Thrift

serviços para comunicação inter-linguagens

Bruno Atrib Zanchet
Yahoo! Brasil
UFPel

Remote Procedure Call

Comunicação inter-processos

Sintaxe “familiar”

“Distributed objects era of the 90s”

(91) Corba

(93) Microsoft COM

(97) Java RMI

(98) XML-RPC (depois SOAP)

(99) EJB

overhead

REST

Roy Fielding, 2000

“estado” abstraído para “recurso”

sintaxe universal para links

operações e content-types bem definidos

stateless, layered, cacheable

Thrift

Facebook, 2007

RPC

(de novo)

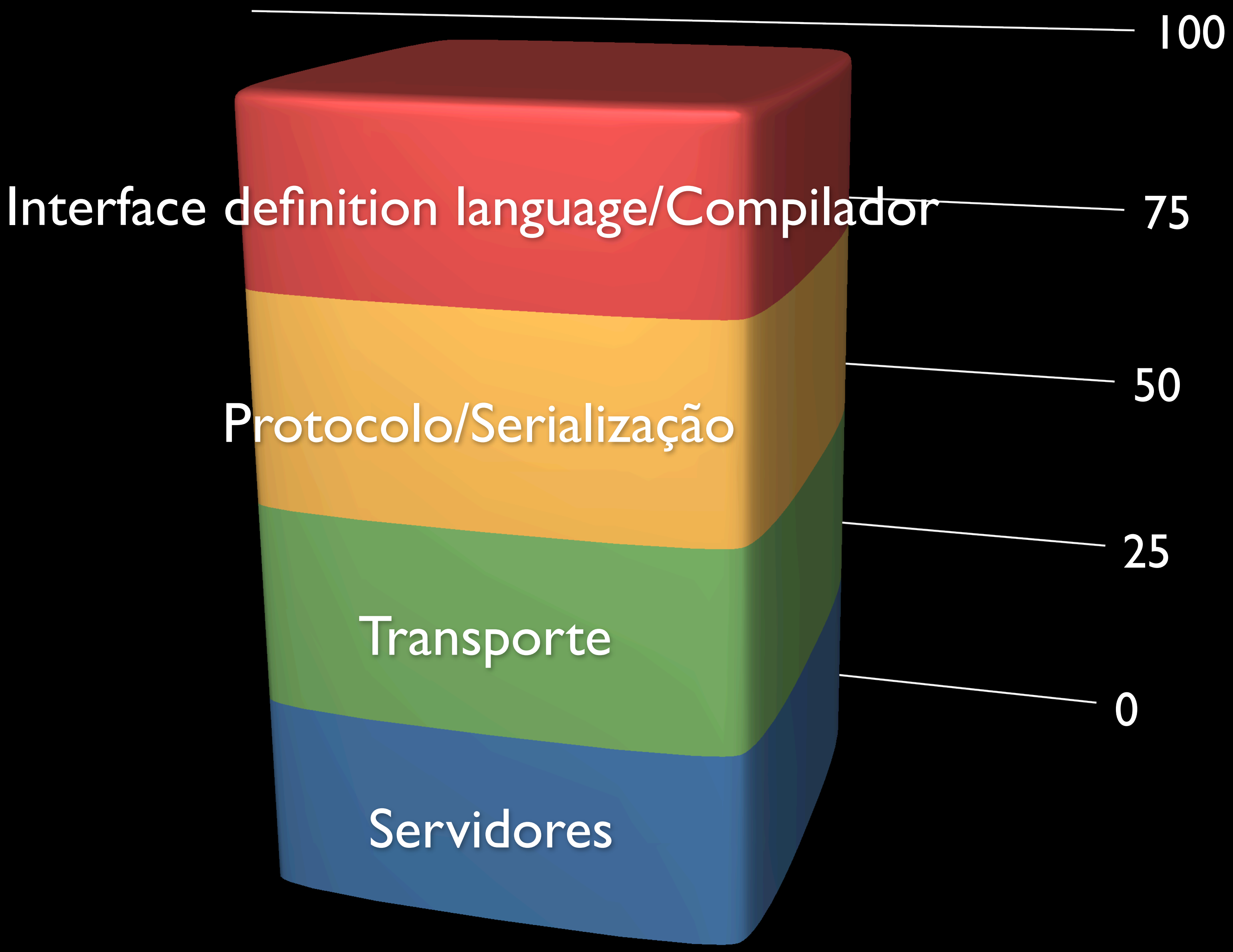
Adotado pelo Apache Incubator (2008)

