STRATA

Open Source Market Risk

OpenGamma

STEPHEN COLEBOURNE

- Java Champion, regular conference speaker
- Best known for date & time - Joda-Time and JSR-310
- More Joda projects - http://www.joda.org
- Major contributions in Apache Commons
- Blog - http://blog.joda.org
- Worked at OpenGamma for 6 years

OPENGAMMA

- Founded in 2009
- Financed by Venture Capital
- Mission to bring Open Source values to finance industry
- Focus on Market Risk Analytics, Derivatives and Clearing
- Cloud-based offering coming soon

MARKET RISK

MARKET RISK

- Providing pricing and analytics on a financial portfolio
  - “present value (NPV) of an interest rate swap”
  - “PV01 of a forward rate agreement (FRA)”
  - “vega and gamma of an FX vanilla option”
  - “examine the portfolio against a set of scenarios”

COMPONENTS FOR MARKET RISK

- Pricing/Analytic models
- Trade representations - for each supported asset class
- Market data representations - quotes, curves, surfaces, etc.
- Calibration - curves, surfaces, etc.
- Market data management and Scenario creation
- Reference data - holiday calendars, securities
- Basics - schedules, day counts, currencies, etc.

BUILD, BUY OR OPEN SOURCE?

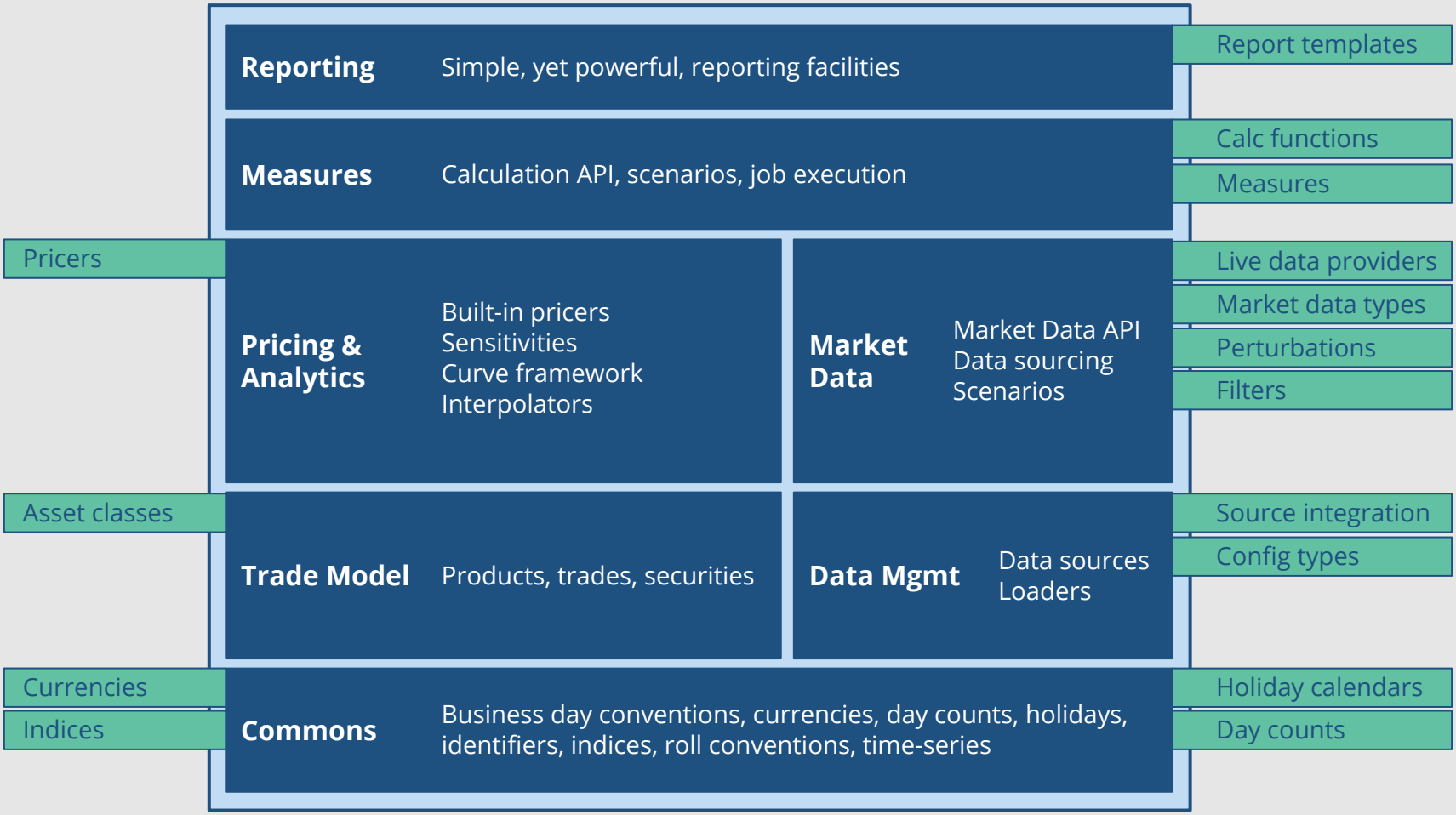
- Build it in-house
- Buy from a vendor
- Open Source
    - **QuantLib** (C++, with exports to other languages)
    - **Strata** (Java/JVM)

STRATA

OVERVIEW

- Open Source - Apache v2 license
- Lightweight, easy-to-use library
- Just jar files - no servers or databases needed
- Released in Maven Central
- Foundation for OpenGamma commercial products

