分散システムソフトウェア
11月28日

コンピュータ科学専攻 M1
大住裕之
osumi@is.s.u-tokyo.ac.jp

Distributed Computing – Fundamentals, Simulations and Advanced Topics –

Chapter 4: Mutual Exclusion in Shared Memory

Shared Memory Systems

- n 個のプロセッサ:
  p_0, p_1, …, p_{n-1}
- m 個の共有(shared)レジスタ:
  R_0, R_1, …, R_{m-1}

Register Types

- 各レジスタには type が定義されている
  1. 取得できる値(value)
  2. 適用できる操作(operation)
  3. 各操作の返り値(もしあれば)
  4. 各操作適用後のレジスタの値

例: integer-valued read/write register R

1. 任意の整数値
2. read(R, v), write(R, v)
3. v
4. read(R, v): Rは不変
   write(R, v): Rの値をvで書き換える

Configuration C

- Configuration C を以下のベクトルとして定義

$$C = (q_0, \ldots, q_{n-1}, r_0, \ldots, r_{m-1})$$

q_i: p_i の状態(state)

r_i: R_i の値

- 特にCにおけるメモリの状態を以下で表す

$$mem(C) = (r_0, \ldots, r_{m-1})$$

Events

- あるプロセッサにおいて、アトミックに実行される以下の一連の計算ステップをevent と呼ぶ
  1. プロセッサp_iは現在のp_iの状態に基づいて、アクセスする共有変数と適用する操作を選択する
  2. 操作を実行する
  3. p_iは現在の状態、操作の帰り値、遷移関数によって、状態を変更する

Execution Segment

- 以下の(有限または無限)列をアルゴリズムの execution segment と定義する

$$C_0, \phi_1, C_1, \phi_2, C_2, \phi_3, \ldots$$

C_k: configuration
ϕ_k: event (プロセッサのindexで表される)

  - C_{k-1} で ϕ_k を実行すると C_k になる

Schedule σ

- schedule
  - プロセッサのindexの列(= eventの列)
    - あるexecution segment中のeventの列を表す
    - 例: 3, 1, 4, 2, 1, 4, ...
- exec(C, σ)
  - configuration Cにschedule σを順に適用したexecution segment (一意に定まる)

「到達可能」

- 「C′はCから到達可能(reachable)」
  - Configuration C, C′に対して、C′ ＝ σ(C)を満たす有限な schedule σが存在すること

Definition 4.1: Configuration の Similarity

- 「configuration C が C′ と、プロセッサの集合 P について similar である」 ($C \sim^{P} C'$ と表す)
  - Pの各プロセッサの状態が C と C′ で等しく、かつ mem(C) = mem(C′) である

Configurationのsimilarity

- 例
  - $P = \{p_1, p_3, p_4\}, n = 5, m = 4$ とする
  - 以下の C, C′ は $C \sim^P C'$ を満たす
    - $C = \{q_0, q_1, q_2, q_3, q_4, r_0, r_1, r_2, r_3\}$
    - $C' = \{q_0', q_1, q_2', q_3, q_4, r_0, r_1, r_2, r_3\}$

- $C \sim^P C'$ のとき、P に含まれるプロセッサから見て C と C′ は同一(違いを判別できない)

Mutual Exclusion 問題

- Mutual exclusion
  - あるコード領域(critical sectionと呼ばれる)に、高々1つのプロセッサだけが入っているように制御すること
- No deadlock
  - 1 つ以上のプロセッサが critical section に入ろうとするとき、永久に critical section にとどまるプロセッサが存在しない限り、どれか１つはそのsectionに入ることができること

