BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY
Office of Science

# Idiomatic Vibe Testing with Julienne

Katherine Rasmussen, Damian Rouson, Dan Bonachea

go.lbl.gov/julienne

# Table of Contents

# Vibe Check

Raise your hand if you
have heard of vibe coding

# Vibe Check

Raise your hand if you
have used AI to help write emails

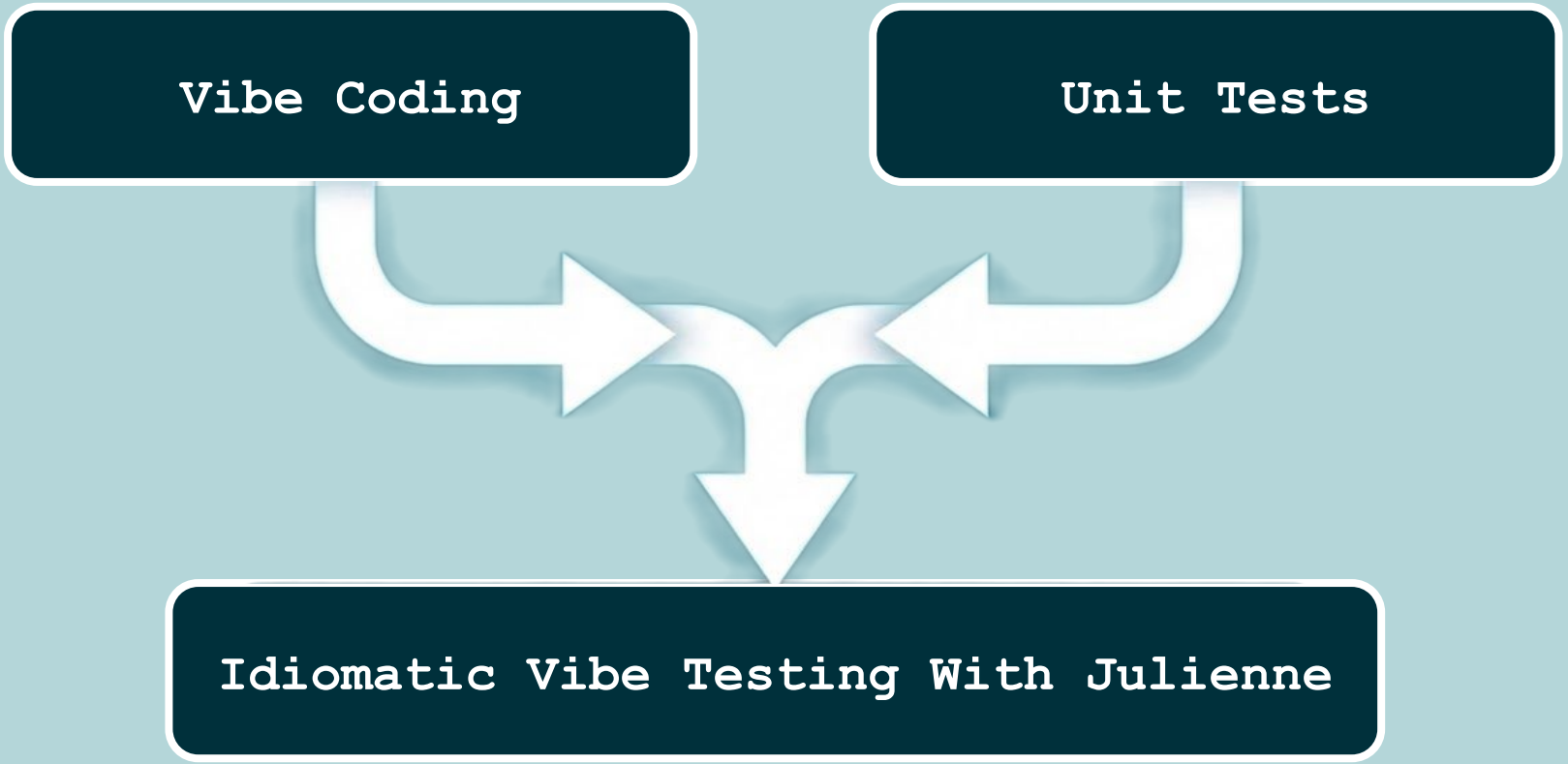
# Vibe Check

Raise your hand if you
used AI to help write code

# Vibe Check

Raise your hand if you
used AI in a vibe coding sense

# Background


Vibe Coding
Unit Tests
Idiomatic Vibe Testing With Julienne

# Vibe Coding

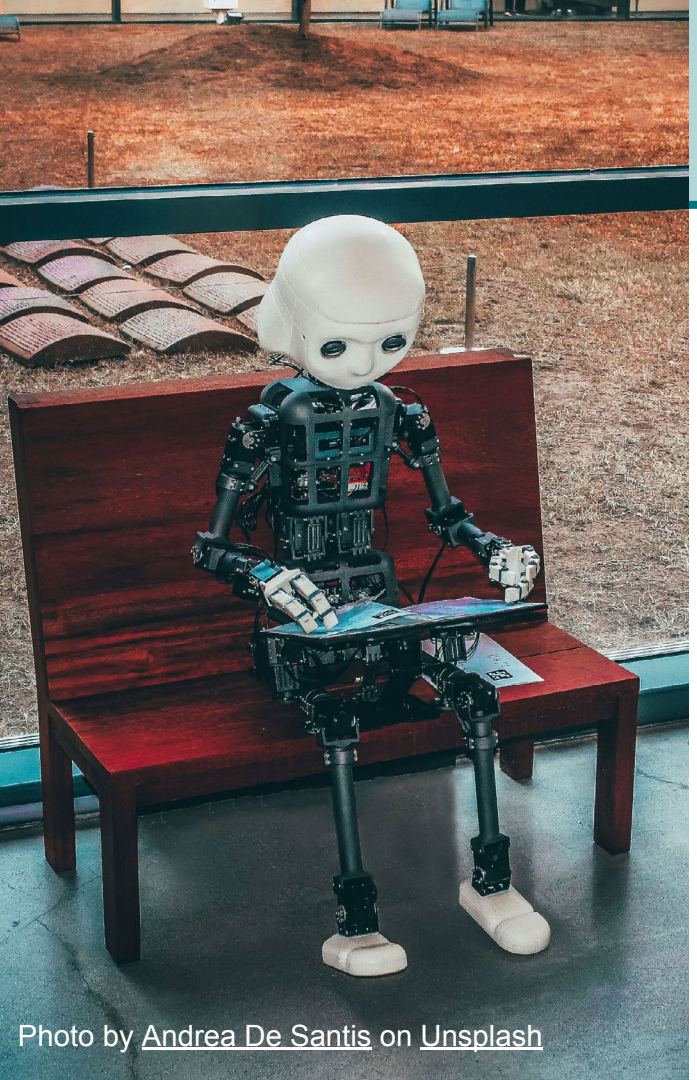


Photo by Andrea De Santis on Unsplash

# Vibe Coding
## — What is it?

- First coined in 2025 – dictionary entry 1 month later (!)
  - Andrej Karpathy
- New concept, definitions emerging
- AI tool (LLM) leads the coding
- Human in the loop, barely
- Lowers the barrier to entry for non-programmers

