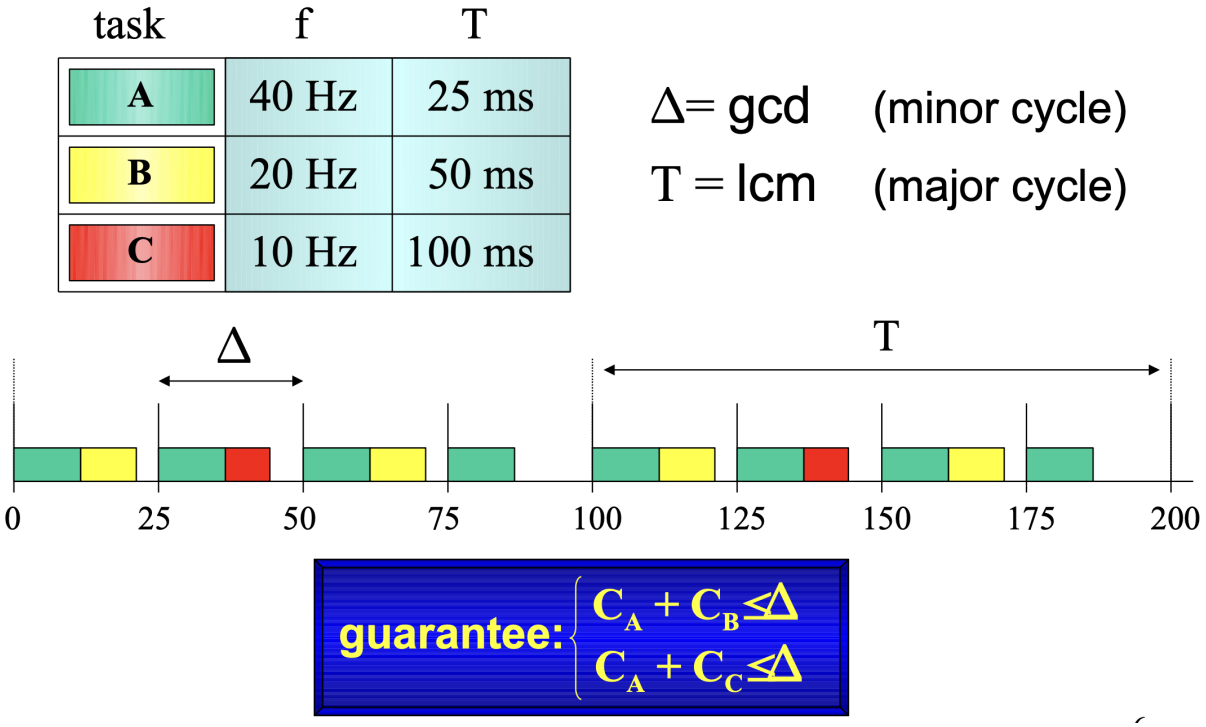
Scheduling of periodic tasks

Task periodici

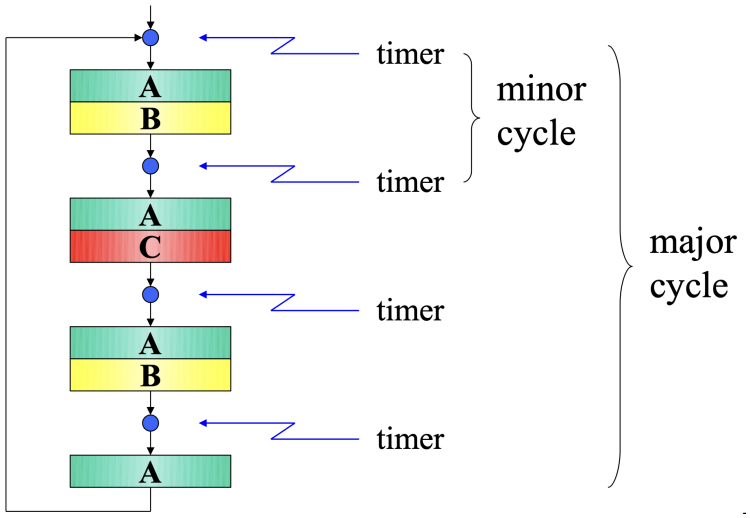
Ci concentriamo su uno scheduling offline, ovvero decide tutto prima di partire, quindi molto predicibile.

L'asse dei tempi è diviso in intervalli della stessa lunghezza δ chiamati **TIME SLOTS.** Ogni task è assegnato ad uno o più slot in modo da rispettare i propri vincoli.

L' esecuzione in ogni slot è attivata da un timer, ad ogni tick arriva un interrupt che causa un context switch ad un altro task.



- Controllore del radar che va a 40 Hz, ogni 25 secondi mi da un sample da computare e così via, sono tutti periodici
- Δ → **Greatest Common Divisor o MINOR CYCLE** dei periodi dei task coinvolti. Massimo comun divisore tra 25, 50 e 100? 25.
- T → **Least Common Multiple o MAJOR CICLE** dei periodi dei task coinvolti. Periodo con il quale il mio schedule globale si ripete nel tempo.
- Con questo setup ho una grande predicibilià e non avrò mai problemi a meno che l'arrivo dei task non sia desincronizzato tra loro, ovvero se ognuno ha un suo clock e non iniziano tutti contemporaneamente.



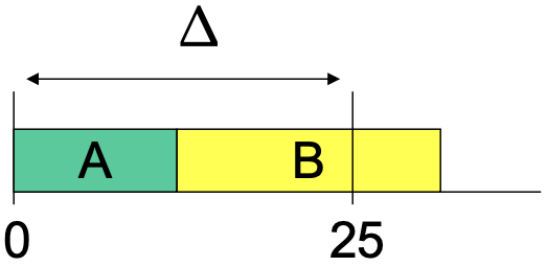
Vantaggi → è praticamente un dispatcher, ho poco overhead, controllo meglio il jitter

Svantaggi → debole in caso di imprevisti/overload, difficile da espandere nel caso si vogla aggiungere un task o cambiarne uno, difficile da inserire task molto lunghi, sorgono problemi per processi che richiedono un tempo variabile per eseguire

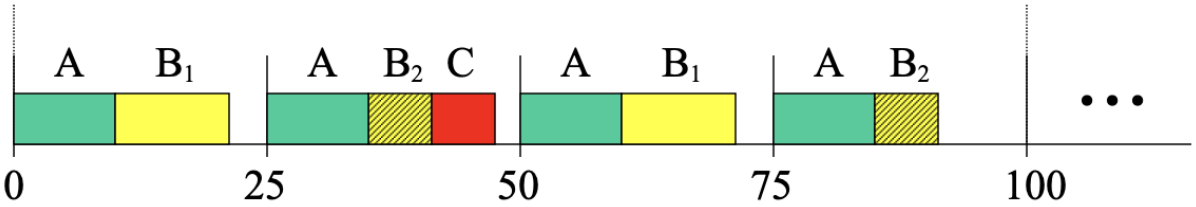
- **Problemi in caso di overload:** che faccio se un task per eseguire ci ha messo 20ms al posto dei programmati 10ms? Devo evitare l'effetto domino, ovvero che il ritardo di un task fa missare la deadline a tutti gli altri in cascata. Posso abortire il task, ma può lasciare il sistema in stato inconsistente

