Analytics Architecture for Equity-Linked Derivatives

Implementing Derivative Valuation Models

February 2006

Dr. Andrew Ferraris
Deutsche Bank
Global Markets Equity
Quantitative Products

Overview

- Aims & Context
- Data
- Implementation
- Environment

What are the aims of an analytics framework ?

- A framework for algorithm development
  - New derivative valuation algorithms are added to it. Frequently-used functionality lives in the framework
  - It provides a common top-level API to client apps
- Plan for it to live a long time – indefinitely
  - Cost of replacement (of poor design) very (unacceptably) high
    - The suite gets built in to business processes
    - Evolution not revolution
  - We design for the things we know we will need
  - And to process data we know about today
  - We accept that there are unforeseeable requirements & unknowable data

What are the aims of an analytics framework ?

- Seen as a piece of software within the total software universe of (say) an investment bank, derivatives analytics is a real hot-spot
  - Usual strategy applies : encapsulation
- We would like it to be re-useable
  - Loose coupling
    - Core without links to data sources or sinks. Distinction between library and application.
    - Client apps get database connectivity and lazy fetching by deriving from base classes in the core
- Product coverage ranges from vanilla to structured
  - There are overheads associated with the *capacity* to handle structured
    - A motivation for “Quant Library Lite” (with its own development overheads)

What are the aims of an analytics framework ?

- And is it really that general-purpose ?
  - S/w companies need to support many clients with differing or conflicting requirements
  - The quantitative research function of an investment bank supports just a few business lines within one organisation
  - Focus on equity- and equity-linked derivatives
    - Need to support some IR and credit, but not optimised for them

Data Capture

- Before we can price anything we must capture (model) all the information we will need
- Fortunately, there are just a few big categories of such data, and the rule is: don't get them confused !
  - Static data
  - Market data
  - Control information
  - Results
- Not confusing them is really important when (generally the case in non-trivial apps) the instantiation of static data is separated from its processing

Data Capture

Static Data

- Things which are true by definition, which do not need to be observed or inferred (i.e. calibrated)
  - The definitions of securities
    - options, stocks, bonds, futures, baskets, indices, etc etc
  - Business day calendars per country, exchange
  - Quoting conventions (for rates)

Data Capture

Static Data

- Our structures can get arbitrarily complex
  - An equity
  - A bond
  - An option on a stock
  - An option on two stocks
  - A basket
  - An option on a convertible bond (asset swap)
  - An option on a CPPI strategy
    - which has a risky and a ‘riskless’ underlying

Data Capture

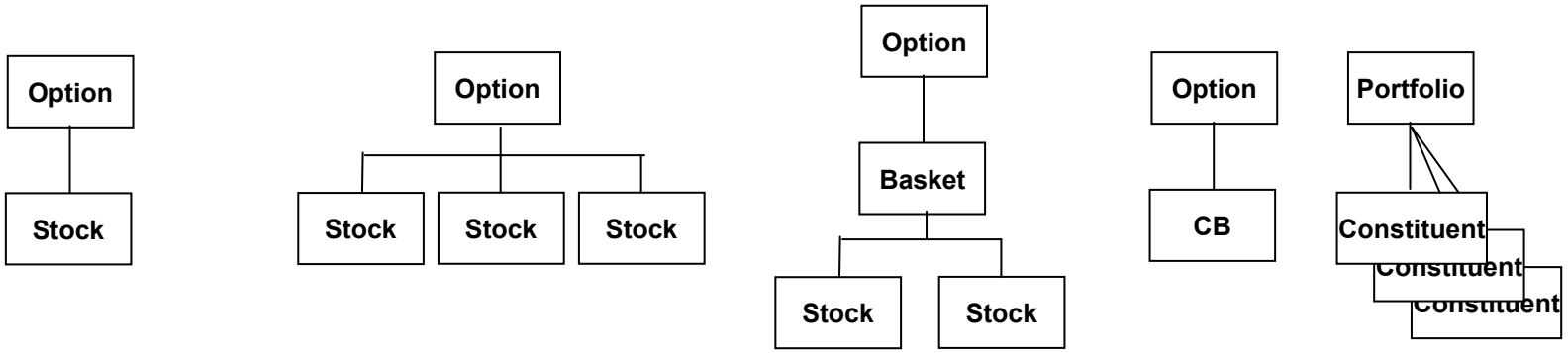
Static Data

- A spread between two baskets
  - which may overlap or differ only in their weightings: the same stock *can* appear twice!
- An option on a future (on an underlying index)
- An option on many underlyings (Altiplano, others, up to 50 underlyings)
  - which may be equities, commodities, …
- A portfolio containing all of the above

Data Capture

Static Data

- We do not want to restrict the data we can capture, whether or not we can price the structure (today)
- A tree structure is natural (inevitable ?)



- This is our representation for calculating with. The same stock or basket (e.g.) can occur many times, so we will retrieve the definitions from a repository and build these trees on demand.

Data Capture

Static Data

- We *combine* security definitions when we calculate: drill down through underlyings
- The algorithm has access to the complete tree
- There is only one definition of a stock (convertible bond, etc), not many, in our repository
- The data items needed to describe a single security are not known when we specify the architecture
  - Addressed by class hierarchies or otherwise: name/value pairs, XML

Data Capture – Digression 1

What is a class hierarchy for (equity) derivatives ?

- There are an *awful* lot of different structures
- With very few truly *universal* features
  - An identifier
  - A currency in which the price is denominated
  - A list (possibly empty) of underlyings
  - A maturity (occasionally invalid)
- And many recurring features
  - A strike
  - Fixings
  - A barrier or time-varying barrier
  - Coupons

Data Capture – Digression 1

What is a class hierarchy for equity derivatives ?

- Any given structure contains none, some or many of the recurring features
- And many unique ones
  - Var Swap : List of strikes in replicating portfolio
  - Lookback : Min or max achieved to date
  - CB : Make Whole definition
  - CPPI : Minimum investment level (zero in a classical CPPI)
  - Generic : Script text (or maybe stack of operations)

Data Capture – Digression 1

What is a class hierarchy for equity derivatives ?

