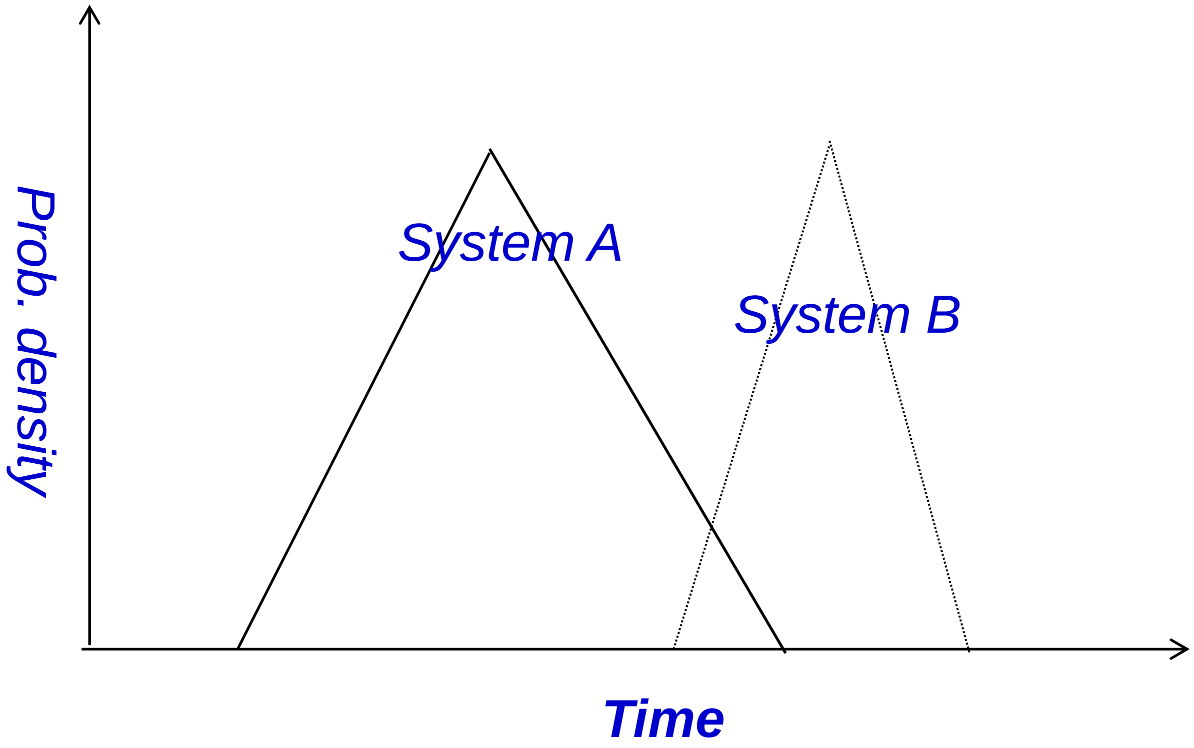
Lecture Note #3
EECS 571
Principles of Real-Time Embedded Systems

Kang G. Shin
CSE/EECS
University of Michigan

Characterization of RTES

- **Two big questions:**
  - How to measure ``goodness'' of RTS ?
  - How to estimate exec time of a program given source code & target architecture?

Which System Is Better?



- *W.r.t. average execution time?*
- *W.r.t. predictability?*
- *What about aM+bV, or (M,V)?*
- *How do we rank the two?*

How to Measure Performance of RTES?

Why not traditional measures, e.g., MIPS?

- **System A is a RISC with 1 instruction per 1.2 clock cycles**
- **Sysemt B is a CISC with 1 instruction per 1.8 clock**

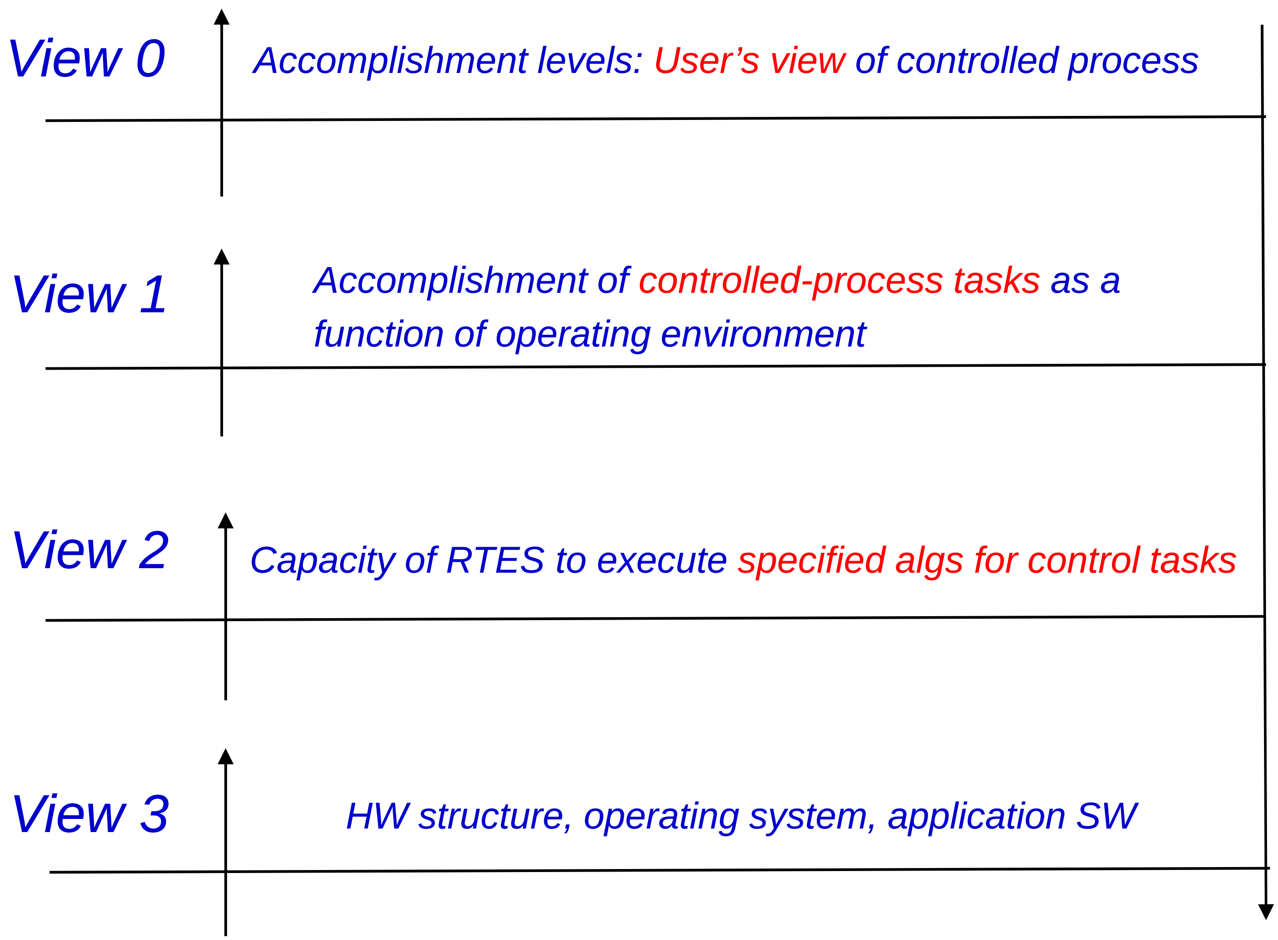
Want RTS performance measures:

- **efficient encoding of relevant information**
- **objective means for ranking candidate systems for a given application**
- **represent verifiable facts**

Traditional Perf. Measures

- **Reliability: R(t)**
- **Availability: A(t)**
- **Throughput**
- **Capacity reliability:** Prob. not being in any of failure states over [0,T].
- **Computational reliability, R(s,t,T):** Prob. system can start task T at time *t* and successfully complete it, where *s* is system state at time *t*
- **Performability:** Given *n* accomplishment levels, A_1, A_2,…,A_n, *performability* is ($P(A_1)$, $P(A_2)$,…, $P(A_n)$), where $P(A_i)$ is probability the computer functions to allow the controlled process to reach A_i.

Hierarchical View of Performability



Cost Functions and Hard Deadlines

- Hard deadline: max controller (computer) ``think'' time that will allow the controlled process to be kept within allowed state space S_A.
- Cost function: of the response time ξ

  $C(\xi) = P(\xi) - P(0)$

  where $P(\xi)$ = performability associated with response time ξ.
- Example 2.8 on pp. 23--25, keeping a body of mass *m* in $S_A = [-b, b]$.

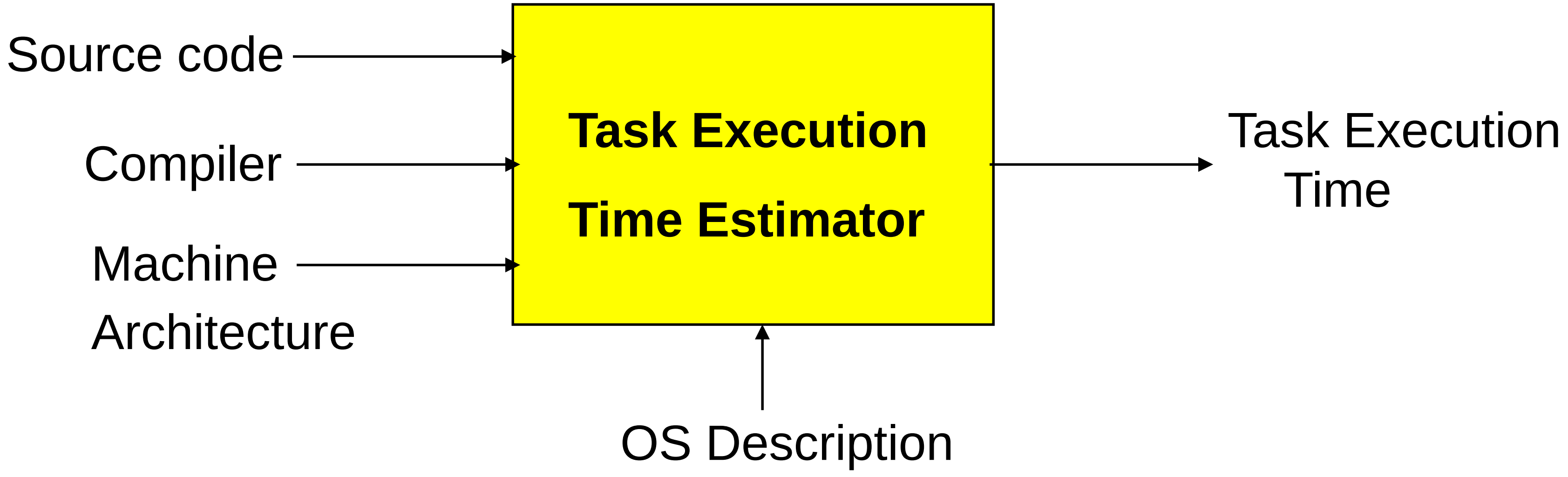
Estimation of Task Execution Times

Reading: Revised chapter on WCET estimation

Task execution time depends on

- **Source code**
- **Compiler: non-unique mapping of source to object code**
- **Machine architecture: regs, cache, memory, pipeline,...**
- **OS: task scheduling, memory management,...**

Need an ideal tool:



Analysis of Straight-line Source Code

1. **L1: a := b*c;**
2. **L2: b := d+e;**
3. **L3: d := a-f;**

L1 can be translated to:

L1.1 Get the address of c
L1.2 Load c
L1.3 Get the address of b
L1.4 Load b
L1.5 Multiply
L1.6 Store into a

- **When does the execution time of L1 become $\sum_{i=1}^{6} T_{exec}(L1.i)$?**
  - no pipeline, no interrupts
  - multiply time depends on data

What about loops and condtional branches?

```
L4:   while (p) do
L5:         Q1;
L6:         Q2;
L7:         Q3;
L8:   end_while;
```

L9: if B_1 then S_1;
  else if B_2 then S_2;
    else if B_3 then S_3;
      else S_4;
  end_if;

If B1 is true then $T(B_1)+T(S_1)+T(JMP)$.