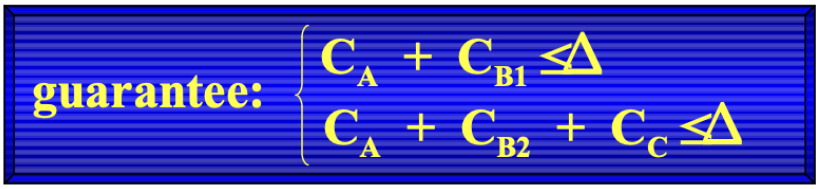
Espandibilità → se uno o più task di quelli che abbiamo a disposizione esige di essere aggiornato, questo può portare ad una modifica di tutti gli altri task.

example: B is updated but $C_A + C_B > \Delta$

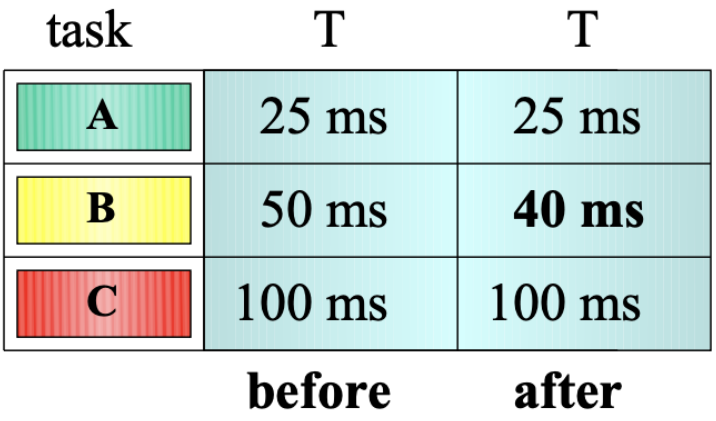


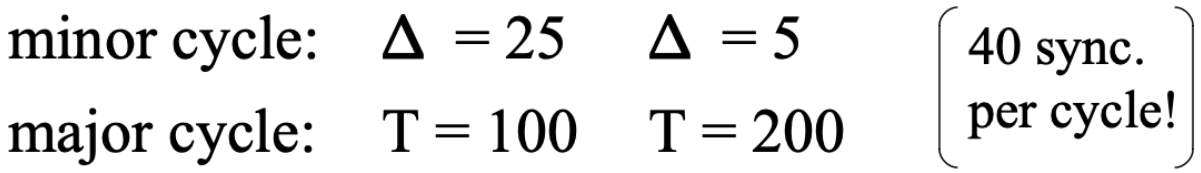
- Riprendendo il caso di prima, in questo caso devo spezzare B in 2 sotto-task.

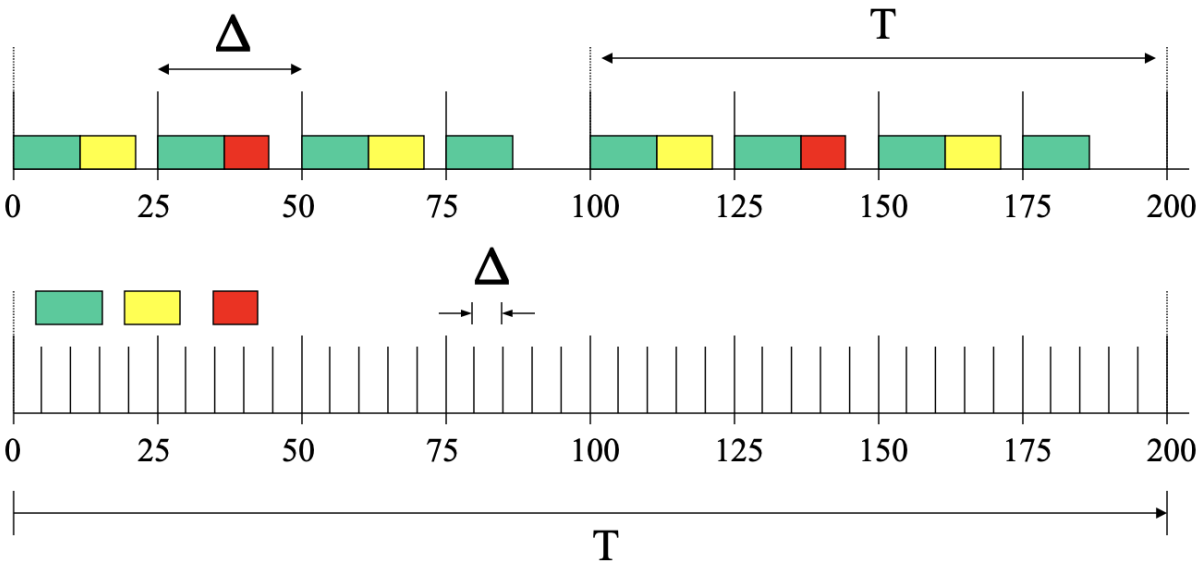




- In caso di aggiorgamento della frequenza del task B, da 50ms a 40ms, succede un casino in quanto il minor e major cicle vengono stravolti. Da 5 intervalli di sincronizzazione ne ho 40!

