Dynamic C++

POCO

C++ PORTABLE COMPONENTS

alex@pocoproject.org

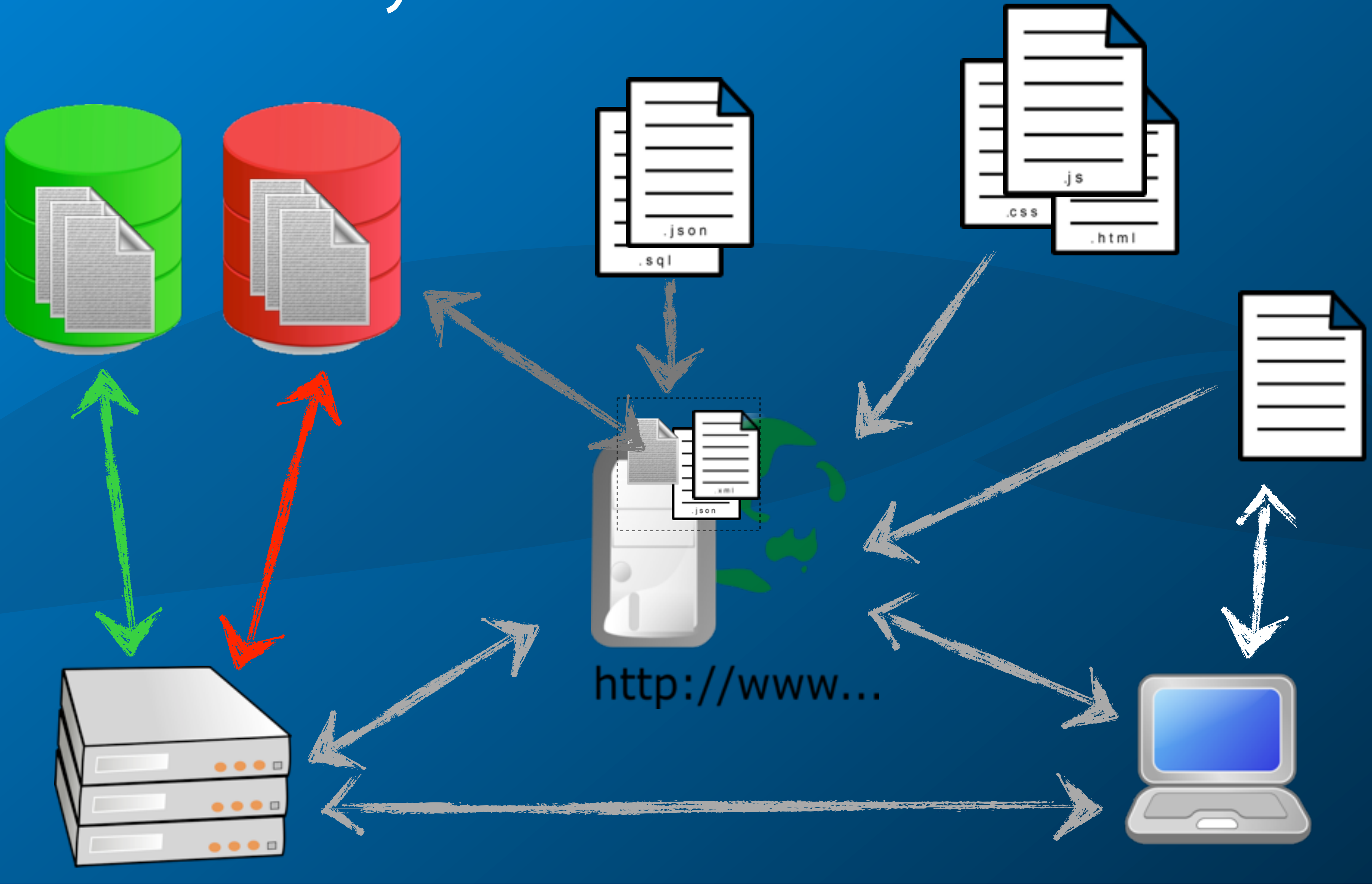
Content

- The Problem
- The Solution
- The Anatomy of the Solution
- The Heart and Soul of the Solution
- Let’s Dance - code example
- Other solutions
- Performance and comparisons
- Conclusion

"Without a good library, most interesting tasks are hard to do in C++; but given a good library, almost any task can be made easy."

