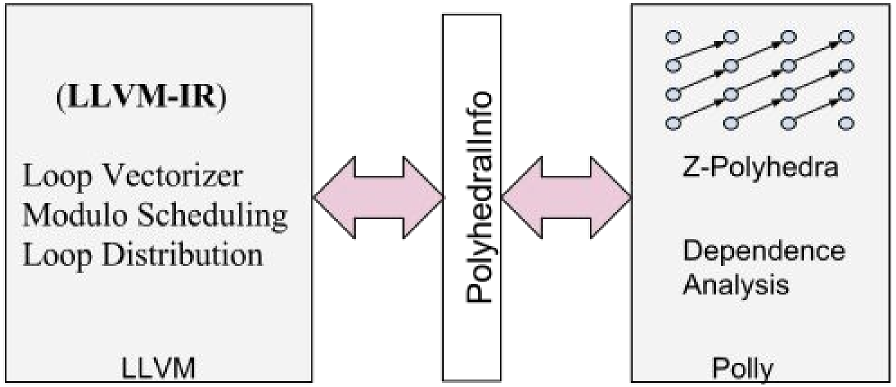
Polly as an analysis pass in LLVM

Utpal Bora[*], Johannes Doerfert[+], Tobias Grosser[$],
Venugopal Raghavan[£] and Ramakrishna Upadrasta[*]

IIT Hyderabad[*], Saarland University[+], ETH Zurich[$], AMD
India Pvt. Ltd.[£]

Goal

- Use precise dependence analysis of Polly in LLVM transformations



Example for vectorization

MultiSource/Benchmarks/TSVC/LinearDependence-flt

```
for (int i = 0; i < LEN2; i++) {
  for (int j = 0; j < i; j++) {
    aa[i][j] = aa[j][i] + bb[i][j];
  }
}
```

Loop Vectorizer **falsely** states memory dependence

Example for vectorization

MultiSource/Benchmarks/TSVC/LinearDependence-flt

```
for (int i = 0; i < LEN2; i++) {
   for (int j = 0; j < i; j++) {
     aa[i][j] = aa[j][i] + bb[i][j];
   }
}
```

No memory dependence because of j < i

Example for vectorization

MultiSource/Benchmarks/TSVC/LinearDependence-flt

```
for (int i = 0; i < LEN2; i++) {
   for (int j = 0; j < i; j++) {
     aa[i][j] = aa[j][i] + bb[i][j];
   }
}
```

No memory dependence because of j < i
Polly correctly determines no dependence

Example for vectorization

MultiSource/Benchmarks/TSVC/LinearDependence-flt

```
for (int i = 0; i < LEN2; i++) {
   for (int j = 0; j < i; j++) {
     aa[i][j] = aa[j][i] + bb[i][j];
   }
}
```