If (not B_1) · B_2 then $T(B_1)+T(B_2)+T(S_2)+T(JMP)$.

…..

What if interrupts?

Why is it hard to estimate (worst-case) task execution time?

- Difficult to determine # of times an instruction will be executed
- Time to execute an instruction is not constant. Why?
  - Pipelining: data, control, and structural hazards
  - Out-of-order execution
  - Cache
  - Branch prediction
  - Multiple instructions per clock cycle are issued
  - Multiple cores on a single die
- Instruction execution time depends not only on the instruction itself and the data it operates on, but also on the state of the machine (execution of previous instructions)

Pipelining and Caches

- **Fetch $\rightarrow$Decode $\rightarrow$Operand Fetch$\rightarrow$Execute**
- **4 concurrent instructions in ``execution"**
- **Timing complexity arises due to:**
  - data inter-dependencies
  - (conditional) branches
  - interrupts
- **Caches to neutralize speed disparity between CPU and memory**
  - Instruction cache (flushed due to branches and context switches)
  - Data cache (dependent on application)

- **SMART cache to avoid cache misses: divide into exclusive and shared areas.**
- **Why not virtual memory for real-time systems?**

Features for Improving Processor Performance

- **Caches**: to offset the gap between processor and main memory speeds
- **Pipelining**: to speed up the execution by overlapping the execution of different instructions
- **Control speculation**: to avoid pipeline stalls caused by conditional branches

How do these affect the estimation of WCET?

1. **R. Heckmann *et al.*: Proceedings of the IEEE, July 2003.**
2. **Krishna and Shin ``Estimating WCET", Sep. 2008**

Execution Times of Concurrent Tasks

- **So far, we discussed a single task/thread, but many RTESes require multiple *dependent* tasks to run concurrently on a multiprocessor/multicore system**
- **Need to model concurrent tasks for their execution times and scheduling, e.g., *article by Peng and Shin 1987***
- **The system model must *simultaneously* consider both the processing architecture (platform) and the tasks (application)**

The System Model

- **Platform architecture: distributed by connecting processing nodes (PNs) with an interconnection network**
  - Each PN's architecture, e.g., uni-/multi-processor, registers, pipeline, cache.
  - Operating system, e.g., VxWorks, QNX, WinCE, Greenhills
  - Network protocols, CAN, FlexRay, Ethernet, WLAN, p2p.
- **Task system:**
  - Application→*communicating* tasks →activities/modules.
  - Object of *assignment*: tasks
  - Object of *scheduling*: modules or activities
  - Activities are modeled by GSPNs (Generalized Stochastic Petri Nets) then converted to CTMCs (Continuous-Time Markov Chains)
  - Precedence constraints on tasks

Applications modeling

- **Task-oriented: too coarse to capture details we want**

- **Module-oriented: difficult to study**
  - message scheduling policies
  - communication protocols
  - task execution stage of each PN

$\Rightarrow$ **Need a new module-oriented model with finer granularity**

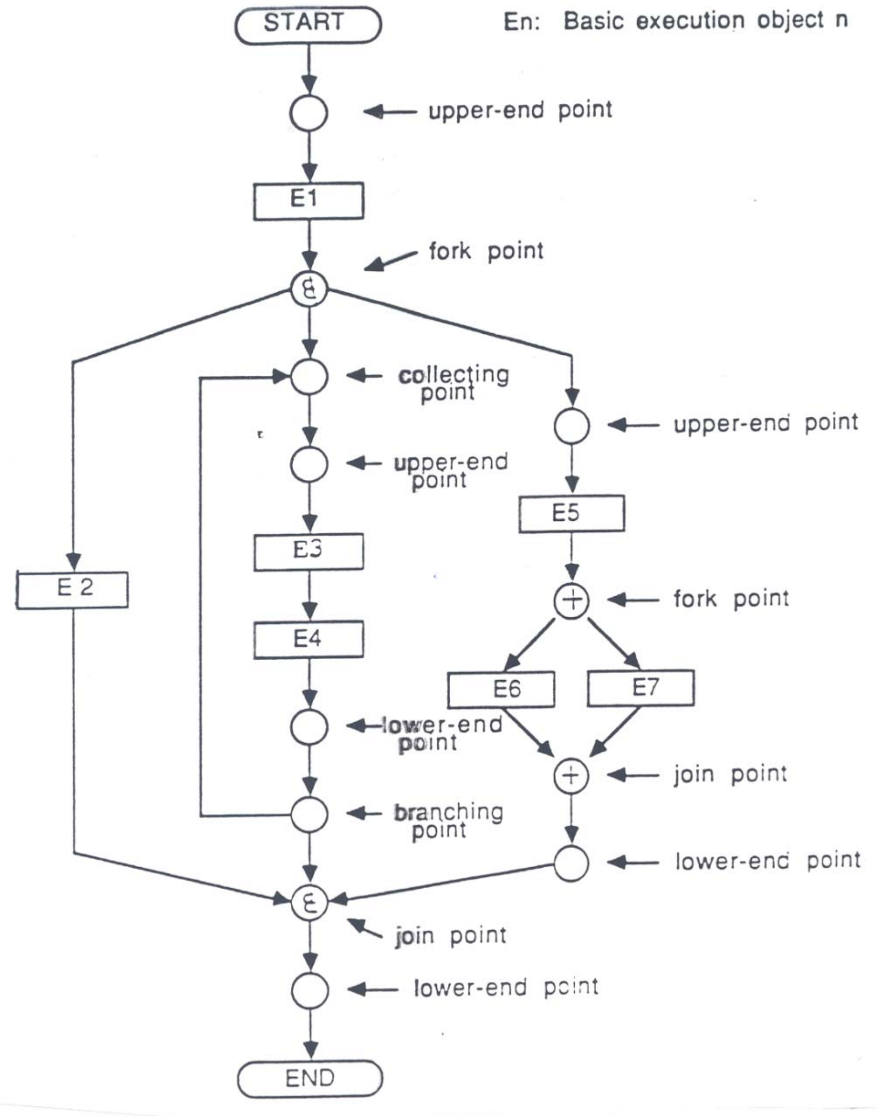
Peng's Approach

- **Contiguous stretches of code are combined into activities while preserving precedence constraints and avg exec times.**
- **GSPN to model activities and precedence constraints**
  ⇒a *sequence of CTMCs* for modeling evolution of a task system.

