Redistribution Aware Two-Step Scheduling for Mixed-Parallel Applications

0853020 **松本優人**

高性能コンピューティング講座 本多・近藤研究室

2009 **年** 1 **月** 5 **日**

論文の趣旨

論文タイトル

Redistribution Aware Two-Step Scheduling for Mixed-Parallel Applications

- **スタティックスケジューリングにおける新しいアルゴリズムの提案**
- **タスク間の並列性**
- **タスク内の並列性**

} の両方を考慮

論文の趣旨

論文タイトル

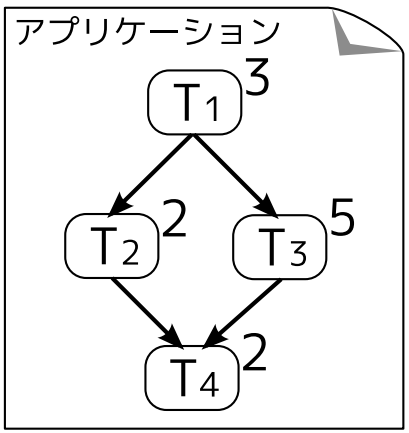
Redistribution Aware Two-Step Scheduling for Mixed-Parallel Applications

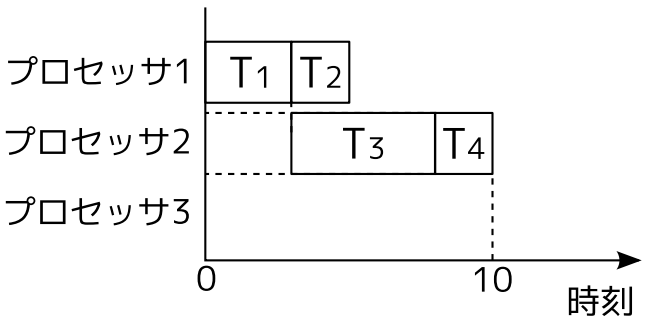
- **スタティックスケジューリングにおける新しいアルゴリズムの提案**
- **タスク間の並列性**
- **タスク内の並列性**

} の両方を考慮

スケジューリング

- アプリケーションをタスクに分割し、グラフで表現
- タスクを**どのプロセッサ**で**どの時刻**に処理するかを決定





- 実行前に行うスケジューリングをスタティックスケジューリングと呼ぶ

論文の趣旨 (再掲)

論文タイトル

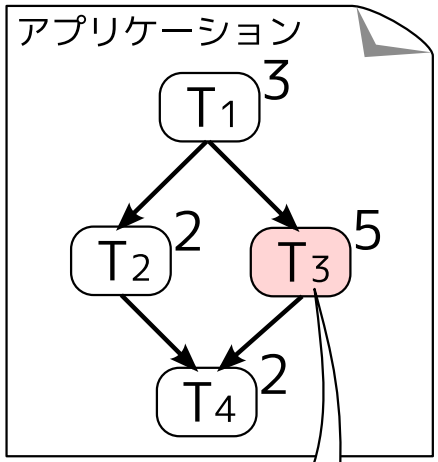
Redistribution Aware Two-Step Scheduling for Mixed-Parallel Applications

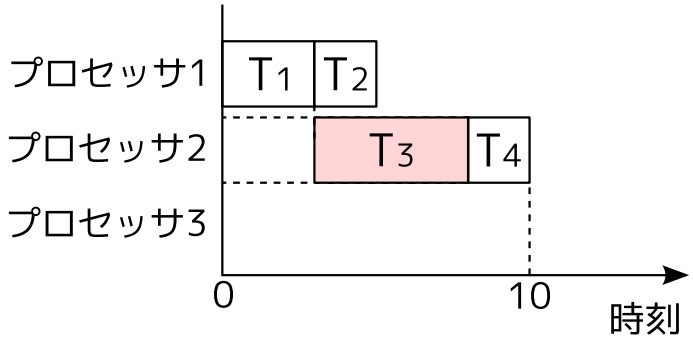
- **スタティックスケジューリングにおける新しいアルゴリズムの提案**
- **タスク間の並列性**
- **タスク内の並列性**

} の両方を考慮

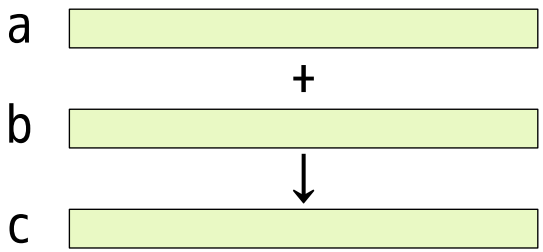
Mixed-Parallel

- **タスク間の並列性:互いに独立したタスクは並列処理できる**
- **タスク内の並列性:1 タスクを複数プロセッサで並列処理できる**





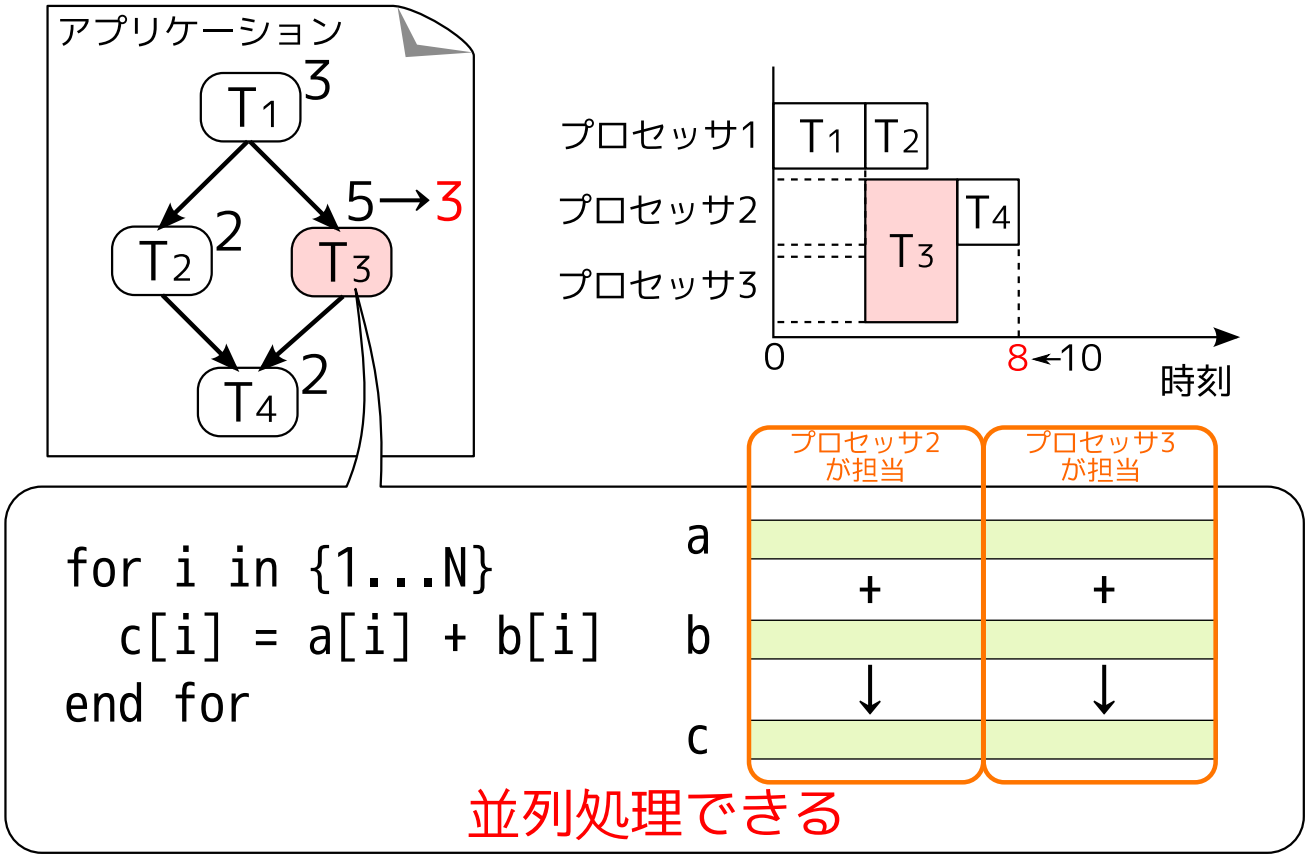

```
for i in {1...N}
  c[i] = a[i] + b[i]
end for
```



