

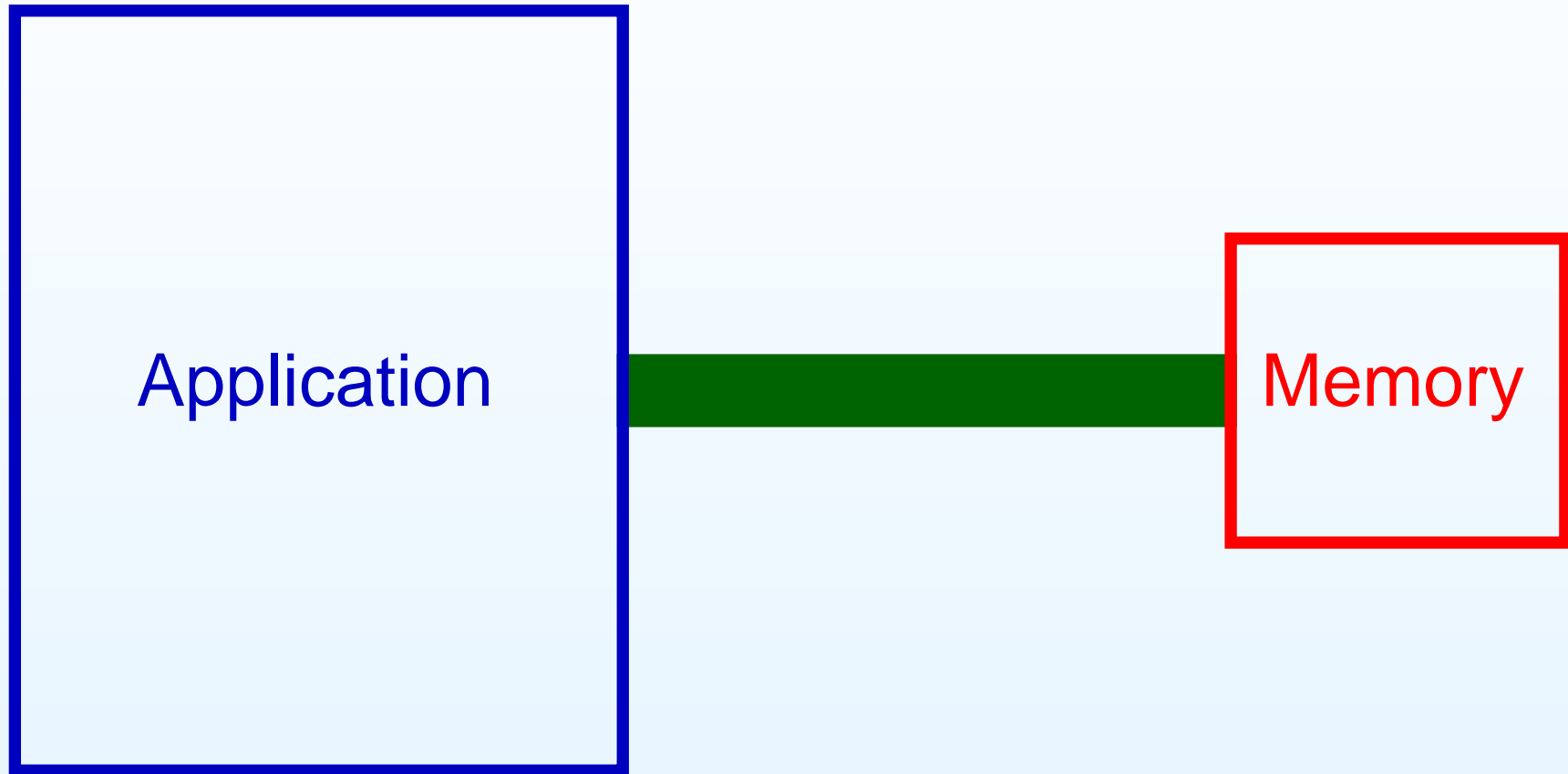
Languages, Compiler, and Tools for Embedded Systems
LCTES'03

