# طراحی سیستم‌های تعبیه‌شده
# Embedded System Design

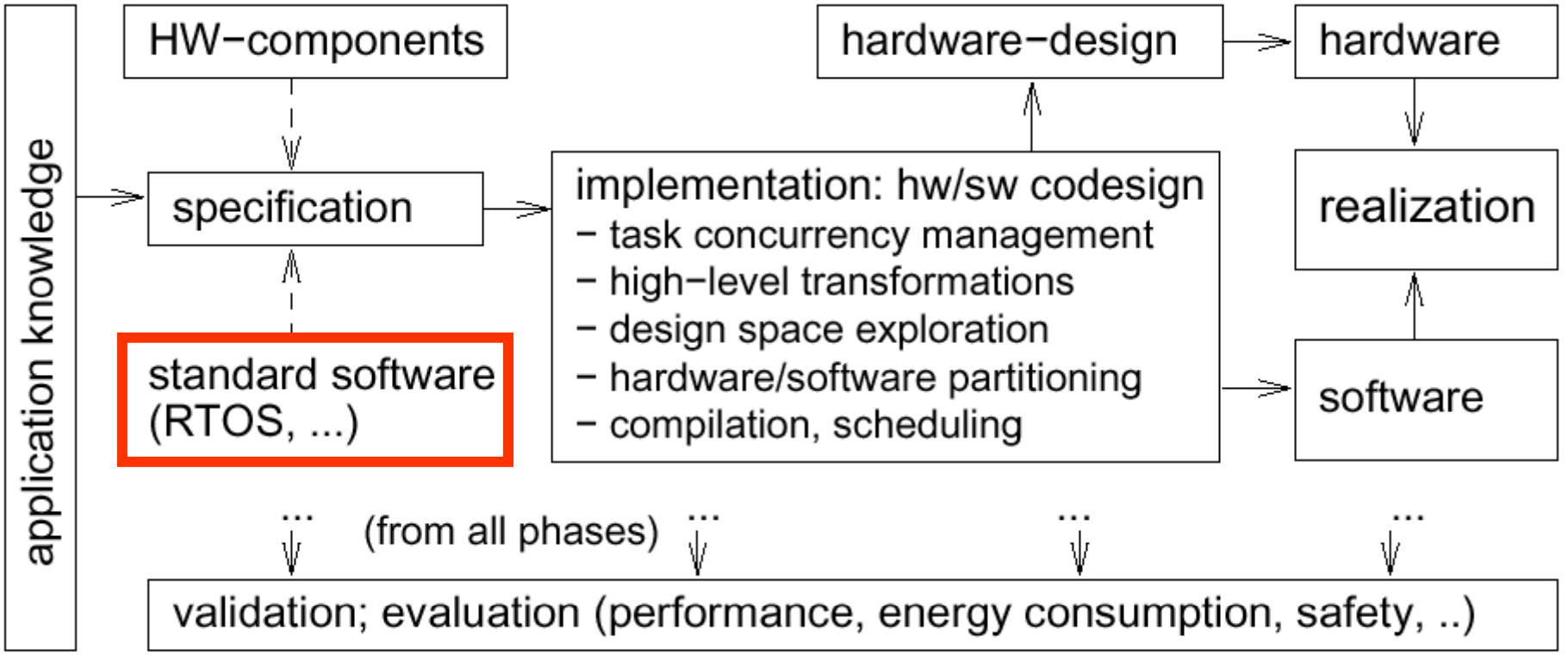
## فصل چهارم ـ قسمت اول

# نرم‌افزار سیستم تعبیه‌شده
# Embedded System Software

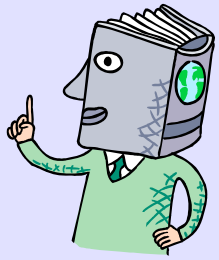
کاظم فولادی
دانشکده‌ی مهندسی برق و کامپیوتر
دانشگاه تهران

kazim@fouladi.ir

# Simplified design flow for embedded systems



# استفاده‌ی مجدد از مؤلفه‌های نرم‌افزاری استاندارد
# Reuse of standard software components



دانش به دست آمده از طراحی‌های قبلی به عنوان یک
**سرمایه‌ی فکری (intellectual property, IP)**
در دسترس قرار می‌گیرد (برای SW و HW):

- Operating systems
- Middleware
- Real-time data bases
- Standard software (MPEG-x, GSM-kernel, …)

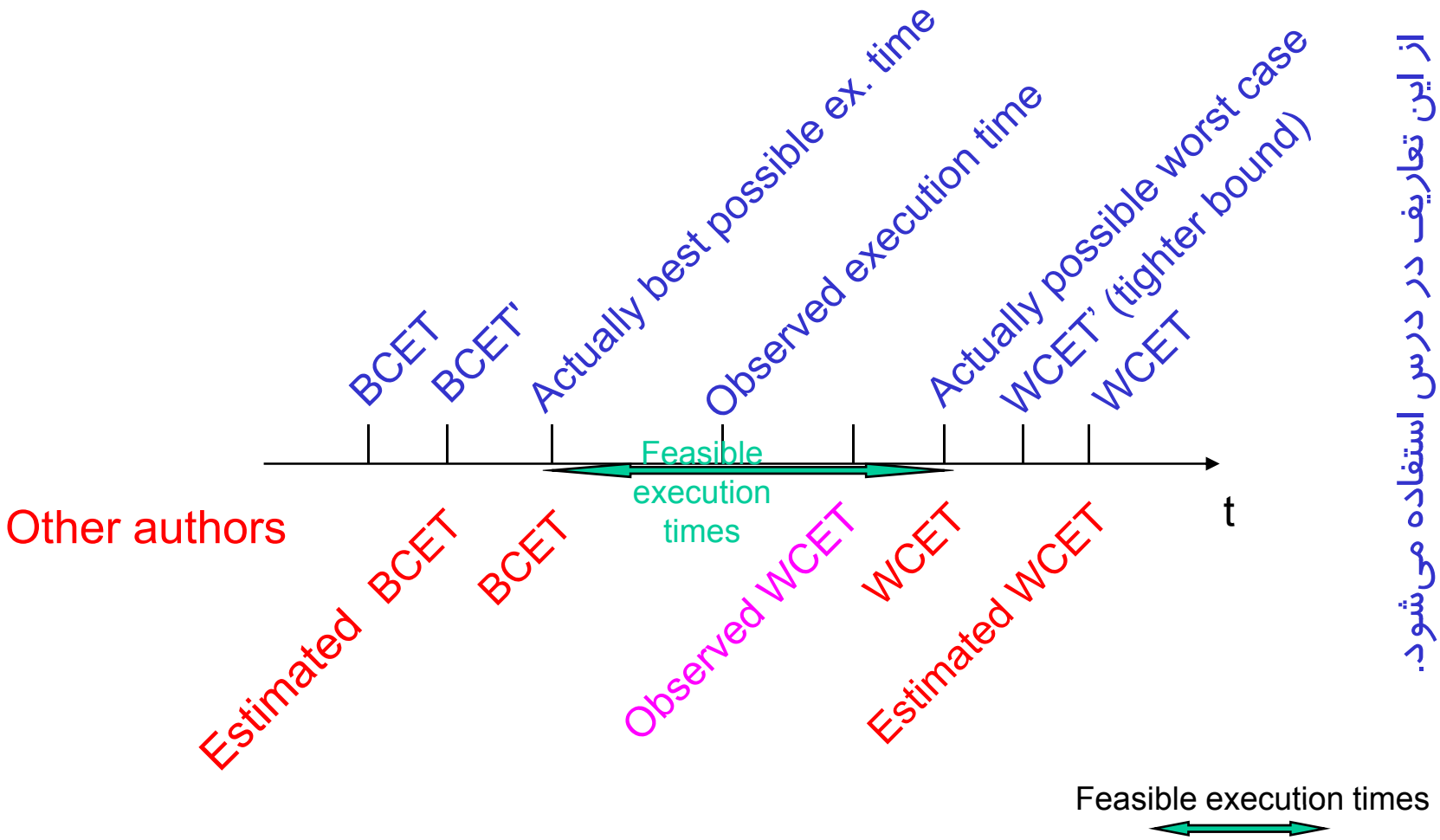
شامل رهیافت‌های استاندارد برای زمان‌بندی
(که نیاز به اطلاعاتی در مورد زمان‌های اجرا دارد).

