Functional Programming and Dependent Types for Metrology

Keith Lines, André Videla,

Data Science, NPL / Mathematically Structured Programming Group,

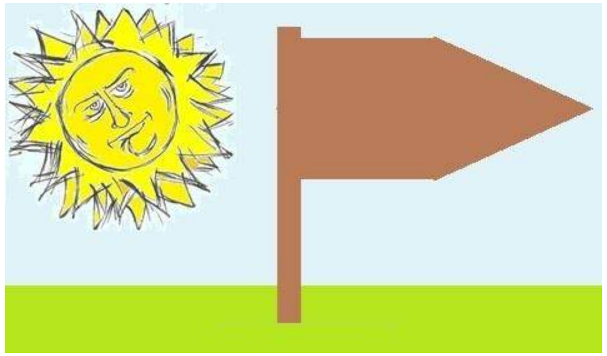
University of Strathclyde

v1.0.0, presentation to BCS 11/02/2025

Contents

Aims

- Present overview of Strathclyde Joint Appointment / PhD supervision work
- Present case for the value of **theoretical computer science** and **functional programming** for NPL / Data Science Dept.:
  - Or for **anyone** who codes…
  - ...just a little **appreciation** helps
- Show how case is pitched



And get some feedback!

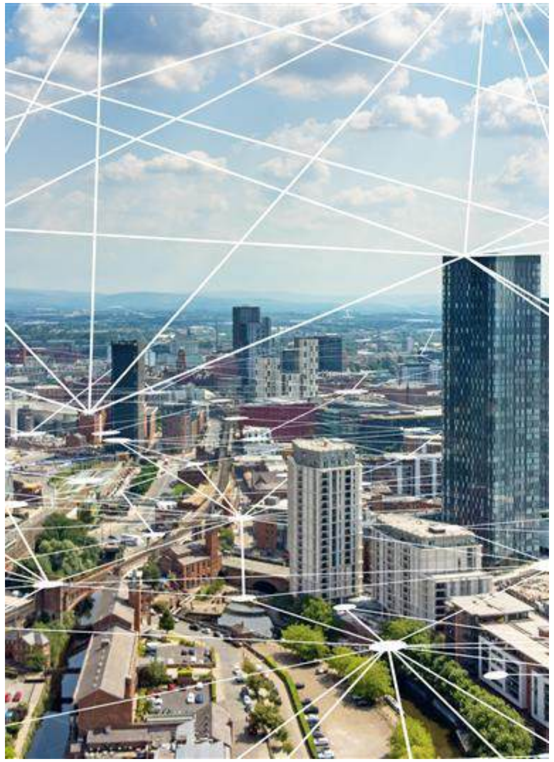
Contents

Introduction

About NPL

- UK's National Metrology Institute founded in 1900
- https://www.npl.co.uk/125
- A public corporation owned by the Department for Science, Innovation and Technology (DSIT)
- Based in Teddington (London) with locations in Strathclyde, Surrey, Cambridge, Huddersfield and Solihull
- Strategic partners DSIT, the University of Surrey and The University of Strathclyde
- 800 scientists with a breadth and depth of metrology expertise.

Introduction



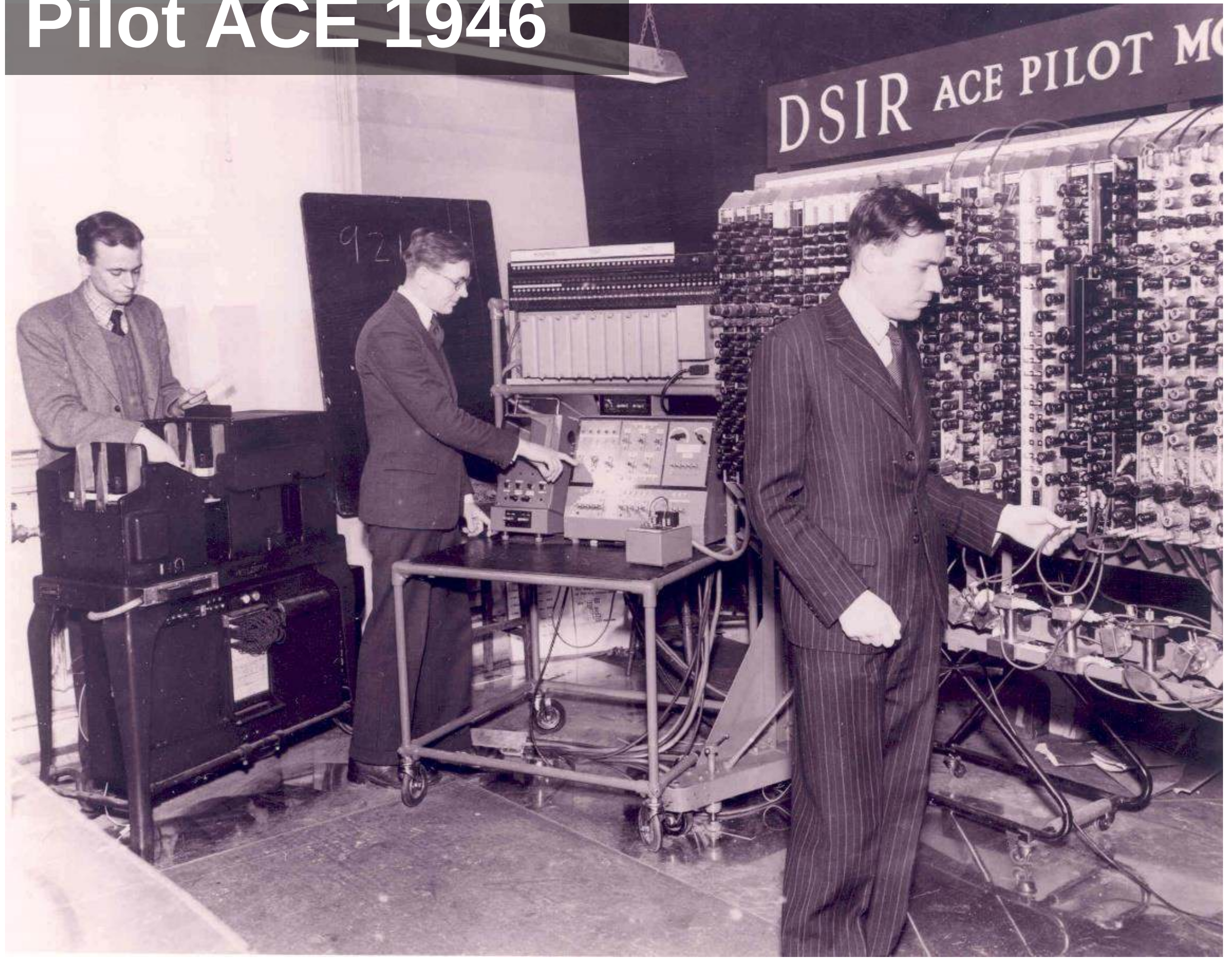
National Challenges

- Prosperity
- Security and resilience
- Environment
- Health

Metrology improves the effectiveness and efficiency of science and trust in its outcomes, which in turn unlocks the potential of innovation, allowing faster routes to market. Evidence-based policy, regulation and decision making are heavily reliant on measurements and data, and NPL is key in providing and digitising that measurement infrastructure.

Introduction

Pilot ACE 1946



Packet-switching developed at NPL 1966

No Introduction Required

Introduction: NPL Data Science Dept.

Aim: Confidence in the intelligent & effective use of data

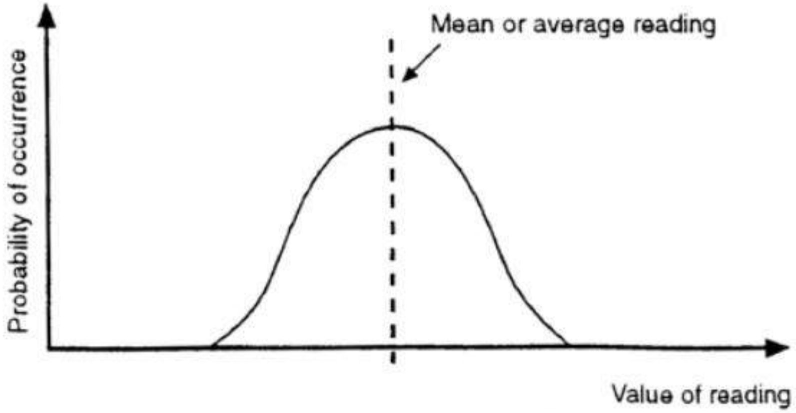
- Mix of mathematicians, data scientists, AI / machine learning experts, statisticians and physicists and software / data quality experts.
- ~50 staff across three sites. Including 11 graduate scientists and joint appointments with Surrey and Edinburgh Universities
- ~25 students (PhDs, sandwich courses)
- **Extensive collaboration:** Can’t do data science without data
  - **Internal:** work with most other departments at NPL
  - **Fellow NMIs** worldwide
  - **External companies:** collaborations and consultancy
  - **Academia:** CDT engagement, grant-funded projects
  - **Other** establishments & industry bodies: UK & worldwide

AfDSP
Alliance for Data Science Professionals

https://www.npl.co.uk/data-science

Introduction

- If there's **<u>one</u>** part of NPL where metrologists and computer scientists should be able to communicate fruitfully it should be **Data Science**



Definition 4.1. A **monoid** $M = (A, \_\cdot\_, 1)$ consists of

$$A \in \mathsf{Set} \quad \text{with} \quad \_\cdot\_ \in A \times A \to A \quad \text{and} \quad 1 \in A.$$

Or fruitful communication should be facilitated

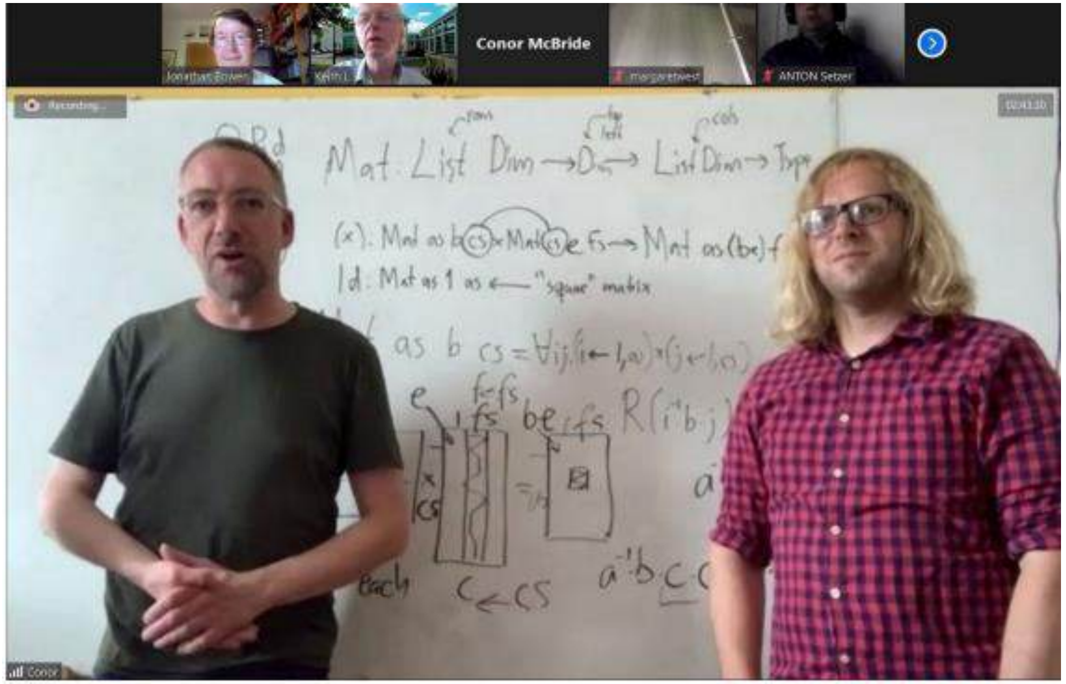
And it's nothing new [1]…

Introduction

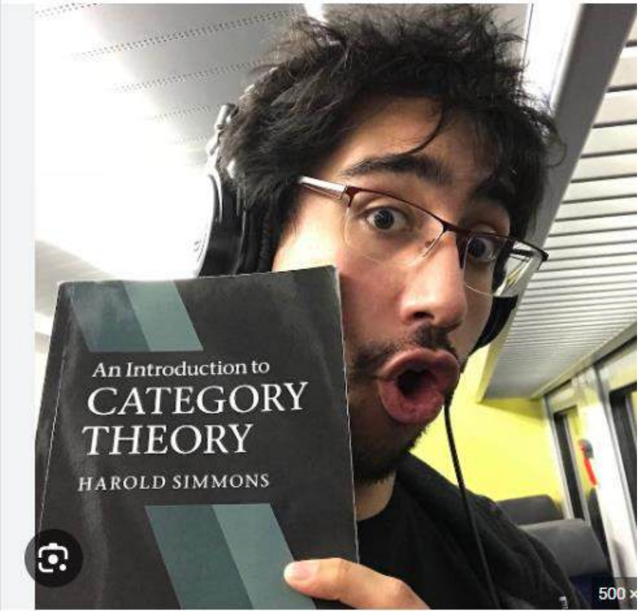
- **Project Trust**[4], “Trust to the fore”
- Engagement between:
  - Strathclyde, **Mathematically Structured Programming Group [2]**
    - Joint appointments: Connor McBride, Fredrik Nordvall Forsberg
    - Head of department: Neil Ghani
    - PhD students: André Videla, Eigil Rischel
  - NPL, Data Science
    - Alistair Forbes, Keith Lines, Ian Smith

Introduction

Some computer scientists:



Fredrik and Conor at the whiteboard.