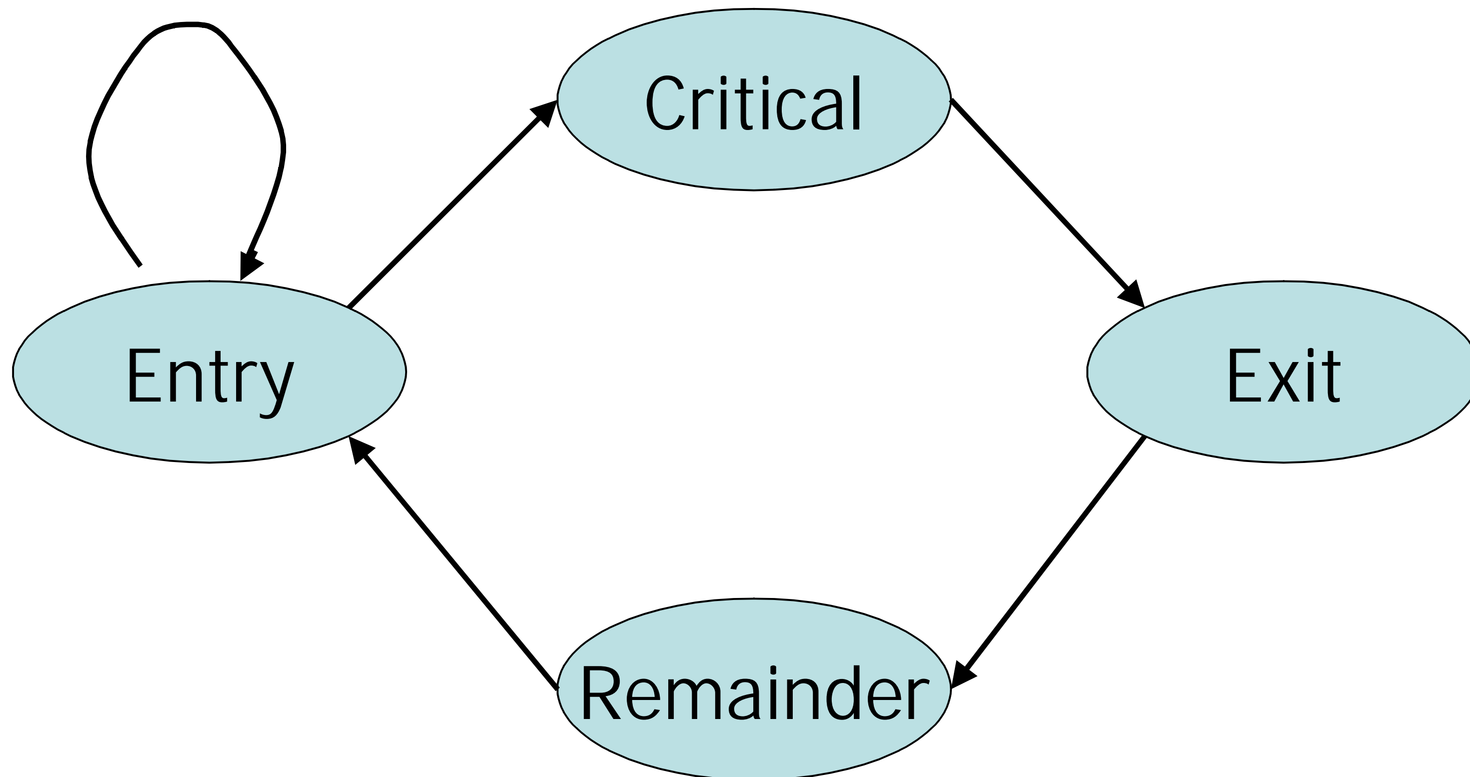
Mutual Exclusion 問題

- No lockout (no starvation)
  - あるプロセッサが critical section に入ろうとするならば、永久にcritical sectionにとどまるプロセッサが存在しない限り、いつかは入ることができるということ
  - no lockout であるとき no deadlock も満たす

プログラムコード領域の区分け

- 各プロセッサはcritical sectionの前後で追加のコードを実行するものとする
- コード領域を以下の4つに分類する
  - Entry (trying)
    - Critical sectionに入る準備のコード領域
  - Critical
    - 複数プロセッサが同時実行しないよう保護されたコード領域
  - Exit
    - Critical sectionから出る際に実行されるコード領域
  - Remainder
    - 上記以外のコード領域

プロセッサの実行サイクル



もうすこし形式的な定義

- 「実行がadmissibleである」
  - 各プロセッサが
    - 無限の長さの実行ステップを実行する
    - (有限の実行ステップで)Remainder sectionで終了する

    のいずれかである

もうすこし形式的な定義

- Mutual exclusion
  - あらゆる実行のあらゆる configuration において、critical section には高々1つのプロセッサが入っていること
- No deadlock
  - あらゆる admissible な実行において、ある configuration で entry section にいるプロセッサがあるならば、それより後の configuration で、プロセッサが critical section に入っているものが存在すること

もうすこし形式的な定義

- No lockout
  - あらゆる admissible な実行において、ある configuration で entry section にいるプロセッサがあるならば、それより後の configurationで、そのプロセッサが critical section に入っているものが存在すること

Mutual Exclusion using Powerful Primitives

- 以下のプリミティブを用いてmutual exclusion問題を解く
  - test&set
  - read-modify-write
- 得られる結論
  - 1 bit あれば mutual exclusion と no deadlock が保証される
  - より強い公平性を実現するためには、Θ(log n) bits 必要となる

Binary Test&Set Registers

- 2つのアトミックな操作が可能な変数 V
  - test&set(V: memory address) returns binary value :

    temp := V
    V := 1
    return (*temp*)

  - reset(V: memory address):

    V := 0

Algorithm 7: Mutual Exclusion using a test&set Register

Initially V equals 0

〈Entry〉:

1: wait until test&set(V) = 0

〈Critical Section〉

〈Exit〉:

2: reset(V)

