DAEDALUS

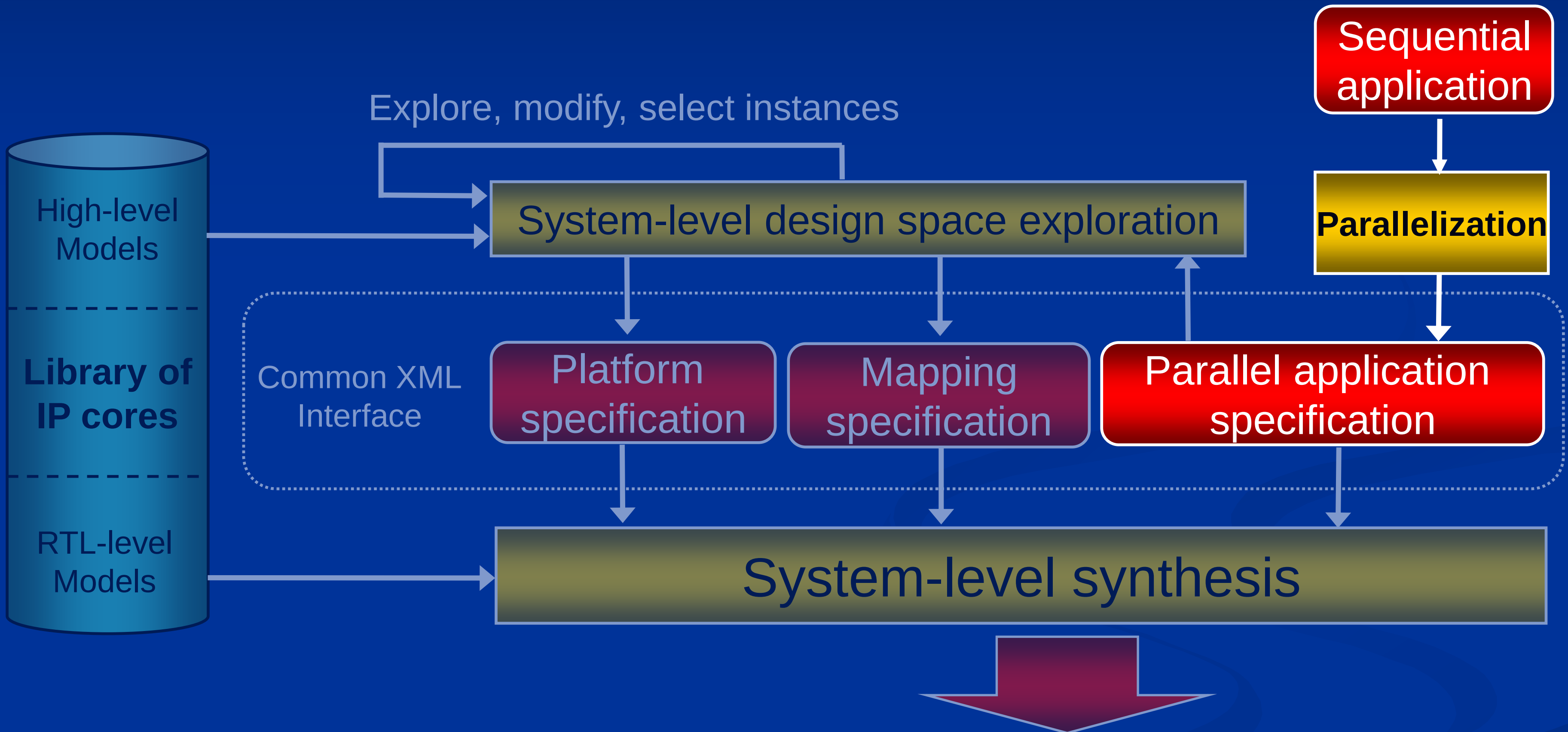
Automated Derivation of Polyhedral Process Networks

Todor Stefanov

Leiden Embedded Research Center,
Leiden Institute of Advanced Computer Science
Leiden University, The Netherlands

Universiteit Leiden

Automated Parallelization: PNgen tool



Multi-processor System on Chip
(Synthesizable VHDL and C/C++ code for processors)

Programming problem

EASY to specify

Static Affine Nested Loop Programs

```
for j = 1:1:N,
  [x(j)] = Source1( );
end
for i = 1:1:K,
  [y(i)] = Source2( );
end
for j = 1:1:N,
  for i = 1:1:K,
    [y(i), x(j)] = F( y(i), x(j) );
  end
end
for i = 1:1:K,
  [Out(i)] = Sink( y( I ) );
end
```

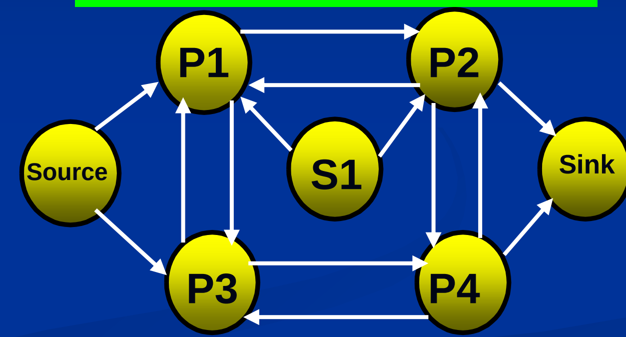
DIFFICULT to map

Application

PNgen tool

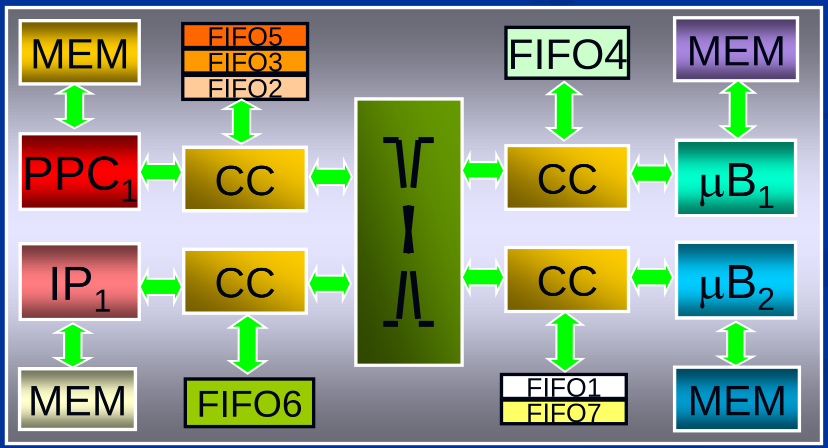
DIFFICULT to specify

Polyhedral Process Networks



EASY to map

ESPAM tool



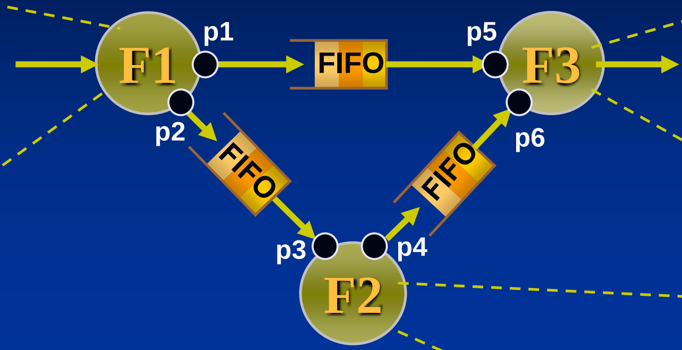
Static Affine Nested Loop Programs

- Restrictions to top-level program code
  - Parameters are ***symbolic constants***, i.e., do not change at run-time
  - *Only* **FOR-loops** with bounds that are affine functions of other loops' indices and parameters
  - *Only* **If-statements** with conditions that are affine functions of loops' indices and parameters
  - *Only* **Explicit data exchange** via arrays or scalars (*NO pointers*)
    - **Arrays are indexed with affine functions** of loops' indices and parameters
- NO restrictions for code in function calls

```
int N =  10;
#pragma parameter N 5 100
int main(void) {
   int  i, j, k;
   MyType   A[600];

   for ( k = 1; k <= 6*N-3; k++ ) {
        A[k] = Func1( );
   }

   for ( j = 1; j <= N; j++ ) {
      for ( i = j; i <= 3*j - 2; i++ ) {
          if ( i + j <= 4*N - 6 ) {
               A[i] = Func2( A[2*i-1],  A[2*i+1]);
          }
          Func3( A[i] );
      }
   } }
```