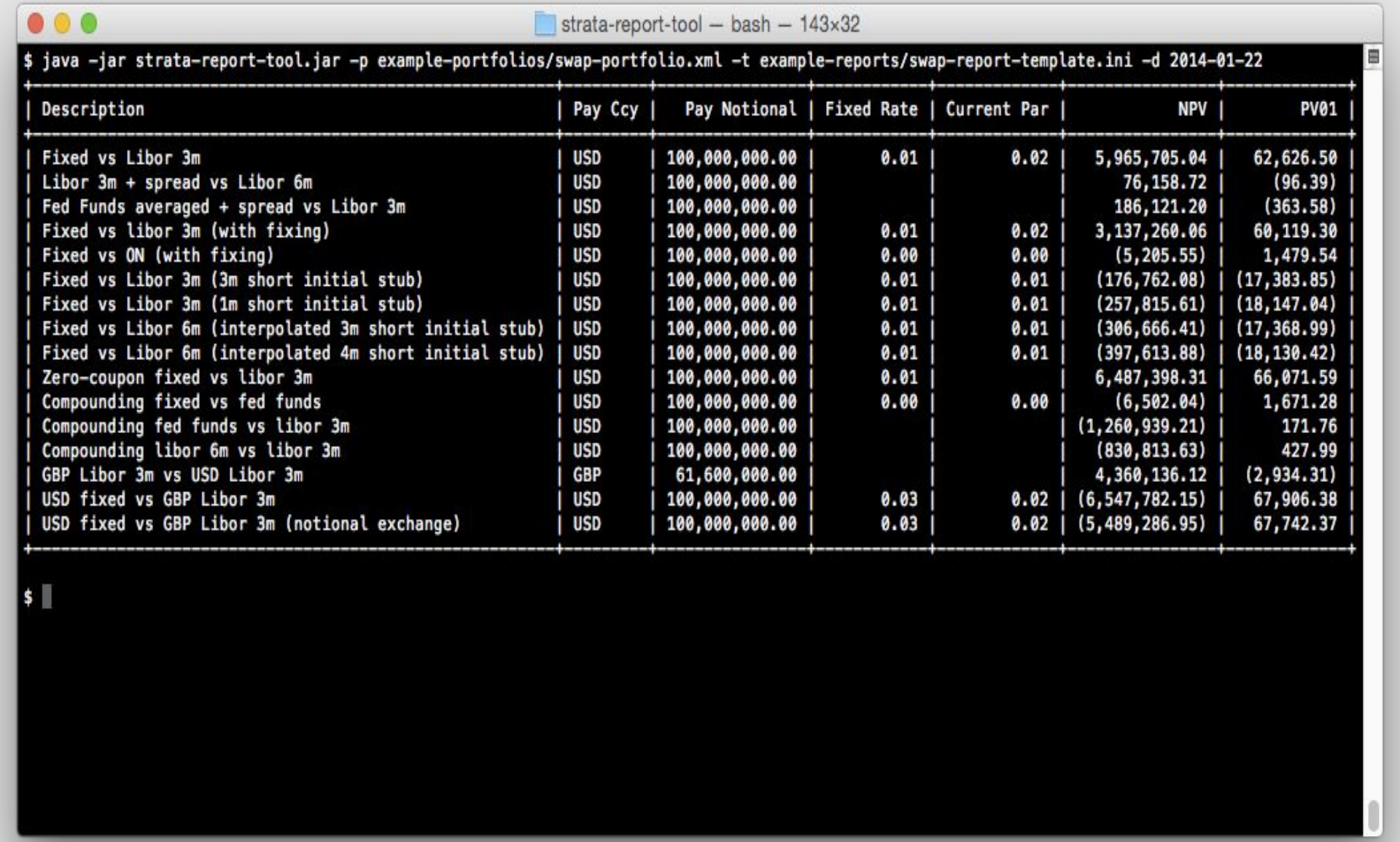
OpenGamma


Reporting
Simple, yet powerful, reporting facilities
Report templates
Measures
Calculation API, scenarios, job execution
Calc functions
Measures
Pricers
Pricing & Analytics
Built-in pricers
Sensitivities
Curve framework
Interpolators
Market Data
Market Data API
Data sourcing
Scenarios
Live data providers
Market data types
Perturbations
Filters
Asset classes
Trade Model
Products, trades, securities
Data Mgmt
Data sources
Loaders
Source integration
Config types
Currencies
Indices
Commons
Business day conventions, currencies, day counts, holidays, identifiers, indices, roll conventions, time-series
Holiday calendars
Day counts

QUICK START

- Built in examples
    - Up and running in minutes
- Hard coded reference data
    - Holidays, indices, conventions
- Simple command line reporting

