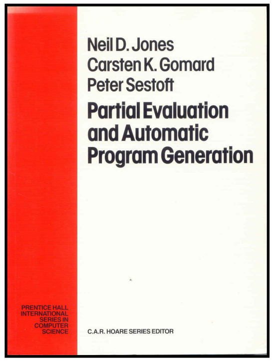
Domain-specific languages and GPGPUs in life insurance and pensions

Peter Sestoft
(presenting work by many people in the Actulus project, at Edlund, U Copenhagen, and ITU)

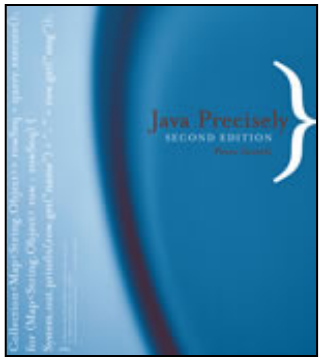
Parallel Functional Programming lecture 7
at Chalmers University of Technology
2015-04-24

The speaker

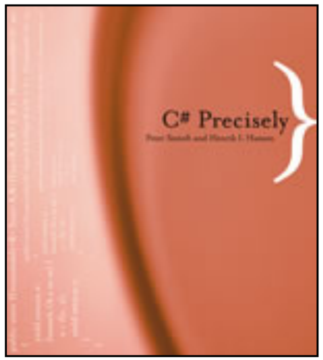
- MSc 1988 computer science and mathematics and PhD 1991, DIKU, Copenhagen University
- KU, DTU, KVL and ITU; and Glasgow U, AT&T Bell Labs, Microsoft Research UK, Harvard University
- Programming languages, software development, ...
- Open source software
  - Moscow ML implementation, 1994…
  - C5 Generic Collection Library, with Niels Kokholm, 2006…
  - Funcalc spreadsheet implementation, 2014



