Introducing taocpp/json

https://github.com/taocpp/json

Daniel Frey

2016-11-10

Disclaimer

- Opinions expressed are solely my own and do not express the views or opinions of my employer

- All mistakes are mine, please point them out

- When taking a note for a question, please write down the slide number!

Origin

- Authors of the PEGTL and taocpp/json:
  Dr. Colin Hirsch and Daniel Frey

- Originally a benchmark for the PEGTL
  https://github.com/ColinH/PEGTL

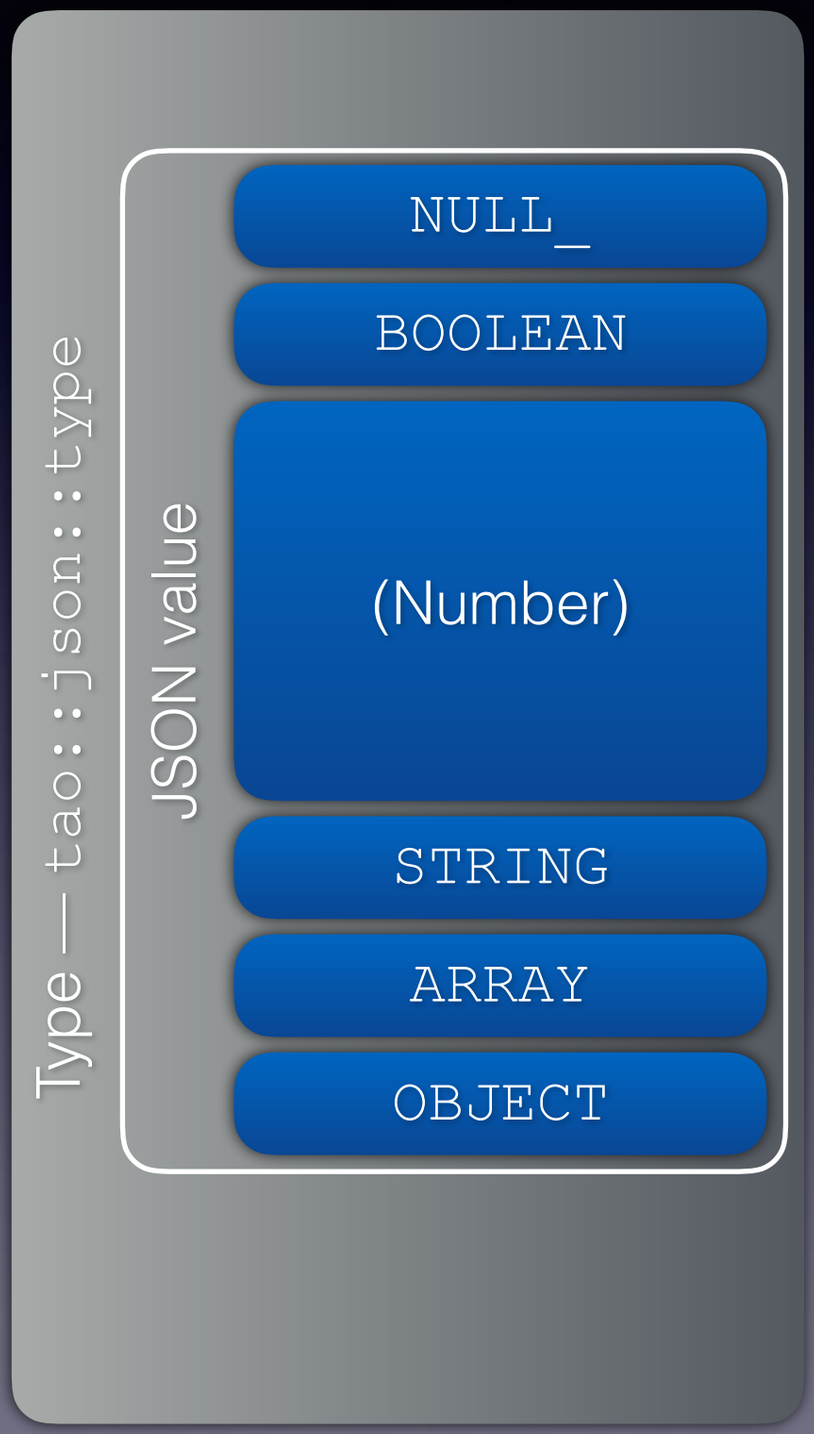
- Benchmarked against 40 other libraries
  https://github.com/miloyip/nativejson-benchmark

- Inspired by Niels Lohmann's JSON library
  https://github.com/nlohmann/json

Our use-cases

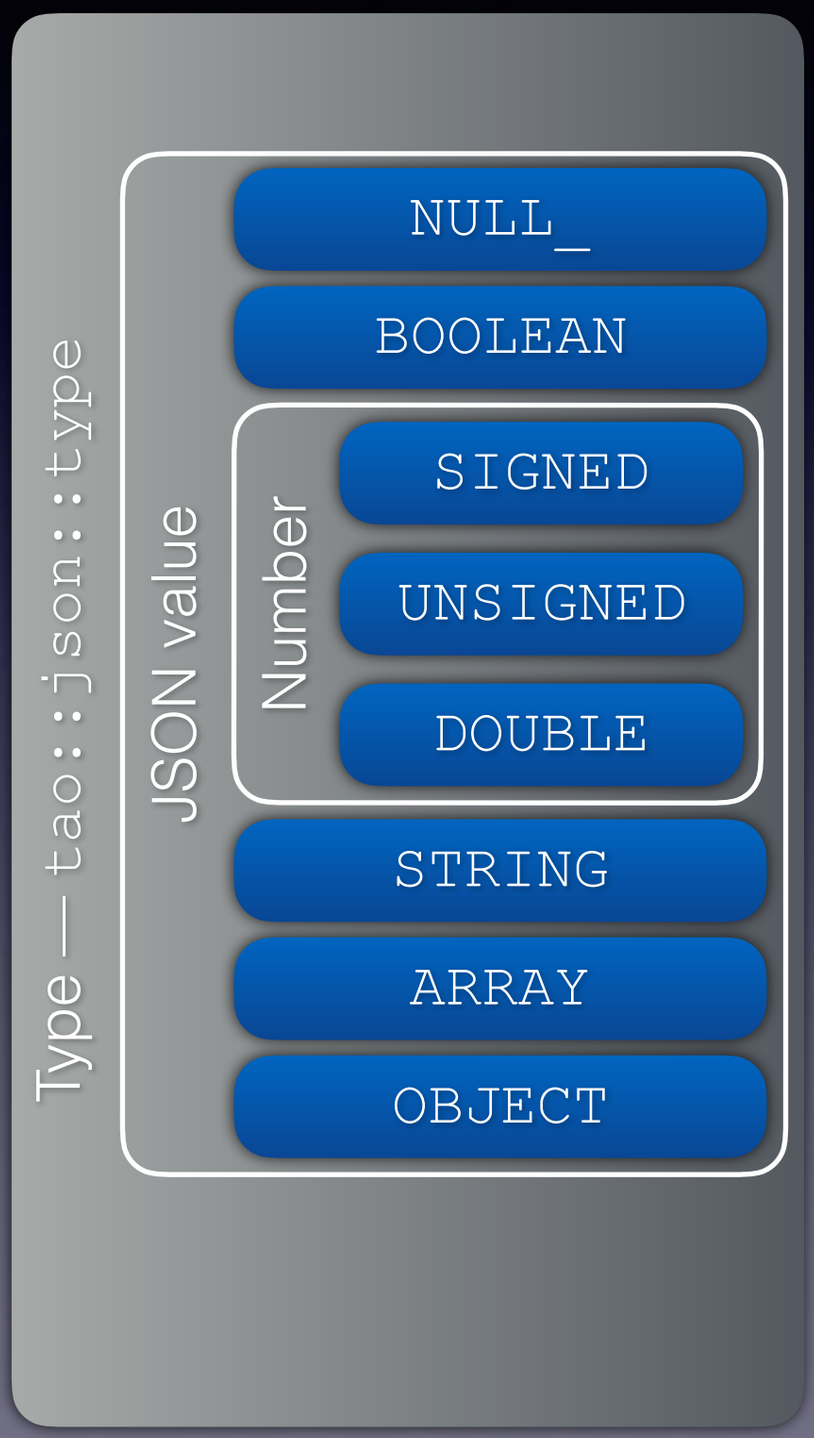
- REST APIs
- Logging: Structured data, supports ELK stack
- Several thousand JSON value constructions and serializations *per second*
- Main development driven by demand
- Used in production code

