RESOURCE SHARING

© Giovanni De Micheli
Stanford University

Outline

- 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 resurce
- ***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.

Optimum sharing problem

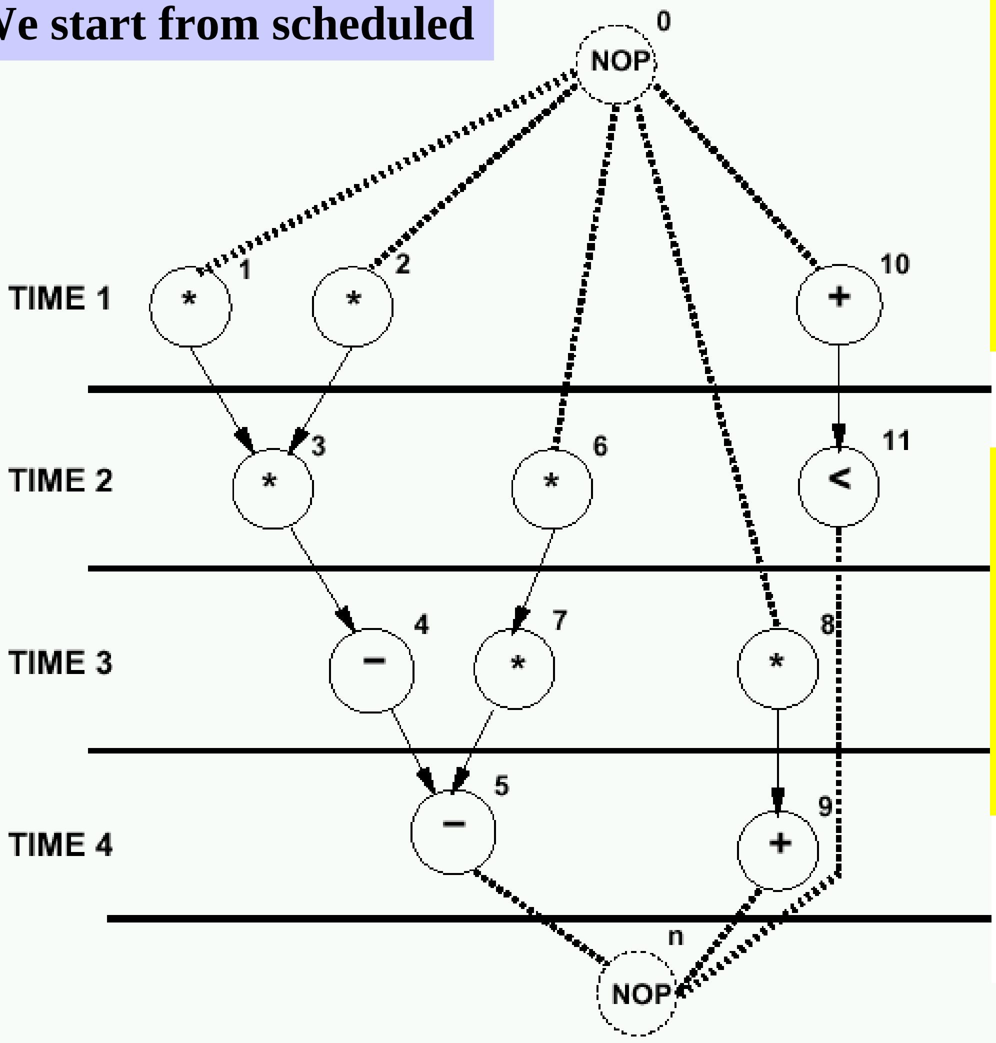
- We start from scheduled sequencing graphs.
  - Operation concurrency is well defined.
- We consider ***operation types*** independently.
  - Problem decomposition.
  - Perform analysis for each resource type.