Polyhedral Process Networks (1)

```
int M = 10, P = 3;
for( i=1; i <= M; i++) {
   out = F1( );
   if( i < = P)
      write( p2, out );
   else
      write( p1, out );
}
```

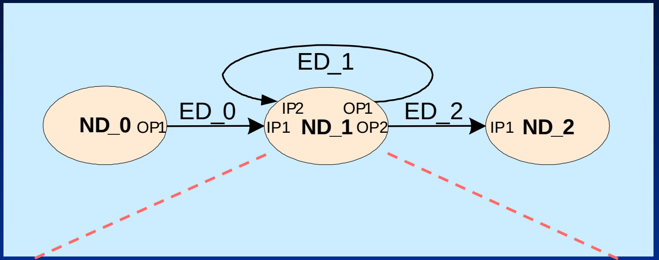



```
int N = 10, P = 3;
for( j=1; j <= N; j++) {
   if( j < = P)
      in = read( p6 );
   else
      in = read( p5 );
   F3( in );
}
```

```
int P = 3;
for( j=1; j <= P; j++) {
      in = read( p3 );
      out = F2( in );
      write( p4, out );
}
```

- Simple formalism to expresses concurrency
  - Autonomously running processes
  - Communicating via bounded FIFOs
  - Synchronization via blocking read/write
- Deterministic
  - For one and the same input, one and the same output is produced, irrespective of the execution order of processes
- Distributed Control
  - no global schedule needed

Polyhedral Process Networks (2)



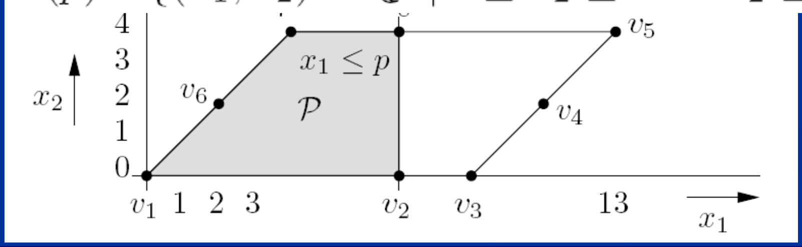

```
// process ND_1
void main( ) {
  for( int  i=2; i<=M; i++ )                          CONTROL
    for( int  j=2; j<=N; j++ ) {

      if( j-2 == 0 )
        read( IP1, in_0 );                            READ
      if( j-3 >= 0 )
        read( IP2, in_0 );

      Transformer( in_0, out_0);                      EXECUTE

      if( -j+N-1 >= 0 )
        write( OP1, out_0 );                          WRITE
      if( j-N == 0 ) {
        write( OP2, out_0 );
    } // for j
} // main
```

- Well defined structure of a process
  - Separate READ - EXECUTE - WRITE code sections
  - CONTROL – determines the number of iterations/firings of a process
- Every Process, Input and Output Port can be represented as Parameterized Polyhedrons
- Parameterized Polyhedron

$$\mathcal{P}(\mathbf{p}) = \{\mathbf{x} \in \mathbb{Q}^n \mid \boxed{A\mathbf{x} = B\mathbf{p} + \mathbf{b} \wedge C\mathbf{x} \geq D\mathbf{p} + \mathbf{d}}\}$$

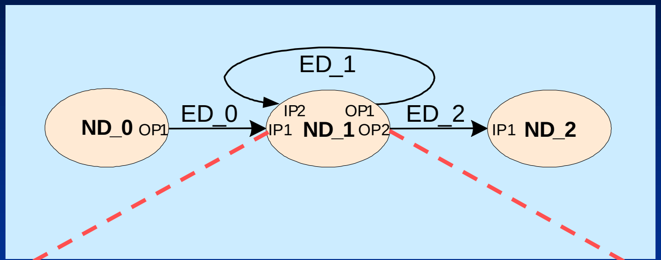
  - Set of points ***x*** in the n-dimensional space satisfying some affine constraints
  - $\mathbf{p} \in \mathbb{Q}^m$ is a vector of parameters

$$\mathcal{P}(p) = \{(x_1, x_2) \in \mathbb{Q}^2 \mid 0 \leq x_2 \leq 4 \ \wedge\ x_2 \leq x_1 \leq x_2 + 9 \ \wedge\ x_1 \leq p \ \wedge\ p \leq 40\}$$



Polyhedral Process Networks (3)

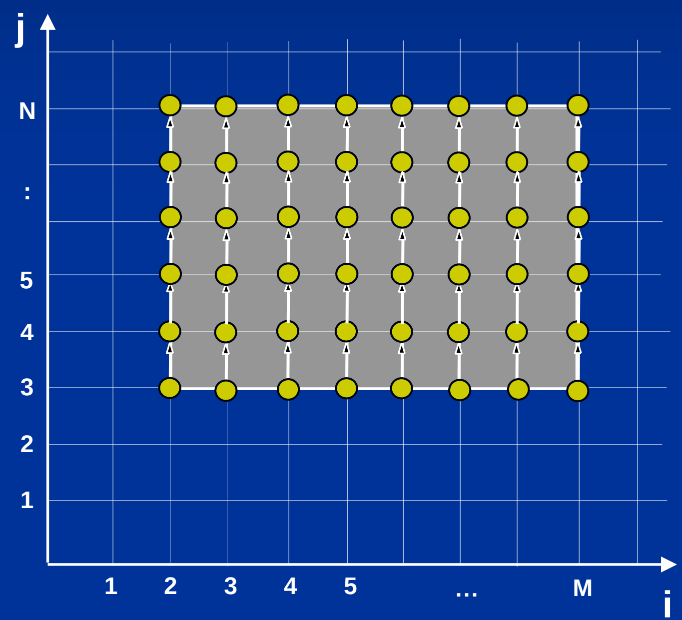
- Example: Input port **IP2** as parameterized polyhedron




```
// process ND_1
void main( ) {
  for( int  i=2; i<=M; i++ )                         CONTROL
    for( int  j=2; j<=N; j++ ) {

        if( j-2 == 0 )
          read( IP1, in_0 );                          READ
        if( j-3 >= 0 )
          read( IP2, in_0 );

        Transformer( in_0, out_0);                    EXECUTE

        if( -j+N-1 >= 0 )
          write( OP1, out_0 );                        WRITE
        if( j-N == 0 ) {
          write( OP2, out_0 );
    } // for j
} // main
```

$$2 \le i \le M,\quad 2 \le j \le N,\quad j - 3 \geq 0$$

$$2 \le i \le M,\quad 3 \le j \le N,$$



$$P(M,N) = \left\{ (i,j) \in Z^2 \mid \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} * \begin{pmatrix} i \\ j \end{pmatrix} \geq \begin{bmatrix} 0 & 0 \\ -1 & 0 \\ 0 & 0 \\ 0 & -1 \end{bmatrix} * \begin{pmatrix} M \\ N \end{pmatrix} + \begin{pmatrix} 2 \\ 0 \\ 3 \\ 0 \end{pmatrix} \right\}$$