json::value



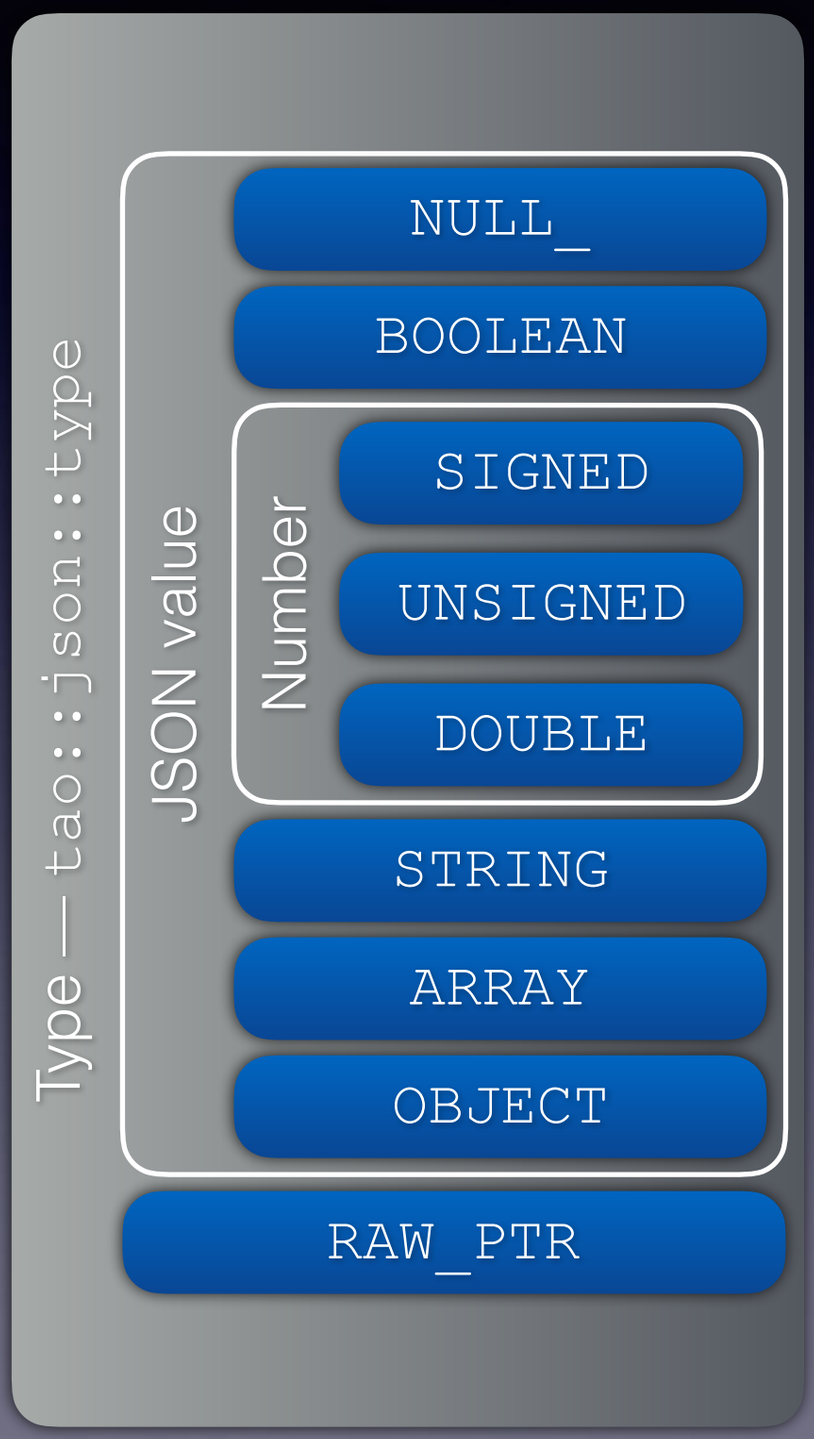
- Contains a type and a C++ `union` to store data
- Query type: `v.type()`
- Basic JSON types

Type



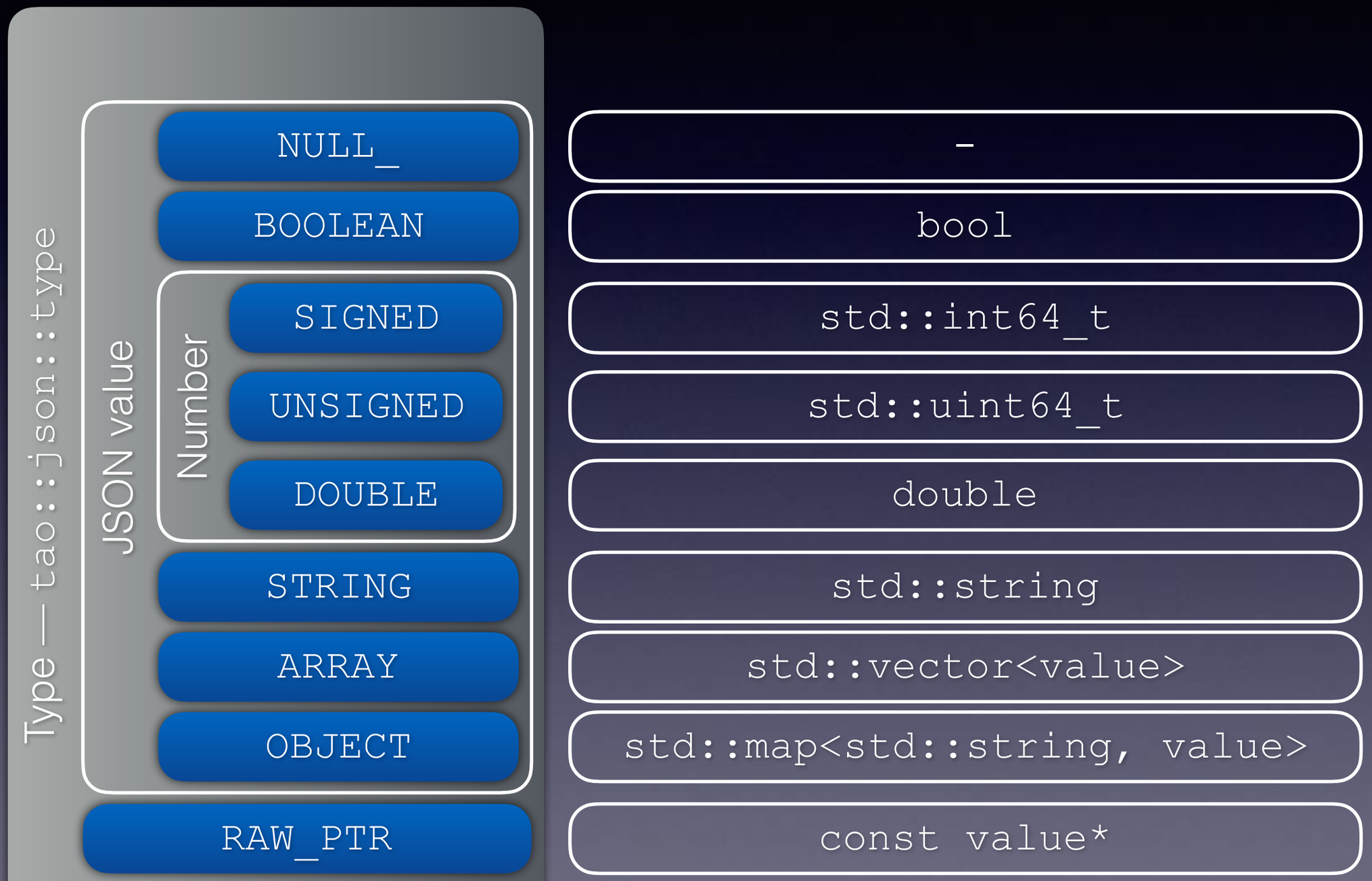
- Contains a type and a C++ `union` to store data
- Query type: `v.type()`
- Basic JSON types
- Numbers come in three flavours

