C++ now 2013

Look Ma,

"*update DB to HTML5 using C++*",

no hands!

POCO C++ PORTABLE COMPONENTS

alex@pocoproject.org

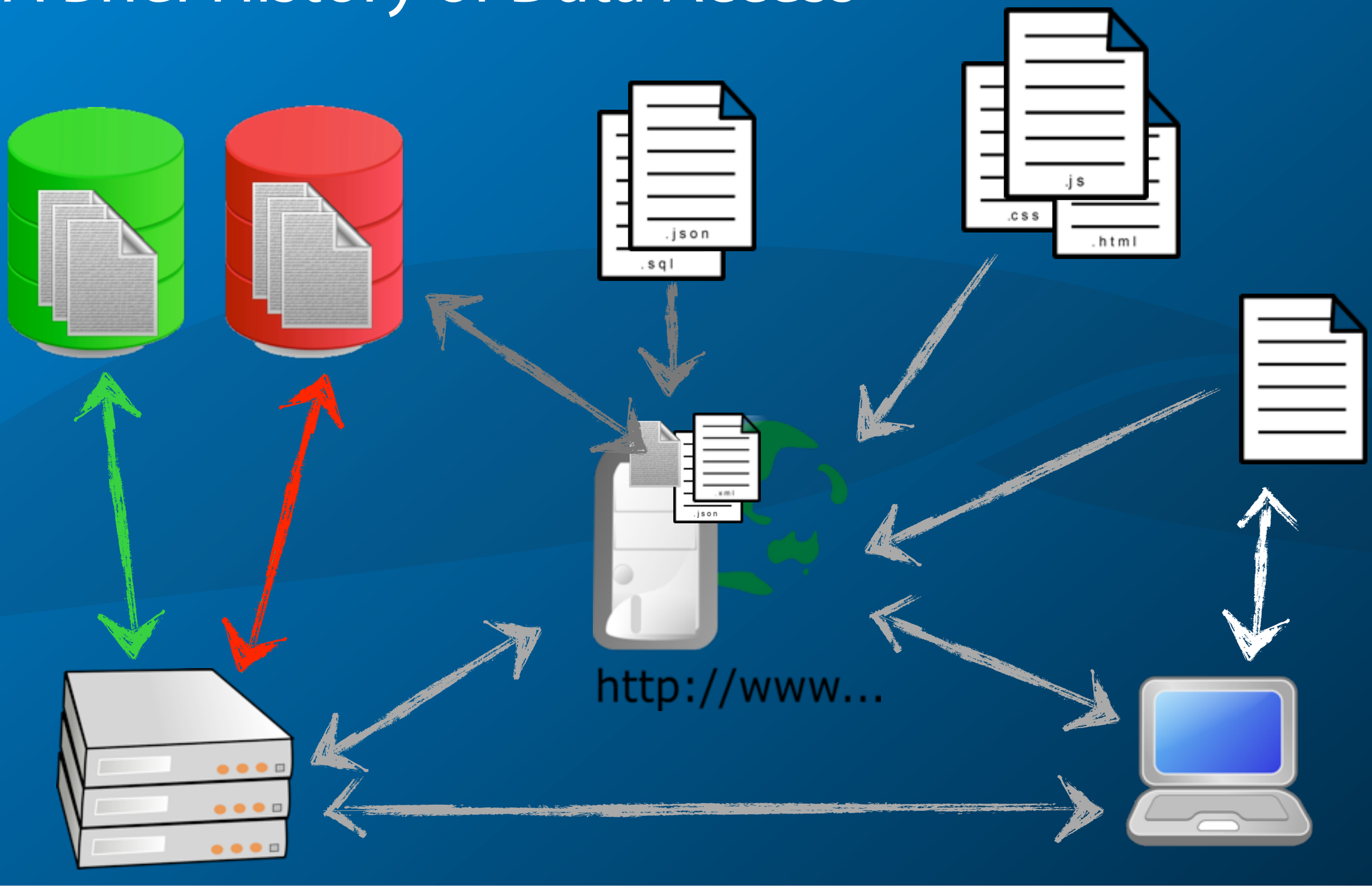
Content

- The Problem
- The Solution
- The Anatomy of the Solution
- The Heart and Soul of the Solution
- Let’s Dance - code example
- Performance-conscious options
- A better solution - from DB to HTML, no hands
- Conclusion

"A man will be imprisoned in a room with a door that's unlocked and opens inwards; as long as it does not occur to him to pull rather than push.