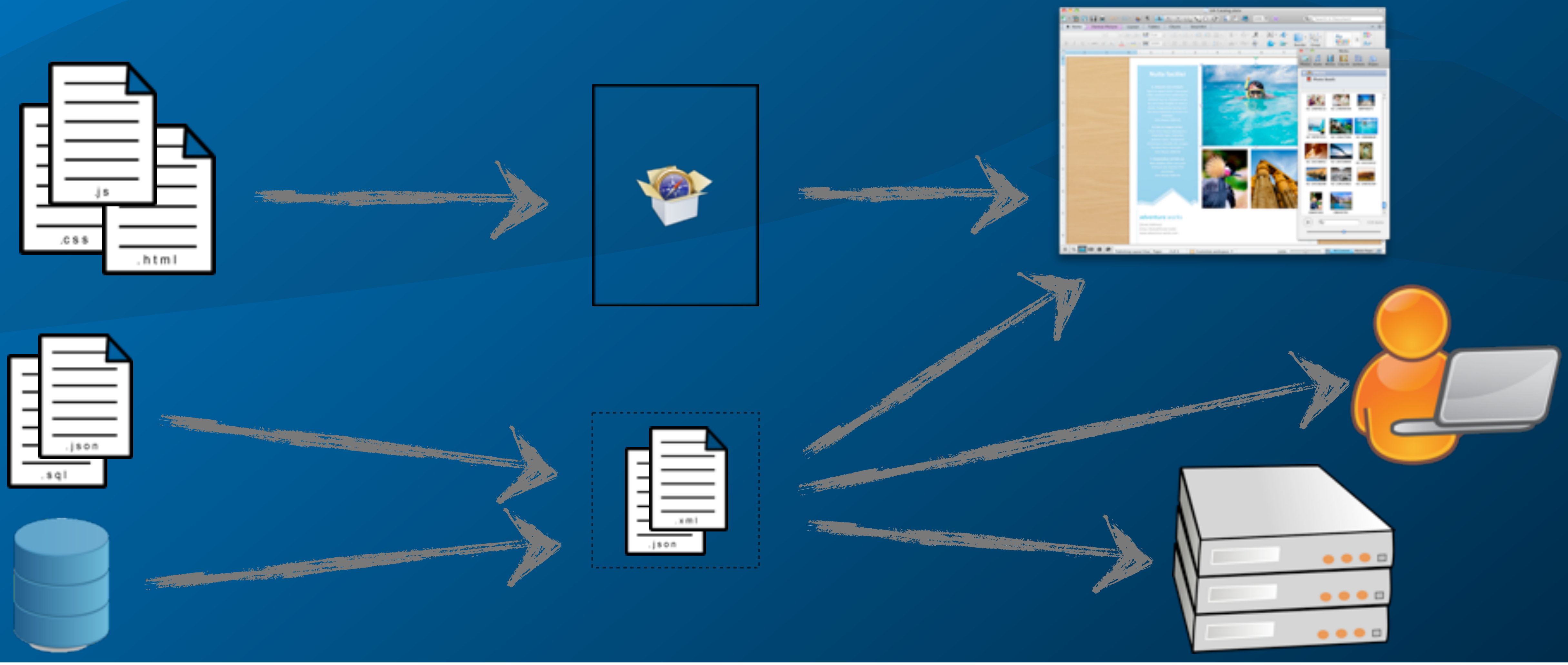
Bjarne Stroustrup
(designer and original implementor of C++)

A Brief History of Data Access

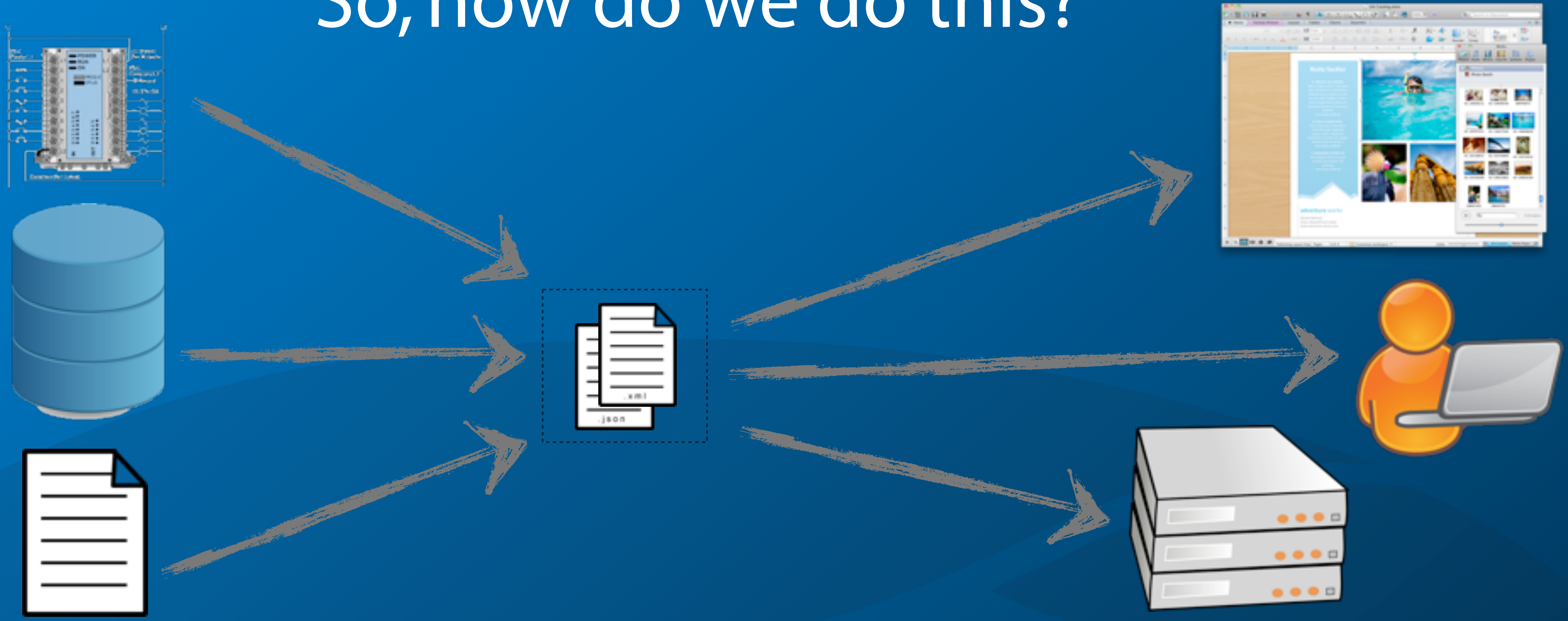


Data Formats

- often in proprietary binary format
- transform into character strings of desired format
- server-side needs an equivalent of HTML rendering engine



So, how do we do this?



