

# 多面体モデルに依るループ変換

Sven Verdoolaege

K.U.Leuven – Dept. of Computer Science

# コンピュータープログラムの命令

$A :$	0	2	4	6	8			
$B :$								



$$B[1] = A[0] + 0$$

$$B[2] = A[1] + 1$$


$$B[3] = A[2] + 2$$

$$B[4] = A[3] + 3$$

$$B[5] = A[4] + 4$$

# コンピュータープログラムの命令

$A :$	0	2	4	6	8			
$B :$		0						


$$\begin{aligned} B[1] &= A[0] + 0 \\ B[2] &= A[1] + 1 \\ B[3] &= A[2] + 2 \\ B[4] &= A[3] + 3 \\ B[5] &= A[4] + 4 \end{aligned}$$

# コンピュータープログラムの命令

$A :$	0	2	4	6	8			
$B :$		0	3					

$$\begin{aligned} & B[1] = A[0] + 0 \\ \rightarrow & B[2] = A[1] + 1 \\ & B[3] = A[2] + 2 \\ & B[4] = A[3] + 3 \\ & B[5] = A[4] + 4 \end{aligned}$$

# コンピュータープログラムの命令

$A :$	0	2	4	6	8			
$B :$		0	3	6				

$$\begin{aligned} B[1] &= A[0] + 0 \\ B[2] &= A[1] + 1 \\ \rightarrow B[3] &= A[2] + 2 \\ B[4] &= A[3] + 3 \\ B[5] &= A[4] + 4 \end{aligned}$$

# コンピュータープログラムの命令


$A :$	0	2	4	6	8			
$B :$		0	3	6	9			

$$B[1] = A[0] + 0$$

$$B[2] = A[1] + 1$$

$$B[3] = A[2] + 2$$

$$B[4] = A[3] + 3$$


$$B[5] = A[4] + 4$$

# コンピュータープログラムの命令


$A :$	0	2	4	6	8			
$B :$		0	3	6	9	12		

$$B[1] = A[0] + 0$$

$$B[2] = A[1] + 1$$

$$B[3] = A[2] + 2$$

$$B[4] = A[3] + 3$$


$$B[5] = A[4] + 4$$

# コンピュータープログラムのループ

$i$ :	0							
$A$ :	0	2	4	6	8			
$B$ :								

→ `for(i = 0; i < 5; ++i)`  
`B[i+1] = A[i] + i`

# コンピュータープログラムのループ

$i$ :	0							
$A$ :	0	2	4	6	8			
$B$ :		0						

```
for(i = 0; i < 5; ++i)
```

```
    B[i+1] = A[i] + i
```



# コンピュータープログラムのループ

$i$ :	1							
$A$ :	0	2	4	6	8			
$B$ :		0						

→ `for(i = 0; i < 5; ++i)`  
`B[i+1] = A[i] + i`

# コンピュータープログラムのループ

$i$ :	1							
$A$ :	0	2	4	6	8			
$B$ :		0	3					

```
for(i = 0; i < 5; ++i)
```

```
    B[i+1] = A[i] + i
```



# コンピュータープログラムのループ

$i$ :	2						
$A$ :	0	2	4	6	8		
$B$ :		0	3				

→ `for(i = 0; i < 5; ++i)`  
`B[i+1] = A[i] + i`

# コンピュータープログラムのループ

$i$ :	2						
$A$ :	0	2	4	6	8		
$B$ :		0	3	6			

```
for(i = 0; i < 5; ++i)
```

```
    B[i+1] = A[i] + i
```



# コンピュータープログラムのループ

$i$ :	3						
$A$ :	0	2	4	6	8		
$B$ :		0	3	6			

→ `for(i = 0; i < 5; ++i)`  
`B[i+1] = A[i] + i`

# コンピュータープログラムのループ

$i$ :	3						
$A$ :	0	2	4	6	8		
$B$ :		0	3	6	9		

```
for(i = 0; i < 5; ++i)
```

```
    B[i+1] = A[i] + i
```



# コンピュータープログラムのループ

$i$ :	4						
$A$ :	0	2	4	6	8		
$B$ :		0	3	6	9		

→ `for(i = 0; i < 5; ++i)`  
`B[i+1] = A[i] + i`

# コンピュータープログラムのループ

$i$ :	4						
$A$ :	0	2	4	6	8		
$B$ :		0	3	6	9	12	

```
for(i = 0; i < 5; ++i)
```

```
    B[i+1] = A[i] + i
```



# ループ変換

$i$ :	4						
$A$ :	0	2	4	6	8		
$B$ :							


→ `for(i = 4; i >= 0; --i)`  
`B[i+1] = A[i] + i`

# ループ変換

$i$ :	4						
$A$ :	0	2	4	6	8		
$B$ :						12	


```
for(i = 4; i >= 0; --i)
```

```
    B[i+1] = A[i] + i
```




# ループ変換

$i$ :	3						
$A$ :	0	2	4	6	8		
$B$ :						12	

 `for(i = 4; i >= 0; --i)`  
`B[i+1] = A[i] + i`

# ループ変換

$i$ :	3						
$A$ :	0	2	4	6	8		
$B$ :					9	12	

 `for(i = 4; i >= 0; --i)`  
`B[i+1] = A[i] + i`

# ループ変換

$i$ :	2						
$A$ :	0	2	4	6	8		
$B$ :					9	12	


→ `for(i = 4; i >= 0; --i)`  
`B[i+1] = A[i] + i`

# ループ変換

$i$ :	2						
$A$ :	0	2	4	6	8		
$B$ :				6	9	12	


```
for(i = 4; i >= 0; --i)
```

```
    B[i+1] = A[i] + i
```




# ループ変換

$i$ :	1							
$A$ :	0	2	4	6	8			
$B$ :				6	9	12		

 `for(i = 4; i >= 0; --i)`  
`B[i+1] = A[i] + i`


# ループ変換

$i$ :	1							
$A$ :	0	2	4	6	8			
$B$ :			3	6	9	12		

 `for(i = 4; i >= 0; --i)`  
`B[i+1] = A[i] + i`

# ループ変換

$i$ :	0						
$A$ :	0	2	4	6	8		
$B$ :			3	6	9	12	


 `for(i = 4; i >= 0; --i)`  
`B[i+1] = A[i] + i`

# ループ変換

$i$ :	0						
$A$ :	0	2	4	6	8		
$B$ :		0	3	6	9	12	

```
for(i = 4; i >= 0; --i)
```

```
    B[i+1] = A[i] + i
```



# DTSE

## Data Transfer and Storage Exploration

→ Optimize memory usage to reduce power consumption

- Remove unnecessary buffers
- Data reuse

Domain: embedded systems, ...

# Basic Idea

*Global* loop transformations

Loop transformation = reordering of loop iterations

Aim:

- reduce life-times of array elements
- increase locality and regularity of accesses

How:

Transformation of polytopes in the polyhedral model

# Example

```
for (j=0; j<=10; ++j)
    A[3][j] = ...
for (i=0; i<=5; ++i)
    for (j=0; j<=5; ++j)
        ... = A[3][i+2*j+3];
```

# Example

```
for (j=0; j<=10; ++j)
  A[3][j] = ...
for (i=0; i<=5; ++i)
  for (j=0; j<=5; ++j)
    ... = A[3][i+2*j+3];
```

Buffer size: 8

# Example

```
for (j=0; j<=10; ++j)
  A[3][j] = ...
for (i=0; i<=5; ++i)
  for (j=0; j<=5; ++j)
    ... = A[3][i+2*j+3];
```

Buffer size: 8

```
...
for (i = 3; i <= 10; i++) {
  A[3][i] = ...
  for(j=max(ceil(i-2, 2), 3); j<=floor(i+3, 2); j++)
    ... = A[3][i]; }
...
```

# Example

```
for (j=0; j<=10; ++j)
  A[3][j] = ...
for (i=0; i<=5; ++i)
  for (j=0; j<=5; ++j)
    ... = A[3][i+2*j+3];
```

Buffer size: 8

```
...
for (i = 3; i <= 10; i++) {
  A[3][i] = ...
  for(j=max(ceil(i-2, 2), 3); j<=floor(i+3, 2); j++)
    ... = A[3][i]; }
...
```

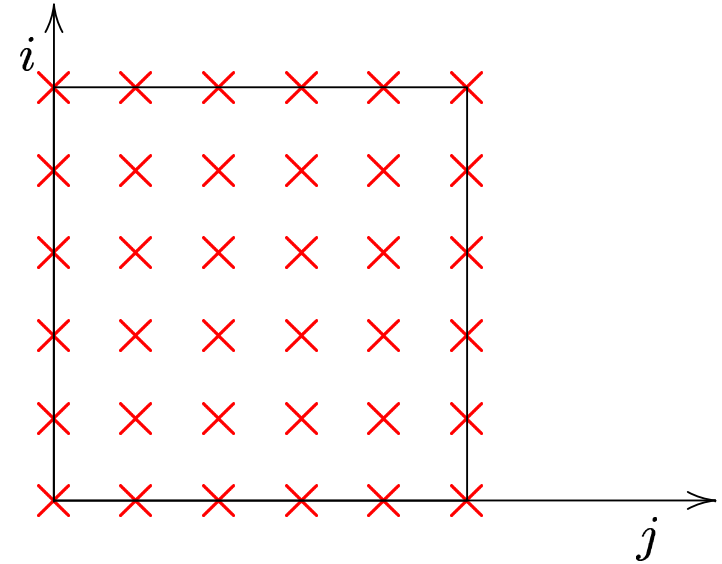
Buffer size: 1

# Example

```
for (i=0;i<=0;++i)
  for (j=0;j<=10;++j)
    A[3][j] = ...
for (i=0;i<=5;++i)
  for (j=0;j<=5;++j)
    ... = A[3][i+2*j+3];
```

# Example

```
for (i=0; i<=0; ++i)
  for (j=0; j<=10; ++j)
    A[3][j] = ...
for (i=0; i<=5; ++i)
  for (j=0; j<=5; ++j)
    ... = A[3][i+2*j+3];
```



# Steps

- Model extraction  
N level loop nest  $\rightarrow$  N-dimensional polytope
- Placement  
 $\Rightarrow$  Polytopes in common iteration space
- Ordering  
 $\Rightarrow$  Schedule
- Code generation

# Illustrative example

Initial specification:

```
(i: 1..N)::
```

```
  (j: 1..N-i+1)::
```

```
    a[i][j] = in[i][j] + a[i-1][j];
```

```
(p: 1..N)::
```

```
  b[p][1] = f( a[N-p+1][p], a[N-p][p] );
```

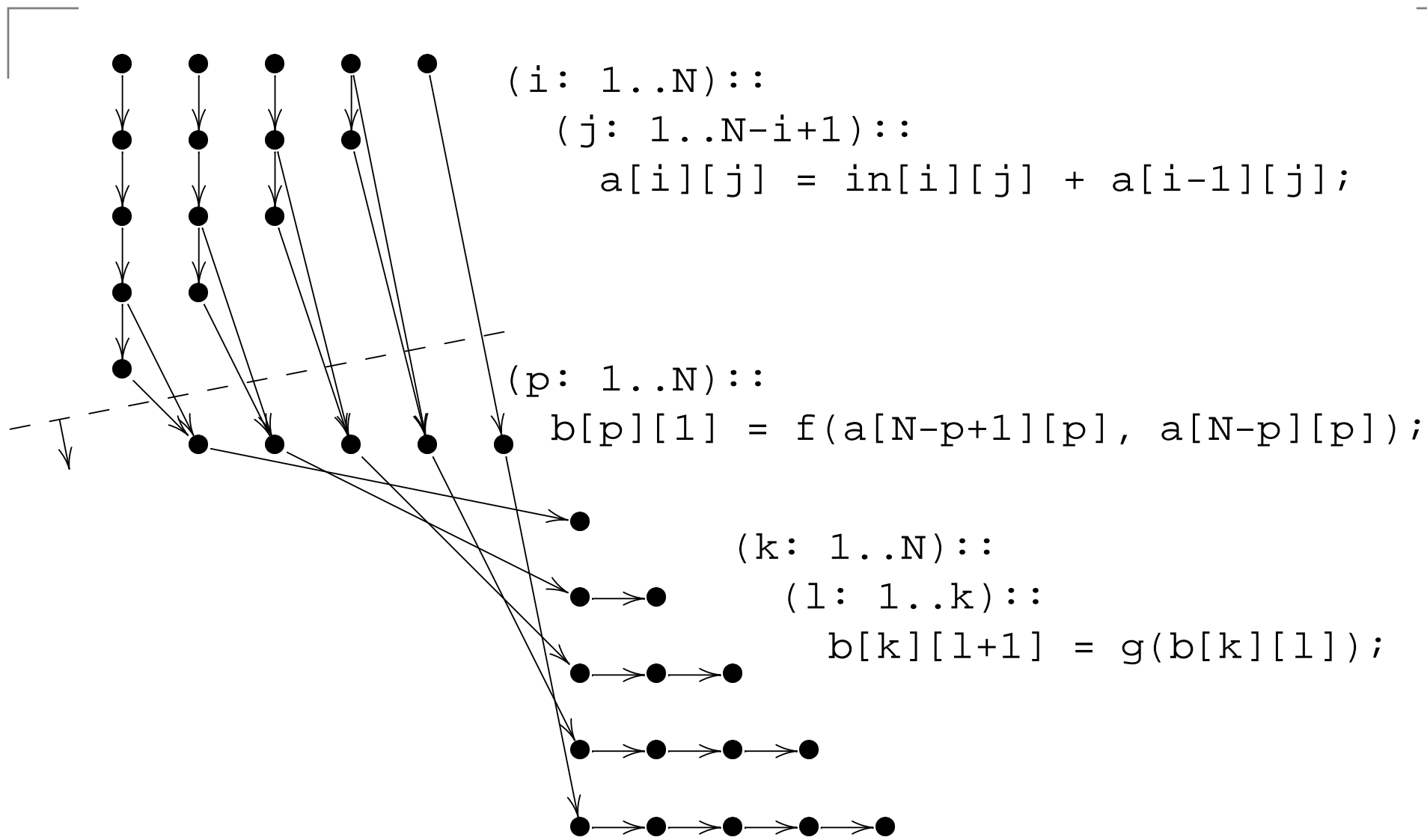
```
(k: 1..N)::
```

```
  (l: 1..k)::
```

