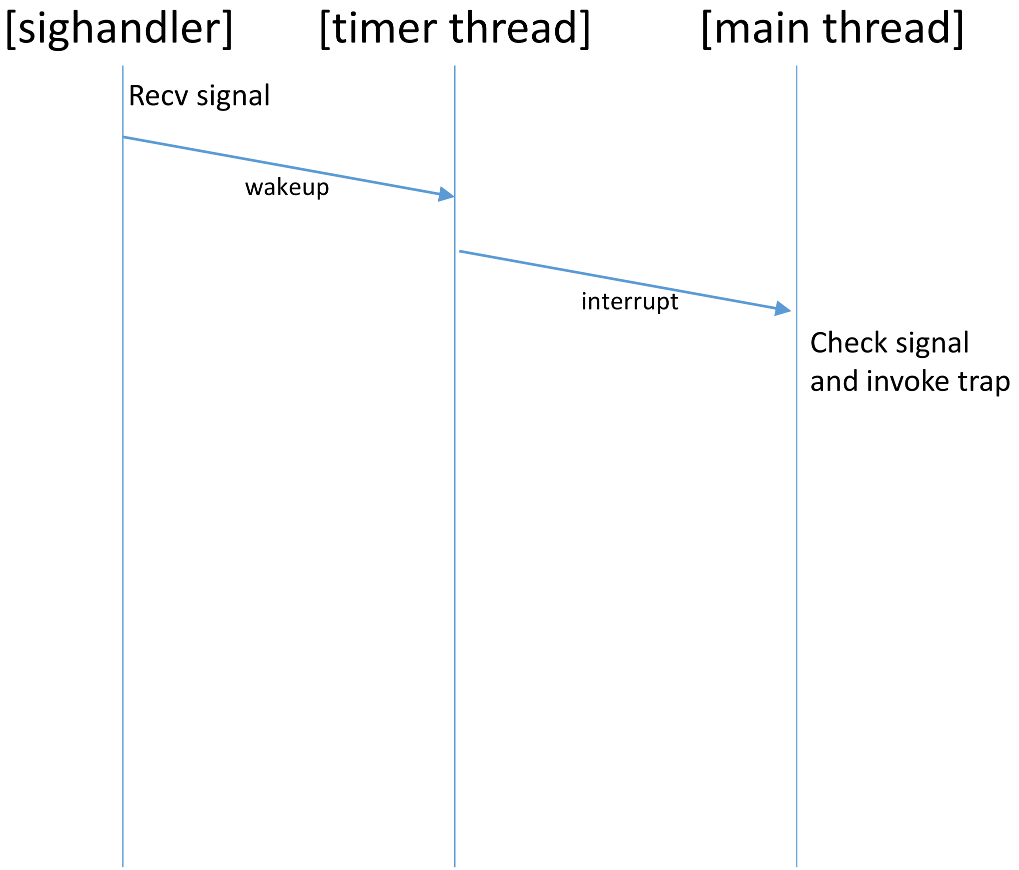
