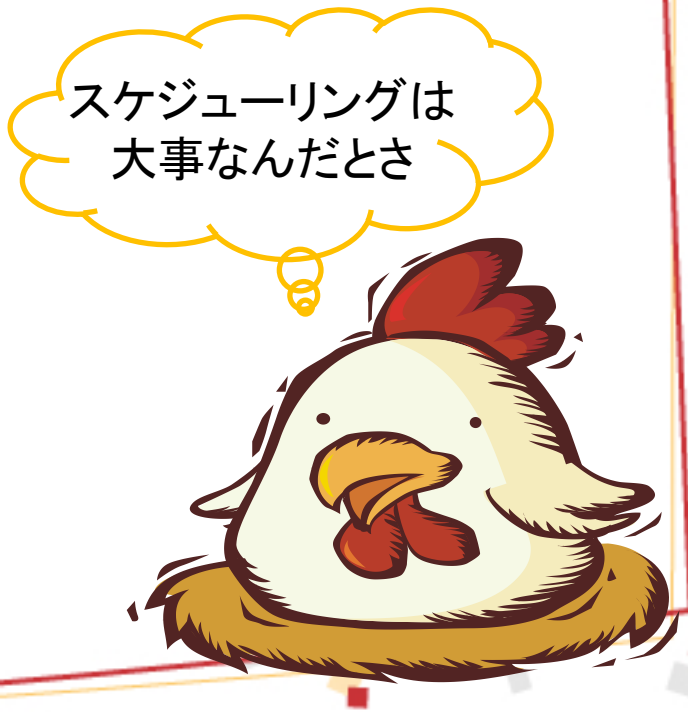
Multitasking

南山大学大学院　数理情報研究科　数理情報専攻
M2012MM022　森下　月菜

シナリオ

- 概要
- タスク管理システム
- タスクの種類
- 周期的実行の例
- 非周期マルチタスク
- タスクの状態遷移
- スケジューリング
  - ✓ 固定優先順位を用いたスケジューリング
  - ✓ レート単調スケジューリング
  - ✓ LLFスケジューリングアルゴリズム
  - ✓ EDFスケジューリングアルゴリズム
- スケジュール可能性のための十分条件
- EDFアルゴリズムの条件修正
- オーバーヘッドのコスト
- 参考文献



概要

- タスク
  - 非同期プログラミングの基本的要素
  - プログラムの実行単位
  - 登録されて取り消されるまで存在

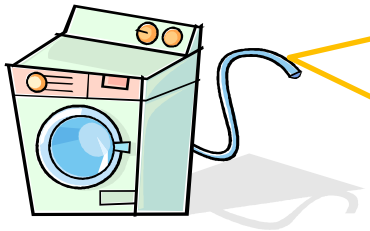
- マルチタスク
  - コンピュータにおいて複数のタスクを切り替えて実行できるシステム

タスクとスケジューリング
について説明します

タスク管理システム

- タスクの実行
  - ✓ システムのプログラムによって管理
  - ✓ スケジュールしてから実行
    - ✓ 事前にシステムの振る舞いが分かる場合，設計中にもスケジュール可能

    例：洗濯機



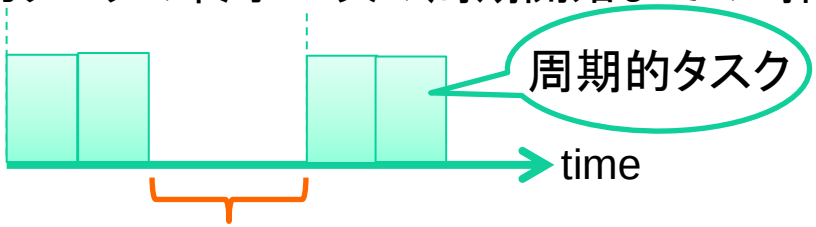
処理の順序
1. 水を入れる
2. 温める
3. 左右にドラムを回す
4. 水を抜く

- 静的なスケジューリング
  - ✓ 組み込みアプリケーションを実装するための最も安全な手段

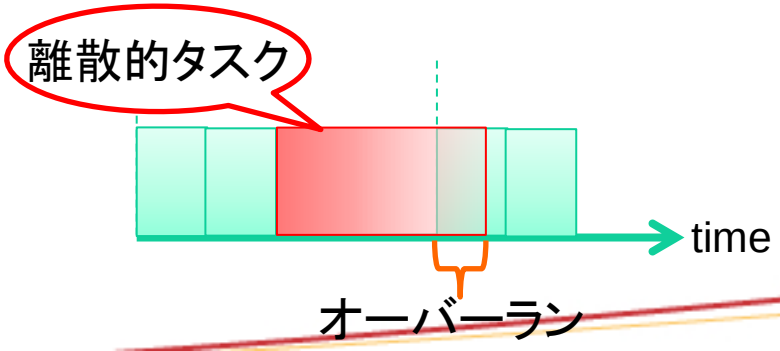
- 大体のシステムは動的
  - ✓ タスクによる処理を必要とするイベントが散発的
  - ✓ 動的にタスクを割り当てる必要がある
  - ✓ 実行時間にスケジュールされる必要がある
  - ✓ 柔軟性があるが，設計も難しい

