Location, Location, Location

CPSC 418

November 5

THE UNIVERSITY OF BRITISH COLUMBIA

Re-visiting some assumptions

- Processor centric view of computing – concentrating on processors (why?)
- Stateful processors tightly coupled to memory, no first class communication
- Sharing and statistical multiplexing
- Parallel processing, just in time, not fastest
- Applications not algorithms
- Forever computing, streams not datasets
- Real-world computing
- Just in time and just in place computing

Why “where” is important

Even a little imbalance in work can cause serious loss of efficiency and to do better we need to “share” the work. (Amdahl’s law)

Why “where” is important

Even a little imbalance in work can cause serious loss of efficiency and to do better we need to “share” the work.

Suppose we have 5 painters each painting 5 rooms but one of the rooms is twice the size.

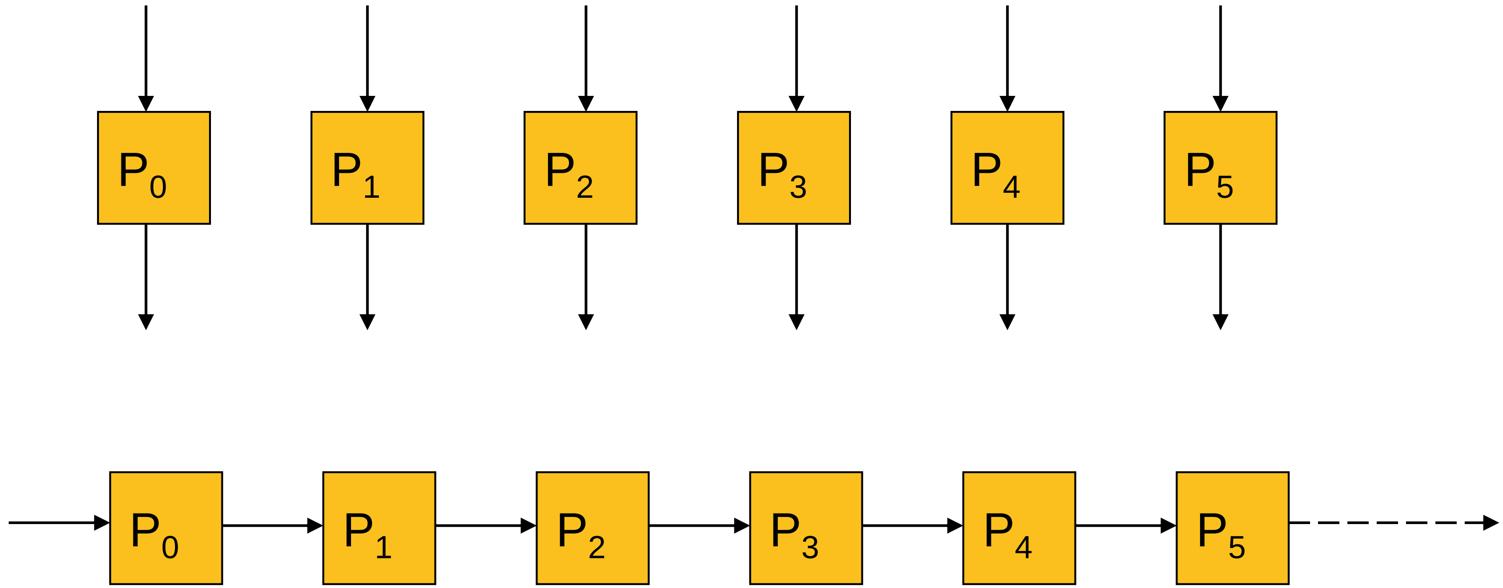
What does Amdahl’s model tell us?

$$S(p) = \frac{1}{\frac{1}{6} + \left(1 - \frac{1}{6}\right)\Big/5} = \frac{1}{\frac{1}{6} + \frac{1}{6}} = 3$$

Efficient is only 60% and in general, assuming one processor does twice as much work, as p increases the efficiency goes to 50%

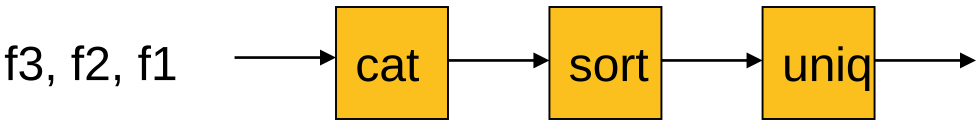
Another way to balance --pipelined

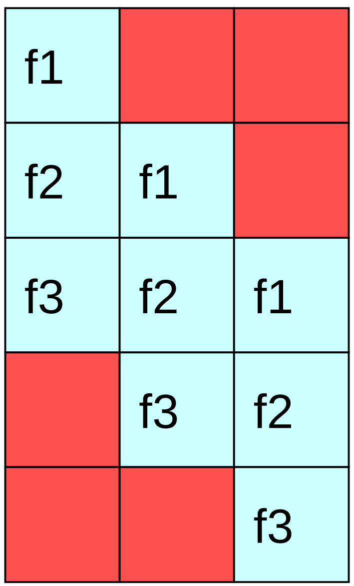
- Problem divided into a series of tasks that have to be completed one after the other (the basis of sequential programming). Each task executed by a separate process or processor.



Example

Unix pipes: cat file | sort | uniq > unique.out





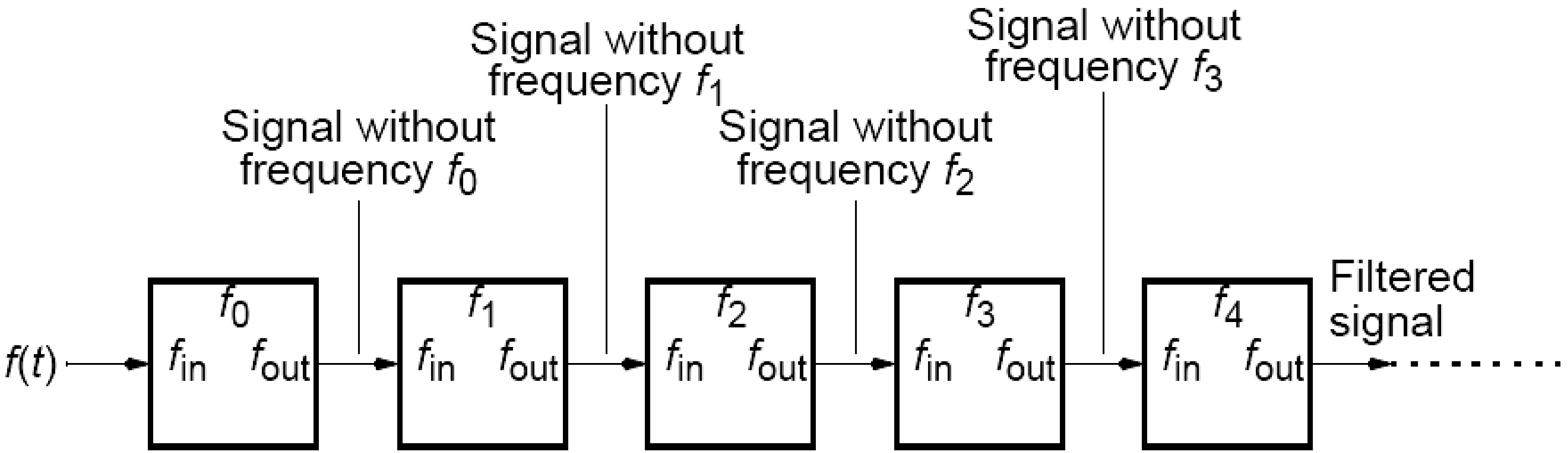
As fast as the slowest part.

Achieves something else, communication from one end of the pipe to the other.

Position data for later processing!

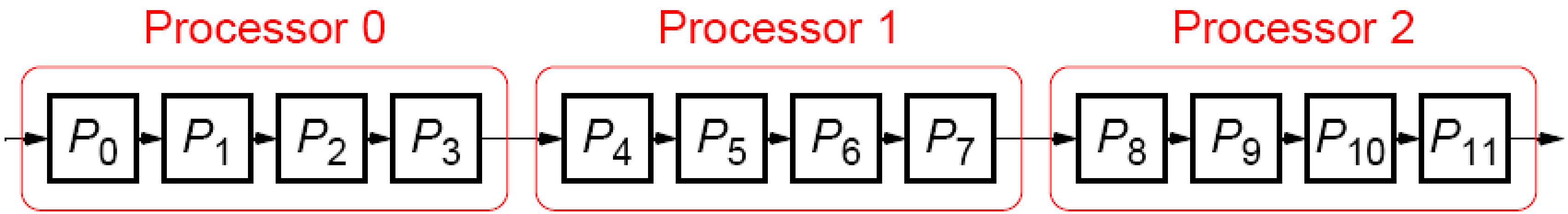
Frequency Example

- Frequency filter - Objective to remove specific frequencies (*f*0, *f*1, *f*2,*f*3, etc.) from a digitized signal, *f*(*t*). Signal enters pipeline from left:



New strategies for load-balancing

- If the number of stages is larger than the number of processors in any pipeline, a group of stages can be assigned to each processor:



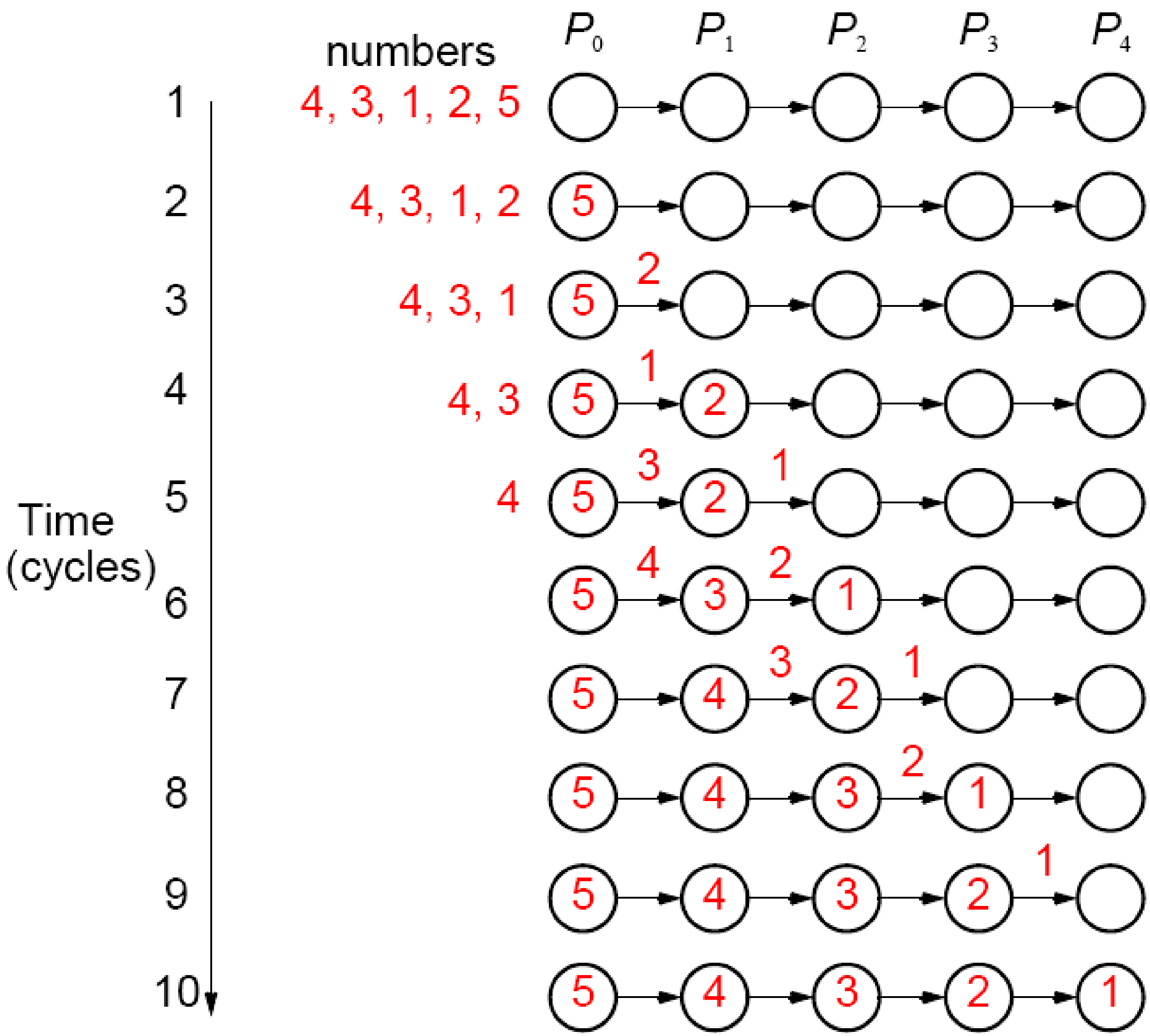
Simple Systolic Computation

- Simple Algorithms – tend to be fine-grain

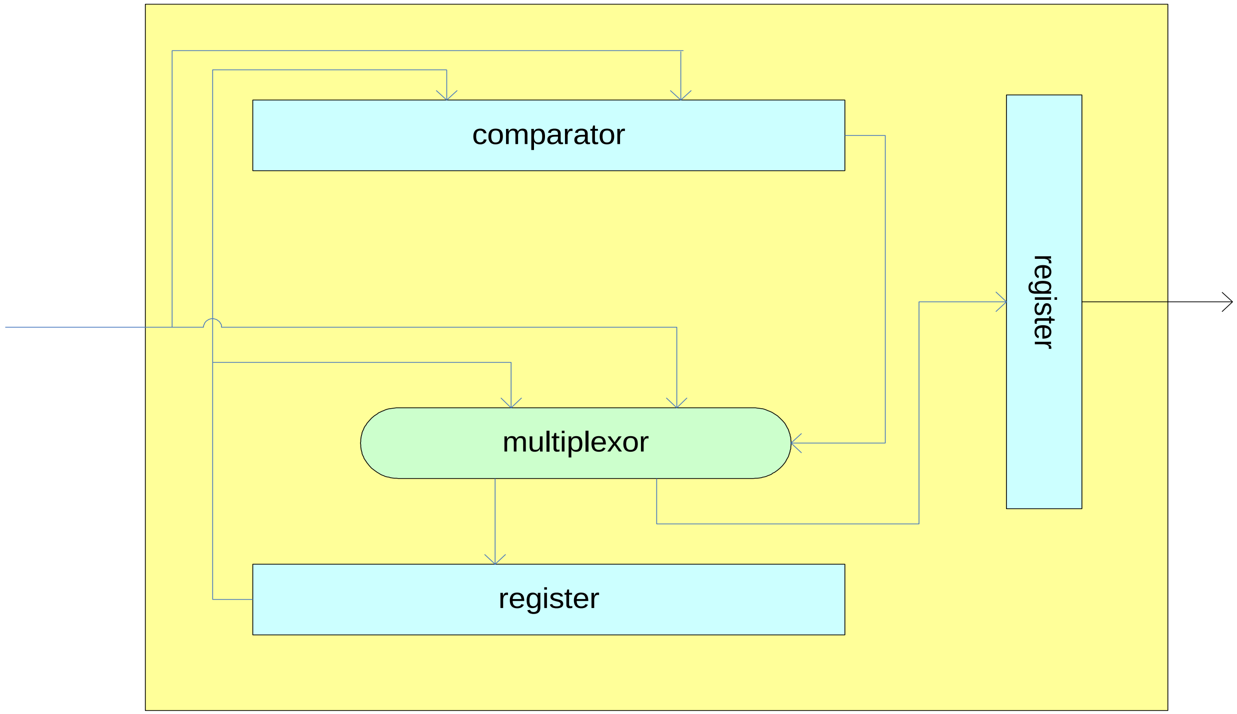
- Capture both (data in motion)
  - Data parallel
  - Functional decomposition (pipelining)

