The Art of Execution Control for Ruby's Debugger

Koichi Sasada
Cookpad Inc.

About this talk

- Introduce “debug.gem” https://github.com/ruby/debug
  - Newly created debugger for Ruby 2.6 and later
  - Will be bundled with Ruby 3.1 (Dec/2021)
- Demonstrate “debug.gem”
  - Basic usage instructions
  - Advanced features
- Show “How to make a debugger” on Ruby
  - Introduction of recent features of `TracePoint`
  - Some techniques for a debugger

About Koichi Sasada

- Ruby interpreter developer employed by Cookpad Inc. (2017-) with @mame
  - YARV (Ruby 1.9-)
  - Generational/Incremental GC (Ruby 2.1-)
  - Ractor (Ruby 3.0-)
  - …
- Ruby Association Director (2012-)
  - 2021 Call for Grant Proposals

What is a debugger?

- A tool to help debugging
  - To investigate the cause of problems
  - To know the program live state
  - To understand the program
- Basic features
  - CONTROL execution
    - STOP at breakpoints
    - STEP forward to the next line
    - …
  - QUERY program status

Ruby's existing debuggers

- lib/debug.rb
  - `ruby -r debug script.rb`
  - Standard library, but maybe nobody uses it
- byebug
  - `byebug script.rb`
- debase / ruby-debug-ide
  - Used by IDE (rubymine, vscode, …)

Why create yet another debugger?

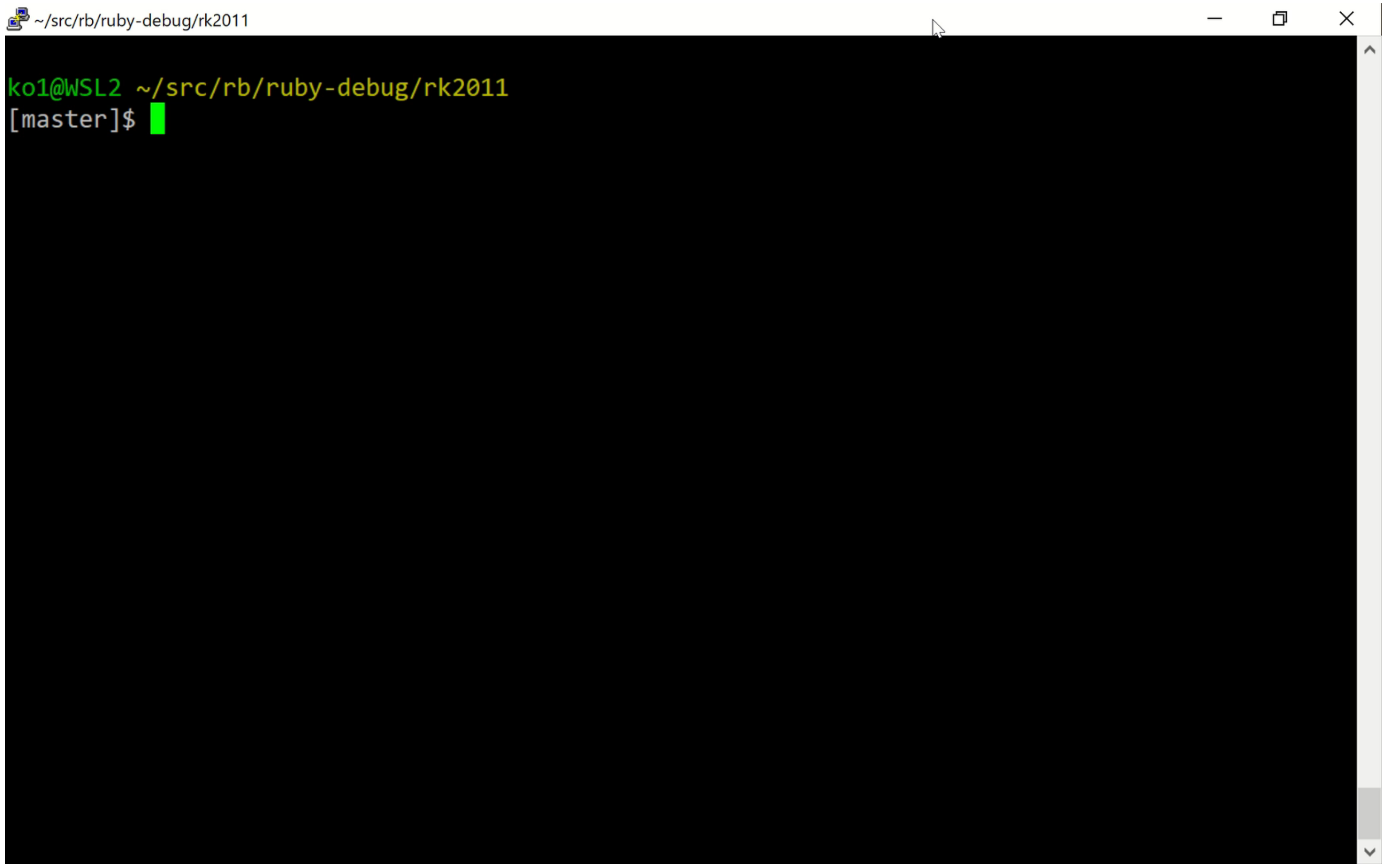
- Performance
  - Existing debuggers slow with breakpoints
  - Recent TracePoint API support line-specific
- Native support for remote execution and IDE
- Native support for Ractors
- (and I like to make this kind of tools)

How to use a debugger?