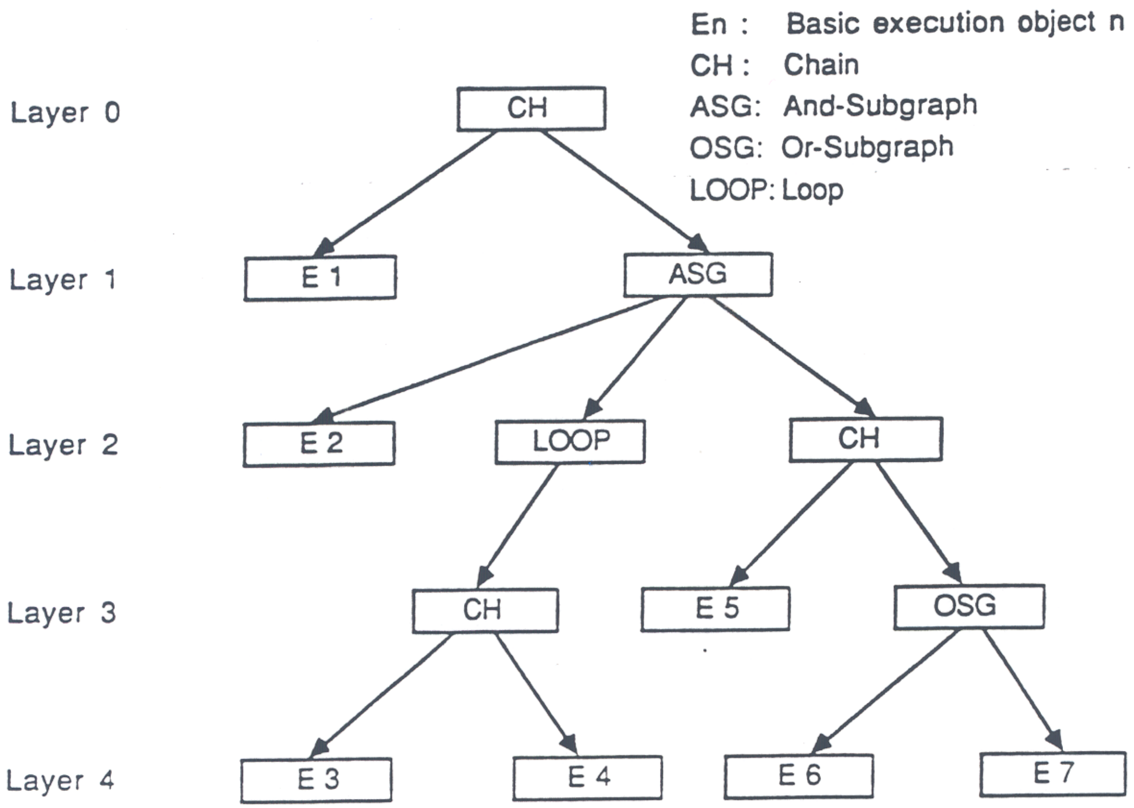
- **State of each CTMC = execution stage each PN is in**
- **State transition = execution of an activity**

- **TFG (Task Flow Graph) describes a task to be executed by a PN and consists of: Chain, AND-FORK & AND-JOIN, OR-FORK & OR-JOIN, Loop**
- **Task tree describes organization TFG with 4 subgraphs and basic execution objects (BEOs) with Root=TFG, leaf = BEO, layer #**