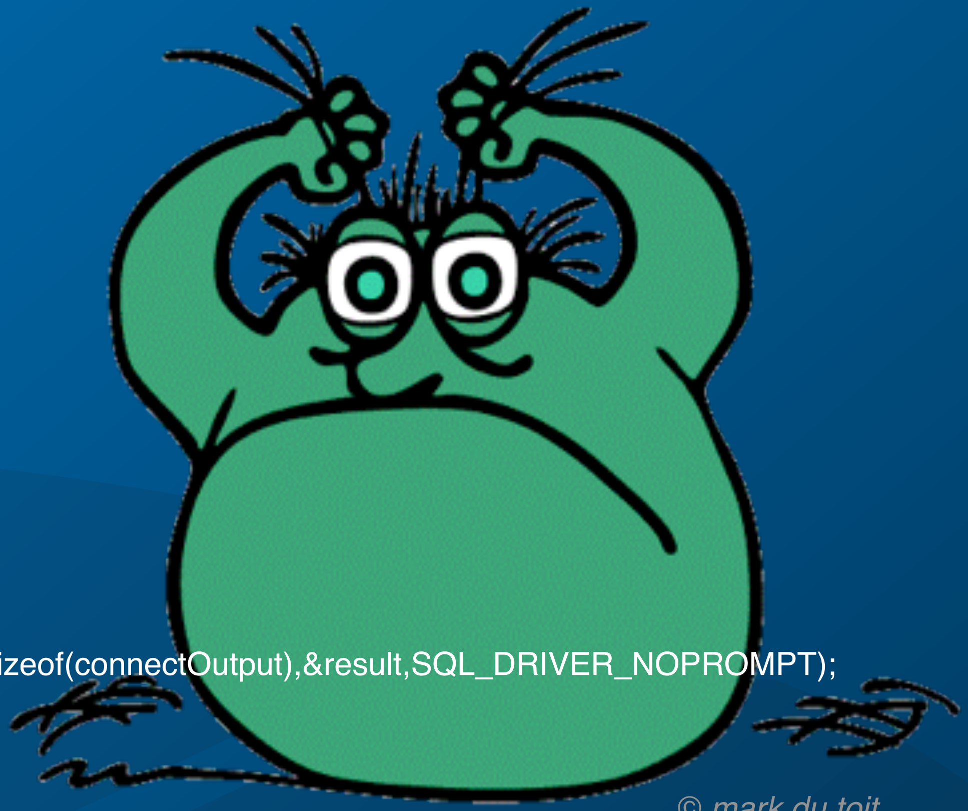
- generate the desired format in the database :-\
- use dynamic language
- mix HTML with server-side code and compile on the fly (shudder)
- browser plugin (double-shudder)
- or ... use static language on the server-side and AJA(X|J) in the browser?

The Problem

```
SELECT * FROM Simpsons
```

- discover column count ✓
- discover column data types ✓
- bind returned data to variables ✗

“solution”



© mark du toit

```
SQLRETURN rc;
SQLHENV henv = SQL_NULL_HENV;
SQLHDBC hdbc = SQL_NULL_HDBC;
SQLHSTMT hstmt = SQL_NULL_HSTMT;

rc = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
odbc_check_env (rc, henv);
rc = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, 0);
odbc_check_env (rc, henv);

rc = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
odbc_check_dbc (rc, hdbc);

SQLCHAR connectOutput[1024] = {0};
SQLSMALLINT result;
rc = SQLDriverConnect(hdbc,NULL,(SQLCHAR*)dbConnString.c_str(),(SQLSMALLINT)SQL_NTS,connectOutput,sizeof(connectOutput),&result,SQL_DRIVER_NOPROMPT);
odbc_check_dbc (rc, hdbc);

sql = "SELECT * FROM Simpsons";
SQLCHAR* pStr = (SQLCHAR*) sql.c_str();
rc = SQLPrepare(hstmt, pStr, (SQLINTEGER) sql.length());
odbc_check_stmt (rc, hstmt);

char name[50] = { 0 };
SQLLEN lengths[3] = { 0 };
int age = 0;
float weight = 0.0f;
std::memset(&sixth, 0, sizeof(sixth));
rc = SQLBindCol(hstmt, (SQLUSMALLINT) 1,  SQL_C_CHAR, (SQLPOINTER) chr, (SQLINTEGER) sizeof(chr[0]), &lengths[0]);
odbc_check_stmt (rc, hstmt);

rc = SQLBindCol(hstmt, (SQLUSMALLINT) 2, SQL_C_INTEGER, (SQLPOINTER) &age, (SQLINTEGER) sizeof(age), &lengths[1]);
odbc_check_stmt (rc, hstmt);

rc = SQLBindCol(hstmt, (SQLUSMALLINT) 3, SQL_C_BINARY, (SQLPOINTER) &weight, (SQLINTEGER) sizeof(weight), &lengths[2]);
odbc_check_stmt (rc, hstmt);

printf(“Name: %s, Age: %d, Weight: %f”, name, age, weight);
```

The Solution

© mark du toit

```
using namespace Poco::Data::SQLite;

int main()
{
   Session session("SQLite", "simpsons.db");

   std::cout << RecordSet(session,
                 "SELECT * FROM Simpsons");

   return 0;
}
```

The Anatomy of the Solution
(step - by - step)

```
Statement stmt =
(session << "SELECT * FROM Simpsons", now);

RecordSet rs(stmt);

ostream& operator << (ostream &os,
                      const RecordSet& rs)
{
   return rs.copy(os);
}
```

The Anatomy of the Solution
(under the hood)

```
using namespace std;

ostream& RecordSet::copy(ostream& os, size_t offset = 0, size_t length = END)
{
    RowFormatter& rf = (*_pBegin)->getFormatter();
    os << rf.prefix();
    copyNames(os);
    copyValues(os, offset, length);
    os << rf.postfix();
    return os;
}

ostream& RecordSet::copyValues(ostream& os, size_t offset, size_t length)
{
    RowIterator begin = *_pBegin + offset;
    RowIterator end = (RowIterator::END != length) ? it + length : *_pEnd;
    std::copy(begin, end, std::ostream_iterator<Row>(os));
    return os;
}
```

The Anatomy of the Solution, contd.