- Computation and Communication
  - Temporal
  - Spatial

Sorting Example



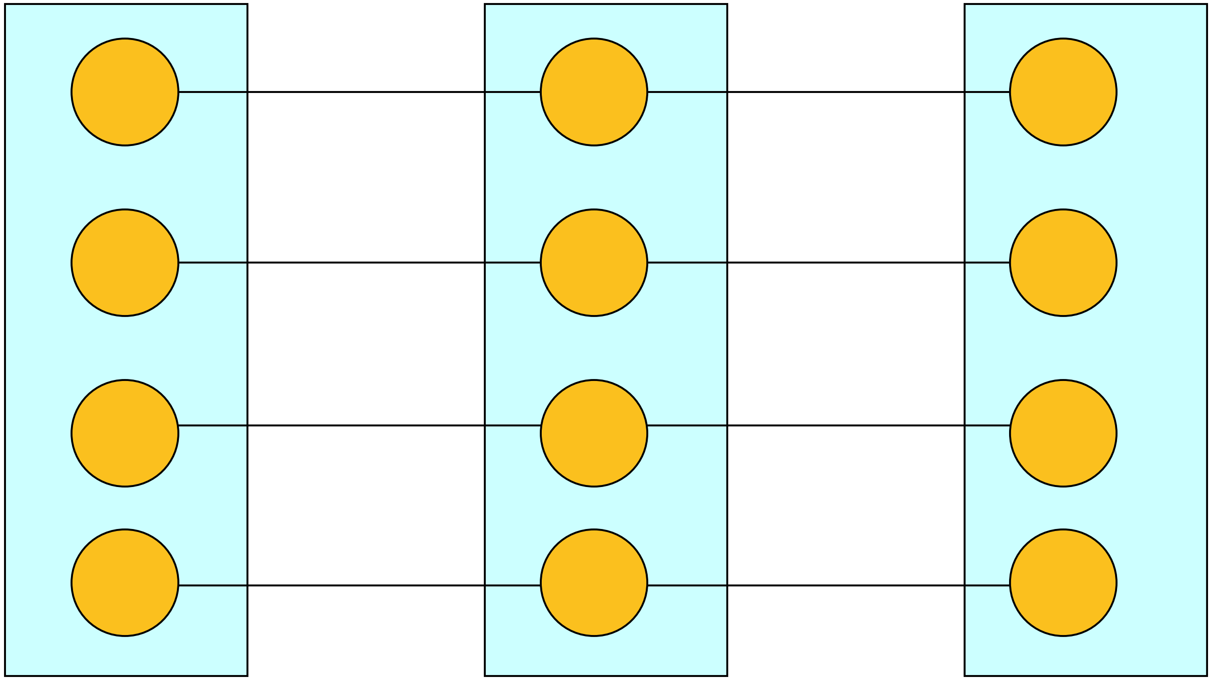
Inside the cell



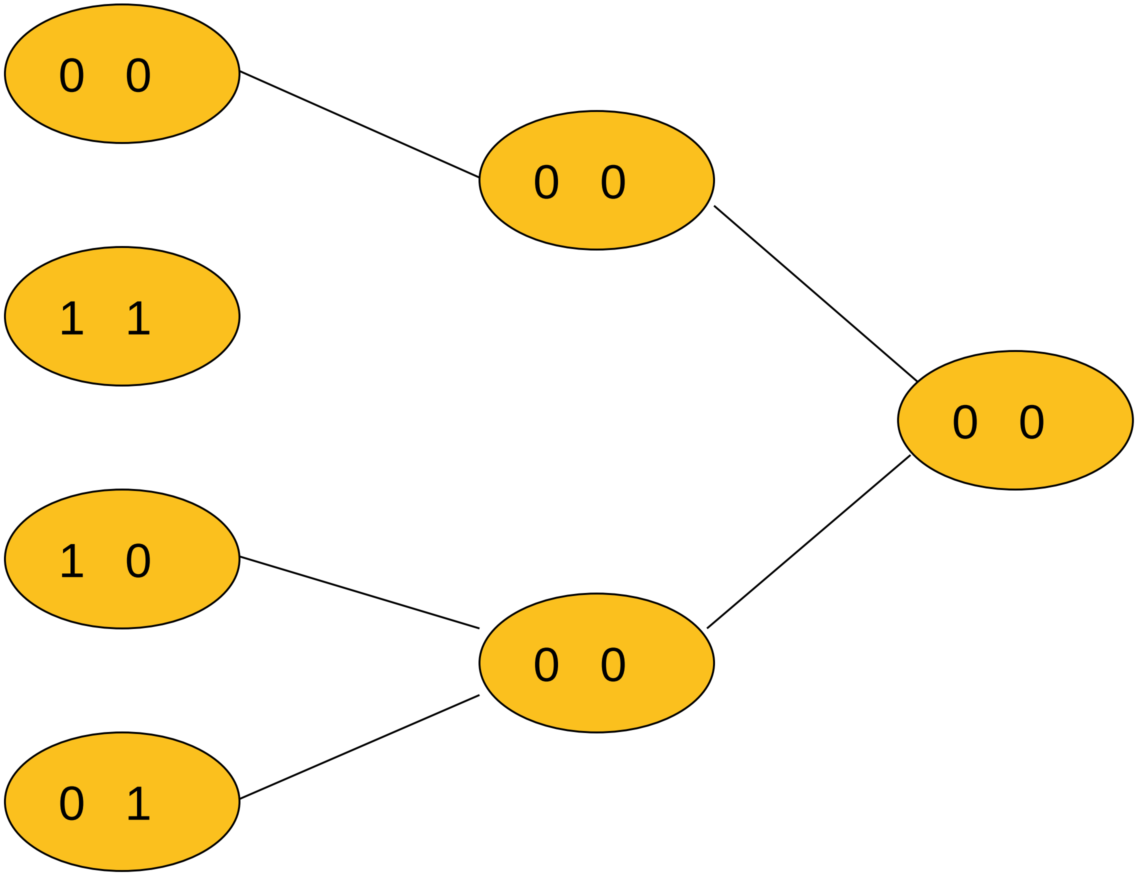
How about output?

- Method 1
- Method 2
- Method 3
- Method 4

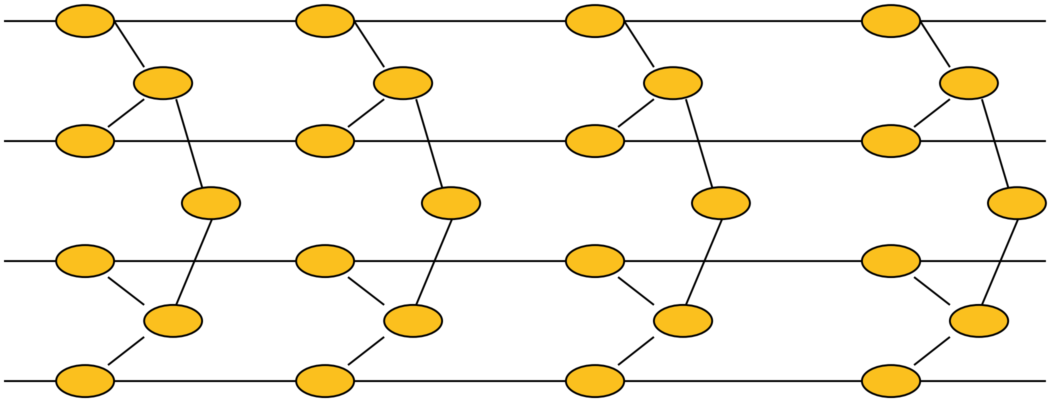
How about the bit level?