タスクの種類

- 周期的タスク
  - ✓ 特定の期間内で実行される必要がある
  - ✓ 完全に互いに独立
  - ✓ 一定間隔で規則的に実行
- 離散的タスク
  - ✓ 一度実行するとしばらく実行しない
  - ✓ 一番最後の周期的タスクの終了～次の周期開始までの時間で実行

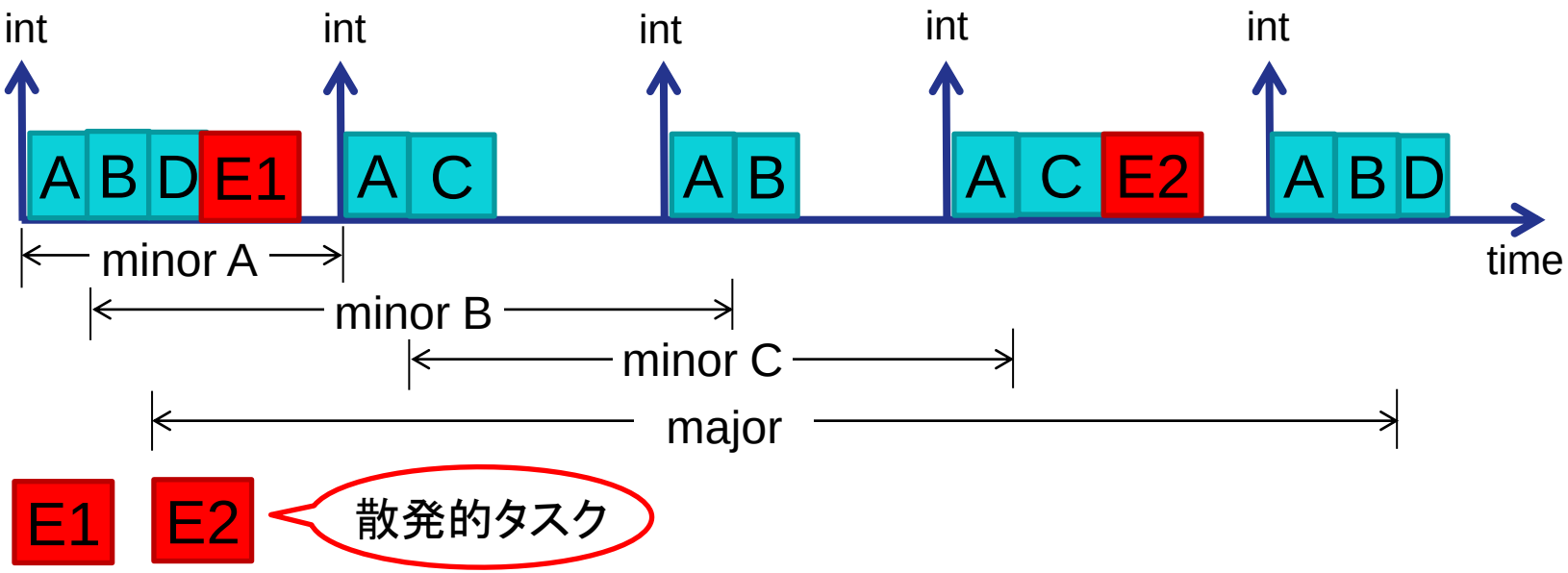


  - ✓ 離散的タスクの実行時間が実行可能時間より長い時，オーバーラン発生



周期的実行の例

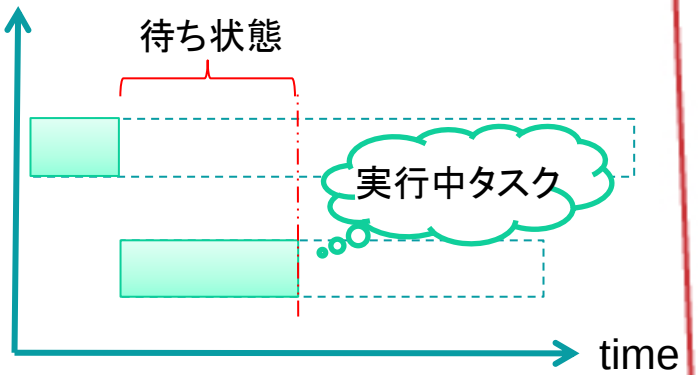
- 周期的タスクと散発的タスクの例



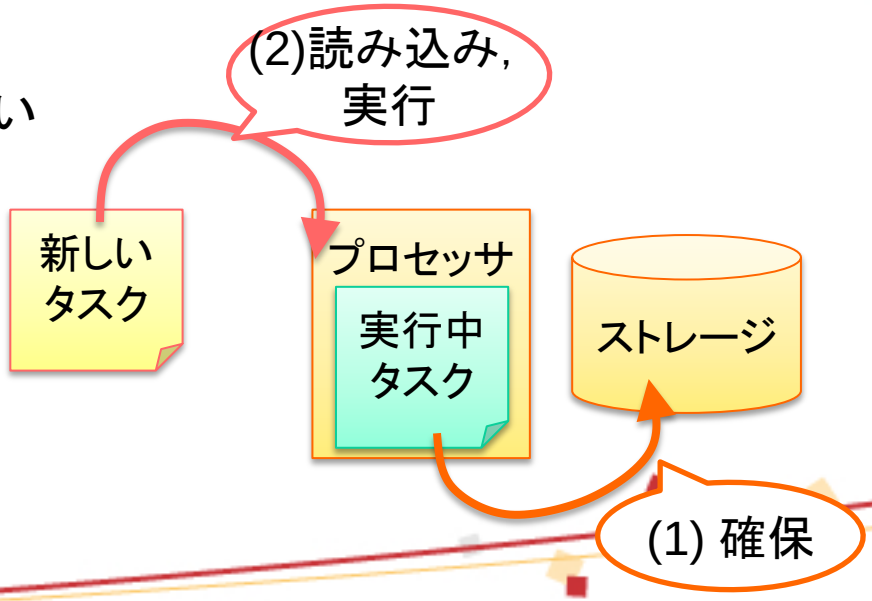
- 周期的プロセスの欠点
  - 周期はすべてのプロセスを考慮するために十分長い必要がある
  - 制御の安定性を保証するために十分短くする必要がある