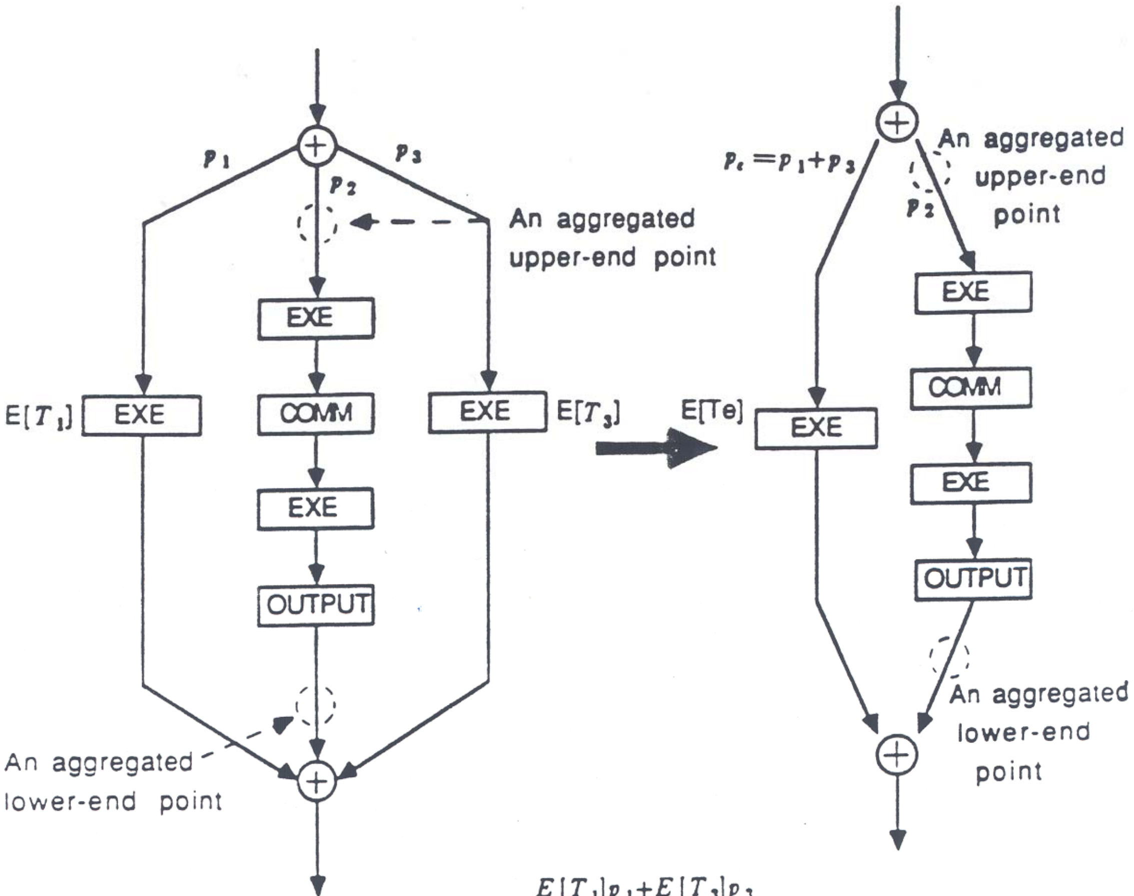
Example Task Flow Graph



Task Tree for the Example TFG



Combination Process for an OR Graph

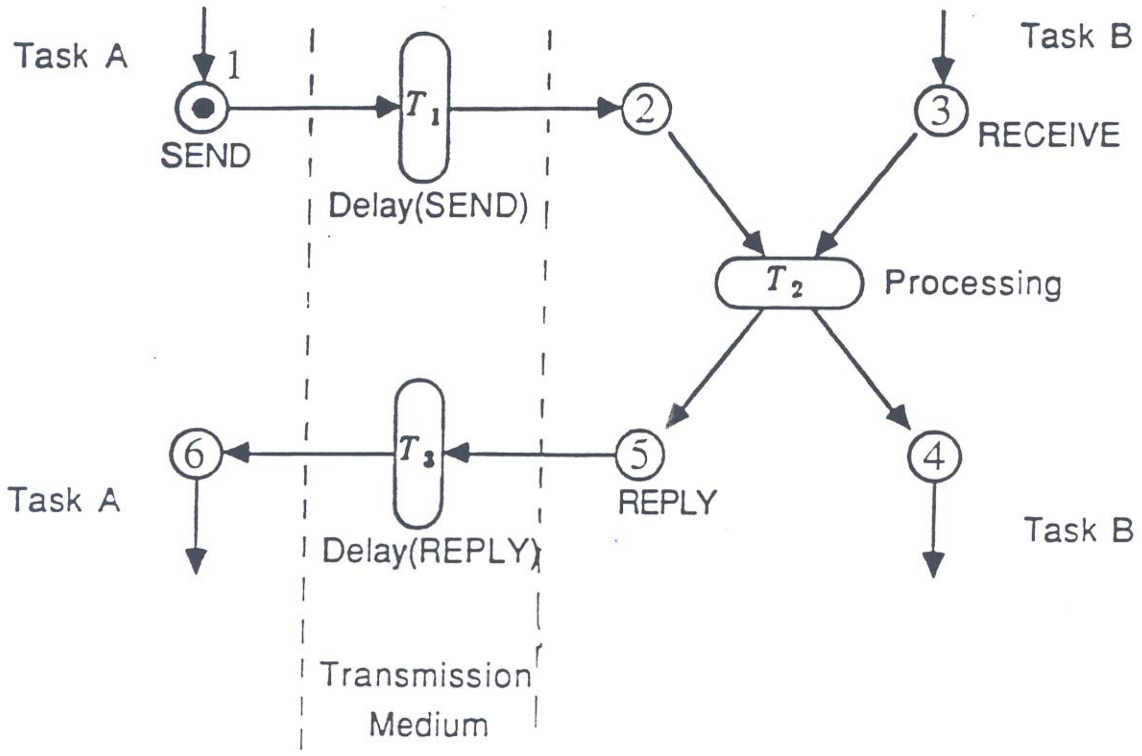


$$E[T_e] = \frac{E[T_1]p_1 + E[T_3]p_3}{p_e}$$

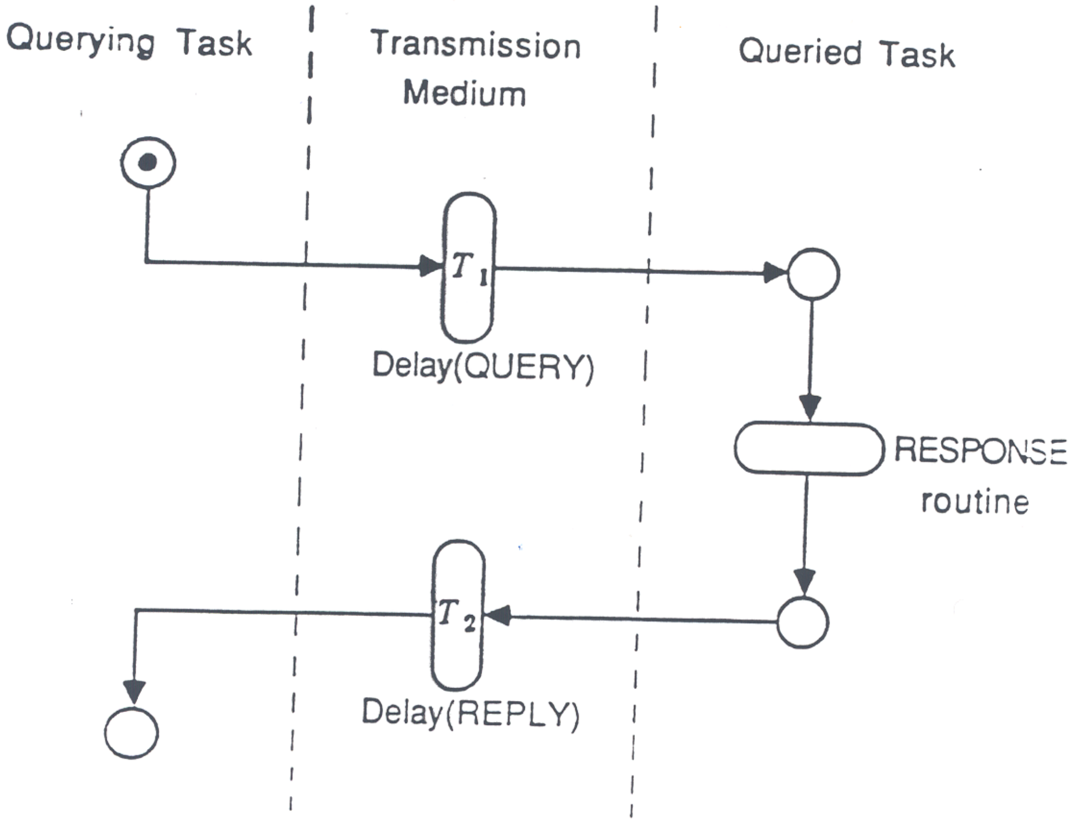
Definitions

- **Module: combination of 2 or more code stretches or modules (*recursive*)**
- **Activity: *largest* module that can be formed w/o violating any precedence constraints**
- **Marked Petri Net, C = (P,T,I,O, μ) where μ: P$\rightarrow$# of tokens for place p $\in$ P.**
- **GSPN: marked Petri Net w/ a nonnegative random *firing delay* for each transition t $\in$ T.**
- **Example GSPN models for SEND-RECEIVE-REPLY, REQUEST-RESPONSE, WAITFOR**

