REAContracts

Adding sematics to business contracts

Jesper Kiehn (Simon Peyton-Jones)

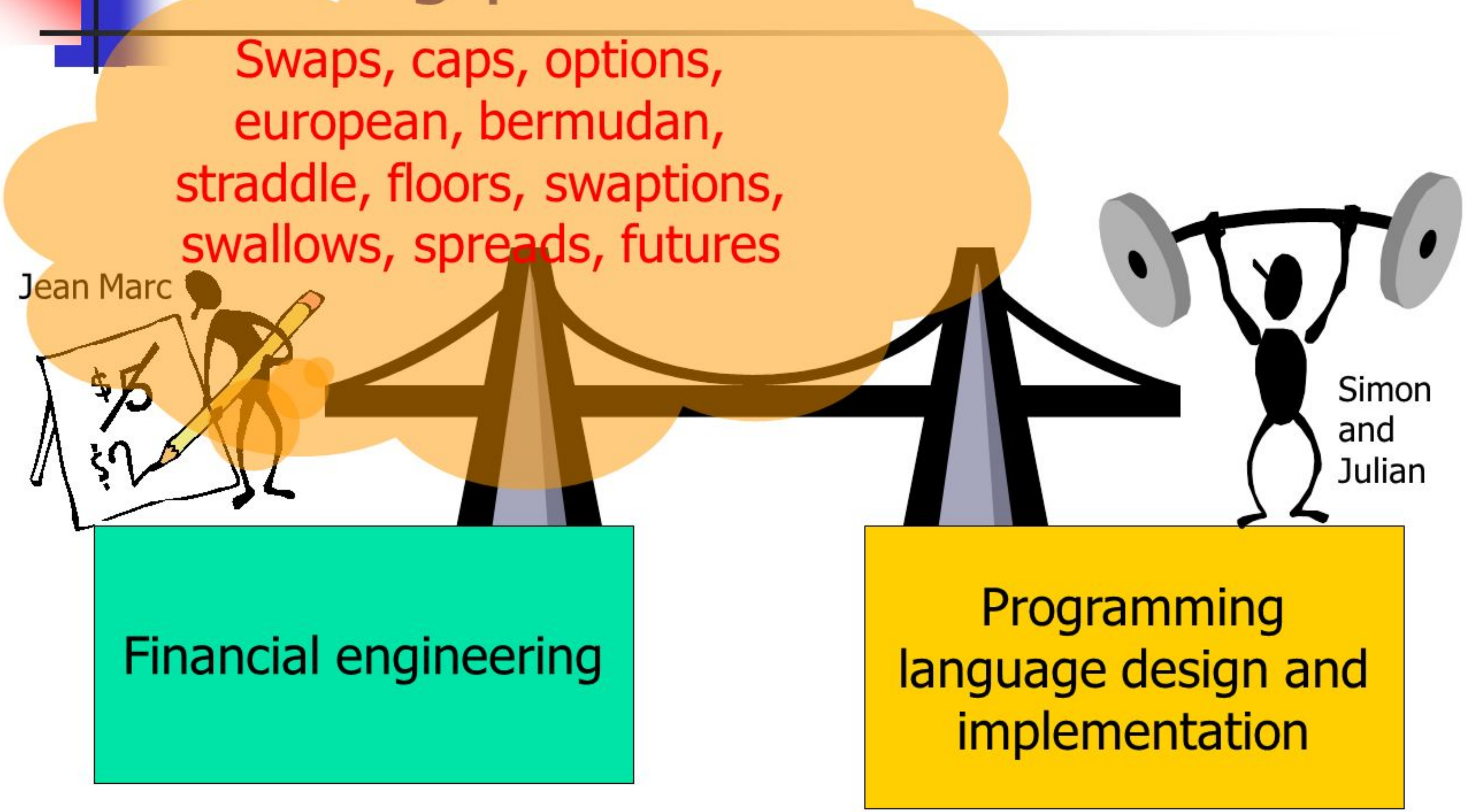
Short bio

- Worked with ERP since graduating as Computer science Engineer
- 5 years as training ERP programming
- 2 years at PWC with implementation and computer auditing
- 10 years Microsoft – wrote a book about REA implementations
- 8 years with different dealers doing standard add-ons
  for expense management and advanced manufacturing

