Embedded Systems

TMS320C6x Datapath

- ❖ **2 Data Paths**
- ❖ **8 Functional Units**
  - ◆ **Orthogonal/Independent**
  - ◆ **6 Arithmetic Units**
  - ◆ **2 Multipliers**
- ❖ **Control**
  - ◆ **Independent**
  - ◆ **Up to 8 32-bit Instructions in parallel**
- ❖ **Registers**
  - ◆ **2 Files**
  - ◆ **32, 32-bit Registers Total**
- ❖ **Cross paths (1X, 2X)**

- ❖ **L-Unit (L1, L2)**
  - ◆ **40-bit Integer ALU**
  - ◆ **Comparisons**
  - ◆ **Bit Counting**
  - ◆ **Normalization**
- ❖ **S-Unit (S1, S2)**
  - ◆ **32-bit ALU**
  - ◆ **40-bit Shifter**
  - ◆ **Bitfield Operations**
  - ◆ **Branching**
- ❖ **M-Unit (M1, M2)**
  - ◆ **16 x 16 -> 32**
- ❖ **D-Unit (D1, D2)**
  - ◆ **32-bit Add/Subtract**
 - ◆ **Address Calculations**

CS - ES
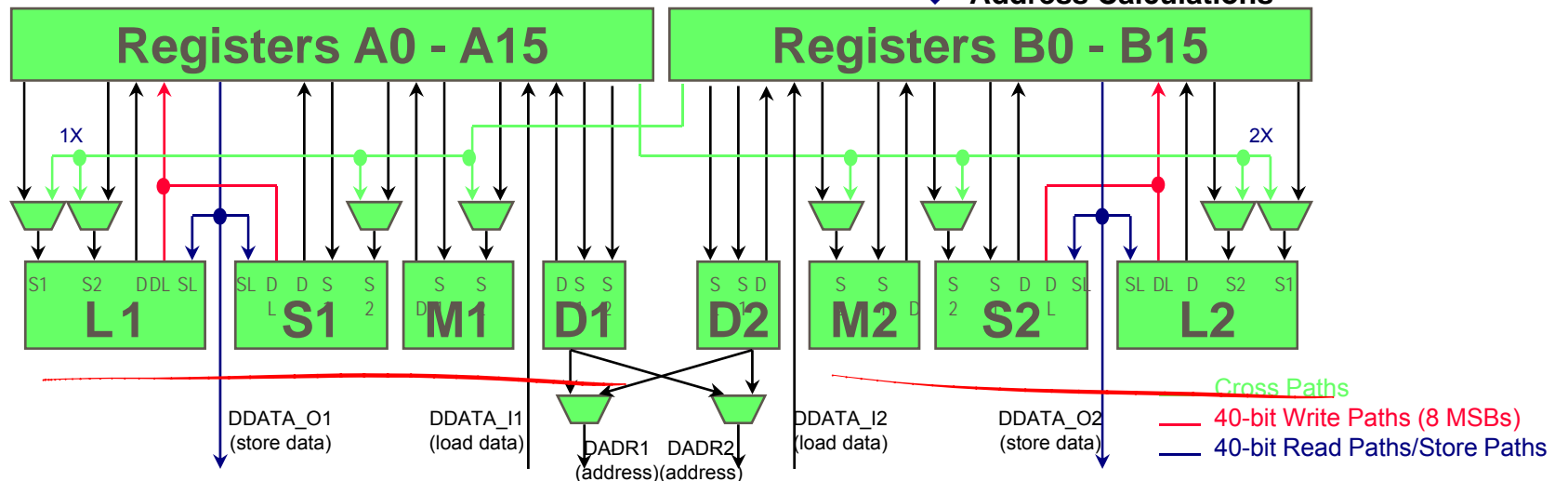
Overview XILINX FPGA

- All Xilinx FPGAs contain the same basic resources
  - Slices grouped into Configurable Logic Blocks (CLBs)
    - Contain combinatorial logic and register resources

  - IOBs
    - Interface between the FPGA and the outside world

  - Programmable interconnect

  - Other resources
    - Memory
    - Multipliers
    - Global clock buffers
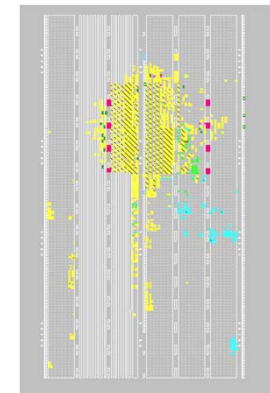   - Boundary scan logic

Embedded Processors in FPGAs

- **Hard Core**
  - EP is a dedicated physical component of the chip separate from the programmable logic
  - E.g. Xilinx Virtex families (PowerPC 405)
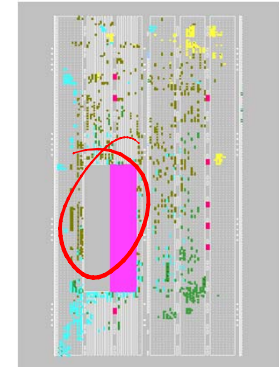
- **Soft Core**
  - Embedded processor is also a synthesized to the FPGA to th programmable logic on the chip
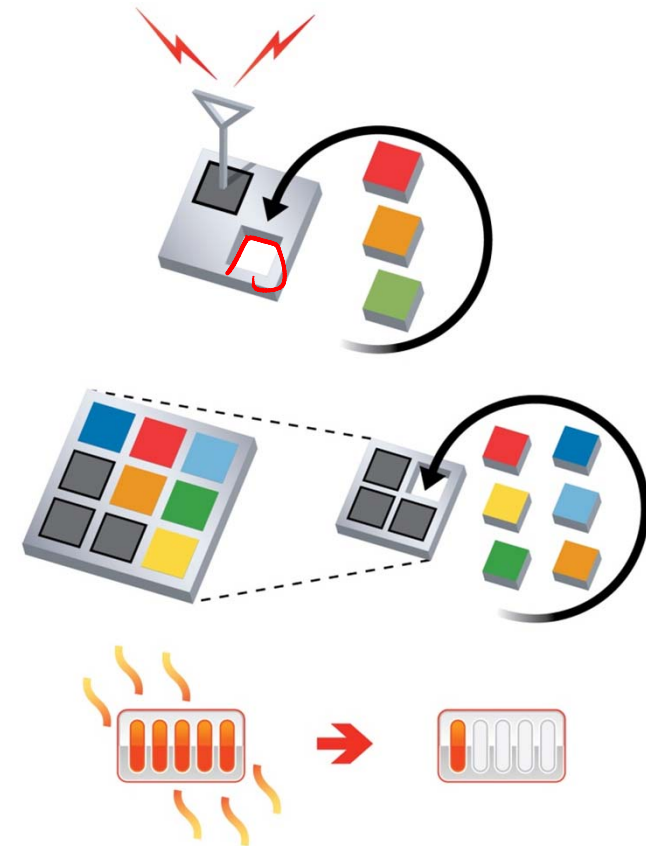 - E.g. Altera (NIOS), Xilinx (MicroBlaze)

Partial Reconfiguration
*Technology and Benefits*

- Partial Reconfiguration enables:
  - System Flexibility
    - Perform more functions while maintaining communication links

  - Size and Cost Reduction
    - Time-multiplex the hardware to require a smaller FPGA

  - Power Reduction
    - Shut down power-hungry tasks when not needed

CS - ES

Embedded System Hardware

- Embedded system hardware is frequently used in a loop (*"hardware in a loop"*):
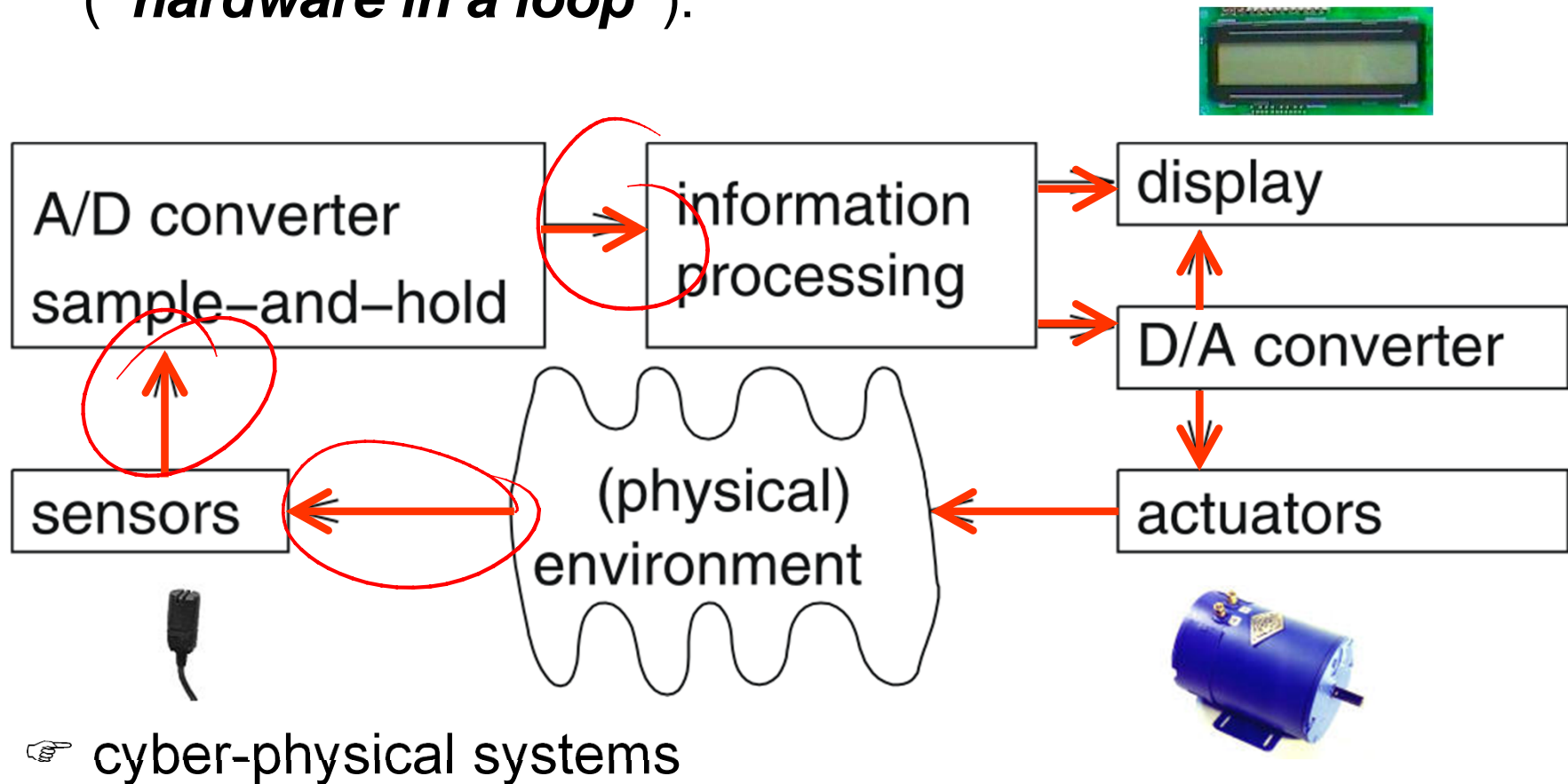
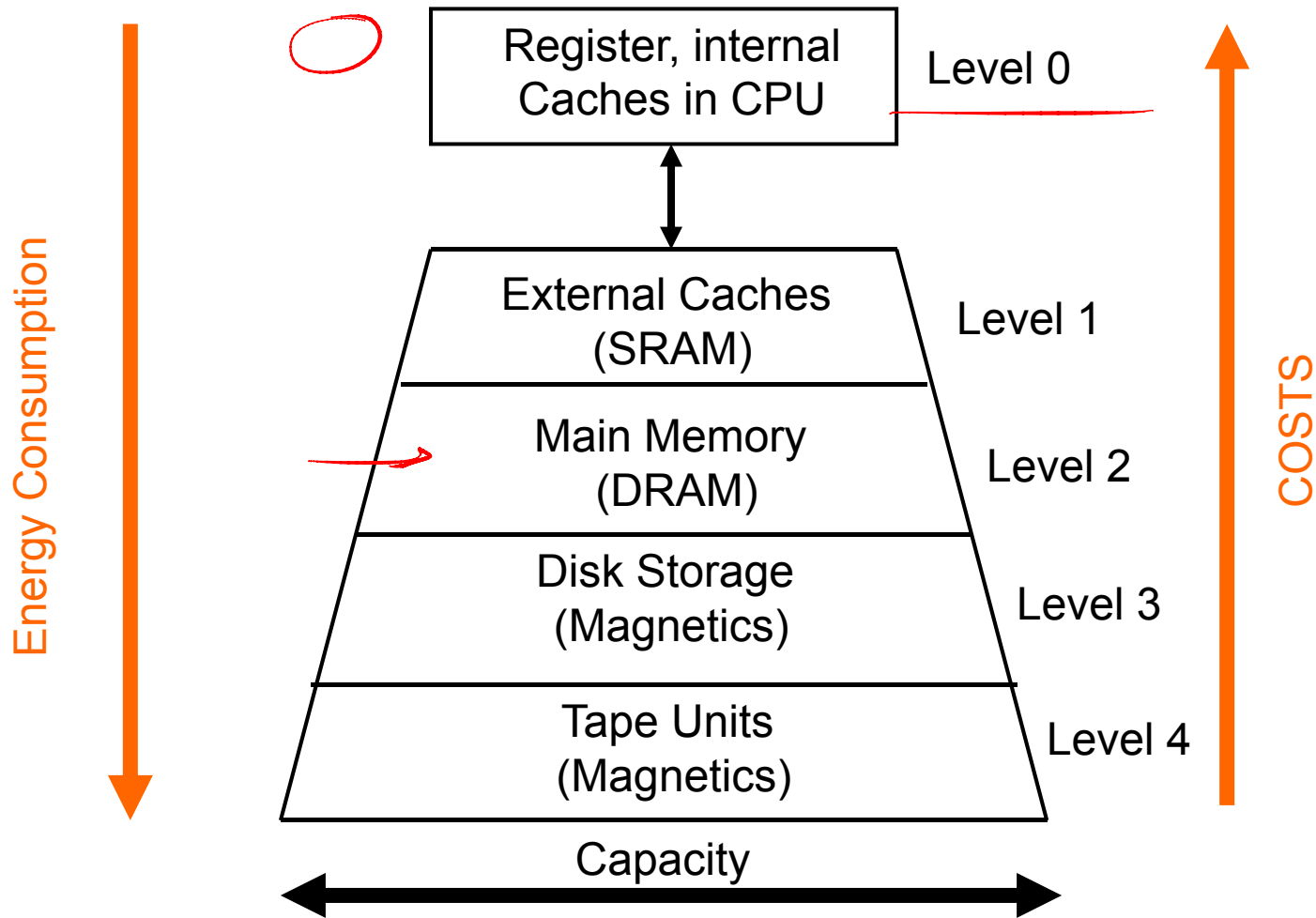☞ cyber-physical systems

