Lightweight Verification For Computational Science Models

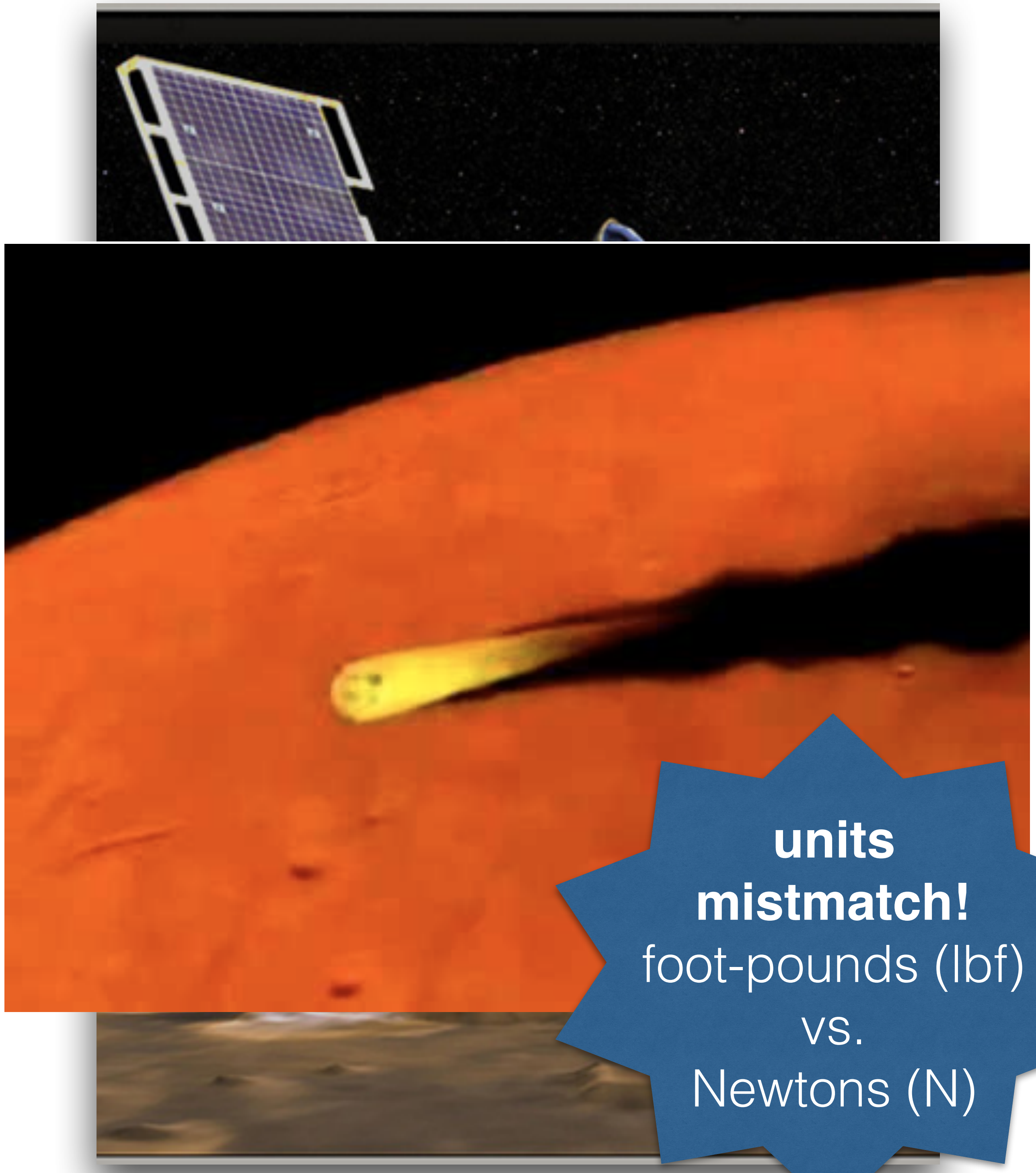
Dominic Orchard

Joint work with colleagues at the University of Cambridge:
Mistral Contrastin, Matthew Danish, Andrew Rice

University of Kent

UNIVERSITY OF CAMBRIDGE

CamFort



units mistmatch!
foot-pounds (lbf)
vs.
Newtons (N)

NASA/JPL/Corby Waste

indexing error!*
minus sign flipped
in array access

average bug-rate in industry software is 15-50 errors per 1000 lines**

---

* Z. Merali, Computational science: Error, why scientific programming does not compute, 2010

** S. McConnell, *Code complete*, O'Reilly Media, Inc., 2004.

How to ensure correctness?

- Testing
  - Unit testing
  - Integration testing
  - Combine with code-coverage checkers
  - Requires significant effort

- Formal verification
  - Bug finding tools (e.g. Clang analyser for C clang-analyzer.llvm.org)
  - Specification-based systems

Specification-based approaches to verification

- User specifies some aspect of the program
- A verification tool checks conformance