並列処理できる

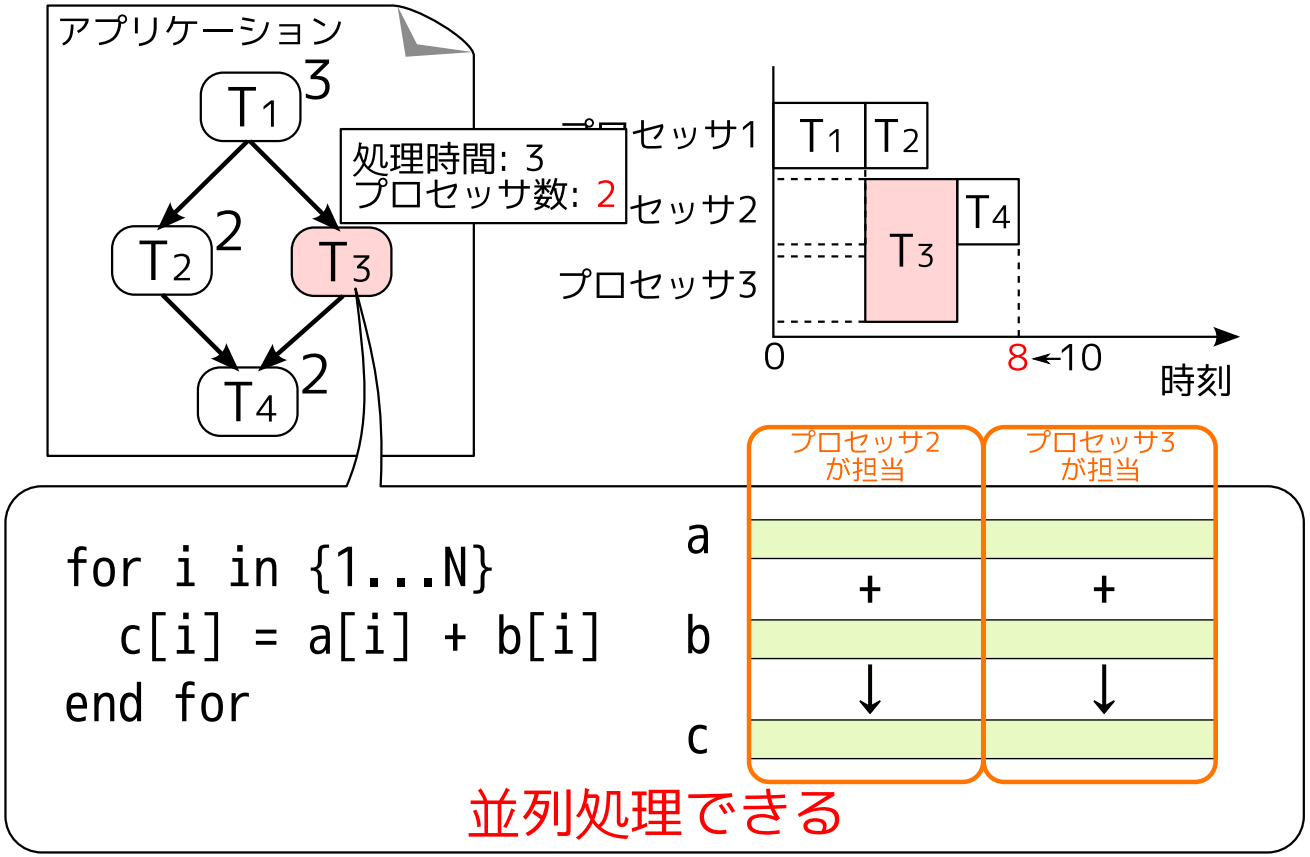
Mixed-Parallel

- **タスク間の並列性:互いに独立したタスクは並列処理できる**
- **タスク内の並列性:1 タスクを複数プロセッサで並列処理できる**



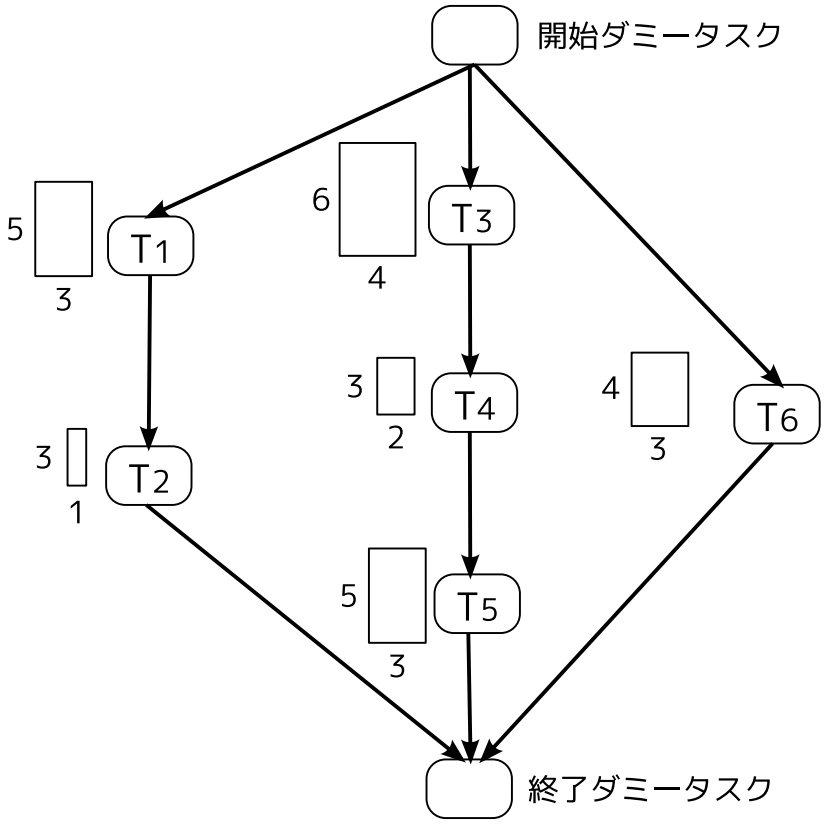
Mixed-Parallel

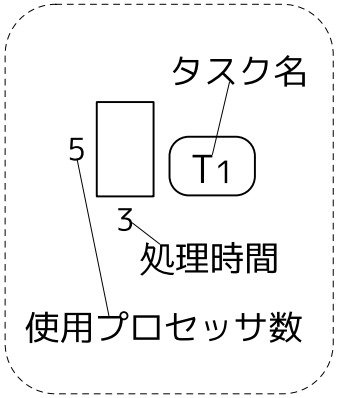
- **各タスクには、処理に要する時間だけでなく**
  使用するプロセッサ数の情報も必要