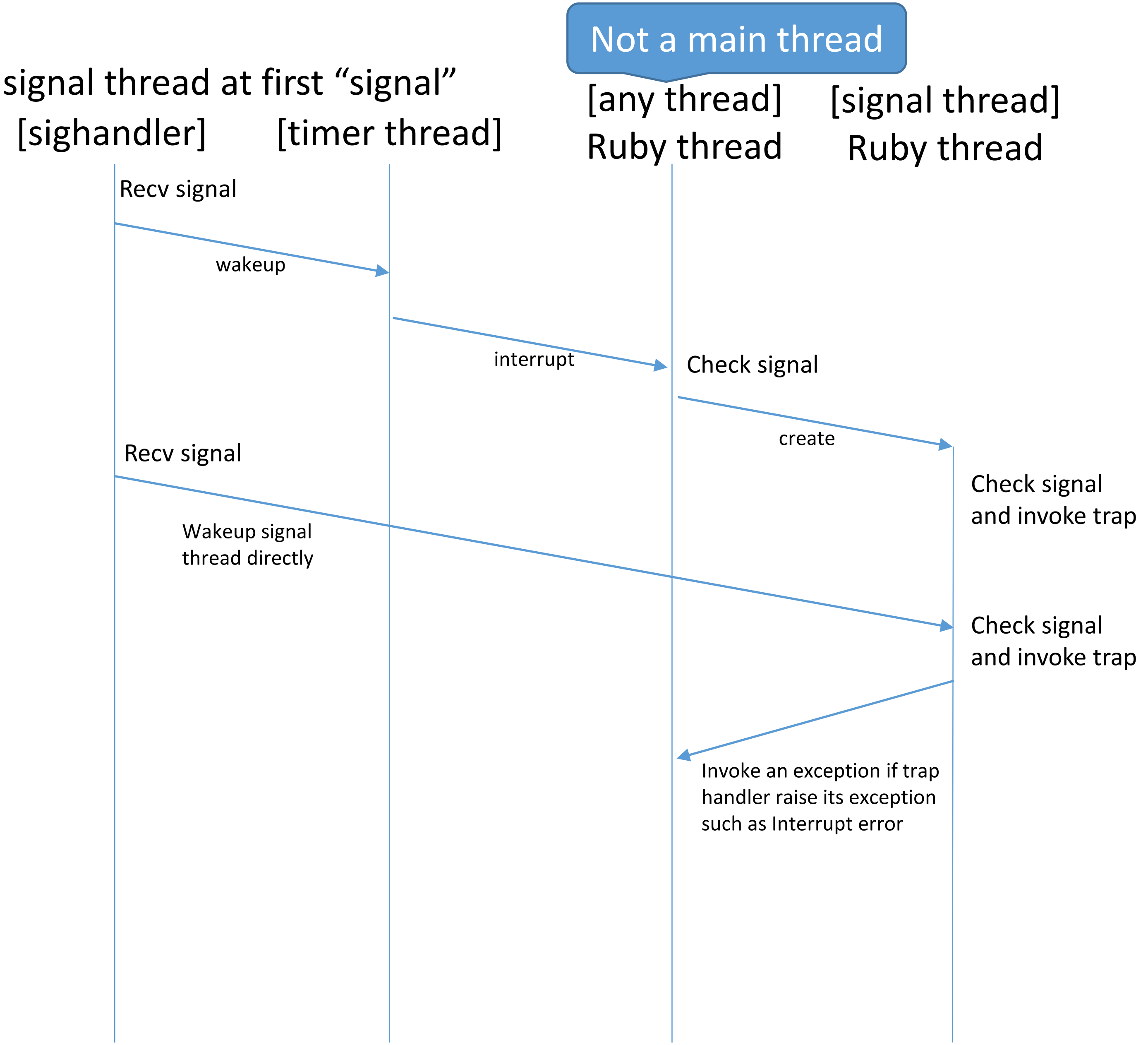
Proposal:

Creating a independent signal thread for trap handler

Current signal/trap handling



Plan1:
Create a signal thread at first “signal”



Advantage:

(1) Signal thread is independent on main thread, this means that you can use thread synchronization between trap handler and main thread. In other words, you can run any program in trap handler.

(2) Simplify a path from sighandler to trap invocation thread (after creation of a signal thread)

(3) Doesn’t need a difficult implementation (modify is limited).

Disadvantage:

(1) There is a small compatibility issue because “Thread.current” on a trap handler is not a main thread.

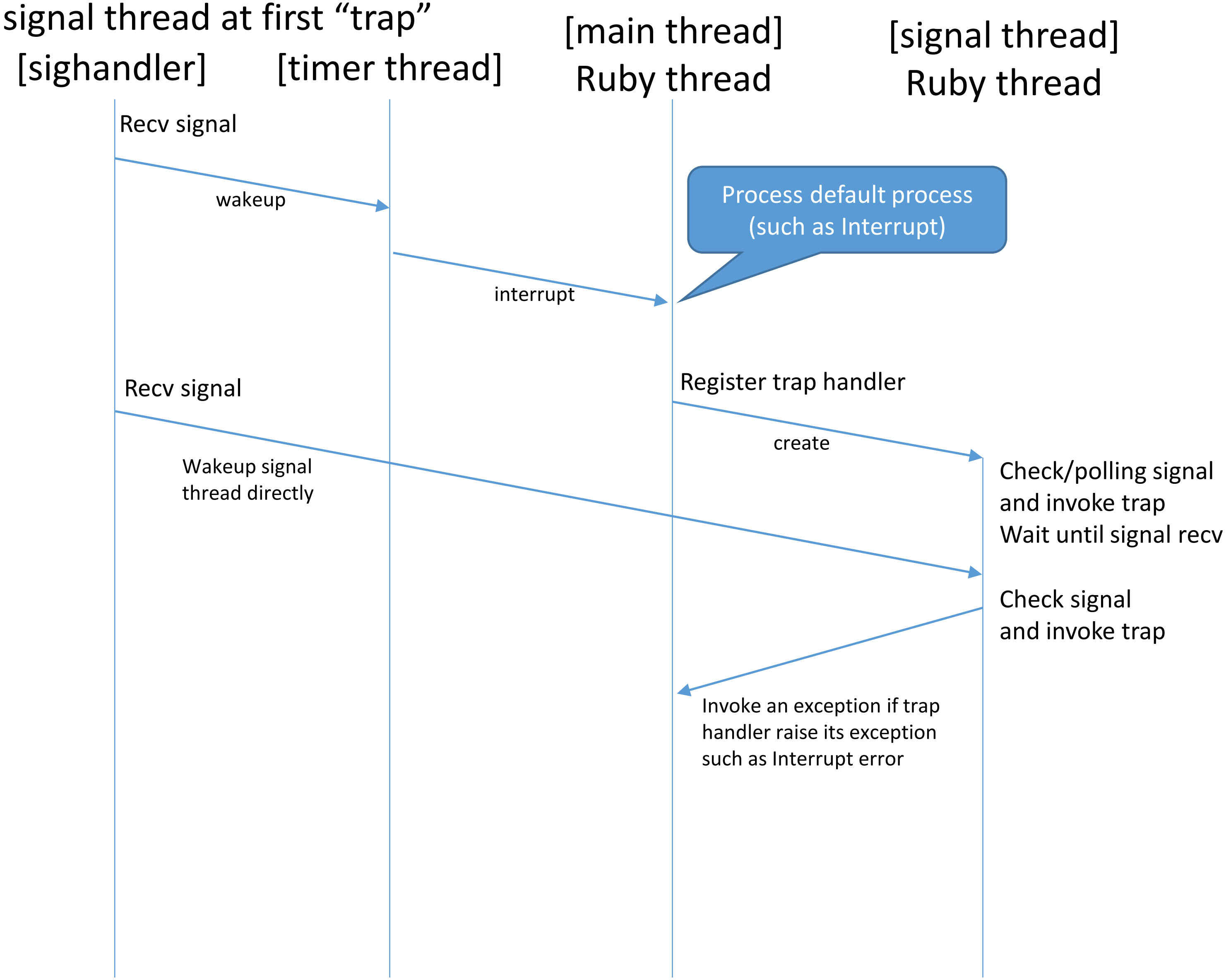
(2) A first time we create a signal thread, it has delays.

Discussion:

(1) Create signal thread at first like timer thread is **high** cost. Without `trap’, we don’t need a signal thread any more.

(2) In signal handler and timer thread, we can’t make a signal thread because creating “Ruby thread” (== signal thread) needs GVL. So the process path from timer thread to main thread is remained.

Plan2:
Create a signal thread at first “trap”



[sighandler]

[timer thread]

[main thread]
Ruby thread

[signal thread]
Ruby thread

Recv signal

wakeup

interrupt

Process default process
(such as Interrupt)

Register trap handler

Recv signal

create

Wakeup signal
thread directly

Check/polling signal
and invoke trap
Wait until signal recv

Check signal
and invoke trap

Invoke an exception if trap
handler raise its exception
such as Interrupt error

Advantage:

(1) No creation time delay at first signal receiving compared with Plan1.

Disadvantage:

(1) Complex? Because we need to separate “need trap handler” and “not need” signal.
(2) There is a possibility for race. For example, after receiving SIGINT (default trap), and just after that a trap handler is set for SIGINT, it is a race.

We may solve “Disadvantage (2)” with some techniques.