CREDIT SUISSE

Experience Report:

Paradise

A two-stage DSL embedded in Haskell

Lennart Augustsson

Howard Mansell

Ganesh Sittampalam

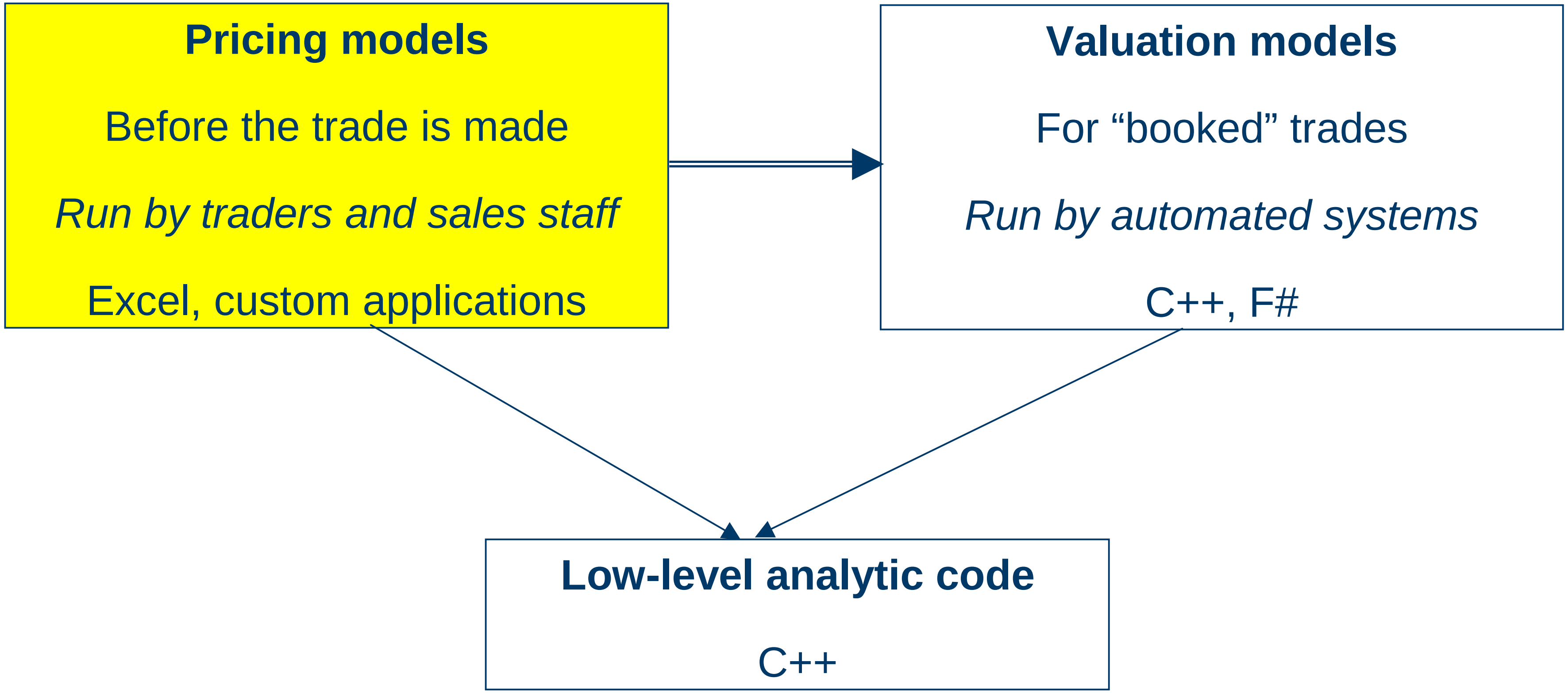
GMAG

- Credit Suisse: ~40,000 people: A large financial services firm
- Securities Division: ~6,000 people:

  Buys and sells financial products, including derivatives:
  - Potentially complex instruments based on underlying real asset(s), e.g.
    - Right to buy/sell some stock at price y at time z
    - Pay out x for every day in period z that some interest rate is above y
  - Used to hedge against liabilities, or “take a view” on market movements
- Global Modelling and Analytics Group: ~140 people

  Writes and delivers “analytics” for:
  - Pricing (what is this worth?)
  - Risk analysis (how will this value change if the market moves by x?)

