# Systolic Computing

## Fundamentals

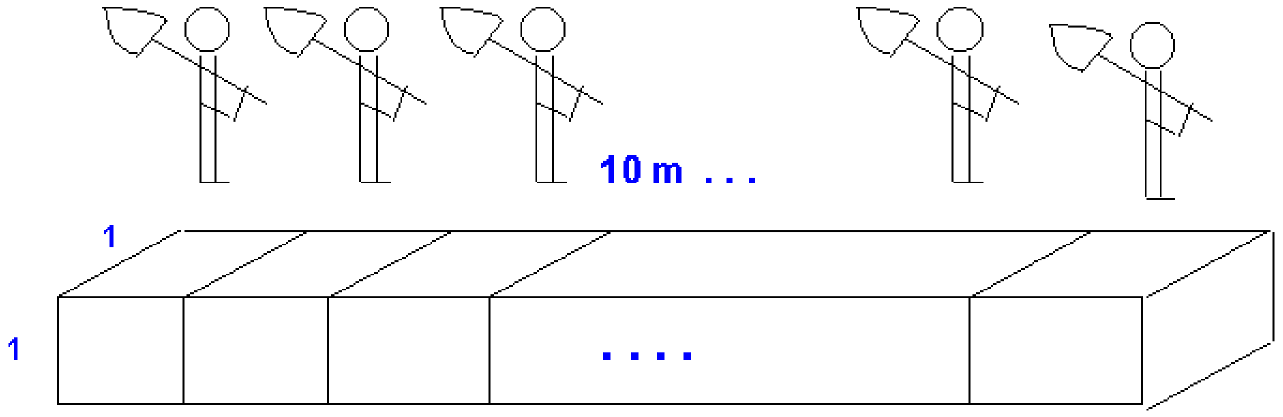
## Motivations for Systolic Processing

X ← 2; Y ← 4

X ← 3; Y ← X

parallel Y= 2

sequence Y=3



•10 hours needed if a single person can dig a one-by-one-by-one hole in one hour

• What happens if the hole is to be 10 meter deep?

Even with 100 persons 10 hours are still needed!!

- PARALLEL ALGORITHMS
  - WHICH MODEL OF COMPUTATION IS THE BETTER TO USE?
  - HOW MUCH TIME WE EXPECT TO SAVE USING A PARALLEL ALGORITHM?
  - HOW TO CONSTRUCT EFFICIENT ALGORITHMS?
- MANY CONCEPTS OF THE COMPLEXITY THEORY MUST BE REVISITED
  - » IS THE PARALLELISM A SOLUTION FOR HARD PROBLEMS?
  - ARE THERE PROBLEMS NOT ADMITTING AN EFFICIENT PARALLEL SOLUTION, THAT IS INHERENTLY SEQUENTIAL PROBLEMS?
  - » INDECIDABLE PROBLEMS REMAIN **UNDECIDABLE**!

# Data Parallel Systems

- **Programming model**
  - Operations performed in parallel on each element of data structure
  - Logically single thread of control, performs sequential or parallel steps
  - Conceptually, a processor associated with each data element
- **Architectural model**
  - Array of many simple, cheap processors with little memory each
  - Processors don’t sequence through instructions
  - Attached to a control processor that issues instructions
  - Specialized and general communication, cheap global synchronization
- **Original motivations**
  - Matches simple differential equation solvers
  - Centralize high cost of instruction fetch/sequencing

# Application of Data Parallelism

- Each PE contains an employee record with his/her salary

```
If salary > 100K then
salary = salary *1.05
else
salary = salary *1.10
```

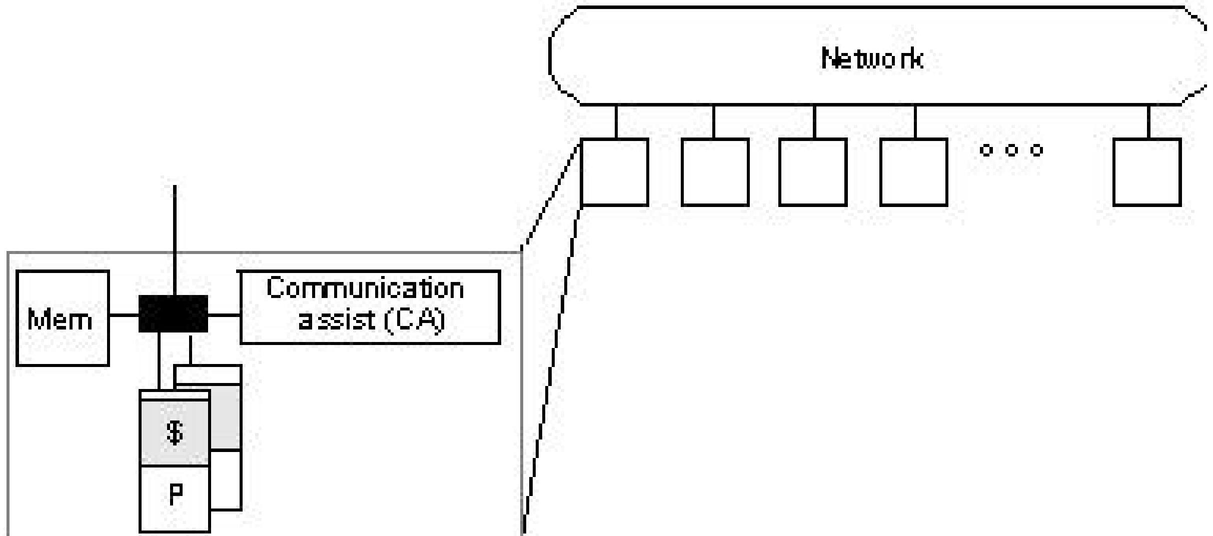
- Logically, the whole operation is a single step
- Some processors enabled for arithmetic operation, others disabled
- **Other examples:**
  - Finite differences, linear algebra, ...
  - Document searching, graphics, image processing, ...
- **Some famous machines:**
  - Thinking Machines CM-1, CM-2 (and CM-5)
  - Maspar MP-1 and MP-2,

## Flynn’s Taxonomy

- **# instruction x # Data**
  - Single Instruction Single Data (SISD)
  - Single Instruction Multiple Data (SIMD)
  - Multiple Instruction Single Data
  - Multiple Instruction Multiple Data (MIMD)
- **Everything is MIMD!**

## Convergence: Generic Parallel Architecture

- **Node: processor(s), memory system, plus communication assist**
  - Network interface and communication controller
- **Scalable network**
- **Convergence allows lots of innovation, within framework**



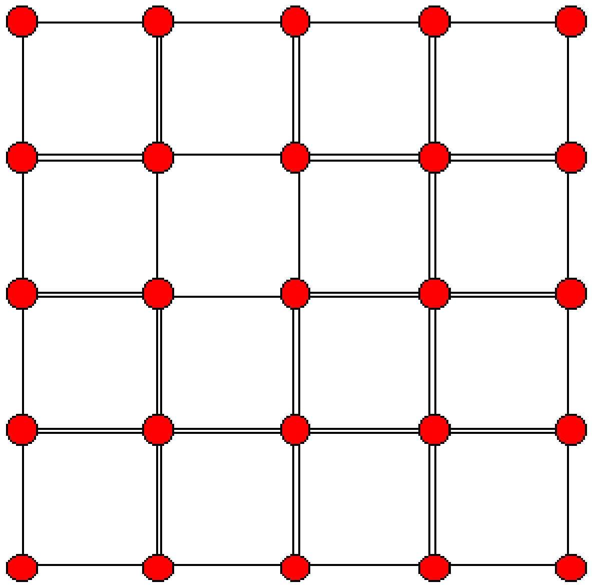
## We need a model of computation

• NETWORK (VLSI) MODEL

- The processors are connected by a network of **bounded degree.**
- No **shared memory** is available.
- Several interconnection topologies.
- Syncronous way of operating.

### MESH CONNECTED ARRAY

degree = 4



diameter = ¦N

A model for VLSI processing also needs parallelism

## ODD-EVEN MERGE-SORT

N elements are sorted on a Butterfly of order log N in

$$2 \log 2 + 2 \log 4 + 2 \log 8 + \dots + 2 \log N = 2(1 + 2 + \dots + \log N) = \log N(\log N + 1)$$
$$= O(\log^2 n)$$

Merge 2 lists A and B of $m = 2^k$ elements in L

$A = a_0, \dots, a_{m-1}$ $B = b_0, \dots, b_{m-1}$ set

even(A) = $a_0, \dots, a_{m-2}$ odd(A) = $a_1, \dots, a_{m-1}$ even(B) = $b_0, \dots, b_{m-2}$ odd(B) = $b_1, \dots, b_{m-1}$

Merge even(A) and odd (B) to form C Merge odd(A) and even (B) to form D

Merging C and D its easier than merging A and B

(A and B are distributed among C and D so as to balance the num ber of *small* items)

Form L' by interleaving $C = c_0, \dots, c_{m-1}$ and $D = d_0, \dots, d_{m-1}$

$L' = c_0 d_0, c_1 d_1, \dots, c_{m-1} d_{m-1}$

Compare $c_i d_i$ only and flip those out of order; Magically everything is sorted !!

A = 2, 3, 4, 8 B = 1, 5, 6, 7

even(A) = 2, 4 odd(A) = 3, 8 even(B) = 1, 6 odd(B) = 5, 7

C = 2, 4, 5, 7 D = 1, 3, 6, 8

L' = 2, 1, 4, 3, 5, 6, 7, 8

Example of parallel computation

- In the network model a PARALLEL MACHINE is a very complex ensemble of small interconnected units, performing elementary operations.
- Each processor has its own memory.
- Processors work synchronously.
- LIMITS OF THE MODEL
  - different topologies require different algorithms to solve the same problem
  - it is difficult to describe and analyze algorithms (the migration of data have to be described)
  - A shared-memory model is more suitable by an algorithmic point of view

## THREE TYPES OF MULTIPROCESSING FRAMEWORKS, CLOSELY RELATED

- CONCURRENT
- PARALLEL
- DISTRIBUTED

MULTIPROCESSING ACTVITIES TAKE PLACE IN A SINGLE MACHINE (POSSIBLY USING SEVERAL PROCESSORS), SHARING MEMORY AND TASKS.

### TECHNICAL ASPECTS

- PARALLEL COMPUTERS (USUALLY) WORK IN TIGHT SYNCRONY, SHARE MEMORY TO A LARGE EXTENT AND HAVE A VERY FAST AND RELIABLE COMMUNICATION MECHANISM BETWEEN THEM.

- DISTRIBUTED COMPUTERS ARE MORE INDEPENDENT, COMMUNICATION IS LESS FREQUENT AND LESS SYNCRONOUS, AND THE COOPERATION IS LIMITED.

### PURPOSES

- PARALLEL COMPUTERS COOPERATE TO SOLVE MORE EFFICIENTLY (POSSIBLY) DIFFICULT PROBLEMS

- DISTRIBUTED COMPUTERS HAVE INDIVIDUAL GOALS AND PRIVATE ACTIVITIES. *SOMETIME* COMMUNICATIONS WITH OTHER ONES ARE NEEDED. (E. G. DISTRIBUTED DATA BASE OPERATIONS).

PARALLEL COMPUTERS: COOPERATION IN A *POSITIVE* SENSE

DISTRIBUTED COMPUTERS: COOPERATION IN A *NEGATIVE* SENSE, ONLY WHEN IT IS NECESSARY

# Systolic arrays

- '*Laying out algorithms in VLSI*'
  - efficient use of hardware
  - not general purpose
  - not suitable for large I/O bound applications
  - control and data flow must be regular
- Achieve pipelining and parallel execution
- Simple cells
- Each cell performs one operation
  - (usually)

# Systolic Computing

- **Definition**
- sys·to·le (sîs¹te-lê) noun
- The rhythmic contraction of the heart, especially of the ventricles, by which blood is driven through the aorta and pulmonary artery after each dilation or diastole.
- [Greek sustolê, contraction, from sustellein, to contract. See systaltic.]
- — sys·tol¹ic (sî-stòl¹îk) adjective
- American Heritage Dictionary
- Data flows from memory in a rhythmic fashion, passing through many processing elements before it returns to memory.
- H.T.Kung

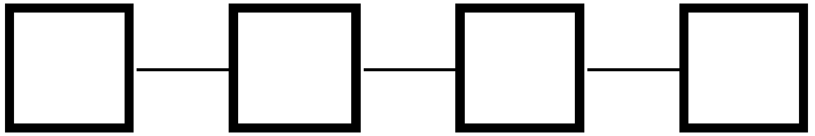
- Systolic' is normally used to describe the regular pumping action of the heart
- By analogy, systolic computers pump data through
- The idea is to exploit VLSI efficiently by laying out algorithms (and hence architectures) in 2-D (not all systolic machines are 2-D, but probably most are)
- The architectures thus produced are not general but tied to specific algorithms
- This is good for computation-intensive tasks but not I/O-intensive tasks
    - e.g. signal processing
- Most designs are simple and regular in order to keep the VLSI implementation costs low
    - programs with simple data and control flow are best
- Systolic computers show both pipelining and parallel computation

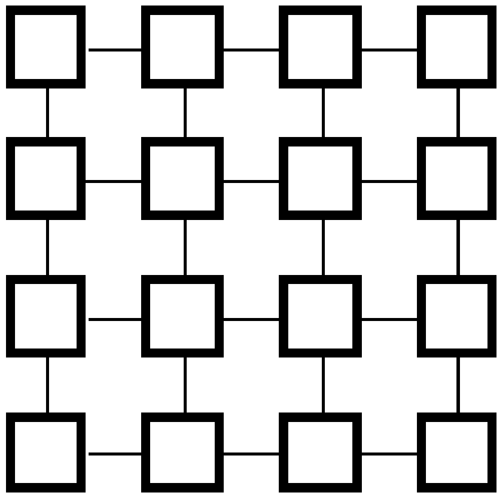
# Structures for Systolic Computing

# Systolic Computing

## Definition

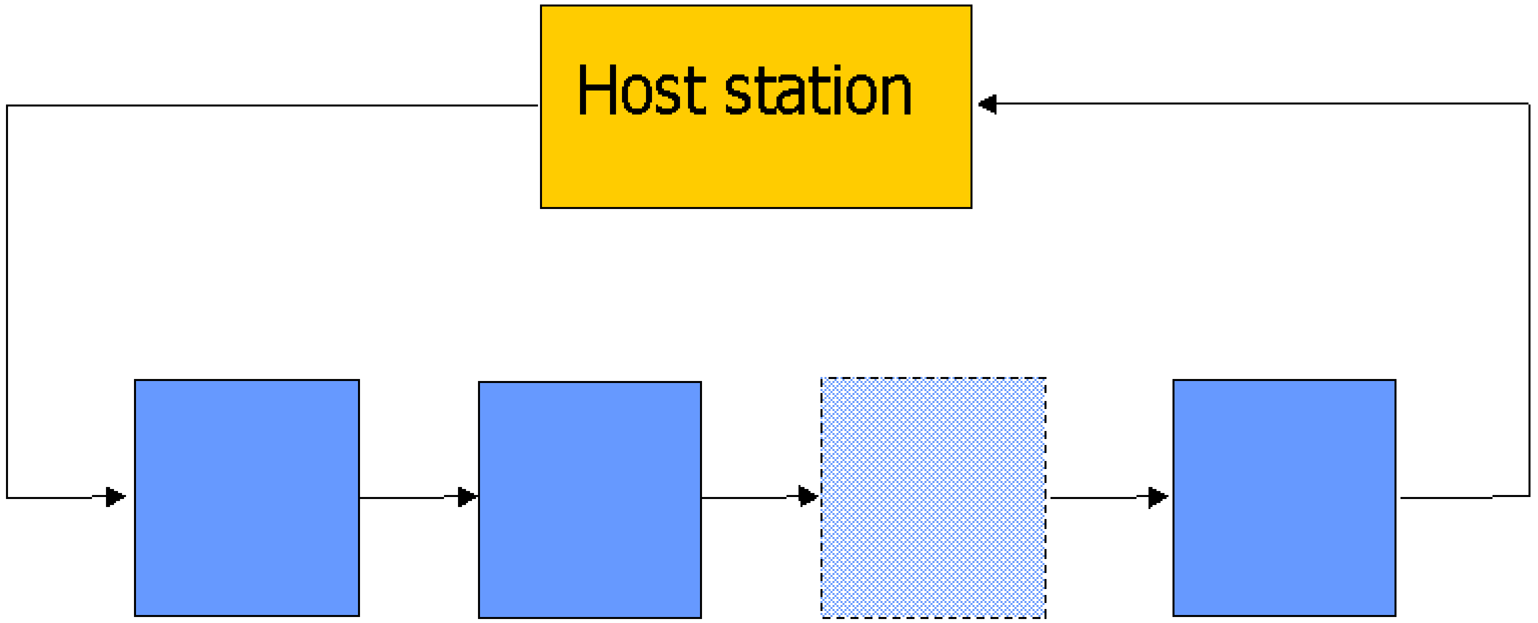
A set of simple processing elements with regular and local connections which takes external inputs and processes them in a predetermined manner in a pipelined fashion





