krypt

the next level of ruby cryptography

martin boßlet
@_emboss_

ruby-core

openssl extension

diversity

basis for innovation

diversity

leads to the ability to choose

ruby cryptography

should **not** just be openssl

krypt

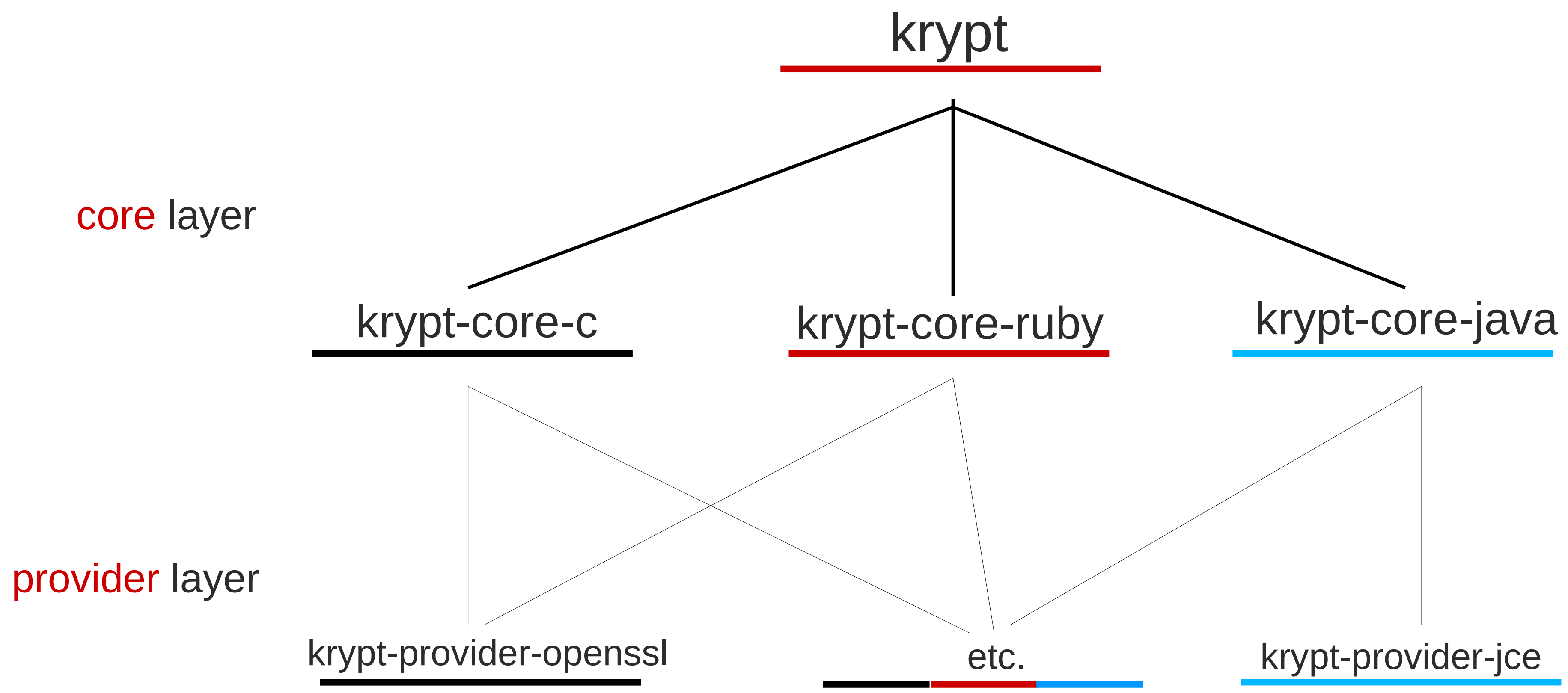
is also about **diversity**

platform- & library-independent

cryptography

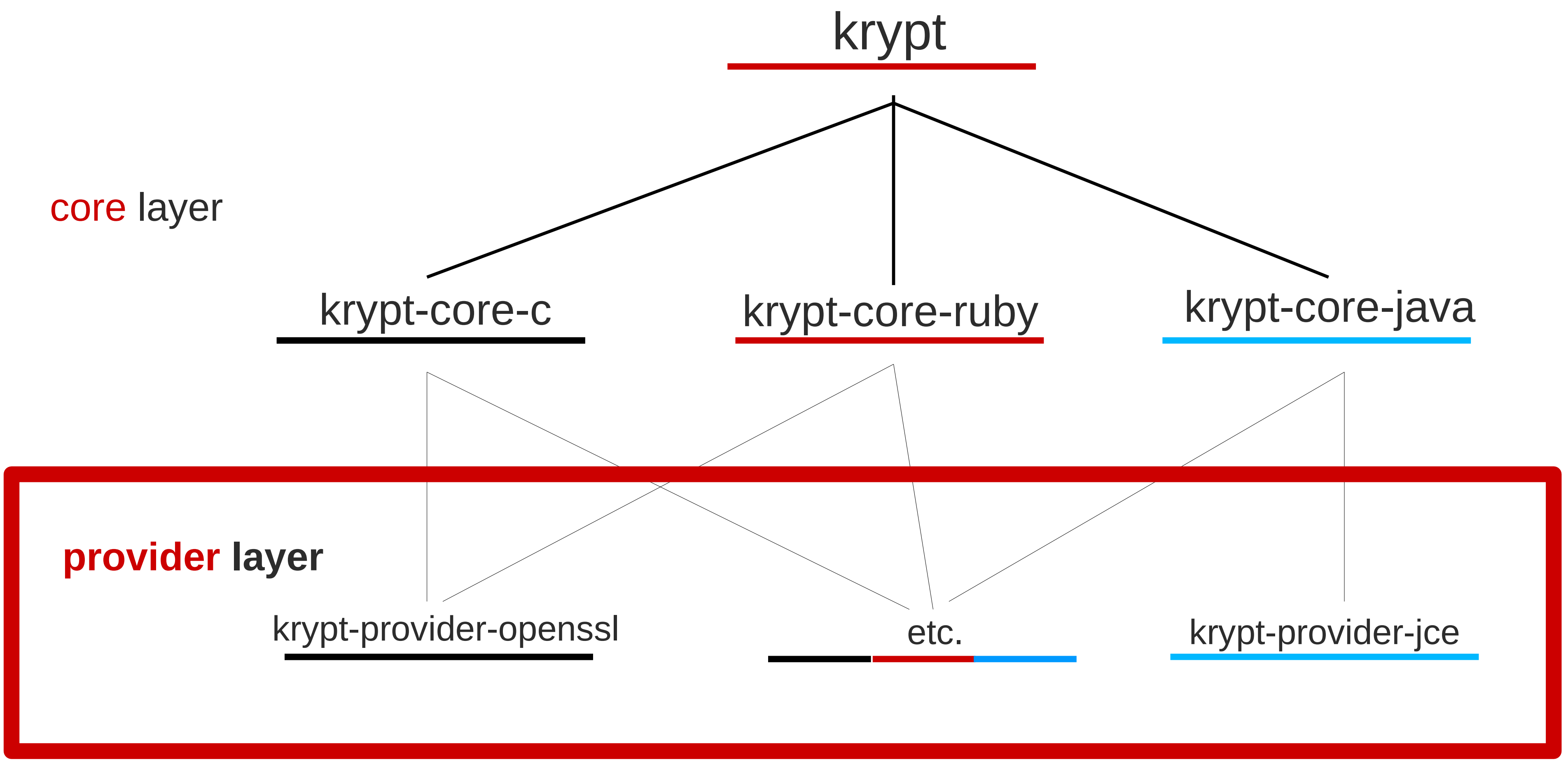
for ruby

replace openssl extension


krypt
core layer
krypt-core-c
krypt-core-ruby
krypt-core-java
provider layer
krypt-provider-openssl
etc.
krypt-provider-jce

each layer is a separate gem

combine as needed


krypt
core layer
krypt-core-c
krypt-core-ruby
krypt-core-java
provider layer
krypt-provider-openssl
etc.
krypt-provider-jce

krypt-provider

native c/java implementation

low-level primitives

well-defined interface

c: krypt-provider.h
java: KryptProvider

minimal

-› many implementations

partial implementations

for specific features

use multiple providers in parallel

openssl
capi (windows)
cng (windows)
commoncrypto (os x)
mozilla nss
nacl
...