〈Remainder〉

Theorem 4.1

- Algorithm 7 は、１つの test&set レジスタを用いて、mutual exclusionとno deadlockを実現している

Theorem 4.1の証明

- mutual exclusion
  - 背理法の仮定: 2つのプロセッサ p_i と p_j が同時に critical section に入っている
    - p_i が critical section にいる状態で、p_jがそのcritical sectionに入ったときとする
    - この状況が起こったのは、この実行時に初めてだとする

Theorem 4.1の証明

- コードより、p_i は critical section に入るとき、V = 0 を確かめ、V を 1 にセットする
- コードより、V は critical section から出るプロセッサがあるまで 0 である
- 仮定より、p_i以外のプロセッサはp_jが入るまで critical section にいない
  - 今回初めてmutual exclusionに失敗したため

Theorem 4.1の証明

- よって、p_i が V を 1 にしてから p_j が critical section に入るまでの間に critical section を出たプロセッサは存在せず、V は 1 のままである
- よって、p_j が critical section に入るとき、V をテストした結果は 1 となる
- したがって、p_j は critical section に入ることができなくなり、仮定と矛盾する

Theorem 4.1の証明

- no deadlock
  - 背理法の仮定: 以下のような admissible な実行が存在する
    - ある実行ポイント以降で、少なくとも 1 つのプロセッサは entry section にいるが、critical section に入るプロセッサはない

Theorem 4.1の証明

- admissible な実行であることから、以下のような実行ポイント h が存在する
  - h 以降、少なくとも 1 つのプロセッサが entry section にいて、critical section に入っているプロセッサはない

（admissible な実行では、永久に critical section にいるプロセッサは存在しない）

Theorem 4.1の証明

- V = 0 であるのは critical section に入っているプロセッサが存在しないとき、かつ、そのときだけである
- よって、あるプロセッサが実行ポイント h 以降で V を test すると、V = 0 を得て critical section に進むことになり、仮定に矛盾する

[証明終]

Read-Modify-Write Registers

- read-modify-write と呼ばれる操作が可能な共有変数 V
  - rmw(V: memory address, f: function) returns value:

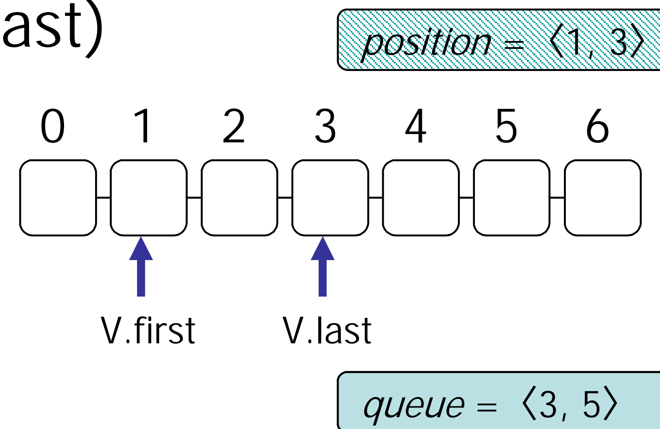
    temp := V
    V := f(V)
    return (*temp*)

- 任意のVの値で f(V) ＝ 1 なのが test&set

Algorithm 8: Mutual Exclusion using a read-modify-write Register

Initially V = 〈0, 0〉

〈Entry〉 :
 1: *position* := rmw(V, (V.first, V.last + 1))
 2: repeat
 3:     *queue* := rmw(V, V)
 4: until (*queue*.first = *position*.last)
〈Critical Section〉
〈Exit〉 :
 5: rmw(V, (V.first + 1, V.last))
〈Remainder〉



Theorem 4.2

- $2\lceil \log_2 n \rceil$bitsのread-modify-writeレジスタを1つ用いて、no lockout および no deadlock を満たす mutual exclusion アルゴリズムが存在する

Theorem 4.2の証明

- Mutual exclusion
  - キューの head のプロセッサだけが critical section に入ることができる
  - そのプロセッサは、critical section から出るまでキューの head である
  - したがって、critical section には高々1つのプロセッサしか入ることができず、mutual exclusion を実現している

Theorem 4.2の証明

- No lockout
  - 全てのプロセッサは必ず critical section から出ると仮定する
  - キューにおける、critical section に入るプロセッサの管理は FIFO である
  - したがって、キューに入れられたプロセッサはいつか必ず critical section に入ることが保証され、no lockout を満たす (no deadlock も満たされることになる)

Theorem 4.2の証明

- read-modify-writeレジスタは$2\lceil \log_2 n \rceil$bits
  - n個以上のプロセッサが、同時にキューに入っていることはない
  - よって、V.firstとV.lastの最大値は n-1 である
  - したがって、Vは最大で $2\lceil \log_2 n \rceil$bits 必要である

[証明終]

Algorithm 8の問題点

- spinning
  - 順番待ちをしている複数のプロセッサが、同じ共有変数Vを繰り返しreadしている
- 共有メモリアーキテクチャでは、これは共有メモリアクセスに要する時間が増加する可能性
  - 各プロセッサは共有メモリのローカルなキャッシュを持っているため

Algorithm 9: Mutual Exclusion using Local Spinning

Initially *Last* = 0;

Flags [0] = *has-lock*; *Flags* [i] = *must-wait*, $0 < i < n$.

