

# Mapping of Applications to Platforms

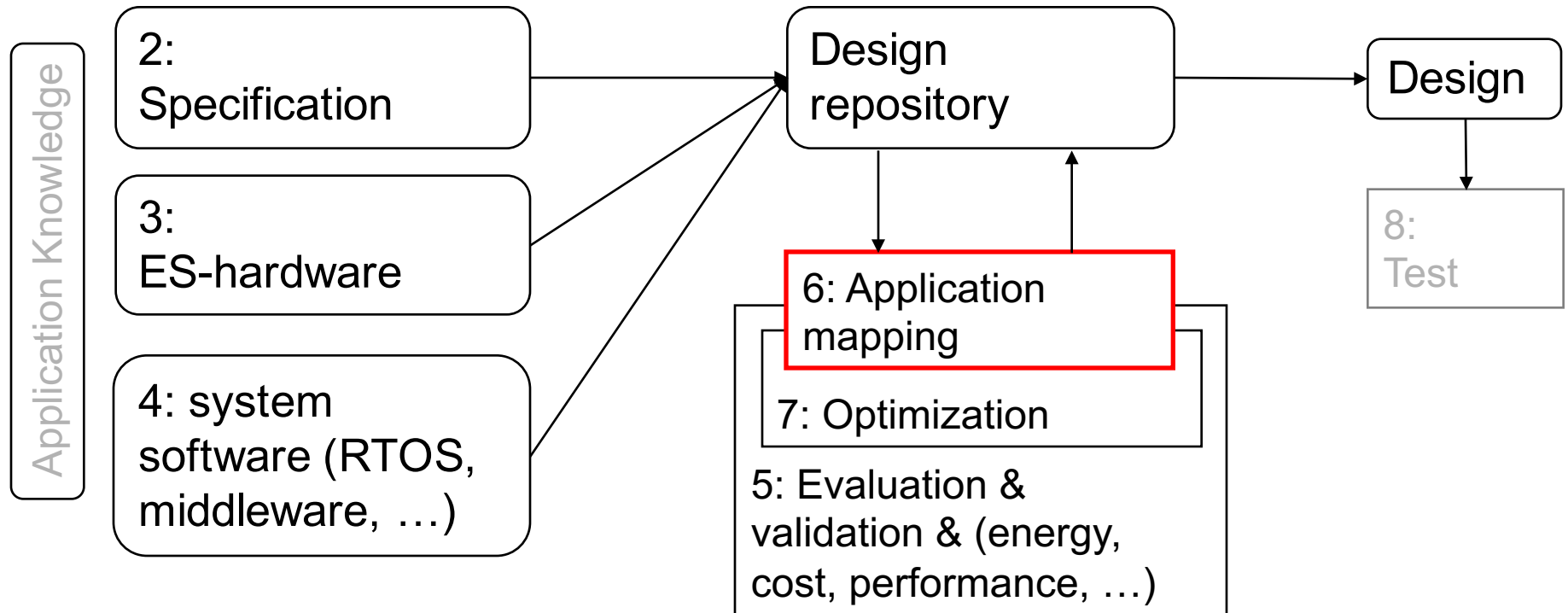
Jian-Jia Chen

(slides are based on  
Peter Marwedel)

TU Dortmund, Informatik 12  
Germany

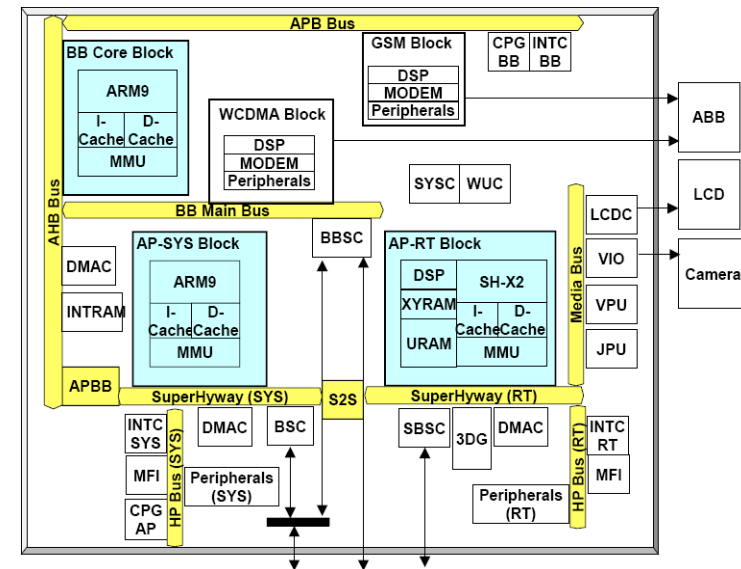
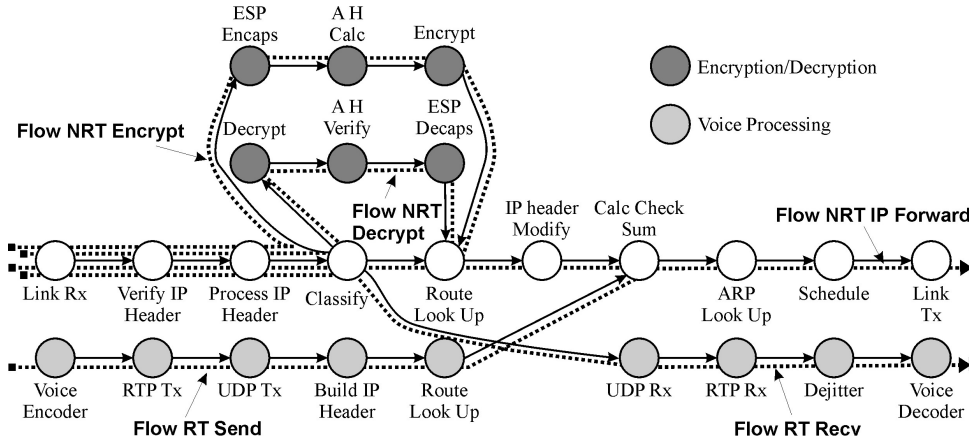
2020年 01 月 22 日

# Structure of this course



Numbers denote sequence of chapters

# Mapping of Applications to Platforms



© Renesas, Thiele

# Distinction between mapping problems

	Embedded	PC-like
Architectures	Frequently heterogeneous very compact	Mostly homogeneous not compact (x86 etc)
x86 compatibility	Less relevant	Very relevant
Architecture fixed?	Sometimes not	Yes
Model of computation (MoCs)	C+multiple models (data flow, discrete events, ...)	Mostly von Neumann (C, C++, Java)
Optim. objectives	Multiple (energy, size, ...)	Average performance dominates
Real-time relevant	Yes, very!	Hardly
Applications	Several concurrent apps.	Mostly single application
Apps. known at design time	Most, if not all	Only some (e.g. WORD)

# Problem Description

---

## Given

- A set of applications
- Scenarios on how these applications will be used
- A set of candidate architectures comprising
  - (Possibly heterogeneous) processors
  - (Possibly heterogeneous) communication architectures
  - Possible scheduling policies

**Tools urgently needed!**

## Find

- A mapping of applications to processors
- Appropriate scheduling techniques (if not fixed)
- A target architecture (if DSE is included)

## Objectives and constraints

- deadlines, temperatures
- Cost, performance, energy, reliability

# Related Work

---

- Mapping to ECUs in automotive design
- Scheduling theory:  
Provides insight for the mapping *task* → *start times*
- Hardware/software partitioning:  
Can be applied if it supports multiple processors
- High performance computing (HPC)  
Automatic parallelization, but only for
  - single applications,
  - fixed architectures,
  - no support for scheduling,
  - memory and communication model usually different
- High-level synthesis  
Provides useful terms like scheduling, allocation, assignment
- Optimization theory

# Scope of mapping algorithms

---

## Useful terms from hardware synthesis:

- **Resource Allocation**

Decision concerning type and number of available resources

- **Resource Assignment**

Mapping: Task  $\rightarrow$  (Hardware) Resource

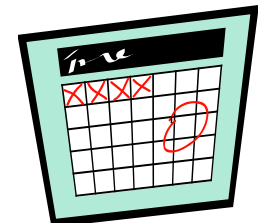
- **xx to yy binding:**

Describes a mapping from behavioral to structural domain, e.g. task to processor binding, variable to memory binding

- **Scheduling**

Mapping: Tasks  $\rightarrow$  Task start times

Sometimes, resource assignment is considered being included in scheduling.



# Classes of mapping algorithms considered in this course

---

- ➔ **Classical scheduling algorithms**  
Mostly for independent tasks & ignoring communication, mostly for mono- and homogeneous multiprocessors (EDF, EDD, RM, DM, etc.)
- **Dependent tasks as considered in architectural synthesis**  
Initially designed in different context, but applicable
- **Hardware/software partitioning**  
Dependent tasks, heterogeneous systems, focus on resource assignment
- **Design space exploration using genetic algorithms**  
Heterogeneous systems, incl. communication modeling



# Classes of mapping algorithms considered in this course

---

- **Classical scheduling algorithms**

Mostly for independent tasks & ignoring communication, mostly for mono- and homogeneous multiprocessors (EDF, EDD, RM, DM, etc.)

- ➡ ■ **Dependent tasks as considered in architectural synthesis**

Initially designed in different context, but applicable

- **Hardware/software partitioning**

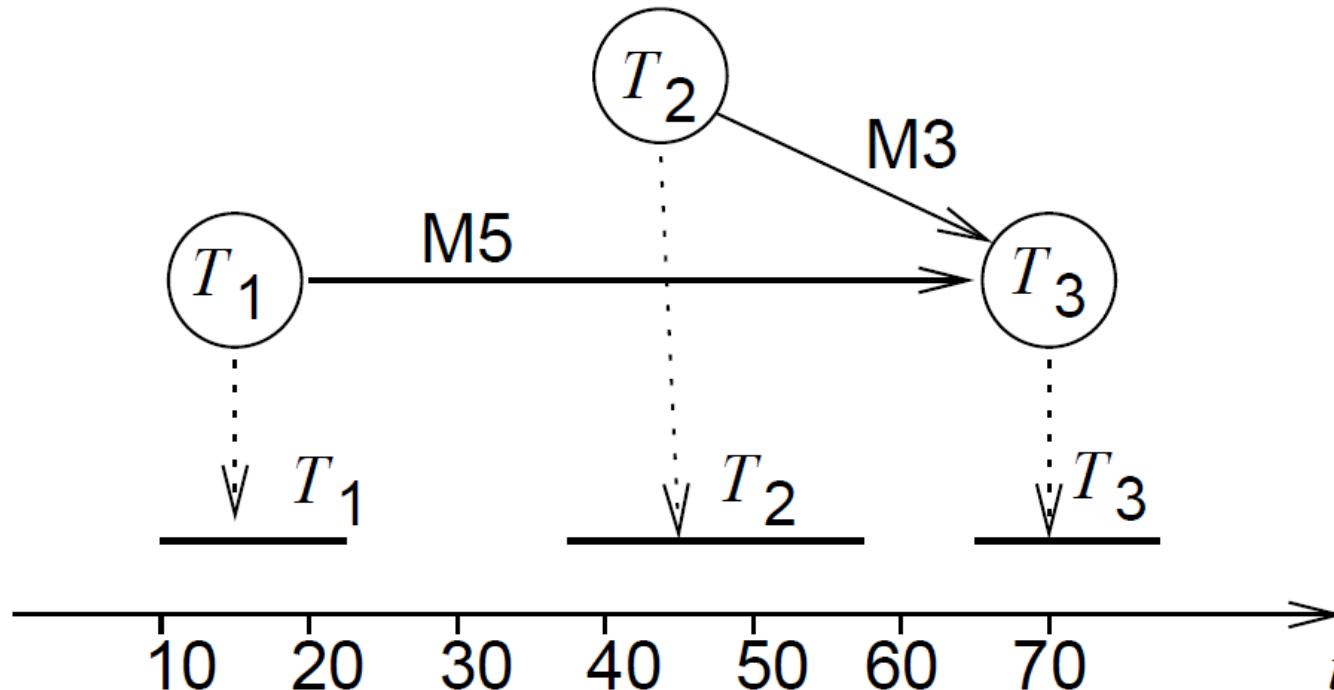
Dependent tasks, heterogeneous systems, focus on resource assignment

- **Design space exploration using genetic algorithms**

Heterogeneous systems, incl. communication modeling

# Scheduling with precedence constraints

Task graph and possible schedule:

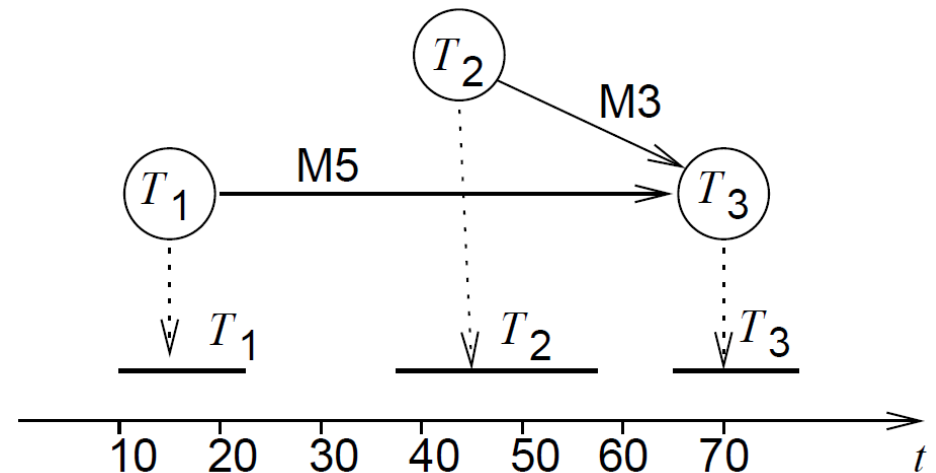


# Simultaneous Arrival Times: The Latest Deadline First (LDF) Algorithm

LDF [Lawler, 1973]: reads the task graph and **among the tasks with no successors inserts the one with the latest deadline** into a queue. It then repeats this process, putting tasks whose successor have all been selected into the queue.

At run-time, the tasks are executed in an order **opposite** to the generated total order.

