Especificação, Modelação e Projecto de Sistemas Embutidos

Process Networks/Task graphs

Paulo Pedreiras
pbrp@ua.pt

Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro

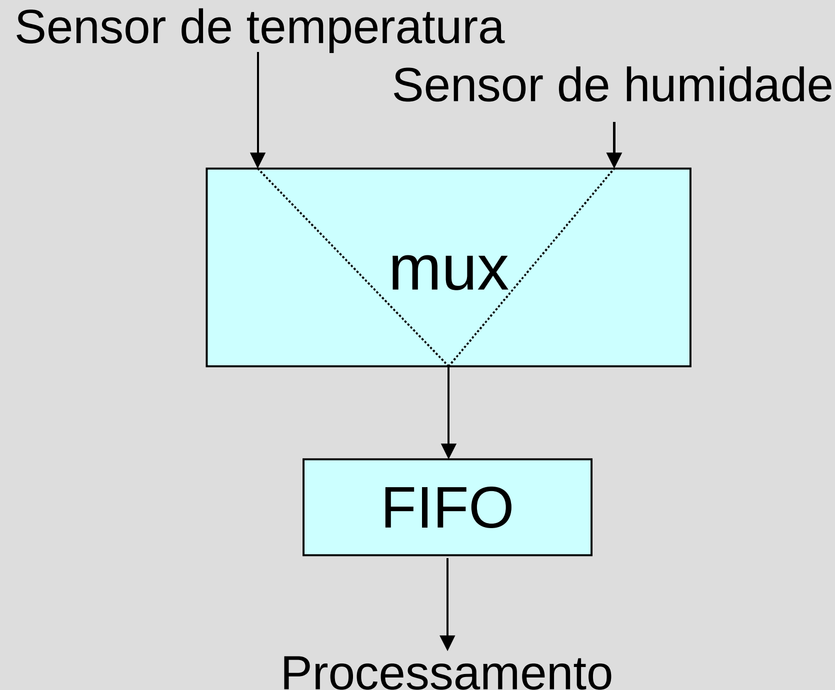
Apresentação baseada nos recursos didácticos disponibilizados com o livro “Embedded Systems Design”, por P. Marwedel

V1.1 Novembro/2008

Process networks

Muitas aplicações podem ser especificadas na forma de **processos comunicantes**

- Exemplo: sistema com dois sensores:



Modelação com linguagens imperativas

```
MODULE main;
 TYPE some_channel =
          (temperature, humidity);
     some_sample : RECORD
                    value : integer;
                    line  : some_channel
                   END;
```

```
PROCESS get_temperature;
VAR sample : some_sample;
BEGIN
 LOOP
  sample.value := new_temperature;
  IF sample.value > 30 THEN ....
  sample.line := temperature;
  to_fifo(sample);
 END
END get_temperature;
```

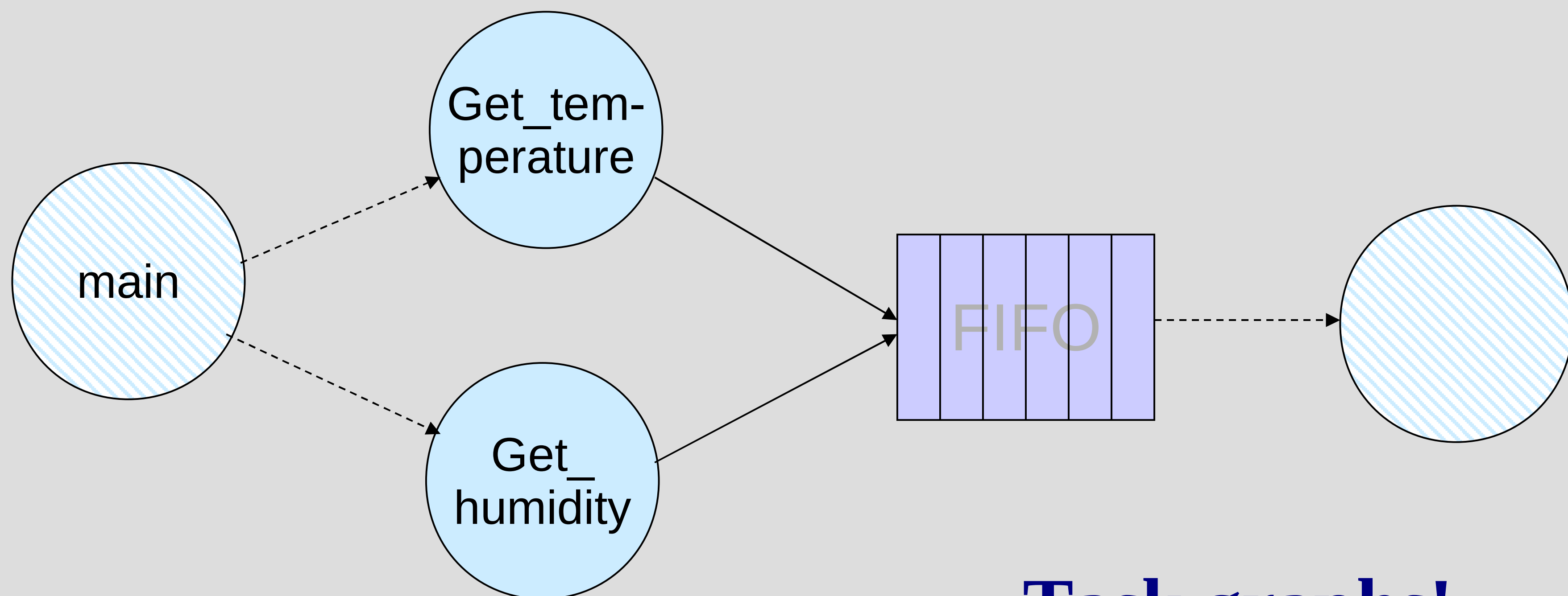
```
PROCESS get_humidity;
 VAR sample : some_sample;
 BEGIN
  LOOP
   sample.value := new_humidity;
   sample.line := humidity;
   to_fifo(sample);
  END
 END  get_humidity;
```

```
BEGIN
       get_temperature;
       get_humidity;
       ...
END;
```

- Chamadas bloqueantes a “new_temperature”, “new_humidity”
- **Estrutura clara** para **múltiplos processos** com **interdependências**?

