Building a maintainable bi-directional cross platform protocol

Pavel Dovbush
William Lewis
@ badoo

Background

Solution and Implementation

Examples

API

- Application Programming Interface

- Operations
- Inputs
- Outputs
- Type definitions

- Independent of implementation


MAKE AN API CALL THEY SAID
IT'LL BE EASY THEY SAID
makeameme.org

Requirements

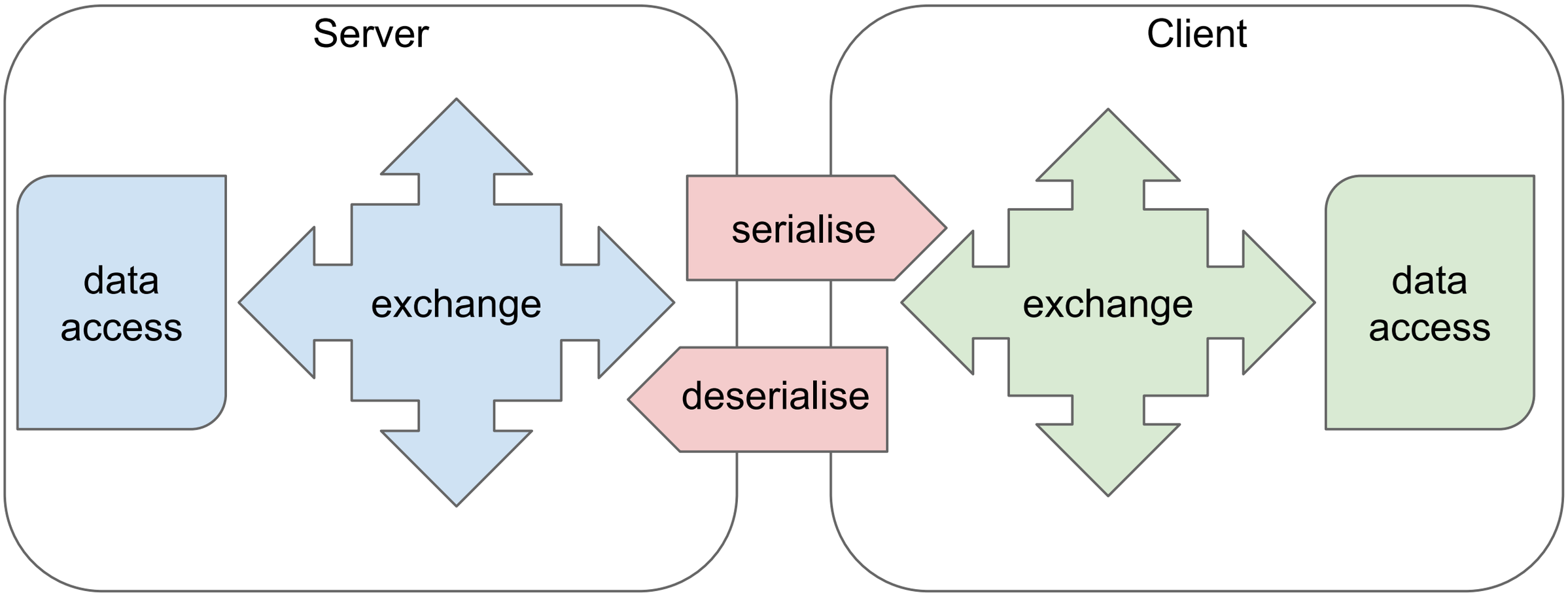
Perfect Client Server API:

- Just works – Magic! :)

