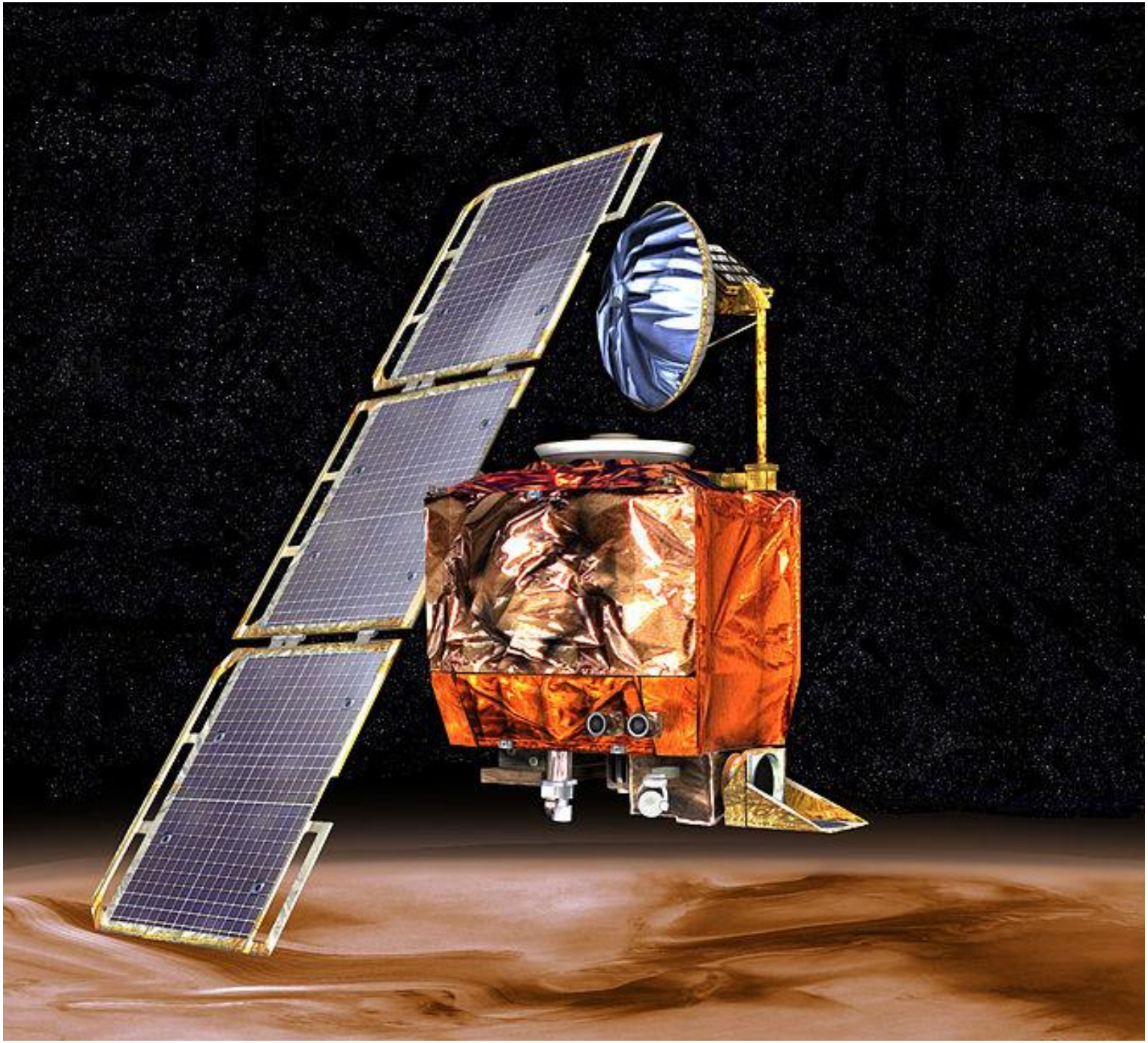
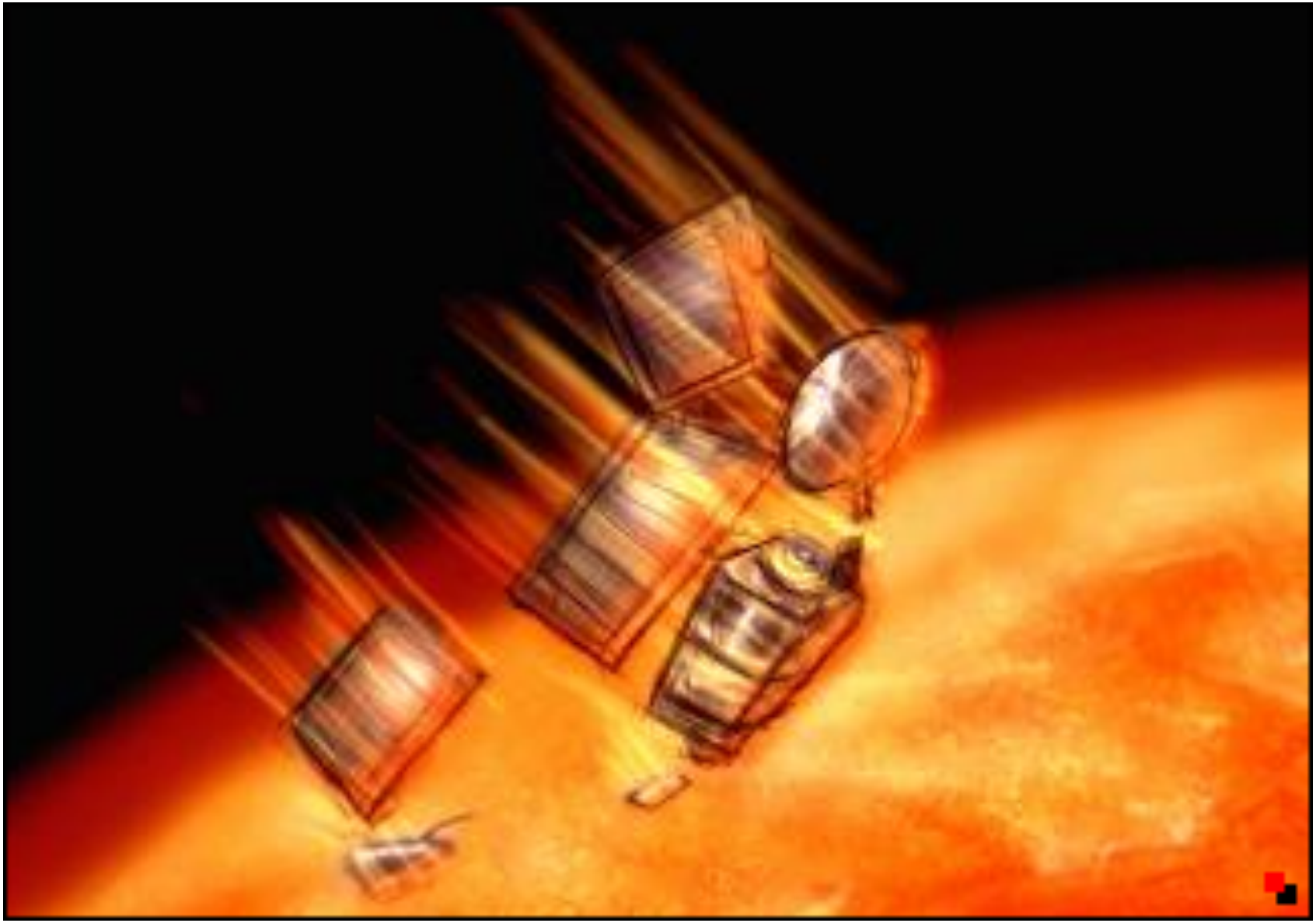
Evolving Fortran types with inferred units-of-measure

