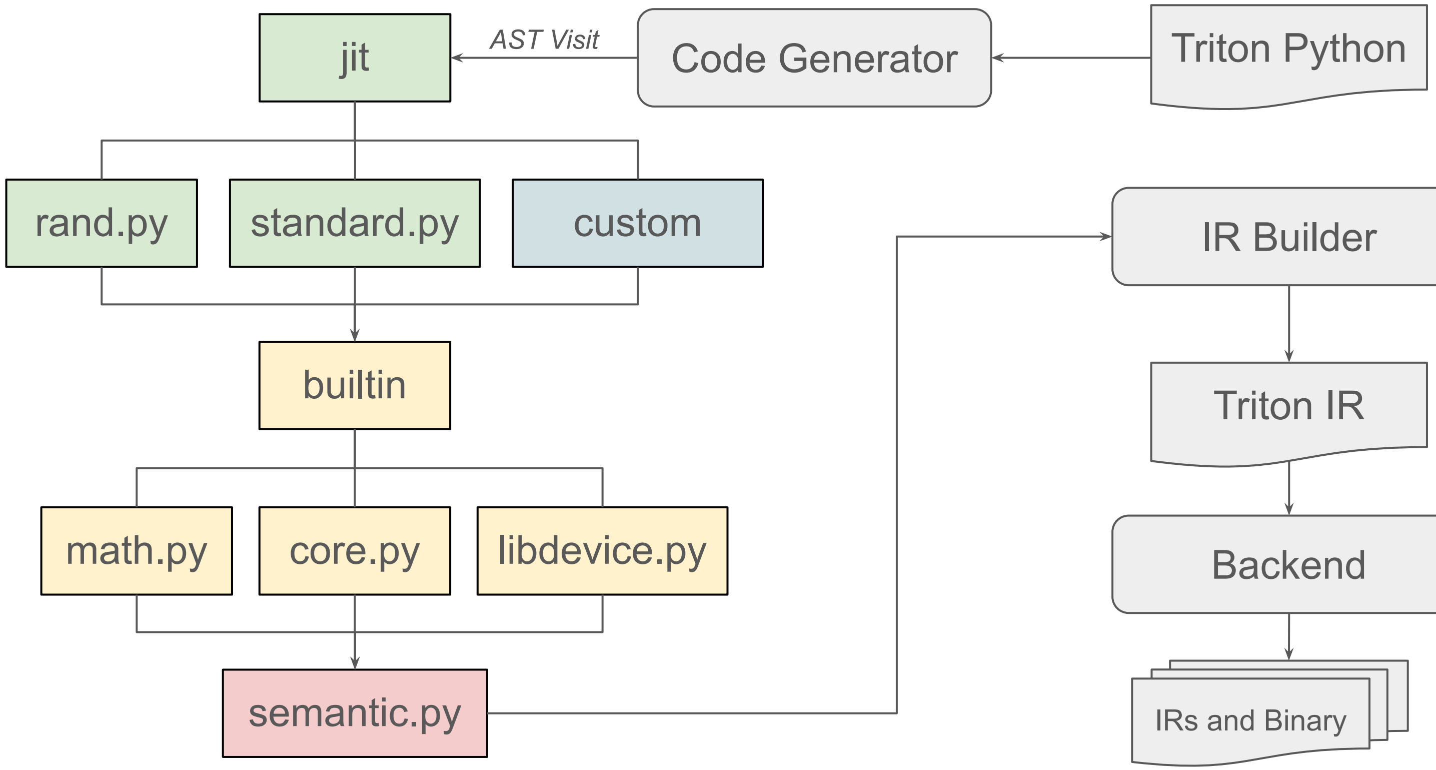
Triton Interpreter Update

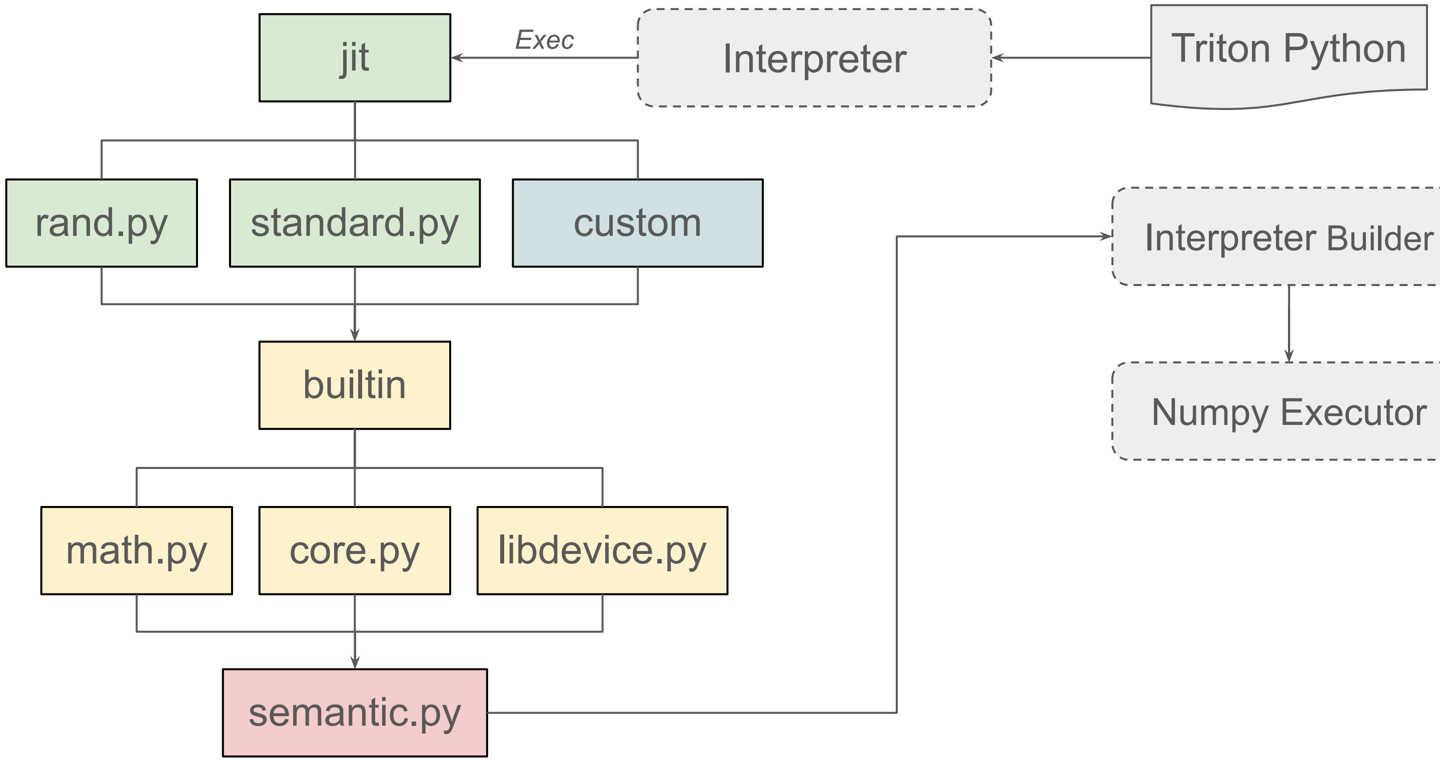
Goal

- Debug Triton code by applying print or attaching a debugger to step through the execution of individual Triton programs
  - Mostly designed for frontend users
  - Also help compiler developers get the expected output without writing a corresponding torch/pallas program
- Related files
  - *python/triton/runtime/interpreter.py*
  - *python/src/interpreter.cc*
  - *python/test/unit/language/*.py*
    - `@pytest.mark.interpreter`