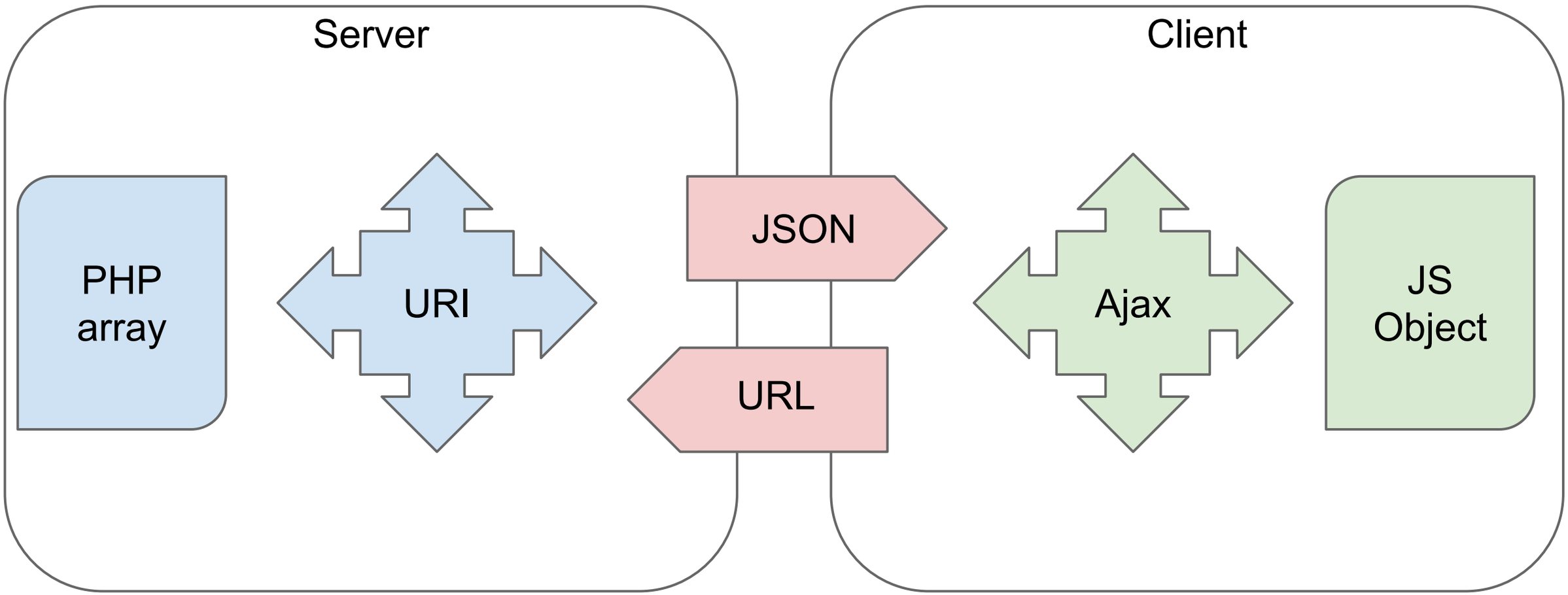
- Flexible
- Extensible
- Testable
- Maintainable
- Platform and language neutral
- Focused on features, not bytes over the wire

Overview

- Encoding
- Message exchange
- Data access



REST+JSON



REST+JSON problems

Encoding:

- Server: JSON
- Client: URLEncode

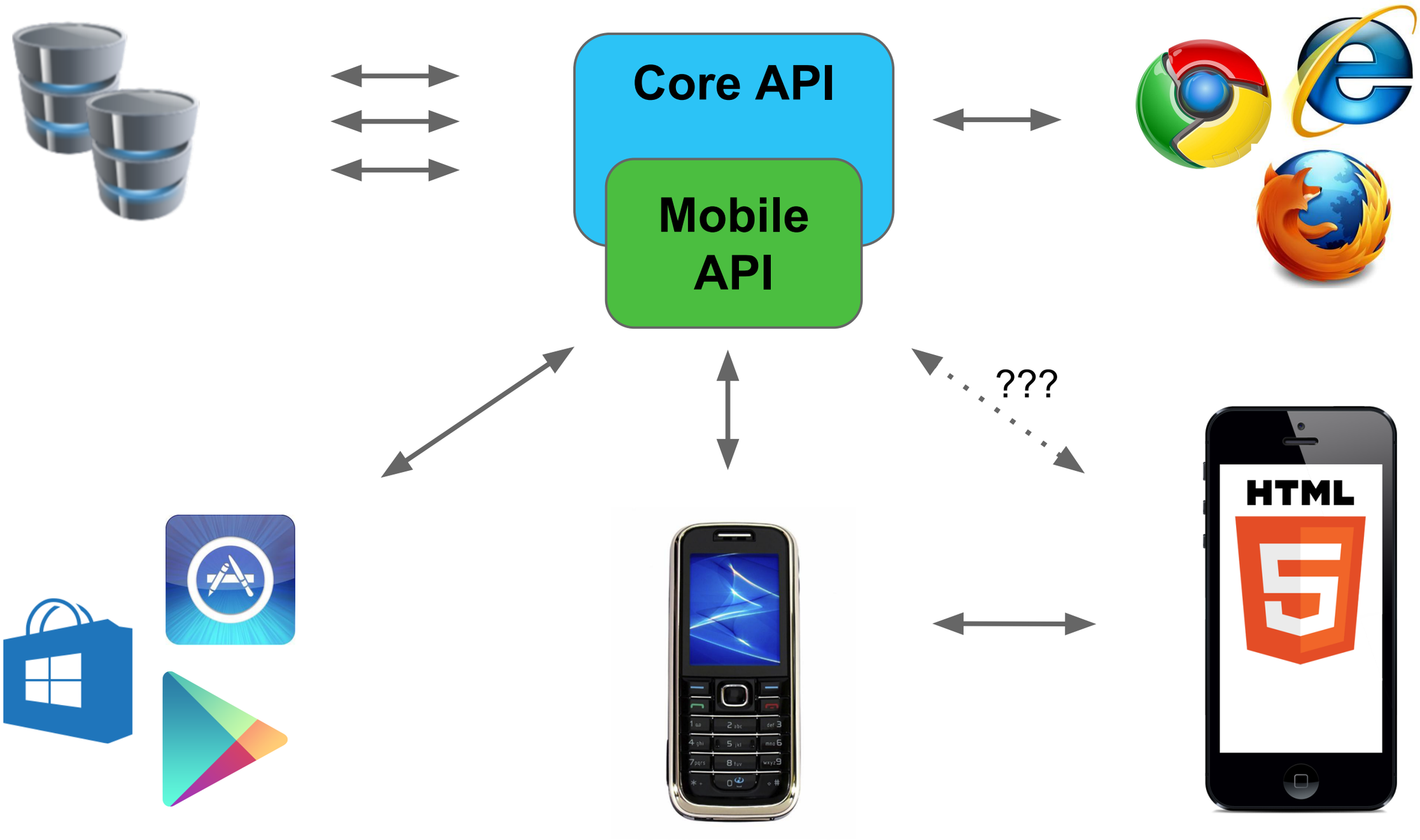
Message exchange:

- Client: HTTP request/response model
- Server: URI-based, config on web- or app- server

Data access:

- No canonical definition
- No versioning
- Duplicate implementations and configuration

Badoo APIs



Background

Solution and Implementation

Examples

Protocol implementations

SOAP

{JSON}
schema

protobuf
Protocol Buffers - Google's data interchange format

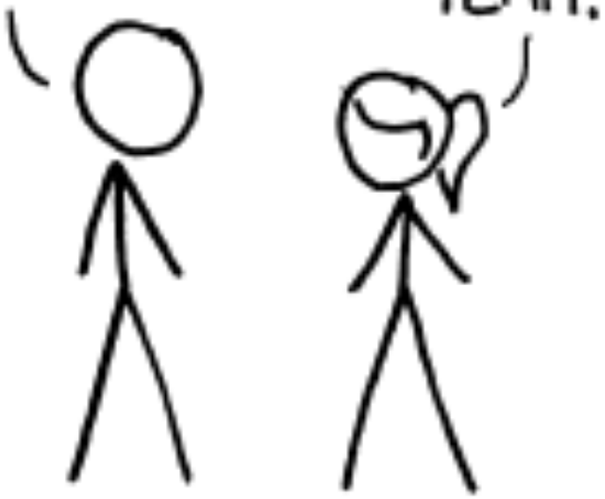
Thrift

AVRO

HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)

SITUATION:
THERE ARE
14 COMPETING
STANDARDS.

14?! RIDICULOUS!
WE NEED TO DEVELOP
ONE UNIVERSAL STANDARD
THAT COVERS EVERYONE'S
USE CASES.