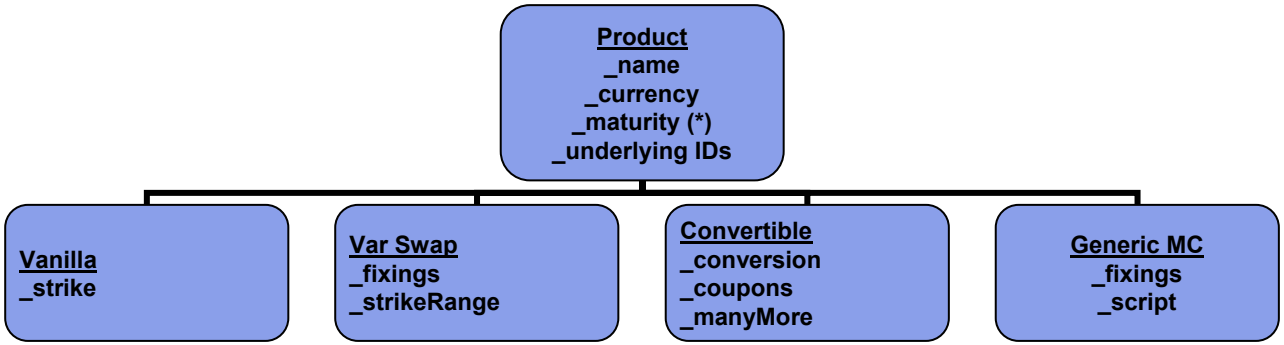
- One (very) big class
  - It really does get massive: hundreds of data members
  - It's simple: you can mix-and-match
  - You lose self-documentation (and self-description)
    - But you could partially regain it with many constructors

Product
_name
_currency
_maturity
_underlyings
_fixings
_strike
_barrier
_coupons
_script
_conversion
_redemption
_more
_andmore
_and_more

Data Capture – Digression 1

What is a class hierarchy for equity derivatives ?

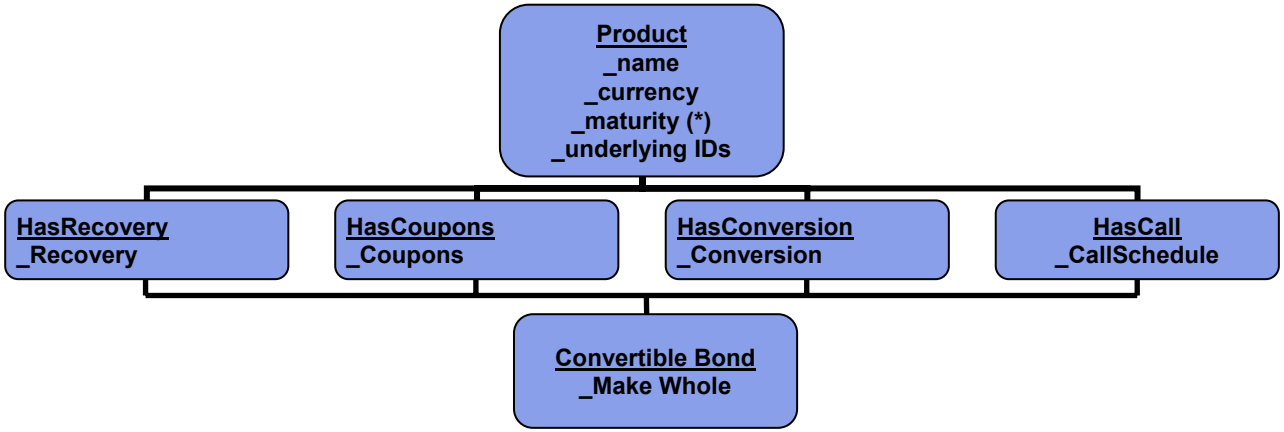
- A flat class hierarchy with containment of simple objects for recurring data items



Data Capture – Digression 1

What is a class hierarchy for equity derivatives ?

- A diamond class hierarchy with inheritance for recurring data items



Data Capture – Digression 2

Self-description

- For various reasons …
  - MC splitting
  - Report distribution (e.g. large greek reports, projections)
  - Debugging
- … it may become desireable to serialise static and market data
  - We have used Boost serialisation, and had real problems with it
    - (and Data Synapse for the distribution, and had more problems with that)
  - Intrusive : you have to maintain the serialisation operators by hand ☹

Data Capture – Digression 2

Self-description

- Objects may be self-describing
  - Avoids writing serialisation code by hand
  - Prefer objects to contain exactly the data they need
    - Suggests mechanism for registering data members at construction with a base which handles the serialisation
    - Other schemes are possible: XML / FpML
  - *Expensive to retro-fit to an existing class hierarchy*

Data Capture

Market Data

- Things which need to be observed (or calibrated?)
  - Yield curves
  - Implied volatilities
  - Dividend estimates
  - Correlations
  - Possibly other data which we cannot forsee when we design the framework
    - Variance swap term stucture
    - Correlation skew & term structure (→ a correlation hypercube)
    - Calibrated parameters for the model of your choice (jumps, vol of vol etc). This grows without limit.

Data Capture

Market Data

- Since our static (security) data structures may grow without (designed-in) limit, the amounts of market data required by a pricing problem is likewise not restricted
- So our market data repository must be open-ended in the *amount* and *types* of data it can store
  - So a container of <template> containers

Data Capture

Controls

- Often overlooked: we need to tell our pricing application how we want it to behave, as well as giving it all the data it needs
  - Tell it *what* to calculate
  - Tell it *how* to calculate it
- There are types of control information, categorised by whether they depend on the product, model or method

Product

- Delta
- Theta
- Implied Recovery
- [Variants (e.g. quanto)]

Model

- Parameter sensitivities (short rate mean reversion, Vega)
- [Parameter overrides ("Volatility=")]
- Perturbation sizes

Method

- Time steps
- Spatial mesh points
- FD theta
- FD spot list
- MC paths
- MC seed
- MC precision

Data Capture

Controls

- Portfolios present a new problem.
  - They can have unrelated product types as their constituents
    - A vanilla swap and a callable note
  - They can have closely related ones, but which nonetheless require different control information
    - American options with significantly different maturities
- Different instances of control information are needed per portfolio constituent
  - More generally, per item in the static data tree
  - Keyed on the unique name of the security

