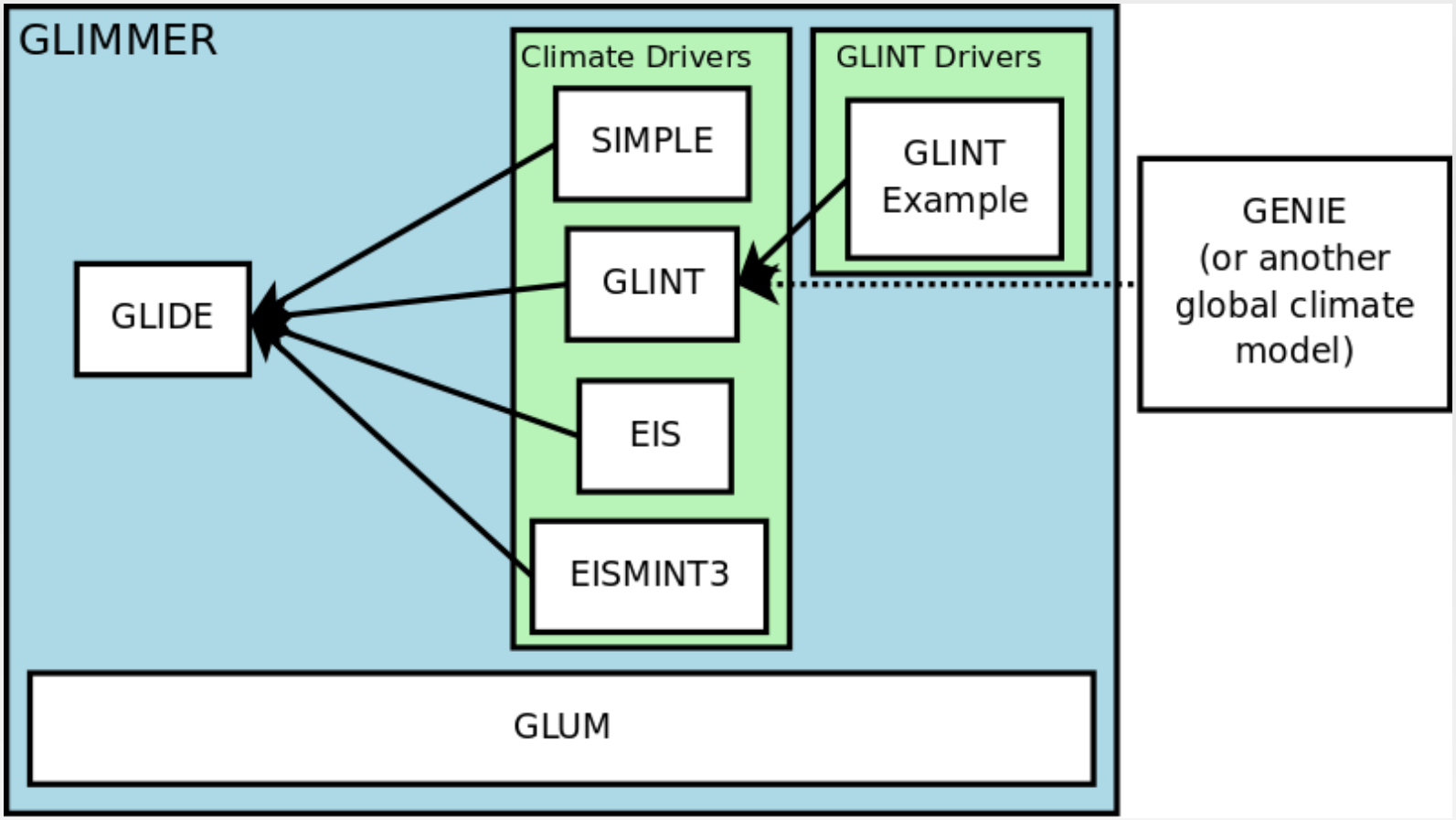
Introduction to Glimmer: part 1

- Shallow Ice Approximation
- Based on model by Tony Payne (pub. 1999)
- Developed into Glimmer as part of the GENIE Earth System Model (2003 onwards)
- Code released under GPL
- Tested against EISMINT and Bueler Isothermal
- Adopted as land ice model of CCSM
- Combined project: Glimmer-CISM (2009)