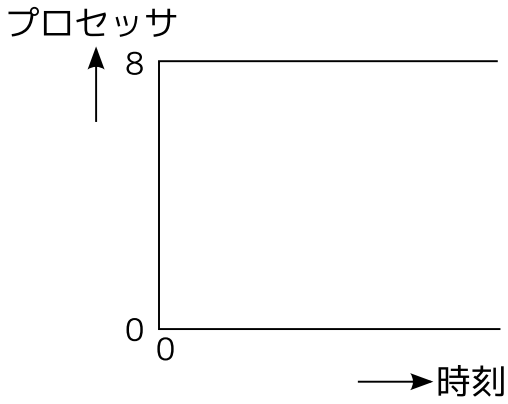


Mixed-Parallel に対するスケジューリング

- 各タスクは**処理時間**および**使用プロセッサ数**を持つ

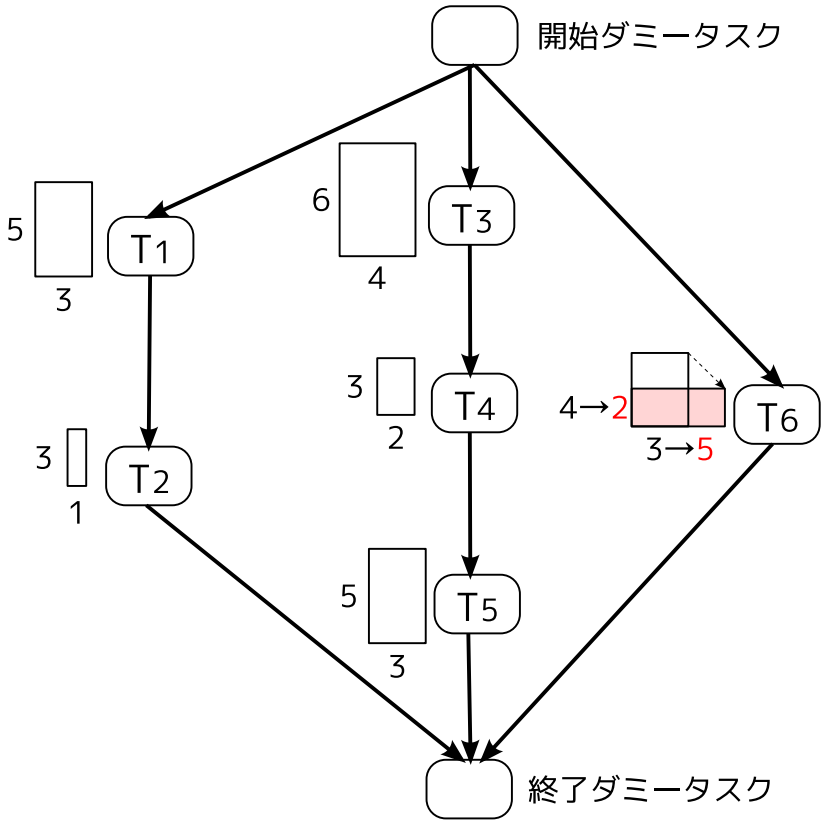


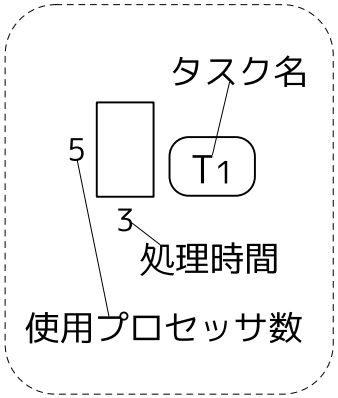


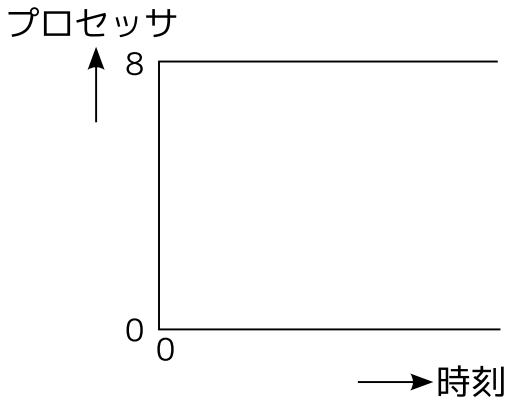


Mixed-Parallel に対するスケジューリング

- **使用プロセッサ数は変更可能で、処理時間は使用プロセッサ数に従う**

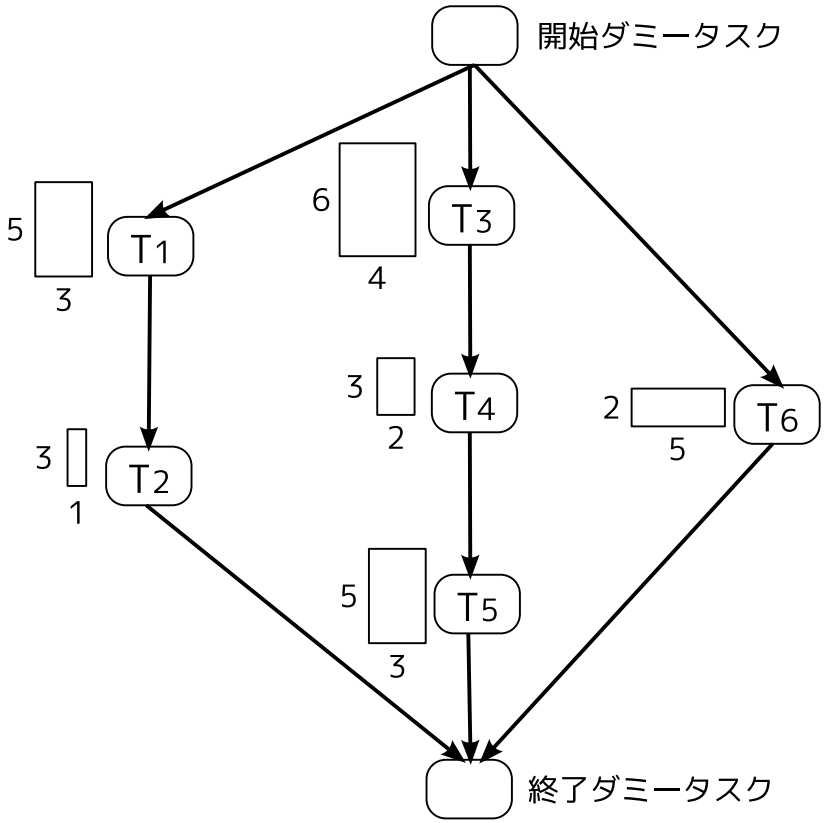


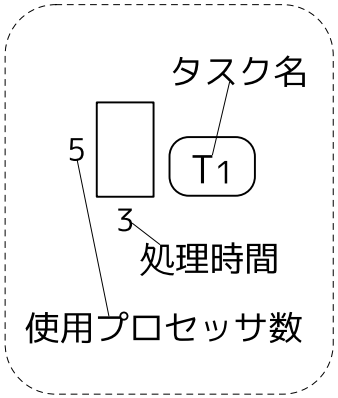


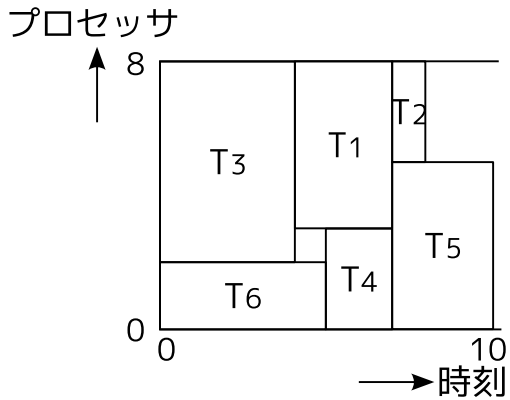


Mixed-Parallel に対するスケジューリング

- **使用プロセッサ数は変更可能で、処理時間は使用プロセッサ数に従う**

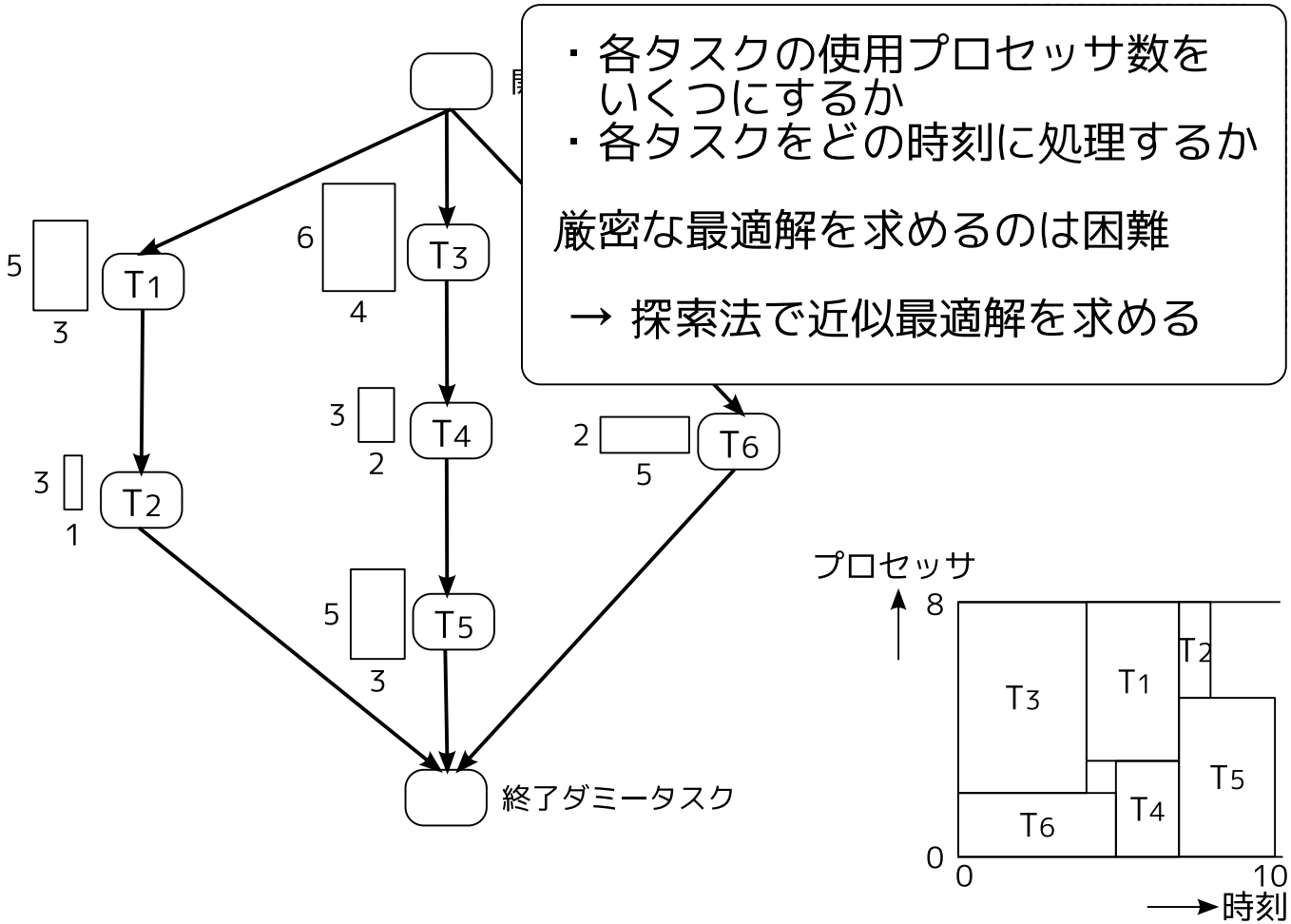






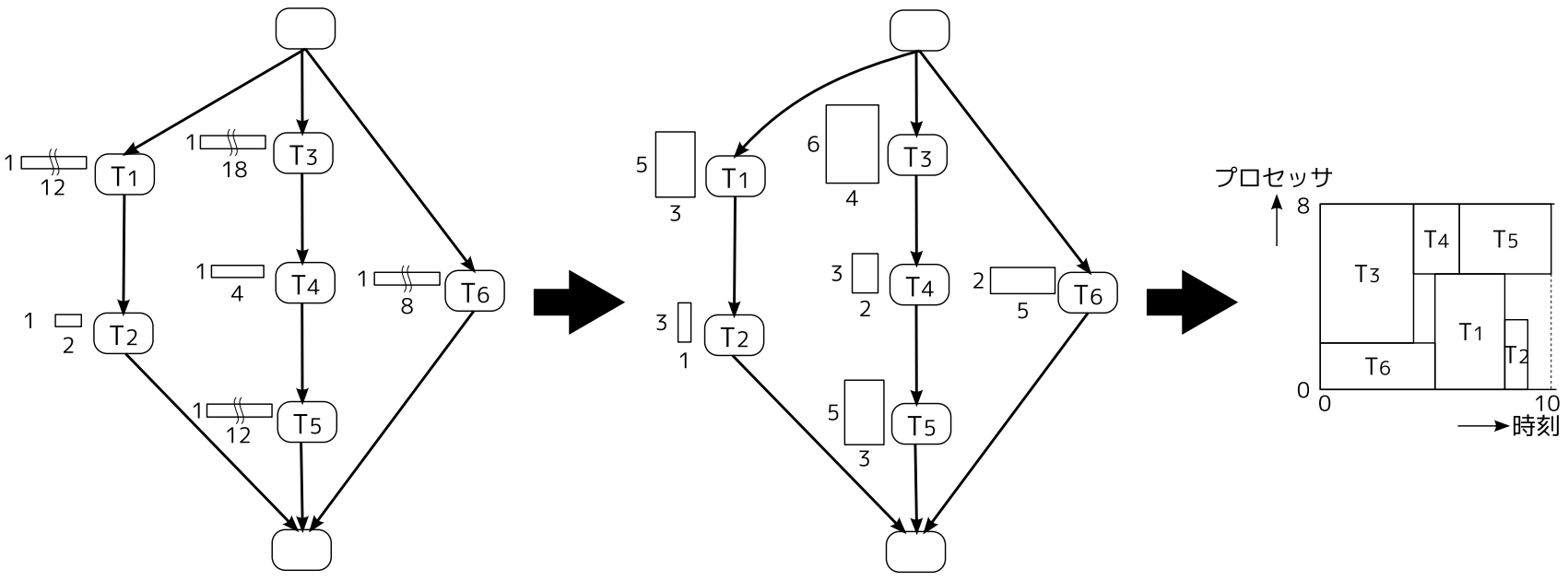
Mixed-Parallel に対するスケジューリング

- **使用プロセッサ数は変更可能で、処理時間は使用プロセッサ数に従う**



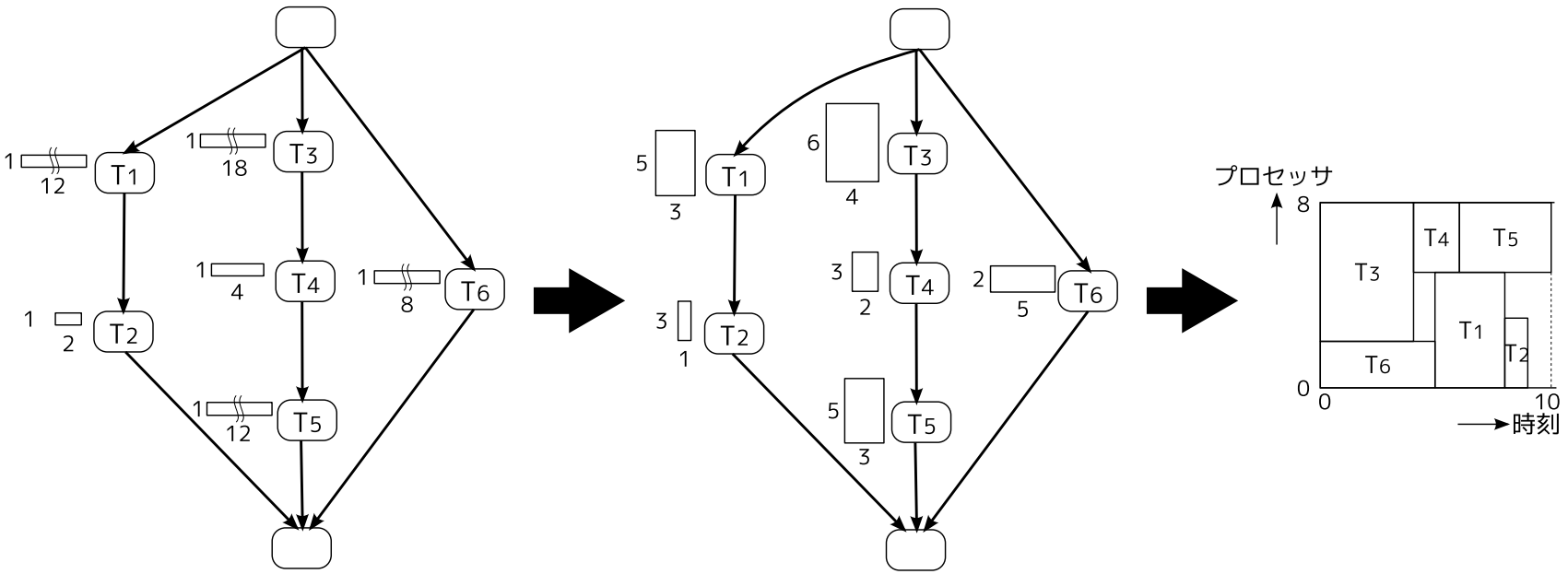
CPA(Critical Path and Area based scheduling)

- CPA: **本論文の手法の基になっている手法**
- **貪欲法で近似最適解を求めている**



CPA(Critical Path and Area based scheduling)

1. 全てのタスクの使用プロセッサ数を 1 に設定する
2. 使用プロセッサ数を 1 増やしたときに処理時間が最も減少するタスクを選び、1 増やす
3. 条件を満たす間、前項をくりかえす
4. リストスケジューリングを行う

