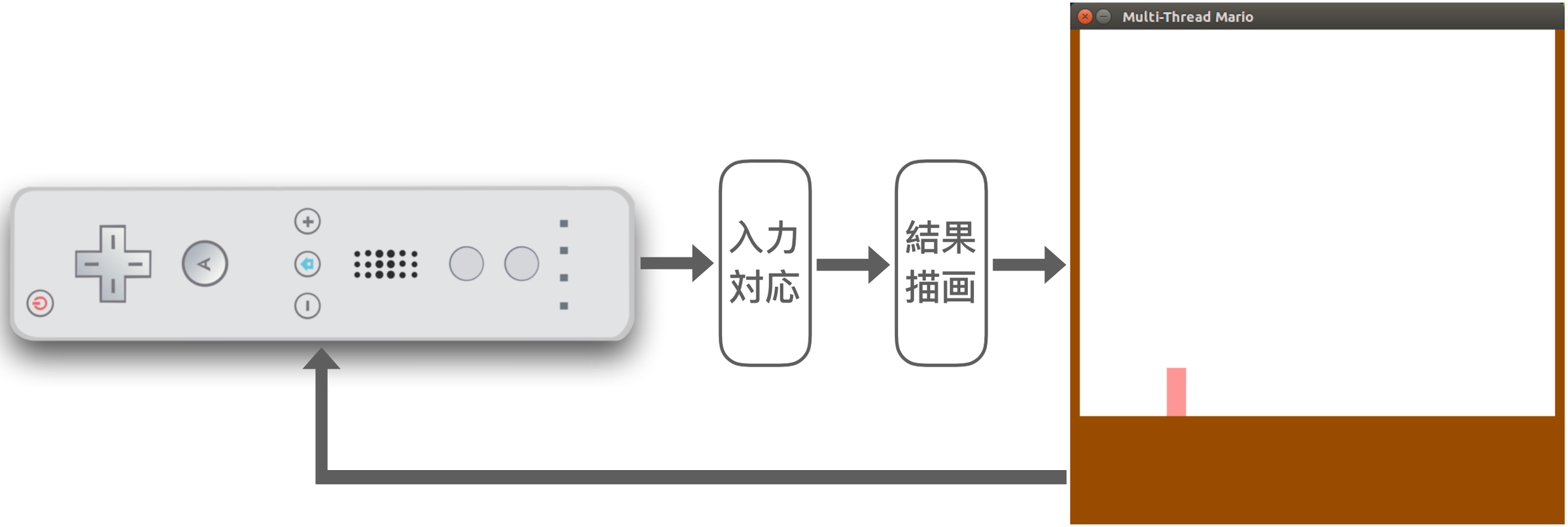
# ソフトウェア設計及び実験

## マルチスレッドプログラミング

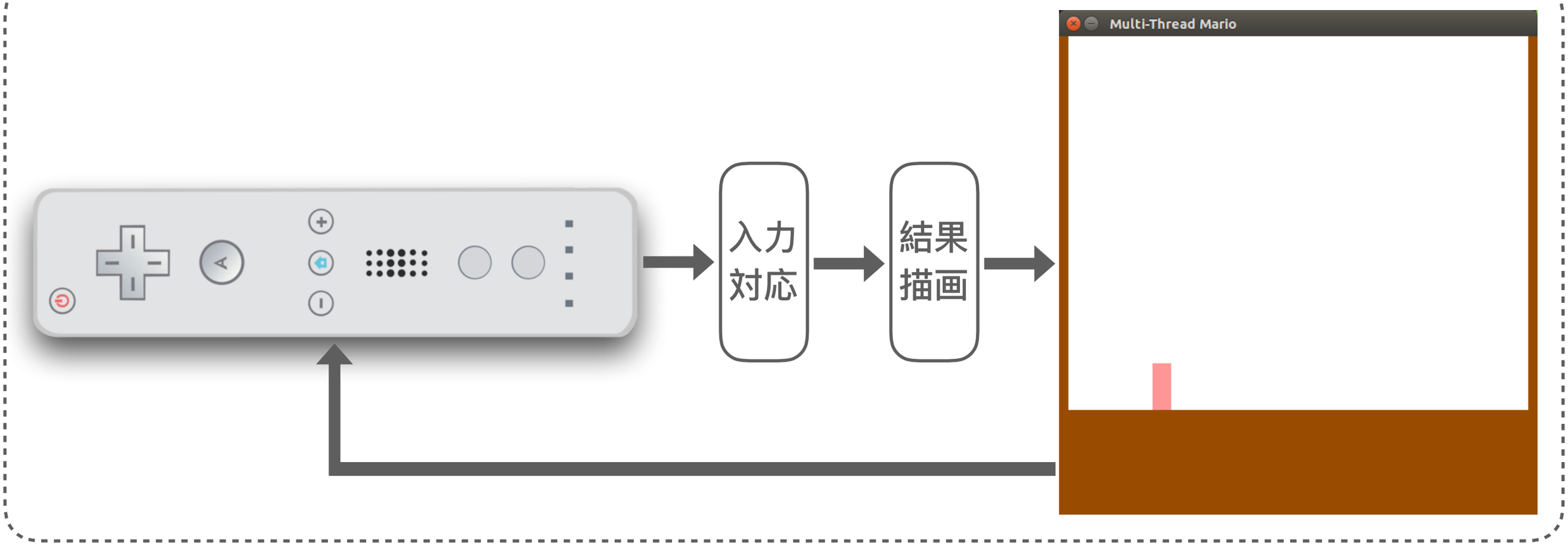
2019年6月4日

担当：康　鑫

# シングルスレッドプログラミング

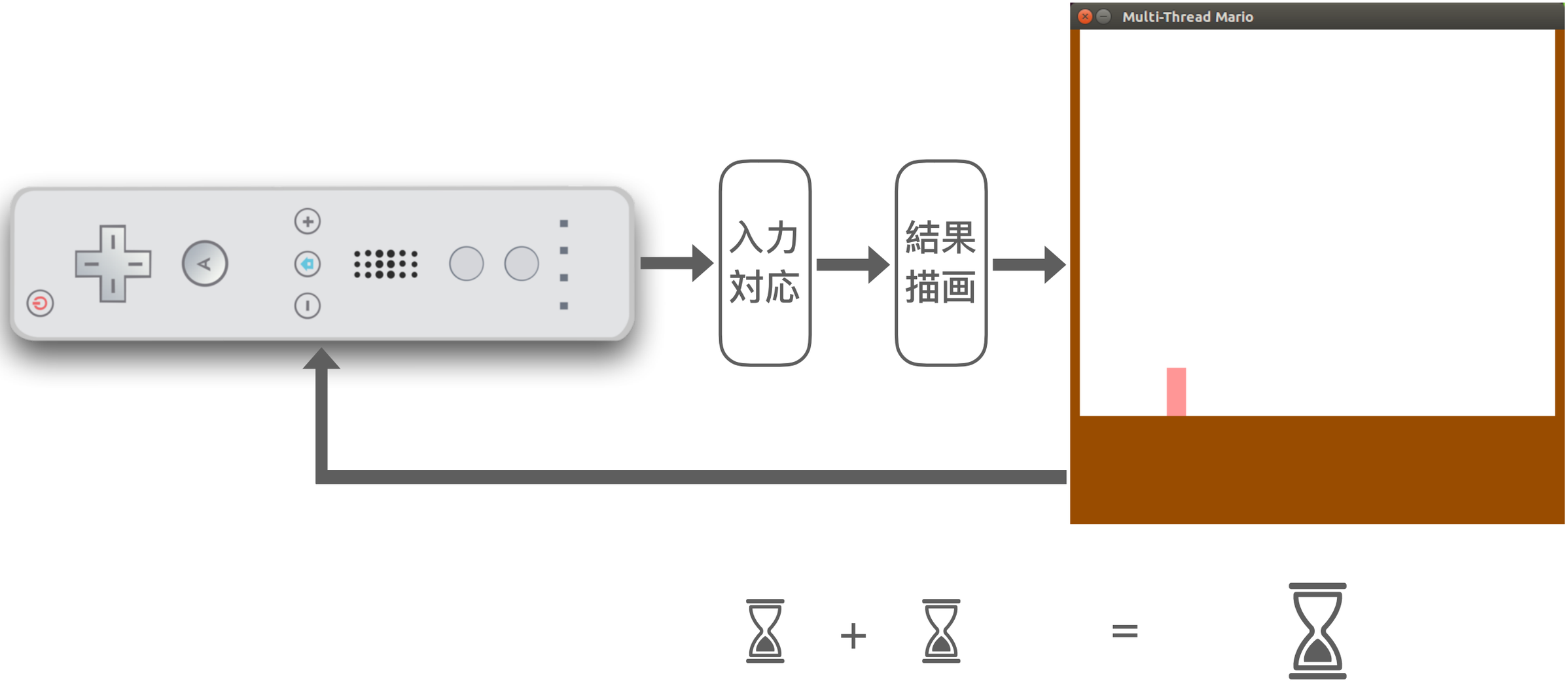


# シングルスレッドプログラミング

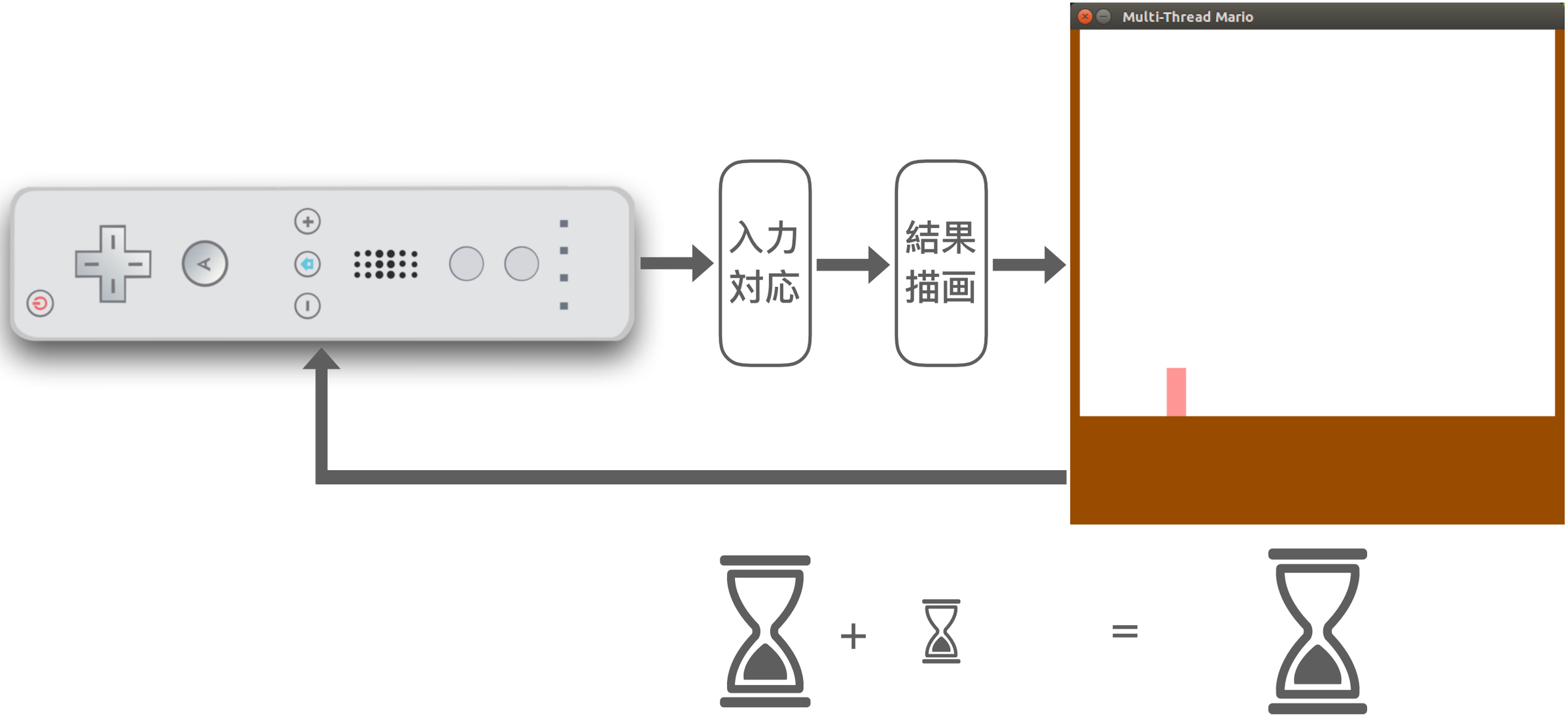



シングルスレッド

