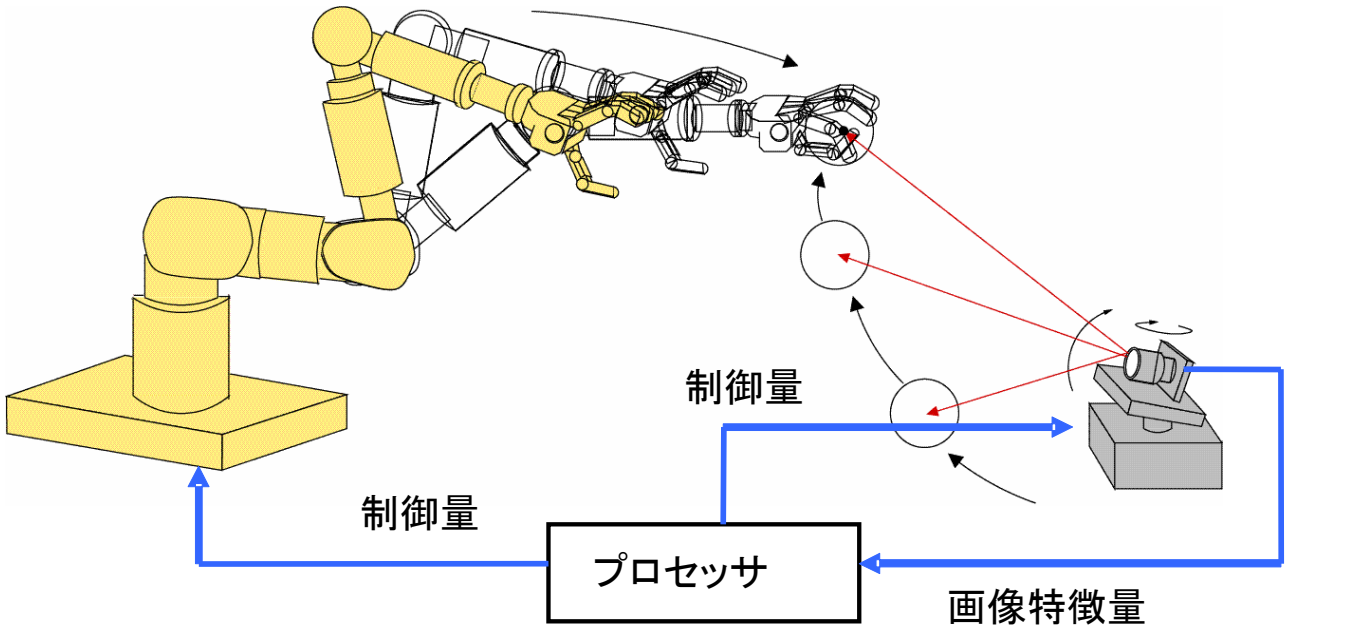
知能制御システム学

リアルタイムシステム

東北大学 大学院情報科学研究科
鏡 慎吾
swk(at)ic.is.tohoku.ac.jp

2007.07.10

ビジュアルフィードバック制御



制御量の計算　　画像処理

→ リアルタイム処理

リアルタイムとは?

広義には,

ある処理の正当性が，その論理的な計算結果だけではなく，計算のなされた時間にも依存すること

狭義には,

ある処理の完了に対してデッドラインが定められていること

さまざまなリアルタイム性の例

軌道計画
~ 1[s]

ダイナミクスモデル計算
~100 [ms]

モータ制御 ~1[ms]

レーザ飛行時間距離計測
~100[ps]

音声波形生成
~20 [μs]

力覚センサ処理
~1 [ms]

画像処理
~33 [ms]

音声信号処理
~100 [ms]

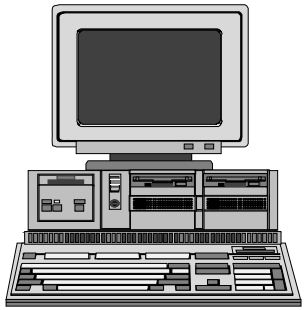
リアルタイム ≠ 高速

何を以ってリアルタイムとするかはそのシステムがインタラクトする物理現象で決まる

リアルタイム計算システム

レーザ飛行時間距離計測
~100[ps]

軌道計画
~ 1[s]



音声波形生成
~20 [μs]

ダイナミクスモデル計算
~100 [ms]

力覚センサ処理
~1 [ms]

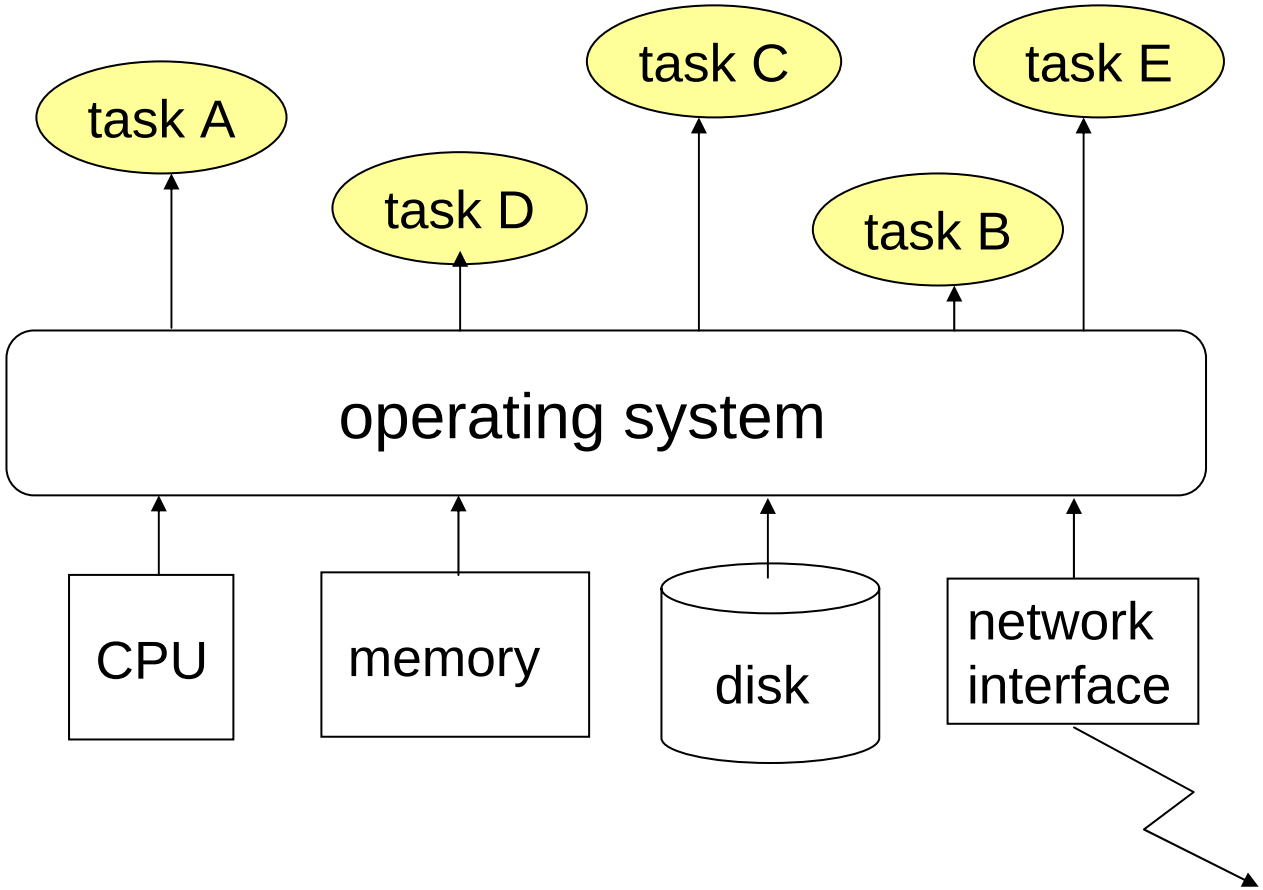
モータ制御 ~1[ms]

