xDSL

Developing Building Blocks for HPC in MLIR

Anton Lydike, The University of Edinburgh

ExCALIBUR

epcc

Imperial College London

THE UNIVERSITY of EDINBURGH informatics

Previously in Compilers

CRAY

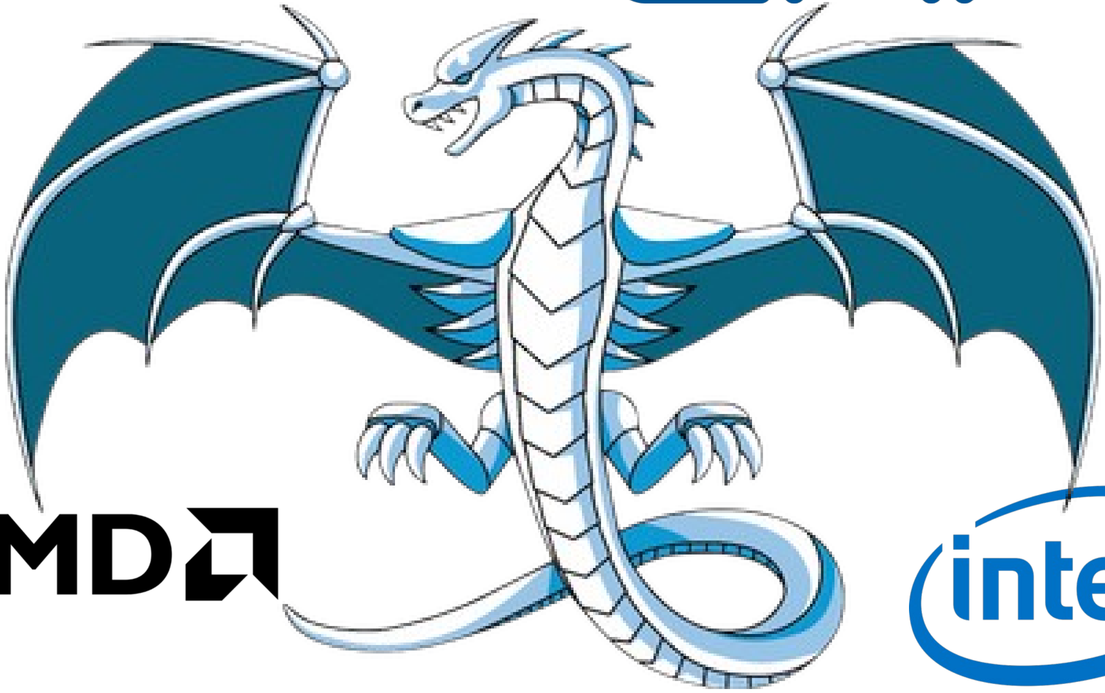
arm
COMPILER

AMD

intel®

PGI® COMPILERS & TOOLS

Previously in Compilers



CRAY

arm COMPILER

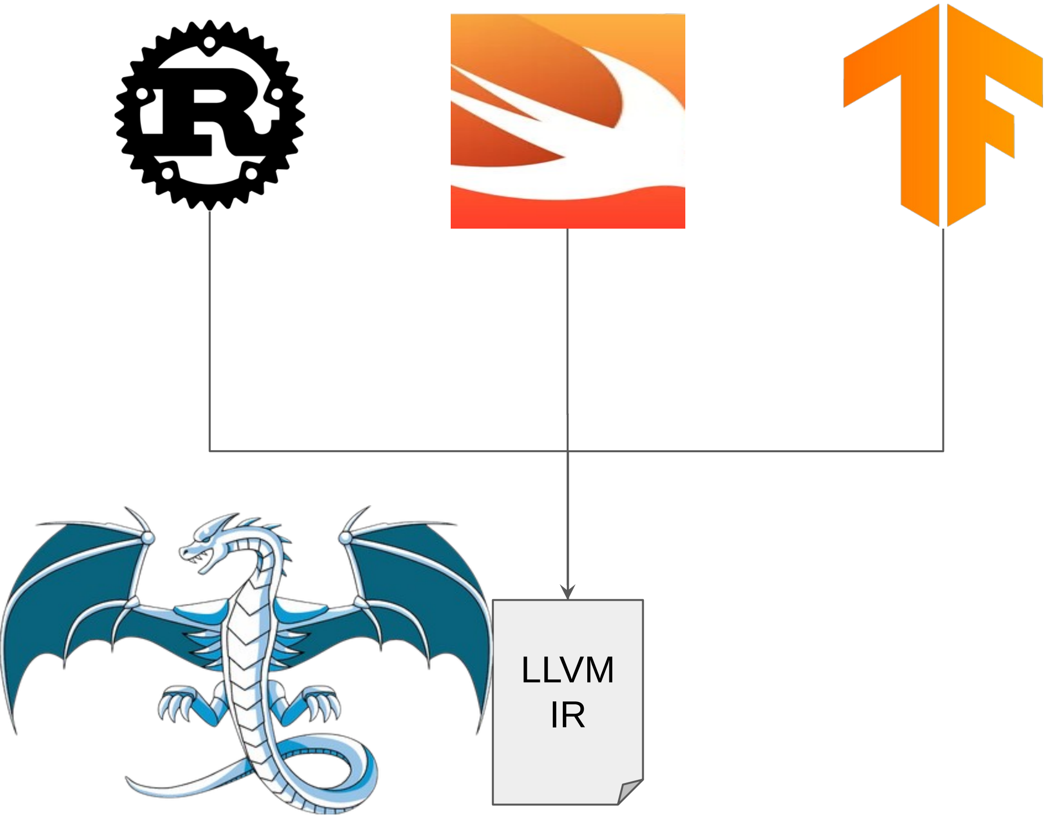
AMD

intel

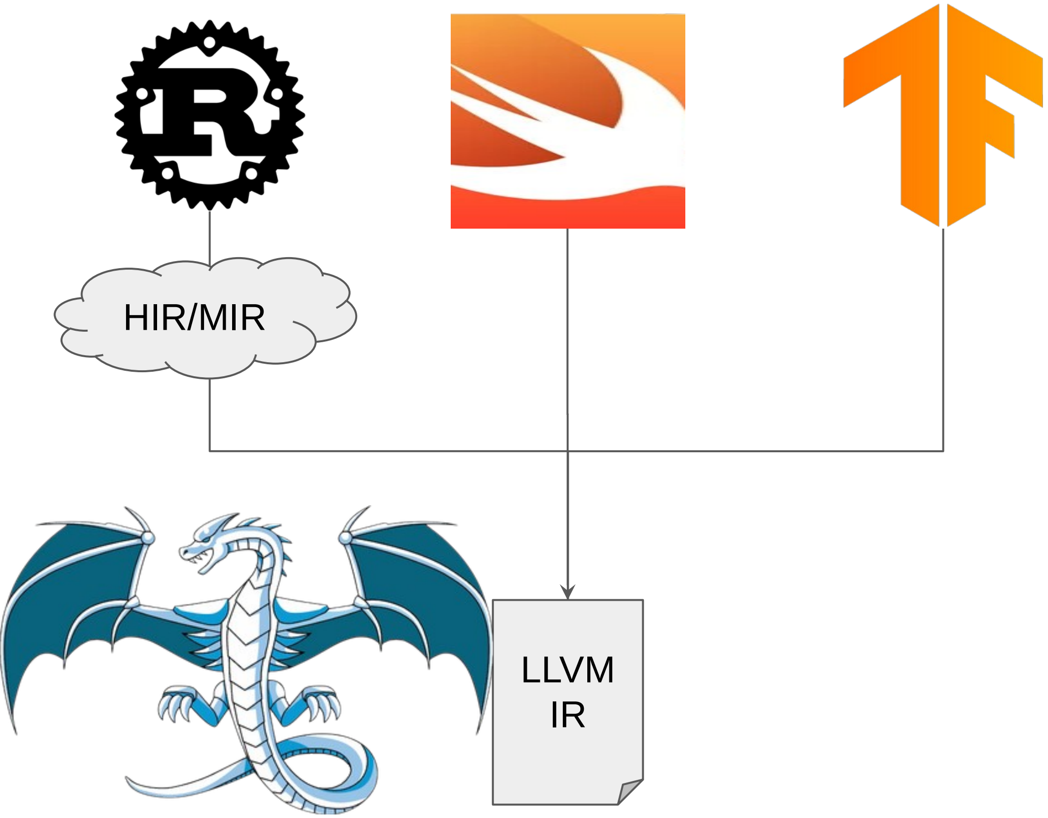
PGI COMPILERS & TOOLS

There be Dragons!
Where's the Treasure?