- Set breakpoints.
- Run a program with a debugger.
- At the breakpoint, enter the debugger console.
- Use debug commands.
    - Control execution
        - Move to next line.
        - Continue the execution and stop at the next breakpoint.
        - Set another breakpoint (e.g. stop when Exception is raised).
    - Query the program status (e.g. see the local variables of another frame).
    - Change the configuration (e.g. change coloring option).

Introduction of “debug.gem”

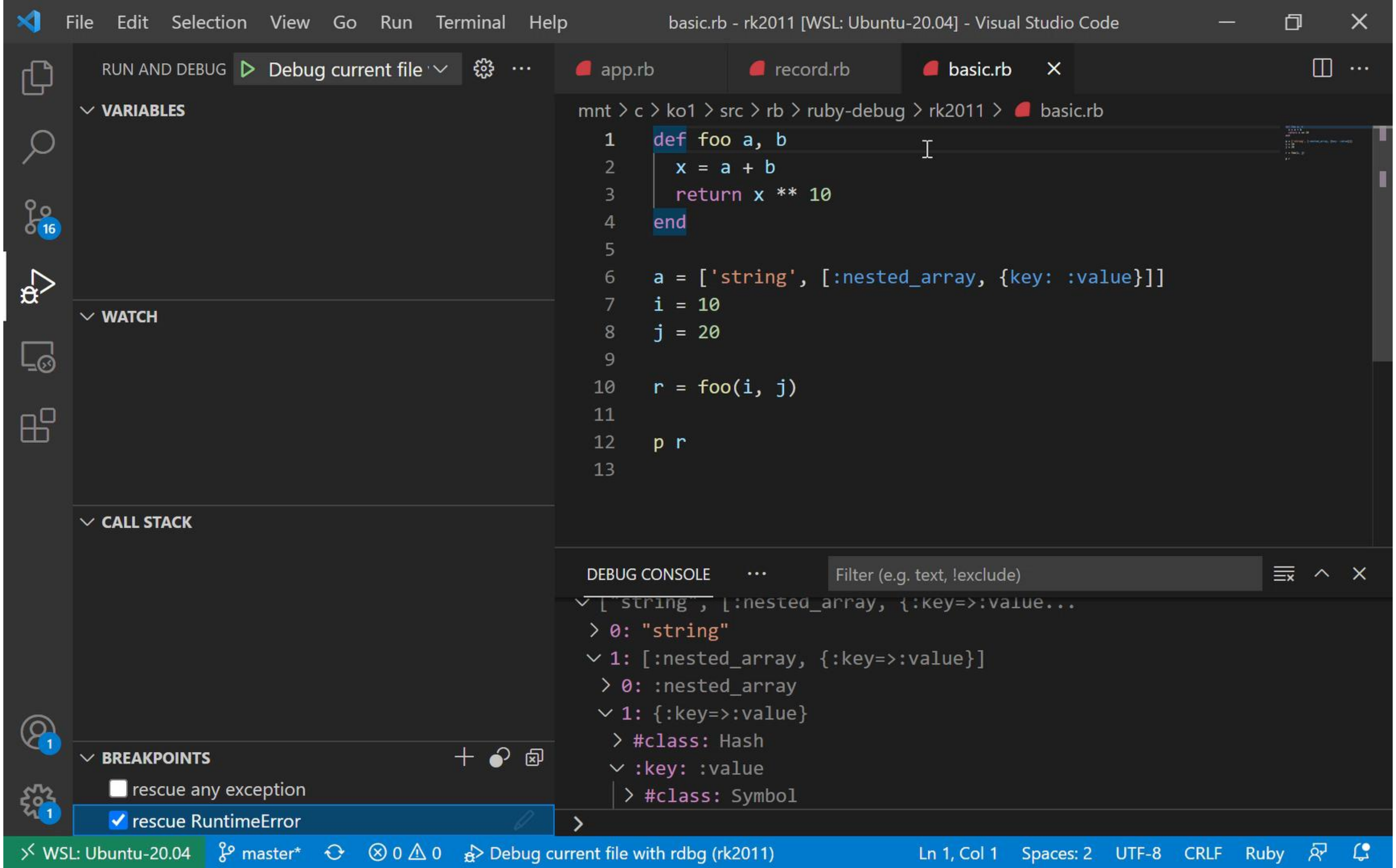
Demo: Basic usage


~/src/rb/ruby-debug/rk2011

```
ko1@WSL2 ~/src/rb/ruby-debug/rk2011
[master]$
```