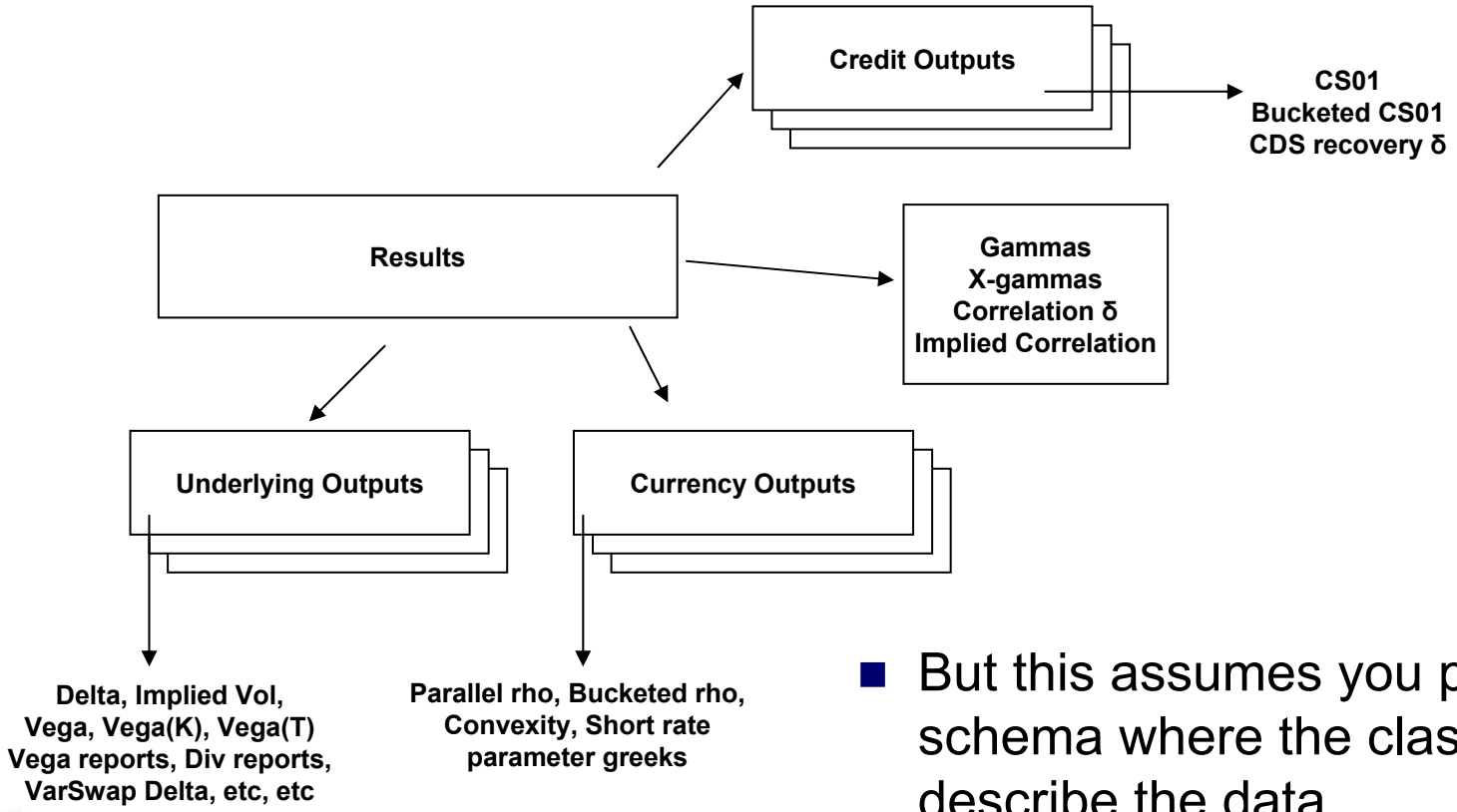
Data Capture

Results

- We can identify different categories
  - Items of which there is exactly one per structure
    - price, theta
  - Items of which there is exactly one per underlying in the structure
    - delta, vega, (gamma), nu (div risk), more
  - Items of which there is exactly one per currency in the structure
    - rho, convexity
  - Items of which there is one per name pair
    - gamma
- Based on the keys needed to label them

Data Capture
Results

- Clearly suggests a data structure



- But this assumes you prefer a schema where the class does describe the data

Some Issues in the Core Analytics Library

Greeks

- Hundreds of models times dozens of greeks
  - (At least one per type of market data)
- Equals a combinatorial explosion
- And greeks by finite difference estimates (i.e. perturbation)
  - Built into the core analytics or external ?
- Arbitrage-freeness in the perturbed objects
  - They can self-test, but this might not be cheap

Some Issues in the Core Analytics Library

Market Data Management

- Easy if the data are uncoupled
  - Data requiring calibration are coupled
    - Interest rate models
    - Local vol.
    - Stoch. vol.
- Otherwise
  - Objects may changed or deleted
  - Objects may be perturbed during a greek calculation
- This is only an issue if market data objects ‘live a long time’
  - But they will do so either in the core analytics or client apps
  - So take the complexity in the core else is it duplicated

Some Issues in the Core Analytics Library

Market Data Management

- What are primary data ?
- Are some types of data more fundamental than others ?
  - Rates depo, futures, swaps ⇒ discount curve
  - Credit CDS rates and discount curve ⇒ hazard rates
  - Equity ("most-derived")
    - Credit and Implied vol arbitrage (dividend assumptions)
    - Credit, dividend model, IR model, IV, spot ⇒ local vol, stoch vol
- When we calculate delta (for example) we typically mean under constant IV. So re-calibrate dependent LV etc objects. Likewise CS01, rho etc perturbation implies re-calibrate (e.g.) LV
  - So the calibration has to be stable under small perturbations !!

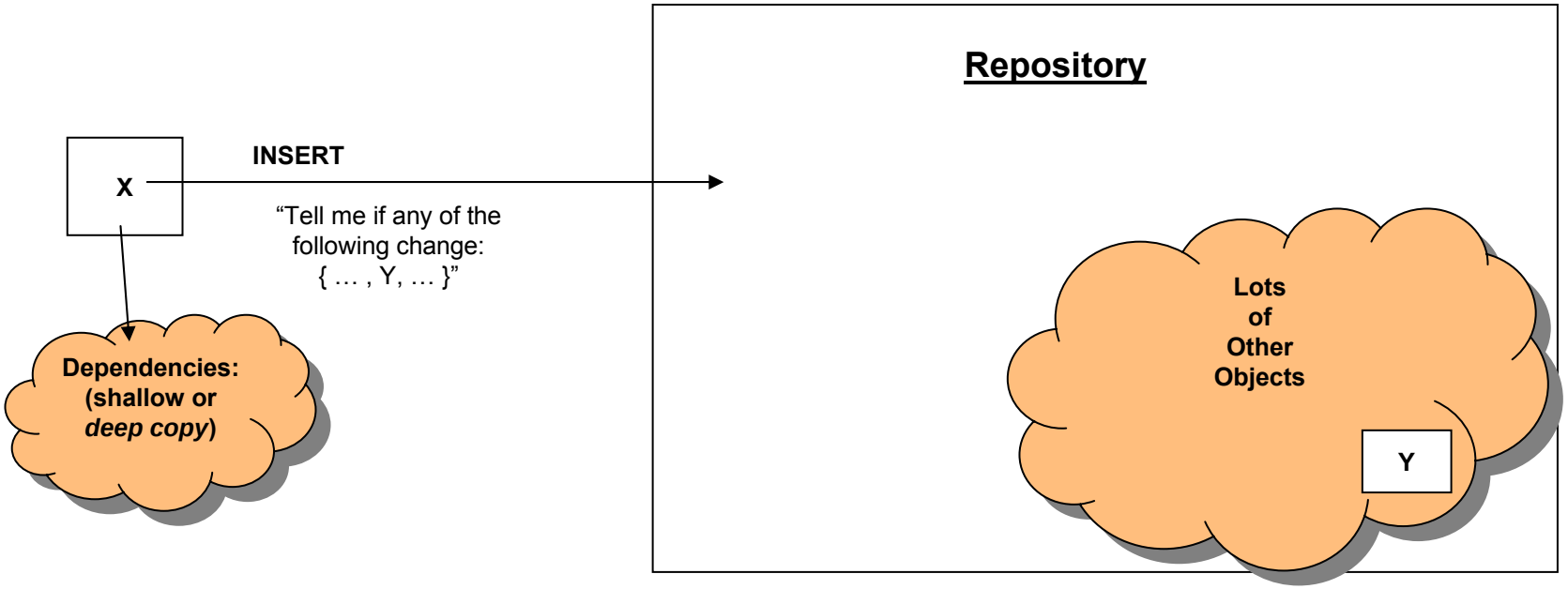
Some Issues in the Core Analytics Library