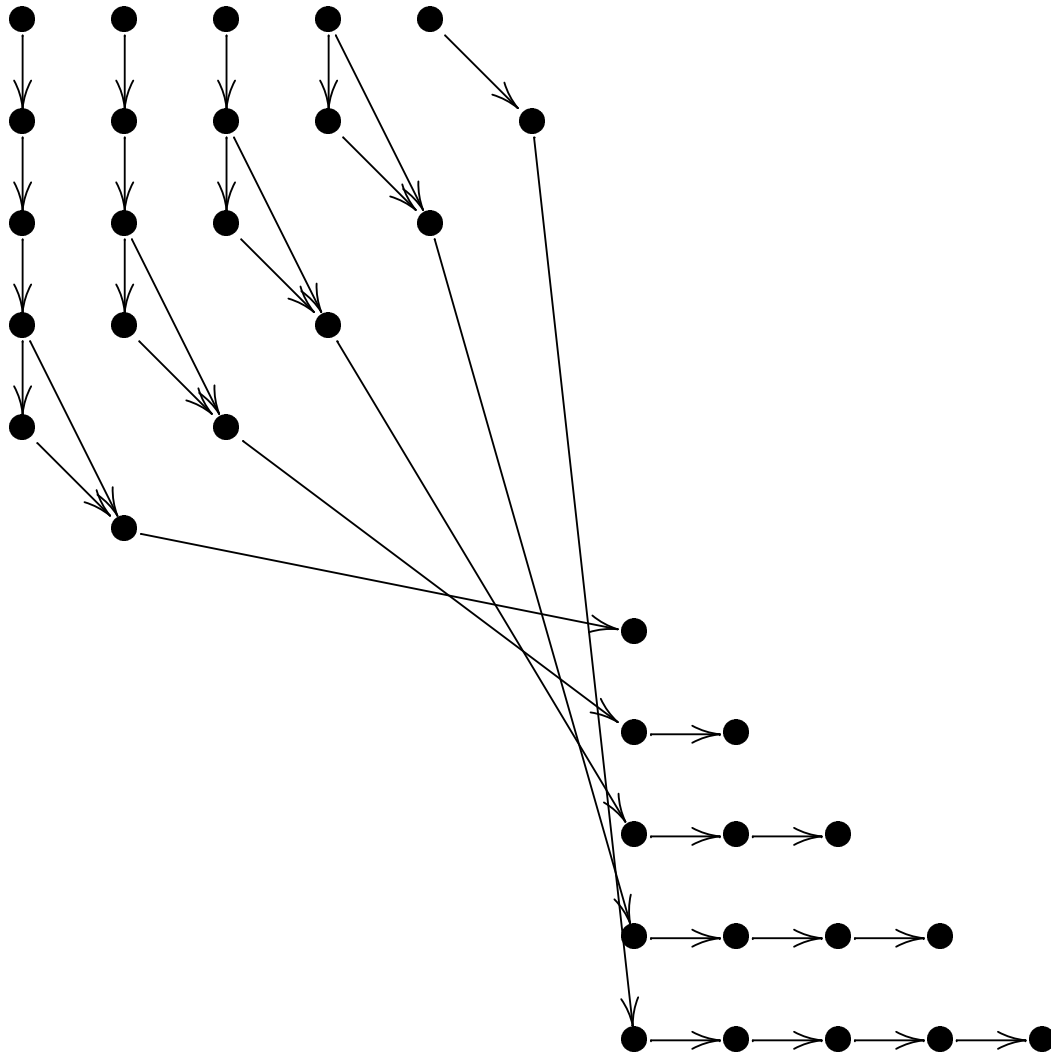
```
    b[k][l+1] = g( b[k][l] );
```

Buffer space required: 2N

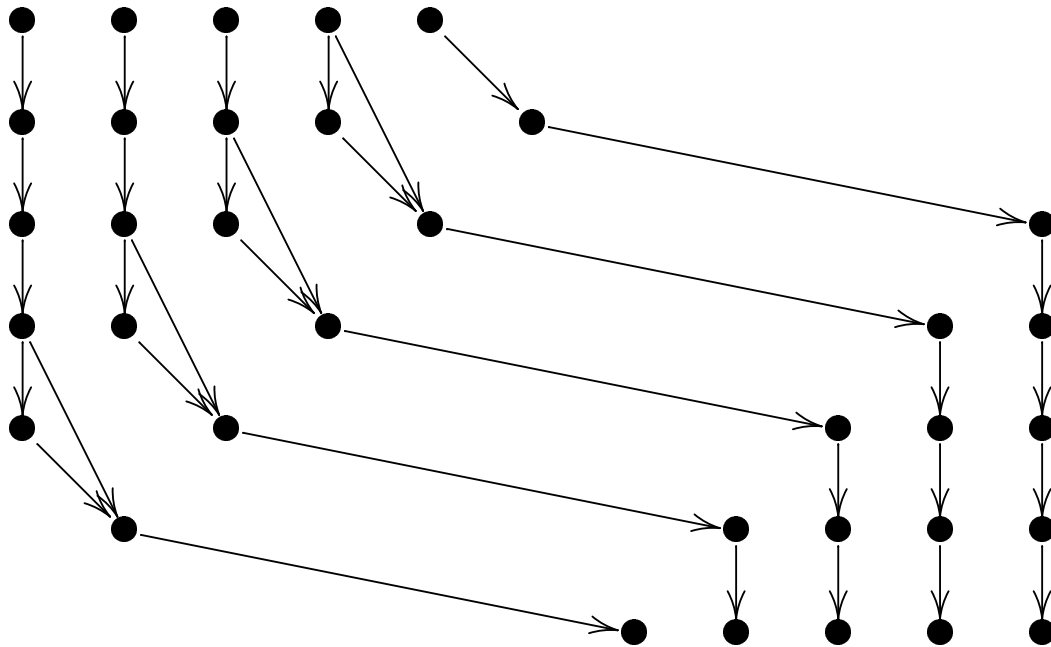
# Model Extraction



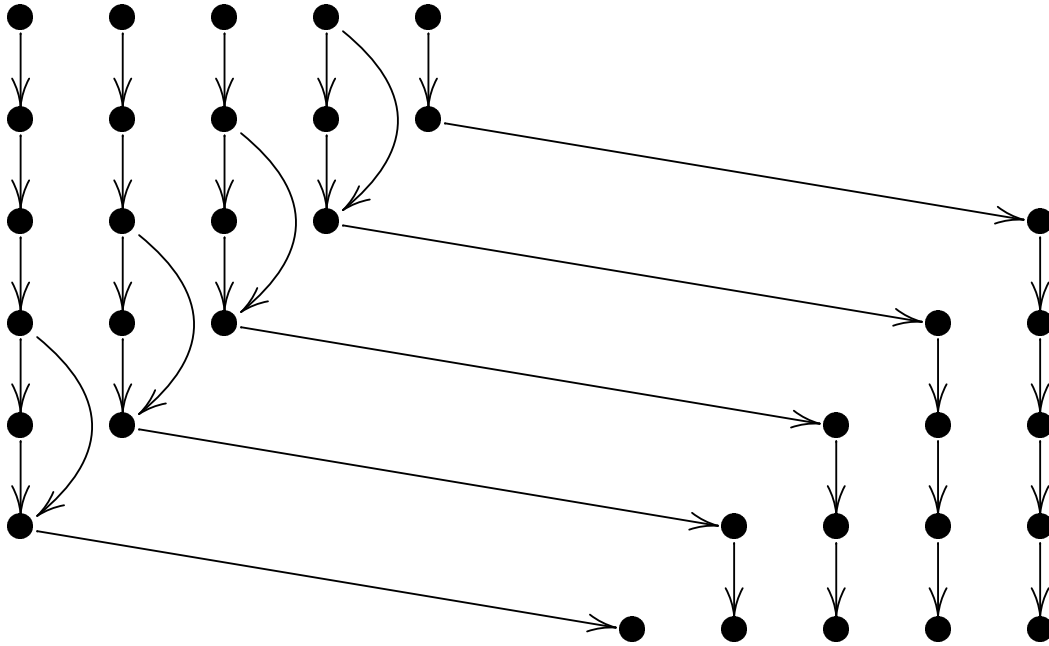
# Linear Placement



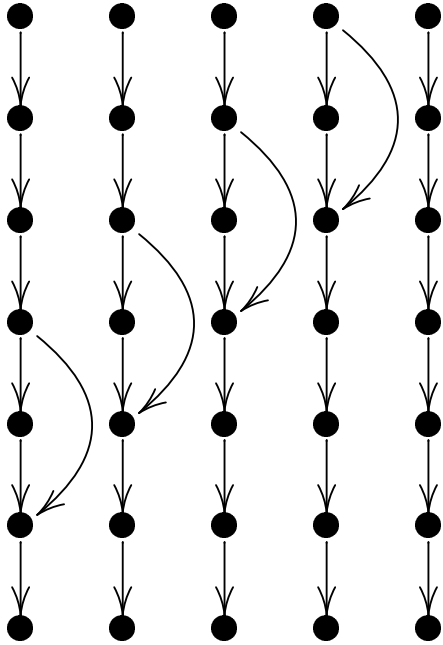
# Linear Placement



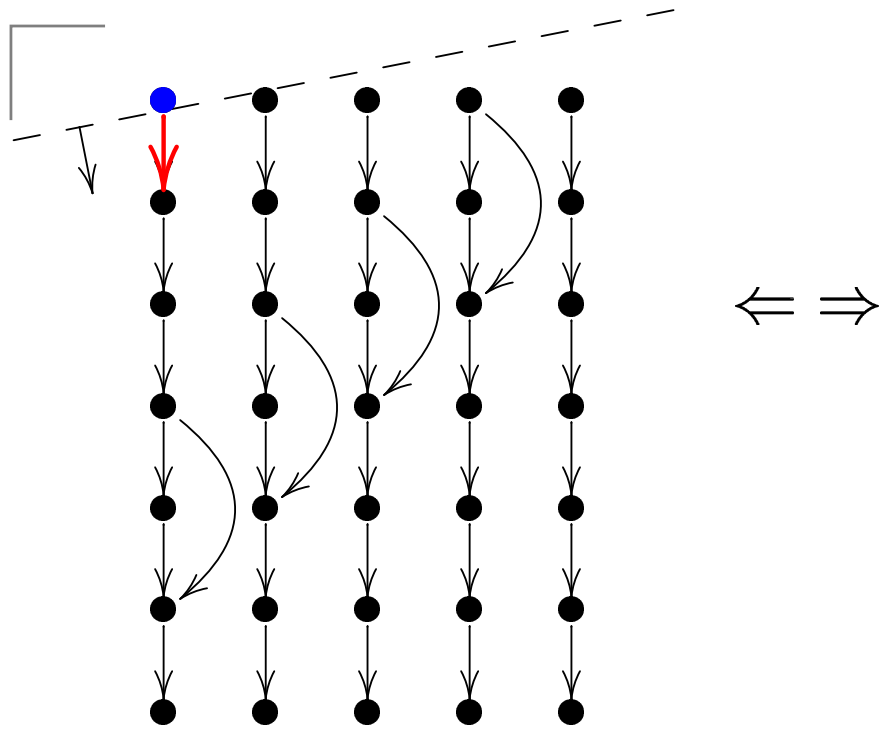
# Translation



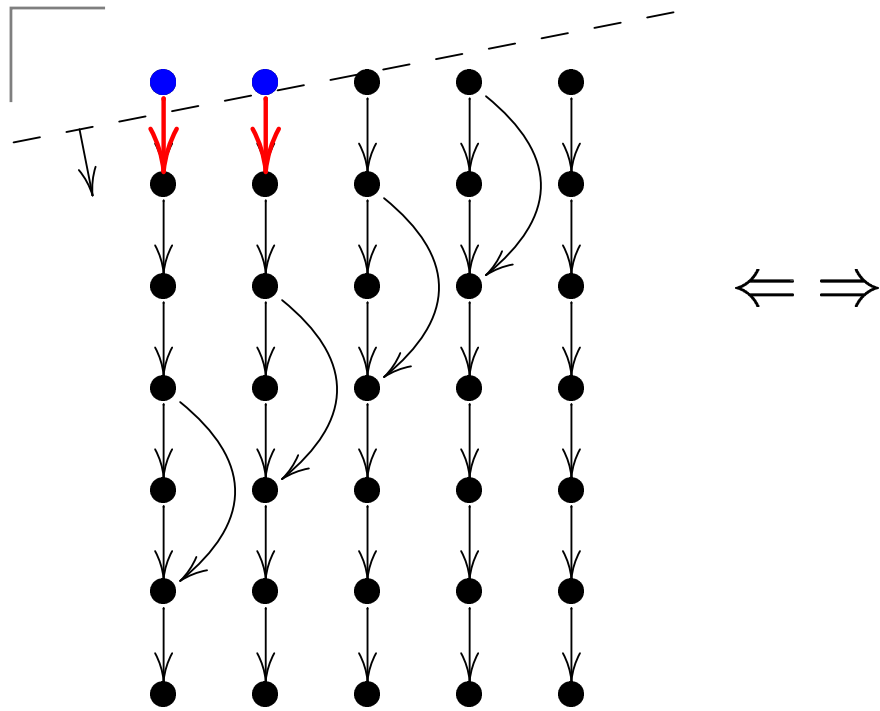
# Translation



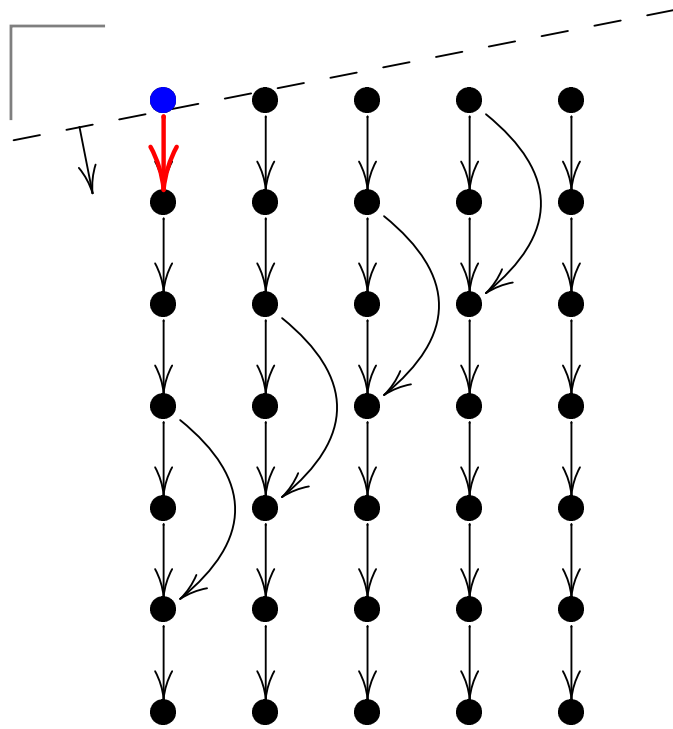
# Ordering-I



# Ordering-I

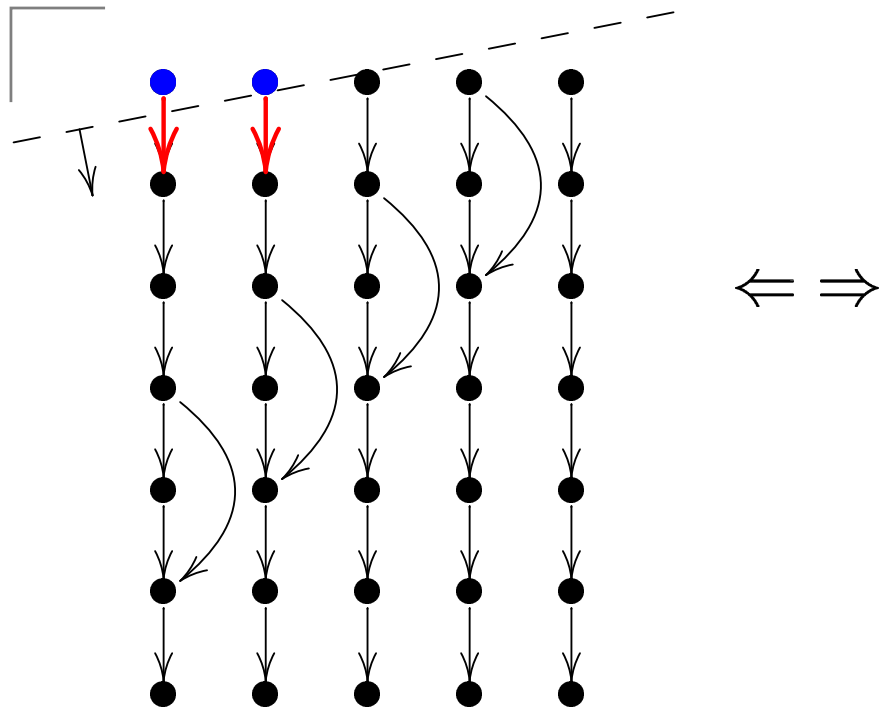


# Ordering-I

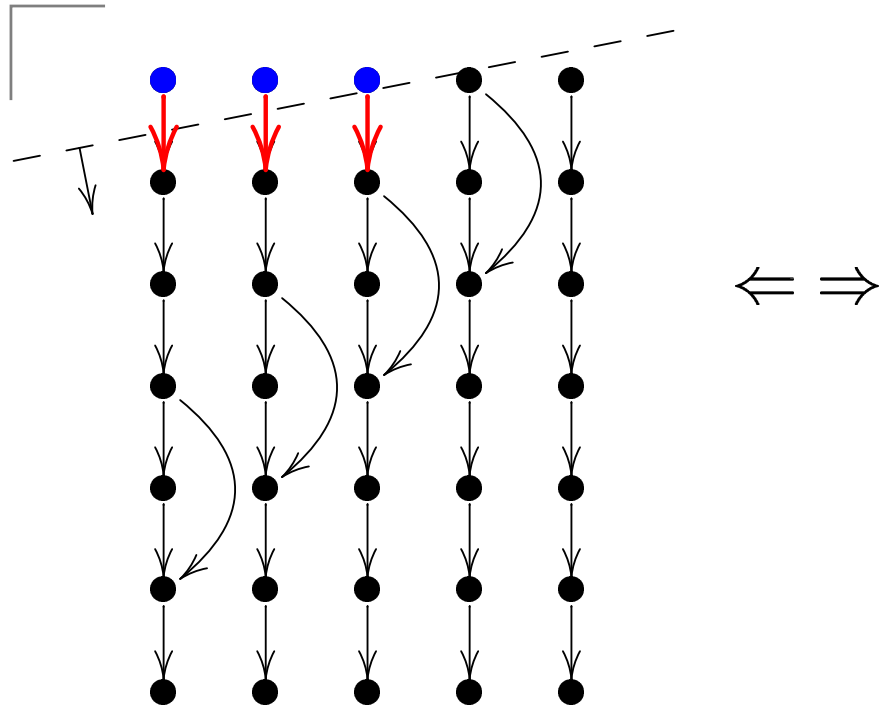


