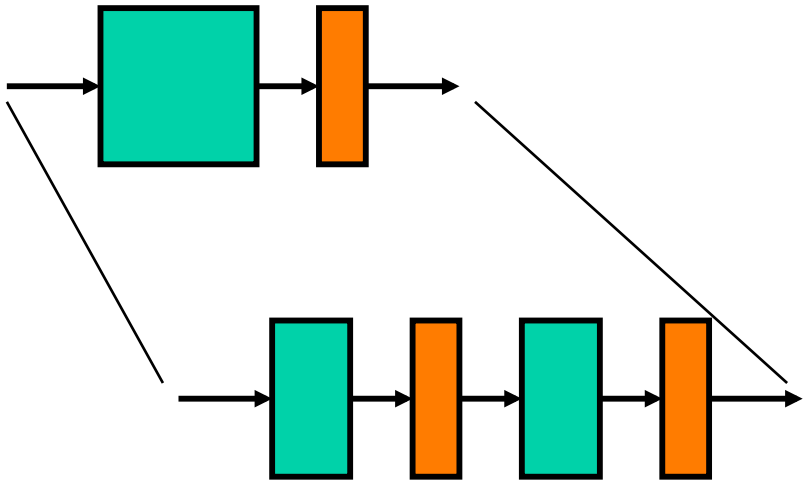
Review

- Placement – Greedy vs. Simulated Annealing
- Routing – Shortest Path (Djkstra), A*, Pathfinder
- Transmission lines – high frequency loss, dispersion, ringing, overshoot
- Differential signaling – advantages, eye diagrams
- Crosstalk – inductive, capacitive

Pipelining

- Adding registers along a path
  - split combinational logic into multiple cycles
  - each cycle smaller than previously
  - increase throughput



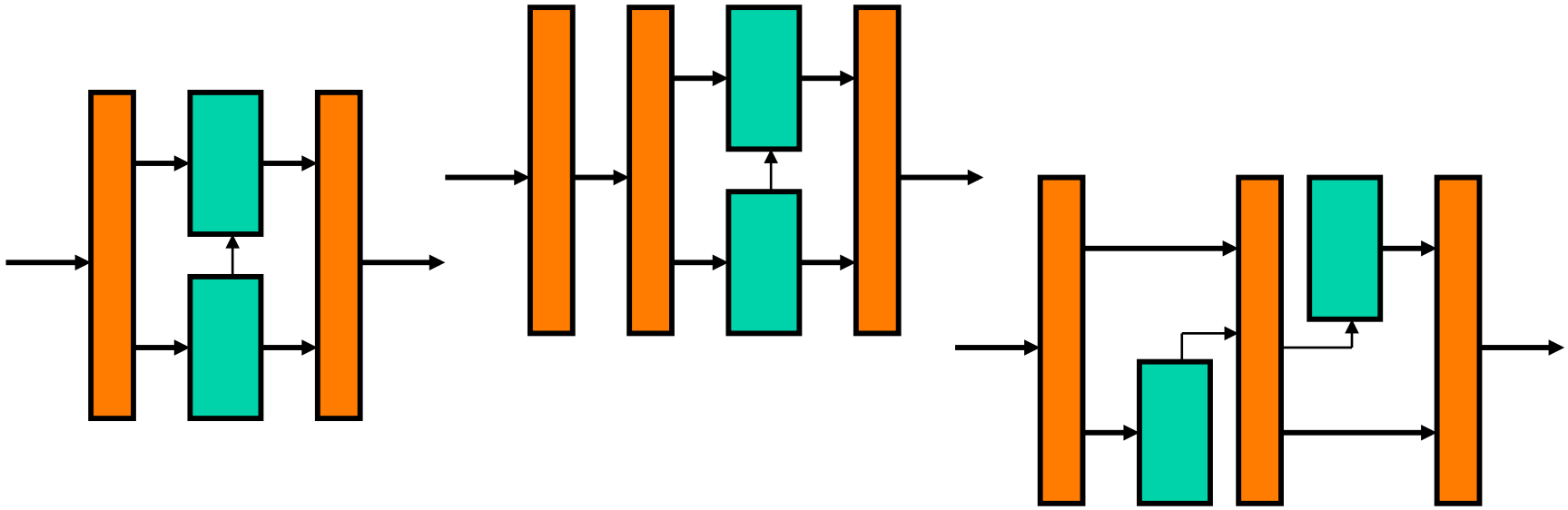
Pipelining

- Delay, d, of slowest combinational stage determines performance
- Throughput = 1/d : rate at which outputs are produced
- Latency = n•d : number of stages * clock period
- Pipelining increases circuit utilization
- Registers slow down data, synchronize data paths

- Wave-pipelining
  - no pipeline registers - waves of data flow through circuit
  - relies on equal-delay circuit paths - no short paths

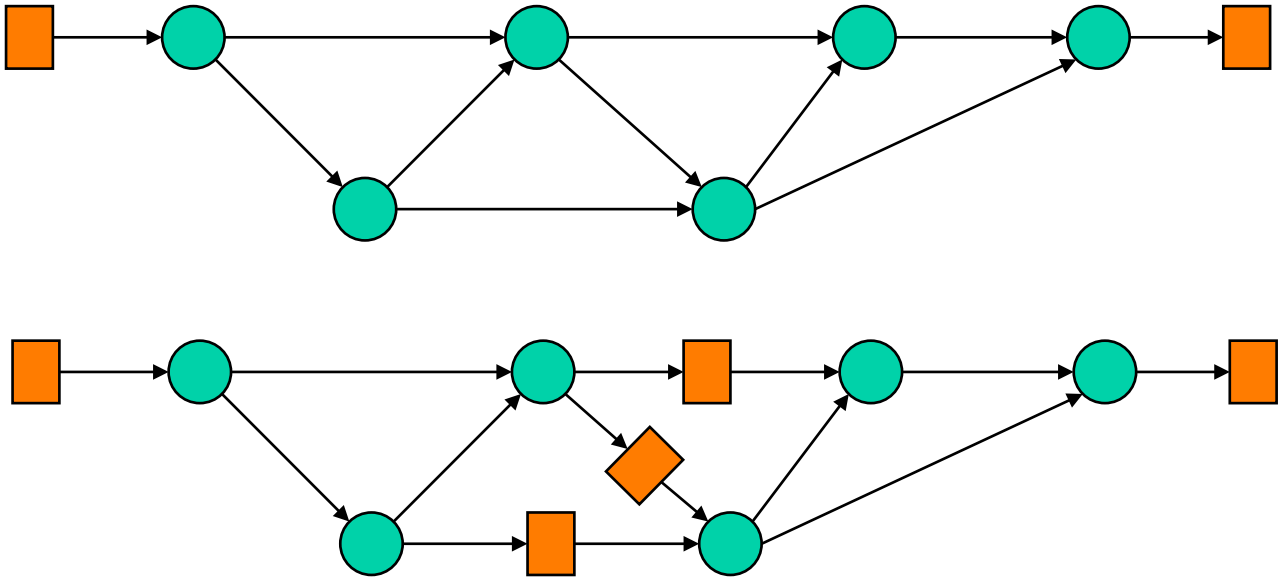
When and How to Pipeline?

- Where is the best place to add registers?
  - splitting combinational logic
  - overhead of registers (propagation delay and setup time requirements)
- What about cycles in data path?
- Example: 16-bit adder, add 8-bits in each of two cycles



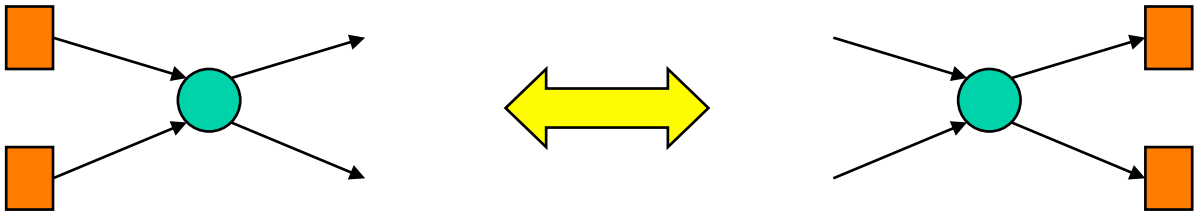
Retiming

- Process of optimally distributing registers throughout a circuit
  - minimize the clock period
  - minimize the number of registers



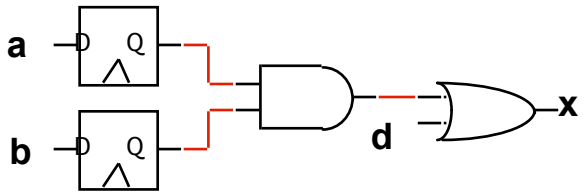
Retiming (cont'd)

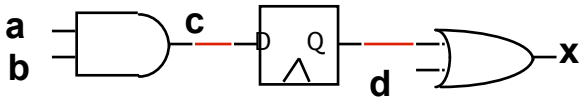
- Fast optimal algorithm (Leiserson & Saxe 1983)
- Retiming rules:
  - remove one register from each input and add one to each output
  - remove one register from each output and add one to each input



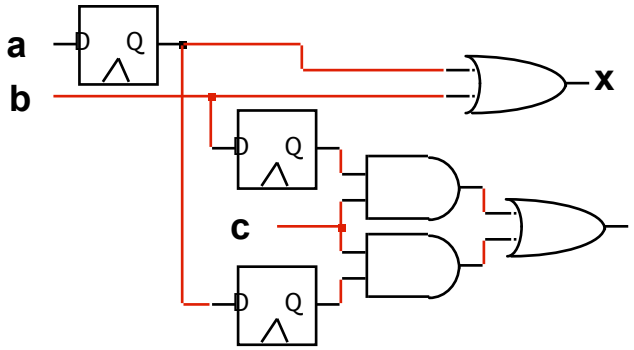
Retiming examples

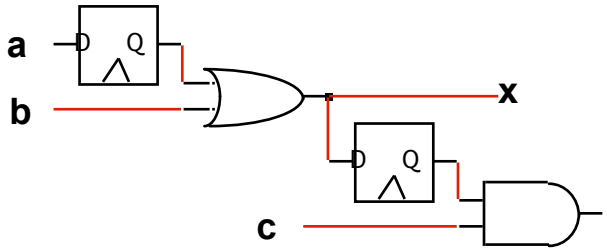
- Shortening critical paths





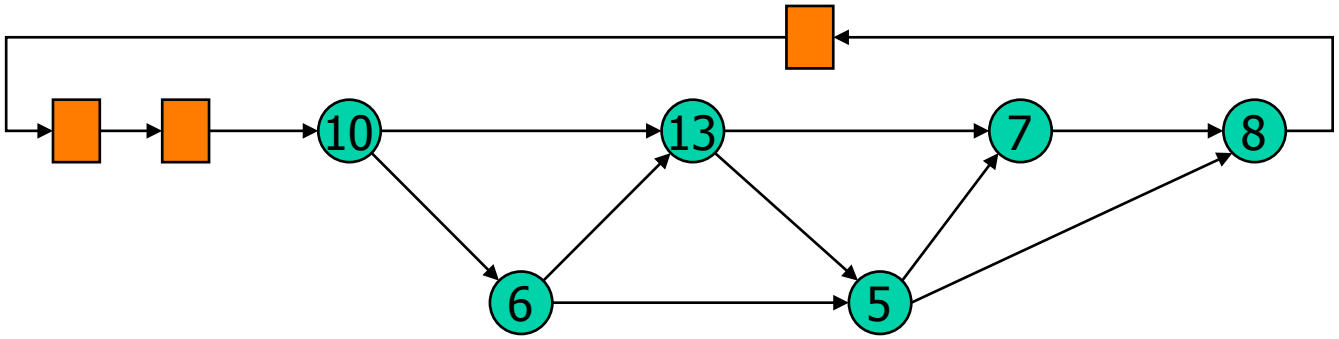
- Create simplification opportunities





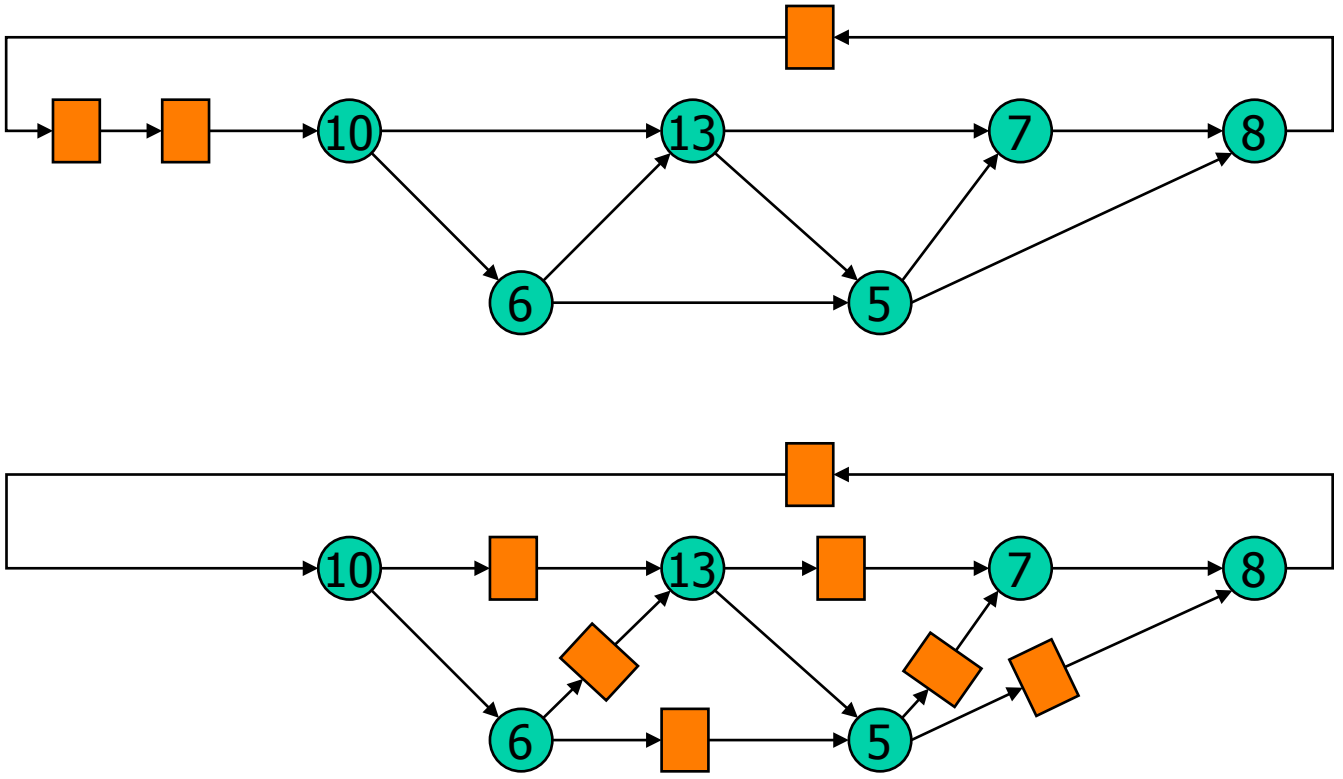
Optimal Pipelining

- Add registers - use retiming to find optimal location



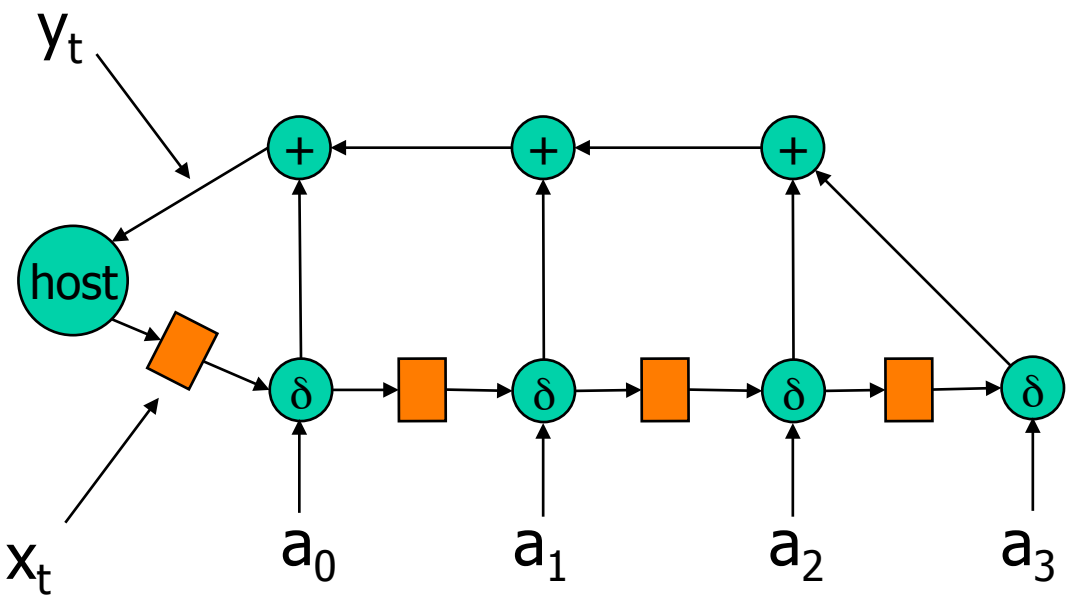
Optimal Pipelining

- Add registers - use retiming to find optimal location



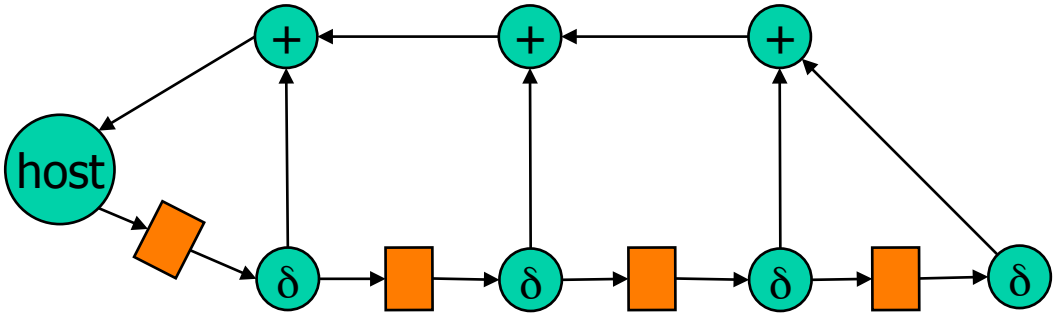
Example - Digital Correlator

- $y_t = \delta(x_t, a_0) + \delta(x_{t-1}, a_1) + \delta(x_{t-2}, a_2) + \delta(x_{t-3}, a_3)$
- $\delta(x_t, a_0) = 0$ if $x \neq a$, 1 otherwise (and passes x along to the right)