非同期マルチタスク

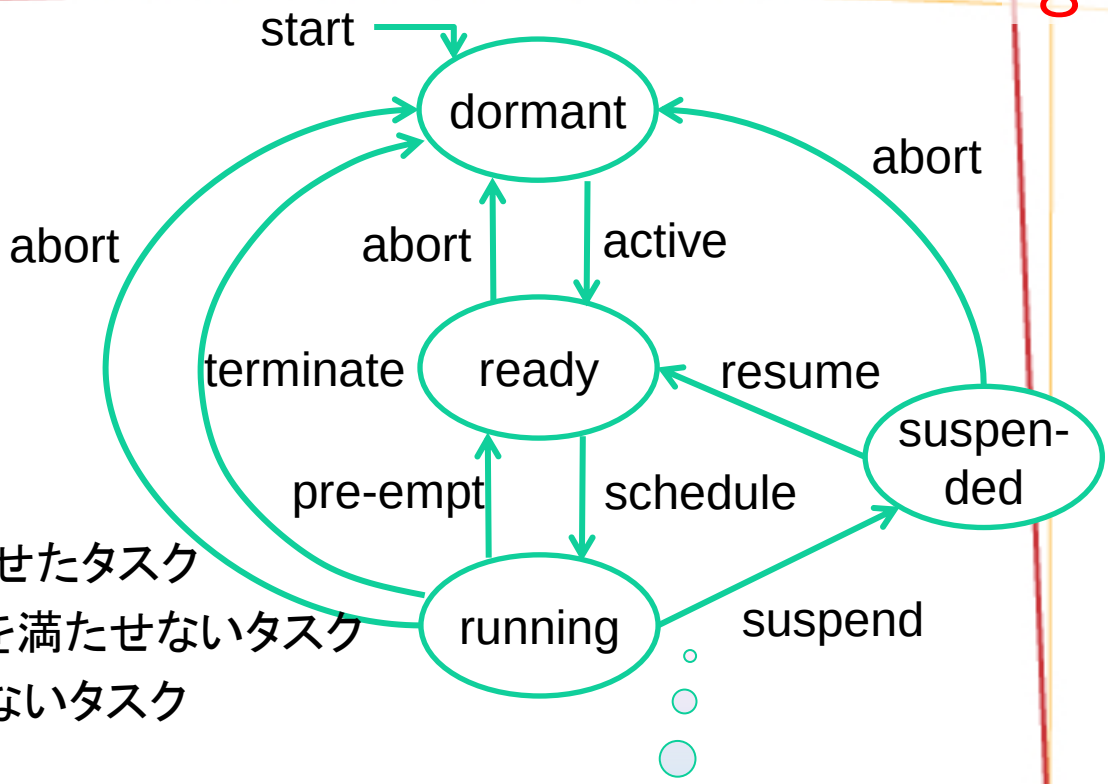
- 非同期マルチタスク
  - ✓ 動的システムに有効
    - ✓ タスクの瞬時性と同時性が提供できる
  - ✓ 実行しているタスクはプロセッサを占領可能
    - ✓ 他のタスクが実行すべき時は待ち状態へ移行



- プロセッサ上で実行するタスクの入れ替え
  - ✓ ストレージに実行中のタスクのコンテキストを確保
  - ✓ 新たにスケジュールされたタスクの状態の読み込み，実行
  - ➡ 何度も入れ替えると生産効率が悪い
    - ✓ 最小限に抑えるべき



タスクの状態遷移



- dormant
  - 初期化されたタスク
  - 実行中状態で目的を終了させたタスク
  - 実行中状態でデッドライン※を満たせないタスク
  - その他の状態で必要とされないタスク
- ready
  - イベントによって活性化されたタスク
  - 停止中状態から再開されたタスク
  - 実行中状態から割り込まれたタスク
    - スケジューリングを行う
- running
  - スケジューリング結果，最初のタスク
- suspended
  - 実行するためのリソース不足のタスク

特定のイベントが発生すると
タスクが遷移

※ デッドライン：処理開始から
処理を終了すべき時刻までの時間

スケジューリング

- プロセッサを得るためにタスクは競合
  - ✓ すべてのタスクが時間内に完了するよう適切なスケジュールが必要
  - ➡ スケジューリングの目的
    - ✓ 所定の時間内のタスクの機能の完了

