Loopy: Programmable and Formally Verified Loop Transformations

Kedar Namjoshi
Bell Labs, Nokia

Nimit Singhania
University of Pennsylvania

Static Analysis Symposium 2016, Edinburgh

Problem

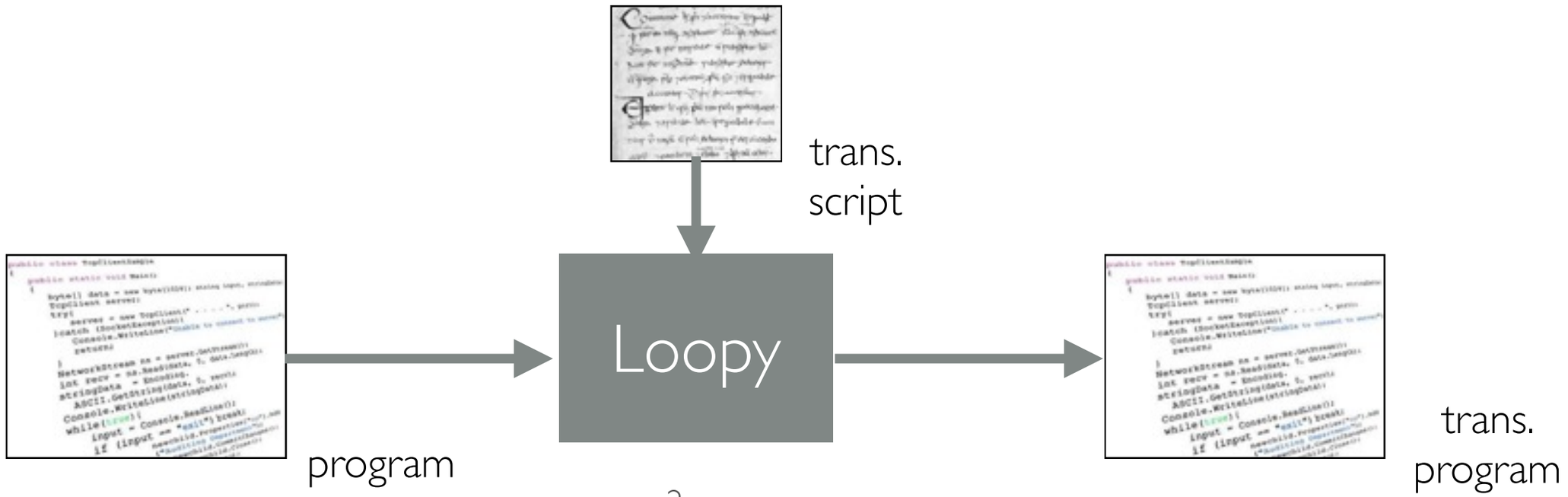
How to optimize loops in a program?

Optimizing Compilers?

Variable performance gains

Manual Rewriting?

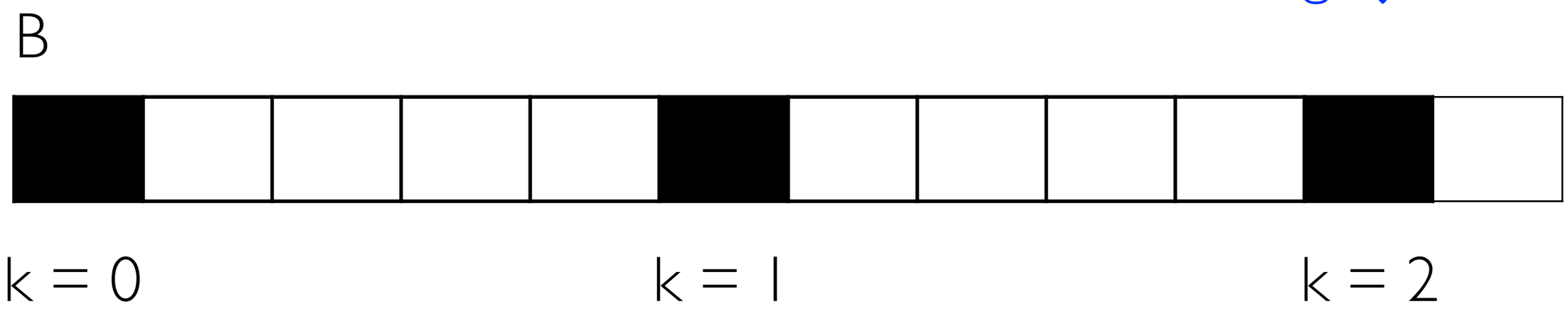
Tedious and Error Prone



Example

```
for (i = 0; i < N; i++) {
  for (j = 0; j < N; j++) {
    {
      Init: C[i][j] = 0;
    }
    for (k = 0; k < N; k++) {
      Mult:
        C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

Interchange **j** and **k**



Operation 1: Loop Splitting

```
for (i = 0; i < N; i++) {
  for (j = 0; j < N; j++) {
      Init: C[i][j] = 0;
  }
}
for (i = 0; i < N; i++) {
  for (j = 0; j < N; j++) {
    for (k = 0; k < N; k++) {
      Mult:
        C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

realign(Init, Mult, 0)

Operation 2: Loop Interchange

```
for (i = 0; i < N; i++) {
  for (j = 0; j < N; j++) {
      Init: C[i][j] = 0;
  }
}
for (i = 0; i < N; i++) {
  for (k = 0; k < N; k++) {
    for (j = 0; j < N; j++) {
      Mult:
        C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

5x speedup!