画像処理
~33 [ms]

音声信号処理
~100 [ms]

いろいろな処理を，それぞれの時間制約を満たしながら実行したい

マルチタスクOS



複数のタスクにハードウェア資源を（多くの場合時分割で）割り当てる

並列処理と並行処理

並列処理 (Parallel Processing)
　実際に複数の処理が同時に実行される

並行処理 (Concurrent Processing)
　複数の処理の流れが存在する (並列かどうかは話が別)

タスク? プロセス? スレッド?

厳密な定義があるわけではないようだが，おおむねこんな感じか．（人によって使い方が違うので空気を読む必要がある）

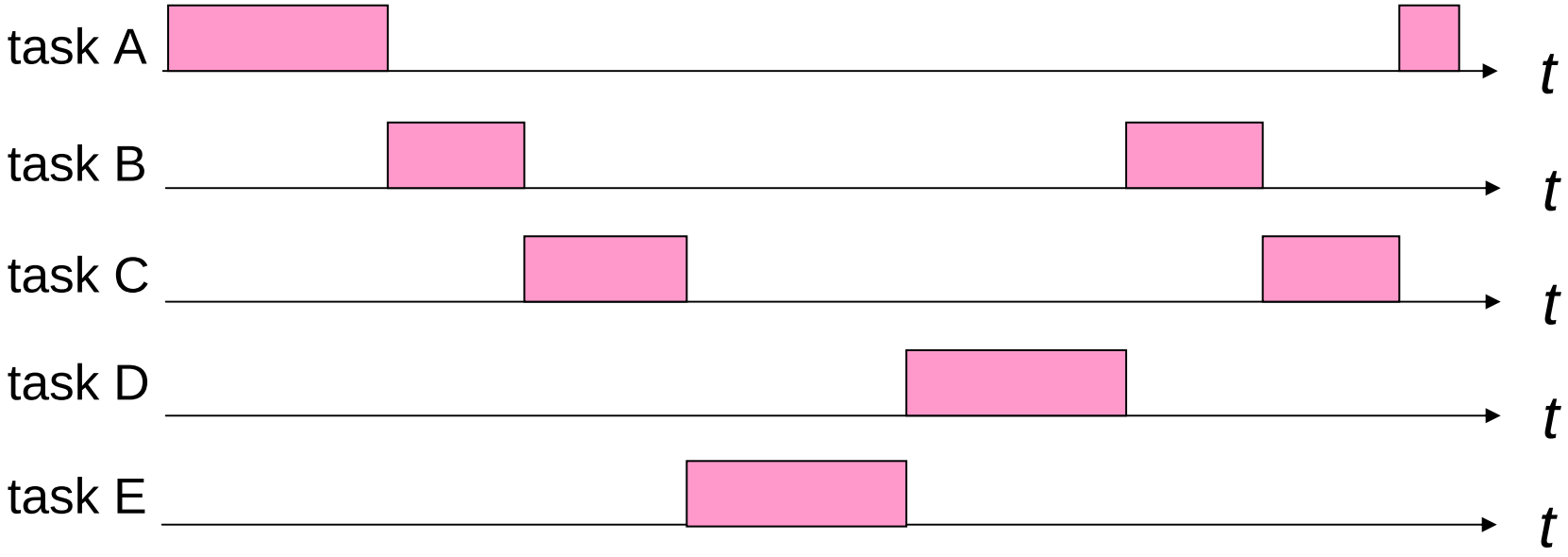
プロセス
　メモリ空間が独立している

スレッド
　メモリ空間を互いに共有している

タスク
　上記の区別を気にしない場合の呼び方

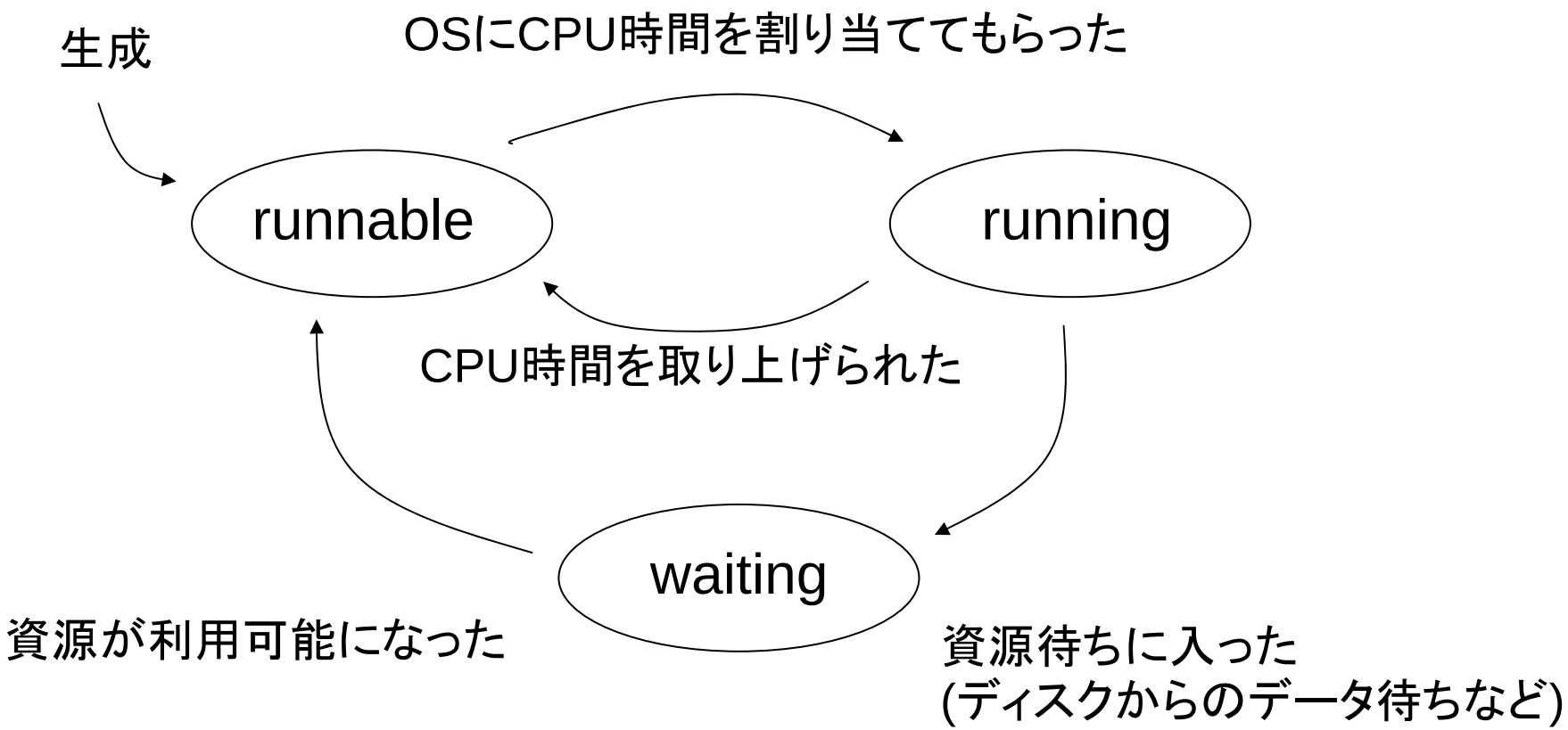
ジョブ
　周期タスクの場合，その各回の実行をジョブと呼ぶ

CPU時間の割り当て

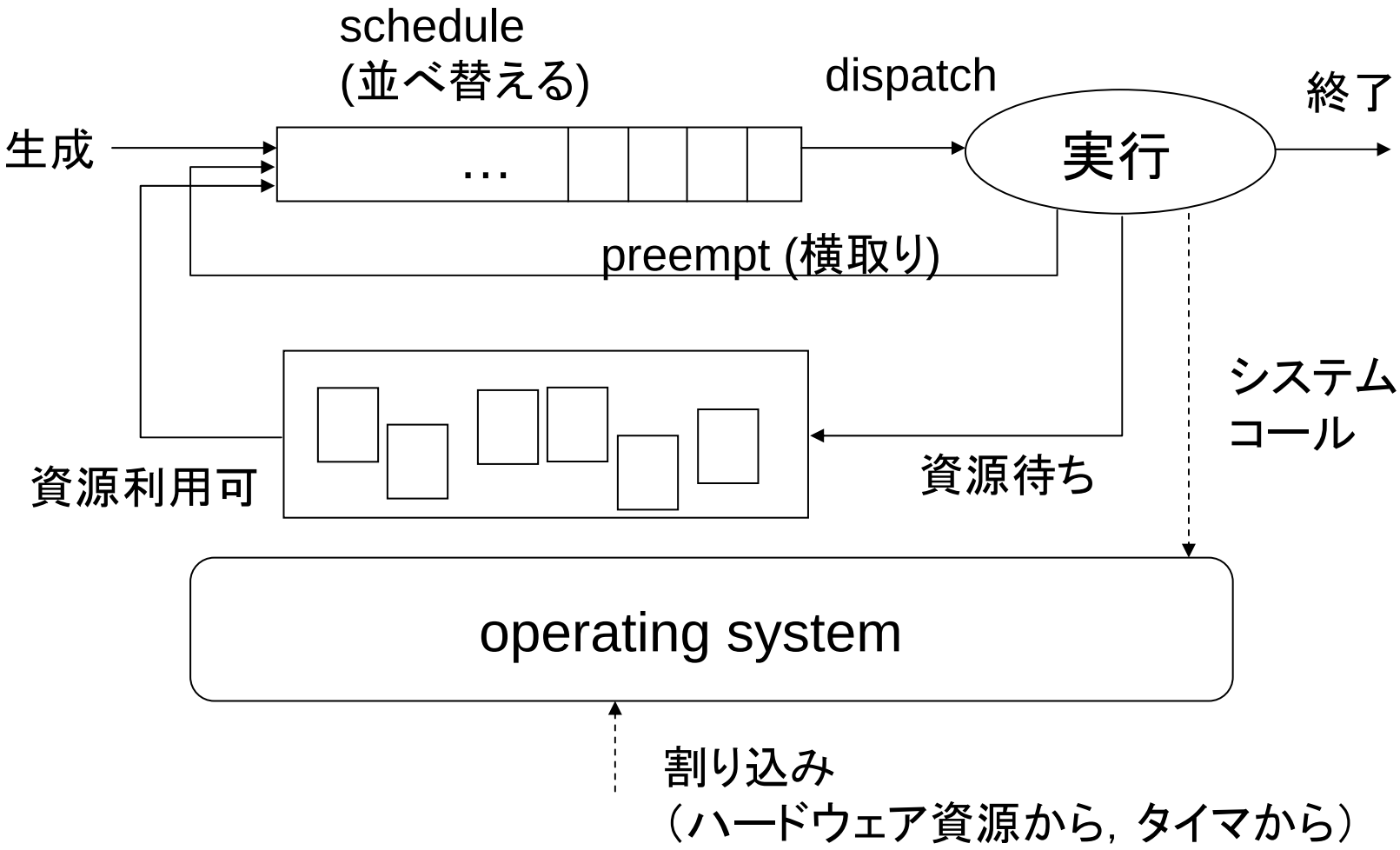


CPUが1個の場合

タスクの側から見た場合

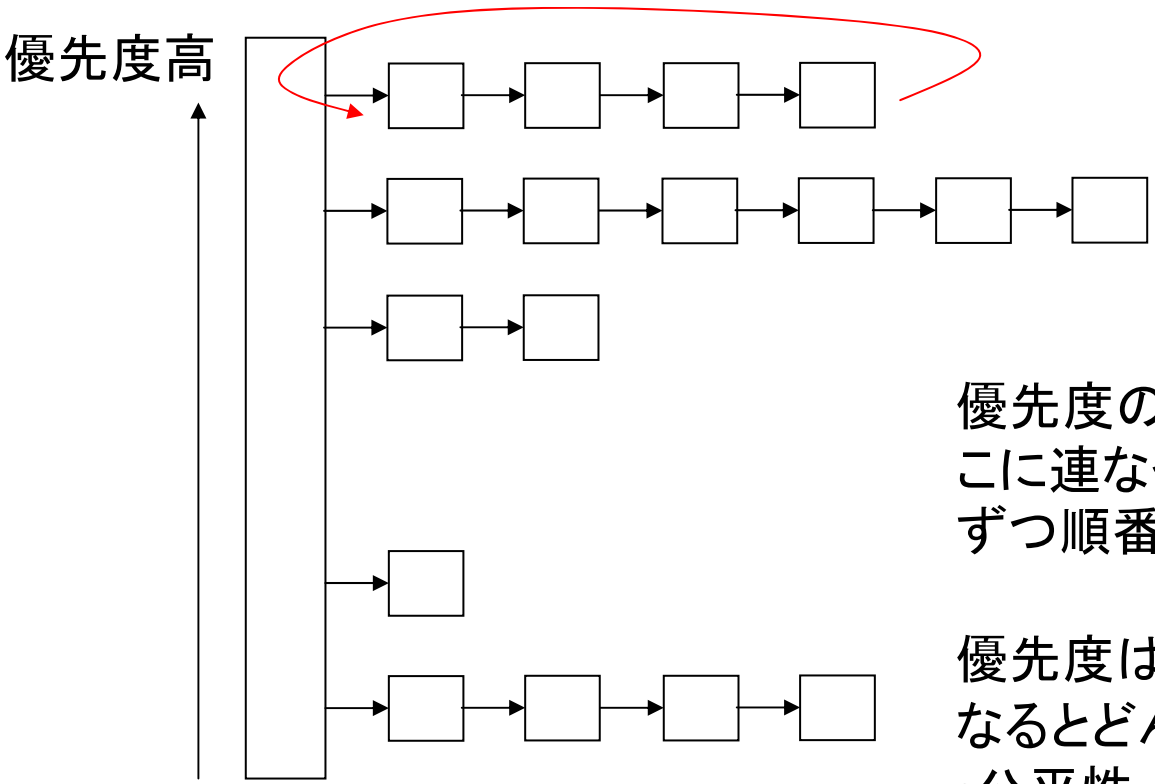


OSの側から見た場合


schedule