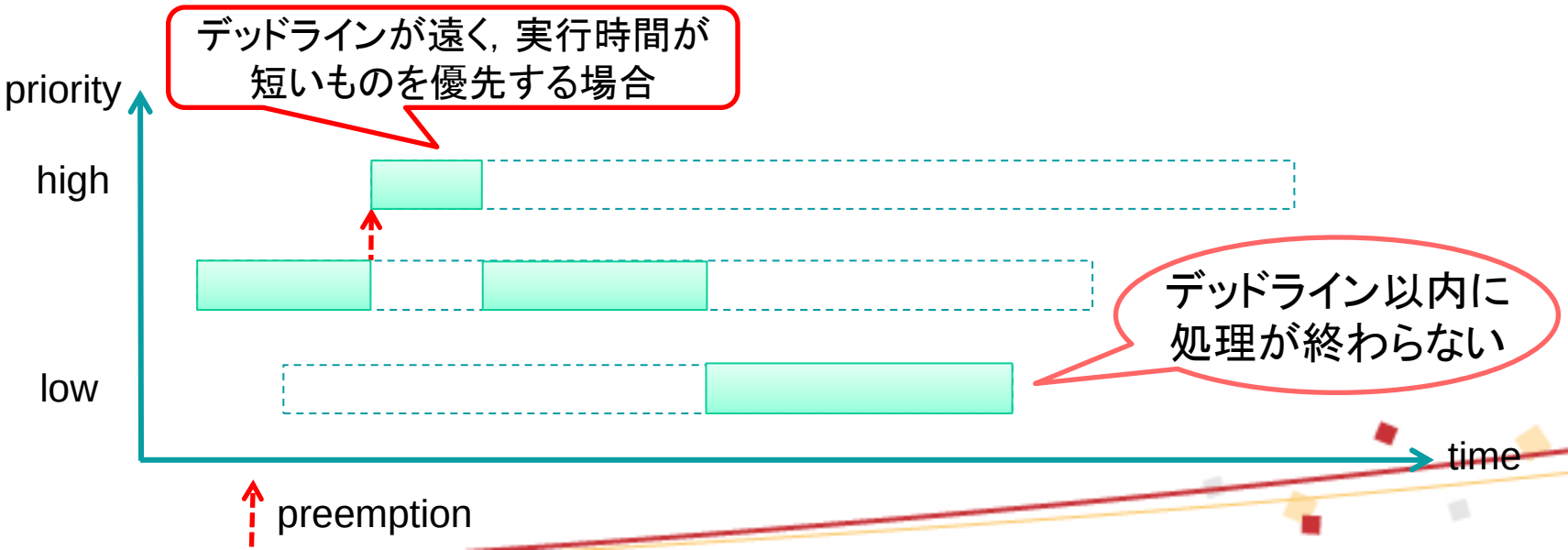
- タスクがデッドラインを達成不可能
  - ✓ ハードリアルタイムシステムにおいて重大な障害
  - ✓ 不適切なスケジューリング　or　システムの過負荷　が原因