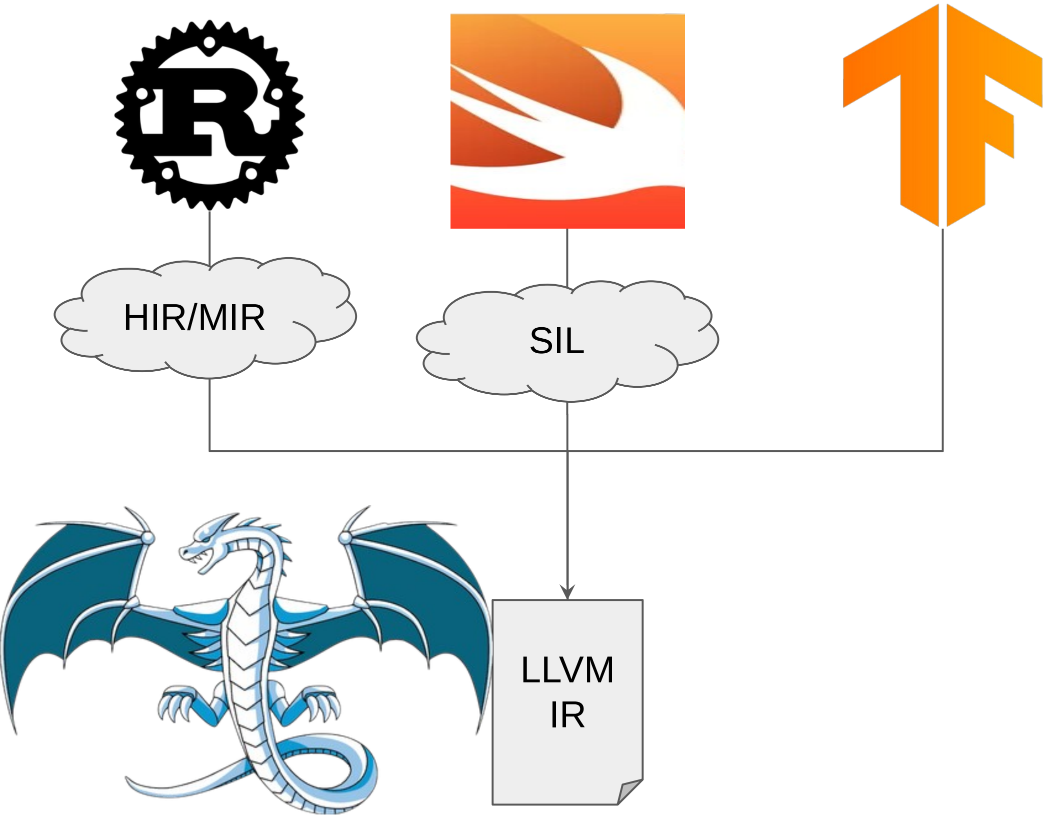
LLVM Happened



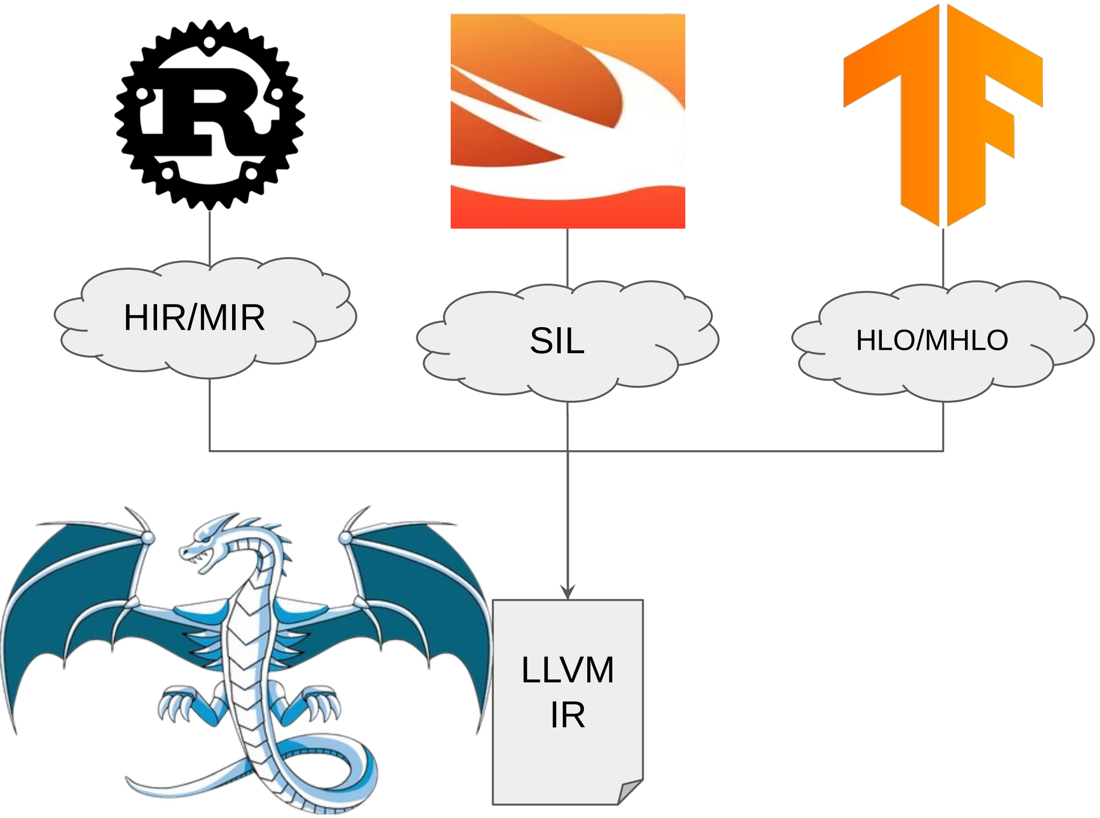
LLVM Happened



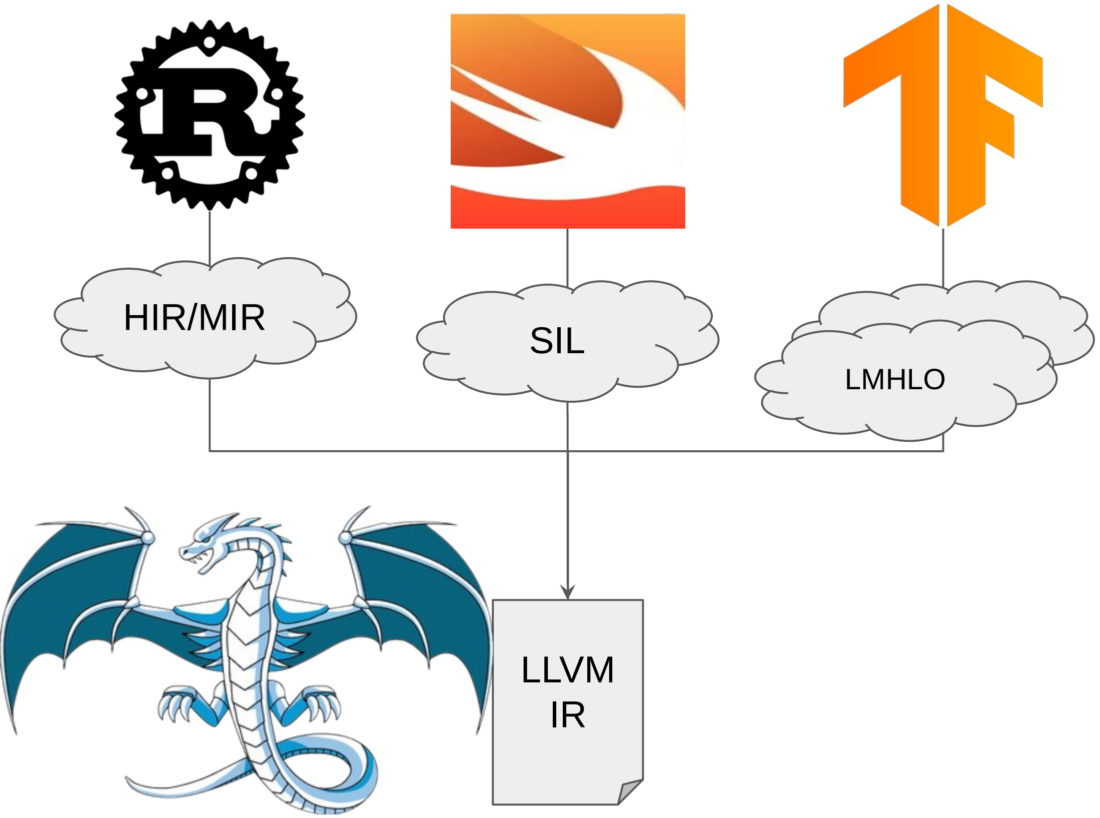
LLVM Happened



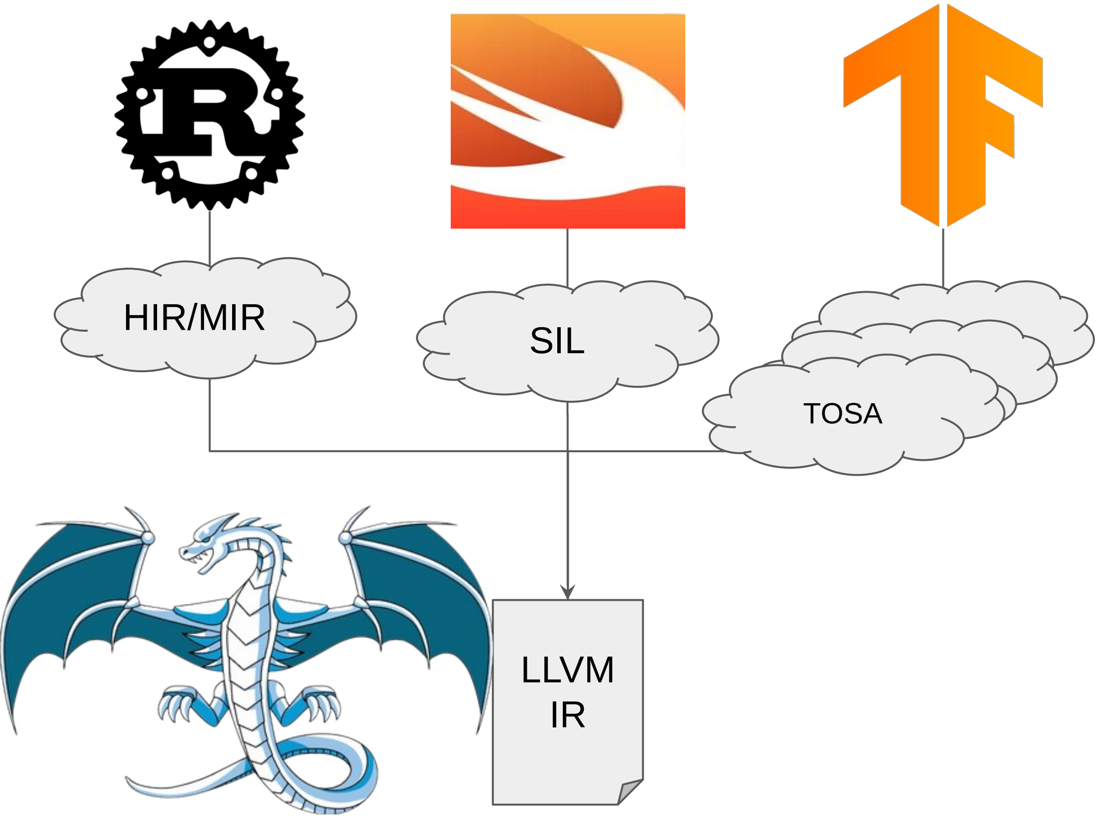
LLVM Happened



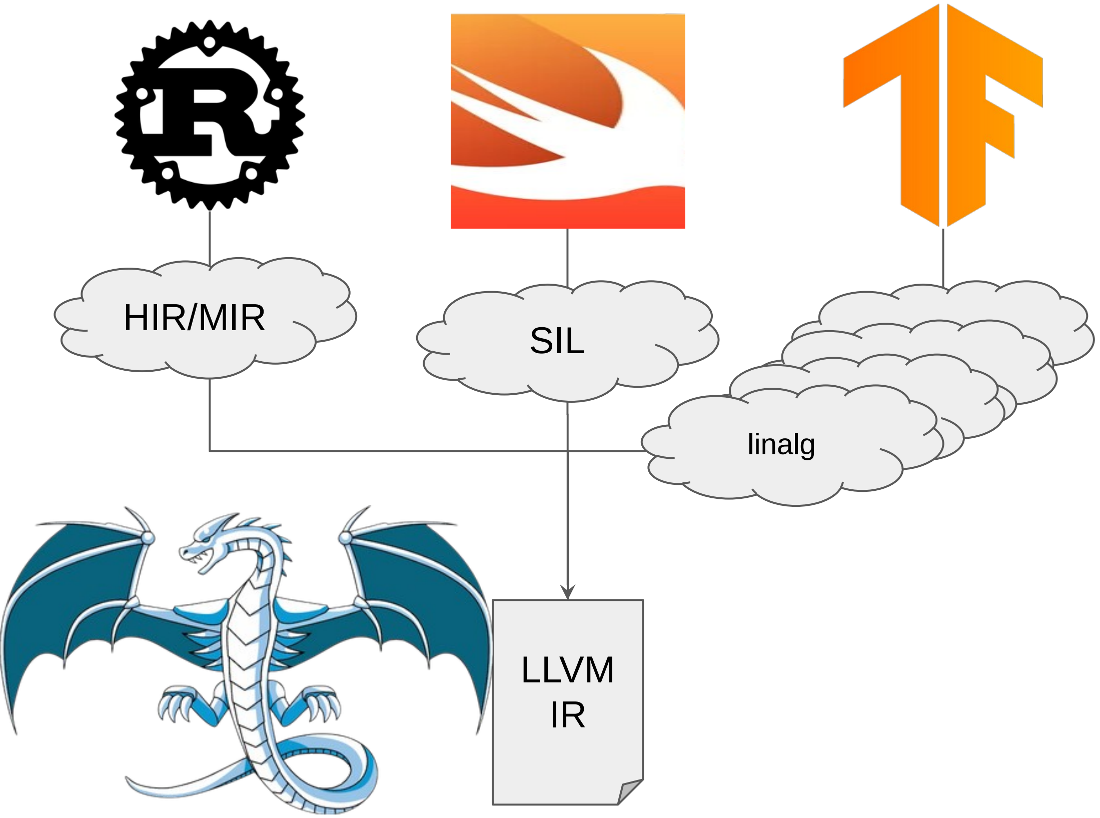
LLVM Happened



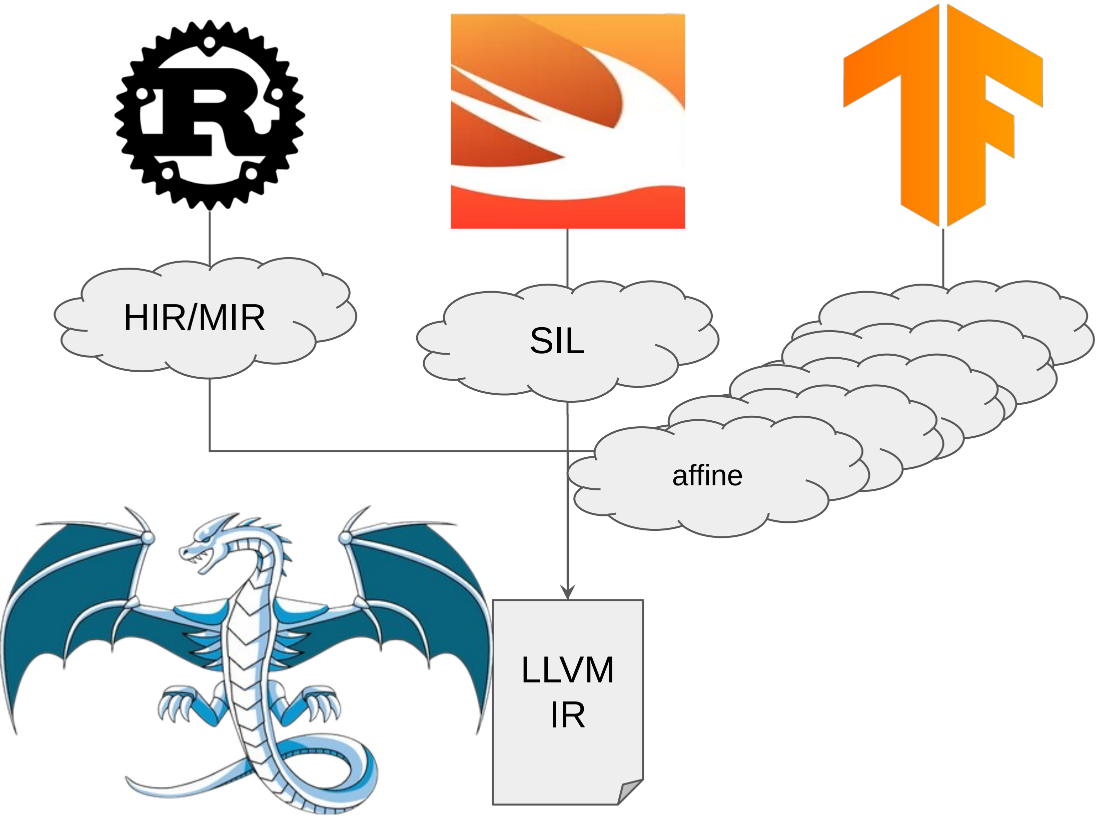