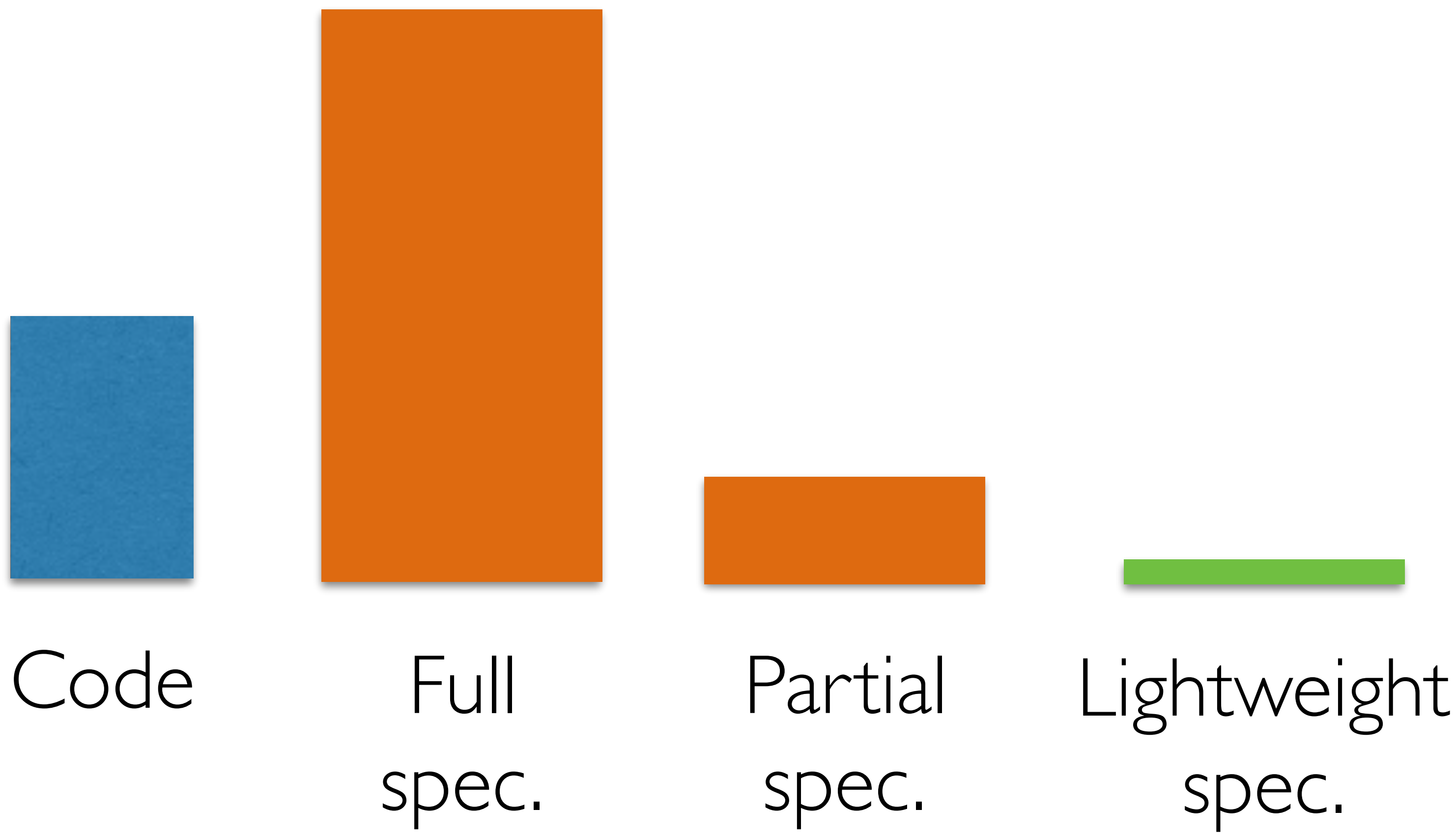
- built into a language e.g. type systems
  - Specify the broad range values that should be input and output

    e.g. `integer :: x;  character :: y;  x = x / y`

- additional specification language
  - e.g. ACSL behaviour specs. for C (https://frama-c.com/acsl.html)

    relationship between input/outputs, ranges of values, and more

Specification-based approaches to verification



Time consuming
Specification completeness?

How to chose
which parts?

Focussed on one
aspect

CamFort

https://github.com/camfort/camfort/wiki

- Lightweight specification / verification of numerical Fortran
  - units-of-measure typing
  - stencil specifications (shape of array access)

- Specifications are comments
- Some specifications can be auto-generated for legacy code

Dimensional analysis

(“Great Principle of Similitude”, Isaac Newton, 1686)

x is a length (dimension)
x is in metres (unit of measure)

unit(x * y) = (unit x) * (unit y)

unit(x / y) = (unit x) / (unit y)