```
affine(Mult, {[i, j, k] -> [i, k, j]})
```

Operation 3: Loop Tiling

```
for (i = 0; i < N; i++) {
  for (j = 0; j < N; j++) {
    Init: C[i][j] = 0;
  }
}
for (i1 = 0; i1 < N/32; i1++) {
  for (k1 = 0; k1 < N/32; k1++) {
    for (j1 = 0; j1 < N/32; j1++) {
  for (i2 = 0; i2 < min(32, N - i1*32); i2++) {
    for (k2 = 0; k2 < min(32, N - k1*32); k2++) {
      for (j2 = 0; j2 < min(32, N - j1*32); j2++) {
        Mult:
          C[i1*32+i2][j1*32+j2] += A[i1*32+i2][k1*32+k2]
                                 * B[k1*32+k2][j1*32+j2];
    }}
  }}
}}
```

2x speedup!

```
affine(Mult, {[i, j, k] -> [i1, j1, k1, i2, j2, k2]:
                           i1 = [i/32] and i2 = i%32 and
                           j1 = [j/32] and j2 = j%32 and
                           k1 = [k/32] and k2 = k%32})
```

Transformation Script

```
realign(Init, Mult, 0)

★ affine(Mult, {[i, j, k] -> [i, k, j]})

affine(Mult, {[i, j, k] ->
  [i1, j1, k1, i2, j2, k2]:
    i1 = [i/32] and i2 = i%32 and
    j1 = [j/32] and j2 = j%32 and
    k1 = [k/32] and k2 = k%32})
```

★ Not implemented by Polly

Loopy

```
realign(Init, Mult, 0)

affine(Mult, {[i, j, k] -> [i, k, j]})

affine(Mult, {[i, j, k] ->
          [i1, j1, k1, i2, j2, k2]:
           i1 = [i/32] and i2 = i%32 and
           j1 = [j/32] and j2 = j%32 and
           k1 = [k/32] and k2 = k%32})
```

```
for (i = 0; i < N; i++) {
  for (j = 0; j < N; j++) {
    {
      Init: C[i][j] = 0;
    }
    for (k = 0; k < N; k++) {
      Mult:
        C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

Loopy

```
for (i = 0; i < N; i++) {
  for (j = 0; j < N; j++) {
    Init: C[i][j] = 0;
  }
}

for (i1 = 0; i1 < N/32; i1++) {
  for (k1 = 0; k1 < N/32; k1++) {
    for (j1 = 0; j1 < N/32; j1++) {
    for (i2 = 0; i2 < min(32, N - i1*32); i2++) {
      for (k2 = 0; k2 < min(32, N - k1*32); k2++) {
        for (j2 = 0; j2 < min(32, N - j1*32); j2++) {
          Mult:
            C[i1*32+i2][j1*32+j2] += A[i1*32+i2][k1*32+k2]
                                   * B[k1*32+k2][j1*32+j2];
      }}
  }}
}}
```

Verify
Equivalence

Other Scripting Approaches

- CHiLL, POET, Orio, X Language, …
  - No correctness guarantees
- URUK
  - Transformations specified on internal program representation
  - Difficult for a programmer to describe
- Loopy
  - **Source-level** transformations with **correctness** guarantees

Loop Components

```
for (i = 0; i < N; i++) {
  for (j = 0; j < N; j++) {
    {
      L1: C[i][j] = 0;
    }
    for (k = 0; k < N; k++) {
      L2:
        C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

Program