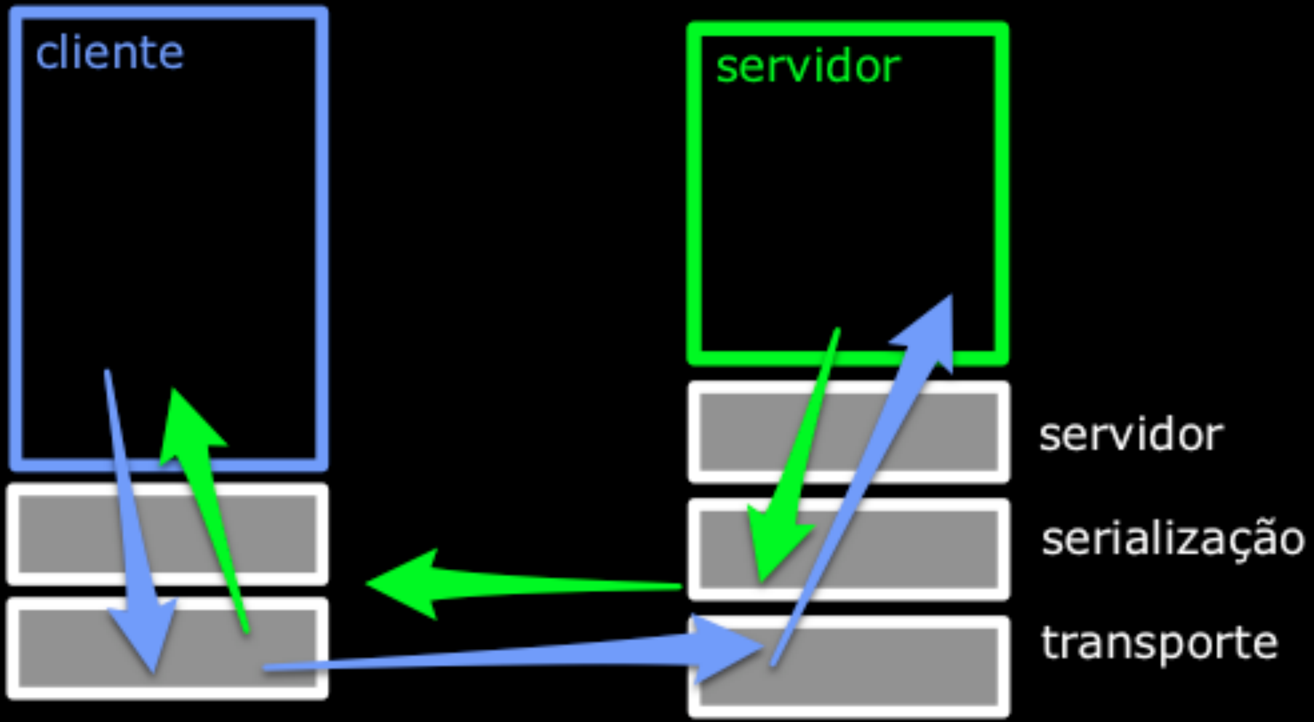
C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, Smalltalk, OCaml, ...

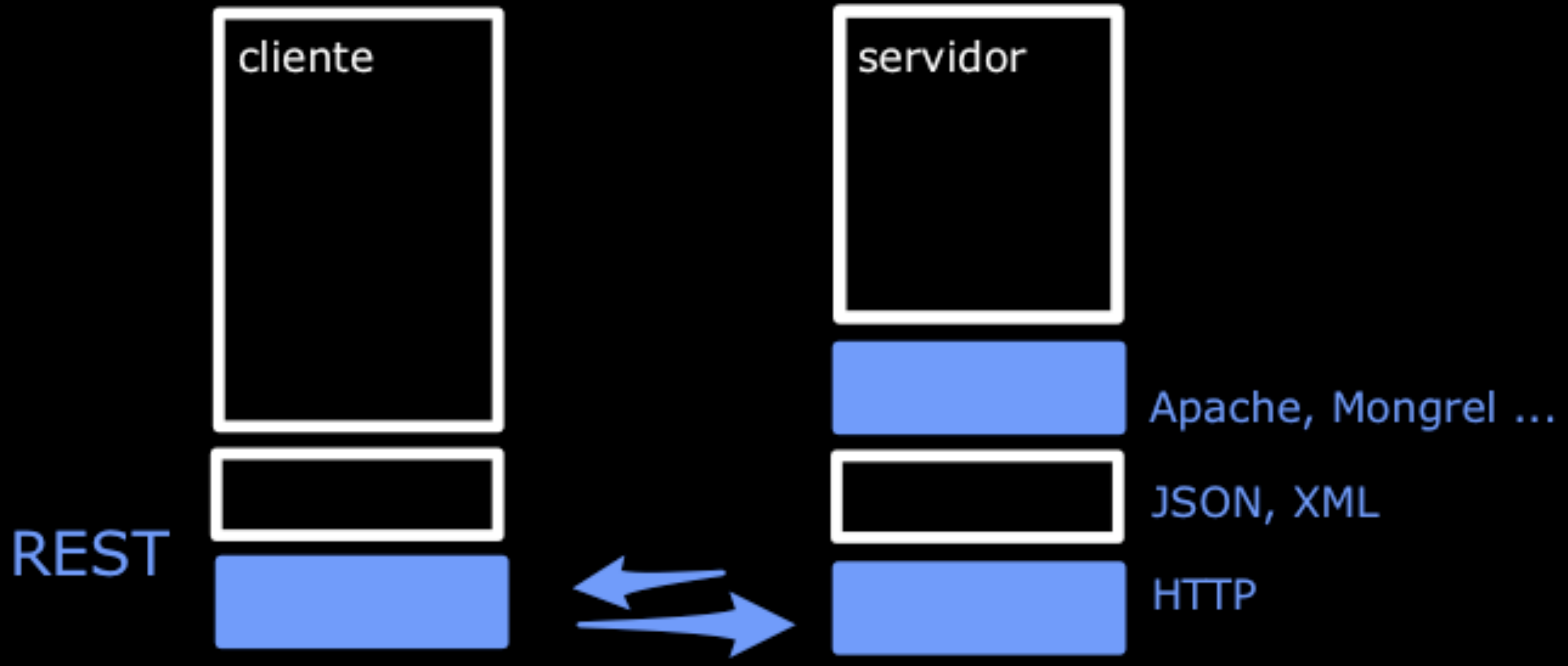
“Rápido, realmente rápido”

