# Assessing Representativeness of Kernels using Descriptive Statistics

**Youngsung Kim and John M. Dennis**

National Center for Atmospheric Research

**WRAp 2017**
September 5, 2017

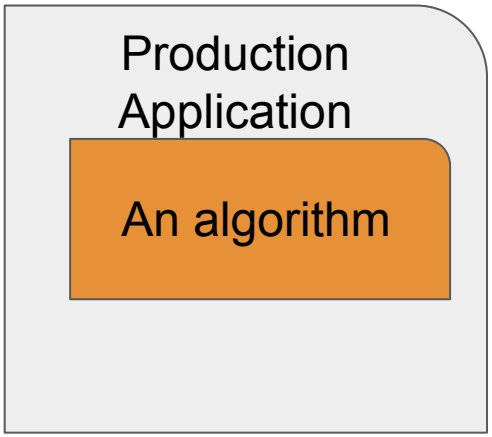

# Contents



- KGen: an automated kernel generation tool
- KGen representativeness extension
- Case studies
  - Two cases from a large Fortran applications
  - One case from using a LAPACK routine

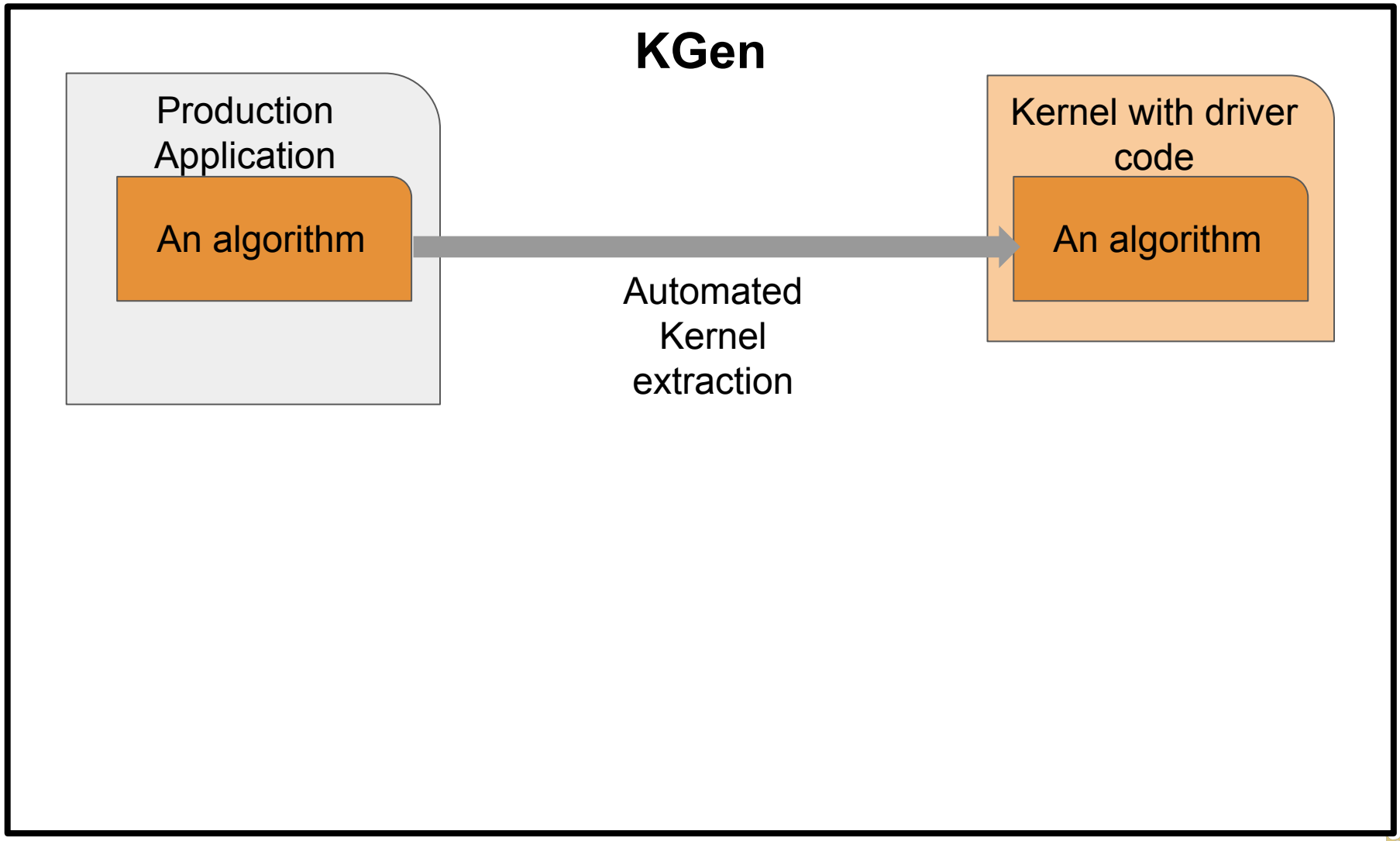
# Overview





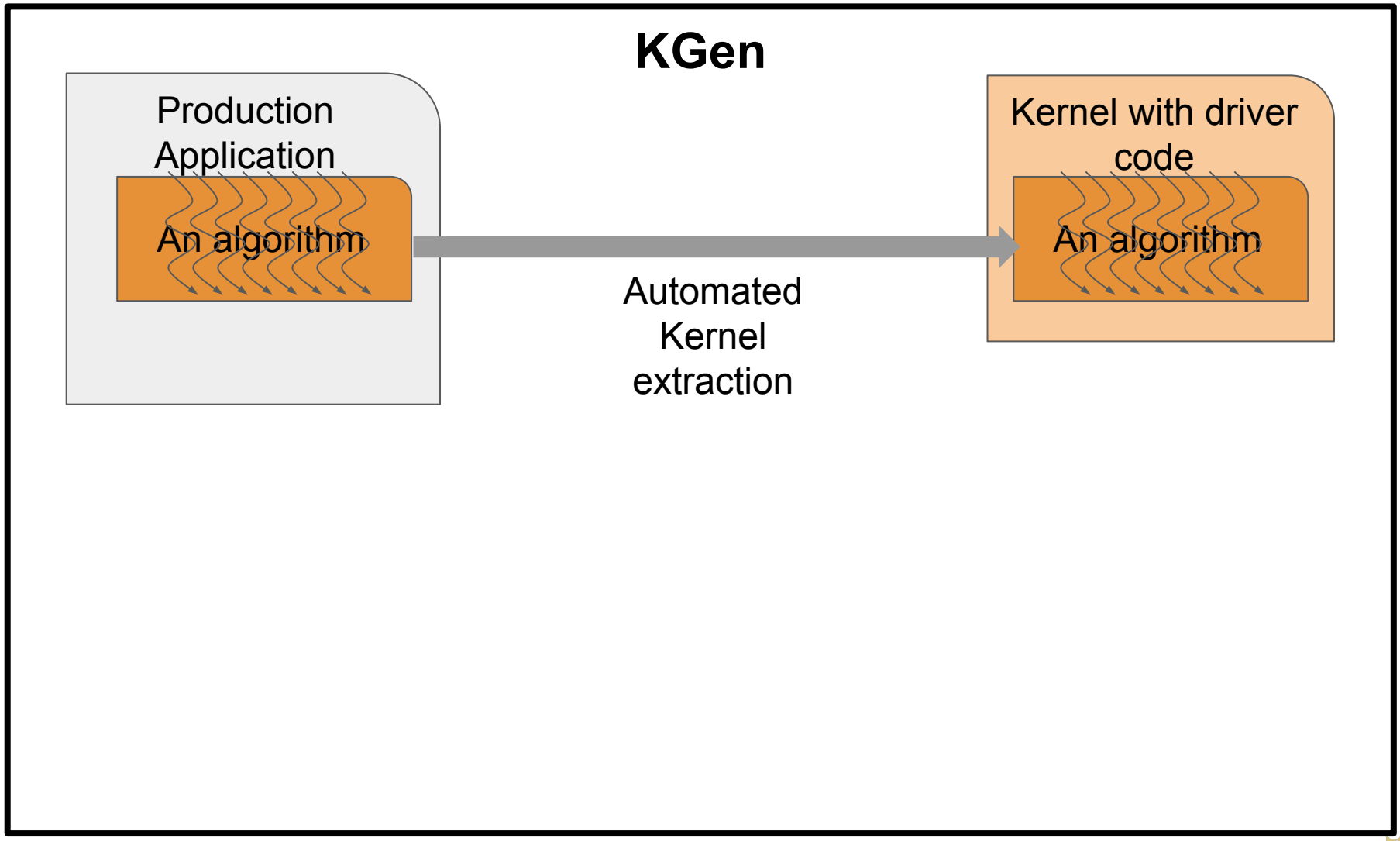
# Overview




KGen
Production Application
An algorithm
Automated Kernel extraction
Kernel with driver code
An algorithm

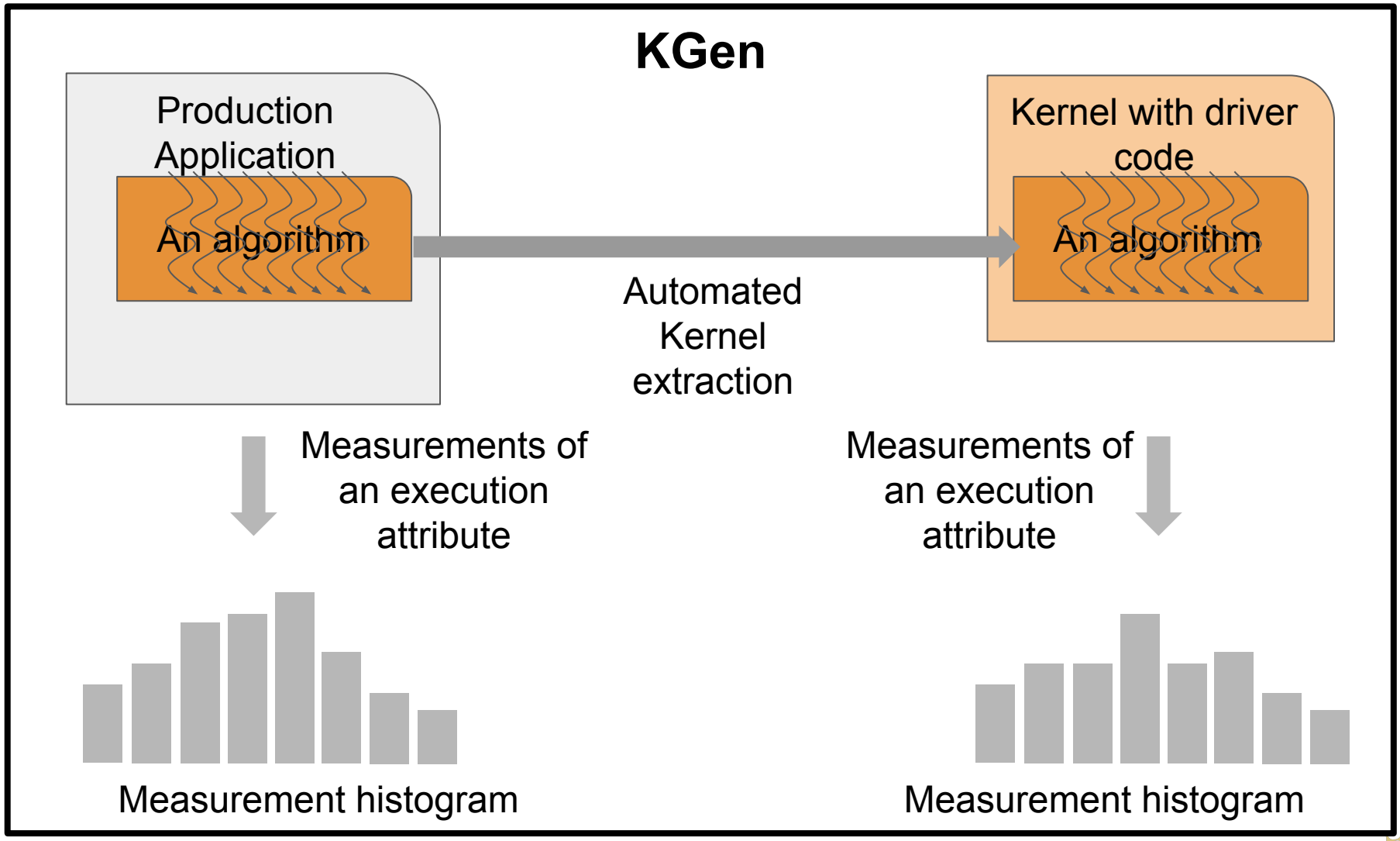
# Overview




KGen
Production Application
An algorithm
Automated Kernel extraction
Kernel with driver code
An algorithm

