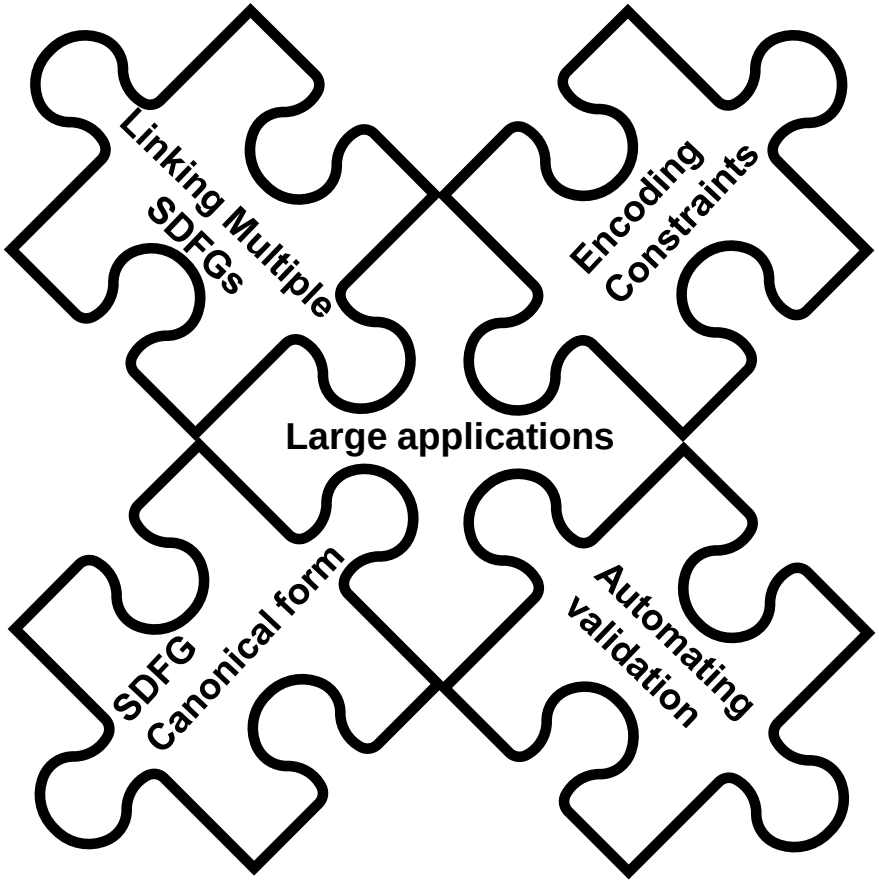
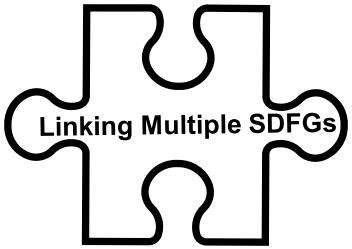
DaFlEx
Advances in DaCe towards handling larger code bases

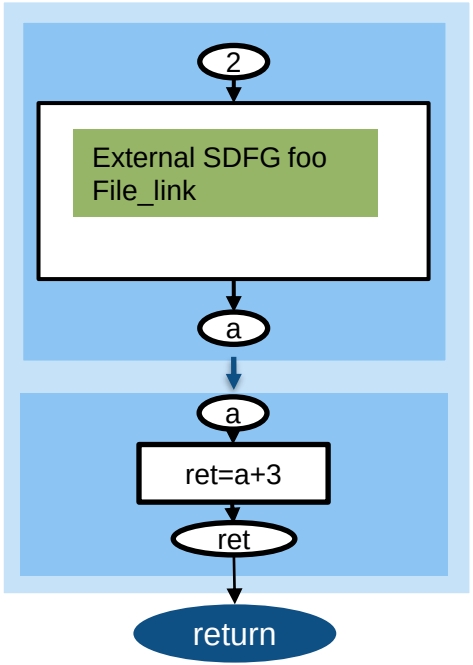

Linking Multiple SDFGs
Encoding Constraints
Large applications
SDFG Canonical form
Automating validation



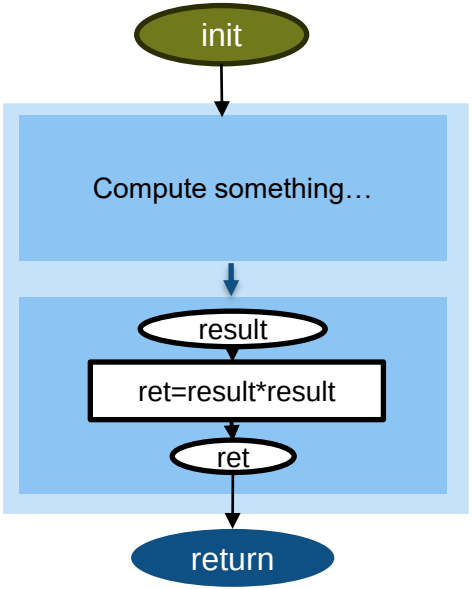

```
int foo(int init){
    … compute something…
    return result*result;
}

int main(){
   int a,b;
   a=foo(2);
   return a+3;
}
```