(並べ替える)
dispatch
終了
生成
…
実行
preempt (横取り)
システム
コール
資源利用可
資源待ち
operating system
割り込み
（ハードウェア資源から，タイマから）

汎用OSの場合

UNIXの例



優先度の一番高いキューを選び，そこに連なっているタスクを一定時間ずつ順番に実行 (round robin)

優先度は，CPU割り当て時間が長くなるとどんどん下がっていく
- 公平性
- インタラクティブなタスクの応答重視

リアルタイムOSの場合

- 公平なスケジューリングでは嬉しくない
- 各タスクに課せられた制約に応じた，適切なスケジューリングが必要 → むしろ，不公平でもいいから優先度を割り当てて，それをしっかり守ることが重要
  - 時間制約
  - タスク間の先行制約
  - 資源要求，資源の共有・競合

どのようにスケジュール（≒ 優先度）を定めるか?
→ リアルタイムスケジューリング理論

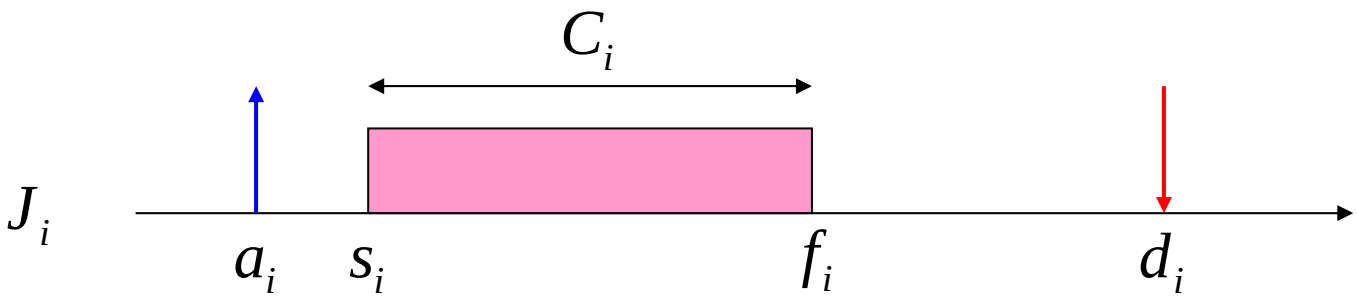
リアルタイムOSの例

UNIX系
　RT-Linux, ART-Linux, QNX, LynxOS, …

Windows系
　RTX, INtime, …

専用OS系
　VxWorks, ITRON, …

リアルタイムタスクの時間制約



J_i i 番目のジョブ