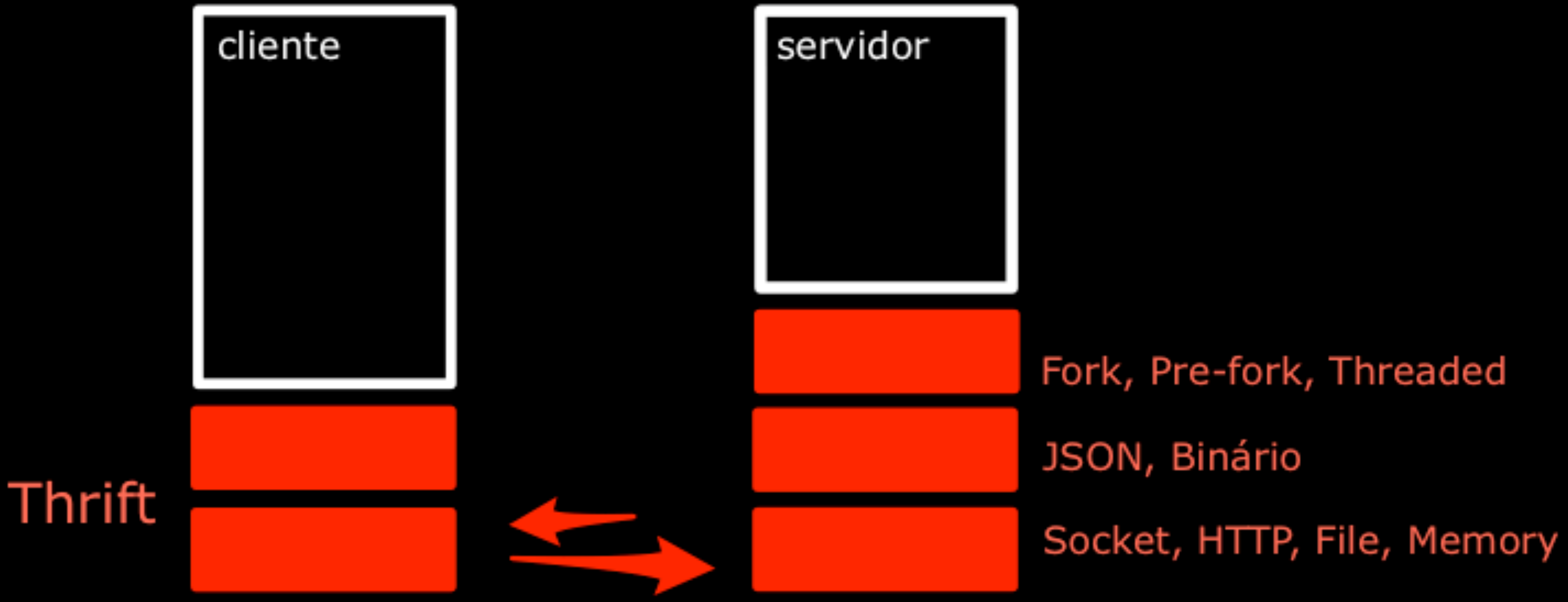
hein?