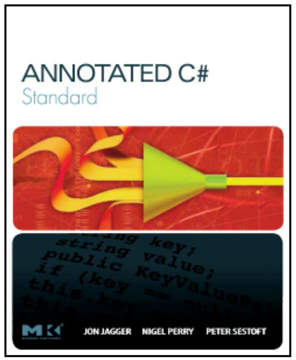
1993



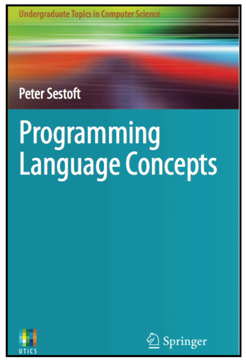
2002, 2005, 2016



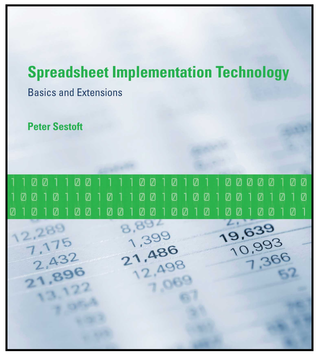
2004 & 2012



2007



2012



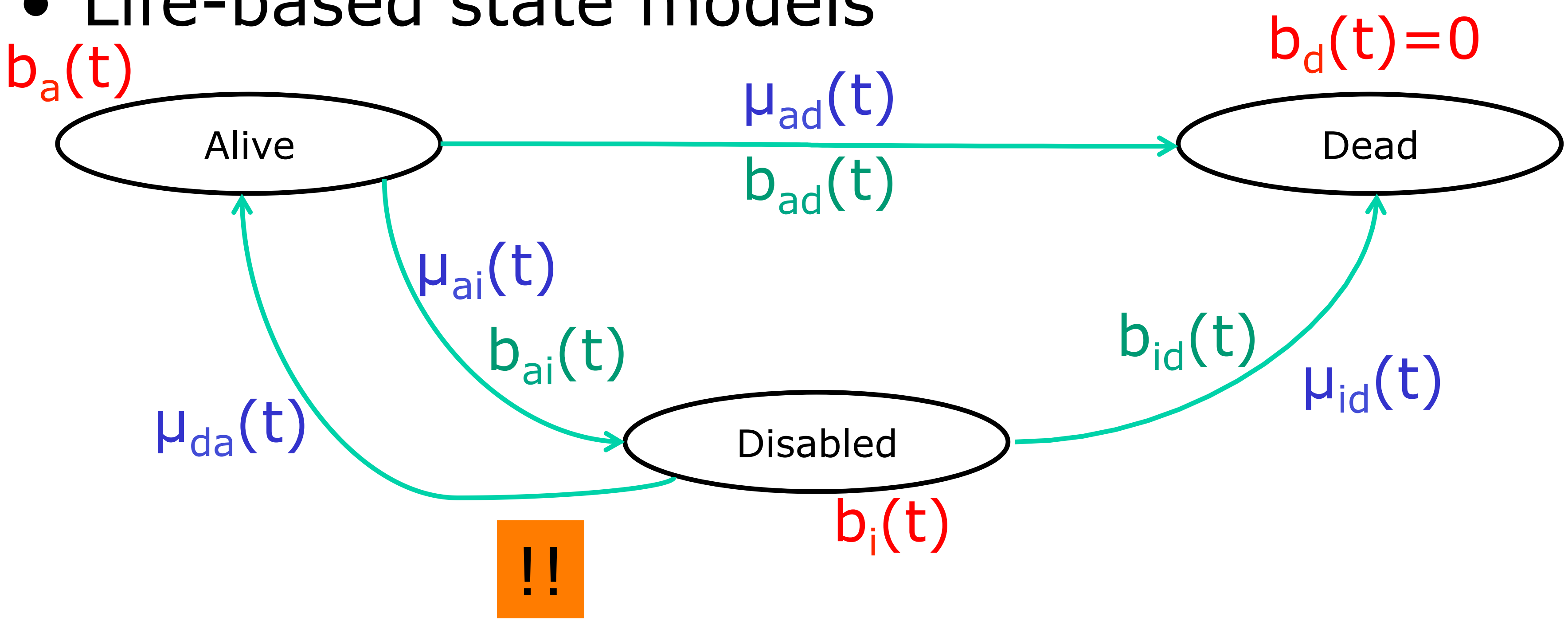
2014

Example life and pension products

- Pay 1m DKK on insured’s death, if before 65 (term life insurance, GF115)
- Pay 1m DKK on the day insured turns 65, if alive (pure endowment, GF125)
- Pay 200k DKK/year from age 65 while insured is alive (life annuity, GF211)
- Pay 200k DKK/year while insured is disabled, alive, and not yet 65 (disability insurance, GF415)
- If insured dies before 65 years, pay 100k DKK/year to spouse, if any, while alive (spouse pension, GF810)
- NB: Conditioned on insured’s **life**, unlike private savings (that pass to the estate)

Formalizing pension contracts

- Life-based state models



- States
- Transition intensities, eg mortality rate $\mu_{ad}(t)$
- Payment in state $b_a(t)$ eg. "while t>65 ..."
- Payment on transitions $b_{ad}(t)$: "upon death ..."