LDF is non-preemptive and is optimal for mono-processors.



If no local deadlines exist, LDF performs just a topological sort.

# Asynchronous Arrival Times: Modified EDF Algorithm

---

This case can be handled with a modified EDF algorithm. The key idea is to transform the problem from a given set of dependent tasks into a set of independent tasks with different timing parameters [Chetto90]. This algorithm is optimal for mono-processor systems.

If preemption is not allowed, the heuristic algorithm developed by Stankovic and Ramamritham can be used.

# Dependent tasks

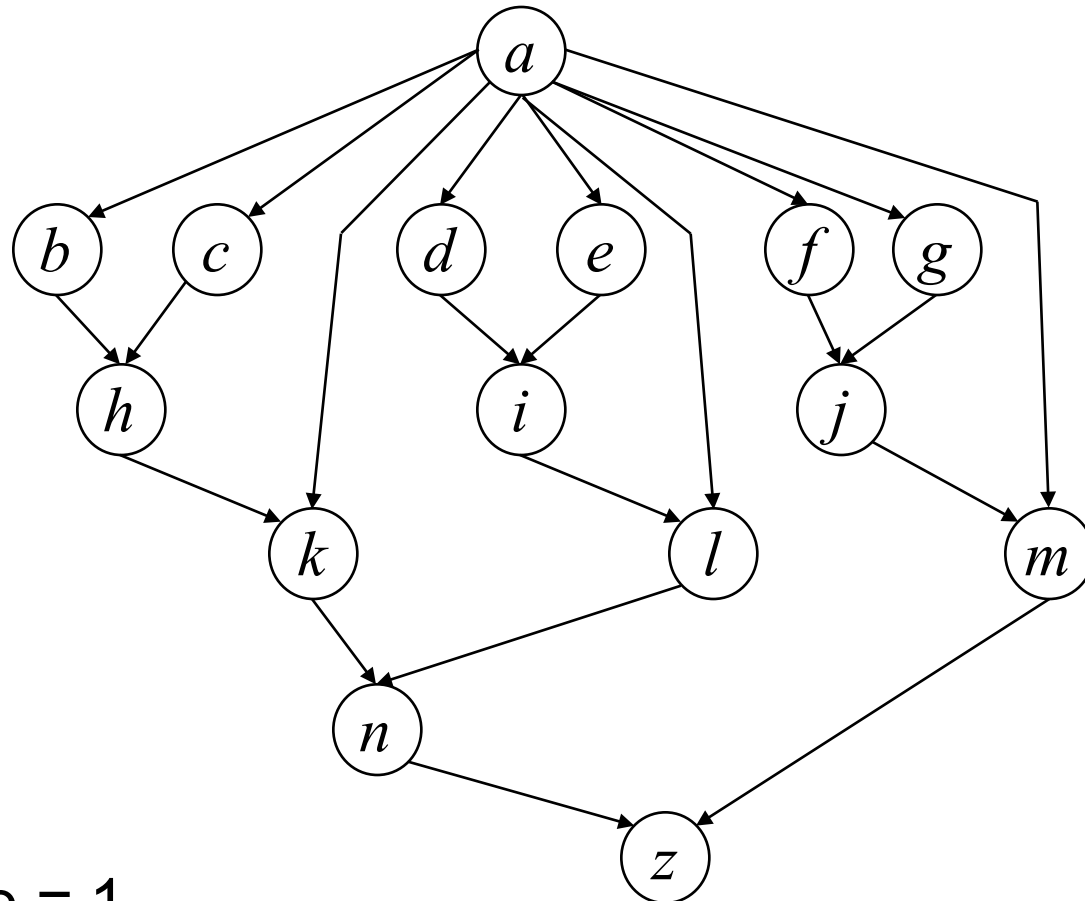
---

The problem of deciding whether or not a schedule exists for a set of dependent tasks and a given deadline is NP-complete in a strong sense in general [Garey/Johnson].

Strategies:

1. Add resources, so that scheduling becomes easier
2. Split problem into static and dynamic part so that only a minimum of decisions need to be taken at run-time.
- ➔ 3. Use scheduling algorithms from high-level synthesis

# Task graph



Assumption:  
execution time = 1  
for all tasks

# As soon as possible (ASAP) scheduling

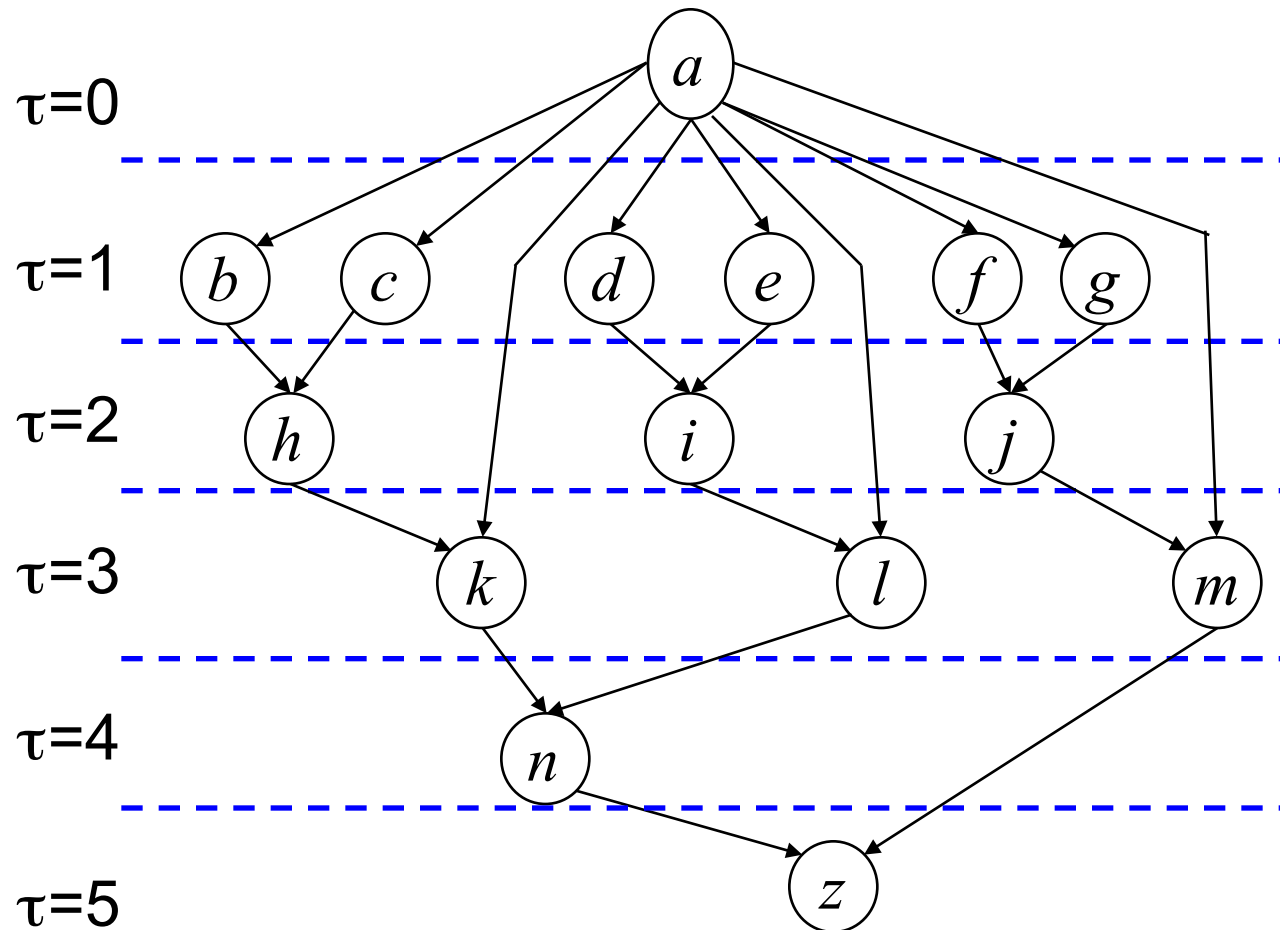
---

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.

# As soon as possible (ASAP) scheduling: Example



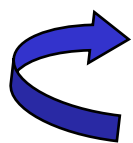


# As-late-as-possible (ALAP) scheduling

---

ALAP: All tasks are scheduled as late as possible

Start at last time step\*:

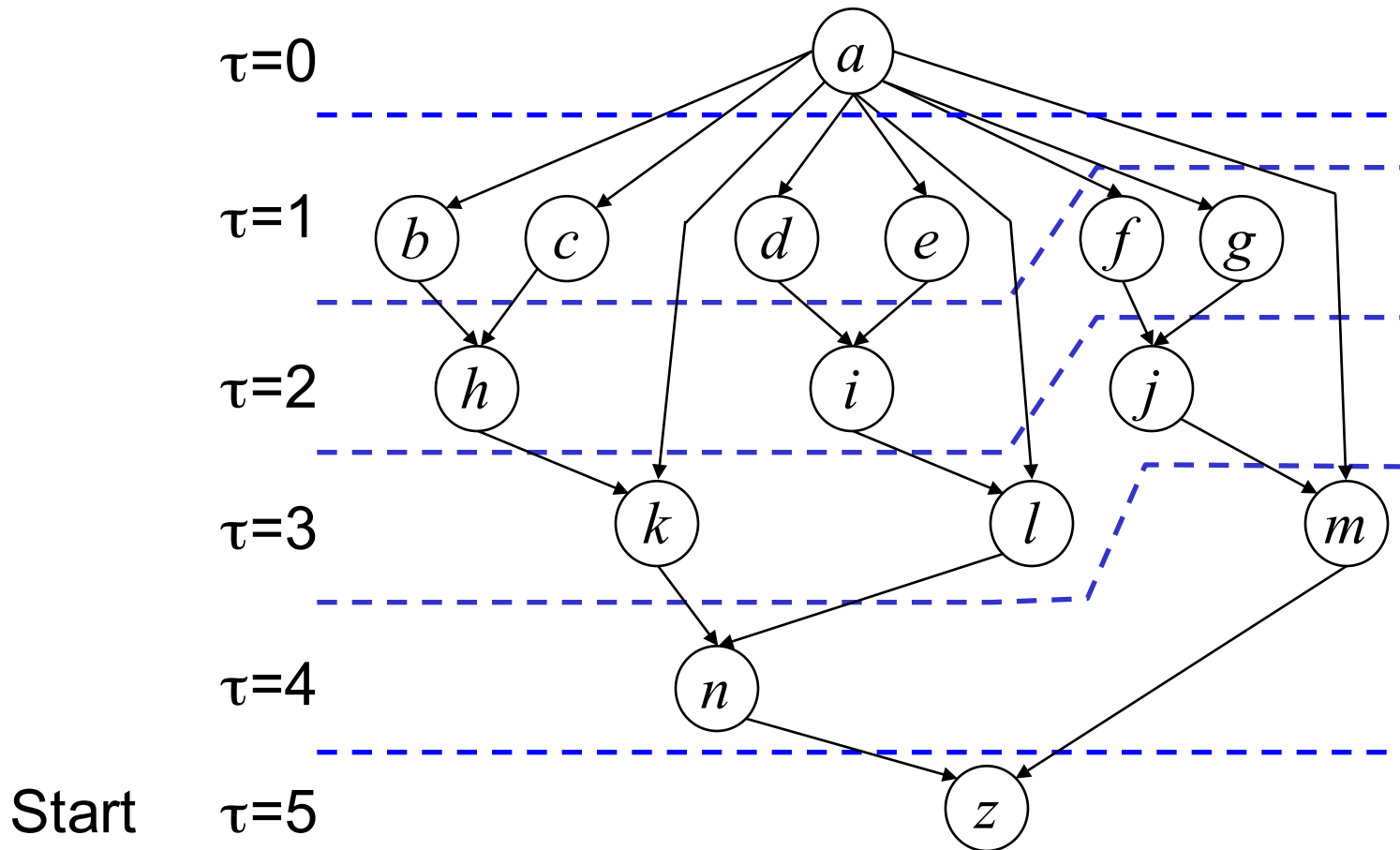


Schedule tasks with no successors and tasks for which all successors have already been scheduled.

