# RESOURCE SHARING

© Giovanni De Micheli

Stanford University

# Outline

Resource sharing is the assignment of a resource to more than one operation

- Resource-dominated circuits.
  - Flat and hierarchical graphs.
  - Functional and memory resources.
- **Extensions**.
  - Non resource-dominated circuits.
  - Concurrent scheduling and binding.
  - Module selection.

# Allocation and binding

- ***Allocation:***
  - Number of resources is available. Which resource for which operation.
- ***Binding:***
  - Binding is a relation between operations and resources.
- ***Sharing:***
  - Many-to-one relation. Several operations share one resource
- ***Optimum binding/sharing:***
  - Minimize the resource usage.

# Binding

- **Limiting cases of binding:**
  - *Dedicated resources:*
    - One resource per operation.
    - No sharing.
  - *One multi-task resource:*
    - ALU.
  - One resource per type.

Resource binding can be applied to sequencing graphs that are scheduled or unscheduled.

# Examples of types of sharing.

- The simplest case is when operations can be matched to resources with the same type
- **Generalization** - operations with different types can be implemented (covered) by one resource of appropriate type.
  - Addition, subtraction and comparison by ALU
- A further generalization is the case when any operation can be implemented by more than one resource type, possibly with different area and performance.
  - **Example:** addition operation can be implemented in different kind of adders.

# Sharing and Binding for Resource-Dominated Circuits

- Two or more operations may be bound to the same resource ***if they are not concurrent*** **and** they can be implemented by resources of the same type.
- When these conditions are met, the operations are said to be **compatible.**

# Optimum sharing problem

- We start from scheduled sequencing graphs.
  - Operation concurrency is well defined.
- We consider ***operation types*** independently.
  - Problem decomposition.
  - Perform analysis for each resource type.

# Compatibility graphs and conflict graphs

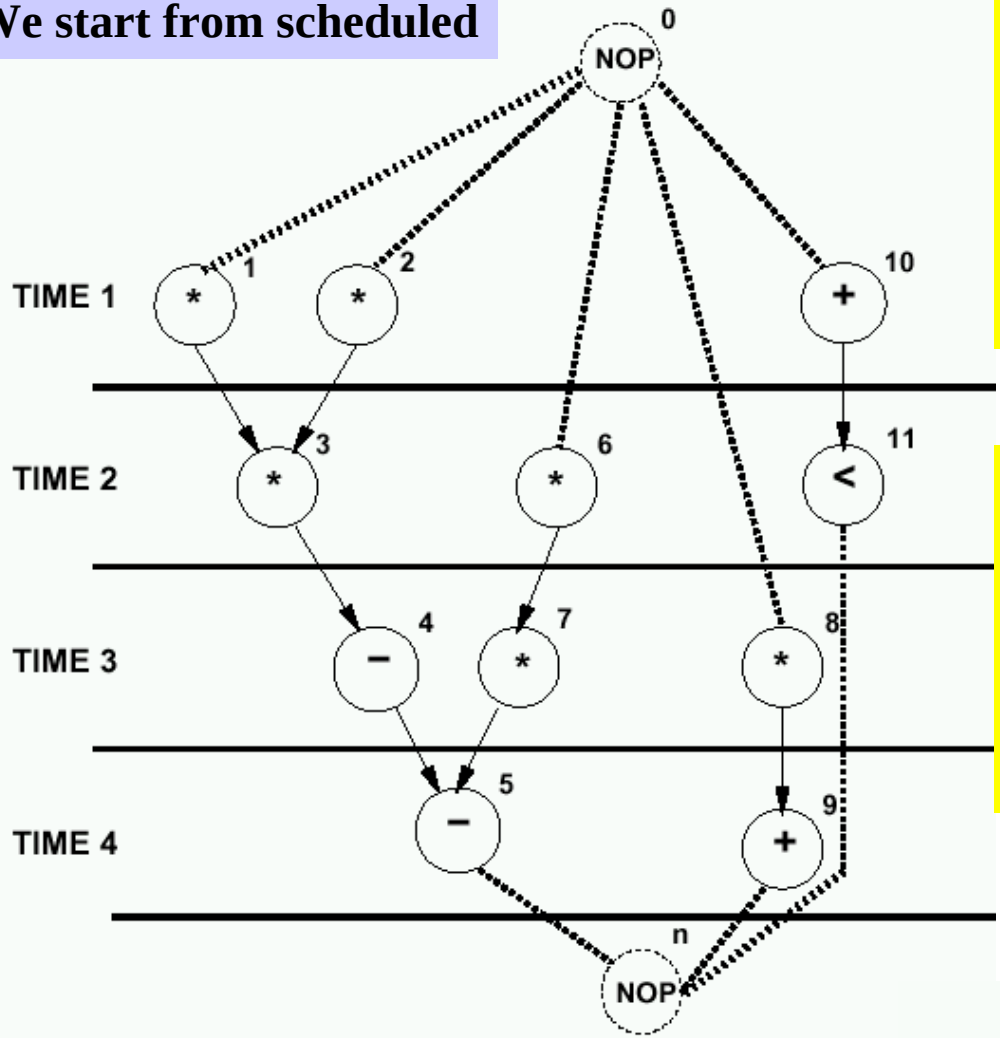
- Operation compatibility:
  - Same type.
  - Non concurrent.
- (Resource) ***Compatibility*** graph:
  - Vertices: operations.
  - Edges: compatibility relation.
- ***Conflict*** graph:
  - Complement of compatibility graph.

These are the same compatibility and incompatibility graphs as we discussed already many times

# Definitions

- **Definition 6.2.1.**
  - The resource compatibility graph is a graph whose vertex set $V=\{v_i, i=1,2,\ldots n_{ops}\}$is in one-to-one correspondence with the operations and whose edge set $E=\{\{v_i, v_j\},i,j=1,2,\ldots n_{ops}$ denotes the compatible operation pairs.
- **Definition 6.2.2.**
  - The resource conflict graph is a graph whose vertex set $V=\{v_i, i=1,2,\ldots n_{ops}\}$is in one-to-one correspondence with the operations and whose edge set $E=\{\{v_i, v_j\},i,j=1,2,\ldots n_{ops}$ denotes the conflict operation pairs

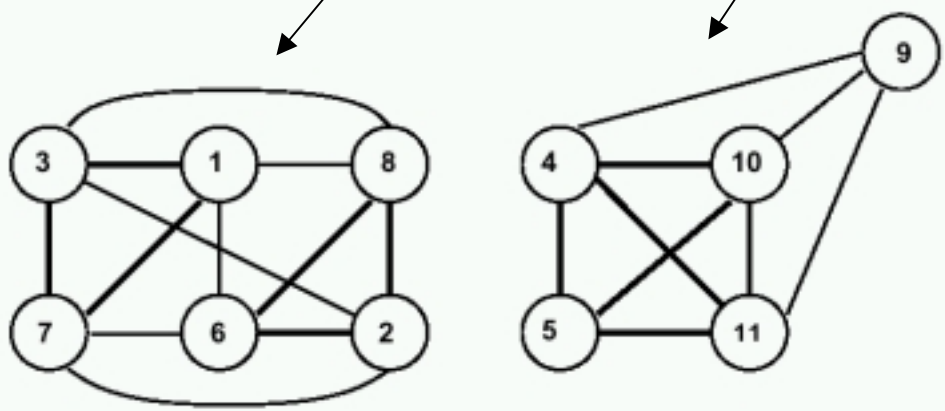
We start from scheduled




# Examples of Compatibility graphs and conflict graphs

**Compatibility graph is a complement of a conflict graph**

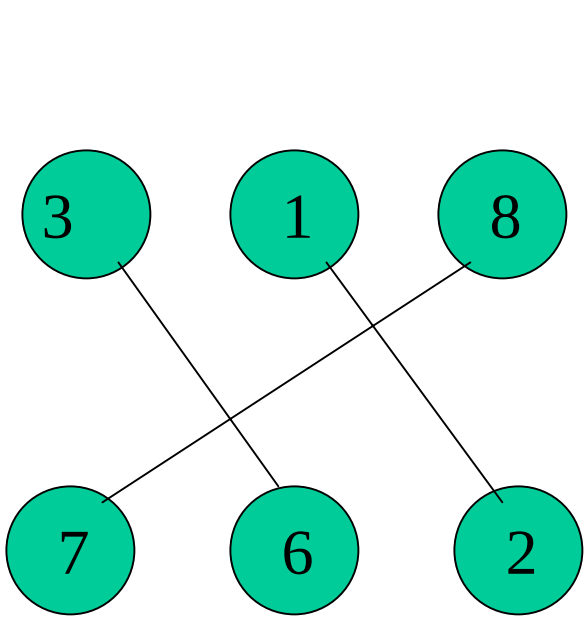
Multiplier ALU

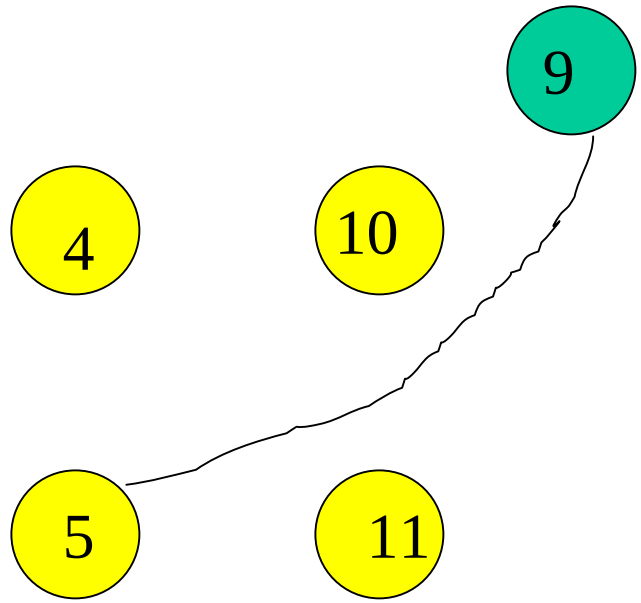



**Compatibility graphs**

**Observe that 1 and 2 are <u>not compatible</u> since they are executed concurrently**

# Conflict graphs for the multiplier and ALU types



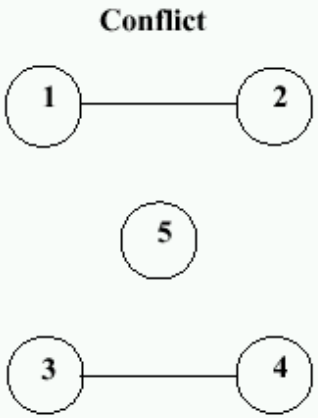



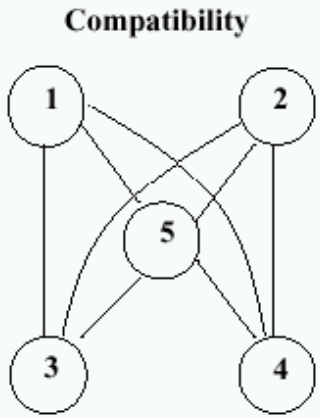
# Algorithmic solution to the optimum binding problem

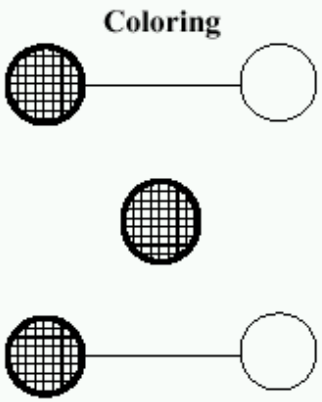
- Compatibility graph.
  - Partition the graph into a minimum number of cliques.
  - Find clique cover number $\kappa(G_+)$.
- (Resource) Conflict graph.
  - Color the vertices by a minimum number of colors.
  - Find *chromatic number* $\gamma(G_-)$
- NP-complete problems - Heuristic algorithms.

