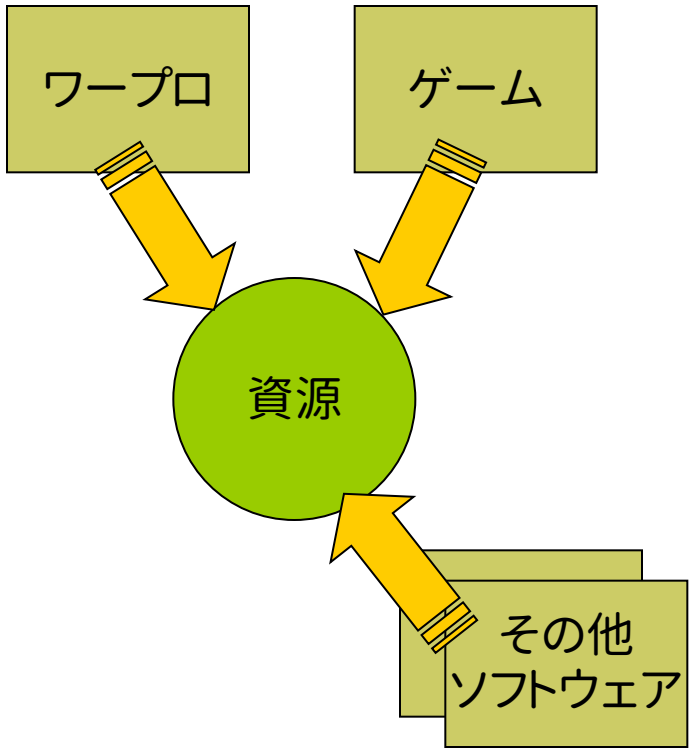
Operating Systems

共有資源と資源管理

2020-08

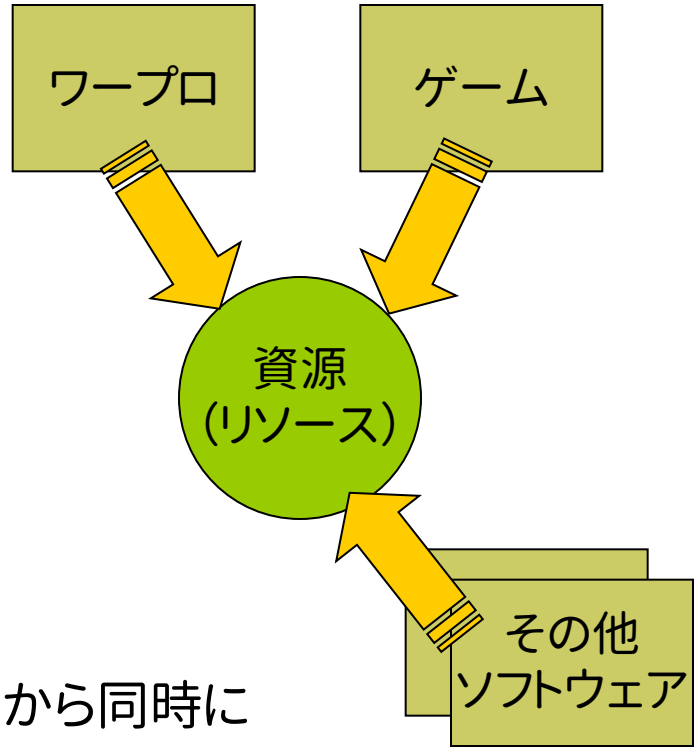
OSの役割(復習)

- 資源の管理
  - コンピュータの資源=リソース
    - CPU,メモリ,ディスク,画面,etc…
  - コンピュータ資源の効率的利用を図る
- コンピュータシステムの制御
  - メモリやディスクをコントロールする
  - 資源へのアクセス制御を行う
  - コンピュータシステムの交通整理をする
- 仮想計算機の提供
  - ソフトウェアに分かりやすいハードウェアモデルを提供する
  - ハードウェアや周辺機器の違いを吸収する



