Polly
Polyhedral Transformations for LLVM

Tobias Grosser - Hongbin Zheng

November 4, 2010

Outline

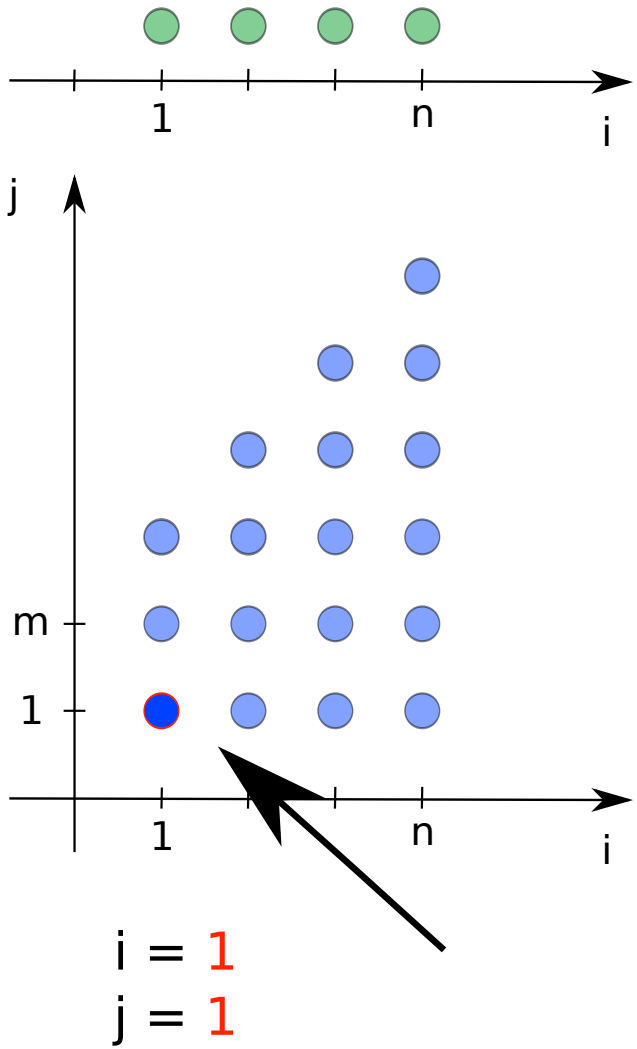
The SCoP - Static Control Part

```
for i = 1 to (5n + 3)
    for j = n to (4i + 3n + 4)
        A[i-j] = A[i]
    if i < (n - 20)
        A[i+20] = j
```

- Structured control flow
  - Regular for loops
  - Conditions
- Affine expressions in induction variables and parameters for:
  - Loop bounds, conditions, access functions
- Side effect free

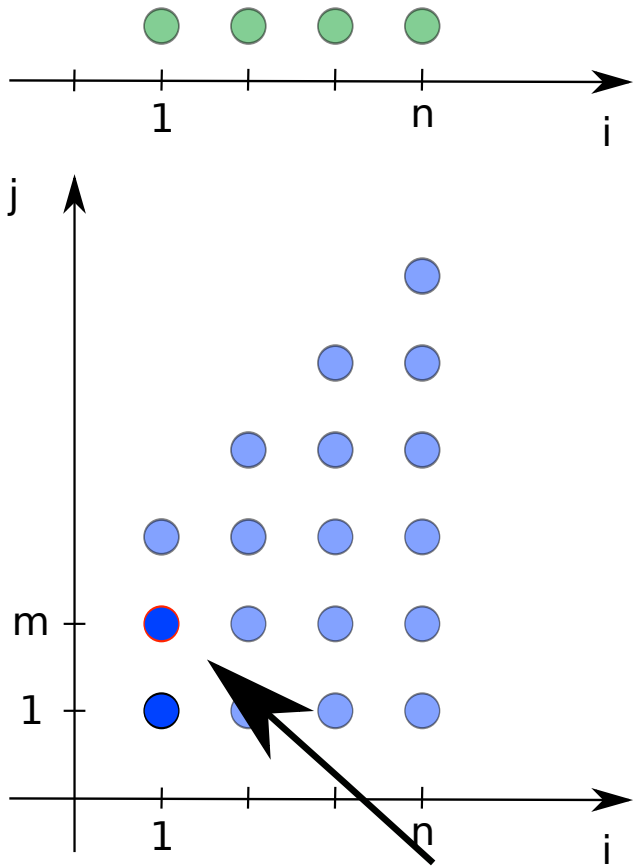
The Iteration Domain / Schedule

```
for (i = 1; i <= n; i++) {

  for (j = 1; j <= i + m; j++)
    A[i][j] = A[i-1][j] + A[i][j-1]

  A[i][i+m+1] = A[i-1][i+m] + A[i][i+m]
}
```



The Iteration Domain / Schedule

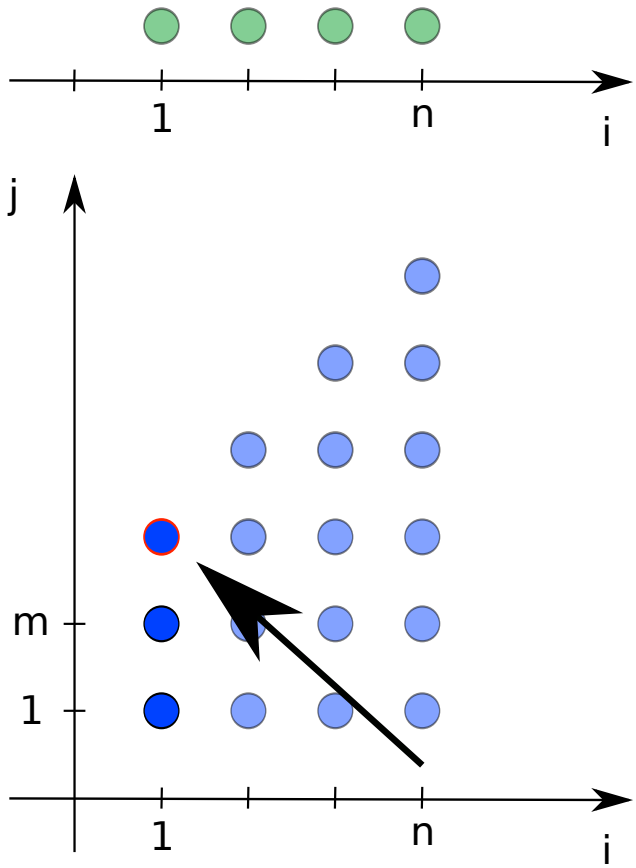
```
for (i = 1; i <= n; i++) {

  for (j = 1; j <= i + m; j++)
    A[i][j] = A[i-1][j] + A[i][j-1]

  A[i][i+m+1] = A[i-1][i+m] + A[i][i+m]
}
```