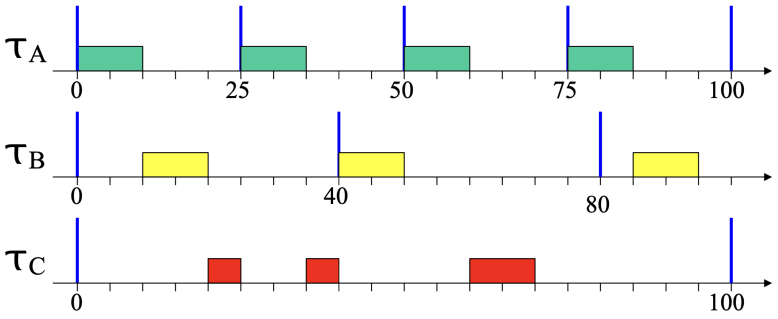




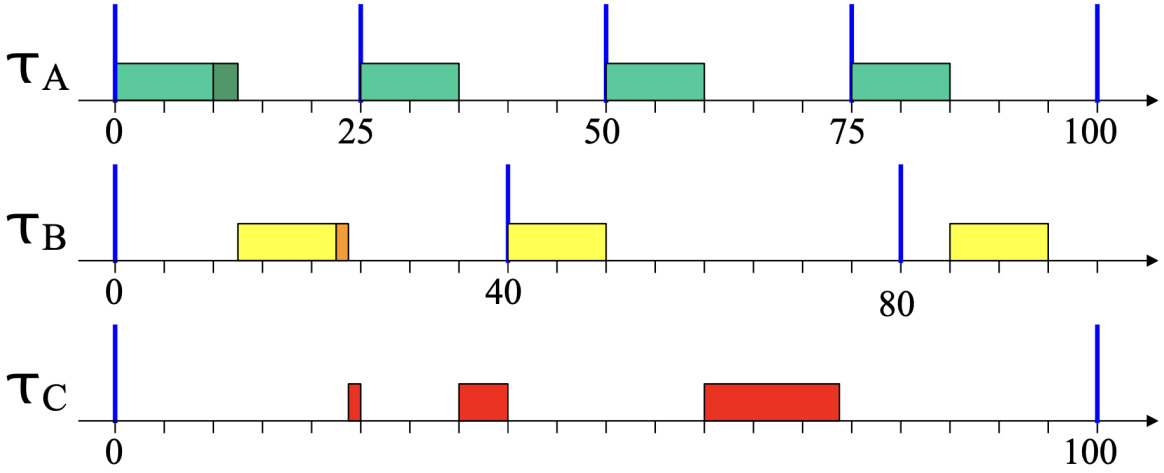


- Tipico schedule che uso in assenza di sistema operativo. Faccio **BARE METAL**, ovvero i miei task vanno direttamente in esecuzione, c'è un while infinito con dei checkpoint (esegui la chiamata ad A, poi B, poi A ecc.)

Come fissiamo le priorità? Conviene assegnarle secondo **RATE MONOTONIC RM** → priotità più alta al task che ha il periodo più piccolo



- 4 cambi di contesto → meno di prima
- Come assegamento di priorità è molto buono e massimizza la schedulabilità, se RM non è in grado di trovare un feasible schedule, allora nessun altro assegnemento di priorità fisse sarà in grado di trovarlo.
- In caso di imprevisti (task che ci mette un po di più del dovuto) o aggiustamenti sulla durata dei task il sistema si autoadatta.



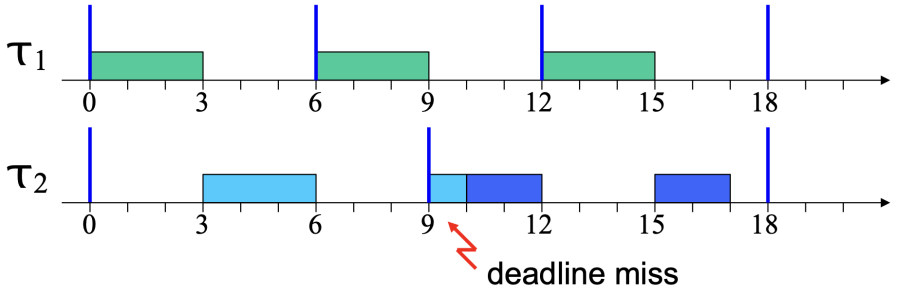
Come verifico la fattibilità/utilizzazione?

Utilizzazione di un task (frazione di tempo per cui lui utilizza il processore su cui esegue) → $U_i = \frac{C_i}{T_i}$, computation time diviso periodo.

Utilizzione del processore (somma delle utilizzazioni di ogni task) → $U_p = \sum_{i=1}^{n} \frac{C_i}{T_i}$

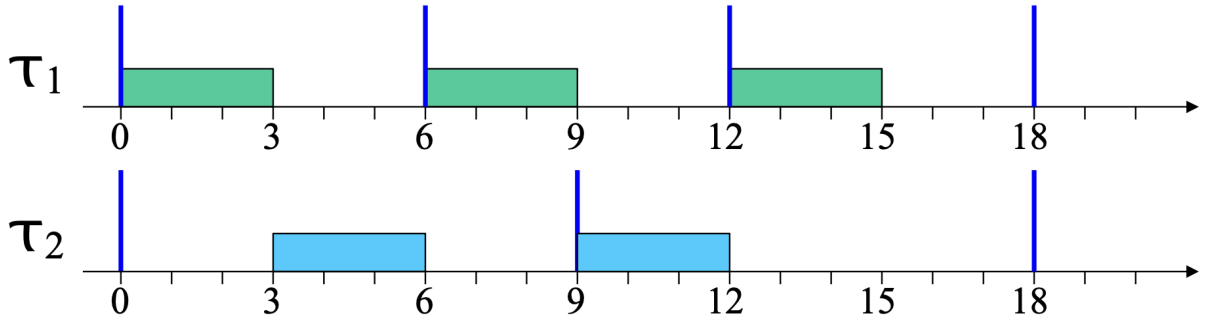
- Se $U_p > 1$ non esiste alcun algoritmo che ti possa dare la schedulabilità in quanto causa sovraccarico ed è condizione necessaria ***MA*** non sufficente
- Non è detto che se $U_p < 1$ RM riuscirà sempre a schedulare il mio task

$$U_p = \frac{3}{6} + \frac{4}{9} = 0.944$$



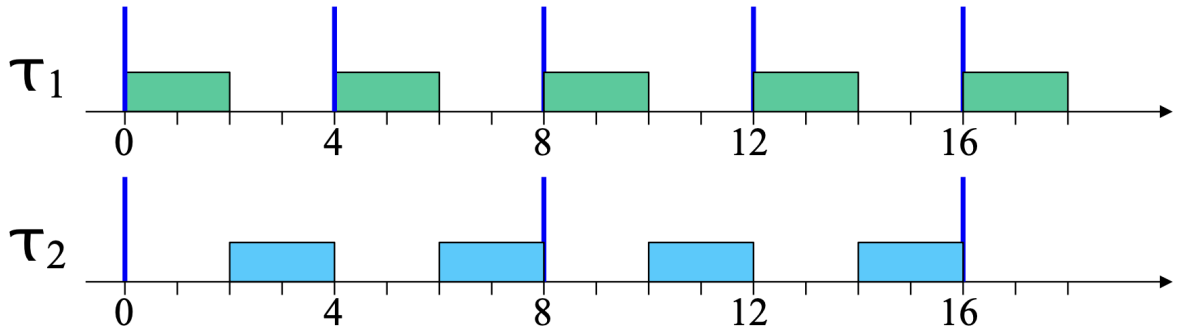
Qual'è l'utilizzazione più piccola che posso raggiungere?