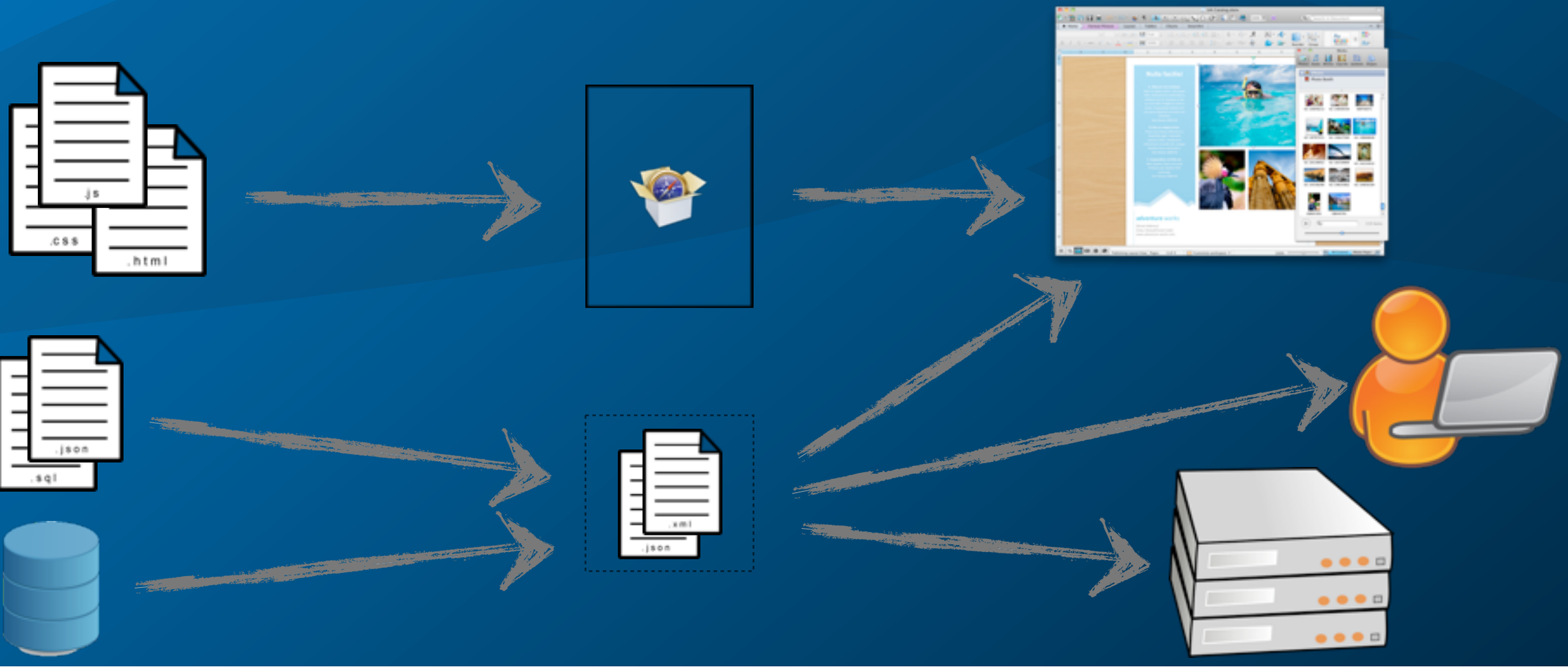
Ludwig Wittgenstein

A Brief History of Data Access

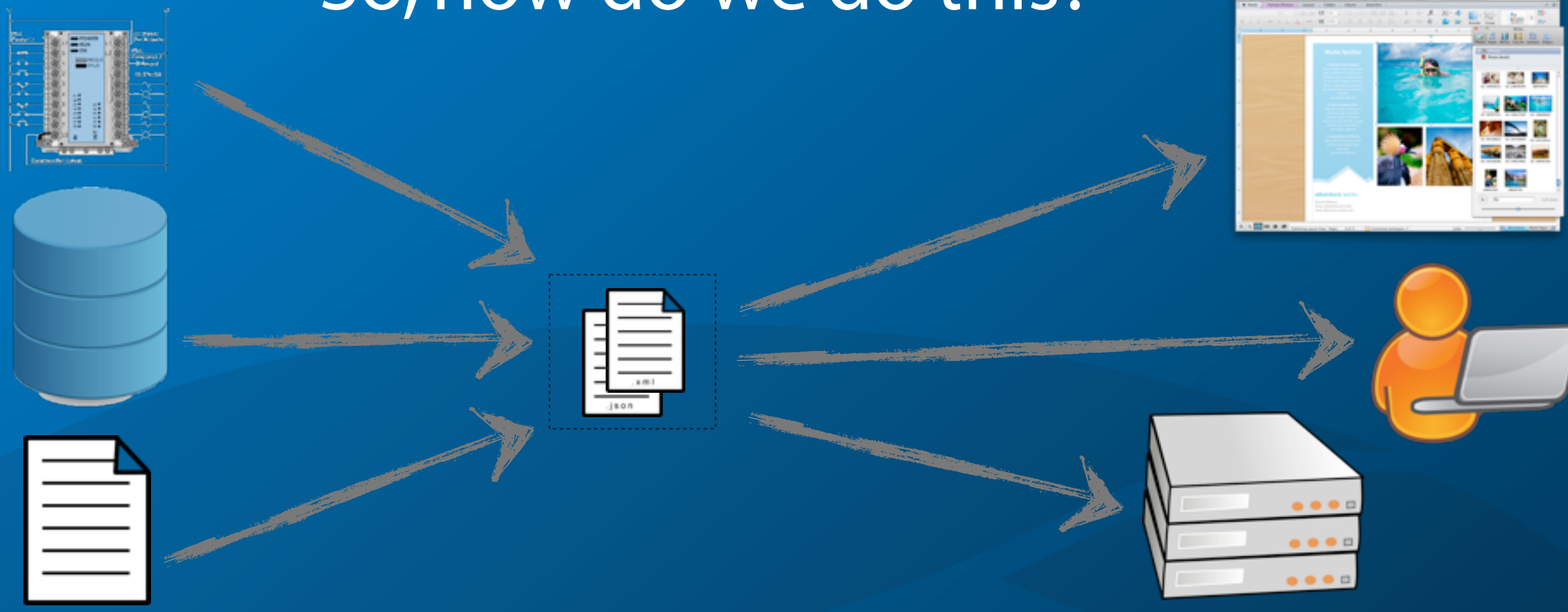


Data Formats

- 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”

```
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


```
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;
};

} } // namespace Poco::Dynamic
```

* 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]
```