# زمان‌های اجرای بدترین/ بهترین حالت
# Worst/best case execution times (1)

**تعریف: زمان اجرای بدترین حالت (WCET):**
یک **کران بالا** بر روی زمان‌های اجرای وظایف.
این اصطلاح ایده‌آل نیست، زیرا برنامه‌ای که برای اجرای خود به WCET نیاز داشته باشد، لزوماً نبایستی وجود داشته باشد (WCET یک کران است)

**تعریف: زمان اجرای بهترین حالت (BCET):**
یک **کران پایین** بر روی زمان‌های اجرای وظایف.
این اصطلاح ایده‌آل نیست، زیرا برنامه‌ای که برای اجرای خود به BCET نیاز داشته باشد، لزوماً نبایستی وجود داشته باشد (BCET یک کران است)

# زمان‌های اجرای بدترین / بهترین حالت
# Worst/best case execution times (2)



# زمان‌های اجرای بدترین حالت
# Worst case execution times (2)

**پیچیدگی:**

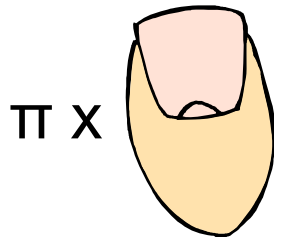
- در حالت کلی: تصمیم‌ناپذیر است (اگر یک کران وجود داشته باشد)
- برای برنامه‌های محدودشده: معماری‌های ساده یا «قدیمی»، بسیار پیچیده برای معماری‌های جدید با خط لوله، حافظه‌ی نهان، حافظه‌ی مجازی و غیره، . . .

**رهیافت‌ها:**

- برای **سخت‌افزار**: نیاز به جزئیات رفتار زمانی دارد
- برای **نرم‌افزار**: نیاز به وجود برنامه‌های زبان ماشین دارد؛ تحلیل‌های پیچیده (برای مثال www.absint.de را ببینید).

# زمان‌های اجرای متوسط
# Average execution times

- **مقادیر هزینه و کارایی براوردشده:**
  دشواری تولید براوردهای به اندازه‌ی کافی دقیق؛
  تعادل بین زمان اجرا و دقت




- **مقادیر هزینه و کارایی دقیق:**
  می‌تواند با ابزارهای عادی (مانند کامپایلرها) انجام شود.
  به اندازه‌ی دقت داده‌های ورودی.

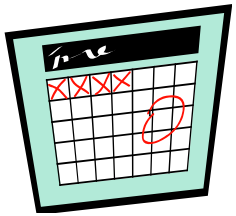
# زمان‌بندی بی‌درنگ
# Real-time scheduling (1)

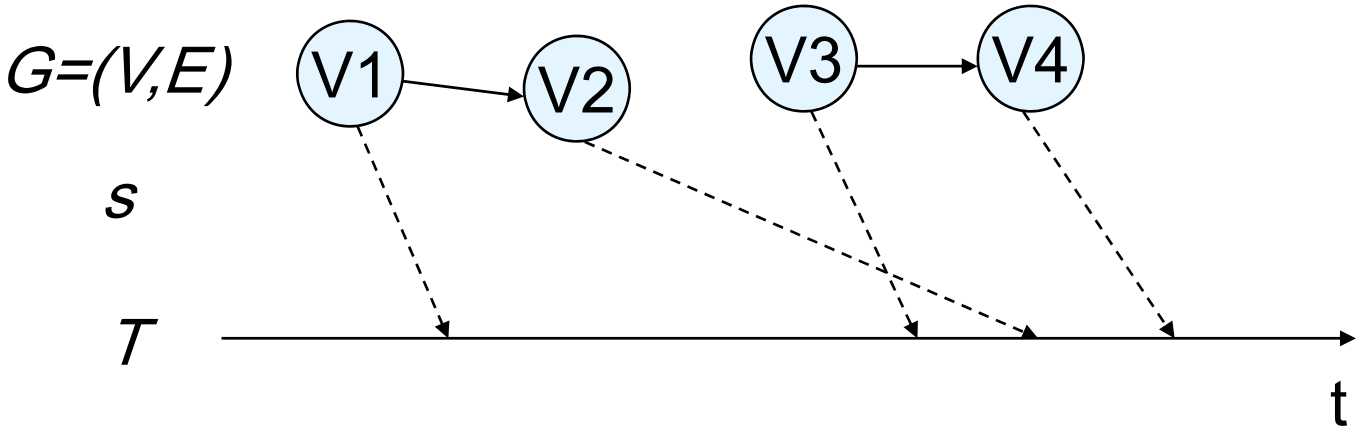
فرض می‌کنیم یک گراف وظیفه $G=(V,E)$ داده شده باشد.

یک **زمان‌بندی** s از G یک نگاشت به صورت

$$V \rightarrow T$$

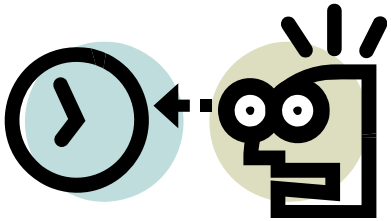
از مجموعه‌ی وظایف $V$ به زمان‌های شروع از دامنه‌ی $T$ است.





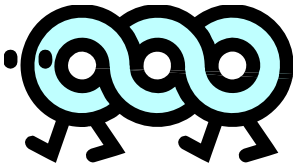
# زمان‌بندی بی‌درنگ
# Real-time scheduling (2)

معمولاً، زمان‌بندی‌ها باید تعدادی محدودیت را در نظر بگیرند، شامل: محدودیت‌های منابع، محدودیت‌های وابستگی، مهلت‌های زمانی.

