Systolic and Pipelined Processors

Sources:

台灣大學電機系

吳安宇

Andre Hon, Jason Handuber, Michelle Gunning, Berman, J. Kim, Heiko Schroeder, Hai Tao, Shaaban

Systolic Processors Discussed on Monday, May 26

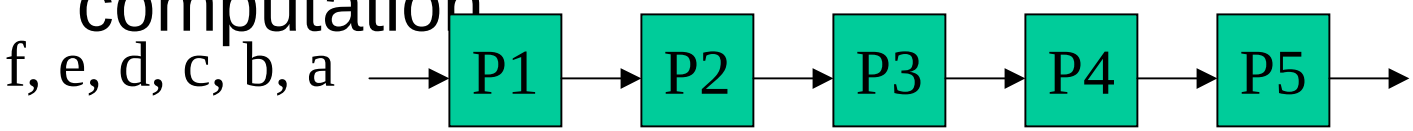
- 1. Architecture for Matrix Matrix multiplication
- 2. Architecture for Matrix Vector Multiplication
- 3. Convolution, one-dimensional and polynomial multiplication
- 4. Pipelined Systolic matcher for many gene patterns in a long chromosome. Count 100%, 75% and 50 % matches.
- 5. Systolic processor that solves Petrick Function, SAT, and SAT with minimal literal cost problems.
  - (not shown here, if you are interested, ask Newton for the solution).

Methodology of designing systolic processors.

- 1. Write full equation for results.
- 2. Skew the input flows or put inputs to processors.
- 3. The inputs may come from two different directions, these directions may be opposite
- 4. Sometimes the processors must be separated by other types of processors or just delays (do-nothing registers clocked by general clock). However, sometimes skewing input data is sufficient for this.
- 5. Draw the structure of processors.
- 6. Design data path of each processor.
- 7.Verify the correct operation. Draw snapshots, use color pencils.
- 8. Design the controller of each processor or a SIMD global controller. Remember about synchronization. Each processor must complete its work in the exactly same moment (clock pulse). It is somehow similar to Sieve of Eratostenes but has interesting realization of logic operators and use of random number generator..

Pipelined Computations

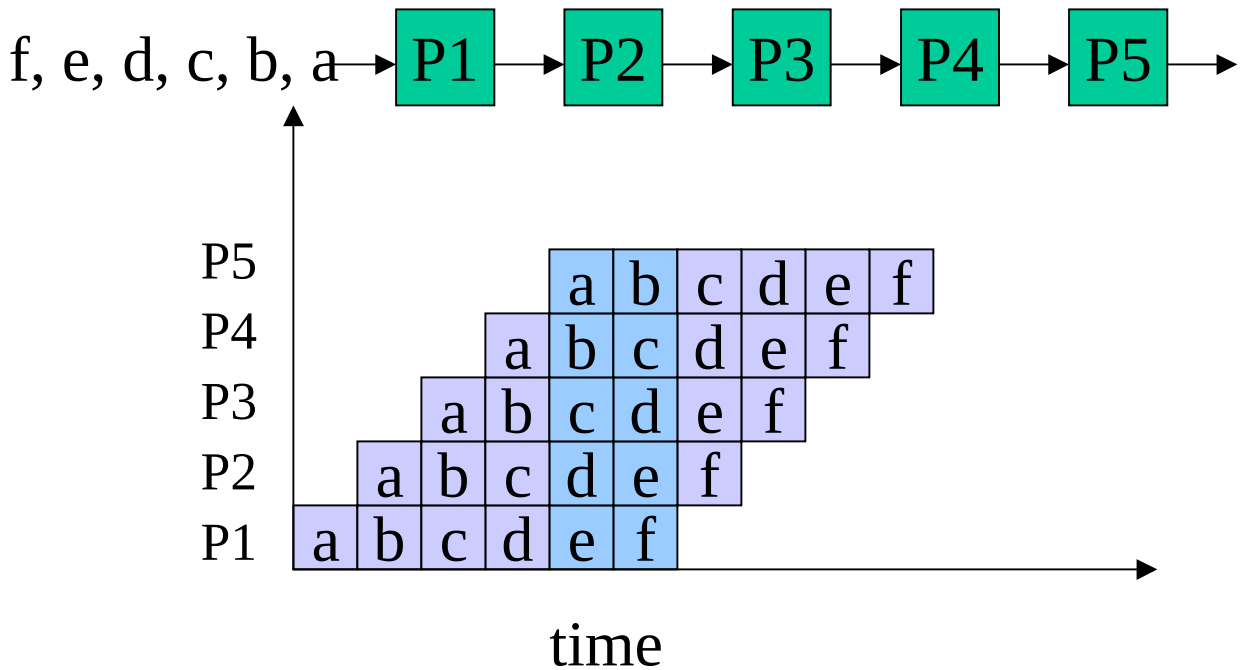
- Pipelined program divided into a series of tasks that have to be completed one after the other.
- Each task executed by a separate pipeline stage
- Data streamed from stage to stage to form computation



Pipelined Computations

- Computation consists of data streaming through pipeline stages
- Execution Time = Time to fill pipeline (P-1)
  + Time to run in steady state (N-P+1)
  + Time to empty pipeline (P-1)

P = # of processors
N = # of data items
(assume P < N)



Pipelined Example: Sieve of Eratosthenes

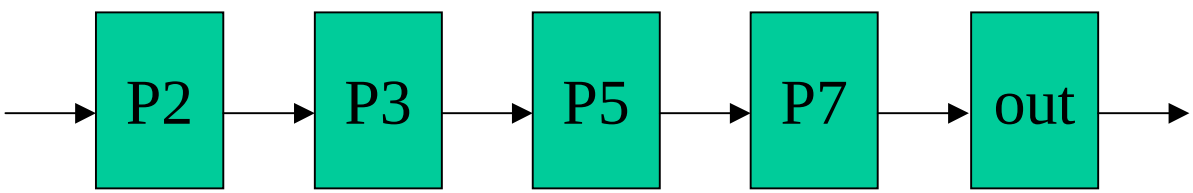
- Goal is to take a list of integers greater than 1 and produce a list of primes
  - E.g. For input 2 3 4 5 6 7 8 9 10, output is 2 3 5 7

- Fran's pipelined approach (a little different than the book):
  - Processor P_i divides each input by the ith prime
  - If the input is divisible (and not equal to the divisor), it is marked (with a negative sign) and forwarded
  - If the input is not divisible, it is forwarded
  - Last processor only forwards unmarked (positive) data [primes]

Sieve of Eratosthenes Pseudo-Code

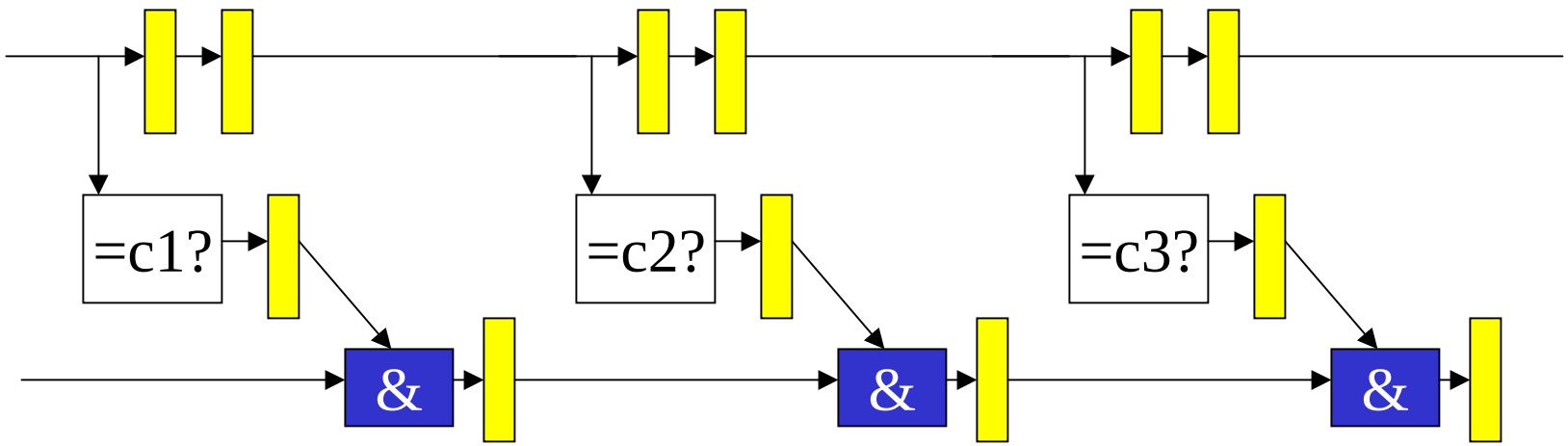
- Code for processor Pi (and prime p_i):
  - x=recv(data,P_(i-1))
  - If (x>0) then
    - If (p_i divides x and p_i = x) then send(-x,P_(i+1)
    - If (p_i does not divide x or p_i = x) then send(x, P_(i+1))
  - Else
    - Send(x,P_(i+1))

- Code for last processor
  - x=recv(data,P_(i-1))
  - If x>0 then send(x,OUTPUT)



Simple String Matching

- Stream text past set of matches



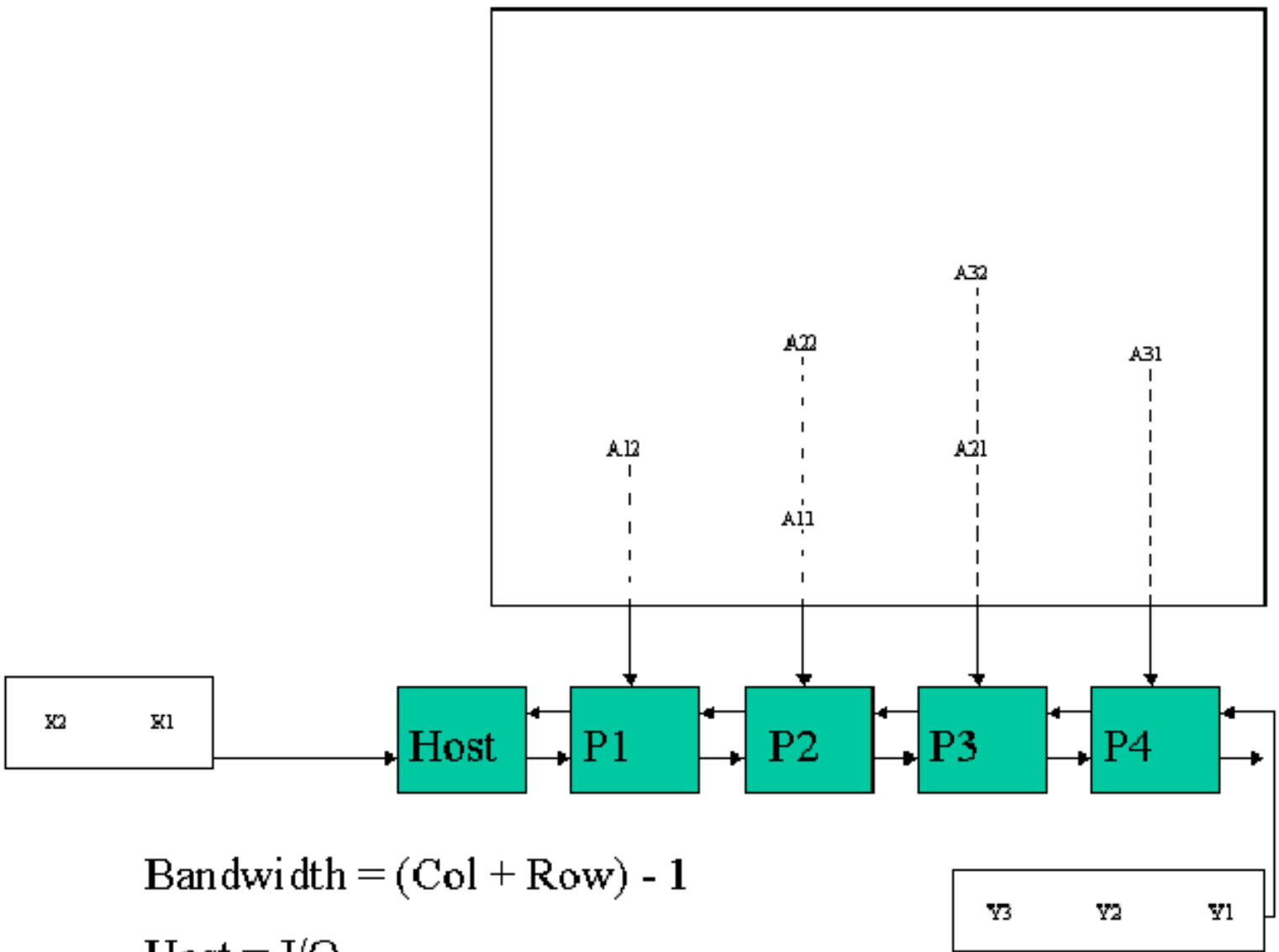
Matrix Vector Multiplication

- Consider multiplying a 3x2 X 2x1 matrix:

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

$$y_1 = (a_{11} * x_1) + (a_{12} * x_2)$$
$$y_2 = (a_{21} * x_1) + (a_{22} * x_2)$$
$$y_3 = (a_{31} * x_1) + (a_{32} * x_2)$$

