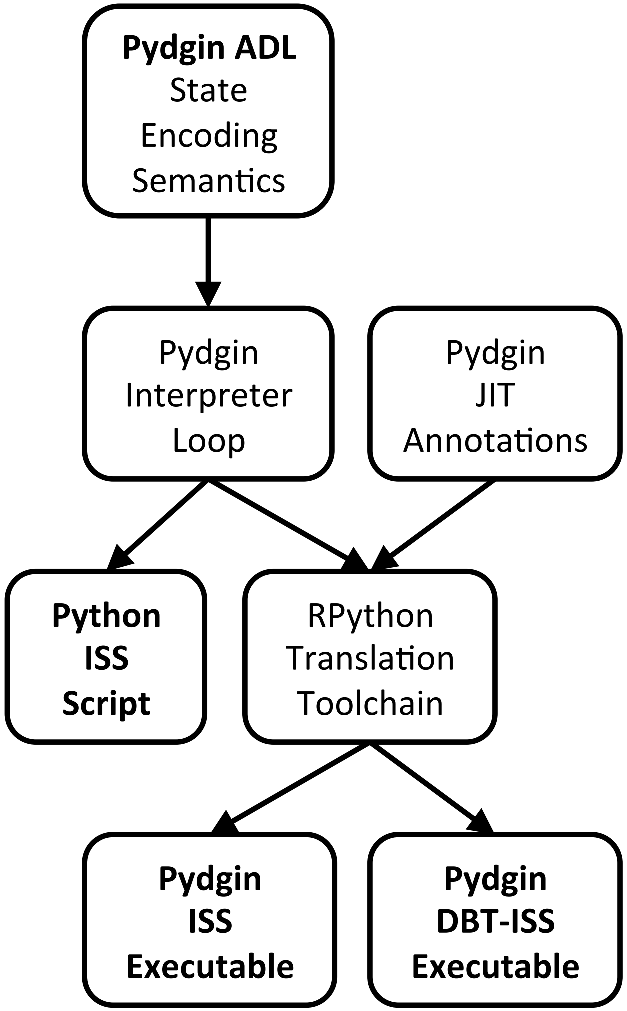
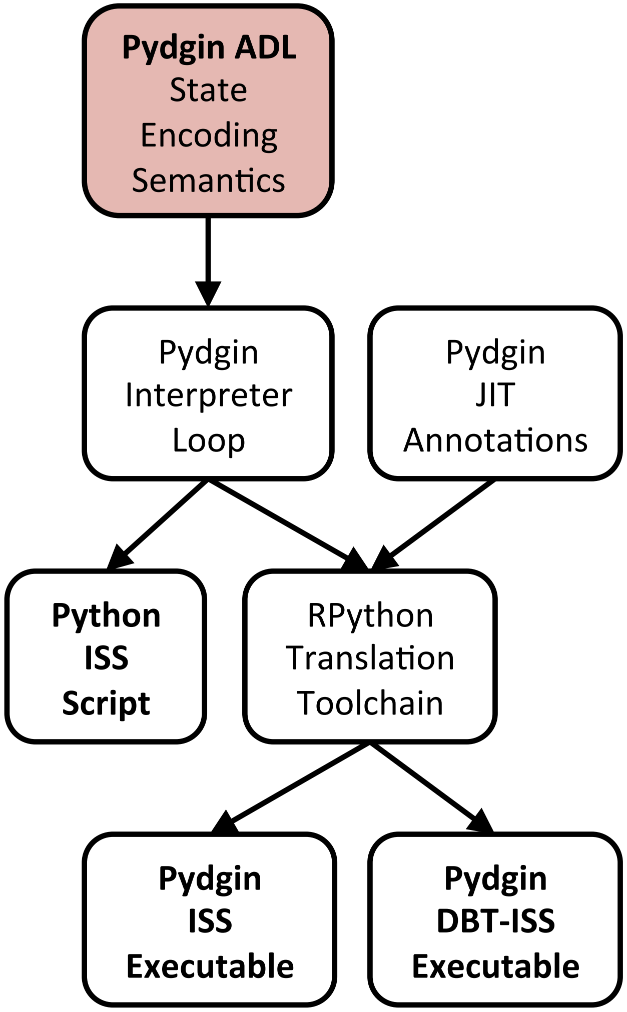
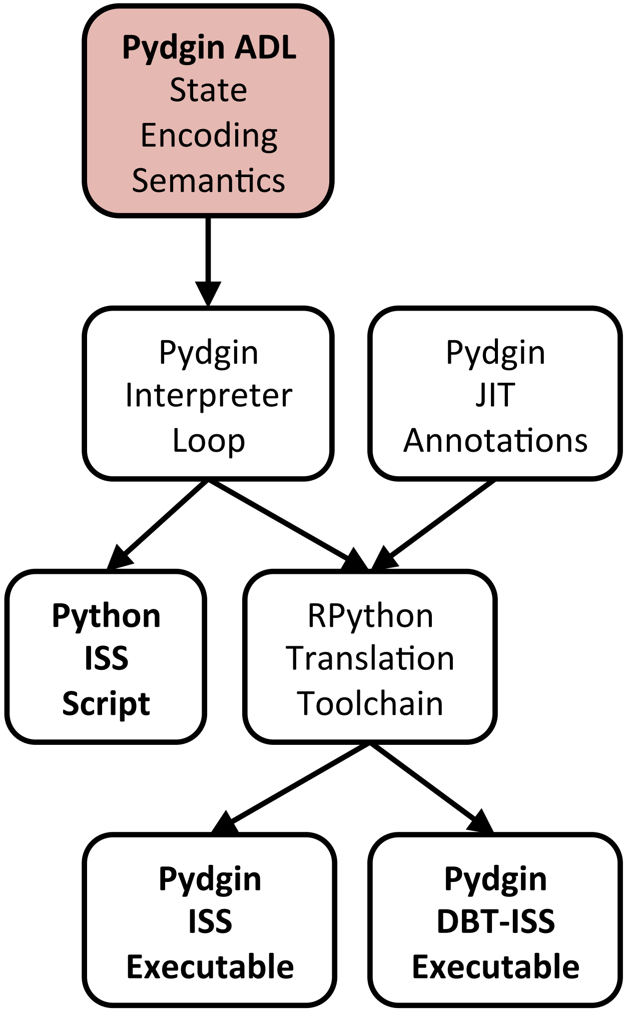
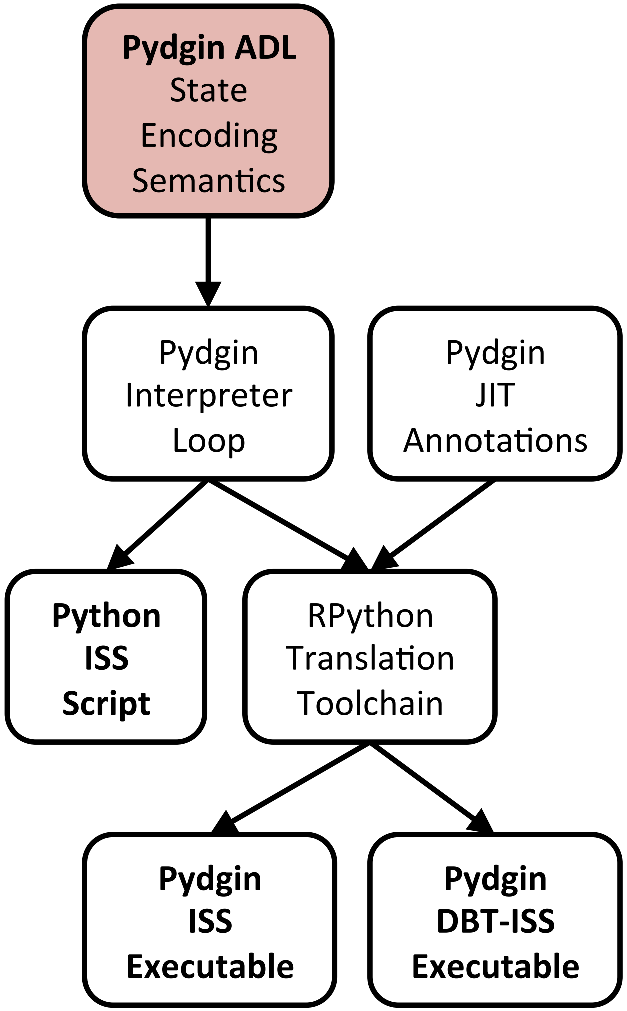