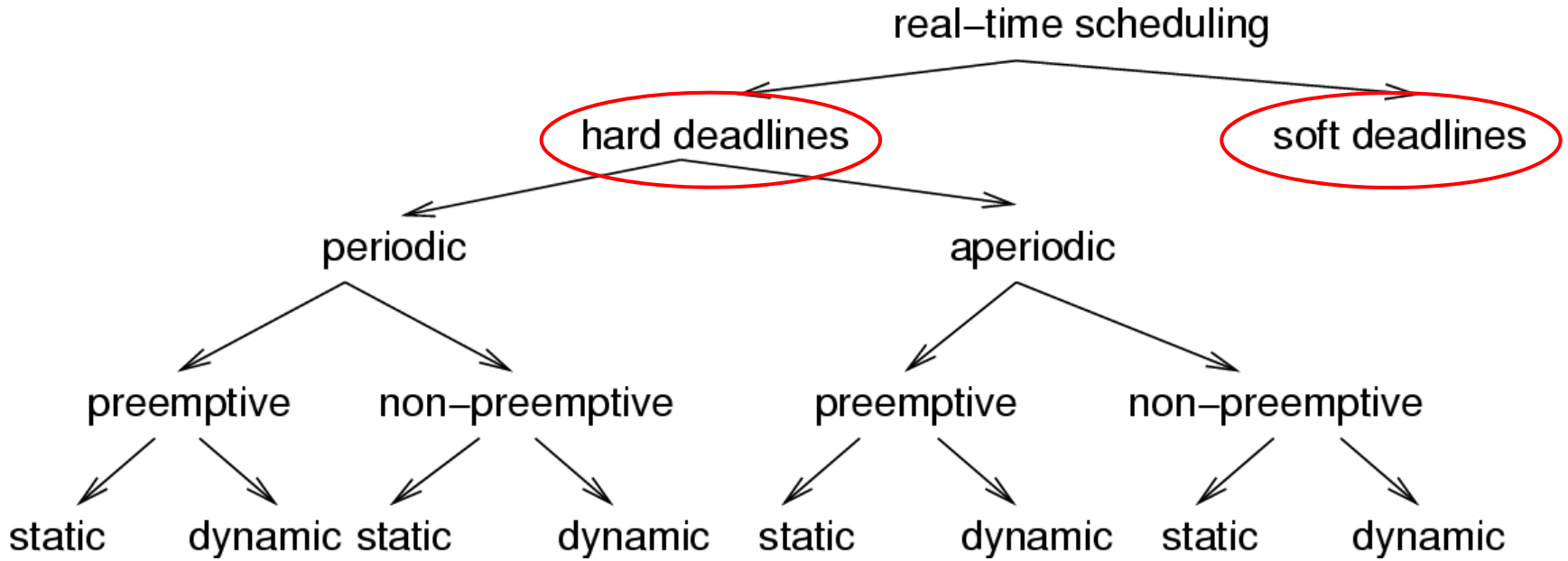


**زمان‌بندی** = یافتن یک چنین نگاشتی است.

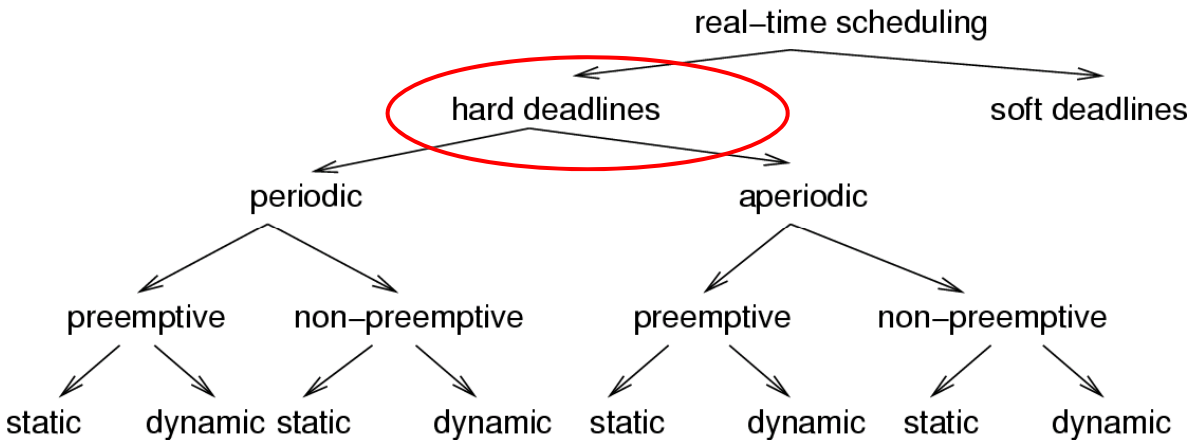
در خلال طراحی سیستم تعبیه‌شده، زمان‌بندی باید به دفعات متعدد انجام شود (زمان‌بندی ابتدایی نادقیق تا زمان‌بندی انتهایی دقیق)

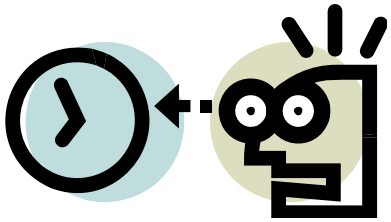


# طبقه‌بندی الگوریتم‌های زمان‌بندی
# Classification of scheduling algorithms



# مهلت‌های سخت و نرم
# Hard and soft deadlines



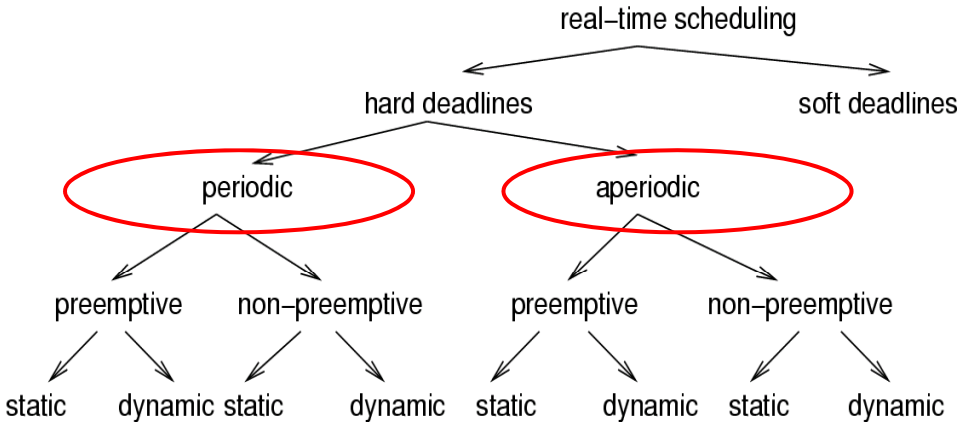



**تعریف:** یک محدودیت زمانی (مهلت deadline) **سخت** نام دارد اگر عدم رعایت آن محدودیت منجر به یک فاجعه شود [Kopetz, 1997].

سایر محدودیت‌های زمانی، **نرم** خوانده می‌شوند.

ما بر روی مهلت‌های سخت تمرکز می‌کنیم.

# وظایف متناوب و نامتناوب
# Periodic and aperiodic tasks

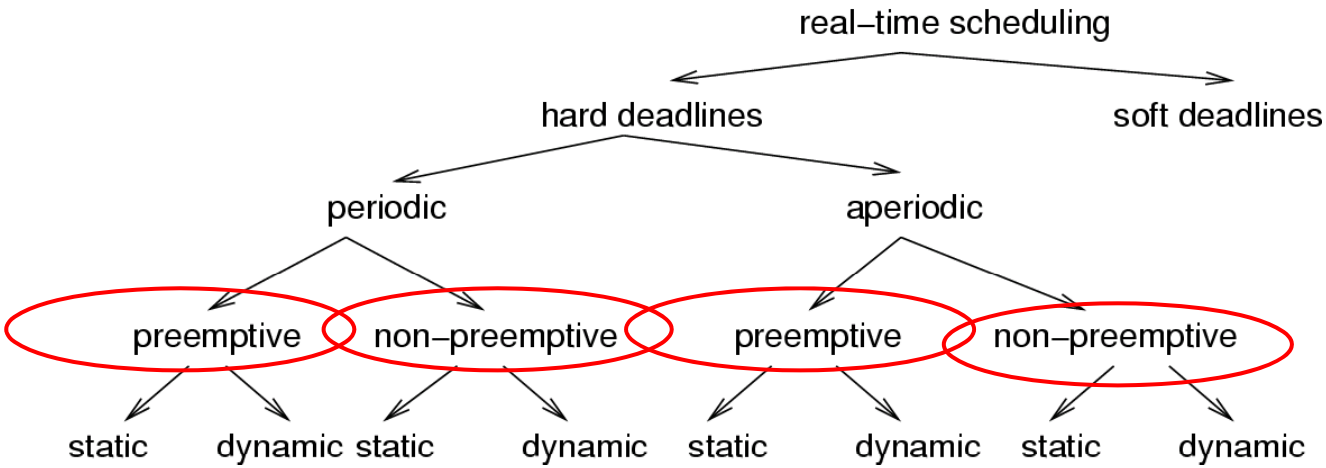



**تعریف:** وظایفی که باید هر p واحد زمانی یک بار اجرا شوند، وظایف **متناوب** نام دارند.
P دوره‌ی تناوب آنها خوانده می‌شود.
هر اجرای یک وظیفه‌ی متناوب، یک **کار (job)** نامیده می‌شود.

سایر وظایف **نامتناوب** نامیده می‌شوند.

**تعریف:** وظایفی که پردازنده را در زمان‌های غیرقابل پیش‌بینی درخواست می‌کنند، **گاه و بیگاه (sporadic)** نامیده می‌شوند، اگر یک تفکیک می‌نیمم بین زمان‌های درخواست پردازنده توسط آنها وجود داشته باشد.

