Scheduling and Resource Allocation

Shao-Yi Chien

Scheduling and Resource Allocation

- **Scheduling** and **resource allocation** are two important tasks in hardware or software synthesis of DSP systems
- Scheduling – **when** to do the process?
  - Assign every node of the DFG to a control time step, the fundamental sequencing units in synchronous systems and correspond to clock cycles
- Resource allocation – **who** to execute the process?
  - Assign operations to hardware with the goal of minimizing the amount of hardware required to implement the desired behavior

Scheduling

- Static scheduling
  - If all processes are known in advance
  - Perform scheduling before run time
  - Most DSP algorithms are amenable to static scheduling
- Dynamic scheduling
  - When the process characteristics are not completely known
  - Decide dynamically at run time by scheduler that runs concurrently with the program

Criteria of Scheduling Optimization (1/2)

- Sample period optimal
  - Optimal if sample period = iteration bound
- Delay optimal
  - Optimal if delay = delay bound, the shortest possible delay from input to output of the algorithm
- Resource optimal
  - Minimum amount of resource
  - Processor based system corresponds to a resource limited scheduling problem

Criteria of Scheduling Optimization (2/2)

- Processor optimal
  - If it uses as minimum number of PEs of each type, processor bound

$$P_i = \left\lceil \frac{D_{op\ i}}{T_{\min}} \right\rceil$$

P_i : Processor bound of the processing element of type i

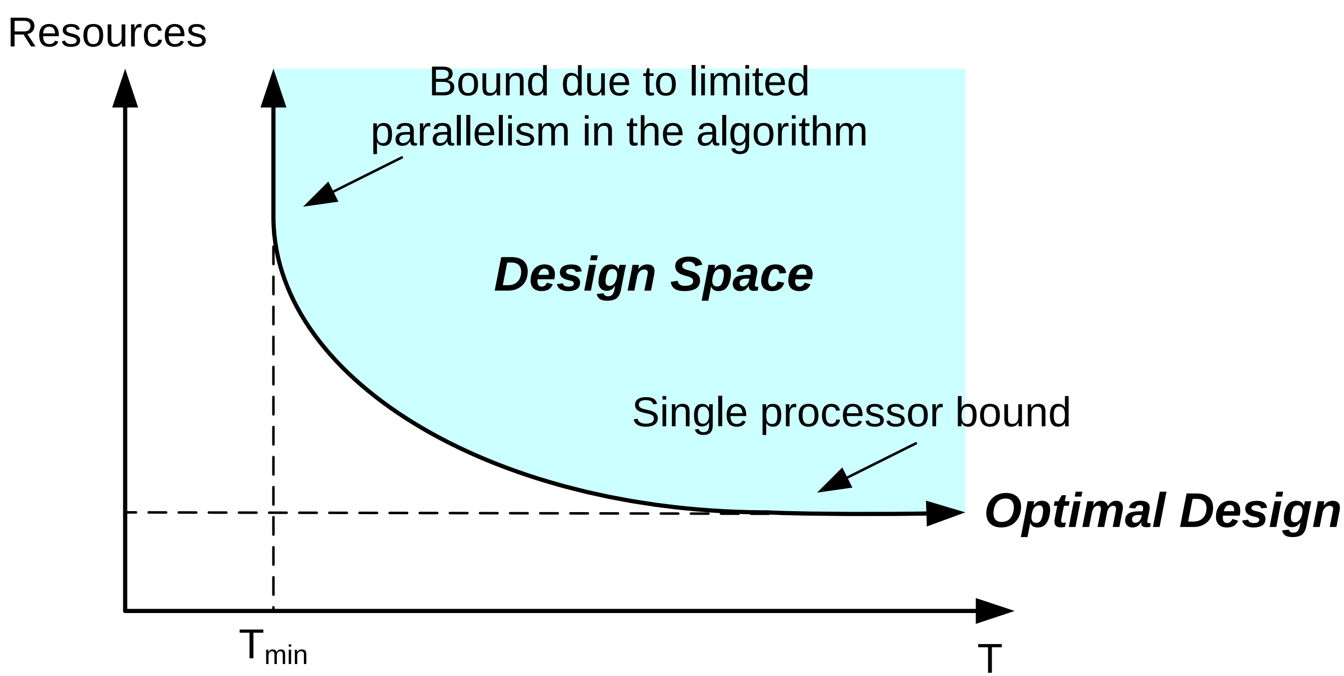
$D_{op\ i}$: Total execution time for all operations of type i

$T_{\min}$: Minimal sample period

- Memory optimal
  - Minimum amount of memory

Scheduling Formulations (1/2)

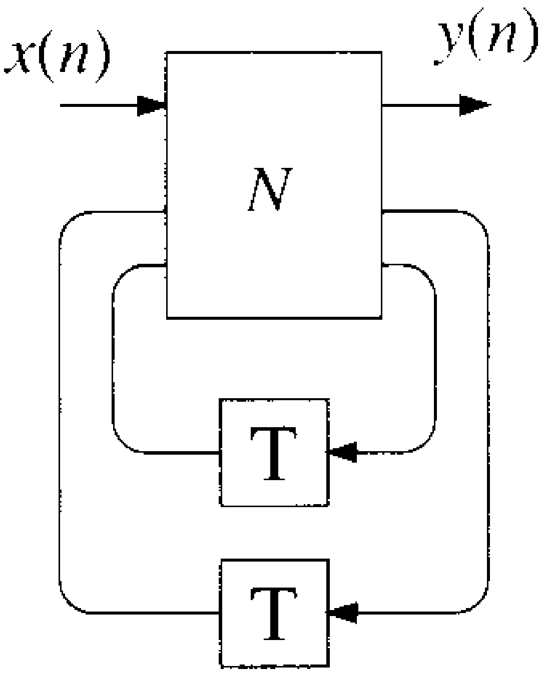
- Resource v.s. sample rate for different schedules



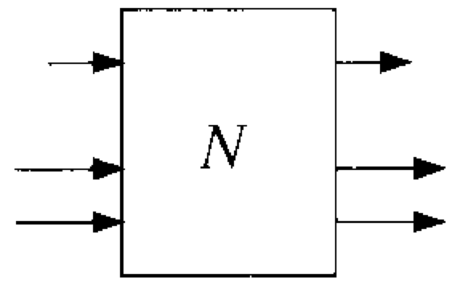
Scheduling Formulations (2/2)

- Single interval formulation
- Block formulation
- Loop-folding
- Periodic formulation

Single Interval Formulation (1/3)



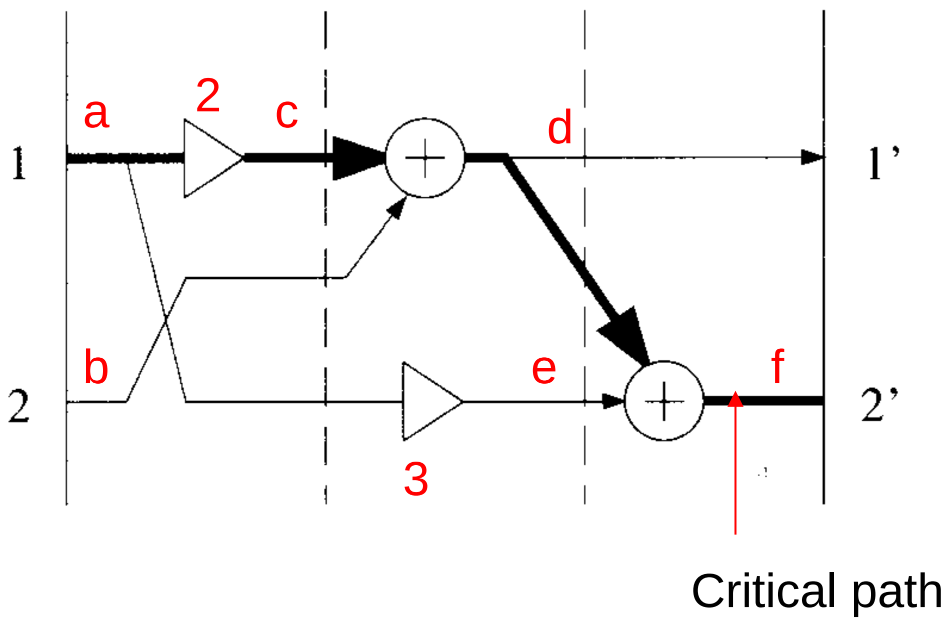
DFG



Computation graph,
DAG (directed acyclic graph):
Combinational circuits in a single interval

Single Interval Formulation (2/3)

- Schedule with uniform boundaries
  (assume the processor has one multiplier PE and one adder PE)



Sample period = 3

For i=0→∞
{
    a(i)=d(i-1)
    b(i)=f(i-1)

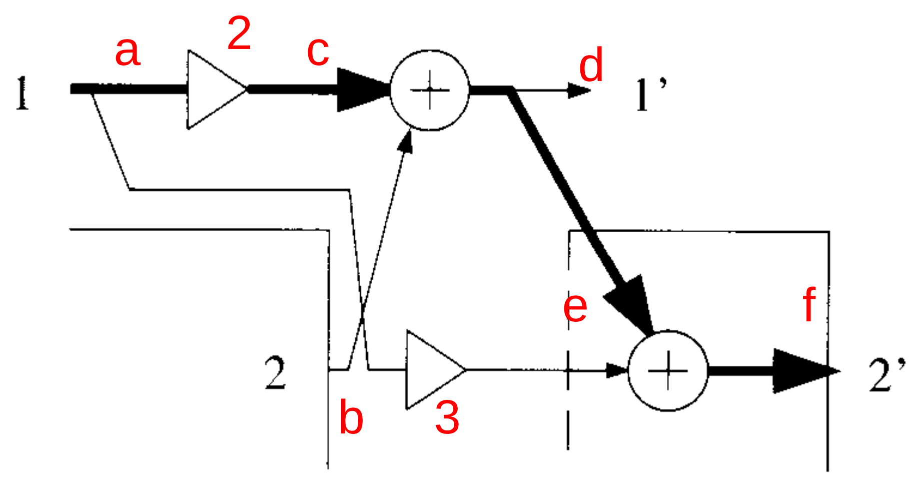
    c(i)=a(i)*2

    d(i)=c(i)+b(i)
    e(i)=a(i)*3

    f(i)=d(i)+e(i)
}

Single Interval Formulation (3/3)