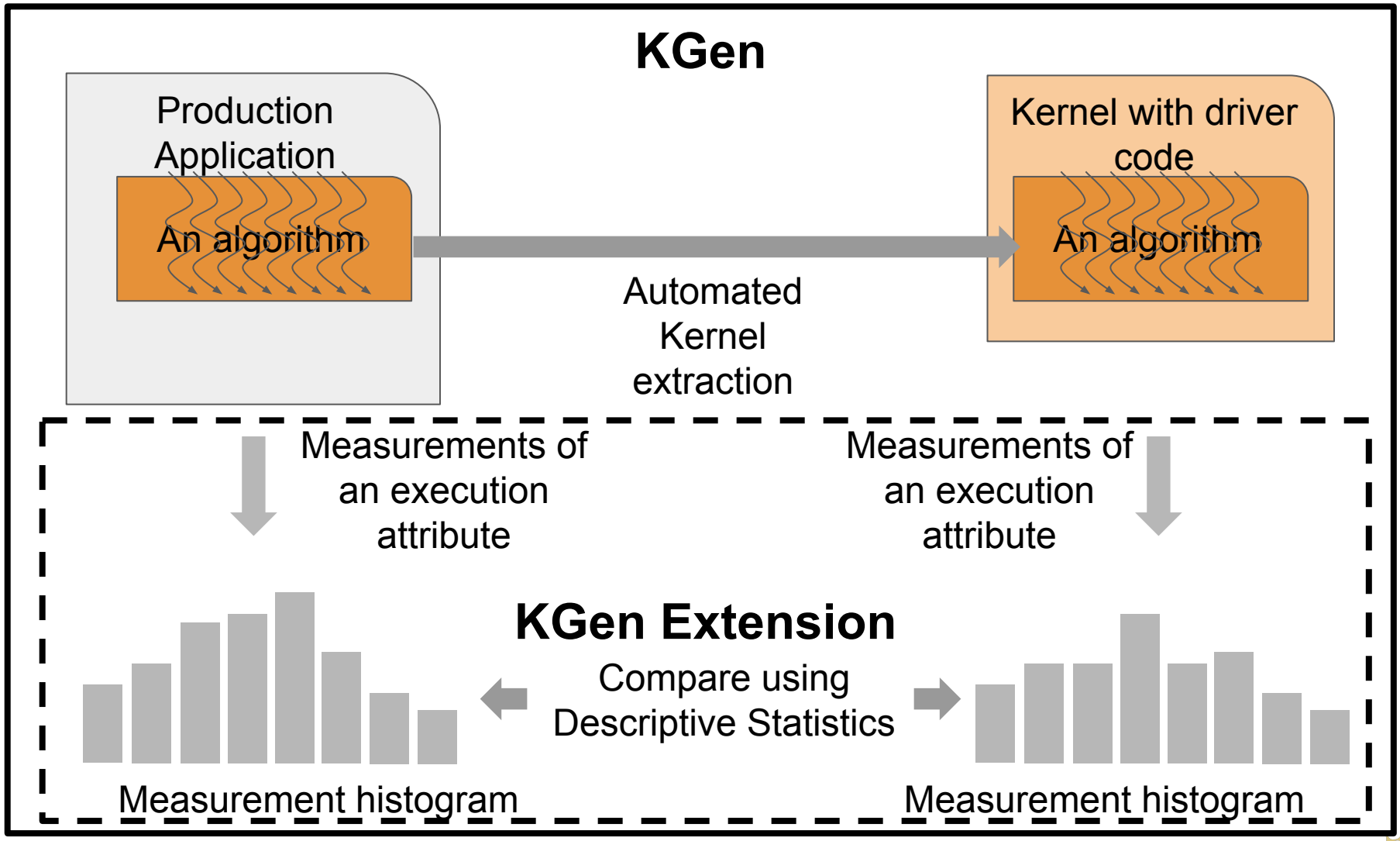
# Overview




KGen
Production Application
An algorithm
Kernel with driver code
An algorithm
Automated Kernel extraction
Measurements of an execution attribute
Measurements of an execution attribute
Measurement histogram
Measurement histogram

# Overview




KGen
Production Application
An algorithm
Automated Kernel extraction
Kernel with driver code
An algorithm
Measurements of an execution attribute
Measurements of an execution attribute
KGen Extension
Compare using Descriptive Statistics
Measurement histogram
Measurement histogram

# KGen Introduction

# Manual extraction vs. automated extraction



**SUBROUTINE** micro_mg_tend(microp_uniform, pcols, pver, ncol, top_lev, deltatin, tn, qn, qc, qi, nc, ni, p, pdel, cldn, liqcldf, relvar, accre_enhan, icecldf, rate1ord_cw2pr_st, naai, npccnin, rndst, nacon, tlat, qvlat, qctend, qitend, nctend, nitend, effc, effc_fn, effi, prect, preci, nevapr, evapsnow, am_evp_st, prain, prodsnow, cmeout, deffi, pgamrad, lamcrad, qsout, dsout, rflx, sflx, qrout, reff_rain, reff_snow, qcsevap, qisevap, qvres, cmeiout, vtrmc, vtrmi, qcsedten, qisedten, prao, prco, mnuccco, mnuccto, msacwio, psacwso, bergso, bergo, melto, homoo, qcreso, prcio, praio, qireso, mnuccro, pracso, meltsdt, frzrdt, mnuccdo, nrout, nsout, refl, arefl, areflz, frefl, csrfl, acsrfl, fcsrfl, rercld, ncai, ncal, qrout2, qsout2, nrout2, nsout2, drout2, dsout2, freqs, freqr, nfice, do_cldice, tnd_qsnow, tnd_nsnow, re_ice, errstring)

- 1921 lines without comment and blank lines
- 11 functions called directly or indirectly

**END SUBROUTINE** micro_mg_tend

# Why we needed a kernel generation tool?



Community Earth System Model (CESM)

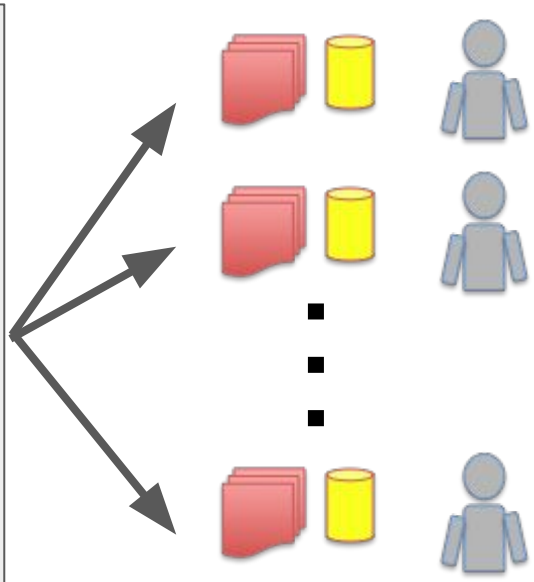
- > 1.5 mil. LOC
- > 400 registered developers
- > 5100 registered users

# Why we needed a kernel generation tool?



Community Earth System Model (CESM)

- > 1.5 mil. LOC
- > 400 registered developers
- > 5100 registered users

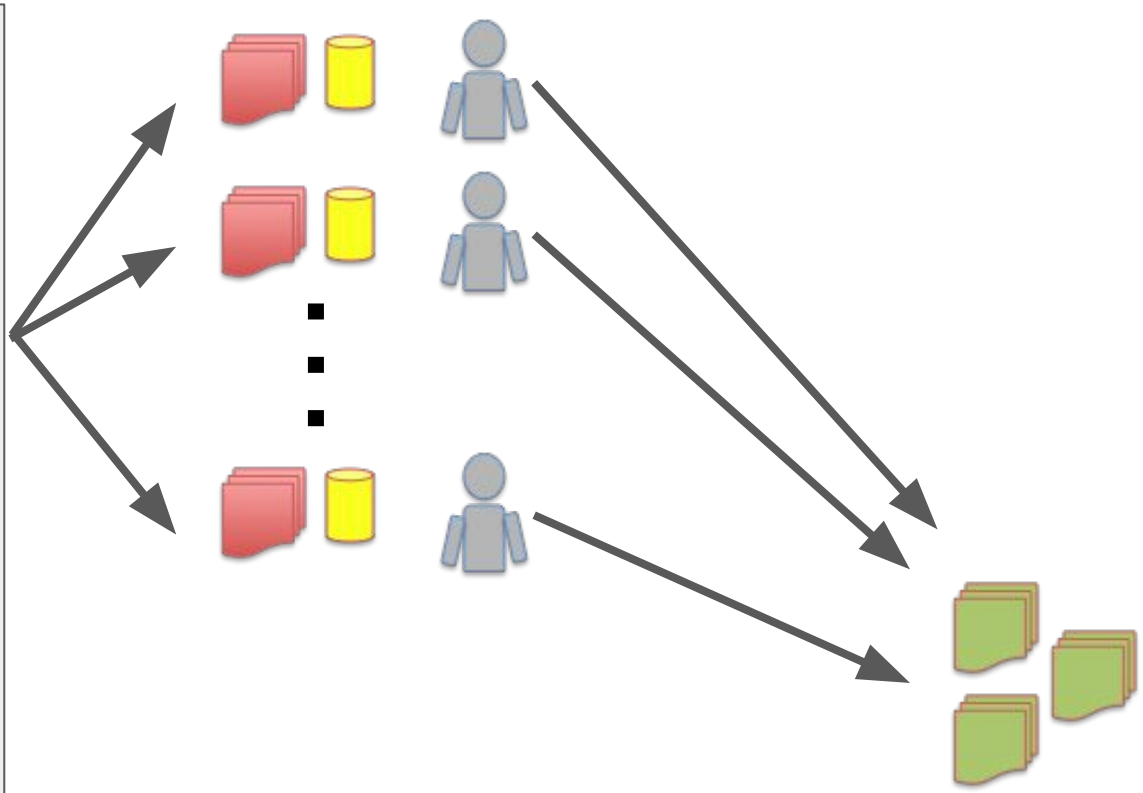


# Why we needed a kernel generation tool?



Community Earth System Model (CESM)