LLVM Happened



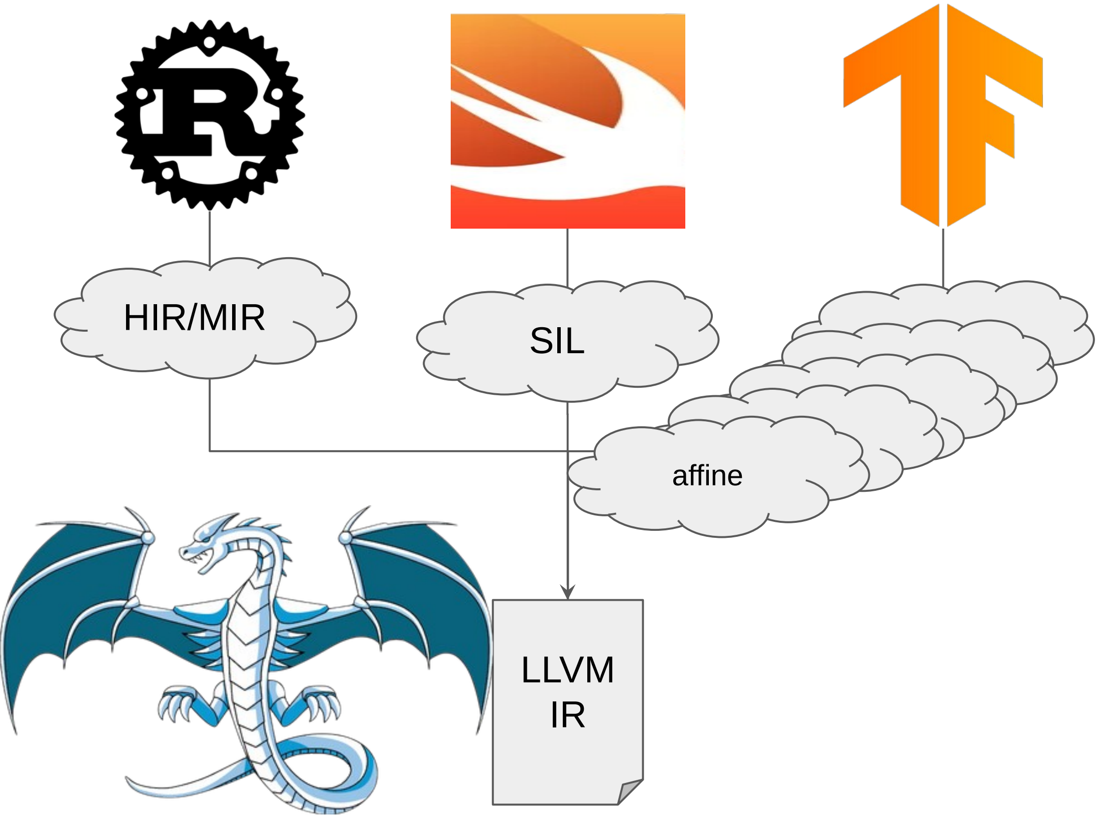
LLVM Happened



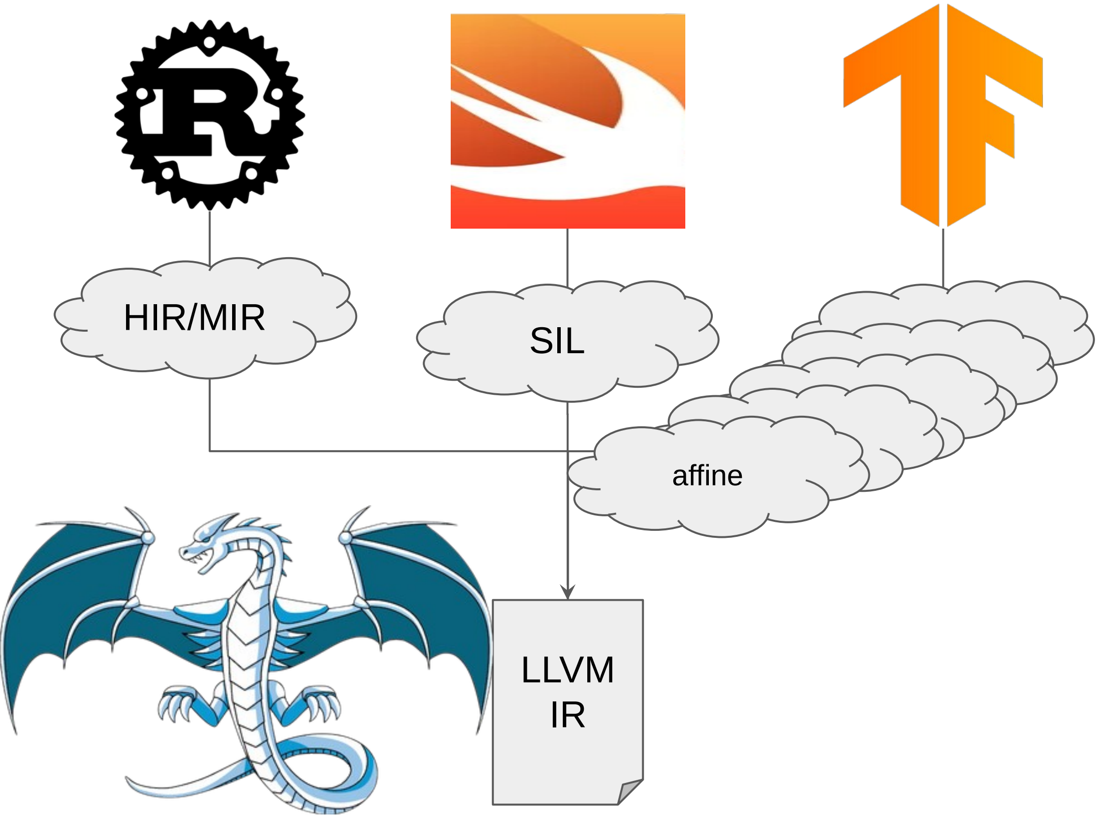
LLVM Happened



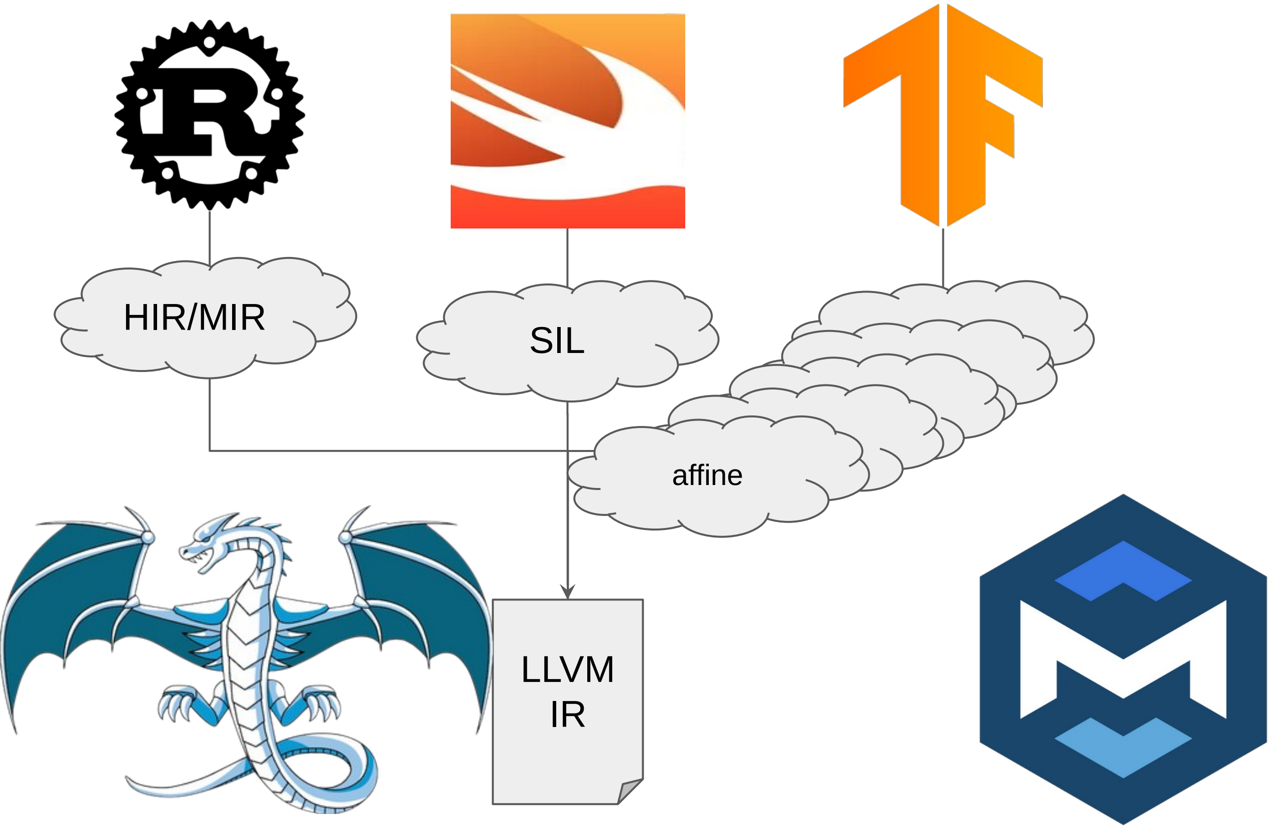
~~LLVM~~ Happened



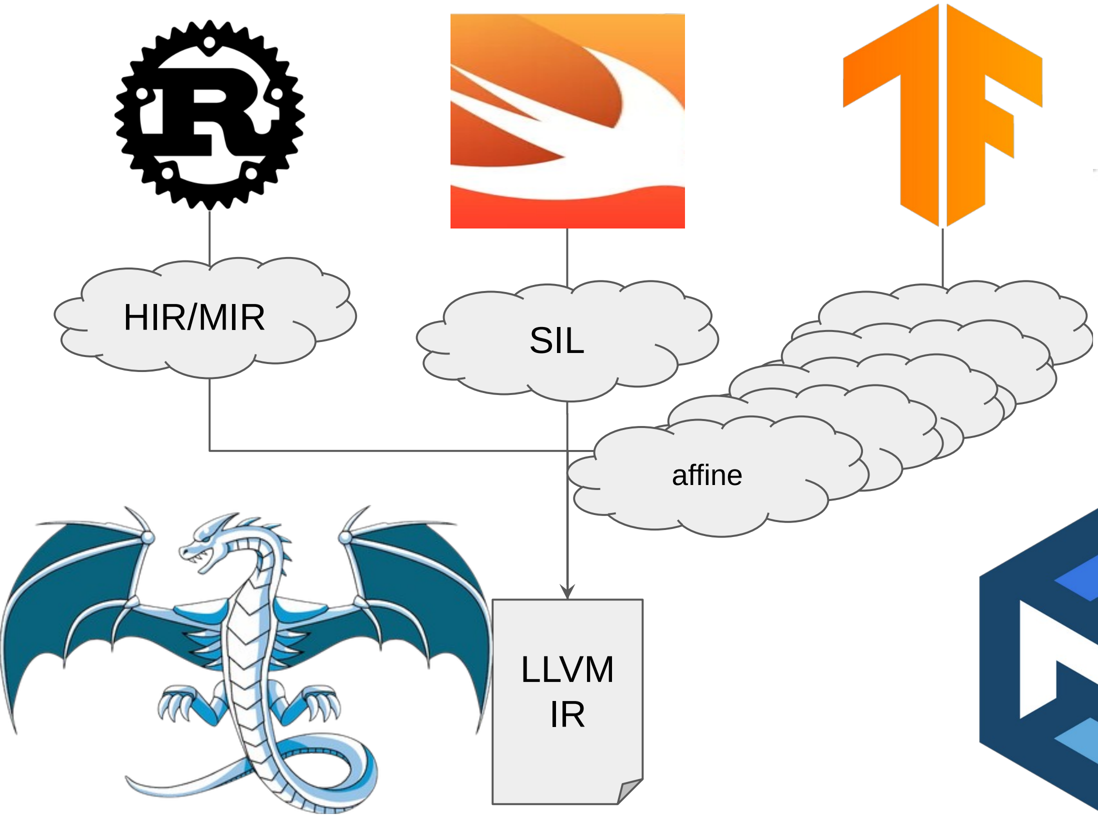
MLIR ~~LLVM~~ Happened

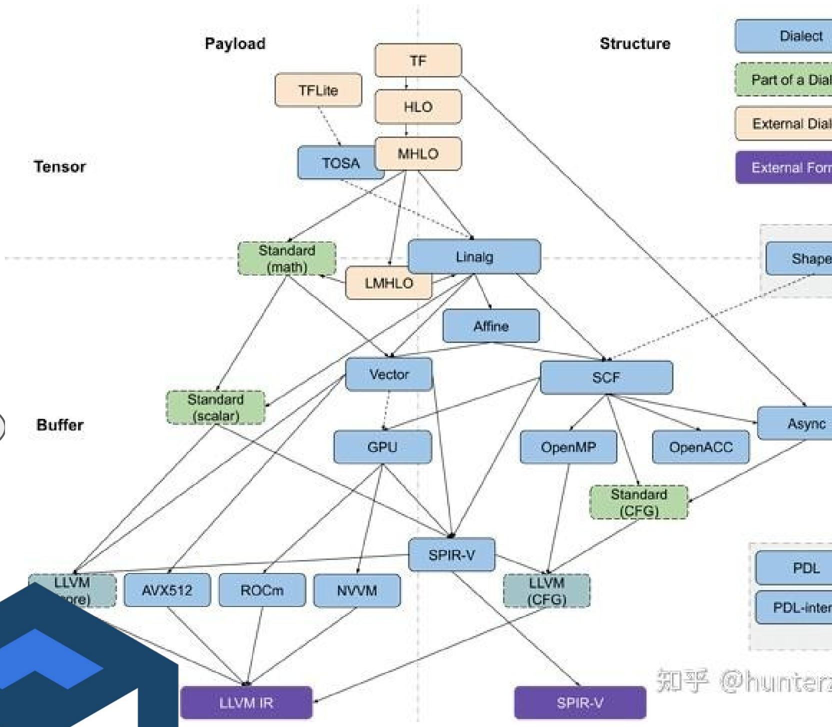