Dimensionally correct by construction: Type systems for programs [3]

Date presented: June 24, 2021

Dependent types for practical use [4]

Date presented: March 29, 2022

Introduction

Amongst many other things

They publish papers, full of things that look like this example [5]:

T-LET

$$\frac{P = \forall \overline{Z}.A \qquad \Phi, \overline{Z}; \Gamma\,[\emptyset] \vdash n : A \qquad \Phi; \Gamma, f : P\,[\Sigma] \vdash n' : B}{\Phi; \Gamma\,[\Sigma] \vdash \mathbf{let}\ f : P = n\ \mathbf{in}\ n' : B}$$

T-LETREC

$$\frac{(P_i = \forall \overline{Z}_i.\{C_i\})_i \qquad (\Phi, \overline{Z}_i; \Gamma, \overline{f : P} \vdash e_i : C)_i \qquad \Phi; \Gamma, \overline{f : P}\,[\Sigma] \vdash n : B}{\Phi; \Gamma\,[\Sigma] \vdash \mathbf{letrec}\ \overline{f : P = e}\ \mathbf{in}\ n : B}$$

- What do such examples mean?
- Is it worthwhile, for NPL, to try and find out?
- Is any of this work directly applicable to developing trustworthy software tools for metrology?

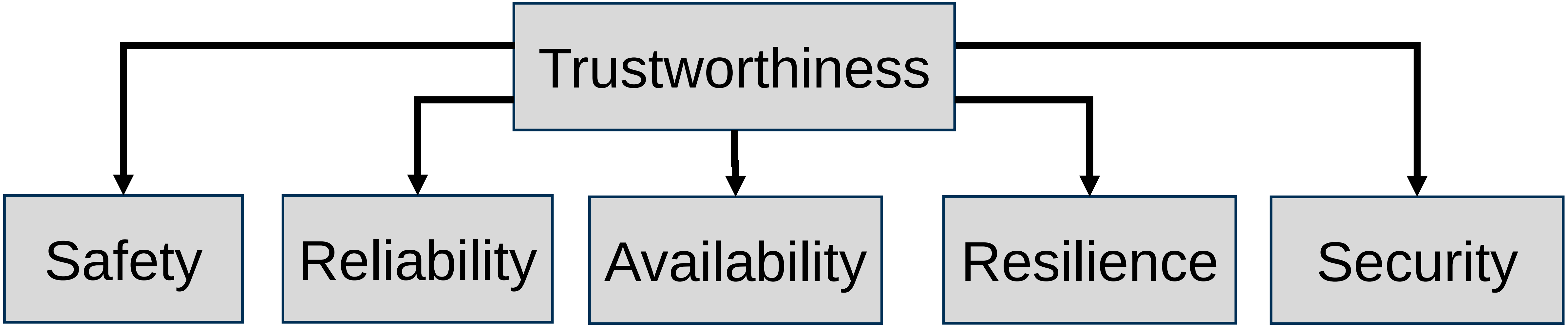
Introduction

- **BS 10754-1:2018 [6]** defines **trustworthy** as:

  Appropriately addresses safety, reliability, availability, resilience and security issues

But that's a discussion for another presentation…

- Five **facets** of trustworthiness:



Contents

Background: Theoretical computer science (TCS)

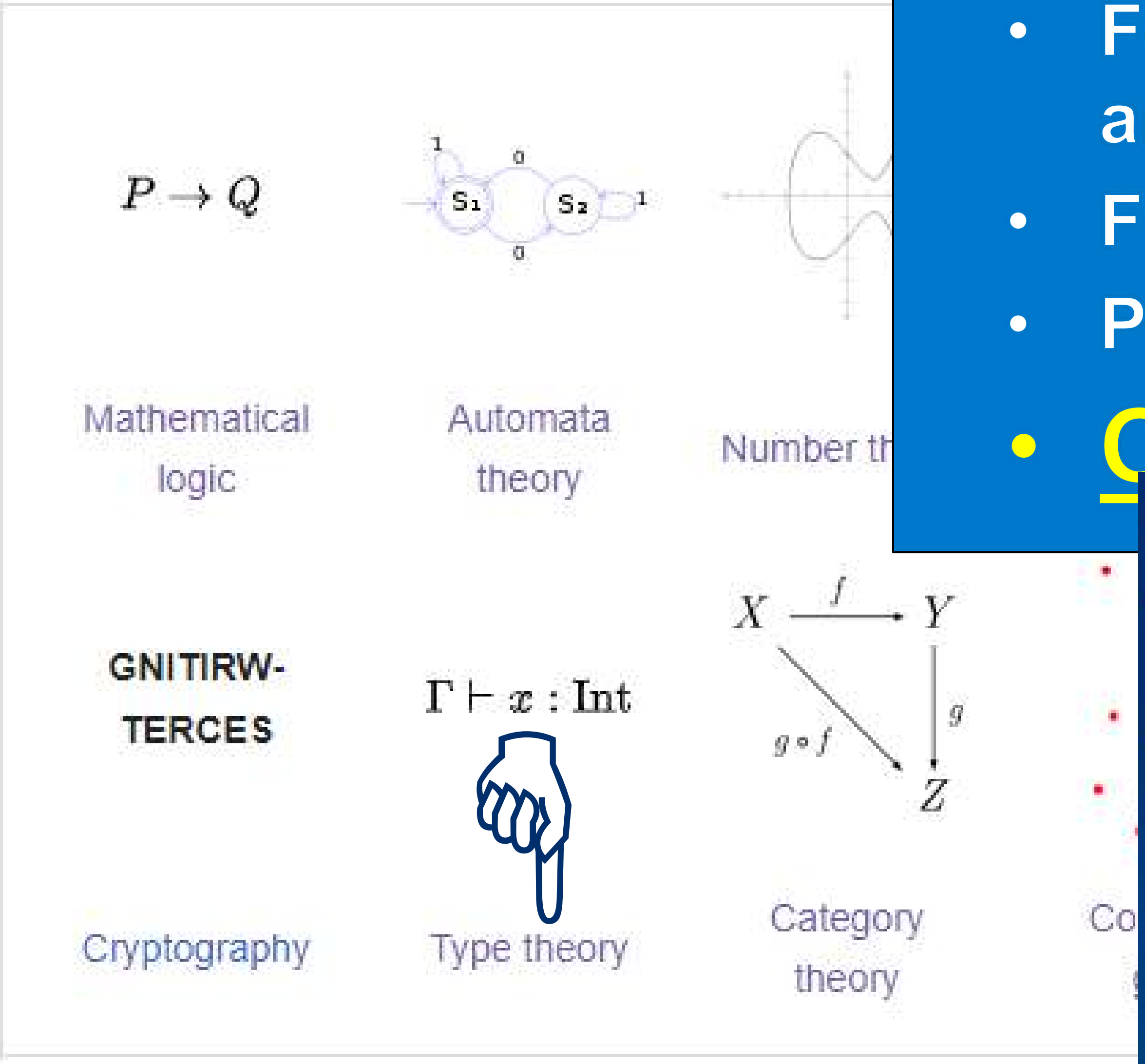
- This work concerns theoretical computer science (TCS)
- What is **TCS**?
- **UK Research and Innovation [7]** says:

  This area explores the fundamental and foundational aspects of computers and computation. Aiming to improve understanding of computation and its capabilities, limitations and future potential, this research area encompasses research around logic and semantics, and the study of algorithms, complexity and automata.

Some theoretical computer scientists would object to "computers" in the above definition.

Background: Theoretical computer science (TCS)

Over to Wikipedia: [8]



Also:

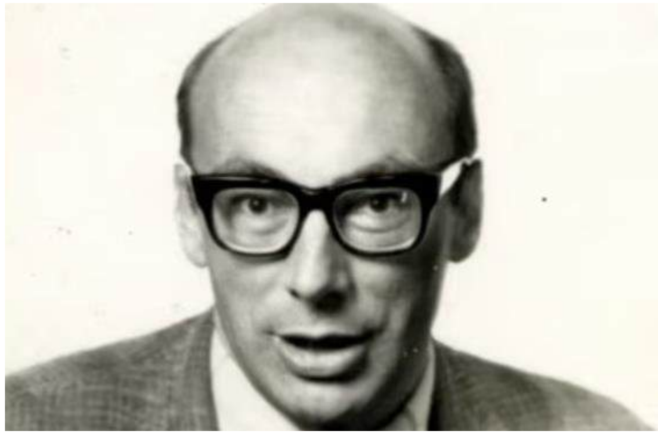
- Formal specification, verification and validation
- Functional programming
- Programming language semantics
- ONTOLOGIES

[9]

Background: Functional Programming: Reasons…

NPL

"LISP brought the class of entities that are denoted by expressions a programmer can write nearer to those that arise in models of physical systems and in mathematical and logical systems."



Peter Landin, 1930 - 2009

P. J. Landin, The Next 700 Programming Languages, March 1966 [10]

- Also applies to newer functional languages, such as Haskell and Idris2
- Worth exploring by NPL…?

Case Studies: Resistance Calibration [11]

Unitless, no dimensions … Time / s T Voltage / V $ML^2T^{-3}I^{-1}$

$$\begin{pmatrix} d_{nc1} & d_{c1} & n_1 & c_1 & t_1 \\ d_{nc2} & d_{c2} & n_2 & c_2 & t_2 \\ & & \vdots & & \\ d_{ncN} & d_{cN} & n_N & c_N & t_N \end{pmatrix} \times \begin{pmatrix} a \\ a_{cal} \\ c \\ c_{cal} \\ m \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{pmatrix}$$

What if the type of columns is a function from the column index to a combination of numeric / dimension?

Contents

Case Studies

- Realistic but not too complicated:
  1. Law of Propagation of Uncertainty
  2. GravCalc
  3. Resistance Calibration (Certificates DB)

NOTHING TO DO WITH LIVE SOFTWARE.

Case Study 1: Law of Propagation of Uncertainty

- Simplified example from calibration certificate:

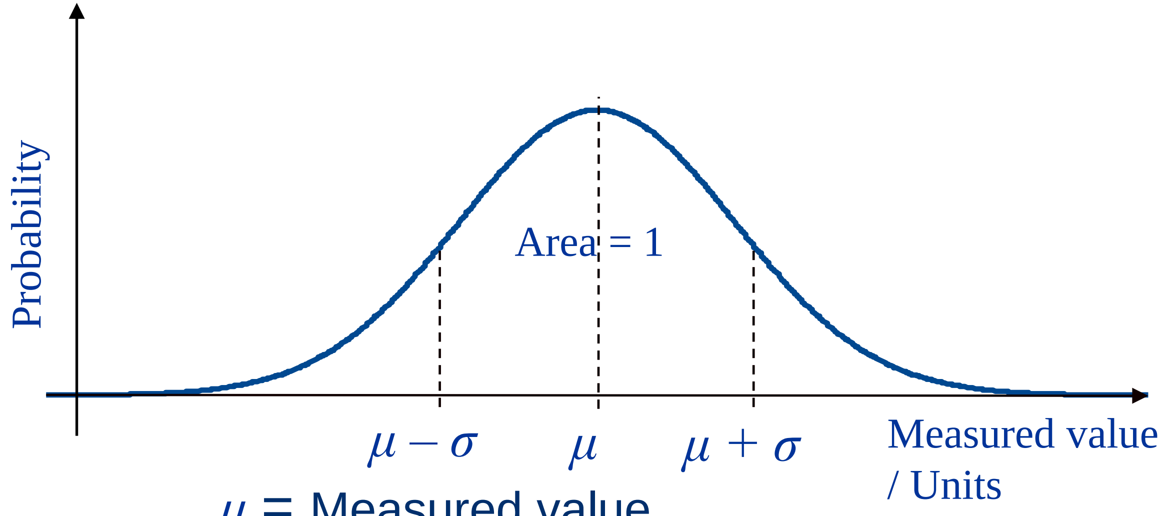
Approx. 95 % probability true value lies within quoted interval.

Value	Uncertainty	Mean Date
1·000 003 97 kΩ	± 0·05 ppm	4 September 2023

The reported expanded uncertainty is based on a standard uncertainty multiplied by a coverage factor $k = 2$, providing a coverage probability of approximately 95%.

Expanded uncertainty = $k \times \sigma$

Other distributions are available…



μ = Measured value

σ = Standard uncertainty

- Based on a **normal distribution**
- Often need to combine standard uncertainties from a variety of inputs (e.g., sensors)
- **Law of Propagation of Uncertainty**

Case Studies: Law of Propagation of Uncertainty

$$u_y = \sqrt{\sum_{j=1}^{N} c_j^2 u_j^2}$$

From Guide to the Expression of Uncertainty in Measurement (GUM) [12]

Where:

- u_y is the standard uncertainty of the output quantity.
- There are N input quantities, indexed by j
- c_j is a sensitivity coefficient associated with input j
- u_j is a standard uncertainty associated with input j
- The following assumptions are made:
  - There is one output quantity.
  - The input quantities are independent.
  - The output quantity can be expressed as an explicit function of the input quantities: $Y = f(X_1, \cdots, X_N)$

A measurement result is incomplete without a quantitative statement about the quality of the measured value (in the form of an uncertainty), and hence the importance to metrology of making such statements trustworthy.