- > 1.5 mil. LOC
- > 400 registered developers
- > 5100 registered users

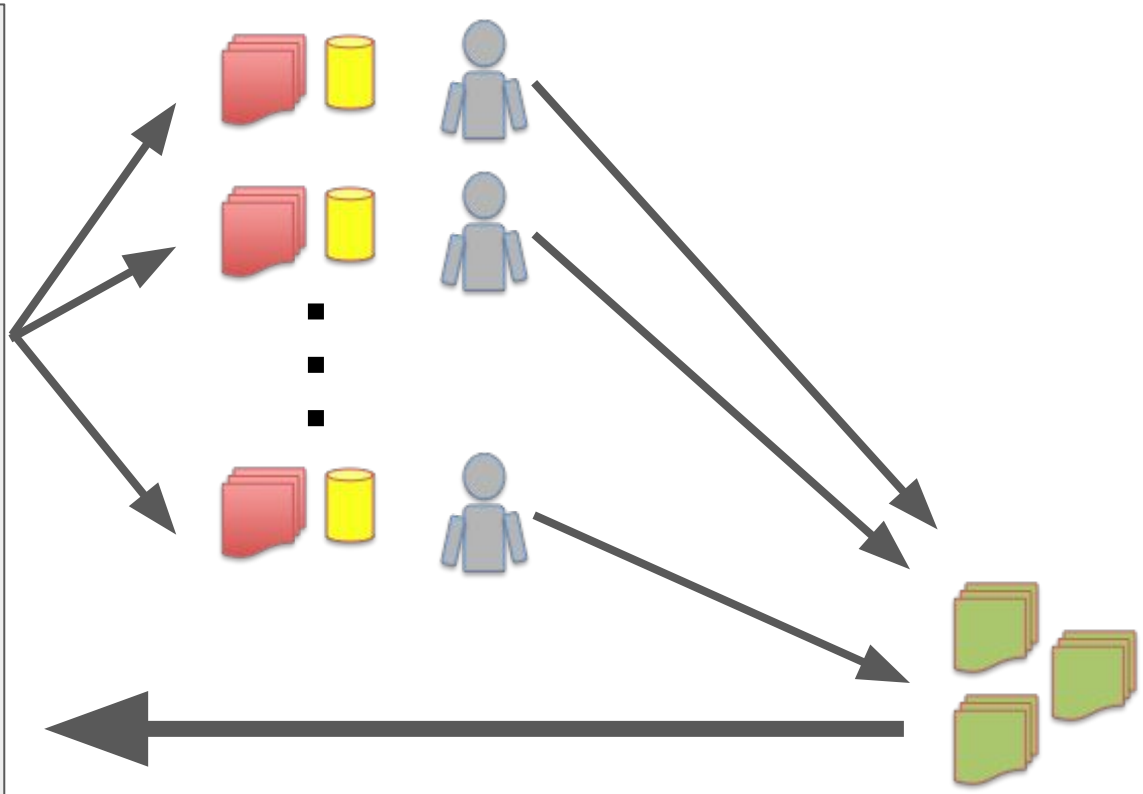


# Why we needed a kernel generation tool?



Community Earth System Model (CESM)

- > 1.5 mil. LOC
- > 400 registered developers
- > 5100 registered users

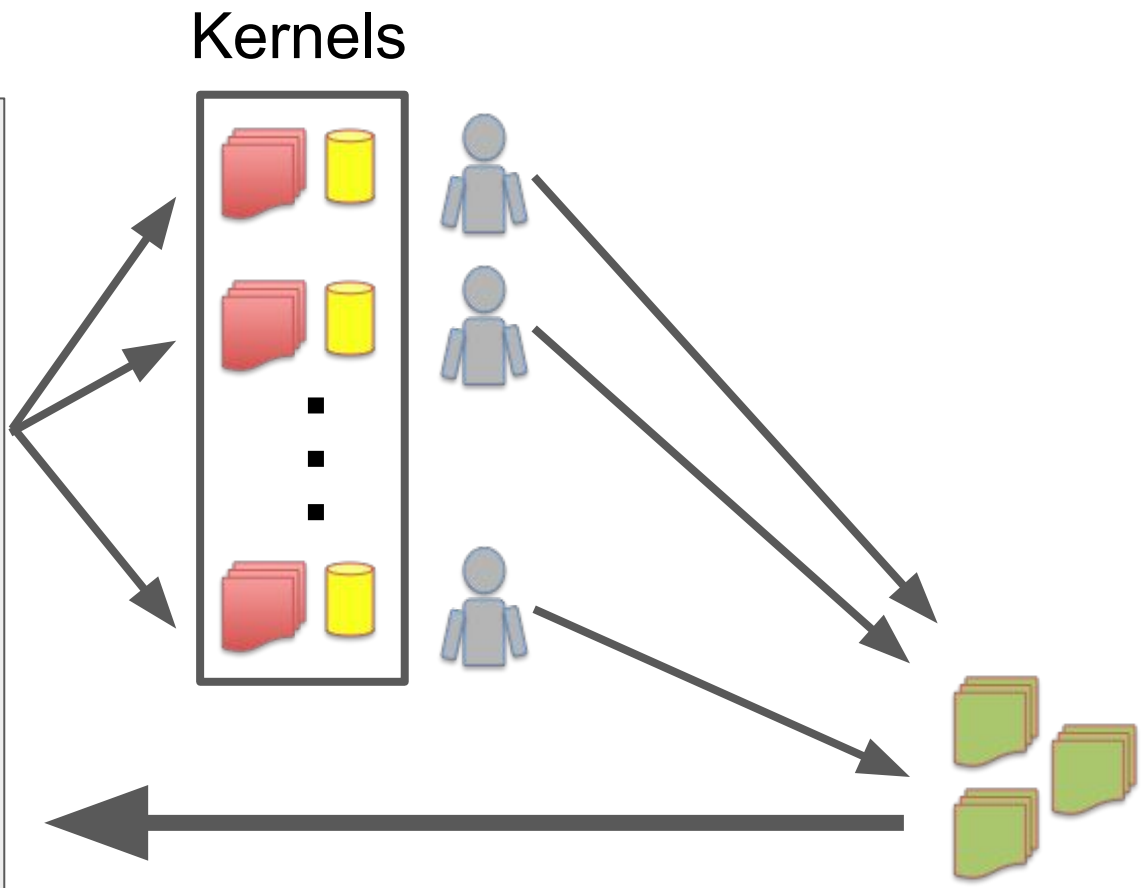


# Why we needed a kernel generation tool?



Community Earth System Model (CESM)

- > 1.5 mil. LOC
- > 400 registered developers
- > 5100 registered users

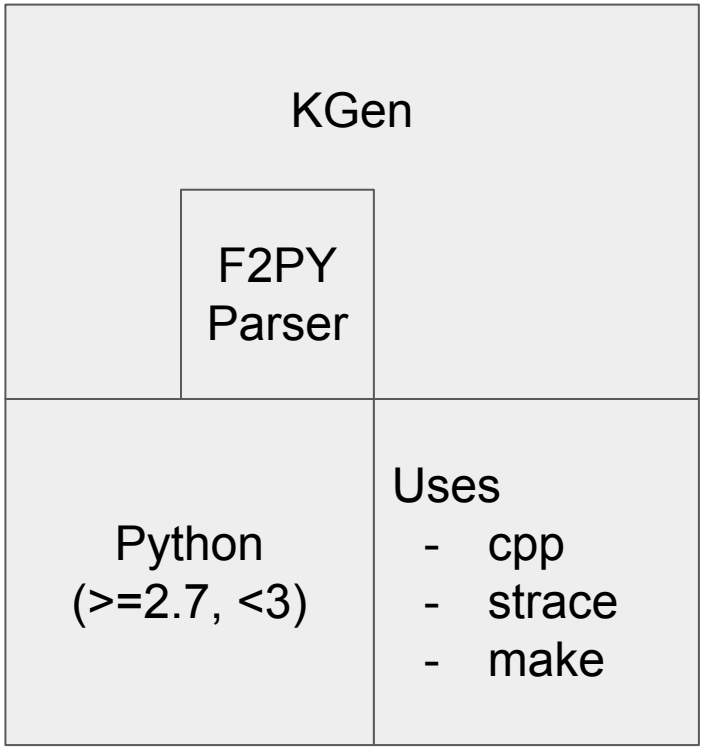


- KGen **extracts** a region of source-code as a stand-alone software out of a large Fortran software application
- In addition, it generates **input & output data** for execution and verification of the generated kernel
- **Correctness check and timing measurement** are included in the generated kernel

KGen github: https://github.com/NCAR/KGen

# KGen: self-contained static analysis tool



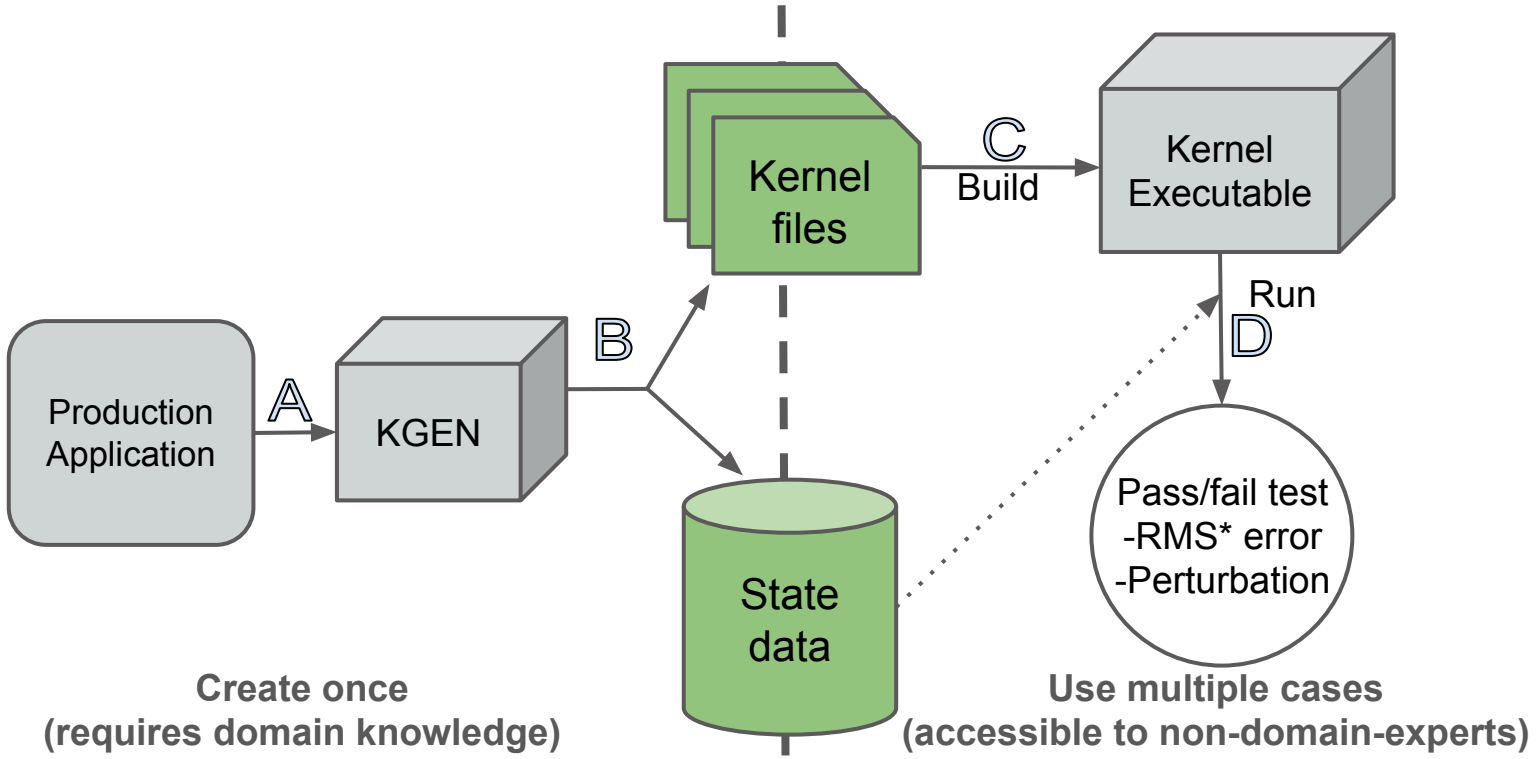



- Written in Python
  - No external compiler or analysis framework
  - Performs static analysis on ASTs
- Uses F2PY* Fortran Parser
  - Integrated and extended in KGen
  - Produces ASTs for analysis
- Uses three Linux utilities
  - cpp: preprocessing
  - strace: collect macros and include paths
  - make: executes KGen-generated kernel