Example - Digital Correlator (cont'd)

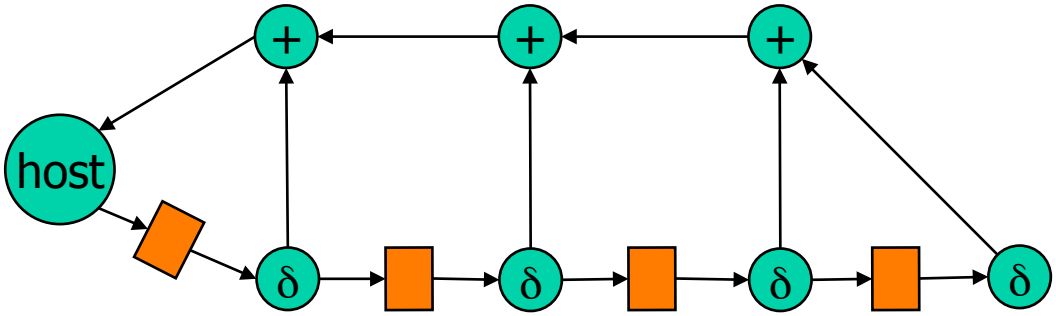
- Delays: adder, 7; comparator, 3; host, 0



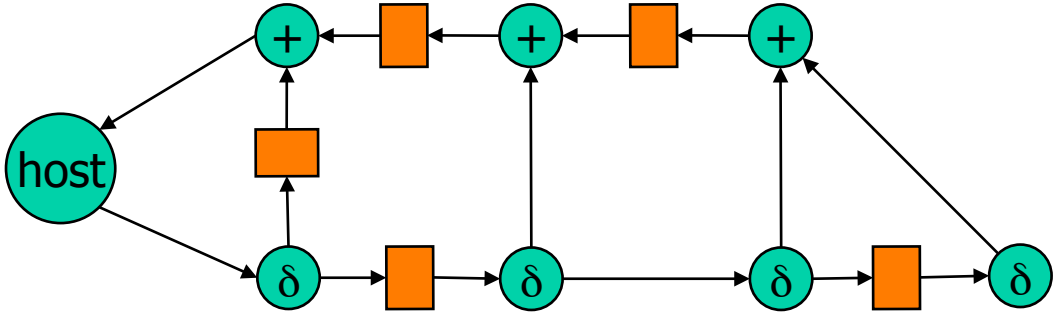
cycle time =

Example - Digital Correlator (cont'd)

- Delays: adder, 7; comparator, 3; host, 0



cycle time = 24



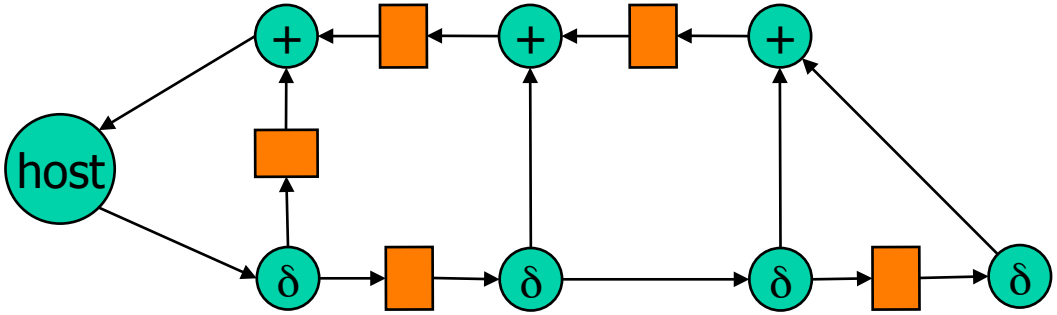
cycle time = 13

Extensions to Retiming

- Host interface
  - add latency
  - multiple hosts
- Area considerations
  - limit number of registers
  - optimize logic across register boundaries
    - peripheral retiming
    - incremental retiming
    - pre-computation
- Generality
  - different propagation delays for different signals
  - widths of interconnections

Digital Correlator Revisited

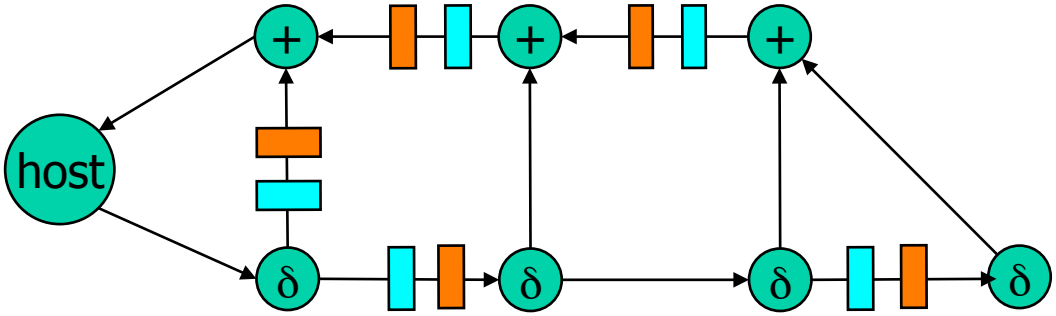
- Optimally retimed circuit (clock cycle 13)



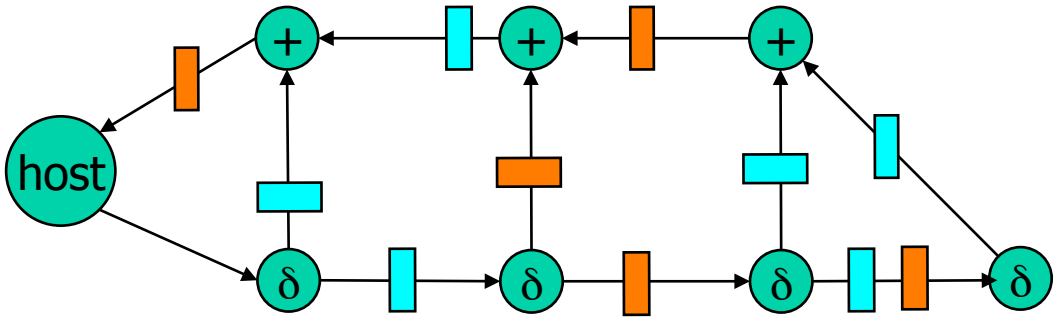
- How can we increase the clock frequency?
  - Work on multiple data sets at the same time

C-slow'ing a Circuit

- Replace every register with C registers

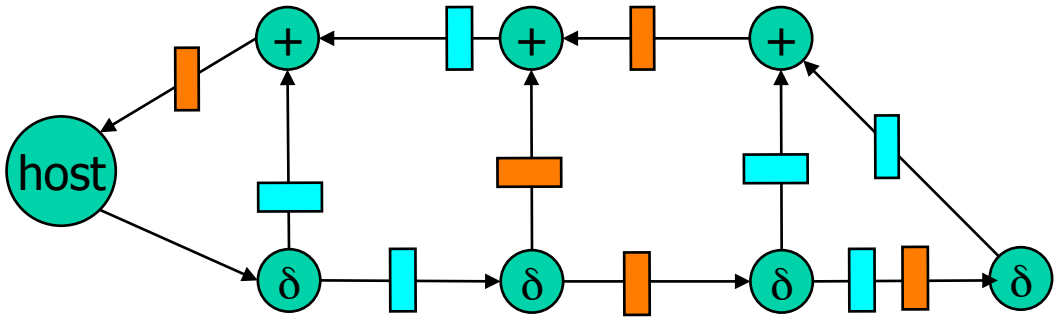


- Now retime: (clock cycle now 7)



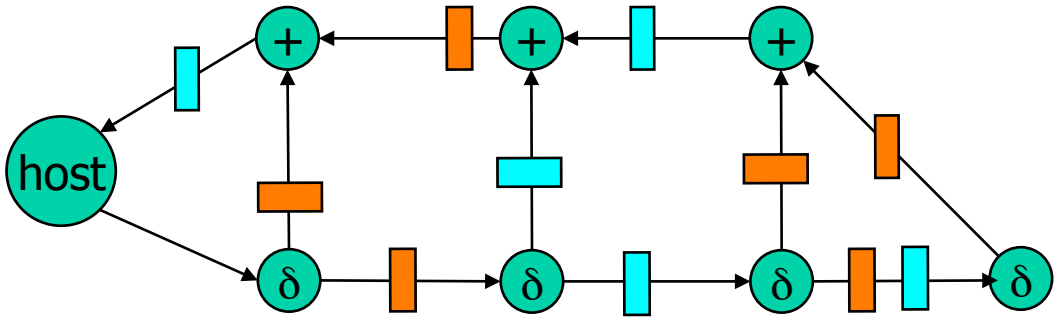
C-slow'ing a Circuit = Multi-Threading

- In this case there are two threads (blue and orange)
  - Host alternates between the two threads
- Input blue data, remove orange results



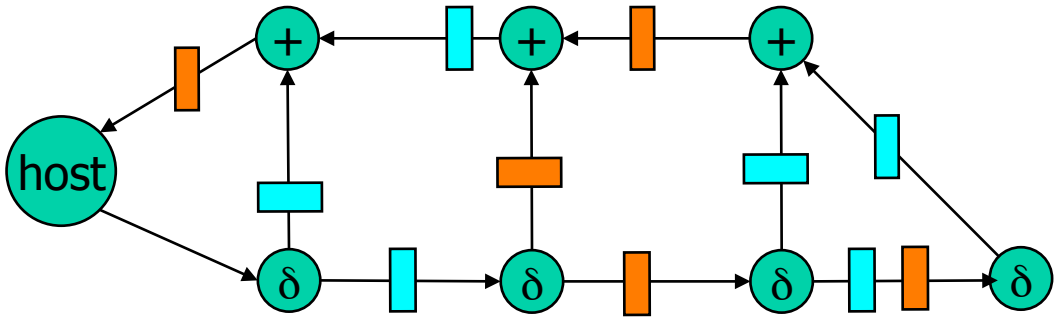
C-slow'ing a Circuit = Multi-Threading

- In this case there are two threads (blue and orange)
  - Host alternates between the two threads
- Input orange data, remove blue results



C-slow'ing a Circuit = Multi-Threading

- In this case there are two threads (blue and orange)
  - Host alternates between the two threads
- Input blue data, remove orange results
  - Throughput of each thread (1/14) is almost what it was before (1/13) !



Pipelining Processors

- Pipelined processors are complex because of feedback loops (cycles)
  - Forwarding
  - Branch prediction
  - Long latency ops (e.g. cache miss) cause stalls
- Solution – C-slowing!
  - Start with non-pipelined processor
    - Program counter, register file
  - C-slow by N
    - N program counters, register files
  - Pipeline datapath, ignoring hazards!

Multi-Threaded Processor

- Pipelined, c-slowed processor is "multi-threaded"
  - Executing N different instruction streams simultaneously
  - Each executes an instruction every N cycles
  - Allows op feedback latency of N cycles
    - e. g. cache read miss

- Tera Computer MTA (now Cray) is multi-threaded
  - N = 1024
  - There is no cache!
    - Remote memory access < 1024 cycles (2 usec)
- Requires huge parallelism (N threads!)

Multi-Threaded Processors

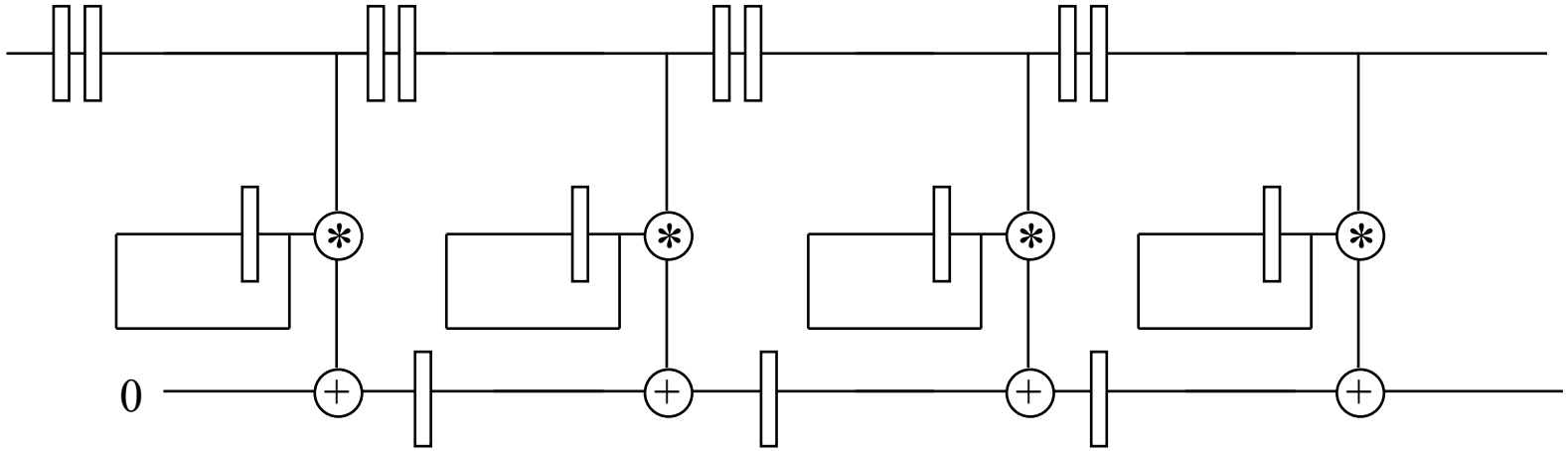
- Multi-threaded processors are very simple
  - No stalls, no forwarding, etc.
- Great for FPGAs
  - Muxes are expensive and slow!
  - Registers are almost free
- Typical N = 4 for multi-threaded FPGA processors
  - Almost 4x performance increase
  - Higher utilization (no stalls) (2x)
  - Higher frequency (2x)

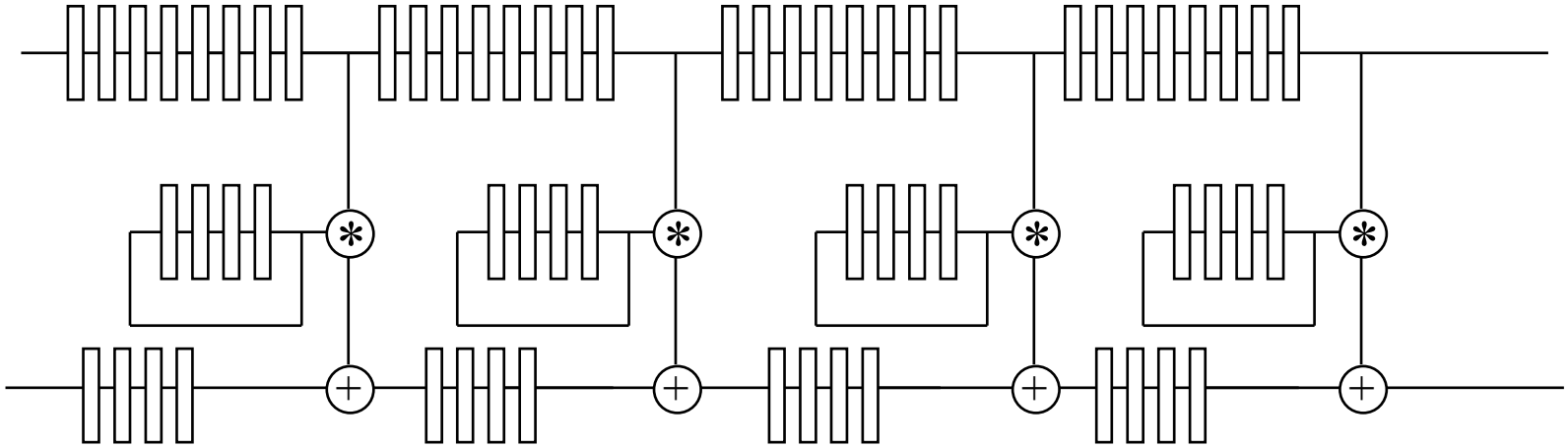
SMT Processors

- Dynamic multi-threaded, super-scalar, out-of-order processors
  - Hugely complicated
- Take CSE 471 !!