Type

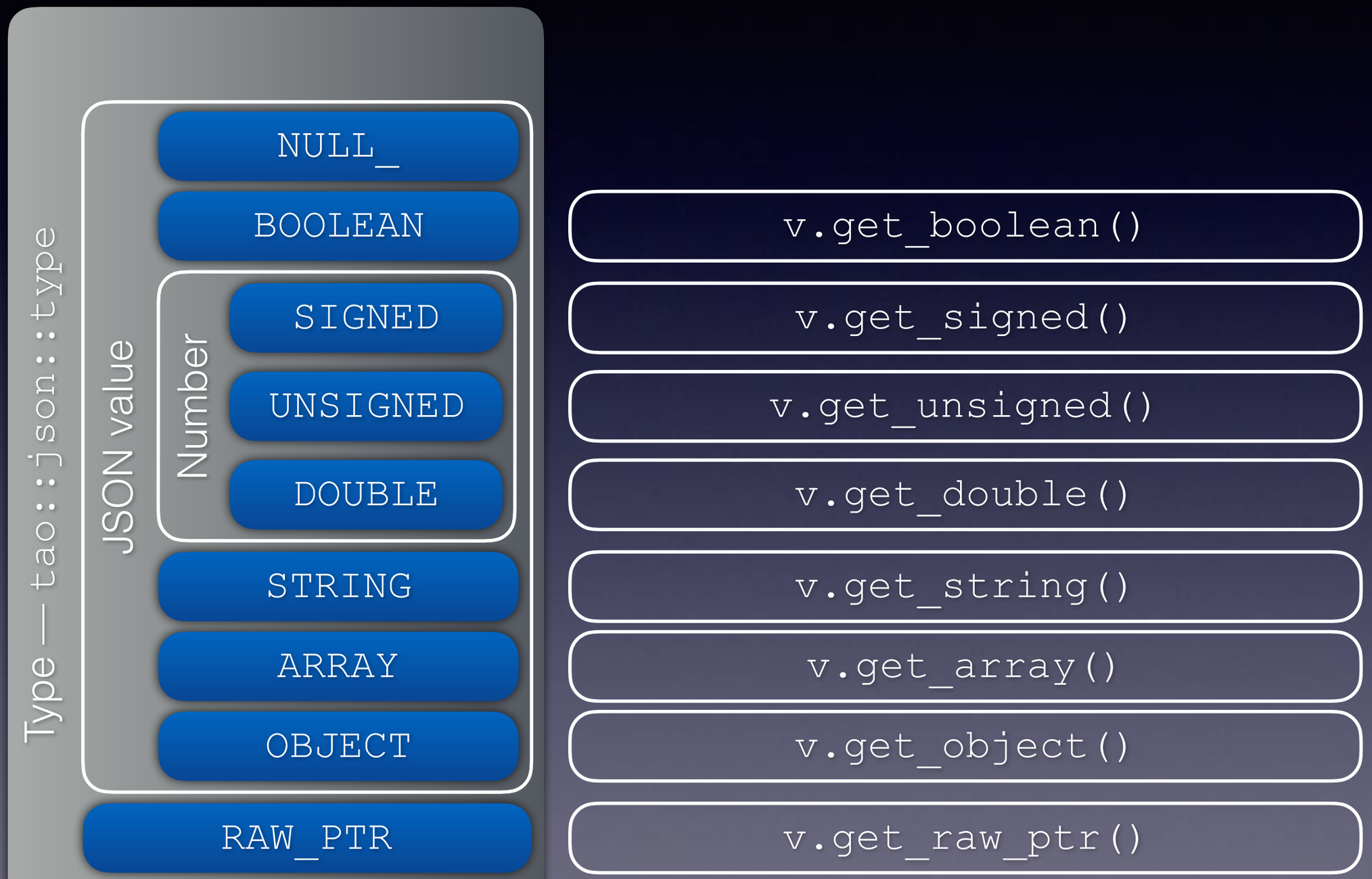


- First extension:
  A plain old raw pointer
- Rarely used…
- …but allows important optimisations!
- Non-owning, dumb.
  Like… *really* dumb!

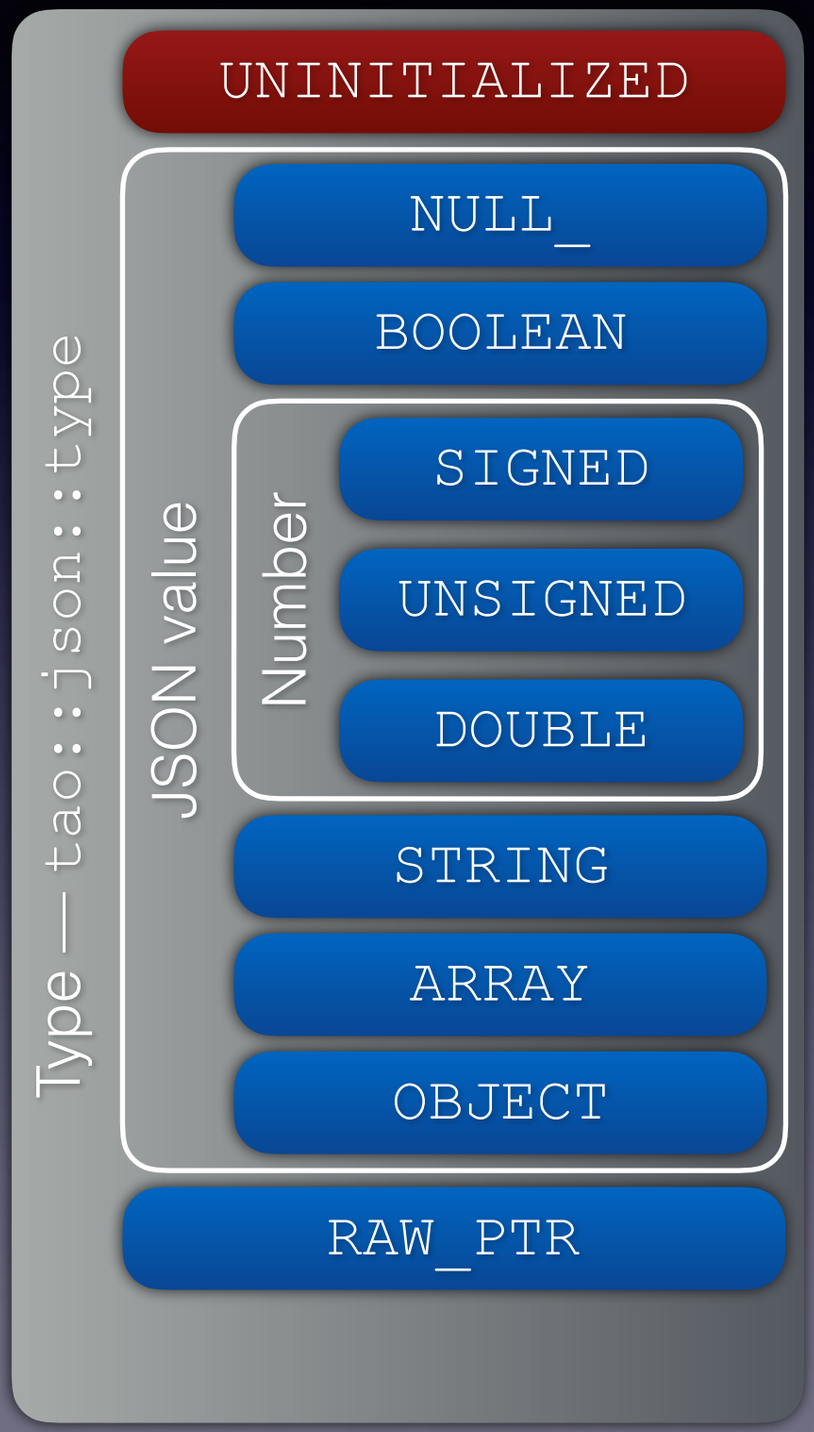
Data


Type — tao::json::type
JSON value
NULL_
BOOLEAN
Number
SIGNED
UNSIGNED
DOUBLE
STRING
ARRAY
OBJECT
RAW_PTR
-
bool
std::int64_t
std::uint64_t
double
std::string
std::vector<value>
std::map<std::string, value>
const value*

Getter

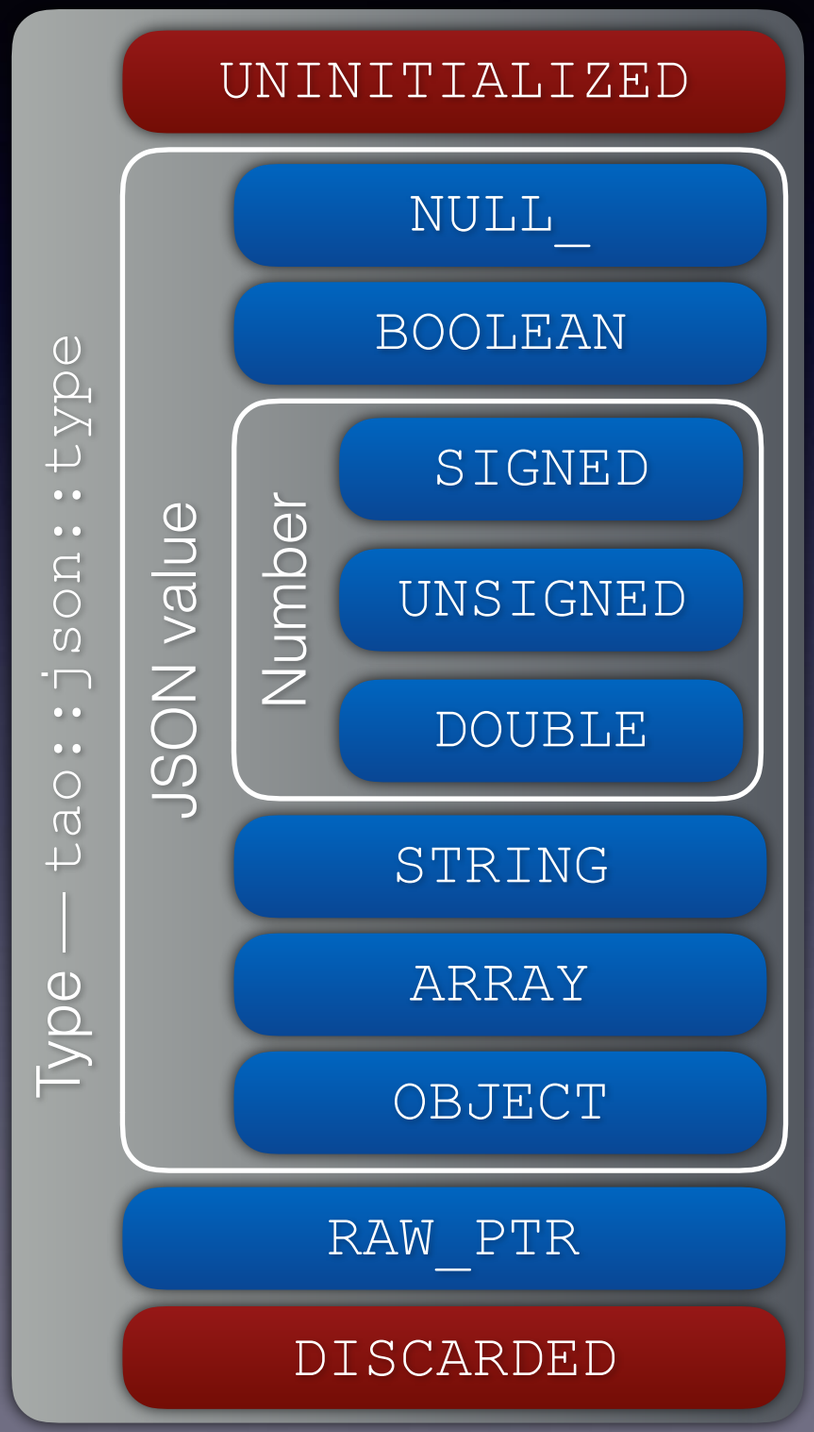

Type — tao::json::type
JSON value
NULL_
BOOLEAN
Number
SIGNED
UNSIGNED
DOUBLE
STRING
ARRAY
OBJECT
RAW_PTR
v.get_boolean()
v.get_signed()
v.get_unsigned()
v.get_double()
v.get_string()
v.get_array()
v.get_object()
v.get_raw_ptr()