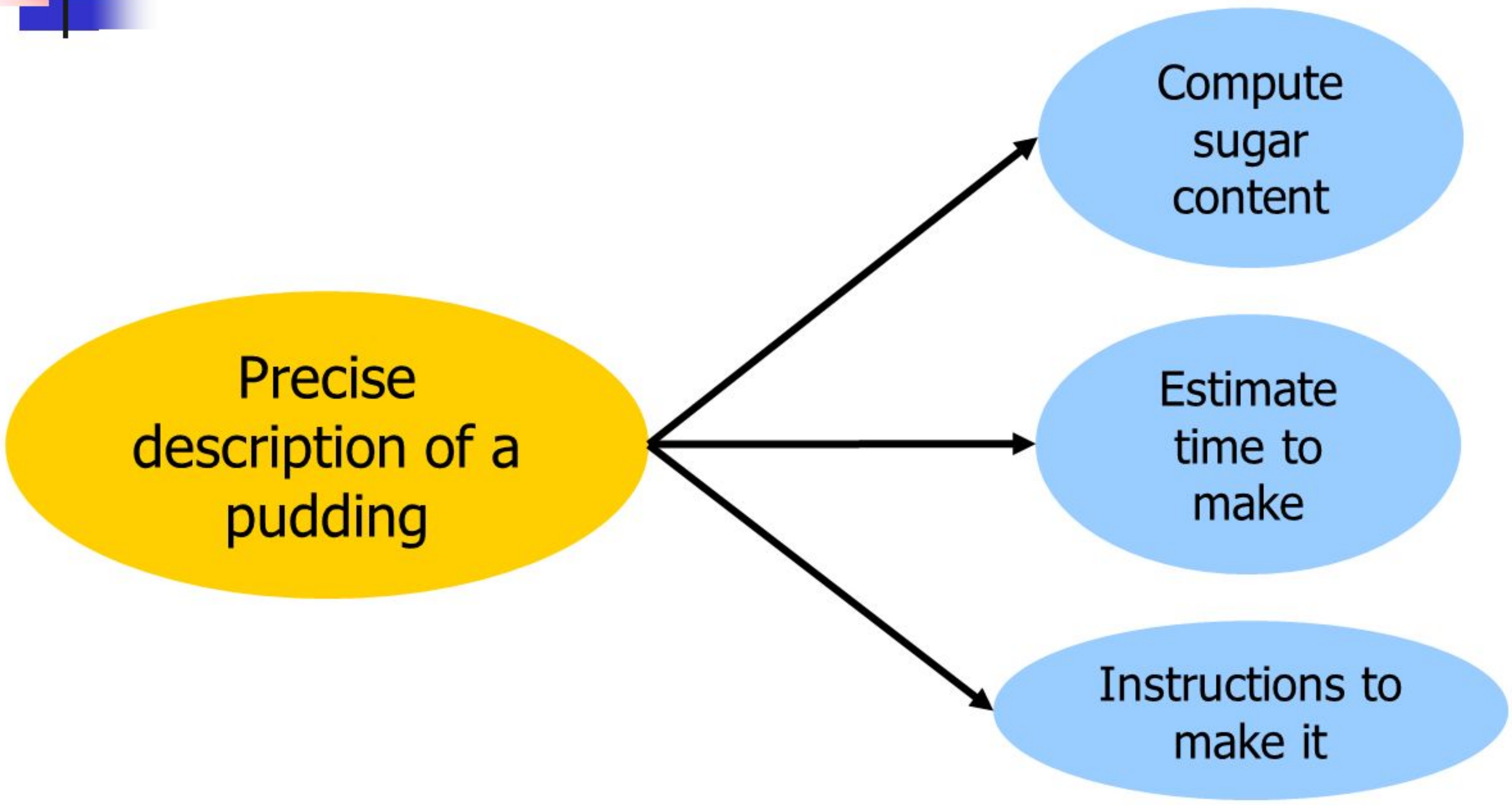
Composable REA Contracts

- Idea is from Modelling Financial contracts by Simon Peyton-Jones
- REA Contracts are a subset of the possible financial contracts
- REA contracts are normally about money for goods but can also be for services

