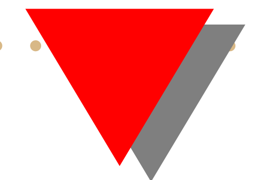


open-uri, Easy-to-Use and Extensible Virtual File System

Tanaka Akira

akr@m17n.org

Free Software Initiative Group,
Information Technology Research Institute,
National Institute of Advanced Industrial Science and Technology (AIST)

2005–10–14

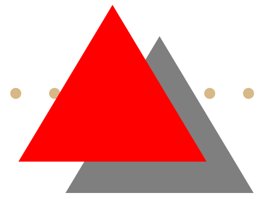
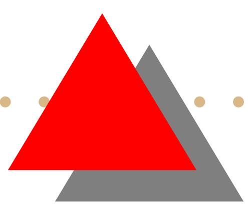
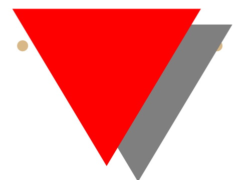


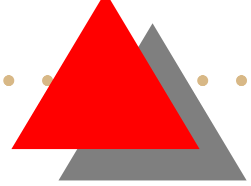
Table of Contents

- Who am I?
- How to use open-uri
- Why open-uri?
- open-uri and net/http
- How to design easy-to-use API
- Easy-to-use v.s. security
- VFS – Virtual File System





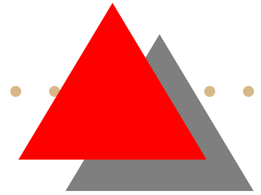
Who am I?



Who am I (1)

The author of open-uri and several standard libraries:

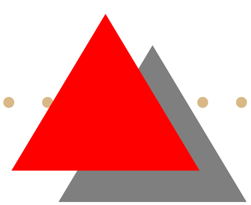
open-uri.rb, pathname.rb, time.rb, pp.rb, prettyprint.rb, resolv.rb, resolv-replace.rb, tsort.rb



Who am I (2)

Contribution for various classes and methods.

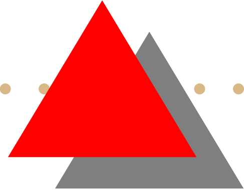
```
IO without stdio
IO#read and readpartial

Time       Time.utc       Time#utc_offset

allocate marshal_dump marshal_load

Regexp#to_s                Regexp.union

Process.daemon
fork kills all other threads
```



Who am I (3)

I reports many bugs, over 100/year.

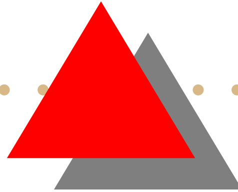
- core dump
- test failure
- build problem
- mismatch between doc. and imp.
- etc.

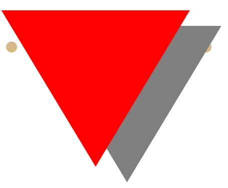


Who am I (4)

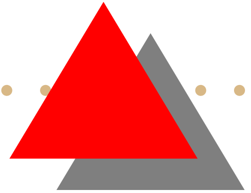
I wrote several non-standard libraries.

- htree
- webapp
- amarshal
- ruby-tzfile
- vfs-simple





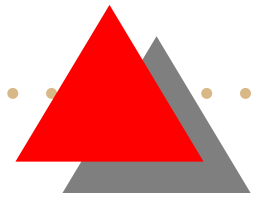
How to Use open-uri

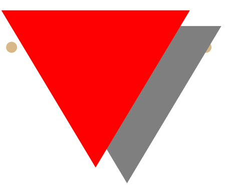


Simple Usage

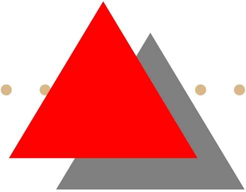
```
require 'open-uri'
open("http://www.ruby-lang.org"){
  |f|
  print f.read
}
```

Similar to open files.

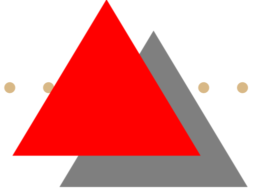


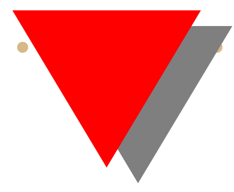


Why open-uri?