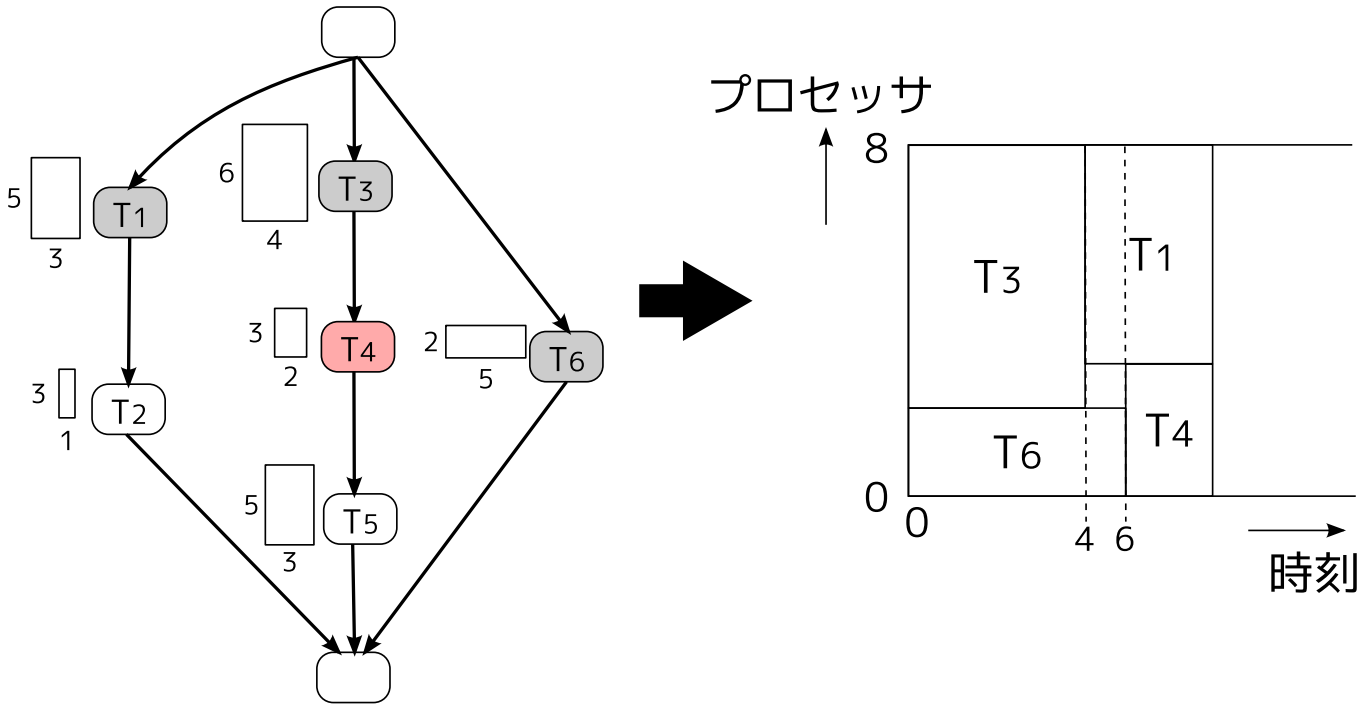


リストスケジューリング

配置可能なタスク (先行タスクが全て配置されているタスク) について

- **全ての先行タスクが完了している時刻以降**
- **使用プロセッサ数を確保できる最小の時刻**

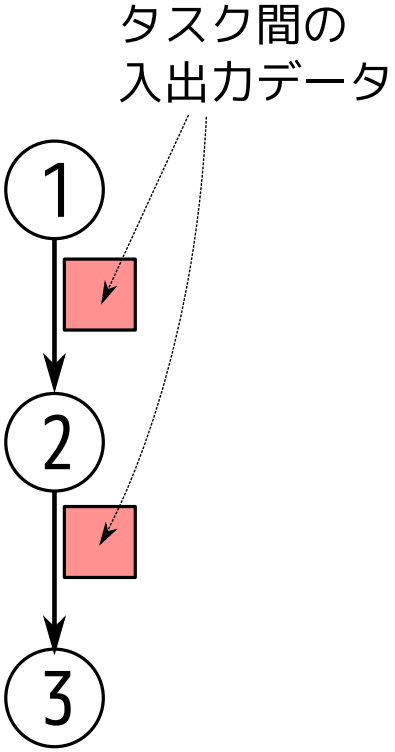
を満たす時刻に配置する

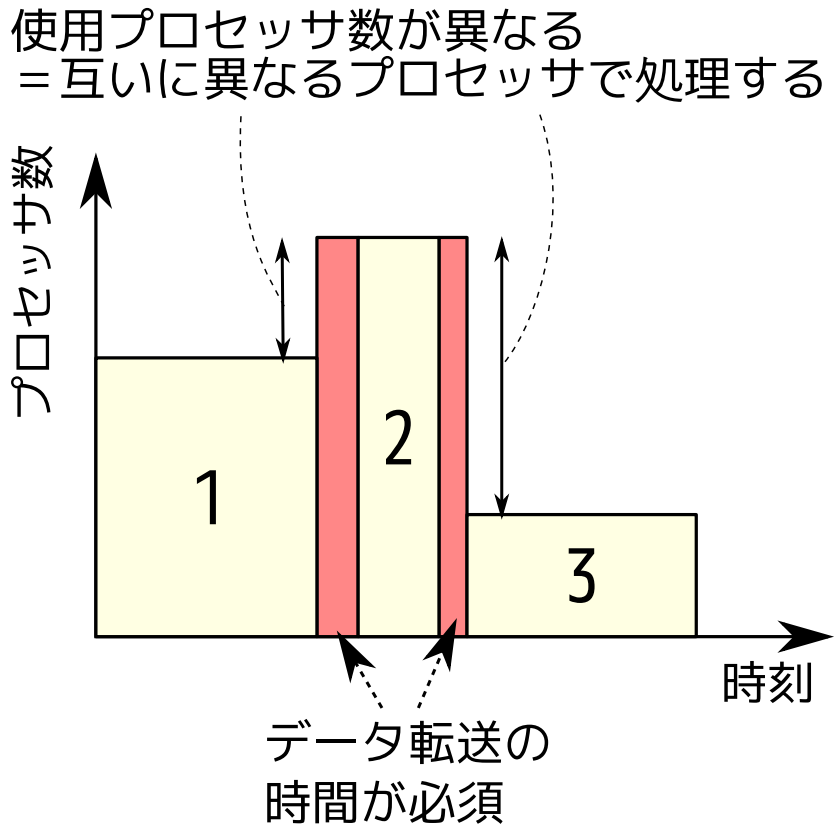


RATS(Redistribution Aware Two-Steps scheduling)

CPA **の問題点**: **タスク間のデータのやりとりを考慮していない**

連続するタスクを互いに異なるプロセッサに割当てる
→ タスクの処理時間と別にデータ転送の時間を要する
→ 実際の処理時間とスケジューリング結果に差異

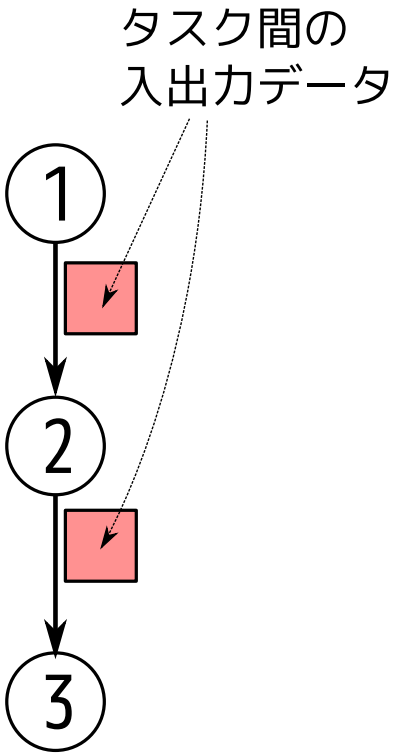


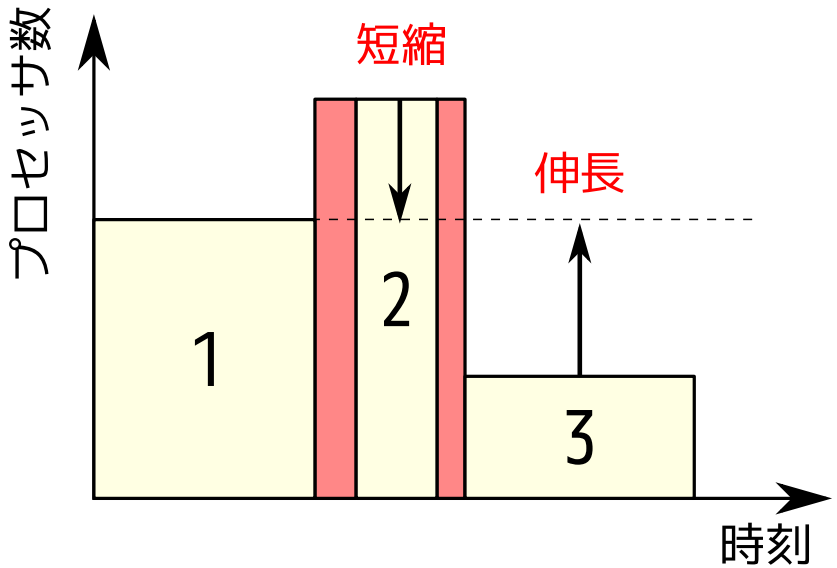


RATS(Redistribution Aware Two-Steps scheduling)

提案手法: RATS

連続するタスクのプロセッサ数を揃える
→ データ転送にかかる時間をゼロとみなす

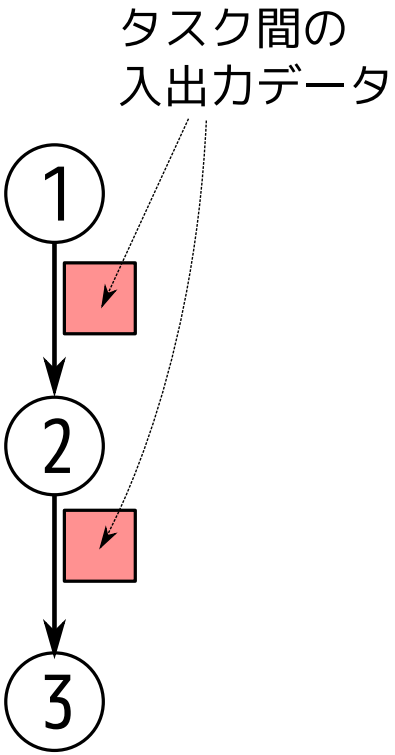


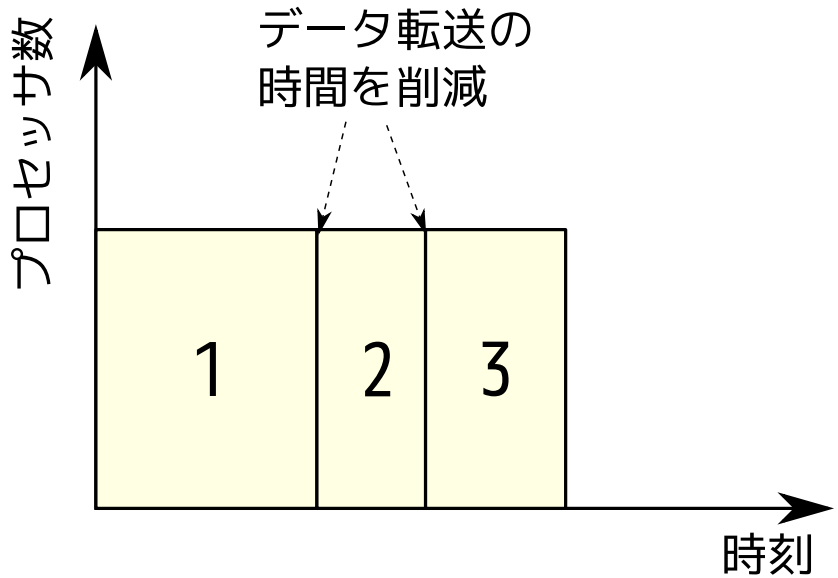


RATS(Redistribution Aware Two-Steps scheduling)