i = 1
j = 2 = m

The Iteration Domain / Schedule

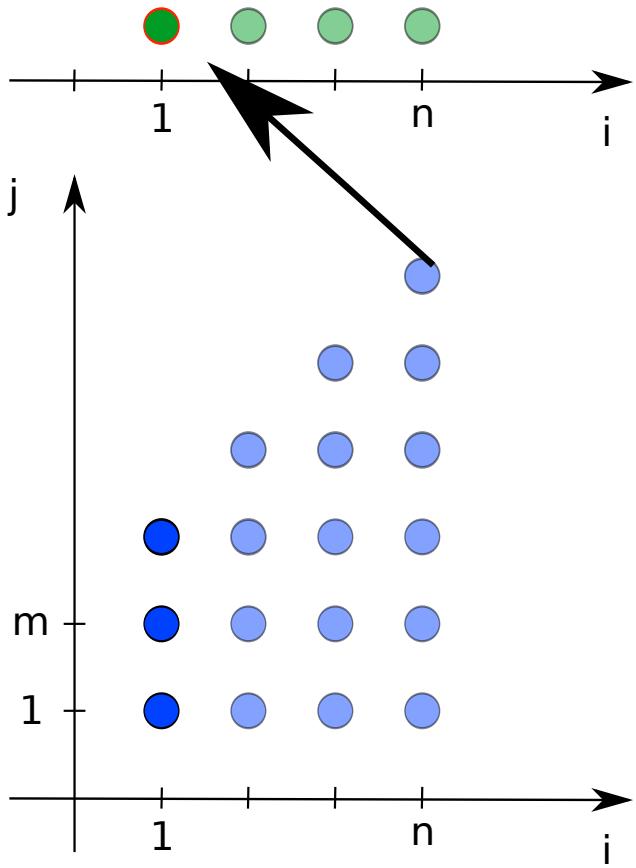
```
for (i = 1; i <= n; i++) {

  for (j = 1; j <= i + m; j++)
    A[i][j] = A[i-1][j] + A[i][j-1]

  A[i][i+m+1] = A[i-1][i+m] + A[i][i+m]
}
```



i = 1
j = 3 = (i + m)

The Iteration Domain / Schedule

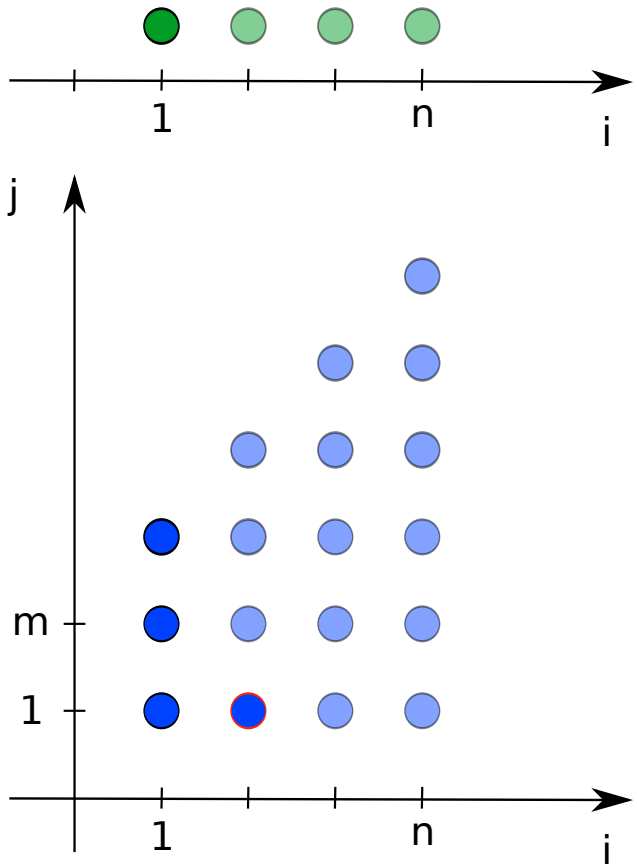
```
for (i = 1; i <= n; i++) {

  for (j = 1; j <= i + m; j++)
    A[i][j] = A[i-1][j] + A[i][j-1]

  A[i][i+m+1] = A[i-1][i+m] + A[i][i+m]
}
```



