Sistemi in tempo reale
Anno accademico 2006 - 2007

Introduzione ai sistemi di controllo in tempo reale

Giuseppe Lipari
http://feanor.sssup.it/~lipari

Scuola Superiore Sant'Anna

Outline

Contenuti del corso

Parte di Sistemi operativi:

- Introduzione ai sistemi in tempo reale
- Richiami di architetture dei calcolatori e di sistemi operativi
- Programmazione concorrente
- Sincronizzazione dei processi
- Schedulazione in tempo reale: priorità fisse, analisi di schedulabilità
- Tecniche di implementazione di processi di controllo e di campionamento
- Diagrammi a stati e macchine a stati finiti
- Cambi di modo

Contenuti del corso

Parte di controllo digitale (svolta da Luigi Palopoli)

- Richiami di sistemi tempo discreti e controllo digitale
- Analisi dei ritardi
- Sistemi campionati e selezione del tempo di campionamento
- Sistemi multi-rate

Contenuti del corso (cont.)

Parte pratica (Lipari)

- Il package software “TrueTime” per la simulazione di sistemi di controllo in tempo reale
- Introduzione a RTAI / Linux
- Programmazione di un semplice controllore con RTAI

Testi consigliati

- P. Ancilotti, M. Boari, A. Ciampolini, G. Lipari, “Sistemi Operativi”, Mc Graw Hill
- G. Buttazzo, “Sistemi in tempo reale”, Pitagora Editrice

Modalità di esame

- Un progettino:
  - tramite il sistema operativo RTAI / Linux oppure
  - tramite il package di simulazione TrueTime
- E' possible (consigliabile) riprendere un progettino già sviluppato in altri corsi, ed effettuare uno studio dell'implementazione real-time (simulativa o implementativa).

Modalità di iscrizione all'esame:

- prenotarsi per e-mail o per telefono almeno 2 giorni prima dell'esame
- Il giorno dell'esame, viene prima discusso il progetto, e poi svolto un orale.
- all'orale potrebbe esservi chiesto di svolgere qualche semplice esercizio

Contatti

- Web site: `http://feanor.sssup.it/~lipari`
- E-mail: `lipari@sssup.it`
- Telefono: 050 882 030
- Ricevimento: non c’è un giorno fisso, chiamatemi oppure mandate una e-mail per prendere appuntamento.

Outline

Definizione classica

Secondo la definizione classica,

> *Un sistema si dice real-time se la correttezza dei suoi output dipende non solo dal valore che questi assumono ma anche dal tempo in cui sono prodotti.*

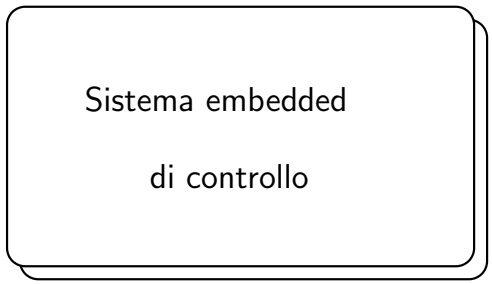
Secondo un'altra defizione più informale

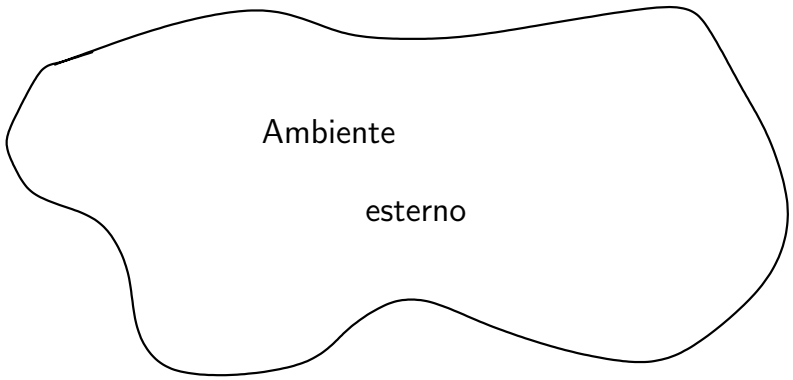
> *Un sistema real-time è un sistema che produce i suoi output in un tempo predicibile, non necessariamente nella maniera più veloce possibile.*

Oltre le definizioni

- Che vuol dire predicibile?
- Quali sono le caratteristiche essenziali di un RTOS?
- Come mai Windows e Linux non sono RTOS?

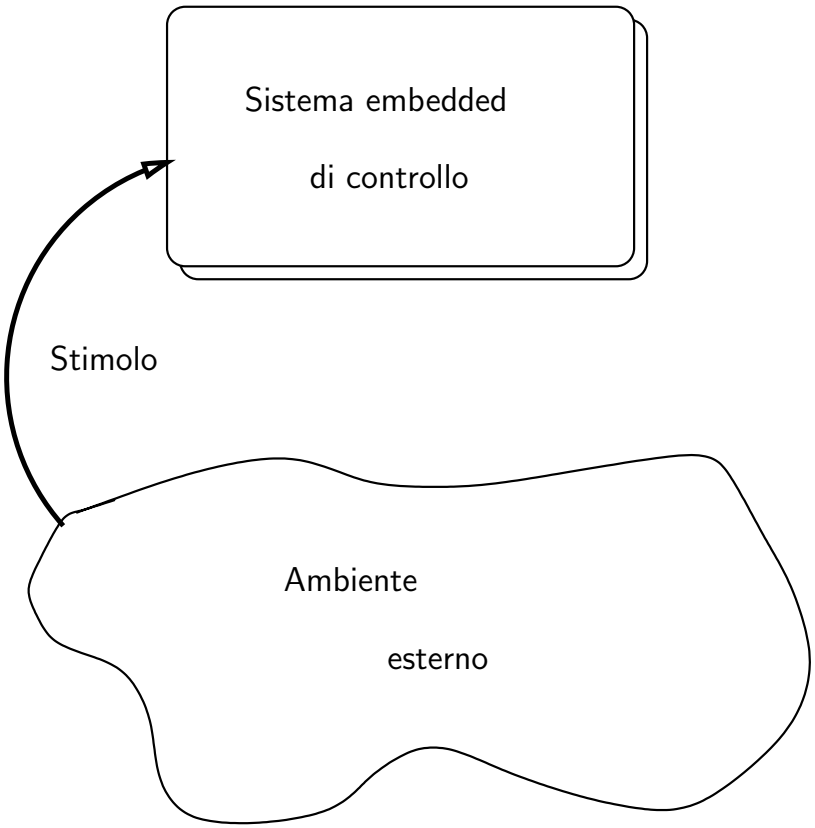
Sistemi reattivi





Una prima qualità fondamentale è la **reattività** agli stimoli esterni.