Thrift


100
Interface definition language/Compilador
75
Protocolo/Serialização
50
25
Transporte
0
Servidores


cliente
servidor
servidor
serialização
transporte


cliente
servidor
REST
Apache, Mongrel ...
JSON, XML
HTTP


cliente
servidor
Thrift
Fork, Pre-fork, Threaded
JSON, Binário
Socket, HTTP, File, Memory

Interface Definition Language

compilador/gerador de código

Tipos

Traduzidos em tipos “nativos”

Sem tipos especiais ou wrappers

Básicos

bool

byte

i16, i32, i64

double

string

Containers

list<type>

set<type>

map<type1, type2>

list<type> →

[ ]	Ruby/Python
array( )	PHP
STL vector	C++
ArrayList	Java

Structs

```
struct Example {
  1:i32      number=10,
  2:i64      big_number,
  3:double   decimal,
  4:string   name="thrifty"
}

```

```
# examples.thrift
```

```ruby
class Example
  include ::Thrift::Struct

  ::Thrift::Struct.field_accessor self, :number, :big_number, :decimal, :name

  ...
end
```

```python
class Example(object):

  def __init__(self, number=10, big_number=None, decimal=None, name="thrifty",):
    self.number = number
    self.big_number = big_number
    self.decimal = decimal
    self.name = name

  ...
```

```php
<?php
class Example {
  public $number = 10;
  public $big_number = null;
  public $decimal = null;
  public $name = "thrifty";

  public function __construct($vals=null) {
    if (is_array($vals)) {
      if (isset($vals['number'])) {
        $this->number = $vals['number'];
      }
      if (isset($vals['big_number'])) {
        $this->big_number = $vals['big_number'];
      }
      if (isset($vals['decimal'])) {
        $this->decimal = $vals['decimal'];
      }
      if (isset($vals['name'])) {
        $this->name = $vals['name'];
      }
    }
  }
}
?>
```

```java
public class Example implements TBase, java.io.Serializable, Cloneable {
  public int number;
  public long big_number;
  public double decimal;
  public String name;

  public Example() {
    this.number = 10;
    this.name = "thrifty";
  }

  public Example(int number, long big_number, double decimal, String name)
  {
    this();
    this.number = number;
    this.big_number = big_number;
    this.decimal = decimal;
    this.name = name;
  }

  ...
}
```

Exceções

```
exception ExampleException {
  1:i32     number=10,
  2:i64     big_number,
  3:double  decimal,
  4:string  name="thrifty"
}

```

```
# examples.thrift
```

Python	`class ExampleException(Exception):`
Java	`public class ExampleException extends Exception {`
PHP	`class ExampleException extends TException {`
Ruby	`class ExampleException < ::Thrift::Exception`

Serviços

```
service RemoteHashMap {
    void          set(1:i32 key, 2:string value),
    string        get(1:i32 key) throws (1: KeyNotFound knf),
    async void delete(1:i32 key)
}

```

```
# examples.thrift
```

Protocolo

Métodos para leitura e escrita

Encoding dos tipos básicos, structs e containers

TProtocol

```
readMessageBegin()
writeMessageBegin(name, type, seq)
readStructBegin()
writeStructBegin(name)
readI16()
writeI16()
readI32()
...
```

TBinaryProtocol

TCompactProtocol

TJSONProtocol

...

Transporte

Transferência de dados

Duas interfaces

TTransport	TServerTransport
open	open
close	listen
isOpen	accept
read	close
write	
flush	

TSocket

TFileTransport

TMemoryBuffer

THttpClient

...

Servidores

TThreadedServer

TThreadPoolServer

TForkingServer

...

Fim.

WS-*, alguém?

SOAP (formerly known as Simple Object Access Protocol)
SOAP Message Transmission Optimization Mechanism
WS-Notification
WS-BaseNotification
WS-Topics
WS-BrokeredNotification
WS-SoapOverUDP
WS-Addressing
WS-Transfer
WS-Eventing
WS-Enumeration
WS-MakeConnection
WS-Policy
WS-PolicyAssertions
WS-PolicyAttachment
WS-Discovery
WS-Inspection
WS-MetadataExchange
Universal Description, Discovery, and Integration (UDDI)
WSDL 2.0 Core
WSDL 2.0 SOAP Binding
Web Services Semantics (WSDL-S)
WS-Resource Framework (WSRF)
WS-Security
XML Signature
XML Encryption
XML Key Management (XKMS)
WS-SecureConversation
WS-SecurityPolicy
WS-Trust
WS-Federation
WS-Federation Active Requestor Profile
WS-Federation Passive Requestor Profile
Web Services Security Kerberos Binding
Web Single Sign-On Interoperability Profile
Web Single Sign-On Metadata Exchange Protocol
Security Assertion Markup Language (SAML)
XACML
ISO/IEC 20000-2:2005 Information technology