Communication
- Requirements -

- Real-time behavior

- Efficient, economical
(e.g. centralized power supply)

- Appropriate bandwidth and communication delay

- Robustness

- Fault tolerance

- Maintainability

- Diagnosability

- Security

- Safety

Memory

- For the memory, efficiency is again a concern:
  - speed (latency and throughput); predictable timing
  - energy efficiency
  - size
  - cost
  - other attributes (volatile vs. persistent, etc)

Memory hierarchy

Energy Consumption

COSTS

Register, internal Caches in CPU	Level 0
External Caches (SRAM)	Level 1
Main Memory (DRAM)	Level 2
Disk Storage (Magnetics)	Level 3
Tape Units (Magnetics)	Level 4

Capacity

"Small is beautiful"
(in terms of energy consumption, access times, size)

Static Timing Analysis

producing the input to schedulability analysis

REVIEW

Schedulability analysis has assumed the knowledge of the execution time of tasks.
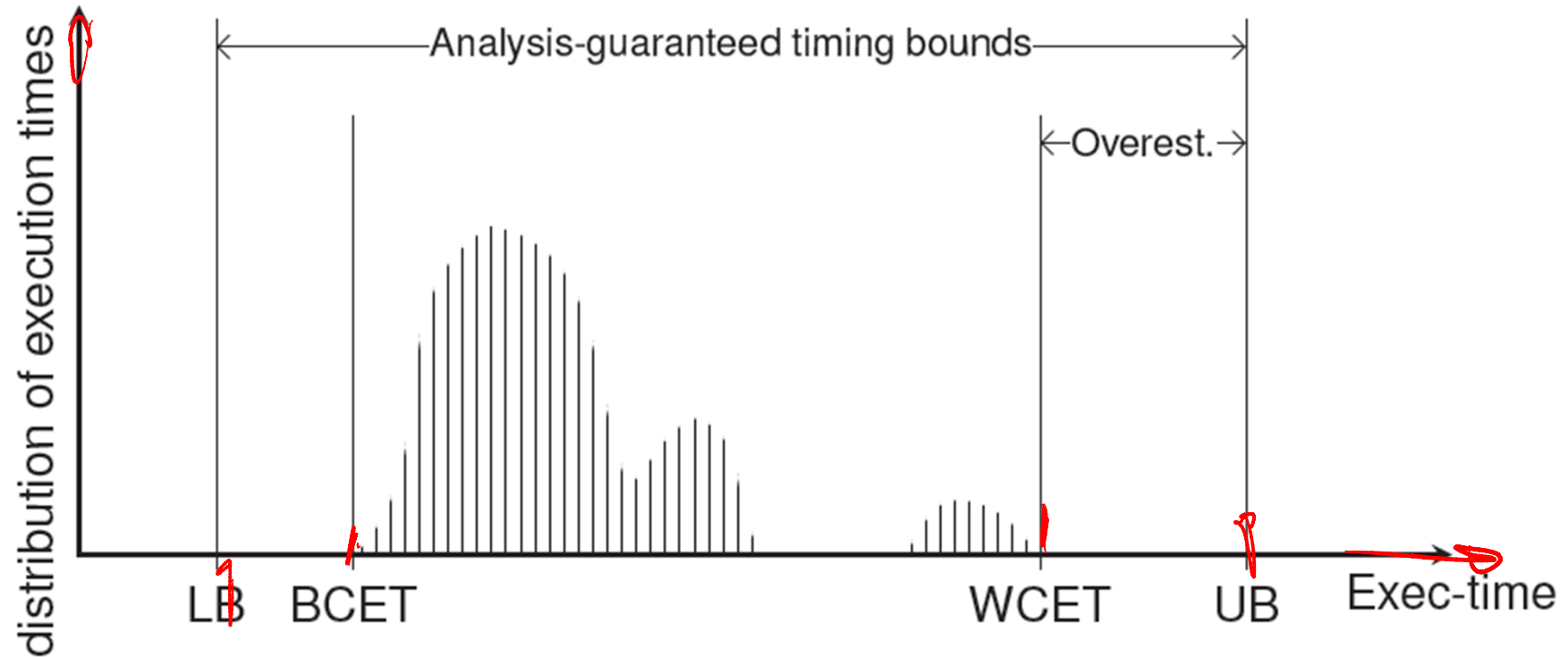
So, the problem to solve:

- Given
    1. a software task to produce some reaction,
    2. a hardware platform, on which to execute the software,
    3. a required reaction time, e.g. the period of the task.
- Derive:
    - ► a reliable (and precise) upper bound on the execution times.

Timing Analysis

REVIEW

- ▪ Architecture Synthesis

- ▪ HW/SW Codesign

- ▪ Power Aware Computing

- ▪ 3.2.2011 Lecture by Bernd Finkbeiner, Head of Reactive Systems Group at Saarland University(http://react.cs.uni-sb.de/

Architecture Synthesis

Design a hardware architecture that efficiently executes a given algorithm.

- **tasks**:
    - **allocation** (determine the necessary hardware resources)
    - **scheduling** (determine the timing of individual operations)
    - **binding** (determine relation between individual operations of the algorithm and hardware resources)
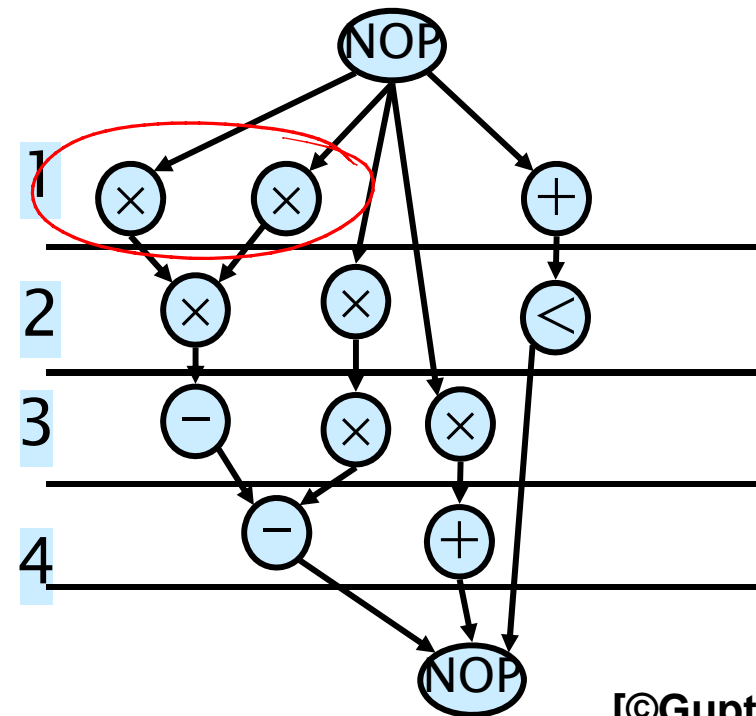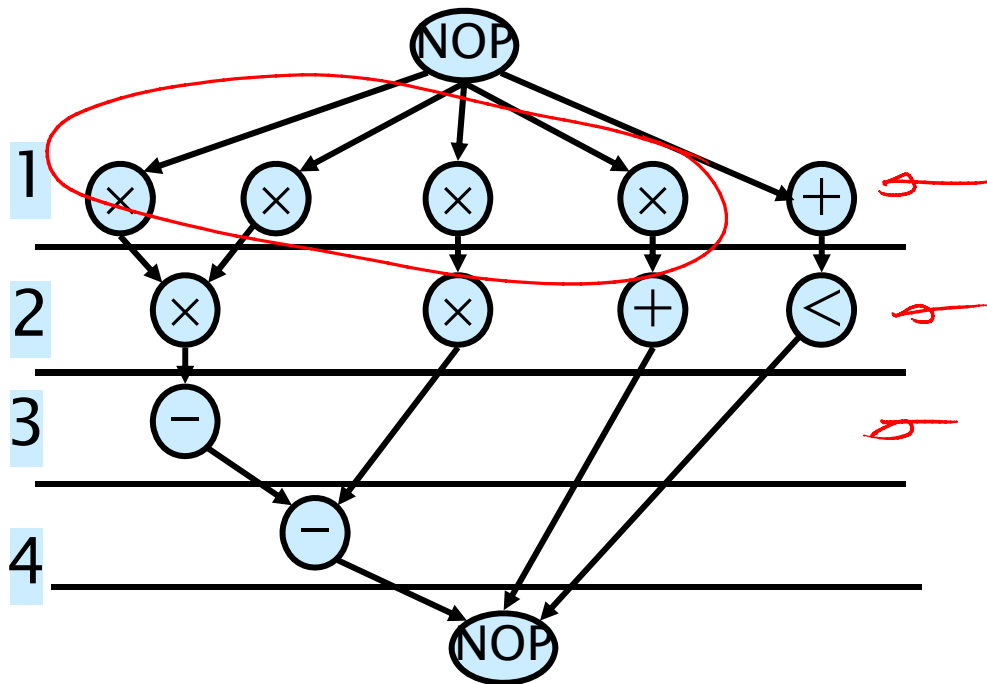
Classification of synthesis algorithms →

- Synthesis methods can often be applied **independently of granularity**
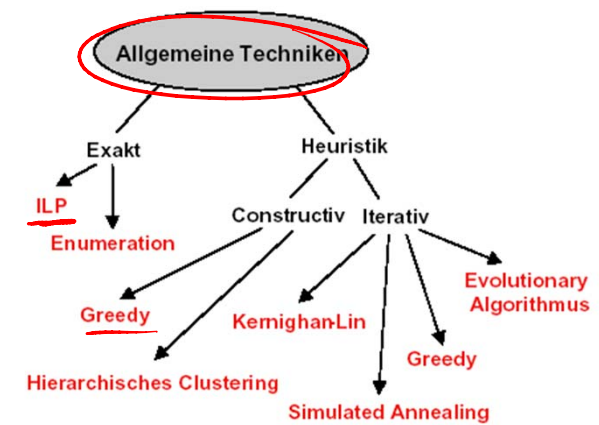
Synthesis in Temporal Domain

- Scheduling and binding can be done in different orders or together
- Schedule:
  - Mapping of operations to time slots (cycles)
  - A scheduled sequencing graph is a labeled graph

Schedule in Spatial Domain

- Resource sharing
  - More than one operation bound to same resource
  - Serialized operations

[©Gupta]
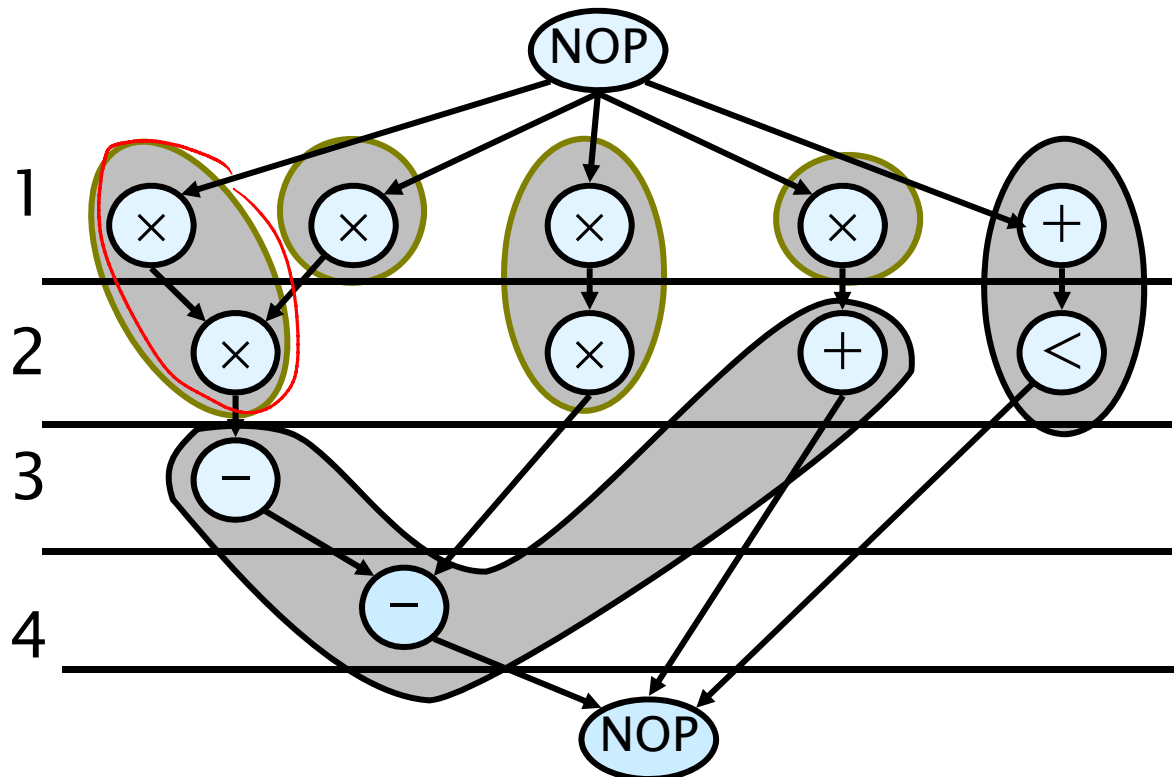
BASICS

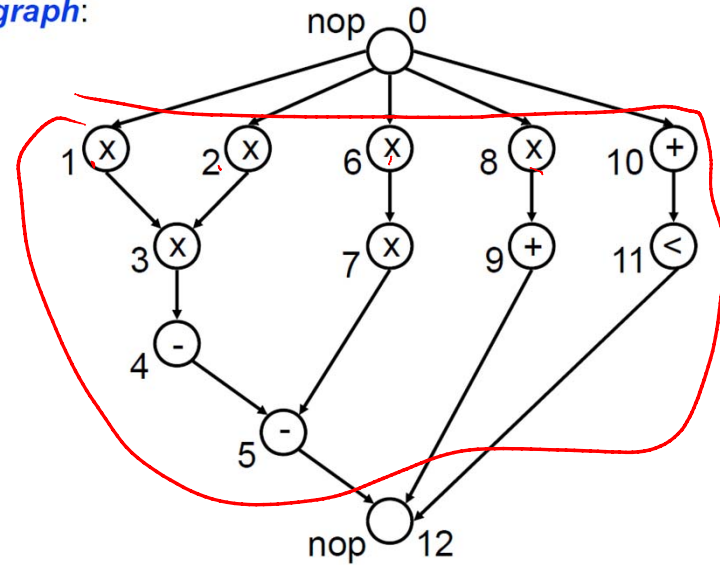- Source: Teich: Dig. HW/SW Systeme;Thiele ETHZ

Models

- **Sequence graph** $G_S = (V_S, E_S)$
  where V_S denotes the operations of the algorithm and E_S the dependence relations.