Case Studies: Law of Propagation of Uncertainty

NPL

MATLAB

```
% Assign number of input quantities.
  N = 4;
%
% Assign uncertainties associated with estimates of input quantities.
% Note this vector is a column vector (implemented in this case as a
% transpose of a row vector).
  ux = [0.1, 0.2, 0.3, 0.4]';
%
% Assign sensitivity coefficients. Again, it's a column vector.
  c = [0.5, 0.4, 0.3, 0.2]';
%
% Apply law of propagation of uncertainty (LPU) to evaluate standard
% uncertainty associated with estimate of output quantity.
  uy = 0;
  for j = 1:N
    uy = uy + c(j)^2*ux(j)^2;
  end % for j
  uy = sqrt(uy);
  uy_efficient = sqrt((c.^2)'*(ux.^2));
```

```
uy =
    0.152970585407784
uy_efficient =
    0.152970585407784
```

Case Studies: Law of Propagation of Uncertainty

NPL

```haskell
import Data.Matrix

-- Assign sensitivity coefficients.
c :: Matrix Double
c = fromLists [[0.5, 0.4, 0.3, 0.2]]

ux :: Matrix Double
ux = transpose (fromLists [[0.1, 0.2, 0.3, 0.4]])

uy :: Matrix Double
uy = fmap sqrt ((fmap (\x -> x*x) c) `multStd` (fmap (\x -> x*x) ux))

main :: IO ()
main = do
  putStrLn "Law of propagation of uncertainty (LPU)"
  putStrLn "Haskell example version 0.1.0 with Hackage."
  putStrLn (prettyMatrix uy)
```

Hackage :: [Package]

```
Law of propagation of uncertainty (LPU)
Haskell example version 0.1.0 with Hackage.
┌                     ┐
│ 0.15297058540778355 │
└                     ┘
```

Case Studies: Law of Propagation of Uncertainty

```
-- Use implementation of vectors by André Videla.
-- Overloads operators ^ and * for applying to elements of vectors.
import Data.Vector

-- Assign sensitivity coefficients.
c :: Vec Double
c = Vec [0.5, 0.4, 0.3, 0.2]

-- Assign uncertainties associated with estimates of input quantities.
ux :: Vec Double
ux = Vec [0.1, 0.2, 0.3, 0.4]

uy :: Double
uy = sqrt (sumVec (c.^2 * ux.^2))

main :: IO ()
main = do
  putStrLn "Law of propagation of uncertainty (LPU)"
  putStrLn "Haskell example version 0.1.0 with Andre vectors."
  putStrLn (show uy)
```

```
Law of propagation of uncertainty (LPU)
Haskell example version 0.1.0 with Andre vectors.
0.15297058540778355
```

Vector module by André

Case Studies: Law of Propagation of Uncertainty

```
import Data.Vector

-- Assign sensitivity coefficients.
c : Vect 4 Double
c = [0.5, 0.4, 0.3, 0.2]

-- Assign uncertainties associated with estimates of input quantities.
ux : Vect 4 Double
ux = [0.1, 0.2, 0.3, 0.4]

uy : Double
uy = sqrt (sumVect ((c.^2) * (ux.^2)))

main : IO ()
main = do putStrLn "Law of propagation of uncertainty (LPU)"
          putStrLn "Idris2 example version 0.1.0 with Andre vectors."
          putStrLn (show uy)
```

Sized vectors

```
Law of propagation of uncertainty (LPU)
Idris2 example version 0.1.0 with Andre vectors.
0.15297058540778355
```

Vector module by André

Case Studies: Law of Propagation of Uncertainty

```python
# Assign sensitivity coefficients.
c = np.array([0.5, 0.4, 0.3, 0.2])
c = np.square(c)


# Assign uncertainties associated with estimates of input quantities.
# Note this vector is a column vector (implemented in this case as a
# transpose of a row vector)
ux = np.array([0.1, 0.2, 0.3, 0.4])
ux = np.square(ux)

uy = np.sqrt(np.matmul(c,ux.transpose()))

print("Law of propagation of uncertainty (LPU)")
print("Python example version 0.1.0 with NumPy")
print(f'{uy:.17f}')
```


```
Law of propagation of uncertainty (LPU)
Python example version 0.1.0 with NumPy
0.15297058540778355
```

Case Studies: GravCalc

- Realistic example of metrology software for coding in a functional language
- GravCalc [13], calculates the **amount fraction** and **uncertainty** of all components in **gravimetrically** prepared gas mixtures using the method described in ISO 6142 [14].
- The composition of the final gas mixture is, by the principle of the gravimetric method, defined by the mass of each component.
- Version currently in use coded in Visual Basic 6

C# version to be released

Case Study 2: GravCalc: Amount Fraction

$$y_k = \frac{\sum_{j=1}^{n} \frac{m_j y_{k,j}}{\sum_{i \in I(:j)} y_{i,j} M_i}}{\sum_{j=1}^{n} \frac{m_j}{\sum_{i \in I(:j)} y_{i,j} M_i}}$$

y_k = amount fraction of component k

m_j = mass taken from parent cylinder j

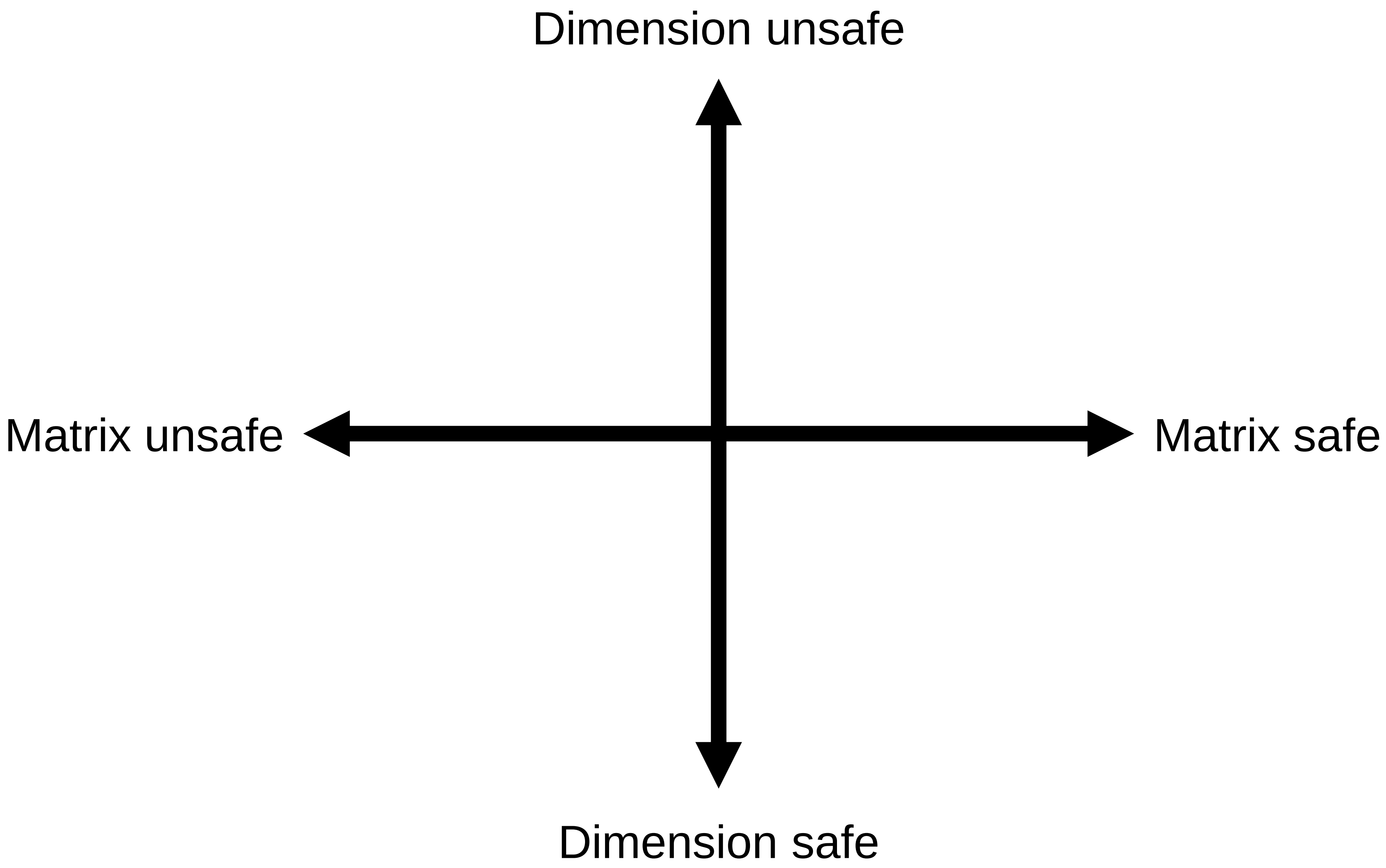
$y_{k,j}$ = amount fraction of component k in parent cylinder j

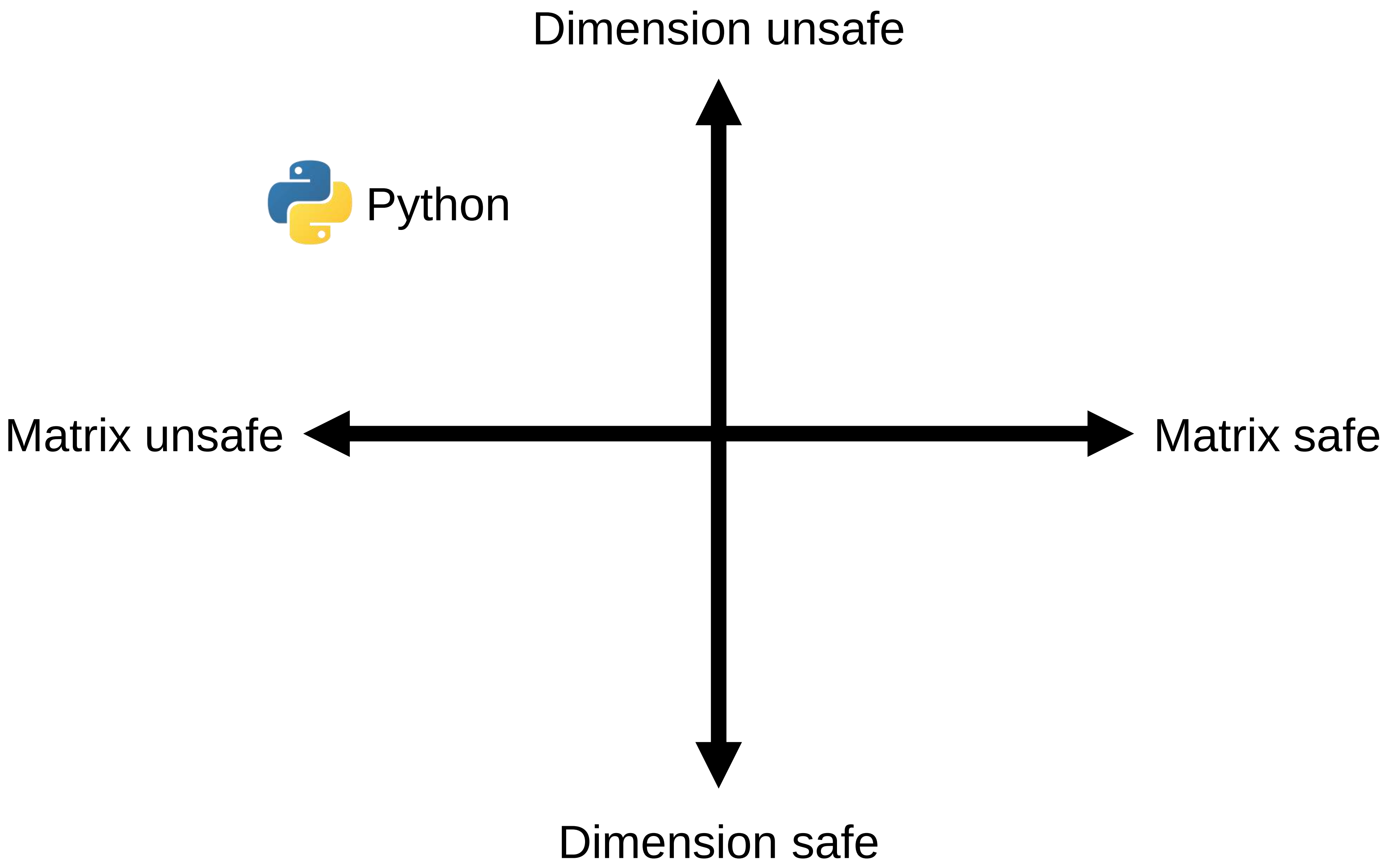
$y_{i,j}$ = amount fraction of component i from parent cylinder j

M_i = relative molar mass of component i

$I_{i,j}$ = indices of those components in parent cylinder j

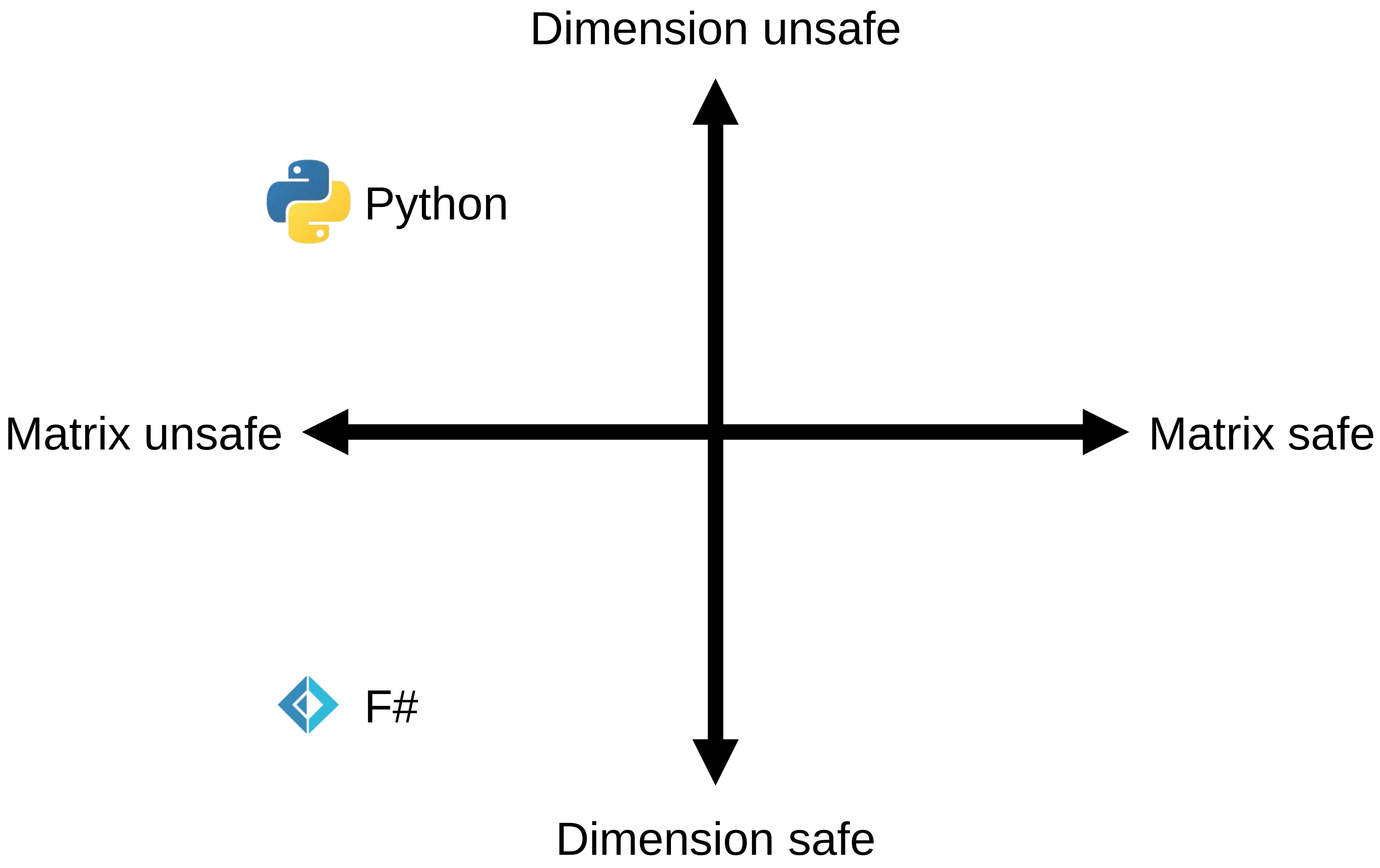
n = number of parent cylinders


Dimension unsafe
Matrix unsafe
Matrix safe
Dimension safe


Dimension unsafe
Python
Matrix unsafe
Matrix safe
Dimension safe

```
calc (components : Component array,masses: decimal array): (string * decimal) array
```