(STL - compliance)

```
Row& RowIterator::operator * ()
{
    if (END == _position)
        throw InvalidAccessException("End of iterator reached.");

    return _pRecordSet->row(_position);
}

ostream& operator << (ostream &os, const Row& row)
{
    os << row.valuesToString();
    return os;
}

const string& Row::valuesToString() const
{
    return _pFormatter->formatValues(values(), _valueStr);
}
```

The Heart of the Solution

(Row::set)

```
class Row
{
public:
    // ...
    template <typename T>
    void set(size_t pos, const T& val)
    {
        try { _values.at(pos) = val; }
        catch (out_of_range&)
        { throw RangeException("Invalid column."); }
    }
    // ...
private:
    vector<Poco::Dynamic::Var> _values;
};
```

The Soul of the Machine

(Poco::Dynamic::Var)

```
namespace Poco {
namespace Dynamic {

class Var
{
public:
    // ...
    template <typename T>
    Var(const T& val):
        _pHolder(new VarHolderImpl<T>(val))
    {
    }

    // ...
private:
    VarHolder* _pHolder;
};

} }
```

* Design based on boost::any

So, where was boost::any found lacking ?

It’s a great idea with limited applicability -
dynamic on receiving, but *static* on the giving end.

```
using boost::any;
using boost::any_cast;

typedef std::list<any> many;

int ival = 42;
std::string sval = “fourty two”;

values.push_back(ival);
values.push_back(sval);

std::string sival = values[0]; // oops!, compile error
sival = any_cast<std::string>(values[0]); // still oops!, throw
```

Var in Practical Use

```
std::string str("42");
Var v1 = str; // "42"
double d = v1; // 42.0
Var v2 = d + 1.0; // 43.0
float f = v2 + 1; // 44.0
```

```
DynamicStruct aStruct;
aStruct["First Name"] = "Junior";
aStruct["Last Name"] = "POCO";
aStruct["Age"] = 1;
Var a1(aStruct);
std::string res = a1.convert<std::string>();
// { "Age”: 1, "First Name": "Junior", "Last Name" : "POCO" }
```

```
std::string s1("string");
Poco::Int8 s2(23);
std::vector<Var> s16;
s16.push_back(s1);
s16.push_back(s2);
Var a1(s16);
std::string res = a1.convert<std::string>();
// ["string", 23]
```