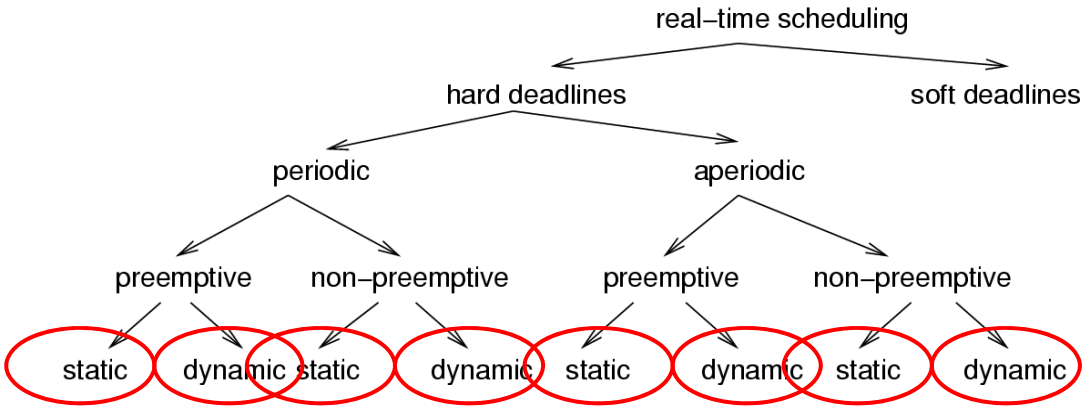
# زمان‌بندی پس‌دادنی و پس‌ندادنی
# Preemptive and non-preemptive scheduling




- **زمان‌بندهای پس‌ندادنی:**
  - وظایف اجرا می‌شوند تا وقتی که اجرای آنها کامل شود.
  - زمان پاسخ برای رویدادهای خارجی می‌تواند بسیار طولانی باشد.
- **زمان‌بندهای پس‌دادنی:** در صورتی استفاده می‌شود که
  - برخی وظایف دارای زمان‌های اجرای طولانی باشند، یا
  - زمان پاسخ برای رویدادهای خارجی بایستی کوتاه باشد.

# زمان‌بندی پویا / برخط
# Dynamic/online scheduling

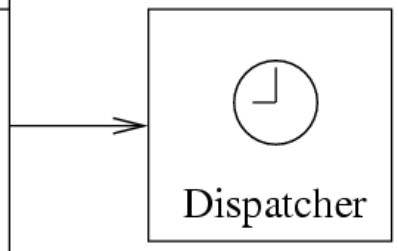
- **زمان‌بندی پویا / برخط:**
  تصمیم‌گیری برای تخصیص پردازنده (زمان‌بندی)
  در زمان اجرا انجام می‌شود؛
  مبتنی بر اطلاعاتی راجع به وظایف وارد شده تاکنون

# زمان‌بندی ایستا / برون‌خط
# Static/offline scheduling

- **زمان‌بندی ایستا / برون‌خط:**
  زمان‌بندی‌ای که اطلاعات پیشین در مورد زمان‌های ورود، زمان‌های اجرا و مهلت‌های زمانی را در نظر می‌گیرد.
  اعزام‌کننده پردازنده را تخصیص می‌دهد، هنگامی که به وسیله‌ی زمان‌سنج بدان وقفه داده شود. زمان‌سنج به وسیله‌ی یک جدول تولید شده در زمان طراحی کنترل می‌شود.

| Time | Action | WCET |
|---|---|---|
| 10 | start T1 | 12 |
| 17 | send M5 | |
| 22 | stop T1 | |
| 38 | start T2 | 20 |
| 47 | send M3 | |

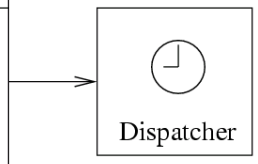



# سیستم‌های حساس به زمان
# Time-triggered systems (1)

در یک سیستم کاملاً حساس به زمان، ساختار کنترل زمانی همه‌ی وظایف به وسیله‌ی ابزارهای پشتیبانی برون‌خط **از پیش** بنا می‌شود. این ساختار کنترل زمانی در **فهرست توصیف‌گر وظیفه (TDL)** کد می‌شود که حاوی زمان‌بندی چرخه‌ای برای همه‌ی فعالیت‌های آن گره است. این زمان‌بندی اولویت لازم را در نظر می‌گیرد و روابط انحصار متقابل بین وظایف مانند هماهنگی صریح وظایف به وسیله‌ی سیستم عامل در زمان اجرا لازم نیست ...

اعزام‌کننده به وسیله‌ی یک تیک ساعت همگام‌شده فعال می‌شود. این در TDL دیده می‌شود و سپس عملی که برای این لحظه برنامه‌ریزی شده است، را اجرا می‌کند [Kopetz].

| Time | Action | WCET |
|---|---|---|
| 10 | start T1 | 12 |
| 17 | send M5 | |
| 22 | stop T1 | |
| 38 | start T2 | 20 |
| 47 | send M3 | |



# سیستم‌های حساس به زمان
# Time-triggered systems (2)

***... زمان‌بندی پیش از زمان اجرا معمولاً تنها وسیله‌ی عملی است که در یک سیستم پیچیده قابلیت پیش‌بینی را فراهم می‌کند.*** [Xu,Parnas].

به راحتی می‌توان بررسی کرد که آیا محدودیت‌های زمانی رعایت می‌شود یا خیر. عیب آن این است که پاسخ به رویدادهای گاه و بیگاه می‌تواند ضعیف باشد.

# زمان‌بندی متمرکز و توزیع‌شده
# Centralized and distributed scheduling

- **زمان‌بندی متمرکز و توزیع‌شده:**
  زمان‌بندی چندپردازنده‌ای به طور محلی بر روی یک یا چند پردازنده.
- **زمان‌بندی تک ـ و چند پردازنده‌ای**
  - الگوریتم‌های زمان‌بندی ساده تک پردازنده‌ها را اداره می‌کنند،
  - الگوریتم‌های پیچیده‌تر پردازنده‌های چندگانه را اداره می‌کنند.
    - الگوریتم‌هایی برای سیستم‌های چندپردازنده‌ی همگن homogeneous
    - الگوریتم‌هایی برای سیستم‌های چندپردازنده‌ی ناهمگن heterogeneous

(شامل شتاب‌دهنده‌های سخت‌افزاری به عنوان یک حالت خاص).

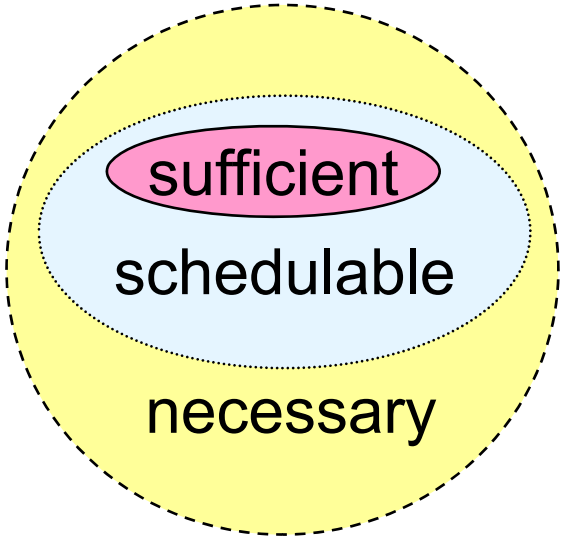
# قابلیت زمان‌بندی
# Schedulability

مجموعه‌ای از وظایف تحت مجموعه‌ای از محدودیت‌ها **قابل زمان‌بندی** است، اگر یک زمان‌بندی برای آن مجموعه از وظایف و محدودیت‌ها وجود داشته باشد.

**آزمون‌های دقیق**، در بسیاری از موارد NP-hard هستند.

**آزمون‌های کافی:** شرایط کافی برای زمان‌بندی بررسی می‌شود. (به طور امیدوارانه) احتمال کوچکی وجود دارد که هیچ زمان‌بندی‌ای موجود نباشد.

**آزمون‌های لازم:** بررسی شرایط لازم. برای نشان دادن عدم وجود زمان‌بندی. مواردی می‌تواند وجود داشته باشد که در آن‌ها هیچ زمان‌بندی‌ای وجود ندارد و ما نمی‌توانیم آن را ثابت کنیم.