Pydgin ADL
State
Encoding
Semantics
Pydgin Interpreter Loop
Pydgin JIT Annotations
Python ISS Script
RPython Translation Toolchain
Pydgin ISS Executable
Pydgin DBT-ISS Executable

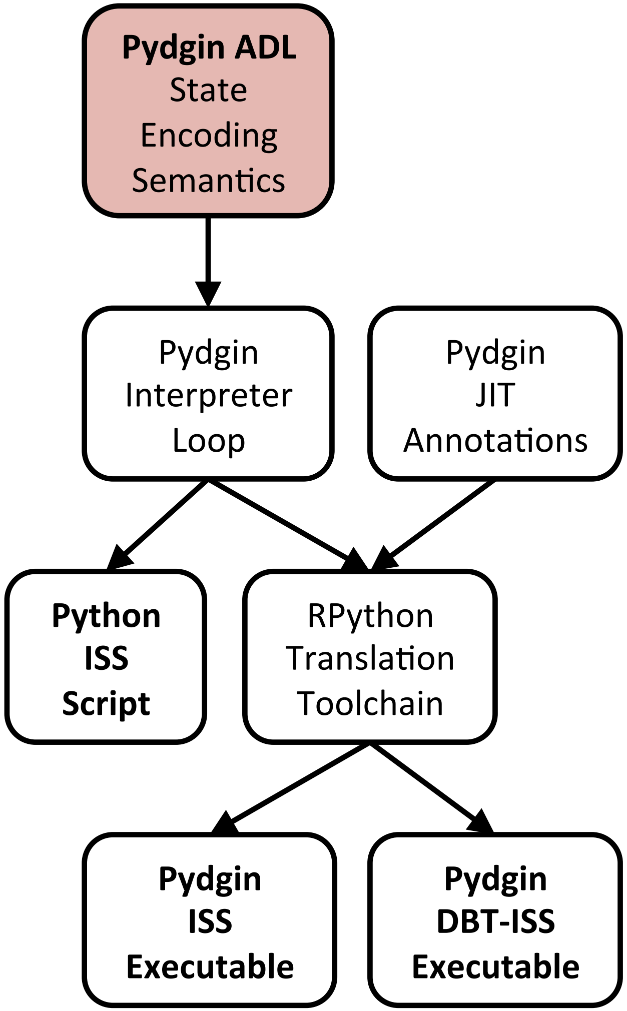

Pydgin ADL
State
Encoding
Semantics
Pydgin
Interpreter
Loop
Pydgin
JIT
Annotations
Python
ISS
Script
RPython
Translation
Toolchain
Pydgin
ISS
Executable
Pydgin
DBT-ISS
Executable



```
class State( object ):

  def __init__( self, memory, reset_addr=0x400 ):
    self.pc  = reset_addr
    self.rf  = ArmRegisterFile( self, num_regs=16 )
    self.mem = memory

    self.rf[ 15 ] = reset_addr

    # current program status register (CPSR)
    self.N    = 0b0      # Negative condition
    self.Z    = 0b0      # Zero condition
    self.C    = 0b0      # Carry condition
    self.V    = 0b0      # Overflow condition

  def fetch_pc( self ):
    return self.pc
```



```
encodings = [

  ['nop',   '00000000000000000000000000000000'],
  ['mul',   'xxxx0000000xxxxxxxxxxxxx1001xxxx'],
  ['umull', 'xxxx0000100xxxxxxxxxxxxx1001xxxx'],
  ['adc',   'xxxx00x0101xxxxxxxxxxxxxxxxxxxxx'],
  ['add',   'xxxx00x0100xxxxxxxxxxxxxxxxxxxxx'],
  ['and',   'xxxx00x0000xxxxxxxxxxxxxxxxxxxxx'],
  ['b',     'xxxx1010xxxxxxxxxxxxxxxxxxxxxxxx'],
  ['bl',    'xxxx1011xxxxxxxxxxxxxxxxxxxxxxxx'],
  ['bic',   'xxxx00x1110xxxxxxxxxxxxxxxxxxxxx'],
  ['bkpt',  '111000010010xxxxxxxxxxxx0111xxxx'],

  # ...

  ['teq',   'xxxx00x10011xxxxxxxxxxxxxxxxxxxx'],
  ['tst',   'xxxx00x10001xxxxxxxxxxxxxxxxxxxx'],

]
```



```
def execute_add( s, inst ):

  if condition_passed( s, inst.cond ):
    a,    = s.rf[ inst.rn ]
    b, _ = shifter_operand( s, inst )
    result = a + b
    s.rf[ inst.rd ] = trim_32(result)

    if inst.S:
      # ...
      s.N = (result >> 31)&1
      s.Z = trim_32(result) == 0
      s.C = carry_from(result)
      s.V = overflow_from(a, b, result)

    if inst.rd == 15:
      return
  s.rf[PC] = s.fetch_pc() + 4
```

