# pFlogger: The Parallel Fortran Logging Utility

Tom Clune[1] and Carlos Cruz[1,2]

[1]NASA Goddard Space Flight Center

[2]SSAI, Inc.

CoDeSE17: Denver, CO

# POSITION

Use of text-base messages in HPC applications is typically undisciplined, leading to a chaotic hodgepodge that is of limited value to developers and users. Logging frameworks can bring order to the chaos and significantly improve our ability to extract useful information.

Typical problems:

- Important messages obscured by fountain of routine messages
- Performance
  - User adds a print statement in an inner loop or across all processes
- Anonymity – important message of unknown origin
  - Which process
  - Which software component
- Loss of productivity
  - Recompile to activate low-level debug diagnostics

# If only we could …

- Route warnings and errors to prominent location
  - And profiler data and …
- Suppress low severity (”debugging”) messages
- Suppress duplicate messages on sibling processes
- Annotate messages with:
  - Time stamp
  - Process rank
  - Software component
  - Other application-specific metadata
  - …

**And … do all of this dynamically at run time (without recompilation)**

# The Python Logging Framework

The main classes:

- **LogRecord** – encapsulates a message and its context
  - Severity is determined statically (in source code)
- **Handler** – represents different *audiences* for messages
  - Generalization of a file: could be console, email, SMS, …
  - Has a run-time severit level threshold
- **Logger** – represents different *creators* of messages
  - Typically one per software component/library
  - Each has a run-time severity threshold
  - Has a list of associated Handler objects
    - Also routes messages through ancestor Loggers' handlers.

Other important classes:

- **LoggerManager** – container of Logger objects
- **Formatter** – used by Handler objects to annotate messages (uses dictionary)
- **Filter** – Selectively suppress messages in Loggers and Handlers

**Severity Levels**

| | |
|---|---|
| 10 | DEBUG |
| 20 | INFO |
| 30 | WARNING |
| 40 | ERROR |
| 50 | CRITICAL |
| 0 | NOTSET* |

# Fortran Translation of Python Logger

Wanted something like this for a long time …

Enabled by two technologies:

- Arrival of robust object-oriented capabilities in Fortran compilers
  - But still have several compiler-specific workarounds …
- Internally developed FTL (poor-man analog of C++ STL)
  - Substantially reduces effort to define/use vectors and dictionaries from Fortran
  - In process of being released as open source (more on this later)