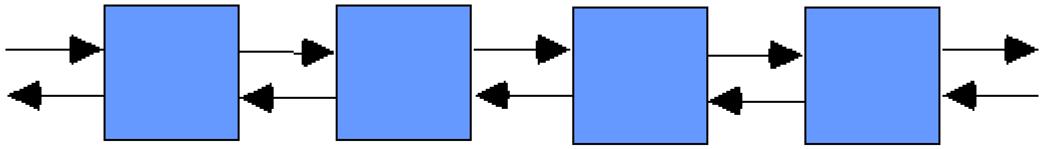
# Systolic Architecture

- A systolic network is often use with a host station responsible for the communication with the outside world.
- As a result of the local-communication scheme, a systolic network is easily extended without to add any burden to the I/O.



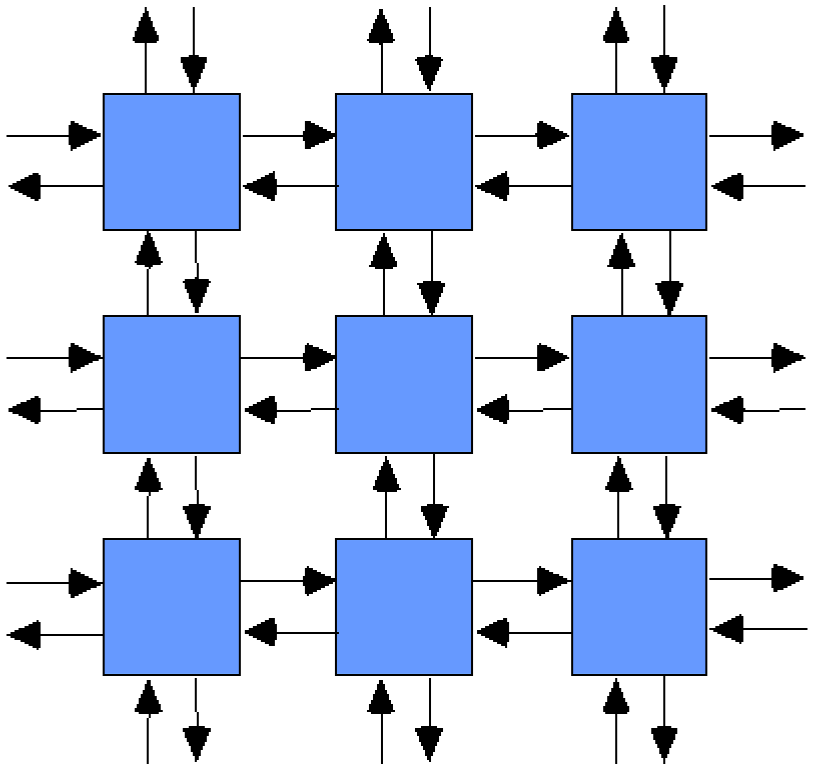
# Systolic Architecture

- Example of systolic architecture: linear network



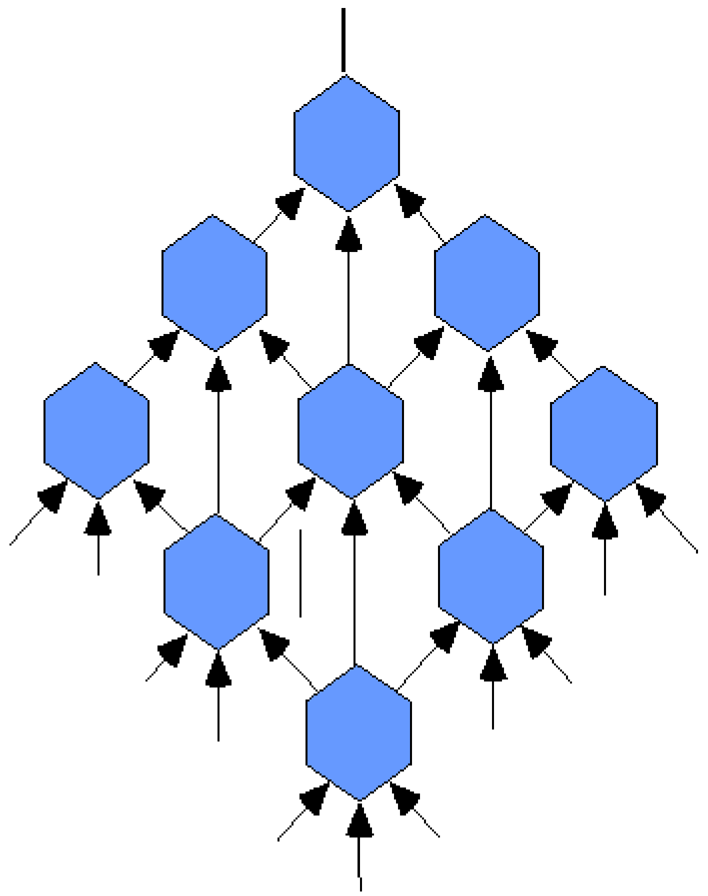
# Systolic Architecture

- Example of systolic network: Bi-dimensional network



# Systolic Architecture

- Example of systolic network: hexagonal network



# Motivation for Systolic

- **Effectively utilize VLSI**
- **Reduce "Von Neumann Bottleneck"**
- **Target compute-intensive applications**
- **Reduce design cost:**
  - Simple
  - Regular
- **Exploit Concurrency**

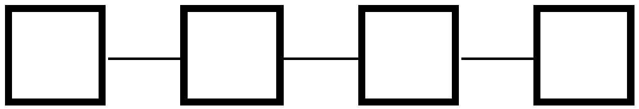
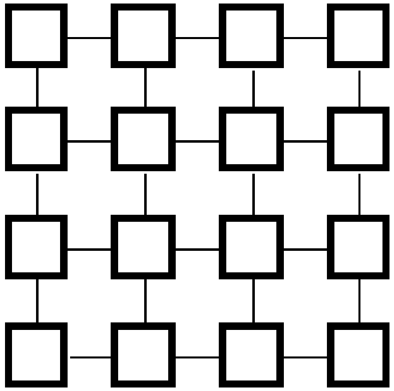
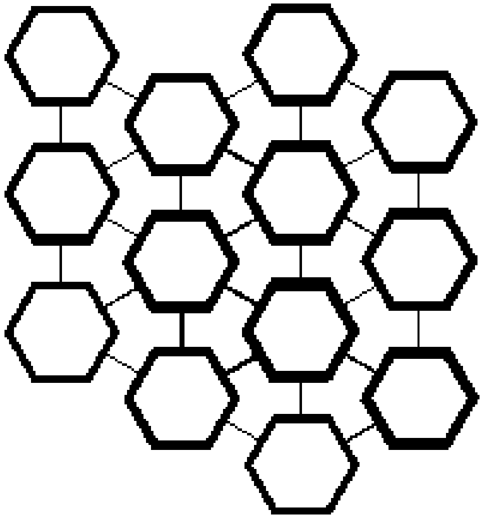
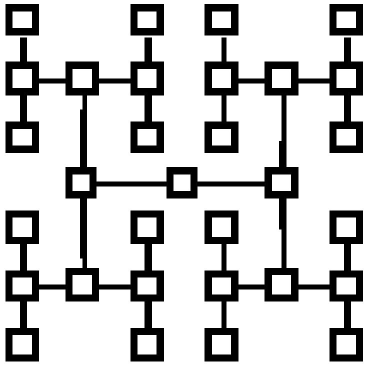
# Using VLSI Effectively

- **Replicate simple cells**
- **Local Communication ==>**
  - Short wires
    - small delay
    - low clock skew
    - small drivers
    - less area
  - Scalable
- **Small number of I/Os**

Routing costs dominate: power, area, *and* time!

# Regular Interconnect: *why good*

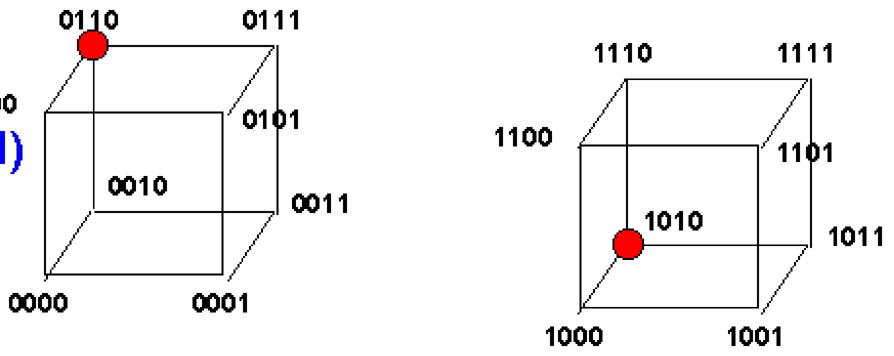
- 1-d Array
- 2-d Array
- Hex Array
- Triangular Array
- Tree

# Hypercubes

**N = $2^4$ PROCESSORS**

**degree = 4 ($\log_2 N$)**



**diameter = 4**

**Sum on the Hypercube**

**input: an array A of n elements such that each element A(i) is stored in the local memory of processor $P_i$**

**output: the sum $S = \square_{i=1,n} A(i)$**

**algorithm for $P_i$**

```
begin
  for l= d-1 to 0 do
    if (0Š i Š $2^{l-1}$) then
      Set A(i) := A(i) + A($i^{(l)}$)
end
```

$i^{(l)}$ index i with bit l complemented

## Characteristics of Systolic Architectures

## Regular Interconnect in 3D

- 3-d Array
- 4-d Array (mapped to 3-D)
- 3-D Hex Array
- 3-D Trees and Lattices

## Eliminating the Von Neuman's Bottleneck

- **Process each input multiple times.**
- **Keep partial results in the PEs.**
- **Does this still present a win today?**
  - Large cost
  - Many registers

## Balancing I/O and Computation

- Can’t go faster than the data arrives
- Reduce bandwidth requirements
- Choose applications well!
- Choose algorithms correctly!

## Exploiting Concurrency

- **Large number of simple PEs**
- **Manage without instruction store**
- **Methods:**
  - Pipelining
  - SIMD/MIMD
  - Vector
- **Limits application space. How severely?**

## Systolic versus Reconfigurable Computing

## Systolic Architectural Model is a good match with FPGAs and Reconfigurable Computing

- **Simple PEs**
- **Regular and local interconnect**
- **Pipeline between PEs**
- **I/O at boundary**

- Characteristics of best RC Designs.
  - RC = **reconfigurable computing**

# Systolic architecture versus RC

- **A systolic architecture has the following characteristics :**
  - A massive and non-centralized parallelism
  - Local communications
  - Synchronous evaluation
  - Only the processors at the border of the architecture can communicate outside.
  - The task of one cell can be summarized as : receive-compute-transmit

## Systolic architecture versus RC

- **Other characteristics :**
  - Data coming from the memory are used several time before to come back to it.
  - These architectures are well suited for a VLSI or FPGA network implementation

## Systolic Is Good RC Starting Point.

- **Original motivation: Custom silicon for an application.**
- **Model Architecture is efficient for RC**
- **Target Algorithms match**
- **Well developed theory**
- **Compilation technology well studied**

- Was Systolic Computing ahead of its time?

## One Big Difference

- **Kung et al's approach fabricated silicon.**
- **RC compiles to fabric.**

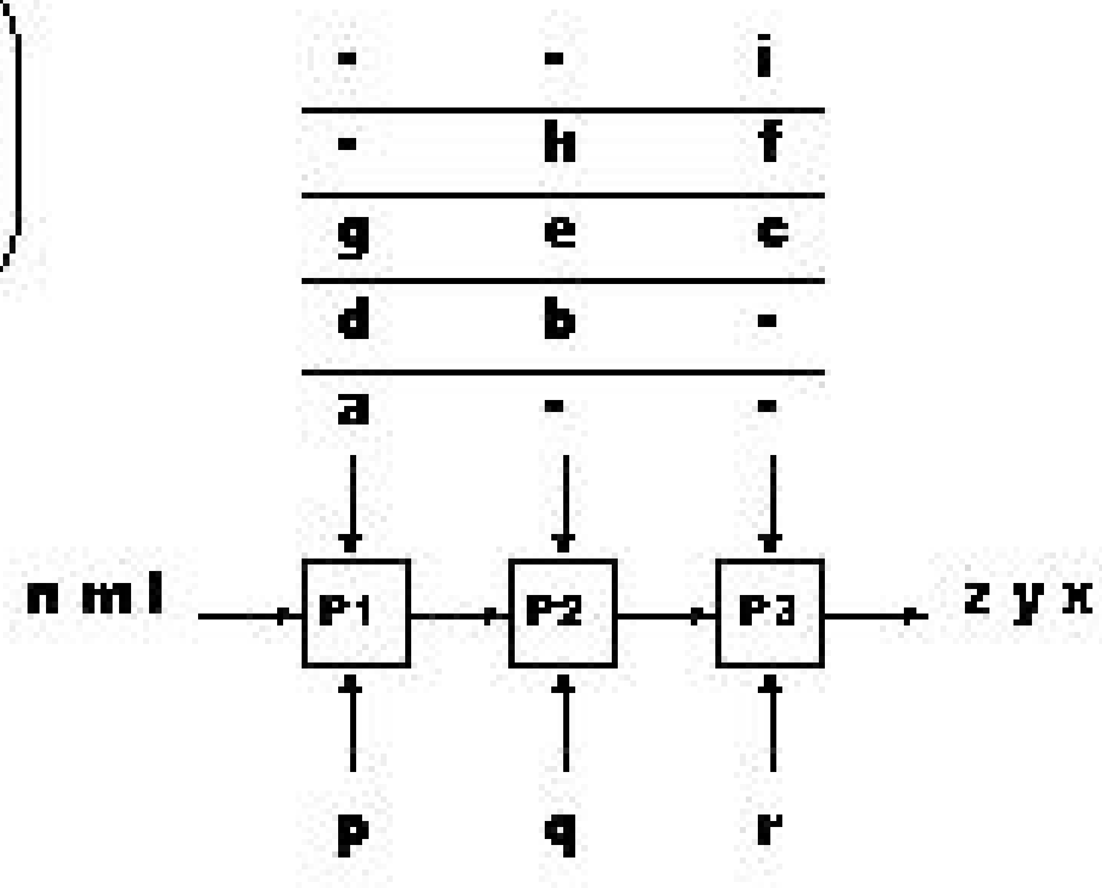
- Kung's PEs must be "general purpose."