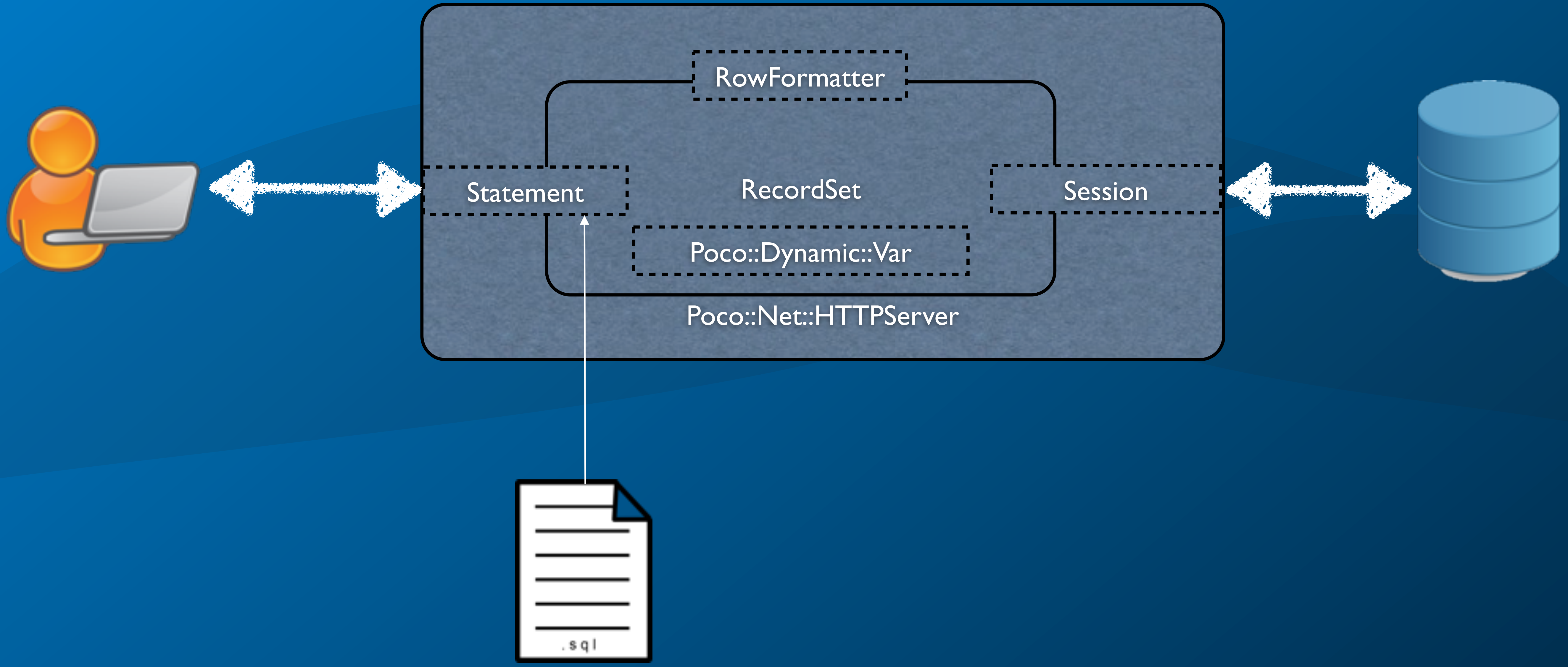
What Else is in the Var Box

- Dynamic array, pair and struct (map) support (Poco::Dynamic::Pair/Struct)
- JSON (de)serialization of the above
- Empty value support (very handy with null DB fields)
- Strict conversion checks

The Soul of the Machine
(Poco::Dynamic::VarHolder)

```
namespace Poco {
namespace Dynamic {

class VarHolder
{
public:
    virtual ~VarHolder();
    virtual void convert(int& val) const;
    // ...
protected:
    VarHolder();
    // ...
};

template <typename T> // for end-user extensions
class VarHolderImpl: public VarHolder
{
    //...
};

template <> // native and frequently used types specializations provided by POCO
class VarHolderImpl<int>: public VarHolder
{
    //...
};

//...
}}
```

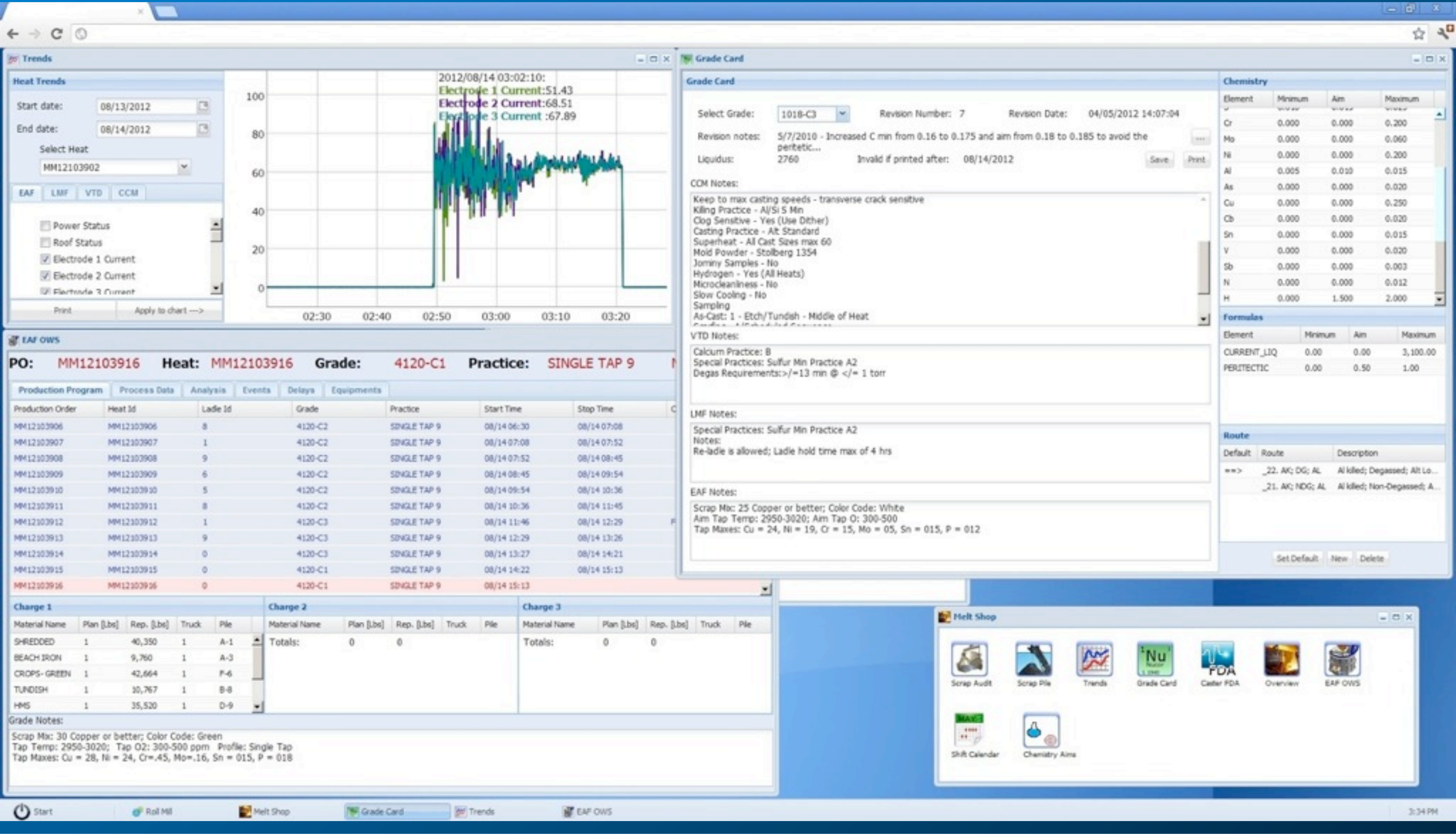
The Machine Assembled



Let’s Dance

```
class DataRequestHandler: public HTTPRequestHandler
{
public:
  void handleRequest(HTTPServerRequest& request,
                HTTPServerResponse& response)
  {
    response.setChunkedTransferEncoding(true);
    response.setContentType("text/xml");

    ostream& ostr = response.send();
    Session  sess("SQLite", "sample.db");

    ostr << RecordSet(sess,
                "SELECT * FROM Simpsons",
                new XMLFormatter);
  }
};
```