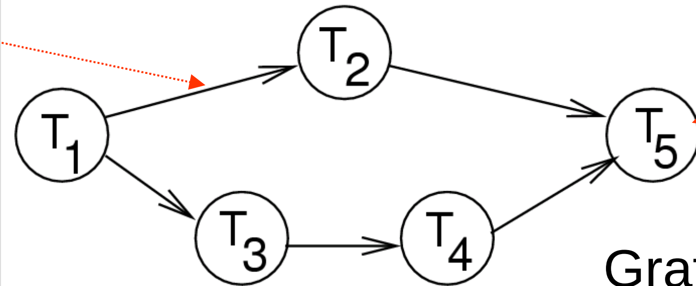
Dependências entre tarefas/processos

- **Como modelar então dependências entre tarefas/processos?**



Task graphs

Dependência causal

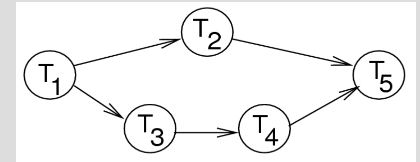


Nó

Grafo de dependências

- **Nó** (*vertice*)
  - Representam **processos** que executam **operações** (**programas** descritos numa **linguagem** de **programação** e.g. “C”,Java)
  - Convertem **fluxos** de dados de **entrada** em **fluxos** de dados de **saída**
  - Tipicamente os processos são **iterativos;** cada **iteração consome** dados de entrada, **processa**-os e **gera** dados de saída
- **Ramos dirigidos** (*edge*)
  - Representam **relações** entre processos
  - Indicam **dependência causal** *(sequence constraints)*

Task graphs



Def.: Um **grafo de dependências** é um grafo dirigido $G=(V,E)$, em que V representa os nós, E os ramos e $E \subseteq V \times V$ é uma ordem parcial.

- Se $(v1, v2) \in E$, então $v1$ é denominado um **predecessor imediato** de $v2$ e $v2$ denomina-se um **sucessor imediato** de $v1$.
- Seja E^* o fecho transitivo de E. [1]
  Se $(v1, v2) \in E^*$, então $v1$ designa-se **predecessor** de $v2$ e $v2$ é um **sucessor** de $v1$.

[1] O Fecho transitivo de R é definido como: R^+
- Se $(a, b) \in R$, então $(a, b) \in R^+$
- Se $(a, b) \in R^+$ e $(b, c) \in R^+$ então $(a, c) \in R^+$

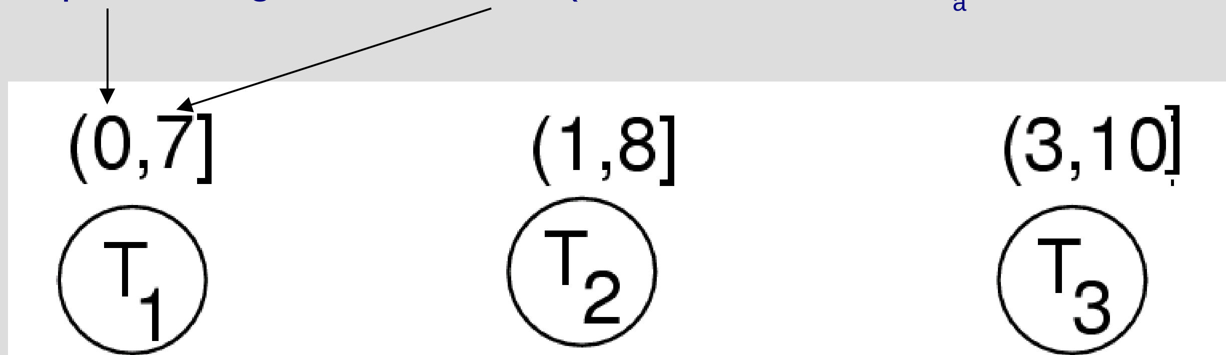
Task graphs: informação temporal

Os *task graphs* podem conter **informação adicional** respeitante a **propriedades temporais**

- Esta propriedades podem ser e.g. tempo de chegada, tempo de execução, *deadline*, período
- Esta informação é **fundamental** para a fase de **escalonamento** dos processos