# توابع هزینه
# Cost functions

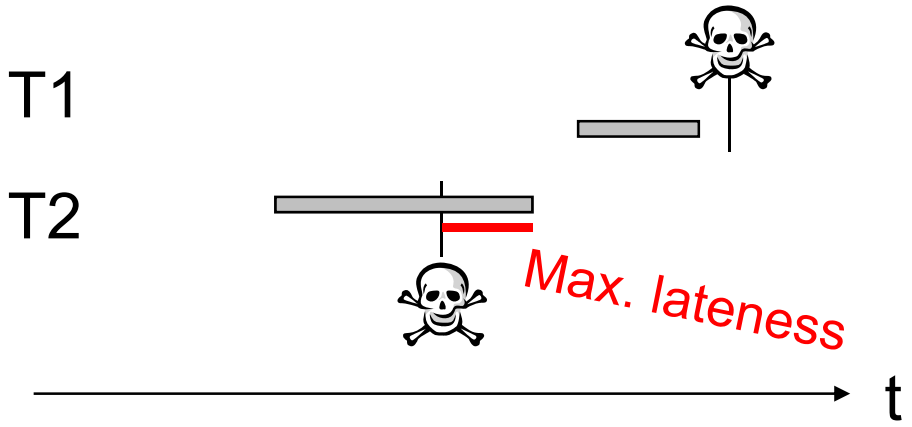
**توابع هزینه:** هدف الگوریتم‌های مختلف، می‌نیمم کردن توابع هدف مختلف است.

تعریف: **ماکزیمم دیرکرد**

**Maximum lateness =**

$\mathbf{max}_{\text{all tasks}}$ (completion time – deadline)

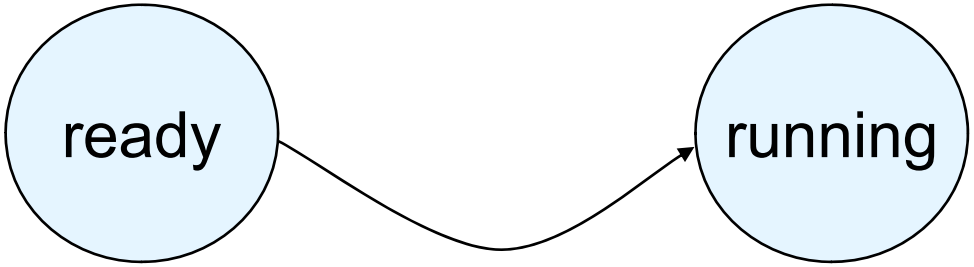
Is <0 if all tasks complete before deadline.



# وظایف ساده
# Simple tasks

وظایف بدون ارتباطات میان‌پردازشی، **وظایف ساده (S-tasks)** نام دارند.

S-taskها می‌توانند در یکی از این دو حالت باشند: آماده، اجرا.

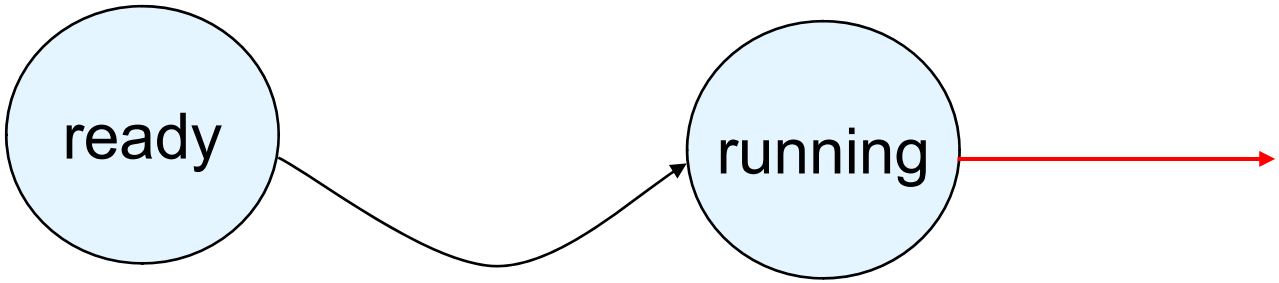


# وظایف ساده
# Simple tasks

The API of a TT-OS supporting S-tasks is .. simple [Kopetz]:

It consists of 3 data structures & 2 OS calls. ... The calls are TERMINATE TASK & ERROR.

- *The TERMINATE TASK system call is executed whenever the task has reached its termination point.*
- *In case of an error that cannot be handled within the application task, the task terminates its operation with the ERROR system call.*

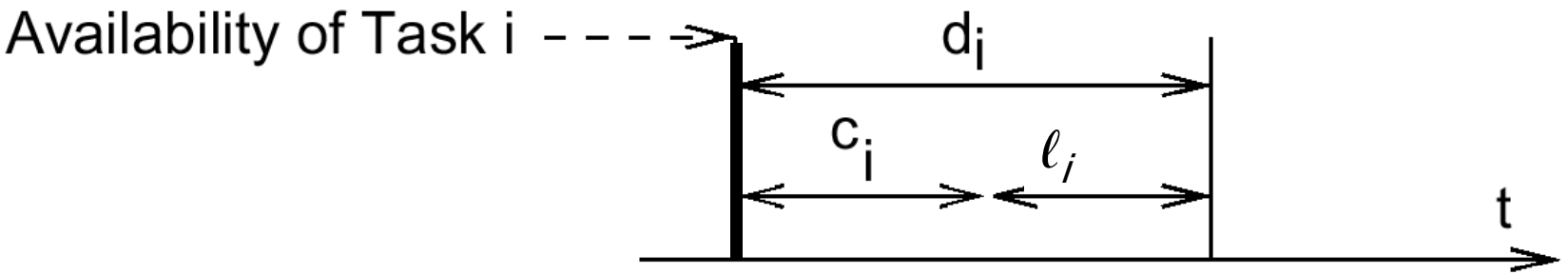



# زمان‌بندی نامتناوب ـ زمان‌بندی بدون محدودیت‌های تقدمی

## Aperiodic scheduling - Scheduling with no precedence constraints -

Let $\{T_i\}$ be a set of tasks. Let:

- $c_i$ be the execution time of $T_i$,
- $d_i$ be the **deadline interval**, that is,
  the time between $T_i$ becoming available
  and the time until which $T_i$ has to finish execution.
- $\ell_i$ be the **laxity** or **slack**, defined as $\ell_i = d_i - c_i$
- $f_i$ be the finishing time.

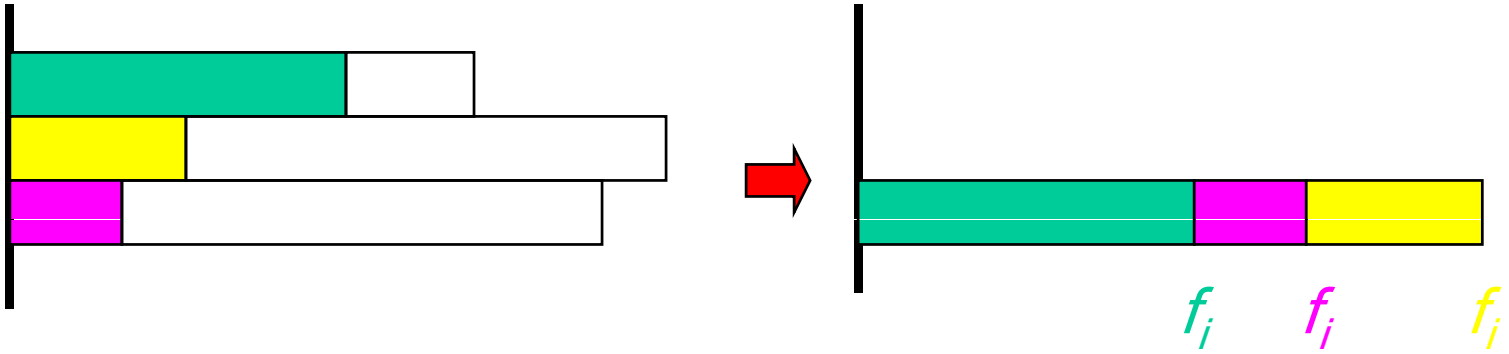


# تک‌پردازنده با زمان‌های ورود مساوی
# Uniprocessor with equal arrival times

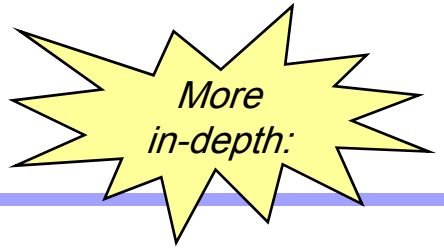
پس‌دادن سودمند نیست.