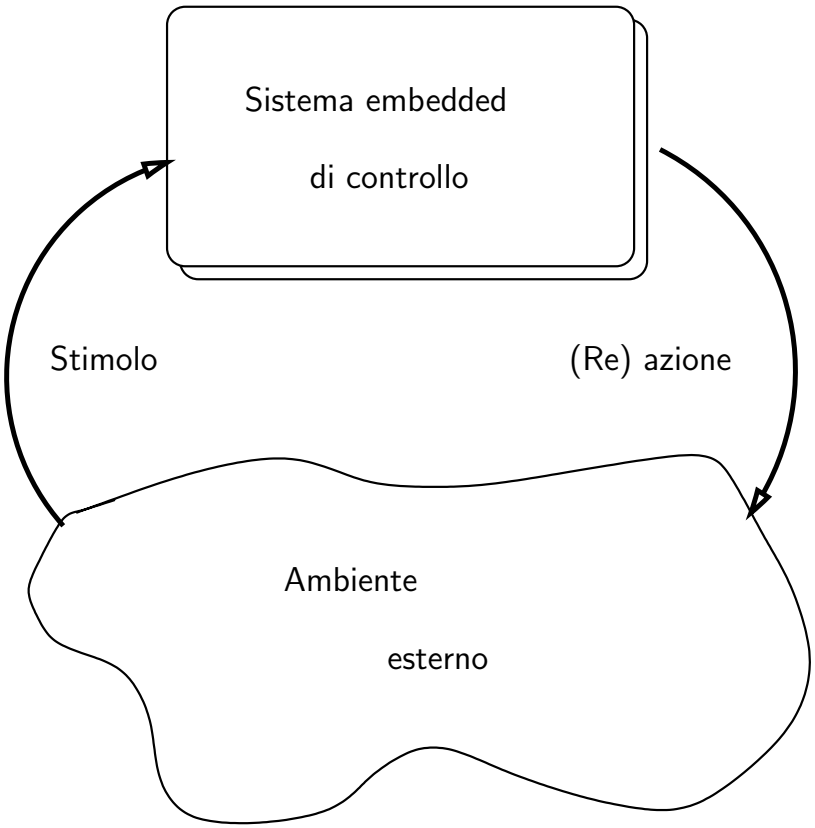
Sistemi reattivi



Una prima qualità fondamentale è la **reattività** agli stimoli esterni.

- Il sistema reagisce agli stimoli esterni in tempo utile.

Sistemi reattivi



Una prima qualità fondamentale è la **reattività** agli stimoli esterni.

- Il sistema reagisce agli stimoli esterni in tempo utile.
- La *velocità* dipende dalle costanti di tempo dell'ambiente esterno.

Velocità

Comprare un hardware più veloce?

- Un sistema ultra veloce va sempre bene.
- Se avessimo un hardware infinitamente veloce non ci sarebbero problemi di real-time!
- Indipendentemente dal sistema operativo!

Un sistema *troppo veloce* può portare altri problemi . . .

Velocità

Comprare un hardware più veloce?

- Un sistema ultra veloce va sempre bene.
- Se avessimo un hardware infinitamente veloce non ci sarebbero problemi di real-time!
- Indipendentemente dal sistema operativo!

Un sistema *troppo veloce* può portare altri problemi . . .

- Costo
- Surriscaldamento
- Fault-robustness

Ottimizzare l'hardware

- Il problema del costo può essere più o meno importante
- dipende dal tipo di prodotto
- per prodotti prodotti e venduti in alti volumi, alcuni centesimi di euro sul singolo prodotto possono fare la differenza.
- Se si riesce a fare tutto con un sistema *piccolo*, il costo per unità si riduce e l'azienda è più contenta.
- A parità di hardware, utilizzare un sistema operativo o un altro può fare la differenza in termini di prestazioni.

Sistema reattivo

Ma cosa deve fare un sistema real-time (reattivo)?

1. Rispondere a vari stimoli esterni;

Sistema reattivo

Ma cosa deve fare un sistema real-time (reattivo)?

1. Rispondere a vari stimoli esterni;
2. Tali stimoli arrivano con intervalli temporali diversi;

Sistema reattivo

Ma cosa deve fare un sistema real-time (reattivo)?

1. Rispondere a vari stimoli esterni;
2. Tali stimoli arrivano con intervalli temporali diversi;
3. Ogni stimolo richiede un'azione diversa . . .

Sistema reattivo

Ma cosa deve fare un sistema real-time (reattivo)?

1. Rispondere a vari stimoli esterni;
2. Tali stimoli arrivano con intervalli temporali diversi;
3. Ogni stimolo richiede un'azione diversa . . .
4. . . . con tempi di risposta diversi;

Sistema reattivo

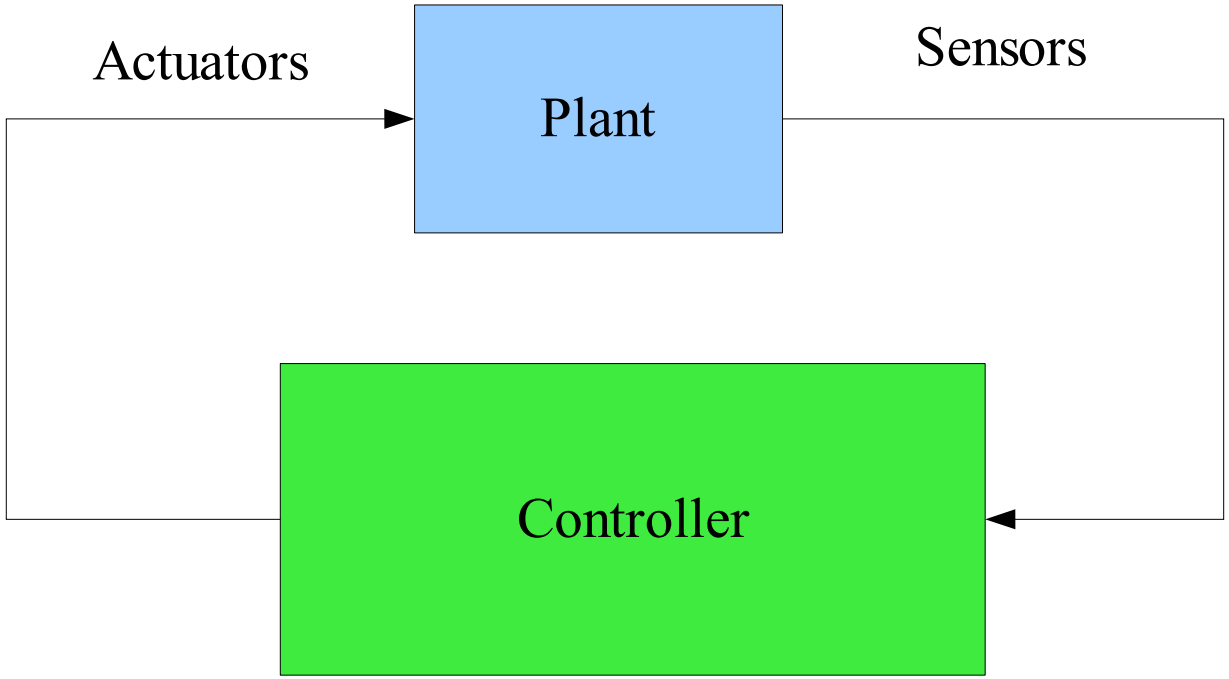
Ma cosa deve fare un sistema real-time (reattivo)?