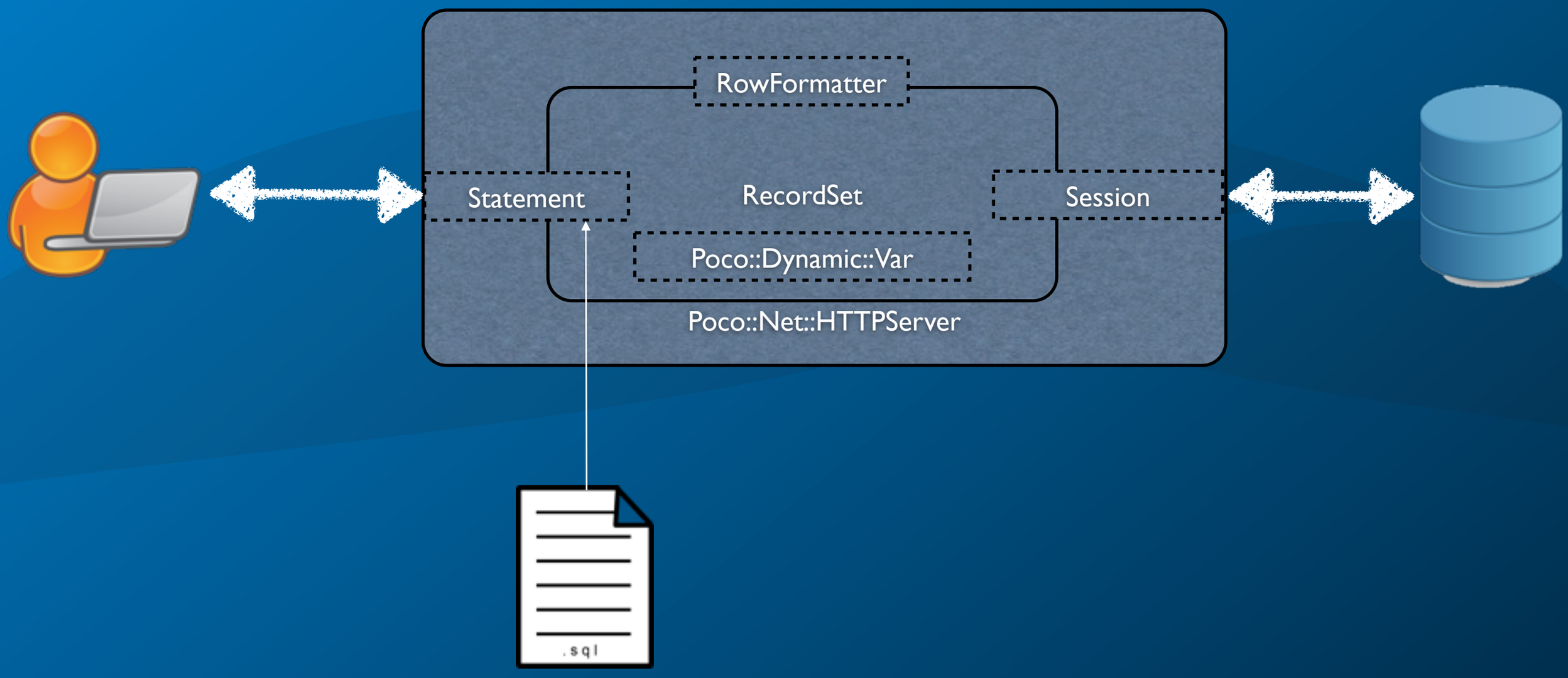
What Else is in the Var Box

- 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
{
    //...
};

//...
}}
```

