Pydgin for RISC-V:
A Fast and Productive Instruction-Set Simulator

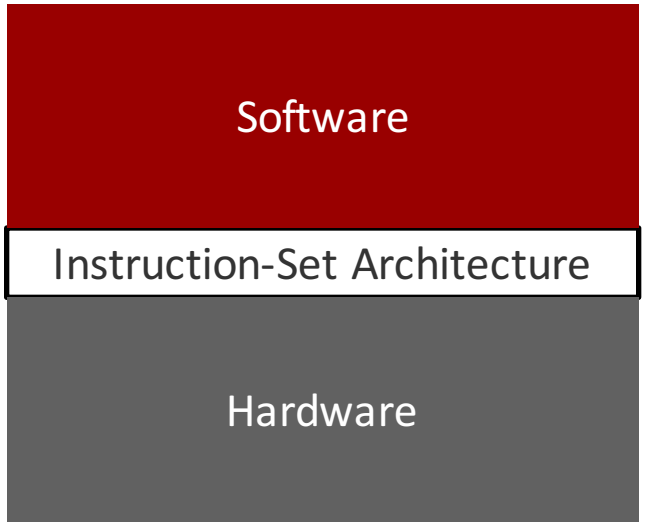
Berkin Ilbeyi

In collaboration with Derek Lockhart (Google), and Christopher Batten

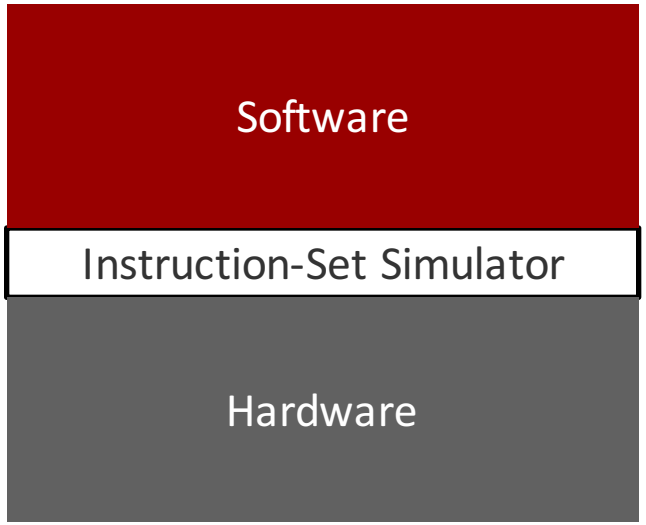
3rd RISC-V Workshop, Jan 2016

Cornell University
Computer Systems Laboratory

Motivation



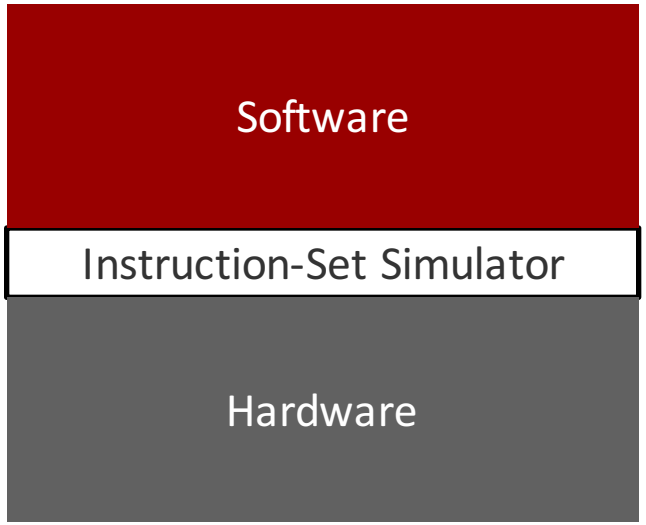
Motivation



Motivation

Performance

Interpretive: 1-10 MIPS (1-10 days)

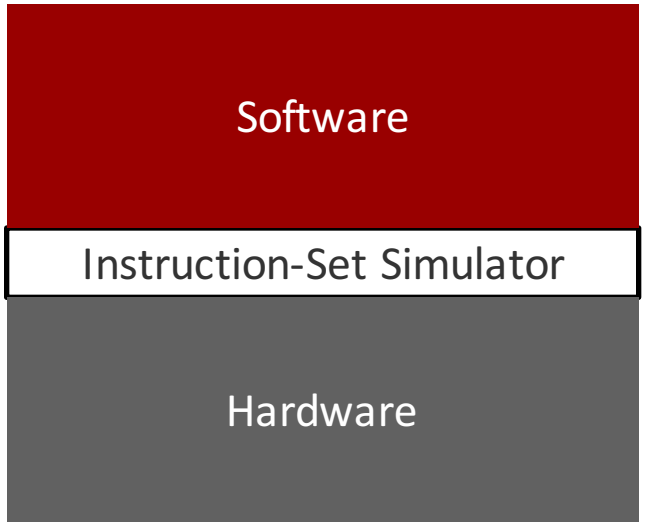


Motivation

Performance

Interpretive: 1-10 MIPS (1-10 days)

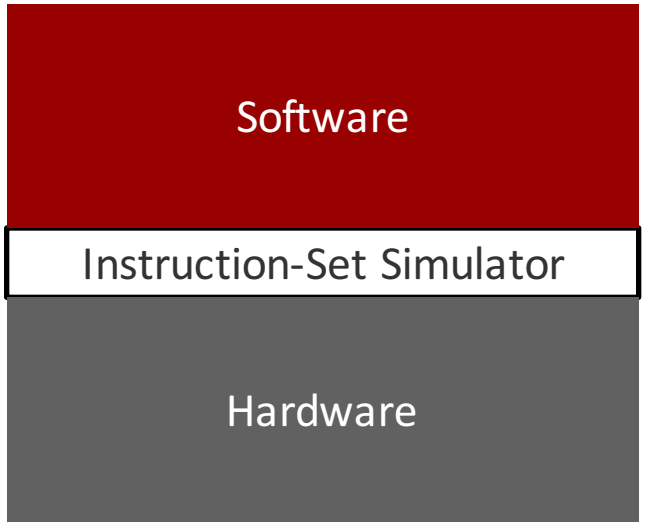
Typical DBT: 100s MIPS (1-3 hours)



Motivation

Performance

Interpretive:	1-10 MIPS (1-10 days)
Typical DBT:	100s MIPS (1-3 hours)
QEMU DBT:	1000 MIPS (0.5 hours)



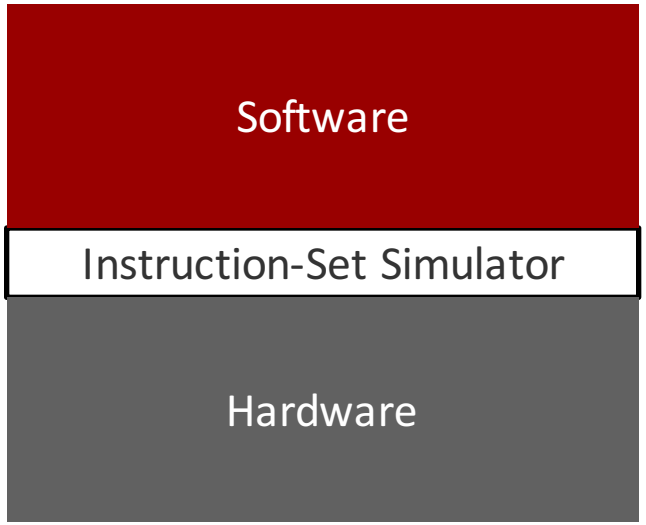
Motivation

Productivity

- Develop
- Extend
- Instrument

Performance

Interpretive:	1-10 MIPS (1-10 days)
Typical DBT:	100s MIPS (1-3 hours)
QEMU DBT:	1000 MIPS (0.5 hours)



Motivation

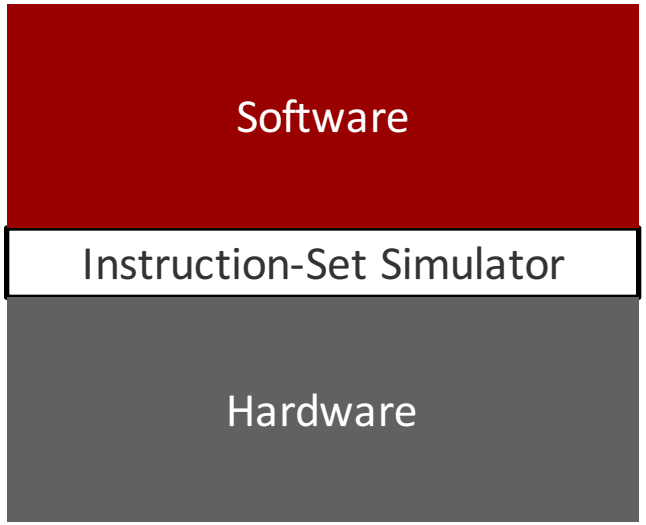
Productivity  Performance

- Develop
- Extend
- Instrument

Interpretive: 1-10 MIPS (1-10 days)

Typical DBT: 100s MIPS (1-3 hours)

QEMU DBT: 1000 MIPS (0.5 hours)



Motivation

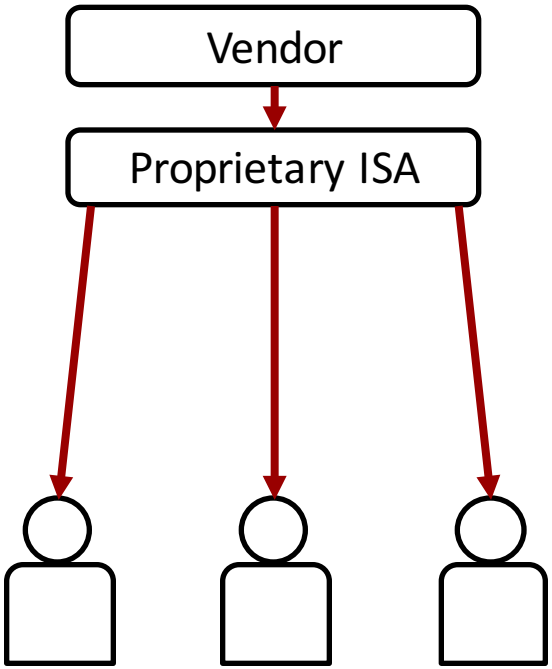
Productivity  Performance

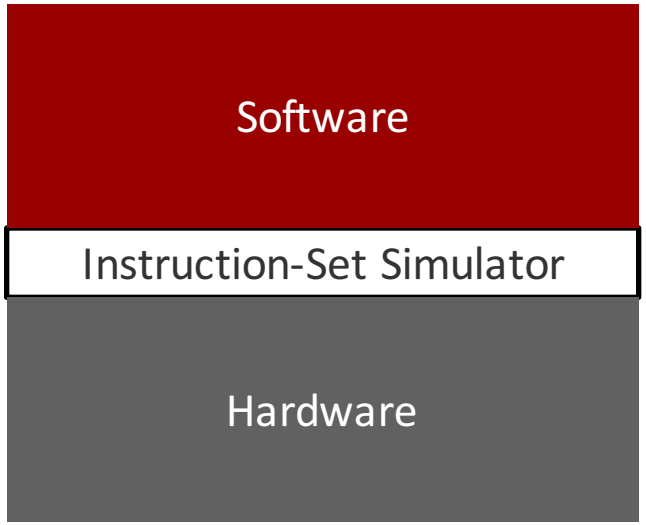
- Develop
- Extend
- Instrument

Interpretive: 1-10 MIPS (1-10 days)

Typical DBT: 100s MIPS (1-3 hours)

QEMU DBT: 1000 MIPS (0.5 hours)





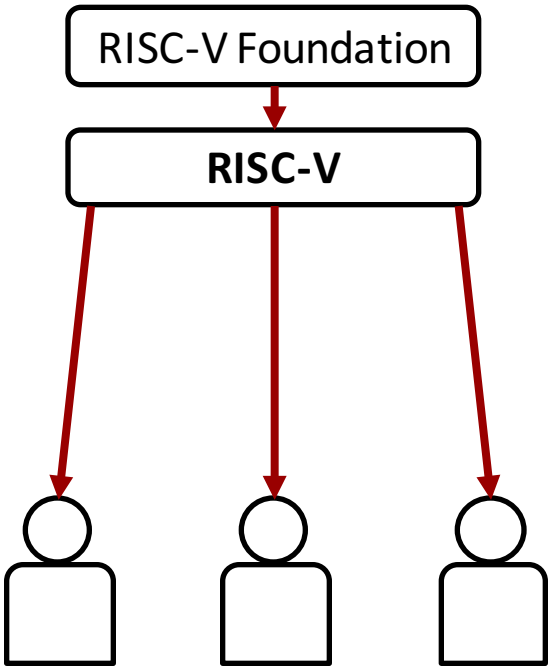
Motivation

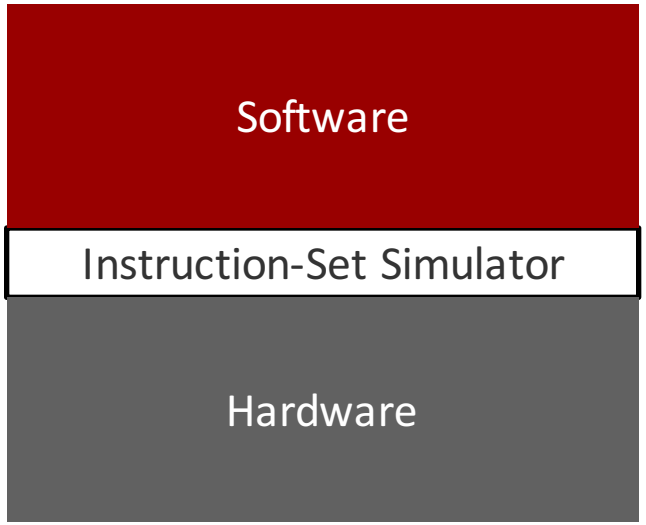
- Develop
- Extend
- Instrument

Interpretive: 1-10 MIPS (1-10 days)

Typical DBT: 100s MIPS (1-3 hours)

QEMU DBT: 1000 MIPS (0.5 hours)





Motivation

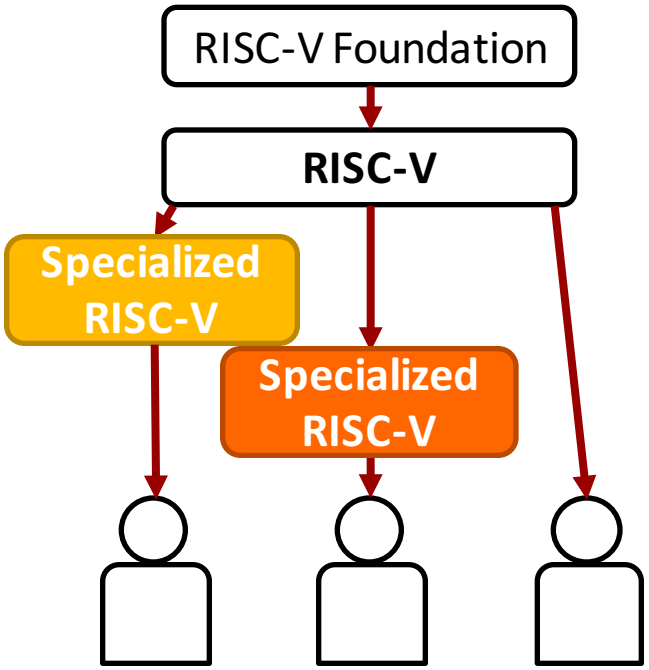
Productivity ⟷ **Performance**

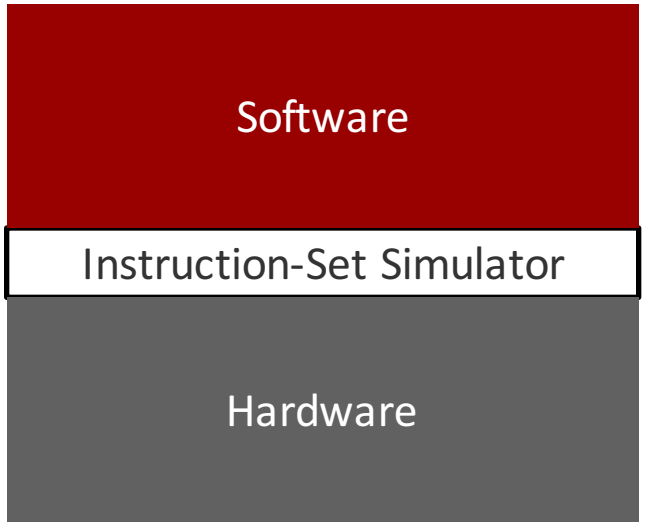
- Develop
- Extend
- Instrument

Interpretive: 1-10 MIPS (1-10 days)

Typical DBT: 100s MIPS (1-3 hours)

QEMU DBT: 1000 MIPS (0.5 hours)





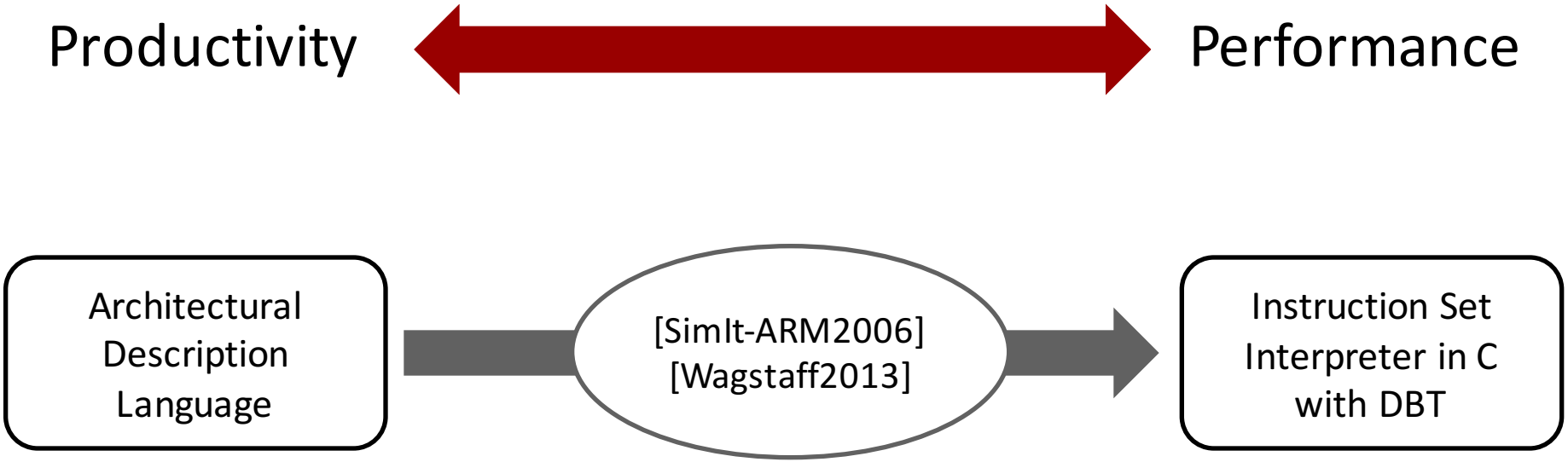
Productivity  Performance

Productivity  Performance

Architectural Description Language

Instruction Set Interpreter in C with DBT

Productivity  Performance



[Simit-ARM2006] J.D'Errico and W.Qin. Constructing Portable Compiled Instruction-Set Simulators — An ADL-Driven Approach. DATE'06.
[Wagstaff2013] H. Wagstaff, M. Gould, B. Franke, and N.Topham. Early Partial Evaluation in a JIT-Compiled, Retargetable Instruction Set Simulator Generated from a High-Level Architecture Description. DAC'13.