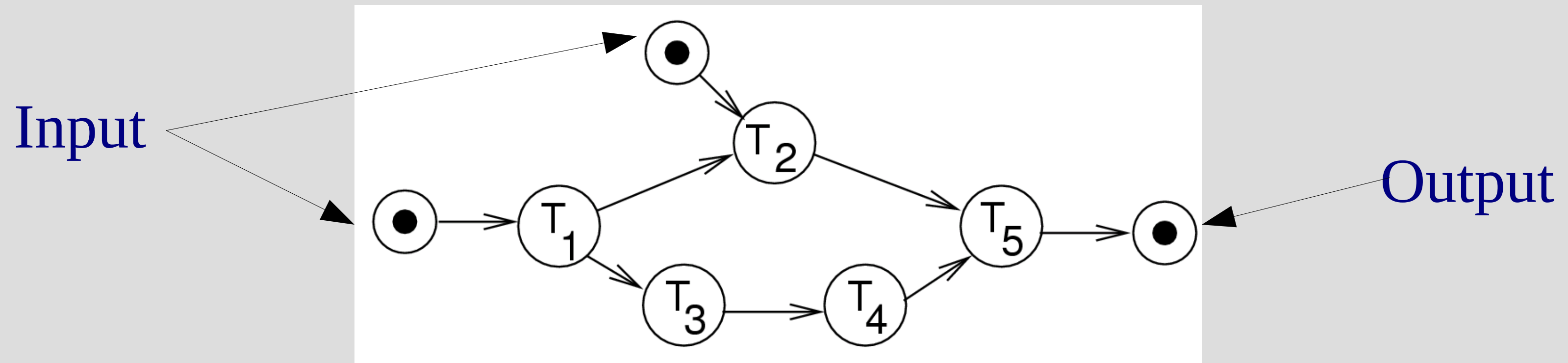
Intervalo de execução:

Tempo de chegada deadline (tarefa activada em $t_a > 0$ e tem de terminar até $t_f \leq 7$)



Nota: Tarefas T1, T2 e T3 independentes
Notação de acordo com [Liu, 2000]

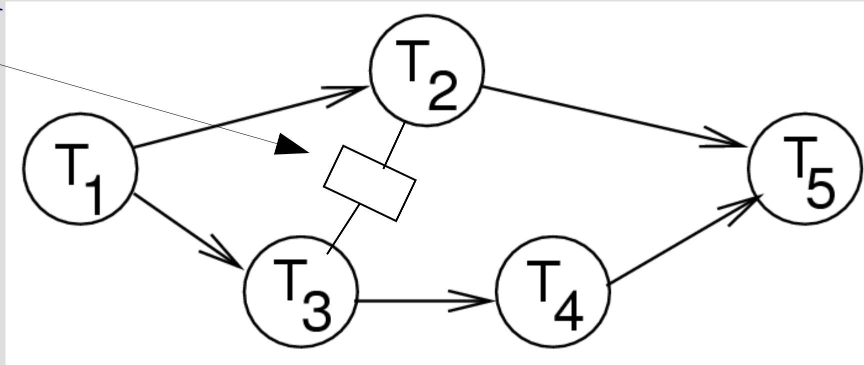
Task graphs: descrição de operações de I/O

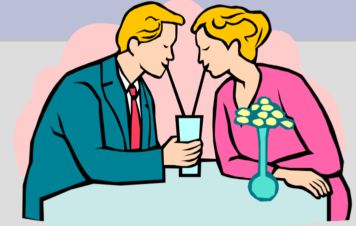


- No exemplo anterior as operações de **I/O não** são **descritas explicitamente**
  - Presume-se que nós **sem predecessores** no grafo eventualmente **recebem input do ambiente**, e
  - Presume-se que nós **sem sucessores** geram output que eventualmente será **processado pelo ambiente**
- A descrição das operações de **I/O** pode ser efectuada de uma forma **explicita**, e.g. por meio de vértices parcialmente preenchidos [Thoen and Catthoor,2000]

Task graphs: recursos partilhados

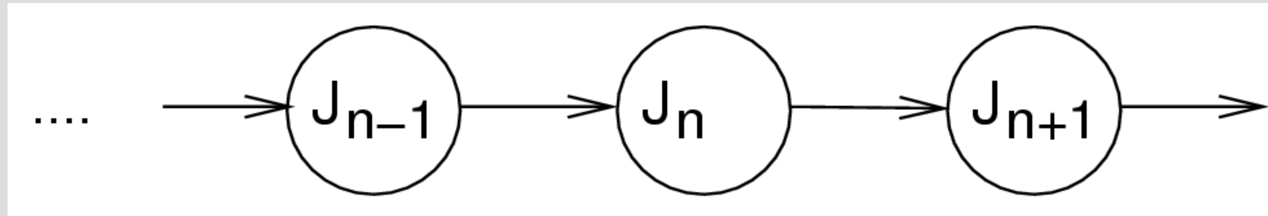
T2 e T3 partilham recursos com acesso exclusivo





- As tarefas podem requerer **acesso exclusivo** a recursos partilhados
  - e.g. dispositivo de I/O, zonas de memória partilhada
- A informação acerca de exclusão mutua pode ser **incluída no gráfico** por forma a ser considerada durante o escalonamento
  - Permite e.g. evitar inversões de prioridade

Task graphs: escalonamento periódico



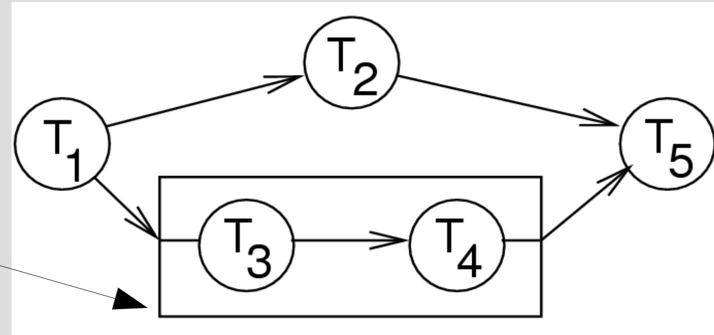
- Em muitas aplicações as tarefas são **executadas periodicamente**
  - e.g. controlo, processamento digital de sinal
  - Cada **execução/instância** de uma tarefa denomina-se “**job**”
  - Os **grafos** de tarefas são, nestes casos, **infinitos**
- A figura ilustra um grafo que **inclui** as **instâncias** J_{n-1} a J_{n+1} de uma **tarefa periódica**