$\Leftrightarrow \Rightarrow$

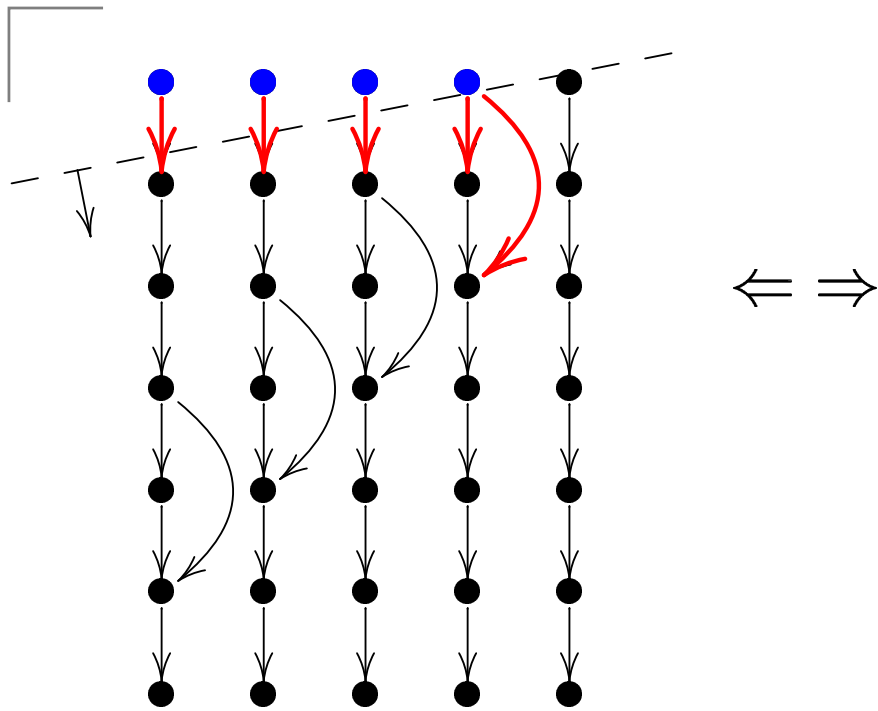
# Ordering-I



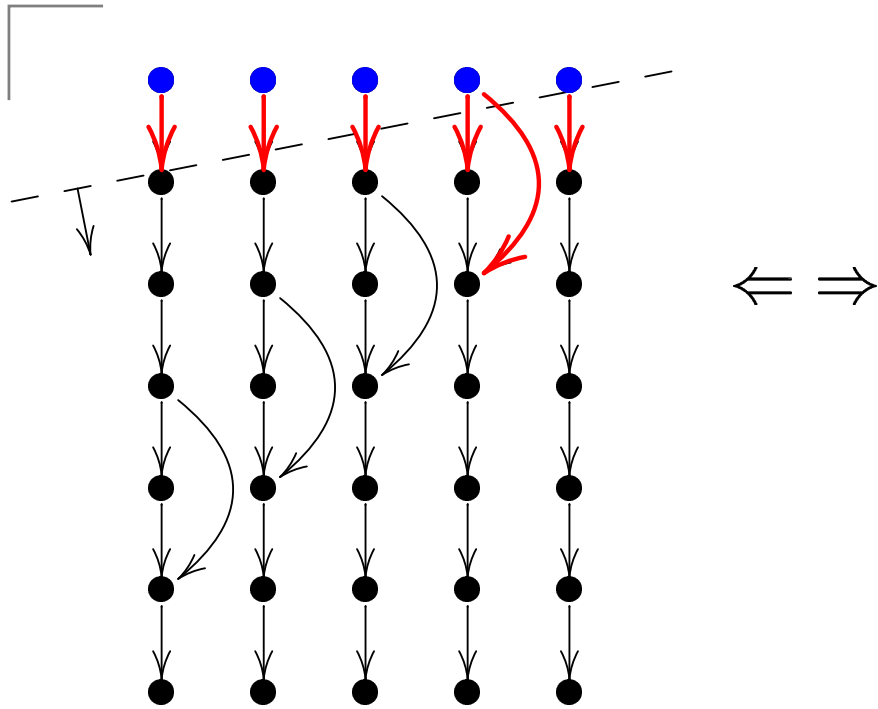
# Ordering-I



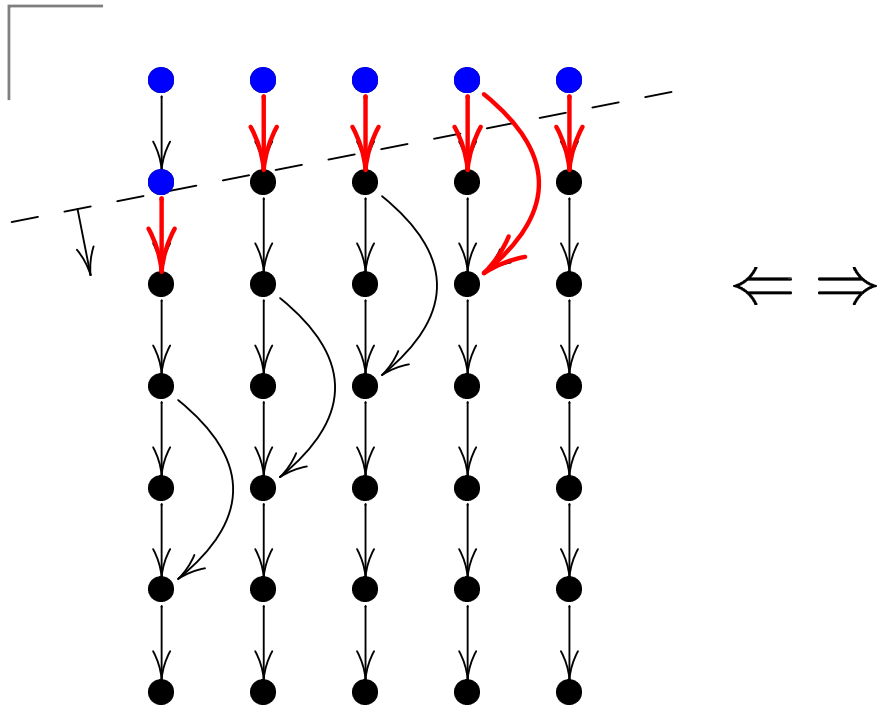
# Ordering-I



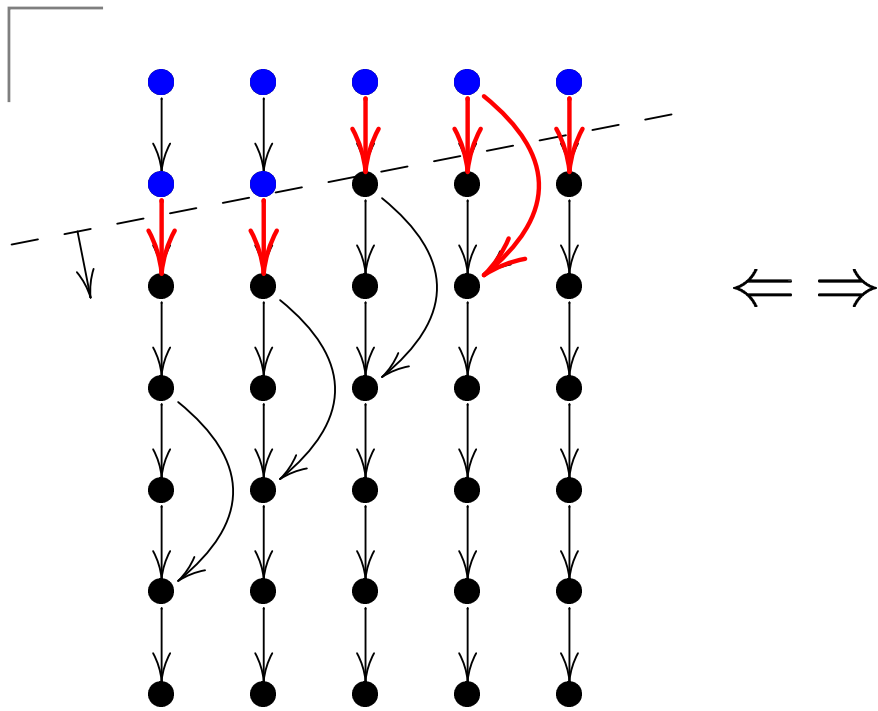
# Ordering-I



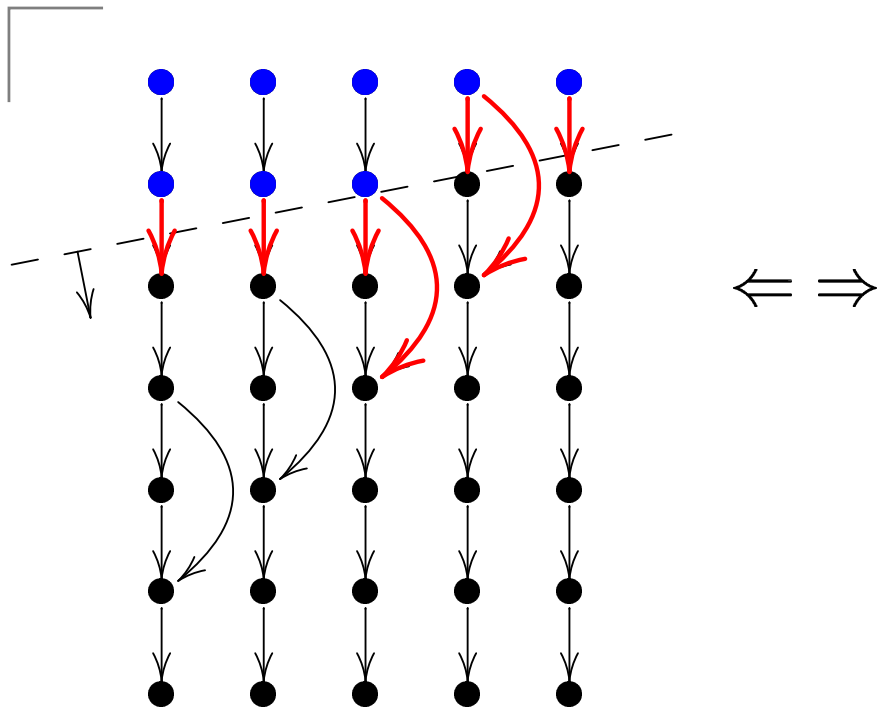
# Ordering-I



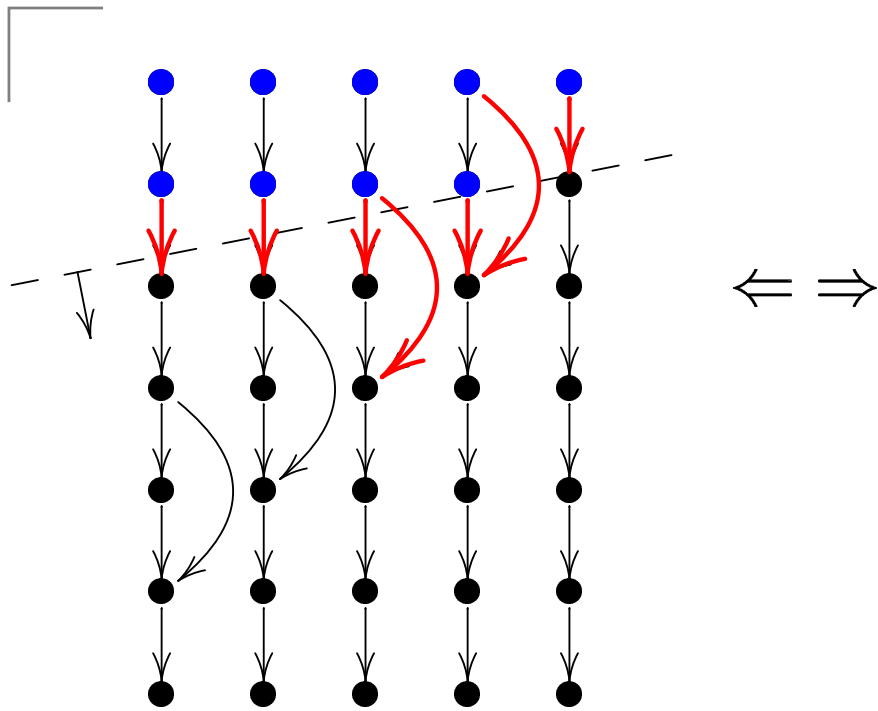
# Ordering-I



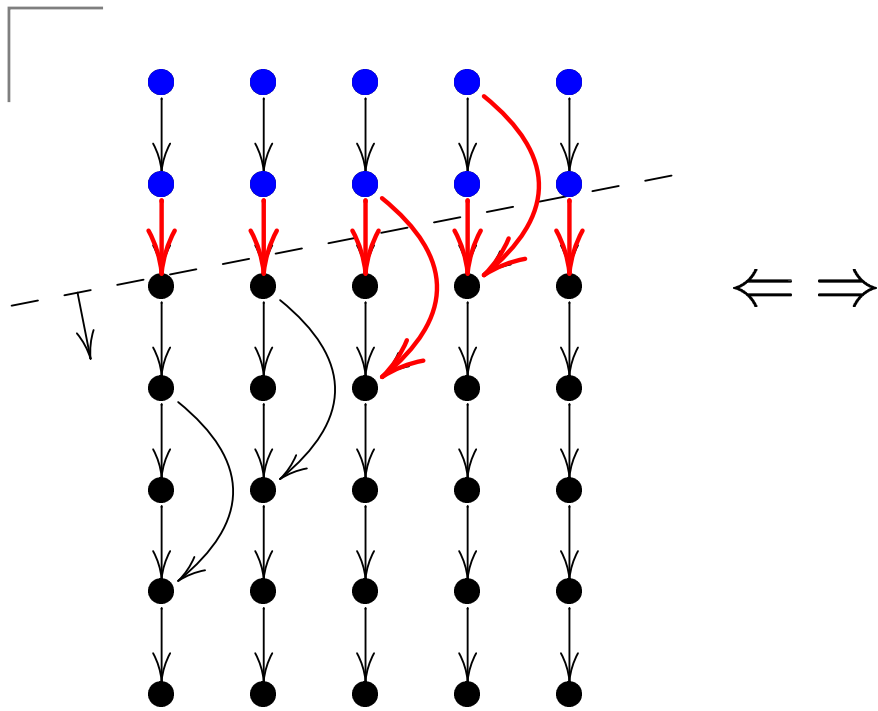
# Ordering-I



# Ordering-I



# Ordering-I



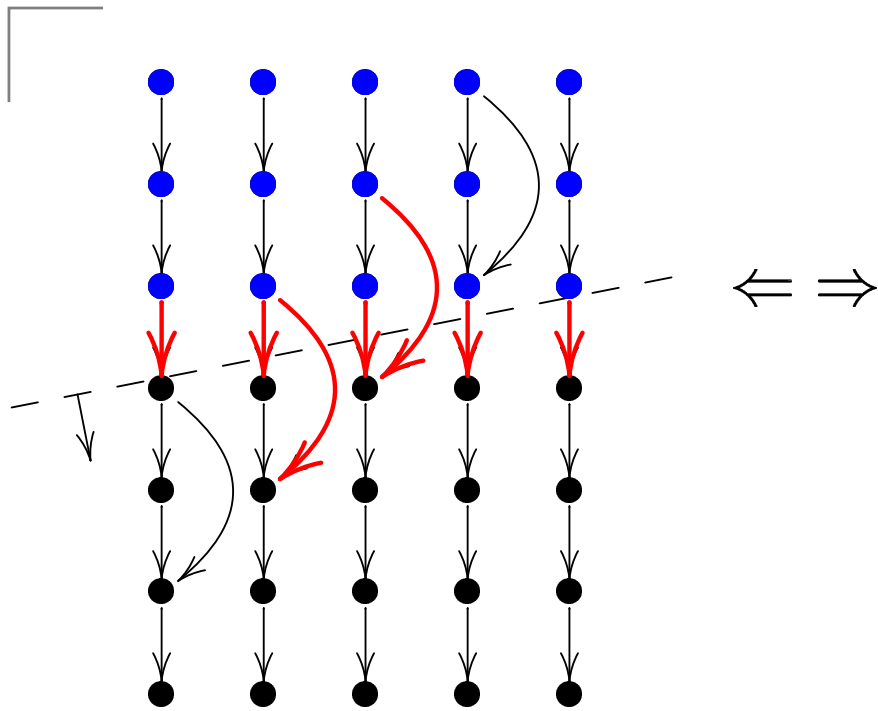




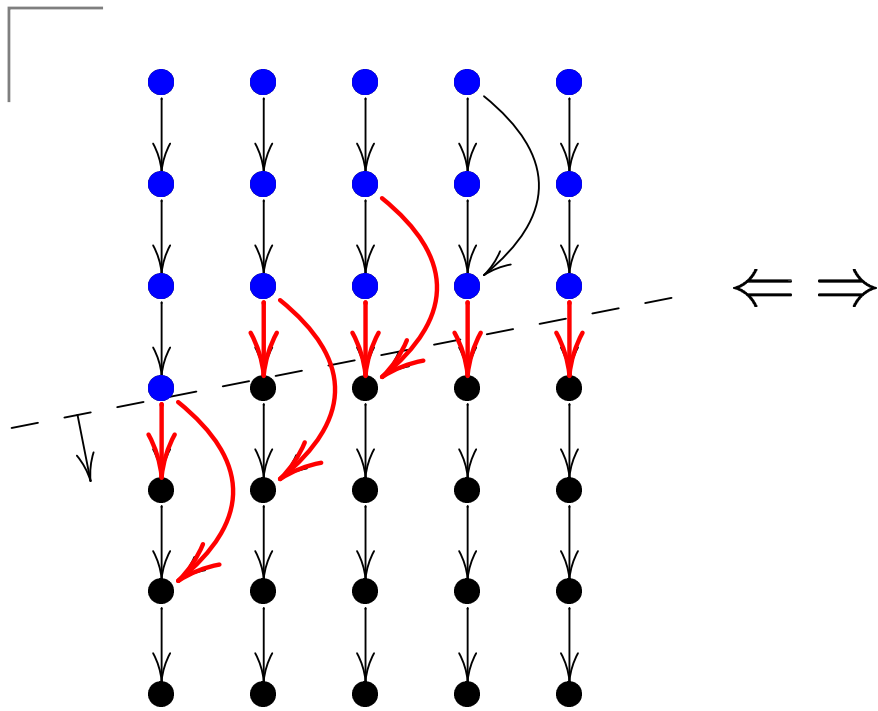