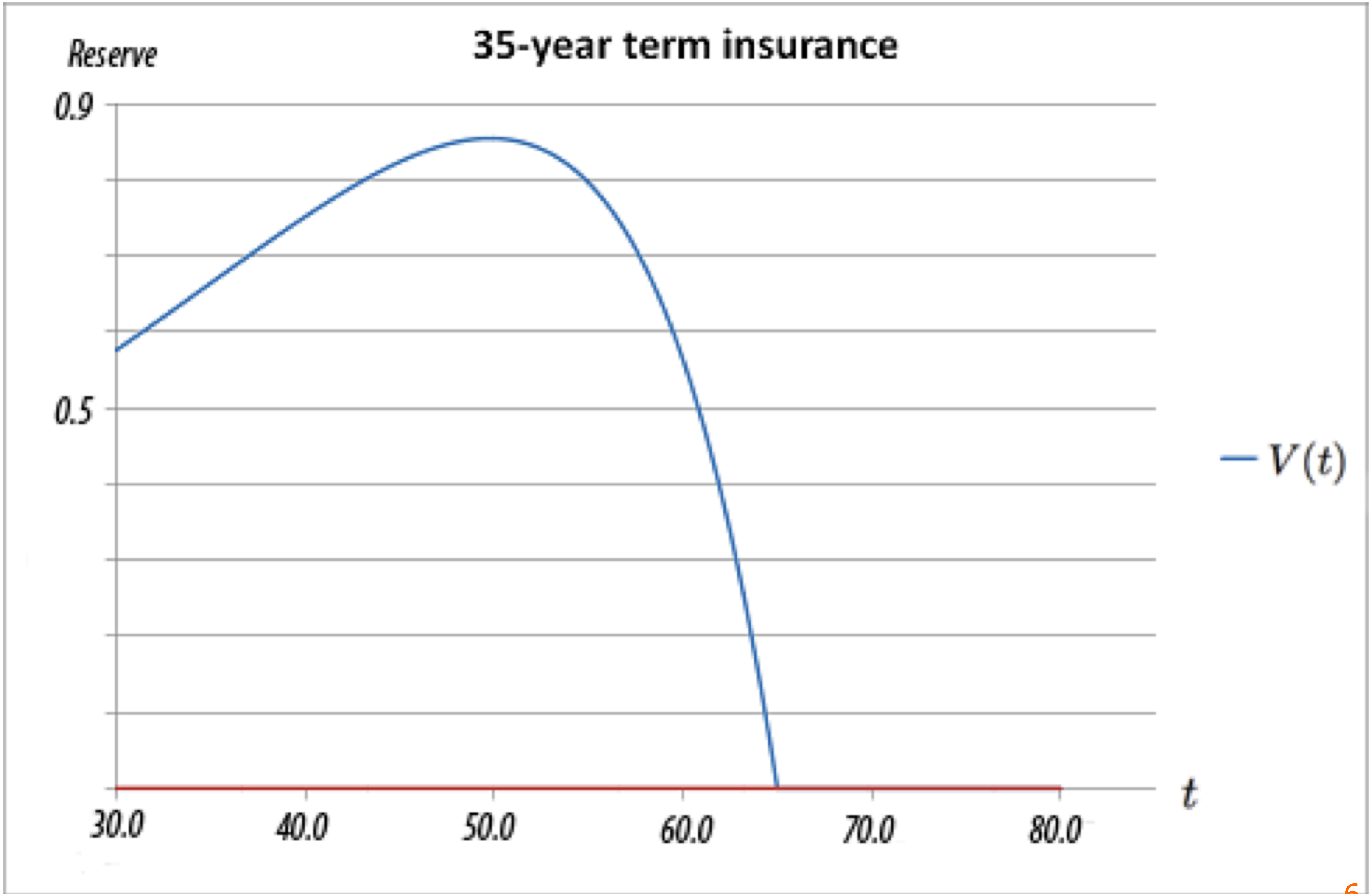
Some basic concepts

- Prospective reserve
  - the net present value of expected future payments
- Discount rate
  - 1 DDK today is worth r = 1.02 DDK a year from now
- Mortality
  - Insured people die with a certain intensity (probability)

- Possible shocks, simulation scenarios:
  - Interest rate: assume adverse (lower) discount rates, that is, less future interest earned on current funds
  - Mortality rate: assume adverse mortality rates, eg. a cure for cancer (lower mortality), or a natural catastrophe or epidemic (higher mortality)

Reserve graph for Term Insurance

Pay 1 krona to insured if dies before 65



The Actulus project, partners

- **Edlund A/S**
  - software for the Danish pensions and life insurance industry (PFA, ATP, Nordea, Skandia, ...)
  - 200 people, math, CS, actuaries, many PhDs
- **Department of Mathematical Sciences** at Copenhagen University
  - actuarial mathematics and numerical algorithms
- **IT University of Copenhagen**
  - programming language technology, domain-specific languages, parallel programming
  - David Christiansen, Peter Sestoft, BSc and MSc students
- Funding: **Advanced Technology Foundation**
- Project duration: April 2011 to March 2016

Goals of the Actulus project

- Overall: *Establish a platform for definition of advanced life insurance and pension products and for efficient computations on them.*

- **But why?**
- Huge societal importance in Denmark
  - Pension provisions 1,756bn DKK = 97% of GDP
  - Net increase is 50bn DKK per year (2013)
- Very long-term: Most contracts entered today (25 yr woman) will still be in force in 2070

And why now?

- Regulation: EU Solvency 2
  - Single market for insurance in Europe (2016)
  - Stronger requirements on risk evaluation
    - Probability of 1-year insolvency: $P(A - R < 0) < 0.5\%$
  - Stronger transparency requirements
- To Edlund, a challenge and an opportunity
  - Can they go beyond the domestic market?
- Technological opportunities
  - GPGPUs allow fast numerical solution of risk models
    - Traditional closed-form formulas no longer suffice
  - Domain-specific languages for products and risk
  - Code generation from those languages

Why in Denmark?

- On sustainability of pension provisions:

> ”We are the country that’s best prepared. In general, Sweden, Finland and the Netherlands are quite well prepared, but we are even further ahead. [...] in today’s Europe you will not find a better situation."