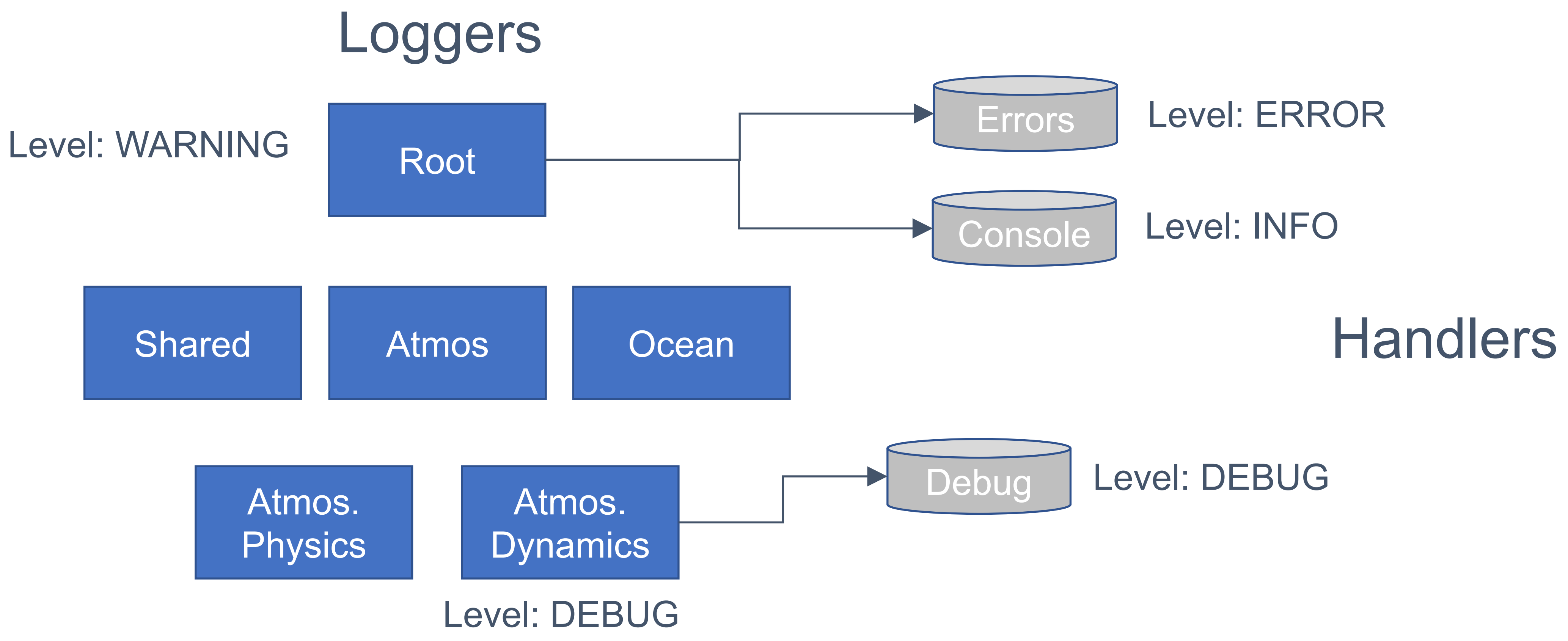
Alternative approach: Provide Fortran wrappers to Python logger.