a_i arrival time (到着時間)

C_i computation time (計算時間)

d_i deadline (〆切)

s_i start time

f_i finishing time

デッドラインの種類

hard deadline

デッドラインをミスすると，壊滅的な結果が訪れる．
あるいは，デッドラインをミスすると即そのタスクの価値がゼロになる (後者を firm deadline と呼んで区別する場合もある)

soft deadline

デッドラインをミスすると，そのタスクの価値が減少する

デッドラインが (hard | soft) である (システム|タスク)を，

(hard | soft) real-time (system | task)

と呼ぶ場合がある．

スケジューリング問題

あるタスクの集合が与えられた場合，各時刻においてプロセッサを（あるいは他の資源を）どのタスクに割り当てるか，を解く問題

完全に一般の場合について解くのは困難.

→ いろいろと限定された状況で有効なスケジューリング手法を考える必要がある

計算が速ければいいというわけではない

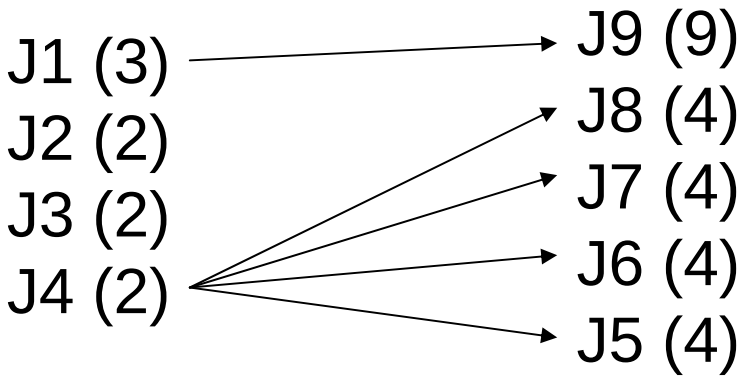
「スケジューリングなんて深く考えなくても，コンピュータが十分速ければいいんじゃないの?」に対する counter exampleを紹介する.

