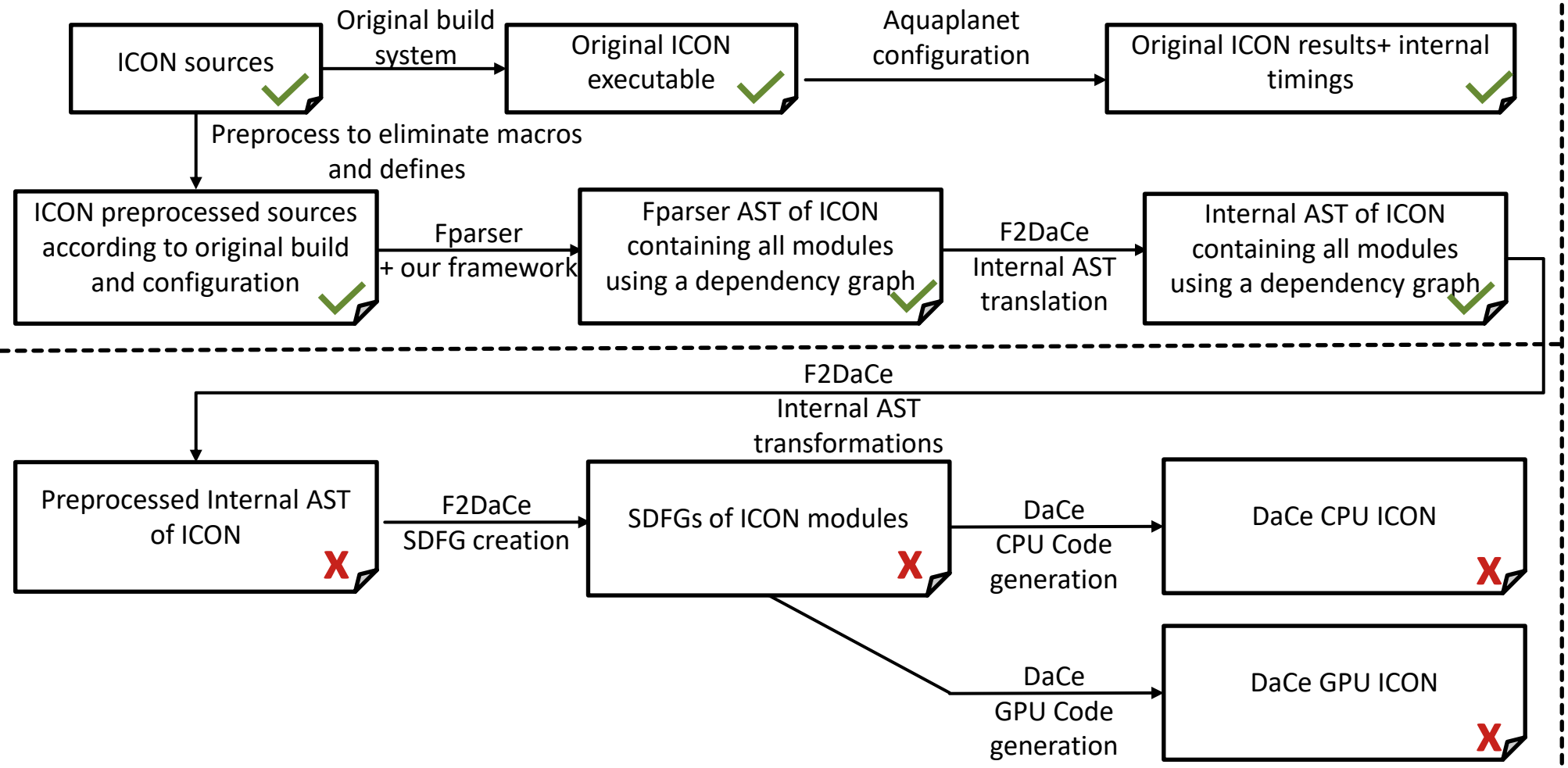
F2DaCe + ICON overall system and overall progress



Additional aspects:

Framework for Value/Timing instrumentation for original ICON

Framework for Value checking/Timing for individual module SDFGs

Together, we need an automatic testing framework to quickly discover bugs

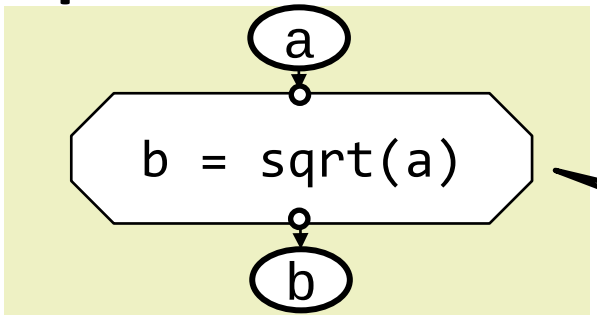
Normalization pass is ready!

- **https://github.com/spcl/dace/pull/1367**

```
def test_fortran_frontend_array_access():
    """
    Tests that the Fortran frontend can parse array accesses and that the accessed indices are correct.
    """
    test_string = """
                    PROGRAM access_test
                    implicit none
                    double precision d(4)
                    CALL array_access_test_function(d)
                    end

                    SUBROUTINE array_access_test_function(d)
                    double precision d(4)

                    d(2)=5.5

                    END SUBROUTINE array_access_test_function
                    """
    sdfg = fortran_parser.create_sdfg_from_string(test_string, "array_access_test")
    sdfg.simplify(verbose=True)
    a = np.full([4], 42, order="F", dtype=np.float64)
    sdfg(d=a)
    assert (a[0] == 42)
    assert (a[1] == 5.5)
    assert (a[2] == 42)
```

Intrinsic Functions – types by example

1. **SQRT, SIN, COSH**



Python tasklet, therefore analyzable!

2. **SUM, ANY, ALL**

   Eliminated via AST transformation.

Some interesting tradeoff questions: when to codegen using a break?

3. **SELECTED_INT_KIND –** direct evaluation during AST processing
4. **PRESENT, OPTIONAL –** (hopefully) direct evaluation during AST processing
5. **MATMUL –** rewrite/lift as einsums and rely on DaCe optimization

Intrinsics and work division.

- **https://github.com/spcl/dace/issues/1368**

```
class SumToLoop(NodeTransformer):
    """
    Transforms the AST by removing array sums and replacing them with loops
    """
    def __init__(self):
        self.count = 0

    def visit_Execution_Part_Node(self, node: ast_internal_classes.Execution_Part_Node):
        newbody = []
        for child in node.execution:
            lister = SumLoopNodeLister()
```

…

Namespaces…

from module X import a=>a_1

integer a

Rename of imported object- can be function, type, symbol or data container.

- **Proposed solution:**
  - Global rename: if renaming(s) exist for a variable **var_name** to **new_var_name$_{1..N}$** rename both **var_name** and all **new_var_name$_{1..N}$** to **__dace_<module_of_var_name>_var_name** everywhere
  - This should eliminate shadowing issues caused by renamings while still ensuring global uniqueness at the granularity of modules
  - Not a full solution -> DaCe might still have issues with shadowing when inlining SDFGs or lose optimization opportunities because of it. (Need to investigate)