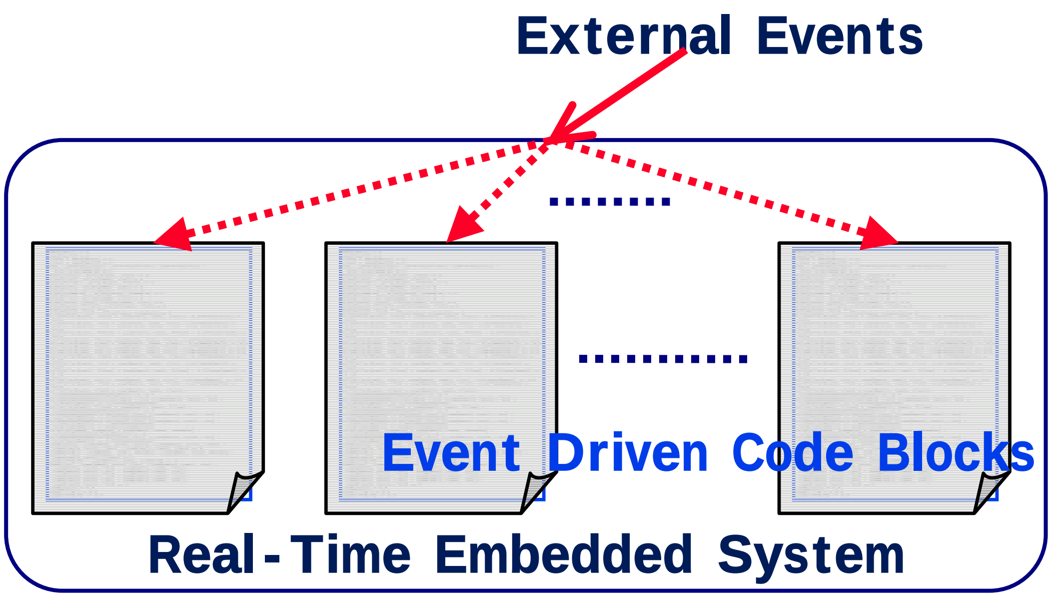
On the Complexity of Scheduling Conditional Real-Time Code

Samarjit Chakraborty

Joint work with Thomas Erlebach and Lothar Thiele

Introduction

- **Real-Time Embedded Systems:**
  - **Collection of independently executing code blocks**
  - **Triggered by external events**
  - **Hard deadlines**



Scheduling

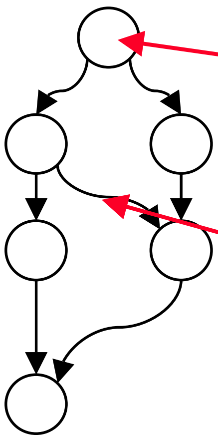
- Given the specifications of the code blocks
  - Execution requirements, deadlines, control flow,…
- And the specifications of the external events
  - Minimum separation time, for example

Feasibility Analysis

Is it possible to schedule all the code blocks,
under all possible triggering sequences,
such that all the associated deadlines are met?

Representing Code Blocks

- Directed Acyclic Graph
  - Vertices represent code with known execution time
  - Edges represent conditional branches



Code Block (Event handler)
with known execution time and deadline

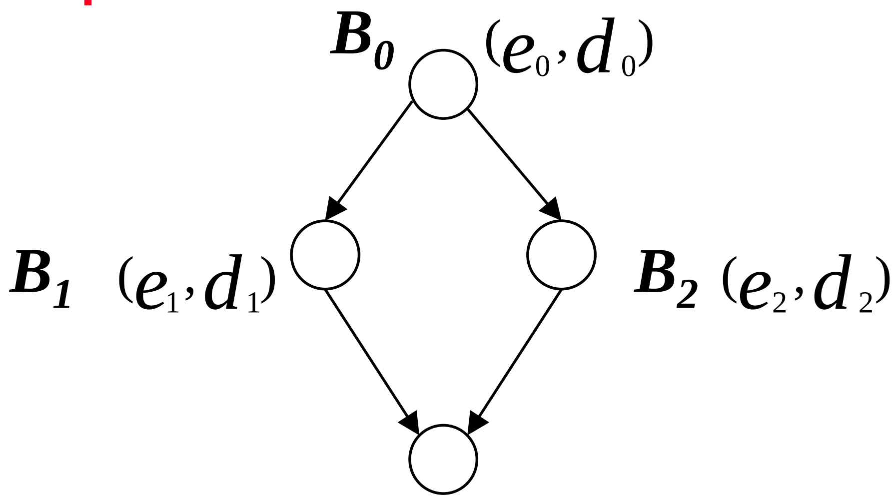
Conditional
Branches

> **Is it possible to schedule a collection of such graphs to meet all deadlines?**

