ZipPy on Truffle

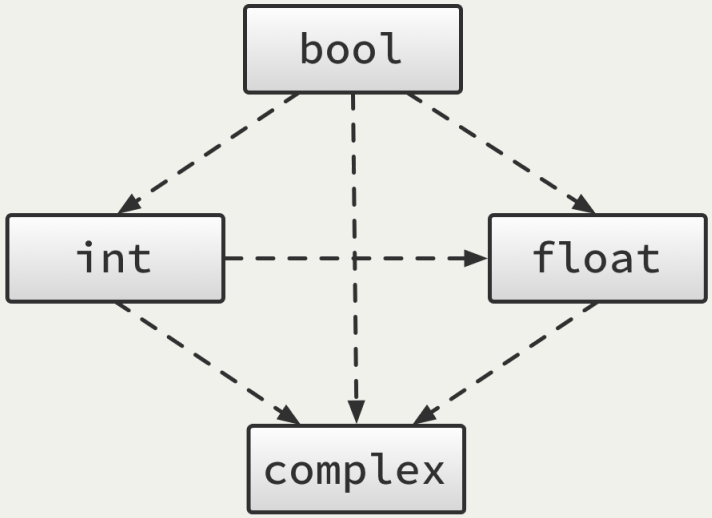
Wei Zhang
University of California, Irvine

is?

- Python 3 using Truffle
- 80% language completeness
- https://bitbucket.org/ssllab/zippy

Trufflization Generators Performance

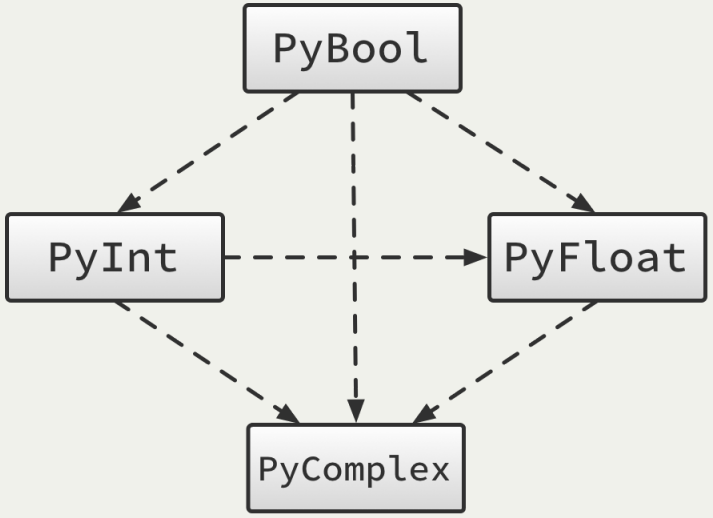
Numeric Types



int has arbitrary precision

type coercion

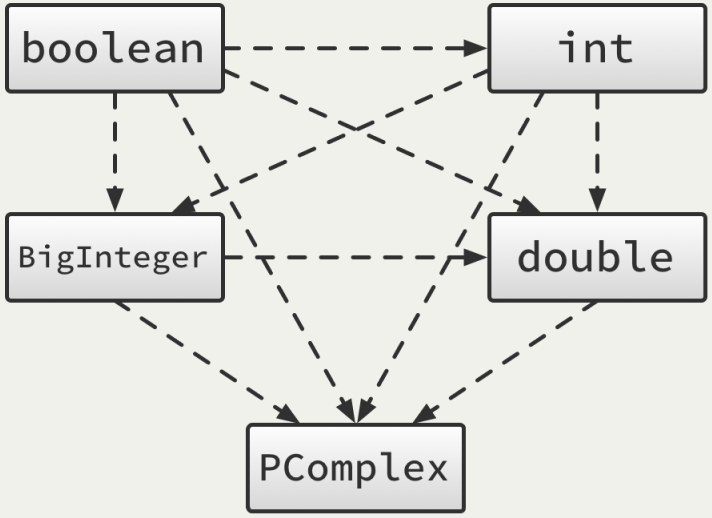
CPython



PyInt has arbitrary precision

type coercion

ZipPy



type coercion

Using Truffle DSL

```
@Specialization(order = 0)
int doBoolean(boolean left, boolean right) {
    final int leftInt = left ? 1 : 0;
    final int rightInt = right ? 1 : 0;
    return leftInt + rightInt;
}

@Specialization(rewriteOn = ArithmeticException.class, order = 1)
int doBoolean(int left, boolean right) {
    final int rightInt = right ? 1 : 0;
    return ExactMath.addExact(left, rightInt);
}

@Specialization(rewriteOn = ArithmeticException.class, order = 2)
int doBoolean(boolean left, int right) {
    final int leftInt = left ? 1 : 0;
    return ExactMath.addExact(leftInt, right);
}

@Specialization(rewriteOn = ArithmeticException.class, order = 5)
int doInteger(int left, int right) {
    return ExactMath.addExact(left, right);
}

@Specialization(order = 6)
BigInteger doIntegerBigInteger(int left, BigInteger right) {
    return BigInteger.valueOf(left).add(right);
}

@Specialization(order = 7)
BigInteger doBigIntegerInteger(BigInteger left, int right) {
    return left.add(BigInteger.valueOf(right));
}

@Specialization(order = 10)
BigInteger doBigInteger(BigInteger left, BigInteger right) {
    return left.add(right);
}

@Specialization(order = 13)
```

Sequence Types

- range: generate indices
- list: mutable, likely homogeneous
- tuple: immutable, heterogeneous

for range loop

```
def sum(n):
  ttl = 0
  for i in range(n):
    ttl += i
  return ttl
```

```
@Specialization(order = 1)
public Object doPRange(VirtualFrame frame, PRangeIterator range) {
    final int start = range.getStart();
    final int stop = range.getStop();
    final int step = range.getStep();
    for (int i = start; i < stop; i += step) {
        ((WriteNode) target).executeWrite(frame, i);
        body.executeVoid(frame);
    }

    return PNone.NONE;
}
```

for range loop

```python
def sum(n):
  ttl = 0
  for i in range(n):
    ttl += i
  return ttl
```

```java
public int sum(int n) {
    int ttl = 0;

    for (int i = 0; i < n; i ++) {
        ttl += i;
    }

    return ttl;
}
```