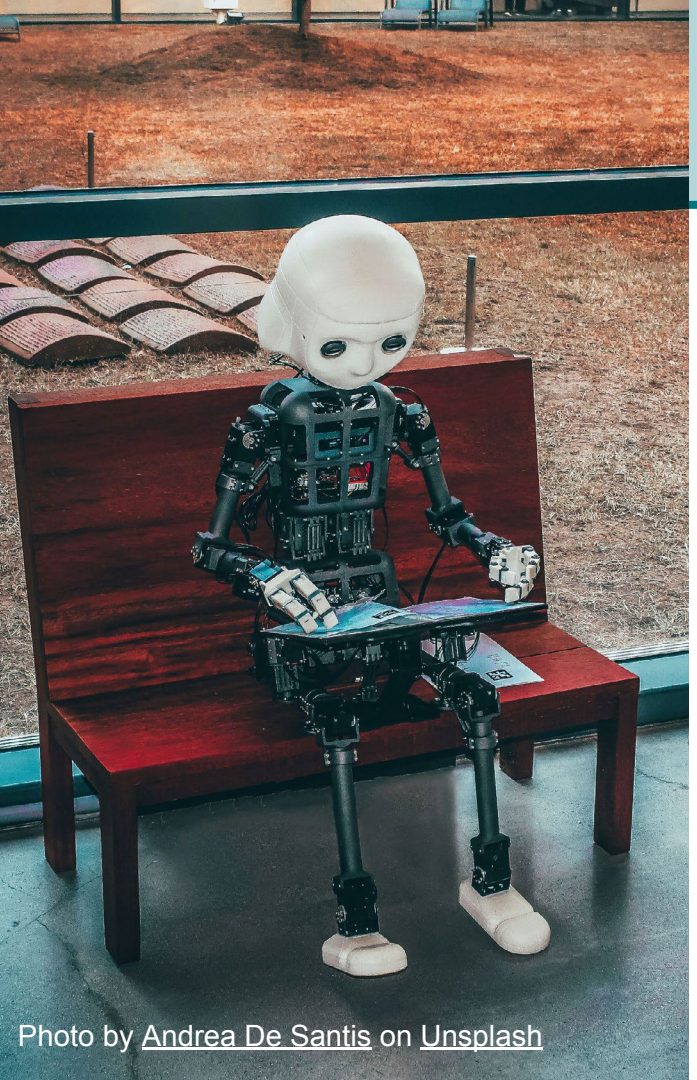


Photo by Andrea De Santis on Unsplash

# Vibe Coding
## — Pros and Cons

- Accessibility
  - Lowers the barrier to entry for non-programmers
- Reliability
  - Developer may not understand or read the generated code

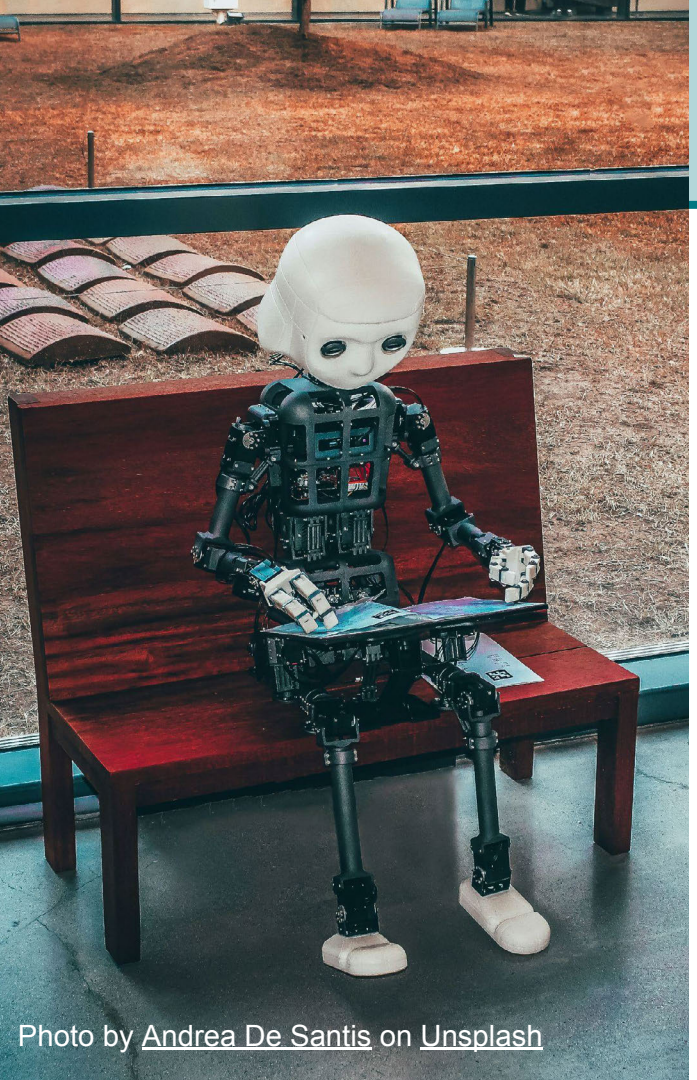


Photo by Andrea De Santis on Unsplash

# Vibe Coding
## — News Reports

Recent headlines:

- “The Vibe Coding Effect? Apple’s App Store Saw 84% Jump in New Apps in Quarter” - The Information
- “Apple Quietly Blocks Updates for Popular 'Vibe Coding' Apps” - MacRumors
- “Apple Pulls Vibe Coding App 'Anything' From App Store, Escalating Enforcement” - MacRumors

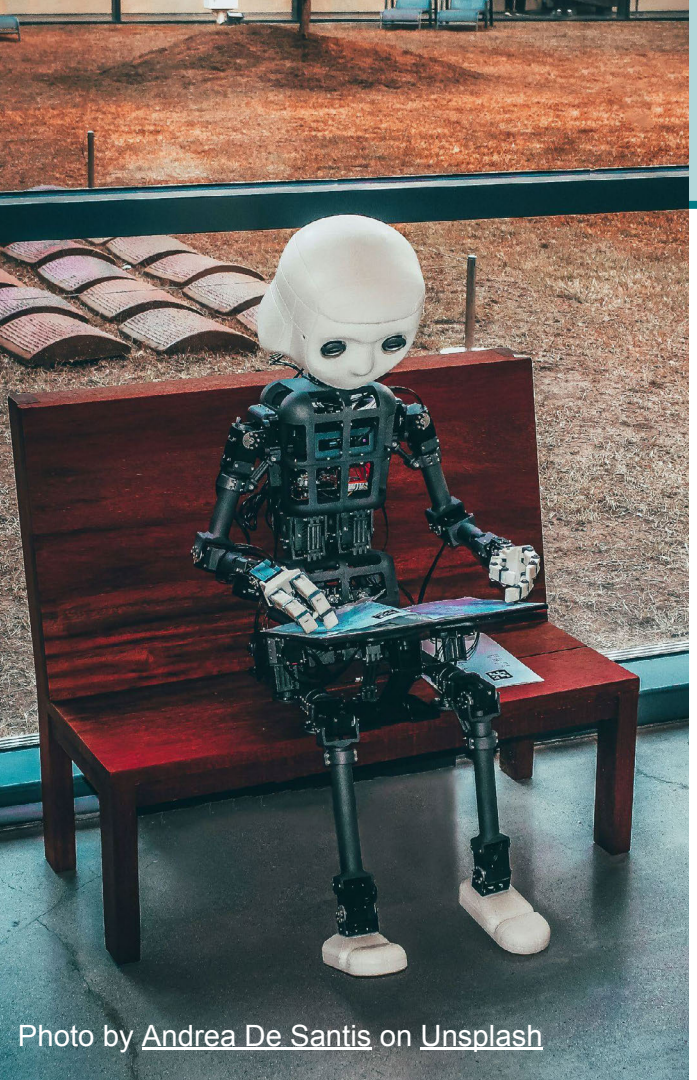


Photo by Andrea De Santis on Unsplash

# Vibe Coding
## — Use of Natural Language

- Coding - realm of programming languages
- LLMs - vibe coding - realm of natural languages
- Ambiguity in natural language
- Julienne natural language idioms
    - More on this later…