---

\* Generate a list, starting at its end

# As-late-as-possible (ALAP) scheduling: Example



# (Resource constrained) List Scheduling

---

List scheduling: extension of ALAP/ASAP method

Preparation:

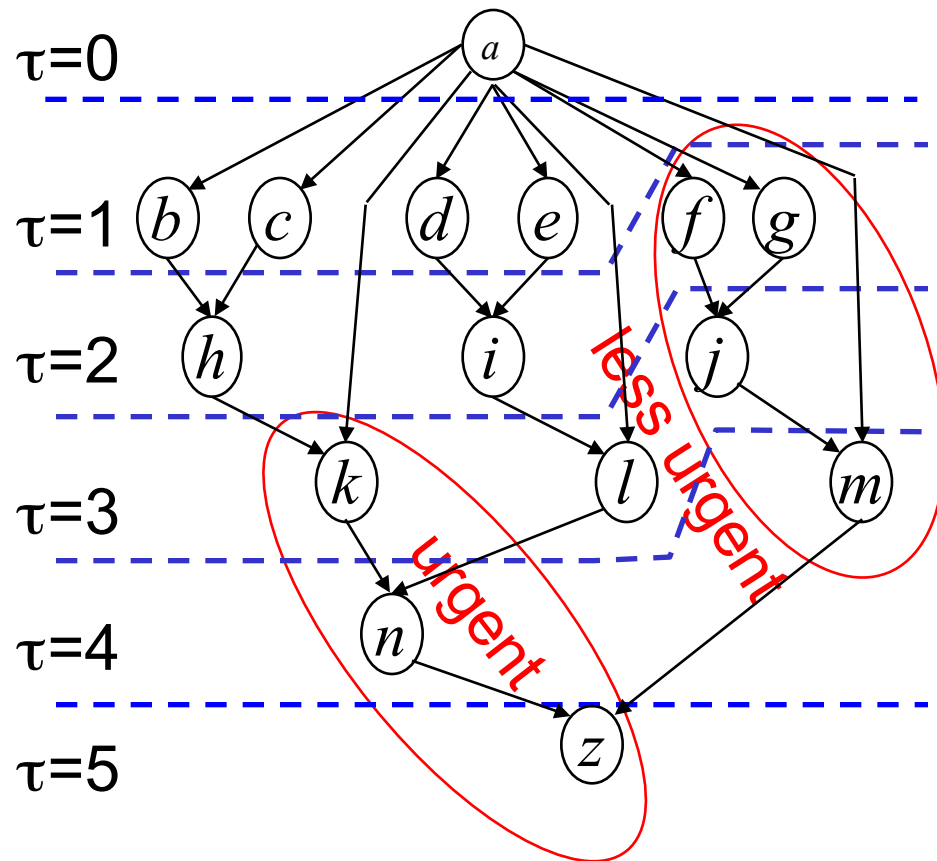
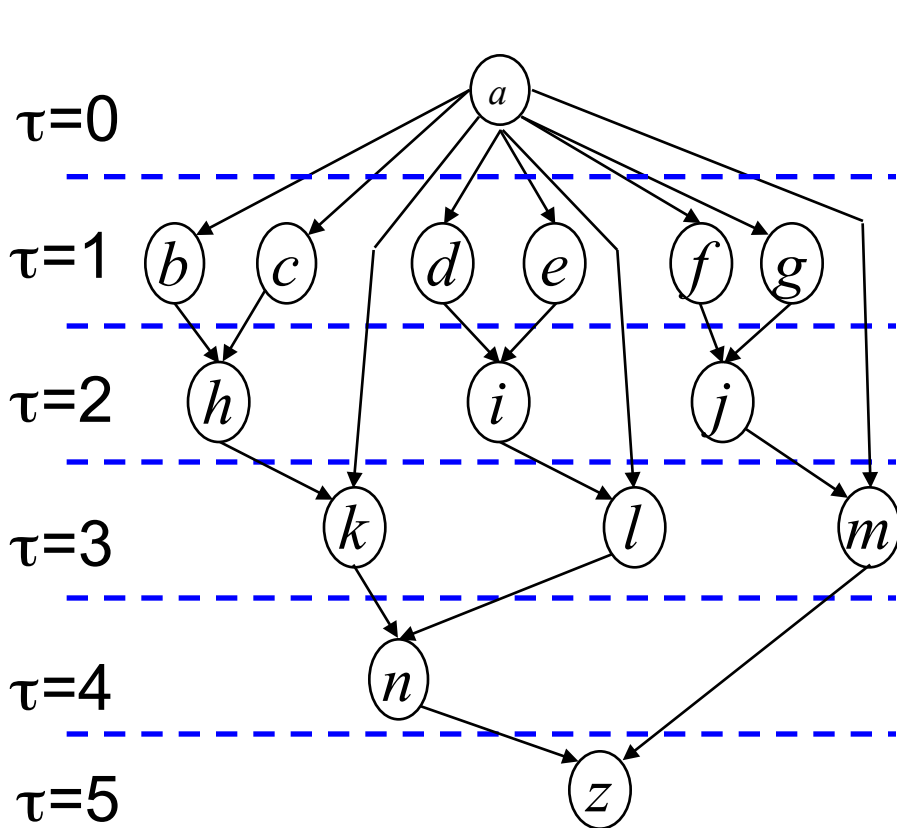
- Topological sort of task graph  $G=(V,E)$
- Computation of priority of each task:

Possible priorities  $u$ :

- Number of successors
- Longest path
- **Mobility** =  $\tau$  (ALAP schedule) -  $\tau$  (ASAP schedule)

# 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** ( $\tau$ );

}

} may be repeated for different task/processor classes

# Example

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

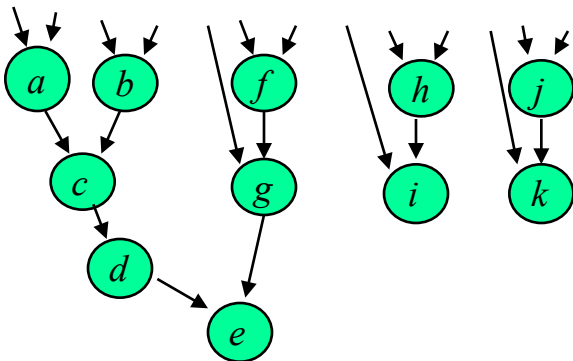
$$u(a) = u(b) = 4$$

$$u(c) = u(f) = 3$$

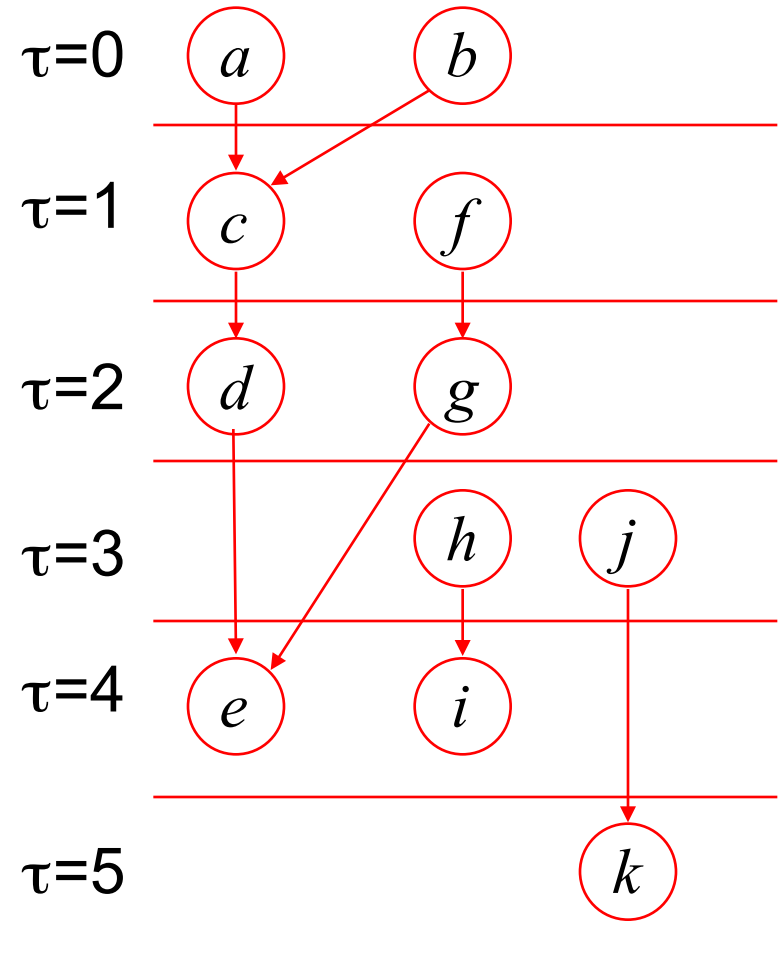
$$u(d) = u(g) = u(h) = u(j) = 2$$

$$u(e) = u(i) = u(k) = 1$$

$$\forall i : G_i = 0$$



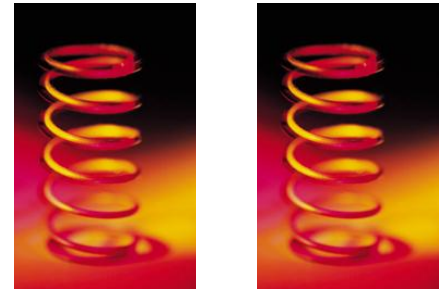
Modified example  
based on J. Teich



# (Time constrained) Force-directed scheduling

---

- Goal: balanced utilization of resources
- Based on spring model;
- Originally proposed for high-level synthesis



---

Pierre G. Paulin, J.P. Knight, Force-directed scheduling in automatic data path synthesis, *Design Automation Conference (DAC)*, 1987, S. 195-202

# Evaluation of HLS-Scheduling

---

- Focus on considering dependencies
- Mostly heuristics, few proofs on optimality
- Not using global knowledge about periods etc.
- Considering discrete time intervals
- Variable execution time available only as an extension
- Includes modeling of heterogeneous systems



# Overview

---

Scheduling of aperiodic tasks with real-time constraints:  
Table with some known algorithms:

	Equal arrival times; non-preemptive	Arbitrary arrival times; preemptive
Independent tasks	EDD (Jackson)	EDF (Horn)
Dependent tasks	LDF (Lawler), ASAP, ALAP, LS, FDS	EDF* (Chetto)

# Conclusion

---

- HLS-based scheduling
  - ASAP
  - ALAP
  - *List scheduling (LS)*
  - *Force-directed scheduling (FDS)*
- Evaluation

# Classes of mapping algorithms considered in this course

---

- **Classical scheduling algorithms**

Mostly for independent tasks & ignoring communication, mostly for mono- and homogeneous multiprocessors (EDF, EDD, RM, DM, etc.)

- **Dependent tasks as considered in architectural synthesis**

Initially designed in different context, but applicable

- ➡ ■ **Hardware/software partitioning**

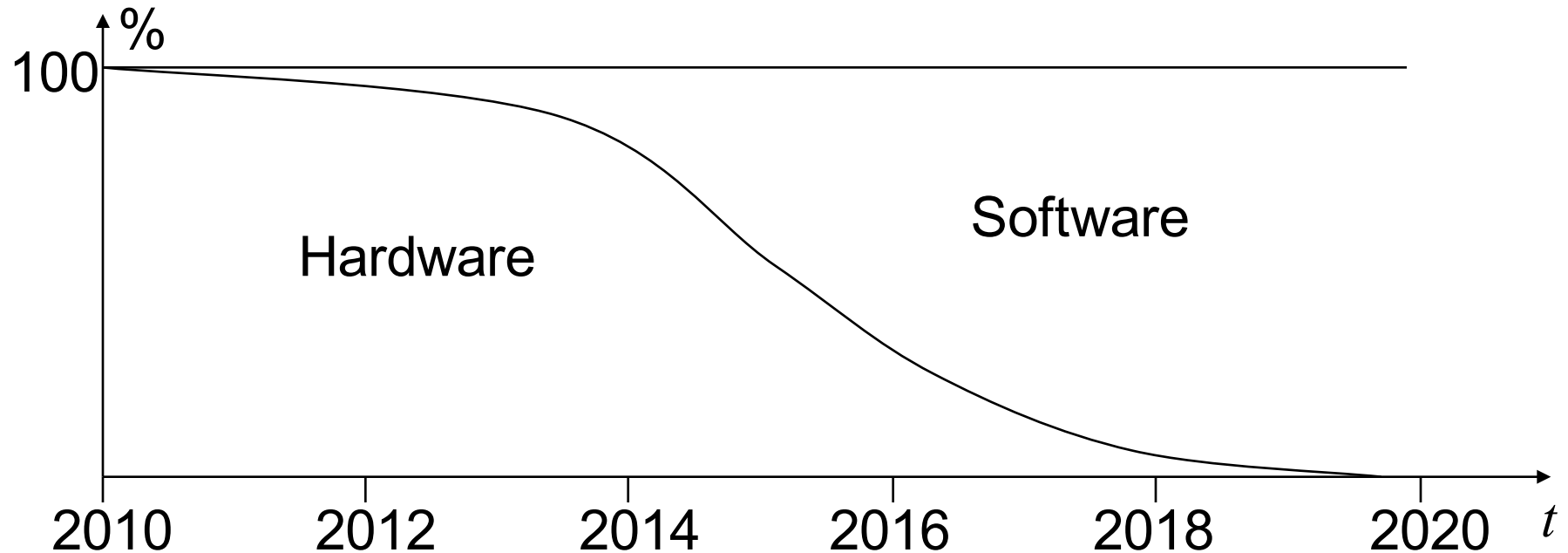
Dependent tasks, heterogeneous systems, focus on resource assignment

- **Design space exploration using genetic algorithms**

Heterogeneous systems, incl. communication modeling

# Hardware/software partitioning

No need to consider special hardware in the future?



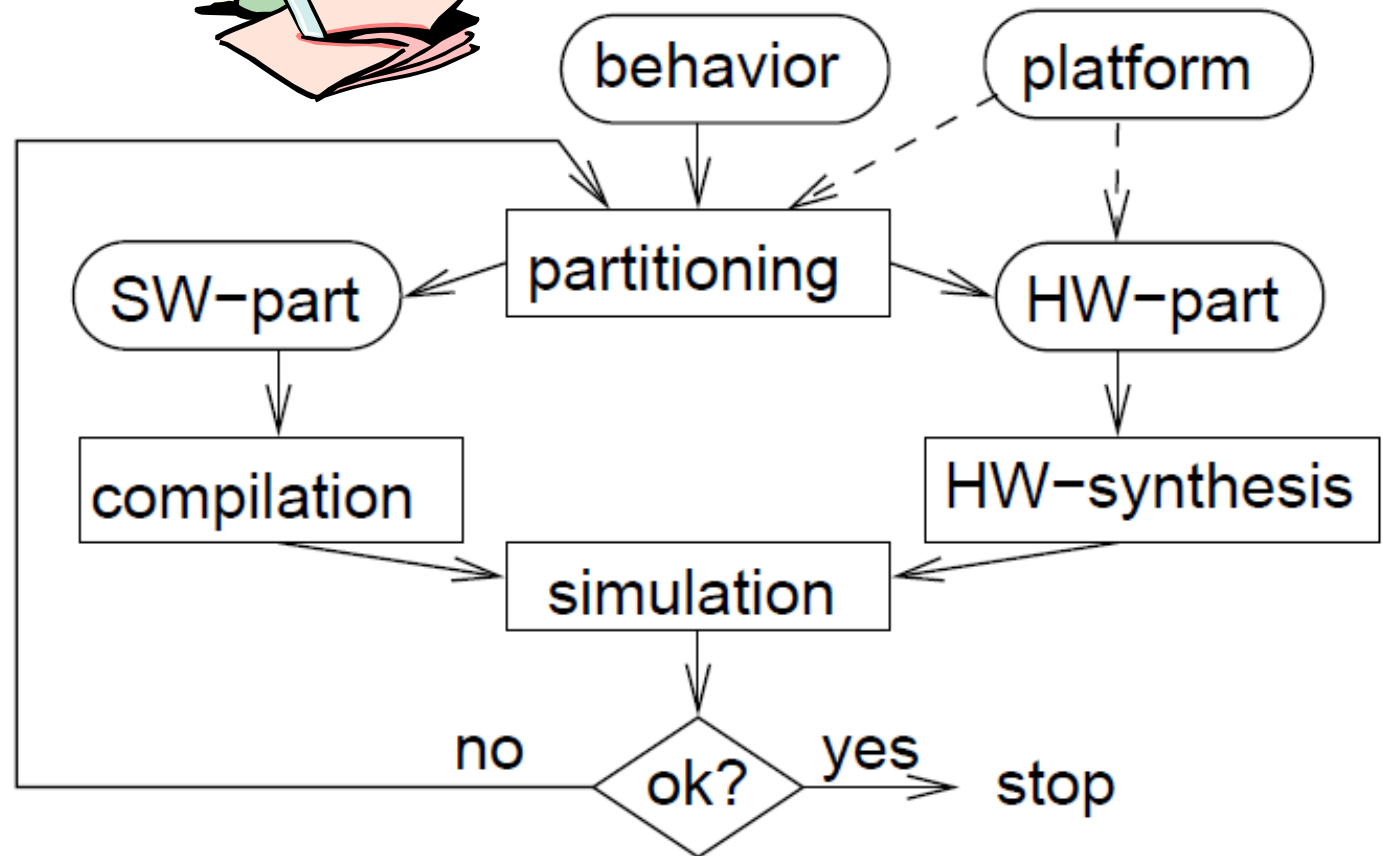
Correct for fixed functionality, but wrong in general: “By the time MPEG- $n$  can be implemented in software, MPEG- $n+1$  has been invented” [de Man]

👉 Functionality to be implemented in software or in hardware?