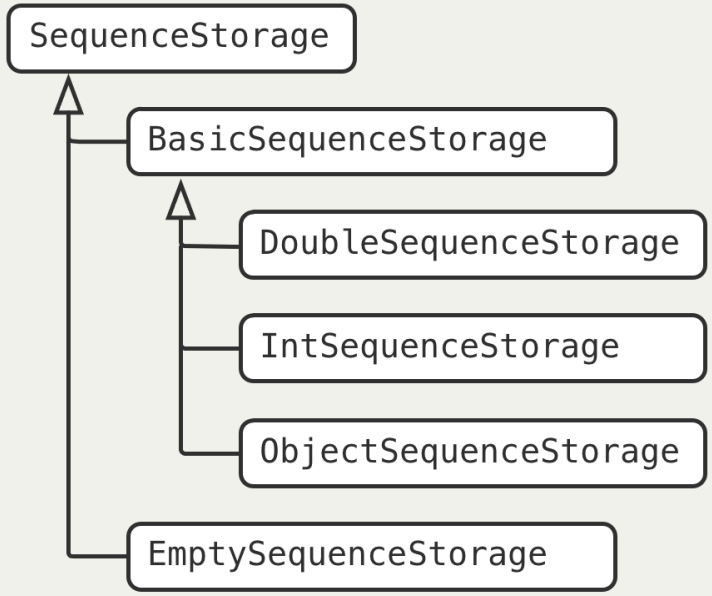
for range loop

```
def sum(n):
  ttl = 0
  for i in range(n):
    ttl += i
  return ttl
```

```
        jmp L7

L6:   mov       ecx, edx
      add       ecx, ebp
      jo        L8
      mov       edx, ebp
      incl      edx
      mov       esi, ebp
      mov       ebp, edx
      mov       edx, ecx
L7:   cmp       eax, ebp
      jle       L9
      jmp       L6
L8:   call      deoptimize()

L9:
```

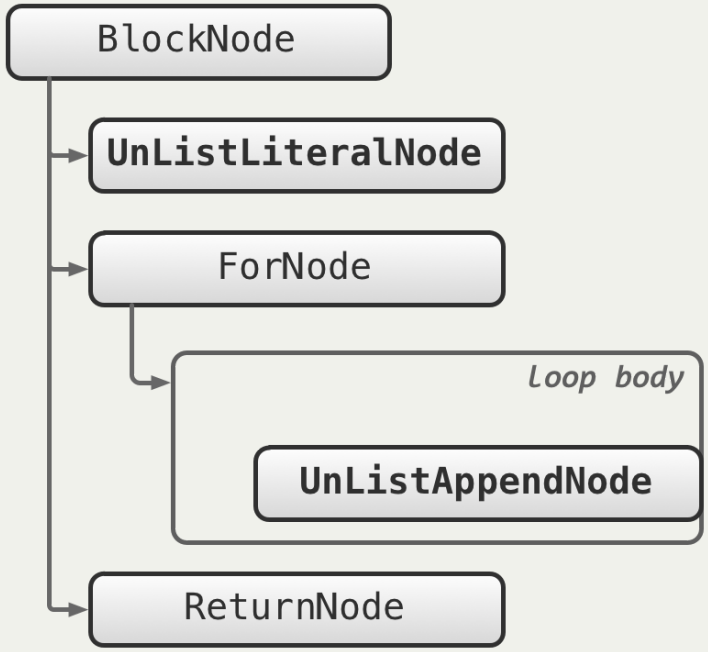
List Storages

```
def makelist(n):
  lst = []
  for i in range(n):
    lst.append(i)
  return lst
```



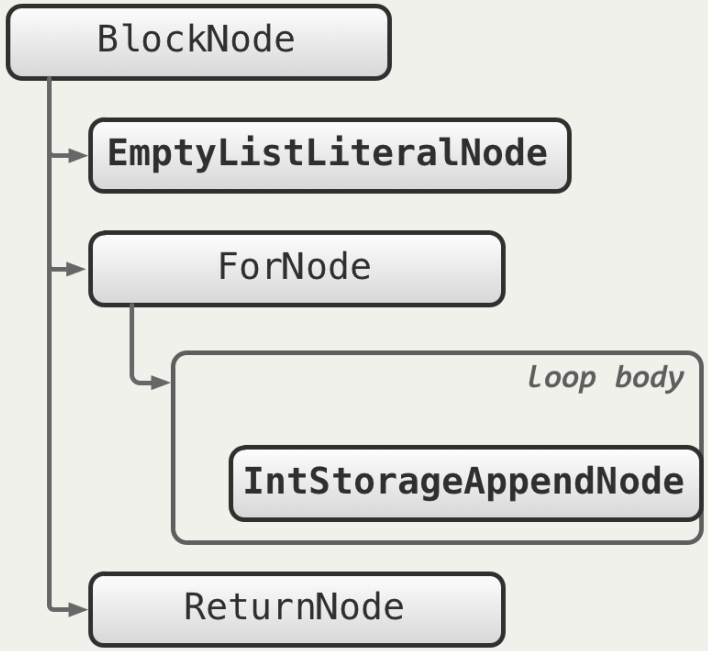
List Storages

```
def makelist(n):
  lst = []
  for i in range(n):
    lst.append(i)
  return lst
```