- Schedule with nonuniform boundaries



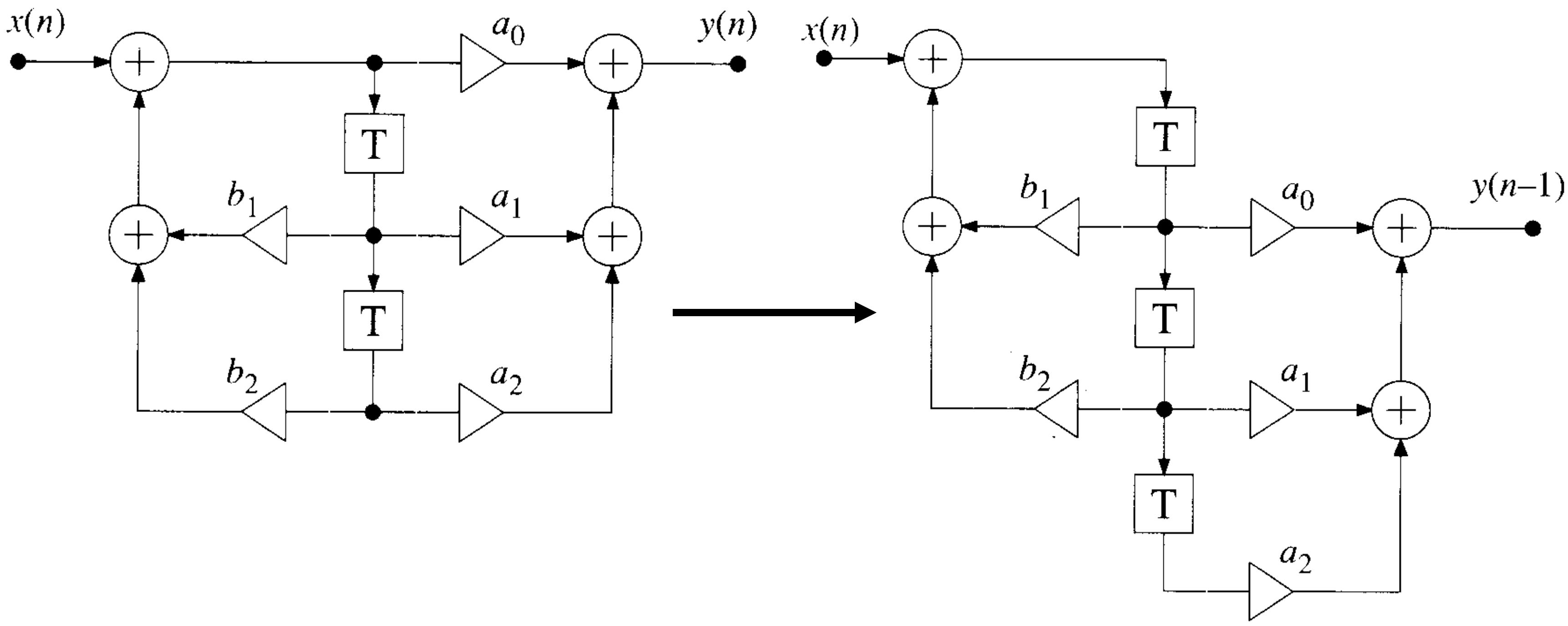
Sample period = 2

For i=0→∞
{
a(i)=d(i-1)
c(i)=a(i)*2
f(i-1)=d(i-1)+e(i-1)

b(i)=f(i-1)
d(i)=c(i)+b(i)
e(i)=a(i)*3
}

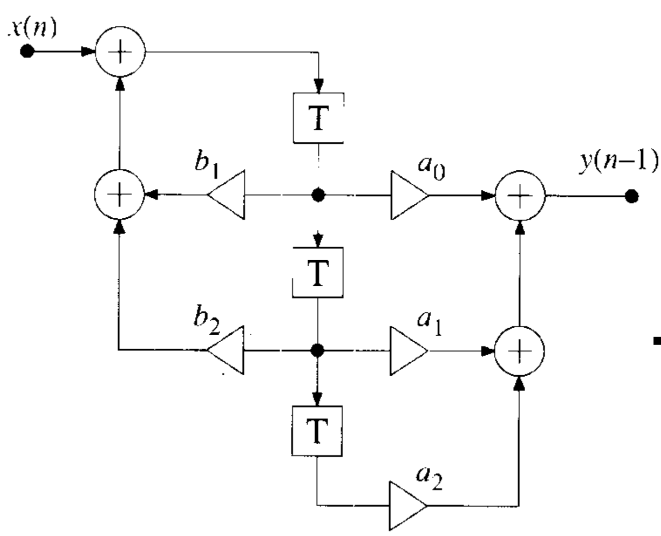
Example (1/5)

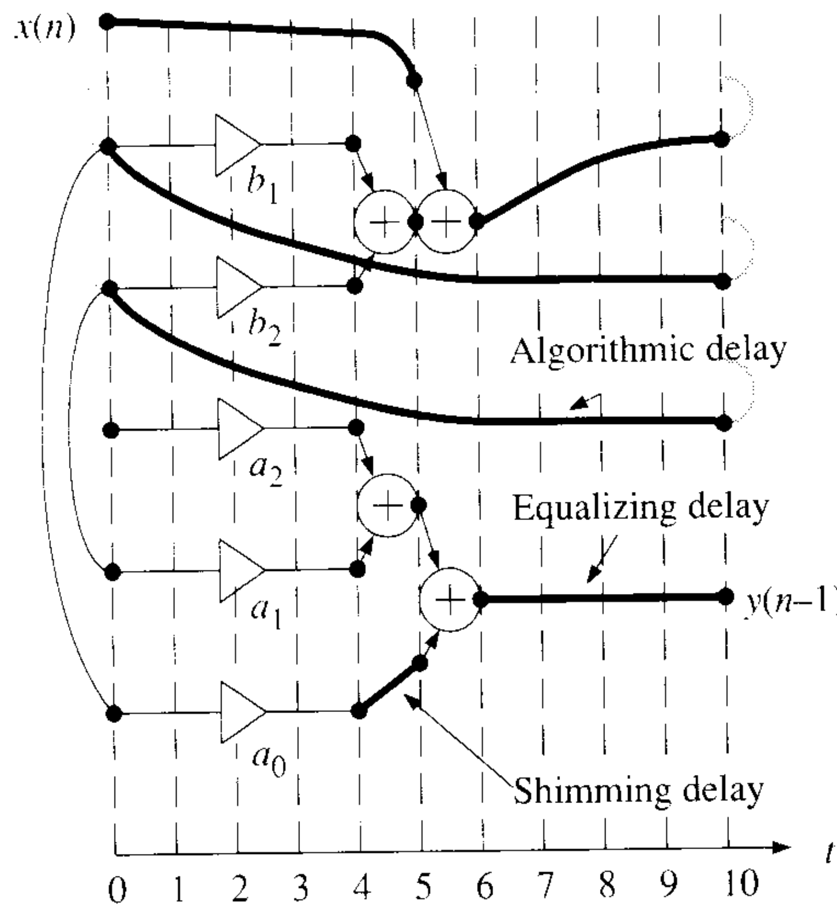
- DFG



Example (2/5)

