Higher Level C++
with
The Boost Libraries

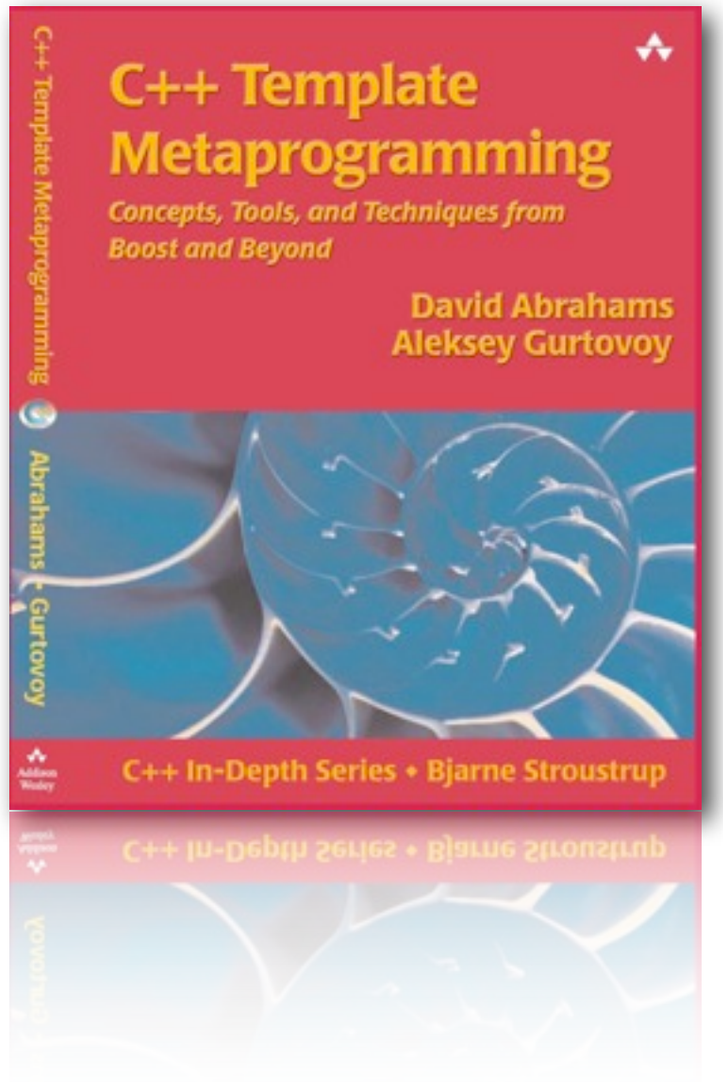
http://github.com/boostpro/bbn-2012-11

boostpro computing

Dave Abrahams

(me)

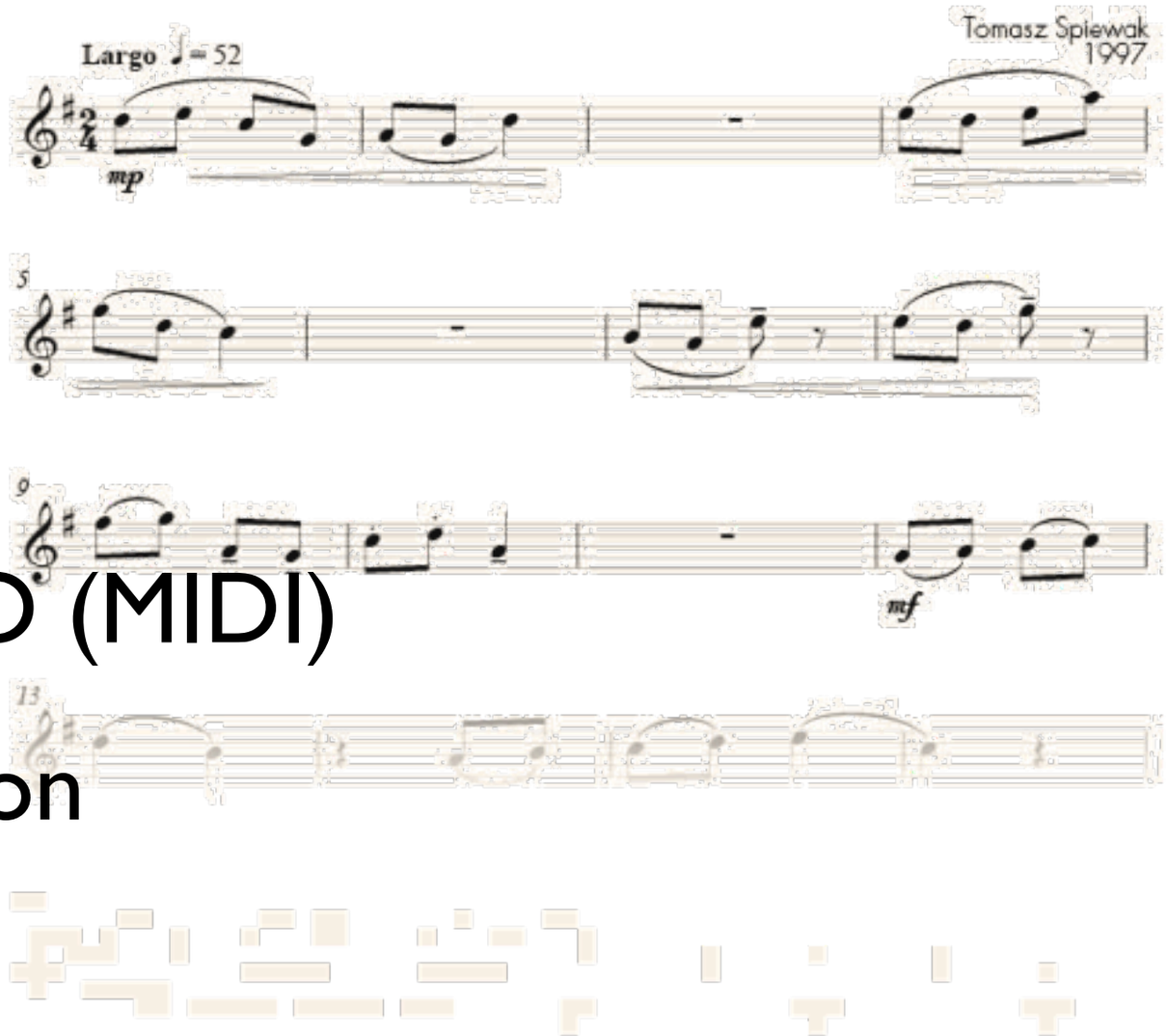
- C++ Committee member since 1996
- Founding member, Boost.org
- Founder, BoostPro Computing
- Trainer, consultant, software developer
- Author



boostpro computing

My World ca. 1995

- Page Layout
- Printing
- GUI Design
- Asynchronous I/O (MIDI)
- AI for Transcription
- Portability



boostpro computing

Infrastructure Requirements

- GUI framework
- Document Framework (undo/serialization)
- Dynamic Arrays
- Algorithms, e.g. Sort, Binary Search
- Fast Dynamic Memory
- Error Handling

Build It Yourself

- Evolves by accretion, not by design
- Limited in capability
- One-offs: not interoperable
- Undocumented
- Not speed-tested
- Buggy (it turns out)

boostpro computing

The Bottom Line

A professional programmer has a lot on his/her mind!

boostpro computing

Enter: STL

- Reliable containers and algorithms
- Documented requirements/guarantees
- Choice of capability/performance tradeoffs
- Interoperability with low coupling
- Standardization ➔ lingua franca
- Programming Paradigm ➔ less thinking!

boostpro computing

Background:

Effective programmers focus on solving problems in their application's domain, not on writing reliable, efficient, general-purpose building blocks.

boostpro computing

But Enough About Me...

- Why are you here?
- What libraries are you using?
- What kinds of things do you work on?

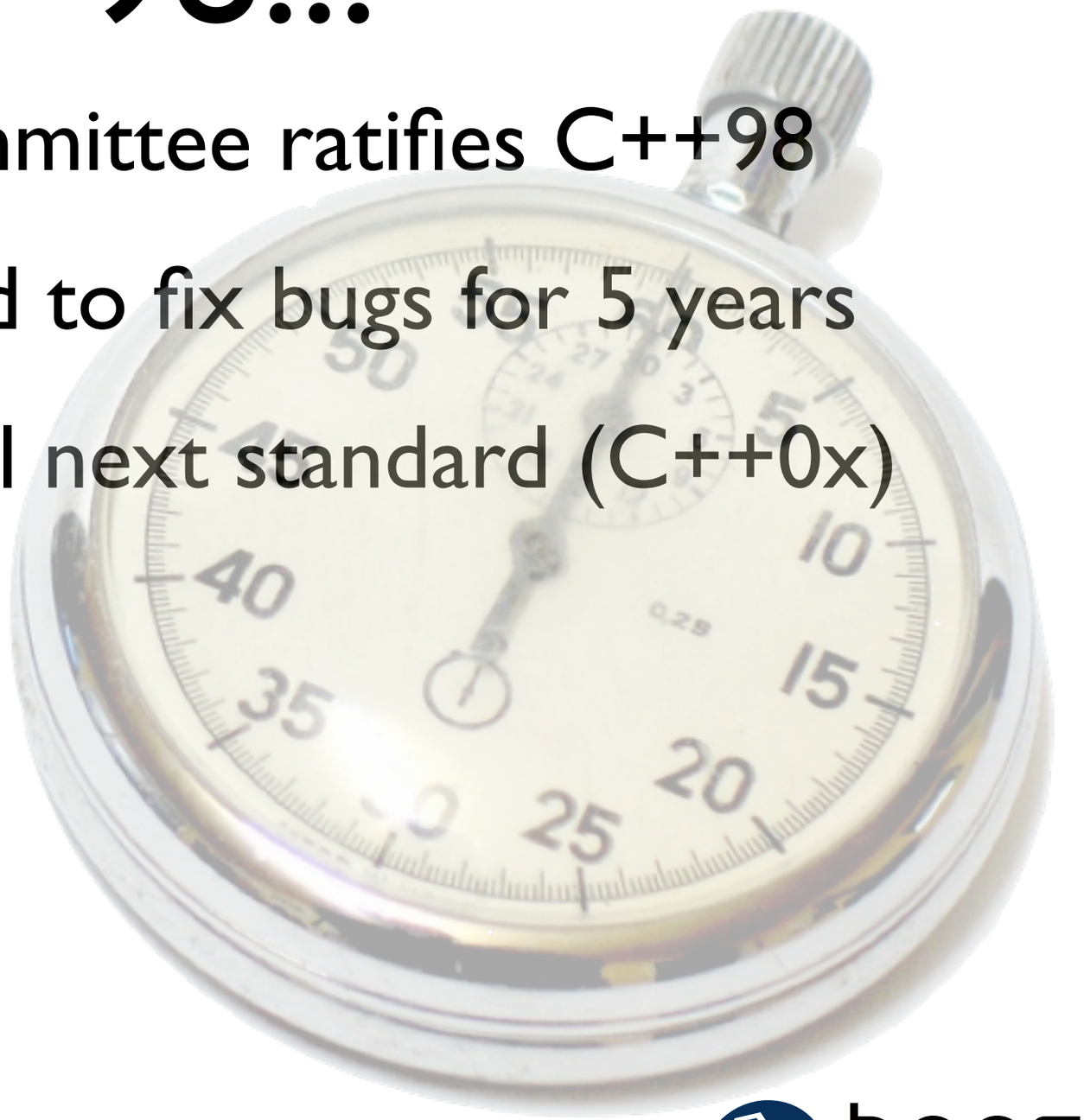
We *Need* High-Level Libraries

- Less code ⇒ Real productivity
  - Less to write
  - Less to debug
- More expressive code
  - Natural to write
  - More self-documenting
  - More likely to be correct the first time

boostpro computing

It All Started Back In '98...

- Standards committee ratifies C++98
- Only supposed to fix bugs for 5 years
- 10+ years until next standard (C++0x)



boostpro computing

It All Started Back In '98...

- Standards committee ratifies C++98
- Only supposed to fix bugs for 5 years
- 10+ years until next standard (C++0x)

Beman Dawes

boostpro computing

It All Started Back In '98...

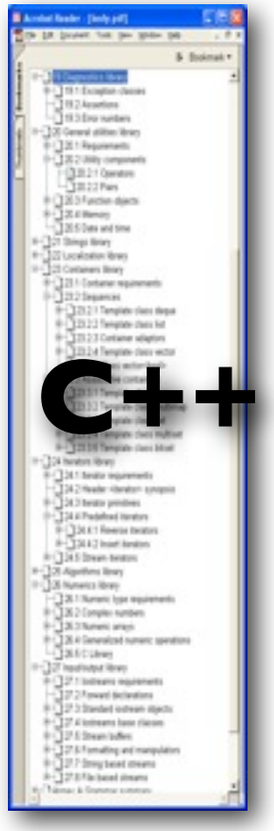
- Standards committee ratifies C++98
- Only supposed to fix bugs for 5 years
- 10+ years until next standard (C++0x)

- How will get the C++0x std:: libraries?
- Will they be based on "existing practice?"