- **Resource graph** $G_R = (V_R, E_R)$, $V_R = V_S \cup V_T$
  where V_T denote the resource types of the architecture and G_R is a bipartite graph. An edge $(v_s, v_t) \in E_R$ represents the availability of a resource type v_t for an operation v_s.

- **Cost function** $c : V_T \to \mathbf{Z}$

- **Execution times** $w : E_R \to \mathbf{Z}^{\geq 0}$
  are assigned to each edge $(v_s, v_t) \in E_R$ and denote the execution time of operation $v_s \in V_S$ on resource type $v_t \in V_T$.

Models
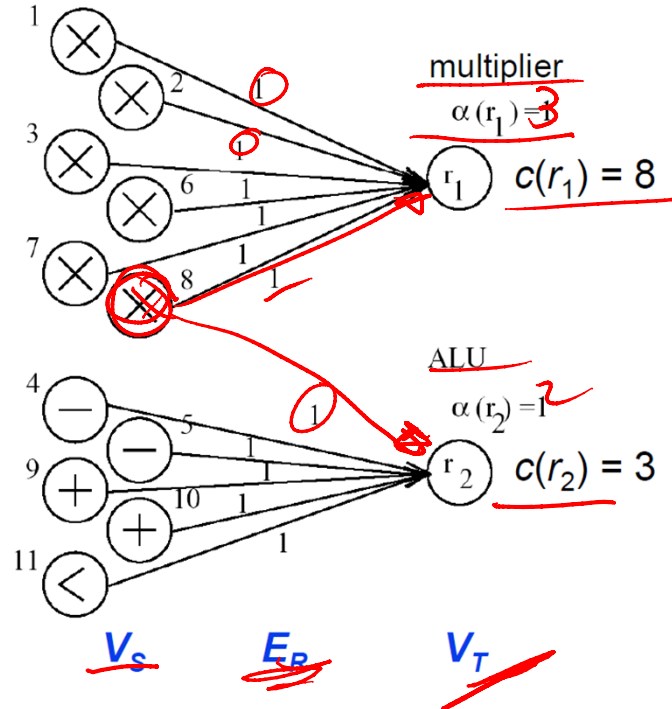
```
int diffeq(int x, int y, int u, int dx, int a)
       { int x1, u1, y1;
       while ( x < a ) {
                      x1 = x + dx;
                      u1 = u - (3 * x * u * dx) - (3 * y * dx);
                      y1 = y + u * dx;
                      x = x1; u = u1; y = y1;
       }
       return y;
```
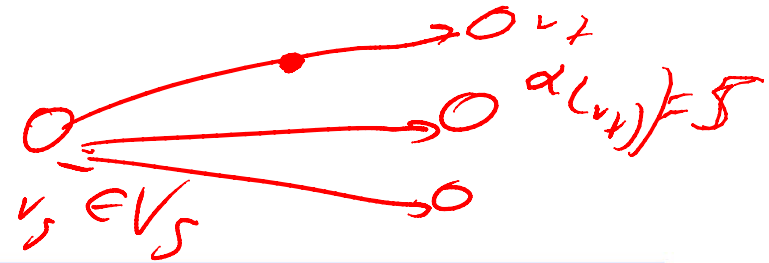

Resource graph:

$\alpha(r_1) = 3$

$c(r_1) = 8$

$\alpha(r_2) = 1$

$c(r_2) = 3$

V_S E_R V_T

CS - ES

Allocation and Binding

An allocation is a function $\alpha : V_T \to \mathbf{Z}^{\geq 0}$ that assigns to each resource type $v_t \in V_T$ the number $\alpha(v_t)$ of available instances.

A binding is defined by functions $\beta : V_S \to V_T$ and $\gamma : V_S \to \mathbf{Z}^{>0}$. Here, $\beta(v_s) = v_t$ and $\gamma(v_s) = r$ denote that operation $v_s \in V_S$ is implemented on the rth instance of resource type $v_t \in V_T$.

Scheduling

A schedule is a function $\tau : V_S \rightarrow \mathbf{Z}^{>0}$ that determines the starting times of operations. A schedule is feasible if the conditions

$$\tau(v_j) - \tau(v_i) \geq w(v_i) \quad \forall (v_i, v_j) \in E_S$$

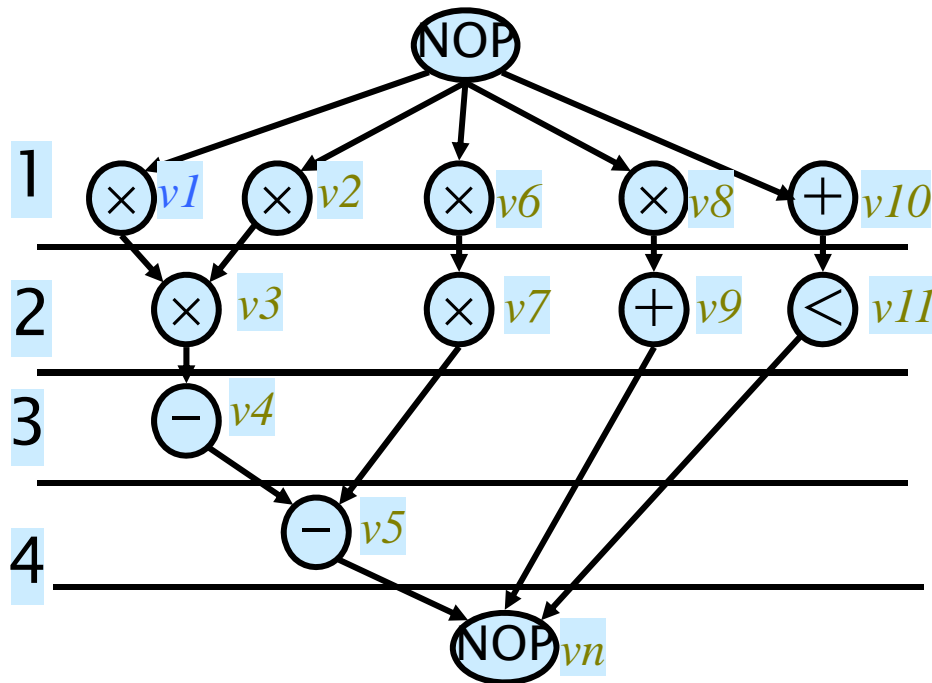are satisfied. $w(v_i) = w(v_i, \beta(v_i))$ denotes the execution time of operation v_i.
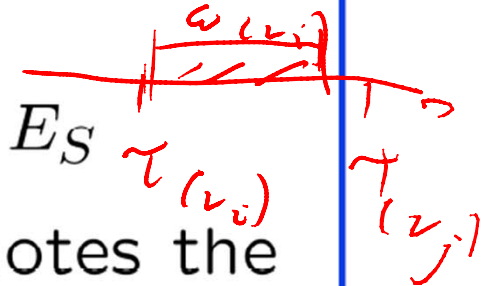
The latency L of a schedule is the time difference between start node v_0 and end node v_n:
$$L = \tau(v_n) - \tau(v_0) .$$

Schedule

$$L = \tau(v_n) - \tau(v_0) = 4$$

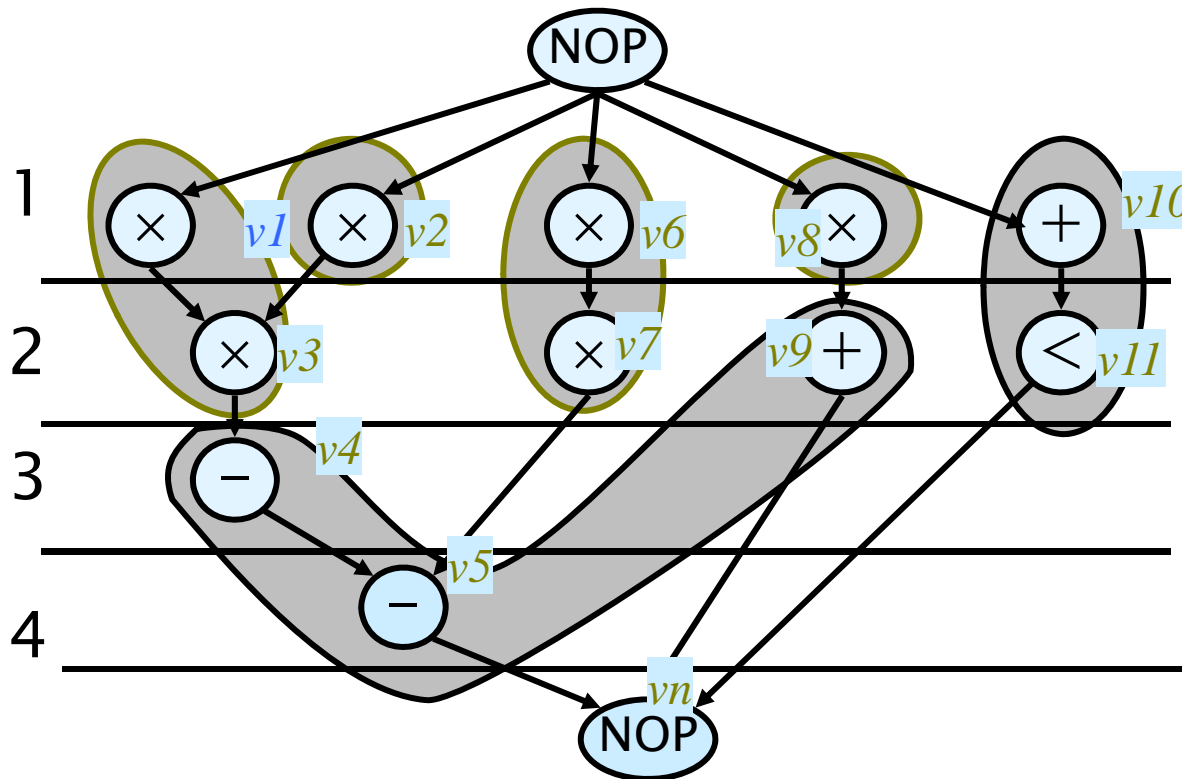$\tau(v_0) = 1$

$\tau(v_1) = \tau(v_2) \ldots = 1$

$\ldots$

$\tau(v_5) = 4$

$\tau(v_n) = 5$

Binding

Example ($\alpha(r_1) = 4$, $\alpha(r_2) = 2$):

$\beta(v_1) = r1$, $\gamma(v_1) = 1$

$\beta(v_2) = r1$, $\gamma(v_2) = 2$

$\beta(v_3) = r1$, $\gamma(v_3) = 2$

...

$\beta(v_6) = r1$, $\gamma(v_3) = 3$

...

As soon as possible (ASAP) scheduling

- ASAP: All tasks are scheduled as early as possible

- Loop over (integer) time steps:

  - Compute the set of unscheduled tasks for which all predecessors have finished their computation

  - Schedule these tasks to start at the current time step.

ASAP Schedules

$\text{ASAP}(G_S(V_S, E_S), w) \; \{$

$\quad \tau(v_0) = 1;$

$\quad \text{REPEAT} \; \{$

$\quad\quad \text{Determine } v_i \text{ whose predec. are planed};$

$\quad\quad \tau(v_i) = \max\{\tau(v_j) + w(v_j) \; \forall (v_j, v_i) \in E_S\}$

$\quad \} \; \text{UNTIL} \; (v_n \text{ is planned});$

$\quad \text{RETURN} \; (\tau);$

$\}$
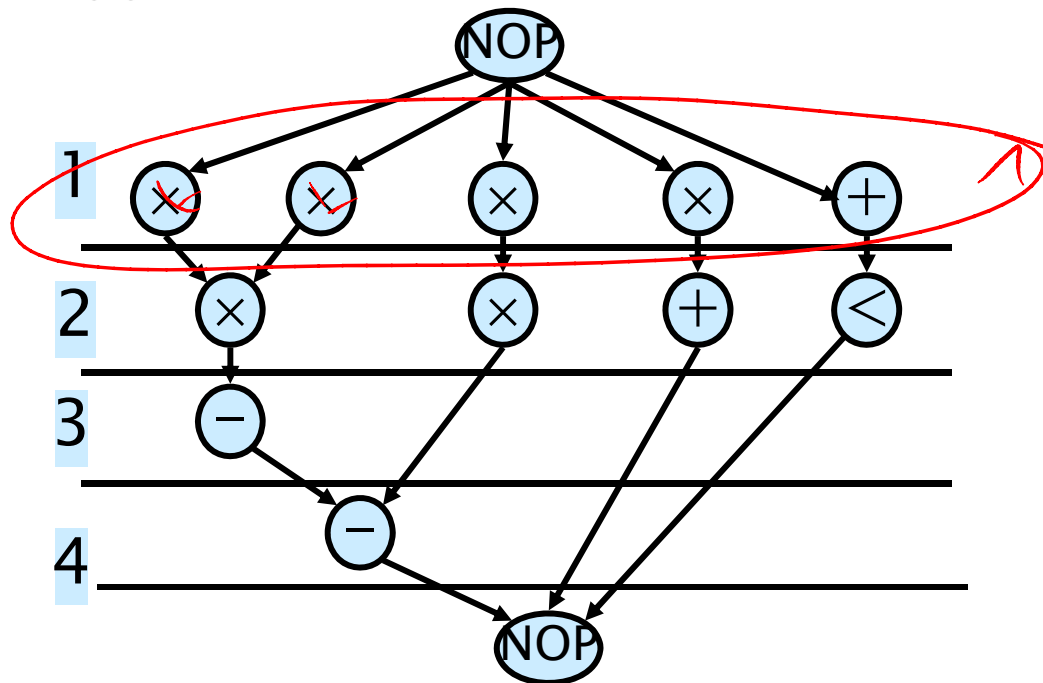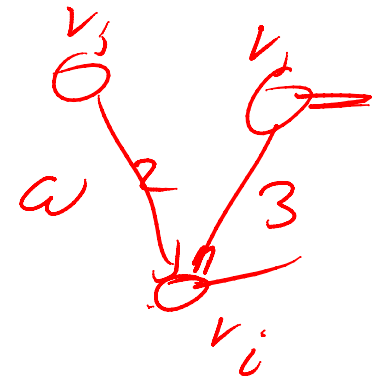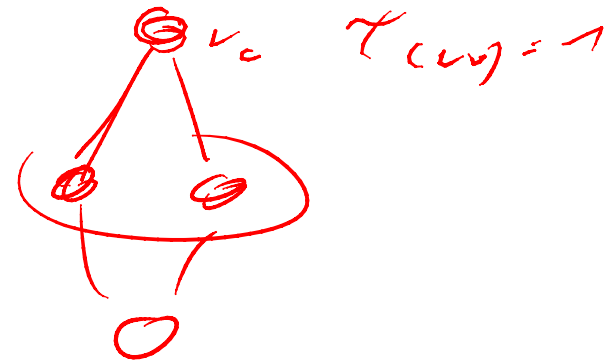
As-late-as-possible (ALAP) scheduling