Task graphs: composição hierárquica

- A **complexidade** das **computações** denotadas pelos nós é **muito variável**
- O nível de complexidade dos nós designa-se **granularidade**
- **Qual o nível de granularidade adequado?**
  - Se se considerar que cada **nó** é um **processo** que deve ser escalonado por um **kernel**, é vantajoso restringir a granularidade ao nível de um processo para **minimizar os overheads**
    - mudanças de contexto, tempo de escalonamento, ...
  - Por outro lado, se houver **hardware especializado** (e.g. DSPs) pode ser vantajoso **modelar operações elementares**
    - Permite alocar computações ao hardware mais adequado

Task graphs: composição hierárquica

Nó hierárquico



- O uso de **nós hierárquicos** permite suportar diferentes níveis de granularidade. E.g.:
  - A um **nível mais alto** os nós podem denotar operações complexas (e.g. processos)
  - A um **nível intermédio** blocos básicos
  - A um **nível inferior** podem mesmo denotar operações aritméticas básicas
- Um rectângulo denota um **nó hierárquico** (ver Fig.)

Multi-thread graphs (MTG)

Def.: Um **grafo multi-thread** M é definido com o $M \equiv (O, E, V, D, \vartheta, \iota, \Lambda, E^{lat}, E^{resp}, \nabla^{i}, \nabla^{av})$, onde:

- ***O*: Conjunto de nós operacionais.**
  - Podem ser: Threads, events, sink e source e nodos "or"
    - **Thread**: sequência de operações com tempo de execução determinístico, que uma vez iniciado pode executar até ao final sem sincronização (c/ ambiente ou outras tarefas);
    - Nós **event** modelam inputs externos;
    - Apenas um nodo **sink** e um nodo **source**
      - Token em cada ramo do *source* no inicio da execução; *sink* termina a execução
    - A regra de activação dos nós requer que todos os ramos de controlo tenham um *token*. Os **nodos "or"** representam situações em que basta apenas que um de um conjunto de ramos de entrada tenha um *token*.

Multi-thread graphs (2)

- ***E*: conjunto de ramos**
  - Ramos pode definir:
    - Dependências de **dados**, relações de **precedências** ou **restrições temporais** entre processos
    - Definem as **condições de activação** dos nós
- ***V*, *D*, ι: Comunicação de dados.**
  - **Interna**:
    - Baseada em **memória partilhada**
    - O modelo define "**nós-variável**" e "**ramos de dados**" (*variable nodes/data edges*, resp.) para ligar os portos de dados das threads aos nós-variável
    - Os ramos de dados **não implicam precedência** de dados; sincronização pode ser conseguida com ramos de controlo ou evento complementares

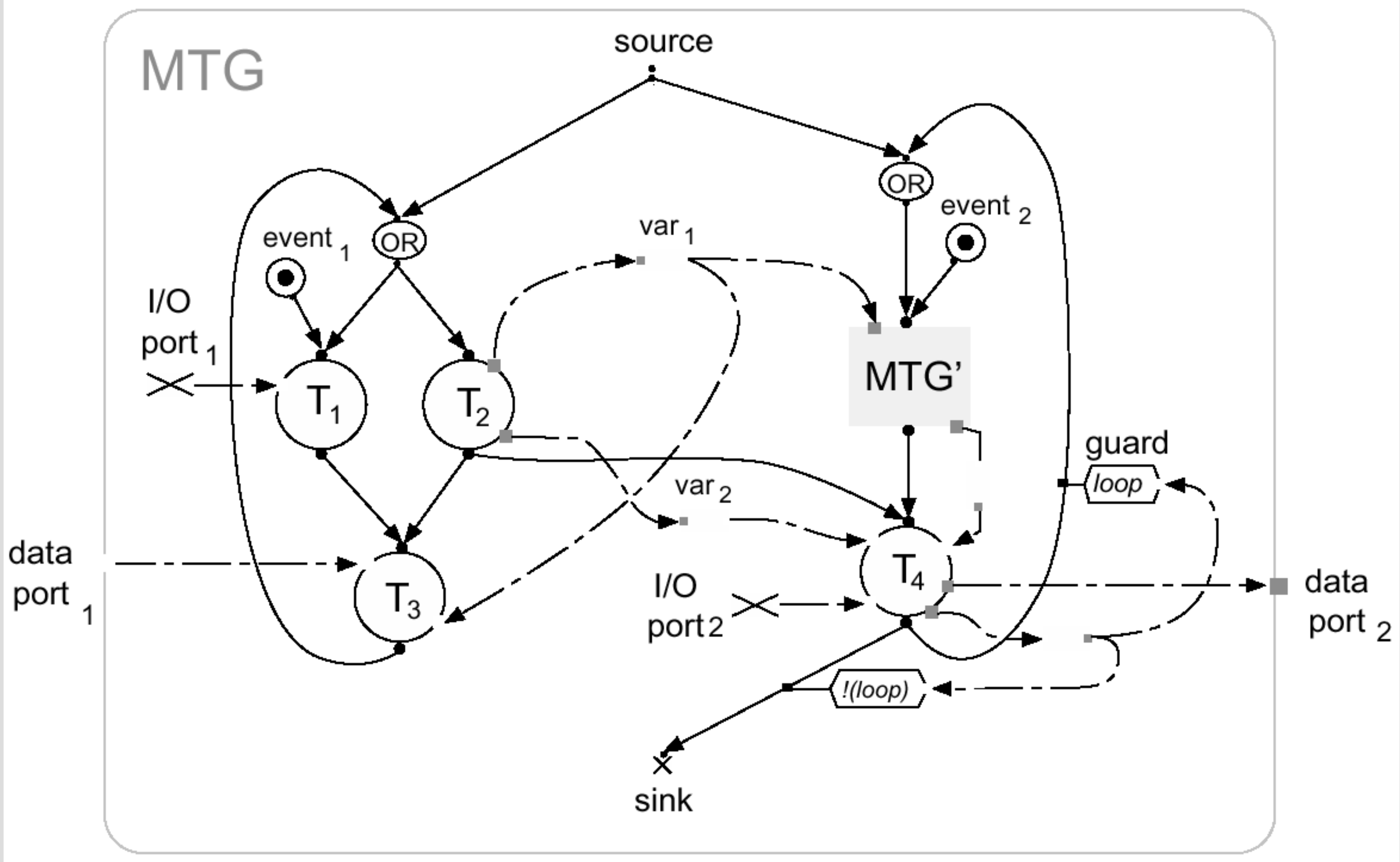
Multi-thread graphs (3)

