# Quiche

## E-Graphs in Python

Rebecca Swords | EGRAPHS 2022

# Introductions

Who am I?

What is Quiche?

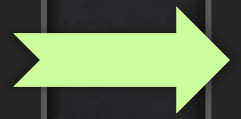
Why is Quiche?

# A Quick Example

```python
def hours_to_seconds(hours):
    return hours * 60 * 60

def days_to_seconds(days):
    return days * 24 * 60 * 60

def seconds_to_days(seconds):
    return seconds / 24 / 60 / 60
```



```python
def hours_to_seconds(hours):
    return hours * 3600

def days_to_seconds(days):
    return days * 86400

def seconds_to_days(seconds):
    return seconds / 86400
```

*Quiche parses the AST and creates an e-graph.*
*Using rewrite rules and e-class analysis, we can do constant folding*

# How?

1. Parse the Python into an ASTQuicheTree and turn it into an e-graph with constant folding analysis.
2. Rewrite rules for multiplication associativity and division rewriting.
3. Apply rules until saturation.
4. Extract the smallest AST.
5. Export to file.

1
```
quiche_tree = ASTQuicheTree("time_conversion.py")
egraph = EGraph(quiche_tree, ASTConstantFolding())
```

2
```
mul_assoc = ASTQuicheTree.make_rule(
    "__quiche__x * __quiche__y * __quiche__z",
    "__quiche__x * (__quiche__y * __quiche__z)")
mul_div = ASTQuicheTree.make_rule(
    "__quiche__x / __quiche__y / __quiche__z",
    "__quiche__x / (__quiche__y * __quiche__z)")
```

3
```
rules = [mul_assoc, mul_div]
while not egraph.is_saturated():
    egraph.apply_rules(rules)
```

4
```
extracted = MinimumCostExtractor().extract(
    ASTSizeCostModel(), egraph, egraph.root,
    ASTQuicheTree.make_node)
```

5
```
extracted.to_file("time_folding.py")
```

# Core Implementation

## Quiche Internals

# Core Components

| | |
|---|---|
| EGraph | E-graph implementation using algorithms described in egg |
| QuicheTree | Tree representation used for building e-graphs, rewriting, and term extraction |
| EClassAnalysis | Running e-class analysis, as described in egg |
| Rule | Rewriting rules, specifying term equivalences |
| CostModel | Calculate cost for each e-node |
| Extractor | Extract terms based on the cost model |

# Core Components

| | |
|---|---|
| EGraph | E-graph implementation using algorithms described in egg |
| QuicheTree | Tree representation used for building e-graphs, rewriting, and term extraction |
| EClassAnalysis | Running e-class analysis, as described in egg |
| Rule | Rewriting rules, specifying term equivalences |
| CostModel | Calculate cost for each e-node |
| Extractor | Extract terms based on the cost model |

# E-Graph Representation

Follows the e-graph algorithms in egg.

Constructor arguments are optional.

`add()` takes a `QuicheTree` object.

```
class EGraph():
  def __init__(self, tree, analysis)
  def add(self, quiche_tree)
  def merge(self, eclass1, eclass2)
  def rebuild(self)
  def repair(self, eclass)
  def apply_rules(self, rules)
```

# Quiche Trees

Quiche requires the user to provide a parsed tree that implements `QuicheTree` (“*bring your own parser*”).

**`value()`**
the e-node key

**`children()`**
list of the node's children

**`is_pattern_symbol()`**
for e-matching; indicates if the node is a pattern

```
class QuicheTree(ABC):
  @abstractmethod
  def value(self)

  @abstractmethod
  def children(self)

  @abstractmethod
  def is_pattern_symbol(self)
```

# Rules and Pattern Symbols

A **`Rule`** is just a pair of **`QuicheTree`**s

For Python ASTs, patterns are variables or strings prefixed with `__quiche__` (more later).