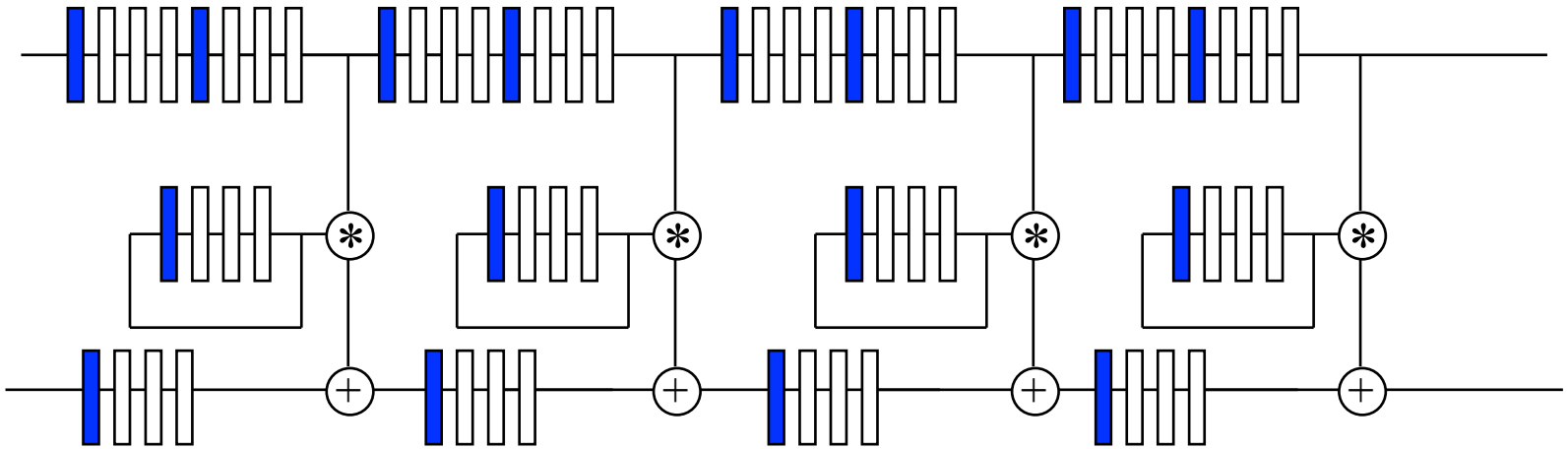
C-slowing/Retiming for Resource Sharing

- Correlator circuit

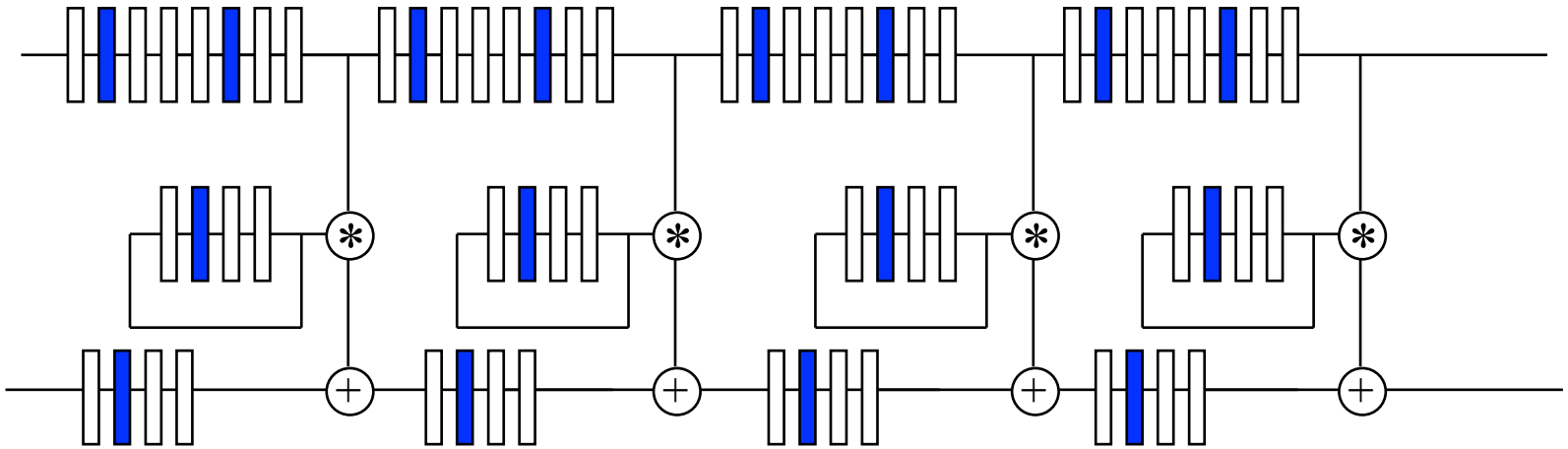


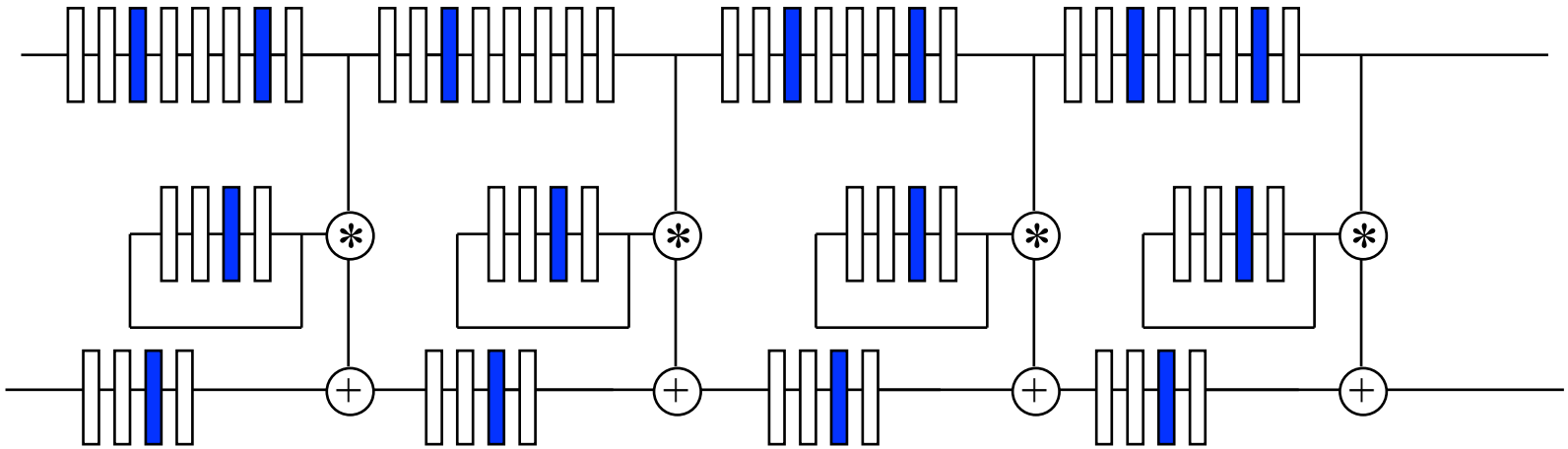


C-slowed by 4

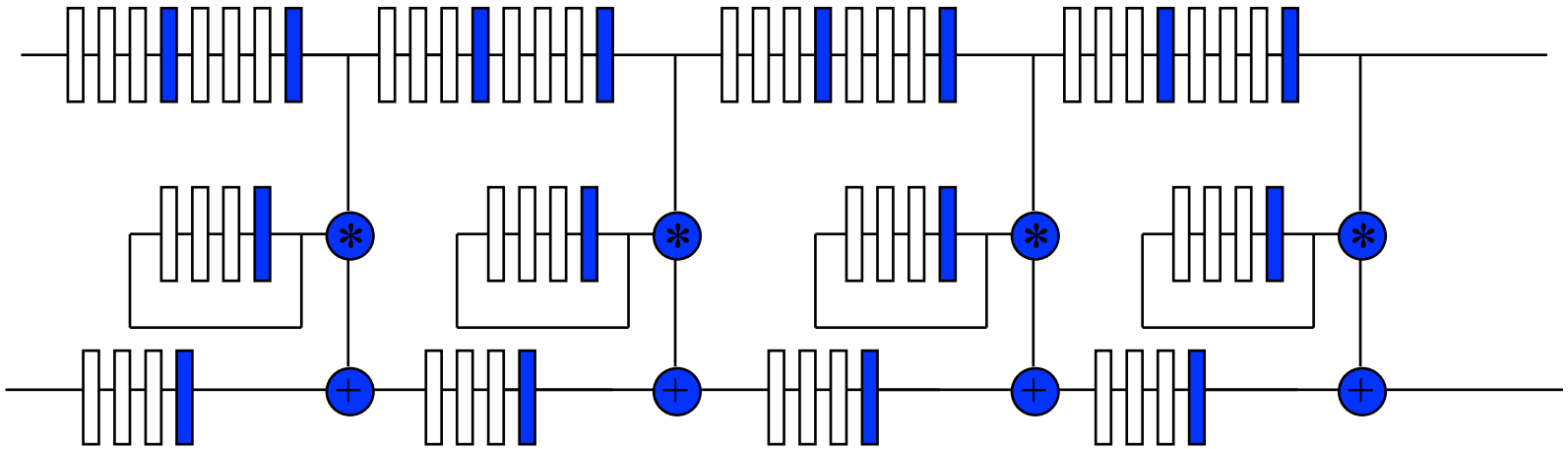


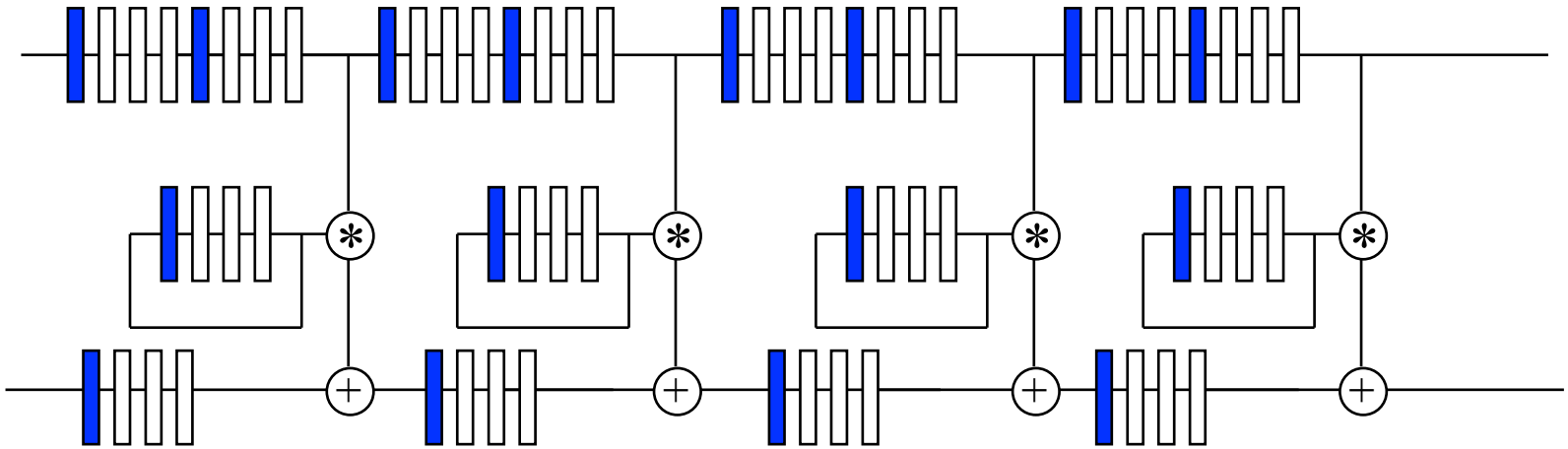
Insert Data every 4 cycles (one data set)