- Easy-to-Use API
- VFS: Not Only HTTP

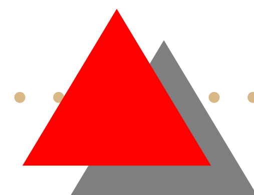




open-uri

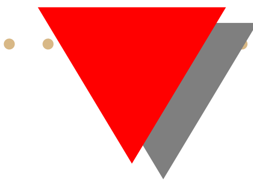
and

net/http



net/http has Too Many Ways

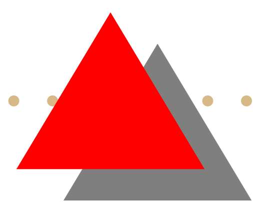
```
Net::HTTP.get_print
Net::HTTP.get
Net::HTTP.start{|h| h.get }
Net::HTTP.start{|h|
 h.request_get{|r|}}
h = Net::HTTP.new; h.start {|h|
 h.request_get {|r| } }
h = Net::HTTP.new; h.start {|h|
 q = HTTP::Get.new
 h.request(q){|r|}}
```



open-uri has Fewer Ways

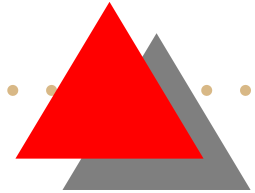
```
open(uri) {|f| }
uri.open {|f| }
uri.read
```

- Save User’s Memory
- Reuse User’s Knowledge



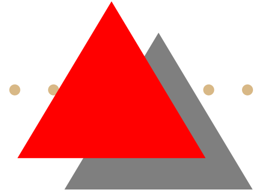
net/http: get and print

```
Net::HTTP.get_print(
  URI("http://host"))

print Net::HTTP.get(
  URI("http://host"))
```



open-uri: get and print

```
open("http://host") {|f|
  print f.read
}

print URI("http://host").read
```



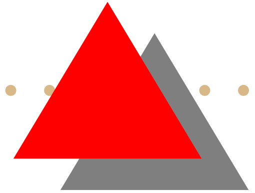
get and print

net/http:

- `Net::HTTP.get_print` : print only
- `Net::HTTP.get` : good

open-uri:

- `open` : conventional
- `URI().read` : short, polymorphism

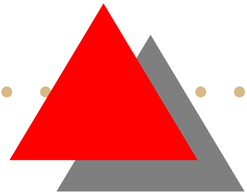


Why Easy?

open("http://host")

- No new construct
- Users don't need to learn.

open-uri respects user knowledges.



net/http: headers

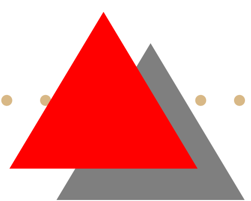
```
Net::HTTP.start("host") {|h|
  r = h.get("/",
            "User-Agent" => "bar")
  p r["Content-Type"]
  print r.body
}
```

- No URI anymore
- No Net::HTTP.get anymore
- Net::HTTP.start, Net::HTTP#get and Net::HTTP::Response#body instead.

open-uri: headers

```
open("http://host",
    "User-Agent" => "bar") {|f|
  p f.content_type
  print f.read
}
```

- Still URI
- Still open method
- Fewer things to learn.



net/http: SSL

```
require "net/https"
h = Net::HTTP.new("host", 443)
h.use_ssl = true
h.ca_file = "/etc/ssl/certs/ca-certificates.crt"
h.verify_mode = OpenSSL::SSL::VERIFY_PEER
h.start {
  r = h.get("/")
  print r.body
}
```

- Different library: net/https
- Net::HTTP.new and Net::HTTP#start
- Different port
- Server verification not by default

open-uri: SSL

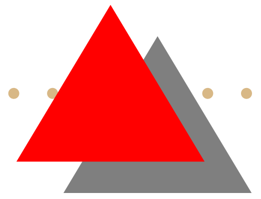
```
open("https://host") {|f|
  print f.read
}
```

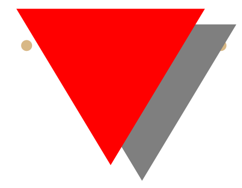
- Still URI
- Still open method
- Server verification by default
- No new library.
- No new methods. Fewer things to learn.

