C24

www.C24.biz

Turn your XML into Binary

Make it smaller and faster

John Davies | CTO

@jtdavies

QCon New York | *12th June 2014*

Computing power

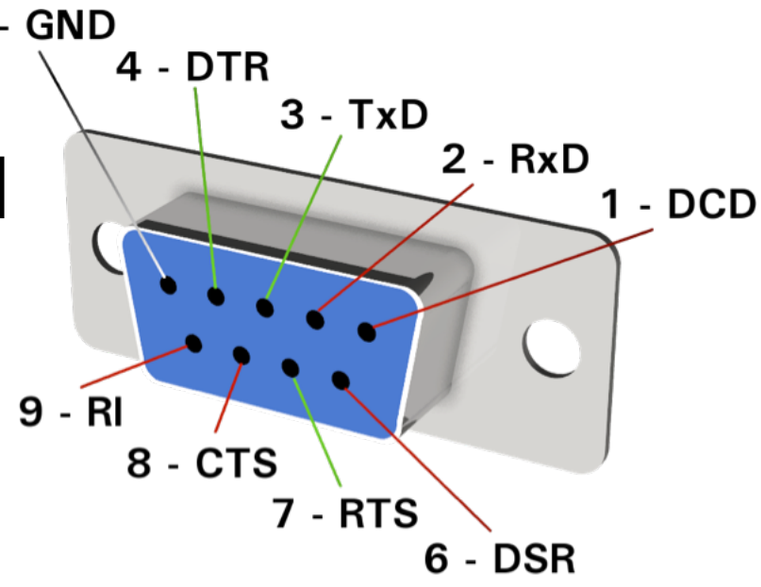
- Apollo 11's guidance computer had just 2k of memory and 32k of read-only storage
  But it got it to the moon - and back!
  - Most of the time :-)

- Today you can compress a full 1080p movie into about 1GB and watch it on your mobile phone
  - 1080p is 2 million pixels, with full colour that's 6MB per image, at 25 images a second that's about 15GB for a 100 minute film

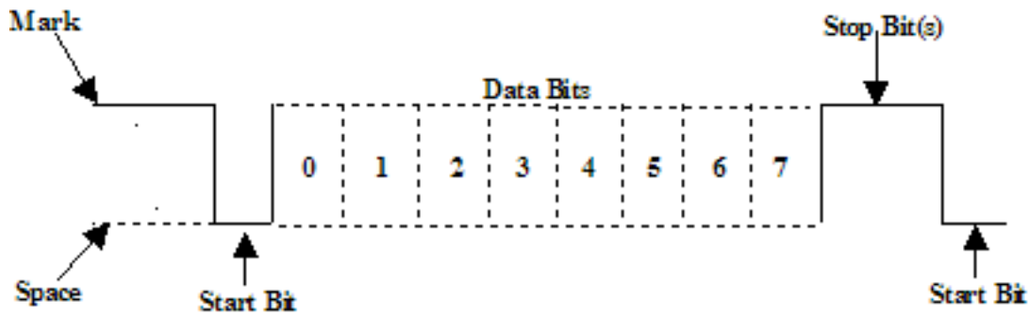
- Why then do we have problems getting XML into memory?

RS-232

- Anyone remember serial interfaces?



- You had to know the settings before it worked
  - Speed, stop bits and start bits



- Everything was 8 bits in fact most text was 7 bits and the top bit was parity, on the other hand parity was also another option

- Them were days!

Size mattered!

- It was less than 20 years ago when the World Wide Web started
  - Every image had to be compressed

- The art of programming was knowing how to read and write data into and out of binary

- Every programmer was familiar with bit-wise operators
  - << & ^ | ~ >>
  - We could all calculate in two's complement
  - We all knew Log_{10} 2 off by heart