$$U_p = \frac{3}{6} + \frac{3}{9} = 0.833$$



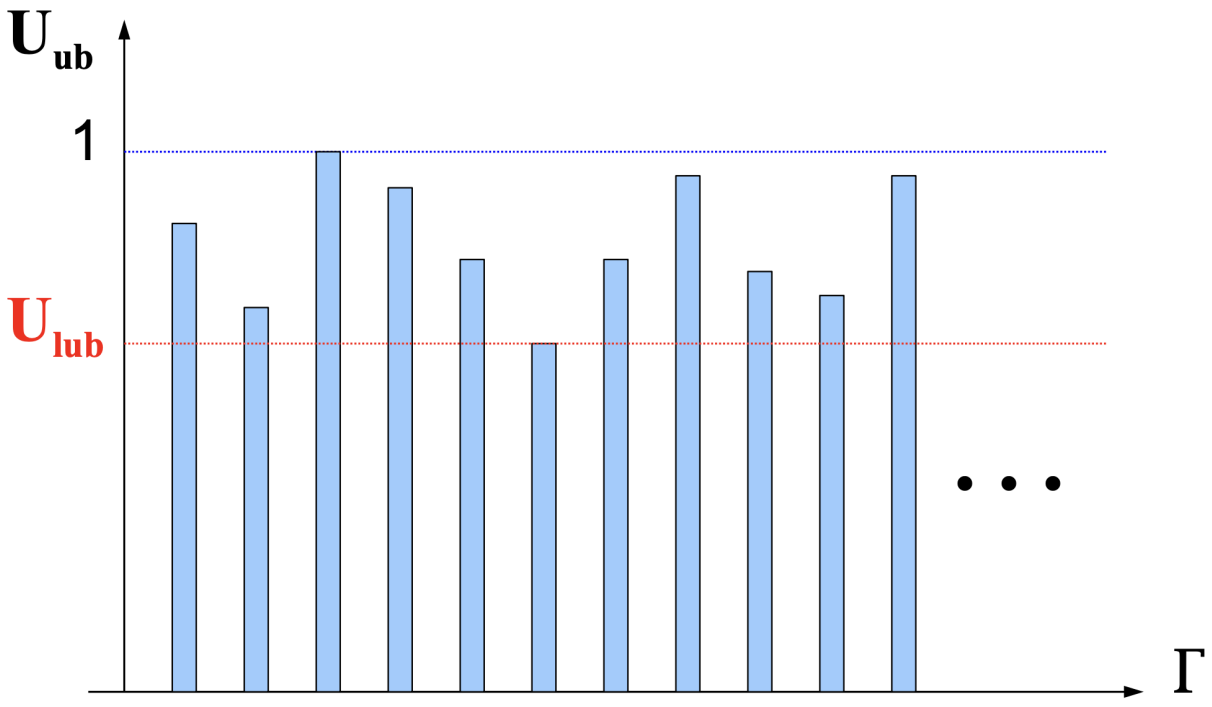
NOTE: if C_1 or C_2 is increased, τ_2 will miss its deadline!

$$U_p = \frac{2}{4} + \frac{4}{8} = 1$$



the upper bound U_{ub} depends on the specific task set

Come si vede dai precedenti due esempi, qualche volta quando ho un utilizzazione minore di 1 posso missare, altre invece nonostante sia uguale ad uno rispetta le deadline. Quale condizione posso usare per determinare la schedulabilità? Sopra 1 certamente non posso averla, ma sotto è tutto variabile.

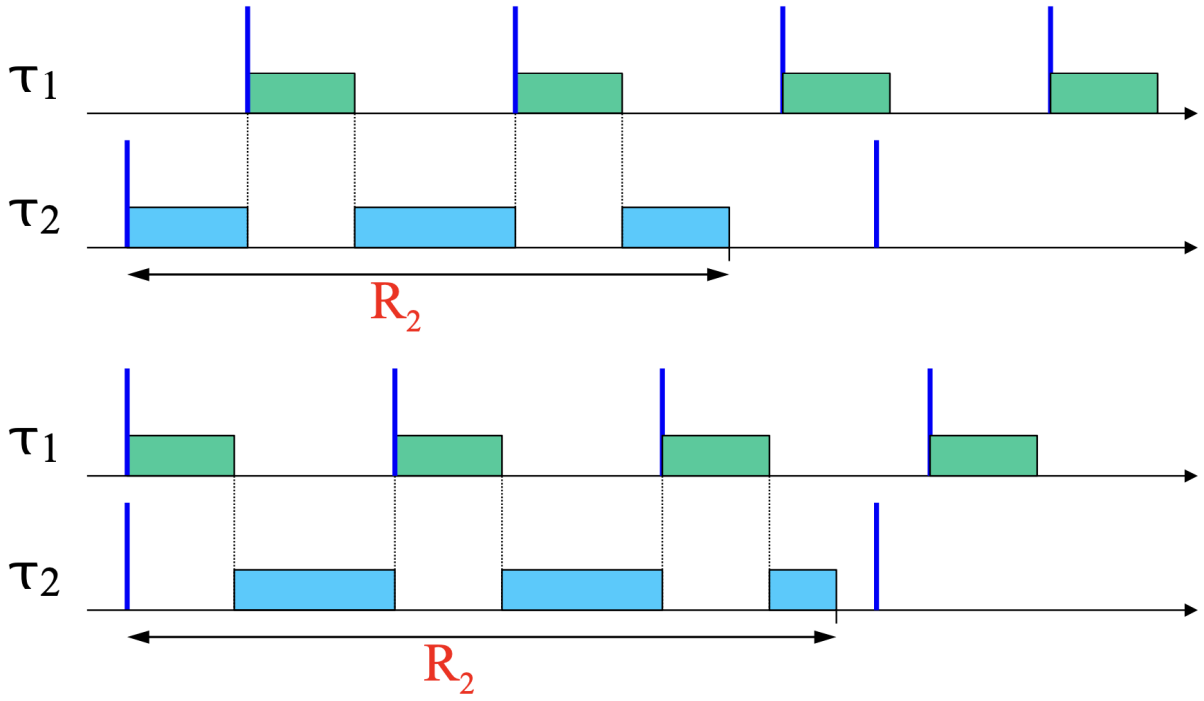


Cerchiamo una soglia di utilizzazione per la quale la schedulabilità è GARANTITA.

$$U_{lub}^{RM} = n(s^{1/n} - 1)$$
$$for\ n \to \infty,\ U_{lub} \to ln2$$

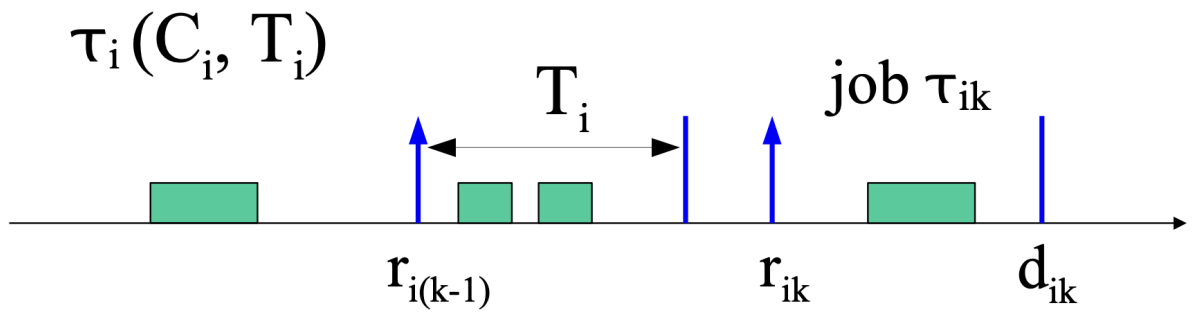
- Dipende dal numero di task.
- Tendendo ad infinito, si arriva ad una utilizzazione di $0,69$ ovvero 69%, quindi se si è sotto questa soglia mi è garantita la schedulabilità.
- Dato un task set, per capire se è schedulabile con RM, prima calcolo l'utilizzazione C_p. Poi verifico che sia minore del Least Upper Bound $C_p \leq n(2^{1/n} - 1)$, in caso positivo è garantita la schedulabilità, in caso contrario no.