Deriving Polyhedral Process Network

Sequential Program

Dependence Analysis

SAC or PDG

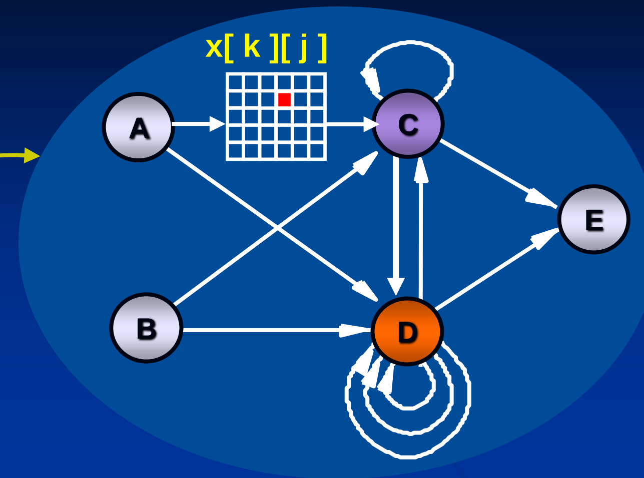
Linearization

Polyhedral Process Network

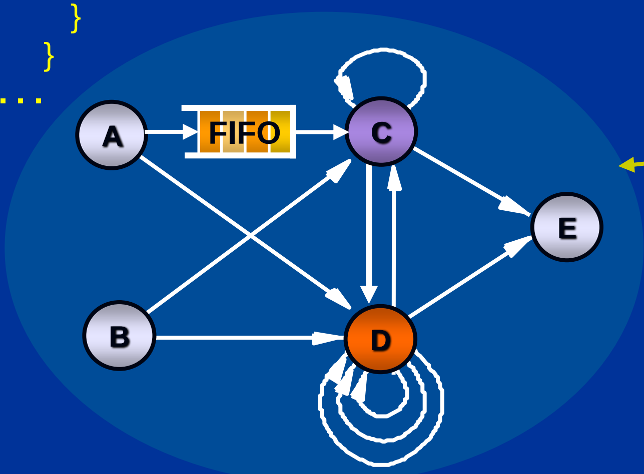
FIFO size calculation

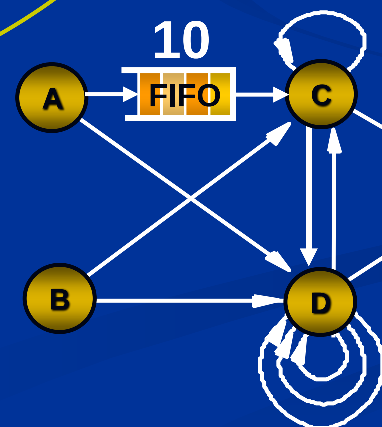
Polyhedral Process Network (optimized)

```
int N = 5;
#pragma parameter N 4 16;
Int K = 100;
#pragma parameter K 100 1000;

. . .
for( k=1; k<=K; k++ )
   for ( j=1; j<=N; j++ ) {
      t = C( r[ j ][ j ], x[ k ][ j ], &r[ j ][ j ], &x[ k ][ j ] );
      for( i = j+1; i<=N; i++ ) {
         t = D( t, r[ j ][ i ], x[ k ][ i ], &r[ j ][ i ], &x[ k ][ i ] );
      }
   }
. . .
```



Polyhedral Dependence Graph



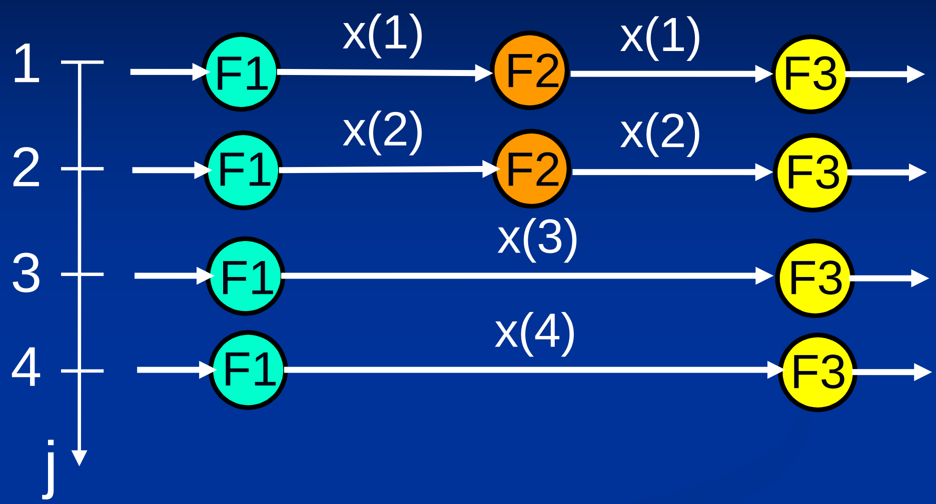


Dependence Analysis (1)

Static affine program

```
for j = 1:1:4,
   [x(j)] = F1( ... );
end
for j = 1:1:4,
  if j < = 2,
     [x(j)] = F2( x(j) );
  end
  [...] = F3( x(j) );
end
```

Dependence Graph (DG)



Exact Array Dataflow Analysis

Q: Given a read from an array element, what was the last write to that array element?

Example: given a read <F3 @ j=2> from x(2) what was the last write <F? @ j=?> to x(2)

A: Can be computed using Parametric Integer Programming (PIP)

$\Rightarrow$ **finds parametric lexicographically maximal element of a set bounded by linear constraints**

Dependence Analysis (2)

Static affine program

```
for j = 1:1:4,
  [x(j)] = F1( … );
end
for j = 1:1:4,
  if j < = 2,
    [x(j)] = F2( x(j) );
  end
  […] = F3( x(j) );
end
```

Dependence analysis for **F3 ← F2**

subject to:

$$1 \le j_W \le 4$$
$$j_W \le 2$$
$$j_W = j_R$$
$$1 \le j_R \le 4$$

objective: $j_W^{max} = \max_{lex}\{ j_W(j_R)\}$

PIP Solution:

if ($j_R \le 2$) **then** $j_W = j_R$
else *no solution;*

Meaning of the Solution:

if ($j_R \le 2$) **then**
<F3 @ j_R> depends on <F2 @ $j_W = j_R$ > via variable *X*($j_W = j_R$)
else
<F3 @ j_R> depends on other functions

Dependencies expressed as SSAC

Static affine program

```
for j = 1:1:4,
  [x(j)] = F1( ... );
end
for j = 1:1:4,
  if j <= 2,
    [x(j)] = F2( x(j) );
  end
  [...] = F3( x(j) );
end
```

