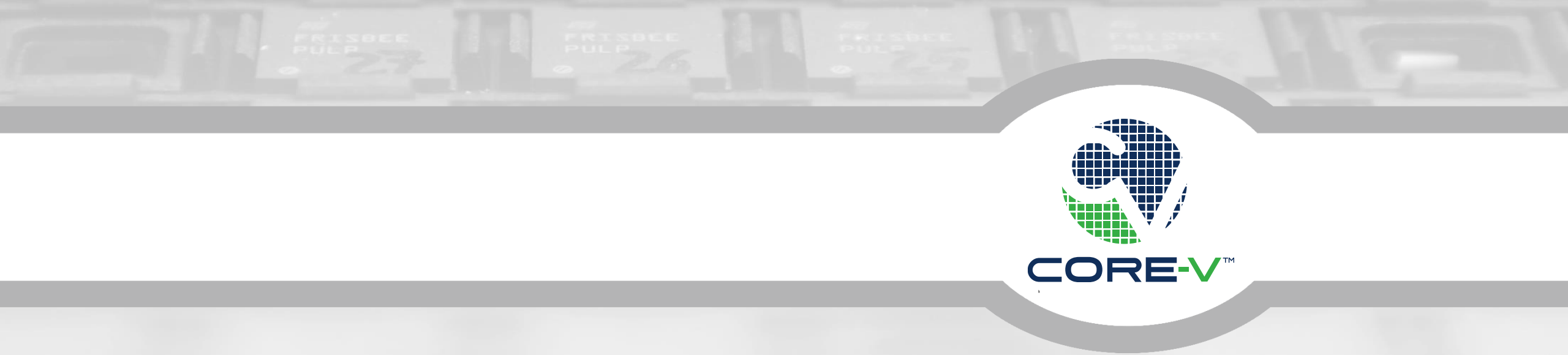


FORCE-RISCV ISG

Powerful CPU verification tool worth investing

Jingliang (Leo) Wang

jwang1@futurewei.com

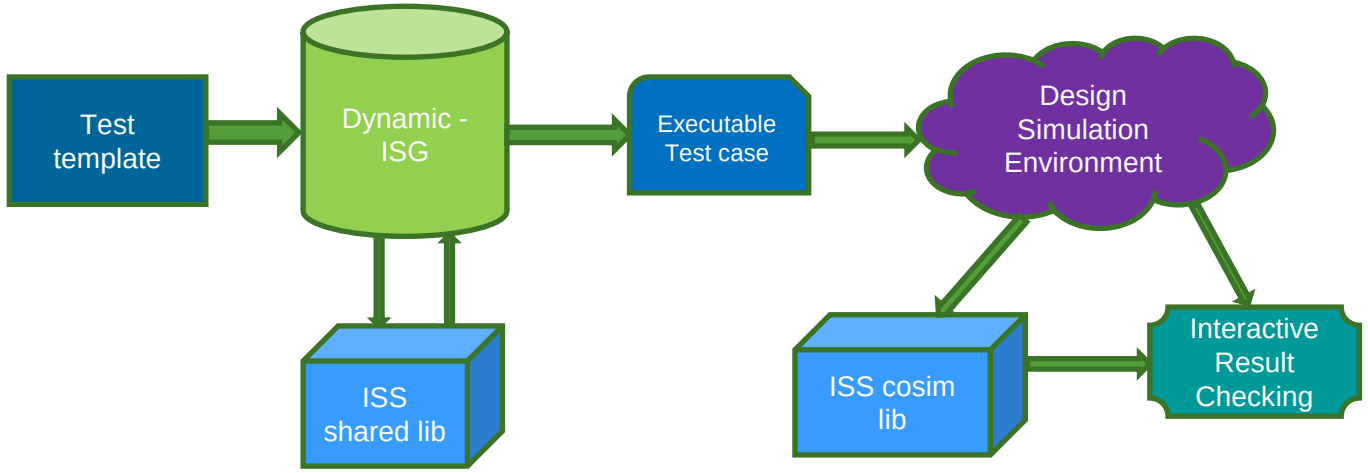
May 5, 2020

Introduction

The **FORCE-RISCV** Instruction Stream Generator (ISG):