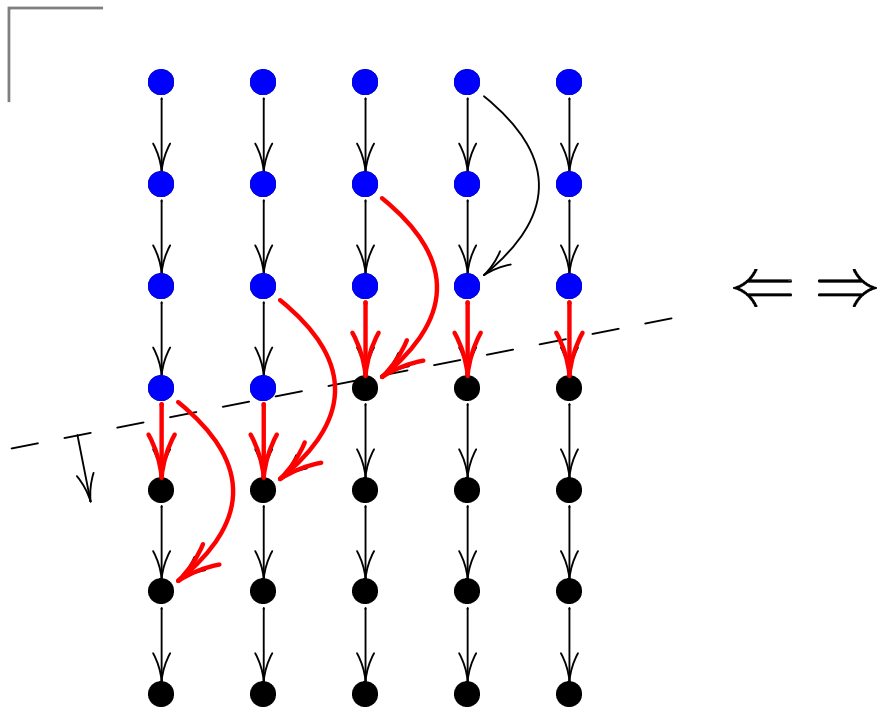
# Ordering-I



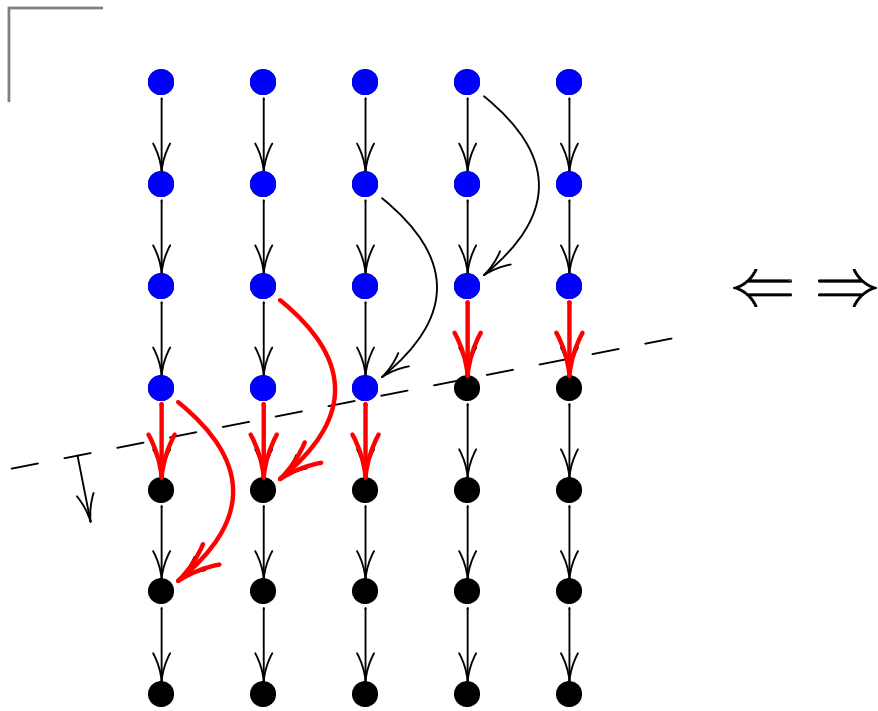
# Ordering-I



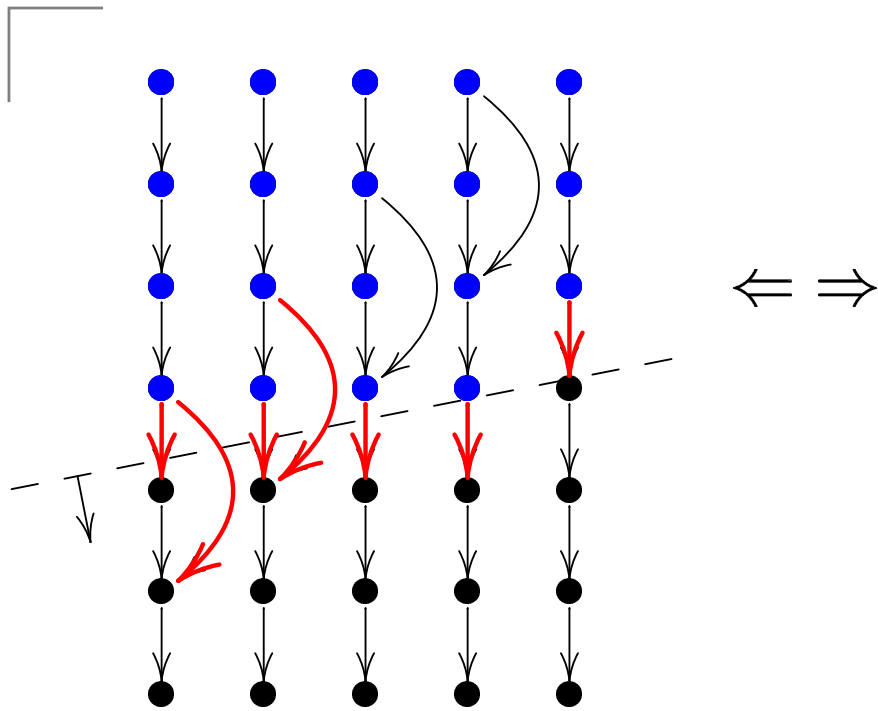
# Ordering-I



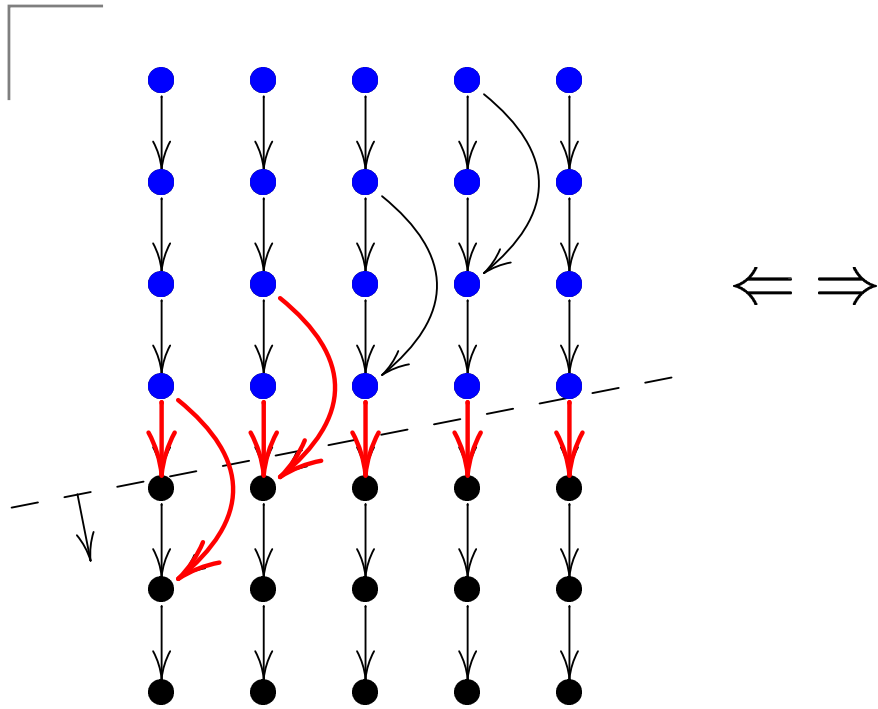
# Ordering-I



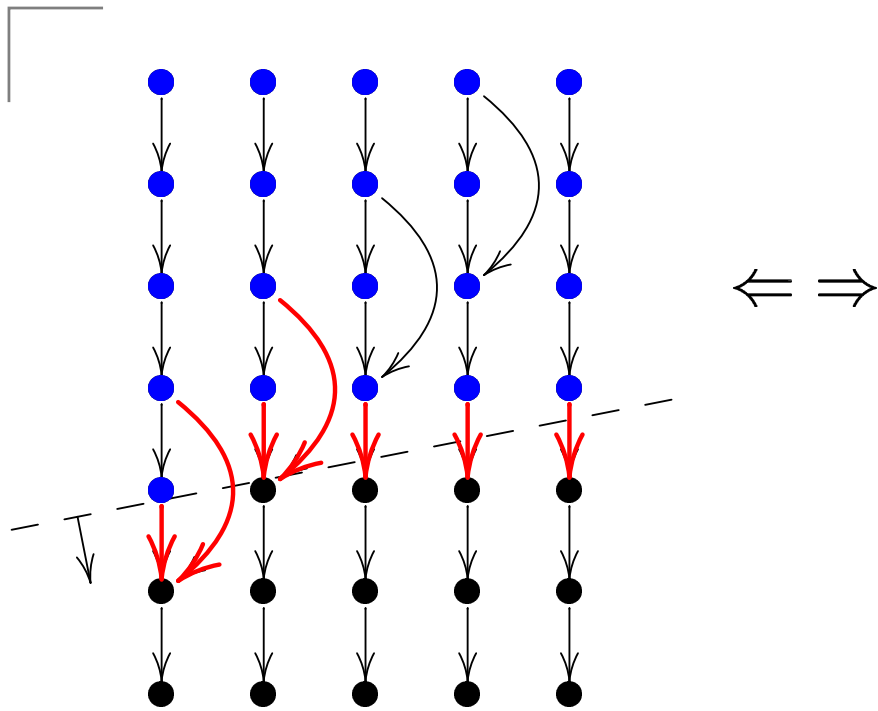
# Ordering-I



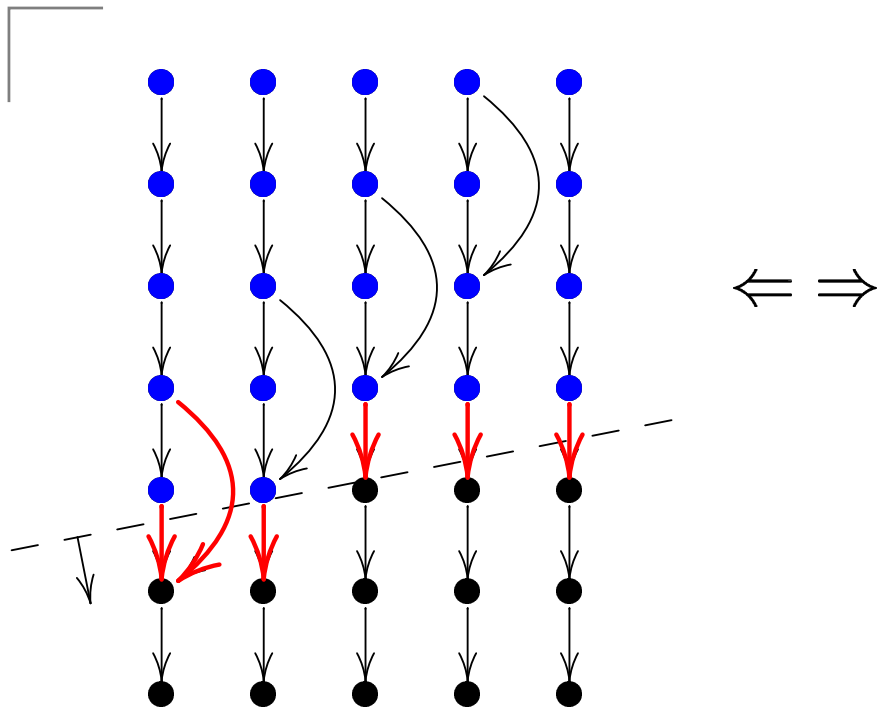
# Ordering-I



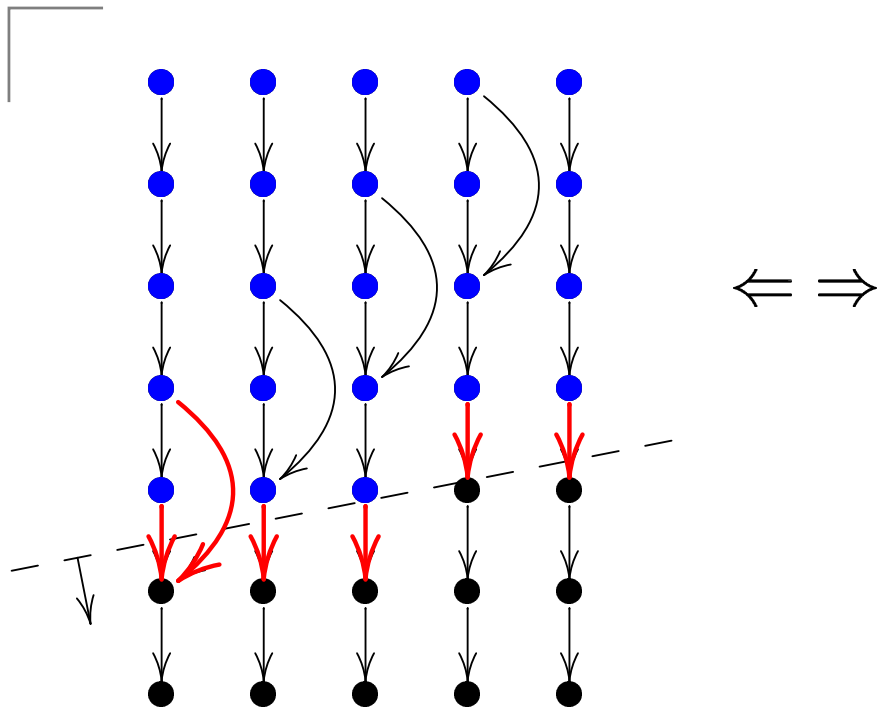
# Ordering-I



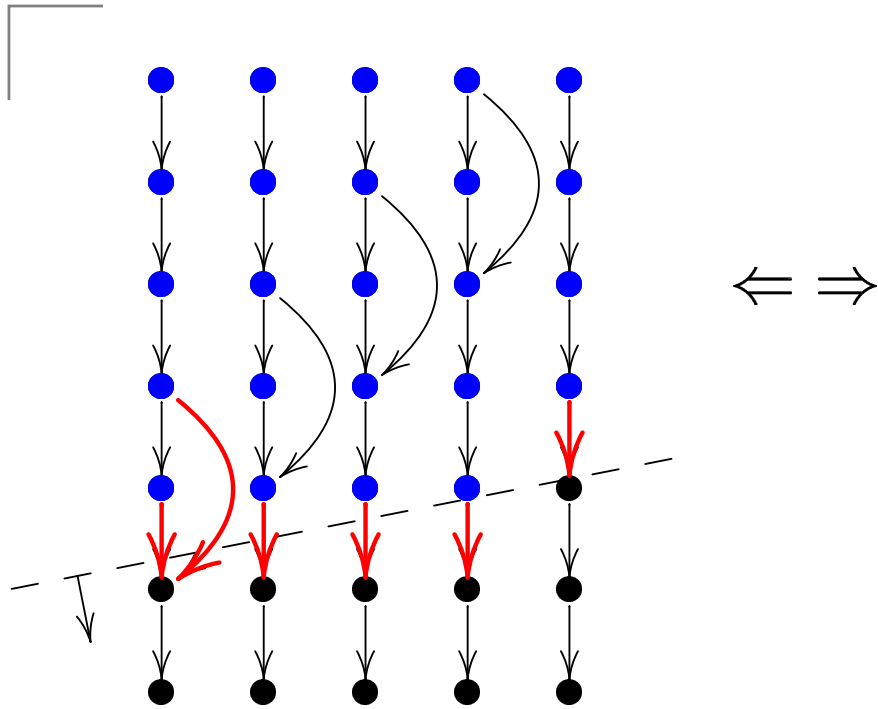
# Ordering-I



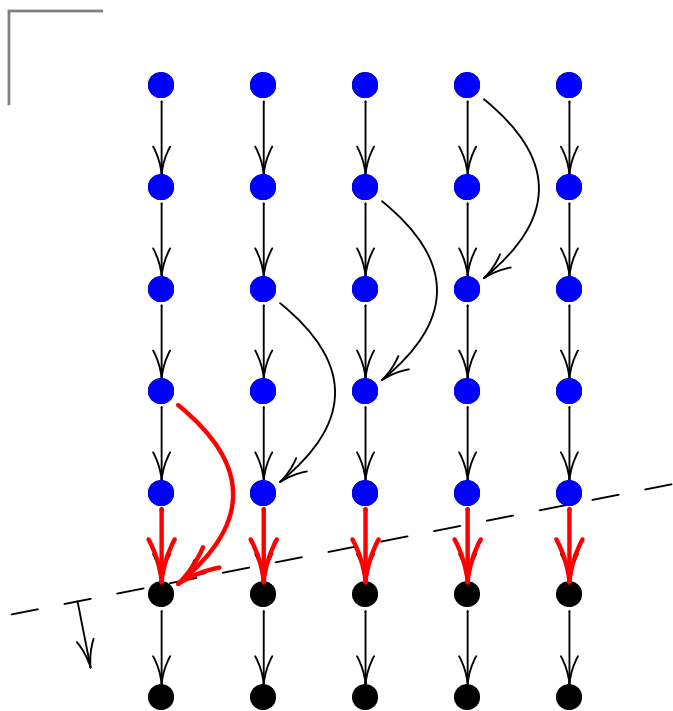
# Ordering-I



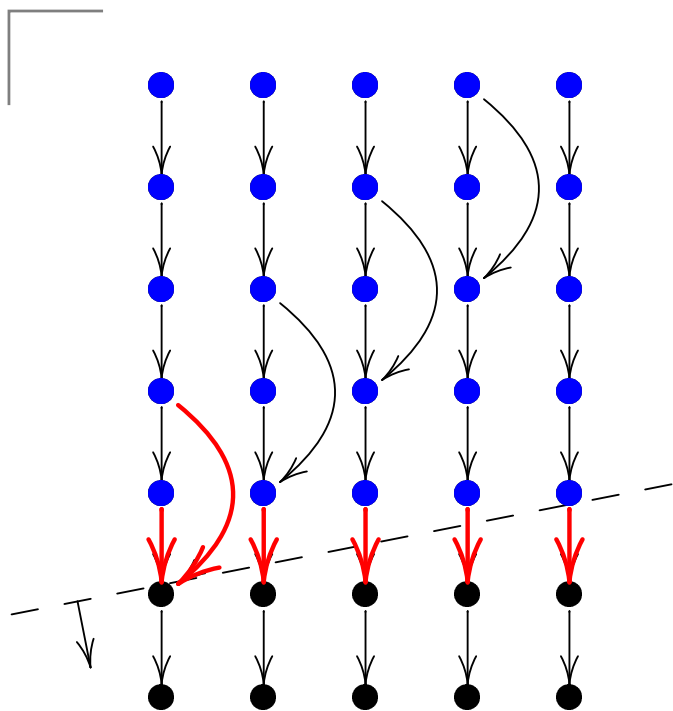