**Earliest Due Date** (EDD):

**زودترین زمان انجام:** کاری که نخستین مهلت را دارد زودتر شروع کنید.




EDD لازم دارد که همه‌ی وظایف بر طبق مهلت‌های (مطلق) آنها مرتب شوند. بنابراین پیچیدگی آن $O(n \log(n))$ است.




# Optimality of EDD

EDD is optimal, since it follows Jackson's rule:

Given a set of *n* independent tasks, any algorithm that executes the tasks in order of non-decreasing (absolute) deadlines is optimal with respect to minimizing the maximum lateness.

Proof (See Buttazzo, 2002):

Let $\sigma$ be a schedule produced by any algorithm *A*

If $A \neq \text{EDD} \rightarrow \exists\ T_a,\ T_b,\ d_a \leq d_b,\ T_b$ immediately precedes $T_b$ in $\sigma$.

Let $\sigma'$ be the schedule obtained by exchanging $T_a$ *and* $T_b$.

# Exchanging $T_a$ and $T_b$ cannot increase lateness

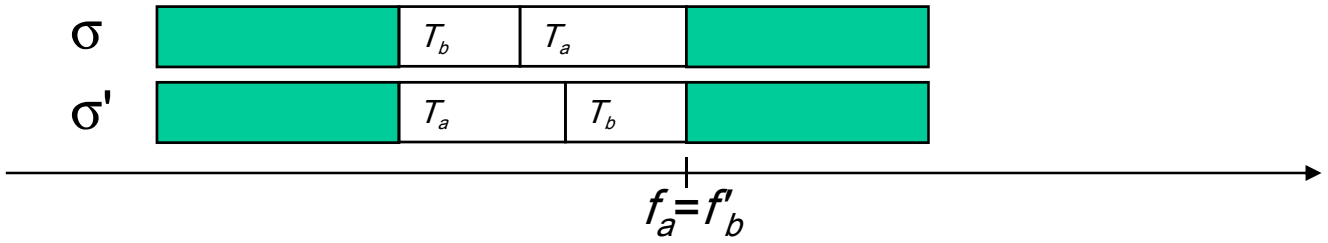
Max. lateness for $T_a$ and $T_b$ in $\sigma$ is $L_{max}(a,b)=f_a-d_a$

Max. lateness for $T_a$ and $T_b$ in $\sigma'$ is $L'_{max}(a,b)=\max(L'_a,L'_b)$

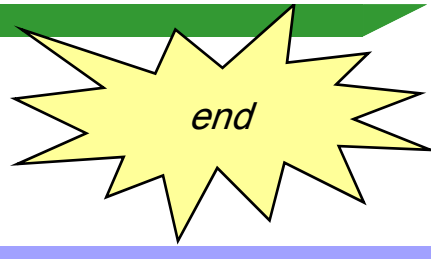
Two possible cases

1. $L'_a \geq L'_b$: $\rightarrow L'_{max}(a,b) = f'_a - d_a < f_a - d_a = L_{max}(a,b)$ since $T_a$ starts earlier in schedule $\sigma'$.
2. $L'_a \leq L'_b$: $\rightarrow L'_{max}(a,b) = f'_b - d_b = f_a - d_b \leq f_a - d_a = L_{max}(a,b)$ since $f_a=f'_b$ and $d_a \leq d_b$

☞ $L'_{max}(a,b) \leq L_{max}(a,b)$



# EDD is optimal




- Any schedule $\sigma$ with lateness $L$ can be transformed into an EDD schedule $\sigma^n$ with lateness $L^n \leq L$, which is the minimum lateness.
- EDD is optimal (q.e.d.)

# زودترین مهلت اول ـ قضیه‌ی هورن ـ
# Earliest Deadline First (EDF) - Horn's Theorem -

زمان‌های ورود مختلف: پس‌دادن به طور بالقوه دیرکرد را کاهش می‌دهد.

**قضیه [Horn 74]:** مجموعه‌ای از n وظیفه‌ی مستقل با زمان‌های ورود دلخواه داده شده است. هر الگوریتمی که در هر مورد، وظیفه‌ای را اجرا کند که زودترین مهلت مطلق بین همه‌ی وظایف آماده را دارد، نسبت به می‌نیمم کردن دیرکرد ماکزیمم بهینه است.

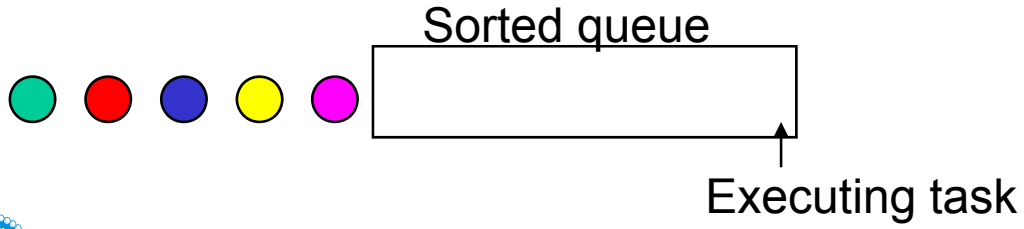
# زودترین مهلت اول ـ الگوریتم ـ
# Earliest Deadline First (EDF) - Algorithm -

**Earliest deadline first** (EDF) algorithm:

هر زمانی که یک وظیفه‌ی آماده‌ی جدید وارد می‌شود:

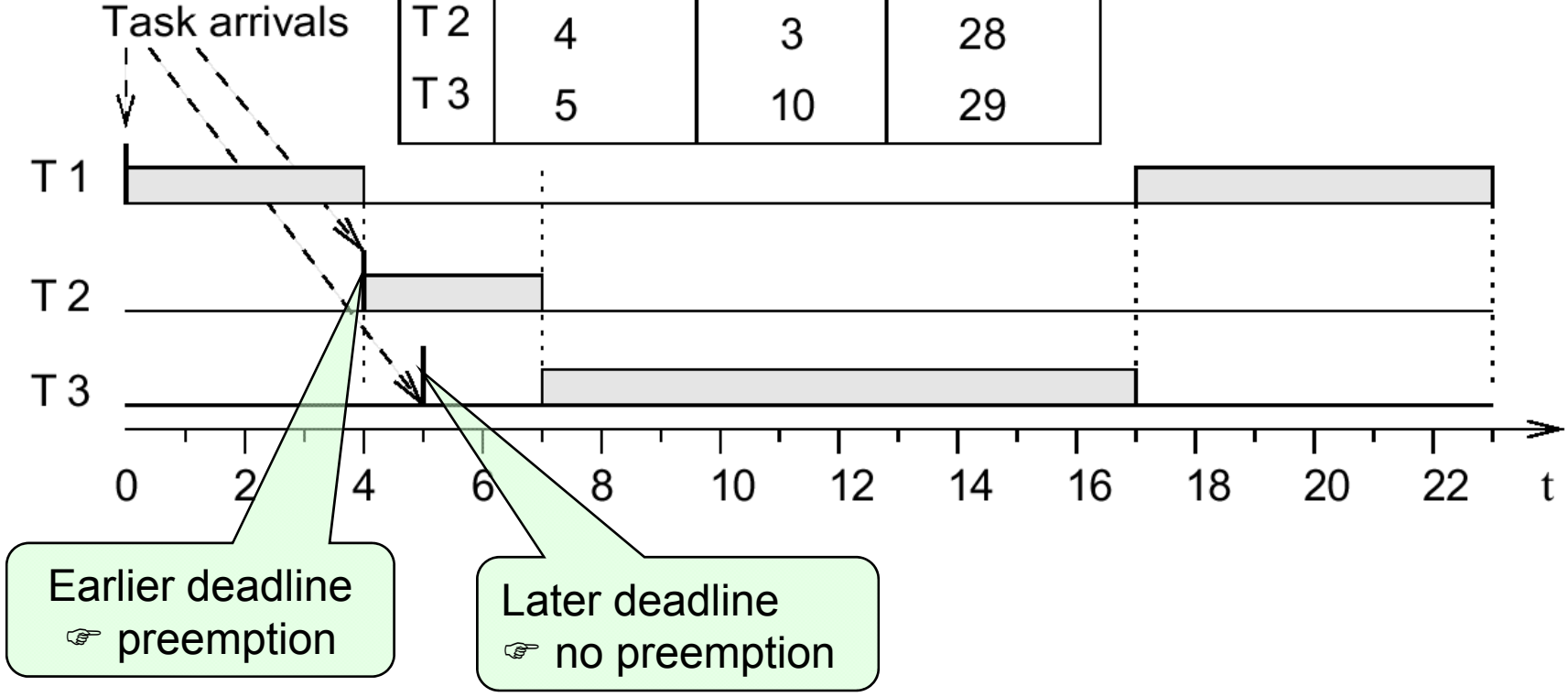
- به صف وظایف آماده اضافه می‌شود که بر اساس مهلت‌های **مطلق** مرتب شده است. وظیفه‌ی موجود در ابتدای صف اجرا می‌شود.
- اگر وظیفه‌ی بتازگی وارد شده در ابتدای صف درج شود، پردازنده از وظیفه‌ی فعلی در حال اجرا پس گرفته می‌شود.