- システムの過負荷
  - ✓ システムの動作の仕様の誤りが原因
    - ✓ ピーク時の負荷の理解が重要
      - ➡ 推定は困難

適切なスケジューリングを考慮
する必要がある

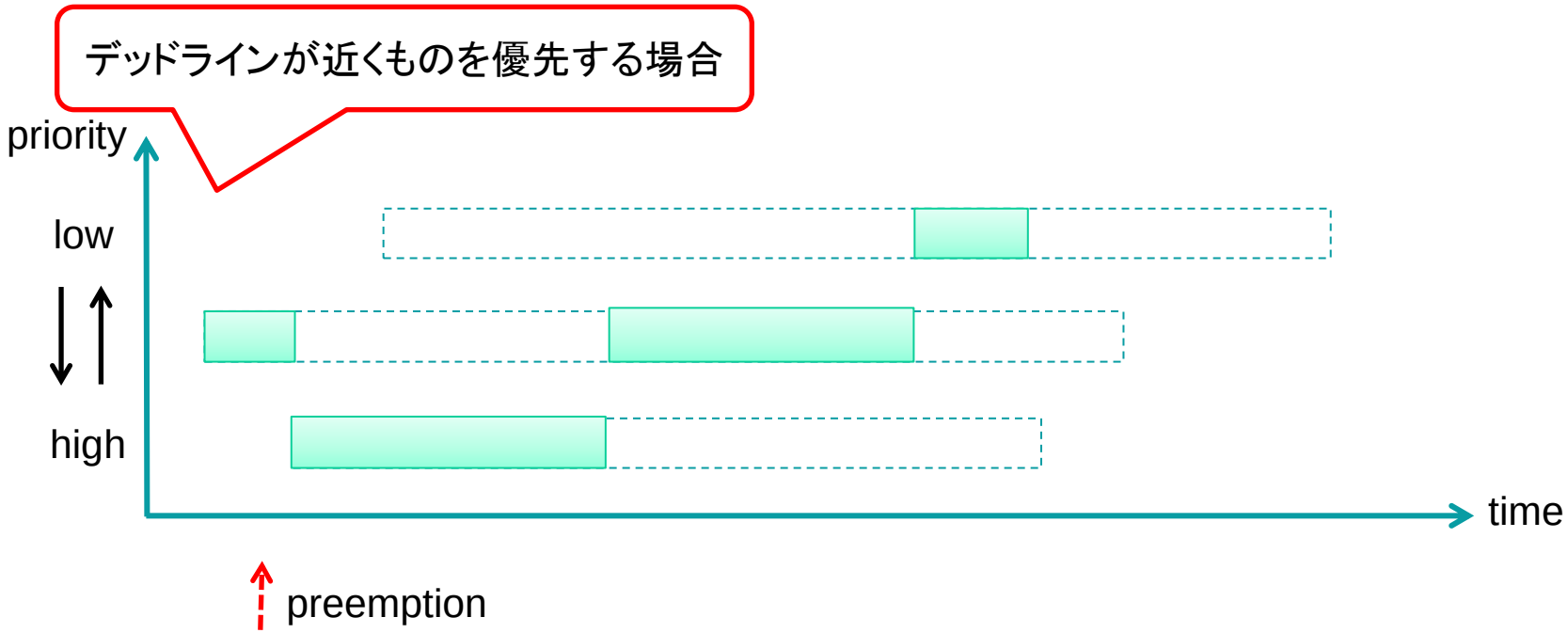
固定優先順位を用いたスケジューリング

- 多くの大衆向けOS
  - ✓ 固定の優先順位に基づいてスケジューリング
  - ✓ 実装は迅速かつ単純
  - ✓ 人為的に優先順位を負荷
    - ✓ 適切な優先順位をタスクに割り当てることが困難
- 固定優先順位によるスケジューリングの問題点
  - ✓ 優先順位が柔軟でない
    - ➡ システムの振る舞いに順応できない
  - ✓ リソースを本当に必要とするタスクに割り当てられない



固定優先順位を用いたスケジューリング

- 優先順位を変えれば，実行可能なスケジュールは簡単に作成可能



レート単調スケジューリング

- 優先順位を周期タスクの期間に応じて割り当て
  - n 個すべてのタスク
    - 時刻 t = 0 で同時に開始
    - 互いに独立

レート単調な優先順位割り当てが
実行可能なスケジュールを生成する条件

$$U \leq n\left(2^{\frac{1}{n}} - 1\right) \qquad (2.1)$$

プロセッサ
使用率

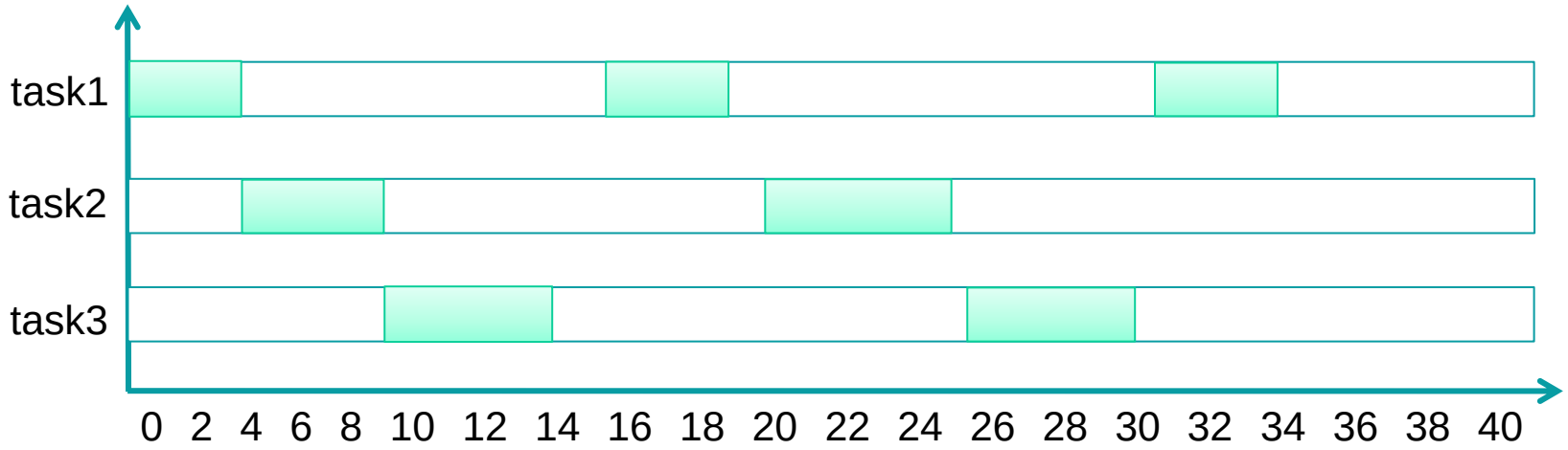
$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \qquad (2.2)$$