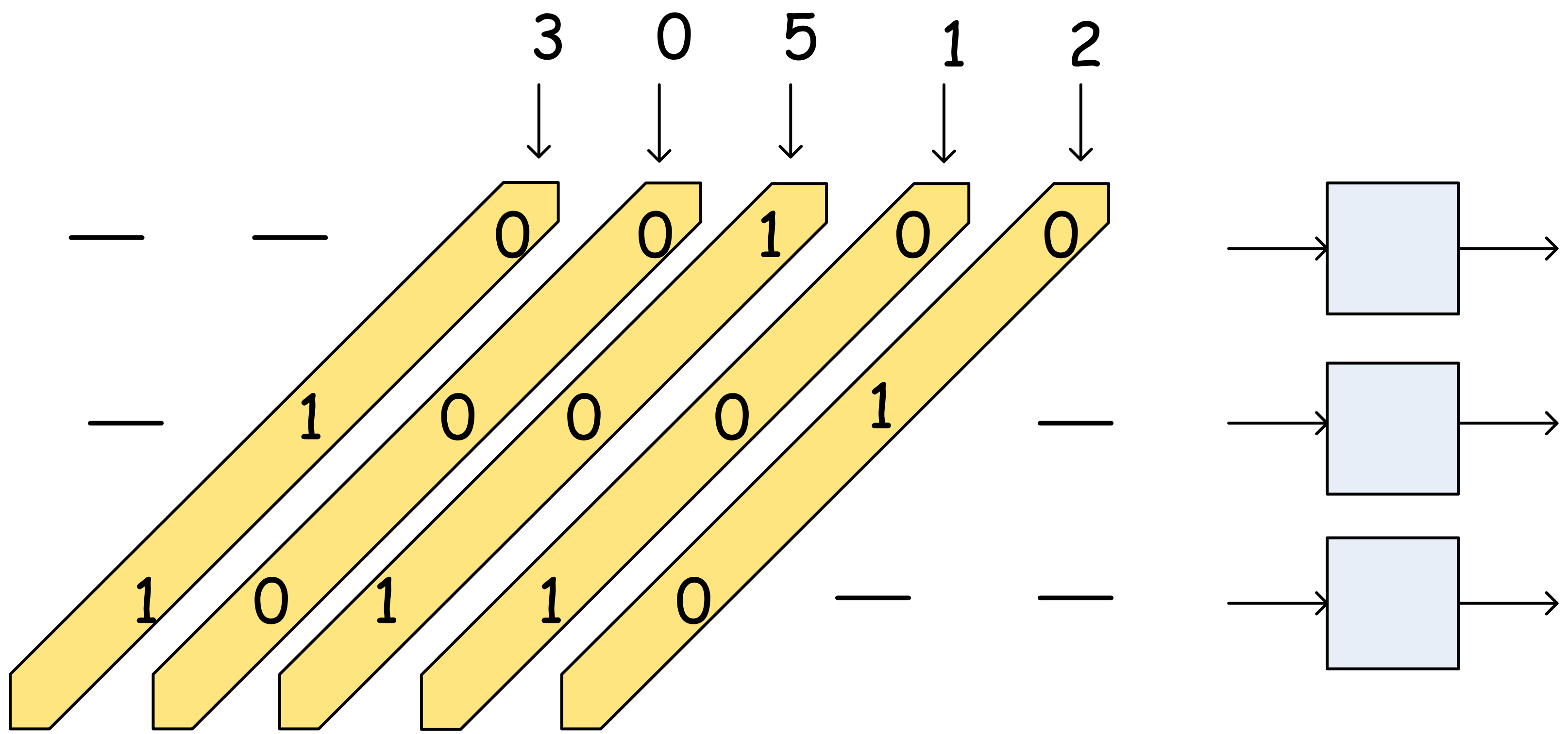
Binary tree comparator



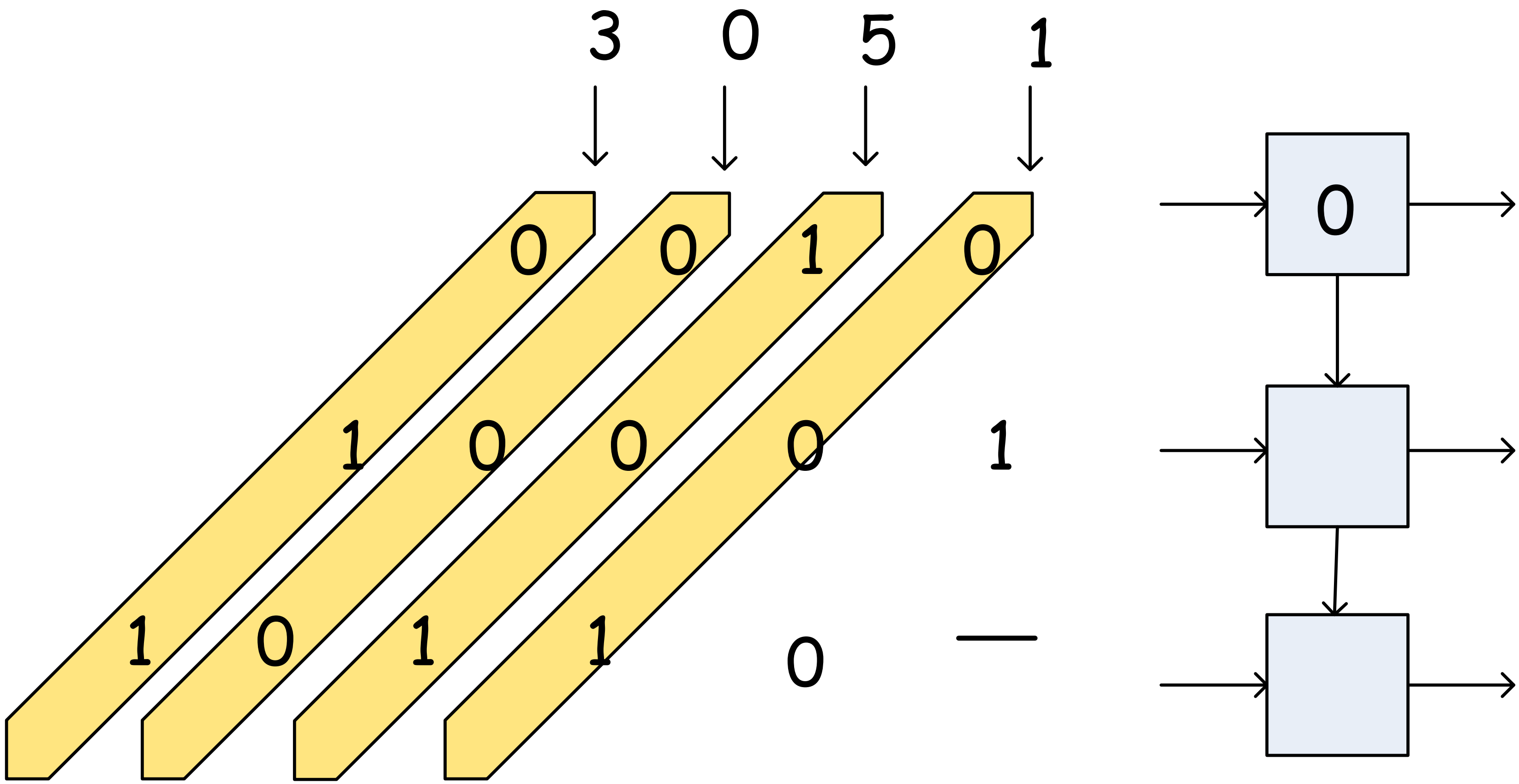
Overall Implementation



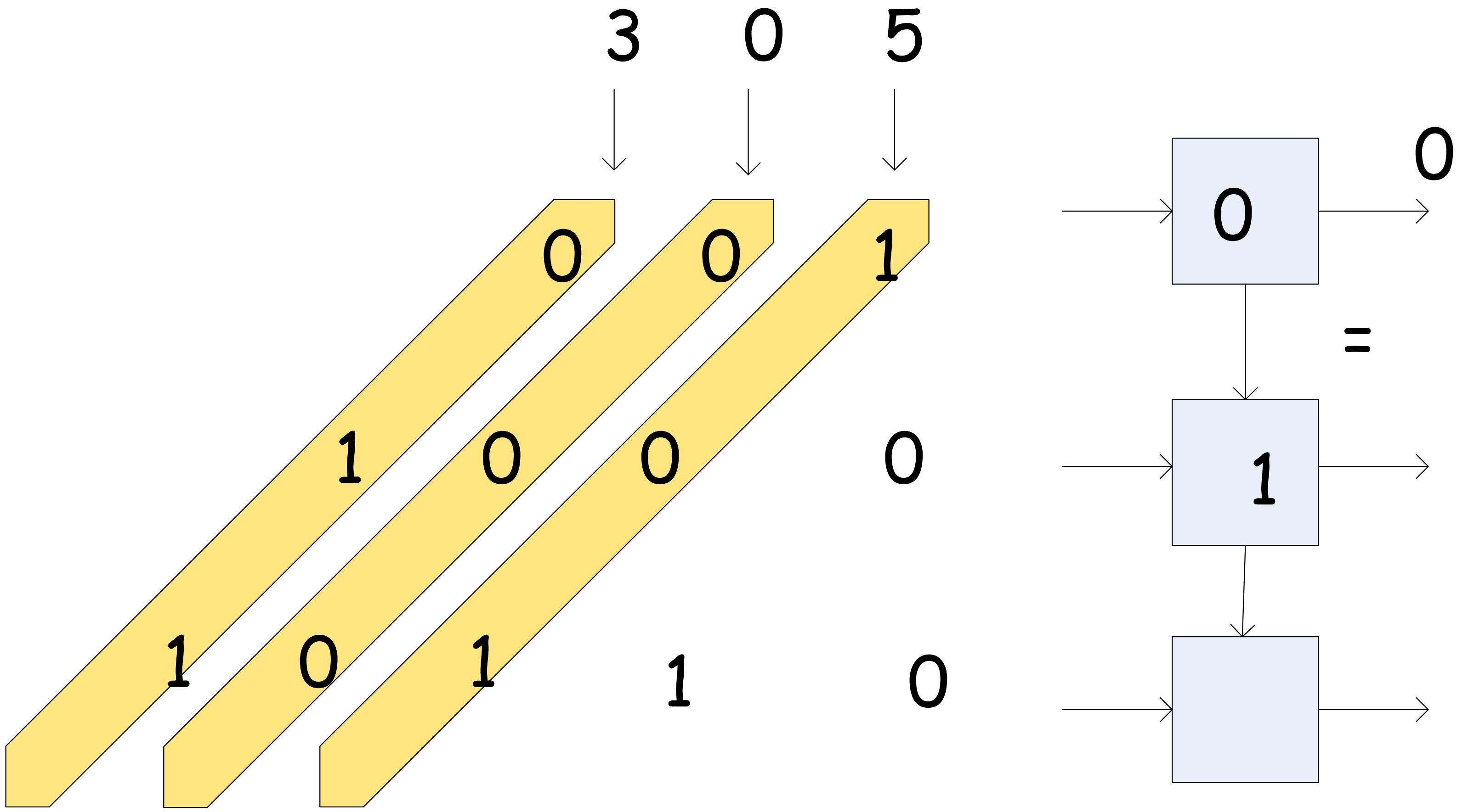
Finding the smallest-initial



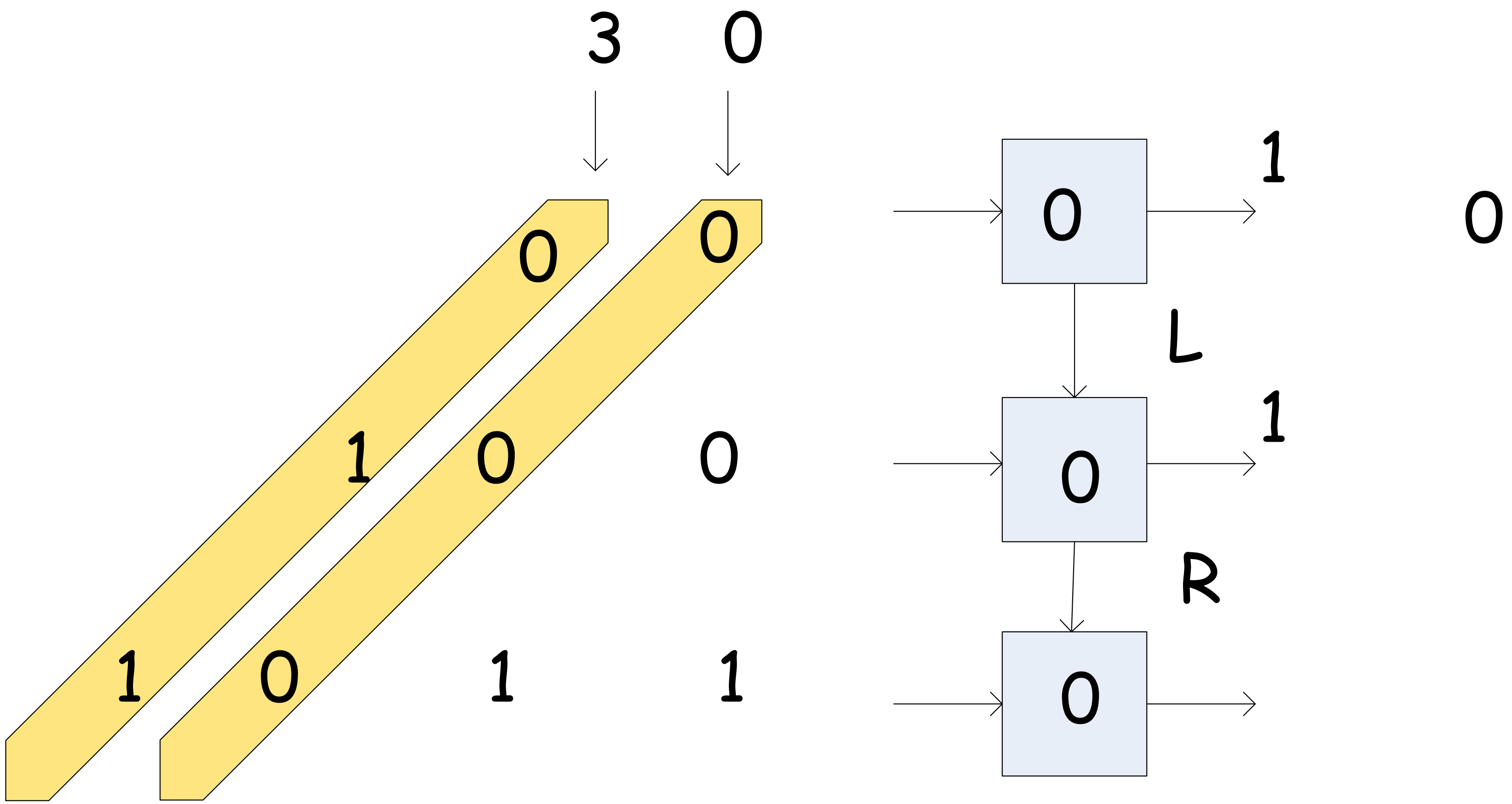
Finding the smallest-step 1



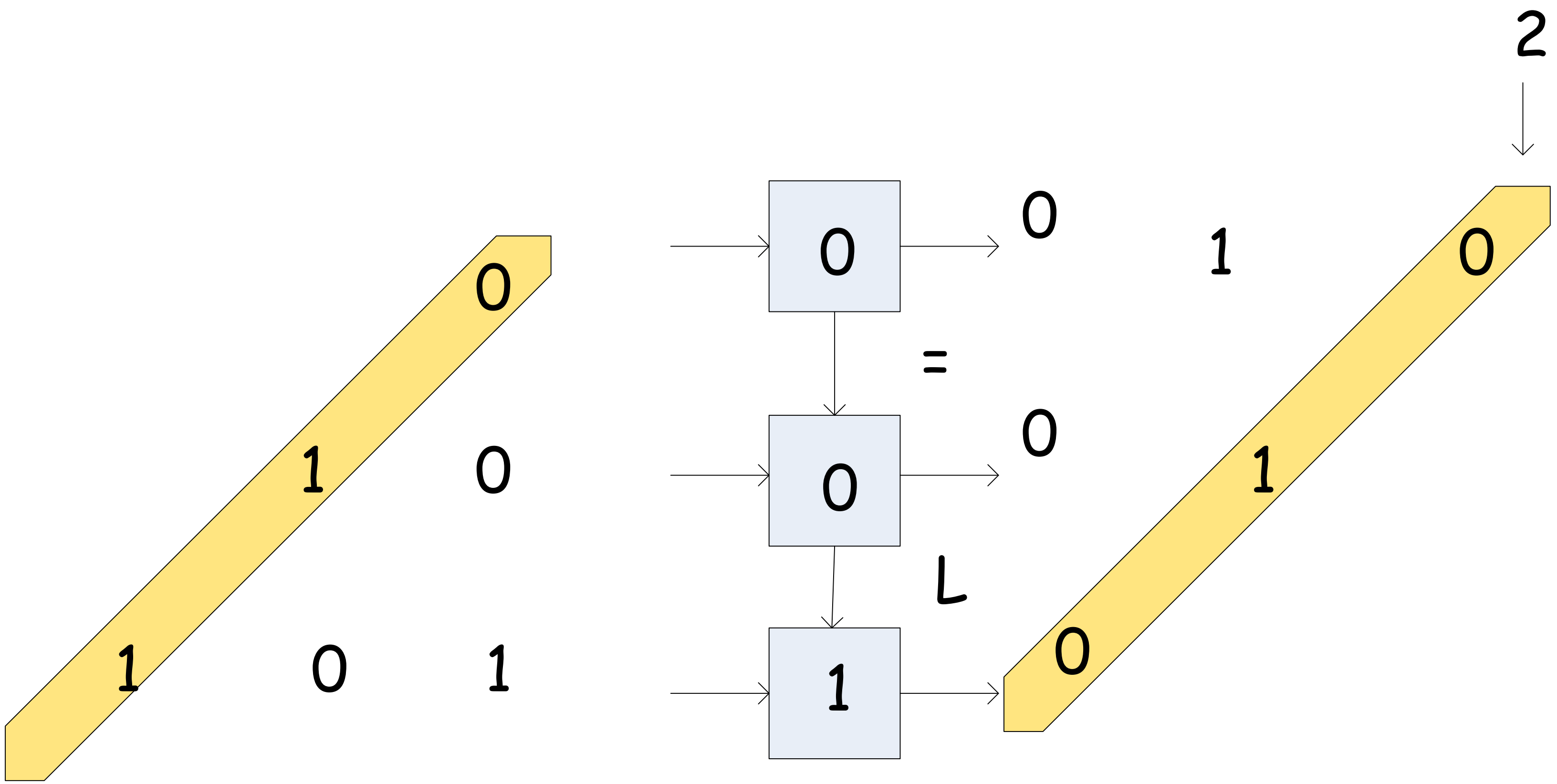
Finding the smallest-step 2



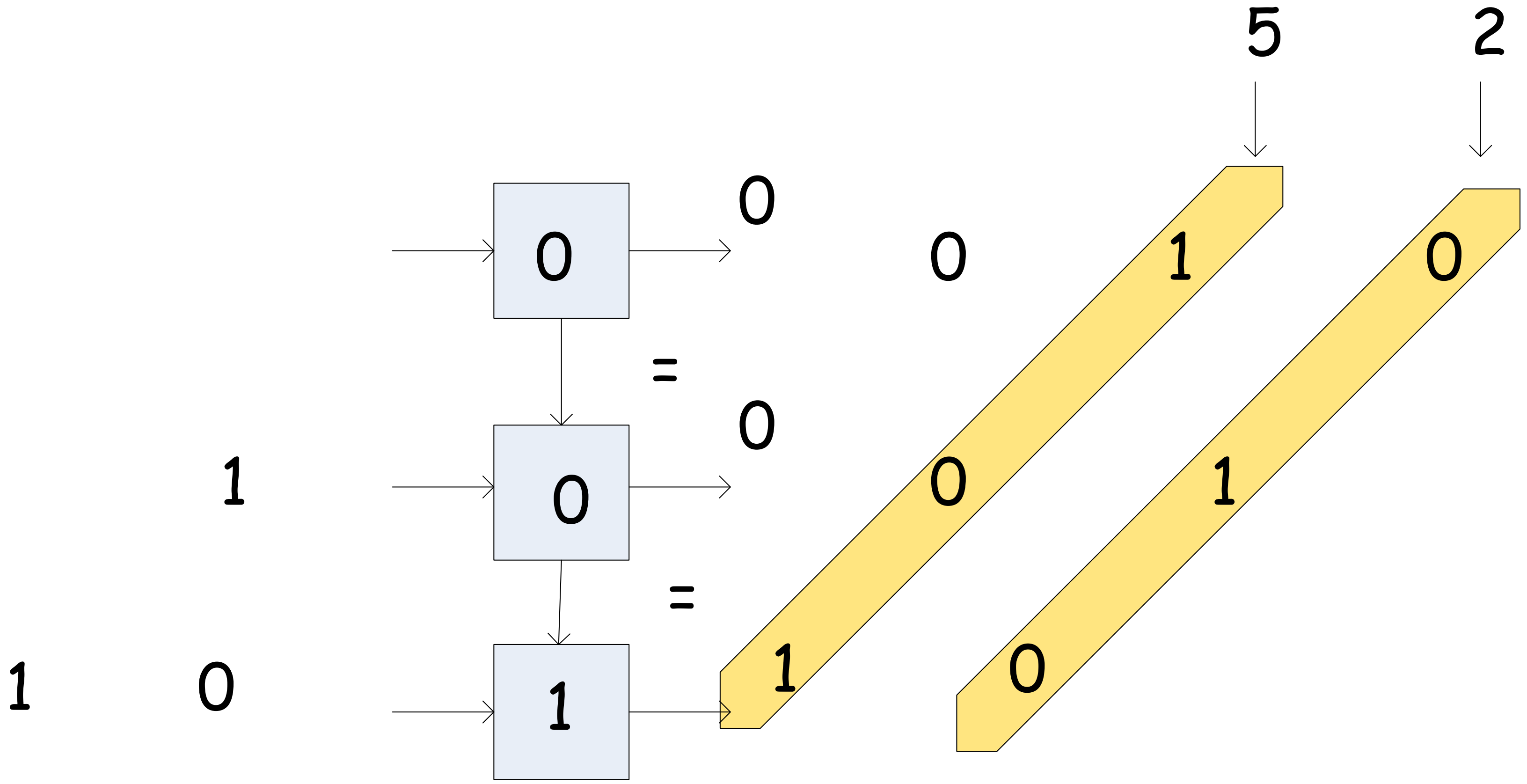
Finding the smallest-step 3



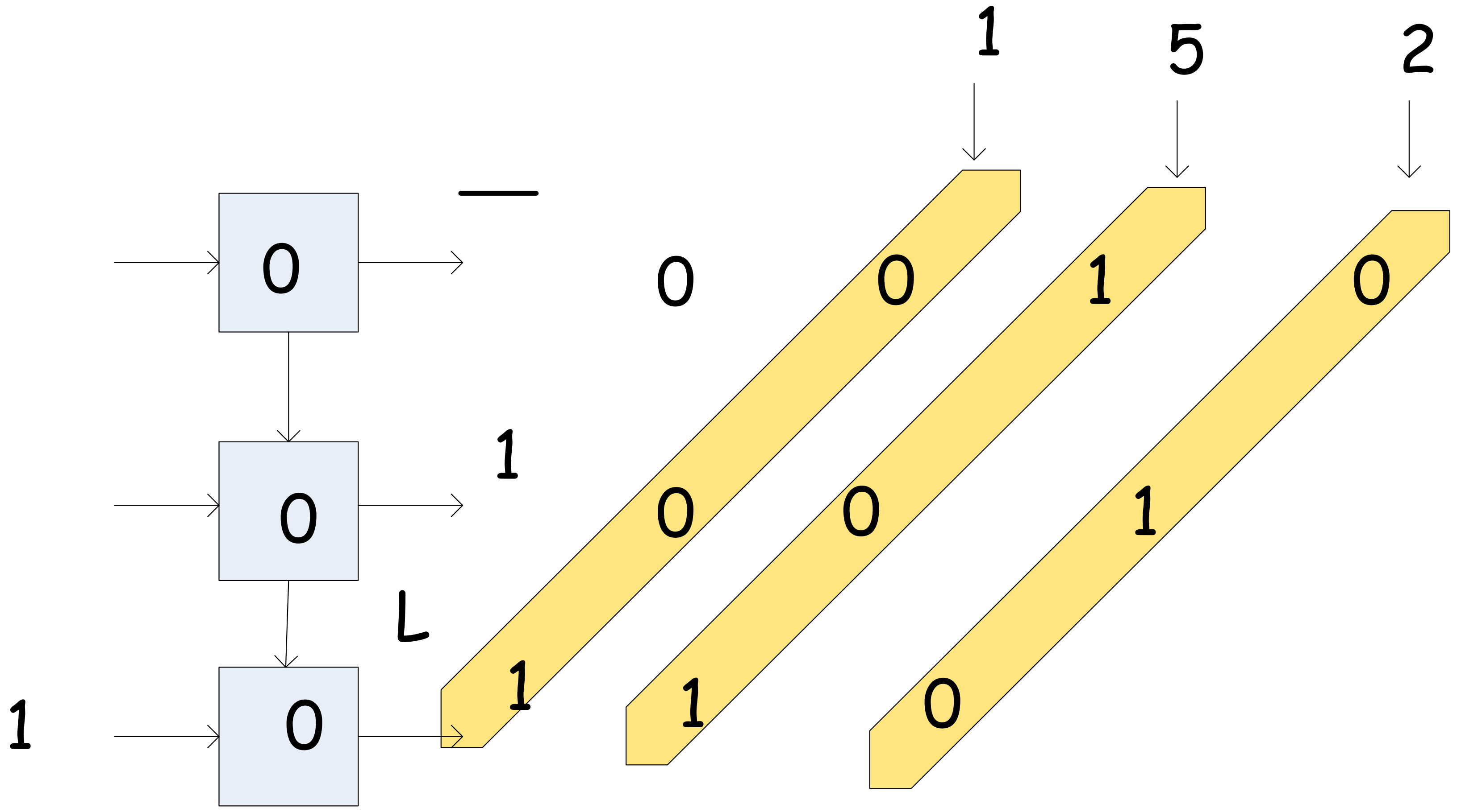
Finding the smallest-step 4



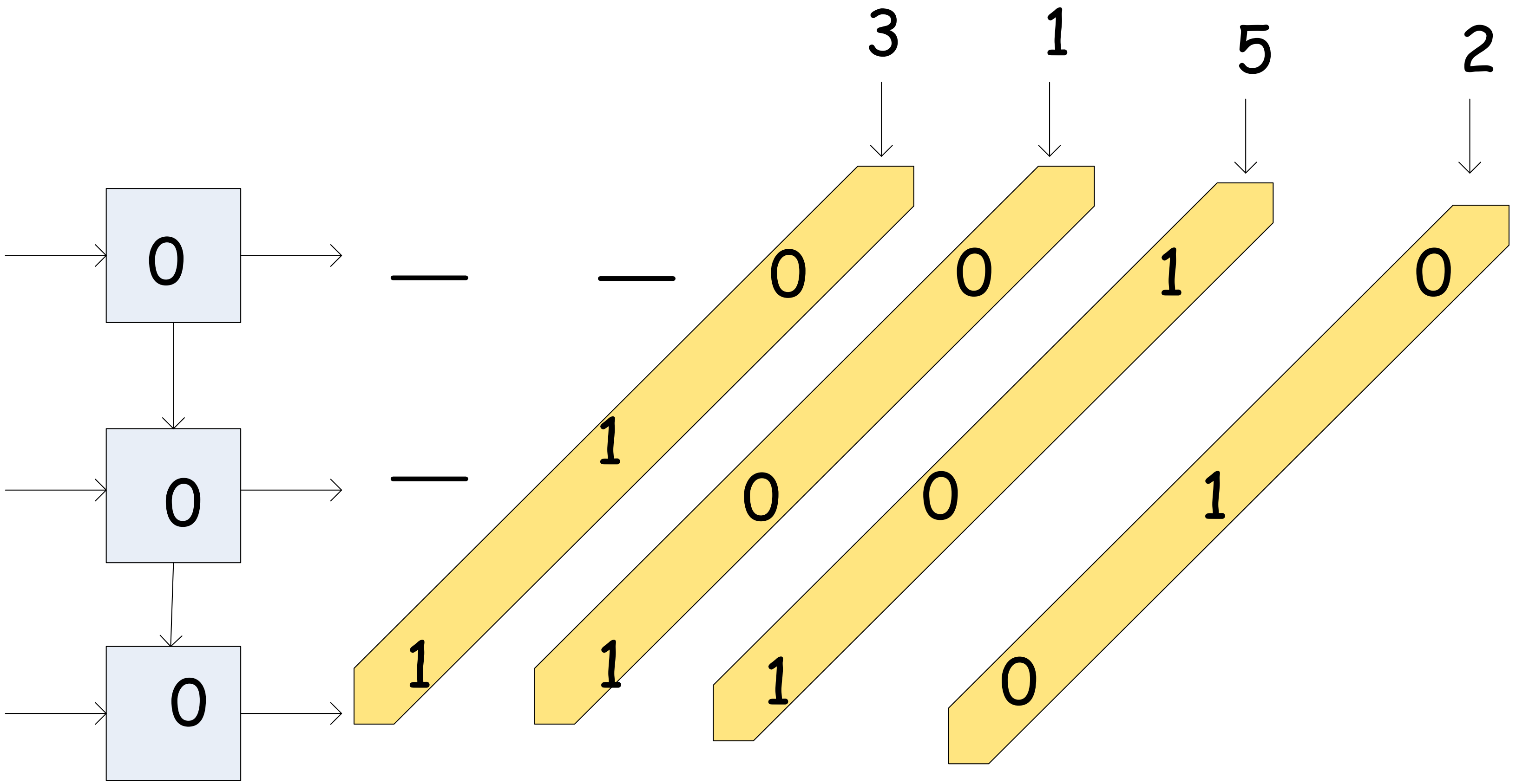
Finding the smallest-step 5



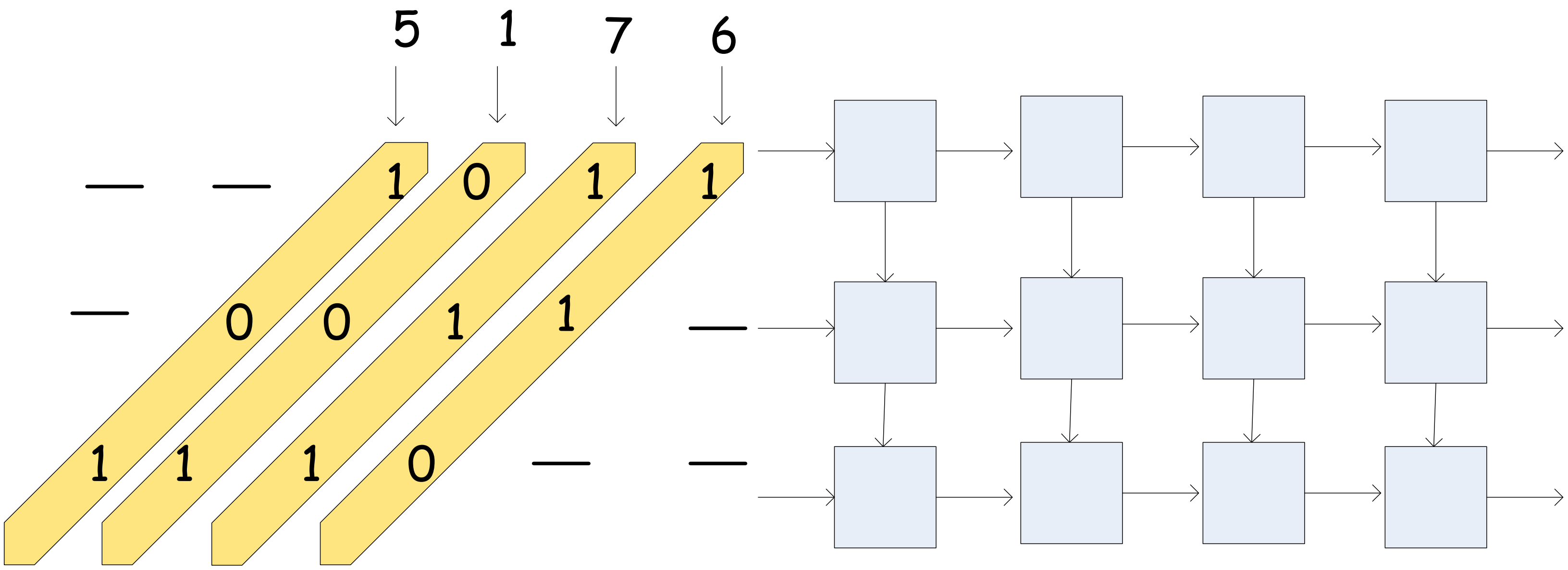
Finding the smallest-step 6



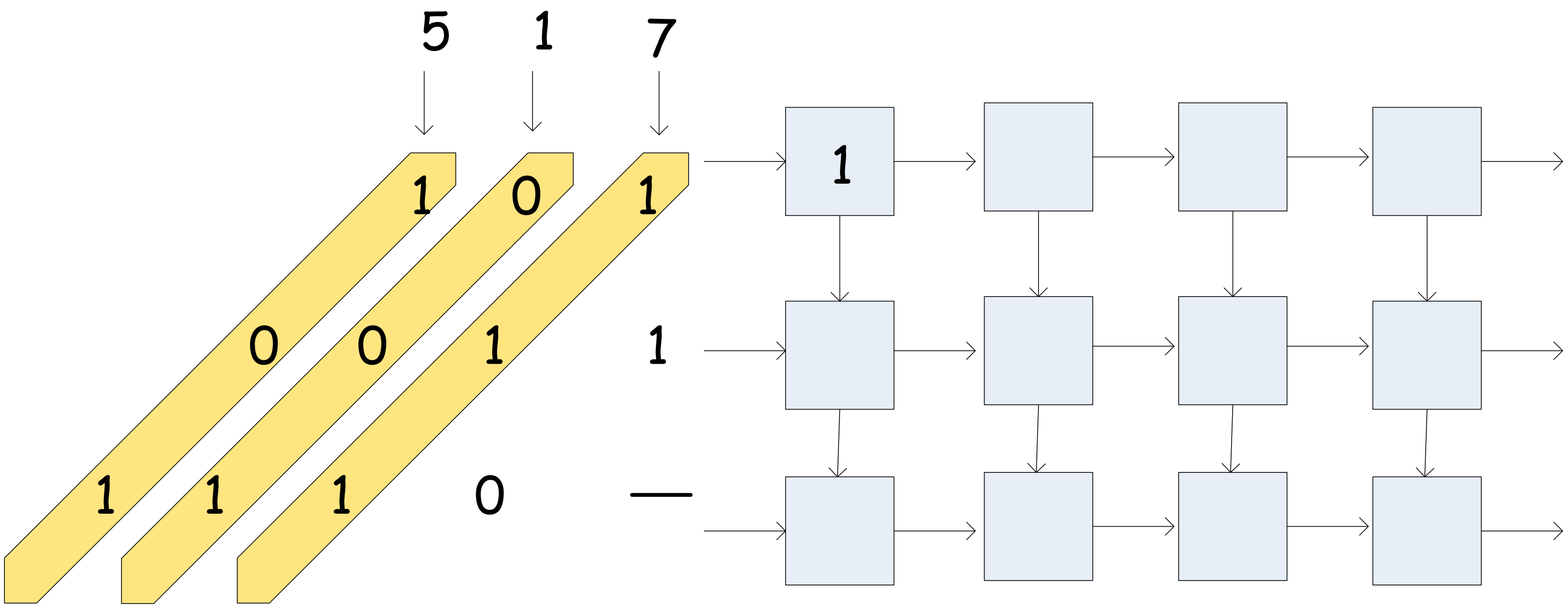