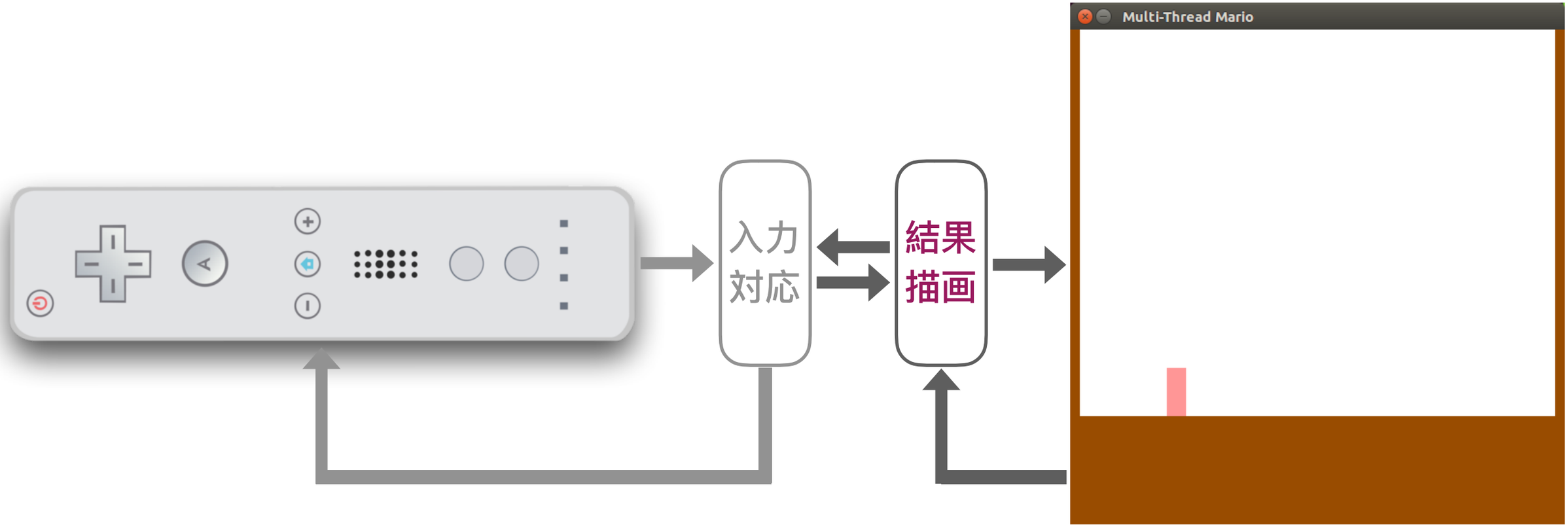
# シングルスレッドプログラミング



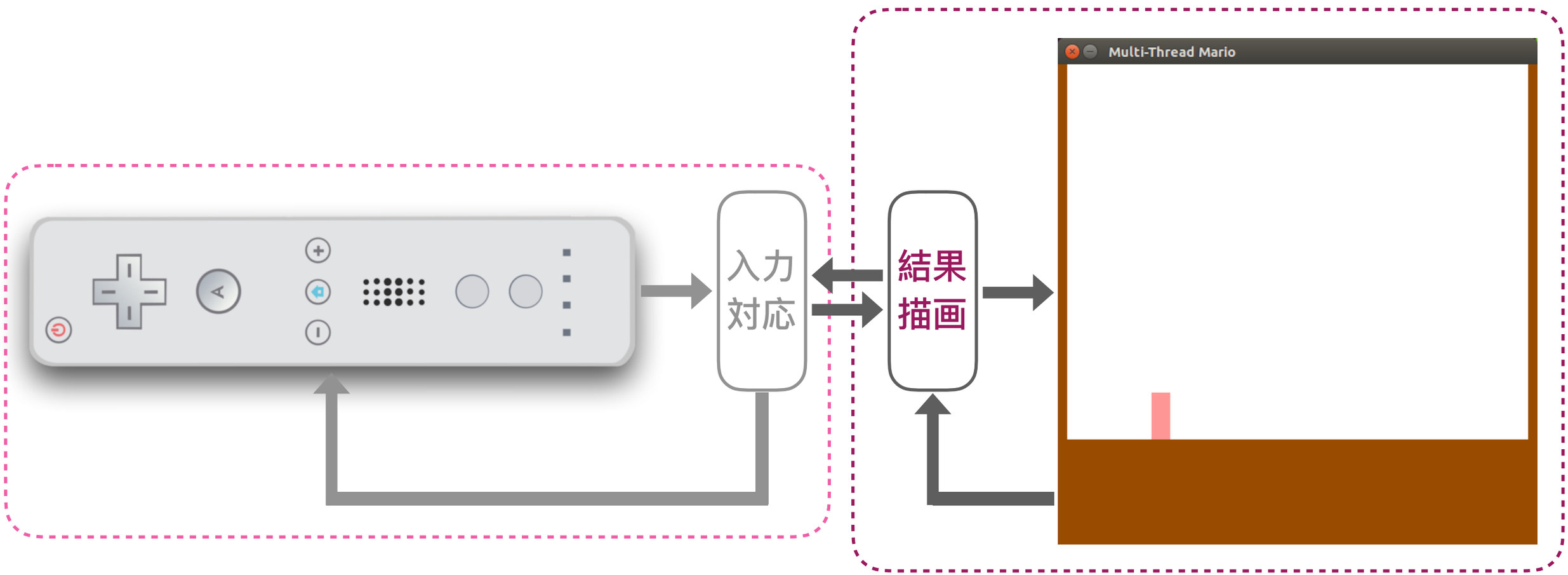
# シングルスレッドプログラミング



# マルチスレッドプログラミング



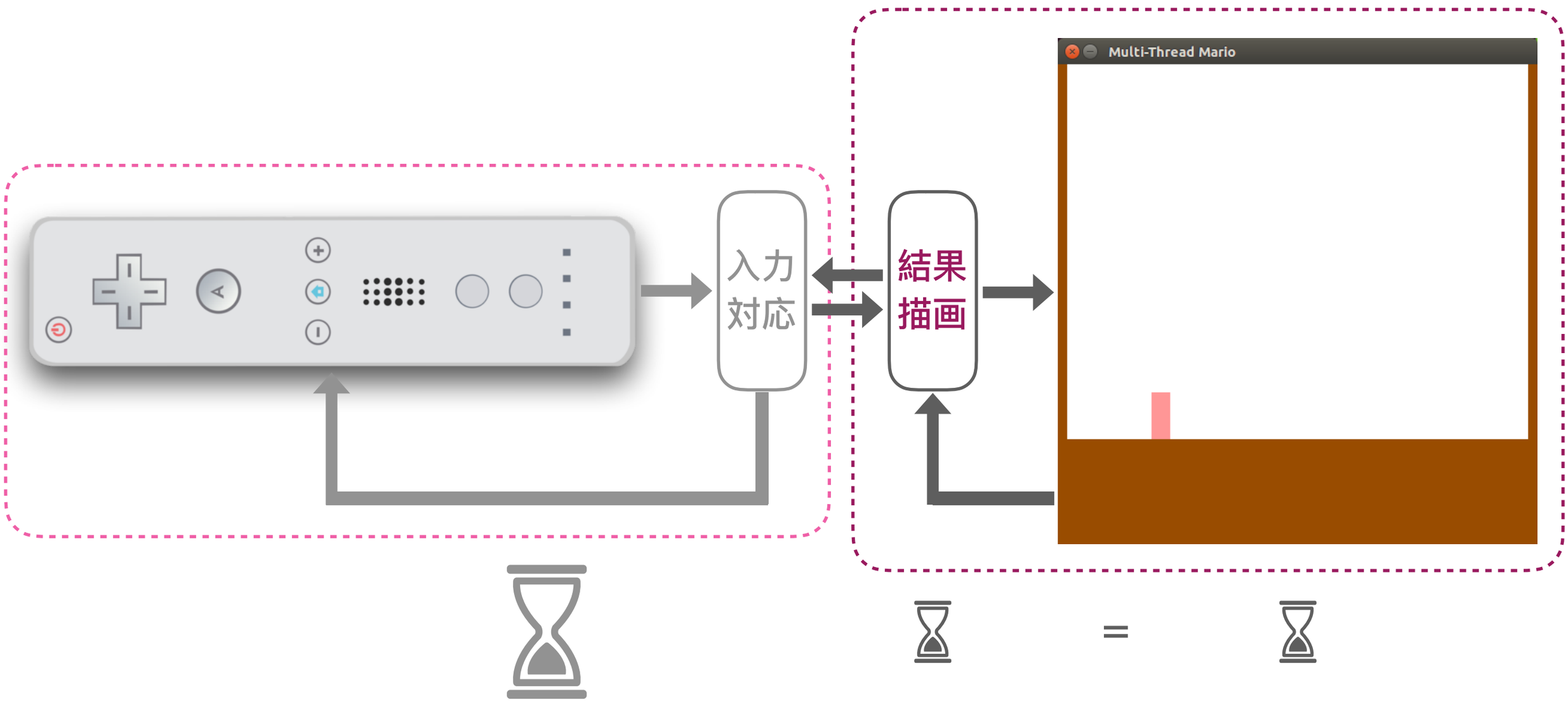
# マルチスレッドプログラミング



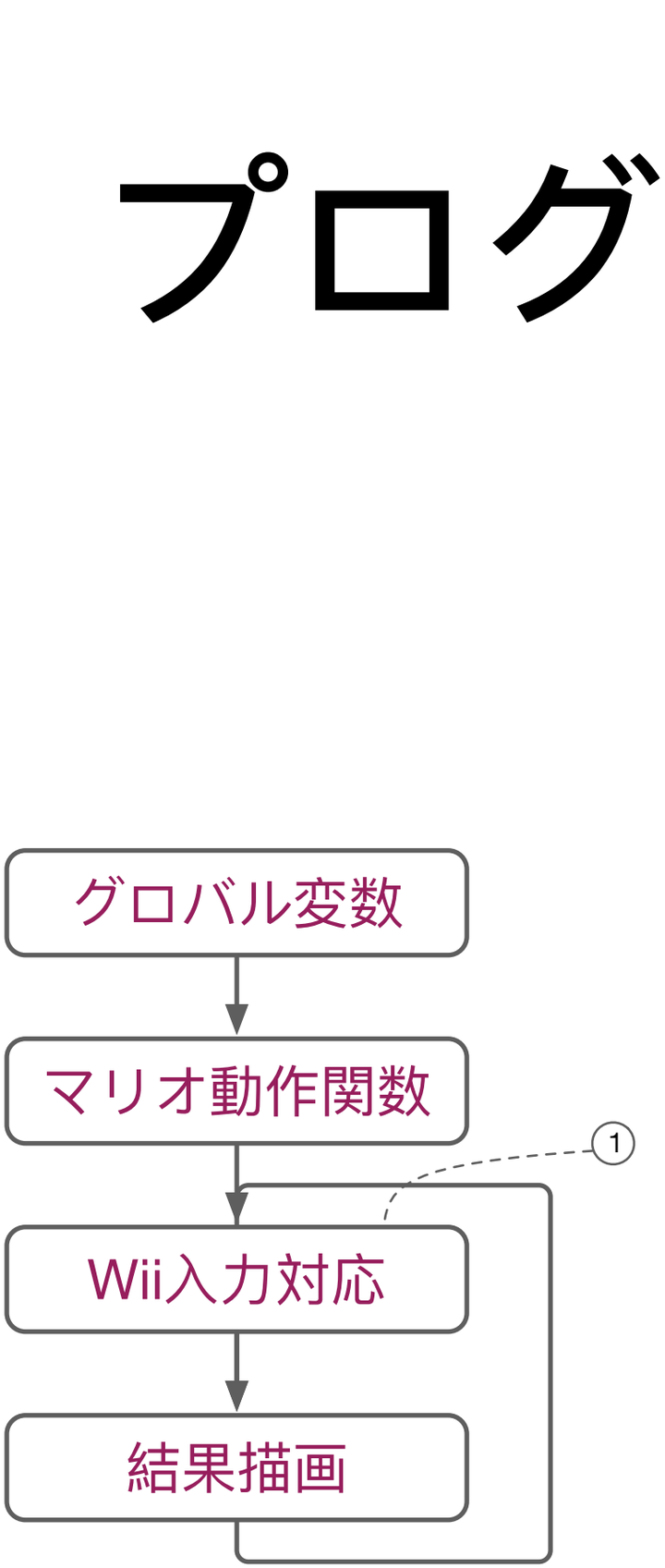

入力対応スレッド

メインスレッド
(元々のシングルスレッド)