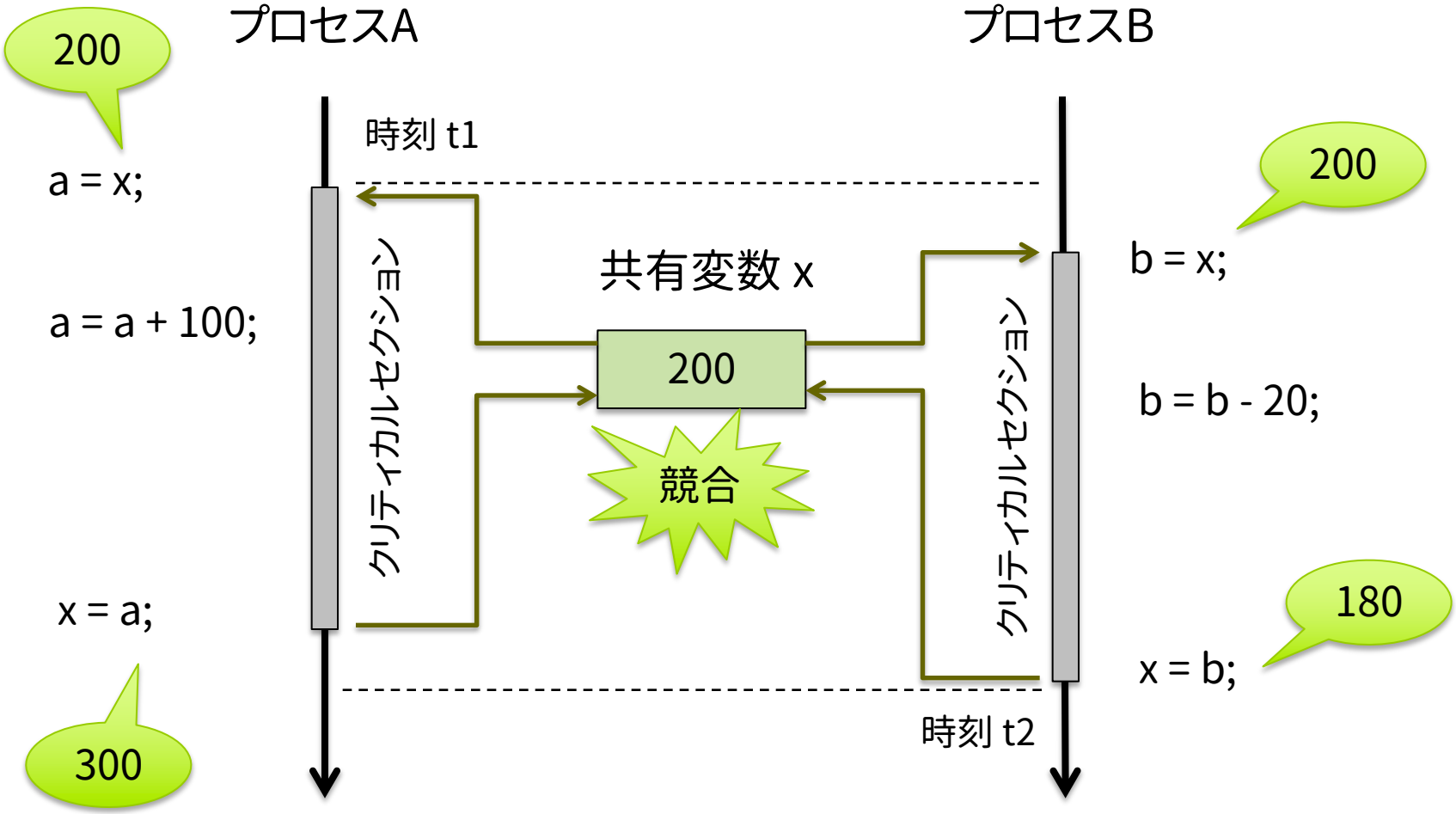
共有資源

- 共有資源とは
  - 複数のプロセスが共有する資源
    shared resource
  - 共有資源の例:
    入出力デバイス,共有メモリ領域,CPU
  - 共有資源でない例:
    各プロセスに割り当てられたメモリ領域



