Ruby処理系の
コンパイル済みコードの設計

笹田耕一、松本行弘（Heroku, Inc.）

ko1@heroku.com, matz@heroku.com

heroku

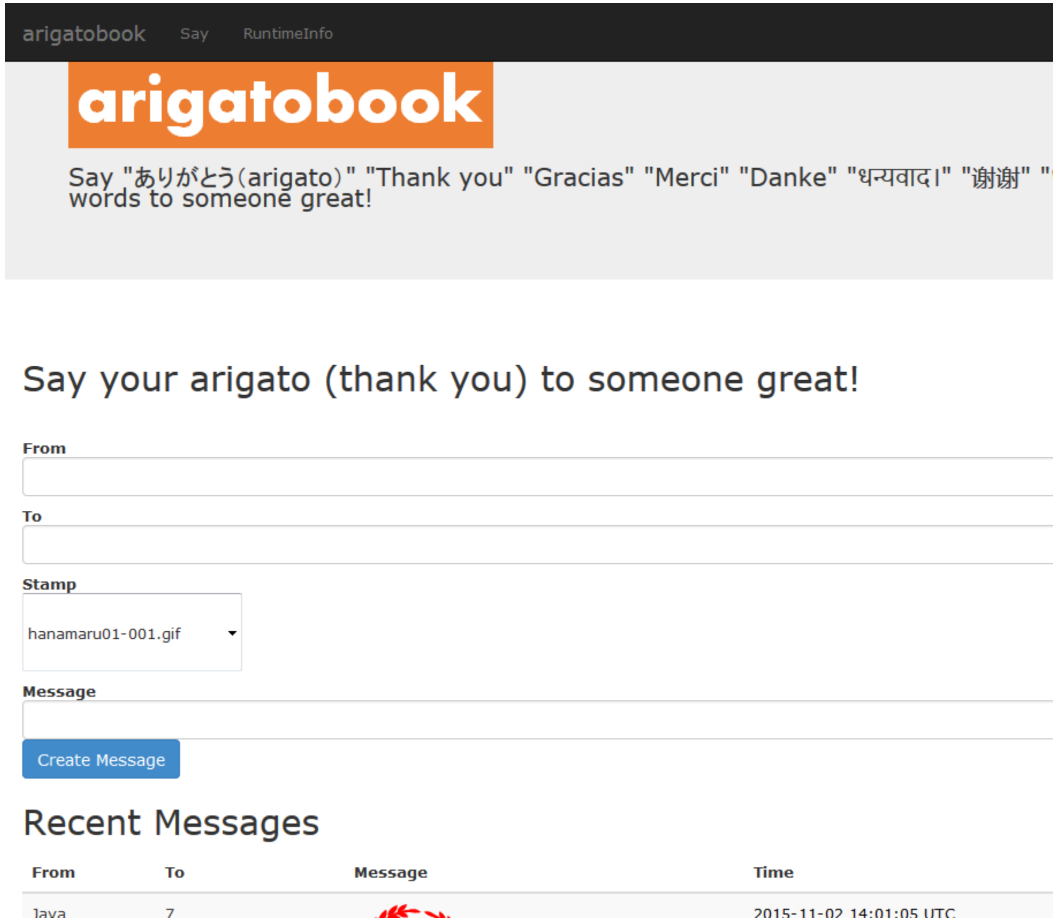
第106回プログラミング研究発表会

今日の話

- Rubyスクリプトをバイトコード列に事前にコンパイル・後でロードする機能を試作
  - JVMクラスファイルみたいなもので、新規性はない
  - 実際に観察すると、遅延ロードによって、ロードするバイトコードは、実際には15%程度でいいのかも、という結果
- 設計する前に、ちゃんと調べろ、という教訓の話

Background
Ruby as a web application dev language

Background
Sample application



Say "ありがとう(arigato)" "Thank you" "Gracias" "Merci" "Danke" "धन्यवाद।" "谢谢" "
words to someone great!

Say your arigato (thank you) to someone great!