Market Data Management

- Many objects each depending on several others
- Subject / Observer pattern
  - Mediated by the repository

Some Issues in the Core Analytics Library

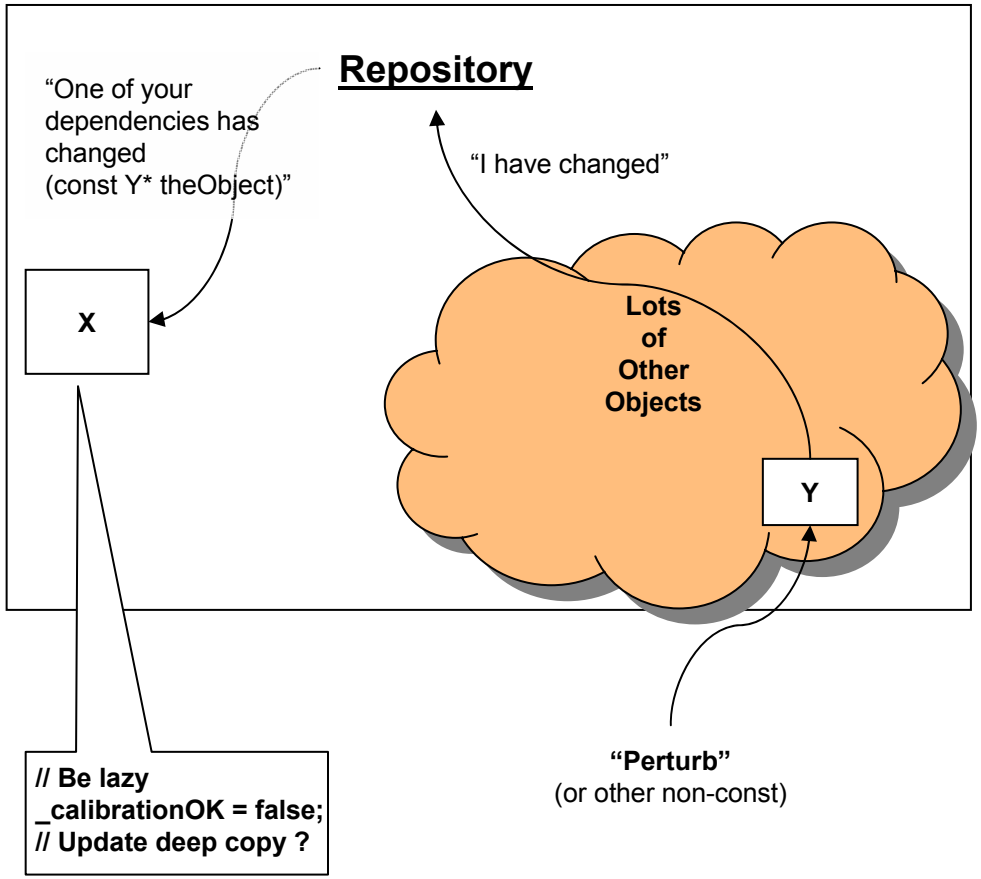
Market Data Management : Subject / Observer



Some Issues in the Core Analytics Library

Market Data Management : Subject / Observer

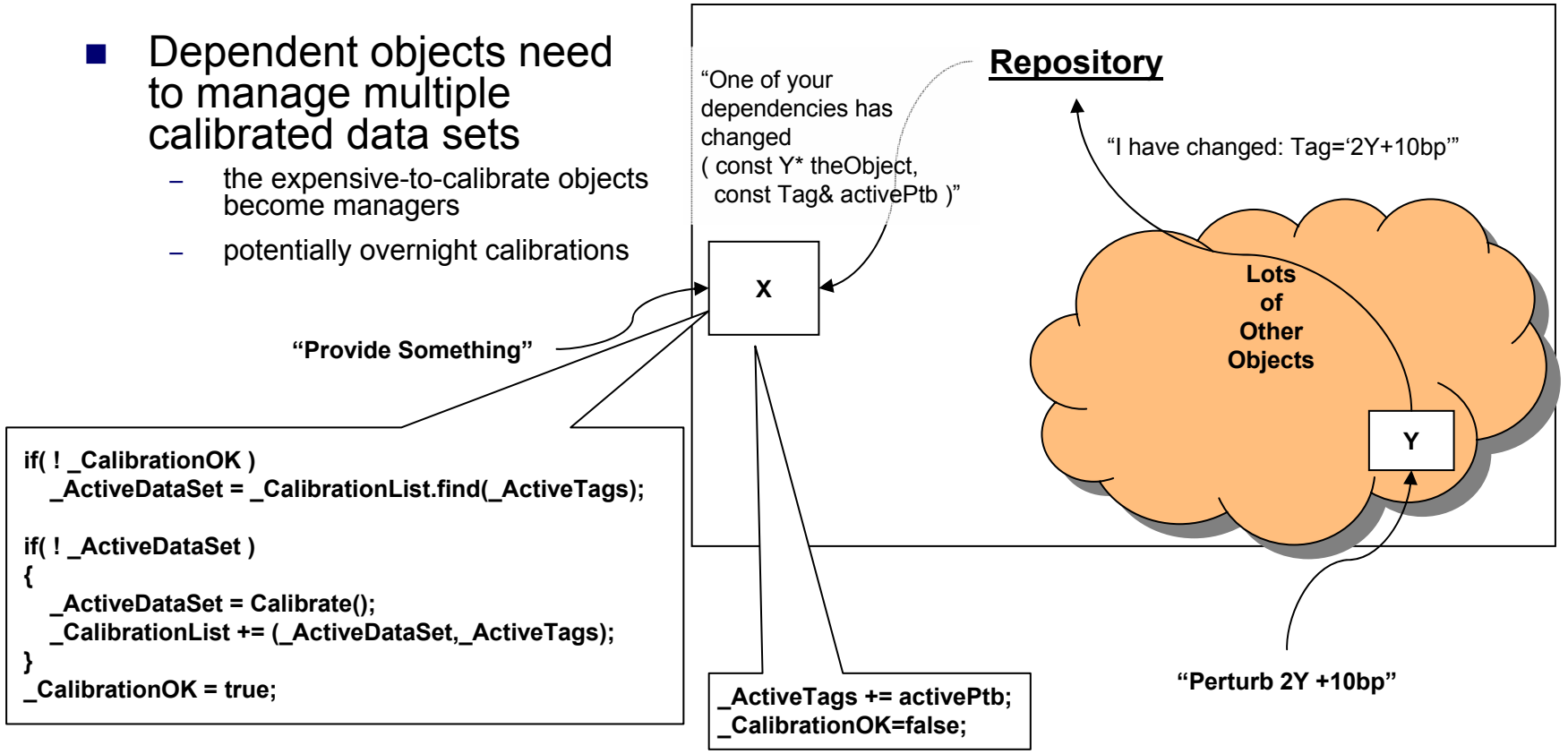
- This needs to be configureable
- Must be able to switch it off !
- Y::~Y() is an extreme case
  - (not const. Try calling delete on a const Y*)