unit(x + y) = unit x = unit y

unit(x - y) = unit x = unit y

unit(x^R) = unit(x)^R

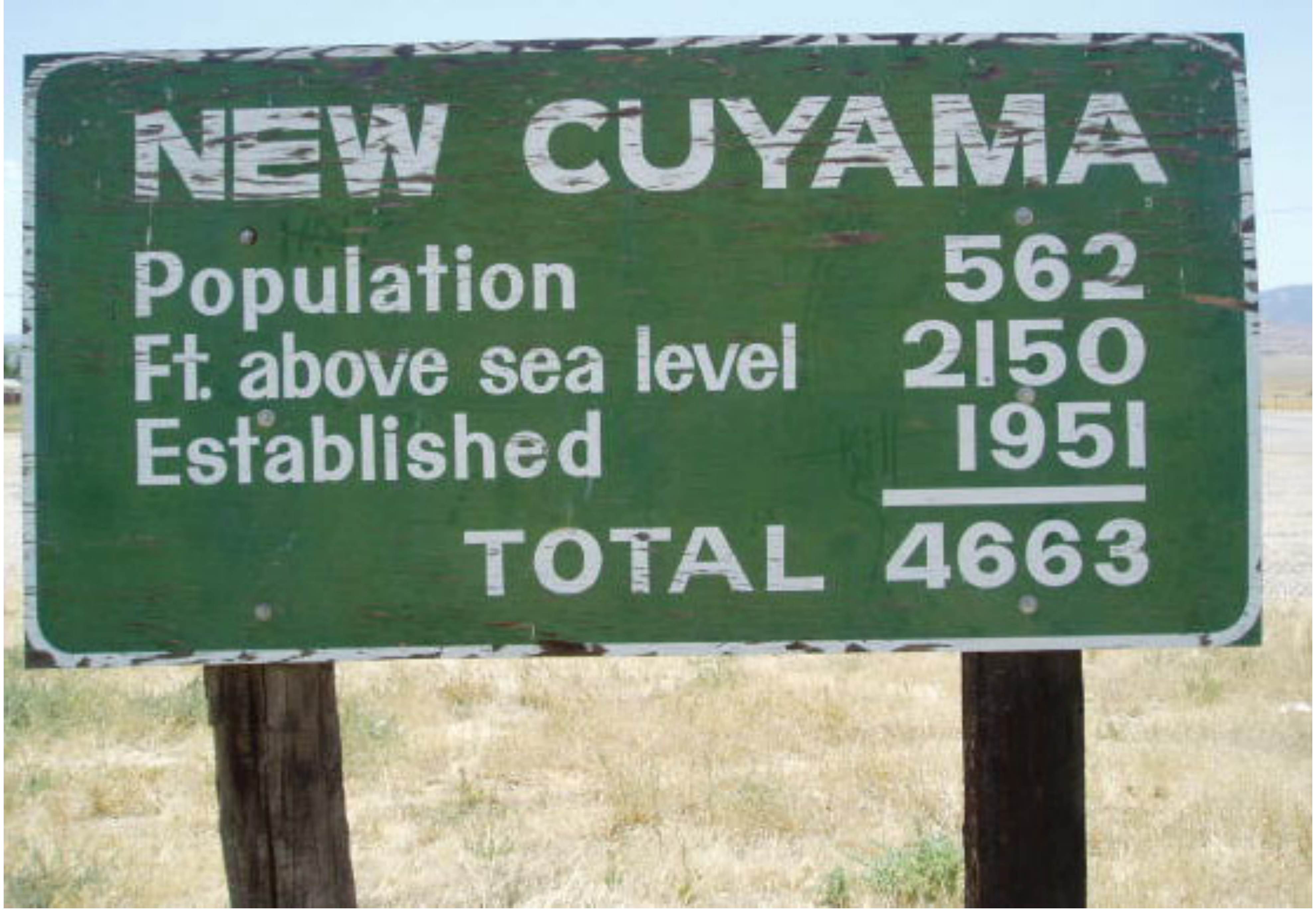


photo from Andrew Kennedy's website
http://research.microsoft.com/en-us/um/people/akenn/units/

Example: units-of-measure specifications

```
program energy
  real :: mass = 3.00, gravity = 9.91, height = 4.20
  real :: potential_energy

  potential_energy = mass * gravity * height
end program energy
```

Suggest

```
$ camfort units-suggest energy1.f90
energy1.f90:
 (2:22)  mass
 (2:51)  height
 (3:11)  potential_energy
```

Example: units-of-measure specifications

```fortran
program energy
  != unit kg :: mass
  != unit m  :: height
  real :: mass = 3.00, gravity = 9.91, height = 4.20
  != unit kg m**2/s**2 :: potential_energy
  real :: potential_energy

  potential_energy = mass * gravity * height
end program energy
```

Check

```
$ camfort units-check energy1.f90

energy1.f90: Consistent. 4 variables checked.
```

Example: units-of-measure specifications

```
program energy
  != unit kg :: mass
  != unit m  :: height
  real :: mass = 3.00, gravity = 9.91, height = 4.20
  != unit kg m**2/s**2 :: potential_energy
  real :: potential_energy

  potential_energy = mass * gravity * height
end program energy
```

Synthesise

```
$ camfort units-synth energy1.f90 energy1.f90

Synthesising units for energy1.f90
```

Example: units-of-measure specifications

```
program energy
  != unit kg :: mass
  != unit m  :: height
  != unit m/s**2  :: gravity
  real :: mass = 3.00, gravity = 9.91, height = 4.20
  != unit kg m**2/s**2 :: potential_energy
  real :: potential_energy

  potential_energy = mass * gravity * height
end program energy
```

Synthesise

```
$ camfort units-synth energy1.f90 energy1.f90

Synthesising units for energy1.f90
```

```
$ camfort units-check energy2.f90

energy2.f90 : Inconsistent:
 - at 17:38 'kinetic_energy' should be '(kg m**2.0) / s**2.0'
   instead 'kinetic_energy' is '1 kg (m / s)'
```

```
    != unit m/s**2  :: gravity
    real :: mass = 3.00, gravity = 9.91, height = 4.20
    != unit kg m**2/s**2 :: potential_energy
    real :: potential_energy
    real :: kinetic_energy, total_energy

    != unit 1 :: half
    != unit m/s :: velocity
    real :: half = 0.5, velocity = 4.00

    potential_energy = mass * gravity * height
    kinetic_energy = half * mass * velocity

    total_energy = potential_energy + kinetic_energy
  end program energy
```

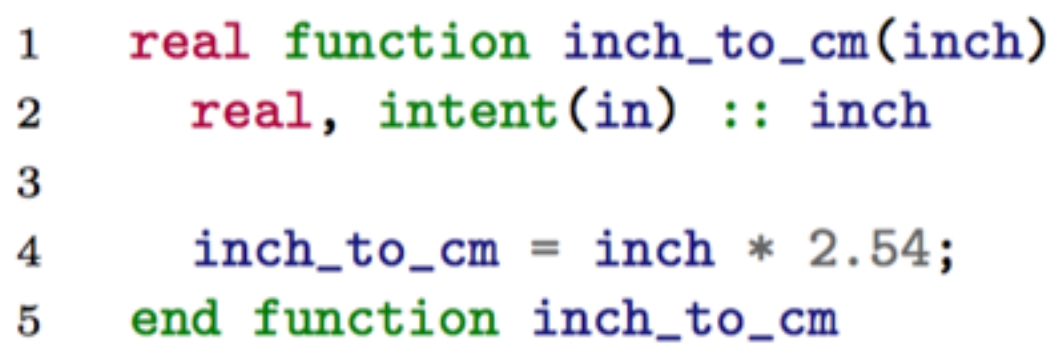
“Unitless” coefficients

BUG! should be velocity**2

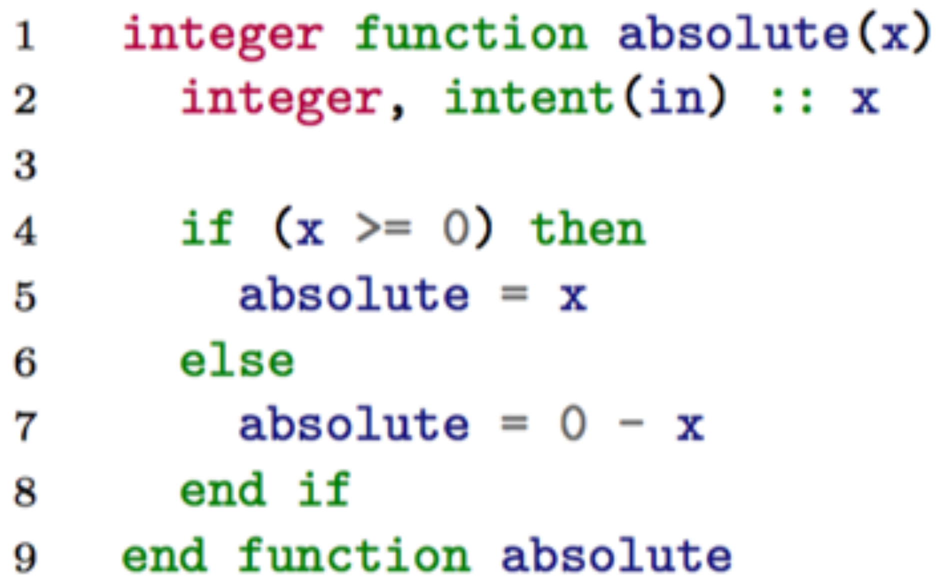
Unit aliases

```
!= unit :: joule = kg m**2 / s**2
!= unit joule :: potential_energy
real :: potential_energy
```