MLIR ~~LLVM~~ Happened

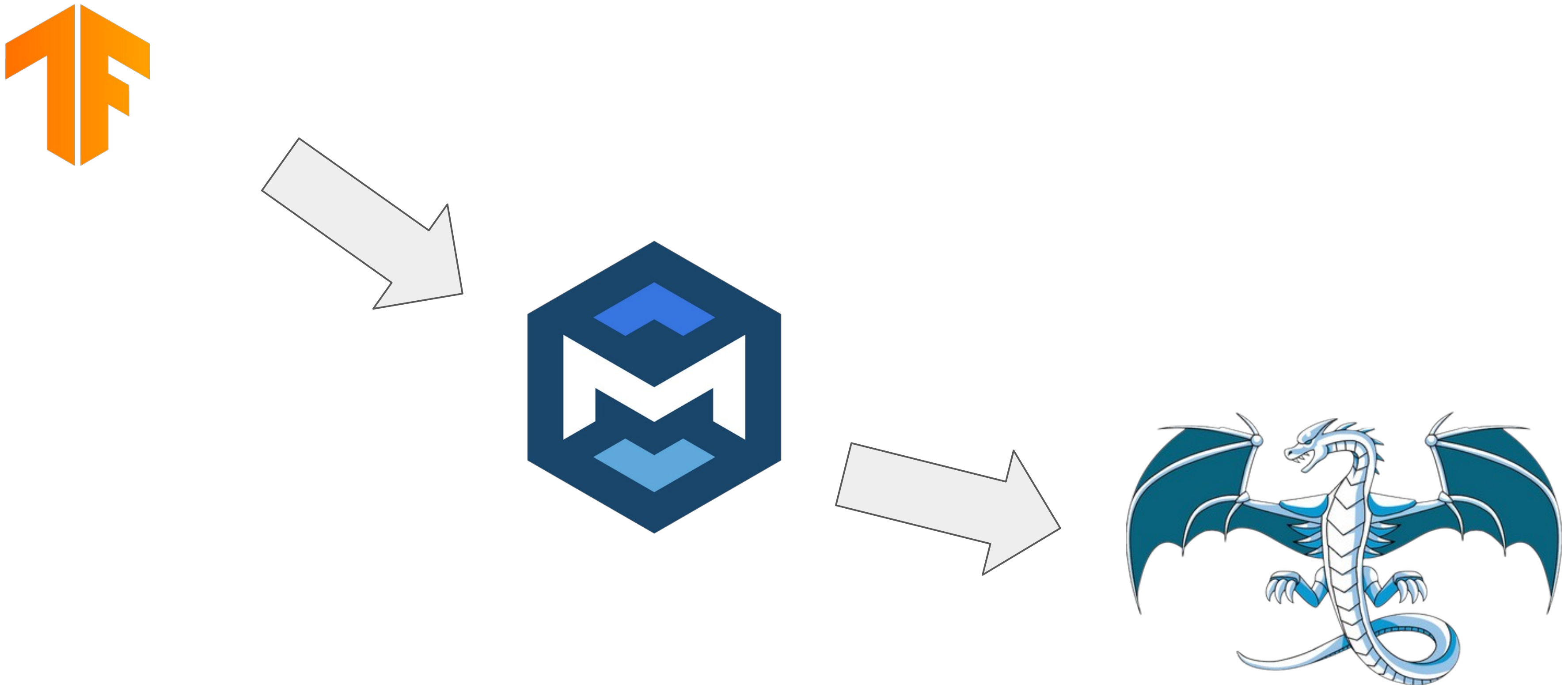


MLIR ~~LLVM~~ Happened

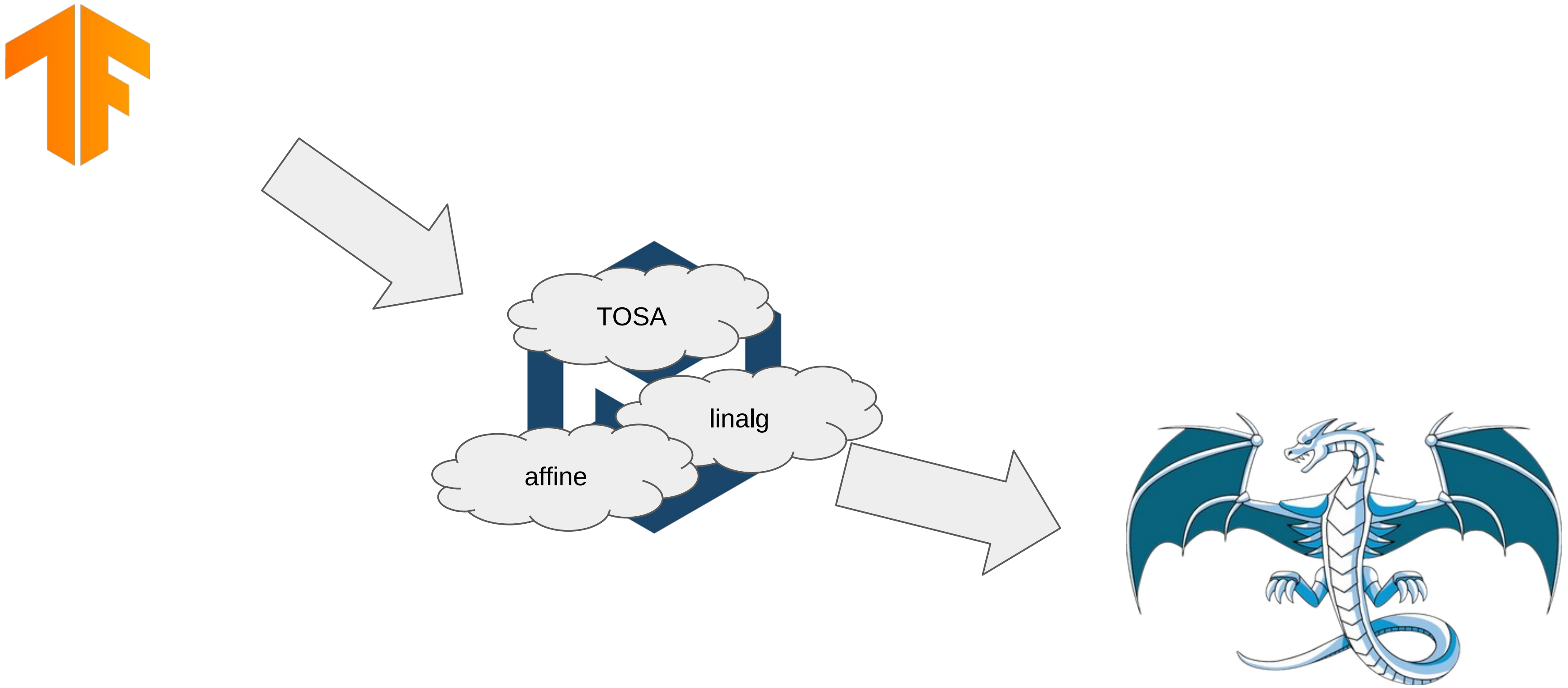




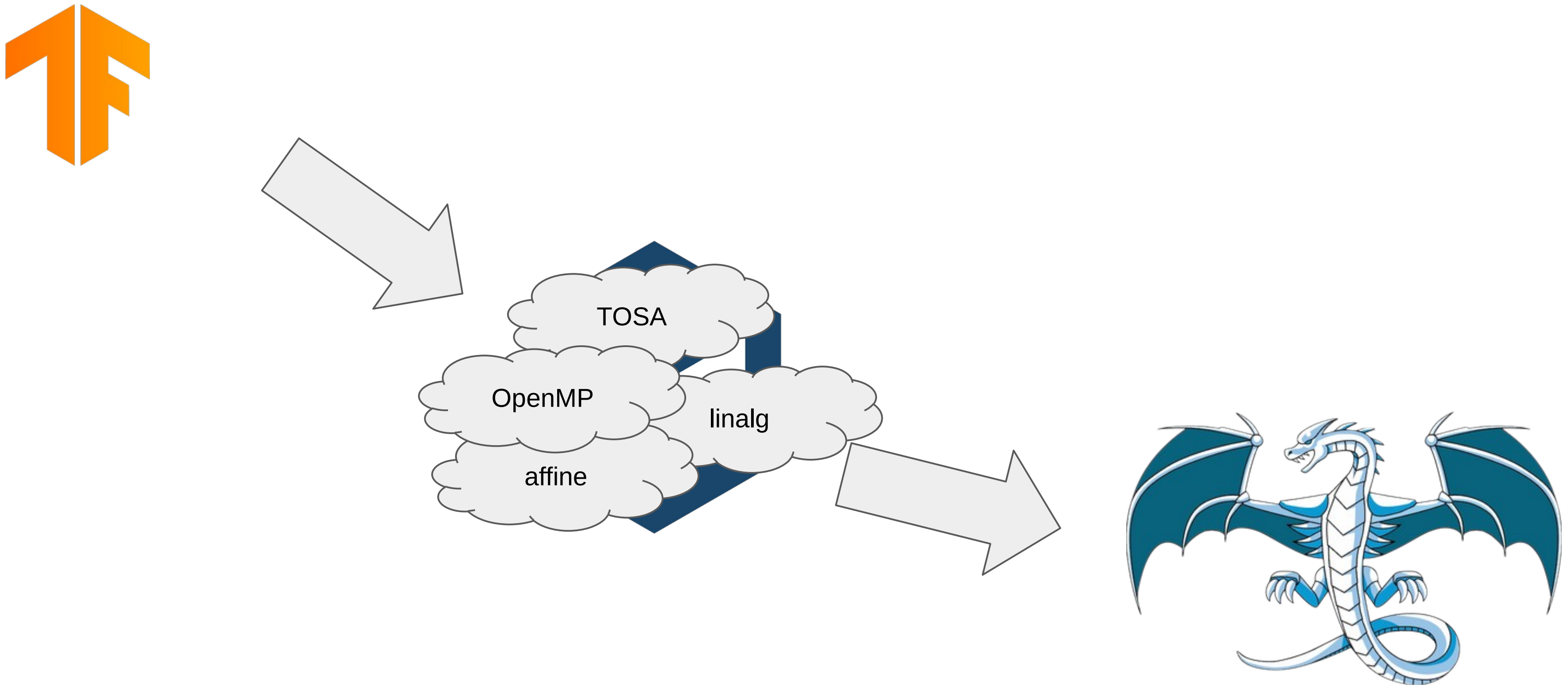
Where are we going?



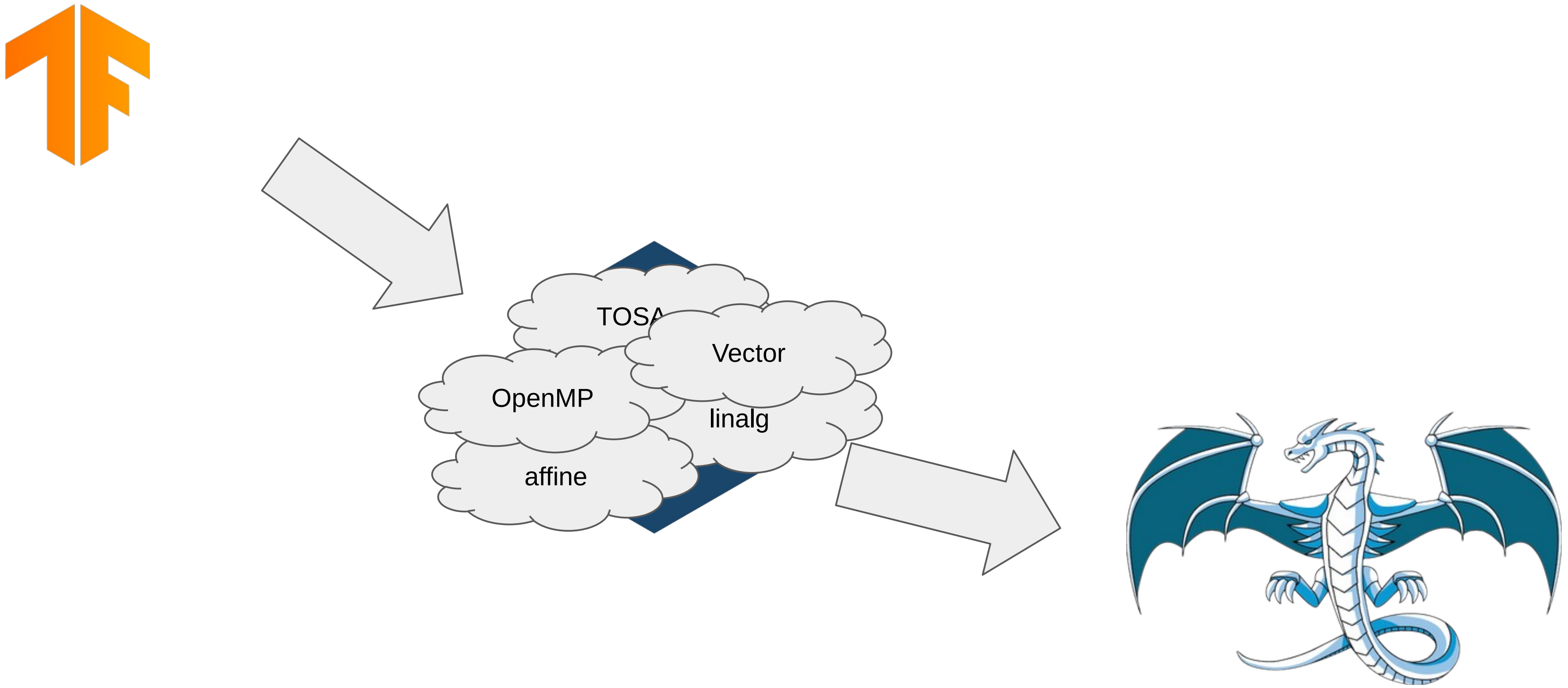
Where are we going?



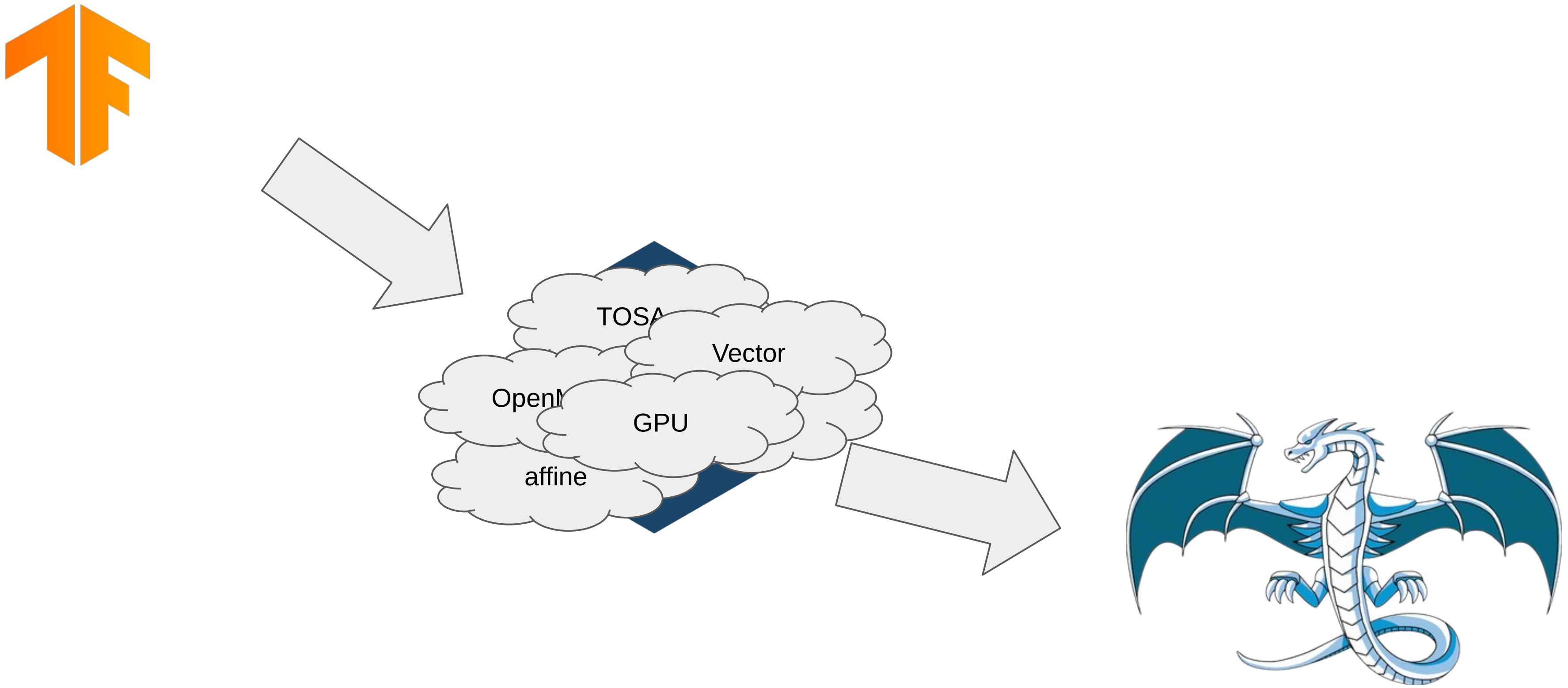
Where are we going?



