Transit

@stuarthalloway
stu@cognitect.com

Agenda

the problem

transit

examples

why not JSON/XML ?

wrapup

The Problem

send values between applications

written in different languages

without context (schemas optional)

extensibly

with good performance

Values

send **values** between applications

written in different languages

without context (schemas optional)

extensibly

with good performance

Values

no identities

no methods

no code

immutable (cacheable!)

information

Applications

send values **between applications**

written in different languages

without context (schemas optional)

extensibly

with good performance

Between Applications

program-to-program

human readability secondary

but possible

data structures

not documents

Different Languages

send values between applications

written in **different languages**

without context (schemas optional)

extensibly

with good performance

Different Languages

don’t share classes

don’t share ADTS

share primitive semantic types

share composition and aggregation

Schemas Optional

send values between applications

written in different languages

without context (schemas optional)

extensibly

with good performance

Schemas

are difficult

try writing one for your insurance or tax form

are manifold

DCD DDML DSDL DSD DTD NRL OASIS CAM RELAX NG

Sox Schematron XDR XER XSD

one size will not fit all

one size will not fit even one! (smuggling)

Smuggling

not all of your data will fit your schema

smuggle it through as as string

```
"SELECT foo, baz FROM bars WHERE foo IN (1 ,2 ,3)"
```

sure to happen if a schema is *required*

Extensibility

send values between applications

written in different languages

without context (schemas optional)

extensibly

with good performance

Extensibility

generic

recursive

optional

without extensibility, you get *conventions*

- if X, then Y is a date, else Y is a UUID
- repeat 10,000 places in a nontrivial system

Performance

send values between applications

written in different languages

without context (schemas optional)

extensibly

with good **performance**

Fast Enough?

if speed dominates

- pick a common, rigid format
- driven by a common language
- hand-tune encoding

speed rarely dominates to this degree

- be fast without pouring concrete on what system can do
- be fast on platforms and languages you must reach

Transit

leverage high performance parsers

reach the browser

rich core type set

tagged extensions

pluggable language mappings

Leveraging JSON

impoverished semantics

available everywhere

with fast native help lots of places!

Leveraging MessagePack

binary format

bytecode-tagged encoding

simple spec

- fast C impls for Ruby, Python

Encoding

situation	solution	example	sample encoding
exactly matches underlying parser	use parser type!	string on JSON	`"hooray"`
scalar not distinguishable in underlying parser	string encoding with well-known prefix	boolean map key on JSON	`"~?t"`
composite not distinguishable in underlying parser	composite encoding first element describes those that follow	set on JSON	`{"~#set" : [1 2 3]}`

Type Sampler

type	example
instant in time	`"~t1985-04-12T23:20:50.52Z"`
UUID	`"~u531a379e-31bb-4ce1-8690-158dceb64be6"`
link	`{"~#link" : {"href": “~rhttp://cognitect.com", "rel": "alternate"}}`
composite keys in a map	`{"~#cmap" : [[1 2], “buckle my shoe”, [3 4], “open the door”}`

Impoverished JSON

you would think that	but…	so you have to	which looks like this
JSON has “numbers”	they are range-limited	encode big numbers as strings	`"~i987654321012345"`
JSON has “maps”	keys must be strings	encode maps as lists	`["^ ", "name", "Scout Finch", "age", 6]`

Take the Tour

```
1 var r = transit.reader("json");
2 r.read('[1,1.5,1e5,"A string!","\u03BB"]');
3 // [1,1.5,100000,"A string!","λ"]
```

EVAL

http://cognitect.github.io/transit-tour/

Take the Tour

```
1 var r = transit.reader("json"),
2     s = r.read('{"~#set":["cat","dog","bird"]}');
3 s.has("bird");
4 // true
```

EVAL

http://cognitect.github.io/transit-tour/

Handler

```
1  var Point = function(x, y) {this.x=x;this.y=y},
2      r     = transit.reader("json", {
3                "handlers": {
4                   "point": function(v) {
5                      return new Point(v[0], v[1]);
6                   }
7                }
8              });
9
10 r.read('[{"~#point":[0.5,1.5]}]');
11 // [{"x":0.5,"y":1.5}]
```

EVAL

http://cognitect.github.io/transit-tour/

No Handler Required

```
1 var r = transit.reader("json"),
2     l = r.read('{"~#line":[0,0,100,100]}'),
3     w = transit.writer("json-verbose");
4 w.write(l);
5 // "{\"~#line\":[0,0,100,100]}"
```