Productivity

Performance

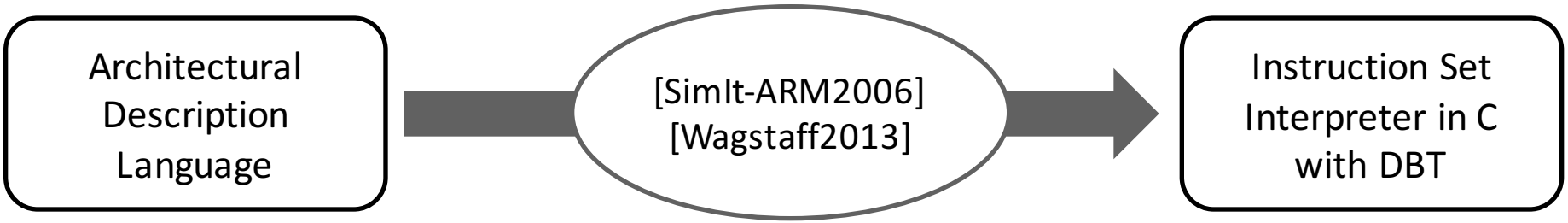
Architectural Description Language

[SimIt-ARM2006]
[Wagstaff2013]

Instruction Set Interpreter in C with DBT

Dynamic Language Interpreter in C with JIT Compiler

Productivity ⟷ Performance

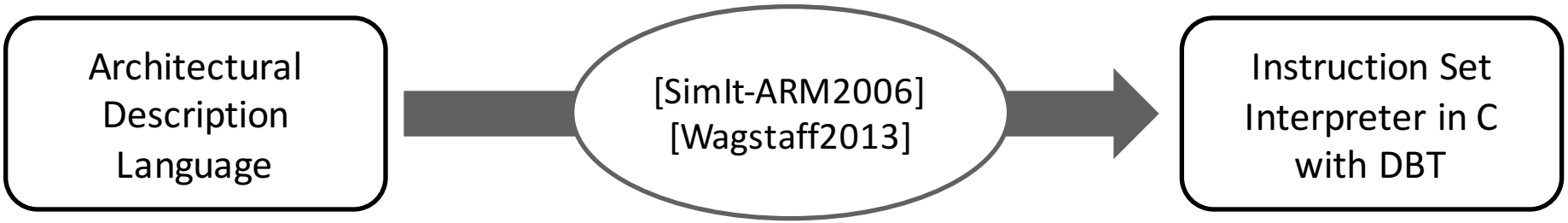


Key Insight:

Similar productivity-performance challenges for building high-performance interpreters of dynamic languages.
(e.g. JavaScript, Python)

Dynamic Language Interpreter in C with JIT Compiler

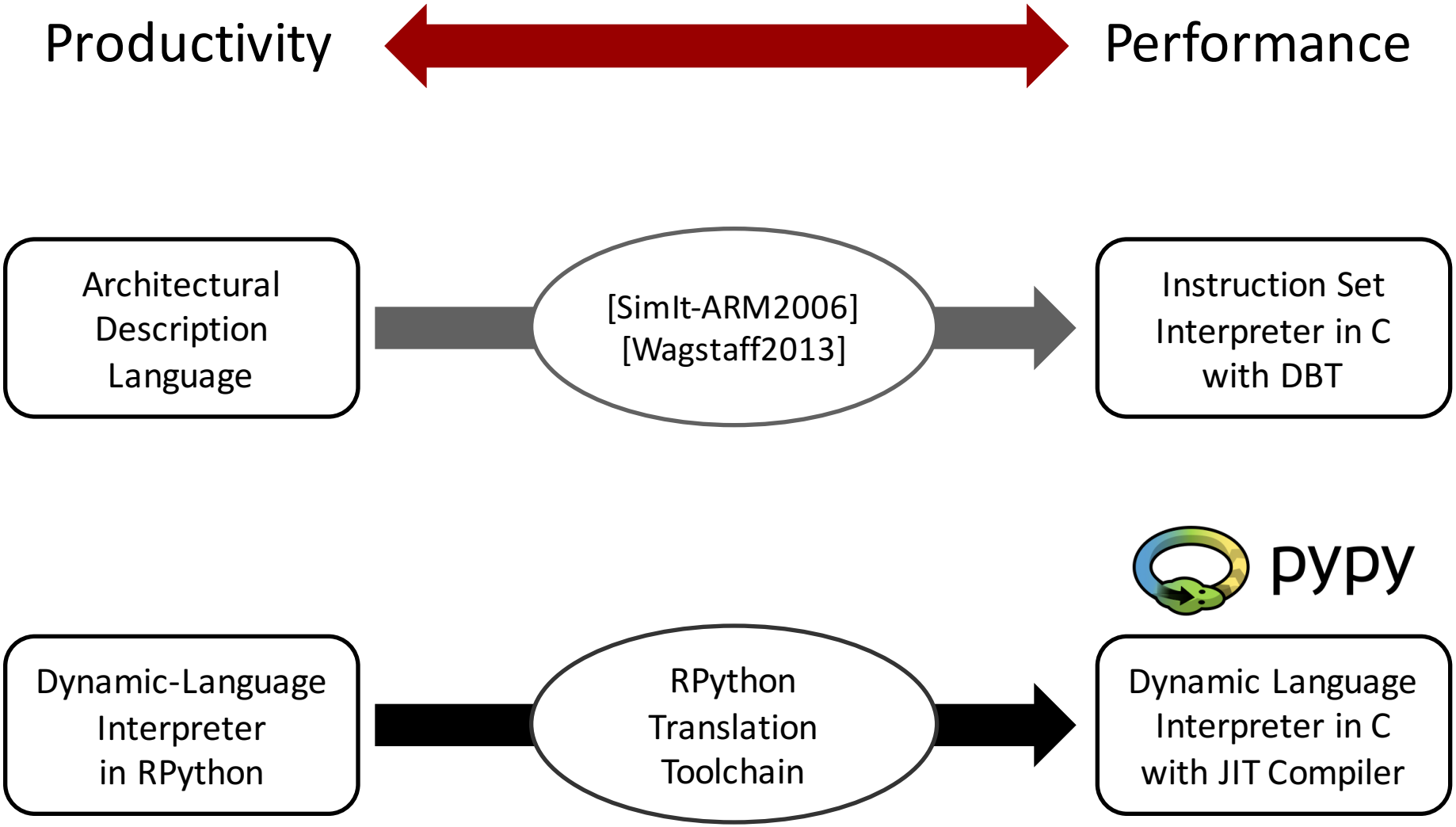
Productivity ⟷ Performance

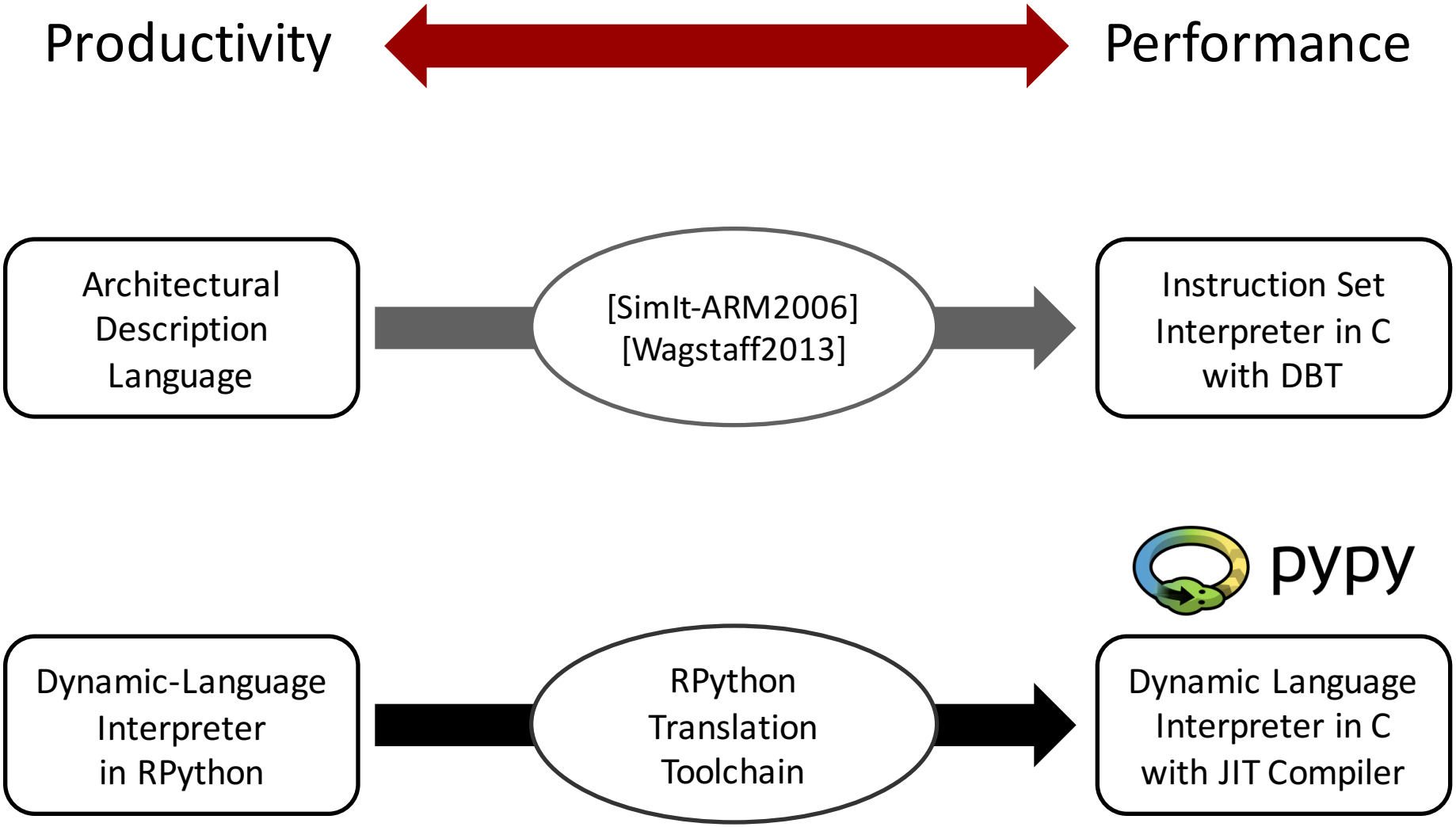


Key Insight:

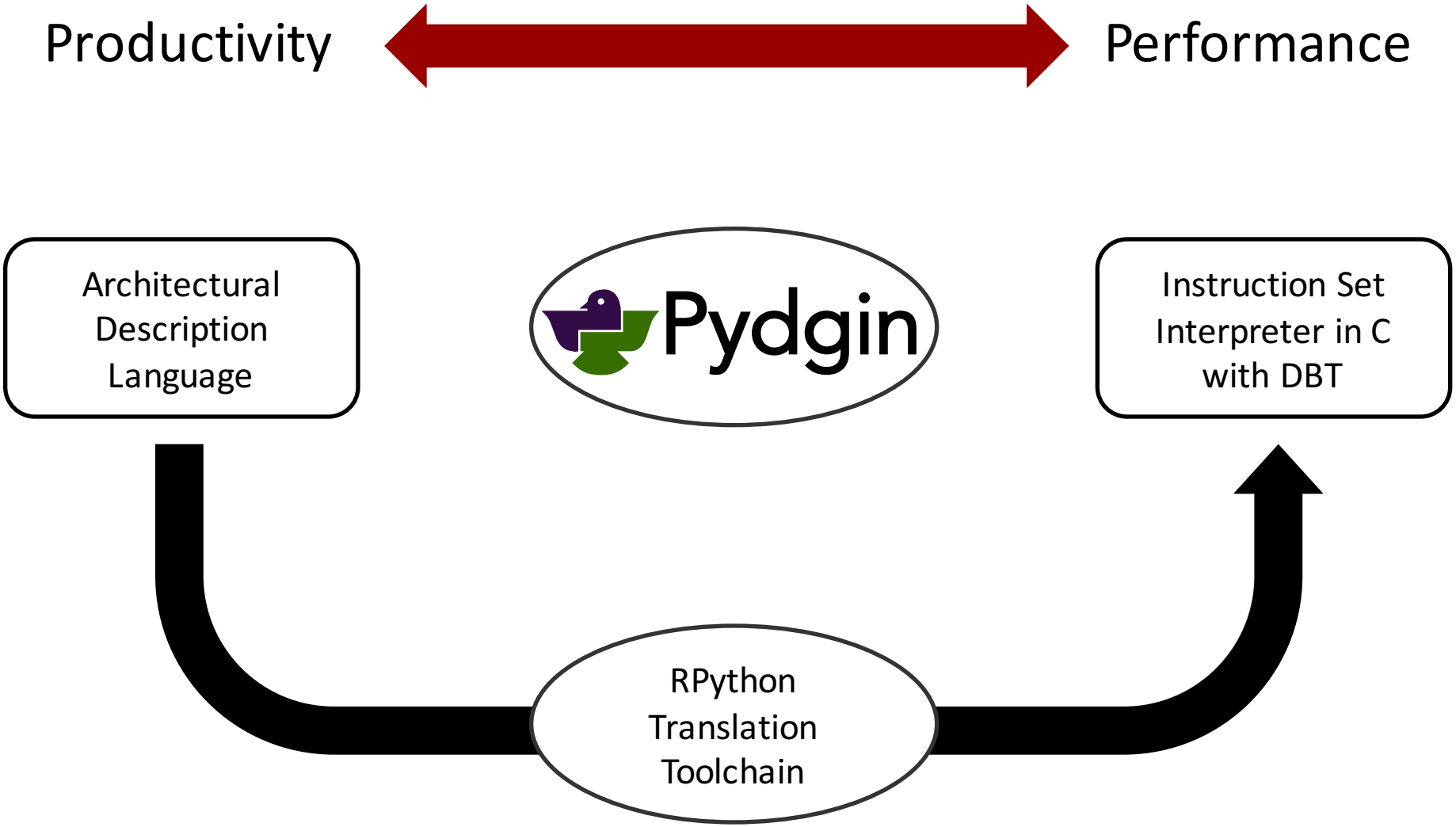
Similar productivity-performance challenges for building high-performance interpreters of dynamic languages.
(e.g. JavaScript, Python)

Dynamic Language Interpreter in C with JIT Compiler

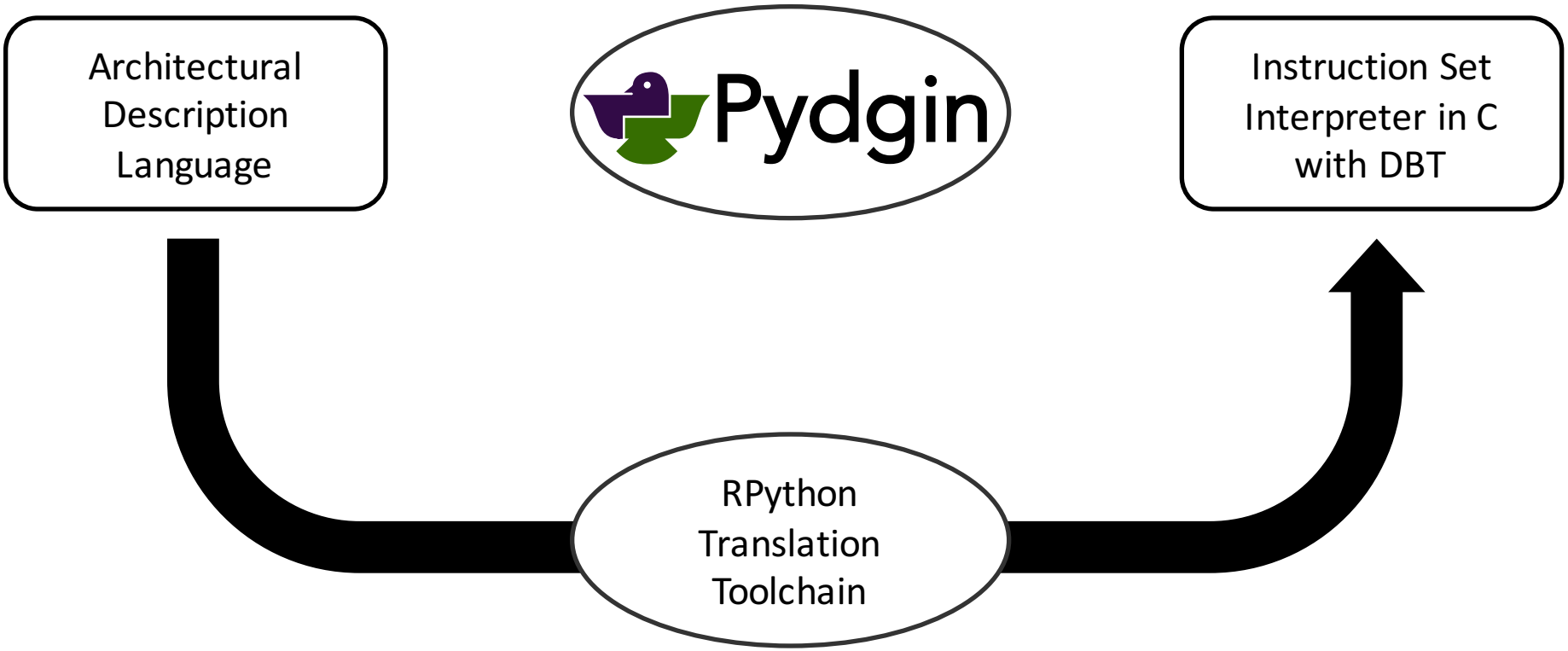

Productivity
Performance
Architectural Description Language
[SimIt-ARM2006] [Wagstaff2013]
Instruction Set Interpreter in C with DBT
pypy
Dynamic-Language Interpreter in RPython
RPython Translation Toolchain
Dynamic Language Interpreter in C with JIT Compiler



Meta-Tracing JIT:
makes JIT generation generic across languages

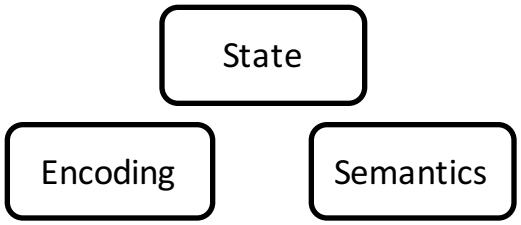

Productivity
Performance
Architectural Description Language
Pydgin
Instruction Set Interpreter in C with DBT
RPython Translation Toolchain

Productivity ⟷ Performance



JIT ≈ DBT

Pydgin Architecture Description Language

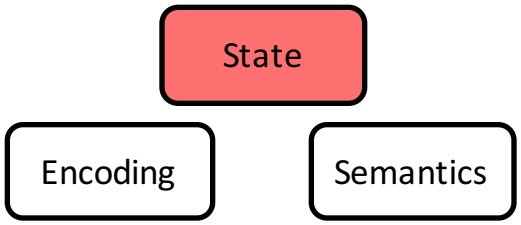


Architectural State

Instruction Encoding

Instruction Semantics

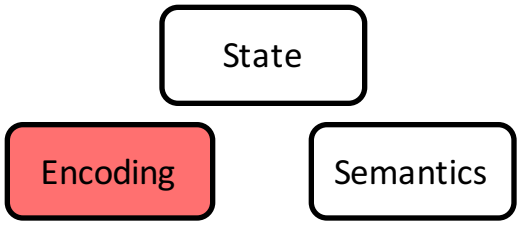
Pydgin Architecture Description Language



Architectural State

```
class State( object ):

  def __init__( self, memory, reset_addr=0x400 ):

    self.pc  = reset_addr
    self.rf  = RiscVRegisterFile()
    self.mem = memory

    # optional state if floating point is enabled
    if ENABLE_FP:
      self.fp   = RiscVFPRegisterFile()
      self.fcsr = 0
```