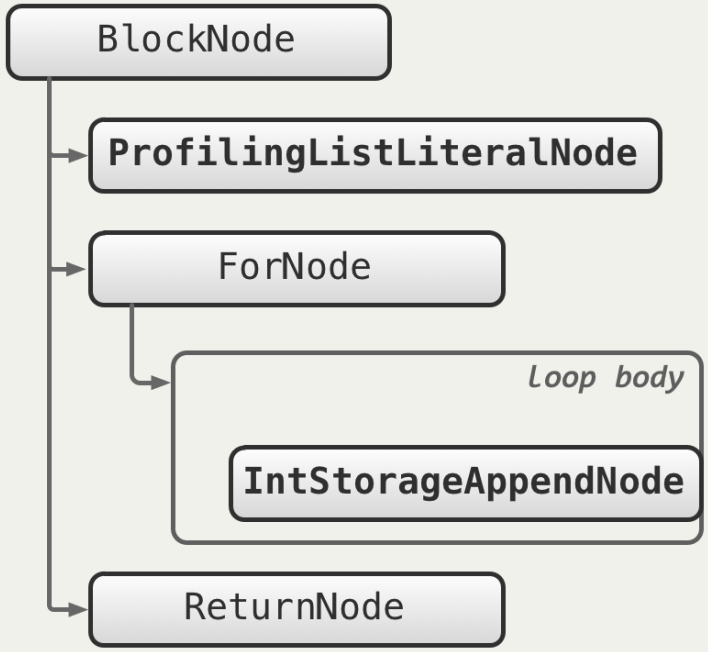
List Storages

```
def makelist(n):
  lst = []
  for i in range(n):
    lst.append(i)
  return lst
```



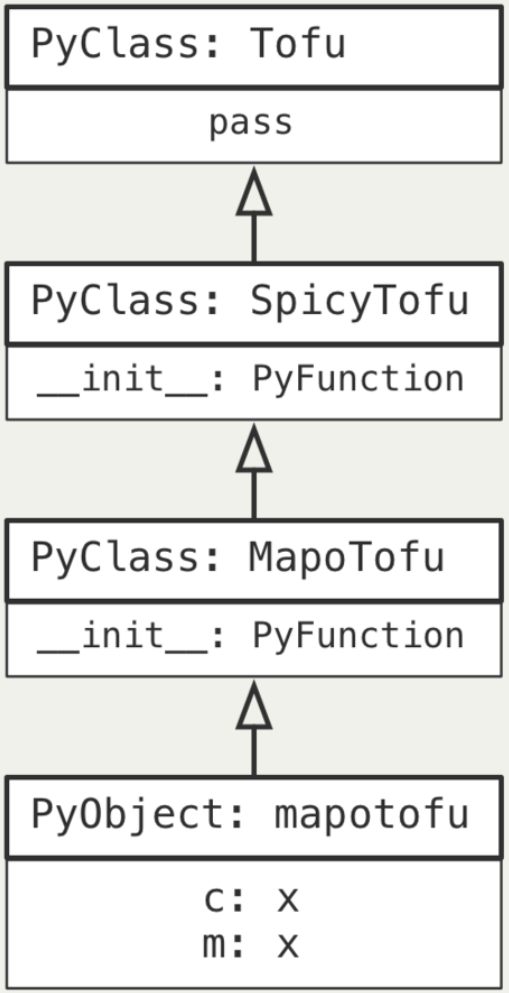
List Storages

```
def makelist(n):
  lst = []
  for i in range(n):
    lst.append(i)
  return lst
```



Object Model

```
class Tofu:
  pass

class SpicyTofu(Tofu):
  def __init__(self, c):
    self.c = c

class MapoTofu(SpicyTofu):
  def __init__(self, c, m):
    SpicyTofu.__init__(self,c)
    self.m = m
```



Object Model

```
class Tofu:
  pass

class SpicyTofu(Tofu):
  def __init__(self, c):
    self.c = c

class MapoTofu(SpicyTofu):
  def __init__(self, c, m):
    SpicyTofu.__init__(self,c)
    self.m = m
```

```
public class FixedPythonObjectStorage
    extends PythonObject {

    protected Object[] arrayObjects;
    protected int primitiveInt0;
    protected int primitiveInt1;
    protected int primitiveInt2;
    protected int primitiveInt3;
    protected int primitiveInt4;

    protected double primitiveDouble0;
    protected double primitiveDouble1;
    protected double primitiveDouble2;
    protected double primitiveDouble3;
    protected double primitiveDouble4;

    protected Object fieldObject0;
    protected Object fieldObject1;
    protected Object fieldObject2;
    protected Object fieldObject3;
    protected Object fieldObject4;
    ...
```

Object Model

```
class Tofu:
  pass

class SpicyTofu(Tofu):
  def __init__(self, c):
    self.c = c

class MapoTofu(SpicyTofu):
  def __init__(self, c, m):
    SpicyTofu.__init__(self,c)
    self.m = m
```

```
public class MapoTofuObjectStorage
    extends PythonObject {

    protected Object[] arrayObjects;
    protected int cInt;
    protected int mInt;

    ...
```

Generators

```
def producer(n):
  for i in range(n):
    yield i

for i in producer(3):
  print(i)

# 0, 1, 2
```