رهیافت سرراست با لیست‌های مرتب شده (مقایسه‌ی کامل با وظایف موجود برای هر وظیفه‌ی وارد شده) نیاز به زمان اجرای $O(n^2)$ دارد. (با جستجوی دودویی یا آرایه‌های باکتی این مقدار کمتر می‌شود).



# Earliest Deadline First (EDF)
# - Example -

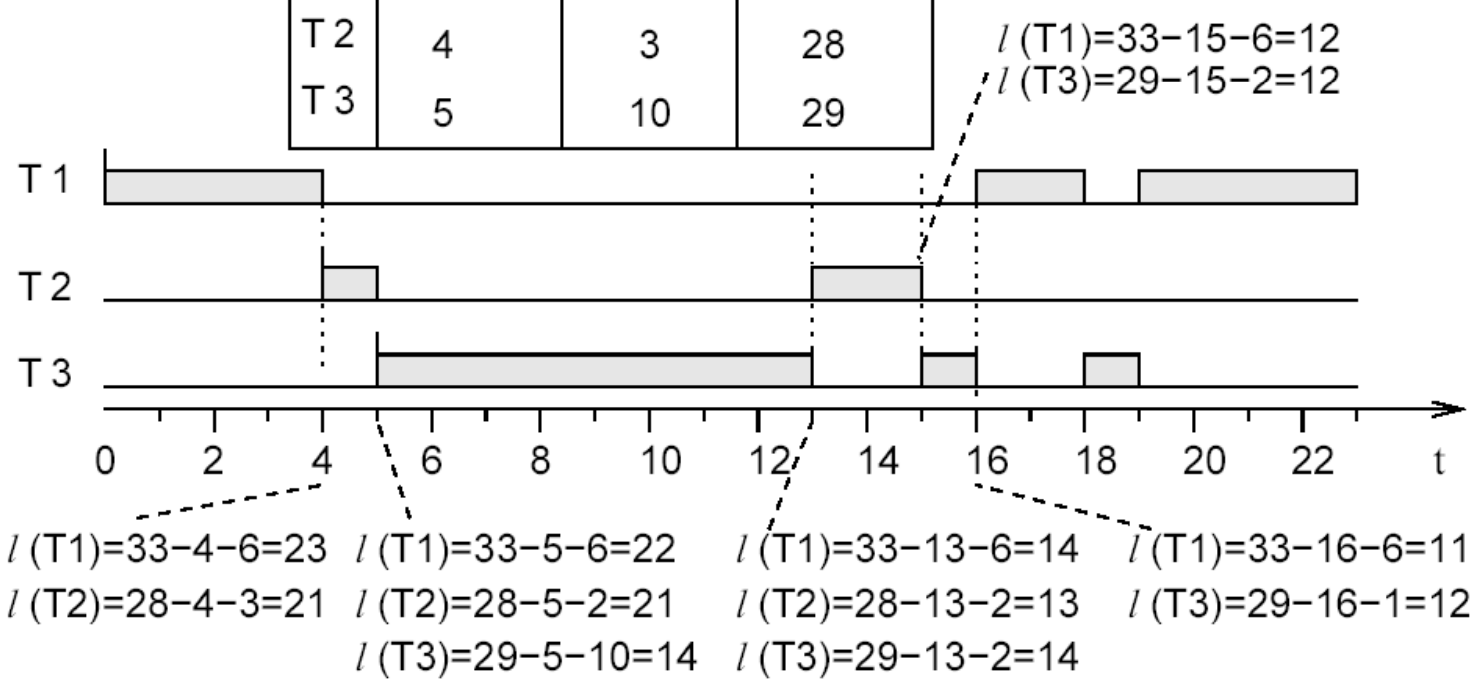
| | arrival | duration | deadline |
|---|---|---|---|
| T 1 | 0 | 10 | 33 |
| T 2 | 4 | 3 | 28 |
| T 3 | 5 | 10 | 29 |



# حداقل سستی، حداقل زمان سستی اول
# Least laxity (LL), Least Slack Time First (LST)

اولویت‌ها = تابع نزولی از سستی (سستی کمتر، اولویت بالاتر)؛
تغییر پویای اولویت‌ها، پس‌دادنی.

| | arrival | duration | deadline |
|---|---|---|---|
| T 1 | 0 | 10 | 33 |
| T 2 | 4 | 3 | 28 |
| T 3 | 5 | 10 | 29 |



# ویژگی‌ها
# Properties

- فراخوانی زمان‌بند و محاسبه‌ی مجدد سستی تنها در زمان‌های ورود وظیفه کافی نیست.
- سربار برای فراخوانی‌های زمان‌بند.
- تعویض متن‌های بسیار
- در ابتدا مهلت‌های از دست رفته را تشخیص می‌دهد.
- LL نیز یک الگوریتم بهینه‌ی زمان‌بندی برای سیستم‌های تک‌پردازنده است.
- اولویت‌های پویا: نمی‌تواند با یک اولویت ثابت استفاده شود.