P3P
WS-ReliableMessaging
WS-Reliability
WS-RM Policy Assertion
Web Services Resource Framework
WS-BaseFaults
WS-ServiceGroup
WS-ResourceProperties
WS-ResourceLifetime
WS-Transfer
Resource Representation SOAP Header Block
WS-I Basic Profile
WS-I Basic Security Profile
Simple Soap Binding Profile
WS-BPEL
WS-CDL
Web Services Choreography Interface
WS-Choreography
XML Process Definition Language
WS-BusinessActivity
WS-AtomicTransaction
WS-Coordination
WS-CAF
WS-Transaction
WS-Context
WS-CF
WS-TXM
WS-Management
WS-Management Catalog
WS-ResourceTransfer
WSDM
Web Services for Remote Portlets
WS-Provisioning
Devices Profile for Web Services (DPWS)
ebXML
ISO/IEC 19784-2:2007 Information technology
ISO 19133:2005 Geographic information
ISO/IEC 20000-1:2005 Information technology
ISO/IEC 25437:2006 Information technology

Na prática

1. Definir as estruturas

```

enum MartialArt {
  AIKIDO    = 1,
  KARATE   = 2
}

struct UserProfile {
  1: i32            uid,
  2: string         name,
  3: MartialArt     style
}

service UserStorage {
  void          store(1: UserProfile user),
  UserProfile   retrieve(1: i32 uid)
}

```

```
# service.thrift
```

1. Definir as estruturas

```

enum MartialArt {
  AIKIDO     = 1,
  KARATE    = 2
}

struct UserProfile {
  1: i32               uid,
  2: string            name,
  3: MartialArt        style
}

service UserStorage {
  void             store(1: UserProfile user),
  UserProfile    retrieve(1: i32 uid)
}

```

Enum!

# service.thrift

2. Gerar código “stub”

```
$ thrift --gen rb --gen java service.thrift
```

2. Gerar código “stub”

```
$ thrift --gen php --gen py:new_style service.thrift
```

3. Implementar lógica do serviço

```

class UserStorageHandler:
    def __init__(self):
        pass

    def store(self, user):
        print "stored " + str(user)

    def retrieve(self, id):
        print "retrieved " + str(id)
        return UserProfile(
            uid=id,
            name="Ralph Waldo Emerson",
            style=MartialArt.KARATE
    )

```

```
# server.py
```

3. Implementar lógica do serviço

```python

class UserStorageHandler:
    def __init__(self):
        pass

    def store(self, user):
        print "stored " + str(user)

    def retrieve(self, id):
        print "retrieved " + str(id)
        return UserProfile(
            uid=id,
            name="Ralph Waldo Emerson",
            style=MartialArt.KARATE
    )

```

código gerado!

```
# server.py
```

4. Escolher o servidor

```

handler = UserStorageHandler()
processor = example.UserStorage.Processor(handler)
transport = TSocket.TServerSocket(9090)
tfactory = TTransport.TBufferedTransportFactory()
pfactory = TBinaryProtocol.TBinaryProtocolFactory()

server = TServer.TThreadedServer(processor, transport, tfact...

print 'Starting the server...'
server.serve()

```

```
# server.py
```

5. Implementar o cliente

```php
$socket = new TSocket('localhost', 9090);
$transport = new TBufferedTransport($socket, 1024, 1024);
$protocol = new TBinaryProtocol($transport);
$client = new UserStorageClient($protocol);

$transport->open();

$user = $client->retrieve(1);
var_dump($user);

$new_user = new example_UserProfile(array(
  "uid" => '123',
  "name" => "Ralph Waldo Emerson",
  "style" => example_MartialArt::KARATE
));
$client->store($new_user);

$transport->close();
```

```
# client.php
```

5. Implementar o cliente

```php
$socket = new TSocket('localhost', 9090);
$transport = new TBufferedTransport($socket, 1024, 1024);
$protocol = new TBinaryProtocol($transport);
$client = new UserStorageClient($protocol);

$transport->open();

$user = $client->retrieve(1);
var_dump($user);

$new_user = new example_UserProfile(array(
  "uid" => '123',
  "name" => "Ralph Waldo Emerson",
  "style" => example_MartialArt::KARATE
));
$client->store($new_user);

$transport->close();
```

código gerado!

