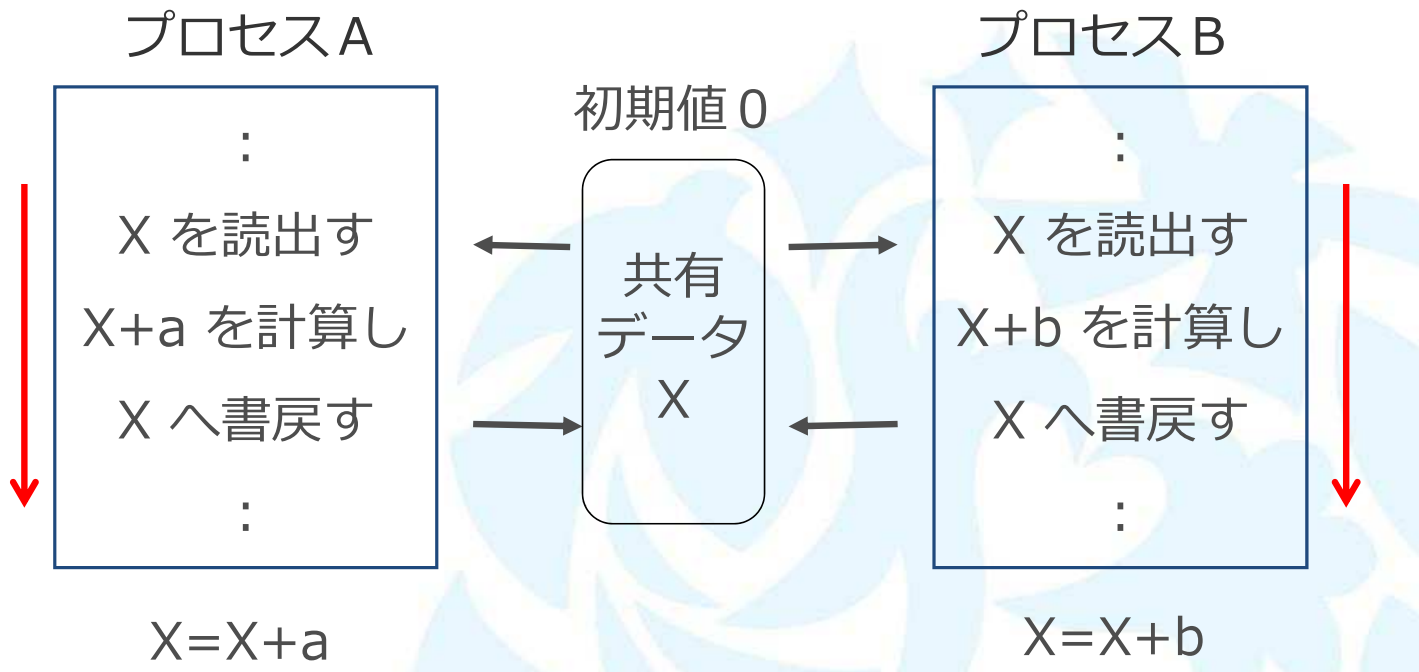
排他制御の仕組1

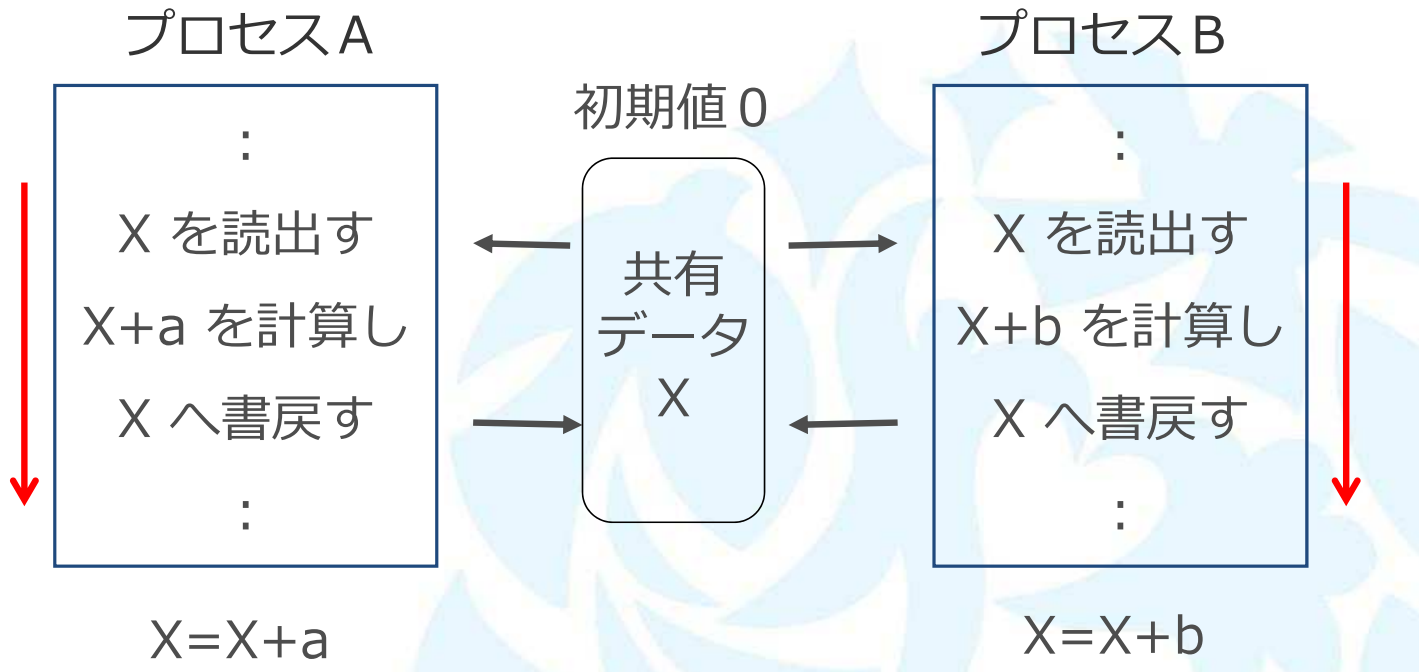
まず
共有変数への排他制御を
考えてみる

共有変数に
「読んで書く」更新をすると
どうなるか

共有変数の更新

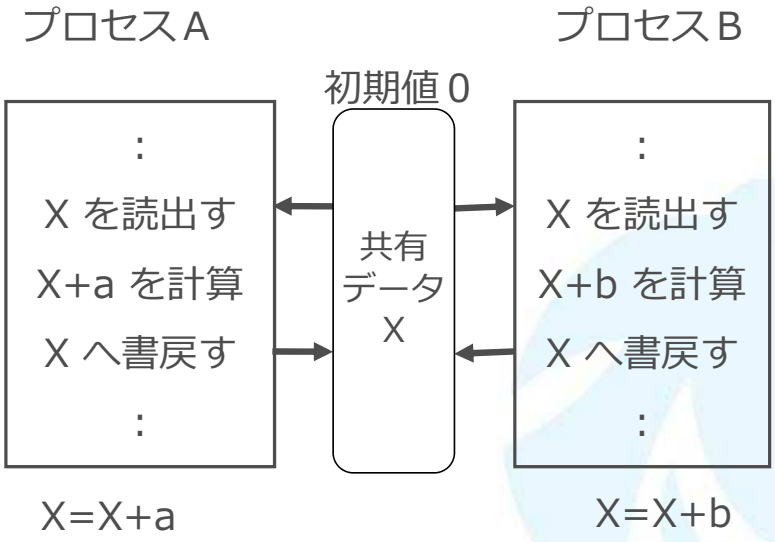


共有変数の更新



同時に走ると、どうなるか？

共有変数の更新



タイミング例1

↓

Aが読出す(0)

Aが書戻す(a)

Bが読出す(a)

Bが書戻す(a+b)

↓

X=a+b

タイミング例2

↓

Aが読出す(0)

Bが読出す(0)

Aが書戻す(a)

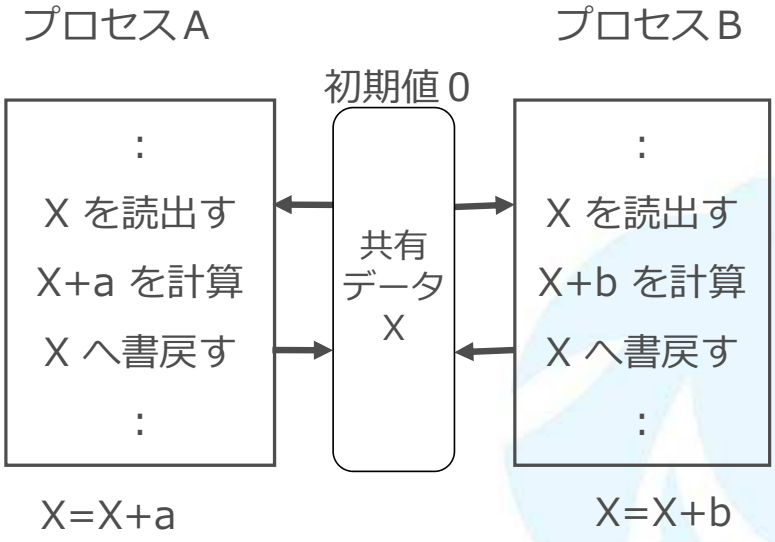
Bが書戻す(b)

↓

X=b

結果が違う

共有変数の更新



タイミング例1

↓

Aが読出す(0)

Aが書戻す(a)

Bが読出す(a)

Bが書戻す(a+b)

↓

X=a+b

タイミング例2

