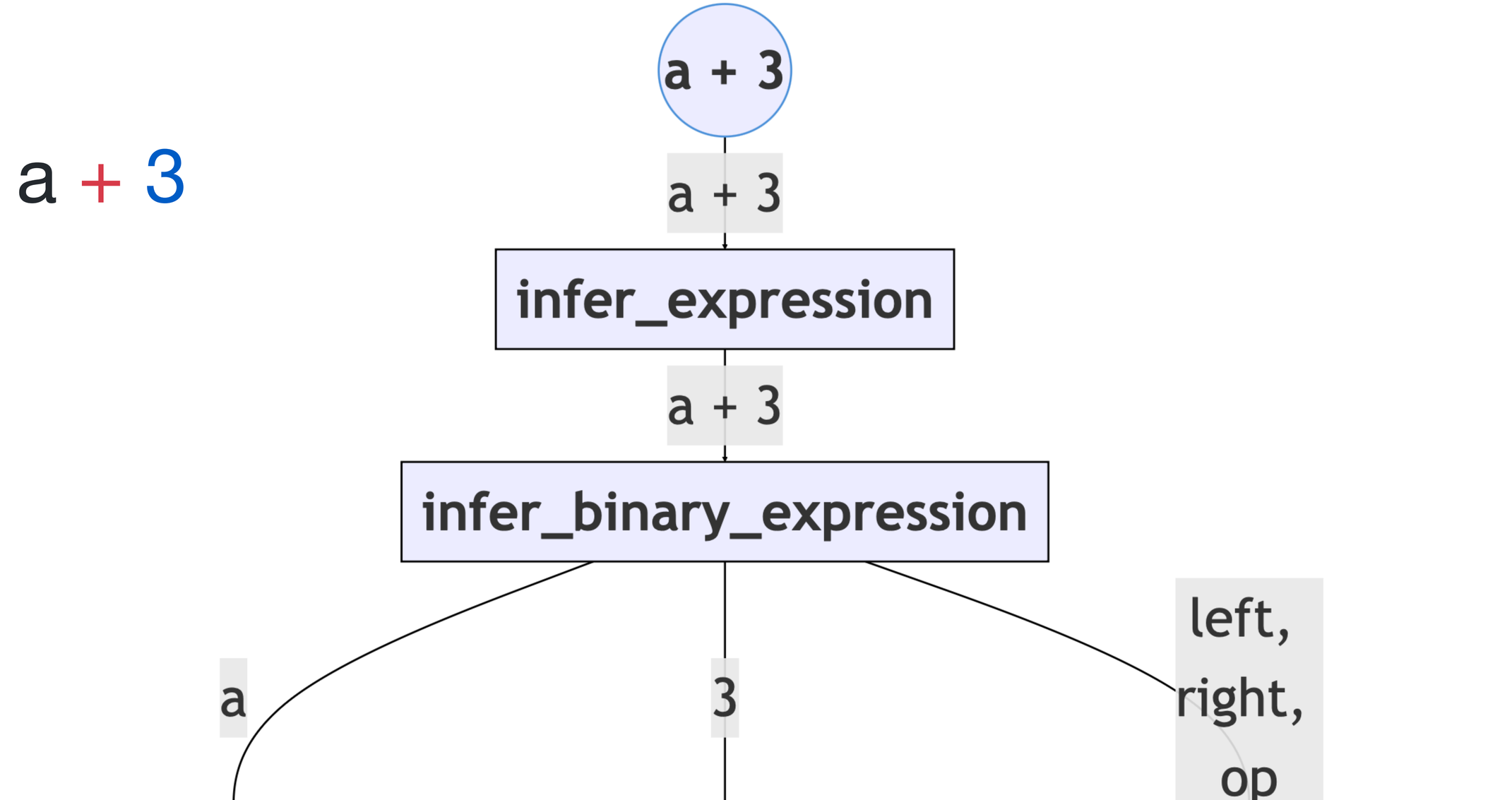
- Constraints: Narrowing and Reachability