Some Issues in the Core Analytics Library

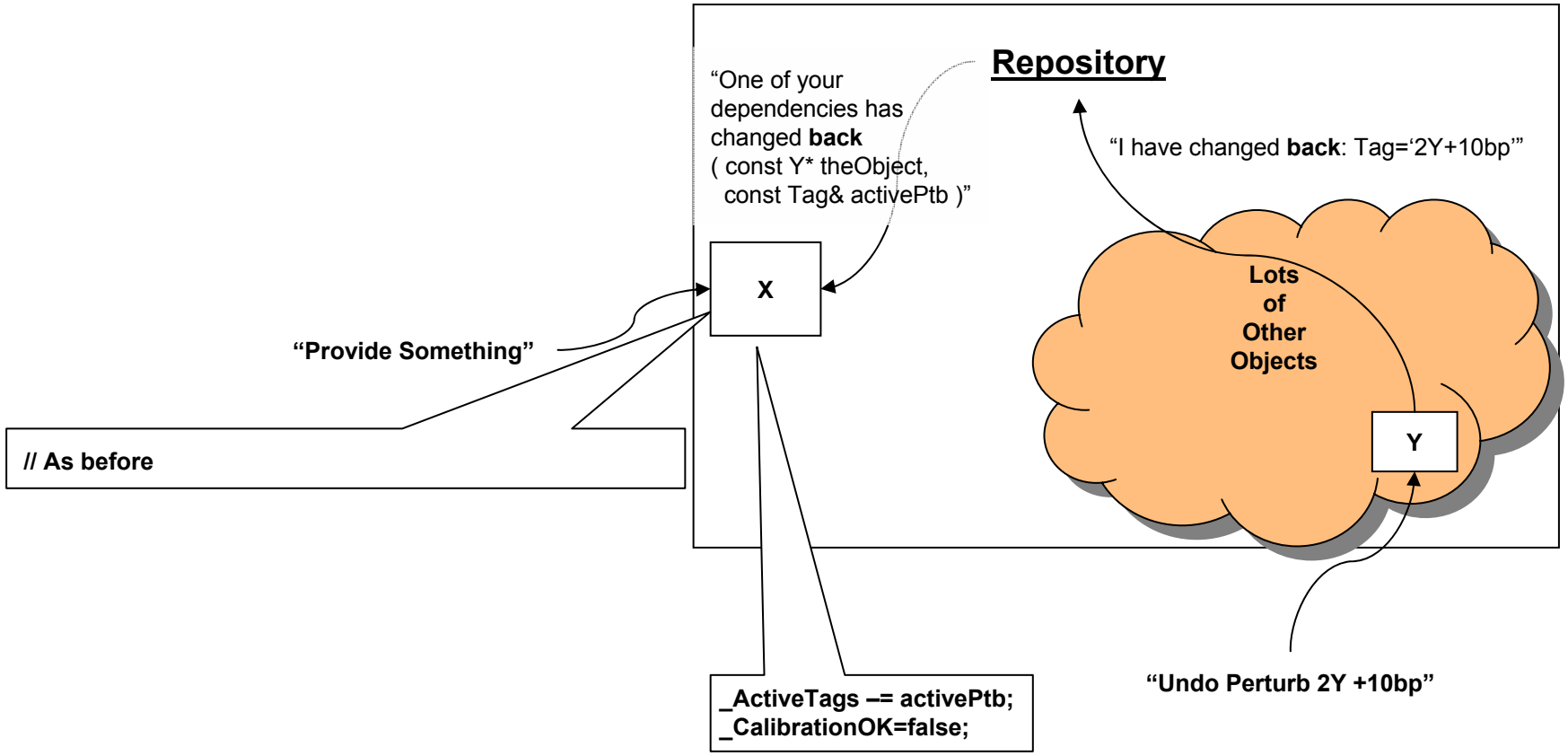
Market Data Management : Subject / Observer

- Dependent objects need to manage multiple calibrated data sets
  - the expensive-to-calibrate objects become managers
  - potentially overnight calibrations