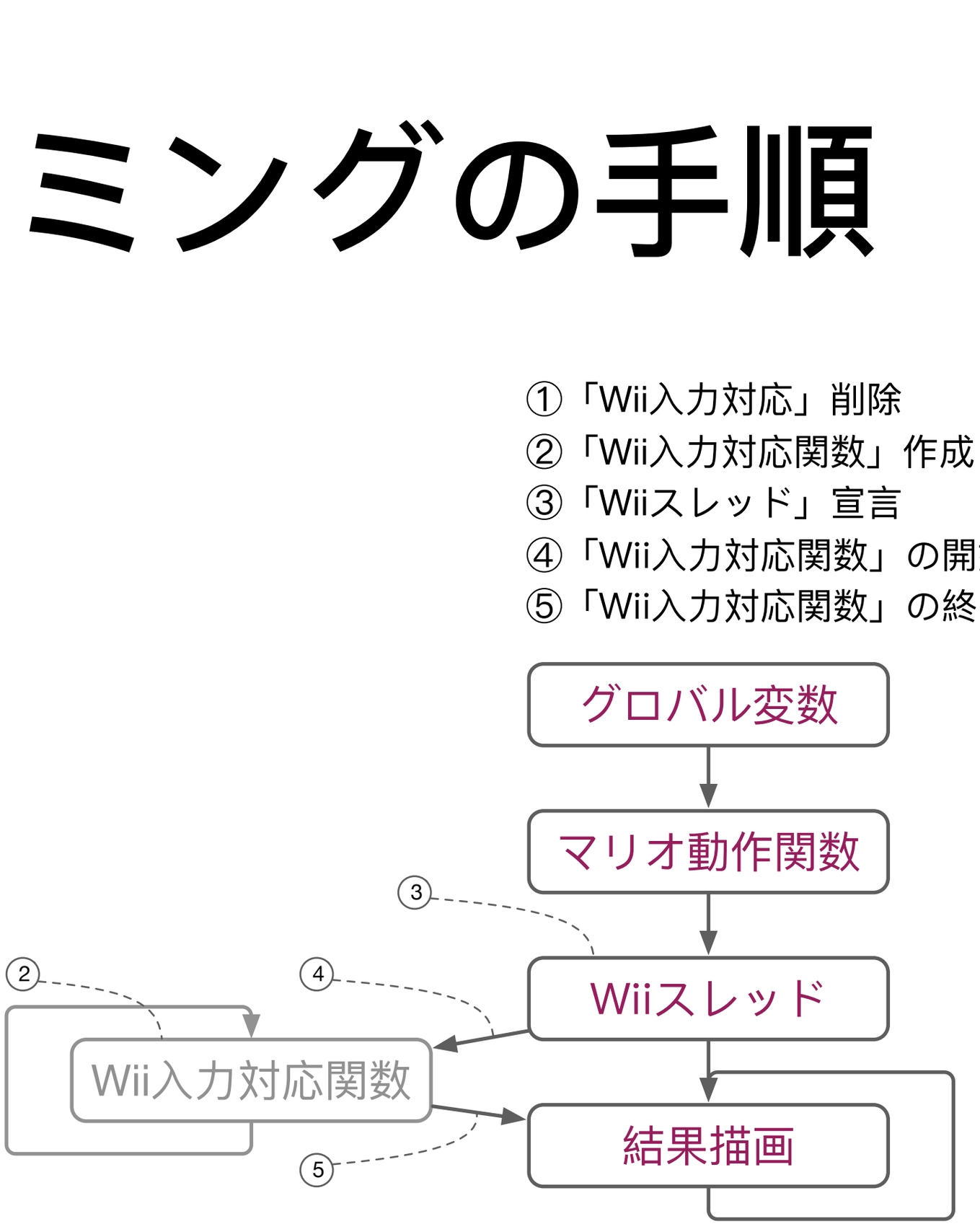
# マルチスレッドプログラミング



# プログラミングの手順

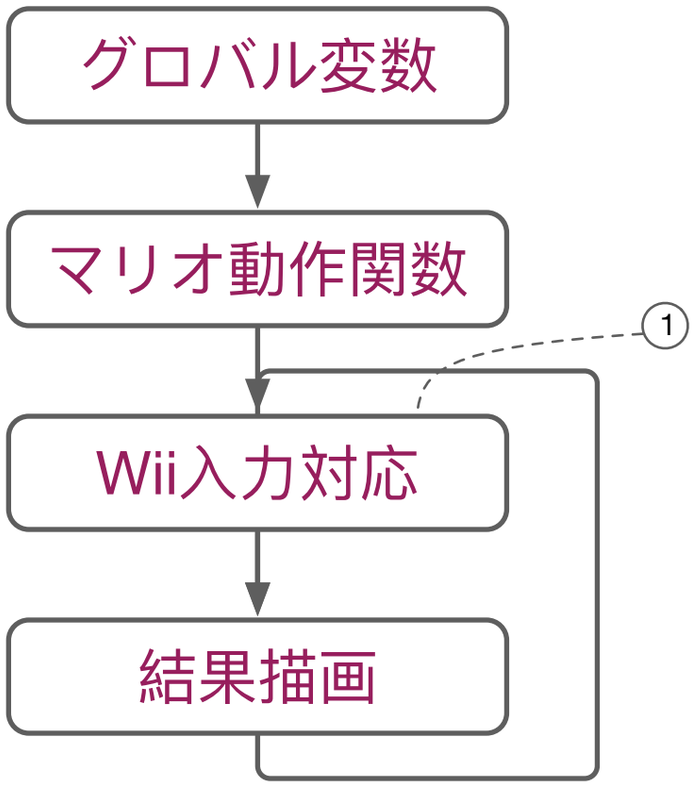



①「Wii入力対応」削除
②「Wii入力対応関数」作成
③「Wiiスレッド」宣言
④「Wii入力対応関数」の開始
⑤「Wii入力対応関数」の終了




**sdl2_thread.c**

# プログラミングの手順




①「Wii入力対応」削除
②「Wii入力対応関数」作成

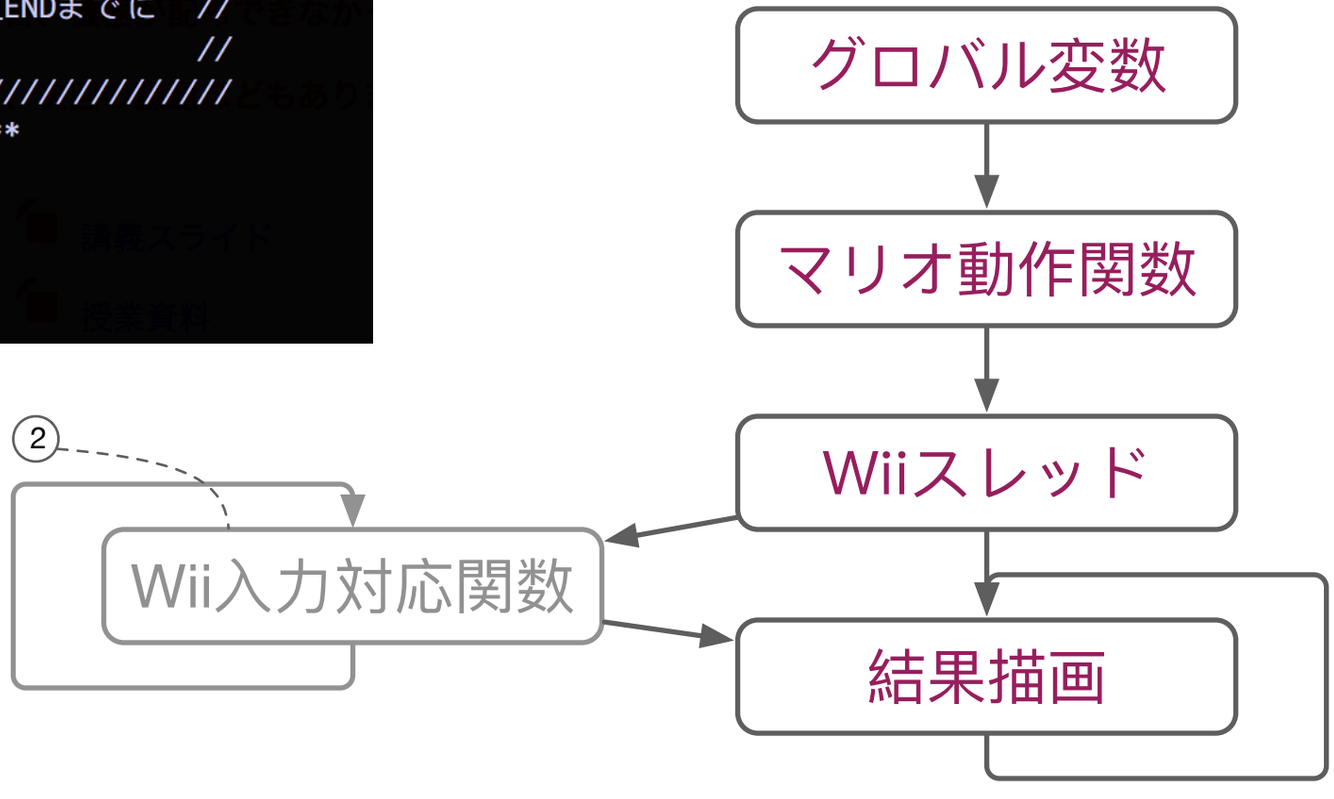
```
////////////////////////////////////////
// STEP. 1                            //
// STEP_1_BEGINからSTEP_1_ENDまでの   //
// コードをStep. 2で指示されたところ  //
// に移動する                         //
////////////////////////////////////////

// ***** STEP_1_BEGIN *****
// Wiiリモコンの状態を取得・更新する
if(wiimote_update(&wiimote))
{
    // ***** Wiiのキー（ボタン）ごとに処理 *****
    // Wii Homeボタンが押された時
    if(wiimote.keys.home)
    {
        wiimote_disconnect(&wiimote); // Wiiリモコンとの接続を解除
    }

    ...

    print_wii(); // Wiiボトン値を表示
}
else
{
    wiimote_disconnect(&wiimote);
}
// ***** STEP_1_END *****
```

# プログラミングの手順

```
// Wiiボタン値によって，マリオの座標を計算
// スレッド実行する関数
int wii_func(void * args)
{