Introduction to Glimmer: part 1

- Modular design
- F95 standard
- NetCDF I/O with CF metadata
- Uses standard Linux tools
- Some code autogenerated
- Consistent version numbering
- Stable API
- Well-documented

Structure



- GLIDE: the core model (**GL**immer **I**ce **D**ynamics **E**lement)
- GLINT: the climate model interface (**GL**immer **INT**erface)

Equations solved by GLIDE

- Continuity Equation:

$$\frac{\partial H}{\partial t} = -\nabla \cdot (\bar{\mathbf{u}} H) + b - S$$

- Shallow Ice Velocities:

$$\bar{\mathbf{u}} = -\frac{2}{H}(\rho_i g)^n |\nabla s|^{n-1} \nabla s \int_h^s \int_h^z A(s - z')^n dz' dz + \mathbf{u}(h)$$

Equations solved by GLIDE

- Continuity Equation:

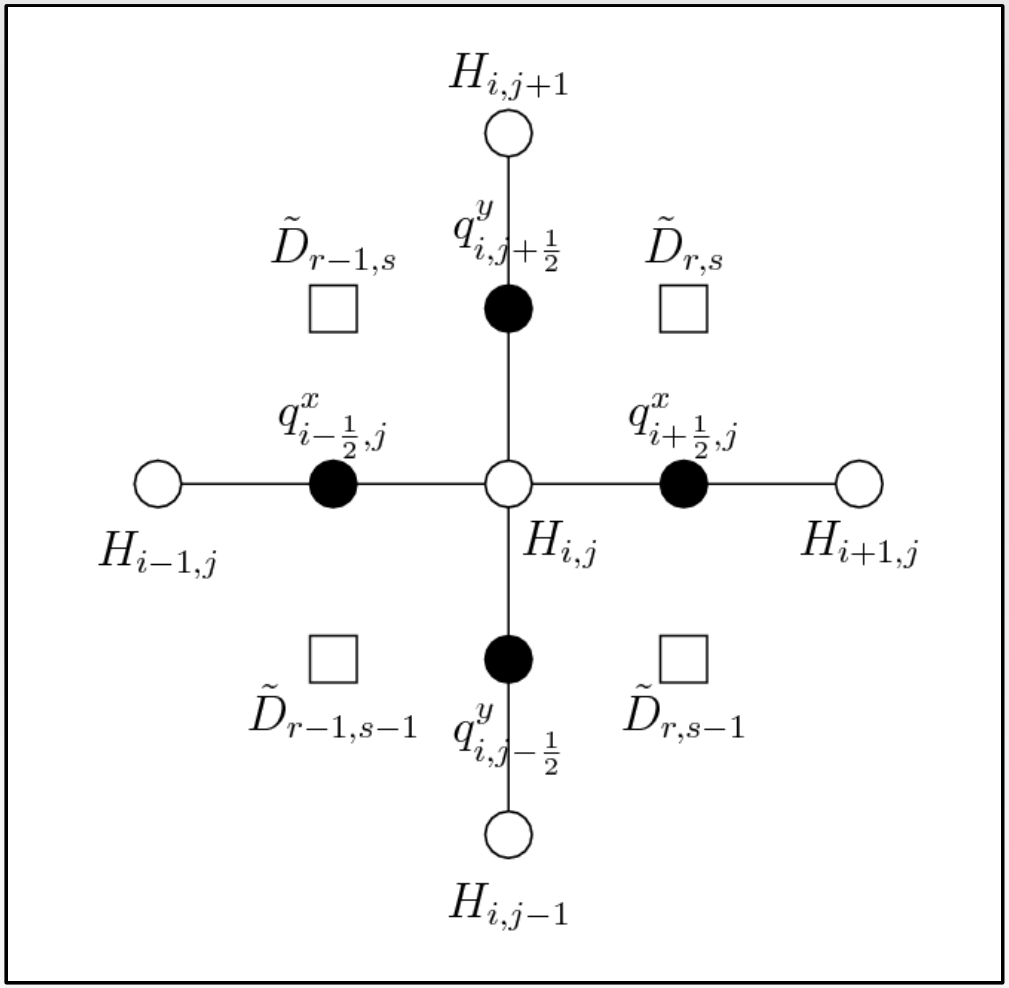
$$\frac{\partial H}{\partial t} = -\nabla \cdot (D \nabla s) + b - S$$

$$\boldsymbol{q} = D \nabla s$$

- Shallow Ice Diffusivities:

$$D = -2(\rho_i g)^n |\nabla s|^{n-1} \int_h^s \int_h^z A(s - z')^n dz' dz - B \rho_i g H^2$$