Advanced Copy Propagation for Arrays

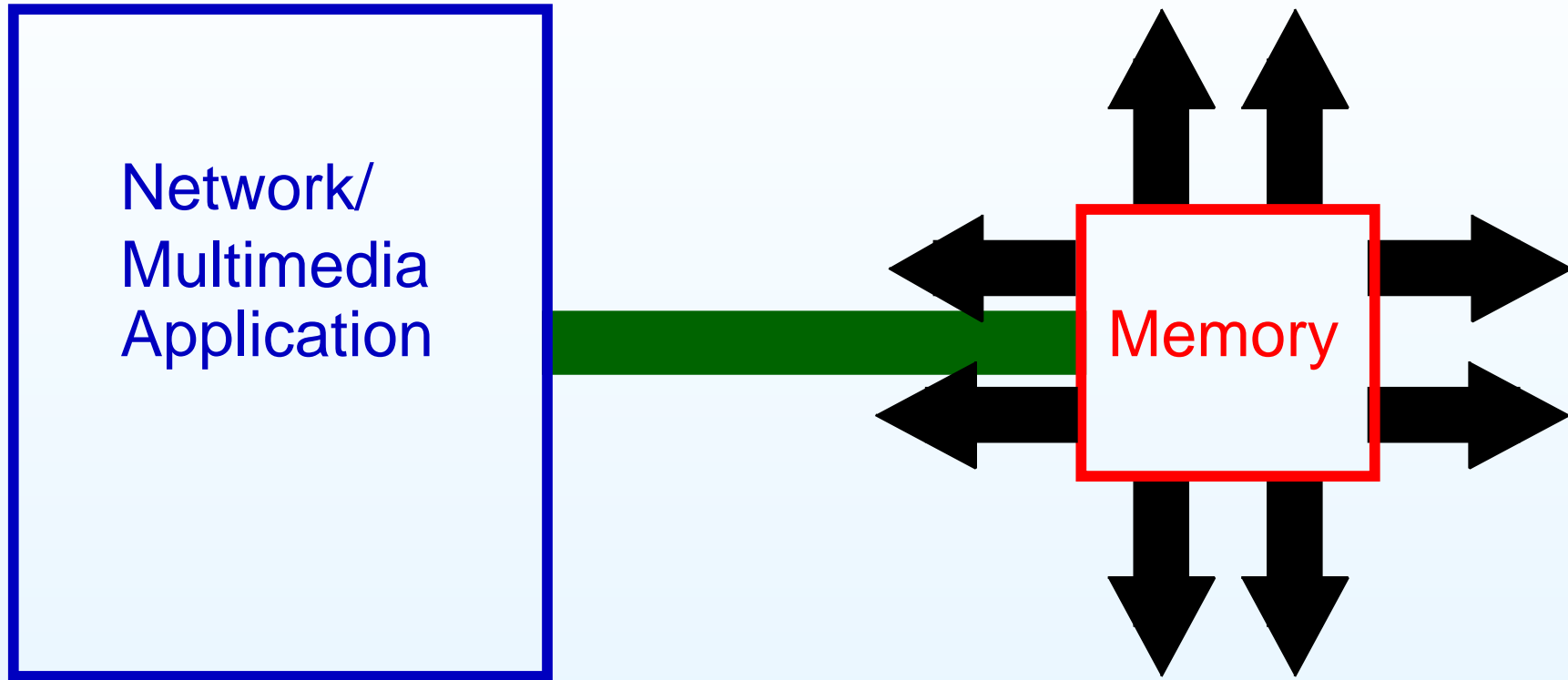
Peter Vanbroekhoven

Dept. Computer Science K.U.Leuven

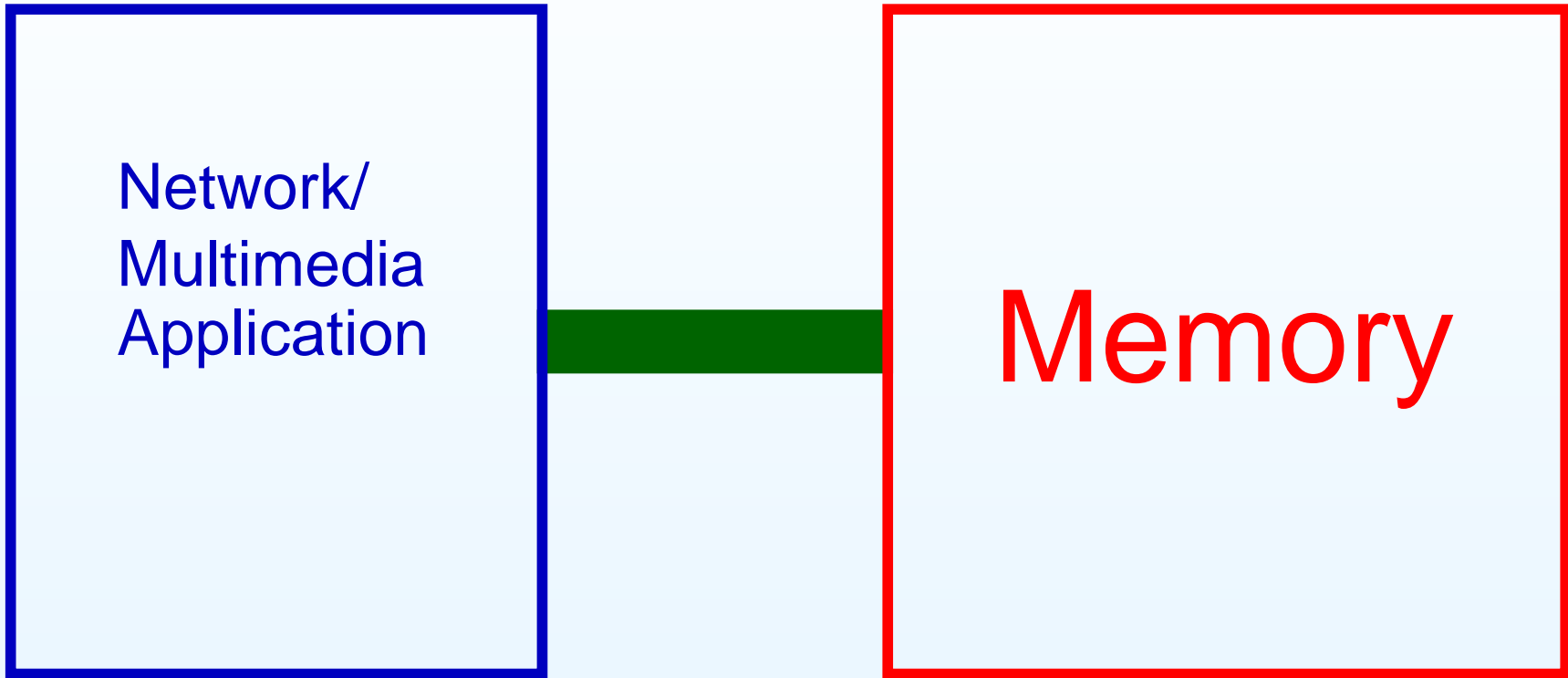
Introduction



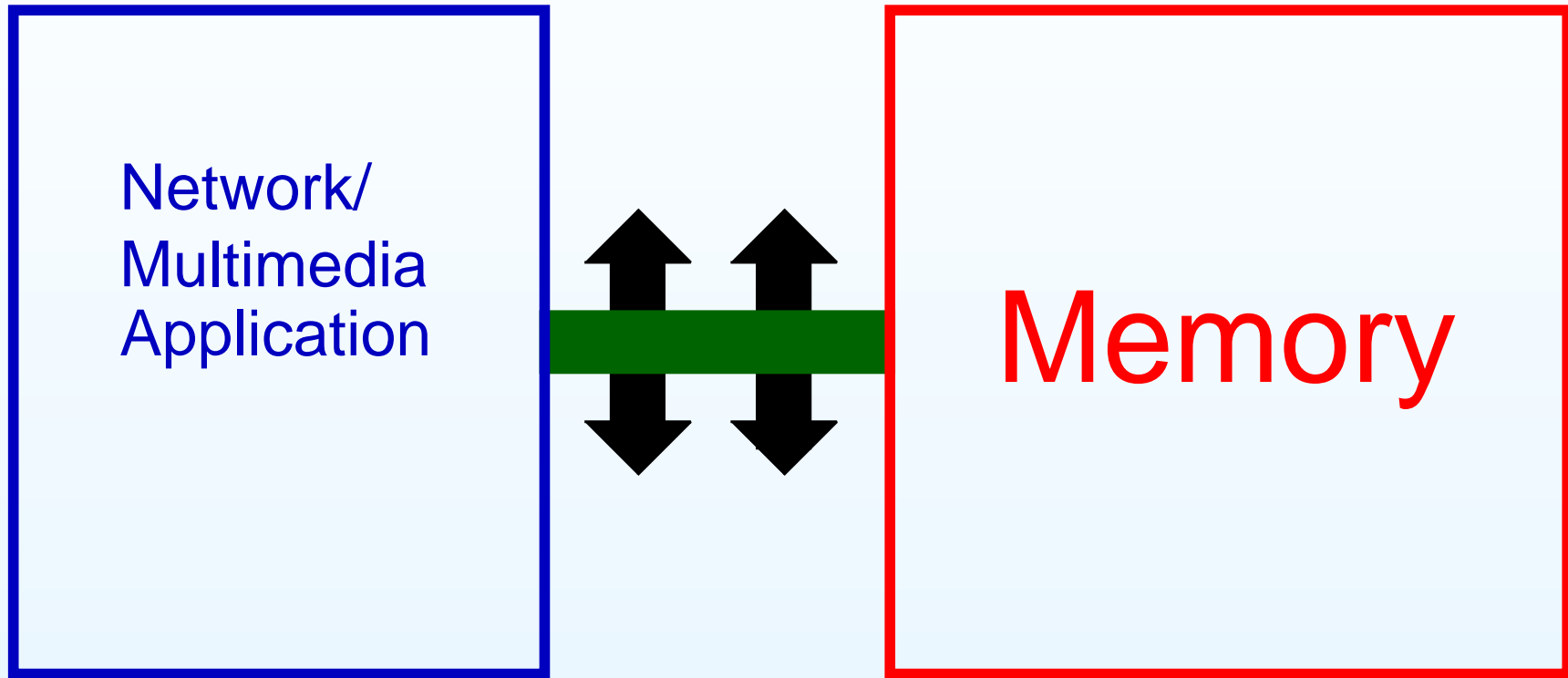
Introduction



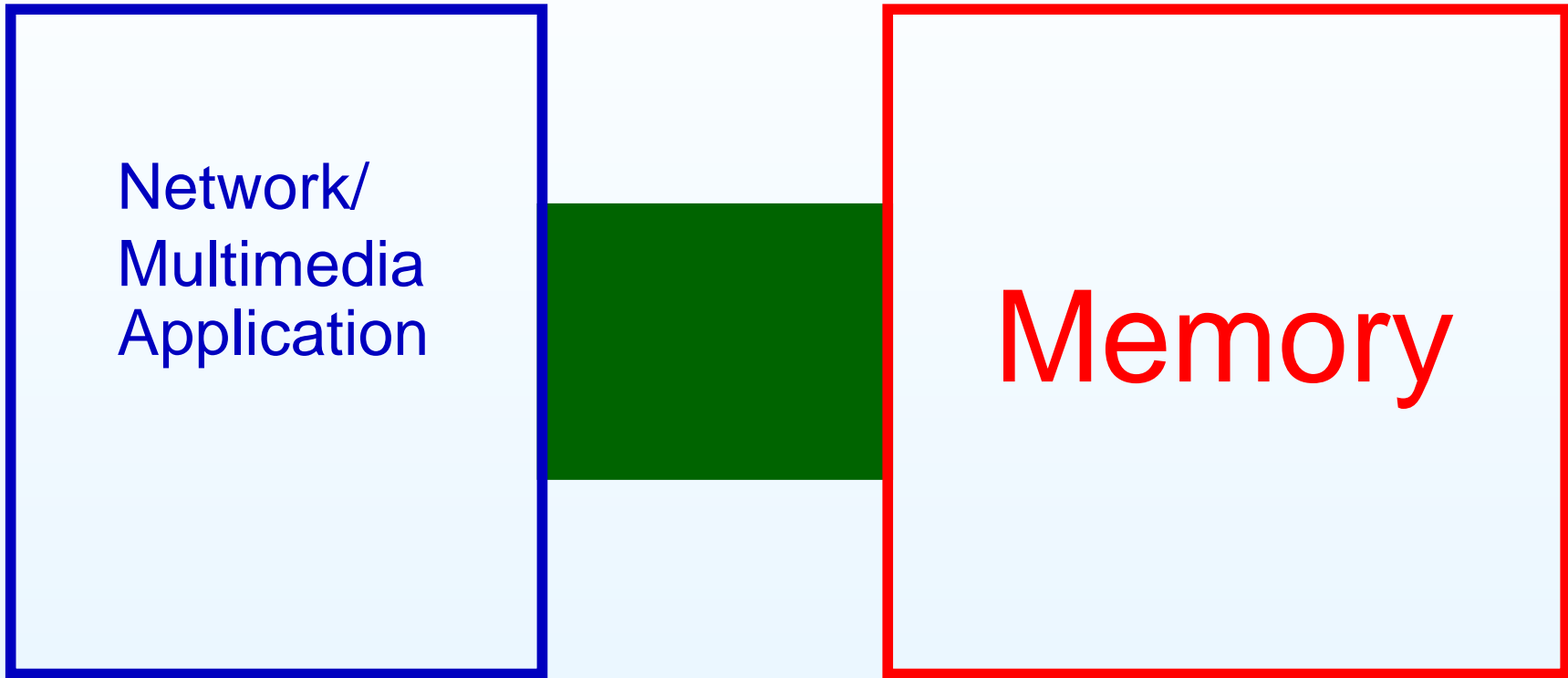
Introduction



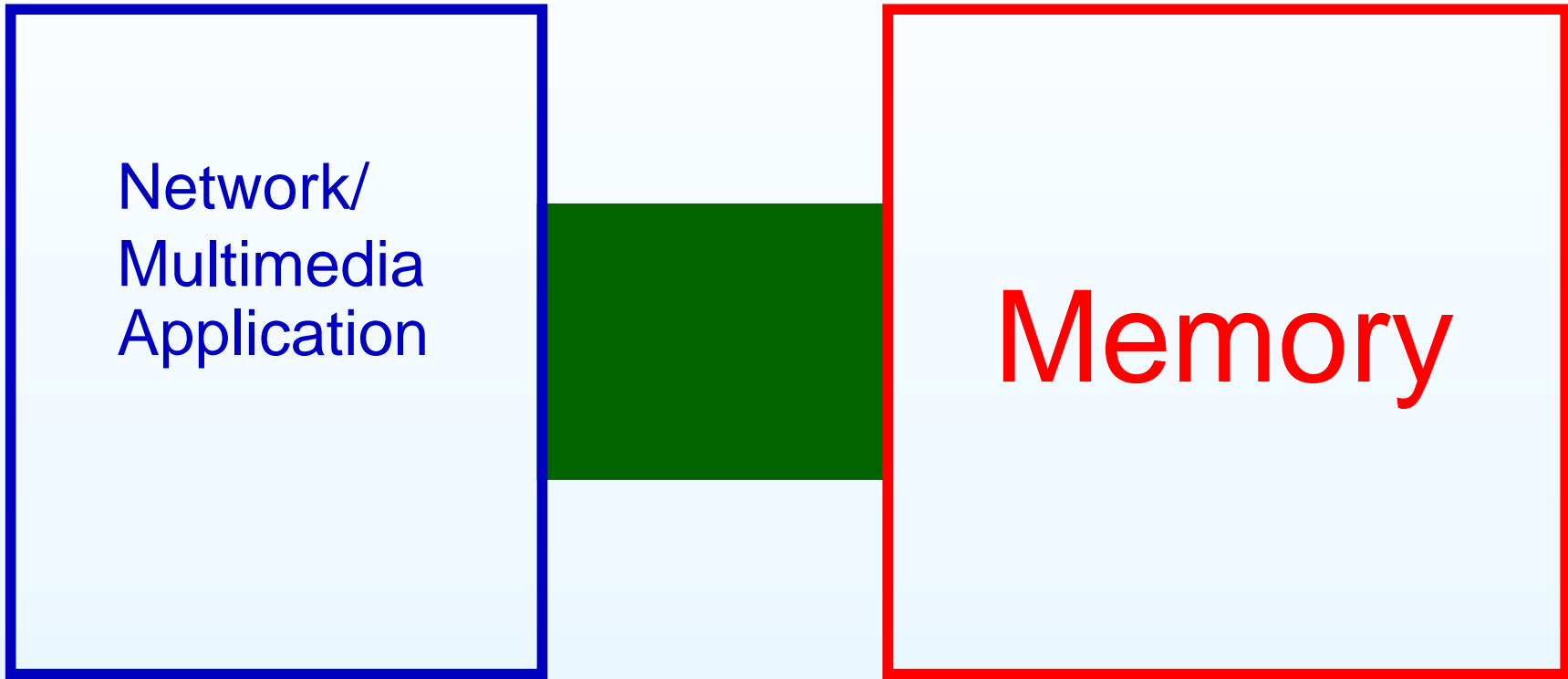
Introduction



Introduction

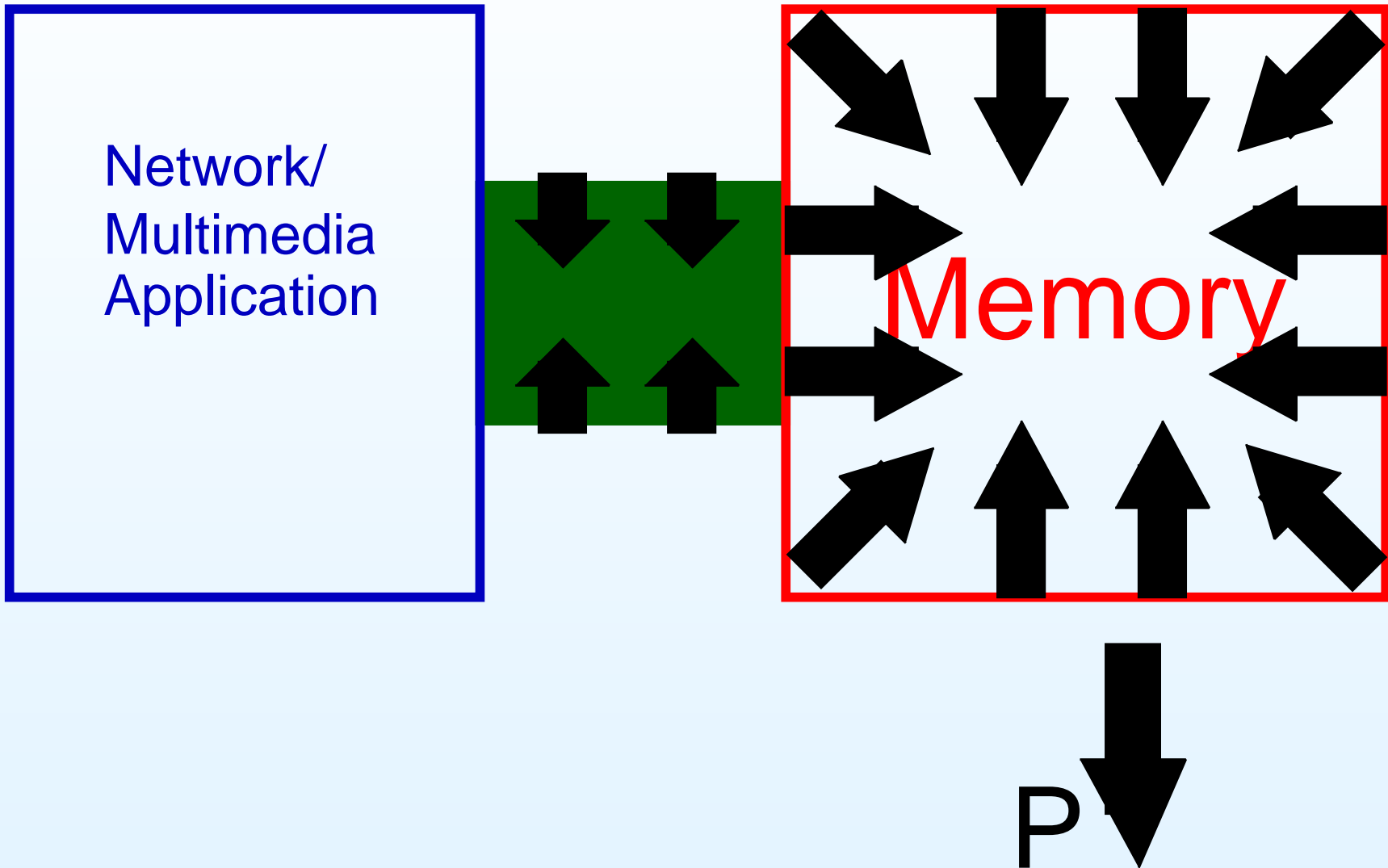


Introduction



P ↑

Introduction



Introduction : limitations “classic” copy propagation

```
for (i = 0; i < 50; i++)
    a[i] = b[i] * b[i];           // writes a[0:49]
for (i = 0; i < 100; i++) {
    if (i < 50) a[i + 50] = b[i]; // w a[50:99], r b[0:49]
    c[i] = a[i];                 // reads a[0:99]
}
temp = a[50];                    // reads a[50]
```

Introduction : limitations “classic” copy propagation

```
for (i = 0; i < 50; i++)
    a[i] = b[i] * b[i];           // writes a[0:49]
for (i = 0; i < 100; i++) {
    if (i < 50) a[i + 50] = b[i]; // w a[50:99], r b[0:49]
    c[i] = a[i];                 // reads a[0:99]
}
temp = a[50];                   // reads a[50]

for (i = 0; i < 50; i++)
    a[i] = b[i] * b[i];           // writes a[0:49]
for (i = 0; i < 100; i++) {
    if (i == 0) a[i + 50] = b[i]; // writes a[50]
    if (i < 50) c[i] = a[i];      // reads a[0:49]
    else c[i] = b[i - 50];        // reads b[0:49]
}
temp = a[50];                   // reads a[50]
```

Introduction

Where do copy operations come from?

- To facilitate programming
 - Swap operations
 - Adding border to image before applying filter
 - ...
- Operations that are really copy operations
 - Rotating/mirroring/resizing an image
 - Transpose of a matrix, swapping rows/columns of a matrix
 - ...
- Can be introduced by other optimizations

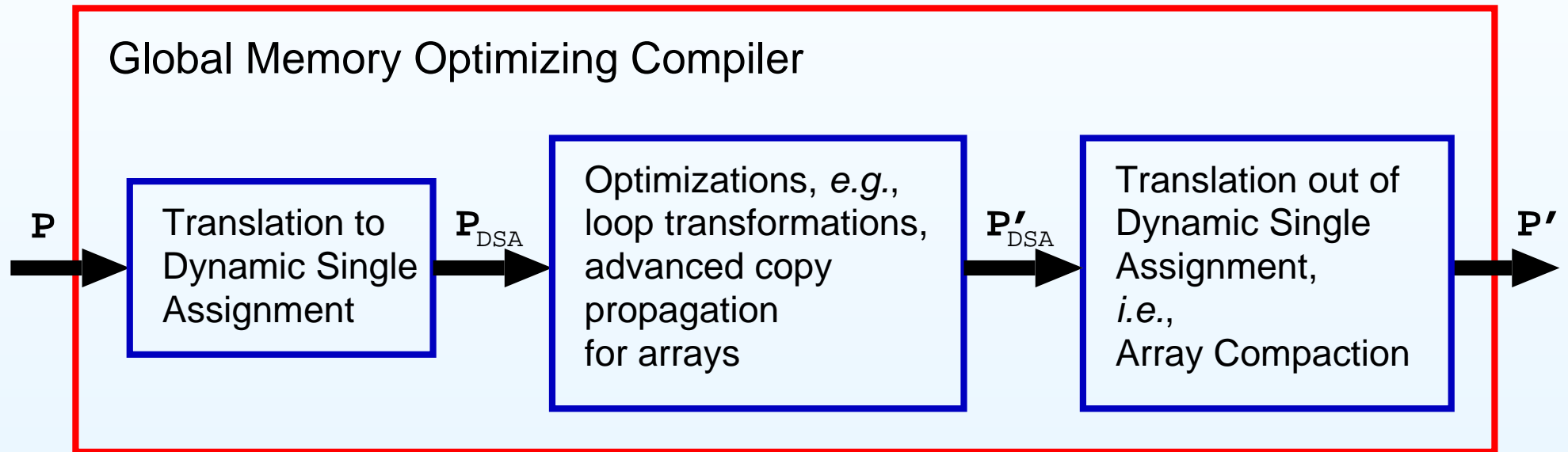
Introduction

- Copy propagation + dead code elimination is done, but for scalar variables
- Distinction between array elements / iterations is done in *e.g.*, parallelization, systolic arrays
- But the combination is not done
- We want to do global copy propagation for arrays by distinguishing between array elements and iterations
- Advanced copy propagation for arrays can substantially reduce the array accesses and array sizes and hence decrease memory power consumption

Overview

- Intermediate representation
- Non-recursive propagation
- Recursive propagation
- Implementation
- Results
- Conclusion

Intermediate representation



IR : Dynamic Single Assignment

= a single assignment to each separate array element during execution of the program