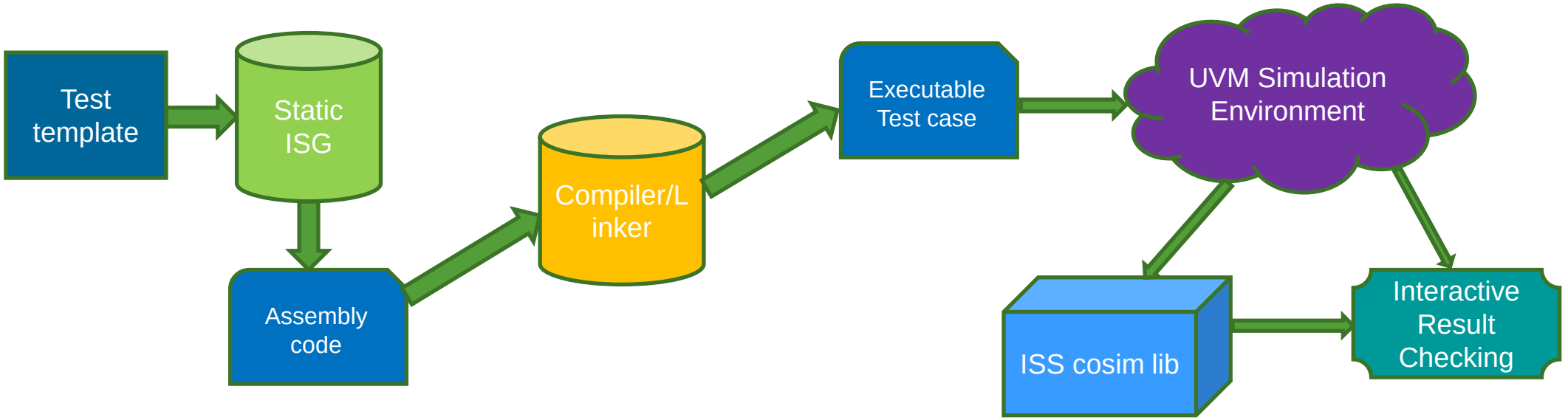
- Available at: https://github.com/openhwgroup/force-riscv
- Powerful verification tool for RISC-V based CPU design.
- Developed and released by **IC-Lab** of **Futurewei,** contributed to the **OpenHW Group**.
- Currently supports *64-bit version of RISC-V* with *RV64I, M, A, Zicsr, F, D, C* extensions, *Machine, Supervisor, User* modes, dynamic ISS interaction, dynamic Virtual Memory management, loops, configurable state transition, powerful and extensible Python scriptable test template writing framework etc.
- Features close to be done: *Vector extension 0.9* support, full paging exception control and advanced memory sub-system verification features.
- Collaborating with Hongkong City U on *32-bit version of RISC-V* support.

Dynamic ISG verification architecture



- Dynamic ISG generate machine code, for example, ELF file, that can be directly loaded and executed in the design simulation environment.
- Controls on test generation is extensive and instruction stream granule is fine-grained.
- Reaching good coverage is possible.
- Much harder to implement and provide long term value.