# client.php

5. Implementar o cliente

```php
$socket = new TSocket('localhost', 9090);
$transport = new TBufferedTransport($socket, 1024, 1024);
$protocol = new TBinaryProtocol($transport);
$client = new UserStorageClient($protocol);

$transport->open();

$user = $client->retrieve(1);
var_dump($user);

$new_user = new example_UserProfile(array(
  "uid" => '123',
  "name" => "Ralph Waldo Emerson",
  "style" => example_MartialArt::KARATE
));
$client->store($new_user);

$transport->close();
```

código gerado!

```
# client.php
```

6. Deploy!

```
$ python server.py
```

?!

invocação remota

```
# client.php

$transport->open();

$user = $client->retrieve(1);
var_dump($user);

$new_user = new example_UserProfile(array(
  "uid" => '123',
  "name" => "Ralph Waldo Emerson",
  "style" => example_MartialArt::KARATE
));
$client->store($new_user);

$transport->close();
```

```
# server.py

class UserStorageHandler:
    def __init__(self):
        pass

    def store(self, user):
        print "stored " + str(user)

    def retrieve(self, id):
        print "retrieved " + str(id)
        return UserProfile(
            uid=id,
            name="Ralph Waldo Emerson",
            style=MartialArt.KARATE
    )

```

mágica!

```php
# client.php

$transport->open();

$user = $client->retrieve(1);
var_dump($user);

$new_user = new example_UserProfile(array(
  "uid" => '123',
  "name" => "Ralph Waldo Emerson",
  "style" => example_MartialArt::KARATE
));
$client->store($new_user);

$transport->close();
```

```python
# server.py

class UserStorageHandler:
    def __init__(self):
        pass

    def store(self, user):
        print "stored " + str(user)

    def retrieve(self, id):
        print "retrieved " + str(id)
        return UserProfile(
            uid=id,
            name="Ralph Waldo Emerson",
            style=MartialArt.KARATE
    )

```

```
# client.php

$transport->open();

$user = $client->retrieve(1);
var_dump($user);

$new_user = new example_UserProfile(array(
  "uid" => '123',
  "name" => "Ralph Waldo Emerson",
  "style" => example_MartialArt::KARATE
));
$client->store($new_user);

$transport->close();
```

```
# server.py

class UserStorageHandler:
    def __init__(self):
        pass

    def store(self, user):
        print "stored " + str(user)

    def retrieve(self, id):
        print "retrieved " + str(id)
        return UserProfile(
            uid=id,
            name="Ralph Waldo Emerson",
            style=MartialArt.KARATE
    )

```

... e caímos aqui

```php
# client.php

$transport->open();

$user = $client->retrieve(1);
var_dump($user);

$new_user = new example_UserProfile(array(
  "uid" => '123',
  "name" => "Ralph Waldo Emerson",
  "style" => example_MartialArt::KARATE
));
$client->store($new_user);

$transport->close();
```

```python
# server.py

class UserStorageHandler:
    def __init__(self):
        pass

    def store(self, user):
        print "stored " + str(user)

    def retrieve(self, id):
        print "retrieved " + str(id)
        return UserProfile(
            uid=id,
            name="Ralph Waldo Emerson",
            style=MartialArt.KARATE
    )

```

return

mágica!

```php
# client.php

$transport->open();

$user = $client->retrieve(1);
var_dump($user);

$new_user = new example_UserProfile(array(
  "uid" => '123',
  "name" => "Ralph Waldo Emerson",
  "style" => example_MartialArt::KARATE
));
$client->store($new_user);

$transport->close();
```

```python
# server.py

class UserStorageHandler:
    def __init__(self):
        pass

    def store(self, user):
        print "stored " + str(user)

    def retrieve(self, id):
        print "retrieved " + str(id)
        return UserProfile(
            uid=id,
            name="Ralph Waldo Emerson",
            style=MartialArt.KARATE
    )

```

de volta!

```
# client.php

$transport->open();

$user = $client->retrieve(1);
var_dump($user);

$new_user = new example_UserProfile(array(
  "uid" => '123',
  "name" => "Ralph Waldo Emerson",
  "style" => example_MartialArt::KARATE
));
$client->store($new_user);

$transport->close();
```

```
# server.py

class UserStorageHandler:
    def __init__(self):
        pass

    def store(self, user):
        print "stored " + str(user)

    def retrieve(self, id):
        print "retrieved " + str(id)
        return UserProfile(
            uid=id,
            name="Ralph Waldo Emerson",
            style=MartialArt.KARATE
    )

```

```
# client.php

$transport->open();

$user = $client->retrieve(1);
var_dump($user);

$new_user = new example_UserProfile(array(
  "uid" => '123',
  "name" => "Ralph Waldo Emerson",
  "style" => example_MartialArt::KARATE
));
$client->store($new_user);

$transport->close();
```

invocação remota

```
# server.py

class UserStorageHandler:
    def __init__(self):
        pass

    def store(self, user):
        print "stored " + str(user)

    def retrieve(self, id):
        print "retrieved " + str(id)
        return UserProfile(
            uid=id,
            name="Ralph Waldo Emerson",
            style=MartialArt.KARATE
    )

```

```
# client.php
```

```php
$transport->open();

$user = $client->retrieve(1);
var_dump($user);

$new_user = new example_UserProfile(array(
  "uid" => '123',
  "name" => "Ralph Waldo Emerson",
  "style" => example_MartialArt::KARATE
));
$client->store($new_user);

$transport->close();
```

```
# server.py
```