- Λ: Associa intervalos de **latência de execução** (a tarefas) e **taxa** de ocorrência de **eventos** (a eventos)
  - **Tarefas** : baseado no pior tempo de execução (*worst-case execution time*), pois é necessário garantir o correcto funcionamento em todas as circunstâncias
  - **Eventos**: período ou tempo minimo entre activações (*minimum inter-arrival time*) para eventos periódicos/ esporádicos, resp.
  - Esta informação é **essencial** para o **escalonamento do sistema!!!**

Multi-thread graphs (4)

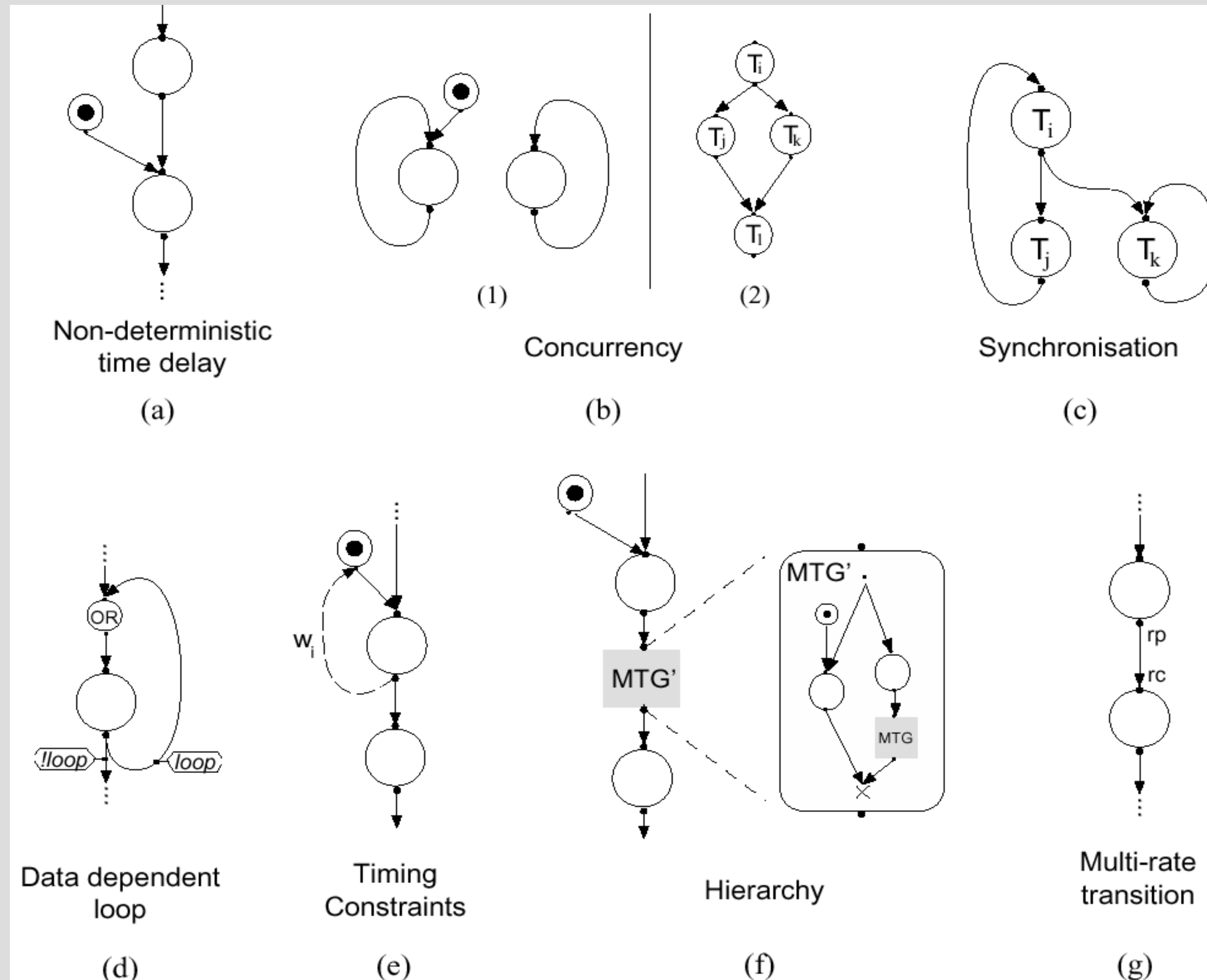
- E^{lat}, E^{resp}, ∇^{i}, ∇^{av} **definem restrições temporais**
  - **Restrição de latência**
    - Definir a **distância temporal entre duas tarefas**.
      - E.g. o inicio de execução de uma tarefa T_j tem de acontecer w unidades temporais após o final de uma tarefa T_i
  - **Tempo de resposta**
    - Semelhante à anterior, mas **relaciona** um **evento com uma tarefa**
      - E.g. uma tarefa T_i deve terminar no máximo w unidades temporais após o evento v
  - **Taxa de activação**
    - Limita a **taxa de execução** de um (sub) grafo.
      - Especialmente útil e.g. para nós de I/O, pois permite garantir um serviço mínimo ou limitar a taxa máxima com que um dispositivo é servido

Multi-thread graphs: notação gráfica (1)



E.g. de F. Thoen, G. Goossens, J. Der Steen, H. De Man, "The Multi-Thread Graph Model for Embedded Software Synthesis", Tech report.