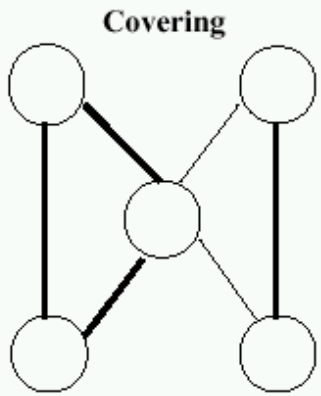
It is obvious that conflict graph is the complement of the compatibility graph

# Examples of using conflict and compatibility graphs for binding




ALU1: 1,3,5

ALU2: 2,4

| | | | | |
|---|---|---|---|---|
| t1 | x=a+b | y=c+d | 1 | 2 |
| t2 | s=x+y | t=x−y | 3 | 4 |
| t3 | z=a+t | | 5 | |

**We start from scheduled graph**

resources

y,t

x, s, z

Since operations with different types are always conflicting, it is convenient to consider the conflict graphs for each type independently.

# Perfect graphs

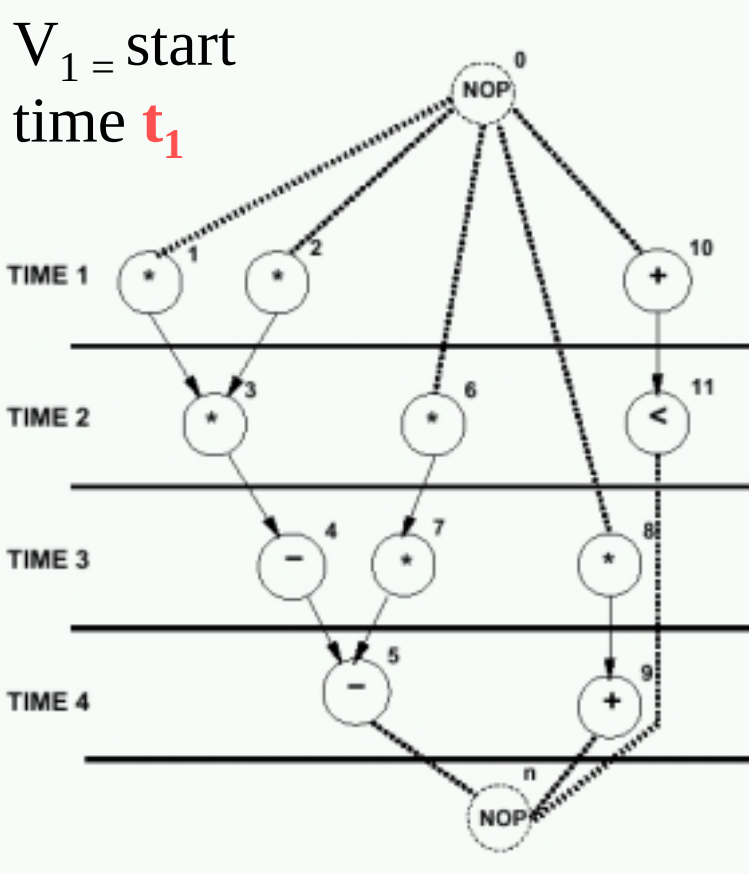
- ***Comparability graph:***
  - Graph G(V, E) has an orientation G(V, F) with the transitive property.
  - $(v_i, v_j) \in F \wedge (v_j, v_k) \in F) \Rightarrow (v_i; v_k) \in F.$
- Interval graph:
  - Vertices correspond to ***intervals.***
  - Edges correspond to ***interval*** intersection.
  - Interval graphs are a subset of ***chordal*** graphs:
    - Every loop with more than three edges **has a chord**.

# What is a Perfect graph?

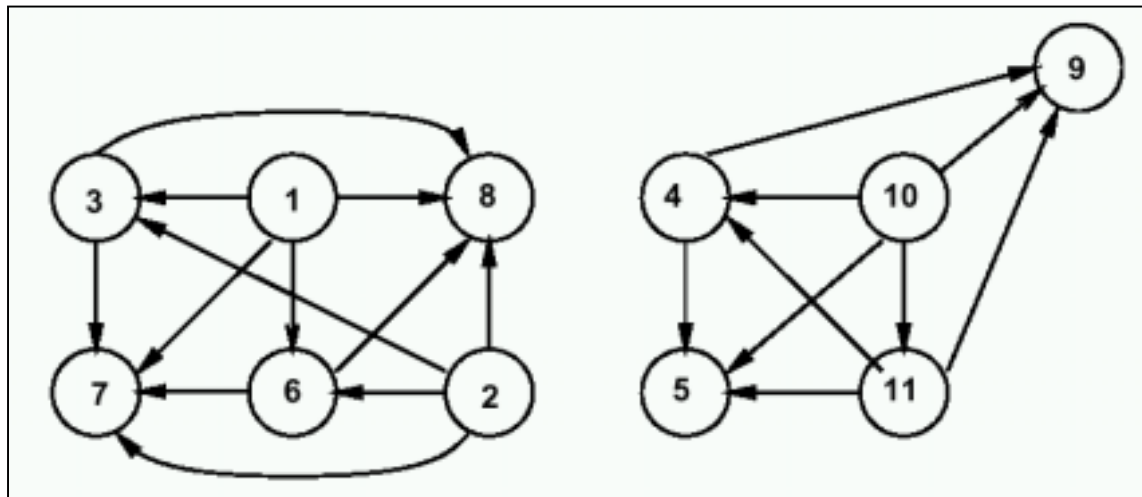
# Data-flow graphs ( at sequencing graphs)

- The compatibility/conflict graphs have special properties.
  - Compatibility:
    - Comparability graph.
  - Conflict:
    - Interval graph.
- Polynomial time solutions:
  - **Golumbic's** algorithm.
  - **Left-edge** algorithm.

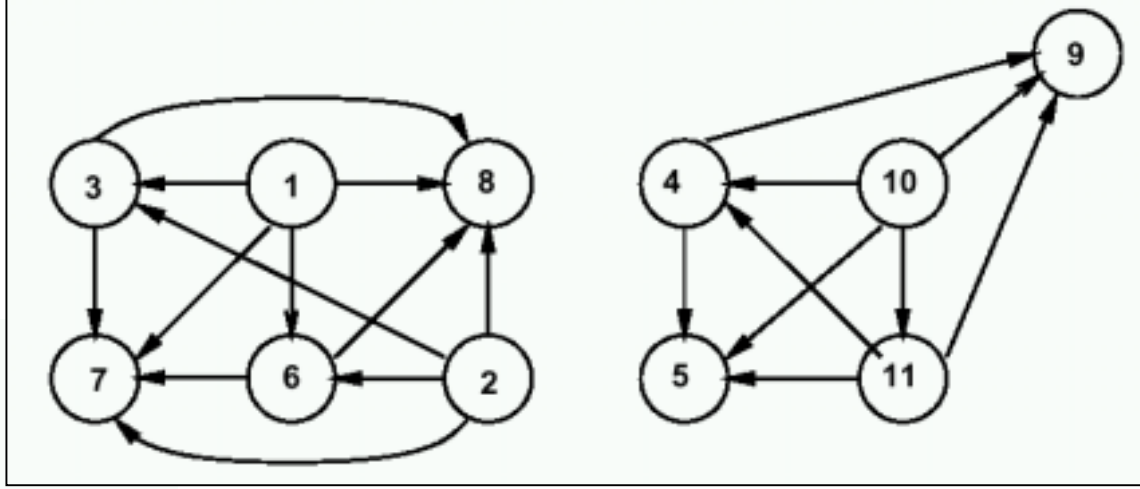
$V_1$ = start time $t_1$



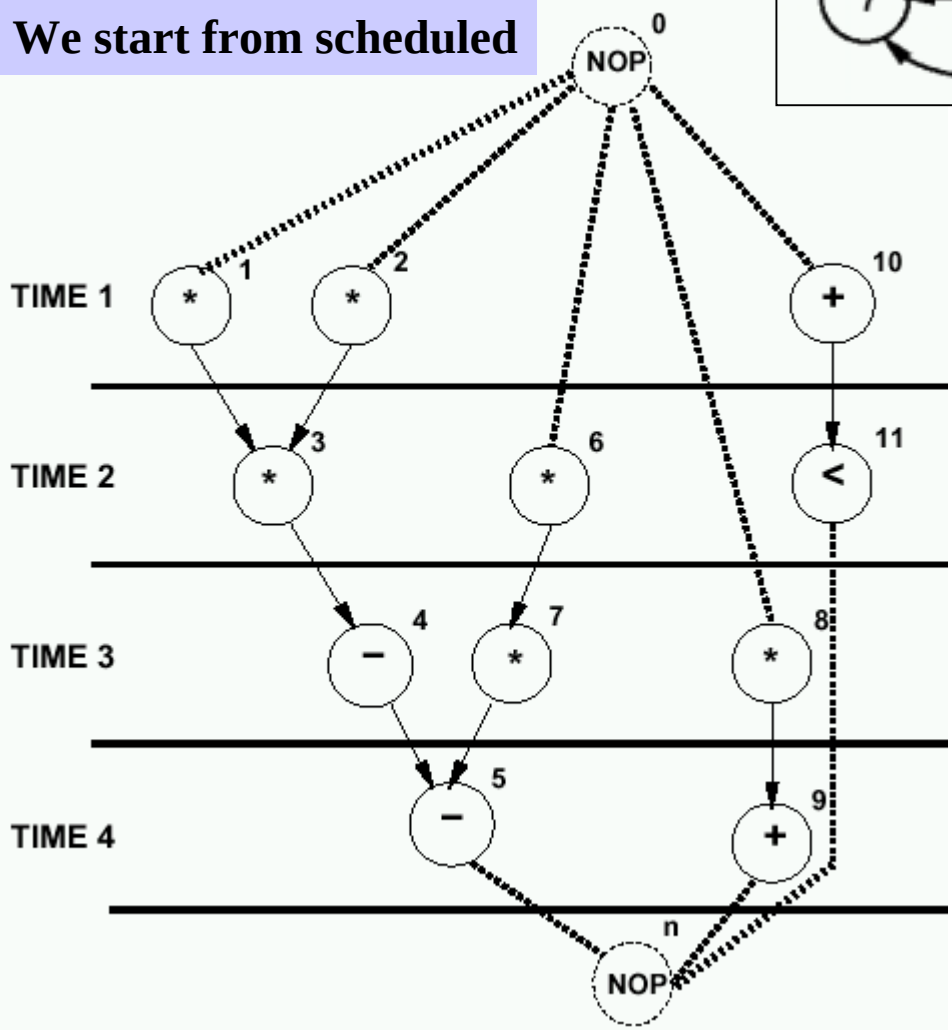

- All operations have unit execution delay
- All operations whose type is a multiplier and whose start time is larger than or equal 2 are compatible with $v_1$
- Acomparability graph can be constructed by traversing the sequencing graph in $O(|V|^2)$ time
- This is a comparability graph because it has a **transitive orientation property**.
- Relation of time ordering is transitive.