    // Wiiリモコンがオープン（接続状態）であればループ
    while(wiimote_is_open(&wiimote))
    {
        ////////////////////////////////////////
        // STEP. 2                            //
        // STEP_1で指示されたコードを         //
        // STEP_2_BEGINからSTEP_2_ENDまでに   //
        // に移動する                         //
        ////////////////////////////////////////
        // ***** STEP_2_BEGIN *****
        // ***** STEP_2_END *****
    }
    return 0;
}
```

- wii_func: Wii入力対応関数
- args: void *型の引数
  （実際の引数の型に変更できる）
- 返値: 通常は0

① 「Wii入力対応」削除
② 「Wii入力対応関数」作成
③ 「Wiiスレッド」宣言
④ 「Wii入力対応関数」の開始
⑤ 「Wii入力対応関数」の終了



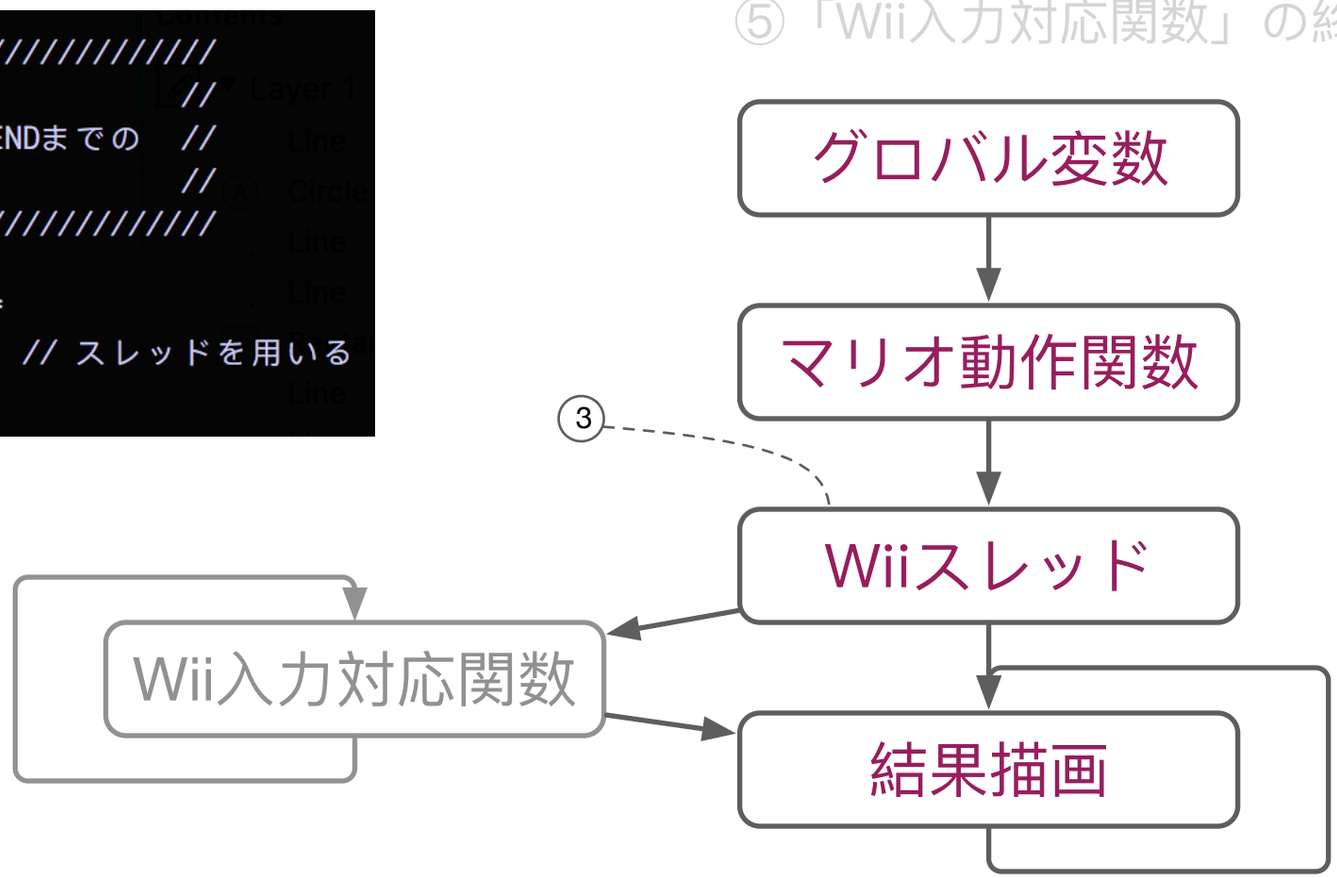

sdl2_thread.c

# プログラミングの手順

SDL_Thread: SDLスレッドの型

```
//////////////////////////////////////
// STEP. 3                          //
// STEP_3_BEGINからSTEP_3_ENDまでの  //
// コードのコメントを外す             //
//////////////////////////////////////

// ***** STEP_3_BEGIN *****
// SDL_Thread * wii_thread; // スレッドを用いる
// ***** STEP_3_END *****
```