# Functionality to be implemented in software or in hardware?



Decision based on hardware/software partitioning, a special case of hardware/software codesign.



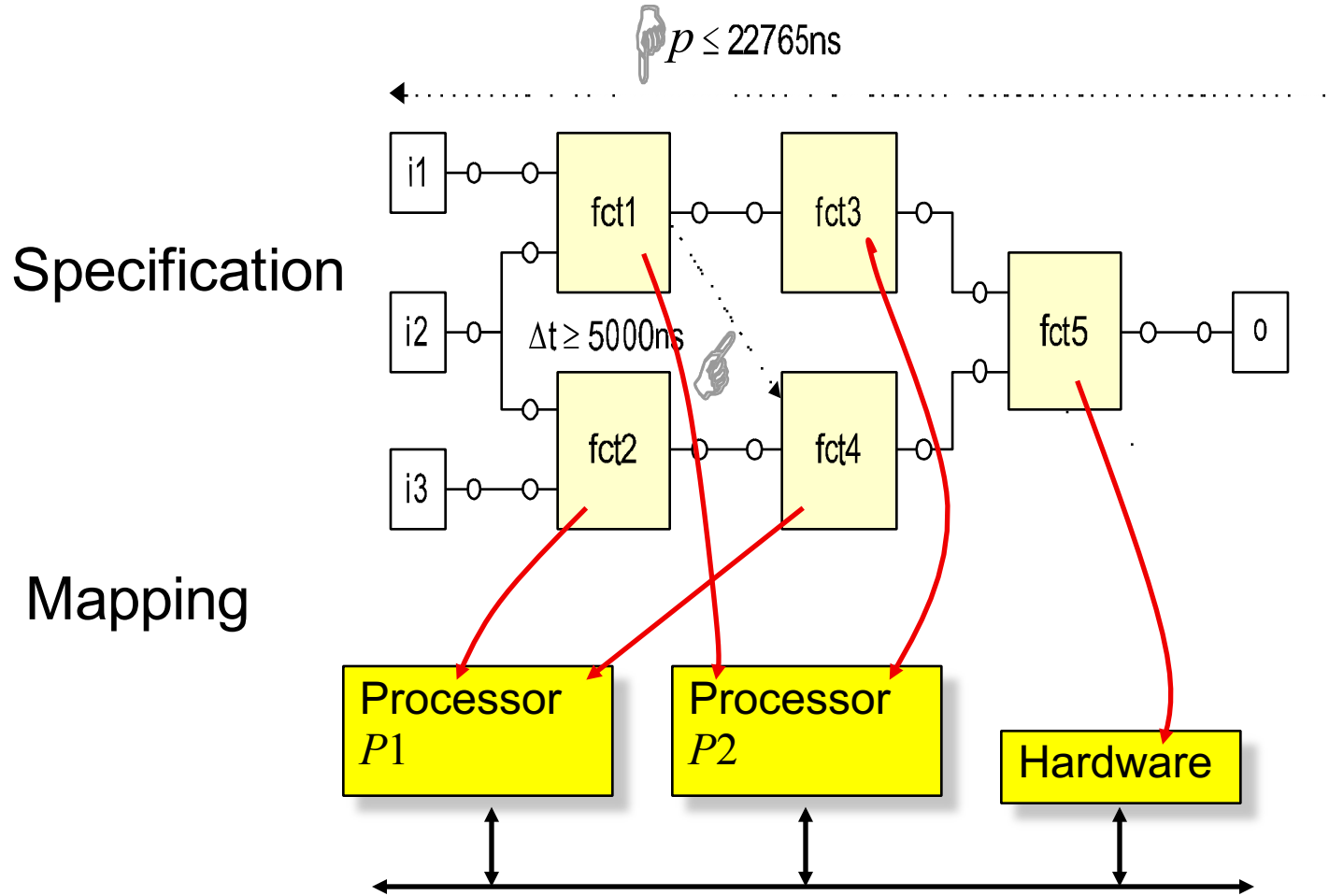
# Codesign Tool (COOL) as an example of HW/SW partitioning

---

Inputs to COOL:

1. Target technology
2. Design constraints
3. Required behavior

# Hardware/software codesign: approach



[Niemann, Hardware/Software Co-Design for Data Flow Dominated Embedded Systems, Kluwer Academic Publishers, 1998 (Comprehensive mathematical model)]

# Steps of the COOL partitioning algorithm (1)

---

- 1. Translation of the behavior into an internal graph model**
- 2. Translation of the behavior of each node from VHDL into C**
- 3. Compilation**
  - All C programs compiled for the target processor,
  - Computation of the resulting program size,
  - estimation of the resulting execution time (simulation input data might be required)
- 4. Synthesis of hardware components:**

$\forall$  leaf nodes, application-specific hardware is synthesized.  
High-level synthesis sufficiently fast.



# Steps of the COOL partitioning algorithm (2)

---

## 5. Flattening of the hierarchy:

- Granularity used by the designer is maintained.
- Cost and performance information added to the nodes.
- Precise information required for partitioning is pre-computed

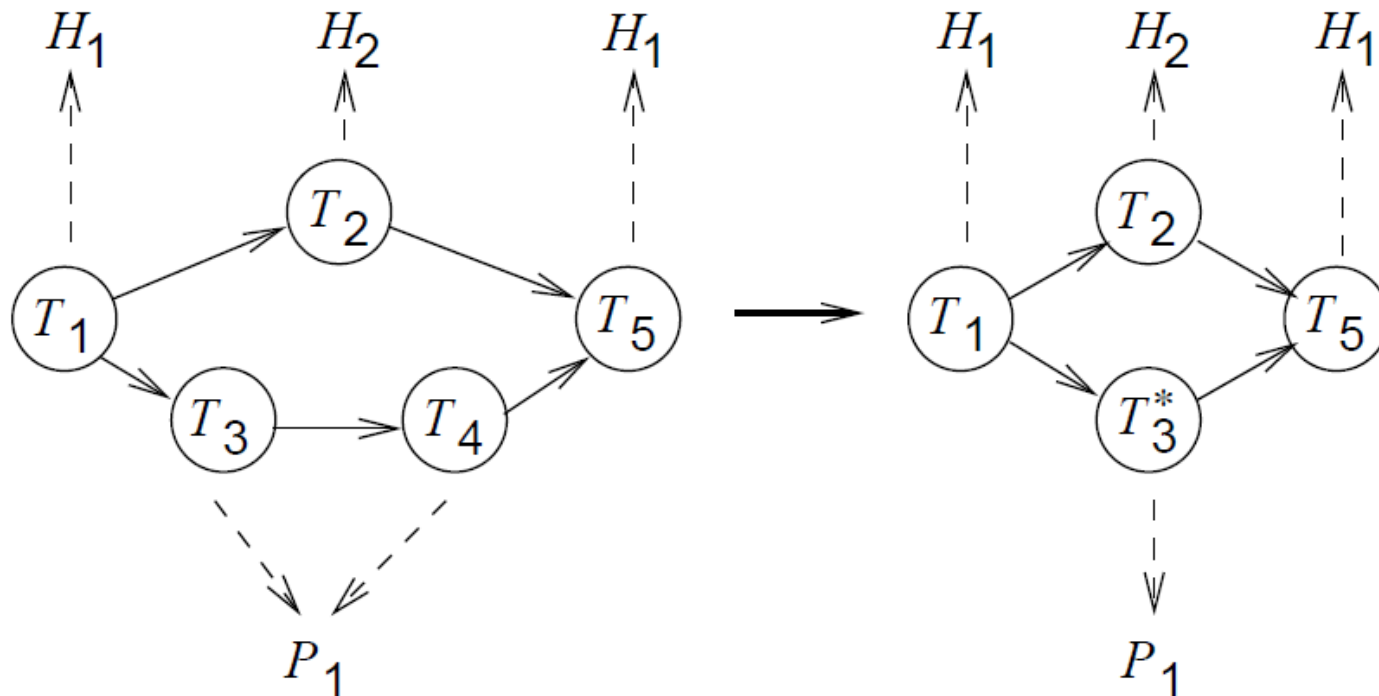
## 6. Generating and solving a mathematical model of the optimization problem:

- Integer linear programming ILP model for optimization.  
Optimal with respect to the cost function  
(approximates communication time)

# Steps of the COOL partitioning algorithm (3)

## 7. Iterative improvements:

Adjacent nodes mapped to the same hardware component are now merged.



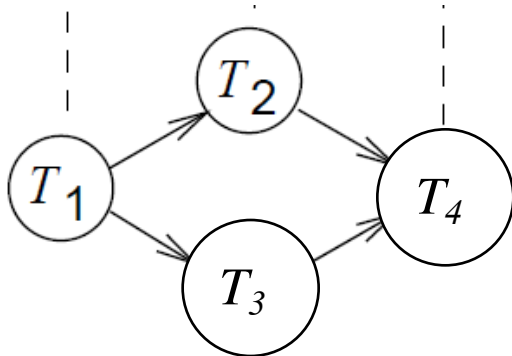
# Steps of the COOL partitioning algorithm (4)

---

## 8. Interface synthesis:

After partitioning, the glue logic required for interfacing processors, application-specific hardware and memories is created.

# Example



$T$	$\#CLBs$	$Time_{hw}$	$Time_{sw}$
1	$C_1$	$t_{1,h}$	$t_{1,s}$
2	$C_2$	$t_{2,h}$	$t_{2,s}$
3	$C_3$	$t_{3,h}$	$t_{3,s}$
4	$C_4$	$t_{4,h}$	$t_{4,s}$

## Hardware/Software Configurations

- Running on FPGA requires  $C_i$  amount of configurable logic blocks (CLBs) and results in execution time  $t_{i,h}$  (purely on FPGA)
- Running on the software (uniprocessor) requires  $t_{i,s}$  (purely on software)

*What is the minimum number of CLBs required for the task graph when the deadline is set to  $D$ ?*

# An ILP model for HW/SW partitioning

- $X_v$ : =1 if node  $v$  is mapped to FPGA and 0 otherwise.
- Cost function: minimize  $\sum_{v \in V} C_v X_v$
- Constraints:
  - Let  $F_i = t_{i,h} X_i + t_{i,s} (1 - X_i)$
  - If  $X_2 = X_3 = 0$ , then the finishing time is
    - $F_1 + F_2 + F_3 + F_4$
  - If  $X_2 = X_3 = 1$ , then the finishing time is
    - $F_1 + \max\{F_2, F_3\} + F_4$
  - If  $X_2 = 1$  and  $X_3 = 0$ , then the finishing time is
    - $F_1 + \max\{F_2, F_3\} + F_4$
  - If  $X_2 = 0$  and  $X_3 = 1$ , then the finishing time is
    - $F_1 + \max\{F_2, F_3\} + F_4$

# An ILP model for HW/SW partitioning

- $X_v$ : =1 if node  $v$  is mapped to FPGA and 0 otherwise.
- Cost function: minimize  $\sum_{v \in V} C_v X_v$
- Constraints:
  - Let  $F_i = t_{i,h} X_i + t_{i,s} (1 - X_i)$
  - If  $X_2 + X_3 = 0$ , then the finishing time is
    - $F_1 + F_2 + F_3 + F_4$
  - If  $X_2 + X_3 \geq 1$ , then the finishing time is
    - $F_1 + \max\{F_2, F_3\} + F_4$
- Logical Constraints:
  - $(X_2 \text{ OR } X_3)$  implies  $F_1 + \max\{F_2, F_3\} + F_4 \leq D$
  - $\neg(X_2 \text{ OR } X_3)$  implies  $F_1 + F_2 + F_3 + F_4 \leq D$

# Transforming Nonlinear Operation “max” (only for your reference)

---

- Method 1:  $G = \max\{F_2, F_3\}$ 
  - $F_2 \leq G$
  - $F_3 \leq G$
  - $F_1 + G + F_4 \leq D$  when ( $X_2$  OR  $X_3$ )
  
- Method 2:  $G \leq \max\{F_2, F_3\}$ 
  - Let  $f$  be a sufficiently large positive integer (i.e.,  $1000000D$ )
  - Let  $z$  be a binary variable, either 0 or 1
  - It can be formulated by using the following four linear constraints:
    - $F_2 \leq F_3 + fz$
    - $F_3 \leq F_2 + f(1-z)$
    - $G \leq F_3 + fz$
    - $G \leq F_2 + f(1-z)$