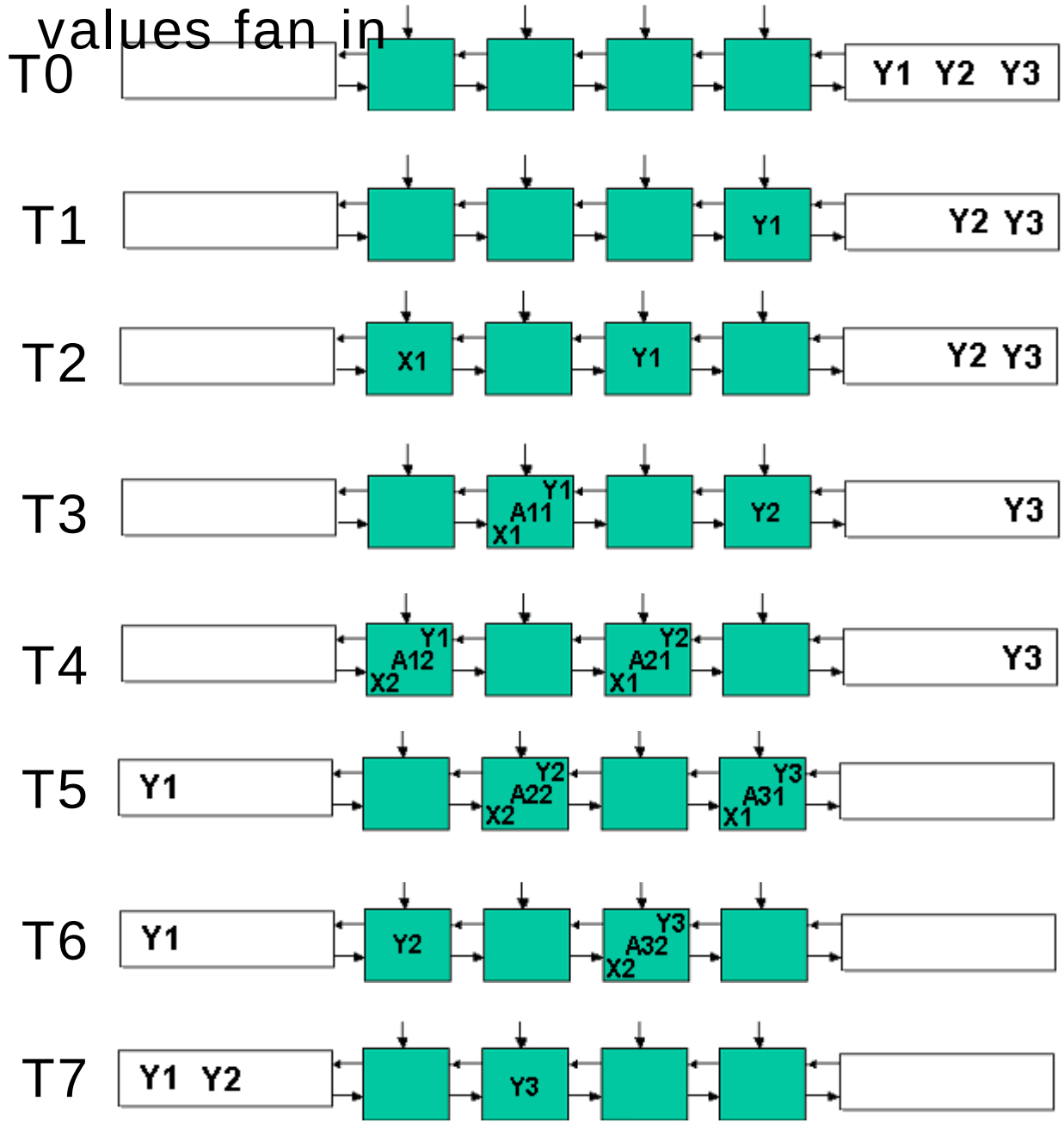
Systolic Arrays



Bandwidth = (Col + Row) - 1

Host = I/O

Y values goes left, X values go right, A values fan in

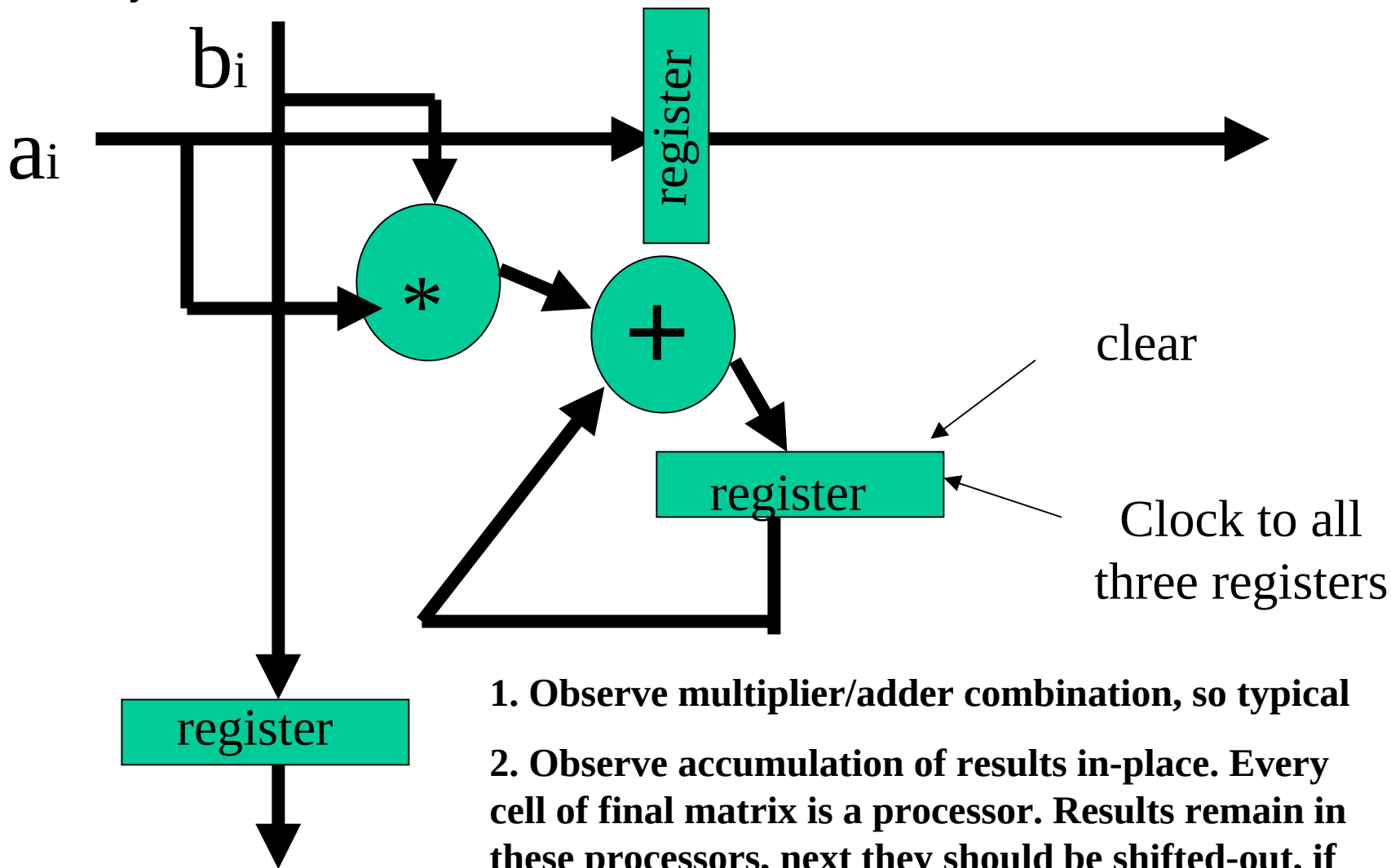


The processor

- 1. The processor was designed on Monday by students.
- 2. Ask for their notes or design your own.
- 3. You need to remember data path design and register transfer design, as well as FSM synthesis, in general.

Systolic Array Example:
3x3 Systolic Array Matrix Multiplication

- Every cell is like this:

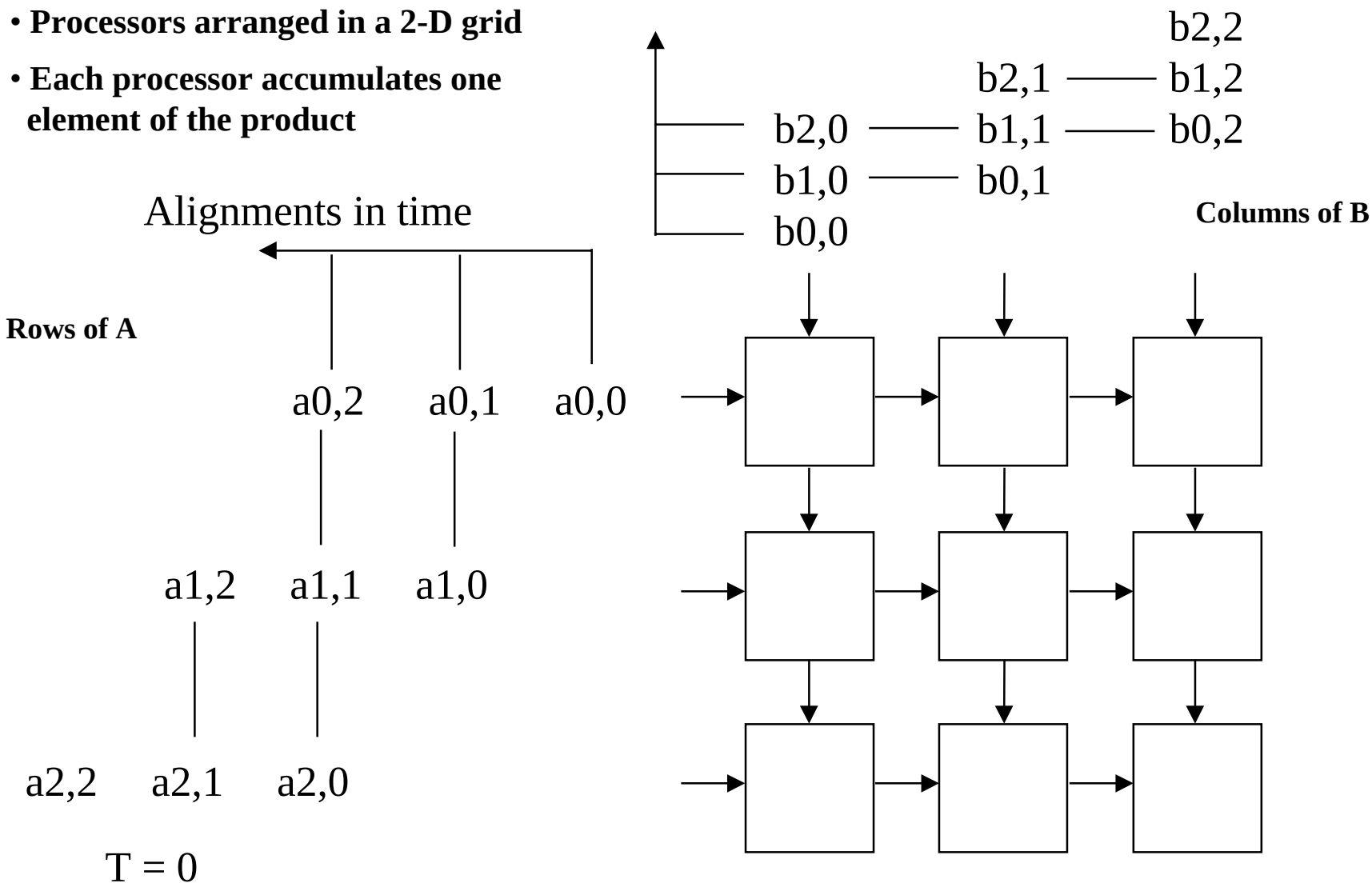


1. Observe multiplier/adder combination, so typical

2. Observe accumulation of results in-place. Every cell of final matrix is a processor. Results remain in these processors, next they should be shifted-out, if necessary.

Systolic Array Example:
3x3 Systolic Array Matrix Multiplication

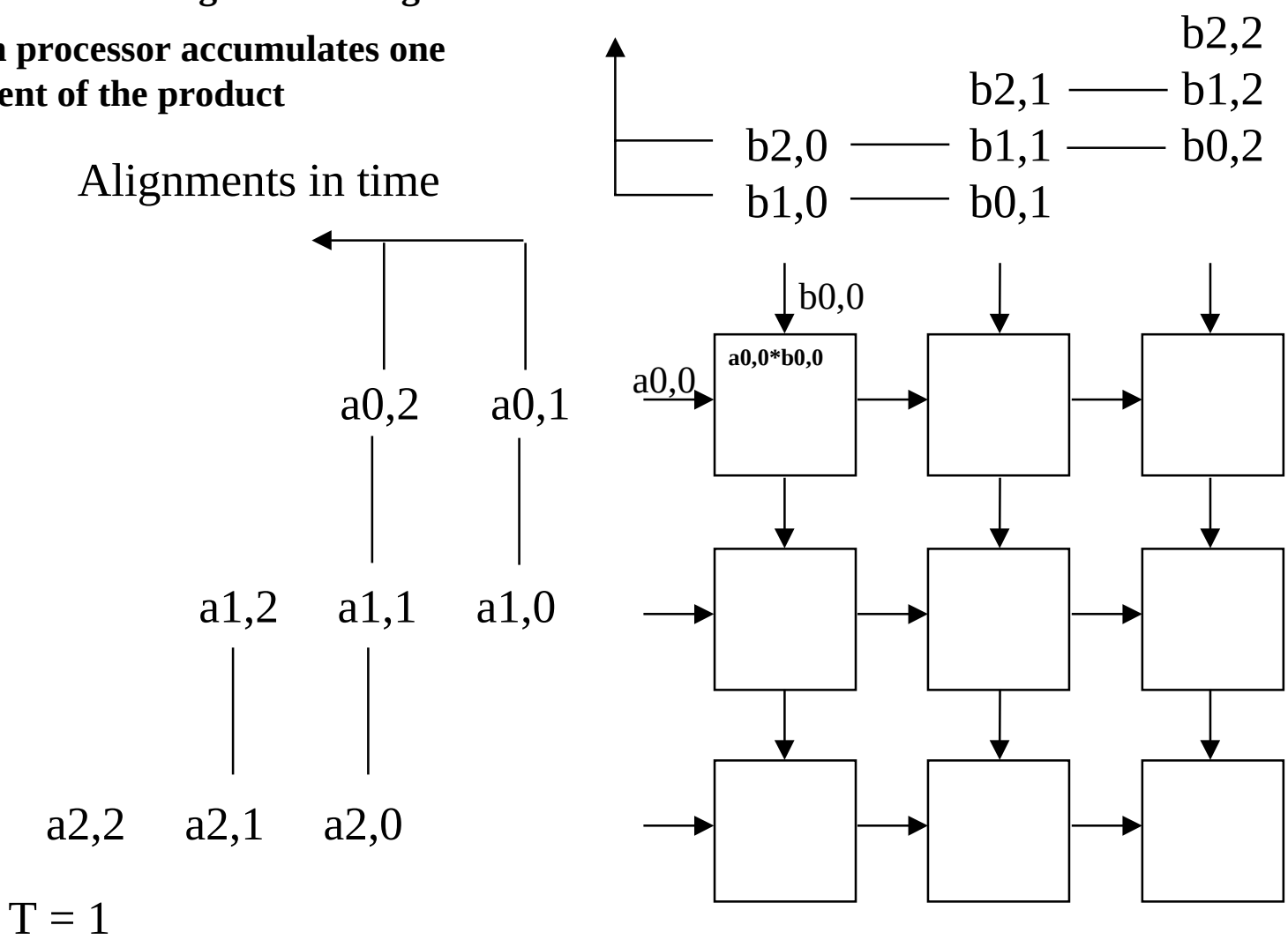
- **Processors arranged in a 2-D grid**
- **Each processor accumulates one element of the product**



Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/

Systolic Array Example:
3x3 Systolic Array Matrix Multiplication

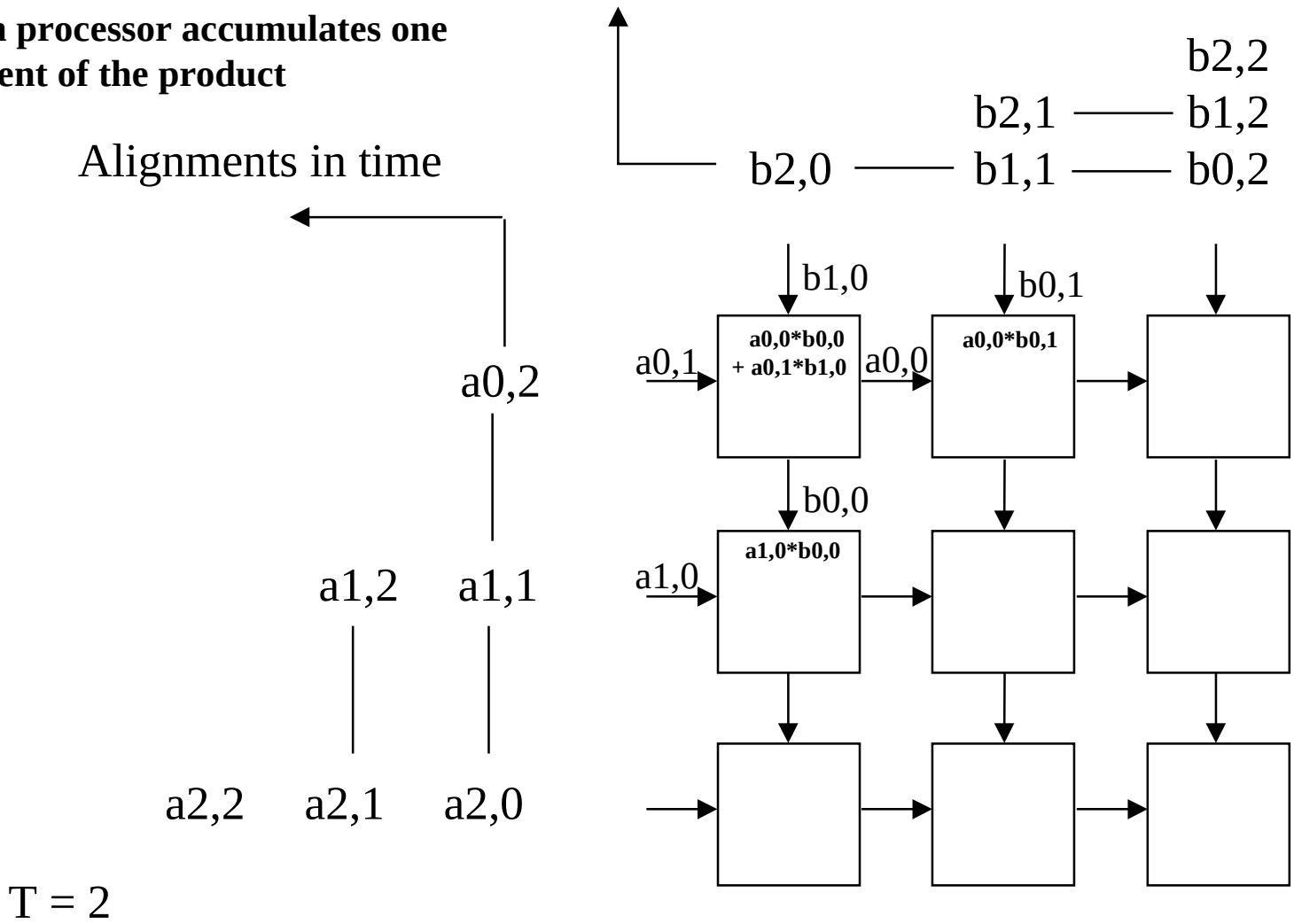
- **Processors arranged in a 2-D grid**
- **Each processor accumulates one element of the product**



Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/

Systolic Array Example:
3x3 Systolic Array Matrix Multiplication

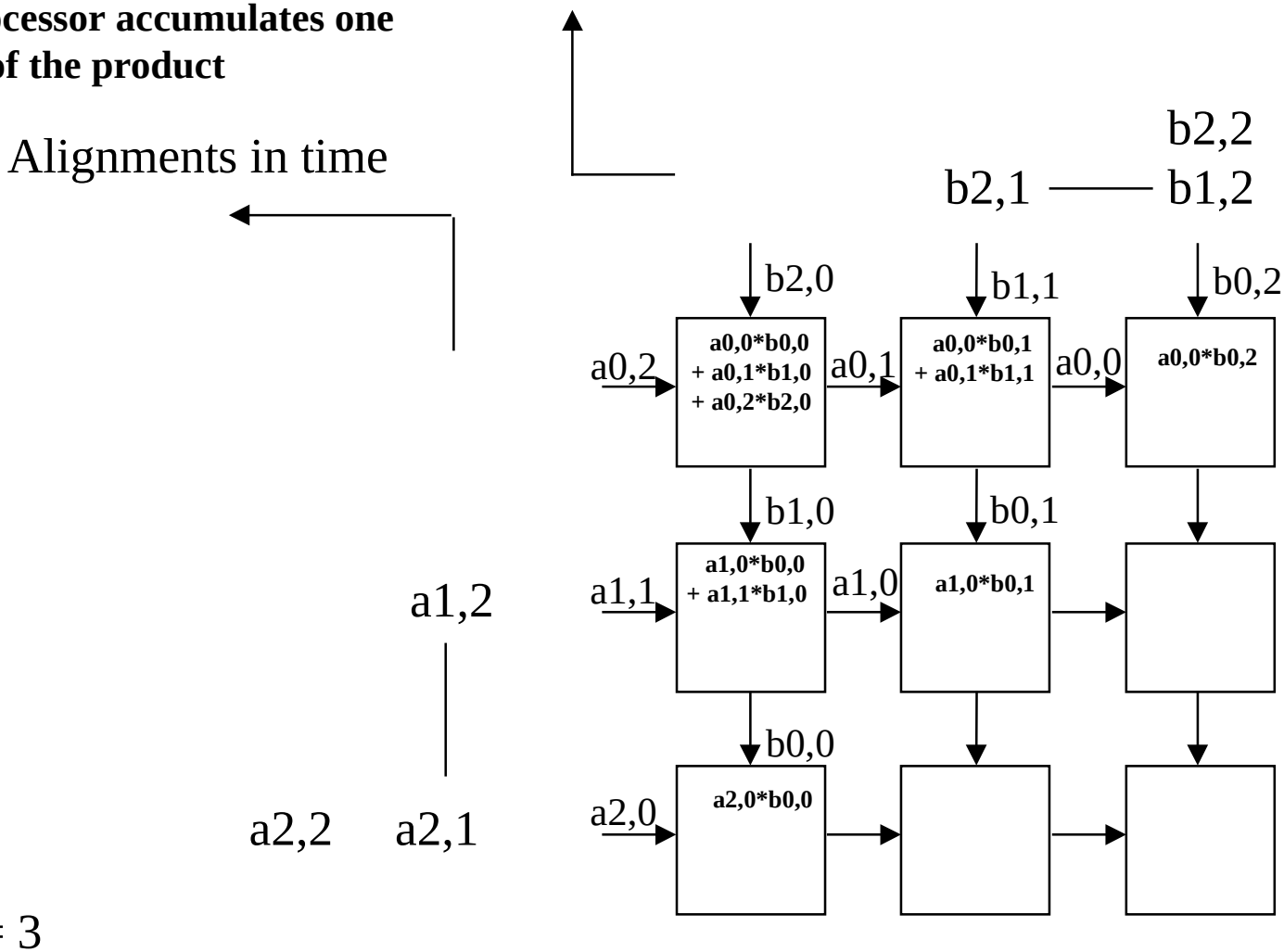
- **Processors arranged in a 2-D grid**
- **Each processor accumulates one element of the product**



Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/

Systolic Array Example:
3x3 Systolic Array Matrix Multiplication

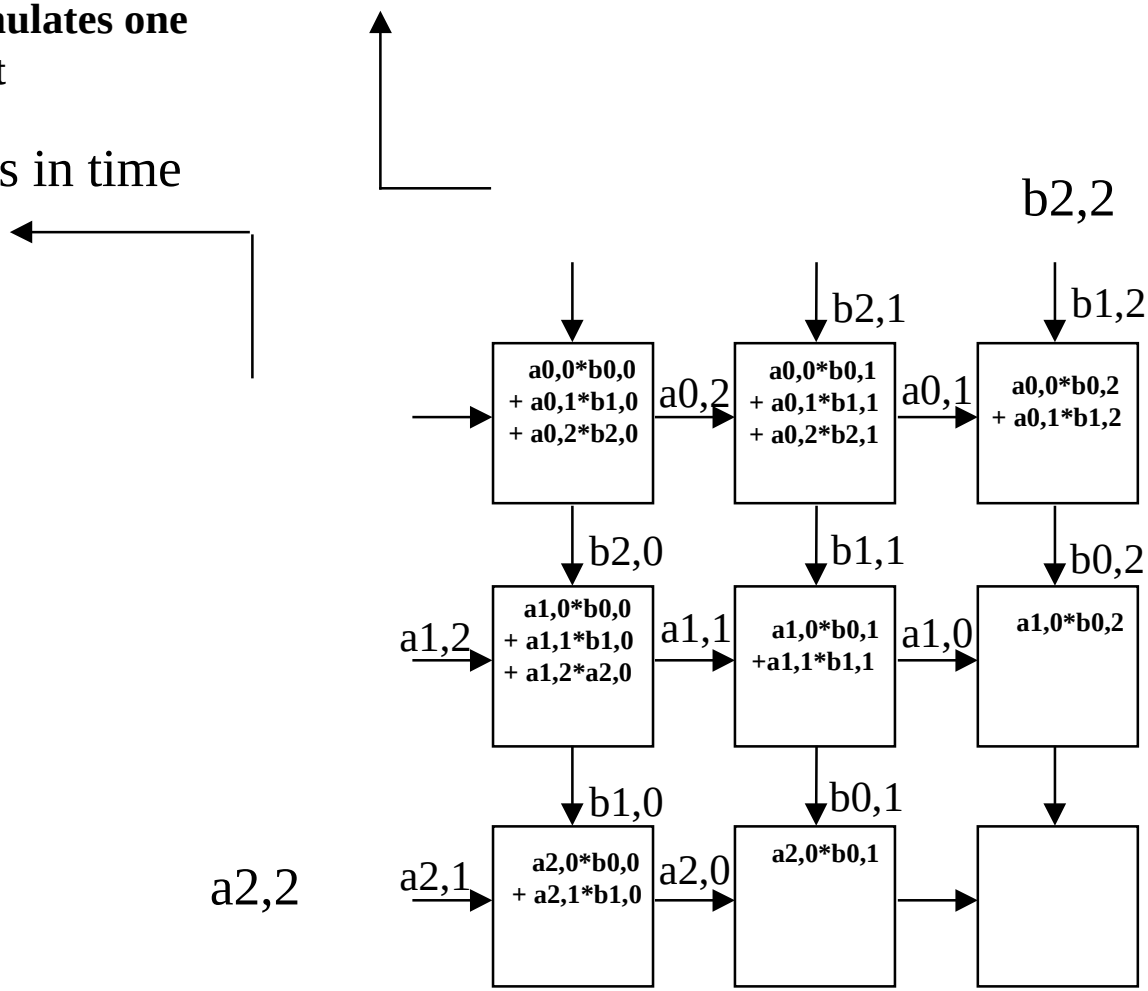
- **Processors arranged in a 2-D grid**
- **Each processor accumulates one element of the product**



Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/

Systolic Array Example:
3x3 Systolic Array Matrix Multiplication

- **Processors arranged in a 2-D grid**
- **Each processor accumulates one element of the product**

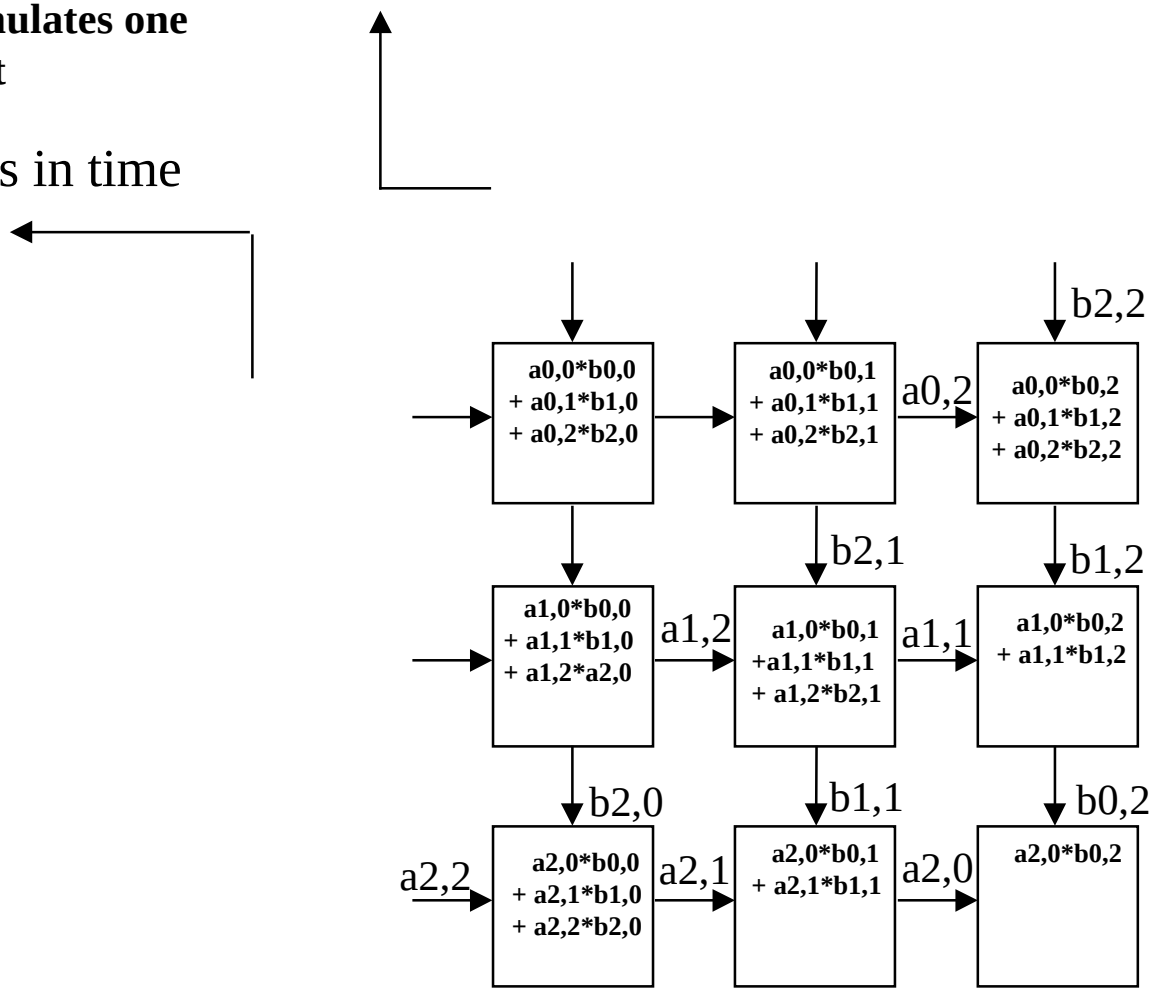


T = 4

Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/

Systolic Array Example:
3x3 Systolic Array Matrix Multiplication

- **Processors arranged in a 2-D grid**
- **Each processor accumulates one element of the product**

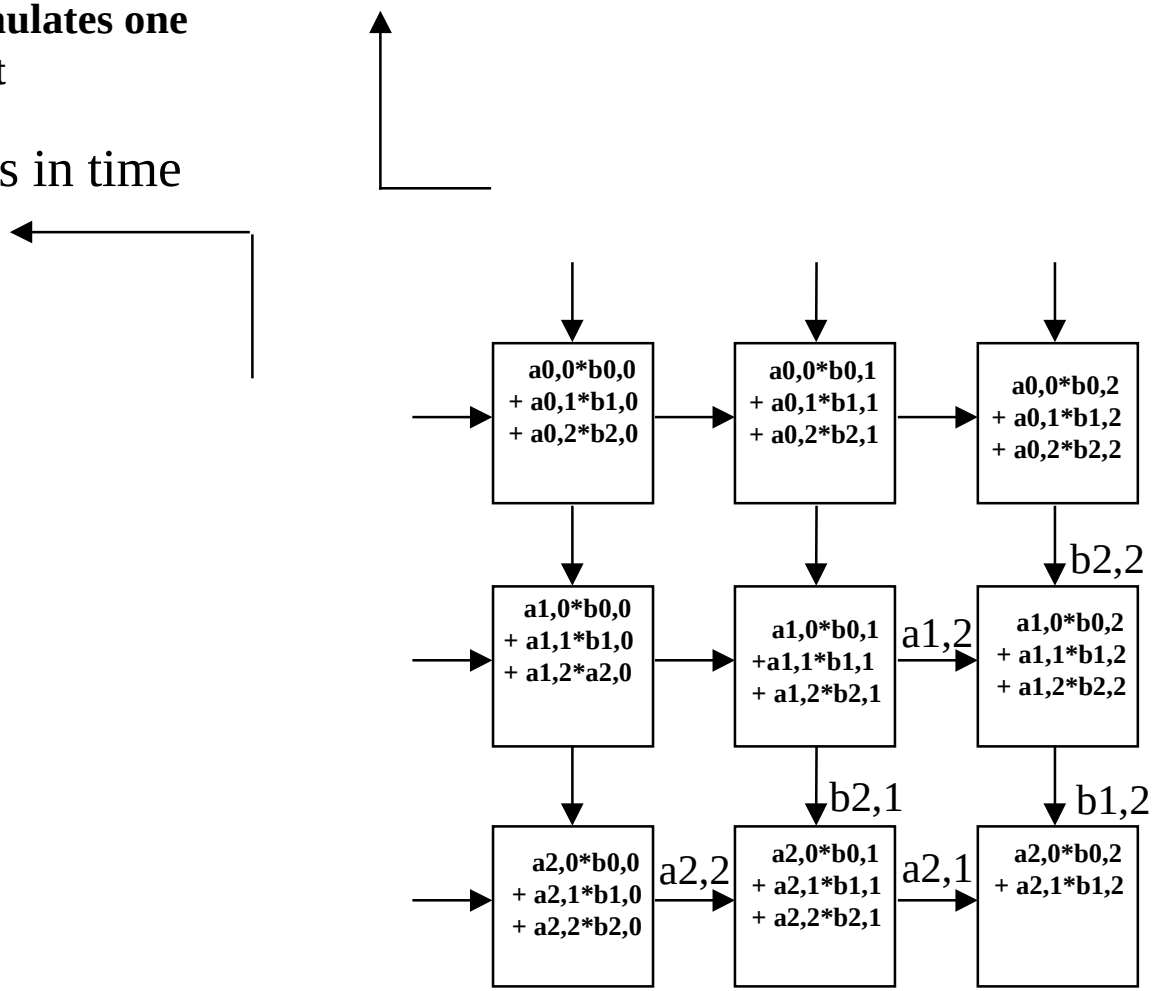


T = 5

Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/

Systolic Array Example:
3x3 Systolic Array Matrix Multiplication

- **Processors arranged in a 2-D grid**
- **Each processor accumulates one element of the product**