レート単調スケジューリングの実行可能な例

n = 3 の場合

	Period	Exec. time	Utilisation
Task1	15	4	0.27
Task2	20	5	0.25
Task3	25	5	0.20

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \qquad (2.2)$$



$$U \leq n\left(2^{\frac{1}{n}} - 1\right) \qquad (2.1) \text{ より}$$

$$U = 0.27+0.25+0.20 = 0.72 \leq 3\,(2^{1/3} - 1) \fallingdotseq 0.7797$$

レート単調スケジューリング

- タスクの数が多い場合，式2.1は以下のように近似可能

$$\lim_{n\to\infty} n\left(2^{\frac{1}{n}}-1\right) = \ln 2 \cong 0.69$$

プロセッサ使用率の30%を
放棄するとスケジュール通り
実行可能

- 周期タスクの集合が実行可能であるための条件

$$\prod_{i=1}^{n}\left(\frac{C_i}{T_i}+1\right) \leq 2.$$

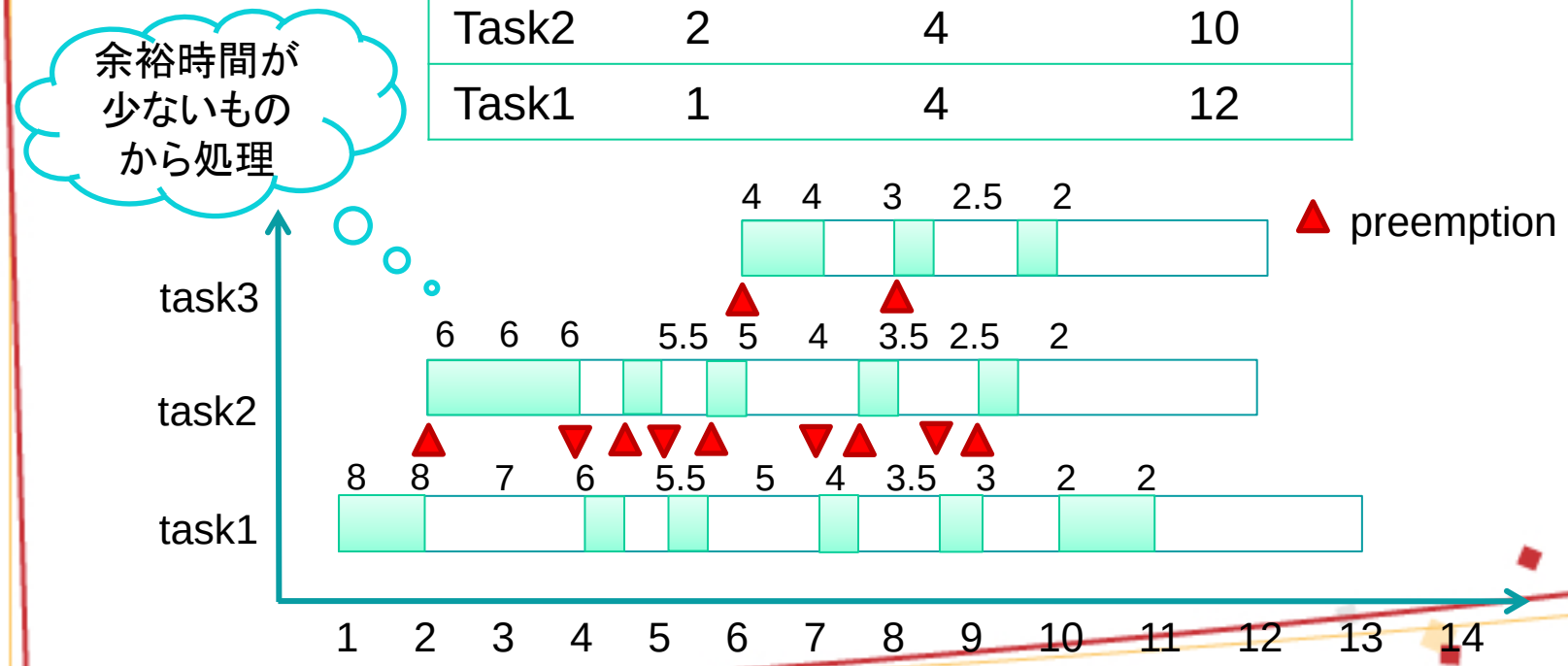
p.18の n = 3 の場合

(0.27+1)(0.25+1)(0.20+1) = 1.905 ≦ 2

LLFスケジューリングアルゴリズム

- LLF(least laxity first)
  - プロセスの余裕時間に基づいて優先度を設定
  - 組み込みシステム
    - 特にマルチプロセッサシステムで利用
  - 実行するタスクの切り替えにかかるオーバヘッドが大きい

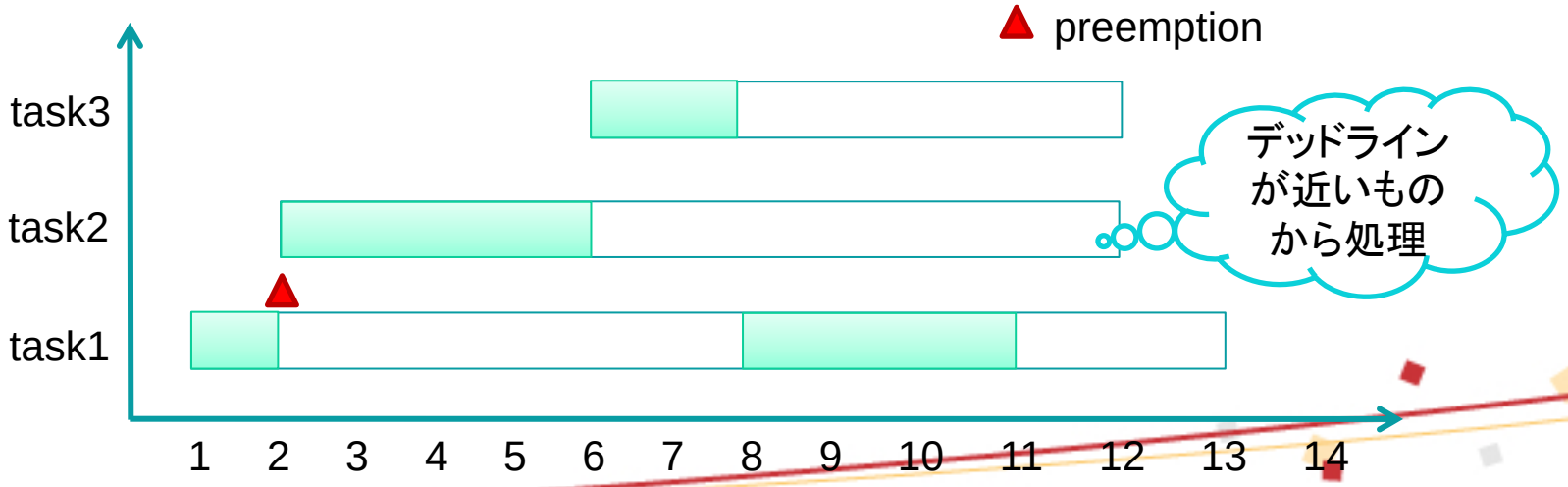
	arrival	exec. time	due after
Task3	6	2	6
Task2	2	4	10
Task1	1	4	12