OpenGamma


strata-report-tool — bash — 143×32
$ java -jar strata-report-tool.jar -p example-portfolios/swap-portfolio.xml -t example-reports/swap-report-template.ini -d 2014-01-22
Description	Pay Ccy	Pay Notional	Fixed Rate	Current Par	NPV	PV01
Fixed vs Libor 3m	USD	100,000,000.00	0.01	0.02	5,965,705.04	62,626.50
Libor 3m + spread vs Libor 6m	USD	100,000,000.00			76,158.72	(96.39)
Fed Funds averaged + spread vs Libor 3m	USD	100,000,000.00			186,121.20	(363.58)
Fixed vs libor 3m (with fixing)	USD	100,000,000.00	0.01	0.02	3,137,260.06	60,119.30
Fixed vs ON (with fixing)	USD	100,000,000.00	0.00	0.00	(5,205.55)	1,479.54
Fixed vs Libor 3m (3m short initial stub)	USD	100,000,000.00	0.01	0.01	(176,762.08)	(17,383.85)
Fixed vs Libor 3m (1m short initial stub)	USD	100,000,000.00	0.01	0.01	(257,815.61)	(18,147.04)
Fixed vs Libor 6m (interpolated 3m short initial stub)	USD	100,000,000.00	0.01	0.01	(306,666.41)	(17,368.99)
Fixed vs Libor 6m (interpolated 4m short initial stub)	USD	100,000,000.00	0.01	0.01	(397,613.88)	(18,130.42)
Zero-coupon fixed vs libor 3m	USD	100,000,000.00	0.01		6,487,398.31	66,071.59
Compounding fixed vs fed funds	USD	100,000,000.00	0.00	0.00	(6,502.04)	1,671.28
Compounding fed funds vs libor 3m	USD	100,000,000.00			(1,260,939.21)	171.76
Compounding libor 6m vs libor 3m	USD	100,000,000.00			(830,813.63)	427.99
GBP Libor 3m vs USD Libor 3m	GBP	61,600,000.00			4,360,136.12	(2,934.31)
USD fixed vs GBP Libor 3m	USD	100,000,000.00	0.03	0.02	(6,547,782.15)	67,906.38
USD fixed vs GBP Libor 3m (notional exchange)	USD	100,000,000.00	0.03	0.02	(5,489,286.95)	67,742.37
$

JAVA 8

- Early decision to use Java 8 for Strata
- Features very beneficial for Market Risk Analytics
  - Date and Time
  - Streams and Lambdas
  - Methods on interfaces
- Affects both macro-level design and micro-level code

DEPENDENCIES

- Guava
- Joda-Beans
- Joda-Convert
- SLF4J
- Commons-Math
- Colt

MODULES

- **strata-collect** - low-level, arrays, time-series, IO, tuples
- **strata-basics** - holidays, schedules, indices, reference data
- **strata-data** - market data containers
- **strata-market** - market data structures - curves, surfaces, etc
- **strata-product** - trades, products, securities
- **strata-loader** - data loaders from csv and xml
- **strata-pricer** - analytic pricers
- **strata-calc** - calculation engine, scenarios, market data building
- **strata-measure** - high-level measures, potentially multi-scenario

TRADES

TRADES

- Trades are simple immutable beans (data objects)
- Built using Joda-Beans
- Use builder or static factory to create
- Real properties and methods with Javadoc

JODA-BEANS

- Source code generator/regenerator
- Just write the fields and add a couple of annotations
- Joda-Beans generates additional high-quality source code
- Mutable and immutable beans
- Provides C# style properties
- Easy and fast serialization to XML, JSON, Binary

OpenGamma

JODA-BEANS

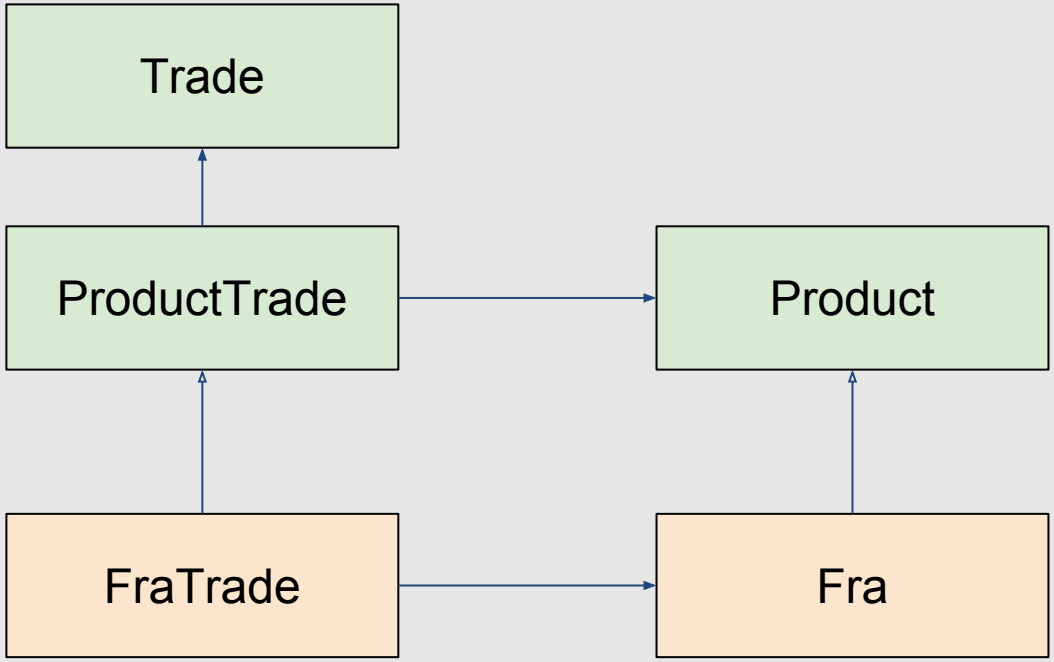
```
@BeanDefinition
public final class Person implements ImmutableBean {
  @PropertyDefinition(validate = "notNull")
  private final String forename;
  @PropertyDefinition(validate = "notNull")
  private final String surname;

  // autogenerated getters, equals, hashCode,
  // toString, builder, metabean
}
```

TRADES AND PRODUCTS

- Trade
  - Trade date
  - Trade identifier
  - Counterparty
- Product
  - Financial details of the trade
  - Effective/Termination date, notional, rate, index, etc.