Conditional Real-Time Code

Can not be evaluated at compile time

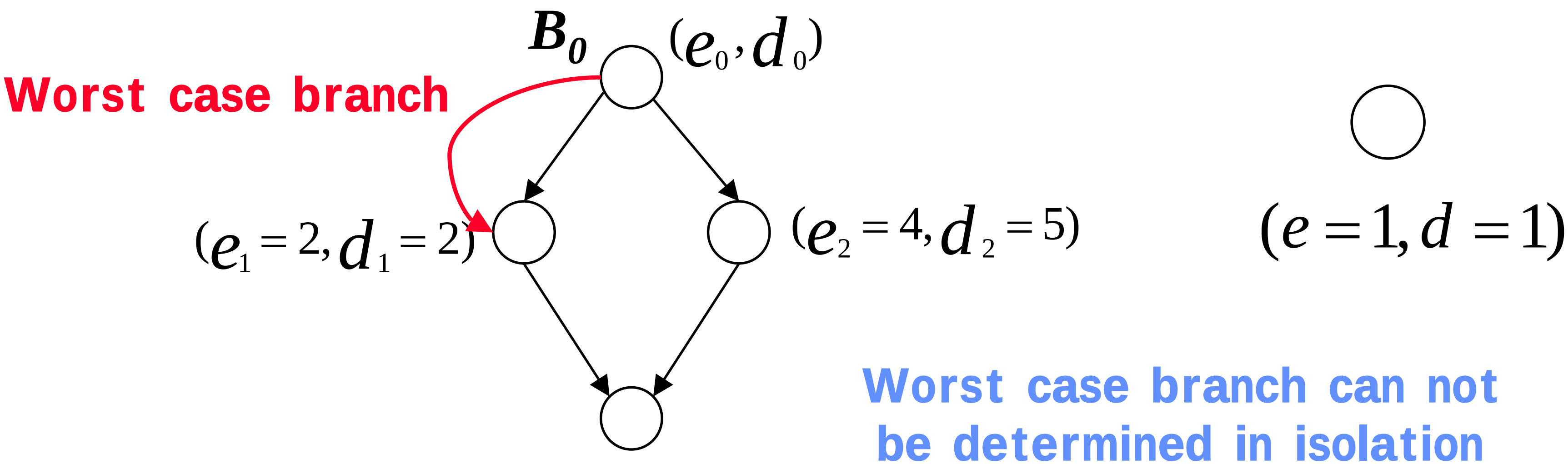
while (*external event*) **do**
    execute code block B_0
    if (*condition C*) **then**
        execute code block B_1
    else
        execute code block B_2
    endif
endwhile