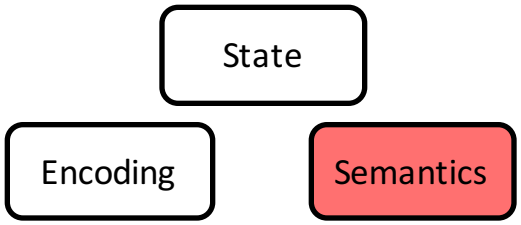
Pydgin Architecture Description Language



Instruction Encoding

```
encodings = [
  # ...

  ['xori', 'xxxxxxxxxxxxxxxxx100xxxxx0010011'],
  ['ori',  'xxxxxxxxxxxxxxxxx110xxxxx0010011'],
  ['andi', 'xxxxxxxxxxxxxxxxx111xxxxx0010011'],
  ['slli', '000000xxxxxxxxxxx001xxxxx0010011'],
  ['srli', '000000xxxxxxxxxxx101xxxxx0010011'],
  ['srai', '010000xxxxxxxxxxx101xxxxx0010011'],
  ['add',  '0000000xxxxxxxxxx000xxxxx0110011'],
  ['sub',  '0100000xxxxxxxxxx000xxxxx0110011'],
  ['sll',  '0000000xxxxxxxxxx001xxxxx0110011'],

  # ...
]
```

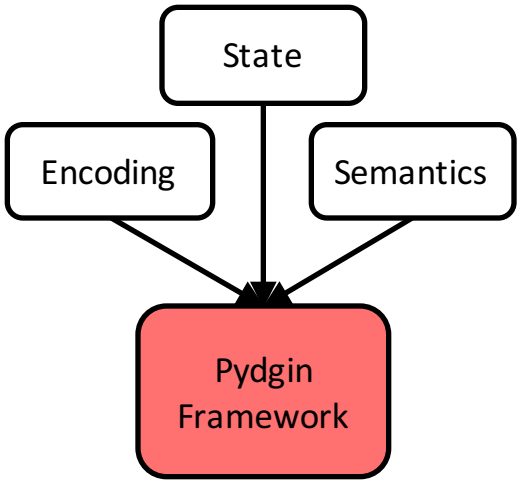
Pydgin Architecture Description Language



Instruction Semantics

```
def execute_addi( s, inst ):
  s.rf[inst.rd] = s.rf[inst.rs1] + inst.i_imm
  s.pc += 4

def execute_sw( s, inst ):
  addr = trim_xlen( s.rf[inst.rs1] + inst.s_imm )
  s.mem.write( addr, 4, trim_32( s.rf[inst.rs2] ) )
  s.pc += 4

def execute_beq( s, inst ):
  if s.rf[inst.rs1] == s.rf[inst.rs2]:
    s.pc = trim_xlen( s.pc + inst.sb_imm )
  else:
    s.pc += 4
```

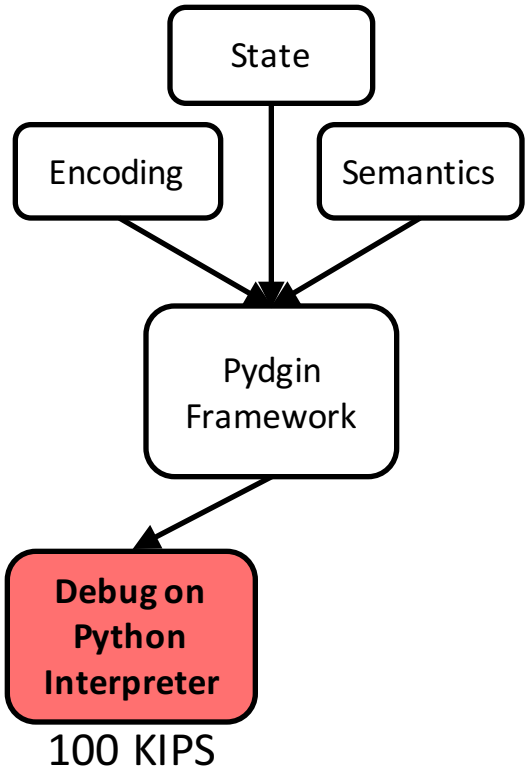
Pydgin Framework



Interpreter Loop

```
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
```

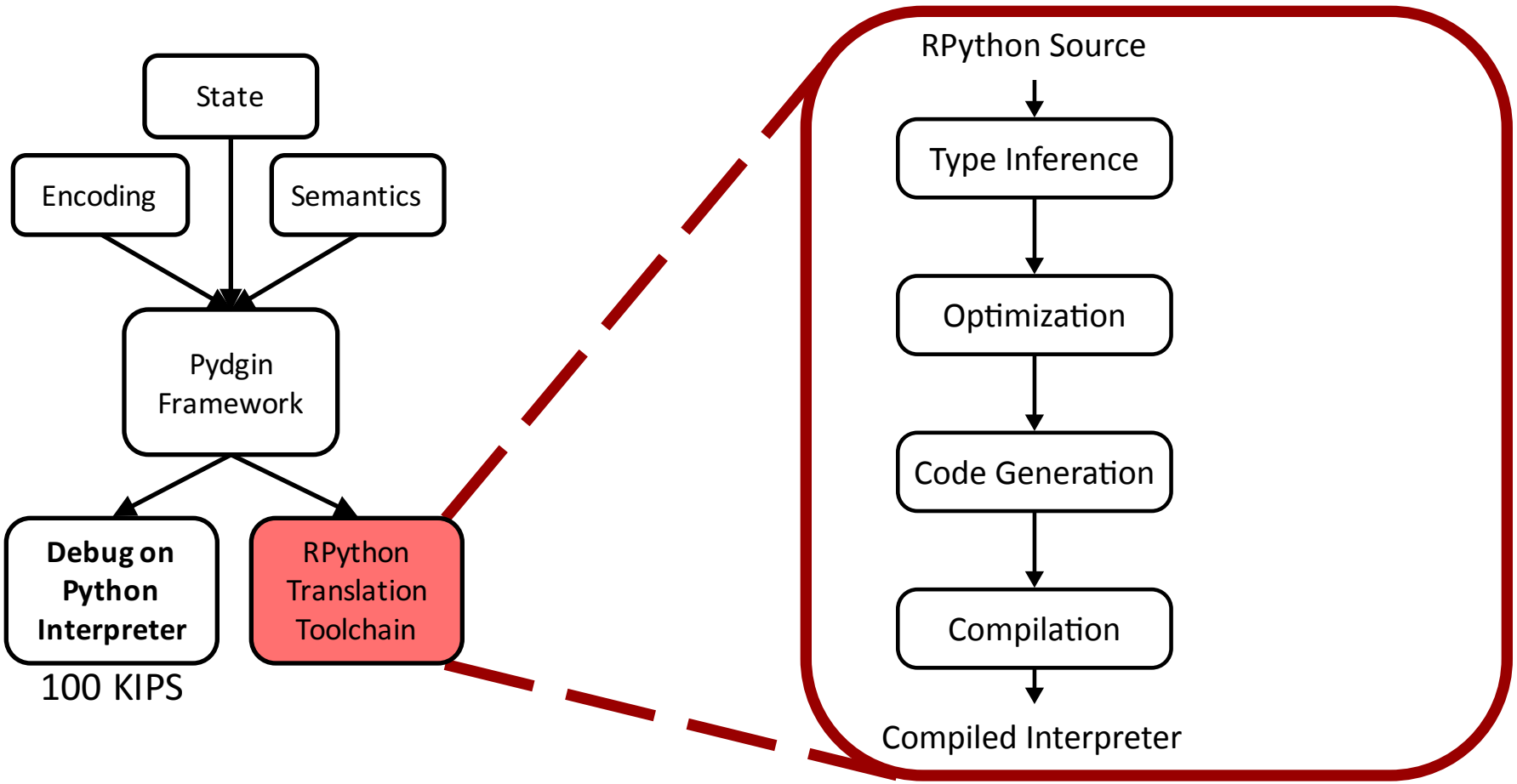
Pydgin Framework



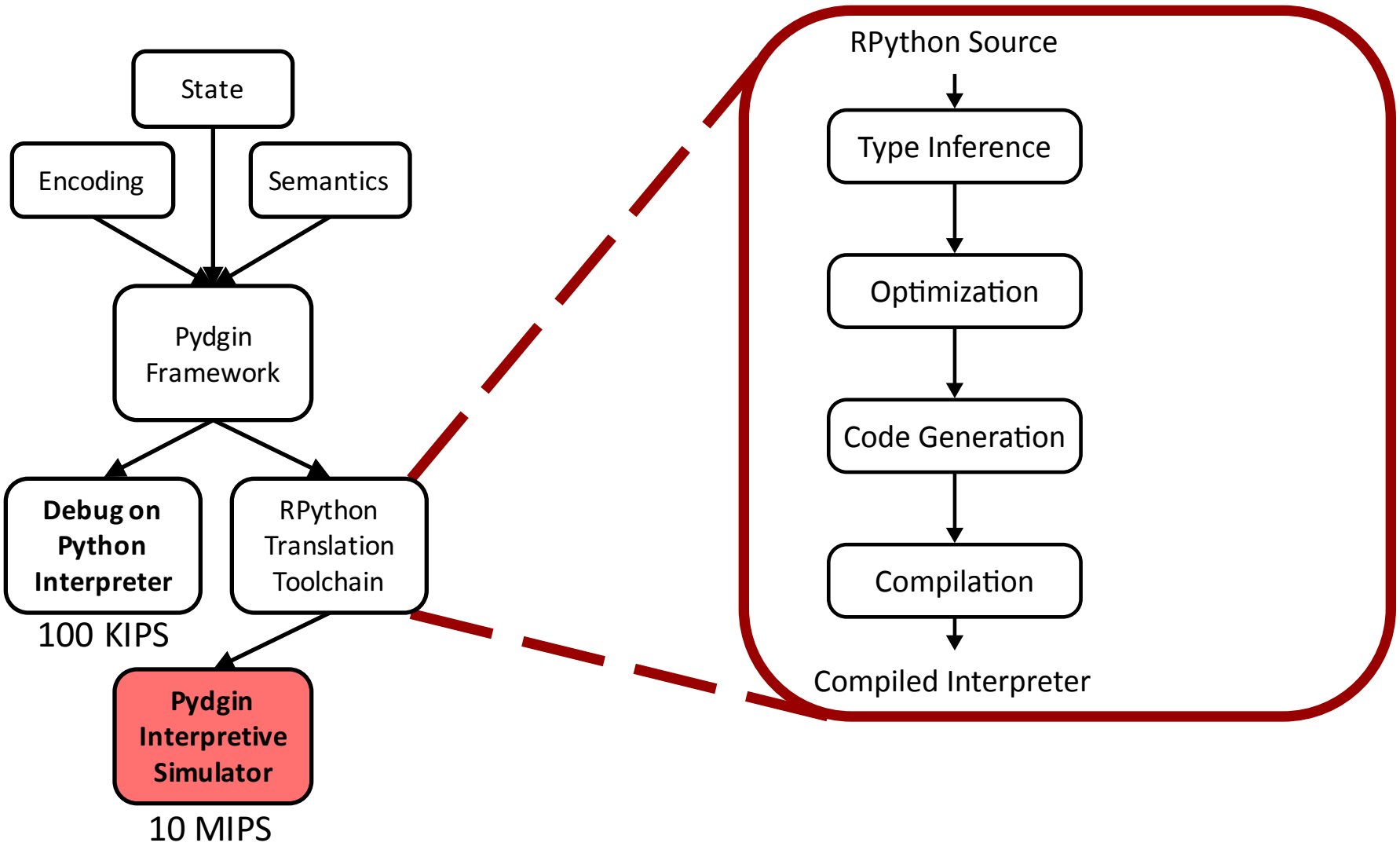
Interpreter Loop

```
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
```