Demo: VSCode integration

- Implement DAP (Debug Adapter Protocol) to communicate IDEs (VSCode, …)



```ruby
def foo a, b
  x = a + b
  return x ** 10
end

a = ['string', [:nested_array, {key: :value}]]
i = 10
j = 20

r = foo(i, j)

p r

```

- [ ] rescue any exception
- [x] rescue RuntimeError

```
["string", [:nested_array, {:key=>:value...
  0: "string"
  1: [:nested_array, {:key=>:value}]
    0: :nested_array
    1: {:key=>:value}
      #class: Hash
      :key: :value
        #class: Symbol
```

Start debugging with debug.gem

- Use “rdbg” command
  - like traditional debuggers such as GDB/LLDB
  - `rdbg script.rb`
  - `rdbg –c rake test`
  - `rdbg --open …` # open remote debugging port
- Require “debug*” files
  - like old lib/debug.rb
  - `require “debug” == “rdbg --nonstop --no-sigint-hook”`
  - `require “debug/start” == “rdbg”`
  - `require “debug/open” == “rdbg --open”`
- Push IDE’s button

Set a breakpoint

- Use “break” command at the beginning
  - `break 10` # break at 10 line on current file
  - `break foo.rb:10` # break at the location
  - `break MyClass#my_method` # break at the method
  - `break … if foo == bar` # break if foo == bar
  - `catch FooException` # break at FooException is raised
- Use “`binding.break`” method in your program
  - You can insert it like “`binding.irb`”
  - “`binding.b`” for short
  - You can write debug command in your program with

```
binding.b do: “debug_command”
```

Set a breakpoint (cont.)

- Use “break” command at the beginning
  - Do not need to modify the source code
  - Cooperation with IDE/Editor (e.g. set BP with F9 on VSCode)
- Use “`binding.break`” method in your program
  - Straight forward for some Ruby users
  - Control debugger from debuggee

Demo: Combination with program and debugger binding.break is a breakpoint like binding.irb

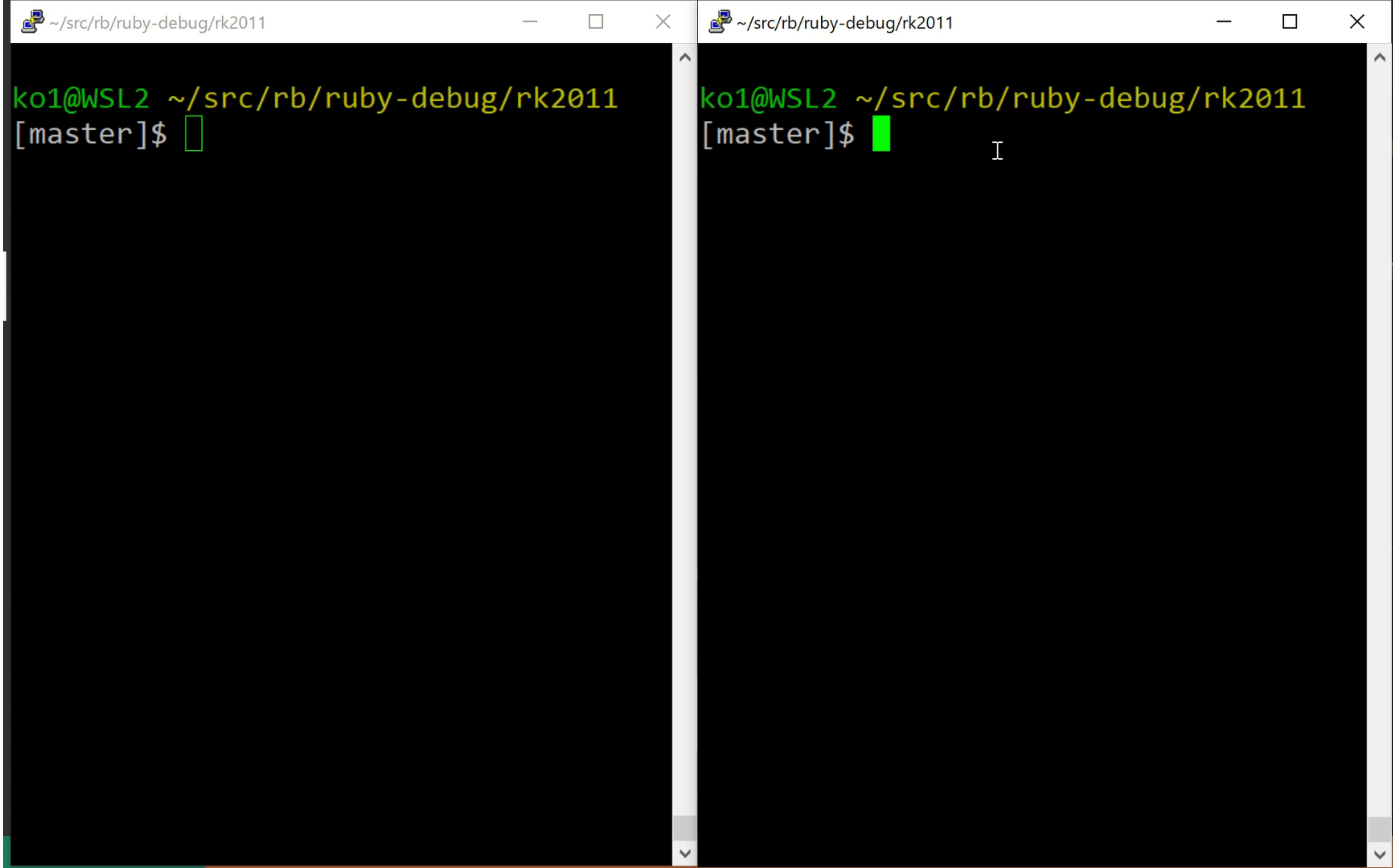
```
# enable “trace line” feature while bar()
def foo
  binding.break do: ‘trace line’
  bar()
  biding.break do: ‘trace off line’
end
```