〈Entry〉:

1: *my-place* := rmw(*Last*, (*Last* + 1) mod n)

2: wait until (*Flags* [*my-place*] = *has-lock*)

3: *Flags* [*my-place*] := *must-wait*

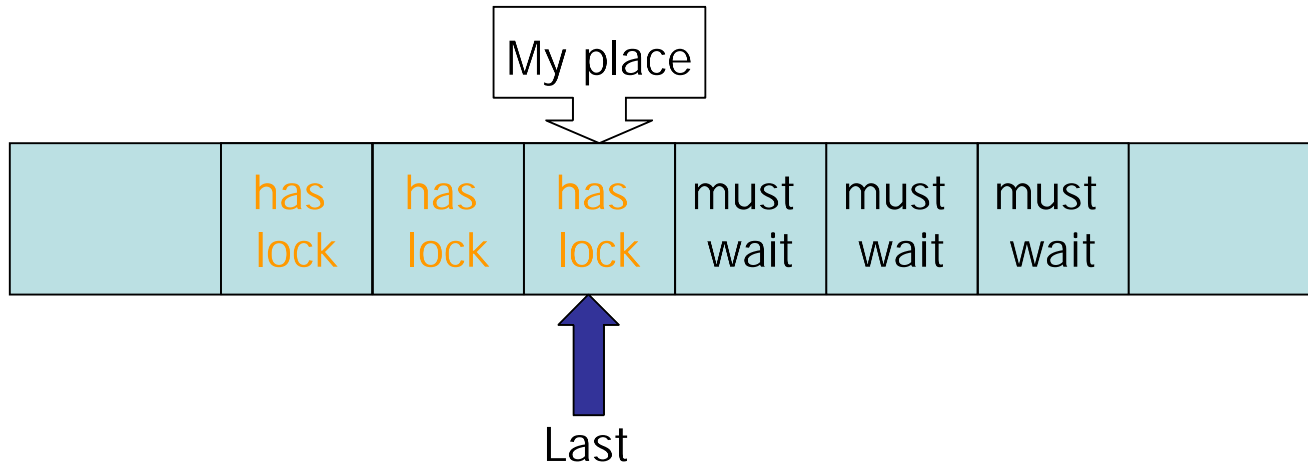
〈Critical Section〉

〈Exit〉:

4: *Flags* [(*my-place* + 1) mod n] := *has-lock*

〈Remainder〉

Algorithm 9



Lemma 4.3

- Algorithm 9は、配列*Flags* に関する以下の不変性を保つ
  1. 高々1つの要素が *has-lock* になる
  2. *has-lock* になっている要素がないならば、critical section内にいるプロセッサがある
  3. *Flags* [*k*] が *has-lock* になっているならば、ちょうど (*k* – *Last* – 1) mod *n* 個のプロセッサがentry sectionにあり、それぞれ異なる*Flags* の要素についてspinningしている

メモリ状態数のLower Bound

- 2値の test&setレジスタ
  - Entry section で starvation が起こりうる
- $2\lceil \log_2 n \rceil$ bits の read-modify-writeレジスタ
  - No lockout を実現している

メモリ状態数のLower Bound

- 実は、lockoutを防ぐためには少なくとも、$\sqrt{n}$ 個の相異なるメモリ状態が必要
- ここでは以下(Theorem 4.4)を示す
  - あるcritical sectionに入ろうとしているプロセッサが、後からきた他のプロセッサに無制限に追い越されること(lockout)を防ぐには、少なくともn個の相異なるメモリ状態が必要

Definition 4.2: *k*-bounded waiting

- 「Mutual exclusion アルゴリズムが *k*-bounded waiting を備えている」
  - あらゆる実行において、他のプロセッサが entry section で待っている一方で、*k* 回よりも多く critical section に入るプロセッサが存在しないこと

Theorem 4.4

- あるアルゴリズムが no deadlock と *k*-bounded waiting を満たして mutual exclusion 問題を解くならば、そのアルゴリズムは少なくともn個の相異なる共有メモリの状態を用いている

Definition 4.3: Quiescence

- 「mutual exclusion アルゴリズムにおける configuration が quiescent である」
  - 全てのプロセッサが remainder section にいる

Theorem 4.4の証明

C: 初期 configuration (quiescentである)