あるポリシーでスケジュールした場合にうまくいったとしよう.

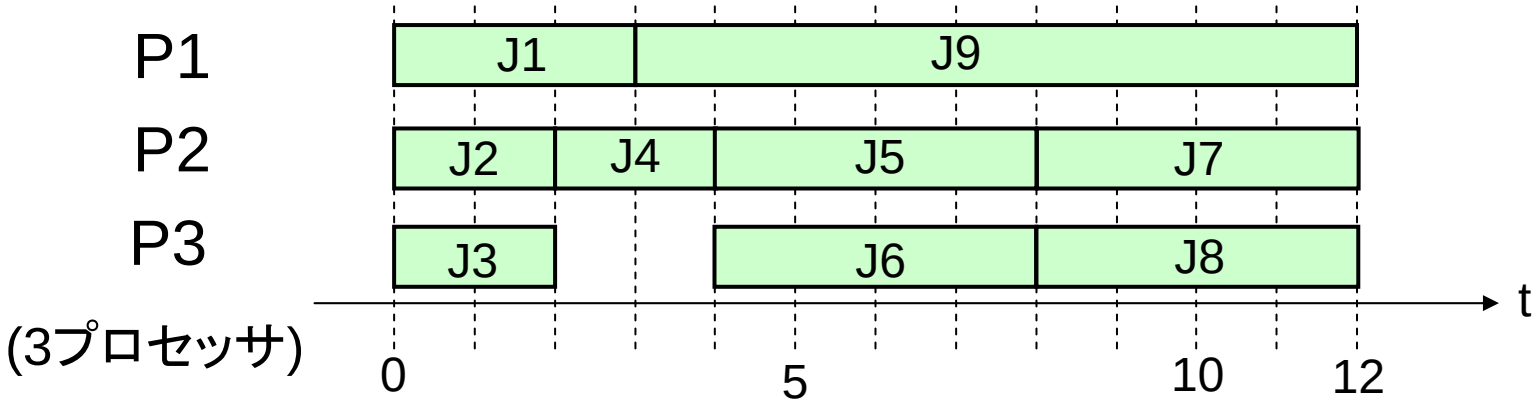
その場合に，プロセッサが速くなったり，タスク間の制約が弱まったり，プロセッサの数が増えたりしたら，さらにうまく行きそうな気がする

しかし，そうとは限らないという例

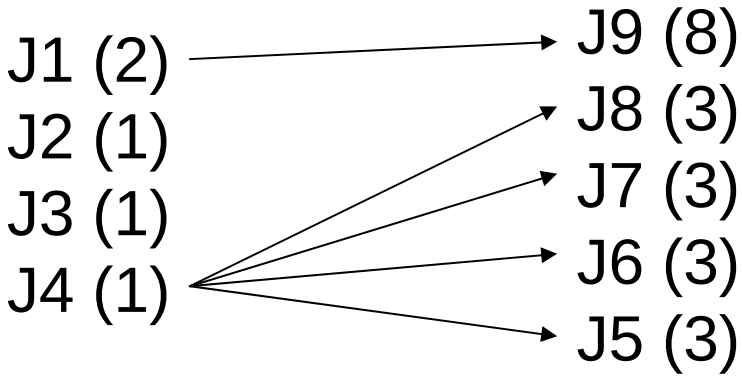
Richard's anomalies



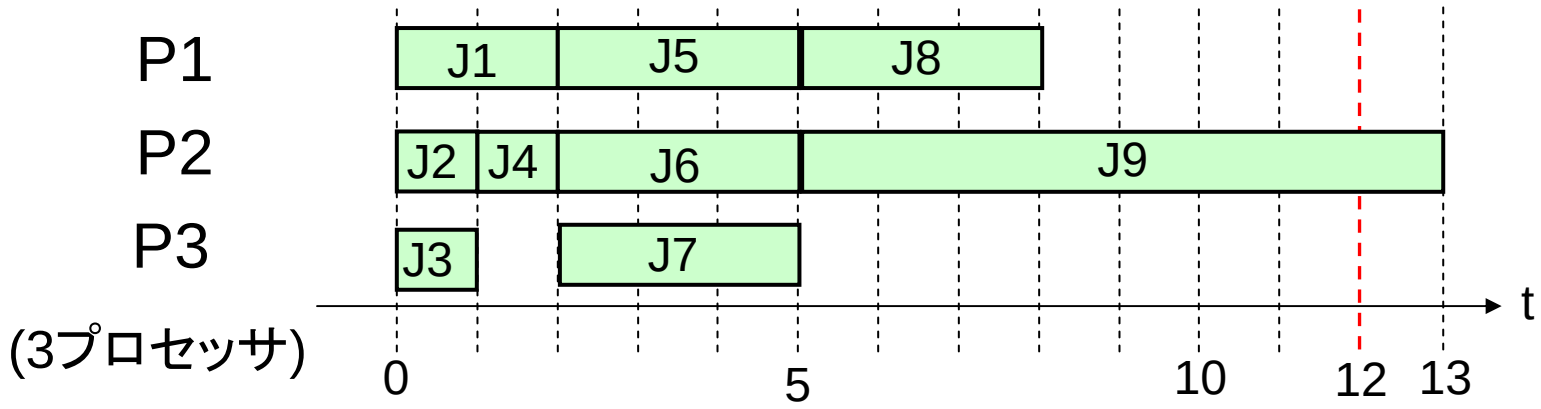
- 矢印は先行制約 (e.g. J9 は J1 の後にしか実行できない)
- 括弧内の数字は計算時間
- 優先度は J1 > J2 > … > J9
- 単純に優先度順にプロセッサを割り当てるとする



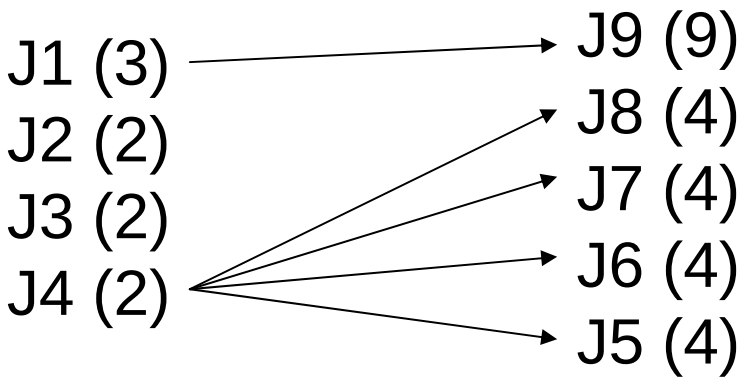
Richard's anomalies (計算の負荷が減った場合)