F2PY*: Fortran to Python interface generator(https://github.com/pearu/f2py)

# KGen: Create a kernel once, use many times






A) User specifies a kernel region and also provides clean/build/run command(s) for application

B) KGen generates a stand-alone kernel files and state data files

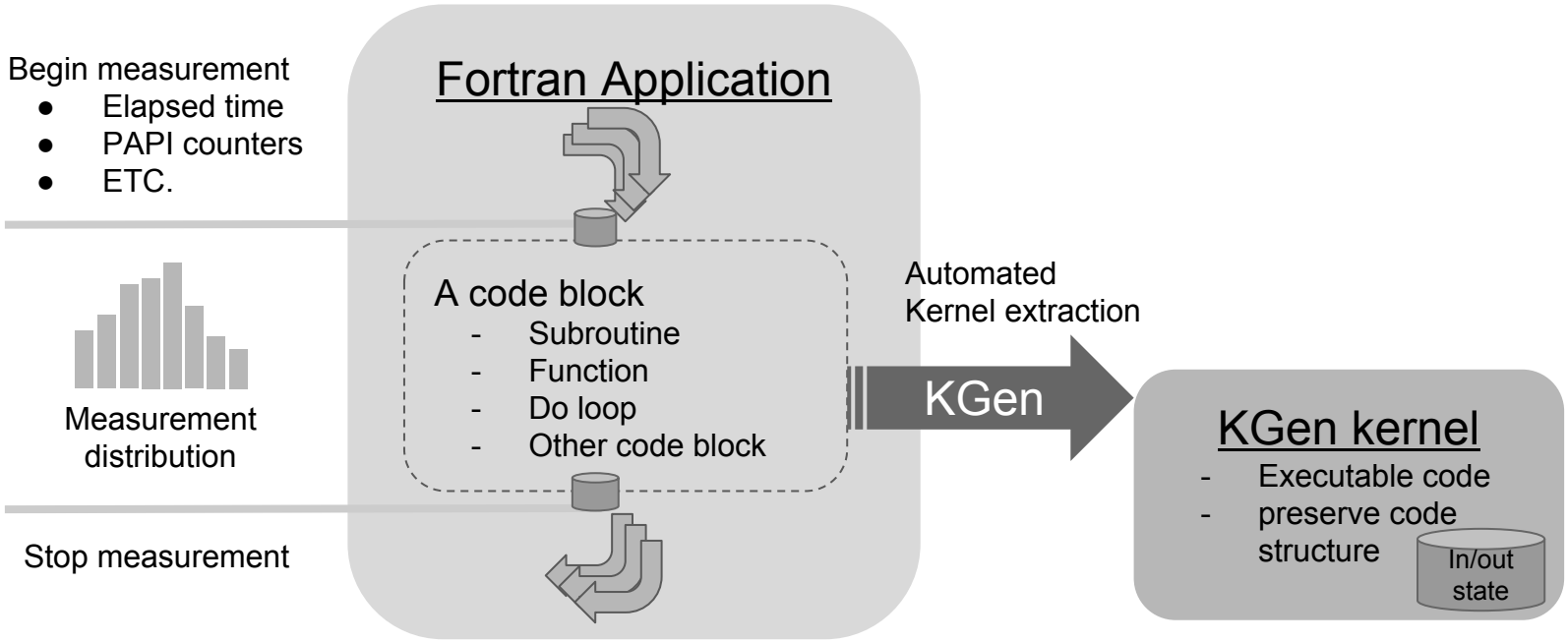
C) User builds kernel through KGen-generated Makefile

D) User runs kernel with state data. KGen kernel includes verification and timing measurement

# KGen Representativeness Extension

# KGen Extension Concept

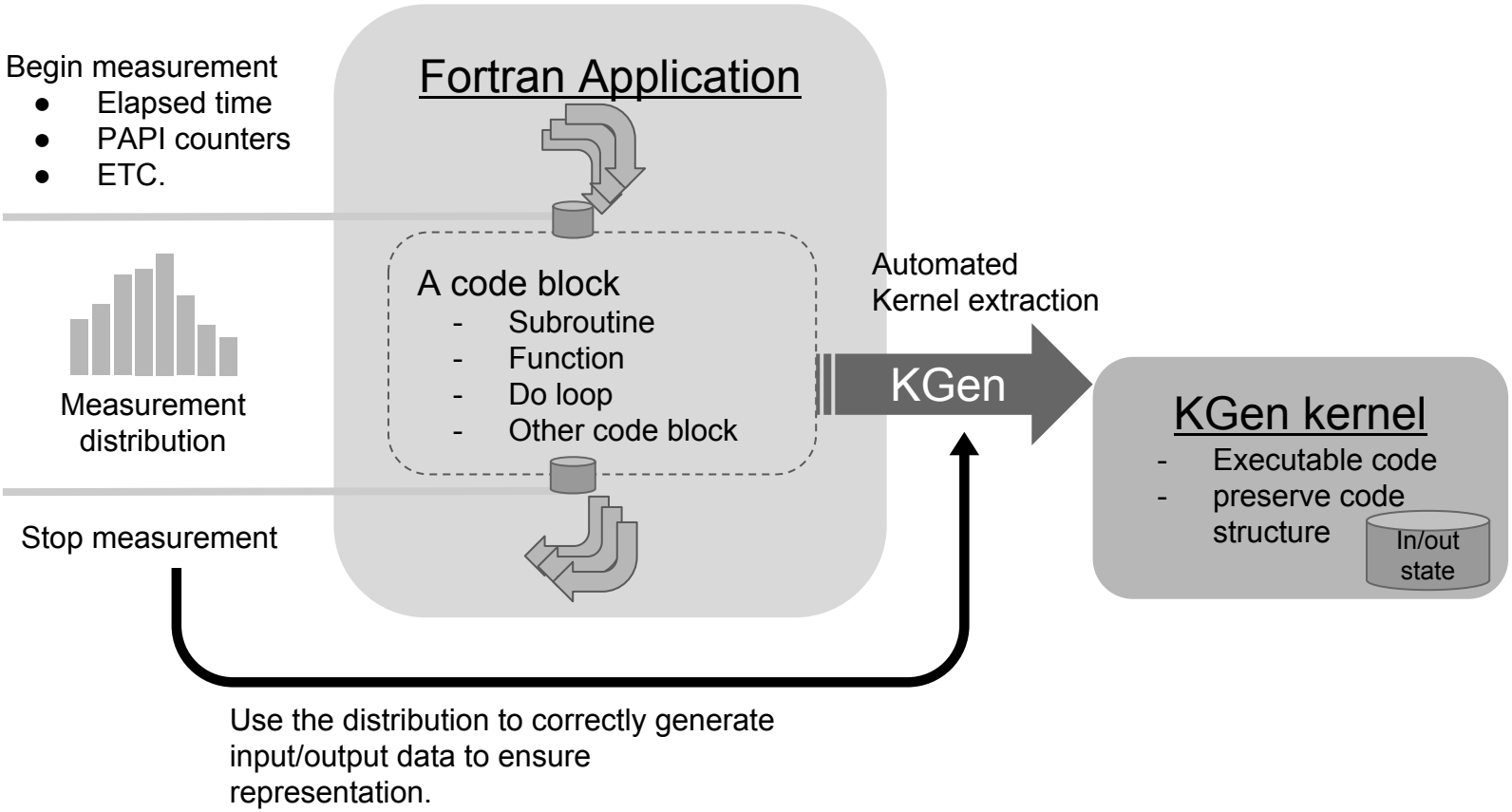





: Program Execution : Data Collection

# KGen Extension Concept

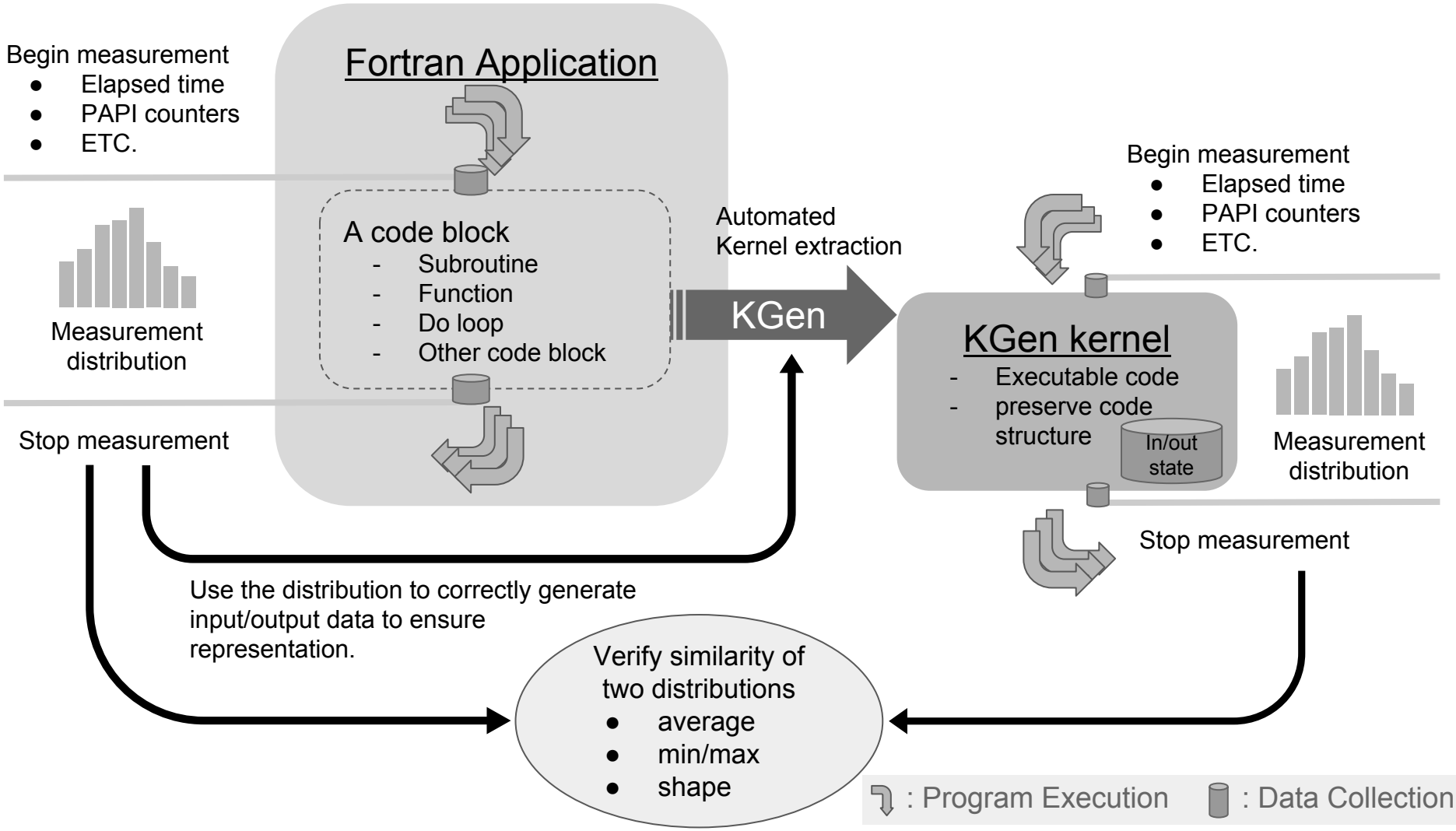





: Program Execution : Data Collection

# KGen Extension Concept





# Content of measurements



- Invocation Triplets
  - MPI rank number
  - OpenMP thread ID
  - Invocation order
- Runtime attributes
  - Elapsed time
  - PAPI counter
  - Source code coverage
  - Will add more attributes in future