- 共有資源の「競合」
  - ほとんどの共有資源は,複数のプロセスから同時にアクセスされるとトラブルの原因になる
  - プロセスが資源を独占するために他のプロセスを排除しようとすると,資源を奪い合う「競合」が起きる
  - 競合を調整するのは,OSの重要な役割(排他制御機能)

共有資源の問題


プロセスA
プロセスB
200
時刻 t1
a = x;
a = a + 100;
共有変数 x
200
競合
クリティカルセクション
クリティカルセクション
200
b = x;
b = b - 20;
x = a;
180
x = b;
300
時刻 t2

クリティカルセクション

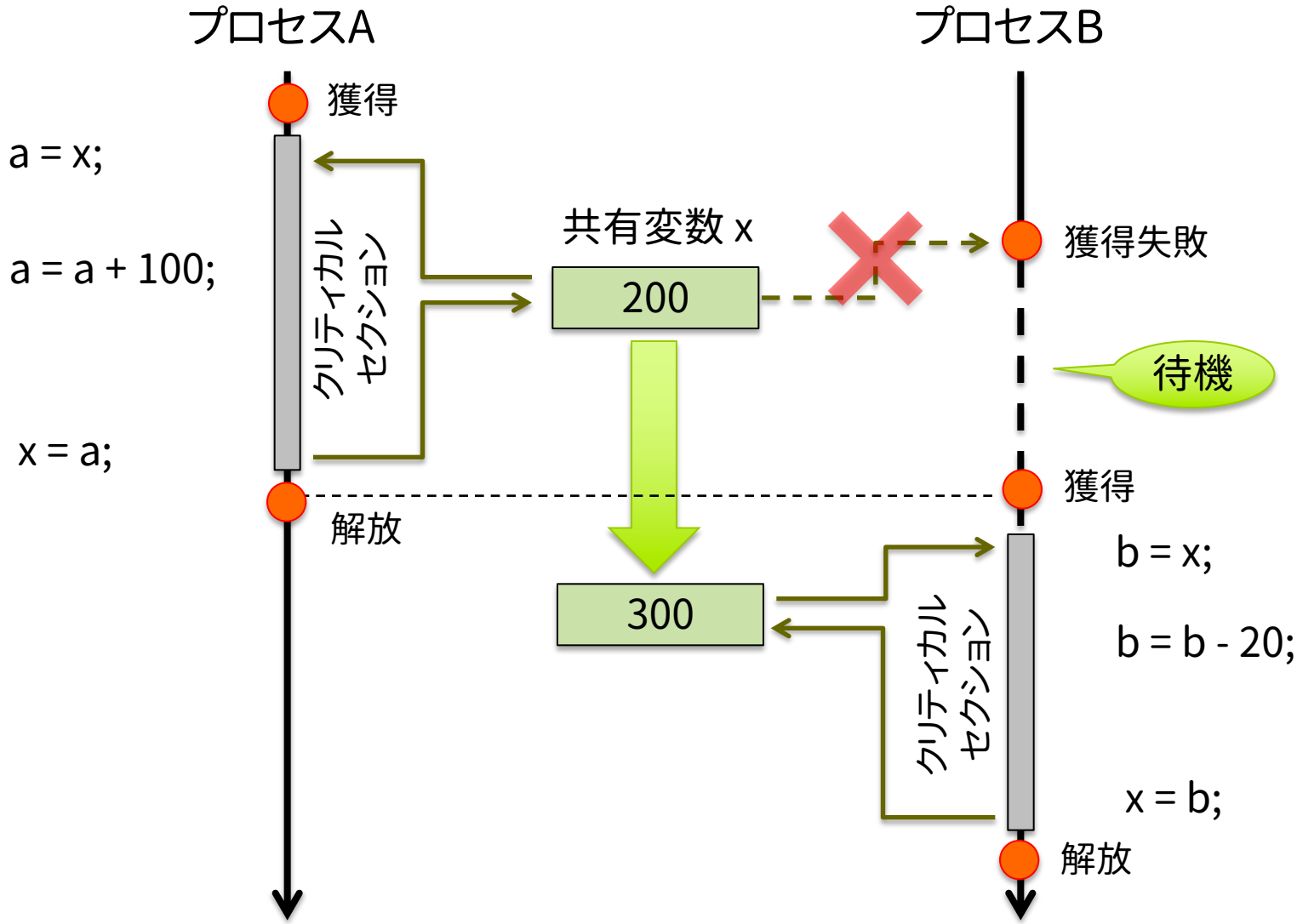
- クリティカルセクション
  - 「臨界領域」「きわどい部分」（critical＝致命的）
  - プログラムの中で，共有資源にアクセスする処理の部分
  - 複数のプロセスが，同じ共有資源に関するクリティカルセクションに入ってしまうと，データや処理の一貫性が保てなくなってしまう
  - 【注】クリティカルセクションは，プログラムの中の部分（処理手順）のことで，機能や方法の名前ではない