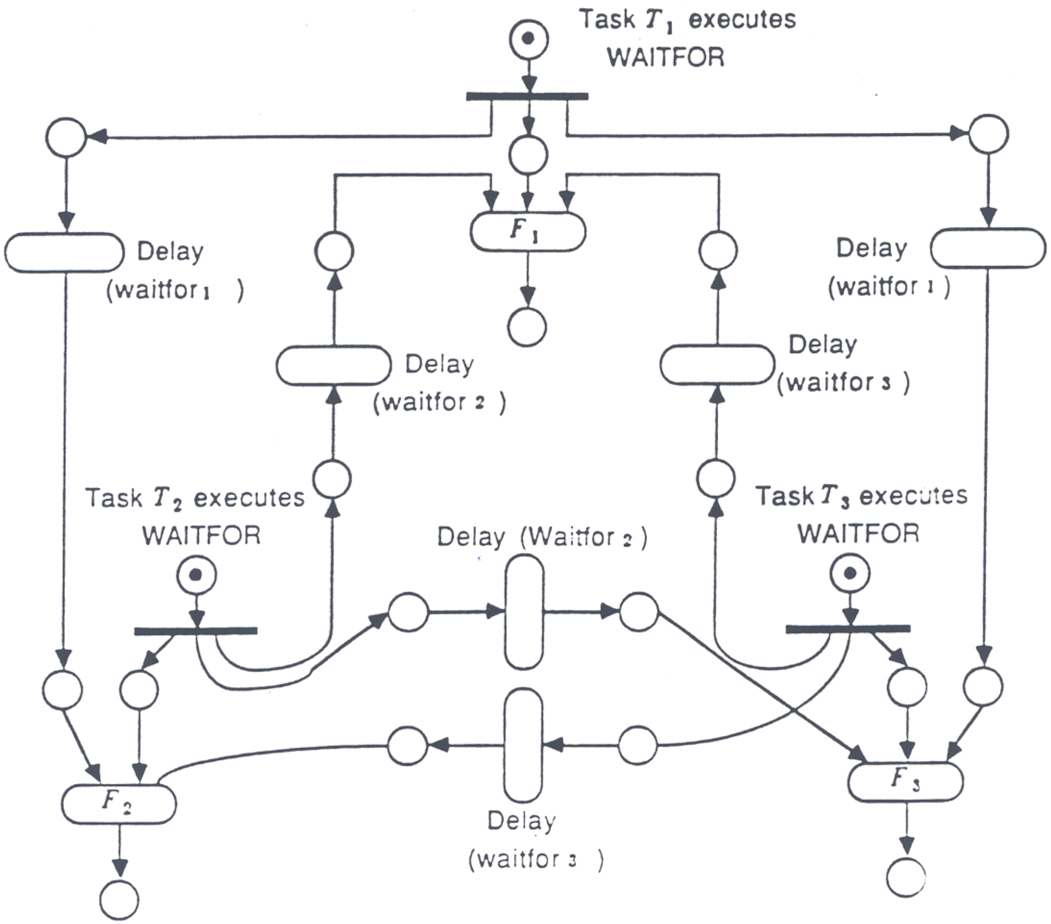
GPSN Model for SEND-RECEIVE-REPLY



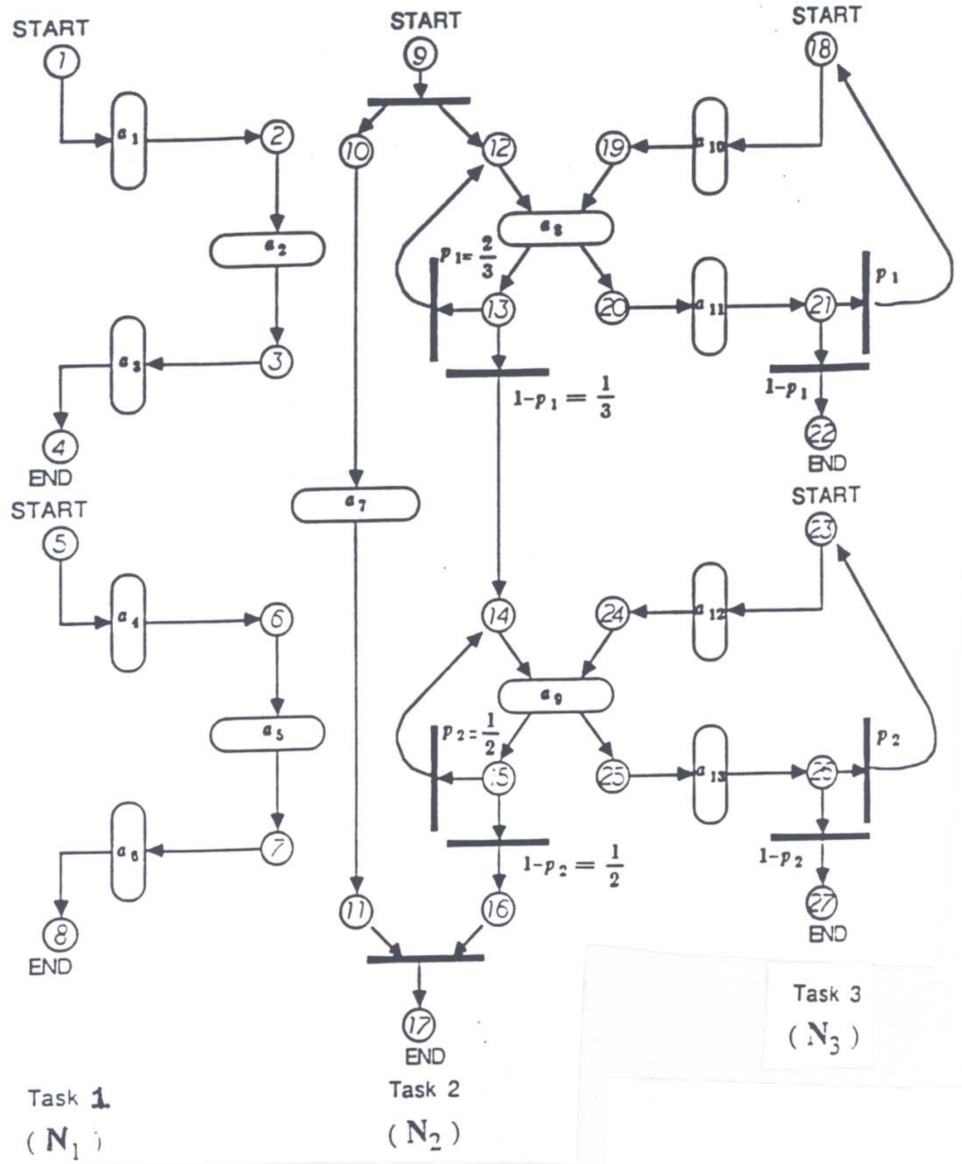
GSPN Model for REQUEST-REPLY



GSPN Model for WAITFOR



System-Wide GSPN Example



CTMC Model

- **Tasks are invoked at $t_0 = w_1, w_2, \cdots, w_{l+1} = t_0 + L$**
- **Sequence of CMTCs $\{S_k, \Lambda_k, \Theta_k): k=1,2,\cdots,\ell\}$ where**