**Saved as 4-ary tuples of**

**( “MPI rank”, “OpenMP TID”, “Invocation order”, “runtime attribute”)**

# Measurement Method

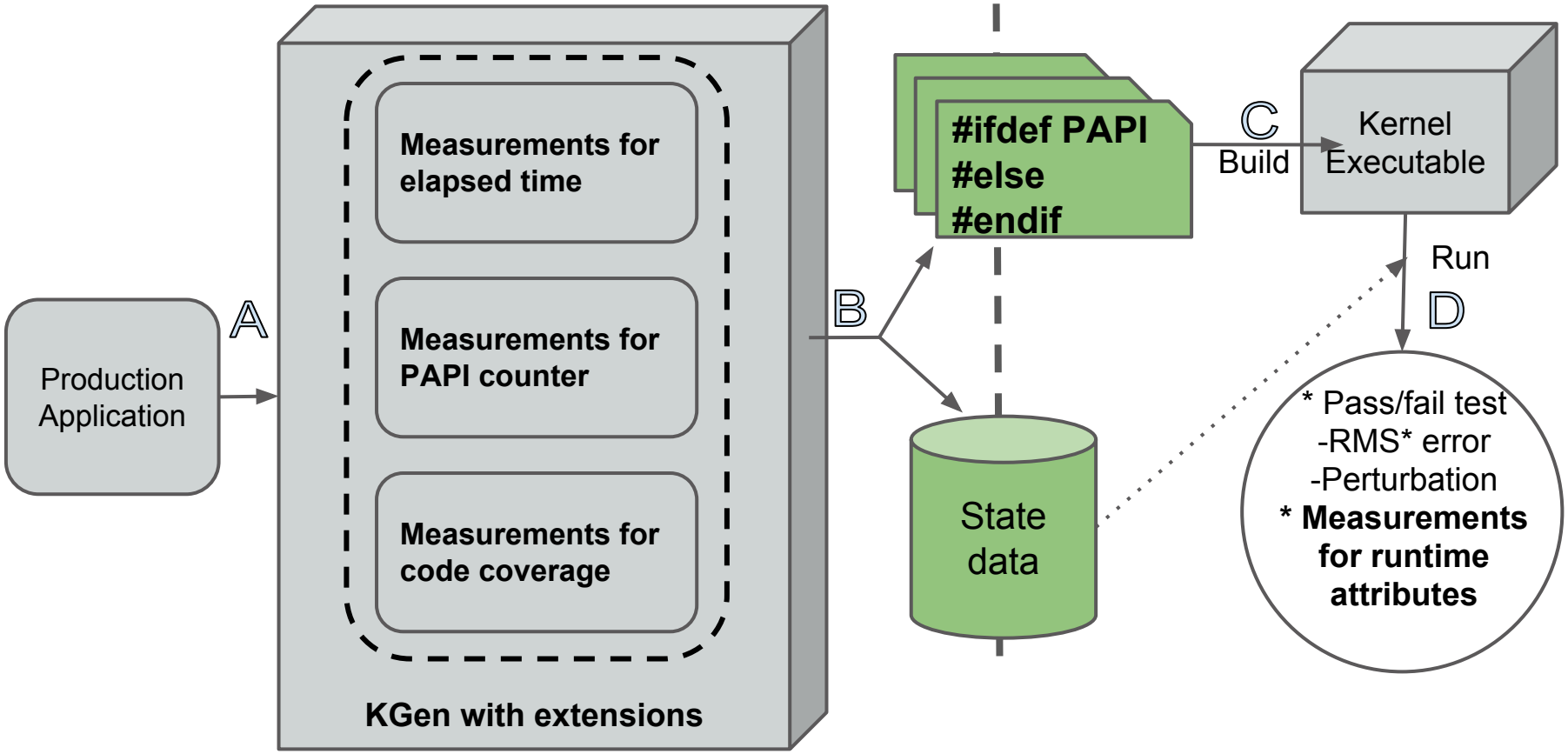


- Source code instrumentation for target application
  - Add collection code just before and after the kernel region
  - Maintains invocation triplets internally

- Additional code for generated kernel
  - Use macros to distinguish runtime attribute
  - Generated makefile helps to build with PAPI library
  - Source code coverage information is presented as annotated source code in each files.

# KGen with an extensions





# Descriptive statistics



- Types of descriptive statistics

| type | statistics |
|---|---|
| Central tendency | mean, median, mode |
| Variability | min, max, standard deviation, variance |
| Shape | kurtosis, skewness |

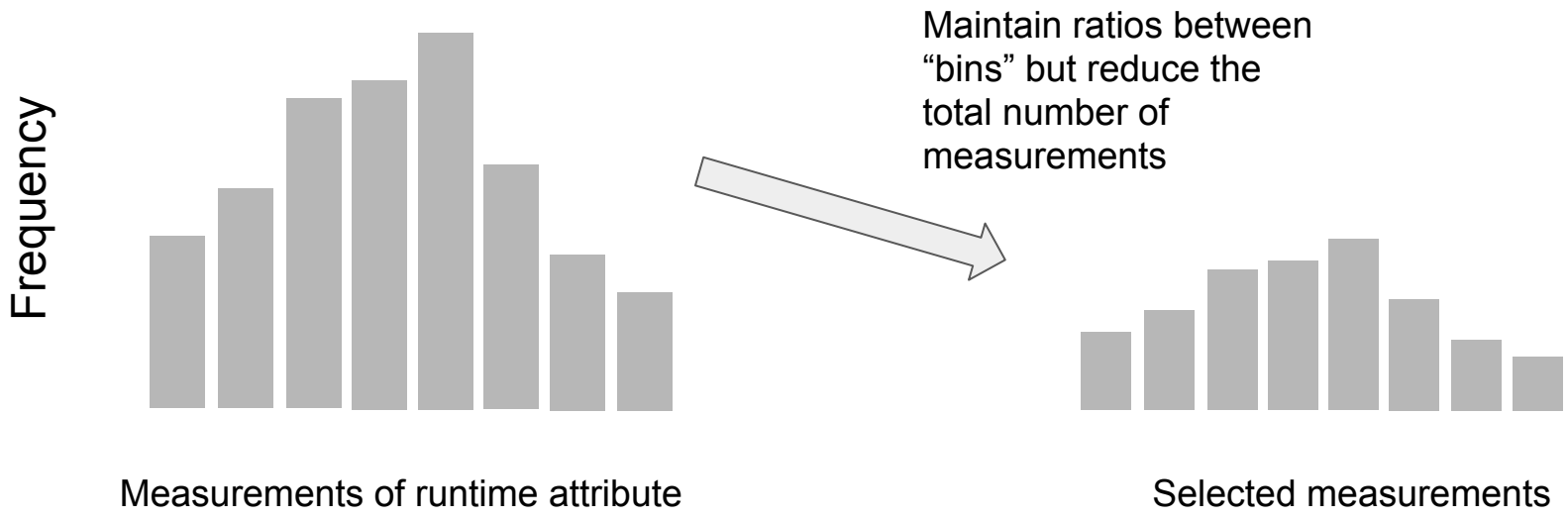
- Calculating a metric for difference

$$DIFF = \frac{STAT_{KERNEL} - STAT_{APP}}{STAT_{APP}} * 100(\%)$$

# How to select "best" set of input data?



- "Best" means differently to different usages
  - Some may be interested in a simple average
  - Or, shape of distribution for others
- In this paper, we introduced an algorithm to preserve the shape of distribution while reducing the size of data



# Case Studies

# Experiment Configuration

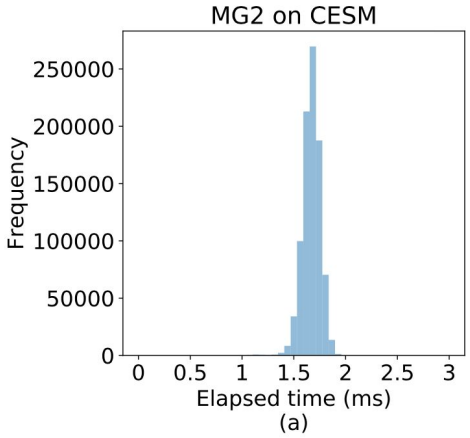


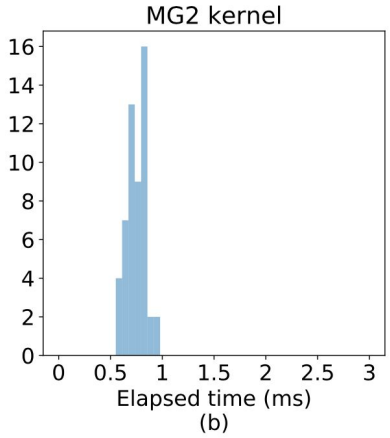
- System
  - Yellowstone computing cluster of National Center for Atmospheric Research (NCAR): 2.6GHz Intel Xeon E5-2670, 16 cores per node
  - Intel Fortran Compiler 16.0.1
- Applications
  - Community Earth System Model (CESM)
    - Morrison-Gettelman Cloud Physics 2 on **630 MPI ranks** and 2 OpenMP threads per a MPI rank on **42 computing nodes**
    - Rapid Radiative Transfer Model for General Circulation Models (RRTMG) Longwave Radiation Physics with the same runtime configuration to MG physics **except only one MPI rank for atmospheric component**
  - Linear Algebra Package (LAPACK)
    - Singular Value Decomposition Routine (**DGESVD**)
    - Sequential Test Program

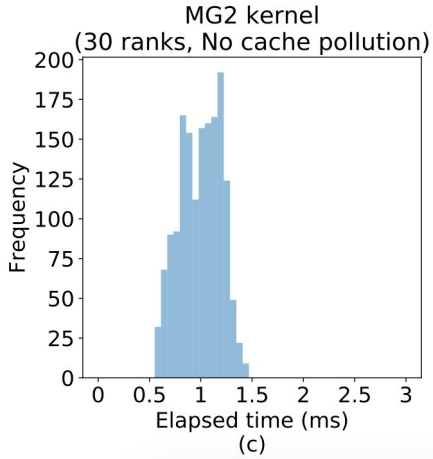
# MG2 kernel

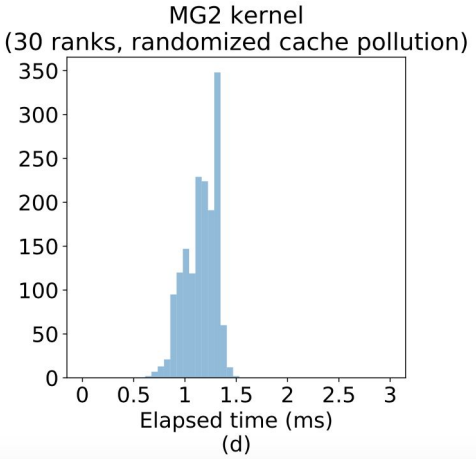


- Elapsed time for MG2 kernel
  - MG2 kernel is one of time consuming cloud physics routine in CESM*




| Stat. type | Value (a) | Value (b) | Diff(%) (a) ~ (b) | Value (d) | Diff(%) (a) ~ (d) |
|---|---|---|---|---|---|
| Mean (ms) | 1.67 | 0.75 | -55 | 1.16 | -30.8 |
| (min., max.) | (1.0, 2.5) | (0.6, 1.0) | (-42.5, -62.4) | (0.65, 1.53) | (-34.6, -38.8) |
| Std. dev. | 0.12 | 0.47 | 299.7 | 0.35 | 198 |
| Skew | -0.76 | -0.44 | -42.2 | -0.54 | -29 |