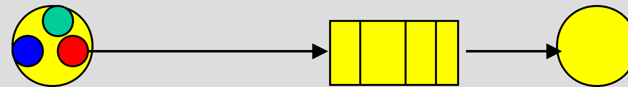
Multi-thread graphs: notação gráfica (2)



E.g. de F. Thoen, G. Goossens, J. Der Steen, H. De Man, “The Multi-Thread Graph Model for Embedded Software Synthesis”, Tech report.

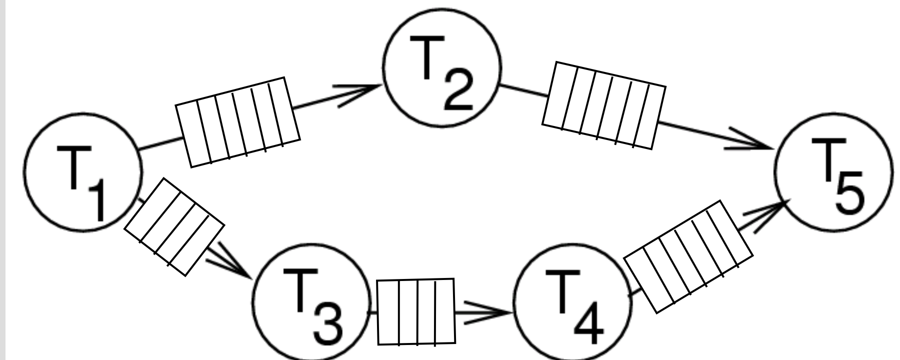
Task graphs com passagem assíncrona de mensagens: Kahn process networks

Para permitir a troca assíncrona de mensagens a comunicação entre tarefas é *buffered*



Kahn process networks:

- Grafo de tarefas executáveis,
- Comunicando por meio de FIFOS com dimensão infinita



Kahn process networks: propriedades (1)

- Cada **nó** corresponde a um **programa/tarefa**
- **Comunicação** realizada **exclusivamente** por **canais**
- Canais possuem **FIFOs** tão **grandes** quanto o **necessário** (inf.)
- Os canais transmitem os dados com uma **latência imprevisível** mas **finita**
- No caso geral os **tempos de execução** das tarefas são **desconhecidos**
- Os operações de **escrita não** são **bloqueantes**; as operações de **leitura** são **bloqueantes**.
- Canais estabelecem uma **relação de 1 para 1**
  - i.e., apenas um transmissor e um receptor por canal

Kahn process networks: propriedades (2)

- Um processo **não pode verificar** se há dados disponíveis **antes** de efectuar **leitura**
- Um processo **não pode esperar** por dados em **mais que um porto** em cada **instante**
- **Corolário**: a **ordem de leitura** dos dados **depende** apenas dos **dados** e **não** do **tempo** de chegada do processo
  - Assim, as **Kahn process networks são determinísticas**; para um certo *input* o resultado é sempre o mesmo
    - Complexidade dos algoritmos, velocidade de processamento dos nós, ... são irrelevantes

Khan process networks: exemplo

```
Process f (in int u, in int v, out int w) {
    int i; bool b = true;
    for (;;) {
            i = b ? wait (u) : wait (v); // wait returns next token in FIFO, blocks if empty
            printf("%i\n", i);
            send (i, w); // writes a token into a FIFO w/o blocking
            b = !b;
    }
}
```

© R. Gupta (UCSD), W. Wolf (Princeton), 2003

- É um modelo de computação paralela **usado na prática**
  - e.g. na Philips/NXP.
- Na prática os **FIFOs são finitos**; escalonar KPNs sem acumular tokens é uma tarefa complexa
- Tipicamente o **escalonamento** é efectuado **on-line** visto ser difícil prever com rigor o seu comportamento preciso

Mais detalhes em http://en.wikipedia.org/wiki/Kahn_process_networks

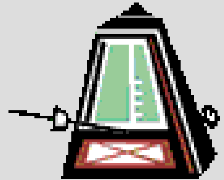
Passagem assíncrona de mensagens: Synchronous Data Flow (SDF)