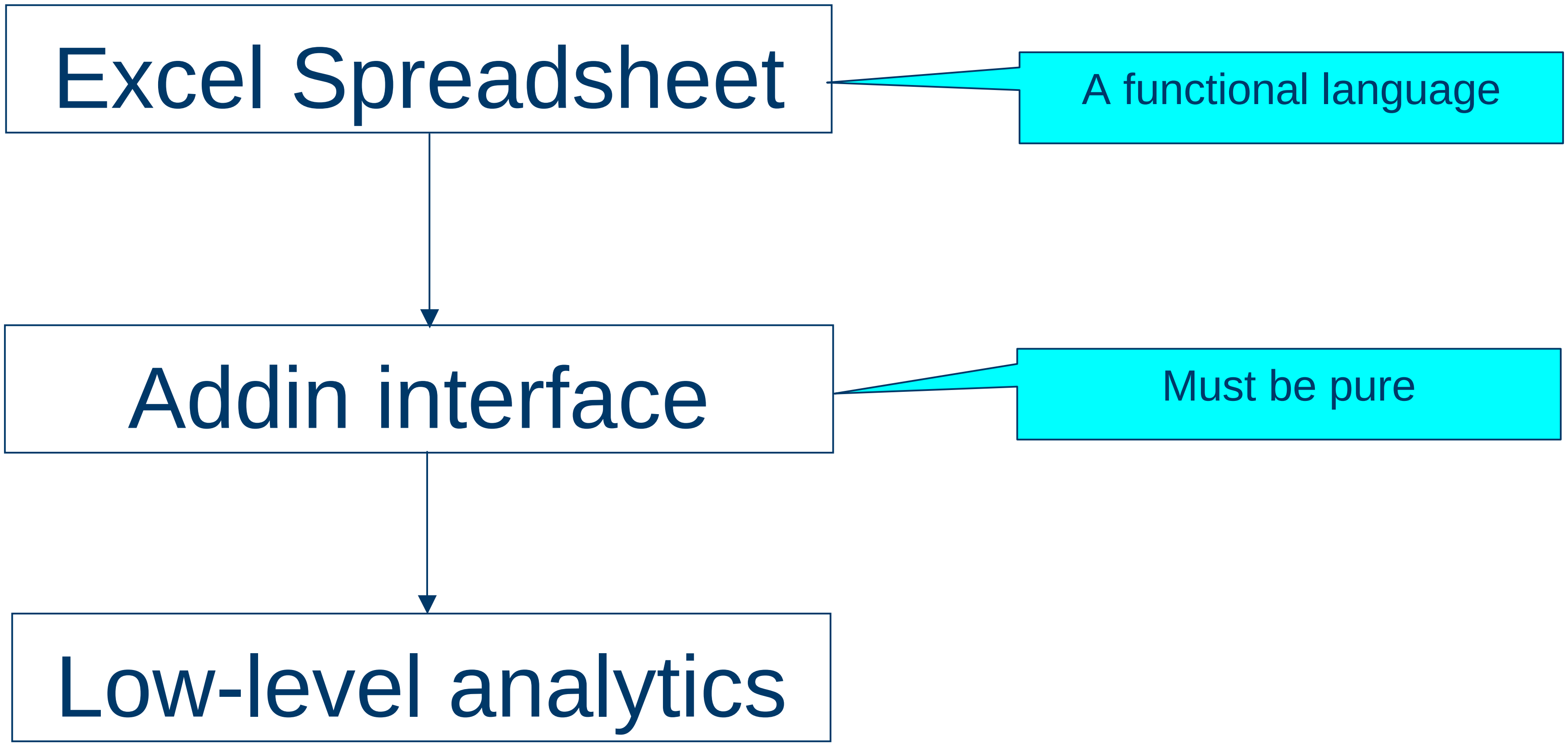
Delivering analytics



Excel as a platform for pricing models

- Users like it
  - Familiar
  - Can tweak it themselves
- Can be good for development too
  - Rapid prototyping
- BUT: a reusability nightmare
  - No modularity
  - No abstraction
  - No automated change tracking

Pricing Models



Paradise

- Generate pricing models
- Two DS(E)Ls
  - Model: describe how the addin calls are plumbed together
  - View: describe how the UI is laid out
- Target-independent
  - Excel
  - C#
  - Future: web application?
- Not specific to finance or pricing models
  - But that’s all we use it for

How does it work

- A two stage language: like Pan (Elliott et al)
  - The Paradise program is compiled into a Haskell executable
  - When run the Haskell executable produces the spreadsheet etc
- Type system distinguishes the stages
  - Second-stage is denoted by a type constructor “E”
- The bits with “Haskell” types (Double, [], …) run at Haskell runtime (stage 1)
- The bits with “Paradise” types (E Double, …) build an AST that will be compiled into the stage 2 program – Excel, C# etc
- The Paradise library contains
  - A type-safe interface to stage 2 + helper functions
  - a compiler for the Paradise DSL