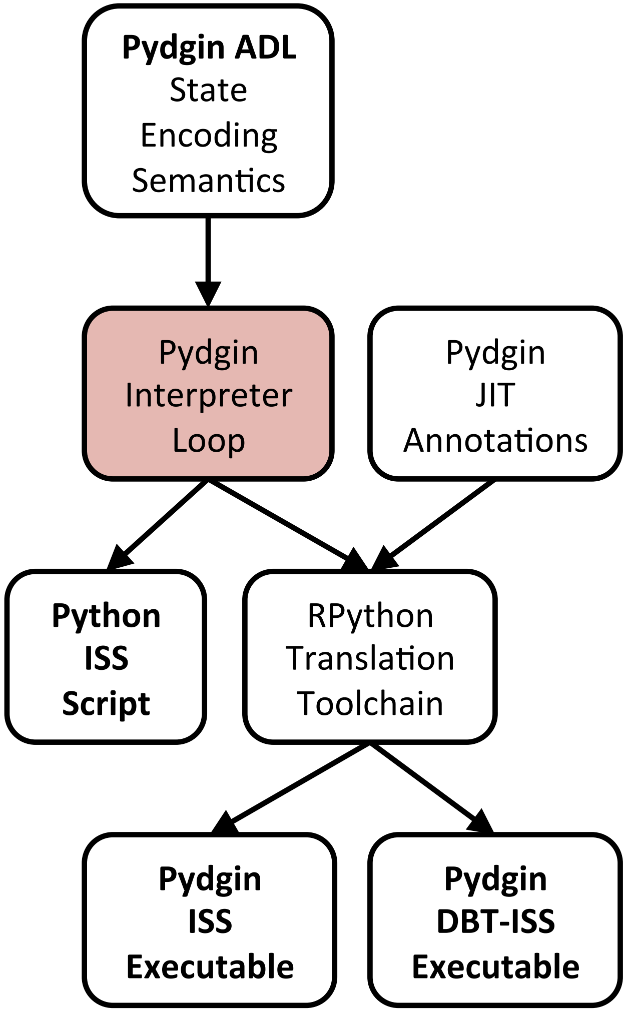
Pydgin ADL
State

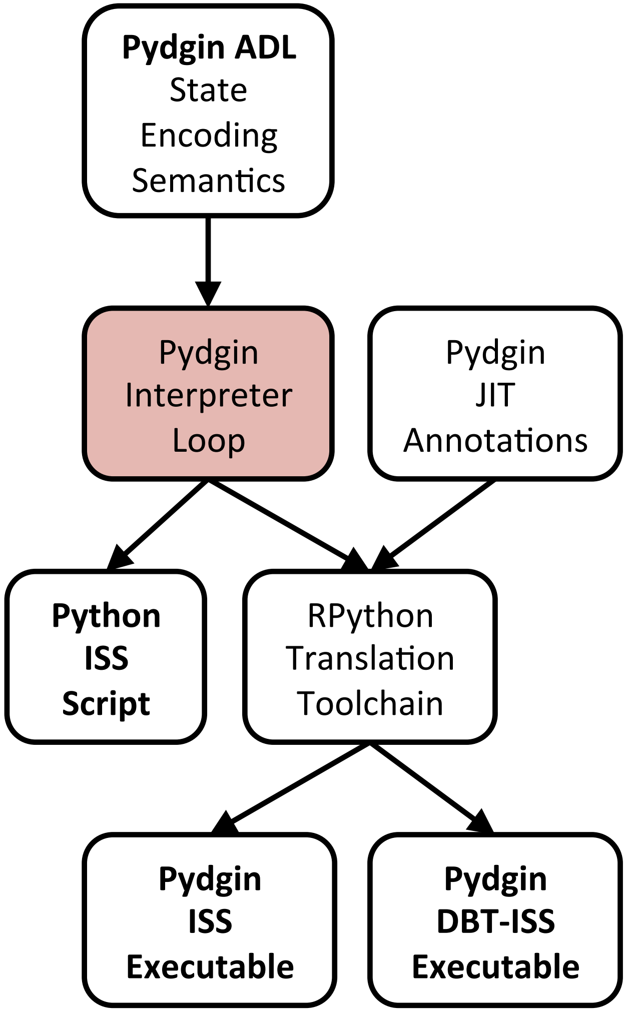
ARM ISA MANUAL SPEC

```
if ConditionPassed(cond) then

  Rd = Rn + shifter_operand

  if S == 1 and Rd == R15 then
    if CurrentModeHasSPSR() then
      CPSR = SPSR
    else UNPREDICTABLE

  else if S == 1 then
    N Flag = Rd[31]
    Z Flag = if Rd == 0 then 1 else 0
    C Flag = CarryFrom(Rn + shifter_operand)
    V Flag = OverflowFrom(Rn + shifter_operand)
```