Compatibility graphs and conflict graphs

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

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

We start from scheduled

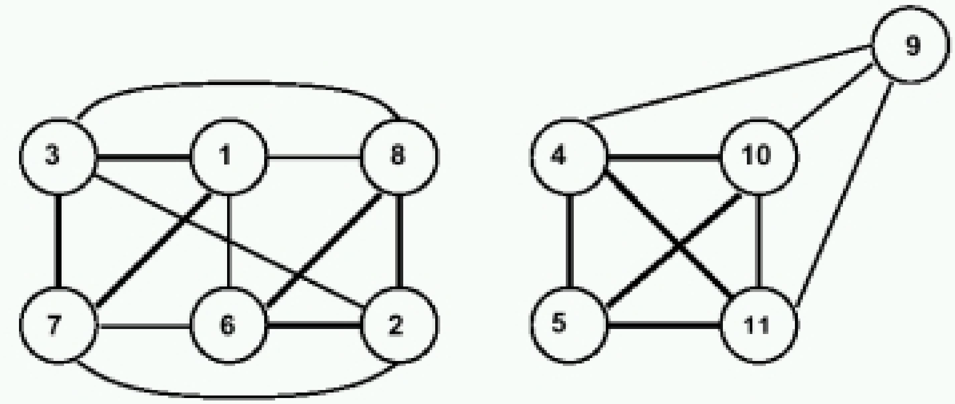


Examples of Compatibility graphs and conflict graphs

Compatibility graph is a complement of a conflict graph

Multiplier ALU

Compatibility graphs

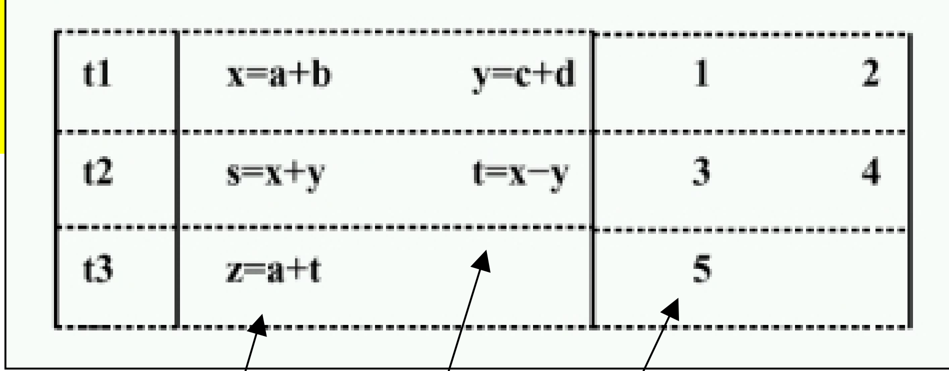


Observe that 1 and 2 are not compatible since they are executed concurrently

Algorithmic solution to the optimum binding problem

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

Examples of using conflict and compatibility graphs for binding



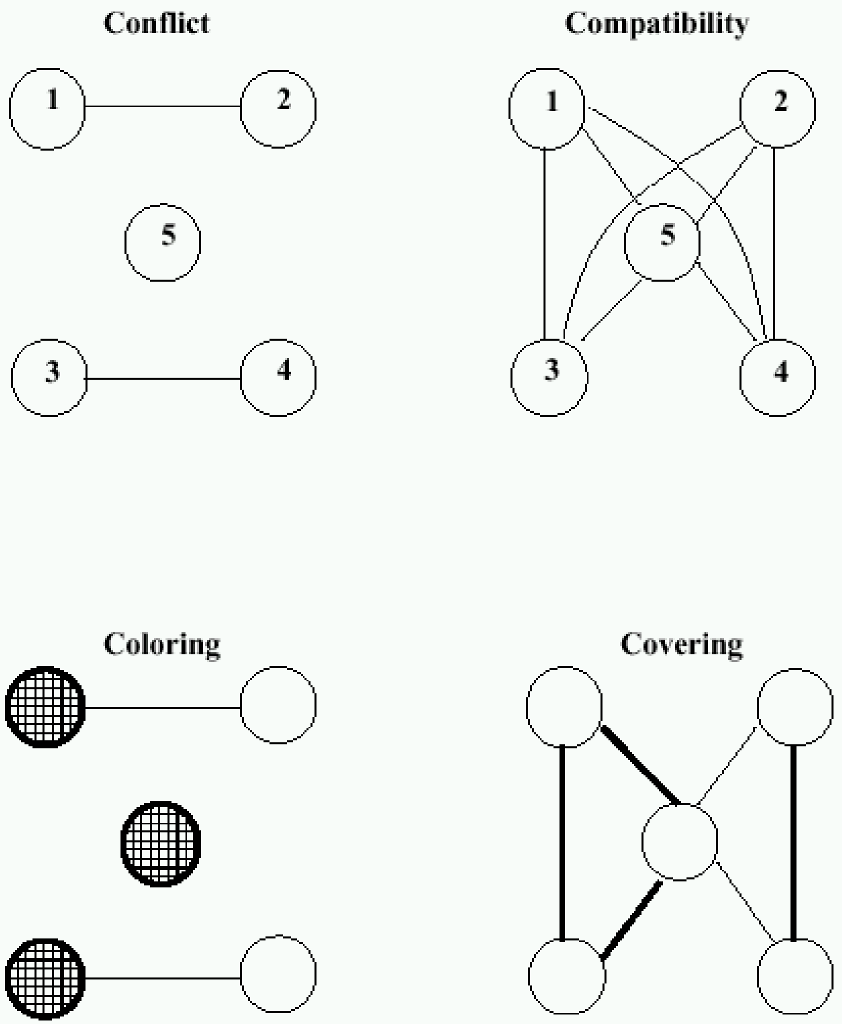
t1	x=a+b	y=c+d	1	2
t2	s=x+y	t=x−y	3	4
t3	z=a+t		5	

We start from scheduled

resources

y,t

x, s, z



ALU1: 1,3,5

ALU2: 2,4

Perfect graphs

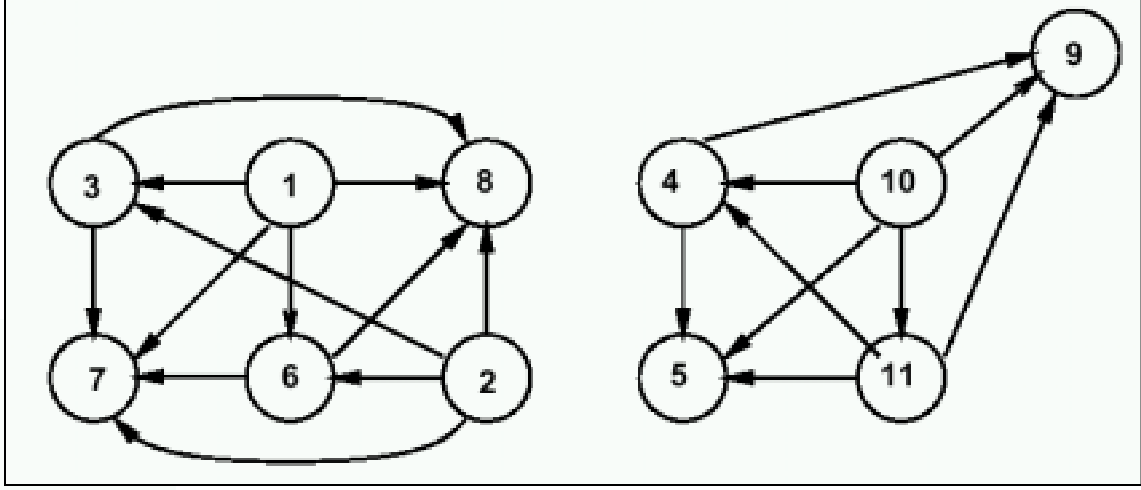
- ***Comparability graph:***
  - Graph G(V, E) has an orientation G(V, F) with the transitive property.
  - $(v_i, v_j) \in F \wedge (v_j, v_k) \in F) \Rightarrow (v_i; v_k) \in F$.
- Interval graph:
  - Vertices correspond to ***intervals.***
  - Edges correspond to ***interval*** intersection.
  -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

What is a Perfect graph?

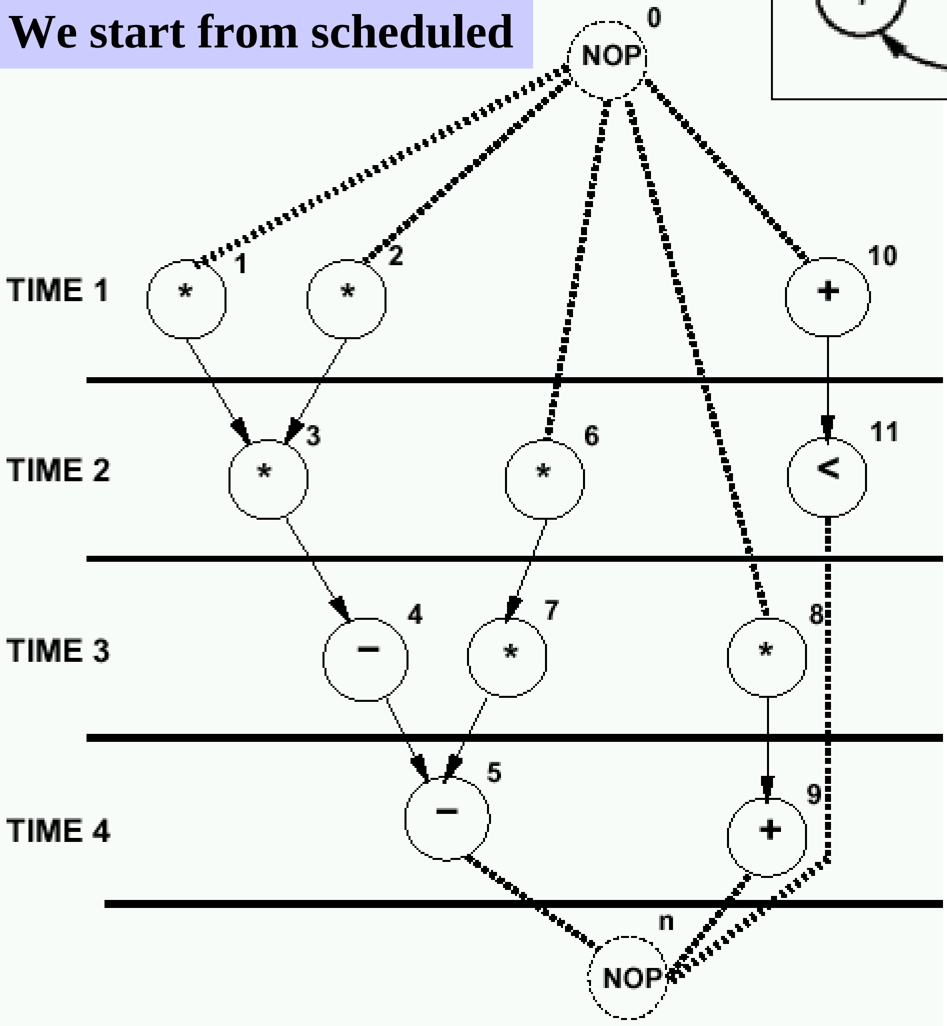
Data-flow graphs (at sequencing graphs)