# Techniques to improve kernel representation



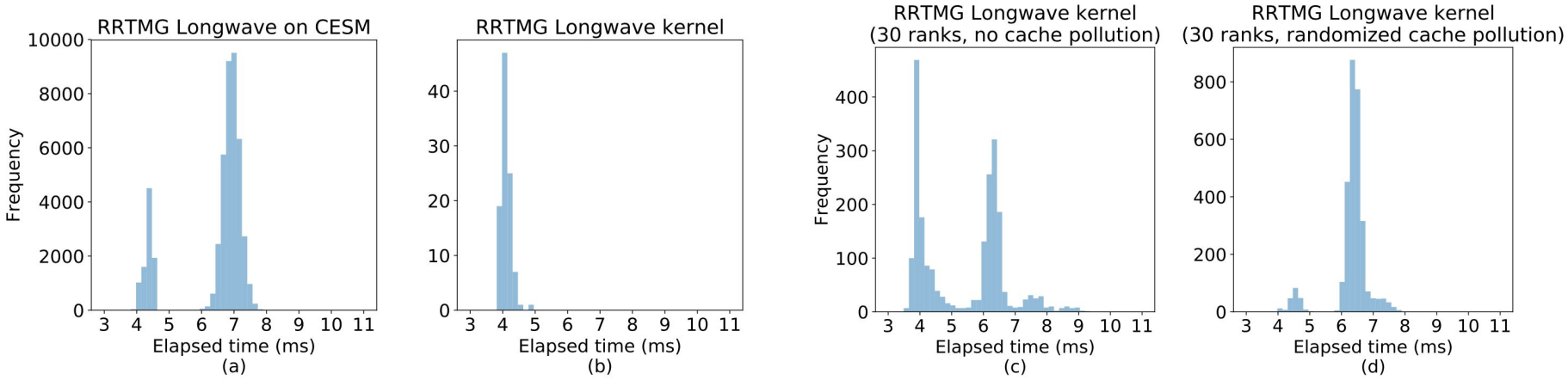
- A cache pollution method
  - Increment values of each entry of an array from an element index that is randomly selected
  - Repeat the writing one more time

- Repeat kernel execution on multiple MPI ranks
  - Create multiple MPI ranks and run the same kernel on each ranks

# RRTMG Longwave kernel



- Elapsed time for RRTMG Longwave kernel

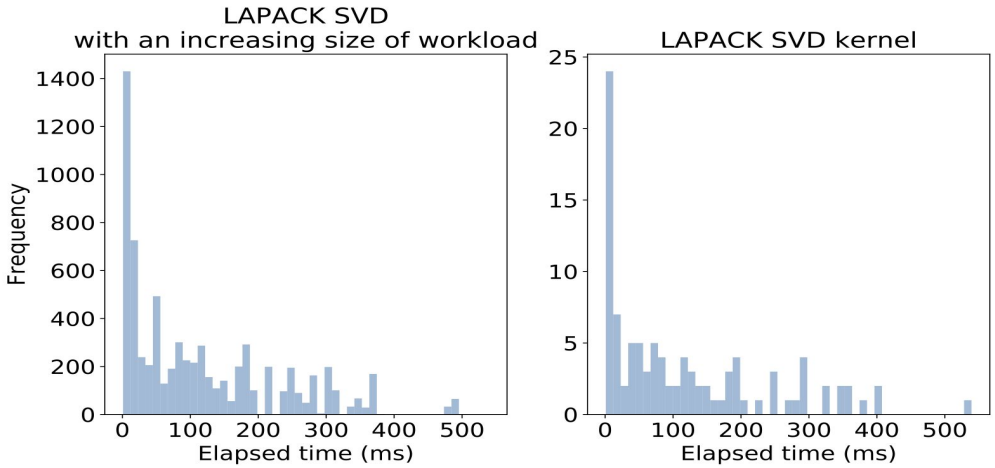



| Stat. type | Value (a) | Value (b) | Diff(%) (a) ~ (b) | Value (d) | Diff(%) (a) ~ (d) |
|---|---|---|---|---|---|
| Mean (ms) | 6.44 | 4.12 | -36 | 6.35 | -1.4 |
| (min., max.) | (3.84 7.8) | (3.83, 4.95) | (-0.3, -36.5) | (4.02, 7.9) | (-4.7, -1.3) |

# SVD routine in LAPACK



- Elapsed time is measured for SVD routines of LAPACK
  - Increasing workload from 64X64 matrix to 120X92

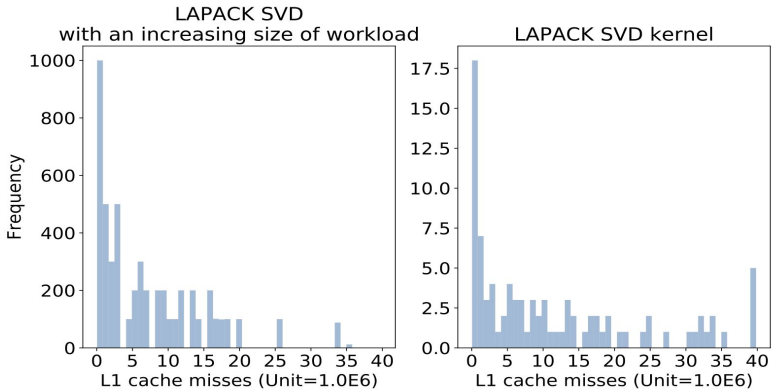



| Stat. type | SVD | SVD kernel | Diff(%) |
|---|---|---|---|
| Mean (ms) | 114 | 118 | 2.9 |
| (min., max.) | (1.0, 498) | (2.7, 539) | (-17.1, 8.3) |
| Std. dev. | 0.46 | 0.46 | 0.1 |
| Skew | 1.07 | 0.98 | -8.5 |

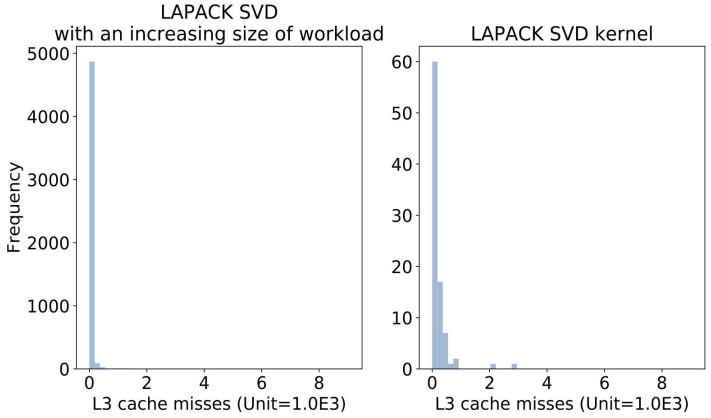
# SVD routine in LAPACK



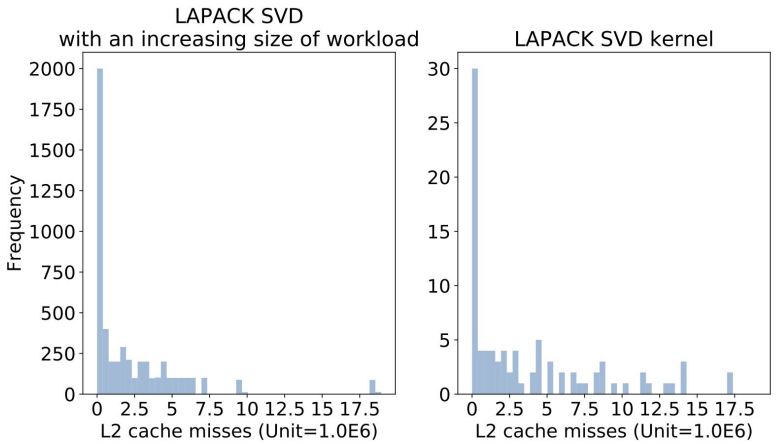
- PAPI cache counters for the same SVD routine




**L1 cache misses**




**L3 cache misses**




**L2 cache misses**

- Not fully automated yet.
- User may need to modify build process to link with PAPI library

# Source code coverage



- Original code

```
MODULE calc_mod
   PUBLIC calc
CONTAINS
   SUBROUTINE calc(i, j, output, out2, out3)
      INTEGER, INTENT(IN) :: i, j
      real, INTENT(OUT), dimension(:,:) :: out3, output, out2
      IF ( i > j ) THEN
         output(i,j) = i - j
         out2(i, j) = 2*(i-j)
         out3(i, j) = 3*(i-j)
      ELSE
         output(i,j) = j - i
         out2(i, j) = 2*(j-i)
         out3(i, j) = 3*(j-i)
      END IF
   END SUBROUTINE
END MODULE
```

- Annotated code

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!! 2 conditional blocks exist in this file
!! 2 conditional blocks are executed at least once among all
the conditional blocks.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