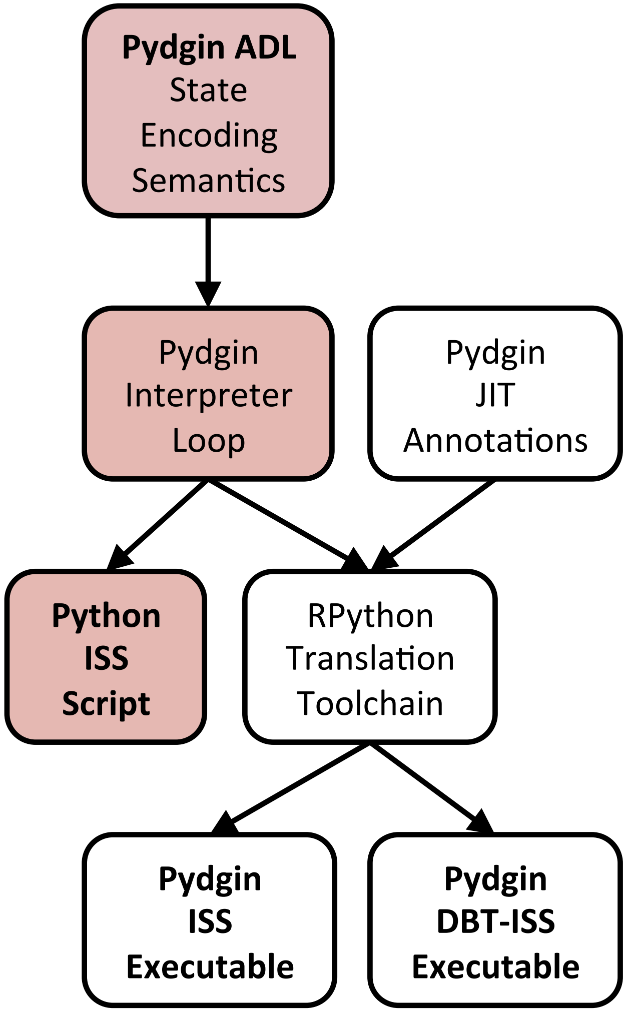
Executable   Executable

```
def execute_add( s, inst ):

  if condition_passed( s, inst.cond ):
    a,   = s.rf[ inst.rn ]
    b, _ = shifter_operand( s, inst )
    result = a + b
    s.rf[ inst.rd ] = trim_32(result)

    if inst.S:
      # ...
      s.N = (result >> 31)&1
      s.Z = trim_32(result) == 0
      s.C = carry_from(result)
      s.V = overflow_from(a, b, result)

    if inst.rd == 15:
      return
  s.rf[PC] = s.fetch_pc() + 4
```


Pydgin ADL
State
Encoding
Semantics
Pydgin Interpreter Loop
Pydgin JIT Annotations
Python ISS Script
RPython Translation Toolchain
Pydgin ISS Executable
Pydgin DBT-ISS Executable