Status



- Second extension: Other statuses, part 1
- Default-constructed values do *not* have a type assigned. The type/status is `UNINITIALIZED`

Status



- Second extension: Other statuses, part 2
- When a value is destructed, moved, … the type/status is set to `DISCARDED` (sometimes)

Logging

```
LOG(INFO, 1234, “Hello, world!”);

// LOG(LVL,ID,…) macro expands to:
if(tao::log::is_active(LVL, ID))
{
   tao::json::value v = __VA_ARGS__;
   tao::log::write(LVL, ID, v);
}
```

Value construction

```
json::value v = “Hello, world!”;
```

Value construction

```
json::value v = “Hello, world!”;
json::value v = true;
json::value v = 42u;
json::value v = -42;
json::value v = 1.23;
```

Value construction

```
json::value v = “Hello, world!”;
json::value v = true;
json::value v = 42u;
json::value v = -42;
json::value v = 1.23;

json::value v = json::null;
json::value v = json::empty_array;
json::value v = json::empty_object;
```

Array construction

```
json::value v = json::empty_array;
v.emplace_back(true);
v.emplace_back(42);
v.emplace_back("Hello, world!");
```

Array construction

```
json::value v = json::array({
   true, 42, “Hello, world!”
});
```

- Think `std::initializer_list<value>`.
- Reality is somewhat more complicated.

Object construction

```
json::value v = json::empty_object;
v.emplace(“foo”, true);
v.emplace(“bar”, 42);
v.emplace(“baz”, “Hello, world!”);
```

```
v[“foo”][“bar”]; // will throw
v[“foo”] = 123u;
v[“x”][“y”] = “nice”;
```

Object construction

```
json::value v = {
   { “foo”, true },
   { “bar”, 42 },
   { “baz”, “Hello, world!” }
};
```

- Keys are always strings, values are anything a direct construction of `value` accepts.
- No duplicate keys are allowed.

Object construction

```
json::value v = {
   { “foo”, true },
   { “bar”, 42 },
   { “baz”, “Hello, world!” }
};
```

- Keys are always strings, values are anything a direct construction of `value` accepts.
- No duplicate keys are allowed.

Object construction

```
json::value v = {
   { “Some dwarfs”,
      {
         { “Thorin”, “Leader” },
         { “Kili”, “Nephew” },
         { “Balin”, “Third-cousin” },
         { “Dori”, “Remote kinsman” },
         { “Bifur”, “From Moria” }
      }
   }
};
```

Object construction

```
json::value dwarfs = {
  { “leader”, “Thorin” },
  { “nephews”, json::array({
    “Kili”, “Fili”
  })},
  { “third_cousins”, json::array({
    “Balin”, “Dwalin”, “Oin”, “Gloin”
  })},
  …
};
```

Logging

```
LOG(INFO, 1234, “Hello, world!”);
```

Logging

```
LOG(INFO, 1234, {
    { “foo”, true },
    { “bar”, 42 },
    { “baz”, “Hello, world!” }
});
```

Logging

```
struct user
{
   bool is_human;
   std::string name;
   unsigned age;
};

user u = …;
```

Logging

```
LOG(INFO, 1234, {
    { “msg”, “add user” },
    { “user”,
        {
            { “is_human”, u.is_human },
            { “name”, u.name },
            { “age”, u.age }
        }
    }
});
```

Logging

```
template<> struct traits<user>
{
   static void assign(
      value& v, const user& u)
   {
      v = {
         { “is_human”, u.is_human },
         { “name”, u.name },
         { “age”, u.age }
      };
   }
};
```

Logging

```
LOG(INFO, 1234, {
    { “msg”, “add user” },
    { “user”, u }
});
```

Logging

```
template<> struct traits<user>
{
   static void assign(
      value& v, const user& u
   );

   static const char* default_key;
};
```

Logging

```
LOG(INFO, 1234, {
   { “msg”, “add user” },
   u
});
```

Logging

```
LOG(INFO, 1234, {
   { “msg”, “add user” },
   u
});
```

```
LOG(INFO, 1235, {
   { “msg”, “assign manager” },
   u,
   { “manager”, u2 }
});
```

Traits

```
template<> struct traits<user>
{
   static user as(const value& v);
};
```

```
auto u = v.as<user>();
```

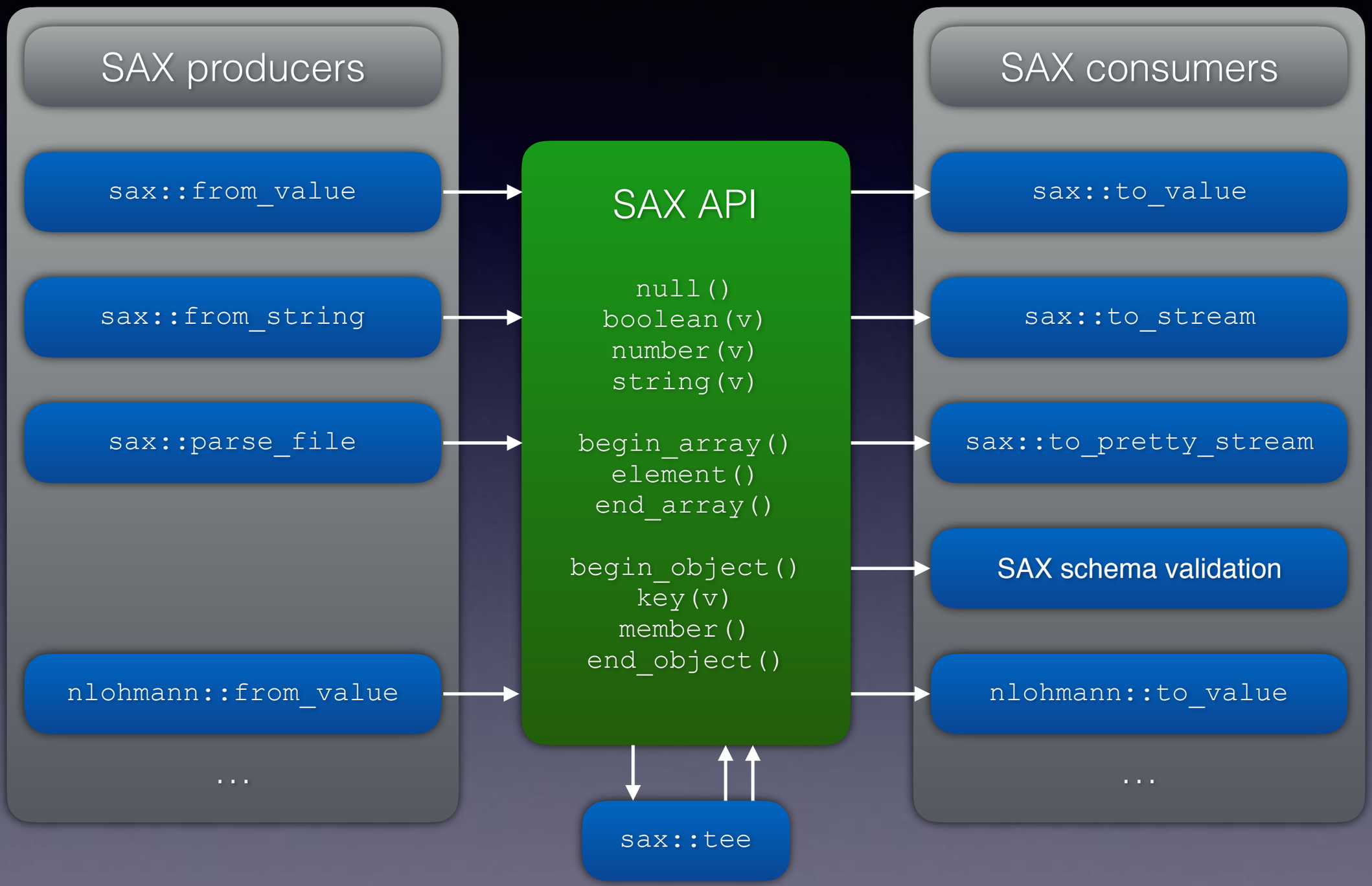
Traits

```
template<> struct traits<user>
{
   static user as(const value& v) {
      return user(
         v.at("is_human").get_boolean(),
         v.at("name").get_string(),
         v.at("age").as<unsigned>()
      );
   }
};
// other options under consideration
```

Traits

- Construct a value from any T
- Convert a value to any T
- Modify default behaviour
- `value` is *actually* `basic_value<traits>`

SAX


SAX producers
sax::from_value
sax::from_string
sax::parse_file
nlohmann::from_value
...
SAX API
null()
boolean(v)
number(v)
string(v)
begin_array()
element()
end_array()
begin_object()
key(v)
member()
end_object()
sax::tee
SAX consumers
sax::to_value
sax::to_stream
sax::to_pretty_stream
SAX schema validation
nlohmann::to_value
...