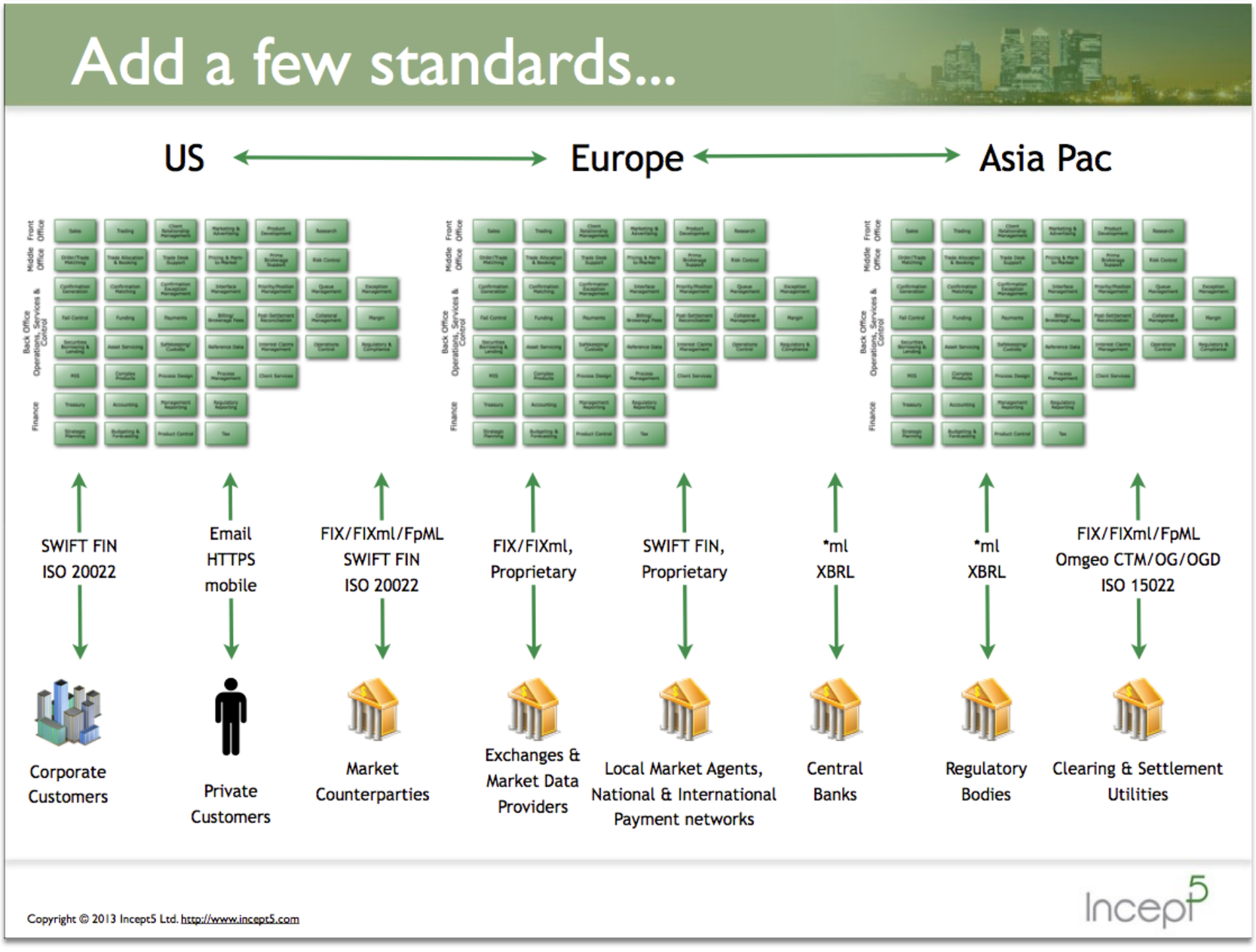
- Real programmers don't eat quiche!

The quiche eaters

- With Assembler, C and C++ we had to allocate the memory we used and free it when we'd finished
  - If we wrote the constructor then we knew exactly how much memory we were using - down to the byte
- With Java, VB, C# and other "quiche eater" languages everything is taken care of with memory management and garbage collection
  - We become very lazy, with machines getting faster and more memory to tend we forget about writing efficient code
- Take a complex derivative in XML, bind it to Java, stick it into a cache, distribute the cache, job done!
  - Volumes have got the better of us though

Last year…

- Last year a similar talk on "In-Memory Message & Trade Repositories"

