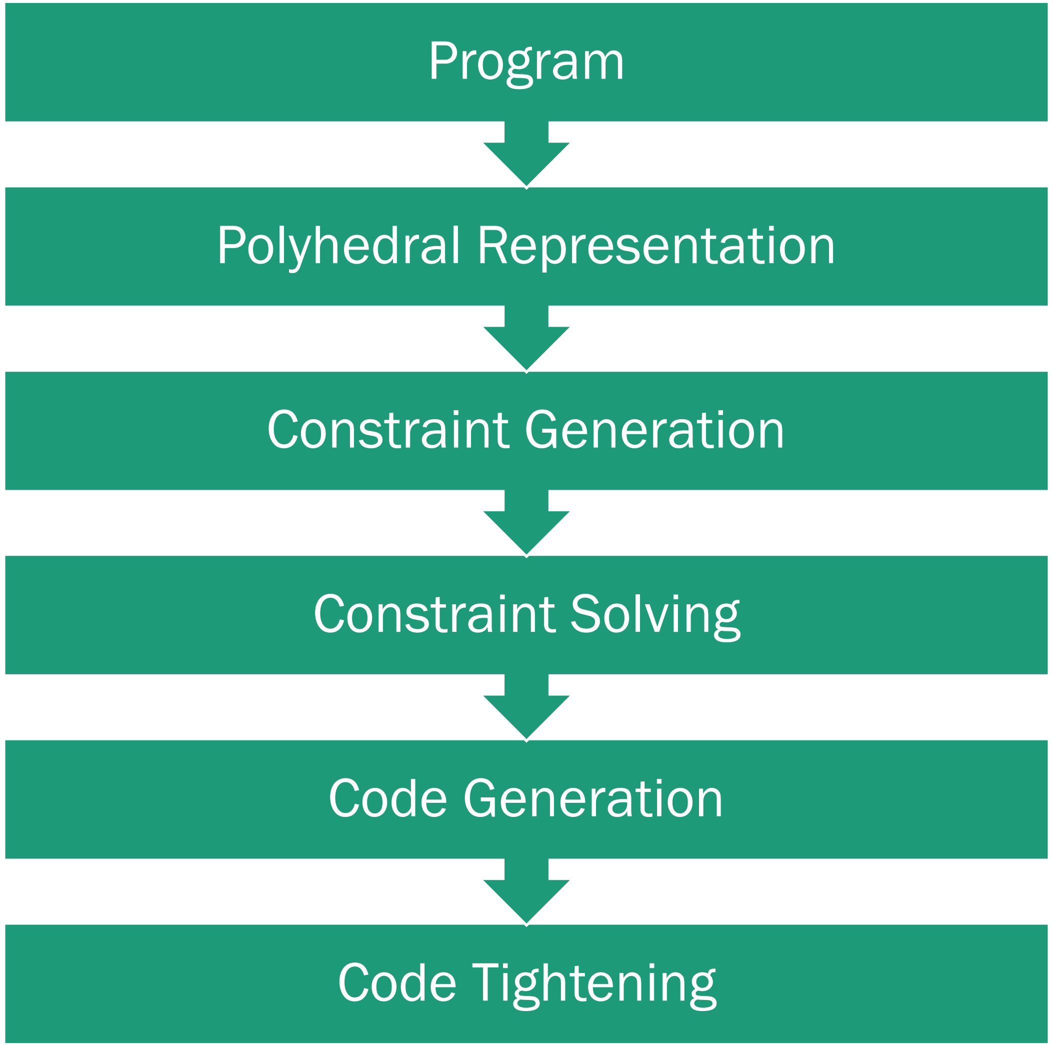
Primitive Affine Transformations

Modules 151-156

Advanced Compiler Techniques

Rupanshu Soi

Affine Transformation Pipeline



Seven Primitive Affine Transforms

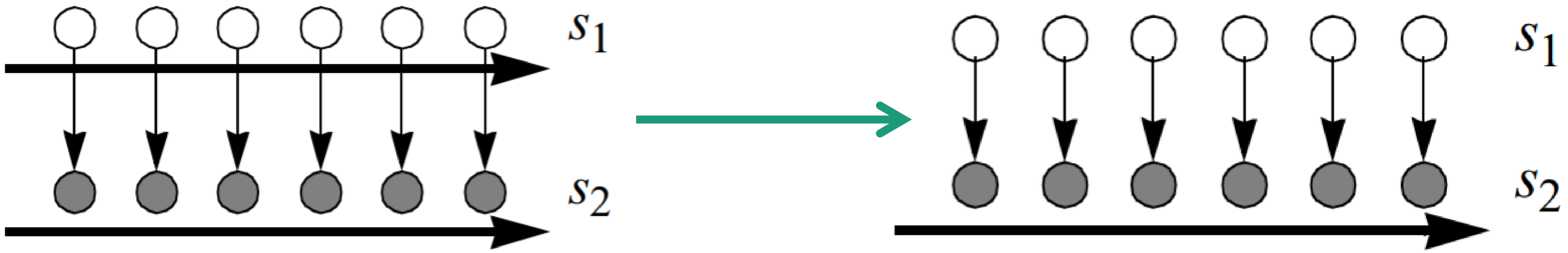
- Every affine transform can be expressed as a series of primitive affine transforms.
- Each will simply fall out of our space partitioning technique for maximizing synchronization-free parallelism.

Affine Transform I: Fusion

```
// Original
for (i = 0; i <= N; i++)
    Y[i] = Z[i];
for (j = 0; j <= N; j++)
    X[j] = Y[j];
```

$s_1 \colon [0, N]$
$s_2 \colon [0, N]$
$p(i) = p(j)$ whenever $i = j$

$s_1 \colon p = i$
$s_2 \colon p = j$



Affine Transform I: Fusion

$s_1 : p = i$