```
a = 1; b = 0;
for (i = 0; i < 100; i++) {
    out[i] = a+b;
    a = b;
    b = out[i];
}
```

IR : Dynamic Single Assignment

= a single assignment to each separate array element during execution of the program

```
a = 1; b = 0;
for (i = 0; i < 100; i++) {
    out[i] = a+b;
    a = b;
    b = out[i];
}
```

```
a1 = 1; b1 = 0;
for (i = 0; i < 100; i++) {
    out[i] = (i == 0 ? a1+b1 : a2[i - 1]+b2[i - 1]);
    a2[i] = b2[i - 1];
    b2[i] = out2[i];
}
```

IR : Dynamic Single Assignment

Programs should satisfy following conditions

- Any possible nesting of `for`-loops and `if`-statements
- Assignment to array elements. Indexing should be of the following form (affine):

$$A \cdot \vec{i} + \vec{c}$$

with A a non-singular, square matrix

- Righthand side of assignments can be any expressions with array references
- Bounds of loops and conditions of `if`-statements should be affine (*i.e.*, linear combination of iterators + constant)

IR : Polyhedral Model

```
for (i = 0; i < 100; i++)
  for (j = 99 - i; j < 100; j++)
    if (i + j < 105) {
      a[j + i][j] = in[j + i];           // S1
    }
    else
      a[j + i][j] = a[j + i - 5][j - 3]; // S2
for (i = 0; i < 100; i++)
  out[i] = f(a[i + 99][i]);             // S3
```

IR : Polyhedral Model

```
for (i = 0; i < 100; i++)
  for (j = 99 - i; j < 100; j++)
    if (i + j < 105) {
      a[j + i][j] = in[j + i];           // S1
    }
    else
      a[j + i][j] = a[j + i - 5][j - 3]; // S2
for (i = 0; i < 100; i++)
  out[i] = f(a[i + 99][i]);           // S3
```

Iteration domains

IR : Polyhedral Model

```
for (i = 0; i < 100; i++)
  for (j = 99 - i; j < 100; j++)
    if (i + j < 105) {
      a[j + i][j] = in[j + i];           // s1
    }
    else
      a[j + i][j] = a[j + i - 5][j - 3]; // s2
for (i = 0; i < 100; i++)
  out[i] = f(a[i + 99][i]);             // s3
```

Iteration domains

$$I_1 = \{(i, j) \mid 0 \leq i < 100 \wedge 99 - i \leq j < 100 \wedge j + i < 105 \wedge (i, j) \in \mathbb{Z}^2\}$$

IR : Polyhedral Model

```
for (i = 0; i < 100; i++)
  for (j = 99 - i; j < 100; j++)
    if (i + j < 105) {
      a[j + i][j] = in[j + i];           // s1
    }
    else
      a[j + i][j] = a[j + i - 5][j - 3]; // s2
for (i = 0; i < 100; i++)
  out[i] = f(a[i + 99][i]);             // s3
```

Iteration domains

$$I_1 = \{(i, j) \mid 0 \leq i < 100 \wedge 99 - i \leq j < 100 \wedge j + i < 105 \wedge (i, j) \in \mathbb{Z}^2\}$$

$$I_2 = \{(i, j) \mid 0 \leq i < 100 \wedge 99 - i \leq j < 100 \wedge j + i \geq 105 \wedge (i, j) \in \mathbb{Z}^2\}$$