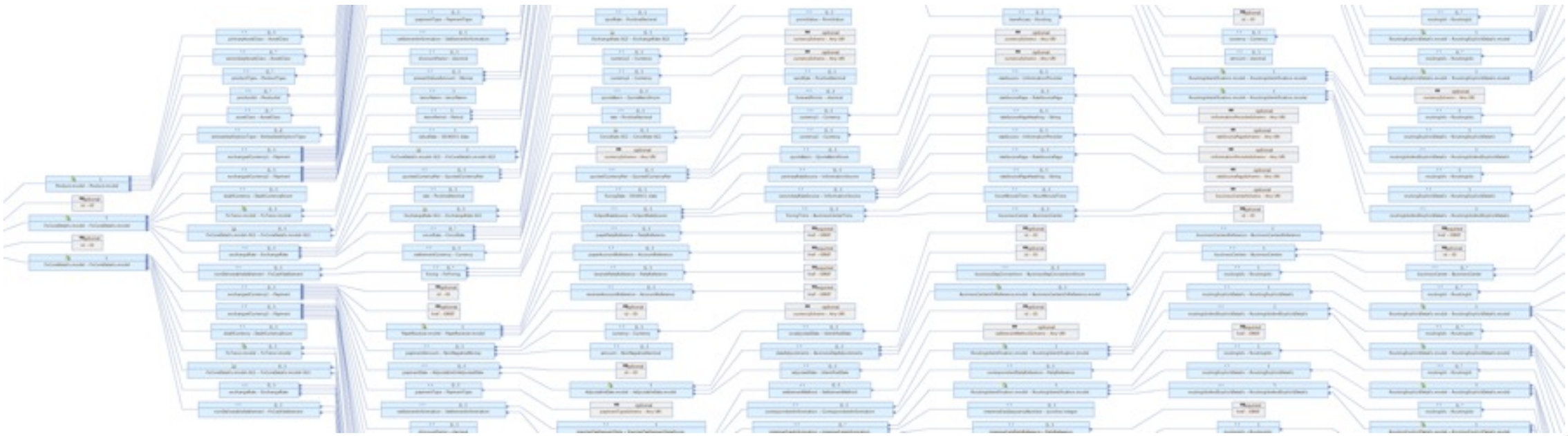


Back to last year's slides…

- An FX Swap
  - 14 Level of hierarchy
  - Over 3,000 elements

- To the right is the fill message zoomed out

- Below is the part in the box zoomed in a little...



Nasty XML

- The schema describes the worst-case but many of the simpler "contracts" (XML instances) are vastly simpler
- The FX swap on the right is pretty much all of the information needed to describe the contract
  - There are no options in this example

```
<trade>
  <tradeHeader>
    ...
    <tradeDate>2002-01-23</tradeDate>
  </tradeHeader>
  <fxSwap>
    <productType>FxSwap</productType>
    <nearLeg>
      <exchangedCurrency1>
        ...
        <paymentAmount>
          <currency>GBP</currency>
          <amount>10000000</amount>
        </paymentAmount>
      </exchangedCurrency1>
      <exchangedCurrency2>
        ...
        <paymentAmount>
          <currency>USD</currency>
          <amount>14800000</amount>
        </paymentAmount>
      </exchangedCurrency2>
      <valueDate>2002-01-25</valueDate>
      <exchangeRate>
        ...
        <rate>1.48</rate>
      </exchangeRate>
    </nearLeg>
    <farLeg>
      ...
    </farLeg>
  </fxSwap>
</trade>
```

ORM - OMG!

- It's not just memory that has issues with size and complexity

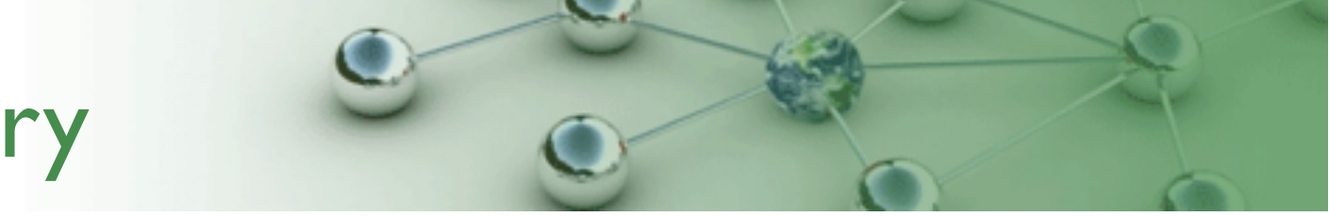
ORM - OMG!