EVAL

http://cognitect.github.io/transit-tour/

Tradeoffs

Readability

- less so than JSON — tolerable

Size

- tags
- long tags
- repetition of tags — must consider perf impact

Dynamic Caching

circular rotating cache

- caching keywords/symbols/tags mitigates tags
- caching map keys optimizes common use case

reader cache mandatory, writer cache optional

“Transit over JSON” beats raw JSON for size

- *sometimes beats raw JSON for speed!*

Caching Example

```
["^ ",
 "firstName", "Scout",
 "lastName", "Finch",
 "firstName", "Jem",
 "lastName", "Finch"]
```

caching off

```
["^ ",
 "firstName", "Scout",
 "lastName", "Finch",
 "^0", "Jem",
 "^1", "Finch"]
```

caching on

Browser Perf

Testing in Chrome 39.0.2171.65 on OS X 10.9.5		
Test		**Ops/sec**
JSON.parse	`JSON.parse(jsonStr);`	1,398 ±1.98% fastest
transit read	`r.read(transitJSONStr);`	1,174 ±5.33% 19% slower
JSON.parse w/ hydrate	`JSON.parse(jsonStr, function(k, v) { return v });`	689 ±3.14% 51% slower

all the way
to domain data

nontrivial programs
postprocess JSON

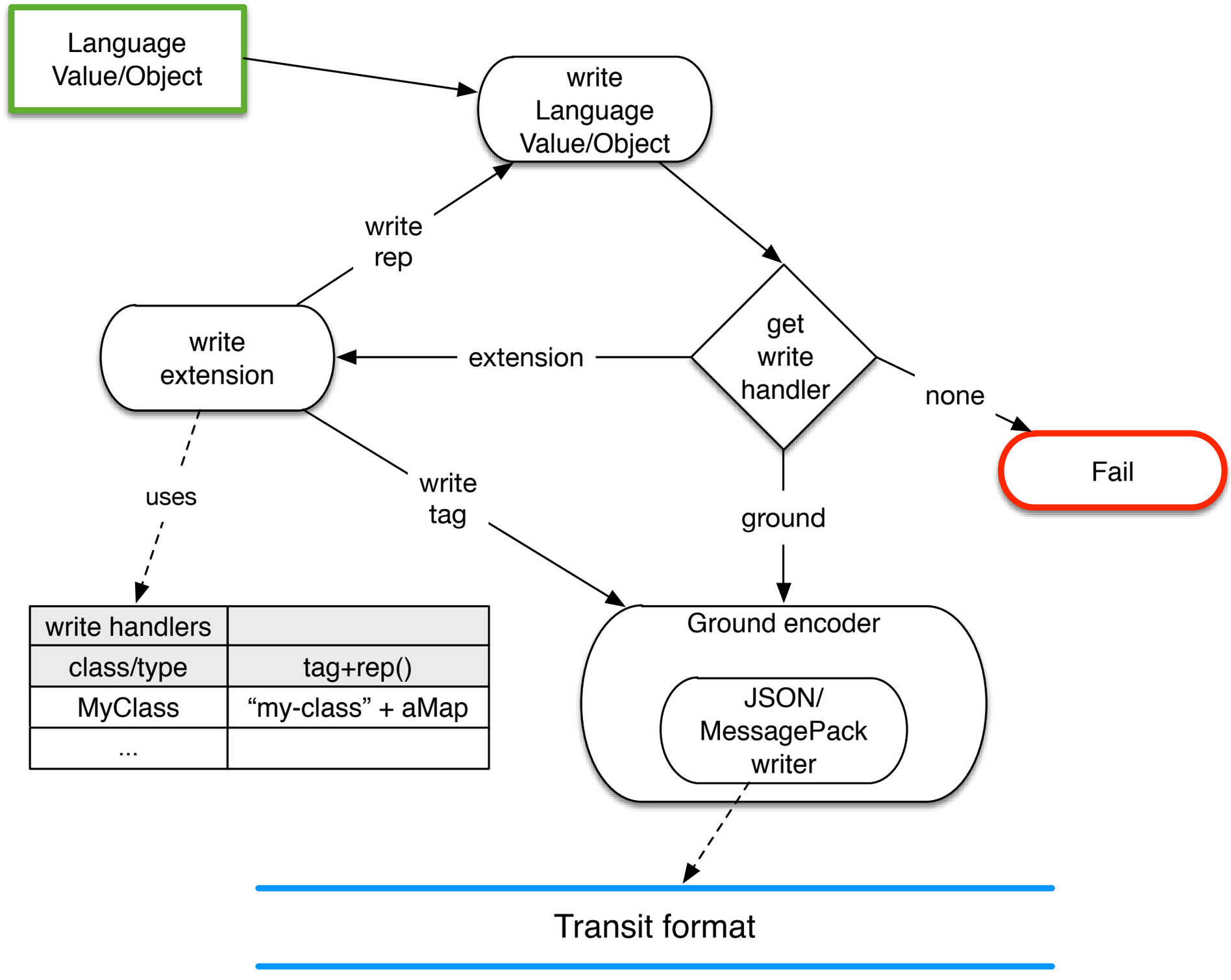
see for yourself

http://jsperf.com/json-vs-transit/2

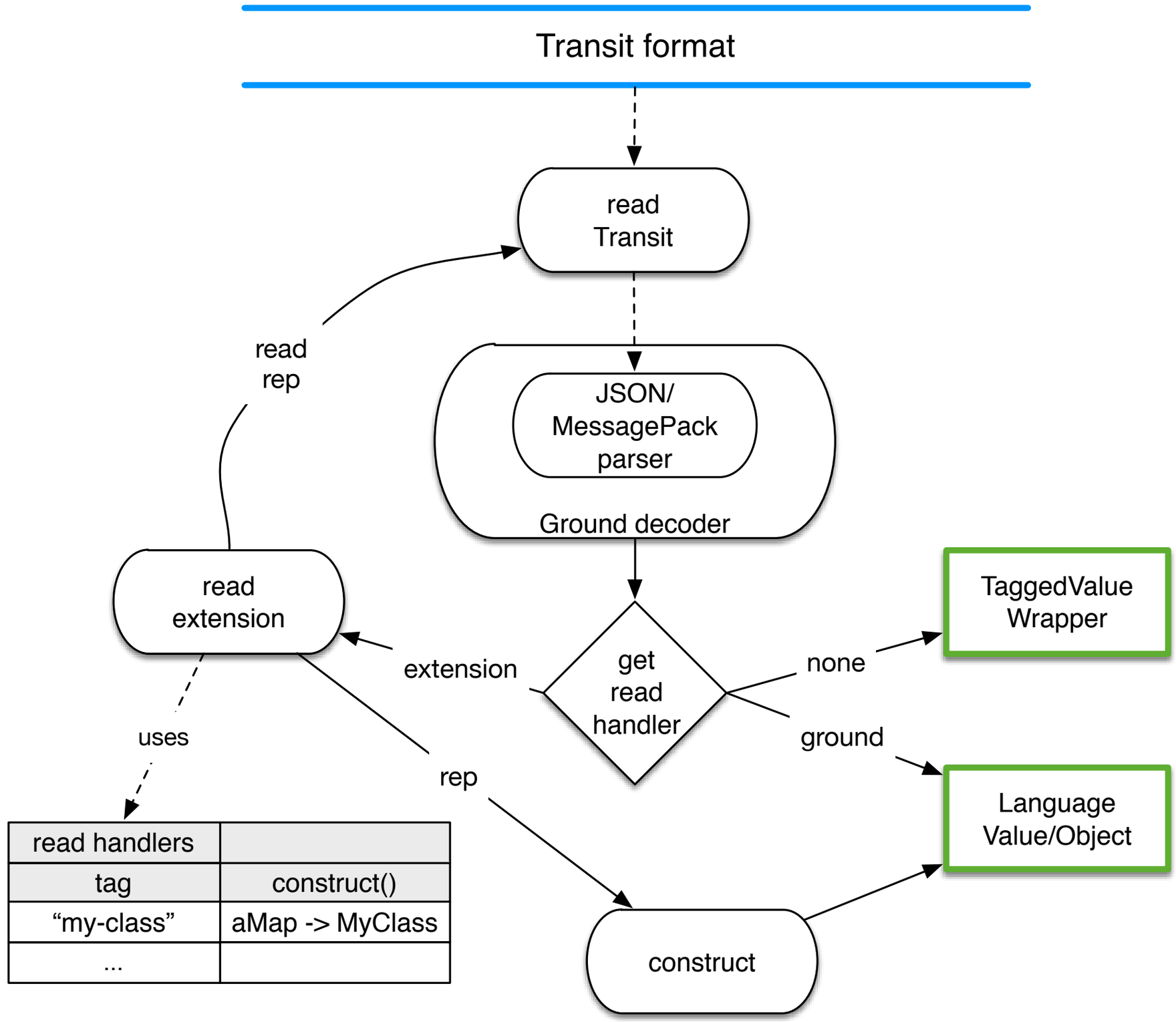
Source

language	source
Clojure	http://github.com/cognitect/transit-clj
ClojureScript	http://github.com/cognitect/transit-cljs
Java	http://github.com/cognitect/transit-java