Dominic Orchard, Andrew Rice,
Oleg Oshmyan, Alan Mycroft

UNIVERSITY OF CAMBRIDGE

PLE'14, Uppsala, Sweden - Monday 28th July





What what went wrong?

A unit mismatch!

pounds (*lbf*) instead of **Newtons (*N*)**

cost: $327.6 million

Dimensional analysis

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

x is a length (*dimension*)

x is in metres (*unit of measure*)

dim(x * y) = (dim x) * (dim y)

dim(x / y) = (dim x) / (dim y)

dim(x + y) = dim x = dim y

dim(x - y) = dim x = dim y

dim(x^R) = dim(x)^R

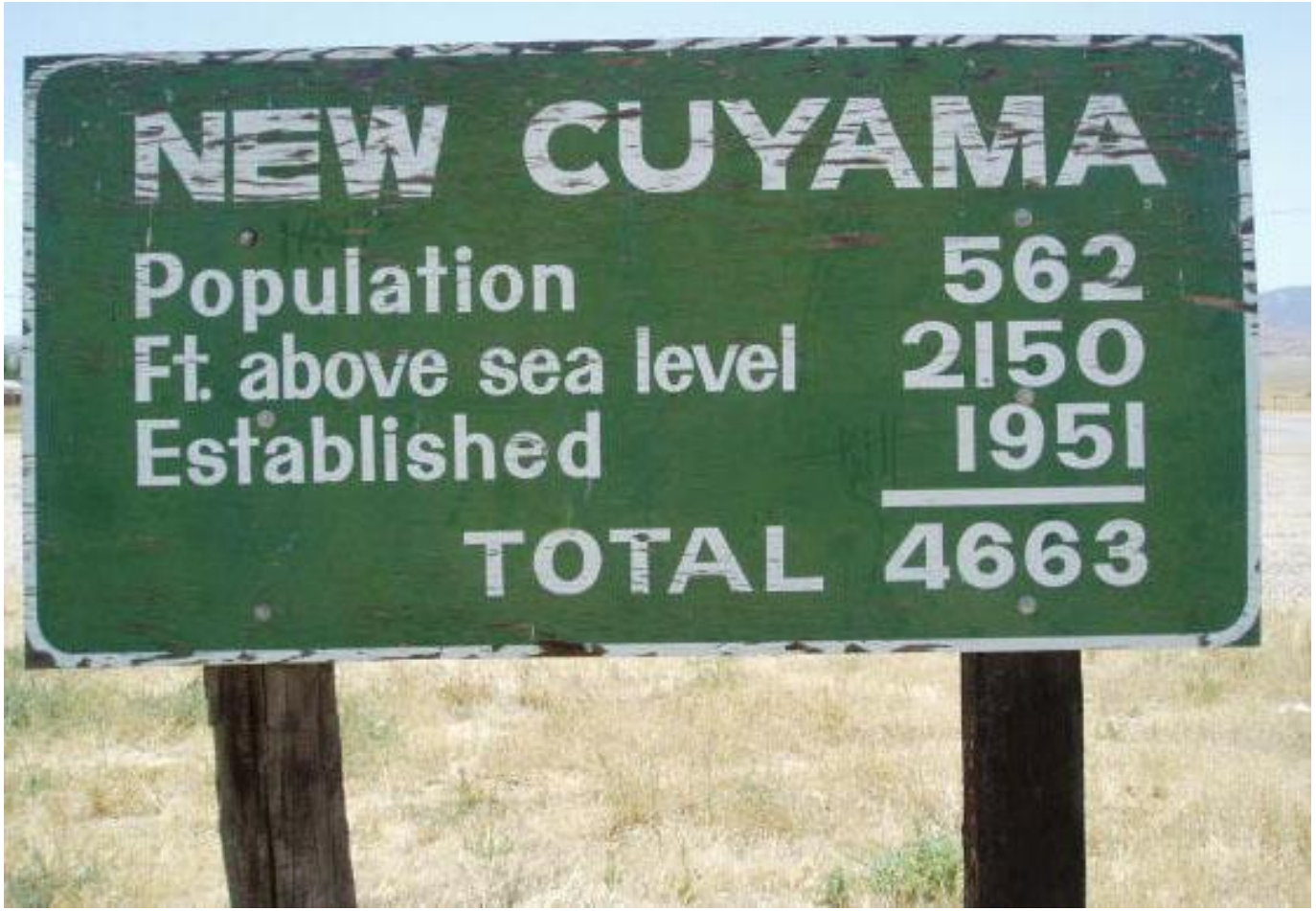


photo from Andrew Kennedy's website http://research.microsoft.com/en-us/um/people/akenn/units/

Dimensional analysis = a type sytem

- House 1983
  - “A proposal for an extended form of type checking of expressions”
- Kennedy 1994 (has more of the history)
  - “Dimension Types”
- How many (popular) languages have this today?
  - **1!** F#

http://research.microsoft.com/en-us/um/people/akenn/units/

Fortran: an important target

- Fortran very popular in science
- Evolved considerably over 60 years
- Lots of long-running projects
- Many numerical programs
- A serious need for more verification
  - automatic tools can help!

Recent ISO proposal for Fortran units

```
unit :: m, s
unit :: mps = m / s
real, unit(m) :: x
real, unit(s) :: t
real, unit(mps):: v
real, unit(mps) :: s
...
v = x / t
s = abs(v)
```

ftp://ftp.nag.co.uk/sc22wg5/N1951-N2000/N1969.pdf

Recent ISO proposal for Fortran units

- All units must be declared
- All variables must have a unit
- All derived units must have a unique name

Follows Fortran type system traditions

ftp://ftp.nag.co.uk/sc22wg5/N1951-N2000/N1969.pdf

‘Explicitness’ tradition hinders evolution

Two long-running climate modelling projects at our University:

- (Hybrid 8) 10kloc - 1k variable declarations
- (Hybrid 4) 8.5kloc - 1.2k variable declarations

Lightweight approach

- Type inference
- Implicitly-introduced unit names
- Polymorphism

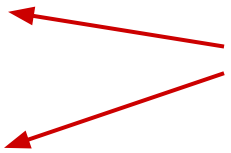
```
abs :: ∀d. real, unit(d) → real, unit(d)
```

About CamFort

- Cambridge Fortran research infrastructure
- Program analyses and transformations
- Refactorings
- Type-system extensions
  - Units-of-measure

Live demo

```
real, unit(m) :: x
real, unit(s) :: t
real          :: v
real          :: s
...
v = x / t
s = abs(v)
```



inferred as
unit(m/s)

Initial results on adoption barrier

- Around 10% of variables need explicit annotations

Program	**Variables**	**Explicit units required**	**%**
Program 1	8	0	0%
Program 2	20	2	10%
Program 3	24	2	11%
Program 4	11	2	5.5%
Program 5	16	0	0%
Program 6	10	0	0%
Program 7	23	3	7.7%

Lessons learned…

- Automatic verification tools are good
- Inference eases evolution, reduces effort
- Breaking traditions can be good (when they hinder upgrading a code base)

Download: https://github.com/dorchard/camfort

See more: http://www.cl.cam.ac.uk/research/dtg/naps/

Thanks!