- Object Relational Mapping (ORM) is sheer craziness!
- The ORM version of the FpML swap has well over 1,000 tables and a single join is several 'k' in size

- We could create new tables for each contract but that's what we started doing in 2000 and that didn't work
  - Many of these systems are what we have today and this is causing more and more pain
- ORM - Hibernate, JPA etc. was designed for simpler cases

Incept5

Chuck it into memory



- Either people have been listening to me or I've just been talking about what everyone's doing
  - I like to think it's partly the former because I tend to talk about things BEFORE they happen, not afterward otherwise you wouldn't be interested
- The problem with putting things into memory is cost
- It works really fast and most people tend to think it's as fast as you're going to get so just pay the money
- One client has 400 nodes with over 15TB of in-memory data
  - That's VERY expensive to run, several $million per year!

Let's take a step back

- Take a look at this XML, the bits in red are data, the rest is meta-data
  - The structure is also part of the metadata
- The actual information here is relatively small
- Pre-XML in the 90s we'd have stored this in a much more efficient way
- But without XML Schema we didn't have the standards we have toady

```
<trade>
  <tradeHeader>
    ...
    <tradeDate>2002-01-23</tradeDate>
  </tradeHeader>
  <fxSwap>
    <productType>FxSwap</productType>
    <nearLeg>
      <exchangedCurrency1>
        ...
        <paymentAmount>
          <currency>GBP</currency>
          <amount>10000000</amount>
        </paymentAmount>
      </exchangedCurrency1>
      <exchangedCurrency2>
        ...
        <paymentAmount>
          <currency>USD</currency>
          <amount>14800000</amount>
        </paymentAmount>
      </exchangedCurrency2>
      <valueDate>2002-01-25</valueDate>
      <exchangeRate>
        ...
        <rate>1.48</rate>
      </exchangeRate>
    </nearLeg>
    <farLeg>
      ...
    </farLeg>
  </fxSwap>
</trade>
```

Java Binding

- XML is really fast when bound to Java but it's often even more bloated…

```
<Row>
        <Name>Tim Cook</Name>
        <CardNumber>4924-7264-1264-8532</CardNumber>
        <ExpiryDate>04/09</ExpiryDate>
        <Amount>12250</Amount>
        <Currency>USD</Currency>
        <TransactionDate>2006-09-16</TransactionDate>
        <Commission>1.3</Commission>
        <VendorID>67434435</VendorID>
        <Country>US</Country>
</Row>
```

```
public class Row extends biz.c24.io.api.data.ComplexDataObject {
    private java.lang.String name;
    private java.lang.String cardNumber;
    private java.lang.String expiryDate;
    private double amount;
    private boolean isamountSet;
    private java.lang.String currency;
    private java.util.Date transactionDate;
    private double commission;
    private boolean iscommissionSet;
    private long vendorID;
    private boolean isvendorIDSet;
    private java.lang.String country;
```

Complex Objects

- Every Java String is minimum 48 bytes in size, whether you're on the heap or not objects get fragmented in memory

```
public class Row extends biz.c24.io.api.data.ComplexDataObject {
    private java.lang.String name;
    private java.lang.String cardNumber;
    private java.lang.String expiryDate;
    private double amount;
    private boolean isamountSet;
    private java.lang.String currency;
    private java.util.Date transactionDate;
    private double commission;
    private boolean iscommissionSet;
    private long vendorID;
    private boolean isvendorIDSet;
    private java.lang.String country;
```

Binary models

- Many of the network-critical standards like telcos and older standards are defined in binary

- The following is an extract from a binary standard called RADIUS used by Telcos
  - They use bit-fields to indicate the presence of data (or not)