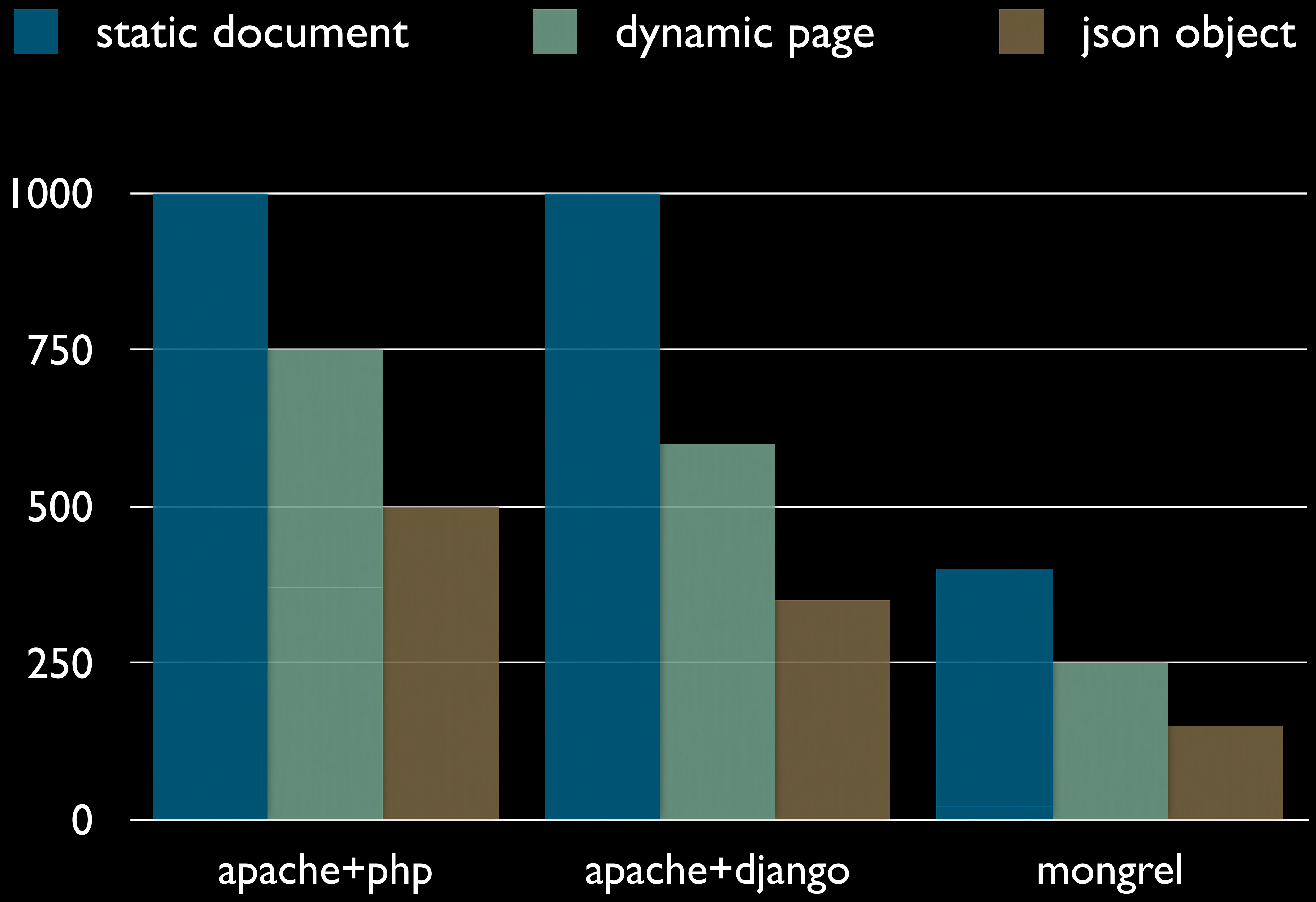
```python
class UserStorageHandler:
    def __init__(self):
        pass

    def store(self, user):
        print "stored " + str(user)

    def retrieve(self, id):
        print "retrieved " + str(id)
        return UserProfile(
            uid=id,
            name="Ralph Waldo Emerson",
            style=MartialArt.KARATE
    )