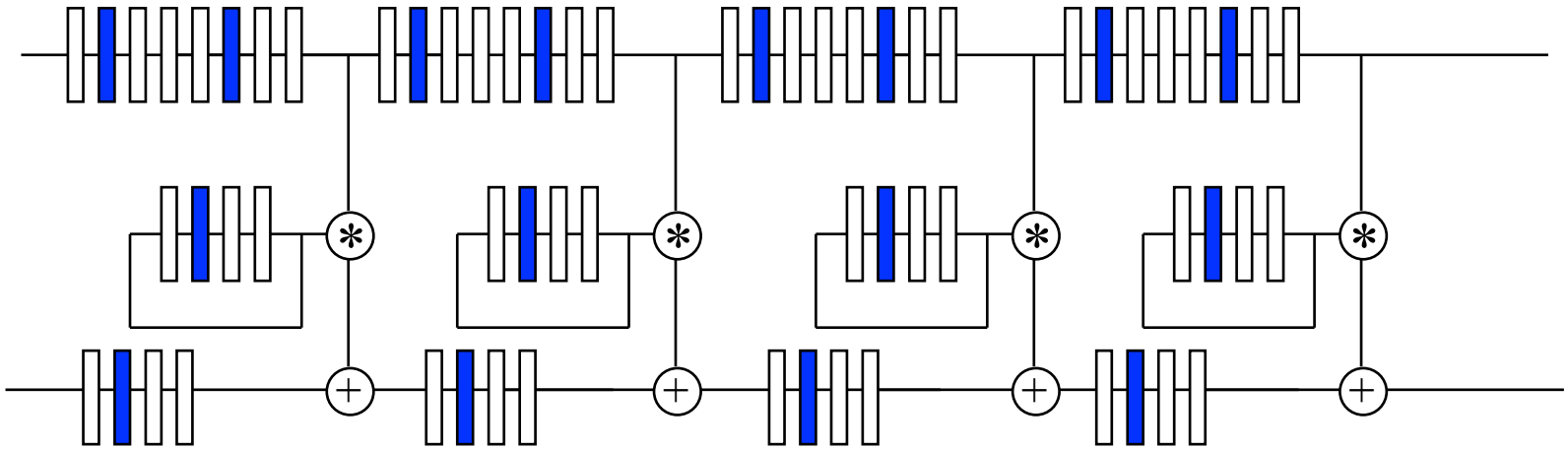


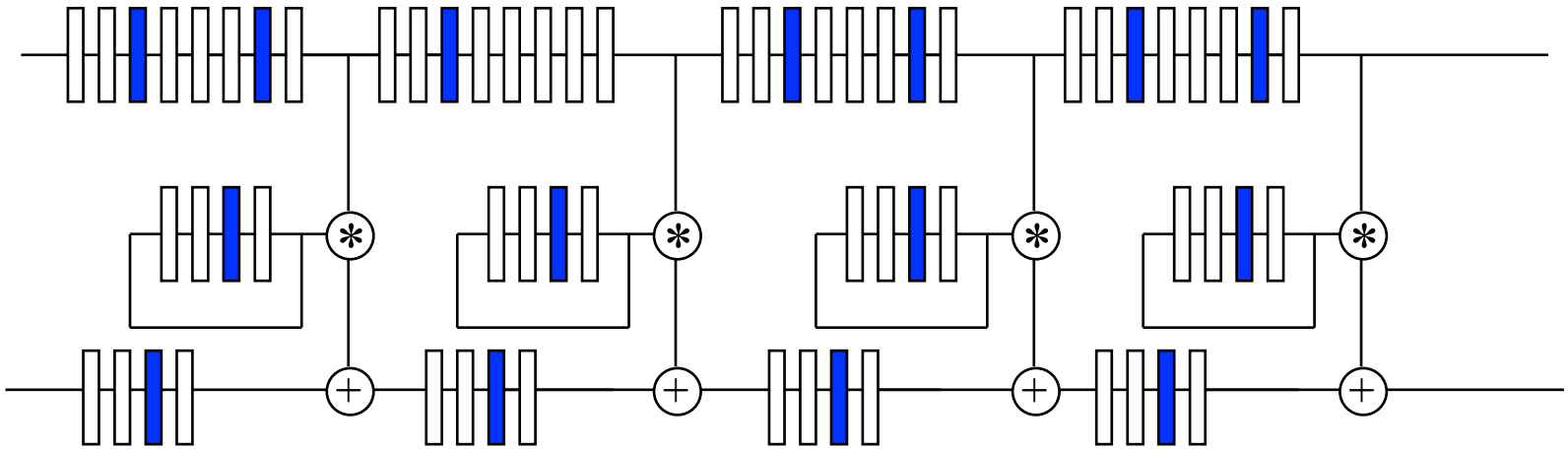


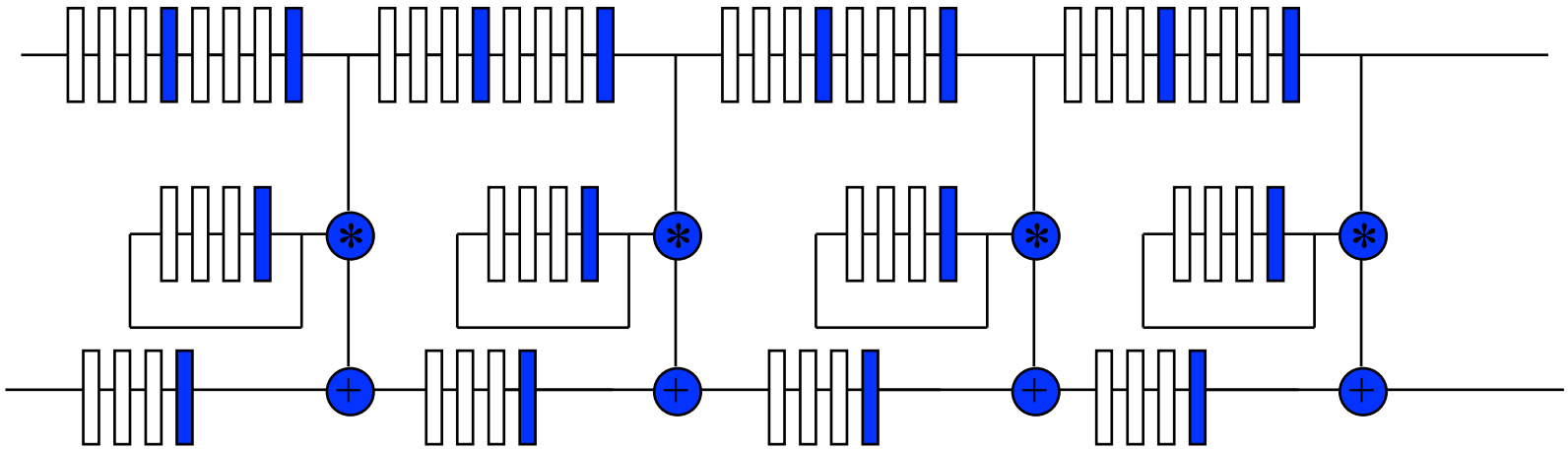
Computation Active only every 4 Cycles



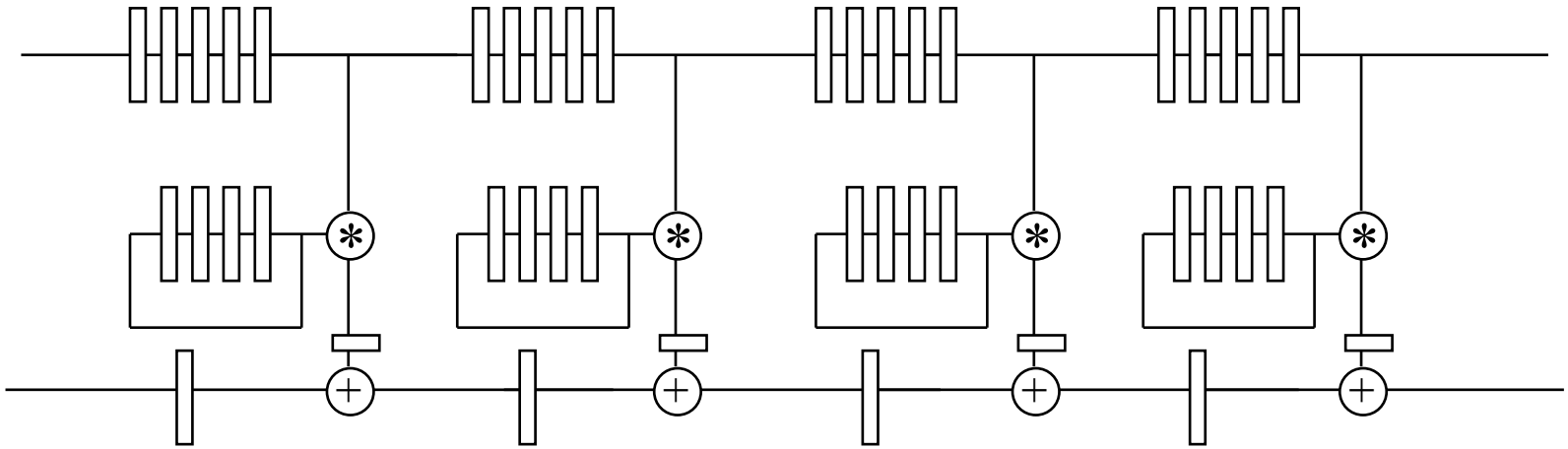


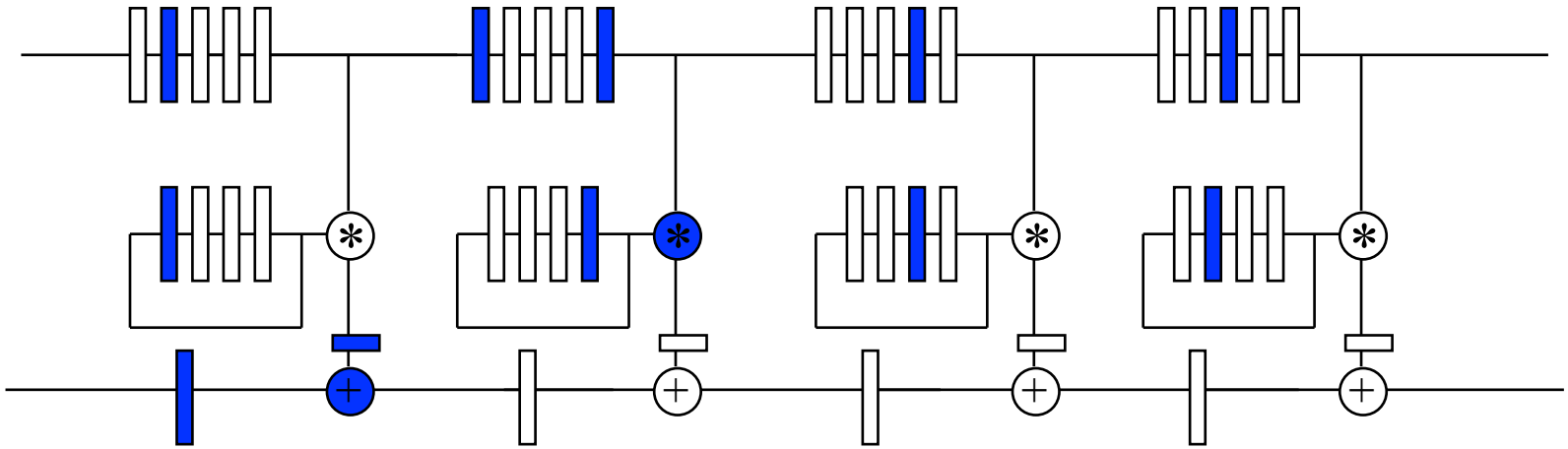


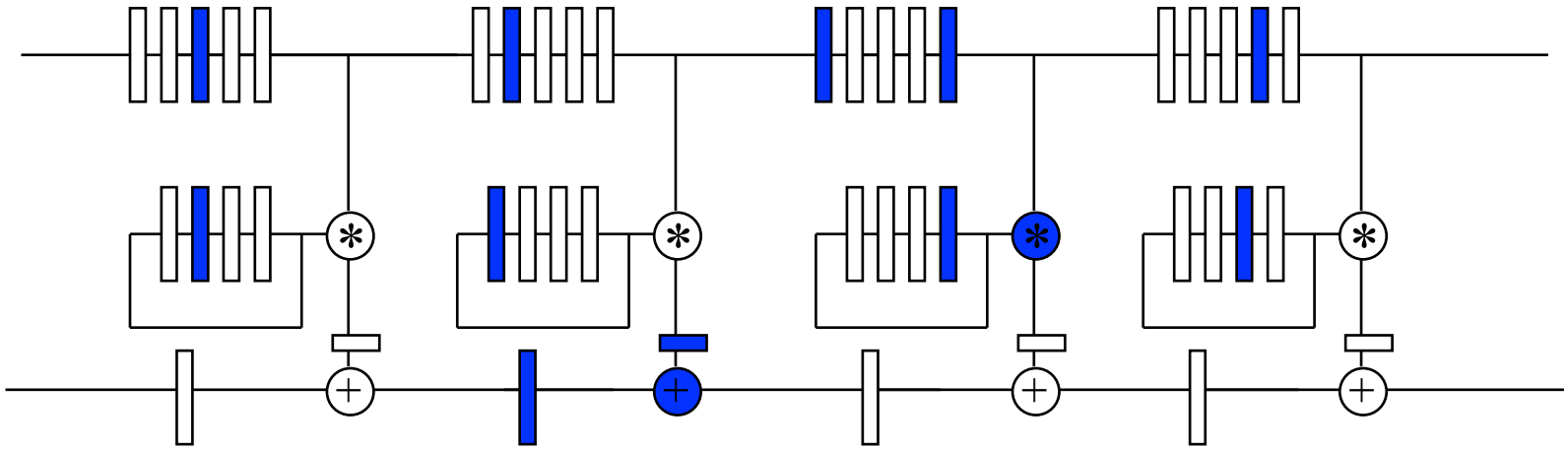


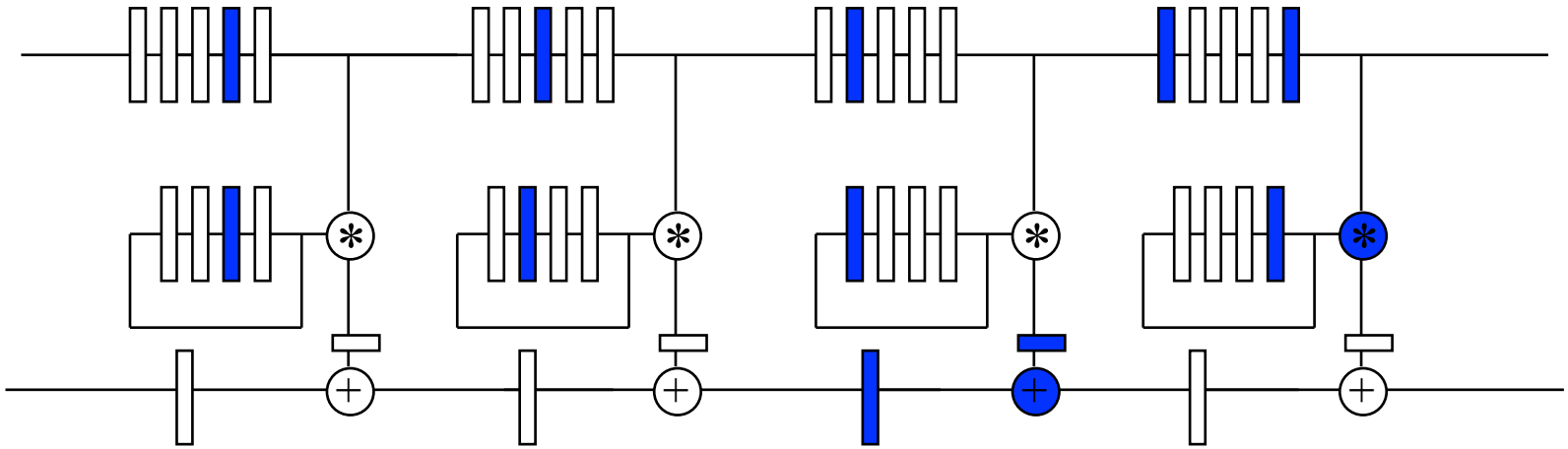


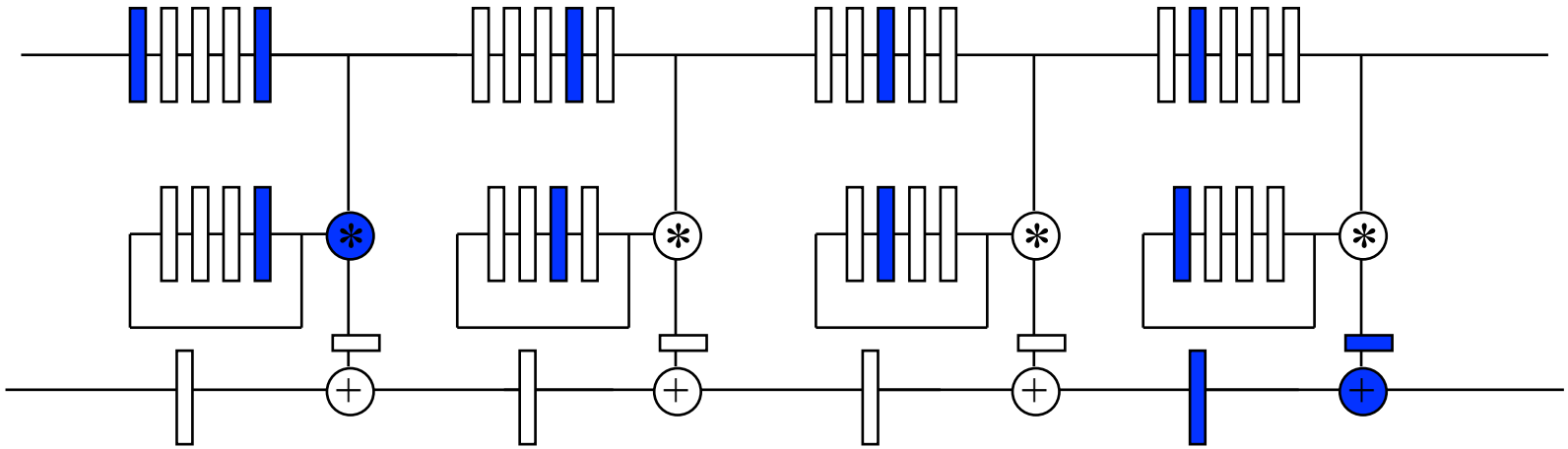
Retime and remove extra Pipelining

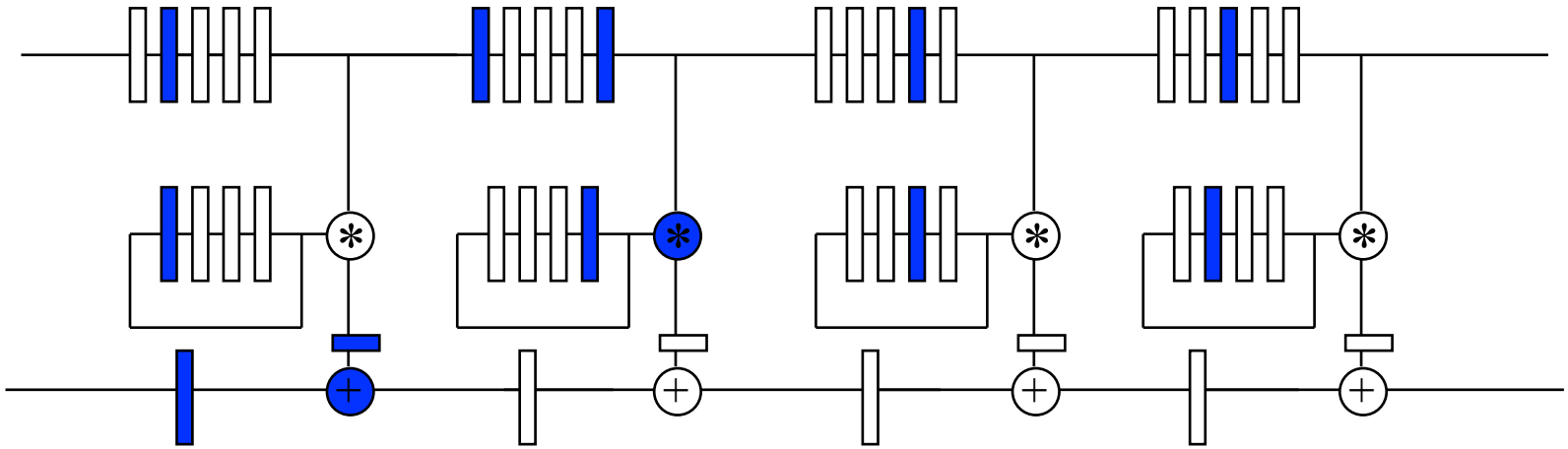


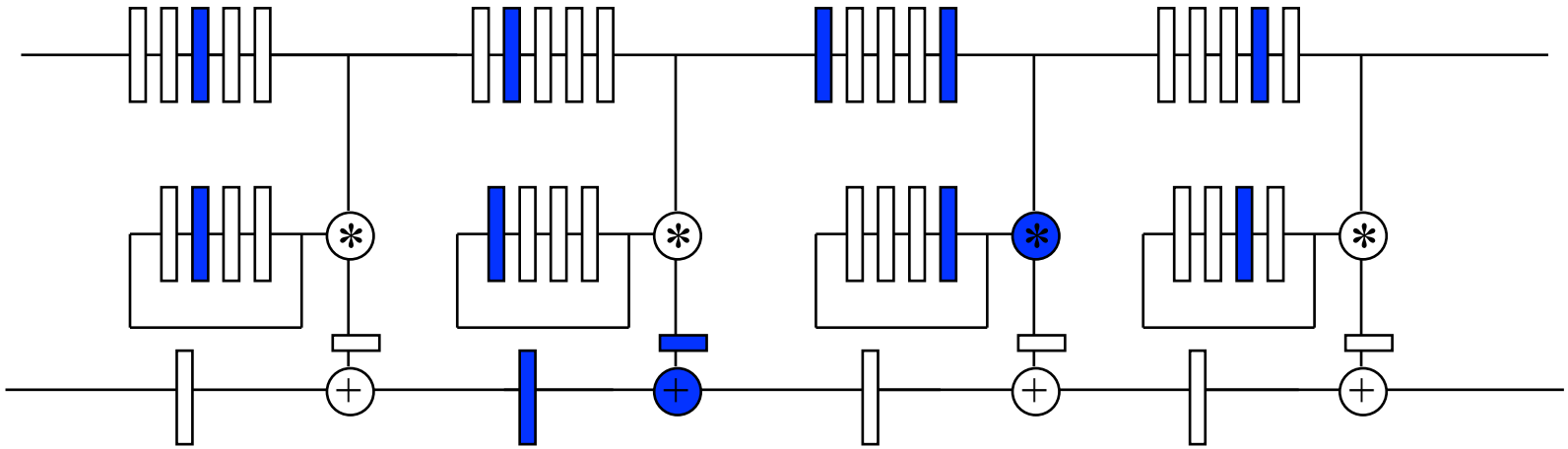


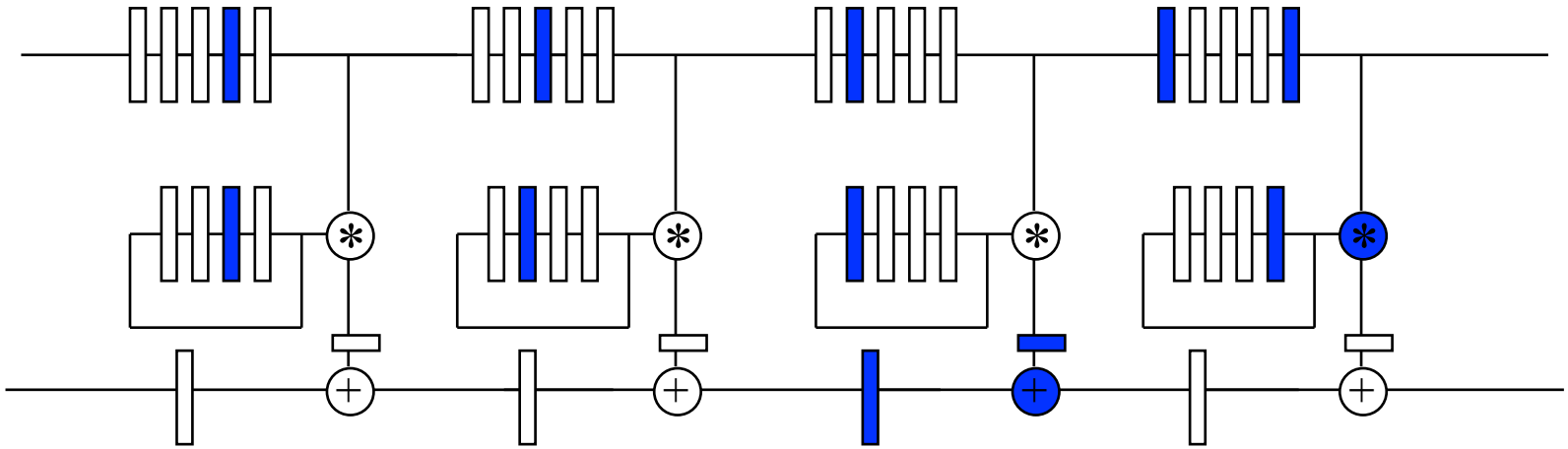






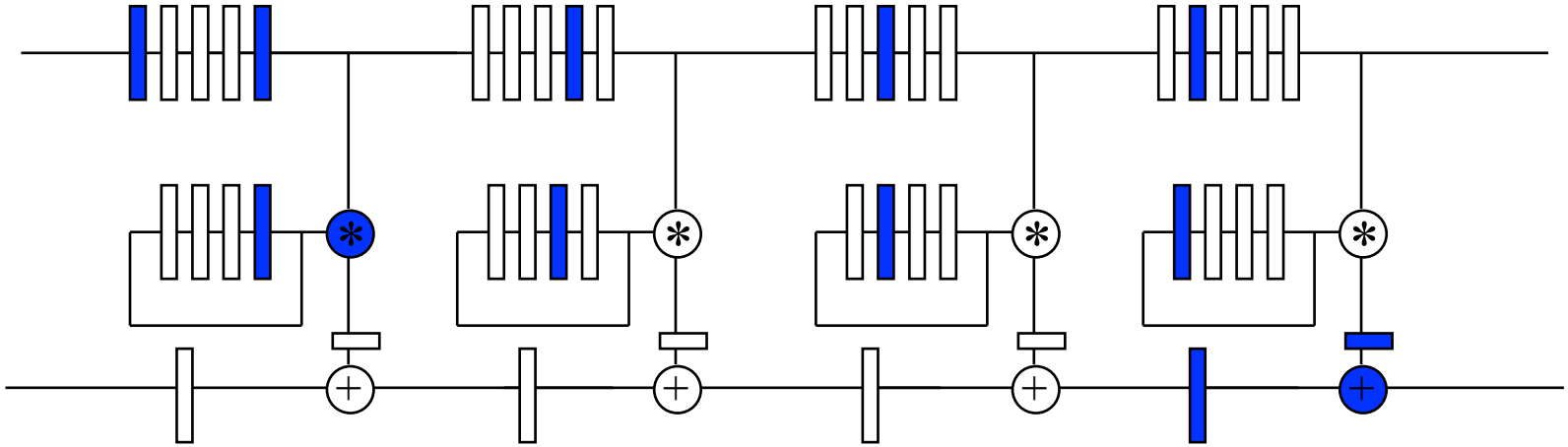






Computation spread over time

- Only need one multiplier and one adder
- We can use this method to schedule for any number of resources