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

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

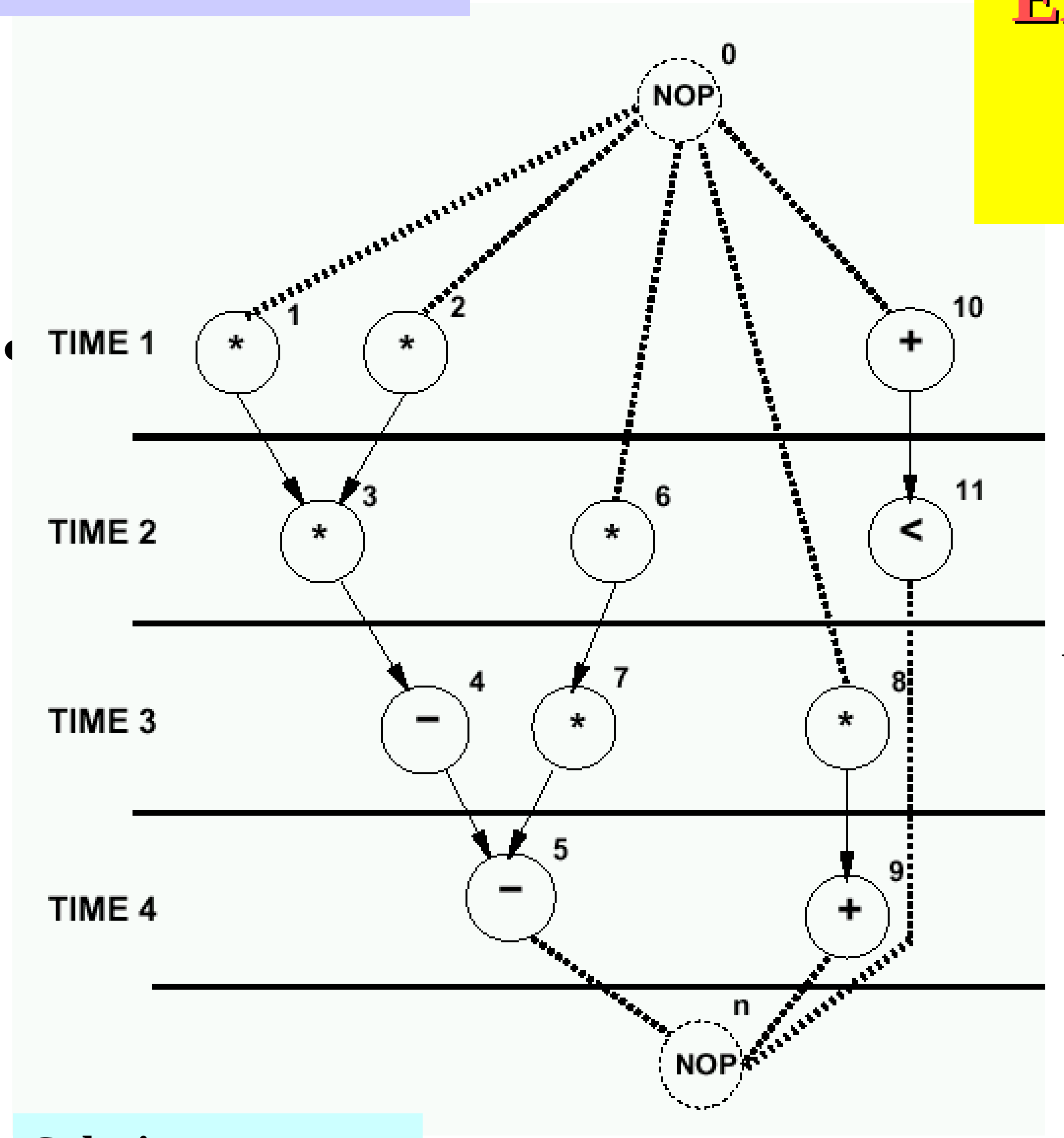
5 = ALU

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

Solution
Latency = 4
Multipliers = 2
ALU = 2

We start from scheduled

Example of using conflict graph which is an interval graph



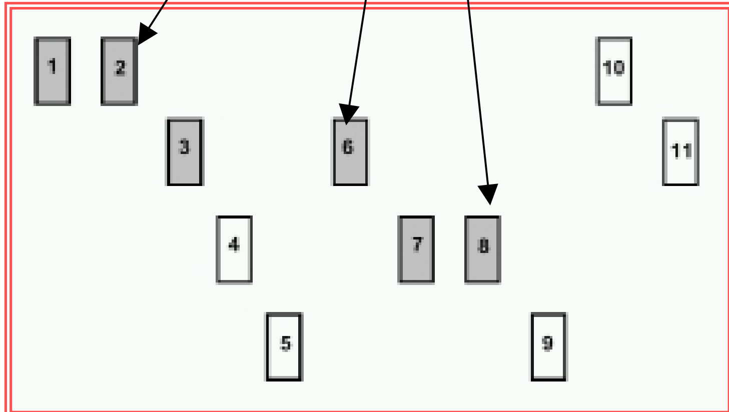
{1,2,10}

{3,6,11}

{4,7,8}

{5,9}

As we see, 1,2,3,6,7,8 can have the same color grey



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

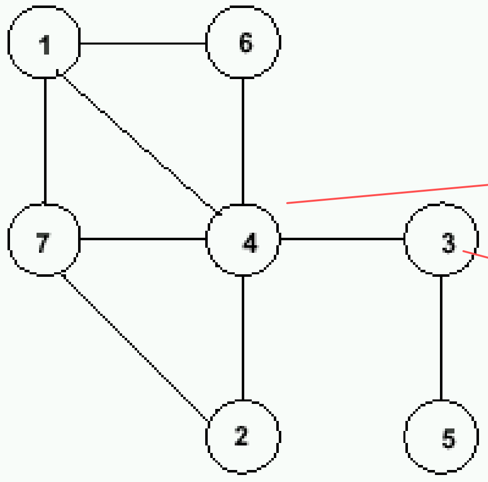
- Input:
  - Set of intervals with ***left*** and ***right*** edge.
- Rationale:
  - Sort intervals by ***left*** edge.
  - Assign non overlapping intervals to first color using the sorted list.
  - When possible intervals are exhausted *increase color counter* and repeat.