Polymorphism



```
real function inch_to_cm(inch)
  real, intent(in) :: inch

  inch_to_cm = inch * 2.54;
end function inch_to_cm
```



```
integer function absolute(x)
  integer, intent(in) :: x

  if (x >= 0) then
    absolute = x
  else
    absolute = 0 - x
  end if
end function absolute
```

Monomorphic

```
!= unit in :: inch
!= unit cm :: inch_to_cm
```

```
inch_to_cm : in → cm
```

Polymorphic

```
!= unit ‘u :: x
!= unit ‘u :: absolute
```

```
absolute : ∀u . u → u
```

Check

Does it do what I think it does?

Infer

What does it do?

Synthesise

Capture what it does for documentation & future-proofing

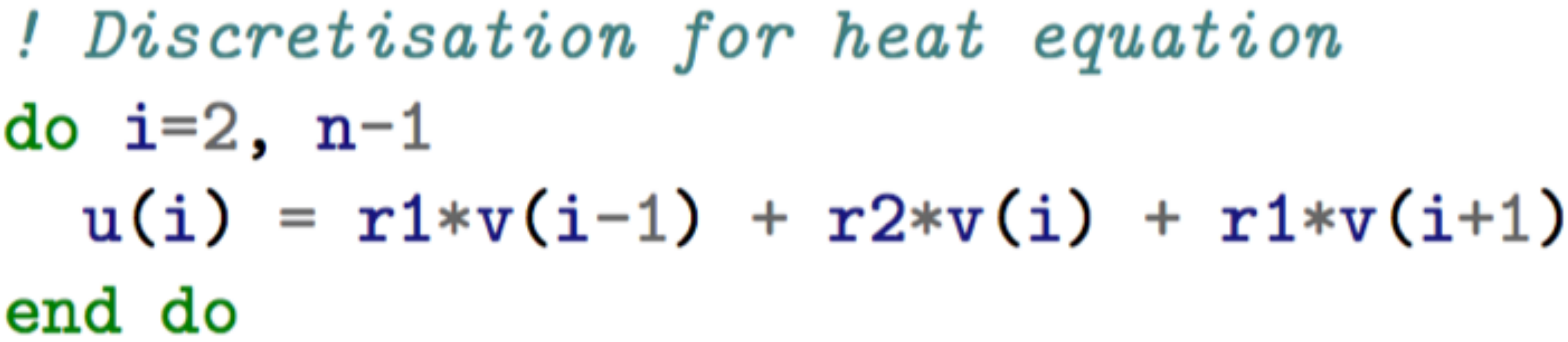
Suggest

Where should I add a specification to get the most information?

Units-of-measure in other languages

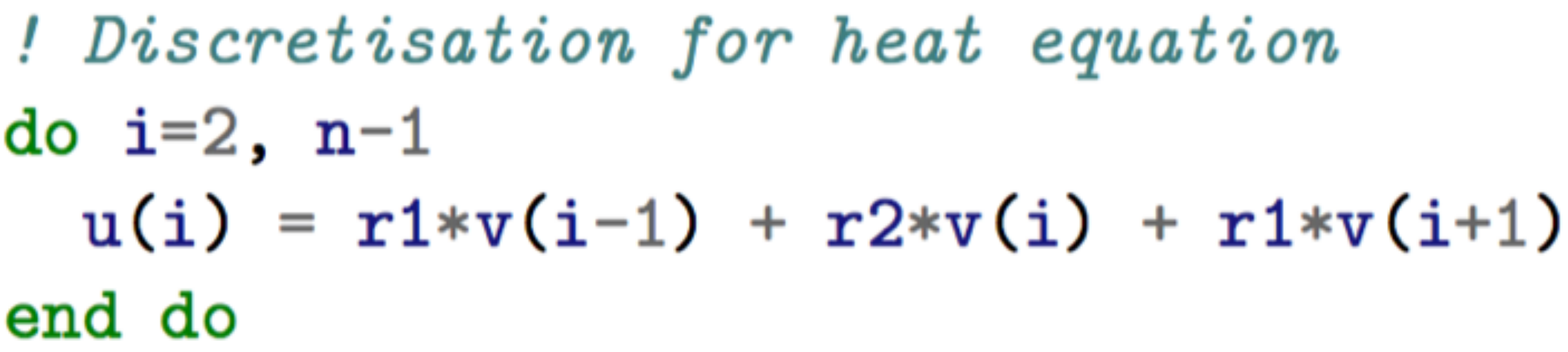
- **F#** - built-in
- **Python** - Pint http://pint.readthedocs.io
- **C** - Osprey (not sure if available yet)

Example: stencil specifications



```
! Discretisation for heat equation
do i=2, n-1
  u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
end do
```

Example: stencil specifications



```
! Discretisation for heat equation
do i=2, n-1
  u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
end do
```

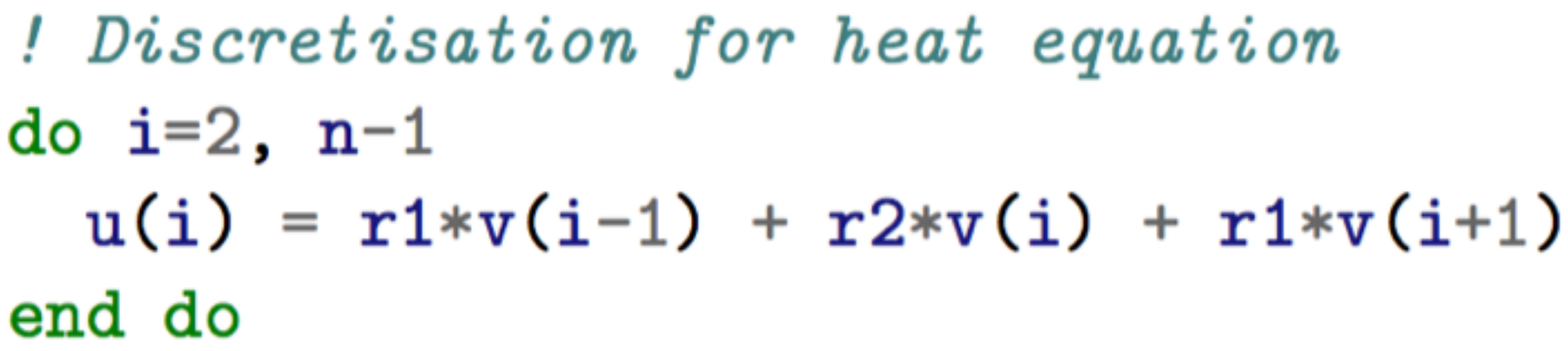
Example: stencil specifications

```
! Discretisation for heat equation
do i=2, n-1
  u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
end do
```