```
op1(L1, L2, 0)
L3 = op2(L1, f1)
(L4, L5) = op4(L3, p)
op3(L4, f2)
op3(L5, k)
```

Script

Supported Operations

realign(L1, L2, n): Splits or merges adjacent loops to have **n** common loops

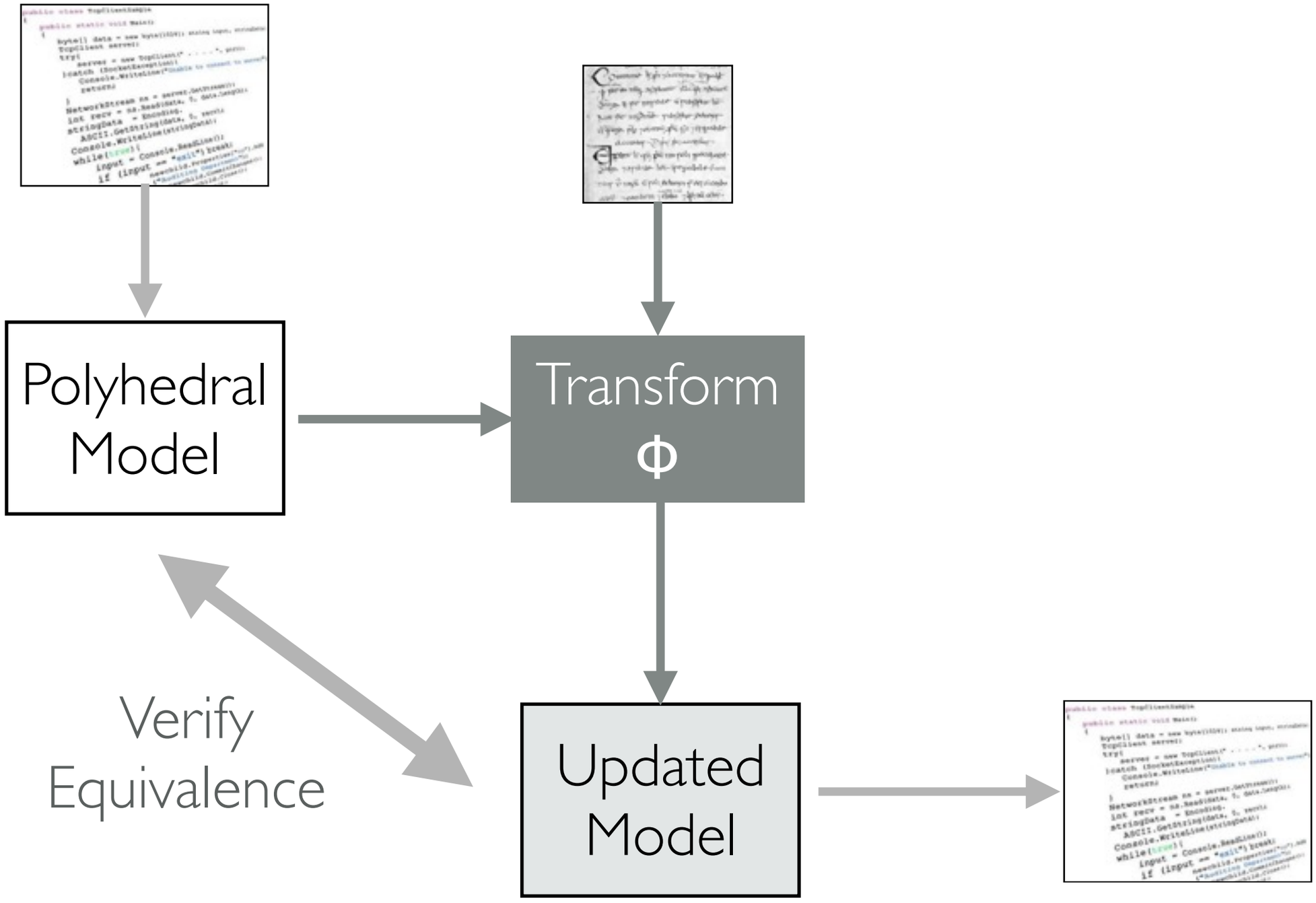
affine(L, f): Applies affine transformation **f** on L

(L1, L2) = **isplit**(L, p, n): Splits L into L1 and L2 with index sets satisfying **p** and **~p**

L1 = **lift**(L, n): Returns handle L1 to **n**th loop of L

Piecewise Affine Reordering

Internal Workflow

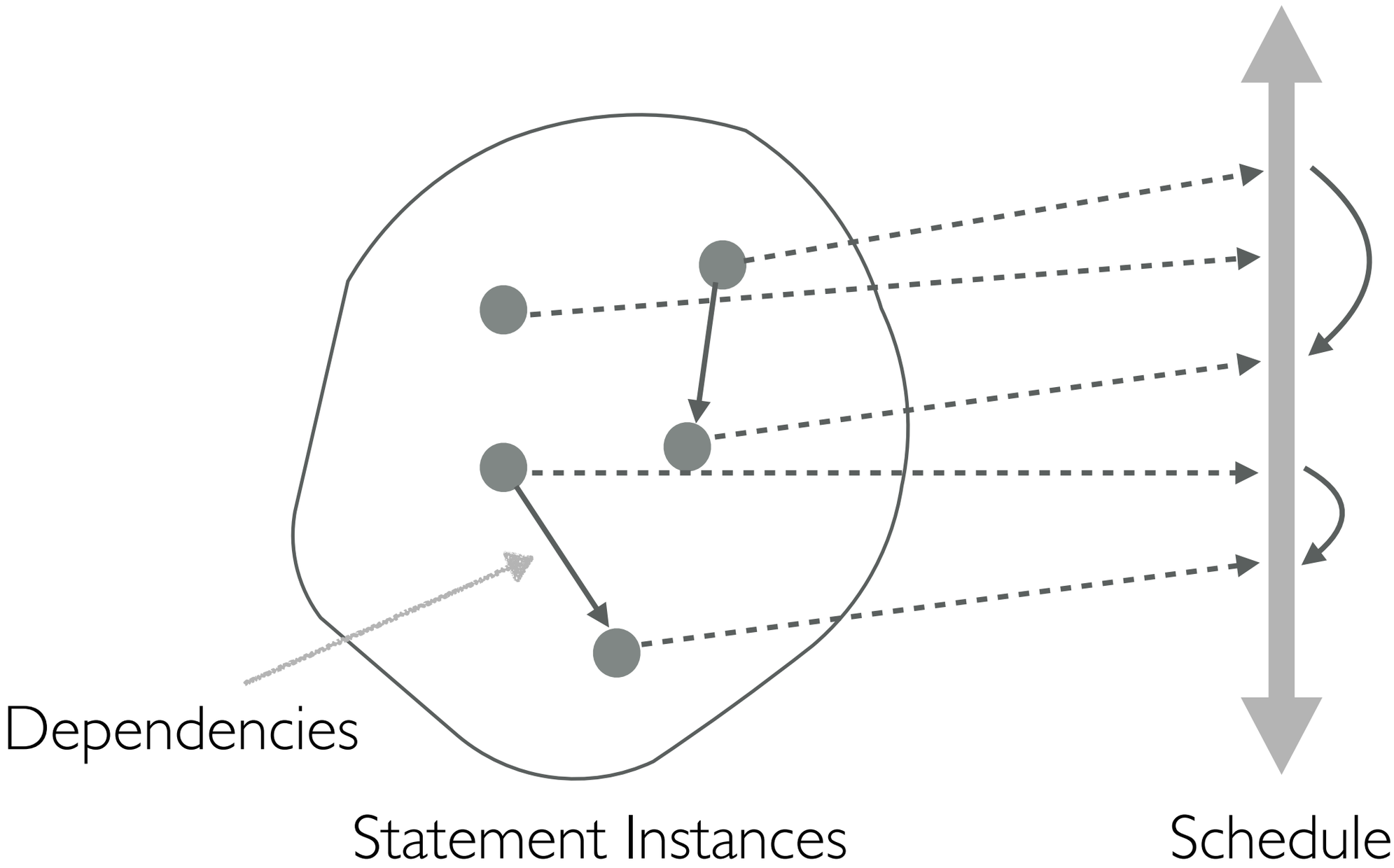


Polyhedral Model