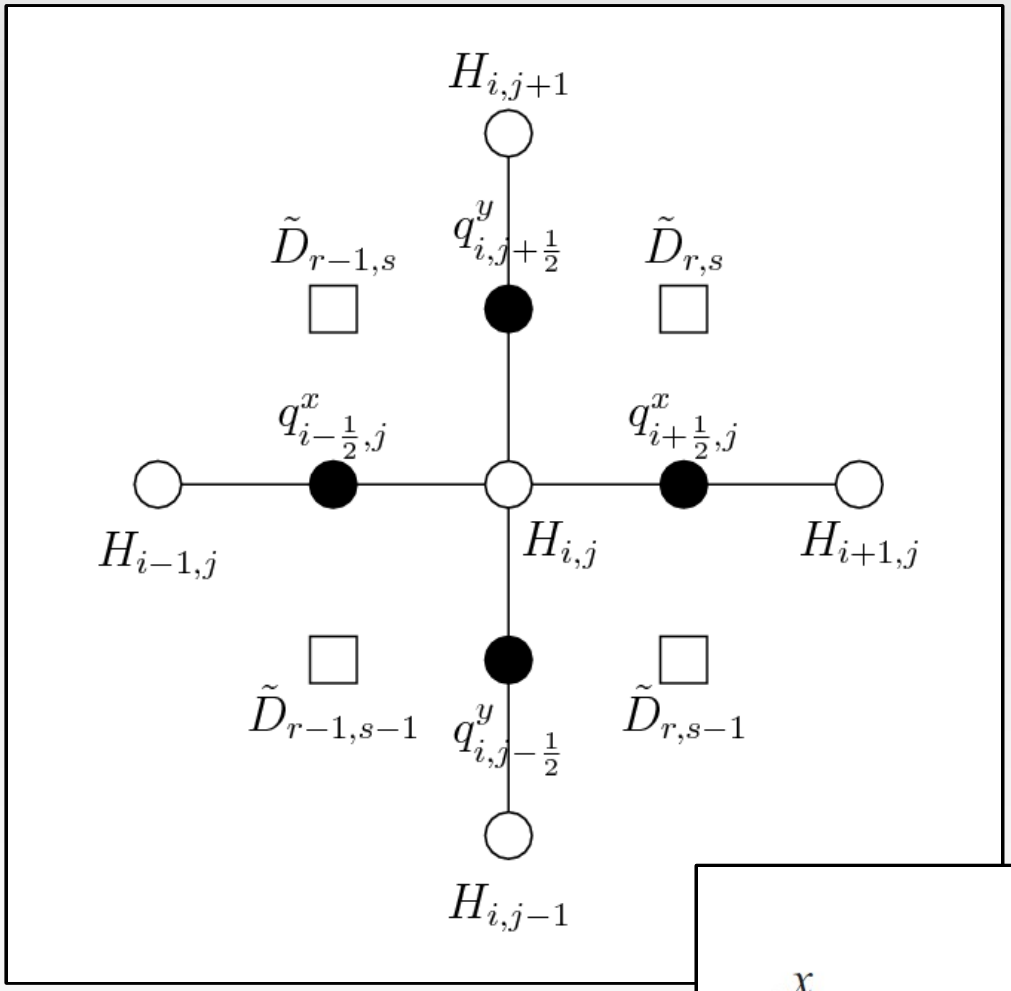
Horizontal Discretization



- Velocity and diffusivity calculated on staggered grid
- Flux (q) is calculated at point between thickness points
- This is the same principle as the Arakawa C-grid (1977)

$$q = D \nabla s$$

Horizontal Discretization



$$\frac{\partial H}{\partial t} = -\nabla \cdot (D \nabla s) + b - S$$

$$q^x_{i+\frac{1}{2},j} = -\frac{1}{2}\left(\tilde{D}_{r,s} + \tilde{D}_{r,s-1}\right)\frac{s_{i+1,j} - s_{i,j}}{\Delta x}$$