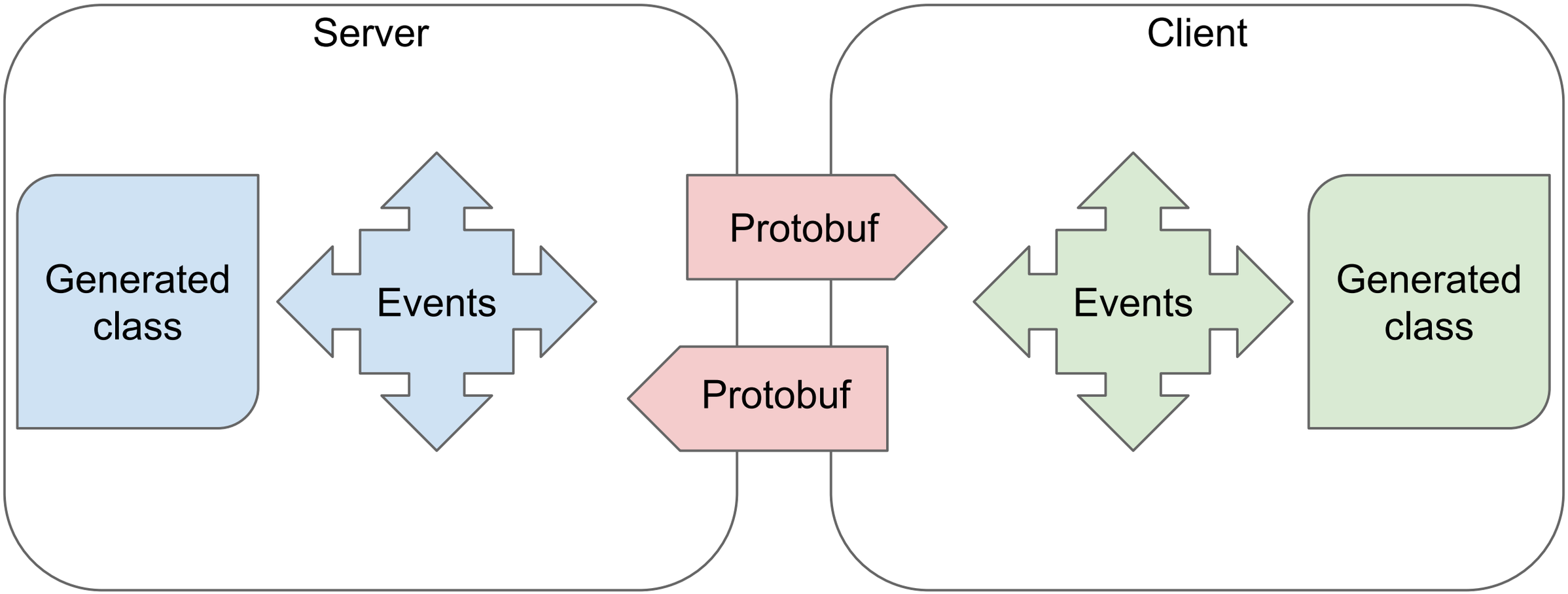
SOON:

SITUATION:
THERE ARE
15 COMPETING
STANDARDS.

Protocol values

- Protocol description
- Encoding
- Data access
- Message exchange
- Versioning

Protobuf + Own RPC



Description, versioning

Google Protocol Buffers

- Interface description language
- Internal representation
- Language support
  - v2.3 plugin support
- Encoding and network efficiency

Interface description language

- enum
- message
- field
- service
- option

Protobuf is self-describing - [descriptor.proto]

Interface description language

Label	Type	Name	Number
optional required repeated	bool string message enum float int32 + more numeric	field_name	= 1;

required	**string**	**user_name**	**= 1;**
optional	**uint32**	**age**	**= 2;**

Interface description language

```
enum Role {
    ADMIN = 1;
    USER = 2;
}
```

```
message User {
    required string name = 1;
    repeated string nickname = 2;
    optional uint32 age = 3;
    required Role role = 4;
}
```

Protobuf usage

Encoding:

- Only binary

Message exchange:

- Simple RPC

Data access:

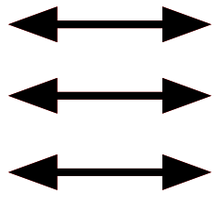
- No support for PHP
- No support for JS