Left-edge algorithm

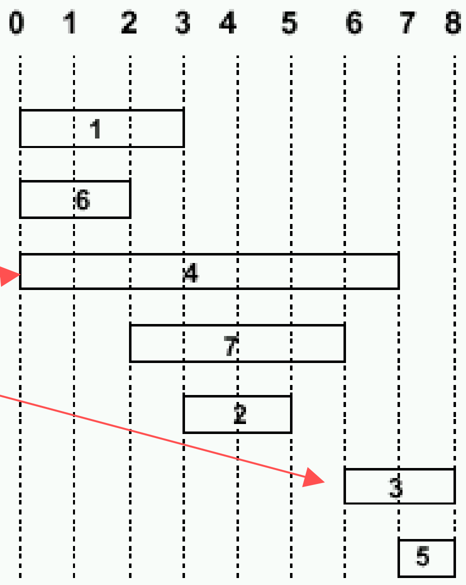
```
LEFT_EDGE(I) {
        Sort elements of I in a list L in ascending order of $l_i$ ;
        c = 0;
        while (some interval has not been colored ) do {
                S = $\phi$ ;
                r = 0;
                while ($\exists s \in$ L such that $l_s > r$) do {
                     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

Example of Left-edge algorithm

Interval graph

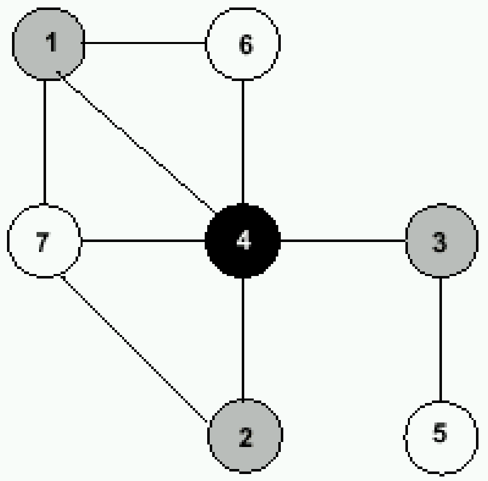


(a)

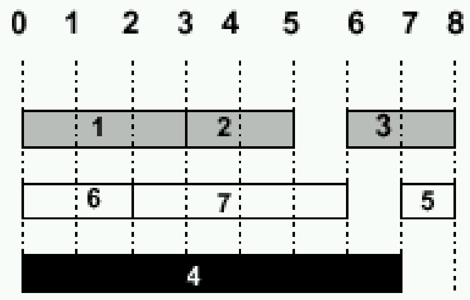


(b)

Coloring of interval graph



(c)



(d)

Last slide for today

ILP formulation of binding

- Boolean variables b $_{ir}$
  - Operation **i** bound to resource **r**.
- Boolean variables **x** $_{i\,l}$
  - Operation **i** scheduled to start at step **l**.