Beman Dawes

boostpro computing

Standard Library
DEATHMATCH

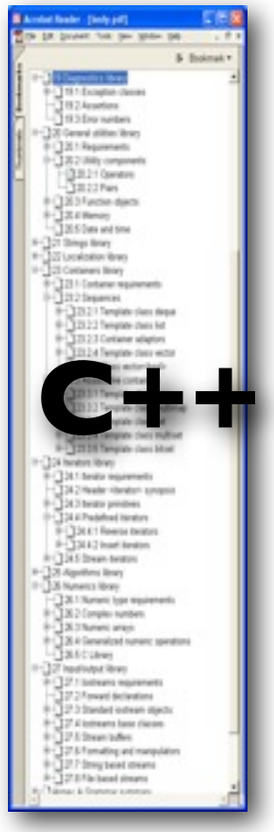


boostpro computing

Standard Library
DEATHMATCH

Python

Java



C++

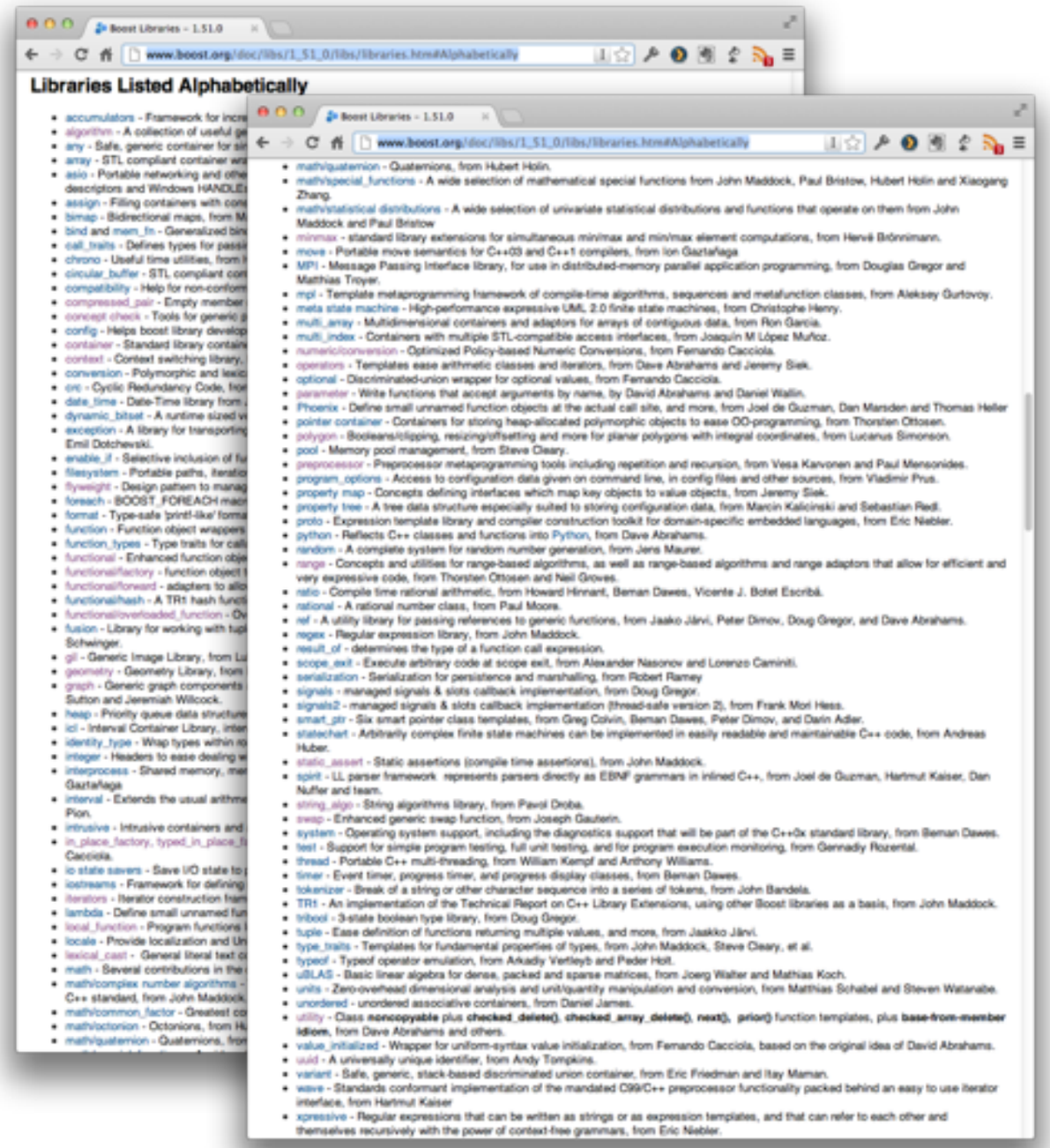
CPAN

Perl

boostpro computing

- To Encourage Adoption ("Practice"):
  - Open Source
  - Peer Reviewed
  - Licensed Non-Virally
- Suitable for Standardization
  - Portable
  - Well Documented

Where Are the Libraries?



Boost:
117 libraries
and growing!
(10 in C++11)

boostpro computing

What's In Boost: Domains

- Text Processing / Parsing
- Data Structures
- Iterators
- Algorithms
- Function Objects
- Generic Programming Utilities
- Concurrency
- Metaprogramming / Code Generation
- Numerics
- Correctness / Testing
- Input / Output
- Language Binding
- Memory Management
- Programming Interfaces
- …etc.

Who's Using Boost

- Adobe
- Microsoft
- RealNetworks
- CERN (Large Hadron Collider)
- SAP NetWeaver
- National Labs (LLNL, LBL, Sandia)
- …etc.

boostpro computing

FOSS Using Boost (Mac)

```
$ grep -r -l port:boost . | sed -e 's#\./.*/\(.*\)/Portfile#\1#g'
```


boostpro computing

FOSS Using Boost (Mac)

```
$ grep -r -l port:boost . | sed -e 's#\./.*/\(.*\)/Portfile#\1#g'
```

LyX
mumble
qtiplot
ardour2
xmms2
mongodb
mysql-connector-cpp
simplevoc-open
soci-devel
akonadi
arabica
boost-build
boost-gil-numeric
flusspferd
json_spirit
libbert
libnifalcon
librets
monotone
orocos-kdl
orocos-rtt
thrift
xmlwrapp
bitcoin
QuantLib
encfs
fife
glob2
PlasmaClient
wesnoth
cgal
agave
assimp
enblend
exempi
field3d
hugin-app
inkscape
inkscape-devel
lib2geom
libopenraw
mkhexgrid
ogre
openvrml
scantailor
vigra
vtk-devel
digikam
kdepimlibs4
kdesdk4
kdevplatform
kgraphviewer
ktorrent4
libktorrent
rocs
prothon
fityk
vowpal_wabbit
mkvtoolnix
XBMC
cclive
deluge
libtorrent-rasterbar
libtorrent-rasterbar-devel
metaproxy
mosh
murmur
yazproxy
./PortIndex
scribus
py-graph-tool
py26-mapnik
bali-phy
collada-dom
cufflinks
gnuradio-gruel
gnuradio-omnithread
iAIDA
indi
ompl
peekabot-client
peekabot-server
playerstage-player
uhd
source-highlight
zorba

FOSS Using Boost (Ubuntu)

```
$ grep-dctrl -F Build-Depends libboost -s Package /var/lib/apt/lists/*_Sources
```

FOSS Using Boost (Ubuntu)

```
$ grep-dctrl -F Build-Depends libboost -s Package /var/lib/apt/lists/*_Sources
```

0ad
abyss
actionaz
agave
akonadi
akonadi-facebook
akonadi-googledata
anytun
aptitude
aqsis
ardour
armadillo
asc
asio
aspcud
assimp
autodock-vina
avogadro
ball
barada-pam
barry
bastet
bibletime
bitcoin
blender
blobby
bombono-dvd
boost-defaults
boost-mpi-source1.46
boost-mpi-source1.49
btag
calibre
calligra
ccbuild
cclive
cegui-mk2
ceph
cgal
clamfs
clementine
compiz
compizconfig-backend-gconf
compiz-plugins-extra
compiz-plugins-main
composite
condor
csound
cufflinks
cupt
dans-gdal-scripts
dc-qt
deal.ii
debian-xcontrol
diet
digikam
disulfinder
dolfin
drizzle
dssp
dvswitch
dynare
easystroke
eiskaltdcpp
ekiga
ember
enblend-enfuse
encfs
esperanza
esys-particle
evolvotron
fatrat
feel++
fgrun
fife
fityk
flamerobin
flann
flightgear
flowcanvas
flush
foundry
fracplanet
freecad
frogatto
fsl
gearmand
gecode
getfem++
gle-graphics
glob2
glogg
glom
gnash
gnelib
gnote
gnuradio
gofigure2
goldencheetah
gource
gpick
gpsdrive
gpsshogi
grfcodec
gringo
guitarix
highlight
hugin
ibus-pinyin
imagevis3d
inkscape
innoextract
json-spirit
k3d
kalzium
kcollectd
kdenetwork
kdepim
kdepimlibs
kdepim-runtime
kdeplasma-addons
kdesdk
kdevplatform
kdewebdev
kde-workspace
kgraphviewer
kicad
klick
kmymoney
kopete-cryptography
kraft
ktorrent
ladish
launchy
libavg
libclaw
libcmis
libcompizconfig
libftdi
libgdf
libindi
libkolab
libkolabxml
libktorrent
libmpikmeans
libopenraw
liborigin2
libosl
libreoffice
librime
libtorrent-rasterbar
libvigraimpex
libvisio
libwps
libzeep
licq
lightspark
linuxdcpp
lordsawar
love
luabind
lv2-c++-tools
lyx
magics++
mapnik
mcrl2
mdds
merkaartor
metalink
mididings
mira
mkvtoolnix
monav
mongodb
monotone
mosh
mp3diags
mudlet
mumble
mysql-connector-c++
mysql-workbench
ncbi-blast+
nemiver
noiz2sa
ns3
nux
ogre
ogre-1.8
openchange
opencsg
openimageio
openscad
openwalnut
openyahtzee
osmium
ovito
paraview
patchage
pathfinder
pcb2gcode
pdfcube
pdfedit
pdns
pdns-recursor
pentobi
performous
pingus
pinot
pion-net
plasma-widget-mail
player
plee-the-bear
pokerth
polybori
pp-popularity-contest
promoe
pycuda
pyexiv2
pykde4
pyopencl
pytango
python-demgengeo
pythonmagick
python-pgmagick
python-visual
qbittorrent
qpid-cpp
qsapecng
qsource-highlight
qt-gstreamer
qtiplot
quantlib
quantlib-swig
quickfix
qutecom
raul
rdkit
referencer
regina-normal
resiprocate
rheolef
rhmessaging
rivet
rlvm
rocs
rquantlib
rubyluabridge
scantailor
schroot
scribus
sdcc
seqan
sfftobmp
sffview
showq
sigviewer
sigx
simgear
sinfo
sitplus
smc
sofa-framework
solarpowerlog
source-highlight
source-highlight-ide
source-highlight-qt
spring
springlobby
srecord
sslsniff
stimfit
strigi
subcommander
supercollider
supertux
swift-im
syncevolution
systemtap
tagpy
tagua
tardy
thrift-compiler
tintii
toonloop
tophat
twinkle
ufc
uhd
undertaker
unity
vdr-plugin-live
viennacl
vmware-view-open-client
vocproc
vowpal-wabbit
voxbo
vtk
wesnoth-1.10
widelands
witty
xbmc
xboxdrv
xmlcopyeditor
xmms2
xorp
xsd
xylib
yade
yapgvb
yoshimi
zanshin
zookeeper

boostpro computing

Why Boost Matters

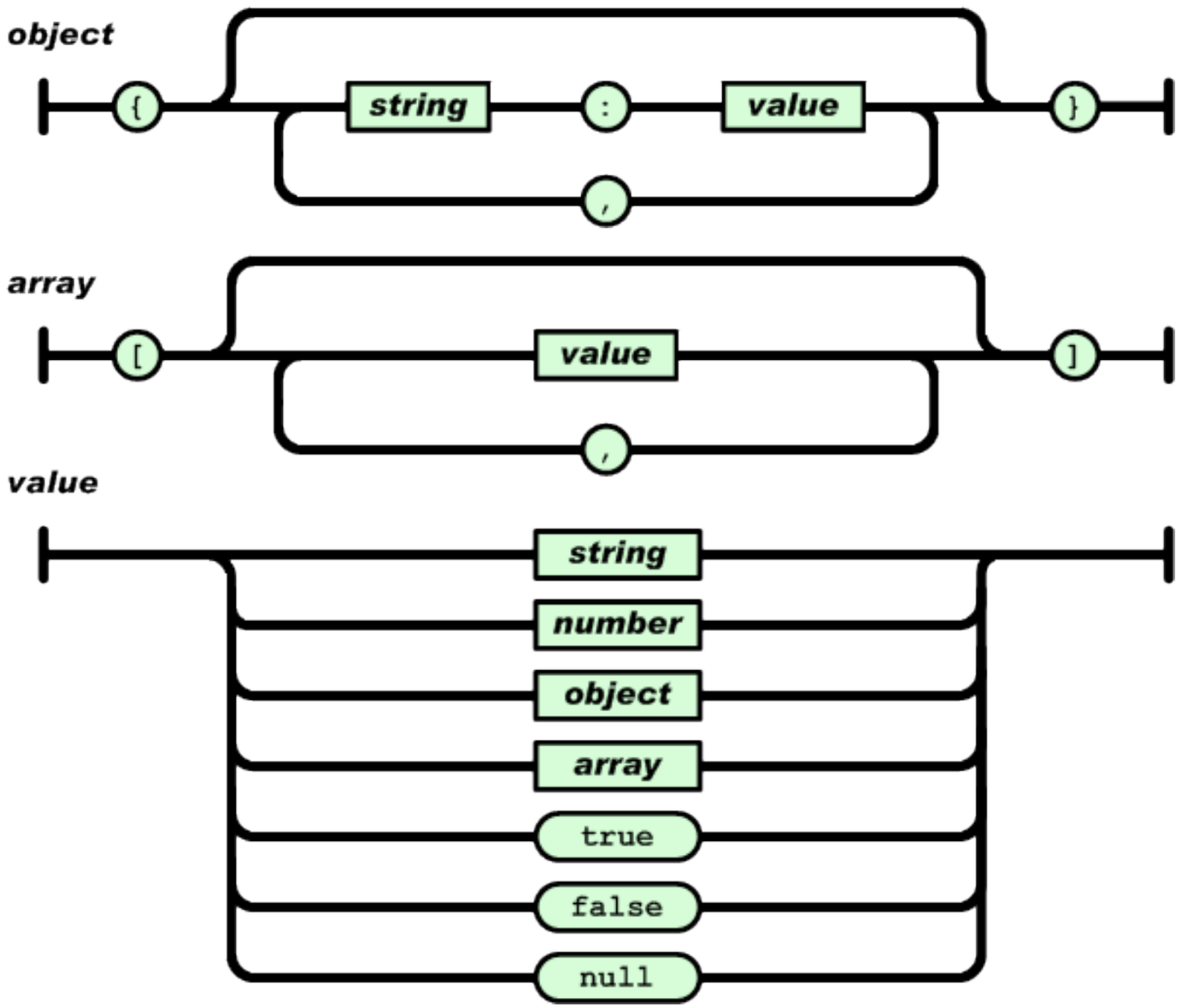
- Free high-level libraries
- Cutting-edge C++ practiced here!
  - Idioms & abstractions
  - Techniques
  - Best practices
- High quality
- Major contributor to C++ standard
- Widely used in major commercial software