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |  Length       |             Vendor-Id
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     Vendor-Id (cont)            | Vendor type   | Vendor length |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Attribute-Specific...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Multiple subattributes MAY be encoded within a single Vendor-
Specific attribute, although they do not have to be.
```

The RADIUS Model

Document Root (Read Only)

Component	Type	Cardinality	Size
Document Root	Document Root (local)		16 - *
Packet File	Packet File	1	16 - *
packet	Packet	1..*	16 - *
code	code (local)	1	0
identifier	identifier (local)	1	0
length	length (local)	1	0
authenticathor	authenticator	1	16
attribute	attributes	1..*	0 - *
type	type (local)	1	0 - *
length	attribute length	0..1	0 - *
value	value (local)	0..1	0 - *
user	user (local)	1	0
user-id	Unbounded Byte Type	1	0
vendor	vendor (local)	1	0 - *
vendor-id	Unsigned 4-byte Word	1	0
attributes	attributes	0..*	0 - *
type	type (local)	1	0 - *
length	attribute length	0..1	0 - *
value	value (local)	0..1	0 - *
user	user (local)	1	0
vendor	vendor (local)	1	0 - *
Calling-Station	Calling-Station-Id (local)	1	0
user-passworc	user-password (local)	1	0
CHAP-Passwor	CHAP-Password (local)	1	16
NAS-IP-Addre	NAS-IP-Address (local)	1	0
NAS-Port	NAS-Port (local)	1	0
Calling-Station-Id	Calling-Station-Id (local)	1	0
user-password	user-password (local)	1	0
CHAP-Password	CHAP-Password (local)	1	16
NAS-IP-Address	NAS-IP-Address (local)	1	0
NAS-Port	NAS-Port (local)	1	0

Binary XML?

- What we got was an incredibly small derivative message based on FpML
- Rather than being the implementation it was simply our end-goal
  - If we could get the XML version down to this size we'd achieved our goal
- Debugging this stuff was like stepping back to the 80s

```
jd-server:Radius TestData jdavies$ hexdump -C radius.dat
00000000  01 02 00 74 ea d5 7c 62  1f d0 f6 fe a3 bf 36 4c  |...t..|b......6L|
00000010  35 25 e5 8c 1a 17 00 00  28 af 01 11 32 33 34 34  |5%......(...2344|
00000020  35 37 30 36 32 37 38 38  35 33 36 01 11 32 33 34  |57062788536..234|
00000030  31 35 39 30 36 32 35 38  38 35 33 36 1f 11 33 35  |159062588536..35|
00000040  33 34 32 31 30 32 30 39  34 35 35 36 38 5e 0e 34  |3421020945568^.4|
00000050  34 37 30 30 34 31 38 38  36 37 33 1a 0d 00 00 28  |47004188673....(|
00000060  4e 97 57 a8                                       |N.W.|
```

Old + Modern

- Now we could combine binary and Spring and run it on Java 8 😀

```
<filter input-channel="filter-message-channel"
output-channel="process-message-channel"
ref=”payload” method=”isAbcSet"/>
```

```
<filter input-channel="filter-message-channel"
output-channel="process-message-channel"
expression="payload.versionId == 5"/>
```

Lazy getters

- Going back to our classic bound Java object the getters simply returned the object

```
public class Row extends biz.c24.io.api.data.ComplexDataObject {
    private java.lang.String name;
    private java.lang.String cardNumber;
    private java.lang.String expiryDate;
    private double amount;
    private boolean isamountSet;
    private java.lang.String currency;
    private java.util.Date transactionDate;
    private double commission;
    private boolean iscommissionSet;
    private long vendorID;
    private boolean isvendorIDSet;
    private java.lang.String country;
```

  - getNumberOfElements() { return numberOfElements; }

- Now we have to find the value in the byte[] and return the calculated value

```
public int getNumberOfElements() {
    return ((data.get(2) & 0x18) >> 3);
}
```

- Performance is about the same but we only use about 1/25th of the memory

SDOs

- SDOs or Simple Data Objects are basically Java Binding into a compact binary codec - From any XML format to binary

- We analyse the data model (or XML schema) not just the instance data so can do things like…
  - Reducing the 7 days of the week to just 3 bits
  - Commonly used Strings become lookups into a static table (1 or 2 bytes)
  - Currencies for example only need 1 byte
  - Date/Time with timezone can be stored in 6 bytes

- Bit-fields are compacted resulting in excellent compaction-ratios
  - Getters calculate the offset on the fly, mask and shift the data and return it

- There is NO change to the getter API between standard binding and SDOs

Standard Java Binding

```
<resetFrequency>
    <periodMultiplier>6</periodMultiplier>
    <period>M</period>
</resetFrequency>
```

- JAXB, JIBX, Castor and standard C24 generate something like …

```
public class ResetFrequency {
    private BigInteger periodMultiplier; // Positive Integer
    private Object period;               // Enum of D, W, M, Q, Y

    public BigInteger getPeriodMultiplier() {
        return this.periodMultiplier;
    }
    // constructors & other getters and setters
```

- In memory - 3 objects - at least 144 bytes
  - The parent, a positive integer and an enumeration for Period
  - 3 Java objects at 48 bytes is 144 bytes and it becomes fragmented in memory

Java Binding with SDOs