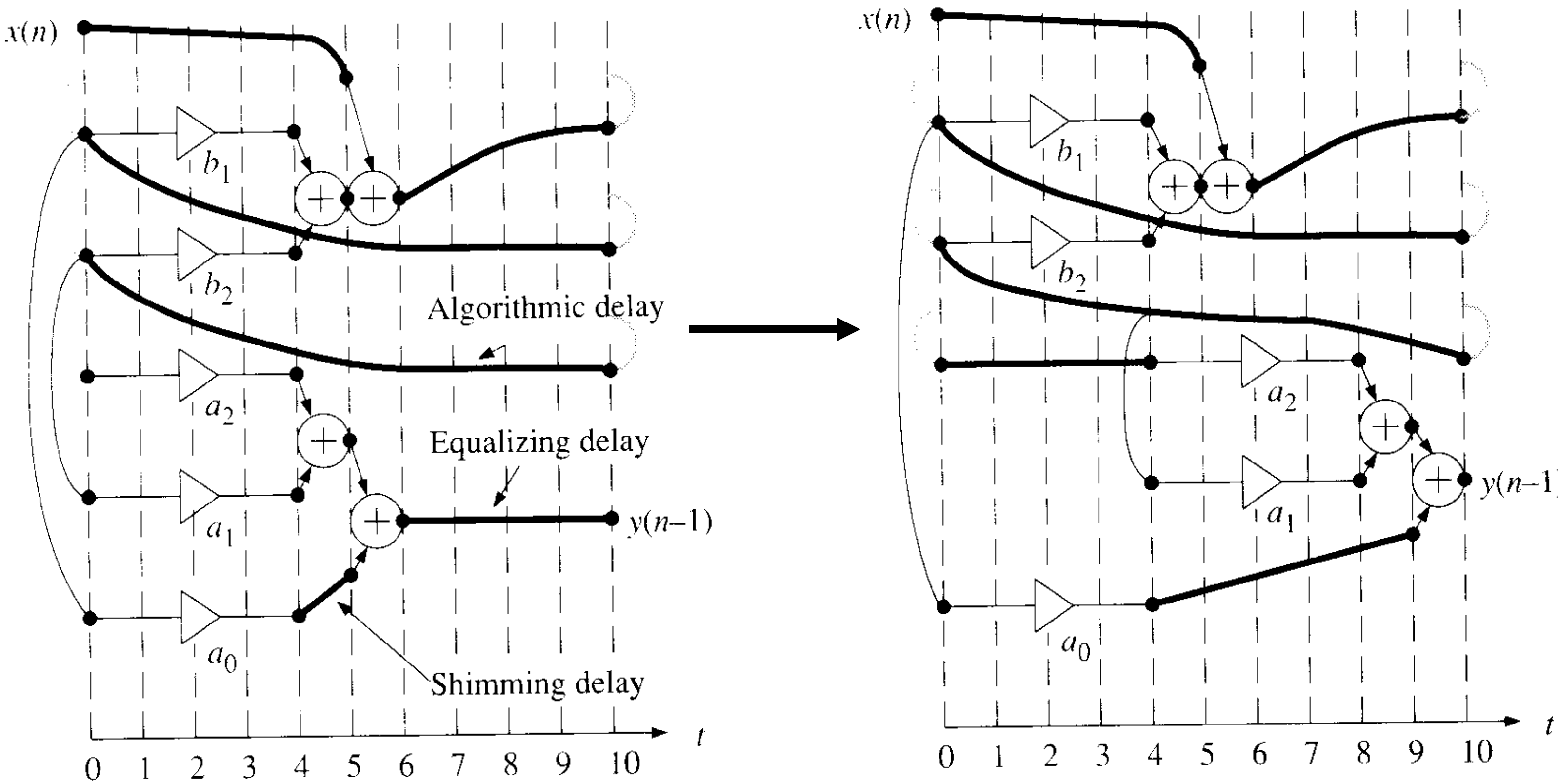
- Computation graph



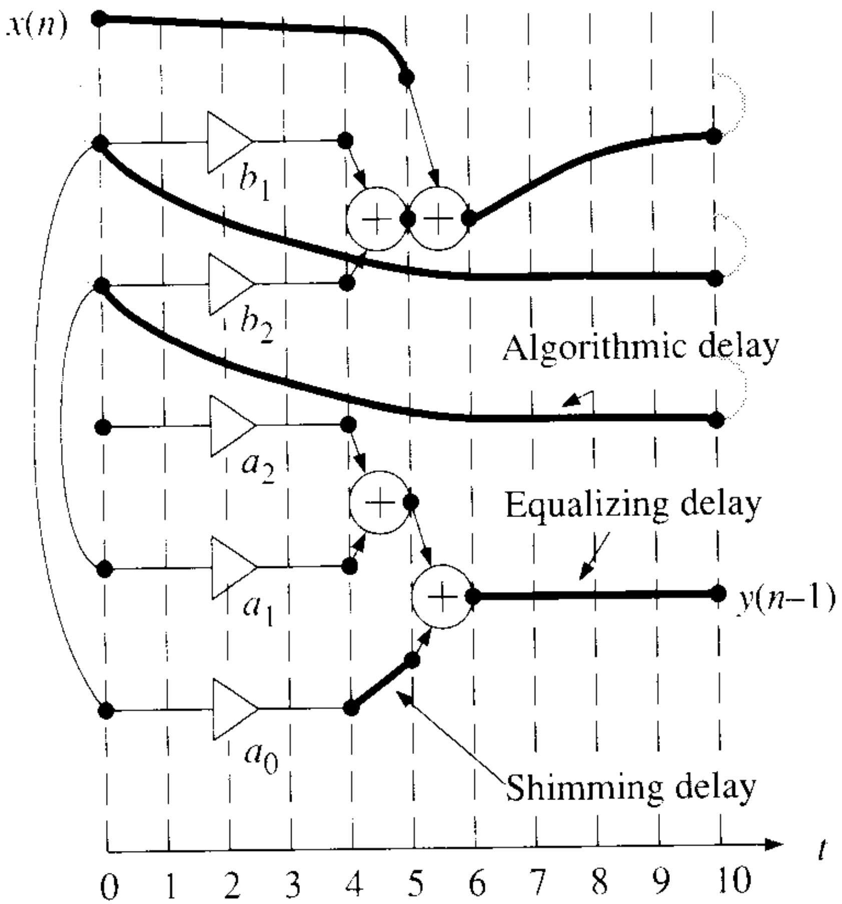


Example (3/5)

- Rescheduling

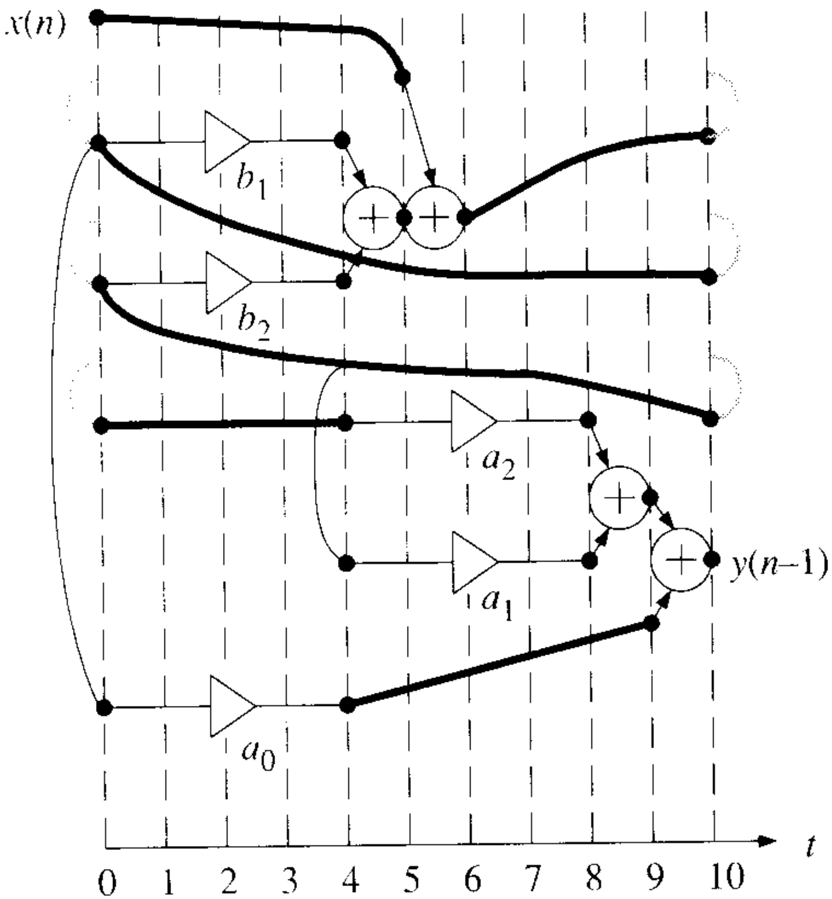


Example (4/5)



- Need 5 multipliers and 2 adders simultaneously
- 34 time units of storage are required

Example (5/5)

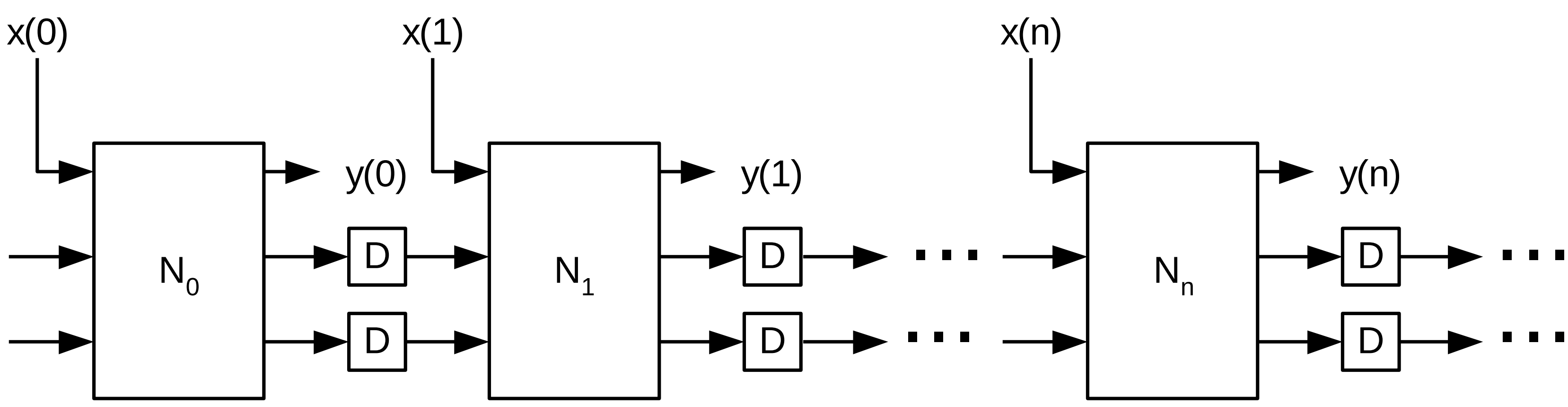


•Need 3 multipliers and 1 adder simultaneously
•38 time units of storage are required

Trade-off between **computational resources** and the **amount of memory** can be made by proper scheduling

Block Formulation (1/2)