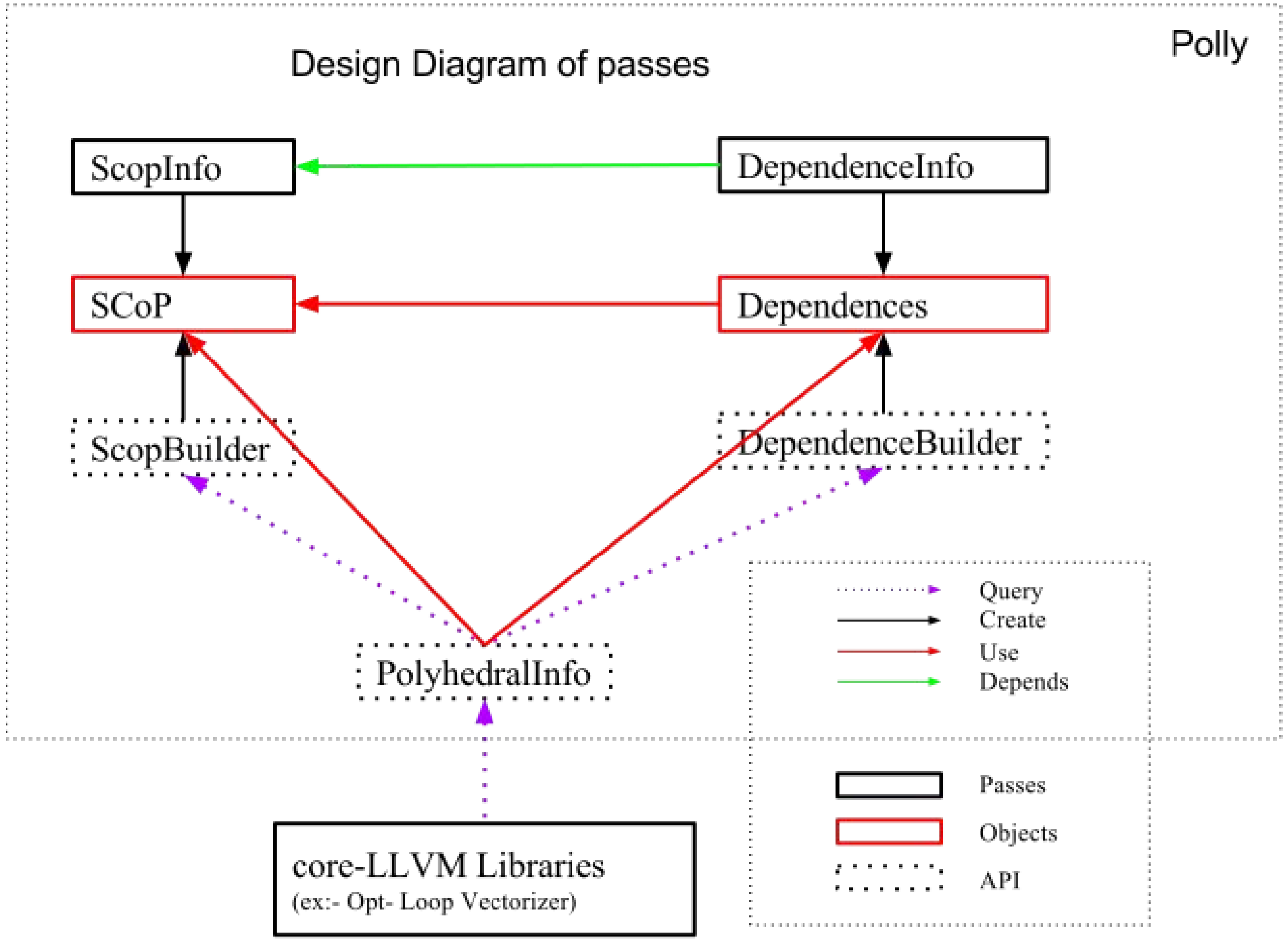
No memory dependence because of j < i
Polly correctly determines no dependence
Loop is parallel and vectorizable

This loop **can be vectorized** using **PolyhedralInfo**.

Implementation Detail

- **PolyhedralInfo-** a new interface to Polly
- APIs exposed for LLVM transformations
  - check loop is parallel: **isParallel**(Loop *L)
  - check vectorization legality: **isVectorizable**(Loop *L, unsigned int *VF)

VF - Vectorization Factor
We compute the maximum VF for the given loop if it is vectorizable. It is set to UINT_MAX for parallel loops


Polly
Design Diagram of passes
ScopInfo
DependenceInfo
SCoP
Dependences
ScopBuilder
DependenceBuilder
PolyhedralInfo
core-LLVM Libraries
(ex:- Opt- Loop Vectorizer)
Query
Create
Use
Depends
Passes
Objects
API

Checking loop parallelism

```
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    A[i] = 1;
```

```
$ opt -polly-process-unprofitable \
       -polyhedral-info \
       -polly-check-parallel \
       -analyze 1.ll
```

```
loop.i:     Loop is parallel.
loop.j:     Loop is not parallel.
```

Checking loop parallelism

```
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    A[j] = 1;
```

```
$ opt -polly-process-unprofitable \
        -polyhedral-info \
        -polly-check-parallel \
        -analyze 2.ll
```

```
loop.i:     Loop is not parallel.
loop.j:     Loop is parallel.
```

Checking loop vectorization legality

```
void f ( int *A, int N ) {
  for ( int j = 0; j < N; j++ )
    for ( int i = 0; i < N; i++ )
      A[i + 8] = A[i] + 1;
}
```

```
$ opt -polly-process-unprofitable \
        -polyhedral-info \
        -polly-check-vectorizable \
        -analyze 3.ll
```

```
loop.j:      Loop is not vectorizable
loop.i:      Loop is vectorizable with max VF = 8
```

Using PolyhedralInfo in LLVM

Include the header PolyhedralInfo

```
#include "polly/PolyhedralInfo.h"

void getAnalysisUsage(AnalysisUsage &AU) const override {
  AU.addRequired<PolyhedralInfo>();
  ...
}

bool runOnFunction(Function &F) override {
   auto *PHInfo = &getAnalysis<PolyhedralInfo>();
   auto IsParallel = PHInfo->isParallel(TheLoop);
   ...
   unsigned int VF = 0;
   auto IsVectorizable = PHInfo->isVectorizable(TheLoop, &VF);
   ...
}
```

Future Work

- Derive runtime checks in LLVM for assumptions in Polly
- Modeling dependences at instruction granularity
- Parametric dependence distances
- Demand driven computation of dependences

Thank You!