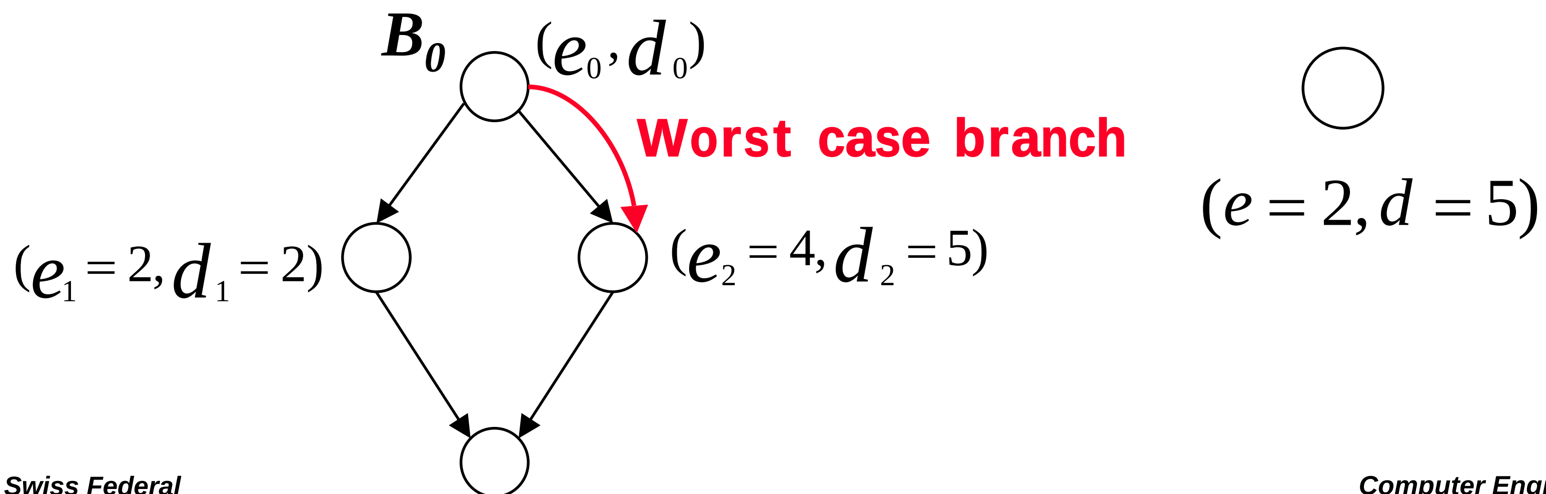
Code Block

Corresponding Graph

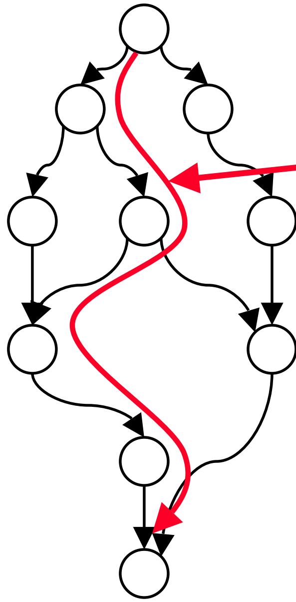
Scheduling Conditional Real-Time Code



Worst case branch can not be determined in isolation



Scheduling Conditional Real-Time Code



Worst ~~case~~ path

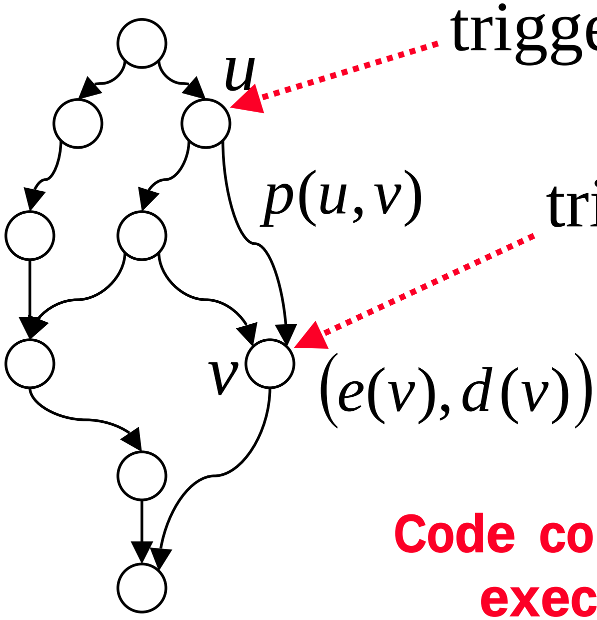
Usual method for feasibility analysis:
Approximating a piece of code by its worst case behaviour, does not work in the presence of such conditional branches

Model

- **Event driven conditional real-time code**

triggered at time t_u

triggered at time $t_v \geq t_u + p(u,v)$



Code corresponding to vertex v must be executed within the time interval $\left(t_v, t_v + d(v)\right]$

Task graph T

Model

- **Task set** $\tau = \{T_1, T_2, \ldots\ldots, T_k\}$
- **Vertices of the task graphs get triggered independently of each other**
- **Legal triggering sequence of τ is formed by merging together sequences from $\{T_1, \ldots\ldots\ldots, T_k\}$**

Feasibility analysis of τ :

Is it possible to schedule all the graphs of τ under all legal triggering sequences such that all deadlines are met?