```
for (i = 0; i < N; i++) {
  for (j = 0; j < N; j++) {
    {
      Init: C[i][j] = 0;          [0, 0, 0, 0]
    }
    for (k = 0; k < N; k++) {
      Mult:
        C[i][j] += A[i][k] * B[k][j];
                                  [0, 0, 1, 0]
    }
  }
}
```

- Statement Instances:
  {Init[i, j], Mult[i, j, k]}
- Iteration Domain:
  $0 \leq i < N \wedge 0 \leq j < N \wedge 0 \leq k < N$
- Dependencies:
  $Init[i, j] \longrightarrow Mult[i, j, k]$
  $Mult[i, j, k] \longrightarrow Mult[i, j, k+1]$
- Schedule:
  Init[i, j] : (**0**, i, **0**, j, **0**, 0, **0**)
  Mult[i, j, k] : (**0**, i, **0**, j, **1**, k, **0**)

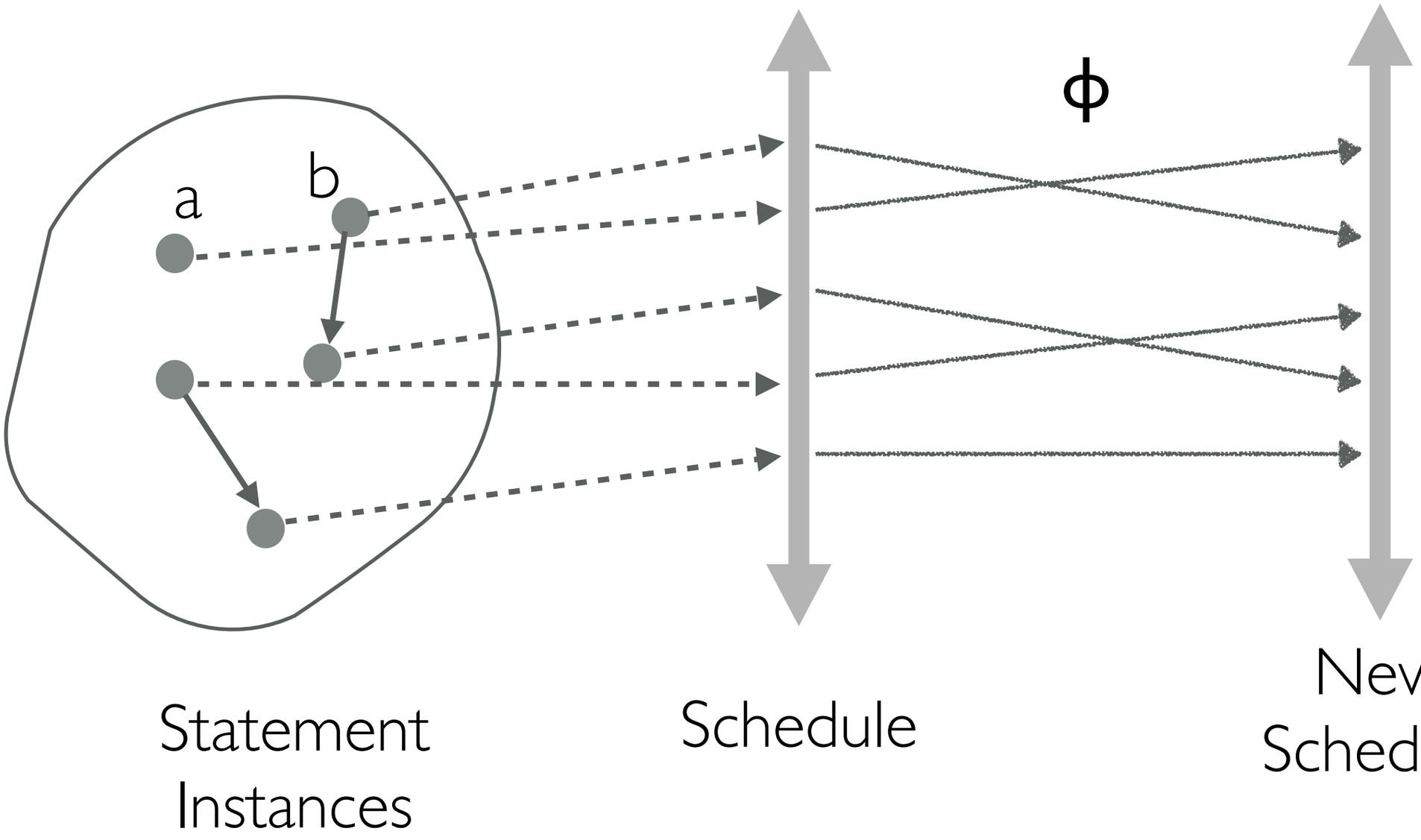
Polyhedral Model



Transform Function

- Transformation of the schedule