# Example of compatibility graph being a comparability graph




We start from scheduled




This graph shows both scheduling order and compatibility

This graph is a comparability graph

1,3,7 = Multiplier

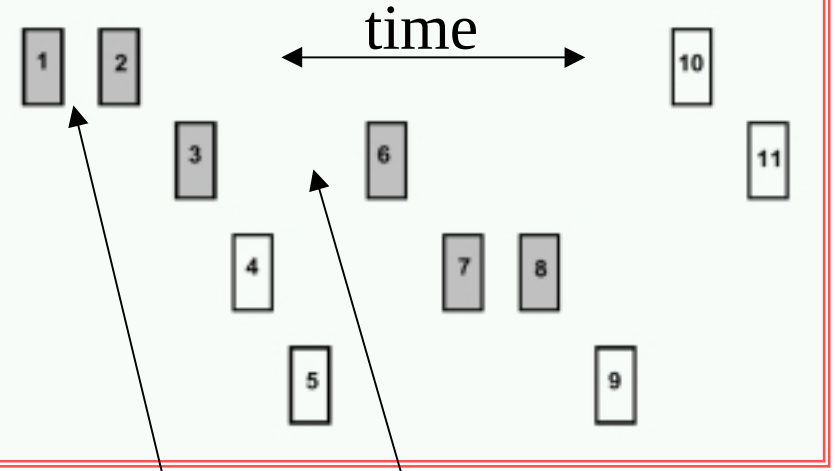
6,8 = Multiplier

10,11,4,9 = ALU

5 = ALU

Solution is not unique, 4,10,11,5

Solution
Latency = 4
Multipliers = 2
ALU = 2




1 and 2 have conflict

3 and 6 have conflict

•Let us consider the conflict graph for each resource type.

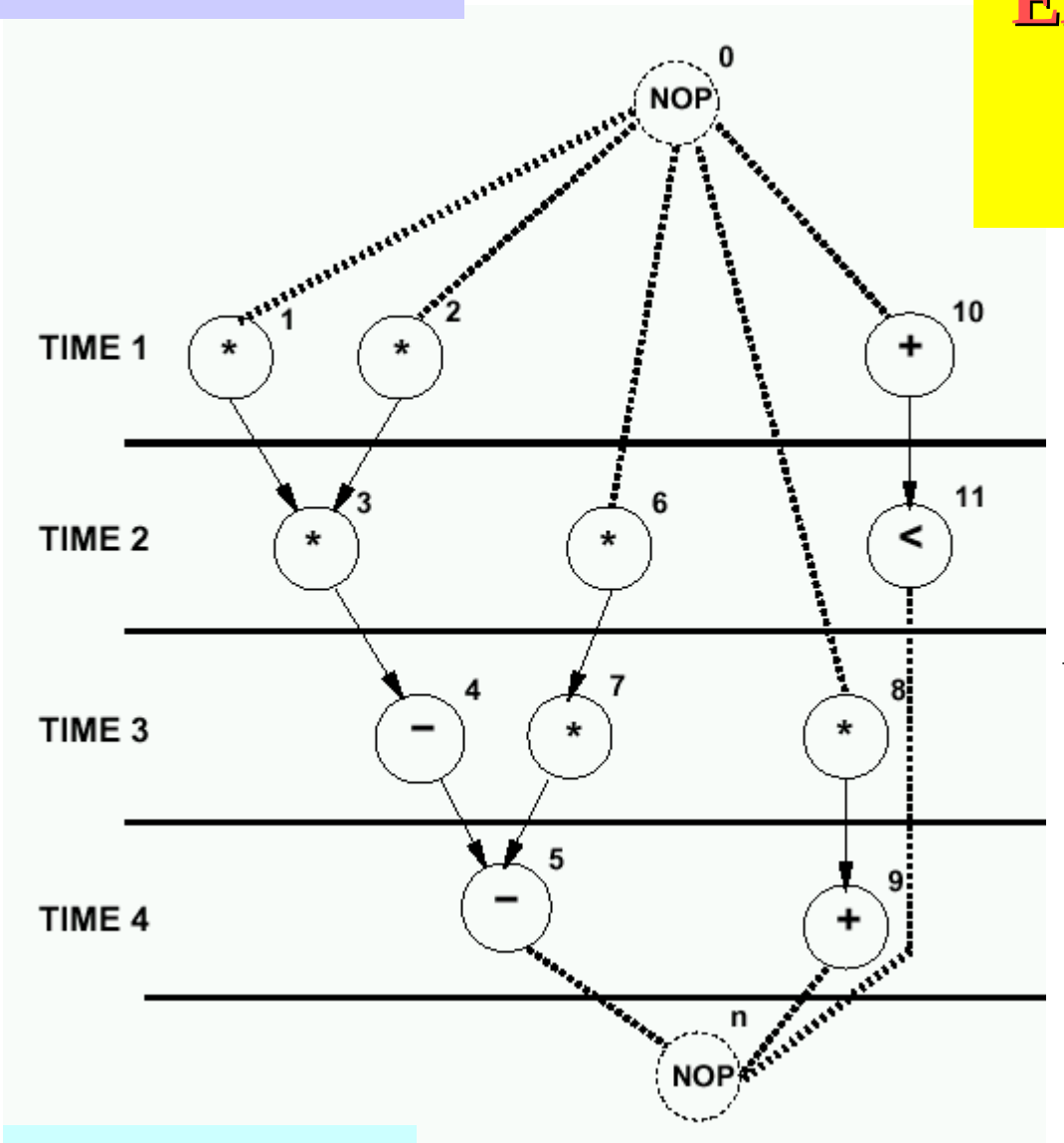
•The edges of the conflict graph denote intersections among intervals; hence they are interval graphs.

•The search for minimum coloring can be achieved in polynomial time. Left-edge algorithm.

•Usually, resource sharing and binding is achieved by considering the conflict graphs for each type, because resources are assumed to have a single type.Thus the overall conflict graph is of limited interest.

We start from scheduled

# Example of using conflict graph which is an interval graph



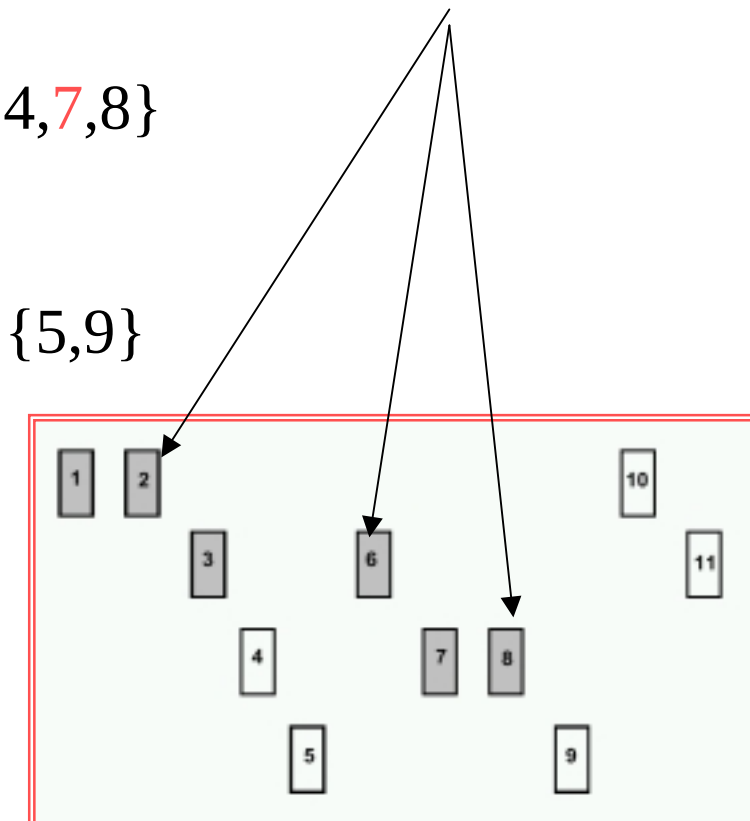

{1,2,10}

{3,6,11}

{4,7,8}

{5,9}

As we see, 1,3,7 can have the same color




**Solution**
**Latency = 4**
**Multipliers = 2**
**ALU = 2**

**{1,3,7}=multiplier**

**{2,6,8}=multiplier**

**{4,5,10,11}=ALU**

**{9}=ALU**

# Left-edge algorithm for coloring interval graph

- Input:
  - *Set of intervals* with ***left*** and ***right*** edge.
- Rationale:
  - Sort intervals by ***left*** edge.
  - Assign non overlapping intervals to first color using the sorted list.
  - When possible intervals are exhausted *increase color counter* and repeat.

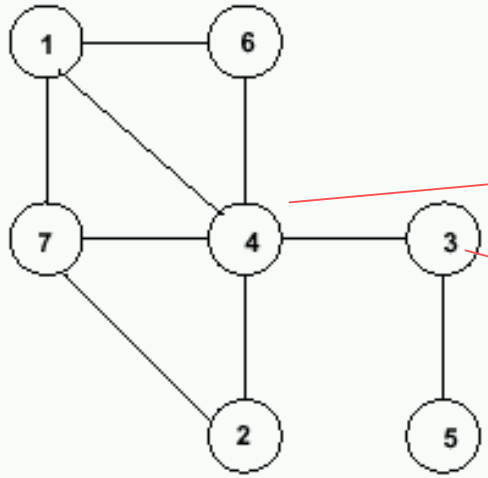
# Left-edge algorithm

```
LEFT_EDGE(I) {
        Sort elements of I in a list L in ascending order of l_i ;
        c = 0;
        while (some interval has not been colored ) do {
                S =ϕ ;
                r = 0;
                while (∃s ∈ L such that l_s > r) do {
                        s = First element in the list L with l_s > r ;
                        S = S ∪ { s } ;
                        r = r_s ;
                        Delete s from L;
                        }
                c = c +1;
                Label elements of S with color c;
            }
}
```

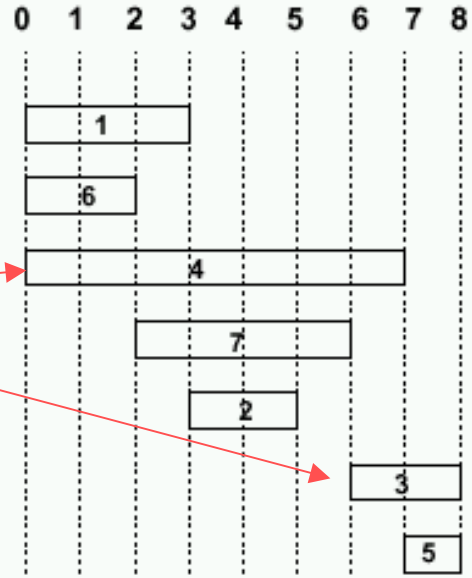
# Example of Left-edge algorithm

There is an edge in the graph when intersection of intervals is not empty

## Interval graph

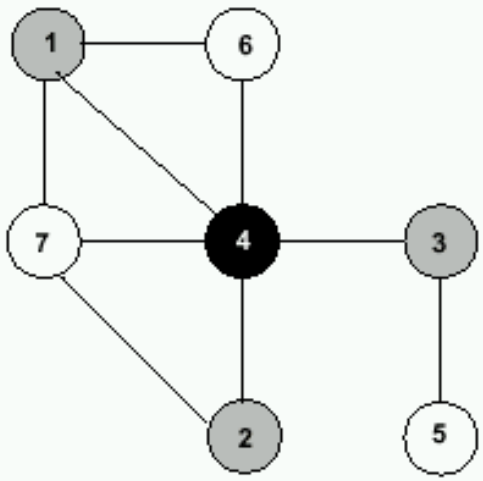



(a)

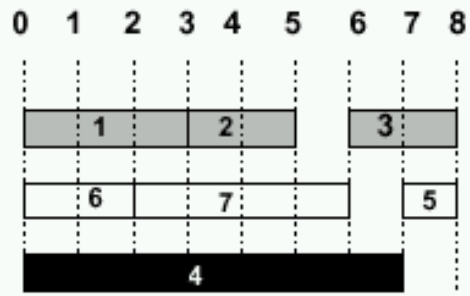



(b)

## Coloring of interval graph




(c)




(d)

# ILP formulation of binding

•We consider a similar framework to the one presented for scheduling.