## Mapping Approach for RC

- **Allocate PEs**
- **Schedule computation**
  - schedule PEs structure
  - schedule data flow driver
- **Optimize**

# **Example:** Matrix Multiplication

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} l \\ m \\ n \end{pmatrix} + \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix}$$



- Analyze each row

- Each cell (P1, P2, P3) does just one instruction
  - Multiply the top and bottom inputs, add the left input to the product just obtained, output the final result to the right
- The cells are simple
  - Just an adder and a few registers
- The cleverness comes in the order in which you feed input into the systolic array
  - At time t0, the array receives l, a, p, q, and r (the other inputs are all zero)At time t1, the array receives m, d, b, p, q, and r And so on.
- Results emerge after 5 (?) steps

**Processor P**

local operations

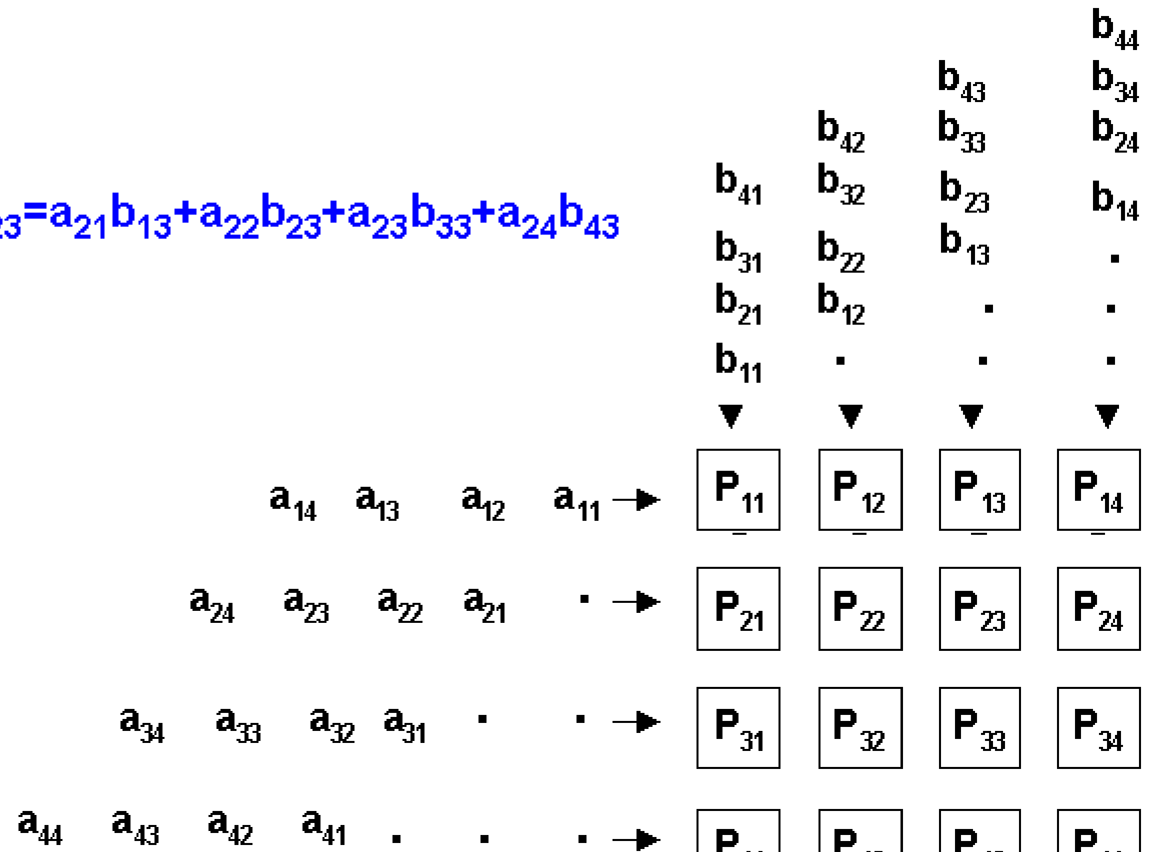
communication operations

- send (X,i)
- receive (y,j)

## Systolic Matrix Moltiplication on the Mesh

input: A, B nxn matrices; output: C = AxB where $c_{ij} = \Box_{k=1,n} \; a_{ik} b_{kj}$

$c_{23}=a_{21}b_{13}+a_{22}b_{23}+a_{23}b_{33}+a_{24}b_{43}$



$b_{44}$ $b_{34}$ $b_{24}$ $b_{14}$ . . .
$b_{43}$ $b_{33}$ $b_{23}$ $b_{13}$ . .
$b_{42}$ $b_{32}$ $b_{22}$ $b_{12}$ .
$b_{41}$ $b_{31}$ $b_{21}$ $b_{11}$
$a_{14}$ $a_{13}$ $a_{12}$ $a_{11}$
$a_{24}$ $a_{23}$ $a_{22}$ $a_{21}$ .
$a_{34}$ $a_{33}$ $a_{32}$ $a_{31}$ . .
$a_{44}$ $a_{43}$ $a_{42}$ $a_{41}$ . . .
$P_{11}$ $P_{12}$ $P_{13}$ $P_{14}$
$P_{21}$ $P_{22}$ $P_{23}$ $P_{24}$
$P_{31}$ $P_{32}$ $P_{33}$ $P_{34}$

when $P_{ij}$ receives $A_{il}$ and $B_{lj}$ computes $C_{ij} := C_{ij} + A_{il} B_{lj}$

then sends $A_{il}$ to the right and $B_{lj}$ down

**Example:** Matrix-Matr

Multiplication

## Example:

## FIR Filter or Convolution

## Various Possible Implementations

- Broadcast
  - Y moves
  - W moves
- Fan-In
- Systolic
  - Bidirectional
    - Y in place
    - X in place
  - Unidirectional
    - Y in place
    - W in place

Running Example: Convolution

```
for (i=1; i ≤ n; i++)
    for (j=1; j ≤ k; j++)
        $y_i$ += $w_j$ * $x_{i+j-1}$
```

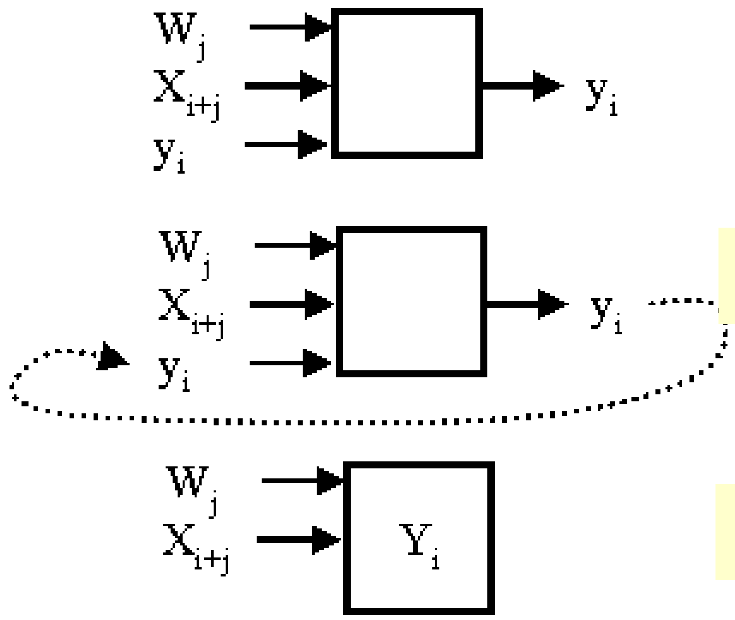
# Bag of Tricks

- Preload-repeated-value
- Replace-feedback-with-register
- Internalize-data-flow
- Broadcast-common-input
- Propagate-common-input
- Retime-to-eliminate-broadcasting

# Bogus Attempt at Systolic FIR

```
for i=1 to n in parallel
    for j=1 to k in place
        $y_i$ += $w_j * x_{i+j-1}$
```

Inner Loop:

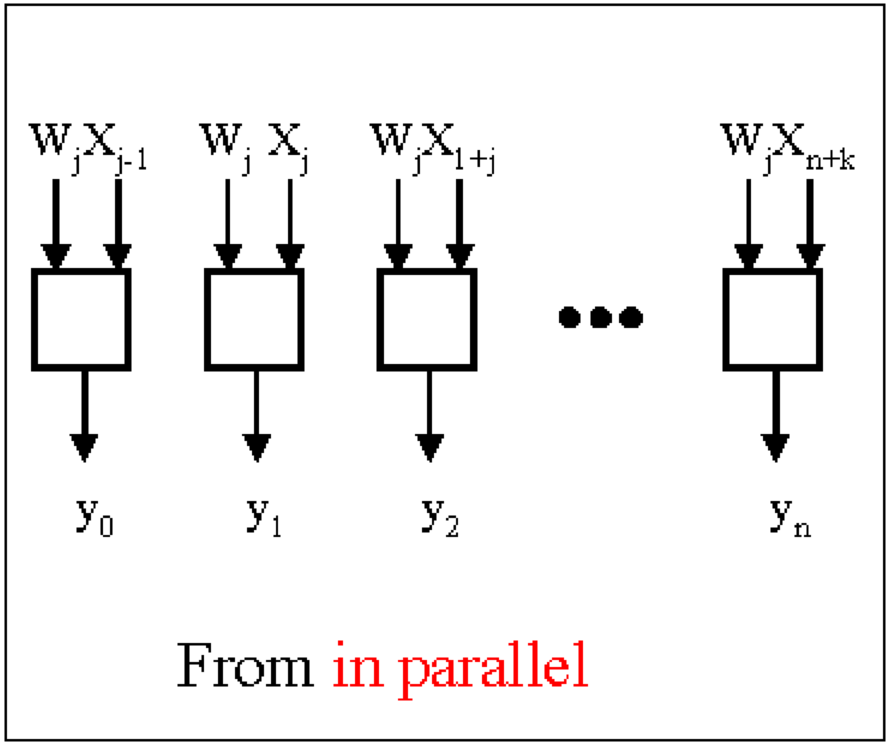


feedback from sequential implementation

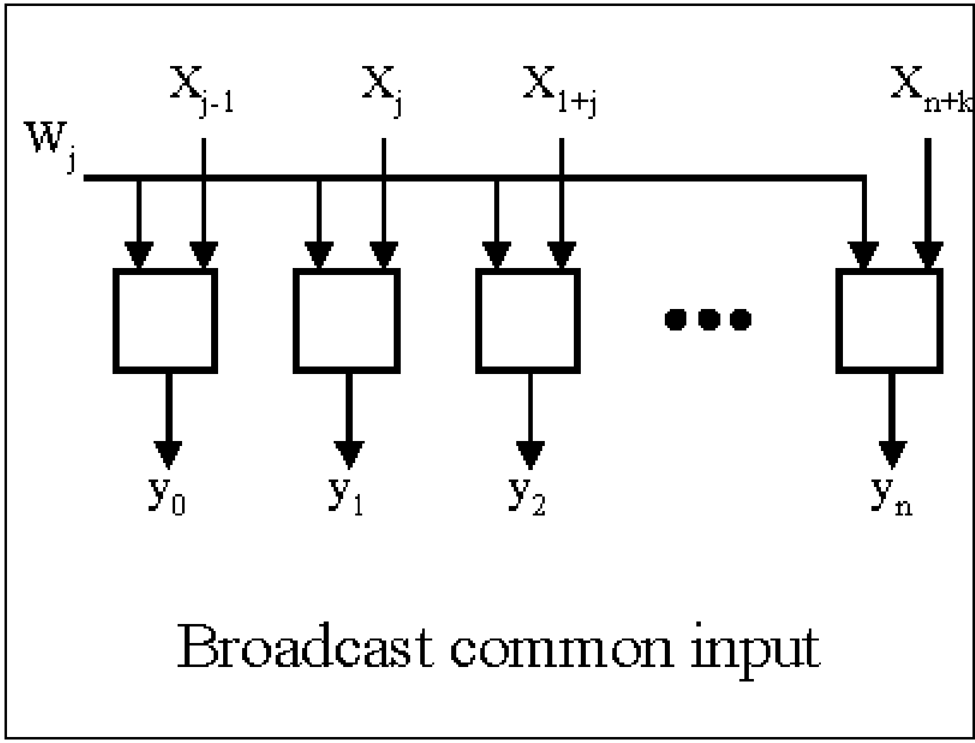
Replace with register

# Bogus Attempt: Outer Loop

```
for i=1 to n in parallel
    for j=1 to k in place
        $y_i$ += $w_j$ * $x_{i+j-1}$
```