The Boost Core

Basics, Touchstones and Idioms

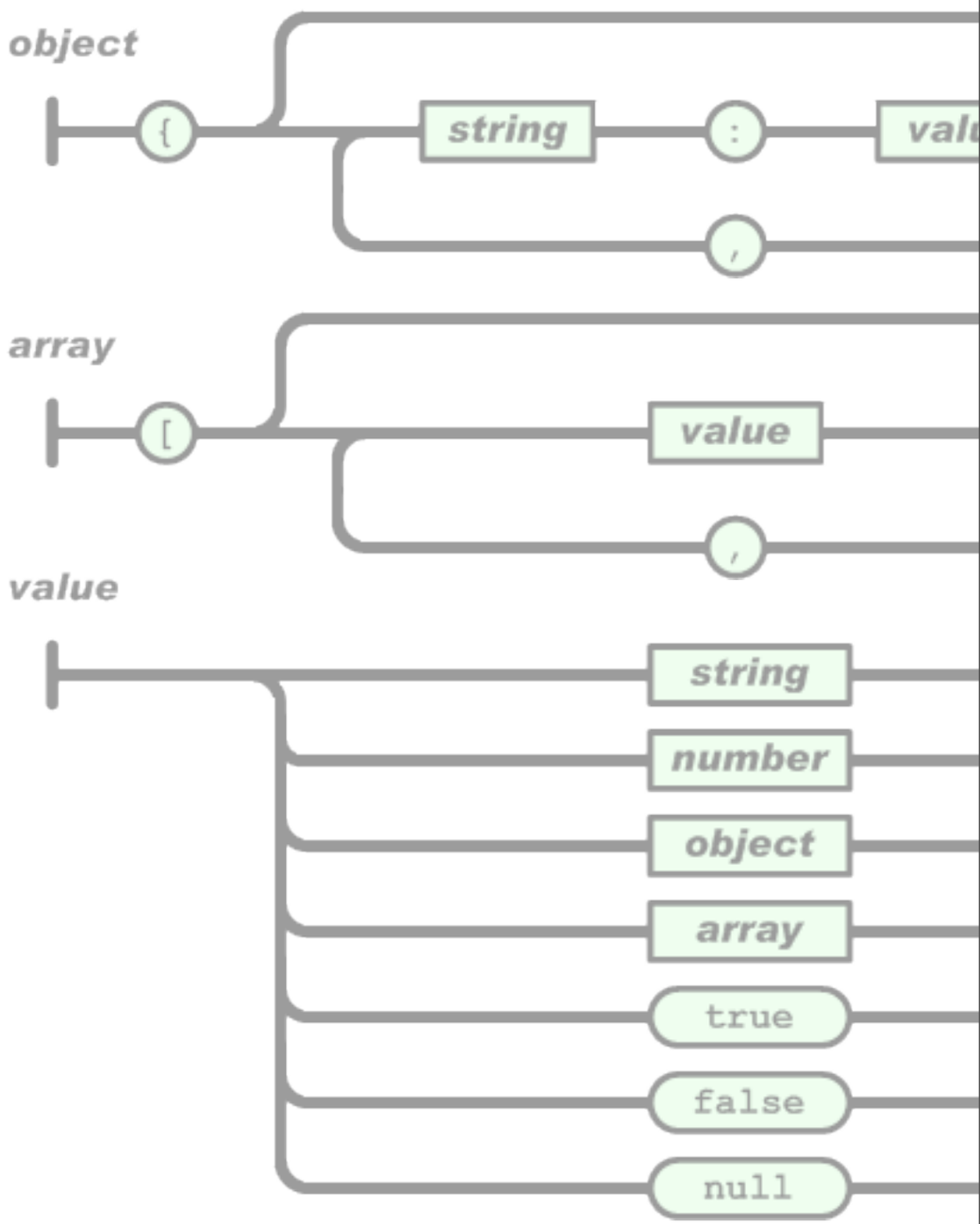
boostpro computing

JSON



boostpro computing

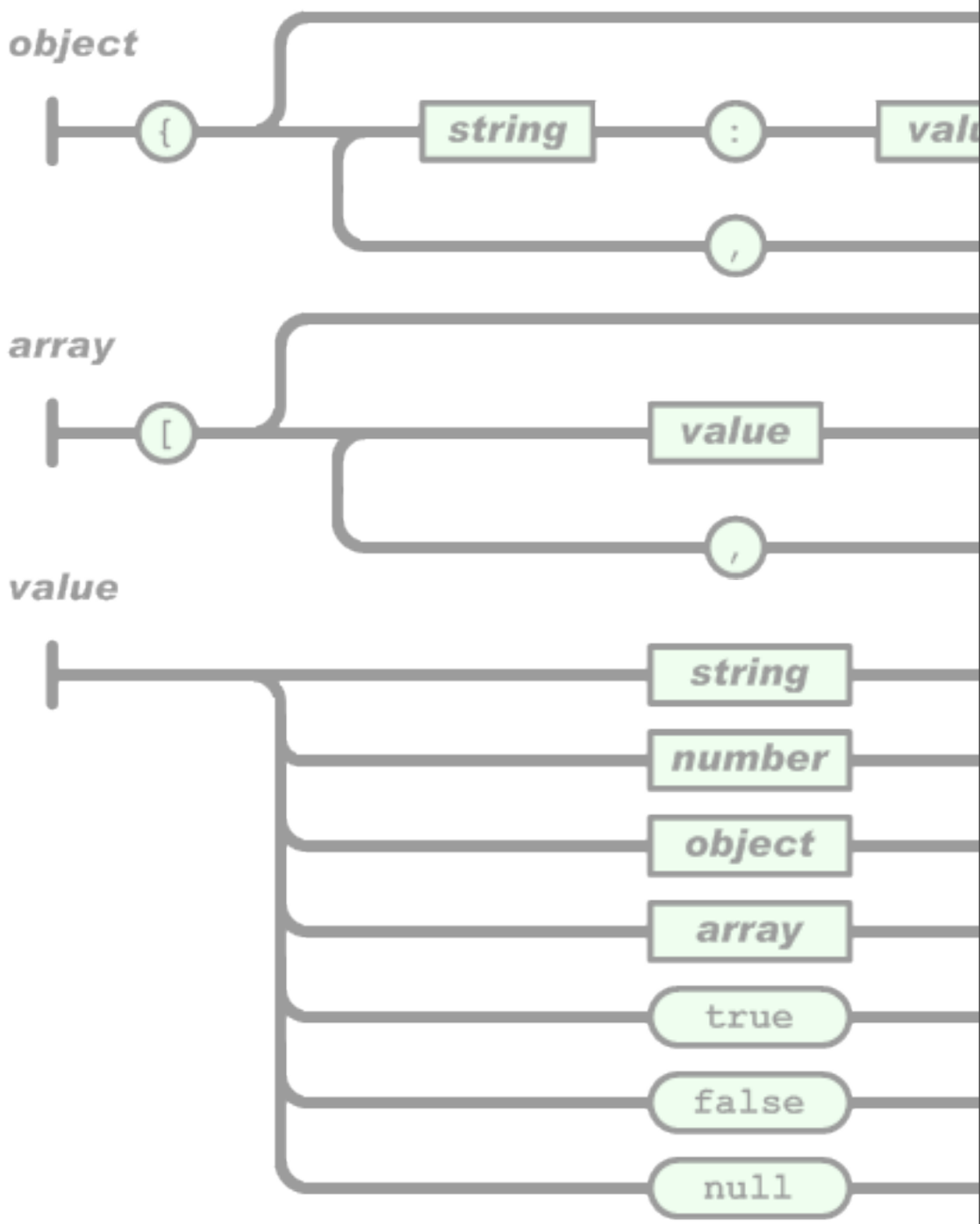
JSON



boostpro computing

JSON

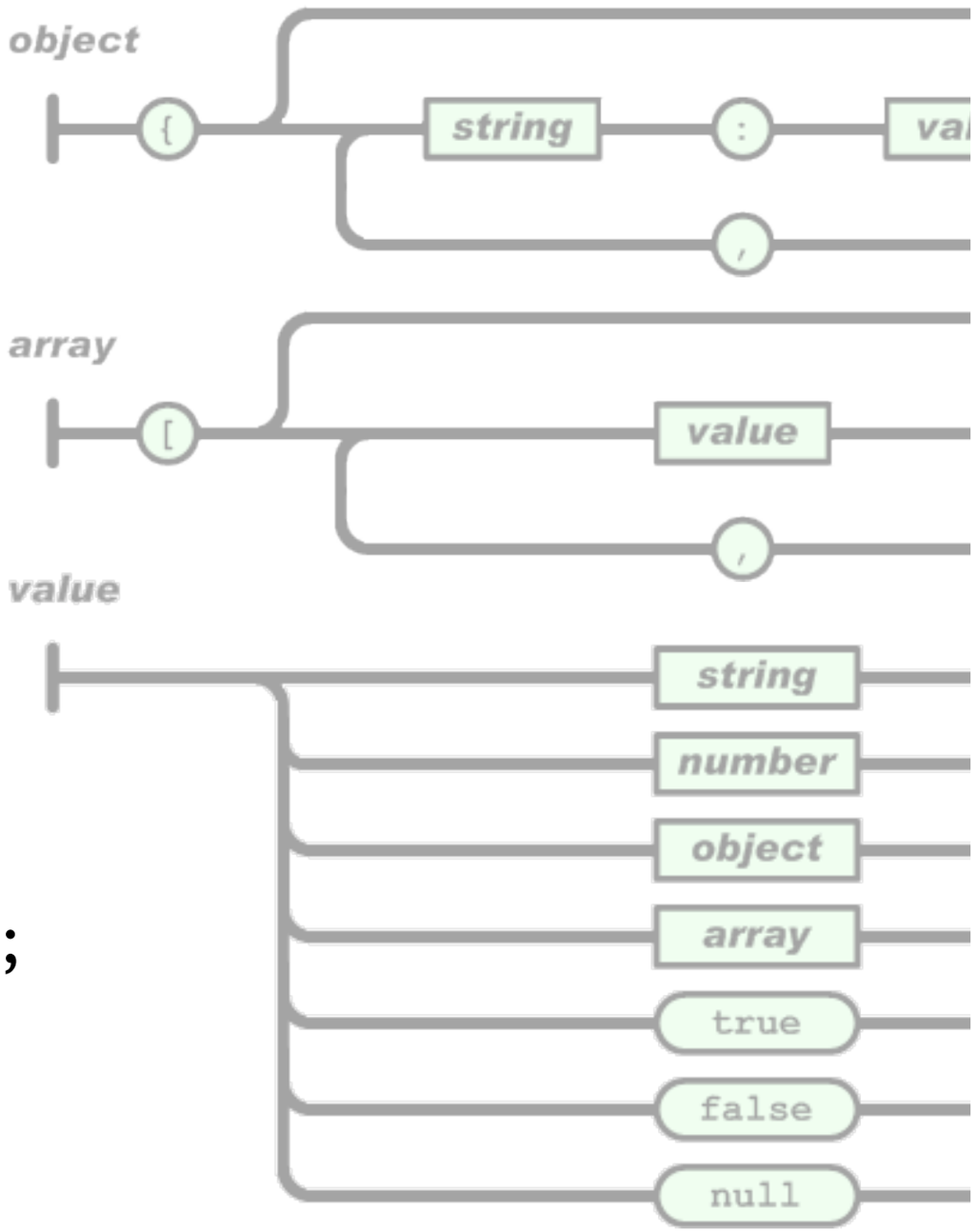
```
{
    "first": "Dave",
    "last": "Abrahams",
    "age": 48.2,
    "sex": "M",
    "zip code": "02143",
    "registered": true,
    "catchphrase": null
    "interests": [
        "Jamming",
        "Biking",
        "Hacking"
    ]
}
```