- たとえ話…
  - 漫画家のX氏とY氏は，2人共同で1つの漫画を描いている。
  - ある日，X氏が原稿（＝共有資源）の1ページに誤りを見つけて書き直し始めた（＝クリティカルセクションに入った）。
  - ところが，次の日に原稿を差し替えようとすると，Y氏も同じ箇所を書き換えていたため，どちらを優先すべきか競合が起きた。

排他制御

- 排他制御とは
  - 共有資源の使用に不可欠なOSの機能
  - あるプロセスがクリティカルセクションに入っているとき，別のプロセスが新しく入ってこないように，“お互いに排除”するための機能
- 資源の「獲得」
  - プロセスは，クリティカルセクションの入口で資源の「獲得」を試みる
  - 資源を獲得できた場合は，そのままクリティカルセクションに入る
  - 資源が獲得できなかった場合は，獲得できるまでそこで待機する
  - データベース等では，「ロック」（鍵をかける）ということが多い
- 資源の「解放」
  - プロセスは，クリティカルセクションの出口で使った資源を「解放」する
  - データベース等では，「アンロック」（開錠する）ということが多い

排他制御


プロセスA
プロセスB
獲得
a = x;
a = a + 100;
クリティカル
セクション
共有変数 x
200
獲得失敗
待機
x = a;
解放
獲得
b = x;
300
b = b - 20;
クリティカル
セクション
x = b;
解放