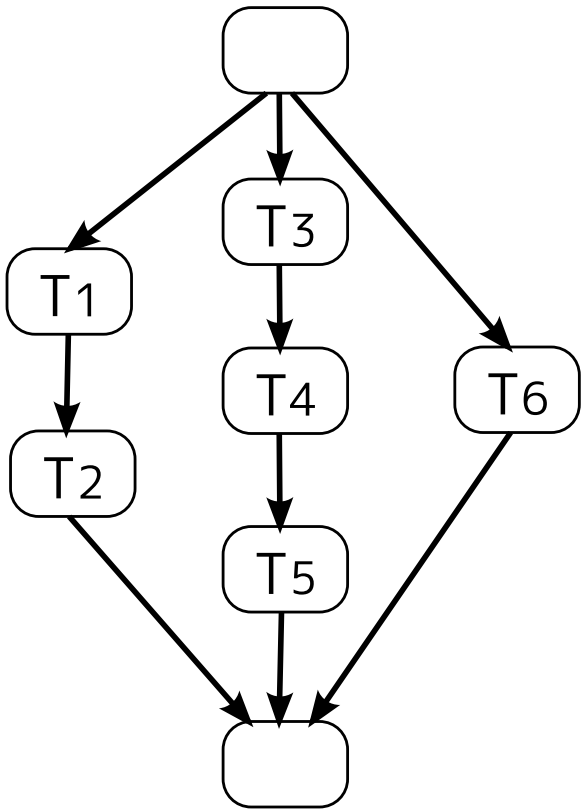
提案手法: RATS

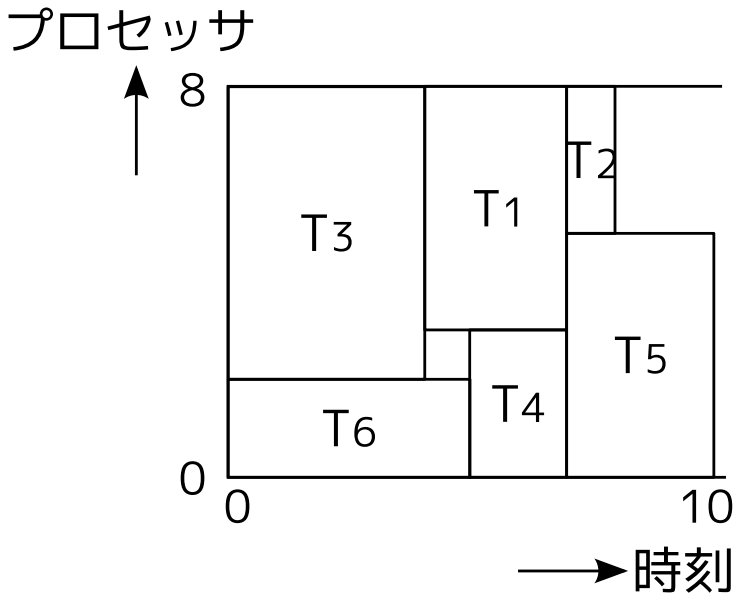
連続するタスクのプロセッサ数を揃える
→ データ転送にかかる時間をゼロとみなす





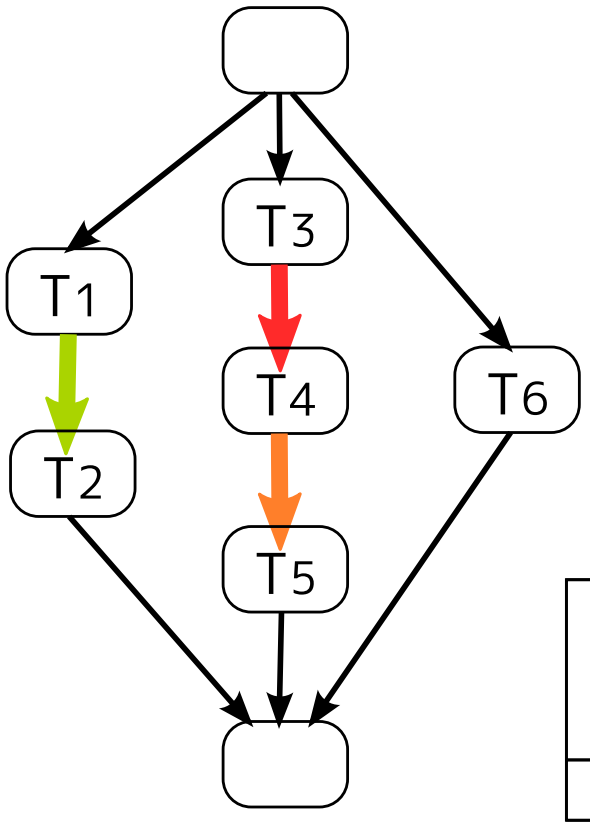
RATS の概要 2/2

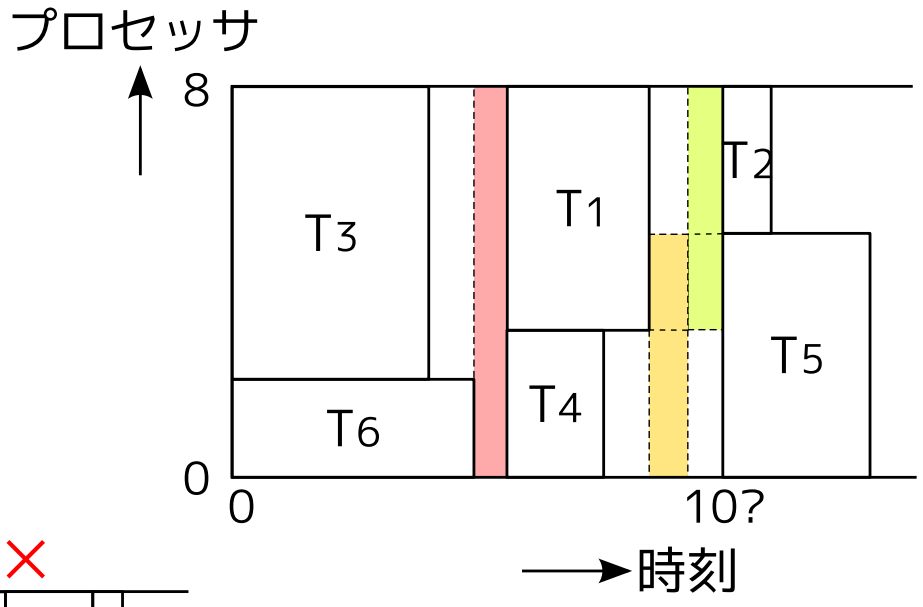


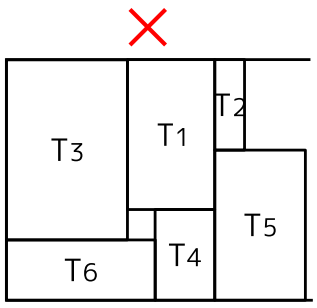


CPAによる
スケジューリング結果

RATS の概要 2/2



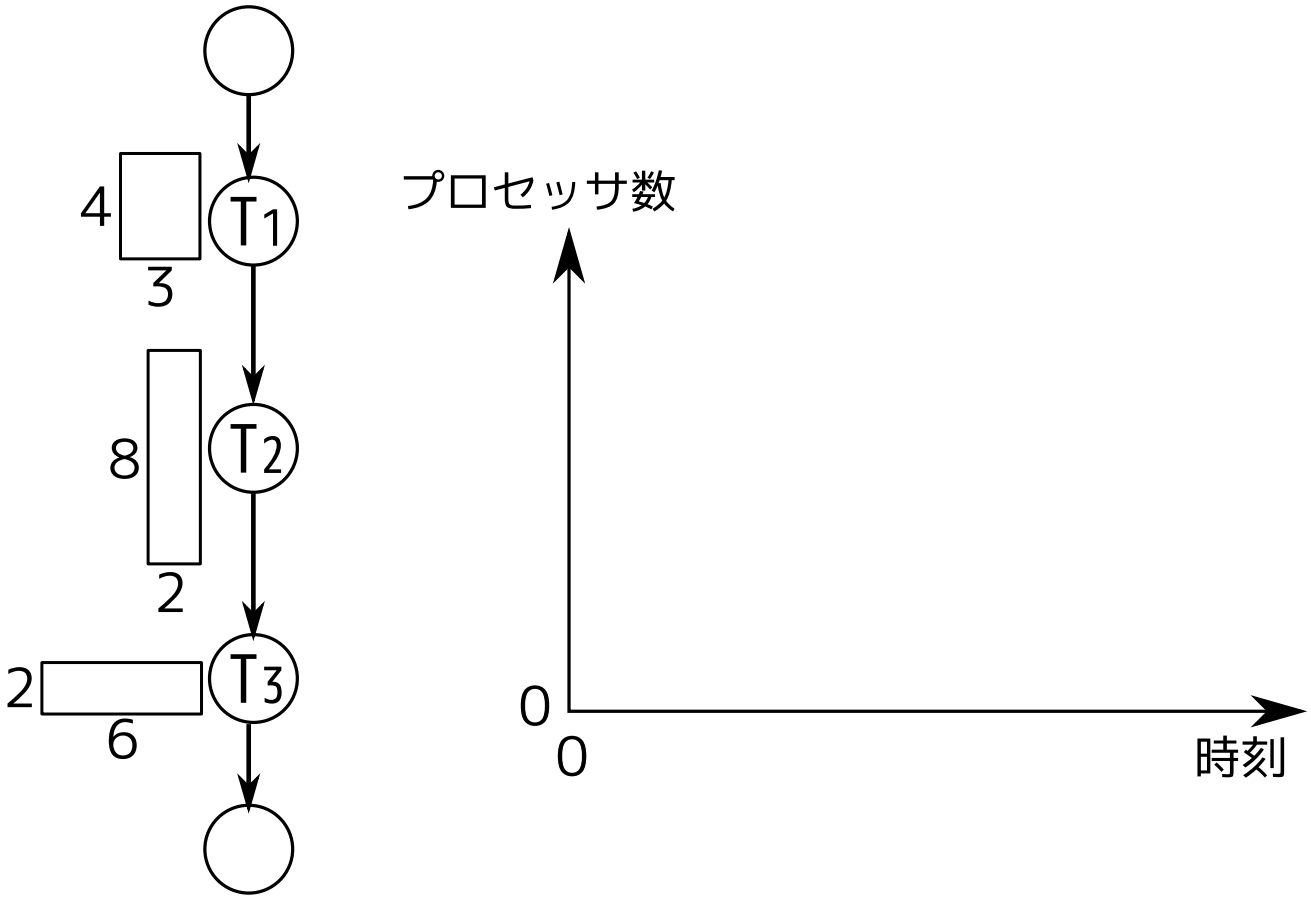




でも実際には…

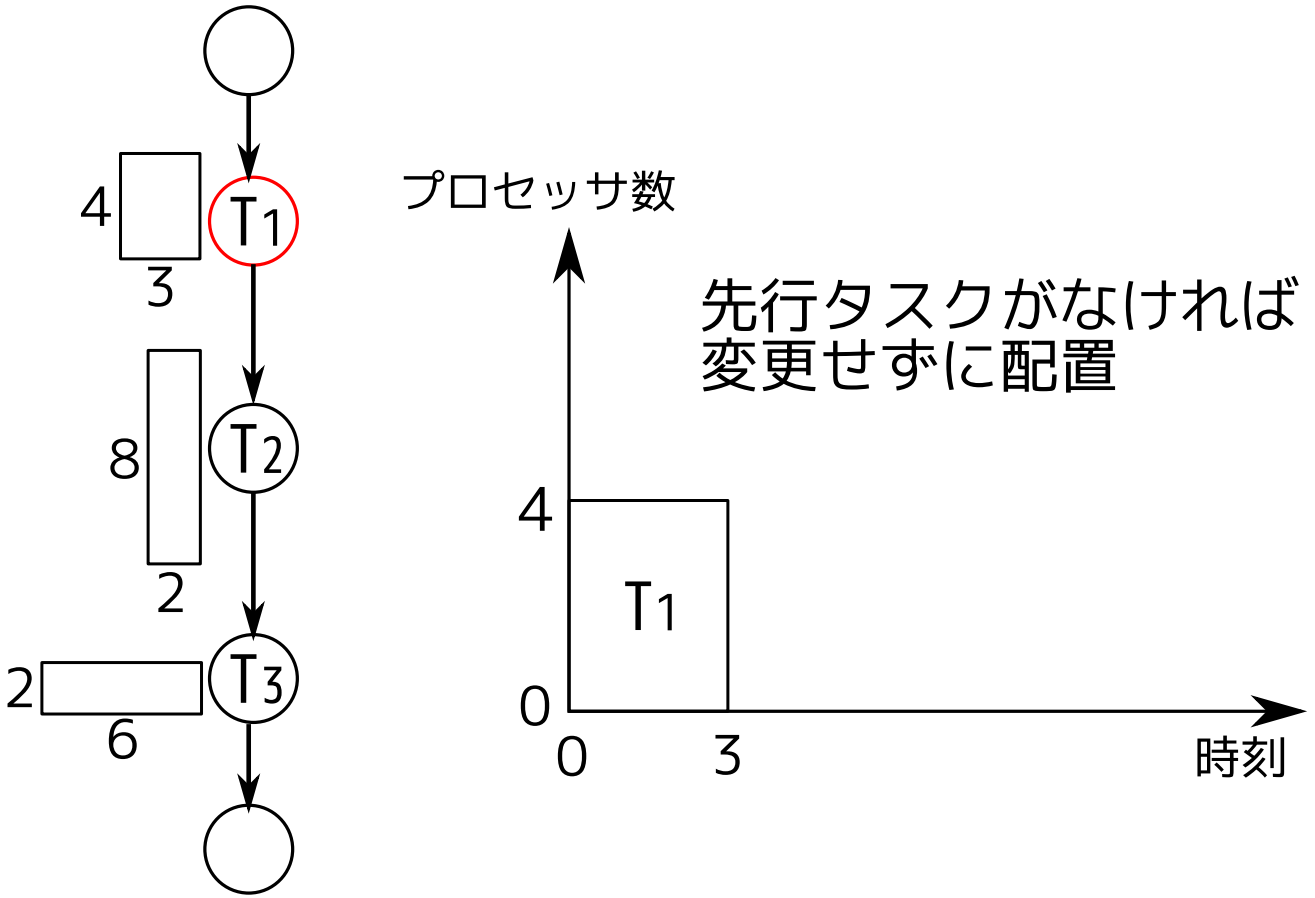
RATS の手順 1/3

1. CPA と同じ方法で各タスクの使用プロセッサ数を決定する
2. リストスケジューリングにおいてタスクを配置するとき、使用プロセッサ数を先行タスクと同じ値に変更する



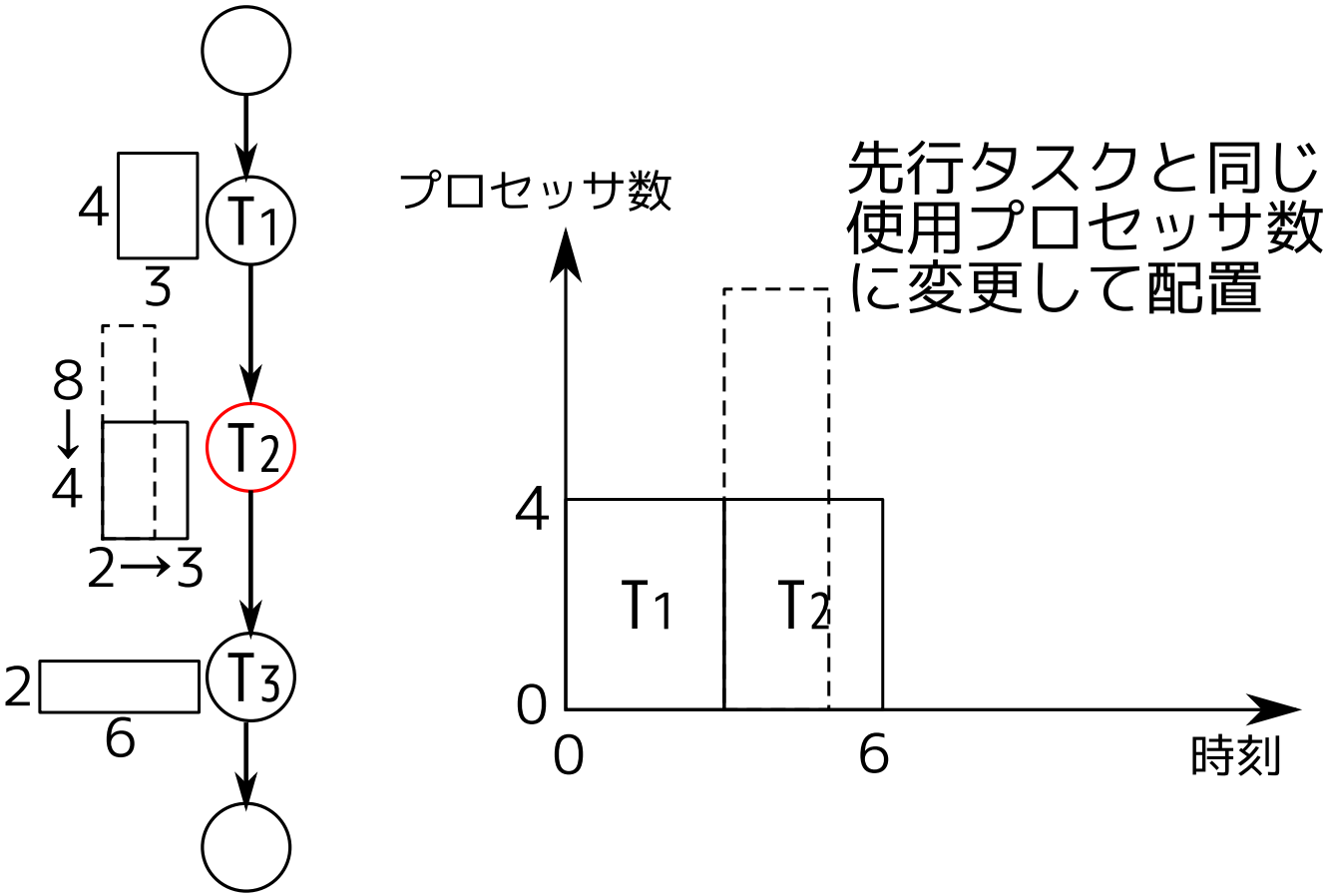