# Configuring Logger (via YAML)

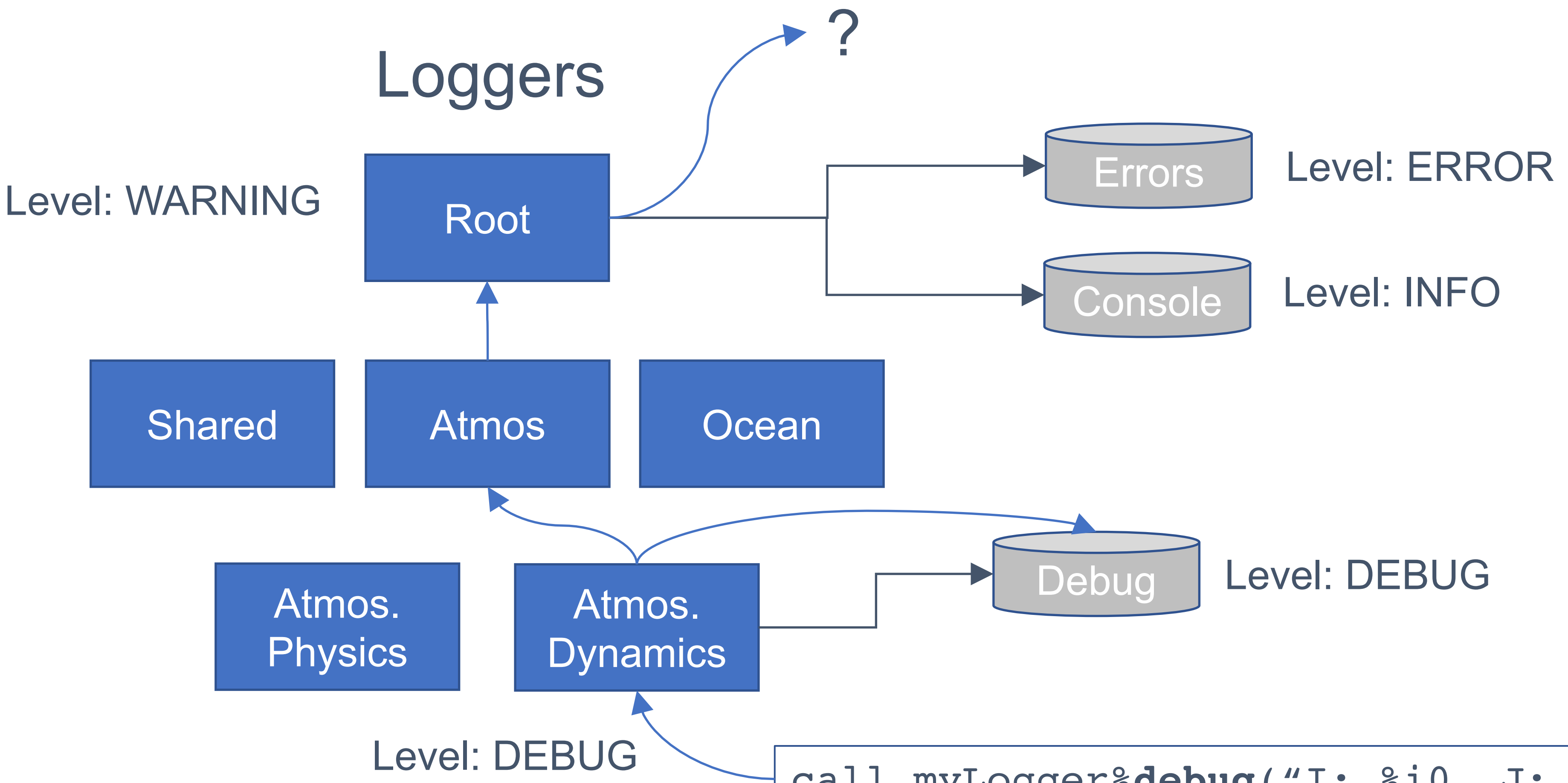
```
formatters:
  basic:
    class: Formatter
    format:'%(name)a~: %(levelName)a~: %(message)a'
  column:
    class: Formatter
    format: '(%(i)i3.3,%(j)i3.3): %(levelName)’

handlers:
  console:
    class: streamhandler
    formatter: basic
    unit: OUTPUT_UNIT
    level: WARNING
  warnings:
    class:  FileHandler
    filename: warnings.log
    level: WARNING
    formatter: basic
```

```
…

root:
  handlers: [console,warnings]

loggers:
  main:
    level: INFO
  main.A:
    level: WARNING
  main.B:
    level: INFO
```