Allan Polack, chair of working group on future pension systems in Europe, quoted in Nordea Private Banking magazine, March 2013

- DK: Wide coverage, strong regulation, strong formalization, collaboration and competition

Actulus Modeling Language, A domain-specific language

- What?
  - Notation specially designed for the application area
  - Supported by tools: editors, checkers, compilers ...
- A state model: insured is alive or dead

```
statemodel LifeDeath where
  states      = alive
              | dead
  transitions = alive -> dead
```

- A contract: Pay 200000/year from year n while insured is alive

```
contract GF211(n : TimePoint) : LifeDeath where
  obligations =
    at t pay 200000 per year provided(n < t and alive)
```

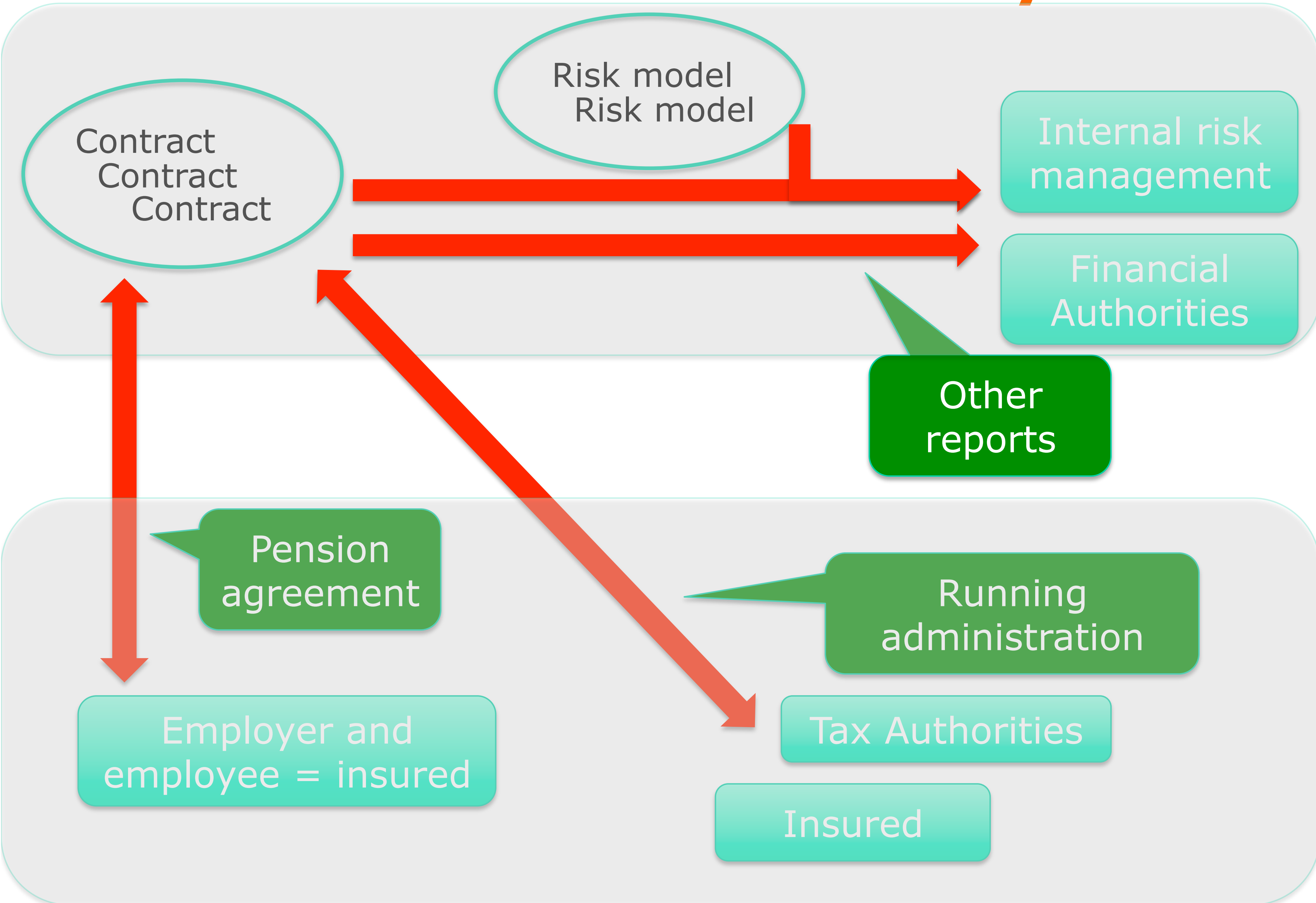
Actulus domain-specific languages

- *Actulus modeling language* (AML)
- *AML Product*, to describe pension products and life insurance products
  - A *contract* (p,dp) is a product p and product-specific data dp, eg policy holder's age etc
- *AML Computation*, to describe computations
  - on individual contracts, and
  - on portfolios (collections of contracts)
  - under assumptions about interest rate developments, mortality rates, and more
  - should permit a range of different risk models
- *AML Admin*, to describe payments, reporting

Why a domain-specific language?