i = 1
j = n/a

The Iteration Domain / Schedule

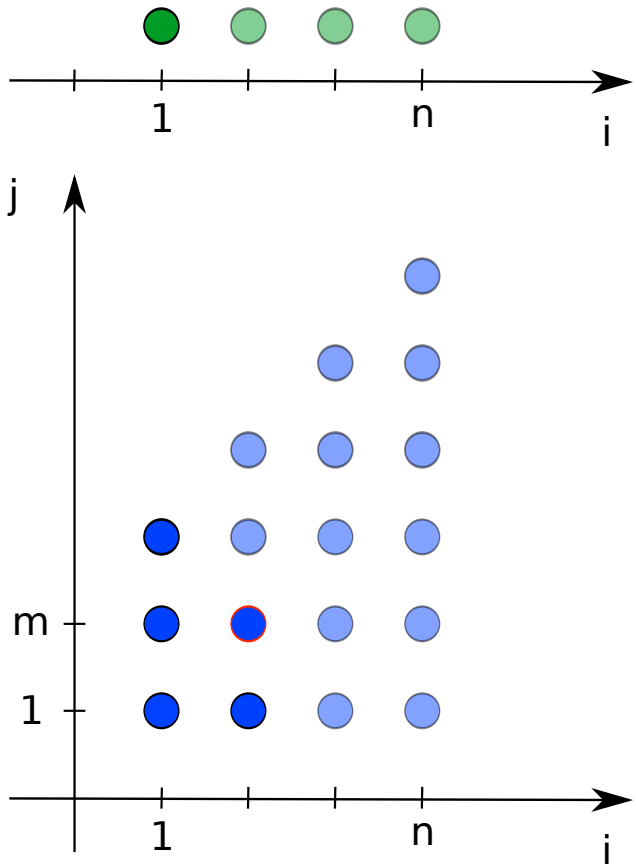
```
for (i = 1; i <= n; i++) {

  for (j = 1; j <= i + m; j++)
    A[i][j] = A[i-1][j] + A[i][j-1]

  A[i][i+m+1] = A[i-1][i+m] + A[i][i+m]
}
```



i = 2
j = 1

The Iteration Domain / Schedule

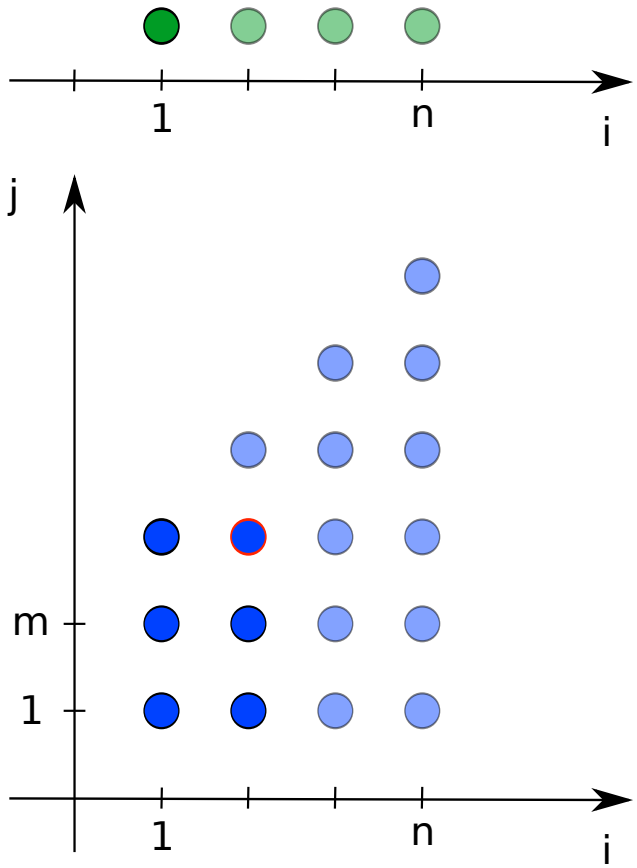
```
for (i = 1; i <= n; i++) {

  for (j = 1; j <= i + m; j++)
    A[i][j] = A[i-1][j] + A[i][j-1]

  A[i][i+m+1] = A[i-1][i+m] + A[i][i+m]
}
```



i = 2
j = 2

The Iteration Domain / Schedule

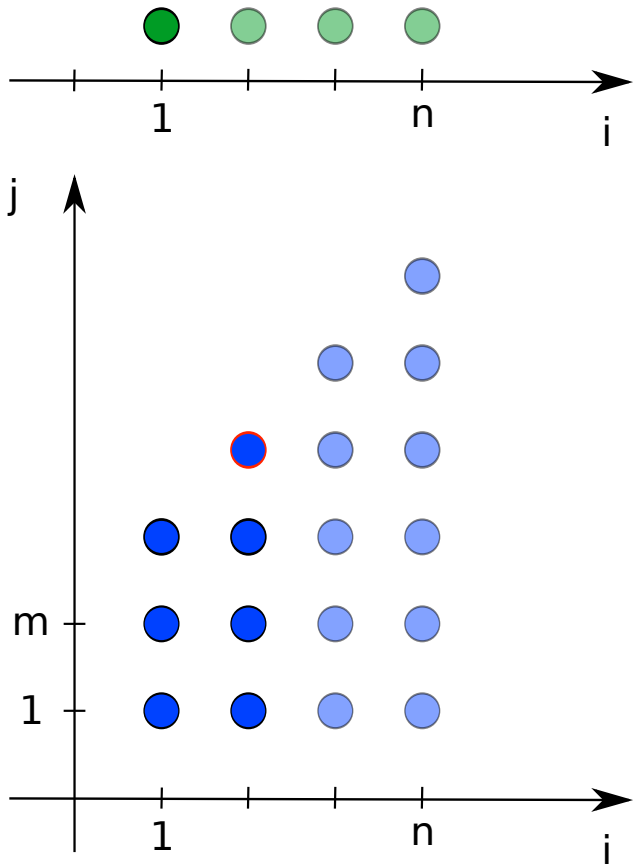
```
for (i = 1; i <= n; i++) {

  for (j = 1; j <= i + m; j++)
    A[i][j] = A[i-1][j] + A[i][j-1]

  A[i][i+m+1] = A[i-1][i+m] + A[i][i+m]
}
```