**`make_rule()`** is provided for convenience

```
class Rule:
  def __init__(self, lhs, rhs):
      self.lhs = lhs
      self.rhs = rhs

# Makes a Python AST rule.
mul_assoc = ASTQuicheTree.make_rule(
  "__quiche__x * __quiche__y * __quiche__z",
  "__quiche__x * (__quiche__y * __quiche__z)")
```

# Cost Models

Similar to egg:

**`enode_cost`**: calculate cost based on e-node value

**`enode_cost_rec`**: recursive cost calculation, including children;

costs: [EClassID → (cost, ENode)]

```
class CostModel(ABC):
  @abstractmethod
  def enode_cost(self, enode)

  @abstractmethod
  def enode_cost_rec(self, enode, costs)
```

# Term Extraction

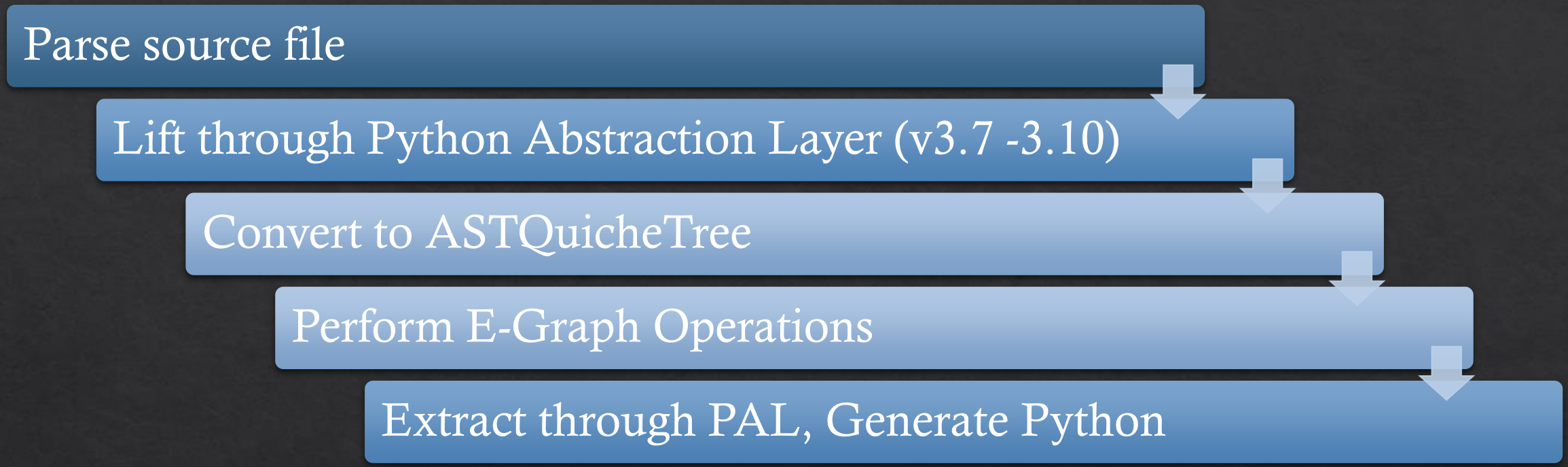
Use a cost model to extract the `QuicheTree` term for an e-class in the e-graph.