1. Rispondere a vari stimoli esterni;
2. Tali stimoli arrivano con intervalli temporali diversi;
3. Ogni stimolo richiede un'azione diversa . . .
4. . . . con tempi di risposta diversi;
5. In generale, gli stimoli e le reazioni possono accavallarsi.

Outline

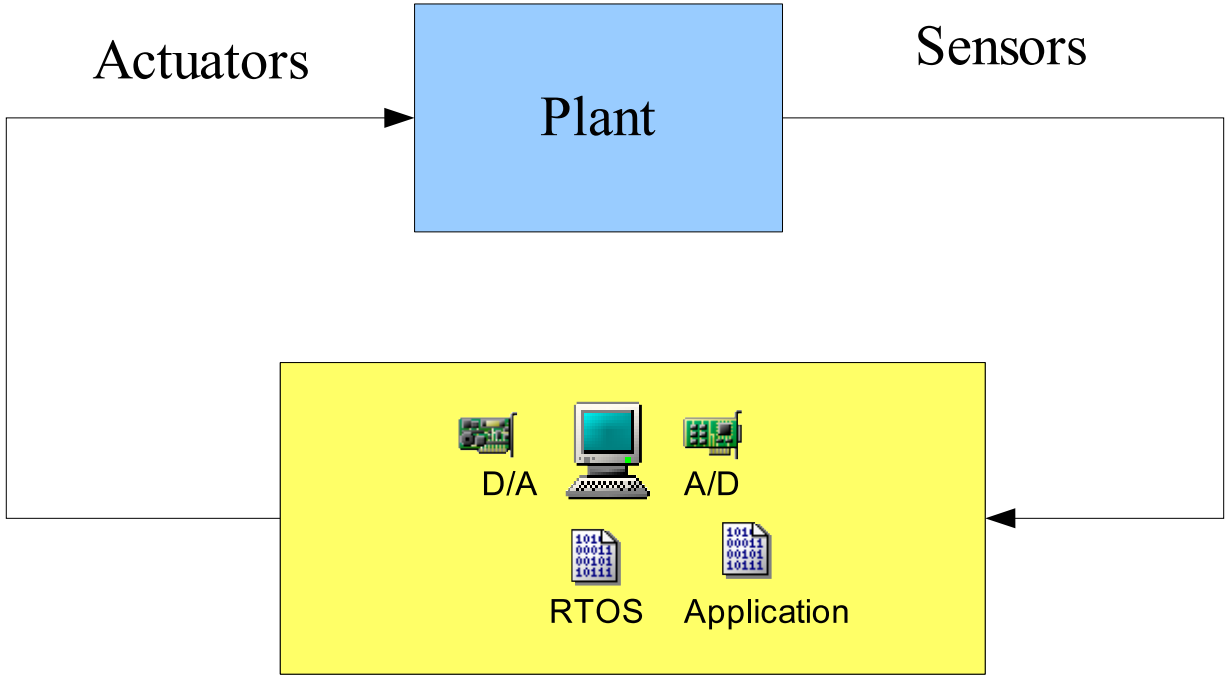
Sistemi di controllo

Un generico sistema di controllo ha la solita struttura ben conosciuta:



Sistema di controllo real-time embedded

Nel caso di un *Embedded Control System* (ECS), il controllore viene implementato tramite un calcolatore che esegue del software di controllo.



Vantaggi: programmabilità, estendibilità, potenza computazionale.

Sistemi embedded

Un sistema embedded differisce da un normale PC in quanto:

- E' nascosto nel sistema, e non viene percepito come un calcolatore. Esempi:
  - Controllore di un aeroplano;
  - Controllo motore in un automobile;
  - Controllore elettrodomestico.
- Si possono aver realizzazioni solo HW oppure a HW e SW.

Problemi nella realizzazione dei controllori embedded

- Discretizzazione del tempo:
  - potrebbe essere necessario avere più rate di campionamento, uno per ogni tipo di ingresso.
- Discretizzazione di input e output.
- Ritardi introdotti dall'implementazione.
- Problemi di programmazione:
  - device drivers;
  - presenza di tante attività da svolgere in concorrenza.
- Analisi e test:
  - Conformità tra specifica ed implementazione,
  - analisi di performance,
  - test di correttezza.

Problematiche nella realizzazione di un ECS

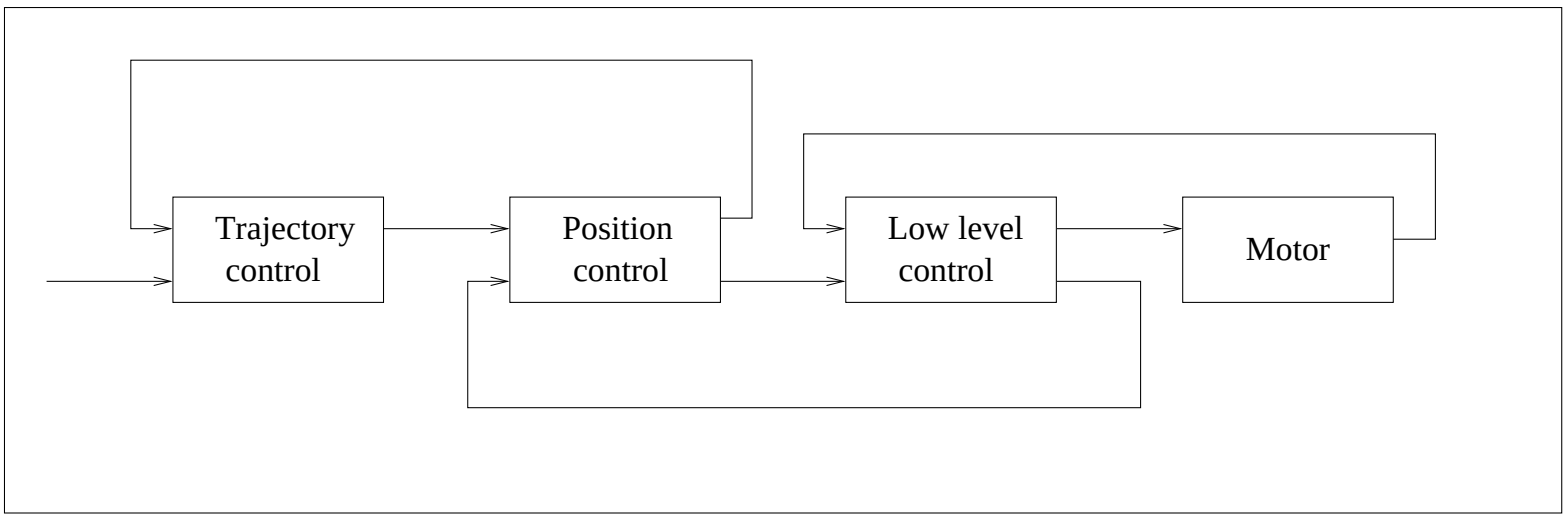
Il sistema deve essere in grado di rispondere agli eventi esterni (sistema *reattivo*) entro un certo tempo limite (sistema real-time).