Cos'è il **CRITICAL INSTANT**?



Dobbiamo considerare il tempo di risposta peggiore (**WORST CASE** → tutti i task finiscono più avanti possibile), quindi il tempo di risposta più alto di un task lo ottieni quando quel task arriva assieme a tutti i task a priorità più alta di lui.

Task sporadici



- Posso parlare solo di minimo tempo di inter-arrivo. Dopo il k-1esimo job, il prossimo mi può arrivare da un certo momento, non prima ma comunque rispettando la sua deadline relativa.

Il **CRITICAL INSTANT** (response time peggiore) nel caso di task sporadici si ha sempre se i task arrivano tutti sincronizzati e se i task sono esattamenti spaziati col loro tempo di inter-arrivo, cioè se arrivano il prima possibile.

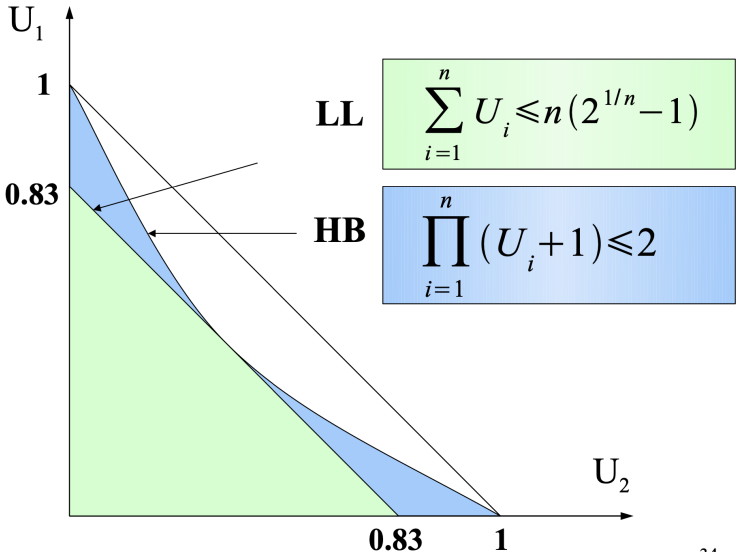
Assumiamo che tutti i job eseguano sempre per il **WCET,** che il minimo tempo T_i sia costante, ***che le deadline siano uguali al periodo*** e tutti i task sono indipendenti (no sezioni critiche, no vincoli sulle risorse ecc). Se esiste

un assegnamento di priorità fisse che mi permette la fattibilità, allora tale fattibilità sarà raggiunta anche con RM (se non ce la fa RM, allora non ce la fa nessuno!!!).

Andiamo a vedere ora un test più raffinato di $C_p \leq n(2^{1/2} - 1)$ (complessità $O(1)$), con complessità più grande $O(n)$ che mi permette di discriminare lo spazio tra 1 e U_{lub}:

$$\sum_{i=1}^{n}(U_i + 1) \leq 2$$

Se faccio la produttoria dell'utilizzazione dei task più uno ed è minore di 2, allora è schedulabile. Al contrario non potrei dire niente, in quanto non è sufficente.

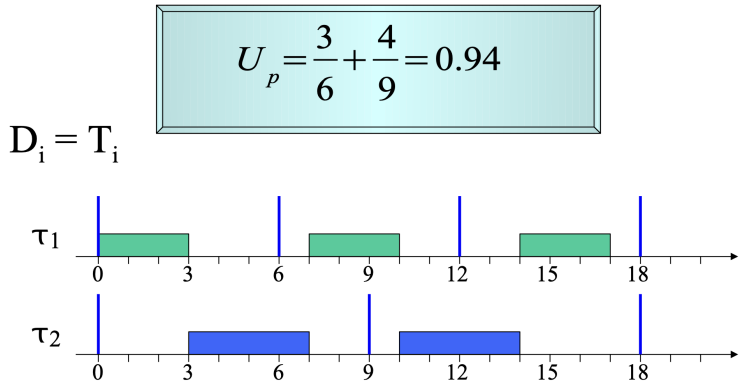


Riesco a mangiare un po più di spazio.

Scheduler a priorità dinamiche

Reminder: EDF schedula in base alla deadline assoluta → priorità maggiore a chi ha la deadline minore.

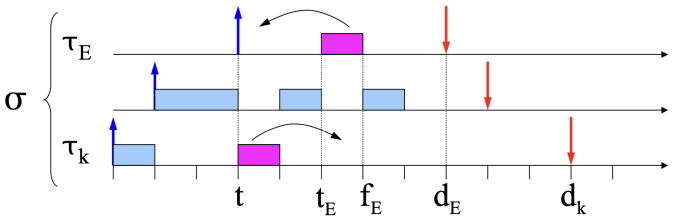
Il least upper bound di taskset schedulabili con EDF è 1, cioè il massimo possibile, posso andare a schedulare al 100% del mio processore. Questa è una condizione sufficente e necessaria.



EDF pertanto è ottimale in assoluto, cioè fra tutti i task schedulabili ritornerà sempre con uno schedule fattibile, se un taskset non è schedulabile da EDF allora non esiste nessuno scheduler che eseguirà quel task system.