Dynamic- and Static-Priority Scheduling

- Dynamic-Priority Scheduling
  - allows the switching of priorities between tasks
  - example - EDF

- Static-Priority Scheduling
  - priority switching is not allowed
  - example – rate-monotonic scheduling

- There can be preemptive and non-preemptive versions of both

Relevance of the Model

- Forms the core of the recently proposed recurring real-time task model (due to S. Baruah) [IEEE Real-Time Systems Symp. (1998); Real-Time Systems (to appear)] used for modelling code blocks running in an infinite loop

- The recurring real-time task model generalizes many previous models of real-time systems such as the periodic, sporadic, multiframe, generalized multiframe, and recurring branching models.

Recurring Real-Time Task Model

Known algorithms for feasibility analysis:

- Exponential (in the number of vertices of the graphs) running time.
- Complexity unknown.
- Feasibility analysis for previous (less general) models can be done in pseudo-polynomial time. But no pseudo-polynomial algorithm known for this model.

Results

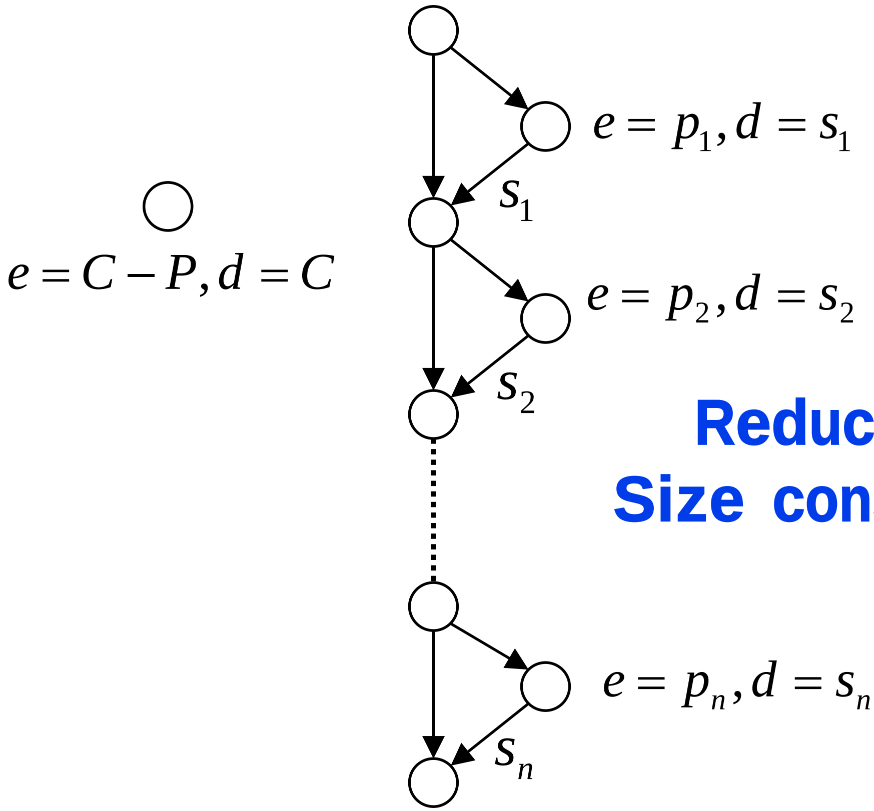
- Preemptive uniprocessor case
- Feasibility analysis, both for dynamic- and static-priorities is NP-Hard
- Dynamic-priority feasibility analysis
  - can be done in pseudo-polynomial time
  - fully polynomial-time approximation scheme for approximate feasibility testing
  - if all the vertices of a task graph have equal execution requirements, it is polynomially solvable

Results cont'd

- Static-priority feasibility analysis
  - tighter condition for sufficiency
  - pseudo-polynomial time algorithm
  - polynomial-time approximate decision algorithm for approximate feasibility testing