# Ordering-I



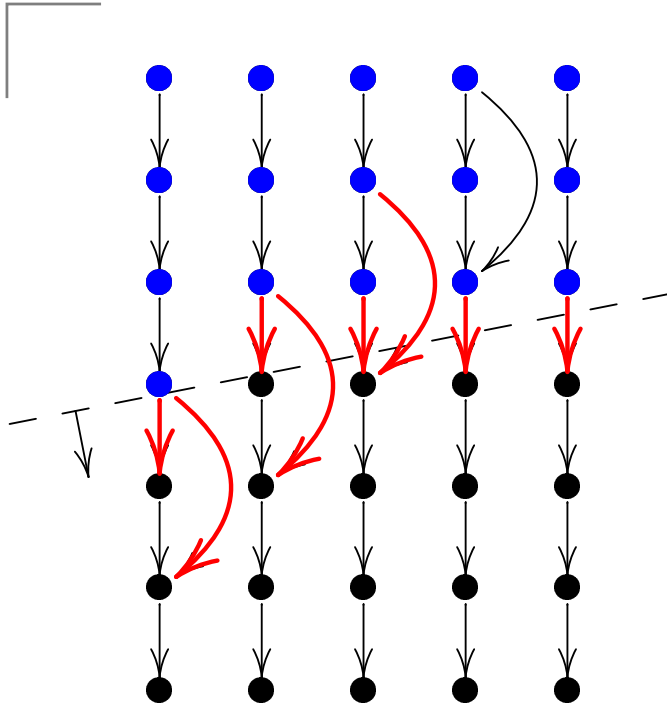
# Ordering-I



# Ordering-I



# Ordering-I



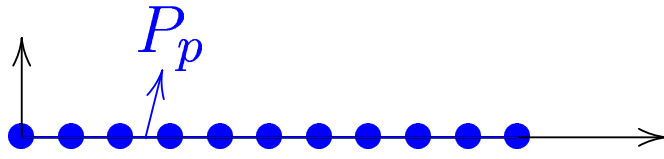
```
for (i=1; i<=N+2; ++i) {  
  for (j=1; j<=N-i+1; ++j)  
    a[i][j] = in[i][j] + a[i-1][j];  
  if (...) b[N-i+2][1] = f(a[i-1][N-i+2],  
                          a[i-2][N-i+2]);  
  for (k=N-i+3; k<=N; ++k)  
    b[k][i+k-N-1] = g(b[k][i+k-N-2]);  
}
```

Choosing an ordering vector: 1st possibility (buffer space required:  $N+3$ )

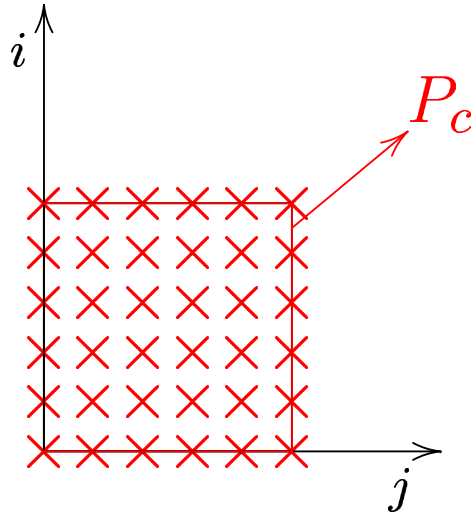
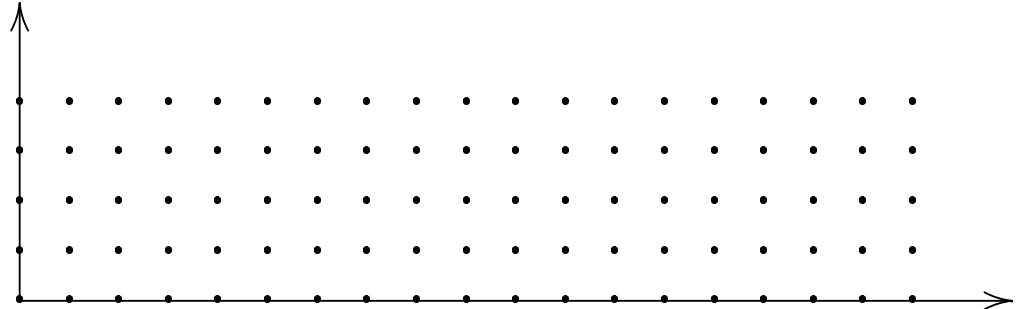


