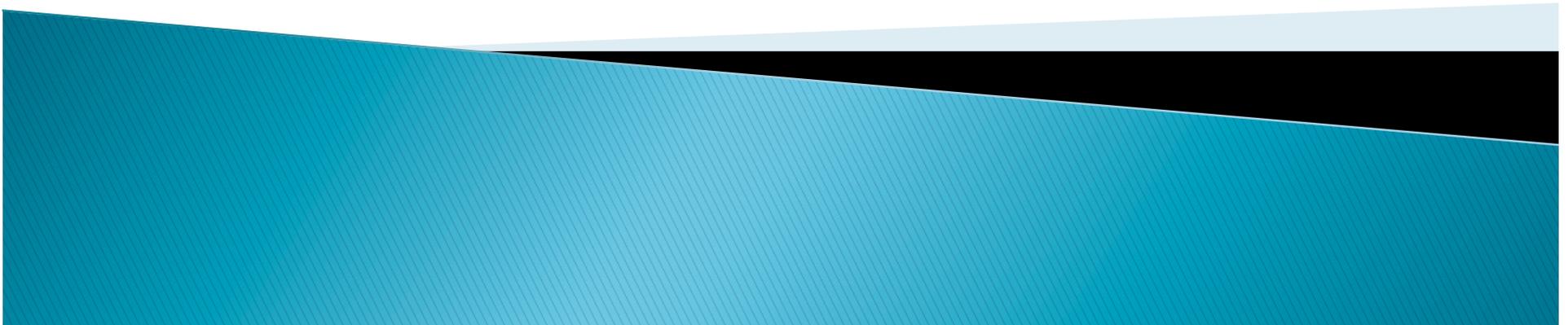


# **Workshop on Tracers (and Diagnostics) in ModelE**

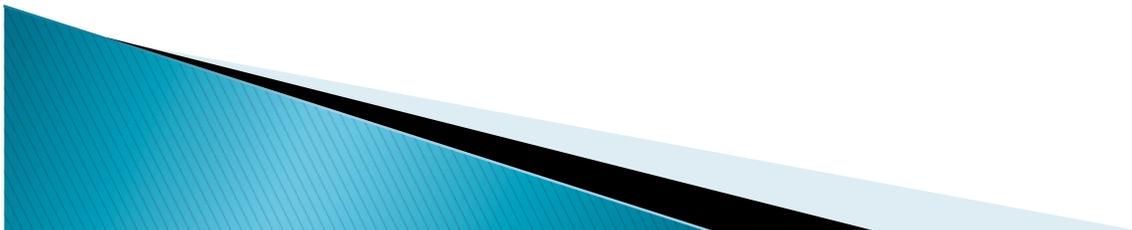
T. Clune



# Goals

---

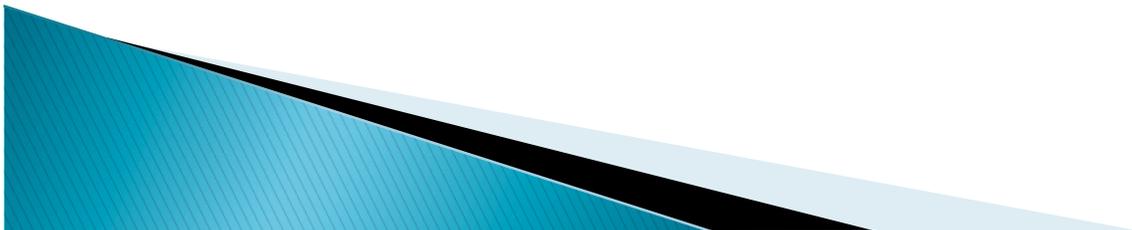
- ▶ Establish consensus on desirable modifications to implementation of tracers in modelE.
  - Defer on issues that lack strong support and/or clear implementation mechanism.
- ▶ Improve *long-term* scientific productivity.
  - Short term inconvenience as an investment in future.