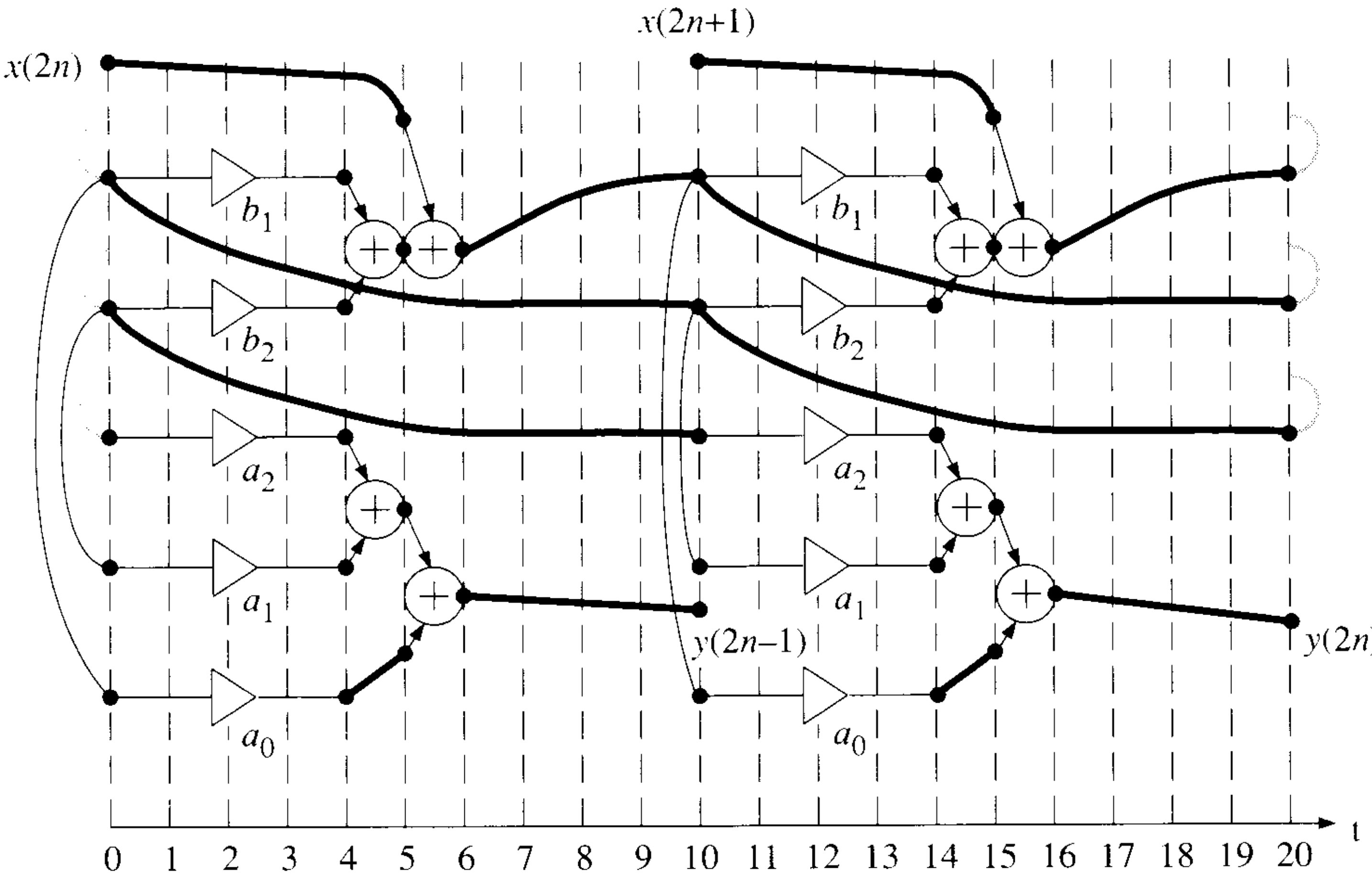
- The repeated execution of the algorithm can be represented by an infinite sequence of computation graphs



Block Formulation (2/2)

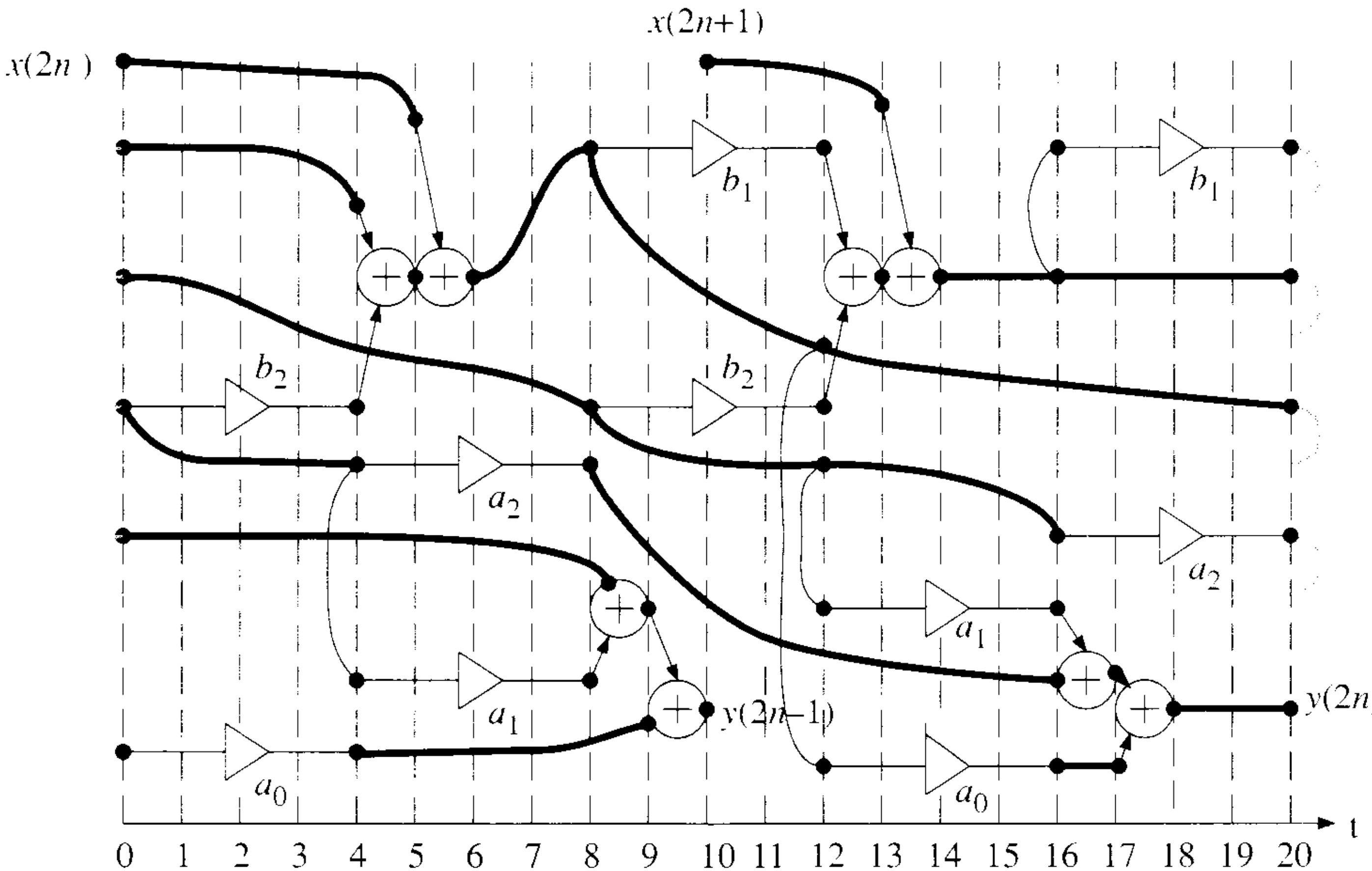
- Allows the operations to be freely scheduled across the sample interval boundaries
- Inefficiency in resource utilization due to the artificial requirement of uniform scheduling boundaries can be reduced
- The price is longer schedules that require longer control sequence

Example (1/2)



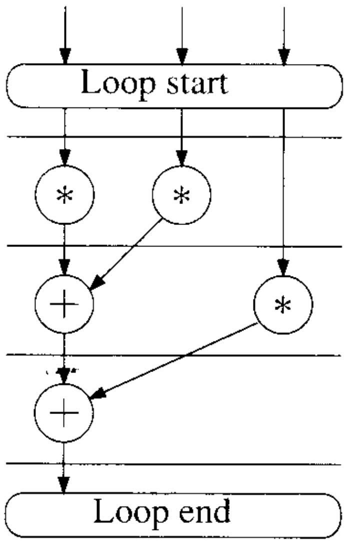
- Need 5 multipliers and 2 adders concurrently
- Need 34 delay elements per sample

Example (2/2)

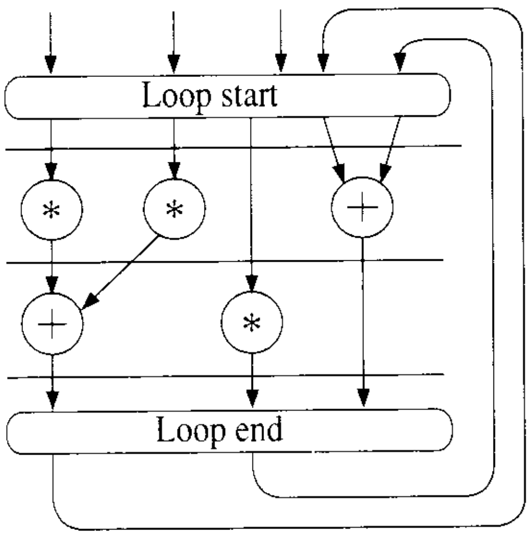


- Need 2 multipliers and 1 adders concurrently
- Need 92/2=46 delay elements per sample