# Life of a Message

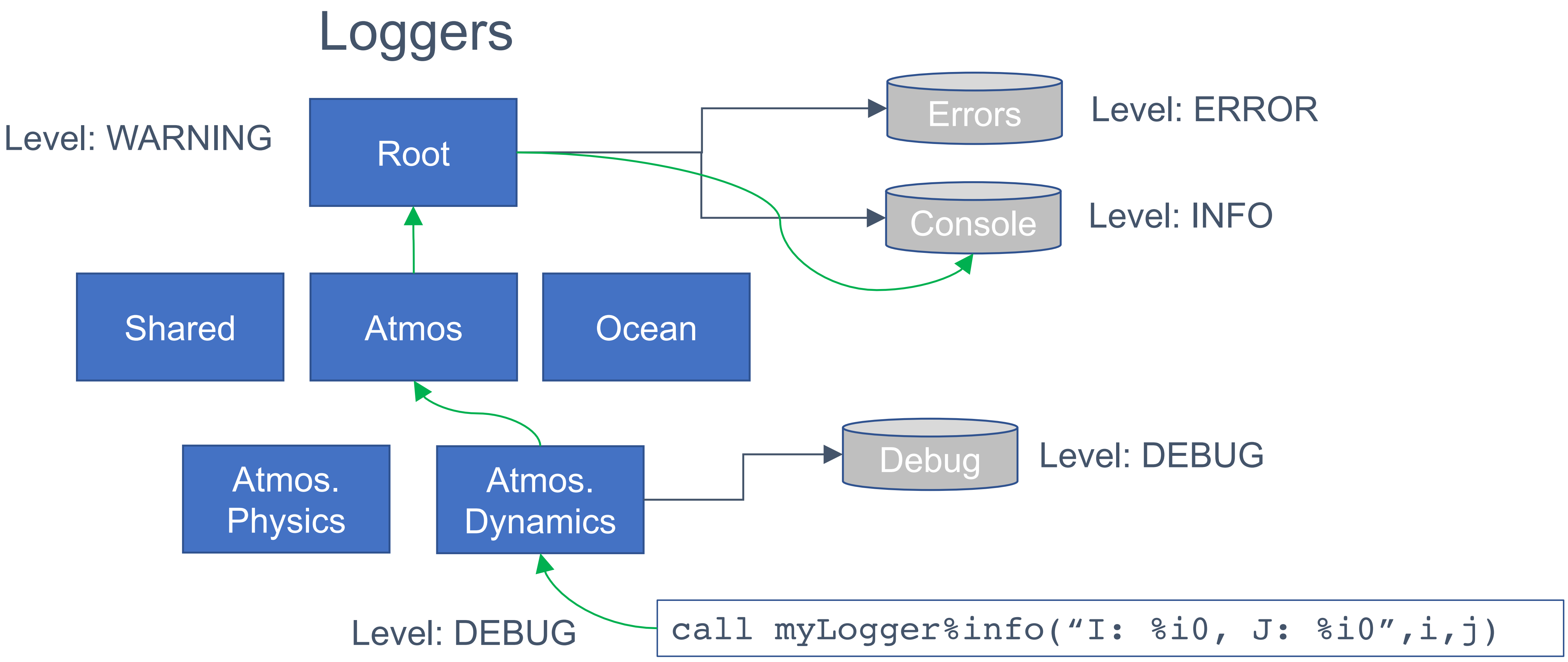


# Life of a Message

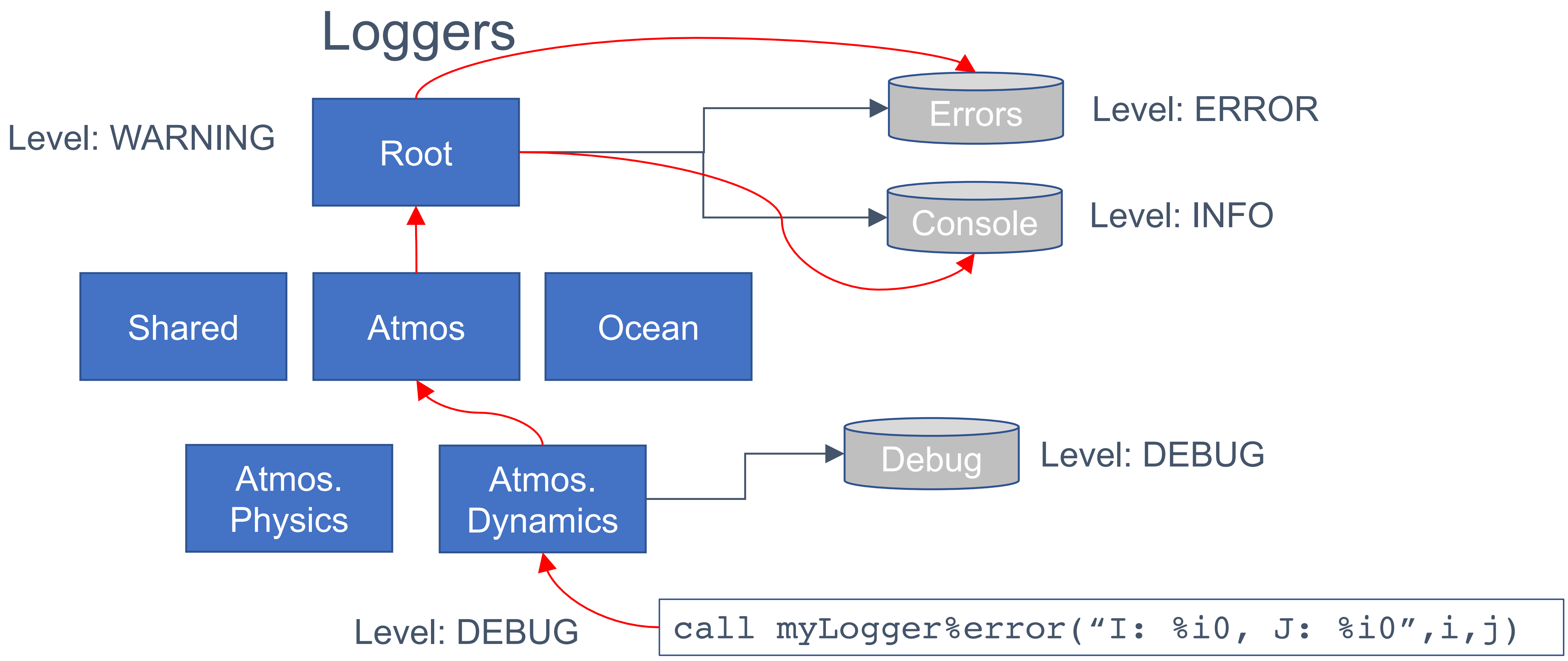



```
call myLogger%debug("I: %i0, J: %i0",i,j)
```