Example: stencil specifications

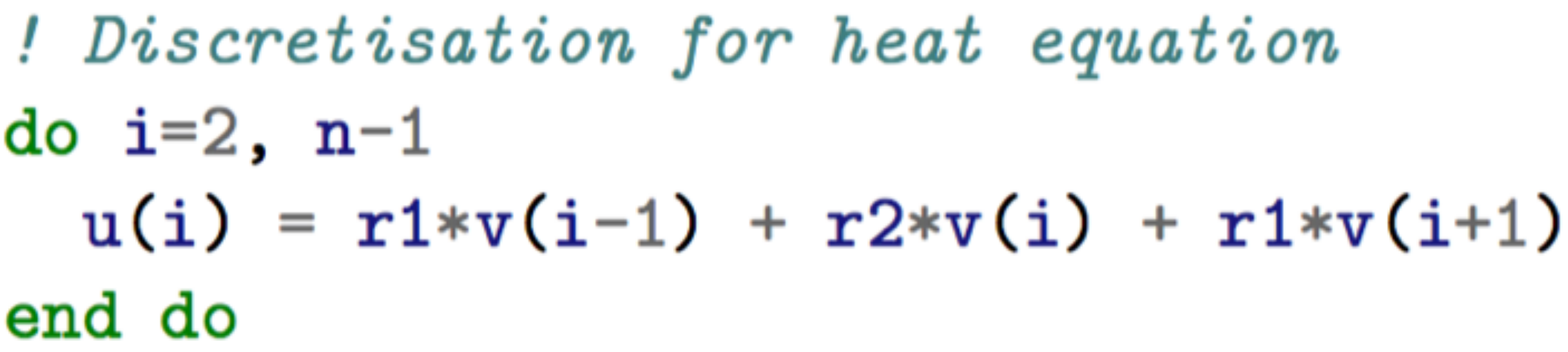
```
! Discretisation for heat equation
do i=2, n-1
  u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
end do
```

Example: stencil specifications



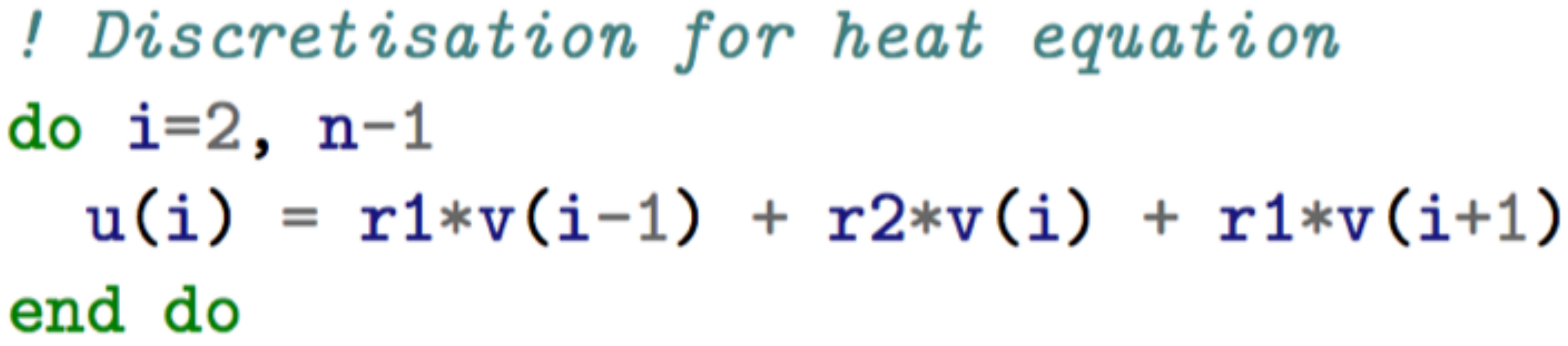
```
! Discretisation for heat equation
do i=2, n-1
  u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
end do
```

Example: stencil specifications



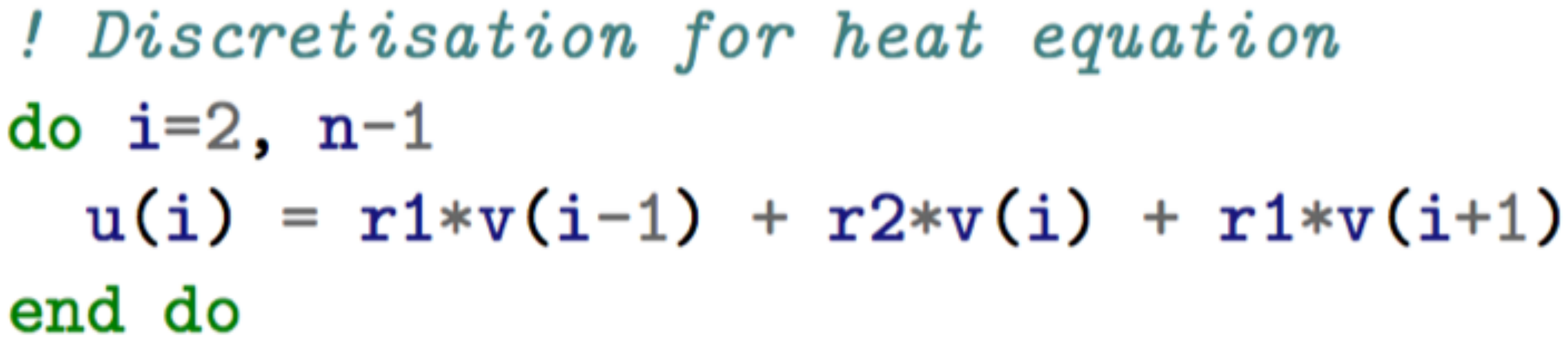
```
! Discretisation for heat equation
do i=2, n-1
  u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
end do
```