Advanced features

- Remote debugging
- Postmortem debugging
- Event tracing
- Record and replay debugging

Demo: Remote debugging
Connect over network

- Easy to open remote debug port and attach
  - rdbg --open script.rb
    - rdbg -O script.rb
    - require ‘debug/open’ # in script
  - rdbg --attach
    - rdbg -A
- Debug deamon like process
- Query the process status like sigdump but more details



~/src/rb/ruby-debug/rk2011

```
ko1@WSL2 ~/src/rb/ruby-debug/rk2011
[master]$
```

~/src/rb/ruby-debug/rk2011

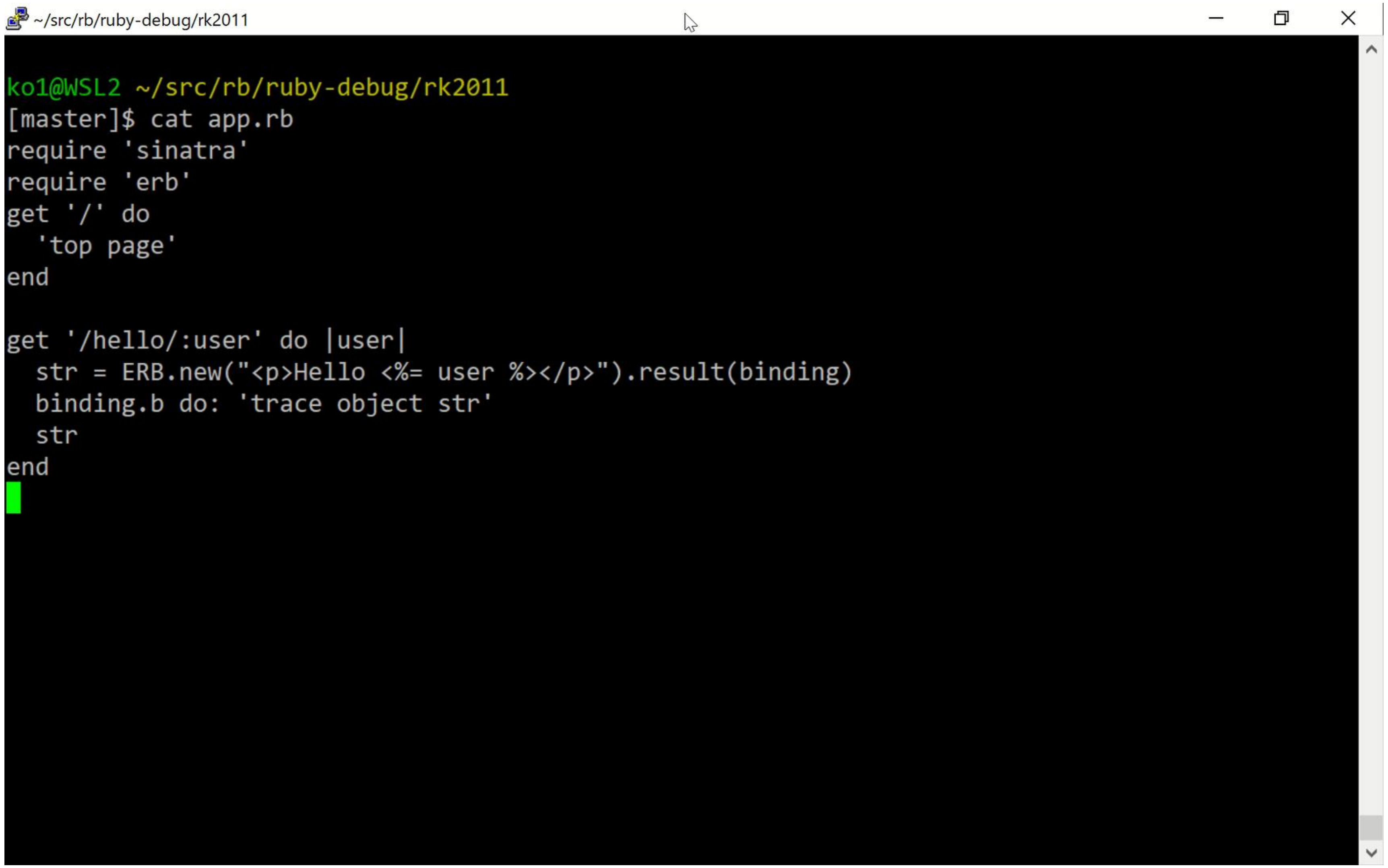
```
ko1@WSL2 ~/src/rb/ruby-debug/rk2011
[master]$
```