net/http: proxy

```
klass = Net::HTTP::Proxy("proxy", 8080)
klass.start("host") {|h|
  r = h.get("/")
  print r.body
}
```

- New method: Net::HTTP::Proxy

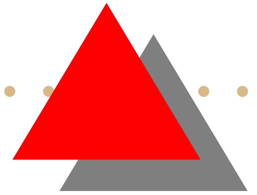




open-uri: proxy

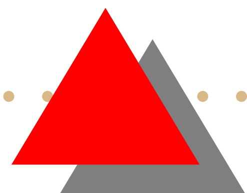
```
% http_proxy=http://proxy:8080/
% export http_proxy
```

- Conventional environment variable supported
- No new methods. An user might know this already.
- Fewer things to learn.



```
Net::HTTP.start("host") {|h|
  q = Net::HTTP::Get.new("/")
  q.basic_auth "user", "pass"
  r = h.request(q)
  print r.body
}
```

- New class: Net::HTTP::Get
- New method: Net::HTTP#request

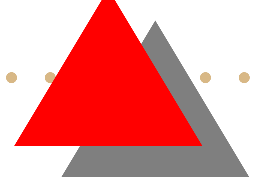


open-uri: basic auth

```
open("http://host",
  :http_basic_authentication=>
    ["user", "pass"]) {|f|
  print f.read
}
```

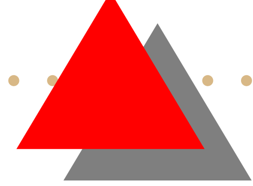
- Still URI
- Still open method
- New option: :http_basic_authentication
- No new methods. Fewer things to learn.

How to Design Easy-to-Use API



How to Design Easy-to-Use API

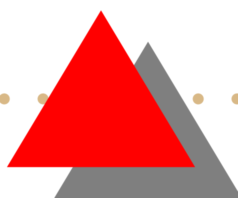
- Save brain power
- Evolve gradually



Save Brain Power

Fewer Things to Learn

- Fewer constructs for pragmatic usages
- Huffman coding
- DRY
- No configuration is good configuration
- Reuse user knowledge
- Infrastructure friendly



Fewer Constructs for Pragmatic Usages

Fewer constructs decrease things to learn

- open v.s. Net::HTTP.get, Net::HTTP#get, etc.
- This is not minimalism.
- The target of "fewer" is not all constructs.

Pragmatic usages should be supported by small constructs.

Fewer Constructs (2)

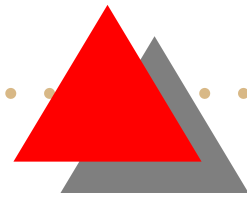
rarely used

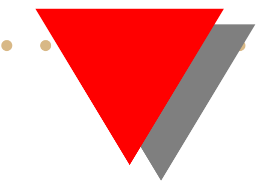
frequently used for pragmatic usages

primitives

convenience methods

should be smaller





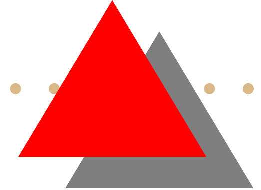
Ex. net/http and open-uri

Methods frequently used:

net/http Net::HTTP.start, Net::HTTP#get

open-uri open

open-uri's fewer constructs supports much more features.

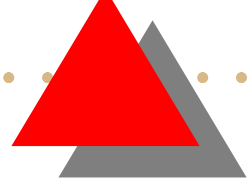


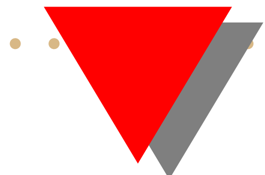
Huffman Coding

- Shorter for frequent things
- Longer for rare things

Optimize for frequent things.