Example: stencil specifications



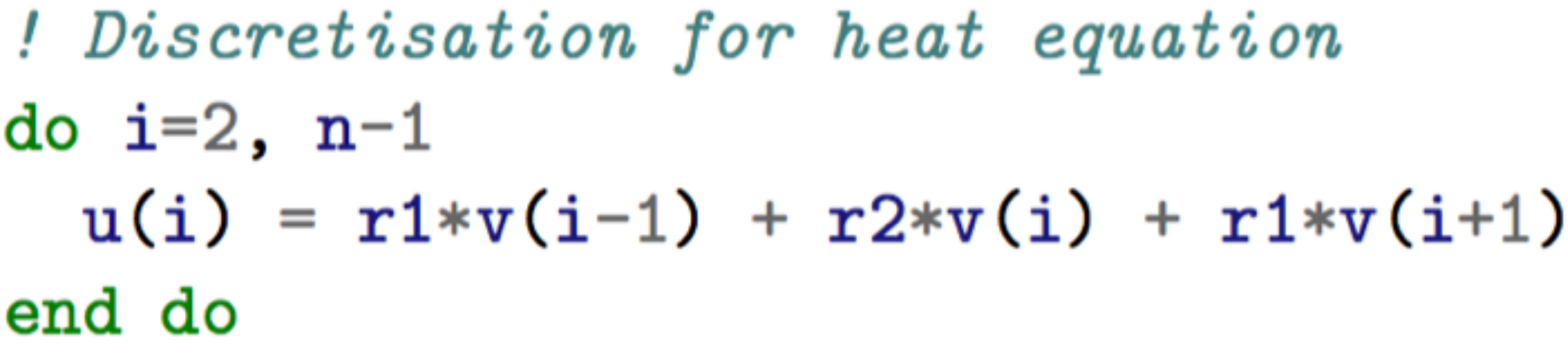
```
! Discretisation for heat equation
do i=2, n-1
  u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
end do
```

Example: stencil specifications



```
! Discretisation for heat equation
do i=2, n-1
  u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
end do
```

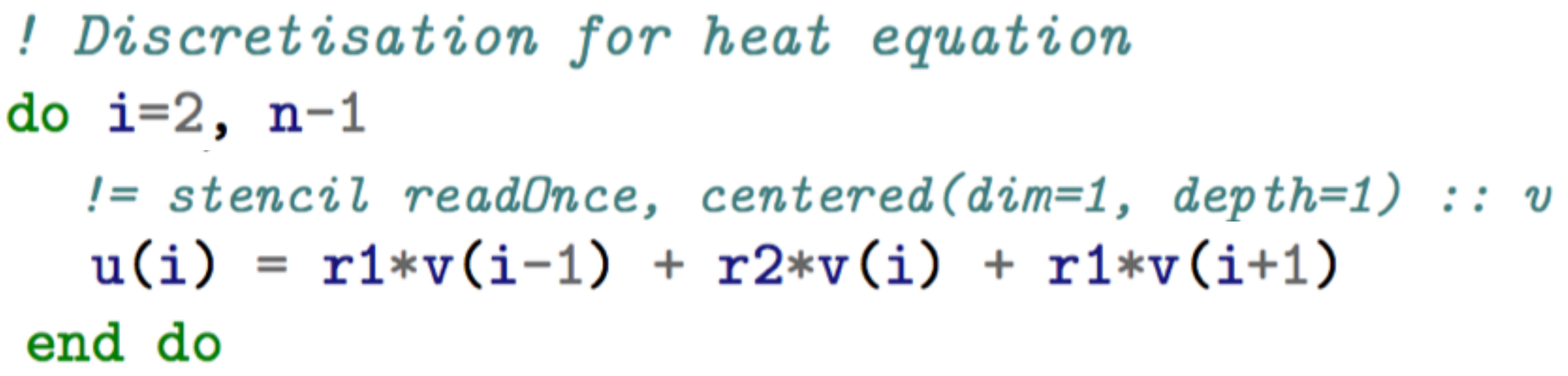
Example: stencil specifications



```
! Discretisation for heat equation
do i=2, n-1
  u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
end do
```

```
$ camfort stencils-infer heat.f90
Inferring stencil specifications for heat.f90

heat.f90
(9:6)-(9:43)  stencil readOnce, (centered(depth=1, dim=1)) :: v
```

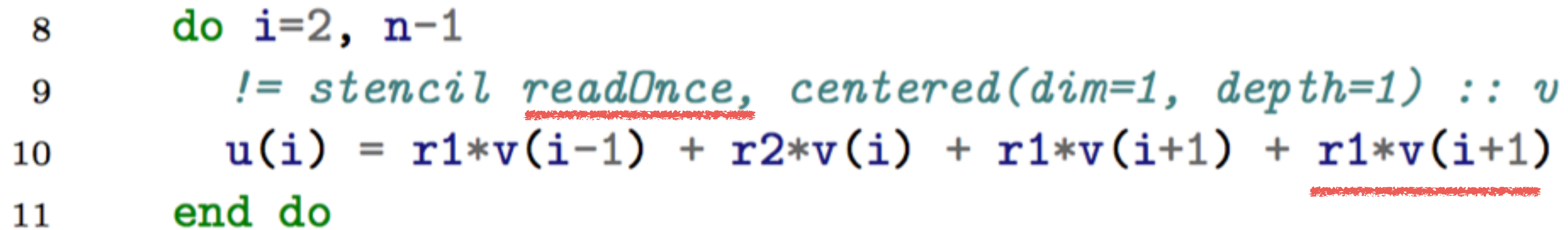
Example: stencil specifications



```
! Discretisation for heat equation
do i=2, n-1
  != stencil readOnce, centered(dim=1, depth=1) :: v
  u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
end do
```