Dimostrazione identica ad EDD. (La chiede all'esame)

Assumo per assurdo che esista uno schedule σ diverso da EDF, per il quale un task-set è schedulabile mentre EDF avrebbe missato la deadline.



transforming σ in σ'

$$\begin{cases} \sigma'(t) = \sigma(t_E) \\ \sigma'(t_E) = \sigma(t) \end{cases}$$

feasibility is preserved

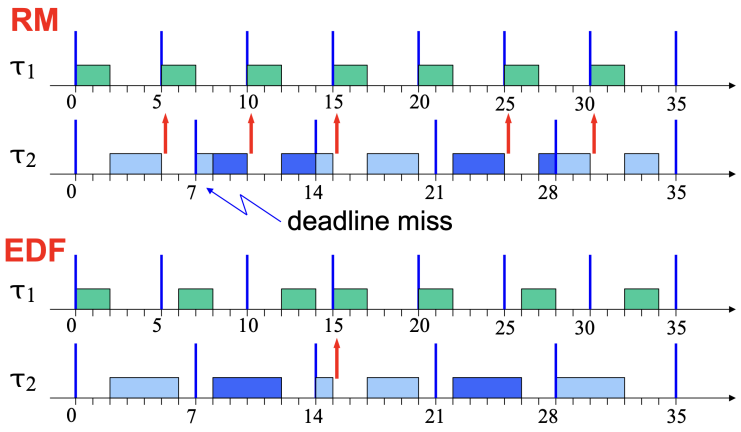
$$f_k' = f_E \leq d_E \leq d_k$$

Dopo aver preso le prime 2 scelte come EDF, arrivo all'istante t dove schedulo τ_k al posto di τ_E, discostandomi così da EDF che avrebbe predilletto l'esecuzione del task con deadline più imminente.

La condizione necessaria e sufficente provata da Liu e Layland è questa: $U_{lub}^{EDF} = 1$, quindi utilizzazione fino al 100% del processore. Quindi un task-set è schedulabile in generale o per EDF se l'utilizzazione è minore o uguale a 1 → $U_p \leq 1$.

Confronto tra RM e EDF

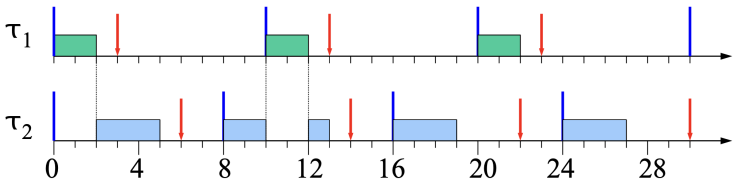
- **EDF →** molto più efficente in quanto abbiamo il 100% di utilizzazione e riduce i cambi di contesto ***MA*** non è molto usato negli scheduler Real-time, dove troveremo quadi sempre **Fixed Priority.** Esso è poco flessibile a **overload**, ovvero quando un task esegue per un periodo maggiore di quanto preventivato. Il jitter ($\max(R_1) - \min(R_2)$) è maggiore di quello di RM.
- **RM →** più facile da implementare, quindi preferito a EDF nei sistemi operativi Real-Time. Inoltre gestisce meglio gli overload, in questo modo i task a priorità più alta sono immuni.



- Linee blu → deadline (una sola deadline miss nel caso di RM)
- Linee rosse → context switch (ce ne sono troppe nel caso di RM)

Fin'ora abbiamo assunto che le deadline fosserro uguali ai periodi, ma potrei voler finire un task prima che arrivi l'istanza successiva, ovvero $D \leq T$, in questo caso l'utilizzazione che intendevamo prima $U = \frac{C_i}{T_i}$ non è più utilizzabile ai fini della schedulabilità. Al suo posto uso $U_i^* = \frac{C_i}{D_i}$. Posso utilizzare ancora $U^* \leq 1$, ma non è più una condizione necessaria e sufficente, è solo sufficente (se non è valida non posso concludere niente, se è maggiore o uguale a 1 non si può concludere la non schedulabilità).

Gli algoritmi di scheduling rimangono **EDF** ($p_i \propto 1/D_i$) e **Deadline Monotonic DM** ($p_i \propto 1/d_i$)(al posto di RM)



The (modified) Utilization Bound is not useful:

$$U_p = \sum_{i=1}^{n} \frac{C_i}{D_i} = \frac{2}{3} + \frac{3}{6} = 1.16 > 1$$

but the task set is schedulable!

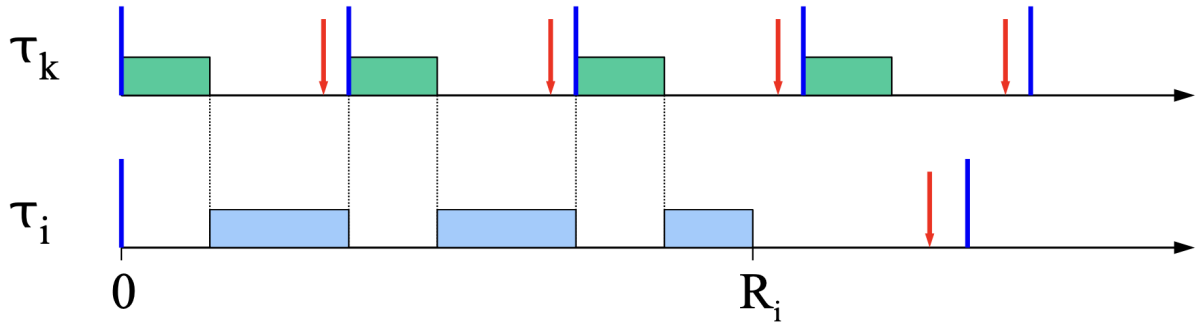
Deadline Monotonic

Dobbiamo andare alla ricerca di altre condizioni al posto di U^* che siano **necessarie e sufficenti.**

Response Time Analysis → Il response time è la somma tra C_i e gli intervalli di tempo in cui sono pronto ad eseguire ma sono fermo perchè c'è un task ad alta priorità in esecuzione (sommati insieme sono detti **INTERFERENZA**)

Se $R_i \leq D_i$ allora ho la schedulabilità, altrimenti concludo che ho un deadline miss, questo algortimo è applicabile solo per Fixed Priority

Come calcolo l'interferenza I_i sul task i-esimo?



Inteferenza da parte del task k-esimo sul task i-esimo → I_{ik} = $\left\lceil \frac{R_i}{T_k} \right\rceil C_k$

In caso di tanti task basta sommarli tutti → $I_i = \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$

Così non trovo ancora il response time R_i in quanto siamo in un equazione che dobbiamo risolvere:

$$R_i = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

Ciò non può essere ricavato facilmente perchè R_i è all'interno del ceiling (approssimazione per eccesso), va risolto iterativamente dove si parte dall'affermare che R_i è sicuramente almeno uguale di C_i (si dice R_i al passo 0 → R_i^0). Poi al posto di R_i nell'equazione metto C_i, se ottengo ancora C_i ho raggiunto il punto fisso dell'interazione, in caso contrario continuo con il passo s-esimo fino ad ottenerlo ($R_i^{(s+1)} = R_i^{(s)}$).

Esercizio: Questo task-set è schedulabile con RM?

	C_i	$D_i = T_i$
τ_1	3	6
τ_2	7	28
τ_3	5	30

Con il tesk basato su utilizzazione ho $U = \frac{3}{6} + \frac{7}{28} + \frac{5}{30} = 0,916$ e $U_{lub} = 3(\sqrt[3]{2} - 1) = 0,779$ → Forse si può fare perchè $U \geq U_{lub}$, sono nel margine di incertezza.