Generally, use **`MinimumCostExtractor`**

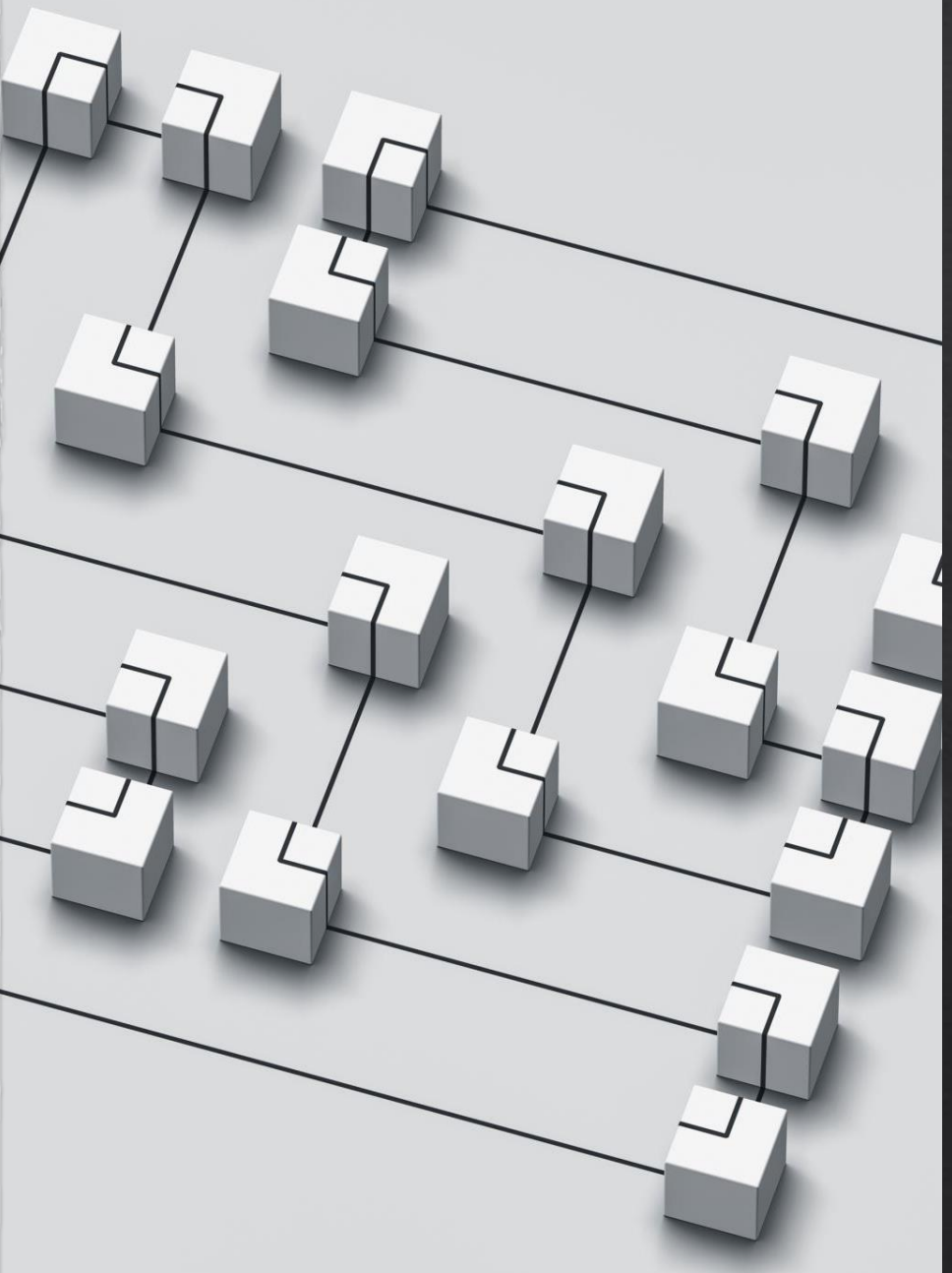
**`build_tree(value, children):`** method for re-constructing a `QuicheTree` from an e-node

```
class CostExtractor(ABC):
  @abstractmethod
  def extract(self, cost_model, egraph,
    eclassid, build_tree)
```

# Python ASTs in Quiche




# Python Pipeline



# Python Abstraction Layer (PAL)

Abstract differences between Python versions for Quiche input and output (versions 3.7 – 3.10 supported)

Ensure all tree nodes extend AST

- Wrap lists of statements, expressions, etc. in blocks
- Collapse constants and vars into a simpler leaf node

# ASTQuicheTree

**`value()`**

**AST Node type or a special tuple for leaves**

`children()`

List of values from each field in `_fields`, wrapped in an `ASTQuicheTree`

`is_pattern_symbol()`

Variables or strings beginning with `__quiche__`

```
def value(self):
  root_type = type(root)
  if is_primitive_type(root_type):
    return (root.kind, root.constr, *root.args)
  else:
    return root_type
```

Examples:

`If(test, body, orelse)` → `If`

`Constant(5, None)` →

`("int", Constant, 5, None)`

# ASTQuicheTree

`value()`

AST Node type or a special tuple for leaves

**`children()`**

**List of values from each field in `_fields`, wrapped in an `ASTQuicheTree`**

`is_pattern_symbol()`

Variables or strings beginning with `__quiche__`

```
self._children = []
for field in getattr(self.root, "_fields", []):
  child = getattr(self.root, field, None)
  self._children.append(ASTQuicheTree(child))
```

Examples:

```
If(test, then_block, else_block) →
  [ASTQuicheTree(test),
   ASTQuicheTree(then_block),
   ASTQuicheTree(else_block)]
```

# ASTQuicheTree

`value()`

AST Node type or a special tuple for leaves

`children()`

List of values from each field in `_fields`, wrapped in an `ASTQuicheTree`

**`is_pattern_symbol()`**

**Variable names or strings beginning with `__quiche__`**

Examples:

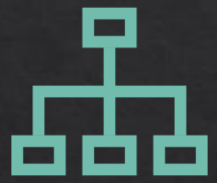
`Constant("__quiche__x", None)`

`Name("__quiche__y", Load)`

→ Why not "?x" like egg?