Systolic Arrays

- Set of identical processing elements
  - specialized or programmable
- Efficient nearest-neighbor interconnections (in 1-D, 2-D, other)
- SIMD-like
- Multiple data flows, converging to engage in computation

Analogy: data flowing through the system in a rhythmic fashion – from main memory through a series of processing elements and back to main memory

Example - Convolution

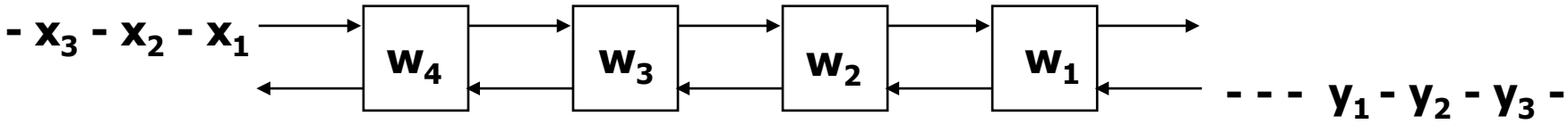
- $y_j = x_j w_1 + x_{j+1} w_2 + \ldots + x_{j+n-1} w_n$

$$y_1 = x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4$$

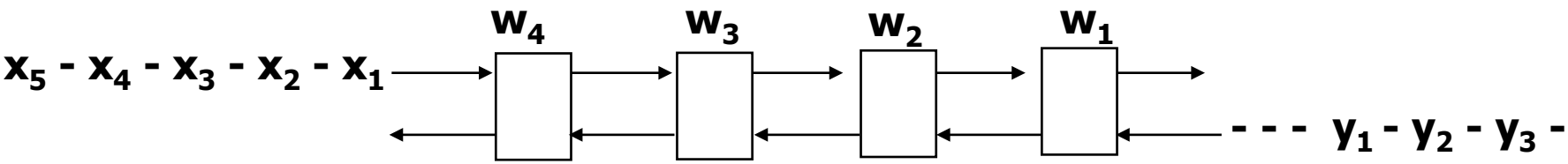
$$y_2 = x_2 w_1 + x_3 w_2 + x_4 w_3 + x_5 w_4$$

$$y_3 = x_3 w_1 + x_4 w_2 + x_5 w_3 + x_6 w_4$$

$$\ldots$$



Example - Convolution

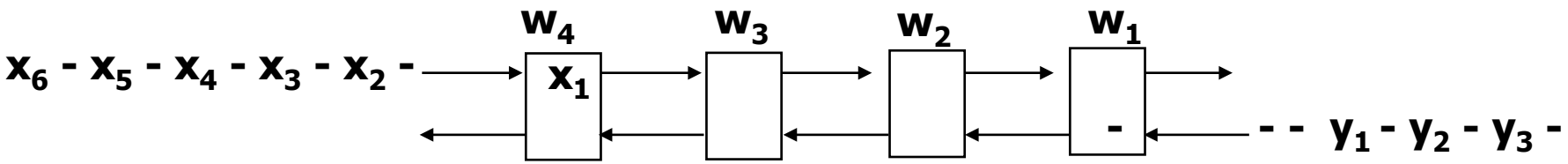


$$y_1 = x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4$$

$$y_2 = x_2w_1 + x_3w_2 + x_4w_3 + x_5w_4$$

$$y_3 = x_3w_1 + x_4w_2 + x_5w_3 + x_6w_4$$

Example - Convolution

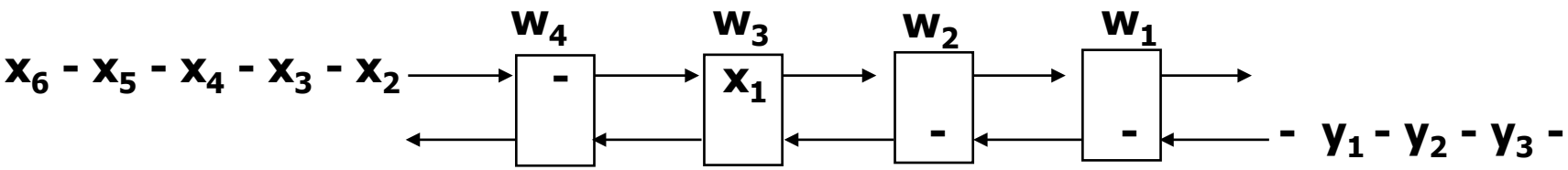


$$y_1 = x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4$$

$$y_2 = x_2w_1 + x_3w_2 + x_4w_3 + x_5w_4$$

$$y_3 = x_3w_1 + x_4w_2 + x_5w_3 + x_6w_4$$

Example - Convolution

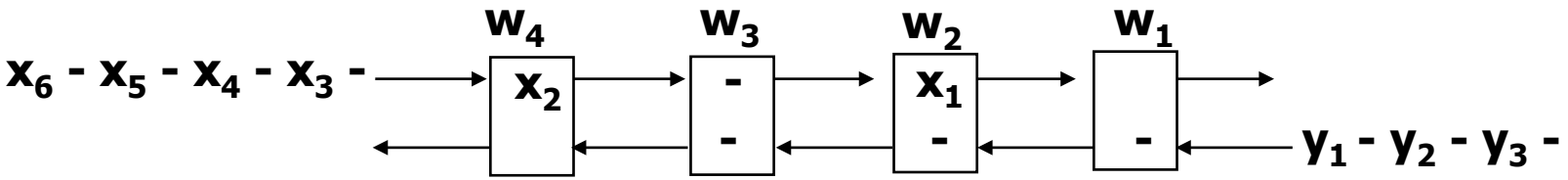


$$y_1 = x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4$$

$$y_2 = x_2w_1 + x_3w_2 + x_4w_3 + x_5w_4$$

$$y_3 = x_3w_1 + x_4w_2 + x_5w_3 + x_6w_4$$

Example - Convolution

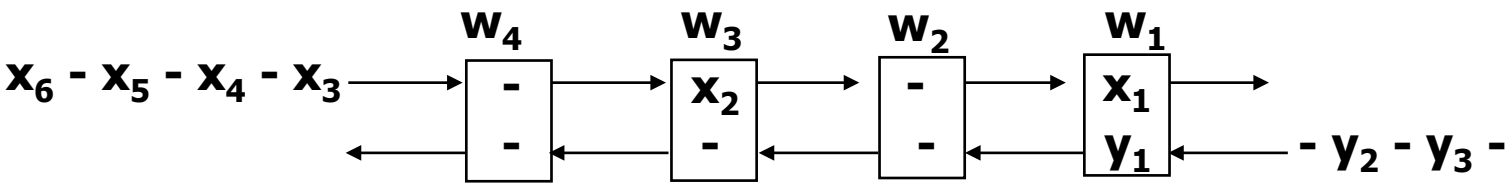


$$y_1 = x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4$$

$$y_2 = x_2w_1 + x_3w_2 + x_4w_3 + x_5w_4$$

$$y_3 = x_3w_1 + x_4w_2 + x_5w_3 + x_6w_4$$

Example - Convolution

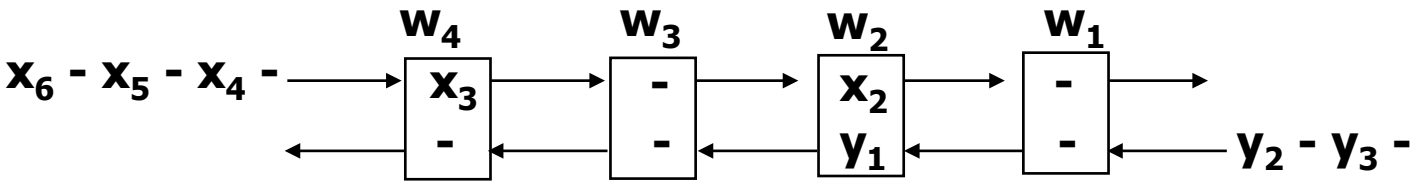


$$y_1 = x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4$$

$$y_2 = x_2w_1 + x_3w_2 + x_4w_3 + x_5w_4$$

$$y_3 = x_3w_1 + x_4w_2 + x_5w_3 + x_6w_4$$

Example - Convolution

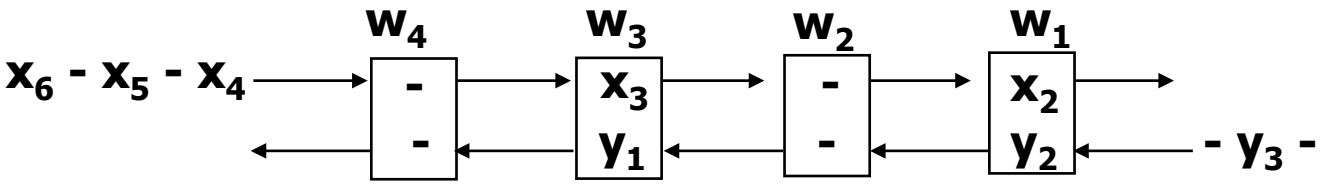


$$y_1 = x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4$$

$$y_2 = x_2w_1 + x_3w_2 + x_4w_3 + x_5w_4$$

$$y_3 = x_3w_1 + x_4w_2 + x_5w_3 + x_6w_4$$

Example - Convolution

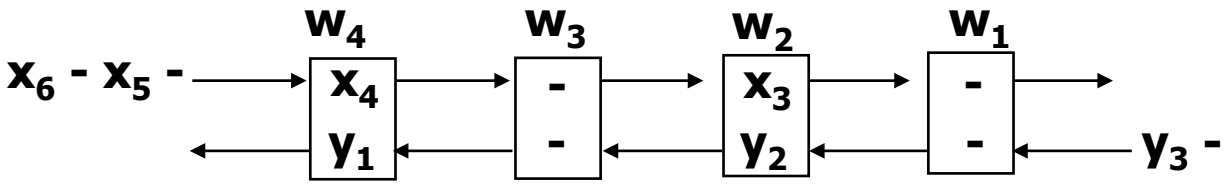


$$y_1 = x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4$$

$$y_2 = x_2w_1 + x_3w_2 + x_4w_3 + x_5w_4$$

$$y_3 = x_3w_1 + x_4w_2 + x_5w_3 + x_6w_4$$

Example - Convolution

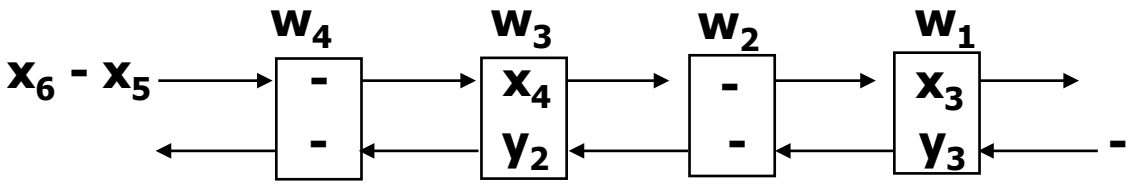


$$y_1 = x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4$$

$$y_2 = x_2w_1 + x_3w_2 + x_4w_3 + x_5w_4$$

$$y_3 = x_3w_1 + x_4w_2 + x_5w_3 + x_6w_4$$

Example - Convolution

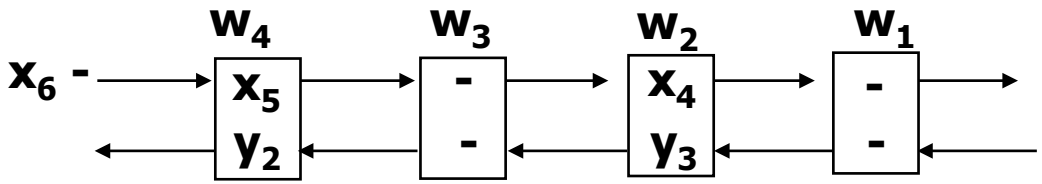


$$y_1 = x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4$$

$$y_2 = x_2w_1 + x_3w_2 + x_4w_3 + x_5w_4$$

$$y_3 = x_3w_1 + x_4w_2 + x_5w_3 + x_6w_4$$

Example - Convolution



$$y_1 = x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4$$

$$y_2 = x_2w_1 + x_3w_2 + x_4w_3 + x_5w_4$$

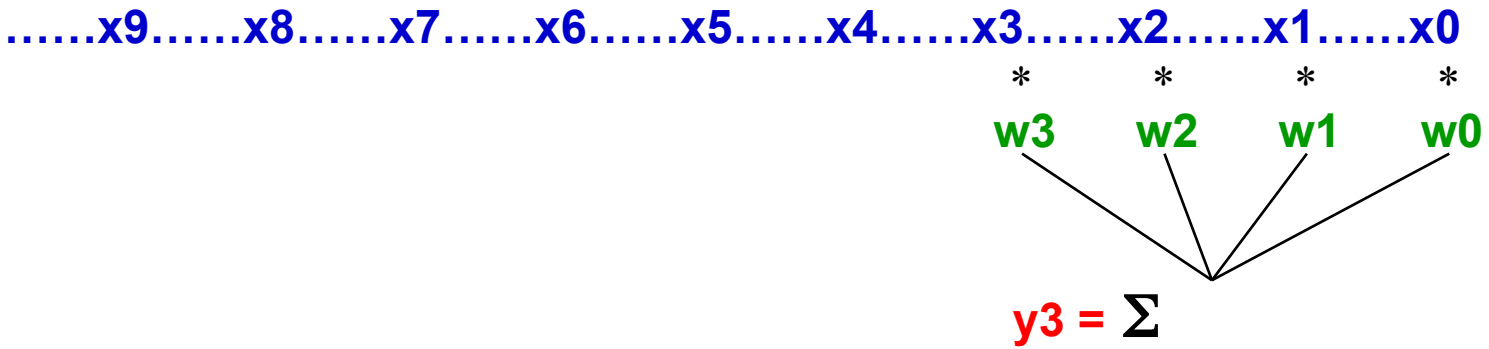
$$y_3 = x_3w_1 + x_4w_2 + x_5w_3 + x_6w_4$$

Example - Convolution

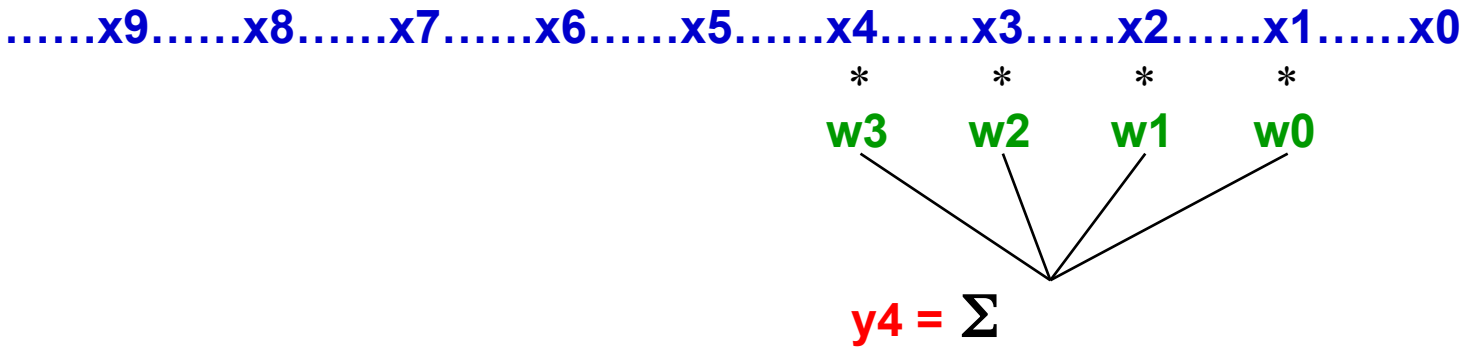
	w_4	w_3	w_2	w_1	
$x_6 - x_5 - x_4 - x_3 - x_2 -$	x_1			$-$	$- - y_1 - y_2 - y_3$
$x_6 - x_5 - x_4 - x_3 - x_2$	$-$	x_1			
			$-$	$-$	$- y_1 - y_2 - y_3$
$x_6 - x_5 - x_4 - x_3 -$	x_2	$-$	x_1		
		$-$	$-$	$-$	$y_1 - y_2 - y_3$
$x_6 - x_5 - x_4 - x_3$	$-$	x_2	$-$	x_1	
	$-$	$-$	$-$	y_1	$- y_2 - y_3$
$x_6 - x_5 - x_4 -$	x_3	$-$	x_2	$-$	
	$-$	$-$	y_1	$-$	$y_2 - y_3$
$x_6 - x_5 - x_4$	$-$	x_3	$-$	x_2	
	$-$	y_1	$-$	y_2	$- y_3$
$x_6 - x_5 -$	x_4	$-$	x_3	$-$	
	y_1	$-$	y_2	$-$	y_3
$x_6 - x_5$	$-$	x_4	$-$	x_3	
	$-$	y_2	$-$	y_3	