TRADES AND PRODUCTS



OpenGamma

FRA

```
// create Fra using builder
Fra fra = Fra.builder()
  .buySell(BuySell.BUY)                  // Buy
  .index(IborIndices.GBP_LIBOR_3M)       // GBP LIBOR 3M
  .notional(10_000_000)                  // 10 million GBP
  .fixedRate(0.0085)                     // 0.85%
  .startDate(LocalDate.of(2016, 9, 14))
  .endDate(LocalDate.of(2016, 12, 14))
  .build();
```

OpenGamma

FRA

```
// create FraTrade
Fra fra = ...
TradeInfo info = TradeInfo.builder()
  .tradeDate(LocalDate.of(2016, 6, 14))
  .id(StandardId.of("Trade", "123456"))
  .counterparty(StandardId.of("Party", "654321"))
  .build();
FraTrade trade = FraTrade.of(info, fra);
```

CONVENTIONS

- Most OTC trades follow market conventions
- Strata includes definitions of some of these conventions
- Avoids repetitive code

OpenGamma

FRA CONVENTION

```
// create FRA from a convention
FraTrade trade =
  FraConvention.of(IborIndices.GBP_LIBOR_3M)
    .createTrade(
      LocalDate.of(2015, 7, 14),  // Trade date
      Period.ofMonths(2),         // start in 2 months
      BuySell.BUY,                // Buy
      10_000_000,                 // 10 million GBP
      0.012,                      // 1.2%
      ReferenceData.standard());  // Holiday calendars
```