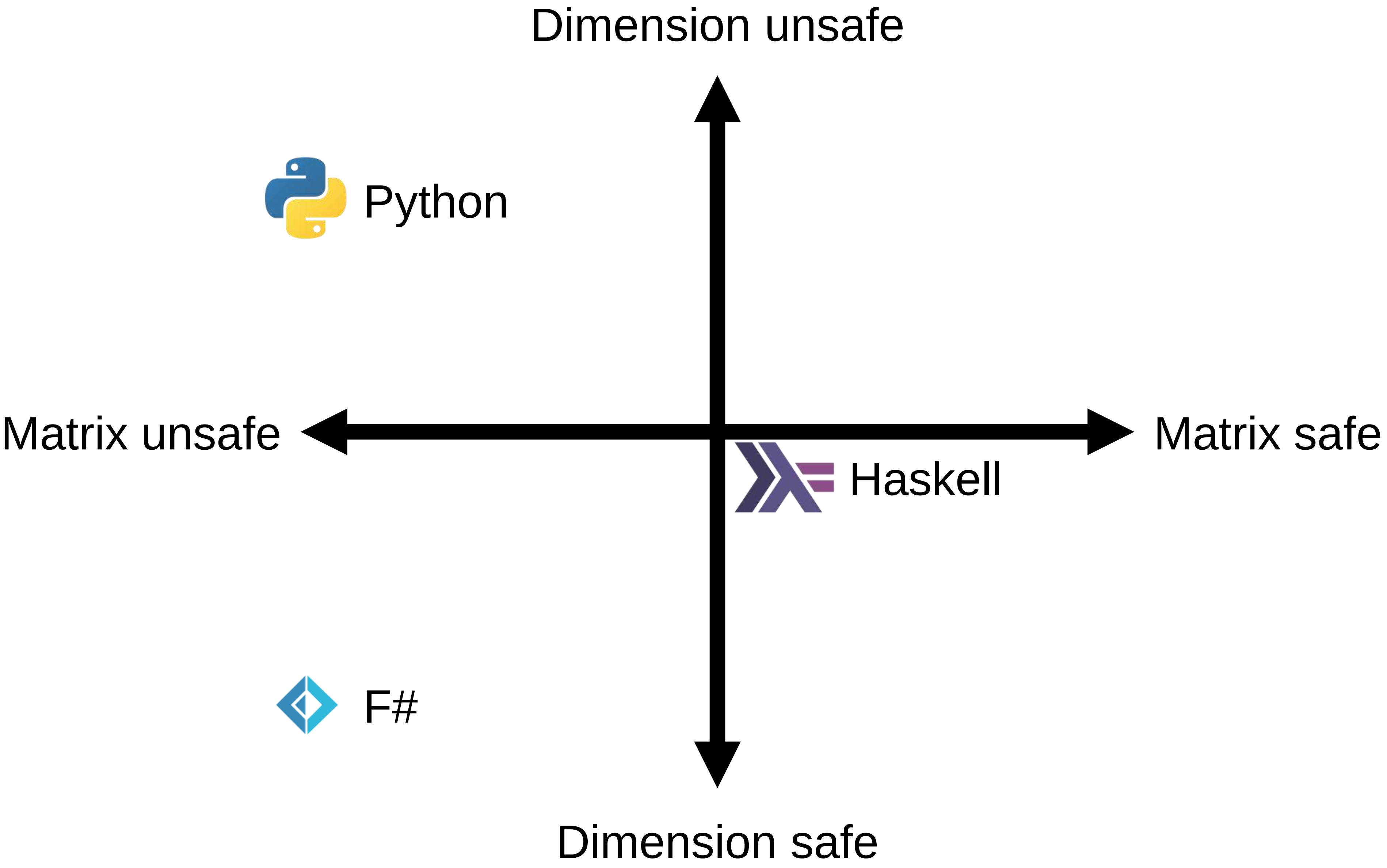
```
calc (components : Component array,masses: decimal<mol> array): (string * decimal) array
```


Dimension unsafe
Python
Matrix unsafe
Matrix safe
F#
Dimension safe

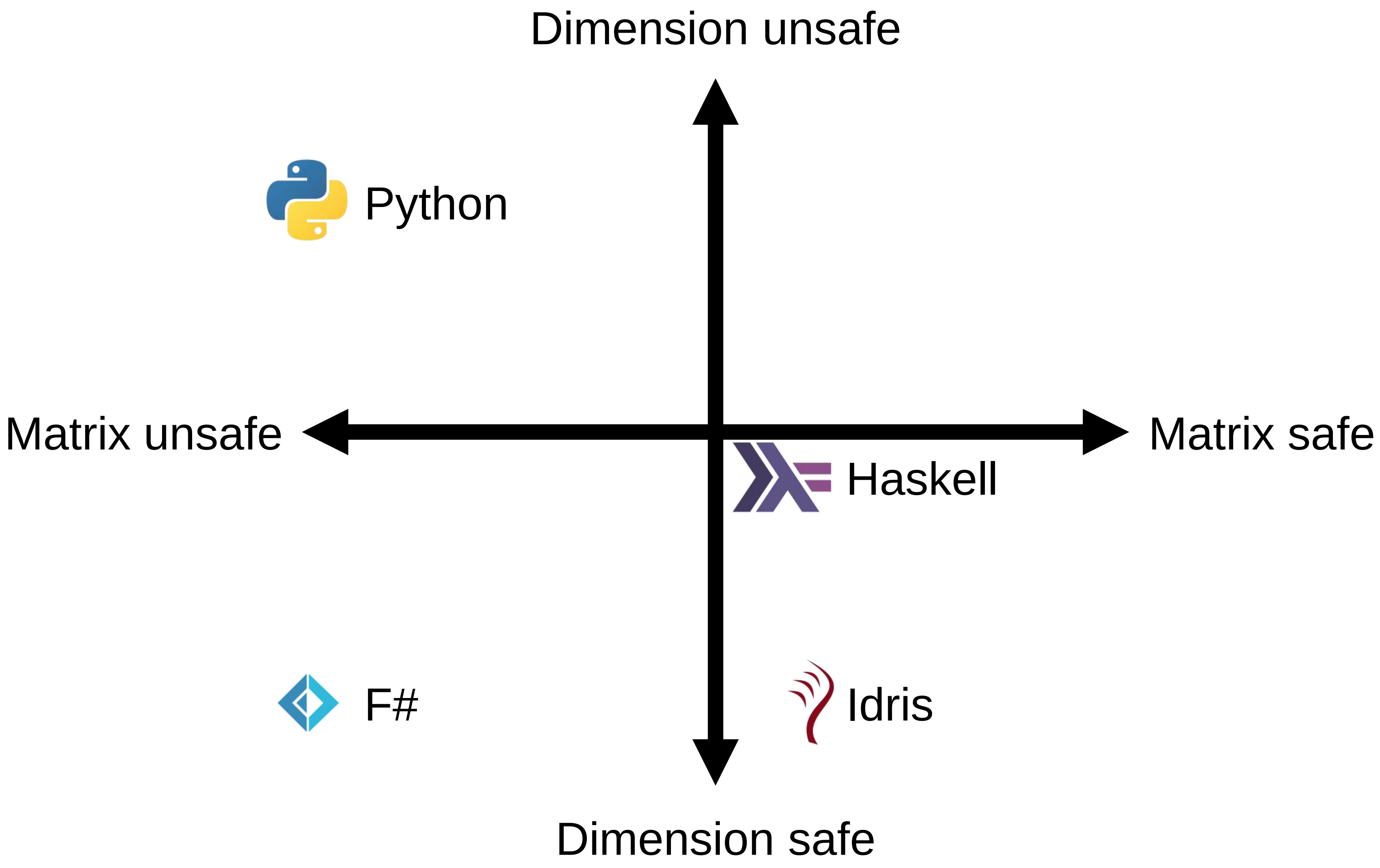
```haskell
gasMixCalc :: [Component] -> [Rational] -> [Rational]
```

```haskell
forall a. (Fractional a, Num a) => [Component a] -> [a] -> [(String, a)]
```

```haskell
gasMixCalc :: forall a. (Fractional a)
    => [Component (AmountOfSubstance SI) a]
    -> [AmountOfSubstance SI a]
    -> [(String, a)]
```


Dimension unsafe
Python
Matrix unsafe
Matrix safe
Haskell
F#
Dimension safe

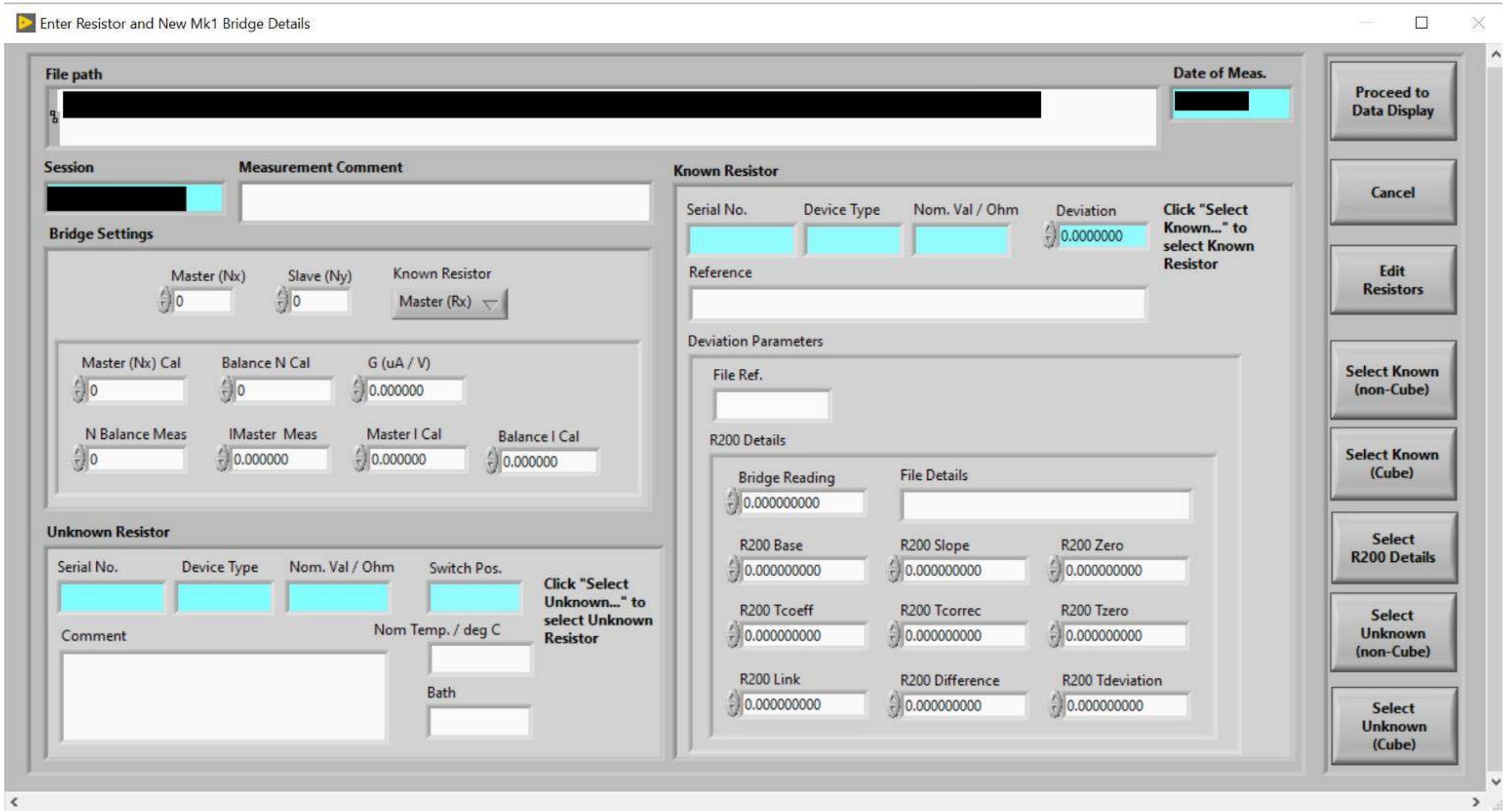
```
gasMixCalc : {0 g : Type} -> {0 num : g -> Type} ->
             (gn : GradedNum g num) =>
             {0 gr : g} ->
             {sampleCount, massCount : Nat} ->
             Vect sampleCount (Component massCount (num gr) (num u)) ->
             Vect massCount (num gr) ->
             Vect sampleCount (String, num u )
```