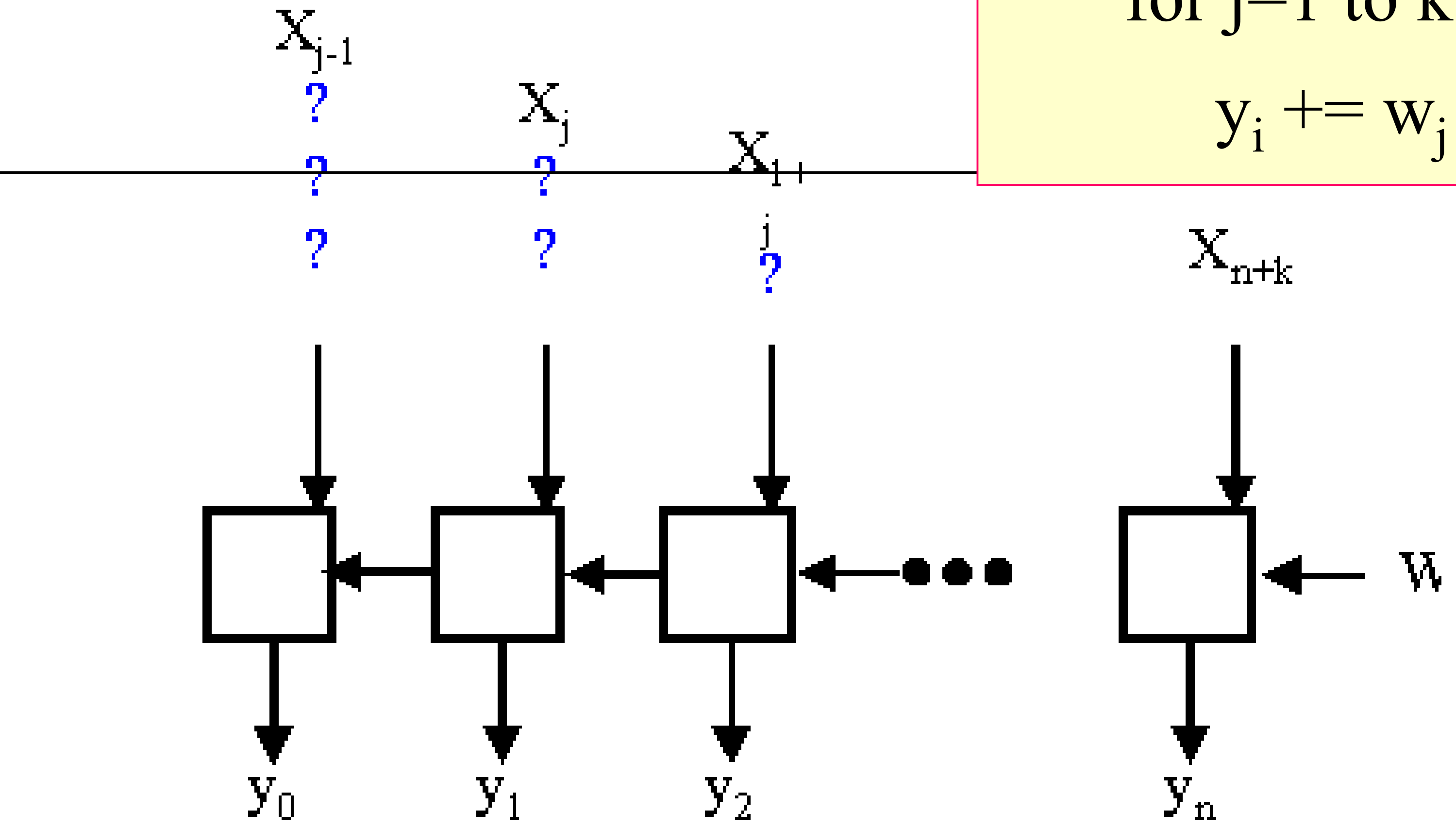
From in parallel



Broadcast common input

# Bogus Attempt: Outer Loop - 2

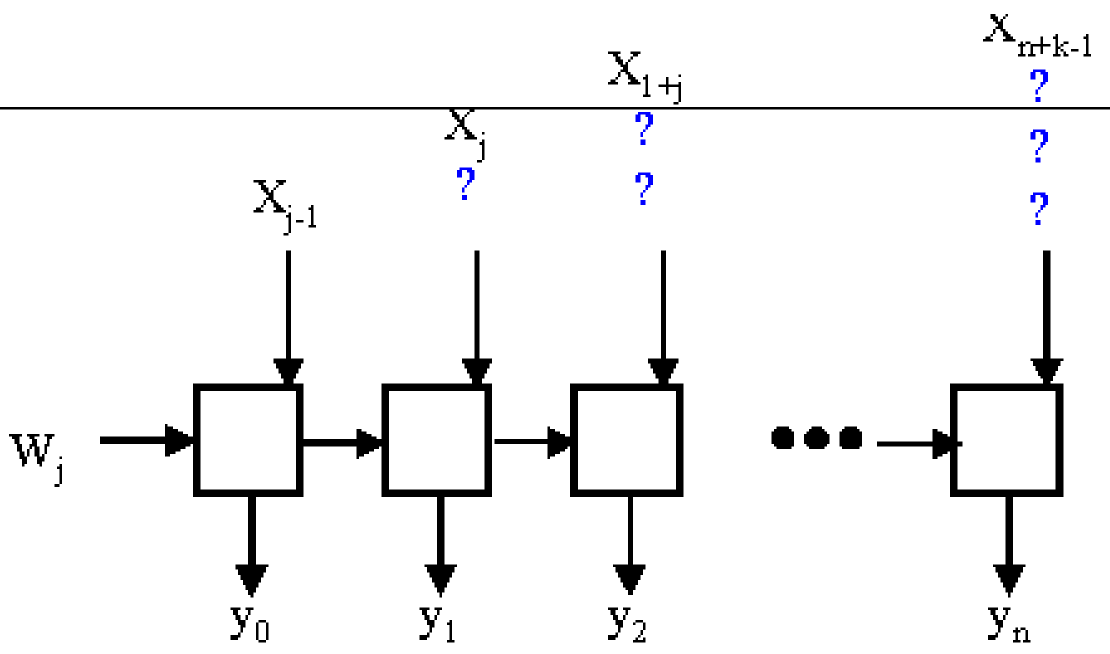
```
for i=1 to n in parallel
    for j=1 to k in place
        $y_i$ += $w_j$ * $x_{i+j-1}$
```



Retime to eliminate broadcast

# Bogus Attempt: Outer Loop - 2a

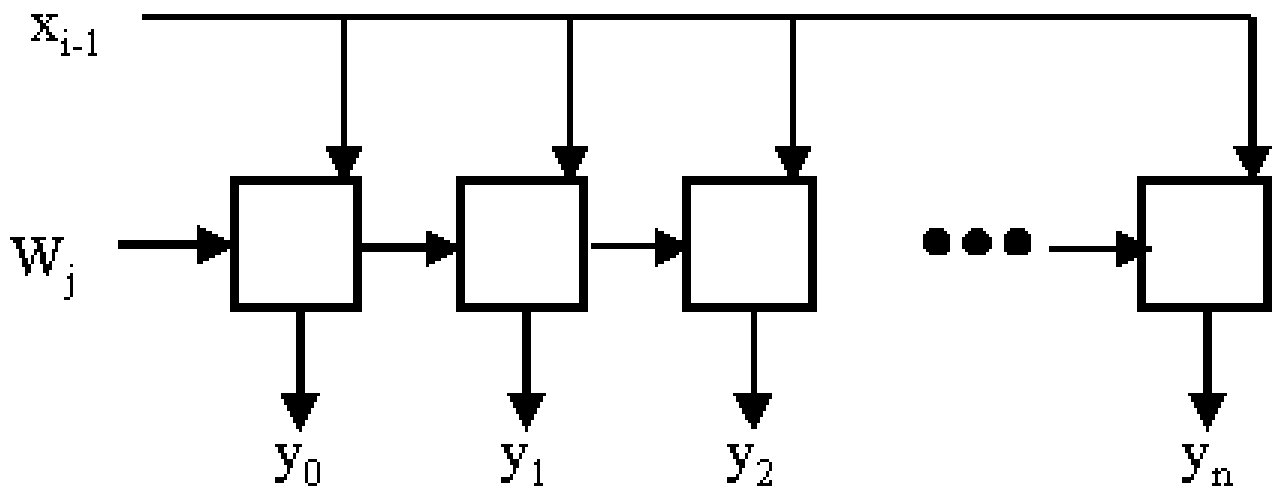
```
for i=1 to n in parallel
    for j=1 to k in place
        $y_i$ += $w_j$ * $x_{i+j-1}$
```



Retime to eliminate broadcast

# Bogus Attempt: Outer Loop - 3

```
for i=1 to n in parallel
    for j=1 to k in place
        $y_i$ += $w_j$ * $x_{i+j-1}$
```

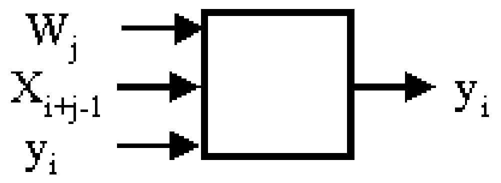


Broadcast common input

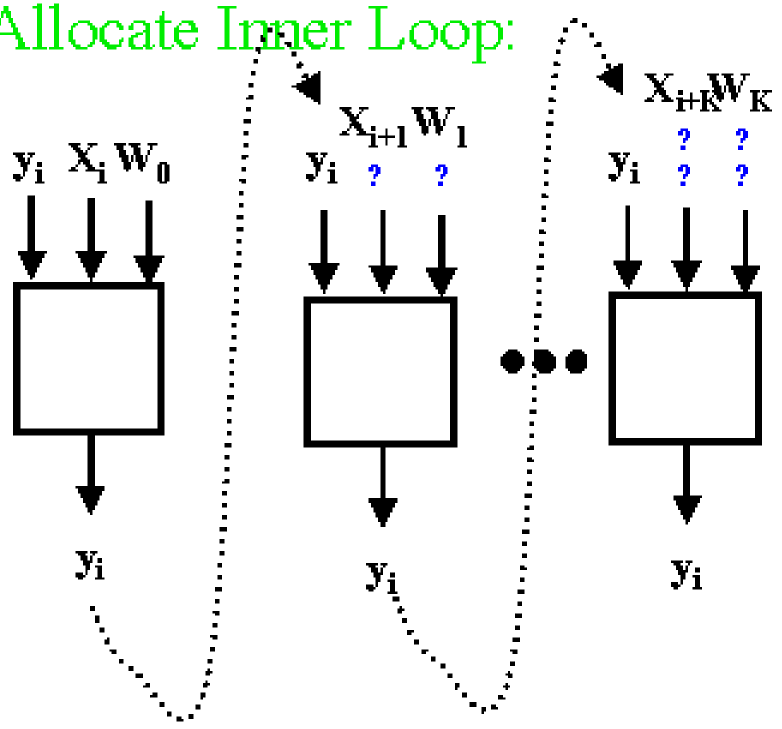
# Attempt at Systolic FIR

```
for i=1 to n in place
    for j=1 to k in parallel
        $y_i$ += $w_j$ * $x_{i+j-1}$
```

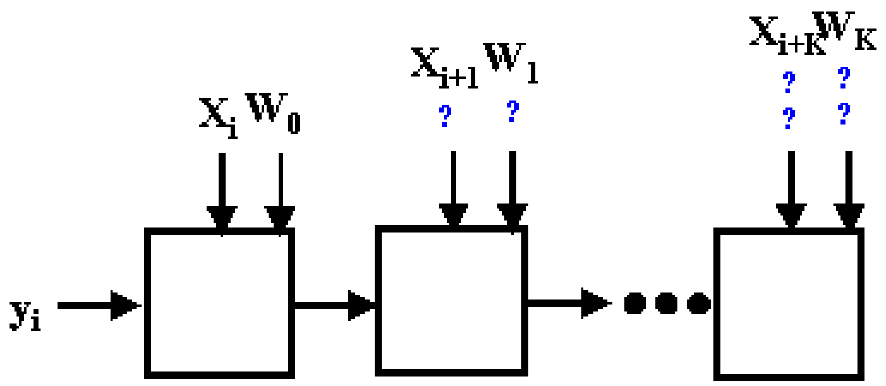
Allocate Inner Loop Body:



Allocate Inner Loop:



Internalize Dataflow:

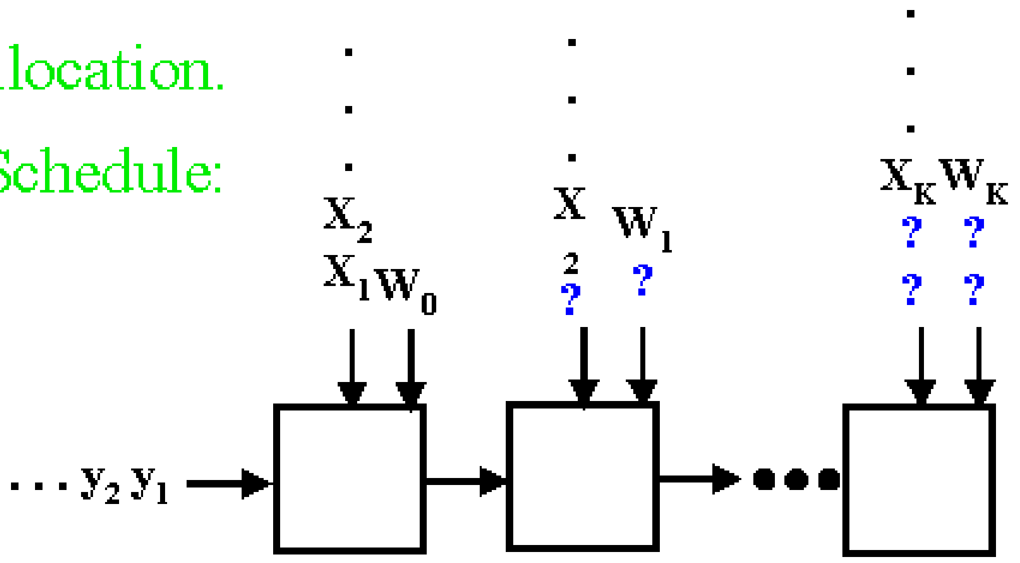


# Outer Loop

```
for i=1 to n in place
    for j=1 to k in parallel
        $y_i$ += $w_j$ * $x_{i+j-1}$
```

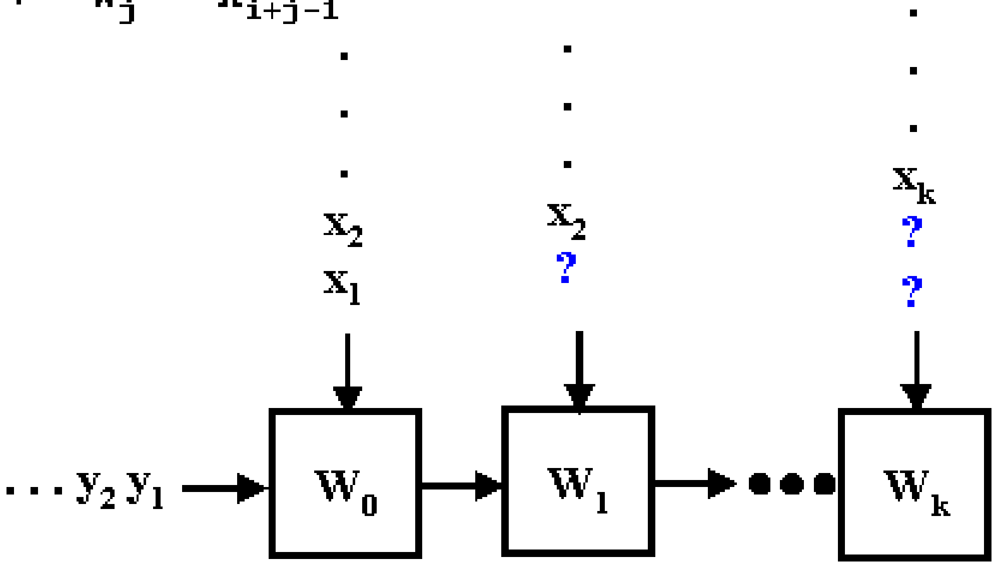
Allocation.