From

To

Stamp

hanamaru01-001.gif

Message

Recent Messages

From	To	Message	Time
Java	7		2015-11-02 14:01:05 UTC

Very simple sample application
http://atdot.net/ab/

Background
Load many libraries

Loaded Gems (Gem ≒ Library)	91
Loaded Ruby scripts	1,550
Average line number of loaded Ruby scripts	140
Maximum line number of loaded Ruby scripts	2,970

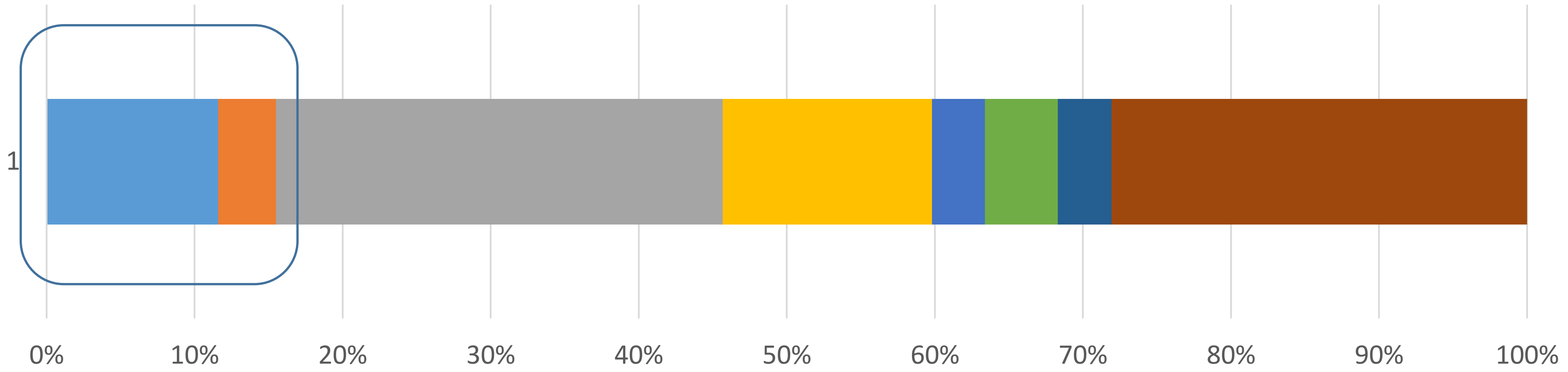
Problem
Increasing loading time

1. Specify loaded scripts
   - Tools such as “Bundler” help.
   - Some other ideas (out of scope from our research)
2. Read loaded scripts
   - Traditional “Disk cache” will help (out of scope)
3. **<u>Parse and compile loaded scripts to generate Bytecode</u>**
   - **<u>We need to repeat this process for all of ruby interpreter</u>**

- Loading time is important, especially for application development phase

Problem
Increasing memory consumption

Bytecode consumes 15% (20MB)

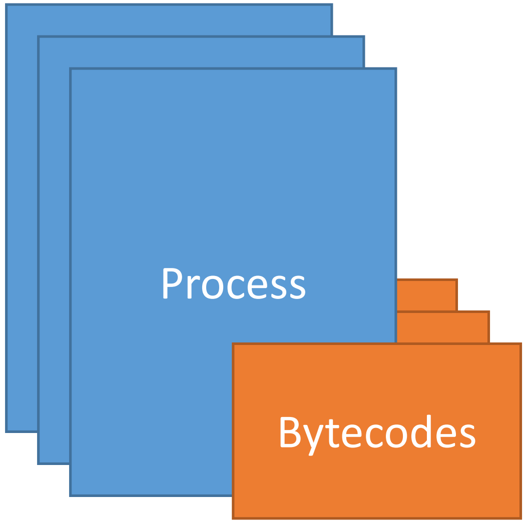


	1
🟦 iseq_setup@compile.c	15,595,764
🟧 rb_iseq_new_with_opt@iseq.c	5,231,136
⬜ heap_assign_page@gc.c	40,518,400
🟨 st_init_table_with_size@st.c	18,994,480
🟦 rb_str_buf_new@string.c	4,817,252
🟩 st_update@st.c	6,578,736
🟦 onig_region_resize@regexec.c	4,891,968
🟫 others	37,676,810