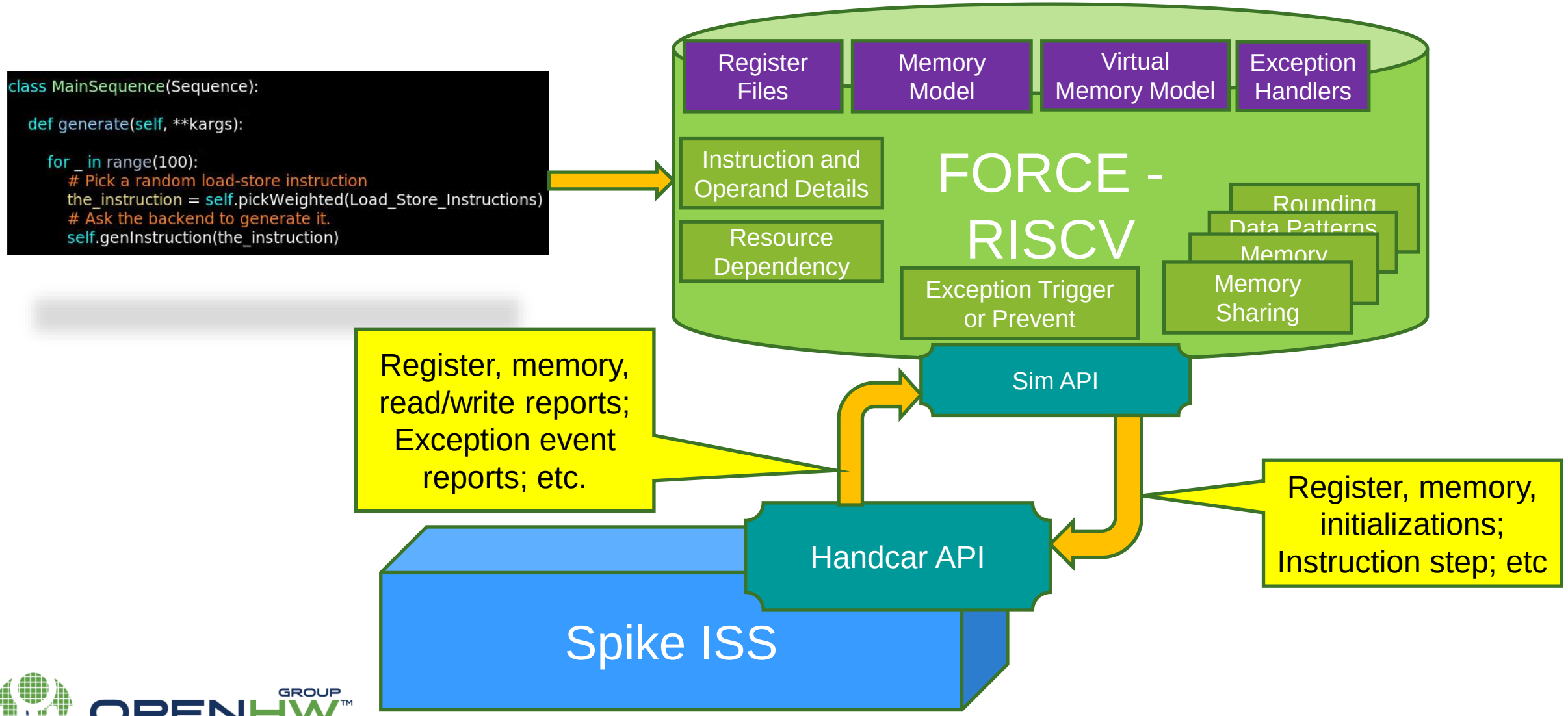
Static ISG verification architecture



- Static ISG generate assembly code file(s) that need to be compiled/linked to produce executable test case.
- Controls on test generation is limited and instruction stream granule is coarser.
- Reaching good coverage is not possible.
- Easier to implement and still good to have.

FORCE-RISCV Dynamic ISG

```
class MainSequence(Sequence):

   def generate(self, **kargs):

      for _ in range(100):
         # Pick a random load-store instruction
         the_instruction = self.pickWeighted(Load_Store_Instructions)
         # Ask the backend to generate it.
         self.genInstruction(the_instruction)
```