Protobuf v2.3.0 compiler plugins

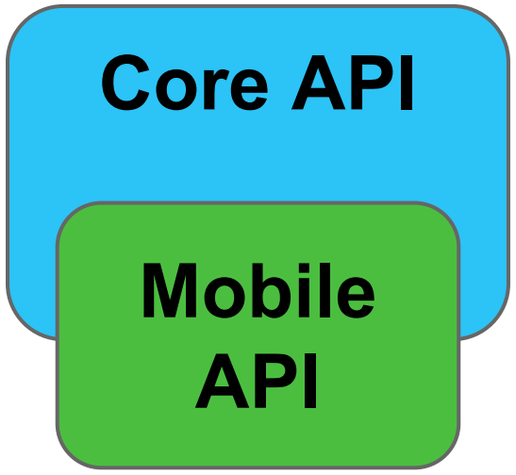
- Brilliant internal architecture
- Very simple plugin system
- Can generate any code in any language
- IDL is completely separated from serialization part

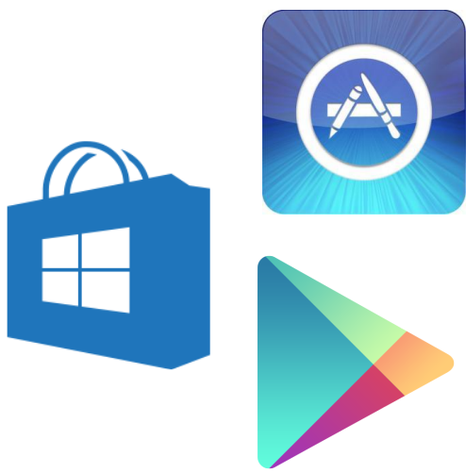
- gist with example

Encoding



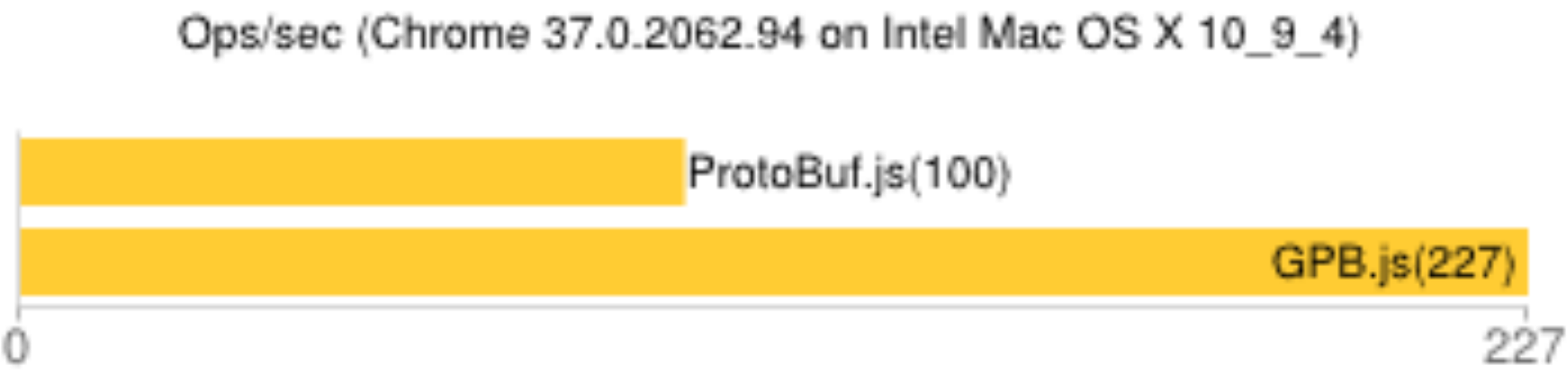
Native binary serialization





Custom JSON serialization

Performance



Code auto-generation

- WebSite DEVEL - On any request if ‘.proto’ file is newer than generated code - regenerate
- Mobile DEVEL - Grunt task - regenerate on file change
- PRODUCTION - generate before deploy

Message exchange

```
service SearchService {
rpc Search (SearchRequest)
      returns (SearchResponse);
}
```