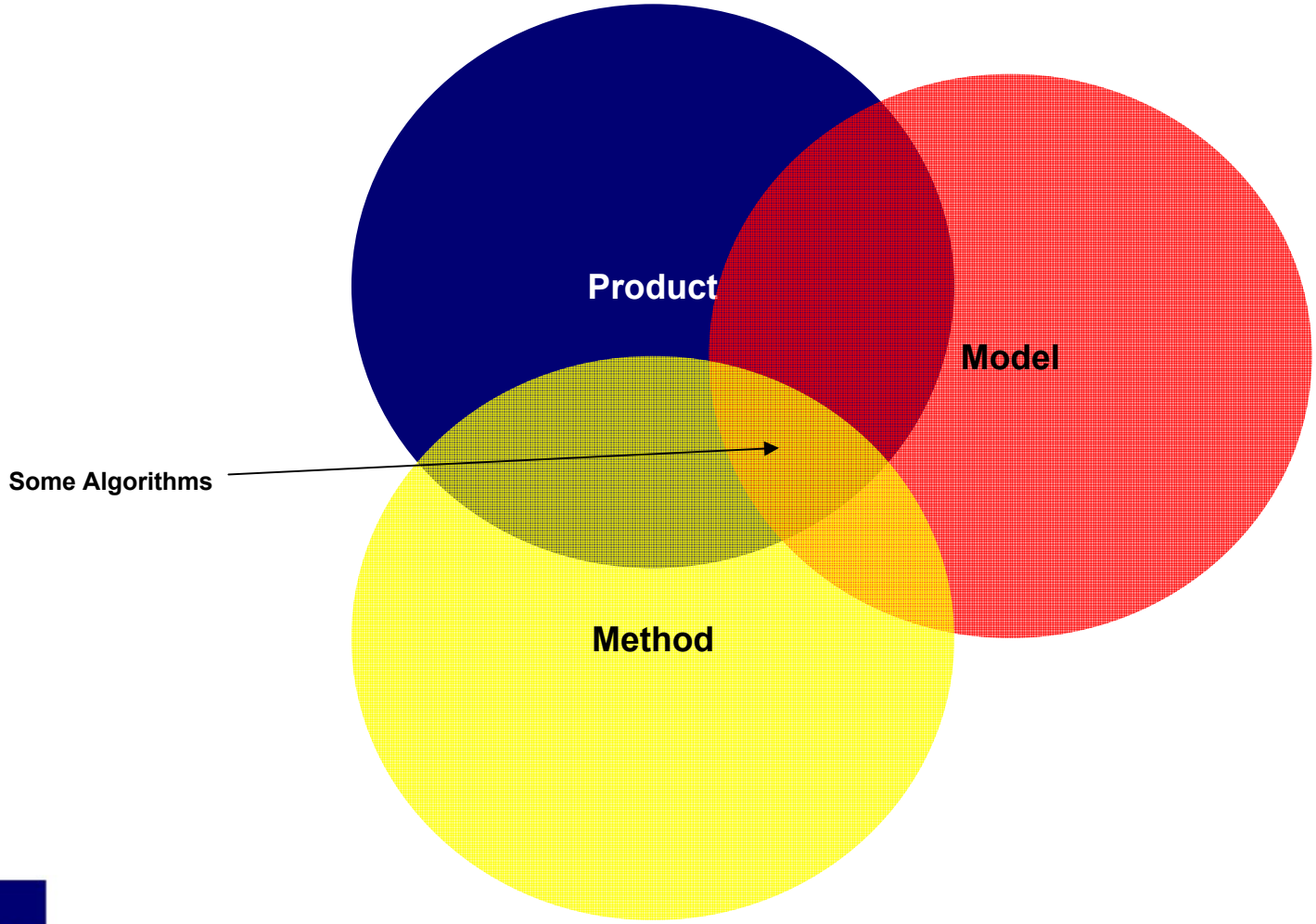
Some Issues in the Core Analytics Library

Market Data Management : Subject / Observer



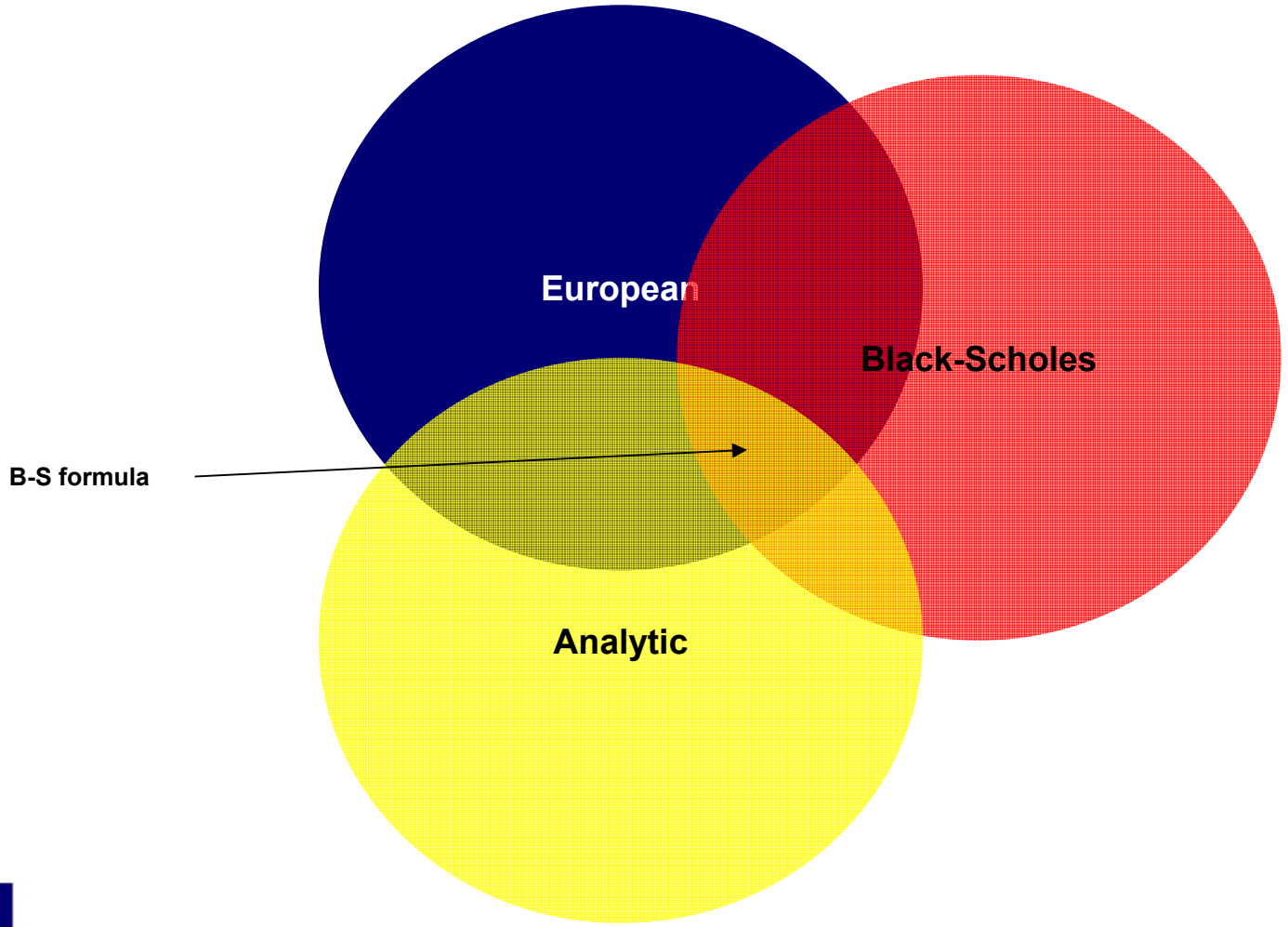
Some Issues in the Core Analytics Library