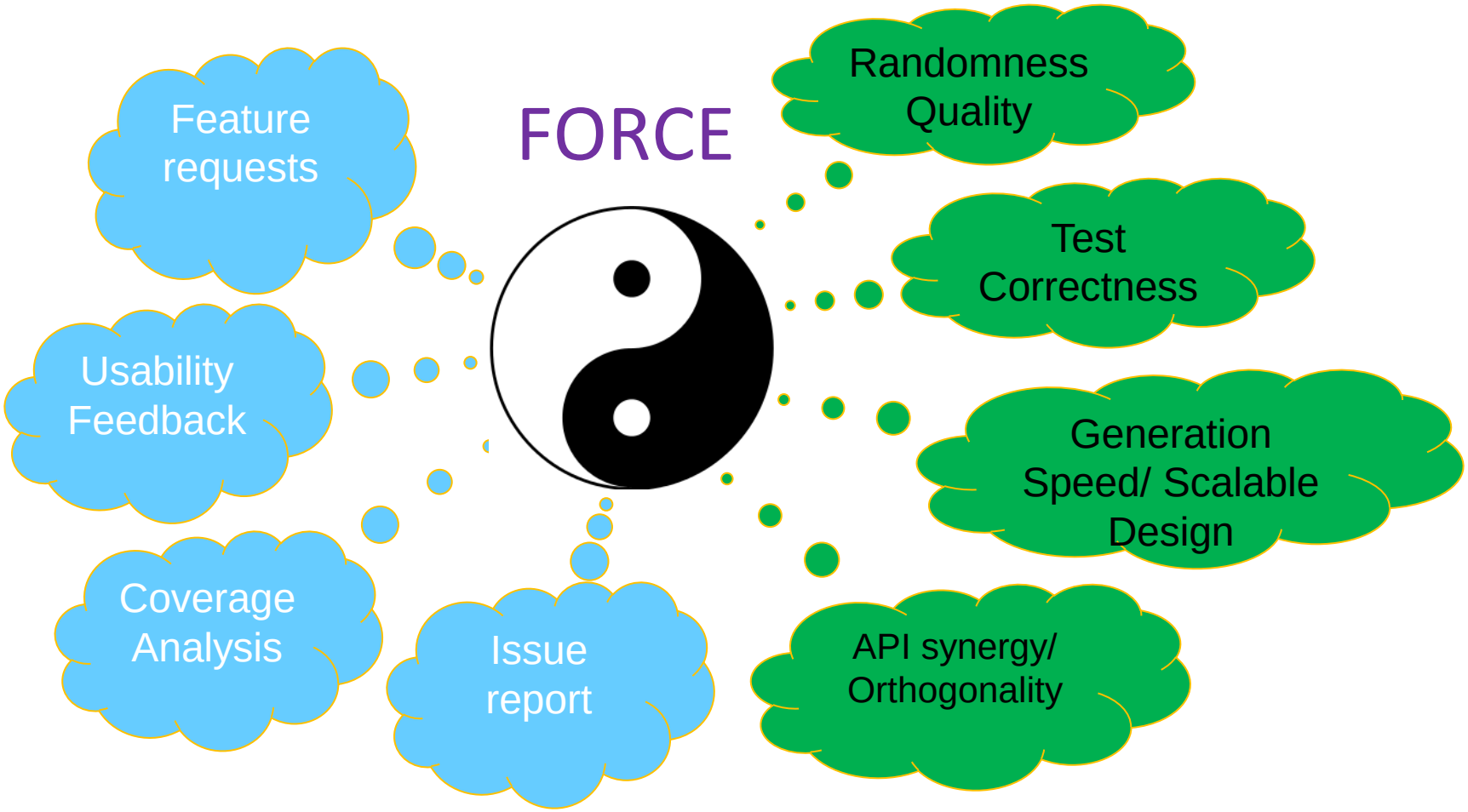


Why Python API?

- FORCE-RISCV ISG embedded a Python interpreter.
- Python is easy to learn, very popular (for good reasons).
- Great binding with C++, great synergy.
- Powerful Object-Oriented and functional programming language features itself, enables many ways for the user to further extend the capability of the ISG.
- Avoid downfalls of GUI based approach where there is a lot of upfront cost in implementing a GUI, rigid structure in the declarative test template without much control capabilities, and the power users skip the GUI anyway.

```
class MyMainSequence(Sequence):

  def generate(self, **kargs):
     choices_mod = ChoicesModifier(self.genThread)
     # Modify the choices settings.
     choices_mod.modifyOperandChoices("Rounding mode", {"RNE":0, "RTZ":0, "RDN":50, "RUP":0, "RMM":0, "DYN":50})
     # Adjust address reuse settings.
     choices_mod.modifyOperandChoices("Read after write address reuse", {"No reuse":50, "Reuse":50})
     # Adjust page size settings.
     choices_mod.modifyPagingChoices("Page size#4K granule#S#stage 1", {"4K":10, "2M":10, "1G":0, "512G":0})
     choices_mod.commitSet()

     for _ in range(100):
        # pick random RVC load/store instruction.
        instr = self.pickWeighted(RVC_load_store_instructions)
        # pick a random address aligned to 4K page boundary.
        target_addr = self.genVA(Align = 0x1000)
        self.notice(">>>>>  Instruction: {} Target addr: {:012x}".format(instr, target_addr))
        self.genInstruction(instr, {"LSTarget":target_addr})

        # Generate a mix of floating point and LDST instructions.
        instr_mix = { ALU_Float_All_map:10, LDST_All_map:10 }
        instr = self.pickWeighted(instr_mix)
        self.genInstruction(instr)

     # undo the choices settings - revert back to prior
     choices_mod.revert()
```