  - Re-ordering execution of statements
- Two types of transformations
  - Positional changes - e.g. loop splitting
  - Iterator changes - e.g. loop reversal
- $\phi = \phi_m \circ \phi_{m-1} \circ \ldots \circ \phi_1$

Transform Function



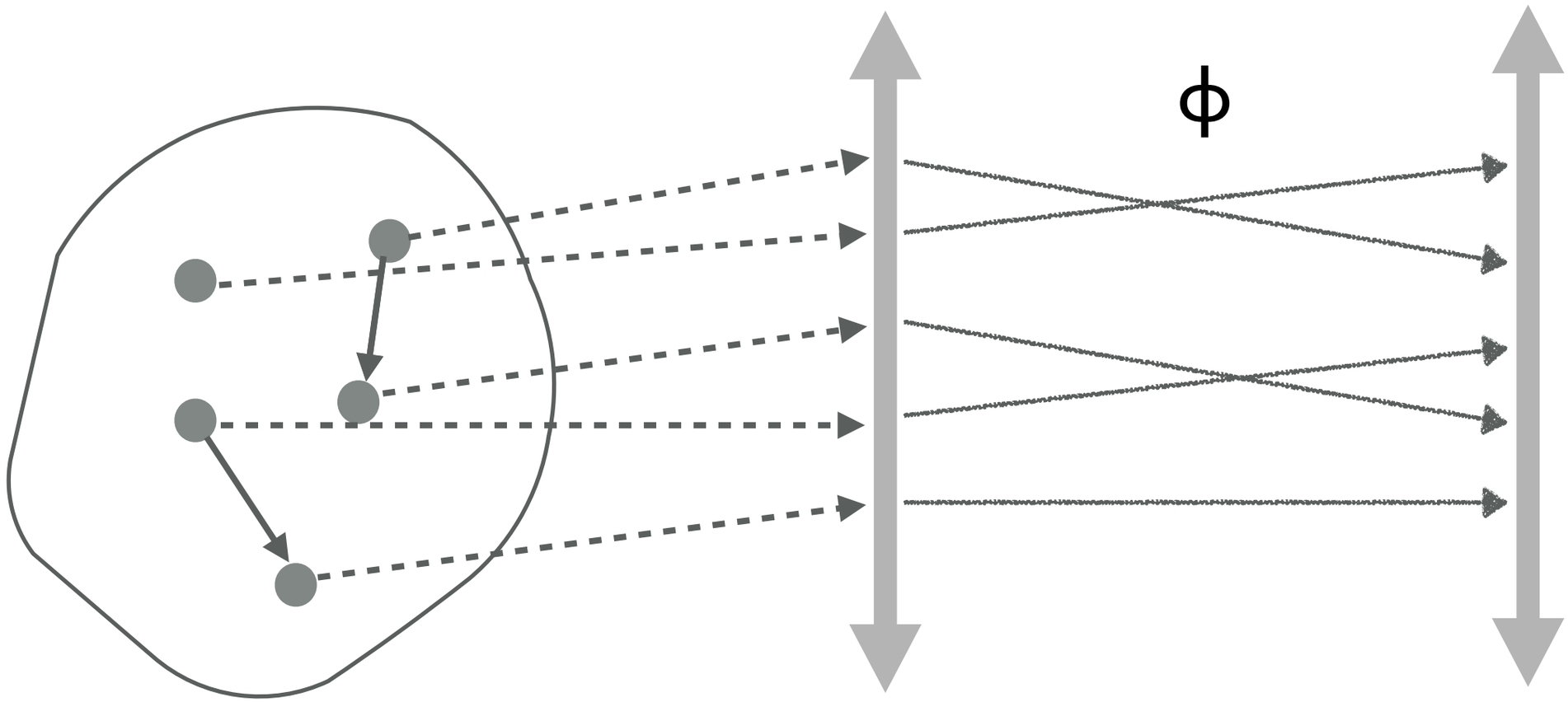
Verification

- Composed transformation ϕ must be one-one
- For each dependency ($s \longrightarrow t$), execution order must be preserved:

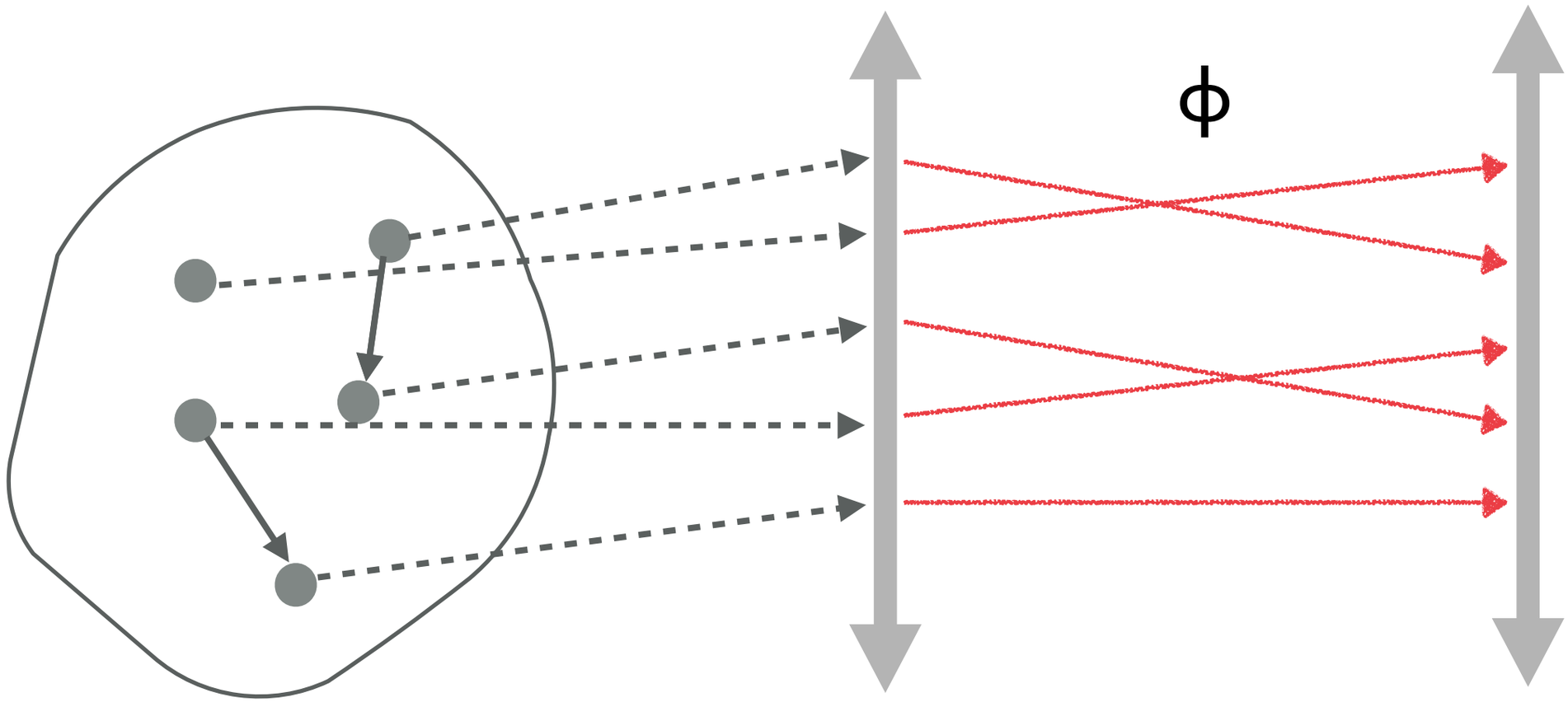
$$\phi(\text{schedule}(s)) < \phi(\text{schedule}(t))$$

- Checked using an Integer Constraint Solver
  - Counterexample generated if incorrect
- Verification for the composed transformation ϕ only
  - Dependencies might be broken in between