i = 2
j = 3

The Iteration Domain / Schedule

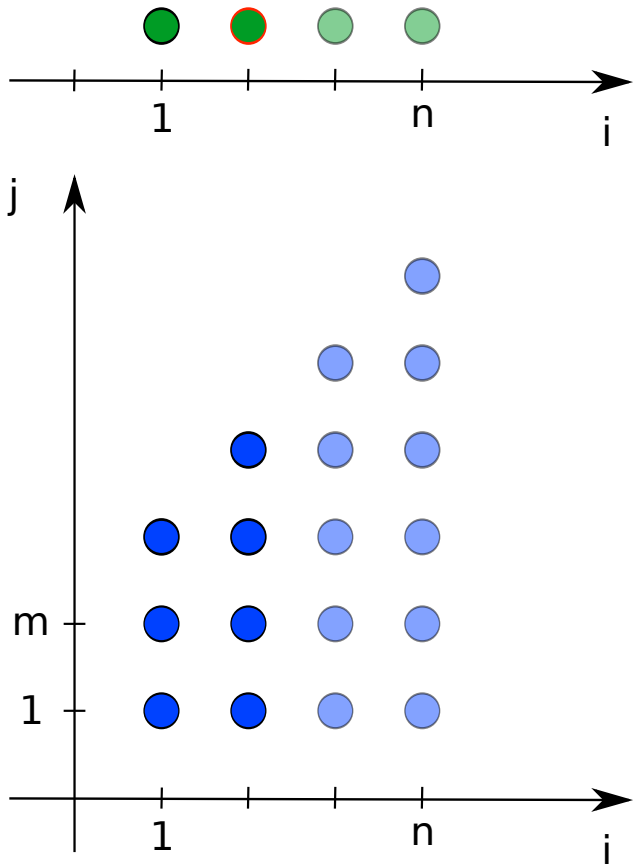
```
for (i = 1; i <= n; i++) {

  for (j = 1; j <= i + m; j++)
    A[i][j] = A[i-1][j] + A[i][j-1]

  A[i][i+m+1] = A[i-1][i+m] + A[i][i+m]
}
```



i = 2
j = 4 = (i + m)

The Iteration Domain / Schedule

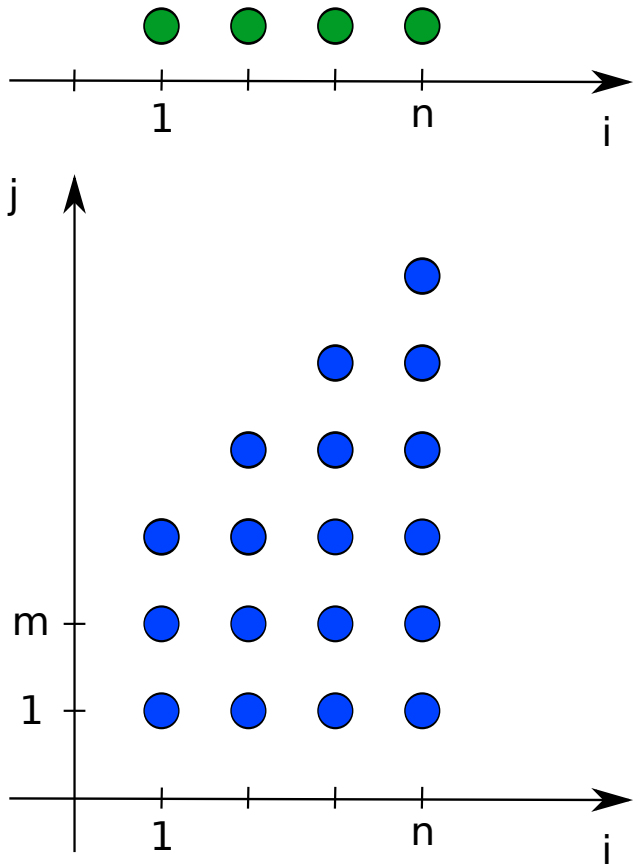
```
for (i = 1; i <= n; i++) {

  for (j = 1; j <= i + m; j++)
    A[i][j] = A[i-1][j] + A[i][j-1]

  A[i][i+m+1] = A[i-1][i+m] + A[i][i+m]
}
```



i = 2
j = n/a

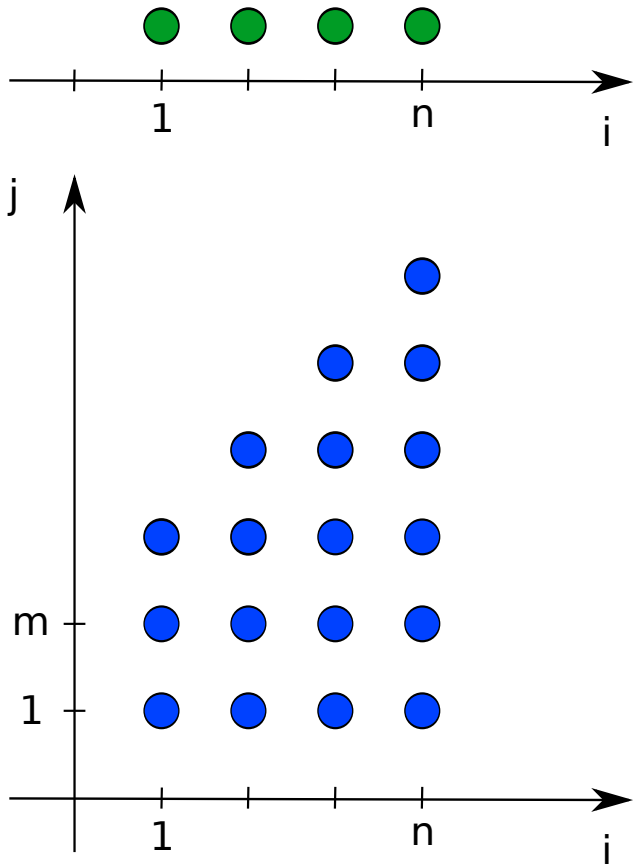
The Iteration Domain / Schedule

```
for (i = 1; i <= n; i++) {

  for (j = 1; j <= i + m; j++)
    A[i][j] = A[i-1][j] + A[i][j-1]

  A[i][i+m+1] = A[i-1][i+m] + A[i][i+m]
}
```



The Iteration Domain / Schedule

```
for (i = 1; i <= n; i++) {

  for (j = 1; j <= i + m; j++)
    A[i][j] = A[i-1][j] + A[i][j-1]

  A[i][i+m+1] = A[i-1][i+m] + A[i][i+m]
}
```