Ex: p





Huffman Coding (2)

rarely used | frequently used

primitives

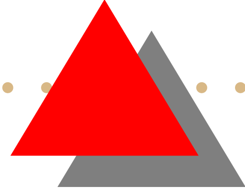
convenience methods

longer methods | shorter methods

Ex. p

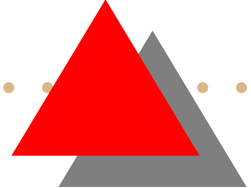
```
p obj
```

- Very frequently used
- Bad name in common sense
- Almost no problem because everyone knows



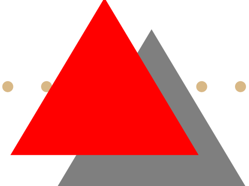
Ex. pp and y

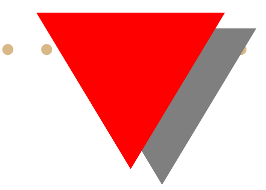
- Bad name in common sense
- Problematic than p because not everyone knows



to_s shorter. frequently used.

to_str longer. internal use.

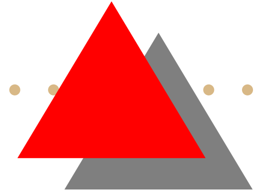


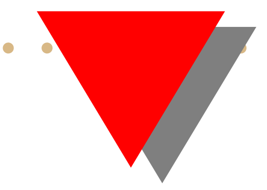


Ex. def

def shorter. frequently used.

define_method longer. not encouraged.



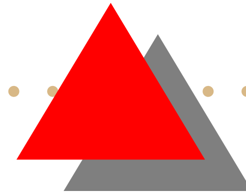


Ex. time.rb

Time.parse frequently used.

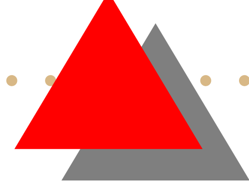
Time.strptime generic. needs to learn the format.

Time.parse is less flexible but enough for most cases, and easy to learn.



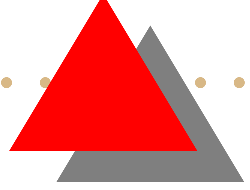
Candidates for Huffman Coding

- Method name
- Other name
- Convenience method
- Language syntax
- etc.



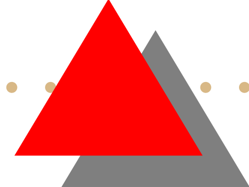
Length for Huffman Coding

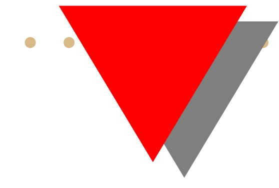
- Number of characters
- Number of nodes in AST
- Editor keystrokes
- etc.



Encourage Good Style

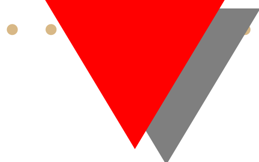
- Programmers like short code
- Short code should be designed as good style





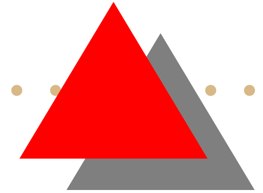
DRY

Don’t
Repeat
Yourself



DRY Violation

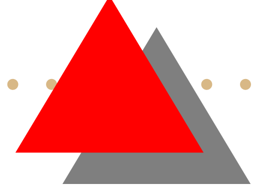
```
Net::HTTP.start("host") {|h|
  q = Net::HTTP::Get.new("/")
  q.basic_auth "user", "pass"
  r = h.request(q)
  print r.body
}
```



No Configuration is Good Configuration

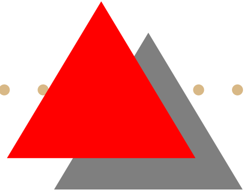
Things should be work well out-of-box.

- SSL CA certificates
- http_proxy environment variable



Bad Examples

- ext/iconv/config.charset
- soap_use_proxy
- require "irb/completion"
- RUBYOPT=rubygems

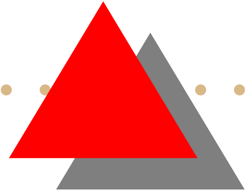


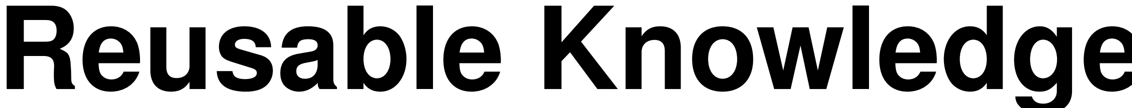
Reuse User Knowledge

open-uri reuse user knowledge.