  `IndexVec<FileScopeId, UseDefMap>`

Type Inference

a = 1

result = a + 3


a + 3
a + 3
infer_expression
a + 3
infer_binary_expression
a
3
left,
right,
op
infer_expression
infer_expression
infer_binary_result

a = 1

```
let ty = match expression {
    Expr::NumberLiteral(literal) => {
        self.infer_number_literal(literal)
    },
    ...
}
```

```
result = a + 3
```

```
fn infer_binary_expression(…) -> Type {
  let left_ty = self.infer_expr(left);
  let right_ty = self.infer_expr(right);
  self.infer_binary_result(
    left_ty, right_ty, op
  )
}
```

result = a + 3

```
match (left_ty, right_ty, op) {
(Type::IntLiteral(n), IntLiteral(m), Add)
=> IntLiteral(n + m),
    …
}
```

Incremental Type Checking

main.py

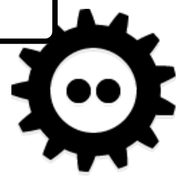
```
from lib import
square

print(square(5))
```

lib.py

```
def square(
  x: int
):
  return x * x


def sqrt(
  x: float
): ...
```



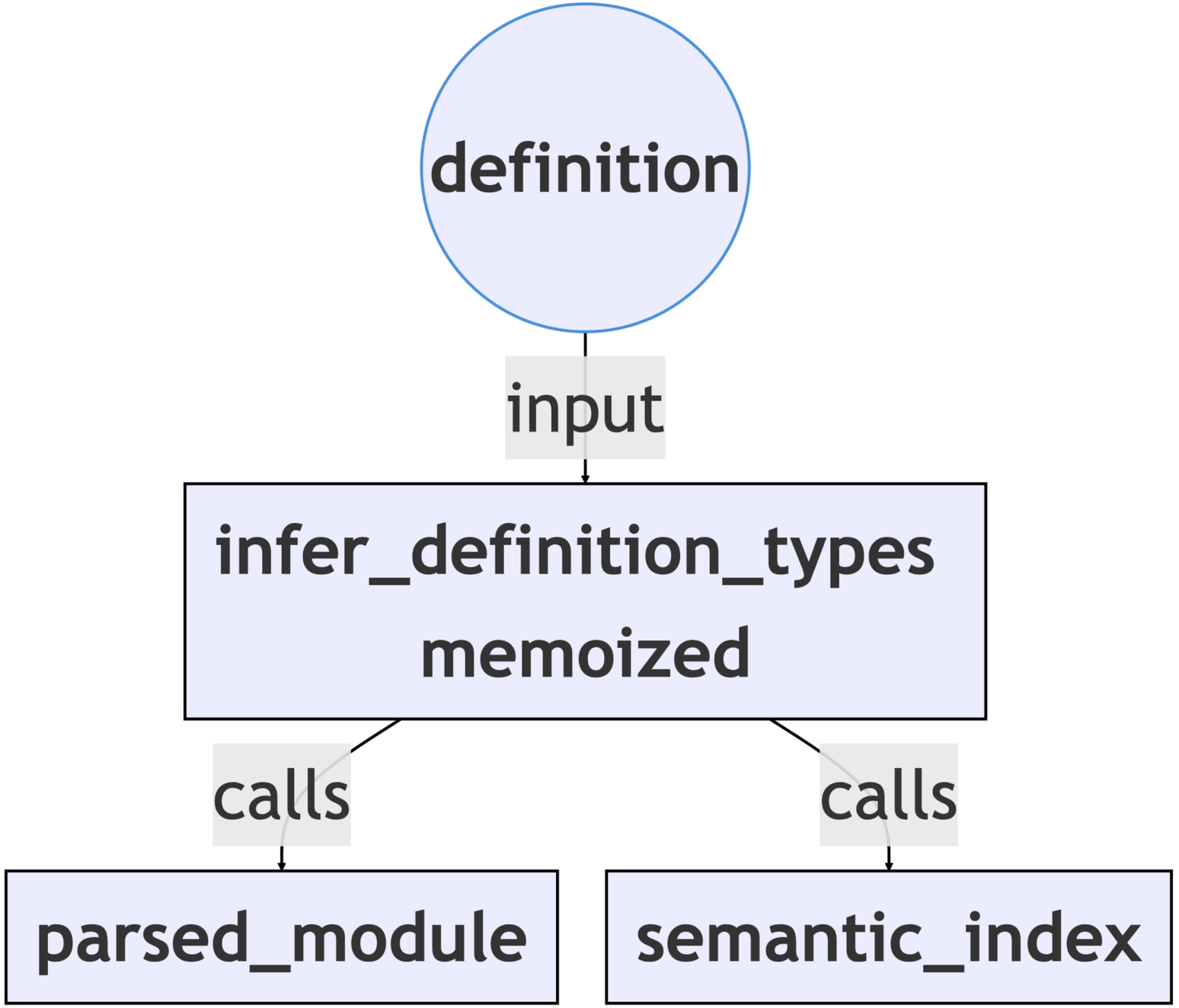
Checkout my new single it’s called my

```
#[salsa::tracked]
fn infer_definition_types<'db>(
    db: &Db, def: Def,
) -> DefinitionInference<'db> {
    let file = def.file(db);
    let module = parsed_module(db,
file).load(db);
    let index = semantic_index(db, file);
    ...
}
```

```
#[salsa::tracked]
fn infer_definition_types<'db>(
    db: &Db, def: Def,
) -> DefinitionInference<'db> {
    let file = def.file(db);
    let module = parsed_module(db,
file).load(db);
    let index = semantic_index(db, file);
    ...
}
```

```
#[salsa::tracked]
fn infer_definition_types<'db>(
    db: &Db, def: Def,
) -> DefinitionInference<'db> {
    let file = def.file(db);
    let module = parsed_module(db,
file).load(db);
    let index = semantic_index(db, file);
    ...
}
```

```
#[salsa::tracked]
fn infer_definition_types<'db>(
    db: &Db, def: Def,
) -> DefinitionInference<'db> {
    let file = def.file(db);
    let module = parsed_module(db,
file).load(db);
    let index = semantic_index(db, file);
    ...
}
```