- **Esempio** in un sistema di controllo con periodo di campionamento T, bisogna:
  - campionare gli ingressi;
  - calcolare la funzione di controllo;
  - effettuare l'attuazione;

  entro T istanti di tempo. In questo semplice caso, abbiamo un solo periodo di campionamento, tutti gli ingressi vengono campionati allo stesso istante. La struttura è ciclica.
- **Problema** cosa succede se ci sono più intervalli di campionamento? Ad esempio, può darsi che ingressi diversi abbiano bisogno di essere campionati a frequenze diverse.

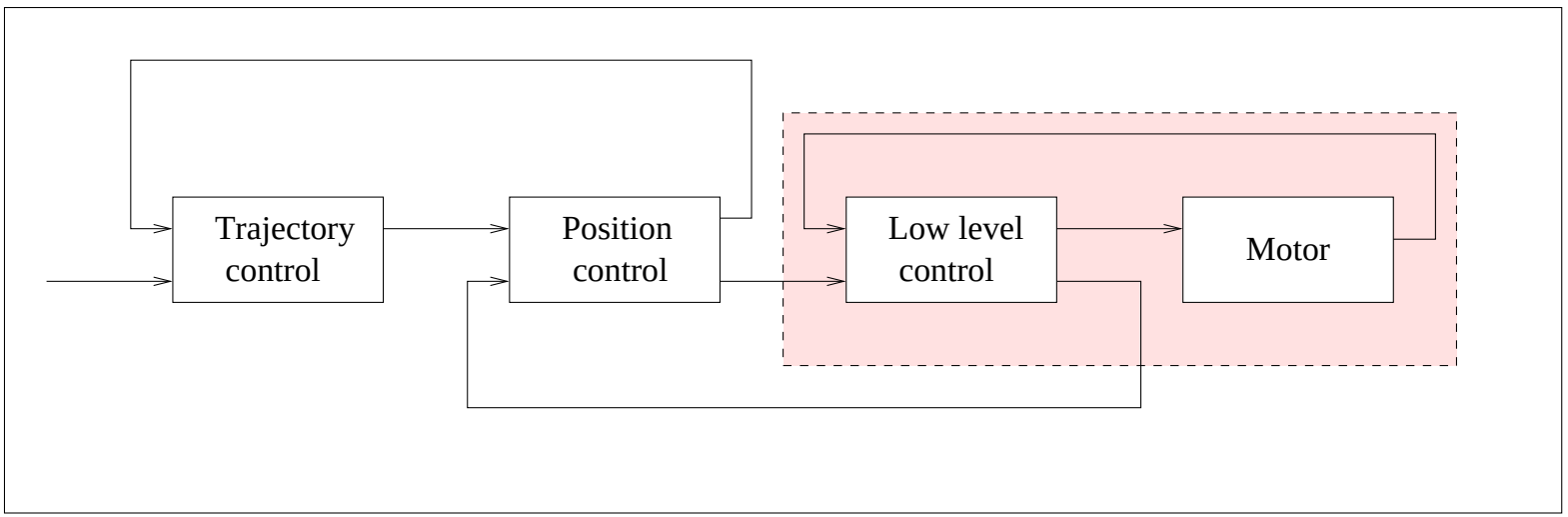
Sistemi di controllo multi-livello

- **Esempio** Consideriamo un sistema di controllo a due livelli per un robot mobile:
  - Livello di controllo dei motori: servono frequenze di campionamento molto elevate.
  - Controllo di alto livello: serve a controllare la direzione di marcia del robot in modo da raggiungere un certo obiettivo. Servono frequenze più basse.



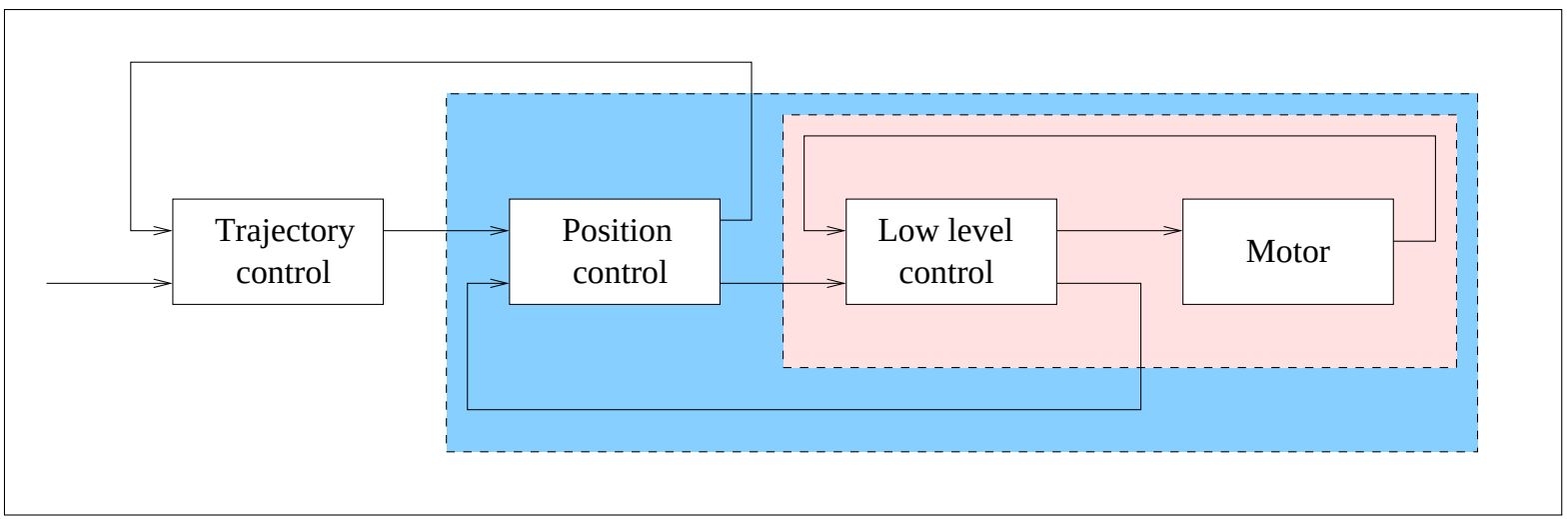
Sistemi di controllo multi-livello

- **Esempio** Consideriamo un sistema di controllo a due livelli per un robot mobile:
  - Livello di controllo dei motori: servono frequenze di campionamento molto elevate.
  - Controllo di alto livello: serve a controllare la direzione di marcia del robot in modo da raggiungere un certo obiettivo. Servono frequenze più basse.