# Unit Tests

# Unit Tests

## The Basics

Important

Under Utilized in Scientific Software

# Unit Testing

- Test-driven development
  - Tests = requirement specifications
  - Write unit test first
  - Write the minimal passing code
  - Rinse and repeat
- Suite of unit tests that can run in seconds or minutes
- Continuous integration/Continuous Deployment (CI/CD)
  - Merge frequently
  - Automate unit testing
  - Test every merge

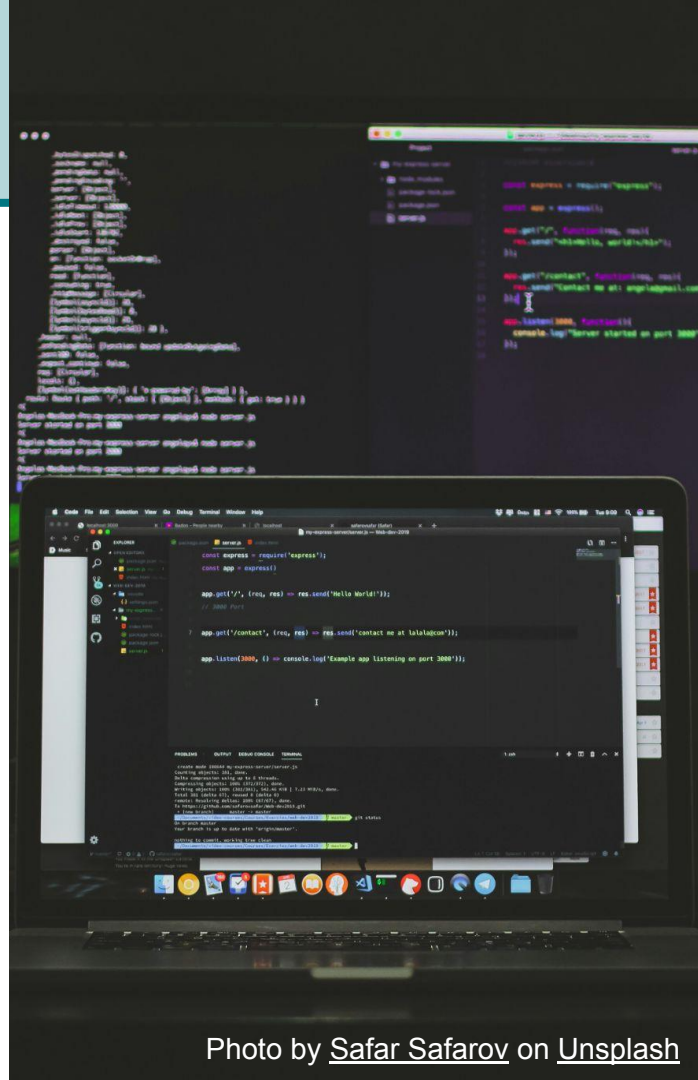


Photo by Safar Safarov on Unsplash

# Correctness Checking

- Assertions in unit tests to verify expected behavior:
  - Preconditions
    - Must be true before a procedure executes
  - Postconditions
    - Must be true after a procedure executes
  - Invariants
    - Must always be true (pre & post every function)
- Julienne provides assertions? …



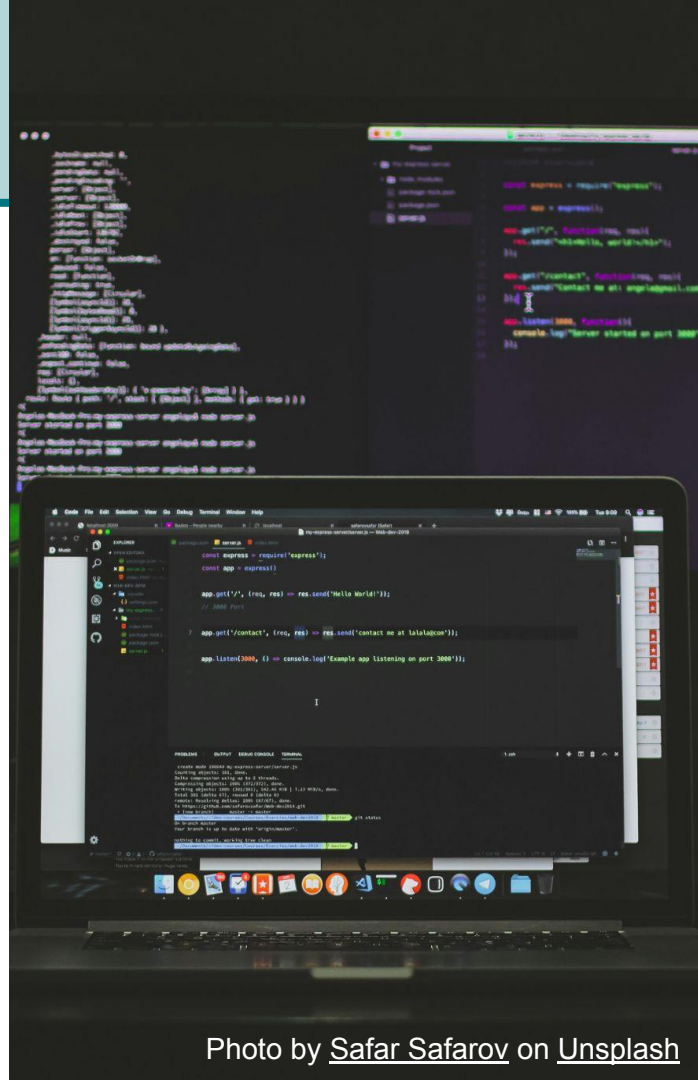


Photo by Safar Safarov on Unsplash

# Julienne

go.lbl.gov/julienne

Photo by Igor Miske on Unsplash

# Thinly Sliced Vegetables

## Julienne

A lightweight utility inspired by Veggies

# Julienne

- Began as Fortran unit-testing framework
- Grew to support natural-language idioms for
  - Assertions inside user code
  - Tests external to user code
- Tests can be single- or multi-image (parallel)
- Julienne's `scaffold` program
  - Reads class names from a JSON file
  - Generates a test-suite driver and sample tests
- Testing as specification
- `fpm` package

go.lbl.gov/julienne

Photo credit: Paul Hargrove

# fpm

- fpm (Fortran Package Manager)
  - Package manager and build system
  - Written in Fortran
  - Maintained by Fortran developer community
  - Automatically detects file dependencies
  - Builds Fortran, C, and C++ source code [e.g, SuperLU (C), Armadillo (C++)]
  - fpm itself is very easy to install
    - Available through package managers (Homebrew, Spack, etc.)
    - If using gfortran 13 or later, can also compile a one-file version of the `fpm` source code to install it

# Julienne Test Output

- Julienne output includes:
  - Test subject (e.g., a class or type-bound procedure)
  - Test description
  - Test outcome (pass, fail or skip)
  - Test diagnostics (only if a test fails)
- Will see more examples in Fiats and Caffeine Use Cases

# Testing with Fortran's Parallel Features

- Julienne cannot run (different) tests in parallel. However, it *can* run parallel tests.
- Parallel tests: Tests that invoke multi-image Fortran features
  - Every image runs every test
  - Only report that a test passes if it passes on all images
  - Only image 1 outputs the test outcome (pass/fail/skip)
  - Images collectively print diagnostics via Julienne's `co_characterize` subroutine
    - Option: Asynchronous diagnostic output via `-DASYNCHRONOUS_DIAGNOSTICS`

# Julienne Recent Developments

- Expanded compiler support:
  - LFortran 0.60.0: first supported version
  - LLVM 22: first multi-image (parallel) version
  - GCC 15 (updated version)
  - Intel 2025.21 (updated version)
  - NAG 7.2
- Expanded parallel testing:
  - **New:** LLVM Flang automatically calls PRIF/Caffeine/GASNet-EX
  - Still supported: GFortran/OpenCoarrays, NAG, and Intel
- **New:** `scaffold` program eases adoption by generating a test-suite driver program and sample tests for examplar classes named in a JSON file. (See the `demo` directory.)