▪ALAP: All tasks are scheduled as late as possible

Start at last time step*:

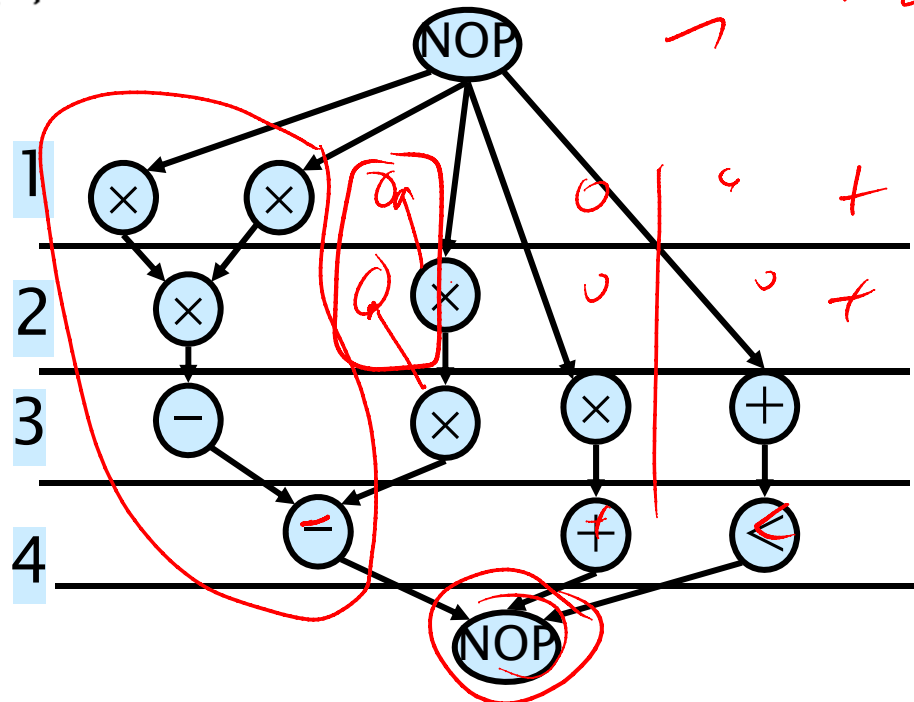Schedule tasks with no successors and tasks for which all successors have already been scheduled.

---

* Generate a list, starting at its end

ALAP Schedules

$$\text{ALAP}(G_S(V_S, E_S), w, L_{max}) \{$$
$$\tau(v_n) = L_{max} + 1;$$
$$\text{REPEAT} \{$$
$$\quad \text{Determine } v_i \text{ whose succ. are planed;}$$
$$\quad \tau(v_i) = \min\{\tau(v_j) \; \forall (v_i, v_j) \in E_S\} - w(v_i)$$
$$\} \text{ UNTIL } (v_0 \text{ is planned});$$
$$\text{RETURN } (\tau);$$
$$\}$$

$$L_{max} = \tau_{(v_n)} - \tau_{(v_0)}$$

$$\tau_{(v_i)}$$

$$\tau_{(v_j)}$$

# Scheduling under Detailed Timing Constraints

- Motivation
  - Interface design.
  - Control over operation start time.

- Constraints
  - Upper/lower bounds on start-time difference of any operation pair.

- Minimum timing constraints between two operations
  - An operation follows another by *at least* a number of prescribed time steps

- Maximum timing constraints between two operations
  - An operation follows another by *at most* a number of prescribed time steps

CS - ES

Scheduling under Detailed Timing Constraints

- Example

    - Circuit reads data from a bus, performs computation, writes result back on the bus.

    - Bus interface constraint: data written three cycles after read.

    - Minimum and maximum constraint of 3 cycles between read and write operations.

Constraint graph model

- Start from a sequencing graph

- Model delays as weights on edges

- Add forward edges for minimum constraints

- Add backward edges for maximum constraints

Add this edge for min constraint

MAX TIME 3

MIN TIME 4

Add this edge for max constraint
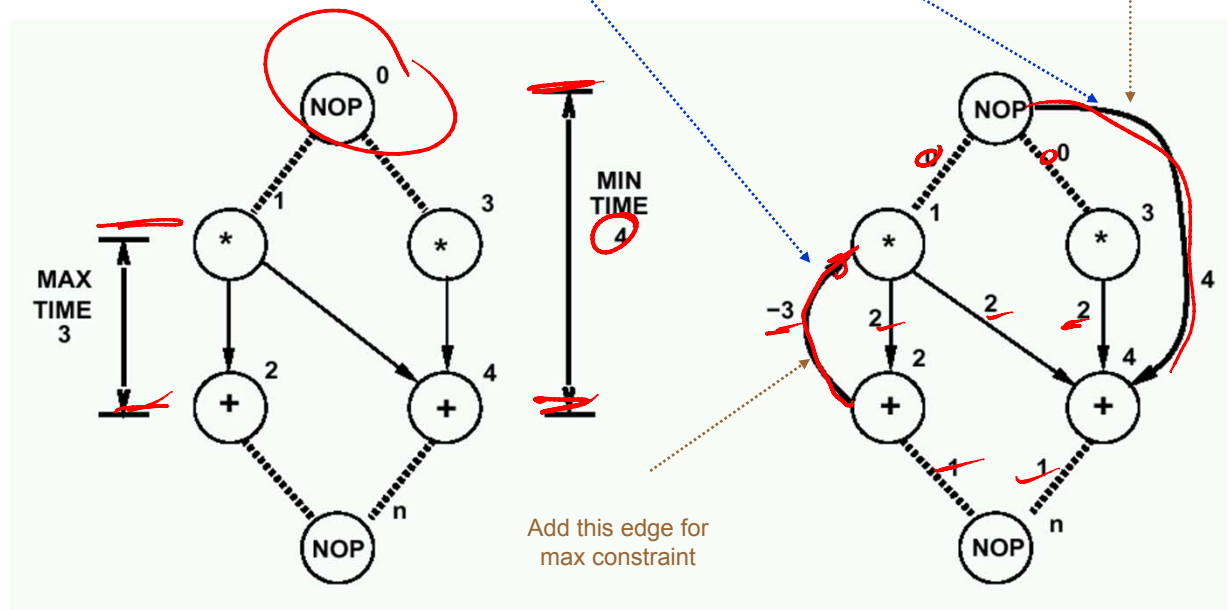
Weighted Constraint Graph

► In order to represent a *feasible schedule*, we have one edge corresponding to each precedence constraint with

$$d(v_i, v_j) = w(v_i)$$

where $w(v_i)$ denotes the execution time of v_i.

► A consistent assignment of starting times $\tau(v_i)$ to all operations can be done by solving a *single source longest path* problem.

► A possible algorithm (*Bellman-Ford*) has complexity $O(|V_c| \, |E_c|)$:

Iteratively set $\tau(v_j) := \max\{\tau(v_j), \tau(v_i) + d(v_i, v_j) : (v_i, v_j) \in E_C\}$ for all $v_j \in V_C$ starting from $\tau(v_i) = -\infty$ for $v_i \in V_C \backslash \{v_0\}$ and $\tau(v_0) = 1$.
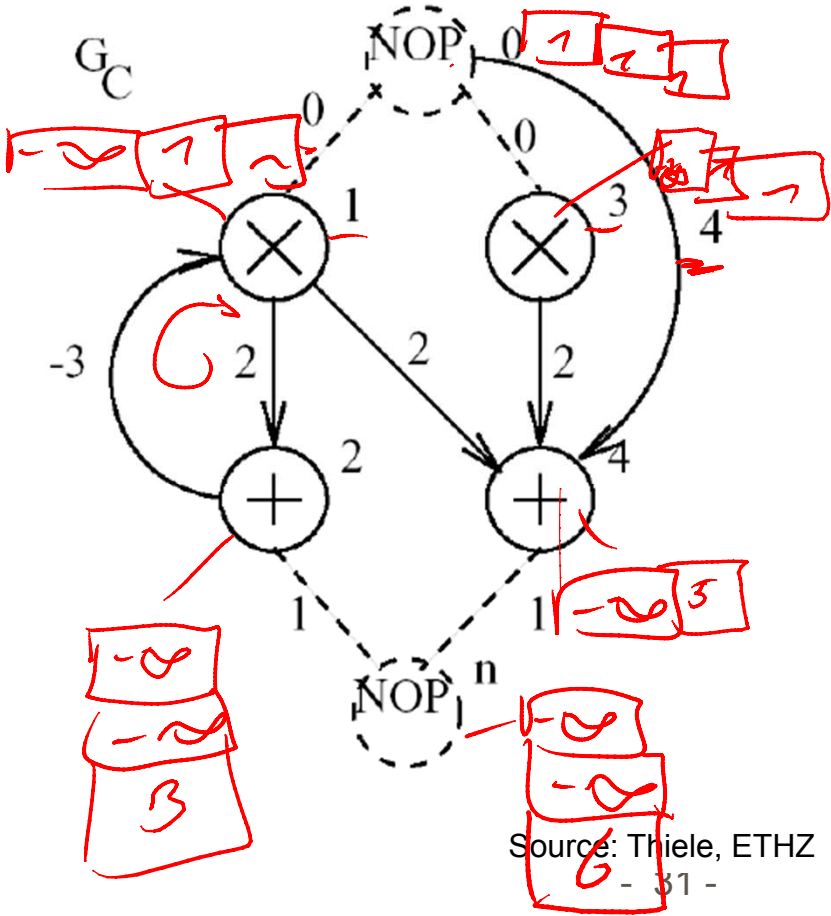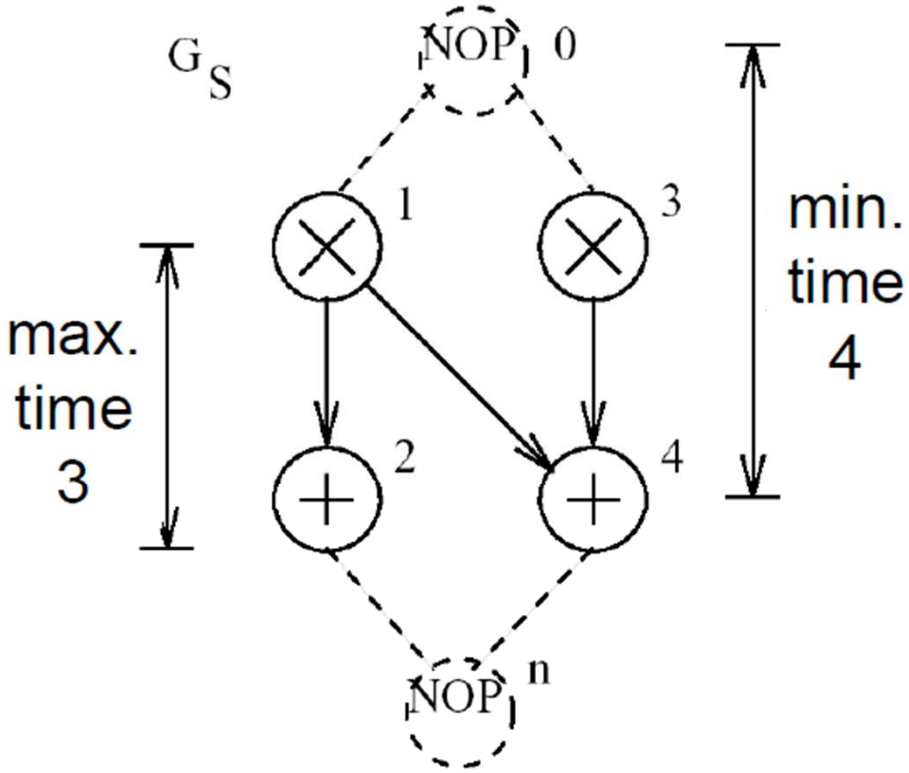
CS - ES

Weighted Constraint Graph

Example: $w(v1) = w(v3) = 2$ $w(v2) = w(v4) = 1$

$$\tau(v_j) := \max\{\tau(v_j), \tau(v_i)+d(v_i, v_j)\}$$

CS - ES

Solution - Constraint Graph Model

Mul delay = 2
ADD delay =1

| Vertex	Start time
v_0	1
v_1	1
v_2	3
v_3	1
v_4	5
v_n	6

(Resource constrained) List Scheduling

▪List scheduling: extension of ALAP/ASAP method

▪Preparation:

- Greedy strategy (does NOT guarantee optimum solution)

- Topological sort of task graph $G=(V,E)$

- Computation of priority of each task:

  Possible priorities u:

  - Number of successors

  - Longest path

  - **Mobility** = τ (ALAP schedule)- τ (ASAP schedule)
    - Defined for each operation
    - Zero mobility implies that an operation can be started only at one given time step
    - Mobility greater than 0 measures span of time interval in which an operation may start → Slack on the start time