boostpro computing

Representation

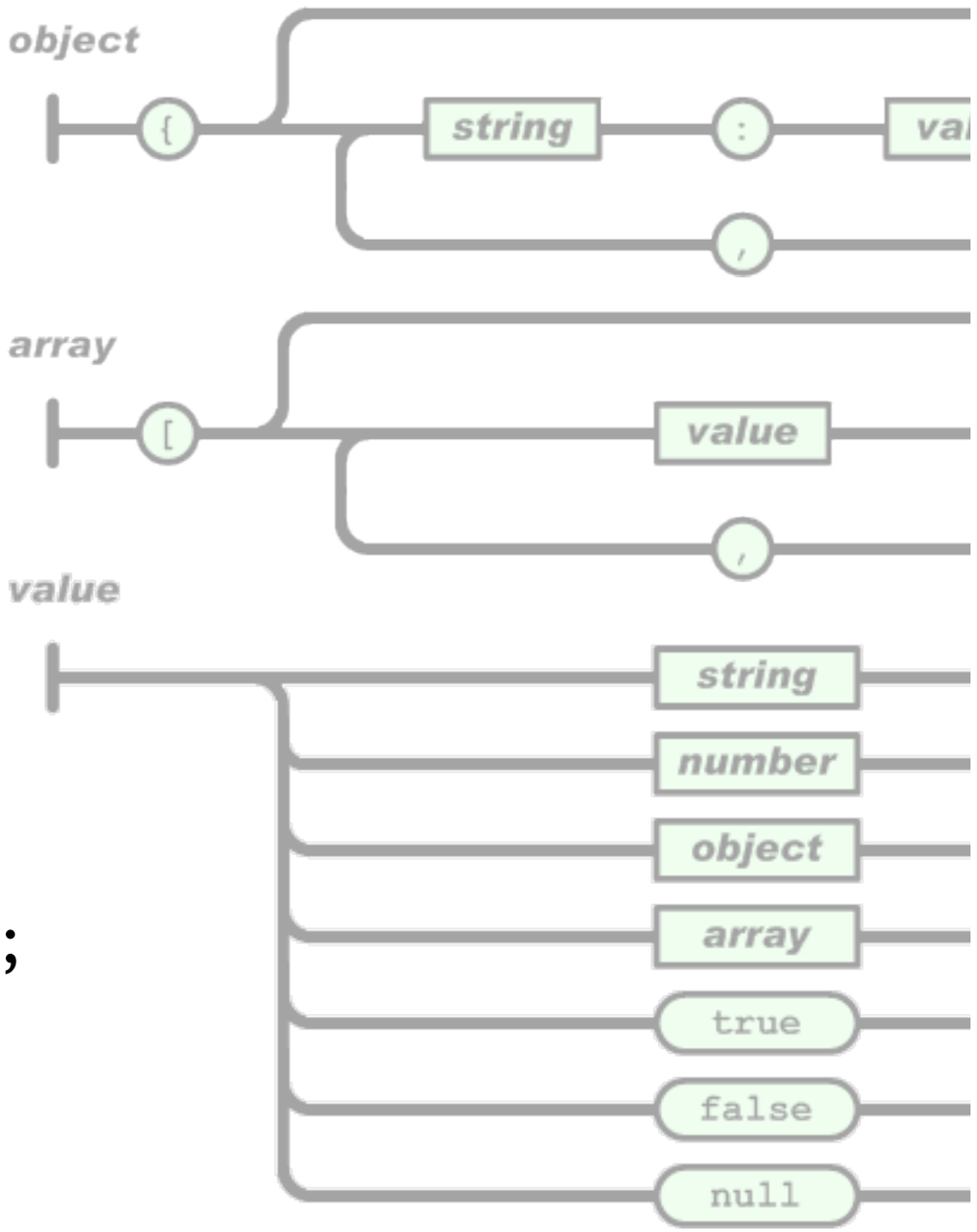
```
namespace json {

typedef
  std::map<std::string, value>
object;

typedef std::vector<value> array;

typedef std::string          string;
typedef long double          number;
typedef bool                 boolean;
typedef struct null {}       null;

}
```



boostpro computing

Representation

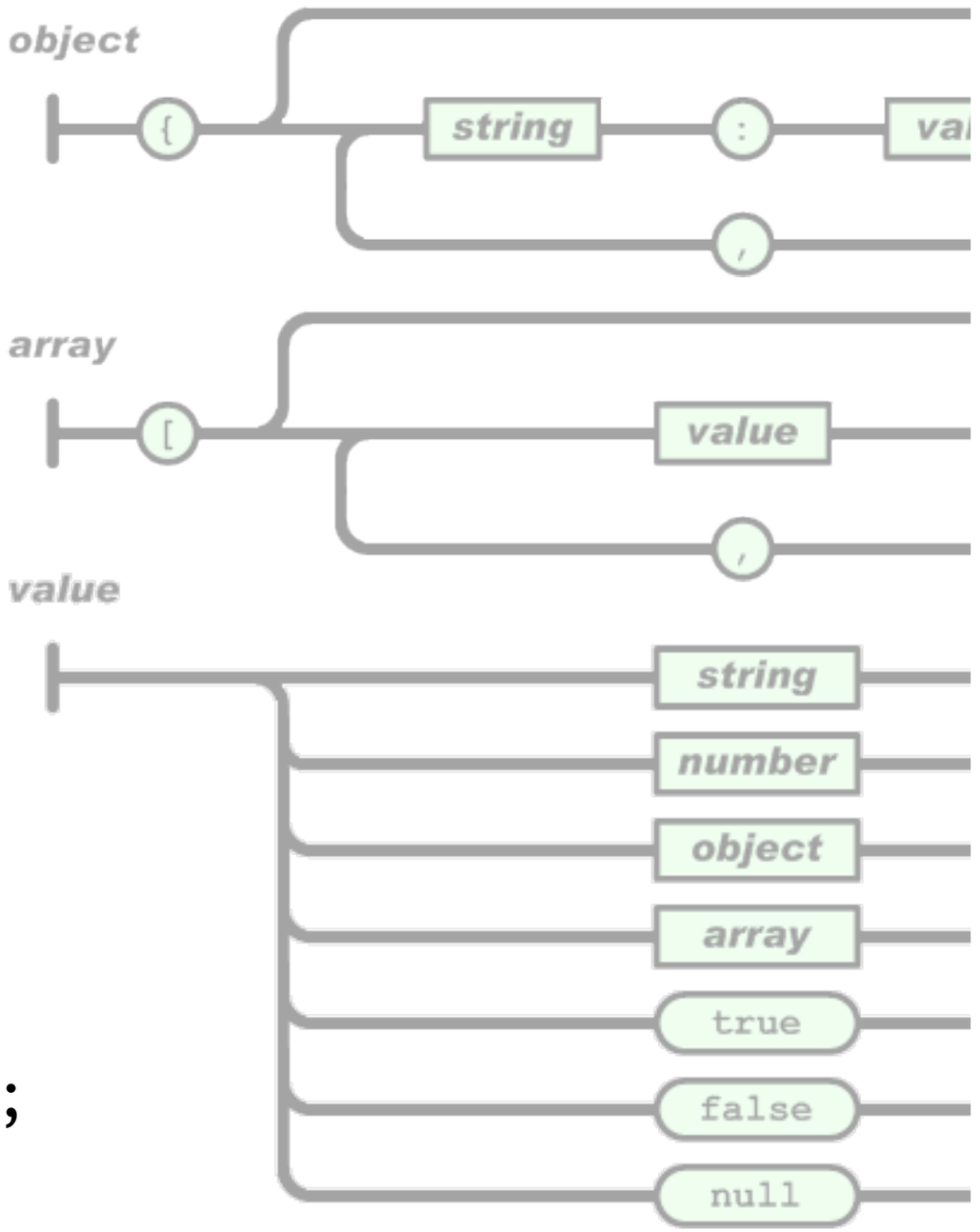
```
namespace json {

typedef
  std::map<std::string, value>
object;

typedef std::vector<value> array;

typedef std::string          string;
typedef long double          number;
typedef bool                 boolean;
typedef struct null {}       null;

}
```



boostpro computing

Representation

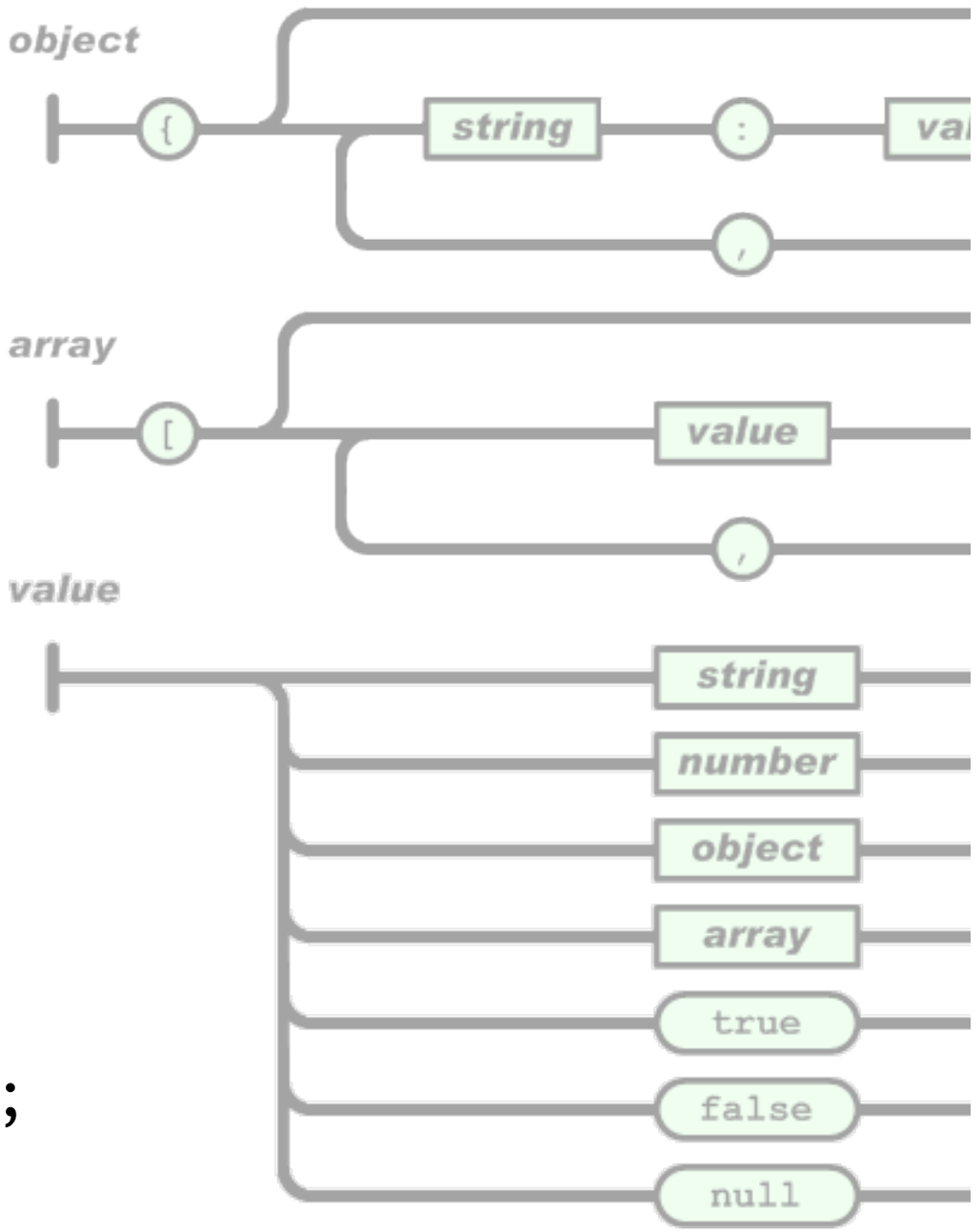
```
namespace json {



typedef
  std::map<std::string, value>
object;

typedef std::vector<value> array;

typedef std::string          string;
typedef long double          number;
typedef bool                 boolean;
typedef struct null {}       null;

}
```



boostpro computing

Representation

```
namespace json {


typedef
  std::map<std::string, value>
object;

typedef std::vector<value> array;

typedef std::string          string;
typedef long double          number;
typedef bool                 boolean;
typedef struct null {}       null;
}
```