Schedule:



# Optimize Outer Loop
# Preload-repeated Value

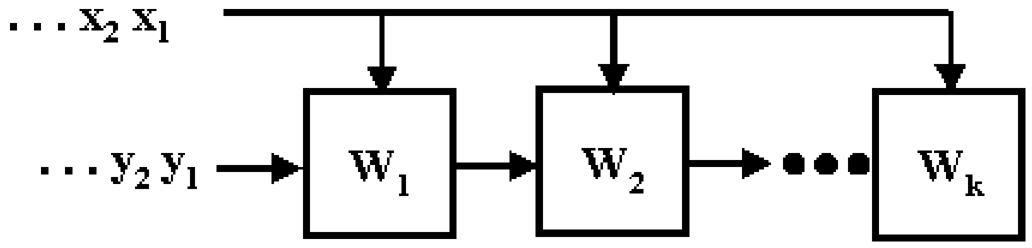
```
for i=1 to n in place
    for j=0 to k in parallel
        $y_i$ += $w_j$ * $x_{i+j-1}$
```



# Optimize Outer Loop Broadcast Common Value

```
for i=1 to n in place
    for j=1 to k in parallel
        $y_i$ += $w_j$ * $x_{i+j-1}$
```

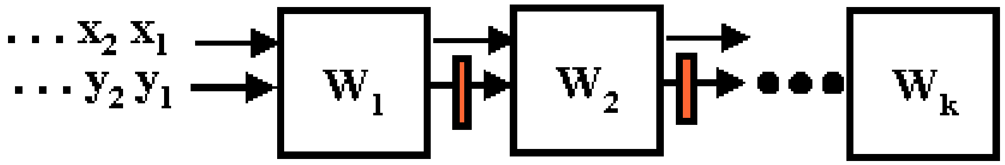
B1 From [Kung82]



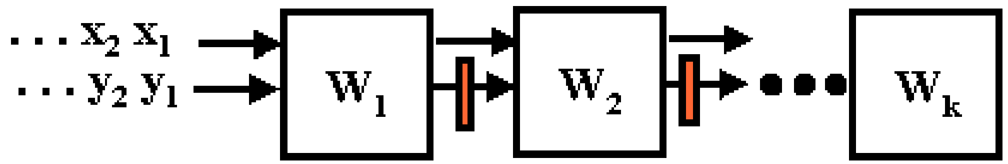
# Optimize Outer Loop
# Retime to Eliminate Broadcast

```
for i=1 to n in place
    for j=1 to k in parallel
        $y_i$ += $w_j$ * $x_{i+j-1}$
```

Similar to W2 From [Kung82]

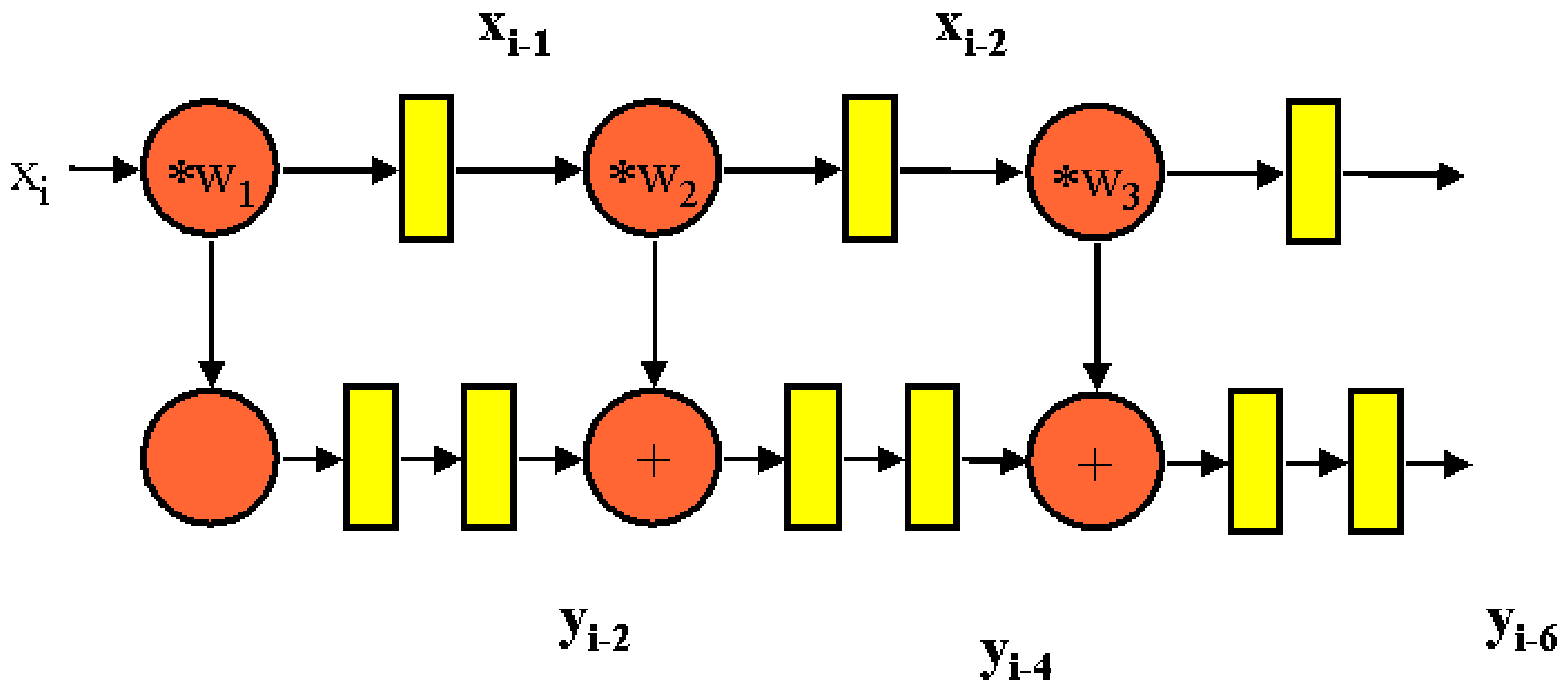


# How it works

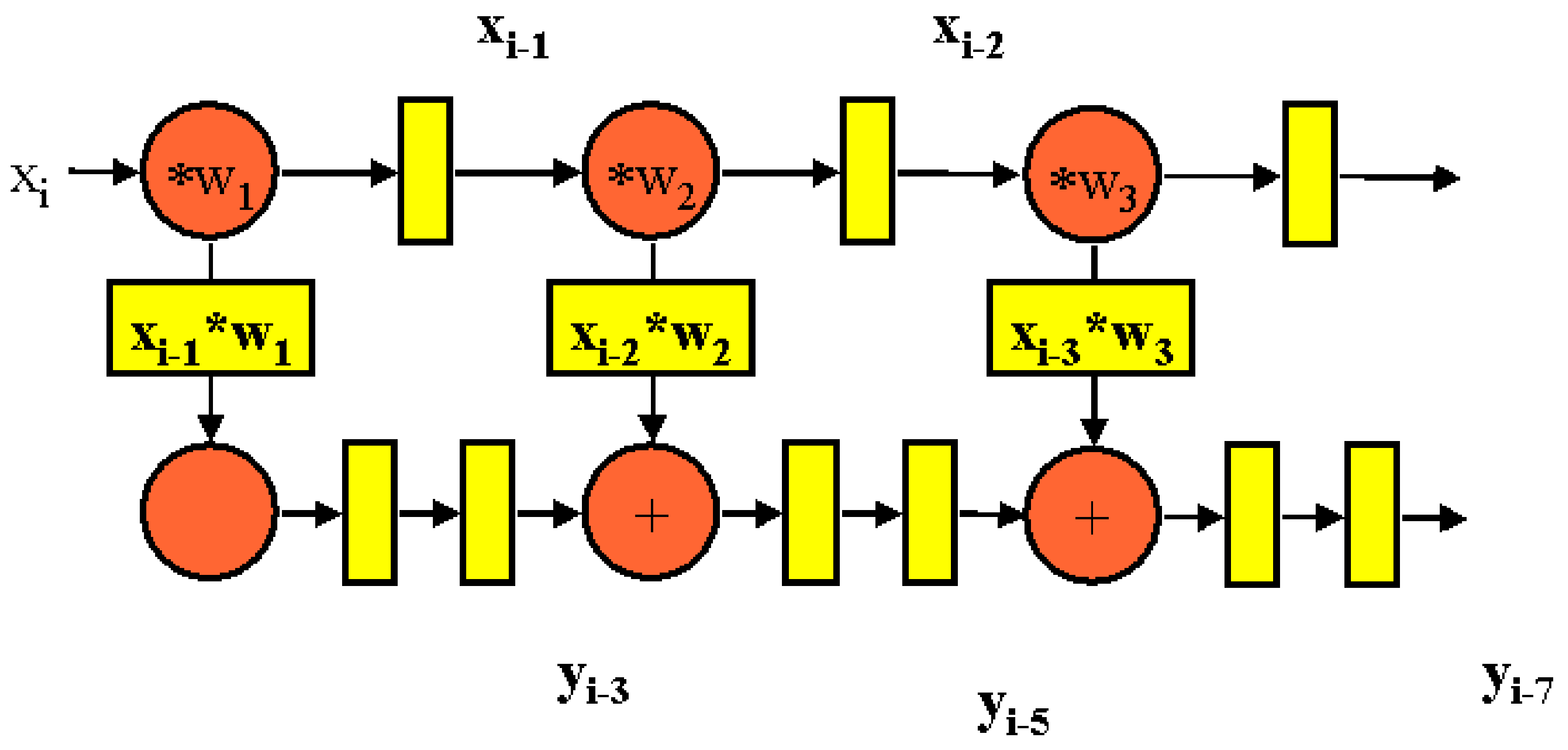


| | | | | |
|---|---|---|---|---|
| $y_1$ | | | | |
| $x_1$ | | | | |
| $y_2$ | $y_1$ | | | |
| $x_2$ | | $x_1$ | | |
| $y_3$ | $y_2$ | $y_1$ | | |
| $x_3$ | | $x_2$ | | $x_1$ |
| $y_4$ | $y_3$ | $y_2$ | $y_1$ | |
| $x_4$ | | $x_3$ | | $x_2$ |
| $y_5$ | $y_4$ | $y_3$ | $y_2$ | $y_1$ |
| $x_5$ | | $x_4$ | | $x_3$ |

# FIR details



# FIR details



# FIR Summary

- Sequential:
  - Memory Bandwidth per output: 3k+1
    each x k times. k w's. plus one y.
  - 1 result o(k) cycles
  - o(1) hardware
- Systolic:
  - Memory Bandwidth per output: 2
    1 x + 1 y.
  - 1 result every cycle - o(1)
  - o(k) hardware

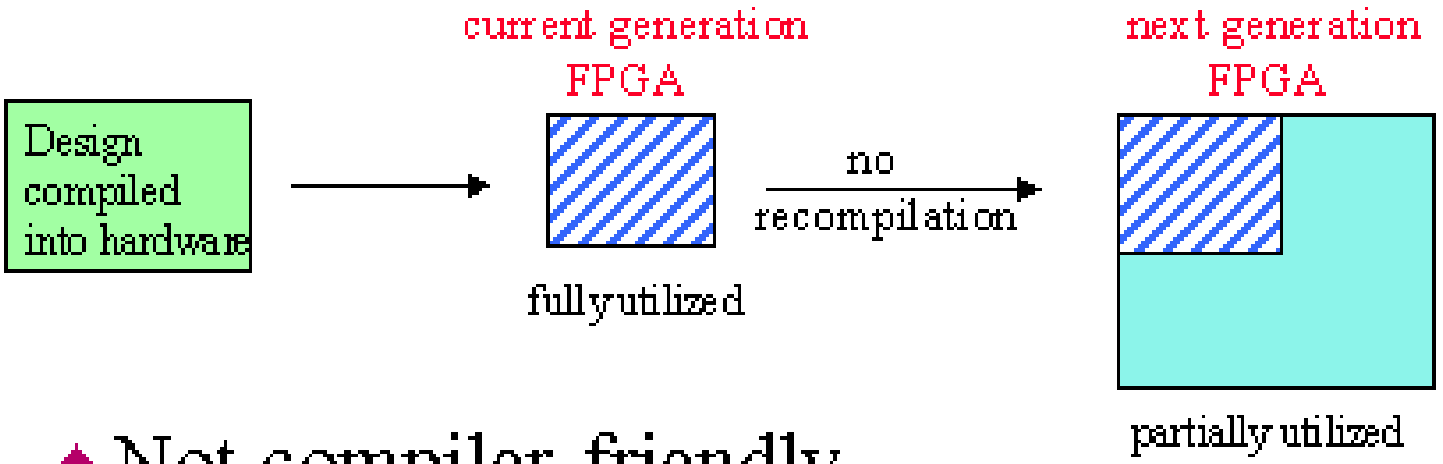
## Example:

## Pipeline-Reconfigurable FPGAs

# Reconfigurable Computing

Two big problems

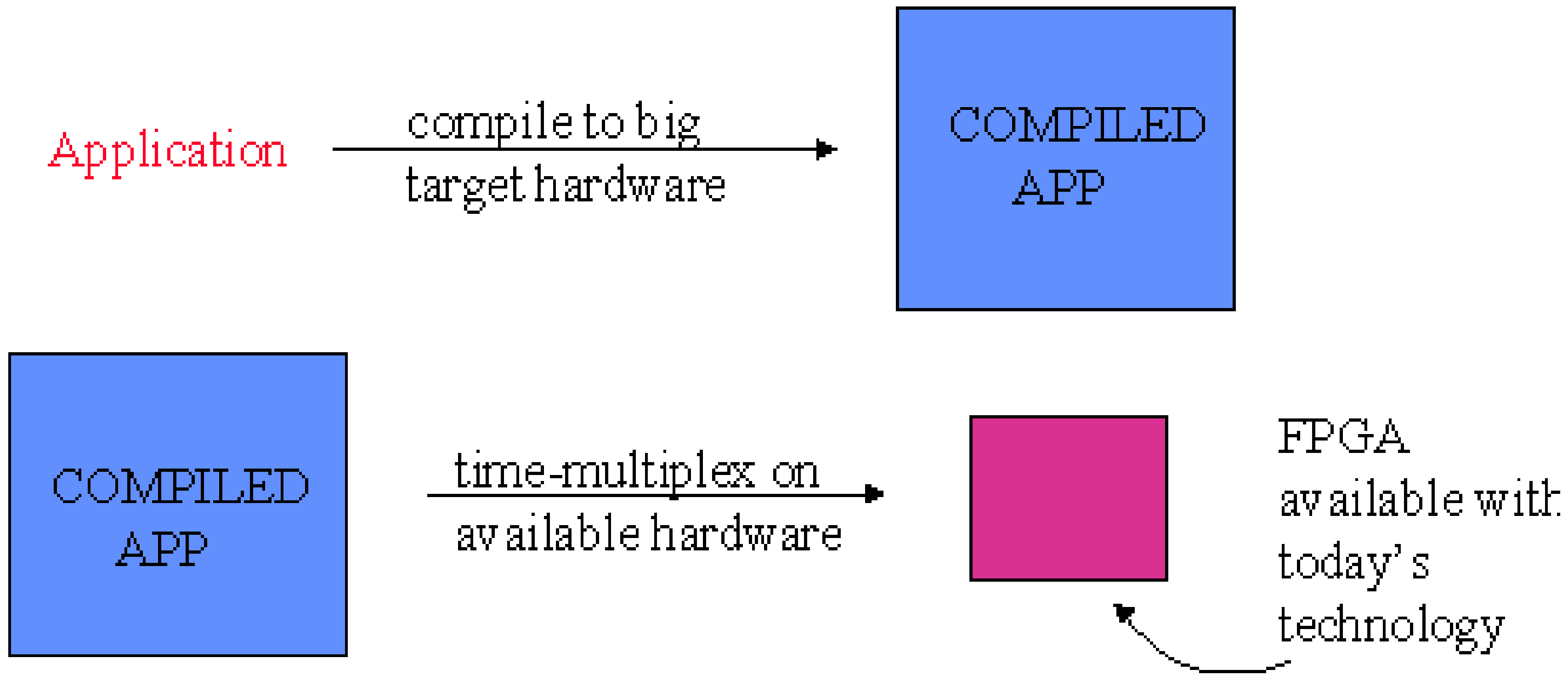
- No forward compatibility
  - performance does not scale with technology



- Not compiler-friendly
  - applications have to fit in hardware!