Revisit the Frontend



Interpreter



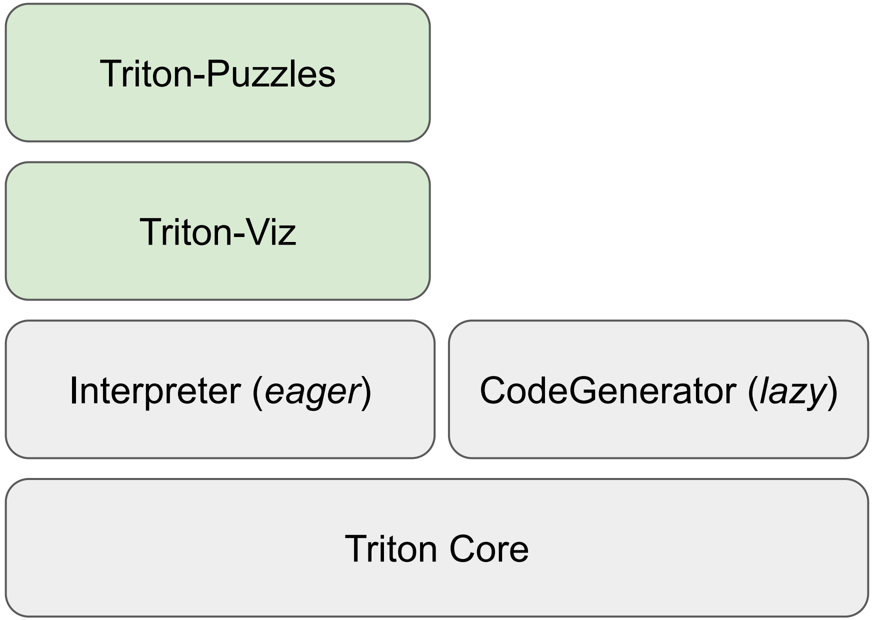
Exceptions

- `tl.reduce` and `tl.scan` are not lowered through the interpreter builder
    - `make_combine_region` invokes the code generator
    - We directly replace `tl.reduce` and `tl.scan` with custom implementations
        - Native ops like `np.sum` are accelerated through numpy
        - Custom `combine_fn` might be slow
- Functions and classes that do not go through the IR builder
    - `range`, `static_range`, `static_assert`, `static_print`
    - We replace them with python implementations