Product, Model and Method



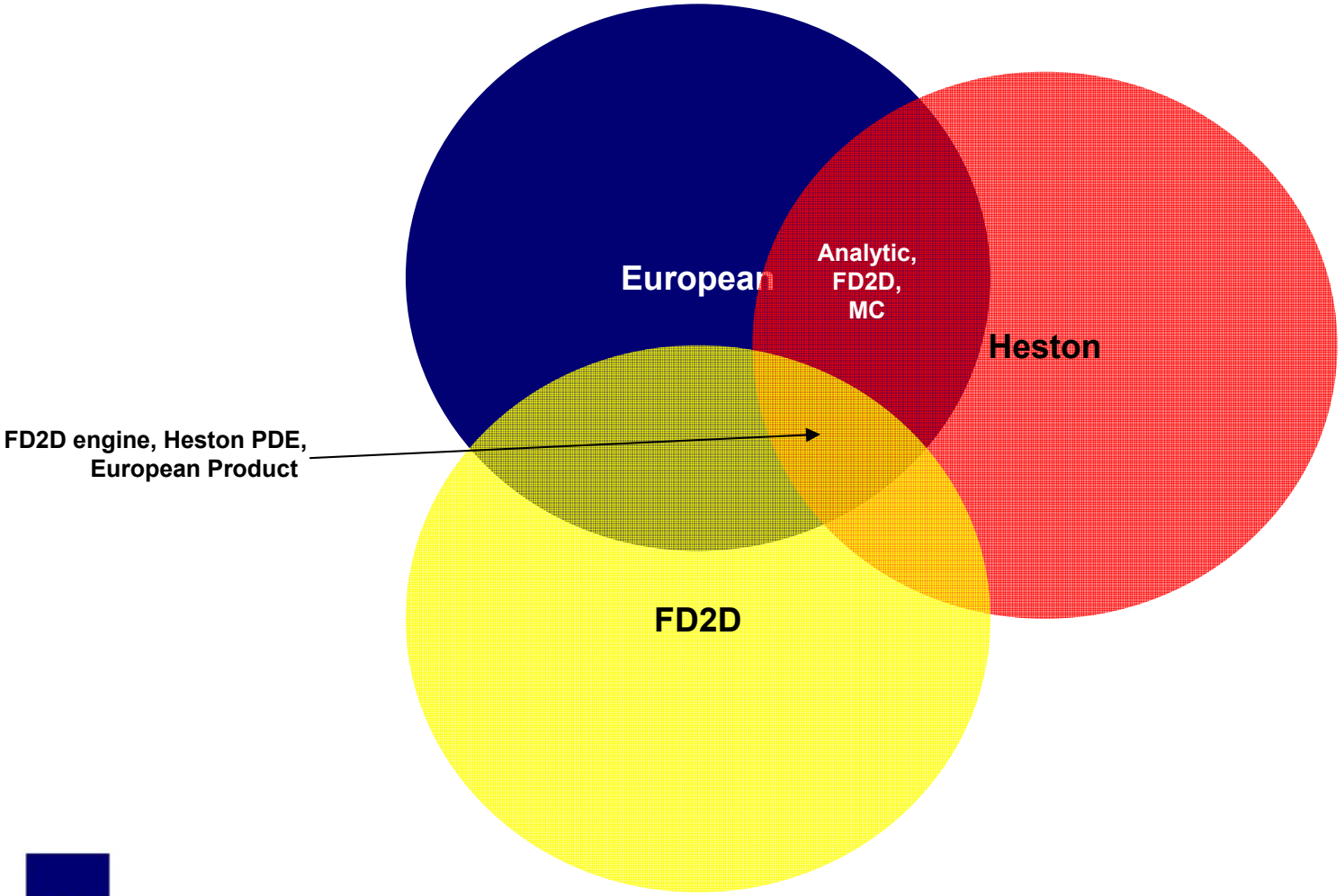
Some Issues in the Core Analytics Library

Product, Model and Method



Some Issues in the Core Analytics Library

Product, Model and Method



Some Issues in the Core Analytics Library

Product, Model and Method

- Why is this useful ?
- We request that the analytics suite process a given product [under a given model [, using a given algorithm]]
- Because we find that the items of market data which are retrieved from the repository depend on the *model*
  - So we do not need to aggressively query the repository for every possible market data type
  - Which helps client applications which implement lazy fetching, a lot !
    - The rules for deciding on which data are needed occur in only one place, not in each client application. This is very valuable.
- The product may come with an implicit default model request, often B-S
  - Sometimes not. Eg. cliquets in stoch. vol., barriers in local vol, …
- New algorithms are *registered* with their corresponding model
- The model object accepts the request if it has at least one algorithm which addresses it
  - It will provide a default choice of algorithm for each product type (often only one)

Algorithm Frameworks

- Who implements standard, highly-applicable methods many times?
- Frameworks, parametrised by suitable objects
- MC and FD/FE are obvious examples

Algorithm Frameworks

Finite Difference

A FD engine might be parametrised by

- An equation object
  - This is our choice of model
    - 1D BS equation, LV equation, short rate equation
    - 2D Heston, stock+short rate, stock+hazard rate
    - 3D LV stock + mean-rev hazard rate + short rate, stock + 2F rate model, stock + 2 hazard rates, etc
  - It creates the transformation objects
- A FD product object
  - This creates BC *description* objects and the terminal condition object
  - This is not the same as the Product object in the repository: algorithm-specific
  - This calculates obstacles (early exercise, for solver)
- A scheme object
  - Theta, three-level
  - This creates BC *discretisation* objects
- A solver
  - For the system generated by the scheme and BC discretisation, using the equation's coefficients
  - Tridiagonal, PGS/PSOR, multigrid

Algorithm Frameworks

Finite Difference

```
FD1DLVDivEquation theEquation(
                                    evaluationDate                          ,
                                    maturity                                ,
                                    spot                                    ,
                                    dividends                               ,
                                    undgYieldCurve                          ,
                                    localVol                                ,
                                    pytYieldCurve                           ,
                                    control.SpatialControls.TransformLinLog ,
                                    control.SpatialControls.TransformDrift  );

FD1DEngine engine( evaluationDate, theEquation, control );

underlyingOption = auto_ptr<FD1DProduct>(
      new FD1DVanillaProduct(  strike          ,
                               maturity        ,
                               callOrPut       ,
                               cEuroOrAmer::European));
engine.AddProduct( *underlyingOption, "UnderlyingOption" );

genericBarrierOption = std::auto_ptr<FD1DProduct>(
      new FD1DProductGenericBarrier(          strike                  ,
                                              maturity                ,
                                              callOrPut               ,
                                              lowerBarriers           ,
                                              upperBarriers           ,
                                              engine("UnderlyingOption" ) ,
                                              &pytYieldCurve          ));
engine.AddProduct(*genericBarrierOption);
engine.Calculate();
```

Which scheme?
Grid size, …

Creates a solution domain and names it

// Alternative :
Option myOption(spot, strike, maturity, vol, rate, divs, Call, American, timeSteps, meshPoints);
// ☹

Algorithm Frameworks

Finite Difference

Concepts

- Equation
  - base makes no commitment to asset class (e.g. divs)
  - creates transformations (if any)
  - delivers time-and-space-dependent coefficients
- Domain
  - associated with an equation and a product
  - may be different in extent from other domains e.g. KI barrier
- Sub-Domain
  - associated with a time sub-period characterised by product BCs e.g. changes in barrier level, call feature etc
  - may be different in extent from neighbouring SDs
  - import solution from fully-evolved SD
- Scheme
  - uses the equation
  - creates BC discretisation objects (e.g. centred or one-sided Neumann discretisation)
  - builds the linear system **M** $\underline{x}$ = $\underline{r}$
- Solver
  - as it says, and no more ! (e.g. no knowledge of equations etc)

Algorithm Frameworks

Finite Difference

Where are potential inefficiencies ?