Hardness Results

- **Dynamic-Priority Feasibility Analysis is NP-Hard**



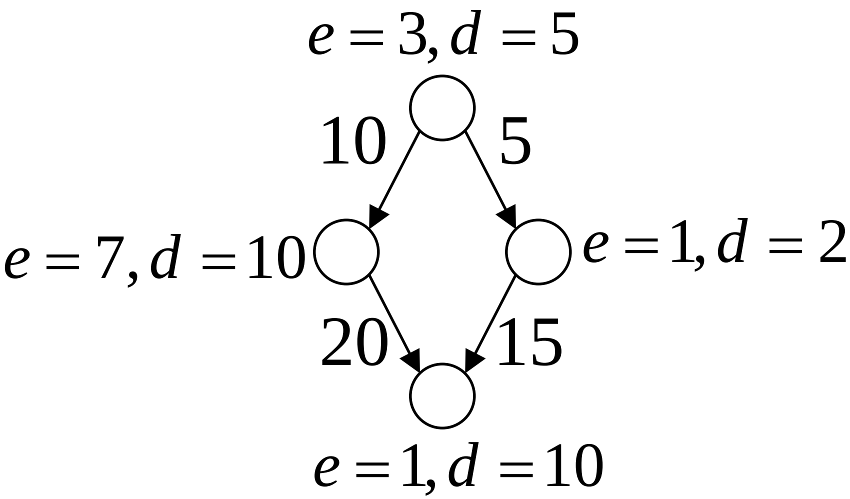
Reduction from Knapsack with Size constraint C and Profit goal P

- **Static-Priority Feasibility Analysis is NP-Hard**

Dynamic-Priority Feasibility Analysis

- **Demand-Bound Function of a task** T

$T.dbf(t)$ = **Max. execution requirement of T within any time interval t for all deadlines to be met**



$$T.dbf(2)=1$$
$$T.dbf(5)=3$$
$$T.dbf(20)=10$$

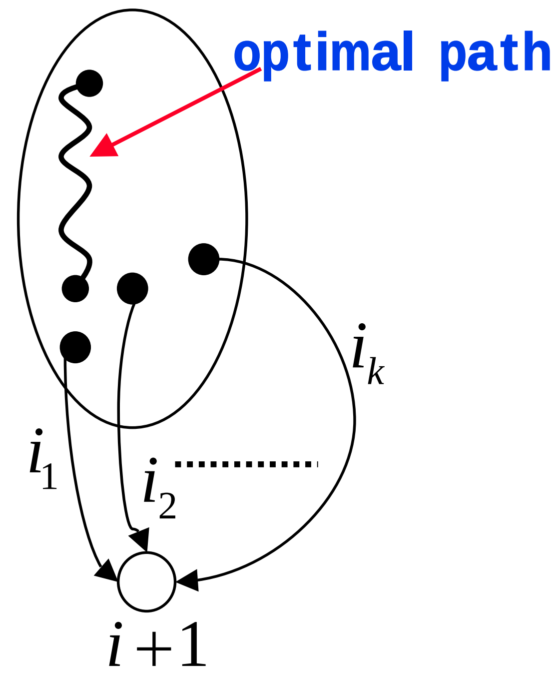
Task set τ is dynamic-priority feasible iff

$$\forall t \geq 0, \sum_{T\in\tau} T.dbf(t) \leq t$$

Demand-bound function *T.dbf(t)*

- **Computing $T.dbf(t)$ is NP-hard**
- **We give a pseudo-polynomial algorithm and a FPTAS**

Let there be directed edges from $\{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ to v_{i+1}



$$t_{i+1,e}^{i+1} \leftarrow \min\{t_{i_j,e-e(v_{i+1})}^{i_j} - d(v_{i_j}) + p(v_{i_j}, v_{i+1}) + d(v_{i+1}) \mid j = 1, \ldots, k\}$$

$$t_{i+1,e} \leftarrow \min\{t_{i,e}, t_{i+1,e}^{i+1}\}$$

$$T.dbf(t) \leftarrow \max\{e \mid t_{n,e} \le t\}$$

Running time = $O(n^3 E)$, where $E = \max_{i=1,\ldots,n} e(v_i)$

Pseudo-polynomial time algorithm for computing *T.dbf(t)*

FPTAS for *T.dbf(t)*

- Based on the previous dynamic programming algorithm
- For any $t \geq 0$ and $\varepsilon \geq 0$ the algorithm outputs a value $\geq (1-\varepsilon) T.dbf(t)$ and runs in time $O(n^4/\varepsilon)$