IR : Polyhedral Model

```
for (i = 0; i < 100; i++)
  for (j = 99 - i; j < 100; j++)
    if (i + j < 105) {
      a[j + i][j] = in[j + i];           // s1
    }
    else
      a[j + i][j] = a[j + i - 5][j - 3]; // s2
for (i = 0; i < 100; i++)
  out[i] = f(a[i + 99][i]);             // s3
```

Iteration domains

$$I_1 = \{(i, j) \mid 0 \leq i < 100 \wedge 99 - i \leq j < 100 \wedge j + i < 105 \wedge (i, j) \in \mathbb{Z}^2\}$$

$$I_2 = \{(i, j) \mid 0 \leq i < 100 \wedge 99 - i \leq j < 100 \wedge j + i \geq 105 \wedge (i, j) \in \mathbb{Z}^2\}$$

$$I_3 = \{(i) \mid 0 \leq i < 100 \wedge i \in \mathbb{Z}\}$$

IR : Polyhedral Model

```
for (i = 0; i < 100; i++)
  for (j = 99 - i; j < 100; j++)
    if (i + j < 105) {
      a[j + i][j] = in[j + i];           // s1
    }
    else
      a[j + i][j] = a[j + i - 5][j - 3]; // s2
for (i = 0; i < 100; i++)
  out[i] = f(a[i + 99][i]);           // s3
```

Iteration domains

$$I_1 = \{(i, j) \mid 0 \leq i < 100 \wedge 99 - i \leq j < 100 \wedge j + i < 105 \wedge (i, j) \in \mathbb{Z}^2\}$$

$$I_2 = \{(i, j) \mid 0 \leq i < 100 \wedge 99 - i \leq j < 100 \wedge j + i \geq 105 \wedge (i, j) \in \mathbb{Z}^2\}$$

$$I_3 = \{(i) \mid 0 \leq i < 100 \wedge i \in \mathbb{Z}\}$$

Definition mapping: $w_2(i, j) \mapsto (i + j, j)$

IR : Polyhedral Model

```
for (i = 0; i < 100; i++)
  for (j = 99 - i; j < 100; j++)
    if (i + j < 105) {
      a[j + i][j] = in[j + i];           // s1
    }
    else
      a[j + i][j] = a[j + i - 5][j - 3]; // s2
for (i = 0; i < 100; i++)
  out[i] = f(a[i + 99][i]);             // s3
```

Iteration domains

$$I_1 = \{(i, j) \mid 0 \leq i < 100 \wedge 99 - i \leq j < 100 \wedge j + i < 105 \wedge (i, j) \in \mathbb{Z}^2\}$$

$$I_2 = \{(i, j) \mid 0 \leq i < 100 \wedge 99 - i \leq j < 100 \wedge j + i \geq 105 \wedge (i, j) \in \mathbb{Z}^2\}$$

$$I_3 = \{(i) \mid 0 \leq i < 100 \wedge i \in \mathbb{Z}\}$$

Definition mapping: $w_2(i, j) \mapsto (i + j, j)$

Operand mapping: $r_2(i, j) \mapsto (i + j - 5, j - 3)$

IR : Polyhedral Model

```
for (i = 0; i < 100; i++)
  for (j = 99 - i; j < 100; j++)
    if (i + j < 105) {
      a[j + i][j] = in[j + i];           // s1
    }
    else
      a[j + i][j] = a[j + i - 5][j - 3]; // s2
for (i = 0; i < 100; i++)
  out[i] = f(a[i + 99][i]);             // s3
```

Iteration domains

$$I_1 = \{(i, j) \mid 0 \leq i < 100 \wedge 99 - i \leq j < 100 \wedge j + i < 105 \wedge (i, j) \in \mathbb{Z}^2\}$$

$$I_2 = \{(i, j) \mid 0 \leq i < 100 \wedge 99 - i \leq j < 100 \wedge j + i \geq 105 \wedge (i, j) \in \mathbb{Z}^2\}$$

$$I_3 = \{(i) \mid 0 \leq i < 100 \wedge i \in \mathbb{Z}\}$$

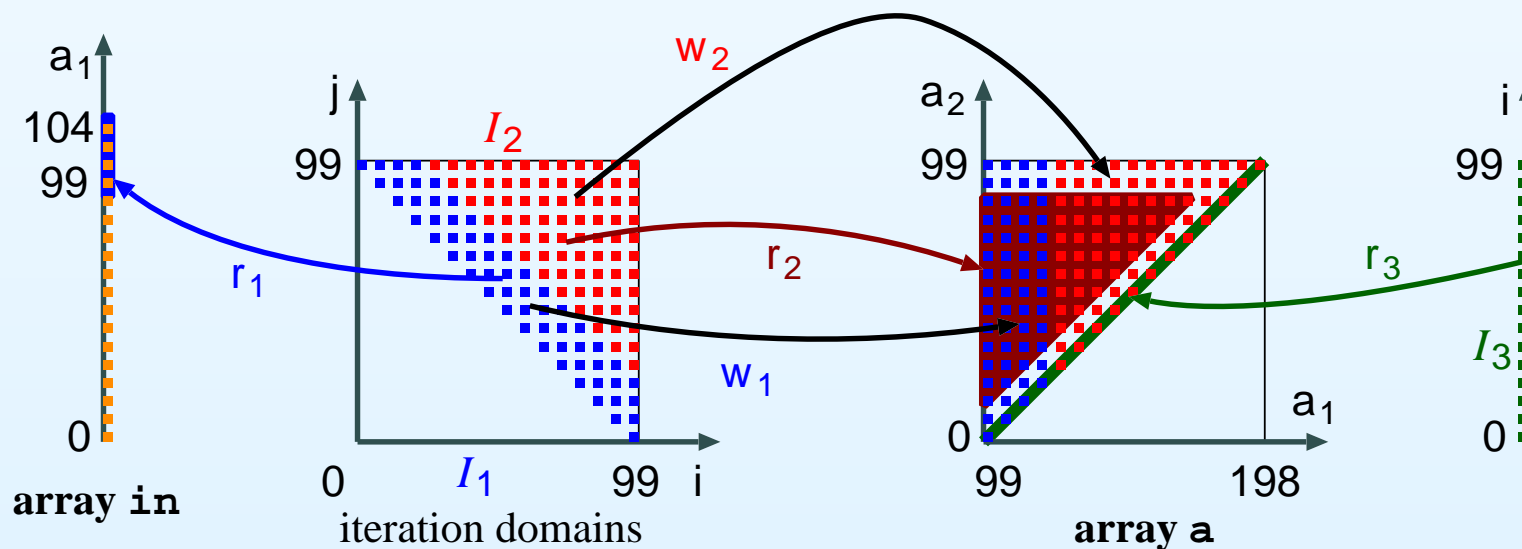
Definition mapping: $w_2(i, j) \mapsto (i + j, j)$

Operand mapping: $r_2(i, j) \mapsto (i + j - 5, j - 3)$

s2(80, 90) writes element $w_2(80, 90) = (170, 90)$ of a
reads element $r_2(80, 90) = (165, 87)$ of a

IR : DSA / Polyhedral Model

```
for (i = 0; i < 100; i++)  
  for (j = 99 - i; j < 100; j++)  
    if (i + j < 105) {  
      a[j + i][j] = in[j + i];           // S1  
    }  
    else  
      a[j + i][j] = a[j + i - 5][j - 3]; // S2  
for (i = 0; i < 100; i++)  
  out[i] = f(a[i + 99][i]);           // S3
```

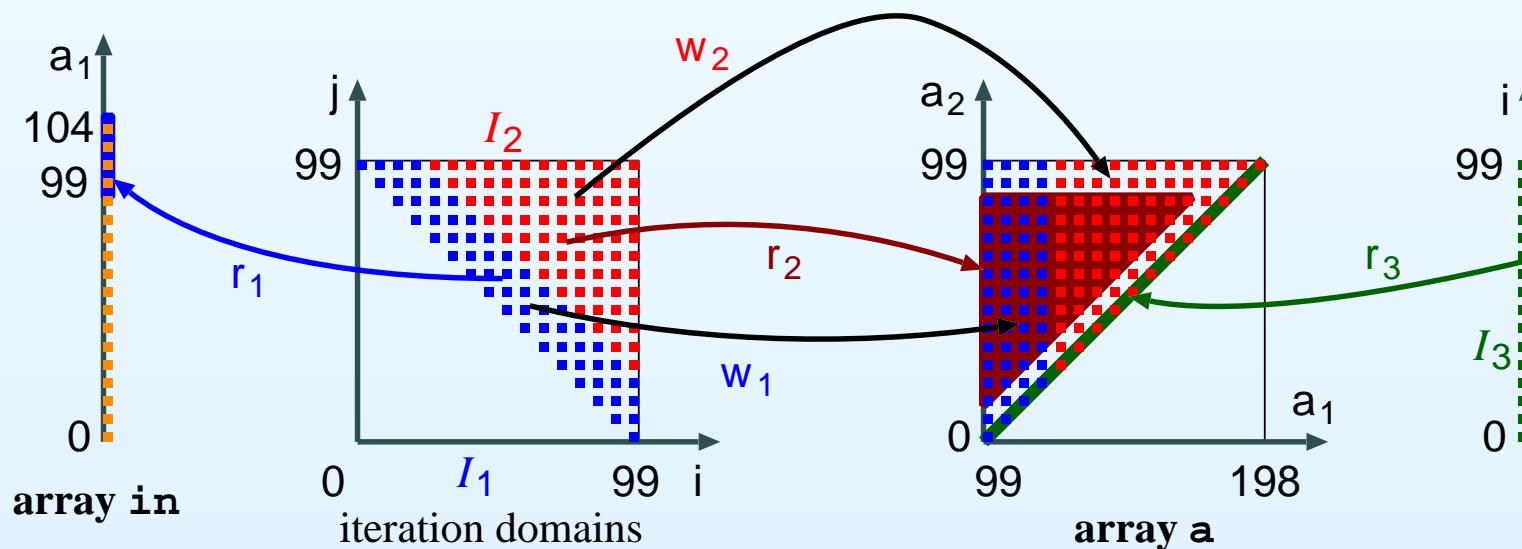


Non-recursive propagation : S1 to S3

```

for (i = 0; i < 100; i++)
  for (j = 99 - i; j < 100; j++)
    if (i + j < 105) {
      a[j + i][j] = in[j + i];           // S1
    }
    else
      a[j + i][j] = a[j + i - 5][j - 3]; // S2
for (i = 0; i < 100; i++)
  out[i] = f(a[i + 99][i]);           // S3