CS - ES

- 33 -

Mobility as a priority function

Mobility is not very precise
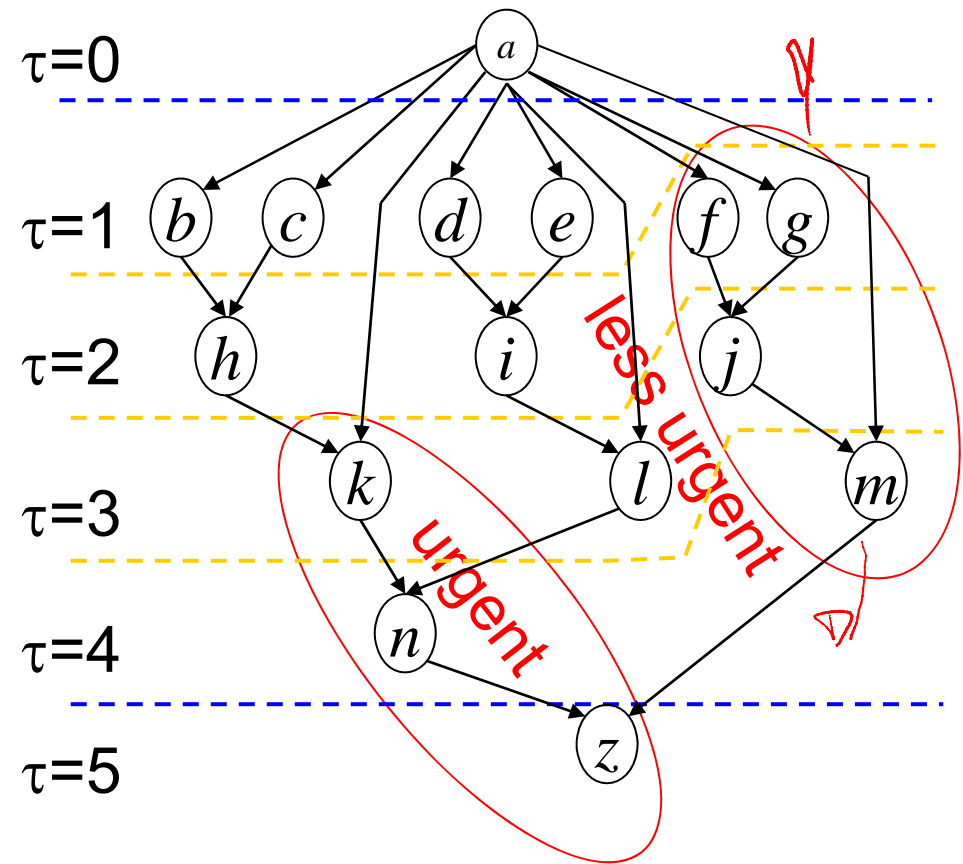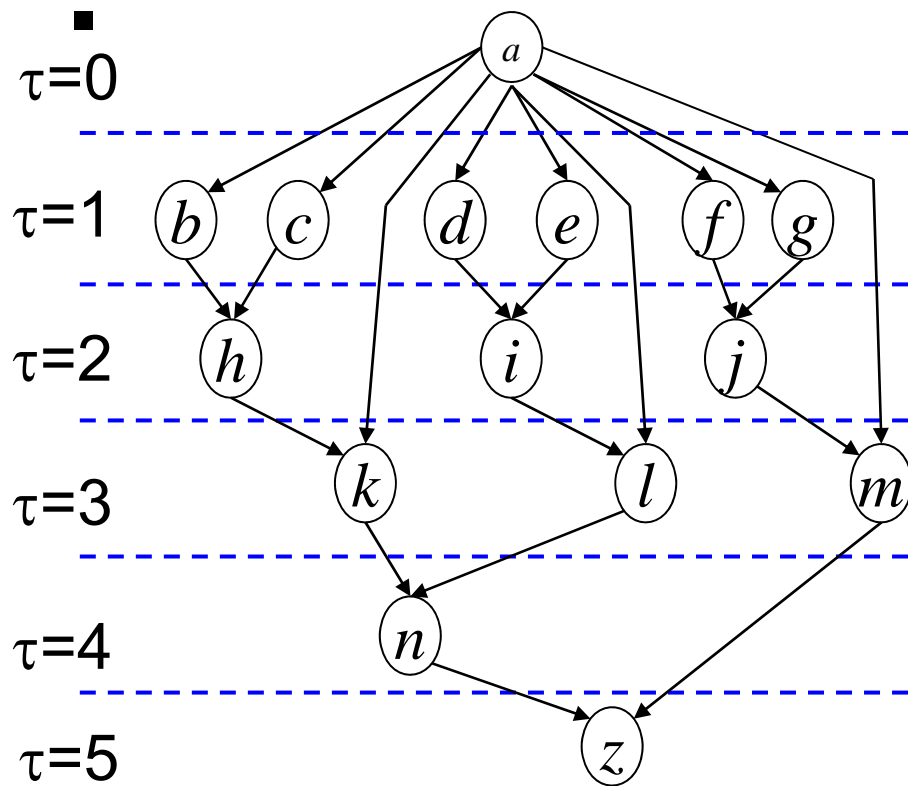
Algorithm

List($G(V,E)$, B, u){
$i :=0$;
  repeat {
    Compute set of candidate tasks A_i;
    Compute set of not terminated tasks G_i;
    Select $S_i \subseteq A_i$ of maximum priority r such that
    $| S_i | + | G_i | \leq B$ (***resource constraint***)

     foreach ($v_j \in S_i$): $\tau (v_j):=i$;   (*set start time*)
    $i := i +1$;
  }
  until (all nodes are scheduled);
  return (τ);
}

may be repeated for different task/ processor classes

Complexity: $O(|V|)$

Example

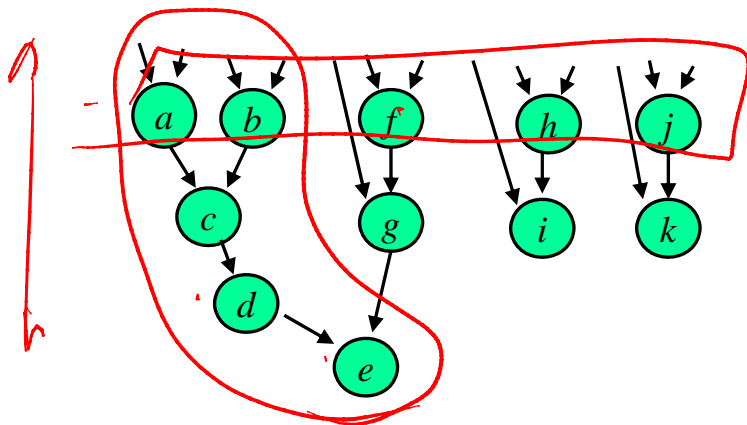■Assuming $B = 2$, unit execution time and u : path length

$u(a) = u(b) = 4$
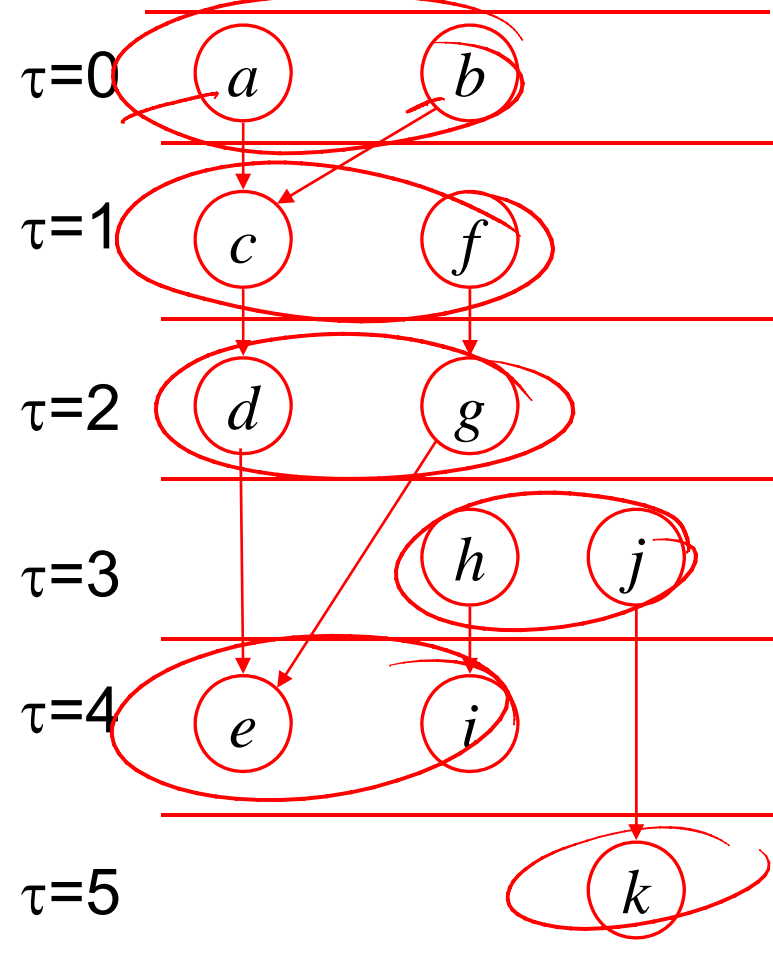$u(c) = u(f) = 3$
$u(d) = u(g) = u(h) = u(j) = 2$
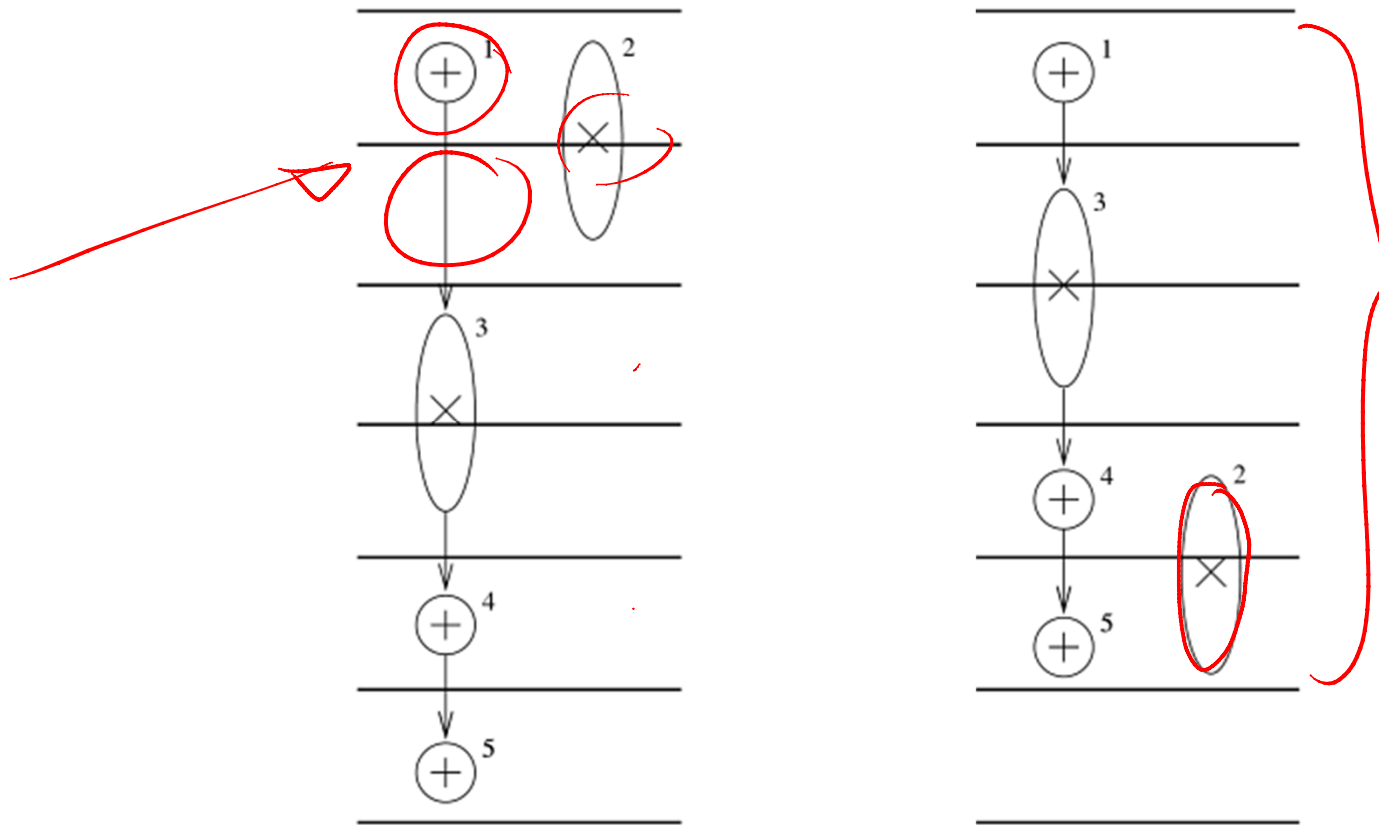$u(e) = u(i) = u(k) = 1$
$\forall\ i\ :\ G_i = 0$

$\tau = 0$ a   b

$\tau = 1$ c   f

$\tau = 2$ d   g

$\tau = 3$ h   j

$\tau = 4$ e   i

$\tau = 5$ k

does NOT guarantee optimum solution e.g.

List Scheduling

Integer linear programming models

- Ingredients:
  - Cost function
  - Constraints

  Involving linear expressions of integer variables from a set X

Cost function
$$C = \sum_{x_i \in X} a_i x_i \text{ with } a_i \in R, x_i \in \mathbb{N} \quad (1)$$

Constraints: $\forall j \in J : \sum_{x_i \in X} b_{i,j} x_i \geq c_j \text{ with } b_{i,j}, c_j \in \mathbb{R} \quad (2)$

Def.: The problem of minimizing (1) subject to the constraints (2) is called an **integer linear programming (ILP) problem**.

If all x_i are constrained to be either 0 or 1, the IP problem said to be a **0/1 integer linear programming problem**.

Example

$$C = 5x_1 + 6x_2 + 4x_3$$

$$x_1 + x_2 + x_3 \geq 2$$

$$x_1, x_2, x_3 \in \{0,1\}$$

x_1	x_2	x_3	C
0	1	1	10
1	0	1	9
1	1	0	11
1	1	1	15

Optimal

Remarks on integer programming

- Integer programming is NP-complete

- Running times depend exponentially on problem size,
  but problems of >1000 vars solvable with good solver (depending on
  the size and structure of the problem)

- ILP/LP models good starting point for modeling, even if heuristics are
  used in the end.

- Solvers: lp_solve (public), CPLEX (commercial), …

ILP Formulation of ML-RCS

- Minimize latency given constraints on area or the resources (ML-RCS)

- Use binary decision variables
  - $i = 0, 1, ..., n$
  - $l = 1, 2, ..., \lambda'+1$ λ' given upper-bound on latency
  - $x_{il} = 1$ if operation i starts at step l, 0 otherwise.