# Julienne Recent Developments (cont.)

- Addition of `test_diagnosis_t` string concatenatation operator (`//`)
  - Used for appending user-defined messages to the automated diagnostic output
  - Ex: `test_diagnosis = i .equalsExpected. j // "my additional message here"`
- Idioms as a unifying theme across two disparate technologies:
  - Unit tests
  - **New:** Assertions (compiled only with `-DASSERTIONS`)
    - Ex: `call_julienne_assert(i .equalsExpected. j)`

# Julienne Idioms

```
test_diagnosis = test_diagnosis .also. &
     .all. (actual_coarse .approximates. expected_coarse &
           .within. crude_tolerance) &
     // " (coarse-grid 2nd-order .laplacian. sin(x))"
```

# Julienne Idioms – operators

| | |
|---|---|
| Less than operator: | `x .lessThan. y` |
| Greater than operator: | `x .greaterThan. y` |
| **Asymmetric** equality operator: | `i .equalsExpected. j` |
| Greater than or equal operator: | `i .isAtLeast. j` |
| Less than or equal to operator: | `i .isAtMost. j` |
| Verify alphabetical order operator: | `s .isBefore. t` |
| Verify reverse alphabetical order operator: | `s .isAfter. t` |

- Above operators operate on intrinsic types (numeric, character types, `type(c_ptr)`)
  - See README at [go.lbl.gov/julienne](go.lbl.gov/julienne) for details
- Above expressions return `test_diagnosis_t` object

# Julienne Idioms - operators

And operator: `a .also. b`

- Operates on `test_diagnosis_t` object
- Returns `test_diagnosis_t` object
- Putting it together:

```
(i .lessThan. j) .also. (k .equalsExpected. m)
```

# Julienne Idioms – operators

All operator for arrays:

```
.all. ([i,j] .lessThan. k)
.all. ([i,j] .lessThan. [k,m])
.all. (i .lessThan. [k,m])
```

`Logical` operator: `.expect. allocated(A) // " (expected allocated A)"`

String concatenation operator: `//`

# Julienne Idioms – operators

Equality within a tolerance operators:

```
x .approximates. y .within. tolerance

x .approximates. y .withinFraction. tolerance

x .approximates. y .withinPercentage. tolerance
```

# Julienne Idioms

```
test_diagnosis = test_diagnosis .also. &
     .all. (actual_coarse .approximates. expected_coarse &
           .within. crude_tolerance) &
     // " (coarse-grid 2nd-order .laplacian. sin(x))"
```

Test reads like a specification, similar to a natural language sentence

# Julienne Assertions

- Provides assertions: `call_julienne_assert(expr)`
  - Can be compiled away
  - Activated using `-DASSERTIONS` preprocessing compiler flag
  - When active and assertion fails, program will `ERROR STOP` with useful message
- Supports asserting the preconditions, postconditions, invariants
- Frequently used in source code
- Also can be used in unit tests

# Use Case: Fiats

# Fiats: Deep Learning with Fortran

- Machine learning and AI impacts on HPC
- Fiats (**Functional inference and training for surrogates**)
  - go.lbl.gov/fiats
  - Alternative name:

    **Fortran inference and training for science**
  - “training and deployment of neural-network surrogate models for computational science”
  - Automatic parallelization of batch inference



Photo by Max Di Capua on Unsplash

# Fiats: Deep Learning with Fortran

- Assertions used for:
  - Validating input data is in correct format
    - Multiple ways to input data to construct neural nets
    - Need to validate the various ways
  - Validate the construction of the layers of the neural network
  - And more!

Photo by Max Di Capua on Unsplash


Photo by Marcos Llerena on Unsplash

## Fiats needs to ensure that:

An object of type `neural_network_t` that encodes an XOR gate can correctly perform elemental inference with **step** activation and 1 hidden layer


Photo by Marcos Llerena on Unsplash

## Fiats needs to ensure that:

An object of type `neural_network_t` that encodes an XOR gate can correctly perform elemental inference with **relu** activation and 1 hidden layer


Photo by Marcos Llerena on Unsplash

## Fiats needs to ensure that:

An object of type `neural_network_t` that encodes an XOR gate can correctly perform elemental inference with 2 hidden layers


Photo by Marcos Llerena on Unsplash

# Fiats uses Julienne unit tests to help validate all these behaviors and more

# One unit test for Fiats
## Part of `neural_network_t` unit test collection

```
212  function elemental_infer_1_hidden_layer_xor_step() result(test_diagnosis)
213    type(test_diagnosis_t) test_diagnosis
214    type(neural_network_t) neural_network
215
216    neural_network = one_hidden_layer_xor_net("step")
217
218    block
219      type(tensor_t), allocatable :: truth_table(:)
220      real, parameter :: tolerance = 1.E-08, false = 0., true = 1.
221      integer i
222
223      associate(array_of_inputs => [tensor_t([true,true]), tensor_t([true,false]), &
224                                    tensor_t([false,true]), tensor_t([false,false])])
225        truth_table = neural_network%infer(array_of_inputs)
226      end associate
227      test_diagnosis =  &
228               (.all. (truth_table(1)%values() .approximates. (false) .within. tolerance)) &
229        .also. (.all. (truth_table(2)%values() .approximates. ( true) .within. tolerance)) &
230        .also. (.all. (truth_table(3)%values() .approximates. ( true) .within. tolerance)) &
231        .also. (.all. (truth_table(4)%values() .approximates. (false) .within. tolerance))
232    end block
233  end function
```

# One unit test for Fiats
## Part of `neural_network_t` unit test collection

## Result of unit test

```
227       test_diagnosis =  &
228               (.all. (truth_table(1)%values() .approximates. (false) .within. tolerance)) &
229         .also. (.all. (truth_table(2)%values() .approximates. ( true) .within. tolerance)) &
230         .also. (.all. (truth_table(3)%values() .approximates. ( true) .within. tolerance)) &
231         .also. (.all. (truth_table(4)%values() .approximates. (false) .within. tolerance))
```

# One unit test collection for Fiats
# Tests `neural_network_t`

```
36  pure function subject() result(specimen)
37    character(len=:), allocatable :: specimen
38    specimen = "An neural_network_t that encodes an XOR gate"
39  end function
40
41  function results() result(test_results)
42    type(test_result_t), allocatable :: test_results(:)
43    type(neural_network_test_t) neural_network_test
44
45    test_results = neural_network_test%run([ &
46       test_description_t("elemental inference with step activation and 1 hidden layer", elemental_infer_1_hidden_layer_xor_step) &
47      ,test_description_t("elemental inference with relu activation and 1 hidden layer", elemental_infer_1_hidden_layer_xor_relu) &
48      ,test_description_t("elemental inference with 2 hidden layers", elemental_infer_with_2_hidden_layer_xor_net) &
49      ,test_description_t("converting a network with 2 hidden layers to and from JSON format", multi_hidden_layer_net_to_from_json) &
50      ,test_description_t("converting a network with varying-width hidden layers to/from JSON", varying_width_net_to_from_json) &
51      ,test_description_t("performing inference with a network with hidden layers of varying width", infer_with_varying_width_net) &
52      ,test_description_t("double-precision inference", double_precision_inference) &
53    ])
54  end function
```

# Output for `neural_network_t` unit test collection

```
An neural_network_t that encodes an XOR gate
   passes on elemental inference with step activation and 1 hidden layer.
   passes on elemental inference with relu activation and 1 hidden layer.
   passes on elemental inference with 2 hidden layers.
   passes on converting a network with 2 hidden layers to and from JSON format.
   passes on converting a network with varying-width hidden layers to/from JSON.
   passes on performing inference with a network with hidden layers of varying width.
   passes on double-precision inference.
 7 of 7 tests passed. 0 tests were skipped.
```