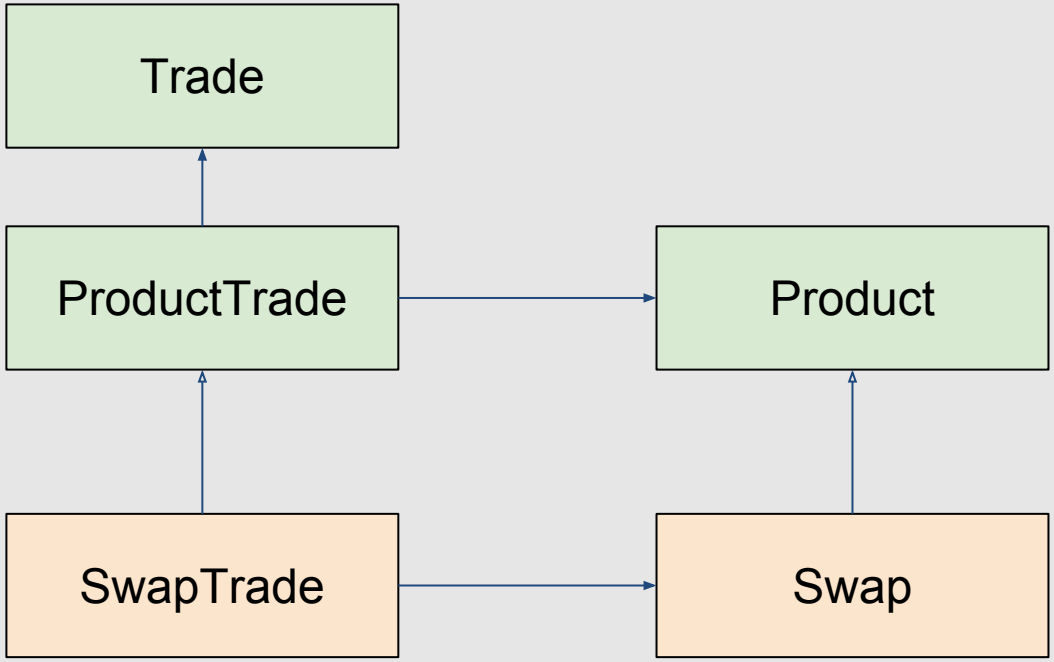
NOTES

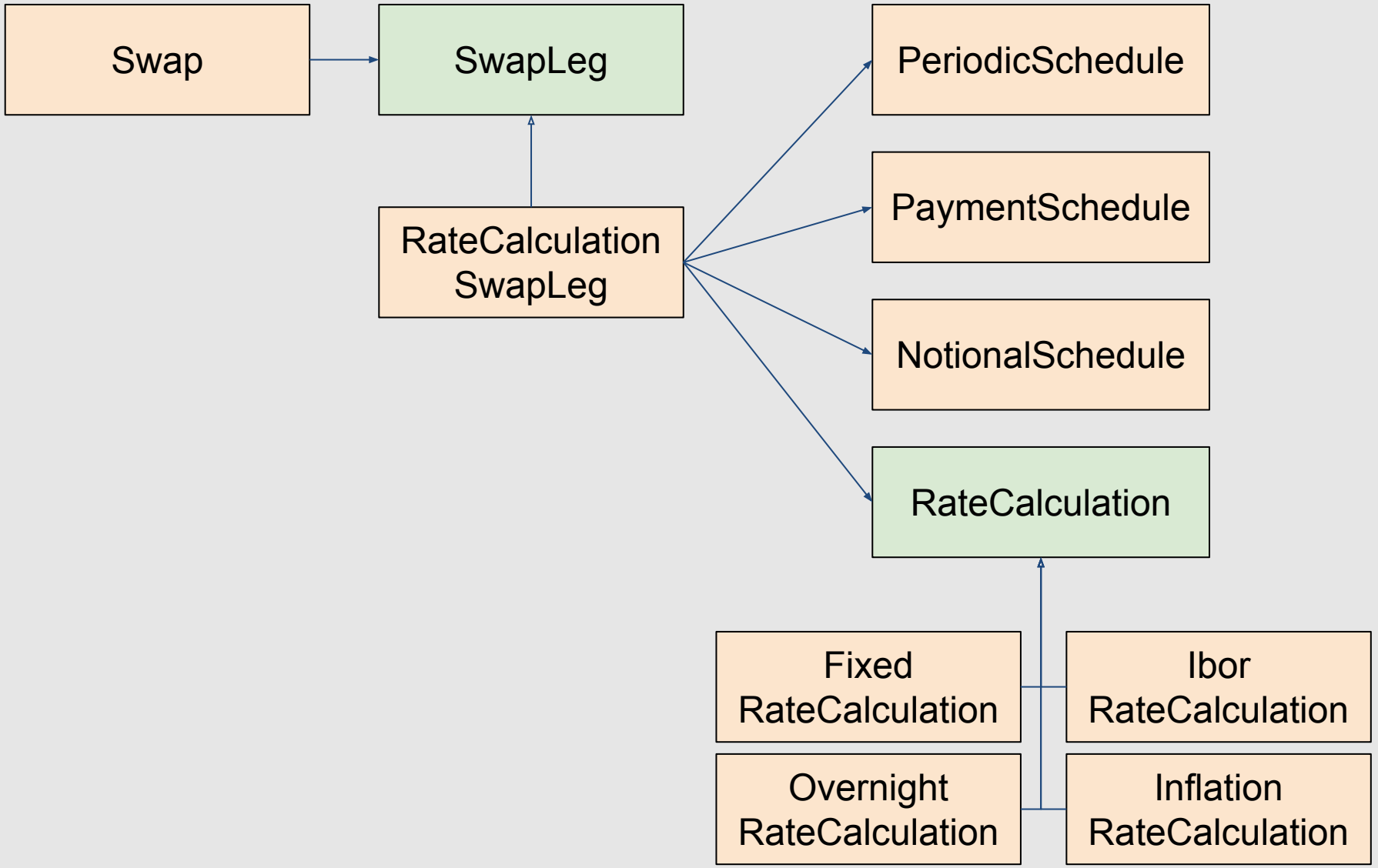
- Built in conventions for FRAs
- Built in conventions for 18 ibor-like indices, such as GBP LIBOR
- Built in conventions for 12 overnight indices, such as GBP SONIA
- Built in conventions for currencies
- Built in holiday calendar data, suitable for evaluation

SWAP

- Flexible interest rate swap
- Fixed legs support variable interest rates and known amounts
- Float legs support Ibor, Overnight and Inflation
- Stubs support fixed, floating, interpolated and known amount
- Support for variable notional, gearing and spread
- Conventions and Templates available

SWAP




Swap
SwapLeg
PeriodicSchedule
RateCalculation SwapLeg
PaymentSchedule
NotionalSchedule
RateCalculation
Fixed RateCalculation
Ibor RateCalculation
Overnight RateCalculation
Inflation RateCalculation

OpenGamma

SWAP

```
// create swap from a convention
SwapTrade trade =
  FixedIborSwapConventions.GBP_FIXED_1Y_LIBOR_3M
    .createTrade(
      LocalDate.of(2015, 7, 14),  // Trade date
      Tenor.TENOR_10Y,            // 10 year swap
      BuySell.BUY,                // Buy
      10_000_000,                 // 10 million GBP
      0.014,                      // 1.4%
      ReferenceData.standard());  // Holiday calendars
```