- Set of linear inequalities (constraints), and an objective function (min latency)

ILP Formulation of ML-RCS

- Observation

$$x_{il} = 0 \quad for \quad l < t_i^S \quad and \quad l > t_i^L$$

$$(t_i^S = ASAP(v_i), t_i^L = ALAP(v_i))$$

- $\quad t_i = \sum_l l \cdot x_{il}$ t_i = start time of op i.

- $\quad \displaystyle\sum_{m=l-d_i+1}^{l} x_{im} \overset{?}{=} 1 \quad \Rightarrow$ is op v_i (still) executing at step l?

Start Time vs. Execution Time

- For each operation v_i, only one start time
- If $d_i=1$, then the following questions are the same:
  - Does operation v_i **start** at step l?
  - Is operation v_i **running** at step l?
- But if $d_i>1$, then the two questions should be formulated as:
  - Does operation v_i **start** at step l?
    - Does $x_{il} = 1$ hold?
  - Is operation v_i **running** at step l?
    - Does the following hold?

$$\sum_{m=l-d_i+1}^{l} x_{im} = 1 \ ?$$

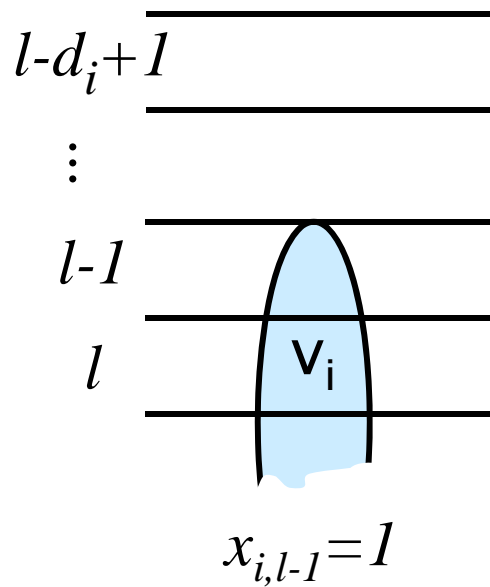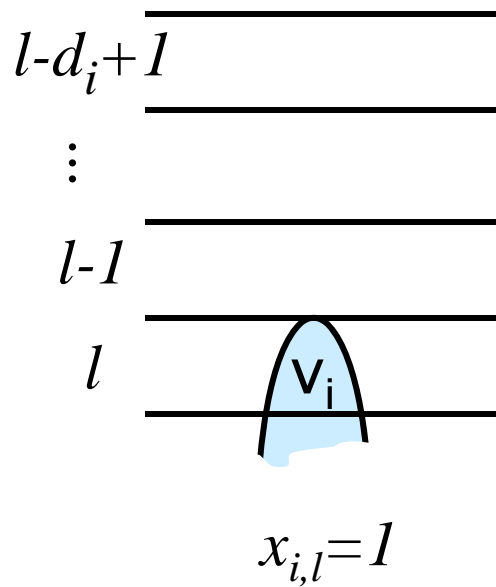# Operation v_i Still Running at Step I ?

- Is v_9 running at step 6?
    - Is $\quad x_{9,6} + x_{9,5} + x_{9,4} = 1 \quad$?

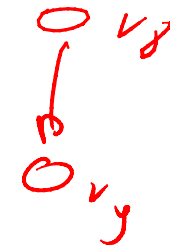$x_{9,6}=1$ $\qquad$ $x_{9,5}=1$ $\qquad$ $x_{9,4}=1$
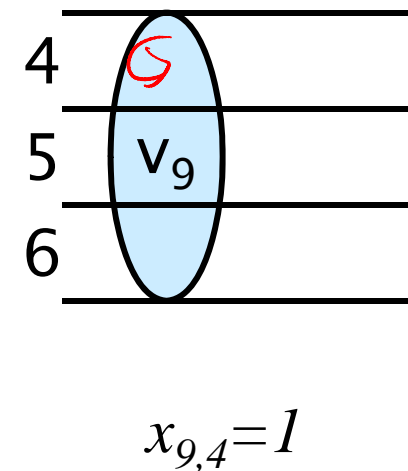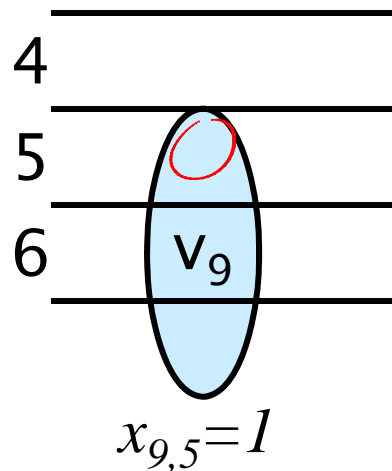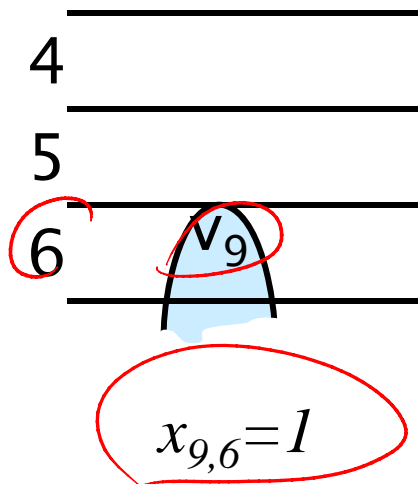
- Note:
    - Only one (if any) of the above three cases can happen
    - To meet resource constraints, we have to ask the same question for ALL steps, and ALL operations of that type

Operation v_i Still Running at Step l ?

- Is v_i running at step l ?
  - Is $\quad x_{i,l} + x_{i,l-1} + ... + x_{i,l-di+1} = 1$?

$x_{i,l}=1$

$x_{i,l-1}=1$

$x_{i,l-di+1}=1$
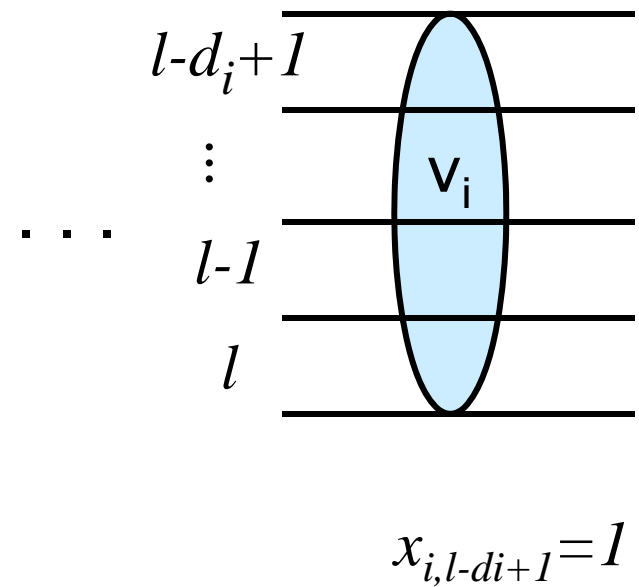
ILP Formulation of ML-RCS (cont.)

- Constraints:
  - Unique start times: $$\sum_l x_{il} = 1, \quad i = 0, 1, \ldots, n$$

  - Sequencing (dependency) relations must be satisfied

$$t_i \geq t_j + d_j \ \forall (v_j, v_i) \in E \Rightarrow \sum_l l \cdot x_{il} \geq \sum_l l \cdot x_{jl} + d_j$$
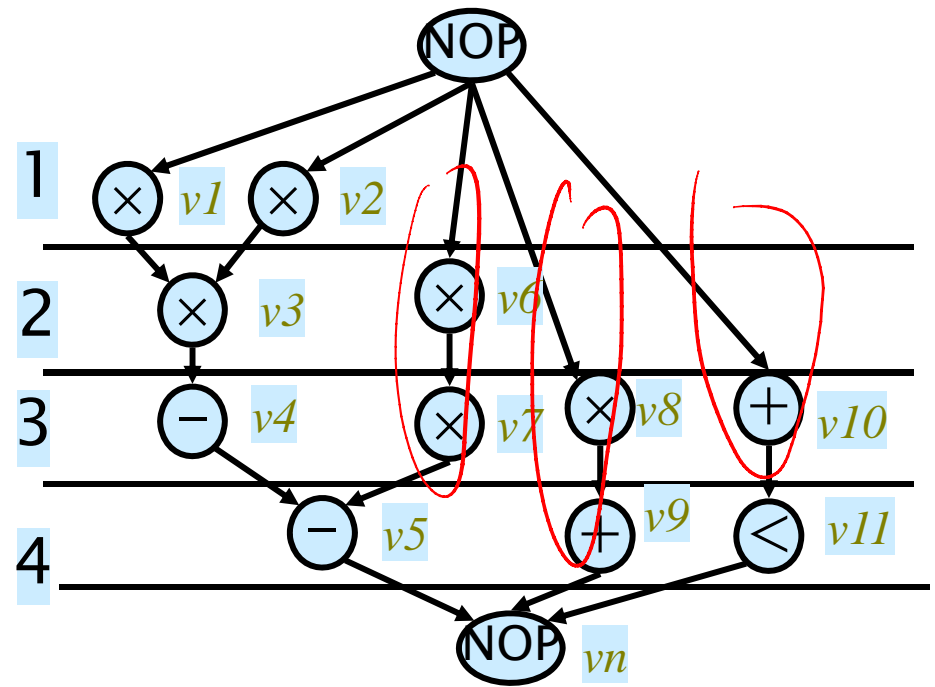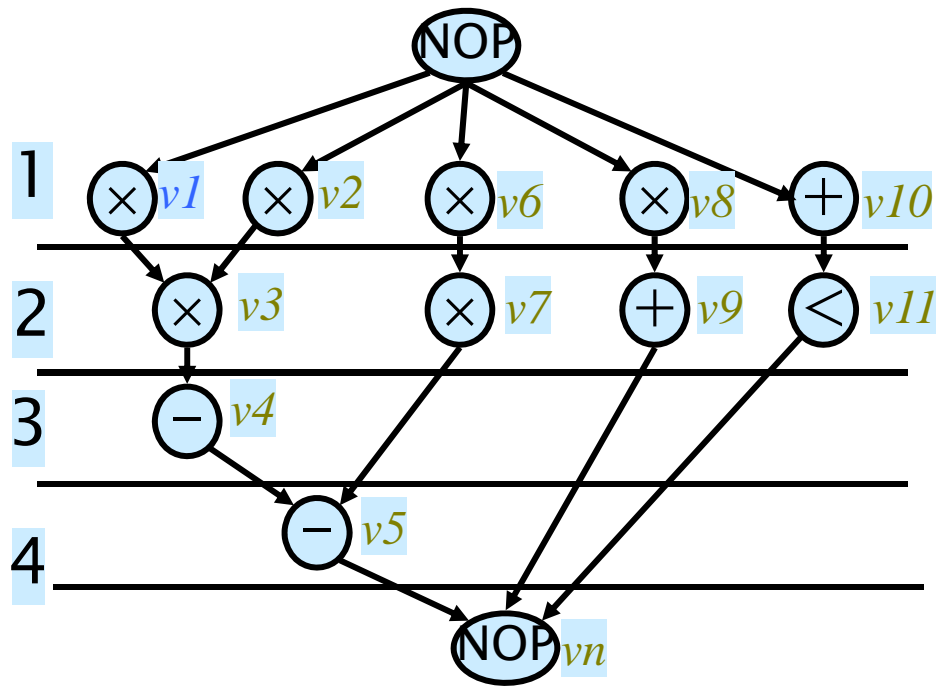
  - Resource constraints

$$\sum_{i:T(v_i)=k} \sum_{m=l-d_i+1}^{l} x_{im} \leq a_k, \quad k = 1, \ldots, n_{res}, \quad l = 1, \ldots, \bar{\lambda} + 1$$

- Objective: min $c^T t$.
  - t =start times vector, c =cost weight

ILP Example

- Assume $\overline{\lambda} = 4$

- First, perform ASAP and ALAP
  - (we can write the ILP without ASAP and ALAP, but using ASAP and ALAP will simplify the inequalities)
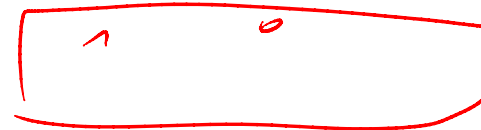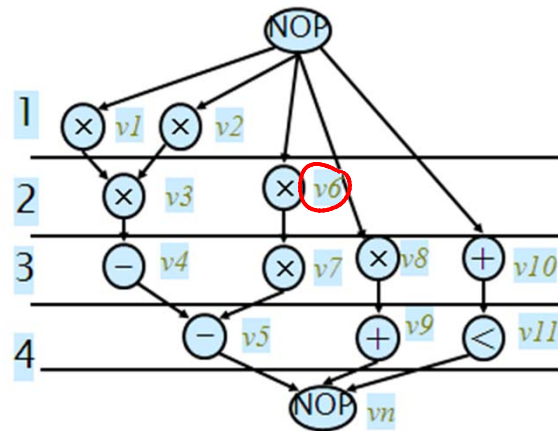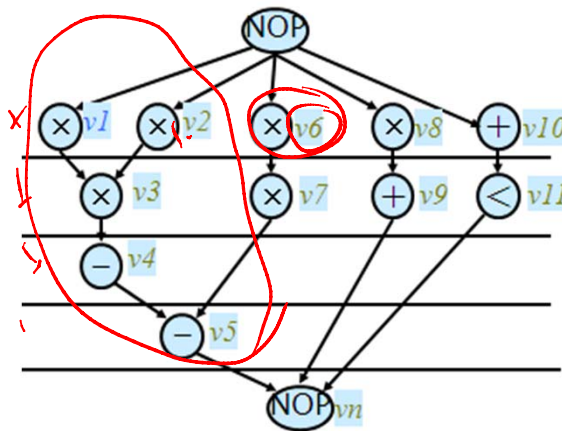
ILP Example: Unique Start Times Constraint

- Without using ASAP and ALAP values:

- Using ASAP and ALAP:

$$x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} = 1$$

$$x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} = 1$$

...