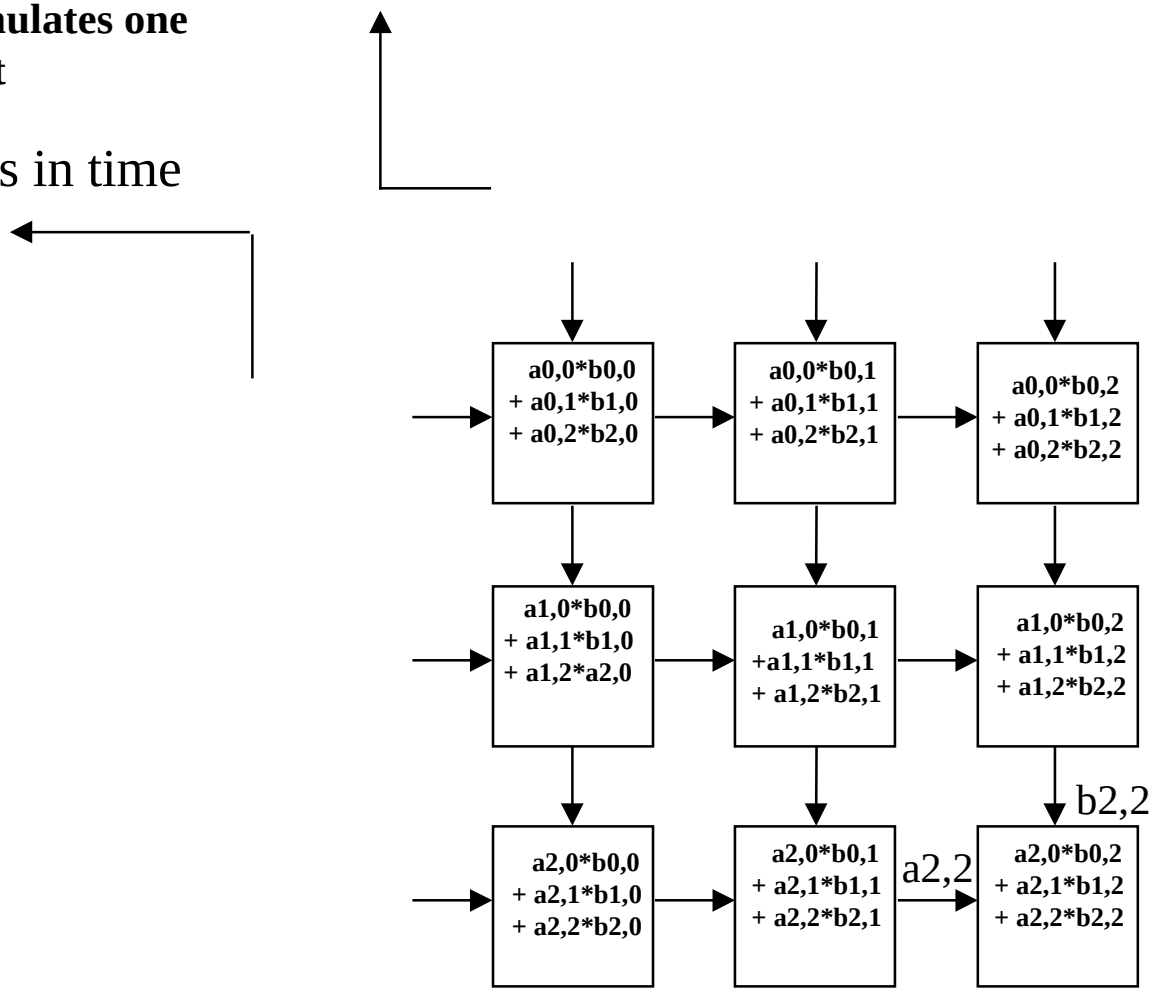


T = 6

Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/

Systolic Array Example:
3x3 Systolic Array Matrix Multiplication

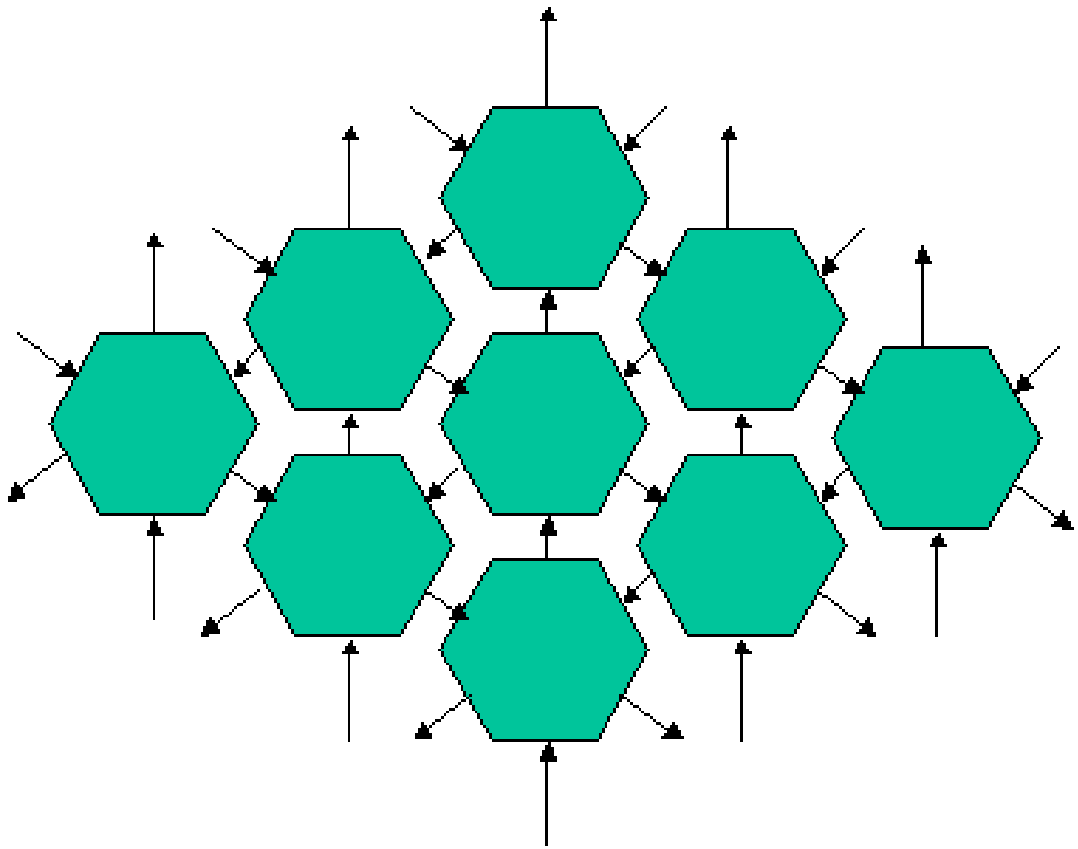
- **Processors arranged in a 2-D grid**
- **Each processor accumulates one element of the product**



Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/

Hexagonal Systolic Array for matrix-matrix multiplication

This is an interesting modification of previous design in which movement is in three directions and the results are not stored but flow out



HEXAGONAL SYSTOLIC ARRAY

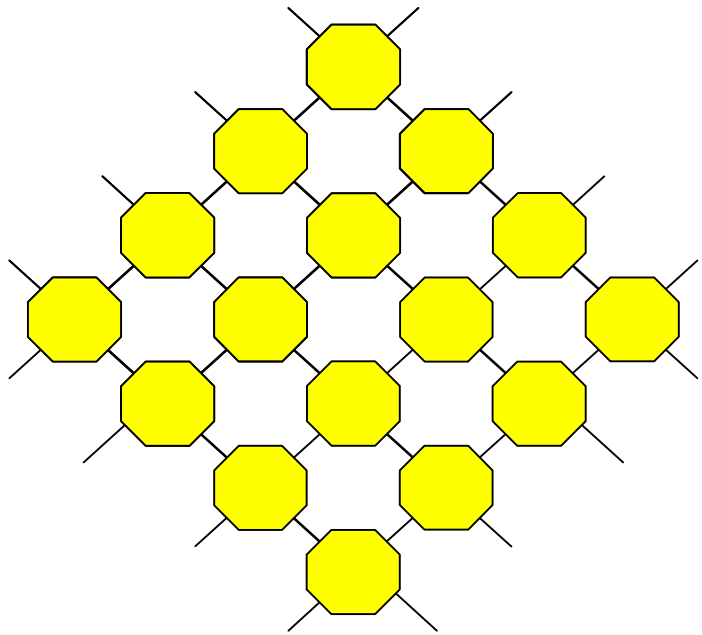
THINK: MULTIPLE MATRIX MULTIPLICATION

3N + MIN (W1, W2)

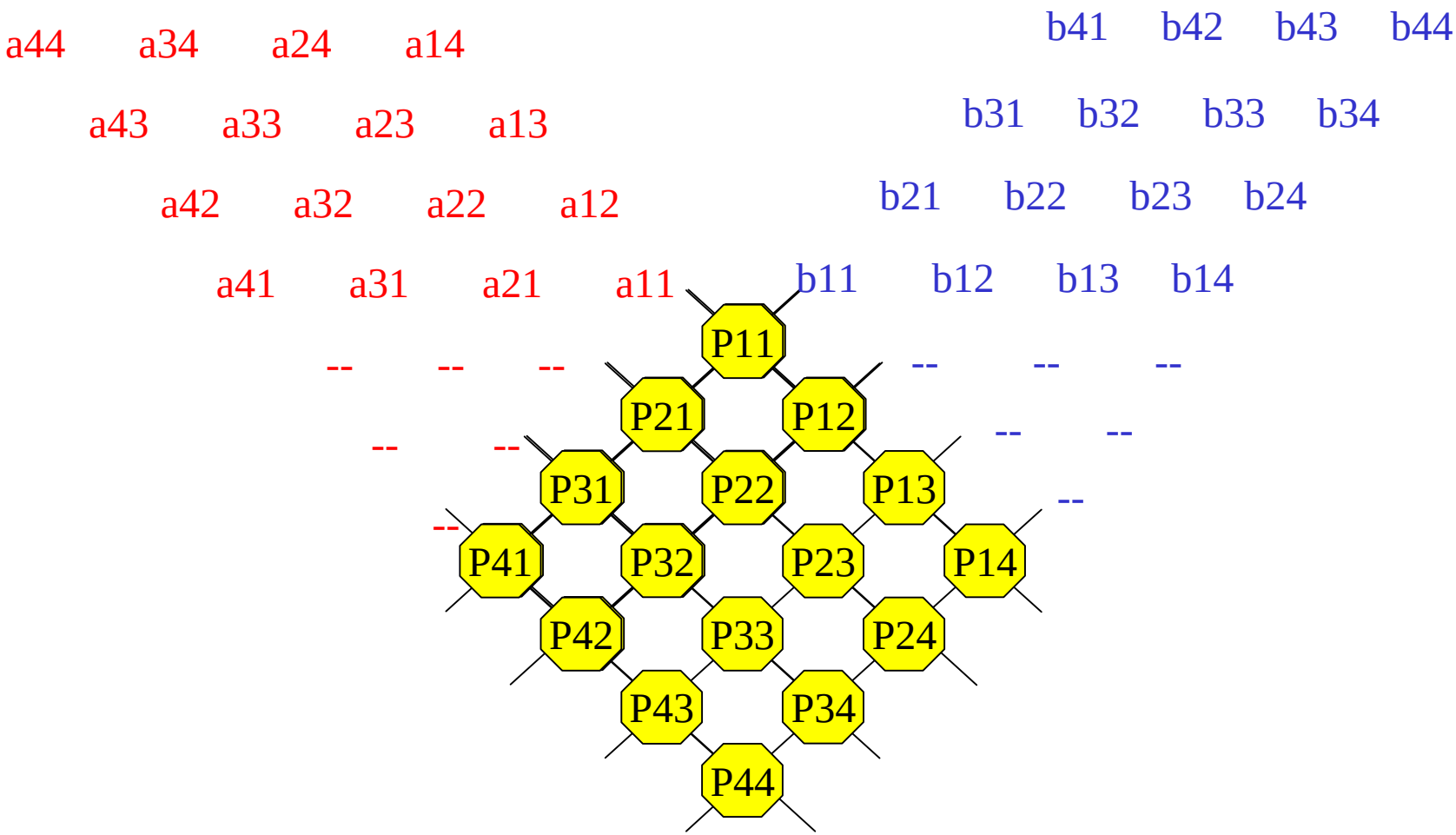
It is your task to design each processor in detail

Example Systolic Algorithm: Matrix Multiplication

- Problem: multiply two nxn matrices A ={a_ij} and B={b_ij}. Product matrix will be R={r_ij}.
- Systolic solution uses 2D array with NxN cells, 2 input streams and 2 output streams

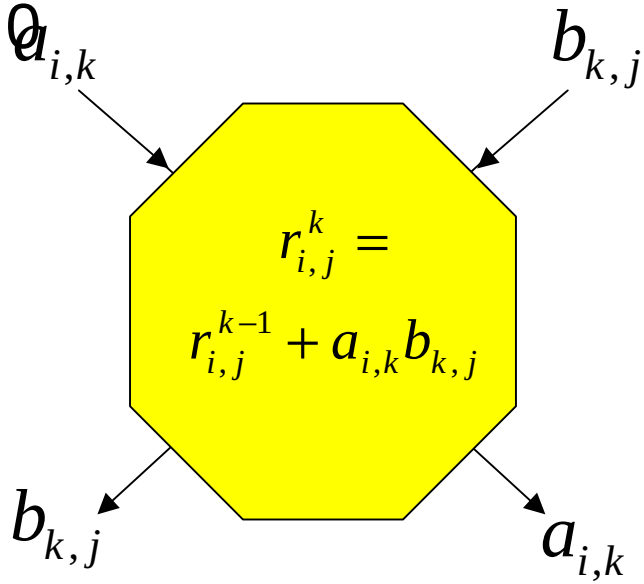


Systolic Matrix Multiplication



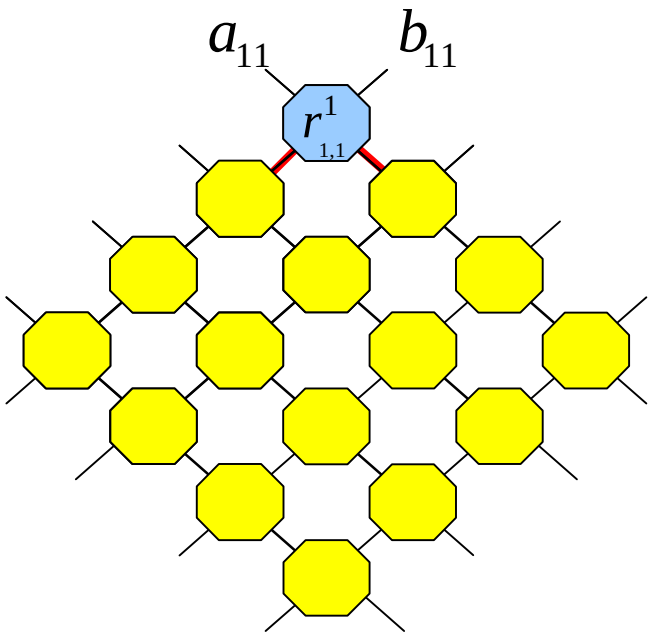
Operation at each cell

- Each $r_{i,j}^{k}$ cell updates at each time step as shown below
- $r_{i,j}^{0}$ initialized to 0