- open is used to access an external resource
- If a block is given for open, it is called with a file object

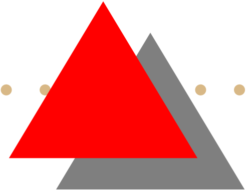
Various knowledge about open is reused.
Fewer things to learn.





Reusable Knowledge

- Ruby builtin (popular) method
- Consistency
- Unix
- Standards: POSIX, RFC, etc.
- Metaphor

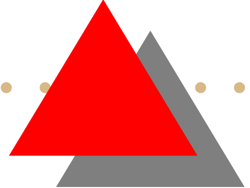


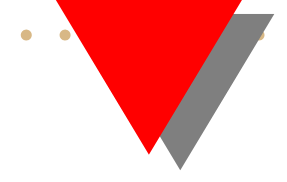
Consistency

- bang methods
- each_with_index
- etc.

Consistency violation:

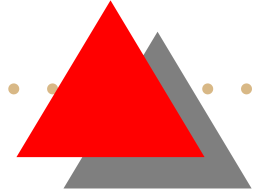
- Time#utc is destructive





Metaphor

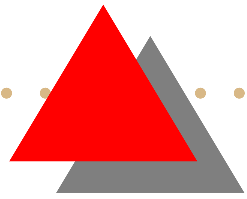
- HTTP is a kind of a network file system
- open-uri doesn’t support beyond file system: POST, etc



Infrastructure Friendly

- emacs, vi
- line oriented tools
- shell and file system
- web browser

Prefer
"It is easy using the legacy tool XXX"
over
"It is easy using the new tool YYY"



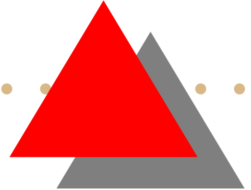
Evolve Gradually

- Adaptive Huffman coding
- How to find bad API
- How to avoid incompatibility
- Incompatible change

Adaptive Huffman Coding

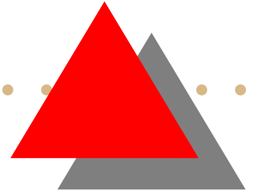
What methods are used frequently?

- Long method name at first
- Alias to short name later
- Define convenience methods for idioms

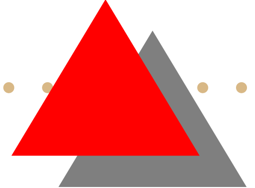


Adaptive Huffman Coding (2)

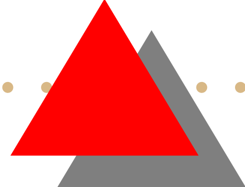
- Short names and operators should be used carefully
- Use a long name if hesitate
- Alias is not a bad thing (TMTOWTDI)
- Primitives should have long names
- Define new method for idiom



- CGI#[] and CGI#params
  CGI#[] was defined unsuitably.
- Hash#[]
  primitive: Hash#fetch



- Repeated surprise
- Often cannot remember
- Idiom

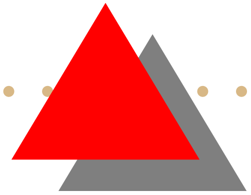


Repeated Surprise

Example:

- Time#utc is destructive
- Iconv.iconv returns an array
- String#gsub(/\\/, '\\\\') has no effect
- etc.

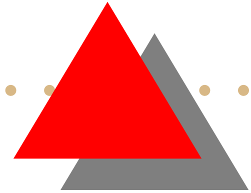
Violate POLS

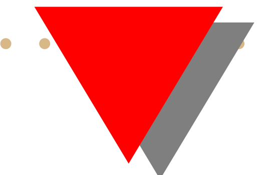


Often Cannot Remember

Manual is required again and again for same issue.