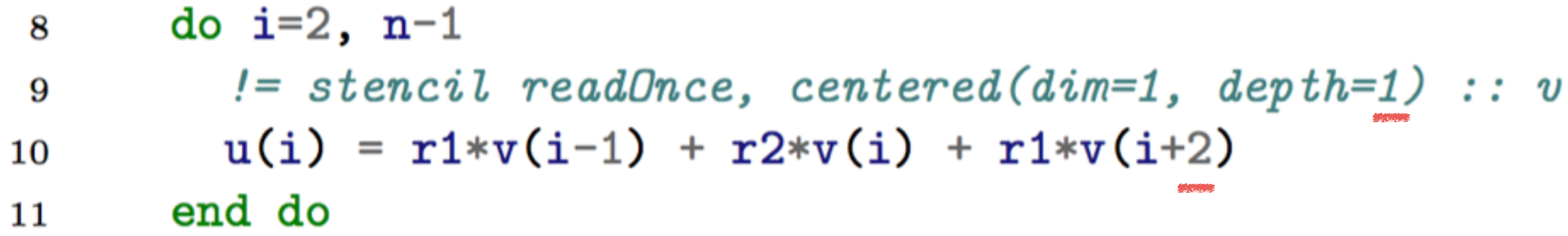
```
$ camfort stencils-synth heat.f90
Synthesising stencil specifications for heat.f90

heat.f90
(9:6)-(9:43)  stencil readOnce, (centered(depth=1, dim=1)) :: v
```

Two potential mistakes



```
do i=2, n-1
  != stencil readOnce, centered(dim=1, depth=1) :: v
  u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1) + r1*v(i+1)
end do
```

Illegal repetition of access pattern



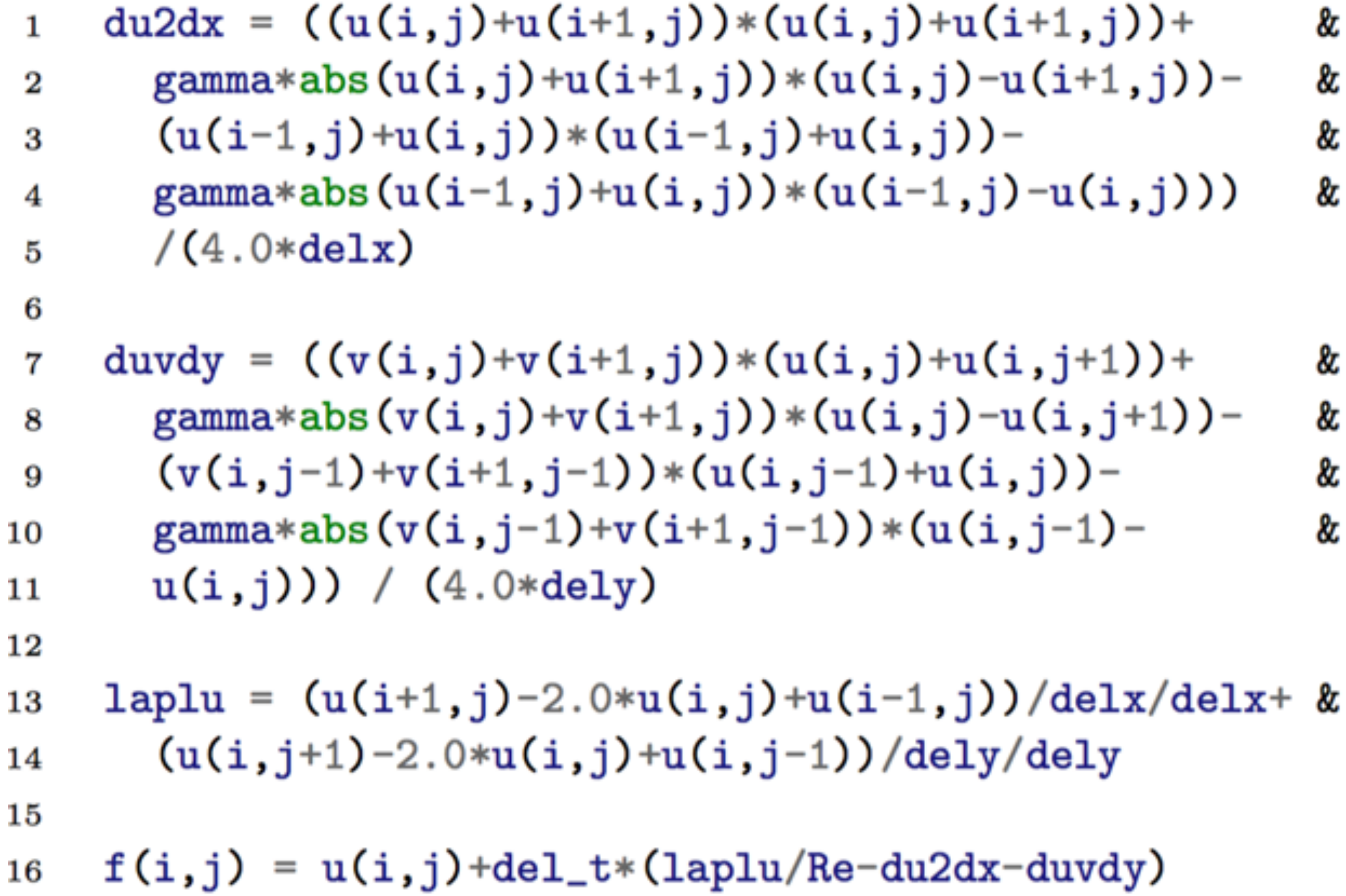
```
do i=2, n-1
  != stencil readOnce, centered(dim=1, depth=1) :: v
  u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+2)
end do
```

Out of bounds stencil access/
Does not conform with the shape.

More advanced specifications

- There are other primitive regions: `pointwise`, `forward`, and `backward`
- Two operators for composition: +, *
- Specifications acting on multiple dimensions