# Logical Operations “AND/OR/NOT/Implication” (only for your reference)

---

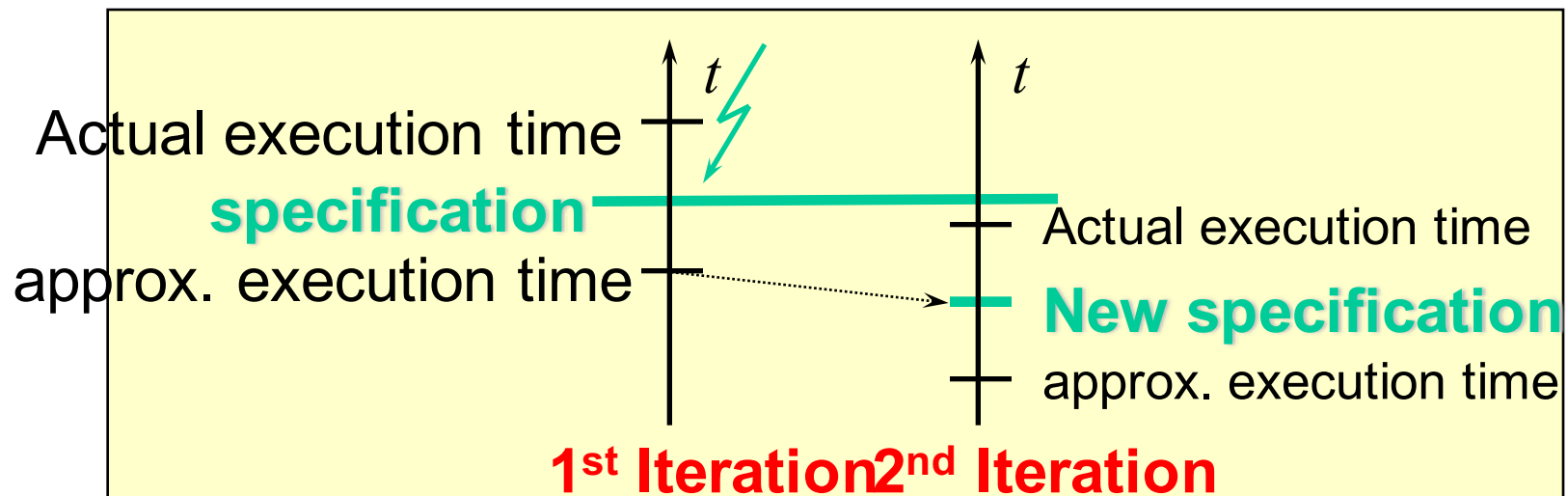
- Logical  $x_1$  AND  $x_2$ :
  - Use the linear constraints  $y_1 \geq x_1 + x_2 - 1$ ,  $y_1 \leq x_1$ ,  $y_1 \leq x_2$ ,  $0 \leq y_1 \leq 1$ , where  $y_1$  is constrained to be an integer. This enforces the desired relationship.
- Logical  $x_1$  OR  $x_2$ :
  - Use the linear constraints  $y_2 \leq x_1 + x_2$ ,  $y_2 \geq x_1$ ,  $y_2 \geq x_2$ ,  $0 \leq y_2 \leq 1$ , where  $y_2$  is constrained to be an integer.
- Logical NOT  $x_1$ :
  - Use  $y_3 = 1 - x_1$ .
- Logical implication: To express  $y_4 = (x_1 \Rightarrow x_2)$  (i.e.,  $y_4 = \neg x_1 \vee x_2$ ), we can adapt the construction for logical OR.
  - Use the linear constraints  $y_4 \leq 1 - x_1 + x_2$ ,  $y_4 \geq 1 - x_1$ ,  $y_4 \geq x_2$ ,  $0 \leq y_4 \leq 1$ , where  $y_4$  is constrained to be an integer.



# Separation of scheduling and partitioning

Combined scheduling/partitioning very complex;

- ☞ Heuristic: Compute estimated schedule
- Perform partitioning for estimated schedule
  - Perform final scheduling
  - If final schedule does not meet time constraint, go to 1 using a reduced overall timing constraint.



# HW/SW partitioning in the context of mapping applications to processors

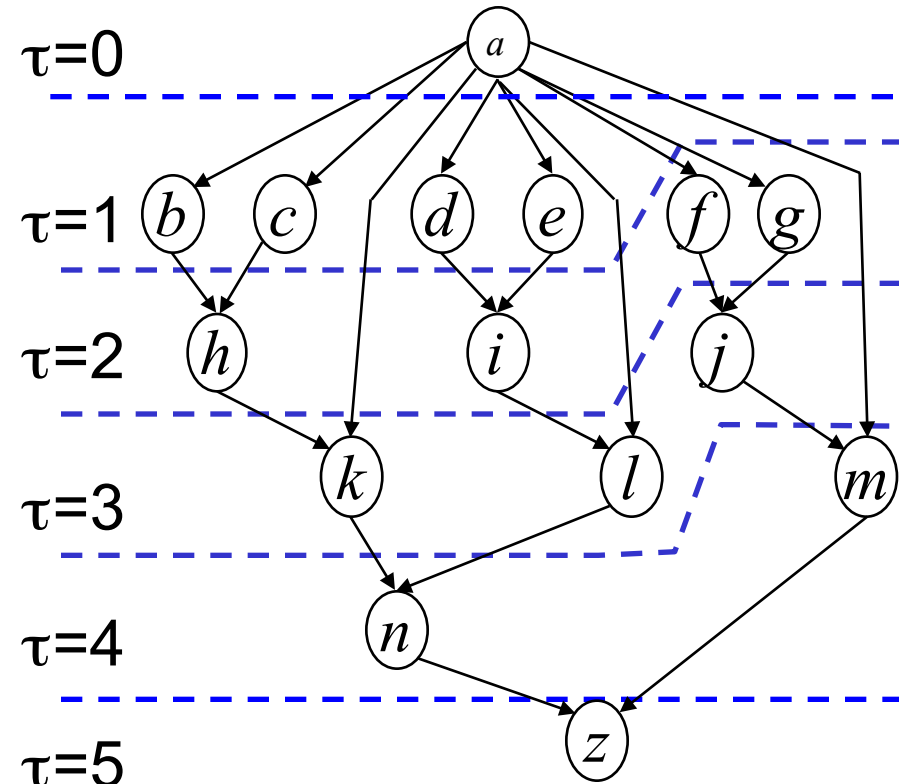
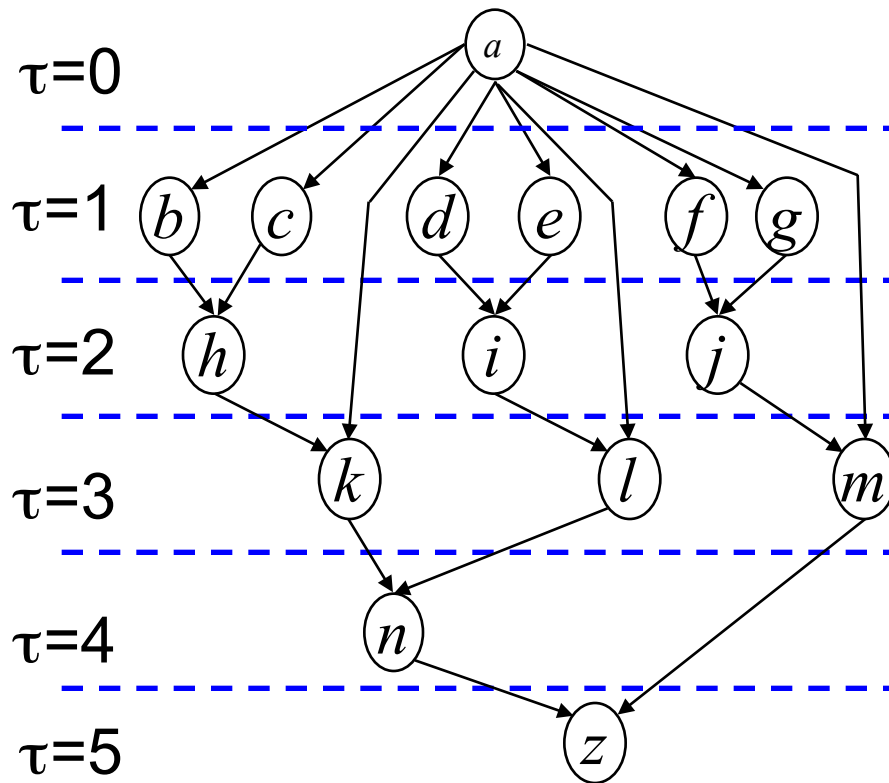
---

- Handling of heterogeneous systems
- Handling of task dependencies
- Considers of communication (at least in COOL)
- Considers memory sizes etc (at least in COOL)
- For COOL: just homogeneous processors
- No link to scheduling theory

# SPARE Slides for FDS

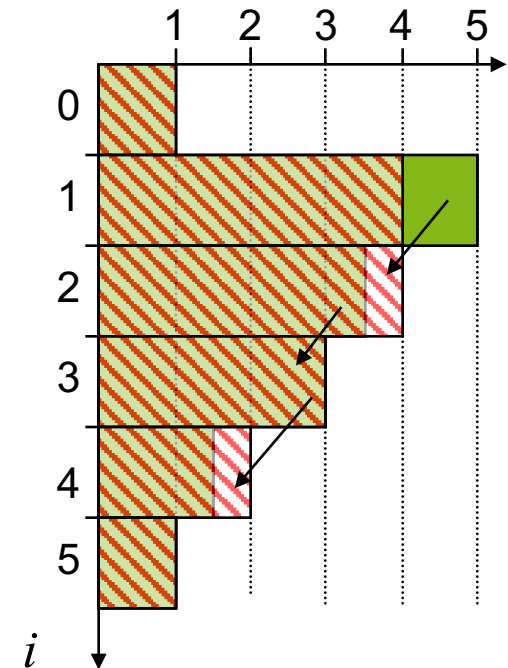
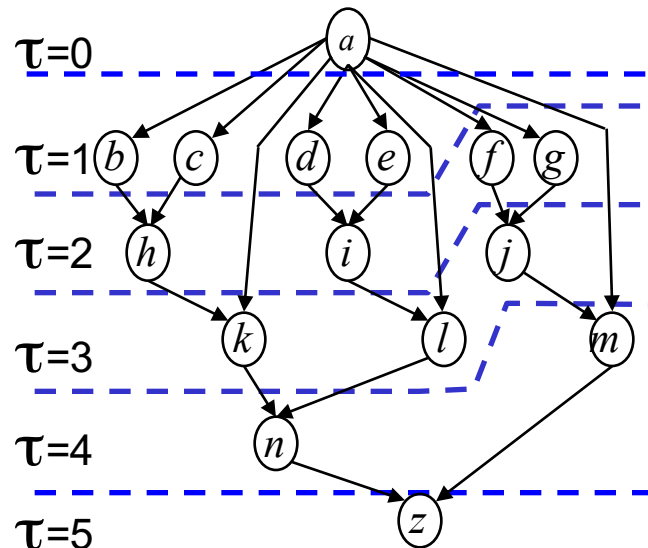
---