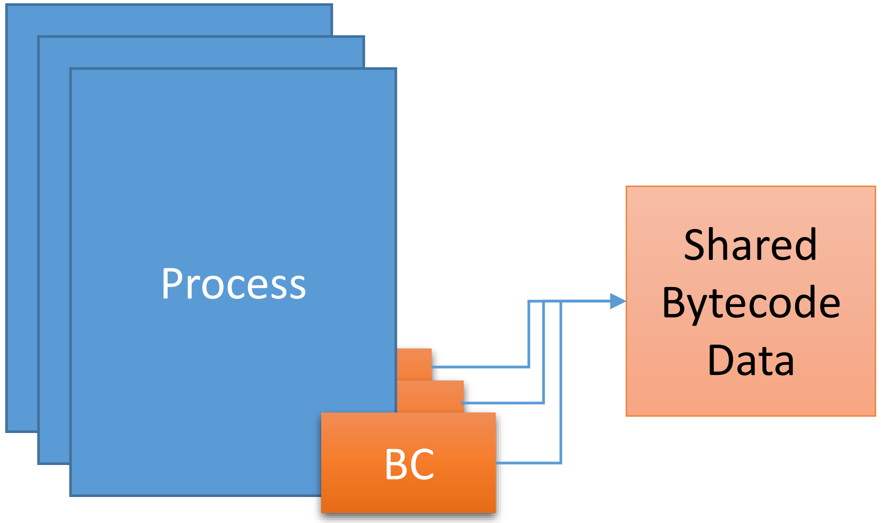
Measured by valgrind/massif

Problem
Increasing memory on multi-process

- Only small application consume 20MB by bytecodes
- N processes can consumes N times 20MB (or more)
- CoW can help, but not guaranteed
- **Shared bytecode data is required**