① 「Wii入力対応」削除
② 「Wii入力対応関数」作成
③ 「Wiiスレッド」宣言
④ 「Wii入力対応関数」の開始
⑤ 「Wii入力対応関数」の終了



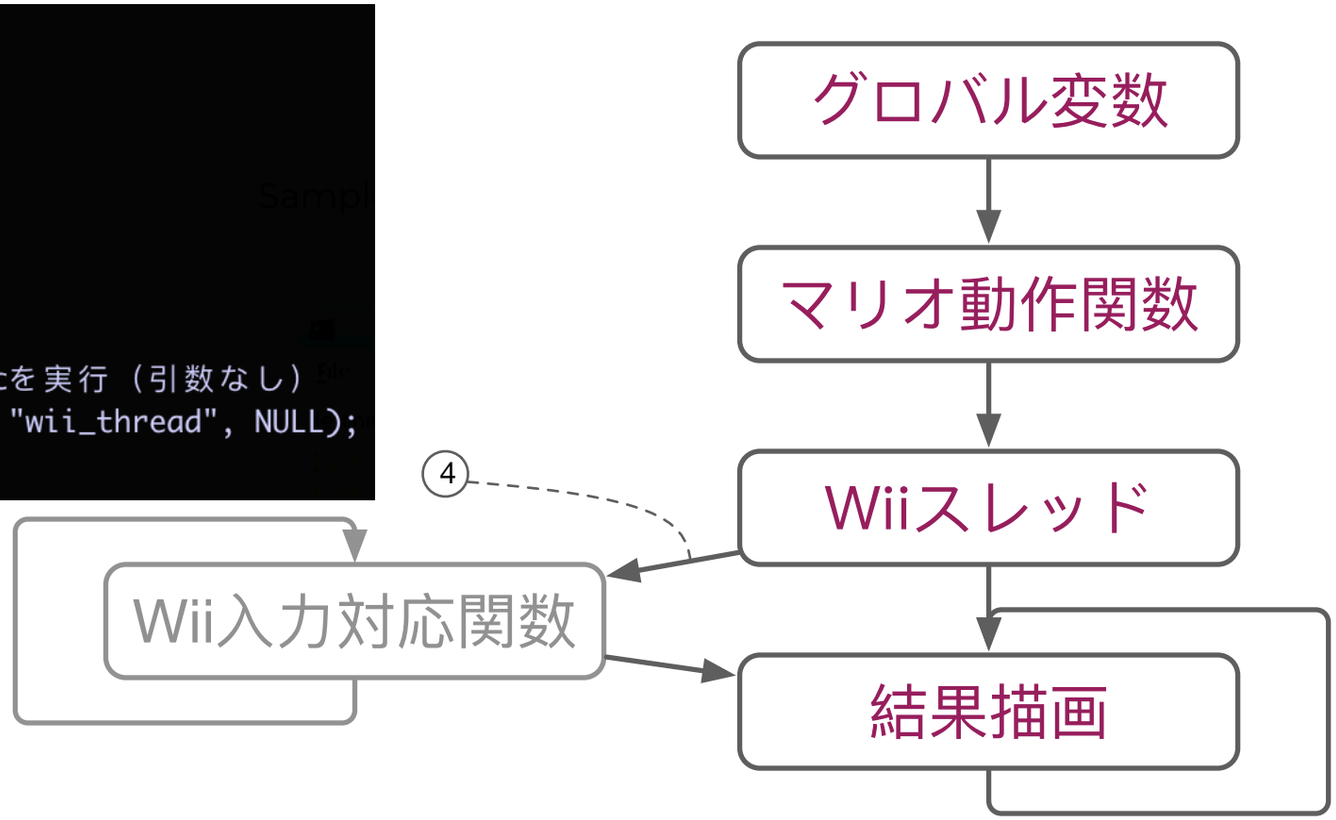

sdl2_thread.c

# プログラミングの手順

- SDL_CreateThread: スレッド作成・実行
- 引数１(wii_func): ②で作成した関数名
- 引数２(“wii_thread”): スレッドの名前
- 引数３(NULL): ②で作成した関数の引数
- 返値(wii_thread): 作成されたスレッド③に

```
////////////////////////////////////////
// STEP. 4                            //
// STEP_4_BEGINからSTEP_4_ENDまでの   //
// コードのコメントを外す             //
////////////////////////////////////////

// ***** STEP_4_BEGIN *****
// スレッドを作成・実行
// wii_threadを作成し、スレッド関数wii_funcを実行（引数なし）
// wii_thread = SDL_CreateThread(wii_func, "wii_thread", NULL);
// ***** STEP_4_END *****
```

① 「Wii入力対応」削除
② 「Wii入力対応関数」作成
③ 「Wiiスレッド」宣言
④ 「Wii入力対応関数」の開始
⑤ 「Wii入力対応関数」の終了



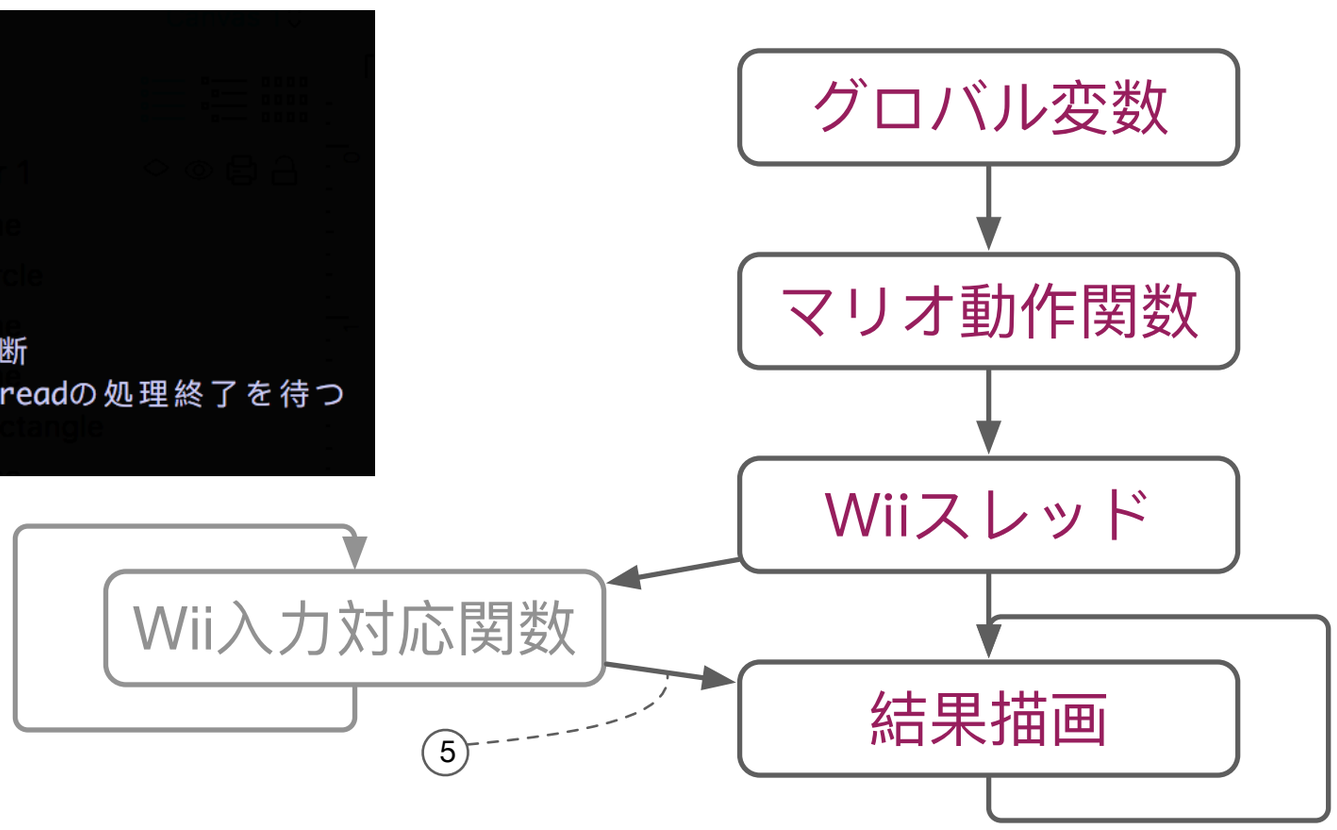

sdl2_thread.c

# プログラミングの手順

- SDL_WaitThread: スレッドが終了するのを待つ
- 引数 1 (wii_thread): ③で宣言されたスレッド
- 引数 2 (NULL): スレッドの状態（通常はNULL）
- 返値: なし

```
////////////////////////////////////
// STEP. 5                        //
// STEP_5_BEGINからSTEP_5_ENDまでの  //
// コードのコメンドを外す             //
////////////////////////////////////

// ***** STEP_5_BEGIN *****
// 各スレッドが終了するまでmain関数の処理を中断
// SDL_WaitThread(wii_thread, NULL);// wii_threadの処理終了を待つ
// ***** STEP_5_END *****
```

① 「Wii入力対応」削除
② 「Wii入力対応関数」作成
③ 「Wiiスレッド」宣言
④ 「Wii入力対応関数」の開始
⑤ 「Wii入力対応関数」の終了




sdl2_thread.c

# プログラミングのまとめ

- 目的：効率向上
- 方法
    - 非効率部分のコードを見付ける
    - スレッド関数に移動
    - スレッド変数を宣言
    - スレッドの開始
    - **メインスレッドで他のコードを実行**
    - スレッドの終了

# 試しに実行してみよう

- コード修正

  「sdl2_thread.c」をStep. 1~5に従って修正

- コンパイル

  「make」で

- 実行

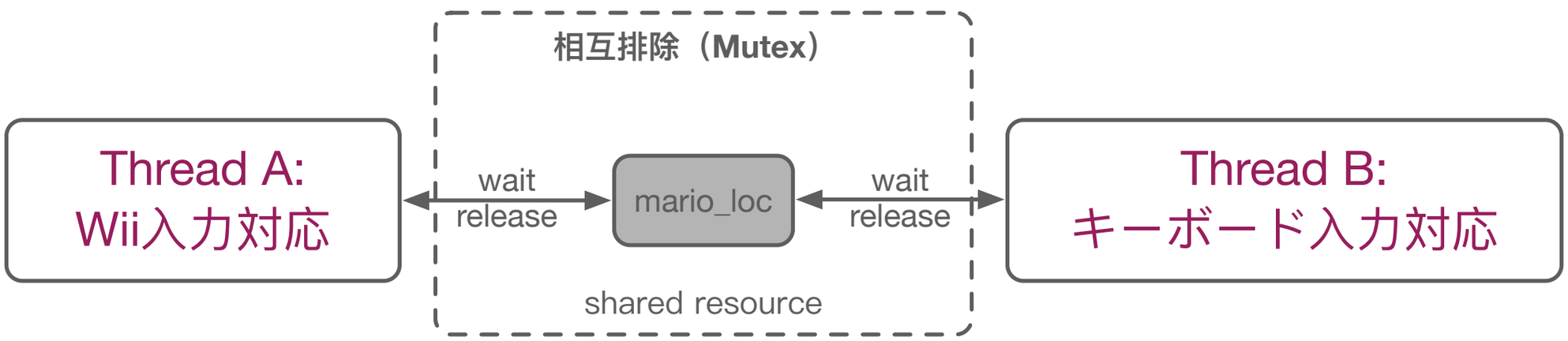
  ./sdl2_thread 識別ID

# 相互排除

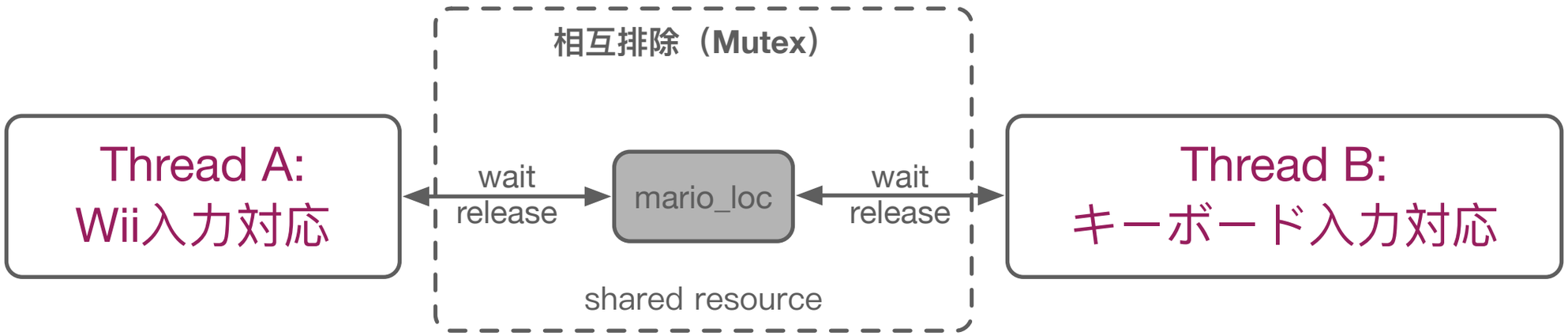
- ２つ以上のスレッドが同時に１つの変数値を変化（❌）
  - 例：Wiiとキーボード２つのデバイスで、マリオをコントロール