↓

Aが読出す(0)

Bが読出す(0)

Aが書戻す(a)

Bが書戻す(b)

↓

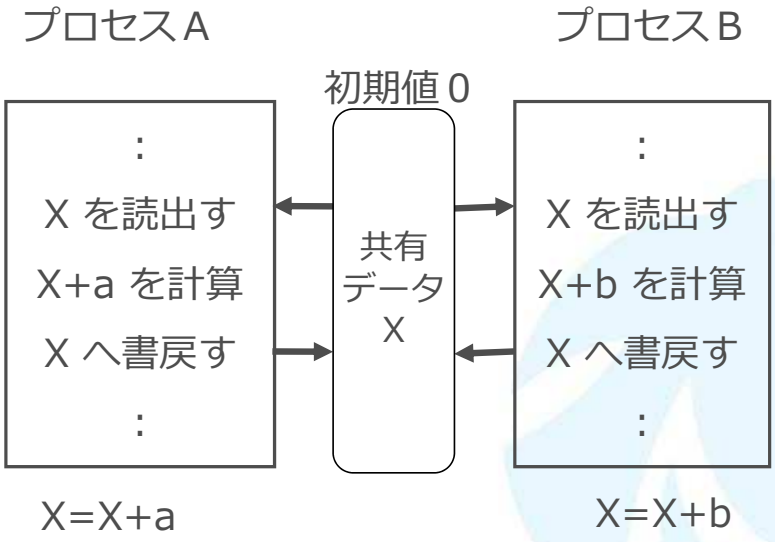
X=b

結果が違う

本来なら0に a を足して b を足すので
X は a+b になる

このタイミングだと
X は b になる

共有変数の更新



タイミング例1

↓

Aが読出す(0)

Aが書戻す(a)

Bが読出す(a)

Bが書戻す(a+b)

↓

X=a+b

タイミング例2

↓

Aが読出す(0)

Bが読出す(0)

Aが書戻す(a)

Bが書戻す(b)

↓

X=b

結果が違う

本来なら0に a を足して b を足すので
X は a+b になる

このタイミングだと
X は b になる

なぜか

東邦

共有変数の更新

順序がおかしい

↓

Aが読出す(0)

Aが書戻す(a)

Bが読出す(a)

Bが書戻す(a+b)

↓

X=a+b

↓

Aが読出す(0)

Bが読出す(0)

Aが書戻す(a)

Bが書戻す(b)