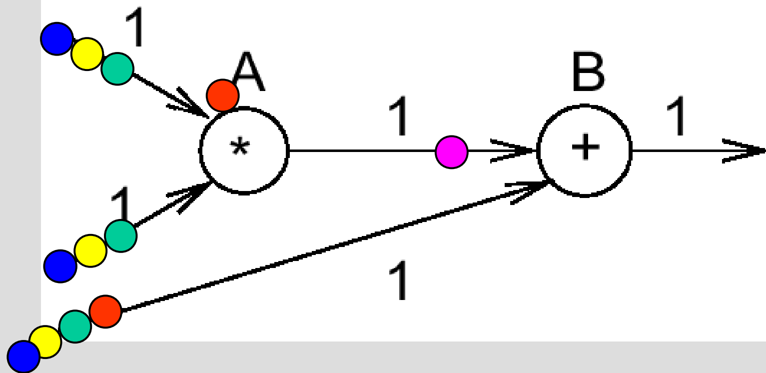
Passagem assíncrona de mensagens

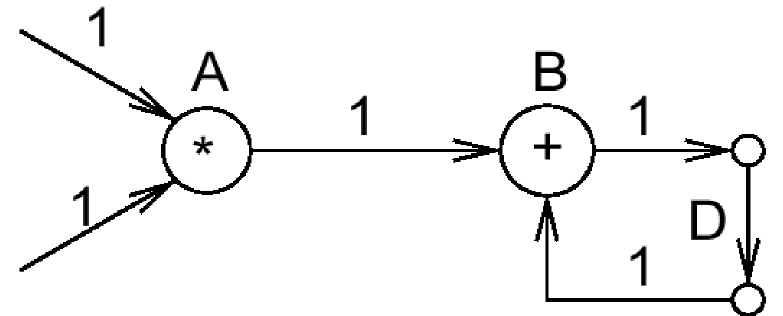
⇒ tarefas não têm de esperar que o seu output seja aceite (escrita não bloqueante)

Fluxo de dados síncrono (Synchronous data flow)

⇒ **todos** os ***tokens*** são **consumidos** ao **mesmo tempo**







- Nós indicam operações
- Ramos indicam dependências de dados
- O número de *tokens* produzidos/consumidos em cada activação é constante (indicado na *label* dos ramos)

Synchronous Data Flow (SDF)

Taxa de produção/consumo de tokens é fixa

- Equação de balanço (uma por canal):

Número de disparos por iteração

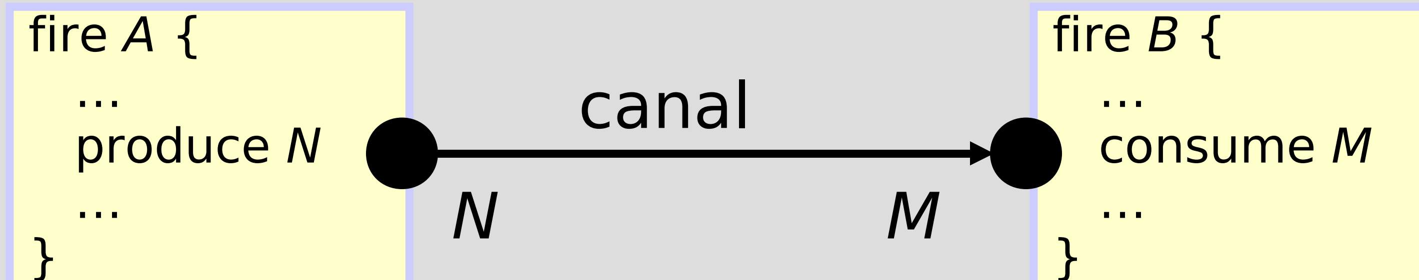
Número de tokens produzidas por "disparo"

$$f_A N = f_B M$$

Número de tokens consumidas por "disparo"

Número de "disparos" por iteração

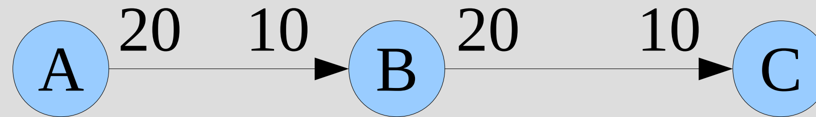
- É possível determinar **em tempo de compilação**
  - **Ordem** de **execução** (escalonável estaticamente)
  - **Dimensão** dos **buffers**
  - Existência de ***deadlocks***



fonte: ptolemy.eecs.berkeley.edu/presentations/03/streamingEAL.ppt

SDF: exemplo

- Considere-se o seguinte grafo SDF



- **Problema**: encontrar o menor set $S=\{r_A, r_B, r_C\}$, em que r_X representa a taxa de activação da tarefa X

$$\begin{cases} r_A o_A = r_B i_B \\ r_B o_B = r_C i_C \end{cases} \quad \begin{cases} r_A * 20 = r_B * 10 \\ r_B * 20 = r_C * 10 \end{cases} \quad \begin{cases} r_B = 2 r_A \\ r_C = 2 r_B \end{cases}$$

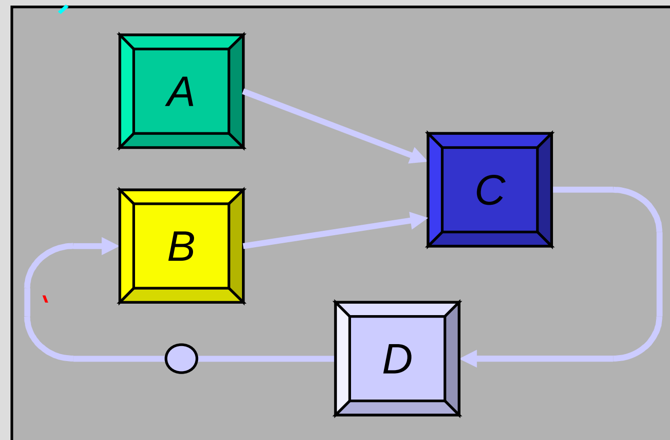
$\{o_x, i_x\}$ representam os tokens produzidos/consumidos resp. pela tarefa X