Domains

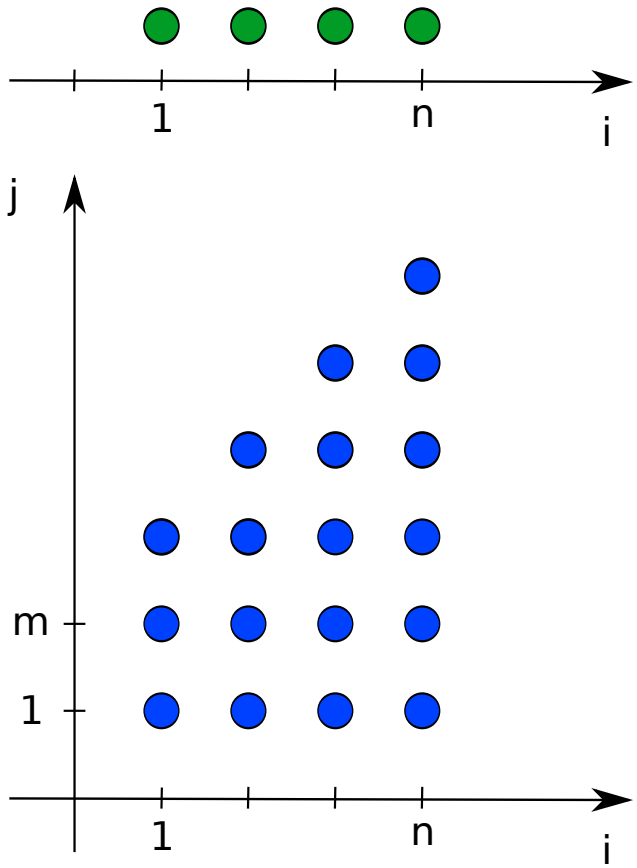
$1 \leq i \leq n$

$1 \leq j \leq i + m$

$1 \leq i \leq n$

The Iteration Domain / Schedule

```
for (i = 1; i <= n; i++) {

  for (j = 1; j <= i + m; j++)
    A[i][j] = A[i-1][j] + A[i][j-1]

  A[i][i+m+1] = A[i-1][i+m] + A[i][i+m]
}
```



Domains

$1 \leq i \leq n$

$1 \leq j \leq i + m$

$1 \leq i \leq n$

Schedules

$s_1 = i$

$s_2 = 0$

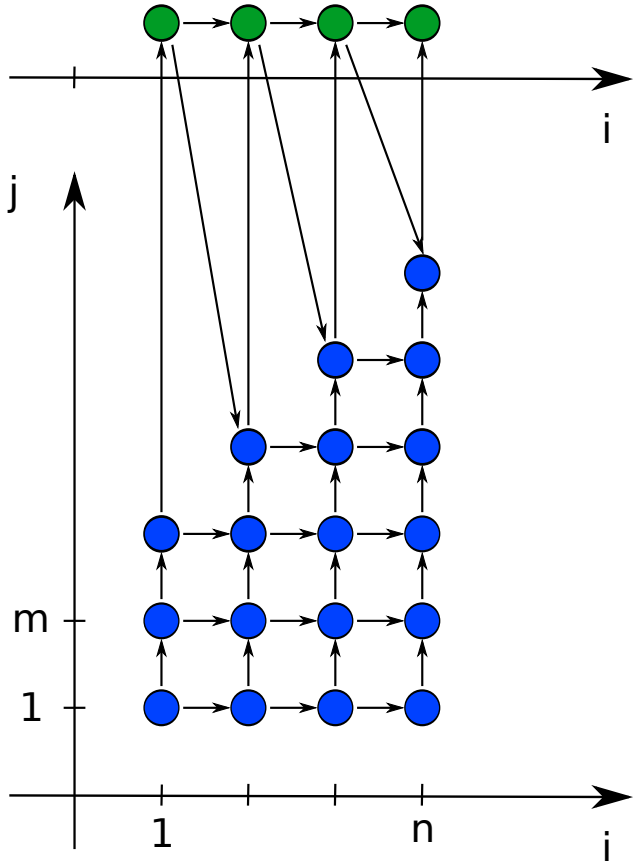
$s_3 = j$

$s_1 = i$

$s_2 = 1$

The Iteration Domain / Schedule

```
for (i = 1; i <= n; i++) {

  for (j = 1; j <= i + m; j++)
    A[i][j] = A[i-1][j] + A[i][j-1]

  A[i][i+m+1] = A[i-1][i+m] + A[i][i+m]
}
```