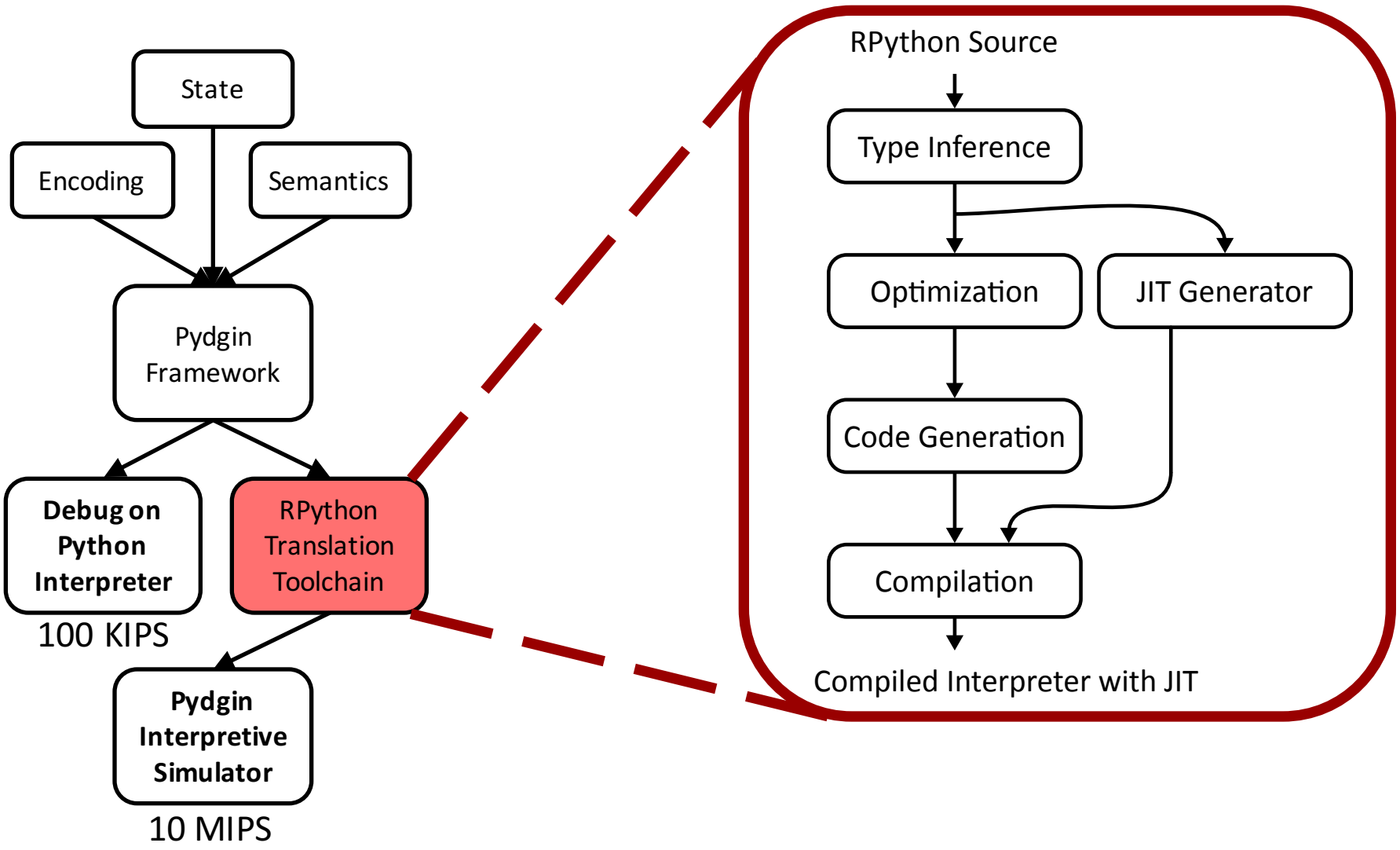
The RPython Translation Toolchain



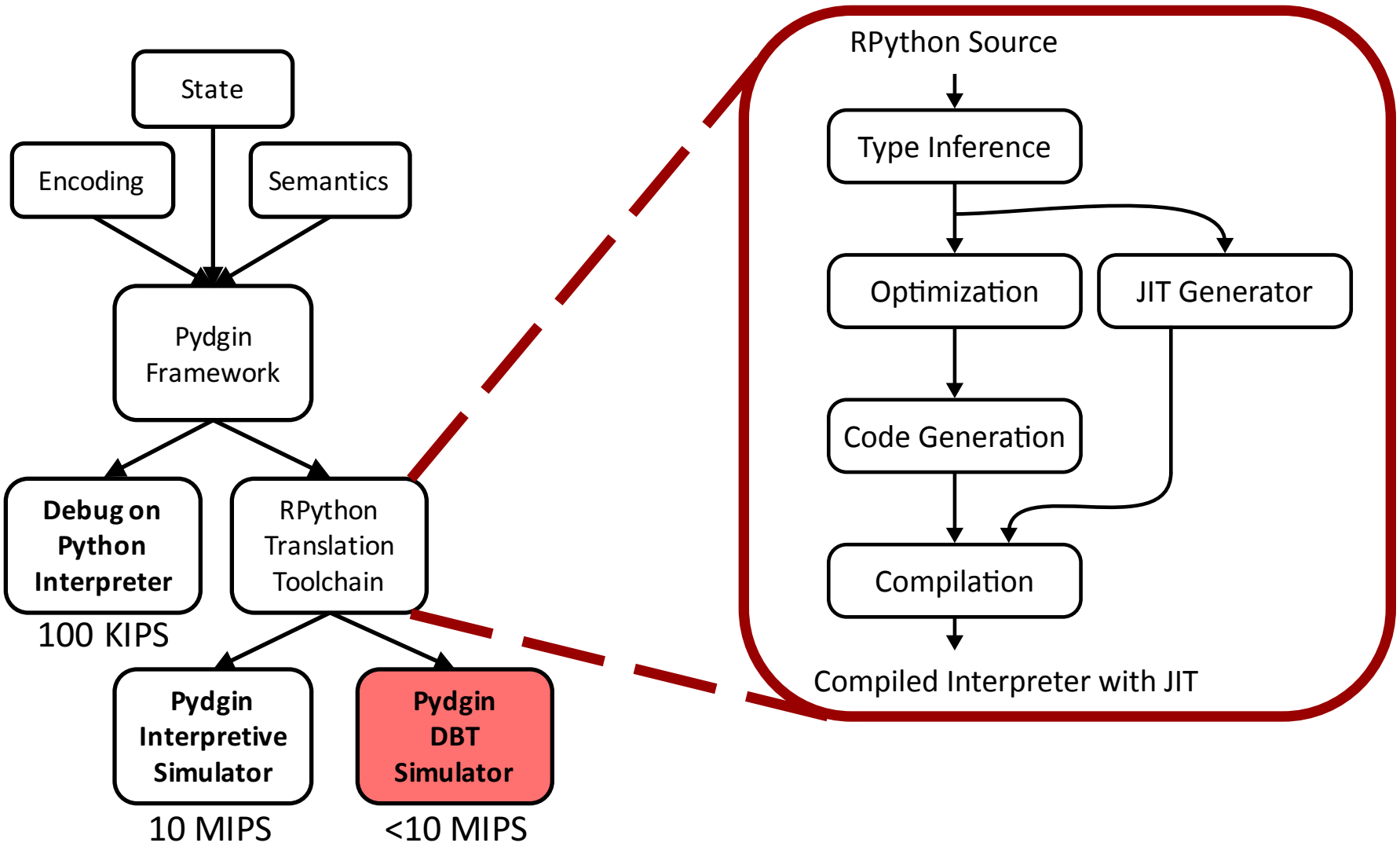
The RPython Translation Toolchain



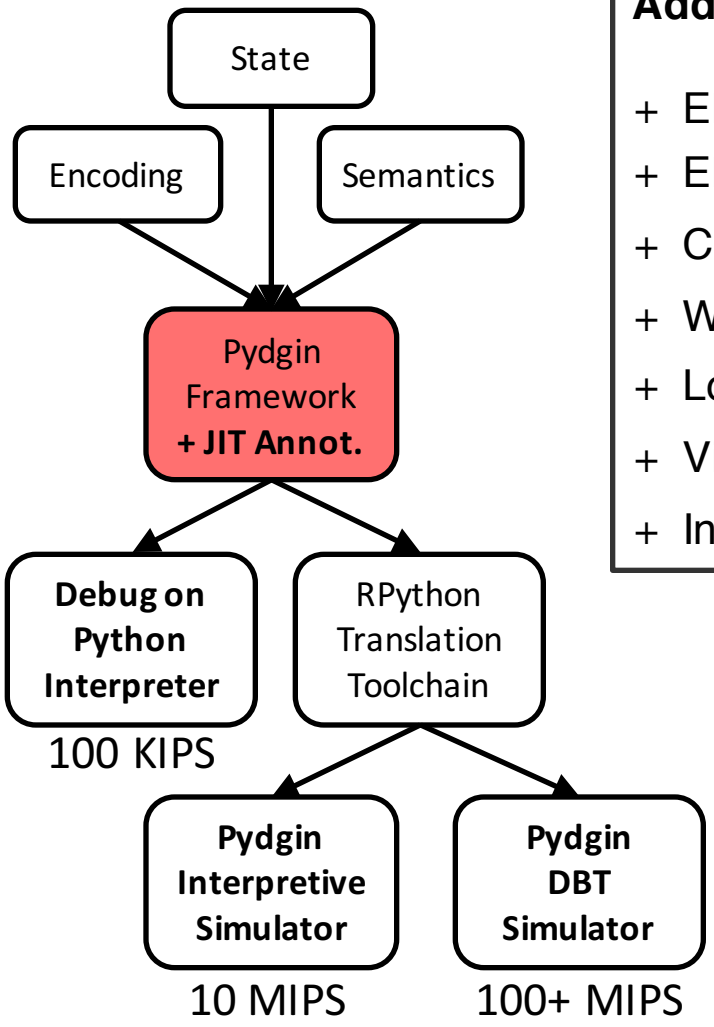
The RPython Translation Toolchain



The RPython Translation Toolchain

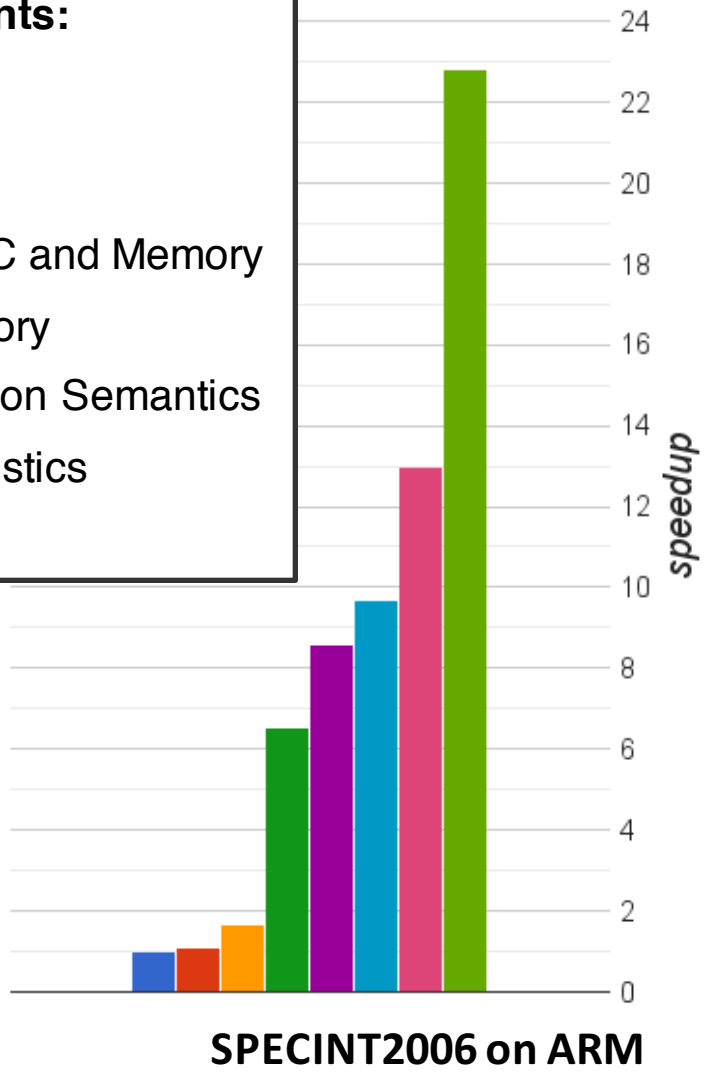


JIT Annotations and Optimizations

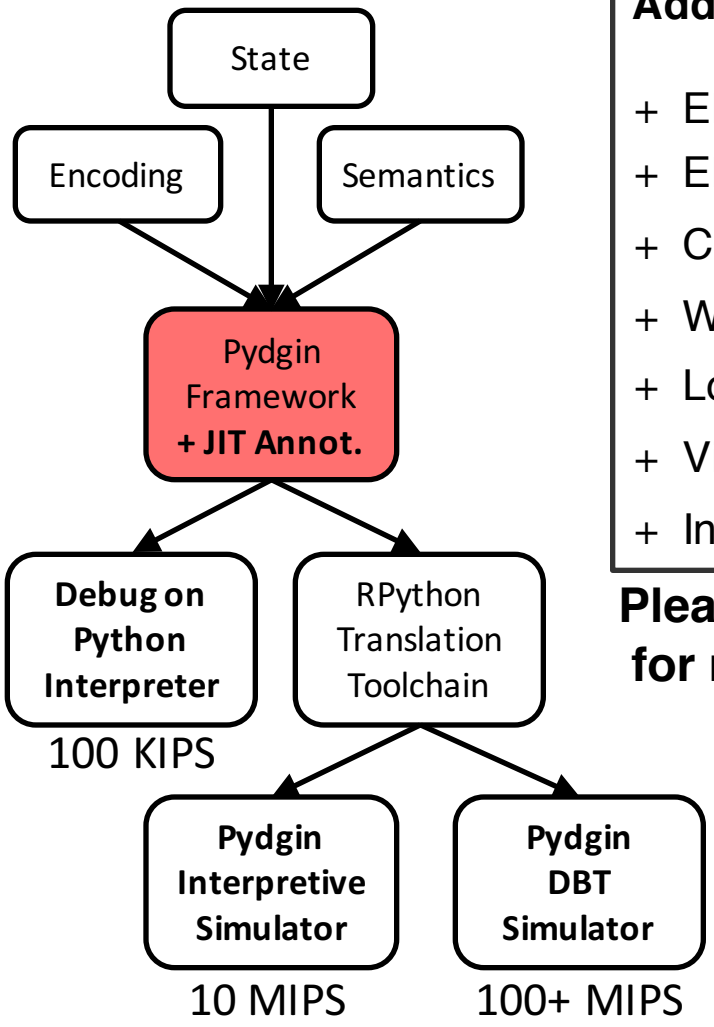


Additional RPython JIT hints:

+ Elidable Instruction Fetch
+ Elidable Decode
+ Constant Promotion of PC and Memory
+ Word-Based Target Memory
+ Loop Unrolling in Instruction Semantics
+ Virtualizable PC and Statistics
+ Increased Trace Limit



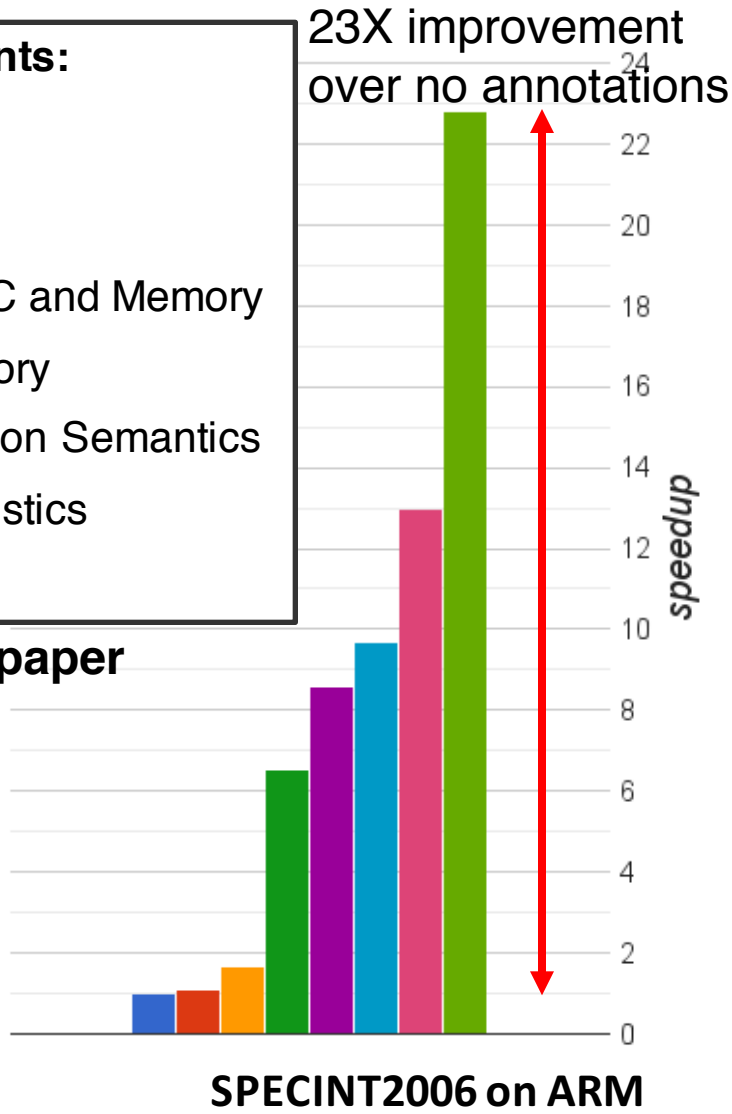
JIT Annotations and Optimizations



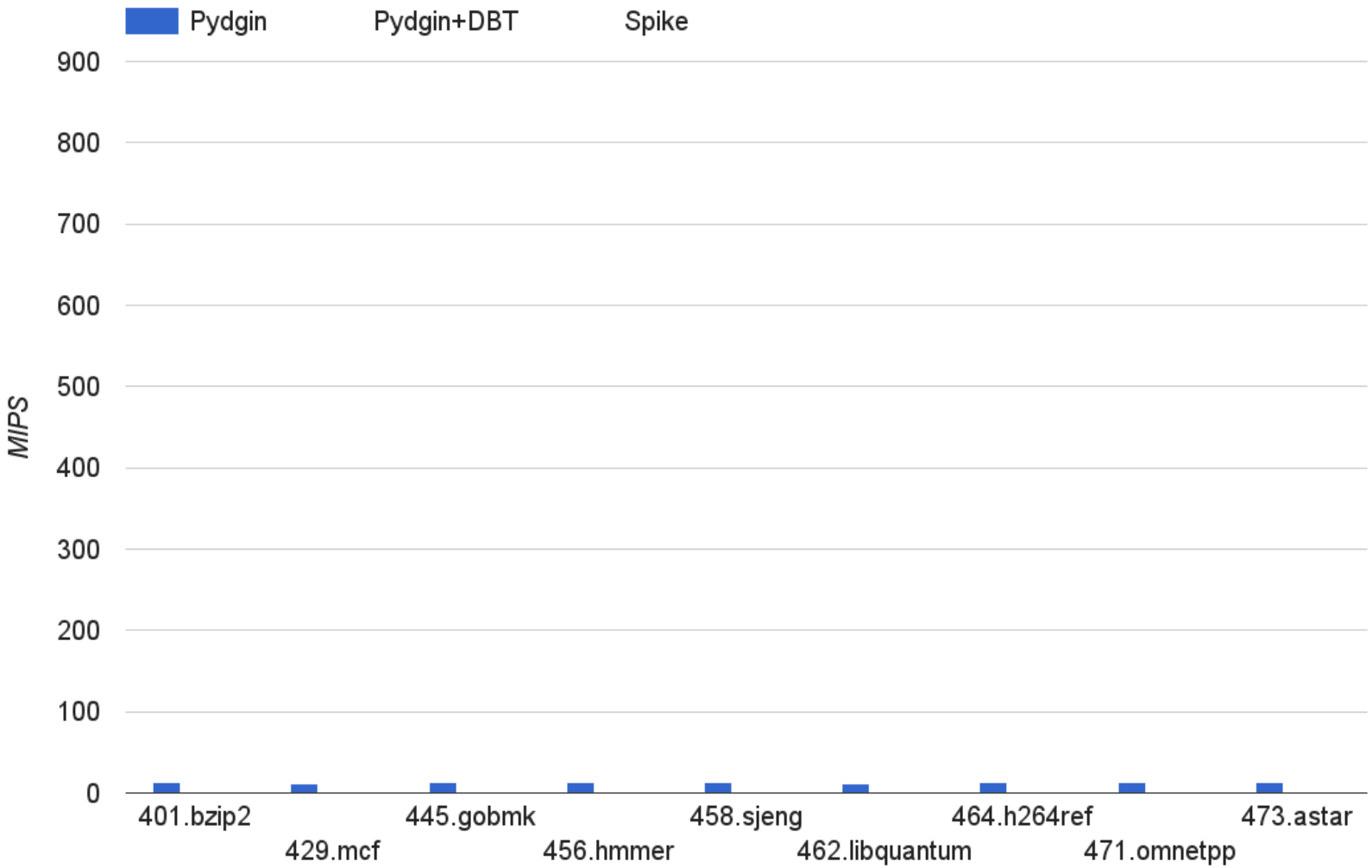
Additional RPython JIT hints:

+ Elidable Instruction Fetch
+ Elidable Decode
+ Constant Promotion of PC and Memory
+ Word-Based Target Memory
+ Loop Unrolling in Instruction Semantics
+ Virtualizable PC and Statistics
+ Increased Trace Limit

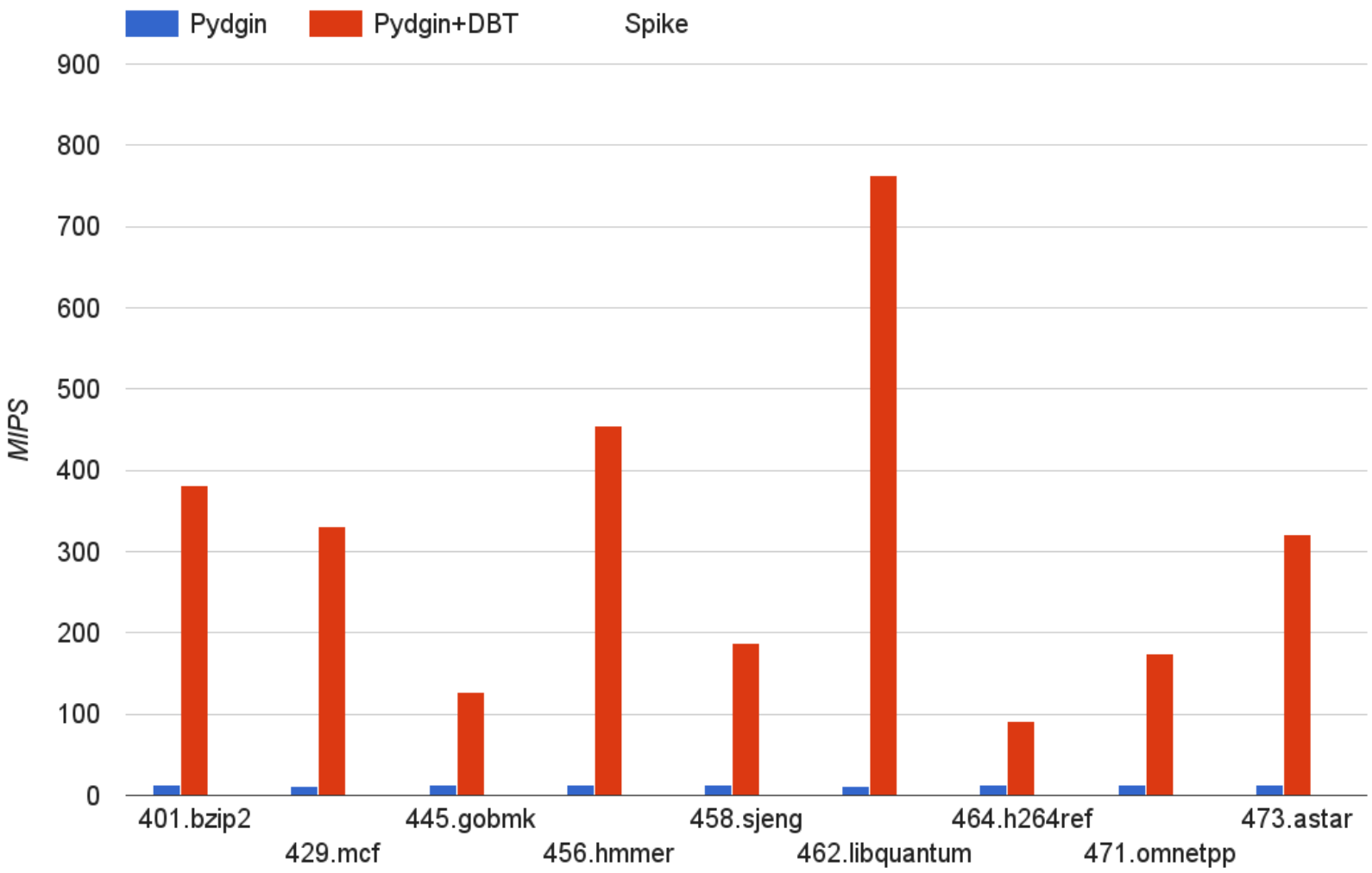
Please see our ISPASS paper for more details!