→ Why include strings?

# Extraction



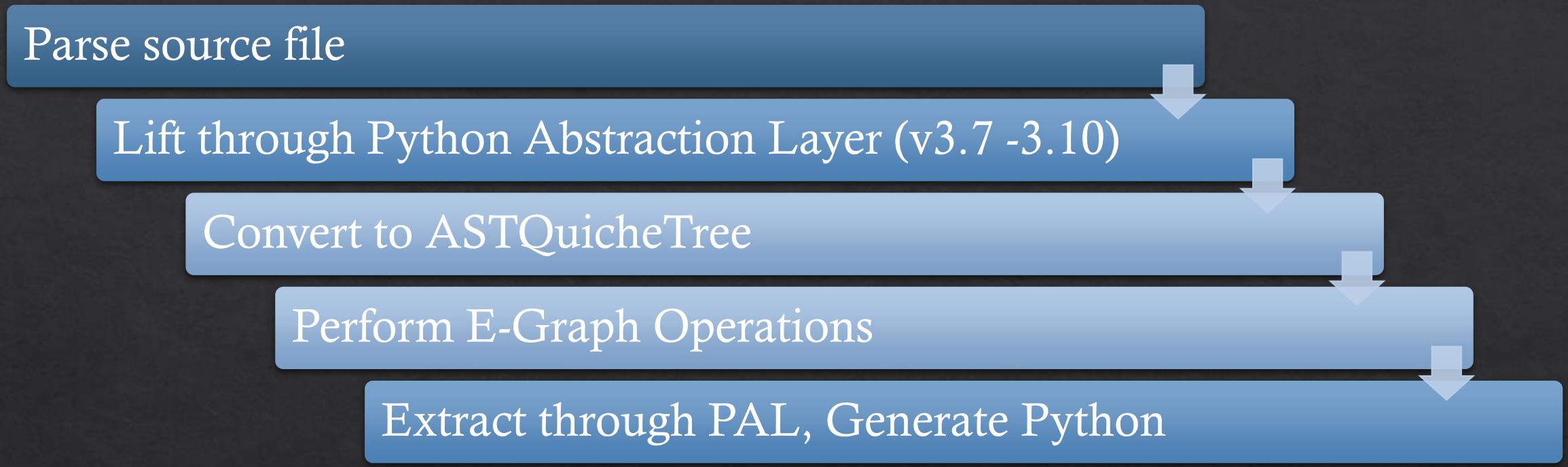
**Rebuild ASTQuicheTree**

**Extract through PAL**

Remove extra blocks

Re-expand leaf nodes

**Generate source code with `astor`**




# Python Pipeline

# Future Work

- Implement new analyses and cost models
- Create a full suite of rewrite rules for known Python optimizations
- Add better project support
- Explore options for legacy Python systems

# Thank You!

github.com/riswords/quiche

- Parse source file
- Lift through Python Abstraction Layer (v3.7 -3.10)
- Convert to ASTQuicheTree
- Perform E-Graph Operations
- Extract through PAL, Generate Python

# Python 3.9 AST Highlights

https://docs.python.org/3.9/library/ast.html

ASDL's 4 builtin types are: identifier, int, string, constant

```
mod = Module(stmt* body, type_ignore* type_ignores)
      | Expression(expr body)



stmt = FunctionDef(identifier name, arguments args,
           stmt* body, expr* decorator_list,
           expr? returns, string? type_comment)
      | Assign(expr* targets, expr value,
           string? type_comment)
      | While(expr test, stmt* body, stmt* orelse)
      | If(expr test, stmt* body, stmt* orelse)
      | Expr(expr value)
```

# Python 3.9 AST Highlights

https://docs.python.org/3.9/library/ast.html

```
expr = BinOp(expr left, operator op, expr right)
     | UnaryOp(unaryop op, expr operand)
     | Lambda(arguments args, expr body)
     | Constant(constant value, string? kind)
     | Name(identifier id, expr_context ctx)
```