Demo: Postmortem debugging
Debug dead Ruby process

~/src/rb/ruby-debug/rk2011

```
ko1@WSL2 ~/src/rb/ruby-debug/rk2011
[master]$
```

Demo: Event tracing
Show interesting events

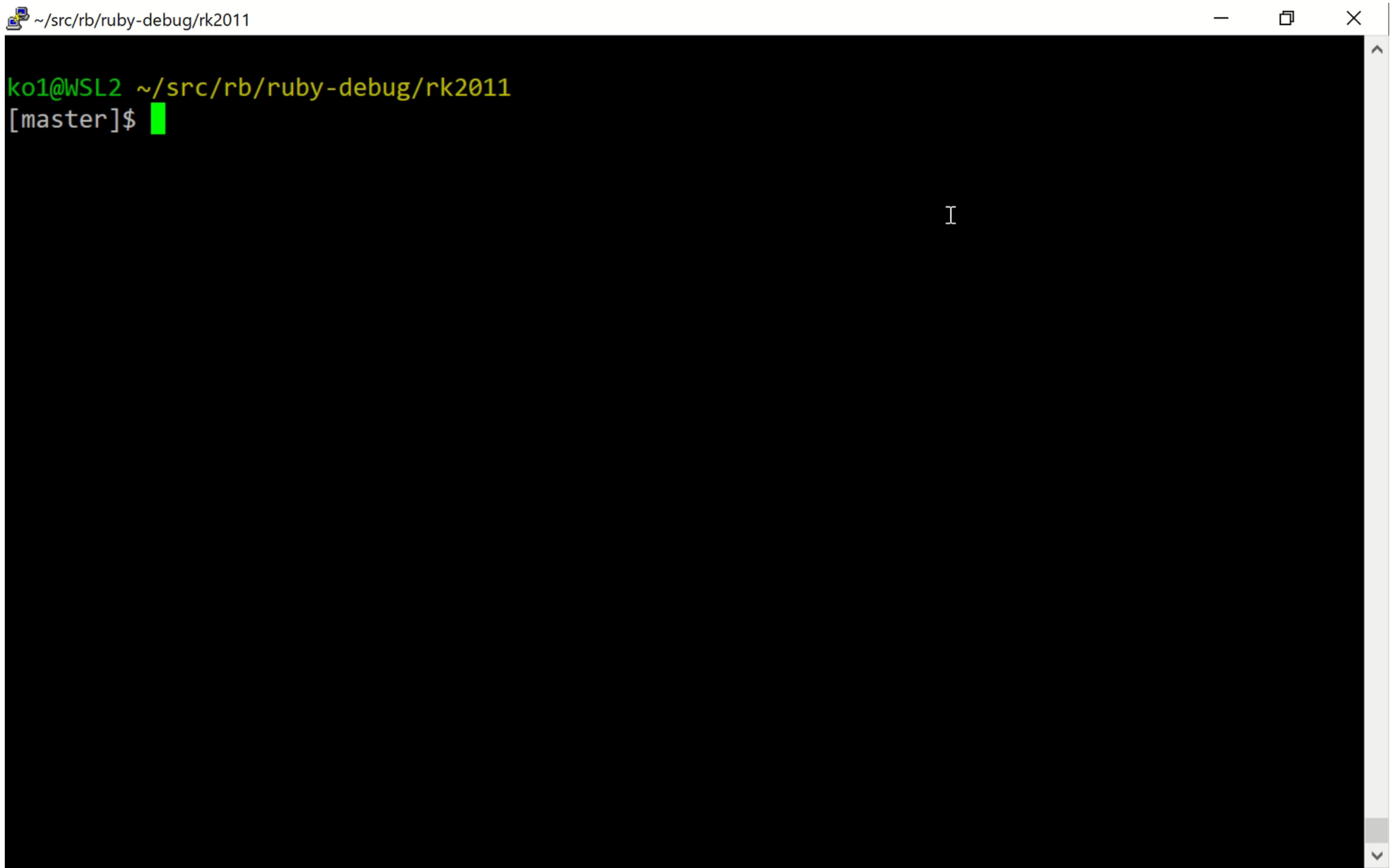


~/src/rb/ruby-debug/rk2011

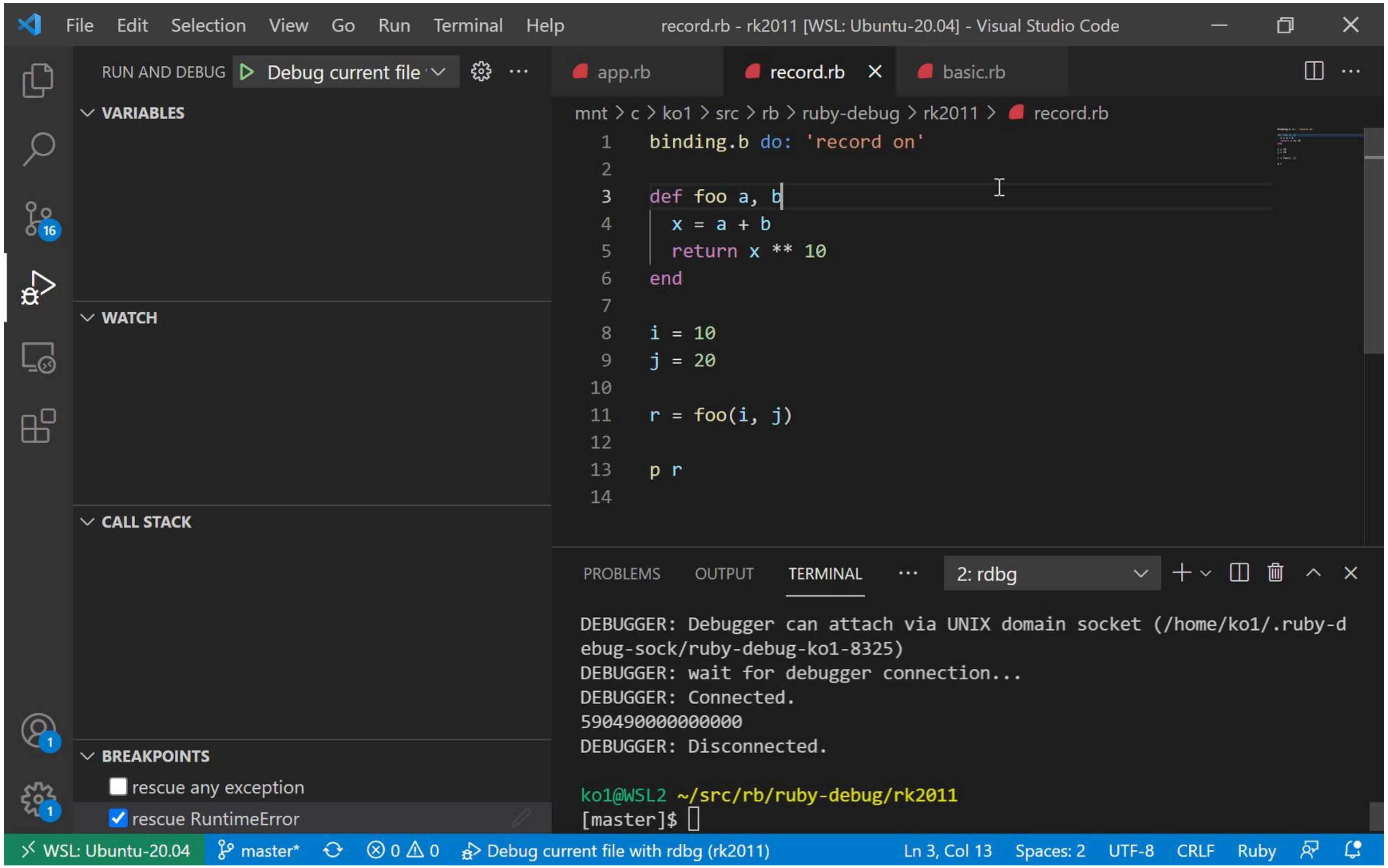
```
ko1@WSL2 ~/src/rb/ruby-debug/rk2011
[master]$ cat app.rb
require 'sinatra'
require 'erb'
get '/' do
  'top page'
end

get '/hello/:user' do |user|
  str = ERB.new("<p>Hello <%= user %></p>").result(binding)
  binding.b do: 'trace object str'
  str
end
```

Demo: Record and replay debugging
Backward stepping execution


~/src/rb/ruby-debug/rk2011

```
ko1@WSL2 ~/src/rb/ruby-debug/rk2011
[master]$
```



```ruby
binding.b do: 'record on'

def foo a, b
  x = a + b
  return x ** 10
end

i = 10
j = 20

r = foo(i, j)

p r

```

```
DEBUGGER: Debugger can attach via UNIX domain socket (/home/ko1/.ruby-d
ebug-sock/ruby-debug-ko1-8325)
DEBUGGER: wait for debugger connection...
DEBUGGER: Connected.
590490000000000
DEBUGGER: Disconnected.

ko1@WSL2 ~/src/rb/ruby-debug/rk2011
[master]$
```