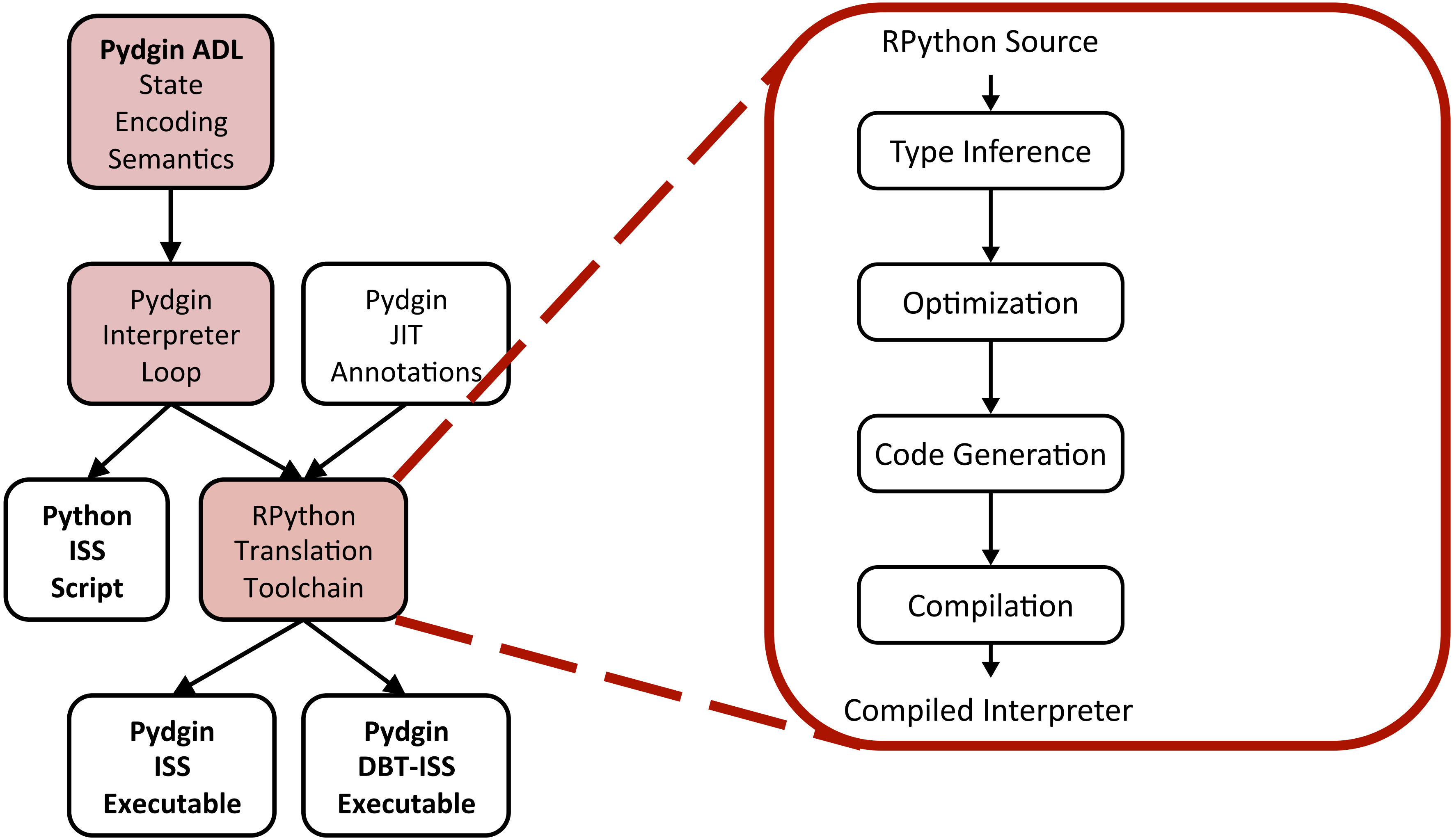


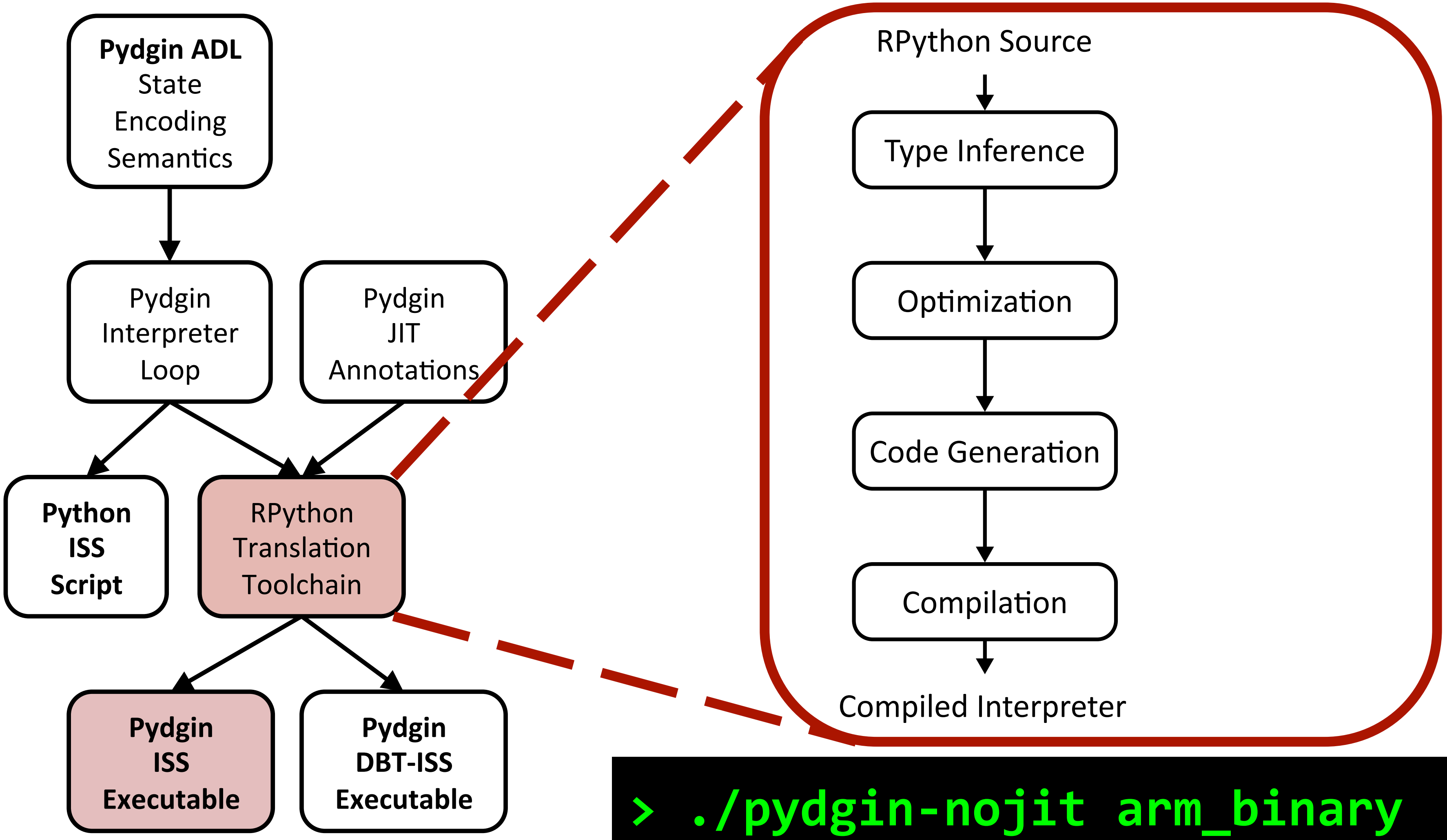
```
def instruction_set_interpreter( memory ):
  state = State( memory )

  while True:


    pc      = state.fetch_pc()

    inst    = memory[ pc ]      # fetch
    execute = decode( inst )    # decode
    execute( state, inst )      # execute
```



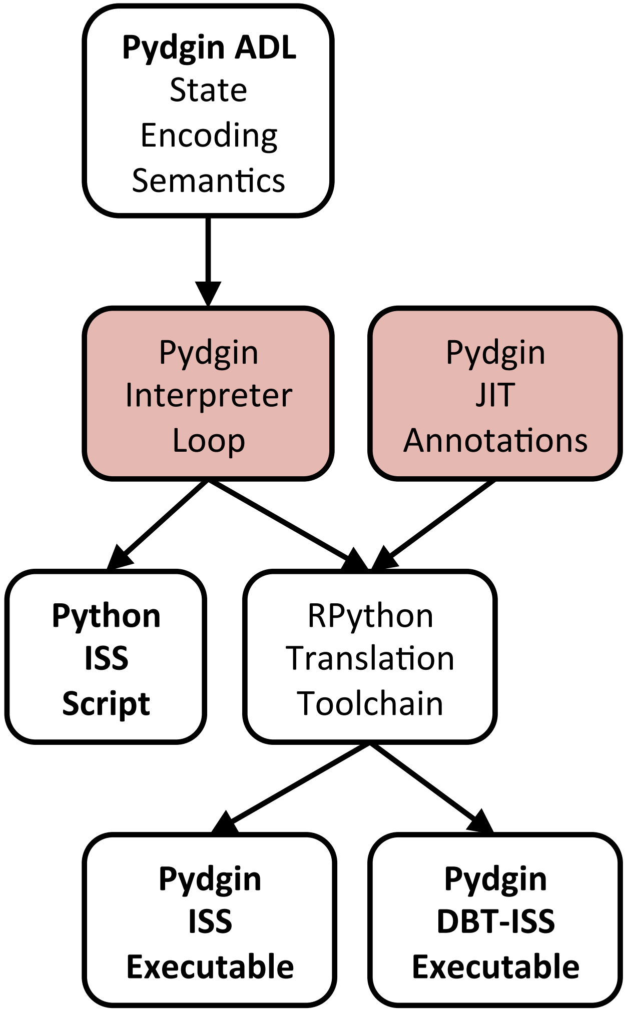
```
def instruction_set_interpreter( memory ):
  state = State( memory )

  while True:


    pc      = state.fetch_pc()

    inst    = memory[ pc ]        # fetch
    execute = decode( inst )      # decode
    execute( state, inst )        # execute
```

```
> python iss.py arm_binary
```