•For the sake of simplicity, we assume all operations and resources have the same type.

- Boolean variables $b_{ir}$
  - Operation **i** bound to resource **r**.
- Boolean variables $x_{il}$
  - Operation **i** scheduled to start at step **l**.

The binary variable , $b_{ir}$ , is 1 only when operation vi is bound to resource r, i.e. $\beta(v_i) = (1,r)$.

The binary constant , $x_{il}$, is 1 only when operation $v_i$ starts in step *l* of the schedule, i.e. $l = t_i$. *These values are known constants , because we consider scheduled sequencing graphs.*

# ILP formulation of binding

- Boolean variables $b_{ir}$
  - Operation **i** bound to resource **r**.
- Boolean variables $x_{il}$
  - Operation **i** scheduled to start at step **l**.

Searching for a binding compatible with a given schedule (represented by X) and a resource bound a is equivalent to searching for a set of values of B satisfying the following constraints:

$$\sum_{r=1}^{a} b_{ir} = 1 \quad \forall i$$

$$\sum_{i=1}^{n_{ops}} b_{ir} \sum_{m=l\_d_i+1}^{l} x_{im} \leq 1 \quad \forall l \quad \forall r$$

$b_{ir}$ are binary

# ILP formulation of binding

- **Boolean variables $b_{ir}$**
  - **Operation i bound to resource r.**
- **Boolean variables $x_{il}$**
  - **Operation i scheduled to start at step l.**

**This constraint states that each operation vi should be assigned to one and only one resource.**

$$\sum_{r=1}^{a} b_{ir} = 1 \quad \forall i$$

$$\sum_{i=1}^{n_{ops}} b_{ir} \sum_{m=l\_d_i+1}^{l} x_{im} \leq 1 \quad \forall l \quad \forall r$$

**•This constraint states that at most one operation can be executing, among those assigned to resource r, at any time step.**

**• Note that it suffices to require the variables in B to be non-negative integers to satisfy that $b_{ir}$ are binary. Hence the problem can be formulated as an ILP and not necessarily as ZOLP.**

Example will be shown later on for analogous problem

# Resource Sharing in Hierarchical sequencing graphs

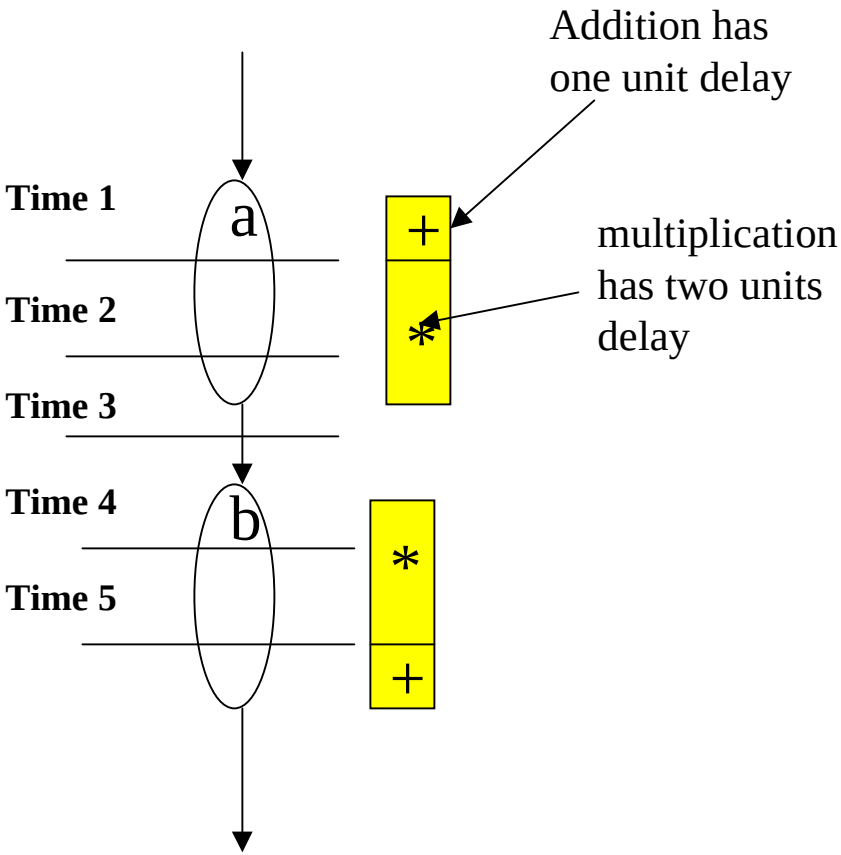
- **Hierarchical conflict/compatibility graphs**.
  - Hierarchical graphs are easy to compute.
  - Hierarchical graphs prevent sharing across hierarchy.
- **Flatten hierarchy.**
  - Flattening the hierarchy produces bigger graphs.
  - It also destroys nice properties.

# Resource Sharing in Hierarchical sequencing graphs

- **A simplistic approach to resource sharing is to perform it independently within each sequencing graph** entity.
- Such an approach is overly restrictive, because it would not allow sharing resources in different entities.
- Therefore we consider resource sharing across the hierarchy levels.
- Let us first restrict our attention to sequencing graphs where the hierarchy is induced by model calls. When two link vertices corresponding to different called models are not concurrent, any operation pair implementable by resources with the same type and in the different called models is compatible.
- Conversely, concurrency of the called models does not necessarily imply conflicts of operation pairs in the models themselves.

# Hierarchical conflicts and compatibility

- Model m1



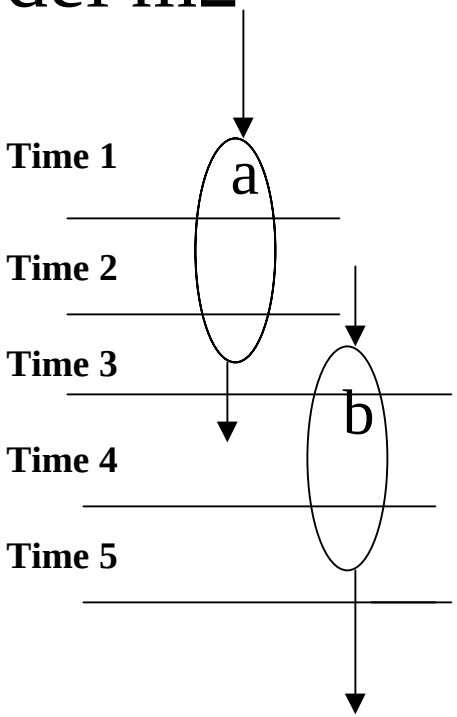

When a model m1 has a call to model **a** followed by a call to model **b**, a and b are not concurrent and the corresponding addition and multiplication are compatible

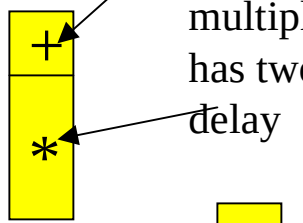
Model a = addition followed by multiplication

Model b = multiplication followed by addition

# Hierarchical conflicts and compatibility

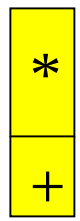
- Model m2




Addition has one unit delay

multiplication has two units delay




Model b = multiplication followed by addition

•In model m2 two calls to a and b overlap in time.

•Then we cannot say a priori that the operations of **a** and **b** are conflicting.

•Indeed the multiplications are not compatible while the additions are compatible.

Therefore the appropriate way of computing the compatibility of operations across different levels of the hierarchy is to flatten the hierarchy.

# Dealing with complex models

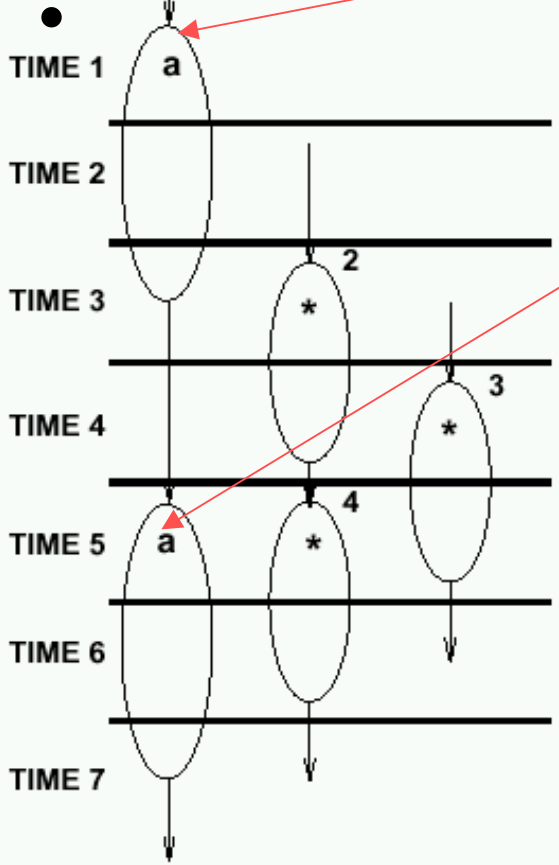
- **Flattening expansion** can be done
  - **explicitly**, by replacing the link vertices by the graphs of corresponding models.
  - **Implicitly**, by computing the execution intervals of each operation with respect to the source operation of the root model in the hierarchy
- To determine the properties of the compatibility and conflict graphs, we need to distinguish the cases when models are called once or more than once.
- In both cases, model calls make the sequencing graph representation **modular**
- In the latter case, model calls express also the sharing of the application-specific resource corresponding to the model.

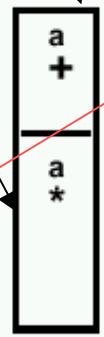
- When all models are called only once, the hierarchy is only a structured representation of the data-flow information.
- Thus compatibility and conflict graphs have the **special properties** described in the previous section.
- Let us consider now **multiple calls to a model**.
- We question the compatibility or conflict of the operations in the called model with those in the calling one, and the properties of the corresponding graphs.

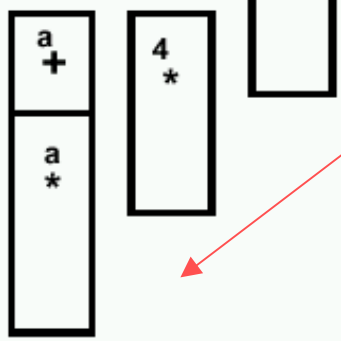
•Addition followed by multiplication

•addition has one unit delay, multiplication two

•Model a has an overall delay of 3 units.




•Consider then model m3 with two calls to model a that are not concurrent scheduled at times 1 and 5, respectively.

•Assume also that model m3 has three other multiplication operations.