Finding the smallest-step 7



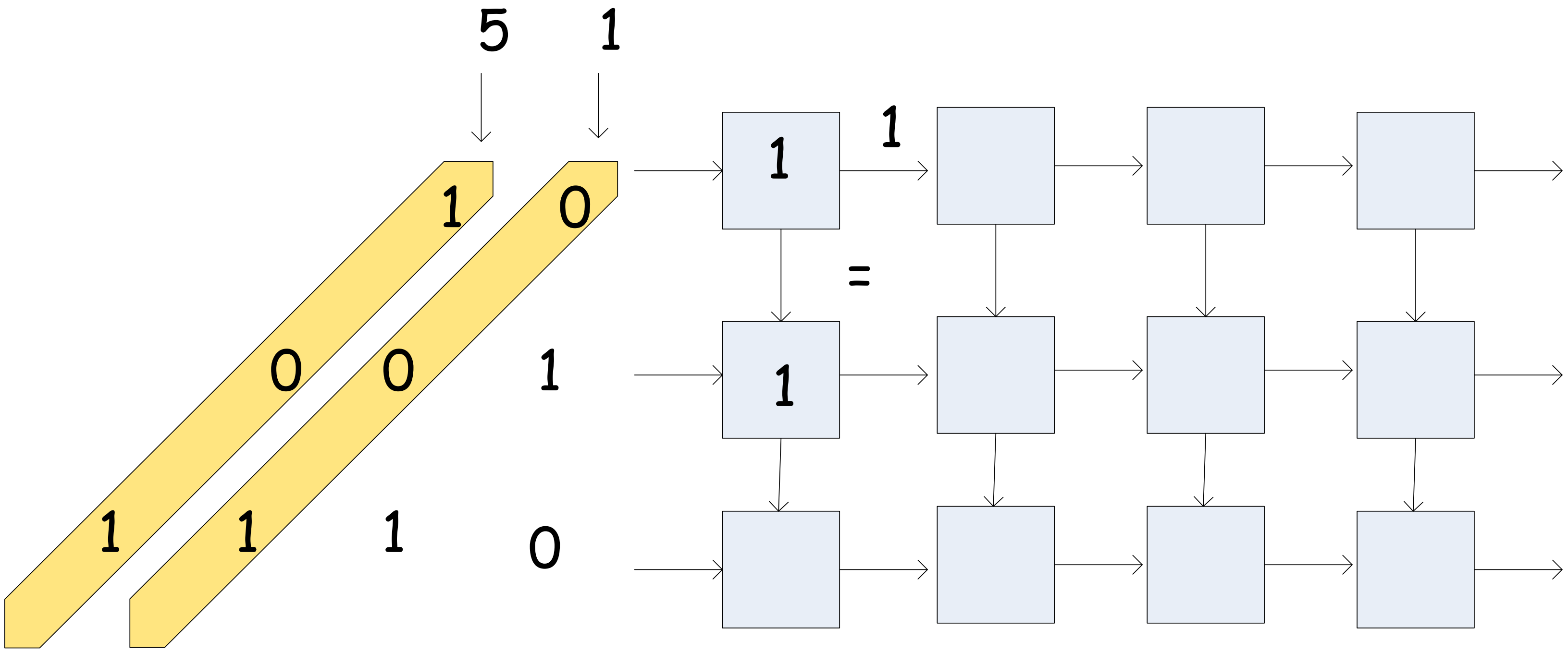
Full Array



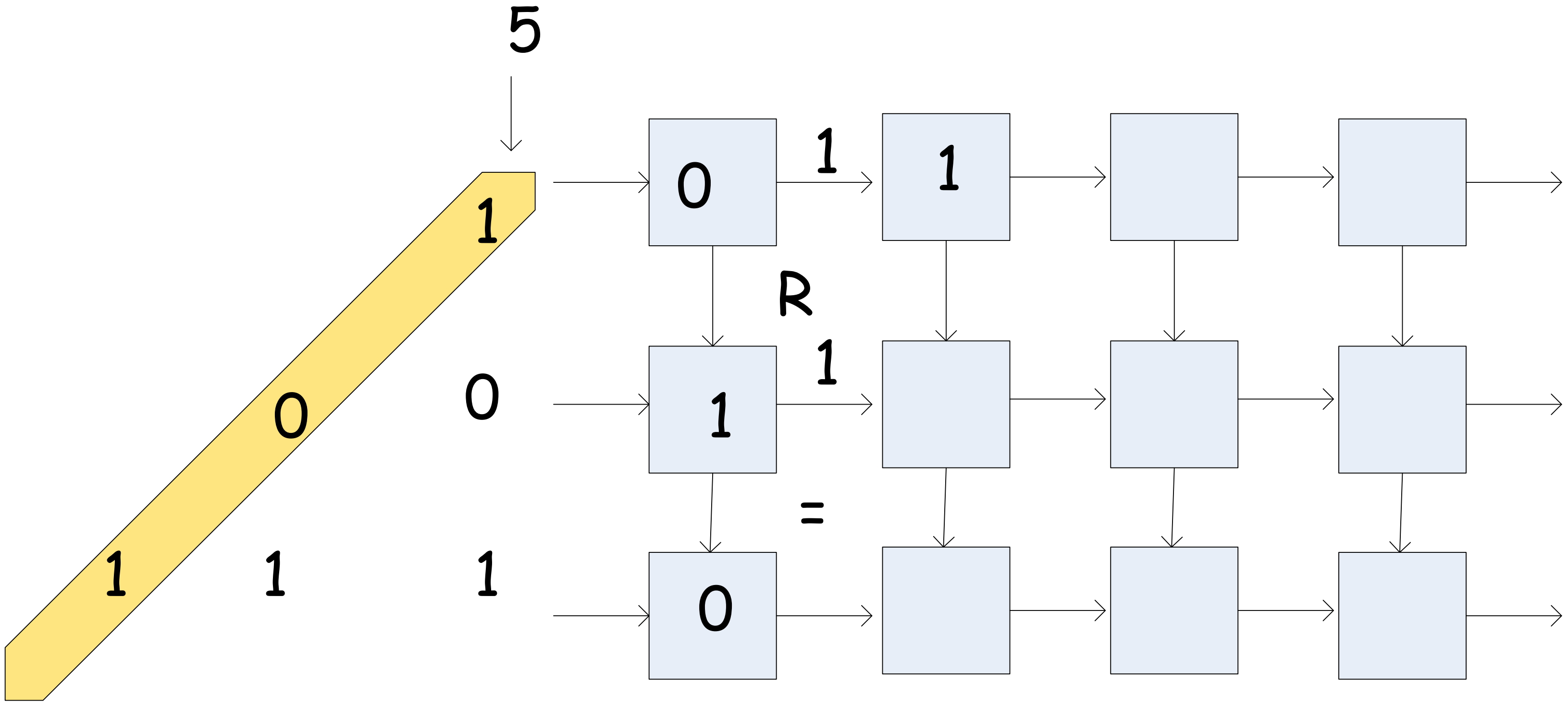
Step 1



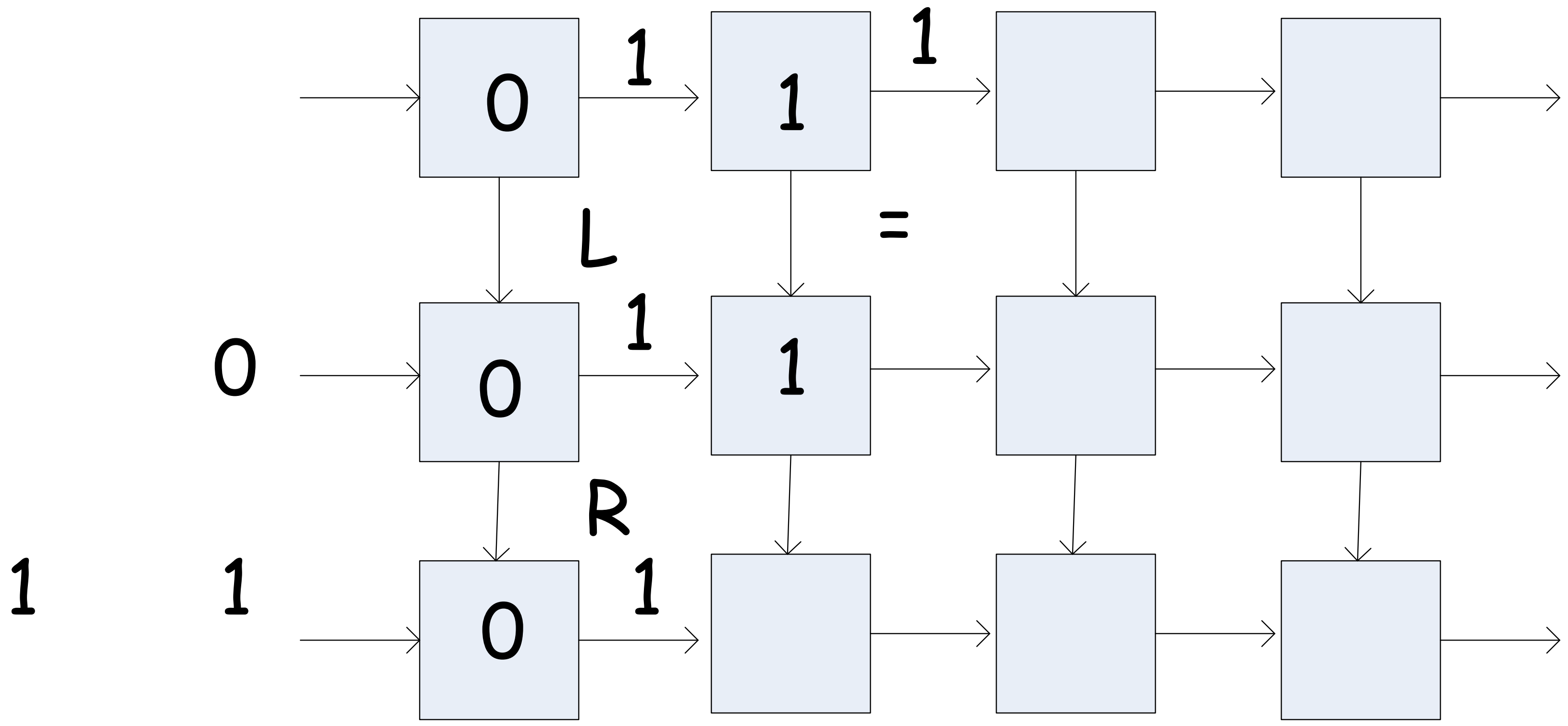
Step 2



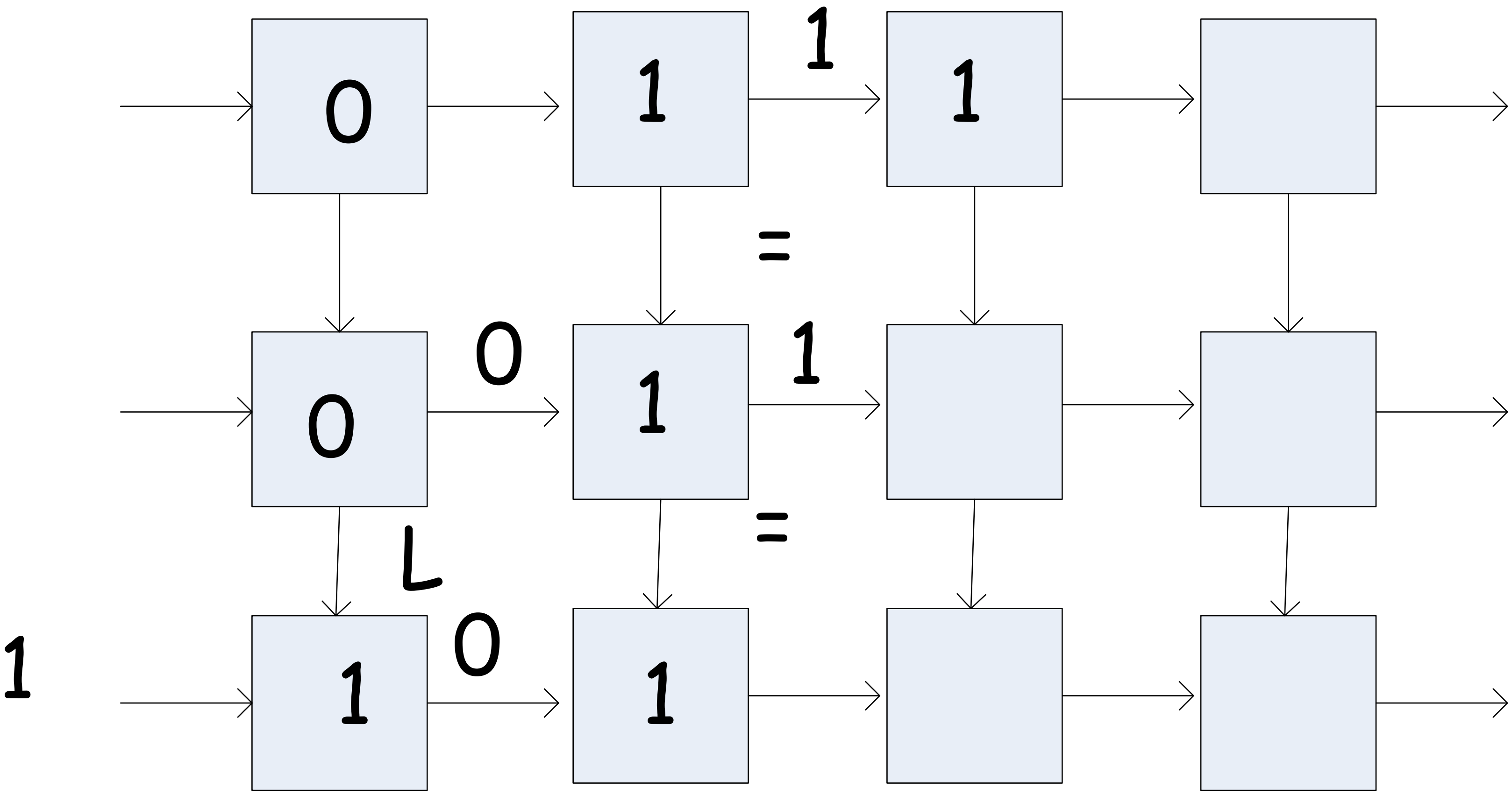
Step 3



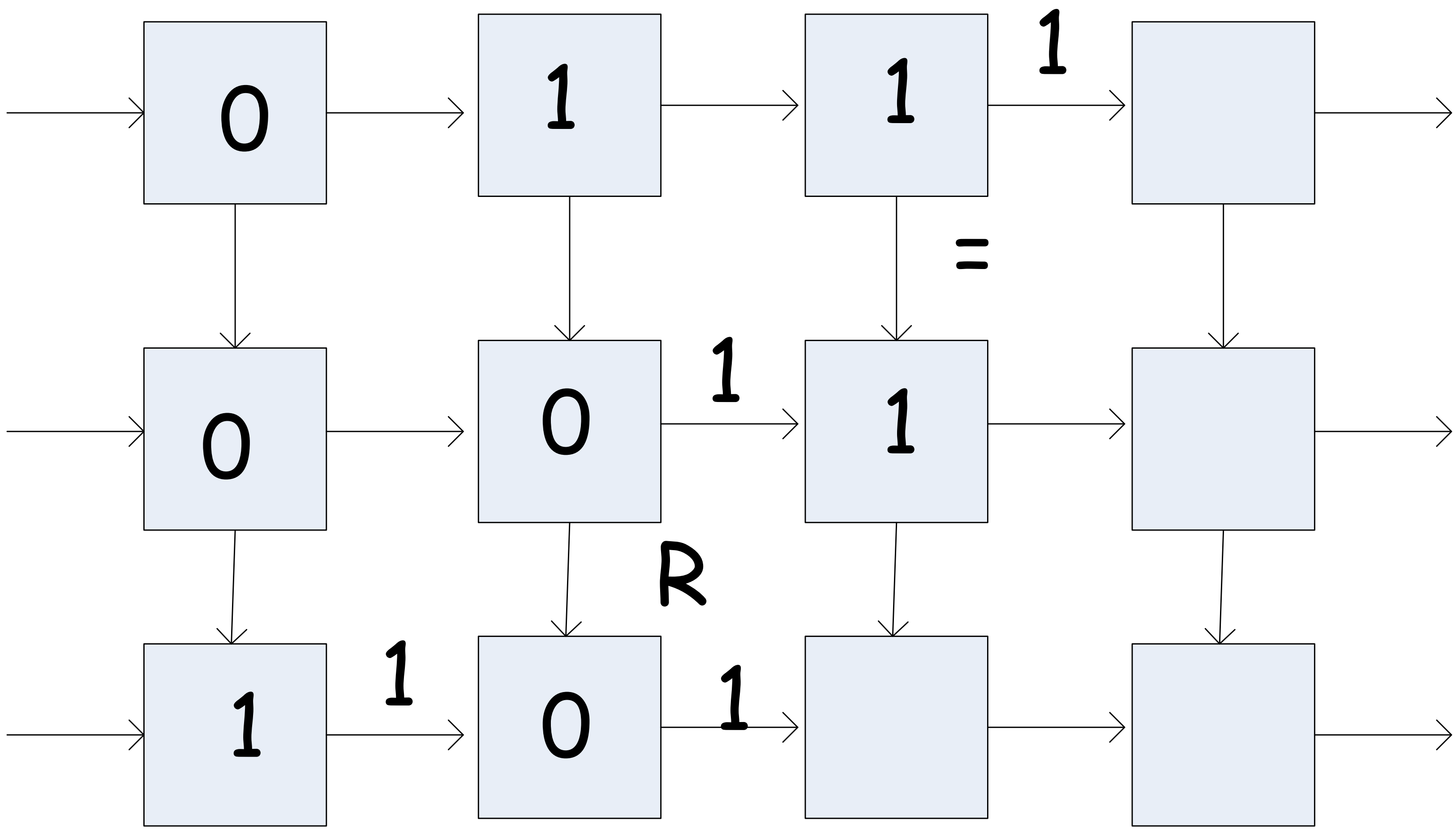
Step 4



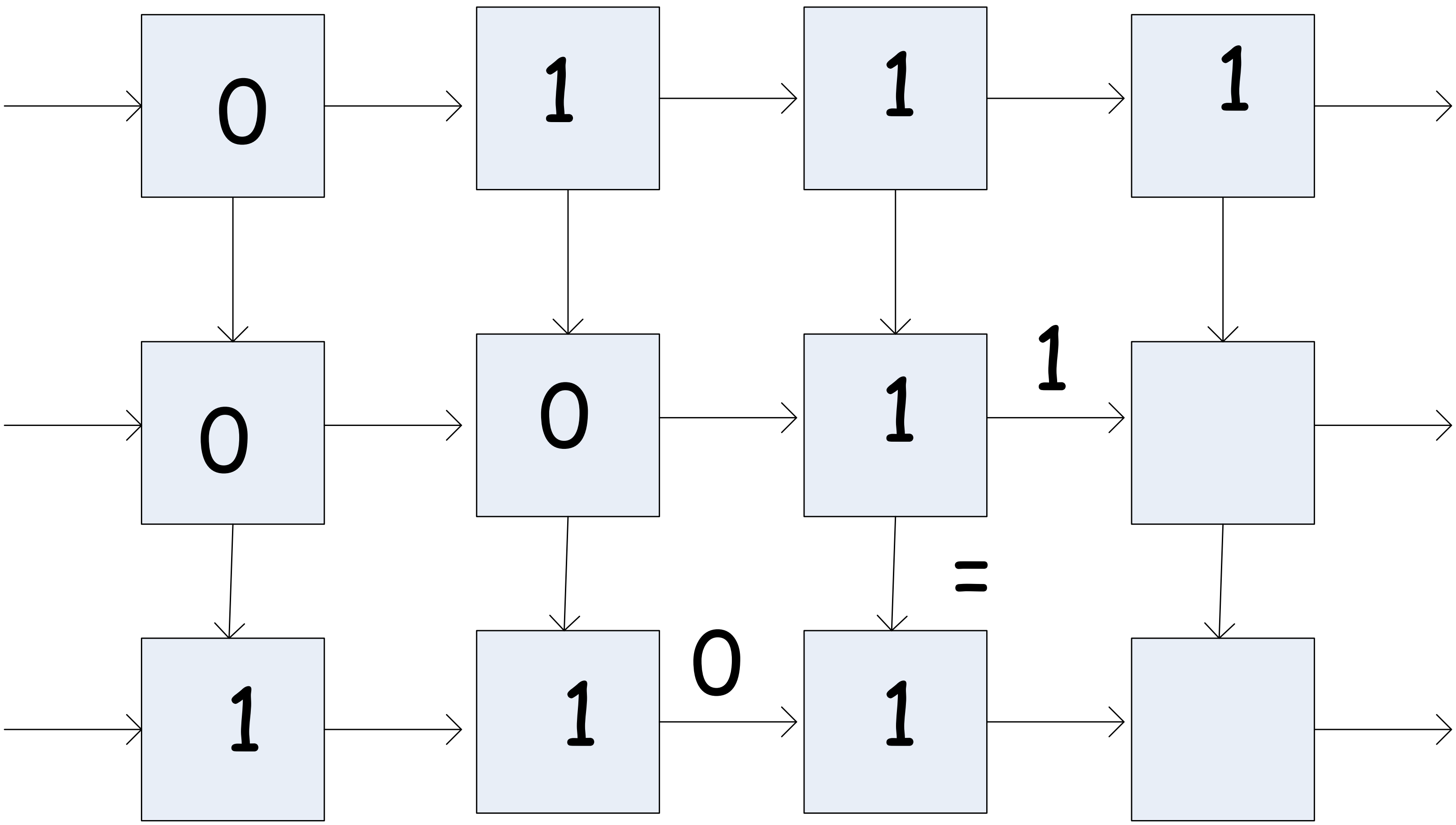
Step 5



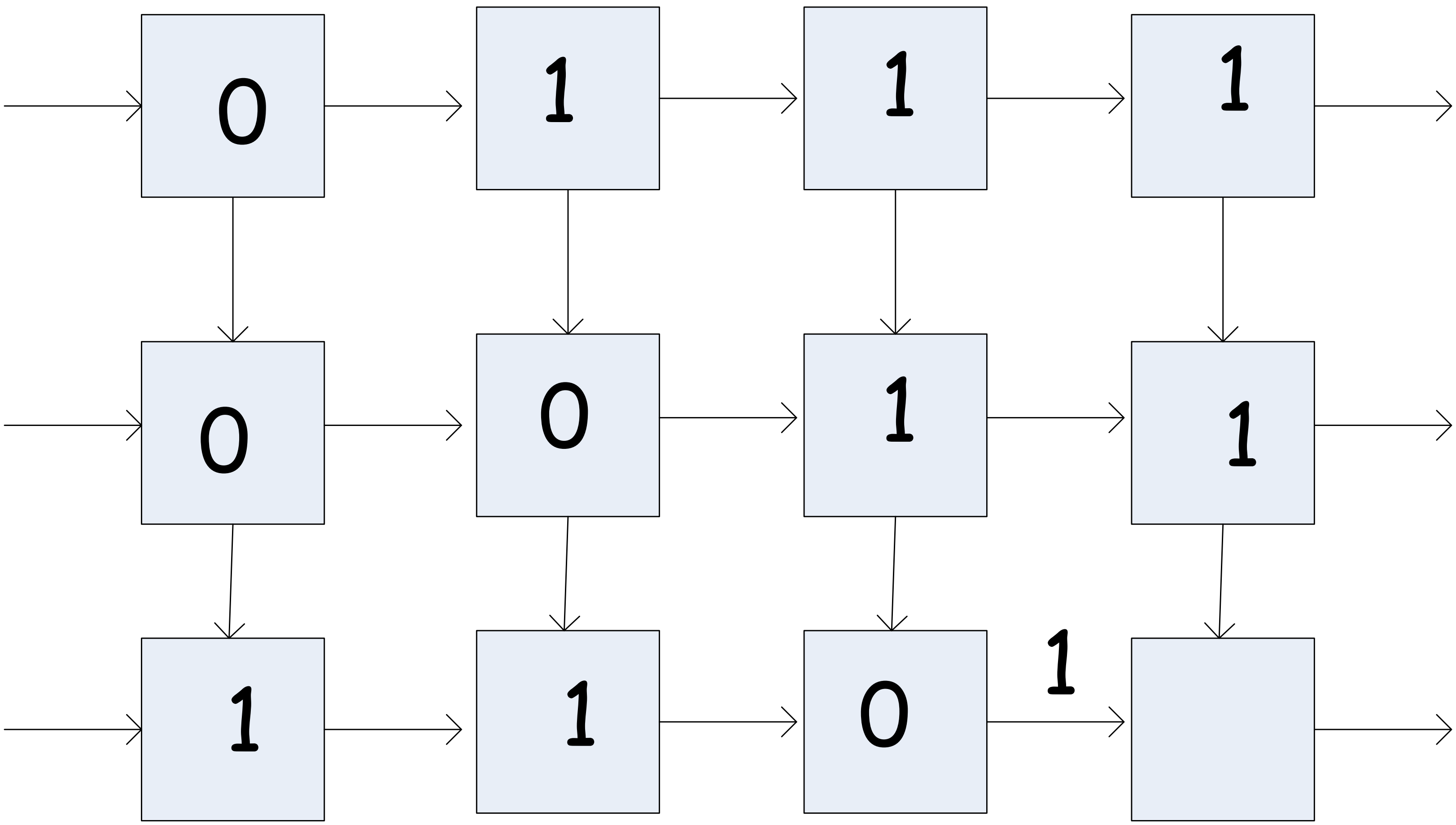
Step 6



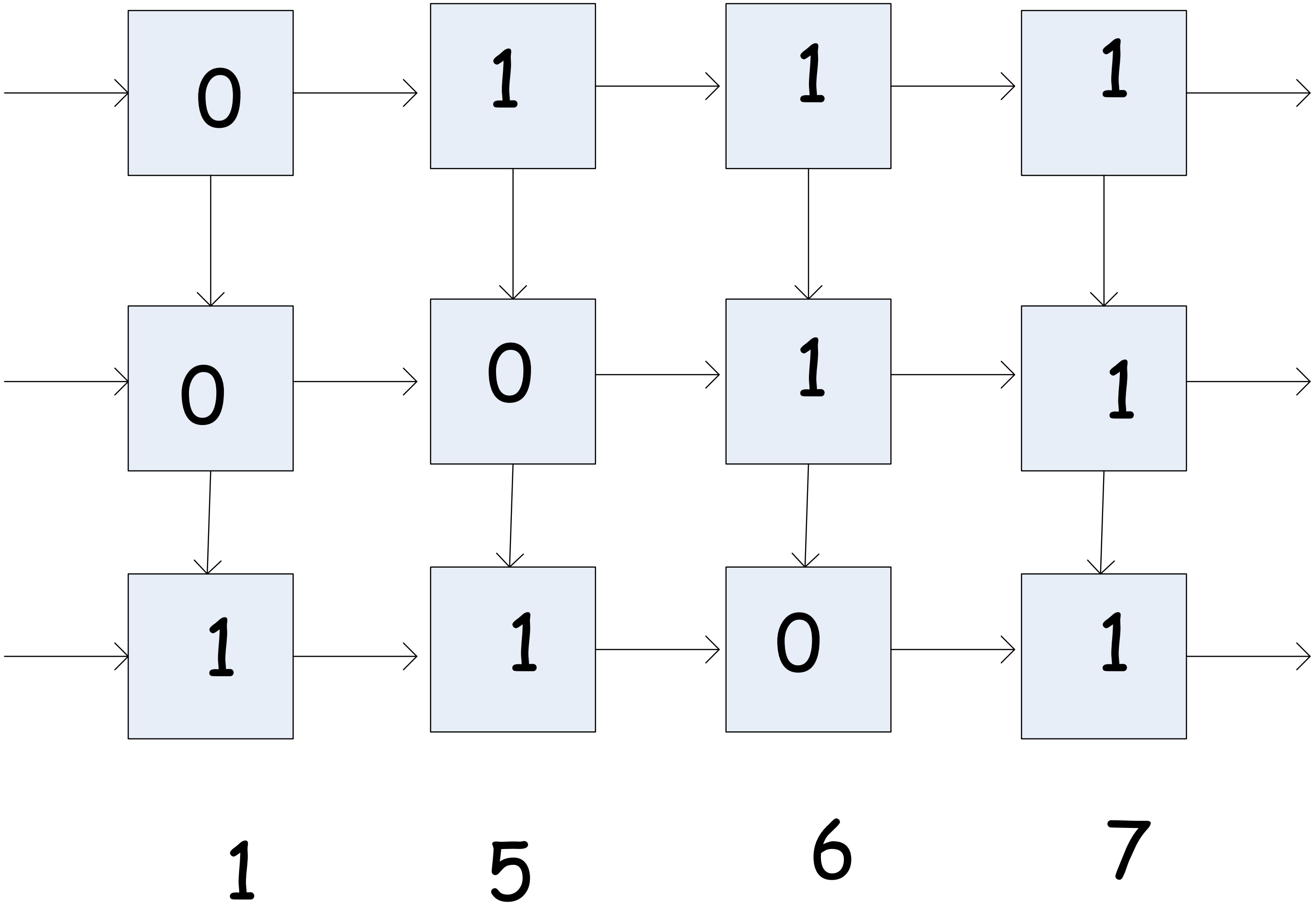
Step 7



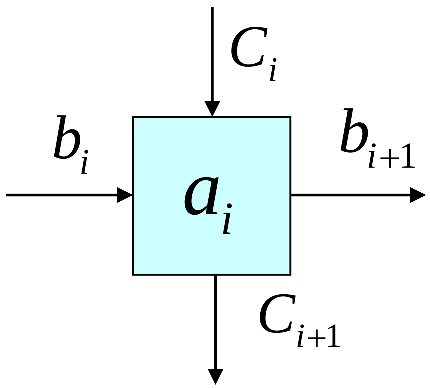
Step 8



Step 9



Systolic Program




```
if (c_i == NULL || c_i == "=")
    if (b_i > a_i)
        b_i+1 = b_i; c_i+1 = "L";
    else if (a_i > b_i)
        b_i+1 = a_i; a_i = b_i; c_i+1 = "R";