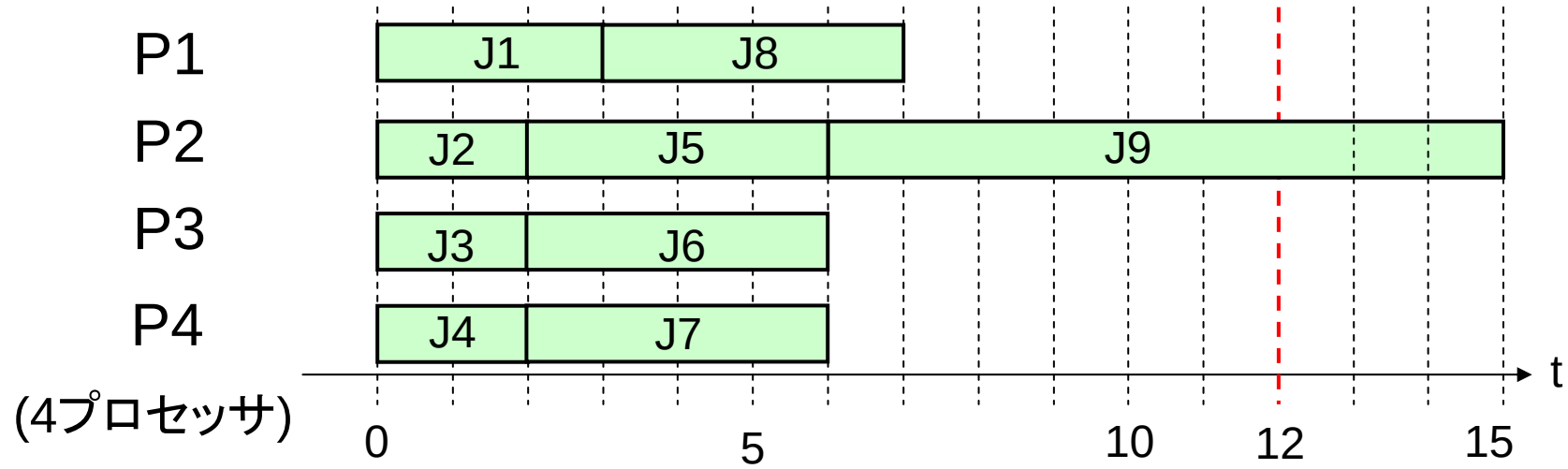
- 計算時間 (括弧内) が 1 ずつ減った



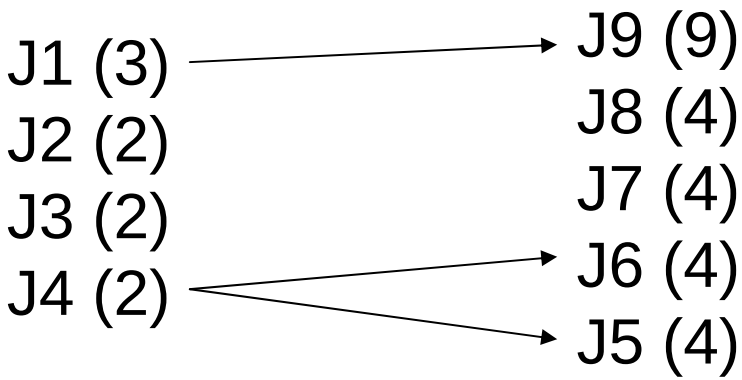
Richard's anomalies (プロセッサが増えた場合)



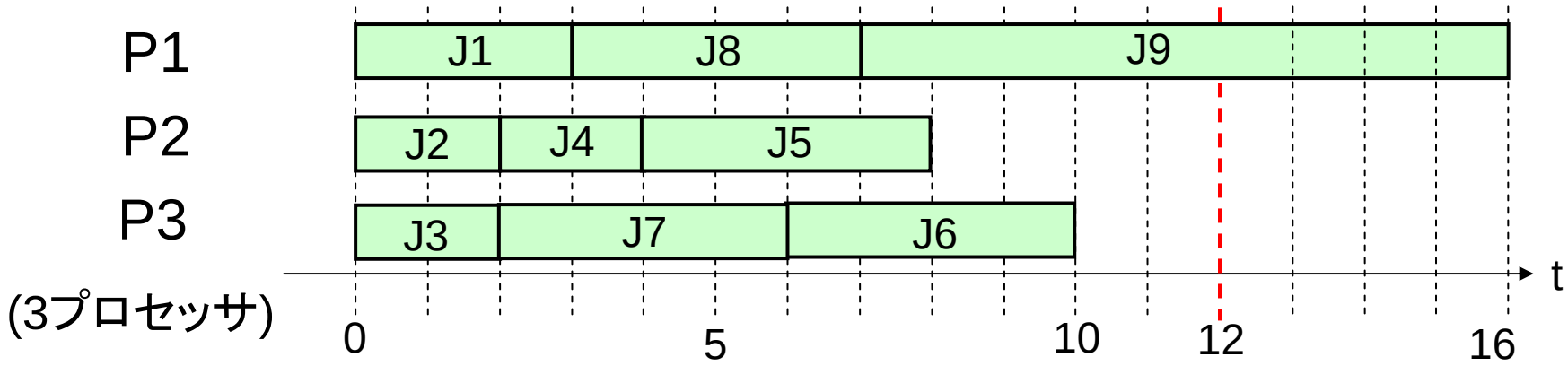
- プロセッサが4つになった場合



Richard's anomalies (先行制約が弱まった場合)



•先行制約が減った



代表的なスケジューリング手法

- EDF (Earliest Deadline First)
  単一プロセッサ，preemptive, 各タスクは独立

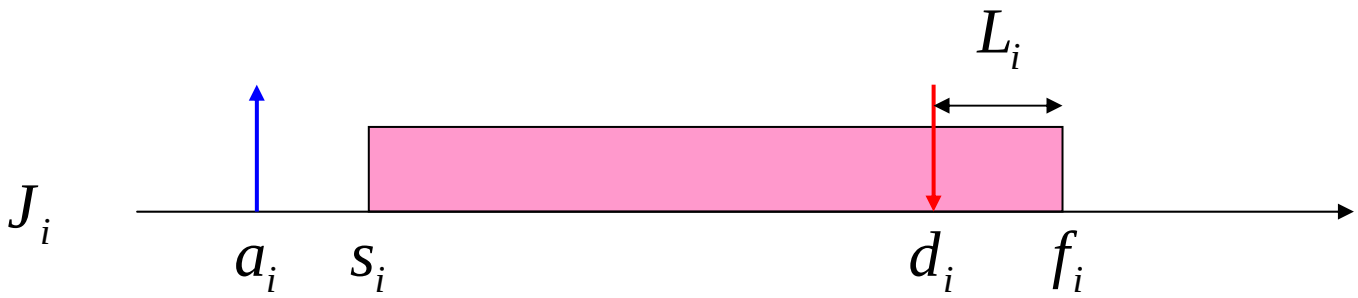
- RM (Rate Monotonic)
  周期タスク，単一プロセッサ，preemptive，各タスクは独立