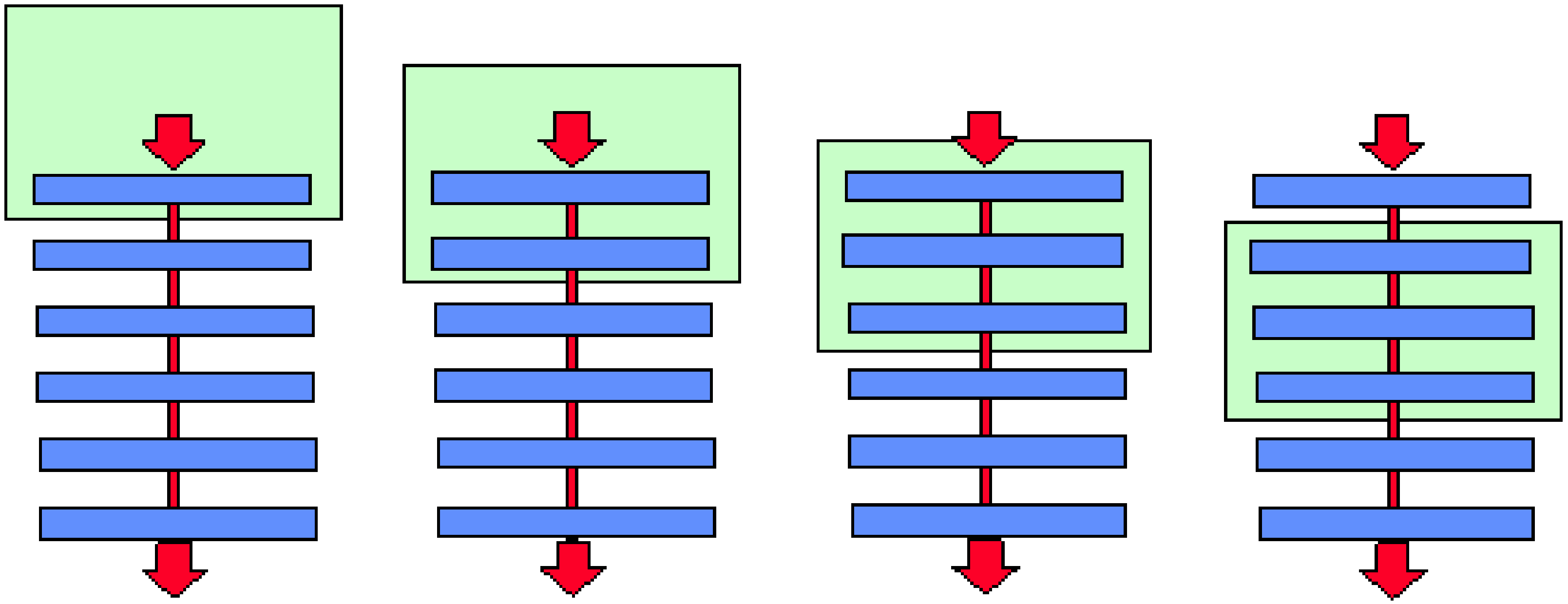
# Solution based on Hardware Virtualization

- assume you have a target hardware that fits the entire application at once


Application
compile to big target hardware
COMPILED APP
COMPILED APP
time-multiplex on available hardware
FPGA available with today's technology

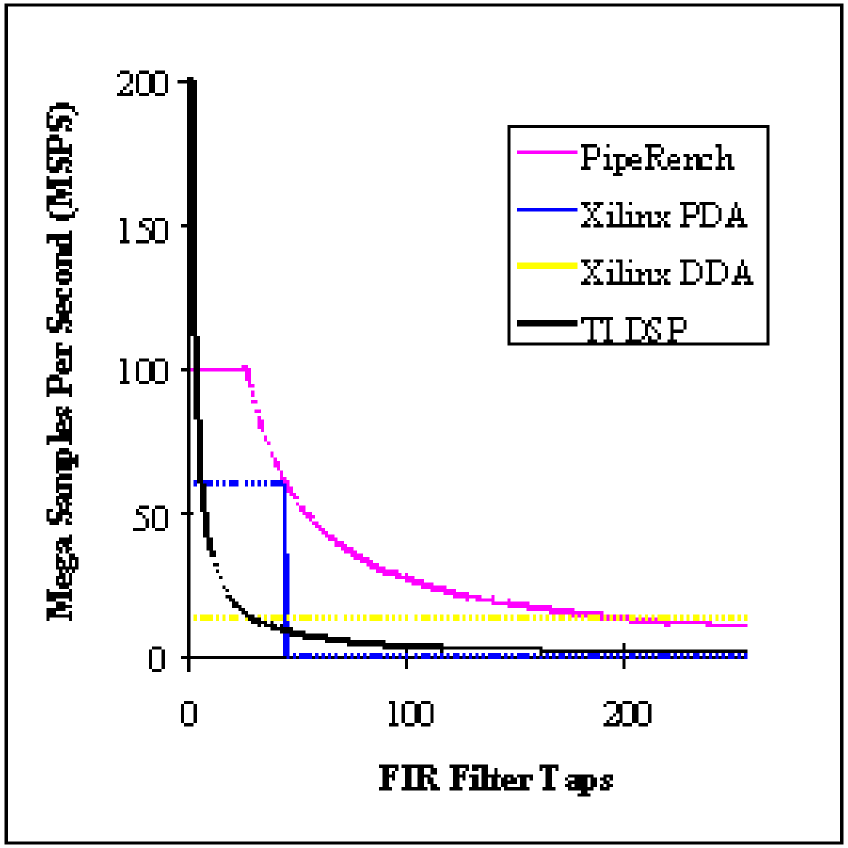
# Making it happen

Incremental Reconfiguration: “scrolling” FPGA fabric



# How good is this concept?

- CMU PipeRench
  - 100 sq. mm die area
  - 128-bit datapath
  - 100 MHz clock
  - 0.35 um technology
- Key Points
  - performance like FPGA
  - degradation like DSP



## What’s coming your way

- **What’s required to make hardware virtualization work?**
- **We will use an example to illustrate**
  - breaks in the input data stream
  - saving and restoring lost state

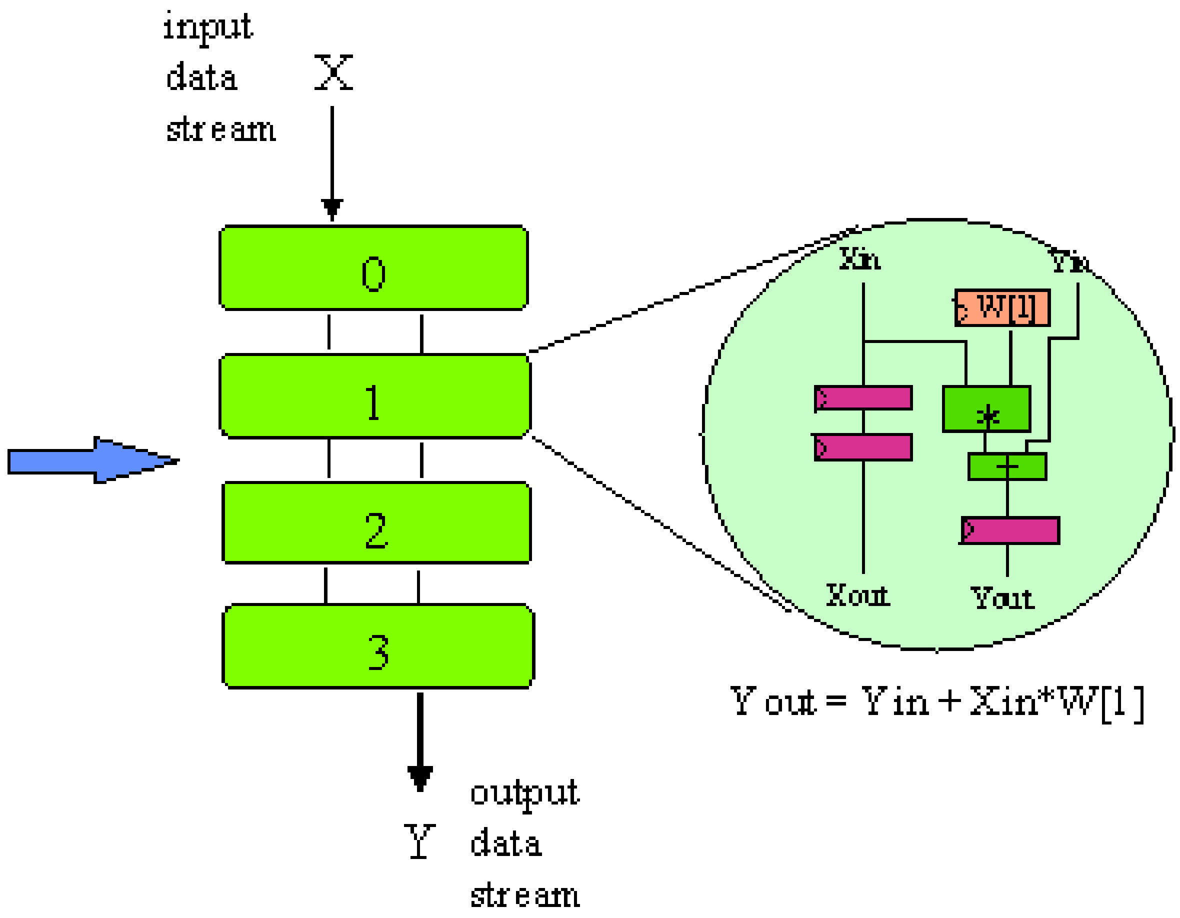
## Example: Convolution Algorithm

```
// 4-tap FIR filter
// 7 input data elements X[0] - X[6]
// 4 output data elements Y[0] - Y[3]
// 4 constant weights W[0] - W[3]

for i = 0 to 3   {
    Y[i] = 0;
    for j = 0 to 3   {
        Y[i] += X[i+j] * W[j];
    }
}
```

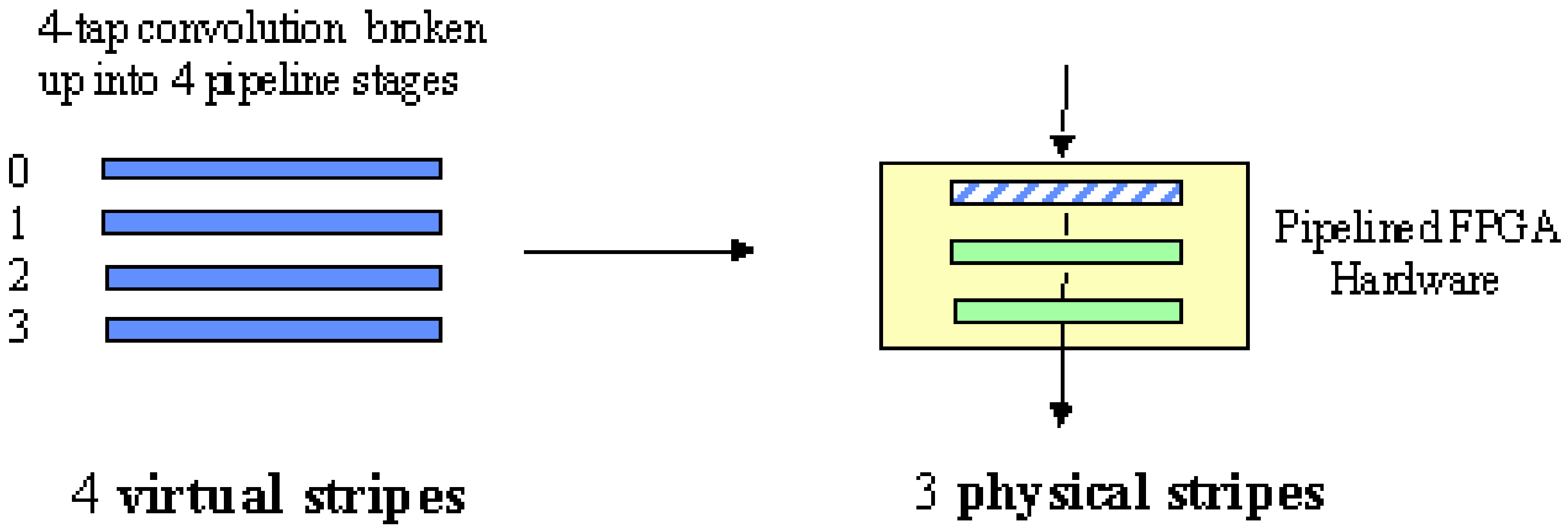
# A Systolic Implementation

```
for i = 0 to 3  {
   Y[i] = 0;
   for j = 0 to 3  {
        Y[i] += X[i+j] * W[j];  }
}
```



Y out = Y in + Xin*W[1]