# Life of a Message



# Life of a Message



# Extensions for MPI Use

- LoggerManager – configured with global comm (defaults to MPI_COMM_WORLD)
- Logger – can be associated with a communicator (defaults to global)
  - root_level:  independent threshold for root process
- Handler
  - Lock – used to allow multiple processes to share access to a file
    - MpiLock – uses one-sided MPI communication
    - FileSystemLock – limited portability, but allows multi-executable sharing
  - MpiFilter – used to restrict which processes' messages are reported
  - MpiFileHandler subclass
    - Messages from each process are routed to separate file
- MpiFormatter subclass:  knows about rank and #PE's for annotations

# Advanced Capabilities

I.e., Things that are harder to use ….

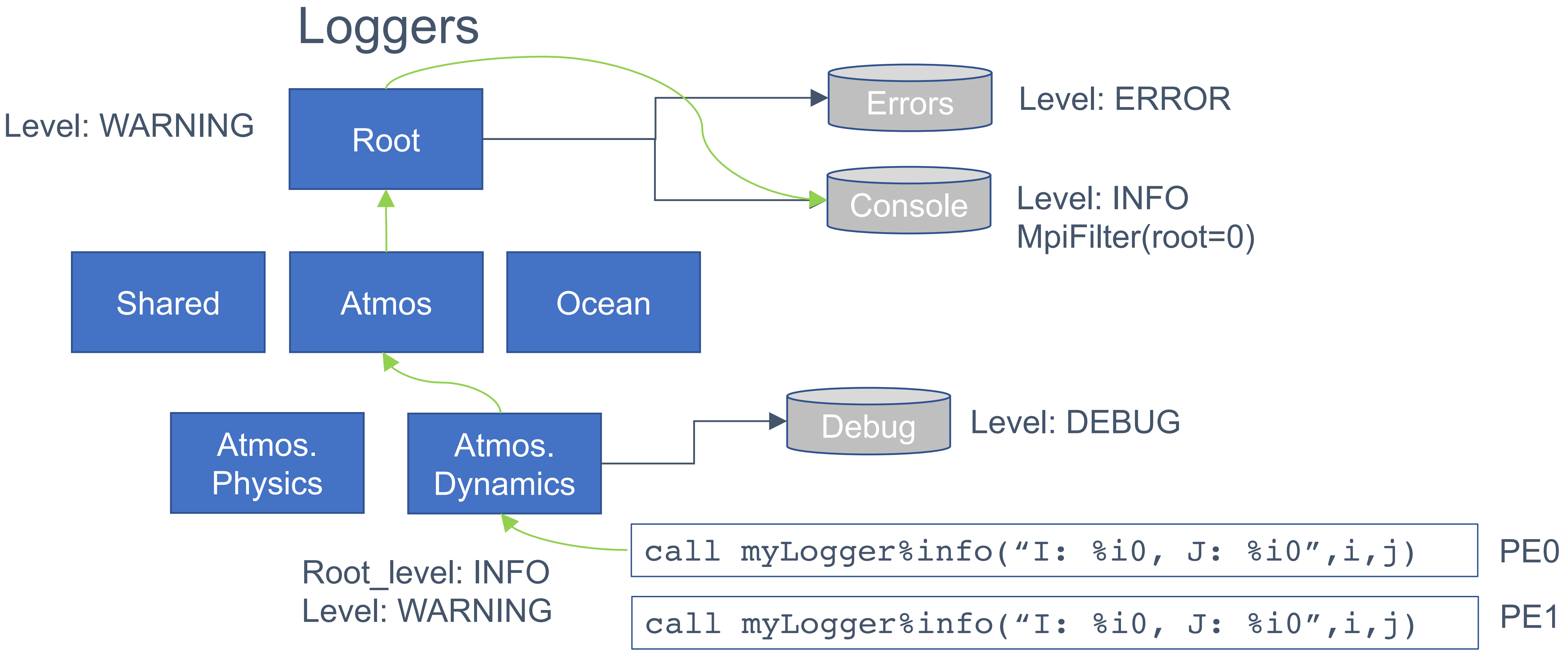
**Subcommunicators:** How to specify in run-time configuration file?