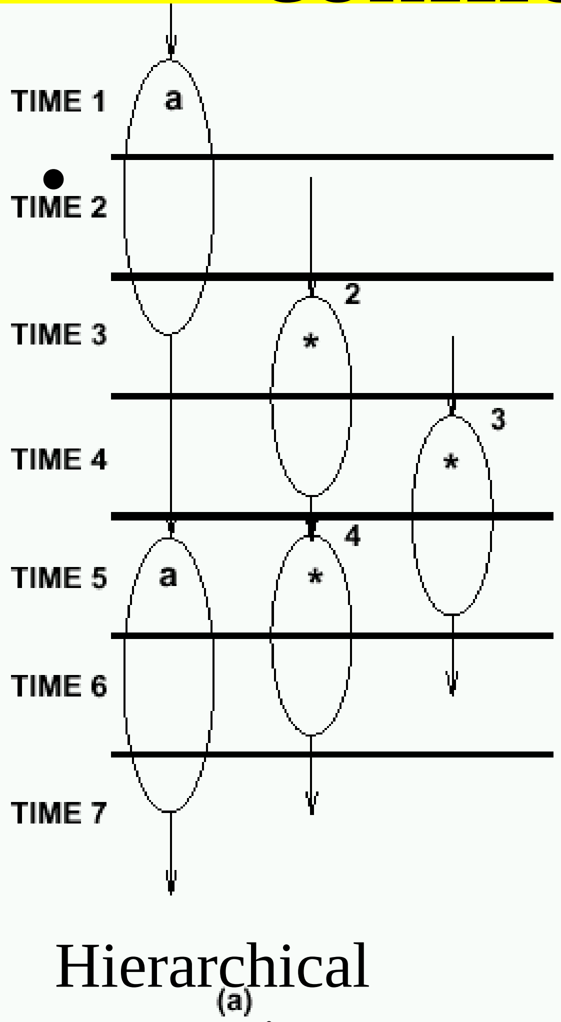
$$\sum_{r=1}^{a} b_{ir} = 1 \qquad \forall i$$

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

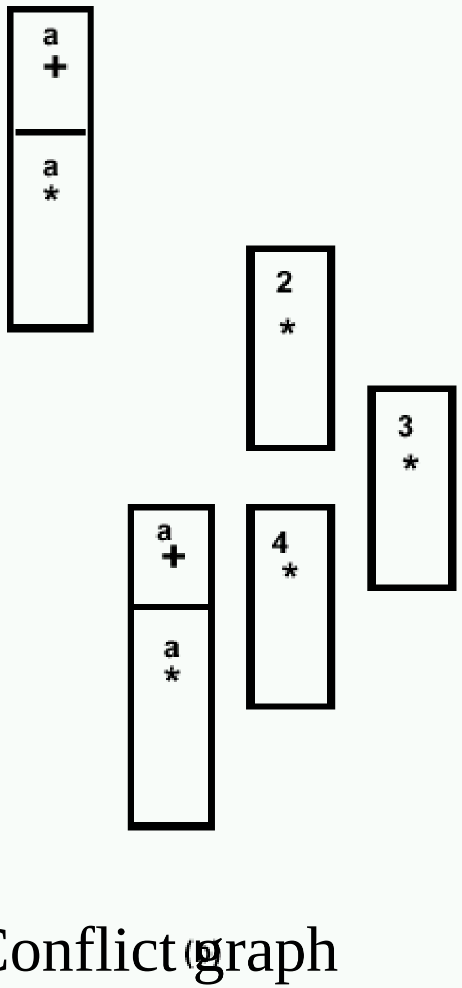
Hierarchical sequencing graphs

- **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.

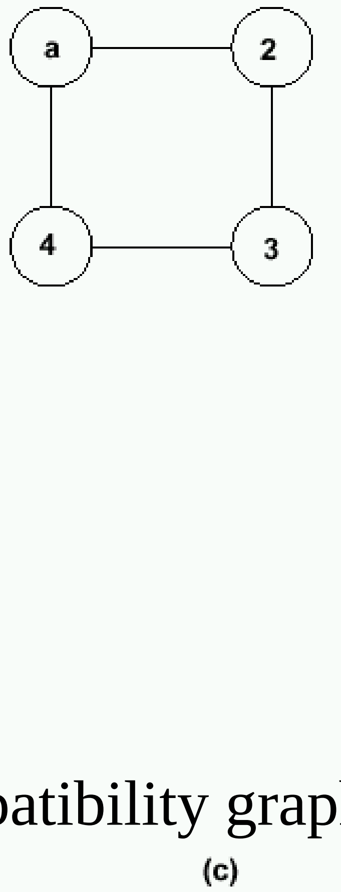
Example of Hierarchical conflict/compatibility graphs



Hierarchical sequencing graphs

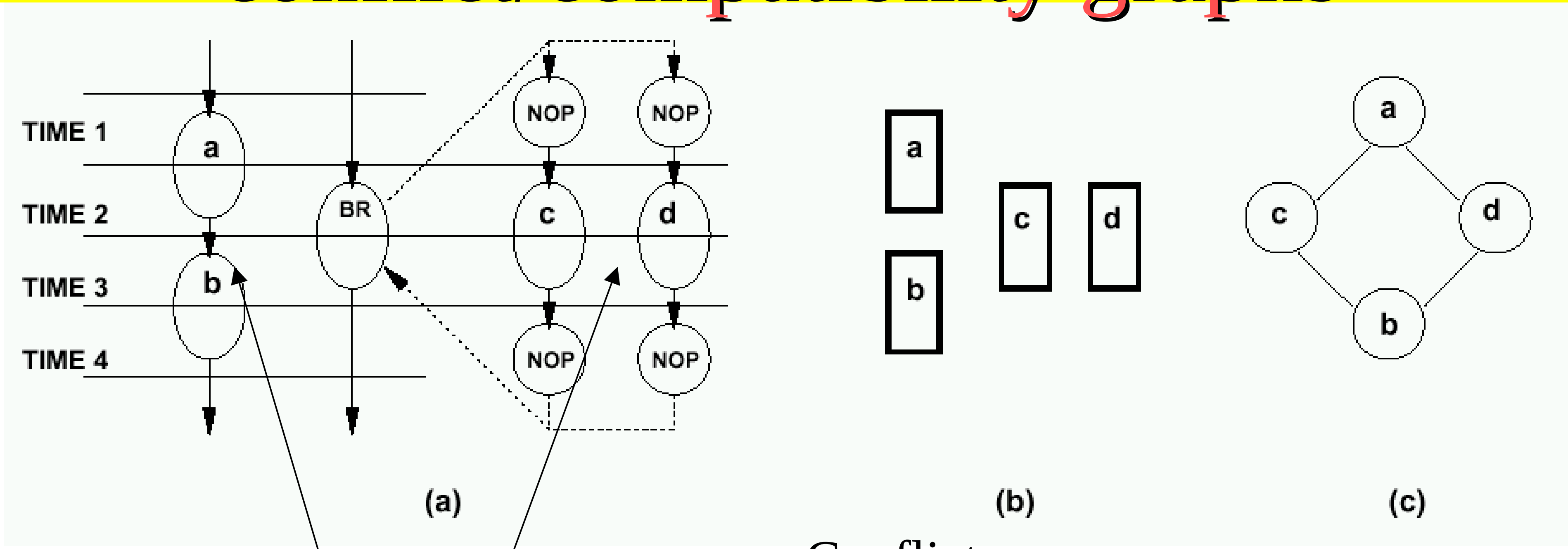


Conflict graph



Compatibility graph

Example of Hierarchical conflict/compatibility graphs



Hierarchical sequencing graphs

Conflict graph

Compatibility graph

c and d are not compatible because executed in parallel

a and b are not compatible

Register binding problem

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

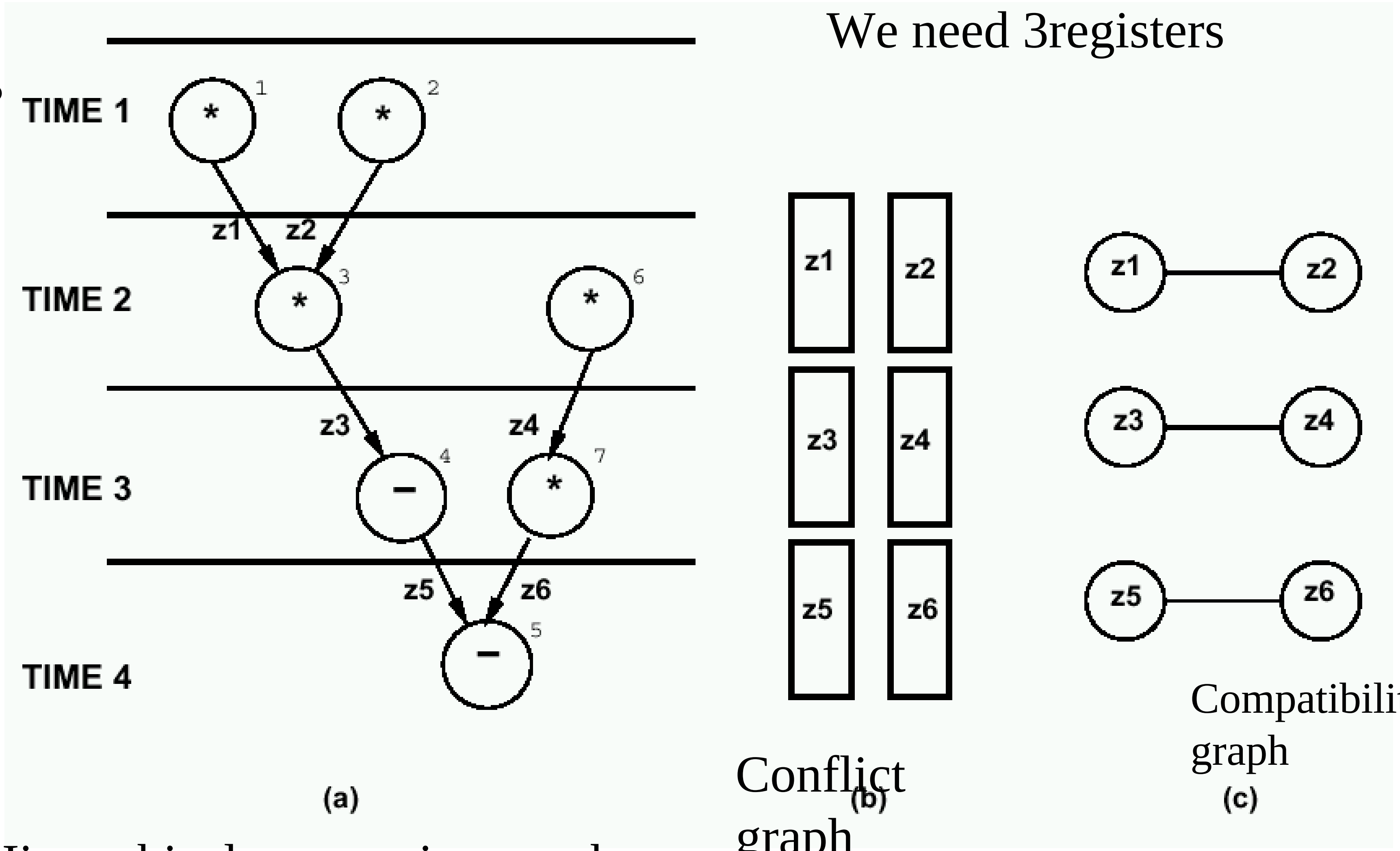