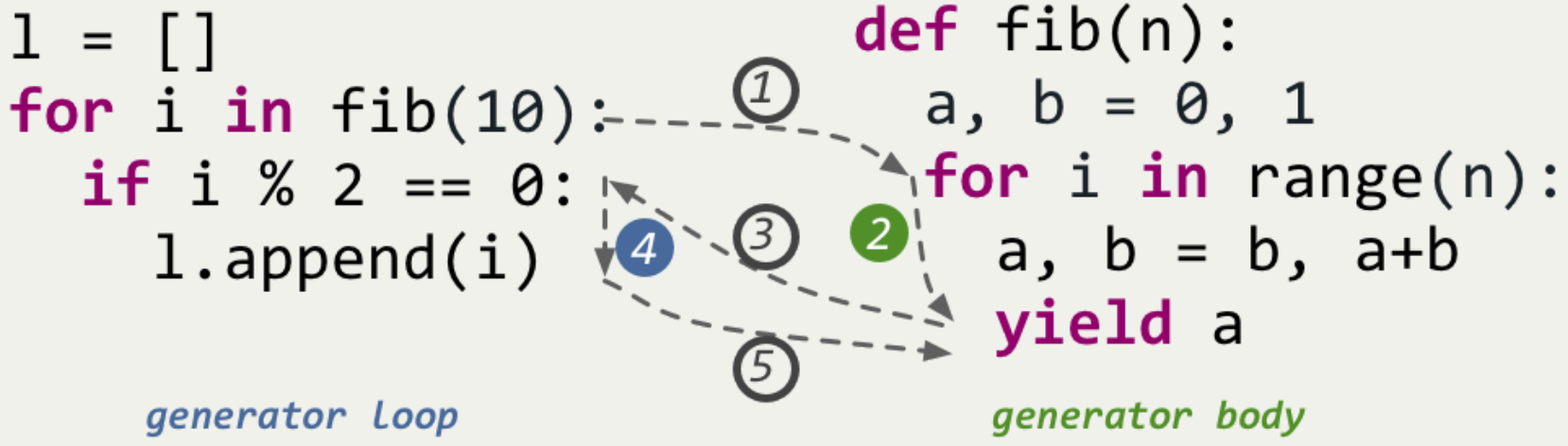
```
g = producer(3)
try:
  while True:
    print(g.__next__())
except StopIteration:
  pass

# 0, 1, 2
```

Generator Expressions

```
n = 3
g =(x for x in range(n))
sum(g)
# 3
```

```
def _producer():
  for x in range(n):
    yield x
_producer()
```

Execution Order



The Problems?

- **suspend** and **resume** prevent frame optimizations
- the __**next**__ call is expensive
- 90% of the top 50 Python projects on PyPI and GitHub use generators

Bytecode Interpreter

- iterative
- store control-flow state in bytecode index

AST Interpreter

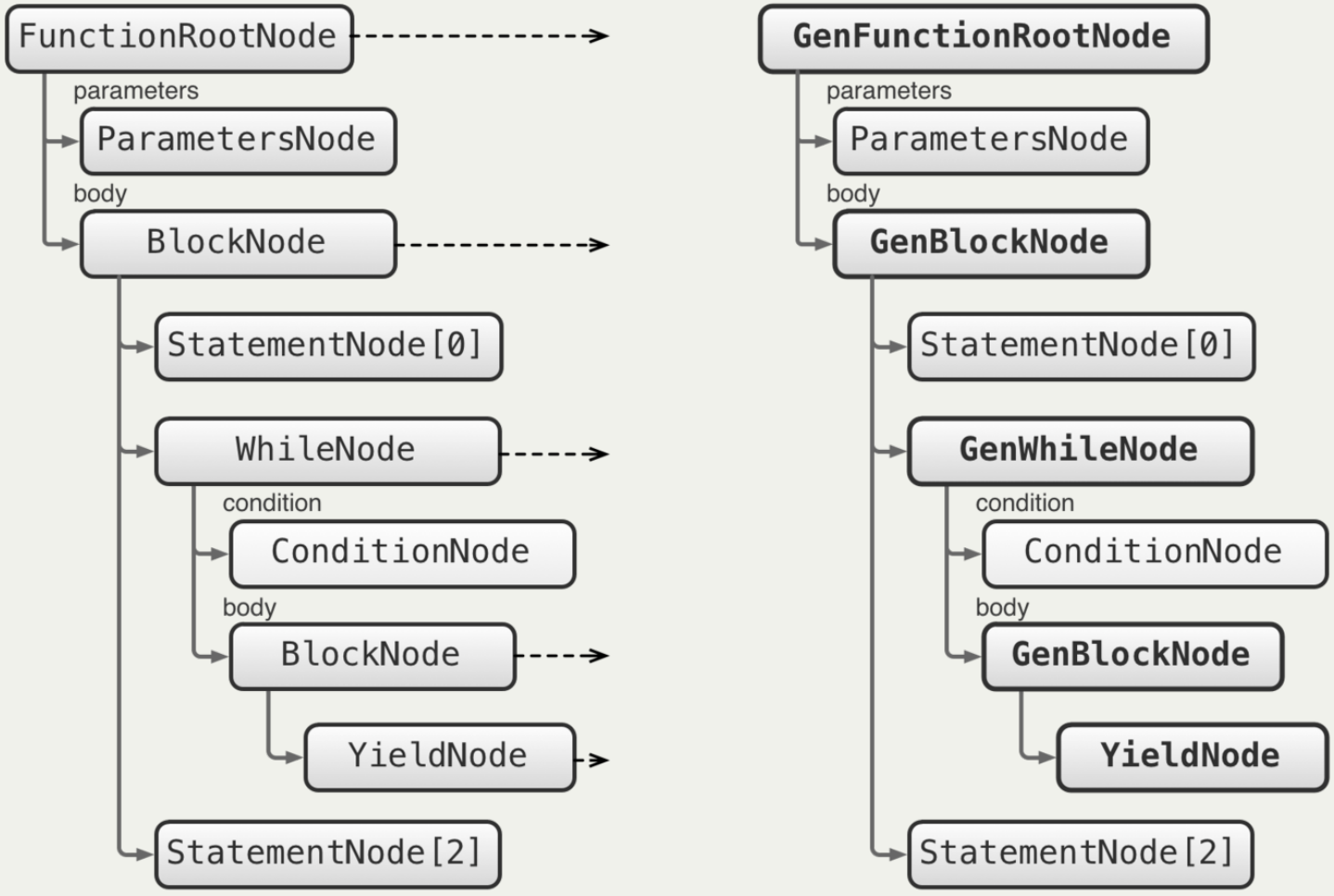
- recursive
- store control-flow state on the call stack