```
<resetFrequency>
    <periodMultiplier>6</periodMultiplier>
    <period>M</period>
</resetFrequency>
```

- Using C24 SDO binary codec we generate …

```
ByteBuffer data;    // From the root object
public BigInteger getPeriodMultiplier() {
        int byteOffset = 123;  // Actually a lot more complex
        return BigInteger.valueOf( data.get(byteOffset) & 0x1F );
    }
    // constructors & other getters
```

- In memory - 1 byte for all three fields
  - The root contains one ByteBuffer which is a wrapper for byte[]
  - The getters use bit-fields, Period is just 3 bits for values D, W, M, Q or Y

SDOs - So far*

- ISDA's sample Interest Rate Derivative (vanilla swap) is 7.4k
  - We randomised a few fields and created a few million for testing

- Zipped they are average 1,547 bytes
  - 1 million on disk require 1.5GB and takes 200 seconds to read/decompress
  - Parsing at 20k/sec would add another 50 seconds and need a **lot** of memory

- In memory they are roughly 25k in size (in 2-400 objects)
  - It was difficult to fit 400k into 10GB of RAM - Lots of full GCs too

- With SDOs the average size was just 442 bytes
  - It took 9 seconds to read and parse 1 million from disk (SSD)
  - It took 415ms to search through all 1 million IRSs in memory (brute force)
  - 20 million fully parsed IRSs comfortably fit in 10GB of RAM

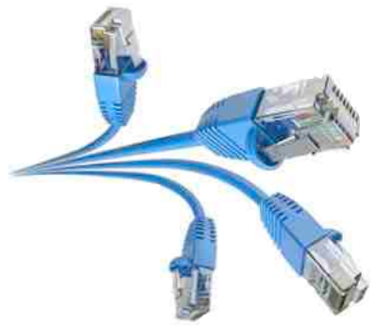
- Total saving on memory with FpML is roughly 50 times

* Tests were run on Java 1.7.0_55 on a MacBook Pro (2.7 GHz Intel i7) on a single core, we continue to improve these figures

Examples

- Take 60GB of XML, bind it to Java and we now have 200+GB
  - Now we need a 4 good machines (64GB) or 8 if we want high availability (HA) to host this in memory

- The average size of the messages means machine synchronisation over the network is slow
  - Each message needs several network (IP) packets (per 1 MTU)



- With binary XML we now need under 5GB to store the same data - OK 2 machines for HA but "small" (cheaper) machines
  - Now each message is smaller than the MTU size so network synchronisation is much faster too

CPU cache fragmentation