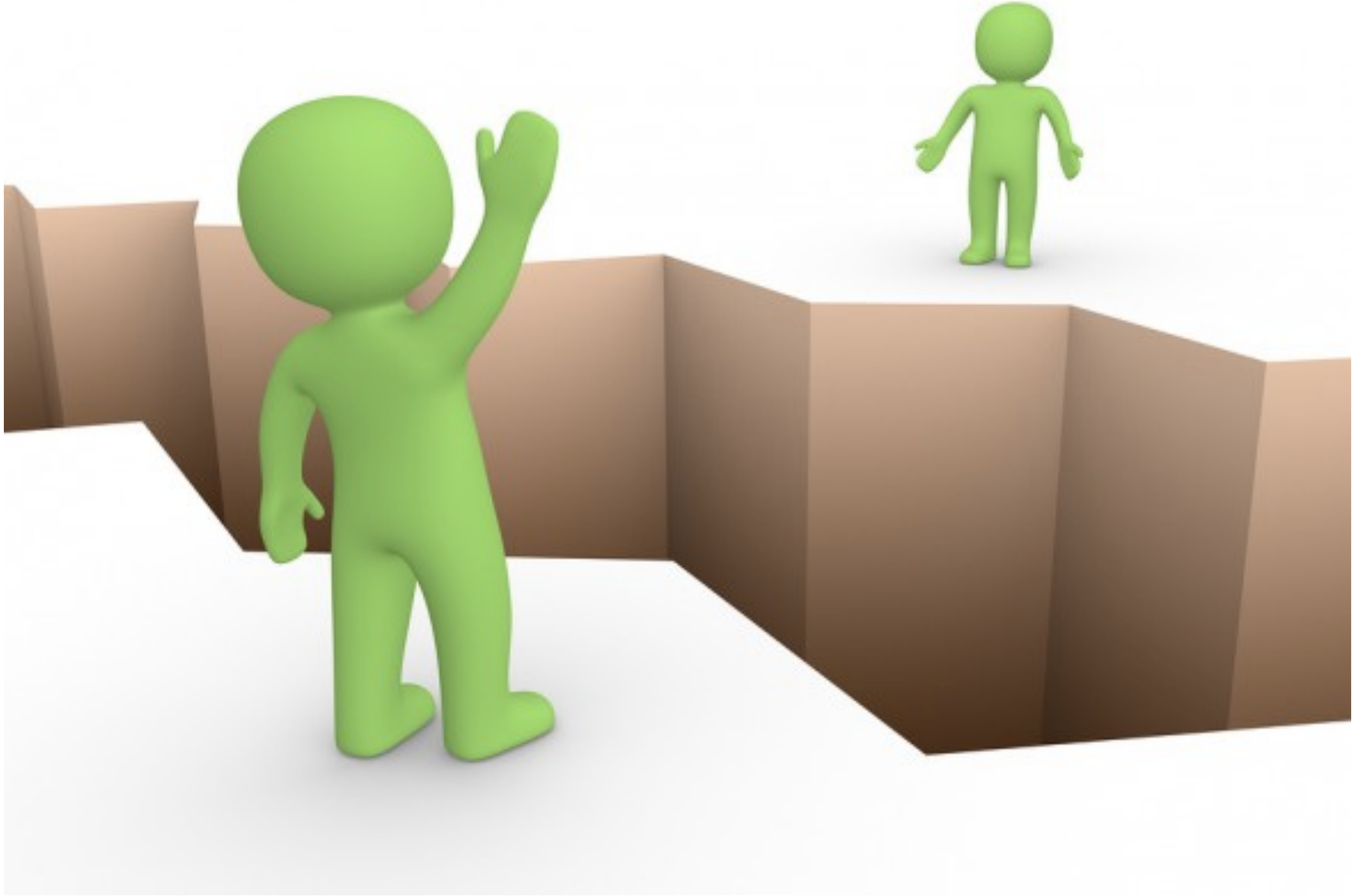
From a Navier-Stokes fluid simulation



```
du2dx = ((u(i,j)+u(i+1,j))*(u(i,j)+u(i+1,j))+      &
  gamma*abs(u(i,j)+u(i+1,j))*(u(i,j)-u(i+1,j))-    &
  (u(i-1,j)+u(i,j))*(u(i-1,j)+u(i,j))-             &
  gamma*abs(u(i-1,j)+u(i,j))*(u(i-1,j)-u(i,j)))    &
  /(4.0*delx)

duvdy = ((v(i,j)+v(i+1,j))*(u(i,j)+u(i,j+1))+      &
  gamma*abs(v(i,j)+v(i+1,j))*(u(i,j)-u(i,j+1))-    &
  (v(i,j-1)+v(i+1,j-1))*(u(i,j-1)+u(i,j))-         &
  gamma*abs(v(i,j-1)+v(i+1,j-1))*(u(i,j-1)-        &
  u(i,j))) / (4.0*dely)

laplu = (u(i+1,j)-2.0*u(i,j)+u(i-1,j))/delx/delx+ &
  (u(i,j+1)-2.0*u(i,j)+u(i,j-1))/dely/dely

f(i,j) = u(i,j)+del_t*(laplu/Re-du2dx-duvdy)
```

```
!= stencil :: (centered(depth=1, dim=1) * pointwise(dim=2))
            + (centered(depth=1, dim=2) * pointwise(dim=1)) :: u

!= stencil :: forward(depth=1, dim=1) * backward(depth=1,dim=2) :: v
```

natural & physical sciences



computer science

Let’s bridge the chasm!

Future plans

- Test generation from properties
- Dependency specifications

Conclusions

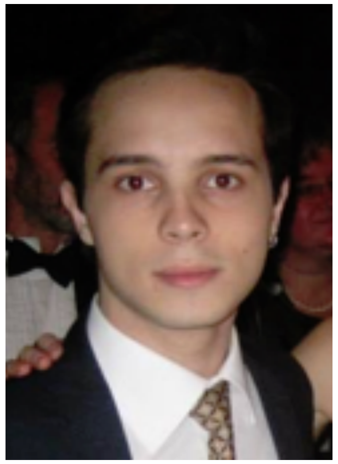
- Correctness is <u>very</u> important
- Testing is good; automated verification better (reduce effort)
- Various tools for different languages
- More interaction needed between CS and sciences to build more effective tools e.g. CamFort

Follow CamFort updates

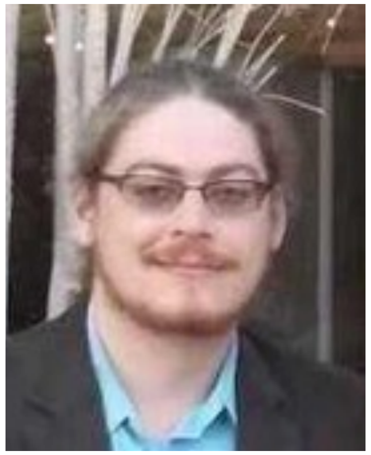
http://github.com/camfort/camfort

@camfort_tool

Thank you!



Mistral Contrastin



Matthew Danish

Dominic Orchard

Andrew Rice

BETTER
SOFTWARE
BETTER
RESEARCH

Software
Sustainability
Institute

https://www.software.ac.uk/