ILP Example: Dependency Constraints
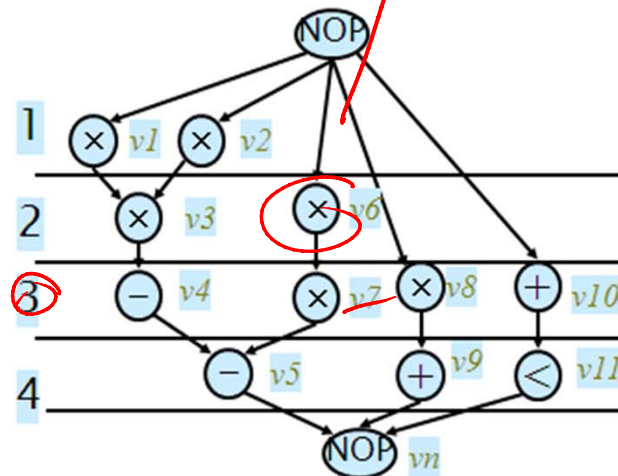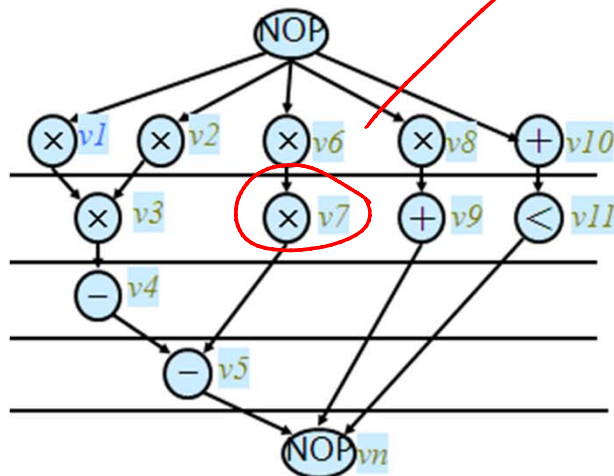
$$\tau_{(v_7)} - \tau_{(v_6)} = 1$$

- Using ASAP and ALAP, the non-trivial inequalities are:
  (assuming unit delay for + and *)

$$2.x_{7,2} + 3.x_{7,3} - x_{6,1} - 2.x_{6,2} - 1 \geq 0$$

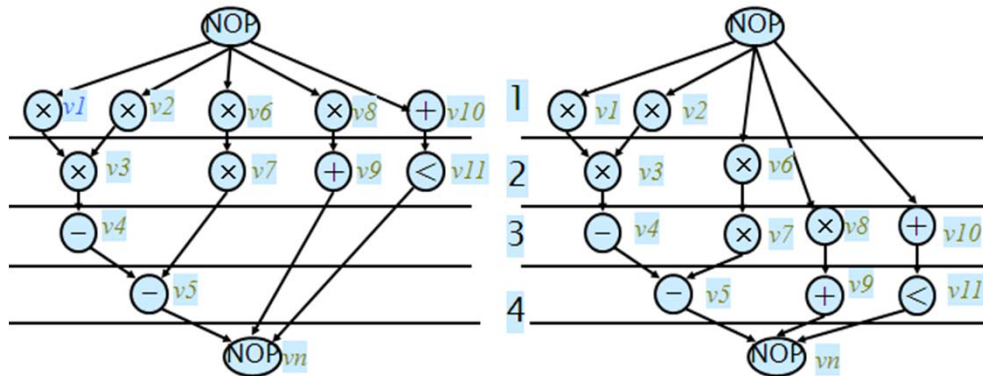$$2.x_{9,2} + 3.x_{9,3} + 4.x_{9,4} - x_{8,1} - 2.x_{8,2} - 3.x_{8,3} - 1 \geq 0$$

$$2.x_{11,2} + 3.x_{11,3} + 4.x_{11,4} - x_{10,1} - 2.x_{10,2} - 3.x_{10,3} - 1 \geq 0$$

$$4.x_{5,4} - 2.x_{7,2} - 3.x_{7,3} - 1 \geq 0$$

…

ILP Example: Resource Constraints

- Resource constraints (assuming 2 adders and 2 multipliers)

$$x_{1,1} + x_{2,1} + x_{6,1} + x_{8,1} \leq 2$$

$$x_{3,2} + x_{6,2} + x_{7,2} + x_{8,2} \leq 2$$

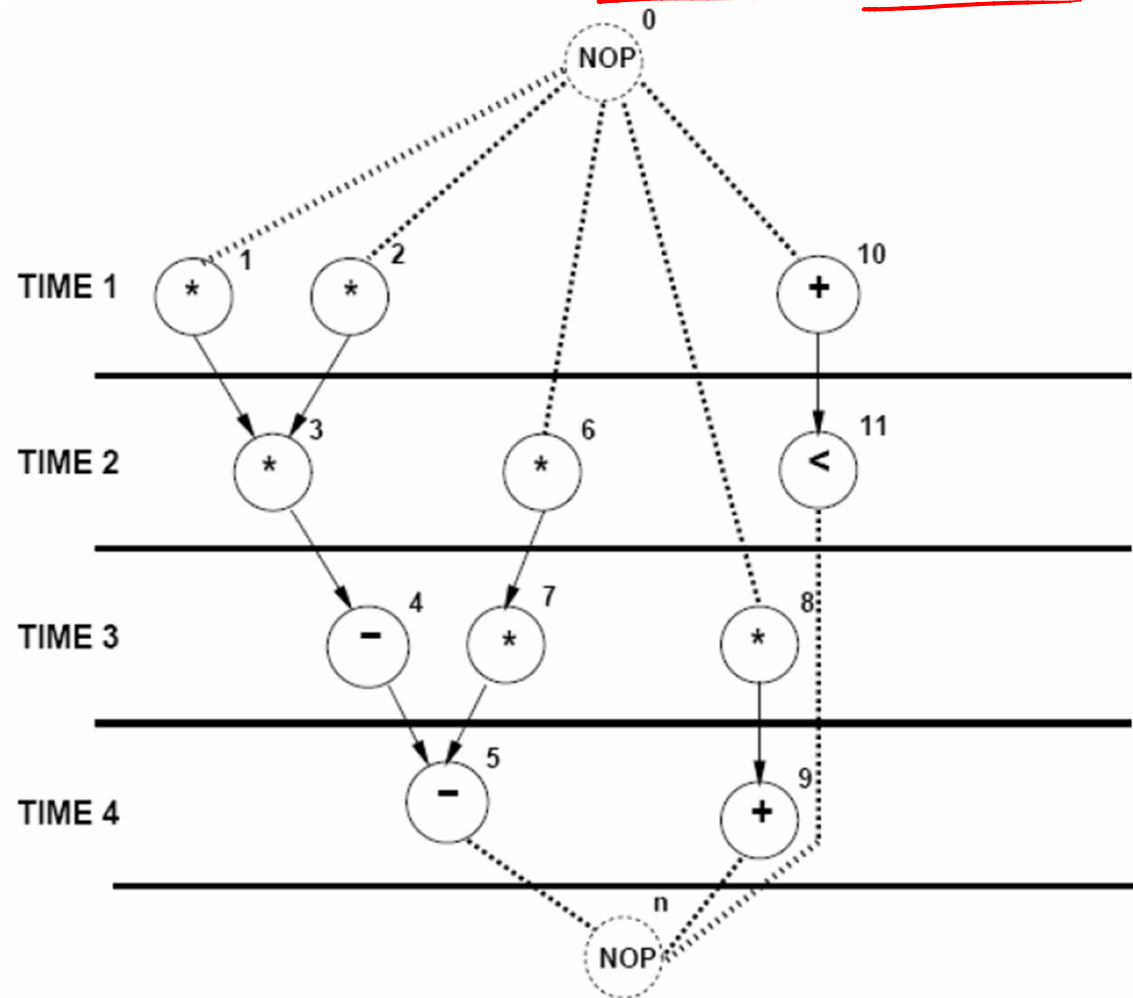$$x_{7,3} + x_{8,3} \leq 2$$

$$x_{10,1} \leq 2$$

$$x_{9,2} + x_{10,2} + x_{11,2} \leq 2$$

$$x_{4,3} + x_{9,3} + x_{10,3} + x_{11,3} \leq 2$$

$$x_{5,4} + x_{9,4} + x_{11,4} \leq 2$$

- Objective: *Min $X_{n,1} + 2X_{n,2} + 3X_{n,3} + 4X_{n,4}$*

Result is different from both ALAP and ASAP schedules

(Time constrained) Force-directed scheduling

© ACM

- Goal: balanced utilization of resources
- Based on spring model
- Originally proposed for high-level synthesis
- Force
    - Used as a priority function
    - Related to concurrency – sort operations for least force
    - Mechanical analogy: Force = constant x displacement
        - Constant = operation-type distribution
        - Displacement = change in probability
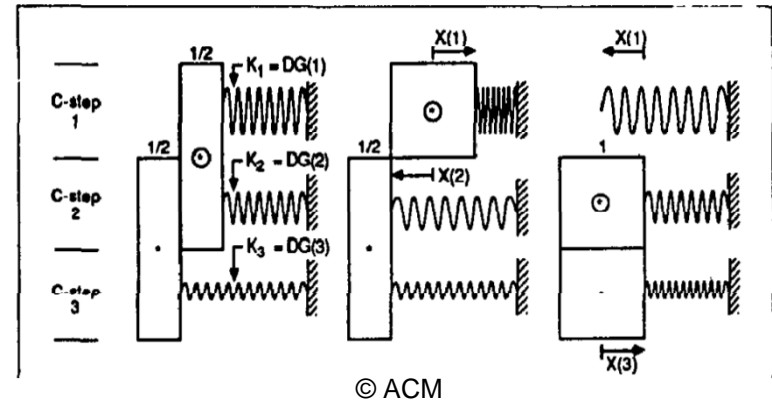
* [Pierre G. Paulin, J.P. Knight, Force-directed scheduling in automatic data path synthesis, *Design Automation Conference* (DAC), 1987, S. 195-202]

Force-Directed Scheduling

The Force-Directed Scheduling approach reduces the amount of:

- Functional Units
- Registers
- Interconnect

This is achieved by balancing the concurrency of operations to ensure a high utilization of each unit.
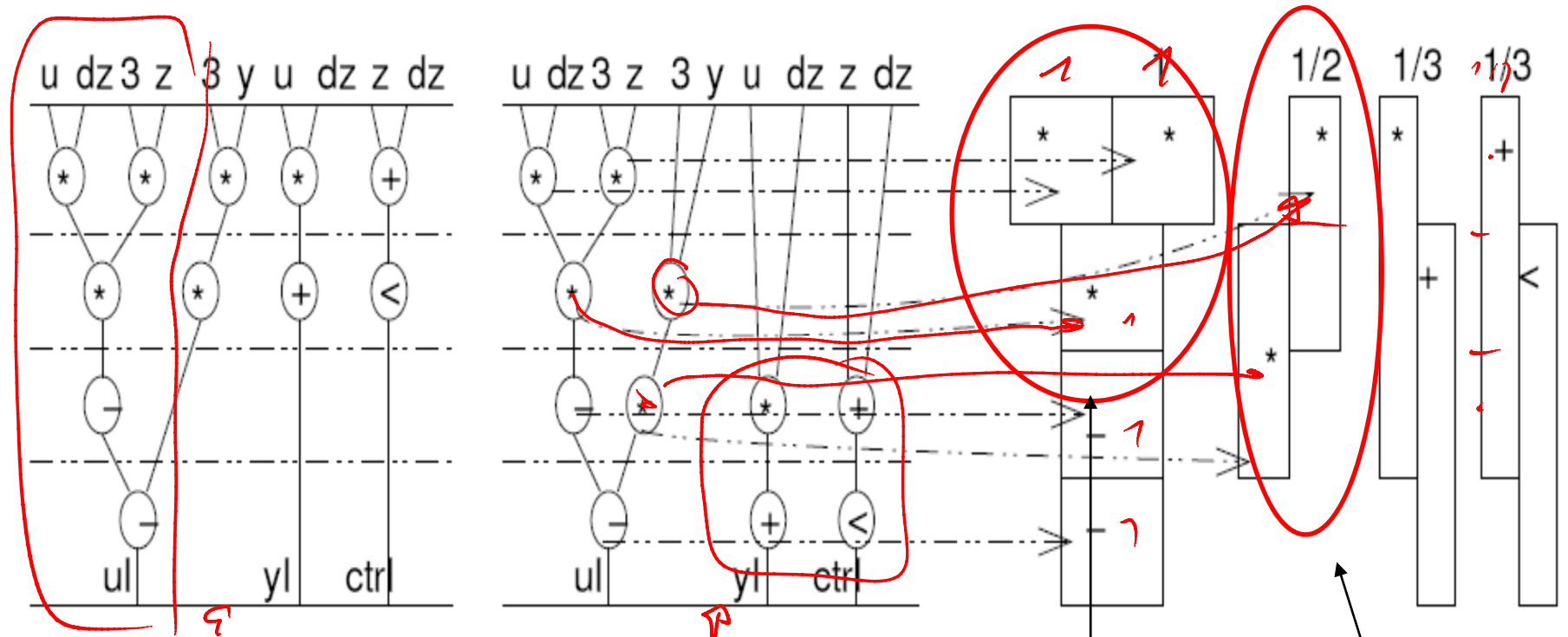
Next: computation of "forces"