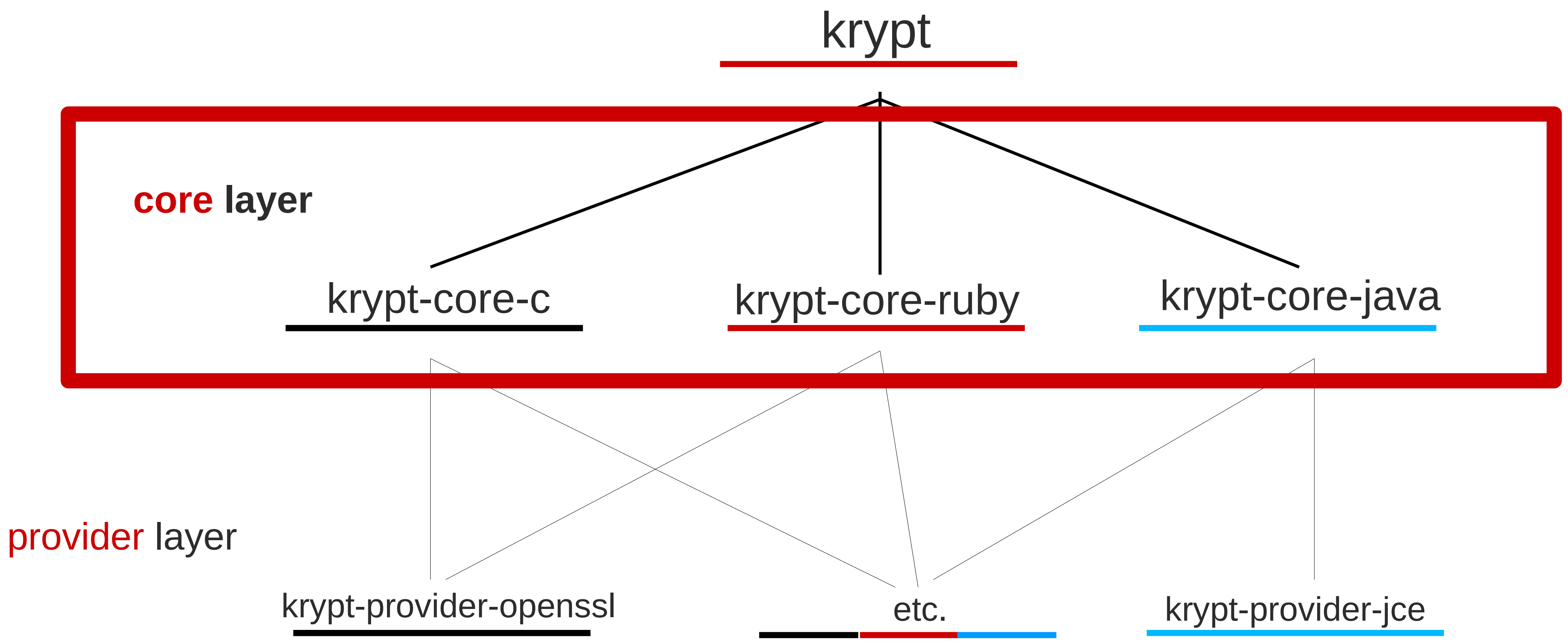
think of krypt-provider as

“writing an adapter for your

favorite crypto library”

future

all-ruby krypt-provider


krypt
core layer
krypt-core-c
krypt-core-ruby
krypt-core-java
provider layer
krypt-provider-openssl
etc.
krypt-provider-jce