Independent BCs

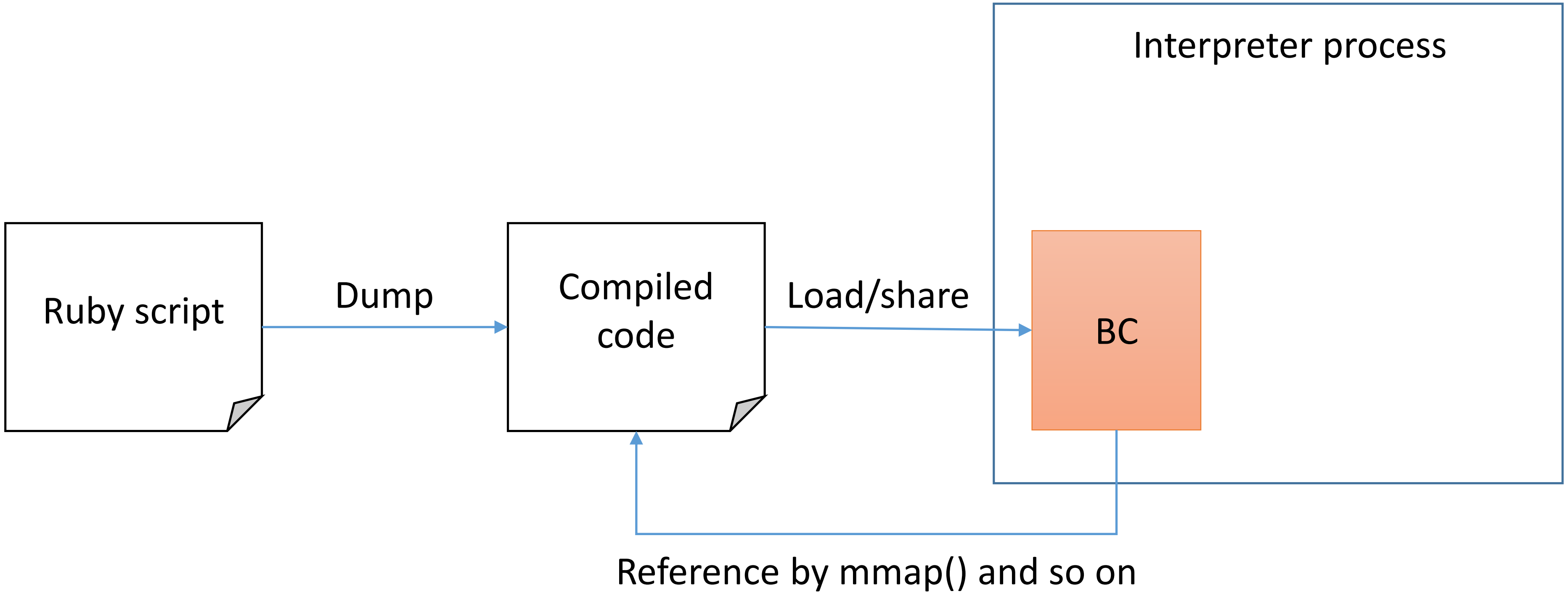


(Partialy) Shared BCs

Approach

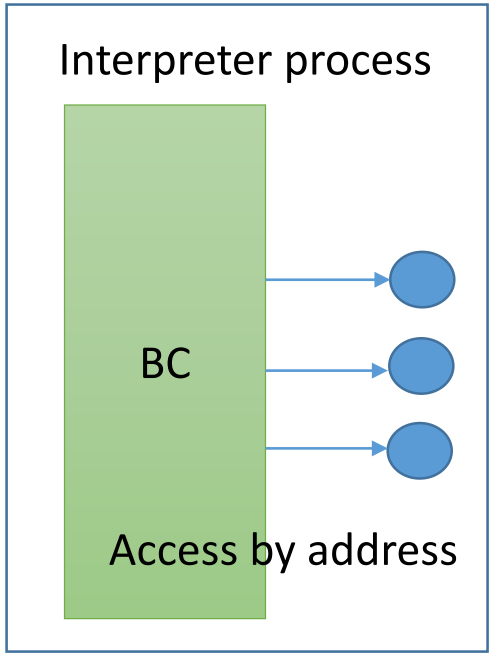
- Goal: Fast load, low memory consumption, non-negative performance impact loading feature
- Our approach
  - Prepare compiled code beforehand
    - General idea (so many languages Java, Python, PHP, emacs, … support)
  - Machine dependent compiled data (word size, endian, etc…)
- Related work: Ruby’s case
  - mruby generates compiled code
  - Ikehara’s code compaction
  - Some native compilers