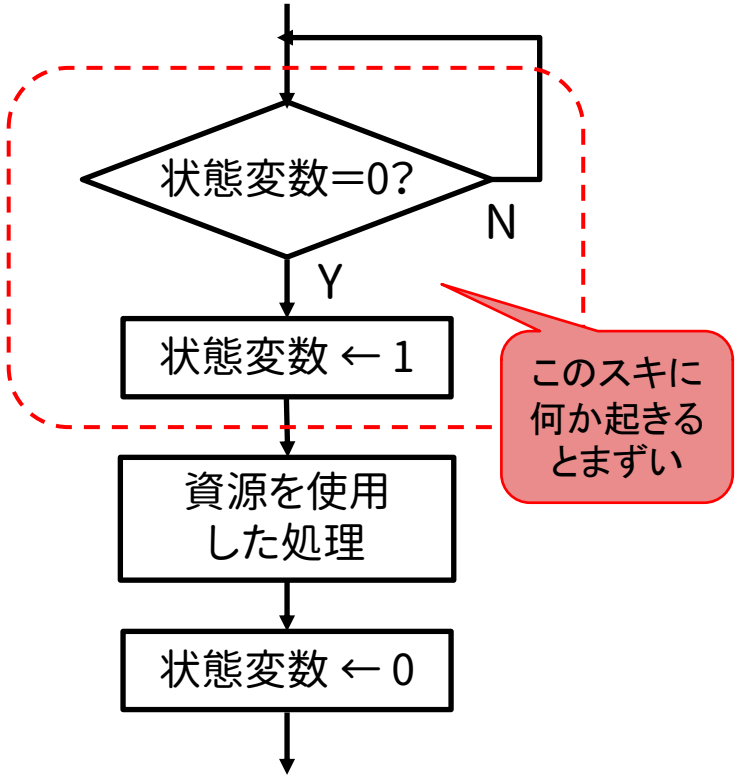
排他制御の難しさ

- 排他制御で必要なこと
  - 資源の獲得と解放は，他の処理よりも優先的に処理されること
  - 使用されなくなった資源は，どんな場合でも直ちに解放されること
  - 実行プロセスが切り替わっても（切り替わる瞬間でも）破綻しないこと

- 問題と対策の例
  - あるプロセスが資源の空きをチェックして獲得を試みている途中に，コンテクストスイッチが起きてプロセスBに切り替わったら？
    ⇒【対策】獲得・解放処理の間は，コンテクストスイッチを禁止する
  - プロセスが資源を獲得したまま，（エラー等で）実行を中止したら？
    ⇒【対策】OSが実行中止を検知して，強制的に資源を解放する
  - プロセスが資源を使用中に，他のプロセスに実行が切り替わったら？
    ⇒【対策】プロセスの中断中でも，資源は保持し続けるようにする

単純な排他制御

- スピンロック(繁忙待機)
  - 最も初期からある単純な方法
  - 資源が空くまでひたすらループで待つ
  - 資源が「使用中」(1)か「空き」(0)かを状態変数の値で管理する
  - 資源の「獲得」は,変数に1にセットする
  - 資源の「解放」は,変数を0にクリアする



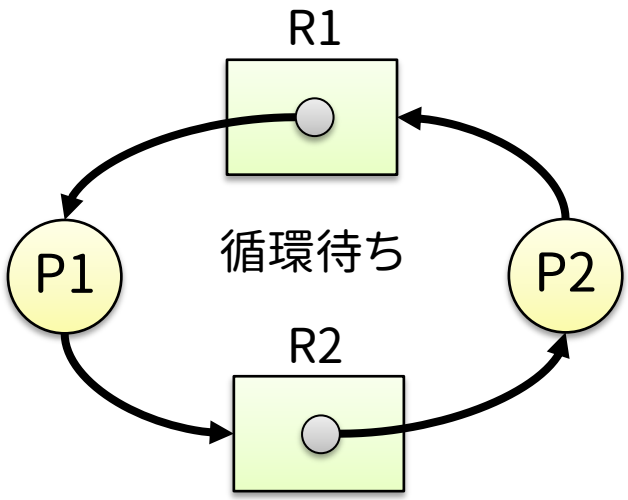
- スピンロックの特徴
  - OSに特別な機能が必要ない
  - CPUには,変数のテストとセット(赤枠部分)を,途中で割り込み等が発生しない不可分な(アトミックな)処理として行う命令が必要
  - マルチプロセッサシステムなどで,待ちが確実に瞬時に終わるようなケースでは,オーバーヘッドが最小なので現在でも用いられる

OSによる排他制御

- スピンロックの問題点
  - 資源の空きをひたすらチェックし続けるので，時間と電力を浪費する
  - その間，他のプロセスの実行時間を奪うことになり，効率が悪い
- ではどうすればいいのか…
  - OSが排他制御を管理する（APIを通した排他制御）
  - 資源が使えない時点で，OSがそのプロセスを“眠らせる”
    ＝ プロセスの状態を「待ち状態」に遷移させる
  - 資源が使えるようになったら，OSがプロセスを“目覚めさせる”
    ＝ プロセスの状態を「実行可能状態」に遷移させる
- OSが提供する排他制御機能
  - セマフォ，ミューテックス，モニタなど ⇒ 次回説明