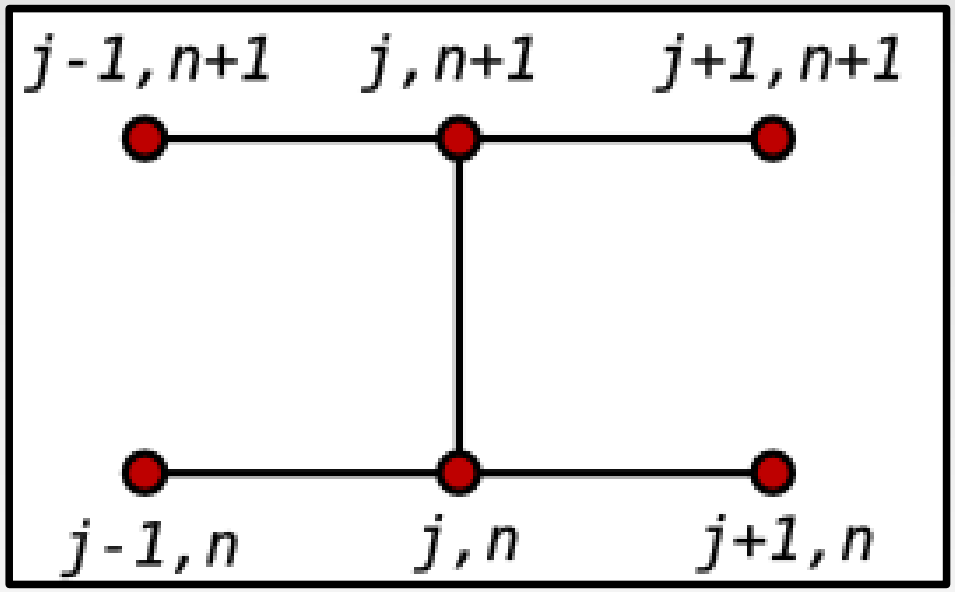
How do we solve this?

- Some level of implicitness is needed for stability...
- Equations are non-linear, because *s* (i.e. *H*) appears in *D*...
- Two distinct methods are implemented:
  - Alternating Direction Implicit (ADI)
  - Semi-implicit (Crank-Nicolson)

Crank-Nicolson Method

- Evaluated as a mean of two time-steps (at n+½)

$$\frac{\partial H}{\partial t} = -\nabla \cdot (D \nabla s) + b - S$$



Crank-Nicolson Method

- Linear scheme uses *D* at current time step:

$$q^{x,t+1}_{i+\frac{1}{2},j} = -\frac{1}{2}\left(\tilde{D}^{t}_{r,s} + \tilde{D}^{t}_{r,s-1}\right)\frac{s^{t+1}_{i+1,j} - s^{t+1}_{i,j}}{\Delta x}$$

$$q^{x,t}_{i+\frac{1}{2},j} = -\frac{1}{2}\left(\tilde{D}^{t}_{r,s} + \tilde{D}^{t}_{r,s-1}\right)\frac{s^{t}_{i+1,j} - s^{t}_{i,j}}{\Delta x}$$

$$\frac{H^{t+1}_{i,j} - H^{t}_{i,j}}{\Delta t} = \underbrace{\frac{q^{x,t+1}_{i+\frac{1}{2},j} - q^{x,t+1}_{i-\frac{1}{2},j}}{2\Delta x} + \frac{q^{y,t+1}_{i,j+\frac{1}{2}} - q^{y,t+1}_{i,j-\frac{1}{2}}}{2\Delta y}}_{\text{Forward time step}} + \underbrace{\frac{q^{x,t}_{i+\frac{1}{2},j} - q^{x,t}_{i-\frac{1}{2},j}}{2\Delta x} + \frac{q^{y,t}_{i,j+\frac{1}{2}} - q^{y,t}_{i,j-\frac{1}{2}}}{2\Delta y}}_{\text{Current time step}} + b_{i,j} - S_{i,j}$$

Crank-Nicolson Method

- Leads to a system of equations we can solve using iterative methods:

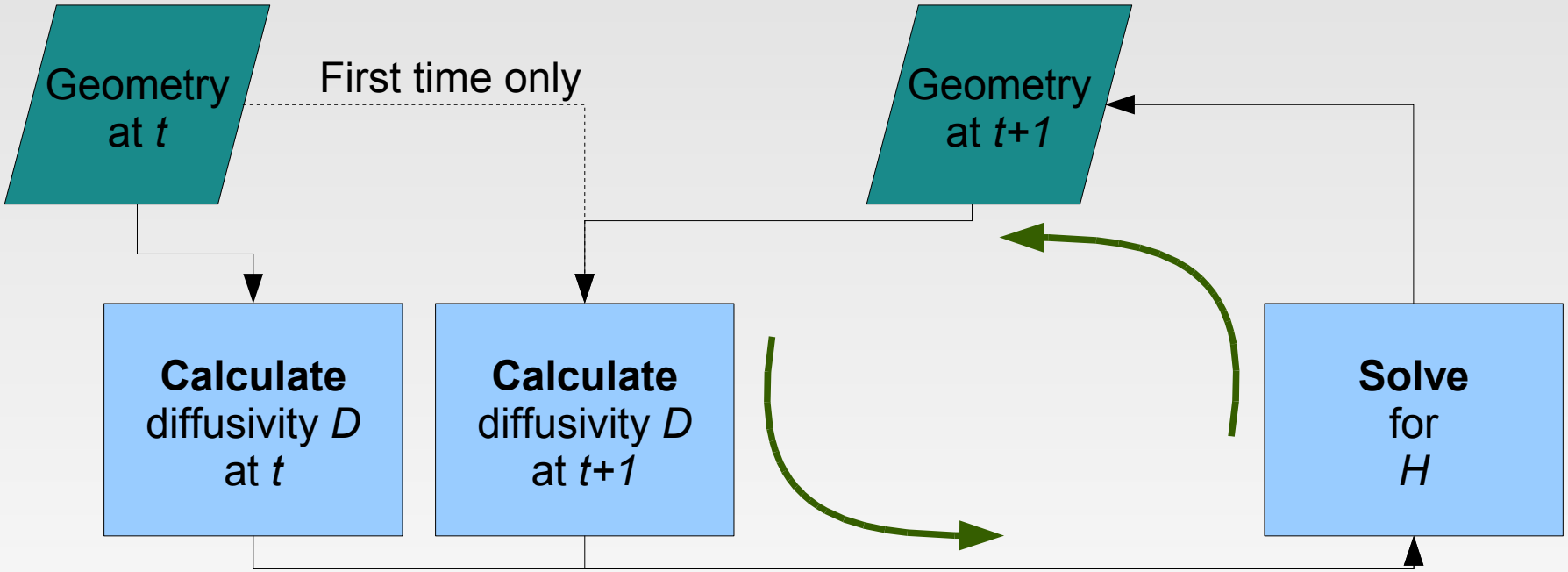
$$-\alpha_{i,j}H_{i-1,j}^{t+1} - \beta_{i,j}H_{i+1,j}^{t+1} - \gamma_{i,j}H_{i,j-1}^{t+1} - \delta_{i,j}H_{i,j+1}^{t+1} + \left(1 - \epsilon_{i,j}\right)H_{i,j}^{t+1} = \zeta_{i,j}$$

Unknown

Known

Non-linearity

- Deal with non-linearity using a Picard iteration



Perform the loop until the geometry at *t+1* stops changing significantly

Solving for Temperature

- **Basic temperature equation:**

$$\frac{\partial T}{\partial t} = \frac{k}{\rho_i c}\left(\nabla^2 T + \frac{\partial^2 T}{\partial z^2}\right) - \mathbf{u}\cdot\nabla T + \frac{\Phi}{\rho_i c} - w\frac{\partial T}{\partial z}$$

Diffusion (horizontal and vertical)

Horizontal advection

Internal heat generation

Vertical advection

Vertical discretization

- Two problems:
  - Temperature tends to change most rapidly at the base of the ice – equal spacing of levels not appropriate
  - Thickness of ice changes, so fixed physical spacing doesn't work - levels would move in and out of ice
- Solution:
  - Introduce a new vertical coordinate, scaled by the ice thickness
  - Use unequally-spaced levels

Vertical discretization

- Sigma coordinates:

$$\sigma = \frac{s - z}{H}$$

So, sigma coordinates run between 0 (ice surface) and 1 (bed)