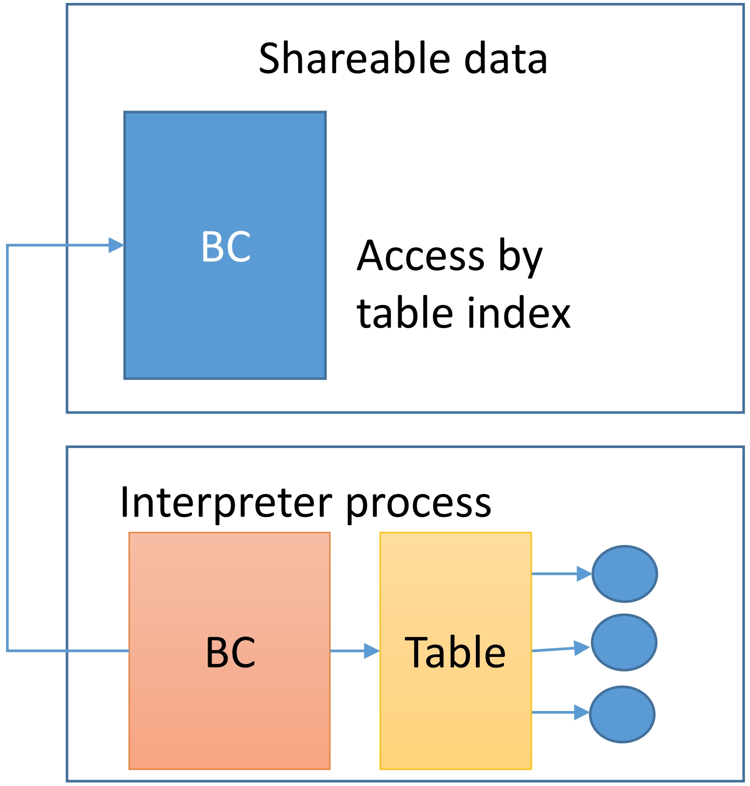
Approach
Bytecode dump/load



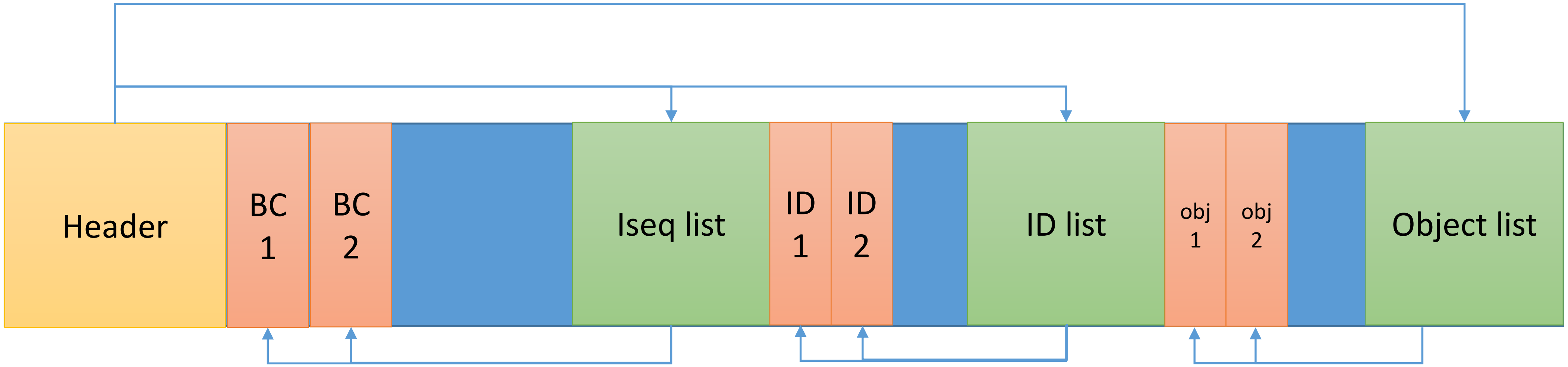
Design trade-off

- Shareable data reduces loading time and memory consumption
- But introduces indirect accesses, slows down performance