$s_2 : p = j$

```
// Original
for (i = 0; i <= N; i++)
    Y[i] = Z[i];
for (j = 0; j <= N; j++)
    X[j] = Y[j];
```

```
// Simple codegen
for (p = 0; p <= N; p++) {
    for (i = 0; i <= N; i++) {
        if (i == p)
            Y[i] = Z[i];
    }
    for (j = 0; j <= N; j++) {
        if (j == p)
            X[j] = Y[j];
    }
}
```

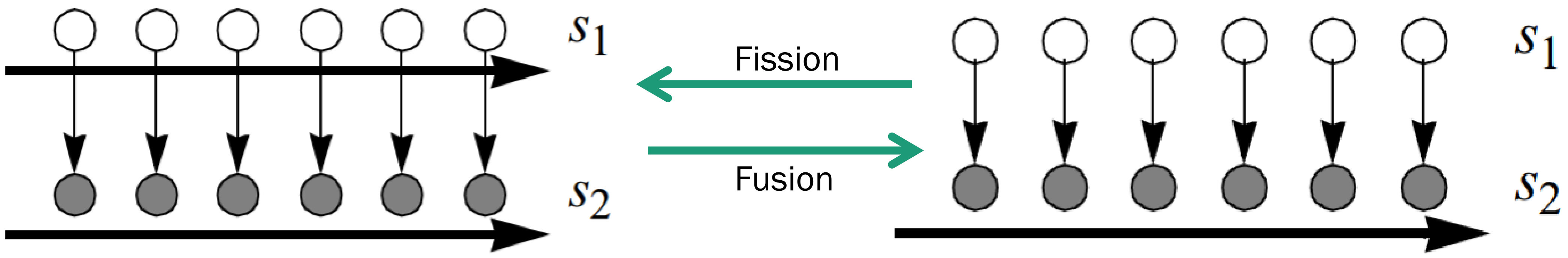
Affine Transform I: Fusion

$s_1 : p = i$
$s_2 : p = j$

```
// Simple codegen
for (p = 0; p <= N; p++) {
    for (i = 0; i <= N; i++) {
        if (i == p)
            Y[i] = Z[i];
    }
    for (j = 0; j <= N; j++) {
        if (j == p)
            X[j] = Y[j];
    }
}
```

```
// Tightened
for (p = 0; p <= N; p++) {
    Y[p] = Z[p];
    X[p] = Y[p];
}
```