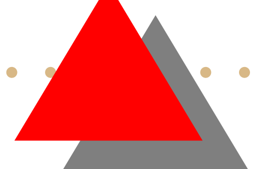
- RubyUnit
- optparse

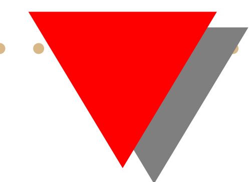




RubyUnit

```
require 'runit/testcase'
require 'runit/cui/testrunner'

class TestC < RUNIT::TestCase
  def test_unit
    ...
  end
end

RUNIT::CUI::TestRunner.run(
  TestC.suite)
```

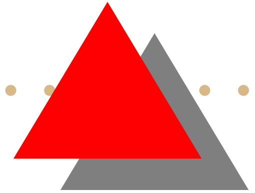




Test::Unit

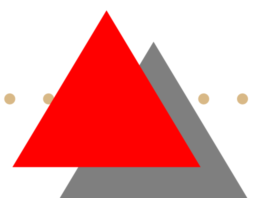
```
require 'test/unit'

class TestC <
          Test::Unit::TestCase
  def test_unit
    ...
  end
end
```

Test::Unit removed code for runner.



optparse

```
require 'optparse'
ARGV.options {|q|
  q.on("-h") { puts q }
  q.on("-v") { $VERBOSE = true }
  q.parse!
}
```

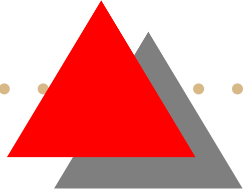


Idiom

- Repeated code
- Violate DRY
- An idiom may be good
- An idiom may be bad

Bad idiom example:

- Iconv.iconv()[0]

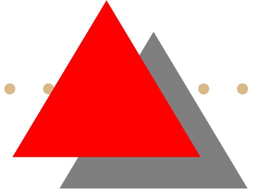


How to Avoid Incompatibility

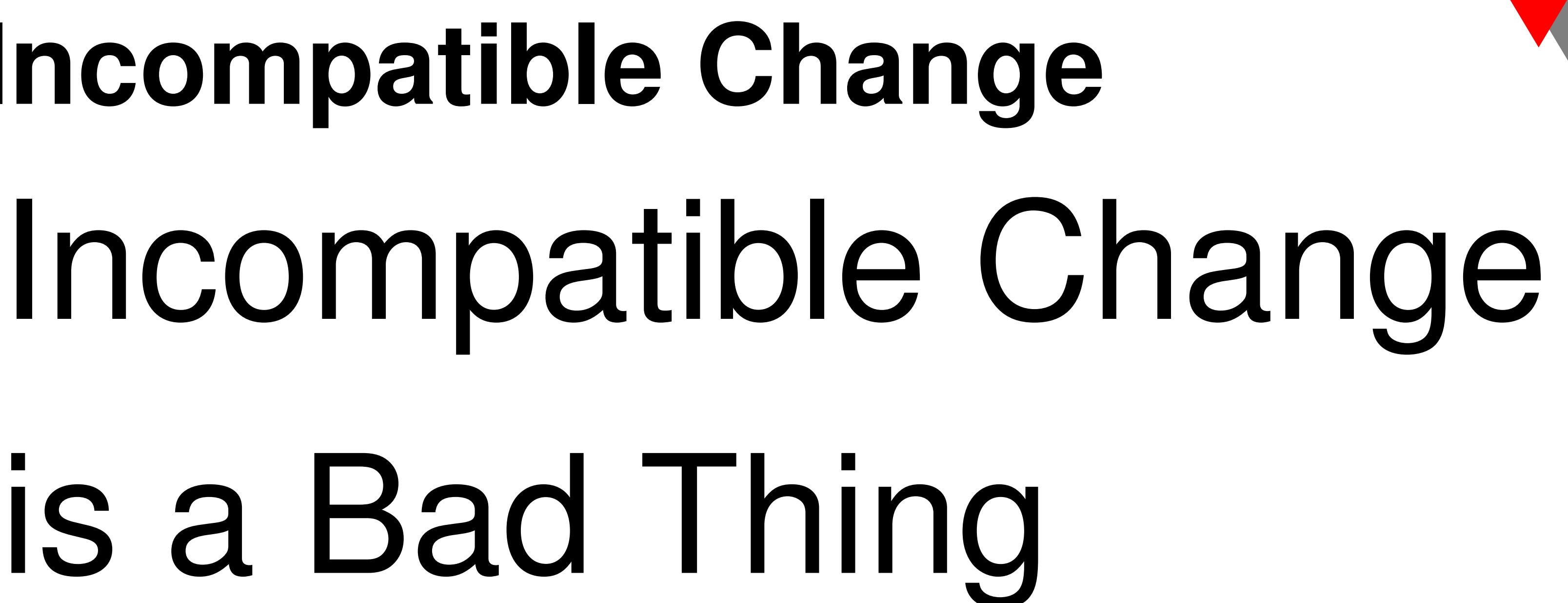
Extension without Incompatibility:

- new method
- new keyword argument
- new constants

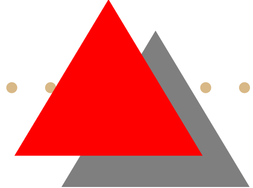
Introducing new names has no compatibility problem. (in most case)



Incompatible Change



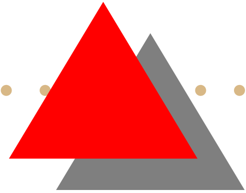
But fixing bad API involves incompatible change, sometimes.



Incompatible Change

API Migration Example

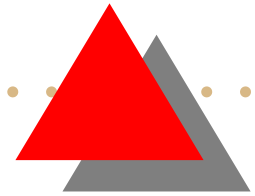
- net/http: API version
- cgi: special implementation for a transition period
- fork: warning after change
- IO#read: warning before change
- etc.



Net::HTTP has two APIs.

- Ruby 1.6: API version 1.1
- Ruby 1.7: API version 1.2

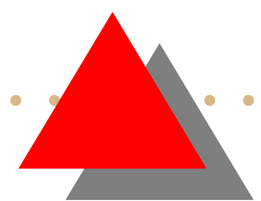
API version can be switched dynamically.



net/http: switch API version

```
Net::HTTP.version_1_1
... use 1.1 API ...

Net::HTTP.version_1_2
... use 1.2 API ...
```

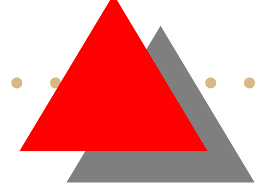
- It tends to forget restore API version
- Global switch – not thread safe



cgi: special implementation for a transition period

CGI#[] returns:

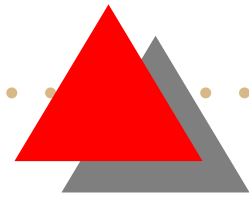
- Ruby 1.6: an array of parameters
- Ruby 1.8: transition period
- future?: a first parameter or nil



cgi: special implementation for a transition period

CGI#[] returns something tweaked on Ruby 1.8.
Try to work as both Array and String.

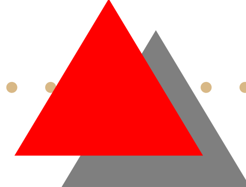
- Ruby 1.8.0: subclass of String
- Ruby 1.8.1: subclass of DelegateClass(String)
- Ruby 1.8.2: extended String



fork: warning after change

Does fork kill other threads in child process?

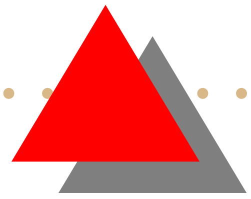
- Ruby 1.6: No
- Ruby 1.8: Yes



fork: warning after change

```
% ruby -e 'Thread.new{sleep};fork'
```

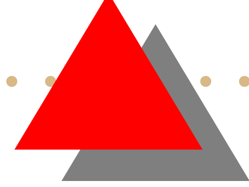
- Ruby 1.6: No warning
- Ruby 1.8.0: No warning
- Ruby 1.8.1: warning:
  fork terminates thread
- Ruby 1.8.2: No warning



IO#read: warning before change

IO#read will block even if O_NONBLOCK is set.

- Ruby 1.8: doesn't block
- Ruby 1.9: block

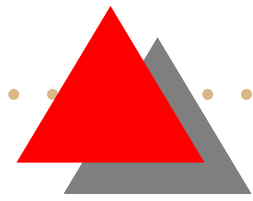


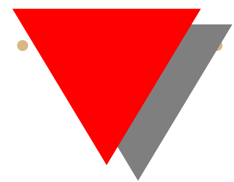
IO#read: warning before change

IO#read will block even if O_NONBLOCK is set.