RATS の手順 1/3

1. CPA と同じ方法で各タスクの使用プロセッサ数を決定する
2. リストスケジューリングにおいてタスクを配置するとき、使用プロセッサ数を先行タスクと同じ値に変更する



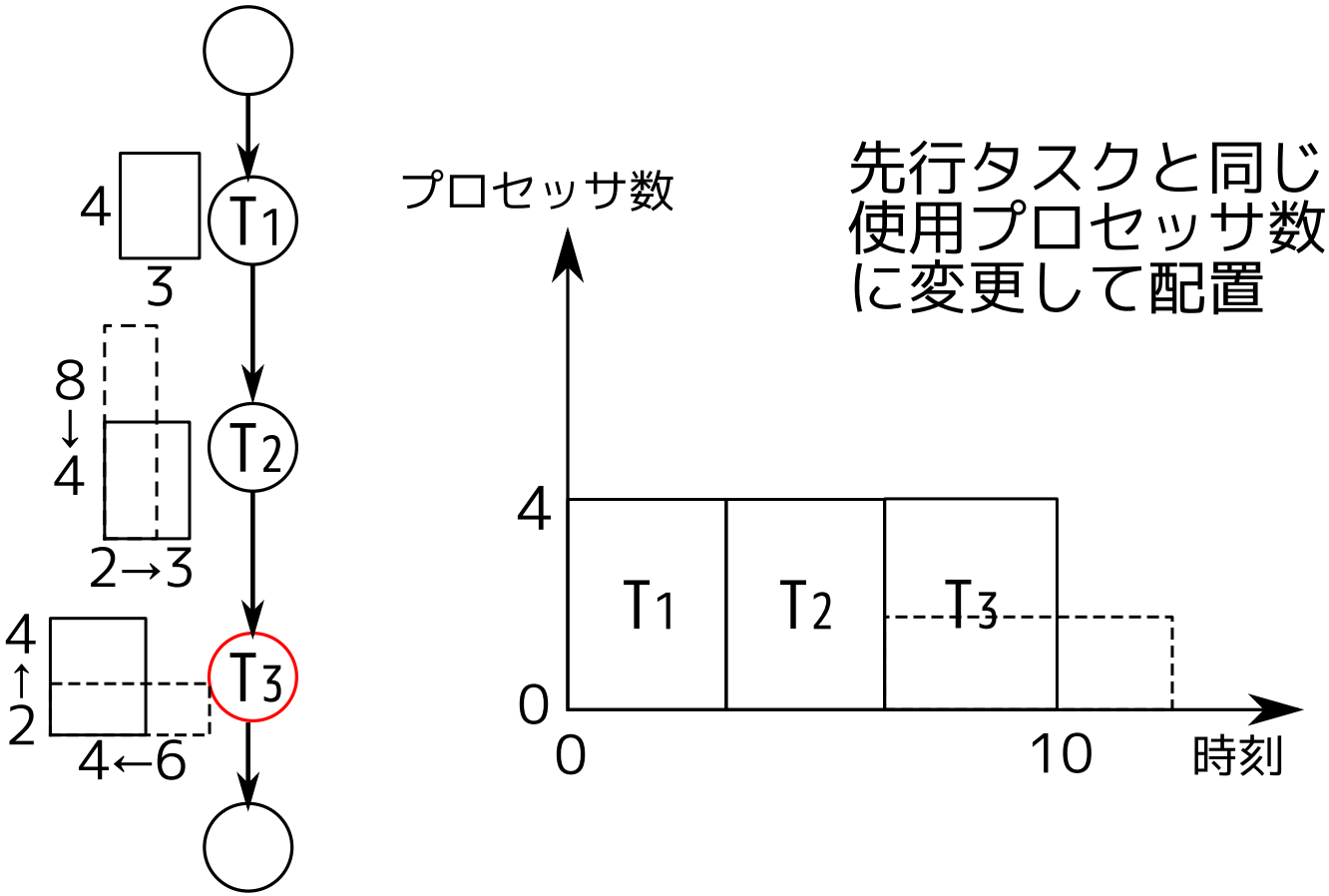
RATS の手順 1/3

1. CPA と同じ方法で各タスクの使用プロセッサ数を決定する
2. リストスケジューリングにおいてタスクを配置するとき、使用プロセッサ数を**先行タスクと同じ値に変更する**



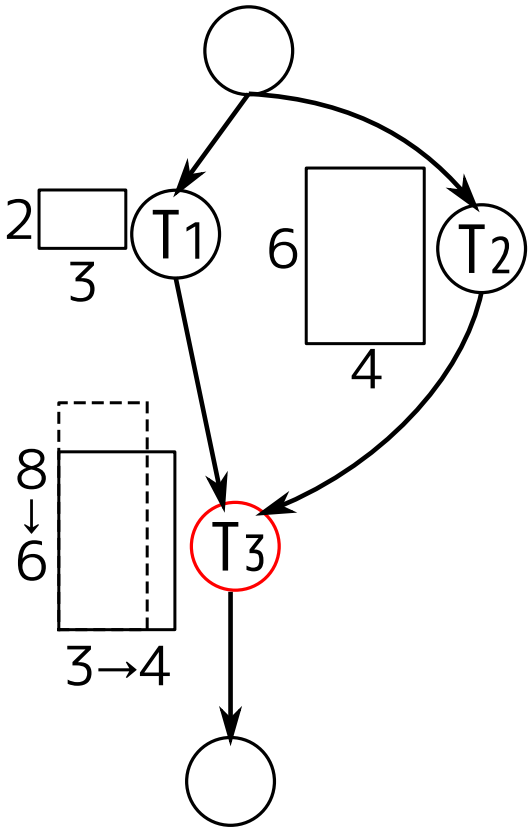
RATS の手順 1/3

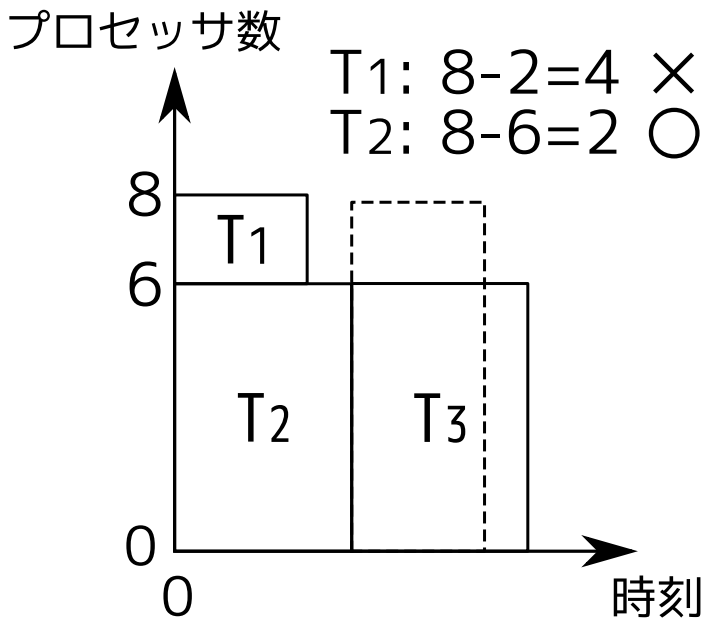
1. CPA と同じ方法で各タスクの使用プロセッサ数を決定する
2. リストスケジューリングにおいてタスクを配置するとき、使用プロセッサ数を**先行タスクと同じ値に変更する**



RATS の手順 2/3

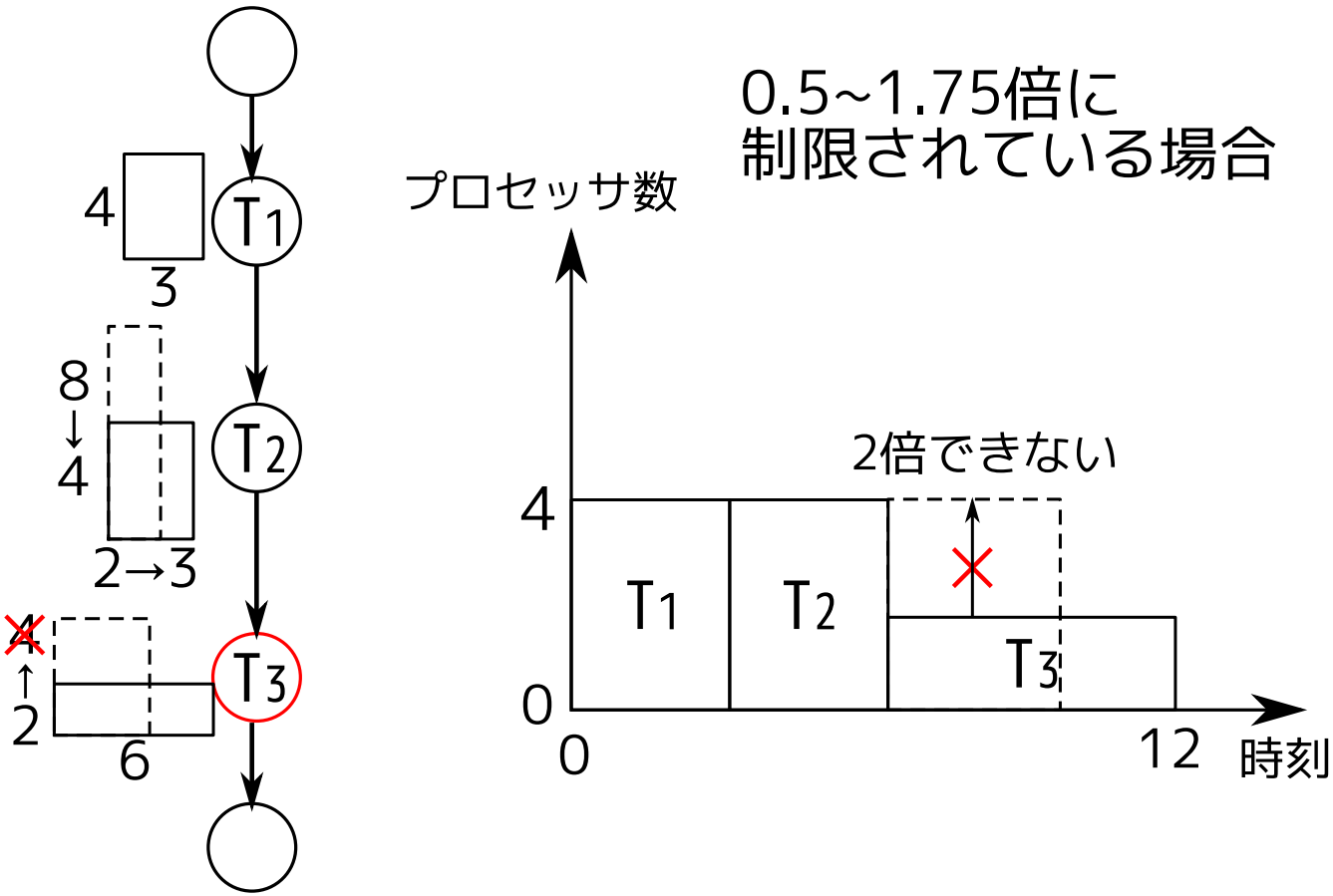
- **先行タスクが複数ある場合、使用プロセッサ数が最も近い先行タスクに合わせる**