    else
         b_i+1 = b_i; c_i+1 = "R";

else if (c_i == R )



else if (c_i == L )




else if (b_i == NULL && a_i NOT == NULL)
```

$$C_i = \{\phi, =, \mathbf{R}, \mathbf{L}\}$$

$$b_i = \{\phi, 0, \mathbf{1}\}$$

$$a_i = \{0, \mathbf{1}\}$$

Convolution

$$
\begin{array}{ccccc}
 & & b_3 & b_2 & b_1 \\
 & & a_3 & a_2 & a_1 \\
\hline
 & & a_1 b_3 & a_1 b_2 & a_1 b_1 \\
 & a_2 b_3 & a_2 b_2 & a_2 b_1 & \\
a_3 b_3 & a_3 b_2 & a_3 b_1 & & \\
\hline
p_5 & p_4 & p_3 & p_2 & p_1
\end{array}
$$

Convolution

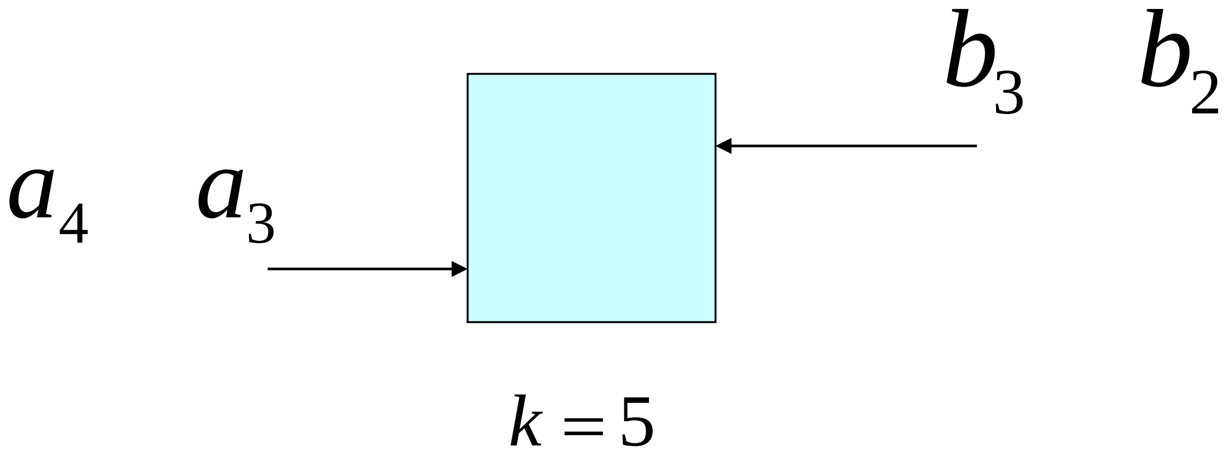
The convolution of two vectors:

$$\langle a_N, a_{N-1}, \ldots, a_1 \rangle \qquad \langle b_N, b_{N-1}, \ldots, b_1 \rangle$$