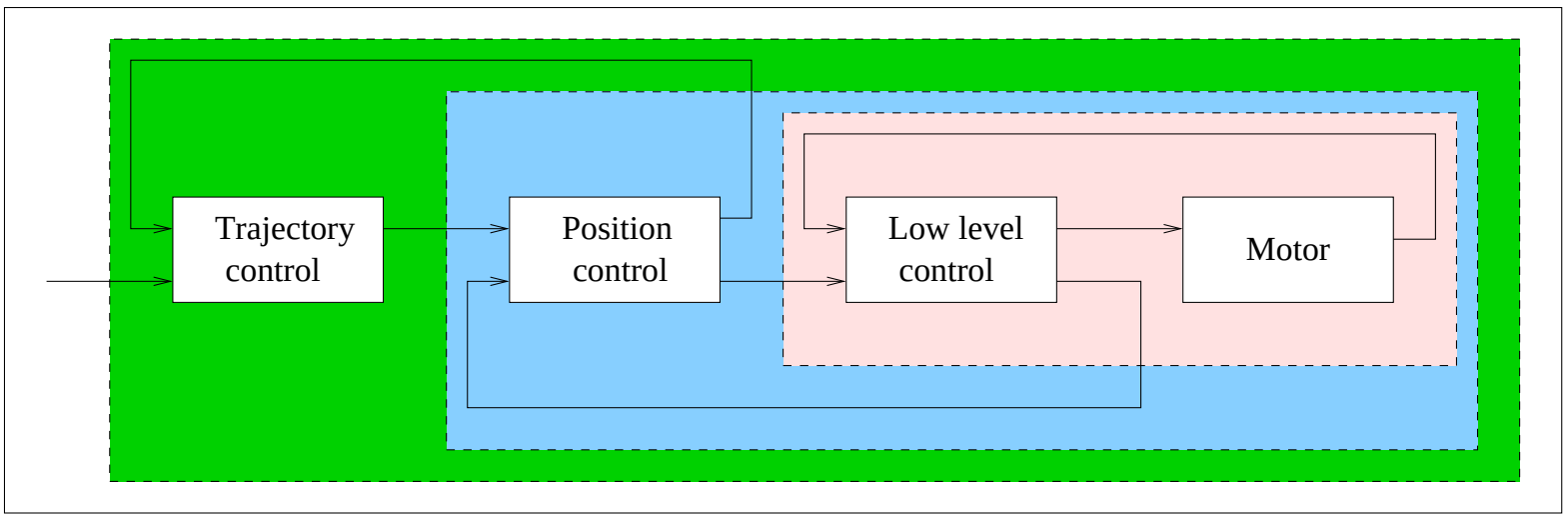
Sistemi di controllo multi-livello

- **Esempio** Consideriamo un sistema di controllo a due livelli per un robot mobile:
  - Livello di controllo dei motori: servono frequenze di campionamento molto elevate.
  - Controllo di alto livello: serve a controllare la direzione di marcia del robot in modo da raggiungere un certo obiettivo. Servono frequenze più basse.



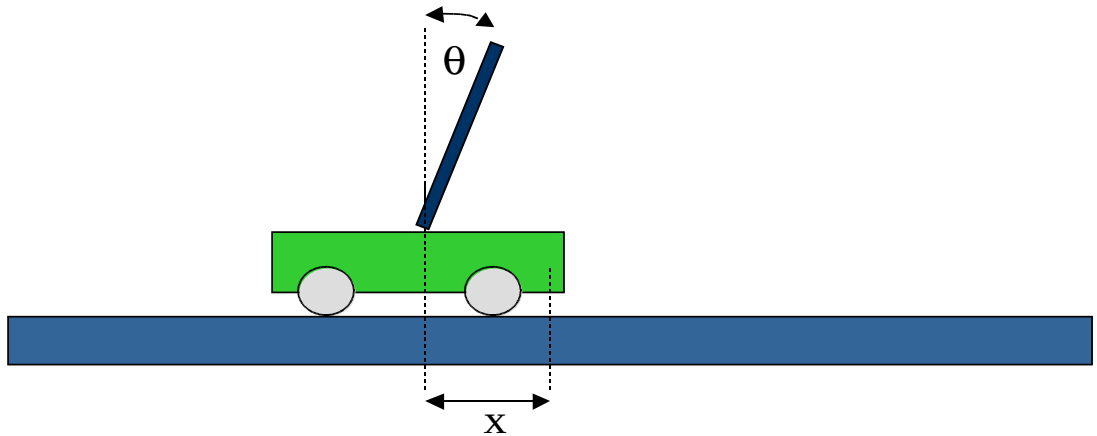
Sistemi di controllo multi-livello

- **Esempio** Consideriamo un sistema di controllo a due livelli per un robot mobile:
  - Livello di controllo dei motori: servono frequenze di campionamento molto elevate.
  - Controllo di alto livello: serve a controllare la direzione di marcia del robot in modo da raggiungere un certo obiettivo. Servono frequenze più basse.



Sistemi di controllo multi-rate

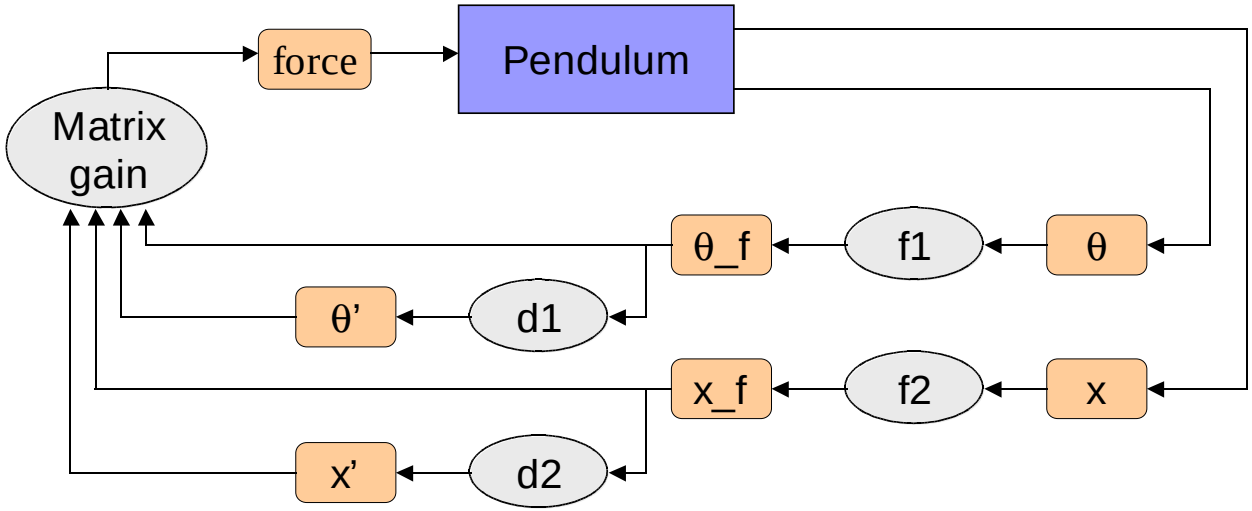
- Come esempio di controllo multi-rate, consideriamo un semplice pendolo inverso;