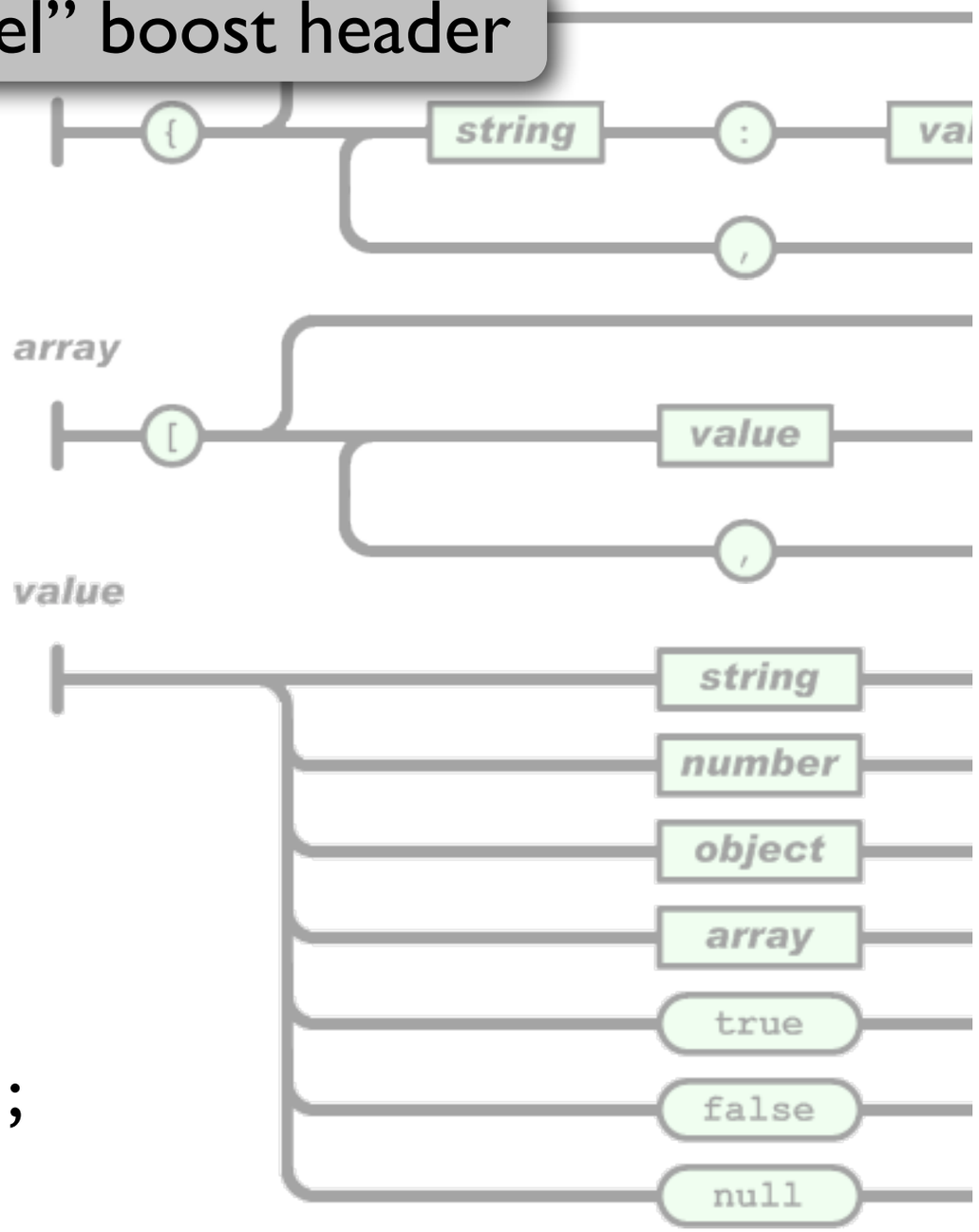


boostpro computing

Representation

"Top-level" boost header

```
#include <boost/any.hpp>
namespace json {



typedef
  std::map<std::string, value>
object;

typedef std::vector<value> array;

typedef std::string          string;
typedef long double          number;
typedef bool                 boolean;
typedef struct null {}       null;
}
```

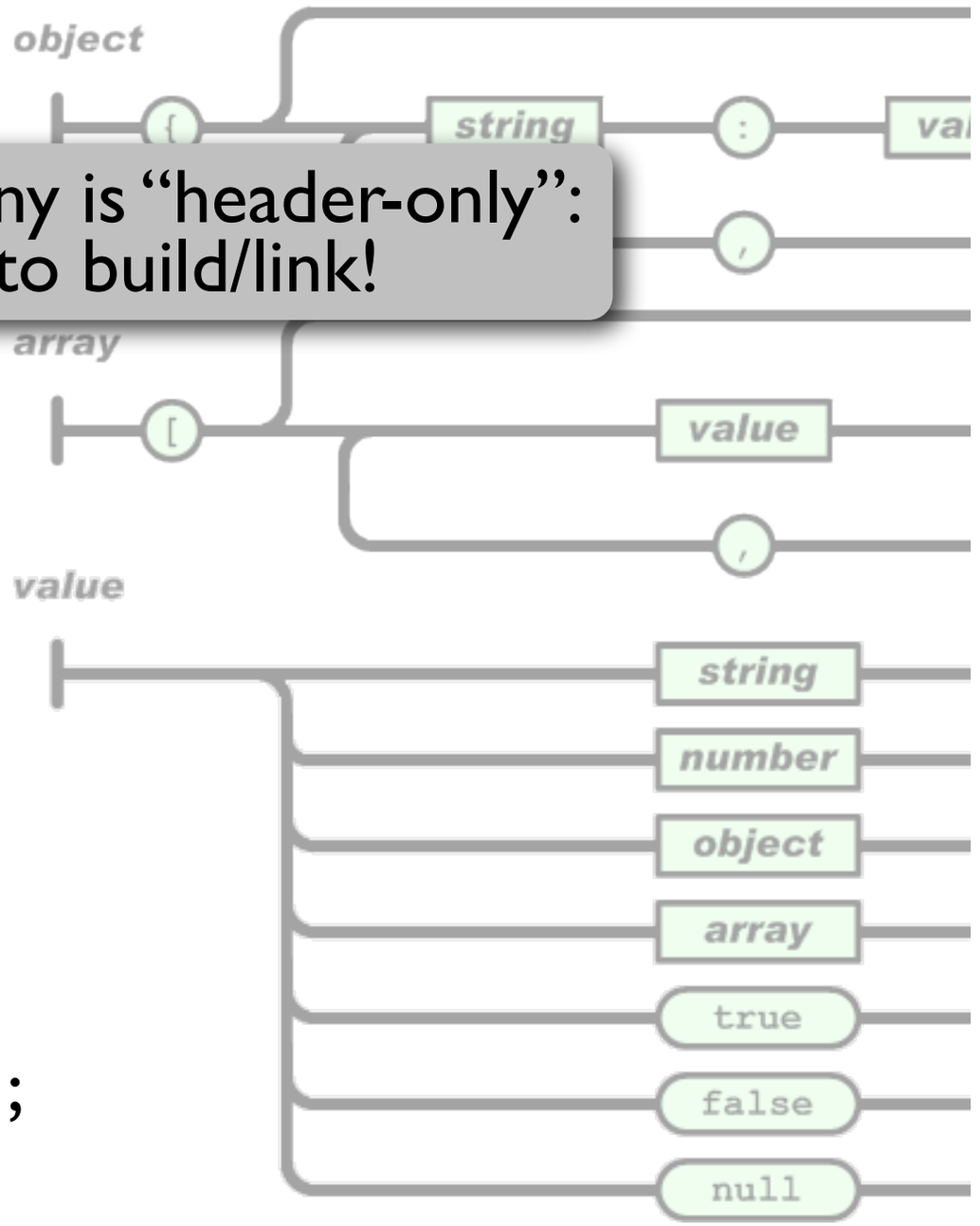


boostpro computing

Representation

```
#include <boost/any.hpp>
namespace json {



typedef
  std::map<std::string, value>
object;

typedef std::vector<value> array;

typedef std::string          string;
typedef long double          number;
typedef bool                 boolean;
typedef struct null {}       null;
}
```

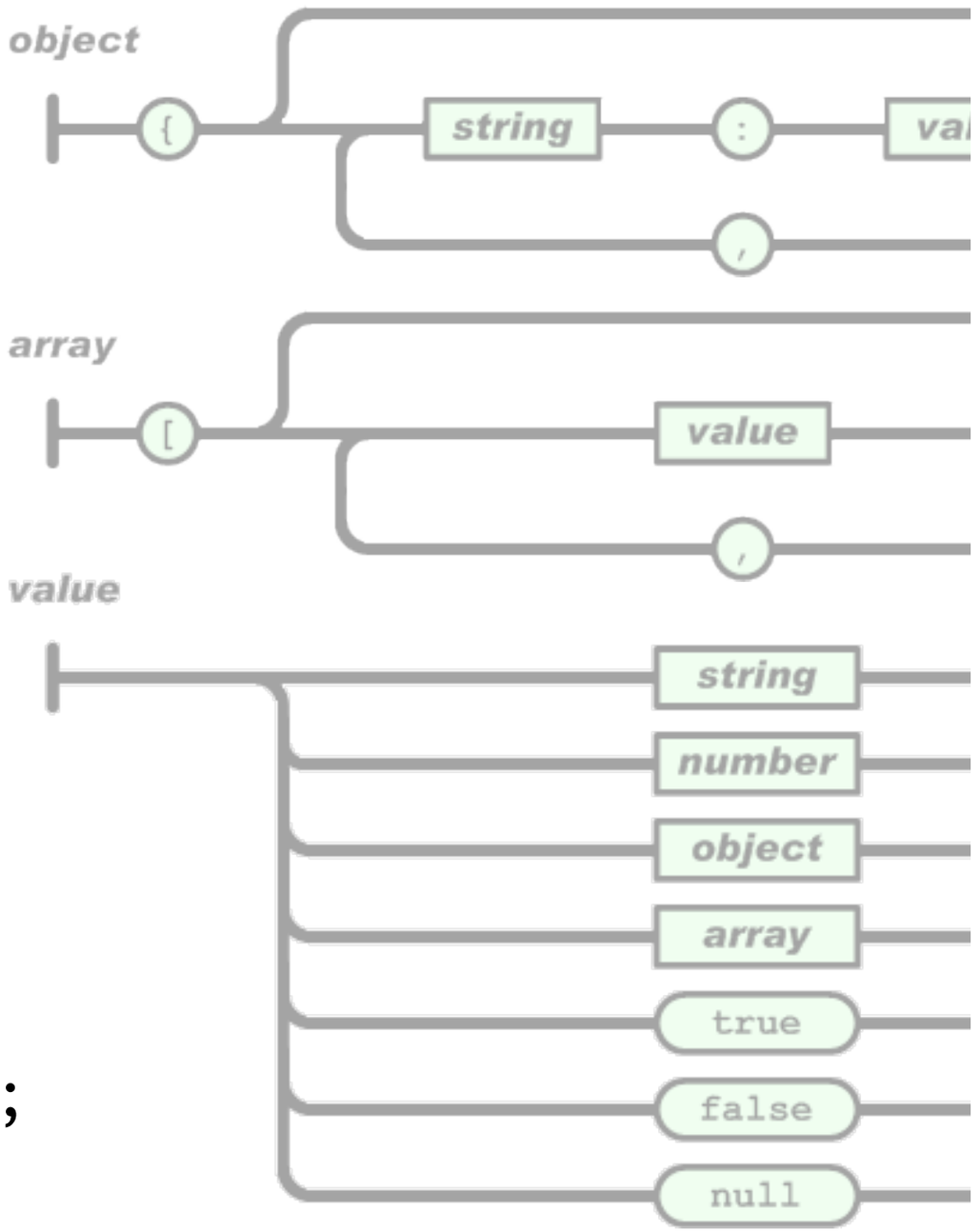
Boost.Any is “header-only”: nothing to build/link!



boostpro computing

Representation

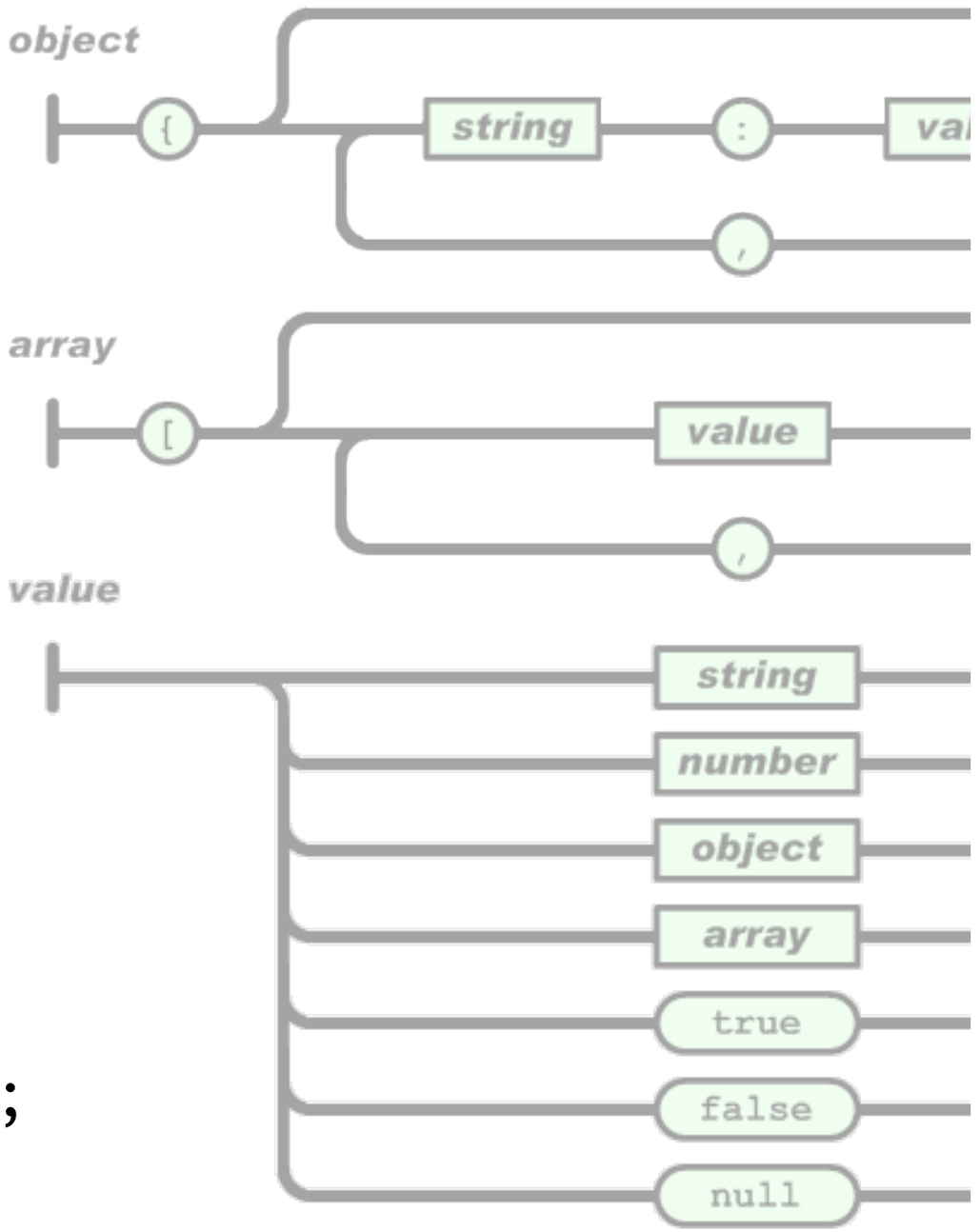
```
#include <boost/any.hpp>
namespace json {

typedef
  std::map<std::string, value>
object;

typedef std::vector<value> array;

typedef std::string          string;
typedef long double          number;
typedef bool                 boolean;
typedef struct null {}       null;
}
```



boostpro computing

Representation

```
#include <boost/any.hpp>
namespace json {

typedef boost::any value;

typedef
  std::map<std::string, value>
object;

typedef std::vector<value> array;

typedef std::string          string;
typedef long double          number;
typedef bool                 boolean;
typedef struct null {}       null;
}
```