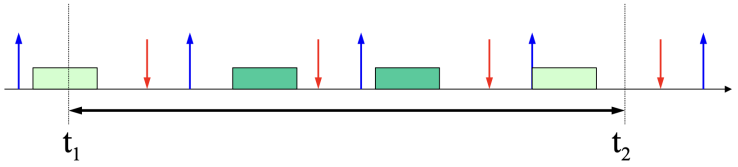
$R_1 = C_1 + 0 = 3 \leq 6$ → perchè non c'è nessuno a priorità più alta.

$R_2 = 7 + \frac{7}{6}3 = 7 + 2 \cdot 3 = 13$
$= 7 + \frac{13}{6}3 = 7 + 3 \cdot 3 = 16$
$= 7 + \frac{16}{6}3 = 7 + 3 \cdot 3 = 16 \leq 28$

$R_3 = 5 + \frac{5}{6}3 + \frac{5}{28}7 = 5 + 1 \cdot 3 + 1 \cdot 7 = 15$
$= 5 + \frac{15}{6}3 + \frac{15}{28}7 = 5 + 3 \cdot 3 + 1 \cdot 7 = 21$
$= 5 + \frac{21}{6}3 + \frac{21}{28}7 = 5 + 4 \cdot 3 + 1 \cdot 7 = 24$
$= 5 + \frac{24}{6}3 + \frac{24}{28}7 = 5 + 4 \cdot 3 + 1 \cdot 7 = 24 \leq 30$

Processor Demand Criterion → corrispondente di EDF dove per ogni intervallo l'ammontare della computazione richiesta dal mio task-set è più piccola dell'intervallo $demand(t) \leq t$

In breve, in un intervallo t_1, t_2 è il tempo di computazione richiesto da tutti i task che hanno un release time contenuto in questo intervallo e deadline prima della fine dell'intervallo.



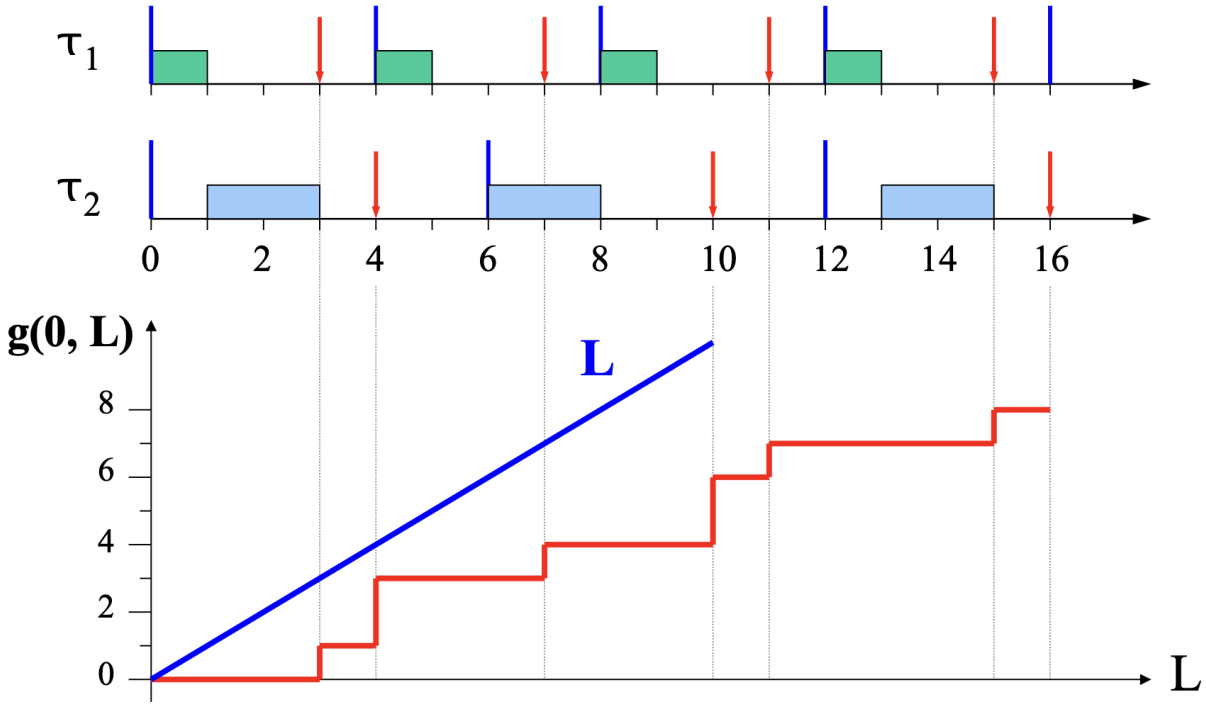
- La demand quindi saranno i due job in verde scuro, in quanto sono interamente dentro l'intervallo insieme alle loro deadline → $g(t_1, t_2) = \sum_{r_i \geq t_i}^{d_i \leq t_2} C_i$ (lo starting time r_i è successivo all'inizio dell'intervallo t_1 e la deadline d_i è prevista prima della fine dell'intervallo t_2). Quest'ultima è massimizzata quando il primo job coincide con l'inizio del mio intervallo.

processor demand in [0, L]

$$g(0,L) = \sum_{i=1}^{n} \left\lfloor \frac{L - D_i + T_i}{T_i} \right\rfloor C_i$$

- Stavolta abbiamo la parte intera inferiore (arrotondare per difetto)

Ora voglio testare che per ogni $L \geq 0$ allora $g(L) \leq L$, ma mi da fastidio il fatto di testarlo per ogni L



- Dobbiamo valutare che la funzione a gradino rossa rimanga sempre sotto la retta L, ciò è difficile perchè L è potenzialmente infinito. Il vantaggio è che prima o poi, essendo il task periodico, il gradino si ripeterà uguale. L'intervallo detto **iperperiodo** che corrisponde a $H = lcm_{i=0}^{n}(T_i)$ (minimo comune multiplo di tutti i periodi) è l'intervallo di tempo dopo il quale lo schedule si ripete identicamente. Questo test

Un altro problema è che il tempo è continuo, tale problema si può aggirare controllando se la riga rossa sia sotto la blu solo nei punti in cui il gradino sale, ovvero in corrispondenza delle deadline. Mi basta guardare l'insieme di L dove $\forall i \rightarrow D_i + kT_i$.

In caso di task primi tra loro → $H = \prod_{i=1}^{n} T_i$, con complessità esponenziale $O(x^n)$.

In caso di task armonici tra loro la complessità è $O(n)$.