•We question the sharing of the multipliers across the hierarchy.

•Note that the double call to a results in two non-contiguous execution intervals for the multiplication in a.

•As a result the conflict graph is not an intersection among intervals.

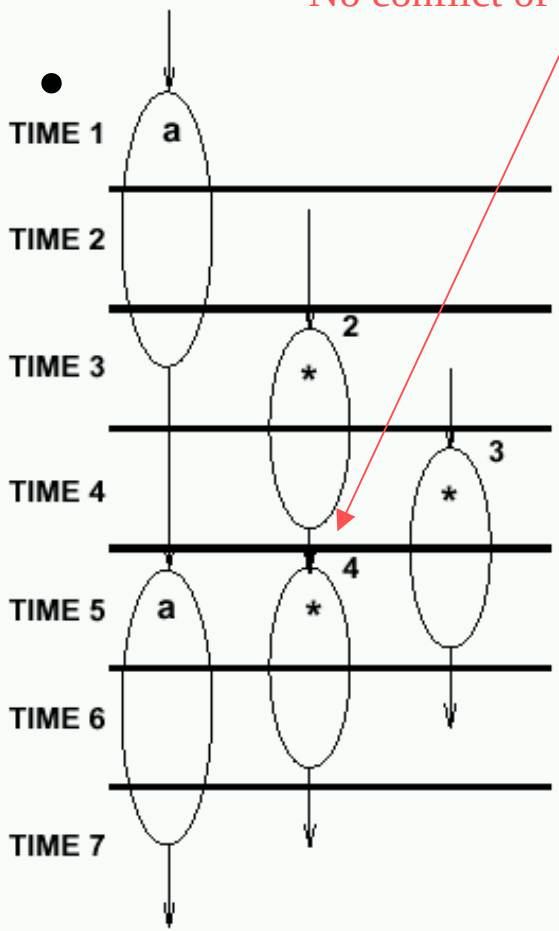
**Hierarchical sequencing graph fragment related to model m3**

Conflict graph

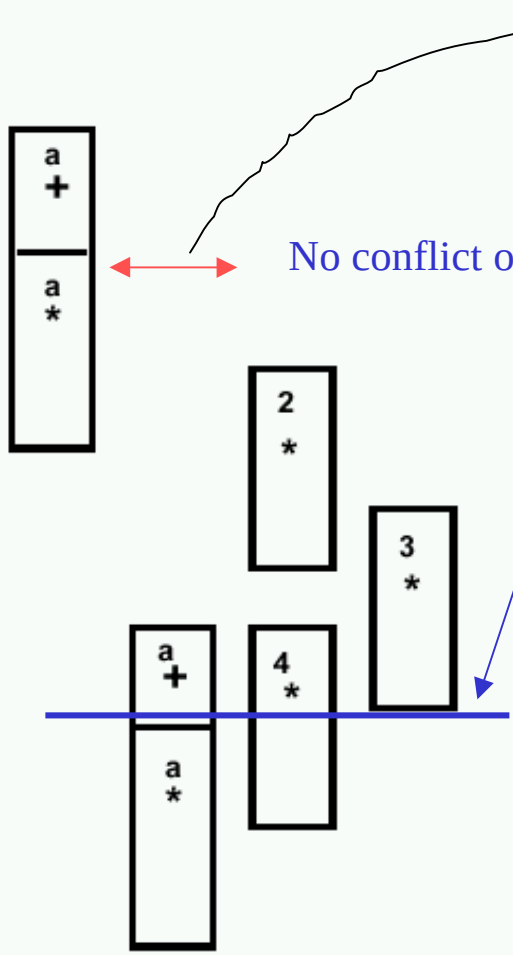
(c)

# Example of Hierarchical conflict/compatibility graphs

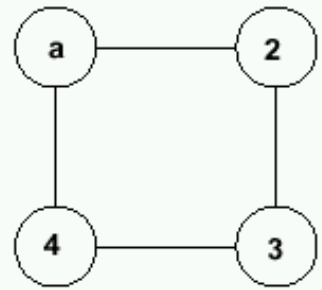
- **Addition followed by multiplication**
- **addition has one unit delay, multiplication two**




**Hierarchical sequencing graph fragment related to model m3**




Conflict graph




**Non-chordal conflict graph**

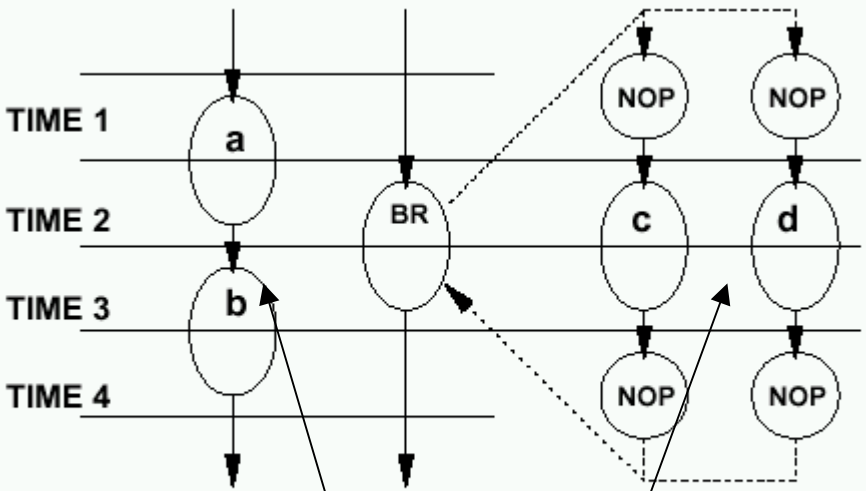
The resulting graphs may no longer have special properties. They clique partitioning and vertex coloring are now intractable problems.

# Iterative constructs that can be unrolled and branching constructs

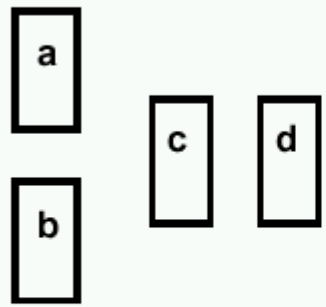
- The Compatibility of the operations across the hierarchy can be computed in a similar way in the presence of iterative constructs that can be unrolled. Note that a resource bound to one operation in a loop corresponds to a resource bound to multiple instances of that operation when the loop is unrolled.
- Moreover that resource may be bound to other operations outside the loop model. Note also that a single model call inside a loop body becomes a multiple call when the loop body is unrolled.
- Let us consider now branching constructs
- When considering operation pairs in two alternative branching bodies, their compatibility corresponds to having the same type.
- The computation of compatibility and conflict graphs can still be done by traversing the hierarchy and using Definitions 6.2.1 and 6.2.2.
- The resulting compatibility and conflict graphs may not have any special property, as shown below.

# Example of Hierarchical conflict/compatibility graphs

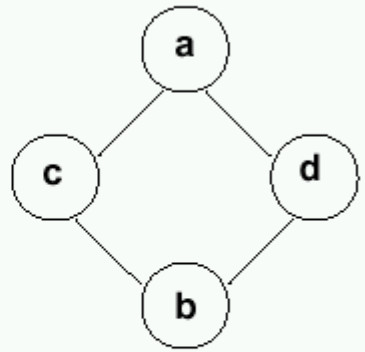
We assume that all operations take two time units.




**Hierarchical sequencing graph fragment - conditional execution**




Execution intervals




Non-chordal conflict graph

c and d are compatible because executed in parallel

a and b are compatible

Note that the alternative nature of operations c and d makes them compatible and prevents a chord $\{v_c, v_d\}$ to be present in the conflict graph. Hence the conflict graph is not an interval graph.

# Register binding problem

Consider registers that hold values of variables

- Given a schedule:
  - ***Lifetime intervals*** for variables. Interval from birth to death.
  - ***Lifetime overlaps***.
- Conflict graph (interval graph).
  - Vertices <--> variables.
  - Edges <--> overlaps.
  - Interval graph.
- Compatibility graph (***comparability graph***).
  - Complement of conflict graph.

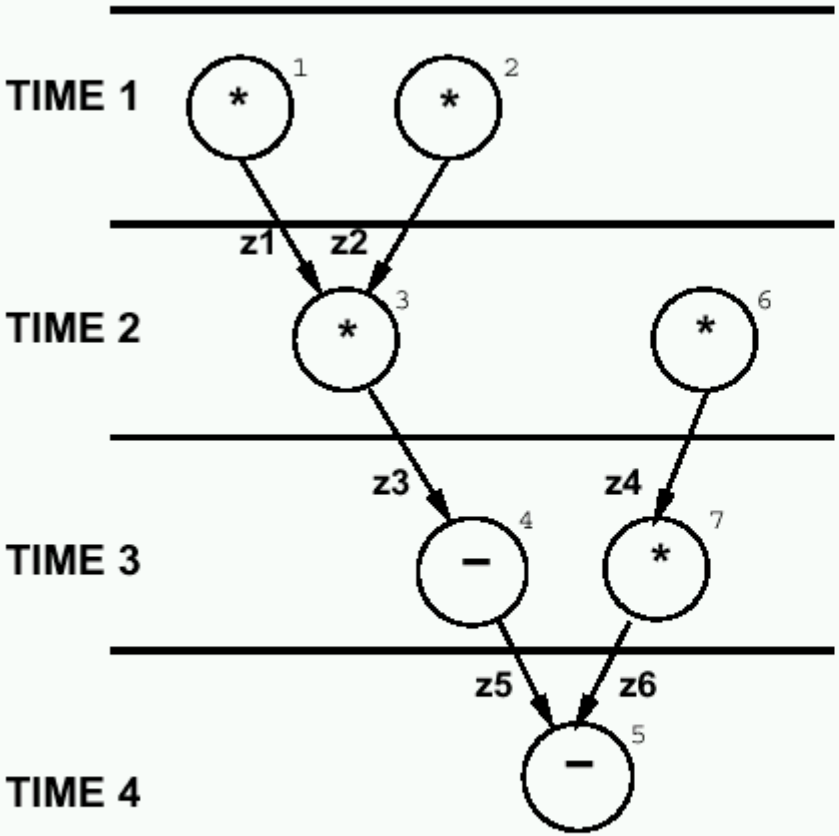
As shown before, the variables with multiple assignments within one model are aliased, so that each variable has a single lifetime in the frame of reference corresponding to the sequencing graph entity where it is used.