Dimension unsafe
Python
Matrix unsafe
Matrix safe
Haskell
F#
Idris
Dimension safe

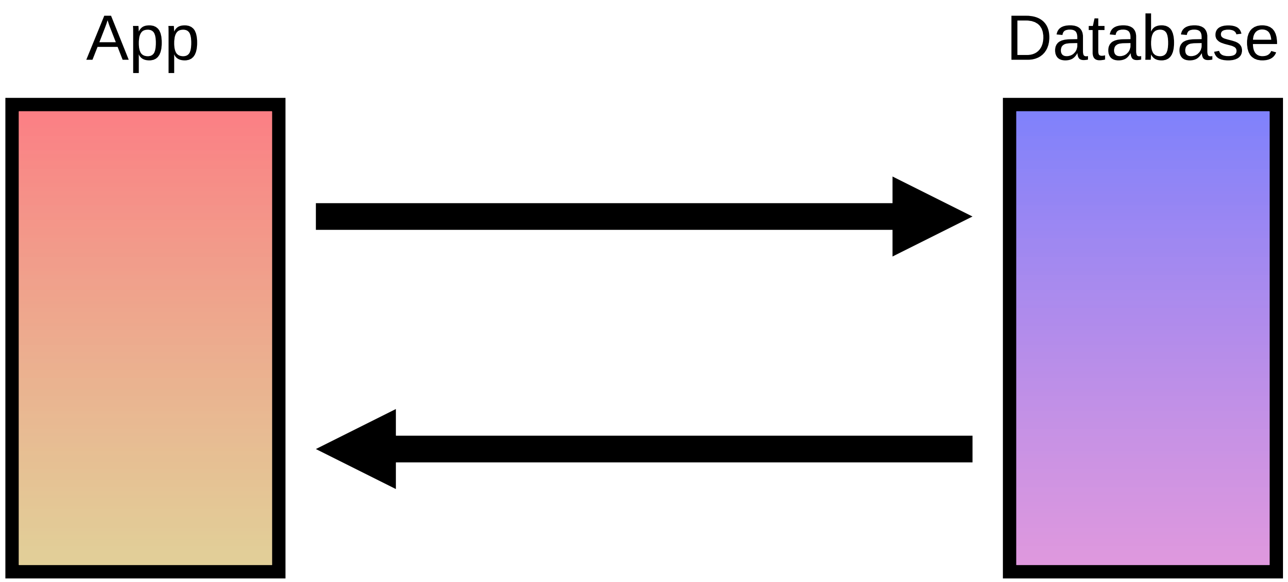
Conclusion

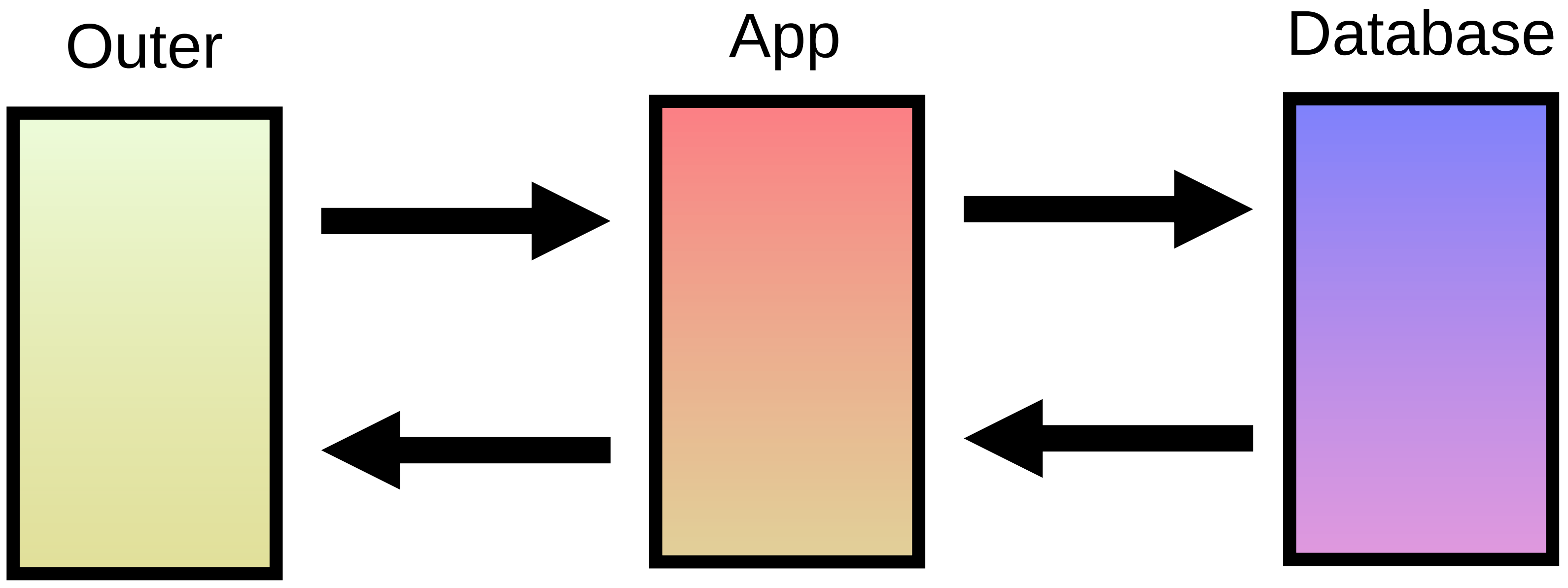
- More Types -> More Safety
- More Types -> More Expressivity
- Usability Questions