# Outline

---

- ▶ Issues with existing tracer interfaces
- ▶ Straw-man proposal for new interfaces
- ▶ Tangential tracer issues – half-baked
  - Sources and diagnostics
  - Chemistry
- ▶ Actions



# Things that are difficult

---



# Basic tracer issues

---

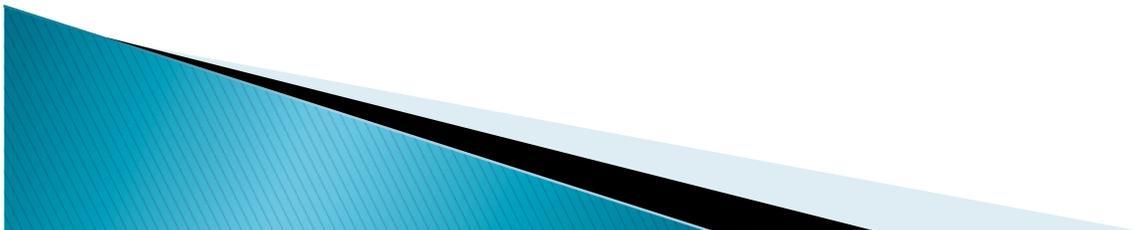
- ▶ Difficult to introduce new tracer
- ▶ Difficult to introduce new tracer *property*
- ▶ Tracer “thrashing”
  - Independent efforts “stepping on each-other’s toes
  - I.e. you are forced to “see” everyone else’s tracers
- ▶ *Static* specification
  - Must recompile to modify tracer or property
- ▶ Hard to learn/understand implementation
- ▶ Separate infrastructure for ocean tracers



# Other tracer issues

---

- ▶ Would like to be able to enable sets of tracers
  - E.g. moments
- ▶ Intrusion into physics components
  - Obscures legibility and degrades maintainability
- ▶ Inconsistent/duplicated functionality
  - Format of source files
  - Source/sink diagnostics



# Observations on implementation

---

- ▶ Fragile constructs (programming by “coincidence”)
  - Multiple changes must be coordinated for consistency
  - E.g. some source diagnostics must precisely duplicate order of sources
- ▶ Implementation is fractured across
  - multiple files/procedures
  - multiple variables/arrays
- ▶ Fine-grained redundancy.
  - E.g. lots of ‘ $k = k + 1$ ’
- ▶ Heavy reliance upon CPP
  - Obscures legibility
  - Impedes testing



# Other issues?

---



# New approach

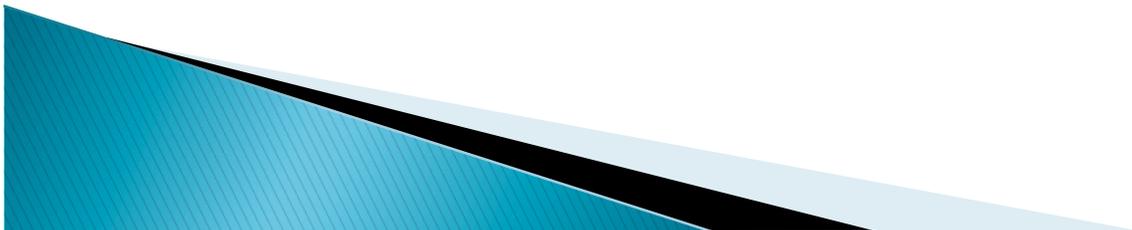
---



# Tracer specification

---

- ▶ Switch from source code to data file
  - More dynamic – change at run time
  - More flexible – choose human friendly format
  - More extensible
- ▶ Proposed format similar to rundeck parameters:
  - Sequence of “key-value” pairs
  - Need separator between tracers (duplicate keys)



# Strawman file format

---

## TracerProperties.txt

```
<header>
!END HEADER
----< separator >-----
name = Air
    molecularMass = 28.9655d0 ! in line comment

----< separator >-----
name = Rn222
    molecularMass = 222.d0
    ntm_power = -21
    radioactiveDecayRate = 2.1d-6 ! sec^-1

----< separator >-----
name = Unobtainium
    molecularMass = -100.d0
    price = 1.d+14 ! $/g
```