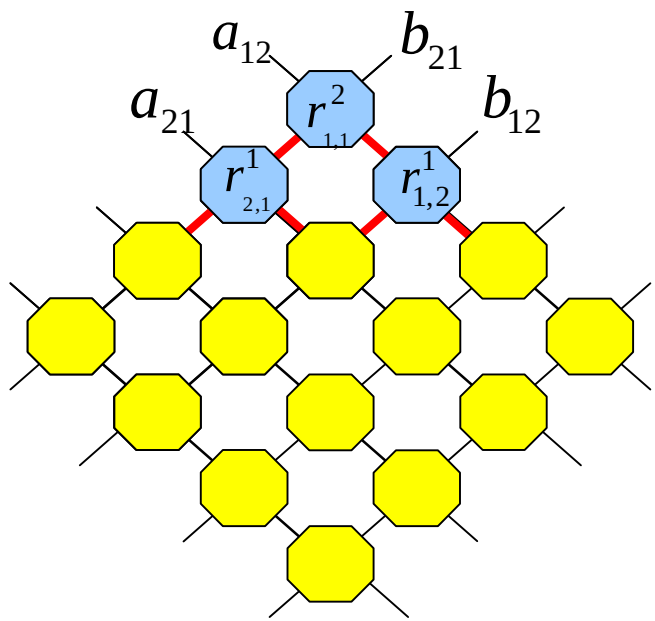


Data Flow for Systolic MM

- Beat 1

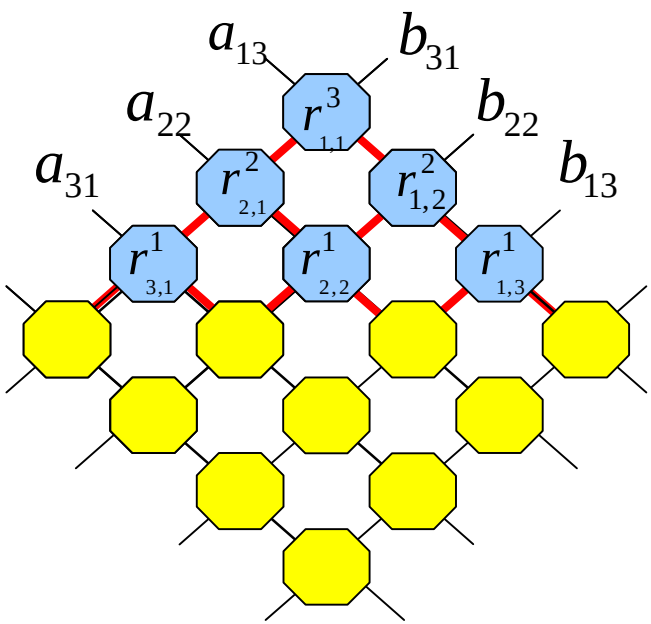


- Beat 2

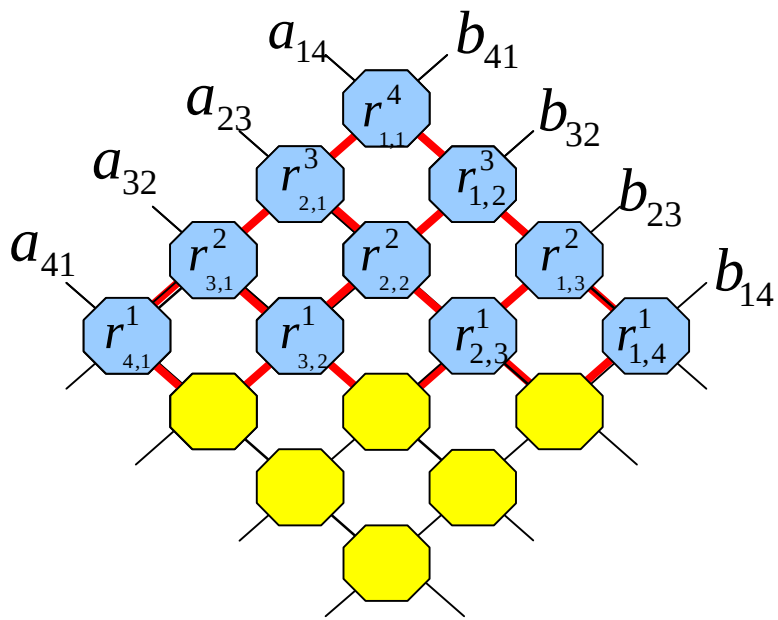


Data Flow for Systolic MM

- Beat 3

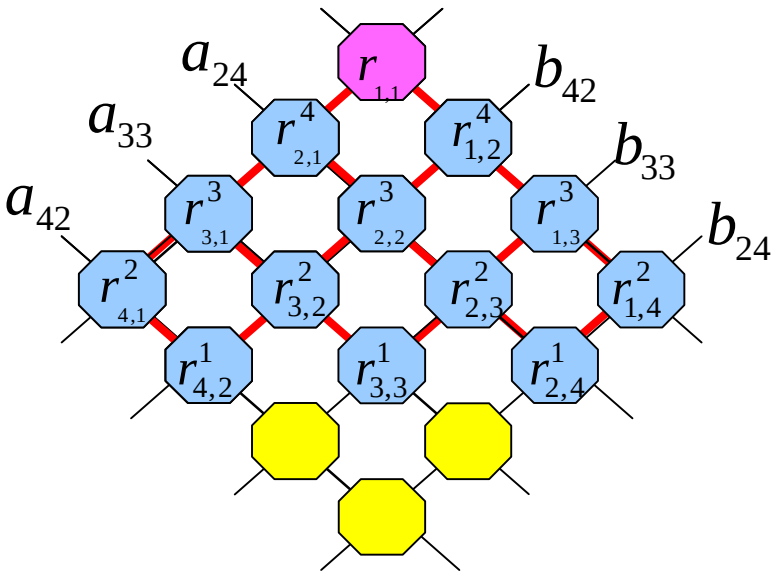


- Beat 4

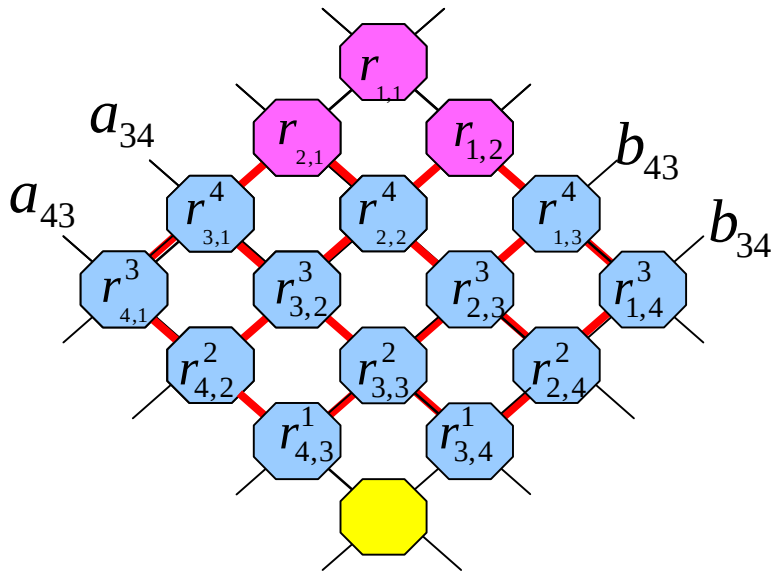


Data Flow for Systolic MM

- Beat 5

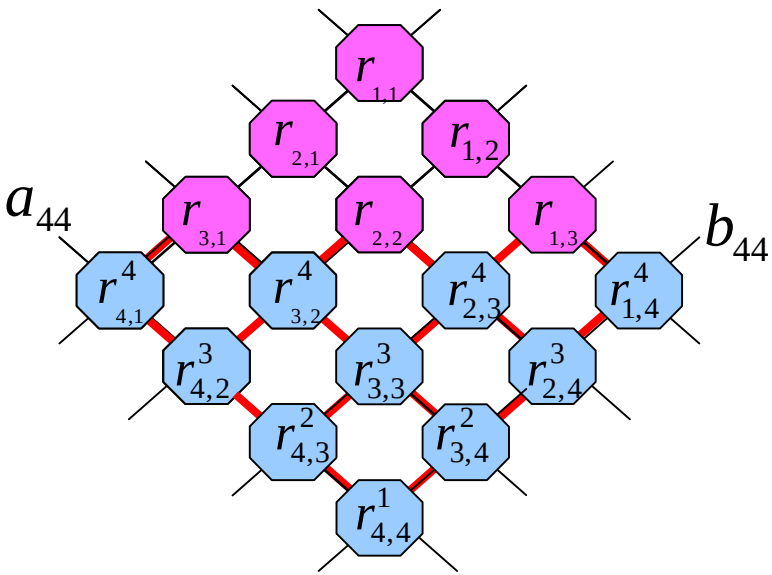


- Beat 6

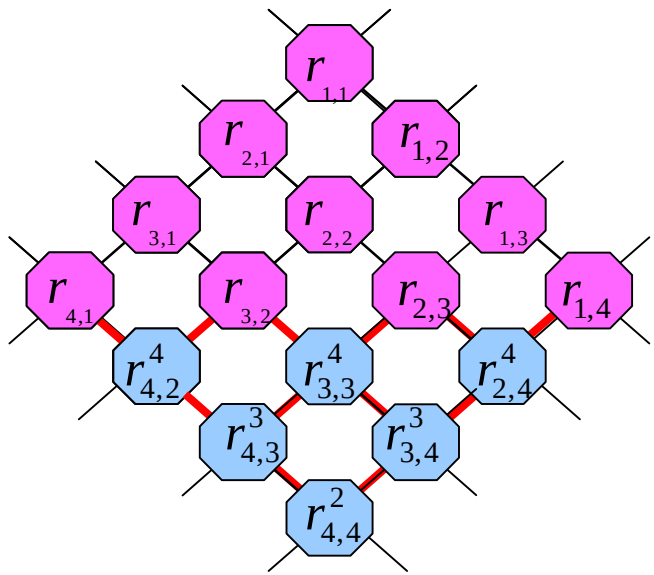


Data Flow for Systolic MM

- Beat 7

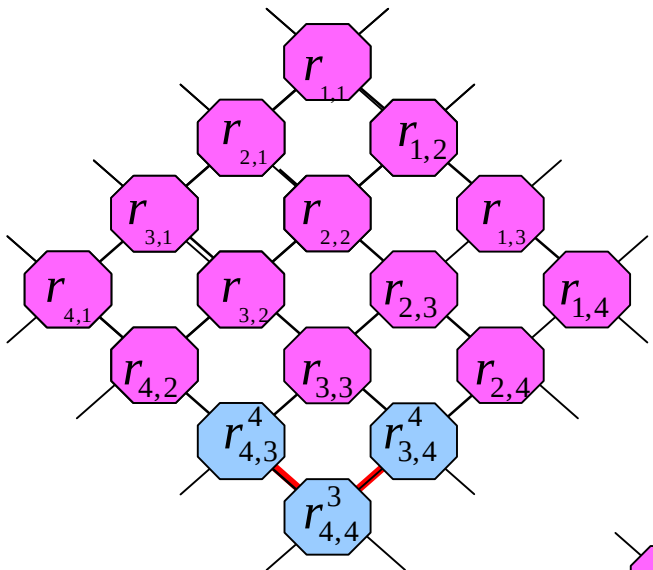


- Beat 8

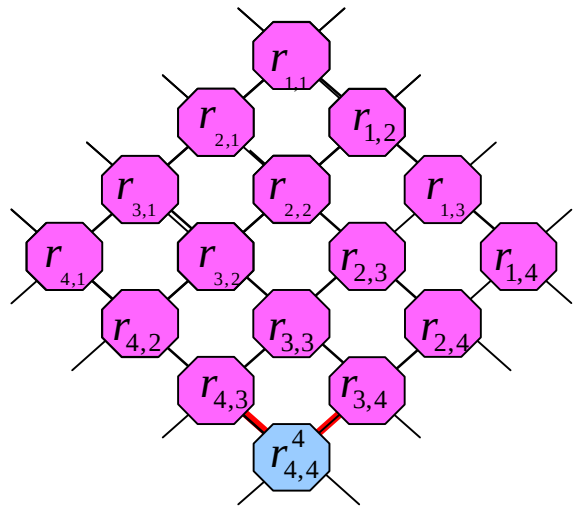


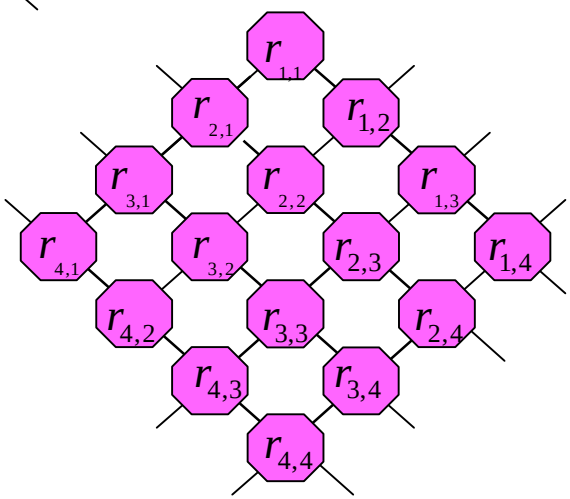
Data Flow for Systolic MM

- Beat 9



- Beats 10 and 11



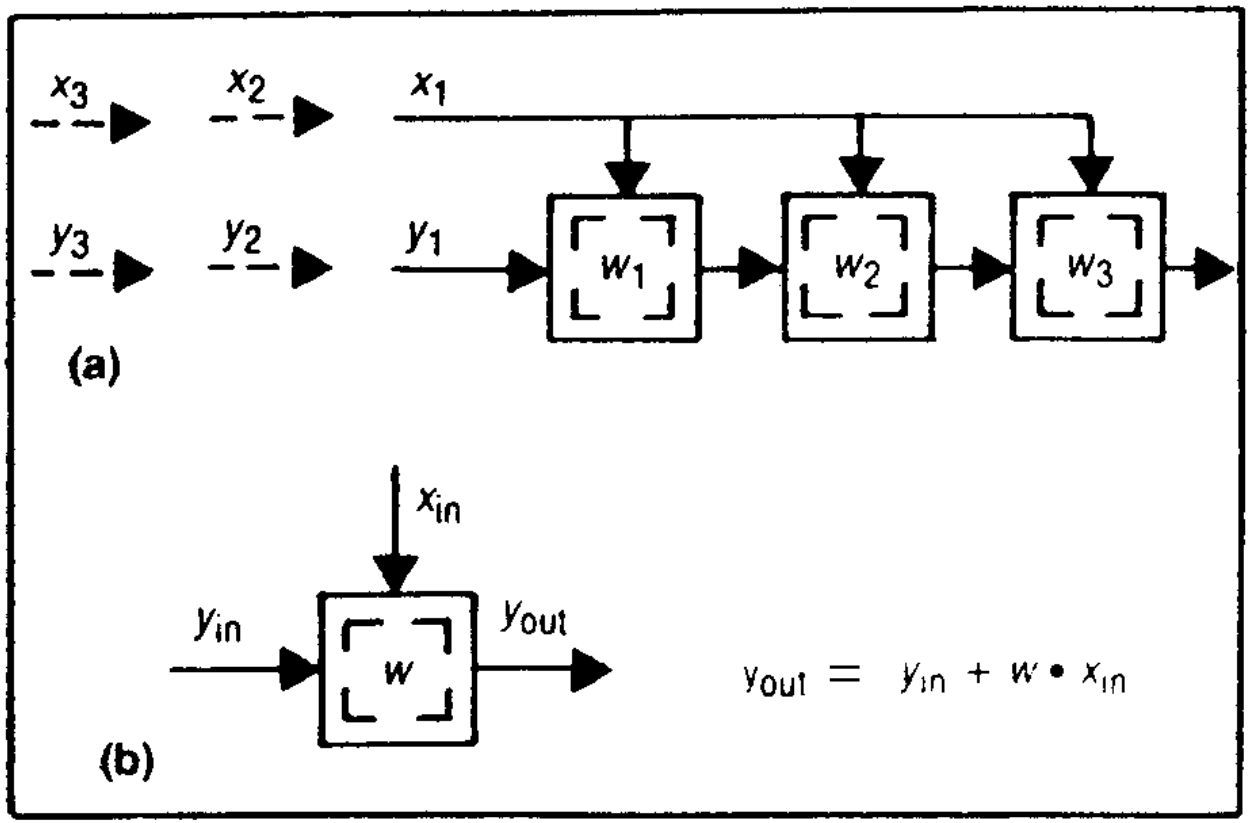


Systolic one-dimensional Convolvers

1. Use this method for 1D polynomial multiplication
2. Generalize to two dimensions.

台灣大學電機系
吳安宇

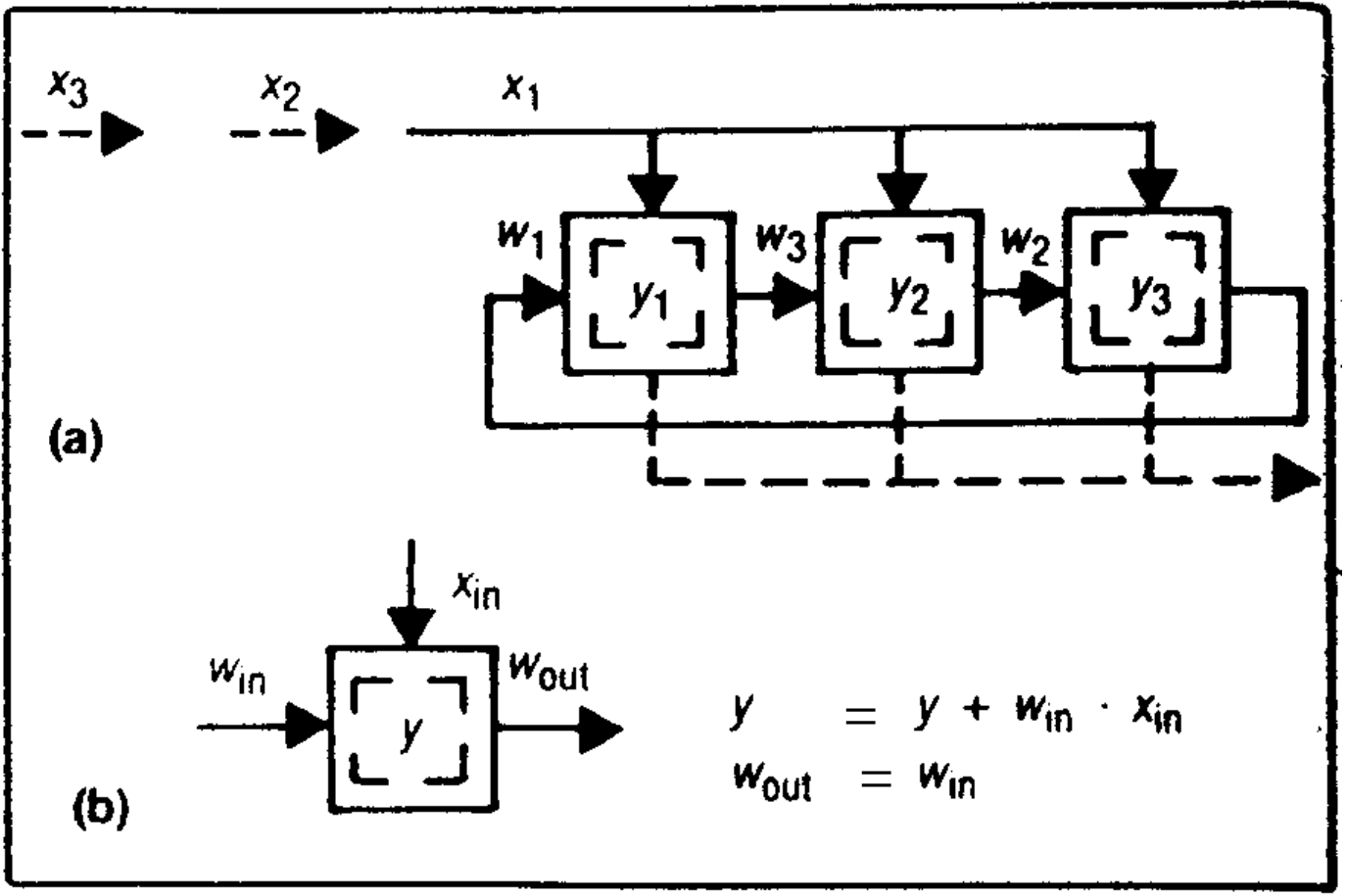
Design B1



• Previously proposed for circuits to implement a pattern matching processor and for a circuit to implement polynomial multiplication.