We denote the result computed by the FPTAS by $T.dbf'(t)$

Approximate Feasibility Testing

Pessimistic Algorithm

$O(|\tau| n^3 / \varepsilon)$

$decision$ = YES

for all values of t at which $T.dbf'(t)$ changes for any T **do**

if $\frac{1}{1-\varepsilon}\sum_{T\in\tau} T.dbf'(t) > t$ **then** $decision$ = NO

endif

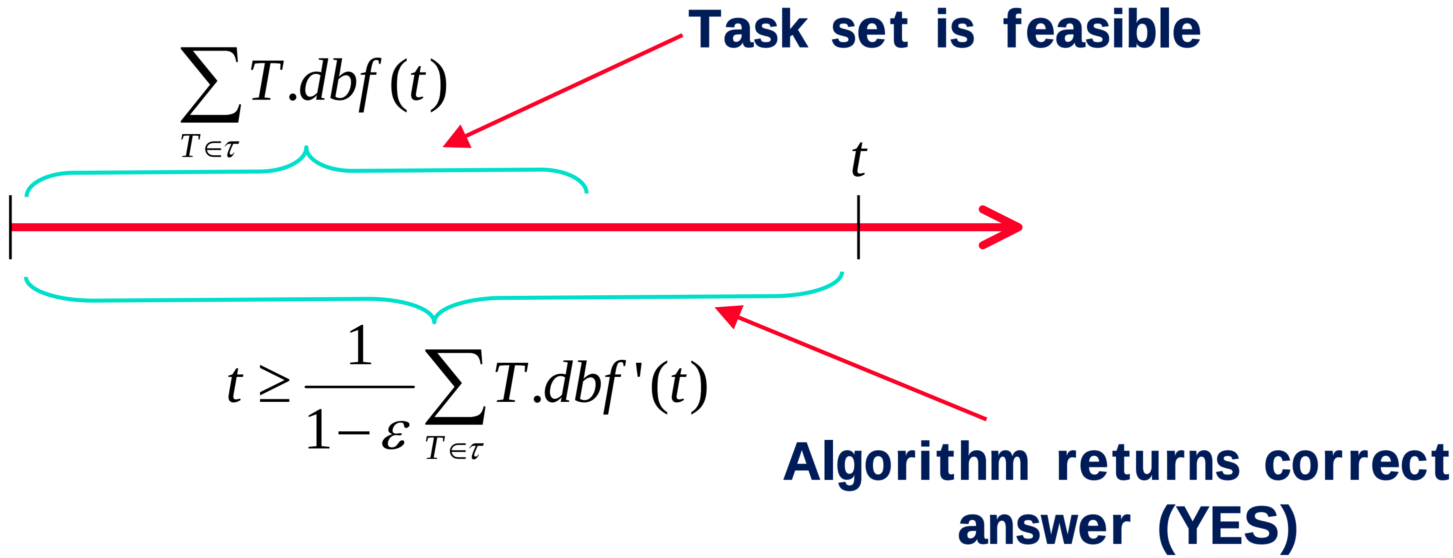
endfor

$O(|\tau| n^2 \varepsilon^{-1} \log n)$

Hence the overall running time is $O(|\tau|^2 n^5 \varepsilon^{-2} \log n)$

Approximate Feasibility Testing

- **Polynomial-time approximation scheme for approximate feasibility testing**
  - τ infeasible always results in correct answer (NO)
  - YES answers are always correct



Dynamic Priorities – Other Results

- Overly **optimistic algorithm** for approximate feasibility testing
- **Pseudo-polynomial** time algorithm for dynamic-priority feasibility analysis
- **Vertices with equal execution** requirements – running time = $O(|\tau| n^3 + |\tau|^2 n \log n)$

Static-Priority Feasibility Analysis

- Test is sufficient but not necessary
  - We give a tighter sufficiency condition
  - For a set of exactly two task graphs the condition is both necessary and sufficient

- For the sufficiency condition:
  - Pseudo-polynomial exact algorithm
  - Approximation scheme for approximate feasibility testing
  - Polynomial algorithm where all the vertices of each task graph have equal execution times

Concluding Remarks

- Settles the complexity of scheduling conditional real-time code

- Although the problem is NP-Hard, there is a pseudo-polynomial time exact algorithm and fully polynomial-time approximation scheme

- Non-preemptive versions? Multiprocessor case?
  - This approach is unlikely to work

Questions? Questions? ? ? Questions?

? ? ? ? Questions?

Questions? ?

? ?

Questions ?

Questions? ? ?

Questions? ? ?

? ?

Questions? Questions?

Questions?

?