SAX producer

```
template<typename Consumer>
void from_value(const value& v, Consumer& c)
{
  switch(v.type()) {
  case type::UNINITIALIZED: throw …;
  case type::DISCARDED: throw …;
  case type::NULL_: c.null(); return;
  case type::BOOLEAN: c.boolean(v.get_boolean()); return;
  case type::SIGNED: c.number(v.get_signed()); return;
  case type::UNSIGNED: c.number(v.get_unsigned()); return;
  case type::DOUBLE: c.number(v.get_double()); return;
  case type::STRING: c.string(v.get_string()); return;
  …
  }
}
```

SAX producer

```
template<typename Consumer>
void from_value(const value& v, Consumer& c)
{
  switch(v.type()) {
  …
  case type::ARRAY:
    c.begin_array();
    for(const auto& e : v.get_array()) {
      from_value(e, c);
      c.element();
    }
    c.end_array();
    return;
  …
  }
}
```

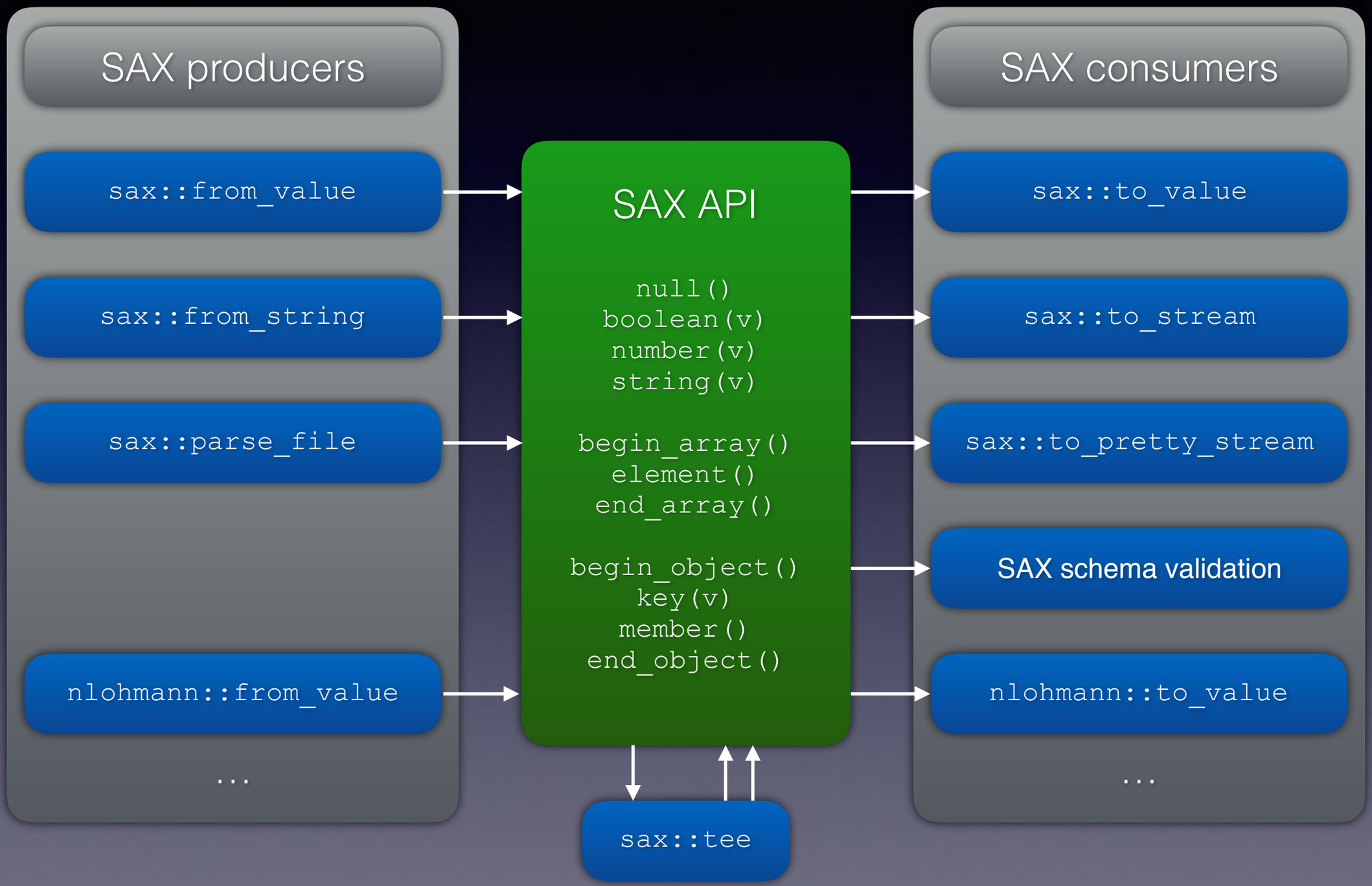
SAX producer

```
template<typename Consumer>
void from_value(const value& v, Consumer& c)
{
  switch(v.type()) {
  …
  case type::OBJECT:
    c.begin_object();
    for(const auto& e : v.get_object()) {
      c.key(e.first);
      from_value(e.second, c);
      c.member();
    }
    c.end_object();
    return;
  …
  }
}
```

SAX producer

```
template<typename Consumer>
void from_value(const value& v, Consumer& c)
{
  switch(v.type()) {
  …
  case type::RAW_PTR:
    if(const auto* p = v.get_raw_ptr()) {
      from_value(*p, c);
    }
    else {
      c.null();
    }
    return;
  }
  throw …;
}
```

SAX


SAX producers
sax::from_value
sax::from_string
sax::parse_file
nlohmann::from_value
...
SAX API
null()
boolean(v)
number(v)
string(v)
begin_array()
element()
end_array()
begin_object()
key(v)
member()
end_object()
sax::tee
SAX consumers
sax::to_value
sax::to_stream
sax::to_pretty_stream
SAX schema validation
nlohmann::to_value
...

SAX consumer

```
class to_stream
{
private:
  std::ostream& os;
  bool first = true;

  void next() { if(!first) os << ','; }

public:
  explicit to_stream(std::ostream& os) noexcept : os(os) {}

  …
};
```

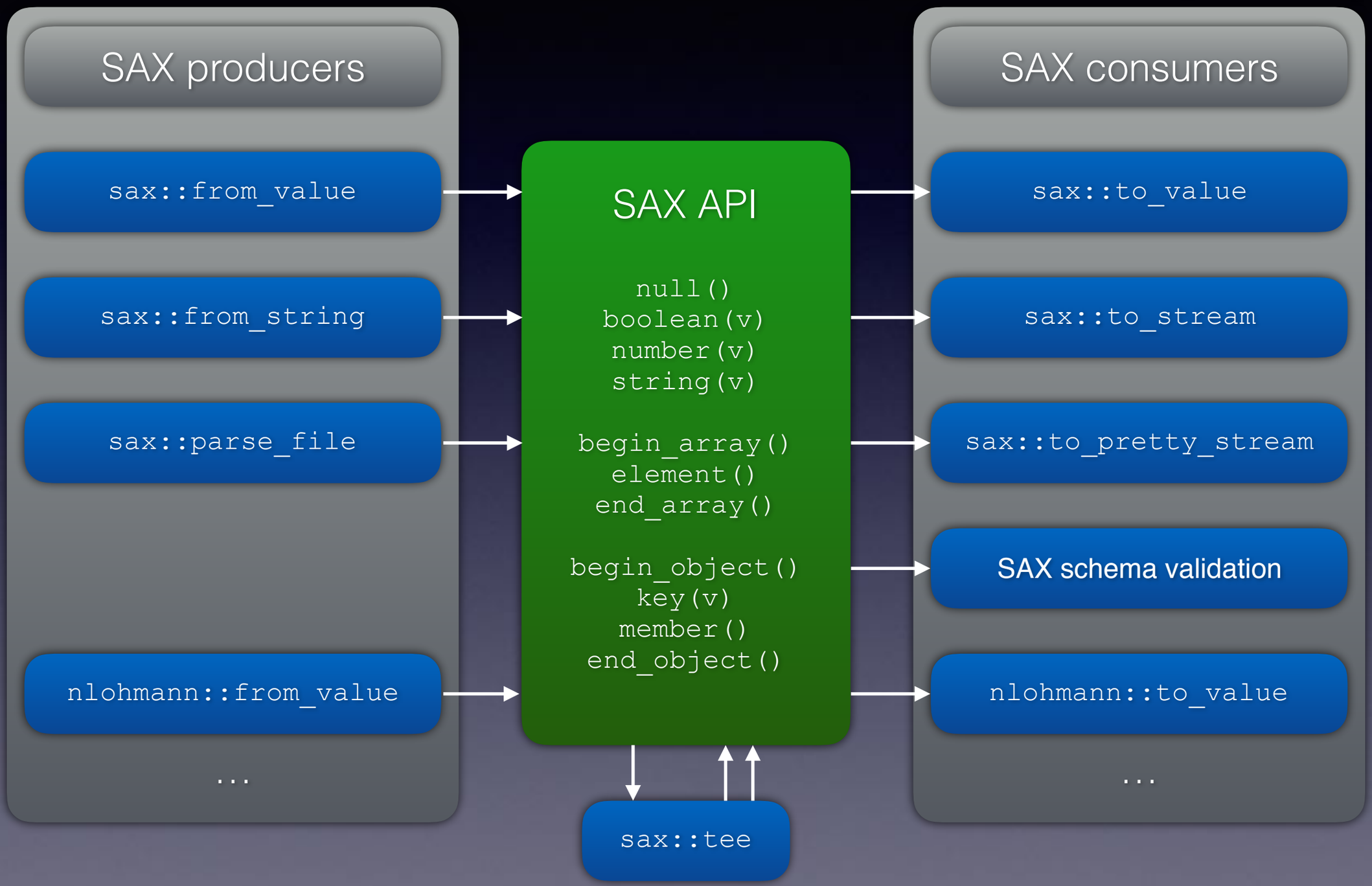
SAX consumer

```
class to_stream
{
  …

  void null() { next(); os << “null”; }

  void boolean(const bool v)
  { next(); os << (v ? “true” : “false”); }

  void number(const std::int64_t v) { next(); os << v; }
  void number(const std::uint64_t v) { next(); os << v; }
  void number(const double v) { next(); os << v; }

  void string(const std::string& v)
  { next(); os << ’”’ << escape(v) << ’”’; }

  …
};
```