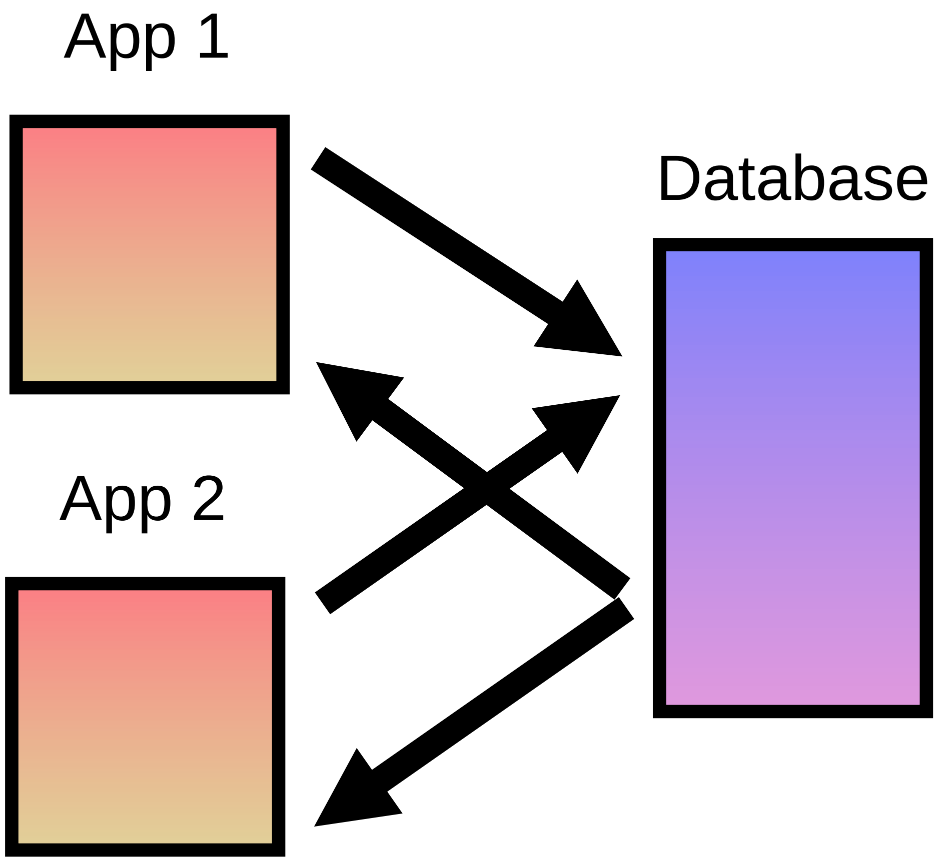
Case Study 3: Resistance Calibration

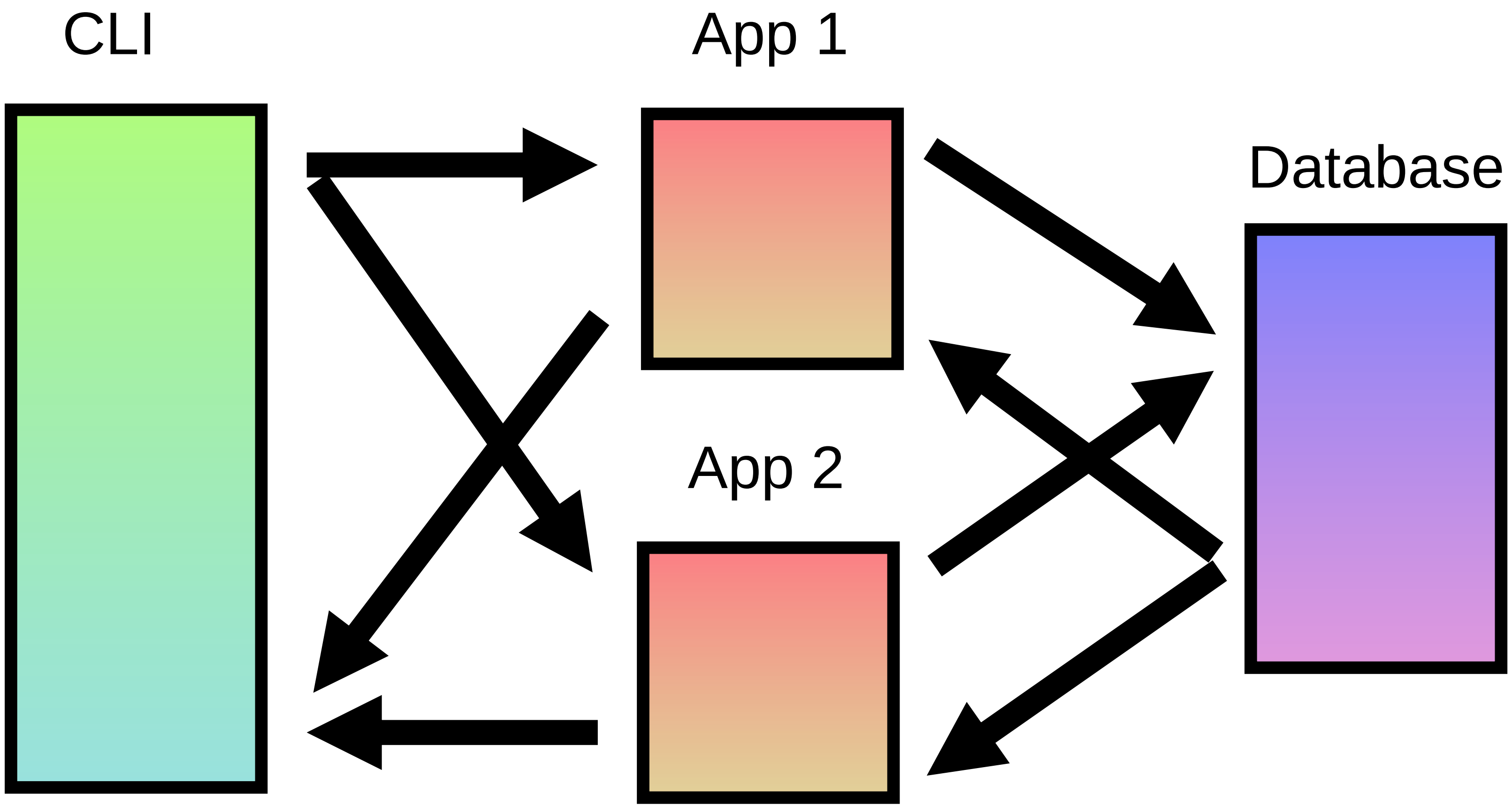


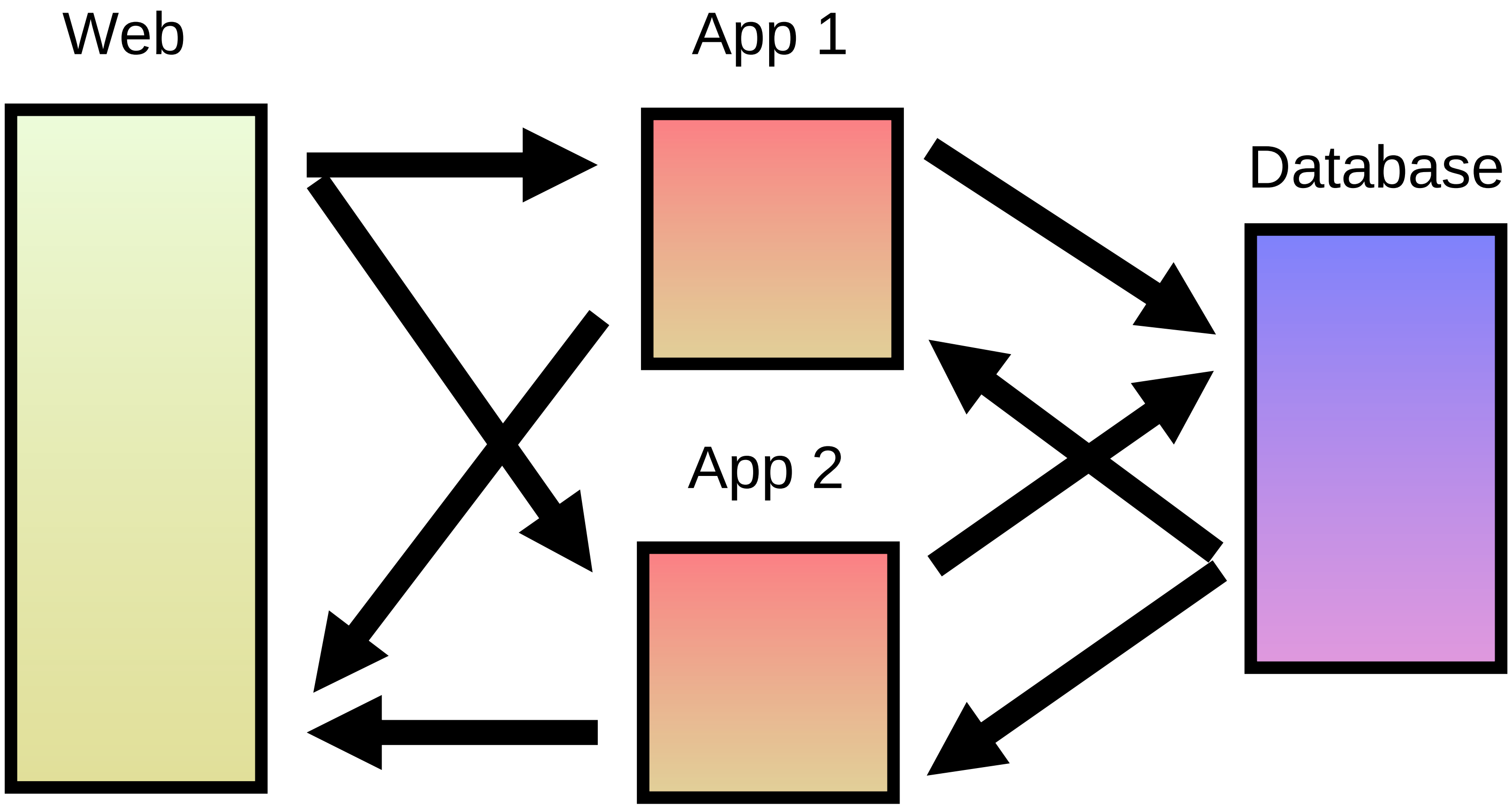
First: Theory


App
Database


Outer
App
Database

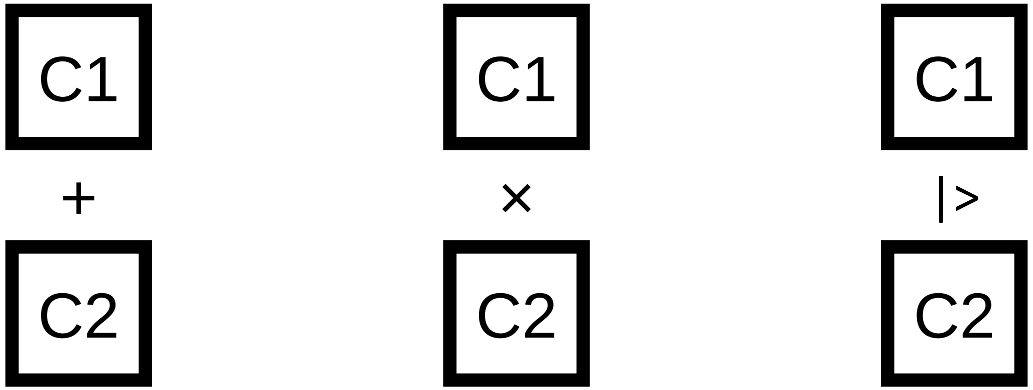

App 1
Database
App 2


CLI
App 1
App 2
Database

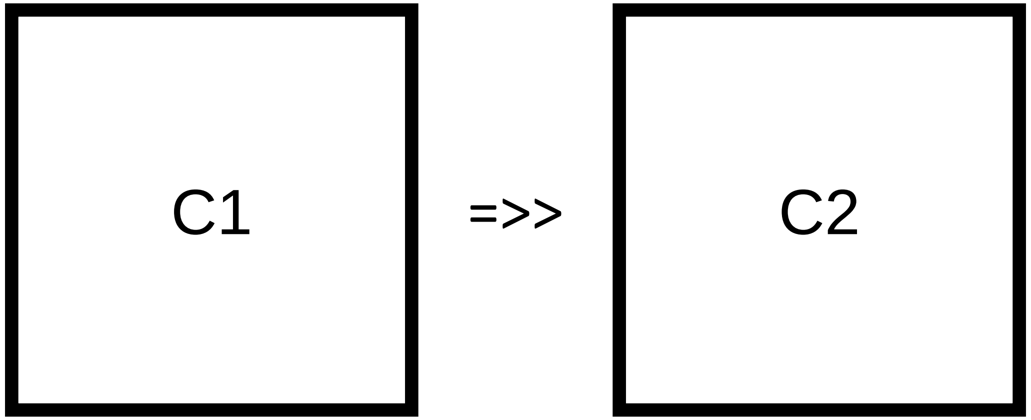

Web
App 1
Database
App 2

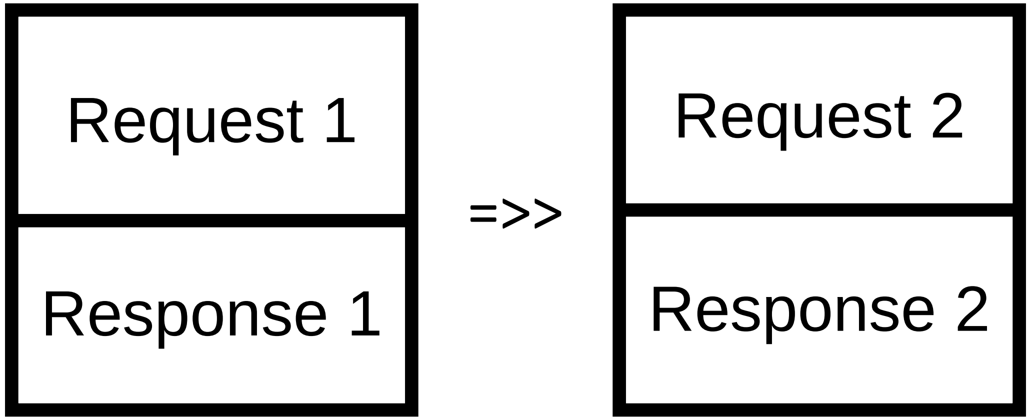
Containers

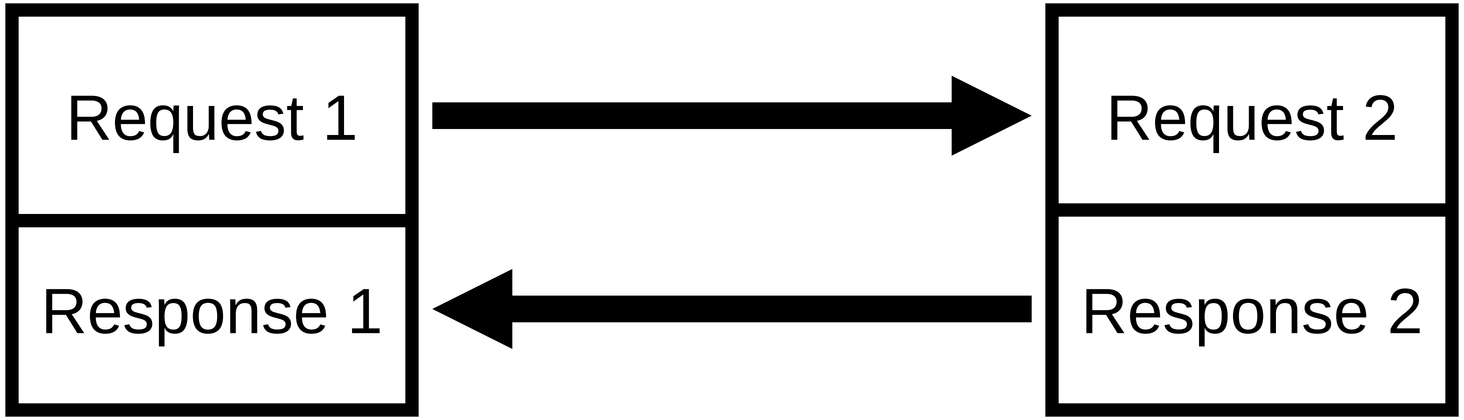
(Polynomial Functors)


C1
+
C2
C1
×
C2
C1
|>
C2

C1 =>> C2


C1
=>>
C2


Request 1
Response 1
=>>
Request 2
Response 2


Request 1
Response 1
Request 2
Response 2

C1 =>> C2 =>> C3

Container = API

Application
Programming
Interface

$$\frac{\Gamma \vdash S \text{ Type} \qquad \Gamma, S : \mathit{Type} \vdash P \text{ Type}}{\Gamma \vdash S \rhd P : \text{Container}}\text{Cont}$$

```
record API where
  constructor (!>)
  ||| The type of messages we send to the  system.
  message : Type
  ||| The type of responses we expect for each message we send.
  response : message -> Type
```

```
record (=%>) (c1, c2 : API) where
  constructor (<!)
  fwd : c1.message -> c2.message
  bwd : (x : c1.message) -> c2.response (fwd x) -> c1.response x
```

```
(|>) : a =%> b -> b =%> c -> a =%> c
(|>) x y = (y.fwd . x.fwd) <! (\z => x.bwd z . y.bwd (x.fwd z))
```