Performance

```
def fib n
  if n < 0
    raise # breakpoint here!!
  elsif n<2
    n
  else
    fib(n-1)+fib(n-2)
  end
end

require 'benchmark'
Benchmark.bm{|x|
  x.report{ fib(35) }
}
```

	Without breakpoint	With breakpoint
ruby	0.93	N/A
rdbg (debug.gem)	0.92	0.92
byebug	1.23	75.15
RubyMine	0.97	22.66
old lib/debug.rb	221.88	285.99

Execution time (sec)

ruby 3.0.1p64
rdbg 1.0.0.rc2
byebug 11.1.3
RubyMine 2021.2.1 w/ debase 0.2.5.beta2

Intel(R) Core(TM) i7-10810U CPU, Windows 10, WSL2

How to make a debugger on Ruby?

Key feature: TracePoint

- “TracePoint” hooks many events in Ruby
  - Replacement of “set_trace_func” from Ruby 1.9
  - “line”, “call”, “return”, …
- Set TracePoint at the specific location (from Ruby 2.6)
  - “enable(target: …, target_line: …)”
  - “target” is specified by a Method/Proc/ISeq (bytecode)

```
def foo a
  b = a + 1 # line 2
  b         # line 3
end
TracePoint.new(:line){|tp| p tp
}.enable(target: method(:foo), target_line: 3){
  foo 10
} #=> #<TracePoint:line t.rb:3 in `foo'>
```

You can implement a line breakpoint …?

- Debug command: **break file:line**
  - Set a breakpoint at file:line
- Question: How to collect the code for “target:”?
- Answer: Use ObjectSpace to collect all ISeqs (bytecode)

How to implement step execution?

- Use “TracePoint” without “#enable(target:…)”
  - “#enable” method **without** “target:” keyword will hook at any hook points
  - Set “:line” event and trap the execution

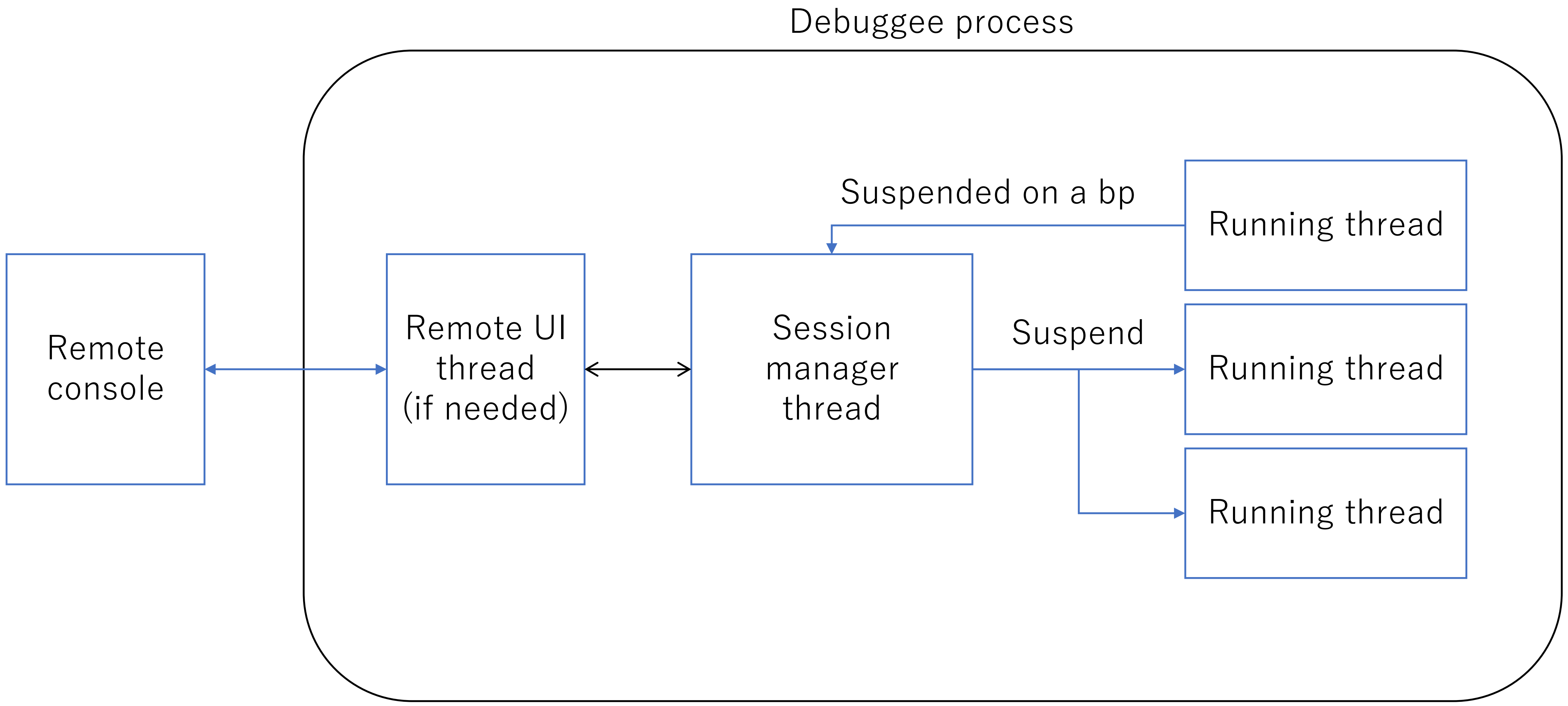
How to implement capturing frames information?