SAX consumer

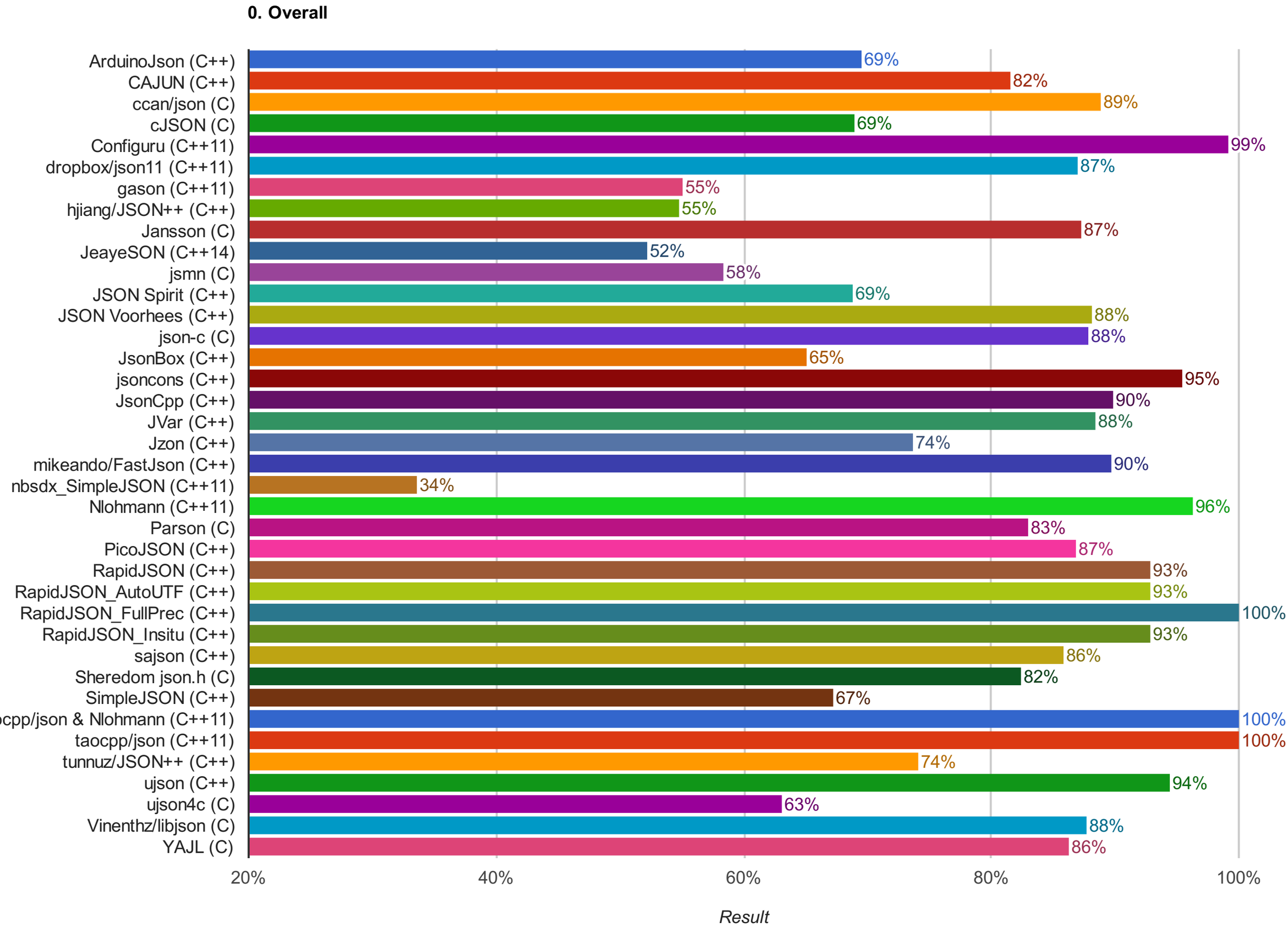
```
class to_stream
{
  …

  void begin_array() { next(); os << '['; first = true; }
  void element() { first = false; }
  void end_array() { os << ']'; }

  void begin_object() { next(); os << '{'; first = true; }
  void key(const std::string& v)
  { string(v); os << ':'; first = true; }
  void member() { first = false; }
  void end_object() { os << '}'; }
};
```

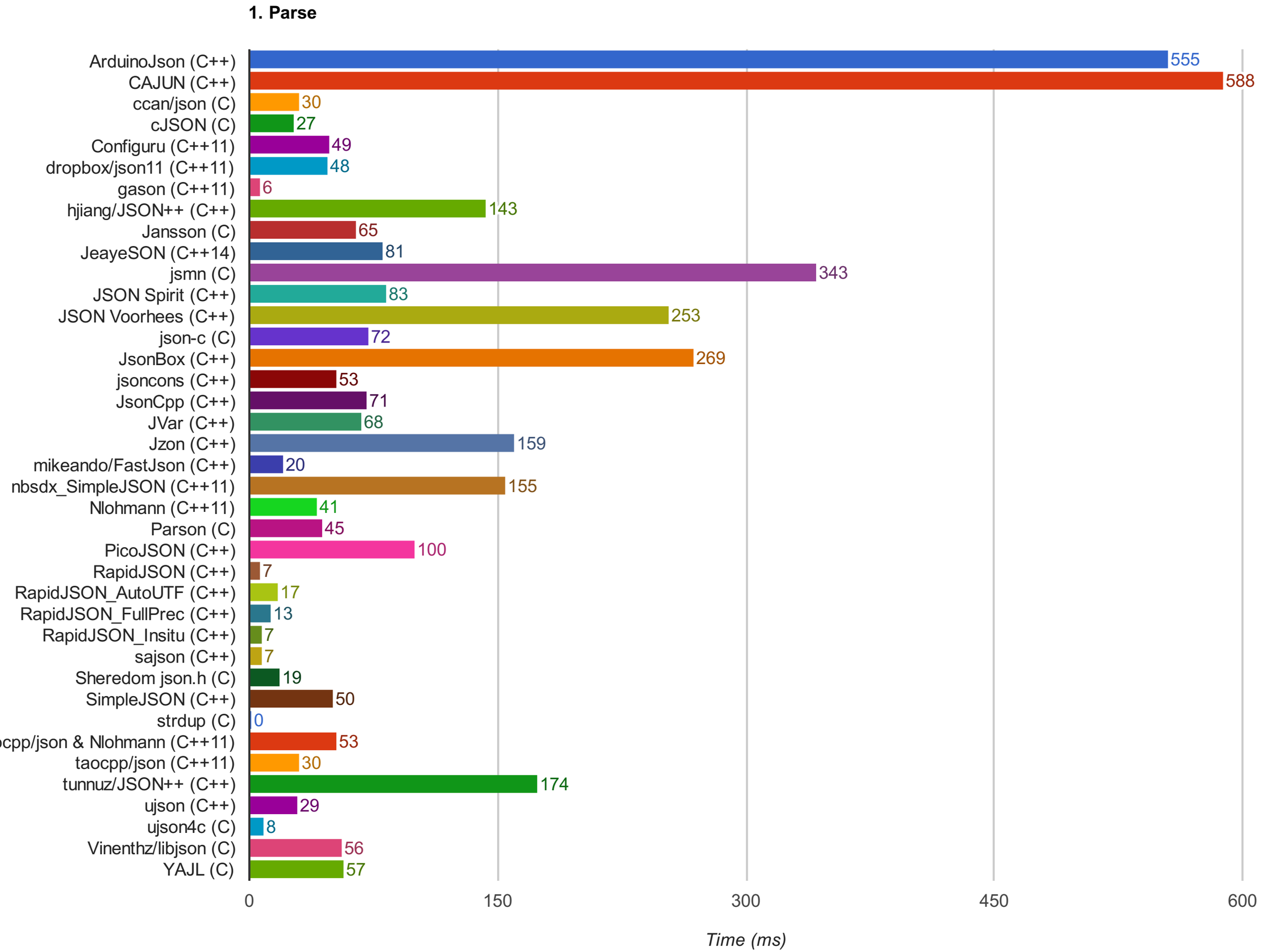
SAX


SAX producers
sax::from_value
sax::from_string
sax::parse_file
nlohmann::from_value
...
SAX API
null()
boolean(v)
number(v)
string(v)
begin_array()
element()
end_array()
begin_object()
key(v)
member()
end_object()
sax::tee
SAX consumers
sax::to_value
sax::to_stream
sax::to_pretty_stream
SAX schema validation
nlohmann::to_value
...