The big picture

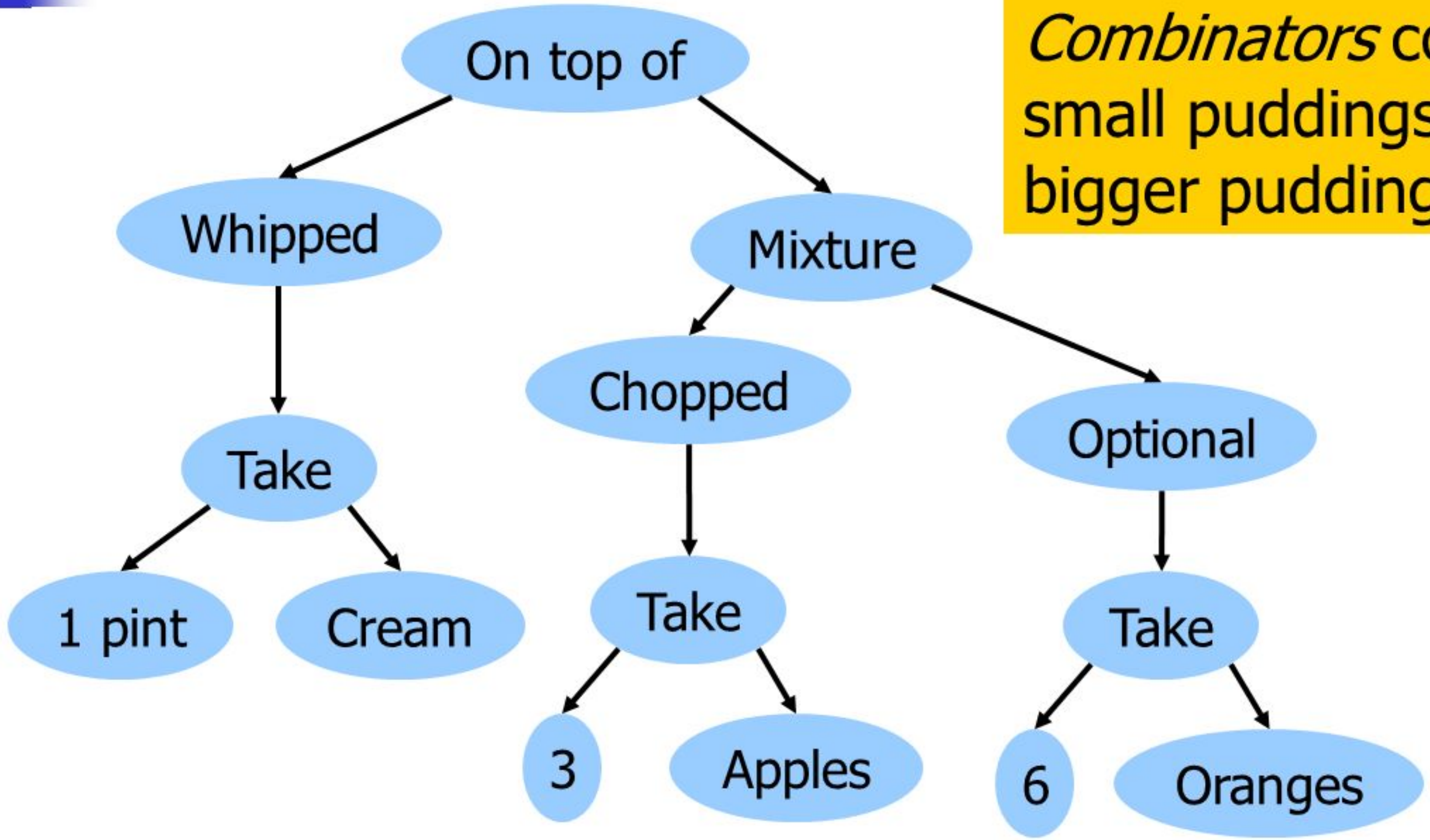


What we want to do



- etc….