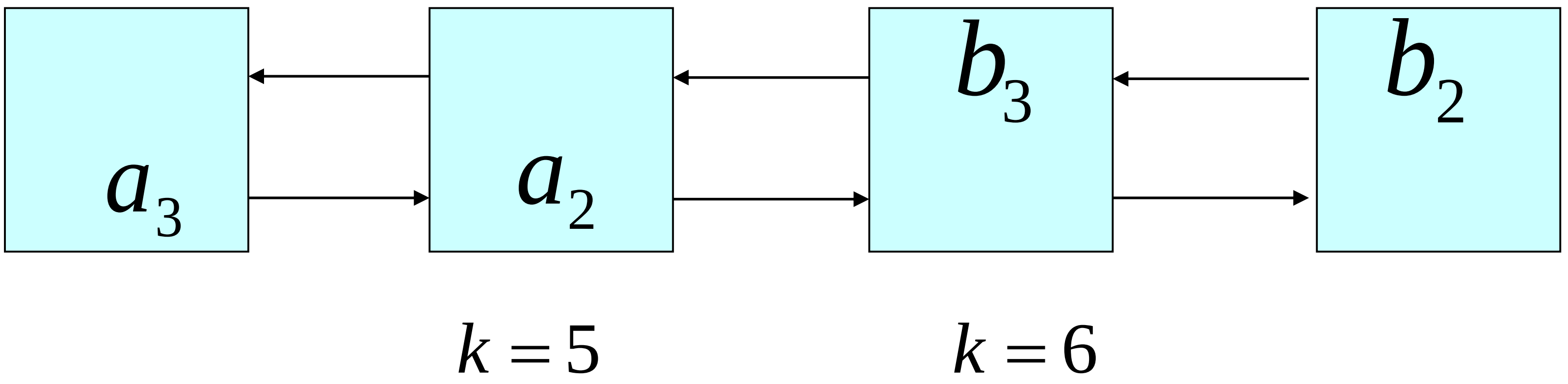
is the vector of length 2N-1, $\langle y_{2N-1}, y_{2N-2}, \ldots, y_1 \rangle$

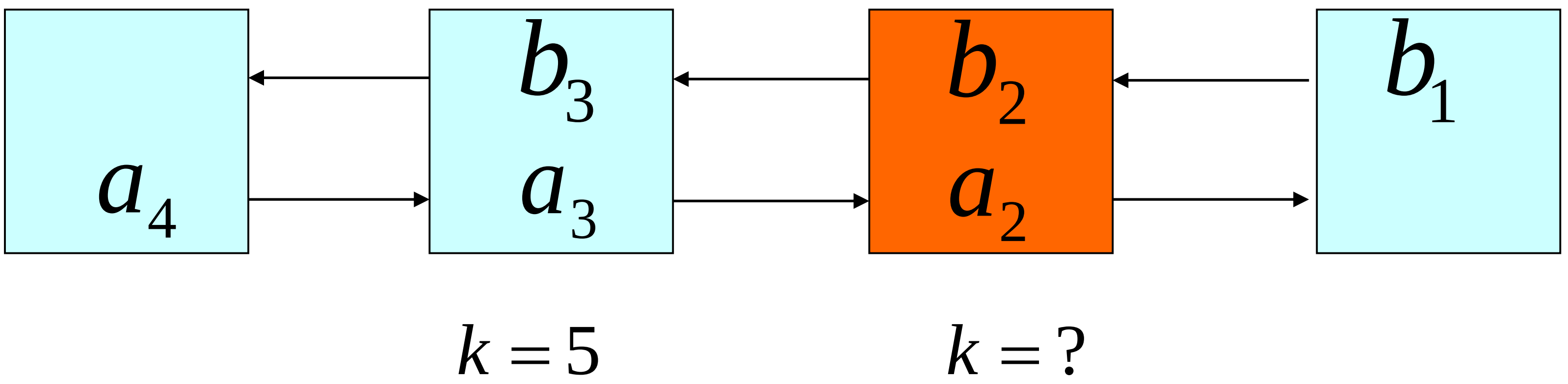
where $$y_k = \sum_{i+j=k+1} a_i b_j \qquad i \le k \le 2N-1$$

Convolution on a linear array?

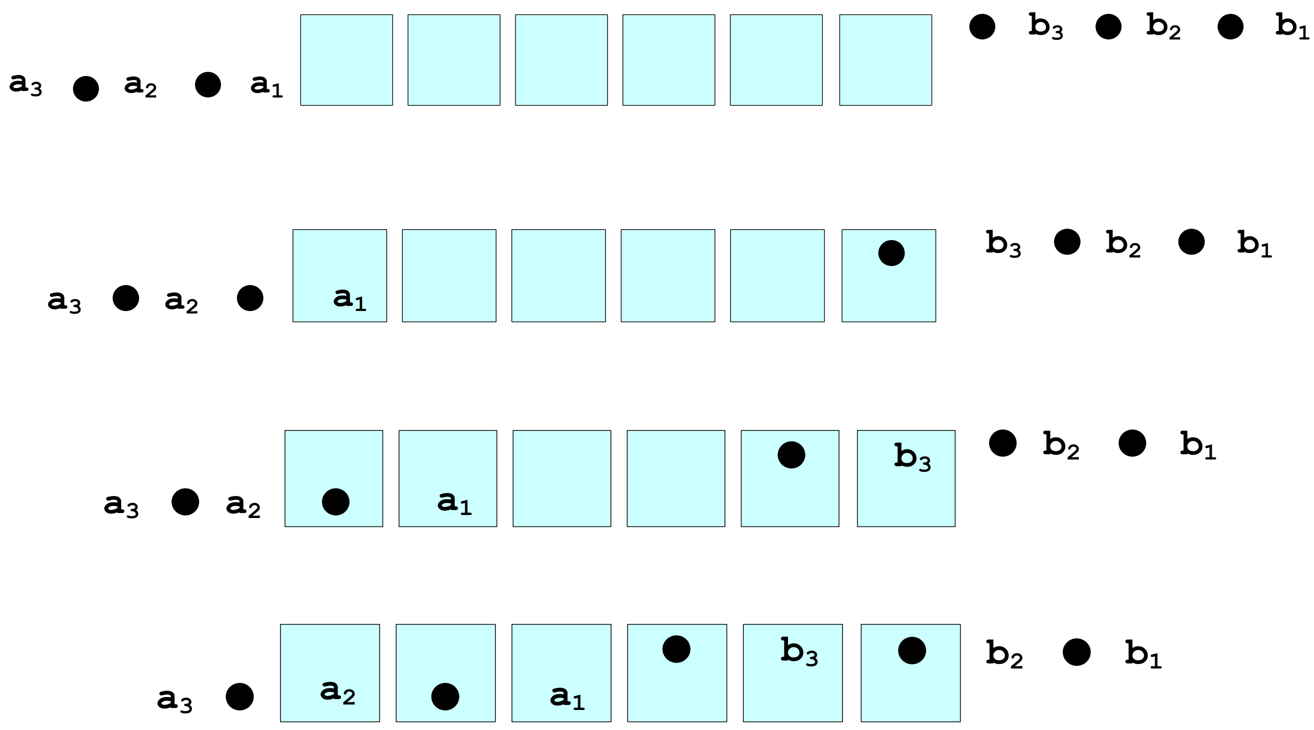


Convolution on a linear array?

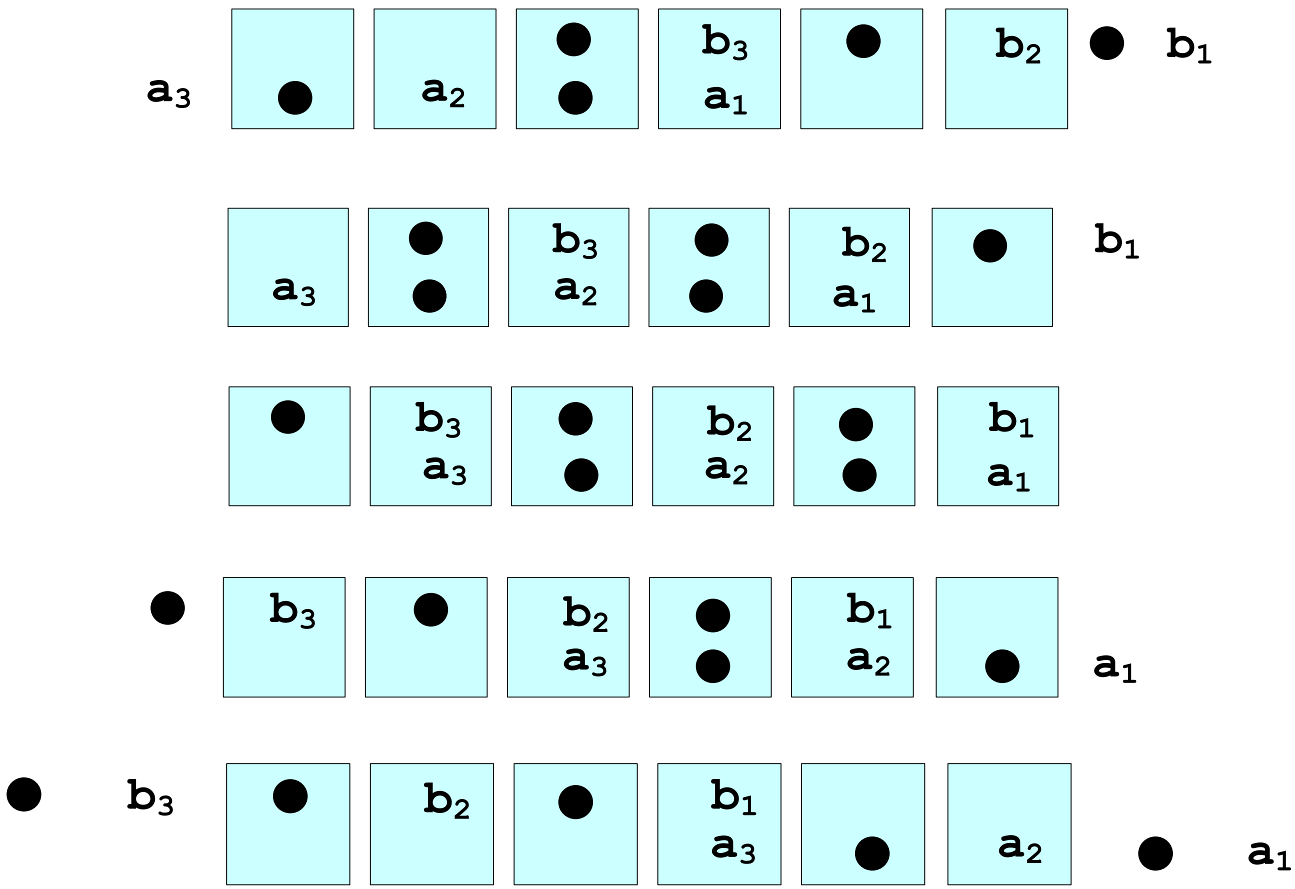




Scheduling of inputs



continued

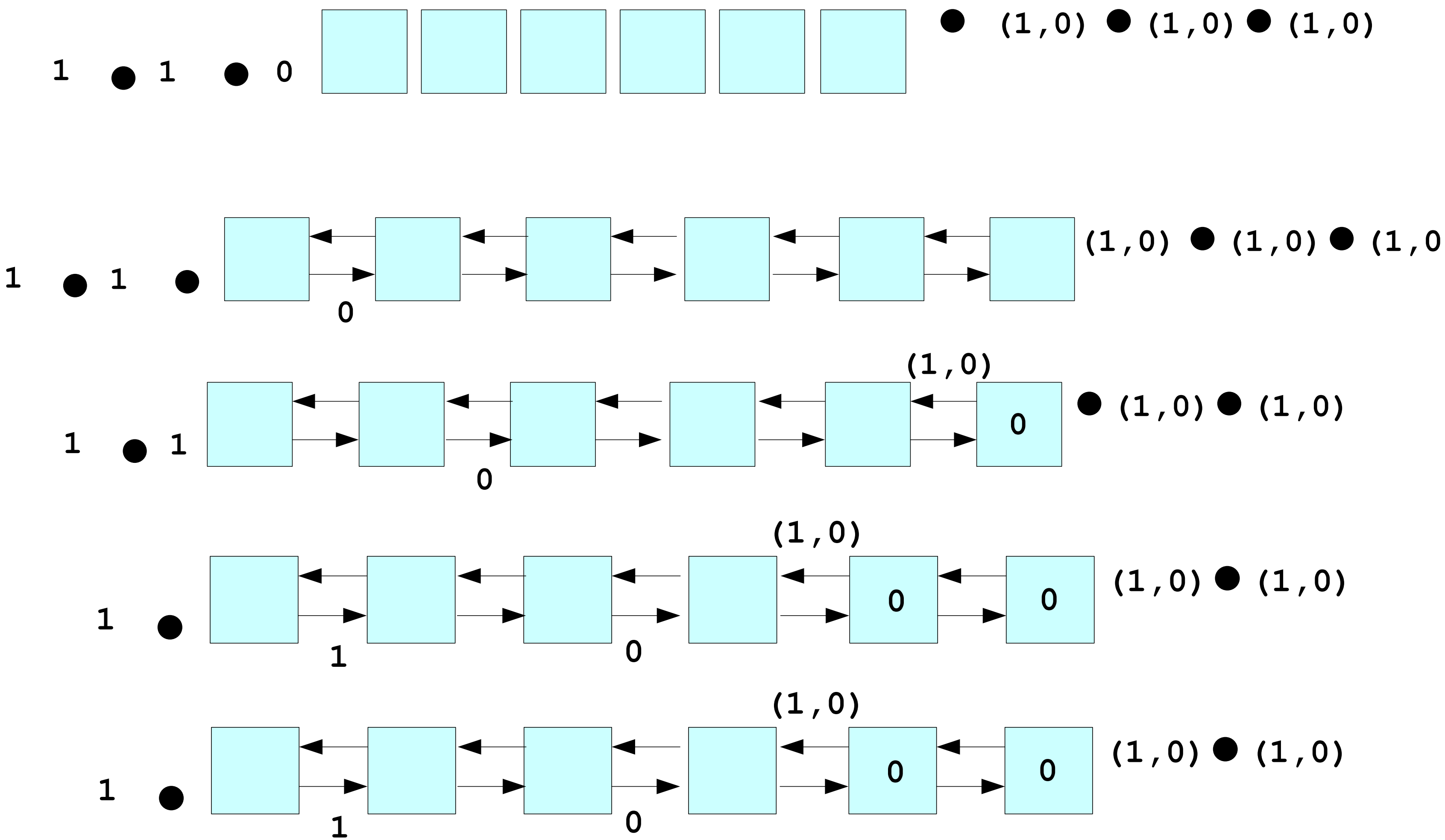


a_3 a_2 b_3 a_1 b_2 b_1
a_3 b_3 a_2 b_2 a_1 b_1
b_3 a_3 b_2 a_2 b_1 a_1
b_3 b_2 a_3 b_1 a_2 a_1
b_3 b_2 b_1 a_3 a_2 a_1

Example

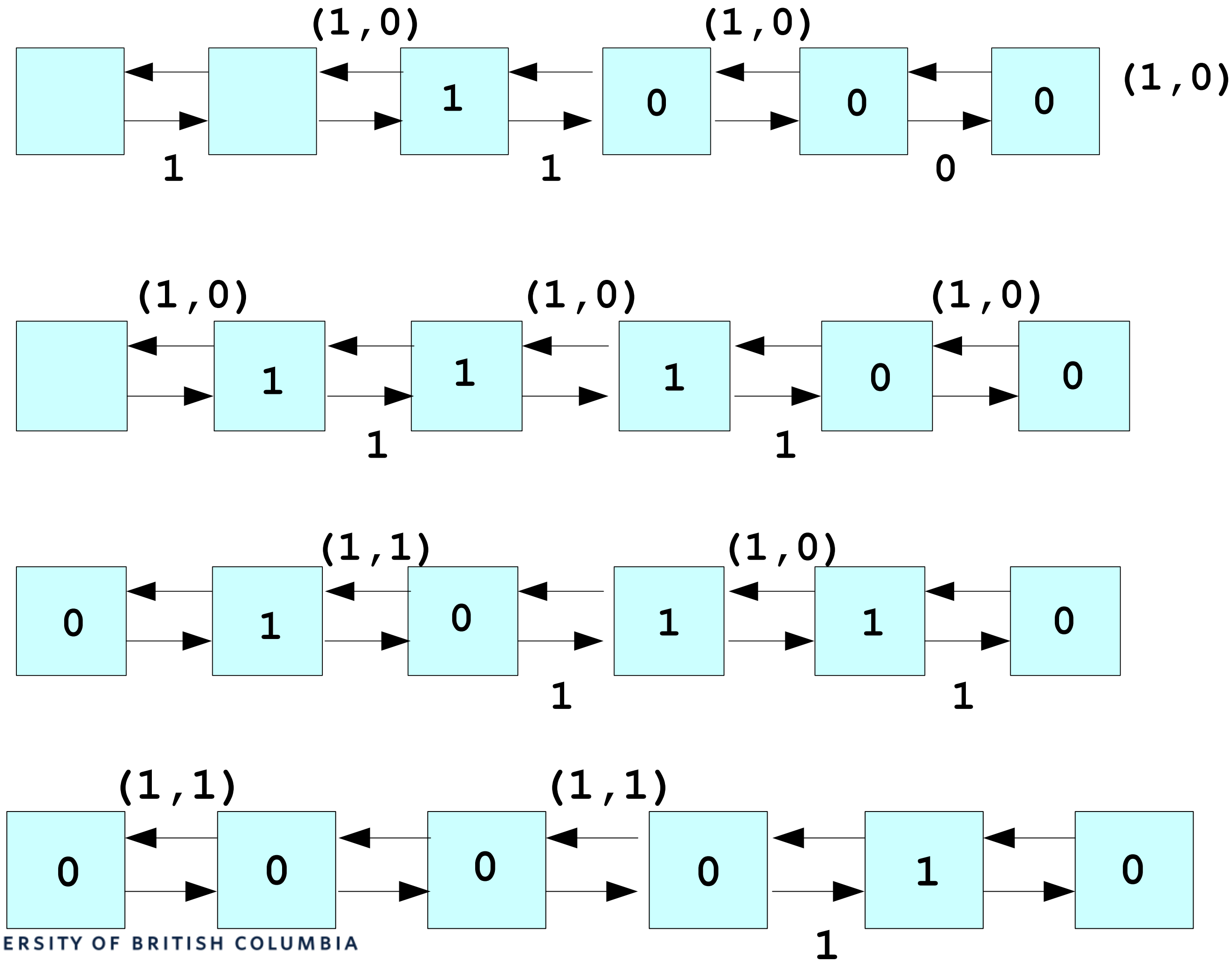
$$
\begin{array}{cccccc}
 & & & 1 & 1 & 1 \\
 & & & 1 & 1 & 0 \\
\cline{3-6}
 & & & 0 & 0 & 0 \\
 & & 1 & 1 & 1 & \\
 & 1 & 1 & 1 & & \\
\hline
1 & 0 & 1 & 0 & 1 & 0
\end{array}
$$

What about the carries?