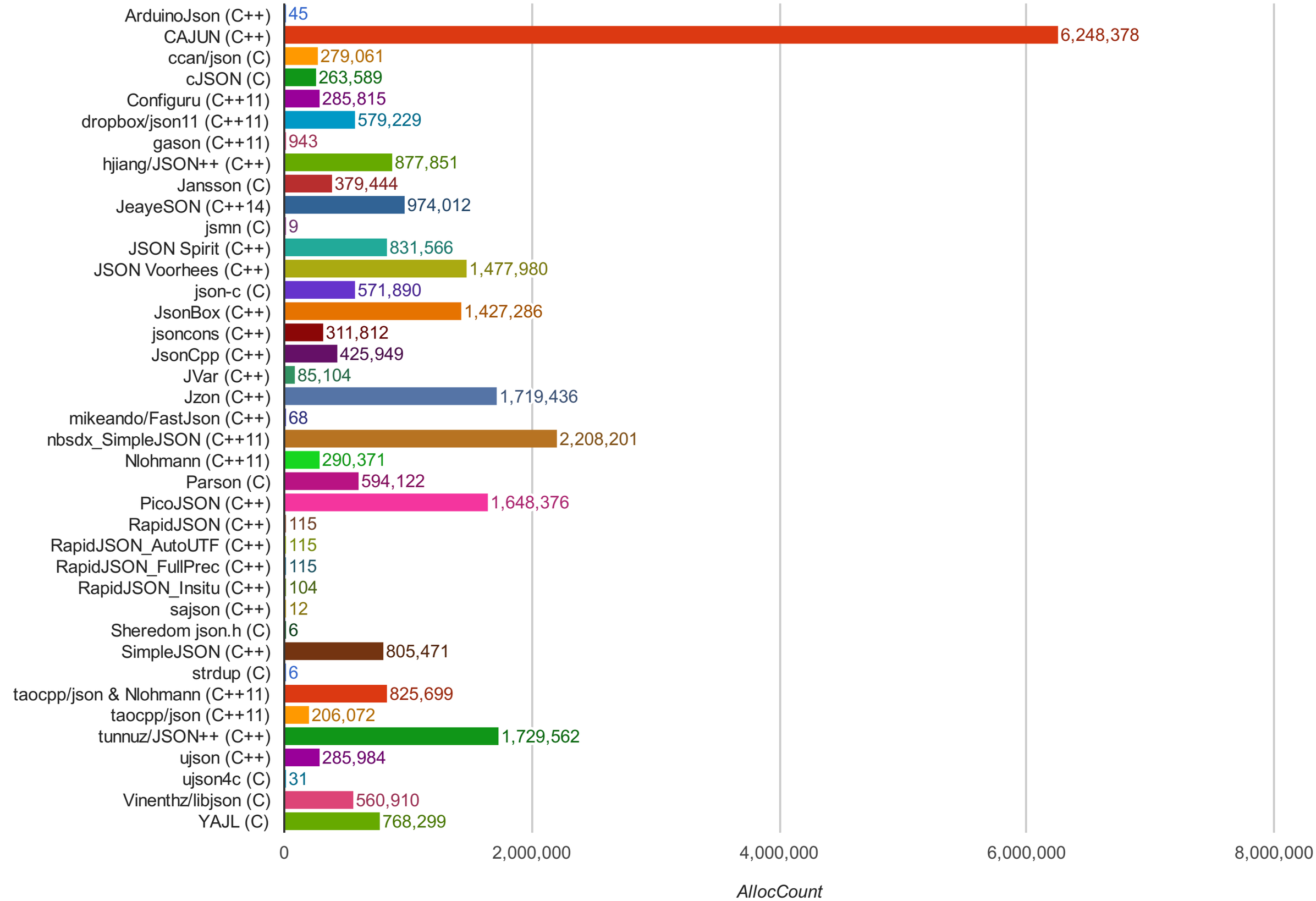
SAX decoupling

- DOM-less usage, e.g., parse and pretty print a file via SAX for extremely large JSON files
- Parse and generate binary formats (BSON, BJSON, UBJSON, …)
- Combine with other JSON libraries
- With `tee`, feed events to multiple consumers

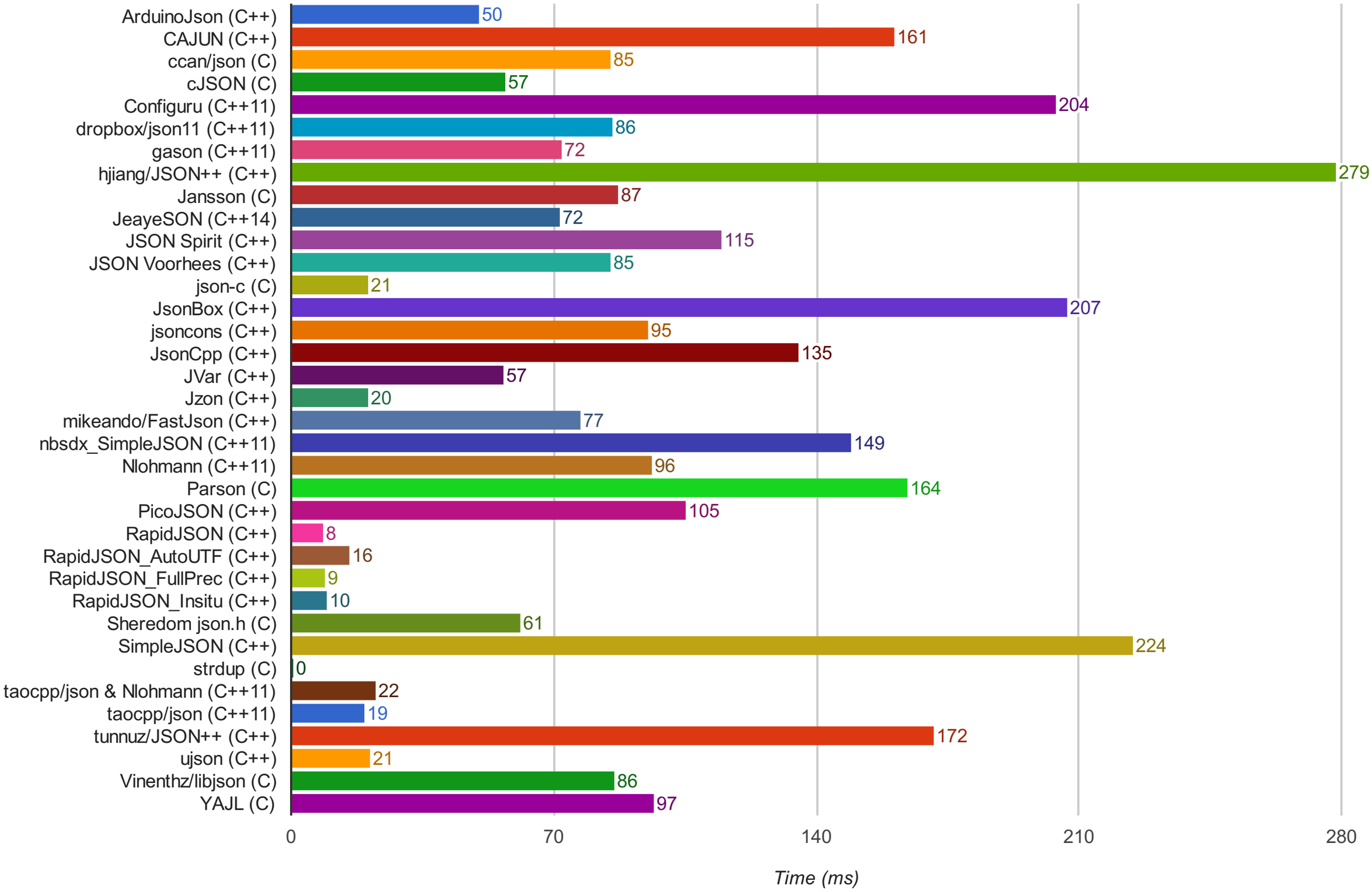

0. Overall
ArduinoJson (C++) 69%
CAJUN (C++) 82%
ccan/json (C) 89%
cJSON (C) 69%
Configuru (C++11) 99%
dropbox/json11 (C++11) 87%
gason (C++11) 55%
hjiang/JSON++ (C++) 55%
Jansson (C) 87%
JeayeSON (C++14) 52%
jsmn (C) 58%
JSON Spirit (C++) 69%
JSON Voorhees (C++) 88%
json-c (C) 88%
JsonBox (C++) 65%
jsoncons (C++) 95%
JsonCpp (C++) 90%
JVar (C++) 88%
Jzon (C++) 74%
mikeando/FastJson (C++) 90%
nbsdx_SimpleJSON (C++11) 34%
Nlohmann (C++11) 96%
Parson (C) 83%
PicoJSON (C++) 87%
RapidJSON (C++) 93%
RapidJSON_AutoUTF (C++) 93%
RapidJSON_FullPrec (C++) 100%
RapidJSON_Insitu (C++) 93%
sajson (C++) 86%
Sheredom json.h (C) 82%
SimpleJSON (C++) 67%
taocpp/json & Nlohmann (C++11) 100%
taocpp/json (C++11) 100%
tunnuz/JSON++ (C++) 74%
ujson (C++) 94%
ujson4c (C) 63%
Vinenthz/libjson (C) 88%
YAJL (C) 86%
20%
40%
60%
80%
100%
Result


