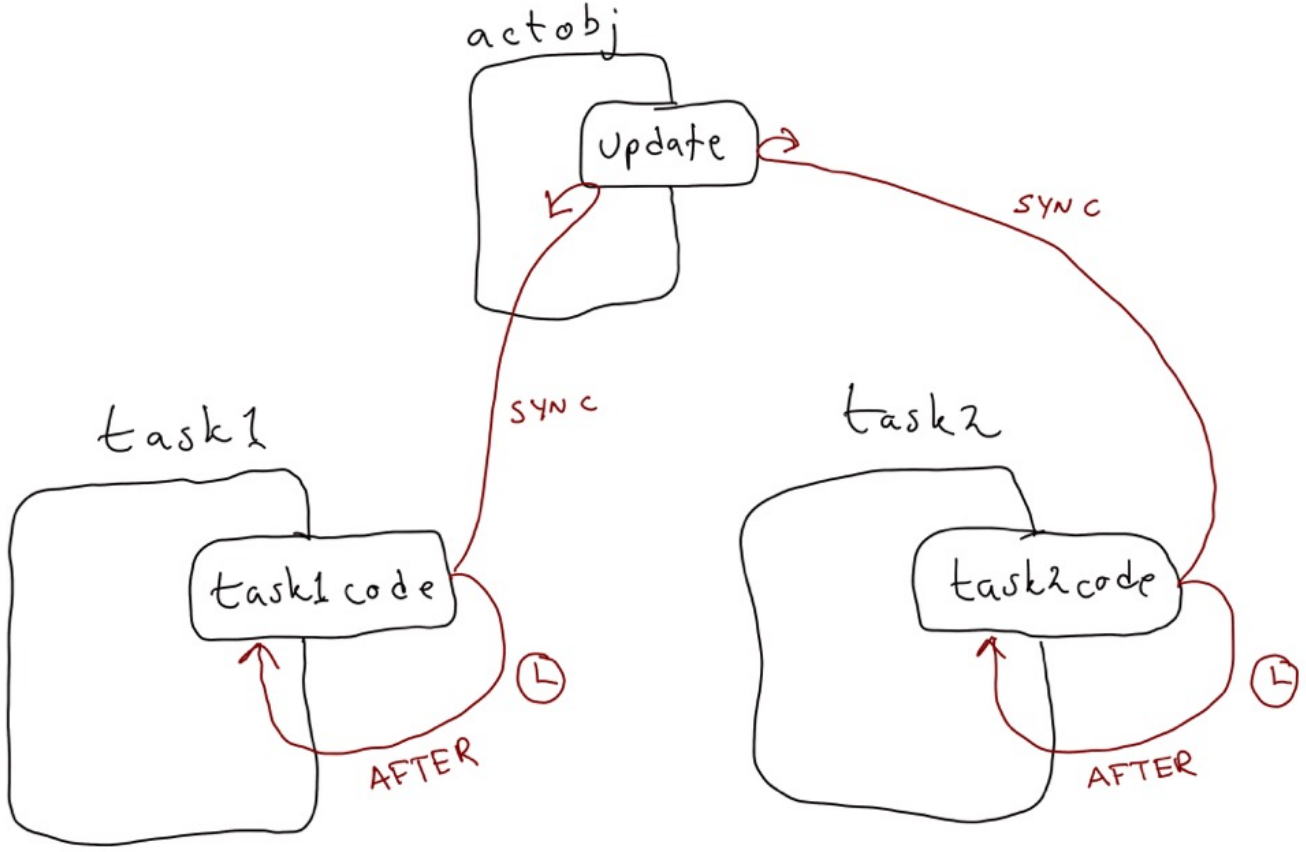
# Exercise #3 – blackboard scribble



# Exercise #3 – blackboard scribble

```
typedef struct {// Class definition
    Object super;
    Time period;

} TaskObject;

// Method declarations
void task1code(TaskObject *, int);
void task2code(TaskObject *, int);

// Initialization macro
#define initTaskObject( P      ) { initObject(), P            }

// Create two objects of type TaskObject
TaskObject task1 = initTaskObject( USEC(300)                  );
TaskObject task2 = initTaskObject(USEC(500)                   );
```