Verification

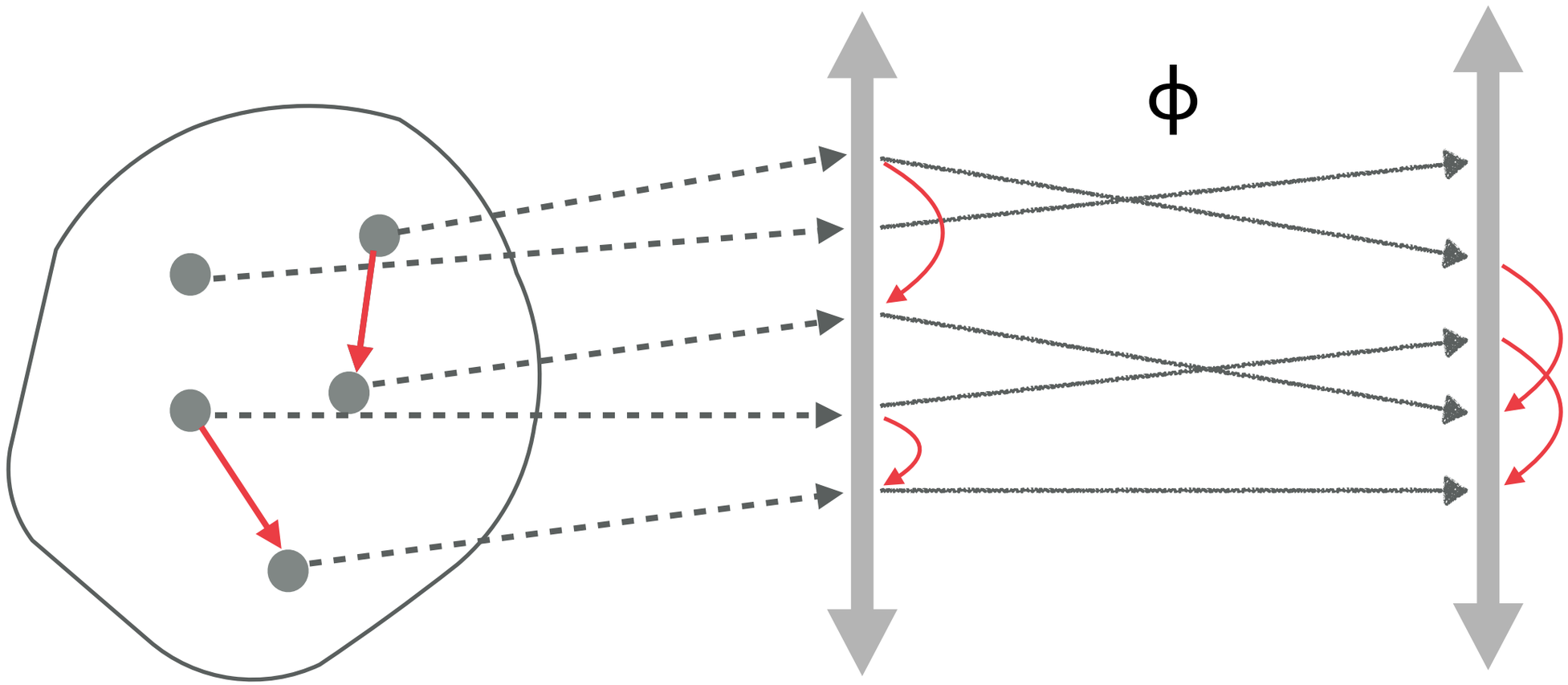


Verification



One-to-one

Verification

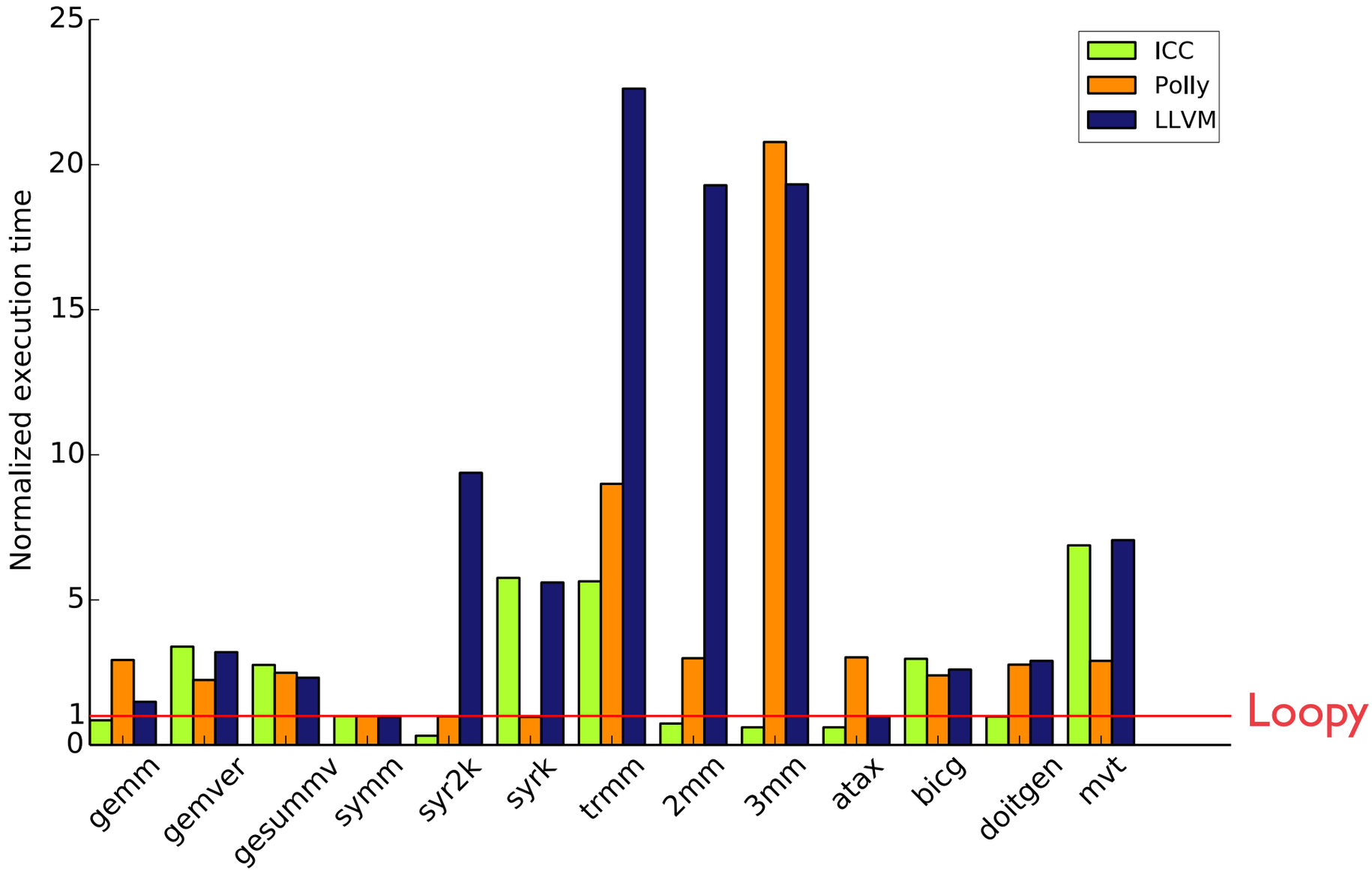


Dependencies must be preserved

Evaluation

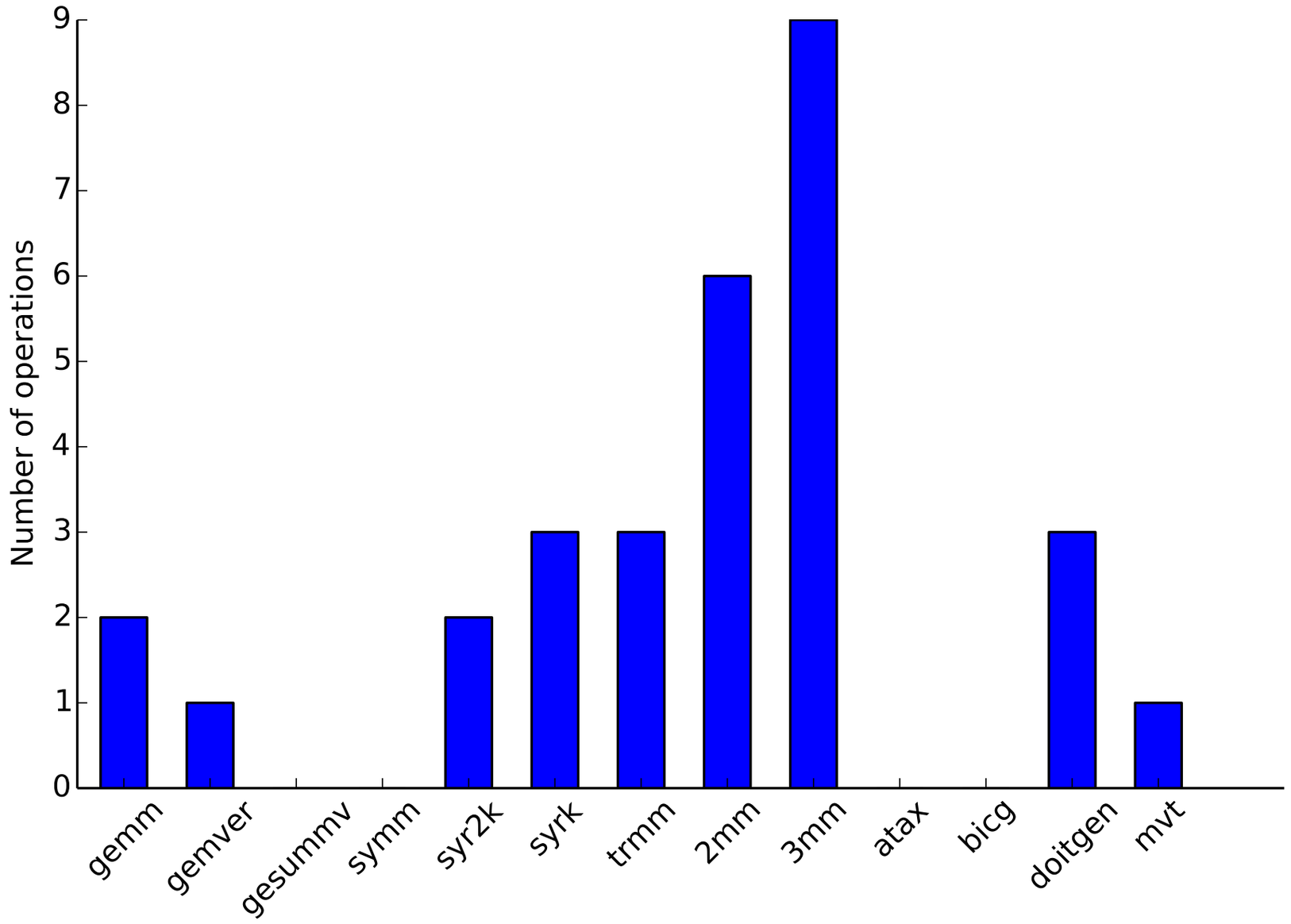
- Implemented in Polly, a polyhedral library for LLVM
- Evaluated on Polybench benchmark suite
  - 30 programs from diverse domains
- Comparison against ICC, PLUTO(Polly) and LLVM on *single-threaded* performance

Benchmark Performance



Linear Algebra Kernels

Script size



Linear Algebra Kernels

Conclusion

- Fun with Loop Transformations!
  - Scripting language lets one *play* with transformations
  - Correctness checking *guards* against mistakes
- Future directions
  - New transformations for Multicore and GPUs