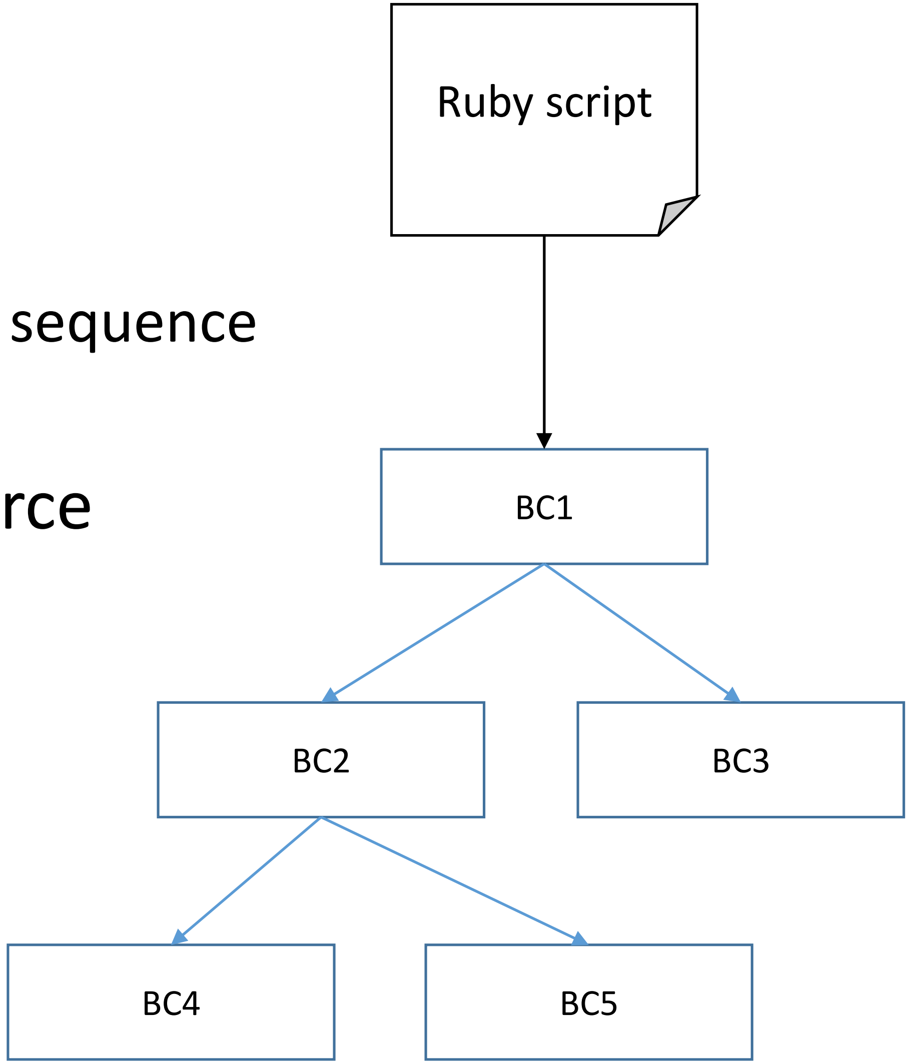
Data format



- Iseq (BC), ID, Objects are pointed by index of each lists in each data
- Objects are serialized by Marhasl (Ruby’s feature)
- Dump machine dependent data (can’t migrate compiled code)
- No verifier (because this file is not for migrations)