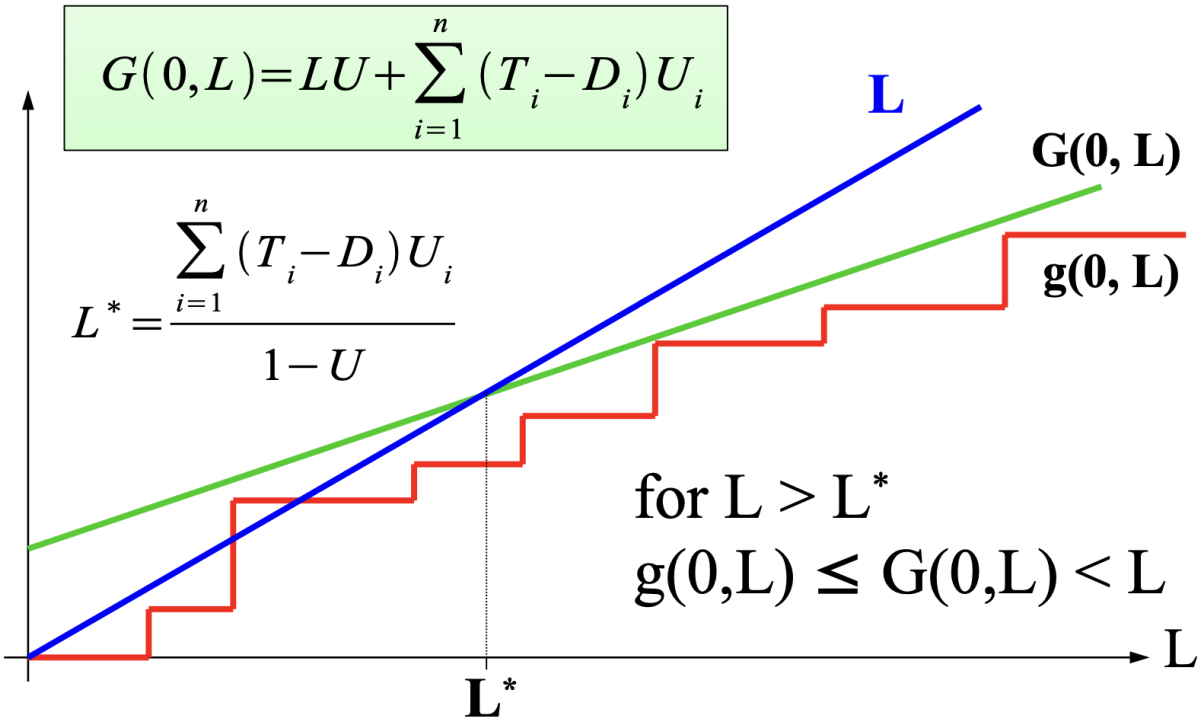
Dobbiamo perciò andare a diminuire la complessita!!! Cerchiamo di finire prima dell'iperperiodo.

$$G(0,L) = \sum_{i=1}^{n} \left(\frac{L + T_i - D_i}{T_i} \right) C_i$$

$$= \sum_{i=1}^{n} L \frac{C_i}{T_i} + \sum_{i=1}^{n} (T_i - D_i) \frac{C_i}{T_i}$$

$$= LU + \sum_{i=1}^{n} (T_i - D_i) U_i$$

- Rimpiazzo la parte intera inferiore con una parentesi semplice. In questo modo ho che $g(L) \leq G(L)$, in quanto prima andavo ad arrotondare per difetto mentre ora no, ragion per cui $G(L)$ sarà più grande.
- Scompongo la sommatoria in due parti.
- A sinistra raccolgo L e lo porto fuori, mi accorgo però che $\sum_{i=1}^{n} \frac{C_i}{T_i} = U$ e vado a sostituirlo.
- A desta faccio la stessa sostituzione con U_i



- Devo dimostrare che la rossa sia sotto la blu, per farlo mi servo del fatto che la linea verde non toccherà mai la rossa.
- La verde ha pendenza minore a quella blu uguale a U, mentre la blu ha pendenza 1.
- Ogni volta che $U < 1$ posso dire che la verde e la blu si toccheranno.
- Dopo il punto di collisione L^*sono sicuro che la verde è sotto la blu, ovvero che $G(L) < L$, e di conseguenza $g(L) \leq G(L) < L$, allora dal punto L^* sono sicuro che $g(L) < L$. In questo modo non ho bisogno di arrivare fino all'iperperiodo, ma basta arrivare a L^*. Per trovarlo basta eguagliare $G(L) = L$ ed ottengo la formula in figura.

$$\forall\, L \in D, \quad g(0, L) \leqslant L$$

$$D = \{ d_k \mid d_k \leq \min(H, L^*) \}$$

$$\begin{cases} H = lcm(T_1, \ldots, T_n) \\ L^* = \dfrac{\sum_{i=1}^{n} (T_i - D_i) U_i}{1 - U} \end{cases}$$

Condizione definitiva

Esercizio: Questo task-set è schedulabile con RM?

	C_i	D_i	T_i
τ_1	3	6	6
τ_2	7	28	28
τ_3	7	28	30

Calcolo H e L^*:

$H = lcm(6, 28, 30) = 3 \cdot 2, 7 \cdot 2^2, 5 \cdot 3 \cdot 2 = 7 \cdot 5 \cdot 2^2 \cdot 3 = 420$

$U = \frac{3}{6} + \frac{7}{28} + \frac{7}{30} = \frac{59}{60}$

$$L^* = \frac{(T_1 - D_1)\frac{C_1}{T_1} + (T_2 - D_2)\frac{C_2}{T_2} + (T_3 - D_3)\frac{C_3}{T_3}}{1 - U} =$$
$$= \frac{(6-6)\frac{3}{6} + (28-28)\frac{7}{28} + (30-28)\frac{7}{30}}{1 - \frac{59}{60}} = \frac{2\frac{7}{30}}{\frac{1}{60}} = 2\frac{7}{30}60 = 28$$

Deadlines → $(6, 12, 18, 24, 28)$ ma mi focalizzo su 28.

$$G(L) = \sum_{i=1}^{n} \left\lceil \frac{L + T_i - D_i}{T_i} \right\rceil C_i$$
$$= \left\lceil \frac{28 + 6 - 6}{6} \right\rceil 3 + \left\lceil \frac{28 + 28 - 28}{28} \right\rceil 7 + \left\lceil \frac{28 + 30 - 28}{30} \right\rceil 7$$
$$= 12 + 7 + 7 = 26 \leq 28 \rightarrow schedulabile$$

Riassunto

3 meccanismi di scheduling:

- **off-line** (molto predicibile ma poco flessibile)
- **fixed-priority** → RM,DM
- **dynamic priority** → EDF, mantiene la periodicità anche nei task periodici

3 tecniche di analisi:

- $U \leq U_{lub}$ → solo sufficente, complessità $O(n)$
- Response Time Analysis, per RM/DM, complessità pseudo-polinomiale $O(n^* D)$→ $\forall i, R_i \leq D_i$
- Processor Demand Criterion, per EDF, complessità pseudo-polinomiale se $U < 1$ → $\forall L, g(0, L) \leq L$