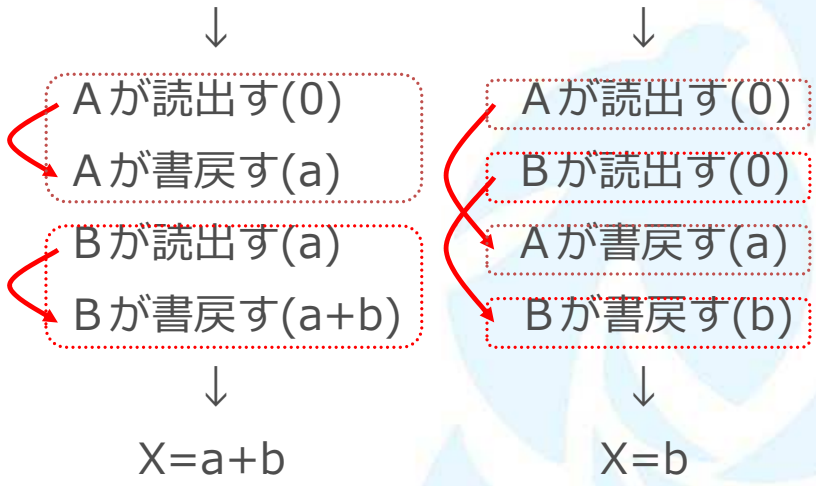
↓

X=b

東邦

共有変数の更新

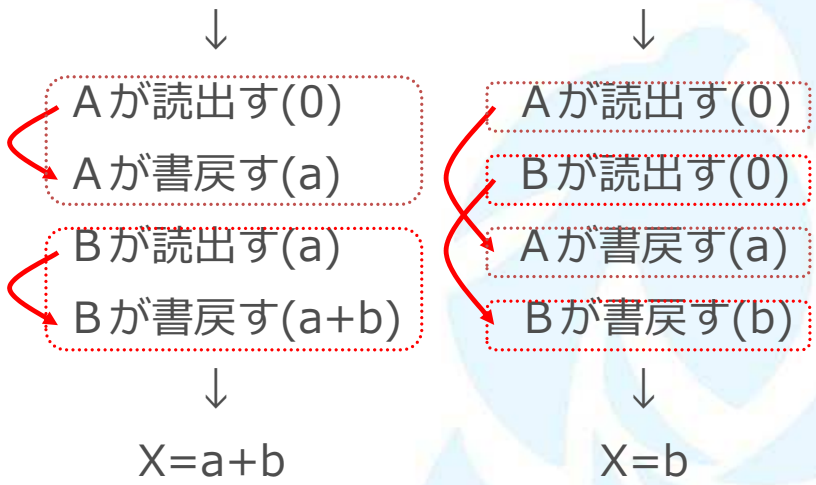
順序がおかしい



Aが書き戻していないので、まだ 0 なのにBが読出した

共有変数の更新

順序がおかしい



Aが書き戻していないので、まだ 0 なのにBが読出した

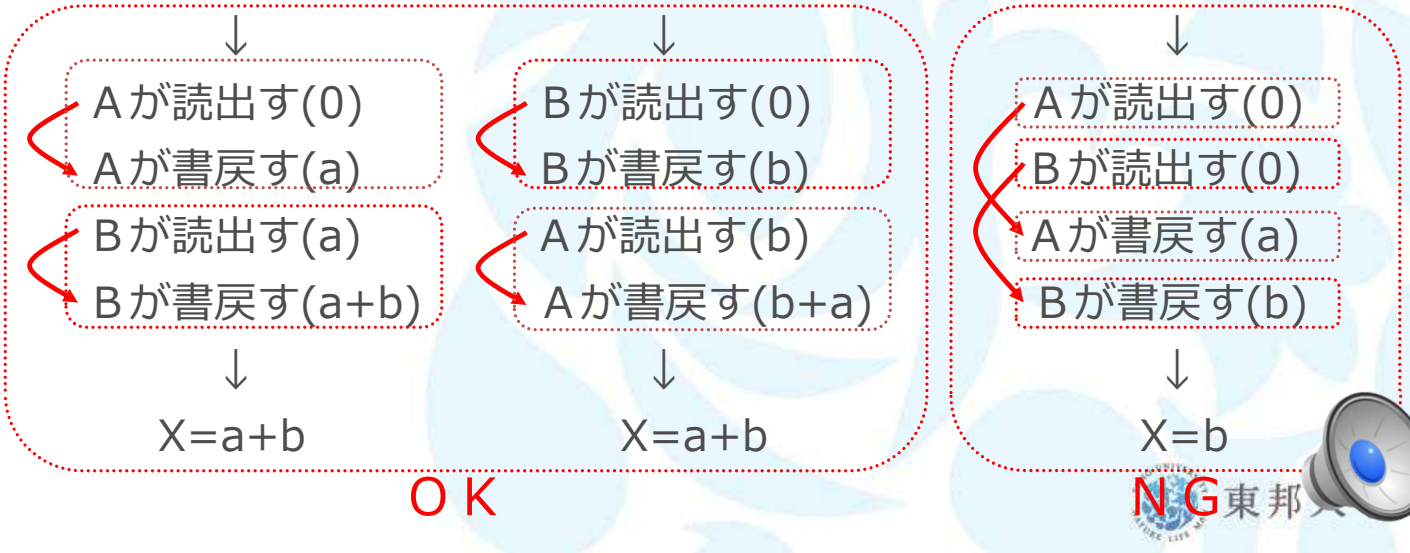
どうすればよいのか？

共有変数の更新

- Ａが先に読む+書くしてから、Ｂが読む+書くか、
- Ｂが先に読む+書くしてから、Ａが読む+書くか、
  どちらかに制限する

共有変数の更新

- Ａが先に読む+書くしてから、Ｂが読む+書くか、
- Ｂが先に読む+書くしてから、Ａが読む+書くか、
  どちらかに制限する　（左の２つならＯＫ）



共有変数の更新

- Aが先に読む+書くしてから、Bが読む+書くか、
- Bが先に読む+書くしてから、Aが読む+書くか、

どちらかに制限する

↓

- Aの(読む+書く)の間は、Bはアクセスさせない