Affine Transform II: Fission

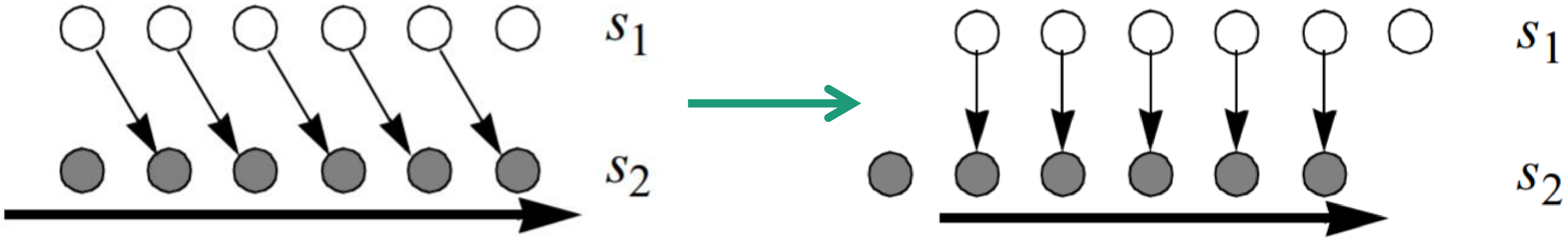


Affine Transform III: Reindexing

```
// Original
for (i = 0; i <= N; i++) {
    Y[i] = Z[i];
    X[i] = Y[i - 1];
}
```

$s_1: [0, N]$
$s_2: [0, N]$
$p(i_1) = p(i_2)$ whenever $i_1 = i_2 - 1$

$s_1: p = i$
$s_2: p = i - 1$



Affine Transform III: Reindexing

$s_1 : p = i$
$s_2 : p = i - 1$

```
// Original
for (i = 0; i <= N; i++) {
    Y[i] = Z[i];
    X[i] = Y[i - 1];
}
```

```
// Simple codegen
for (p = -1; p <= N; p++) {
    for (i = 0; i <= N; i++) {
        if (i == p)
            Y[i] = Z[i];
        if (i - 1 == p)
            X[i] = Y[i - 1];
    }
}
```

Affine Transform III: Reindexing

$s_1 : p = i$
$s_2 : p = i - 1$

```
// Simple codegen
for (p = -1; p <= N; p++) {
    for (i = 0; i <= N; i++) {
        if (i == p)
            Y[i] = Z[i];
        if (i - 1 == p)
            X[i] = Y[i - 1];
    }
}
```

```
// Tightened I
for (p = -1; p <= N; p++) {
    for (i = max(0, p);
         i <= min(p + 1, N);
         i++) {
        if (i == p)
            Y[i] = Z[i];
        if (i - 1 == p)
            X[i] = Y[i - 1];
    }
}
```

Affine Transform III: Reindexing

$s_1 : p = i$
$s_2 : p = i - 1$

```
// Tightened I
for (p = -1; p <= N; p++) {
    for (i = max(0, p);
         i <= min(p + 1, N);
         i++) {
         if (i == p)
             Y[i] = Z[i];
         if (i - 1 == p)
             X[i] = Y[i - 1];
    }
}
```

```
// Tightened II
if (N >= 0) X[0] = Y[-1];
for (p = 0; p <= N - 1; p++) {
    Y[p] = Z[p];
    X[p + 1] = Y[p];
}
if (N >= 0) Y[N] = Z[N];
```