- sensori:
  - angolo θ: tramite un potenziometro;
  - distanza x dal riferimento: tramite una telecamera;
- attuatore: tramite un motorino, imponiamo un impulso F verso destra o verso sinistra.

Struttura dell'applicazione "pendolo inverso"

- La struttura dell'applicazione è la seguente:



- Ci sono due cicli a frequenza diversa;
  - Il ciclo di lettura della posizione è di 40ms (la telecamera campiona a 25 fps). Quindi il tempo di campionamento è imposto dall'esterno;
  - Il ciclo di lettura del potenziometro è fatto con un convetitore A/D e può essere selezionato anche a 1 ms. (1)(2).

Event Triggered or Time Triggered?

Gli eventi nel nostro sistema si possono dividere in due tipi:

- **time triggered** A intervalli periodici ben determinati, il sistema mette in esecuzione una certa attività.
  - Esempio: campionamento di variabili continue (velocità, posizione, ecc.).

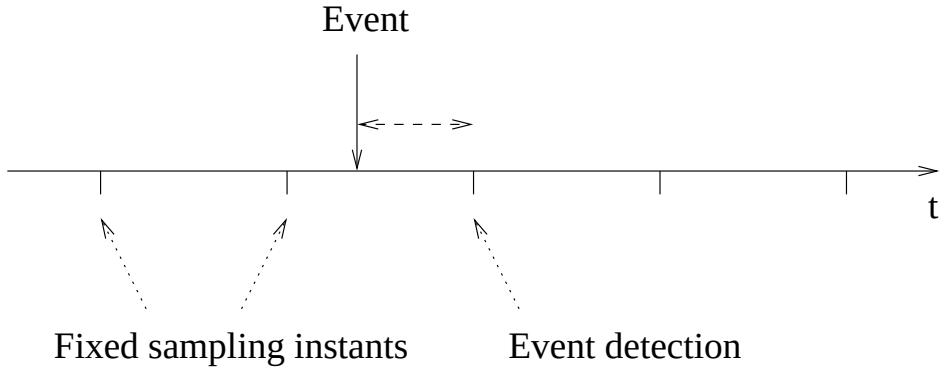
  Di solito una tale struttura periodica viene implemetata tramite servizi del sistema operativo stesso (usando un dispositivo di timer interno)
- **event triggered** si tratta di eventi esterni che occorrono sporadicamente, e a cui il sistema deve dare una risposta (reazione).
  - Esempi: in un termostato, la temperatura scende sotto la soglia $\Rightarrow$ si accende la caldaia.

  Di solito (ma non sempre) sono conseguenza di un evento esterno.

Time Triggered

Ci sono due *filosofie* di base:

- **Approccio Time Triggered** anche gli eventi esterni non periodici vengono considerati solo ad istanti discreti di tempo. Questo comporta un ritardo aggiuntivo nella risposta all'evento dovuto al passo di campionamento.



Tale approccio è più semplice da studiare dal punto di vista del controllo (il tempo è discretizzato). Il problema è che non sempre si riesce a discretizzare il tempo con passo di campionamento costante.

Event Triggered or Time Triggered

- **Approccio Event Triggered**: Anche gli eventi interni di tipo temporizzato vengono considerati come eventi esterni a cui rispondere. In pratica gli eventi periodici vengono considerati come casi particolari degli eventi aperiodici. Tale approccio è più difficile da modellare ed analizzare teoricamente.

In realtà, la distinzione è più concettuale che pratica!

Data flow and control flow

Un'altra distinzione utile è fra *piano di del flusso dati* e *piano del flusso di controllo*. Per capire tale distinzione, consideriamo il seguente esempio:

- In un aeroplano, distinguiamo le fasi di rollaggio sulla pista, decollo, volo in quota, manovra di avvicinamento, atterraggio. Per ognuna di queste fasi (stati del sistema), bisognerà applicare degli algoritmi di controllo diversi.

Naturalmente, la realtà è molto più complessa, e un aereo possiede molti più stati di quelli elencati.

Diagramma degli stati

Per prima cosa, si disegna un *diagramma degli stati* della nostra applicazione.