A Real World Example



Start date: 08/13/2012
End date: 08/14/2012
Select Heat: MM12103902

☐ Power Status
☐ Roof Status
☑ Electrode 1 Current
☑ Electrode 2 Current
☑ Electrode 3 Current

PO: MM12103916 **Heat:** MM12103916 **Grade:** 4120-C1 **Practice:** SINGLE TAP 9

Production Order	Heat Id	Ladle Id	Grade	Practice	Start Time	Stop Time
MM12103906	MM12103906	8	4120-C2	SINGLE TAP 9	08/14 06:30	08/14 07:08
MM12103907	MM12103907	1	4120-C2	SINGLE TAP 9	08/14 07:08	08/14 07:52
MM12103908	MM12103908	9	4120-C2	SINGLE TAP 9	08/14 07:52	08/14 08:45
MM12103909	MM12103909	6	4120-C2	SINGLE TAP 9	08/14 08:45	08/14 09:54
MM12103910	MM12103910	5	4120-C2	SINGLE TAP 9	08/14 09:54	08/14 10:36
MM12103911	MM12103911	8	4120-C2	SINGLE TAP 9	08/14 10:36	08/14 11:45
MM12103912	MM12103912	1	4120-C3	SINGLE TAP 9	08/14 11:46	08/14 12:29
MM12103913	MM12103913	9	4120-C3	SINGLE TAP 9	08/14 12:29	08/14 13:26
MM12103914	MM12103914	0	4120-C3	SINGLE TAP 9	08/14 13:27	08/14 14:21
MM12103915	MM12103915	0	4120-C1	SINGLE TAP 9	08/14 14:22	08/14 15:13
MM12103916	MM12103916	0	4120-C1	SINGLE TAP 9	08/14 15:13	

Charge 1

Material Name	Plan [Lbs]	Rep. [Lbs]	Truck	Pile
SHREDDED	1	40,350	1	A-1
BEACH IRON	1	9,760	1	A-3
CROPS- GREEN	1	42,664	1	F-6
TUNDISH	1	10,767	1	B-8
HMS	1	35,520	1	D-9

Charge 2

Material Name	Plan [Lbs]	Rep. [Lbs]	Truck	Pile
Totals:	0	0		

Charge 3

Material Name	Plan [Lbs]	Rep. [Lbs]	Truck	Pile
Totals:	0	0		

Grade Notes:

Scrap Mix: 30 Copper or better; Color Code: Green
Tap Temp: 2950-3020; Tap O2: 300-500 ppm Profile: Single Tap
Tap Maxes: Cu = 28, Ni = 24, Cr=.45, Mo=.16, Sn = 015, P = 018

Select Grade: 1018-C3 Revision Number: 7 Revision Date: 04/05/2012 14:07:04
Revision notes: 5/7/2010 - Increased C min from 0.16 to 0.175 and aim from 0.18 to 0.185 to avoid the peritetic...
Liquidus: 2760 Invalid if printed after: 08/14/2012

CCM Notes:

Keep to max casting speeds - transverse crack sensitive
Killing Practice - Al/Si S Min
Clog Sensitive - Yes (Use Dither)
Casting Practice - Alt Standard
Superheat - All Cast Sizes max 60
Mold Powder - Stolberg 1354
Jominy Samples - No
Hydrogen - Yes (All Heats)
Microcleanliness - No
Slow Cooling - No
Sampling
As-Cast: 1 - Etch/Tundish - Middle of Heat

VTD Notes:

Calcium Practice: B
Special Practices: Sulfur Min Practice A2
Degas Requirements:>/=13 min @ </= 1 torr

LMF Notes:

Special Practices: Sulfur Min Practice A2
Notes:
Re-ladle is allowed; Ladle hold time max of 4 hrs

EAF Notes:

Scrap Mix: 25 Copper or better; Color Code: White
Aim Tap Temp: 2950-3020; Aim Tap O: 300-500
Tap Maxes: Cu = 24, Ni = 19, Cr = 15, Mo = 05, Sn = 015, P = 012

Chemistry

Element	Minimum	Aim	Maximum
Cr	0.000	0.000	0.200
Mo	0.000	0.000	0.060
Ni	0.000	0.000	0.200
Al	0.005	0.010	0.015
As	0.000	0.000	0.020
Cu	0.000	0.000	0.250
Cb	0.000	0.000	0.020
Sn	0.000	0.000	0.015
V	0.000	0.000	0.020
Sb	0.000	0.000	0.003
N	0.000	0.000	0.012
H	0.000	1.500	2.000

Formulas

Element	Minimum	Aim	Maximum
CURRENT_LIQ	0.00	0.00	3,100.00
PERITECTIC	0.00	0.50	1.00

Route

Default	Route	Description
==>	_22. AK; DG; AL	Al killed; Degassed; Alt Lo...
	_21. AK; NDG; AL	Al killed; Non-Degassed; A...

Is it REALLY Dynamic?

Of course not.
Dig deep enough and there is no such thing as *dynamic*.

Type handlers are templates, hence generated by compiler and statically checked for type.