- Use internal API “rb_debug_inspector_*”
  - You can use it from Ruby with “debug_inspector.gem”.
  - With C-extension we can control the capturing target.

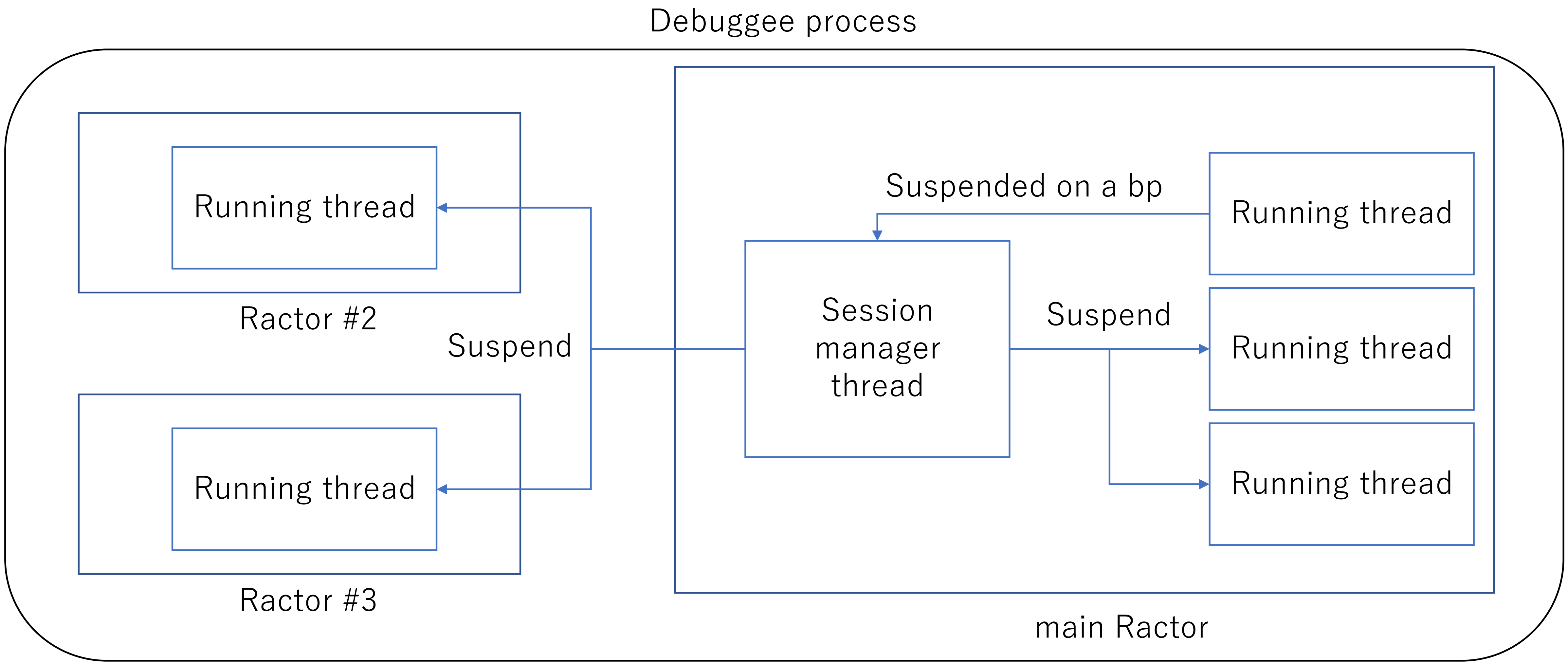
How to support Threads/Ractors?

- We can’t share the objects between Ractors
  → Existing debugger can’t support Ractors.
- Ractor programming is hard
  → Debugger can be important for ractor debugging

Session management thread and communication



Session management thread and communication



But not available now…

Acknowledgements

- Naoto Ono san (@ono-max) implements test-frameworks for the debugger. It is part of his GSoC project.
- Stan Lo san (@st0012) submits tremendous patches to improve the debugger usability such as coloring and so on based on his debugger trials. Also, he makes many tests for the debugger.
- Ruby committers helps me to design and implement the debugger

Conclusion

- “debug.gem” is newly created Ruby debugger from scratch
  - Useful features on modern UI.
  - No performance penalties.
- “`gem install debug`” now!
  - And give us your feedback.
  - I love to introduce the debugger on your meetup, please contact me.
- Ruby already has useful features like “TracePoint” for the debugger.
- Ractor supports is not available, now working on.