デッドロック

- デッドロック
  - 排他制御のせいで，プログラムやシステムが停止してしまう状態
  - deadlock＝「行き詰まり」，「膠着状態」
- なぜ起きるのか
  - 複数のプロセスが，互いが獲得した資源が空くのを待ちあって，どちらも先に進めなくなってしまう
- どんな状態か
  - プロセスP1は資源R1を獲得したまま資源R2が空くのを待ち，プロセスP2は資源R2を獲得したまま資源R1を待つ
  - このように，必ず“循環待ち”が含まれる



考えてみよう…

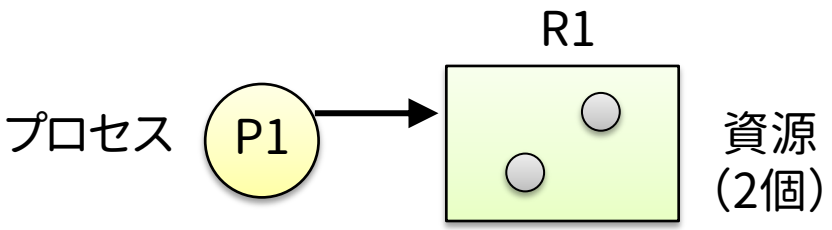
- ある日，A君とB君が，別々に電車のおもちゃで遊ぼうとして遊び場に来たが，おもちゃは1セットしかなかった・・・

- A君は，列車を組み立ててからレールを組み立てようとして，先に列車を「獲得」した。
- B君は，レールを組み立ててから列車を組み立てようとして，先にレールを「獲得」した。
- しばらくすると，お互いの持っているものを待って（または奪おうとして）遊びは中断した。（デッドロック）

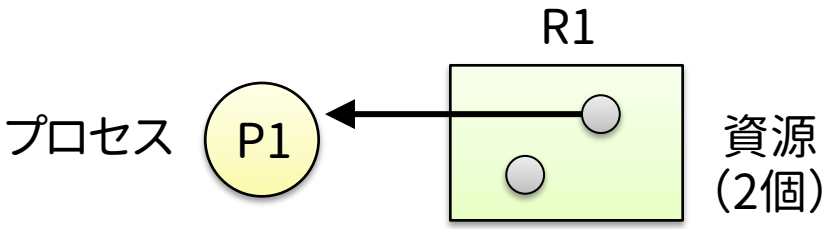
- どのような遊び方のルールを決めておけば，順番に遊ぶなど無駄な中断を防げるか？（仲良く遊ぶという解決策以外で）

資源割付けグラフ

- 資源割付けグラフ
  - プロセスの資源要求状態を表す図(グラフ構造)