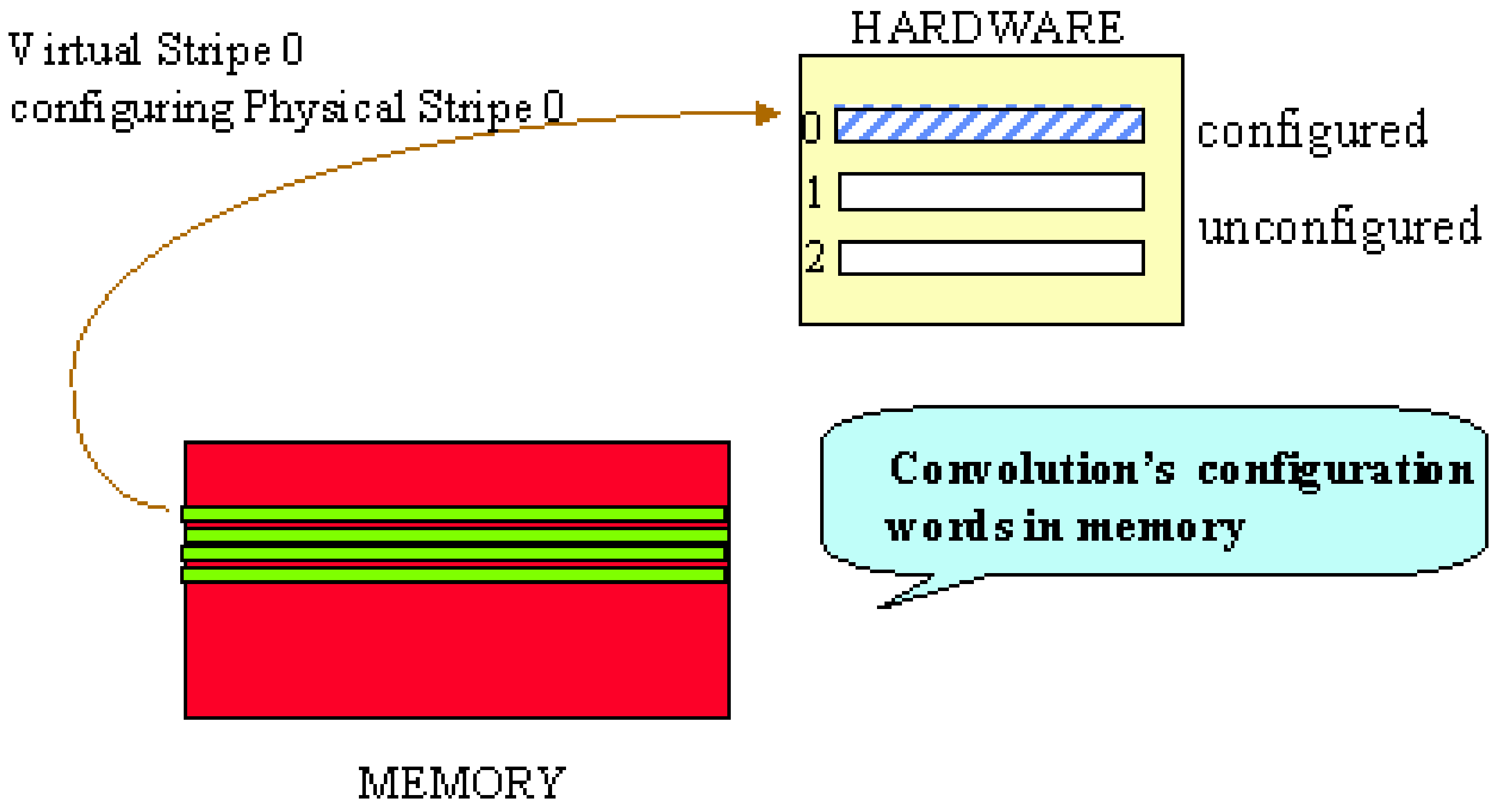
# Target Hardware



- Pipelined
- Concurrent reconfiguration and execution
  - One pipestage reconfigured each cycle
  - Other stages may execute

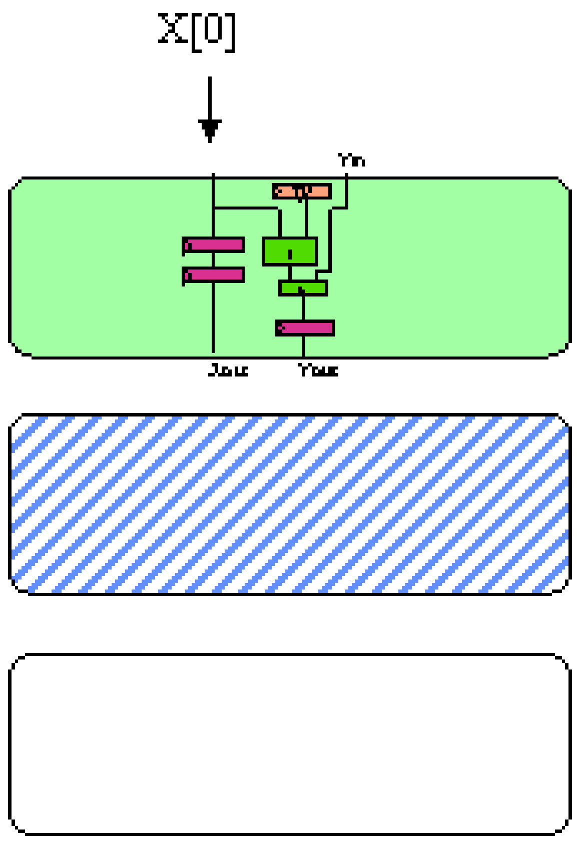
# Clock Cycle 0

- Configure next available physical stripe

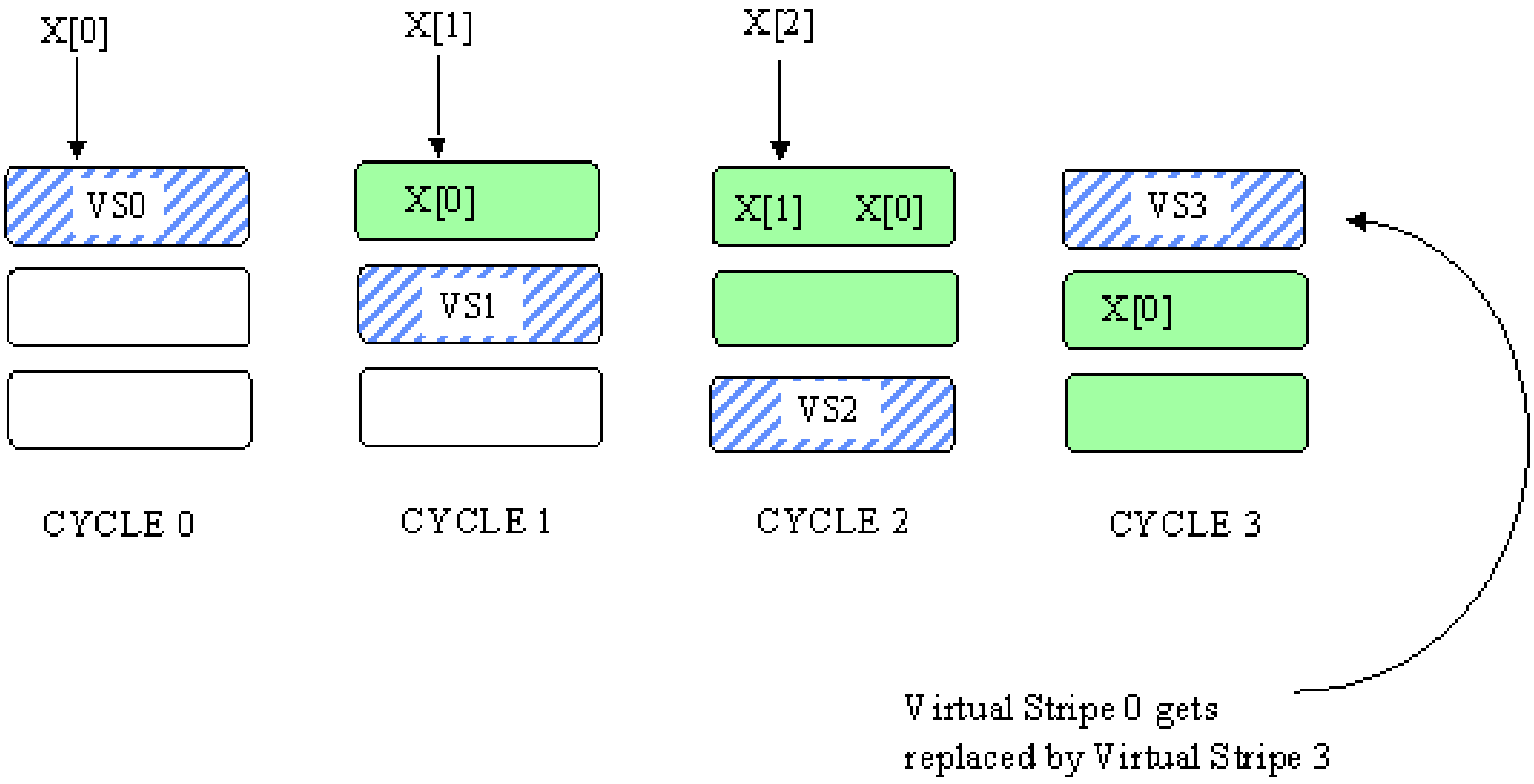

Virtual Stripe 0
configuring Physical Stripe 0
HARDWARE
0
1
2
configured
unconfigured
Convolution's configuration words in memory
MEMORY

# Clock Cycle 1

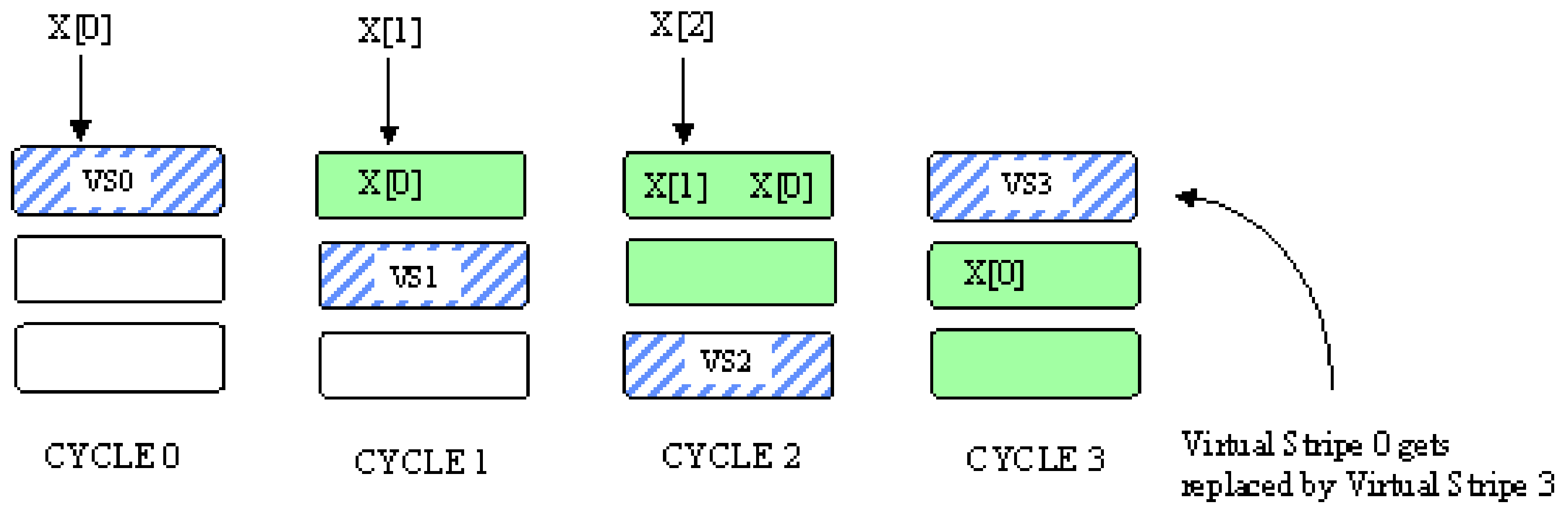
- Physical Stripe 0
  - configured to behave like Virtual Stripe 0
  - now executing
  - needs data X[0]
- Physical Stripe 1 being configured...



# The first four cycles



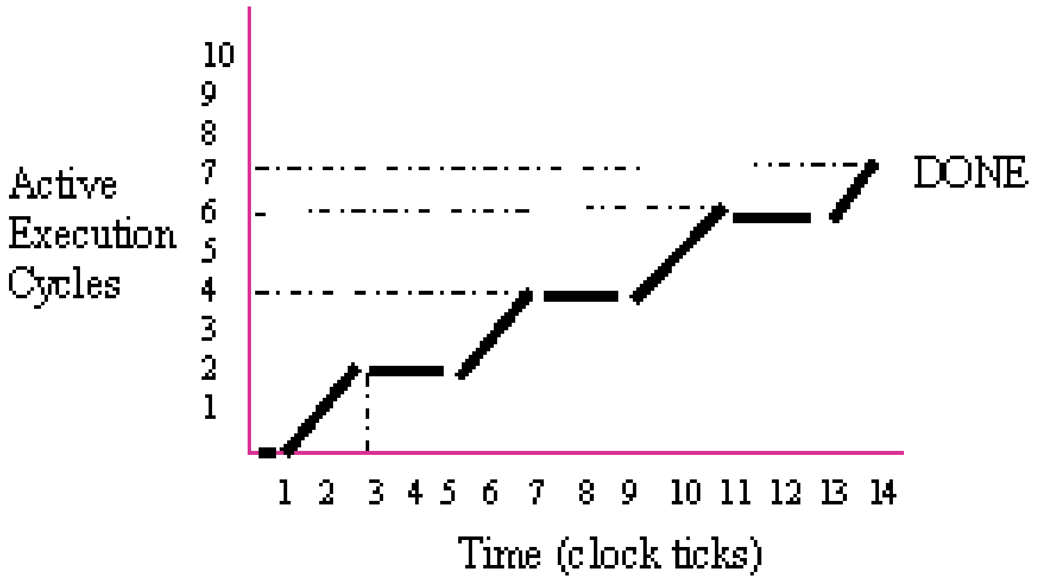
# Pipeline Disruption



- break in data stream: input has to stop
- only done with X[1] and X[0].
  - needs to process X[2]-X[6]
  - 5 more execution cycles to go

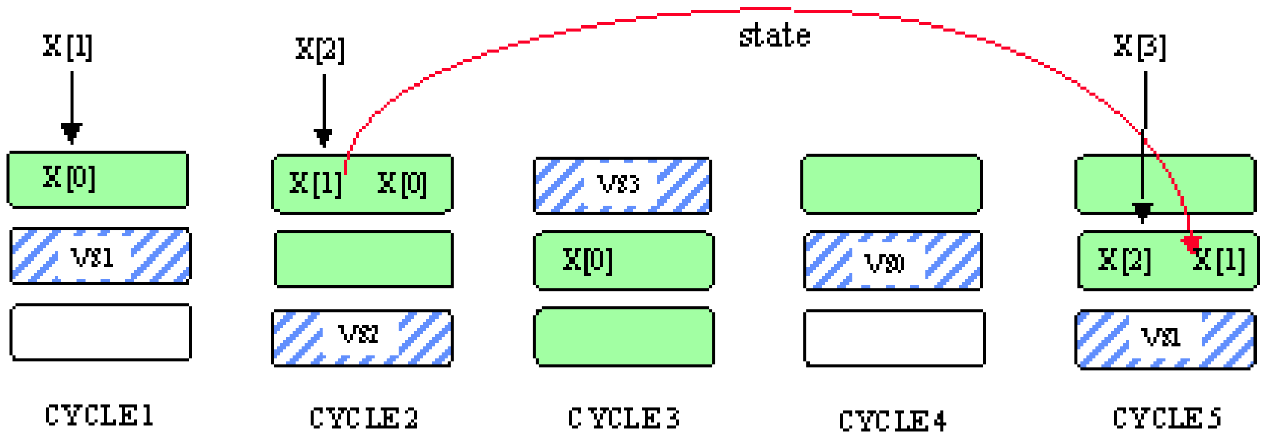
# The life of Virtual Stripe 0

- **Behavior a function of application and architecture**
  - With V virtual and P physical stripes
    - execution cycles at a stretch = P - 1
    - swap-out duration = V - P + 1

