RincGC: Incremental GC for CRuby/MRI interpreter

Koichi Sasada

Heroku, Inc.

Background and motivation

- Ruby 2.1 had introduced generational GC
  - Short marking time on minor GC
  - Improve application throughput
- Still long pause time on major GC
  - Long pause time affects user response time

Proposal:
Introduce incremental GC for major GC

- Introducing incremental GC to reduce pause time
- Can combine with Generational GC

	Generational GC	Incremental GC	Gen+Inc GC
Throughput	High	Low (a bit slow)	High
Pause time	Long	Short	Small

Base idea:
Incremental GC algorithm

- Move forward GC processes incrementally
  - Mark slots incrementally
  - Sweep slots incrementally
- Incremental marking in 3 phase
  - (1) Mark roots (pause)
  - (2) Mark objects reachable from roots (incremental)
  - (3) Mark roots again, and mark remembered objects (pause)
- Mark objects with three state (white/grey/black)
  - White: Untouched objects
  - Grey: Marked, and prepare to mark directly reachable objects
  - Black: Marked, and all directly reachable objects are marked
- Use write barriers to avoid marking miss from marked objects to live objects
  - Detect new reference from black objects to white objects
  - Remember such source black objects (marked at above (3))

RincGC: Incremental GC for CRuby/MRI

- Incremental marking
  - (1) mark roots (gc_mark_roots())
  - (2) Do incremental mark at rb_newobj_of()
  - (3) Make sure write barrier with WB-protected objects
  - (4) Take care of **WB-unprotected objects** (MRI specific)
- Incremental sweeping
  - Modify current lazy sweep implementation

Incremental marking Implementation

- (1) mark roots (gc_mark_roots())
  - Push all root objects onto “mark_stack”
- (2) Do incremental mark at rb_newobj_of()
  - Fall back incremental marking process periodically
  - Consume (pop) some objects from “mark_stack” and make forward incremental marking
- (3) Make sure write barrier with WB-protected objects
  - Push src object onto “mark_stack” again if it is not in “mark_stack”
- (4) Take care of **WB-unprotected objects** (MRI specific)
  - After incremental marking (“mark_stack” is empty), re-scan all roots and all living non-WB-protected objects
  - WB-unprotected objects are represented by bitmap (WB_UNPROTECTED_BITS)

Incremental marking Pseudo code in Ruby

```
def mark(obj)
  return if obj.mark_bit
  obj.mark_bit = true
  obj.marking_bit = true
  $mark_stack.push(obj)
end

def start_marking
  GC.state = :mark
  $root_objects{|o| mark(o)}
end

def incremental_mark(n)
  n.times{
    return if $mark_stack.empty? && finish_marking
    obj = mark_stack.pop
    reachable_objects_from(obj){|o| mark(o)}
    obj.marking_bit = false
  }
end
```

```
def finish_marking
  root_objects{|o| mark(o)} # re-scan root objects
  return false unless mark_stack.empty?
  $marked_wb_unprotected_objects.each{|unprotected_obj|
    unprotected_obj.reachable_objects{|o| mark(o)}
  }
  mark(obj) while obj = $mark_stack.pop

  GC.state = :sweep
  return true
end

def write_barrier(a, b)
  if GC.state == :mark && a.mark_bit && !a.marking_bit && !b.mark_bit
    a.marking_bit = true
    mark_stack.push(a)
  end
end
```