Partitions over p: $[-1, -1], [0, N - 1], [N, N]$

Affine Transform IV: Scaling

```
// Original
for (i = 0; i <= N; i++)
    Y[2 * i] = Z[2 * i];
for (j = 0; j <= 2 * N; j++)
    X[j] = Y[j];
```

s_1: $[0, N]$
s_2: $[0, 2N]$
$p(i) = p(j)$ whenever $2i = j$

$s_1: p = 2i$
$s_2: p = j$

Affine Transform IV: Scaling

$s_1 : p = 2i$
$s_2 : p = j$

```
// Original
for (i = 0; i <= N; i++)
    Y[2 * i] = Z[2 * i];
for (j = 0; j <= 2 * N; j++)
    X[j] = Y[j];
```

```
// Simple codegen
for (p = 0; p <= 2 * N; p++) {
    for (i = 0; i <= N; i++) {
        if (2 * i == p)
            Y[2 * i] = Z[2 * i];
    }
    for (j = 0; k <= 2 * N; j++) {
        if (j == p)
            X[j] = Y[j];
    }
}
```

Affine Transform IV: Scaling

$s_1 : p = 2i$
$s_2 : p = j$

```
// Simple codegen
for (p = 0; p <= 2 * N; p++) {
    for (i = 0; i <= N; i++) {
        if (2 * i == p)
            Y[2 * i] = Z[2 * i];
    }
    for (j = 0; k <= 2 * N; j++) {
        if (j == p)
            X[j] = Y[j];
    }
}
```

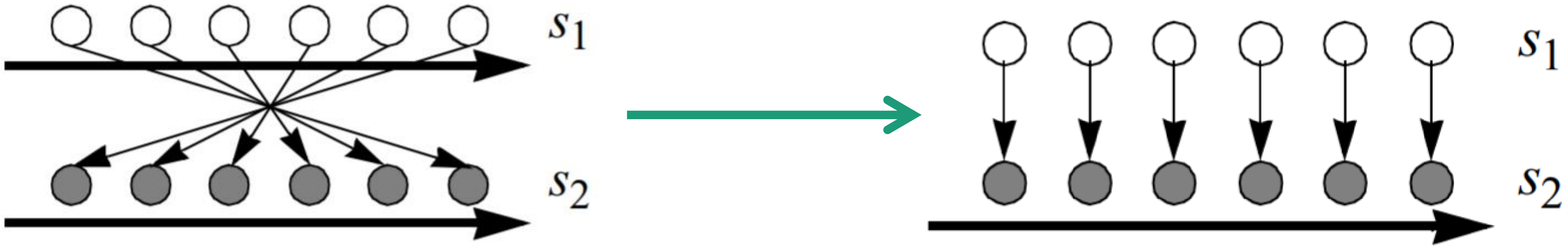
```
// Tightened
for (p = 0; p <= 2 * N; p++) {
    if (p % 2 == 0)
        Y[p] = Z[p];
    X[p] = Y[p];
}
```

Affine Transform V: Reversal

```
// Original
for (i = 0; i <= N; i++)
    Y[N - i] = Z[i];
for (j = 0; j <= N; j++)
    X[j] = Y[j];
}
```

s_1: $[0, N]$
s_2: $[0, N]$
$p(i) = p(j)$ whenever $N - i = j$

$s_1 : p = N - i$
$s_2 : p = j$



Affine Transform V: Reversal

$$s_1 : p = N - i$$
$$s_2 : p = j$$

```
// Original
for (i = 0; i <= N; i++)
    Y[N - i] = Z[i];
for (j = 0; j <= N; j++)
    X[j] = Y[j];
}
```

```
// Simple codegen
for (p = 0; p <= N; p++) {
    for (i = 0; i <= N; i++) {
        if (N - i == p)
            Y[N - i] = Z[i];
    }
    for (j = 0; j <= N; j++) {
        if (j == p)
            X[j] = Y[j];
    }
}
```

Affine Transform V: Reversal

$s_1 : p = N - i$
$s_2 : p = j$

```
// Simple codegen
for (p = 0; p <= N; p++) {
    for (i = 0; i <= N; i++) {
        if (N - i == p)
            Y[N - i] = Z[i];
    }
    for (j = 0; j <= N; j++) {
        if (j == p)
            X[j] = Y[j];
    }
}
```