Store Any Value…

Store Any Value…

```
#include <boost/any.hpp>
#include <vector>
#include <string>

int main()
{
    using boost::any;
    any x = 3;
    any y = 3.14f;
    any z = std::vector<int>(10, 42);
    z = x;
    z = "Hello, World!"
}
```

Store Any Value…

```
#include <boost/any.hpp>
#include <vector>
#include <string>

int main()
{
    using boost::any;
    any x = 3;
    any y = 3.14f;
    any z = std::vector<int>(10, 42);
    z = x;
    z = "Hello, World!"
}
```

x stores an int: 3

Store Any Value…

```
#include <boost/any.hpp>
#include <vector>
#include <string>

int main()
{
    using boost::any;
    any x = 3;
    any y = 3.14f;
    any z = std::vector<int>(10, 42);
    z = x;
    z = "Hello, World!"
}
```

y stores a float

Store Any Value…

```
#include <boost/any.hpp>
#include <vector>
#include <string>

int main()
{
    using boost::any;
    any x = 3;
    any y = 3.14f;
    any z = std::vector<int>(10, 42);
    z = x;
    z = "Hello, World!"
}
```

z stores a vector

Store Any Value…

```
#include <boost/any.hpp>
#include <vector>
#include <string>

int main()
{
    using boost::any;
    any x = 3;
    any y = 3.14f;
    any z = std::vector<int>(10, 42);
    z = x;
    z = "Hello, World!"
}
```

now z stores 3

boostpro computing

Store Any Value…

```
#include <boost/any.hpp>
#include <vector>
#include <string>

int main()
{
    using boost::any;
    any x = 3;
    any y = 3.14f;
    any z = std::vector<int>(10, 42);
    z = x;
    z = "Hello, World!"
}
```


boostpro computing

Store Any Value…

```
#include <boost/any.hpp>
#include <vector>
#include <string>

int main()
{
    using boost::any;
    any x = 3;
    any y = 3.14f;
    any z = std::vector<int>(10, 42);
    z = x;
    z = "Hello, World!"
}
```

Error; can't copy
`char const[14]`

boostpro computing

Store Any Value…

```
#include <boost/any.hpp>
#include <vector>
#include <string>

int main()
{
    using boost::any;
    any x = 3;
    any y = 3.14f;
    any z = std::vector<int>(10, 42);
    z = x;
    z = "Hello, World!"
}
```


boostpro computing

Store Any Value…

```
#include <boost/any.hpp>
#include <vector>
#include <string>

int main()
{
    using boost::any;
    any x = 3;
    any y = 3.14f;
    any z = std::vector<int>(10, 42);
    z = x;
    z = std::string("Hello, World!");
}
```

Store Any Value…

```
#include <boost/any.hpp>
#include <vector>
#include <string>

int main()
{
    using boost::any;
    any x = 3;
    any y = 3.14f;
    any z = std::vector<int>(10, 42);
    z = x;
    z = std::string("Hello, World!");
}
```

OK: `std::string` is a value type

boostpro computing

Store Any Value…

```
#include <boost/any.hpp>
#include <vector>
#include <string>

int main()
{
    using boost::any;
    any x = 3;
    any y = 3.14f;
    any z = std::vector<int>(10, 42);
    z = x;
    z = std::string("Hello, World!");
}
```

...Retrieve Any Value

```
#include <boost/any.hpp>
#include <string>
#include <iostream>
using std::string; using boost::any; using boost::any_cast;

int main()
{
    any s1 = string("pumpkin"),
        s2 = s1;

    if ( string* p = any_cast<string>(&s1) ) {
       p->replace(0, 4, "Rumplestilts");
    }
    std::cout << any_cast<string>(s1) << " turned into a "
      << any_cast<string>(s2) << std::endl;
}
```

…Retrieve Any Value

```
#include <boost/any.hpp>
#include <string>
#include <iostream>
using std::string; using boost::any; using boost::any_cast;

int main()
{
    any s1 = string("pumpkin"),
        s2 = s1;

    if ( string* p = any_cast<string>(&s1) ) {
       p->replace(0, 4, "Rumplestilts");
    }
    std::cout << any_cast<string>(s1) << " turned into a "
      << any_cast<string>(s2) << std::endl;
}
```

s1 and s2: both "pumpkin"

…Retrieve Any Value

```
#include <boost/any.hpp>
#include <string>
#include <iostream>
using std::string; using boost::any; using boost::any_cast;

int main()
{
    any s1 = string("pumpkin"),
        s2 = s1;

    if ( string* p = any_cast<string>(&s1) ) {
       p->replace(0, 4, "Rumplestilts");
    }
    std::cout << any_cast<string>(s1) << " turned into a "
      << any_cast<string>(s2) << std::endl;
}
```

does s1 contain a string?

…Retrieve Any Value

```
#include <boost/any.hpp>
#include <string>
#include <iostream>
using std::string; using boost::any; using boost::any_cast;

int main()
{
    any s1 = string("pumpkin"),
        s2 = s1;

    if ( string* p = any_cast<string>(&s1) ) {
       p->replace(0, 4, "Rumplestilts");
    }
    std::cout << any_cast<string>(s1) << " turned into a "
      << any_cast<string>(s2) << std::endl;
}
```

pointer in, pointer out

...Retrieve Any Value

```
#include <boost/any.hpp>
#include <string>
#include <iostream>
using std::string; using boost::any; using boost::any_cast;

int main()
{
    any s1 = string("pumpkin"),
        s2 = s1;

    if ( string* p = any_cast<string>(&s1) ) {
       p->replace(0, 4, "Rumplestilts");
    }
    std::cout << any_cast<string>(s1) << " turned into a "
      << any_cast<string>(s2) << std::endl;
}
```

...Retrieve Any Value

```
#include <boost/any.hpp>
#include <string>
#include <iostream>
using std::string; using boost::any; using boost::any_cast;

int main()
{
    any s1 = string("pumpkin"),
        s2 = s1;

    if ( string* p = any_cast<string>(&s1) ) {
       p->replace(0, 4, "Rumplestilts");
    }
    std::cout << any_cast<string>(s1) << " turned into a "
      << any_cast<string>(s2) << std::endl;
}
```

…Retrieve Any Value

```
#include <boost/any.hpp>
#include <string>
#include <iostream>
using std::string; using boost::any; using boost::any_cast;

int main()
{
    any s1 = string("pumpkin"),
        s2 = s1;

    if ( string* p = any_cast<string>(&s1) ) {
       p->replace(0, 4, "Rumplestilts");
    }
    std::cout << any_cast<string>(s1) << " turned into a "
      << any_cast<string>(s2) << std::endl;
}
```

reference in, reference out; throws `bad_any_cast` on failure

...Retrieve Any Value

```
#include <boost/any.hpp>
#include <string>
#include <iostream>
using std::string; using boost::any; using boost::any_cast;

int main()
{
    any s1 = string("pumpkin"),
        s2 = s1;

    if ( string* p = any_cast<string>(&s1) ) {
       p->replace(0, 4, "Rumplestilts");
    }
    std::cout << any_cast<string>(s1) << " turned into a "
      << any_cast<string>(s2) << std::endl;
}
```

. . . Retrieve Any Value

```
$ c++ -I /path/to/boost pumpkin.cpp -o test
$
```

```
#in
#in
#in
usi                                                    st;

int main()
{
    any s1 = string("pumpkin"),
        s2 = s1;

    if ( string* p = any_cast<string>(&s1) ) {
       p->replace(0, 4, "Rumplestilts");
    }
    std::cout << any_cast<string>(s1) << " turned into a "
      << any_cast<string>(s2) << std::endl;
}
```


boostpro computing

...Retrieve Any Value

```
$ c++ -I /path/to/boost pumpkin.cpp -o test
$ ./test
Rumplestiltskin turned into a pumpkin
$
```

```
#in
#in
#in
usi                                                    st;

int main()
{
    any s1 = string("pumpkin"),
        s2 = s1;

    if ( string* p = any_cast<string>(&s1) ) {
       p->replace(0, 4, "Rumplestilts");
    }
    std::cout << any_cast<string>(s1) << " turned into a "
      << any_cast<string>(s2) << std::endl;
}
```

A *Value*, not a Pointer

```
$ c++ -I /path/to/boost pumpkin.cpp -o test
$ ./test
Rumplestiltskin turned into a pumpkin
$
```

```
#in
#in
#in
usi                                                    st;

int main()
{
    any s1 = string("pumpkin"),
        s2 = s1;

    if ( string* p = any_cast<string>(&s1) ) {
       p->replace(0, 4, "Rumplestilts");
    }
    std::cout << any_cast<string>(s1) << " turned into a "
      << any_cast<string>(s2) << std::endl;
}
```


boostpro computing

A *Value*, not a Pointer

```
$ c++ -I /path/to/boost pumpkin.cpp -o test
$ ./test
Rumplestiltskin turned into a pumpkin
$
```

s1 and s2 are distinct objects

```
#in
#in
#in
usi                                                    st;

int main()
{
    any s1 = string("pumpkin"),
        s2 = s1;

    if ( string* p = any_cast<string>(&s1) ) {
       p->replace(0, 4, "Rumplestilts");
    }
    std::cout << any_cast<string>(s1) << " turned into a "
      << any_cast<string>(s2) << std::endl;
}
```

A *Value*, not a Pointer

```
$ c++ -I /path/to/boost pumpkin.cpp -o test
$ ./test
Rumplestiltskin turned into a pumpkin
$
```

```
#in
#in
#in
usi                                                    st;

int main()
{
    any s1 = string("pumpkin"),
        s2 = s1;

    if ( string* p = any_cast<string>(&s1) ) {
       p->replace(0, 4, "Rumplestilts");
    }
    std::cout << any_cast<string>(s1) << " turned into a "
      << any_cast<string>(s2) << std::endl;
}
```


boostpro computing

Creating Value

```
{
     "first": "Dave",
     "last": "Abrahams",
     "age": 48.2,
     "sex": "M",
     "zip code": "02143",
     "registered": true,
     "catchphrase": null
     "interests": [
       "Jamming",
       "Biking",
       "Hacking"
    ]
}
```

Creating Value

```
#include "json.hpp"

value dave()
{
    using namespace json;
    object me;
    me["first"] = string("Dave");
    me["last"] = string("Abrahams");
    me["age"] = 48.2;
    me["sex"] = string("M");
    me["zip code"] = string("02143");
    me["registered"] = true;
    me["catchphrase"] = null();
    array my_interests;
    my_interests.push_back(string("Jamming"));
    my_interests.push_back(string("Biking"));
    my_interests.push_back(string("Hacking"));
    me["interests"] = my_interests;
    return me;
}
```

```
{
    "first": "Dave",
    "last": "Abrahams",
    "age": 48.2,
    "sex": "M",
    "zip code": "02143",
    "registered": true,
    "catchphrase": null
    "interests": [
      "Jamming",
      "Biking",
      "Hacking"
   ]
}
```


boostpro computing

Highlights

- C++, STL, and much of Boost run on value semantics
- Boost.Any wraps values, exposing value semantics and erasing type information
- Just one of 100+ *header-only libraries* you can use without building or linking library binaries
- Find docs for library *xyzzy* at http://boost.org/libs/*xyzzy*
- Top-level headers: `#include <boost/`*xyzzy*`.hpp>`
- Your include path (`-I` *whatever*) should *contain a directory that contains a directory* named `boost/`

boostpro computing

Exercise

- http://github.com/boostpro/bbn-2012-11/tree/master/exercises/any-json
- Create json.hpp using Boost.Any for values (hint: slide 25)
- Use it to compile the *dave()* function supplied
- Write `operator<<(std::ostream&, json::value const&)` using a chain of `any_cast` tests to discover the stored type
- Don't worry about formatting, escaping quotes, or unicode
- Call your print function on the result of *dave()*
- BONUS: Find and fix the bug in *dave()*