# Exercise #3 – blackboard scribble

```
// Each task sends a new value to method actobj, and uses
// the old value returned from method actobj
void task1code(TaskObject *self, int value){

   int old_state = SYNC(&actobj, update, value);
   ...       // do something with the old value
   AFTER (self->period, self, task1code, value);

}

void task2code(TaskObject *self, int value){

   int old_state = SYNC(&actobj, update, value);
   ...       // do something else with the old value
   AFTER (self->period, self, task2code, value);

}
```

# Exercise #3 – blackboard scribble

```
// How to begin the initial invocation?

void kickoff(TaskObject *self , int unused) {
   // Give an initial value of 10 for task1, and 20 for task2
   ASYNC(&task1, task1code, 10);
   ASYNC(&task2, task2code, 20);




}

int main() {
   TINYTIMBER(&task1, kickoff, 0);
   return 0;
}
```

# Exercise #3 – blackboard scribble

```
typedef struct {// Class definition
    Object super;
    Time period;
    Time deadline;

} TaskObject;

// Method declarations
void task1code(TaskObject *, int);
void task2code(TaskObject *, int);

// Initialization macro
#define initTaskObject( p, d  ) { initObject(), p, d        }

// Create two objects of type TaskObject
TaskObject task1 = initTaskObject( USEC(300), USEC(100)     );
TaskObject task2 = initTaskObject(USEC(500), USEC(150)      );
```

# Exercise #3 – blackboard scribble

```
// Each task sends a new value to method actobj, and uses
// the old value returned from method actobj
void task1code(TaskObject *self, int value){

   int old_state = SYNC(&actobj, update, value);
   ...       // do something with the old value

   SEND(self→period, self→deadline, self, task1code, value);

}

void task2code(TaskObject *self, int value){

   int old_state = SYNC(&actobj, update, value);
   ...       // do something else with the old value

   SEND(self→period, self→deadline, self, task2code, value);

}
```

# Exercise #3 – blackboard scribble

```
// How to begin the initial invocation?

void kickoff(TaskObject *self , int unused) {
   // Give an initial value of 10 for task1, and 20 for task2
   BEFORE(USEC(100), &task1, task1code, 10);
   BEFORE(USEC(150), &task2, task2code, 20);




}

int main() {
   TINYTIMBER(&task1, kickoff, 0);
   return 0;
}
```

# Exercise #3 – blackboard scribble

```
typedef struct {// Class definition
    Object super;
    Time period;
    Time deadline;
    int running;
} TaskObject;

// Method declarations
void task1code(TaskObject *, int);
void task2code(TaskObject *, int);

// Initialization macro
#define initTaskObject( P, d ) { initObject(), P, d, 1 }

// Create two objects of type TaskObject
TaskObject task1 = initTaskObject( USEC(300), USEC(100) );
TaskObject task2 = initTaskObject(USEC(500), USEC(150) );
```

# Exercise #3 – blackboard scribble

```
// Each task sends a new value to method actobj, and uses
// the old value returned from method actobj
void task1code(TaskObject *self, int value){
 if (self→running) {
   int old_state = SYNC(&actobj, update, value);
   ...       // do something with the old value

   SEND(self→period, self→deadline, self, task1code, value);
 }
}

void task2code(TaskObject *self, int value){
 if (self→running) {
   int old_state = SYNC(&actobj, update, value);
   ...       // do something else with the old value

   SEND(self→period, self→deadline, self, task2code, value);
 }
}
```

# Exercise #3 – blackboard scribble

```
// How to begin the initial invocation?

void kickoff(TaskObject *self , int unused) {
   // Give an initial value of 10 for task1, and 20 for task2
   BEFORE(USEC(100), &task1, task1code, 10);
   BEFORE(USEC(150), &task2, task2code, 20);
   AFTER(MSEC(100), &task1, stop, 0);
   AFTER(MSEC(200), &task2, stop, 0);
}

int main() {
   TINYTIMBER(&task1, kickoff, 0);
   return 0;
}
```

# Exercise #3 – blackboard scribble

```
// Method which stops the periodic task

void stop (Taskobject *self, int unused) {
    self->running = 0;
}
```