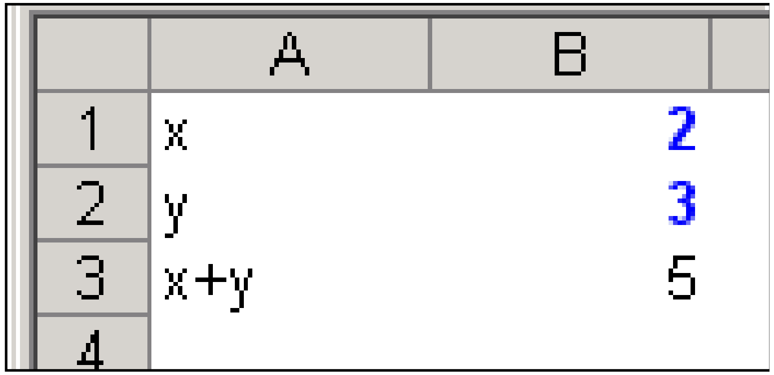
A very simple example

```
data Adder = Adder {
    x :: E Double,
    y :: E Double,
    z :: E Double
    }

adder = do
    x <- input 2
    y <- input 3
    z <- output (x+y)
    return Adder{..}

instance Viewable Adder where
    view Adder{..} =
        column [row [label "x",   view x],
                row [label "y",   view y],
                row [label "x+y", view z]]
```

Data structure



Plumbing DSL

UI DSL

Two adders

```
data TwoAdders = TwoAdders {
   adder1 :: Adder,
   adder2 :: Adder,
   result :: E Double
   }

twoAdders = do
    adder1 <- adder
    adder2 <- adder
    result <- output (z adder1 + z adder2)
    return TwoAdders{..}

instance Viewable TwoAdders where
    view TwoAdders{..} = row [view adder1,
                              view adder2,
                              view result]
```

The E type

This is all we expose to the users

```
data E a = E Exp
data Exp = EVar … | ELit … | ELam … | EApply …

instance Num (E Int) where …
instance Num (E Double) where …
instance IsString (E String) where …
(+++) :: E String -> E String -> E String
…
```

Higher-order functions

- Second-stage array type : E (Array a)

```
map :: (IsEType a, IsEType b)
    => (E a -> E b)
    -> E (Array a)
    -> E (Array b)
```

- User writes normal-looking functions
  - Higher-order abstract syntax
  - E type is abstract so the functions must be parametric
- Turned into explicit lambdas (ELam) for the backend

Embedding issues

- The stage-2 language is quite restricted
  - Can overload:
    - numeric literals, numeric operations, and string literals
  - Can’t overload:
    - If-then-else, Boolean literals, Eq/Ord type classes: we roll our own alternatives
    - Pattern-matching, general recursion, list comprehensions: we do without
- (Un)observable sharing
  - We have a combinator in the state monad
  - We recently decided to also use stable names + unsafePerformIO
- Haskell is still a great language for embedding
  - Static typing + type classes make distinguishing stages 1+2 easy
  - Can write code that is overloaded between the two stages
  - Type classes let us mimic the target language’s type system

Impact

- Gradually rolling it out across the group
- Most of our problems have been with Excel
  - Slow
  - Unreliable
  - Not designed as a compilation target
- Modellers appreciate the type-safety and abstraction
- Turn-around of changes can be an issue
  - No longer instantly visible in the sheet
- Lifecycle management is more complicated
  - Paradise library changes can break models (or fix them)
  - We have a runtime support library: requires a careful binary release process

Disclaimer

This material has been prepared by individual sales and/or trading personnel of Credit Suisse or its subsidiaries or affiliates (collectively "Credit Suisse") and not by Credit Suisse's research department. It is not investment research or a research recommendation for the purposes of FSA rules as it does not constitute substantive research. All Credit Suisse research recommendations can be accessed through the following hyperlink: https://s.research-and-analytics.csfb.com/login.asp subject to the use of approved login arrangements. This material is provided for information purposes, is intended for your use only and does not constitute an invitation or offer to subscribe for or purchase any of the products or services mentioned. Any pricing information provided is indicative only and does not represent a level at which an actual trade could be executed. The information provided is not intended to provide a sufficient basis on which to make an investment decision. Credit Suisse may trade as principal or have proprietary positions in securities or other financial instruments that are the subject of this material. It is intended only to provide observations and views of the said individual sales and/or trading personnel, which may be different from, or inconsistent with, the observations and views of Credit Suisse analysts or other Credit Suisse sales and/or trading personnel, or the proprietary positions of Credit Suisse. Observations and views of the salesperson or trader may change at any time without notice. Information and opinions presented in this material have been obtained or derived from sources believed by Credit Suisse to be reliable, but Credit Suisse makes no representation as to their accuracy or completeness. Credit Suisse accepts no liability for loss arising from the use of this material. Nothing in this material constitutes investment, legal, accounting or tax advice, or a representation that any investment or strategy is suitable or appropriate to your individual circumstances. Any discussions of past performance should not be taken as an indication of future results, and no representation, expressed or implied, is made regarding future results. Trade report information is preliminary and subject to our formal written confirmation.

Why not GADTs?

- Trade-off between effort and safety
- The Excel backend plays fast and loose with types
- We annotate our terms with our own type information (at the value level)
  - Can run a typechecking pass
- So trade-off is between errors at Haskell compile time and at stage 1 runtime
  - Of course, we had to write our own typechecker