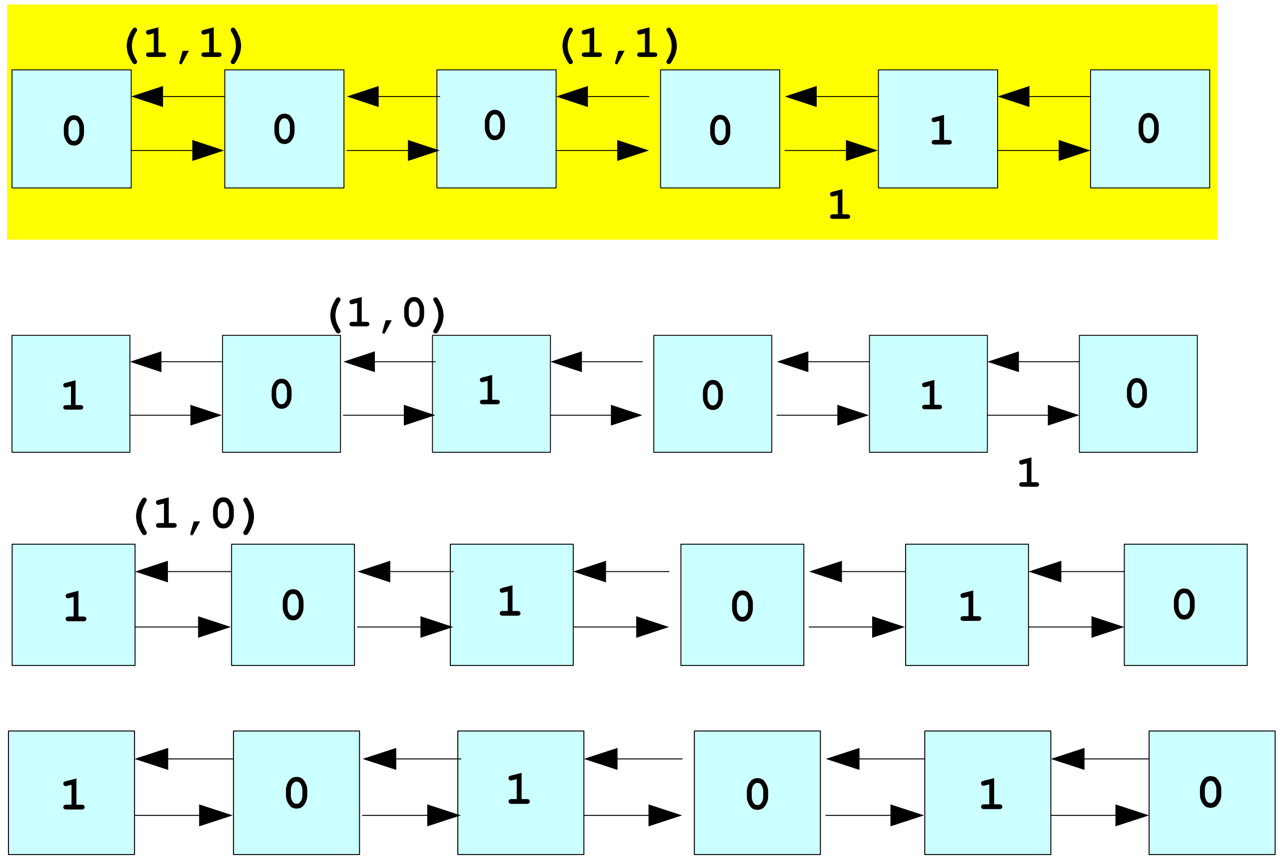


THE UNIVERSITY OF BRITISH COLUMBIA

What about the carries?



What about the carries?



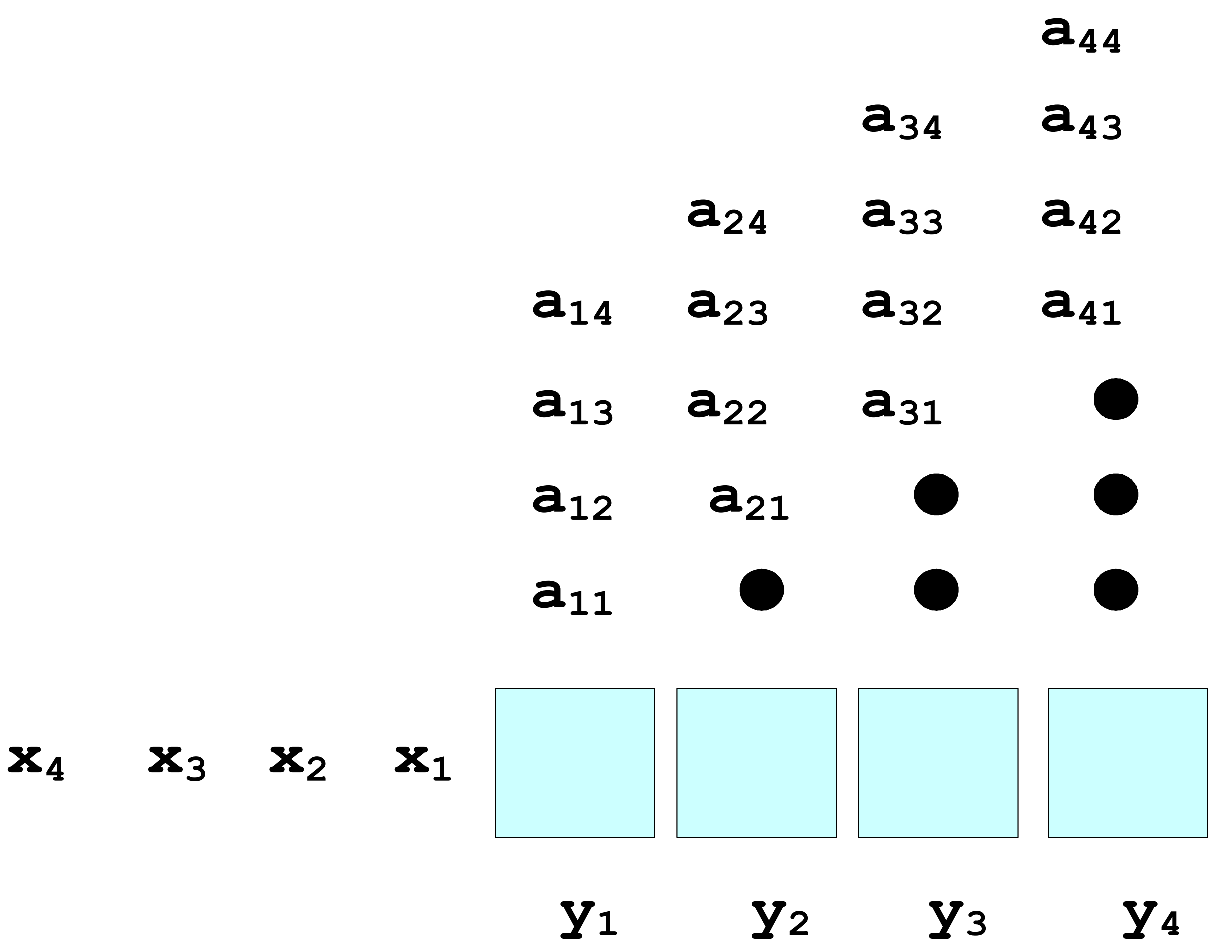
Making it more efficient?

Matrix – Vector product

$$A \bullet x = B \qquad b_i = \sum_{j=1}^{N} a_{ij} x_j$$

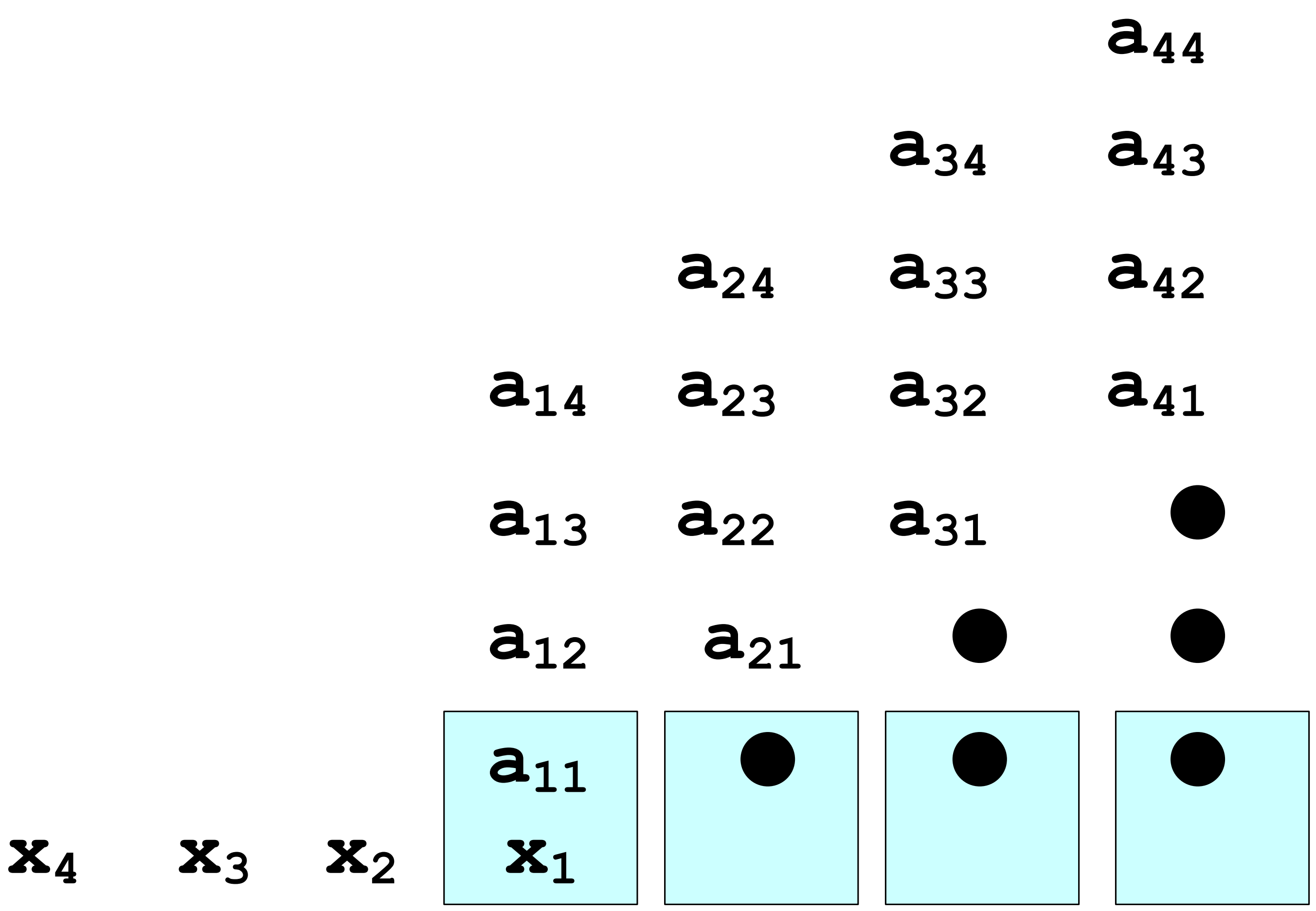
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Scheduling



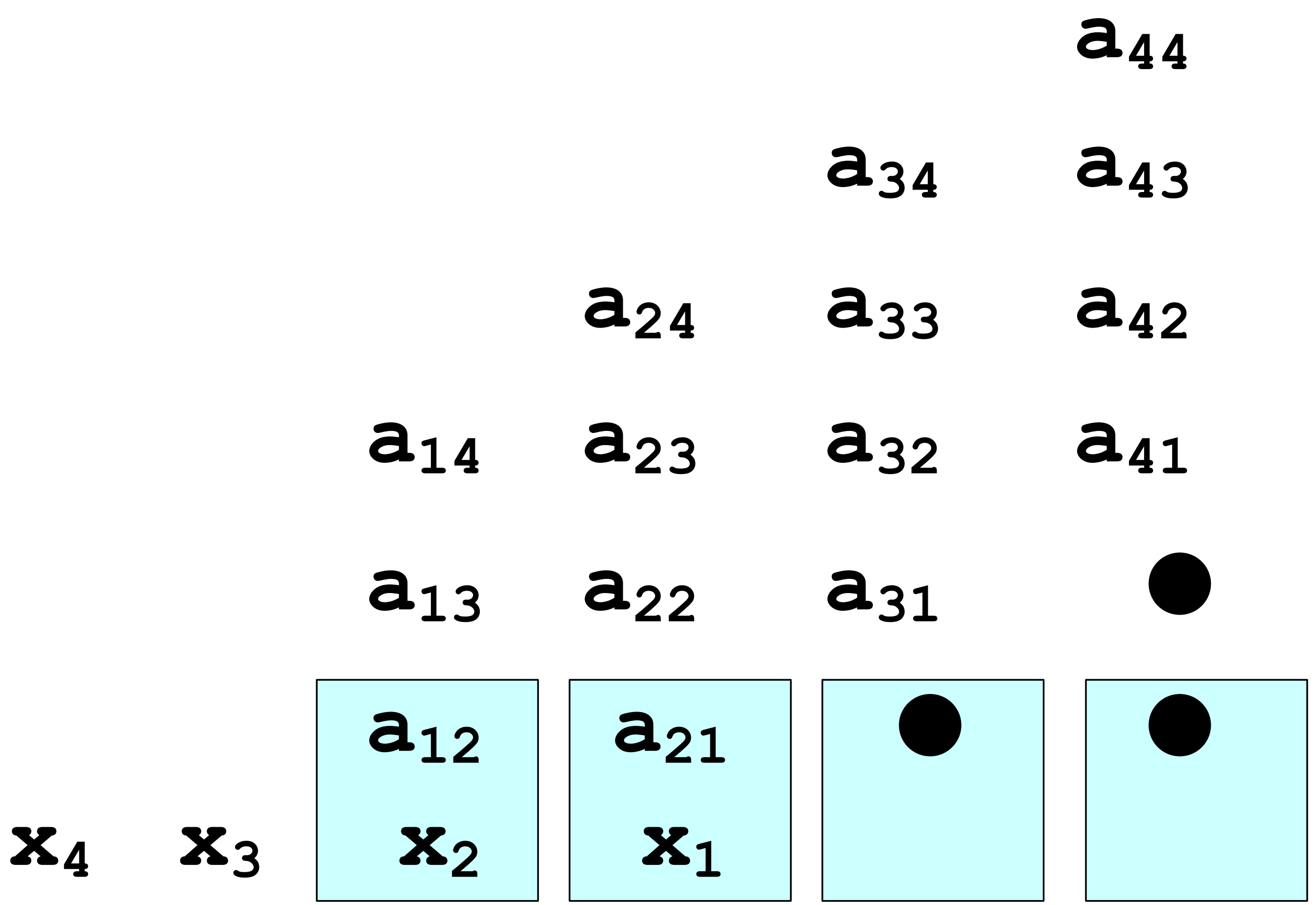
a_{44}
a_{34} a_{43}
a_{24} a_{33} a_{42}
a_{14} a_{23} a_{32} a_{41}
a_{13} a_{22} a_{31}
a_{12} a_{21}
a_{11}
x_4 x_3 x_2 x_1
y_1 y_2 y_3 y_4