- Direct forces push each task into the direction of lower values of $D(i)$.
- Impact of direct forces on dependent tasks taken into account by indirect forces
- Balanced resource usage $\approx$ smallest forces
- For our simple example and time constraint=6: result = ALAP schedule
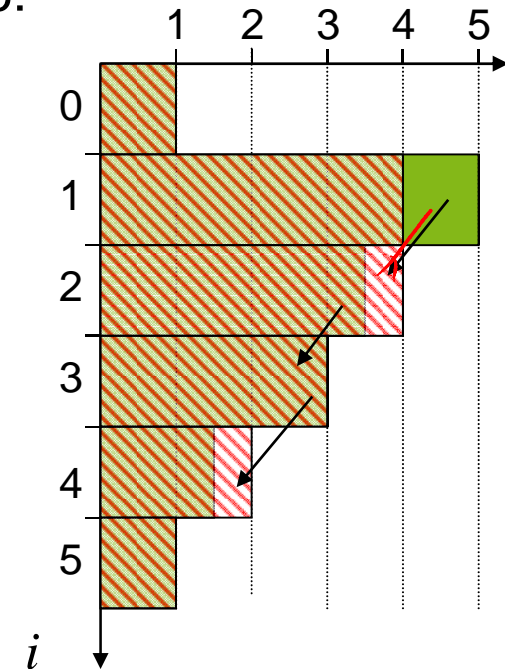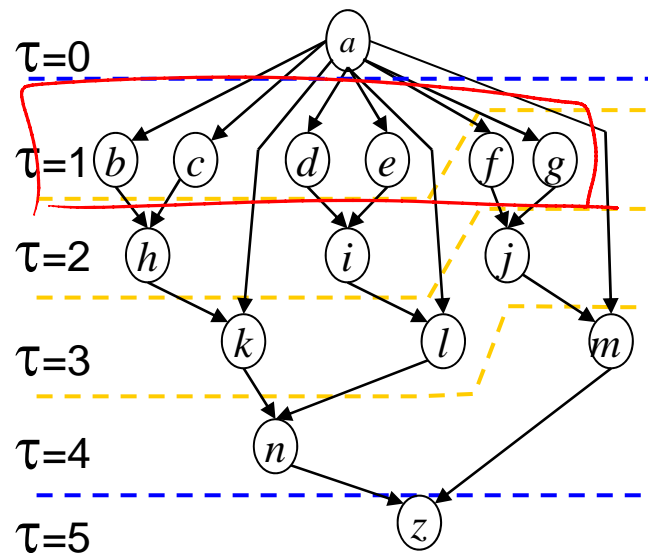
1. Compute time frames $R(j)$
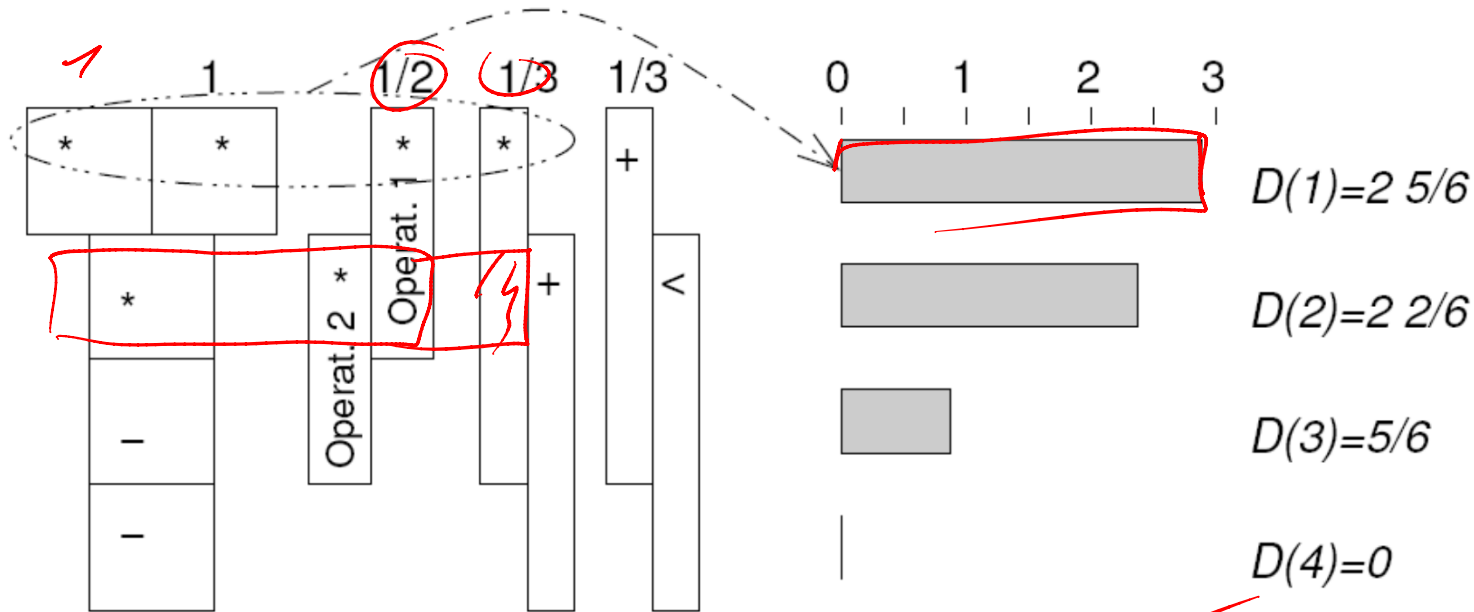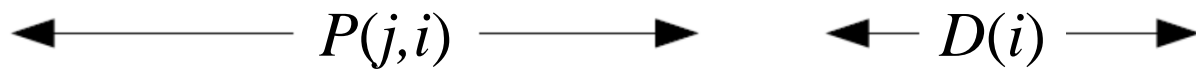# 2. Compute "probability" $P(j,i)$ of assignment $j \to i$

$R(j)=\{\text{ASAP-control step} \dots \text{ALAP-control step}\}$

$$P(j, i) = \begin{cases} \frac{1}{|R(j)|} & \text{if} \quad i \in R(j) \\ 0 \text{ otherwise} \end{cases}$$

Fixed

Free

3. Compute "distribution" $D(i)$ (# Operations in control step i)

$$D(i) = \sum_{j, type(j) \in H} P(j,i)$$

$$\longleftarrow \quad P(j,i) \quad \longrightarrow \qquad \longleftarrow D(i) \longrightarrow$$

$D(1) = 2\ 5/6$

$D(2) = 2\ 2/6$

$D(3) = 5/6$

$D(4) = 0$

MULT

CS - ES

- 56 -

Example

$$q_{add}(1) = \frac{1}{3} = 0.33$$

$$q_{add}(2) = \frac{1}{3} + \frac{1}{3} + \frac{1}{3} = 1$$

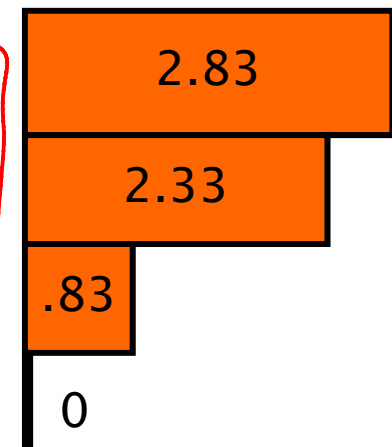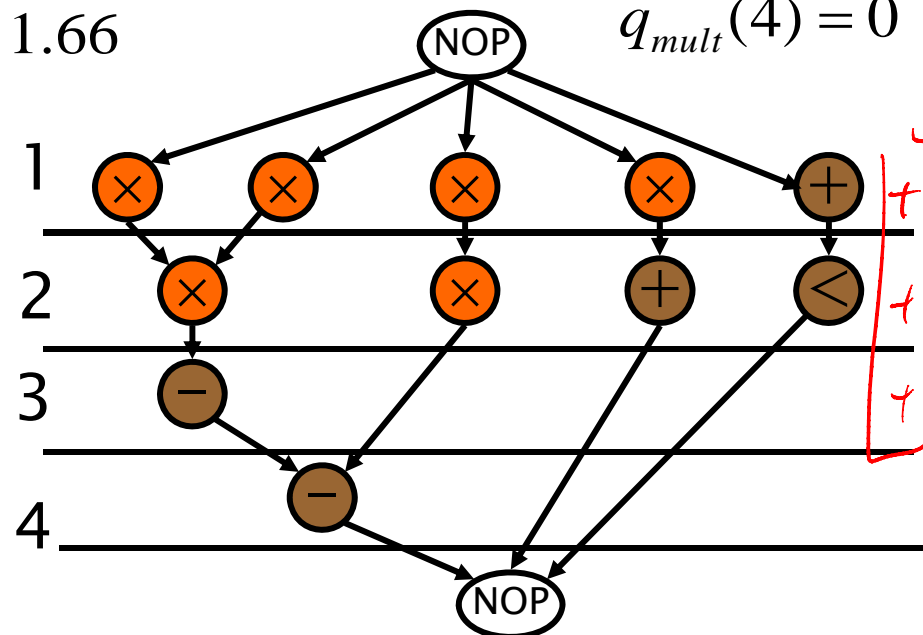$$q_{add}(3) = 1 + \frac{1}{3} + \frac{1}{3} + \frac{1}{3} = 2$$

$$q_{add}(4) = 1 + \frac{1}{3} + \frac{1}{3} = 1.66$$

$$q_{mult}(1) = 1 + 1 + \frac{1}{2} + \frac{1}{3} = 2.83$$

$$q_{mult}(2) = 1 + \frac{1}{2} + \frac{1}{2} + \frac{1}{3} = 2.33$$

$$q_{mult}(3) = \frac{1}{2} + \frac{1}{3} = 0.83$$

$$q_{mult}(4) = 0$$
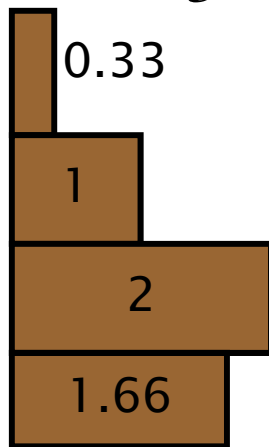
Scheduling – An example

Step 3 : Calculate the *force* (a new metric)

A metric called *force* is introduced. The force is used to optimize the utilization of units. A high positive force value indicates a poor utilization.

$$Force(j) = DG(j) - \sum_{i=t}^{b} \frac{DG(i)}{(b-t+1)}$$

Scheduling – An example

Step 3 : Calculate the *force* (a new metric)
With the operation x' in control-step 1.
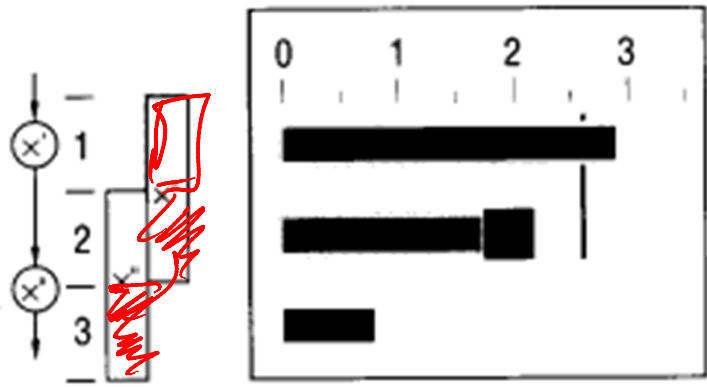
$$Force(1) = DG(1) - \sum_{i=1}^{2} \frac{DG(i)}{2}$$

$$= 2.833 - \frac{2.833 + 2.333}{2} = 0.25$$

$DG(1) = 2.833$ $DG(3) = 0.833$ Poor
$DG(2) = 2.333$ $DG(4) = 0$ utilization

Scheduling – An example

Indirect force (on x'' in control-step 3)

Direct force (calculated as before)

Step 3 : Calculate the *force* (a new metric)
With the operation x' in control-step 2.
(x'' must be in control-step 3)

$$Force\ (2) = DG\ (2) - \sum_{i=1}^{2} \frac{DG\ (i)}{2} + DG\ (3) - \sum_{i=2}^{3} \frac{DG\ (i)}{2}$$

$$= 2.333 - \frac{2.833 + 2.333}{2} + 0.833 - \frac{0.833 + 2.333}{2} = -1$$

$DG(1) = 2.833$ $DG(3) = 0.833$ Good
$DG(2) = 2.333$ $DG(4) = 0$ utilization

Scheduling – An example

By repeatedly assigning operations to various control-steps and calculating the force associated with the choice several force values will be available.

The Force-directed scheduling algorithm chooses the assignment with the lowest force value, which also balances the concurrency of operations most efficiently.

Overall approach

- **procedure** forceDirectedScheduling;
  begin
  AsapScheduling;
  AlapScheduling;
      while not all tasks scheduled **do**
        begin

          select task T with smallest total force;

          schedule task T at time step minimizing forces;

          recompute forces;

        end;
    end

May be repeated for different task/ processor classes

Not sufficient for today's complex, heterogeneous hardware platforms

Force-Directed Scheduling

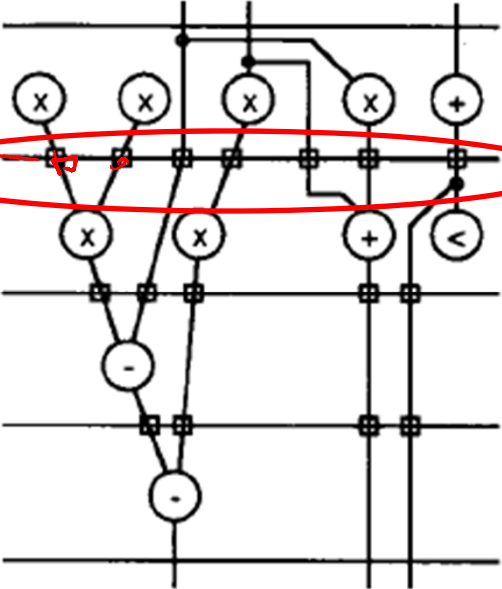The Force-Directed Scheduling approach reduces the amount of:

- Functional Units
- Registers
- Interconnect

By introducing Registers and Interconnect as *storage operations*, the force is calcuted for these as well.
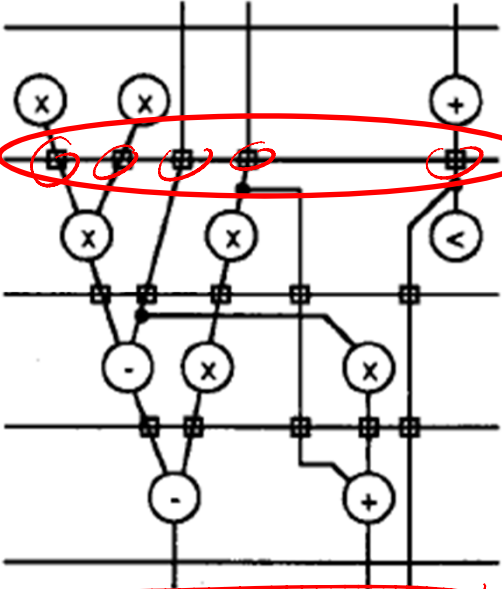
Force-Directed Scheduling

CS - ES

- Architecture Synthesis

→
- HW/SW Codesign

- Power Aware Computing

- 3.2.2011 Lecture by Bernd Finkbeiner, Head of Reactive Systems Group at Saarland University(http://react.cs.uni-sb.de/

Codesign Definition and Key Concepts

- Codesign
  - The meeting of system-level objectives by exploiting the trade-offs between hardware and software in a system through their concurrent design

- Key concepts
  - Concurrent: hardware and software developed at the same time on parallel paths
  - Integrated: interaction between hardware and software development to produce design meeting performance criteria and functional specs

Typical Codesign Process

System Description (Functional) — Concurrent processes / Programming languages

HW/SW Partitioning — Unified representation (Data/control flow)

SW → Software Synthesis

Interface Synthesis

HW → Hardware Synthesis

Another HW/SW partition

System Integration — Instruction set level / HW/SW evaluation