# Additional Fiats Test Output

```
A trainable_network_t object
   passes on preserving an identity map with 2 hidden layers.
   passes on learning identity with Adam from perturbed-identity initial condition.
   passes on learning a 2-hidden-layer AND gate from a skewed AND training data.
   passes on learning NOT-AND from skewed NOT-AND data.
   passes on learning OR from symmetric OR-gate data and random initial weights.
   passes on learning XOR from symmetric XOR-gate data and random initial weights.
 6 of 6 tests passed. 0 tests were skipped.

A training_configuration_t object
   passes on component-wise construction and conversion to/from JSON.
 1 of 1 tests passed. 0 tests were skipped.

A training_data_files_t object
   passes on round-trip to/from JSON yielding equivalent objects.
 1 of 1 tests passed. 0 tests were skipped.

Test-suite run time: .001 seconds
Number of images: 1

_____ 23 of 23 tests passed. 0 tests were skipped _____
```

Subject of the collection of unit tests

# Additional Fiats Test Output

```
A trainable_network_t object
   passes on preserving an identity map with 2 hidden layers.
   passes on learning identity with Adam from perturbed-identity initial condition.
   passes on learning a 2-hidden-layer AND gate from a skewed AND training data.
   passes on learning NOT-AND from skewed NOT-AND data.
   passes on learning OR from symmetric OR-gate data and random initial weights.
   passes on learning XOR from symmetric XOR-gate data and random initial weights.
 6 of 6 tests passed. 0 tests were skipped.

A training_configuration_t object
   passes on component-wise construction and conversion to/from JSON.
 1 of 1 tests passed. 0 tests were skipped.

A training_data_files_t object
   passes on round-trip to/from JSON yielding equivalent objects.
 1 of 1 tests passed. 0 tests were skipped.

Test-suite run time: .001 seconds
Number of images: 1

_____ 23 of 23 tests passed. 0 tests were skipped _____
```

single unit test

describes expected behavior

pass or fail

# Fiats Test Output (Partial - abridged for slide)

```
A trainable_network_t object
   passes on preserving an identity map with 2 hidden layers.
   passes on learning identity with Adam from perturbed-identity initial condition.
   passes on learning a 2-hidden-layer AND gate from a skewed AND training data.
   passes on learning NOT-AND from skewed NOT-AND data.
   passes on learning OR from symmetric OR-gate data and random initial weights.
   passes on learning XOR from symmetric XOR-gate data and random initial weights.
 6 of 6 tests passed. 0 tests were skipped.

A training_configuration_t object
   passes on component-wise construction and conversion to/from JSON.
 1 of 1 tests passed. 0 tests were skipped.

A training_data_files_t object
   passes on round-trip to/from JSON yielding equivalent objects.
 1 of 1 tests passed. 0 tests were skipped.

Test-suite run time: .001 seconds
Number of images: 1

_____ 23 of 23 tests passed. 0 tests were skipped _____
```

Additional information

All results

# Julienne Assertions in Fiats

- Used in both source code and tests
- In source code, used to validate JSON input data before constructing objects

```
426  module procedure default_real_from_json
427
428    character(len=:), allocatable :: justified_line
429    integer l, num_file_lines
430    type(string_t), allocatable :: lines(:)
431    type(tensor_map_t) input_map, output_map
432    type(layer_t) hidden_layers, output_layer
433
434    lines = file_%lines()
435    associate(outermost_object => '{')
436      call_julienne_assert(adjustl(lines(1)%string()) .equalsExpected. outermost_object)
437    end associate
438    call_julienne_assert(lines(2)%get_json_value("minimum_acceptable_tag", mold="") .equalsExpected. minimum_acceptable_tag)
439
440    ...
```

(Many more Julienne assertions in rest of procedure)

# Use Case: Caffeine

# Caffeine: A Parallel Runtime Library

- Caffeine
  - CoArray Fortran Framework of Efficient Interfaces to Network Environments
  - go.lbl.gov/caffeine
  - Implements Parallel Runtime Interface for Fortran (PRIF)
  - Enables multi-image execution for LLVM Flang and lfortran

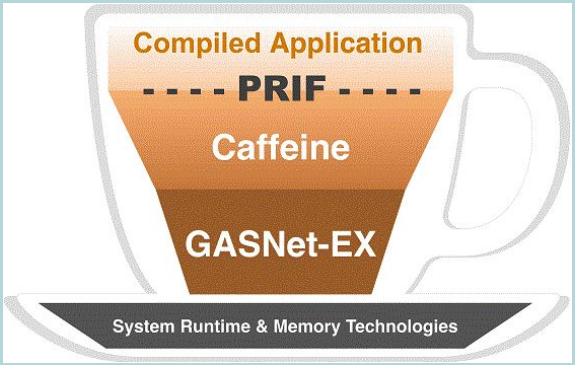





Photo by Mike Kenneally on Unsplash

Caffeine unit testing needs not based on testing the library's derived types

Uniting testing needs focus more on implementing PRIF Functions that implement multi-image Fortran standard behavior


## Caffeine needs to ensure:

Caffeine's PRIF routine provides the behavior determined by the Fortran standard equivalents

Ex: `prif_num_images` returns the number of images, just as `NUM_IMAGES()`

Photo by Mike Kenneally on Unsplash

## Caffeine needs to ensure:

Caffeine's PRIF coarray allocation routines successfully allocate symmetric and asymmetric memory

Photo by Mike Kenneally on Unsplash

## Caffeine needs to ensure:

Caffeine's PRIF remote memory access (RMA) routines successfully put and get data on other images

Photo by Mike Kenneally on Unsplash

# Caffeine uses Julienne unit tests to help validate all these behaviors and much more

```
374   function check_allocation_oom() result(diag)
375     type(test_diagnosis_t) diag
376
377     integer(c_size_t) :: size_in_bytes
378     type(c_ptr) :: allocated_memory
379     integer(c_int) :: stat
380     character(len=:), allocatable :: errmsg
381     type(prif_coarray_handle) :: coarray_handle
382
383     diag = .true.
384
385     size_in_bytes = ishft(500_c_size_t, 40) ! 500TB
386
387     call prif_allocate(size_in_bytes, allocated_memory, stat, errmsg_alloc=errmsg)
388     ALSO(stat .equalsExpected. PRIF_STAT_OUT_OF_MEMORY)
389     ALSO(allocated(errmsg))
390     if (allocated(errmsg)) then
391       ALSO(len(errmsg) > 1)
392       ALSO(index(errmsg, 'out of memory') .isAtLeast. 1)
393     end if
394     deallocate(errmsg)
395
396     call prif_allocate_coarray( &
397       [integer(c_int64_t) :: 1], [integer(c_int64_t) :: ], size_in_bytes, c_null_funptr, &
398       coarray_handle, allocated_memory, stat, errmsg_alloc=errmsg)
399     ALSO(stat .equalsExpected. PRIF_STAT_OUT_OF_MEMORY)
400     ALSO(allocated(errmsg))
401     if (allocated(errmsg)) then
402       ALSO(len(errmsg) > 1)
403       ALSO(index(errmsg, 'out of memory') .isAtLeast. 1)
404     end if
405     deallocate(errmsg)
406
407   end function
```

`ALSO(expr)` expands to (roughly):

```
diag = diag .also. expr
```

# One unit test for Caffeine

Tests catching out of memory errors during coarray allocation.