1. Parse
ArduinoJson (C++) 555
CAJUN (C++) 588
ccan/json (C) 30
cJSON (C) 27
Configuru (C++11) 49
dropbox/json11 (C++11) 48
gason (C++11) 6
hjiang/JSON++ (C++) 143
Jansson (C) 65
JeayeSON (C++14) 81
jsmn (C) 343
JSON Spirit (C++) 83
JSON Voorhees (C++) 253
json-c (C) 72
JsonBox (C++) 269
jsoncons (C++) 53
JsonCpp (C++) 71
JVar (C++) 68
Jzon (C++) 159
mikeando/FastJson (C++) 20
nbsdx_SimpleJSON (C++11) 155
Nlohmann (C++11) 41
Parson (C) 45
PicoJSON (C++) 100
RapidJSON (C++) 7
RapidJSON_AutoUTF (C++) 17
RapidJSON_FullPrec (C++) 13
RapidJSON_Insitu (C++) 7
sajson (C++) 7
Sheredom json.h (C) 19
SimpleJSON (C++) 50
strdup (C) 0
taocpp/json & Nlohmann (C++11) 53
taocpp/json (C++11) 30
tunnuz/JSON++ (C++) 174
ujson (C++) 29
ujson4c (C) 8
Vinenthz/libjson (C) 56
YAJL (C) 57
0
150
300
450
600
Time (ms)

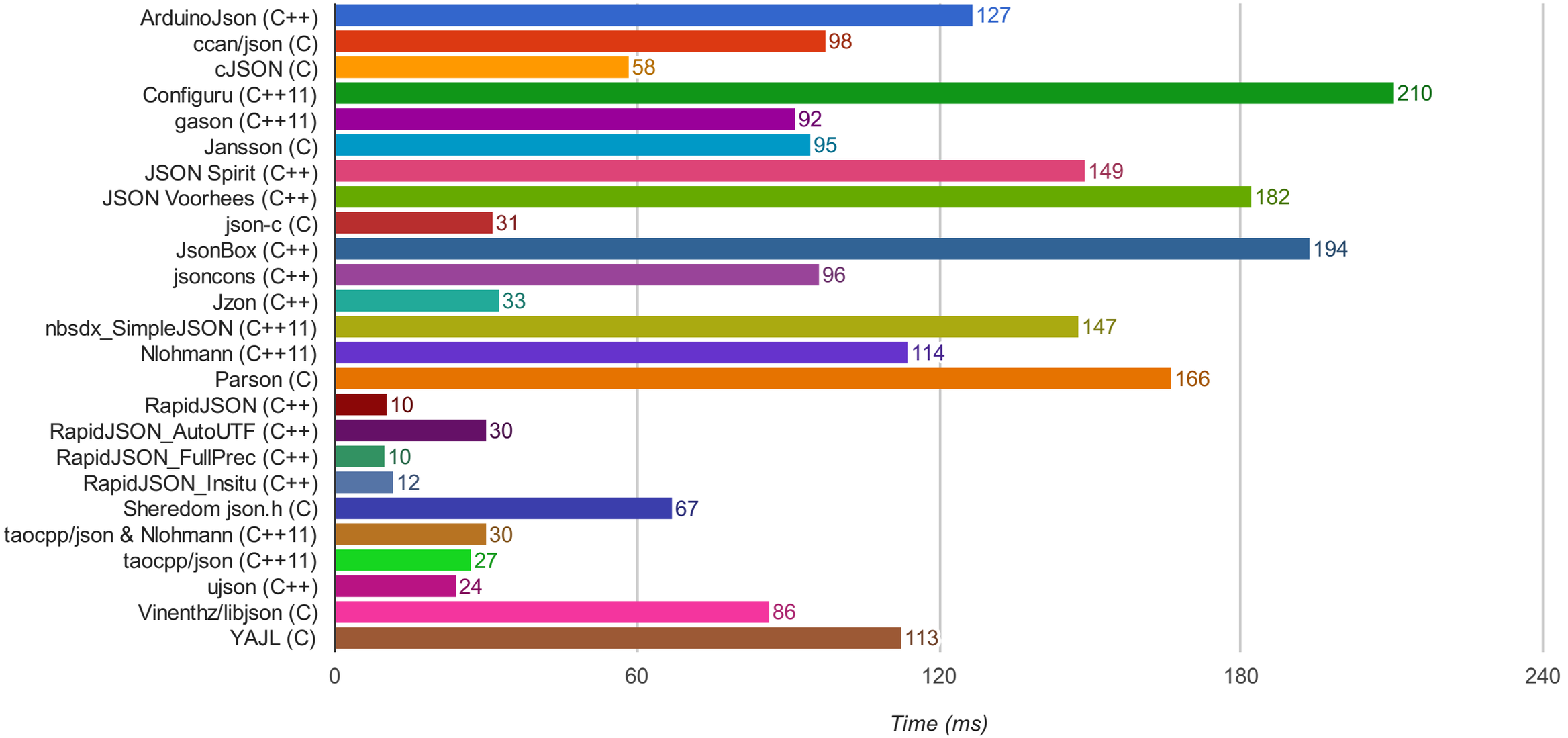
1. Parse


ArduinoJson (C++) 45
CAJUN (C++) 6,248,378
ccan/json (C) 279,061
cJSON (C) 263,589
Configuru (C++11) 285,815
dropbox/json11 (C++11) 579,229
gason (C++11) 943
hjiang/JSON++ (C++) 877,851
Jansson (C) 379,444
JeayeSON (C++14) 974,012
jsmn (C) 9
JSON Spirit (C++) 831,566
JSON Voorhees (C++) 1,477,980
json-c (C) 571,890
JsonBox (C++) 1,427,286
jsoncons (C++) 311,812
JsonCpp (C++) 425,949
JVar (C++) 85,104
Jzon (C++) 1,719,436
mikeando/FastJson (C++) 68
nbsdx_SimpleJSON (C++11) 2,208,201
Nlohmann (C++11) 290,371
Parson (C) 594,122
PicoJSON (C++) 1,648,376
RapidJSON (C++) 115
RapidJSON_AutoUTF (C++) 115
RapidJSON_FullPrec (C++) 115
RapidJSON_Insitu (C++) 104
sajson (C++) 12
Sheredom json.h (C) 6
SimpleJSON (C++) 805,471
strdup (C) 6
taocpp/json & Nlohmann (C++11) 825,699
taocpp/json (C++11) 206,072
tunnuz/JSON++ (C++) 1,729,562
ujson (C++) 285,984
ujson4c (C) 31
Vinenthz/libjson (C) 560,910
YAJL (C) 768,299
0
2,000,000
4,000,000
6,000,000
8,000,000
AllocCount

2. Stringify


ArduinoJson (C++) 50
CAJUN (C++) 161
ccan/json (C) 85
cJSON (C) 57
Configuru (C++11) 204
dropbox/json11 (C++11) 86
gason (C++11) 72
hjiang/JSON++ (C++) 279
Jansson (C) 87
JeayeSON (C++14) 72
JSON Spirit (C++) 115
JSON Voorhees (C++) 85
json-c (C) 21
JsonBox (C++) 207
jsoncons (C++) 95
JsonCpp (C++) 135
JVar (C++) 57
Jzon (C++) 20
mikeando/FastJson (C++) 77
nbsdx_SimpleJSON (C++11) 149
Nlohmann (C++11) 96
Parson (C) 164
PicoJSON (C++) 105
RapidJSON (C++) 8
RapidJSON_AutoUTF (C++) 16
RapidJSON_FullPrec (C++) 9
RapidJSON_Insitu (C++) 10
Sheredom json.h (C) 61
SimpleJSON (C++) 224
strdup (C) 0
taocpp/json & Nlohmann (C++11) 22
taocpp/json (C++11) 19
tunnuz/JSON++ (C++) 172
ujson (C++) 21
Vinenthz/libjson (C) 86
YAJL (C) 97
0
70
140
210
280
Time (ms)

3. Prettify


ArduinoJson (C++)
ccan/json (C)
cJSON (C)
Configuru (C++11)
gason (C++11)
Jansson (C)
JSON Spirit (C++)
JSON Voorhees (C++)
json-c (C)
JsonBox (C++)
jsoncons (C++)
Jzon (C++)
nbsdx_SimpleJSON (C++11)
Nlohmann (C++11)
Parson (C)
RapidJSON (C++)
RapidJSON_AutoUTF (C++)
RapidJSON_FullPrec (C++)
RapidJSON_Insitu (C++)
Sheredom json.h (C)
taocpp/json & Nlohmann (C++11)
taocpp/json (C++11)
ujson (C++)
Vinenthz/libjson (C)
YAJL (C)
127
98
58
210
92
95
149
182
31
194
96
33
147
114
166
10
30
10
12
67
30
27
24
86
113
0
60
120
180
240
Time (ms)

Thank you!

https://github.com/taocpp/json

Questions?