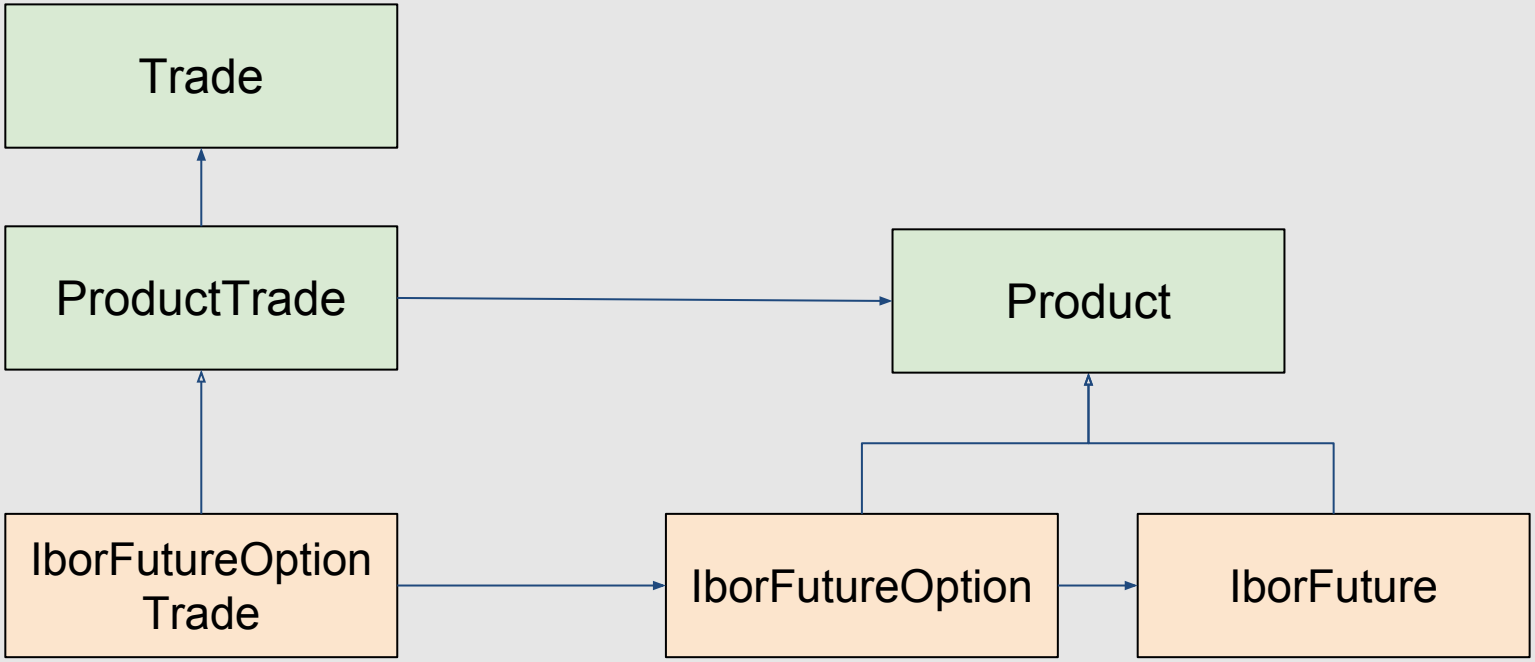
OpenGamma

SECURITIES

- Two approaches supported
  - Define security as and when needed
  - Setup reference data map of securities
- Provides ability to use, or avoid, a big security master

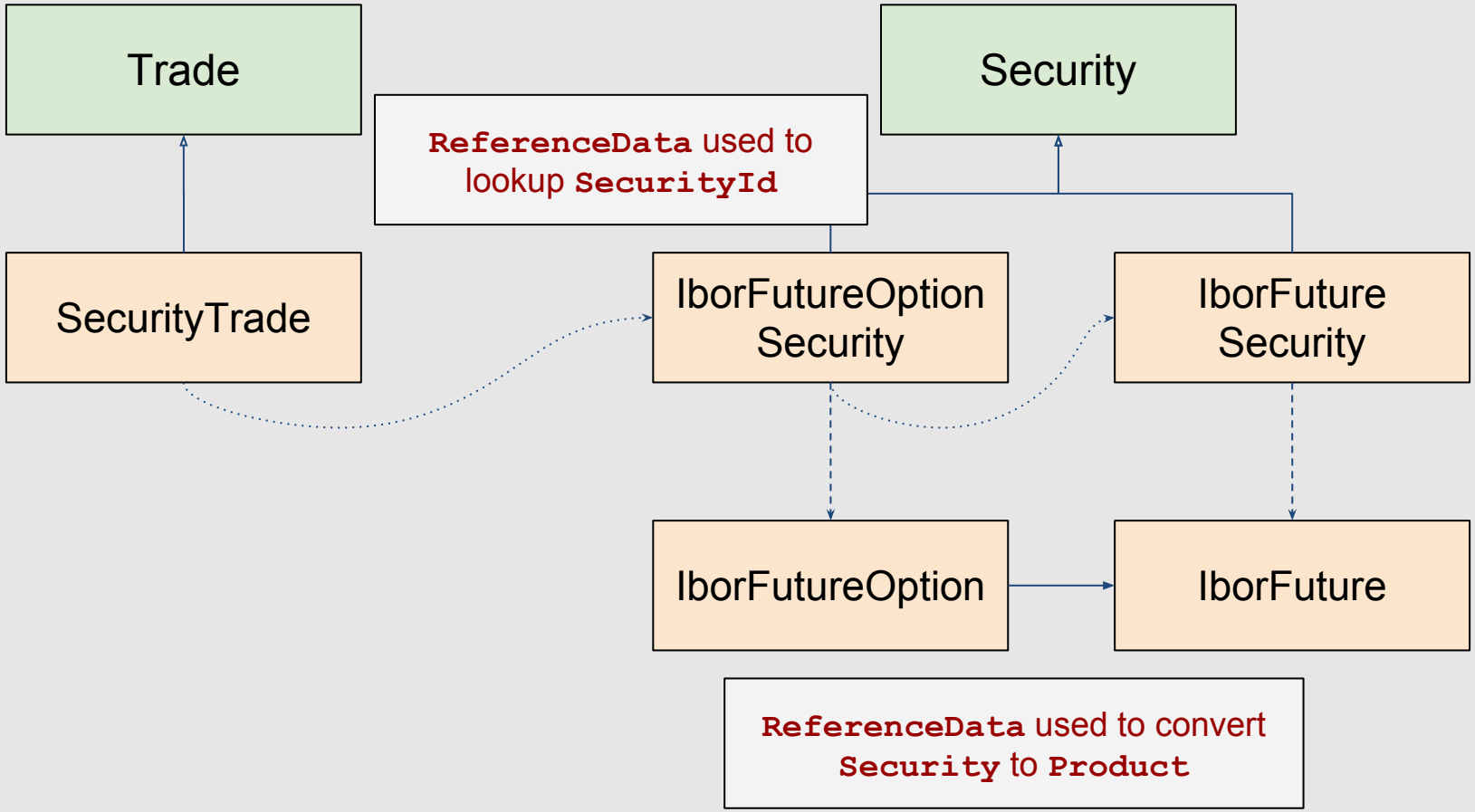
OpenGamma

SECURITIES AS PRODUCTS



OpenGamma

SECURITIES AS REFERENCE DATA



OpenGamma

ASSET CLASSES

- Swaps, Swaptions, DSF, CMS, Cap/Floor
- FRA, STIR futures, STIR future options
- Bond, Bond futures, Bond future options
- FX forward, NDF, FX swap, vanilla option, single barrier option
- CDS
- Term deposit, Bullet payment
- Generic security, ETD future, ETD option
- Additional asset classes may be for commercial customers only