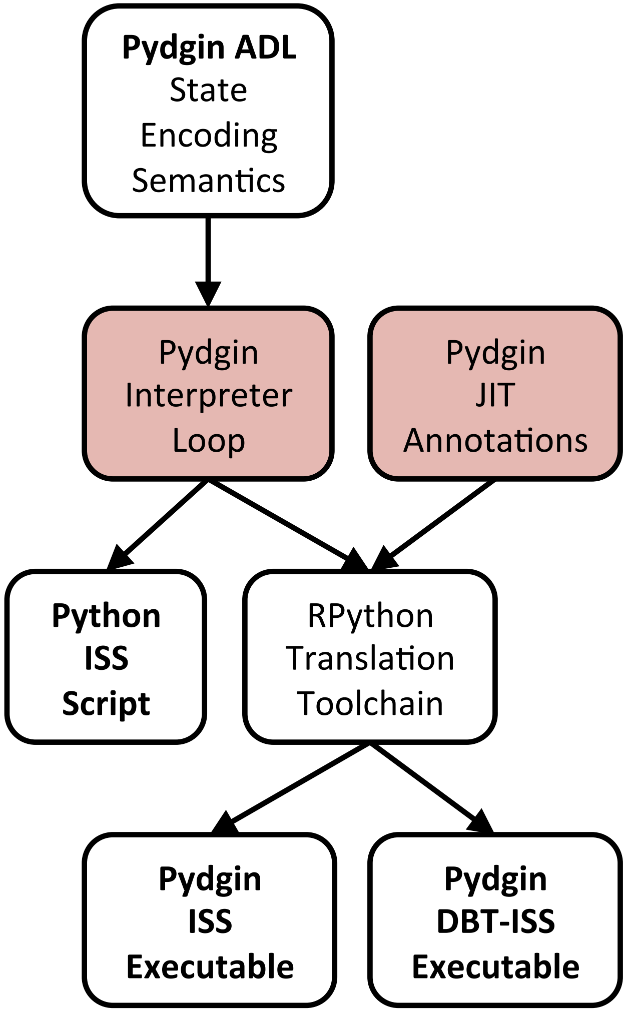

Pydgin ADL
State
Encoding
Semantics
Pydgin Interpreter Loop
Pydgin JIT Annotations
Python ISS Script
RPython Translation Toolchain
Pydgin ISS Executable
Pydgin DBT-ISS Executable
RPython Source
Type Inference
Optimization
Code Generation
Compilation
Compiled Interpreter