Usage

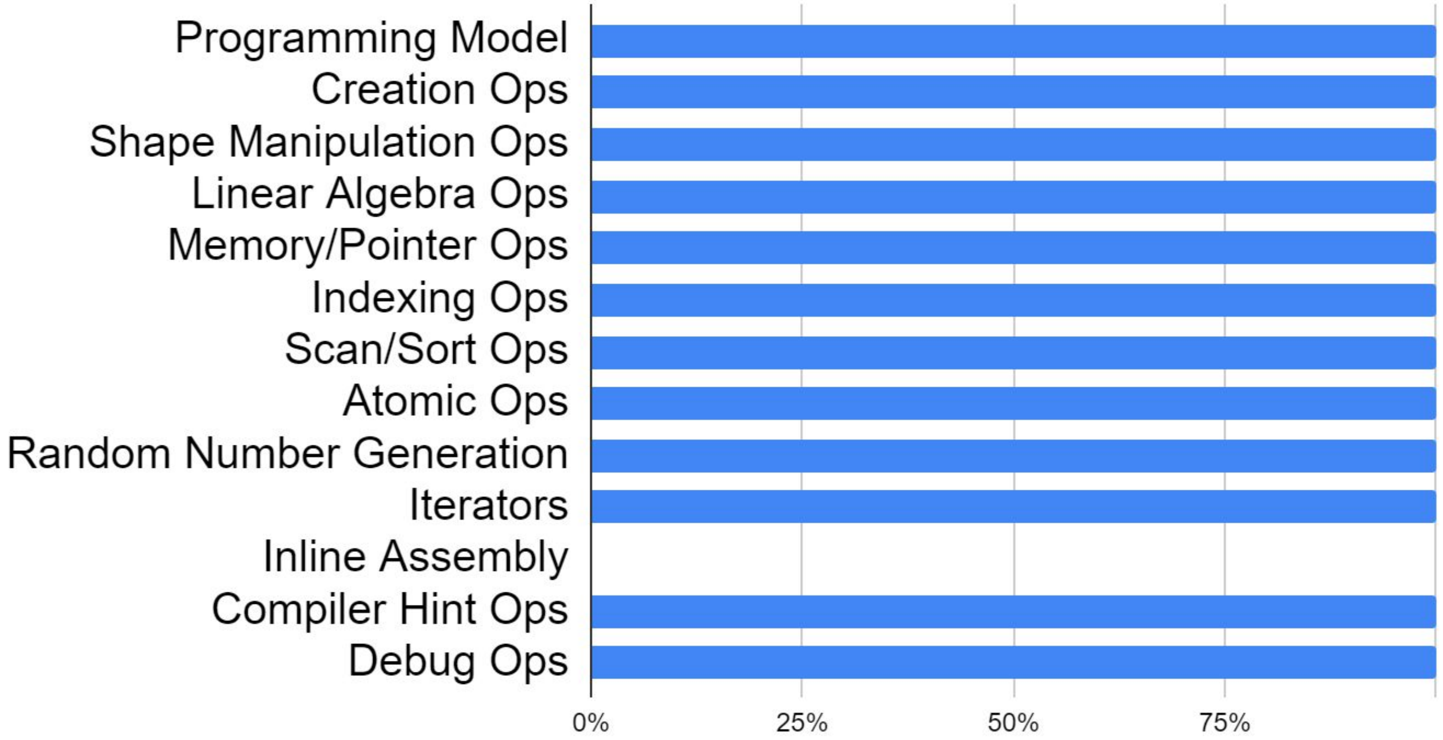
- Enable the interpreter mode
  - `TRITON_INTERPRET=1 <your command>`
- Debug with `pdb`
  - `TRITON_INTERPRET=1 pdb test.py`
  - `b test.py:<line number>`
  - `r`
- Highlights
  - You can set `device='cpu'` to execute code with the interpreter
  - You can print `tl.tensor` using the native python print and check all values of the tensor

Ecosystem



srush/Triton-Puzzles: Puzzles for learning Triton (github.com)
Deep-Learning-Profiling-Tools/triton-viz (github.com)

Coverage


Programming Model
Creation Ops
Shape Manipulation Ops
Linear Algebra Ops
Memory/Pointer Ops
Indexing Ops
Scan/Sort Ops
Atomic Ops
Random Number Generation
Iterators
Inline Assembly
Compiler Hint Ops
Debug Ops
0%
25%
50%
75%

Known Limitations

- No implicit **scalar** to **tensor** conversion
  - The following statements are not supported
    - `a=3`
    - `print(a.dtype) # runtime error`
    - `tl.full # workaround1`
    - `tl.to_tensor # workaround2`
- Indirect memory access is not supported
  - `ptr = tl.load(a)`
  - `a = tl.load(ptr)`
- Some precisions are not supported
  - `bfloat16`
  - `float8 series`

Known Limitations

- Do not support selective interpretation
    - Only interpret all kernels
    - *Triton-Viz* doesn't have this limitation
        - Use `importlib.reload(tl)`
- Each program is executed in a fixed order
    - program id0 -> program id1 -> program id2 -> …
- Overhead might be high
    - Especially for `tl.reduce/tl.scan` with custom associative operators
    - *Triton-Viz* mitigates the problem
        - Can sample programs

Action Items

- Documentation
- Overhead reduction
- Selective kernel interpretation?
- TorchInductor debugging?
- Case studies
    - Correctness
    - Performance estimate
- Anything else?