- Advantages
  - Can assign different semantics to same "text"
  - Eg product as basis for cashflow management and calculating reserves *and* solvency *and* ...
  - Declarative, easier to understand and process, than imperative/object-oriented code
  - Version control and traceability
  - Supports evolution of core system independently of company-specific adaptations (long-term)
  - Reduces technology dependence (long-term)
- Disadvantages
  - Must design, implement, maintain the language
- Yet, a powerful tool and business model
  - Witness Axapta, Navision, Maconomy, ..

An AML-P contract has many uses



AML language design

- Possible features of *Actulus Modeling Language*
  - compositional – simple products, composite products
  - declarative – say what holds, not how it is achieved
  - strongly typed, to catch errors early
  - dimension types, to prevent confusion of time, rates, probabilities, money and other numeric quantities
  - dependent types, to prevent confusion of lives
  - linear types, to prevent duplication of payments

- Related work, inspiration
  - Peyton-Jones, Eber, Seward 2003: Financial contracts
  - Mogensen 2003: Linear types for cashflow reeng.
  - van Deursen et al 1995: Risla
  - Gaillourdet 2011: a language approach to derivatives
  - ...

AML-P language design story

- Whitepapers, actuarial theory and notation, ...
  - Quite foreign to computer science people
  - Lots of interaction, help from Mogens and Edlund
  - Project wiki essential for notation and development
- David Christiansen proposed AML-P designs
  - Lots of feedback
- Edlund people have developed it further
  - Checked G82 coverage
  - Proposed revised syntax
  - Partially implemented
  - Type is system being developed by David

Efficient computation

- AML-P admits complex contracts and state models with cycles
- So the Thiele differential equations for the reserve do not have closed form solutions
- Hence necessary to solve them numerically

- Some differential equation solvers
  - Runge Kutta 4th order (RK4)
  - Runge Kutta Fehlberg 4th/5th order (RKF45)
- Use graphics processors, GPU/Nvidia CUDA?

Thiele's differential equations

- Solve this system of differential equations:

$$\frac{d}{dt}V_j(t) = r(t)V_j(t) - b_j(t) - \sum_{k,k\neq j} \mu_{jk}(t)(V_k(t) - V_j(t) + b_{jk}(t))$$

- Think $V_j(t)$ = funds held for insured in state j
- Accrue interest at rate r(t)
- Pay out benefits to insured at rate $b_j(t)$
- Insured transits j -> k with intensity $\mu_{jk}(t)$
  - And if so we pay out benefits $b_{jk}(t)$ to insured
  - And the state j funds decrease by the difference between the to-state and the from-state funds

Solving Thiele's equations

$$\frac{d}{dt}V_j(t) = r(t)V_j(t) - b_j(t) - \sum_{k,k\neq j} \mu_{jk}(t)(V_k(t) - V_j(t) + b_{jk}(t))$$

- Eg prospective reserve is $V_a(0)$
- Solution approach:
  - Set $V_i(120) = 0$, everybody is dead at 120
  - Solve backwards from t=120 to 30, for instance
  - Obtain graph of reserve over time
  - But: there are discontinuities in r(t) and $b_j(t)$
    - Due to Financial Authority ZCB-based interest rates
    - Due to age-dependent benefits etc
    - Worse, lump sum payments in $b_j(t)$ via Dirac function

Numerical solution of Thiele's differential equations

- Simple Runge-Kutta 4 solver
  - Fixed time-steps, 4th order convergence
  - Easy to implement
  - Easy to handle discontinuities ("when t=65 ...")
  - Little thread divergence on GPGPU
  - One can pre-interpolate interest rate curves
- Experiments with adaptive-step solvers
  - Eg. Runge-Kutta-Fehlberg 4/5, Dormand-Prince
  - In many applications they are much faster
    - But here discontinuities slow them down
  - Likely to have thread divergence on GPGPU
- Conclusion so far: Use Runge-Kutta 4 solver

Runge-Kutta 4 code excerpt (C#)