C1
+
C2

Handle Either 1 of 2 APIs

C1
|>
C2

Handle 2 APIs Sequentially

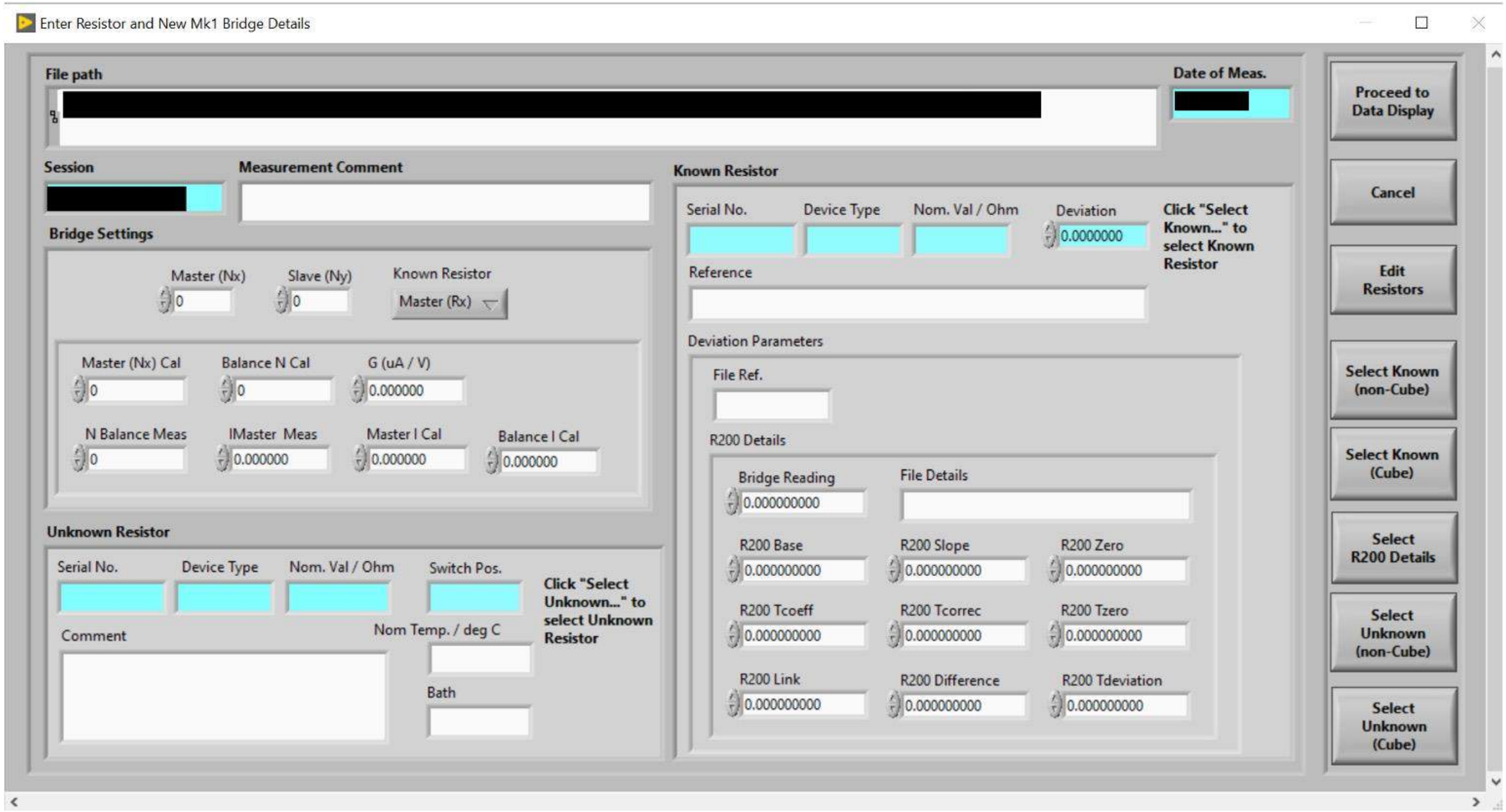
C1
×
C2

Handle 2 APIs Simultaneously

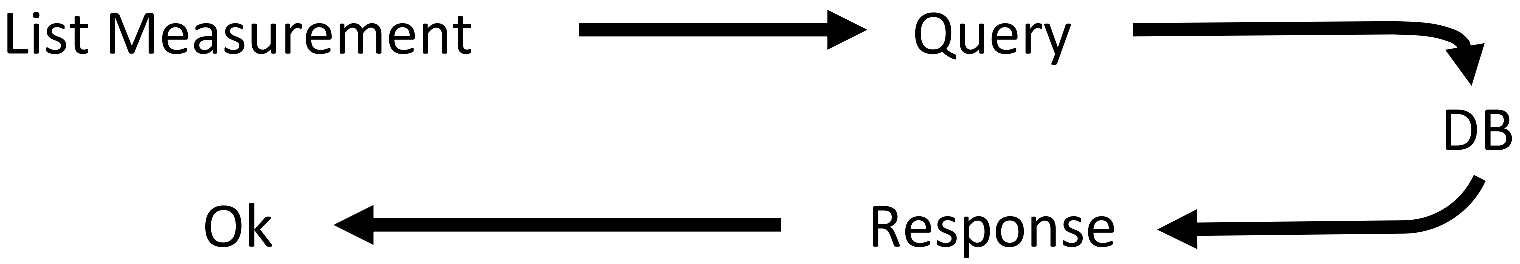
C1 =>> C2

Delegate

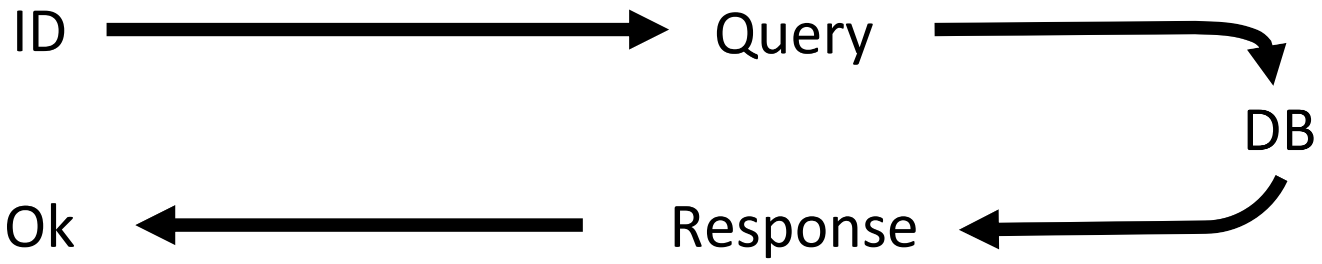
Back to Software



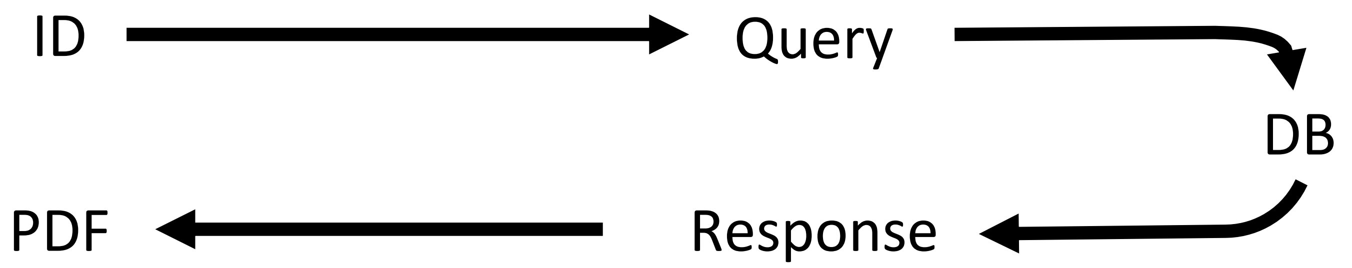
Send Measurement



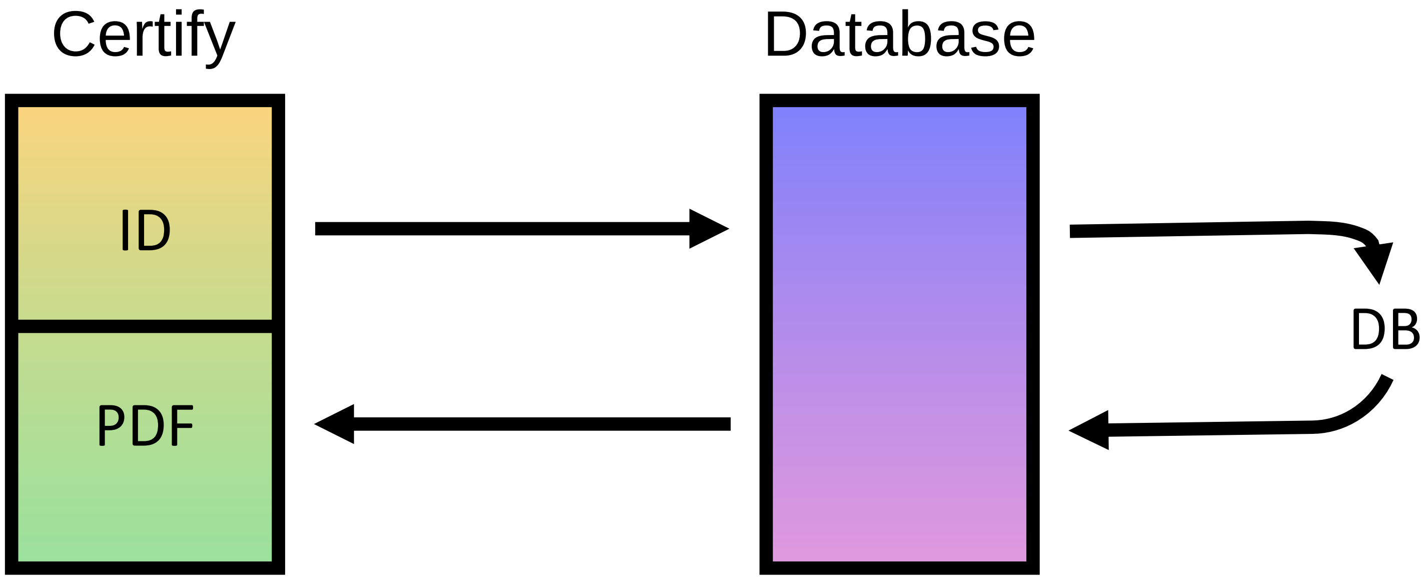
Analyse



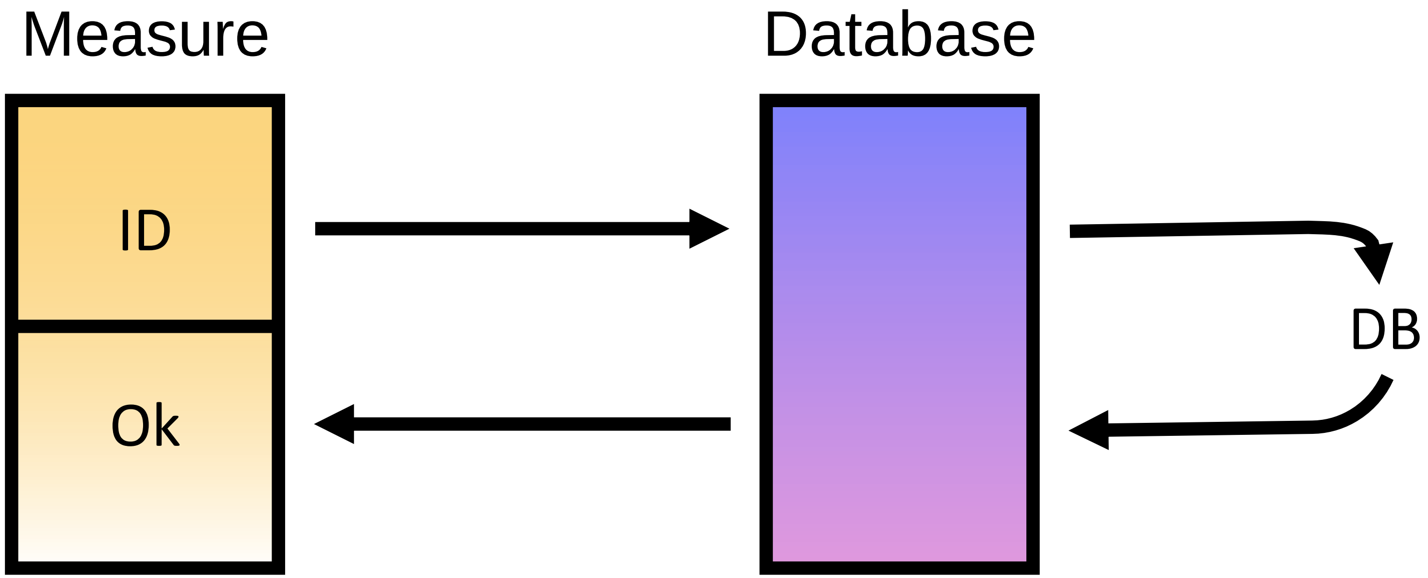
Certify



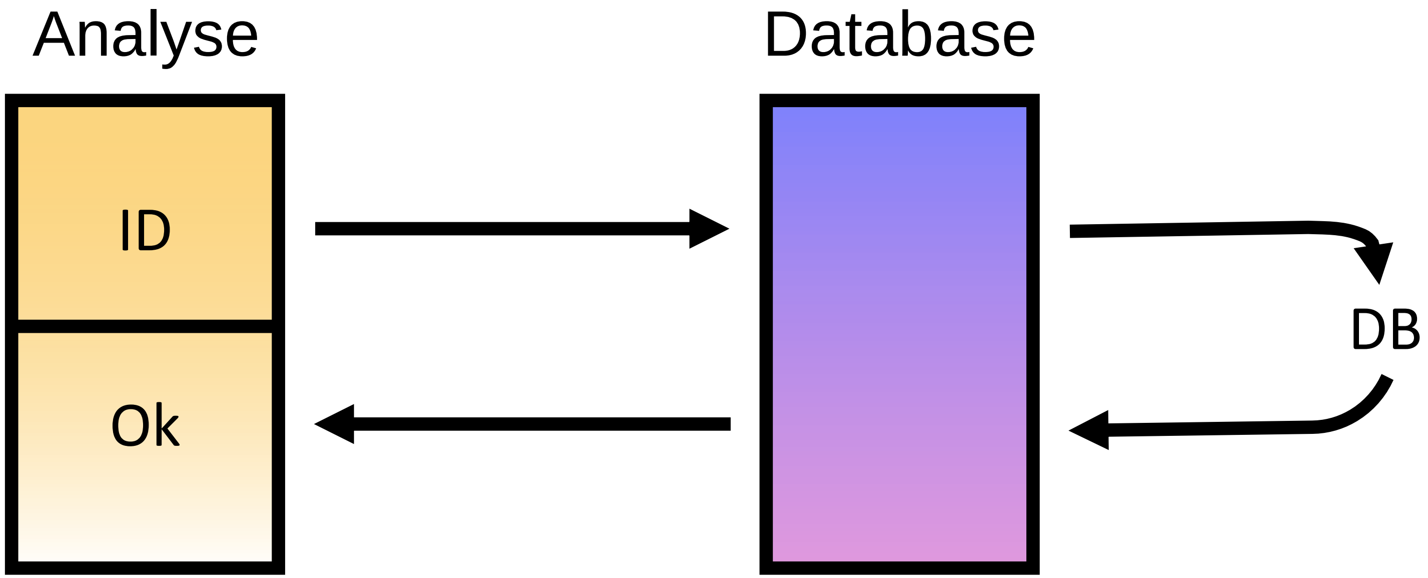
Certify

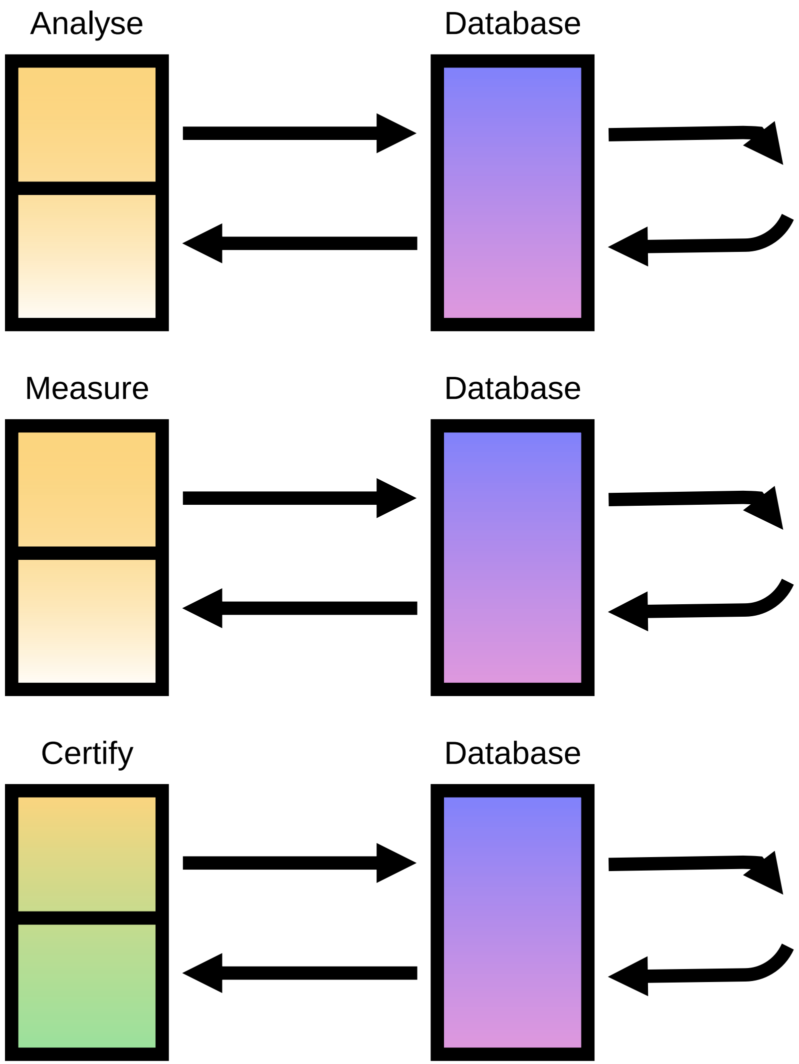


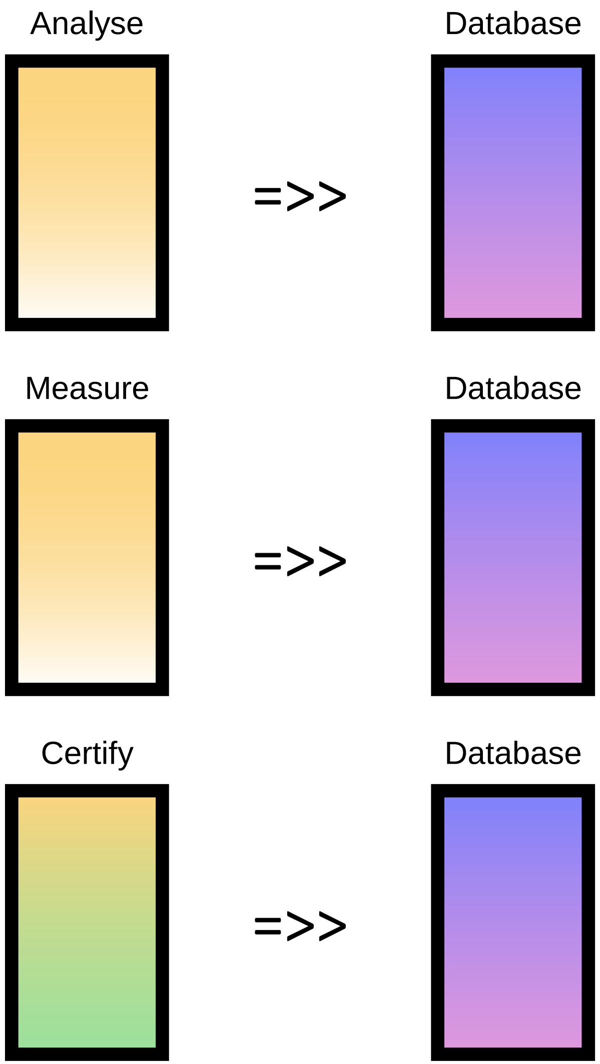
Send Measurements

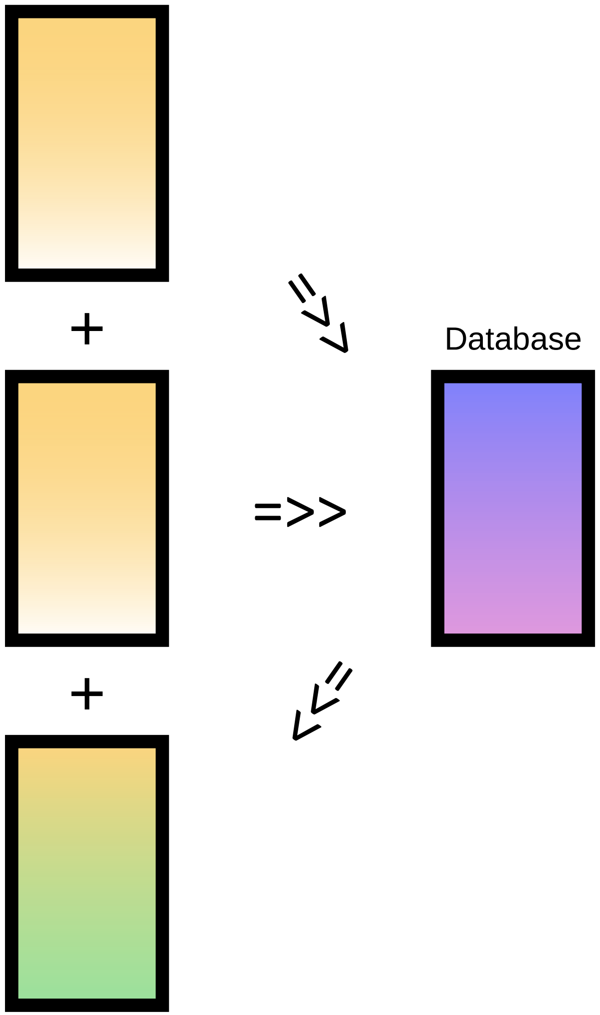


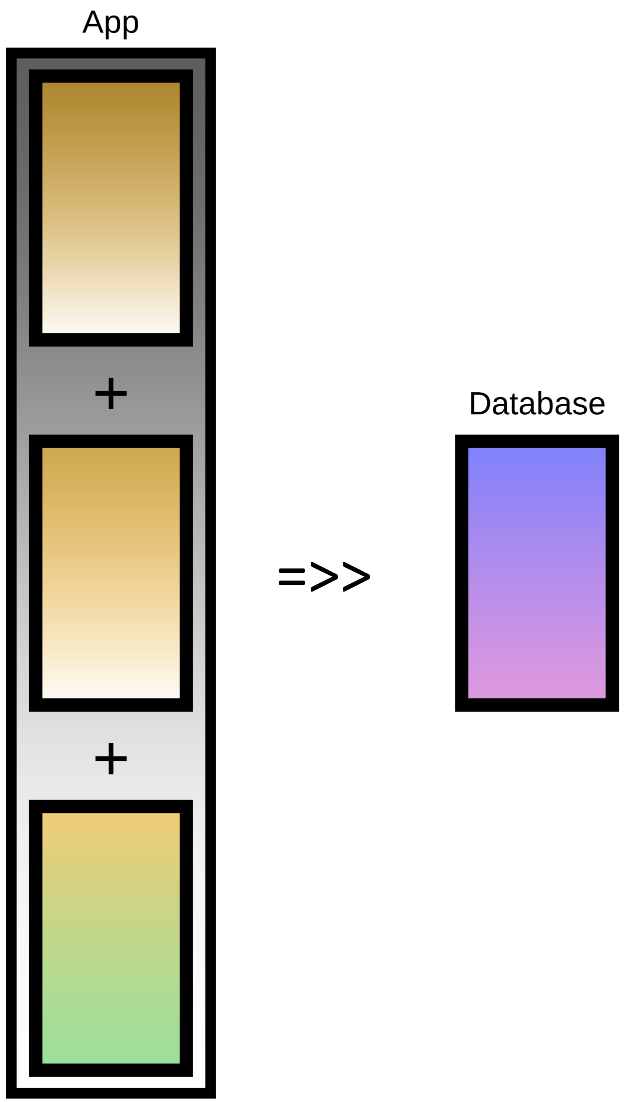
Analyse

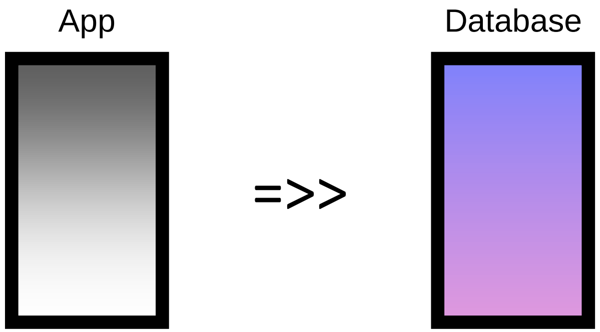


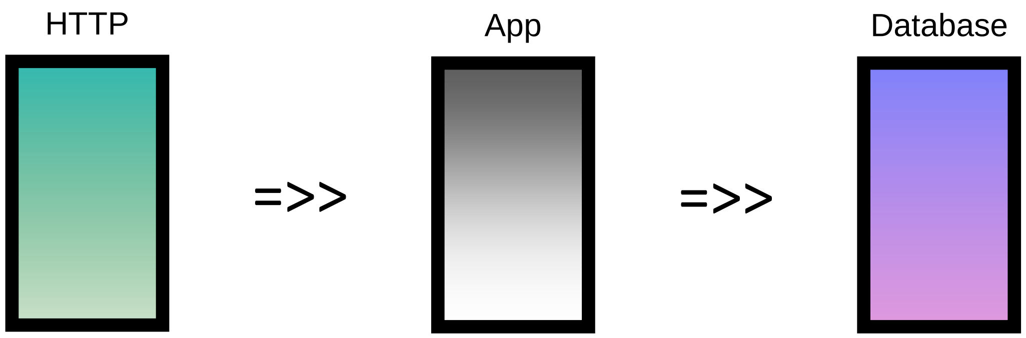

Analyse
Database
Measure
Database
Certify
Database

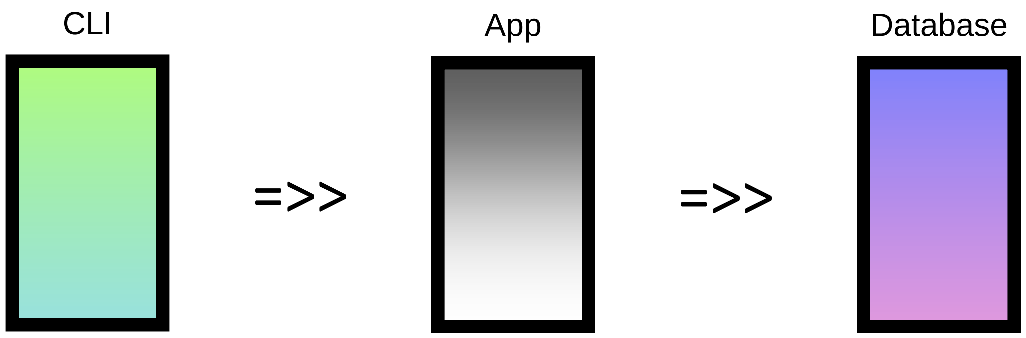

Analyse
=>>
Database
Measure
=>>
Database
Certify
=>>
Database


+
+
=>>
Database


App
+
+
=>>
Database


App
=>>
Database


HTTP
=>>
App
=>>
Database


CLI
=>>
App
=>>
Database

```
mainHTTP : DB => IO ()
mainHTTP = init >> http' (localhost 3000) (plugFrontend httpRouter)

mainREPL : DB => IO ()
mainREPL = init >> repl' (plugFrontend replRouter)

mainCLI : DB => IO ()
mainCLI = init >> cli' (plugFrontend cliRouter)
```

```
httpRouter : HTTP =%> MaybeAll AppAPI

replRouter : REPL =%> MaybeAll AppAPI

  cliRouter : CLI =%> MaybeAll AppAPI
```

Conclusion

- Very Flexible
- Unparalleled Abstraction Level
- Confined to Dependent Types
- Requires Better Ergonomics & Tooling

Contents

Conclusions

- Can this work be made accessible to non-specialists?
  - ❑ Jury is still out. Probably “yes”. More case studies / publications aimed at metrologists needed. Software tools will be vital. LabMate
    - ❖ May be easier to make NPL’s work more accessible to computer scientists.
  - ❑ Existing, more “unit aware”, languages (e.g., F#) worth exploring. Despite limitations noted with these languages.

Conclusions

- Value of theoretical computer science and functional programming for NPL / Data Science Dept.
  - ❑ Plenty of value. Has been for many years [1].
    - ❖ NPL should not be "passive acceptor" of software tools / computer science theory. Should be helping "shape the future".
  - ❑ Functional languages as prototyping tools. Alternative to MATLAB.
  - ❑ Ontologies, big crossover.
- Or for **anyone** who codes… just a little **appreciation** helps.
  - ❑ Example, lambda expressions in Python [15].

Powerful tool to make code more concise and reusable.

Conclusions

And then…

Contents

Acknowledgements