Creamy fruit salad



Combinators combine small puddings into bigger puddings

Building a simple contract

"Receive £100 on 1 Jan 2010"

```
c1 :: Contract
c1 = zcb (date "1 Jan 2010") 100 Pounds
```

```
zcb :: Date -> Float -> Currency -> Contract
-- Zero coupon bond
```

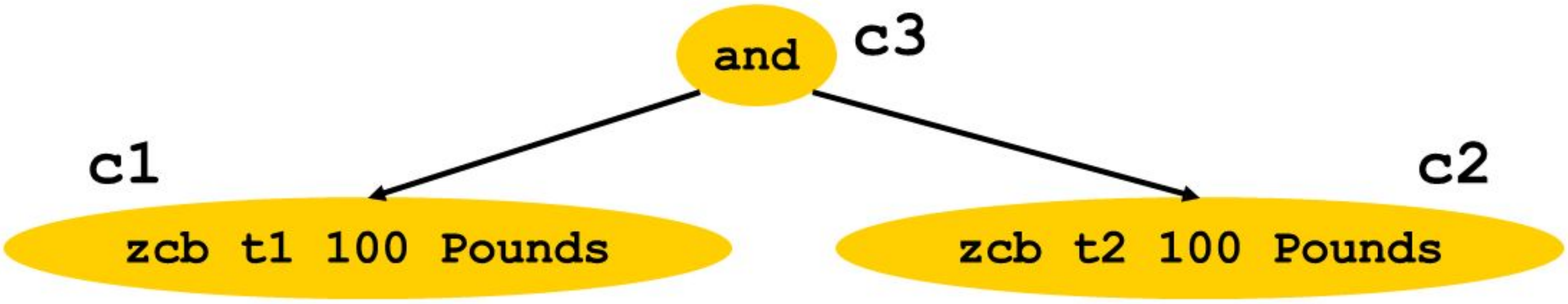
Combinators will appear in blue boxes

Combing contracts

```
c1,c2,c3 :: Contract
c1 = zcb (date "1 Jan 2010") 100 Pounds
c2 = zcb (date "1 Jan 2011") 100 Pounds

c3 = and c1 c2
```

```
and :: Contract -> Contract -> Contract
-- Both c1 and c2
```



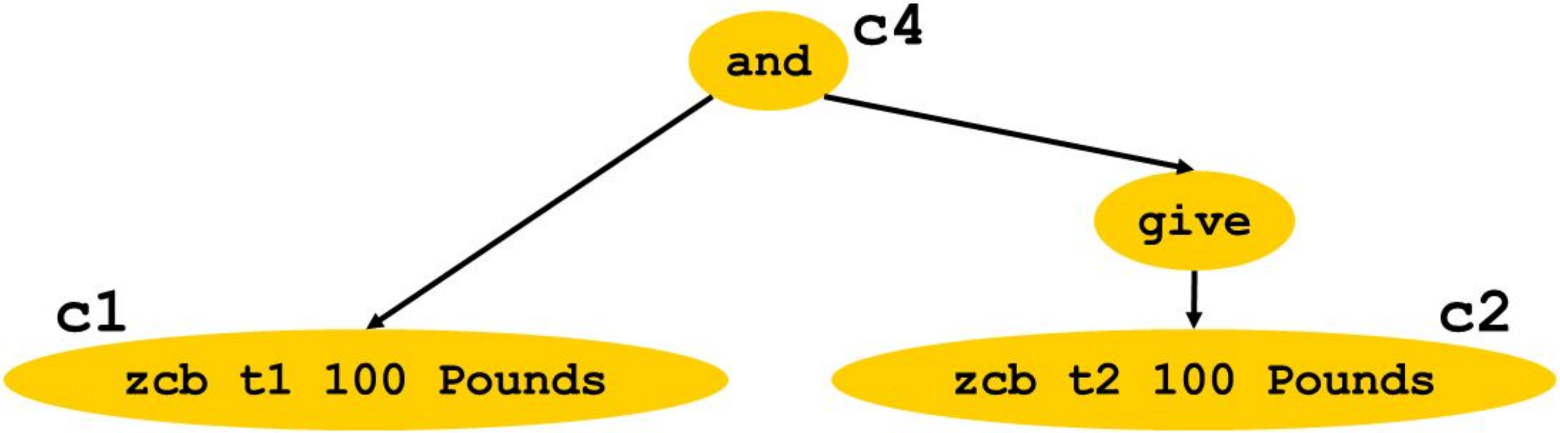
Inverting a contract

Backquotes for infix notation

```
c4 = c1 `and` give c2
```

```
give :: Contract -> Contract
-- Invert role of parties
```