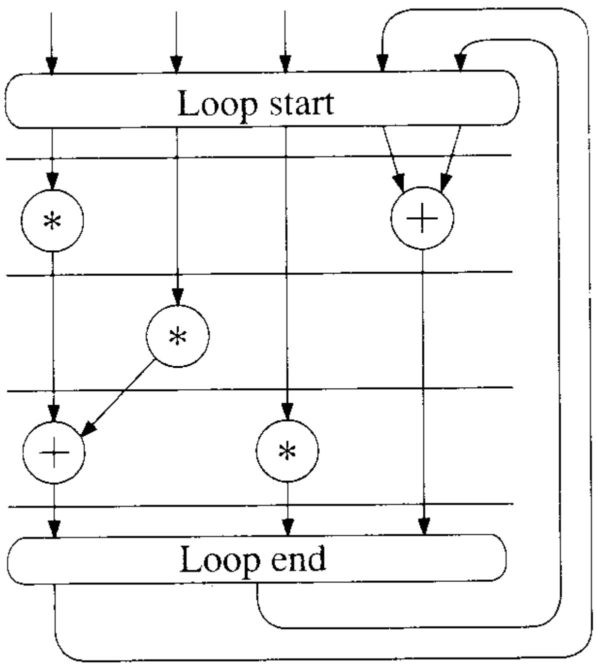
Loop-Folding



Original loop



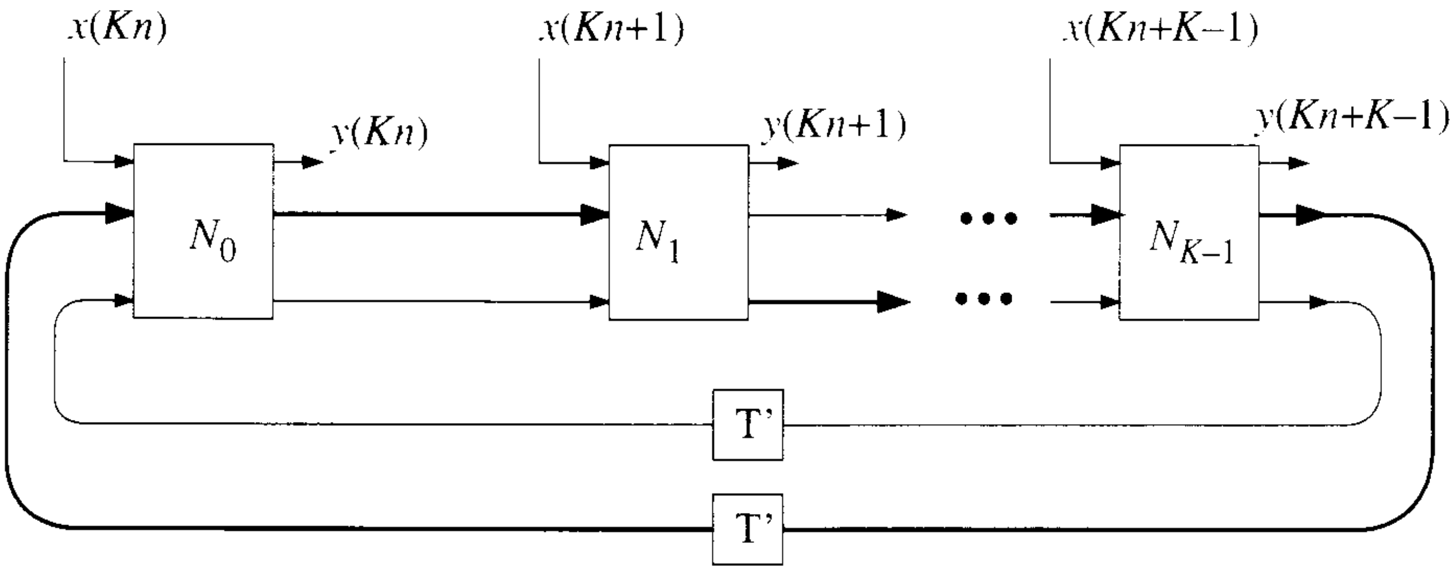
Loop-folding that minimizes the loop time



Loop-folding that minimizes the number of PEs

Periodic Formulation

- Connect K computation graphs as a new computation graph



T'=KT

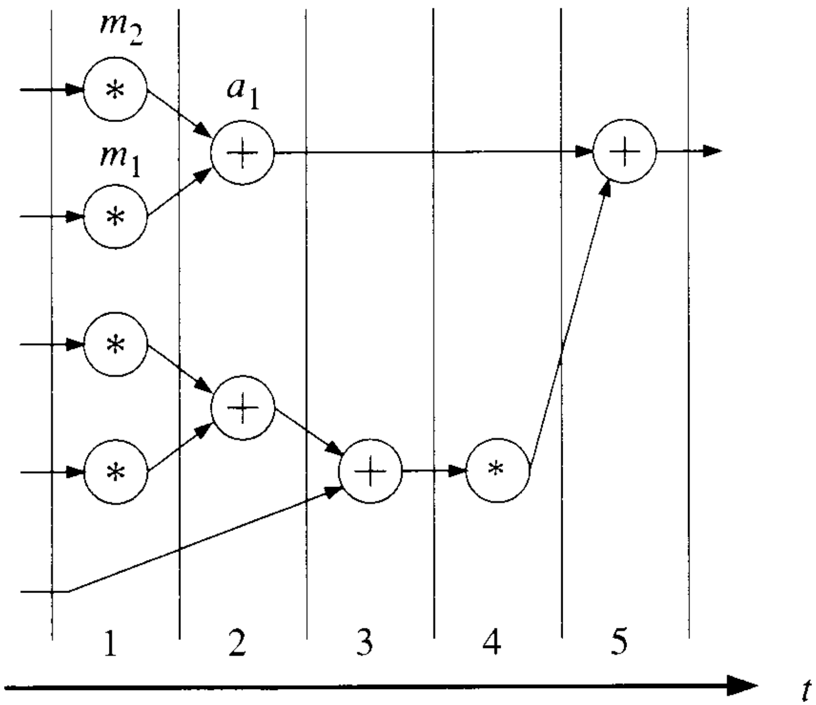
Scheduling Algorithm

- ASAP (as soon as possible) and ALAP (as late as possible) scheduling
- Earliest deadline and slack time scheduling
- Linear programming methods
- Critical path list scheduling
- Force-directed scheduling
- Cyclo-static scheduling
- Simulated annealing
- Genetic algorithms

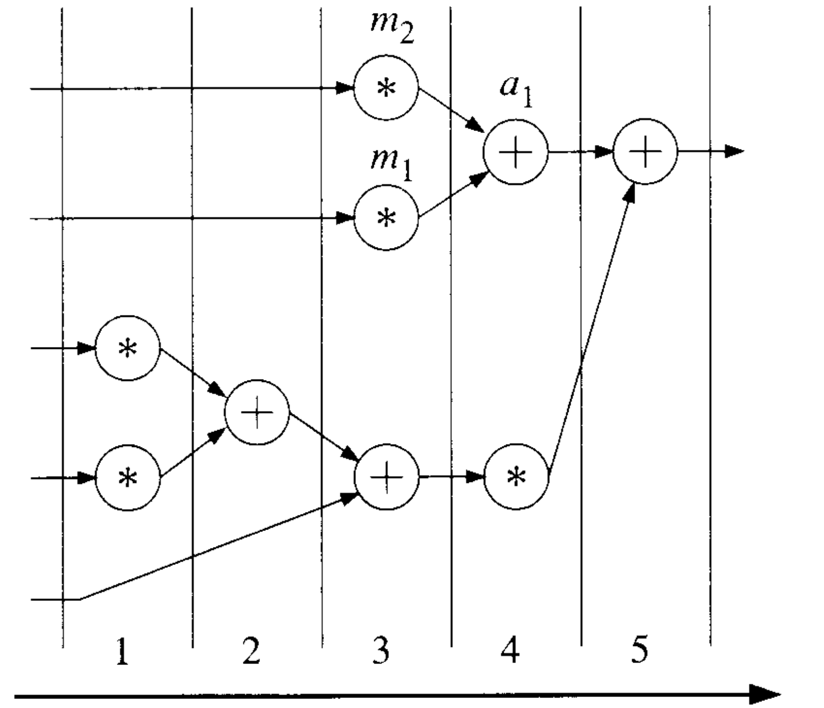
ASAP and ALAP Scheduling (1/2)

- ASAP
  - A computation can be performed as soon as all of its inputs are available
  - The aim is to obtain the shortest possible execution time without considering resource requirements
- ALAP
  - Similar to ASAP, the operations are scheduled as late as possible
- Can be used to determine the time range in which the operations can be scheduled
  - Life-span

ASAP and ALAP Scheduling (2/2)



ASAP scheduling



ALAP scheduling

Life-span: m_1:[1,3], m_2:[1,3], a_1:[2,4]

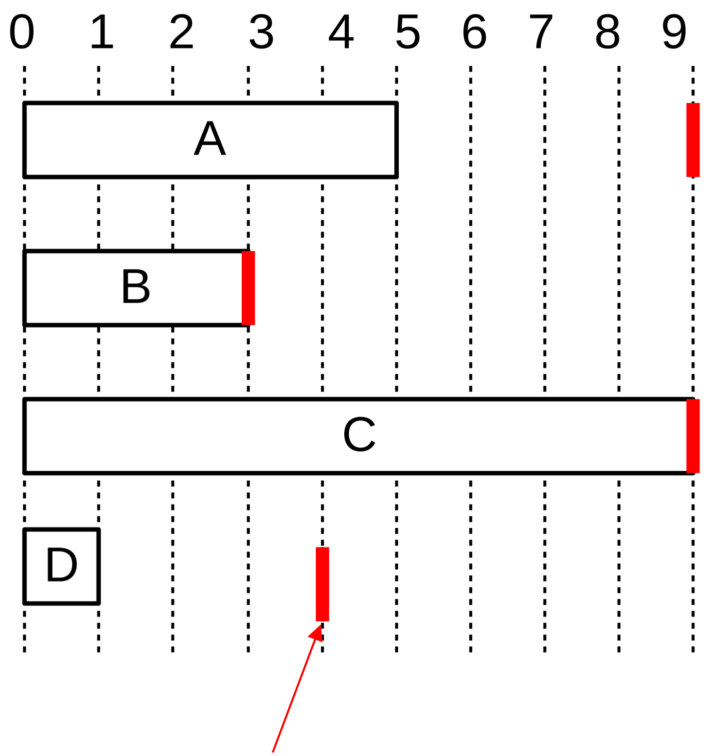
Earliest Deadline and Slack Time Scheduling (1/3)