Register sharing data-flow graphs

- **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

We need 3registers

•



Hierarchical sequencing graphs

Conflict graph

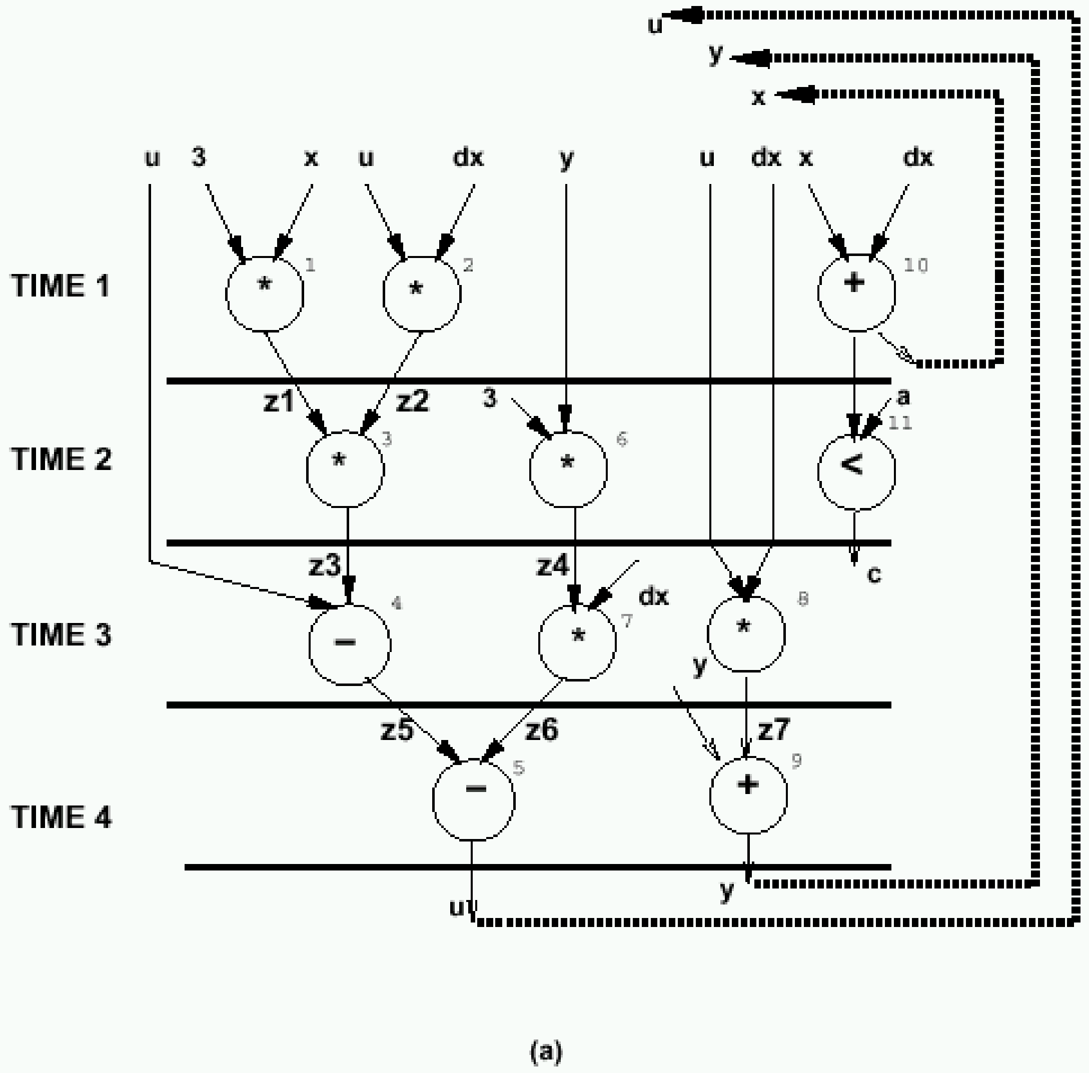
Compatibility graph

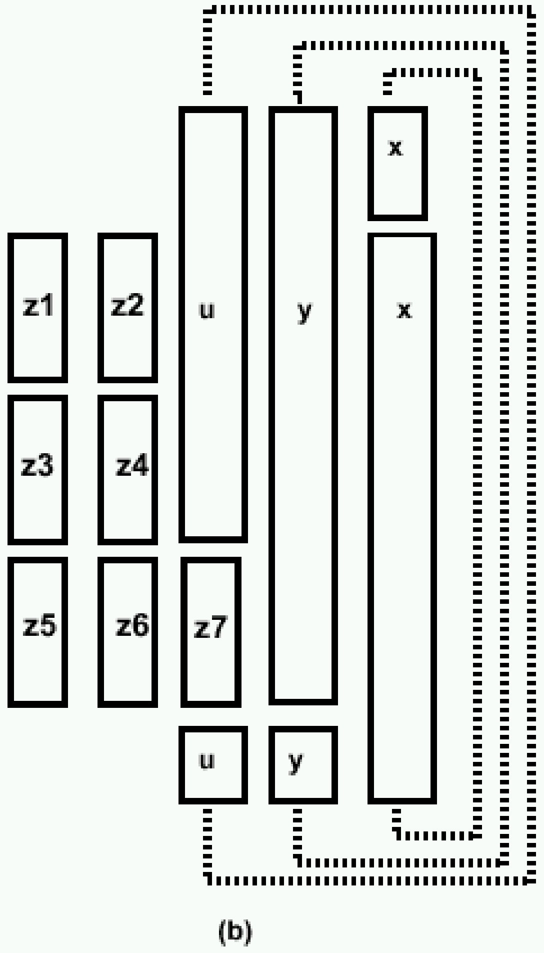
Register sharing general case

- 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

Conflict graph

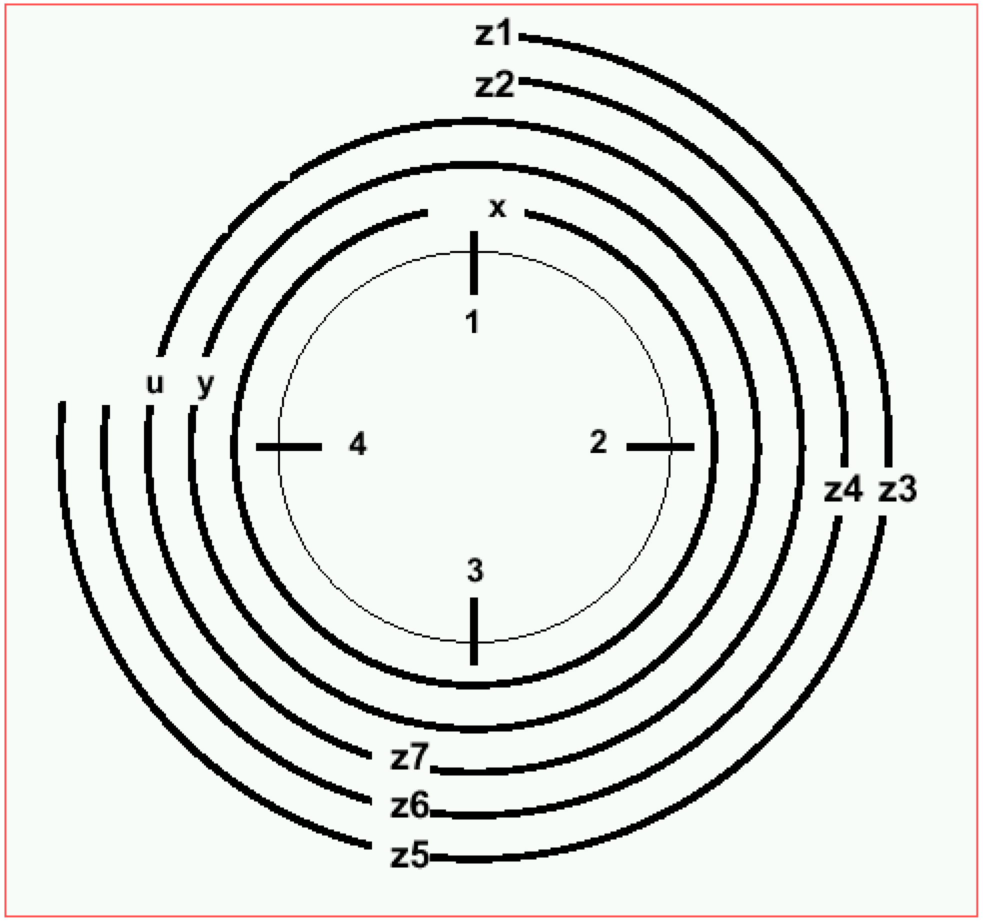




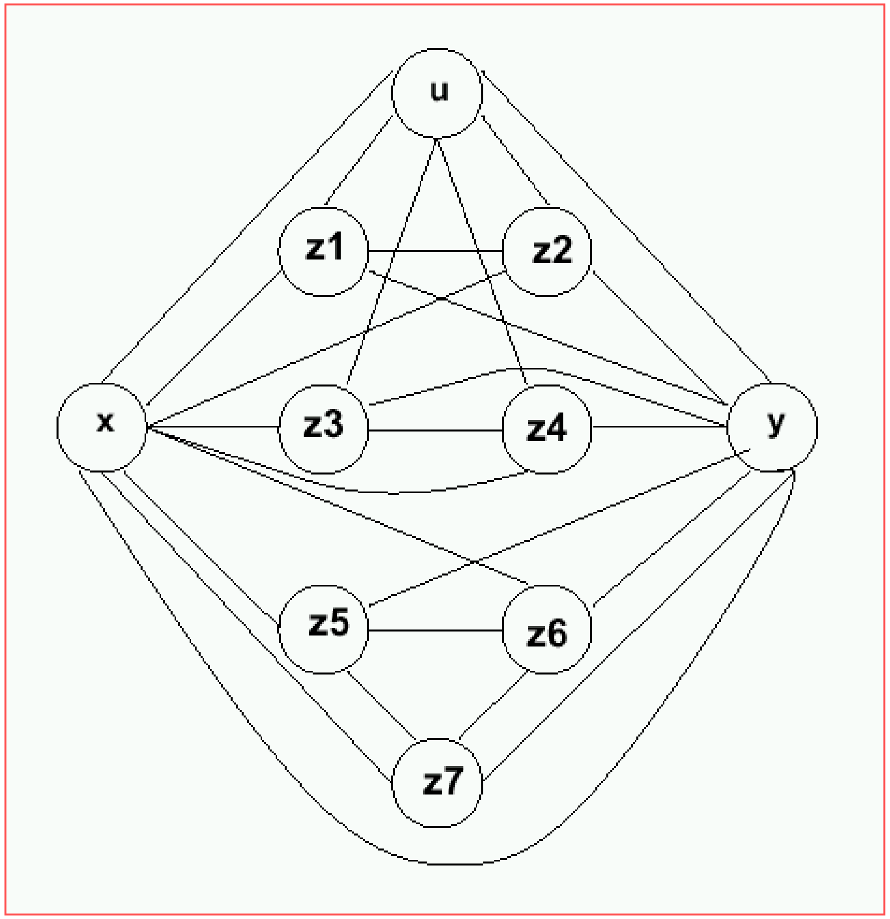
Hierarchical sequencing graphs

This leads to circular conflict graph

Example continued
Variable-lifetimes and circular-arc conflict graph



circular-arc
conflict graph



Compatibility graph

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:**

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

Multiport-memory binding