- Bの(読む+書く)の間は、Aはアクセスさせない

共有変数の更新

- Aが先に読む+書くしてから、Bが読む+書くか、
- Bが先に読む+書くしてから、Aが読む+書くか、

どちらかに制限する

↓

- Aの(読む+書く)の間は、Bはアクセスさせない
- Bの(読む+書く)の間は、Aはアクセスさせない

||

「相互排他」

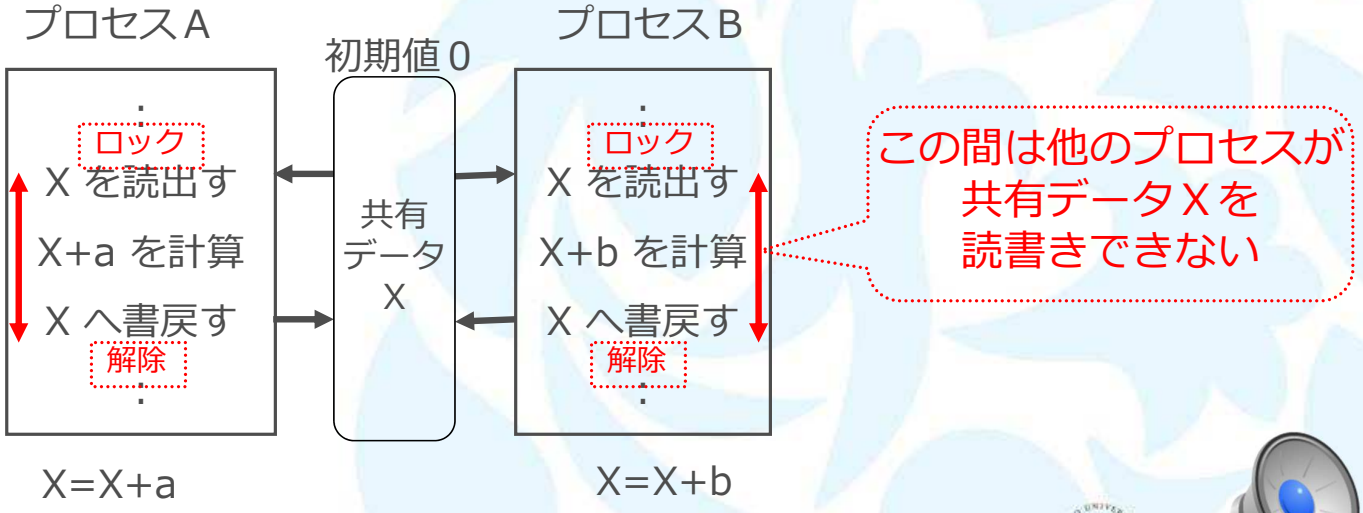
排他の仕組＝ロック

排他の仕組＝ロック

- 共有データXを一方のプロセスがロックすると、
  解除するまで、
  他のプロセスはアクセス（読書き）できない

排他の仕組＝ロック

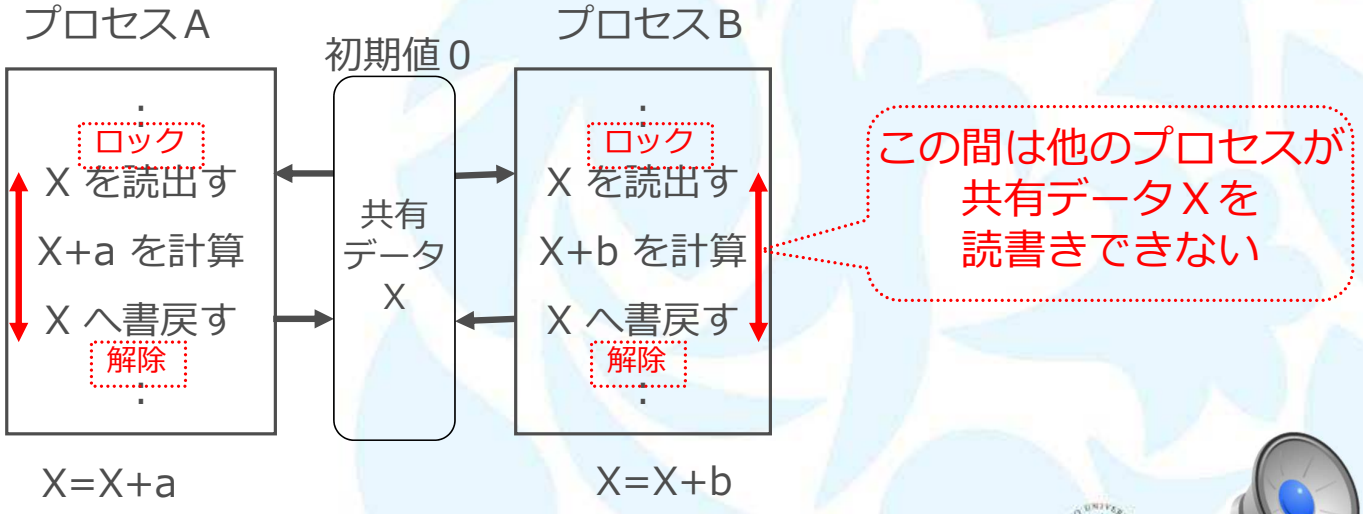
- 共有データXを一方のプロセスがロックすると、解除するまで、
  他のプロセスはアクセス（読書き）できない



排他の仕組＝ロック

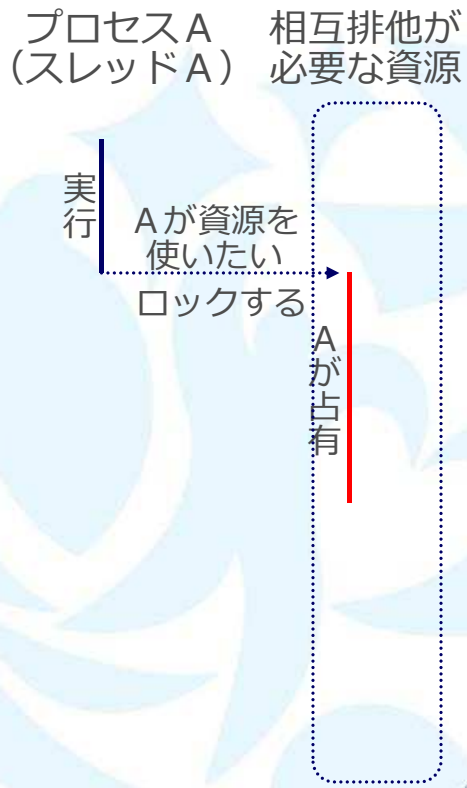
- 共有データXを一方のプロセスがロックすると、解除するまで、
  他のプロセスはアクセス（読書き）できない