Uses **ALSO** function-like macro from Caffeine to test multiple conditions using Julienne **.also.** operator

# One unit test collection for Caffeine Tests PRIF Remote Memory Access

```
26      pure function subject()
27        character(len=:), allocatable :: subject
28        subject = "PRIF RMA"
29      end function
30
31      function results() result(test_results)
32          type(test_result_t), allocatable :: test_results(:)
33          type(prif_rma_test_t) prif_rma_test
34
35          allocate(test_results, source = prif_rma_test%run([ &
36             test_description_t("sending a value to another image", usher(check_put)) &
37            ,test_description_t("sending a value with indirect interface", usher(check_put_indirect)) &
38            ,test_description_t("getting a value from another image", usher(check_get)) &
39            ,test_description_t("getting a value with indirect interface", usher(check_get_indirect)) &
40          ]))
41      end function
```

# One unit test collection for Caffeine Tests PRIF Allocation

```
37   pure function subject()
38     character(len=:), allocatable :: subject
39     subject = "PRIF Allocation"
40   end function
41
42   function results() result(test_results)
43     type(test_result_t), allocatable :: test_results(:)
44     type(prif_allocate_test_t) prif_allocate_test
45
46     allocate(test_results, source = prif_allocate_test%run([ &
47        test_description_t("allocating, using and deallocating an integer scalar coarray with a corank of 1", &
48          usher(check_allocate_integer_scalar_coarray_with_corank1)) &
49       ,test_description_t("allocating, using and deallocating an integer array coarray with a corank of 2", &
50          usher(check_allocate_integer_array_coarray_with_corank2)) &
51       ,test_description_t("allocating, using and deallocating memory non-symmetrically", &
52          usher(check_allocate_non_symmetric)) &
53       ,test_description_t("allocating and deallocating coarrays with finalizers" &
54 #      if HAVE_FINAL_FUNC_SUPPORT
55          , usher(check_final_func) &
56 #      endif
57        ) &
58       ,test_description_t("reporting out-of-memory errors", &
59          usher(check_allocation_oom)) &
60     ]))
61   end function
```

# Use Case: Formal

# Formal

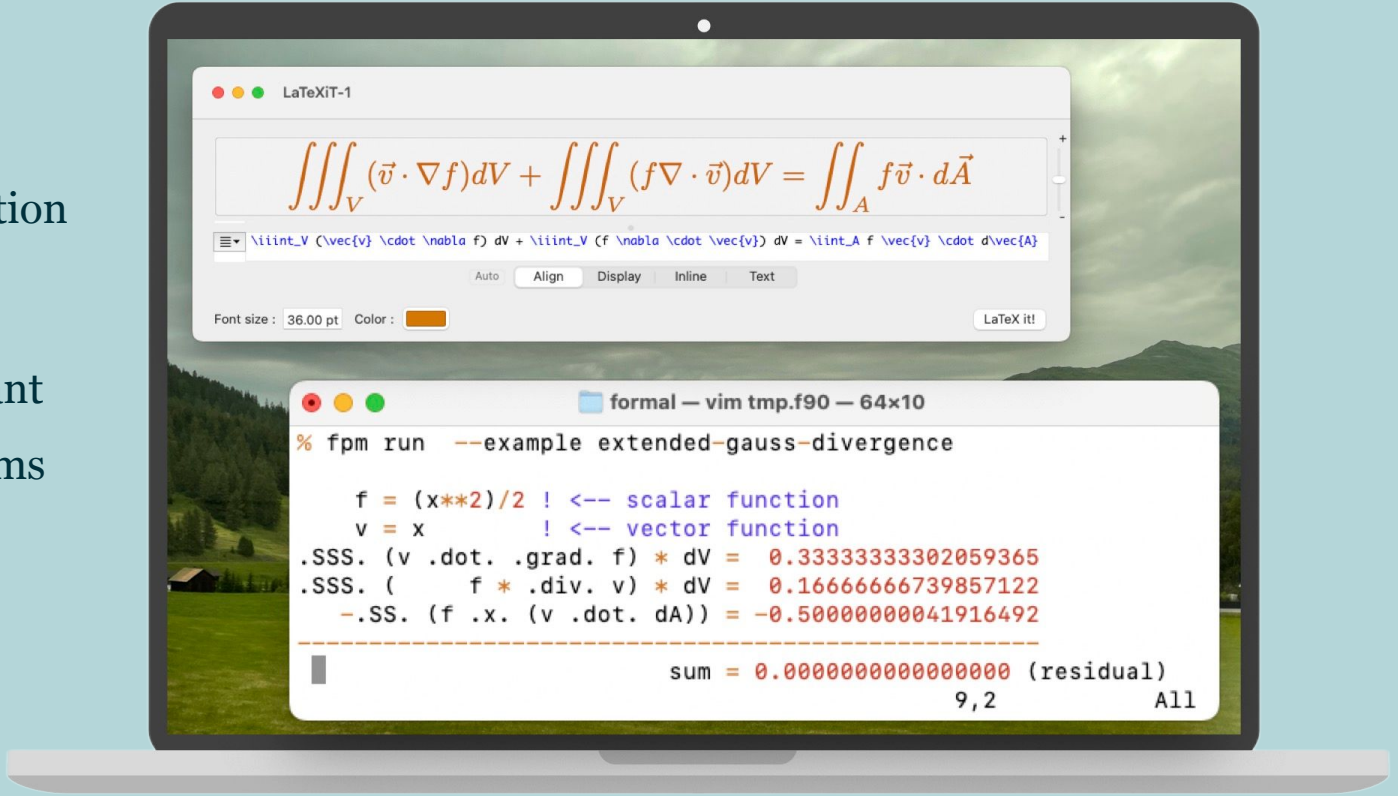
- Formal (**Fortran mimetic abstraction language**)
  - go.lbl.gov/formal
- Derived types and mathematical operators that "mimic" vector calculus
- Supports the formal verification of numerical algorithms
- Foundation for embedded domain-specific language (DSL)



Photo by Irena Carpaccio on Unsplash

# Formal

- Facilitates a degree of expressiveness that mirrors textbook notation
- Guarantees the satisfaction of important vector calculus theorems