Domains

$1 \leq i \leq n$

$1 \leq j \leq i + m$

$1 \leq i \leq n$

Schedules

$s_1 = i$

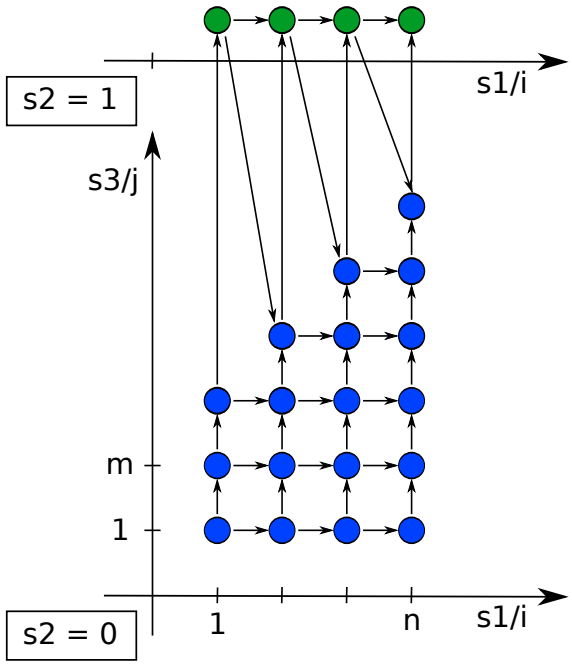
$s_2 = 0$

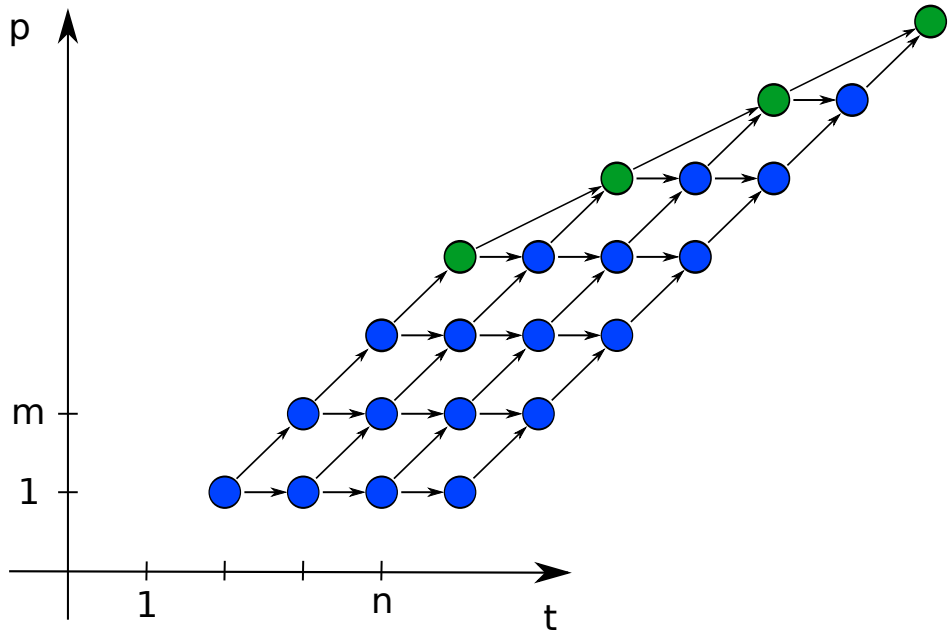
$s_3 = j$

$s_1 = i$

$s_2 = 1$

Scheduling Transformation





Original Schedules

$s_1 = i$ $\quad$ $s_1 = i$

$s_2 = 0$ $\quad$ $s_2 = 1$

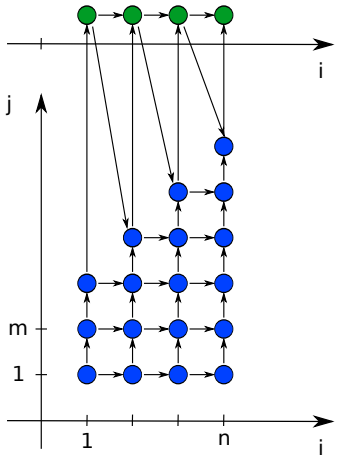
$s_3 = j$

Transformed Schedules

$p = \quad j$ $\quad$ $p = \quad i + m + 1$

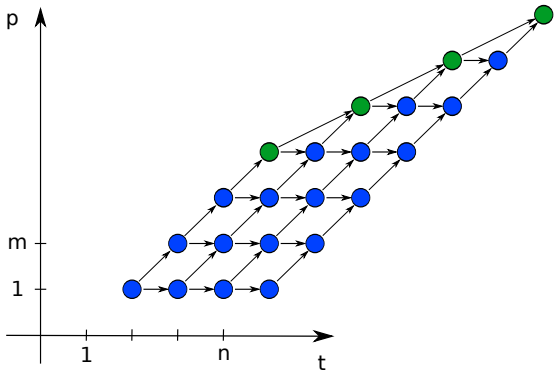
$t = i + j$ $\quad$ $t = 2i + m + 1$

Code Generation



Original Code

```
for (i = 1; i <= n; i++) {

  for (j = 1; j <= i + m; j++)
    A[i][j] = A[i-1][j] + A[i][j-1]

  A[i][i+m+1] = A[i-1][i+m] + A[i][i+m]
}
```



Transformed Code

```
parfor (p = 1; p <= m+n+1; p++) {
  if (p >= m+2)
    A[p-m-1][p] = A[p-m-2][p-1]
  for (t = max(p+1, 2*p-m); t <= p+n; t++)
    A[-p+t][p] = A[-p+t-1][p] + A[-p+t][p-1]
}
```

Outline

Source to source frameworks

Suif | Omega | LooPo | Pluto | ...

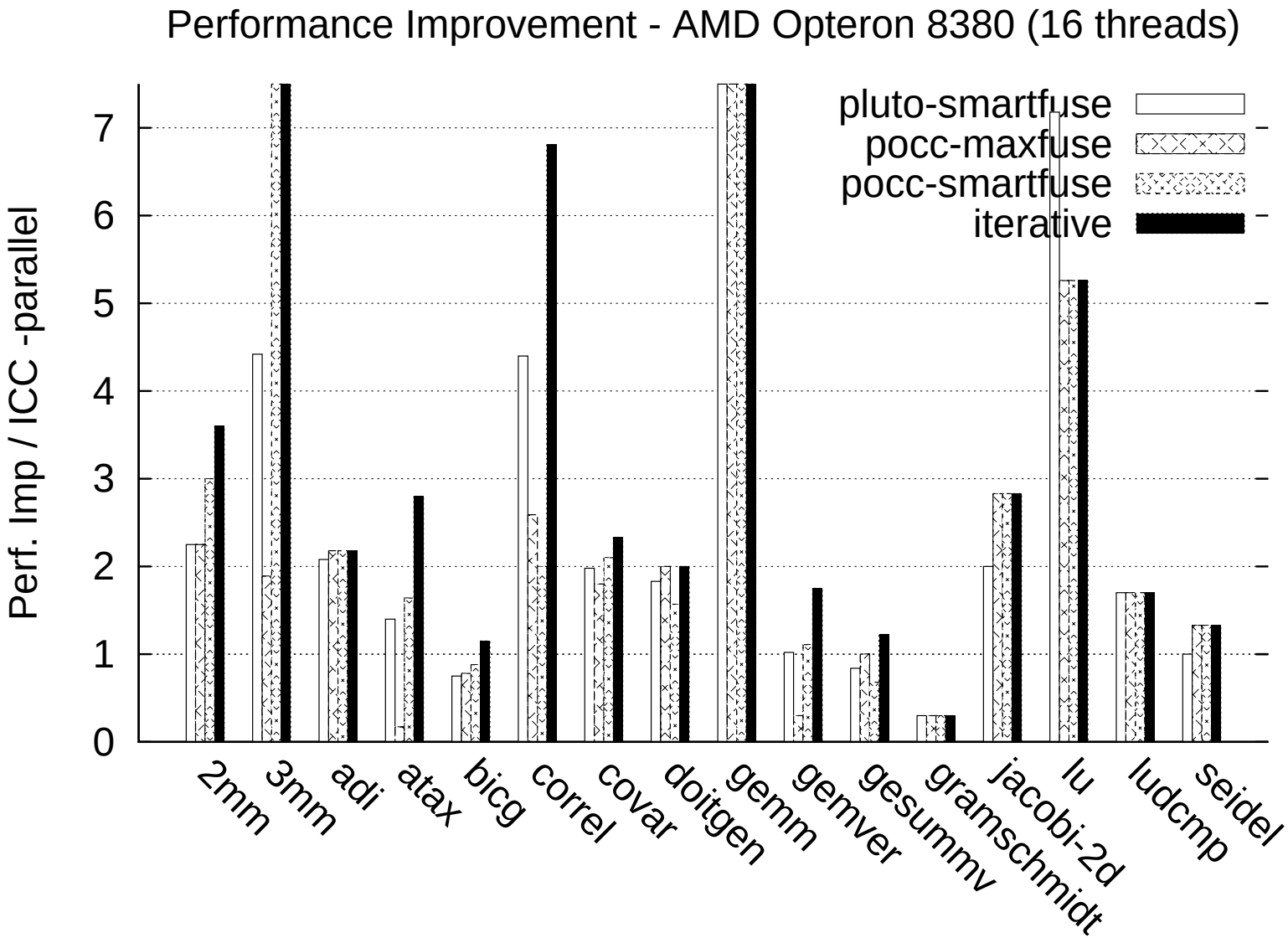
- More than 20 years of research
- Work on
  - Tiling / Parallelization / Prevectorization
  - CUDA / GPGPU
  - Coarse grain parallelism / Grid computing
- Advanced algorithms / Proven performance