Exact Array Dataflow Analysis

Properties of SSAC:
- every variable is written only once
- expose all dependencies

Static Single Assignment Code

```
for j = 1:1:4,
  [x_1(j)] = F1( ... );
end
for j = 1:1:4,
  if j <= 2,
    [x_2(j)] = F2( x_1(j) );
  end
  if j <= 2,
    in_0 = x_2(j) ;
  else
    in_0 = x_1(j) ;
  end
  [...] = F3( in_0 );
end
```

Meaning of the Solution:

if ($j_R \le 2$) **then**

<F3 @ j_R> depends on <F2 @ $j_W = j_R$ > via variable *X*($j_W = j_R$)

else

<F3 @ j_R> depends on other functions

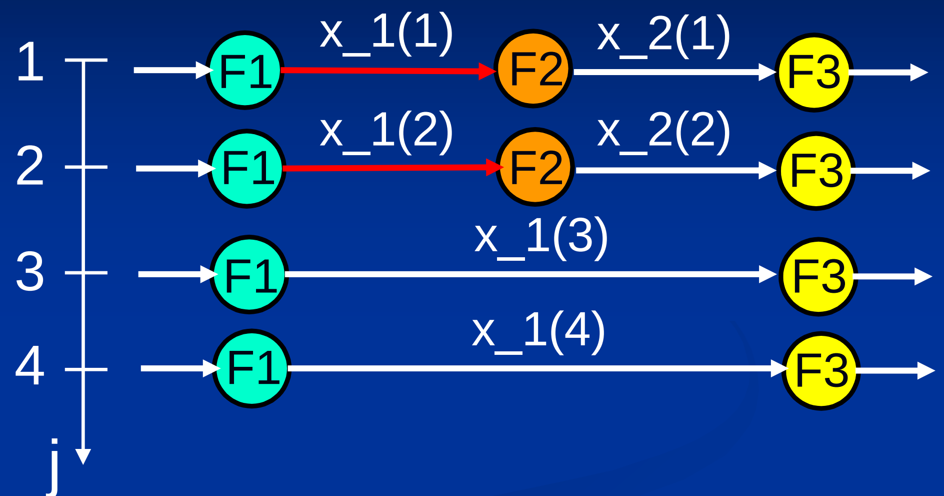
Dependencies expressed as PDG

Static Single Assignment Code

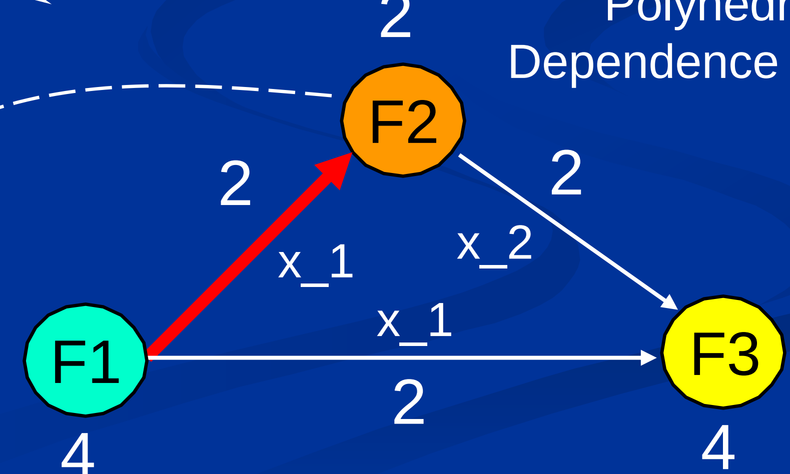
```
for j = 1:1:4,
   [x_1(j)] = F1( … );
end
for j = 1:1:4,
  if j < = 2,
     [x_2(j)] = F2( x_1(j) );
  end
  if j < = 2,
     in_0 =  x_2(j) ;
  else
     in_0 =  x_1(j) ;
  end
  […] = F3( in_0 );
end
```

unroll

Dependence Graph (DG)



Polyhedral Dependence Graph



Polyhedral annotation:

$P_{F2} = \{ j \in Z \mid 1 <= j <= 2 \} = \{1,2\}$

Deriving Polyhedral Process Networks

Sequential Program

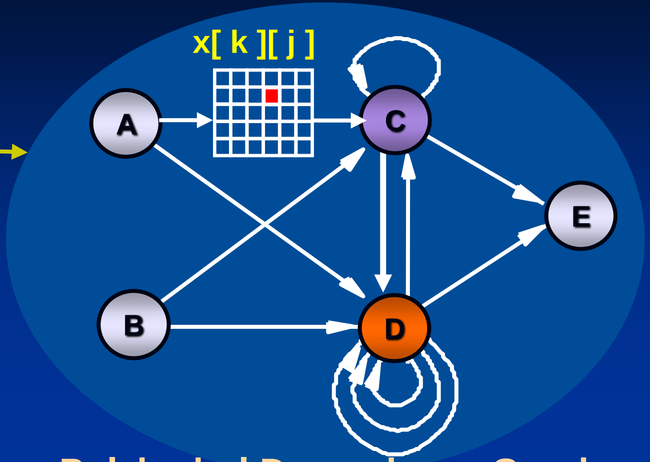
Dependence Analysis

SAC or PDG

Linearization

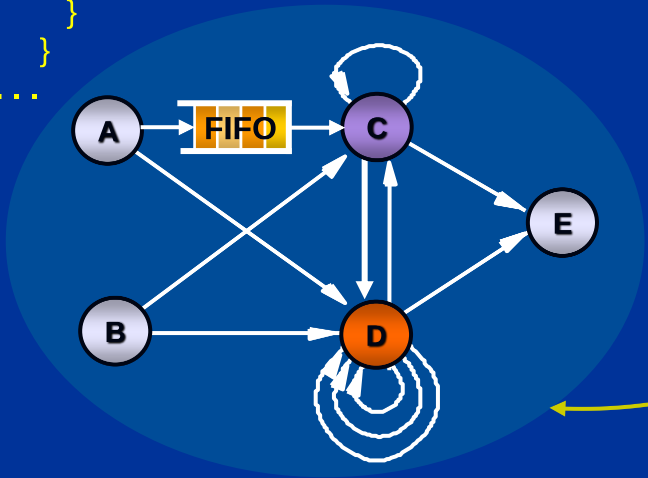
Polyhedral Process Network

```
int N = 5;
#pragma parameter N 4 16;
Int K = 100;
#pragma parameter K 100 1000;

. . .

for( k=1; k<=K; k++ )
   for ( j=1; j<=N; j++ ) {
      t = C( r[ j ][ j ], x[ k ][ j ], &r[ j ][ j ], &x[ k ][ j ] );
      for( i = j+1; i<=N; i++ ) {
         t = D( t, r[ j ][ i ], x[ k ][ i ], &r[ j ][ i ], &x[ k ][ i ] );
      }
   }
. . .
```