- If your entire object is in one block in memory then the entire object is very likely to hit the CPU cache - as is the next one
- Serialize a complex bound Java object and you're serialising hundreds of objects with metadata (to name the objects)
- Serialize our binary XML and we serialize the object ID, the size and the byte[]
  - Use NIO and we can serialise (and de-serialize) a million FpML trades to/from disk in seconds
  - Use SDOs with SSDs can give better performance than distributed RAM
  - You can now get 20-50TB SSD drives from companies like Pure Storage
  - No network means you can take the P out of CAP theorem
- This sort of performance can change your design and architecture

Reference Data - FIGI example

- FIGI - Financial Instrument Global Identifier (from Bloomberg)
  - Run through the OMG, hoping to become an ISO standard
  - Supported by 28 global institutions including…
    - NASDAQ, NYSE, FINRA, Paribas, State Street, Morgan Stanley, Markit, IDC, Moody's etc.

- 185 million identifiers (so far) roughly 250 bytes each
  - Totalling about 50GB of raw data (CSV), a few hundred GB in a database

- With SDOs we've got this down to under 12GB in-memory

- So why both to compact it when it'll easily fit on a database?
  - Well it changes daily, how are you going to keep your world-wide databases in synch?

- Reference data is key to system and needs to be held locally

Caching

- We've released white papers on SDOs with several popular caching technologies
  - GemFire
  - GigaSpaces
  - Ehcache
  - HazelCast
  - Coherence
  - GridGain

EHCACHE

GEMFIRE®

GIGASPACES

GridGain
REAL TIME BIG DATA

hazelcast

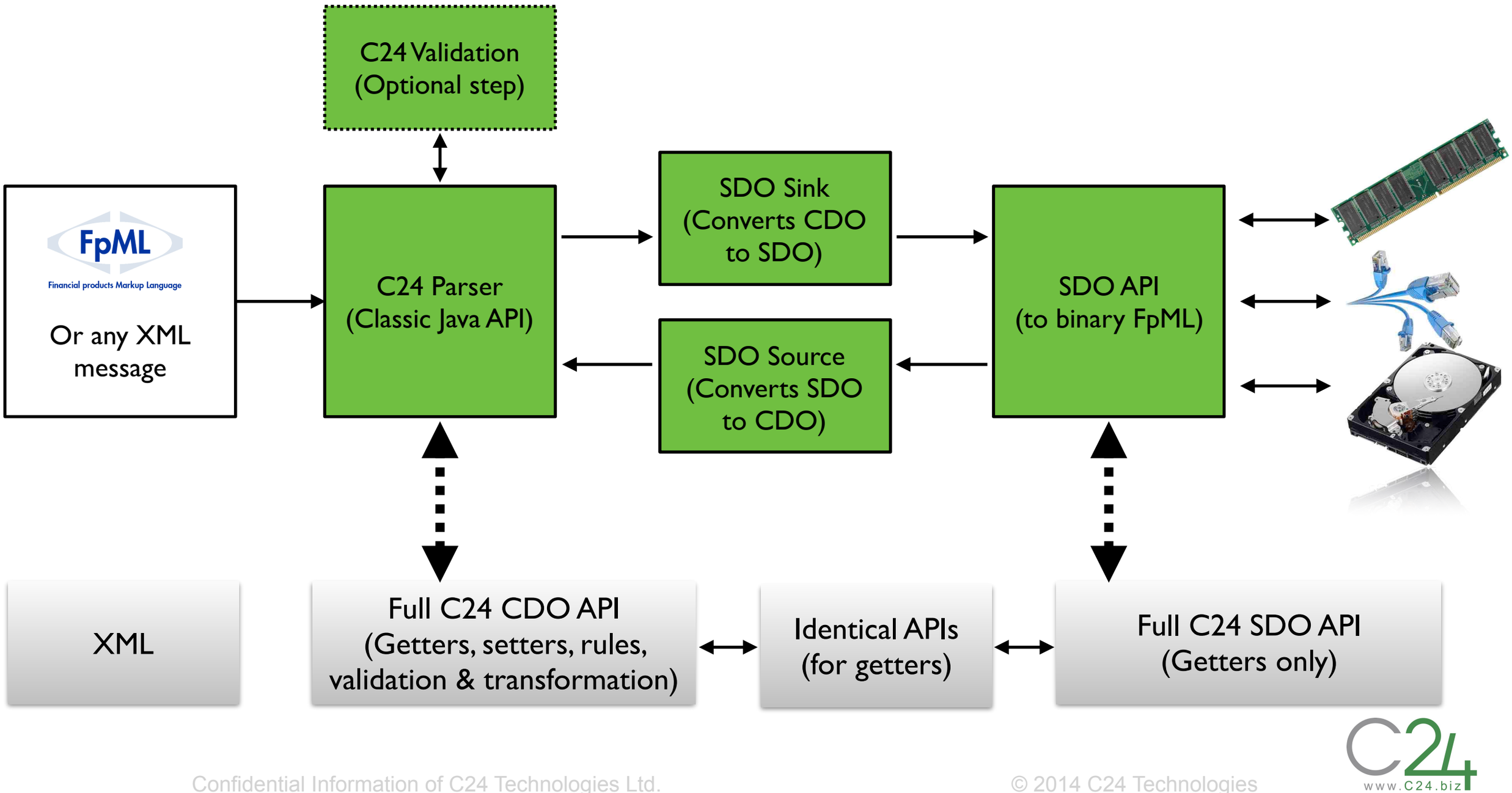
ORACLE®
Coherence

- C24 SDOs combined with these caches improved storage capacity from 22 to 65 times using FpML (an XML message type) over "classic" Java-Bound objects in the same store

- Even non-Java technologies like Redis and Riak run fantastically faster with SDOs, they have less data to manage

How it works

~5-8k	10-25k	**Size**	< 500 bytes
10k/sec	~1m/sec	**Performance**	~1m/sec



Demo

20 million fully parsed FpML messages on a laptop and search ANY field at over 2 million a second