Main:

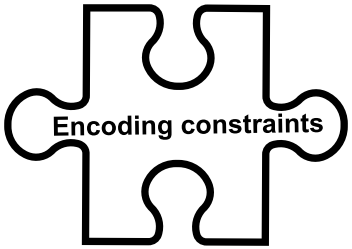


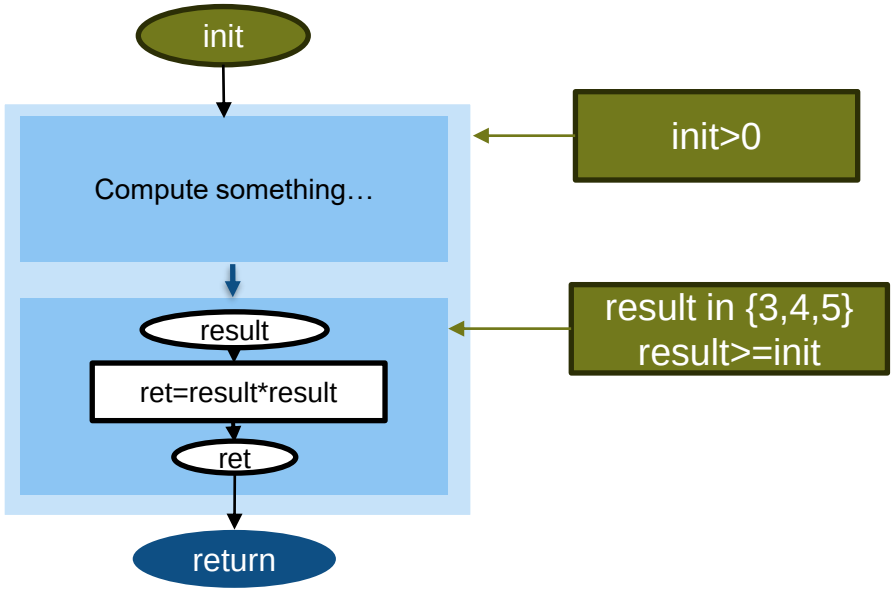
Foo:



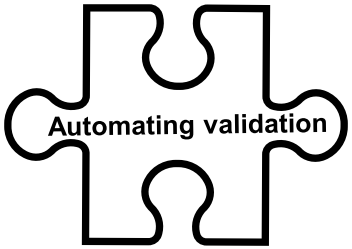
- This source code would generate the SDFGs on the right.
- The “empty” SDFG provides a method and path to load the SDFG.

- Foo could be compiled directly.
- Main only after the external SDFG is loaded.
- Main can still be simplified in this form.

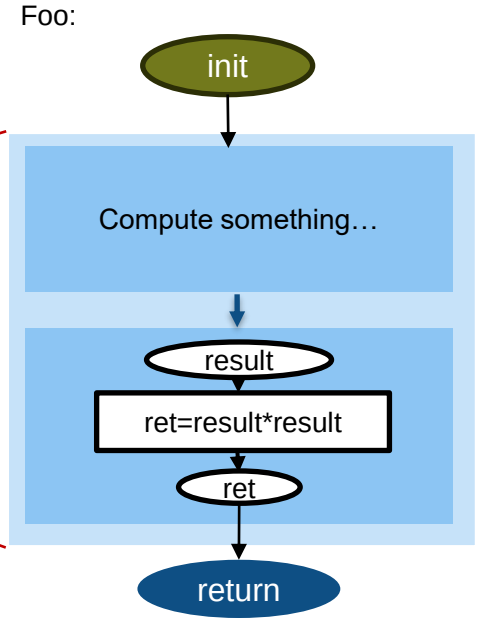




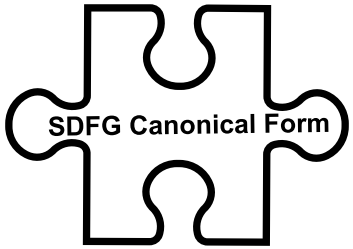
- Sometimes, users are aware of constraints and requirements for symbols and arrays.
- We want to have the options of encoding, inheriting and appending them to different scopes within the SDFGs (SDFGs, States, Nodes)




```
int foo(int init){
    … compute something…
    return result*result;
}
```



- Debug information links SDFG to original source code
- Use comparative code execution and fuzzing to validate at function call granularity



Fundamental question: What is the ideal representation of a given algorithm in the SDFG IR?

Ideal Dataflow Form
- Minimal Depth
- Maximal Fission
- No Language Artifacts (Induced Control Flow)
- Requirements model

?

Device specialization
- Map Fusion
- Memory management
- Tiling

Hardware characteristics

Accelerator offloading
- Node partitioning
- Data movement