# زمان‌بندی بدون پس دادن
# Scheduling without preemption

**لم:** اگر پس دادن اجازه داده نشود، زمان‌بندهای بهینه بایستی در زمان‌های مشخص پردازنده را بیکار نگه دارند.

**اثبات:** فرض کنید زمان‌بندهای بهینه هرگز پردازنده را بیکار نگذارند:

# زمان‌بندی بدون پس دادن
# Scheduling without preemption (2)

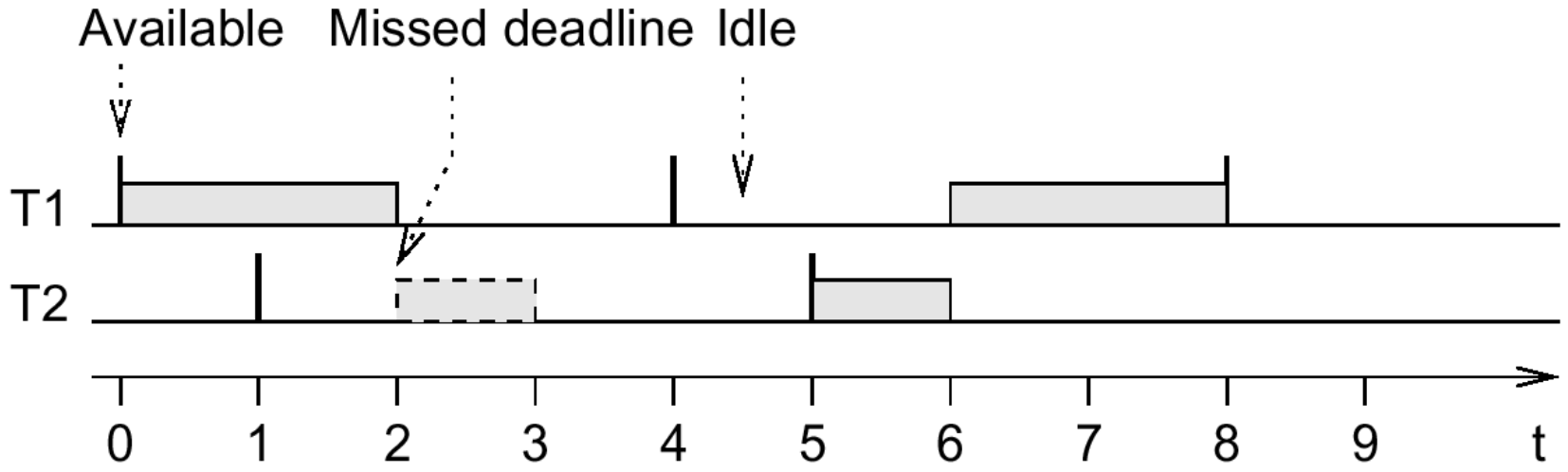
T1: periodic, $c_1 = 2$, $p_1 = 4$, $d_1 = 4$

T2: occasionally available at times $4*n+1$, $c_2 = 1$, $d_2 = 1$

T1 has to start at t=0

☞ deadline missed, but schedule is possible (start T2 first)

☞ scheduler is not optimal ☞ contradiction! q.e.d.



# زمان‌بندی بدون پس دادن
# Scheduling without preemption

پس دادن اجازه داده نمی‌شود: زمان‌بندهای بهینه می‌توانند پردازنده را بیکار نگه دارند تا وظایف با مهلت‌های زودتر که دیرتر وارد می‌شوند، تمام شوند.

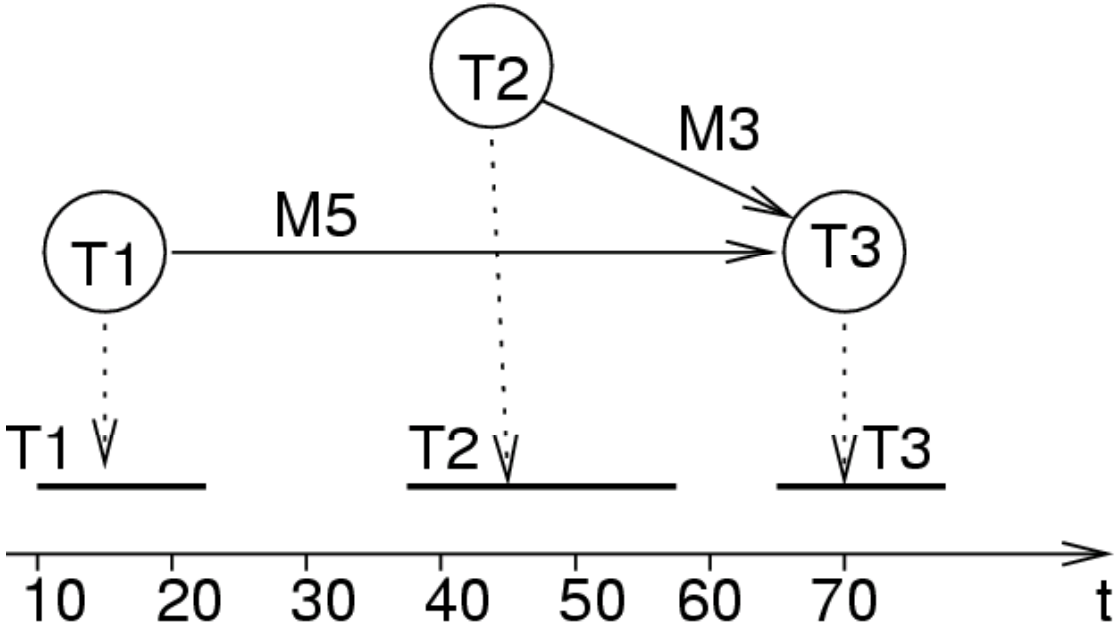
- برای الگوریتم‌های زمان‌بندی بهنیه، اطلاعات در مورد آینده مورد نیاز است.
- هیچ الگوریتم برخطی نمی‌تواند تصمیم بگیرد که آیا پردازنده را بیکار نگه دارد یا خیر.

EDF بین همه‌ی الگوریتم‌های زمان‌بندی که پردازنده را در زمان‌های مشخص بیکار نگه نمی‌دارند، بهینه است.

اگر زمان‌های ورود از قبل معلوم باشند، مساله‌ی زمان‌بندی در حالت کلی NP-hard می‌شود. روش شاخه و حد معمولاً استفاده می‌شود.

# زمان‌بندی با محدودیت‌های تقدم
# Scheduling with precedence constraints

گراف وظیفه و زمان‌بندی ممکن:



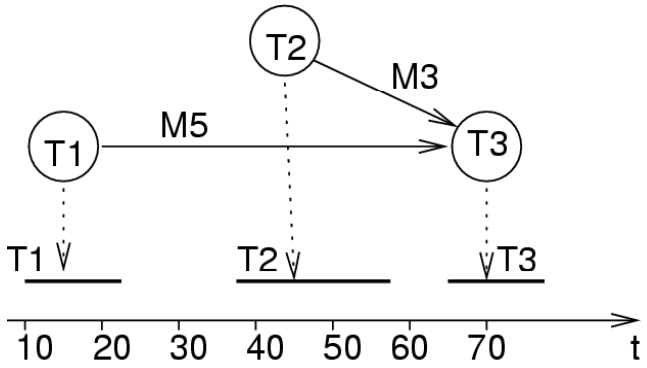

زمان‌بندی می‌تواند در یک جدول ذخیره شود.

# زمان‌های ورود همزمان: الگوریتم دیرترین مهلت اول
## Simultaneous Arrival Times: The Latest Deadline First (LDF) Algorithm

LDF : [Lawler, 1973] گراف وظیف را می‌خواند و بین وظایفی که هیچ مابعدی وجود ندارد، وظیفه‌ای که دیرترین مهلت را دارد در صف درج می‌کند. سپس این فرایند را تکرار می‌کند، وظایفی که مابعد آنها همگی انتخاب شده اند، در صف قرار می‌گیرد.

در زمان اجرا ، وظایف در ترتیب کلی تولید شده اجرا می‌شوند.

LDF پس‌ندادنی است و برای تک‌پردازنده‌ای بهینه می‌باشد.




اگر هیچ مهلت محلی موجود نباشد، LDF تنها یک مرتب‌سازی توپولوژیکی انجام می‌دهد.

# زمان‌های ورود ناهمگام: الگوریتم EDF تغییریافته
## Asynchronous Arrival Times: Modified EDF Algorithm

این مورد می‌تواند با یک الگورتیم EDF تغییریافته اداره شود.
ایده‌ی کلیدی تبدیل مساله از یک مجموعه وظایف داده‌شده‌ی مستقل به مجموعه‌ای از وظایف مستقل با پارامترهای زمانی متفاوت است[Chetto90].
این الگوریتم برای سیستم‌های تک پردازنده بهینه است.
اگر پس دادن اجازه داده نشود، می‌توان از الگوریتم هیوریستیک توسعه داده شده توسط Stankovic و Ramamritham استفاده کرد.

# خلاصه

Worst case execution times (WCET)

Definition of scheduling terms

- Hard vs. soft deadlines
- Static vs. dynamic ☞TT-OS
- Schedulability

Scheduling approaches

- Aperiodic tasks
  - No precedences
    - Simultaneous (☞EDD) & Asynchronous Arrival Times (☞EDF, LL)
  - Precedences
    - Simultaneous (☞ LDF) & Asynchronous Arrival Times (☞ mEDF)