Speed comparison

Compile mode	GCC 4.5.1	clang 2.8	ICC 11.1
-O3	1m 22.0s	1m 22.0s	0m 5.8s
pluto-tiled -O3	0m 6.1s	0m 5.8s	0m 2.5s

Table: Matrix multiplication on Intel i5 M 520 - *double* 2048 *x* 2048

PoCC - SC10


Performance Improvement - AMD Opteron 8380 (16 threads)
Perf. Imp / ICC -parallel
0
1
2
3
4
5
6
7
pluto-smartfuse
pocc-maxfuse
pocc-smartfuse
iterative
2mm
3mm
adi
atax
bicg
correl
covar
doitgen
gemm
gemver
gesummv
gramschmidt
jacobi-2d
lu
ludcmp
seidel

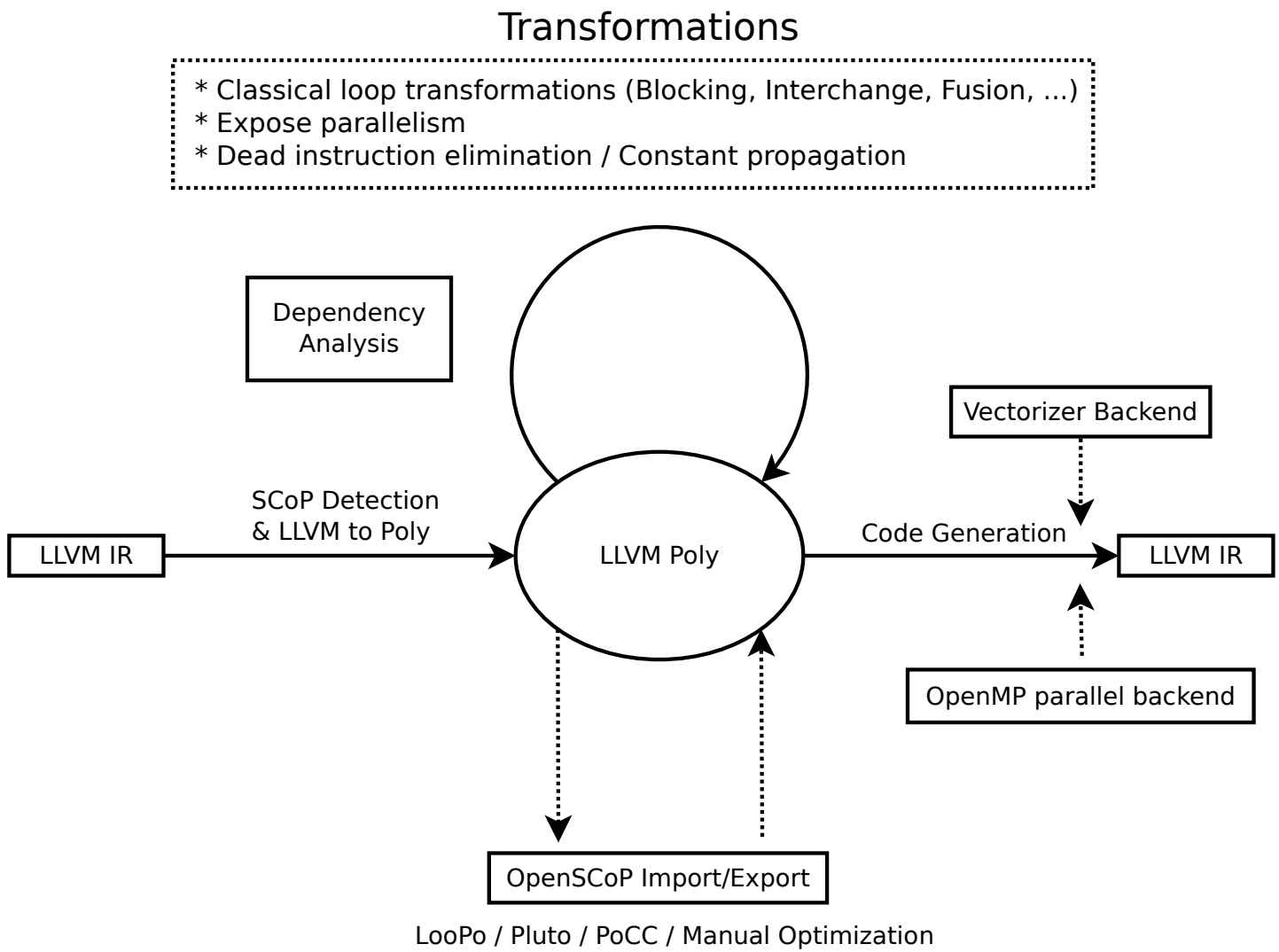
Source to source frameworks

Restrictions

- Limited to subset of C/C++
- Require annotated C code
- Only canonical code
- Correct? (Integer overflow, Operator overloading, ...)