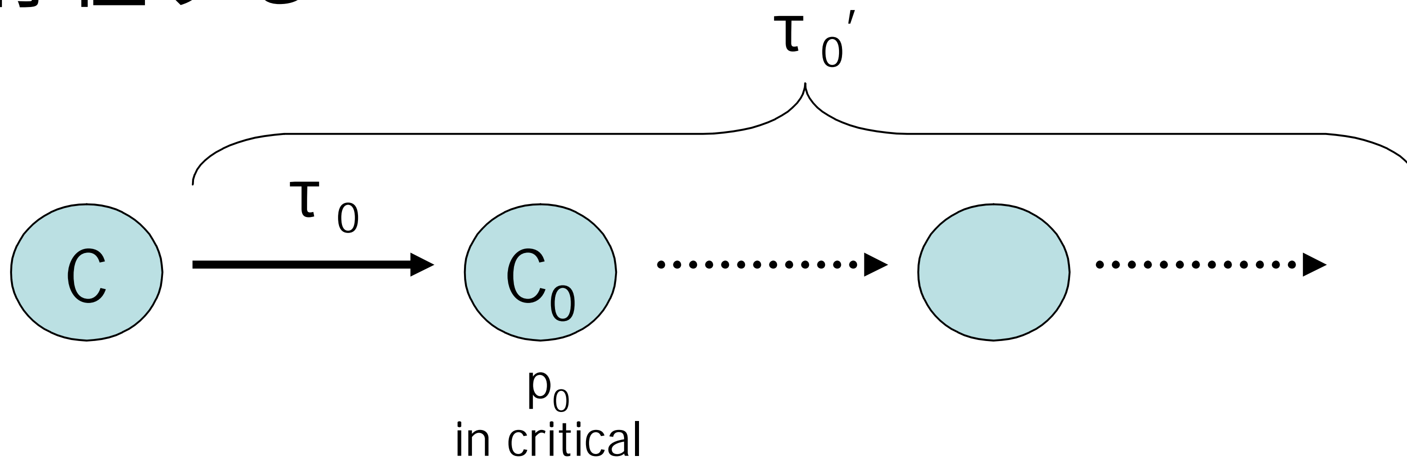
τ_0': 無限 p_0-only schedule

とする

- exec(C, τ_0') は admissibleである
  - p_0は無限の実行ステップ
  - 残りのプロセッサは remainder section で何もしない

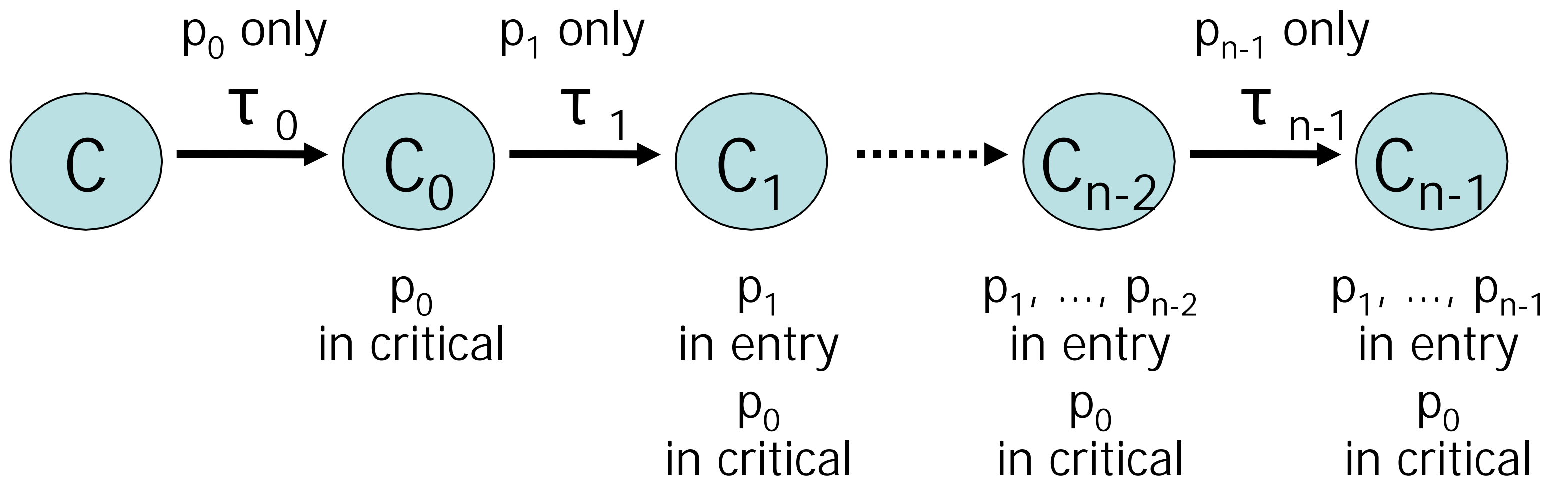
Theorem 4.4の証明

- no deadlock の性質から、
  p_0 が C_0 ＝ τ_0(C) で critical section にいるような τ_0' の有限なプレフィックス τ_0が存在する