- **`and`** is like addition
- **`give`** is like negation



New combinators from old

```
andGive :: Contract -> Contract -> Contract
andGive u1 u2 = u1 `and` give u2
```

- **`andGive`** is a new combinator, defined in terms of simpler combinators
- To the "user", **`andGive`** is no different to a primitive, built-in combinator
- This is the key to extensibility: **users can write their own libraries of combinators to extend the built-in ones**

Defining zcb

Indeed, **zcb** is not primitive:

```
zcb :: Date -> Float -> Currency -> Contract
zcb t f k = at t (scaleK f (one k))
```

```
one :: Currency -> Contract
-- Receive one unit of currency immediately

scaleK :: Float -> Contract -> Contract
-- Acquire specified number of contracts

at :: Date -> Contract -> Contract
-- Acquire the contract at specified date
```

Acquisition dates

```
one :: Currency -> Contract
-- Receive one unit of currency immediately

at :: Date -> Contract -> Contract
-- Acquire the underlying contract at specified date
```

- If you acquire the contract `(one k)`, you receive one unit of currency k **immediately**
- If you acquire the contract `(at t u)` at time `s`<`t`, then you acquire the contract `u` at the (later) time `t`.
- You cannot acquire `(at t u)` after `t`. The latest acquisition date of a contract is its **horizon**.

Choice

An **option** gives the flexibility to

- **Choose which** contract to acquire (or, as a special case, **whether** to acquire a contract)
- **Choose when** to acquire a contract (exercising the option = acquiring the underlying contract)

Choose **which**

- European option: at a particular date you may choose to acquire an "underlying" contract, or to decline

```
european :: Date -> Contract -> Contract
european t u = at t (u `or` zero)
```

```
or :: Contract -> Contract -> Contract
-- Acquire either c1 or c2 immediately

zero :: Contract
-- A worthless contract
```

Reminder…

Remember that the underlying contract is arbitrary

```
c5 :: Contract
c5 =  european t1 (european t2 c1)
```

This is already beyond what current systems can handle

Observables

Pay me $1000 * (the number of inches of snow - 10) on 1 Jan 2002

```
c :: Contract
c = at "1 Jan 2002" (scale scale_factor (one Dollar))

scale_factor :: Observable
scale_factor = 1000 * (snow - 10)
```

```
scale :: Observable -> Contract -> Contract
-- Scale the contract by the value of the observable
-- at the moment of acquisition

snow :: Observable
(*), (-) :: Observable -> Observable -> Observable
```

REA example contracts

- Prepayments:
  - At ”1 Jan 2017” receive ”Iphone 7” and (before ”12 December 2016” pay 125 US$ or before ”1 Jan 2017” pay 140 US$)
- Interests for late payments
  - Pay x or pay x*(100 + rate)/100 *(Date-duedate)/365
- Cash Discount
  - At deliverydate Pay x*(100-y)/100 at delivery or after deliverrydate pay x
- Installments
  - At date receive z and pay 5 times 100 US$ for 5 months

- We can now easily combine these options (and determine which combinations makes sense)

Summary

Routine for us, radical stuff for financial engineers

- A small set of built-in combinators: named and tamed
- A user-extensible library defines the zoo of contracts
- Compositional denotational semantics, leads directly to modular valuation algorithms
- Risk Magazine Software Product of the Year Prize
- Jean-Marc has started a company, LexiFi, to commercialise the ideas

Links

- http://research.microsoft.com/en-us/um/people/simonpj/Papers/financial-contracts/contracts-icfp.htm
- http://www.dslfin.org/resources.html
- https://www.fairmat.com/
- http://hiperfit.dk/