資源確保
資源の解放
資源を占有



相互排他と占有

- プロセス（orスレッド）が排他的な資源を使いたくなる
- 空いていれば占有（ロック）できる

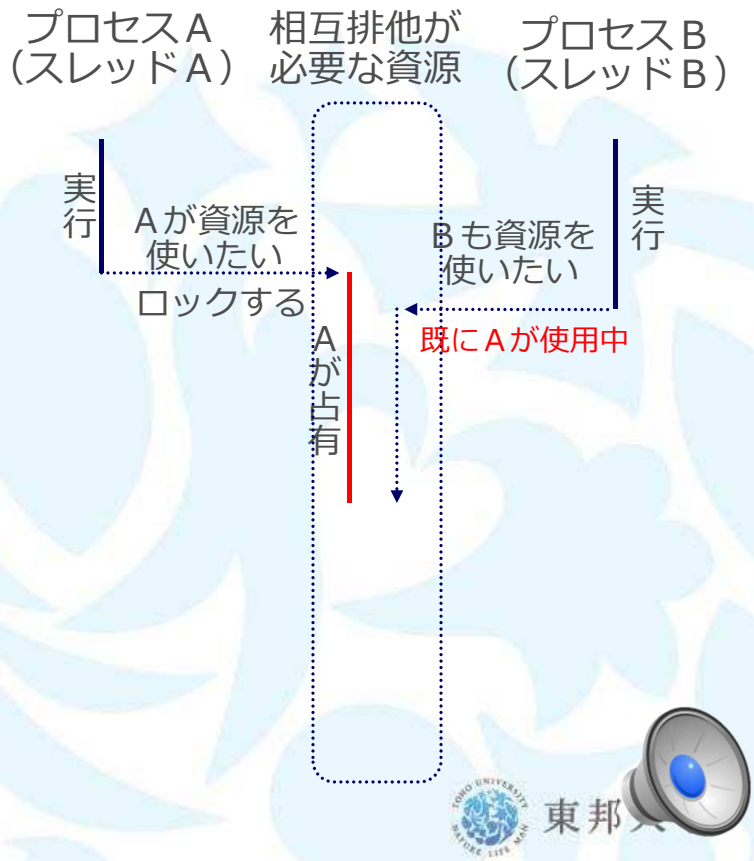


相互排他と占有

- プロセス（orスレッド）が排他的な資源を使いたくなる
- 空いていれば占有（ロック）できる
- 使い終わったら解放

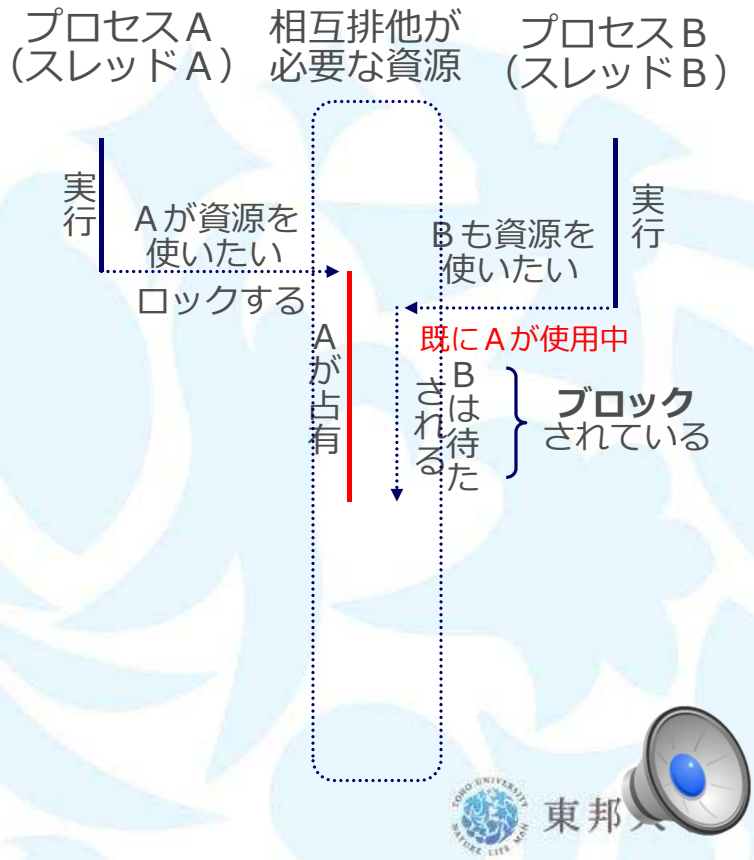
相互排他と占有

- プロセス（orスレッド）が排他的な資源を使いたくなる
- 空いていれば占有（ロック）できる
- 他のプロセスは資源が空くまで待たされる（ブロックされる）



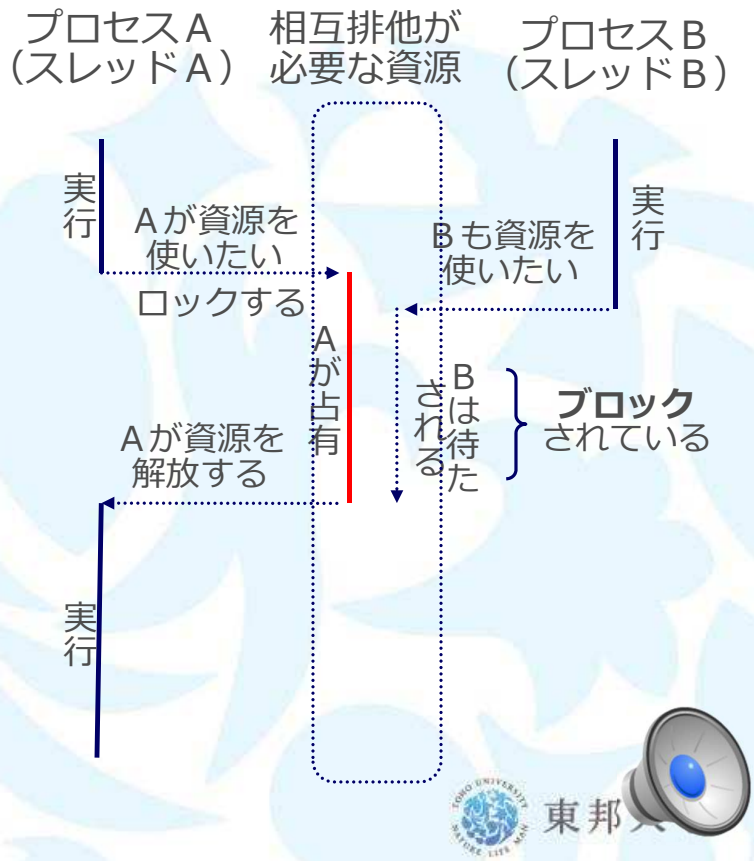
相互排他と占有

- プロセス（orスレッド）が排他的な資源を使いたくなる
- 空いていれば占有（ロック）できる