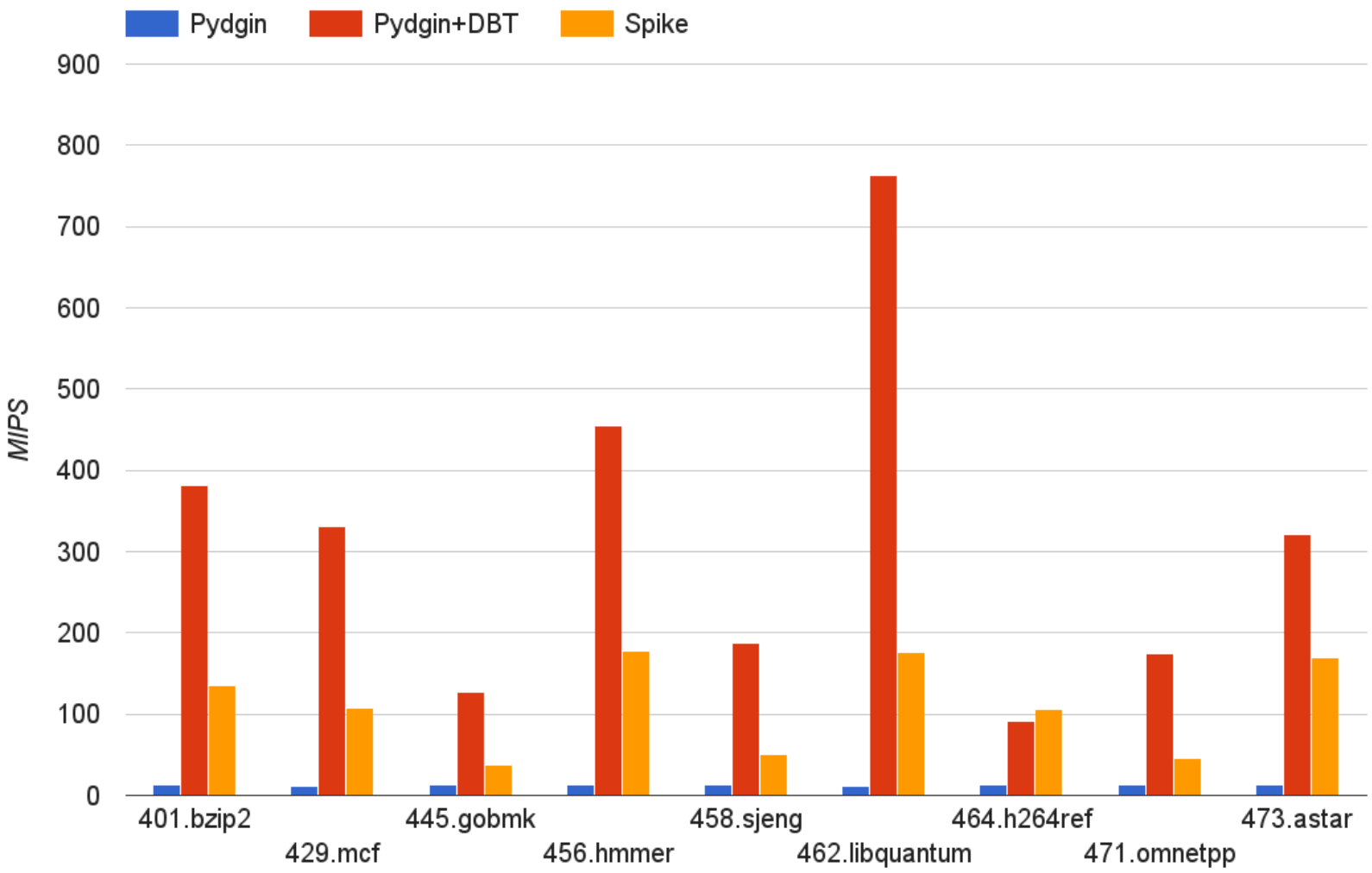
Pydgin Performance



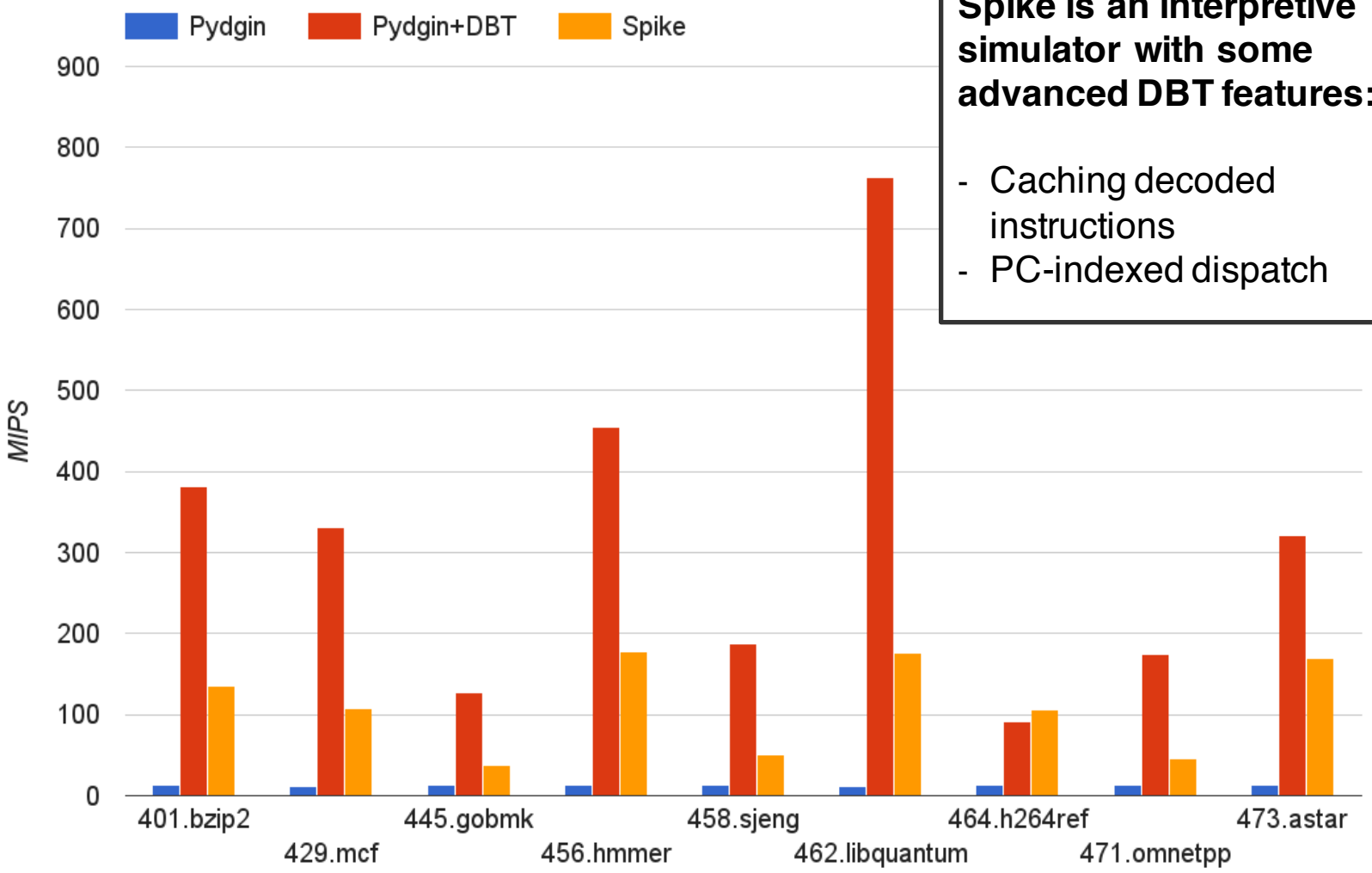
Pydgin Performance



Pydgin Performance



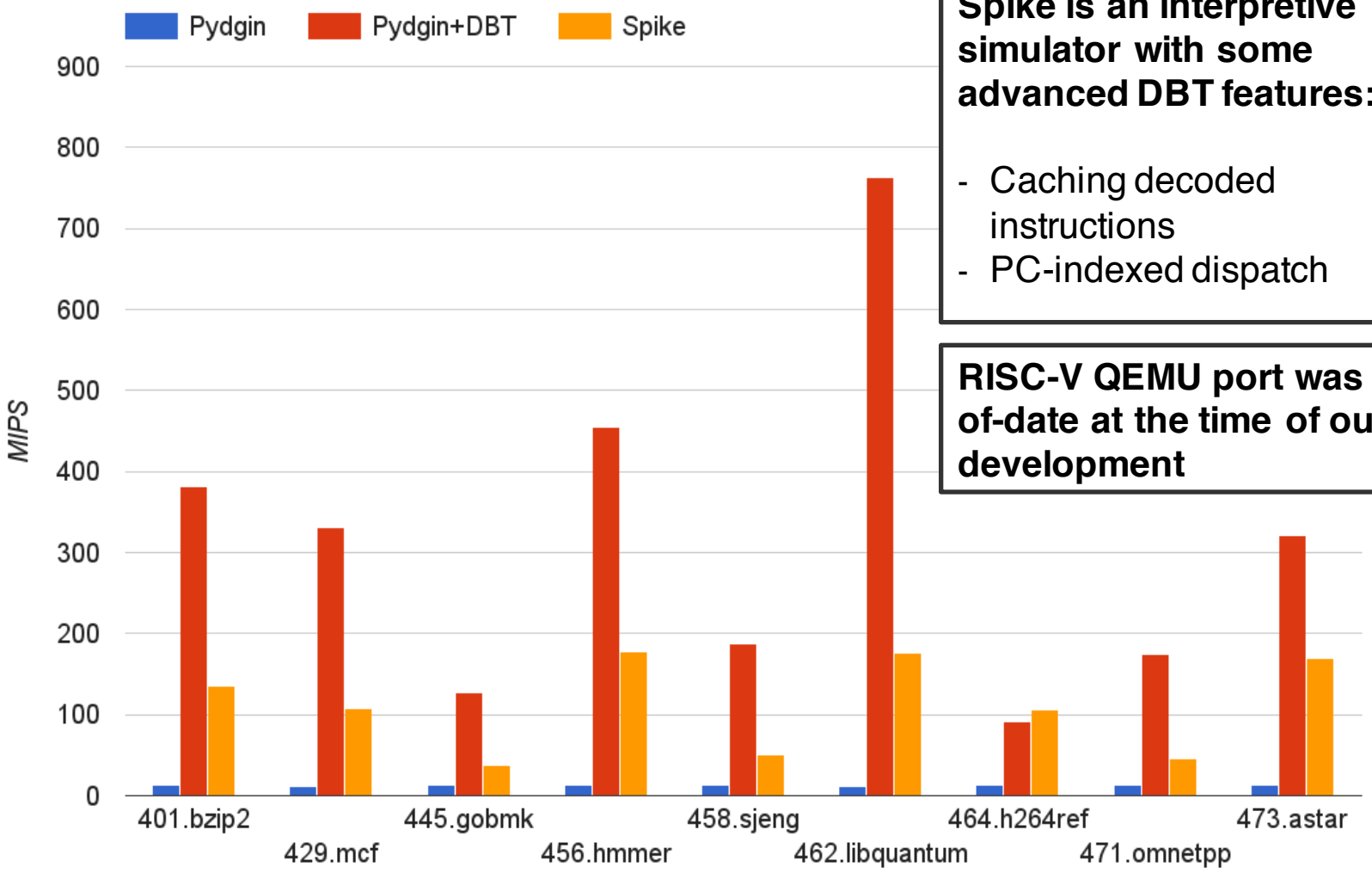
Pydgin Performance



Spike is an interpretive simulator with some advanced DBT features:

- Caching decoded instructions
- PC-indexed dispatch

Pydgin Performance



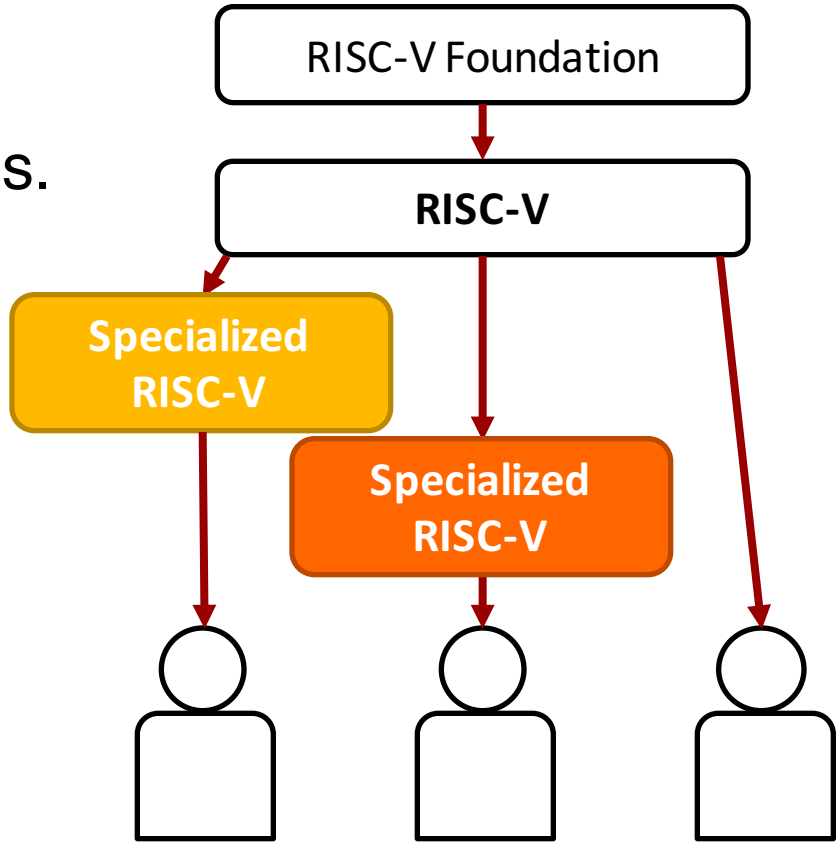
Spike is an interpretive simulator with some advanced DBT features:

- Caching decoded instructions
- PC-indexed dispatch

RISC-V QEMU port was out-of-date at the time of our development

Pydgin Productivity

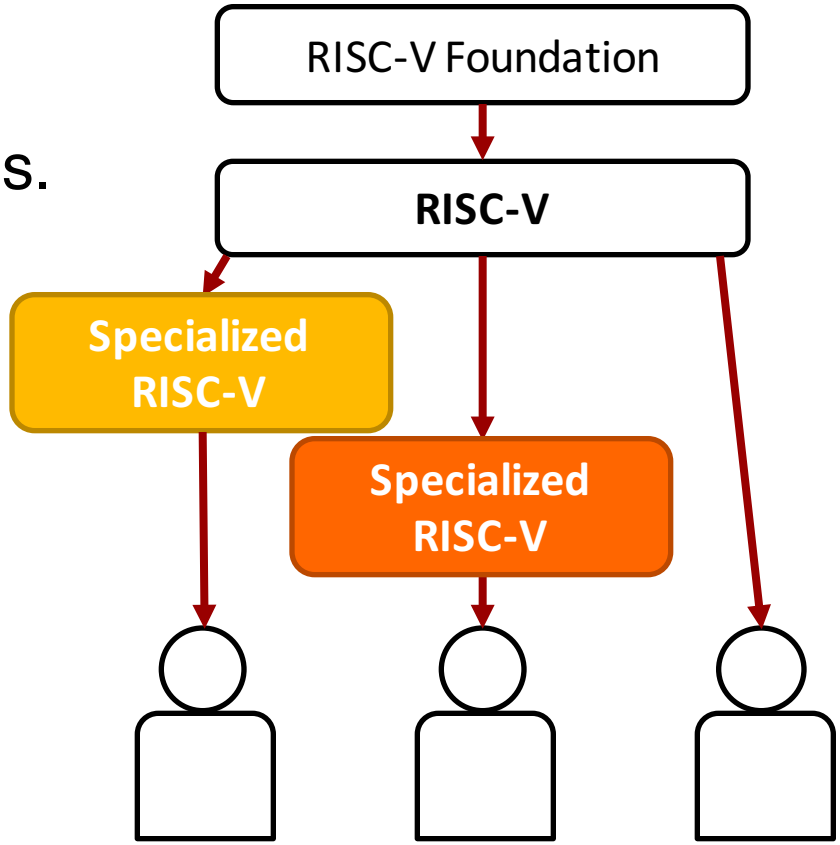
RISC-V encourages ISA extensions.



Pydgin Productivity

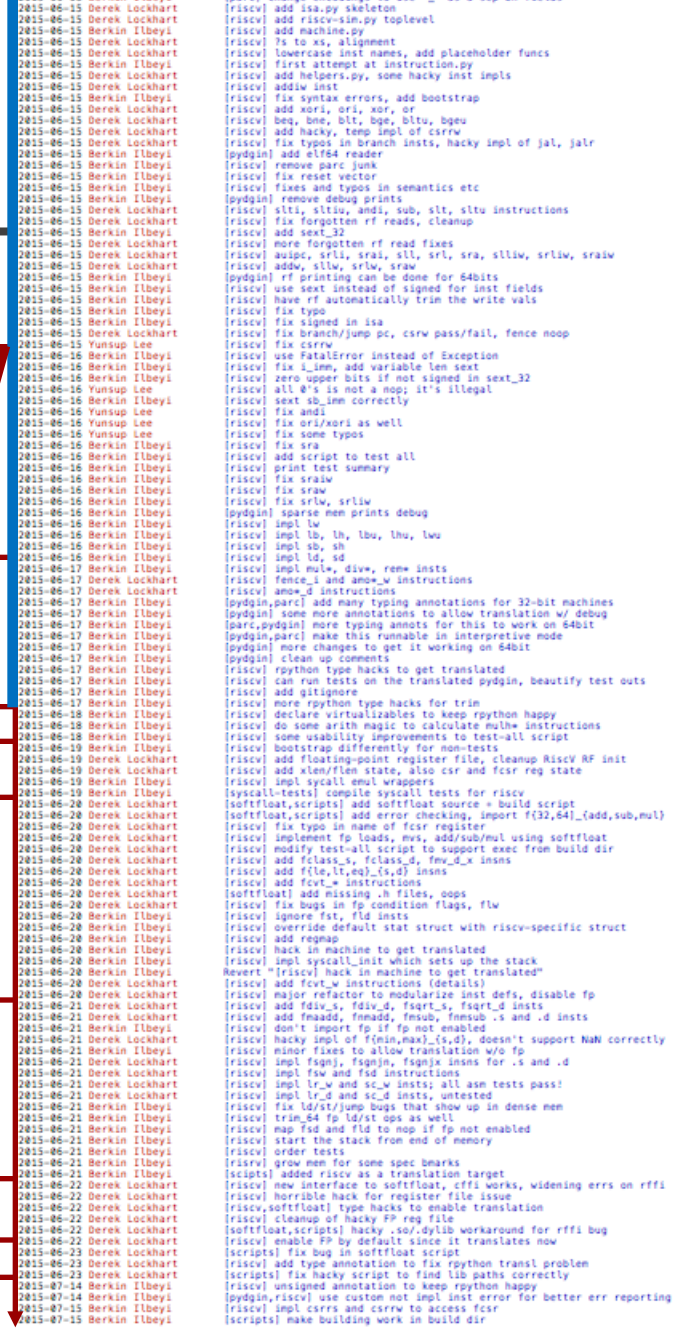
RISC-V encourages ISA extensions.