# Register sharing data-flow graphs

- **Given:**
  - Variable lifetime conflict graph.
- **Find:**
  - Minimum number of registers storing all the variables.
- **Key point:**
  - Interval graph:
    - **Left-edge algorithm. (Polynomial-time).**

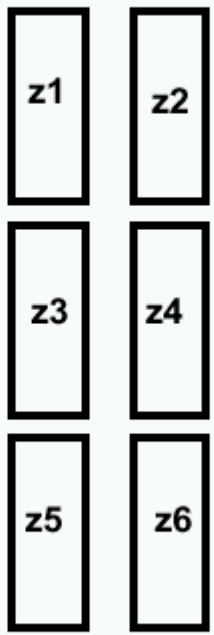
# Example of Register sharing data-flow graphs

There are six intermediate variables that must be stored in registers



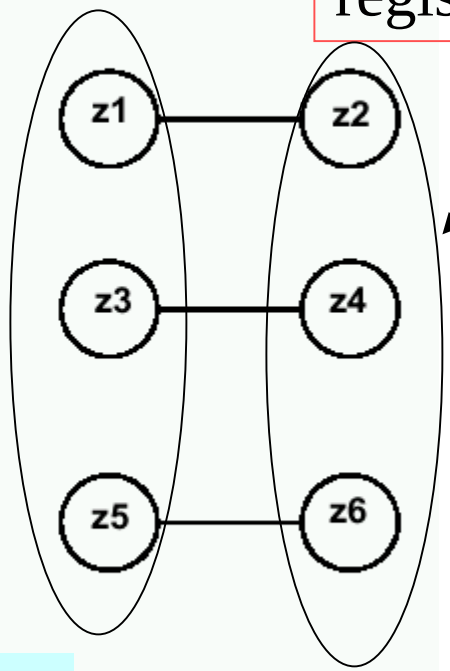

Hierarchical sequencing graphs

Lifetime conflicts




Variable intervals

We need 2 registers




Conflict graph

# Register sharing - general case

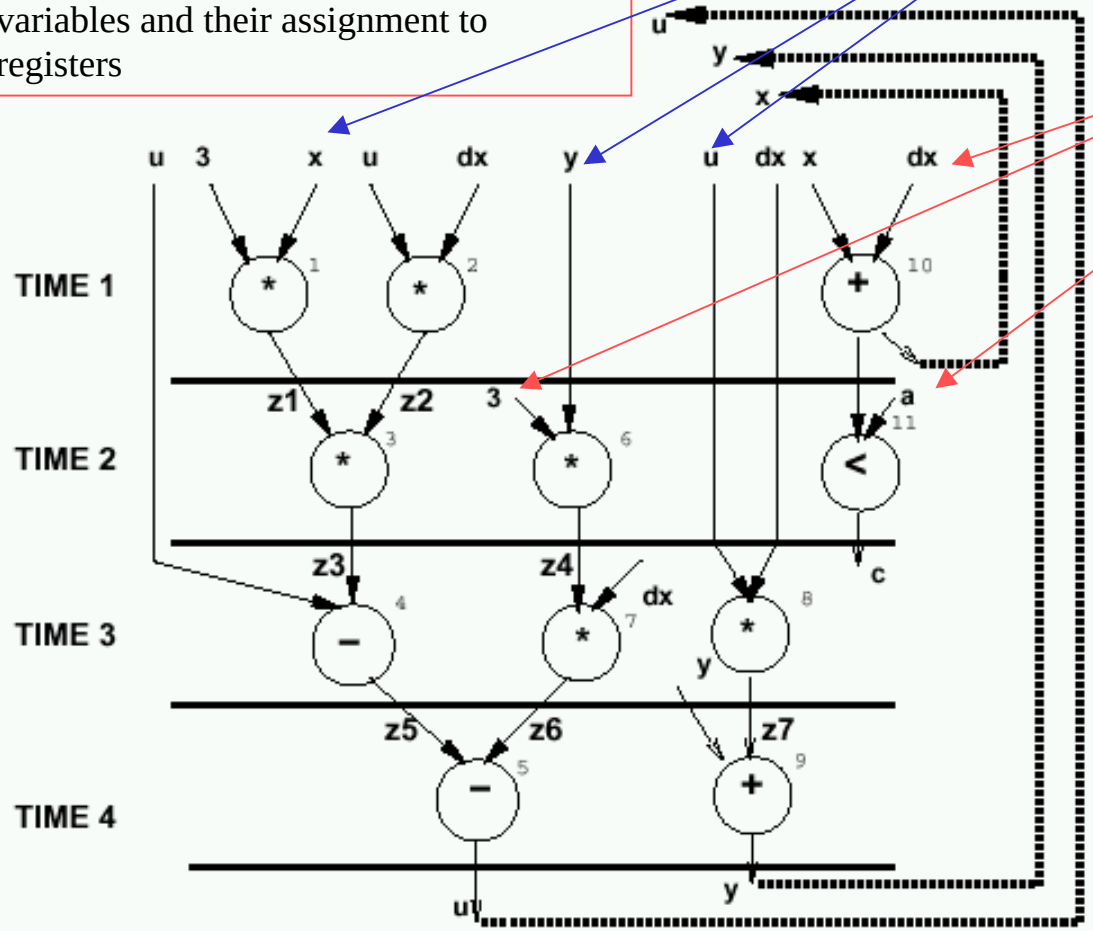
Sequencing models of iterative bodies. In this case, some variables are alive across the iteration boundary. For example, the loop-counter variable. The cyclicity of the lifetimes is modeled accurately by circular-arc graphs that represent intersection of arcs on a circle

- Iterative constructs:
  - Preserve values across iterations.
  - ***Circular-arc*** conflict graph:
    - Coloring is intractable.
- Hierarchical graphs:
  - General conflict graphs:
    - Coloring is intractable.
- Heuristic algorithms.

# Example of Register sharing-general case

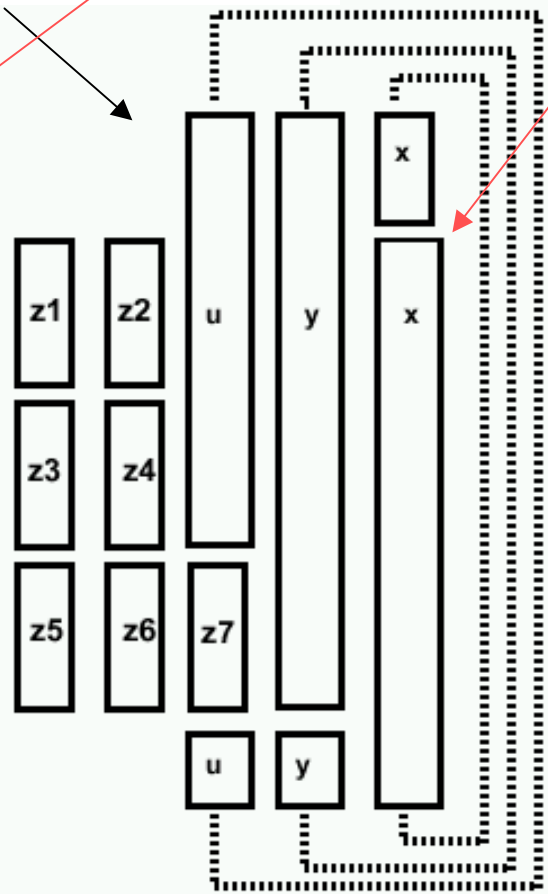
7 intermediate variables, $z_i$ , 3 loop variables (x,y,u), and 3 loop invariants (a,3,dx)

We consider intermediate and loop variables and their assignment to registers




Variable lifetimes

New x created here



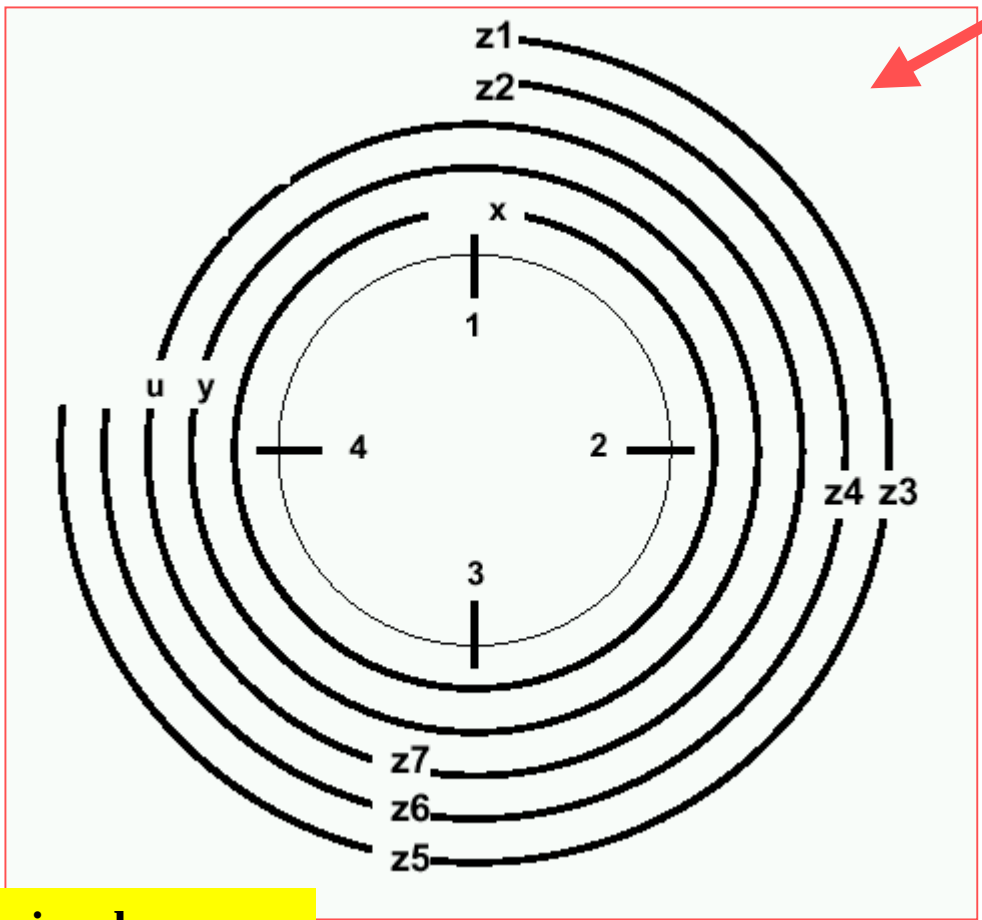

Hierarchical sequencing graph

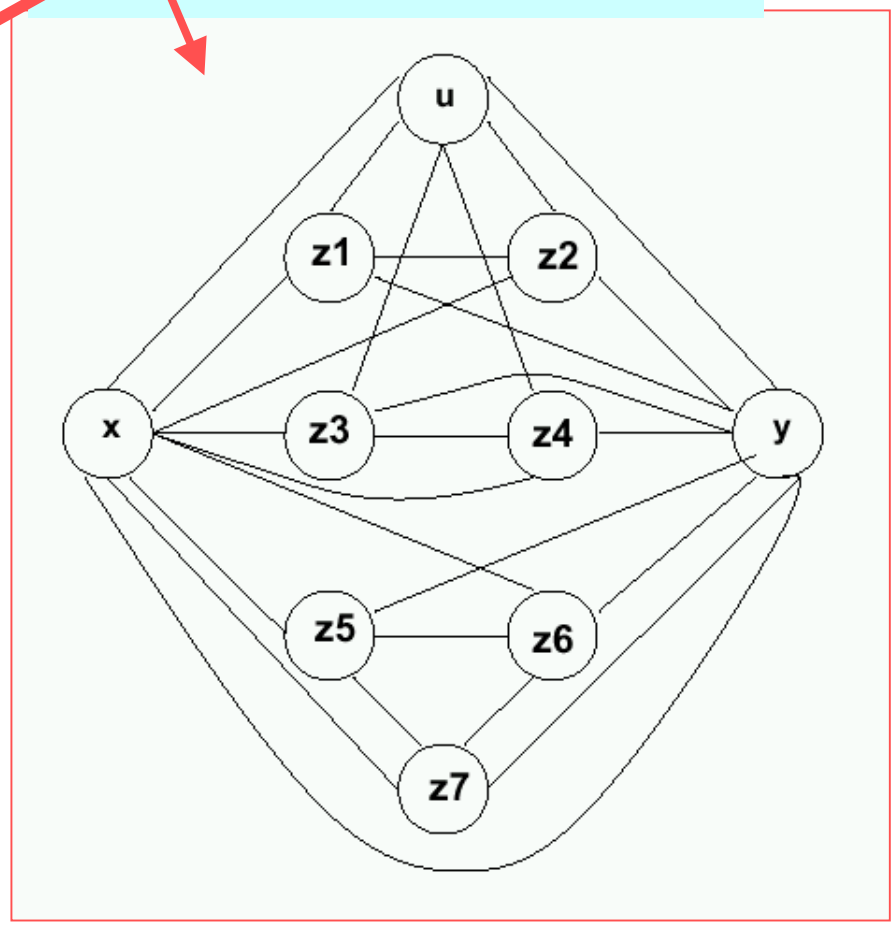
This leads to circular conflict graph

## *Example continued*

# Variable-lifetimes and circular-arc conflict graph

Five registers suffice to store the 7+3=10 intermediate and loop variables




**circular-arc conflict graph**

**Variable lifetimes as arcs on a circle**

**Circular-arc conflict graph**

# Circular graphs

- The register sharing problem can then be cast as a minimum coloring of a circular-arc graph.This problem is intractable.
- Stok has showed that this problem can be transformed to multi-commodity flow problem and solved more efficiently.