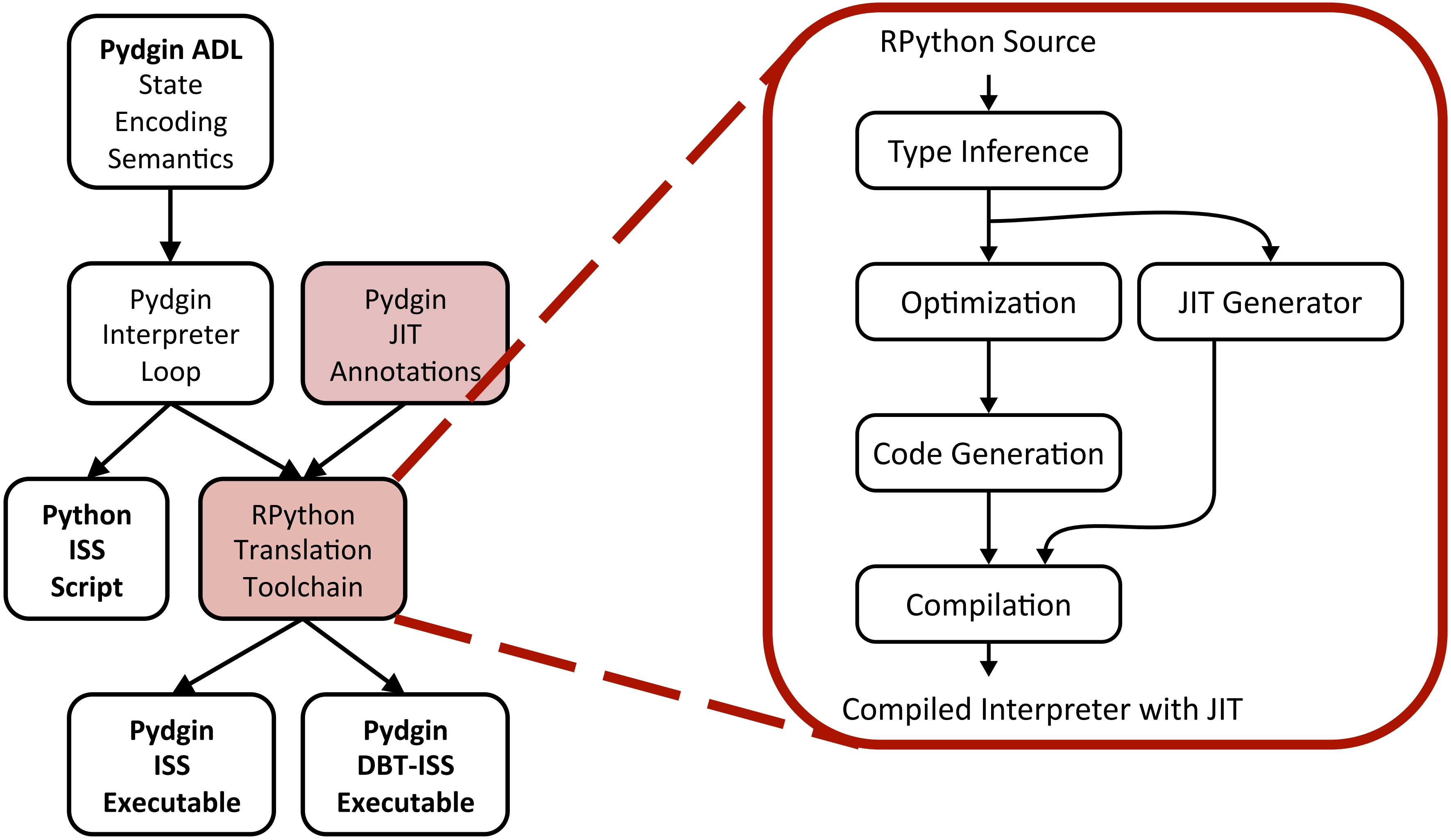
```
> ./pydgin-nojit arm_binary
```

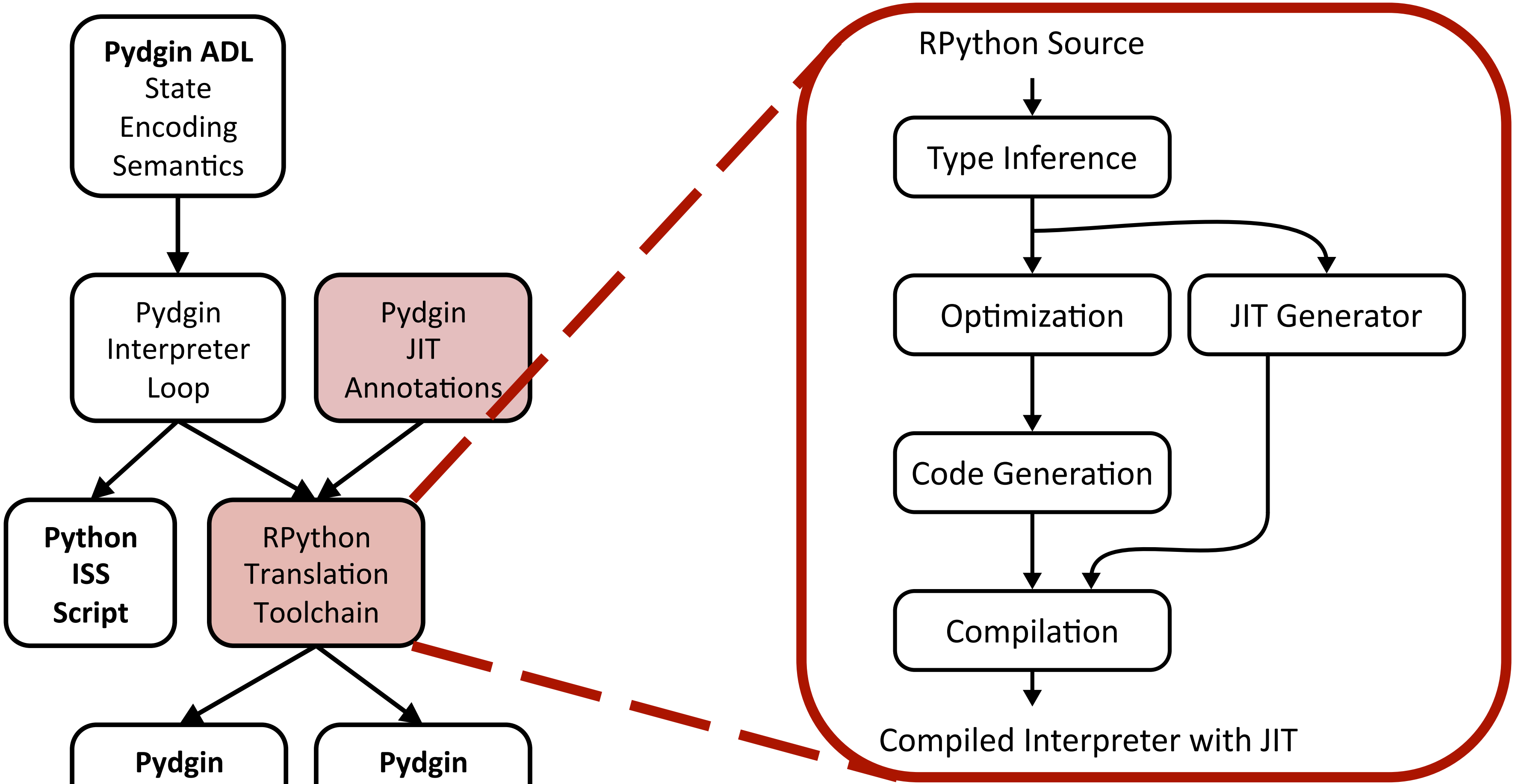


```
def instruction_set_interpreter( memory ):
  state = State( memory )

  while True:


    pc      = state.fetch_pc()

    inst    = memory[ pc ]       # fetch
    execute = decode( inst )     # decode
    execute( state, inst )       # execute
```



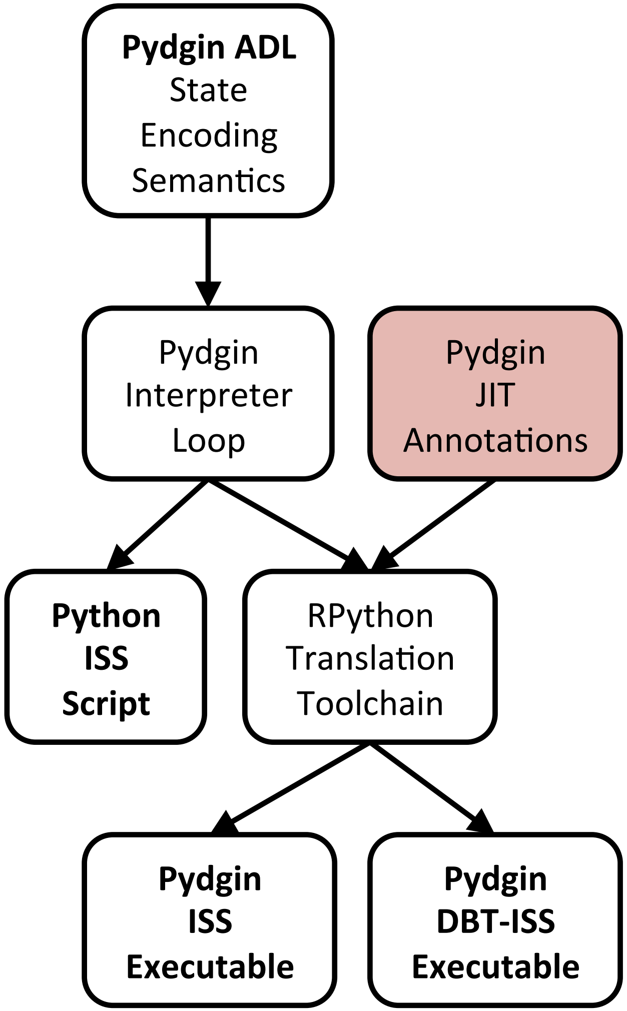
```
jd = JitDriver( greens = ['pc'],
                reds   = ['state'] )

def instruction_set_interpreter( memory ):
  state = State( memory )

  while True:
    jd.jit_merge_point( s.fetch_pc(), state )

    pc      = state.fetch_pc()

    inst    = memory[ pc ]      # fetch
    execute = decode( inst )    # decode
    execute( state, inst )      # execute

    if state.fetch_pc() < pc:
      jd.can_enter_jit( s.fetch_pc(), state )
```