Solução S={1,2,4}

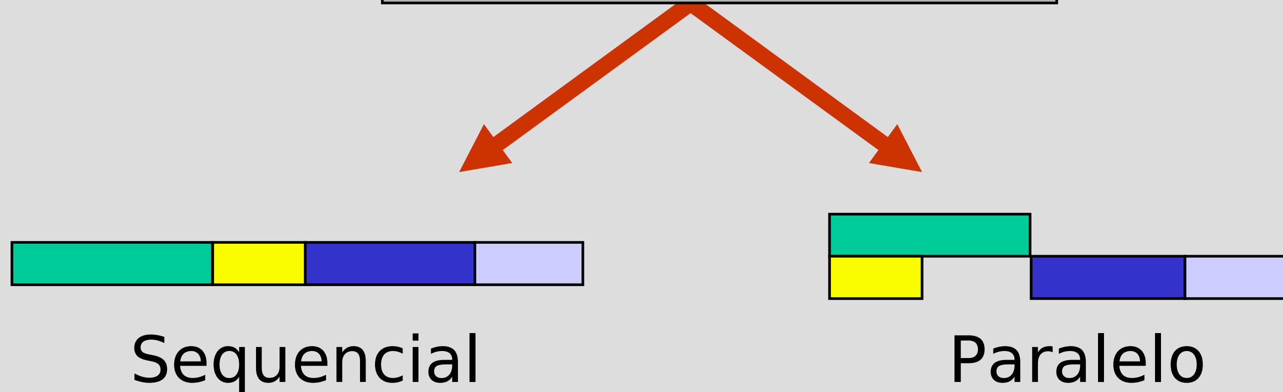
- Um possível **escalonamento** será "ABCCBCC"
  - **Grafo consistente**: eq. de balanço solúvel e escalonamento válido
- Existem algoritmos para cálculo das **taxas de activação** bem como para calcular **escalonamentos**
  - Tempo calculo é função linear/cúbica resp. da dimensão do grafo

Escalonamento paralelo de modelos SDF

O SDF é adequado para o **mapeamento automático** em processadores paralelos bem como para a **síntese** de **circuitos paralelos**

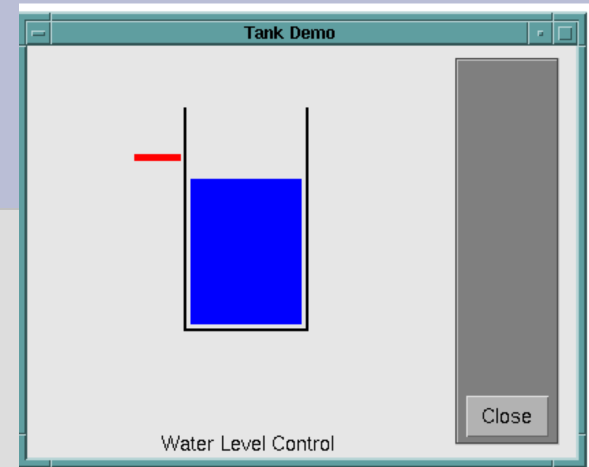


No caso geral a **optimização** do mapeamento / escalonamento é uma **tarefa complexa**!!



fonte: ptolemy.eecs.berkeley.edu/presentations/03/streamingEAL.ppt

O Simulink apresenta um MoC similar



PID
PID Controller
1
Constant
+
−
Sum1
Signal Generator
+
−
error
Mux
Mux1
Fuzzy Logic Controller
-1
const
Switch
VALVE
Subsystem
0.5
tank max inflow
WATER TANK
tank 2
Scope2
Mux
Mux2
animtank
S-Function
Scope4
Comparison
change scope
change
du/dt
Derivative
Try the demo 'sltank2'

From www.mathworks.co.uk/access/helpdesk/help/toolbox/fuzzy/fuzzyt25.shtml

Passagem síncrona de mensagens: CSP

- **CSP** – acrónimo de ***C****ommunicating* ***S****equential* ***P****rocesses* [Hoare, 1985]
- Comunicação baseada em *rendez-vous*
- Exemplo:

```
process A
..
var a ...
  a:=3;
  c!a; -- output
end
```

```
process B
..
var b ...
  ...
  c?b; -- input
end
```

Ambos os processos sincronizam neste ponto

MoC da linguagem OCCAM

Passagem Síncrona de Mensagens: ADA

- Linguagem de programação nomeada em homenagem a Ada Lovelace
  - Considerada a primeira mulher programadora
- Processo iniciado pelo Departamento de Defensa do EUA (DoD) que queria **evitar o uso de múltiplas linguagens de programação**
- DoD efectuou a **definição** dos **requisitos** e efectuou a selecção a partir de um conjunto de propostas; desenho seleccionado **baseado** em **PASCAL**
- ADA’95 é uma extensão orientada a objectos do ADA original

Passagem Síncrona de Mensagens: usando tarefas em ADA

- ADA tem o **conceito de tarefa**

```
procedure example1 is
    task a;
    task b;
    task body a is
      -- local declarations for a
     begin
       -- statements for a
     end a;
```

```
 task body b is
   -- local declarations for b
   begin
     -- statements for b
   end b;
begin
 -- Tasks a and b will start before the first
 -- statement of the body of example1
end;
```

Exemplo de [Burns and Wellings, 1990]

Passagem Síncrona de Mensagens: ADA *rendez-vous*

```
task screen_out is
 entry call_ch(val:character; x, y: integer);
end screen_out;
task body  screen_out is
...
  accept call_ch ...  do ..
  end call_ch;
...
```

Enviando uma mensagem:

```
begin
 screen_out.call_ch('Z',10,20);
 exception
  when tasking_error =>
          (exception handling)
end;
```

- Quando duas tarefas necessitam de comunicar a **primeira a chegar** ao ponto de encontro tem de **aguardar** que a tarefa "**parceira**" também atinja o ponto de controlo
- **Sintacticamente** a comunicação é descrita pela **invocação** de **procedimentos**

Sumário

- **Process networks**
  - Motivação
  - Grafos de tarefas
  - Multi-Thread Graphs
- **Passagem Assíncrona de Mensagens**
  - Kahn process networks
  - SDF
- **Passagem Síncrona de mensagens**
  - CSP
  - ADA