RATS の手順 3/3

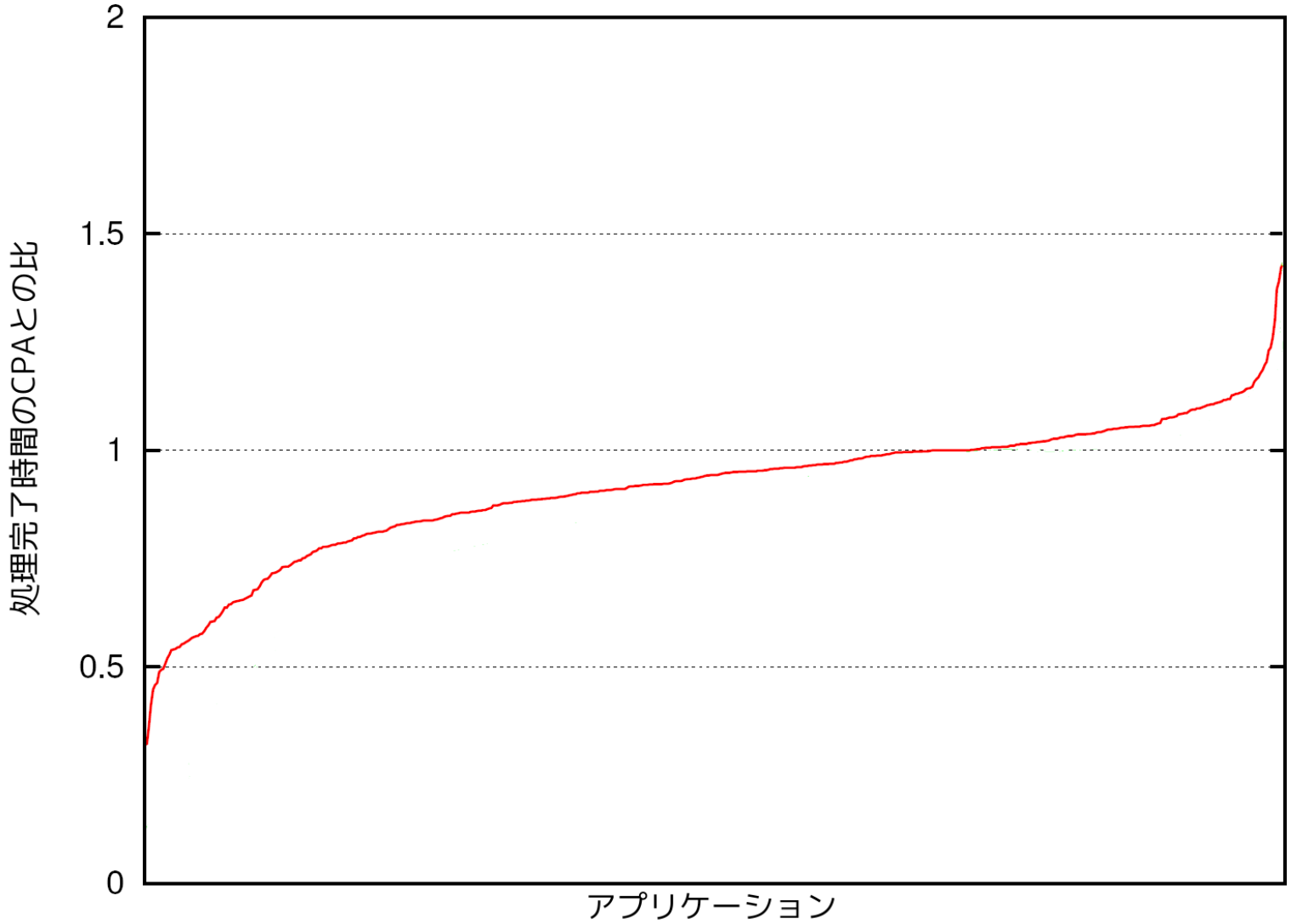
- **伸縮可能な限界を設ける**
- **無制限に伸縮できるようにすると、CPA で定めたプロセッサ数と大きく変わってしまうため**



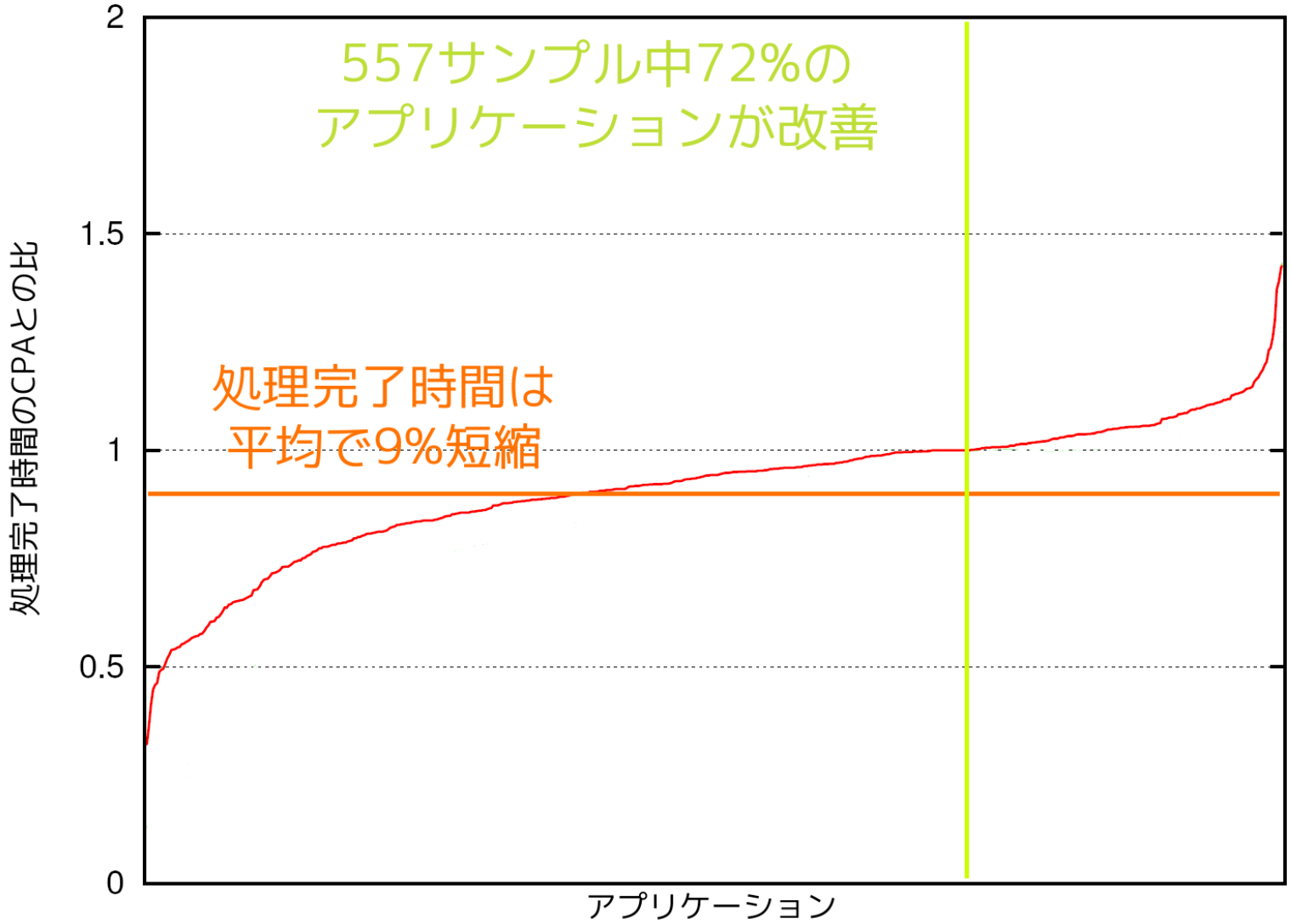
検証

- 処理完了時間を CPA と比較評価
- プラットフォームはプロセッサ数 (ノード数)47 の PC クラスタを使用
- アプリケーションはランダムに作成した 557 サンプルを使用
- 上記のアプリケーションを SimGrid(並列アプリケーションのスケジューリングを評価するシミュレータ) 上で実行
- 伸縮可能な範囲は 0.5〜1.5 倍とする