- Duplicate evaluations of the coefficients of the equation (x,t)
  - they occur in matrix M and RHS vector r
- Poor local vol query method
  - change a 2D interpolation into 1D
- Time-dependent coefficients are evaluated once per time step

If domains are associated with different equations and extents (different grids) then no further optimisations are obvious.

We can consider the case of a resettable CB. We certainly have many solutions, but in domains which share an equation and are of the same extent. The matrix M must clearly evaluated just once.

At DB

- C++ throughout the core analytics
  - Other technologies in client apps
- Third-party libraries
  - Nag
    - optimisers, root-finders, quadrature, special functions, …
  - Boost
    - regular expressions, serialisation
  - SuperLU
  - CBLAS

The System Environment of a Core Analytics Library

Its Client Applications (DB)

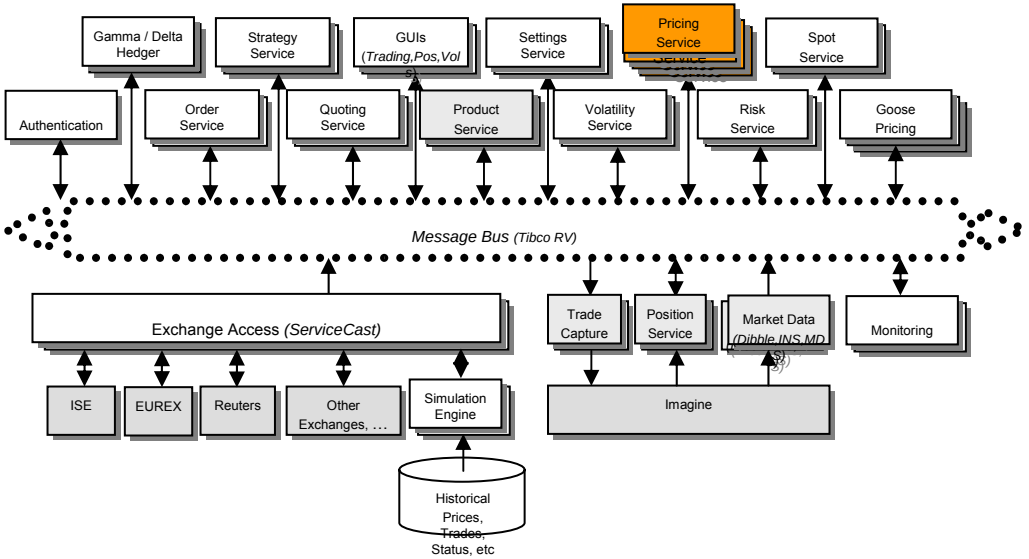
- Avalon
- Imagine
- SOAP Server
- Excel
- DBRO
- Client sites
- CB Risk Servers
- XOL
- Goose
- Others

- Distributed apps
- Monoliths
- Server farms
- Spreadsheet

Windows	MSVC.Net
Linux	gcc

Avalon Framework Overview

- Frameworks
  - Frameworks provide consistent and easy client integration to Avalon
  - Frameworks have bindings to C++, Java and C#
  - Messaging Framework, Trading Framework
  - Agents (specification for implementation of Business Logic)
- Services
  - Pricing, Volatility, Quoting, Strategy, Hedging, Market Access, Market Data, Risk, Portfolio, Simulation, Settings
- GUIs
  - Trading GUI, Strategy GUI, Excel Add-ins, Risk Management GUI, Monitoring GUIs, Admin GUIs



Benefits of SOA

- **Avalon is a Service Oriented Architecture (SOA), the benefits are:**
  - **Reduces Coupling** between components & Services (each Service does a specific task)
  - **Reduces Complexity** of each Service (easier to understand and maintain by developers)
  - Promotes and allows **Reusability** (easy to integrate into other applications)
  - Allows for **Distributed Development** (no monolithic application architecture, so software development can be done across the globe)
  - **Reliability** (each Service uses common frameworks with failover inbuilt, and allows hot standby)
  - **Distributed Processing** (architecture allows multiple instances of Services to be run for distributing processing load, e.g., Pricing)
  - Increases **Developer Productivity** (Services & Framework implement most infrastructure work allowing developers to focus on business logic)

Business Drivers

- **Flow Exotics**
  - Products once considered exotic are now becoming mainstream and traded more frequently like flow vanilla products.
  - Tighter integration of vanilla hedges with corresponding exotic trades.
  - Exotic Price Contribution to retail markets for additional revenue opportunities.
- **Product Innovation**
  - Continuing to create new Equity Derivative products with different underlying and payoff characteristics.
  - Cross-asset class products (such as Commodity/Equity hybrids).
- **Global Risk Management**
  - Seamless and consistent EOD processes across regions.
- **Projections**
  - With increased book risk due to volume increases & product complexity projections (1D & multi-dimensional) are required by traders and regulators.

Technology Drivers

- **Computation**
  - Imagine cannot easily support new Quant library upgrades or the market and model data (eg. large volatility surfaces, parameterised volatility, Heston parameters etc).
  - GOOSE currently best suited to long running calculations. This year we've seen an explosion of vanilla products needing to be on GOOSE, causing scalability & performance issues.
  - Compute farm is at full capacity across a full 24 hours (intraday & EOD, projections, BAC jobs across all regions). 200,000 jobs per day.
- **Data**
  - Massive increases in product, market and result data.
    - Daily throughput of **1 terabyte** across the network
    - **25 million** result rows into the Result Service & database.
  - Cross-asset class products (such as Commodity/Equity hybrids).
- **Supportability & Delivery**
  - Increased duplication between Imagine & GOOSE is support and development intensive.
  - We need to provide quicker turnaround of functionality in both the back-end and front-end systems.

Thank you very much for your attention.

andy.ferraris@db.com
www.dbquant.com