- Register sharing can be extended to hierarchical models.
- In general such graphs may have no special property.
- Springer and Thomas - polynomial time conflict graphs can be achieved by enforcing some restrictions on the model calls and on branch types.
- Can be reduced to ILP.

# Multiport-memory binding

- Find ***minimum number of ports*** to access the required number of variables.
- Variables use the same port:
  - Port compatibility/conflict.
  - Similar to resource binding.
- Variables can use any port:
  - Decision variable $x_{il}$ is TRUE when variable $i$ is accessed at step $l$.
  - **Optimum:**

$$\max_{1 \le l \le \lambda+1} \sum_{i=1}^{n_{var}} x_{il}.$$

# Multiport-memory binding

- We consider now using multi-port memory arrays to store the values of the variables.
- Let us assume a memory with a ports for either read or write requiring one cycle per access
- Such a memory array can be  a general purpose register file common to RISC architectures.
- We assume the memory to be large enough to hold all data.
- We consider in this section non-hierarchical sequencing graphs. Extensions are straightforward.
- First problem - computing the minimum number of ports a required to access as many variables as needed
- If each variable accesses the memory always through the same port, then the problem reduces to binding variables to ports.
- Thus the considerations for functional resource binding from sec. 6.2.1 can be applied to the ports, which can be seen as interface resources.

# Multiport-memory binding: Balakrishnan

- Find ***maximum number of variables*** to be stored through a fixed number of ports **a**.
  - Subject to port limitations.
- Formulation
  - Boolean variables $\{b_i, i = 1, 2, \ldots, n_{var}\}$:
    - Variable i is stored in array.

- $\max \sum_{i=1}^{n_{var}} b_i$ such that

- $\sum_{i=1}^{n_{var}} b_i \, x_{il} \leq a \qquad l = 1, 2, \ldots, \lambda + 1$

# Example formulation for Multiport-memory binding

Consider the following scheduled sequence of operations, which require the storage of variables $z_i$, i=1,…15.

$$
\begin{aligned}
Time-step\ 1 &: r_3 = r_1 + r_2\ ;\ r_{12} = r_1 \\
Time-step\ 2 &: r_5 = r_3 + r_4\ ;\ r_7 = r_3 * r_6\ ;\ r_{13} = r_3 \\
Time-step\ 3 &: r_8 = r_3 + r_5\ ;\ r_9 = r_1 + r_7\ ;\ r_{11} = r_{10}/r_5 \\
Time-step\ 4 &: r_{14} = r_{11} \wedge r_8\ ;\ r_{15} = r_{12} \vee r_9 \\
Time-step\ 5 &: r_1 = r_{14}\ ;\ r_2 = r_{15}
\end{aligned}
$$

$$\max \sum_{i=1}^{15} b_i \text{ such that}$$

$$
\begin{aligned}
b_1 + b_2 + b_3 + b_{12} &\leq a \\
b_3 + b_4 + b_5 + b_6 + b_7 + b_{13} &\leq a \\
b_1 + b_3 + b_5 + b_7 + b_8 + b_9 + b_{10} + b_{11} &\leq a \\
b_8 + b_9 + b_{11} + b_{12} + b_{14} + b_{15} &\leq a \\
b_1 + b_2 + b_{14} + b_{15} &\leq a
\end{aligned}
$$

- Let us consider memory with a ports. Then the problem can be represented by maximizing SUM i=1to15 of $b_i$ under the following constraints.

$$
\begin{aligned}
b_1 + b_2 + b_3 + b_{12} &\leq a \\
b_3 + b_4 + b_5 + b_6 + b_7 + b_{13} &\leq a \\
b_1 + b_3 + b_5 + b_7 + b_8 + b_9 + b_{10} + b_{11} &\leq a \\
b_8 + b_9 + b_{11} + b_{12} + b_{14} + b_{15} &\leq a \\
b_1 + b_2 + b_{14} + b_{15} &\leq a
\end{aligned}
$$

# Example solution for Multiport-memory binding

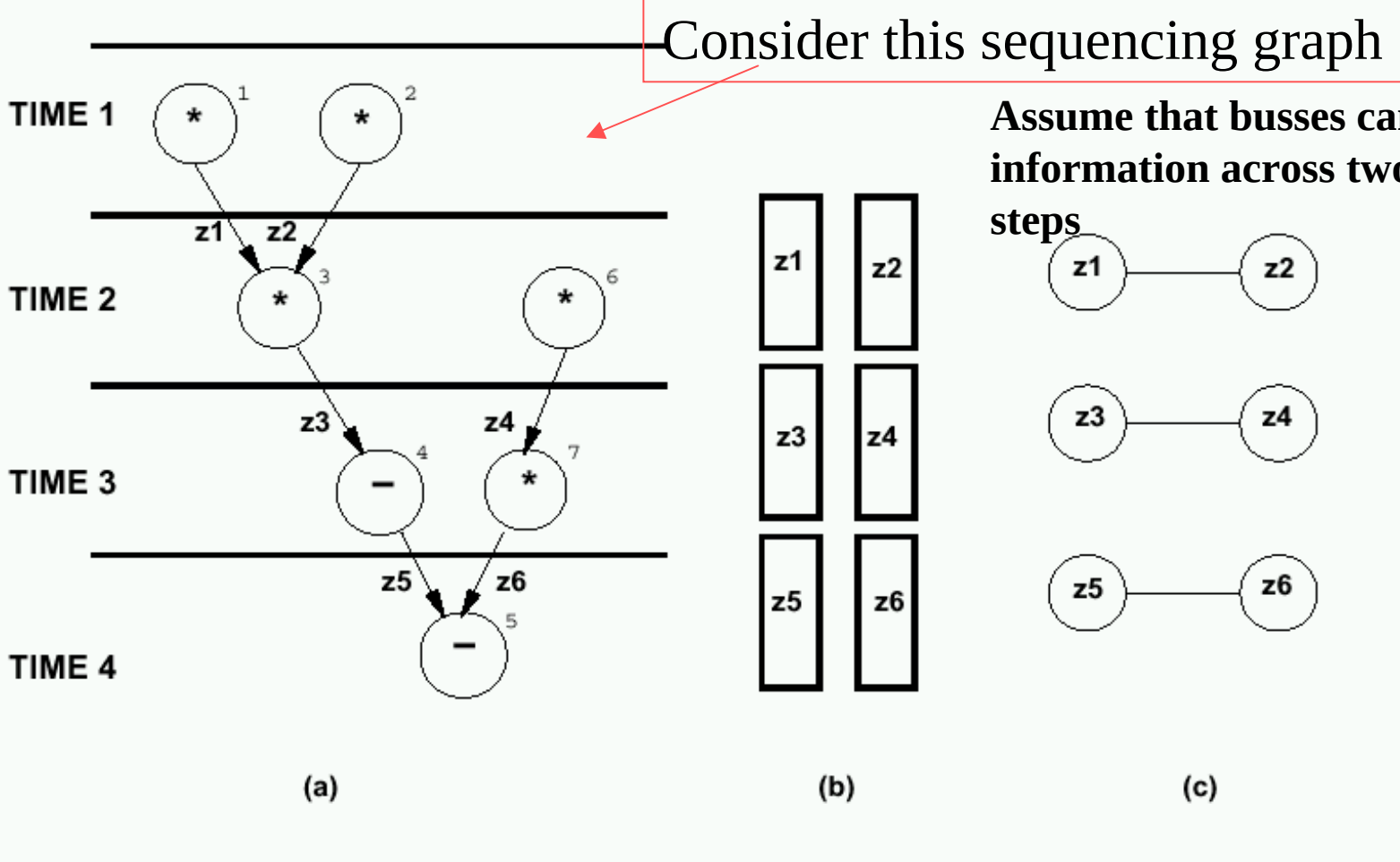
- One port a = 1:
  - $\{b_2, b_4, b_8\}$ non-zero.
  - Only 3 variables stored in memory: $v_2, v_4, v_8$ .
- Two ports a = 2:
  - 6 variables stored: $v_2, v_4, v_5, v_{10}, v_{12}, v_{14}$
- Three ports a = 3:
  - 9 variables stored: $v_1, v_2, v_4, v_6, v_8, v_{10}, v_{12}, v_{13}, v_{14}$

# Bus sharing and binding

- The operation of writing a specific bus can be modeled explicitly as a vertex in the sequencing graph model.
- In this case, the compatible (or conflicting) data transfers may be represented by compatibility (or conflict) graphs, as in the case of functional resources.
- Alternatively, buses may be explicitly described by sequencing graph model.
- Their optimum usage can be derived by exploiting the timing of the data transfers.
- Since busses have no memory, we consider only the transfers of data within each schedule step (or across two adjacent schedule steps, when we assume that the bus transfer is interleaved with computation).
- Two problems arise,
  - first to find the minimum number of buses to accommodate all (Or part of) the data transfers
  - second to find the maximum number of data transfers that can be done through a given number of busses.
- These problems are analogous to the multi-port binding problem and can be modeled using ILP constraints.

# Bus sharing and binding

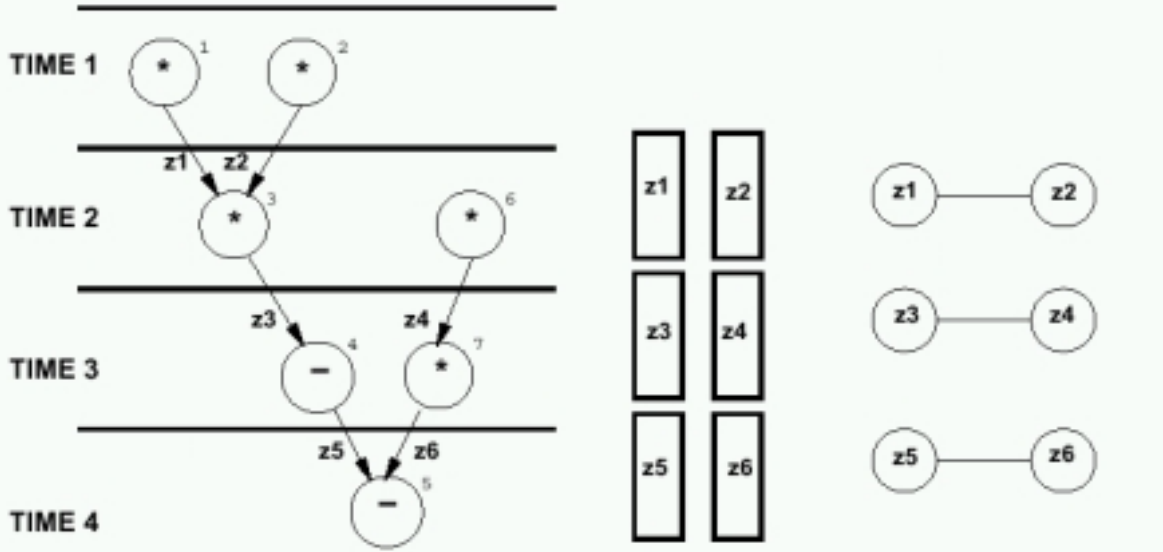
- Find the ***minimum number of busses*** to accommodate all data transfer.
- Find the ***maximum number of data transfers*** for a fixed number of busses.
- Similar to memory binding problem.
- ILP formulation or heuristic algorithms.