- Can be used to schedule periodically or randomly occurring processes
- Earliest deadline scheduling
  - In each time step, the processes whose deadline is closest will be done
  - Proven to be execution-time-optimal for single processor

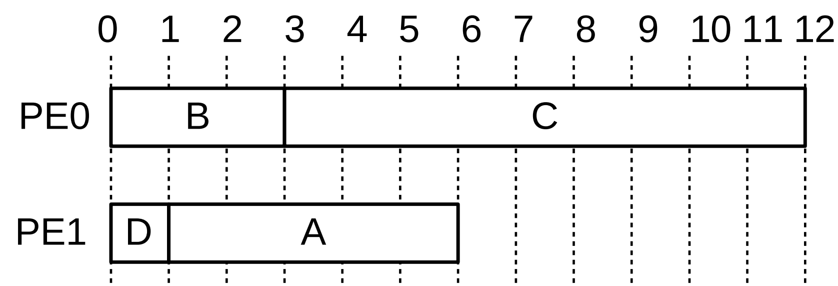
Earliest Deadline and Slack Time Scheduling (2/3)

- Slack time algorithm
  - Schedule the process whose slack time is least
  - Slack time: the time from present to the deadline minus the remaining processing time of a process
  - Better than earliest deadline scheduling when more than one processor is used

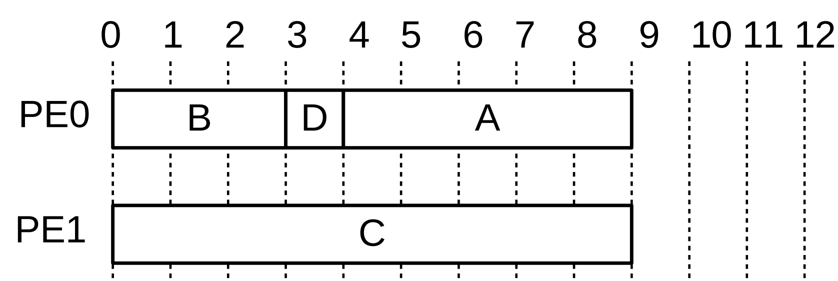
Earliest Deadline and Slack Time Scheduling (3/3)



Deadline



Earliest deadline scheduling



Slack time scheduling

Linear Programming

- Find the optimal value of a linear cost function while satisfying a large set of constraints

Critical Path List Scheduling

- One of the list scheduling method
- Form an ordered list of operations to be performed according to critical paths
- The operations are picked one by one from this list and assigned to a free resource (PE)

Force-Directed Scheduling (1/6)

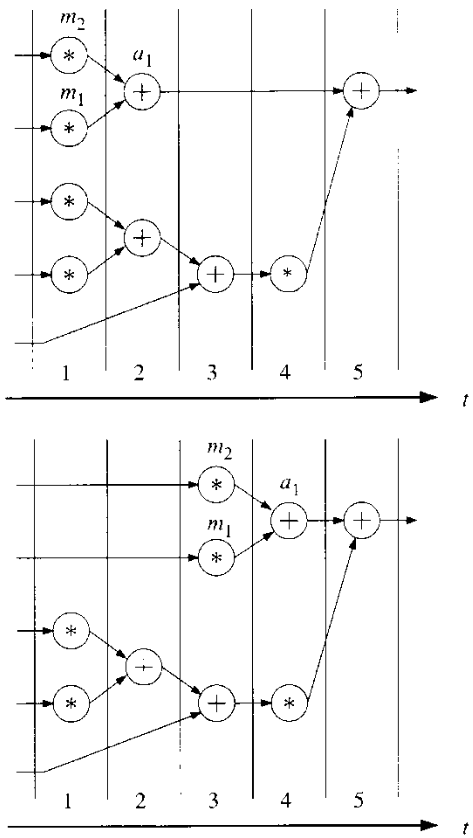
- Target: to distribute the computation on time axis
- While (Unscheduled nodes exist)

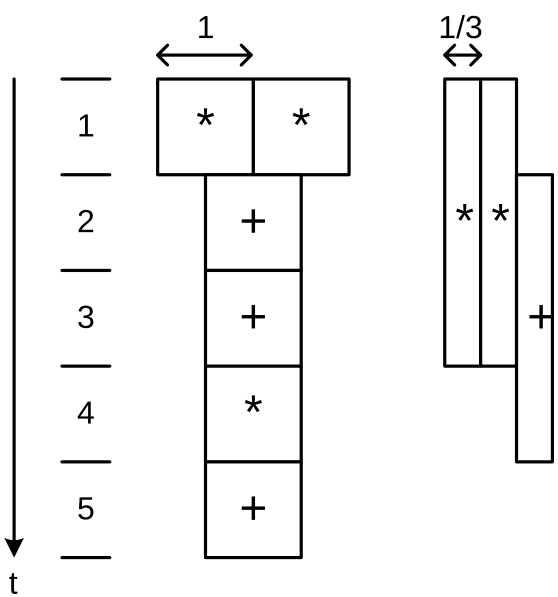
{

- Compute the time frames (life-span) for each node
- Build the distribution graph
- Compute the total force = (self force) + (indirect force)
- Schedule the node into the time step that minimizes the total force

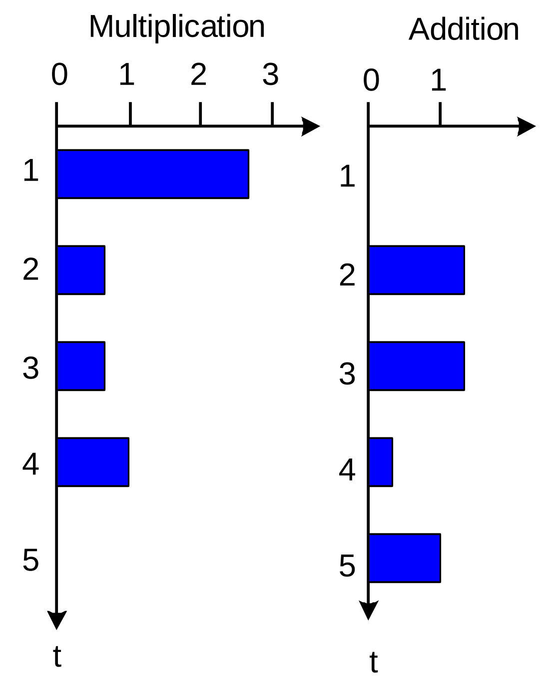
}

Force-Directed Scheduling (2/6)





DG



Execution time frame and associated probabilities

Force-Directed Scheduling (3/6)

- Compute total force
  - Self-force

    If life span of a node $n_j = \lfloor S_j, L_j \rfloor$

    $$\text{Self Force}(j) = \sum_{i=S_j}^{L_j} \left[DG(i) \times x(i) \right]$$

    $DG(i)$: distribution value at time step i

    $x(i)$: the change in the probability associated with time step i
  - Indirect force
    - For the influenced nodes other than j, use the same equation
  - Total force = (self force) + (indirect force)

Force-Directed Scheduling (4/6)

	$\mathbf{m_1}$			$\mathbf{m_2}$			$\mathbf{a_1}$		
	Self	Indirect	Total	Self	Indirect	Total	Self	Indirect	Total
1	$\frac{4}{3}$	0	$\frac{4}{3}$	$\frac{4}{3}$	0	$\frac{4}{3}$	—	—	—
2	$-\frac{2}{3}$	$-\frac{1}{6}$	$-\frac{5}{6}$	$-\frac{2}{3}$	$-\frac{1}{6}$	$-\frac{5}{6}$	$\frac{1}{3}$	$\frac{8}{3}$	3
3	$-\frac{2}{3}$	$-\frac{2}{3}$	$-\frac{4}{3}$	$-\frac{2}{3}$	$-\frac{2}{3}$	$-\frac{4}{3}$	$\frac{4}{3}$	$\frac{2}{3}$	1
4	—	—	—	—	—	—	$-\frac{4}{3}$	0	$-\frac{2}{3}$

$$2\frac{2}{3}\times\left(1-\frac{1}{3}\right)+\frac{2}{3}\times\left(0-\frac{1}{3}\right)+\frac{2}{3}\times\left(0-\frac{1}{3}\right)=\frac{4}{3}$$

$$2\frac{2}{3}\times\left(0-\frac{1}{3}\right)+\frac{2}{3}\times\left(1-\frac{1}{3}\right)+\frac{2}{3}\times\left(0-\frac{1}{3}\right)=-\frac{2}{3}$$

$$2\frac{2}{3}\times\left(0-\frac{1}{3}\right)+\frac{2}{3}\times\left(0-\frac{1}{3}\right)+\frac{2}{3}\times\left(1-\frac{1}{3}\right)=-\frac{2}{3}$$

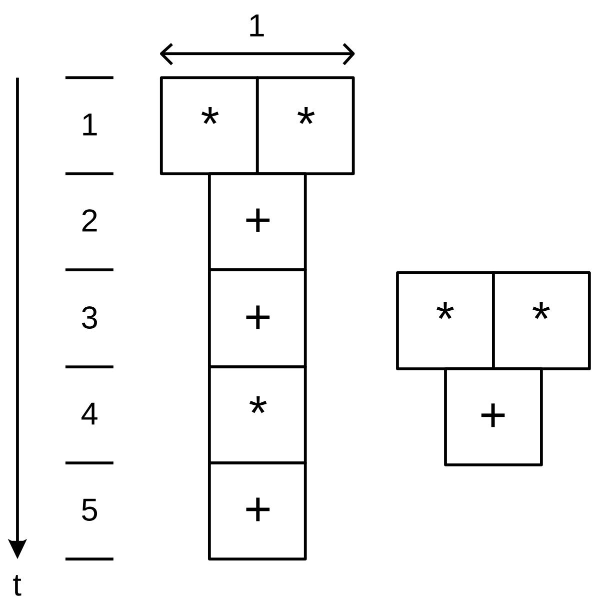
$$\frac{4}{3}\times\left(0-\frac{1}{3}\right)+\frac{4}{3}\times\left(\frac{1}{2}-\frac{1}{3}\right)+\frac{1}{3}\times\left(\frac{1}{2}-\frac{1}{3}\right)=-\frac{1}{6}$$