1. Construct communicators prior to initializing framework
2. Build dictionary of named communicators
3. Pass as optional argument to framework configuration step

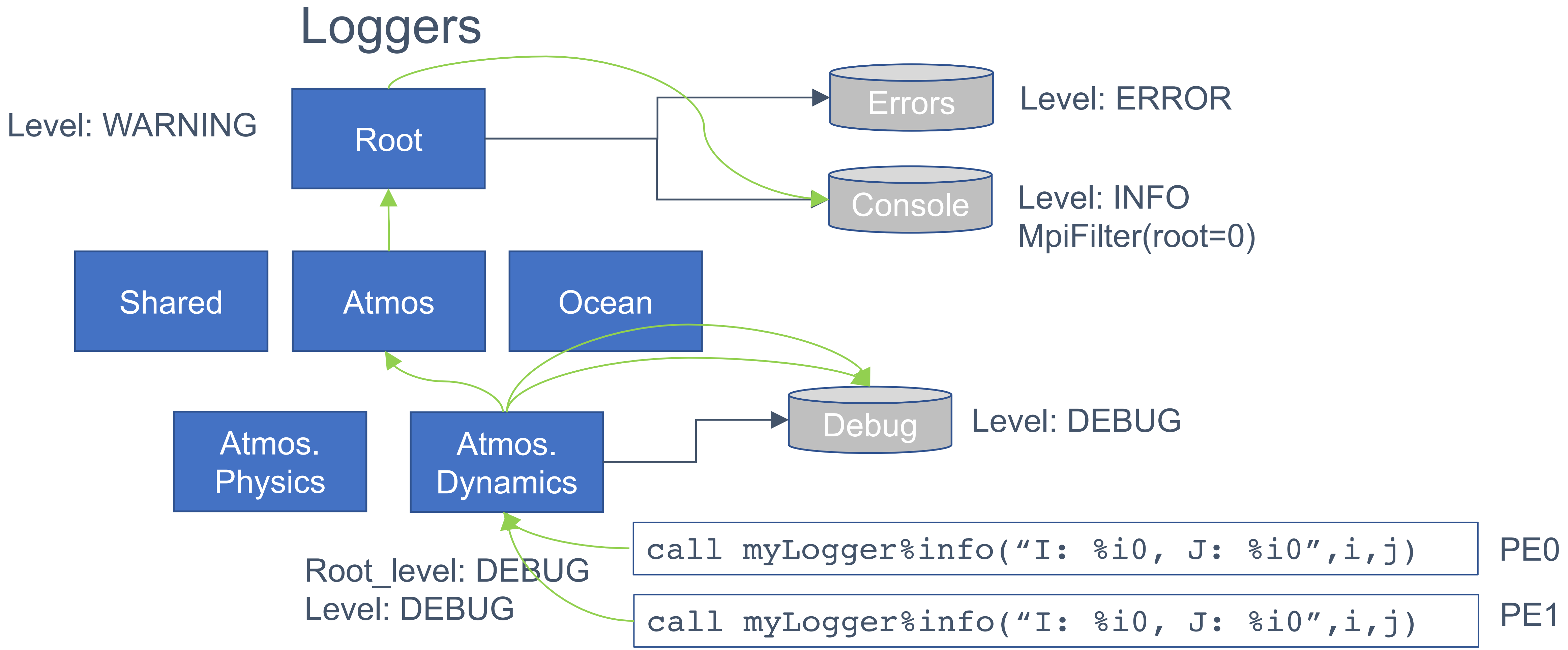
**Simulation time:** Enable annotation of messages with model's internal representation of time/phase information