This work was undertaken jointly by the Mathematically Structured Programming Group of the University of Strathclyde and the National Physical Laboratory's Data Science department as part of Data Science's Tools for Trustworthiness National Measurement System (NMS) project 2023–2024.

Thanks to NPL colleague Nick Fletcher for his help and guidance. Thanks also to NPL colleagues Peter Harris, Louise Wright and Ian Smith for reviewing, and to Professor Neil Ghani for his support.

Contents

References

[1] K. Lines. NPL's Experience with Formal Aspects. https://www.youtube.com/watch?v=CjKIxwv-z6U. Webinar for BCS. 2021.

[2] Mathematically Structured Programming Group, Computer and Information Sciences, University of Strathclyde. https://msp.cis.strath.ac.uk/.

[3] C. McBride and F. Nordvall Forsberg. Dimensionally correct by construction: Type systems for programs | BCS FACS SG. https://www.youtube.com/watch?v=DVDvIoz9vE0. Webinar for BCS introducing concept of dependent types. 2021.

[4] A. Videla. Dependent types for practical use. https://www.youtube.com/watch?v=txJiuDM2gUM. Webinar for BCS on using dependent types. 2022.

[5] S. Lindley. C. McBride and C. McLaughlin. "Do Be Do Be Do". https://doi.org/10.48550/arXiv.1611.09259.

[6] BSI. Information technology — Systems trustworthiness. BS 10754-1:2018.BSI Group, 2018.

[7] UK Research and Innovation. Area of investment and support: Theoretical computer science. https://www.ukri.org/what-we-do/browse-our-areas-of-investment-and-support/theoretical-computer-science/.

[8] Wikipedia. Theoretical computer science. https://en.wikipedia.org/wiki/Theoretical_computer_science.

[9] J-L. Hippolyte. "The SI Digital Framework: Underpinning FAIR measurement data". In: The Newsletter of the Formal Aspects of Computing Science (FACS) Specialist Group (2024). https://www.bcs.org/media/1wrosrpv/facs-jul24.pdf.

[10] P. Landin. "The next 700 programming languages". In: Communications of the ACM 9 (1966). https://dl.acm.org/doi/10.1145/365230.365257.

[11] Conor McBride et al. "LabMate: A prospectus for types for MATLAB". In: Measurement: Sensors (2025), p. 101460. ISSN: 2665-9174. URL: https://www.sciencedirect.com/science/article/pii/S2665917424004367.

[12] BIPM et al. JCGM GUM-1:2023 Guide to the expression of uncertainty in measurement — Part 1: Introduction. Joint Committee for Guides in Metrology, JCGM GUM-1:2023. URL: https://www.bipm.org/en/committees/jc/jcgm/publications.

[13] Gas metrology software. https://www.npl.co.uk/products-services/gas/gas-metrology-software.

[14] ISO. Gas analysis. Preparation of calibration gas mixtures. Part 1: Gravimetric method for Class I mixture. ISO 6142-1:2015. International Organization for Standardization, 2015.

[15] Python. Python Software Foundation. https://www.python.org

NPL

National Physical Laboratory

npl.co.uk