This means we have to transform all our coordinates:

$$x, y, z, t \rightarrow x', y', \sigma, t'$$

Vertical discretization

- Mainly affects derivatives:

$$\frac{\partial f}{\partial t} = \frac{\partial f}{\partial t'} + \frac{1}{H}\left(\frac{\partial s}{\partial t} - \sigma\frac{\partial H}{\partial t}\right)\frac{\partial f}{\partial \sigma}$$

$$\nabla f = \hat{\nabla} f + \frac{1}{H}(\nabla s - \sigma \nabla H)\frac{\partial f}{\partial \sigma}$$

Vertical discretization

- Mainly affects derivatives:

$$\frac{\partial f}{\partial z} = -\frac{1}{H}\frac{\partial f}{\partial \sigma}$$

More detail in Pattyn (2003), and Hindmarsh and Hutter (1988)

Transformed Temperature

$$\frac{\partial T}{\partial t} = \frac{k}{\rho_i c}\left(\nabla^2 T + \frac{\partial^2 T}{\partial z^2}\right) - \mathbf{u}\cdot\nabla T + \frac{\Phi}{\rho_i c} - w\frac{\partial T}{\partial z}$$

$$\frac{\partial T}{\partial t'} = \frac{k}{\rho_i c H^2}\frac{\partial^2 T}{\partial \sigma^2} - \mathbf{u}\cdot\hat{\nabla} T + \frac{\sigma g}{c}\frac{\partial \mathbf{u}}{\partial \sigma}\cdot\nabla s + \frac{1}{H}\frac{\partial T}{\partial \sigma}\left(w - w_{\text{grid}}\right)$$

$$w_{\text{grid}}(\sigma) = \frac{\partial s}{\partial t} + \mathbf{u}\cdot\nabla s - \sigma\left(\frac{\partial H}{\partial t} + \mathbf{u}\cdot\nabla H\right)$$

Solving temperature

Vertical terms (solve using Crank-Nicolson)

$$\frac{\partial T}{\partial t'} = \frac{k}{\rho_i c H^2}\frac{\partial^2 T}{\partial \sigma^2} - \mathbf{u}\cdot\hat{\nabla}T + \frac{\sigma g}{c}\frac{\partial \mathbf{u}}{\partial \sigma}\cdot\nabla s + \frac{1}{H}\frac{\partial T}{\partial \sigma}\left(w - w_{\text{grid}}\right)$$

Horizontal term (use explicit advection, then Picard iterations)

Much quicker than solving the full 3D problem!

GLIDE API

- Software Interface (API) is designed to be simple
- Use of derived types in design allows multiple ice sheets to be defined in a single code
- Code for `simple_glide` is a good example of how to use the API
- Most parameters are read from a config file
- Supply mass-balance and surface temp each time-step

Initialising GLIDE

Use statements:

```
use glide
use glimmer_config
```

Relevant declarations:

```
type(glide_global_type)        :: model
type(ConfigSection), pointer :: config
```

Initialisation calls:

```
call ConfigRead(fname,config)
call glide_config(model,config)
call glide_initialise(model)
call glide_nc_fillall(model)
time = model%numerics%tstart
```

GLIDE timestepping

Time loop statements:

```
do while(time.le.model%numerics%tend)
   call glide_set_acab(model,acab)
   call glide_set_artm(model,artm)
   call glide_tstep_p1(model,time)
   call glide_tstep_p2(model)
   call glide_tstep_p3(model)
   time = time + model%numerics%tinc
end do
```

N.B. Units: mass-balance (m of ice)
surface temp (deg C)
time (years)

Finishing up

- Remember to finalise GLIDE!
  - This closes output files, and generally tidies up

```
call glide_finalise(model)
```

Anatomy of a config file

- Configuration files follow a simple syntax:
  - Divided into sections [*section_name*]
  - Sections contain a list of key-value pairs
  - Allowed sections/keys listed in documentation
  - Where appropriate, Glimmer defines sensible defaults for missing parameters
  - Array-value parameters are possible
  - Config files are read into a data structure at the start
  - Utilities exist for manipulating the data structure

Example GLIDE config file

```
[EISMINT-1 fixed margin]

[grid]
# grid sizes
ewn = 31
nsn = 31
upn = 11
dew = 50000
dns = 50000

[options]
temperature = 1
flow_law = 2
marine_margin = 2
evolution = 0
basal_water = 2
vertical_integration = 1
```