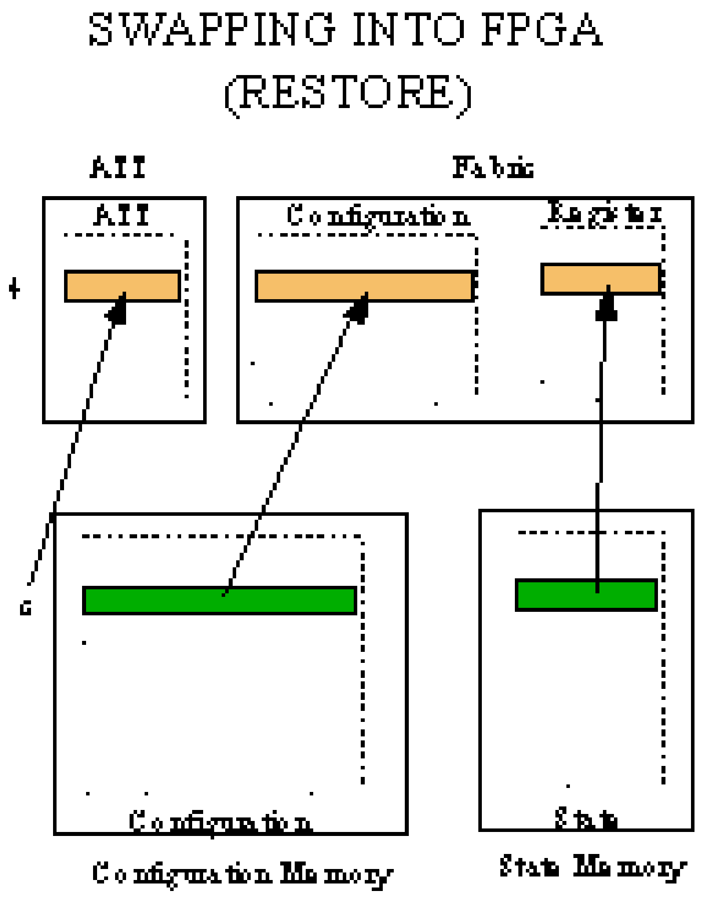


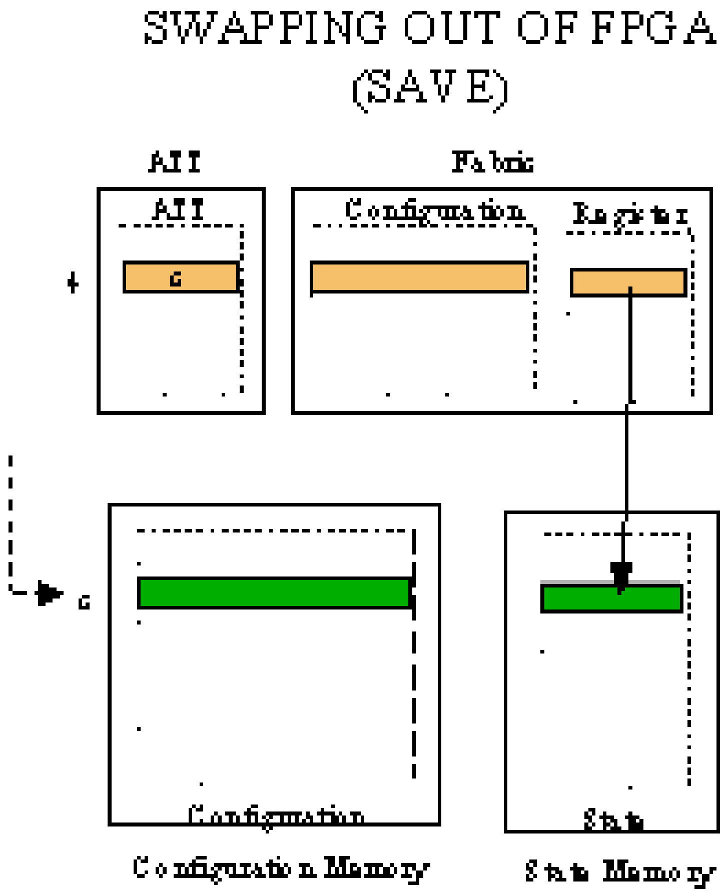
# Further complications!

- **Virtual Stripe 0 has state**
  - required when it resumes executing
  - need to save and restore it when stripe is configured back in



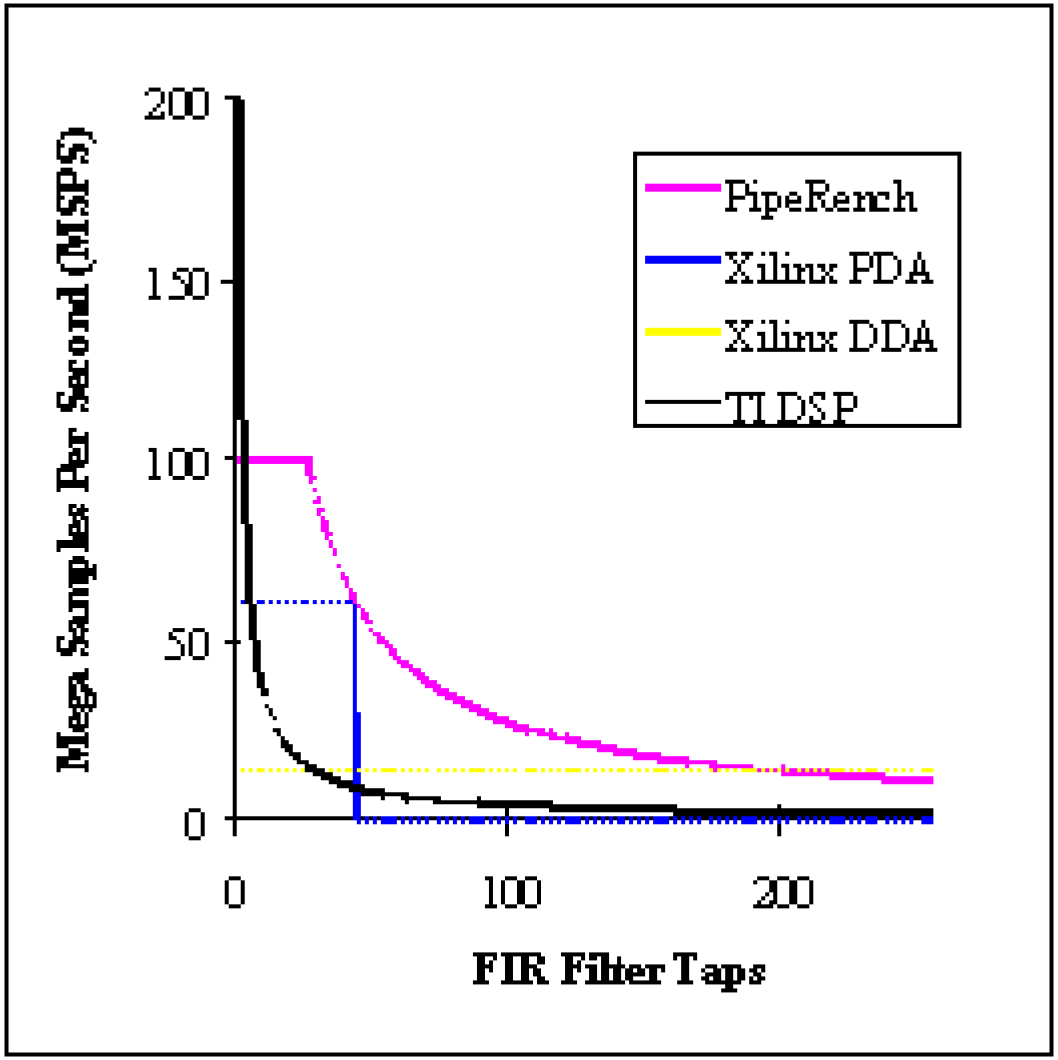
# Implementing Save/Restore



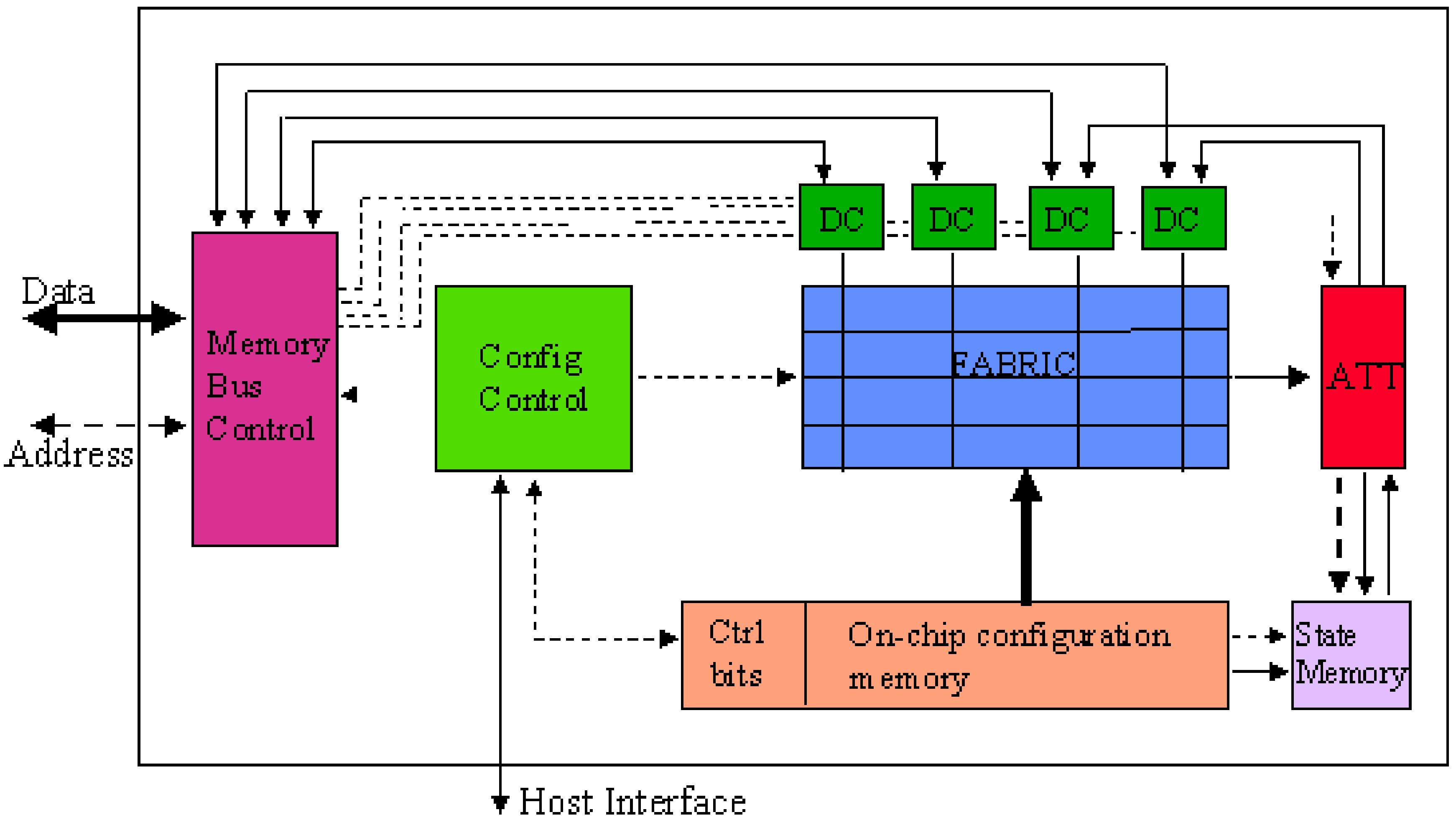


# Conclusions

- **Pipeline-reconfigurable FPGAs provide**
  - forward compatibility
  - robust compilation

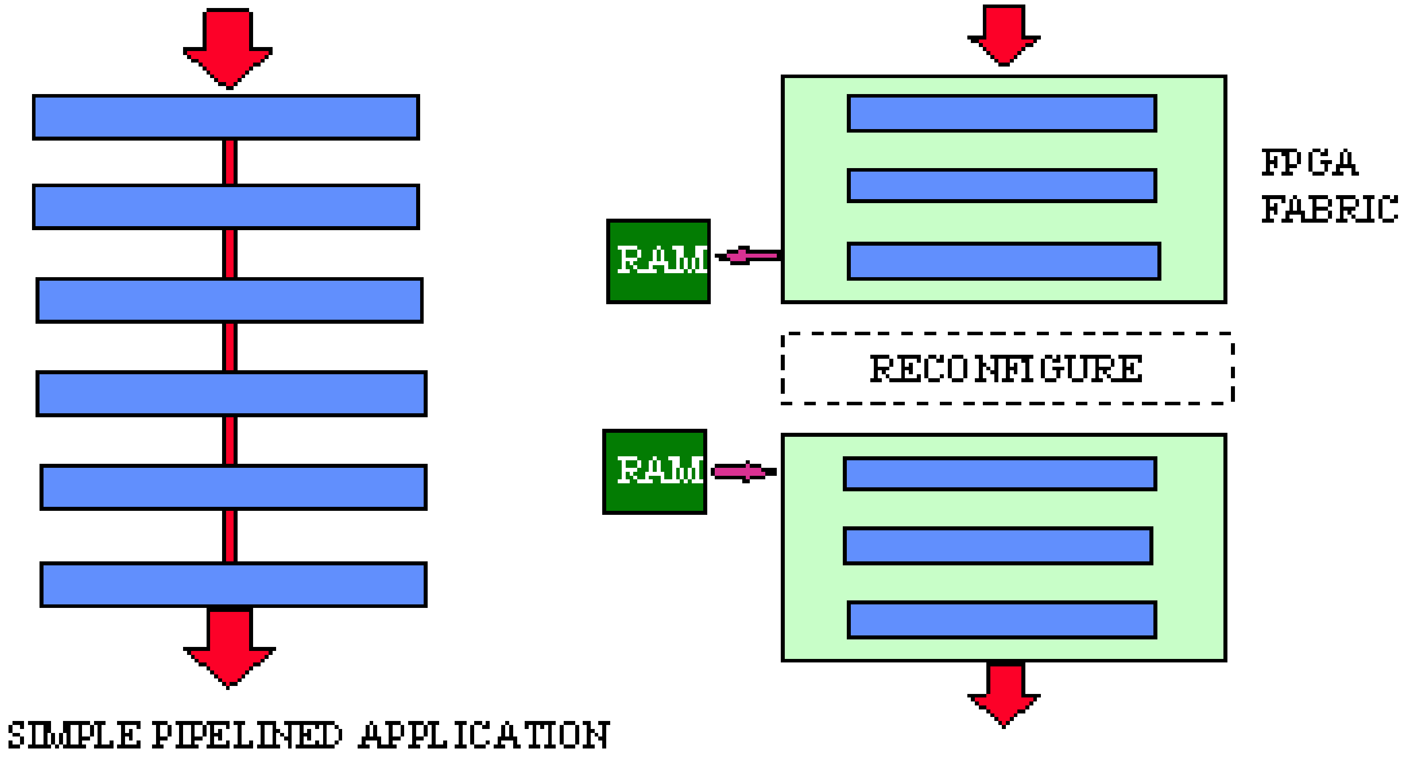


# PipeRench-1


Data
Address
Memory Bus Control
Config Control
DC
DC
DC
DC
FABRIC
ATT
Ctrl bits
On-chip configuration memory
State Memory
Host Interface

# Implementing hardware virtualization

- Reconfigure entire FPGA
  - need more reconfiguration time
  - need intermediate storage


SIMPLE PIPELINED APPLICATION
FPGA FABRIC
RAM
RECONFIGURE
RAM