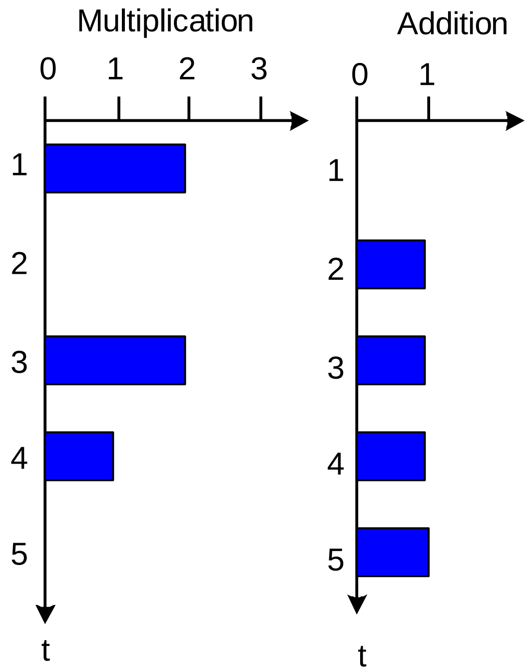
$$\frac{4}{3}\times\left(0-\frac{1}{3}\right)+\frac{4}{3}\times\left(0-\frac{1}{3}\right)+\frac{1}{3}\times\left(1-\frac{1}{3}\right)=-\frac{2}{3}$$

Force-Directed Scheduling (5/6)

	m_1			m_2			a_1		
	Self	**Indirect**	**Total**	**Self**	**Indirect**	**Total**	**Self**	**Indirect**	**Total**
1	$\frac{4}{3}$	0	$\frac{4}{3}$	$\frac{4}{3}$	0	$\frac{4}{3}$	—	—	—
2	$-\frac{2}{3}$	$-\frac{1}{6}$	$-\frac{5}{6}$	$-\frac{2}{3}$	$-\frac{1}{6}$	$-\frac{5}{6}$	$\frac{1}{3}$	$\frac{8}{3}$	3
3	$-\frac{2}{3}$	$-\frac{2}{3}$	$-\frac{4}{3}$	$-\frac{2}{3}$	$-\frac{2}{3}$	$-\frac{4}{3}$	$\frac{1}{3}$	$\frac{2}{3}$	1
4	—	—	—	—	—	—	$-\frac{2}{3}$	0	$-\frac{2}{3}$

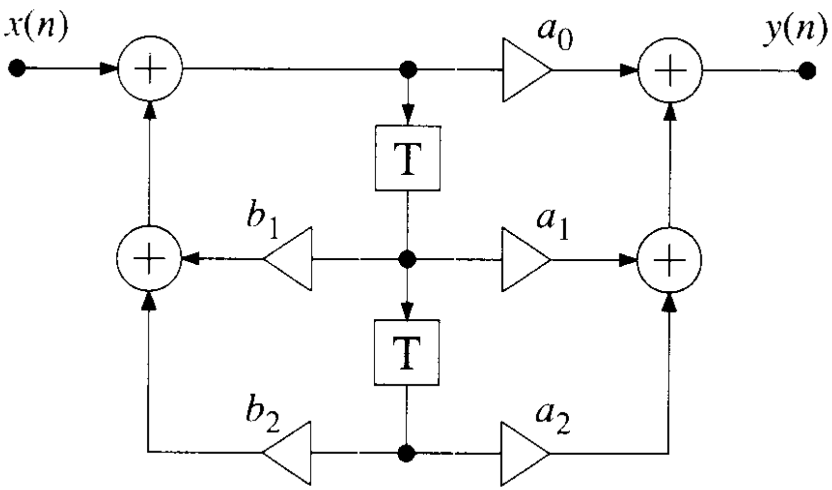
Force-Directed Scheduling (6/6)





Cyclo-Static Scheduling (1/3)

- Considering a pattern in the processor-time space, **P**x**T** find the lattice vector **L**



$$\text{Iteration bound}: T_\infty = \max\left\{\frac{4}{1}, \frac{4}{2}\right\} = 4$$

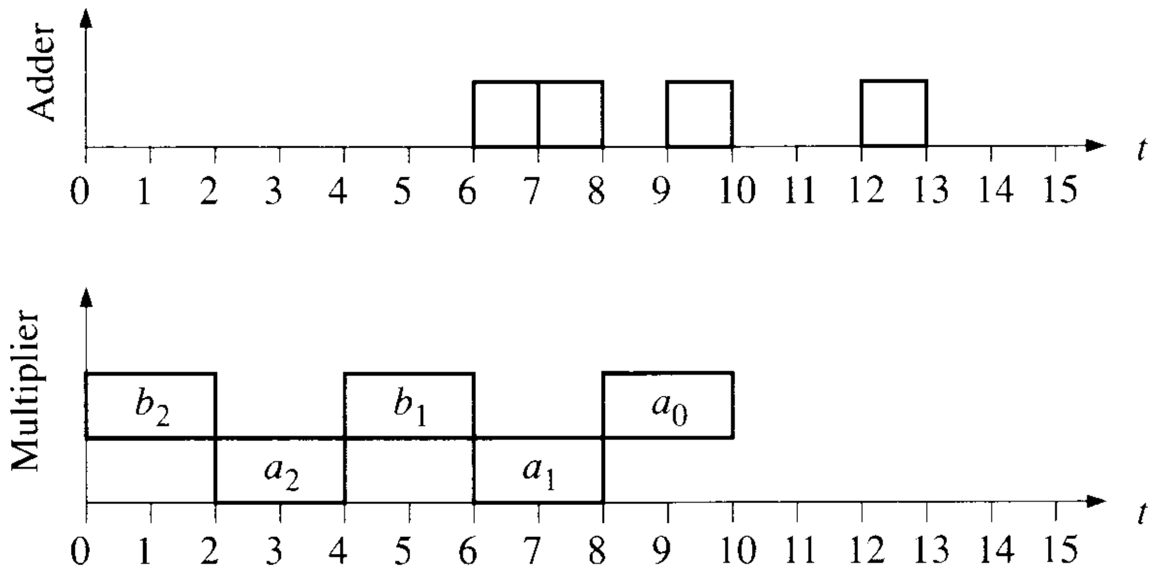
$$\text{Processor bound}: \text{adder}: \left\lceil \frac{1\times 4}{4} \right\rceil = 1,$$

$$\text{multiplier}: \left\lceil \frac{2\times 5}{4} \right\rceil = 3$$

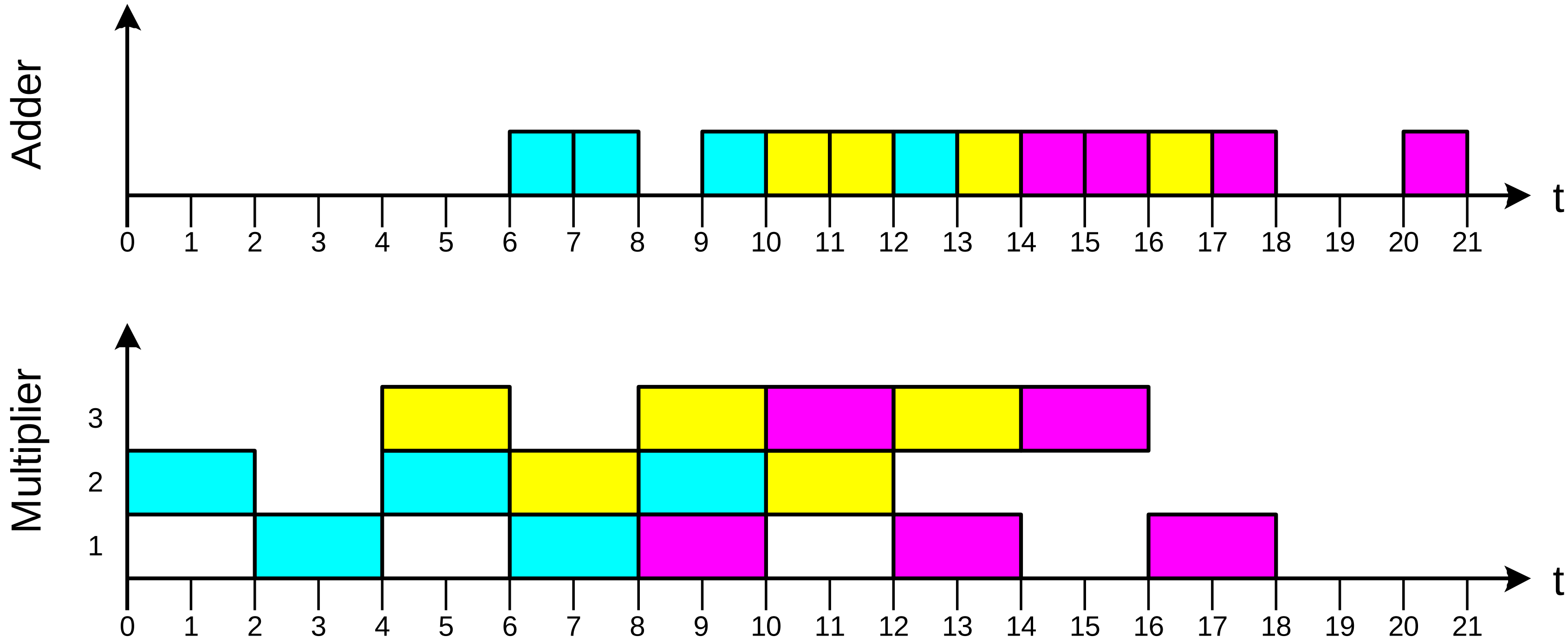
Assume T_A=1, T_M=2

Cyclo-Static Scheduling (2/3)