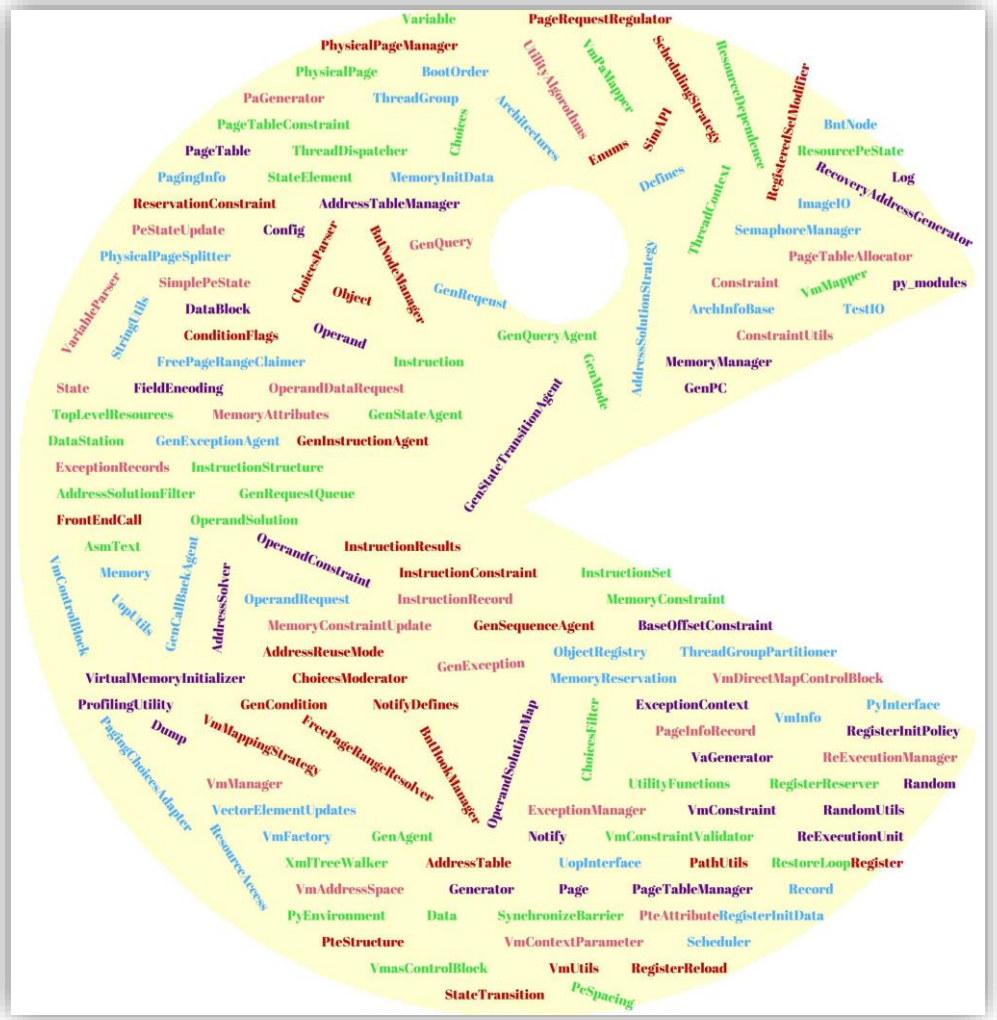
Design Philosophy



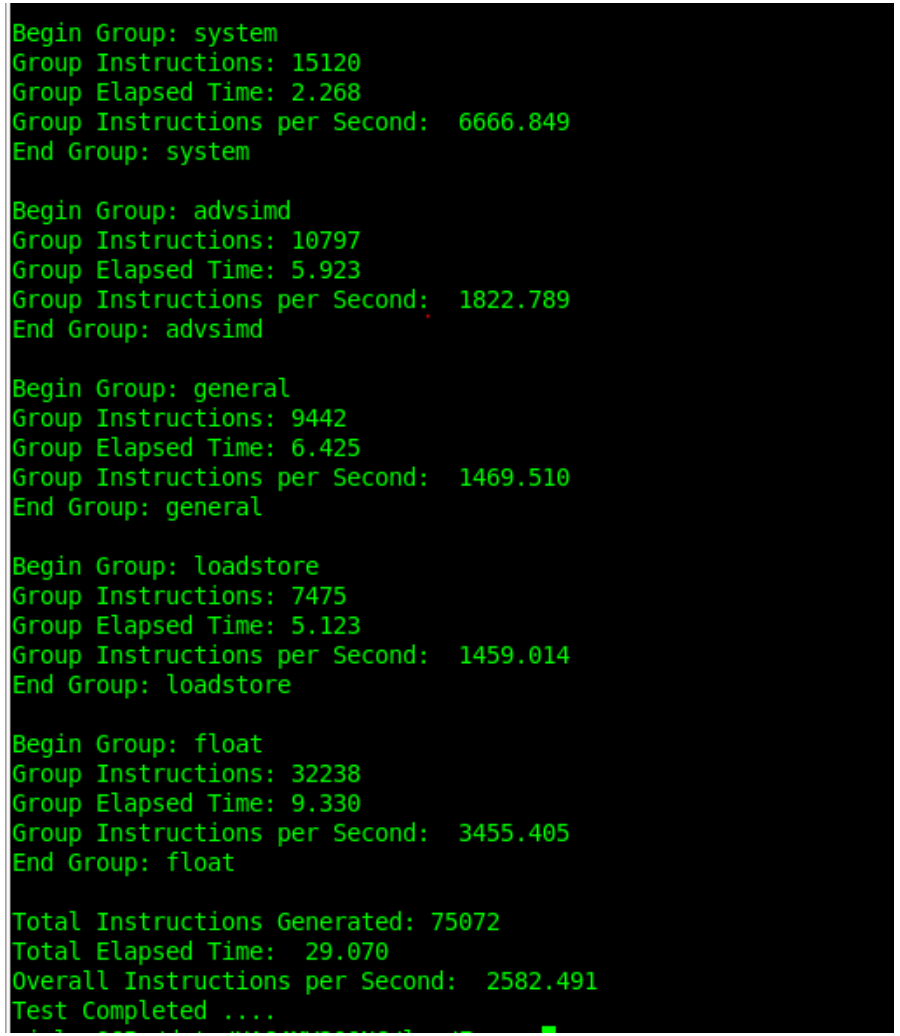
Awesome result is achieved via tireless collaboration from teams on different abstraction layers but work together the best they can to achieve maximum FORCE

Scalability by design

- FORCE backend is written in C++ 11, with many deliberately and pains-takingly designed / implemented / tested classes.
- Close to 75% of code is implemented in the architecture neutral base layer. This enables us to bring-up support for new ISAs (for example, RISC-V) much faster since we have built a solid foundation along the way.
- Effort is continuously made to accumulate architecture neutral functionalities in the base layer.