  - Synthesizing scripts

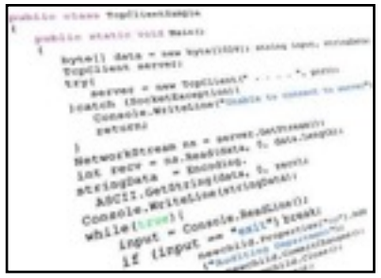
Thank you!

(https://github.com/nimit-singhania/loopy.git)

Acknowledgements

This work was supported, in part, by DARPA under agreement num- ber FA8750-12-C-0166. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

Polyhedral Model Verification



Polyhedral Model

Loop Invariant

Ranking Function

Iteration Domain

Schedule

realign(Init, Mult, 0)

```
[0]   for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
          Init: C[i][j] = 0;
        }
      }
[1]   for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
          for (k = 0; k < N; k++) {
            Mult:
              C[i][j] += A[i][k] * B[k][j];
          }
        }
      }
```

[0, 0, 0, 0]

[1, 0, 0, 0]

$\phi_{realign}([0, 0, 0, 0], \mathbf{j}) = ([0, 0, 0, 0], \mathbf{j})$

$\phi_{realign}([0, 0, 1, 0], \mathbf{j}) = ([1, 0, 0, 0], \mathbf{j})$

realign(Init, Mult, 0)

```
[0]    for (i = 0; i < N; i++) {
         for (j = 0; j < N; j++) {
           Init: C[i][j] = 0;
         }
       }
[1]    for (i = 0; i < N; i++) {
         for (j = 0; j < N; j++) {
           for (k = 0; k < N; k++) {
             Mult:
               C[i][j] += A[i][k] * B[k][j];
           }
         }
       }
```

[0, 0, 0]

[1, 0, 0, 0]

$$\phi_{realign}(p, j) = \begin{cases} (p + p_{init} - p_{mult} + 1(0), j) & \text{if } p \geq p_{mult} \\ (p, j) & \text{otherwise} \end{cases}$$