- Productive Development
- Productive Extensibility
- Productive Instrumentation



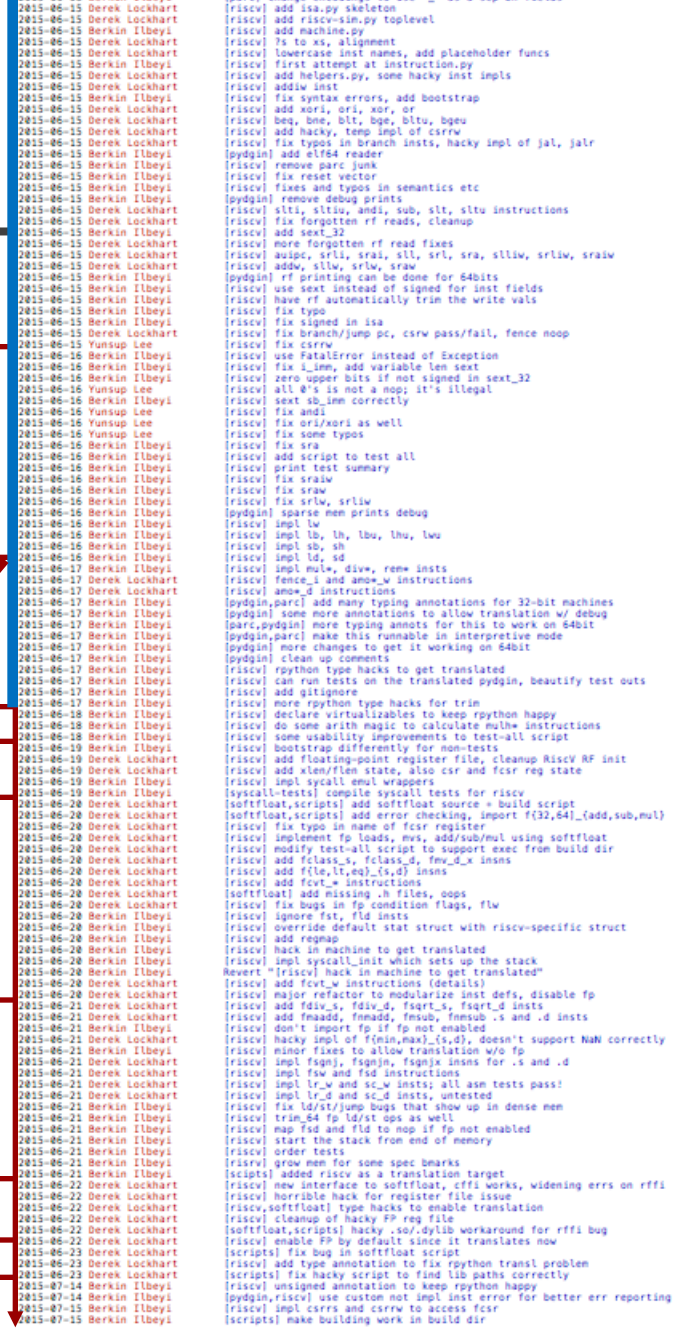
Pydgin RISC-V Development

```
2015-06-15 Derek Lockhart    [riscv] add isa.py skeleton
2015-06-15 Derek Lockhart    [riscv] add riscv-sim.py toplevel
2015-06-15 Berkin Ilbeyi     [riscv] add machine.py
2015-06-15 Derek Lockhart    [riscv] ?s to xs, alignment
2015-06-15 Derek Lockhart    [riscv] lowercase inst names, add placeholder funcs
2015-06-15 Berkin Ilbeyi     [riscv] first attempt at instruction.py
2015-06-15 Derek Lockhart    [riscv] add helpers.py, some hacky inst impls
2015-06-15 Derek Lockhart    [riscv] addiw inst
2015-06-15 Berkin Ilbeyi     [riscv] fix syntax errors, add bootstrap
2015-06-15 Derek Lockhart    [riscv] add xori, ori, xor, or
2015-06-15 Derek Lockhart    [riscv] beq, bne, blt, bge, bltu, bgeu
2015-06-15 Derek Lockhart    [riscv] add hacky, temp impl of csrrw
2015-06-15 Derek Lockhart    [riscv] fix typos in branch insts, hacky impl of jal, jalr
2015-06-15 Berkin Ilbeyi     [pydgin] add elf64 reader
2015-06-15 Berkin Ilbeyi     [riscv] remove parc junk
2015-06-15 Berkin Ilbeyi     [riscv] fix reset vector
2015-06-15 Berkin Ilbeyi     [riscv] fixes and typos in semantics etc
2015-06-15 Berkin Ilbeyi     [pydgin] remove debug prints
2015-06-15 Derek Lockhart    [riscv] slti, sltiu, andi, sub, slt, sltu instructions
2015-06-15 Derek Lockhart    [riscv] fix forgotten rf reads, cleanup
2015-06-15 Berkin Ilbeyi     [riscv] add sext_32
2015-06-15 Derek Lockhart    [riscv] more forgotten rf read fixes
2015-06-15 Derek Lockhart    [riscv] auipc, srli, srai, sll, srl, sra, slliw, srliw, sraiw
2015-06-15 Derek Lockhart    [riscv] addw, sllw, srlw, sraw
2015-06-15 Berkin Ilbeyi     [pydgin] rf printing can be done for 64bits
2015-06-15 Berkin Ilbeyi     [riscv] use sext instead of signed for inst fields
2015-06-15 Berkin Ilbeyi     [riscv] have rf automatically trim the write vals
2015-06-15 Berkin Ilbeyi     [riscv] fix typo
2015-06-15 Berkin Ilbeyi     [riscv] fix signed in isa
2015-06-15 Derek Lockhart    [riscv] fix branch/jump pc, csrw pass/fail, fence noop
2015-06-15 Yunsup Lee        [riscv] fix csrrw
```



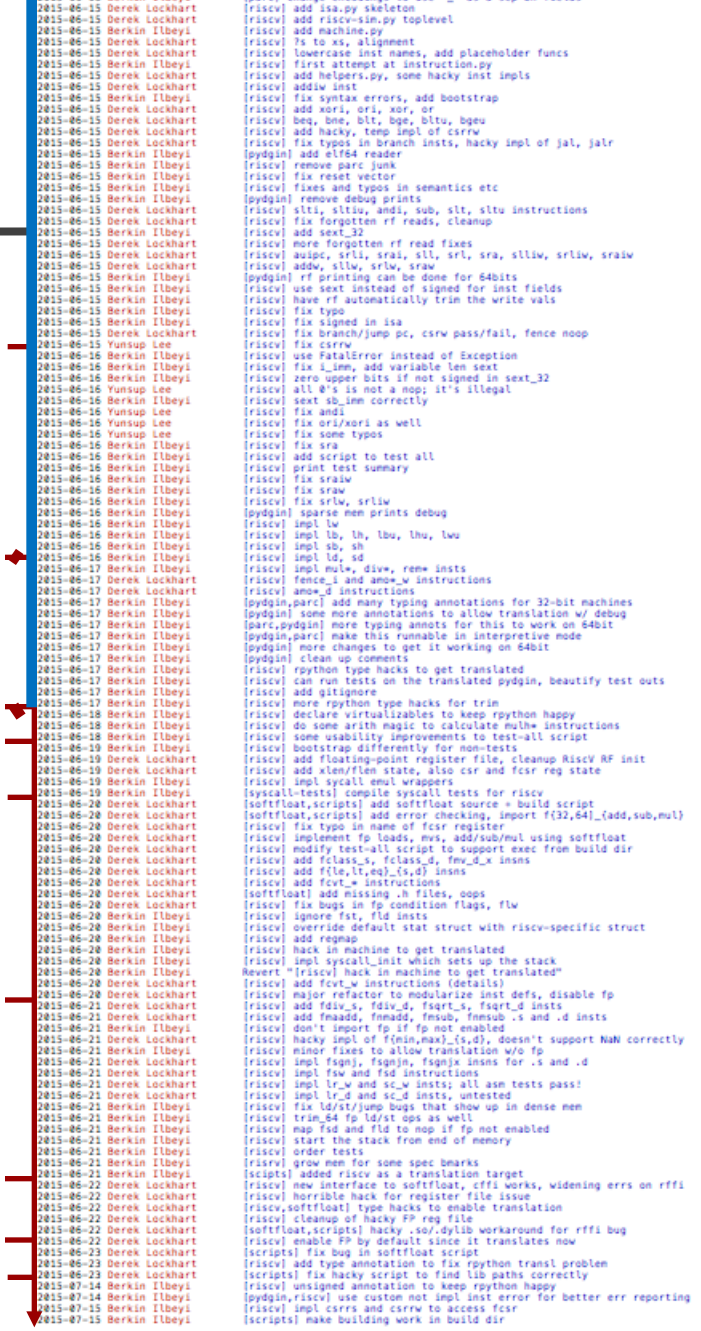
Pydgin RISC-V Development

```
2015-06-16 Berkin Ilbeyi        [riscv] use FatalError instead of Exception
2015-06-16 Berkin Ilbeyi        [riscv] fix i_imm, add variable len sext
2015-06-16 Berkin Ilbeyi        [riscv] zero upper bits if not signed in sext_32
2015-06-16 Yunsup Lee           [riscv] all 0's is not a nop; it's illegal
2015-06-16 Berkin Ilbeyi        [riscv] sext sb_imm correctly
2015-06-16 Yunsup Lee           [riscv] fix andi
2015-06-16 Yunsup Lee           [riscv] fix ori/xori as well
2015-06-16 Yunsup Lee           [riscv] fix some typos
2015-06-16 Berkin Ilbeyi        [riscv] fix sra
2015-06-16 Berkin Ilbeyi        [riscv] add script to test all
2015-06-16 Berkin Ilbeyi        [riscv] print test summary
2015-06-16 Berkin Ilbeyi        [riscv] fix sraiw
2015-06-16 Berkin Ilbeyi        [riscv] fix sraw
2015-06-16 Berkin Ilbeyi        [riscv] fix srlw, srliw
2015-06-16 Berkin Ilbeyi        [pydgin] sparse mem prints debug
2015-06-16 Berkin Ilbeyi        [riscv] impl lw
2015-06-16 Berkin Ilbeyi        [riscv] impl lb, lh, lbu, lhu, lwu
2015-06-16 Berkin Ilbeyi        [riscv] impl sb, sh
2015-06-16 Berkin Ilbeyi        [riscv] impl ld, sd
```

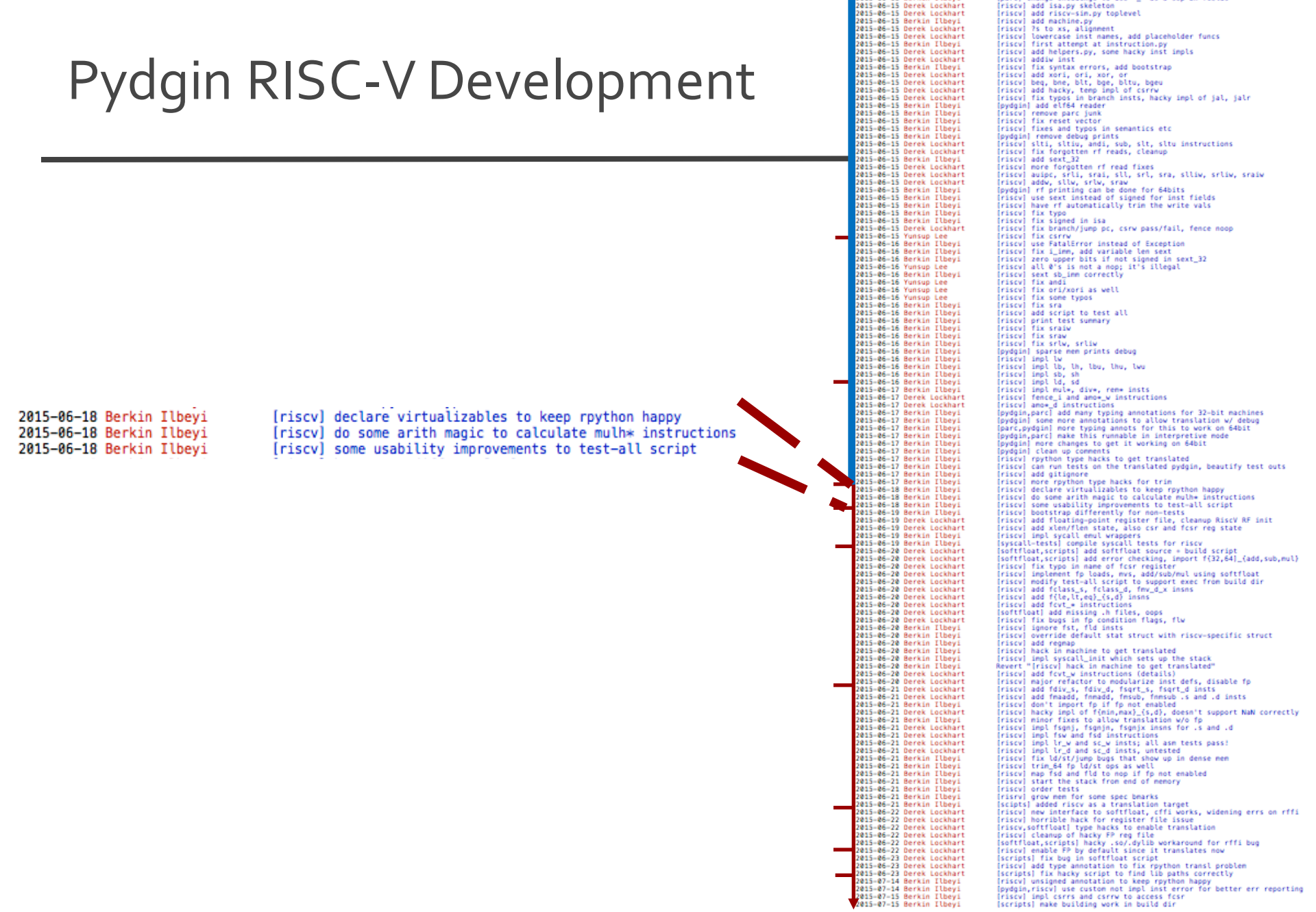


Pydgin RISC-V Development

```
2015-06-17 Berkin Ilbeyi        [riscv] impl mul*, div*, rem* insts
2015-06-17 Derek Lockhart       [riscv] fence_i and amo*_w instructions
2015-06-17 Derek Lockhart       [riscv] amo*_d instructions
2015-06-17 Berkin Ilbeyi        [pydgin,parc] add many typing annotations for 32-bit machines
2015-06-17 Berkin Ilbeyi        [pydgin] some more annotations to allow translation w/ debug
2015-06-17 Berkin Ilbeyi        [parc,pydgin] more typing annots for this to work on 64bit
2015-06-17 Berkin Ilbeyi        [pydgin,parc] make this runnable in interpretive mode
2015-06-17 Berkin Ilbeyi        [pydgin] more changes to get it working on 64bit
2015-06-17 Berkin Ilbeyi        [pydgin] clean up comments
2015-06-17 Berkin Ilbeyi        [riscv] rpython type hacks to get translated
2015-06-17 Berkin Ilbeyi        [riscv] can run tests on the translated pydgin, beautify test outs
2015-06-17 Berkin Ilbeyi        [riscv] add gitignore
2015-06-17 Berkin Ilbeyi        [riscv] more rpython type hacks for trim
```

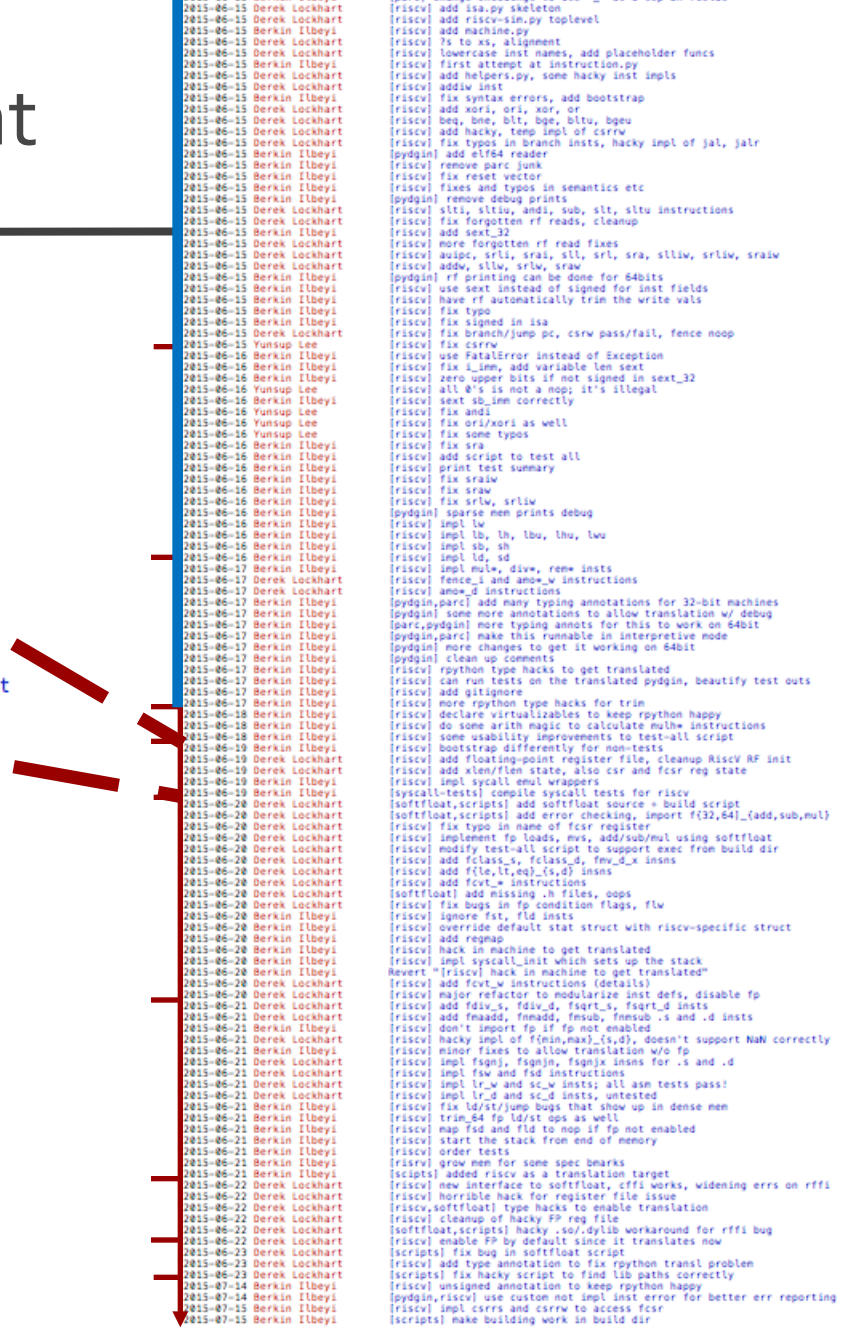


Pydgin RISC-V Development



Pydgin RISC-V Development

```
2015-06-19 Berkin Ilbeyi      [riscv] bootstrap differently for non-tests
2015-06-19 Derek Lockhart     [riscv] add floating-point register file, cleanup RiscV RF init
2015-06-19 Derek Lockhart     [riscv] add xlen/flen state, also csr and fcsr reg state
2015-06-19 Berkin Ilbeyi      [riscv] impl sycall emul wrappers
2015-06-19 Berkin Ilbeyi      [syscall-tests] compile syscall tests for riscv
```