Generator AST

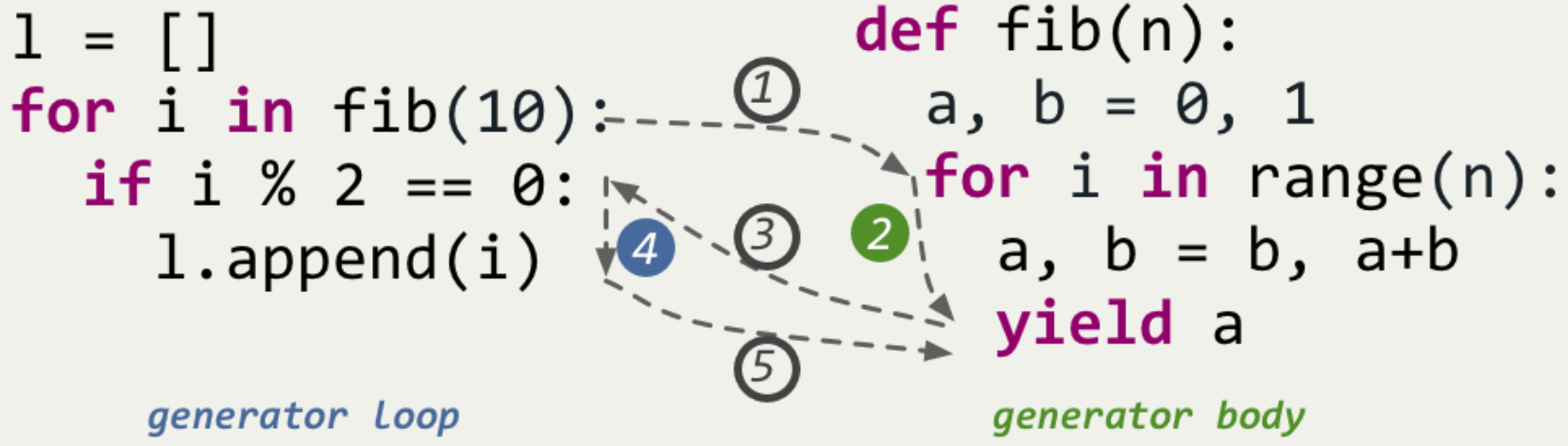
```
class WhileNode extends PNode {
  protected ConditionNode condition;
  protected PNode body;

  public Object execute(Frame frame) {
    try {
      while(condition.execute(frame)) {
        body.execute(frame);
      }
    } catch (BreakException e) {
      // break the loop
    }
    return PNone.NONE;
  }
}
```

```
class GenWhileNode extends WhileNode {
  private final int flagSlot;

  public Object execute(Frame frame) {
    try {
      while(isActive(frame) ||
            condition.execute(frame)) {
        setActive(frame, true)
        body.execute(frame);
        setActive(frame, false);
      }
    } catch (BreakException e) {
      setActive(frame, false);
    }
    return PNone.NONE;
  }
}
```