- Broadcast input , move results , weights stay
- (Semi-systolic convolution arrays with global data communication

Design B2



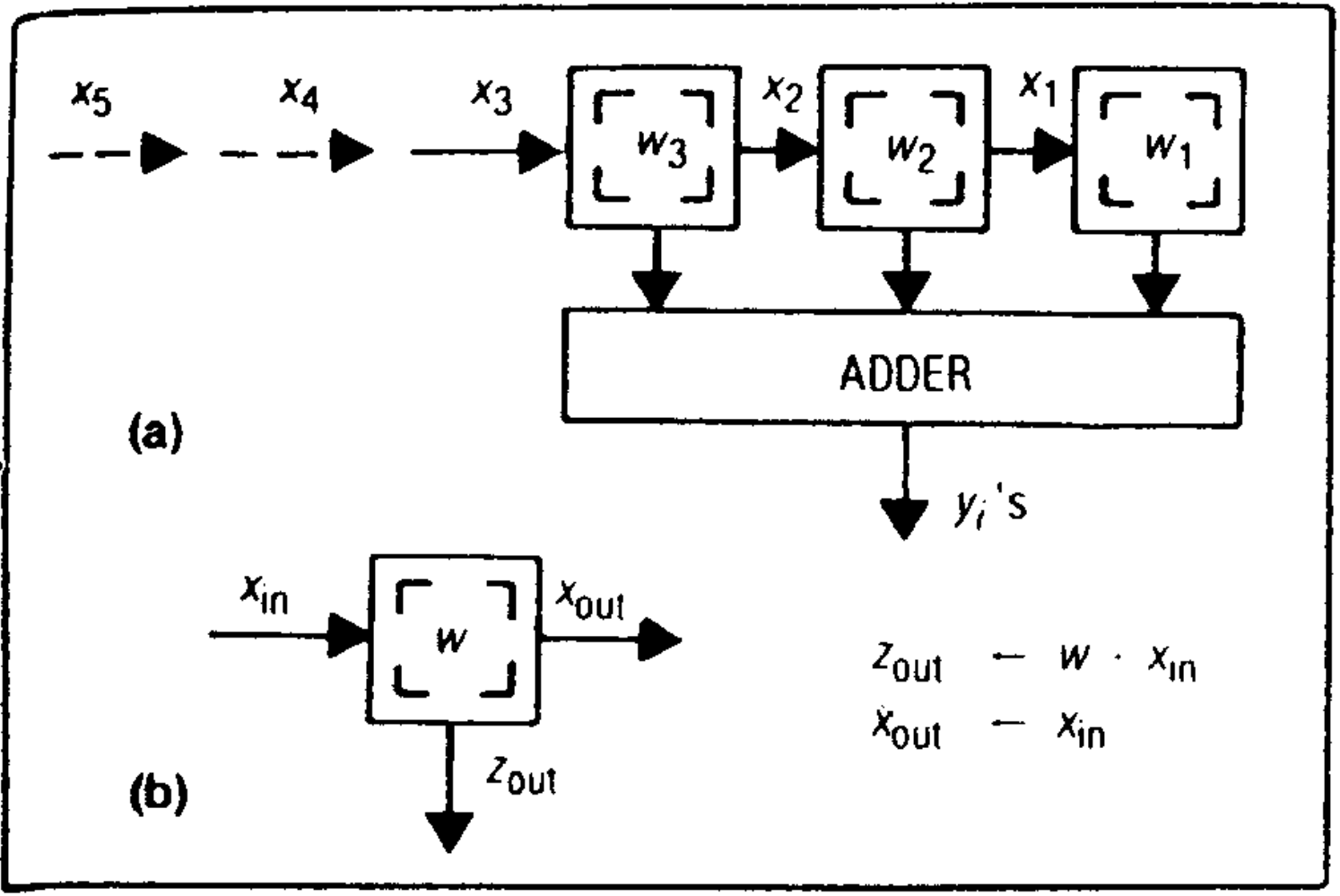
• The path for moving y_i's is wider then w_i's because of y_i's carry more bits then w_i's in numerical accuracy.

• The use of multiplier-accumulators may also help increase precision of the result , since extra bit can be kept in these accumulators with modest cost.

Broadcast input , move weights , results stay

[(Semi-) systolic convolution arrays with global data communication]

Design F



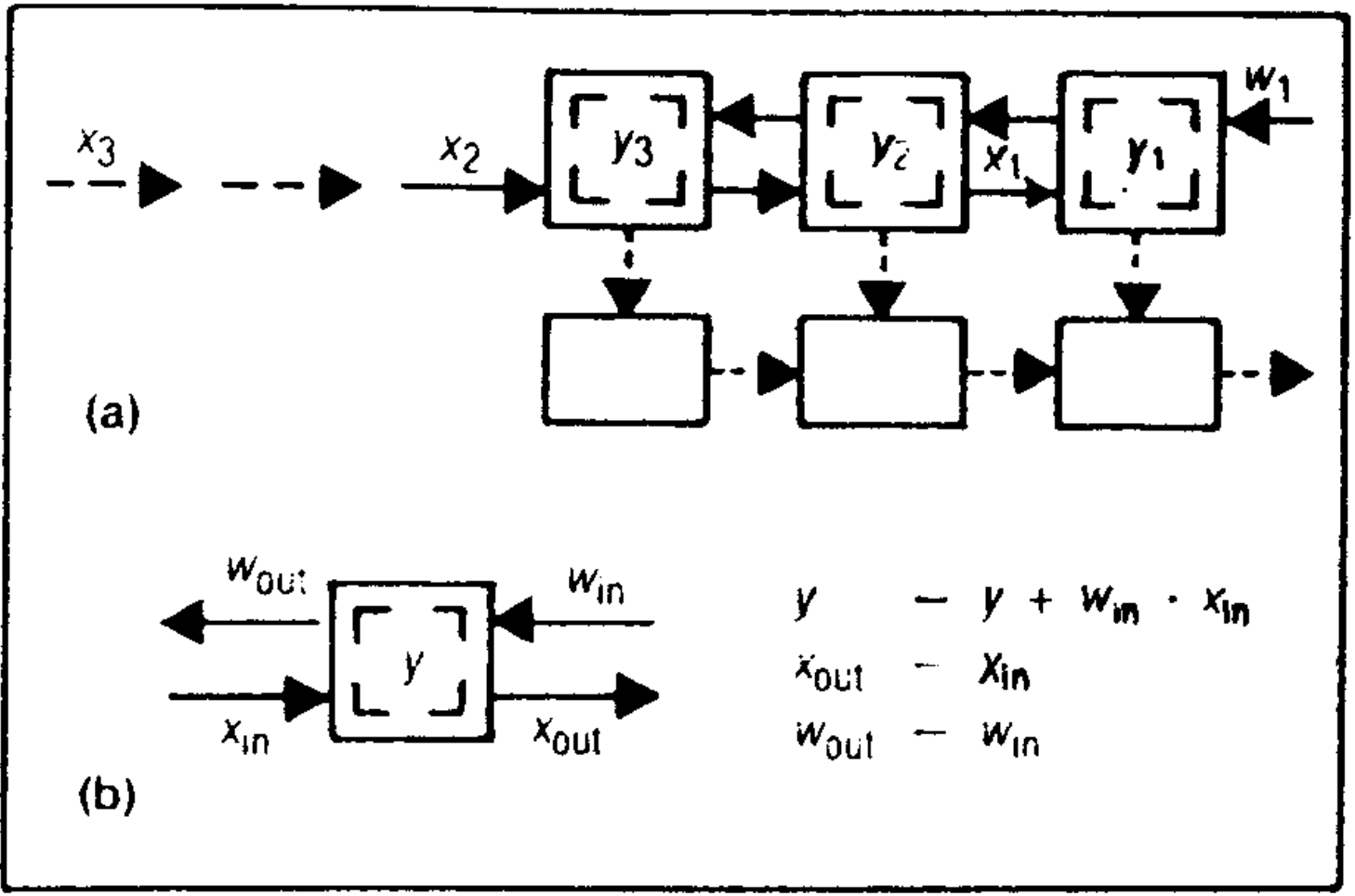
• When number of cell is large , the adder can be implemented as a pipelined adder tree to avoid large delay.

• Design of this type using unbounded fan-in.

- Fan-in results, move inputs, weights stay

- Semi-systolic convolution arrays with global data communication

Design R1



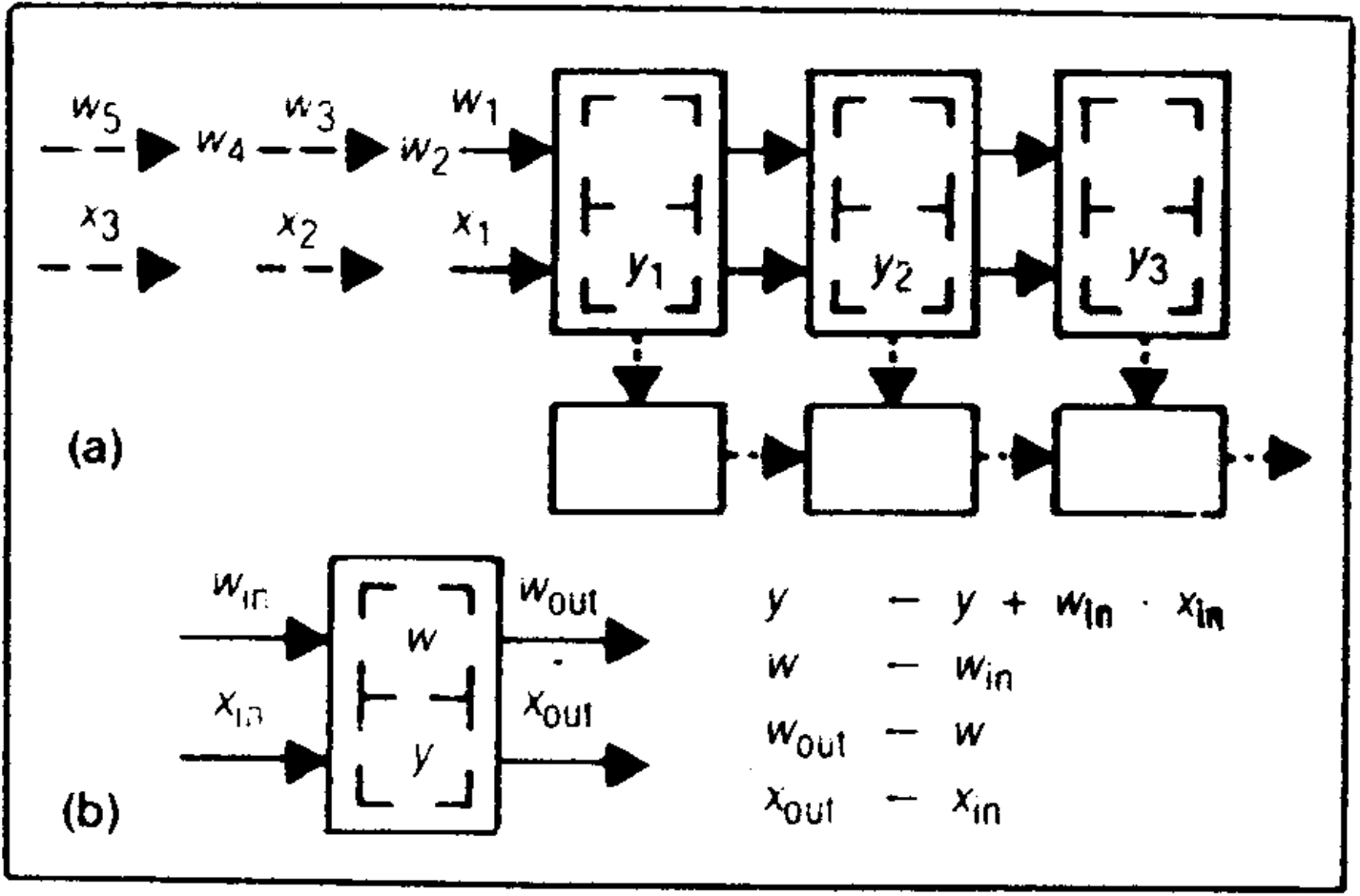
• Design R1 has the advantage that it dose not require a bus , or any other global network , for collecting output from cells.

• The basic ideal of this de-sign has been used to implement a pattern matching chip.

- Results stay, inputs and weights move in opposite directions

- Pure-systolic convolution arrays with global data communication

Design R2

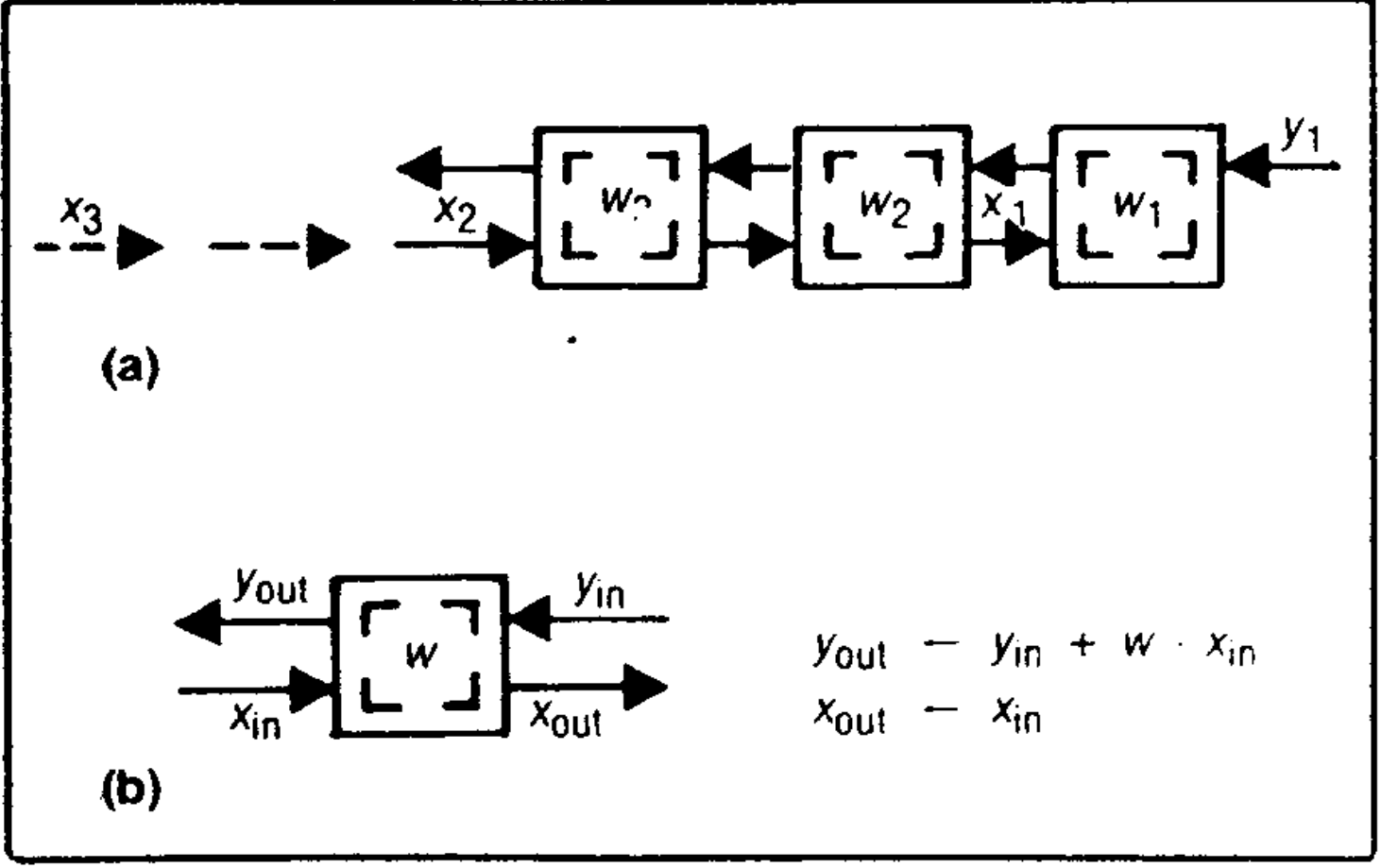


• Multiplier-accumulator can be used effectively and so can tag bit method to signal the output of each cell.

• Compared with R1 , all cells work all the time when additional register in each cell to hold a w value.