- 他のプロセスは資源が空くまで待たされる（ブロックされる）



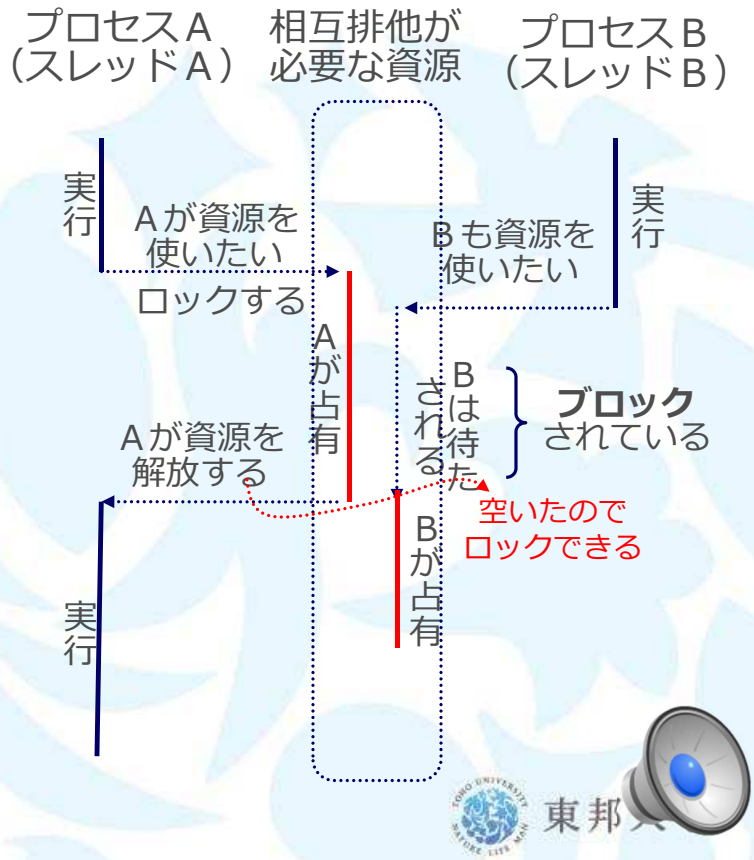
相互排他と占有

- プロセス（orスレッド）が排他的な資源を使いたくなる
- 空いていれば占有（ロック）できる
- 使い終わったら解放
- 他のプロセスは資源が空くまで待たされる（ブロックされる）



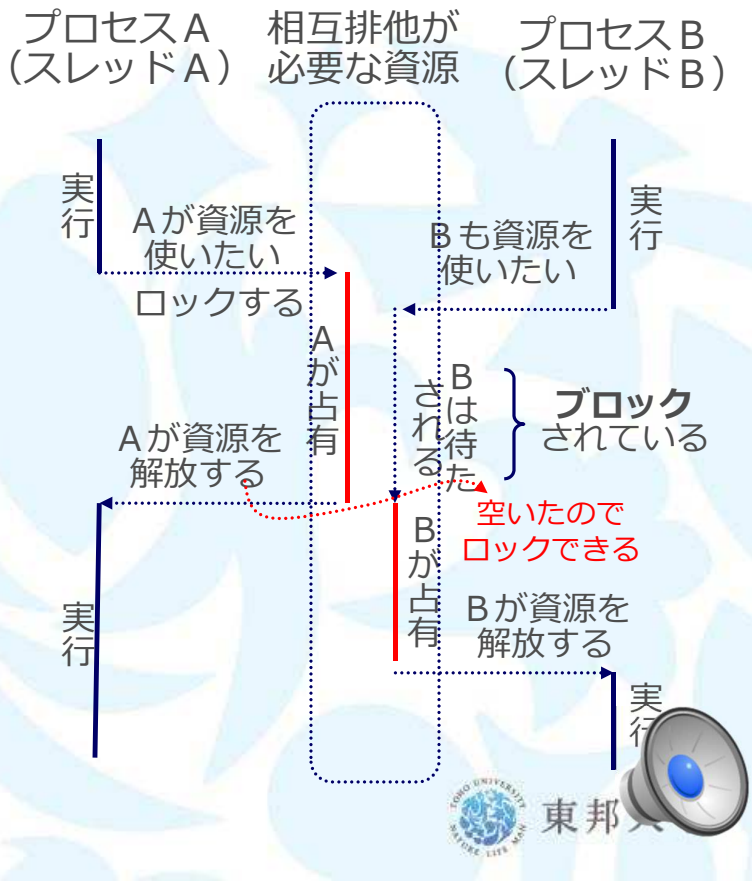
相互排他と占有

- プロセス（orスレッド）が排他的な資源を使いたくなる
- 空いていれば占有（ロック）できる
- 使い終わったら解放
- 他のプロセスは資源が空くまで待たされる（ブロックされる）



相互排他と占有

- プロセス（orスレッド）が排他的な資源を使いたくなる
- 空いていれば占有（ロック）できる
- 使い終わったら解放
- 他のプロセスは資源が空くまで待たされる（ブロックされる）