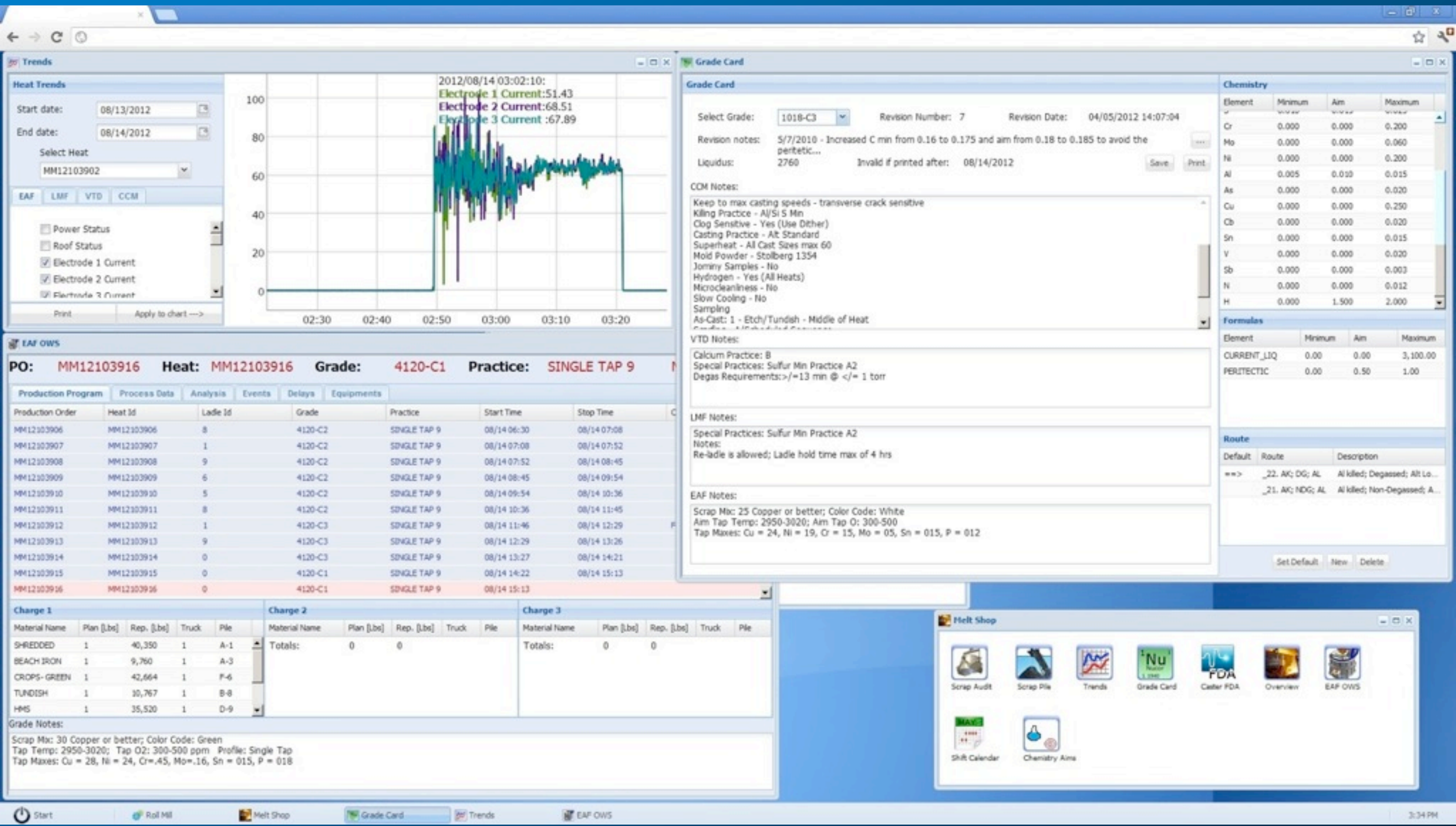
The Machine Assembled



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",
                      XMLFormatter());
  }
};
```