prememptive: 実行中のタスクをいつでも中断できる

タスクが独立でない場合（資源の競合がある場合）や，先行制約がある場合は，さらにいろいろと考慮が必要

Earliest Deadline First スケジューリング

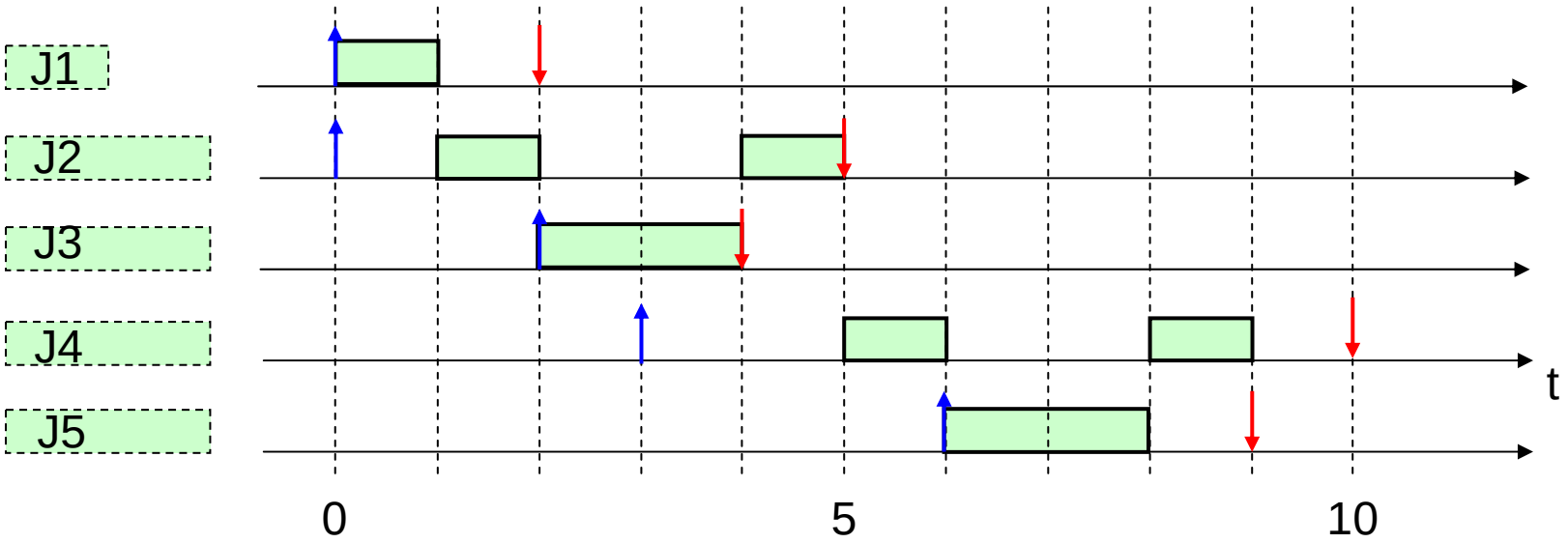
任意の arrival time を持つ独立なタスクの集合が与えられたとき，各時刻において，実行可能なタスクのうちデッドラインが最も早いものを実行するアルゴリズムは，最大遅延時間を最小化する意味で最適である



$$L_{\max} = \max_i (f_i - d_i)$$

例

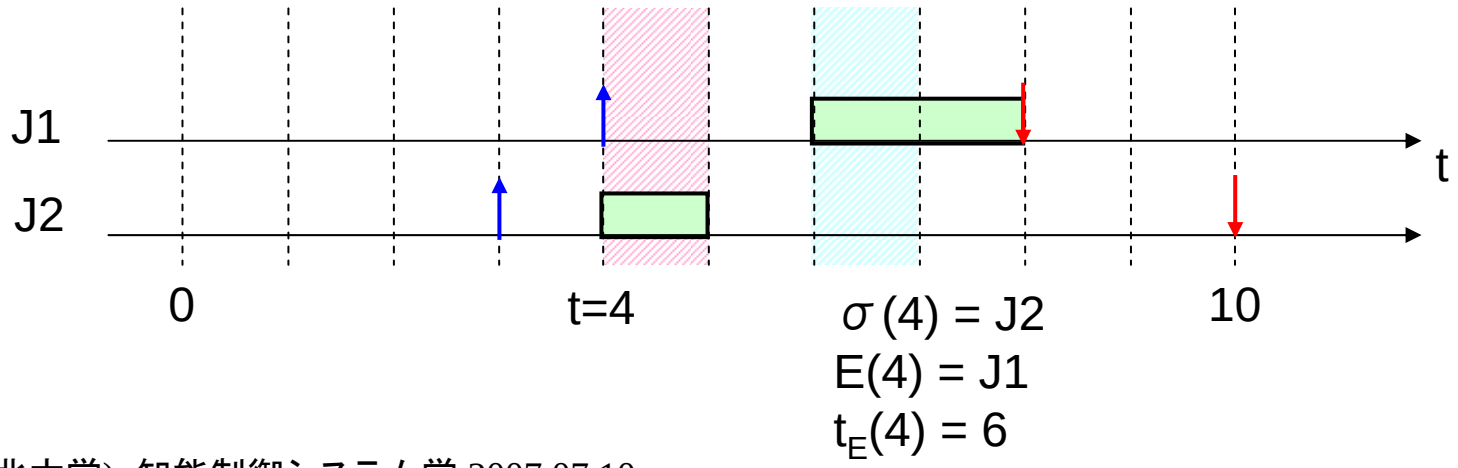
	J1	J2	J3	J4	J5
a_i	0	0	2	3	6
C_i	1	2	2	2	2
d_i	2	5	4	10	9



最適性の証明 (1/3)