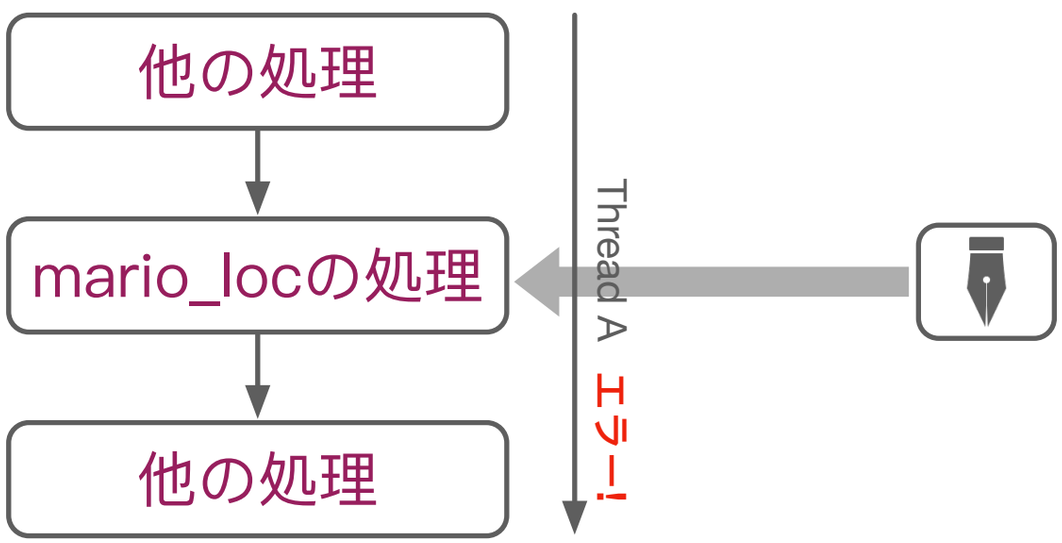
mario_locはThread Aに利用される時、Thread Bが一時中止

mario_locはThread Bに利用される時、Thread Aが一時中止

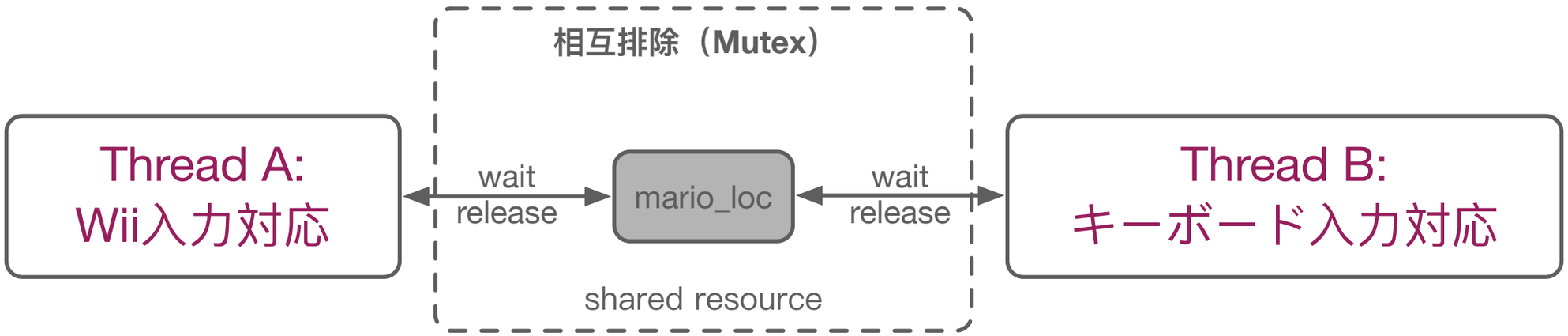


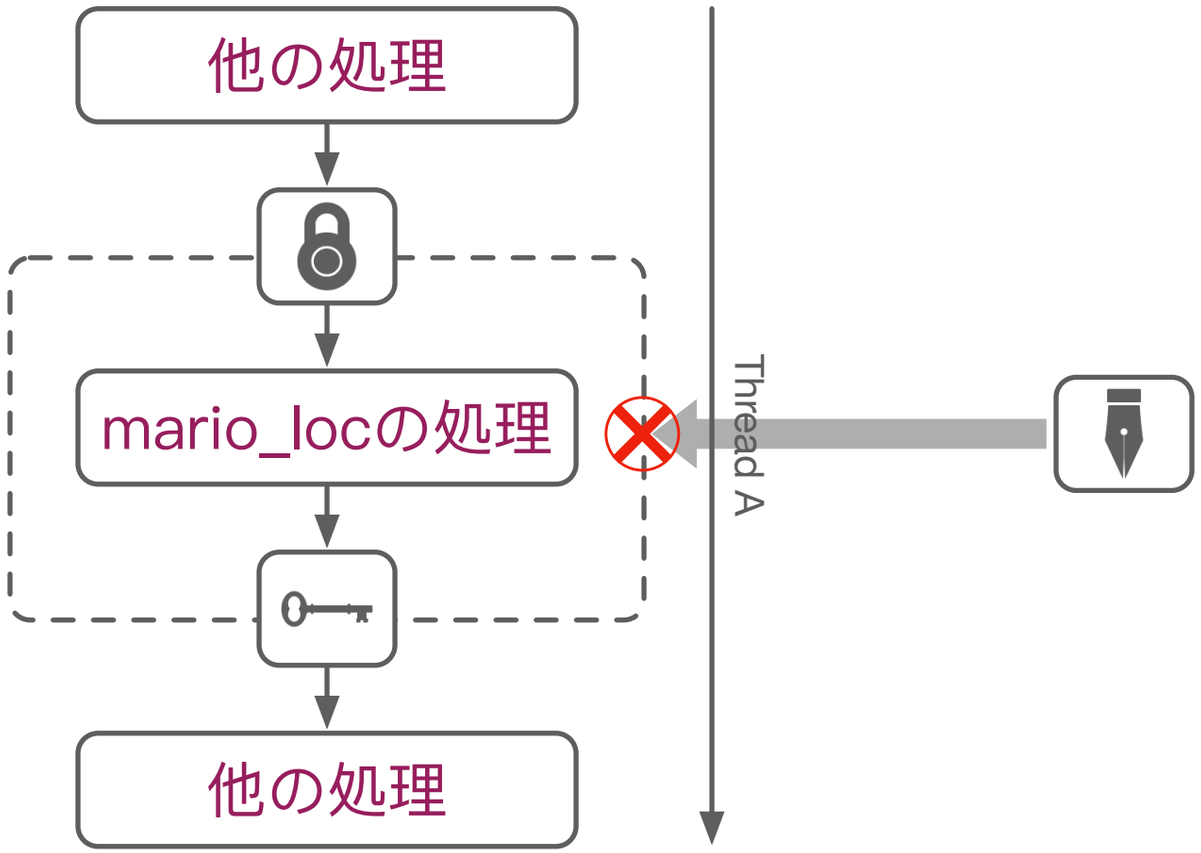
# 相互排除


相互排除（Mutex）
Thread A:
Wii入力対応
wait
release
mario_loc
wait
release
Thread B:
キーボード入力対応
shared resource


他の処理
mario_locの処理
他の処理
Thread A
エラー!

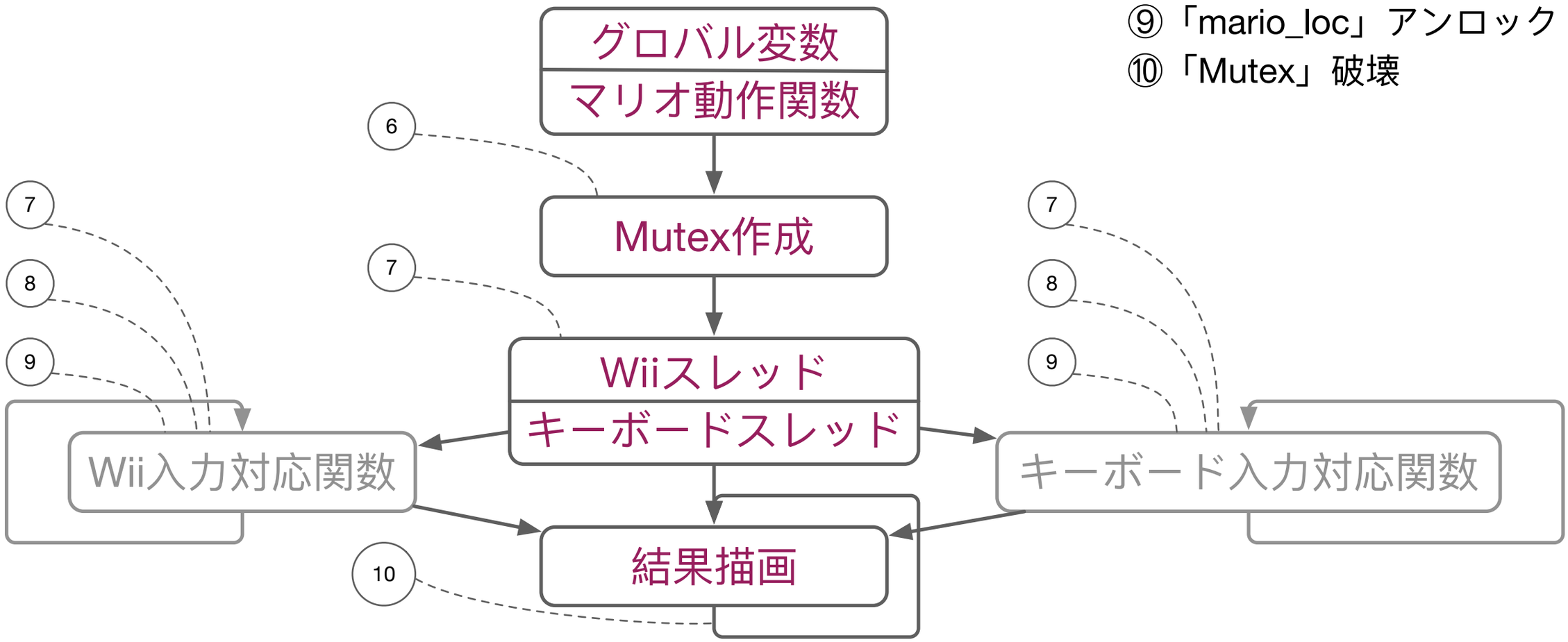
# 相互排除


相互排除（Mutex）
Thread A:
Wii入力対応
wait
release
mario_loc
wait
release
Thread B:
キーボード入力対応
shared resource