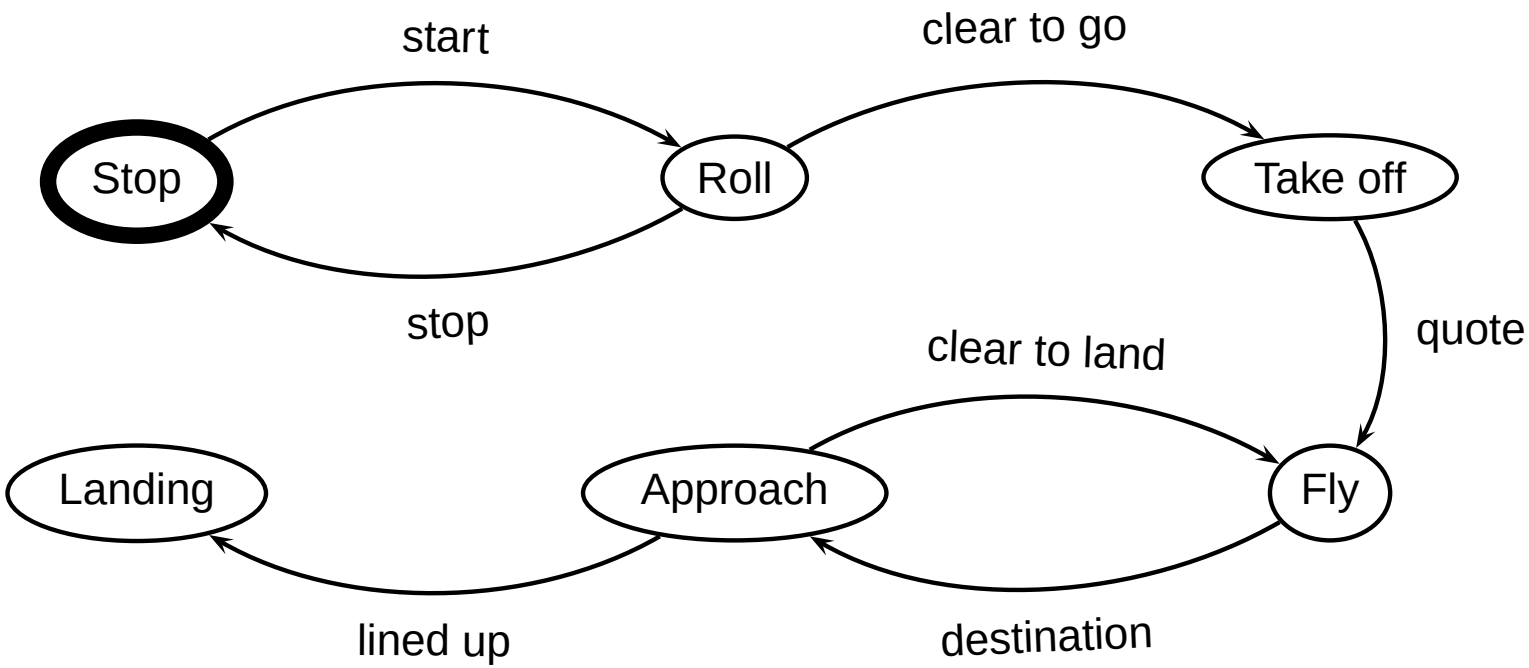


Diagramma degli stati

- Si cambia da uno stato all'altro in seguito ad un *evento* (interno al sistema, o esterno). Per esempio, l'evento *clear to go* è esterno (viene dalla torre di controllo).
- In seguito all'arrivo degli eventi si cambia stato e anche algoritmo di controllo.
- nel cambiare algoritmo di controllo bisogna stare attenti a tanti problemi (ritardo nel cambio dell'algoritmo, continuità degli output, stabilità del sistema, ecc.).
- Il sistema complessivamente è un sistema ibrido!
- I diagrammi di stato verranno approfonditi nel resto del corso.

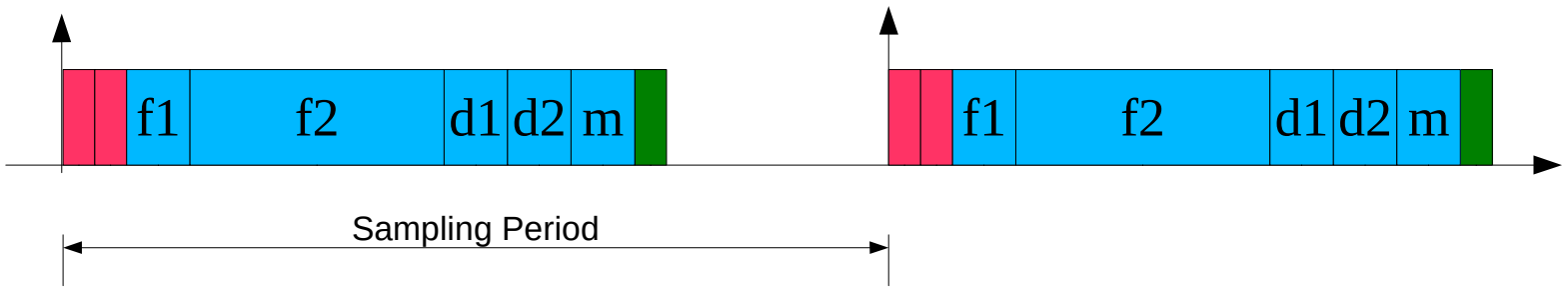
Piano di controllo e piano di flusso dei dati

- Il *piano di flusso dei dati* è quello che siamo abituati a vedere dalla teoria del controllo:
  - un flusso ciclico di campionamento, controllo, attuazione;
- Il *piano di controllo* regola il passaggio da uno stato all'altro del sistema.
  - Schematizzato dal diagramma degli stati di cui prima;
- In realtà, tutti i sistemi hanno un piano di controllo più o meno complesso, con cui bisogna avere a che fare.
- Si potrebbe dire, schematizzando molto che l'approccio time-triggered è più adatto alla progettazione del piano di flusso dei dati;
- l'approccio event-triggered è più adatto al piano di controllo.

Outline

Sistemi in tempo reale

- Per comprendere appieno le problematiche di un sistema embedded in tempo reale, proviamo a pensare come si potrebbe implementare l'applicazione "pendolo inverso".
- Prima ipotesi:
  - Un ciclo che periodicamente legge i sensori, calcola la legge di controllo, effettua l'attuazione.

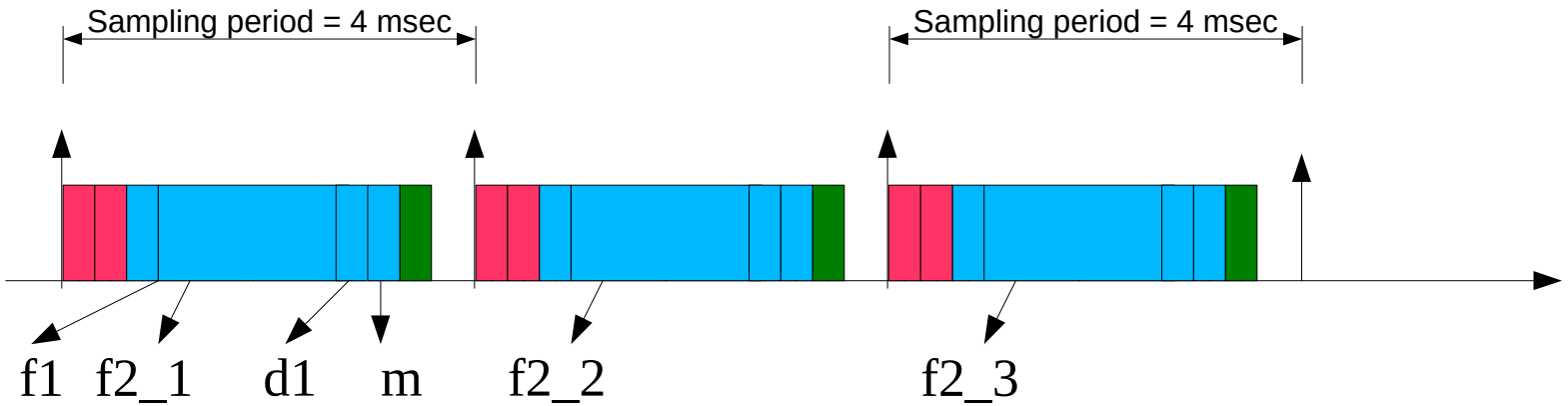


Problemi

- L'approccio precedente non è attuabile in questo caso:
- Infatti, l'algoritmo per riconoscere l'immagine e calcolare la posizione x è molto complesso, e sul PC di riferimento prende 20 ms
- Usare un tempo di campionamento superiore a 10 msec però rende il sistema instabile!
- Soluzione: usare due frequenze di campionamento;
  - Il sensore di angolo lo campioniamo a 4 msec;
  - il sensore di posizione lo campioniamo a 40 msec;
  - l'attuazione la facciamo ogni 4 msec usando il vecchio x.