# Phase 1: Generation of ASAP and ALAP Schedule



# 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



# Scheduling – An example

---

Solve the differential equation

$$y'' + 3zy' + 3y = 0$$

This can be calculated using this iterative algorithm

while( $z < a$ ) repeat

$z_l := z + dz;$

$u_l := u - (3 \cdot z \cdot u \cdot dz) - (3 \cdot y \cdot dz);$

$y_l := y + (u \cdot dz);$

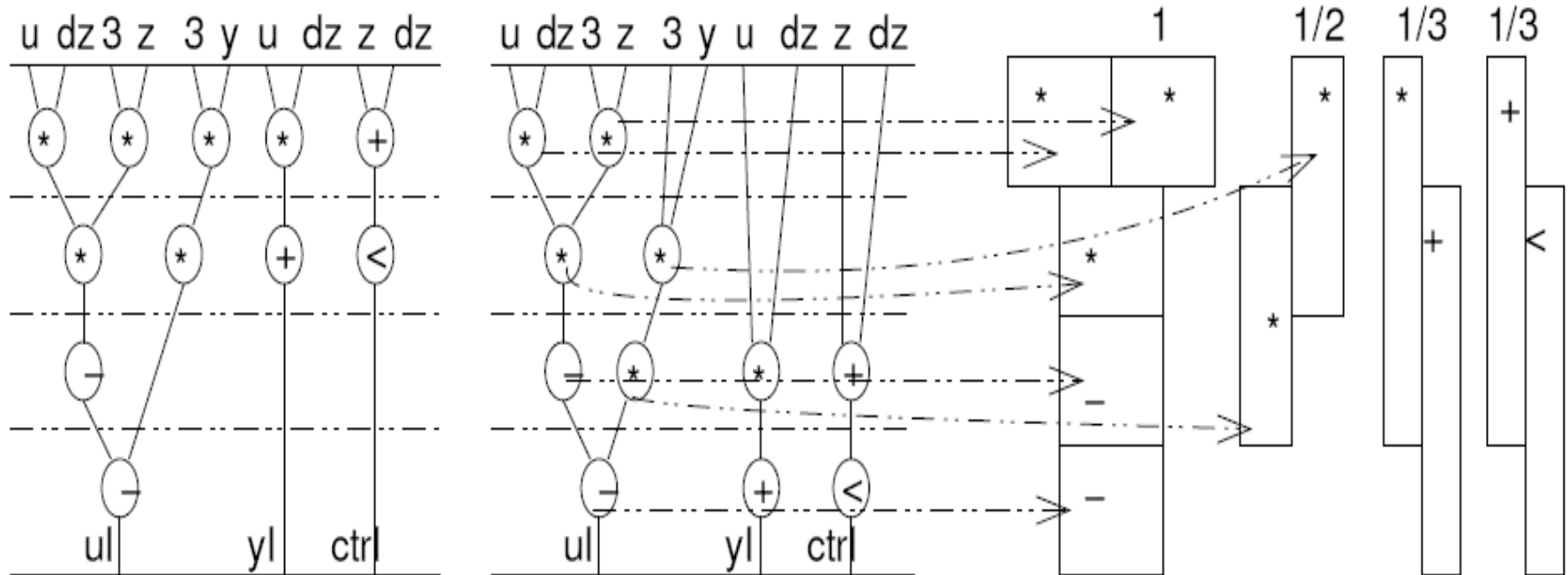
$z := z_l;$

$u := u_l;$

$y := y_l;$

1. Compute time frames  $R(j)$ ;

2. Compute “probability“  $P(j,i)$  of assignment  $j \rightarrow i$

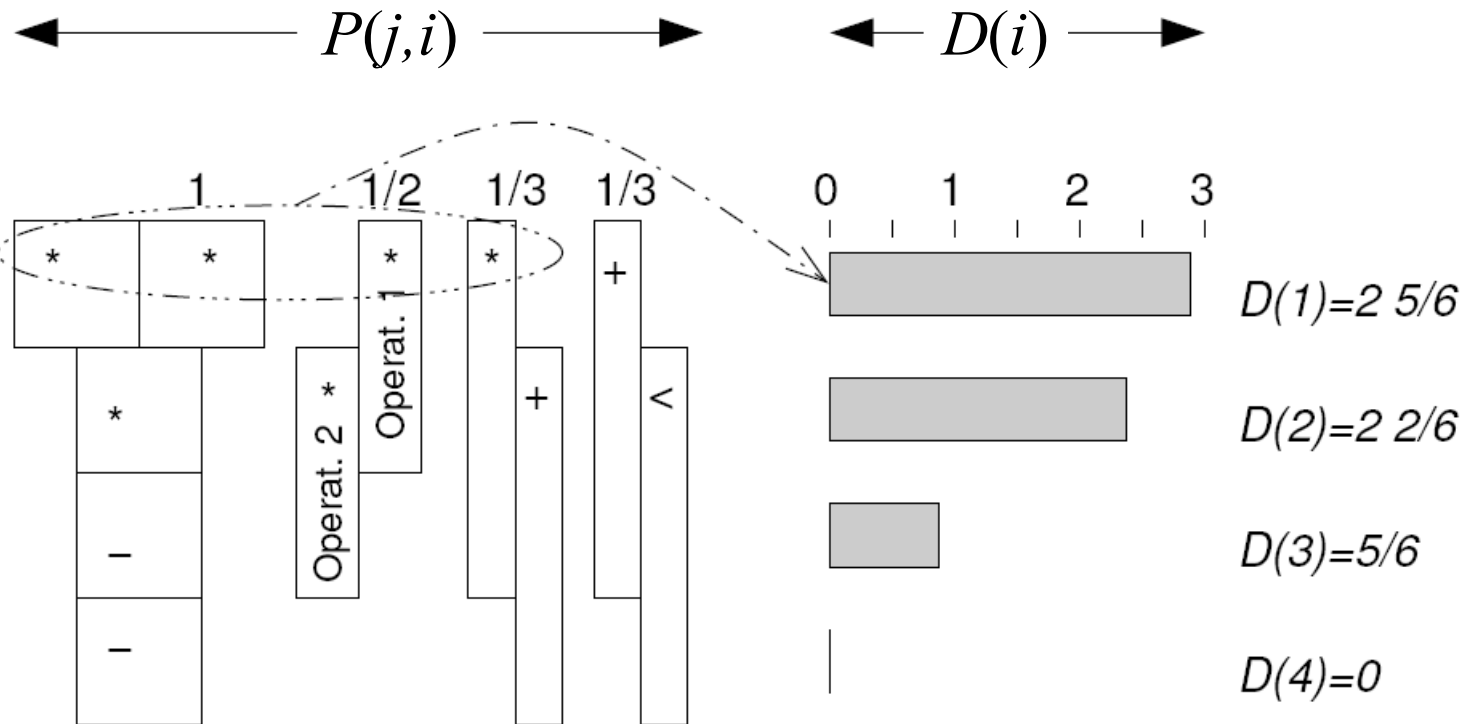


$R(j) = \{\text{ASAP-control step} \dots \text{ALAP-control step}\}$

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

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

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



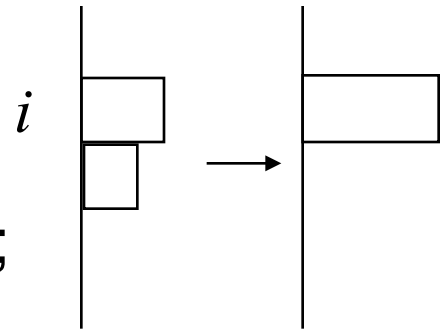


## 4. Compute direct forces (1)

- $\Delta P_i(j, i')$ :  $\Delta$  for force on task  $j$  in time step  $i'$ , if  $j$  is mapped to time step  $i$ .

The new probability for executing  $j$  in  $i$  is 1; the previous was  $P(j, i)$ .

The new probability for executing  $j$  in  $i' \neq i$  is 0; the previous was  $P(j, i')$ .



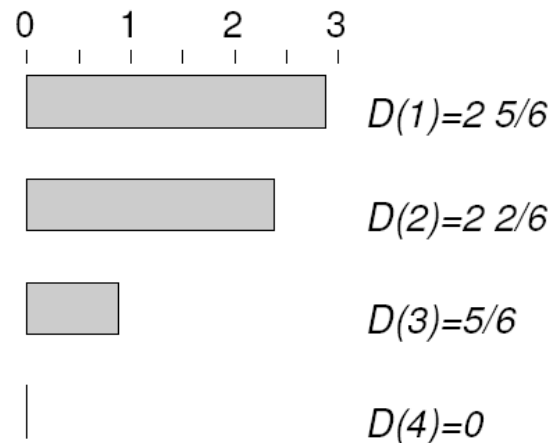
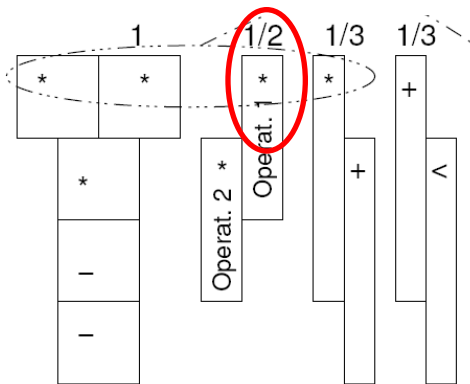
$$\text{⤵} \quad \Delta P_i(j, i') = \begin{cases} 1 - P(j, i) & \text{if } i = i' \\ -P(j, i') & \text{otherwise} \end{cases}$$

# 4. Compute direct forces (2)

- $SF(j, i)$  is the overall change of direct forces resulting from the mapping of  $j$  to time step  $i$ .

$$SF(j, i) = \sum_{i' \in R(j)} D(i') \Delta P_i(j, i') \quad \Delta P_i(j, i') = \begin{cases} 1 - P(j, i) & \text{if } i = i' \\ -P(j, i') & \text{otherwise} \end{cases}$$

## Example

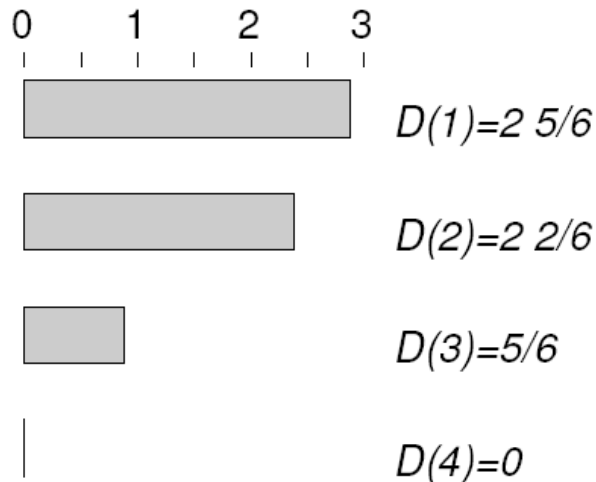
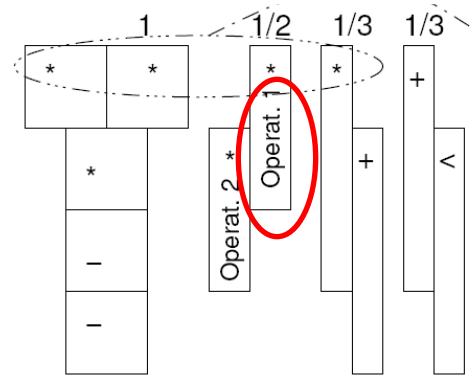


$$SF(1, 1) = 2 \frac{5}{6} (1 - \frac{1}{2}) - 2 \frac{2}{6} (\frac{1}{2}) =$$

$$\frac{1}{2} (\frac{17}{6} - \frac{14}{6}) = \frac{1}{2} (\frac{3}{6}) = \frac{1}{4}$$

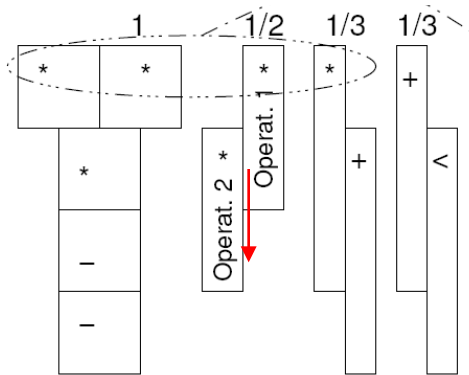
# 4. Compute direct forces (3)

Direct force if task/operation 1 is mapped to time step 2



$$\begin{aligned}
 SF(1, 2) &= D(1) * \Delta P_2(1, 1) + D(2) * \Delta P_2(1, 2) \\
 &= 2\frac{5}{6} * (-0,5) + 2\frac{2}{6} * 0.5 \\
 &= -\frac{17}{12} + \frac{14}{12} \\
 &= -\frac{3}{12} = -\frac{1}{4}
 \end{aligned}$$

# 5. Compute indirect forces (1)



Mapping task 1 to time step 2  
implies mapping task 2 to time step 3

Consider predecessor and  
successor forces:

$$VF(j, i) = \sum_{j' \in \text{predecessor of } j} \sum_{i' \in I} D(i') \Delta P_{j,i}(j', i')$$

$$NF(j, i) = \sum_{j' \in \text{successor of } j} \sum_{i' \in I} D(i') \Delta P_{j,i}(j', i')$$

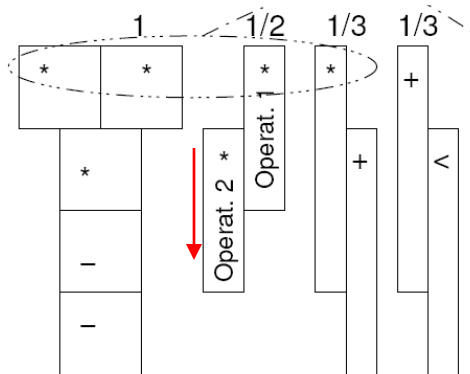
$\Delta P_{j,i}(j', i')$  is the  $\Delta$  in the probability of mapping  $j'$  to  $i'$   
resulting from the mapping of  $j$  to  $i$

# 5. Compute indirect forces (2)

$$VF(j, i) = \sum_{j' \in \text{predecessor of } j} \sum_{i' \in I} D(i') \Delta P_{j,i}(j', i')$$

$$NF(j, i) = \sum_{j' \in \text{successor of } j} \sum_{i' \in I} D(i') \Delta P_{j,i}(j', i')$$

Example: Computation of successor forces for task 1 in time step 2



$$\begin{aligned}
 NF(1, 2) &= D(2) * \Delta P_{1,2}(2, 2) + D(3) * \Delta P_{1,2}(2, 3) \\
 &= 2\frac{2}{6} * (-0,5) + \frac{5}{6} * 0.5 \\
 &= -\frac{14}{12} + \frac{5}{12} \\
 &= -\frac{9}{12} = -\frac{3}{4}
 \end{aligned}$$

# Overall forces

---

The total force is the sum of direct and indirect forces:

$$F(j, i) = SF(j, i) + VF(j, i) + NF(j, i)$$

In the example:

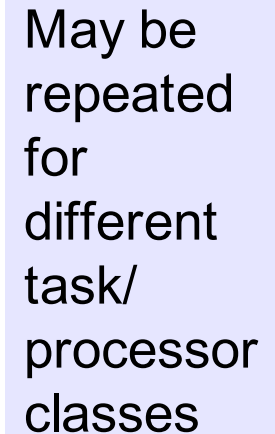
$$F(1, 2) = SF(1, 2) + NF(1, 2) = -\frac{1}{4} + \left(-\frac{3}{4}\right) = -1$$

The low value suggests mapping task 1 to time step 2

# 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

# SPARE Slides for COOL

---



# An integer linear programming (ILP) model for HW/SW partitioning

---

## Notation:

- Index set  $V$  denotes task graph nodes.
- Index set  $L$  denotes task graph node **types**  
e.g. square root, DCT or FFT
- Index set  $M$  denotes hardware component **types**.  
e.g. hardware components for the DCT or the FFT.
- Index set  $J$  of hardware component instances
- Index set  $KP$  denotes processors.  
All processors are assumed to be of the same type

# An ILP model for HW/SW partitioning

---

- $X_{v,m}$ : =1 if node  $v$  is mapped to hardware component type  $m \in M$  and 0 otherwise.
- $Y_{v,k}$ : =1 if node  $v$  is mapped to processor  $k \in KP$  and 0 otherwise.
- $NY_{l,k}$  =1 if at least one node of type  $l$  is mapped to processor  $k \in KP$  and 0 otherwise.
- *Type* is a mapping from task graph nodes to their types:  
 $Type : V \rightarrow L$
- The cost function accumulates the cost of hardware units:  
$$C = \text{cost}(\text{processors}) + \text{cost}(\text{memories}) + \text{cost}(\text{application specific hardware})$$

# Constraints

---

## Operation assignment constraints

$$\forall v \in V : \sum_{m \in M} X_{v,m} + \sum_{k \in KP} Y_{v,k} = 1$$

All task graph nodes have to be mapped either in software or in hardware.

Variables are assumed to be integers.

Additional constraints to guarantee they are either 0 or 1:

$$\forall v \in V : \forall m \in M : X_{v,m} \leq 1$$

$$\forall v \in V : \forall k \in KP : Y_{v,k} \leq 1$$

## Operation assignment constraints (2)

---

$$\forall l \in L, \forall v: \text{Type}(v)=c_l, \forall k \in KP : NY_{l,k} \geq Y_{v,k}$$

For all types  $l$  of operations and for all nodes  $v$  of this type: if  $v$  is mapped to some processor  $k$ , then that processor must implement the functionality of  $l$ .