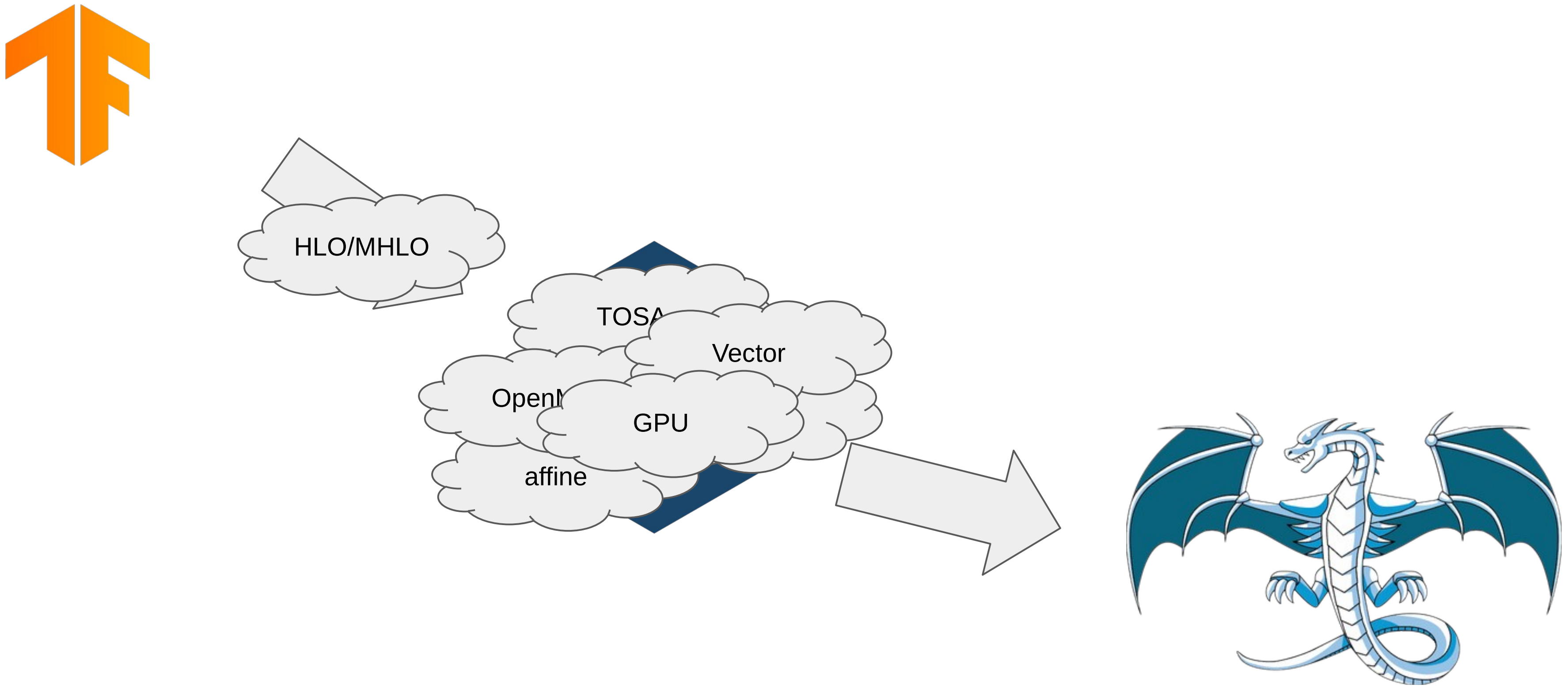
Where are we going?



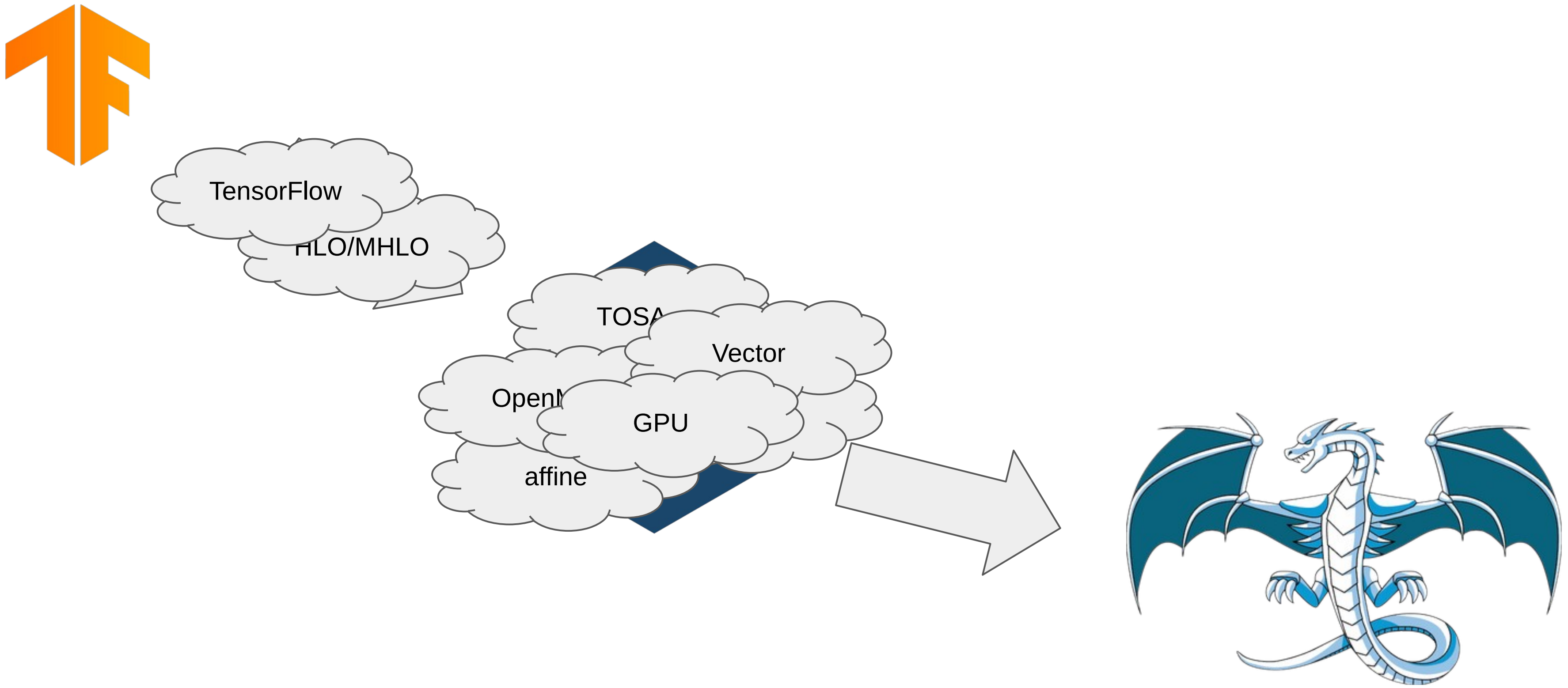
Where are we going?



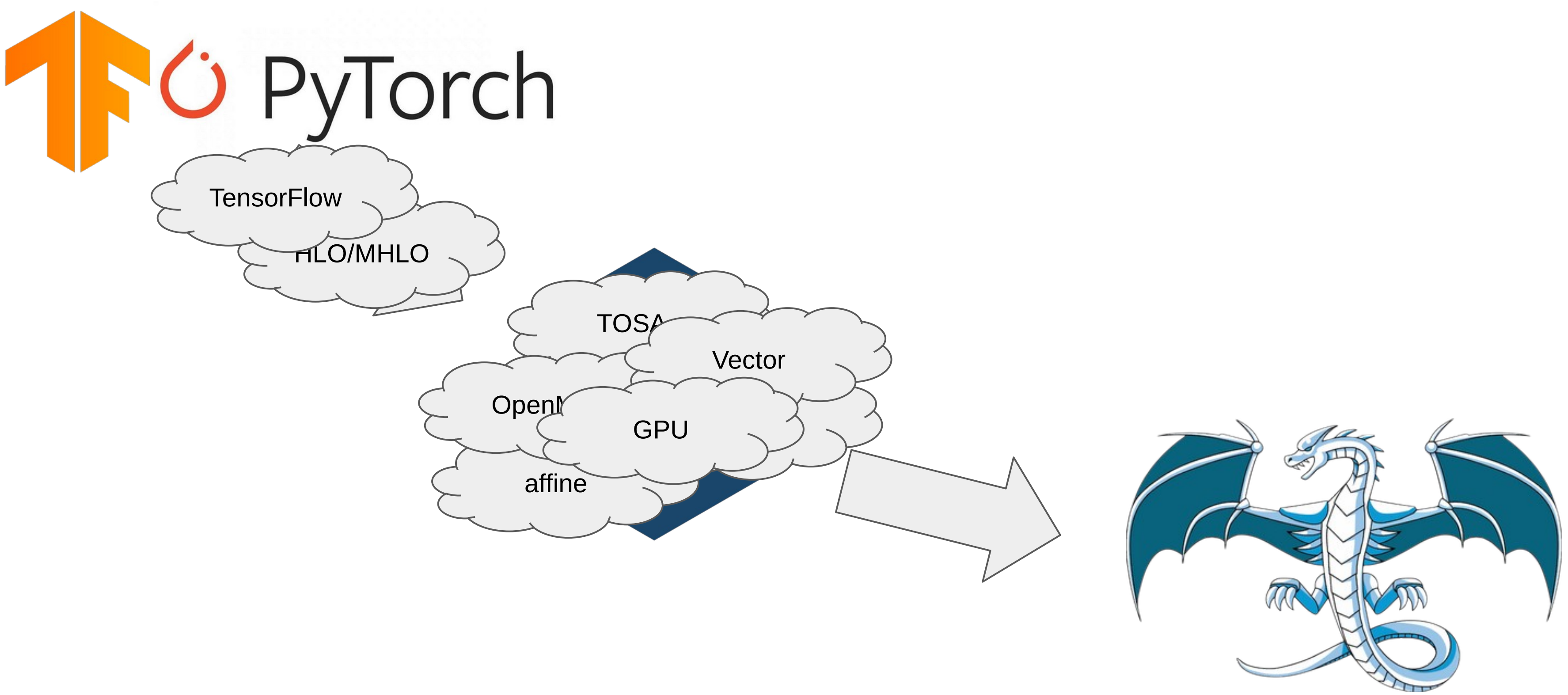
Where are we going?



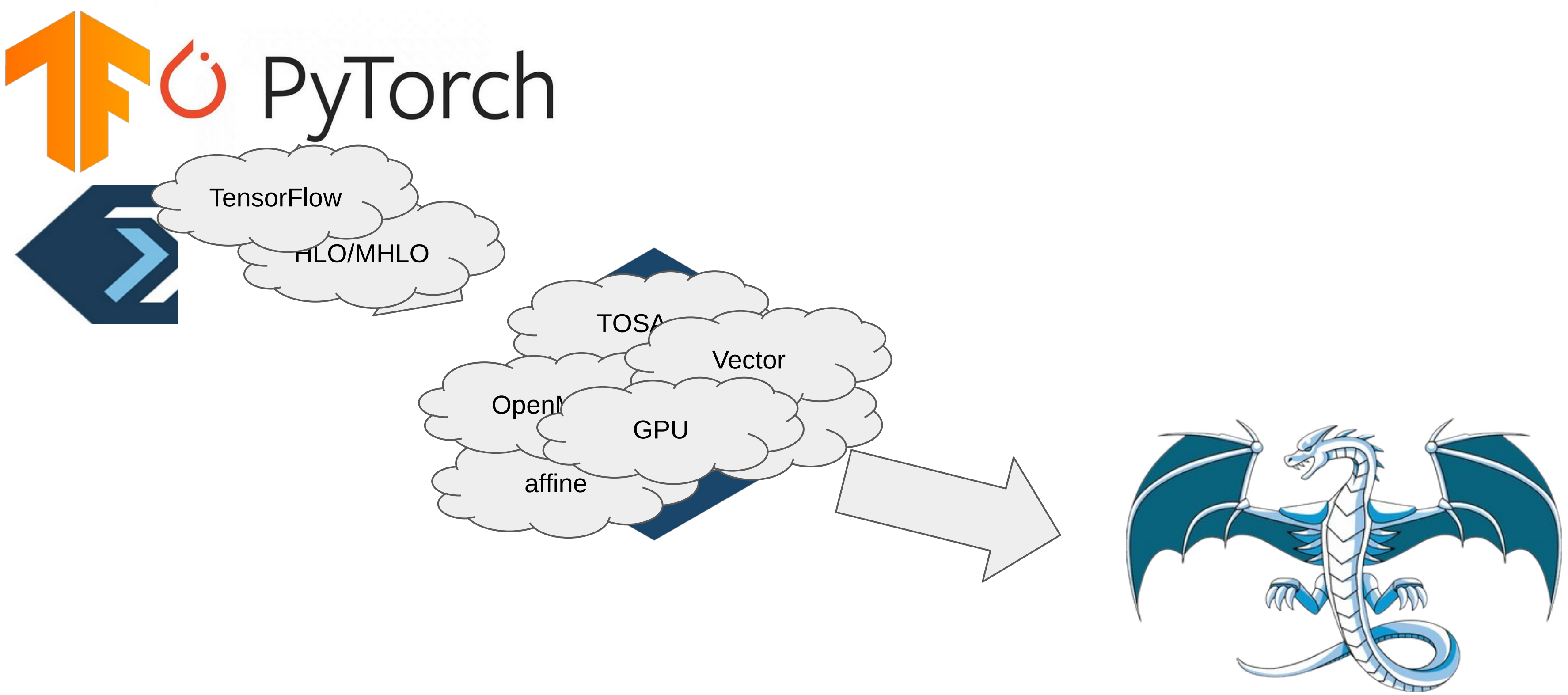
Where are we going?



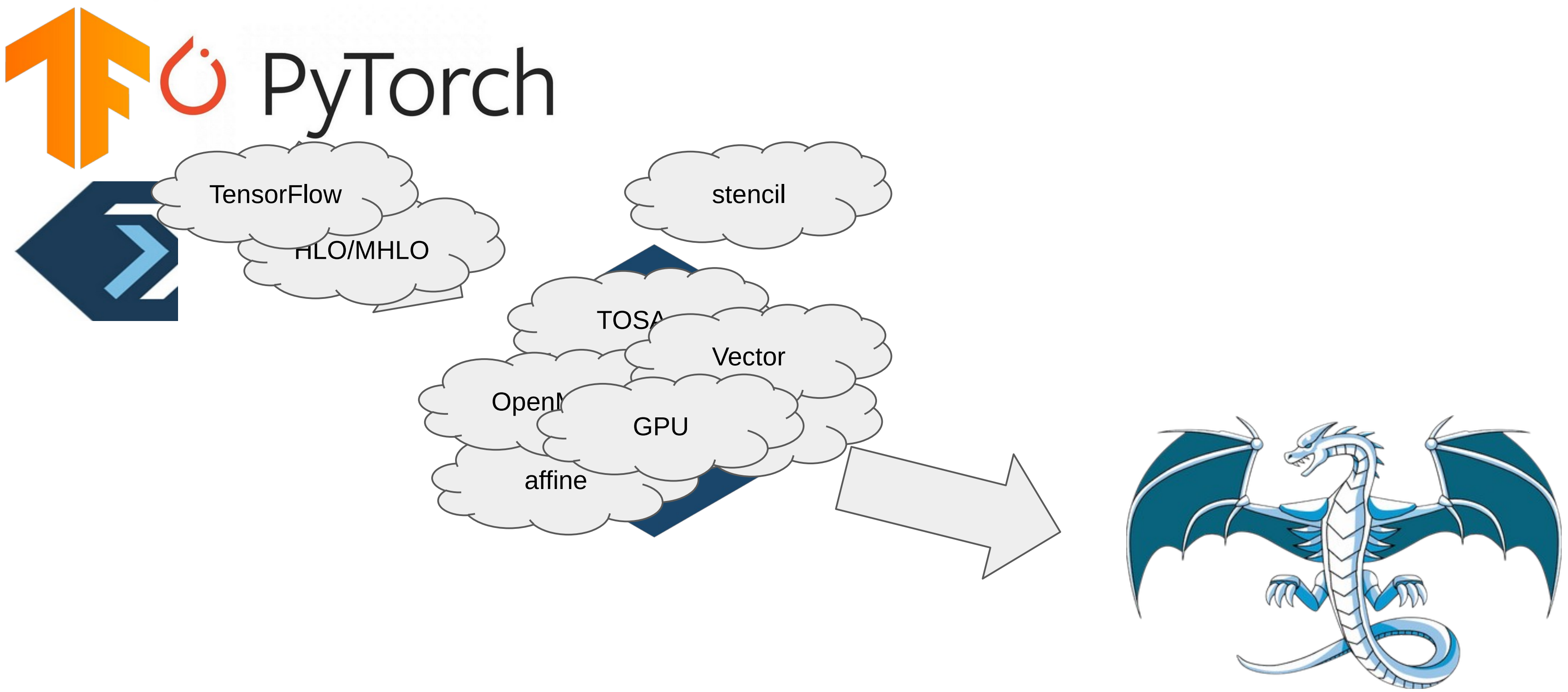
Where are we going?