PRICERS

PRICERS

- Lower-level analytics API
- Provides ability to calculate PV, sensitivities, greeks, etc.
    - Explain facility to understand how result was calculated
- Operates on ***resolved*** trades/products

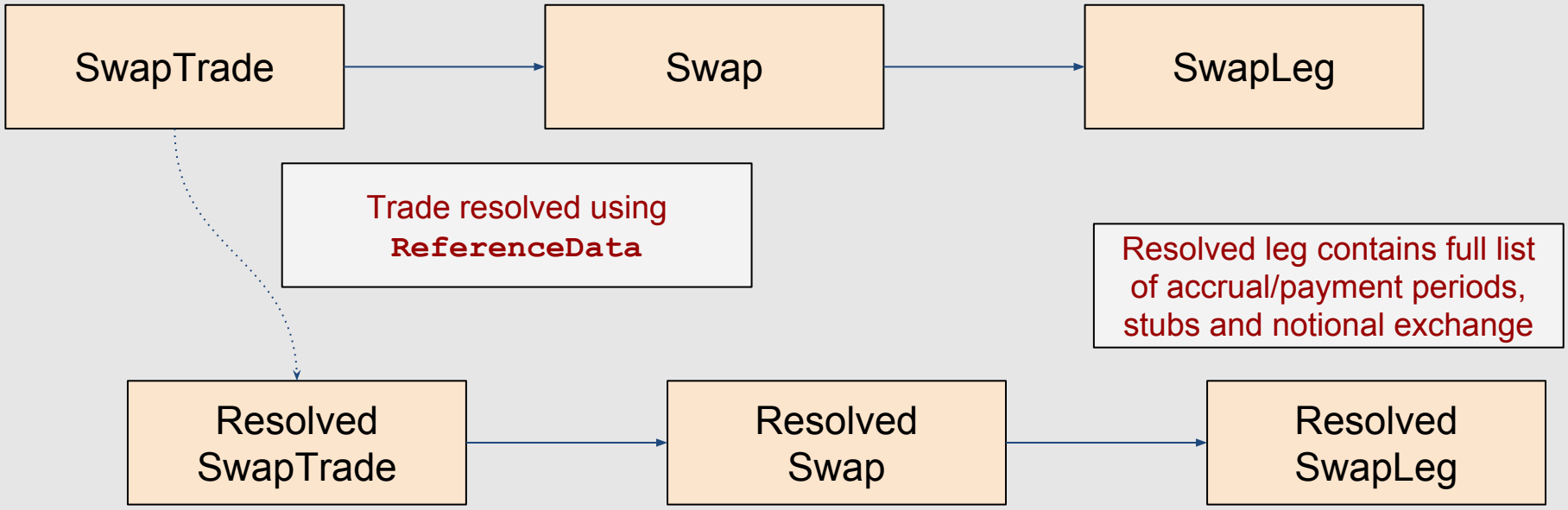
RESOLVING

- Resolving the trade requires reference data
- Locks in dates to the current holiday calendar rules
- Standard reference data contains hard coded holiday rules

```
// resolve a swap
SwapTrade trade = …
ReferenceData refData = ReferenceData.standard();
ResolvedSwapTrade resolved = trade.resolve(refData);
```

OpenGamma

RESOLVING



USING A PRICER

- Stateless - takes resolved trade and any necessary market data
- Calculates for one trade and one set of market data
- Can usually price at trade or product level

OpenGamma

USING A PRICER

```
// obtain the swap and market data to price against
ResolvedSwapTrade trade = ...
RatesProvider market = ...

// calculate the present value
MultiCurrencyAmount pv =
  DiscountingSwapTradePricer.DEFAULT
      .presentValue(trade, market);
```

MARKET DATA

MARKET DATA

- Support for all kinds of market data
- Built in classes for FX, quotes, curves, surfaces, etc.
- Many types can be loaded from CSV
- Multi-curve rates calibration
- Scenarios, stored efficiently as arrays

RATES PROVIDER

- **`RatesProvider`** is a single, coherent, set of market data
  - FX rates, Discount factors, Ibor rates
  - Overnight rates, Inflation price indices, Historic fixings

```
// get discount factors for GBP
DiscountFactors df = ratesProvider.discountFactors(GBP);
double factor = df.discountFactor(date);
```

OpenGamma

MARKET DATA

- **`MarketData`** is a container of market data
- Hash-map like
- Keys are **`MarketDataId<T>`**

```
// get curve by identifier
CurveId id = CurveId.of("Default", "USD-DSC");
Curve curve = marketData.getValue(id);
```

OpenGamma

SCENARIO MARKET DATA

- **`ScenarioMarketData`** is a container of scenario data
- Hash-map like, where values are arrays
- Keys are **`MarketDataId<T>`**

```
// get curves by identifier
CurveId id = CurveId.of("Default", "USD-DSC");
MarketDataBox<Curve> curves = scenarioData.getValue(id);
// process using a stream (for example)
curves.stream().forEach( curve -> … );
```