- Too simple for a complex application
- We need a wrapper for every Request/Response
- Anytime responses

Message exchange (two-way RPC)

```
new RPC(request_type, parameter)
    .on(response_type, callback)
    .on([type1, type2], callback)
    .request();

RPC.any.on(type3, callback);

function callback(err, /** Type1 */ response1) {}
```

request_type & response_type are values of enum MessageType
parameter & response are Protobuf messages

Versioning

Old clients ignore:

- new fields
- unsupported commands

Migrating to new protocol version:

- cause compilation error on field removal

Take aways

- Protocol defined in one place
- Code uses data-access classes
- Validation
- Encoding can vary
- Flexible message exchange
- Versioning

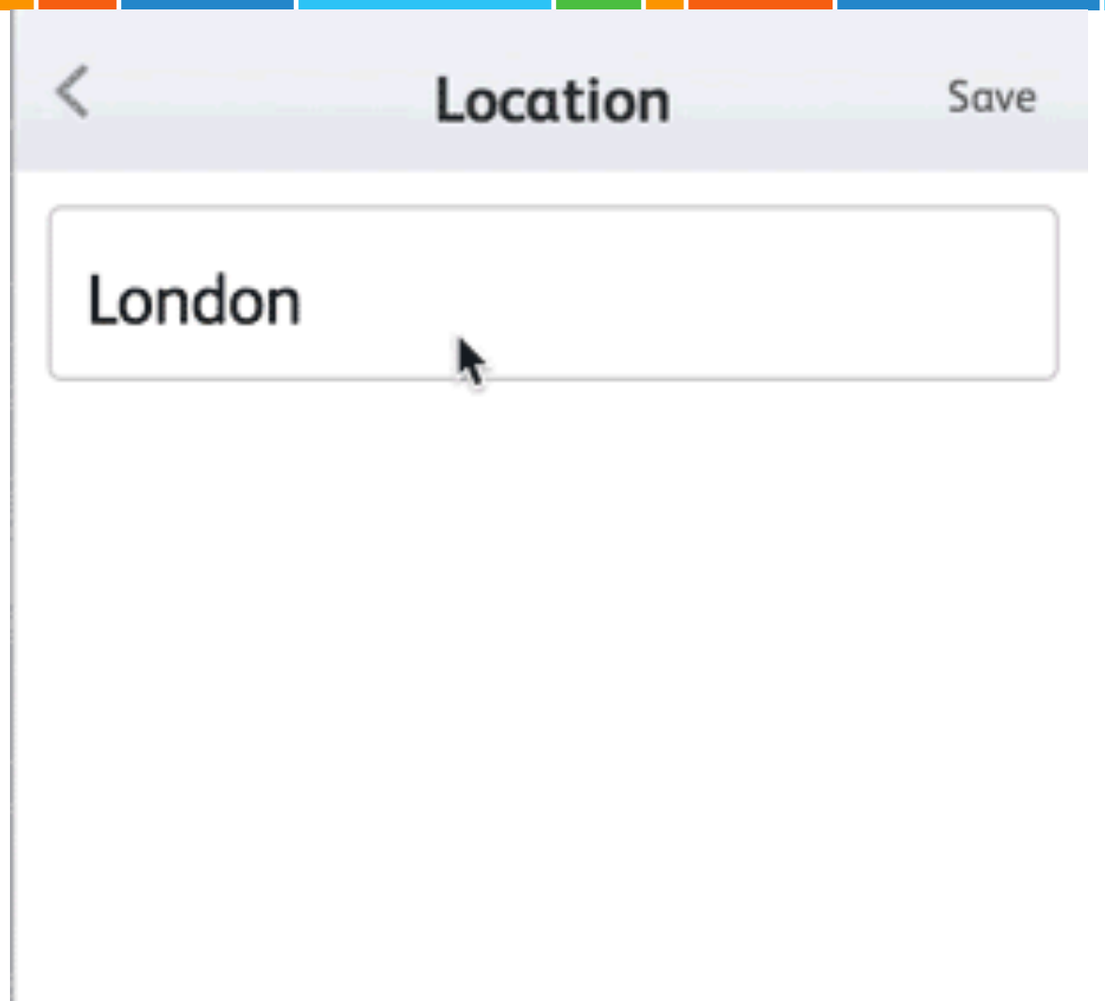
- Any part can be changed without affecting anything else