# Dependencies in PDG

production space



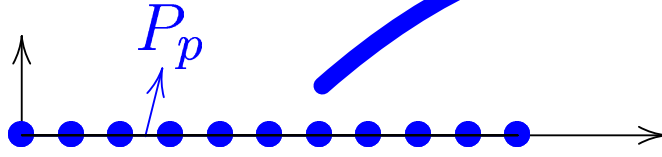
array space



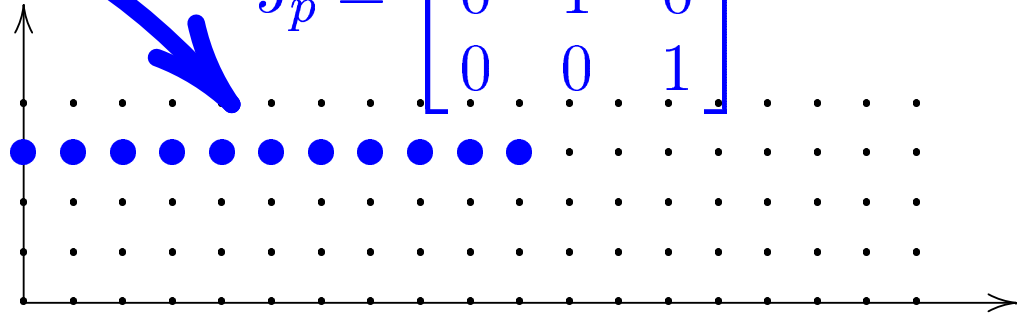
consumption space

# Dependencies in PDG

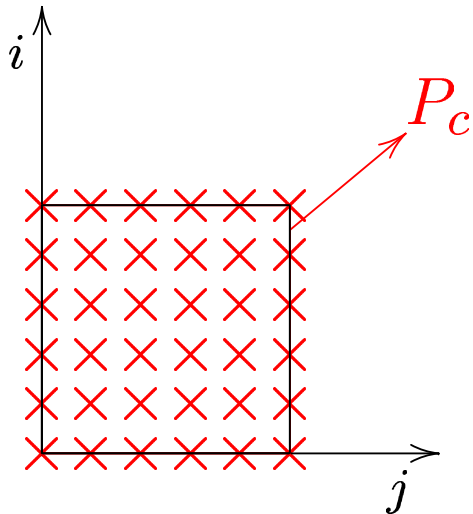
production space



$$\tilde{J}_p = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



array space

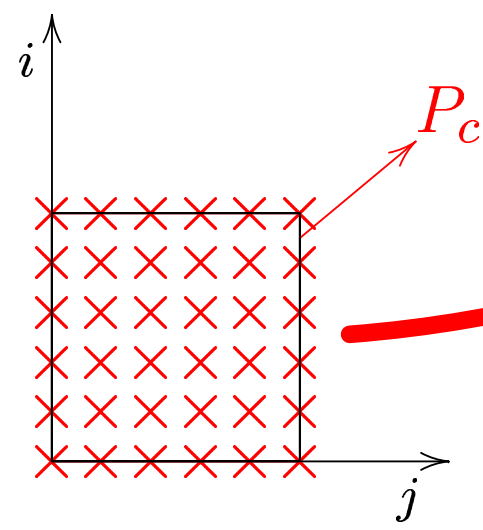
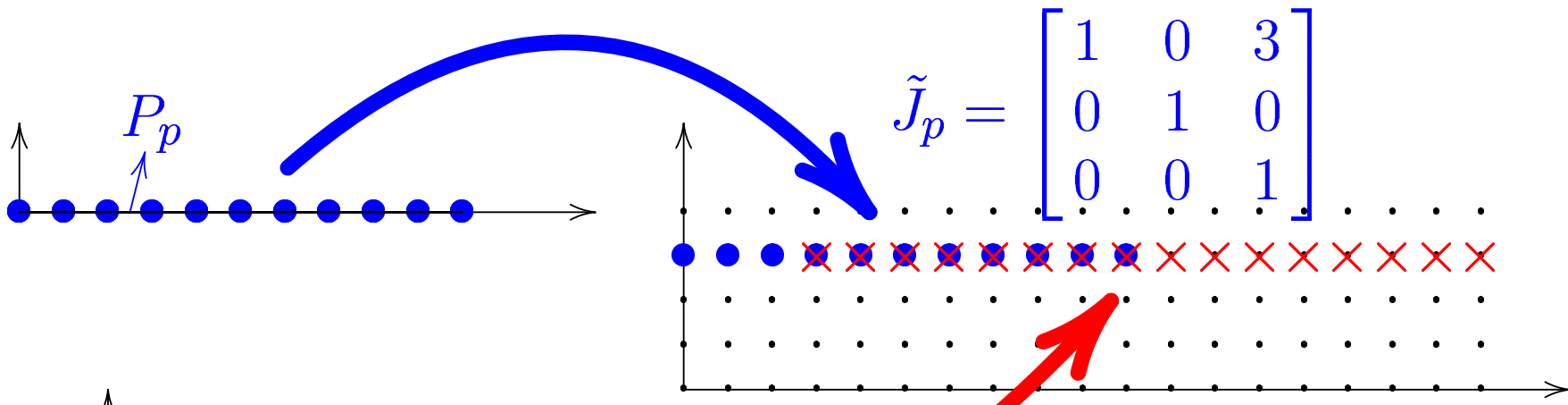


consumption space

# Dependencies in PDG

production space

array space

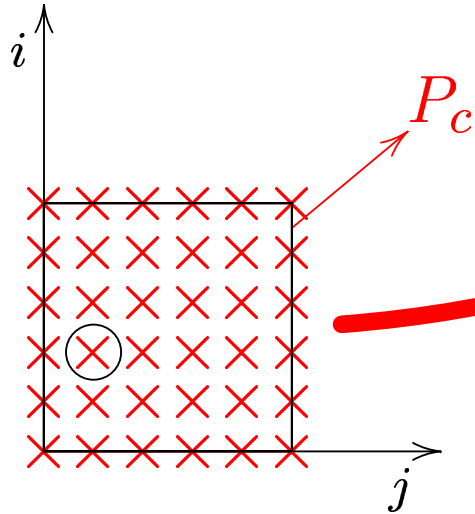
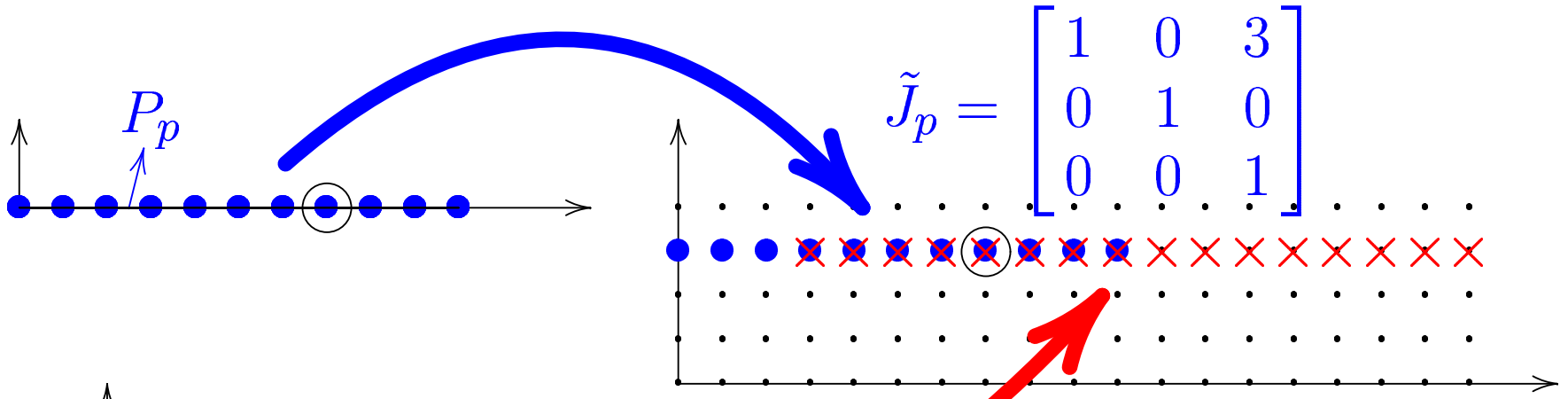


$$\tilde{J}_c = \begin{bmatrix} 0 & 0 & 3 \\ 1 & 2 & 3 \\ 0 & 0 & 1 \end{bmatrix}$$

consumption space

# Dependencies in PDG

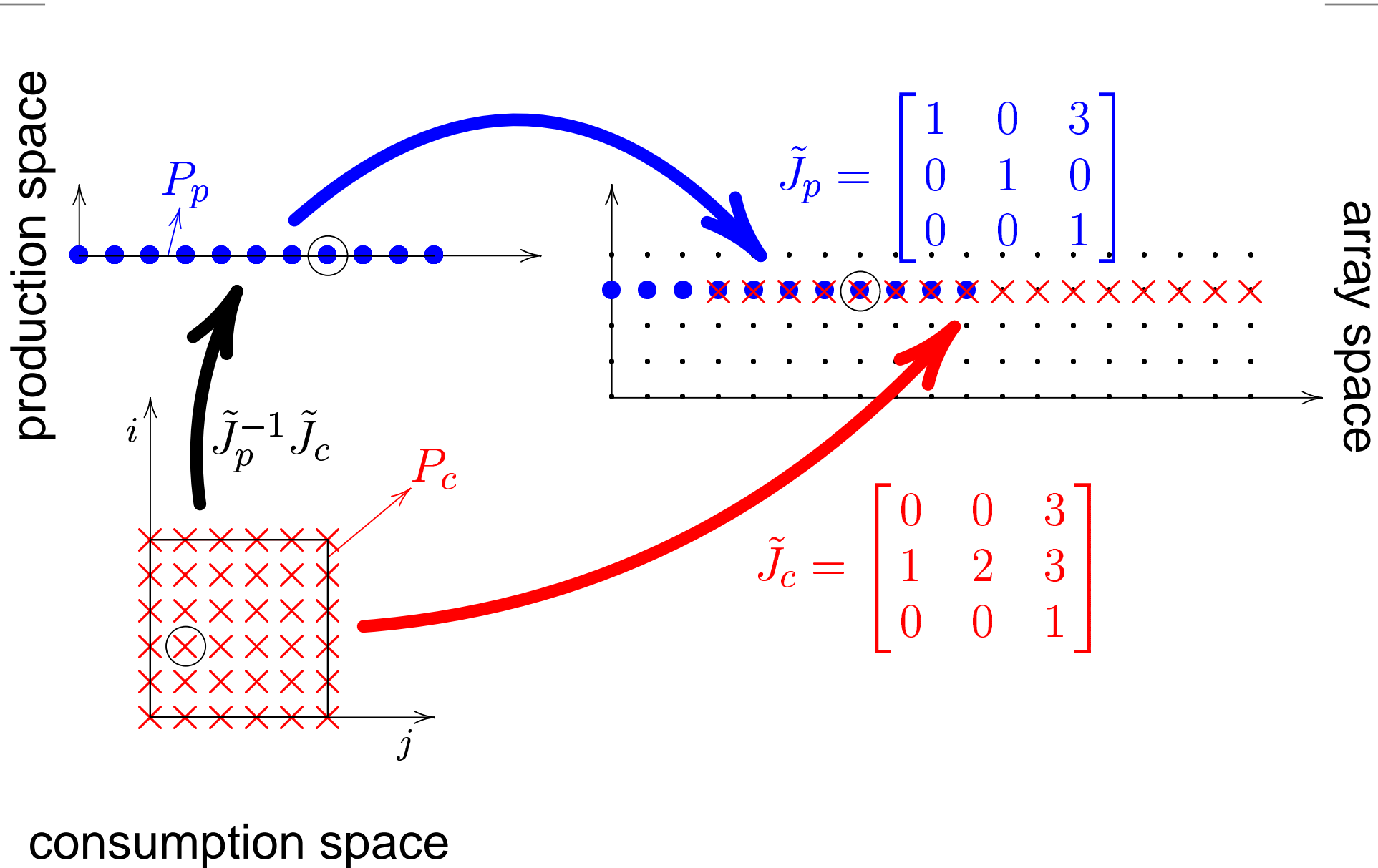
production space



$$\tilde{J}_c = \begin{bmatrix} 0 & 0 & 3 \\ 1 & 2 & 3 \\ 0 & 0 & 1 \end{bmatrix}$$