Decision variables must also be 0/1 variables:

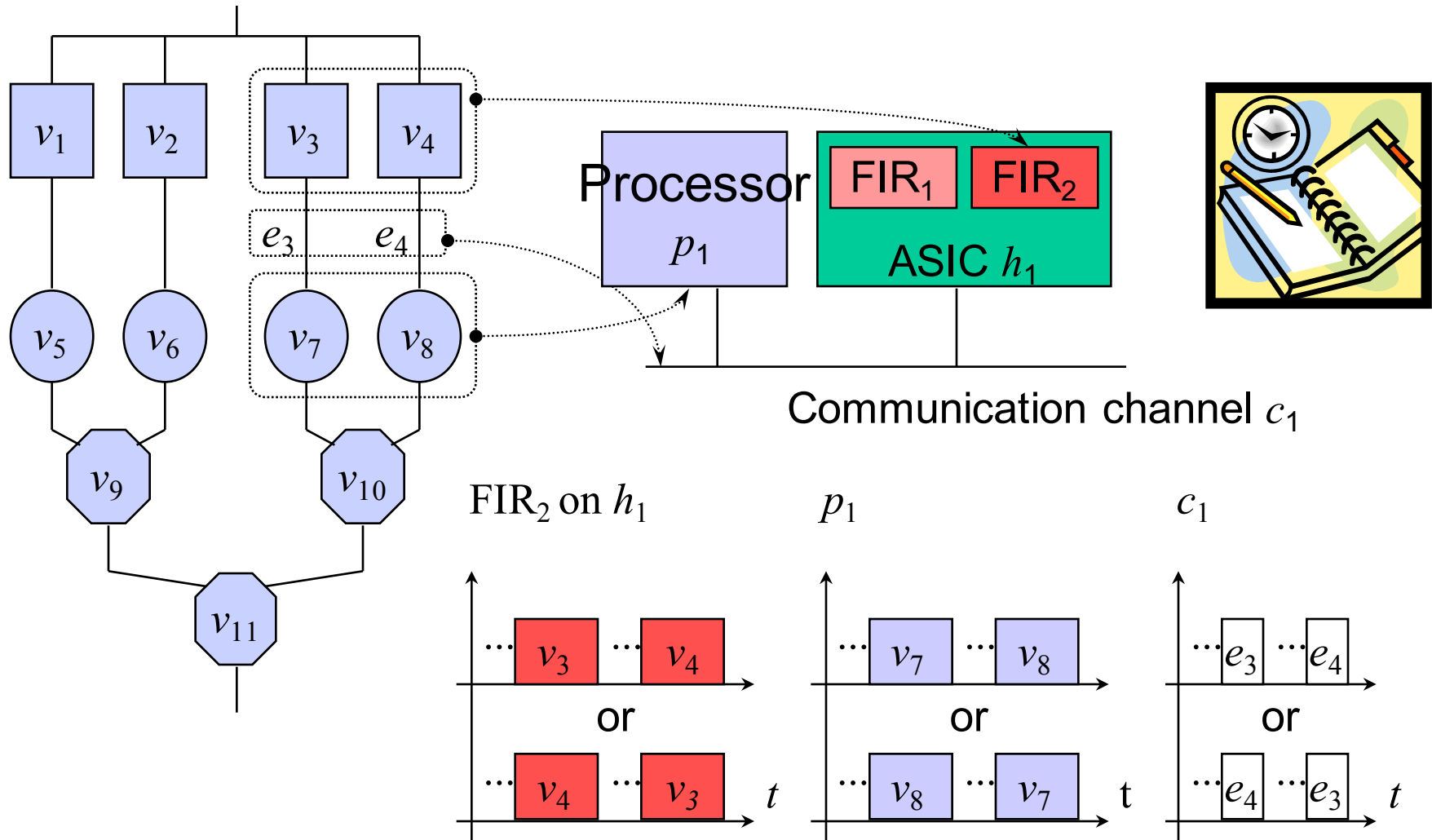
$$\forall l \in L, \forall k \in KP : NY_{l,k} \leq 1.$$

# Resource & design constraints

---

- $\forall m \in M$ , the cost (area) for components of type  $m$  is = sum of the costs of the components of that type. This cost should not exceed its maximum.
- $\forall k \in KP$ , the cost for associated data storage area should not exceed its maximum.
- $\forall k \in KP$  the cost for storing instructions should not exceed its maximum.
- The total cost ( $\sum_{m \in M}$ ) of HW components should not exceed its maximum
- The total cost of data memories ( $\sum_{k \in KP}$ ) should not exceed its maximum
- The total cost instruction memories ( $\sum_{k \in KP}$ ) should not exceed its maximum

# Scheduling



# Scheduling / precedence constraints

---

- For all nodes  $v_{i1}$  and  $v_{i2}$  that are potentially mapped to the same processor or hardware component instance, introduce a binary decision variable  $b_{i1,i2}$  with  $b_{i1,i2}=1$  if  $v_{i1}$  is executed before  $v_{i2}$  and  $= 0$  otherwise.

Define constraints of the type

(end-time of  $v_{i1}$ )  $\leq$  (start time of  $v_{i2}$ ) if  $b_{i1,i2}=1$  and

(end-time of  $v_{i2}$ )  $\leq$  (start time of  $v_{i1}$ ) if  $b_{i1,i2}=0$

- Ensure that the schedule for executing operations is consistent with the precedence constraints in the task graph.
- Approach fixes the order of execution

# Other constraints

---

- **Timing constraints**

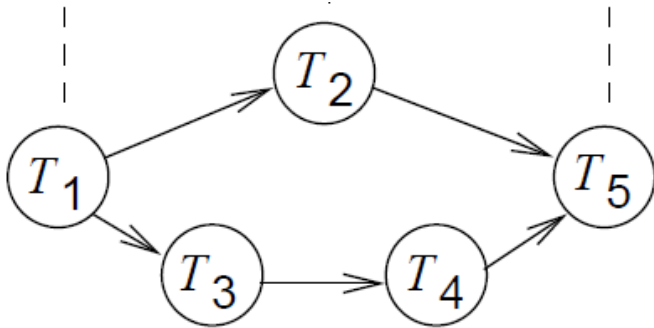
These constraints can be used to guarantee that certain time constraints are met.

- Some less important constraints omitted ..





# Example



HW types  $H1$ ,  $H2$  and  $H3$  with costs of 20, 25, and 30.

Processors of type  $P$ .

Tasks  $T1$  to  $T5$ .

Execution times:

$T$	$H1$	$H2$	$H3$	$P$
1	20			100
2		20		100
3			12	10
4			12	10
5	20			100

# Operation assignment constraints (1)

$T$	$H1$	$H2$	$H3$	$P$
1	20			100
2		20		100
3			12	10
4			12	10
5	20			100

$$\forall v \in V: \sum_{m \in KM} X_{v,m} + \sum_{k \in KP} Y_{v,k} = 1$$

$$X_{1,1} + Y_{1,1} = 1 \text{ (task 1 mapped to } H1 \text{ or to } P)$$

$$X_{2,2} + Y_{2,1} = 1$$

$$X_{3,3} + Y_{3,1} = 1$$

$$X_{4,3} + Y_{4,1} = 1$$

$$X_{5,1} + Y_{5,1} = 1$$

## Operation assignment constraints (2)

Assume types of tasks are  $l = 1, 2, 3, 3,$  and  $1$ .

$$\forall l \in L, \forall v: \text{Type}(v)=c_l, \forall k \in KP : NY_{l,k} \geq Y_{v,k}$$

$$NY_{1,1} \geq Y_{1,1}$$

$$NY_{2,1} \geq Y_{2,1}$$

$$NY_{3,1} \geq Y_{3,1}$$

$$NY_{3,1} \geq Y_{4,1}$$

$$NY_{1,1} \geq Y_{5,1}$$

Functionality 3 to be implemented on processor if node 4 is mapped to it.

# Other equations

---

Time constraints leading to: Application specific hardware required for time constraints  $\leq 100$  time units.

<i>T</i>	<i>H1</i>	<i>H2</i>	<i>H3</i>	<i>P</i>
1	20			100
2		20		100
3			12	10
4			12	10
5	20			100

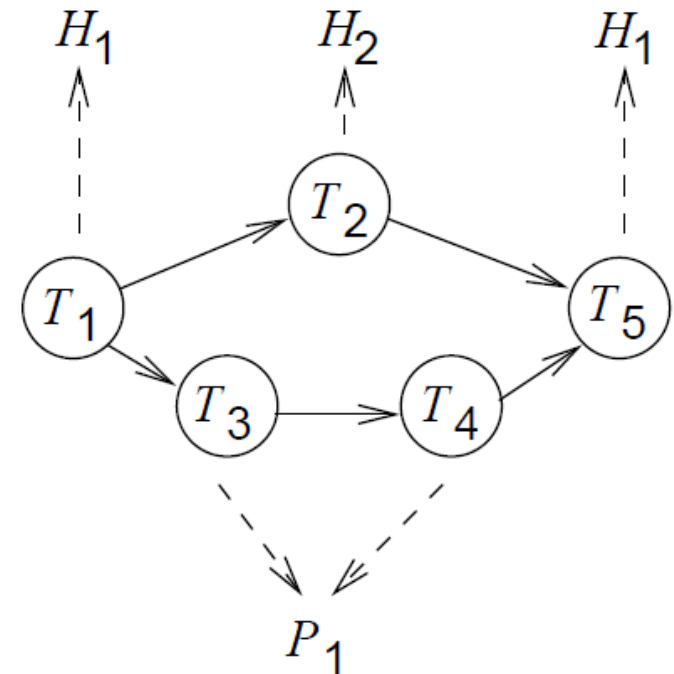
Cost function:

$$C = 20 \#(H1) + 25 \#(H2) + 30 \#(H3) + \text{cost}(\text{processor}) + \text{cost}(\text{memory})$$

# Result

For a time constraint of 100 time units  
and  $\text{cost}(P) < \text{cost}(H3)$ :

$T$	$H1$	$H2$	$H3$	$P$
1	20			100
2		20		100
3			12	10
4			12	10
5	20			100



Solution (educated guessing) :

$T1 \rightarrow H1$

$T2 \rightarrow H2$

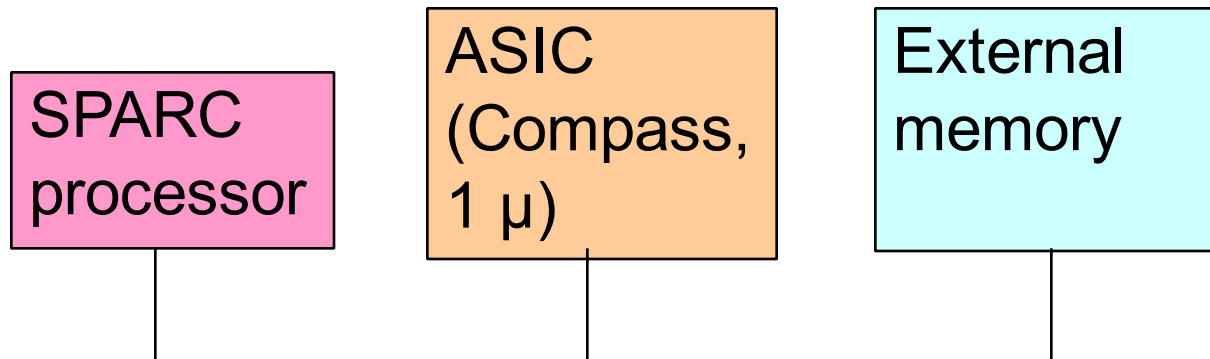
$T3 \rightarrow P$

$T4 \rightarrow P$

$T5 \rightarrow H1$

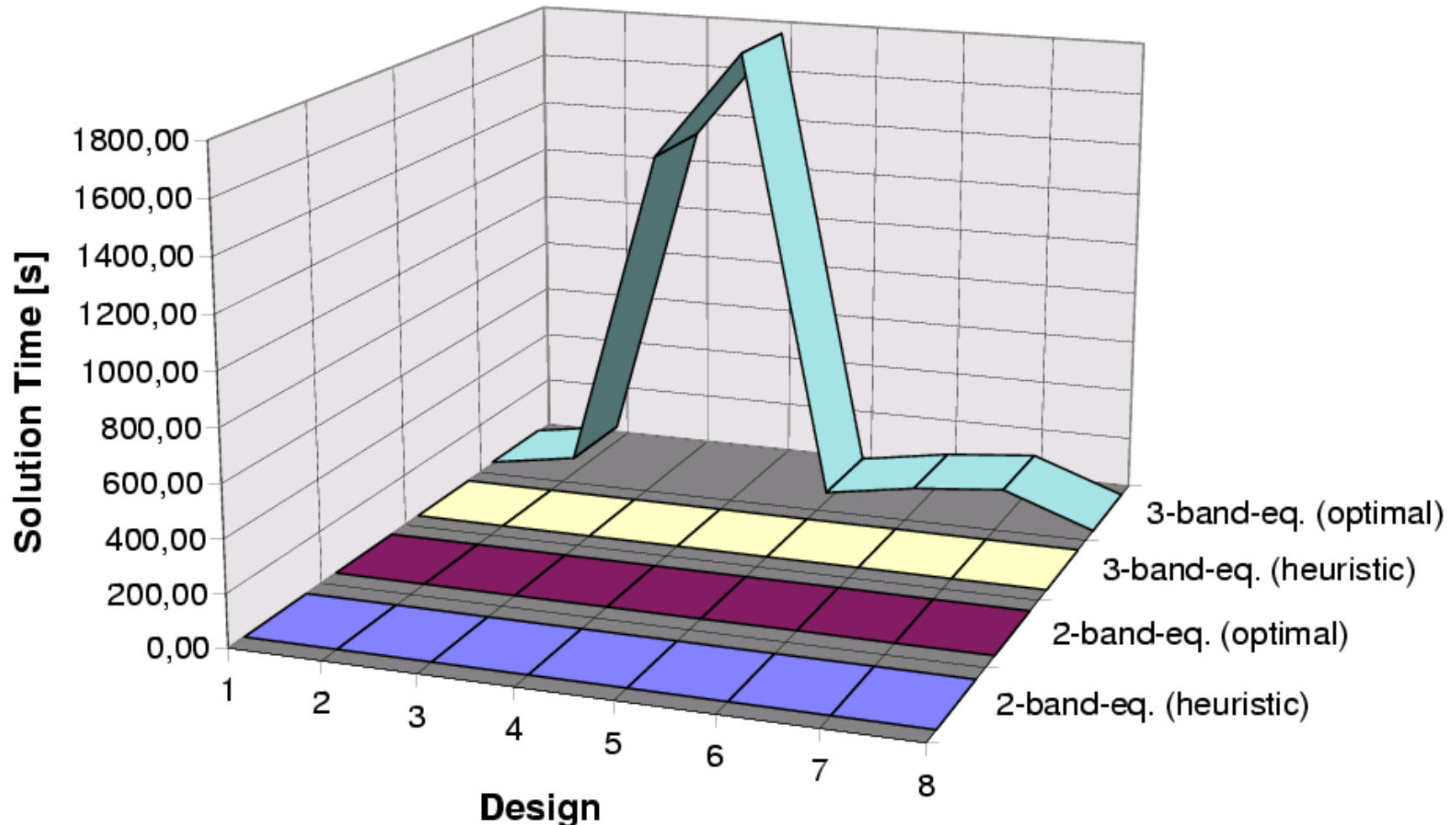
# Application example

Audio lab (mixer, fader, echo, equalizer,  
balance units); slow SPARC processor  
1 $\mu$  ASIC library  
Allowable delay of 22.675  $\mu$ s ( $\sim$  44.1 kHz)