A Real World Example



Is it REALLY Dynamic?

In a sense, yes - values are instantiated at runtime.

But they're strongly typed and early bound.

Dig deep enough and there is no such thing as *dynamic*.

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::Ptr 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::Ptr p)
    {
        TypeHandler<std::string>::extract(pos++, person.name, deflt.name, p);
        TypeHandler<std::string>::extract(pos++, person.address, deflt.address, p);
        TypeHandler<int>::extract(pos++, person.age, deflt.age, p);
    }
};

} }
```

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!

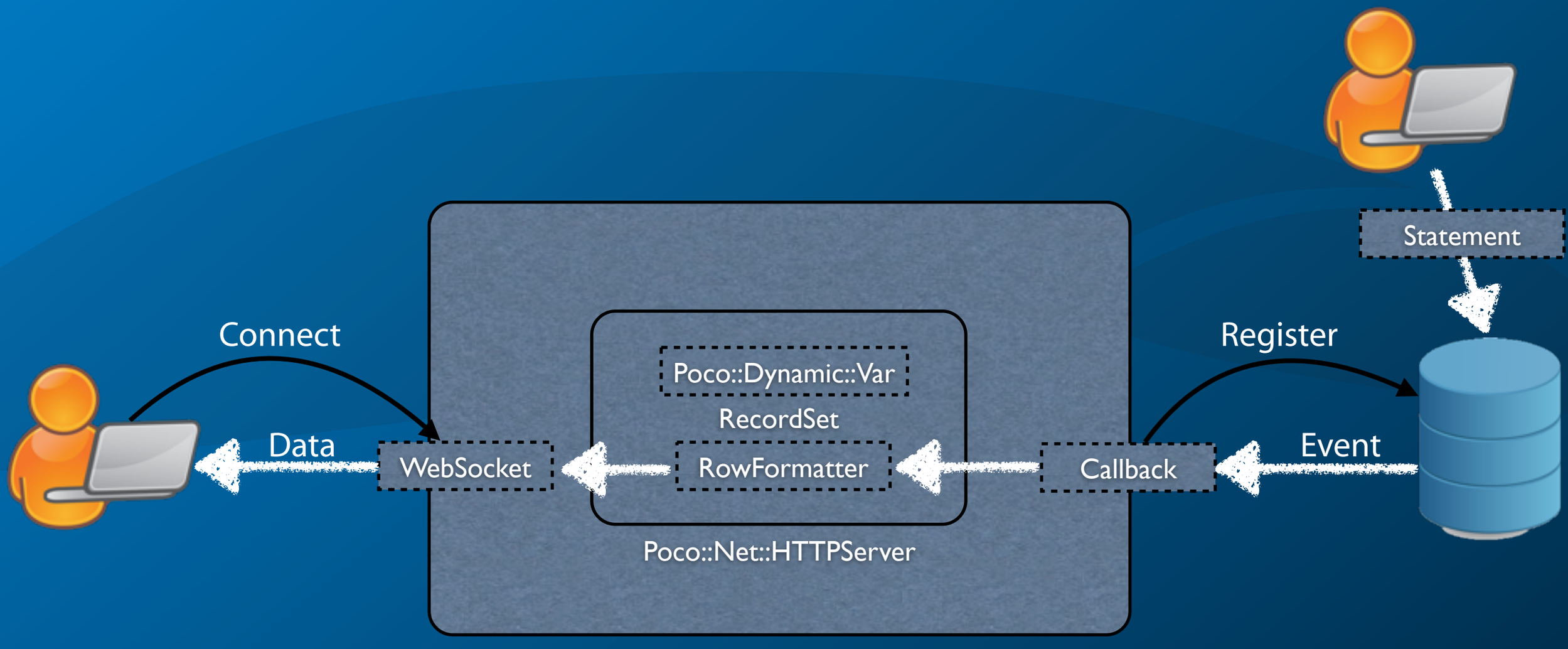
```
using namespace std;
using namespace Poco;
typedef Tuple<string, string, int> Person;
typedef vector<Person> People;


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;
```

SELECT * FROM Simpsons

Nothing but pulling so far :-(

Push, or my money back !

When Push comes to Shove …



Under the Hood - DB Events

```
DBEventHandler():_session("SQLite", "sample.db"),
    _notifier(_session)
{
    _notifier.insert += delegate(this, &DBEventHandler::onInsert);
    _notifier.update += delegate(this, &DBEventHandler::onUpdate);
}


void DBEventHandler::onInsert(const void* pSender)
{
    Notifier* pN = notifier(pSender);
    Poco::Int64 rowID = pN->getRow();
    std::cout << "Inserted row " << rowID << std::endl;
    notify(rowID);
}
```

Under the Hood - DB Event Notification

```
void DBEventHandler::notify(Poco::Int64 rowID)
{
    std::ostringstream os;
    CSVFormatter cf;
    Statement stmt =
        (_session << "SELECT rowid, Name, Address, Age
                     FROM Person
                     WHERE rowid = ?",
                     use(rowID),
                     format(cf),
                     now);

    os << RecordSet(stmt);

    _factory.handler().send(os.str());
}
```

Under the Hood - WebSocket Loop

```
try
{
    if (!_pWS)
        _pWS = new WebSocket(request, response);

    std::cout << "WebSocket connection established.";