krypt-core

ruby <-> krypt-provider

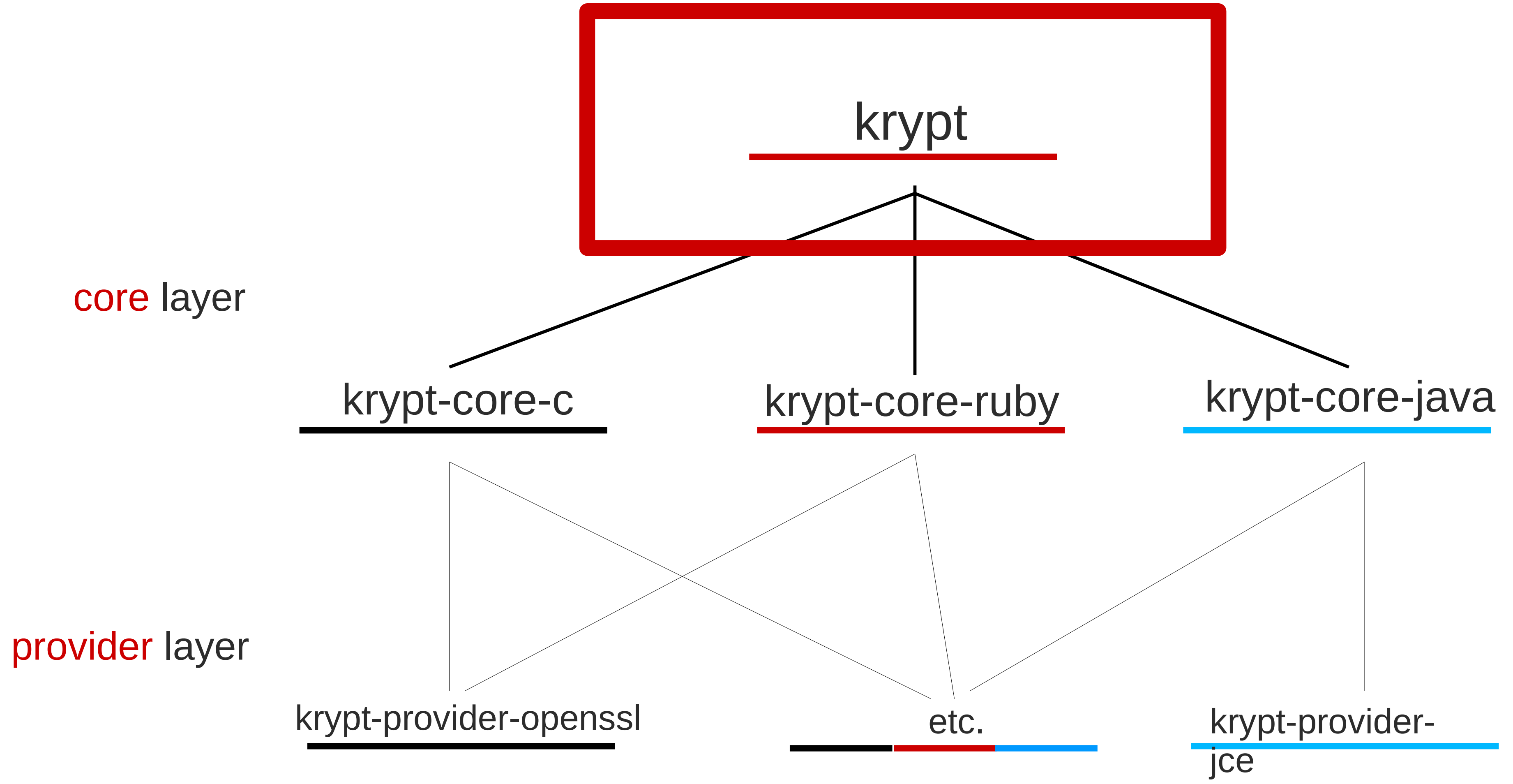
offers “**Provider**” api in ruby

native performance-critical parts

krypt-core-c

krypt-core-java

krypt-core-ruby


krypt
core layer
krypt-core-c
krypt-core-ruby
krypt-core-java
provider layer
krypt-provider-openssl
etc.
krypt-provider-jce

krypt

high-level cryptography

ruby only

pkcs#12
timestamps
cms (pkcs#7)
x.509 certificates
pkix certificate validation

and many more awesome acronyms...

design principles

ruby, as much as possible

run on all rubies, equally well

independence

stability

performance

comparable to native code

security by default

minimal effort to integrate new providers

fix problems of openssl

i.e. certificate validation

"The most dangerous code in the world"

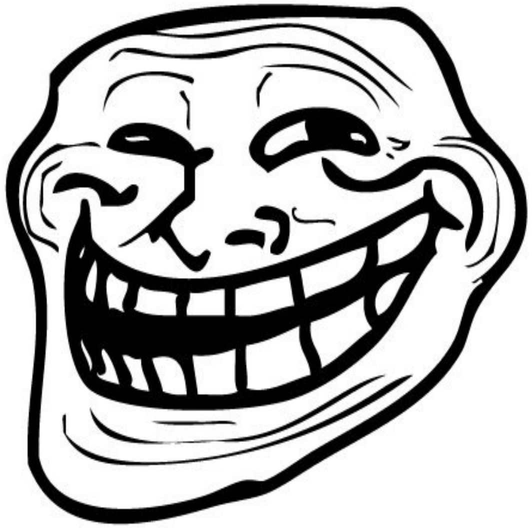
VERIFY_NONE
2003 - 2013
We fucking hate you anyway.

why break with integrating battle-tested c libraries?

like python, perl, erlang, haskell, lua

and (gasp!) php do?

“for crypto,



we need full control of everything!

we need c!”

i respectfully dissent

crypto is hard enough as it is

adding **memory** and **pointers**

is just asking for **trouble**

just look at the nature of recent openssl vulnerabilities need high-level language

giving us just enough control

ruby

fix **weaknesses** on the way (binyo)

showcase #1

asn.1/der

asn.1 / der

essential for most

cryptographic protocols

performance is essential

problem

streaming parsing & encoding

who has the **best** parsers?

java, enterprise ™ edition

event-based ?

pull parser!

pull instead of push

`parser.next_token`

token.io

streaming ✓

Krypt::ASN1::Template

dsl for asn.1 objects

```
class TBSCertificate
  include ASN1::Template::Sequence

  asn1_integer     :version, tag: 0, tagging: :EXPLICIT, default: 0
  asn1_integer     :serial
  asn1_template    :algorithm, ASN1::AlgorithmIdentifier
  asn1_template    :issuer, ASN1::DistinguishedName
  asn1_template    :validity, X509::Validity
  asn1_template    :subject, ASN1::DistinguishedName
  asn1_template    :subject_pkey, SubjectPublicKeyInfo
  asn1_bit_string  :issuer_id, tag: 1, tagging: :IMPLICIT, optional: true
  asn1_bit_string  :subject_id, tag: 2, tagging: :IMPLICIT, optional: true

  asn1_sequence_of :extensions, X509::Extension, tag: 3,
                   tagging: :EXPLICIT, optional: true
end
```

so far:

ad-hoc parsing/serialization

now:

Template::decode

Template#to_der

lazy parsing

cache original encoding

bouncy castle adopts

"indefinite length" encodings

ber encoding – not unique

openssl (and rest)

reencode

these on the fly to der

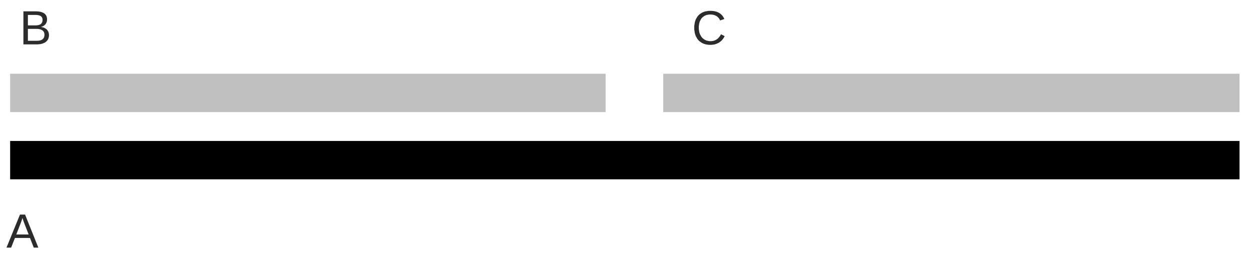
breaks signatures

caching is the **only** way

A consisting of B and C

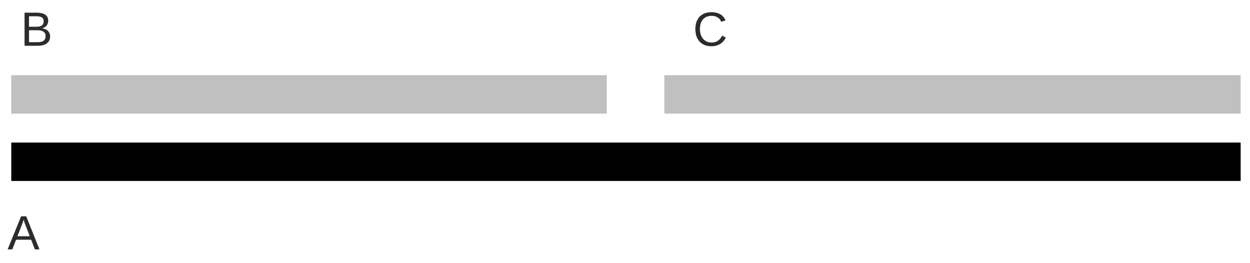
A := [B, C]