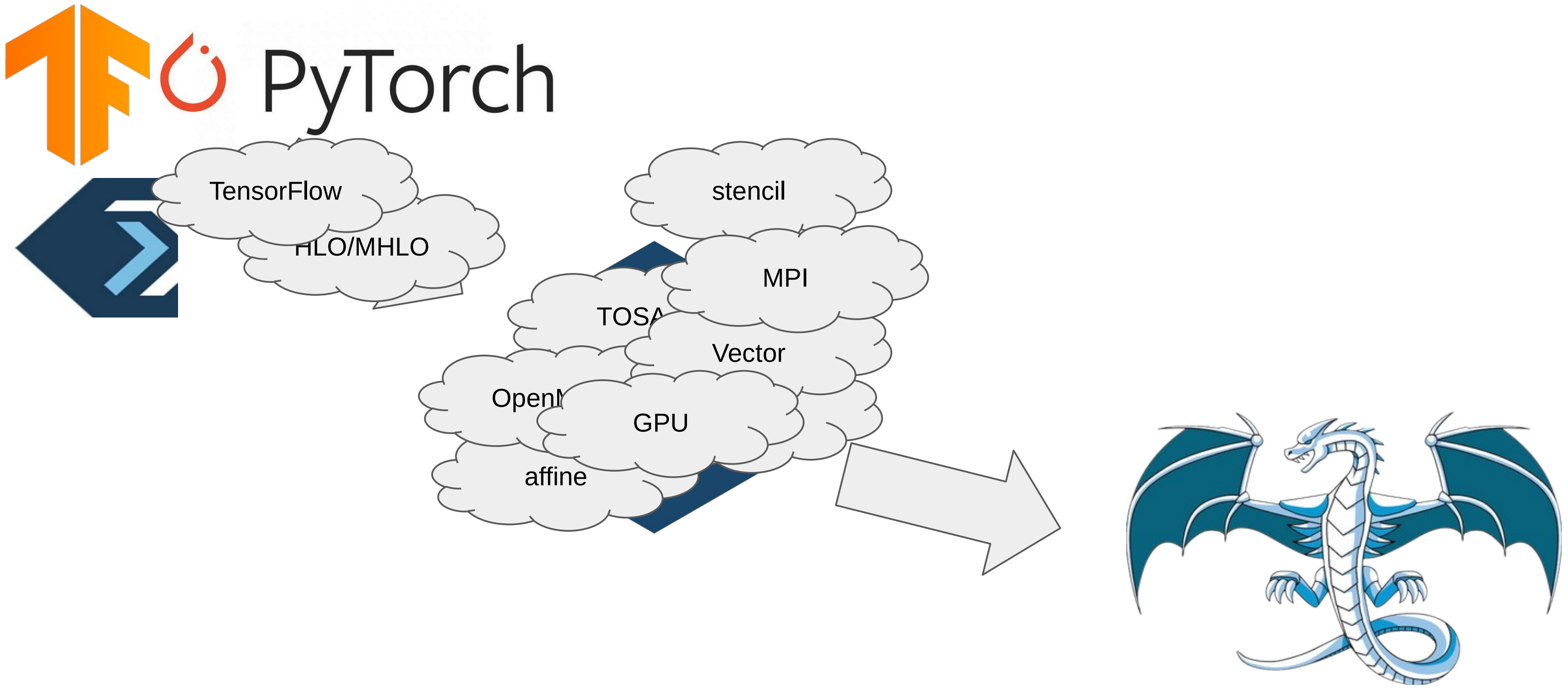
Where are we going?



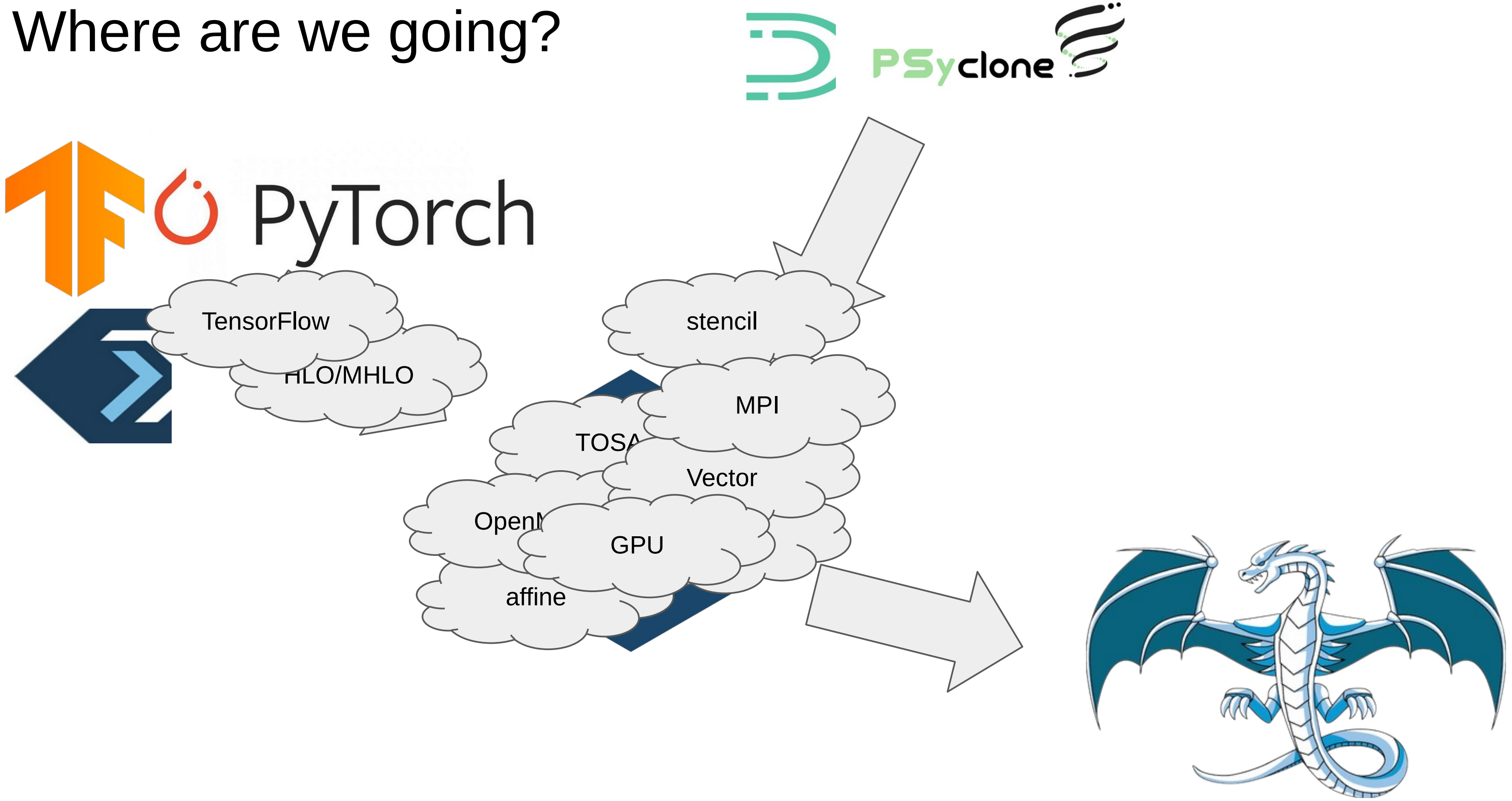
Where are we going?



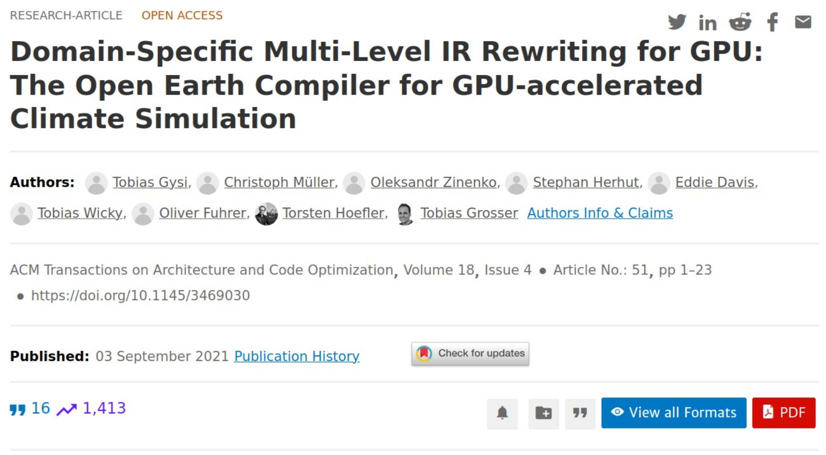
Where are we going?



Where are we going?



A generalized stencil dialect?


RESEARCH-ARTICLE OPEN ACCESS

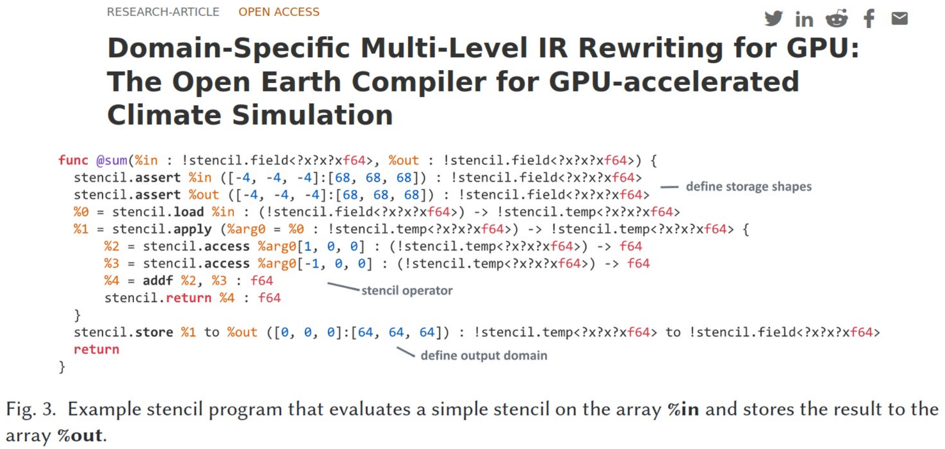
Domain-Specific Multi-Level IR Rewriting for GPU: The Open Earth Compiler for GPU-accelerated Climate Simulation

Authors: Tobias Gysi, Christoph Müller, Oleksandr Zinenko, Stephan Herhut, Eddie Davis, Tobias Wicky, Oliver Fuhrer, Torsten Hoefler, Tobias Grosser Authors Info & Claims

ACM Transactions on Architecture and Code Optimization, Volume 18, Issue 4 • Article No.: 51, pp 1–23 • https://doi.org/10.1145/3469030

Published: 03 September 2021 Publication History

A generalized stencil dialect?



RESEARCH-ARTICLE OPEN ACCESS

Domain-Specific Multi-Level IR Rewriting for GPU: The Open Earth Compiler for GPU-accelerated Climate Simulation

```
func @sum(%in : !stencil.field<?x?x?xf64>, %out : !stencil.field<?x?x?xf64>) {
  stencil.assert %in ([-4, -4, -4]:[68, 68, 68]) : !stencil.field<?x?x?xf64>     — define storage shapes
  stencil.assert %out ([-4, -4, -4]:[68, 68, 68]) : !stencil.field<?x?x?xf64>
  %0 = stencil.load %in : (!stencil.field<?x?x?xf64>) -> !stencil.temp<?x?x?xf64>
  %1 = stencil.apply (%arg0 = %0 : !stencil.temp<?x?x?xf64>) -> !stencil.temp<?x?x?xf64> {
      %2 = stencil.access %arg0[1, 0, 0] : (!stencil.temp<?x?x?xf64>) -> f64
      %3 = stencil.access %arg0[-1, 0, 0] : (!stencil.temp<?x?x?xf64>) -> f64
      %4 = addf %2, %3 : f64         stencil operator
      stencil.return %4 : f64
  }
  stencil.store %1 to %out ([0, 0, 0]:[64, 64, 64]) : !stencil.temp<?x?x?xf64> to !stencil.field<?x?x?xf64>
  return                                 define output domain
}
```

Fig. 3. Example stencil program that evaluates a simple stencil on the array **%in** and stores the result to the array **%out**.

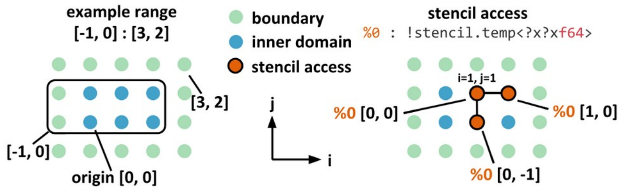


Fig. 4. Example range (left) defined by an inclusive lower and an exclusive upper bound and stencil accesses (right) expressed relative to the current position (i = 1, j = 1).

A generalized stencil dialect?

RESEARCH-ARTICLE OPEN ACCESS

Domain-Specific Multi-Level IR Rewriting for GPU: The Open Earth Compiler for GPU-accelerated Climate Simulation

```
func @sum(%in : !stencil.field<?x?x?xf64>, %out : !stencil.field<?x?x?xf64>) {
  stencil.assert %in ([-4, -4, -4]:[68, 68, 68]) : !stencil.field<?x?x?xf64>
  stencil.assert %out ([-4, -4, -4]:[68, 68, 68]) : !stencil.field<?x?x?xf64>    — define storage shapes
  %0 = stencil.load %in : (!stencil.field<?x?x?xf64>) -> !stencil.temp<?x?x?xf64>
  %1 = stencil.apply (%arg0 = %0 : !stencil.temp<?x?x?xf64>) -> !stencil.temp<?x?x?xf64> {
      %2 = stencil.access %arg0[1, 0, 0] : (!stencil.temp<?x?x?xf64>) -> f64
      %3 = stencil.access %arg0[-1, 0, 0] : (!stencil.temp<?x?x?xf64>) -> f64
      %4 = addf %2, %3 : f64        stencil operator
      stencil.return %4 : f64
  }
  stencil.store %1 to %out ([0, 0, 0]:[64, 64, 64]) : !stencil.temp<?x?x?xf64> to !stencil.field<?x?x?xf64>
  return                         define output domain
}
```

Fig. 3. Example stencil program that evaluates a simple stencil on the array **%in** and stores the result to the array **%out**.