Theorem 4.4の証明

- τ_i $(1 \leq i \leq n - 1)$を以下のようにつくる
  - p_i が $C_i = \tau_i(C_{i-1})$ で entry section にいるような p_i-only の schedule τ_i



Theorem 4.4の証明

- 背理法の仮定
  - 共有メモリの異なる状態数は n より小さい

- 仮定より、同一のメモリ状態、つまり

$$mem(C_i) = mem(C_j)$$

  である configuration C_i, C_j $(0 \leq i < j \leq n-1)$ が存在する

Theorem 4.4の証明

- さらに、$p_0, \ldots, p_i$ は $\tau_{i+1}, \ldots, \tau_j$ では何も実行しないので
  $$C_i \sim^P C_j \quad (P = \{p_0, \ldots, p_i\})$$
  が成り立つ
- したがって、C_i と C_j で、
  - p_0: in critical section
  - $p_0, \ldots, p_i$: in entry section

  となっている

Theorem 4.4の証明

- 以下の無限 schedule ρ' を C_i に適用する
  - $p_0, \ldots, p_i$: 無限の計算ステップを実行
  - 残り: 何もしない (0個のステップを実行)

- すると、exec(C, $\tau_0 \tau_1 \ldots \tau_i \rho'$) は admissible

ここまでで C_i

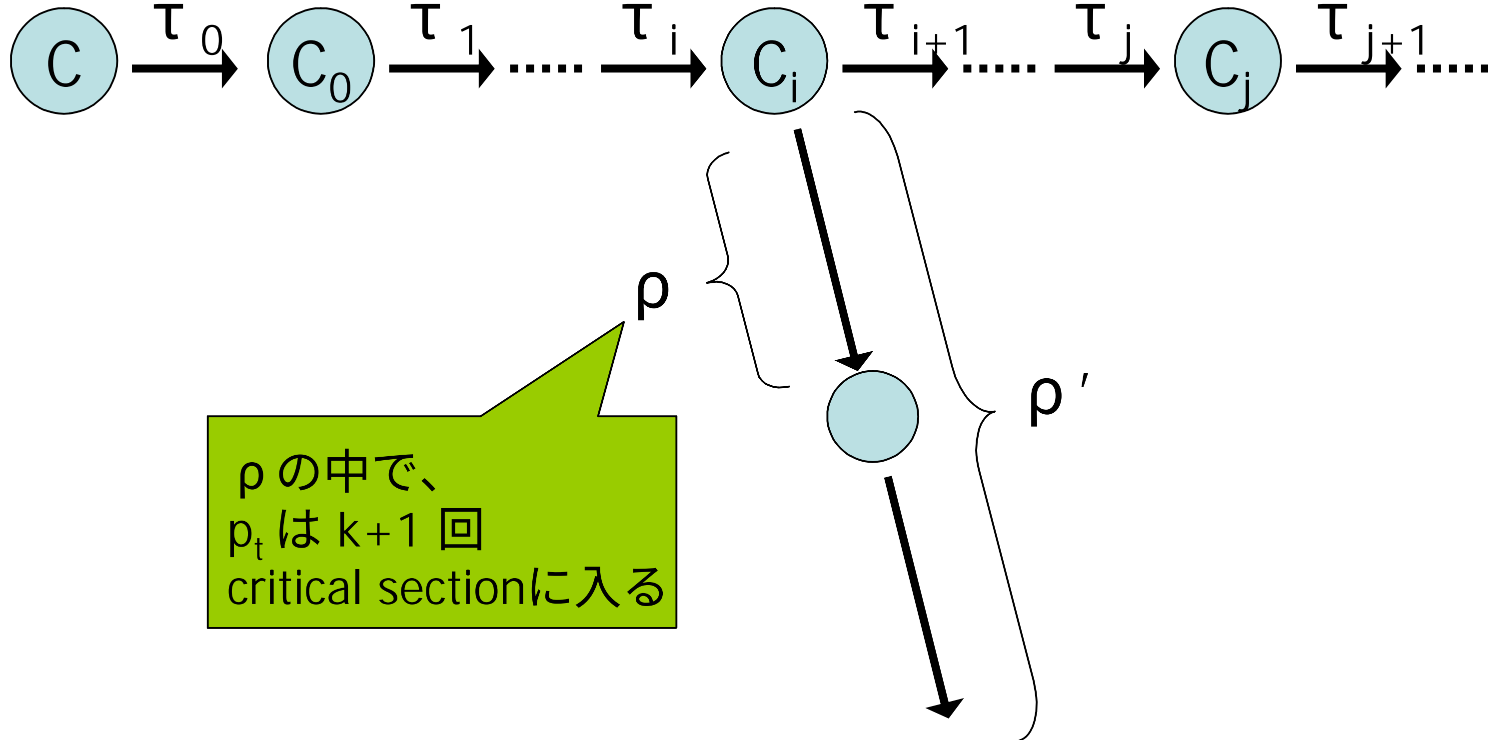
Theorem 4.4の証明

- no deadlock の性質から、exec(C_i, ρ') で無限回 critical section に入っている プロセッサ p_t ($0 \le t \le i$) が存在する
  - プロセッサ p_0, …, p_i は無限の計算ステップを実行するため