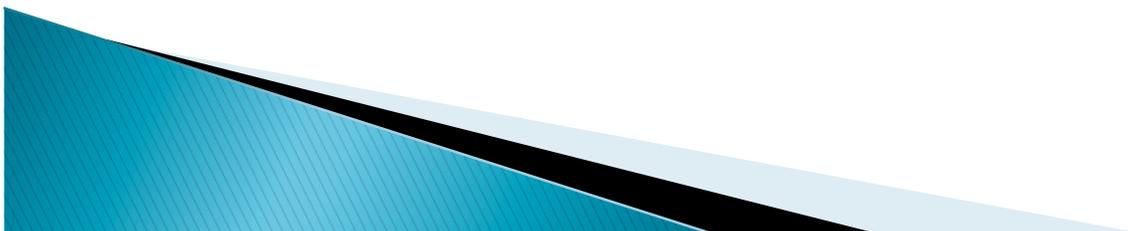
# Adding new property

---

- ▶ Now very easy – just add new “key” under relevant tracers. E.g.

```
-----  
name = H2S  
...  
  smells_like_eggs = .true.  
...
```

- ▶ Note: no need to provide values for tracers that lack a given property.
  - Obviates need for default values



# Initialization: importing tracers

---

- ▶ Desire to treat tracer property file as a database
  - Same file regardless of rundeck/experiment
  - Edit only to *extend*
- ▶ Currently model uses CPP to control tracers used
  - CPP not feasible for new approach
- ▶ Use filtering to select tracers for given rundeck:
  - Rundeck specifies tags/filters (e.g. 'dust', 'isotopes', etc)
  - At initialization only tracers with those tags will be used
  - Can alter at `_run_` time which tracers are used
- ▶ Note: CPP is still used elsewhere in model
  - True dynamic control only possible if CPP is eliminated



# Using new implementation

---

1. Tracers will be represented by an array of derived type (data structure):

```
type (Tracer_type) :: activeTracers(:)
```

2. Accessing tracers:

```
use Tracer_mod, only: activeTracers(:)
```

3. Tracer lookup:

```
idx = getIndex(tracers(:), name='SO2')
```

4. Getting property value:

```
call getValue(tracers(i), 'isotopeIndex', mass)
```

or for a set of tracers:

```
call getValue(tracers(:), 'name', names(:))
```



# Other interfaces

---

1. Check ***if*** tracer has a given property:

```
flag = hasProperty(tracers(i), `isDust`)
```

2. Subsetting tracers:

```
type (Tracer_type), pointer :: subset(:)  
subset => getWithProperty(tracers, `Lerner`)  
subset => getWithValue(tracers, `Lerner`, .true.)
```

or

```
integer, pointer :: indices(:)  
indices => getIndices(tracers, `radioactiveDecayRate`)
```

3. Counting

```
num = countWithProperty(tracers, `tr_wd_TYPE`)  
num = countWithValue(tracers, `tr_wd_TYPE`, nWater)
```



# Emergent capabilities

---

- ▶ Multiple tracer input files?
  - Each sub-discipline manages “private” set
  - Same infrastructure for ocean tracers?
- ▶ Temporarily include a variant tracer
- ▶ ...



# Issues

---

- ▶ How to verify initial conversion from source code?
    - A rundeck to run them all? (apologies to Tolkien)
    - Can we determine disjoint tracer sets a priori?
  - ▶ Do we need mandatory properties?
    - E.g. molecularMass, ntm\_power
    - Should we detect missing values for mandatory properties?
  - ▶ Do we need to allow for default values?
  - ▶ How to guard against *misspelled* properties?
    - Provide list of allowed properties
      - Rundeck? Input file? Source code?
    - Allow alternate spellings/abbreviations?
  - ▶ Will performance be adequate?
- 