JavaScript	http://github.com/cognitect/transit-js
Python	http://github.com/cognitect/transit-python
Ruby	http://github.com/cognitect/transit-ruby
Transit Specification	https://github.com/cognitect/transit-format

Implementing Write



Implementing Read



Why Not

JSON?

XML?

${Your-Favorite}?

edn?

Fressian?

Impoverished JSON

good

- everywhere
- easy

bad

- small set of types
- numbers are broken
- not composable
- inextensible

XML

X is good

add tags

process tags you don’t understand

M, not so much

text/document orientation

gross APIs

${Your-Favorite}

encoding	send values	between apps	different langs	without context	extensibly	with good perf
JSON	could	yes	yes	no	no	yes
Transit	yes	yes	yes	yes	yes	yes
XML	could	text bias	yes	maybe	yes	yes
${Your-Favorite}						

edn

similar design goals

ground-up implementation

- more readable
- cannot match JSON perf in browsers

https://github.com/edn-format/edn

Fressian

similar design goals

more emphasis on performance

binary

less reach

Datomic

https://github.com/Datomic/fressian

Transit

send values between applications

written in different languages

without context (schemas optional)

extensibly

with good performance

@stuarthalloway

https://github.com/stuarthalloway/presentations/wiki. Presentations.

http://pragprog.com/book/shcloj2/programming-clojure. *Programming Clojure*.

mailto:stu@cognitect.com