definition
input
infer_definition_types
memoized
calls
calls
parsed_module
semantic_index

```
#[salsa::db]
pub struct ProjectDatabase {
    storage: Storage<ProjectDatabase>
}
```

```rust
#[salsa::input]
pub struct File

#[salsa::interned]
pub struct FunctionType<'db>


#[salsa::tracked]
fn semantic_index(db: &'db dyn Db, …)
```

- Salsa structs are `Copy` no matter what fields they contain
- Most structs get the `'db` lifetime

```
pub enum Type<'db> {
    Any,
    Never,
    FunctionLiteral(FunctionType<'db>),
    ...
}
```

Compute Split

```
enum InferenceRegion {
    Expression(Expression),
    Definition(Definition),
    Deferred(Definition),
    Scope(ScopeId),
}
```

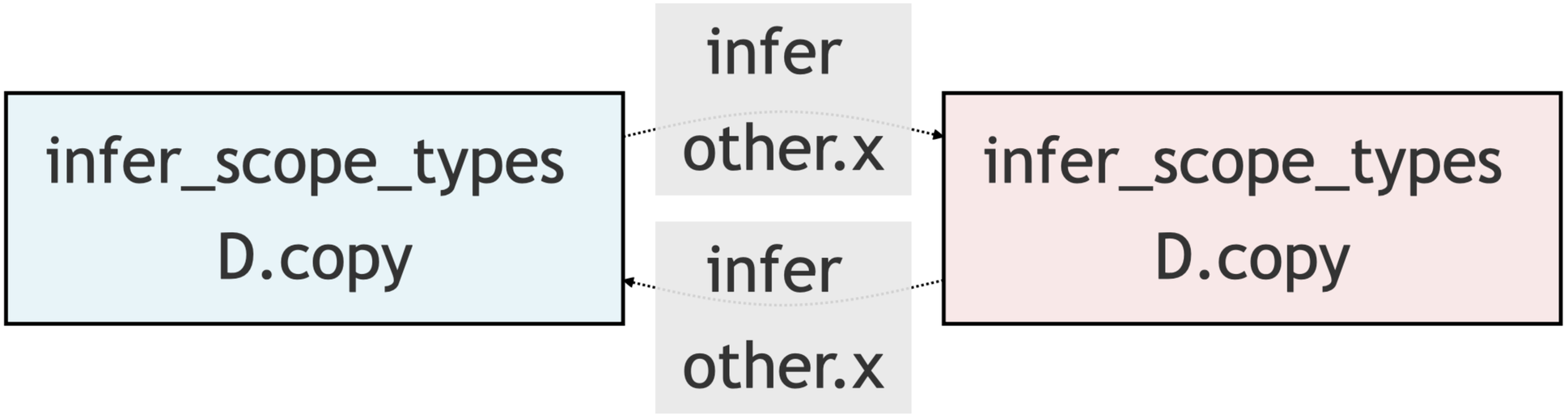
Compute Split

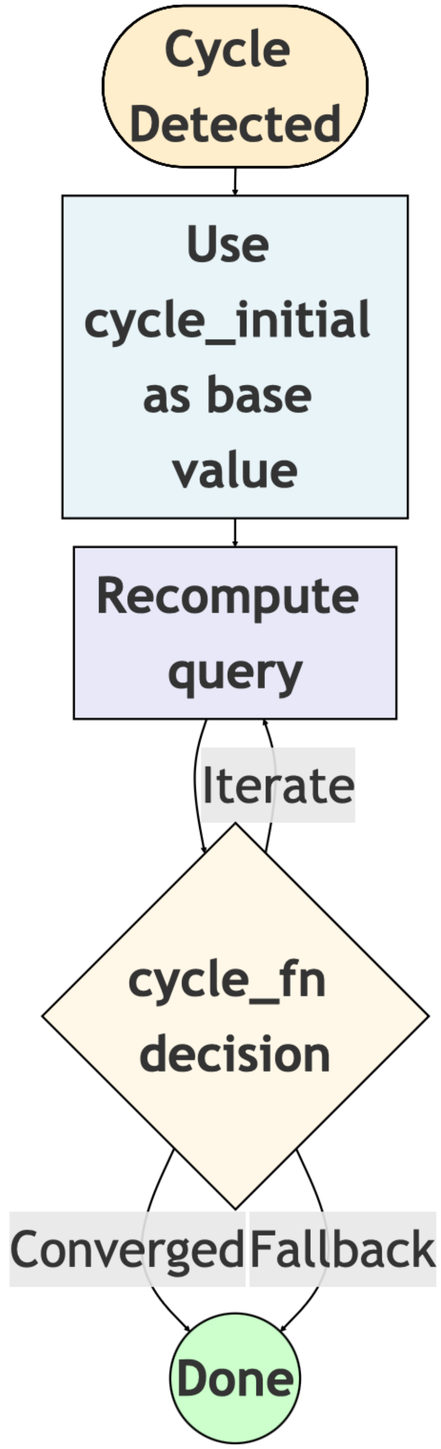
```
#[salsa::tracked]
fn place_table(db: &dyn Db, scope: ScopeId)

#[salsa::tracked]
fn infer_definition_types(
  db: &dyn Db, definition: Definition
)
```

Cycles

```
class D:
    def copy(self, other: "D"):
        self.x = other.x
```






```rust
#[salsa::tracked(