他の処理
mario_locの処理
他の処理
Thread A

# プログラミングの手順

⑥「Mutex」作成
⑦「スレッド関数の引数」作成
⑧「mario_loc」ロック
⑨「mario_loc」アンロック
⑩「Mutex」破壊



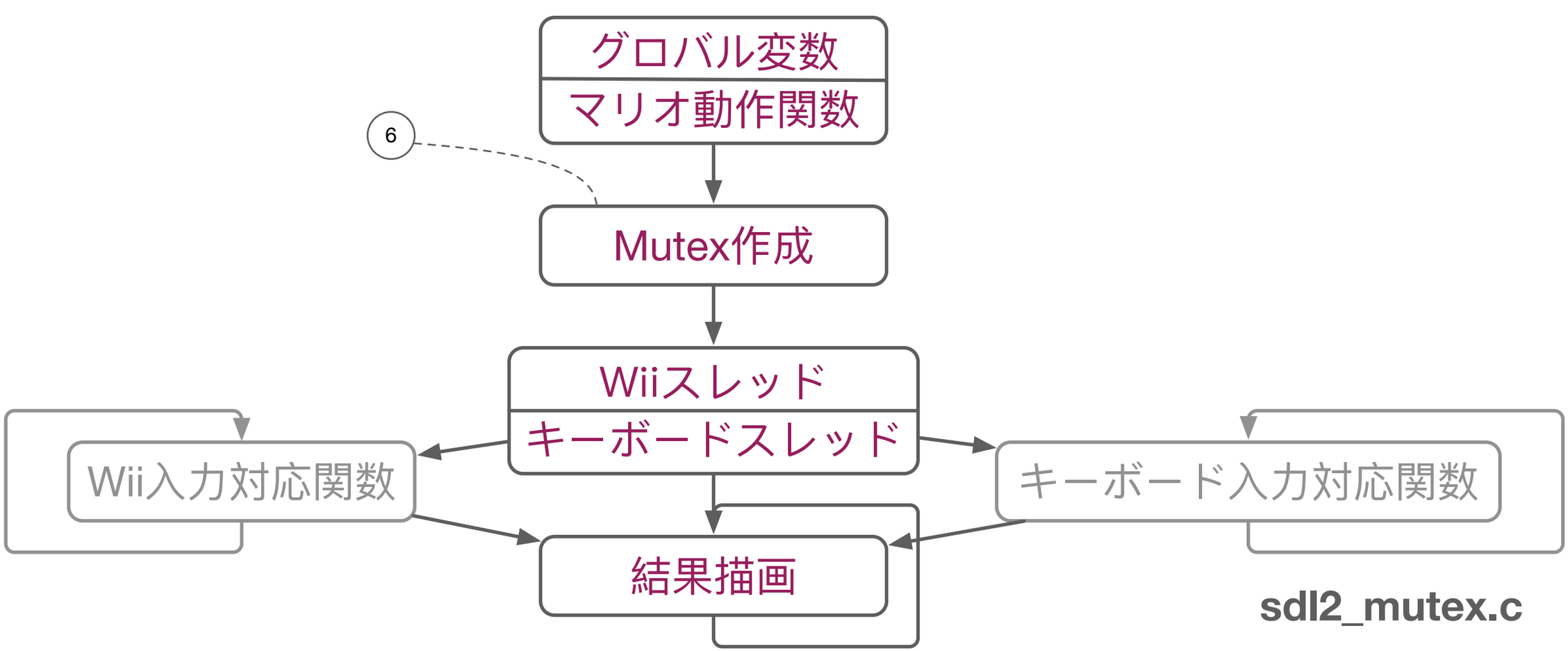

**sdl2_mutex.c**

# プログラミングの手順

```
////////////////////////////////////
// STEP. 6                        //
// STEP_6_BEGINからSTEP_6_ENDまでの  //
// コードのコメンドを外す             //
////////////////////////////////////

// ***** STEP_6_BEGIN *****
// 相互排除（Mutex）あり
// SDL_mutex *mtx = SDL_CreateMutex(); // 相互排除（Mutex）を用いる
// ***** STEP_6_END *****
```

- SDL_Mutex: SDL相互排除の型
- SDL_CreateMutex: Mutex作成

⑥「Mutex」作成
⑦「スレッド関数の引数」作成
ロック
アンロック
破壊




sdl2_mutex.c

# プログラミングの手順

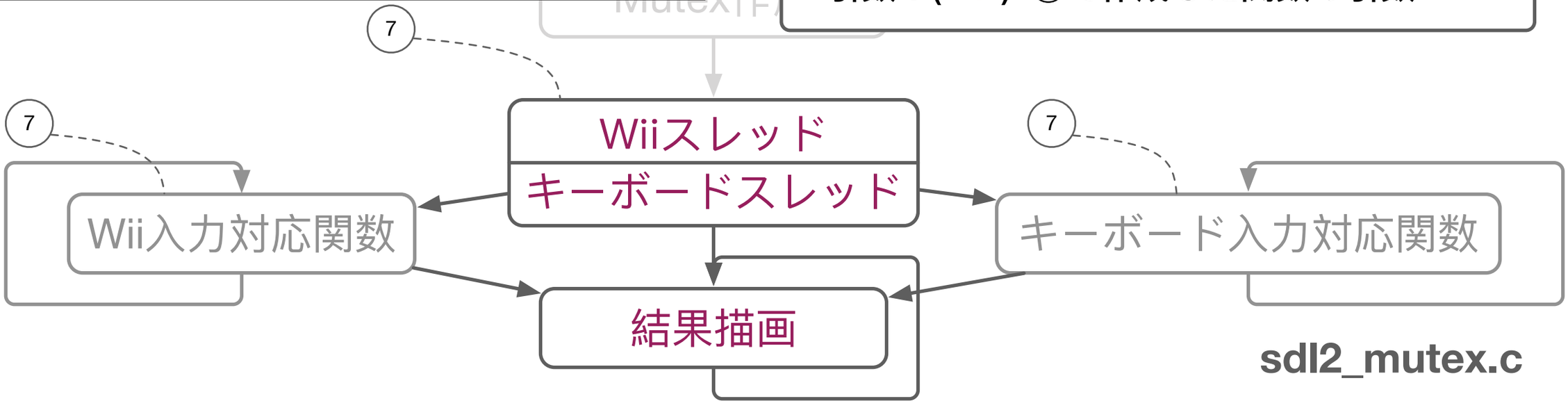
```
// Wiiボタン値によって，マリオの座標を計算
// スレッド実行する関数
int wii_func(void * args)
{
    ////////////////////////////////////
    // STEP. 7                        //
    // STEP_7_BEGINからSTEP_7_ENDまでの  //
    // コードのコメンドを外す             //
    ////////////////////////////////////

    // ***** STEP_7_BEGIN *****
    // SDL_mutex *mtx = (SDL_mutex *)args; // 引数型はmtxに変更
    // ***** STEP_7_END *****

    ...
}
```

```
// wii_threadを作成し、スレッド関数wii_funcを実行（引数mtx）
// wii_thread = SDL_CreateThread(wii_func, "wii_thread", mtx);
```

⑥「Mutex」作成
⑦「スレッド関数の引数」作成
⑧「mario_loc」ロック
loc」アンロック
」破壊

- args: void *型の引数
  （SDL_mutex型に変更）

- SDL_CreateThread: スレッド作成・実行
- 引数 1 (wii_func): ②で作成した関数名
- 引数 2 (“wii_thread”): スレッドの名前
- 引数 3 (mtx): ②で作成した関数の引数



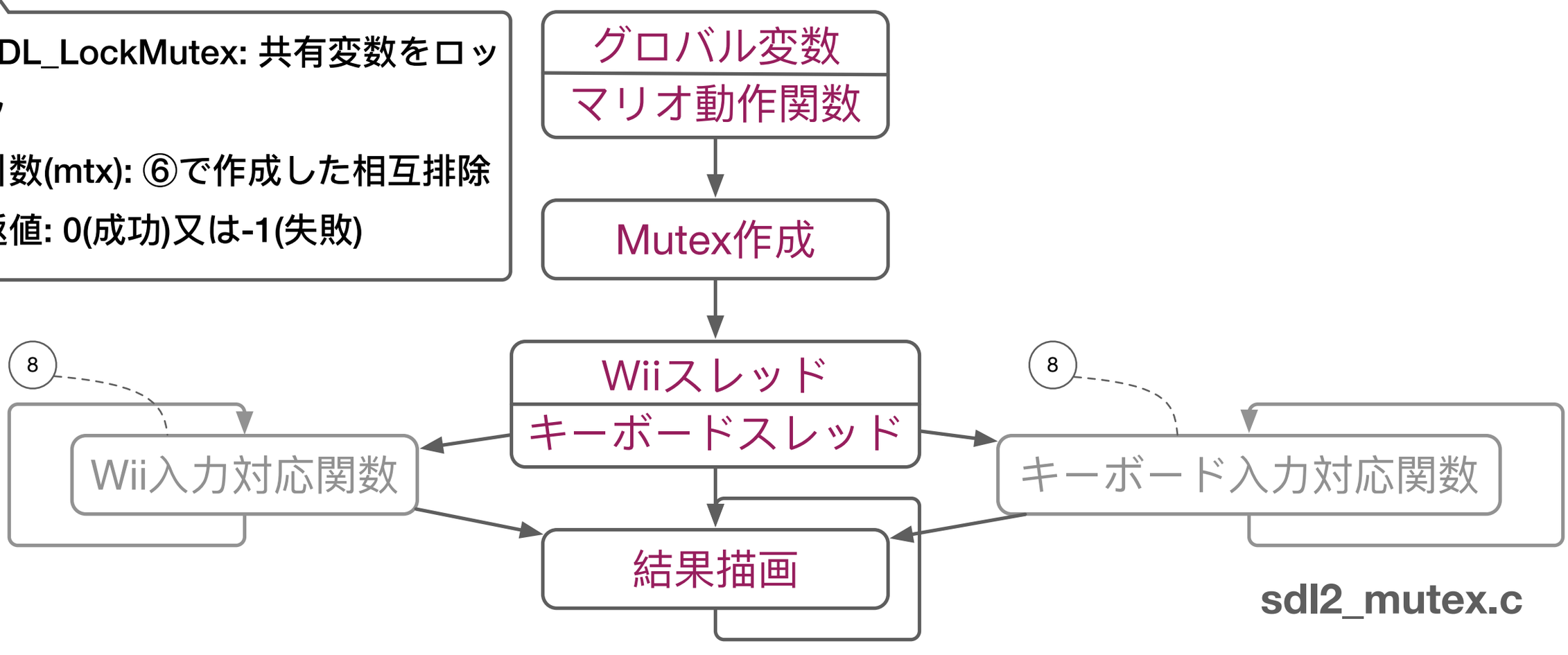

sdl2_mutex.c

# プログラミングの手順

```
//////////////////////////////////////
// STEP. 8                          //
// STEP_8_BEGINからSTEP_8_ENDまでの   //
// コードのコメンドを外す              //
//////////////////////////////////////

// ***** STEP_8_BEGIN *****
// SDL_LockMutex(mtx); // Mutexをロックして、他のスレッドが共有変数にアクセスできないようにする
// ***** STEP_8_END *****
```