**Assume that busses can transfer the information across two adjacent steps**

- **One bus:**
  - 3 variables can be transferred on the bus. For example $z_1$,$z_3$, $z_5$
- **Two busses:**
  - All variables can be transferred.

**Example of Bus sharing and binding**




Let the timing scheduled data transfers be modeled by constants $X=\{x_{il}, I=1,..6, l=1,..5\}$. The values of the elements of X are all zeros, except for $x_{11}$, $x_{21}$, $x_{32}$, $x_{42}$, $x_{53}$ and $x_{63}$ which are 1's.

Then the equation

$$- \max \sum_{i=1}^{n_{var}} b_i \text{ such that}$$

$$- \sum_{i=1}^{n_{var}} b_i \, x_{il} \leq a \qquad l = 1, 2, \ldots, \lambda + 1$$

yields: $b_1+b_2<a$, $b_3+b_4<a$, $b_5+b_6<a$

This gives the same two solutions as before for a=1 and a=2

# Scheduling and binding
# Resource dominated circuits

- Area and delay of resources dominate.
- ***Strategy:***
  - Scheduling under area constraints:
    - Minimize latency.
  - Binding.
    - Share resource within bounds.
- Decoupling between scheduling and binding.

# Scheduling and binding General circuits

- Area and delay influenced by:
  - *Sparse logic,*
  - *wiring,*
  - *registers and control circuit*.
- Binding affects the *cycle-time*:
  - It may invalidate a schedule.
- Scheduling after binding:
  - Binding under restrictive assumptions.
  - Time-frame of operations not yet known.

# Scheduling and binding approaches

- ***Concurrent*** scheduling and binding.
  - ILP model- exact.
  - Some heuristic algorithms.
- *Scheduling before binding*:
  - Good for DSP application.
- *Binding before scheduling*:
- **Iterative** techniques.

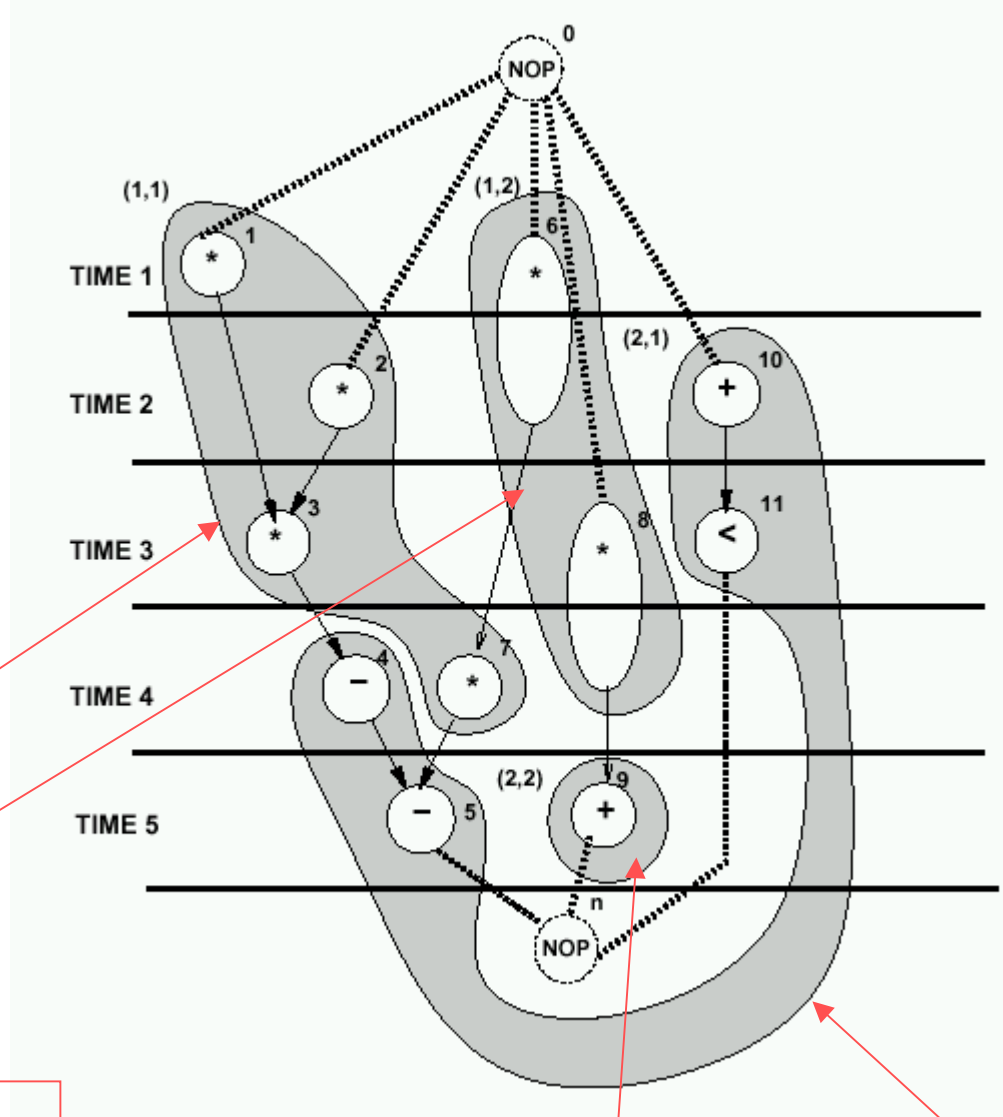
# Module selection problem

- Library of resources:
  - More than one resource per type.
- Example:
  - Ripple-carry adder.
  - Carry look-ahead adder.
- **Resource modeling:**
  - Resource ***subtypes*** with:
    - ***(area, delay)*** parameters.

# Module selection solution

- **ILP formulation:**
  - Decision variables:
    - Select resource sub-type.
    - Determine ***(area, delay).***
- **Heuristic algorithms:**
  - Determine **minimum latency** with fastest resource subtypes.
  - Recover area by using slower resources on non-critical paths.

# Example of Module selection solution




**First multiplier**

**Second multiplier**

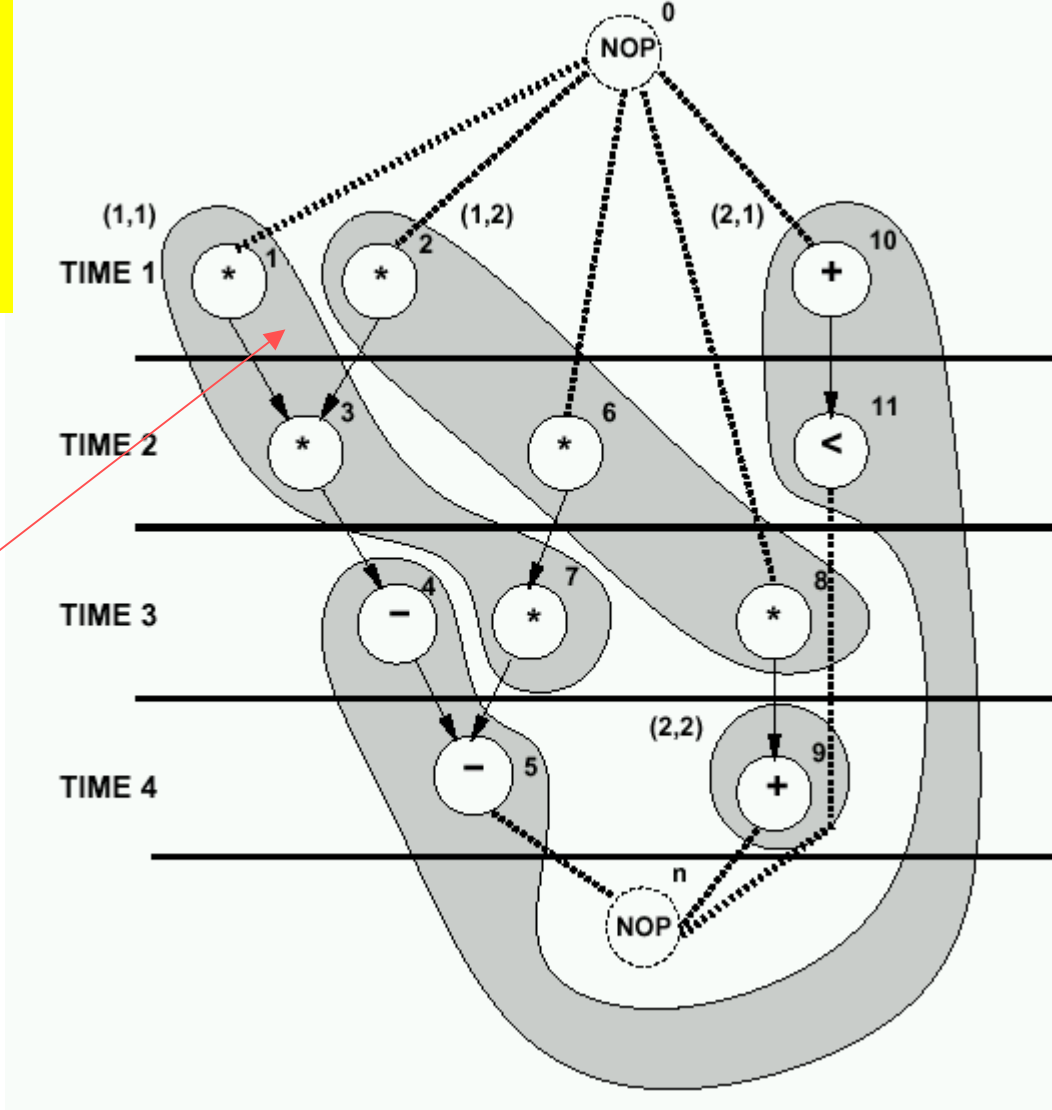
**First ALU**

**Second ALU**

- Multipliers with: area
  - (Area, delay) = (5,1) and (2,2)
- Latency bound of 5.

# Second Example of Module selection solution for the same problem

- Latency bound of 4 (which is better!).
  - Fast multipliers for {$v_1$ , $v_2$ , $v_3$}.
  - Slower multipliers can be used elsewhere.
    - Less sharing.
- *Minimum-area design* uses fast multipliers only.




2 multipliers

2 ALUs

# Summary

- Resource sharing is reducible to ***coloring/clique-covering.***
- Simple for *flat graphs*.
- Intractable, but still easy in practice, for other graphs.
- More complicated for non resource-dominated circuits.
- Extension: module selection.