1. Parse A

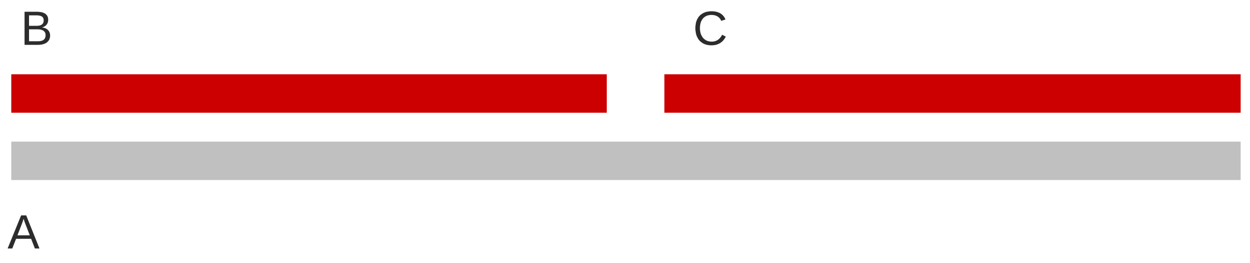


2. Encode again

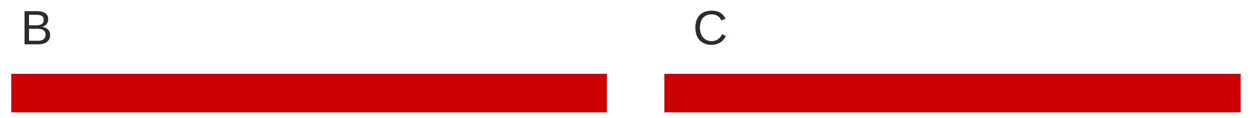
1. Parse A

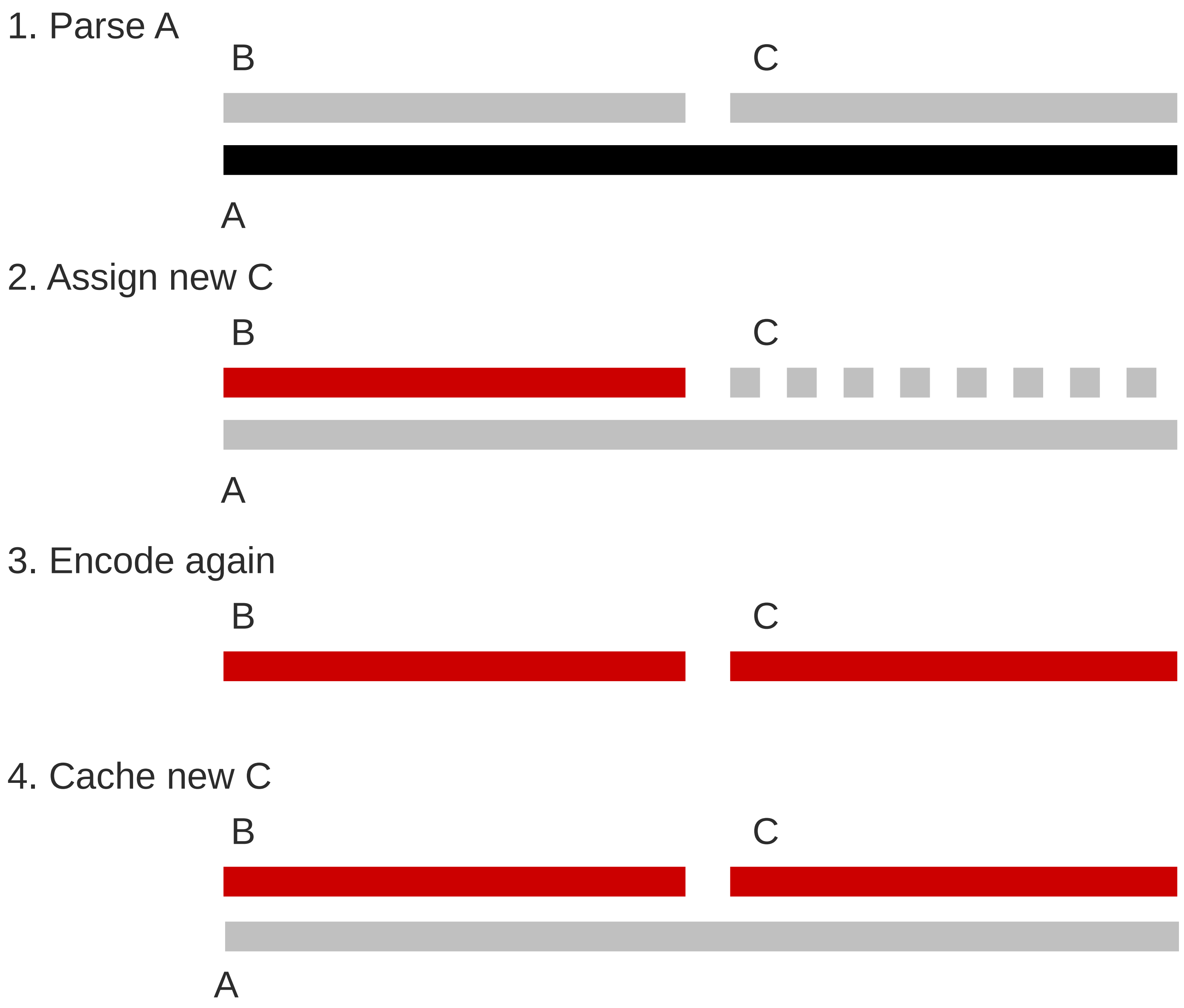


2. Access C or B



3. Encode again




1. Parse A
B
C
A
2. Assign new C
B
C
A
3. Encode again
B
C
4. Cache new C
B
C
A

cool #1

memory consumption stays ‹ 2x

cool #2

as lenient as possible

cool #3

performance

cruby (mri)

Krypt::Asn1::decode	0.395747
OpenSSL::Asn1::decode	6.914664
Krypt::X509::Certificate::parse	0.325850
OpenSSL::X509::Certificate::parse	4.306612