```

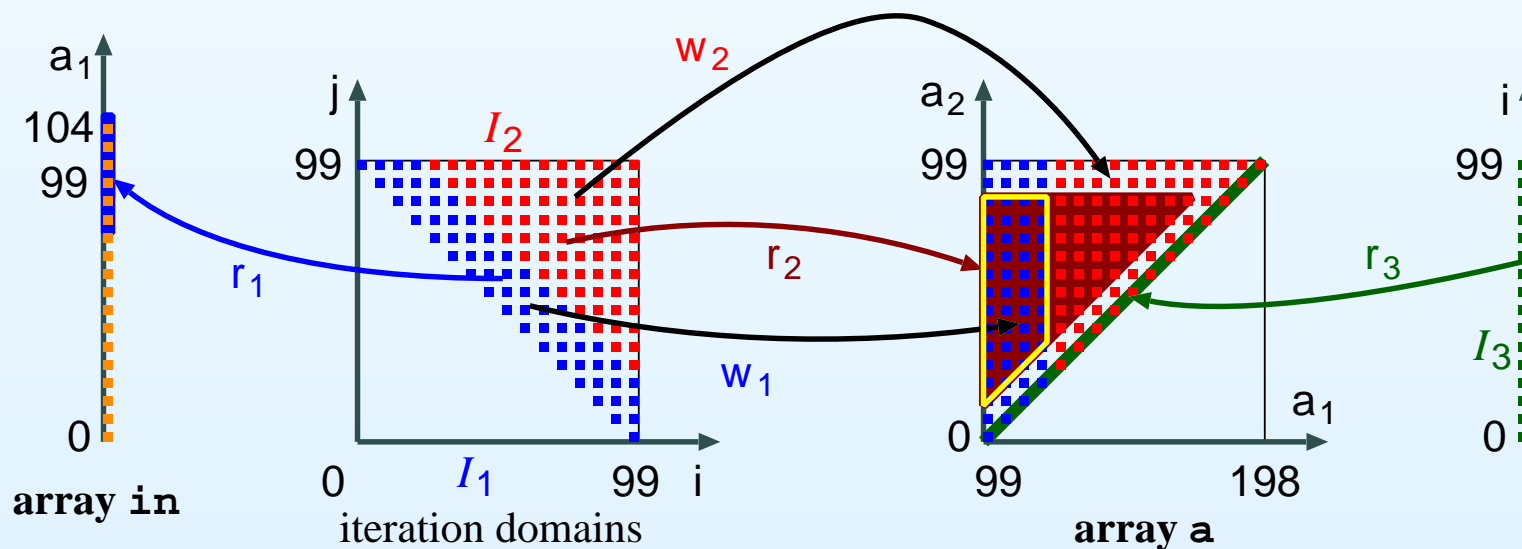


Non-recursive propagation : S1 to S3

```

for (i = 0; i < 100; i++)
  for (j = 99 - i; j < 100; j++)
    if (i + j < 105) {
      a[j + i][j] = in[j + i];           // S1
    }
    else
      a[j + i][j] = a[j + i - 5][j - 3]; // S2
for (i = 0; i < 100; i++)
  out[i] = f(a[i + 99][i]);           // S3

```

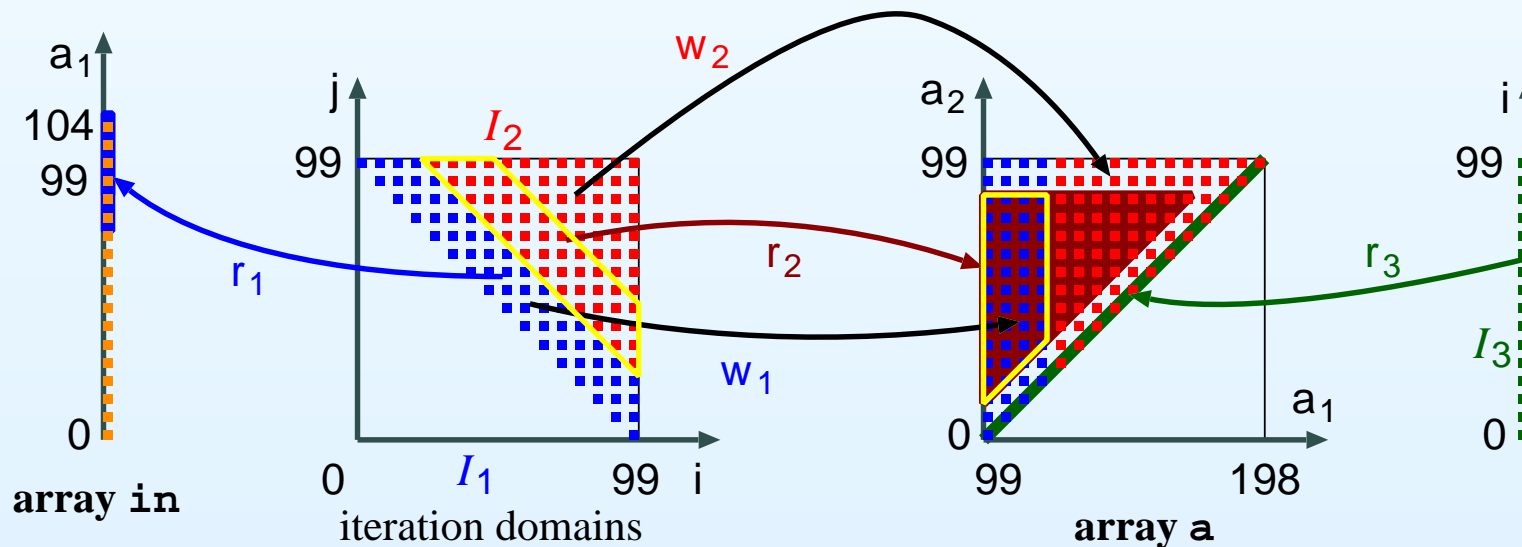


Non-recursive propagation : S1 to S3

```

for (i = 0; i < 100; i++)
  for (j = 99 - i; j < 100; j++)
    if (i + j < 105) {
      a[j + i][j] = in[j + i];           // S1
    }
    else
      a[j + i][j] = a[j + i - 5][j - 3]; // S2
for (i = 0; i < 100; i++)
  out[i] = f(a[i + 99][i]);             // S3

```

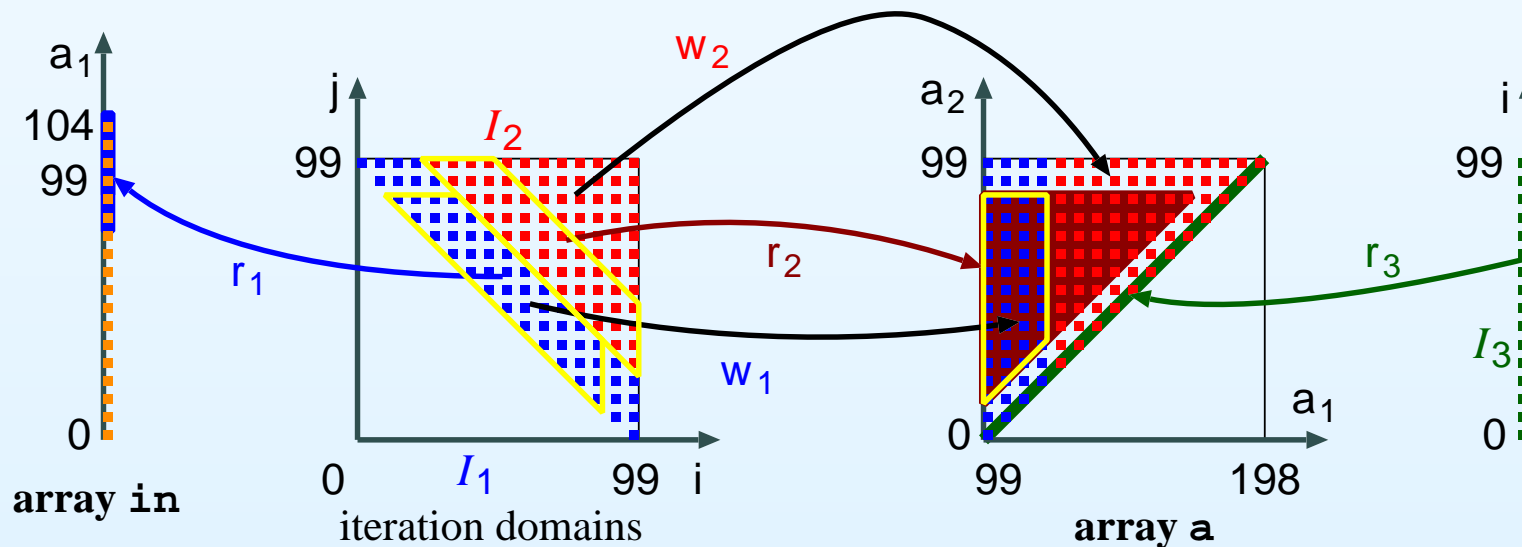


Non-recursive propagation : S1 to S2

```

for (i = 0; i < 100; i++)
  for (j = 99 - i; j < 100; j++)
    if (i + j < 105) {
      a[j + i][j] = in[j + i];           // S1
    }
    else
      a[j + i][j] = a[j + i - 5][j - 3]; // S2
for (i = 0; i < 100; i++)
  out[i] = f(a[i + 99][i]);           // S3

```

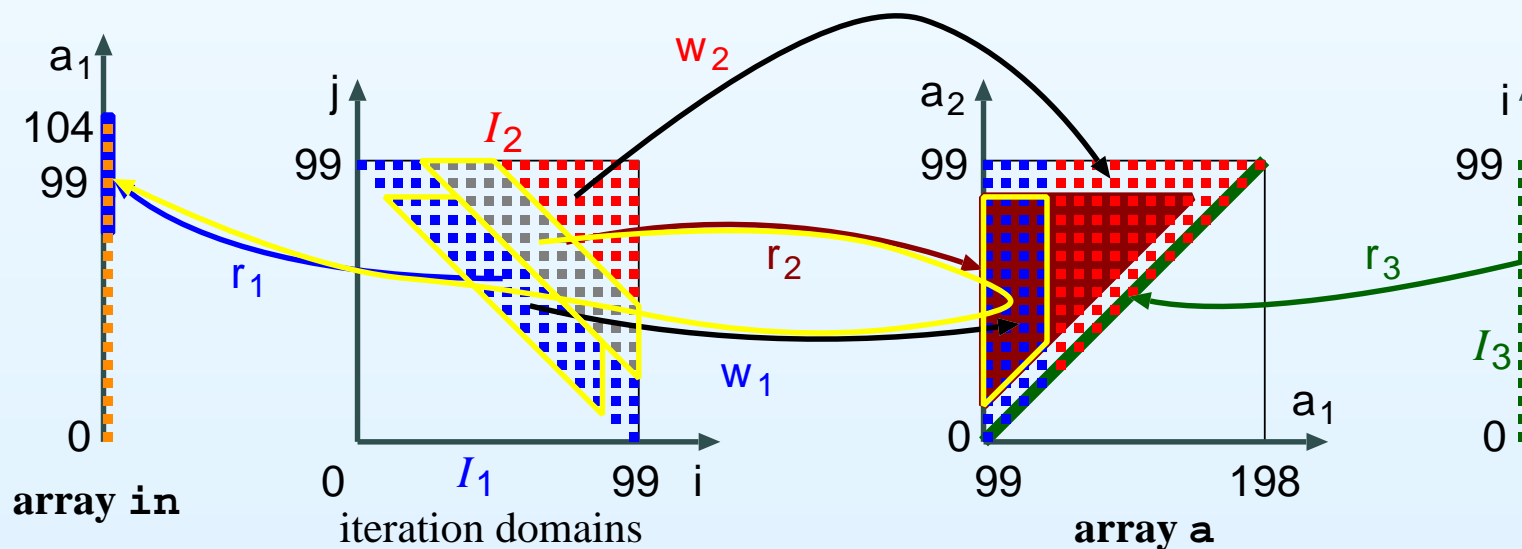


Non-recursive propagation : S1 to S2

```

for (i = 0; i < 100; i++)
  for (j = 99 - i; j < 100; j++)
    if (i + j < 105) {
      a[j + i][j] = in[j + i];           // S1
    }
    else
      a[j + i][j] = a[j + i - 5][j - 3]; // S2
for (i = 0; i < 100; i++)
  out[i] = f(a[i + 99][i]);           // S3