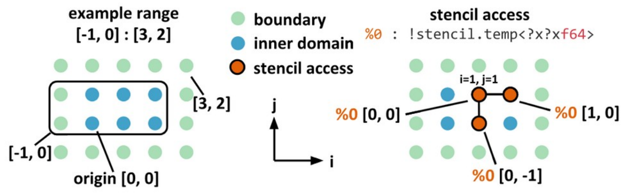
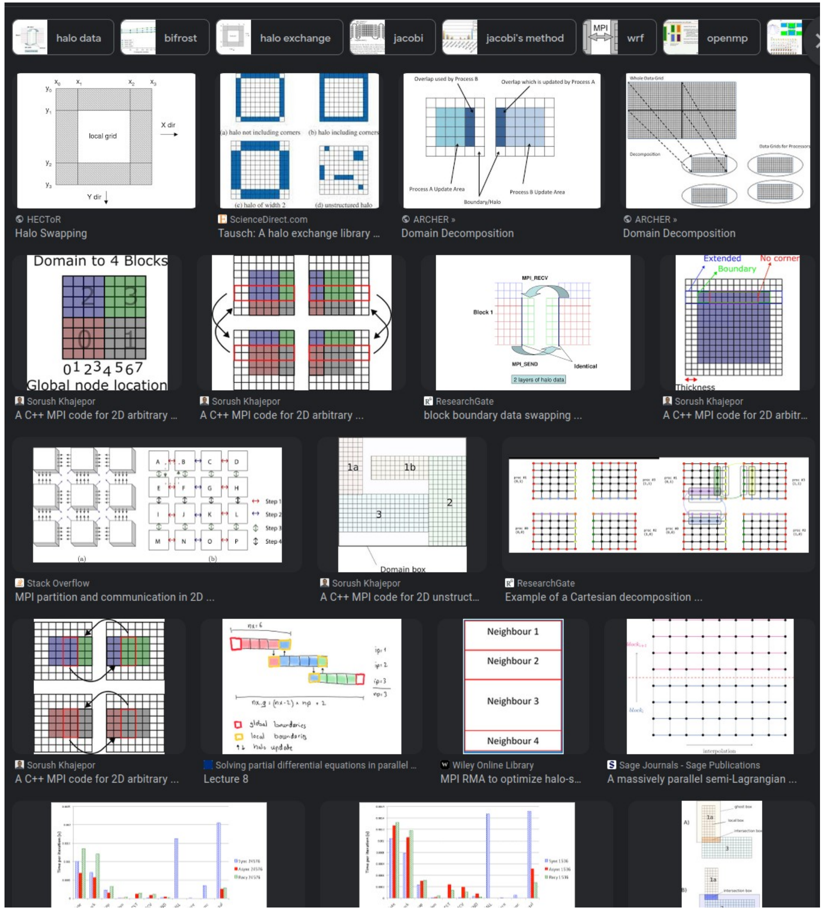


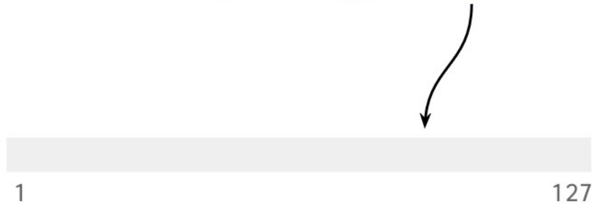
Fig. 4. Example range (left) defined by an inclusive lower and an exclusive upper bound and stencil accesses (right) expressed relative to the current position (i = 1, j = 1).


halo data
bifrost
halo exchange
jacobi
jacobi's method
wrf
openmp
HECToR
Halo Swapping
ScienceDirect.com
Tausch: A halo exchange library ...
ARCHER »
Domain Decomposition
ARCHER »
Domain Decomposition
Sorush Khajepor
A C++ MPI code for 2D arbitrary ...
Sorush Khajepor
A C++ MPI code for 2D arbitrary ...
ResearchGate
block boundary data swapping ...
Sorush Khajepor
A C++ MPI code for 2D arbitr...
Stack Overflow
MPI partition and communication in 2D ...
Sorush Khajepor
A C++ MPI code for 2D unstruct...
ResearchGate
Example of a Cartesian decomposition ...
Sorush Khajepor
A C++ MPI code for 2D arbitrary ...
Solving partial differential equations in parallel ...
Lecture 8
Wiley Online Library
MPI RMA to optimize halo-s...
Sage Journals - Sage Publications
A massively parallel semi-Lagrangian ...

A ~~generalized~~ distributed stencil dialect?

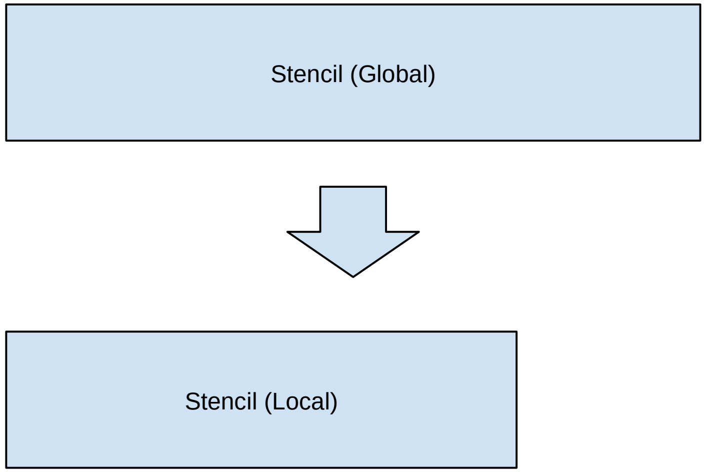
Stencil (Global)

```
%source = stencil.load(%114) : (!field<[0,128]xf64>)
                                -> !temp<?xf64>
%out = stencil.apply(%arg = %source : !temp<?xf64>)
                          -> !temp<?xf64> {
  %l = stencil.access %arg[-1] : f64
  %c = stencil.access %arg[0] : f64
  %r = stencil.access %arg[1] : f64
  // %v = %l + %r - 2.0 * %c
  stencil.return %v : f64
}

stencil.store %out to %target([1]:[127])
```

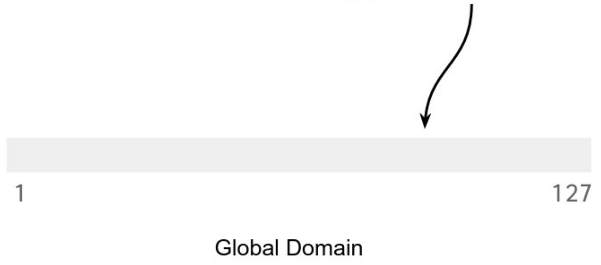


Global Domain

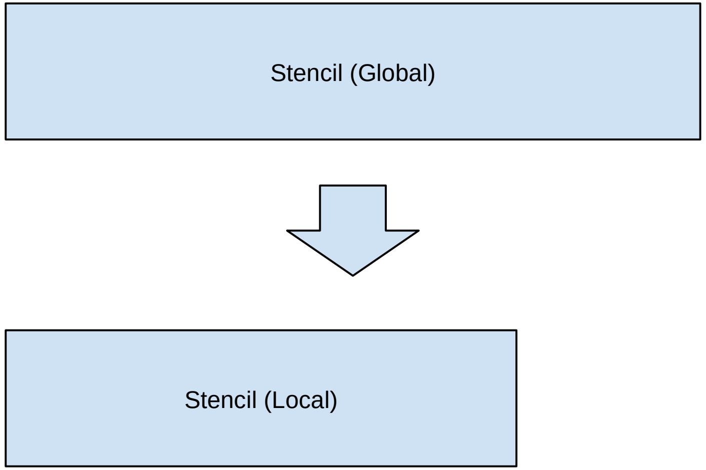
A ~~generalized~~ distributed stencil dialect?



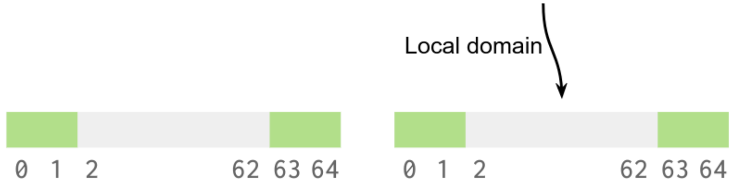
```
%source = stencil.load(%114) : (!field<[0,128]xf64>)
                                 -> !temp<?xf64>
%out = stencil.apply(%arg = %source : !temp<?xf64>)
                      -> !temp<?xf64> {
  %l = stencil.access %arg[-1] : f64
  %c = stencil.access %arg[0] : f64
  %r = stencil.access %arg[1] : f64
  // %v = %l + %r - 2.0 * %c
  stencil.return %v : f64
}

stencil.store %out to %target([1]:[127])
```



A ~~generalized~~ distributed stencil dialect?

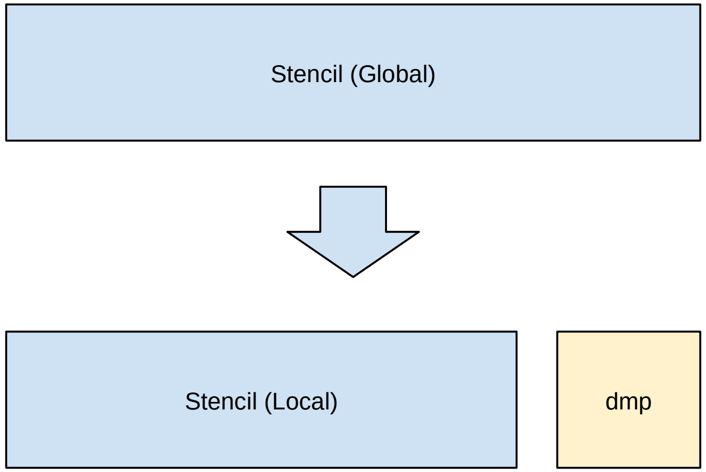


```
%source = stencil.load(%114) : (!field<[0,64]xf64> )
                                 -> !temp<?xf64>
%out = stencil.apply(%arg = %source : !temp<?xf64>)
                         -> !temp<?xf64> {
  %l = stencil.access %arg[-1] : f64
  %c = stencil.access %arg[0] : f64
  %r = stencil.access %arg[1] : f64
  // %v = %l + %r - 2.0 * %c
  stencil.return %v : f64
}

stencil.store %out to %target([1]:[64])
```

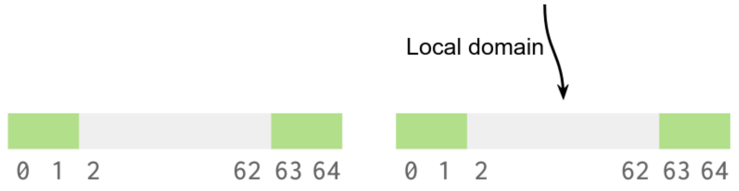


Local Domains with halo exchanges highlighted

A ~~generalized~~ distributed stencil dialect?

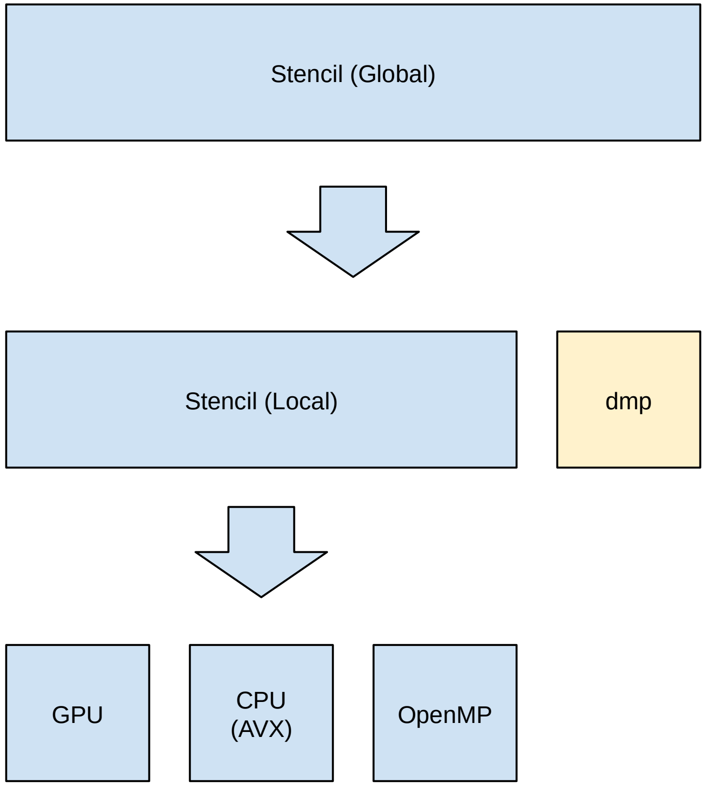


```
dmp.swap(%ref) {
  "grid" = #dmp.grid<2>,
  "swaps" = [
    #dmp.exchange<at [0] size [1]
                  source offset [1] to [-1]>,
    #dmp.exchange<at [64] size [1]
                  source offset [-1] to [1]>
  ]
}

%source = stencil.load(%114) : (!field<[0,64]xf64> )
                                -> !temp<?xf64>
%out = stencil.apply(%arg = %source : !temp<?xf64>)
                    -> !temp<?xf64> {
  %l = stencil.access %arg[-1] : f64
  %c = stencil.access %arg[0] : f64
  %r = stencil.access %arg[1] : f64
  // %v = %l + %r - 2.0 * %c
  stencil.return %v : f64
}

stencil.store %out to %target([1]:[64])
```

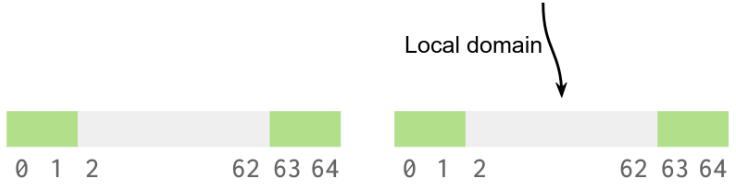


Local Domains with halo exchanges highlighted

A ~~generalized~~ distributed stencil dialect?