Polyhedral Dependence Graph

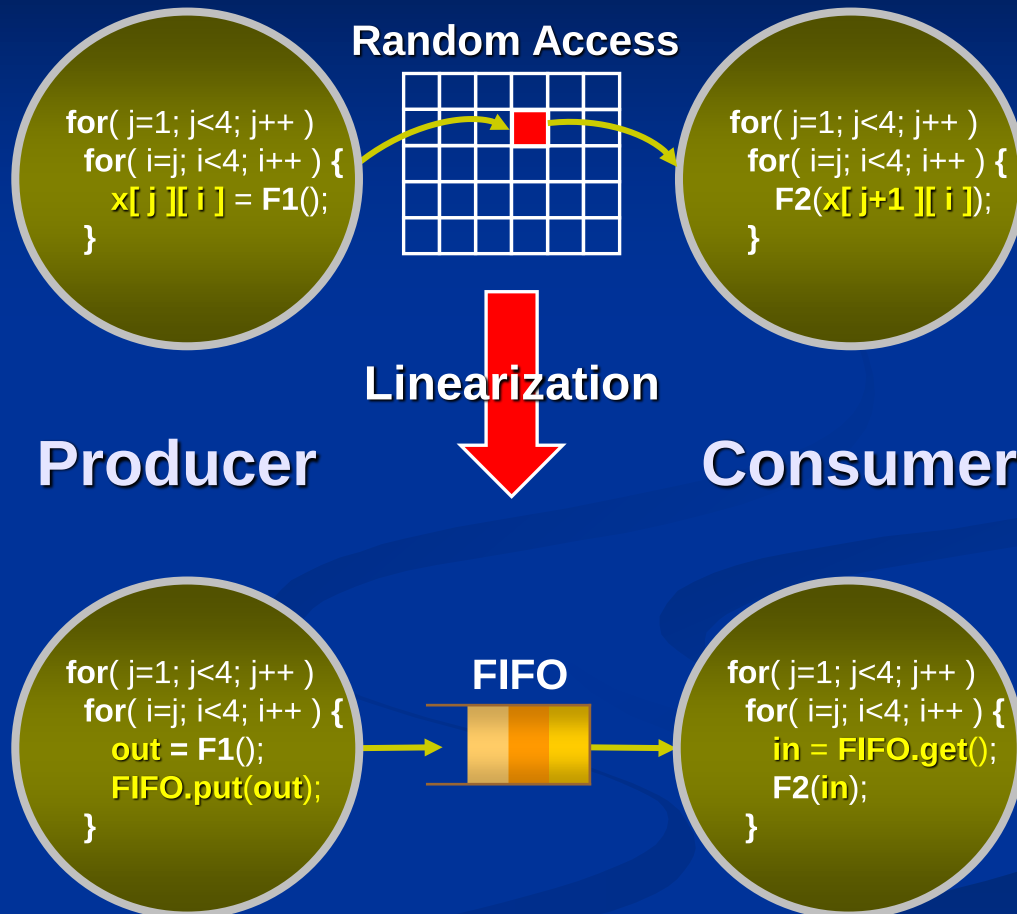
Dependencies via n-dimensional arrays with random access!!!



Linearization

- Linearization: mapping high-order data-structures, e.g., N-dimensional array onto 1-dimensional stream

- Indexing of **x(j,i)** and **x(j+1,i)** is replaced by relative put and get operation on a FIFO buffer

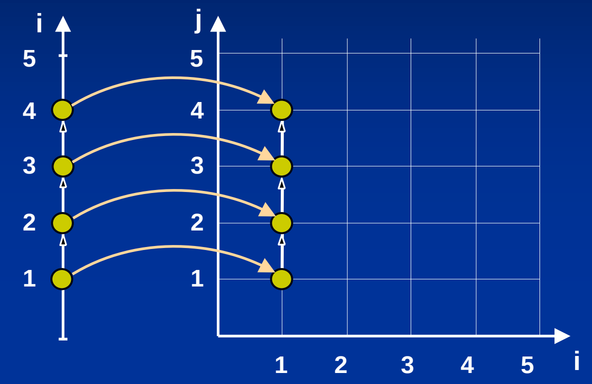


What is actually the problem? Is Linearization always possible?

Possible Communication Scenarios

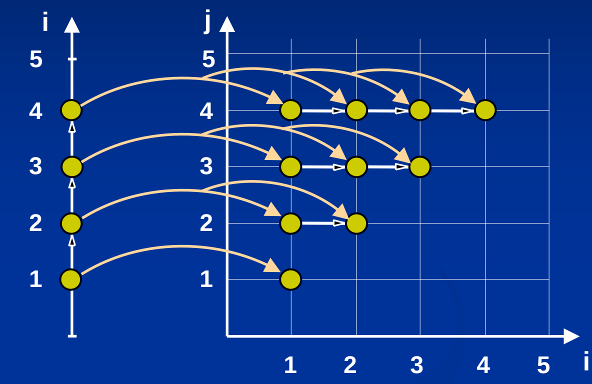
In-order

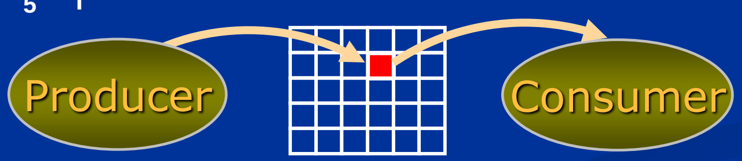
Producer Consumer



In-order with multiplicity

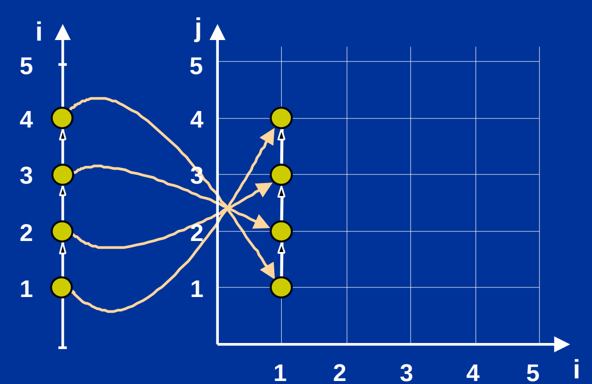
Producer Consumer





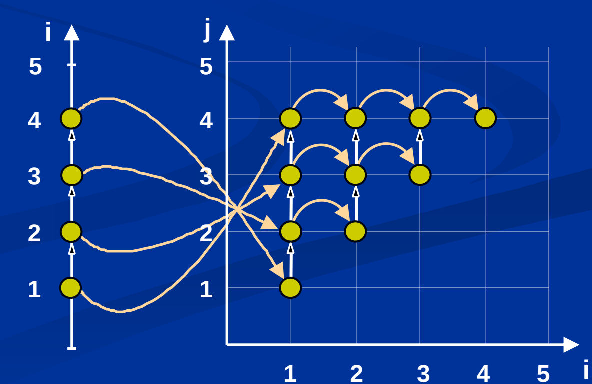
Out-of-order

Producer Consumer



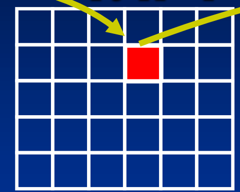
Out-of-order with multiplicity

Producer Consumer



In-order Communication

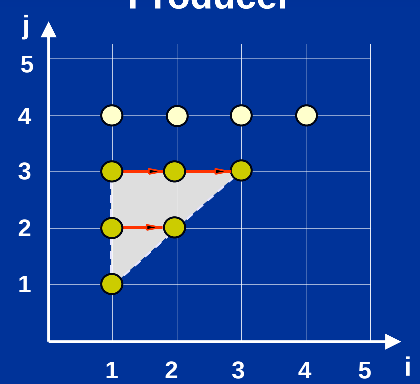
X[j][i]



```
for( j=1; j<4; j++ )
  for( i=1; i<=j; i++ ) {
    x[ j ][ i ] = F1();
  }
```

```
for( j=2; j<5; j++ )
  for( i=1; i<j; i++ ) {
    F2(x[ j-1 ][ i ]);
  }
```