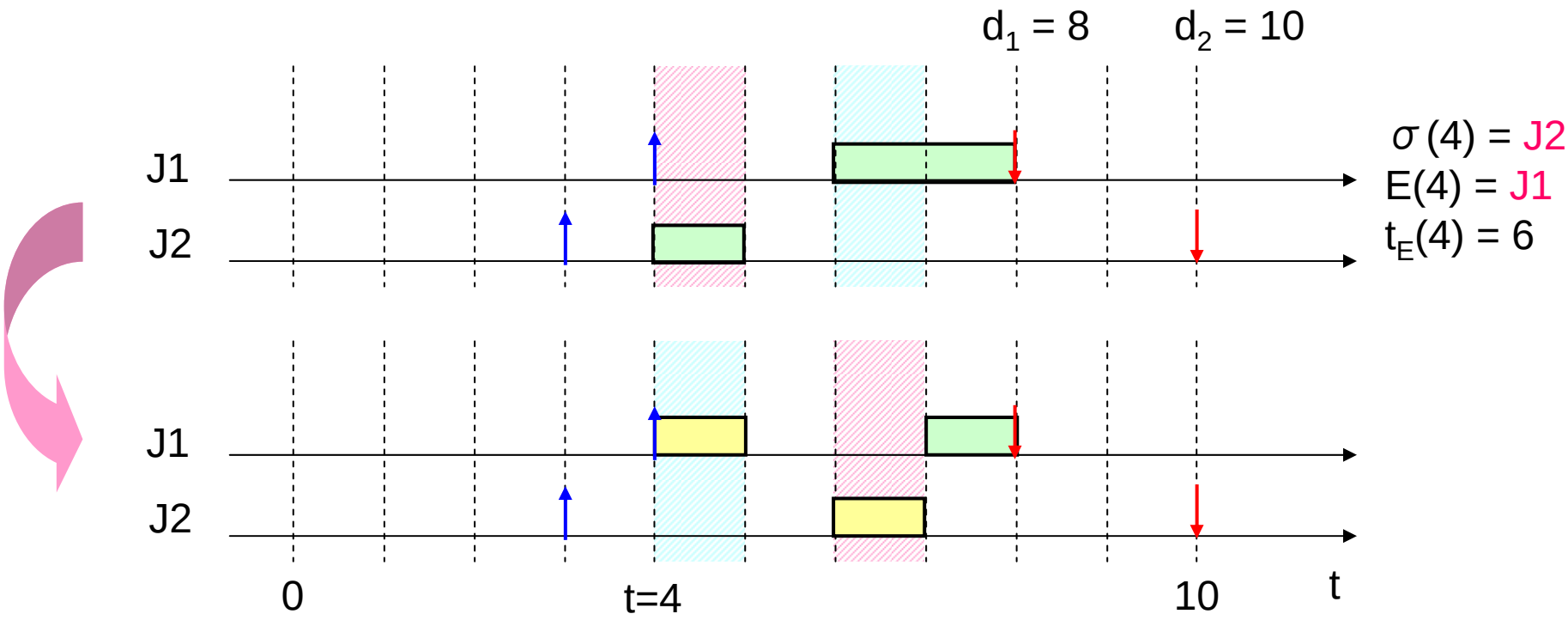
離散時間で考える（一般性は失わない）．単位時間の区間をスライスと呼ぶ

- スライス [t, t+1) で実行中のタスクを $\sigma(t)$ と書く
- ある時刻 t において実行可能なタスクのうちで，デッドラインの最も早いものを E(t) と書く
- 現在考えているスケジュールにおいて，t 以降で E(t) のスライスが最初に実行される時刻を $t_E(t)$ と書く

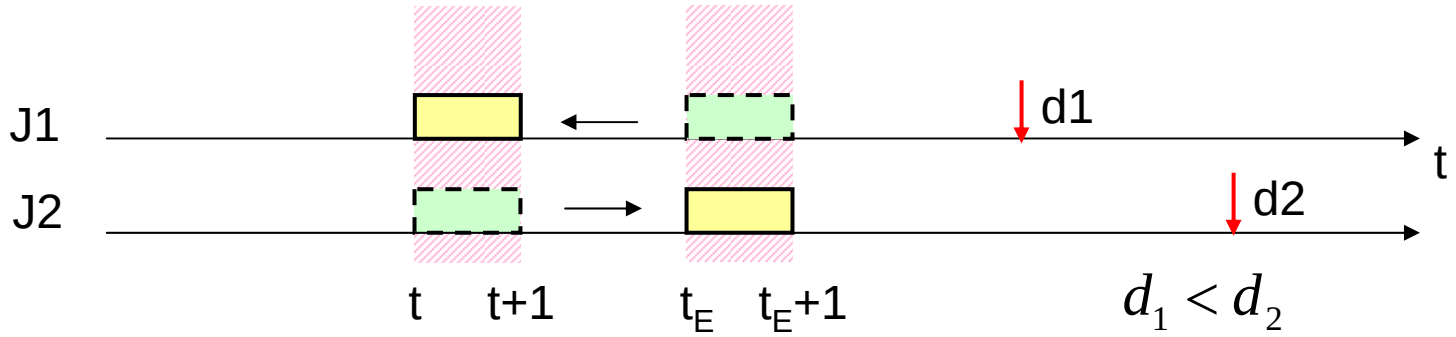


最適性の証明 (2/3)

EDF以外のスケジュールの場合，$\sigma(t) \neq E(t)$ であるような t が存在する.
このとき，スライス $[t, t+1)$ と $[t_E, t_E+1)$を交換しても，最大遅延時間は増加しないことを示せばよい



最適性の証明 (3/3)



交換する2タスクだけを考える. この交換によって, L_{max}が増加しなければよい. 交換後の変数はアポストロフィつきで表す (e.g. $f_1 \rightarrow f_1'$). 元のL_{max}は,

$$L_{\max} = \max(f_1 - d_1, f_2 - d_2)$$

L_{max}が変化し得るのは以下の2つのケース, またはこれらが同時に起きたときだけである.

$t_E + 1 = f_1$ だったとき, $L_1' = f_1' - d_1 < f_1 - d_1 \le \max(f_1 - d_1, f_2 - d_2) = L_{\max}$

$f_2 < t_E + 1$ だったとき,

$$L_2' = f_2' - d_2 \le f_1 - d_2 < f_1 - d_1 \le \max(f_1 - d_1, f_2 - d_2) = L_{\max}$$

よっていずれにせよ L_{max} は増加しない.

EDFな仕事術

「おお，そうか．ということは，いつも〆切ドリブンで仕事している俺は数学的に最適なんじゃないか!」

→ もしもあなたが，

- タスクを preemptive にスイッチすることができて，
- 各タスクが完全に独立していて，
- 各タスクの実行時間（最悪実行時間）が一定ならば

その通りです

References

- G. C. Buttazzo: Hard Real-Time Computing Systems – Predictable Scheduling Algorithms and Applications, Kluwer Academic, 1997.
- 藤倉: リアルタイム/マルチタスクシステムの徹底研究, TECH I Vol.15, CQ出版社, 2003.
- 特集: Windows, Linux を生かして産業用機器をリアルタイム制御しよう! ― 組み込みコンピュータを用いた産業用システム開発, Interface, Vol.33, No.8, 2007.