Background

Solution and Implementation

Examples

Building a service


Location
Save
London

Building a service

- Location search
  - Client query: city name
  - Server response: list of cities
    (ID, name, lat, long)
- Client notification
  - Anytime server response: user message

Protobuf definition

```
enum MessageType {
    // body: CityQuery
    SERVER_SEARCH_CITIES = 1;

    // body: Cities
    CLIENT_FOUND_CITIES = 2;

    // body : ClientNotification
    CLIENT_NOTIFICATION = 3;
}

message ClientNotification {
    required string id = 1;
    optional string title = 2;
    optional string message = 3;
}
```

```
message CityQuery {
    optional string name = 1;
}

message Cities {
    repeated City cities = 1;
}

message City {
    required int32 id = 1;
    required string name = 2;
    optional double longitude = 3;
    optional double latitude = 4;
}
```

Protobuf definition

```
message RPCMessage {
    required int32 version = 1;
    optional int32 message_id = 2;
    repeated MessageBody body = 3;
}

message MessageBody {
    required MessageType message_type = 1;
    optional CityQuery city_query = 2;
    optional Cities cities = 3;
    optional ClientNotification client_notification = 4;
}
```

Generated classes examples

```
define(['GPB/gpb2'], function(/** $gpb */$gpb) {
var Protocol = $gpb.namespace('Demo');
/**
 * CityQuery
 * @class {Protocol.CityQuery}
 * @extends {$gpb.Message}
 */
var CityQuery = Protocol.CityQuery = function() {
    $gpb.Message.apply(this, arguments);
};
$gpb.extend(CityQuery, $gpb.Message);
CityQuery.prototype.$gpb = 'Demo.CityQuery';
CityQuery.prototype._descriptor = {"fields": {"name": {"type": 9, "number": 1,
"label": 1}}};

return CityQuery;

});
```

Generated classes examples

```php
<?php

namespace GPBJS\Demo;

class CityQuery extends \GPBJSBase\Message
{
    protected static $name = 'Demo.CityQuery';
    protected static $fields = array(
        'name' => array('type' => 'string', 'optional' => true, 'repeatable' => false, 'hash' =>
    false, 'raw' => False, 'is_enum' => false, 'is_message' => false),
    );

    public function setName($value)
    {
        $this->_setFieldValue('name', $value);
        return $this;
    }
    public function getName()
    {
        return $this->_getFieldValue('name');
    }
}
```

RPC city query example

```
var cityQuery = new Protocol.CityQuery().setName('london');
new RPC(Protocol.MessageType.SERVER_SEARCH_CITIES, cityQuery)
    .on(Protocol.MessageType.CLIENT_CITIES, onCities)
    .on(Protocol.MessageType.CLIENT_NOTIFICATION, onNotification)
    .request();

function onCities (err, /** Protocol.Cities */ cities) {
    if (err) { /* error handling */ return; }

    for (var city in cities.getCities()) {
        cityListView.update(city.getId(), city.getName());
    }
}

function onNotification ( err, /** Protocol.ClientNotification */ notification) {
     alert(notification.getTitle() + '\n' + notification.getMessage());
}
```

RPC anytime response example

```
RPC.any.on(Protocol.MessageType.CLIENT_NOTIFICATION,
onNotification);

function onNotification ( err, /** Protocol.ClientNotification */ notification) {
    alert(notification.getTitle() + ‘\n’ + notification.getMessage());
}
```

Building a maintainable bi-directional cross platform protocol

- REST + JSON
- Protocol values
  - Interface description language
  - Encoding and performance
  - Protobuf compiler plugins
  - Code auto-generation
  - Message exchange (two-way RPC)
  - Versioning
- Examples

Thanks! Questions?

Pavel Dovbush <dpp@corp.badoo.com>

William Lewis <william.lewis@corp.badoo.com>

@netproteus

Slides: techblog.badoo.com

Thanks:
Google team for Protobuf itself
Andrey Nigmatulin <anight@corp.badoo.com> for bringing GPB in Badoo in 2009