MODULE calc_mod
   PUBLIC calc
CONTAINS
   SUBROUTINE calc(i, j, output, out2, out3)
...
      IF ( i > j ) THEN
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!! Total number of visits: 60
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      ELSE
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!! Total number of visits: 100
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      END IF
   END SUBROUTINE
END MODULE
```

# Conclusion



- Summary
  - KGen: an automated Fortran kernel generator
  - KGen representativeness extension:
    - Provides a quantitative metrics for the degree of kernel representation
    - Allows to improve kernel representation
- Future works
  - Analysis of cache pollution methods
  - Automated PAPI measurements
  - Better support for newer Fortran spec.
  - Better support for pointer analysis

# Thank you!

# Q & A

Funded partially by Intel Parallel Computing Center focused on Weather and Climate Simulation (IPCC-WACS)

KGen Github repository: https://github.com/NCAR/KGen
Further questions and comments to: kgen@ucar.edu

# Use Cases

# Use case 1 - Simulation Verification



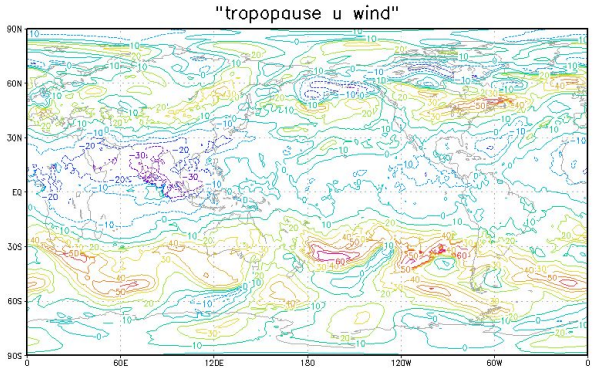
- How to verify that a new simulation is consistent with previous simulation results?

# Use case 1 - Simulation Verification



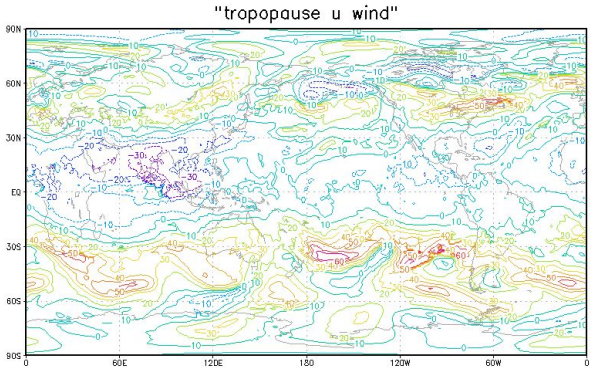
- How to verify that a new simulation is consistent with previous simulation results?

Reference CESM simulation on A system




≠

New CESM simulation on B system



# Use case 1 - Simulation Verification



- We found several variables from two simulations were **slightly different** between system A and B
- **Challenging to find a root cause** mainly due to:
  - Linear combinations of 120 physical variables
  - Several hours to run one-year sim. on 900 cores
- **Morrison-Gettelman(MG)1** microphysics
  - modifies many of the “suspicious” variables
- MG1 **kernel is generated** using KGen
  - Performed experiments using the kernel
  - Located one line that starts to show the inconsistency
- The root cause: **Fused-Multiply-Add(FMA)**
  - Without FMA, the kernel returned a reasonable RMS difference

# Use case 2 - Performance Optimization



- MG2 Kernel
  - One of time consuming routines in CESM*
  - ~2X speedup on Intel SandyBridge CPU
- Customized Benchmark Suite
  - MG2 is a part of benchmark suite for NCAR Wyoming Supercomputing Center (NWSC) procurement
- NCAR Kernel Repository
  - https://github.com/NCAR/kernelOptimization
  - ~40 kernels are generated. Being optimized internally and externally(compiler vendors and Int'l collaborators)
  - Trying to standardize testing method and reporting format

# Other potential use cases



- Unit test generation
  - A region of codes can be extracted as a test case
  - Data for driving test and verifying the code are also generated simultaneously
- Debugging large application
  - No need to use queuing system
  - Focus on the region of interesting code
- And more
  - A kernel is a yet another software ready to be built/run
  - An use case is upto user's needs and imagination

# KGen users



- Current users
  - Being used and/or evaluated by individuals mostly within climate/weather simulation community
    - NOAA Geophysical Fluid Dynamics Lab., ETH Zurich, U of Hamburg, Oak Ridge Nat'l Lab., U of Tennessee, U of Utah
    - Interested in performance optimization, porting to new processor, and enhancing unit test framework
- New users
  - KGen is available to "early-adopters"
    - https://github.com/NCAR/KGen
    - Welcome your opinion and comments on using KGen ( kgen@ucar.edu )
  - Welcome new users from other scientific fields too

# Back-ups

# Kernels in Scientific Computing



- Definition used in the paper
  - “a self-contained program that embodies essential performance characteristics of the node-level aspects of an application.”(Janssen, 2011)
- Related Research
  - Pioneering work by Heroux by defining a new role of kernel (Heroux, 2009)
  - Performance characterization by a kernel (Akel, 2013)
  - Assessing the role of mini-app (Barrett, 2015)
- Usage in practice
  - Benchmarks: STREAMS, Livermore Loops
  - Mantevo Project: A 2013 R&D 100 Award. the Mantevo Miniapp Suite
  - UK Mini-App Consortium: Many mini-apps including CloverLeaf
  - FIBER: A suite of mini-apps maintained by RIKEN Advanced Institute for Computational Science
  - DOE Exascale Co-design Centers: Exascale Co-Design Center for

# Kernel Generation



- Kernel Generation Methods
  - Manual vs. Automatic
  - Creates from scratch by whom knows original software well
  - Extracts partially or entirely from original software
- Difficulties in extracting a kernel manually
  - Manual searching of declarations for variables used in a kernel
  - Data preparation for deriving kernel execution
  - Especially time-consuming (AND error-prone) when:
    - there are large numbers of inputs/outputs
    - deep hierarchy of structured data types
    - scattered in many separate source files
    - heavily uses external libraries
  - Even with correctly generated kernels, it is, in practice, hard problem to verify the extracted kernel behaves “sufficiently close” to original application.

# Automated Kernel Generation



- Static (and dynamic) analysis of software
  - Generally uses features of a compiler or an analysis framework
  - Applies to source-code level or intermittent-code level
  - State information collected from source-code instrumentation or memory/cache H/W
- Previous work
  - Code Isolator(Lee, 2005): Based on SUIF compiler, source-code level
  - Codelet Finder(CAPS[*1]): Based on another CAPS product(ASTEX[*2]), source-code level
  - ROSE Outliner(Liao, 2010): Based on ROSE[*3] framework, source-code level
  - CERE(Castro, 2015): Based on LLVM. intermittent-code level
- Automation approaches in this paper
  - Based on Python. F2PY[*4] Fortran parser is integrated
  - AST search to decide minimum set of Fortran statements for a stand-alone executable

# A kernel generation example



**program.F90**

```
PROGRAM calc
  USE update_mod, only : &
    update
  INCLUDE 'mpif.h'
  INTEGER t, rank, N, err
  CALL mpi_init(err)
  CALL mpi_comm_size(...)
  CALL mpi_comm_rank(...)
  DO t=1,10
    CALL update(rank, N)
  END DO
  CALL mpi_finialize(err)
END PROGRAM
```

**update_mod.F90**

```
MODULE update_mod
  USE calc_mod, only : calc
  PUBLIC update
CONTAINS
  SUBROUTINE update(rank, N)
    INCLUDE 'mpif.h'
    INTEGER, INTENT(IN)::rank,N
    INTEGER :: i, j,err,lsum,gsum(N)
    INTEGER :: output(COL,ROW)
    gsum = 0
    DO i=1,COL
      DO j=1,ROW

        CALL calc(i, j, output)
      END DO
    END DO
    lsum = SUM(output)
    CALL mpi_gather(lsum, …)
    IF (rank==0) THEN
      print *,'global sum=', SUM(gsum)
    END IF
  END SUBROUTINE
END MODULE
```

**calc_mod.F90**

```
MODULE calc_mod
  PUBLIC calc
CONTAINS
  SUBROUTINE calc(i, j, output)
    INTEGER, INTENT(IN) :: i, j
    INTEGER, INTENT(OUT), &
        dimension(:,:) :: output
    CALL print_msg('start')
    output(i,j) = i + j
    CALL print_msg('finish')
  END SUBROUTINE
  SUBROUTINE print_msg(msg)
    CHARACTER(*),INTENT(IN):: &
      msg
    PRINT *, msg
  END SUBROUTINE
END MODULE
```

# A kernel generation example



**program.F90**

```
PROGRAM calc
  USE update_mod, only : &
    update
  INCLUDE 'mpif.h'
  INTEGER t, rank, N, err
  CALL mpi_init(err)
  CALL mpi_comm_size(...)
  CALL mpi_comm_rank(...)
  DO t=1,10
    CALL update(rank, N)
  END DO
  CALL mpi_finialize(err)
END PROGRAM
```

**update_mod.F90**

```
MODULE update_mod
  USE calc_mod, only : calc
  PUBLIC update
CONTAINS
  SUBROUTINE update(rank, N)
    INCLUDE 'mpif.h'
    INTEGER, INTENT(IN)::rank,N
    INTEGER :: i, j,err,lsum,gsum(N)
    INTEGER :: output(COL,ROW)
    gsum = 0
    DO i=1,COL
      DO j=1,ROW
        !$KGEN callsite calc
        CALL calc(i, j, output)
      END DO
    END DO
    lsum = SUM(output)
    CALL mpi_gather(lsum, …)
    IF (rank==0) THEN
      print *,'global sum=', SUM(gsum)
    END IF
  END SUBROUTINE
END MODULE
```

Call-site

**calc_mod.F90**

```
MODULE calc_mod
  PUBLIC calc
CONTAINS
  SUBROUTINE calc(i, j, output)
    INTEGER, INTENT(IN) :: i, j
    INTEGER, INTENT(OUT), &
       dimension(:,:) :: output
    CALL print_msg('start')
    output(i,j) = i + j
    CALL print_msg('finish')
  END SUBROUTINE
  SUBROUTINE print_msg(msg)
    CHARACTER(*),INTENT(IN):: &
      msg
    PRINT *, msg
  END SUBROUTINE
END MODULE
```

Kernel

# A kernel generation example



**program.F90**

```
PROGRAM calc
  USE update_mod, only : &
    update
  INCLUDE 'mpif.h'
  INTEGER t, rank, N, err
  CALL mpi_init(err)
  CALL mpi_comm_size(...)
  CALL mpi_comm_rank(...)
  DO t=1,10
    CALL update(rank, N)
  END DO
  CALL mpi_finialize(err)
END PROGRAM
```

**update_mod.F90**

```
MODULE update_mod
  USE calc_mod, only : calc
  PUBLIC update
CONTAINS
  SUBROUTINE update(rank, N)
    INCLUDE 'mpif.h'
    INTEGER, INTENT(IN)::rank,N
    INTEGER :: i, j,err,lsum,gsum(N)
    INTEGER :: output(COL,ROW)
    gsum = 0
    DO i=1,COL
      DO j=1,ROW
        !$KGEN callsite calc
        CALL calc(i, j, output)
      END DO
    END DO
    lsum = SUM(output)
    CALL mpi_gather(lsum, …)
    IF (rank==0) THEN
      print *,'global sum=', SUM(gsum)
    END IF
  END SUBROUTINE
END MODULE
```

Call-site

**calc_mod.F90**

```
MODULE calc_mod
  PUBLIC calc
CONTAINS
  SUBROUTINE calc(i, j, output)
    INTEGER, INTENT(IN) :: i, j
    INTEGER, INTENT(OUT), &
       dimension(:,:) :: output
    CALL print_msg('start')
    output(i,j) = i + j
    CALL print_msg('finish')
  END SUBROUTINE
  SUBROUTINE print_msg(msg)
    CHARACTER(*),INTENT(IN):: &
      msg
    PRINT *, msg
  END SUBROUTINE
END MODULE
```

Kernel

# Generated kernel files



**kernel_driver.f90**

```
PROGRAM kernel_driver
  ...
  OPEN(UNIT=kgen_unit, &
     FILE=kgen_filepath, …)
  ...
  CALL update(kgen_unit)

  CLOSE(kgen_unit)
  ...
END PROGRAM
```

**update_mod.F90**

```
MODULE update_mod ...
CONTAINS
  SUBROUTINE update(kgen_unit)
    INTEGER, INTENT(IN) :: kgen_unit
    ...
    CALL kgen_init_check(check_status, tolerance)
    READ(UNIT=kgen_unit) i
    READ(UNIT=kgen_unit) j
    …
    READ(UNIT=kgen_unit) ref_output
    CALL calc(i, j, output) ! call to kernel
    CALL kgen_verify_integer_4_dim2(...
    CALL kgen_print_check("calc", check_status)
    CALL system_clock(start_clock, rate_clock)
      …
    CALL system_clock(stop_clock, rate_clock)
    PRINT *, "calc : Time per call (usec): ", ...

    CONTAINS
      SUBROUTINE kgen_verify_integer_4_dim2(...)
        …
        IF (ALL(var==ref_var)) THEN ...
        END IF
      END SUBROUTINE
  END SUBROUTINE
END MODULE
```

**calc_mod.F90**

```
MODULE calc_mod
  ...
CONTAINS
  SUBROUTINE calc(i, j, output)
    ...
    !kgen_excluded CALL &
      print_msg('start')
    output(i,j) = i + j
    !kgen_excluded CALL &
     print_msg('finish')
  END SUBROUTINE
END MODULE
```

# Generated kernel files



**kernel_driver.f90**

```
PROGRAM kernel_d
  OPEN(UNIT=kgen_unit, &
    FILE=kgen_filepath, …)
  ...
  CALL update(kgen_unit)

  CLOSE(kgen_unit)
  ...
END PROGRAM
```

Open data file to read

**update_mod.F90**

```
MODULE update_mod ...
CONTAINS
  SUBROUTINE update(kgen_unit)
    INTEGER, INTENT(IN) :: kgen_unit
    ...
    CALL kgen_init_check(check_status, tolerance)
    READ(UNIT=kgen_unit) i
    READ(UNIT=kgen_unit) j
    …
    READ(UNIT=kgen_unit) ref_output
    CALL calc(i, j, output) ! call to kernel
    CALL kgen_verify_integer_4_dim2(...
    CALL kgen_print_check("calc", check_status)
    CALL system_clock(start_clock, rate_clock)
      …
    CALL system_clock(stop_clock, rate_clock)
    PRINT *, "calc : Time per call (usec): ", ...