```

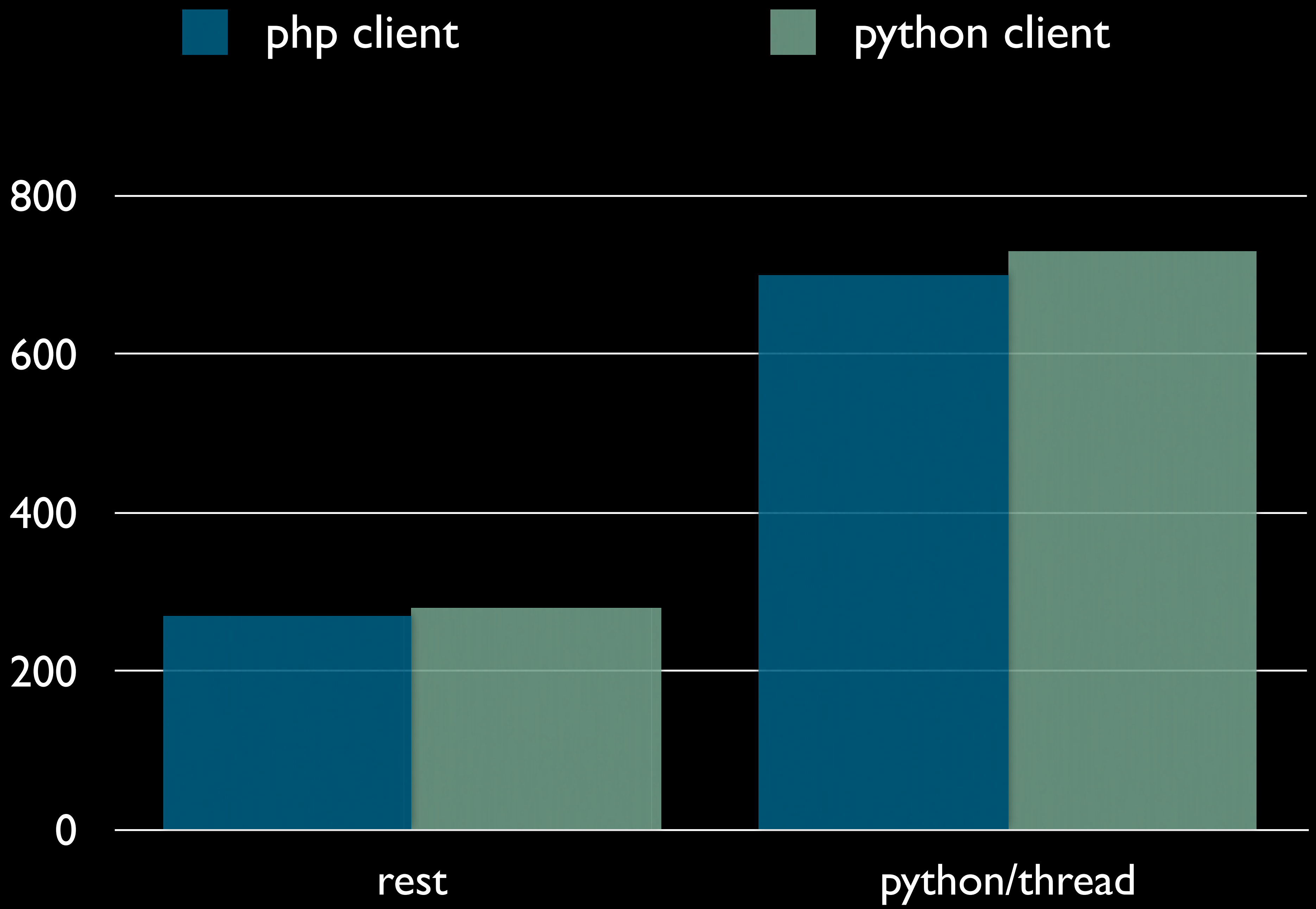
bingo!

```
# client.php

$transport->open();

$user = $client->retrieve(1);
var_dump($user);

$new_user = new example_UserProfile(array(
  "uid" => '123',
  "name" => "Ralph Waldo Emerson",
  "style" => example_MartialArt::KARATE
));
$client->store($new_user);

$transport->close();
```

```
# server.py

class UserStorageHandler:
    def __init__(self):
        pass

    def store(self, user):
        print "stored " + str(user)

    def retrieve(self, id):
        print "retrieved " + str(id)
        return UserProfile(
            uid=id,
            name="Ralph Waldo Emerson",
            style=MartialArt.KARATE
    )

```

Benchmarks


static document
dynamic page
json object
1000
750
500
250
0
apache+php
apache+django
mongrel



php client
python client
800
600
400
200
0
rest
python/thread

Desvantagens

Não tão ubíquo (quanto HTTP)

Não tão maduro (quanto HTTP)

Não tão óbvio (quanto HTTP) ?!

Pontos fortes

Compatibilidade entre linguagens

Serialização built-in

Performance!

meme

DO YAHOO!

http://meme.yahoo.com

Mais

Versionamento da interface

Thrift + Protocol buffers

Referências

http://incubator.apache.org/thrift/

https://github.com/bzanchet/presentation-thrift-fisl10/