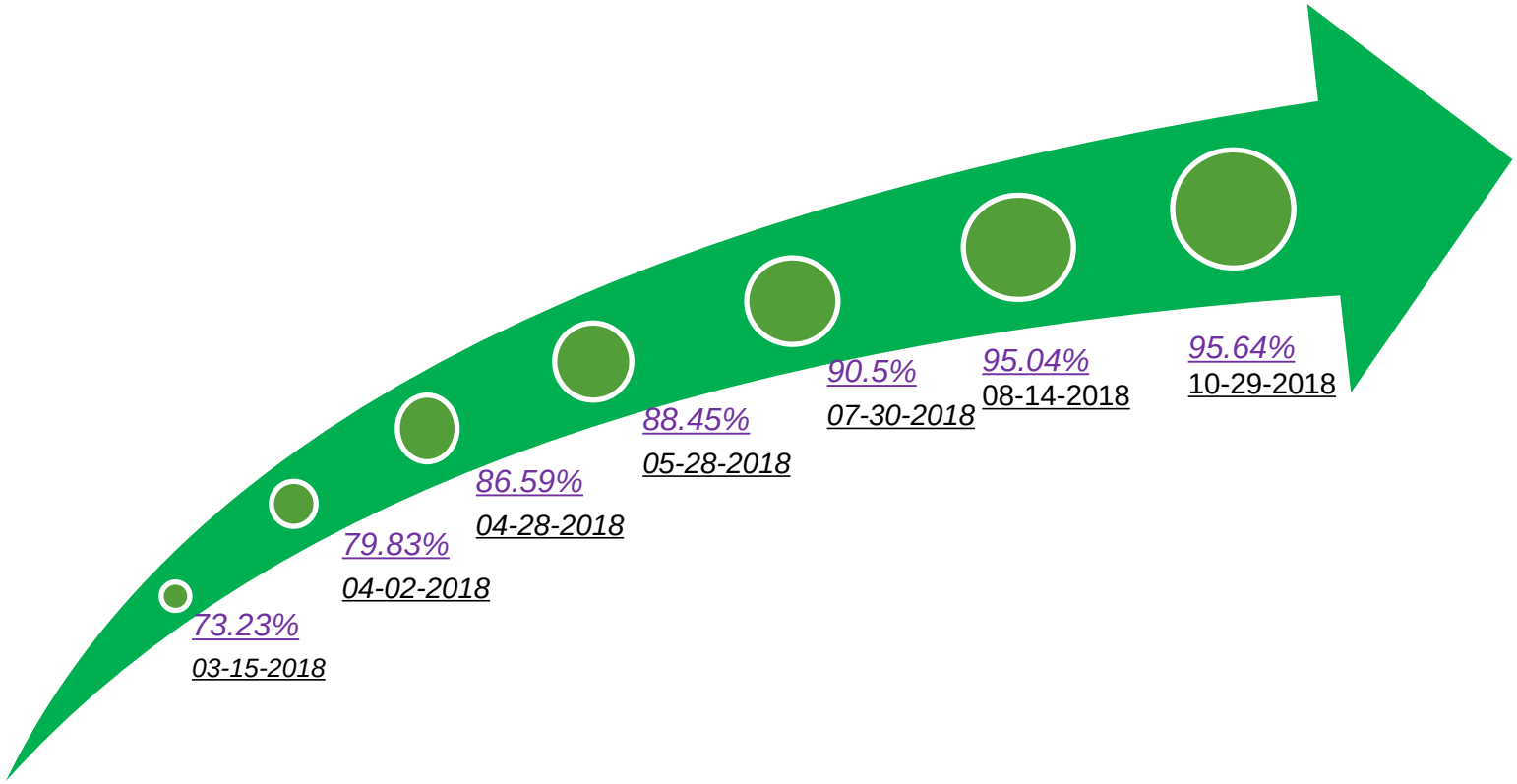
Test generation performance



Performance is super fast, with:

- Paging on
- ISS integrated
- dependency solving
- multi-priority address solving.

Example of Coverage Progress Measured During Development



Improving the coverage while keep everything else working is a highly delicate process. Requires very high skill level from both the ISG development side and the Verification user side.

MP/MT Support

MP/MT is a challenging topic for any CPU project. An entirely different set of problems need to be solved by Design, Verification, Tools. For ISGs available, options are limited regarding MP/MT:

- ISGs tend NOT to be designed with MP/MT in mind.
- ISGs tend to NOT be well suited for MP/MT verification due to scalability issues.
- It is NOT an easy problem.

FORCE-RISCV ISG is strong in MP/MT by design:

- The design philosophy is originated from heavy MP/MT server class CPU design project with more than a hundred harts, with proven results.
- Rich verification experience in MP/MT learned from previous projects.
- Well designed, transparent threading view make it very easy to design tests with co-operations among threads.
- All algorithms/data structures carefully designed to never sacrifice randomness while maintain very good performance. FORCE is blazingly fast comparing to other ISGs, thus can scale very well to large thread counts.

Resource helping new user and developer

- The user manual describes the supported APIs: https://github.com/openhwgroup/force-riscv/blob/master/doc/FORCE-RISCV_User_Manual-v0.8.pdf
- Examples: https://github.com/openhwgroup/force-riscv/tree/master/examples/riscv
- 32-bit version RISC-V support roadmap (estimated about 3 engineering month work): https://github.com/openhwgroup/force-riscv/blob/master/doc/FORCE-RISCV_32BIT_Roadmap.pdf
- Guide on how to add new instructions/registers in the work, will be available very soon.

Thank You!

GROUP OPENHW™ PROVEN PROCESSOR IP