Convolution - Another Look

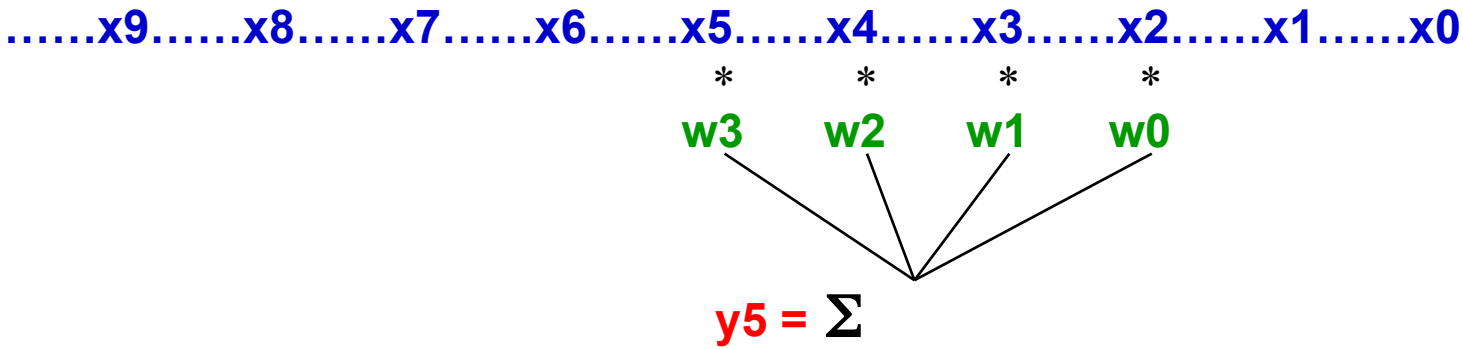
- Repeated vector product



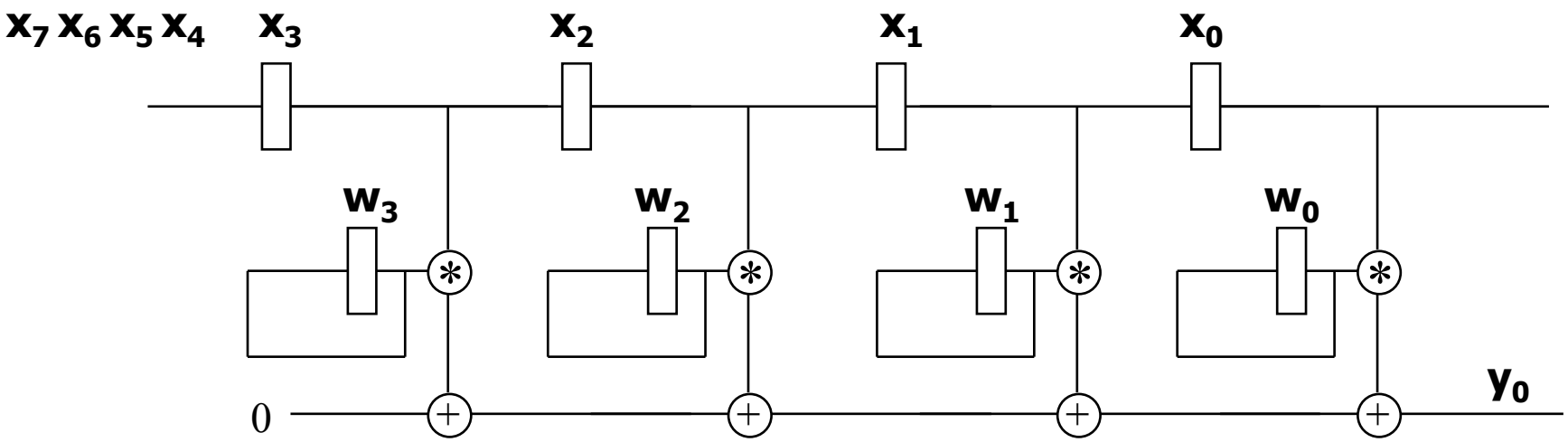
Example: Convolution



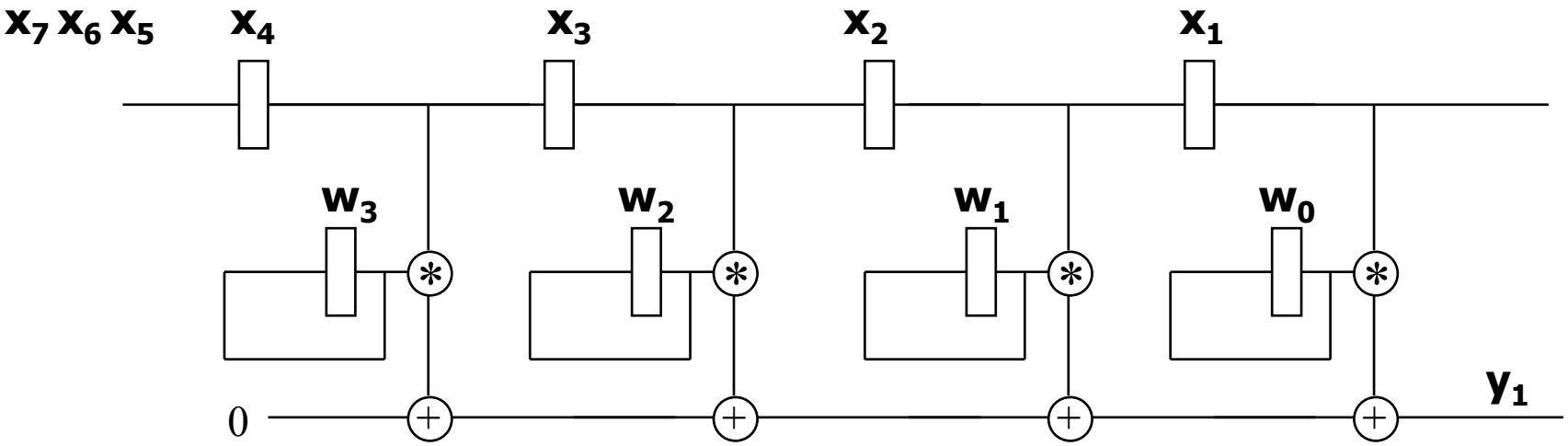
Example: Convolution



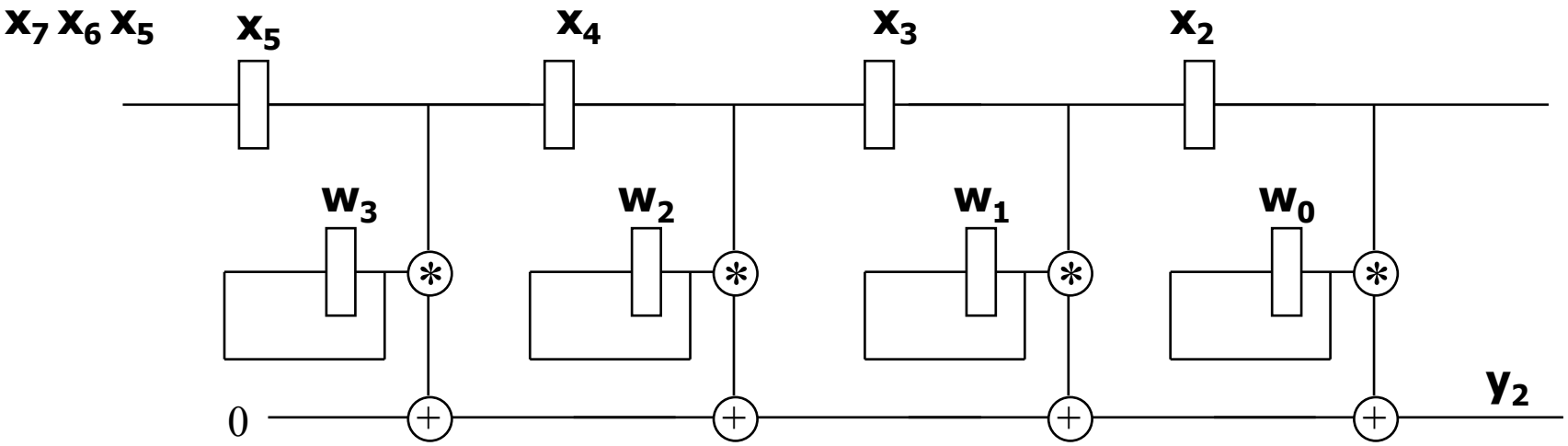
Convolution Example

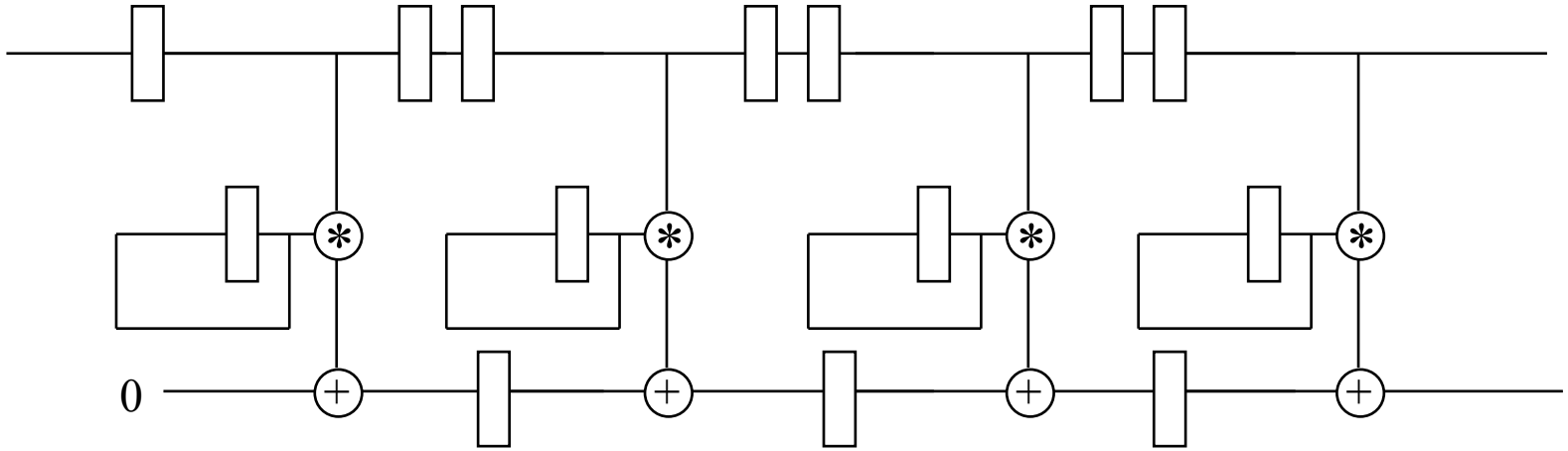


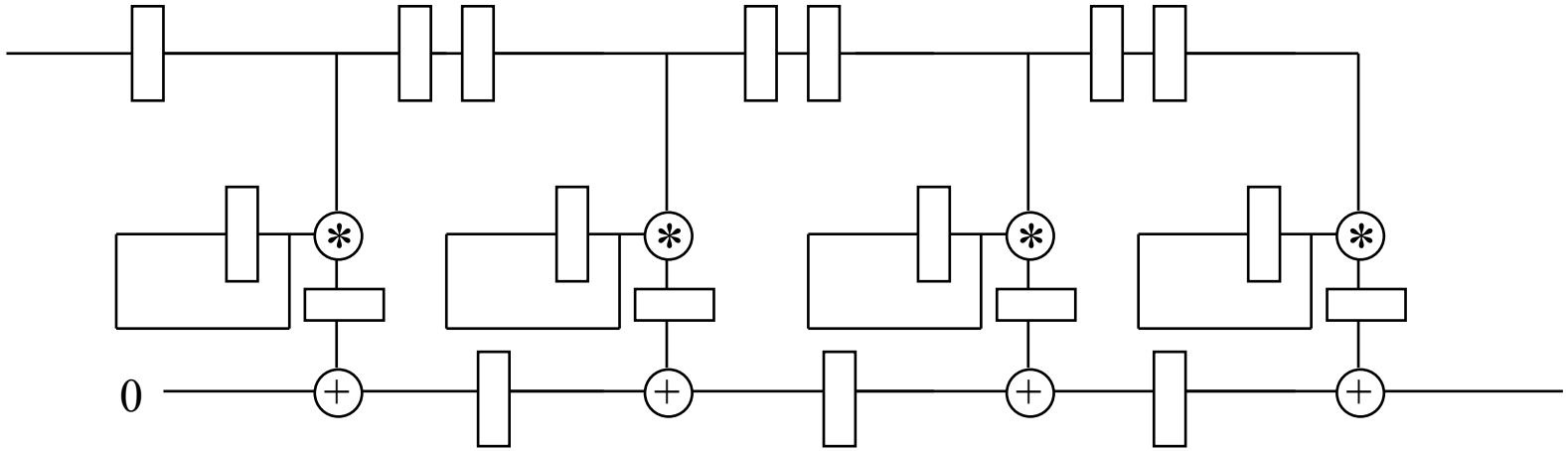
Convolution Example

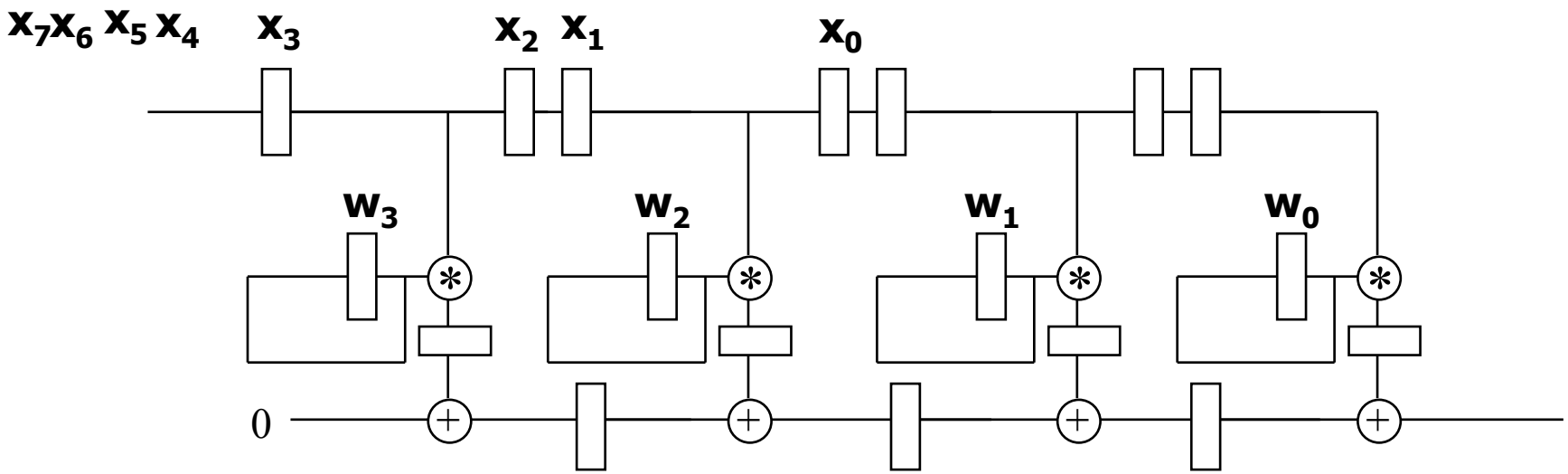


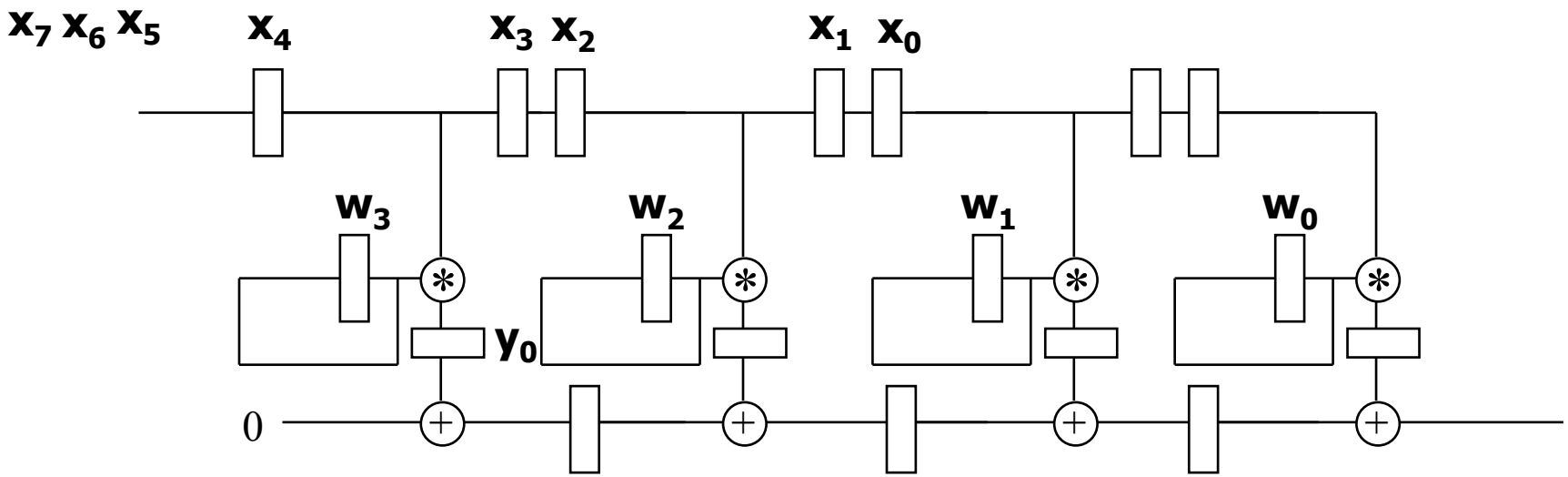
Convolution Example

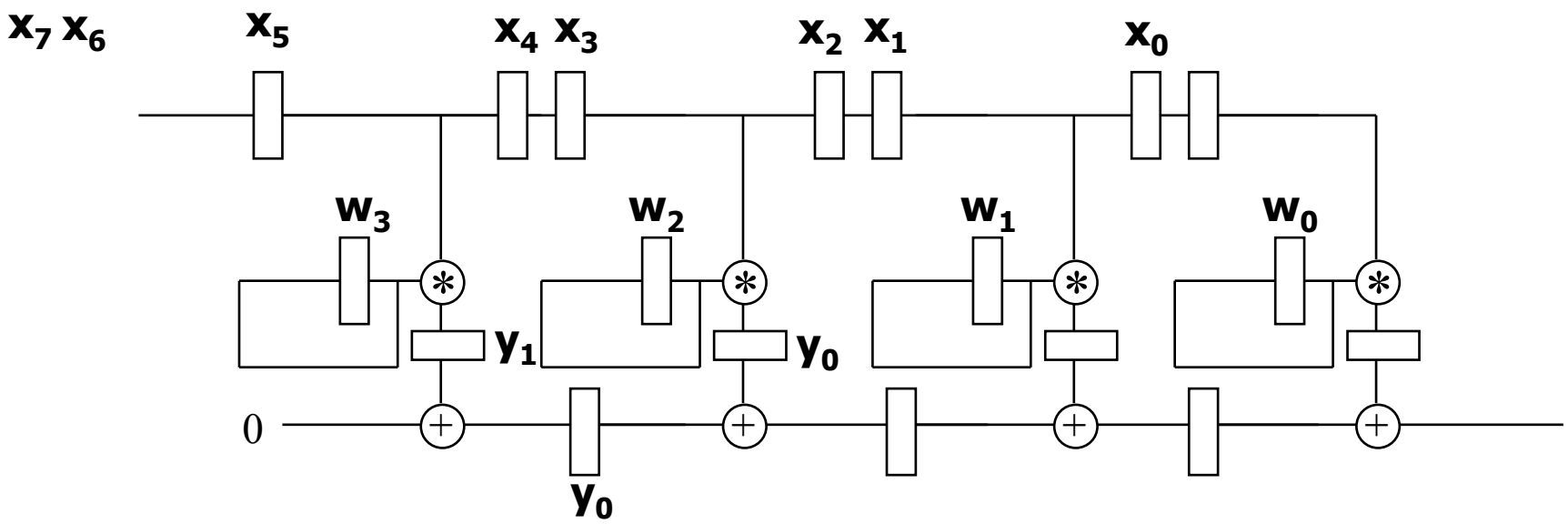


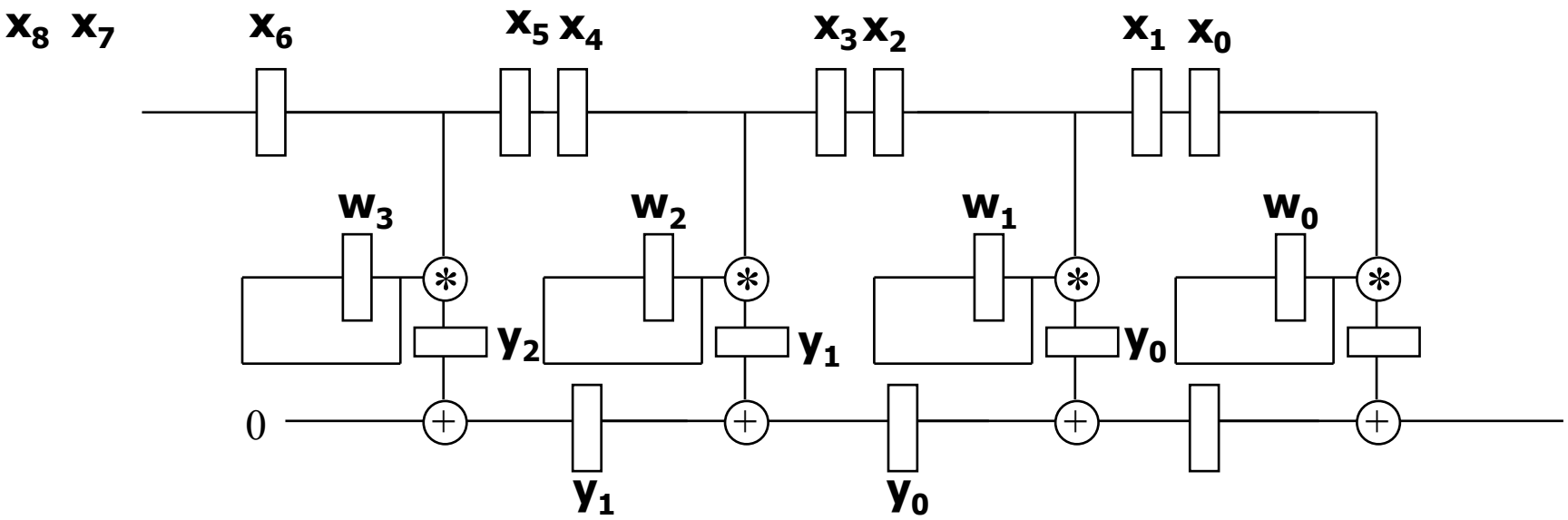

0

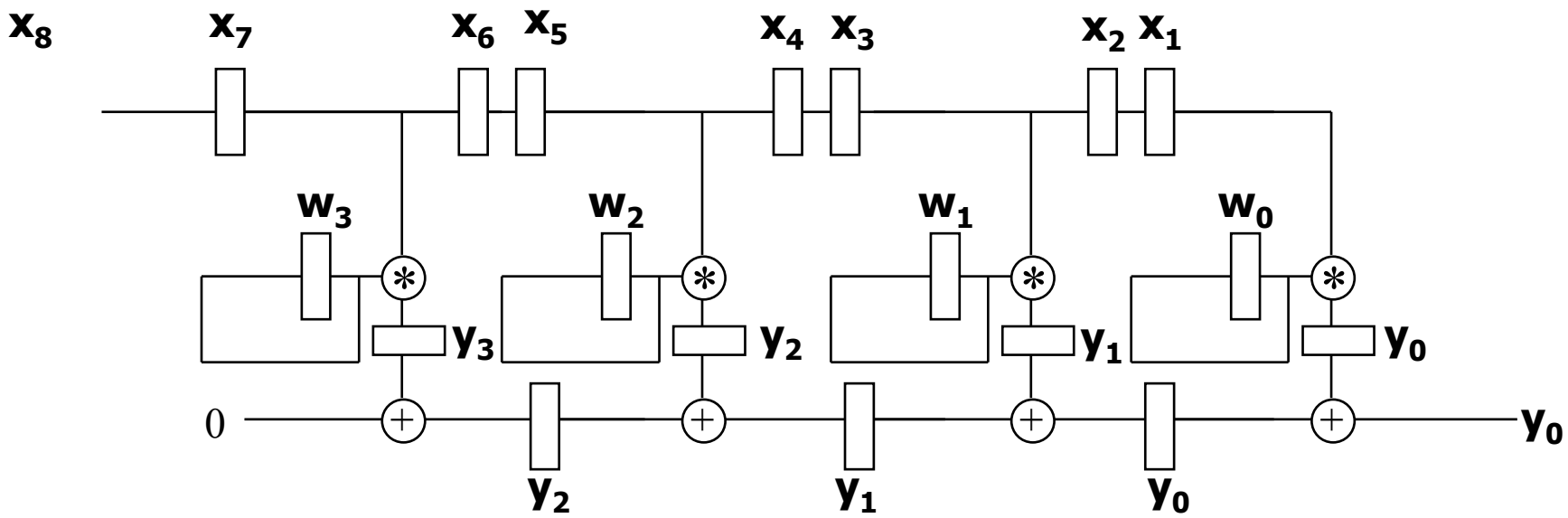

0

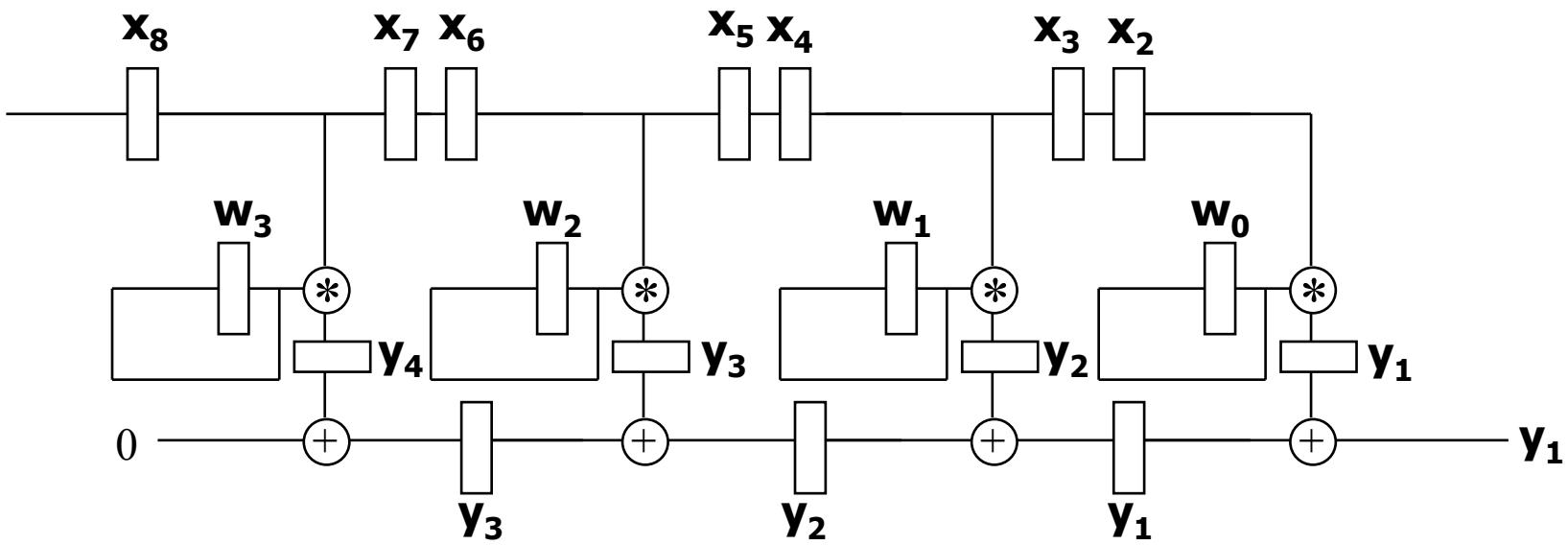

x7x6 x5 x4
x3
x2 x1
x0
w3
w2
w1
w0
0


x7 x6 x5
x4
x3 x2
x1 x0
w3
w2
w1
w0
y0
0


x7 x6
x5
x4 x3
x2 x1
x0
w3
w2
w1
w0
y1
y0
0
y0

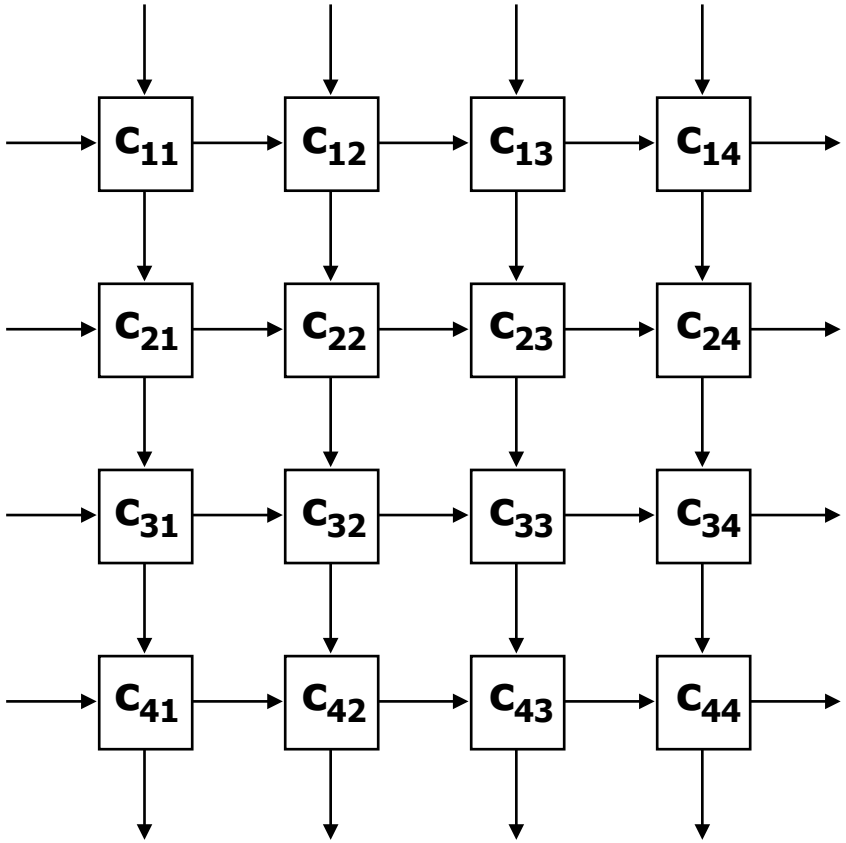

x8 x7
x6
x5 x4
x3 x2
x1 x0
w3
w2
w1
w0
y2
y1
y0
0
y1
y0


x8
x7
x6
x5
x4
x3
x2
x1
w3
w2
w1
w0
y3
y2
y1
y0
0
y2
y1
y0
y0

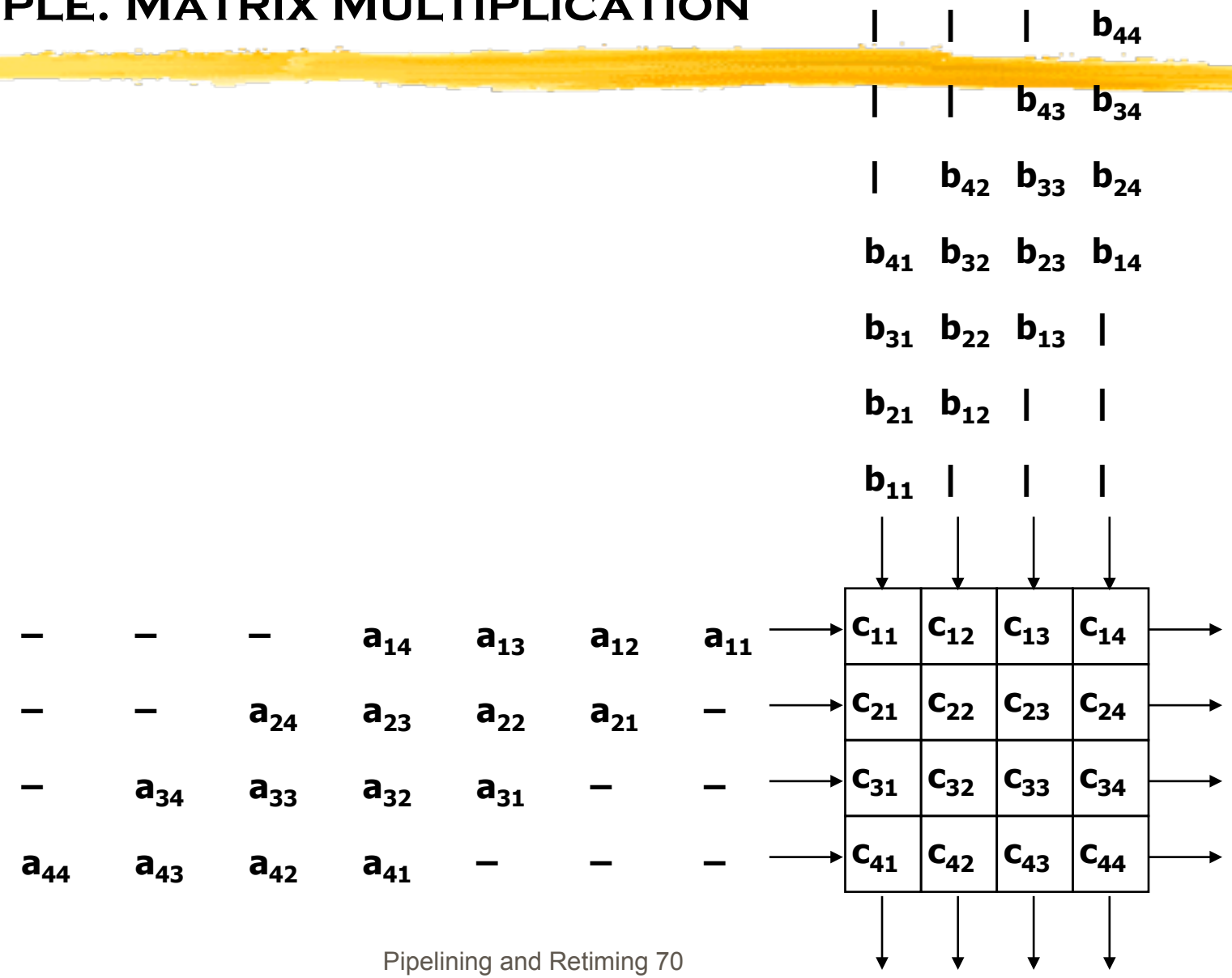