And there’s a price to pay ...

```
Binary sizes:
============
 Linux
 -----
  5160 AnySize.o
 23668 DynamicAnySizeExtract.o

 25152 DynamicAnySizeConvert.o
  9600 lexical_cast_size.o

 Windows
 -------
  26,924 AnySize.obj
  96,028 DynamicAnySizeExtract.obj

 103,943 DynamicAnySizeConvert.obj
  84,217 lexical_cast_size.obj



Lines of code:
=============
 Any              145
 DynamicAny*    3,588
 lexical_cast     971
```

But what if I need performance?

There is, of course, a lean and elegant static workaround.
In fact, several of them ...

```
struct Person
{
   std::string name;
   std::string address;
   int         age;
};
```

Scaffolding - wrap Person into a TypeHandler

```
namespace Poco {
namespace Data {

template <>
class TypeHandler<Person>
{
public:
    static std::size_t size()
    {
        return 3;
    }

    static void bind(size_t pos, const Person& person, AbstractBinder* pBinder, Direction dir)
    {
        TypeHandler<std::string>::bind(pos++, person.name, pBinder, dir);
        TypeHandler<std::string>::bind(pos++, person.address, pBinder, dir);
        TypeHandler<int>::bind(pos++, person.age, pBinder, dir);
    }

    static void extract(size_t pos, Person& person, const Person& deflt, AbstractExtractor* pE)
    {
        TypeHandler<std::string>::extract(pos++, person.name, deflt.name, pExtr);
        TypeHandler<std::string>::extract(pos++, person.address, deflt.address, pExtr);
        TypeHandler<int>::extract(pos++, person.age, deflt.age, pExtr);
    }
};

} }
```

And Life is Good Again

```
Person person =
{
    "Bart Simpson",
    "Springfield",
    12
};

session << "INSERT INTO Person VALUES(?, ?, ?)", use(person);

std::vector<Person> people;
session << "SELECT Name, Address, Age FROM Person",into(people), now;



 std::string name, address;
 int age;
 session << "INSERT INTO Person VALUES(?, ?, ?)",
            use(name),
            use(address),
            use(age);
```

But wait, there's more!

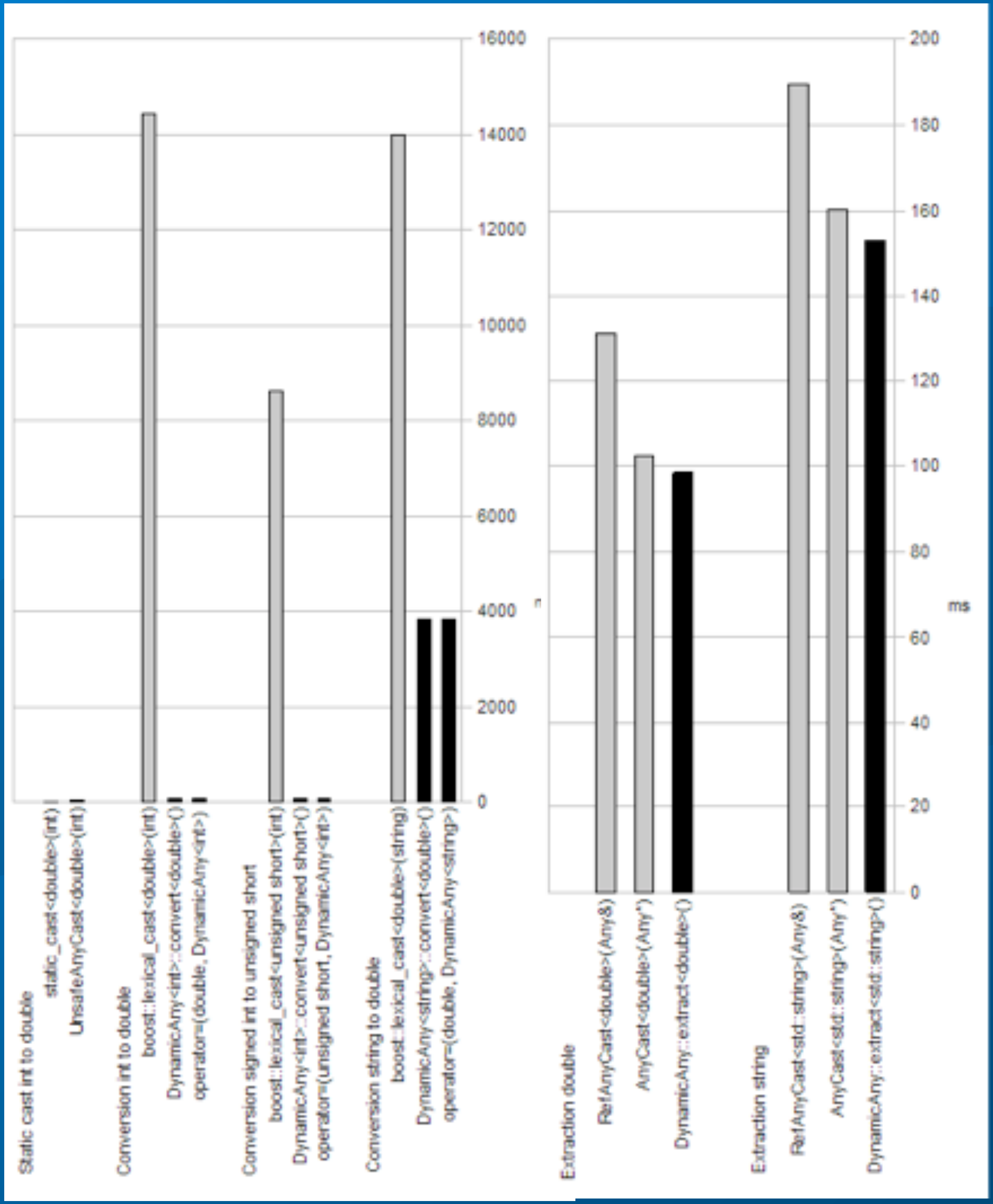
```
using namespace std;
using namespace Poco;
typedef Tuple<string, string, int> Person;
typedef vector<Person> People;


People people;
people.push_back(Person("Bart Simpson", "Springfield", 12));
people.push_back(Person("Lisa Simpson", "Springfield", 10));

session << "INSERT INTO Person VALUES(?, ?, ?)", use(people), now;

people.clear();

session << "SELECT Name, Address, Age FROM Person", into(people), now;
```