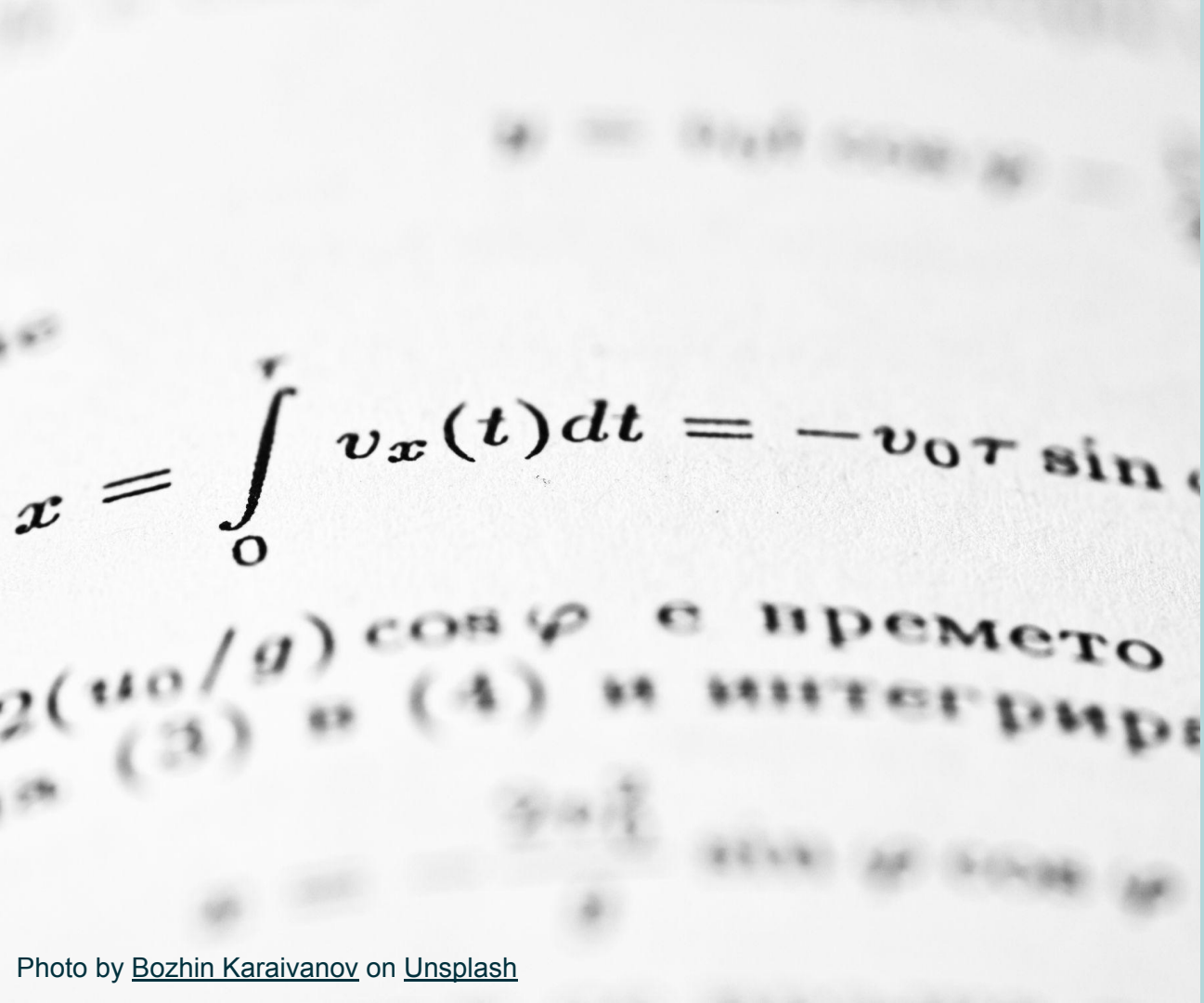


Photo by Bozhin Karaivanov on Unsplash

## Formal Unit Tests Need to:

Verify the accuracy of the discrete calculus defined by
Castillo & Corbino (2020)
Dumett & Castillo (2022)

Exercise Formal's defined operations:

`.sss.` triple integral
`.dot.` dot product

```
85  function check_volume_integral_of_v_dot_grad_f() result(test_diagnosis)
86    type(test_diagnosis_t) test_diagnosis
87    procedure(scalar_1D_initializer_i), pointer :: scalar_1D_initializer
88    procedure(vector_1D_initializer_i), pointer :: vector_1D_initializer
89    double precision, parameter :: x_min = 0D0, x_max = 1D0
90    integer, parameter :: cells = 500, cells_ = cells + 1
91    double precision, parameter, dimension(*) :: expected  = [0, 2, 0, 1]
92    double precision, parameter, dimension(*) :: order_tolerance = [0, 1, 0, 1]
93    double precision, parameter, dimension(*) :: solution_tolerance = [0D0, 5D-7, 0D0, 5D-10]
94    integer order
95
96    scalar_1D_initializer => parabola
97    vector_1D_initializer => line
98
99    test_diagnosis = passing_test()
100
101   do order = 2, 4, 2
102     associate( &
103        f  => scalar_1D_t(scalar_1D_initializer, order, cells , x_min, x_max) &
104       ,f_ => scalar_1D_t(scalar_1D_initializer, order, cells_, x_min, x_max) &
105       ,v  => vector_1D_t(vector_1D_initializer, order, cells , x_min, x_max) &
106       ,v_ => vector_1D_t(vector_1D_initializer, order, cells_, x_min, x_max) &
107       ,expected_integral => SSS_v_dot_grad_f(x_max) - SSS_v_dot_grad_f(x_min) &
108     )
109       associate( &
110          dV  => f%dV() &
111         ,dV_ => f_%dV() &
112       )
113         associate( &
114            lo_res => abs((.SSS. (v  .dot. .grad. f ) * dV  - expected_integral)) &
115           ,hi_res => abs((.SSS. (v_ .dot. .grad. f_) * dV_ - expected_integral)) &
116         )
117           test_diagnosis = test_diagnosis .also. &
118             (hi_res .isAtMost. solution_tolerance(order)) &
119             // " for " // ordinal(order) // "-order discretization of .SSS. (v .dot. .grad. f) * dV"
120             associate(calculated_order => log(lo_res/hi_res)/log(dble(cells_)/cells))
121               test_diagnosis = test_diagnosis .also. &
122                 (calculated_order .approximates. expected(order) .withinPercentage. order_tolerance(order)) &
123                 // " for convergence rate of " // ordinal(order) //  "-order discretization of .SSS. (v .dot. .grad. f) * dV"
124             end associate
125           end associate
126         end associate
127     end associate
128   end do
129
130 end function
```

# One unit test for Formal

Tests that Formal correctly computes the volume integral

.SSS.
(v .dot. .grad. f)*dV

# One unit test for Formal

## Result of unit test

```
121             test_diagnosis = test_diagnosis .also. &
122               (calculated_order .approximates. expected(order) .withinPercentage. order_tolerance(order)) &
123               // " for convergence rate of " // ordinal(order) //  "-order discretization of .SSS. (v .dot. .grad. f) * dV"
```

# One unit test collection for Formal

## Tests the set of 2nd- and 4th-order 1D mimetic integration operators

```
35  pure function subject() result(test_subject)
36    character(len=:), allocatable :: test_subject
37    test_subject = 'The set of 2nd- and 4th-order 1D mimetic integration operators'
38  end function
39
40  function results() result(test_results)
41    type(integration_operators_1D_test_t) integration_operators_1D_test
42    type(test_result_t), allocatable :: test_results(:)
43
44    test_results = integration_operators_1D_test%run([ &
45       test_description_t( &
46           'computing the volume integral .SSS. (v .dot. .grad. f)*dV' &
47         ,usher(check_volume_integral_of_v_dot_grad_f)) &
48      ,test_description_t( &
49           'computing the volume integral .SSS. (f * .div. v)*dV' &
50         ,usher(check_volume_integral_of_f_div_v)) &
51      ,test_description_t( &
52           'computing the surface integral .SS. (f .x. (v .dot. dA))' &
53         ,usher(check_surface_integral_of_vf)) &
54      ,test_description_t( &
55           'satisfying the extended Gauss Divergence Theorem within a residual tolerance of ' &
56           // string_t(residual_tolerance) &
57         ,usher(check_gauss_divergence_theorem)) &
58    ])
59  end function
```