    cycle_fn=...,
    cycle_initial=...,
)]
fn infer_definition_types(...)
```

- `cycle_initial`: Uses `Type::Never`
- `cycle_fn`: Decides to iterate or fallback

Testing Setup

```
## Annotation only transparent to local
inference
```py
x = 1
x: int
y = x
# This case is incorrect
reveal_type(y)  # revealed: int
reveal_type(y)  # revealed: Literal[1]
```
```

Testing Setup

```
$ cargo test
annotations.md:9
unexpected error: [revealed-type]
`Literal[1]`
```

PR checks

- Conformance tests from Python typing team
- Runs ty on open source projects tracking
  - Diagnostic counts
  - Memory usage & timing

Ecosystem report

Old: main (merge base) b80d8ff6 → New: 22495/merge abc00cb7

Lint Rule	Removed	Added	Changed
Total	**126**	**9**	**16**
invalid-argument-type	79	8	2
possibly-missing-attribute	10	0	3
unresolved-attribute	10	0	3
invalid-assignment	6	0	4
invalid-await	9	0	0
invalid-return-type	3	0	4
not-subscriptable	4	0	0
no-matching-overload	2	0	0
unused-ignore-comment	1	1	0
not-iterable	1	0	0
redundant-cast	1	0	0

Thank you!

glyphack.com