Soluzione statica (schedulazione “a tabella”)

- Spezziamo la funzione $f2$ in tante funzioni $f2_1$, $f2_2$, ...
- ognuna delle sottofunzioni deve durare non più di 2 msec.



- Problema: non è semplice spezzare le funzioni in questo modo!
- Non è facilmente portabile o estendibile. Se cambiamo qualcosa, non funziona più niente!
- Fino a 10-15 anni fa, nel settore aerospaziale si faceva così!

Concorrenza

- Dato che i due cicli sono indipendenti, possiamo usare la concorrenza;
- Nei moderni sistemi operativi, due o più thread di esecuzione possono eseguire in *concorrenza*;
  - programma SEQUENZIALE: le istruzioni eseguono uno dopo l'altra;
  - programma CONCORRENTE: composto da due o più thread, le cui istruzioni i thread possono eseguire in parallelo, o in un ordine qualsiasi;

Concorrenza - II

- In un programma CONCORRENTE:
  - Se il numero dei processori è maggiore di 1, allora i thread possono eseguire in parallelo;
  - altrimenti *alternano* la loro esecuzione sul processore singolo secondo certe regole (*algoritmo di scheduling*).
- I sistemi operativi moderni forniscono tutti un supporto per la programmazione concorrente.
- I sistemi operativi *real-time* (RTOS) permettono di controllare l'ordine di esecuzione in maniera che siano rispettati i *vincoli temporali*.

Pendolo inverso: Implementazione concorrente

Una possibile implementazione concorrente del pendolo inverso è la seguente:

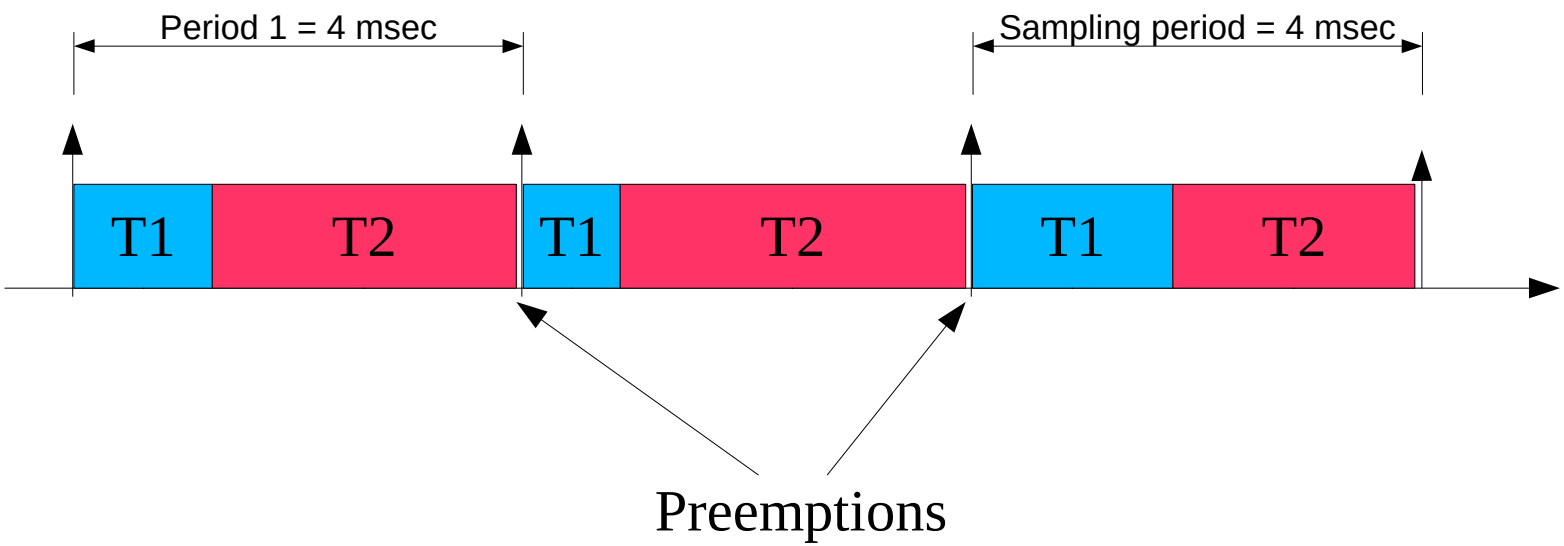
```
void * mytask1(void *) {
  while (1) {
    f1();
    d1();
    matrix();
    actuate();
    task_endcycle();
  }
}
```

```
void * myTask2(void *) {
  while (1) {
    f2();
    d2();
    task_endcycle();
  }
}
```

- Al task myTask1() assegnamo priorità “alta” e periodo 4 msec,
- al task myTask2() assegnamo priorità “bassa” e periodo 40 msec.

Schedulazione

- In un sistema real-time gestito a priorità, l'ordine di esecuzione sarebbe il seguente:



Analisi di schedulabilità

- Come facciamo a sapere che tutto andrà bene?
- Requisiti:
  - Che tutti i task abbiano finito di eseguire quando arriva la loro prossima attivazione;
  - Il task myTask1() deve terminare ogni ciclo entro 4 msec;
  - Il task myTask2() deve terminare ogni ciclo entro 40 msec;
- Questi requisiti sono le “deadline”
- Ci sono altri tipi di requisiti
  - Ritardo “end-to-end”
  - limitazione sul jitter di uscita o di ingresso;
  - ecc.

Analisi di schedulabitià - II

- L'analisi di schedulabilità si occupa di stabilire se tutto andrà bene anche nel caso peggiore;
- Se definisce " tempo di calcolo di caso peggiore" (WCET) il massimo tempo di calcolo richiesto da un task per completare una istanza;
- Nel caso precedente, un test di schedulabilità molto semplice è quello sul "carico"
- Carico:

$$U = \sum \frac{C_i}{T_i}$$

- Se il carico è minore di 1, i due task di prima rispetteranno i requisiti;
- Nell'esempio, $U = 1/4 + 20/40 = 0.75 < 1$, quindi il sistema *è schedulabile*.

Implementazione e analisi

- In realtà l'analisi è spesso più complicata
- Dobbiamo prendere in considerazione:
  - L'influenza dei "device driver" e delle interruzioni
  - Gli istanti di campionamento
  - I ritardi nell'attuazione
  - La correttezza delle sincronizzazioni
  - etc.
- In questo corso cercheremo di dare una panoramica dei problemi e delle possibili soluzioni
- Il campo è vasto e non ancora completamente esplorato: questo corso non può esaurire tutte le problematiche!