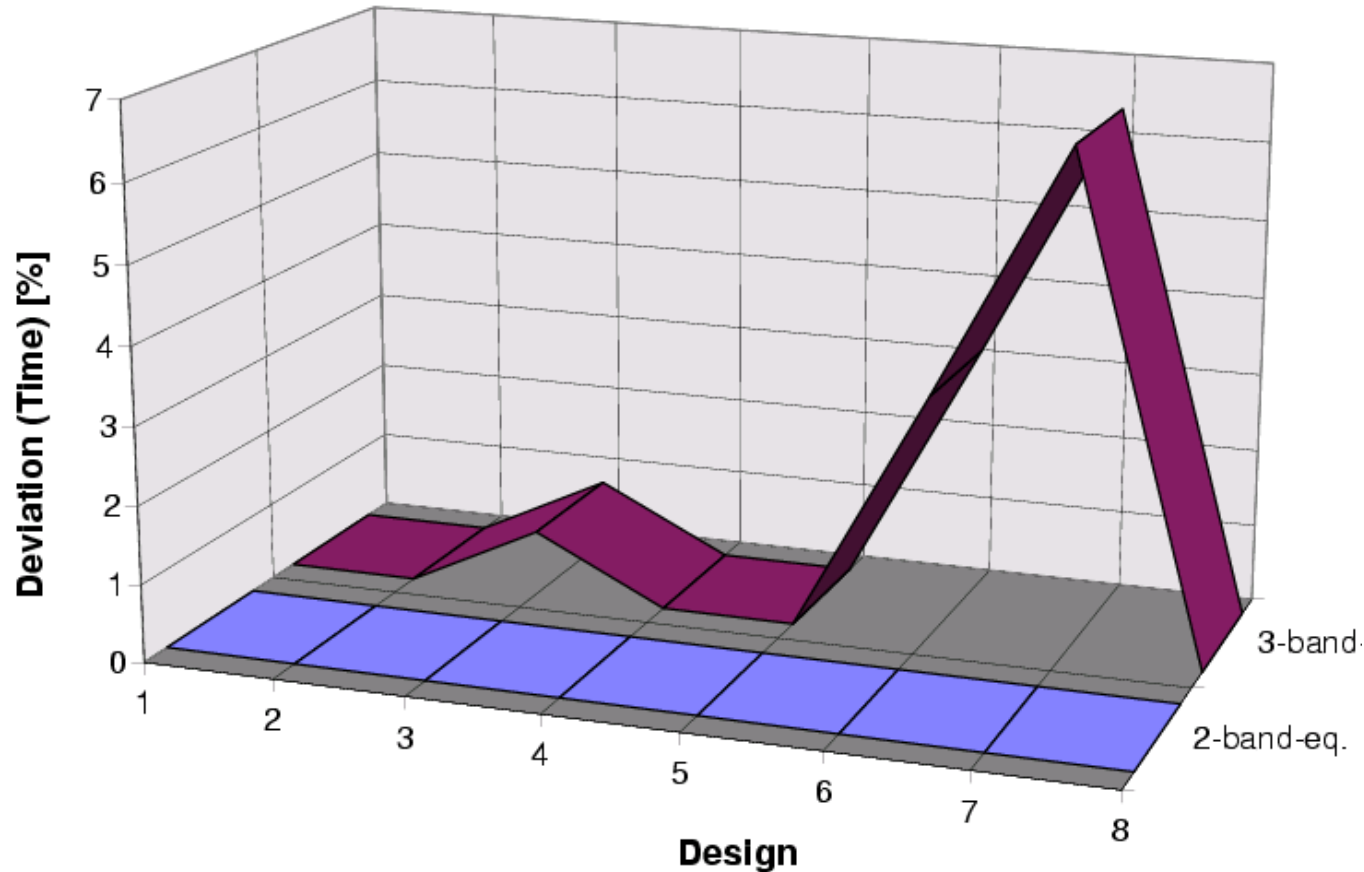
Outdated technology; just a proof of concept.

# Running time for COOL optimization



☞ Only simple models can be solved optimally.

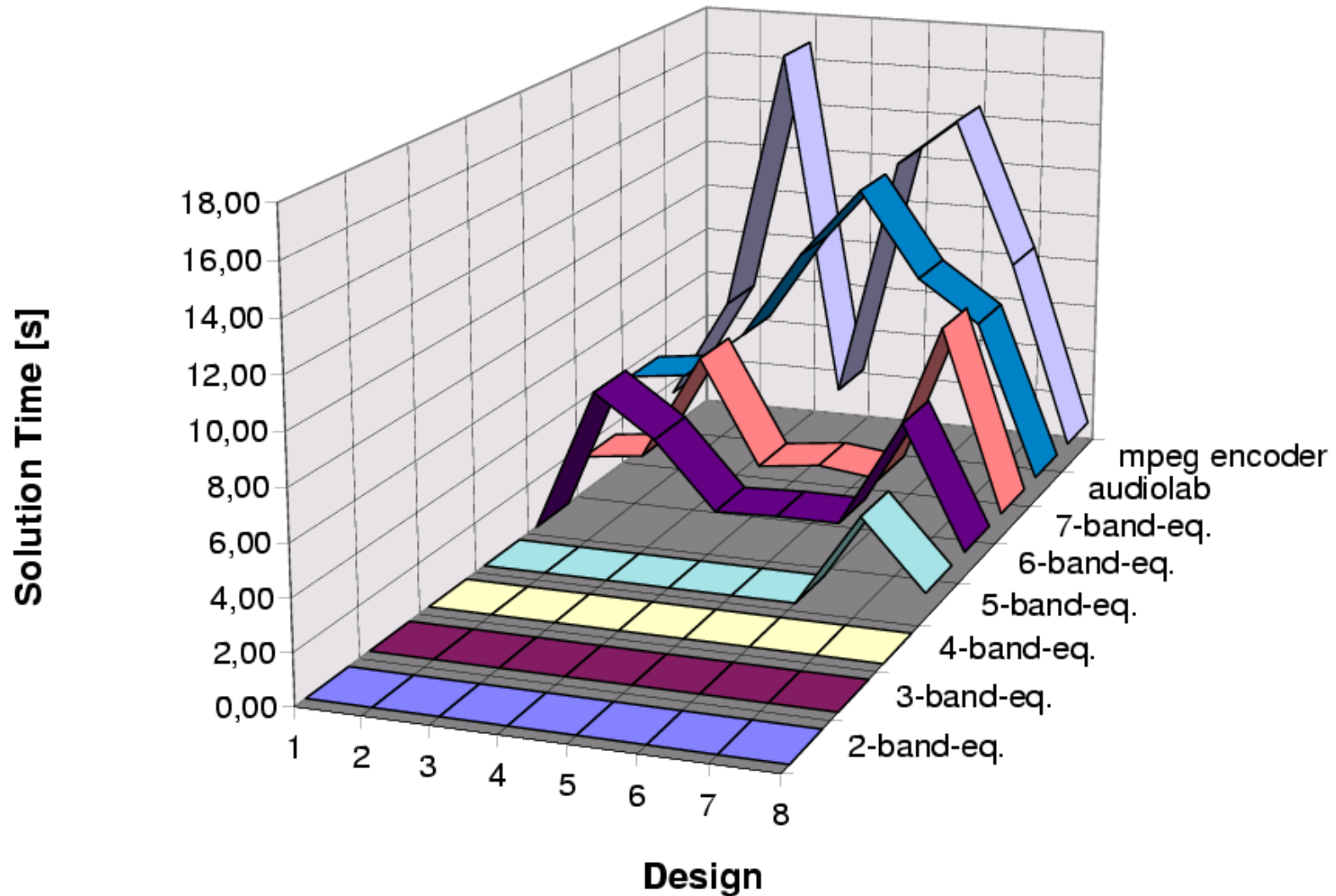
# Deviation from optimal design



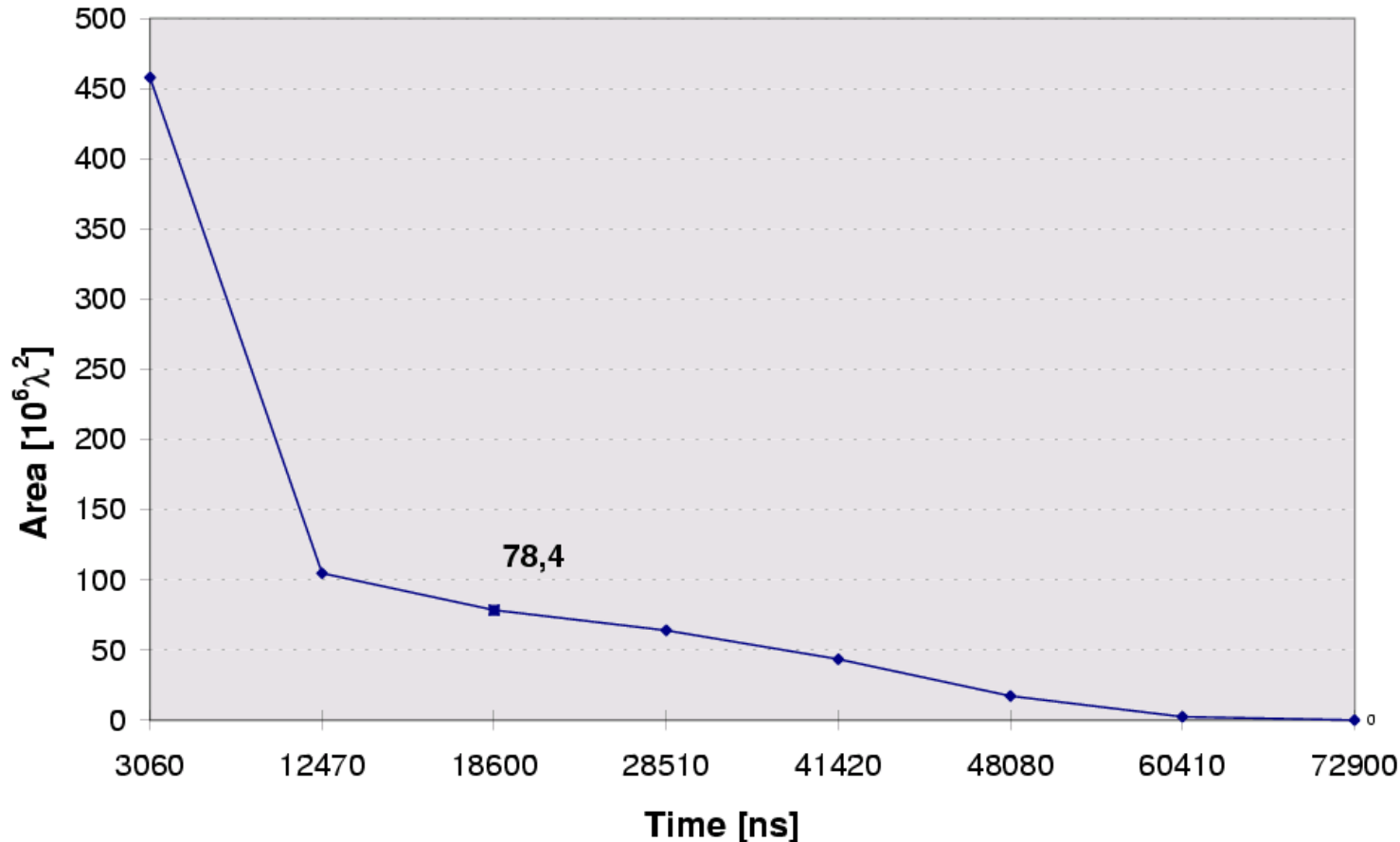
👉 Hardly any loss in design quality.



# Running time for heuristic



# Design space for audio lab



Everything in software: 72.9 μs, 0 λ<sup>2</sup>  
Everything in hardware: 3.06 μs, 457.9x10<sup>6</sup> λ<sup>2</sup>  
Lowest cost for given sample rate: 18.6 μs, 78.4x10<sup>6</sup> λ<sup>2</sup>,

# Positioning of COOL

---

COOL approach:

- shows that a formal model of hardware/SW codesign is beneficial; IP modeling can lead to useful implementation even if optimal result is available only for small designs.

Other approaches for HW/SW partitioning:

- starting with everything mapped to hardware; gradually moving to software as long as timing constraint is met.
- starting with everything mapped to software; gradually moving to hardware until timing constraint is met.
- Binary search.