```

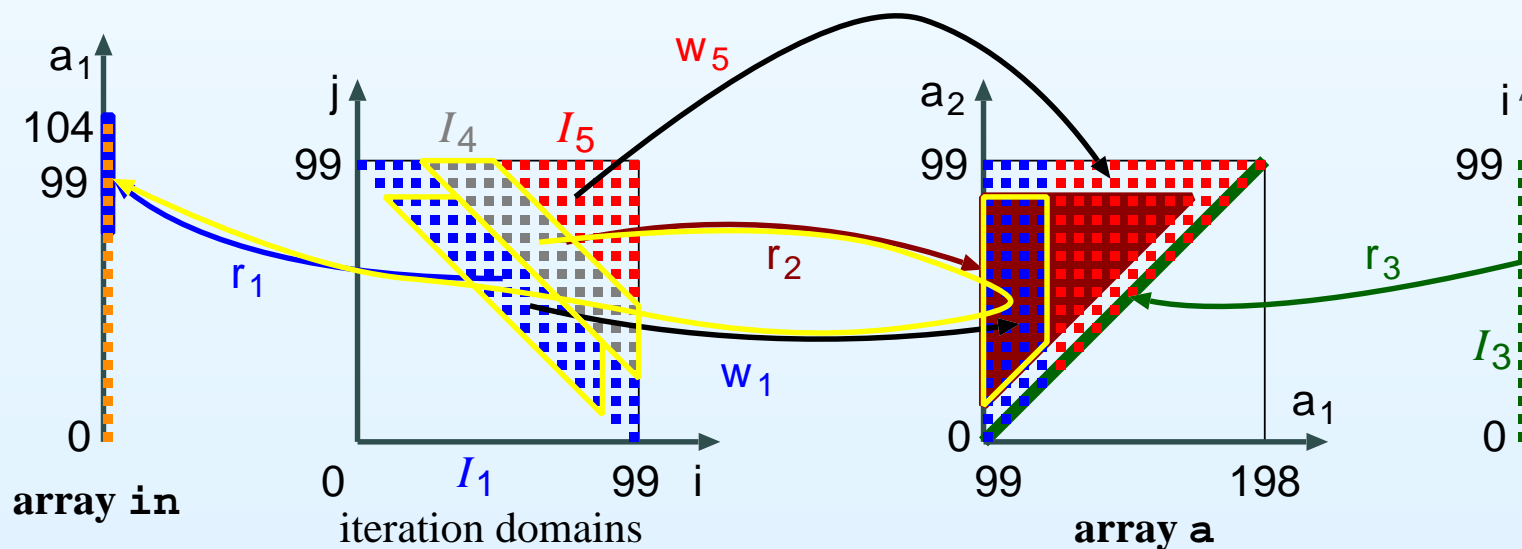


Non-recursive propagation : S1 to S3

```

for (i = 0; i < 100; i++)
  for (j = 99 - i; j < 100; j++)
    if (i + j < 105) {
      a[j + i][j] = in[j + i];           // S1
    }
    else
      a[j + i][j] = a[j + i - 5][j - 3]; // S2
for (i = 0; i < 100; i++)
  out[i] = f(a[i + 99][i]);           // S3

```

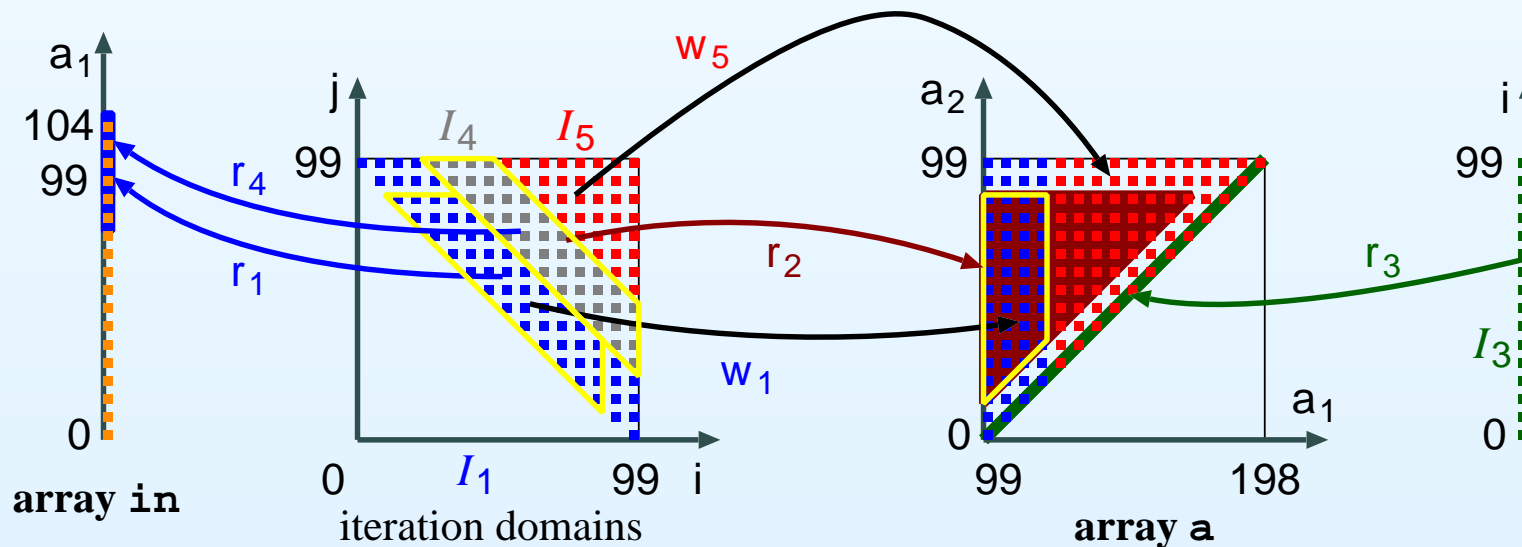


Non-recursive propagation : S1 to S2

```

for (i = 0; i < 100; i++)
  for (j = 99 - i; j < 100; j++)
    if (i + j < 105) {
      a[j + i][j] = in[j + i];           // S1
    }
    else
      a[j + i][j] = a[j + i - 5][j - 3]; // S2
for (i = 0; i < 100; i++)
  out[i] = f(a[i + 99][i]);           // S3

```

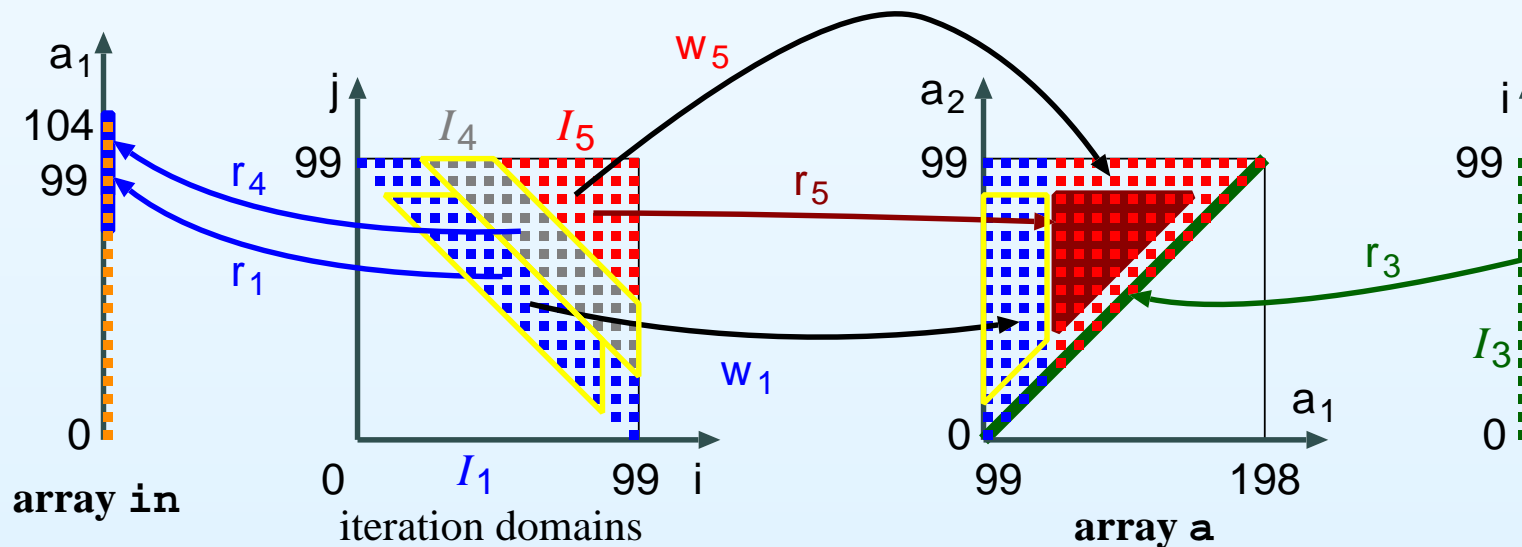


Non-recursive propagation : S1 to S2

```

for (i = 0; i < 100; i++)
  for (j = 99 - i; j < 100; j++)
    if (i + j < 105) {
      a[j + i][j] = in[j + i];           // S1
    }
    else
      a[j + i][j] = a[j + i - 5][j - 3]; // S2
for (i = 0; i < 100; i++)
  out[i] = f(a[i + 99][i]);           // S3