EDFスケジューリングアルゴリズム

- EDF(earliest deadline first)
  - リアルタイムOSで使用される動的スケジューリングの一種
  - スケジューリングイベントが発生すると最もデッドラインが近いプロセスを選択
    - タスクはデッドラインの昇順でスケジューリング
  - デッドラインが近いもの，タスクの終了以外，実行するタスクの切り替えが不要

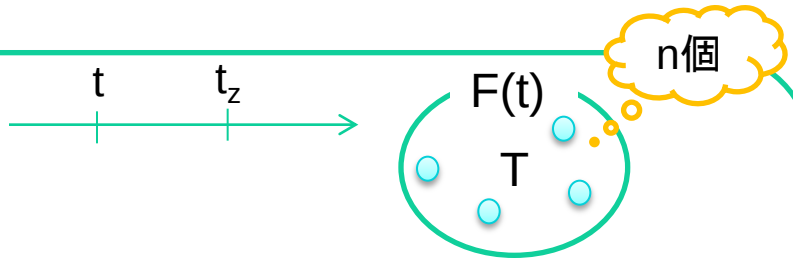
	arrival	exec. time	due after
Task3	6	2	6
Task2	2	4	10
Task1	1	4	12



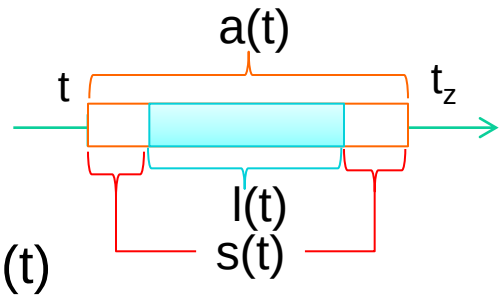
スケジュール可能性のための十分条件

- EDF
  - デッドラインによって実行可能状態のタスクをスケジュール

- 任意の時間 t ：$0 \leq t < \infty$
- タスク　　T：$T \in F(t)$
  - デッドライン t_z：$t_z > t$
- 実行可能状態のタスクの集合 F(t)：n個のタスクを持つ