  S_k =set of states reachable in $[w_k, w_{k+1})$;

  Λ_k: $S_k \times S_k \rightarrow T$ = event-driven transition function;

  Θ_κ: $S_k \rightarrow S_{k+1}$ time-driven transition function.

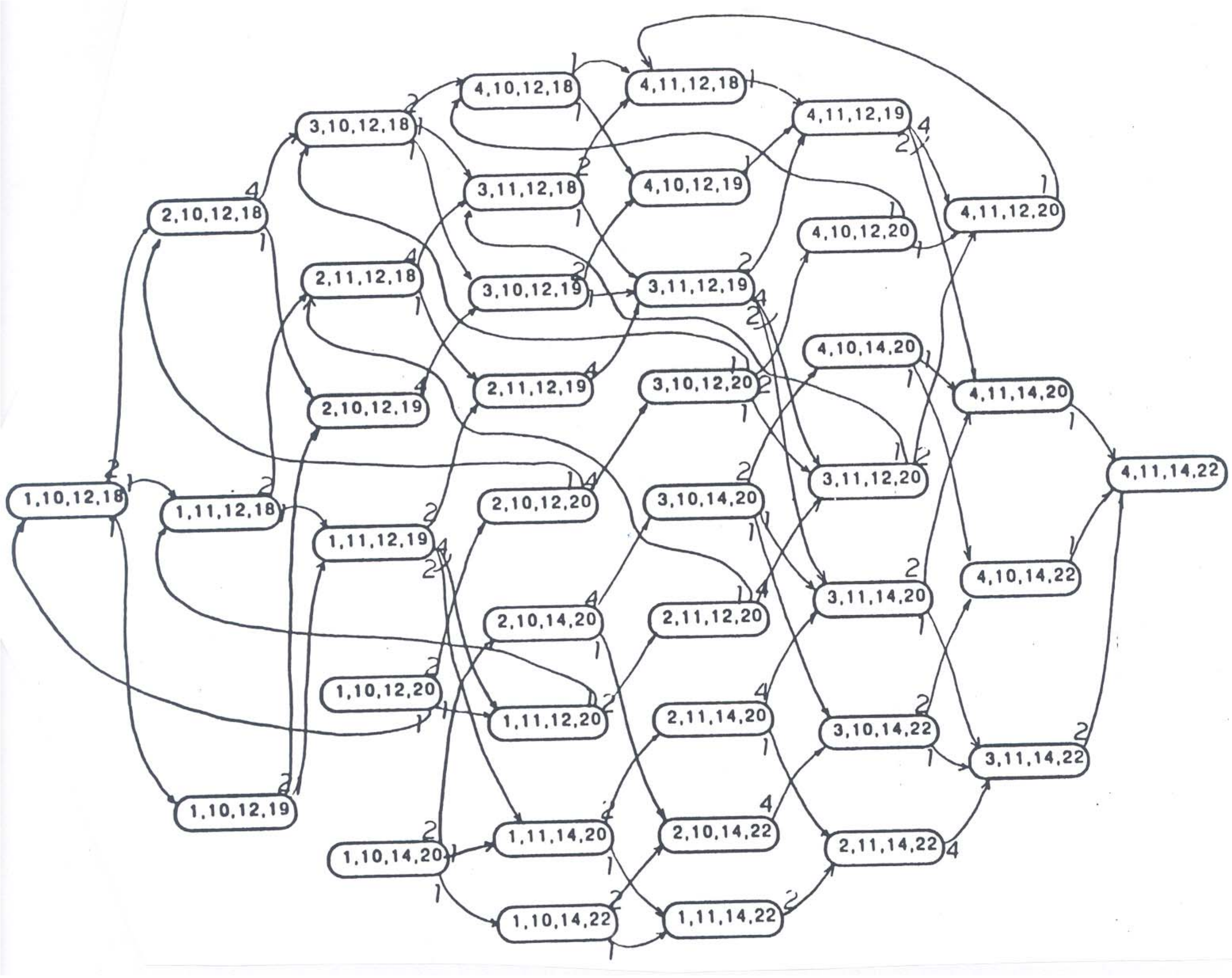
- **At the beginning of L, mark system-wide GSPN by generating a token in each START place**