rubinius

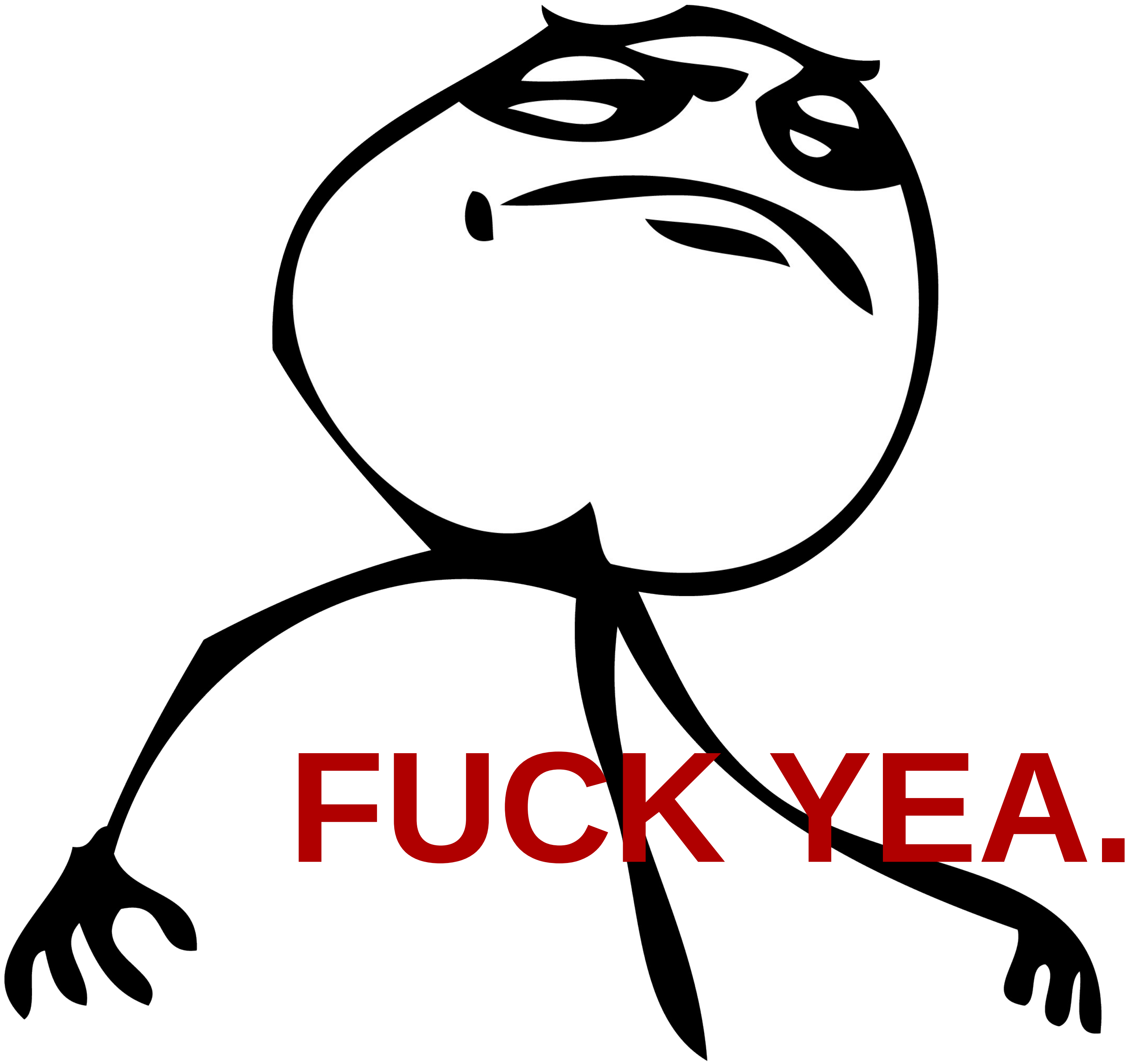
Krypt::Asn1::decode	0.475928
OpenSSL::Asn1::decode	33.436532
Krypt::X509::Certificate::parse	0.509898
OpenSSL::X509::Certificate::parse	4.808672

jruby

Krypt::Asn1::decode	0.294000
OpenSSL::Asn1::decode	8.022000
Krypt::X509::Certificate::parse	0.633000
OpenSSL::X509::Certificate::parse	12.688000

ruby vs. native

Krypt::X509::Certificate::parse	0.325850
OpenSSL native C	2.679
BouncyCastle Java 7u2	3.370648707
only one being competitive so far: JCE CertificateFactory Java 7u2	0.374258382


FUCK YEA.

no outliers

similar numbers for

cruby, jruby & rubinius

showcase #2

FuzzBert

testing is a #1 priority

extensive rspec specification

official **test vectors**

code coverage

(ruby, c & java)

travis ci

valgrind

we cannot test exhaustively

```
def le_test(arg)
```

need a heuristic

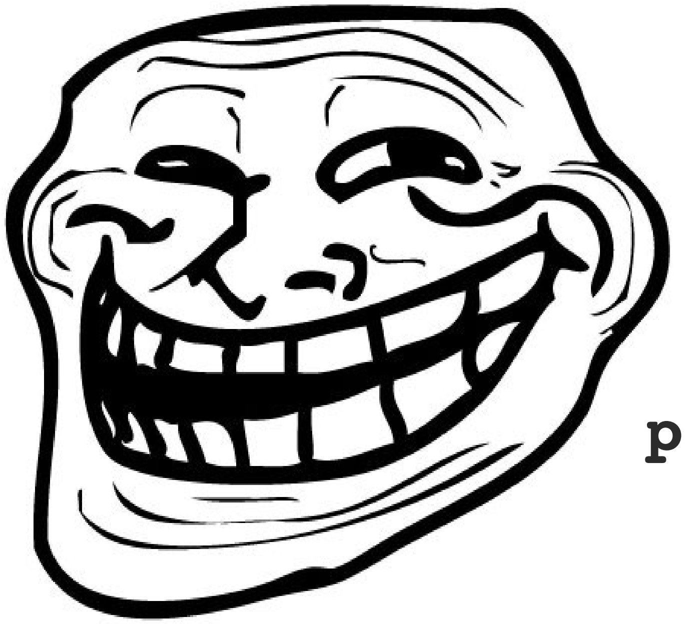
random testing / fuzzing

“yeah, it crashes with that…

but c’mon, that ain’t a bug”