- Very simple core solver

```
for (int y=a; y>b; y--) {
  double[] v = result[y-b];
  v = daxpy(1.0, v, bj_ii(y));
  double t = y;
  for (int s=0; s<steps; s++) {      // Integrate from y to y-1
    double[] k1 = dax(h, dV(t, v));
    double[] k2 = dax(h, dV(t + h/2, daxpy(0.5, k1, v)));
    double[] k3 = dax(h, dV(t + h/2, daxpy(0.5, k2, v)));
    double[] k4 = dax(h, dV(t + h,   daxpy(1.0, k3, v)));
    v = daxpy(1/6.0, k1, daxpy(2/6.0, k2,
                         daxpy(2/6.0, k3, daxpy(1/6.0, k4, v))));
    t += h;
  }
  Array.Copy(v, result[y-1-b], v.Length);
}
```

Why GPGPU, General purpose graphics processor?

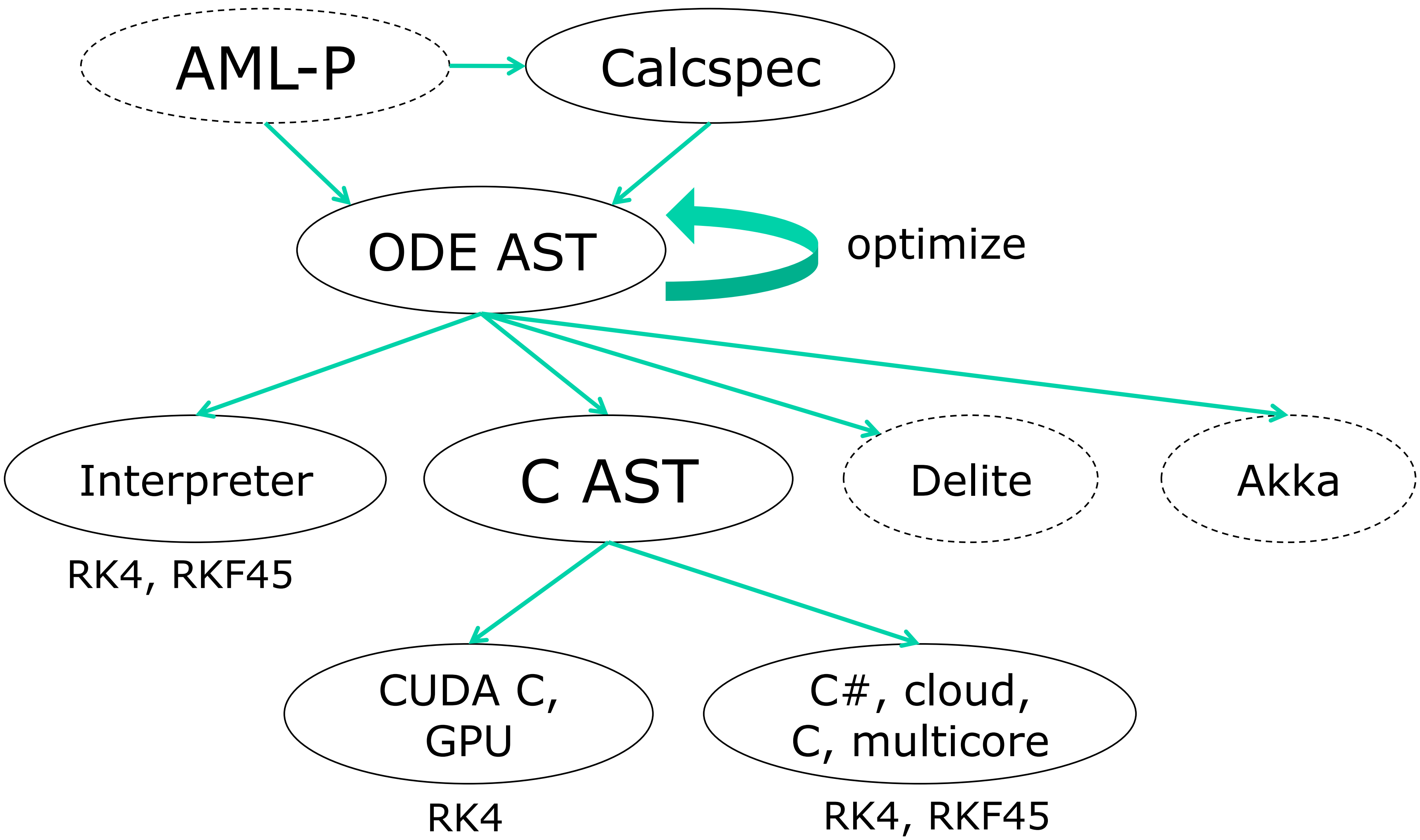
- A modern CPU, eg Intel Core i7, has
  - A few complex cores at 2-3 GHz, each superscalar
  - Deep instruction pipeline, out-of-order execution ...
  - Good for unpredictable mixed compute loads
  - Much "management", little "compute work"
- A GPGPU, eg Nvidia, by contrast, has
  - Many (50-1500) simple compute cores, at <1 GHz
  - No pipelines, no out-of-order execution, etc
  - User-managed memory hierarchy
  - Good for predictable data parallel compute loads
  - Little "management", much "compute work"