- Results stay , inputs and weights move in the same direction but at different speeds
- Pure-systolic convolution arrays with global data communication

Design W1



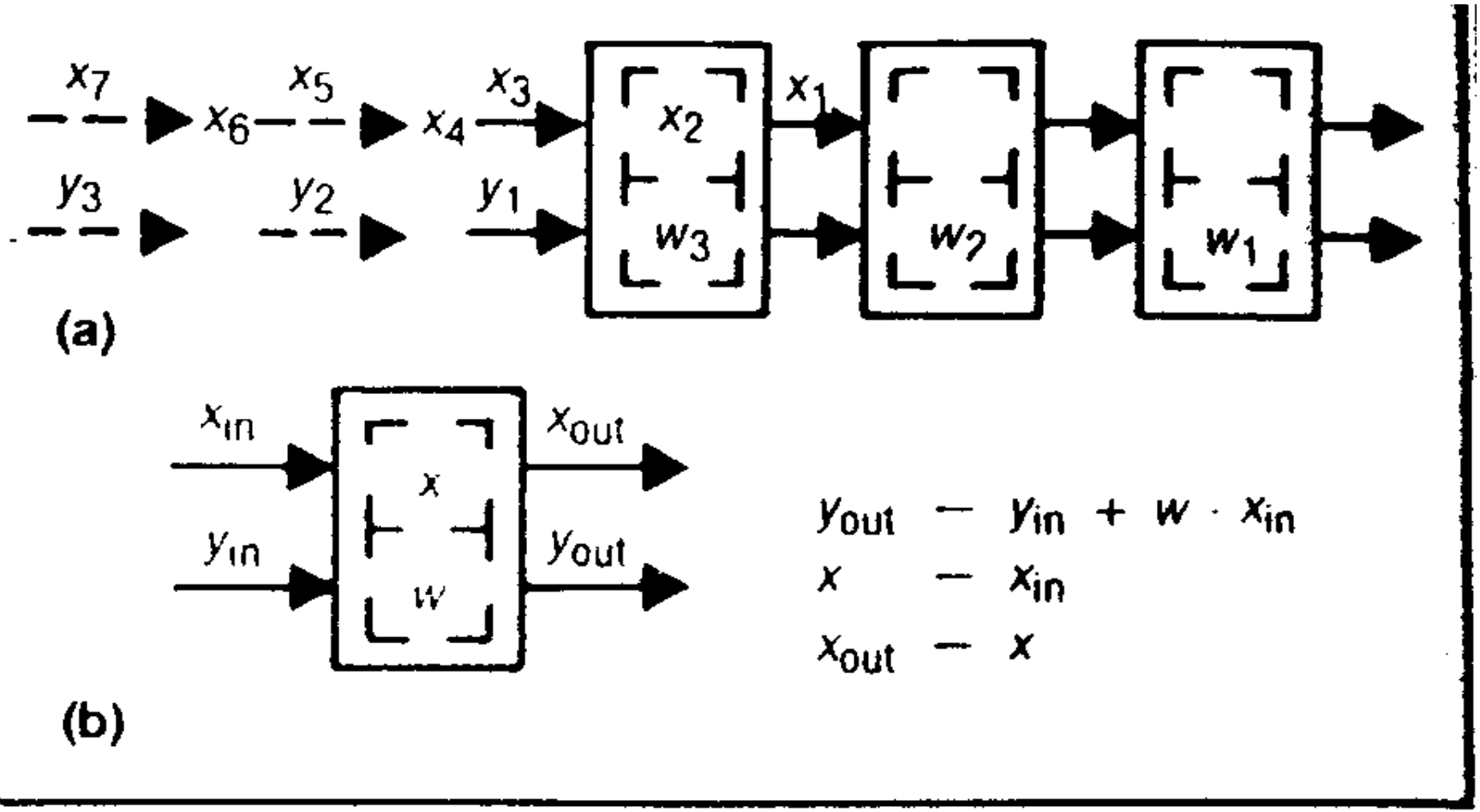
• This design is fundamental in the sense that it can be naturally extend to perform recursive filtering.

• This design suffers the same drawback as R1 , only approximately 1/2 cells work at any given time unless two independent computation are interleaved in the same array.

-Weights stay, inputs and results move in opposite direction

- Pure-systolic convolution arrays with global data communication

Design W2



• This design lose one advan-tage of W1 , the constant response time.

• This design has been extended to implement 2-D convolution , where high throughputs rather than fast response are of concern.

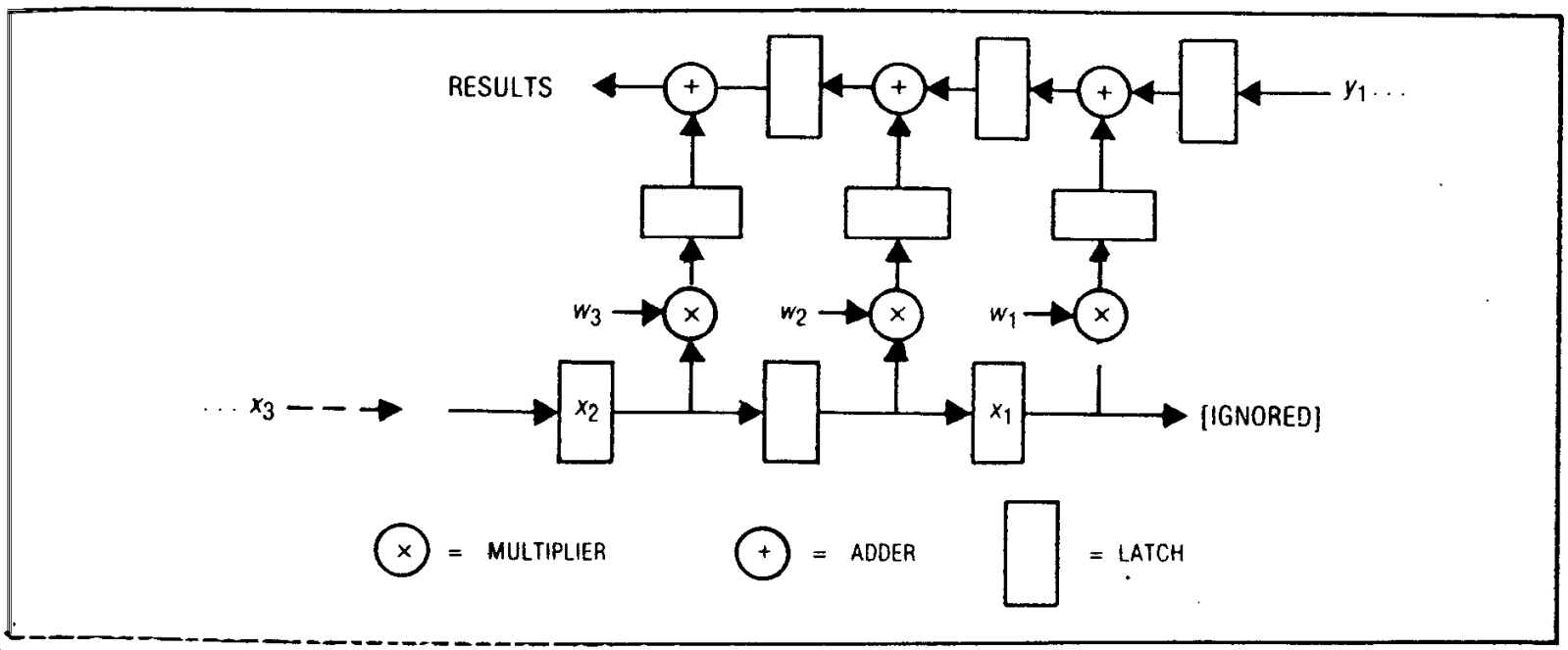
-Weights stay, inputs and results move in the same direction but at different speeds

- Pure-systolic convolution arrays with global data communication

Remarks

- Above designs are all possible systolic designs for the convolution problem.
- Using a systolic control path , weight can be selected on-the-fly to implement interpolation or adaptive filtering.
- We need to understand precisely the strengths and drawbacks of each design so that an appropriate design can be selected for a given environment.
- For improving throughput, it may be worthwhile to implement multiplier and adder separately to allow overlapping of their execution. (Such as next page show)
- When chip pin is considered , pure-systolic requires four; semi-systolic requires three I/O ports.

Overlapping the executions of multiply-and-add in design W1



Criteria and advantages

- The design makes multiple use of each input data item

  Because of this property , systolic systems can achieve high throughputs with modest I/O bandwidths for outside communication.

- The design uses extensive concurrency

  Concurrency can be obtained by pipelining the stages involved in the computation of each single result , by multiprocessing many results in parallel, or by both.

Criteria and advantages

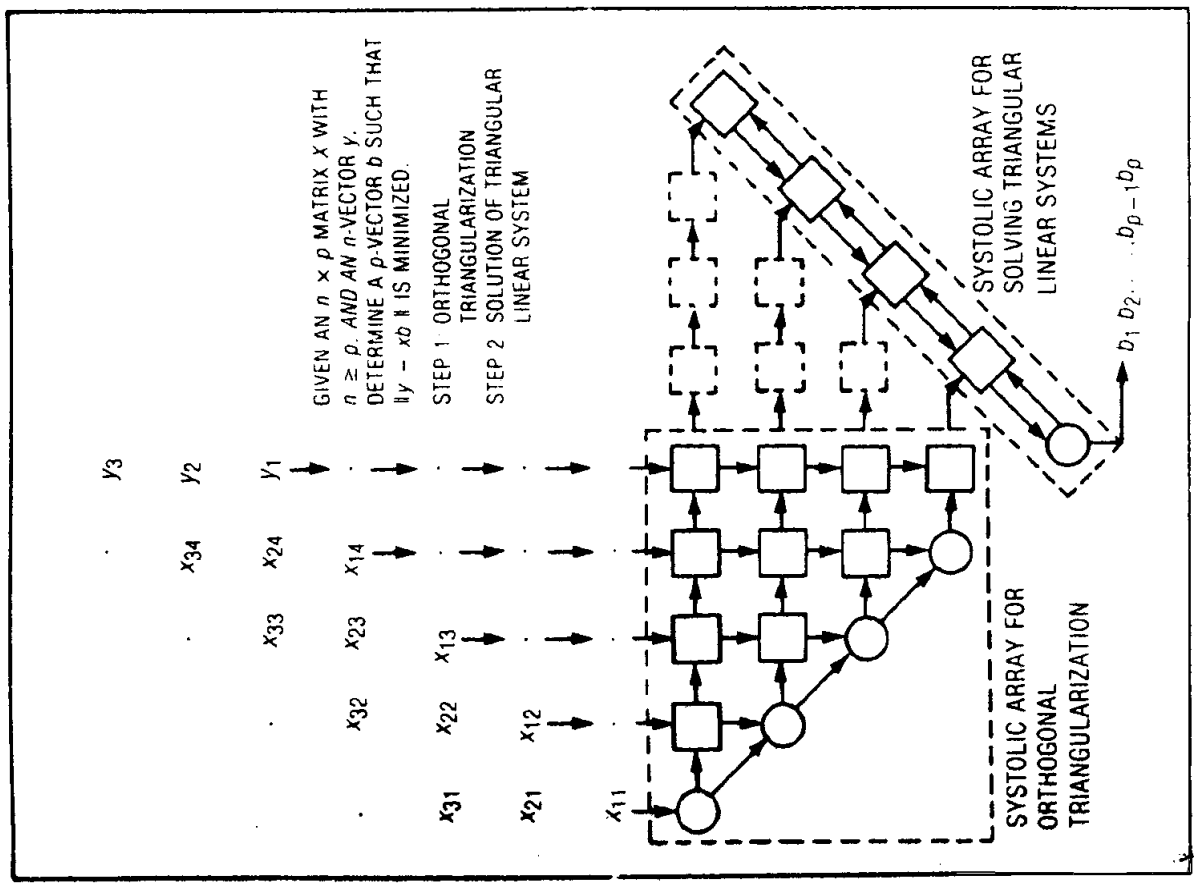
- There are only a few types of simple cells

  To achieve performance goals, a systolic system is likely to use a large number of cells which must be simple and of only a few types to curtail design and implementation cost.

- Data and control flow are simple and regular

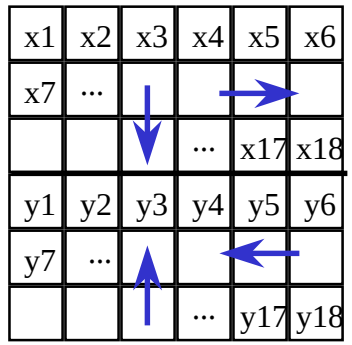
  Pure systolic system totally avoid long-distance or irregular wires for data communication.

Orthogonal Triangularization. On-the-fly least-squares solutions using one and two dimensional systolic array, with p=4.

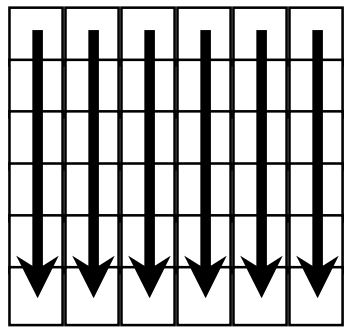


parallel merge

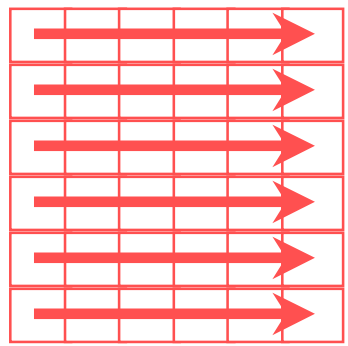
initial situation:



1.) sort columns
(odd-even-transposition sort)



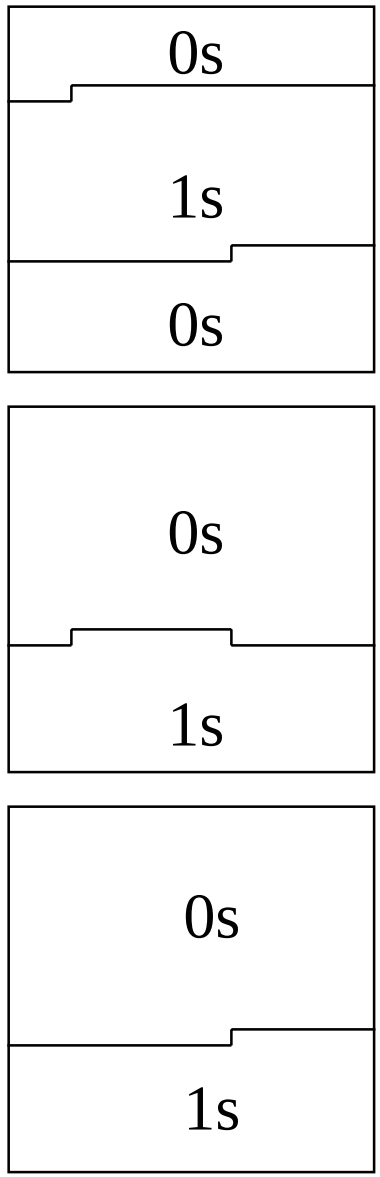
2.) sort rows
(odd-even-transposition sort)



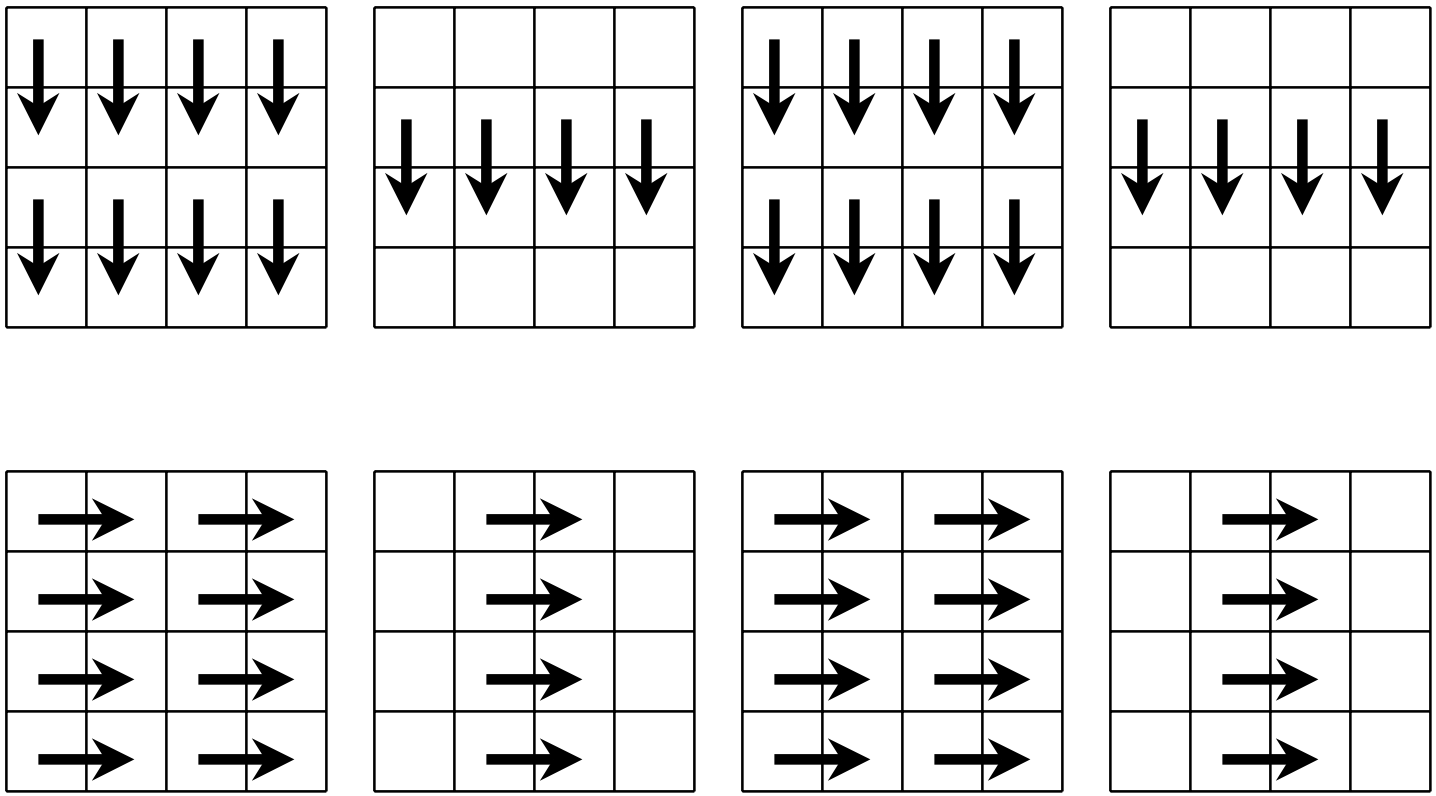
sorted !!!!