“dude, nobody would ever…”

“come on, that ain’t **fair**”



problem?

random testing has no bias

finds the “**weird**” **cases**

that hackers & moms are looking for

cover more in less time

easy

completely random data

scratching on the surface

vs.

not getting edge cases

completely random

vs.

test cases with more structure

a good test suite needs both

FuzzBert

minimal framework

for

random testing

```ruby
fuzz "String.to_i" do

  deploy do |data|
    begin
      String.to_i(data)
    rescue StandardError
      # don't care, interested in hard crashes only
    end
  end

  data("completely random") { FuzzBert::Generators.random }

  data("1..1000") { FuzzBert::Generators.cycle(1..1000) }

  data "leading zero, fixed length of 100 digits" do
    c = FuzzBert::Container.new
    c << FuzzBert::Generators.fixed("0")
    c << FuzzBert::Generators.random_fixlen(99)
    c.generator
  end

end
```

template support

```ruby
fuzz "Web app JSON interface" do

  deploy do |data|
    # send JSON data via HTTP
  end

  data "template" do
    t = FuzzBert::Template.new <<-EOS
      {
        user: {
          id: ${id},
          name: "${name}"
        }
      }
    EOS

    t.set(:id) { FuzzBert::Generators.cycle(1..10000) }
    t.set(:name) { FuzzBert::Generators.random }
    t.generator
  end

end
```

runs tests in a **separate** process

happens in-memory

failed cases are persisted

does it work ?

hell yeah

http://www.udacity.com/course/cs258

not scientific

(neither is traditional testing)

good tests need

time
&
domain knowledge

but you want **science** ?

model failure arrival

exponential distribution

poisson processes

expected time until test fails

hypothesis testing

confidence intervals

→ stay or reject

“i heard you like random tests,

so i made random tests from

your random tests”

dream

completely automated tests

please, r people, jump on it

fuzz **all** the things

evolution of testing

showcase #3

binyo

but: breaking news

WHAT THE F*CK IS
HASHING?

not the hashing i meant

where are hashes used?

parser symbol table
method lookup table
attributes / instance variables
ip addresses in dns
transaction ids
database indexing
session ids
http headers
json representation
url-encoded post form data
deduplication (hashset)
a* search algorithm
dictionaries

where aren’t they used?

http://blog.headius.com/2012/09/avoiding-hash-lookups-in-ruby.html

the story so far

#hashDoS (2011)

easy to produce collisions

for most

general-purpose hash functions

problem has been known (2003)

http://static.usenix.org/event/sec03/tech/full_papers/crosby/crosby_html/

only fixed in perl at the time

last year

fix it for good

the fix

randomize the hash function

outlined in introduction to algorithms

universal hashing

the problem

universal hashing needs

hash function to be

pseudo-random

but they’re **not**

random seed is not good enough

simple example

$$f(seed) = 42$$

jean-philippe aumasson
(@aumasson)

daniel j. bernstein
(@hashbreaker)

found multicollisions for murmurhash

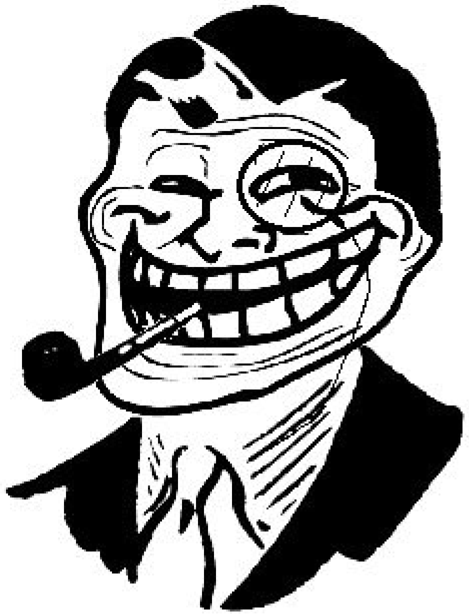
2 & 3

https://www.131002.net/siphash/

used in cruby, jruby and rubinius

the random seed doesn’t matter

produce collisions at will



le demo

not just ruby is affected

java, enterprise ™ edition

so what ?

web server

creates hashes from user input

worst-case behavior for n insertions

$O(n^2)$

what can we do ?

patch each and every library ?

Fix it

at the

root

one way out

light-weight cryptographic hash

more:

http://2012.appsec-forum.ch/conferences/#c17

more:

https://github.com/emboss/schadcode

let's **replace** openssl

thank you

https://github.com/krypt
https://github.com/emboss

http://emboss.github.com/blog

@_emboss_
martin.bosslet@gmail.com