Producer

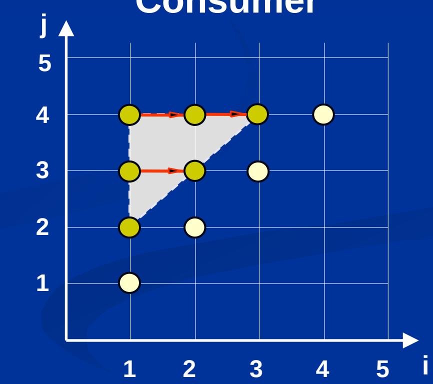


The sequence in which the data is produced

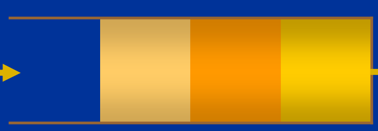
(1,1)	(2,1)	(2,2)	(3,1)	(3,2)	(3,3)
X[1][1]	X[2][1]	X[2][2]	X[3][1]	X[3][2]	X[3][3]
(2,1)	(3,1)	(3,2)	(4,1)	(4,2)	(4,3)

The sequence in which the data is consumed

Consumer




```
for( j=1; j<4; j++ )
  for( i=1; i<=j; i++ ) {
    out = F1();
    FIFO.put(out);
  }
```

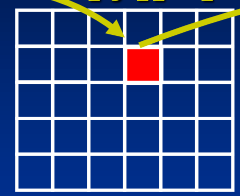


```
for( j=2; j<5; j++ )
  for( i=1; i<j; i++ ) {
    in = FIFO.get();
    F2(in);
  }
```

If data is consumed in the same order as it was produced, N–dimensional array CAN be safely replaced by FIFO

Out-of-order Communication

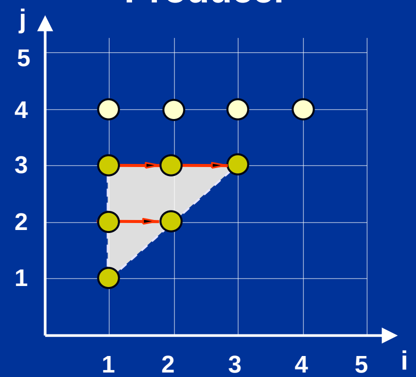
X[j][i]



```
for( j=1; j<4; j++ )
  for( i=1; i<=j; i++ ) {
    x[ j ][ i ] = F1();
  }
```

```
for( i=1; i<4; i++ )
  for( j=i+1; j<5; j++ ) {
    F2(x[ j-1 ][ i ]);
  }
```

Producer



The sequence in which the data is produced

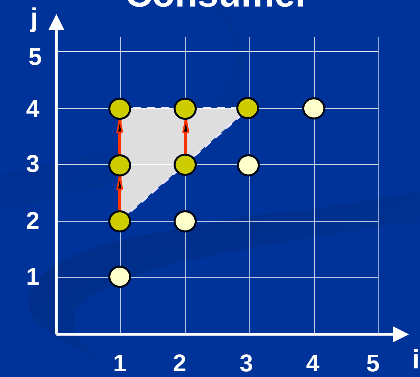
(1,1) (2,1) (2,2) (3,1) (3,2) (3,3)

X[1][1] X[2][1] X[2][2] X[3][1] X[3][2] X[3][3]

(2,1) (3,1) (4,1) (3,2) (4,2) (4,3)

The sequence in which the data is consumed:

Consumer



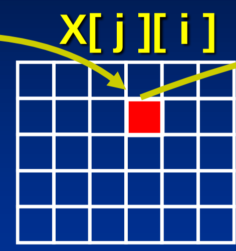
Observation: Data X[3][1] is produced ***after*** data X[2][2]

BUT

Data X[3][1] is consumed ***before*** data X[2][2]

If data is consumed in different order than it was produced, N–dimensional array CAN NOT be replaced by a simple FIFO

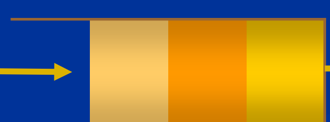
Implementing Out-of-Order



Producer

```
for( j=1; j<4; j++ )
  for( i=1; i<=j; i++ ) {
    x[ j ][ i ] = F1();
  }
```

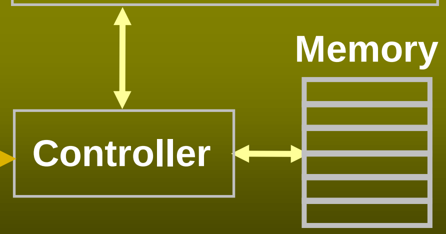
```
for( j=1; j<4; j++ )
  for( i=1; i<=j; i++ ) {
    out = F1();
    FIFO.put(out);
  }
```



Consumer

```
for( i=1; i<4; i++ )
  for( j=i+1; j<5; j++ ) {
    F2(x[ j-1 ][ i ]);
  }
```

```
for( i=1; i<4; i++ )
  for( j=i+1; j<5; j++ ) {
    in = Controller.get(i,j);
    F2(in);
  }
```

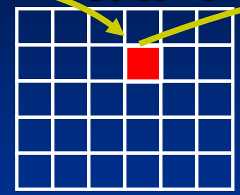


- The data is reordered at the consumer
  - Option1:
    - Simple FIFO
    - Reordering Controller
    - Reordering Memory
  - Option2:
    - Window FIFO
    - Reordering Controller

Identifying Out-of-Order (1)

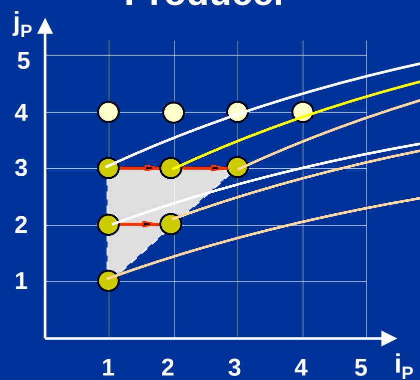
```
for( j=1; j<4; j++ )
  for( i=1; i<=j; i++ ) {
    x[ j ][ i ] = F1();
  }
```

X[j][i]

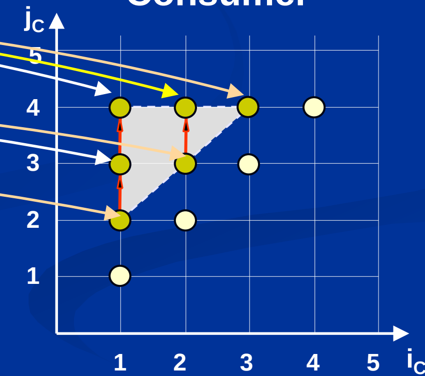


```
for( i=1; i<4; i++ )
  for( j=i+1; j<5; j++ ) {
    F2(x[ j-1 ][ i ]);
  }
```