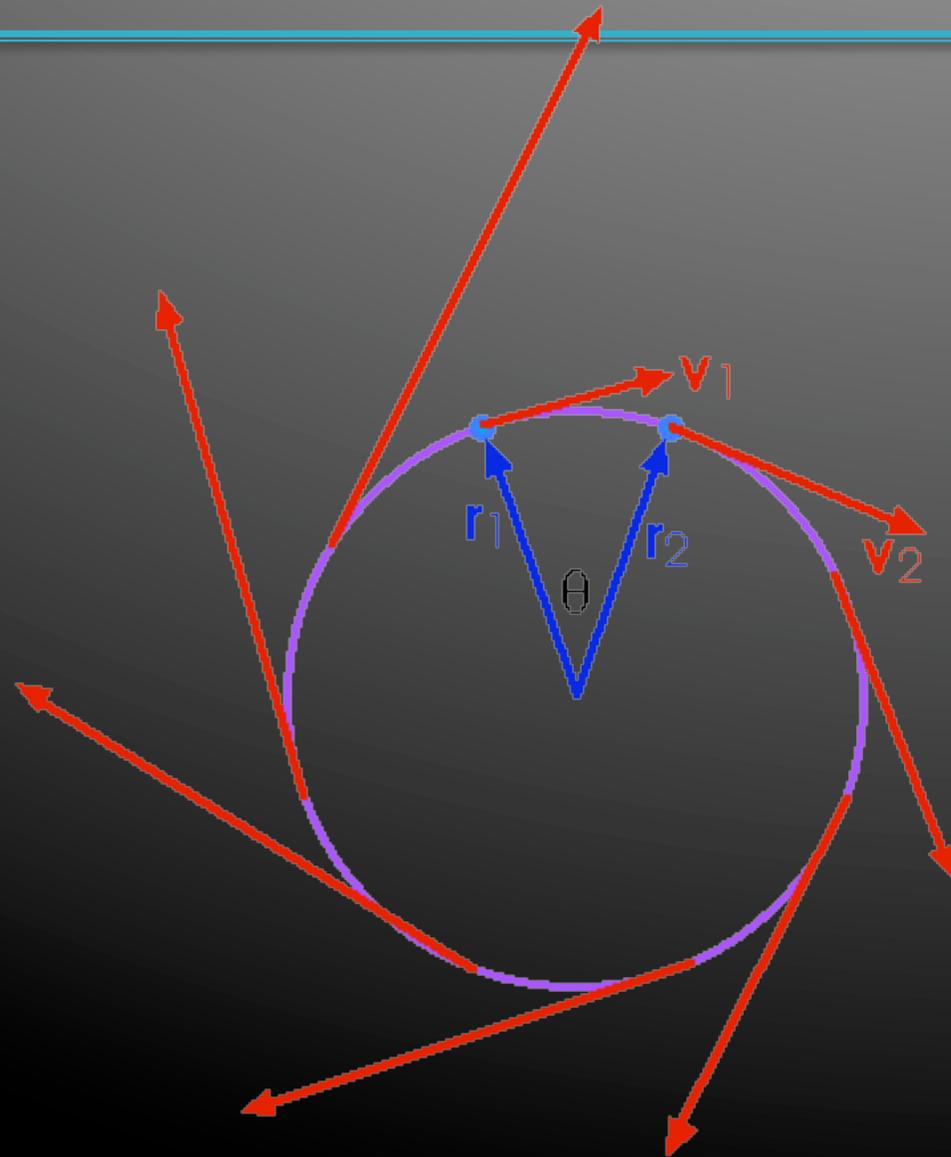
# Conventions

---

- ▶ Can we agree on a convention for file format?
  - Separator, comment characters, header, etc.
  - Do integer, real, string, and logical cover everything?
- ▶ Where should tracer property files be stored
  - Repository? (i.e. with source code)
  - Data directory?
- ▶ Opportunity to improve property names? E.g.
  - `tr_mm` => `molecularMass`
  - `tr_wd_TYPE` => `wetDepositionType`



# Tangential tracer issues



# Tracer sources

---

- ▶ Some file metadata is hardcoded in source
  - E.g. Linear ordering, connection to jls diagnostics
  - Reduces flexibility
    - E.g. difficult to add/remove source for experiment
  - Possible source of errors – requires “coincidence”
- ▶ Hardcoded data formats (multiple formats are ok)
  - Should use common interfaces
  - Minimally should contain appropriate embedded metadata
- ▶ Multiple mechanisms for accessing source files
  - Possible unification?
- ▶ Some issues similar to tracer property issues
  - Scattered infrastructure: multiple files/arrays
  - Long “SELECT CASE” block
  - No partitioning among sub-disciplines

# Sketch of alternative

---

- ▶ Introduce data structure (derived type)
  - Represents abstraction of a data source
  - Contains elements that specify any metadata not available from actual source file
  - Provide multiple initialization mechanisms
    - Different types of sources can be registered in different ways
- ▶ Set of all tracer sources would be array of structs
- ▶ Some metadata for diagnostics could be automated
- ▶ How to drive initialization?
  - Source?
  - Data file? (similar to tracer properties)