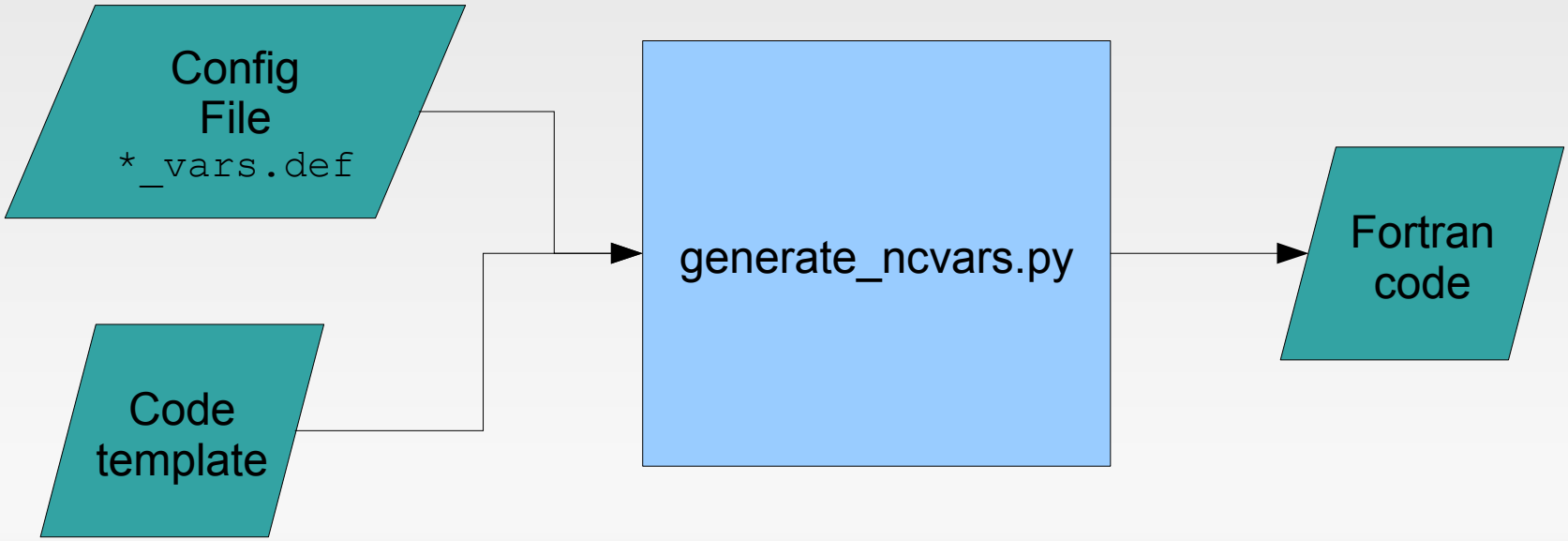
```
[time]
tend = 200000.
dt = 10.
ntem = 1.
nvel = 1.
niso = 1.

[parameters]
flow_factor = 1
geothermal = -42e-3

[CF default]
title: EISMINT-1 fixed margin
comment: forced upper kinematic BC

[CF output]
name: e1-fm.1.nc
frequency: 1000
variables: thk uflx vflx bmlt btemp
temp uvel vvel wvel diffu acab
```

Finding your way around...

- All fortran code is in `src/fortran`
- Use `grep`!
- Most important file prefixes:
  - `glide_*.F90`
  - `glint_*.F90`
  - `glimmer_*.F90`
- Some code is generated automatically...

NetCDF I/O autogeneration

- Writing NetCDF I/O code by hand would be very time-consuming and error-prone
- Use Python to generate I/O code automatically



NetCDF I/O autogeneration

```
[thk]
dimensions:     time, y1, x1
units:          meter
long_name:      ice thickness
data:           data%geometry%thck
factor:         thk0
standard_name:  land_ice_thickness
hot:            1
coordinates:    lon lat
```

Scaling in GLIDE

- In GLIDE *only*, all variables are scaled
- Need to be aware of this when:
  - accessing variables within GLIDE data structures from elsewhere
  - adding/changing code in GLIDE
- Familiarity with existing code is best way to learn
- True value = GLIDE value × factor

Finding out about scaling

- Basic scale factors defined in `glimmer_paramets.F90`
- Scaling of individual variables given in I/O definition files
- You can remind yourself of *how* scaling works by looking at the *end* of auto-generated I/O files (e.g. `glide_io.F90`) – this where get/set code resides

GLIDE Derived Types