16/14 SM x 32 cores = 512/448 cores

Challenges in using GPU

- Multiple kinds of memory
  - On host (CPU):
    - RAM, inaccessible to GPU
  - On device (GPU):
    - Global memory, shared by all blocks (large, slow)
    - Constant memory, shared by all blocks, readonly
    - Shared memory, shared by threads in block
    - Registers, local to thread (small, fast)
  - Bottleneck is *data transport*, not computation
- Thread divergence
  - Operations must proceed in lock-step (as in RK4)
  - If not, many cores may be idle (adaptive solvers)

Dahl & Harrington Scala-based computation framework



Collective life insurance products

- Eg spouse pension, GF810:
  "If insured dies before 65 years, pay 100k DKK/year to spouse, if any, while alive"
  = Give spouse a life annuity at insd's death
- Spouse at time of death is assumed unknown when contract is made
- Reserve computation needs three integrations
  - Over insured's death intensity
  - Over marriage probability and spouse's age
  - Over spouse's death intensity
- Very compute intensive, 30 CPU sec/contract
- Unsuited for GPGPU (adaptive, subroutines)

GF810 by triple integration

- Outer: insured's life

$$\frac{d}{dt}f(t) = r(t)f(t) - \mu_t(x+t)(S^d_{x+t}(t) - f(t))$$

- Middle: spouse age distribution

$$S^d_\tau(t) = \ g_\tau \int h(\eta|\tau) a^I_{[\eta]+k}(t) d\eta$$

- Inner: spouse's life, where $a^I_{[\eta]+k}(t)$ is the solution to

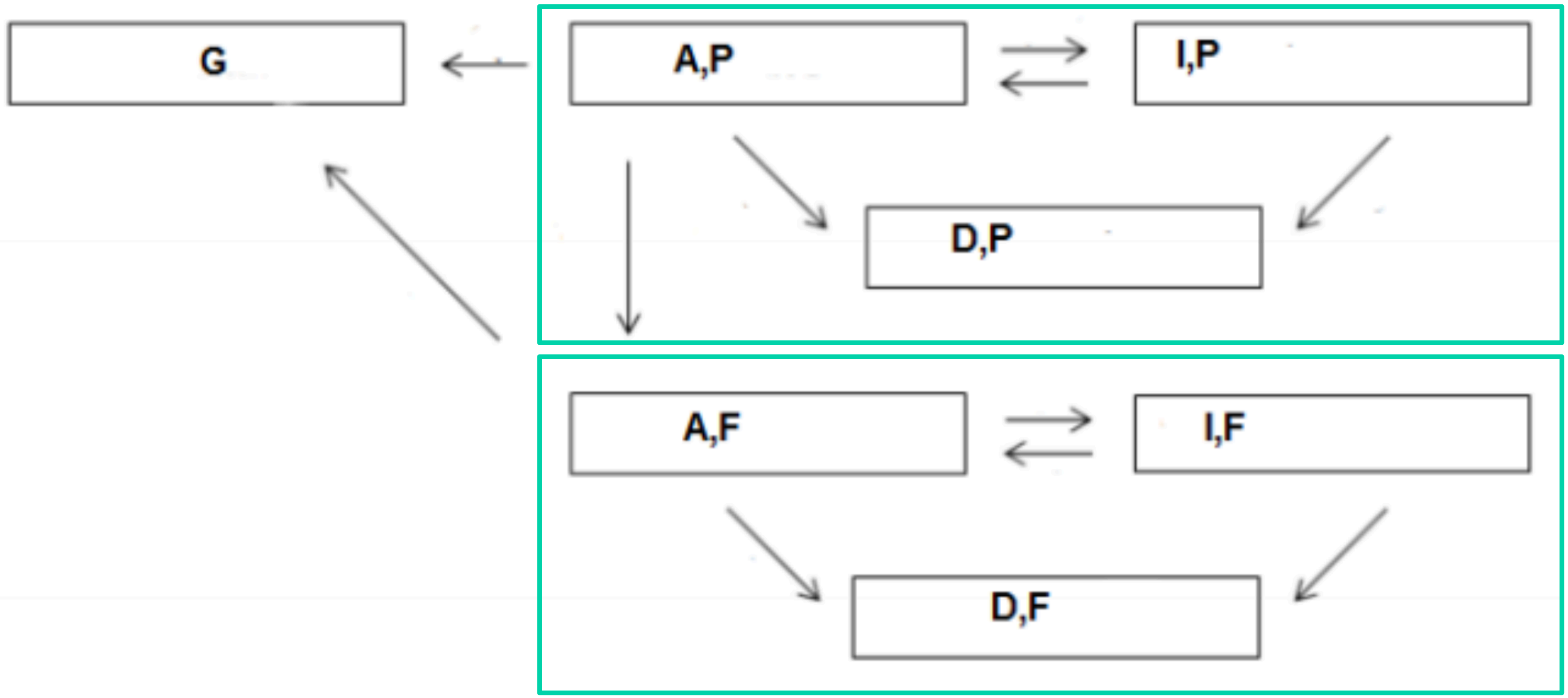
$$\frac{d}{ds}f(s) = r(t+s)f(s) - 1_{s \geq k} - \mu_{t+s}(\eta + s)(0 - f(s))$$