# Tracer jls diagnostics

---

- ▶ Significant source code duplication
- ▶ Frequent trivial variations: E.g.

```
case ('Rn222')  
...  
jls_decay(n) = k  
sname_jls(k) = 'Decay_of_'//trname(n)  
lname_jls(k) = 'LOSS OF RADON-222 BY DECAY'
```

```
case ('Pb210')  
...  
jls_decay(n) = k ! special array for all  
sname_jls(k) = 'Decay_of_'//trname(n)  
lname_jls(k) = 'Loss of Pb210 by decay'
```

- ▶ “per-type” defaults could significantly simplify diag. specifications. (e.g. sname, lname)
- 

# Possible diag infrastructure

---

- ▶ Specification file similar to tracer properties
  - When you have a hammer, everything becomes a nail ...

```
-----  
tracer = Rn222  
type = decay  
shortName = Decay of $  
longName  = LOSS of RADON-222 BY DECAY  
power     = -26  
units     = kg/s/mb/m^2  
weighting = 3 ! Undocumented option  
...  
-----
```

Unnecessary?  
Derived from "type"?



# Chemistry (very raw)

---

- ▶ Source duplication
- ▶ Hardwired reaction indices
- ▶ Ad hoc corrections/implementation



# Tracers and Physics

---

- ▶ Tracer logic often obscures primary physics implementation
  - Particularly severe in some components – e.g. PBL
- ▶ Can anything be done?
  - In some cases, tracer logic duplicates primary logic
    - Create subroutine for duplicated logic
    - Call once for physics, once for tracers
  - Add new “export” quantities for physics component
    - Tracer actions can then be done elsewhere



# Conclusion/Actions

---

- ▶ Implement new tracer infrastructure
  - Straightforward ~ 1 month (Tom)
- ▶ Migrate modelE to *use* new tracer infrastructure
  - Verification tests?
  - Who?
- ▶ Implement new tracer source infrastructure?
  - Set up a follow-up meeting on this?
- ▶ Further refine concepts for tangential bits



# Backup material

---

# Challenges: Adding new tracer

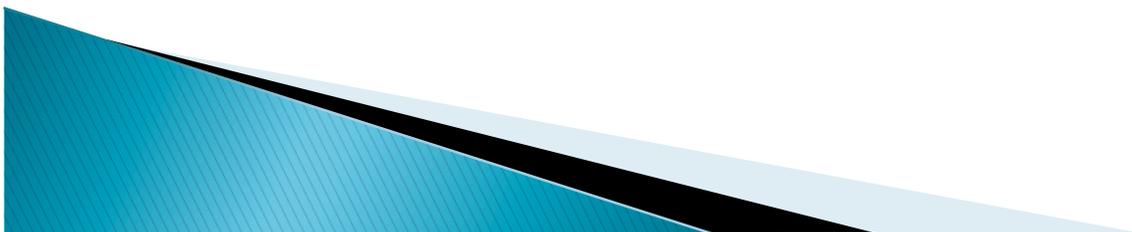
---

- Steps

1. Increment “ntm\_...” (e.g. ntm\_o18, ntm\_gasexch,...)
2. Declare new global integer index for tracer lookup (n\_air, n\_CO2n, n\_CFCn, ...)
3. Add CASE statement for new name
4. Specify any non-default values for of existing tracer properties
5. Introduce new conditional (usually CPP) to control usage per rundeck

## Potential secondary changes

1. Add related sources/diagnostics



# Challenges: Adding new *property*

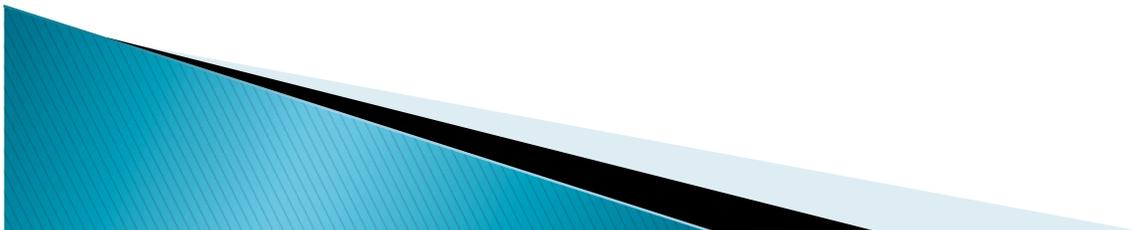
---

- Steps

1. Create new global array of size “ntm” to store values for property (usu. real\*8)
2. Provide default value before top of tracer select case
3. Override default value for all non-default cases
4. Introduce logic to use property elsewhere