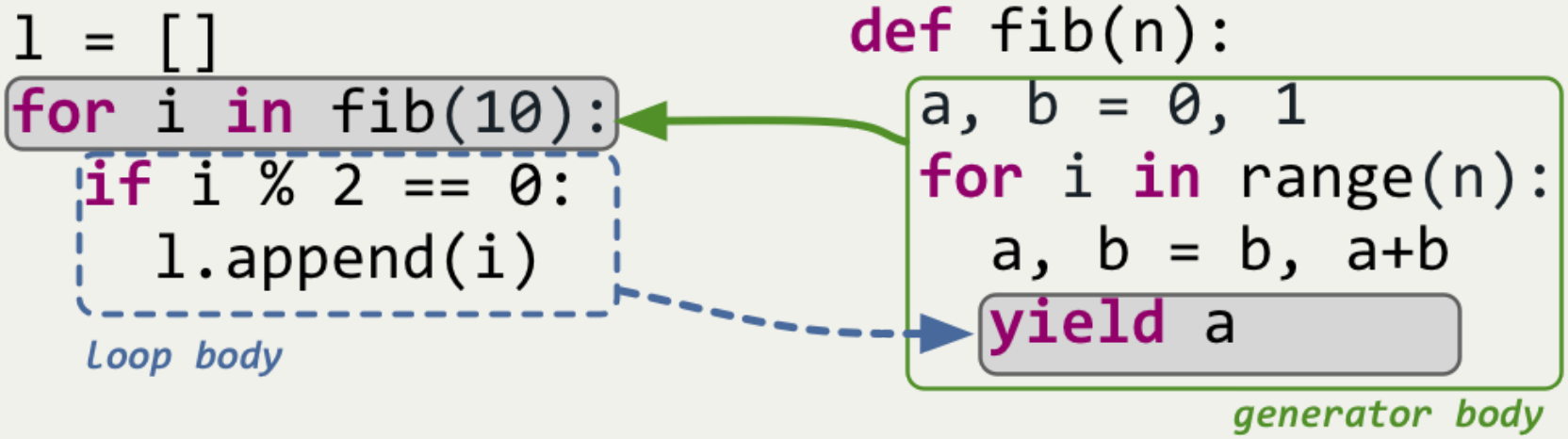
Generator AST



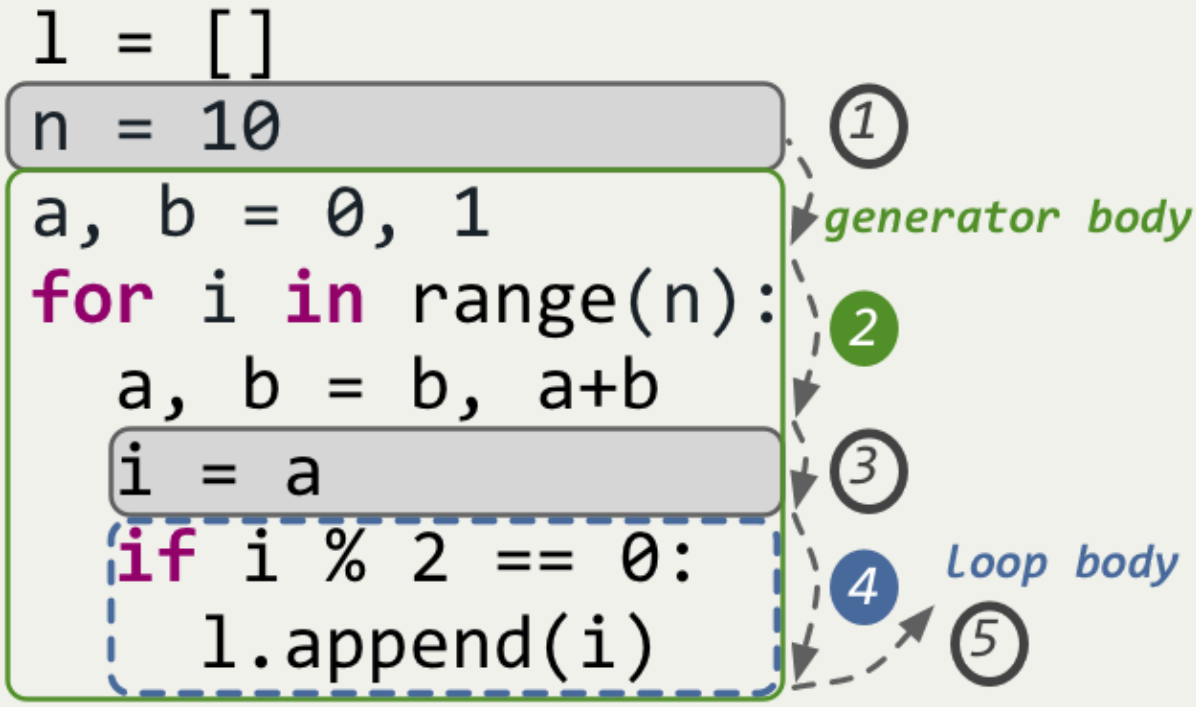
Generator Peeling

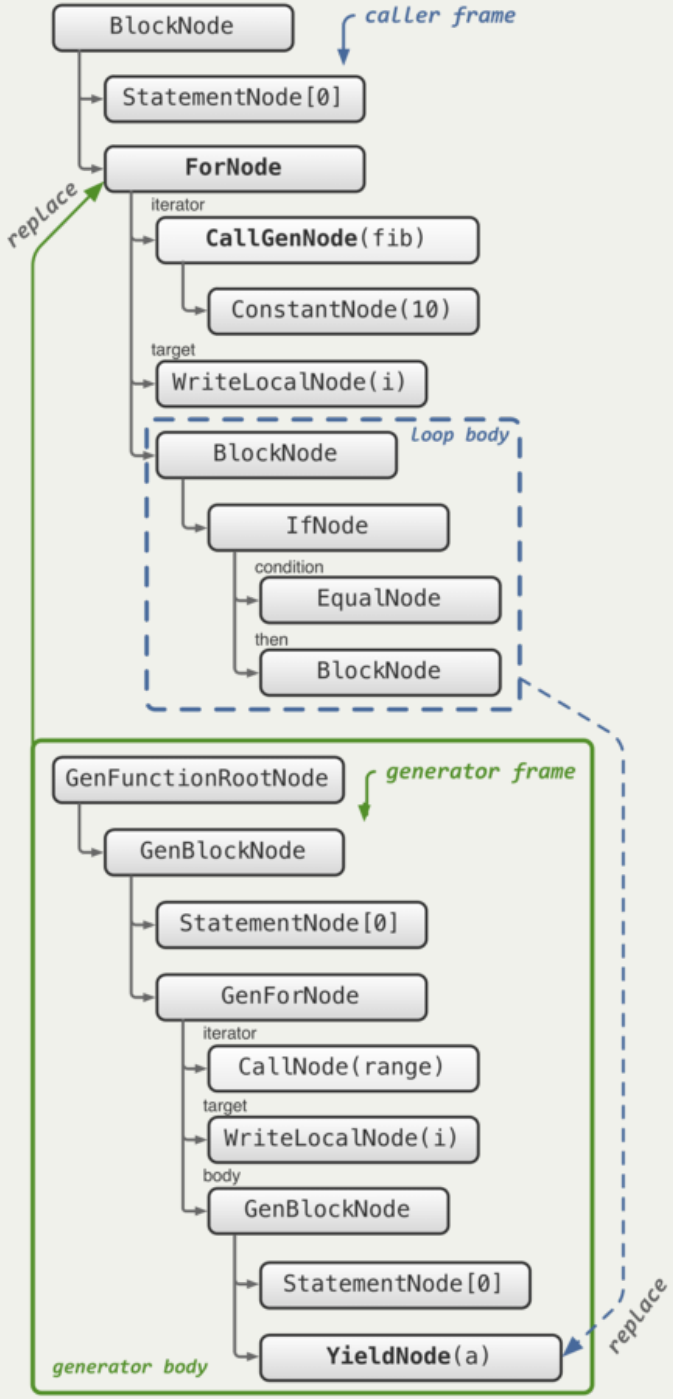


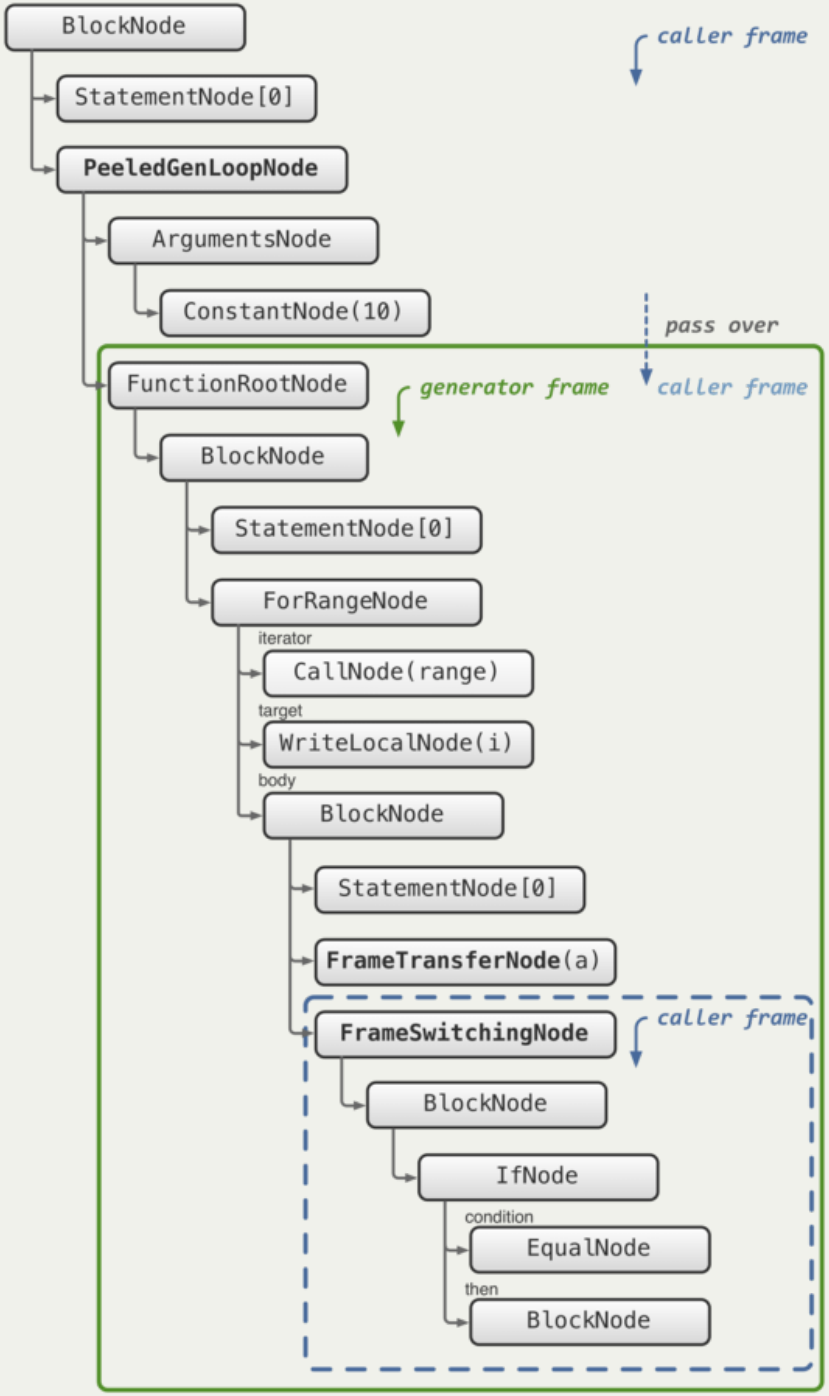
Generator Peeling



Generator Peeling




BlockNode
caller frame
StatementNode[0]
ForNode
replace
iterator
CallGenNode(fib)
ConstantNode(10)
target
WriteLocalNode(i)
loop body
BlockNode
IfNode
condition
EqualNode
then
BlockNode
GenFunctionRootNode
generator frame
GenBlockNode
StatementNode[0]
GenForNode
iterator
CallNode(range)
target
WriteLocalNode(i)
body
GenBlockNode
StatementNode[0]
YieldNode(a)
replace
generator body


BlockNode
caller frame
StatementNode[0]
PeeledGenLoopNode
ArgumentsNode
ConstantNode(10)
pass over
FunctionRootNode
generator frame
caller frame
BlockNode
StatementNode[0]
ForRangeNode
iterator
CallNode(range)
target
WriteLocalNode(i)
body
BlockNode
StatementNode[0]
FrameTransferNode(a)
FrameSwitchingNode
caller frame
BlockNode
IfNode
condition
EqualNode
then
BlockNode