consumption space

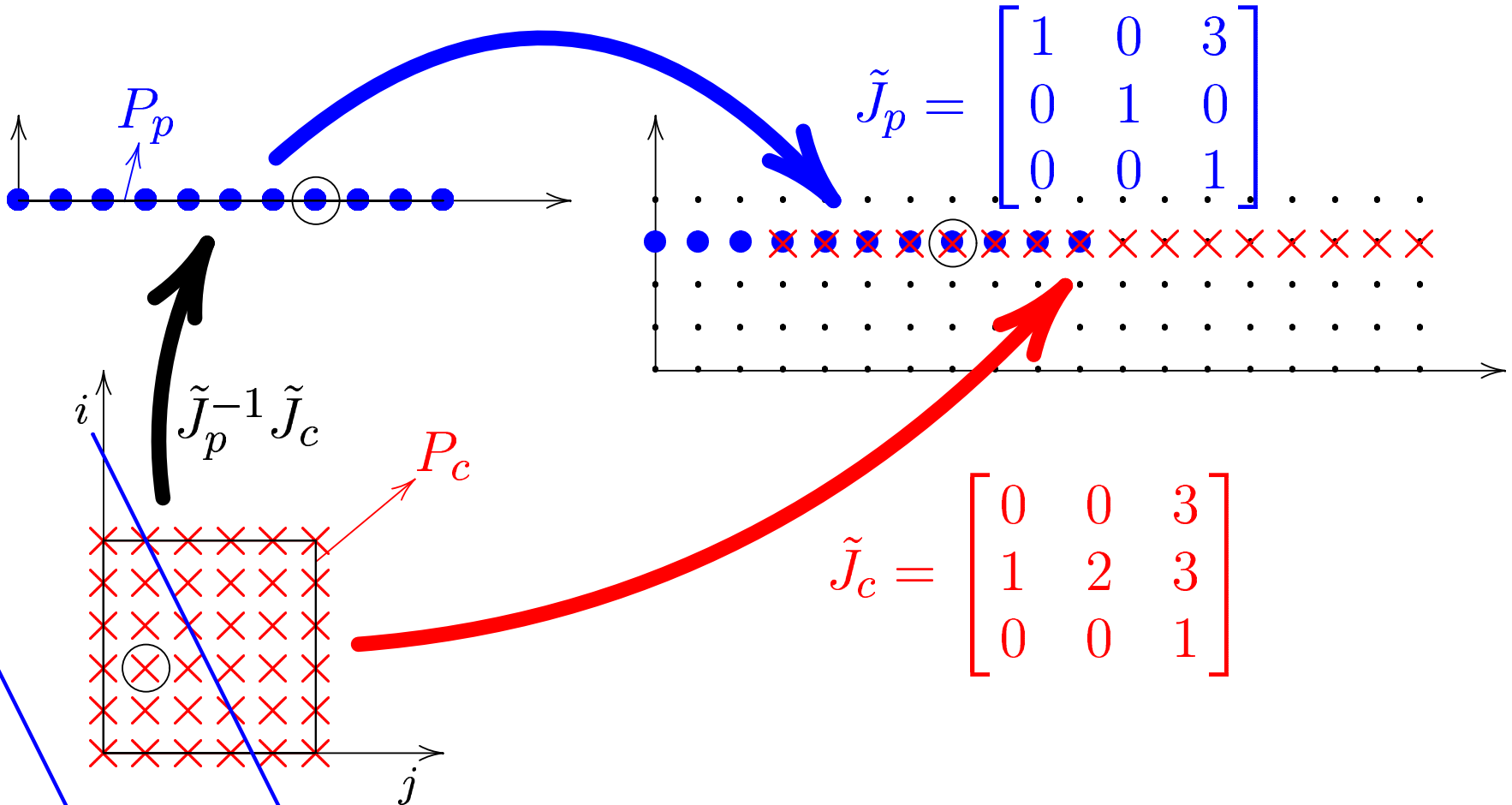
# Dependencies in PDG



# Dependencies in PDG

production space

array space

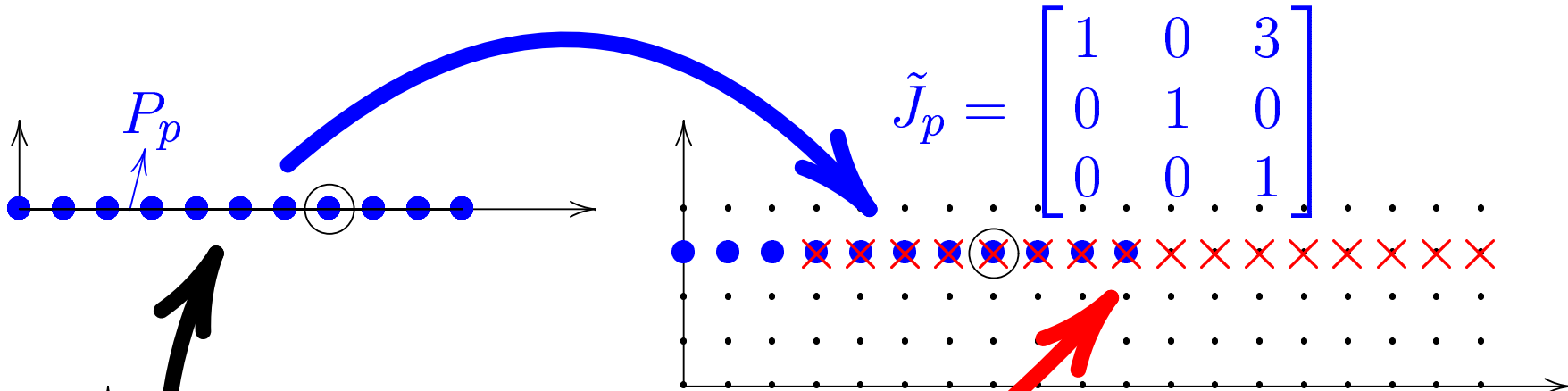


consumption space

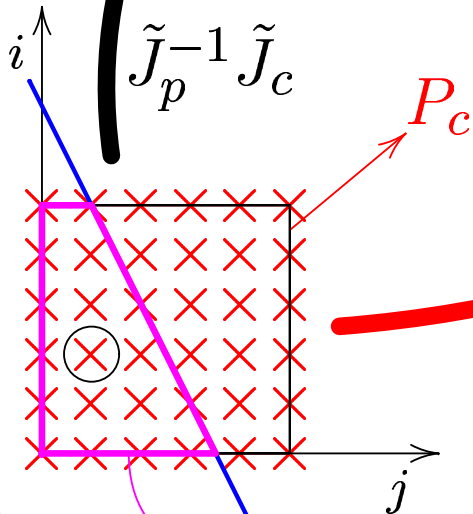
# Dependencies in PDG

production space

array space



$$\tilde{J}_p = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

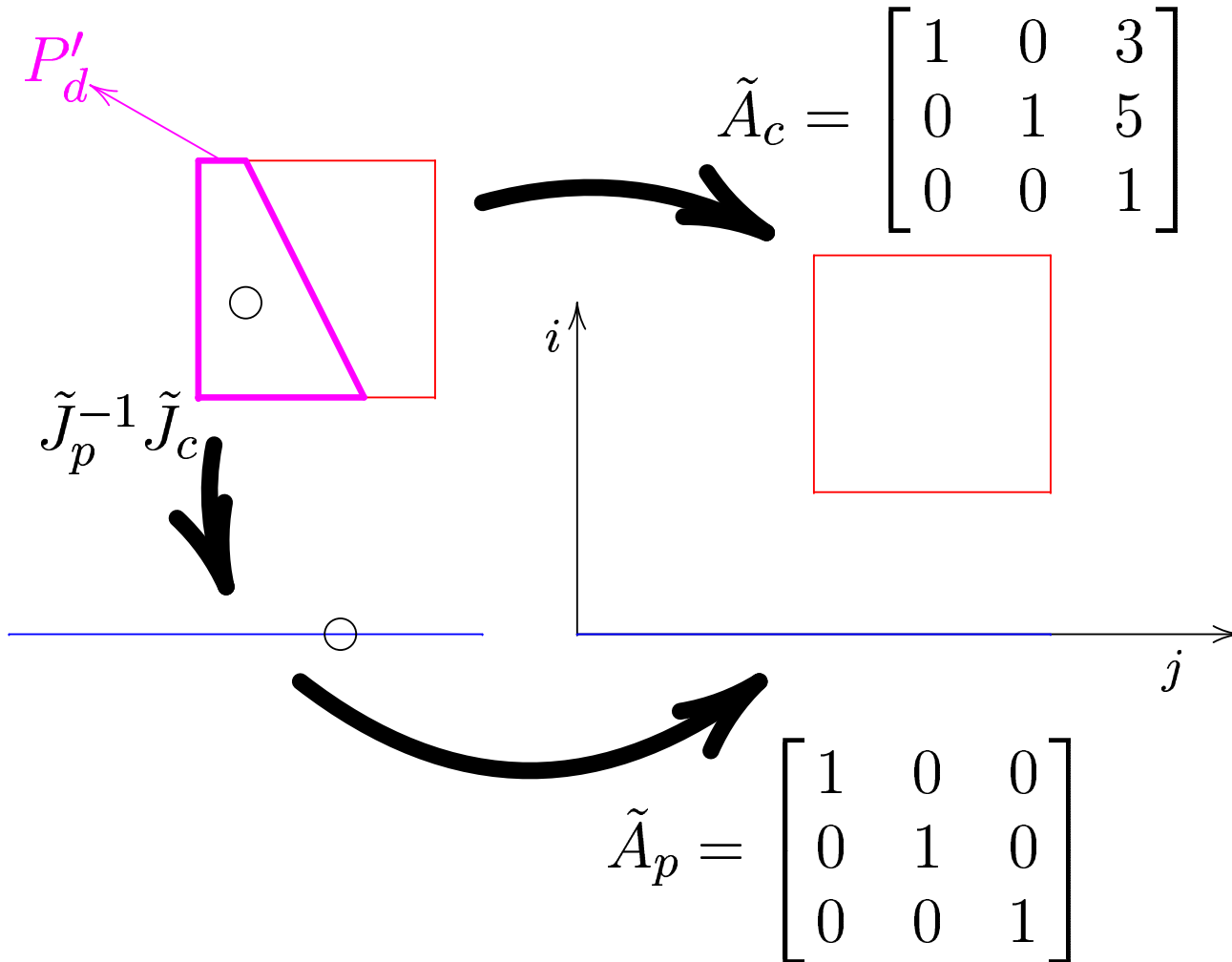


$$\tilde{J}_c = \begin{bmatrix} 0 & 0 & 3 \\ 1 & 2 & 3 \\ 0 & 0 & 1 \end{bmatrix}$$

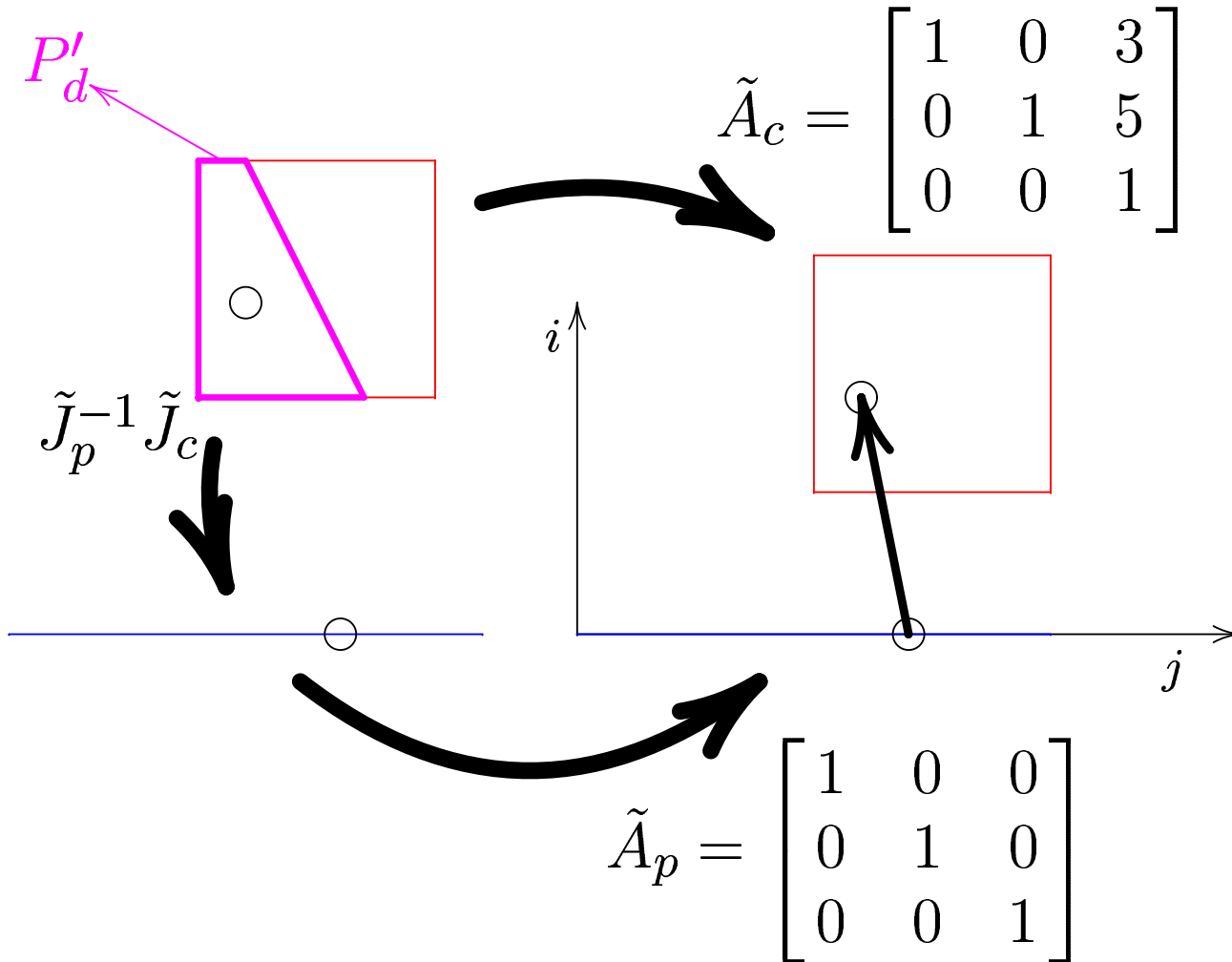
$$\text{PreImg}(P_p, \tilde{J}_p^{-1} \tilde{J}_c) \cap P_c = P'_d$$