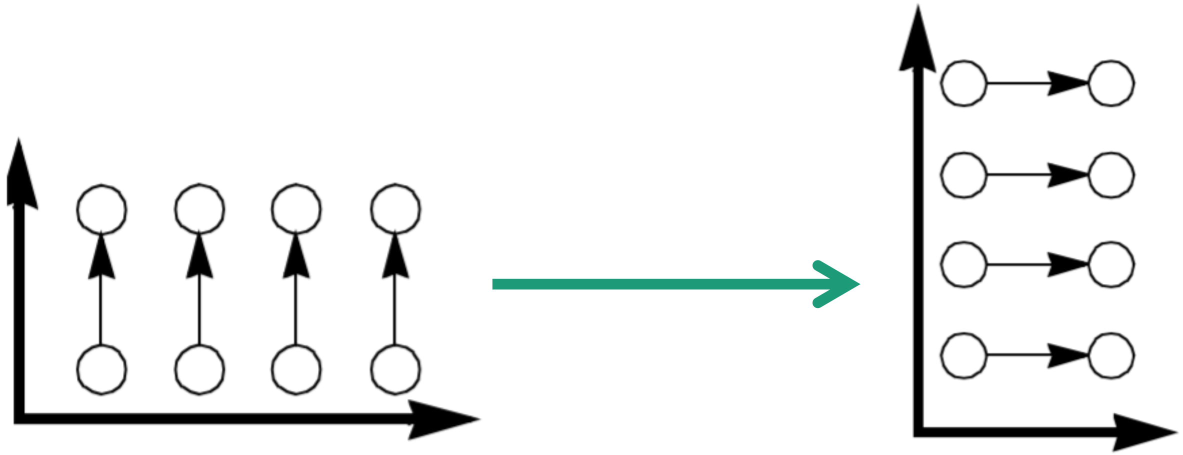
```
// Tightened
for (p = 0; p <= N; p++) {
    Y[p] = Z[N - p];
    X[p] = Y[p];
}
```

Affine Transform VI: Permutation

```
// Original
for (i = 0; i <= N; i++) {
    for (j = 0; j <= M; j++) {
        Z[i, j] = Z[i - 1, j];
    }
}
```

$$s_1 : [0, N] \times [0, M]$$
$$p(i,j) = p(i',j') \text{ whenever } (i,j) = (i'-1, j')$$

$$s_1 : p = j$$



Affine Transform VI: Permutation

$s_1 : p = j$

```
// Original
for (i = 0; i <= N; i++) {
    for (j = 0; j <= M; j++) {
        Z[i, j] = Z[i - 1, j];
    }
}
```

```
// Simple codegen
for (p = 0; p <= M; p++) {
    for (i = 0; i <= N; i++) {
        for (j = 0; j <= M; j++) {
            if (j == p)
                Z[i, j] = Z[i - 1, j];
        }
    }
}
```

Affine Transform VI: Permutation

$s_1 : p = j$

```
// Simple codegen
for (p = 0; p <= M; p++) {
    for (i = 0; i <= N; i++) {
        for (j = 0; j <= M; j++) {
            if (j == p)
                Z[i, j] = Z[i - 1, j];
        }
    }
}
```

```
// Tightened
for (p = 0; p <= M; p++) {
    for (i = 0; i <= N; i++) {
        Z[i, p] = Z[i - 1, p];
    }
}
```