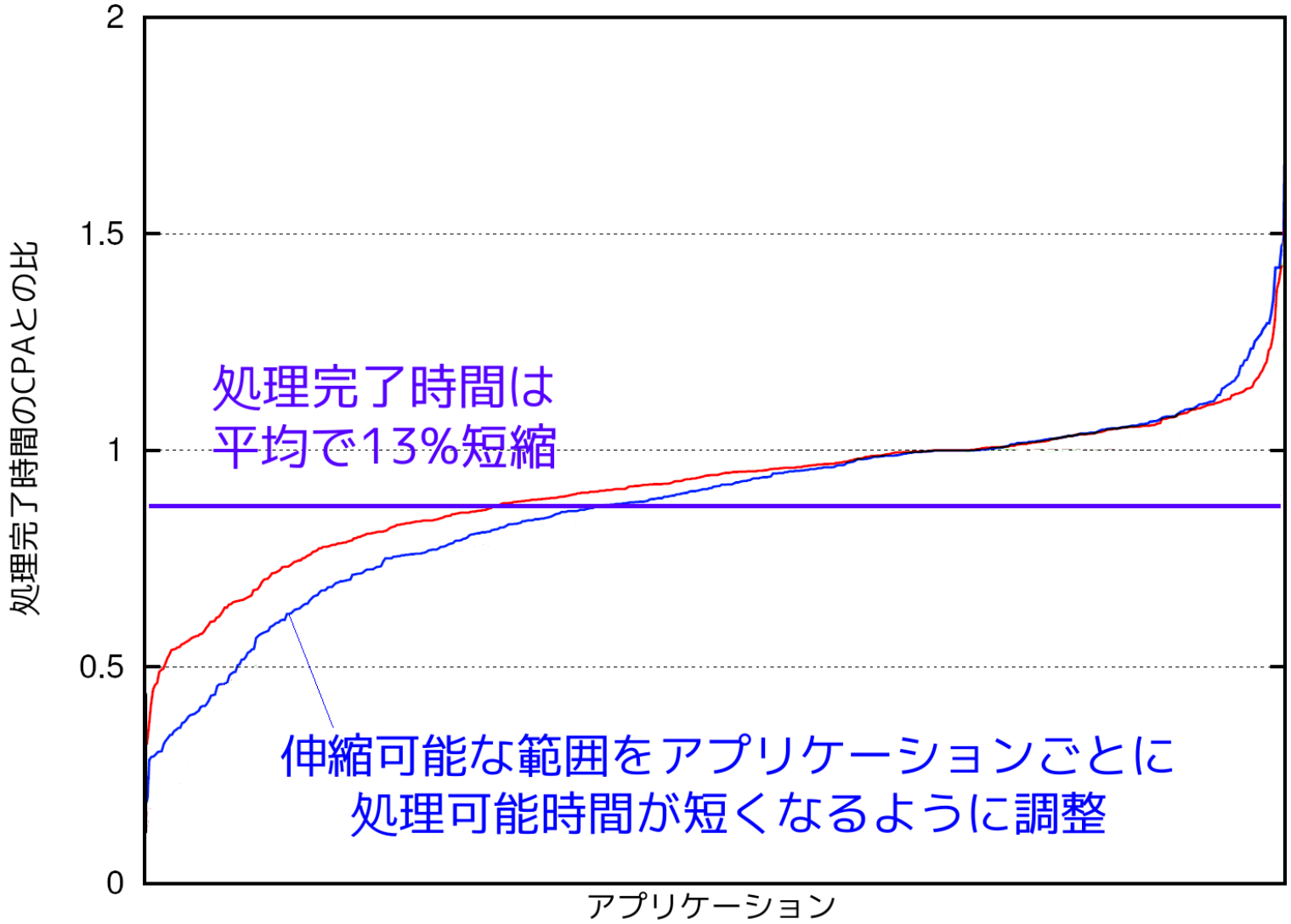
検証



検証


557サンプル中72%の
アプリケーションが改善
処理完了時間は
平均で9%短縮
処理完了時間のCPAとの比
2
1.5
1
0.5
0
アプリケーション

検証

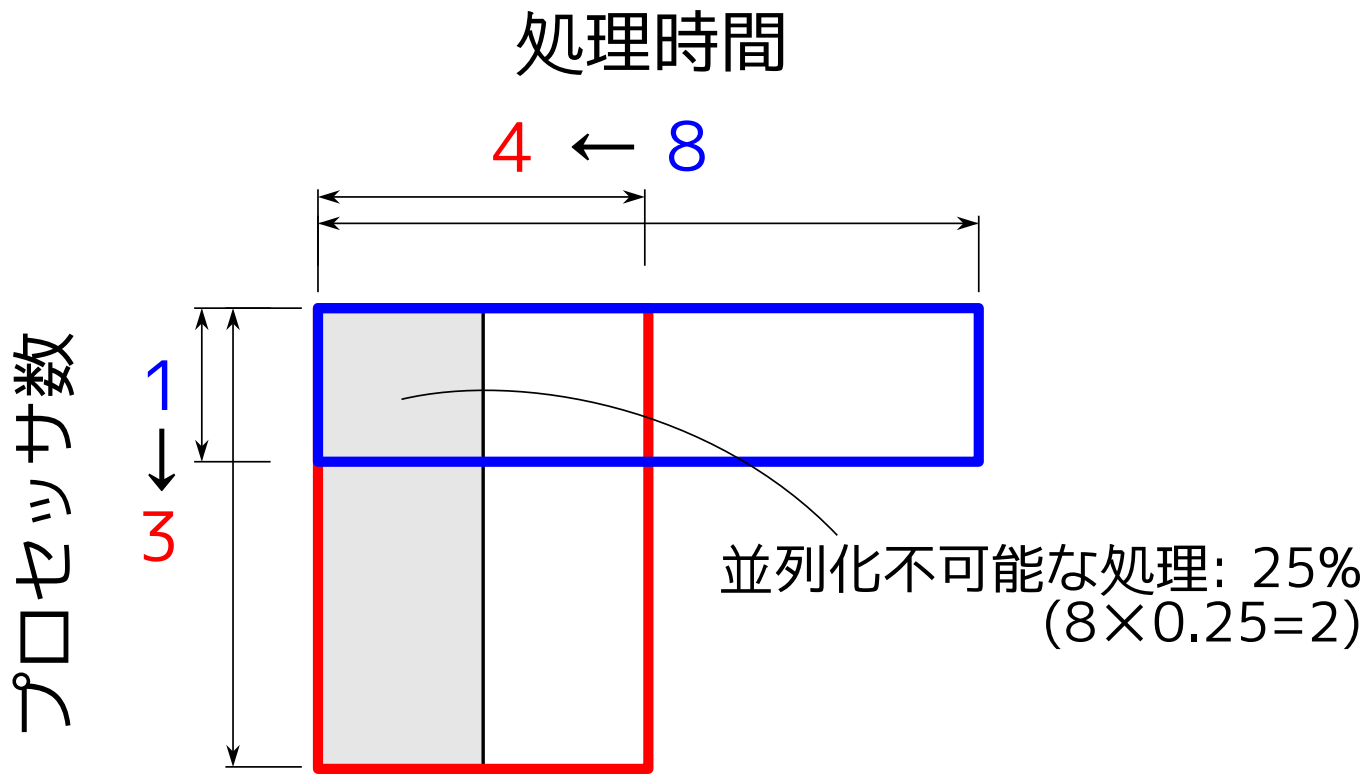

2
1.5
1
0.5
0
処理完了時間のCPAとの比
アプリケーション
処理完了時間は
平均で13%短縮
伸縮可能な範囲をアプリケーションごとに
処理可能時間が短くなるように調整

結論

- Mixed-Parallel アプリケーションに対するスケジューリング手法について、プロセッサ数を再度割り当てるアルゴリズムを提案した。
- 単一クラスタ上のシミュレータを用いて従来手法と性能比較を行ったところ、大部分のアプリケーションでより良い解を得た。
- 今後の課題には、マルチクラスタへ適用できるように拡張することや、性能を左右するパラメータの自動調整機能の付加が挙げられる。

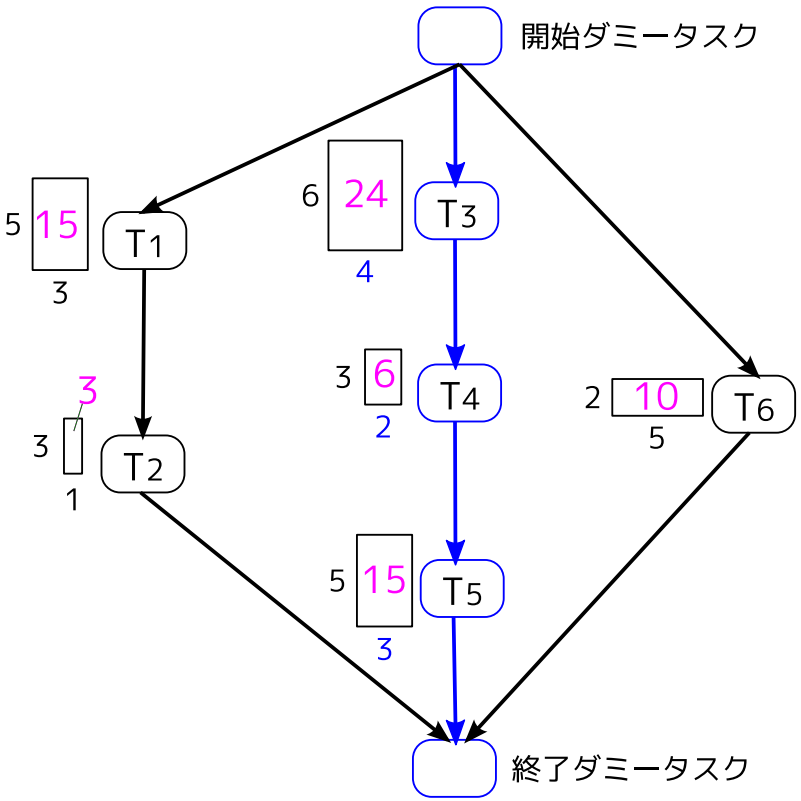
総プロセッサ数と処理時間の関係

- 並列化不可能な部分があるためにタスクの使用プロセッサ数を N 倍にしても処理時間は (1/N) 倍にならない
- 入力として与えられる値は、各タスクの
  - 使用プロセッサ数が 1 のときの処理時間 (図中の 8 に相当)
  - 使用プロセッサ数が 1 のときの並列化不可能な割合 (図の 25%に相当)



CPA のループ終了条件

開始ダミータスクのクリティカルパス長が、1 プロセッサあたりの平均処理時間よりも大きくなったときに終了する



開始ダミータスクのクリティカルパス長

= MAX(3+1, 4+2+3, 5)
= 4+2+3
= 9

1プロセッサあたりの平均処理時間

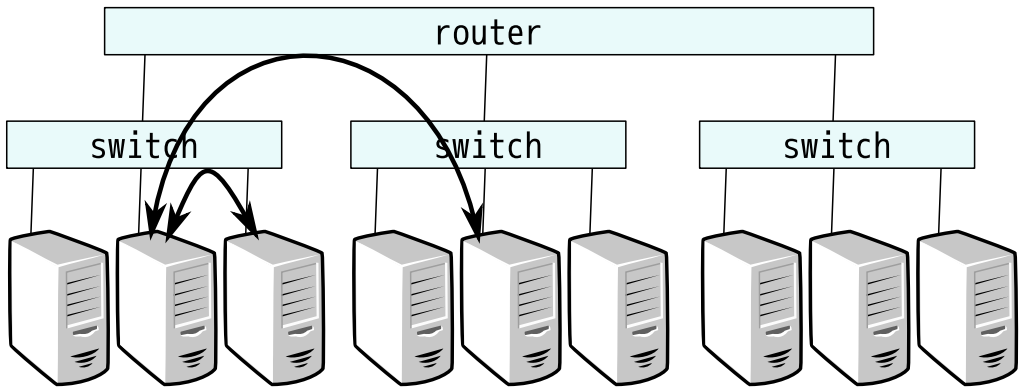
= (15+3+24+6+15+10) / 8
= 9.125

HCPA(CPA on Heterogeneous platforms)

HCPA

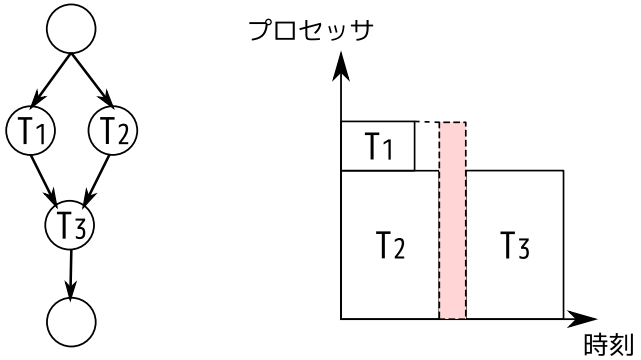
CPA をマルチクラスタのような不均質なネットワークに適用できるように改良した手法

単一クラスタのとき CPA と HCPA は同等のアルゴリズム提案手法は HCPA を改良した手法だが、本論文は単一クラスタの場合のみを考えているので CPA と同等

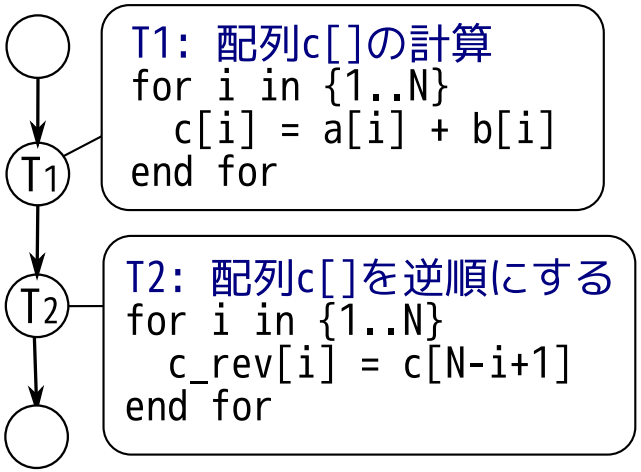


データ転送を無視できない例

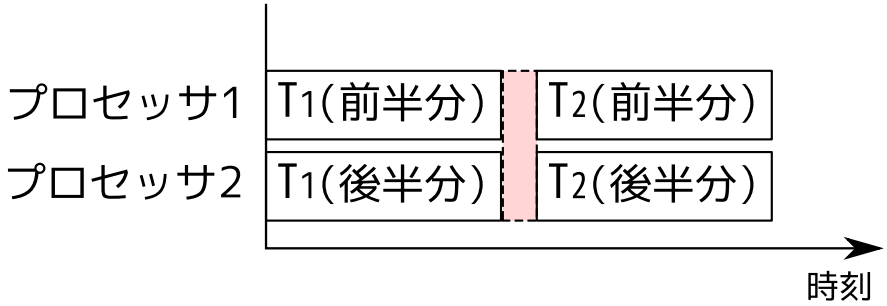
- 先行タスクが複数ある場合



- タスク内のデータ依存関係による場合



2つのプロセッサで以下のように
並列化するとデータ転送が必要



検証で用いたアプリケーション

- FFT **アプリケーション**: 100 **サンプル**
- Strassen **アプリケーション**: 25 **サンプル**
- Layered **型のランダムアプリケーション**: 108 **サンプル**
- Irregular **型のランダムアプリケーション**: 324 **サンプル**

	Layered	Irregular
タスク数	25, 50, 100	25, 50, 100
width	0.2, 0.5, 0.8	0.2, 0.5, 0.8
density	0.2, 0.8	0.2, 0.8
regularity	0.2, 0.8	0.2, 0.8
jump length	-	1, 2, 4
個数	3	3
総サンプル数	108	324