The technology used

- ODE generation and model expansion in F#
- Solver kernels written in F#
  - Using Alea GPU from www.quantalea.com
- Pros
  - Excellent performance
  - Much less hassle than CUDA C code
  - Functional-style F# to generate the kernels
- Cons
  - Imperative-style F# for the generated kernels
  - Must still ponder memory use and access patterns
  - (And MS Azure cloud has no GPU instances)

Lots of additional aspects 1

- "Seven state model"
  - Surrender: "I move to NZ, give me my money"
  - Free policy: "I'm jobless and cannot pay right now but would like to keep my pension savings"



Lots of additional aspects 2

- Technical reserve vs market reserve
- Cashflows: Temporal distribution of insurer's payments (expected investment horizon etc)
  - Compute by forward solution of Thiele-like ODEs
  - NB: reserve = integral of discounted cashflows
- Financial Authorities interest rate curve
- Benchmark mortalities and
- Policy holder behavior (surrender, retirement)
- Stochastic simulation of shocks
  - Interest rate goes up or down
  - Mortalities go up or down

Current state, and plans

- Product sold: Actulus Portfolio Calculator APC
  - Has been developed, is being marketed and bought
  - Written in C#, runs in MS Azure cloud
  - Does not use AML or GPUs
  - *Many* CPU hours for a portfolio reserve (eg 100k policies)
- Plans on the 1-year horizon
  - Continue and finalize design of AML Product DSL
  - Use AML Product in APC
  - Generate ODEs and solvers from AML Product files
  - Use GPU computations for APC
- Longer term
  - Design AML Admin
  - Stochastic retirement, simulations, shocks, ...
  - Customer advice, utility maximization, ...

David Christiansen’s PhD project

- Initial AML design
- Embedded domain-specific languages in Idris
  - dependently typed programming
  - main Idris architect is Edwin Brady, St Andrews
- Type providers (à la F#) in Idris
- Error reflection – clearer DSL error messages
- Quasiquotations in Idris
- Generating Idris declarations from DSL terms
  - Also supports type providers in a new way
- Now applying this to AML type system design

References

- Christiansen, Grue, Niss, Sestoft, Sigtryggsson: *An actuarial programming language for life insurance and pensions*, Intl. Congress Actuaries 2014, http://www.itu.dk/people/sestoft/papers/amlp.pdf
- Harrington, Dahl, Sestoft, Christiansen: *Pension reserve computations on GPUs, FHPC 2014* http://dl.acm.org/citation.cfm?id=2636230&dl=ACM
- The Actulus project
  - www.actulus.dk
- Edlund A/S
  - www.edlund.dk
- Mogens Steffensen
  - www.math.ku.dk/~mogens/
- Peter Sestoft
  - www.itu.dk/people/sestoft

Ad: PhD project *Declarative parallel programming on multicore machines*

- Design, prototype and evaluate well-performing parallel implementations of high-level declarative programming languages.
- These languages may be based on dataflow concepts (including extensions of spreadsheet-style computations), array programming, or other declarative concepts.
- The implementations will primarily target shared-memory multicore machines, through high-level with garbage collection.