0-1 principle

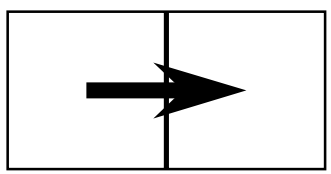
- The 0-1 principle states that if all sequences of 0 and 1 are sorted properly than this is a correct sorter.
- The sorter must be based on moving data.



MIMD-mesh (clocked)



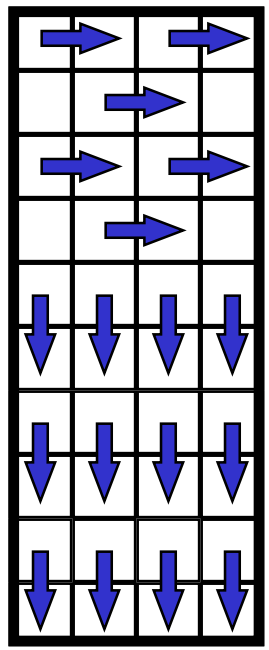
min



max

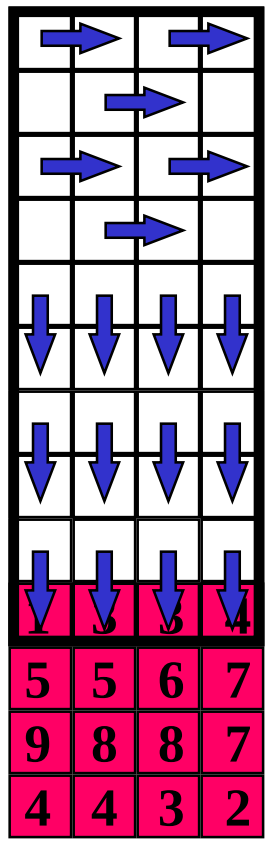
Time: 2n

systolic merge

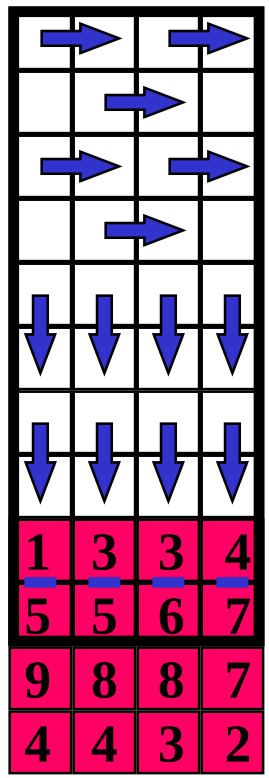


1	3	3	4
5	5	6	7
9	8	8	7
4	4	3	2

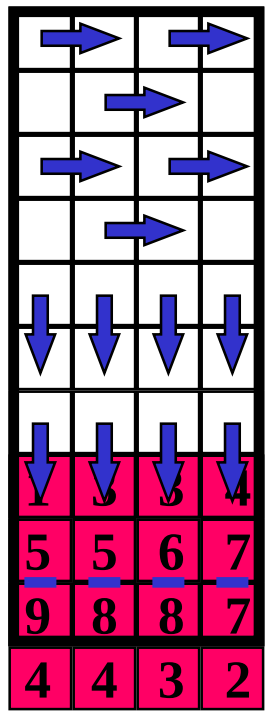
systolic merge



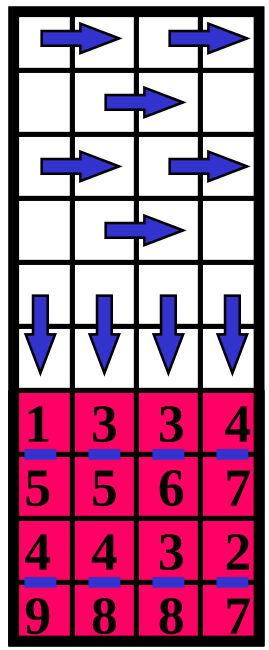
systolic merge



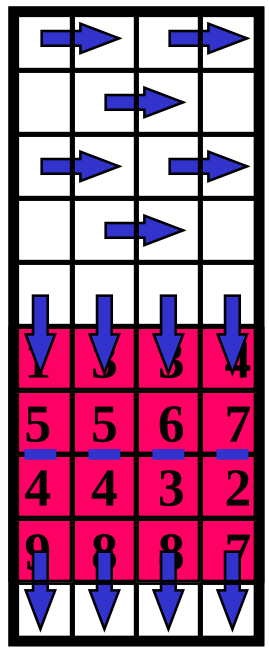
systolic merge



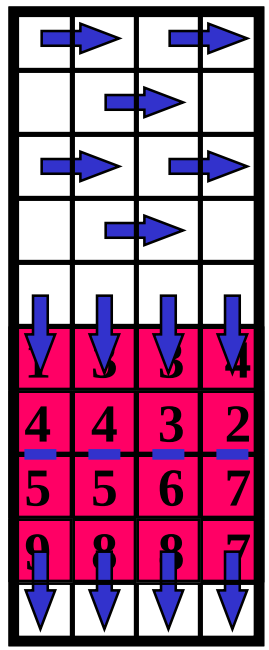
systolic merge



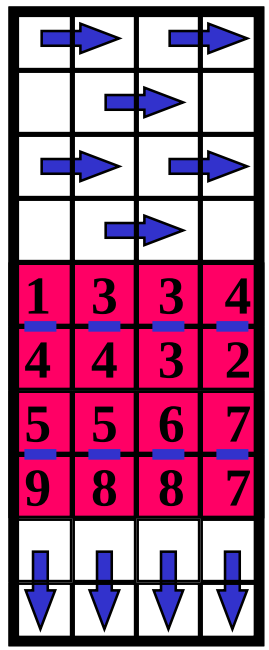
systolic merge



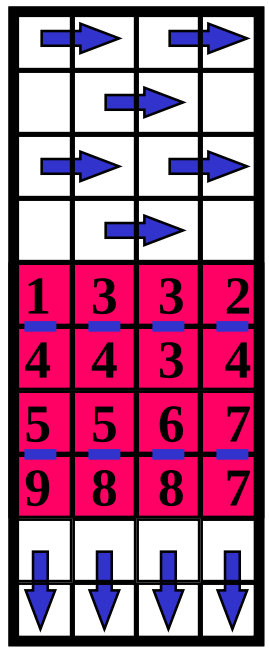
systolic merge



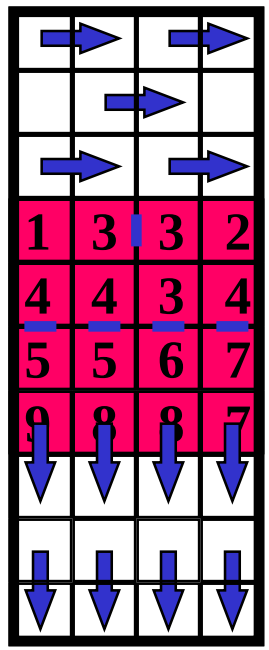
systolic merge



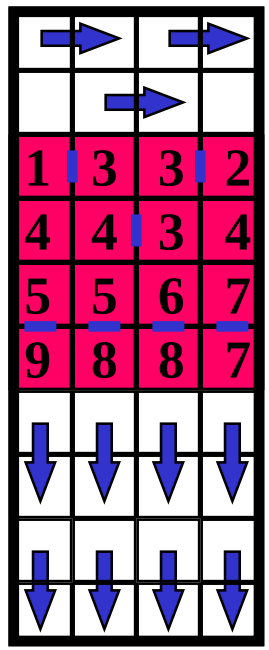
systolic merge



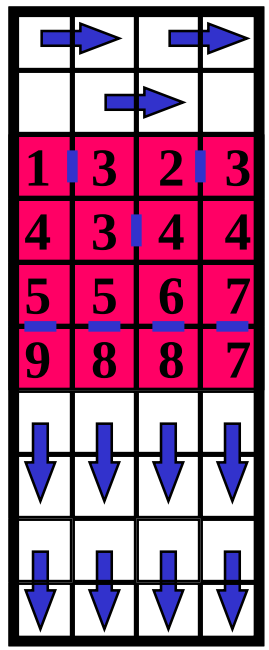
systolic merge



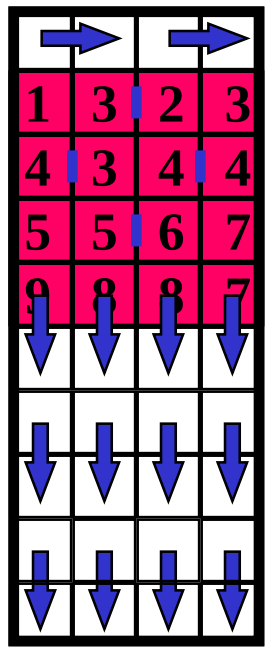
systolic merge



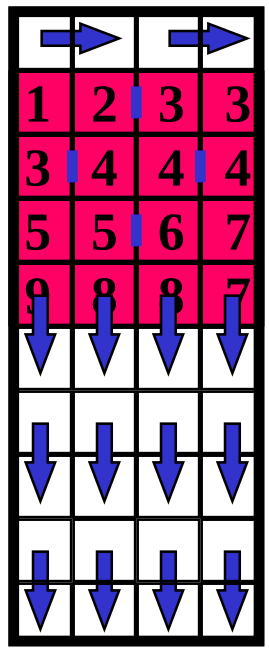
systolic merge



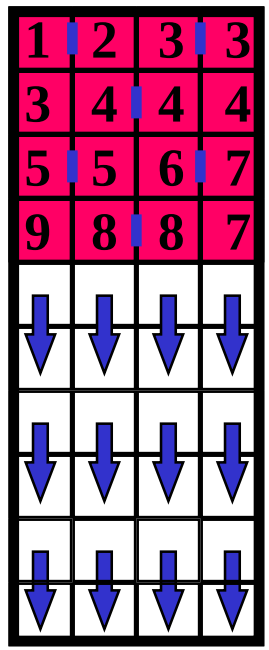
systolic merge



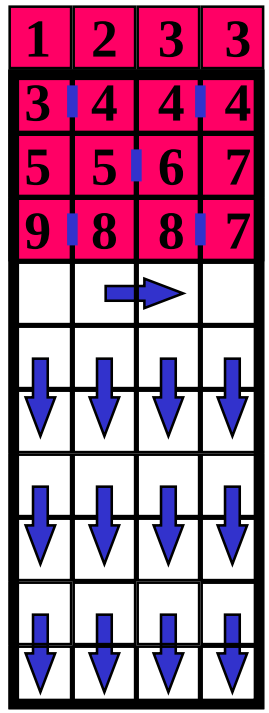
systolic merge



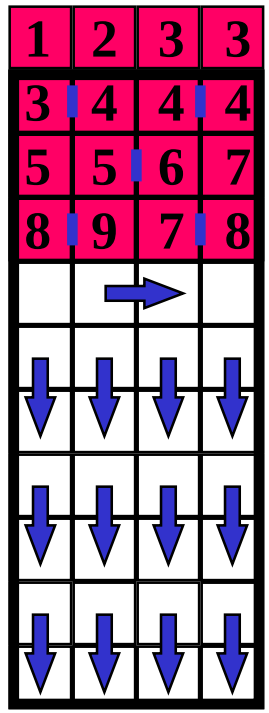
systolic merge



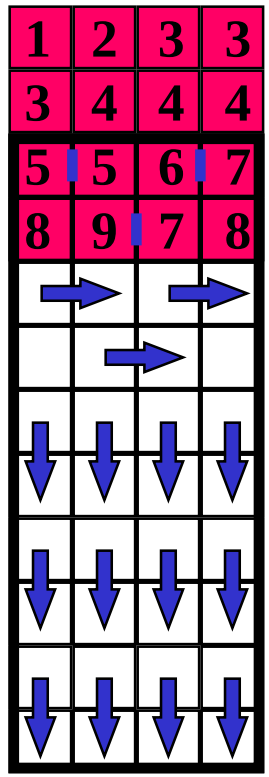
systolic merge



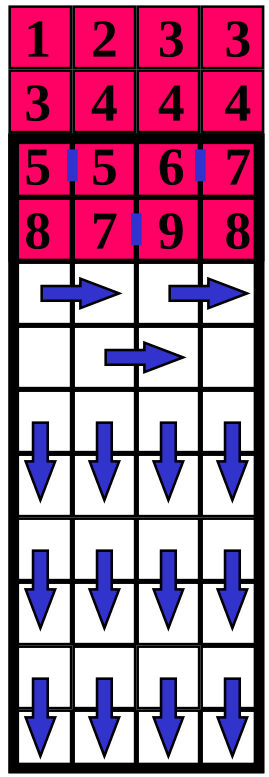
systolic merge



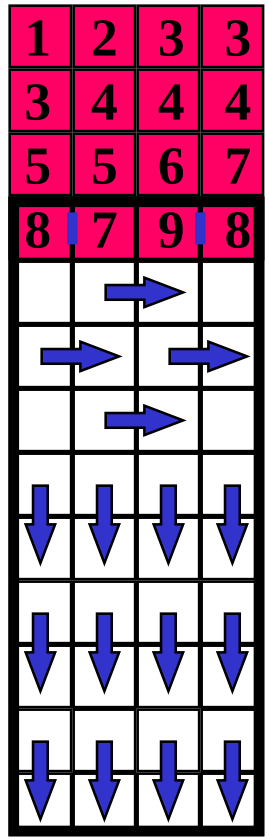
systolic merge



systolic merge

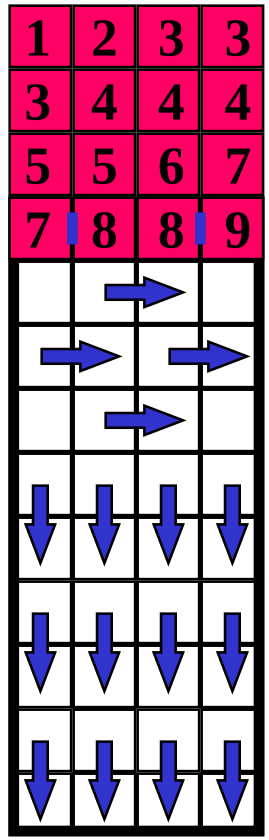


systolic merge



systolic merge

- **sorted !!!**



Problems to think about

- 1. Compare these designs with sorters shown earlier in class
- 2. Use these ideas to redesign sorters/absorbers and sorters that remove repeated elements.
- 3. Think about other applications of the ideas shown here for sort and merge.

Applications of Systolic Array

- **Signal and image processing:**

•FTR , IIR filtering , and 1-D convolution.

•2-D convolution and correlation.

•Discrete Furier transform

•Interpolation

•1-D and 2-D median filtering

•Geometric warping

Applications of Systolic Array

- Matrix arithmetic:

- Matrix-vector multiplication
- Matrix-matrix multiplication
- Matrix triangularization

  (solution of linear systems , matrix inversion)
- QR decomposition

  (eigenvalue , least-square computation)
- Solution of triangular linear systems

Applications of Systolic Array

- Non-numeric applications:

- Data structure
- Graph algorithm
- Language recognition
- Dynamic programming
- Encoder (polynomial division)
- Relational data-base operations