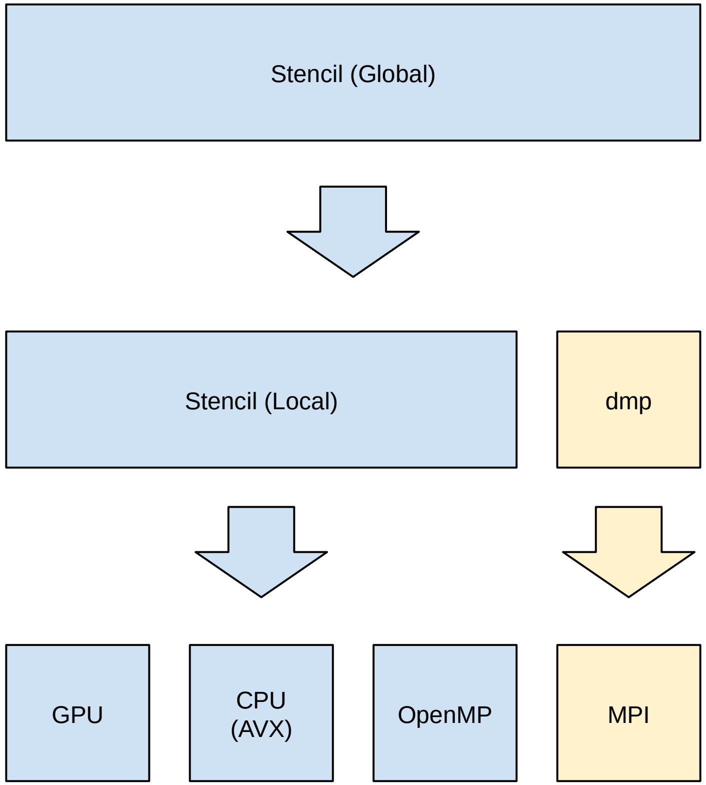


```
dmp.swap(%ref) {
  "grid" = #dmp.grid<2>,
  "swaps" = [
    #dmp.exchange<at [0] size [1]
                  source offset [1] to [-1]>,
    #dmp.exchange<at [64] size [1]
                  source offset [-1] to [1]>
  ]
} :

%source = stencil.load(%114) : (!field<[0,64]xf64> )
                                  -> !temp<?xf64>
%out = stencil.apply(%arg = %source : !temp<?xf64>)
                   -> !temp<?xf64> {
  %l = stencil.access %arg[-1] : f64
  %c = stencil.access %arg[0] : f64
  %r = stencil.access %arg[1] : f64
  // %v = %l + %r - 2.0 * %c
  stencil.return %v : f64
}

stencil.store %out to %target([1]:[64])
```

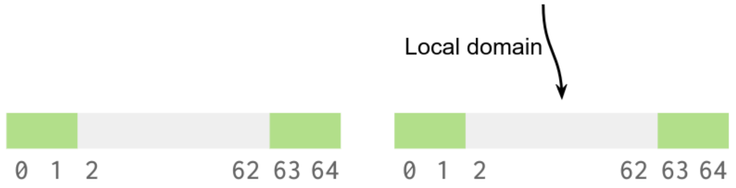


Local Domains with halo exchanges highlighted

A ~~generalized~~ distributed stencil dialect?

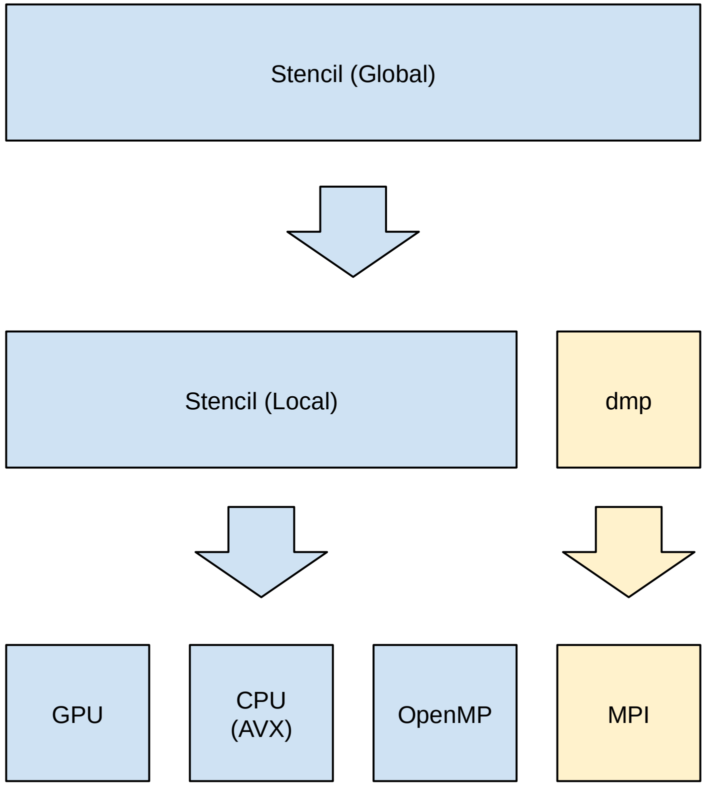


```
dmp.swap(%ref) {
  "grid" = #dmp.grid<2>,
  "swaps" = [
    #dmp.exchange<at [0] size [1]
                  source offset [1] to [-1]>,
    #dmp.exchange<at [64] size [1]
                  source offset [-1] to [1]>
  ]
}

%source = stencil.load(%114) : (!field<[0,64]xf64> )
                               -> !temp<?xf64>
%out = stencil.apply(%arg = %source : !temp<?xf64>)
                 -> !temp<?xf64> {
  %l = stencil.access %arg[-1] : f64
  %c = stencil.access %arg[0] : f64
  %r = stencil.access %arg[1] : f64
  // %v = %l + %r - 2.0 * %c
  stencil.return %v : f64
}

stencil.store %out to %target([1]:[64])
```

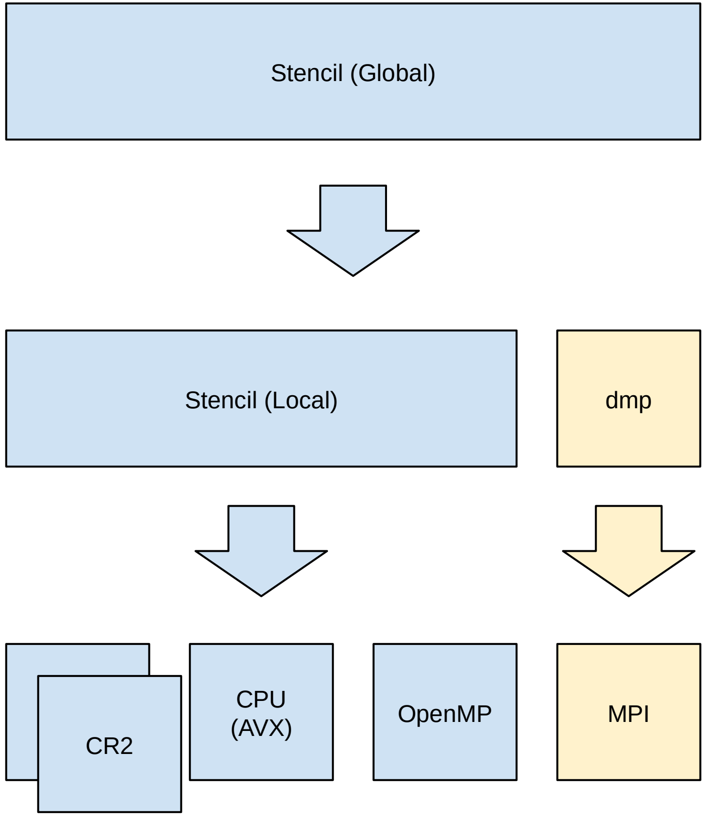


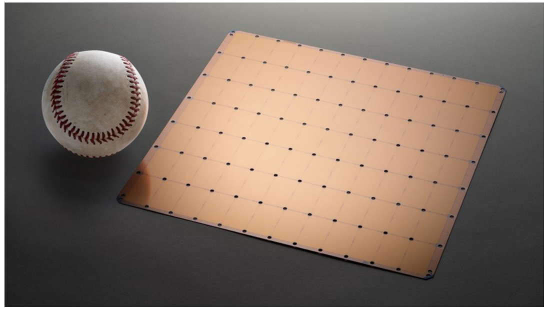
Local Domains with halo exchanges highlighted

Modularity unlocks Reusability

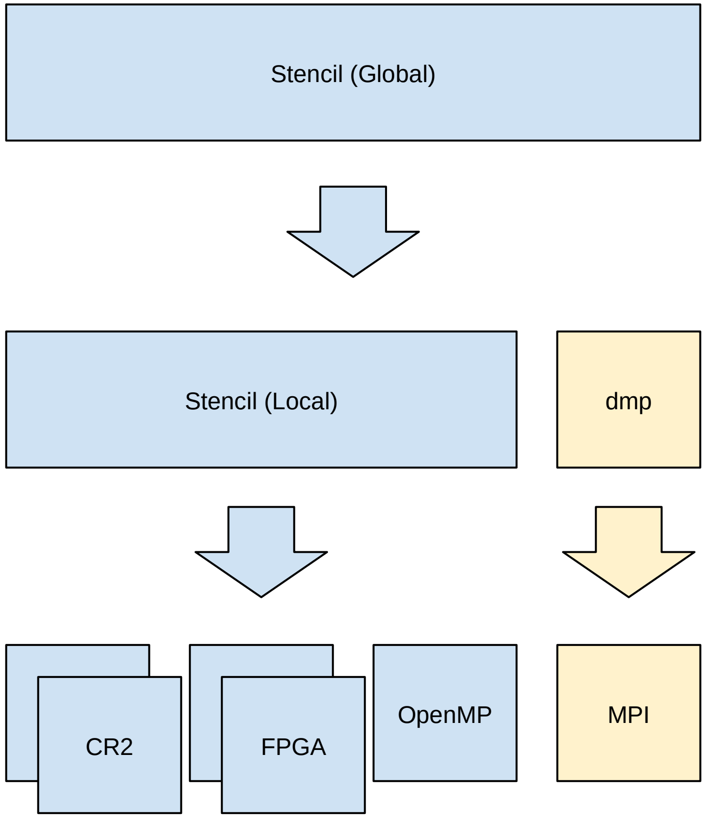


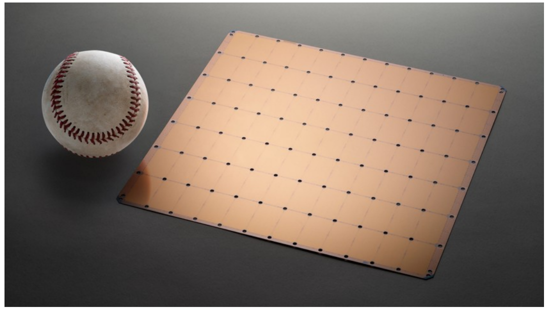
Modularity unlocks Reusability



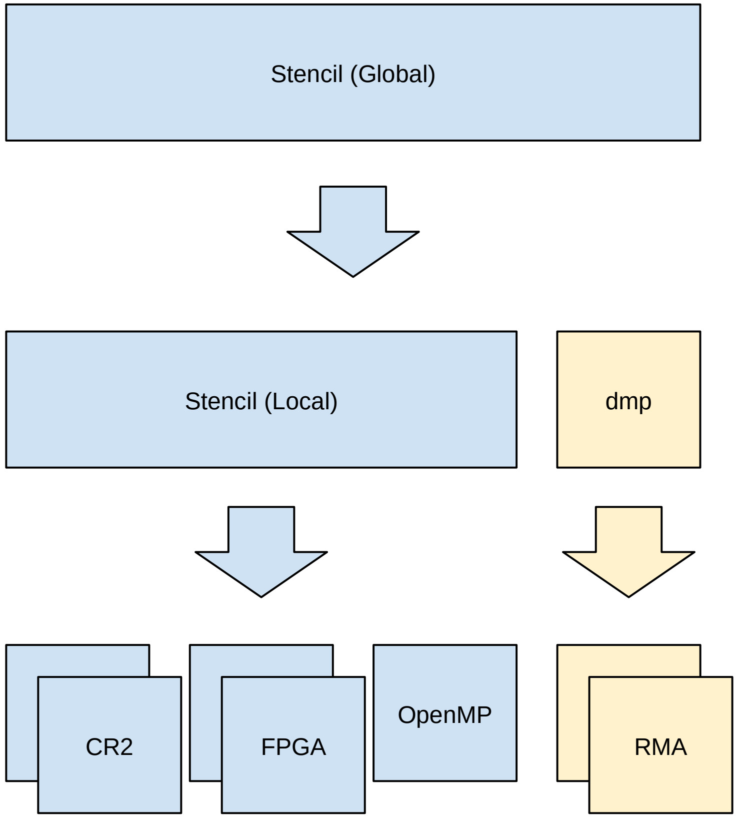


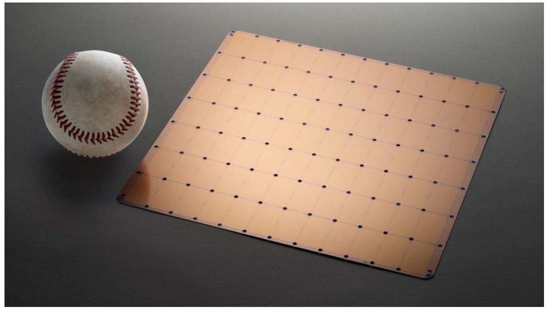
Modularity unlocks Reusability





Modularity unlocks Reusability





Shared Abstractions Benefit the Ecosystem

Shared Abstractions Benefit the Ecosystem

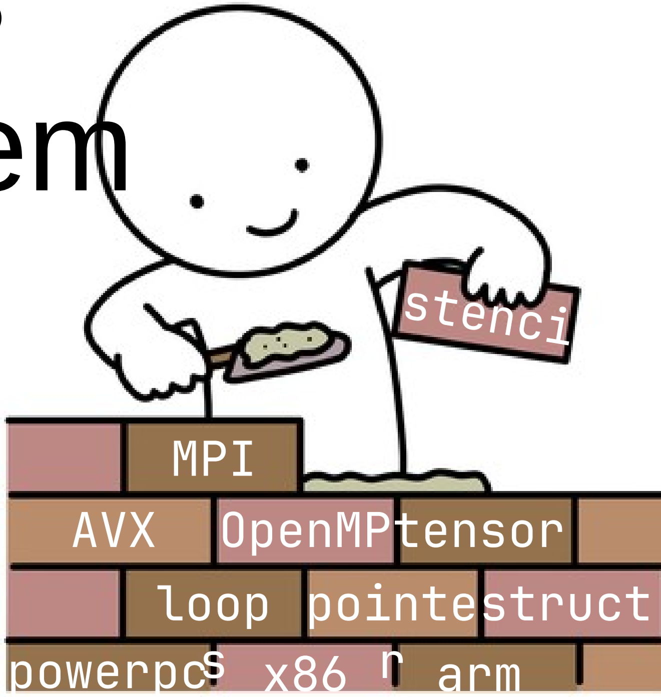
Stencils on Grids has been done to death.

Shared Abstractions Benefit the Ecosystem

Stencils on Grids has been done to death.

So let’s cement them into the compiler foundations!

Shared Abstractions Benefit the Ecosystem



Stencils on Grids has been done to death.

So let’s cement them into the compiler foundations!