- L=(displace for adder, displacement for multiplier, displacement for time)=(0,1,4)



Cyclo-Static Scheduling (3/3)



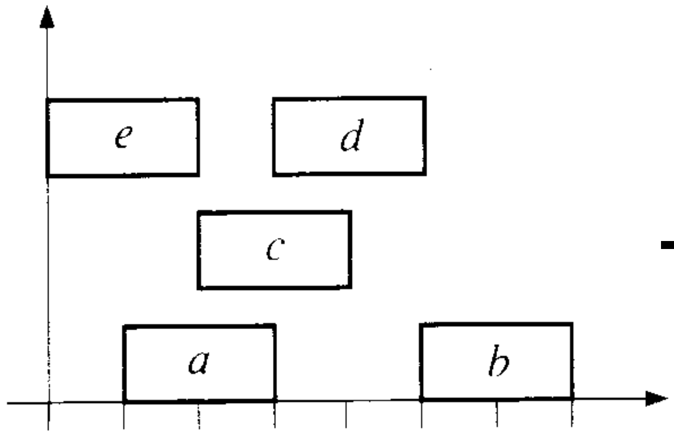
Resource Allocation and Assignment

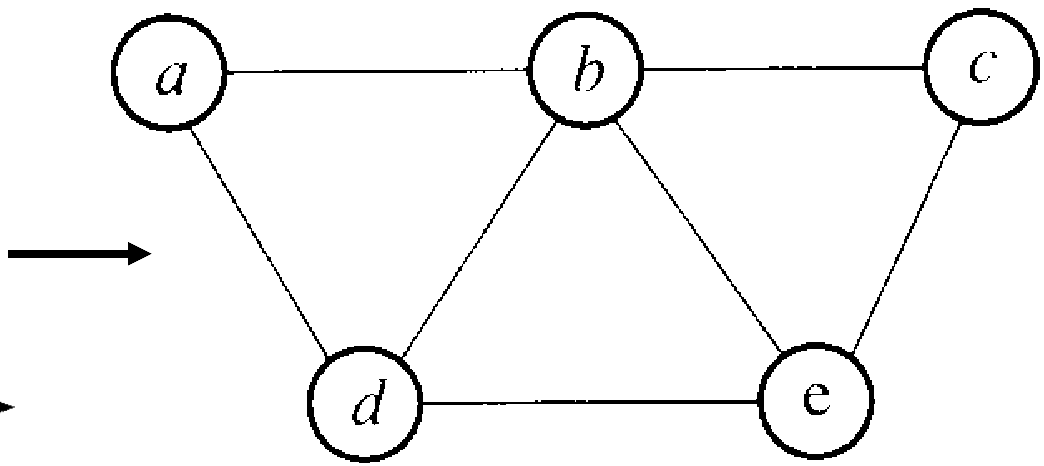
- Usually, the largest number of concurrent processes determines the number of resources required; however, more resource are often required
- Clique partitioning
- Left-edge algorithm

Clique Partition (1/3)

- Draw connectivity graph (or compatibility graph)
  - Connecting two vertices with an edge if the lifetimes of the corresponding processes do not overlap
- Partition the connectivity graph into a number of cliques, which are fully connected sub-graphs that have an edge between all pairs of vertices

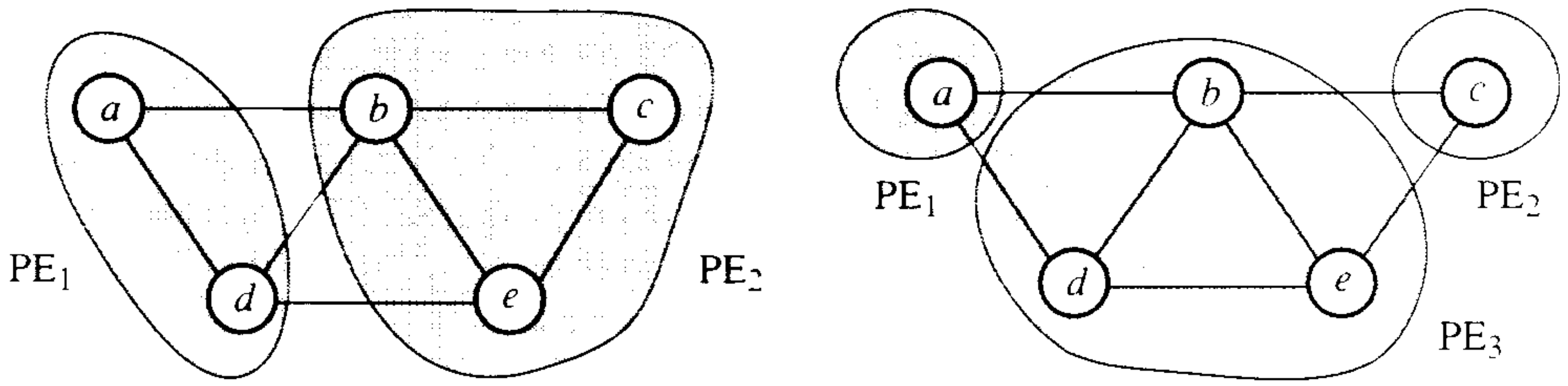
Clique Partition (2/3)





Clique Partition (3/3)

- Two possible clique partitions



- Exclusion graphs are also often used

Left-Edge Algorithm (1/3)

1. Sort the processes into a list according to their starting times. Begin the list with the process having the longest lifetime if the schedule is cyclic. Processes with the same starting time are sorted with the longest lifetime first. Let $i = 1$.

2. Assign the first process in the list to a free resource i, determine its finishing time, and remove it from the list.

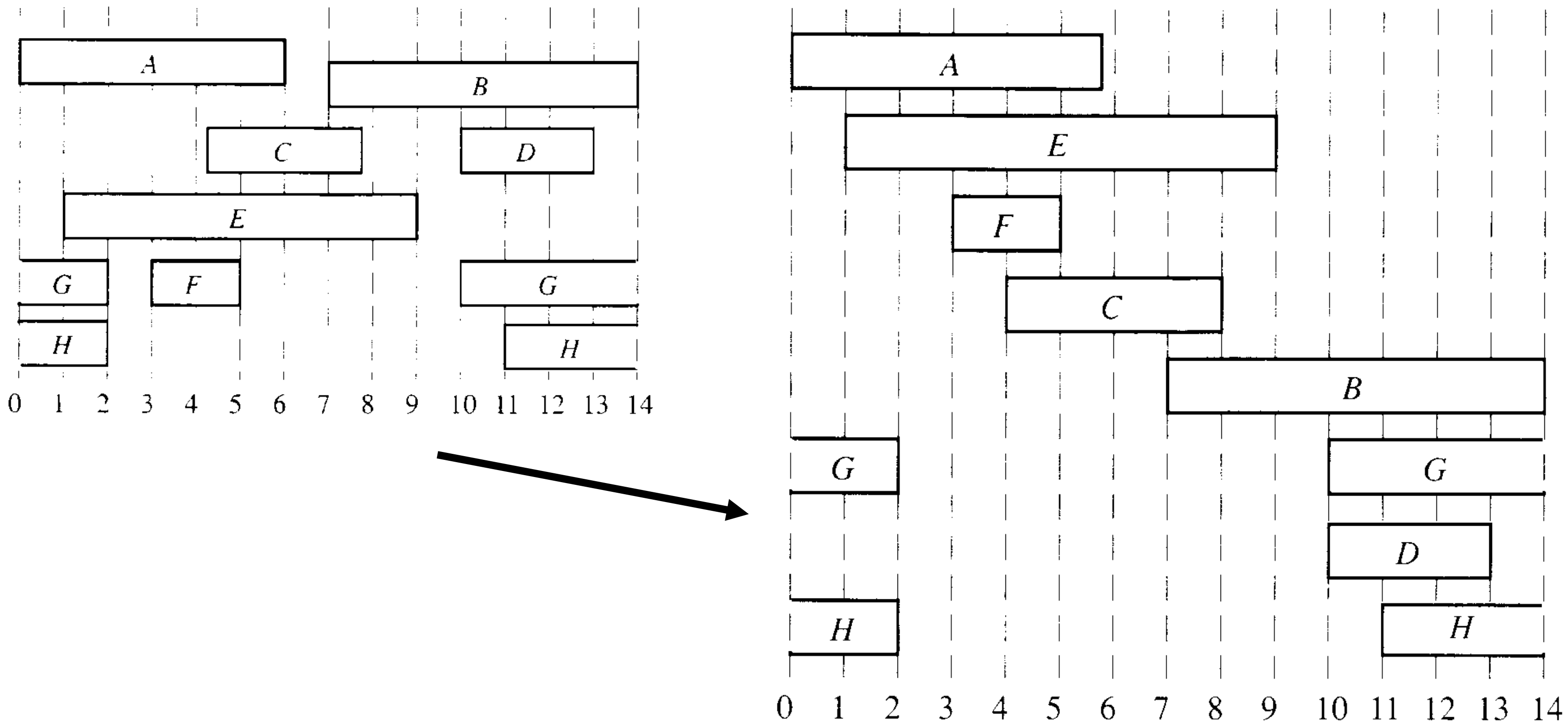
3. Search the list for the first process that has

- **(a)** a starting time equal to, or later than, the finishing time for the previously assigned process; and
- **(b)** a finishing time that is no later than the starting time for the first process selected in step 2.

4. **If** the search in step 3 fails **then**

```
    if there are processes left in the list then
        let i ← i + 1 and repeat from step 2
    else
        Stop
    end if;
else
        assign the process to resource i, determine its finishing time,
        remove it from the list, and repeat from step 3.
end if;
```

Left-Edge Algorithm (2/3)



Left-Edge Algorithm (3/3)