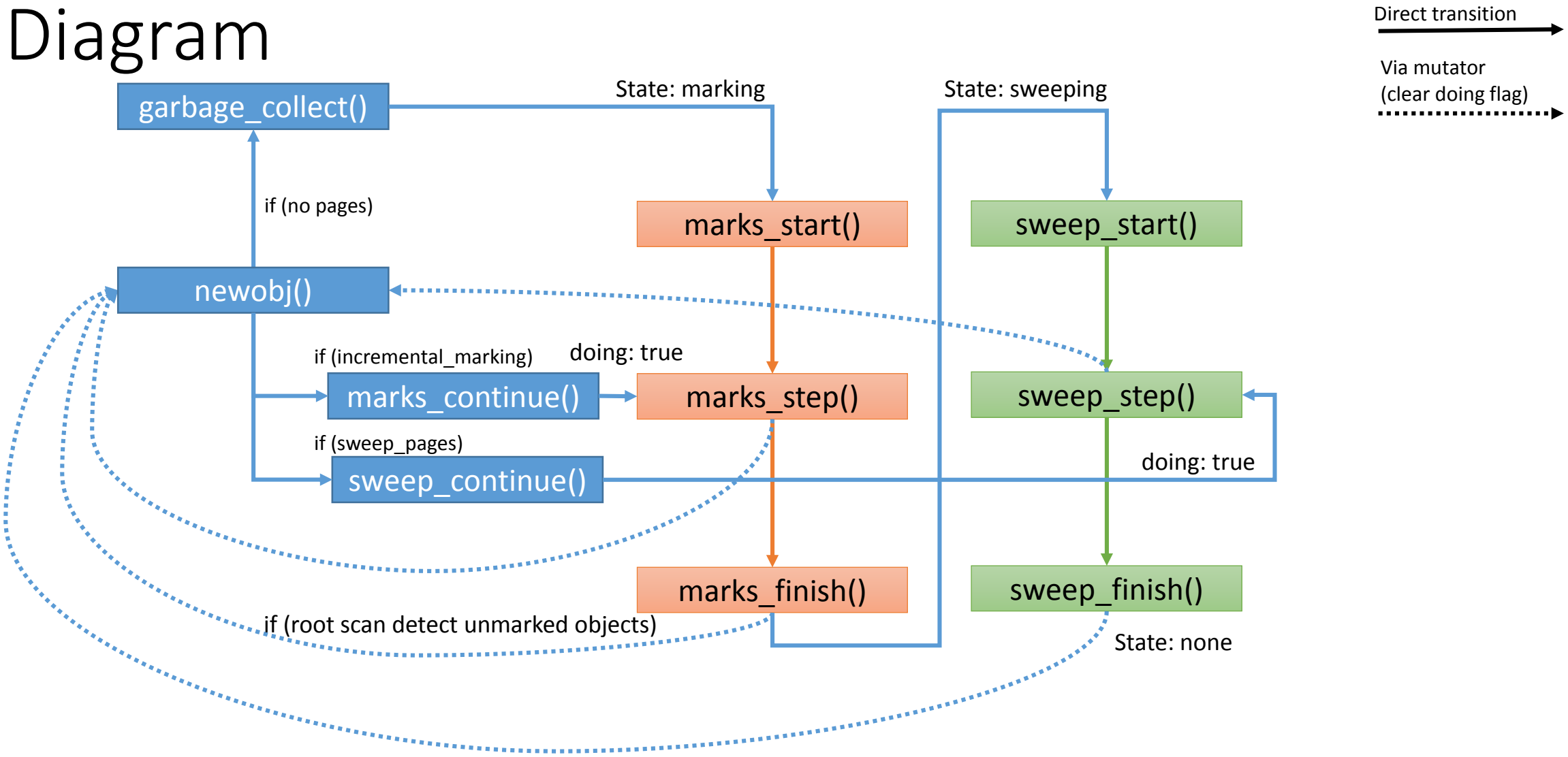
Incremental marking Implementation

Traditional GC coloring terminology	RincGC	
	mark_bit	marking_bit
White	FALS	FALSE
Grey	TRUE	TRUE
Black	TRUE	FALSE

Incremental sweeping Implementation

- Current implementation
  - Iterate until no pages
    - Sweep 1 page (a set of slots)
    - Consume 1 page
  - After that, no empty pages
- Modify implementation
  - Iterate
    - Sweep 2 page (a set of slots)
    - Consume ***1*** page (1 page remain)
  - After that, half of pages are left
  - We can use this half of pages for incremental marking

Implementation Diagram



Future optimization

- Unifying bitmaps
  - Total 5 bitmap planes
    - Mark bits (for marking)
    - Remember set bits (for generational GC)
    - WB unprotected bits (for gen/inc)
    - Oldgen bits (for generational GC)
    - Marking bits (for incremental to represent in mark_stack)
- Reduce the number of root objects and WB-unprotected objects
  - Root objects marked twice (or more)
  - At the last of marking phase, all of living WB-unprotected objects are marked (traverse from this objects)