OpenGamma

MARKET DATA LOOKUP

- Market data containers hold arbitrary sets of market data
  - May hold multiple USD discounting curves
- **`RatesMarketDataLookup`** is used to select a coherent set

```
CurveId usdDscId = CurveId.of("Default", "USD-DSC");
CurveId usdLiborId = CurveId.of("Default", "USD-LIBOR");
// map currency/index to curve
RatesMarketDataLookup md = RatesMarketDataLookup.of(
  ImmutableMap.of(Currency.USD, usdDscId),
  ImmutableMap.of(
    IborIndices.USD_LIBOR_3M, usdLiborId),
    IborIndices.USD_LIBOR_6M, usdLiborId));
```

OpenGamma

COMBINATIONS

MarketData + RatesMarketDataLookup = RatesProvider

- Many curves
- Keyed by curve ID

- Map currency to curve ID
- Map index to curve ID

- DF by currency+date
- Rate by index+date

MARKET DATA BUILDING

- Can create market data manually, loading from CSV or by factory
- **`MarketDataFactory`** can
  - Query quotes from a simple provider interface
  - Query time-series from a simple provider interface
  - Calibrate
  - Create scenarios by shifting/bumping
- See **`SwapPricingWithCalibrationExample`**

MEASURES

MEASURE-LEVEL API

- Higher-level than pricers
- Stateless - takes resolved trade and any necessary market data
- Calculates for one trade and one or more sets of market data
    - ie. supports scenarios
- Only operates on trades, not products
- Scaled output
    - eg. PV01 in basis points

OpenGamma

USING THE MEASURE-LEVEL API

```
// obtain the swap and market data to price against
ResolvedSwapTrade trade = ...
RatesProvider market = ...

// calculate the present value
MultiCurrencyAmount pv =
  SwapTradeCalculations.DEFAULT
      .presentValue(trade, market);
```

OpenGamma

USING THE MEASURE-LEVEL API

```
// obtain the swap and market data to price against
ResolvedSwapTrade trade = ...
RatesMarketDataLookup lookup = ...
ScenarioMarketData market = ...

// calculate the present value for many scenarios
MultiCurrencyScenarioArray scenarioPv =
  SwapTradeCalculations.DEFAULT
      .presentValue(trade, lookup, market);
```

CALCULATIONS

OpenGamma

CALCULATION-LEVEL API

- Highest-level API
- Calculates for many trades and one or more sets of market data
    - ie. supports scenarios
- Optional currency conversion
- Multi-threaded
- Results can be received asynchronously

OpenGamma

CALCULATION-LEVEL API

- Calculation API result is a grid
  - Rows are trades, positions, or similar
  - Columns are measures, such as PV, PV01, Par rate
- Mixed portfolio of trades (PV for Swap, FRA and future in one call)

	NPV (USD)	NPV (GBP)	PV01 (USD)	Par rate
Trade 1 - Swap	13,487.25	10,176.72	12.7365	0.23
Trade 2 - Swap	-34,276.73	-27,273.28	76.2725	0.24
Trade 3 - FRA	12,835.26	9,263.75	-26.8367	0.31
Trade 4 - STIR	-965.76	-754.23	1.2676	0.52

OpenGamma

CALCULATIONS

- **`CalculationRunner`** is entry to Calculation API
- Provides a multi-threaded executor
- Also allow callers to use their own executor

```
// obtains a multithreaded runner
try (CalculationRunner runner =
                    CalculationRunner.ofMultiThreaded()) {
  // use the runner
}
```

RULES

- **`CalculationRules`** defines how to calculate
- Functions mapping from trade type to code
- Reporting currency
- Market data lookup

```
// setup the rules
CalculationRules rules = CalculationRules.of(
  StandardComponents.calculationFunctions(),
  Currency.USD,
  ratesMarketDataLookup);
```

CALCULATIONS

- Each column defined by **`Column`**
- Measure specifies what to calculate
- Can control reporting currency per column

```
// specify the columns
List<Column> columns = ImmutableList.of(
  Column.of(Measure.PRESENT_VALUE),
  Column.of(Measure.PRESENT_VALUE, Currency.GBP),
  Column.of(Measure.PV01_CALIBRATED_SUM),
  Column.of(Measure.PAR_RATE));
```

OpenGamma

CALCULATIONS

- Calculation runner is stateless
- Pass in all inputs, get back results
- Separate API allows results to be received asynchronously

```
// calculate the results
Results results = runner.calculate(
    rules,              // How to calculate
    trades,             // Trades to process
    columns,            // Columns, eg PV, PV01, par rate
    marketData,         // Market data
    referenceData);     // Reference data
```

OpenGamma

CALCULATION-LEVEL API

	NPV (USD)	NPV (GBP)	PV01 (USD)	Par rate
Trade 1 - Swap	13,487.25	10,176.72	12.7365	0.23
Trade 2 - Swap	-34,276.73	-27,273.28	76.2725	0.24
Trade 3 - FRA	12,835.26	9,263.75	-26.8367	0.31
Trade 4 - STIR	-965.76	-754.23	1.2676	0.52
...	...	...	...	...

SUMMARY

STRATA v1.2

- Trades are immutable beans
- Pricing/risk logic is stateless, and separate from the trades
- Three levels of pricing/risk API
  - Pricer - one trade, one set of market data
  - Measure - one trade, one or many sets of market data
  - Calc - many trades, one or many sets of market data

STRATA v1.2

- Modern market risk library in Java 8
- Lightweight and easy-to-use, lots of examples
- Good asset class coverage, lots of built-in conventions
- Open source, Apache v2 license
- Foundation of commercial products
    - cloud-based calculations for derivatives trading
- Commercial support available from OpenGamma

http://strata.opengamma.io/

THANK YOU