Example



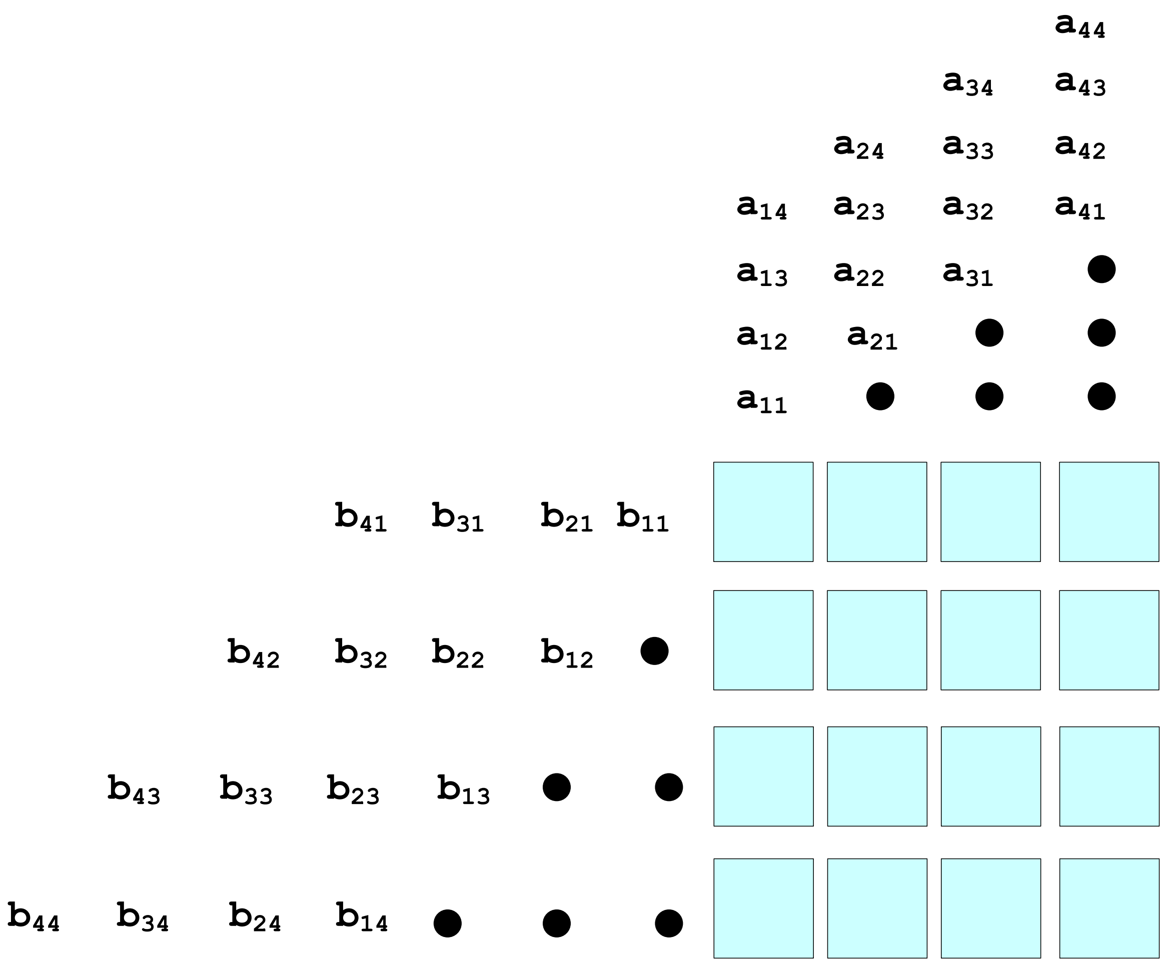
a44
a34 a43
a24 a33 a42
a14 a23 a32 a41
a13 a22 a31
a12 a21
a11
x4 x3 x2 x1

Example

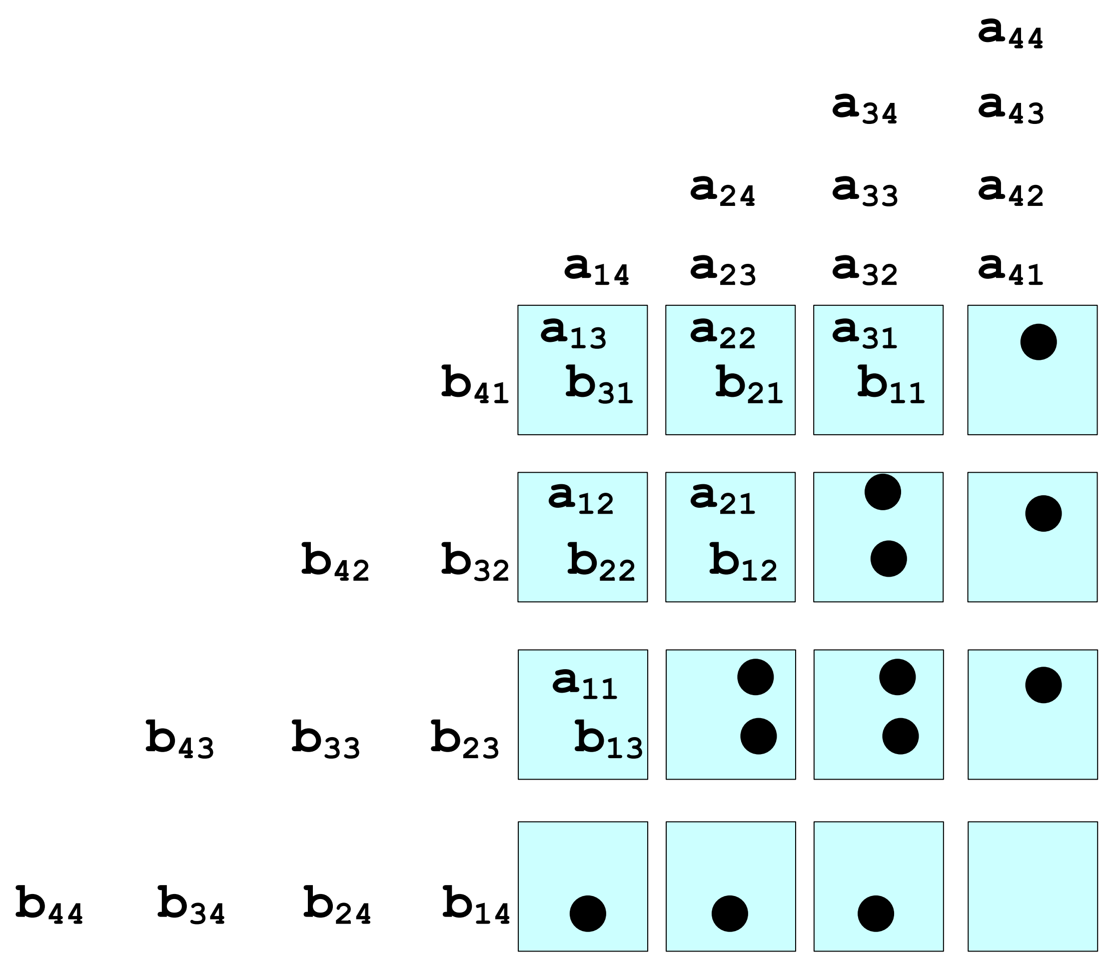


a_{44}
a_{34} a_{43}
a_{24} a_{33} a_{42}
a_{14} a_{23} a_{32} a_{41}
a_{13} a_{22} a_{31}
a_{12} a_{21}
x_4 x_3 x_2 x_1

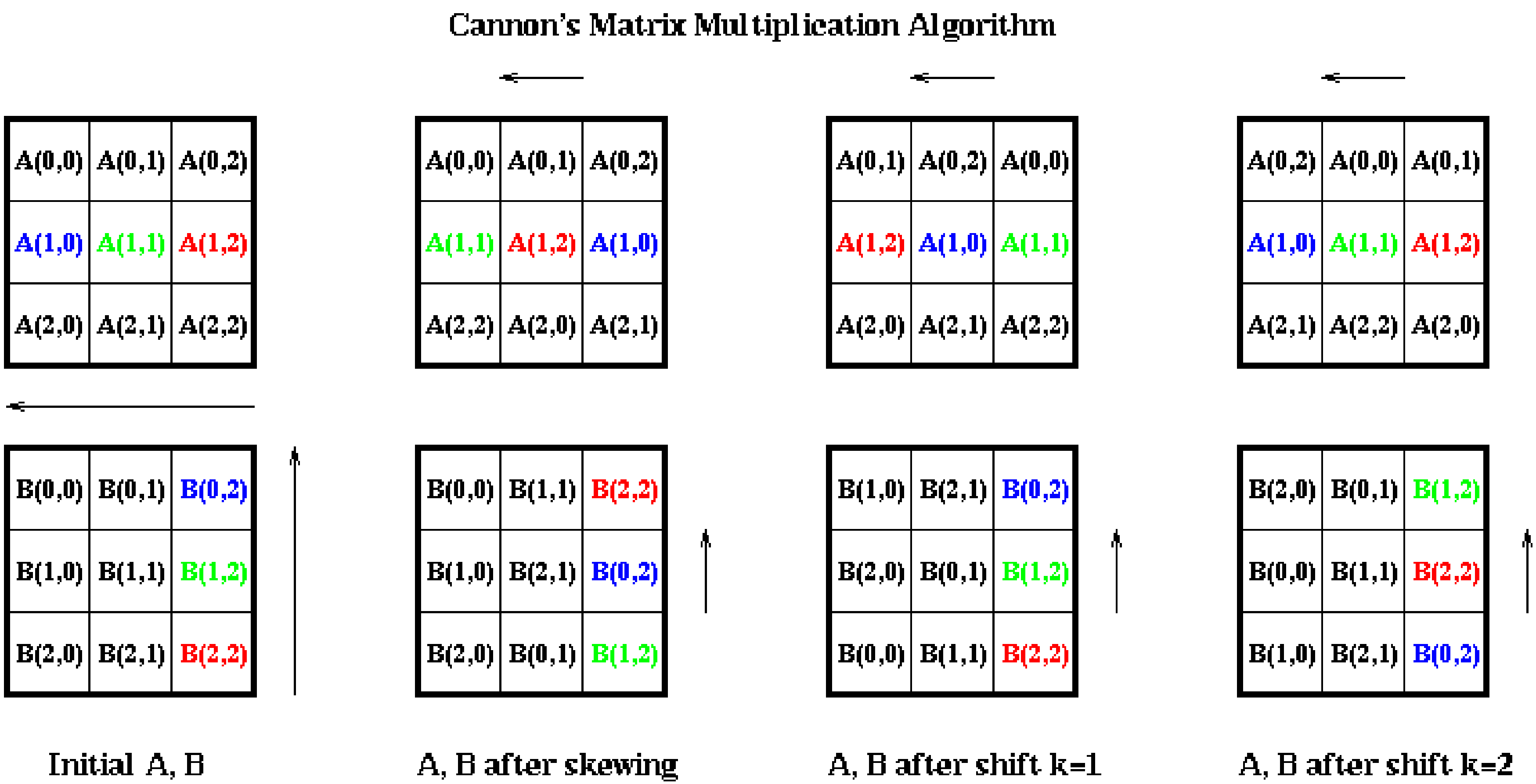
Matrix Multiplication



Matrix Multiplication



Cannon’s Matrix Multiplication



$$C(1,2) = A(1,0) * B(0,2) + A(1,1) * B(1,2) + A(1,2) * B(2,2)$$

Initial Step to Skew Matrices in Cannon

- Initial blocked input

A(0,0)	A(0,1)	A(0,2)
A(1,0)	A(1,1)	A(1,2)
A(2,0)	A(2,1)	A(2,2)

B(0,0)	B(0,1)	B(0,2)
B(1,0)	B(1,1)	B(1,2)
B(2,0)	B(2,1)	B(2,2)

- After skewing before initial block multiplies

A(0,0)	A(0,1)	A(0,2)
A(1,1)	A(1,2)	A(1,0)
A(2,2)	A(2,0)	A(2,1)

B(0,0)	B(1,1)	B(2,2)
B(1,0)	B(2,1)	B(0,2)
B(2,0)	B(0,1)	B(1,2)

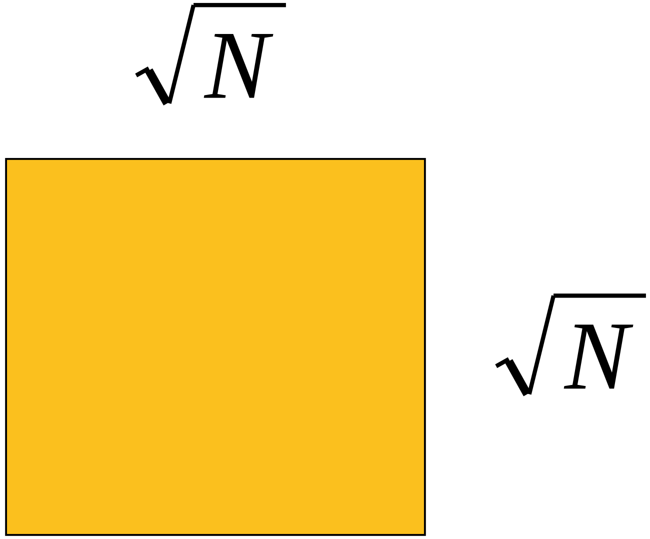
Block Matrix Operations

$$\begin{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} & \begin{bmatrix} a_{13} & a_{14} \\ a_{23} & a_{24} \end{bmatrix} \\ \begin{bmatrix} a_{31} & a_{32} \\ a_{41} & a_{42} \end{bmatrix} & \begin{bmatrix} a_{33} & a_{34} \\ a_{43} & a_{44} \end{bmatrix} \end{bmatrix}$$

$$C_{11} = \overbrace{A_{11}B_{11}}^{\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}} + \underbrace{A_{12}B_{21}}_{\begin{bmatrix} a_{13} & a_{14} \\ a_{23} & a_{24} \end{bmatrix}\begin{bmatrix} b_{31} & b_{32} \\ b_{41} & b_{42} \end{bmatrix}}$$

Block Operations

- Scale up the computation to communication



Triangular Matrix Solve

- Upper Triangular Matrix where *a's* and *b's* are constants and *x*'s are unknowns to be found

$$
\begin{array}{llll}
a_{n-1,0}x_0 + a_{n-1,1}x_1 + a_{n-1,2}x_2 & \ldots & + a_{n-1,n-1}x_{n-1} & = b_{n-1} \\
 & & . & \\
 & & . & \\
a_{2,0}x_0 + a_{2,1}x_1 + a_{2,2}x_2 & & & = b_2 \\
a_{1,0}x_0 + a_{1,1}x_1 & & & = b_1 \\
a_{0,0}x_0 & & & = b_0
\end{array}
$$

Back Substitution

First, unknown x_0 is found from last equation; i.e.,

$$x_0 = \frac{b_0}{a_{0,0}}$$

Value obtained for x_0 substituted into next equation to obtain x_1; i.e.,

$$x_1 = \frac{b_1 - a_{1,0}x_0}{a_{1,1}}$$

Values obtained for x_1 and x_0 substituted into next equation to obtain x_2:

$$x_2 = \frac{b_2 - a_{2,0}x_0 - a_{2,1}x_1}{a_{2,2}}$$

and so on until all the unknowns are found.

Re-write the equations

for lower triangular matrices

$$t_1 = b_1 \qquad t_i = b_i - \sum_{j=1}^{i-1} a_{ij} x_j$$

so that $\quad t_i = a_{ii} x_i$

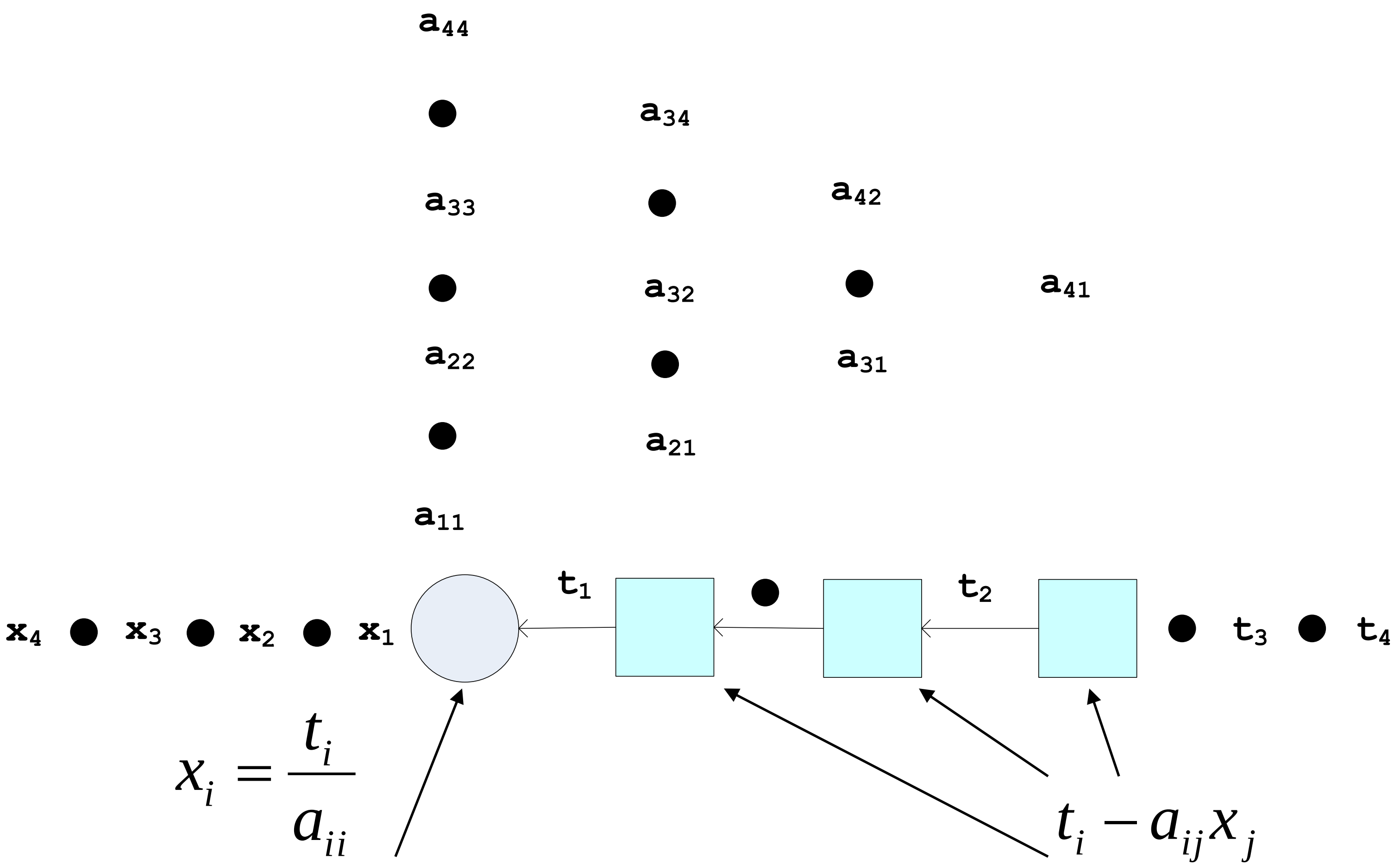
$$\begin{bmatrix} a_{11} & & \\ a_{21} & a_{22} & \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$t_1 = b_1$$

$$t_2 = b_2 - a_{21} x_1$$

$$t_3 = b_2 - \left(a_{31} x_1 + a_{32} x_2\right)$$

Linear Array



Soft Systolic

- Spatial locality
  - Locally connected, finite processing elements, each with a small amount of memory
- Temporal locality
  - Operates synchronously, internally acting as a small FSA
- Regular
  - Small regular collection of identical processing elements called cells
- Pipelinability
  - N cells should achieve order N speed-up
- I/O closeness
  - No inside cells access the outside
- Modularity
  - Can extend to larger designs