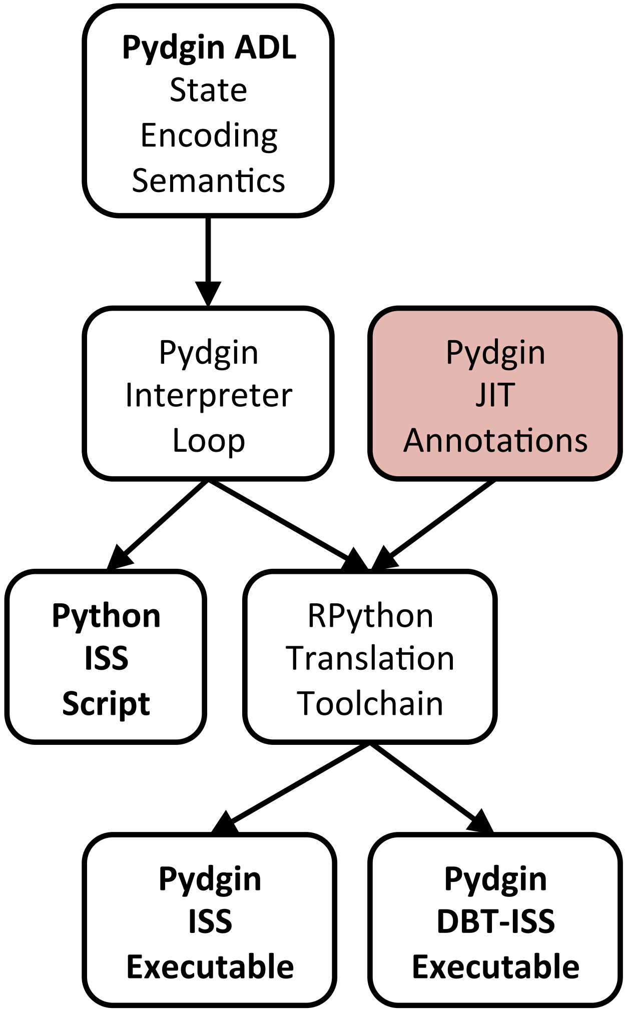

Pydgin ADL
State
Encoding
Semantics
Pydgin Interpreter Loop
Pydgin JIT Annotations
Python ISS Script
RPython Translation Toolchain
Pydgin ISS Executable
Pydgin DBT-ISS Executable
RPython Source
Type Inference
Optimization
JIT Generator
Code Generation
Compilation
Compiled Interpreter with JIT



```
> ./pydgin-jit arm_binary
```

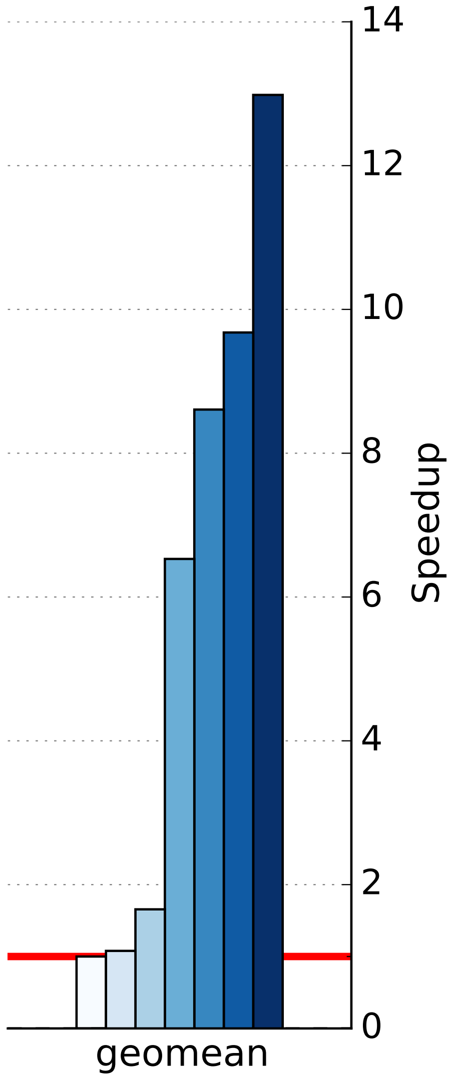
Creating a competitive JIT requires additional RPython JIT hints:





Creating a competitive JIT requires additional RPython JIT hints:

+ Minimal JIT Annotations
+ Elidable Instruction Fetch
+ Elidable Decode
+ Constant Promotion of PC and Memory
+ Word-Based Target Memory
+ Loop Unrolling in Instruction Semantics
+ Virtualizable PC and Statistics



Creating a competitive JIT requires additional RPython JIT hints:

+ Minimal JIT Annotations
+ Elidable Instruction Fetch
+ Elidable Decode
+ Constant Promotion of PC and Memory
+ Word-Based Target Memory
+ Loop Unrolling in Instruction Semantics
+ Virtualizable PC and Statistics

See our paper in ISPASS2015 for performance results!