    char buffer[1024];
    int n;
    do
    {
        n = _pWS->receiveFrame(buffer, sizeof(buffer), _flags);
    }
    while (n > 0 || (_flags & WebSocket::FRAME_OP_BITMASK) !=
                        WebSocket::FRAME_OP_CLOSE);
    std::cout << "WebSocket connection closed." << std::endl;
}
catch (WebSocketException& exc)
{ /* ... */ }
```

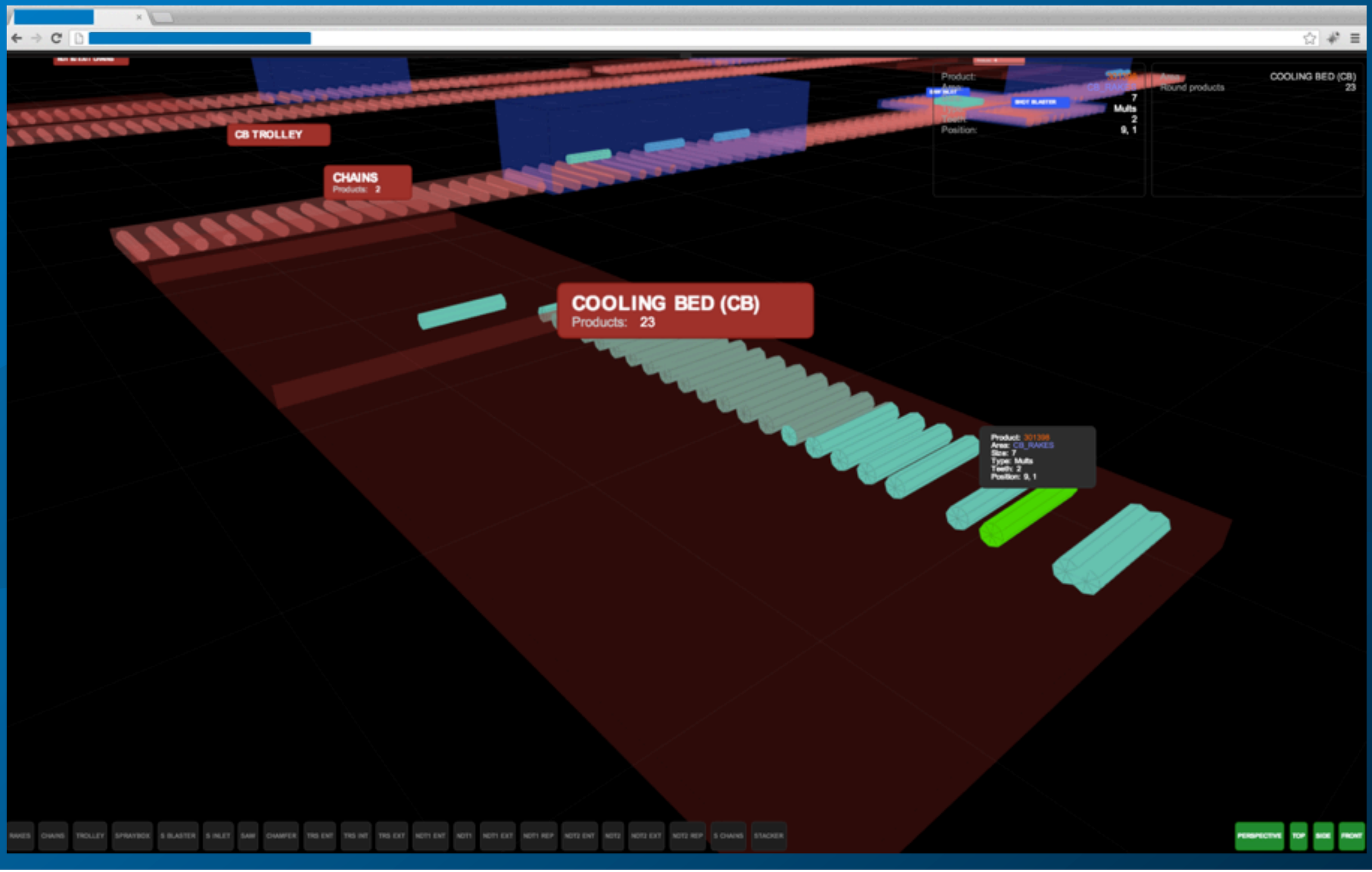
Under the Hood - WebSocket Send

```
class WebSocketRequestHandler
{
 // ...
 void send(const std::string& buffer)
 {
   _pWS->sendFrame(buffer.data(),
                   buffer.size(),
                  _flags);
 }
 // ...
};
```

Under the Hood - WebSocket, the browser end

```
function WebSocketOpen() {
  if ("WebSocket" in window) {
    ws = new WebSocket("ws://localhost:9980/ws");

    ws.onopen = function() {
      ws.send("Hello, world!");
    };

    ws.onmessage = function(evt) {
      var arr = evt.data.split(",");
      if (arr.length >= 4) {
        updateTable(arr[0], arr[1], arr[2], arr[3]);
      }
    }
};
```

A Real Virtual World Example



ACCU Overload Journal Articles

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

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

Upcoming: “Dynamic C++” (June ACCU Overload)

Last but not Least

POCO

http://pocoproject.org

https://github.com/pocoproject

- 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