The Bug

```
#include "json.hpp"

value dave()
{
    using namespace json;
    object me;
    me["first"] = string("Dave");
    me["last"] = string("Abrahams");
    me["age"] = 48.2;
    me["sex"] = string("M");
    me["zip code"] = string("02143");
    me["registered"] = true;
    me["catchphrase"] = null();
    array my_interests;
    my_interests.push_back(string("Jamming"));
    my_interests.push_back(string("Biking"));
    my_interests.push_back(string("Hacking"));
    me["interests"] = my_interests;
    return me;
}
```

```
{
    "first": "Dave",
    "last": "Abrahams",
    "age": 48.2,
    "sex": "M",
    "zip code": "02143",
    "registered": true,
    "catchphrase": null
    "interests": [
      "Jamming",
      "Biking",
      "Hacking"
   ]
}
```


boostpro computing

The Bug

stores a `double`, not a `json::number` (a.k.a. `long double`)

```
#include "json.hpp"

value dave()
{
    using namespace json;
    object me;
    me["first"] = string("Dave");
    me["last"] = string("Abrahams");
    me["age"] = 48.2;
    me["sex"] = string("M");
    me["zip code"] = string("02143");
    me["registered"] = true;
    me["catchphrase"] = null();
    array my_interests;
    my_interests.push_back(string("Jamming"));
    my_interests.push_back(string("Biking"));
    my_interests.push_back(string("Hacking"));
    me["interests"] = my_interests;
    return me;
}
```

```
{
    "first": "Dave",
    "last": "Abrahams",
    "age": 48.2,
    "sex": "M",
    "zip code": "02143",
    "registered": true,
    "catchphrase": null
    "interests": [
      "Jamming",
      "Biking",
      "Hacking"
   ]
}
```


boostpro computing

The Bug

```
#include "json.hpp"

value dave()
{
    using namespace json;
    object me;
    me["first"] = string("Dave");
    me["last"] = string("Abrahams");
    me["age"] = 48.2;
    me["sex"] = string("M");
    me["zip code"] = string("02143");
    me["registered"] = true;
    me["catchphrase"] = null();
    array my_interests;
    my_interests.push_back(string("Jamming"));
    my_interests.push_back(string("Biking"));
    my_interests.push_back(string("Hacking"));
    me["interests"] = my_interests;
    return me;
}
```

```
{
    "first": "Dave",
    "last": "Abrahams",
    "age": 48.2,
    "sex": "M",
    "zip code": "02143",
    "registered": true,
    "catchphrase": null
    "interests": [
      "Jamming",
      "Biking",
      "Hacking"
   ]
}
```


boostpro computing

The Fix

```
#include "json.hpp"

value dave()
{
    using namespace json;
    object me;
    me["first"] = string("Dave");
    me["last"] = string("Abrahams");
    me["age"] = number(48.2);
    me["sex"] = string("M");
    me["zip code"] = string("02143");
    me["registered"] = true;
    me["catchphrase"] = null();
    array my_interests;
    my_interests.push_back(string("Jamming"));
    my_interests.push_back(string("Biking"));
    my_interests.push_back(string("Hacking"));
    me["interests"] = my_interests;
    return me;
}
```

```
{
    "first": "Dave",
    "last": "Abrahams",
    "age": 48.2,
    "sex": "M",
    "zip code": "02143",
    "registered": true,
    "catchphrase": null
    "interests": [
      "Jamming",
      "Biking",
      "Hacking"
   ]
}
```


boostpro computing

The Fix

```
#include "json.hpp"

value dave()
{
    using namespace json;
    object me;
    me["first"] = string("Dave");
    me["last"] = string("Abrahams");
    me["age"] = number(48.2);
    me["sex"] = string("M");
    me["zip code"] = string("02143");
    me["registered"] = true;
    me["catchphrase"] = null();
    array my_interests;
    my_interests.push_back(string("Jamming"));
    my_interests.push_back(string("Biking"));
    my_interests.push_back(string("Hacking"));
    me["interests"] = my_interests;
    return me;
}
```

OK

```
{
    "first": "Dave",
    "last": "Abrahams",
    "age": 48.2,
    "sex": "M",
    "zip code": "02143",
    "registered": true,
    "catchphrase": null
    "interests": [
      "Jamming",
      "Biking",
      "Hacking"
   ]
}
```


boostpro computing

The Fix

```
#include "json.hpp"

value dave()
{
    using namespace json;
    object me;
    me["first"] = string("Dave");
    me["last"] = string("Abrahams");
    me["age"] = number(48.2);
    me["sex"] = string("M");
    me["zip code"] = string("02143");
    me["registered"] = true;
    me["catchphrase"] = null();
    array my_interests;
    my_interests.push_back(string("Jamming"));
    my_interests.push_back(string("Biking"));
    my_interests.push_back(string("Hacking"));
    me["interests"] = my_interests;
    return me;
}
```

OK

but casting is getting painful…

```
{
    "first": "Dave",
    "last": "Abrahams",
    "age": 48.2,
    "sex": "M",
    "zip code": "02143",
    "registered": true,
    "catchphrase": null
    "interests": [
      "Jamming",
      "Biking",
      "Hacking"
   ]
}
```


boostpro computing

What We Want

```
#include "json.hpp"

value dave()
{
    using namespace json;
    object me;
    me["first"] = "Dave";
    me["last"] = "Abrahams";
    me["age"] = 48.2;
    me["sex"] = "M";
    me["zip code"] = "02143";
    me["registered"] = true;
    me["catchphrase"] = null();
    array my_interests;
    my_interests.push_back("Jamming");
    my_interests.push_back("Biking");
    my_interests.push_back("Hacking");
    me["interests"] = my_interests;
    return me;
}
```

```
{
    "first": "Dave",
    "last": "Abrahams",
    "age": 48.2,
    "sex": "M",
    "zip code": "02143",
    "registered": true,
    "catchphrase": null
    "interests": [
      "Jamming",
      "Biking",
      "Hacking"
   ]
}
```


boostpro computing

Asserting Control

```
namespace json {

typedef boost::any                          value;
typedef std::map<std::string, value>        object;
typedef std::vector<value>                  array;
typedef std::string                         string;
typedef long double                         number;
typedef bool                                boolean;
typedef struct null {}                      null;
```


boostpro computing

Asserting Control

```
namespace json {

typedef boost::any                          value;
typedef std::map<std::string, value>        object;
typedef std::vector<value>                  array;
typedef std::string                         string;
typedef long double                         number;
typedef bool                                boolean;
typedef struct null {}                      null;
```


boostpro computing

Asserting Control

```
namespace json {

struct                                  value;
typedef std::map<std::string, value>    object;
typedef std::vector<value>              array;
typedef std::string                     string;
typedef long double                     number;
typedef bool                            boolean;
typedef struct null {}                  null;

struct value
{
    value(object const& x);
    value(array const& x);
    ...
```

Asserting Control

```
namespace json {

struct                                  value;
typedef std::map<std::string, value>    object;
typedef std::vector<value>              array;
typedef std::string                     string;
typedef long double                     number;
typedef bool                            boolean;
typedef struct null {}                  null;

struct value
{
    value(object const& x);
    value(array const& x);
    ...
```


boostpro computing

Asserting Control

```
namespace json {

struct                                  value;
typedef std::map<std::string, value>    object;
typedef std::vector<value>              array;
typedef std::string                     string;
typedef long double                     number;
typedef bool                            boolean;
typedef struct null {}                  null;

struct value
{
    value(object const& x);
    value(array const& x);
    ...
```


boostpro computing

Asserting Control

```
namespace json {

struct                                  value;
typedef std::map<std::string, value>    object;
typedef std::vector<value>              array;
typedef std::string                     string;
typedef long double                     number;
typedef bool                            boolean;
typedef struct null {}                  null;

struct value
{
    value(object const& x);
    value(array const& x);
    ...
```


boostpro computing

Asserting Control

```
struct value
{
    value(object const& x)  : stored(x) {}
    value(array const& x)   : stored(x) {}
    value(string const& x)  : stored(x) {}
    value(number const& x)  : stored(x) {}
    value(boolean const& x) : stored(x) {}
    value(null const& x)    : stored(x) {}

    friend std::ostream& operator<<(
        std::ostream& s, value const& x);
 private:
    boost::any stored;
};
```

Compile It!

```
$ clang++ -I ~/src/boost-1.51 test.cpp
```


boostpro computing

Compile It!

```
$ clang++ -I ~/src/boost-1.51 test.cpp
test.cpp:15:17: error: conversion from 'double' to 'const
json::value' is ambiguous
    me["age"] = 48.2;                   //   "age": 48.2,
                ^~~~
./json.hpp:33:5: note: candidate constructor
    value(number const& x) : stored(x) {}
    ^
./json.hpp:34:5: note: candidate constructor
    value(boolean const& x) : stored(x) {}
    ^
$
```


boostpro computing

Asserting Control

```
struct value
{
    value(object const& x)  : stored(x) {}
    value(array const& x)   : stored(x) {}
    value(string const& x)  : stored(x) {}
    value(number const& x)  : stored(x) {}
    value(boolean const& x) : stored(x) {}
    value(null const& x)    : stored(x) {}

    friend std::ostream& operator<<(
        std::ostream& s, value const& x);
 private:
    boost::any stored;
};
```

Asserting Control

```
struct value
{
    value(object const& x)  : stored(x) {}
    value(array const& x)   : stored(x) {}
    value(string const& x)  : stored(x) {}
    value(number const& x)  : stored(x) {}
    value(boolean const& x) : stored(x) {}
    value(null const& x)    : stored(x) {}

    friend std::ostream& operator<<(
        std::ostream& s, value const& x);
 private:
    boost::any stored;
};
```

Asserting Control

```
struct value
{
    value(object const& x)  : stored(x) {}
    value(array const& x)   : stored(x) {}
    value(string const& x)  : stored(x) {}

    value(       const& x)  : stored(x) {}
    value(boolean const& x) : stored(x) {}
    value(null const& x)    : stored(x) {}

    friend std::ostream& operator<<(
        std::ostream& s, value const& x);
 private:
    boost::any stored;
};
```


boostpro computing

Asserting Control

```
struct value
{
    value(object const& x)  : stored(x) {}
    value(array const& x)   : stored(x) {}
    value(string const& x)  : stored(x) {}
    template <class Number>
    value(Number const& x)  : stored(x) {}
    value(boolean const& x) : stored(x) {}
    value(null const& x)    : stored(x) {}

    friend std::ostream& operator<<(
        std::ostream& s, value const& x);
 private:
    boost::any stored;
};
```


boostpro computing

Compile It!

```
$ clang++ -I ~/src/boost-1.51 test.cpp
```


boostpro computing

Compile It!

```
$ clang++ -I ~/src/boost-1.51 test.cpp
In file included from test.cpp:4:
./json.hpp:39:16: error: functional-style cast from 'const char *' to
'number' (aka 'long double')
      is not allowed
    ) : stored(number(x)) {}
                ^~~~~~~~
test.cpp:13:19: note: in instantiation of function template
specialization
      'json::value::value<char [5]>' requested here
    me["first"] = "Dave";                 //   "first": "Dave",
                  ^
In file included from test.cpp:4:
./json.hpp:39:16: error: functional-style cast from 'const char *' to
'number' (aka 'long double')
      is not allowed
```

Asserting Control

```
struct value
{
    value(object const& x)  : stored(x) {}
    value(array const& x)   : stored(x) {}
    value(string const& x)  : stored(x) {}
    template <class Number>
    value(Number const& x)  : stored(x) {}
    value(boolean const& x) : stored(x) {}
    value(null const& x)    : stored(x) {}

    friend std::ostream& operator<<(
        std::ostream& s, value const& x);
 private:
    boost::any stored;
};
```


boostpro computing

Asserting Control

```
struct value
{
    value(object const& x)  : stored(x) {}
    value(array const& x)   : s
    value(string const& x)  :
    template <class Number>
    value(Number const& x)  : stored(x) {}
    value(boolean const& x) : stored(x) {}
    value(null const& x)    : stored(x) {}

    friend std::ostream& operator<<(
        std::ostream& s, value const& x);
 private:
    boost::any stored;
};
```

Had relied on this one for conversion from "..."

boostpro computing

Asserting Control

```
struct value
{
    value(object const& x)  : stored(x) {}
    value(array const& x)   : stored(x) {}
    value(string const& x)  : stored(x) {}
    template <class Number>
    value(Number const& x)  : s
    value(boolean const& x) : s
    value(null const& x)    : s

    friend std::ostream& operator<<(
        std::ostream& s, value const& x);
 private:
    boost::any stored;
};
```

but *now*, Number is deduced as char[5] — a perfect match!

boostpro computing

Asserting Control

```
struct value
{
    value(object const& x)  : stored(x) {}
    value(array const& x)   : stored(x) {}
    value(string const& x)  : stored(x) {}
    template <class Number>
    value(Number const& x)  : s
    value(boolean const& x) : s
    value(null const& x)    : s

    friend s
        std:
private:
    boost::any stored;
};
```

but *now*, `Number` is deduced as `char[5]` — a perfect match!