Producer



Consumer

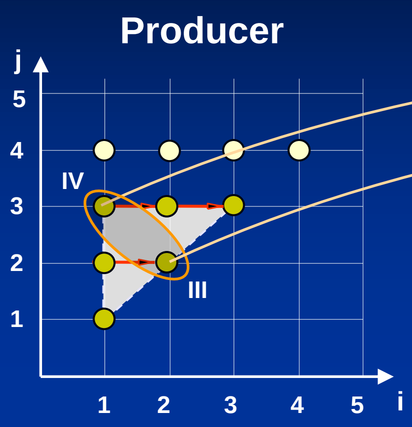


Observation: $j_p = j_c - 1$
$i_p = i_c$

$$(j_p , i_p) = M(j_c , i_c)$$

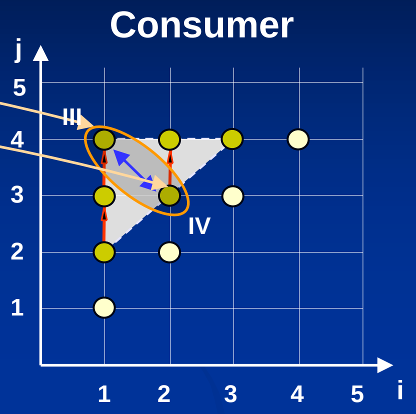
In general, mapping function M exists for any producer-consumer pair

Identifying Out-of-Order (2)



$$(j_p, i_p) = M(j_c, i_c)$$

Reordering *Integer Linear Problem*



Mapped 2 points in Producer:

$$(j_p^1, i_p^1) = M(j_c^1, i_c^1)$$

$$(j_p^2, i_p^2) = M(j_c^2, i_c^2)$$

$$i_c^1 < j_c^1 \le 4$$

$$1 \le i_c^1 \le 3$$

$$i_c^2 < j_c^2 \le 4$$

$$1 \le i_c^2 \le 3$$

$$(j_c^1, i_c^1) <_{lex} (j_c^2, i_c^2)$$

$$M(j_c^1, i_c^1) \;{}_{lex}\!> M(j_c^2, i_c^2)$$

Any 2 points in Consumer:

$$(j_c^1, i_c^1)$$

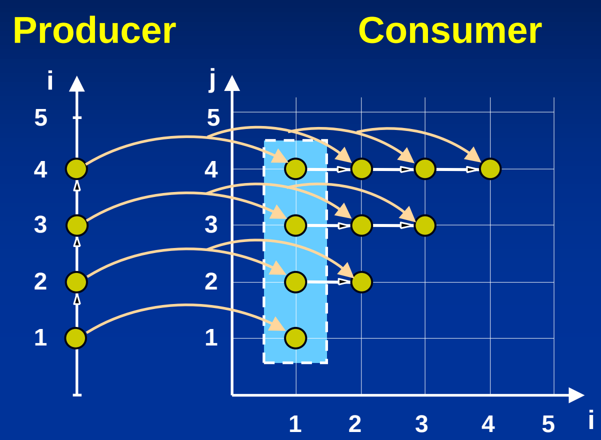
$$(j_c^2, i_c^2)$$

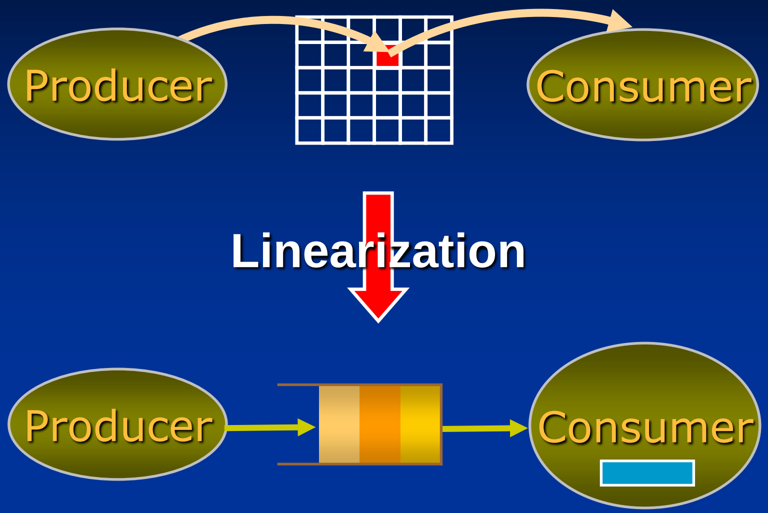
IF *Reordering Problem has a solution* **THEN** ***Out-of-Order*** *communication*

ELSE **In-Order** communication

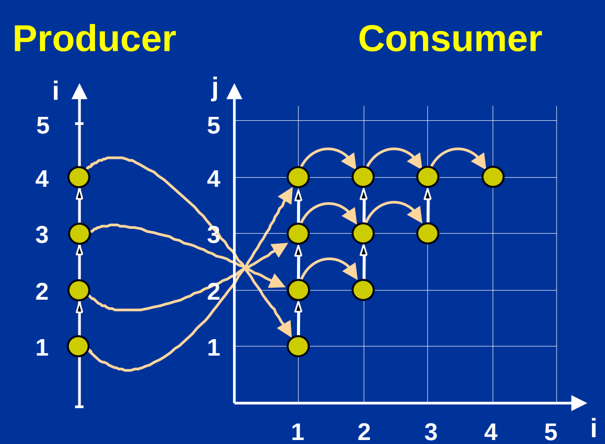
Multiplicity

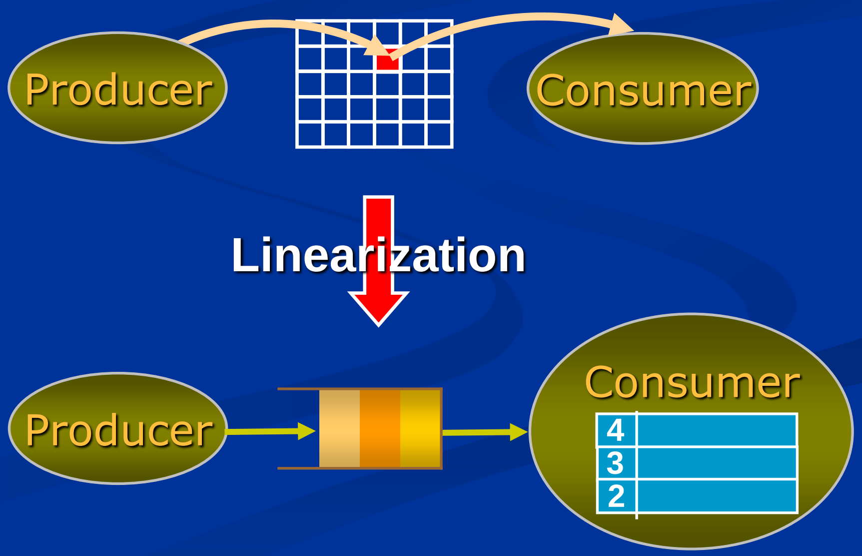
In-order with multiplicity



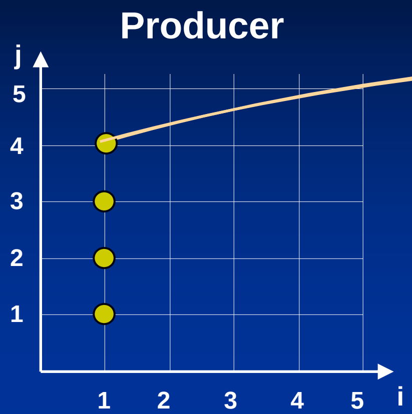


Out-of-order with multiplicity

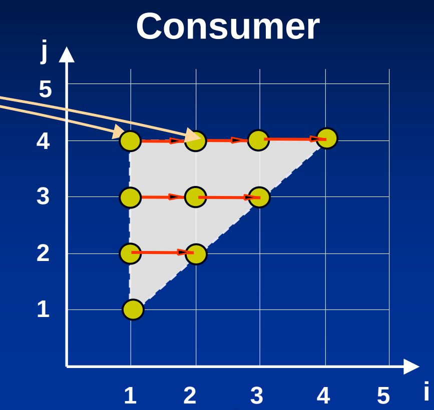




Identifying Multiplicity



$$(j_p, i_p) = M(j_c, i_c)$$



Multiplicity *Integer Linear Problem*

Mapped 2 points in Producer:

$$(j_p^1, i_p^1) = M(j_c^1, i_c^1)$$
$$(j_p^2, i_p^2) = M(j_c^2, i_c^2)$$

$$1 \le j_c^1 \le 4$$
$$1 \le i_c^1 \le j_c^1$$
$$1 \le j_c^2 \le 4$$
$$1 \le i_c^2 \le j_c^2$$
$$(j_c^1, i_c^1) \neq (j_c^2, i_c^2)$$
$$M(j_c^1, i_c^1) = M(j_c^2, i_c^2)$$

Any 2 points in Consumer:

$$(j_c^1, i_c^1)$$
$$(j_c^2, i_c^2)$$

IF *Multiplicity Problem has a solution* **THEN** *there is **Multiplicity***

Deriving Polyhedral Process Networks

Sequential Program

Dependence Analysis

SAC or PDG

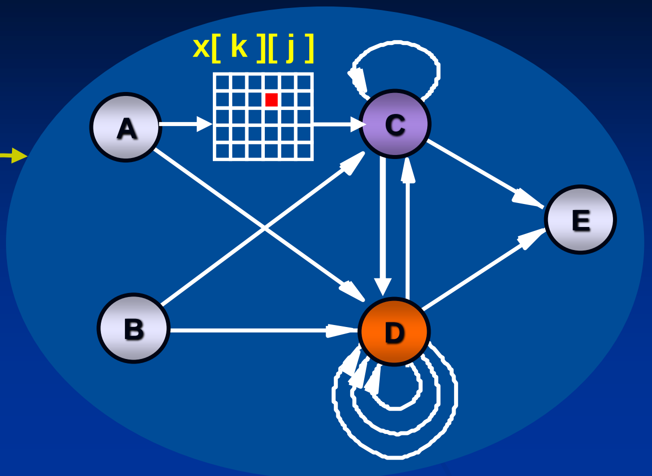
Linearization

Polyhedral Process Network

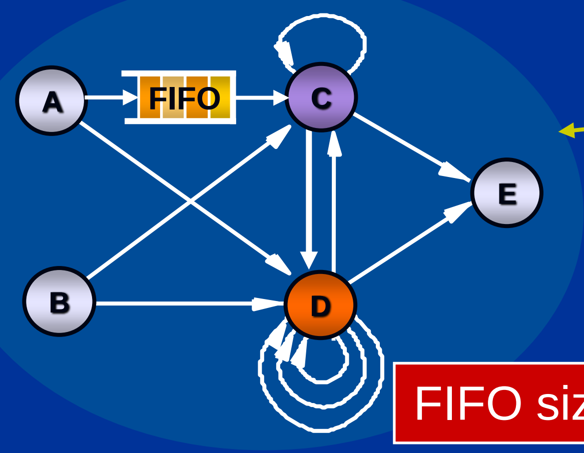
FIFO size calculation

Polyhedral Process Network (optimized)

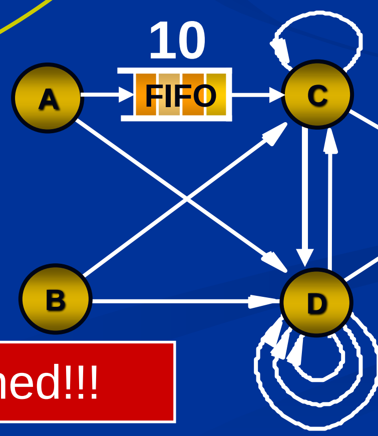
```
int N = 5;
#pragma parameter N 4 16;
Int K = 100;
#pragma parameter K 100 1000;

. . .
for( k=1; k<=K; k++ )
   for ( j=1; j<=N; j++ ) {
      t = C( r[ j ][ j ], x[ k ][ j ], &r[ j ][ j ], &x[ k ][ j ] );
      for( i = j+1; i<=N; i++ ) {
         t = D( t, r[ j ][ i ], x[ k ][ i ], &r[ j ][ i ], &x[ k ][ i ] );
      }
   }
. . .
```



Polyhedral Dependence Graph

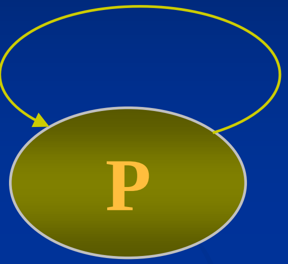


FIFO sizes undefined!!!



FIFO sizes calculation (1)

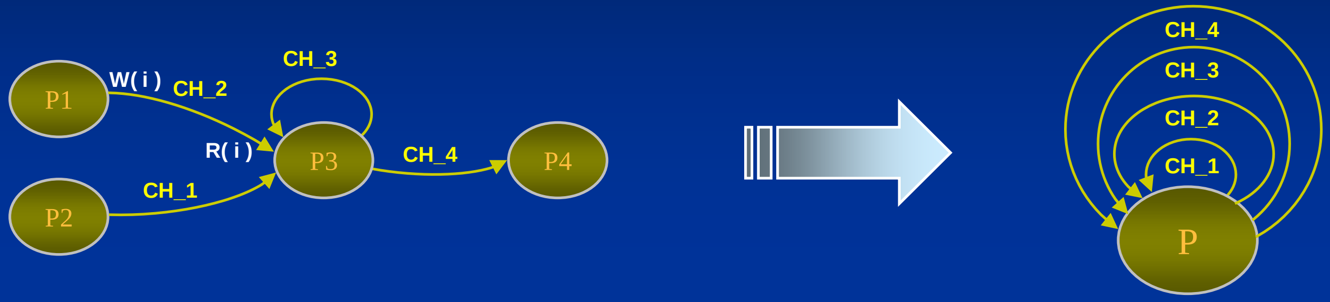
- What is the maximum amount of data in a self-loop FIFO channel?
- For every given process firing i:
  - number of elements written before i : W(i)
  - number of elements read before i : R(i)
  - number of elements in FIFO at i : W(i) - R(i)



- Find max(W(i) - R(i)) over all firings i
- How to compute max(W(i) - R(i))?
  - parametric PPNs: symbolically, using *Barvinok* library
    - W(i) and R(i) are *Ehrhart polynomials*
    - The maximum is found by Bernstein expansions
  - non-parametric PPNs: symbolically or simulate the channels – code generation, using *CLooG*

FIFO sizes calculation (2)

- What is the maximum amount of data in a FIFO channel?



Relative execution order of different processes not known a priori

- Schedule is needed:
  - Put all processes in a common iteration space **i**
    - Offset the processes such that the dependences are respected
    - Try to find minimum offsets → helps to minimize the FIFO sizes!
- All channels are self-loop FIFO channels
- Use the techniques for the self-loop FIFOs

DAEDALUS

Making system-level design take off

http://daedalus.liacs.nl