- 応答時間 $a(t) = t_z - t$
- 要求される残りの実行時間　l(t)
- スラックタイム, マージン $s(t) = a(t) - l(t)$

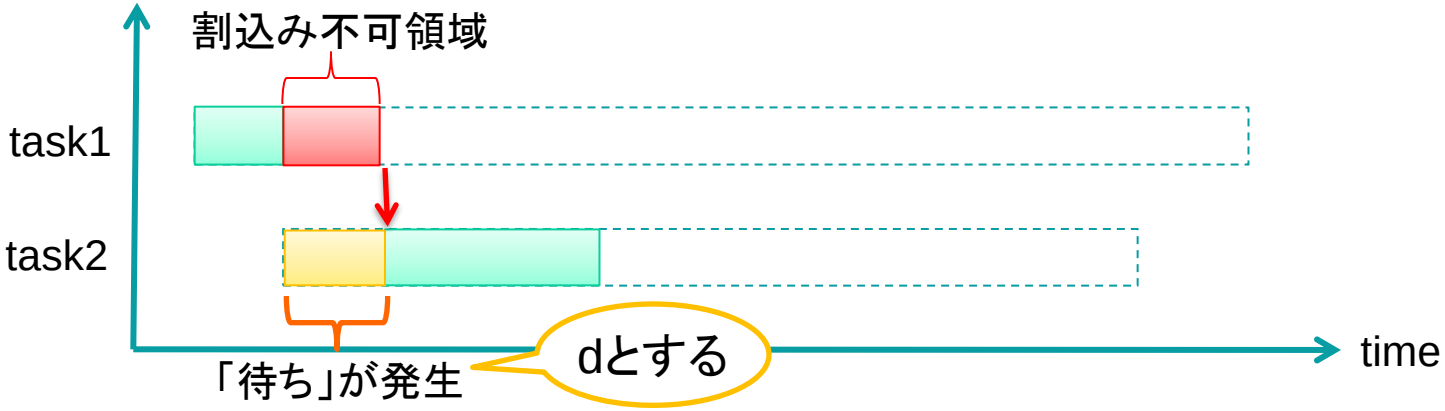


タスクセットF(t)のすべてのタスクが
デッドラインを達成するための必要十分条件

$$a_k \geq \sum_{i=1}^{k} l_i , k = 1, \cdots, n \qquad (2.3)$$

EDFアルゴリズムの条件修正

- EDFアルゴリズム
  - 任意のいずれかの時点で割り込まれる場合に最適な方法
    - 一般的にタスクは割り込みできない領域を持つ



実行中のタスク T_j よりも前にある，すべてのタスクが満たすべき条件

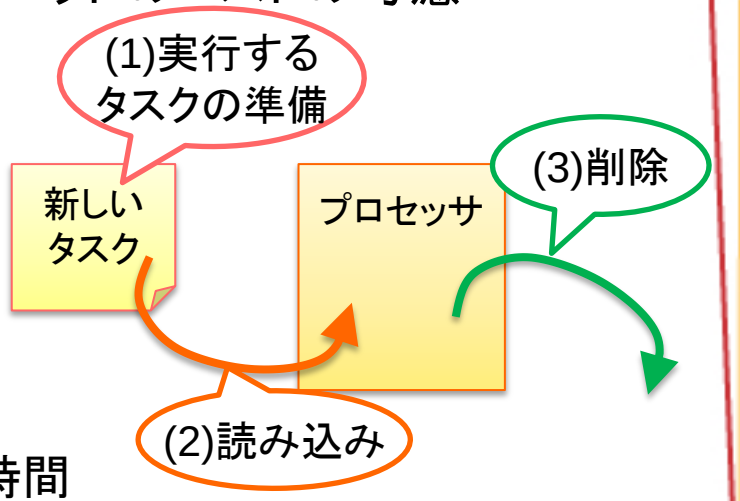
$$a_i(t) \geq d + \sum_{k=1}^{i} l_k(t), i = 1, \cdots, j-1 \qquad (2.4)$$

実行中のタスク T_j よりも後にある，すべてのタスクが満たすべき条件

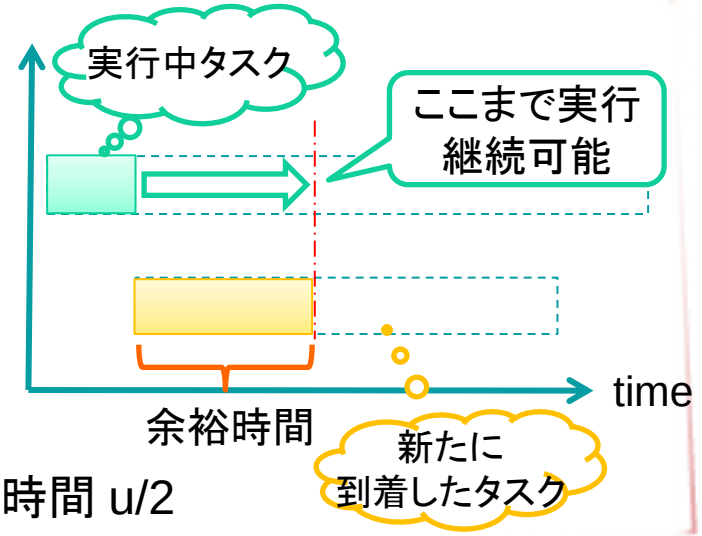
$$a_i(t) \geq \sum_{k=1}^{i} l_k(t), i = j, \cdots, n \qquad (2.5)$$

オーバーヘッドのコスト

- 実行するタスクの切り替えによるオーバーヘッドのコストの考慮
  - 実行するタスクの切り替えの回避可能か
  - 残りの切り替えコストを考慮可能か



- 実行するタスクの切り替え
  - タスクが正常に終了後
    1. 実行する準備をするのに必要な時間
    2. レジスタにタスクをロードするための時間
    3. タスクを削除するための時間
  - タスクの変更はどのような状況下でも必要
    - ➡ 実行するタスクの切り替えの回避は困難



- 残りの切り替えコストの考慮
  - 余分な割り込みを回避することでコスト削減
    - 切り替えに必要な時間: u
    - タスクを保存する or 復元するのに必要な時間 u/2

スケジュールの実行可能条件

実行中のタスクよりも前にある，すべてのタスクが満たすべき条件

$$a_i(t) \geq d + \frac{u}{2} + \sum_{k=1}^{i} l_k(t), i = 1, \cdots, j-1 \qquad (2.6)$$

割り込みの待ち＋
タスクの保存，復元時間

実行中のタスクよりも後にある，すべてのタスクが満たすべき条件

$$a_i(t) \geq u + \sum_{k=1}^{i} l_k(t), i = j, \cdots, n \qquad (2.7)$$

割り込みがないため，タスク切り替え
時間のみを考慮する

参考文献

- Domen Verber, et al., Multitasking, 2008, Distributed Embedded Control Systems, Pages 29-45.
- Wikipedia, http://ja.wikipedia.org/.
- 野口健一郎, オペレーティングシステム, 2002, オーム社.

Multitasking

END

南山大学大学院　数理情報研究科　数理情報専攻
M2012MM022　森下　月菜