Outline

The Architecture



Why optimize on LLVM-IR?

- Frontend independent
- Fully automatic
- High SCoP coverage
- Integration with vectorizer/ OpenMP code generator

The SCoP - Revised

Thanks to

- Scalar evolution
- Loop/Region detection
- LLVM canonicalization passes

SCoP - The LLVM way

- ~~Limited Scalars~~ Any scalars
- Structured control flow
  - ~~Regular for loops~~ Anything that acts like a regular for loop
  - Conditions
- ~~Affine expressions~~ Expressions that calculate an affine result
- Side effect ~~free~~ known
- ~~Memory accesses only through arrays~~

Valid SCoPs

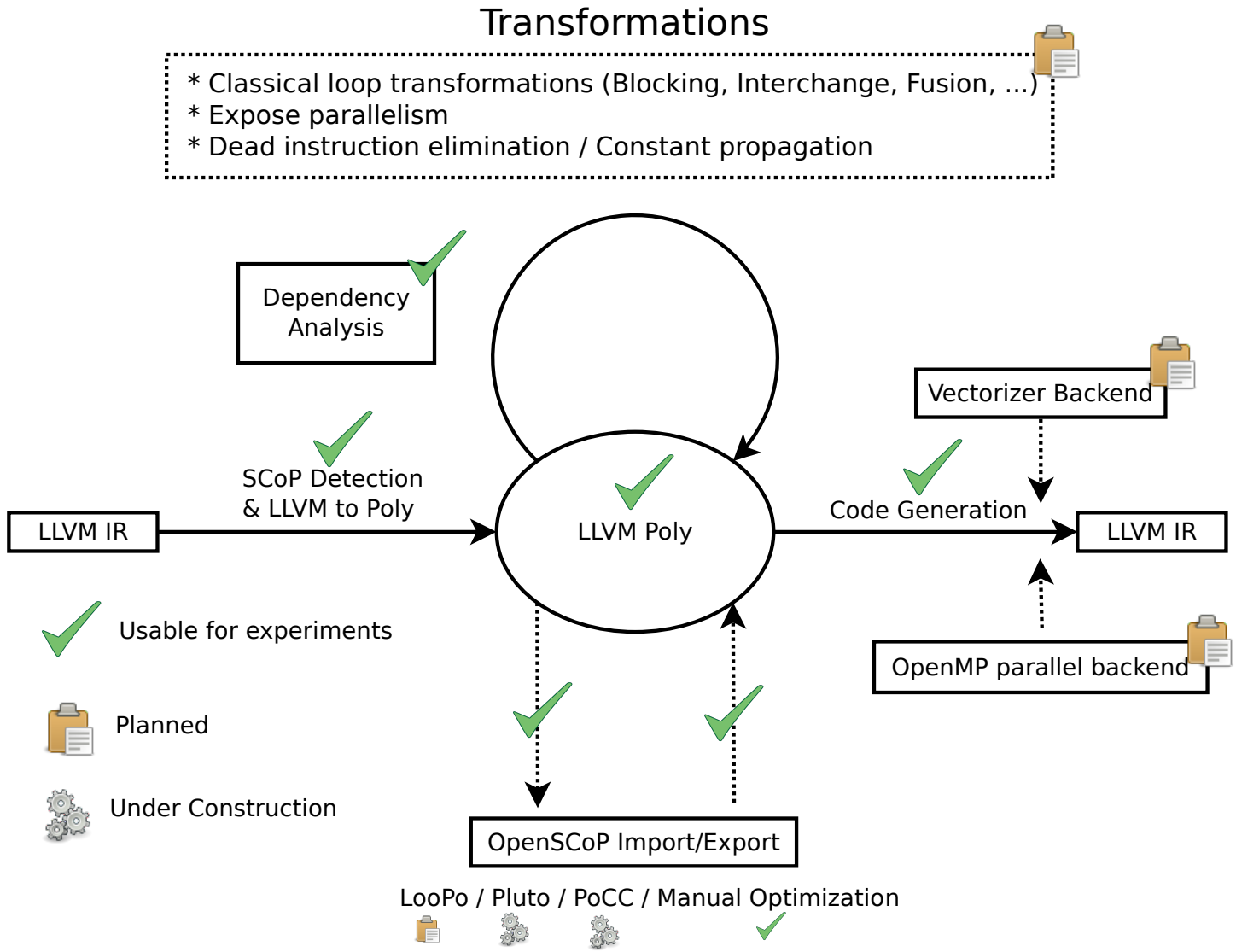
do..while loop

```
i = 0;

do {
    int b = 2 * i;
    int c = b * 3 + 5 * i;
    A[c] = i;
    i += 2;
} while (i < N);
```

pointer loop

```
int A[1024];

void pointer_loop () {
    int *B = A;
    while (B < &A[1024]) {
        *B = 1;
        ++B;
    }
}
```

The Architecture



Open Topics

- Multi dimensional arrays - Delinearization
- Integer modulo arithmetics
- Optimal type selection

Open Topics

- Multi dimensional arrays - Delinearization
- Integer modulo arithmetics
- Optimal type selection

Polly alone is not yet improving performance of your code and may even apply incorrect transformations on the LLVM-IR.

Thanks to Qualcomm for sponsoring this talk.

Thank you.
Any Questions?

http://wiki.llvm.org/Polly