    CONTAINS
      SUBROUTINE kgen_verify_integer_4_dim2(...)
        …
        IF (ALL(var==ref_var)) THEN ...
        END IF
      END SUBROUTINE
  END SUBROUTINE
END MODULE
```

**calc_mod.F90**

```
MODULE calc_mod
  ...
CONTAINS
  SUBROUTINE calc(i, j, output)
    ...
    !kgen_excluded CALL &
      print_msg('start')
    output(i,j) = i + j
    !kgen_excluded CALL &
     print_msg('finish')
  END SUBROUTINE
END MODULE
```

# Generated kernel files



**kernel_driver.f90**

```
PROGRAM kernel_driver
  ...
  OPEN(UNIT=kgen_unit, &
     FILE=kgen_filepath, …)
  ...
  CALL update(kgen_unit)

  CLOSE(kgen_unit)
  ...
END PROGRAM
```

**update_mod.F90**

```
MODULE update_mod ...
CONTAINS
  SUBROUTINE update(kgen_unit)
    INTEGER, INTENT(IN) :: kgen_unit
    ...
    CALL kgen_init_check(check_status, tolerance)
    READ(UNIT=kgen_unit) i
    READ(UNIT=kgen_unit) j
    …
    READ(UNIT=kgen_unit) ref_output
    CALL calc(i, j, output) ! call to kernel
    CALL kgen_verify_integer_4_dim2(...)
    CALL kgen_print_check("calc", check_status)
    CALL system_clock(start_clock, rate_clock)
      …
    CALL system_clock(stop_clock, rate_clock)
    PRINT *, "calc : Time per call (usec): ", ...

    CONTAINS
      SUBROUTINE kgen_verify_integer_4_dim2(...)
        …
        IF (ALL(var==ref_var)) THEN ...
        END IF
      END SUBROUTINE
  END SUBROUTINE
END MODULE
```

Read data

Verify result

Measure timing

**calc_mod.F90**

```
MODULE calc_mod
  ...
CONTAINS
  SUBROUTINE calc(i, j, output)

    !kgen_excluded CALL &
      print_msg('start')
    output(i,j) = i + j
    !kgen_excluded CALL &
    !      int_msg('finish')
  END SUBROUTINE
END MODULE
```

# Generated kernel files



**kernel_driver.f90**

```
PROGRAM kernel_driver
  ...
  OPEN(UNIT=kgen_unit, &
     FILE=kgen_filepath, …)
  ...
  CALL update(kgen_unit)

  CLOSE(kgen_unit)
  ...
END PROGRAM
```

**update_mod.F90**

```
MODULE update_mod ...
CONTAINS
  SUBROUTINE update(kgen_unit)
    INTEGER, INTENT(IN) :: kgen_unit
    ...
    CALL kgen_init_check(check_status, tolerance)
    READ(UNIT=kgen_unit) i
    READ(UNIT=kgen_unit) j
    …
    READ(UNIT=kgen_unit) ref_output
    CALL calc(i, j, output) ! call to kernel
    CALL kgen_verify_integer_4_dim2(...
    CALL kgen_print_check("calc", check_status)
    CALL system_clock(start_clock, rate_clock)
      …
    CALL system_clock(stop_clock, rate_clock)
    PRINT *, "calc : Time per call (usec): ", ...

    CONTAINS
      SUBROUTINE kgen_verify_integer_4_dim2(...)
        …
        IF (ALL(var==ref_var)) THEN ...
        END IF
      END SUBROUTINE
  END SUBROUTINE
END MODULE
```

**calc_mod.F90**

```
MODULE calc_mod
  ...
CONTAINS
  SUBROUTINE calc(i, j, output)
    ...
    !kgen_excluded CALL &
      print_msg('start')
    output(i,j) = i + j
    !kgen_excluded CALL &
     print_msg('finish')
  END SUBROUTINE
END MODULE
```

kernel

# Generated kernel files



**kernel_driver.f90**

```
PROGRAM kernel_driver
  ...
  OPEN(UNIT=kgen_unit, &
     FILE=kgen_filepath, …)
  ...
  CALL update(kgen_unit)

  CLOSE(kgen_unit)
  ...
END PROGRAM
```

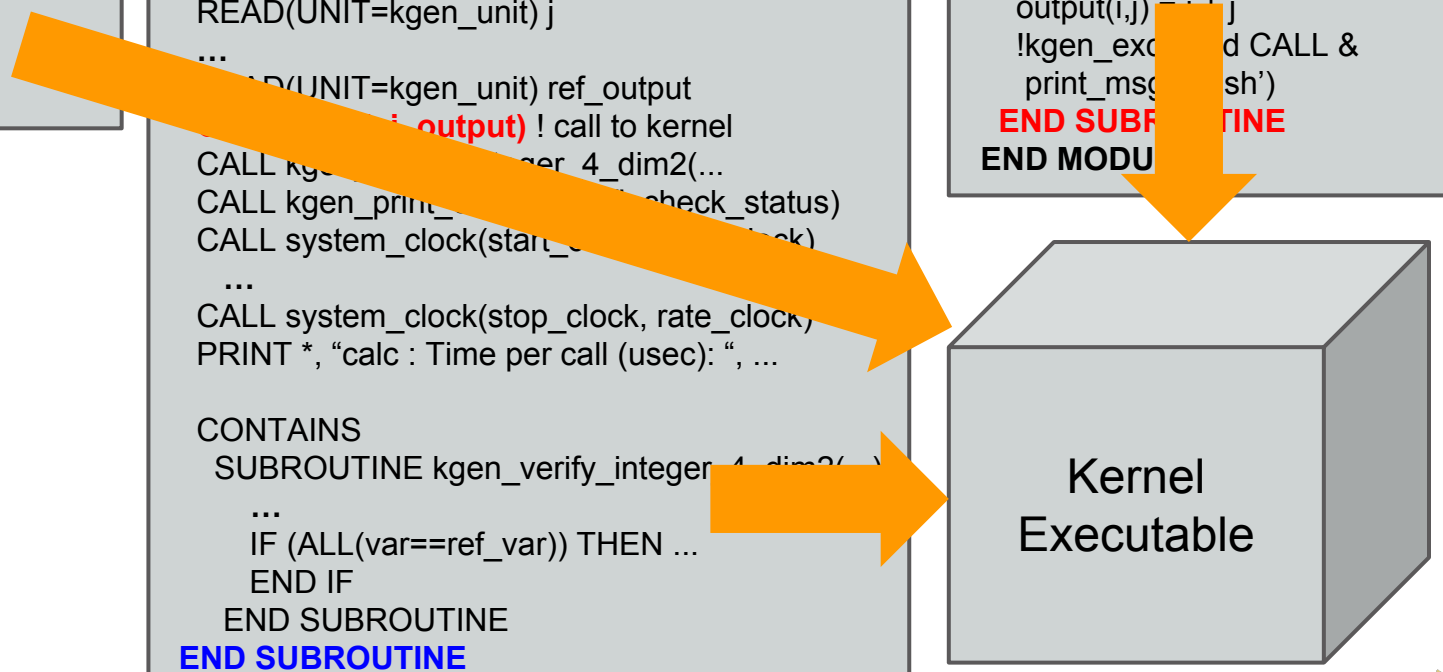
**update_mod.F90**

```
MODULE update_mod ...
CONTAINS
  SUBROUTINE update(kgen_unit)
    INTEGER, INTENT(IN) :: kgen_unit
    ...
    CALL kgen_init_check(check_status, tolerance)
    READ(UNIT=kgen_unit) i
    READ(UNIT=kgen_unit) j
    ...
    AD(UNIT=kgen_unit) ref_output
    output) ! call to kernel
    CALL kg                ger_4_dim2(...
    CALL kgen_print_       heck_status)
    CALL system_clock(start_c      ck)
      ...
    CALL system_clock(stop_clock, rate_clock)
    PRINT *, “calc : Time per call (usec): “, ...

    CONTAINS
      SUBROUTINE kgen_verify_integer_4_dim2(...)
        ...
        IF (ALL(var==ref_var)) THEN ...
        END IF
      END SUBROUTINE
  END SUBROUTINE
END MODULE
```

**calc_mod.F90**

```
MODULE calc_mod
  ...
CONTAINS
  SUBROUTINE calc(i, j, output)
    ...
    !kgen_excluded CALL &
      print_msg(‘start’)
    output(i,j) = i + j
    !kgen_exc      d CALL &
     print_msg     sh’)
  END SUBR      TINE
END MODU
```



# Generated instrumentation file



**update_mod.F90**

```
MODULE update_mod
CONTAINS
  SUBROUTINE update(rank, N)
    ...
    DO i=1,COL
      DO j=1,ROW
        !$KGEN callsite calc
        ...
        OPEN(UNIT=kgen_unit, FILE=kgen_filepath, …)
        ...
        WRITE(UNIT=kgen_unit) i
        WRITE(UNIT=kgen_unit) j
        ...
        CALL calc(i, j, output)
        ...
        WRITE(UNIT=kgen_unit) output
        ...
        CLOSE(UNIT=kgen_unit)
        ...
      END DO
    END DO
  END SUBROUTINE
END MODULE
```

# Generated instrumentation file



**update_mod.F90**

```
MODULE update_mod
CONTAINS
  SUBROUTINE update(rank, N)
    ...
    DO i=1,COL
      DO j=1,ROW
        !$KGEN callsite calc
        OPEN(UNIT=kgen_unit, FILE=kgen_filepath, …)
        WRITE(UNIT=kgen_unit) i
        WRITE(UNIT=kgen_unit) j
        ...
        CALL calc(i, j, output)
        ...
        WRITE(UNIT=kgen_unit) output
        …
        CLOSE(UNIT=kgen_unit)
        ...
      END DO
    END DO
  END SUBROUTINE
END MODULE
```

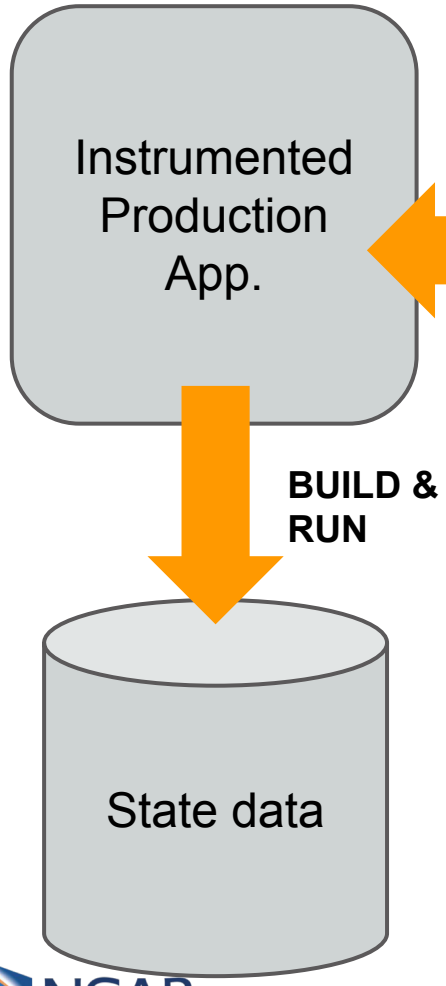
Open data file to write

write input variable

Write output variable

# Generated instrumentation file






**update_mod.F90**

```
MODULE update_mod
CONTAINS
  SUBROUTINE update(rank, N)
    ...
    DO i=1,COL
      DO j=1,ROW
        !$KGEN callsite calc
        ...
        OPEN(UNIT=kgen_unit, FILE=kgen_filepath, …)
        ...
        WRITE(UNIT=kgen_unit) i
        WRITE(UNIT=kgen_unit) j
        ...
        CALL calc(i, j, output)
        ...
        WRITE(UNIT=kgen_unit) output
        ...
        CLOSE(UNIT=kgen_unit)
        ...
      END DO
    END DO
  END SUBROUTINE
END MODULE
```