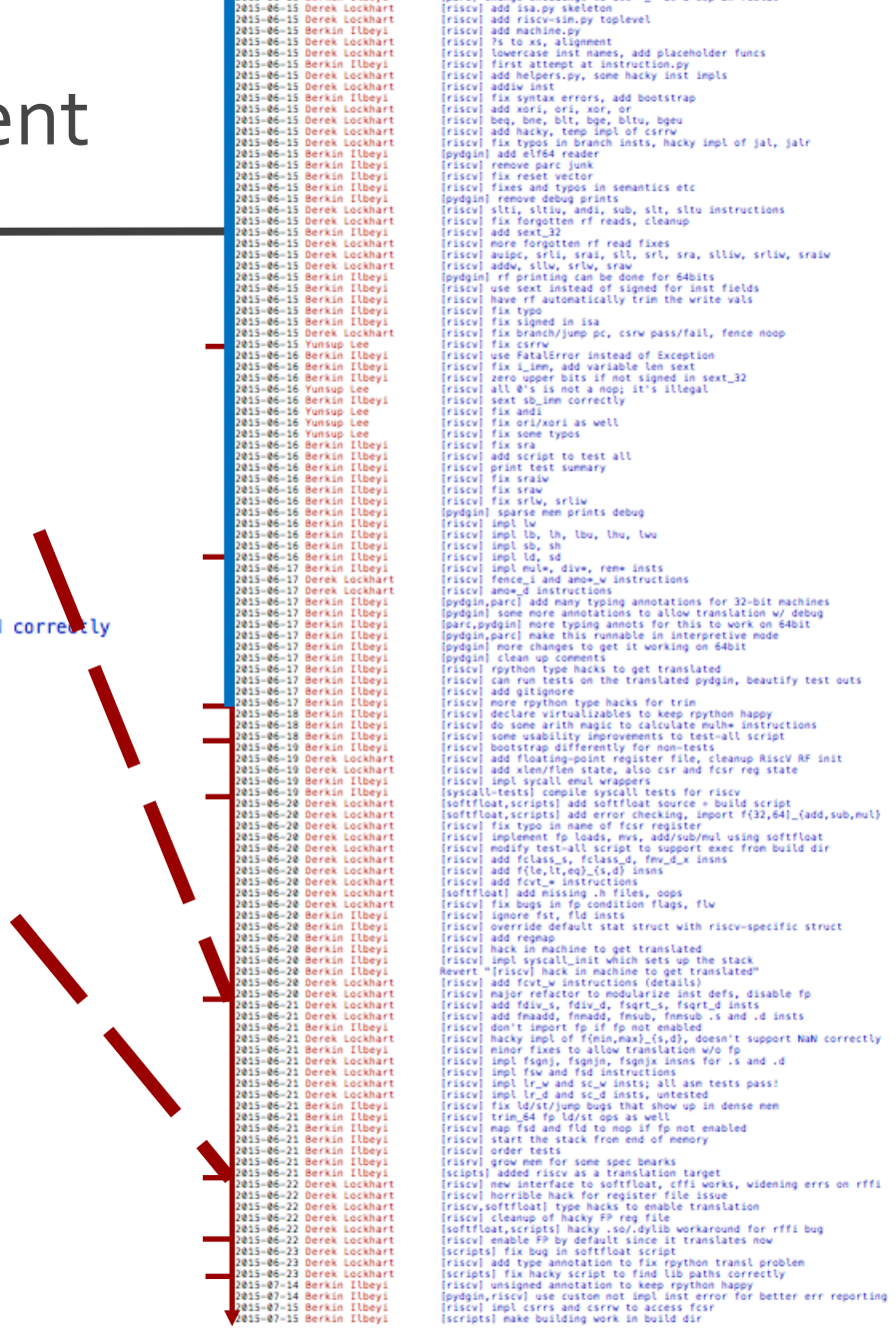


Pydgin RISC-V Development

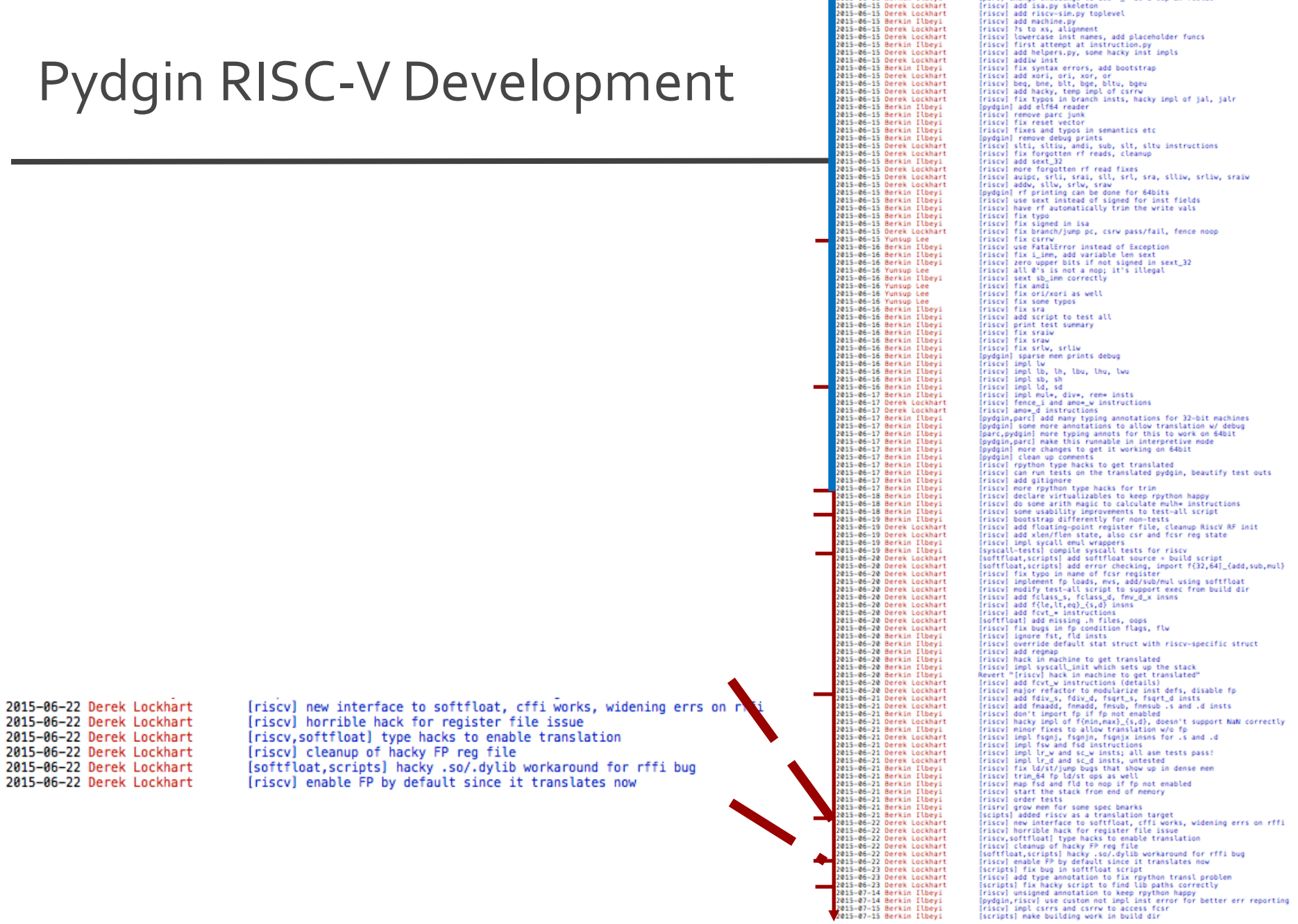
```
2015-06-20 Derek Lockhart       [softfloat,scripts] add softfloat source + build script
2015-06-20 Derek Lockhart       [softfloat,scripts] add error checking, import f{32,64}_{add,sub,mu
2015-06-20 Derek Lockhart       [riscv] fix typo in name of fcsr register
2015-06-20 Derek Lockhart       [riscv] implement fp loads, mvs, add/sub/mul using softfloat
2015-06-20 Derek Lockhart       [riscv] modify test-all script to support exec from build dir
2015-06-20 Derek Lockhart       [riscv] add fclass_s, fclass_d, fmv_d_x insns
2015-06-20 Derek Lockhart       [riscv] add f{le,lt,eq}_{s,d} insns
2015-06-20 Derek Lockhart       [riscv] add fcvt_* instructions
2015-06-20 Derek Lockhart       [softfloat] add missing .h files, oops
2015-06-20 Derek Lockhart       [riscv] fix bugs in fp condition flags, flw
2015-06-20 Berkin Ilbeyi        [riscv] ignore fst, fld insts
2015-06-20 Berkin Ilbeyi        [riscv] override default stat struct with riscv-specific struct
2015-06-20 Berkin Ilbeyi        [riscv] add regmap
2015-06-20 Berkin Ilbeyi        [riscv] hack in machine to get translated
2015-06-20 Berkin Ilbeyi        [riscv] impl syscall_init which sets up the stack
2015-06-20 Berkin Ilbeyi       Revert "[riscv] hack in machine to get translated"
2015-06-20 Derek Lockhart       [riscv] add fcvt_w instructions (details)
2015-06-20 Derek Lockhart       [riscv] major refactor to modularize inst defs, disable fp
```

Pydgin RISC-V Development

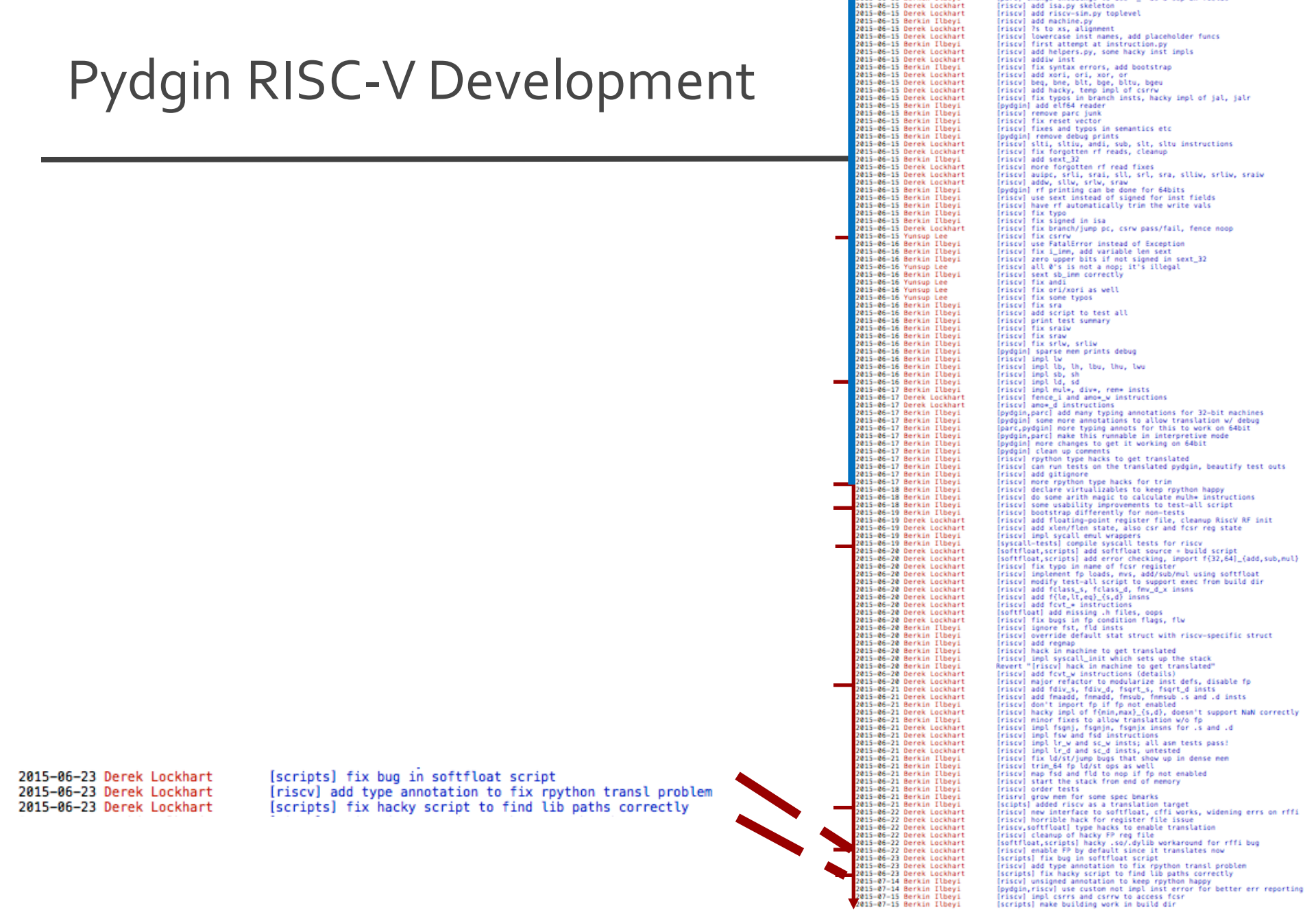
```
2015-06-21 Derek Lockhart       [riscv] add fdiv_s, fdiv_d, fsqrt_s, fsqrt_d insts
2015-06-21 Derek Lockhart       [riscv] add fmaadd, fnmadd, fmsub, fnmsub .s and .d insts
2015-06-21 Berkin Ilbeyi        [riscv] don't import fp if fp not enabled
2015-06-21 Derek Lockhart       [riscv] hacky impl of f{min,max}_{s,d}, doesn't support NaN correctly
2015-06-21 Berkin Ilbeyi        [riscv] minor fixes to allow translation w/o fp
2015-06-21 Derek Lockhart       [riscv] impl fsgnj, fsgnjn, fsgnjx insns for .s and .d
2015-06-21 Derek Lockhart       [riscv] impl fsw and fsd instructions
2015-06-21 Derek Lockhart       [riscv] impl lr_w and sc_w insts; all asm tests pass!
2015-06-21 Derek Lockhart       [riscv] impl lr_d and sc_d insts, untested
2015-06-21 Berkin Ilbeyi        [riscv] fix ld/st/jump bugs that show up in dense mem
2015-06-21 Berkin Ilbeyi        [riscv] trim_64 fp ld/st ops as well
2015-06-21 Berkin Ilbeyi        [riscv] map fsd and fld to nop if fp not enabled
2015-06-21 Berkin Ilbeyi        [riscv] start the stack from end of memory
2015-06-21 Berkin Ilbeyi        [riscv] order tests
2015-06-21 Berkin Ilbeyi        [risrv] grow mem for some spec bmarks
2015-06-21 Berkin Ilbeyi        [scipts] added riscv as a translation target
```



Pydgin RISC-V Development

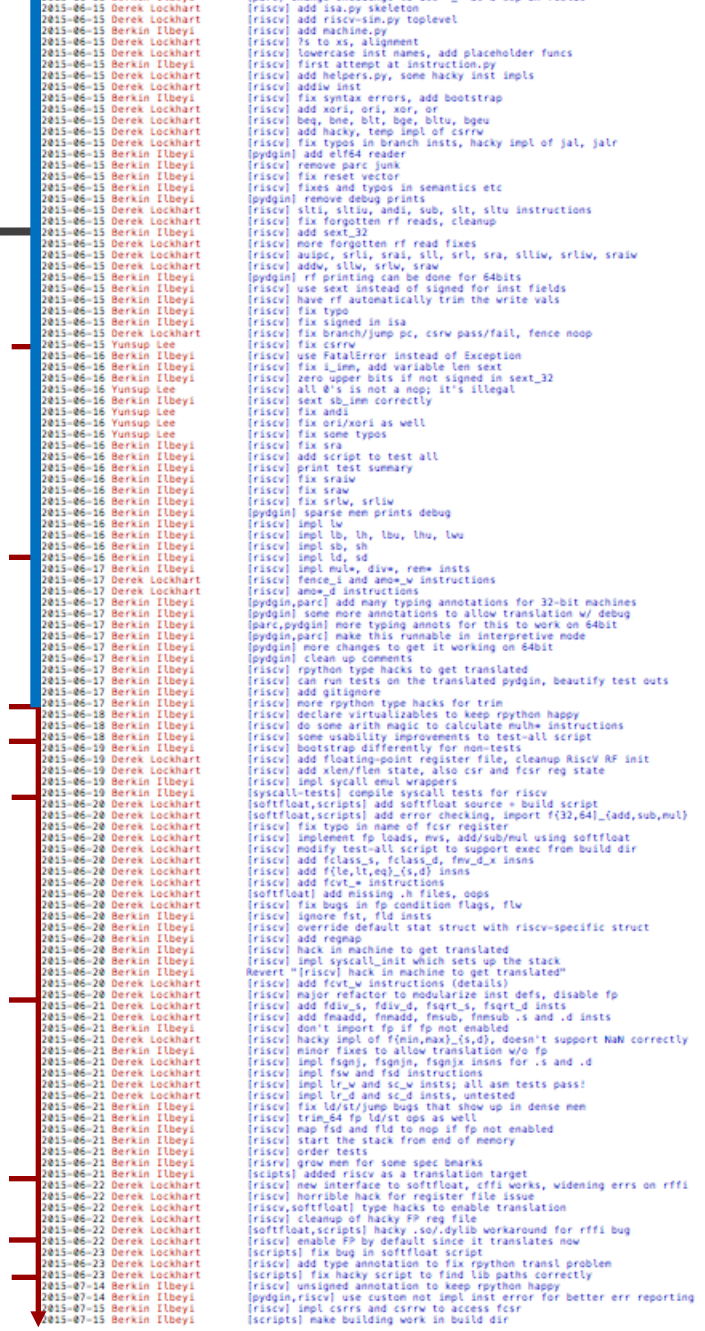


Pydgin RISC-V Development

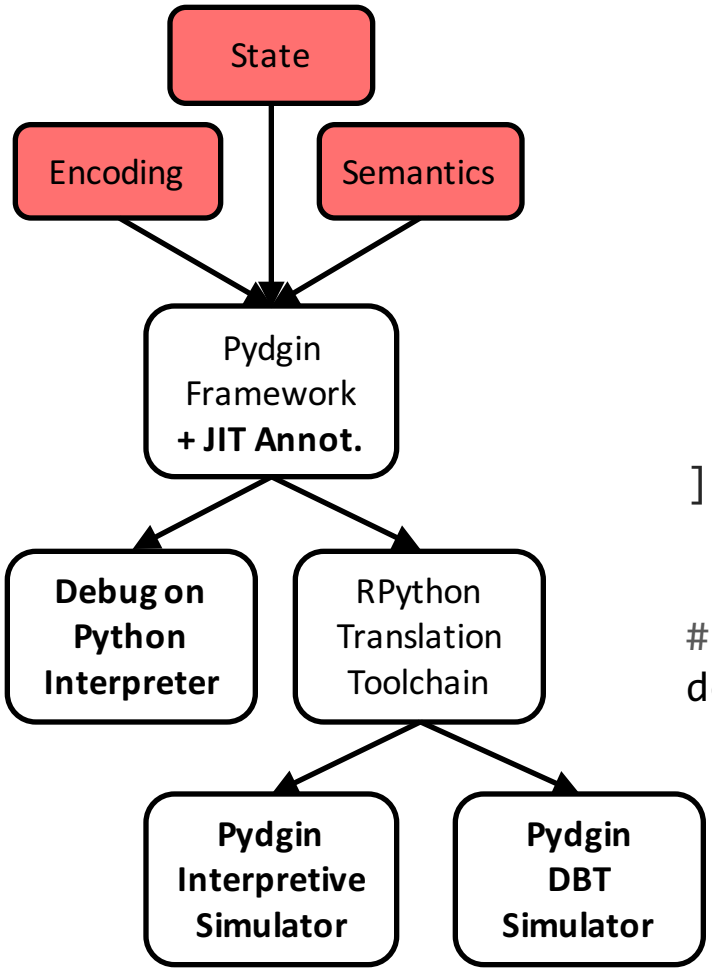


Pydgin RISC-V Development

100+ MIPS simulator
after 9 days of development!