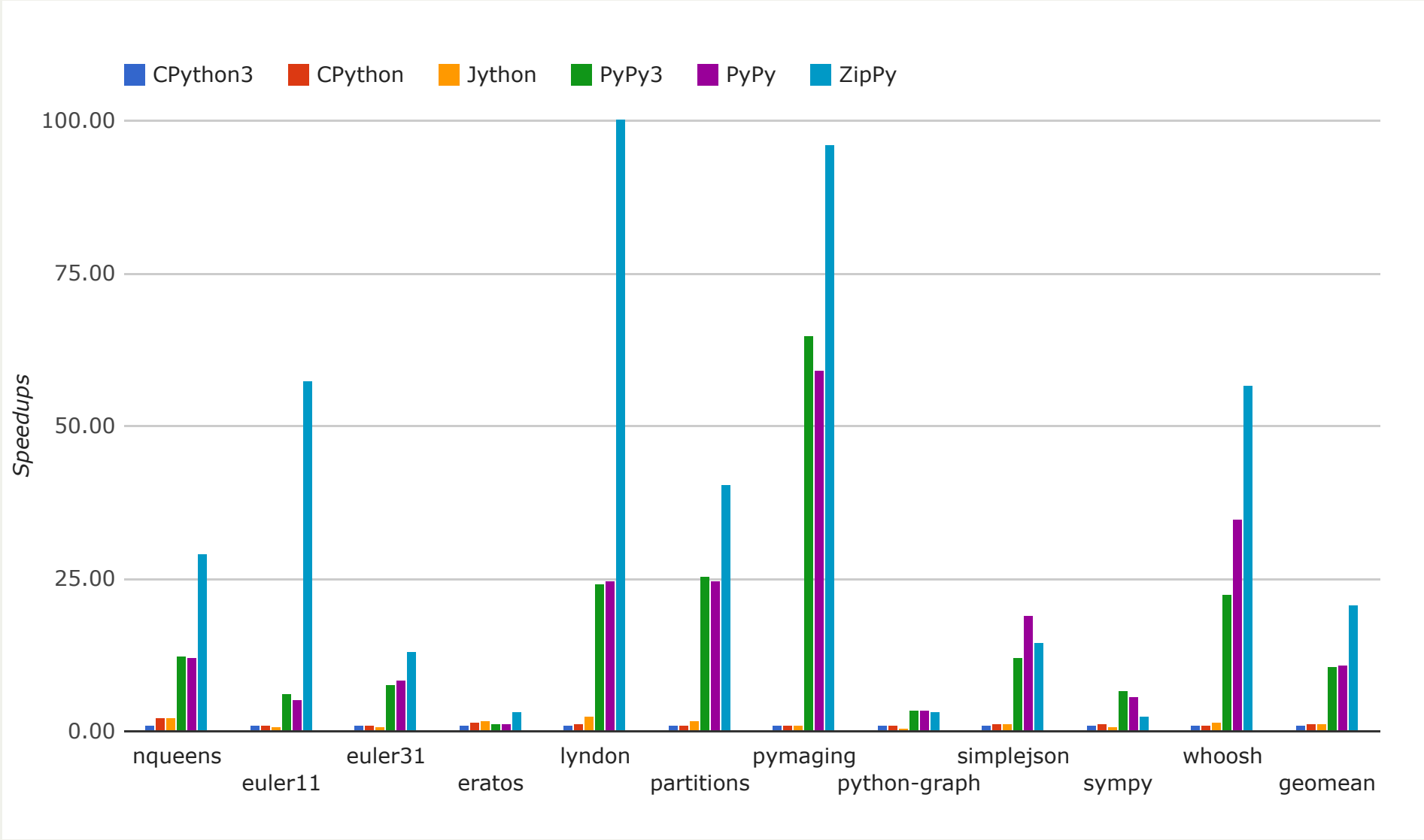
The End Result

- **caller** frame and **generator** frame can be optimized
- peeling inlines the **__next__** call
- no generator AST nodes

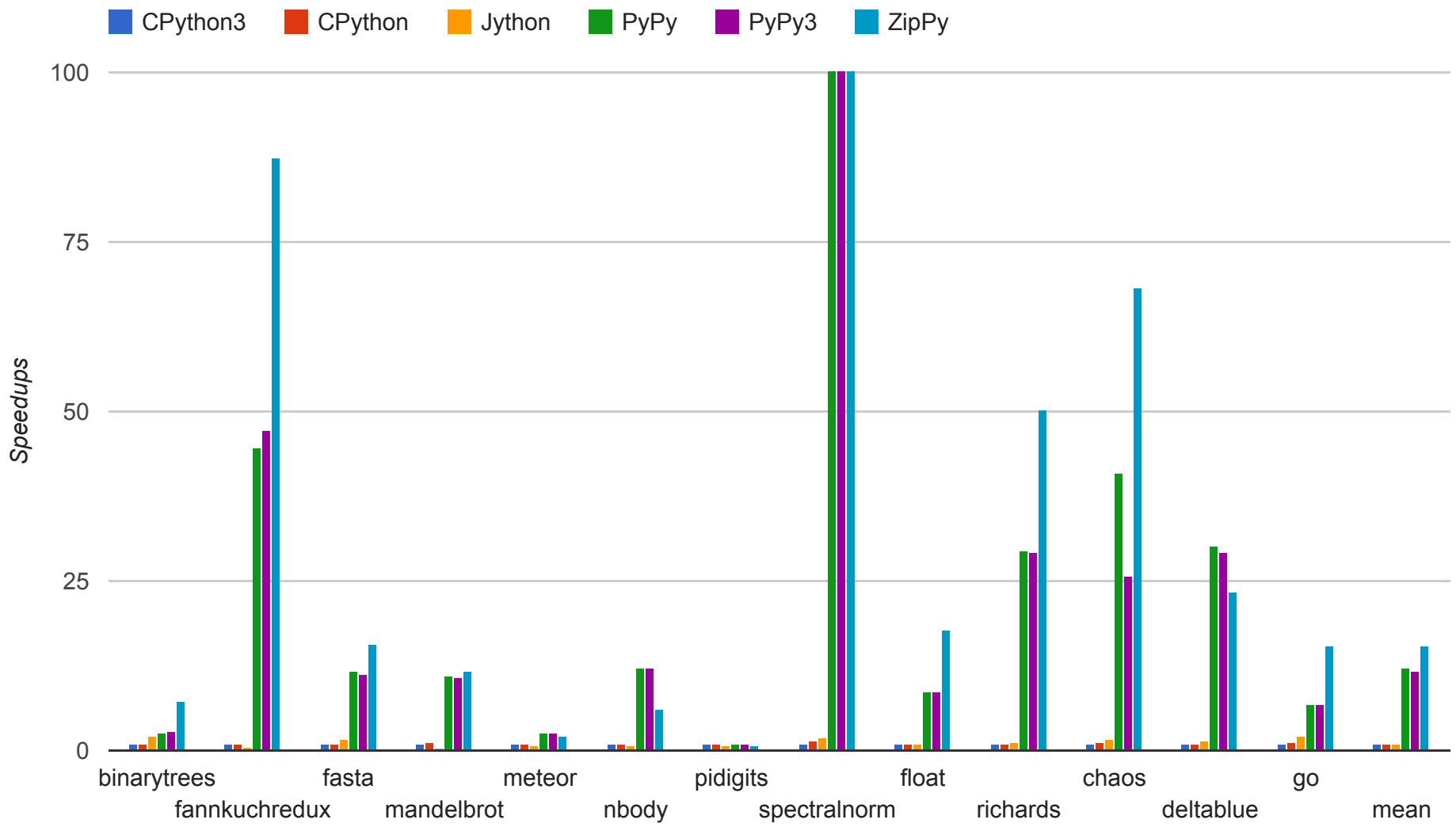
Benchmarking

- against CPython, Jython, PyPy
- measure peak performance
- generator-bound and traditional benchmarks

Performance of Generator Benchmarks



Performance of Traditional Benchmarks



Lesson Learned

- Trufflised AST is a good start
- but there's a lot more...

Thanks!

多谢!