- **Determine marking at time $t \in [w_1, w_2)$ by event-driven transition firings.**

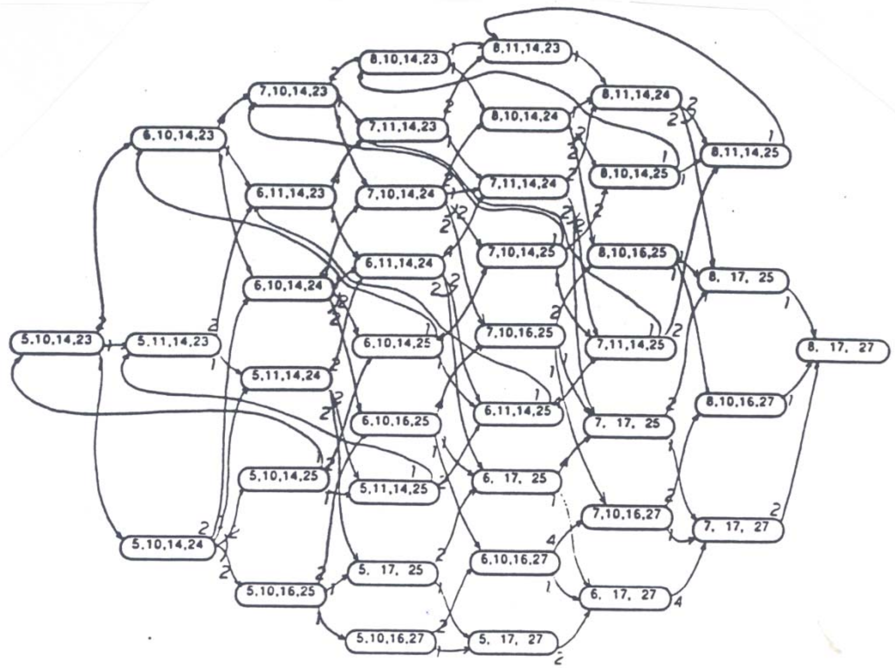
- **Determine marking at time w_j, $2 \leq j \leq \ell$ by**
  - token generation at START place
  - token removal from previous invocations
  - token movement via event transition from w_j

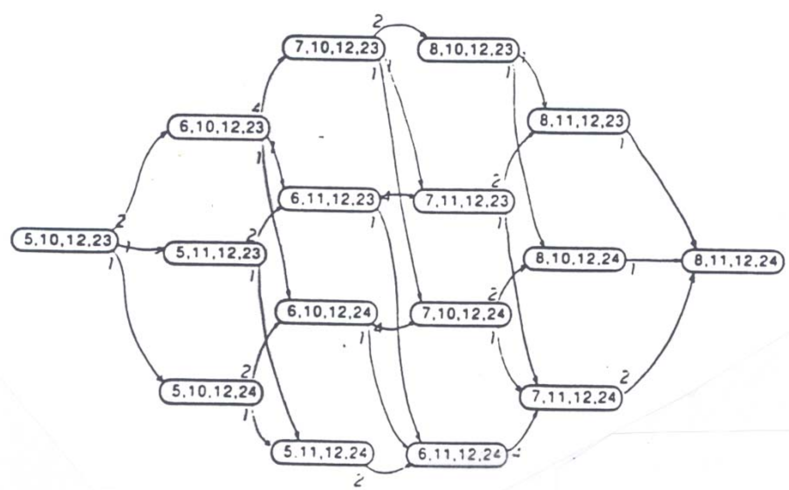
- **Determine marking at time $t \in [w_j, w_{j+1})$ for $j=2,\cdots,\ell$ by event transition firings**

CTMC Model for t $\in$ [0,5)



CTMC Model for t ∈ [5,10)





What Do Models Say?

- **Each state in $S = \bigcup_{k=1}^{\ell} S_k$ represents which stage of task each PN is currently executing**
- **S_1 contains START state; S_l contains END state;**
- **$S_i \bigcap S_j = \emptyset, \forall\, i \neq j$.**
- **<u>If</u> CFG contains loops <u>then</u> all state-transition rate (STR) diagrams are cyclic <u>else</u> acyclic**
- **Depending on markings at w_j, the STR diagram in $[w_j, w_{j+1})$ could be disconnected**
- **Preserves precedence constraints**
- **Some states are time-critical, e.g., (4,*,*,*).**
- **# of simultaneously executable activities can be greater than # of processors available at the PN, e.g., (6,10,14,24).**

  $\Rightarrow$ Need to consider task assignment & scheduling