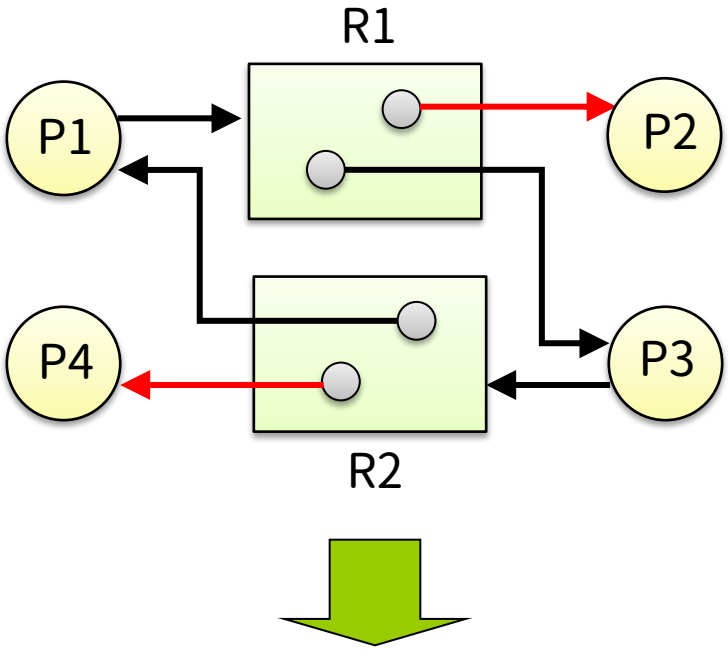


資源を要求している



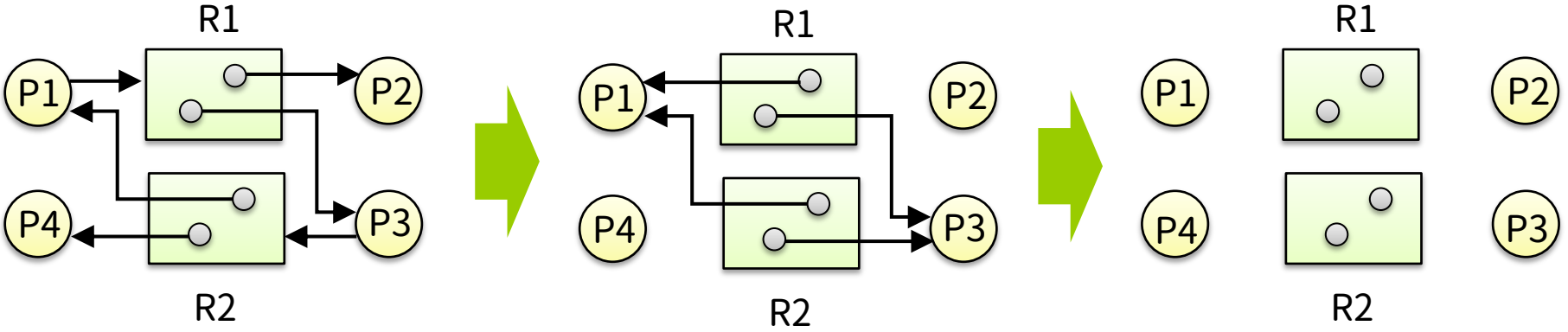
資源が割り付けられている

- デッドロックの検出
  - 資源割付グラフを簡約化する

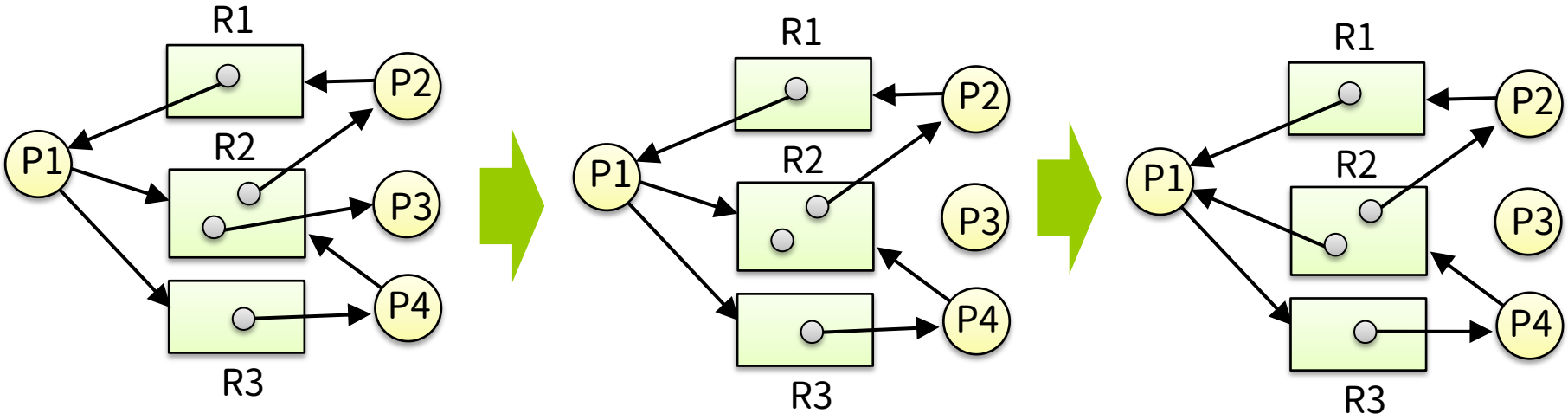


プロセスのすべての資源要求が満足されたら,矢印を削除していく

資源割付けグラフの簡約化



P2，P4は順調に実行　　P1, P3が資源を獲得　　デッドロックせず終了



P3は順調に実行　　R2をP1が獲得してしまうと　　デッドロック発生

デッドロックの発生条件

- 条件1「相互排除条件」
  - プロセスは，獲得した資源を独占し，他のプロセスの利用を排除する
- 条件2「待ち条件」
  - プロセスは，複数の資源が必要な場合，獲得した資源を保持したまま，他の資源が空くのを待つ
- 条件3「横取り不可能条件」
  - プロセスは，資源を自ら解放するまで，それを取り上げられない
- 条件4「循環待ち条件」
  - プロセスAが要求している資源をプロセスBが保持し，プロセスBが要求している資源をプロセスAが保持するような，循環待ちが存在する
- 以上の条件すべてが当てはまると，デッドロックが発生！

デッドロックへの対策

- デッドロックの発生を防止する
  - デッドロックの発生条件が絶対に成り立たないようにする
  - または，なるべく成り立たないように工夫する
- デッドロックを検出し回復する
  - 資源割り付けグラフを一定時間ごとにチェックし，デッドロックを検出したら回復する，またはデッドロック発生前に戻ってやり直す
  - ある時点まで戻ってやり直すことを，ロールバック（rollback）という