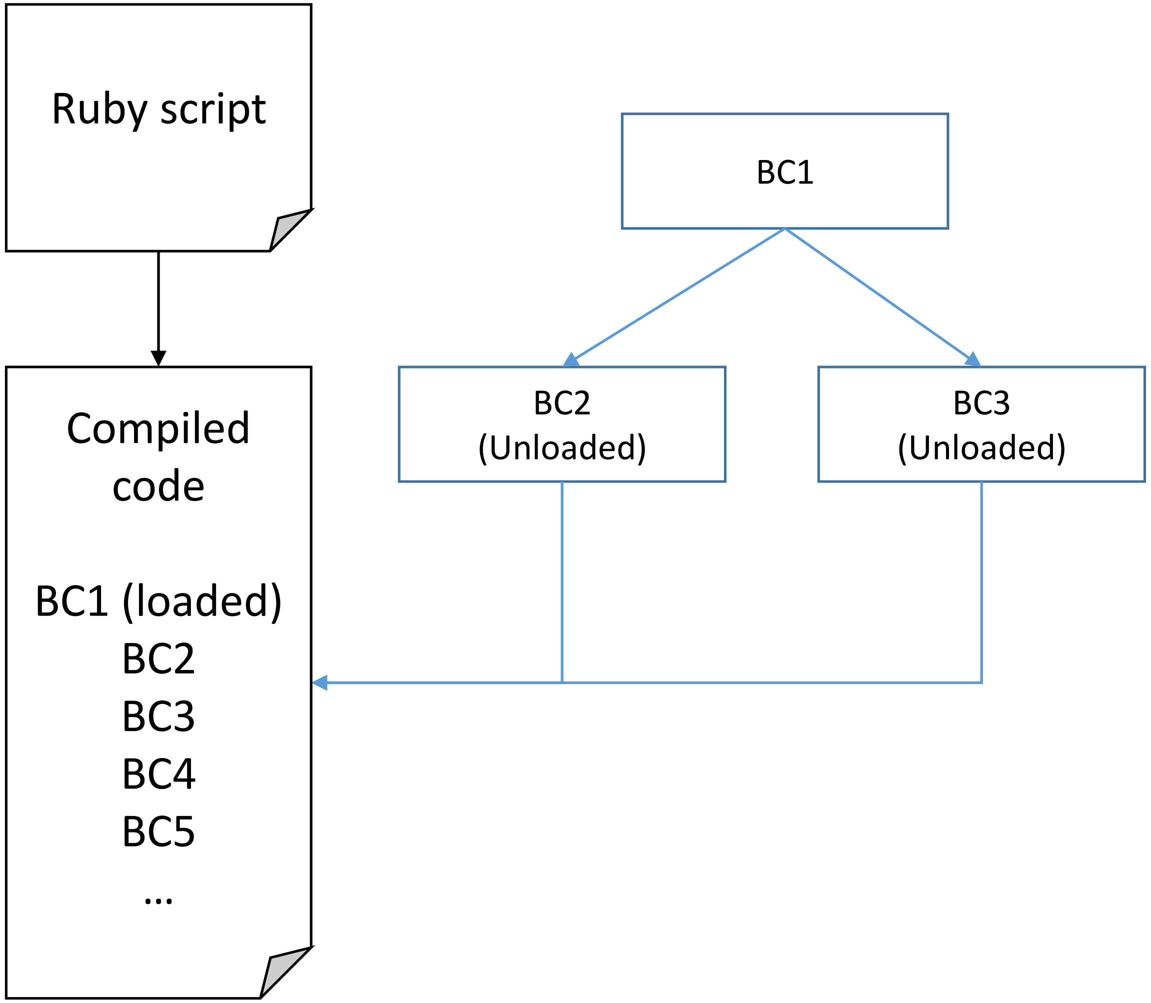
Implementation technique
Lazy loading

- A Ruby script has several bytecodes
  - Each scope has own independent bytecode sequence
  - Bytecodes are tree data structure (like AST)
- Each bytecode consumes memory resource
  - Bytecode header
  - Bytecode sequence
  - …



Implementation technique
Lazy loading

- Load bytecodes on demand
- Make “unloaded” empty BC
  - Points compiled code
- Load bytecode when it is needed
- To execute BC1, empty BC2 and BC3 are created, BC4 and BC5 is not created completely



Experiment

- Ubuntu 14.04.2 LTS on VirtualBox on Windows 7 on Intel i5-3380M (2.90GHz) CPU
- 1,400 lines Ruby script
    - 100 class definition
    - Each class has 3 simple methods
    - 401 bytecodes will be generated
- Ruby script and compiled code are already on memory (not from FS)
- Current implementation **copy** all data from compiled code

```
class C0
  def foo
    x = y = z = :hello
    p(x, y, z)
  end
  def bar
    x = y = z = :hello
    p(x, y, z)
  end
  def baz
    x = y = z = :hello
    p(x, y, z)
  end
end
…
```

Experiment
Load time

Class/method definitions are execution statements

	(Initial) Load time	Load + Execution	Execution
Parse+compile	7.05	8.42	1.37
Compiled code	2.22	3.41	1.19
Compiled code (lazy)	0.00	2.06	2.06

(seconds)

(result of repeating 2,000 times)

# 101 bytecodes (25%) are loaded by lazy load

Experiment
Compiled data

	Data
Ruby script lines	1,400 lines
Ruby script size	19,050 bytes
Classes	100 classes
Methods	300 methods
Compiled code	237,536 bytes
Compare with a script	**x12.5**

Experiment
Lazy load

- Run simple sample web application with 10 accesses
- Count loaded bytecodes and executed bytecode

Loaded bytecode	Executed bytecode	Ratio
30,485	4,698	**15.4%**

Discussion

- Only “15%” of bytecodes are needed …
    - Lazy load is good idea
    - We can consume loading time for each bytecodes, don’t need to use shared compiled data

  → We need to consider to use “Compaction techniques”

まとめ

- Rubyスクリプトを事前にコンパイル・後でロードする機能を試作
    - JVMクラスファイルみたいなものが欲しい、新規性はない
    - 実際に観察すると、遅延ロードによって、ロードするバイトコードは、実際には15%程度でいいのかも、という結果
    - 他プロセスとデータを共有するために、共有可能で性能があまり落ちないデータフォーマットを考えていたけど、単純に（15%だけ）コピーして展開するだけでよさそう

Thank you for your attention

Koichi Sasada

<ko1@heroku.com>

heroku