- Find ***maximum number of variables*** to be stored through a fixed number of ports **a**.
  - Boolean variables $\{b_i, i = 1, 2, \ldots, n_{var}\}$:

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

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

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

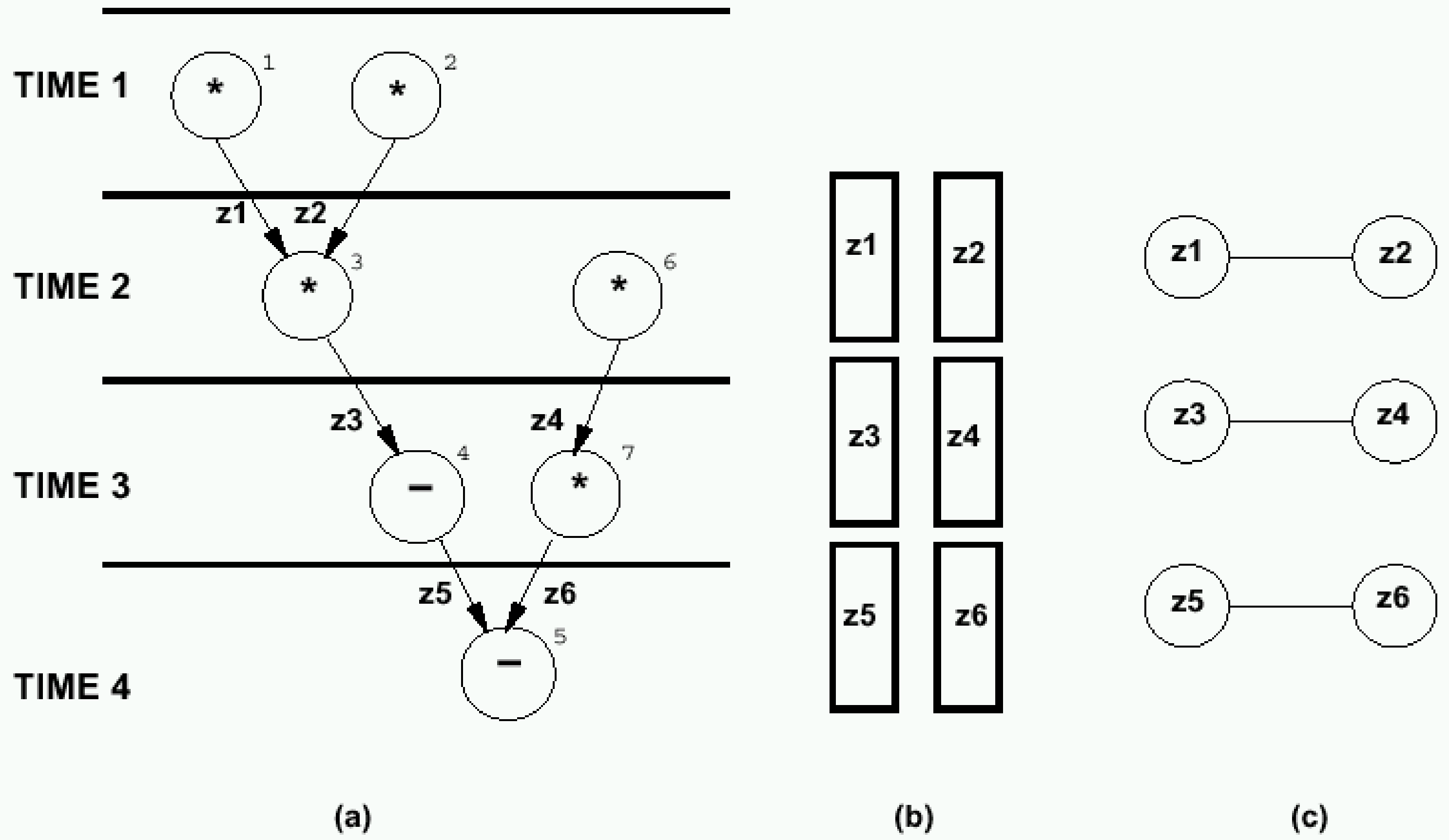
$$
\begin{array}{rcl}
b_1 + b_2 + b_3 + b_{12} & \leq & a \\
b_3 + b_4 + b_5 + b_6 + b_7 + b_{13} & \leq & a \\
b_1 + b_3 + b_5 + b_7 + b_8 + b_9 + b_{10} + b_{11} & \leq & a \\
b_8 + b_9 + b_{11} + b_{12} + b_{14} + b_{15} & \leq & a \\
b_1 + b_2 + b_{14} + b_{15} & \leq & a
\end{array}
$$

Example solution for Multiport-memory binding

- One port a = 1:
  - {$\mathbf{b}_2$, $\mathbf{b}_4$, $\mathbf{b}_8$} non-zero.
  - 3 variables stored: $\mathbf{v}_2$, $\mathbf{v}_4$, $\mathbf{v}_8$.
- Two ports a = 2:
  - 6 variables stored: $\mathbf{v}_2$, $\mathbf{v}_4$, $\mathbf{v}_5$, $\mathbf{v}_{10}$, $\mathbf{v}_{12}$, $\mathbf{v}_{14}$
- Three ports a = 3:
  - 9 variables stored: $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_4$, $\mathbf{v}_6$, $\mathbf{v}_8$, $\mathbf{v}_{10}$, $\mathbf{v}_{12}$, $\mathbf{v}_{13}$, $\mathbf{v}_{14}$

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.



- **One bus:**
  – 3 variables can be transferred.
- **Two busses:**
  – All variables can be transferred.

Example of Bus sharing and binding

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.

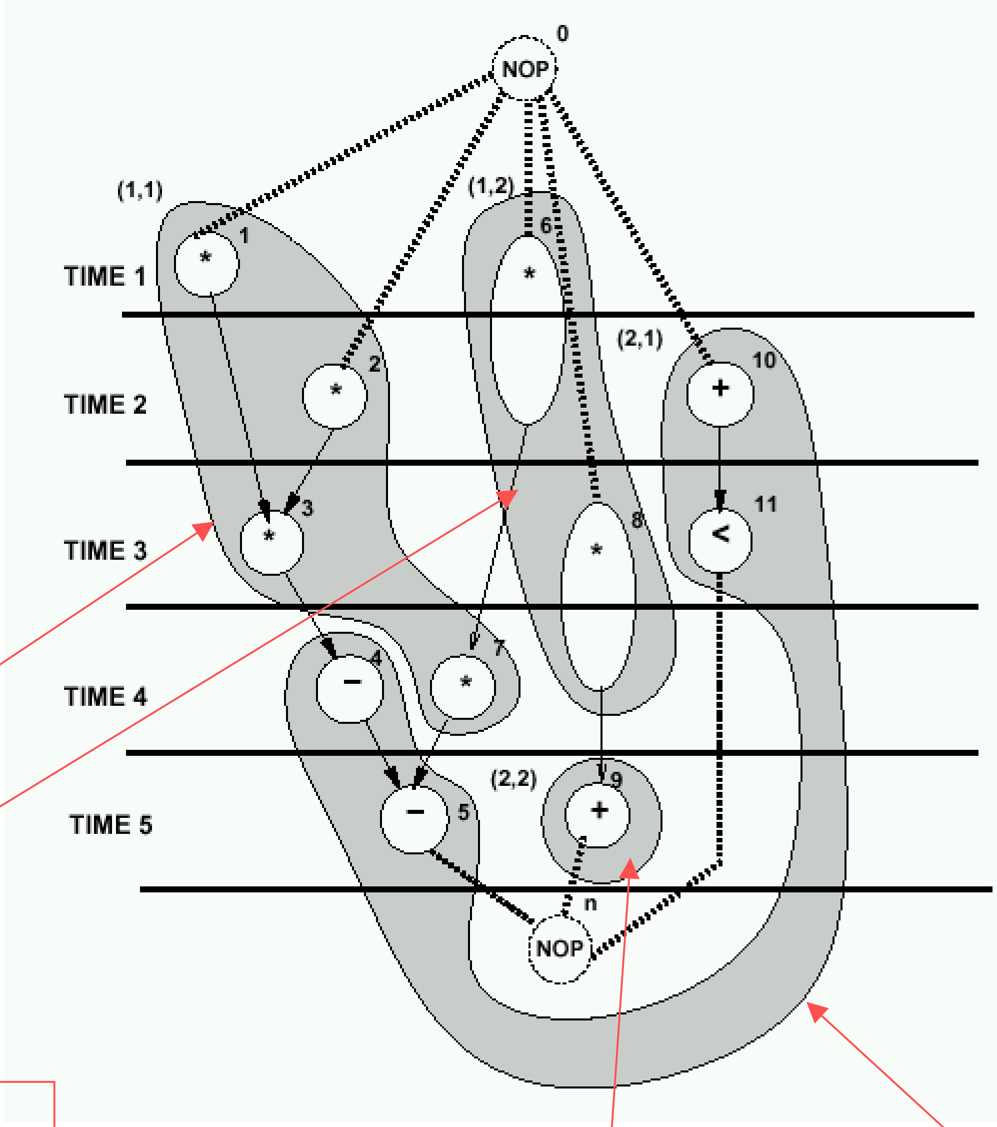
Module selection problem

- 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

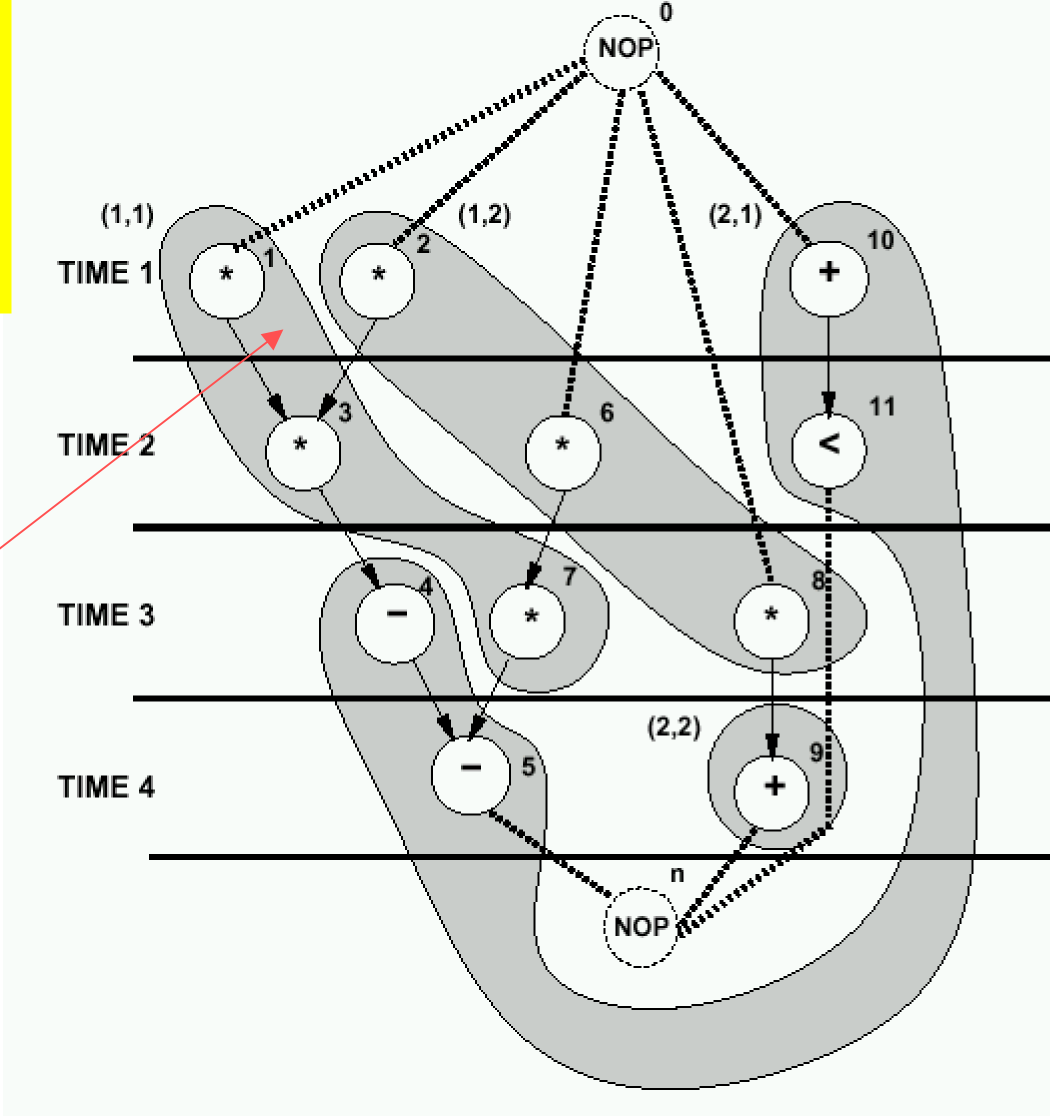
First ALU

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.