Theorem 4.4の証明

- 以下のような ρ' の有限なプレフィックス ρ を考える
  - ρ では、p_t は k+1回 critical section に入る (k: 任意の0以上の整数)

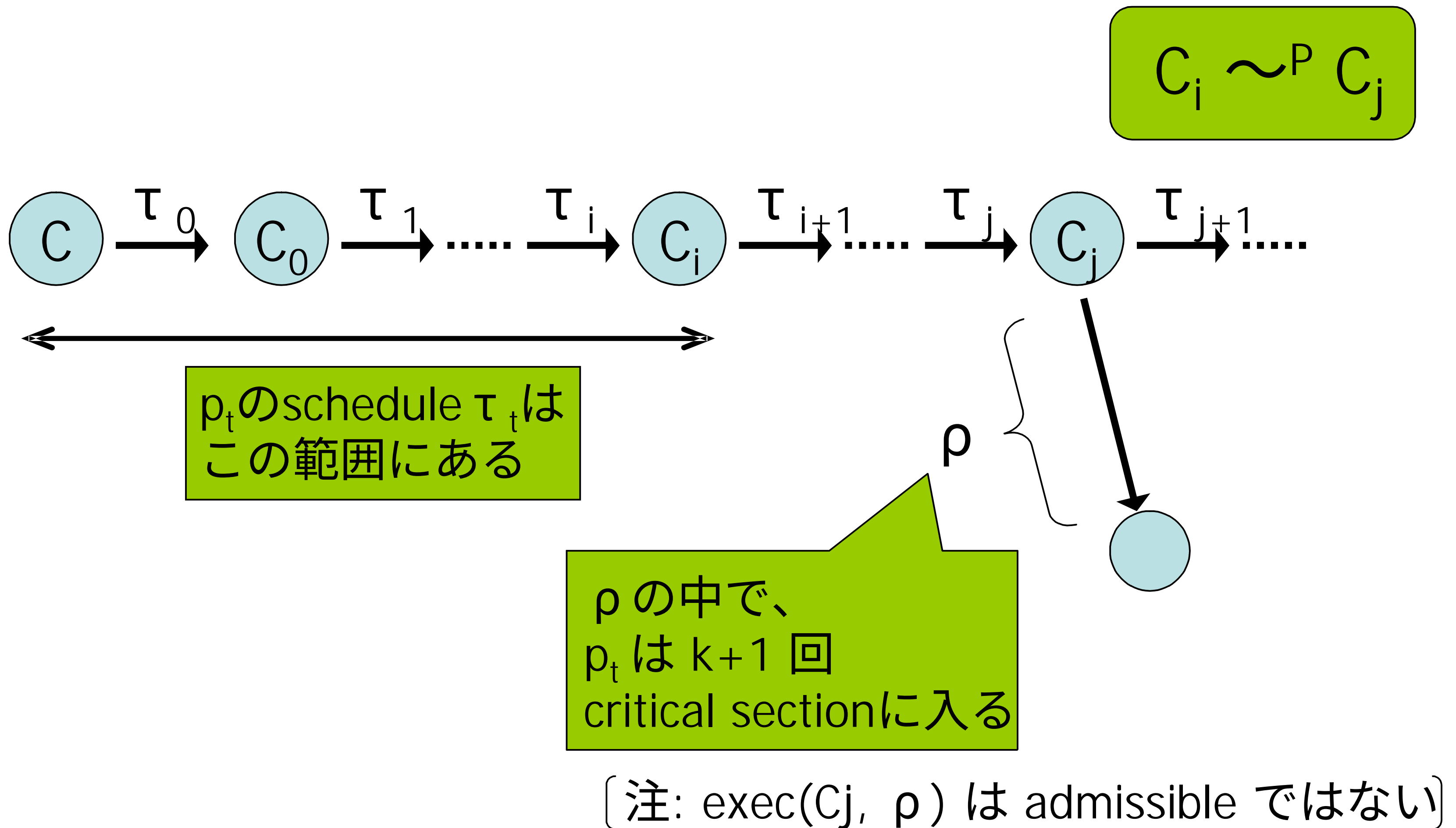
Theorem 4.4の証明



Theorem 4.4の証明

- 以下から、p_t は exec(C_j, ρ) でも k+1回 critical sectionに入ることになる
  - $C_i \sim^P C_j$ $(P = \{p_0, \ldots, p_i\})$
  - ρ は $\{p_0, \ldots, p_i\}$-only な schedule
- exec(C_j, ρ) では、p_j が entry section で待っている一方で、p_t は k+1回 critical section に入っていることになる
  - p_j は C_j で entry section に入るため