x8
x7 x6
x5 x4
x3 x2
w3
w2
w1
w0
y4
y3
y2
y1
0
y3
y2
y1
y1

Example: Matrix Multiplication

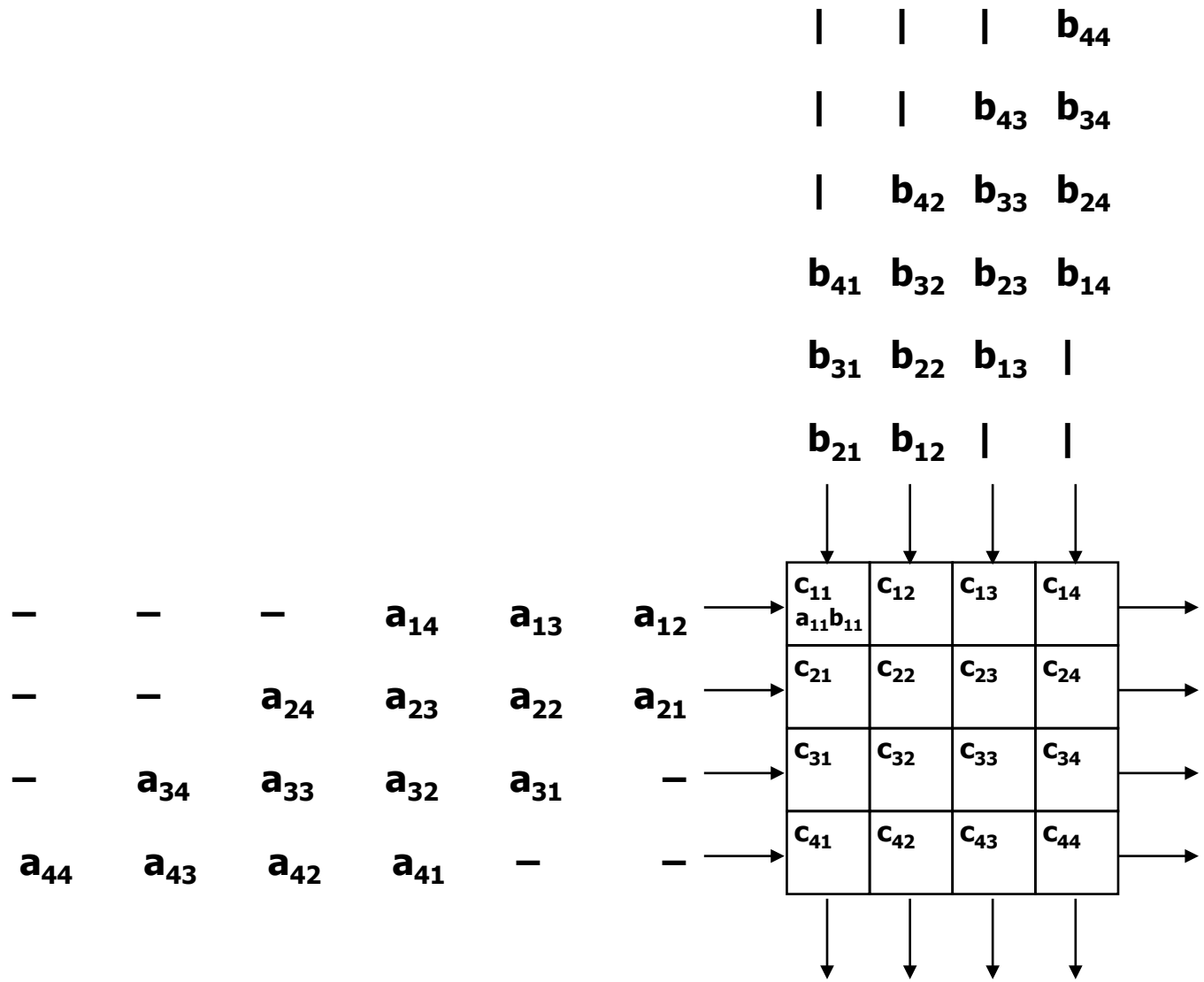
- $C = A \times B$ $\quad c_{ij} = \Sigma_{k=1}^{n} a_{ik} b_{kj}$



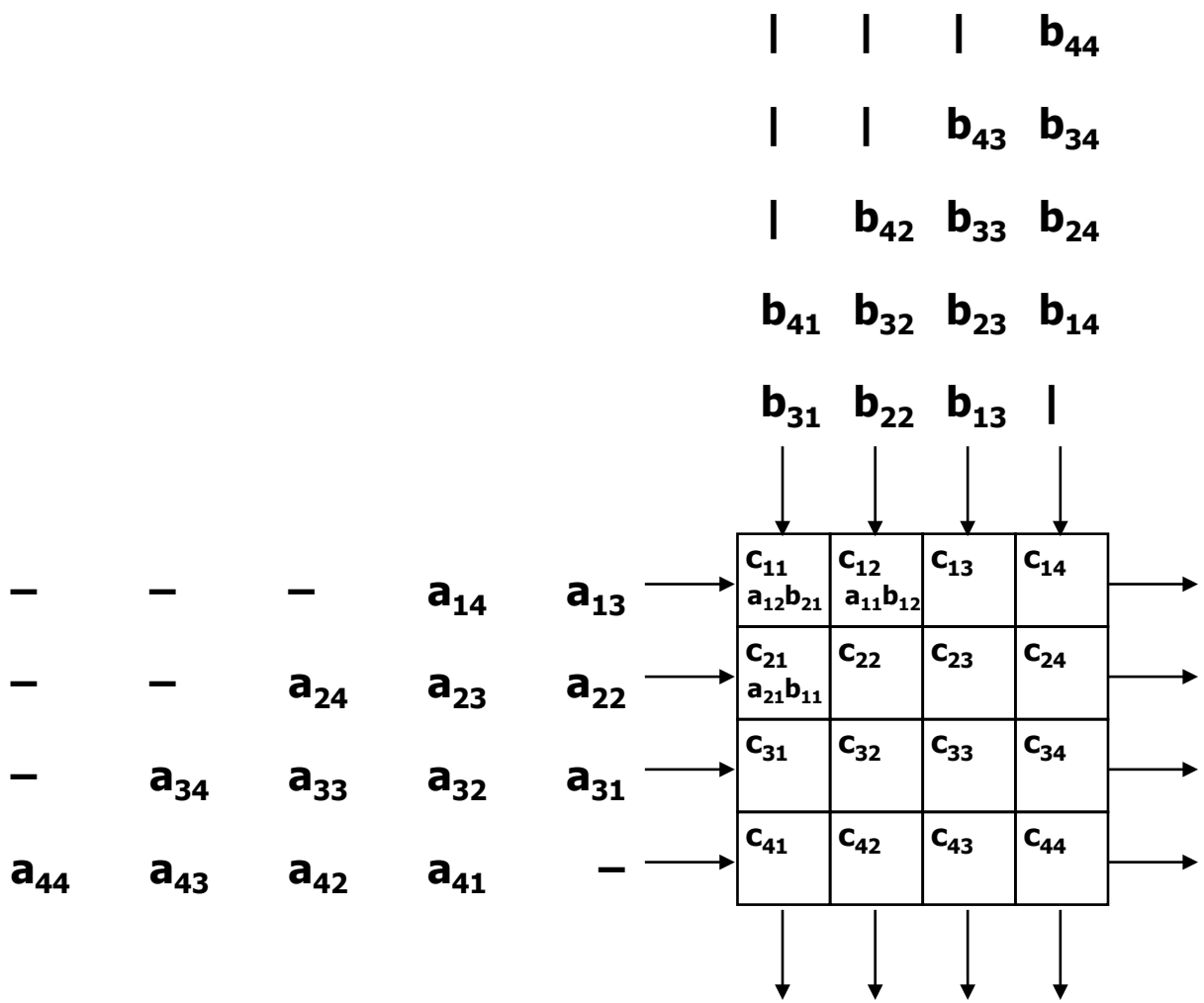
Example: Matrix Multiplication



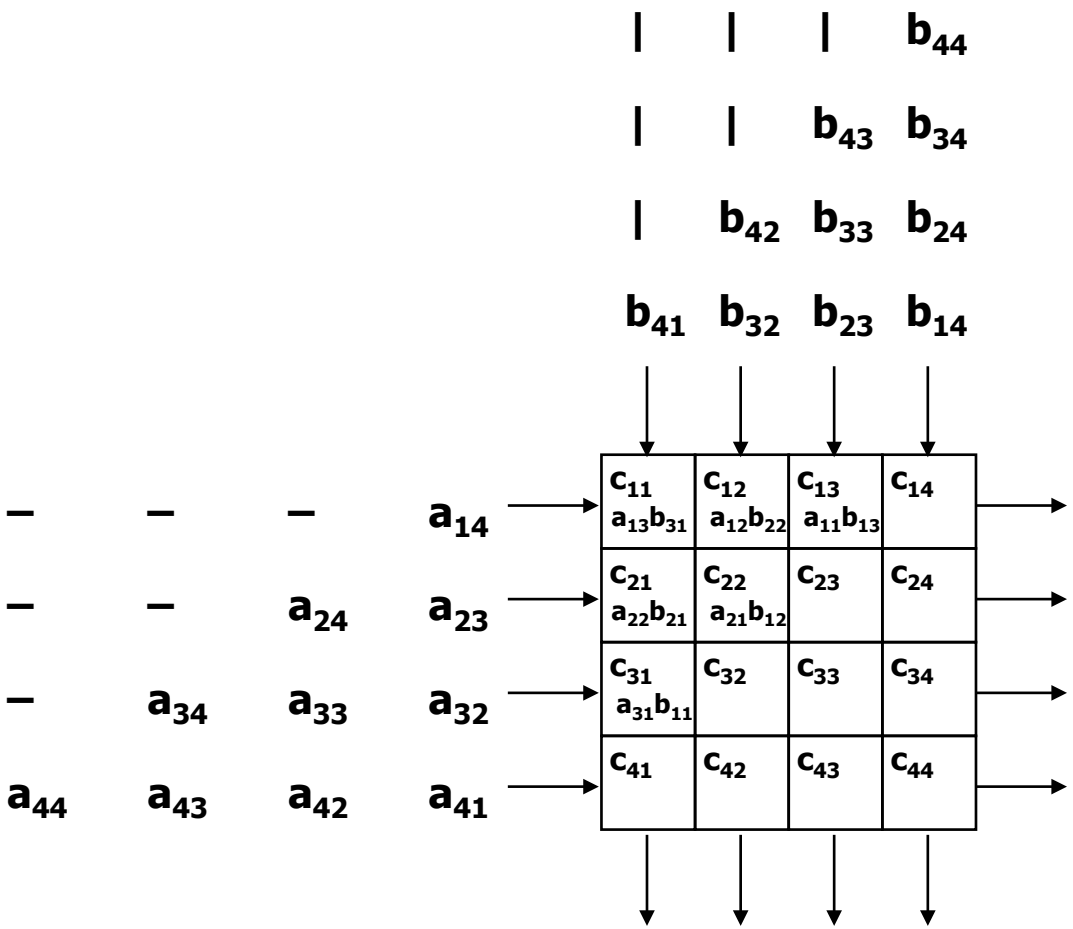
Example: Matrix Multiplication



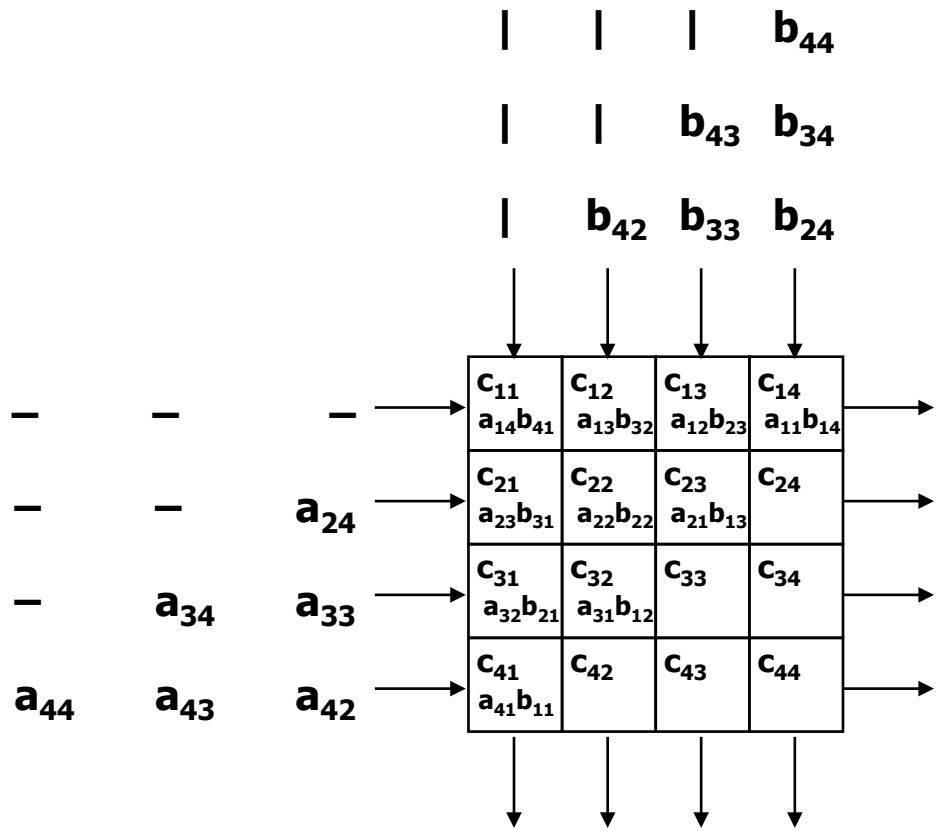
Example: Matrix Multiplication



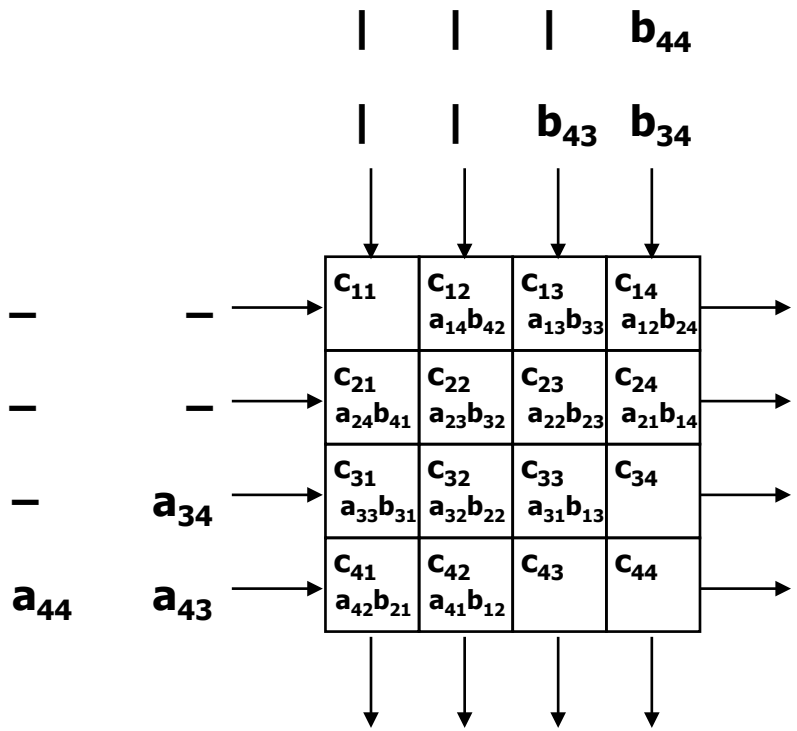
Example: Matrix Multiplication



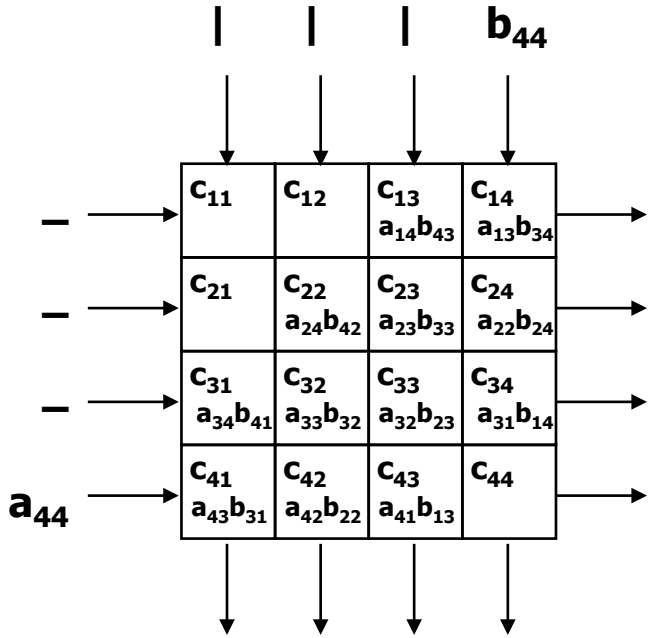
Example: Matrix Multiplication



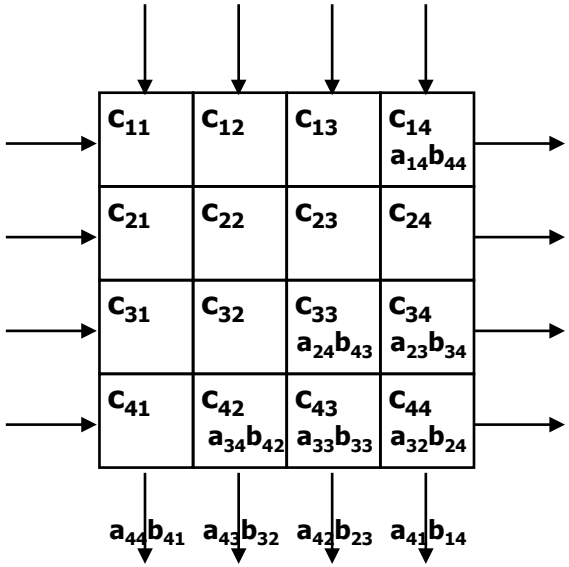
Example: Matrix Multiplication



Example: Matrix Multiplication



Example: Matrix Multiplication



Systolic Algorithms

- 2D Convolution
  - Image processing
- FFT
- String matching
  - Dynamic programming
  - DNA comparison
- Matrix computations
  - LU decomposition
  - QR factorization

Systolic Architectures

- Highly parallel
  - "fine-grained" parallelism
  - deep pipelining
- Local communication
  - wires are short - no global communication (except CLK)
    - linear array → no clock skew
  - increasingly important as wire delays increase (relative to gate delays)
- Linear arrays
  - most systolic algorithms can be done with a linear array
  - include memory in each cell in the array
  - linear array a better match to I/O limitations
- Contrast to superscalar and vector architectures

Systolic Computers

- Custom chips - early 1980's
- Warp (CMU) - 1987
  - linear array of 10 or more processing cells
  - optimized inter-cell communication for low-latency
  - pipelined cells and communication
  - conditional execution
  - compiler partitions problem into cells and generates microcode
- i-Warp (Intel) - 1990
  - successor to Warp
  - two-dimensional array
  - time-multiplexing of physical busses between cells
  - 32x32 array has 20Gflops peak performance
  - not a commercial success
- Currently confined to ASIC implementations