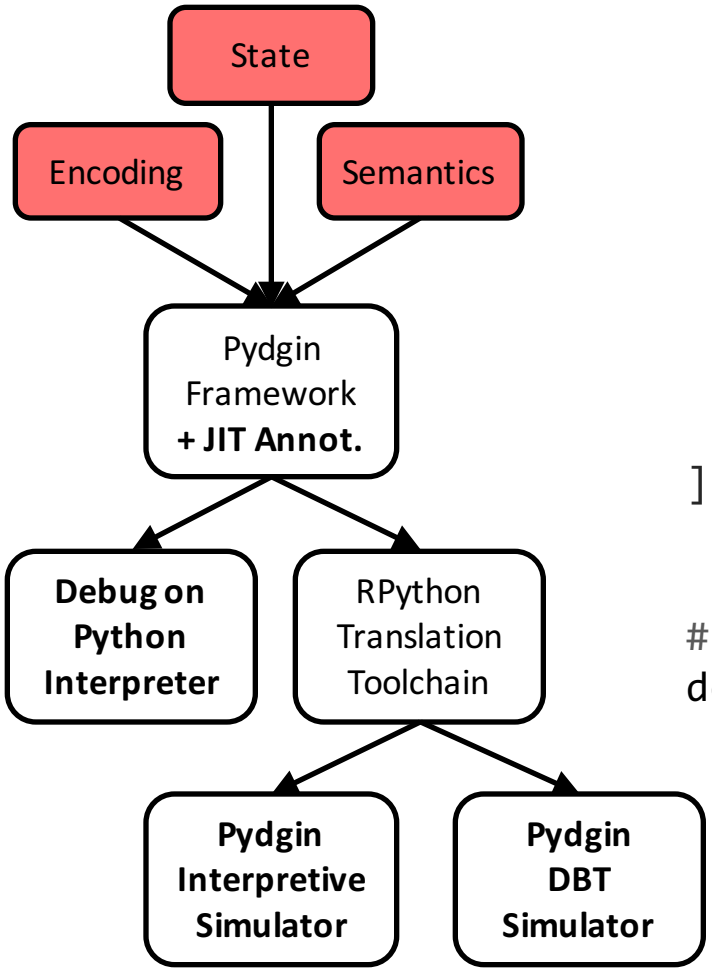
Pydgin Extensibility




```
encodings = [
  # ...
  ['andi',    'xxxxxxxxxxxxxxxxx111xxxxx0010011'],
  ['slli',    '000000xxxxxxxxxxx001xxxxx0010011'],
  ['srli',    '000000xxxxxxxxxxx101xxxxx0010011'],
  ['srai',    '010000xxxxxxxxxxx101xxxxx0010011'],
  ['add',     '0000000xxxxxxxxxx000xxxxx0110011'],
  # ...
  ['custom2', 'xxxxxxxxxxxxxxxxx000xxxxx1011011'],
  # ...
]
```

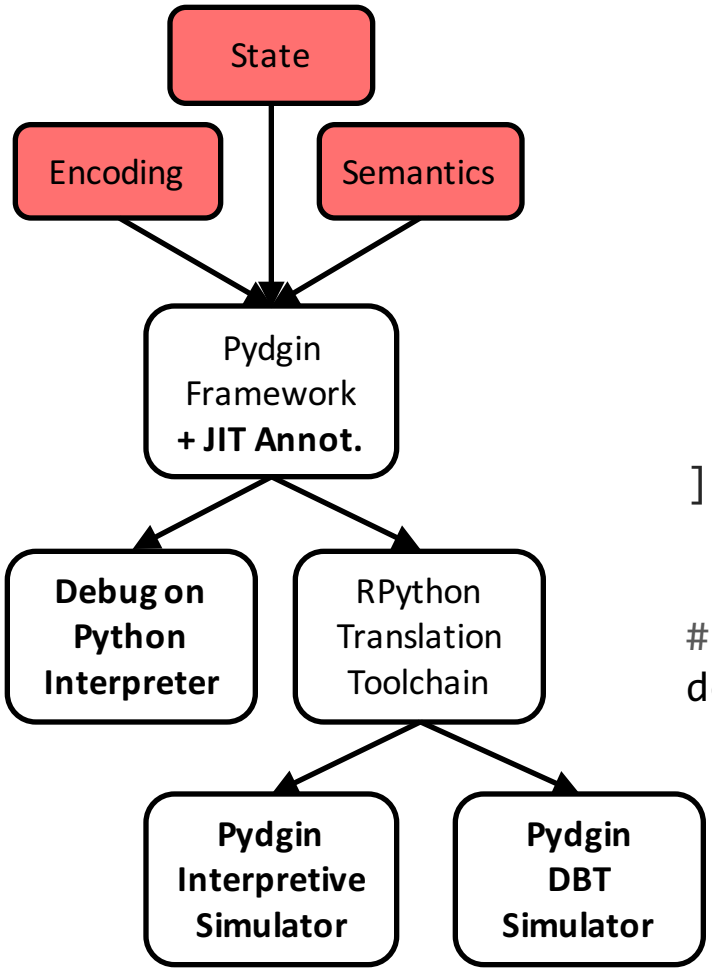
```
# greatest common divisor semantics
def execute_gcd( s, inst ):
```

Pydgin Extensibility




```
encodings = [
  # ...
  ['andi',    'xxxxxxxxxxxxxxxxx111xxxxx0010011'],
  ['slli',    '000000xxxxxxxxxxx001xxxxx0010011'],
  ['srli',    '000000xxxxxxxxxxx101xxxxx0010011'],
  ['srai',    '010000xxxxxxxxxxx101xxxxx0010011'],
  ['add',     '0000000xxxxxxxxxx000xxxxx0110011'],
  # ...
  ['gcd',     'xxxxxxxxxxxxxxxxx000xxxxx1011011'],
  # ...
]


# greatest common divisor semantics
def execute_gcd( s, inst ):
```

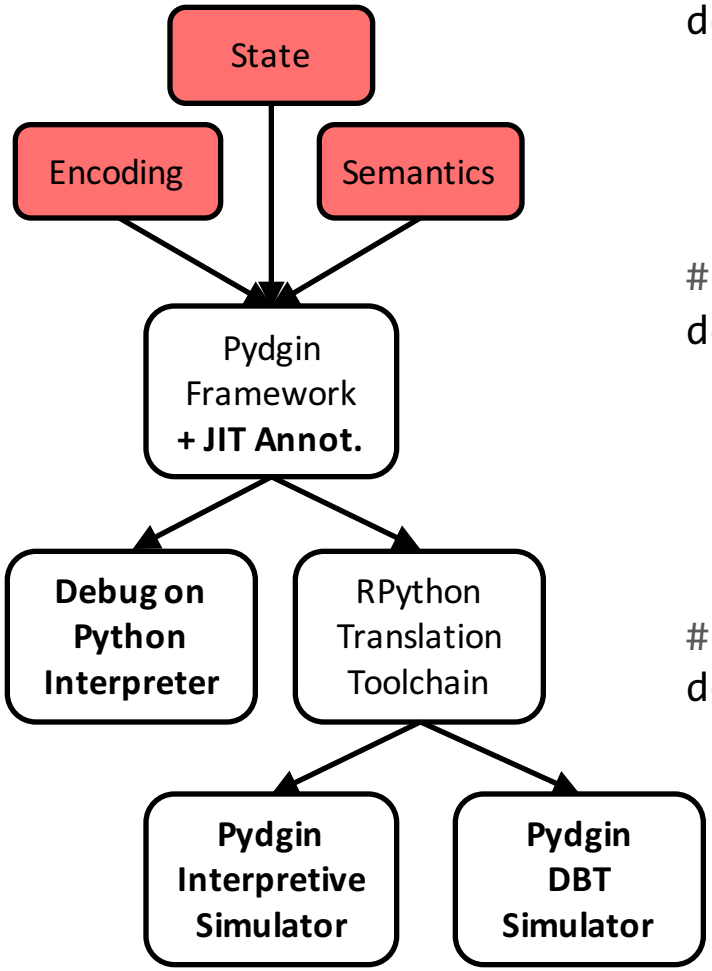
Pydgin Extensibility




```
encodings = [
  # ...
  ['andi',    'xxxxxxxxxxxxxxxxx111xxxxx0010011'],
  ['slli',    '000000xxxxxxxxxxx001xxxxx0010011'],
  ['srli',    '000000xxxxxxxxxxx101xxxxx0010011'],
  ['srai',    '010000xxxxxxxxxxx101xxxxx0010011'],
  ['add',     '0000000xxxxxxxxxx000xxxxx0110011'],
  # ...
  ['gcd',     'xxxxxxxxxxxxxxxxx000xxxxx1011011'],
  # ...
]
```

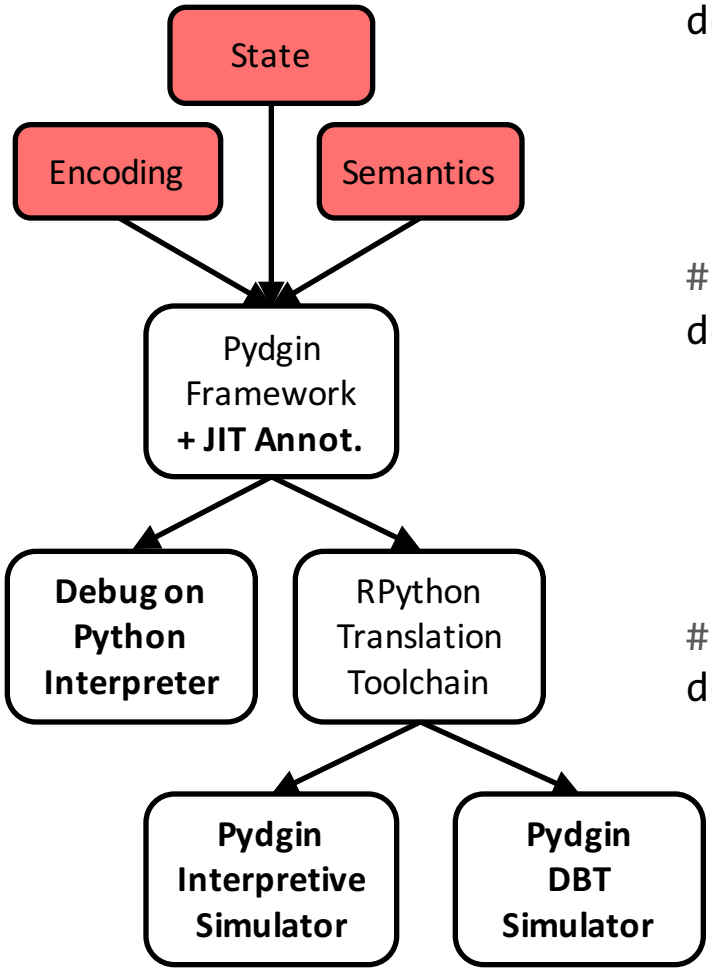
```
# greatest common divisor semantics
def execute_gcd( s, inst ):
  a, b = s.rf[inst.rs1], s.rf[inst.rs2]
  while b:
    a, b = b, a%b
  s.rf[inst.rd] = a
  s.pc += 4
```

Pydgin Instrumentation



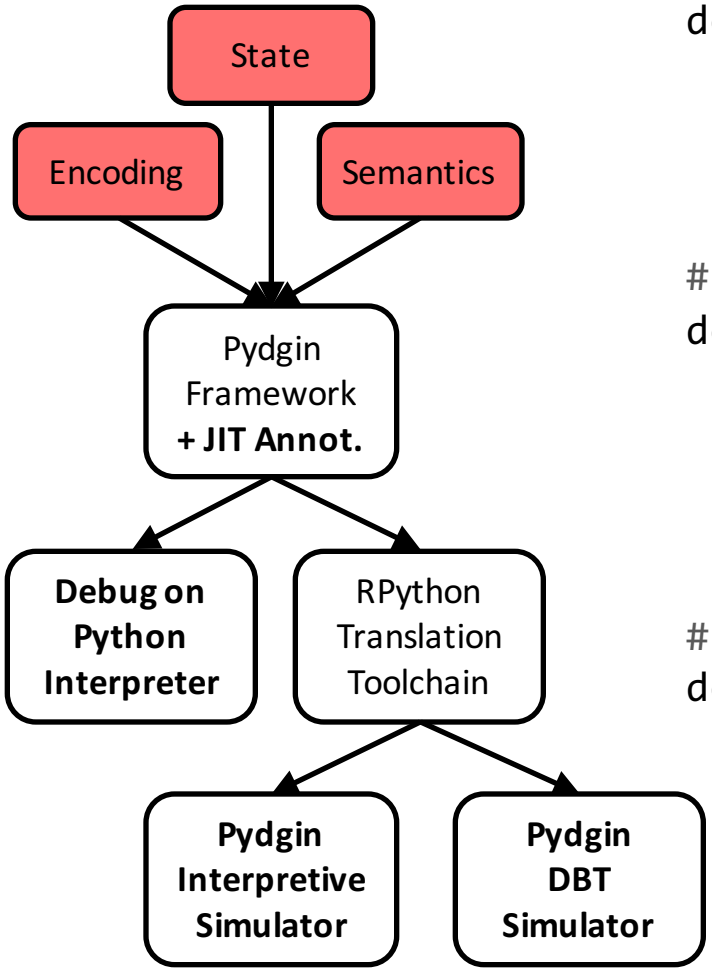

```
# count number of adds
def execute_addi( s, inst ):
  s.rf[inst.rd] = s.rf[inst.rs1] + inst.i_imm

  s.pc += 4

# count misaligned stores
def execute_sw( s, inst ):
  addr = trim_xlen( s.rf[inst.rs1] + inst.s_imm )

  s.mem.write( addr, 4, trim_32( s.rf[inst.rs2] ) )
  s.pc += 4

# record and count all executed loops
def execute_beq( s, inst ):
  if s.rf[inst.rs1] == s.rf[inst.rs2]:

    s.pc = trim_xlen( s.pc + inst.sb_imm )

  else:
    s.pc += 4
```

Pydgin Instrumentation



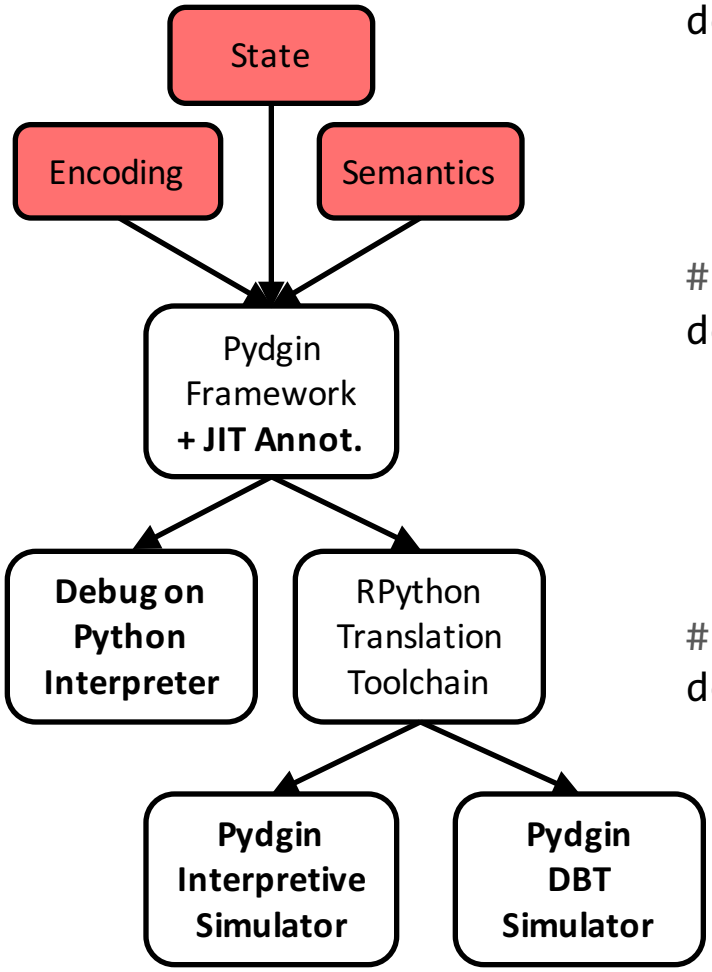

```
# count number of adds
def execute_addi( s, inst ):
  s.rf[inst.rd] = s.rf[inst.rs1] + inst.i_imm
  s.num_adds += 1
  s.pc += 4

# count misaligned stores
def execute_sw( s, inst ):
  addr = trim_xlen( s.rf[inst.rs1] + inst.s_imm )

  s.mem.write( addr, 4, trim_32( s.rf[inst.rs2] ) )
  s.pc += 4

# record and count all executed loops
def execute_beq( s, inst ):
  if s.rf[inst.rs1] == s.rf[inst.rs2]:

    s.pc = trim_xlen( s.pc + inst.sb_imm )

  else:
    s.pc += 4
```

Pydgin Instrumentation




```
# count number of adds
def execute_addi( s, inst ):
  s.rf[inst.rd] = s.rf[inst.rs1] + inst.i_imm
  s.num_adds += 1
  s.pc += 4

# count misaligned stores
def execute_sw( s, inst ):
  addr = trim_xlen( s.rf[inst.rs1] + inst.s_imm )
  if addr % 4 != 0: s.num_misaligned += 1
  s.mem.write( addr, 4, trim_32( s.rf[inst.rs2] ) )
  s.pc += 4

# record and count all executed loops
def execute_beq( s, inst ):
  if s.rf[inst.rs1] == s.rf[inst.rs2]:

    s.pc = trim_xlen( s.pc + inst.sb_imm )

  else:
    s.pc += 4
```

Pydgin Instrumentation




```
# count number of adds
def execute_addi( s, inst ):
  s.rf[inst.rd] = s.rf[inst.rs1] + inst.i_imm
  s.num_adds += 1
  s.pc += 4

# count misaligned stores
def execute_sw( s, inst ):
  addr = trim_xlen( s.rf[inst.rs1] + inst.s_imm )
  if addr % 4 != 0: s.num_misaligned += 1
  s.mem.write( addr, 4, trim_32( s.rf[inst.rs2] ) )
  s.pc += 4

# record and count all executed loops
def execute_beq( s, inst ):
  if s.rf[inst.rs1] == s.rf[inst.rs2]:
    old_pc = s.pc
    s.pc = trim_xlen( s.pc + inst.sb_imm )
    if s.pc <= old_pc: s.loops[(s.pc, old_pc)] += 1
  else:
    s.pc += 4
```

Pydgin in Our Research Group

- Statistics for software-defined regions

Pydgin in Our Research Group

- Statistics for software-defined regions
- Data-structure specialization experimentation

Pydgin in Our Research Group

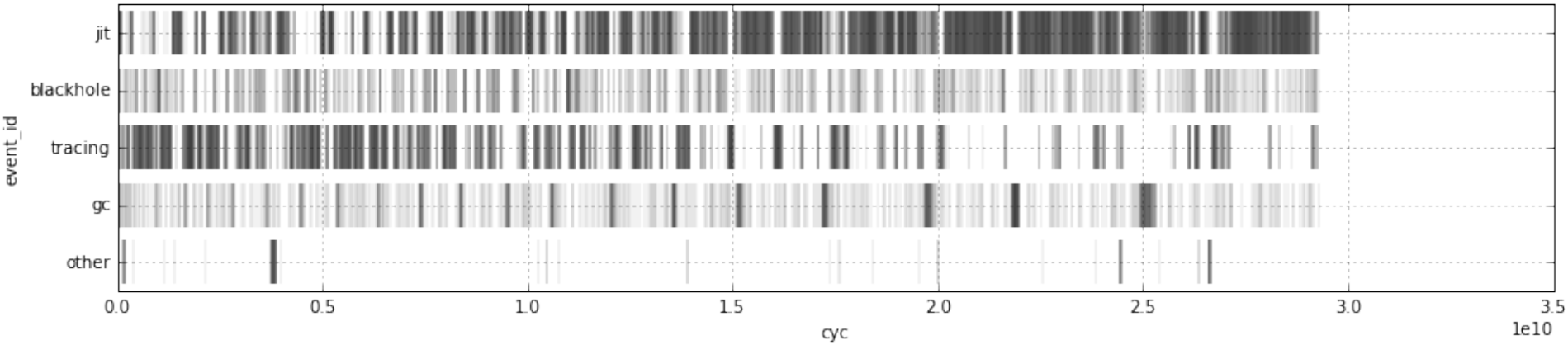
- Statistics for software-defined regions
- Data-structure specialization experimentation
- Control- and memory-divergence for SIMD

Pydgin in Our Research Group

- Statistics for software-defined regions
- Data-structure specialization experimentation
- Control- and memory-divergence for SIMD
- Basic Block Vector generation for SimPoint

Pydgin in Our Research Group

- Statistics for software-defined regions
- Data-structure specialization experimentation
- Control- and memory-divergence for SIMD
- Basic Block Vector generation for SimPoint
- Analysis of JIT-enabled dynamic language interpreters



Conclusions

Pydgin leverages the RPython translation toolchain into **high-performance**, DBT Instruction Set Simulator.

Conclusions

Pydgin leverages the RPython translation toolchain into **high-performance**, DBT Instruction Set Simulator.

Pydgin provides a succinct architecture description language within Python to give users a **productive development**, **extension**, and **instrumentation** experience.

Conclusions

Pydgin leverages the RPython translation toolchain into **high-performance**, DBT Instruction Set Simulator.

Pydgin provides a succinct architecture description language within Python to give users a **productive development**, **extension**, and **instrumentation** experience.

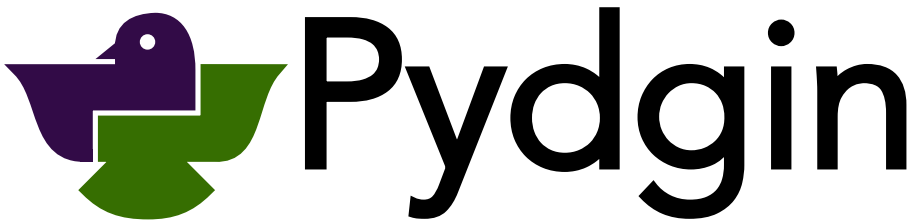
Current State: RV64IMAFD (RV64G) Bare-Metal on 64-bit host

Conclusions

Pydgin leverages the RPython translation toolchain into **high-performance**, DBT Instruction Set Simulator.

Pydgin provides a succinct architecture description language within Python to give users a **productive development**, **extension**, and **instrumentation** experience.

Current State: RV64IMAFD (RV64G) Bare-Metal on 64-bit host



https://github.com/cornell-brg/pydgin

Thank you to our sponsors for their support:
NSF, DARPA, and donations from Intel Corporation and Synopsys, Inc.