Theorem 4.4の証明



［注: exec(Cj, ρ) は admissible ではない］

Theorem 4.4の証明

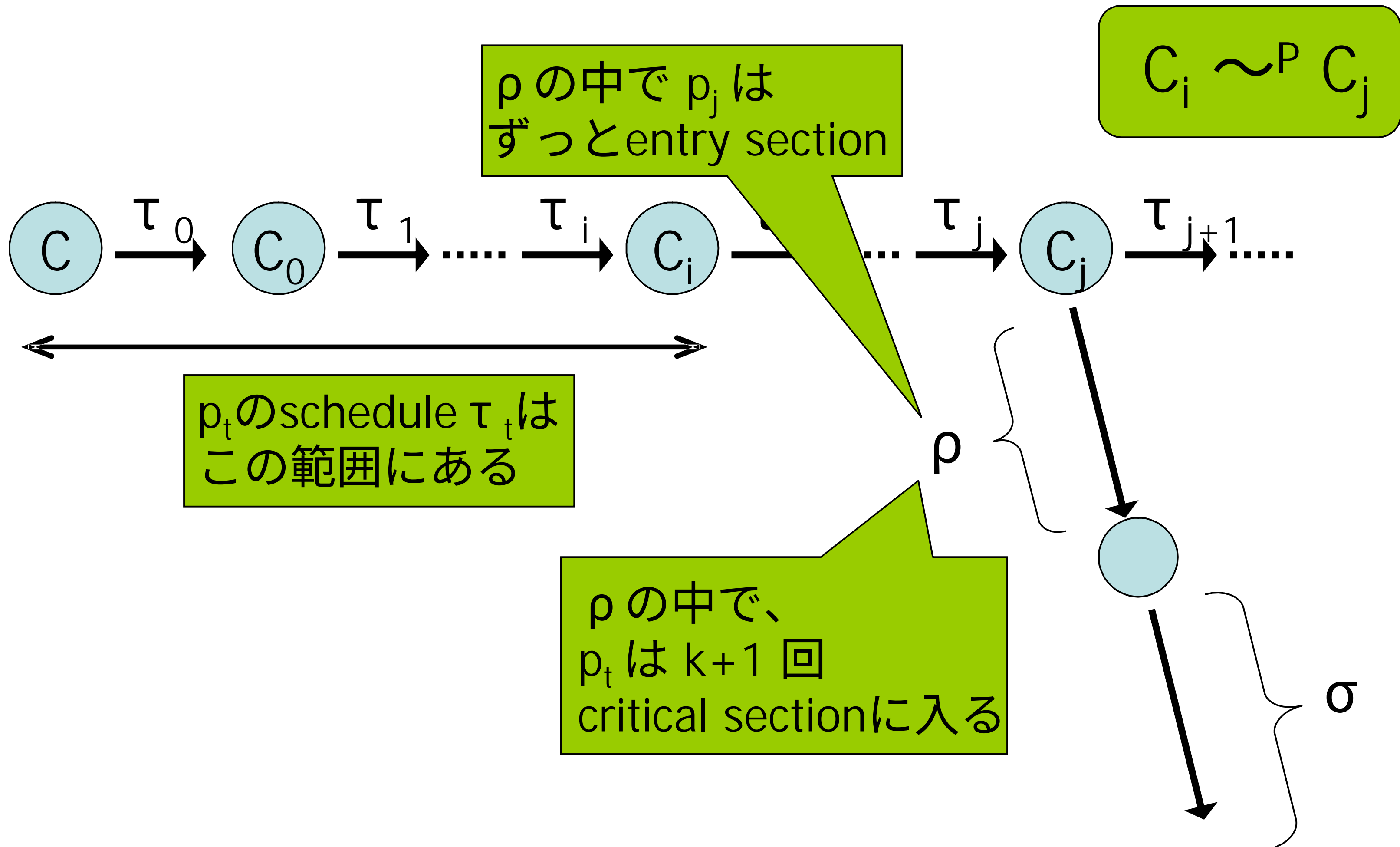
- 以下を満たす無限 schedule σ を考える
  - $p_0, \ldots, p_j$: 無限回の計算ステップを実行
  - 残り: 何もしない
- すると、$exec(C, \tau_0 \tau_1 \ldots \tau_j \rho \sigma)$は admissible

Theorem 4.4の証明

- exec(C, $\tau_0 \tau_1 \ldots \tau_j \rho \sigma$)の ρ の部分の execution segment を考える
  - p_t は p_j を k+1回以上追い越して critical section に入っている
- よって、矛盾
  - kは任意であったから、k-bounded waiting に反する

[証明終]

Theorem 4.4の証明



補足

- Theorem 4.4 が示しているのはk-bounded waiting であり、no lockout ではない
  - p_j は p_t に k+1 回先を越された後にcritical section に入るかもしれない