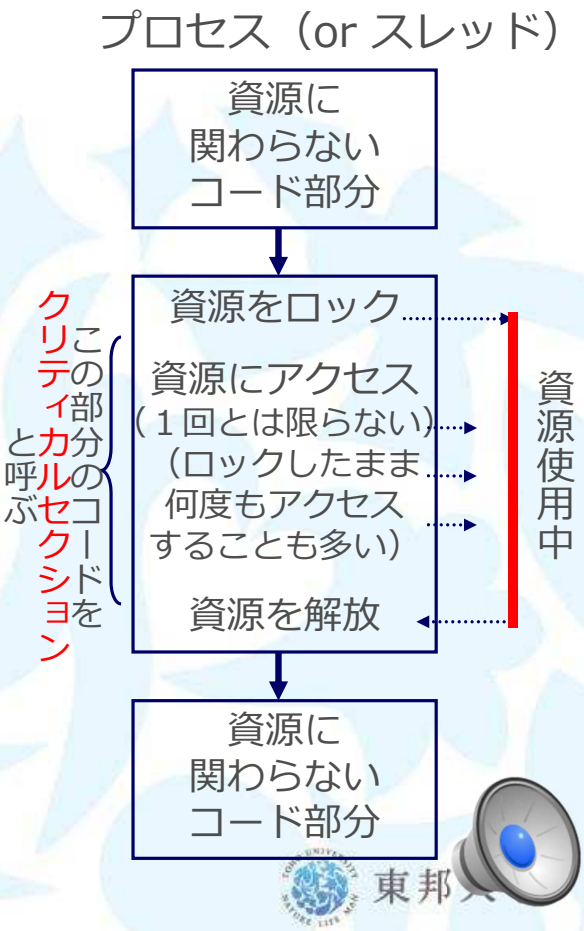


ことば： クリティカルセクション

- 相互排他
  ＝ mutual　exclusion
  短縮してmutexと呼ぶ
- ロック ＝ かぎ（錠）～を掛けて他が入れない、占有

ことば： クリティカルセクション

- 相互排他
  = mutual　exclusion
  短縮してmutexと呼ぶ
- ロック = かぎ（錠）～を掛けて他が入れない、占有
- クリティカルセクション
  （クリティカルリージョン）
  ||
  プログラム中で、資源に排他的にアクセスする部分のこと



（脱線）ロックの実現

（脱線）ロックの実現

- 共有変数「ロック中」（使用中）を用意し、

（脱線）ロックの実現

- 共有変数「ロック中」（使用中）を用意し、
- もし「１」なら使用中なので待つ

（脱線）ロックの実現

- 共有変数「ロック中」（使用中）を用意し、
- もし「１」なら使用中なので待つ
- もし「０」なら空いているので
  「１」にして、使い始める
  使い終わったら「０」に戻す

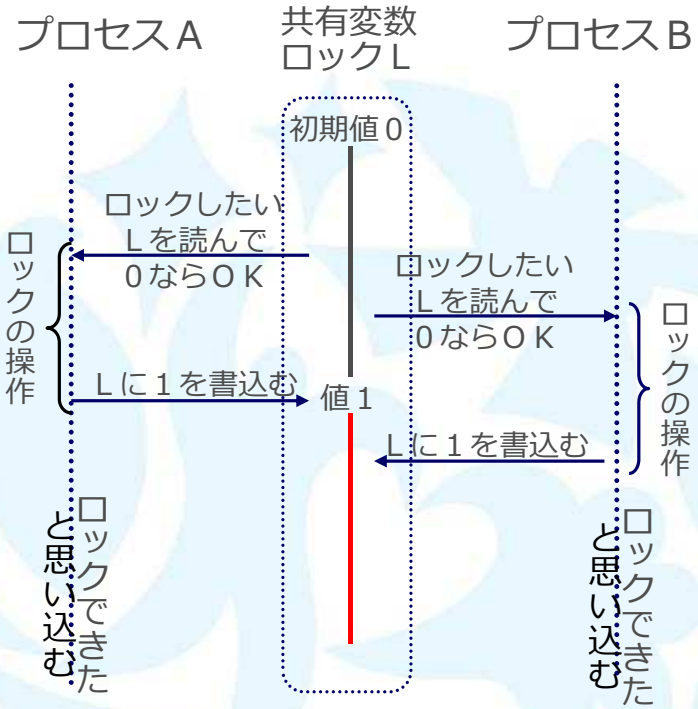
（脱線）ロックの実現

- 共有変数「ロック中」（使用中）を用意し、
- もし「１」なら使用中なので待つ
- もし「０」なら空いているので
  「１」にして、使い始める
  使い終わったら「０」に戻す