- Potential secondary issues

1. Additional logic if property values depend on rundeck
2. May need counter for number of tracers with property



# Challenges: New diagnostics

---

- Multiple types, but some fairly common themes
- Less need for user-defined attributes.
  - Use static data structure
  - Might still be useful for fine-grained control
- Use a more object-based approach to structure:
  - Eliminate ad-hoc collection of global arrays
  - Co-locate diagnostic data and procedures
  - Use registration to “add” new diagnostic
  - Tracer attributes can be used to effect which diagnostics are active.
    - E.g. If a diag only applies to species A,B, and C. Diag can specify attribute to select those tracers.



# Tracer Specification

- In source code
- Each attribute stored in unique global array
- Implementation scattered in multiple files.
- “Tracers” are disconnected collection of attribute arrays and 4D state.
- Activation/deactivation of individual tracers with compile time CPP tokens
- No procedure to “find” tracer “ABC”

Current

- Input file/files
- Attribute stored in “hash” (key-value pairs)
- Implementation in single source file.
- “Tracers” are 1D list of tracer objects and 4D state.\*
- Activation/deactivation of tracers specified with run-time parameter which queries relevant attribute
- Will provide lookup procedure to return index of tracer “ABC”

Proposed

# Tracer Use

- ▶ Loop over tracers. Apply operation if given attribute is nonzero (or non default).

Current

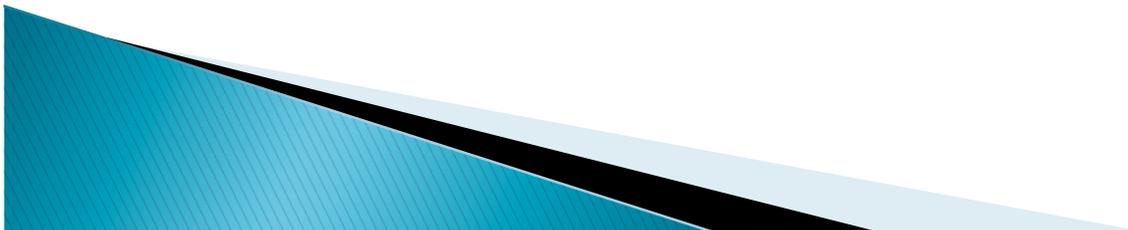
- ▶ Loop over tracers. Apply operation if tracer has given attribute.
  - If (hasAttribute(n, 'decayRate') ...
- ▶ Look-up value for attribute.
  - `decayRate = get(n, 'decayRate')`

Proposed

# Implementation Issues

---

- May be useful to have registry of allowed attributes *in the source code*.
  - Guards against typos in specification input.
  - Requires recompilation for new attributes
- File format should be similar to rundeck and ESMF config files:
  - <attribute> = <value>
- Might make sense to have separate tracer spec files for subsets of tracers
  - Only if coherent non-overlapping sets can be agreed upon
- Can have “duplicate” tracers if researcher wants to customize without impacting other results.
- Need procedure to count number of tracers with given attribute  
n = countAttribute(tracers, attribute)
- Intermediate time frame must coordinate CPP tokens with runtime attributes:
  - ```
#ifdef TOKEN_X  
logical :: runtime_X = .true.  
#else  
logical :: runtime_X = .false.  
#endif
```
- Long time frame – reduce reliance upon CPP tokens.
  - runtime\_X becomes an input parameter.
  - Can proceed one token at a time



# Implementation Issues (cont'd)

---

- Less urgent how the 4D array of tracer values is implemented:
  - Side-stepping overall modelE “registry” of arrays for now.
  - Relatively less complex than the meta-data
  - Must allocate *after* all tracer specs have been entered.
  - Could have 3D pointers in each tracer object
- ESMF?
  - Provides hash via ESMF\_Attributes.
  - Probably cumbersome as an interface given heavy use in modelE
  - Could be the back-end implementation though.