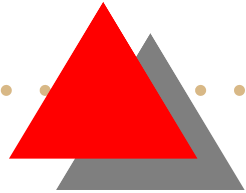
- Ruby 1.8.2: No warning
- Ruby 1.8.3: warning: nonblocking IO#read is obsolete; use IO#readpartial or IO#sysread
- Ruby 1.9: No warning

warning only if verbose mode.



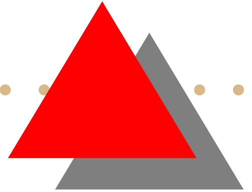


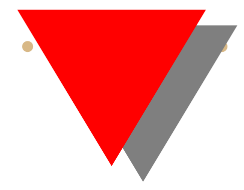
Easy-to-Use v.s. Security



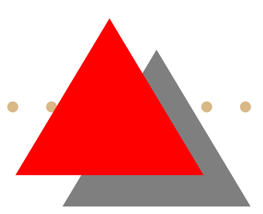
Easy-to-Use v.s. Security

- HTTP_PROXY
- http://user:pass@host/
- redirection and taint
- File.open(uri)



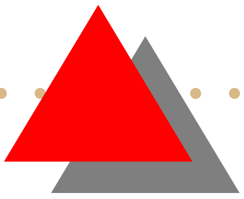


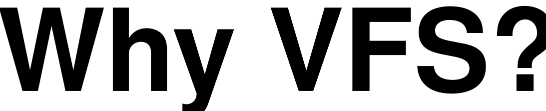
VFS
Virtual File System



VFS

- Why VFS?
- What is VFS
- VFS and polymorphism
- Polymorphic open
- Other Resources
- Other Operations
- Security Considerations



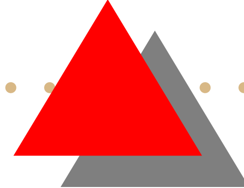


Why VFS?

Typical simple program:

- Load an external resource
- Process the resource
- Store the result

VFS ease the first step.

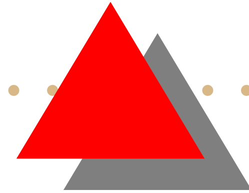


What is VFS

VFS provides:

- open a http/ftp/... resource
- read a http/ftp/... resource
- etc.

filesystem like operations for non-filesystem target
Polymorphism of filesystem

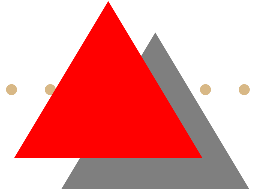


VFS and polymorphism

The polymorphism can be implemented by:

- usual method dispatch mechanism
- own mechanism

open-uri uses the method dispatch for the polymorphism.



Polymorphic open

If open-uri is in effect:

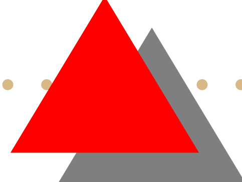
- open("http://...") calls URI("http://...").open
- open("ftp://...") calls URI("ftp://...").open
- etc.

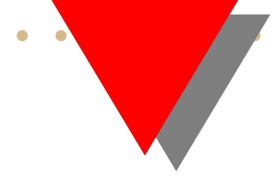
Any URI can be opened if the URI has open method.

Other Resources

LDAP:

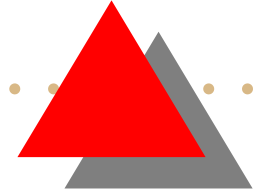
```
class URI::LDAP
  def open(*args)
    ...
  end
end

open("ldap://...") { ... }
```





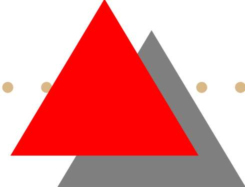
Other Operations

- URI("http://...").read
- Other operations should be defined for polymorphic to Pathname in future.



Security Considerations

- open("|...")
- File.open is not affected



Summary

- How to design Easy-to-Use API
  - Save brain power
  - Evolve gradually
- VFS by open-uri