$$\begin{pmatrix} p \\ i' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

Permutation Matrix

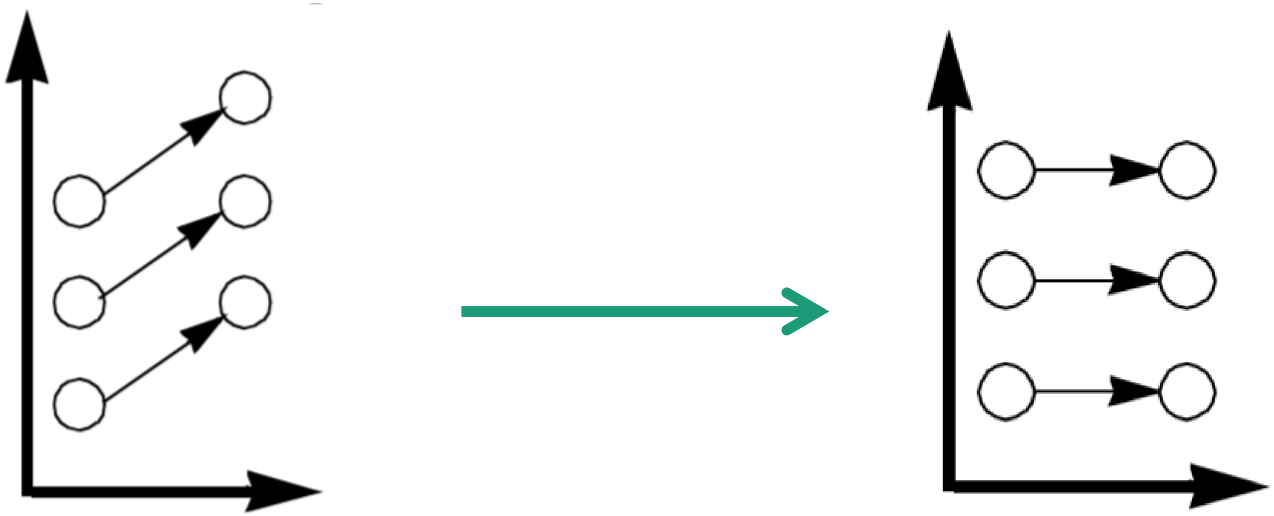
Affine Transform VII: Skewing

```
// Original
for (i = 0; i <= N; i++) {
    for (j = 0; j <= M; j++) {
        Z[i, j] = Z[i - 1, j - 1];
    }
}
```

$$s_1 : [0, N] \times [0, M]$$

$$p(i, j) = p(i', j')$$
whenever
$$(i, j) = (i' - 1, j' - 1)$$

$$s_1 : p = i - j$$



Affine Transform VII: Skewing

$s_1 : p = i - j$

```
// Original
for (i = 0; i <= N; i++) {
    for (j = 0; j <= M; j++) {
        Z[i, j] = Z[i - 1, j - 1];
    }
}
```

```
// Simple codegen
for (p = -M; p <= N; p++) {
    for (i = 0; i <= N; i++) {
        for (j = 0; j <= M; j++) {
            if (i - j == p)
                Z[i, j] =
                Z[i - 1, j - 1];
        }
    }
}
```

Affine Transform VII: Skewing

$s_1 : p = i - j$

```
// Simple codegen
for (p = -M; p <= N; p++) {
    for (i = 0; i <= N; i++) {
        for (j = 0; j <= M; j++) {
            if (i - j == p)
                Z[i, j] =
                Z[i - 1, j - 1];
        }
    }
}
```

```
// Tightened I
for (p = -M; p <= N; p++) {
    for (i = 0; i <= N; i++) {
        if (i - p >= 0
        && i - p <= M)
            Z[i, i - p] =
            Z[i - 1, i - p - 1];
    }
}
```

Affine Transform VII: Skewing

$s_1 : p = i - j$

```
// Tightened I
for (p = -M; p <= N; p++) {
    for (i = 0; i <= N; i++) {
        if (i - p >= 0
        && i - p <= M)
            Z[i, i - p] =
            Z[i - 1, i - p - 1];
    }
}
```

```
// Tightened II
for (p = -M; p <= N; p++) {
    for (i = max(0, p);
         i <= min(N, p + M);
         i++) {
         Z[i, i - p] =
         Z[i - 1, i - p - 1];
    }
}
```

$$\begin{pmatrix} p \\ i' \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

Skewing Matrix

Geometric Interpretation

- Angle of dependence edges will be in $[0, 180^{\circ})$ because of lexicographic ordering of the iteration space
- Space partitioning tries to ensure that the outer loops are data independent

Thank You!