consumption space

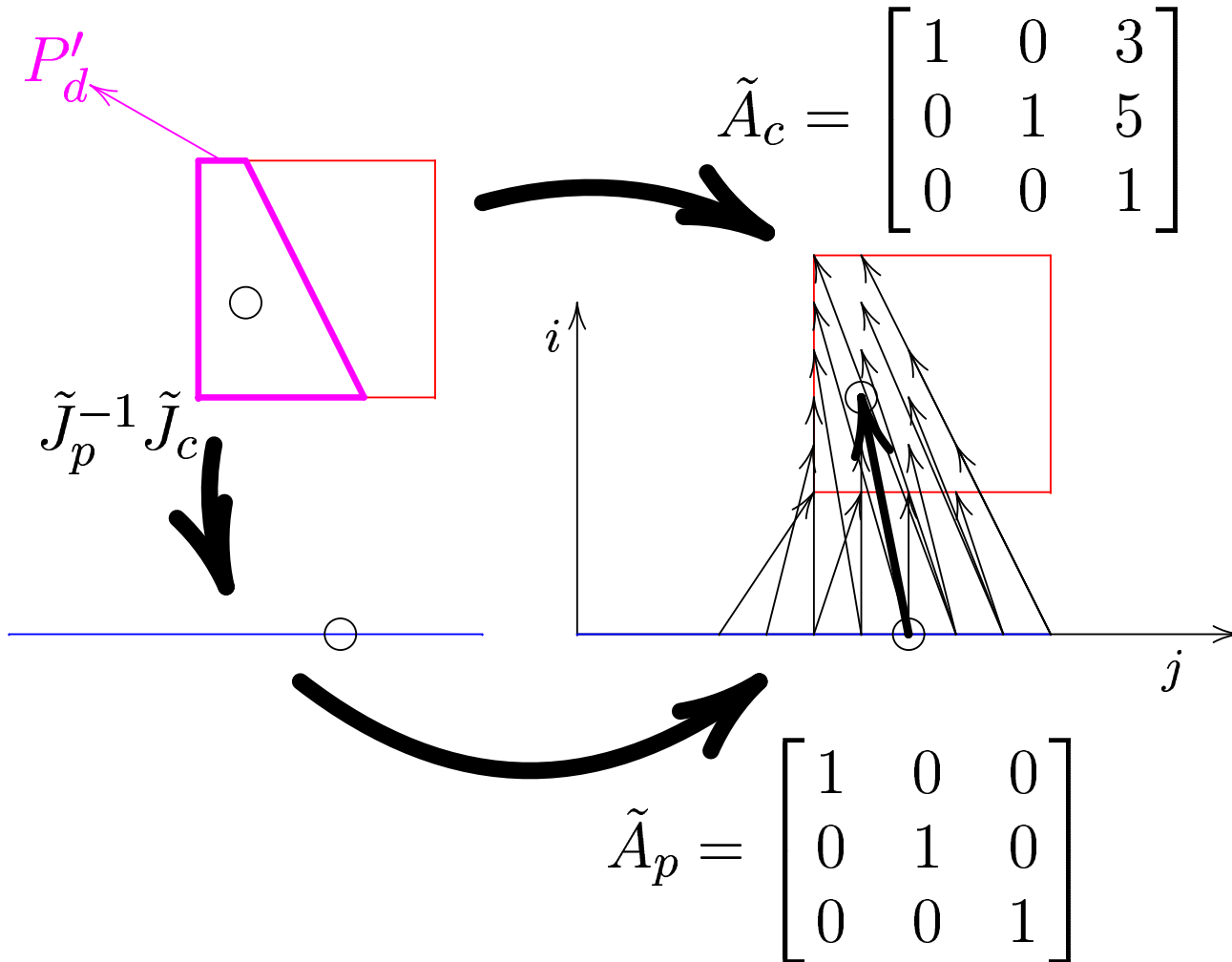
# Dependencies in PDG



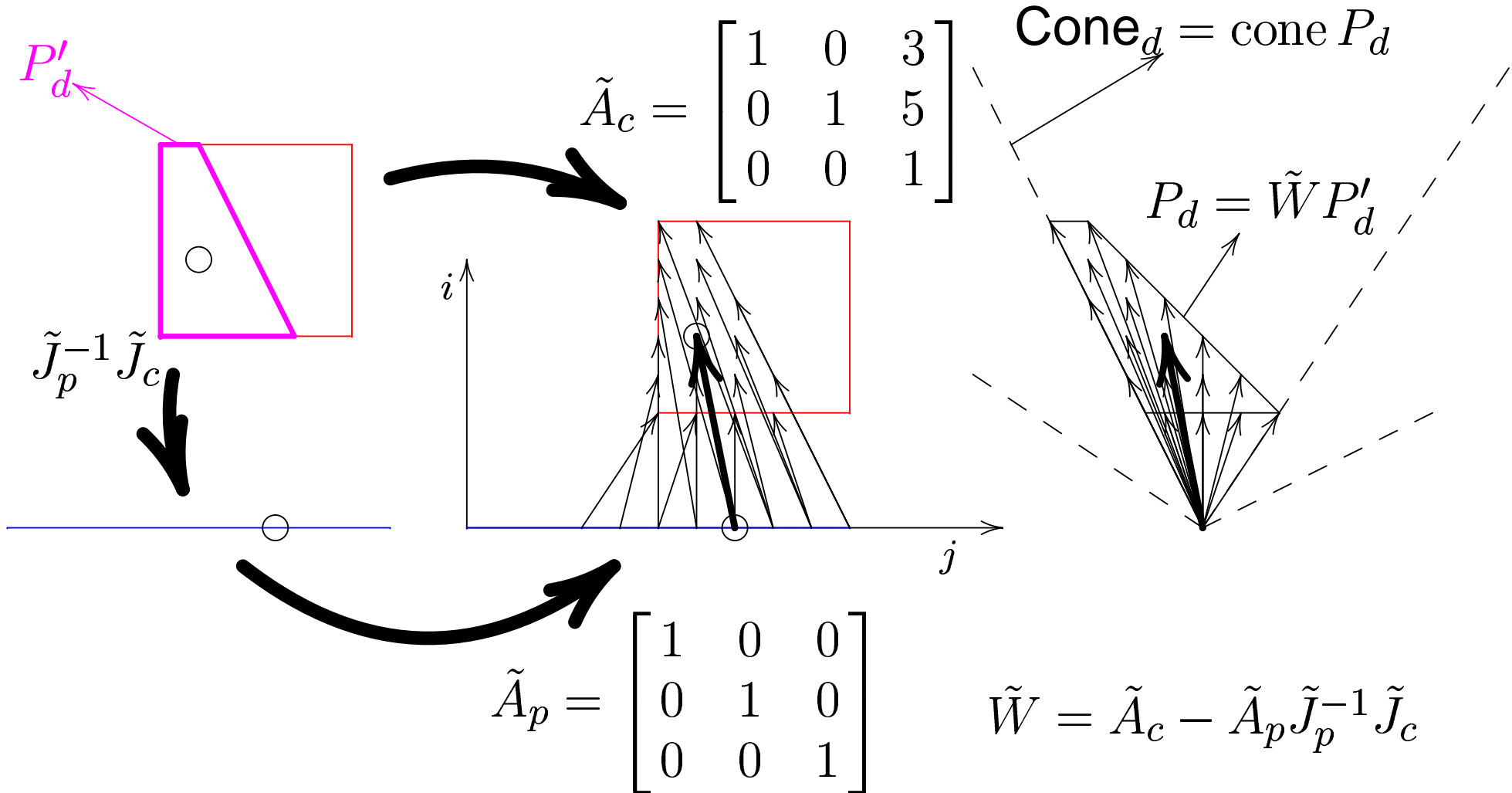
# Dependencies in PDG



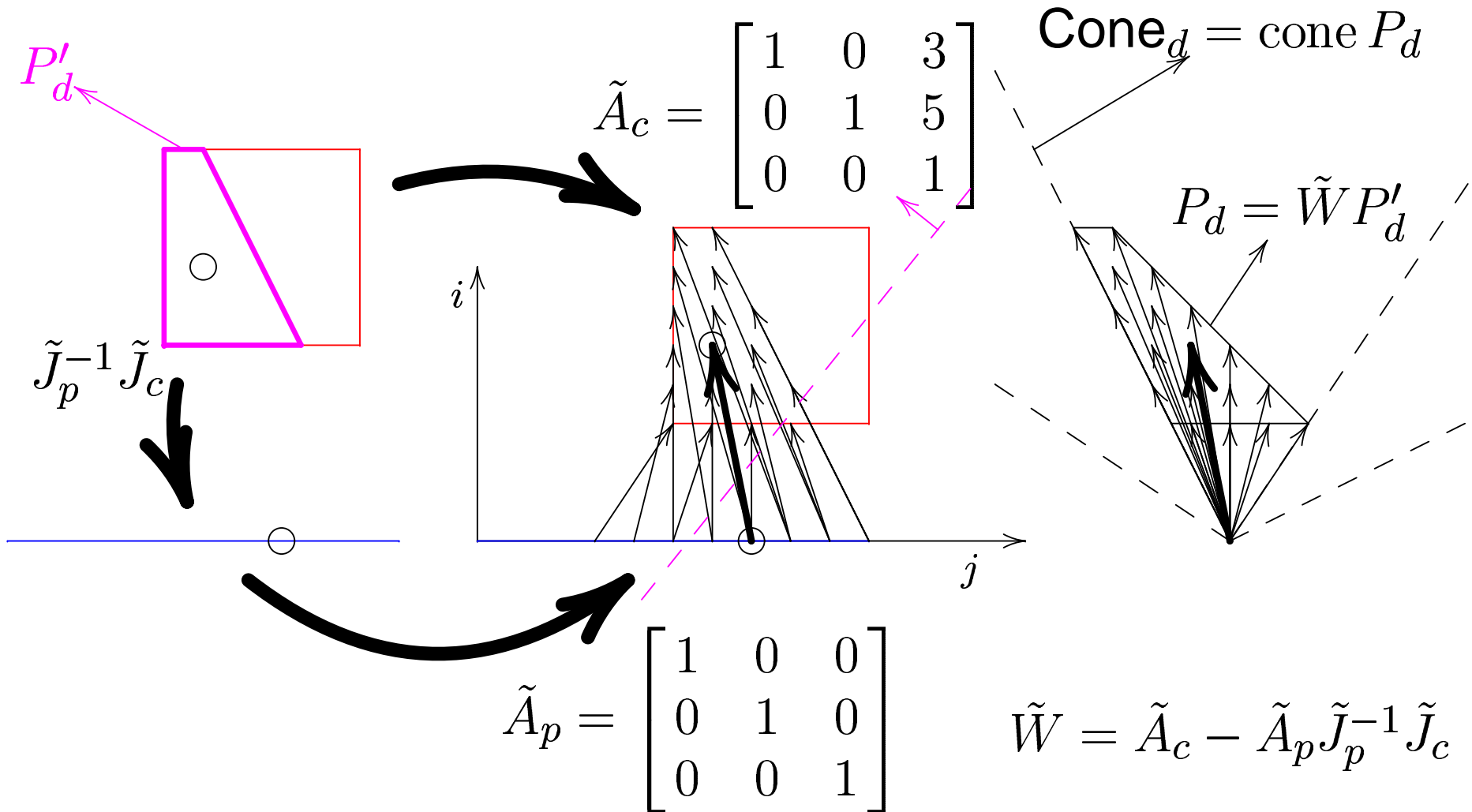
# Dependencies in PDG



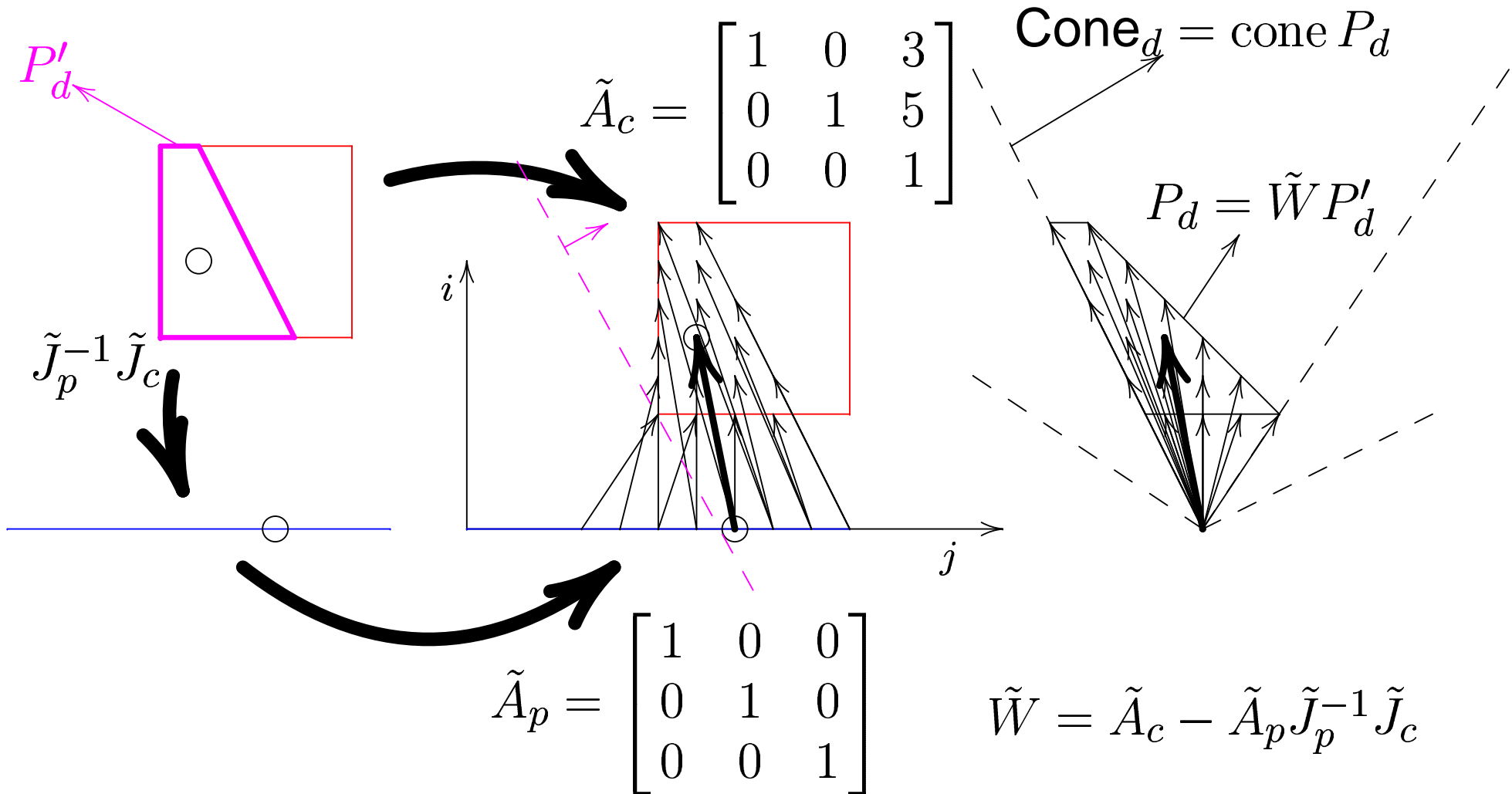
# Dependencies in PDG



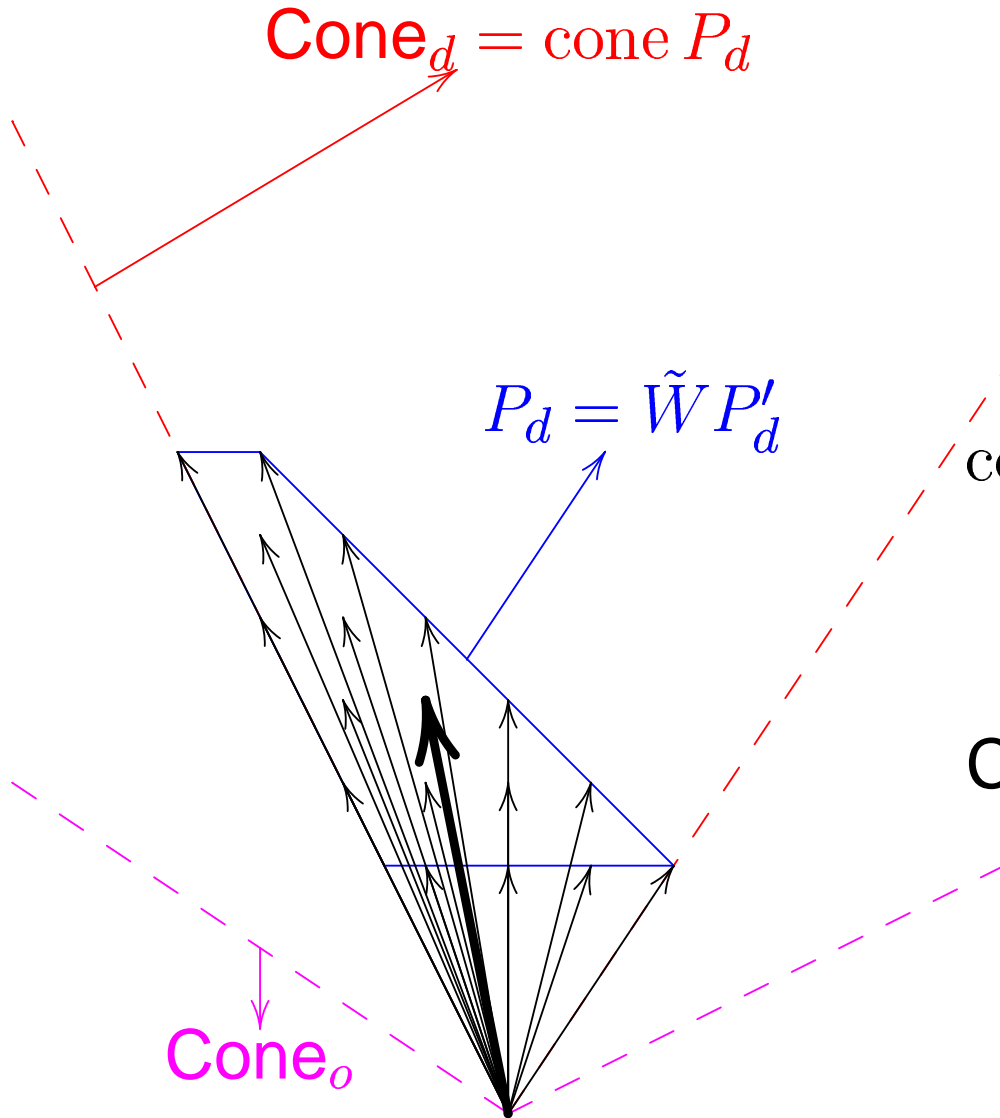
# Dependencies in PDG



# Dependencies in PDG



# Distance vector polytope/cone



$$\text{cone} \{ \vec{d}_i \} = \left\{ \sum_i \lambda_i \vec{d}_i \mid \lambda_i \geq 0 \right\}$$

$$\text{Cone}_o = \left\{ \vec{\pi} \mid \forall \vec{d} \in P_d, \vec{\pi}^T \vec{d} \geq 0 \right\}$$

# Dependencies in PDG