- 他の解決策…
  - 最後は無視する！（他の致命的エラーとの頻度の問題）
  - 個人作業向けのOSなら，デッドロックの発生確率を十分に下げれば，発生時の処理は省く方法もある ⇒ もし発生したら，再起動…

デッドロックの防止

- 防止策1「待ち条件の防止」
  - 例）各プロセスは，処理に必要な全資源を一度に要求するようにする
  - つまり，ある資源を保持したまま，別の資源を待たないようにする
- 防止策2「横取り不可能条件の防止」
  - 例）プロセスがある資源の獲得に失敗したら，そこで処理を中断して保持している資源をすべて手放す（いったん他のプロセスに譲る）
  - つまり，資源を使い続けたくても，途中で使用を止めて解放する
- 防止策3「循環待ち条件の防止」
  - 例）資源に順番（番号）をつけ，獲得するときはその順番を守る
- 以上は，デッドロックの発生条件2・3・4にそれぞれ対応し，どれか1つでも実現できればデッドロックを防止できる

演習課題

- 課題 8a デッドロックの防止
  - 電車のおもちゃのたとえ話について，デッドロックの状況を、今回授業で説明した形式の資源割付けグラフを描いて説明せよ。
  - デッドロックの発生を防止するために，3つのデッドロックの防止策のそれぞれに対応した3つ（以上）の遊びのルールを考えよ。
  - 注：人間的な交渉による解決策（例：話し合って一緒に遊ぶ）はダメ。

【たとえ話】

- A君とB君が，別々に電車のおもちゃで遊ぼうとして遊び場に来たが，おもちゃは1セットしかなかった。
- A君は，列車→レールの順に組み立てようとして，列車を「獲得」した。
- B君は，レール→列車の順に組み立てようとして，レールを「獲得」した。
- しばらくして，2人とも相手が使い終わるのを待つことになり，遊びが中断してしまった。