- 問題点：
  「ロック中」変数を共有メモリ上に取って
  上の仕組を作ると　⇒　うまくいかない

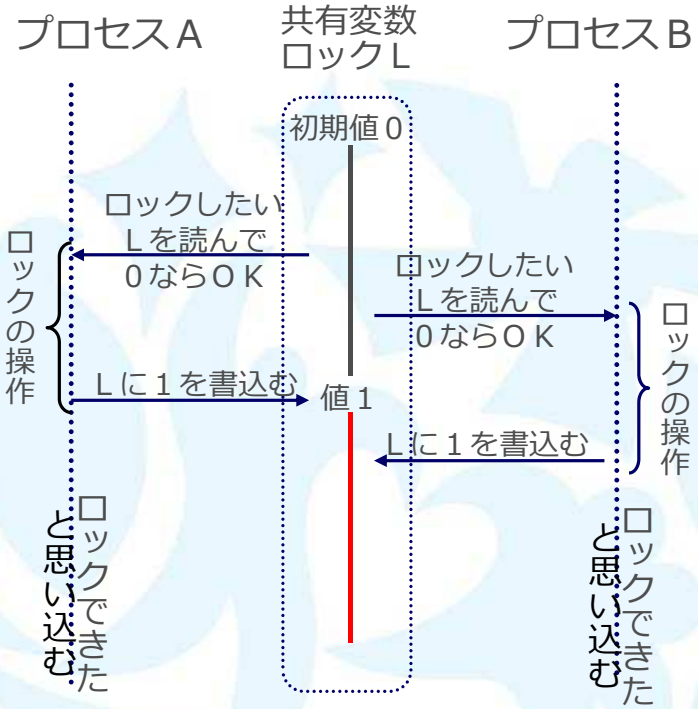
（脱線）ロックの実現

- 共有変数「ロック中」がうまくいかない



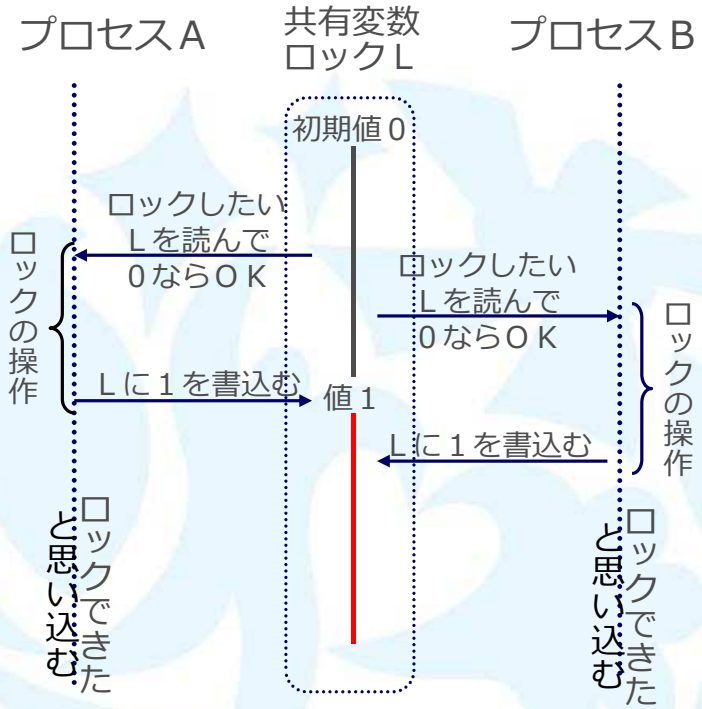
（脱線）ロックの実現

- 共有変数「ロック中」がうまくいかない
- Aが変数Lに 1 を書き込まないうちに、Bが L を読出してまだ空いていると思う



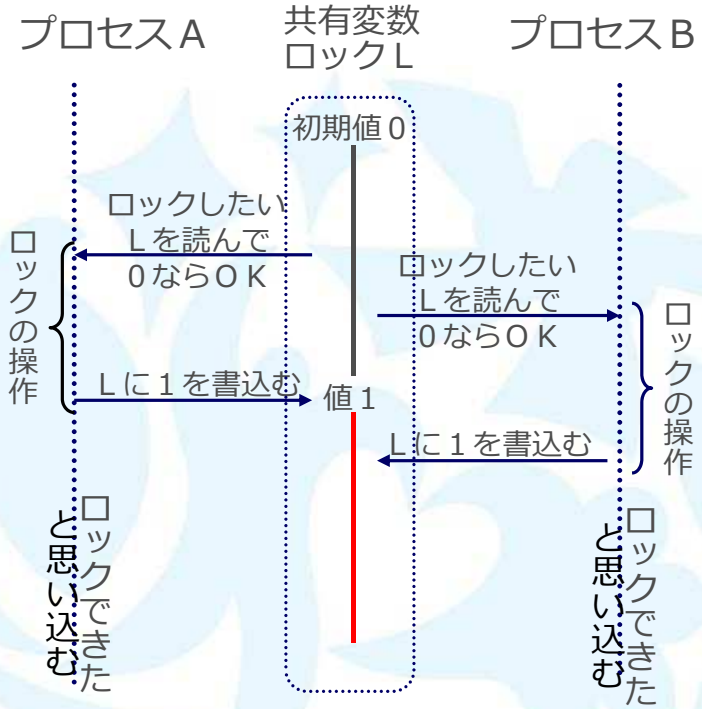
（脱線）ロックの実現

- 共有変数「ロック中」がうまくいかない
- Ａが変数Lに １ を書き込まないうちに、Ｂが Ｌ を読出してまだ空いていると思う
- ＡもＢもロックできたと思い込む



（脱線）ロックの実現

- 共有変数「ロック中」がうまくいかない
- Ａが変数Lに １ を書き込まないうちに、Ｂが Ｌ を読出してまだ空いていると思う
- ＡもＢもロックできたと思い込む

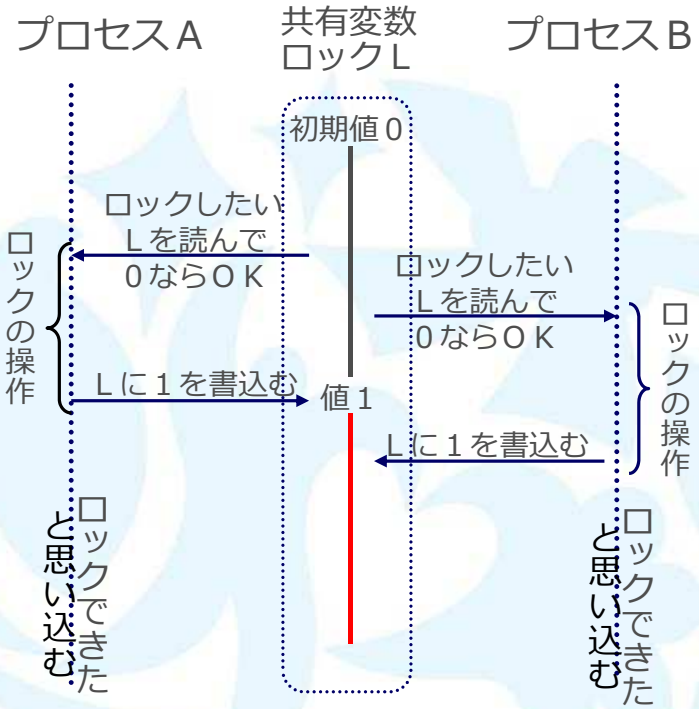


なぜ？ ⇒ Ａが「読んで書く」間に(書く前に)Ｂが読出す

（脱線）ロックの実現～解決策

1. ハードウェアに工夫して
「読む⇒書く」の動作を
不可分（その間他の操作を
できない）にする
例：Test & Set 命令など

2. １CPU上での多プロセス
の場合に限って有効だが、
プロセス切替（割込み）を
起こさせない（禁止）よう
にする
例：割込み禁止による



排他制御の仕組のまとめ

- 共有データ（共有ファイルも）を
「読んで書く」などしたいとき、

排他制御の仕組のまとめ

- 共有データ（共有ファイルも）を
  「読んで書く」などしたいとき、
  その間で他プロセスがアクセスすると
  動作の不統一が起こる

排他制御の仕組のまとめ

- 共有データ（共有ファイルも）を
  「読んで書く」などしたいとき、
  その間で他プロセスがアクセスすると
  動作の不統一が起こる
- それを防ぐには

排他制御の仕組のまとめ

- 共有データ（共有ファイルも）を
  「読んで書く」などしたいとき、
  その間で他プロセスがアクセスすると
  動作の不統一が起こる
- それを防ぐには
  他プロセスが間に入って困る不可分の動作中は

排他制御の仕組のまとめ

- 共有データ（共有ファイルも）を
  「読んで書く」などしたいとき、
  その間で他プロセスがアクセスすると
  動作の不統一が起こる
- それを防ぐには
  他プロセスが間に入って困る不可分の動作中は
  ロックによって排他すればよい

排他制御の必要性と ロックによる排他制御の方法が 理解できましたか？

○

次へ

×