What else is out there ?

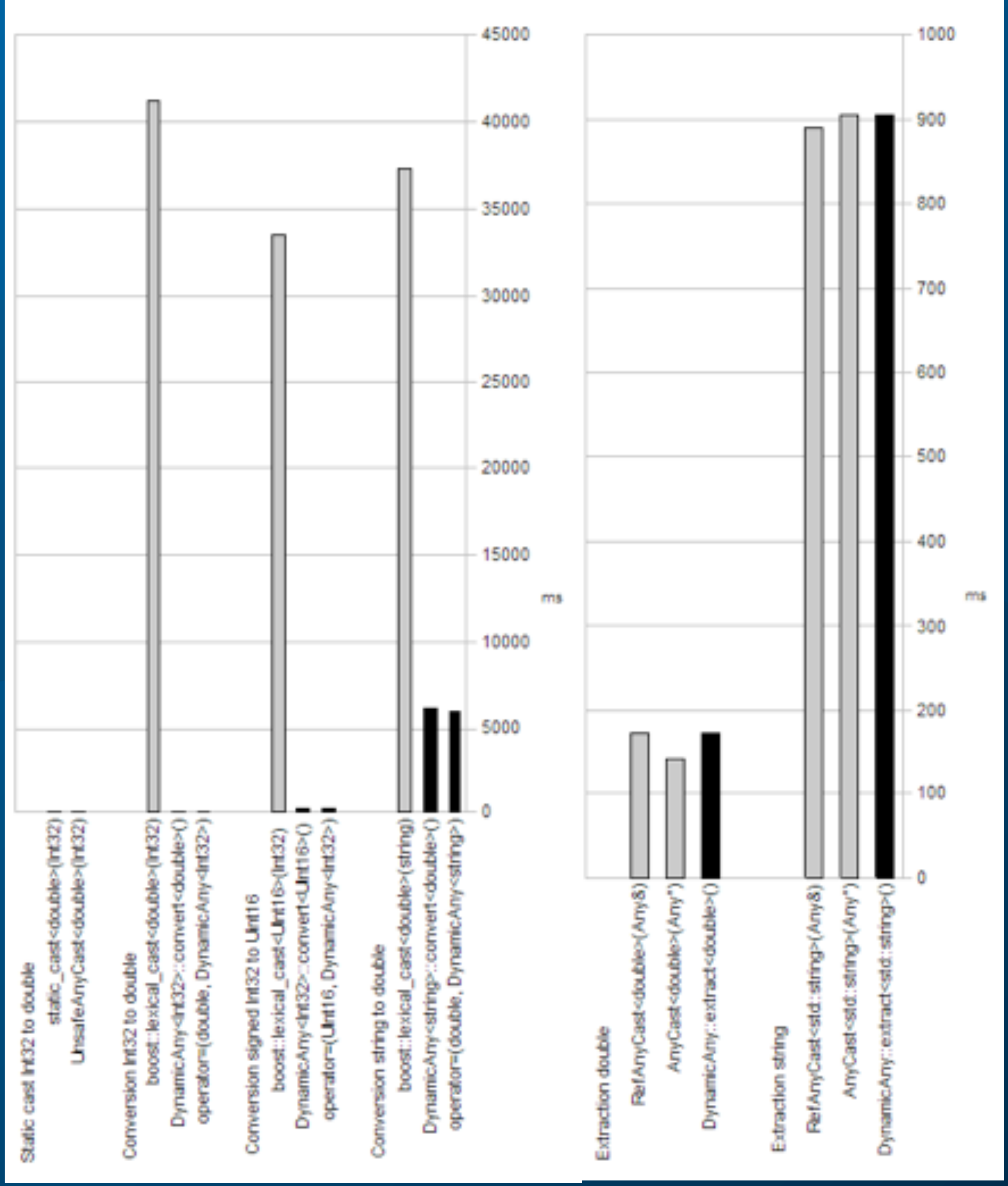
- void*
- C union
- MS COM Variant
- boost::variant
- boost::lexical_cast
- boost::type_erasure
- folly::dynamic

Performance

Linux



Windows



ACCU Overload Journal Articles

http://accu.org/index.php/journals/1502

http://accu.org/index.php/journals/1511

Last but not Least

POCO

http://pocoproject.org

http://poco.svn.sourceforge.net/viewvc/poco/poco/trunk

https://poco.svn.sourceforge.net/svnroot/poco/poco/trunk

- large, comprehensive, well-designed framework
- designed for practical everyday use, with end-user in mind
- makes C++ programming fun
- 100% standard C++
- not reinventing the wheel (except when necessary ;-)

got POCO ?

C++ PORTABLE COMPONENTS

http://pocoproject.org alex@pocoproject.org