- SDL_LockMutex: 共有変数をロック
- 引数(mtx): ⑥で作成した相互排除
- 返値: 0(成功)又は-1(失敗)

⑥「Mutex」作成
⑦「スレッド関数の引数」作成
⑧「mario_loc」ロック
⑨「mario_loc」アンロック
⑩「Mutex」破壊



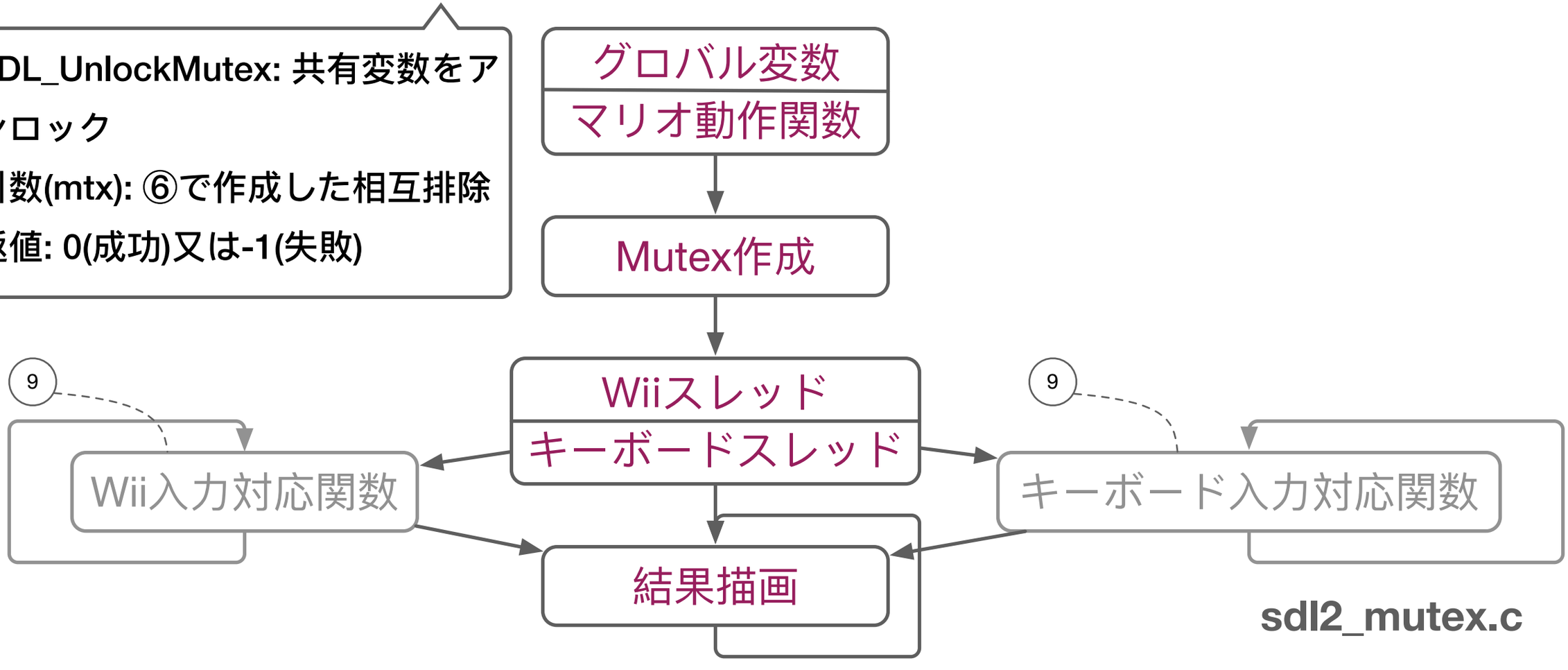

sdl2_mutex.c

# プログラミングの手順

```
////////////////////////////////////
// STEP. 9                        //
// STEP_9_BEGINからSTEP_9_ENDまでの  //
// コードのコメンドを外す             //
////////////////////////////////////

// ***** STEP_9_BEGIN *****
// SDL_UnlockMutex(mtx); // Mutexをアンロックし、他のスレッドが共有変数にアクセスできるようにする
// ***** STEP_9_END *****
```

- SDL_UnlockMutex: 共有変数をアンロック
- 引数(mtx): ⑥で作成した相互排除
- 返値: 0(成功)又は-1(失敗)

⑥「Mutex」作成
⑦「スレッド関数の引数」作成
⑧「mario_loc」ロック
⑨「mario_loc」アンロック
⑩「Mutex」破壊

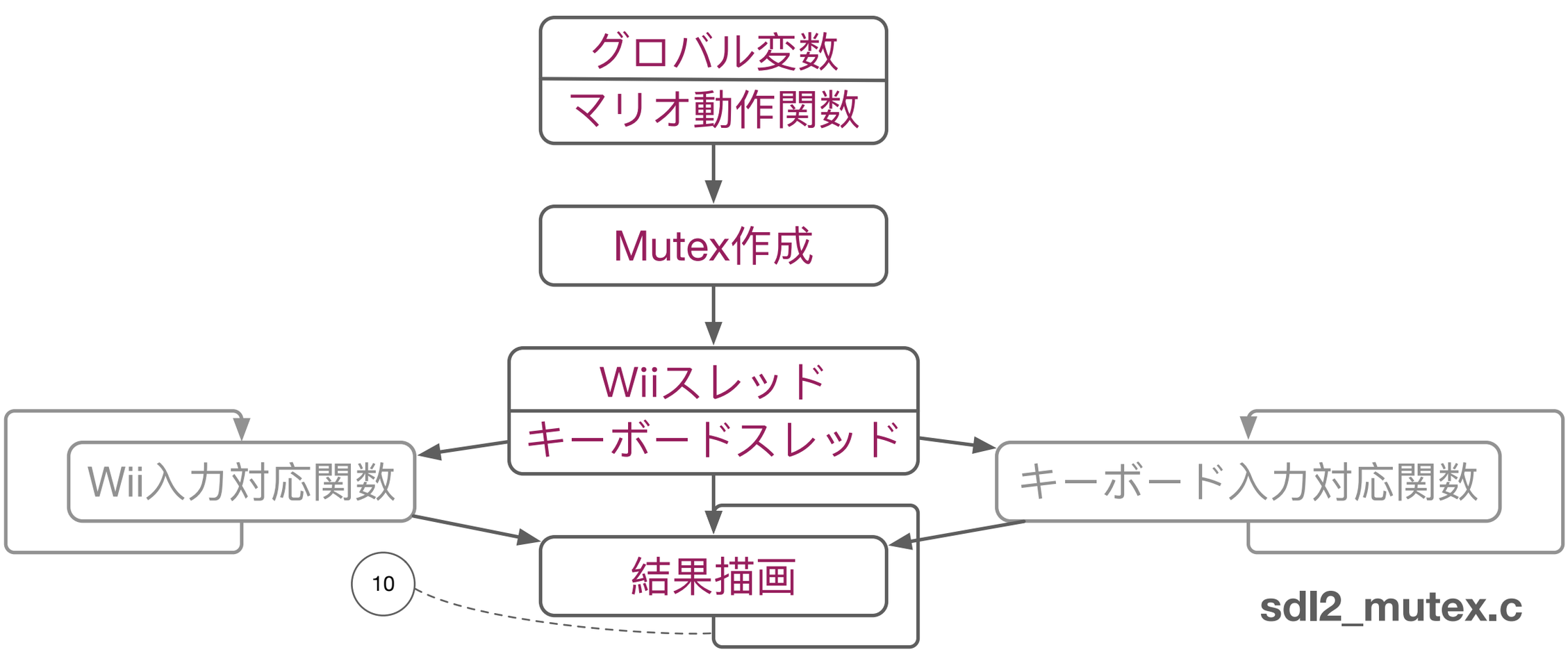



sdl2_mutex.c

# プログラミングの手順

```
////////////////////////////////////
// STEP. 10                       //
// STEP_10_BEGINからSTEP_10_ENDまでの//
// コードのコメントを外す             //
////////////////////////////////////

// ***** STEP_10_BEGIN *****
// SDL_DestroyMutex(mtx); // Mutexを破棄
// ***** STEP_10_END *****
```

⑥「Mutex」作成
⑦「スレッド関数の引数」作成
⑧「mario_loc」ロック
⑨「mario_loc」アンロック
⑩「Mutex」破壊






**sdl2_mutex.c**

# 相互排除のまとめ

- 目的：スレッドで共有変数を独占する為に
- 方法
    - 相互排除変数Mutex作成
    - スレッド関数にMutexを渡す
    - 共有変数をロック
    - **共有変数の処理**
    - 共有変数をアンロック
    - Mutexを破壊

# 課題

マリオゲームのマルチスレッド化：

1. ソースコード「sdl2_mutex.c」
2. makefileの修正「sdl2_thread」→「sdl2_mutex」
3. Step. 1~5に従って「Wii入力対応」スレッドを作成
4. Step. 1~5を参照して「キーボード入力対応」スレッドを作成
5. Step. 6~10に従って「mario_loc処理」相互排除を作成

提出物（学籍番号.zip）：

1. makefile
2. sdl2_mutex.c