# RAG and Vibe Testing

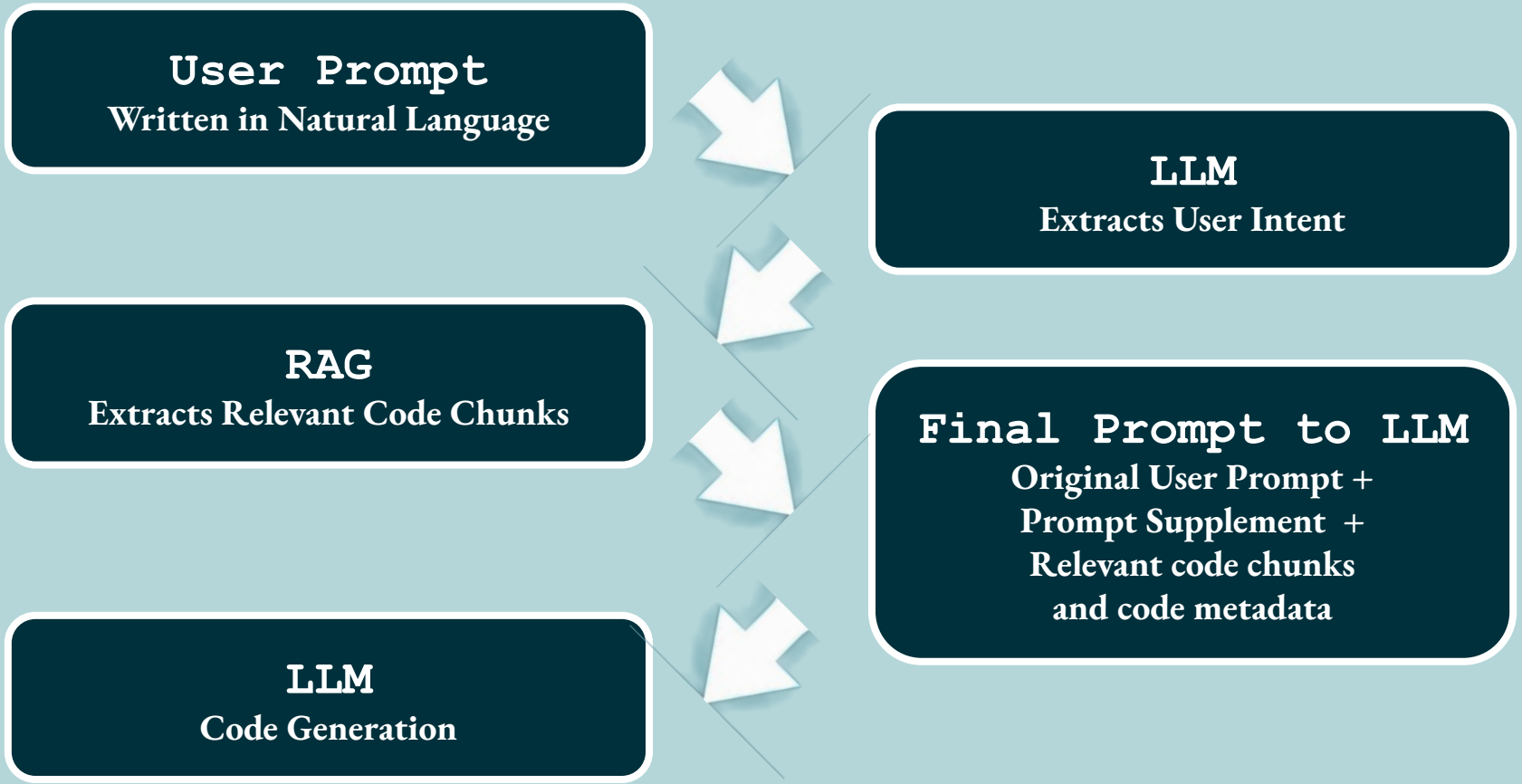
# Retrieval Augmented Generation (RAG)

- Retrieve relevant data to augment an LLM's response
- AstraAI
  - LBL's work as part of AI4HPC, a Modcon project
  - Sherry Li, Mahesh Natarajan, Damian Rouson, Kate Rasmussen
  - https://github.com/AIForHPC/AstraAI
  - Tool to support developers and users of scientific software

# How AstraAI uses RAG

- AstraAI receives repository with source code that is annotated with metadata
  - Code can be annotated with metadata manually or with an LLM (if you trust it)
- AstraAI analyzes the repository and an LLM creates embeddings based on metadata
- AstraAI analyzes the user prompt (asking for production of source code) and compares it to the embeddings to select the most relevant code chunks
- User prompt is supplemented with the most relevant code chunks, their metadata, additional useful instructions and fed to the LLM for a final response

# AstraAI Workflow



# Formal - Vibe Testing

- Formal has been used with AstraAI workflow
  - Success in prompting LLM to produce Fortran source code that correctly invokes Formal library
  - Julienne unit tests annotated with metadata
  - Used expert-written, very detailed prompt
- Future research:
  - Improve workflow to produce code with less detailed prompts
  - Explore production of source code as well as tests to verify source code


Photo by Irena Carpaccio on Unsplash

# How to Use All This for Your Project

# Live walkthrough

# (Overview of Content of live walkthrough)

- Create Julienne boilerplate using the `scaffold` program
- Show Julienne driver that runs collections of unit tests
- Walk through unit test collections and `subjects` and `results` functions
- Walk through unit test and `test_diagnosis_t` result
- How to skip tests
- How to run Julienne test suite
- How to filter tests while running test suite

# Writing Julienne Tests - new collection of unit tests

When creating a new collection of unit tests:

- Create new test object for a new collection of unit tests
  - Extends the `test_t` type from the framework
- Write a `subject` function
  - Describes the collection of unit tests
- Write a `results` functions
  - Lists and invokes each of the unit tests through the `test_description_t` objects
- Add new collection of unit tests to `test_harness_t` array in driver

# Skipping Tests

- Julienne provides functionality to skip some tests
- When to skip a test?
  - The test triggers a compile-time or runtime crash
  - Example: gfortran runtime bug in Julienne test suite for the Julienne repository
- To skip a test:
  - When constructing the list of `test_description_t` objects, omit the function name (or procedure pointer) to the `test_description_t` constructor

# Filtering Tests

- Julienne provides functionality to run only a subset of tests
  - Can run a specific collection of unit tests
  - Can run specific unit tests
  - Invoke by using `--contains` flag
    - `fpm test -- --contains string_t`
    - `fpm test -- --contains comma`

# Can You Try Vibe Testing?

# Can You Try Vibe Testing?

- Ideas around how to use unit tests with LLMs still developing but…
- Yes!
  - AstraAI publicly available (https://github.com/AIForHPC/AstraAI)
    - Research product still in development
  - You can use integrate unit tests into your usage of LLM

# Using Unit Tests As Specifications and in Prompts

- Write Julienne unit tests as specifications
- Can use unit tests to supplement LLM prompts
  - Julienne unit tests mimic natural language, normally what is involved in LLM prompts, but tests are compilable and checkable
- Consider how you can utilize tests to improve responses from LLMs
- Remember test driven development?
  - Write unit test first (human in the loop)
  - Use unit test in prompt to produce source code
  - Iterate until unit test passes and code is verified

# Communities & Where to Find More Fortran

# Grabbag of Fortran tools

- Codee - Static analysis tool for Fortran and C/C++
  - Code correctness
  - Modernization
  - Codee Youtube Channel
- fortran-linter
- rojff - Return of JSON for Fortran - Utility to support use of JSON files in Fortran source
- iso_varying_string - An implementation of the ISO_VARYING_STRING module as proposed for the ISO standard

# Grabbag of Fortran tools

- FORD - (FORtran Documentation)
  - Automatic documentation generator for Fortran projects
  - Can build documentation locally, provides html files
  - Can deploy documentation in Github Actions
  - Example Ford Documentation
- Various tools to convert fixed form code to free form code
- A grabbag of more tools - Beliavsky/Fortran-Tools

# Upcoming Fortran Features

- Upcoming features:
  - Generic programming (templates, etc)
    - Type-safe templates: requirements (strong concepts)
      - Must state properties of types
      - Compiler can provide error messages in template definition code
  - Asynchrony: tasks, collectives
  - Standardized Fortran preprocessor
- International body - WG5
- Working group - INCITS US National body (informally known as J3)

# Fortran Resources & Communities

- Stereotypes about Fortran include ideas of being antiquated
- Vibrant, engaged developer community
- For more Fortran questions or to engage and **share** with the Fortran community, visit:
  - Fortran Lang: fortran-lang.org
    - Tutorials, links to playgrounds, compilers info
    - Helpful language information and advice
    - Link to LFortran, a Fortran compiler and interpreter
  - Discourse: fortran-lang.discourse.group
  - Fortran Wiki: fortranwiki.org
- Fortran at LBNL: fortran.lbl.gov

# Acknowledgements


- This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research

- As mentioned, Julienne was initially inspired by the testing framework Veggies
  - Veggies lead developer: Brad Richardson
  - Veggies repository: https://gitlab.com/everythingfunctional/veggies

# Thank You

## Questions?

Email: fortran@lbl.gov