```

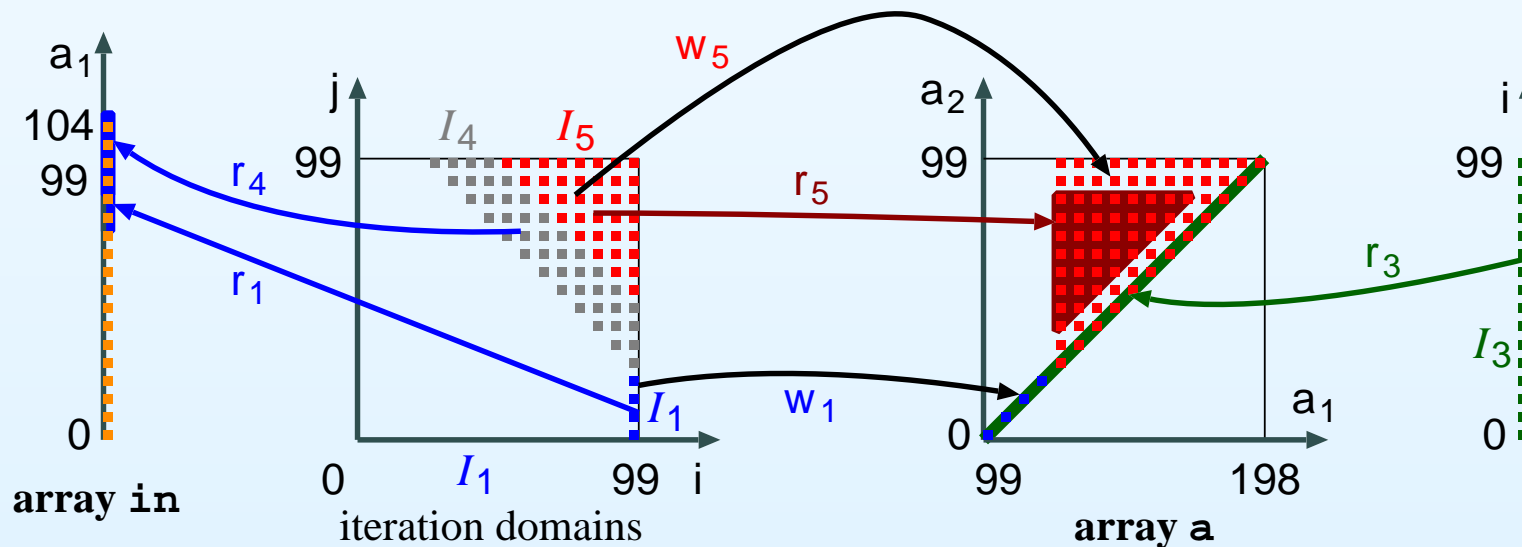


Non-recursive propagation : S1 to S3

```

for (i = 0; i < 100; i++)
  for (j = 99 - i; j < 100; j++)
    if (i + j < 105) {
      a[j + i][j] = in[j + i];           // S1
    }
    else
      a[j + i][j] = a[j + i - 5][j - 3]; // S2
for (i = 0; i < 100; i++)
  out[i] = f(a[i + 99][i]);           // S3

```

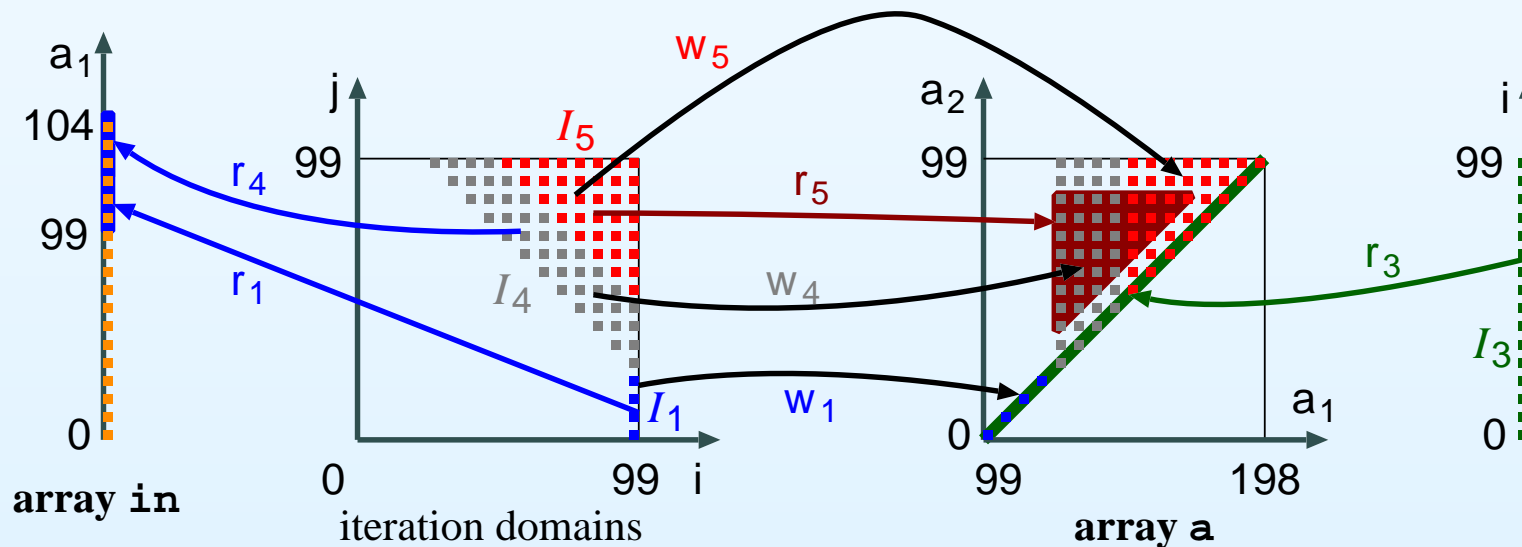


Non-recursive propagation : S1 to S2

```

for (i = 0; i < 100; i++)
  for (j = 99 - i; j < 100; j++)
    if (i + j < 105) {
      if (i == 99)
        a[j + i][j] = in[j + i];           // s1
    } else
      if (j + i < 110)
        a[j + i][j] = in[j + i - 5];      // s4
      else
        a[j + i][j] = a[j + i - 5][j - 3]; // s5
for (i = 0; i < 100; i++)
  out[i] = f(a[i + 99][i]);              // s3

```

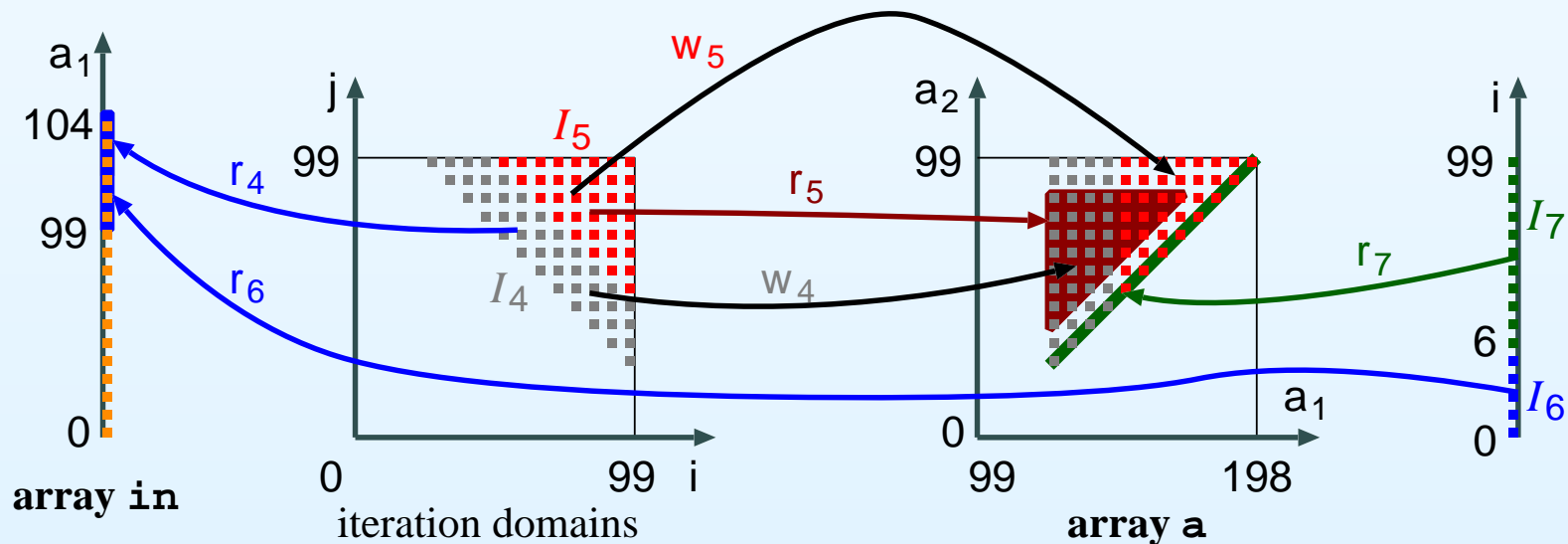


Non-recursive propagation : S1 to S3

```

for (i = 0; i < 100; i++)
  for (j = 0; j < 100; j++)
    if (i + j >= 105)
      if (j + i < 110)
        a[j + i][j] = in[j + i - 5];          // S4
      else
        a[j + i][j] = a[j + i - 5][j - 3]; // S5
for (i = 0; i < 100; i++)
  if (i < 6)
    out[i] = f(in[i + 99]);                  // S6
  else
    out[i] = f(a[i + 99][i]);                // S7

```

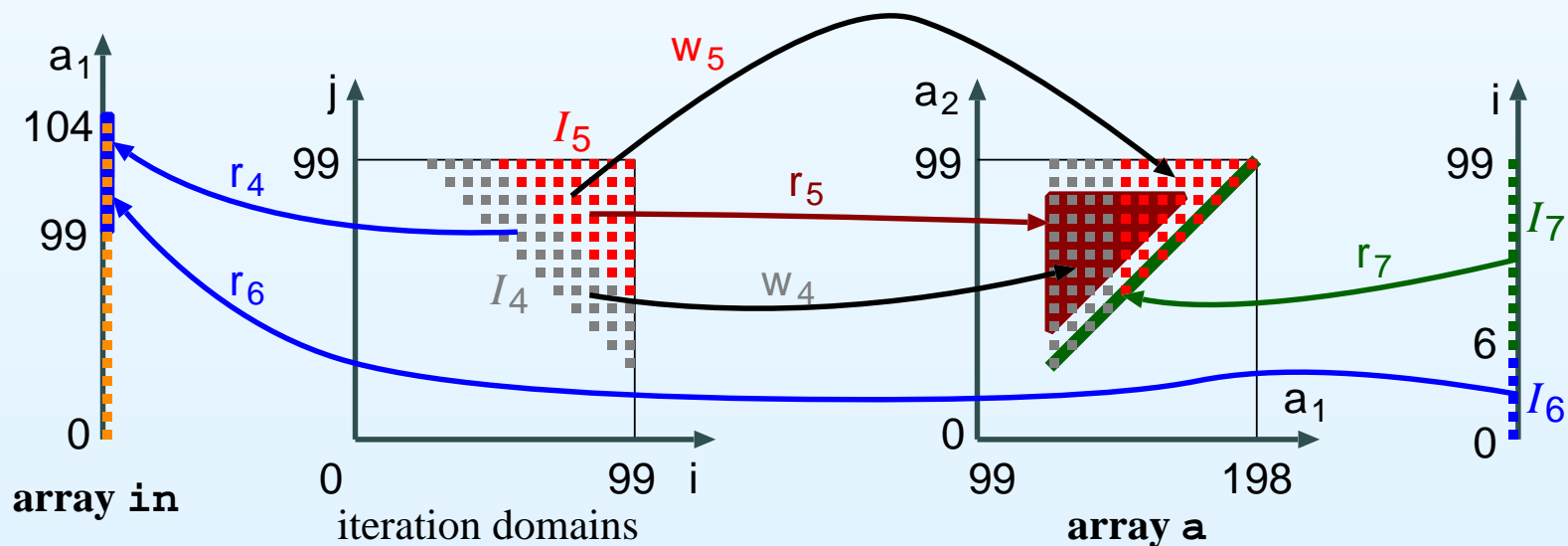


Recursive propagation : S5 to S5

```

for (i = 0; i < 100; i++)
  for (j = 0; j < 100; j++)
    if (i + j >= 105)
      if (j + i < 110)
        a[j + i][j] = in[j + i - 5];          // S4
      else
        a[j + i][j] = a[j + i - 5][j - 3]; // S5
for (i = 0; i < 100; i++)
  if (i < 6)
    out[i] = f(in[i + 99]);                  // S6
  else
    out[i] = f(a[i + 99][i]);                // S7

