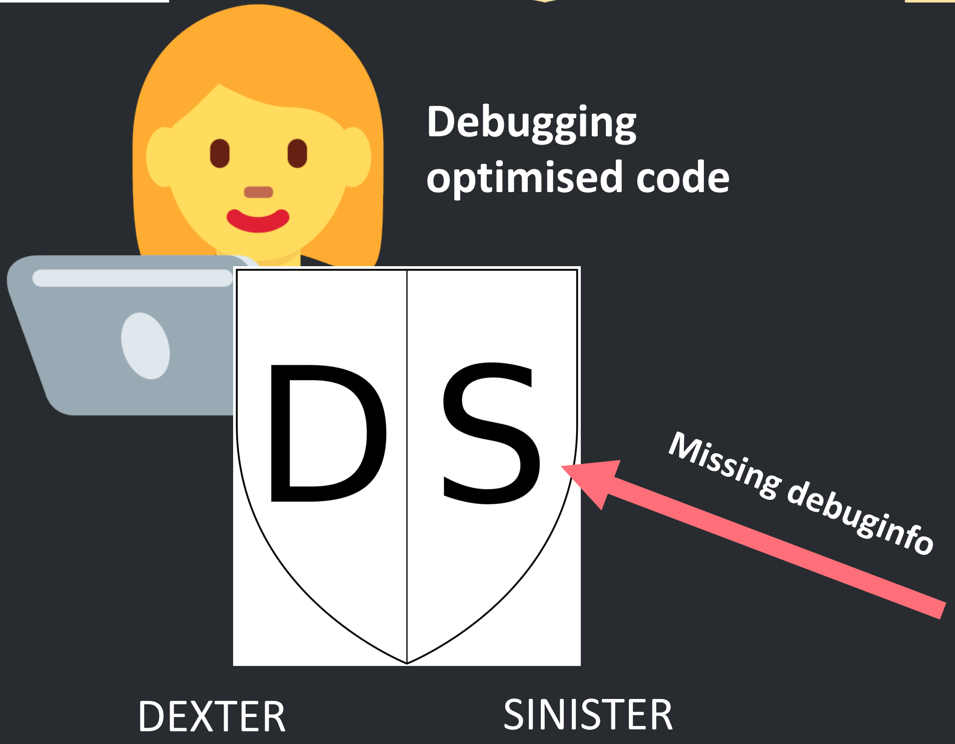
TOOLS FOR CHECKING AND WRITING DWARF PROGRAMS

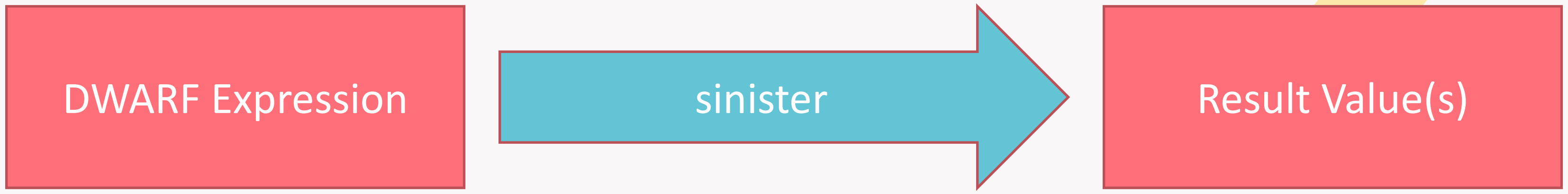


GRAPHCORE

Graphcore LLVM Dev Meeting 2022

THE TOOLS

- A set of tools loosely inspired by **Dexter**, the Debugging Experience Tester.
  - *Tests user-specified debugging expectations are met*
- **sinister**
  - A DWARF expression interpreter. Interprets DWARF-conformant expressions and computes a result.
- Means it is not necessary to launch a debugger to test DWARF expressions. Utilises the comprehensive DWARF stack machine interpreter defined in lldb.



MOTIVATION

Currently, to verify the correctness of a DWARF program its necessary to build an executable and launch a debugger.

- Limits the verification of the DWARF program to the inputs set in the executable
- Heavyweight process just to run some stack machine instructions

So build tools to enable interactive DWARF testing and writing.

- Provide a lightweight DWARF expression checking environment.
- A lightweight REPL for DIExpression and DWARF expressions.
- It’s good fun.

DWARF EXPRESSION INTERPRETER

```
# RUN: sinister %s | FileCheck %s

# CHECK: Result: 42

DW_OP_const1u(38) DW_OP_const1u(4) DW_OP_plus DW_OP_stack_value
```

- Currently limited to constants – register context as an input is the current priority
- Also the addition of flags to print the result in different ways e.g. fragments / DW_OP_piece

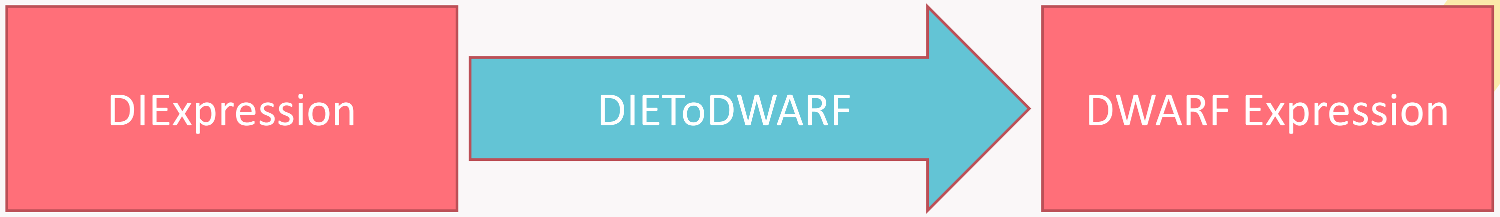
LLVM DWARF EXTENSIONS

- The *dbg.value* intrinsic uses a superset of DWARF expressions:
  - **DIArglist** (DIA) – A list of SSA values
  - **DIExpression** (DIE) – An expression consisting of ‘Extended DWARF’ with LLVM specific operators e.g. *DW_OP_LLVM_arg*

- So development is ongoing of the utility ‘llvm-dietodwarf’
  - Inputs are the DIArgList and DIExpression
  - Outputs a ‘pure’ DWARF program
  - This can be input to sinister (the interpreter) or other utilities

THE TOOLS

- **DIEToDwarf**
  - DIExpression string to DWARF expression string convertor
  - Means that sinister doesn’t have to support multiple input formats.



THE TOOLS – PROGRESS

- Critically, haven’t implemented setting register context. This means currently interpretation is limited to DWARF expressions with constants.

- Figuring out how to do this is the current focus, it’ll make the utilities much more useful.

- Open to suggestions about how best to do this. ATM have attempted launching an lldb instance an trying to take a context from this. But its not ideal – part of the goal is to not need a debugger instance.

SUMMARY

- On track for an initial release in December, but progress is currently on github

  https://github.com/chrisjbris/llvm-debugy [Work In Progress!]

More fun tools

- Write a simple optimisation pipeline for DWARF expressions (DWARF->IR->DWARF)?
- Expressions can contain effective no-ops or trivially combinable instructions
- Optimisations are scattered e.g. *isIdentityFunction()* in LSR and *constantFold()* in DIExpression
- Why not have all optimisations in one place?