1. Create a custom procedure that accesses model internal state and returns a dictionary of time-related fields. E.g. {'year':2000, 'month':'May', 'phase':'quality control'}
2. Set logger global procedure pointer "get_sim_time()" to custom procedure.

# Life of a Message: MPI



# Life of a Message: MPI



# Instrumenting with pFlogger

Easy and straightforward:

- ❖ Initialization – near beginning of application
- ❖ Declare logger in each component
- ❖ Replace print/write statements

```
use pFlogger
…
call initialize(MPI_COMM_WORLD)
Call logging%load_file(‘my_config.yaml’)
```

```
use pFlogger
…
class (Logger), pointer :: myLogger
…
myLogger => logging%get_logger(‘full.name’)
```

```
if (am_i_root()) write(*,*)’mass: ’,m
```

```
call my_Logger%info(‘mass: %*’,m)
```

# Benchmarks

Synthetic use case performance ratios

| Use Case | Intel-17 | GCC 7.1 | NAG 6.1 |
|---|---|---|---|
| Simple text message | 5.5x | 10x | 15x |
| Message with scalar items | 8x | 16x | 5x |
| Suppressed message | 0.004x | 0.03x | 0.15x |
| Split parallel log message | 1.1x | 7x | - |
| Shared parallel log message | 5x | 8x | - |

Build times of GEOS GCM (Large Earth system model)

| Compiler | Optimization | T baseline | T pflogger |
|---|---|---|---|
| Intel-17 | O0 | 218.6 | 250.3 |
| Intel-17 | O3 | 735 | 797. |
| Gfortran-7 | O0 | 178.3 | 198.3 |

# Open Source?

In progress, but the gears at NASA turn slowly …

- Can arrange for project license for groups that have NASA or other US gov't affiliation.
- Will otherwise collect email addresses from interested parties for when open source is achieved.
- If interested, send me email Tom.Clune@nasa.gov

# Summary

- pFlogger appears ready for beta testing in HPC applications
  - Further optimizations needed for intensive use cases
  - Some tweaks to initialization interfaces are expected
- Plan to integrate pFlogger into dev branch of GEOS in the near future
  - Full instrumentation will proceed on a longer time scale

- Largest problem – too much flexibility!

# Thanks to …

NASA's Modeling and Prediction Program for funding this work.

# References

V. Sajip, “logging -- logging facility for Python”,
https://docs.python.org/2/library/logging.html

# In defense of the PRINT statement …

PRINT is very versatile:

- Arbitrary number *and* type of items
- Convenient default (*) formatting
- Flexible edit descriptors that allow for precision formatting

But … most of this versatility is frozen at <u>compile time</u>.

OTOH, very difficult to emulate the versatility in a procedure interface.