```

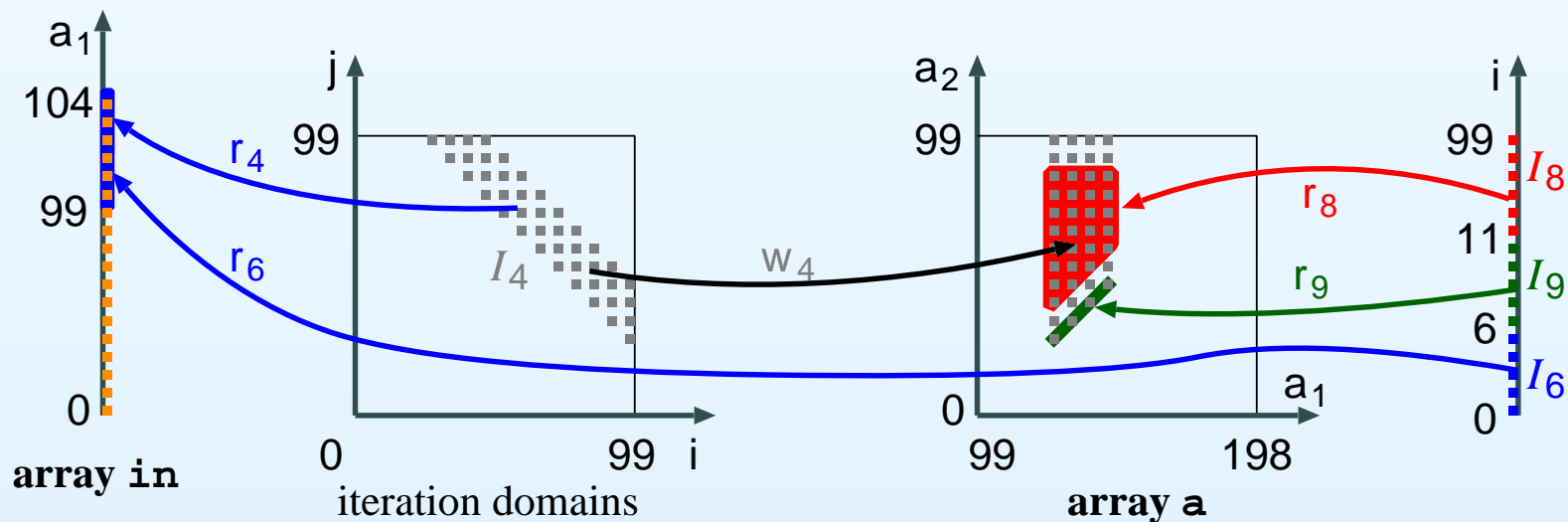


Recursive propagation : finishing

```

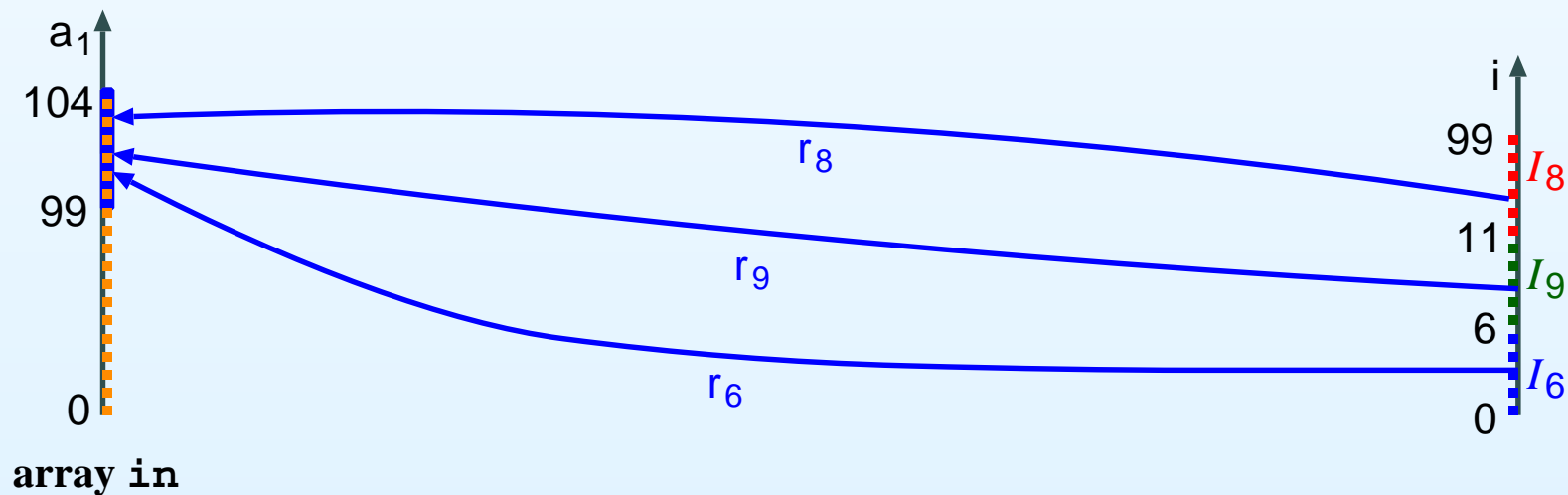
for (i = 0; i < 100; i++)
  for (j = 0; j < 100; j++)
    if (i + j >= 105 && j + i < 110)
      a[j + i][j] = in[j + i - 5]; // S4
for (i = 0; i < 100; i++)
  if (i < 6)
    out[i] = f(in[i + 99]); // S6
  else if (i >= 11)
    out[i] = f(a[i + 99 - 5 * ((i - 6) / 5)] // S8
              [i - 3 * ((i - 6) / 5)]);
  else
    out[i] = f(a[i + 99][i]); // S9

```



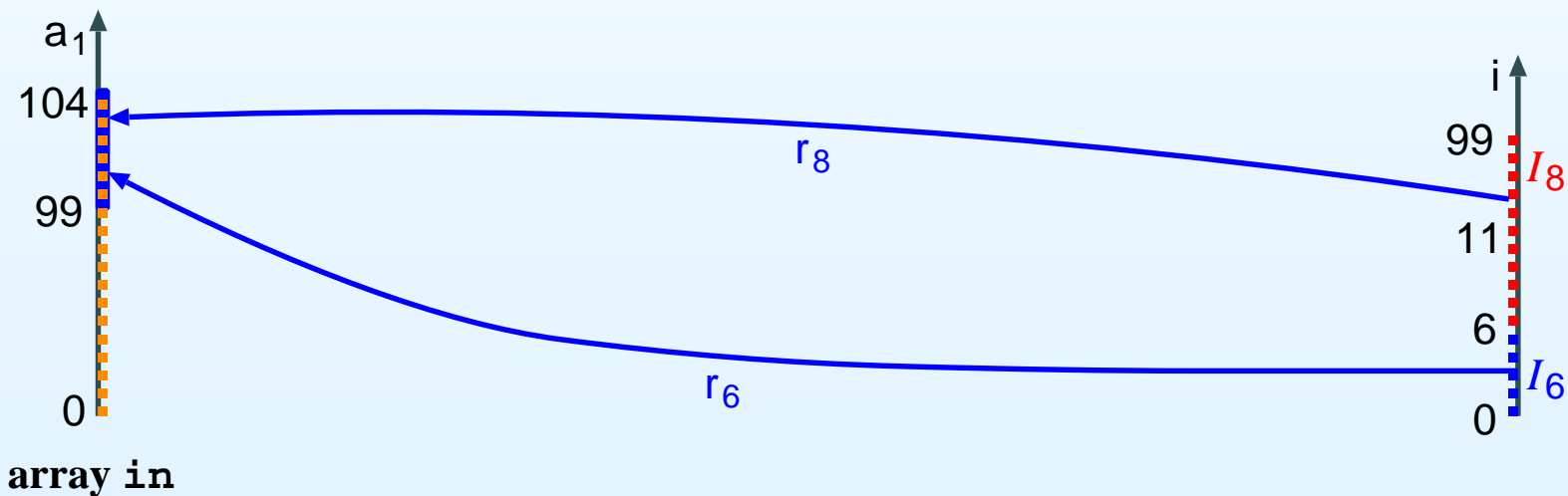
Recursive propagation : finishing

```
for (i = 0; i < 100; i++)  
  if (i < 6)  
    out[i] = f(in[i + 99]);           // S6  
  else if (i >= 11)  
    out[i] = f(in[i + 94 - 5 * ((i - 6) / 5)]); // S8  
  else  
    out[i] = f(in[i + 94]);          // S9
```



Recursive propagation : finishing

```
for (i = 0; i < 100; i++)  
  if (i < 6)  
    out[i] = f(in[i + 99]);           // S6  
  else  
    out[i] = f(in[i + 94 - 5 * ((i - 6) / 5)]); // S8
```



Implementation

- Implemented using PolyLib (developed at IRISA Rennes)
- Execution time of the polyhedral operations depends only on dimension of polyhedra
- Execution time is $O(k \cdot c + r \cdot k^2)$ where
 - r is the number of direct and indirect recursions
 - c is the number of read statements from arrays that are copies from other arrays
 - k is a factor depending on the complexity of the program
- If we can guarantee a small upper bound on k , execution time is $O(c + r)$

Results

Name	Δacc	Δmem	t (s)
LU1	26%	40%	0.191
LU2	26%	45%	0.332
im1	43%	56%	0.367
im2	94%	100%	1.700
mp3	22%	37% ²	0.929
divx	31%	100% 0	0.134

Legend

Δacc the decrease in memory accesses (reads + writes)

Δmem decrease in memory use for program in DSA-form,
equivalent to decrease in write operations

t transformation time in seconds

Conclusions

- By using DSA, we have developed a simple, elegant method to do copy propagation for arrays
- DSA is only an intermediate representation and so translation out of DSA is necessary, but this reduces memory size often beyond original size
- We can distinguish between run-time instances of statements
- By representing the program in the polyhedral model, we can use a polyhedral library to do all calculations
- For the moment all copy operations are removed if possible, without regard for complexity
- Further research is needed for the trade-off between removing copy operations and complexity