Want to keep it from being chosen unless `Number` is really a number

boostpro computing

Asserting Control

```
struct value
{
    value(object const& x)  : stored(x) {}
    value(array const& x)   : stored(x) {}
    value(string const& x)  : stored(x) {}
    template <class Number>
    value(Number const& x)  : stored(x) {}
    value(boolean const& x) : stored(x) {}
    value(null const& x)    : stored(x) {}

    friend s
        std:
 private:
    boost::any stored;
};
```

Want to keep it from being chosen unless Number is really a number

boostpro computing

Two Tools

- Boost.TypeTraits—Compile-Time Type Info (CTTI)

```
#include <boost/type_traits/is_arithmetic.hpp>

bool const x = is_arithmetic<Number>::value;
typedef        is_arithmetic<Number>::type x_t;
bool const xprime =                  x_t::value;
```

- Boost.EnableIf—Enabling/disabling templates

```
#include <boost/utility/enable_if.hpp>

enable_if<is_arithmetic<T>, U>::type
```

Two Tools

"*Granular header*"

- Boost.TypeTraits—Compile-Time Type Info (CTTI)

```
#include <boost/type_traits/is_arithmetic.hpp>

bool const x = is_arithmetic<Number>::value;
typedef        is_arithmetic<Number>::type x_t;
bool const xprime =                  x_t::value;
```

- Boost.EnableIf—Enabling/disabling templates

```
#include <boost/utility/enable_if.hpp>

enable_if<is_arithmetic<T>, U>::type
```

boostpro computing

Two Tools

- Boost.TypeTraits—Compile-Time Type Info (CTTI)

```
#include <boost/type_traits/is_arithmetic.hpp>

bool const x = is_arithmetic<Number>::value;
typedef      is_arithmetic<Number>::type x_t;
bool const xprime =                      x_t::value;
```

compile-time constant

- Boost.EnableIf—Enabling/disabling templates

```
#include <boost/utility/enable_if.hpp>

enable_if<is_arithmetic<T>, U>::type
```

boostpro computing

Two Tools

- Boost.TypeTraits—Compile-Time Type Info (CTTI)

```
#include <boost/type_traits/is_arithmetic.hpp>

bool const x = is_arithmetic<Number>::value;
typedef      is_arithmetic<Number>::type x_t;
bool const xprime =                      x_t::value;
```

true_type *or* false_type

- Boost.EnableIf—Enabling/disabling templates

```
#include <boost/utility/enable_if.hpp>

enable_if<is_arithmetic<T>, U>::type
```

boostpro computing

Two Tools

- Boost.TypeTraits—Compile-Time Type Info (CTTI)

```
#include <boost/type_traits/is_arithmetic.hpp>

bool const x = is_arithmetic<Number>::value;
typedef      is_arithmetic<Number>::type x_t;
bool const xprime =                      x_t::value;
```

same as x

- Boost.EnableIf—Enabling/disabling templates

```
#include <boost/utility/enable_if.hpp>

enable_if<is_arithmetic<T>, U>::type
```

Two Tools

- Boost.TypeTraits—Compile-Time Type Info (CTTI)

```
#include <boost/type_traits/is_arithmetic.hpp>

bool const x = is_arithmetic<Number>::value;
typedef        is_arithmetic<Number>::type x_t;
bool const xprime =                  x_t::value;
```

- Boost.EnableIf—Enabling/disabling templates

```
#include <boost/utility/enable_if.hpp>

enable_if<is_arithmetic<T>, U>::type
```

U if T is arithmetic, otherwise not a type

boostpro computing

Two Tools

- Boost.TypeTraits—Compile-Time Type Info (CTTI)

```
#include <boost/type_traits/is_arithmetic.hpp>

bool const x = is_arithmetic<Number>::value;
typedef        is_arithmetic<Number>::type x_t;
bool const xprime =                  x_t::value;
```

- Boost.EnableIf—Enabling/disabling templates

```
#include <boost/utility/enable_if.hpp>

enable_if<is_arithmetic<T> >::type
```

void if T is arithmetic, otherwise not a type

boostpro computing

Two Tools

- Boost.TypeTraits—Compile-Time Type Info (CTTI)

```
#include <boost/type_traits/is_arithmetic.hpp>

bool const x = is_arithmetic<Number>::value;
typedef        is_arithmetic<Number>::type x_t;
bool const xprime =                  x_t::value;
```

- Boost.EnableIf—Enabling/disabling templates

```
#include <boost/utility/enable_if.hpp>

enable_if<is_arithmetic<T> >::type
```

Aside: Dispatching with Traits

- Boolean-valued traits derive from true_type and false_type

```
template <class T>
struct is_pointer : false_type {};
template <class T>
struct is_pointer<T*> : true_type {};
```

- Use this fact to select overloads:

```
template <class T>
T f_impl(T x, true_type) { /* handle floating point */ }
template <class T>
T f_impl(T x, false_type) { /* handle other types */ }

T f(T x) { return f_impl(x, is_floating_point<T>()); }
```

Asserting Control

```
struct value
{
    value(object const& x)  : stored(x) {}
    value(array const& x)   : stored(x) {}
    value(string const& x)  : stored(x) {}
    template <class Number>
    value(Number const& x)  : stored(x) {}
    value(boolean const& x) : stored(x) {}
    value(null const& x)    : stored(x) {}

    friend std::ostream& operator<<(
        std::ostream& s, value const& x);
 private:
    boost::any stored;
};
```


boostpro computing

Asserting Control

```
struct value
{
    value(object const& x)  : stored(x) {}
    value(array const& x)   : stored(x) {}
    value(string const& x)  : stored(x) {}
    template <class Number>
    value(Number const& x)  : stored(x) {}
    value(boolean const& x) : stored(x) {}
    value(null const& x)    : stored(x) {}

    friend std::ostream& operator<<(
        std::ostream& s, value const& x);
 private:
    boost::any stored;
};
```


boostpro computing

Asserting Control

```
struct value
{
    value(object const& x)  : stored(x) {}
    value(array const& x)   : stored(x) {}
    value(string const& x)  : stored(x) {}
    template <class Number>
    value(Number const& x,
                            : stored(x) {}
    value(boolean const& x) : stored(x) {}
    value(null const& x)    : stored(x) {}

    friend std::ostream& operator<<(
```

Asserting Control

```
struct value
{
    value(object const& x)  : stored(x) {}
    value(array const& x)   : stored(x) {}
    value(string const& x)  : stored(x) {}
    template <class Number>
    value(Number const& x,
      typename boost::enable_if<
        boost::is_arithmetic<T> >::type* = 0)
                            : stored(x) {}
    value(boolean const& x) : stored(x) {}
    value(null const& x)    : stored(x) {}

    friend std::ostream& operator<<(
```

Compile It!

```
$ clang++ -I ~/src/boost-1.51 test.cpp -o tst && ./tst
```


boostpro computing

Compile It!

```
$ clang++ -I ~/src/boost-1.51 test.cpp -o tst && ./tst
{ "age": 48.2, "catchphrase": null, "first": true,
"interests": [ true, true, true ], "last": true,
"registered": true, "sex": true, "zip code": true }
$
```


boostpro computing

Exercise

- http://github.com/boostpro/bbn-2012-11/tree/master/exercises/wrapped-any-json
- Use the `is_convertible<T,U>` type trait to make a `json::value` constructor that catches types convertible to `json::string`
- Use it to fix the example so it yields:

```
{ "age": 48.2, "catchphrase": null, "first": "Dave", "interests":
[ "Reading", "Biking", "Hacking" ], "last": "Abrahams",
"registered": true, "sex": "M", "zip code": "02143" }
$
```

- BONUS: there solution file contains a way to get the desired result with NO_ENABLE_IF. What cases might this approach fail to handle?

Type Traits Foundations

```
struct true_type
{
    typedef true_type type;
    static bool const value = true;
};

struct false_type
{
    typedef false_type type;
    static bool const value = false;
};
```

Type Traits Foundations

```
struct true_type
{
    typedef true_type type;
    static bool const value = true;
};
```

T::type is just *T*

```
struct false_type
{
    typedef false_type type;
    static bool const value = false;
};
```


boostpro computing

Type Traits Foundations

```
struct true_type
{
    typedef true_type type;
    static bool const value = true;
};

struct false_type
{
    typedef false_type type;
    static bool const value = false;
};
```

T::value is the corresponding compile-time constant

boostpro computing

Type Traits Foundations

```
struct true_type
{
    typedef true_type type;
    static bool const value = true;
};

struct false_type
{
    typedef false_type type;
    static bool const value = false;
};
```

Container Printing

```
#include <ostream>

template <class Cont>
void print(std::ostream& os, Cont const& x)
{
    char const* prefix = "[ ";

    typedef typename Cont::const_iterator iter;
    for (iter p = x.begin(); p != x.end(); ++p)
    {
        os << prefix << *p;
        prefix = ", ";
    }
    os << " ]";
}
```

Container Printing

```
#include <ostream>

template <class Cont>
void print(std::ostream& os, Cont const& x)
{
    char const* prefix = "[ ";

    typedef typename Cont::const_iterator iter;
    for (iter p = x.begin(); p != x.end(); ++p)
    {
        os << prefix << *p;
        prefix = ", ";
    }
    os << " ]";
}
```


boostpro computing

Container Printing

```
#include <ostream>

template <class Cont>
void print(std::ostream& os, Cont const& x)
{
    char const* prefix = "[ ";

    typedef typename Cont::const_iterator iter;
    for (iter p = x.begin(); p != x.end(); ++p)
    {
        os << prefix << *p;
        prefix = ", ";
    }
    os << " ]";
}
```

Container Printing

```
#include <boost/foreach.hpp>
#include <ostream>

template <class Cont>
void print(std::ostream& os, Cont const& x)
{
    char const* prefix = "[ ";

    typedef typename Cont::value_type elt;
    BOOST_FOREACH( elt const& e, x )
    {
        os << prefix << e;
        prefix = ", ";
    }
    os << " ]";
}
```


boostpro computing

Container Printing

```
#include <boost/foreach.hpp>
#include <ostream>

template <class Cont>
void print(std::ostream& os, Cont const& x)
{
    char const* prefix = "[ ";

    typedef typename Cont::value_type elt;
    BOOST_FOREACH( elt const& e, x )
    {
        os << prefix << e;
        prefix = ", ";
    }
    os << " ]";
}
```

Container Printing

```
#include <boost/foreach.hpp>
#include <ostream>

template <class Cont>
void print(std::ostream& os, Cont const& x)
{
    char const* prefix = "[ ";

    typedef typename Cont::value_type elt;
    BOOST_FOREACH( elt const& e, x )
    {
        os << prefix << e;
        prefix = ", ";
    }
    os << " ]";
}
```

Container Printing

```
#include <boost/foreach.hpp>
#include <ostream>

template <class Cont>
void print(std::ostream& os, Cont const& x)
{
    char const* prefix = "[ ";

    typedef typename Cont::value_type elt;
    BOOST_FOREACH( elt const& e, x )
    {
        os << prefix << e;
        prefix = ", ";
    }
    os << " ]";
}
```

Container Printing

```
#include <boost/foreach.hpp>
#include <ostream>

template <class Cont>
void print(std::ostream& os, Cont const& x)
{
    char const* prefix = "[ ";

    typedef typename Cont::value_type elt;
    BOOST_FOREACH( elt const& e, x )
    {
        os << prefix << e;
        prefix = ", ";
    }
    os << " ]";
}
```

Range Printing

```
#include <boost/foreach.hpp>
#include <boost/range.hpp>
#include <ostream>
template <class Cont>
void print(std::ostream& os, Cont const& x)
{
    char const* prefix = "[ ";

    typedef typename boost::range_value<Cont>::type elt;
    BOOST_FOREACH( elt const& e, x )
    {
        os << prefix << e;
        prefix = ", ";
    }
    os << " ]";
}
```

Range Printing

```
#include <boost/foreach.hpp>
#include <boost/range.hpp>
#include <ostream>
template <class Cont>
void print(std::ostream& os, Cont const& x)
{
    char const* prefix = "[ ";

    typedef typename boost::range_value<Cont>::type elt;
    BOOST_FOREACH( elt const& e, x )
    {
        os << prefix << e;
        prefix = ", ";
    }
    os << " ]";
}
```

Range Printing

```
#include <boost/foreach.hpp>
#include <boost/range.hpp>
#include <ostream>
template <class Cont>
void print(std::ostream& os, Cont const& x)
{
    char const* prefix = "[ ";

    typedef typename boost::range_value<Cont>::type elt;
    BOOST_FOREACH( elt const& e, x )
    {
        os << prefix << e;
        prefix = ", ";
    }
    os << " ]";
}
```

We'll come back to Boost.Range

Range Printing

```
#include <boost/foreach.hpp>
#include <boost/range.hpp>
#include <ostream>
template <class Cont>
void print(std::ostream& os, Cont const& x)
{
    char const* prefix = "[ ";

    typedef typename boost::range_value<Cont>::type elt;
    BOOST_FOREACH( elt const& e, x )
    {
        os << prefix << e;
        prefix = ", ";
    }
    os << " ]";
}
```


boostpro computing

Yes, We Can

```
int rng[3] = { 42, 314, 77 };
BOOST_FOREACH( int var, rng )
{
    . . .
}
```


boostpro computing

Yes, We Can

```
std::vector<int> rng;
BOOST_FOREACH( int var, rng )
{
    . . .
}
```


boostpro computing

Yes, We Can

```
std::vector<int> rng;
BOOST_FOREACH( float var, rng )
{
    . . .
}
```


boostpro computing

Yes, We Can

```
YourType rng;
BOOST_FOREACH( int var, rng )
{
    . . .
}
```


boostpro computing

Yes, We Can

```
BOOST_FOREACH( int var, rng )
{
    continue;
}
```


boostpro computing

Yes, We Can

```
BOOST_FOREACH( int var, rng )
{
    break;
}
```


boostpro computing

Yes, We Can

```
BOOST_FOREACH( int var, rng )
{
    return;
}
```


boostpro computing

Yes, We Can

```
BOOST_FOREACH( int var, rng )
{
    goto considered_harmful;
}
```


boostpro computing

Yes, We Can

```
BOOST_FOREACH( int var, rng )
{
    goto considered_harmful;
}
```

google it

boostpro computing

Yes, We Can

```
BOOST_FOREACH( int var, rng )
{
    goto considered_harmful;
}
```


boostpro computing

Yes, We Can

```
int var;
BOOST_FOREACH( var, rng )
{
    ...
}

use_last_value_of(var);
```


boostpro computing

Yes, We Can

```
BOOST_FOREACH( int& var, rng )
{
    var += 2;
}
```


boostpro computing

Yes, We Can

```
extern std::vector<int> get_ints();
BOOST_FOREACH( int var, get_ints() )
{
    . . .
}
```


boostpro computing

Exercise

(this one's a "gimme")

- Use Boost.ForEach to re-implement JSON array and object printing

- Play around with it. Can you think of places to apply it in your work?